CPP-Summit 2022

全球C++及系统软件技术大会

C++ and System Software Summit

嘉宾演讲稿合集

主办方：Boolan 高端IT咨询与教育平台

大会简介

系统级软件是数字世界的基础设施，C++自1985年由Bjarne Stroustrup博士在贝尔实验室发明以来，一直被誉为系统级编程“皇冠上的明珠”。 秉承“全球专家，卓越智慧“的理念，全球C++及系统软件技术大会（Cpp-Summit）于2005年由李建忠老师发起创办，并得到Bjarne Stroustrup和众多C++大师专家的支持。

每一年，我们特邀国内外C++和系统软件领域的大师、专家、学者，汇聚一堂，在北京、上海、深圳等城市轮流举办，深度探讨相关领域的最佳工程实践和前沿方法。在各方支持下，全球C++及系统软件技术大会已经成长为系统软件领域公认的风向标会议。

大会官网：www.cpp-summit.org

联席主席



Bjarne Stroustrup

C++之父

美国国家工程院、ACM、IEEE 院士



李建忠

Boolan 首席软件专家

全球C++及系统软件技术大会主席



Michael Wong

C++ 嵌入式开发委员会SG14主席

Codeplay研发副总裁



潘爱民

指令集公司CEO

著名操作系统专家

Boolan 高端IT咨询与教育平台

8大
技术咨询领域

Boolan依托雄厚的专家团队和丰富的行业经验，通过八大技术咨询板块，提供涵盖产品管理、架构设计、人工智能等全栈技术咨询解决方案，帮助企业在数字化、智能化、万物互联的变革中打造关键创新能力，加速技术和产品迭代升级，助力企业快速成长。

4大
高端技术会议

Boolan高端技术会议汇聚国内外技术与产品大咖，分享一线实战经验，解析前沿技术趋势，探讨行业创新机遇！Boolan四大技术会议品牌涵盖：全球C++及系统软件技术大会（CPP-Summit）、全球软件研发技术峰会（SDCon）、全球机器学习技术大会（ML-Summit）、全球产品经理大会（PM-Summit）。

300+
培训课程主题

Boolan针对企业特定培训需求，通过多年的技术课程体系建设，涵盖机器学习、软件架构、产品经理、C/ C++、Java平台等14个领域，300+课程主题；满足不同领域，不同阶段公司差异化的人才培养需求，实现企业团队人力资本的提升。

600+
位国内外专家

Boolan拥有遍布全球的600多位国内外专家资源.既有C++之父Bjarne Stroustrup,硅谷产品大师Marty Cagan,机器学习之父Michael.I.Jordan，全球软件架构大师Martin Abbott等在IT互联网领域有极强号召力的国际大师；也有来自国内外著名互联网科技企业的技术骨干和行业精英。

3000+
高端企业客户

Boolan助力企业全方位成长，迄今为止，我们已为华为、腾讯、百度、eBay、微软、SAP、HP、思科等国内外3000+企业提供高端IT技术咨询与培训服务，客户遍及互联网、通信、金融等各个行业领域，并且持续扩展中。



C/C++ 与系统软件研发咨询

C++系统软件性能工程与优化

以现代C++的程序优化与性能调优为中心，通过系统建设性能工程、梳理软硬件性能热点，提供全流程的性能优化策略、方法和工具实践，全方位提升大型系统软件的性能和资源效率。

领域驱动设计与系统建模

以领域场景为横切面，驱动从战略设计到战术设计的全生命周期的完整开发过程，通过领域分析、设计和实现建模，大幅提升系统设计和建模质量。

开发者测试与质量工程

在软件过程“左移”的驱动下，通过引入先进的开发者测试流程、框架和最佳实践，为软件质量工程注入新时代的变革支持，打造质量为先的卓越产品。

可信计算与软件程序安全

通过漏洞挖掘、攻击面分析、模糊测试、安全测试、逆向技术、恶意代码检测等方式，为企业提供一揽子系统安全可信设计和实施方案。

C++大型系统软件架构重构

以业务需求为导向，关键技术为支撑，从领域驱动模型到组件化架构，从架构风格到质量评审，从设计原则到性能优化，我们致力于全栈技术架构方案和咨询。

大型软件工程与研发效能提升

帮助企业建立和改善IT研发管理体系，通过提供流程、工具、方法论等赋能型支撑，大幅度提升团队研发效能和软件过程成熟度。

持续集成与DevOps体系建设

持续集成和DevOps是加速软件服务化转型的必然选择，通过全面重塑软件开发、测试、部署流程，帮助研发团队打造可靠的CI/CD流程和DevOps体系。

异构计算架构加速与优化

异构计算是后摩尔定律时代计算架构迎来范式转换的主旋律，是系统加速和整体优化的关键布局。助力企业引领计算潮流，驾驭技术变革时代。



2022技术会议 预告



随着信息革命的飞速发展，软件已深入人类生活的方方面面，深刻而彻底地改造了人类世界。秉承“全球专家、卓越智慧”的宗旨，我们特邀近40位全球软件领域的技术领袖以及一线实战专家，融合主题演讲、互动研讨、案例分享、高端培训等多种形式，共同探讨软件领域的前沿发展、最佳实践和创新应用，共同打造大师智慧+实践干货的技术盛宴。

官网：www.sdcon.com.cn

大会时间：12月2-3日

大会地点：上海



自1936年阿兰·图灵提出「图灵机」以及机器具备「思维」的可能性以来，以机器学习为代表的人工智能经过飞速发展，深刻地改变着我们的世界。Boolan 秉承“全球专家，卓越智慧”的宗旨，特邀近40位机器学习领域的技术领袖和行业应用专家，与1000+来自电商、金融、汽车、智能制造、通信、工业互联网、医疗、教育等众多行业的精英参会听众，共同探讨人工智能领域的前沿发展和行业最佳实践。

官网：www.ml-summit.org

大会时间：12月2-3日

大会地点：上海



全球产品经理大会 （PM-Summit）是汇聚全球互联网与科技产品领域专业人士的高端交流和分享平台。Boolan秉承“全球专家、卓越智慧”的宗旨，我们将邀请来自全球互联网与科技产品领域的大师、领袖与一线专家，与众多行业精英参会者共同开展一场面向全球产品人的高端盛宴！

官网：www.pm-summit.org

大会时间：12月18-19日

大会地点：上海

GDK套件

软件高手的私人实验室



JTAG 是一种芯片层的调试和测试协议，

因为不依赖上层软件而具有纯软件工具无法实现的高级功能，
基于JTAG协议的调试工具一直是用于探索芯片内部逻辑最重要的工具。
使用英特尔提供的ITP调试工具是在x86平台上进行JTAG调试的一种主要方法，
但是ITP工具不仅价格昂贵，有条件供应，
而且要求被调试系统的主板上有ITP/XDP接口，很少适用。

GDK7 是格灵科技针对X86平台开发的高级调试套件

使用英特尔的直接连接接口(DCI)通过USB3.0端口
就可以进行JTAG调试和芯片级追踪(Processer Trace)。
“到达没有到达过的领地，看见原本无法看见的芯片内部逻辑”



CoreSight （核景）是ARM平台上一系列调试和追踪技术的总称，

具有丰富而且强大的调试和优化功能，在ARM生态中扮演着重要角色。

NTP （挥码枪）是基于ARM CoreSight技术研发的新一代硬件调试器

可以通过CoreSight协议操控ARM SoC芯片，
让风驰电掣的CPU核心戛然而止，进入停顿状态，接受审查。
挥码枪可以与格灵科技的NDB软件调试器一起工作，
获取目标系统的高层语义，以符号化的方式来调试安卓和其它LINUX系统，
既具有强大的底层控制力，又具有丰富的上层语义。



GDK8 是格灵科技针对ARMv8平台研发的微型计算机套件

超低功耗，无风扇设计，体积小巧，功能强大。
既可以当作物联网的微型终端，又可以作为开发和调试套件。
预装了GNU的开发工具链，能在ARM系统本机开发ARM应用软件和驱动程序，
彻底改变了传统的交叉编译方式，大大提高了开发和调试效率。

GDK8 可以与格灵科技的Nano Debugger（NDB）一起工作，

使用WinDBG的各种调试命令来调试Linux程序，
将Linux平台的调试技术推上一个新的台阶，
是学习和研究ARMv8架构、LINUX操作系统和嵌入式软件技术的有力助手。



ArmV7-M 是ARMv7架构的M规格CPU

ArmV7-M的位宽为32位，包含支持保护内存系统架构的MPU，
广泛应用于深度嵌入式、实时控制和物联网等领域。

GDK3 是基于ARM Cortex-M3 SoC设计的开发板，包含ArmV7-M

GDK3为学习嵌入式编程提供了一个高效的试验环境，
支持多种RTOS系统，在上面可以使用C语言来开发各种嵌入式程序。
用户可以使用GDK3来模拟要研究的系统环境，
并借助与GDK3配套的强大调试功能，对系统的软硬件进行深入研究。



客户经理：Wayne
联系电话：13764289159
咨询邮箱：wayne.zhang@boolan.com
咨询热线：400-821-5876

目 录

现代C++的发展与演化	Bjarne Stroustrup	C++之父，美国国家工程院、ACM、IEEE 院士	10
使用概念重塑“ 零开销抽象 ”设计模式	李建忠	Boolan 首席软件专家	66
整洁DCI架构实践	李永顺	中兴通讯5G软件架构总工，资深架构师	105
UWASM -- An Enhanced WASM Framework	李隆	鉴释科技首席科学家	146
C++开发者测试最佳实践	赵永刚	Boolan 资深咨询师	189
C++ 编译期编程的过去、现在和未来	吴咏炜	Boolan首席咨询师	233
现代 C++ 基于性能的重构优化	尉刚强	资深软件技术顾问	295
RCU无锁编程探究与实践	Paul McKenney	前IBM杰出工程师, 世界级并行编程专家	336
风险指针（Hazard Pointer）无锁编程精要	Maged Michael	ACM 杰出科学家，Facebook技术专家	511
使用内存分配器优化性能	John Lakos	大规模软件设计世界级权威	530
嵌入式软件产品化架构	温昱	Boolan 首席咨询师	1082
现代化工具链在大规模C+项目中的实践	许传奇	之阿里巴巴开发工程师，C++标准委员会成员	1117

Curve分布式存储架构与设计 向东 网易杭州研究院架构师 1155

超大规模C++项目构建优化 谢茂培 金山办公研发中台事业部office产研架构师 1186

异构计算与SYCL 张银奎 著名系统内核专家 1210

Workflow的性能优化实践 李颖欣 搜狗高级架构师 1259

大规模软件性能设计与实践 Fedor Pikus Mentor 首席工程科学家 1299

C++内存与性能优化最佳实践 Michael Spertus Amazon 首席工程师 1401

Max功能实践：API设计、悬挂引用以及约束 Amir Kirsh Incredibuild开发布道师 1446

底层操作系统上的C++ 熊谱翔 RT-Thread 创始人 1457

Google 开源 C++ 工具、库、文档 陈硕 Google高级工程师，muduo开源库作者 1483

利用Ray快速构建系统级C++分布式应用 宋顾杨 蚂蚁集团技术专家 1534

性能优化的路有多远？ 左春伟 美团基础研发平台-数据平台架构师 1589

SYCL编码实践与编译设计浅析 谢丙堃 《现代C++语言核心特性解析》作者 1636

使用模板元编程优化嵌入式开发 杨文波 资深软件专家 1678

拥抱C++ 20 元编程 Kris Jusiak Quantlab Financial资深架构师 1725

eBPF工具的开发和资源优化在多内核版本的实践 毛文安 阿里云操作系统高级技术专家 1868

Windows、macOS and the Web:跨平台实践经验 Sebastian Theophil Think-cell资深软件工程师 1911

C++程序编译优化：AI for Compiler方法初探 袁鹏 腾讯专家工程师 1997

当C++遇见Redis 刘四维 阅文集团智能业务中心后端技术负责人 2020

proxy库：驾驭多态、无惧规模 王铭鑫 微软软件工程师，C++标准委员会成员 2073

高质量代码生命周期的思考与实践 Phil Nash 开发者测试专家， Catch创始人 2118

C++ 类型分类、陷阱与最佳实践 Nina Ranns ISO C++ Foundation 总监 2213

CPP-Summit 2022



Bjarne Stroustrup

C++之父，美国国家工程院、ACM、IEEE 院士

C++之父，美国国家工程院、 ACM 、 IEEE 院士，被誉为“近 20 年来计算机工业最具影响力的 20 人”。Bjarne Stroustrup 于 1979 年获得剑桥大学计算机博士学位，毕业后加入贝尔实验室，在那里发明了 C++。他的研究领域包括：分布式系统、编程语言、软件开发工具。曾于2018年荣获美国国家工程院颁发 查尔斯·斯塔克·德拉普尔奖，被誉为工程学界的诺贝尔奖。现任摩根士丹利技术部董事总经理，哥伦比亚大学客座教授。代表作品：《 C++ 程序设计语言》《 C++ 语言的设计和演化》。

主办方：Boolan 高端IT咨询与教育平台

Thriving in a crowded and changing world: C++ 2006-2020

在拥挤和变化的世界中茁壮成长:
C++ 2006-2020

Bjarne Stroustrup

Morgan Stanley, Columbia University

摩根士丹利，美国哥伦比亚大学



Overview

概述

- What C++ is
- C++ 是什么
  - Ideals, strategy, evolution, and stability
  - 理想、策略、演化及稳定性
  - Early C++ (1979-1998)
  - 早期 C++ (1979-1998)
- How it got there
- 一路走来
  - The world in 2006
  - 2006 年的世界
  - Standardization
  - 标准化
  - C++11 and C++20
  - C++11 和 C++20
  - Technical highlights with code
  - 技术亮点加代码片段
  - The world in 2020
  - 2020 年的世界
- The near future
- 不久的将来
  - Hopes, direction, and cautions
  - 希望、方向和警惕



For “details” see
“Thriving in a crowded and changing world C++ 2006-2020”
ACM SIGPLAN HOPL-IV paper (168 pages).
Chinese version available
详见
ACM SIGPLAN HOPL4 论文 （原文168 页）《在拥挤和变化的世界中茁壮成长：C++ 2006-2020》
有中文译本

Thanks!

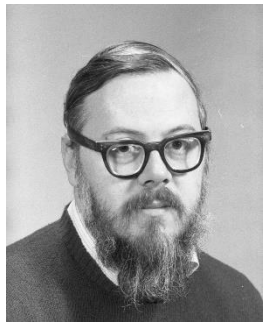
感谢!



- HOPL-IV paper reviewers
- HOPL4 论文审稿人
  - Official: Yannis Smaragdakis, Shigeru Chiba, and Brent Hailpern
- 官方:
  - Friends: Al Aho, A. Bratterud, J.Daniel García, Howard Hinnant, Roger Orr, Aaron Satlow, David Vandevoorde, J.C. Van Winkel, Michael Wong, …
- 朋友们:
- C++ and software experts
- C++ 及软件专家们
  - C++ language, C++ implementation, library design, software design and implementation, education, development tools, programming environments, …
  - C++ 语言、C++实现、库设计、软件设计和实现、教育、开发工具、编程环境等
  - Contributors to the C++ standardization
  - C++ 标准的贡献者们
- C++ users
- C++ 用户们
  - Offering ideas, problems, and suggested solutions
  - 贡献想法、问题以及推荐解决方案
- The hundreds of people that commented on the ideas and examples over the years
- 多年来成百上千为理念和例子提供意见建议的人们

Innovation = invention + development

创新 = 发明 + 开发



Dennis Ritchie

The Bell Labs definition of technological progress

贝尔实验室对技术进步的定义

C
C++
awk
Unix
…



Transistors
CCDs
Fiber
…

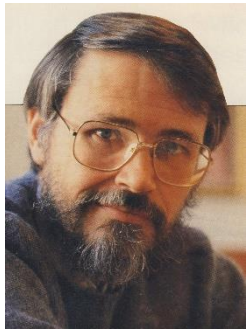
晶体管
CCD
光纤



Doug McIlroy



Al Aho



Brian Kernighan

Rich technical culture

丰厚的技术文化

Huge range of interesting technical problems

广泛而有趣的技术问题

Academia ≠ Education ≠ Industry

学术界 ≠ 教育界 ≠ 工业界



- They don’t look at programming languages in the same way
- 他们不以同样的方式看待编程语言
- Over-simplification:
- 过于简单化:
  - ***Academia***: cleverness, novelty, reviewed publication
  - 学术界：聪明、新奇、评审后发表
  - ***Education***: simplicity, ease of teaching
  - 教育界：简单性、易于教学
  - ***Industry***: familiarity, performance, completeness, stability, tools, systems, support, interoperability, industry standards, …
  - 工业界：熟悉、性能、完整性、稳定性、工具、系统、支持、互操作性、工业标准.....

- C++ was designed for industry
- C++是为工业界设计的
  - is pervasive in industry
  - 在工业界无处不在
  - is widely used in academia and education
  - 在学术界和教育界被广泛使用



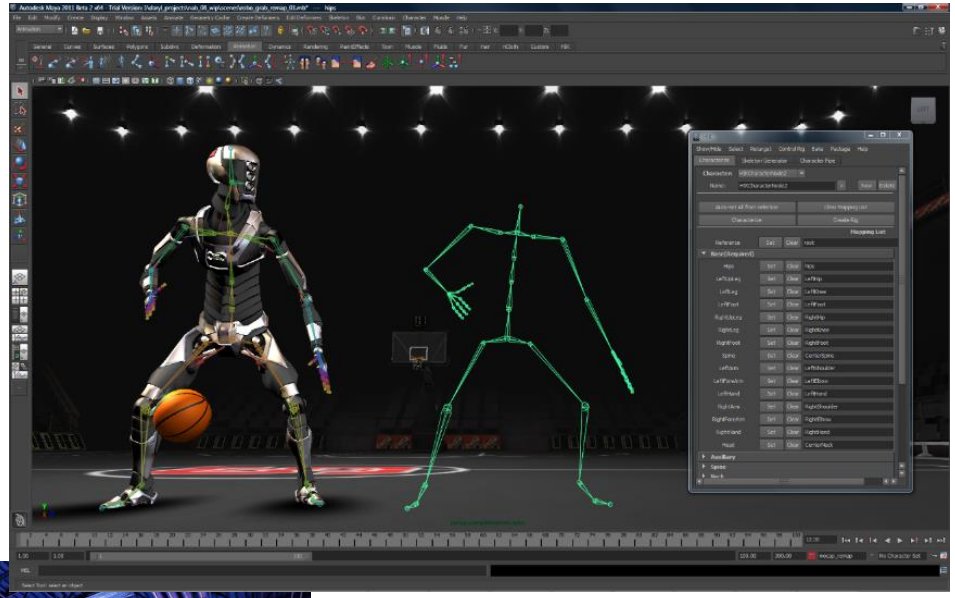


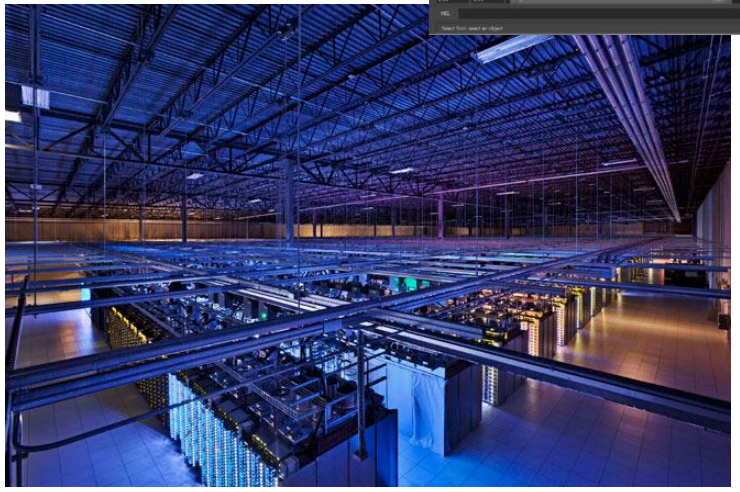
The value of a programming language is in the quality of its applications

编程语言的价值在于它应用的质量

Morgan Stanley











C++ design aims and strategy

C++ 设计的目标和策略

- To improve the state of real-world software
- 改善现实世界软件的状态
  - Direct expression of ideas in code
  - 在代码中直接表达想法
  - Reliability
  - 可靠性
  - Performance
  - 性能
  - Maintainability
  - 可维护性
  - Development time
  - 开发时间
- Start small and grow
- 从小事做起，不断成长
  - Rely on feedback from real-world use
  - 依靠真实世界的使用反馈
  - Don’t grow too fast
  - 不要发展得过快
- Know roughly where you’d like to go
- 大致知道你想去哪里
  - Or you get lost
  - 否则你就会迷失方向
- Rely on articulated principles
- 依靠明确的原则
  - Or you get steamrollered by requests
  - 否则你就会被请求压垮

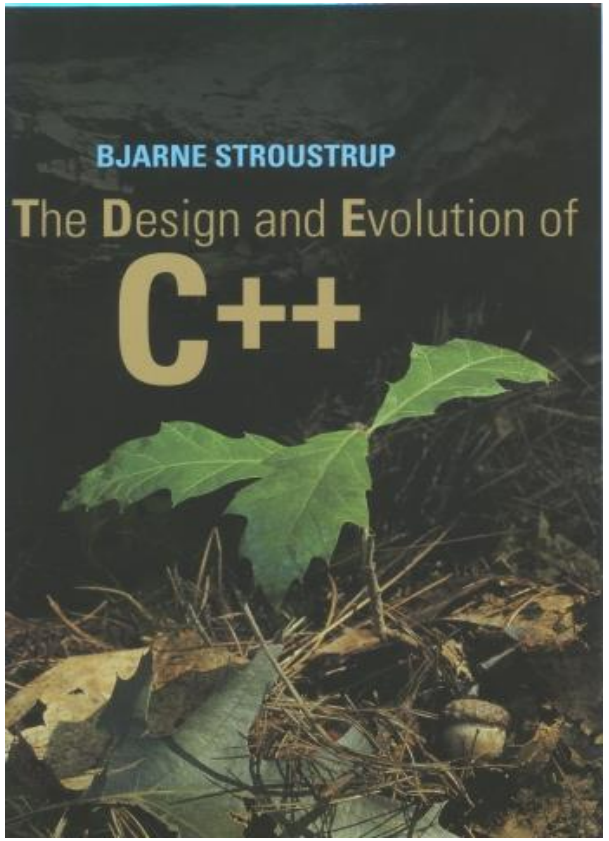


Bjarne Stroustrup 1984

C++ design rules of thumb

C++ 设计的经验法则

- A selection from “The Design and Evolution” 1994
- 1994 年《设计与演化》选录
  - Don’t get involved in a sterile quest for perfection
  - 不要陷入对完美的徒劳追求
  - Always provide a transition path
  - 始终提供过渡路径
  - Say what you mean
  - 说出你的意图
    - i.e., enable direct expression of higher-level ideas
    - 即，能够直接表达高层次的思路
  - Provide as good support for user-defined types as for built-in types
  - 为用户定义类型提供和内置类型同样好的支持
  - All features must be affordable
  - 所有的功能都必须是可以负担得起的
  - No implicit violations of the static type system
  - 不要隐式地在静态类型系统方面违规
  - Preprocessor usage should be eliminated
  - 应当取消预处理器的使用
  - Leave no room for a lower-level language below C++
  - 不要给 C++ 以下的低级语言留有余地
    - except assembler
    - 汇编语言除外



Abstraction
抽象

Type safety
Expressive type system
类型安全
富于表现力的类型系统

Direct use of hardware
Zero-overhead abstraction
直接使用硬件
零开销抽象

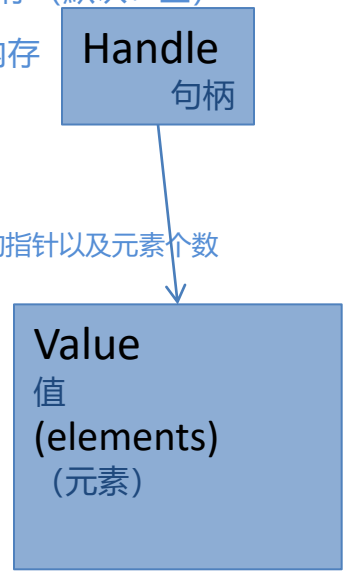
Early C++ (1980): abstraction and scope-based resource management

早期的 C++（1980）：抽象和基于作用域的资源管理

```
class Vector {        // vector of Elements of type double 元素类型为双精度浮点数的 vector
public:
        Vector(int n =0);     // constructor: acquire memory for n elements (default: empty) 构造函数：为 n 个元素获取内存（默认：空）
        ~Vector();            // destructor: destroy elements; release memory  …析构函数：销毁元素；释放内存

private:
        double* elem;         // representation, e.g., pointer to elements plus #elements 表示，例如指向元素的指针以及元素个数
        int sz;               // #elements 元素个数
};

void fct(int n)
{
        Vector v (n);         …     v[0]=1.618;          // vector of n doubles; subscripting n 个浮点数的 vector; 下标运算
        // …
}  // all memory released here 所有内存在此释放
```



Early C++: what I really wanted – parameterized types

早期的 C++：我真正想要的东西—参数化类型

I didn't get this until 1990.

Sometimes, you must be patient

我直到 1990 年才得到这些

有时，你必须耐心

```
template<class T>
class Vector {              // vector of Elements of type T 元素类型为T 的 vector

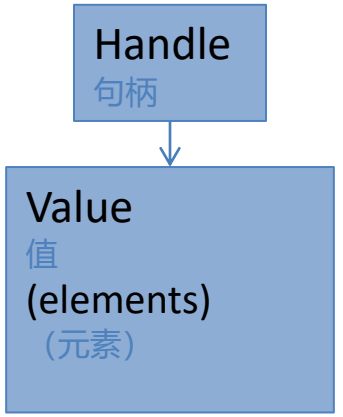
public:
        Vector(int n =0);           // constructor: acquire memory for n elements (default: empty) 构造函数：为 n 个元素获取内存（默认：空）

        ~Vector();    // destructor: destroy elements; release memory 析构函数：销毁元素；释放内存

        // …
private:
        T* elem;      // representation, e.g., pointer to elements plus #elements 表示，例如指向元素的指针以及元素个数
        int sz;       // #elements 元素个数
};

void fct(int n)
{
        Vector<double> v(n);        …    v[0]=1.618;                    // vector of n doubles; subscripting  n 个浮点数的 vector; 下标运算

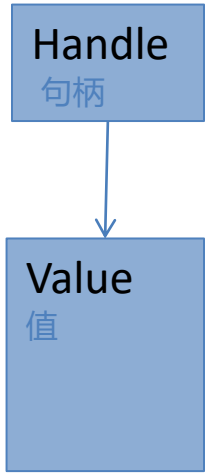
        Vector<string> vs;          …    vs.push_back("Strachey");      // empty vector of strings; grow it 空的字符串 vector；增长它
        Vector<File_handle> f(3);   …    f[0] = File_handle("Myfile", "r");   // open a file allocate buffers, etc. 打开为文件分配的缓冲区等
        // …
}  // memory, strings, file handles, etc. released here 内存，字符串，文件句柄等在此释放
```



Resource management (..., C++98, C++11, ...)

资源管理（..., C++98, C++11, ...）

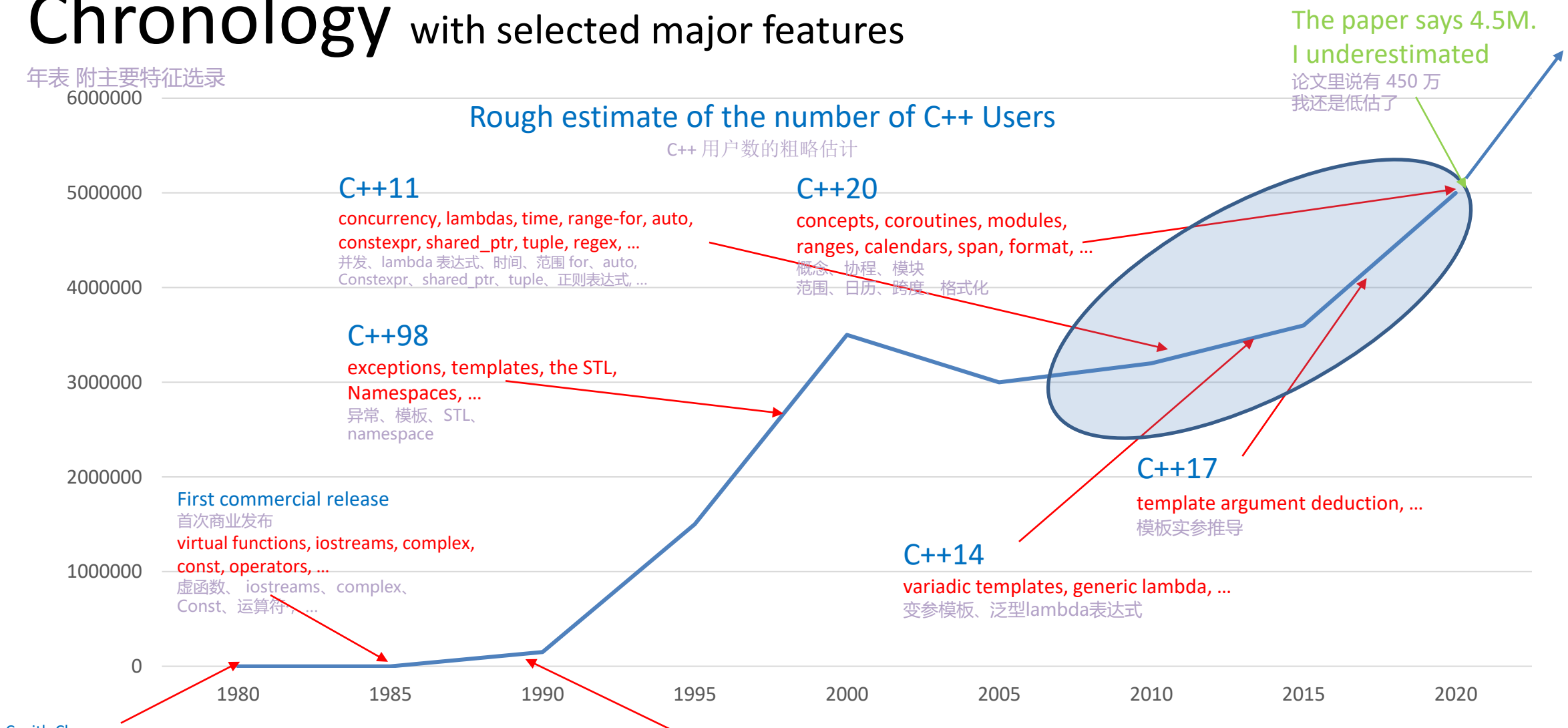
- Make resource release implicit and guaranteed (RAII)
- 使资源释放既是隐式的又有保证(RAII)
- All standard-library containers manage their elements
- 所有标准库容器都管理其元素
  - **vector**
  - **list, forward_list** (singly-linked list), …
  - **map, unordered_map** (hash table),…
  - **set, multi_set, …**
  - **string**
  - …
- Resource management pointers handle lifetime of what they point to
- 资源管理指针处理它们所指向的东西的生存期
  - **unique_ptr**, **shared_ptr**, …
- Some standard-library classes represent non-memory resources
- 一些标准库类表示的是非内存资源
  - **thread**, **scoped_lock**, …
  - **istream**, **fstream**, …
  - …
- A container can hold non-memory resources
- 容器可以容纳非内存资源
  - This all works recursively
  - 这些都可以递归地发挥作用



GC is neither sufficient nor ideal
垃圾收集既不充分也不理想

Chronology with selected major features

年表 附主要特征选录



C++ in 2006 (nadir)

2006 年的 C++（低谷）

- C++ was no longer new and exciting
- C++不再新奇和令人激动
  - 25 years old
  - 25 岁
  - About 3 million users (and declining)
  - 大约 3 百万用户（还在减少）
  - No major commercial backer
  - 没有主要的商业支持者
  - Controlled by a large ISO standards committee
  - 被大型 ISO 标准化委员会控制
- Heavily marketed proprietary languages: Java, C#, …
- 大肆推广的专有语言：Java、C#，…
  - With massive ecosystems
  - 拥有庞大的生态系统
- Heavy academic push for alternatives
- 学术界对替代语言的强力推动
  - Massive switch to Java for early programming education
  - 早期编程教育大批转到 Java
- Single thread hardware performance stalled
- 单线程硬件性能止步不前
- Vastly increased emphasis on energy efficiency
- 对能效的重视程度大幅提高
- Fight for commercial platform dominance
- 争夺商业平台的主导地位
  - Java (Sun, IBM, Google), Microsoft (C#), Apple (Objective C)



The standards process

标准化流程

- Requested by HP, IBM, and Sun in 1989
- 1989 年由惠普、IBM 和 Sun 公司提出要求
  - Specifically, an ANSI/ISO standard
  - 具体来说，是一套 ANSI/ISO 标准
    - Consensus-based decisions
    - 基于共识的决定
    - Many national standards bodies
    - 许多国家标准机构
- First standard in 1998
- 第一个标准在 1998 年出台
  - Then C++11, C++14, C++17, and C++20
  - 然后是 C++11、C++14、C++17 和 C++20
    - All unanimously approved after years of work
    - 经过多年的工作，所有的标准都被一致通过
- Aims
- 目标
  - Stability
  - 稳定性
  - Support for wide use
  - 支持广泛使用
  - Defense against vendor lock-in
  - 防范供应商的锁定
  - Defense against dialects
  - 防止方言的出现
  - Improvements of language and standard library
  - 改进语言和标准库





WG21 “the committee”

WG21 “委员会”

1990



2011



2014



2017



C++20 is here

C++20 在此

Prague
February 2020

布拉格
2020 年 2 月



The standards process

标准化流程



- Large membership
- 成员众多
  - ~40 in 1990
  - 1990 年~40 人
  - ~100 in 2006 (~60 attendees)
  - 2006 年~100 人(~60 人参加)
  - ~350 in 2020 (~250 attendees)
  - 2020 年~350 人(~250 人参加)
  - Much larger than all other ISO language committees put together
  - 比所有其他 ISO 语言委员会的总和还要大得多
- Many nations
- 许多国家
  - Australia, Bulgaria, Canada, Czech Republic, Finland, France, Germany, Israel, Japan, Netherlands, Poland, Russia, Spain, Sweden, Switzerland, UK, USA, …
  - 澳大利亚、保加利亚、加拿大、捷克共和国、芬兰、法国、德国、以色列、日本、荷兰、波兰、俄罗斯、西班牙、瑞典、瑞士、英国、美国 …
- Mostly industry
- 主要是工业界
  - Adobe, Argonne NL, AT&T, Bloomberg, CERN, Ericsson, Google, HP, IBM, Intel, Microsoft, Morgan Stanley, National Instruments, Nokia, NVIDIA, Red Hat, Sandia NL, Qt, …
  - A few universities
  - 一些大学
  - Many dozens of smaller firms
  - 数十家小型公司
  - Many individuals
  - 许多个人

The standards process

标准化流程

- --> C++98 --> C++11 --> C++14 --> C++17 --> C++20 --> C++23 -->

9 years

13 years

Unacceptable
无法接受

- "ISO normal" was 10 years between standards
- "ISO 的正常情况" 是标准之间间隔 10 年
  - As in from C++11 to C++20
  - 如从 C++11 到 C++20
- Post-C++11: "Train model"
- 后 C++11： "列车模型"
  - The train leaves the station on schedule
  - 列车如期离站
  - Major – minor – major – …
  - 主要—次要—主要—…
- Dangers of the ISO standards process
- ISO 标准化流程的危险性
  - Lack of direction
  - 缺少方向
  - "Design by committee" – really "design by a federations of committees"
  - "由委员会设计" — 实际上是 "多委员会的联合会设计"
  - Fossilization
  - 僵化
  - Delays
  - 延迟
  - Acceptance of untried ideas
  - 接受未经试验的想法



Herb Sutter

Direction is essential

方向至关重要

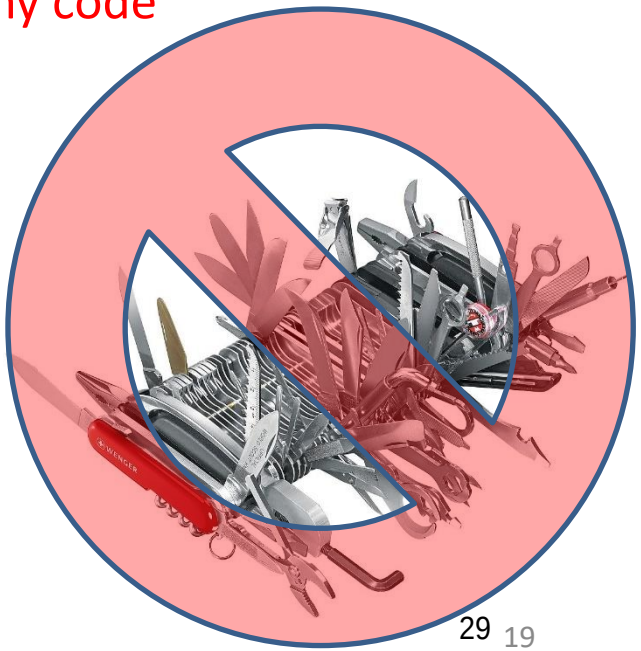


- Aim: A dense web of interrelated mutually-supporting features
- 目标：一个由相互关联、相互支持的功能组成的密集网络
  - Not a set of independent cool features supporting disjoint design paradigms
    而不是一套彼此独立的、支持不相干的设计范式的很酷的功能集合
- Setting and keeping direction is very hard
- 设定和保持方向是非常困难的
  - Designing for good feature interaction requires a global view
  - 为良好的功能互动而设计需要一个全局观
  - It often takes years to get solid feedback
  - 往往需要几年的时间才能得到可靠的反馈
  - Fashions have strong pulls
  - 时尚有很强的牵引力
  - Many developers want help with their immediate problems
  - 许多开发者希望能够帮助他们解决眼前的问题
  - Even the smallest extension has significant cost
    即使是最小的扩展也有很大的成本

Typical request

典型请求

- Simplify C++
- 简化 C++
- Add these two new features
- 加入这两个新特性
- Don’t break my code
- 不要搞砸我的代码

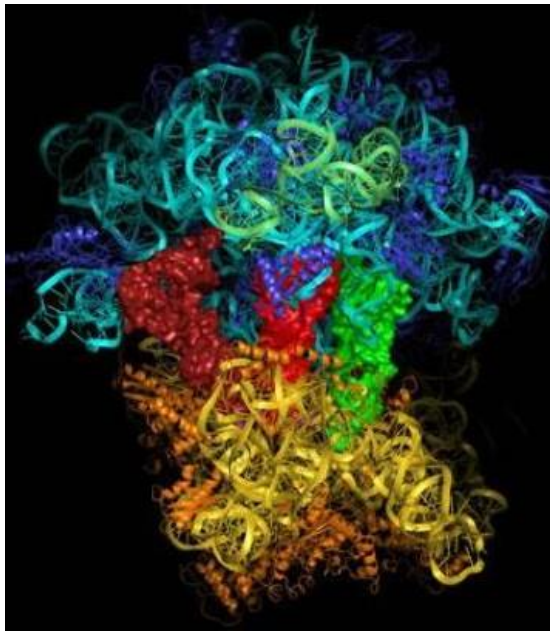


Why change/evolve?

为什么改变/演化?



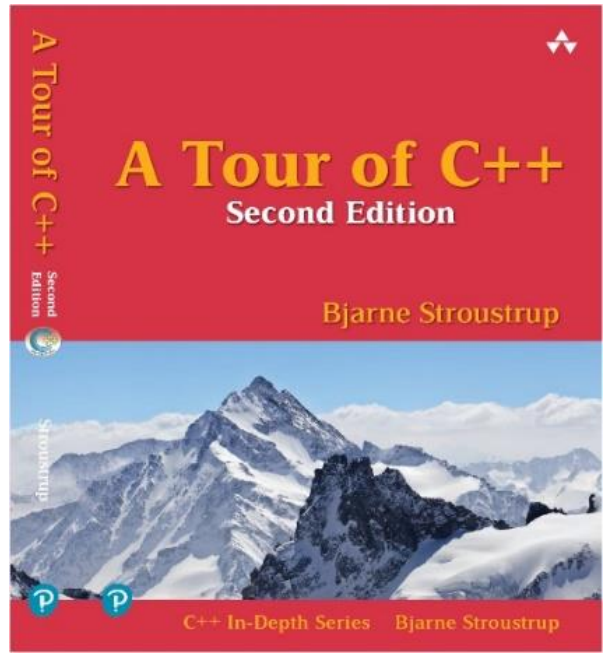
- The world changes
- 世界在变
  - no mobile apps until the early 2000s
  - 直到 21 世纪初才有移动应用程序
  - power consumption wasn't a major issue until the 2000s
  - 在 2000 年之前，功耗并不是一个主要问题
- We change
- 我们也变
  - few industrial programmers appreciated object-oriented programming in 1979 or generic programming in 1985
  - 在1979 年，很少有工业界的程序员懂得面向对象的编程，在 1985 年，很少人懂泛型编程。
- No language is perfect for everything and for everybody
- 没有一种语言完美适用于所有事物和所有人
  - that includes C++
  - 这包括C++
  - that includes every new language
  - 包括每一种新的语言
- Aim for major changes
- 以重大改变为目标
  - By "major", I mean "changes the way we think"
  - 我所说的 "重大 "是指 "改变我们的思维方式"
  - "major" doesn't imply "hard to use"
  - "重大 "并不意味着 "难用"
- Minimize minor changes
- 尽量减少小的变化
  - That's hard, most people argue for minor changes
  - 这很难，大多数人主张小改动。
  - "minor" doesn't imply "easy to implement"
  - "小 "并不意味着 "容易实现



Stability & evolution

稳定性与演化

- How to get both?
- 如何兼得?
  - We can't simplify the language
  - 我们不能简化语言
    - Don't break working code
    - 不要破坏工作的代码
  - We can simplify use
  - 我们可以简化使用
    - Add improved facilities
    - 增加改进的机制
    - Offer concrete guidance on how to avoid outdated features
    - 就如何避免过时的特性提供具体的指导
    - Offer concrete guidance on how to use improvements
    - 为如何使用改进的功能提供具体的指导
- C++ Core guidelines
- C++ 核心指南
  - No resource leaks
  - 没有资源泄漏
  - No memory corruption
  - 没有内存损坏
  - No garbage collector
  - 不需要垃圾收集器
  - No limitation of expressiveness
  - 没表达能力的限制
  - No performance degradation
  - 没有性能下降
- Guidelines need static analysis support to scale
- 指南需要静态分析支持来规模化
  - Microsoft Visual Studio and (partially) Clang Tidy
  - Microsoft Visual Studio和（部分）Clang Tidy





C++11 – "it feels like a new language"

C++11 – 感觉像是门新语言

A dense web of
Interrelated
mutually-supporting features
相互支持的特性组成的密集网络。

- Support for concurrency
- 支持并发
  - Memory model, lock-free programming, threads and locks, futures, …
  - 内存模型，无锁编程，线程和锁，期值，…
- Simplify use
- 简化使用
  - Auto, range-for, move semantics, resource-management pointers, lambdas, uniform initialization, constexpr, user-defined literals, …
  - auto、范围 for、移动语义、资源管理指针、lambda 表达式、统一初始化、constexpr、用户定义字面量，…
- Improve support for generic programming
- 改进对泛型编程的支持
  - Lambdas , variadic templates, template aliases, tuples, constexpr, enum classes,
  - Lambda 表达式、变参模板、模板别名、tuple、constexpr、enum class，…
- Increase type safety
- 提高类型安全
  - Type-safe threading and locking, range-for, move semantics, resource-management pointers, user-defined literals, std::array, …
  - 类型安全的线程和锁定、范围 for、移动语义、资源管理指针、用户定义字面量、std::array, …
- Improve support for library building
- 支持对库的开发
  - Lambdas, variadic templates, constexpr, template aliases, tuples, type traits, enable_if, …
  - Lambda 表达式、变参模板、constexpr、模板别名、tuple、类型特征、enable_if，…
- Add and improve standard-library components
- 增加和改进标准库组件
  - threads and locks, regular expressions, time, random numbers, smart pointers, …
  - 线程和锁、正则表达式、时间、随机数、智能指针，…

C++20 – "it approximates my D&E aims"

C++20 – "它接近了我在《设计与演化》中的目标"



- Major language features
- 主要语言特性
  - Modules
  - 模块
  - Concepts
  - 概念
  - Coroutines
  - 协程
  - Improved compile-time programming support
  - 改进的编译期编程支持
- Major standard-library components
- 主要标准库组件
  - Ranges
  - 范围
  - Dates
  - 日期
  - Formats
  - 格式化
  - Parallel algorithms
  - 并行算法
  - Span
  - 跨度
- Many minor language features and standard-library components
- 许多次要语言特性和标准库组件
- Most shipped by early 2021
- 大部分到 2021 年初已经发布
  - In multiple compilers
  - 多个编译器中支持



By "major" I mean
"changes the way we think"
"主要"的意思是
"改变我们的思维方式"



Edison Design Group



Language feature examples – C++11 … C++20

语言特性举例 – C++11 … C++20

- Simplify code
  - 简化代码
- Move semantics
  - 移动语义
- Improved templates
  - 改进的模板
- Support for compile-time computation
  - 支持编译期计算
- Generic programming
  - 泛型编程
- Modules
  - 模块
- Concurrency and parallelism
  - 并发与并行



Deserves its own paper and talk
应当有它自己的文章和演讲

- No serious discussion of libraries
  - 没有库的严肃讨论

Sorry

Simplify code – "little things" matter

简化代码—"小事"也是事

"Small features" can simplify the use of major features
"小特性"能简化主要特性的使用

- Deduced types
- 推导出的类型
  - **auto x = 10;** // *x is an int* 是个整型
  - **for (auto p = v.begin(); p!=v.end(); ++p) cout << *p;** // *p is an iterator* 是个迭代器
- Range-for
- 范围 for
  - **for (int x : v)**
    cout << x; // *eliminate loop variable* 消除循环变量
  - **for (const auto& x : image_vec)**
    cout << x; // *deduce element type* 推导元素类型

Don't say the same thing twice
同一件事不要说两遍

- More advanced type deduction
- 更高级的类型推导
  - **vector v = { 1,2,3,4,5,6 };** // *deduce element type* 推导元素类型
  - **scoped_lock lck { mut } ;** // *deduce lock type from mutex type* 从互斥锁类型推导锁类型
- Much, much more
- 还有许多许多

Eliminate opportunities for errors
消除错误产生的机会

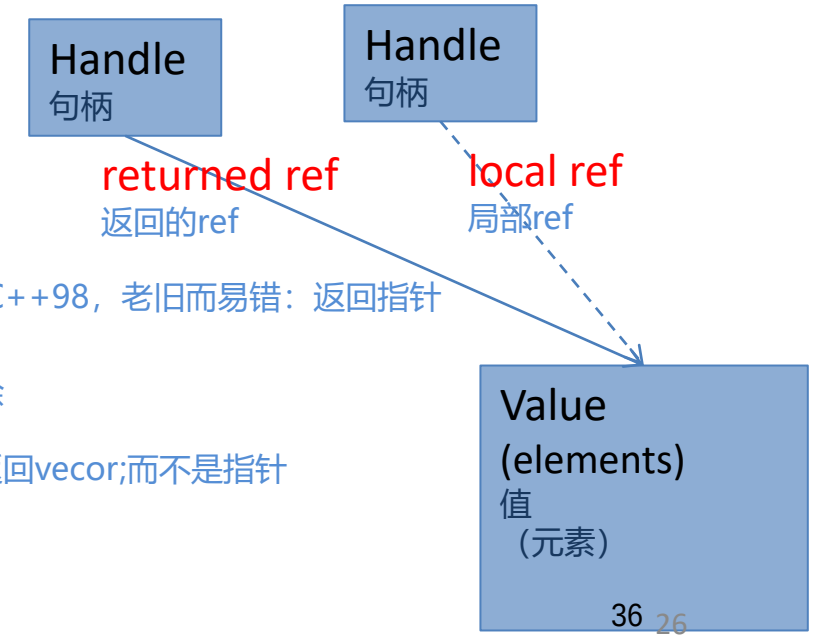
Move semantics (C++11)

移动语义（C++11）

- A general mechanism for moving resources
- 移动资源的通用机制
  - Completes the C++ object model
  - 完善了 C++ 的对象模型
    - Safely and cheaply move objects between scopes
    - 在作用域之间安全而廉价地移动对象
    - No cost and no leaks
    - 没有成本，也没有泄漏
  - Simplifies code that returns large objects
  - 简化了返回大型对象的代码
    - Removes a need for explicit memory management
    - 消除了对显式内存管理的需要
  - Pervasive in the standard library
  - 普遍存在于标准库中



Howard Hinnant




```
vector<double>* p = compute();        // C++98, old and error-prone: return a pointer  C++98，老旧而易错：返回指针
// …
delete p;                                    // remember to delete  一定要记得删除

vector<double> x = compute_vec();     // C++11: return a vector; not a pointer  C++11:返回vecor;而不是指针
```

Improved templates

改进模板



Bjarne Stroustrup



Gabriel Dos Reis

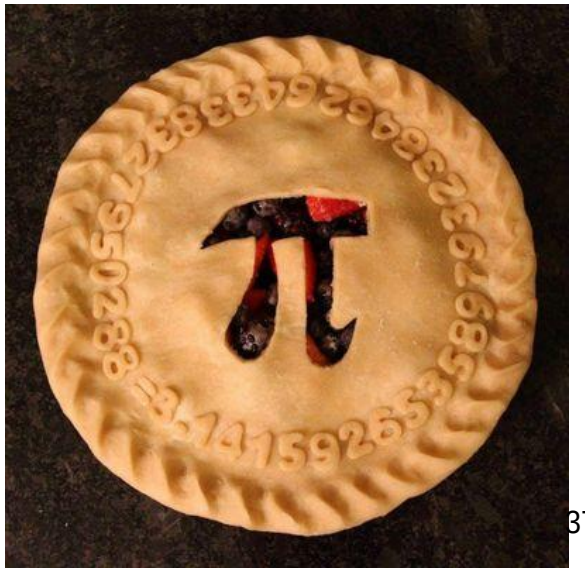
- Lambdas
  - lambda 表达式
- Variadic templates
  - 变参模板
- Variable templates
  - 变量模板

```
template<typename T> constexpr T pi_v = T(3.14159265358979323846264338327950288l);
constexpr double pi = pi_v<double>;
```

- Alias templates
  - 别名模板

```
template<typename T> using Vec = vector<T,My_allocator<T>>;
Vec<int> vi;
```

$$e^{i\pi} + 1 = 0$$



Lambdas

lambda 表达式

- A function object
- 函数对象
  - can carry state and be called as a function
  - 能携带状态并当作函数来调用
  - can be generic
  - 可以是泛型的
  - function objects have been used since 1983
  - 1983 年以来函数对象就被使用了
  - Extremely useful as arguments to algorithms
  - 作为算法的实参，极为有用
  - Can be blindingly fast (inlining)
  - 极其快速（内联）



Jaakko Järvi

- A lambda
- lambda 表达式
  - is a notation for generating a function object
  - 是生成函数对象的写法
  - can be created where it's needed
  - 能在需要用它的地方现场创造

```
sort(v, [](auto& a, auto& b) { return abs(a)>abs(b); });
```

  - can access its environment
  - 能访问它的环境

```
void f(int x, vector<int>& v)
{
        auto val = find_if(v, [x](auto& a) { return a>x; });    // find an element greater than x
        // …
}
```

Call operator
调用运算符

找到一个大于 x 的元素

Defines and initializes state
定义和初始化状态

Variadic templates

变参模板

- Safely pass an arbitrary number of arguments of arbitrary types to a function
- 安全地把任意个任意类型的参数传给某个函数

```
template < typename T, typename... Args >
void printf ( const char * s, const T& value , const Args &... args )
{
   while (*s) {
      if (*s == '%' && *++s != '%') {        // handle formatting characters 处理格式化字符
         std :: cout << value ;              // output first/next argument 输出第一个/下一个实参
         return printf (++s, args...);       // recursive call 递归调用
      }
      std :: cout << *s++;     // output ordinary character 输出普通字符
   }
   throw std :: runtime error (" extra arguments provided to printf ");
}
```



Doug Gregor

```
printf ( "The value of %s is about %g ( unless you live in %s).\n", msg , std::string (" pi "), 3.14159 , " Indiana ");
```

Improved support for compile-time computation

改进对编译期计算的支持

- Make use of abstractions simpler and more affordable
- 让抽象的使用更简单、更能负担得起
  - Traditionally, C++ Gains performance by moving work from run-time to compile-time
  - 传统上，C++通过将工作从运行期转移到编译期来提高性能
    - Virtual tables, inlining, templates, …
    - 虚表、内联、模板……
    - Often eliminates run-time checks and error handlers
    - 通常可以消除运行期的检查和错误处理程序
- Help eliminate error-prone practices
- 有助于消除易出错的做法
  - Macros for computing values
  - 用于计算数值的宏
  - Macros as a untyped constants
  - 作为无类型常量的宏
  - Complex template metaprogramming yielding values
  - 复杂的模板元编程产生的值
  - Magic constants
  - 魔法常量

Compile-time computation

编译期计算

- Ordinary functions that can be evaluated at compile time
- 可以在编译期求值的普通函数

```
constexpr int fac(int n) { return n<2 ? 1 : n *fac(n-1); }
```

- Constexper functions are pure
- constexpr 函数是纯的
  - No sideefects
  - 没有副作用
  - No access to non-local variables
  - 没有对非局部变量的访问
  - No undefined behavior
  - 没有未定义行为
- Steadily generalized over the years
- 多年来稳步泛化
  - Not quite “everything constexpr” (and it shouldn’t be)
  - 不完全是“一切皆 constexpr”（也不应该是）

```
constexpr int fact(int n) { int res = 1;  while(n>1) res *= n--; return res; }
```

- Not just built-in types
- 不只是内建类型
  - **static_assert(weekday{August/25/2021}==Wednesday);**

Local variables
局部变量



Bjarne Stroustrup



Gabriel Dos Reis

C++11 example: Range-**for**, **auto**, and move

C++11 举例：范围 for、auto 以及移动

- Features must work in combination
  - 特性必须能组合起来使用

```
template<typename C, typename V>
        vector<typename C::value_type*>
                find_all(C& c, V v)     // find all occurrences of v in c 找到 c 中出现的所有v
{
        vector<typename C::value_type*> res;
        for (auto& x : c)
            if (x==v)
                res.push_back(&x);
        return res;
}

string m {“Mary had a little lamb”};          // simple test code 简单的测试代码
for (const auto p : find_all(m,‘a’))     // p is a char*          p 是个 char*
    if (*p!='a')
        cerr << "string bug!\n";
```

No explicit allocation
没有显式分配
No deallocation
没有解除分配
No pointer manipulation
没有指针操作
No copy loops
没有拷贝循环
No overflow
没有溢出
No copy overhead
没有拷贝开销

About a third of the size of a C++98 equivalent
大小只有 C++98 相应程序的一半

Generic programming

泛型编程

The backbone of the C++ standard library

C++ 标准库的支柱

- Containers and algorithms (“The STL”)
- 容器与算法（“STL”）
  - **vector**, **list**, **stack**, **queue**, **priority_queue**, **map**, **unordered_map**, ...
  - **sort()**, **partial_sort()**, **is_sorted()**, **merge()**, **find()**, **find_if()**,...
  - Ranges
  - 范围
- Concurrency support (type safe)
- 并发支持（类型安全）
  - Threads, locks, futures, parallel algorithms, ...
  - 线程、锁、期值、并行算法,...
- Time
- 时间
  - **time_point**, **duration**, calendars (**day**, **month**, **year**), time zones, ...
  - time_point、 duration、日历 (day, month, year)、时区、...
- Random numbers
- 随机数
  - distributions and engines
  - 随机分布和引擎
- Numeric types and algorithms
- 数值类型和算法
  - **complex, accumulate()**, **inner_product()**, **iota()**, ...
- Strings and Regular expressions
- 字符串和正则表达式
- I/O streams and Formats
- I/O 流和格式化



Alexander Stepanov

“everything” is parameterized with types;
“每样东西”都以类型来参数化;

some also parameterized with values
有些也以值来参数化

Generic Programming – Templates

泛型编程—模板

- 1980: Use macros to express generic types and functions
- 1980: 使用宏来表达泛型类型和函数
- 1987 (and current) aims for templates:
- 1987 年（和现在）模板的目标：
  - Extremely general/flexible
  - 极为通用/灵活
    - “must be able to do much more than I can imagine”
    - "必须能够做得比我能想象的多得多"
  - Zero-overhead
  - 零开销
    - vector/Matrix/… to compete with C arrays
    - 向量/矩阵/...与C语言数组竞争
  - Well-specified interfaces
  - 规范的接口
    - Implying overloading, good error messages, and maybe separate compilation
    - 意味着重载、良好的错误信息，以及可能的单独编译
- 2004-2009: C++0x Concepts
- 2004-2009: C++0x 概念
  - Modeled on classes
  - 以类为模型
    - failed: grew complex, costly, incomplete
    - 失败：越来越复杂、成本高、不完整
- 2003-2020: Concepts
- 2003-2020: 概念
  - Compile-time predicates
  - 编译期谓词
  - Precise specification of a template’s requirements on its argument
  - 精确说明模板对其参数的要求



Concepts – constraining template arguments

概念—约束模板实参

- C++98 … C++17 checks template arguments at the point of template instantiation
- C++98 … C++17 在模板实例化的时候检查模板参数

```
template<class Iter>
 void sort(Iter first, Iter last) { /* use first and last as iterators使用 first 和 last 作为迭代器 */ } // the definition is needed for use 定义是需要的

sort(vec.begin(),vec.end()); // fine, assuming vec is a vector 可以，假定 vec 是 vector
sort(7,9); // error: ints are not iterators; the error message is horrible 错误：整型不是迭代器；错误信息很糟糕
```

Any type: no detailed requirements

任何类型：没有详细要求

- C++20 checks template arguments at the call point
- C++20 在调用点检查模板参数

```
template<sortable Iter>
void sort(Iter,Iter);          // just a declaration is needed for use 只需要声明就可以了

sort(vec.begin(),vec.end()); // fine, assuming vec is a vector 可以，假定 vec 是 vector
sort(7,9);                                  // error: ints are not sortable 错误：整型不是 可排序的
```

how do we define "sortable"?

我们如何定义 "可排序"

Morgan Stanley

Concepts – use patterns

概念—使用模式

- A concept is a compile-time predicate
- 概念是编译期谓词
  - A function run at compile time yielding a Boolean
  - 编译期运行并得出一个布尔值的函数
  - Takes a set of types and/or values as arguments
  - 接受一组类型和/或值作为参数
- A concept can be built from fundamental language properties
- 概念可以从基本语言属性中创建出来
  - Use patterns: show what operations you need
  - 使用模式：表明你需要什么操作
  - Handles mixed mode arithmetic and implicit conversions
  - 处理混合模式算数和隐式转换



Bjarne Stroustrup



Gabriel Dos Reis

```
template<typename T, typename U = T>
concept equality_comparable = requires(T a, U b) {
        {a==b} -> Boolean;
        {a!=b} -> Boolean;
        {b==a} -> Boolean;
        {b!=a} -> Boolean;
}
```

There are libraries of concepts <concepts>: equality_comparable

我们有概念库 <concepts>: equality_comparable

G. Dos Reis and B. Stroustrup: Specifying C++ Concepts. POPL'06.

规约 C++ 概念

Concepts – composition

概念—组合

- A concept is usually constructed from other concepts
- 概念通常由其他概念构建而成

```
template<typename R>
concept Sortable_range =
        random_access_range<R>// has begin()/end(), ++, [], +, … 有 beign()/end(),++,[],+, …
        && sortable<iterator_t<R>>;// can compare and swap elements, 能够比较和交换元素

template<typename R>
concept Forward_sortable_range =
        forward_range<R>          // has begin()/end(), ++; no [] or + 有begin()/end(), ++; no [] or +
        && sortable<iterator_t<R>>;       // can compare and swap elements， 能够比较和交换元素
```



Andrew Sutton

There are libraries of concepts

<ranges>: random_access_range and sortable

我们有概念库

<ranges>: random_access_range 和 sortable

Concepts – Overloading

概念—重载

We can have many sort() templates
我们可以有许多 sort() 模板

- We can select based on template argument requirements
- 我们可以基于模板实参要求进行选择

```
template<sortable Iter> void sort(Iter,Iter);    // a C++98 sort()
void sort(Sortable_range auto&);                 // a C++20 sort
void sort(Forward_sortable_range auto&);         // sort lists, etc. 对列表排序

sort(vec.begin(),vec.end());                     // OK, as ever 可以，像以往一样
sort(lst.begin(),lst.end()); // error, as ever, but with good immediate error message 错误，像以往一样，但是现在有了良好而即时的错误信息
sort(lst);                                       // sort(Forward_sortable_range auto&)
sort(vec)                                        // sort(Sortable_range auto&)
```

Zero overhead compared to unconstrained templates
跟不受约束的模板相比来说，是零开销

- The compiler knows that
- 编译器知道
  - "**Sortable_range** is stricter/better than **Forward_sortable_range**"
  - "Sortable_range 比 Forward_sortable_range 严格/更好"
  - We don't have to explicitly specify that
  - 我们无需显式规约这一点

Don't force the user to do what a machine does better
不要强制用户去干机器能干得更好的事情

Concepts – representing relationships

概念—表示关系

- Many (most?) concepts take more than one argument
- 许多（大多数）概念接受大于一个参数
  - Represent the relationships among the arguments
  - 表示参数之间的关系
  - Concepts are not just “types of types”
  - 概念不只是“类型的类型”

```
template<
        random_access_range R,
        sortable<iterator_t<R>> Pred     // operates on iterators to elements of R
            >
void sort(R& r, Pred p = ranges::less); // sort r using p
```

指向 R 元素的迭代器上面的操作

使用 p 来对 r 排序



Concepts – and types

概念—和类型

- A type
- 类型
  - Specifies the set of operations that can be applied to an object
  - 规约可以在某对象上进行的操作
    - Implicitly and explicitly
    - 隐式地和显式地
    - Relies on function declarations and language rules
    - 依赖于函数声明和语言规则
  - Specifies how an object is laid out in memory
  - 规约对象在内存中如何布局
- A ***single-argument*** concept
- 单参数概念
  - Specifies the set of operations that can be applied to an object
  - 规约可以在某对象上进行的操作
    - Implicitly and explicitly
    - 隐式地和显式地
    - Relies on use patterns
    - 依赖于使用模式
      - reflecting function declarations and language rules
      - 反映函数声明和语言规则
  - Says nothing about the layout of the object
  - 不关心对象的布局
  - Enables the use of a set of types
  - 可以使用集合中的类



Ideal: Use concepts where we now use types, except for defining layout

理想：在我们现在使用类型的场合使用概念，除了需要定义内存布局的情况之外

Concepts – definition checking

概念—定义检查

- We dreamed of testing template definitions in isolation
- 我们曾梦想孤立地测试模板定义

```
void advance(std::input_iterator* p, int n)   // input_iterator is a concept
{
        p+= n;        // error? input_iterator doesn't require +=
}
```

input_iterator 是个概念

错误？Input_iterator 不需要+=?

- We figured out how to implement that
- 我们想出了如何实现这点
- We changed our mind
- 我们改变了主意
  - We need stable interfaces
  - 我们需要稳定的接口
    - Change interface every time we change "housekeeping details"? (no!)
    - 每次改动"琐碎细节"的时候都要改变接口？（不行！）
      - Debug output
      - 调试输出
      - Telemetry
      - 遥测
      - Data collection
      - 数据收集

Ranges - Concepts in the standard library

范围—标准库中的概念



Casey Carter

- Notational convenience (incl. overloading)
- 写法上的便利（包括重载）

```
sort(v.begin(),v.end());
sort(v);
```

- Views and filters
- view 和 filter

```
vector vi { 1, -1, 12, 14, 28, 42, 24, -99 100, 200};
```

- `auto p = ranges::find(vi|views::take(8), 42);` // *look for 42 in the first 8 elements of vi* 在 vi 的前 8 个元素中寻找 42

  `ranges::sort(vi|views::drop(2));` // *sort every second element of vi* 对于 vi 每第二个元素排序



Eric Niebler

Modules

模块

- The C header-file model doesn’t offer modularity
- C 头文件模型不提供模块化
  - **#include** is textual inclusion
  - #include 是文本包含
- This has been known “forever”
- “自古以来”
  - Mentioned in D&E
  - 在《设计和演化》中提及
  - Attempts to remedy
  - 尝试补救
    - Daveed Vandevoorde (2003-2012)
    - Doug Gregor (2012)
- C++20 design (2009 – 2019)
- C++20 中的设计（2009-2019）
  - Gabriel Dos Reis, Richard Smith, Nathan Sidwell



Gabriel Dos Reis



Richard Smith



Nathan Sidwell

Modules – Modularity

模块—模块化

- Order dependence
- 依赖于顺序

```
#include "a.h"
#include "b.h"
```

can be different from

可以不同于

```
#include "b.h"
#include "a.h"
```

- **#include** is transitive
- #include 是传递性的
  - much repeated compilation
  - 许多重复的编译
  - Subtle bugs
  - 难以察觉的缺陷

- Modularity
- 模块化

```
import a;
import b;
```

is same as

等同于

```
import b;
import a;
```

- **import** is not transitive
- Import 没有传递性
  - Much compilation can be done once only
  - 大量编译只需一次完成

- The transition is fiendishly hard to accomplish
- 过渡工作很难完成
  - Huge code bases
  - 巨大的代码仓库
  - Established tool chains and procedures
  - 既有的工具链和流程

Modules – Code hygiene

模块—代码卫生

- Better code hygiene: modularity (especially protection from macros)
- 更好的代码卫生：模块化（尤其防止宏）

```
export module sequence_printer;        // we are defining a module
```
我们定义一个模块

```
import iostream;        // iostream details not leaked to module user
using namespace std;// not leaked to module user
```
iostream 的细节不会泄露给模块用户

不会泄露给模块用户

```
export          // this is the (only) function seen by users
    template<printable_input_range R>          // a range of elements that can be printed using <<
```
这是用户能看见的（唯一）函数

//可以使用 << 打印的某范围中的元素

```
void print(R& s)
{
        cout << "{\n";
        for (const auto& val: s)  cout << "   " << val << '\n';
        cout << '}';
}
```

Modules – Compile time

模块—编译期

- Header files (character based)
- 头文件（基于字符）

```
#include<libGalil/DmcDevice.h>

int main() {

        LibGalil::DmcDevice("192.168.55.10");}
```

```
// 457440 lines, 151268 non-blank   457440 行， 151268 非空行

// 1546 milliseconds to compile      编译时间 1546 毫秒
```

A 25 times speedup!

(Don't expect that in every cas

25 倍的加速

（别期望每个用例都能达到）

- Modules (semantic graph)

```
Import libGalil;

int main() {

        LibGalil::DmcDevice("192.168.55.10");
}
```

```
// 5 lines, 4 non-blank      5 行， 4 行非空

// 64 milliseconds to compile      编译时间 64 毫秒
```

Modules – simplify library use

模块 – 简化库的使用

- We don't yet have modules for the standard library
- 我们还没有标准库的模块
  - The 98 standard headers are a source of complexity and confusion – a barrier to entry
  - 98 标准的头文件是复杂性和困惑的来源 – 拦路虎

C++23

```
import std;
int main()
{
        std::cout << "Hello modern world!\n";
}
```

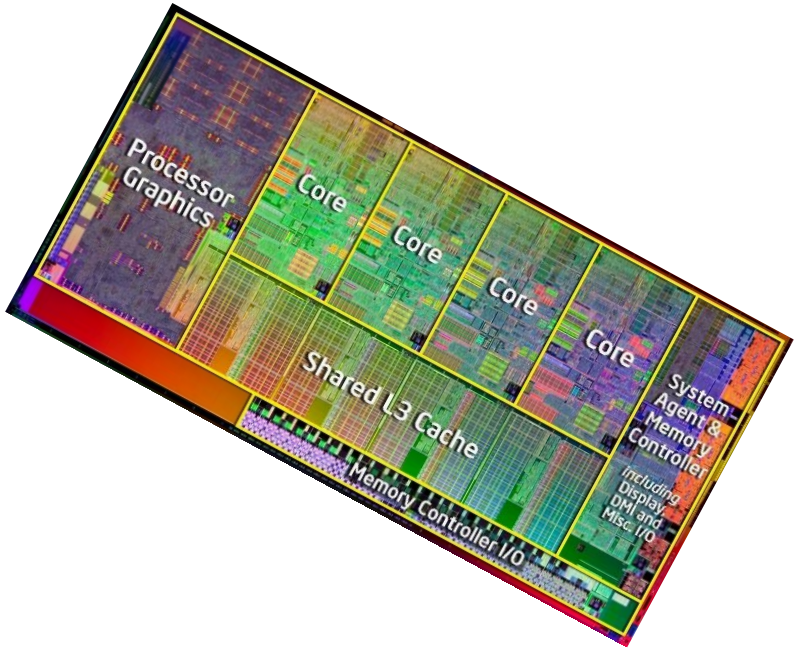
- We can afford that
- 我们能负担得起它
- The idea generalizes
- 思想的泛化
  - Larger more logical modules
  - 更大更有逻辑的模块

	#include needed headers 需要的头文件	Import needed headers	import std	#include all headers	Import all headers 所有头文件
"Hello world" (<iostream>)	0.87s	0.32s	0.08s	3.43s	0.62s
"Mix" (9 popular headers) 9 个流行的头文件	2.20s	0.77s	0.44s	3.53s	0.99s

Concurrency and parallelism

并发和并行

- C++: used for concurrency and parallelism from at least 1984
- C++: 至少从 1984 年以来就用于并发和并行
  - Coroutines: 1981-1990
  - 协程：1981-1990
- C++11 – the traditional foundation
- C++11 – 传统的基础
  - Memory model
  - 内存模型
  - Lock-free programming
  - 无锁编程
  - Atomics
  - atomic
  - Type-safe POSIX/Windows-style treads, mutex, etc.
  - 类型安全的 POSIX/Windows 风格的线程、互斥锁等等
  - Future and promise
  - 期值 和 promise
- C++14, C++17, and C++20
  - Many minor improvements (stop tokens, scoped_lock, Fences and barriers, …)
  - 许多次要改进（停止令牌、scoped_lock、栅栏和屏障
  - Parallel algorithms
  - 并行算法
  - Coroutines (synchronous and asynchronous)
  - 协程（同步和异步）
  - Networking (a Technical Specification)
  - 网络（技术规范 TS）



They're back!

Coroutines

协程

- Coroutines was the key to C++ in the early years
- 协程在早年对于 C++ 很关键
  - Task library 1981-1990
  - 任务库 1981-1990
  - Killed by lack of support on SPARC
  - 由于缺乏 SPARC 上的支持被干掉了
- C++20 coroutines: synchronous and asynchronous
- C++20 的协程：同步和异步
  - Scales to hundreds of thousands of tasks
  - 扩展到成百上千个任务上



Gor Nishanov

```
generator<int> fibonacci() // generate 0,1,1,2,3,5,8,13 … 生成 0，1，1，2，3，5，8，13
{
        int a = 0; int b = 1;          // initial state 初始状态
        while (true) {
                int next = a+b;
                co_yield a;     // return next Fibonacci number 返回下一个Fibonacci 数
                a = b;  b = next;              // update state 更新状态
        }
}
```

Preserve local state between calls C++23
在调用间保存局部状态
C++23

```
for (auto v: fibonacci())   // use
    cout << v << '\n';
```

Fast pipelines and generators
Simplify asynchronous programming
高速的管线和生成器简化了异步编程

Parallel algorithms

并行算法

- Optional parallel and/or vectorized execution
- 可选的并行和/或矢量执行
  - e.g, **sort(par,b,e)** and **sort(unseq,b,e)**
- The traditional STL algorithms,
- 传统的 STL 算法
  - e.g., **find(seq,b,e,x)**
  - but no **find_all(par,b,e,x)** or **find_any(par_unseq,b,e,x)**
  - 但还没有find_all(par,b,e,x) 或 find_any(par_unseq,b,e,x)
- parallel algorithms:
- 并行算法:
  - **for_each**
  - **reduce** // *parallel accumulate* 并行累加
  - **exclusive scan**
  - **inclusive scan**
  - **transform reduce**
  - **transform exclusive scan**
  - **transform inclusive scan**
  - …



C++ in 2020 (the peak so far)

2020 年的 C++（到目前为止的峰值）

- An invisible foundation of “everything”
- "一切 "的无形基础
  - All industries and all countries (and beyond)
  - 所有工业和所有国家（并超越这些）
  - Better than ever (Shorter, safer, faster code)
  - 比以往更好（更短、更安全、更快的代码）
  - More users than ever (at least 5M and increasing)
  - 比以往任何时候都多的用户（至少有 500 万，而且还在增加）
- Huge standards committee participation
- 巨大的标准委员会参与
  - Dangers of overenthusiasm and impatience
  - 过度热情和丧失耐心的危险
- Weak in tools
- 工具较弱
  - But improving
  - 但正在改善
- Often mischaracterized
- 经常被错误描述
  - People relying on decades old and/or secondary sources
  - 人们依赖几十年前的资料和/或二手资料
- Weak in education
- 在教育方面很薄弱
- Focus on correctness/safety
- 注重正确性/安全性
  - Core guidelines
  - 核心指南



Retrospective – What could have been done better?

回顾—什么可以做得更好?

- Some key ideas weren't communicated effectively
- 一些关键的想法没有得到有效的沟通
  - Especially to the academic and educational communities
  - 特别是对学术和教育界
- Lack of emphasis on tools
- 缺少对工具的强调
  - Standard packaging and build system
  - 标准的打包和构建系统
  - Standard non-textual program representation
  - 标准的非文本程序表示法
- Lack of emphasis on use of and distribution of common libraries
- 对通用库的使用和分发缺乏重视
  - Centralized repository for libraries
  - 库的集中仓库
  - General framework for interoperability of libraries
  - 库的互操作性的通用框架
- Standard library
- 标准库
  - Higher-level concurrency support (e.g. a message queue, callbacks for futures)
  - 更高级别的并发支持（如消息队列、期值的回调）
  - More libraries support for specific large communities
  - 为特定的大型技术社区提供更多的库支持
- Language
- 语言
  - "Smart references": **operator.()**
  - "智能引用": operator.()
  - Unified function call: **x.f(y) == f(x,y)**
  - 统一函数调用: x.f(y) == f(x,y)
  - Multi-methods: **bool intersect(virtual Shape*, virtual Shape*);**
  - 多方法: bool intersect(virtual Shape*, virtual Shape*);



And many, many "details"

还有很多很多的"细节"

Retrospective – What did C++ get right?

回顾—C++ 做对了什么?

- Early articulation of principles
- 早早阐明了原则
- Stability ***and*** evolution
- 稳定性和演化
- The standards effort
- 标准化工作
  - Difficult, hard work, but it kept the community together
  - 困难的、艰苦的工作，但它使社区团结一致

- Static type system
- 静态类型系统
  - Zero-overhead abstractions
  - 零开销抽象
  - Resource management
  - 资源管理
- Direct access to hardware and system resources
- 直接访问硬件和系统资源

- Is C++20 better than C++98, C++11, C++14, and C++17?
- C++20 比 C++98、C++11、C++14 和 C++17 好吗?
  - Yes, significantly so
  - 是的，显著地好
  - More expressive type system
  - 更具表现力的类型系统
  - Shorter, safer, faster code
  - 更短、更安全、更快的代码



C++23, C++26, … – we have a plan

C++23、C++26、 … —我们有个计划

- Top priorities:
- 最高优先级：
  - Library support for coroutines (C++23)
  - 库中对于协程的支持（C++23）
  - A modular standard library (C++23)
  - 一个模块化的标准库（C++23）
  - Executors (C++26)
  - 执行器（C++26）
  - Networking (C++26)
  - 网络（C++26）
- Also make progress on
- 同时在以下方面取得进展
  - Multidimensional access (C++23)
  - 多维访问（C++23）
  - Flat_map and print (C++23)
  - flat_map 和 print(C++23)
  - Static reflection (C++26)
  - 静态反射（C++26）
  - Pattern matching (C++26)
  - 模式匹配（C++26）
  - Contracts (C++26)
  - 契约（C++26）
- After that
- 此外
  - Everything else
  - 所有其他的
  - Many dozens of proposals, many excellent
  - 几十个提案，很多都很棒



Be careful

Remember the Vasa!

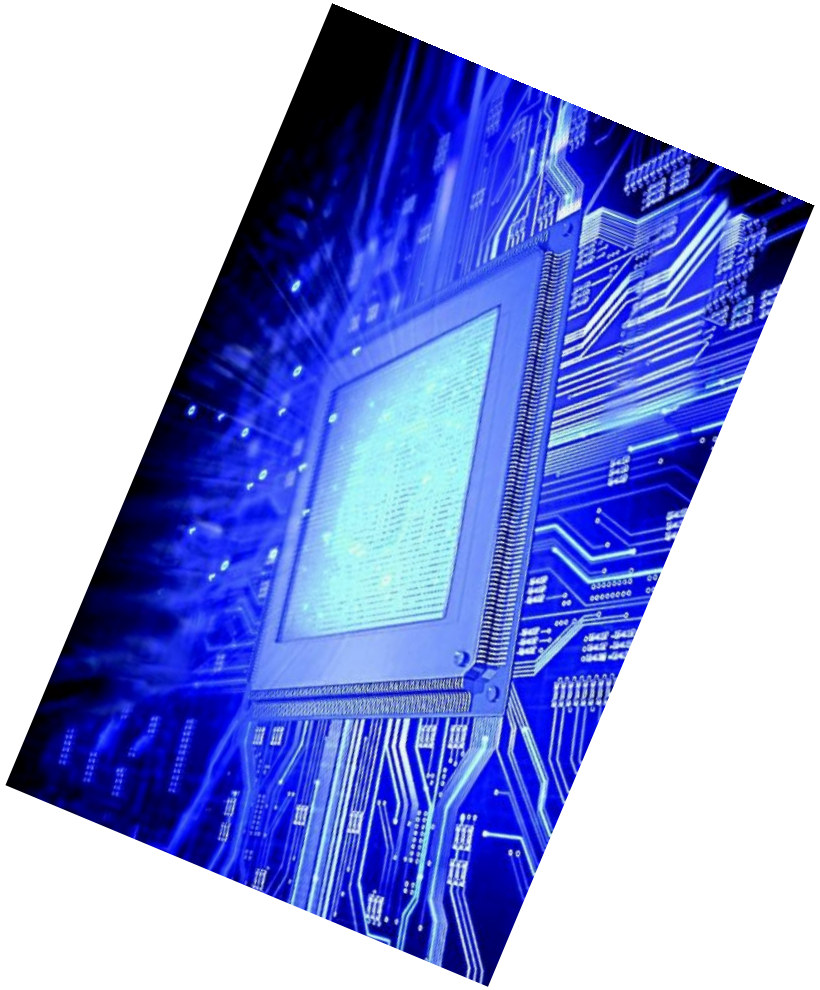
当心

记得瓦萨号!

Key areas of the C++ language

C++ 语言的关键领域

- Static type system
- 静态类型系统
  - Equal support for built-in types and user-defined types
  - 对内置类型和用户定义类型的平等支持
  - Value and reference semantics
  - 值和引用语义
- Systematic and general resource management
- 系统而通用的资源管理
  - RAII: scope based with constructors and destructors
  - RAII：基于作用域的构造函数和析构函数
- Efficient object-oriented programming
- 高效的面向对象编程
- Flexible and efficient generic programming
- 灵活而高效的通用编程
- Compile-time programming
- 编译期编程
- Direct use of machine and operating system resources
- 直接使用机器和操作系统资源
  - Concurrency support through libraries supported by intrinsics
  - 由基于内建函数的库提供并发支持
- HOPL papers
- HOPL 论文
  - A History of C++: 1979-1991. March 1993
  - 《C++的历史：1979-1991》1993 年 3 月
  - Evolving a language in and for the real world: C++ 1991-2006. June 2007
  - 《在真实世界演化语言并服务真实世界：C++ 1991-2006》2007 年 6 月
  - Thriving in a Crowded and Changing World: C++ 2006–2020. June 2020
  - 《在拥挤和变化的世界中茁壮成长：C++ 2006–2020》2020 年 6 月



CPP-Summit 2022



李建忠

Boolan 首席软件专家

Boolan首席软件专家，全球C++及系统软件技术大会主席。对面向对象、设计模式、软件架构、技术创新有丰富经验和深入研究。 主讲《设计模式纵横谈》，《面向对象设计》课程，影响近百万软件开发人员，享有盛誉。曾于 2005年-2010年期间担任微软最有价值技术专家，区域技术总监。拥有近二十年软件技术架构与产品经验，为包括众多世界500强公司在内的知名品牌提供高端IT技术咨询和教育服务，广受好评。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

使用概念重塑“零开销抽象”设计模式

李建忠

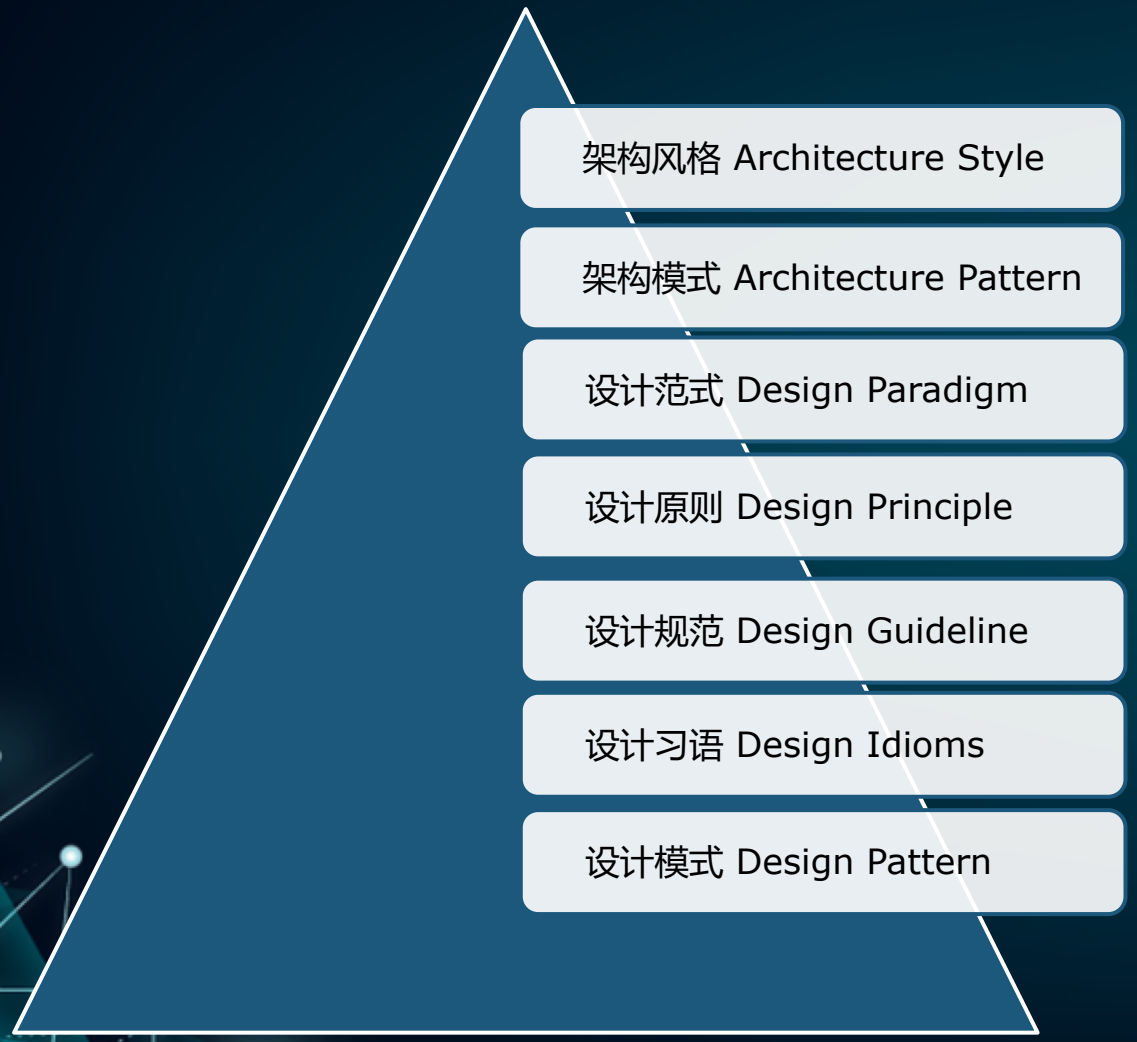
Boolan 首席软件专家

演讲摘要

面向对象设计模式自GOF引入业界以来一直是软件架构师实现“松耦合、高内聚”的设计法宝之一。但面向对象固有的性能开销也是相关模式在实践中的一大障碍。

以“零开销抽象”为核心精神的C++自C++20以来引入的概念（concept）机制，为泛型设计模式在性能攸关的领域开拓了全新的方法论和工具支撑。本演讲将结合该领域的前沿发展，研究如何使用基于“零开销抽象”的概念机制，重塑“性能关键”领域的抽象设计，实现“高性能”和“设计抽象”的双重目标。

设计模式是软件设计的“最后一公里”

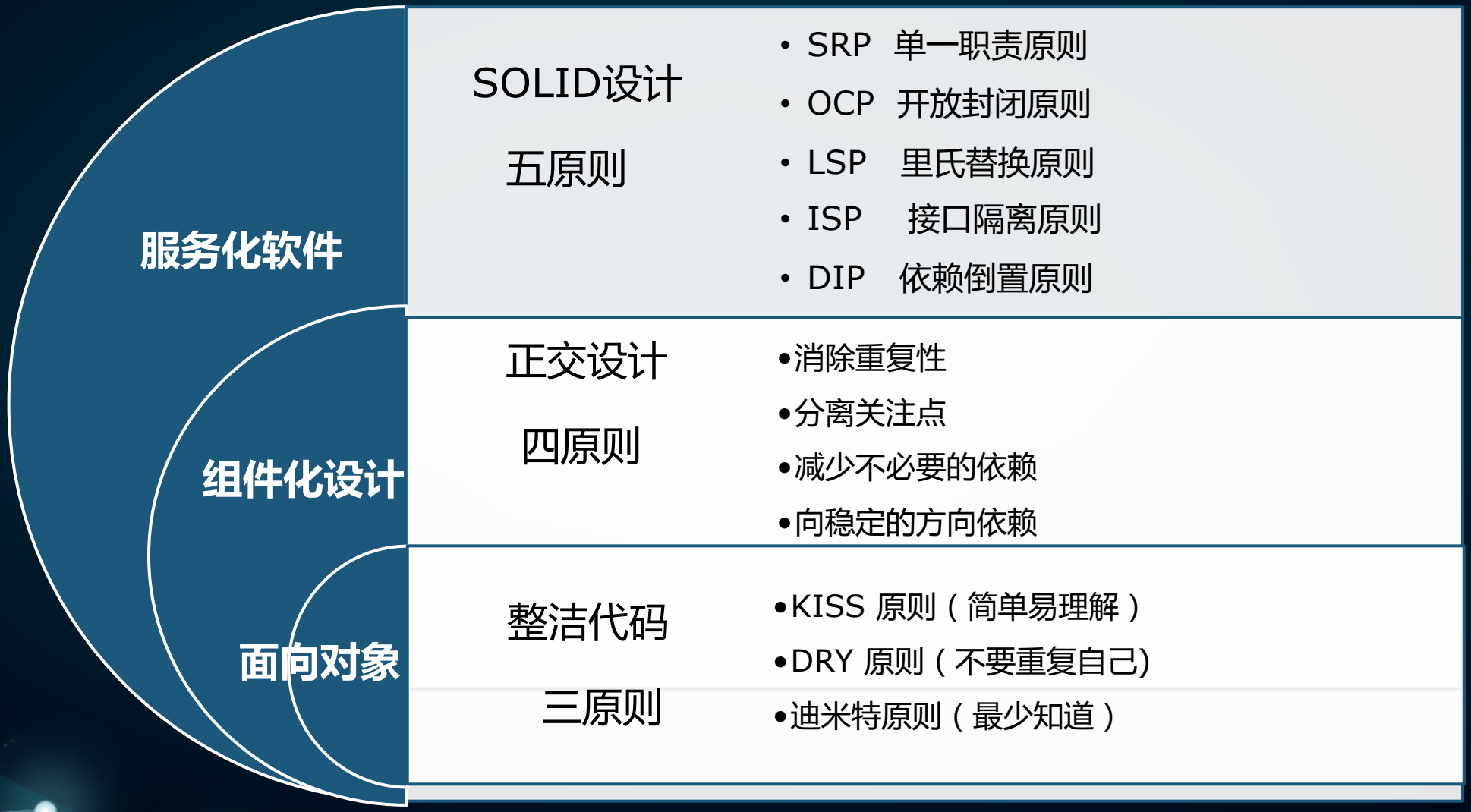


“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次次地复用前人的设计经验而不必重新发明轮子”。

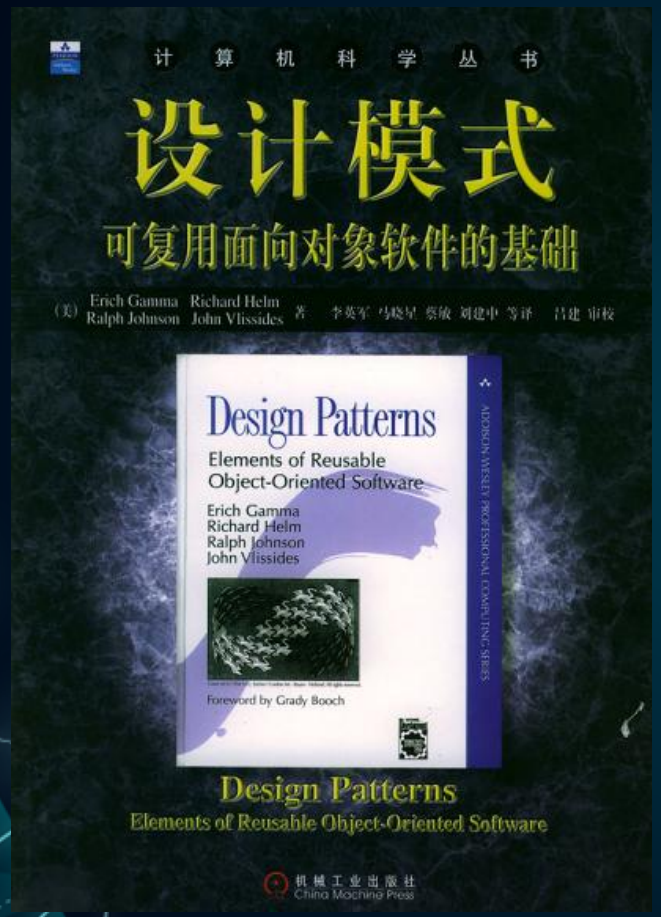
——Christopher Alexander

面向接口设计—— 从原则到模式的中枢桥梁

面向接口设计



GOF 面向对象设计模式



历史性著作《设计模式：可复用面向对象软件的基础》一书中描述了23种经典面向对象设计模式，创立了模式在软件设计中的地位。

但GOF主要使用面向对象，更具体地来说是以虚函数和继承为代表的运行时多态，作为主要设计手段；这种面向对象多态在性能和设计两方面都有不可避免的负担。

C++语言的“立国精神”：零开销抽象

我当初设计C++是为了回答这样一个问题：如何直接操作硬件，同时又支持高效、高级的抽象？经过多年发展，C++一直保持着两方面的关注：

1. 语言构造直接映射到硬件设备
2. 零开销抽象

—— Bjarne Stroustrup， C++之父

“零开销”的含义为：

1. 不用的东西，不需要付出代价
2. 使用的东西，手工代码不会做得更好

“抽象”指的是函数、类、模板、概念、Lambda、别名等。

面向对象接口 —— 抽象基类

抽象基类是面向对象接口设计的核心，也是诸多设计模式的基石。实现运行时多态辨析，也称动态多态（ Dynamic Polymorphism ）。

```cpp
class Document{
public:
    virtual void process()=0;
    virtual ~Document()=default;
};

void invoke(){

    auto pDoc=make_unique<Markdown>();

    pDoc->process();
}
```

```cpp
class Markdown: public Document{
public:
    void process() override {
        //...
    }
};

class HTML: public Document{
public:
    void process() override {
        //...
    }
};
```

概念（Concept）的发展简介

C++之父Bjarne Stroustrup在早期曾提出模板应符合以下三个标准

- 全面的通用性和表达力
- 零开销，与手工编码相比
- 规范化的接口

STL之父 Alex Stepanov在 70年代研究泛型编程时提出“代数结构 Algebraic Structures”，90年代开始使用“概念 Concept”来明确表示C++模板的接口，此后被C++标准委员会采纳。

概念（Concept）从2003提上日程、经过众多专家努力，直到2020年正式成为C++20标准，成为“第一等的语言公民”。

泛型编程接口 —— 概念 Concept

概念为泛型编程定义了“显式约束”的接口合同，实现编译时多态辨析，也称静态多态（ Static Polymorphism ）。

```
template <typename T>
concept Document = std::semiregular<T> &&
requires(T t) {
    t.process() ;
};

template<Document T>
void invoke(){
    T doc;
    doc.process();
}
```

```
class Markdown{
public:
    void process() {
        //...
    }
};

class HTML{
public:
    void process() {
        //...
    }
};

invoke<Markdown>();
```

概念的主要特点

```
template <typename T>
concept Document =requires(T t) {
    t.process() ;
};
```

- C++概念是一种显式的类型接口合同，在编译时执行类型约束检查。
- 概念的本质是一组对类型参数T的编译时谓词，表达式求值 true 或false
- 概念仅仅是约束，不是代码、没有类型、存储、生命周期、地址....
- 概念可以帮助更好的理解泛型组件之间的合约；易维护，易沟通、更友好的出错信息、
- 有概念约束的版本，可以参与重载辨析，相对于通用版本是一个特化版（更优先辨析）
- 概念是比类型Traits、编译时if表达式都更强大的抽象。

性能比较 —— 概念 VS. 抽象类

	抽象类	概念
值/引用语义	引用语义友好	值语义友好
内存模型	参与实现类内存布局	概念不定义任何内存布局
多接口重复	菱形继承的重复性内存	菱形实现的重复性被消除
虚函数表	运行时信息存储成本	无添加运行时信息存储成本
接口函数调用	运行时绑定	编译时绑定
系统优化支持	很难支持	很好支持
总结	有一定开销的抽象机制	零开销抽象机制

抽象类—— 引用语义友好

- 对象实例必须采用引用语义
- 强制对象堆分配
  - 分配/释放成本高
  - 对象碎片化、缓存不友好
- 增加间接层数据结构
  - 裸指针
  - unique_ptr
  - shared_ptr
- 容易倾向共享引用
  - 共享引用增加成本
    - 指针双倍尺寸
    - 引用计数块
  - 多线程同步代价高

```
class Document{
public:
    virtual void process()=0;
    virtual ~Document()=default;
};

class Markdown: public Document{ };

class HTML: public Document{ };

Document* pDoc1=new Mardown{};
unique_ptr<Document> pDoc2=make_unique<Markdown>();
shared_ptr<Document> pDoc3=make_shared<Markdown>();

delete pDoc1;
```

概念—— 值语义友好

- 对象实例可以采用值语义
- 存储性能好，分配/释放成本低
  - 优先栈存储
  - 即使在容器中，容易实现内存连续
- 无间接层数据结构
  - “正则对象”管理便捷
  - 拷贝、移动、赋值性能好
- 避免共享引用
  - 无多线程同步问题

```
template <typename T>
concept Document = std::semiregular<T> &&
requires(T t) {
    t.process() ;
};

template<Document T>
void invoke(){
    T doc;
    doc.process();

    int count=10;
    vector<T> vec(count);
    vec.emplace_back(...);
    for(auto& elem : vec) {
        elem.process();
    }
}
```

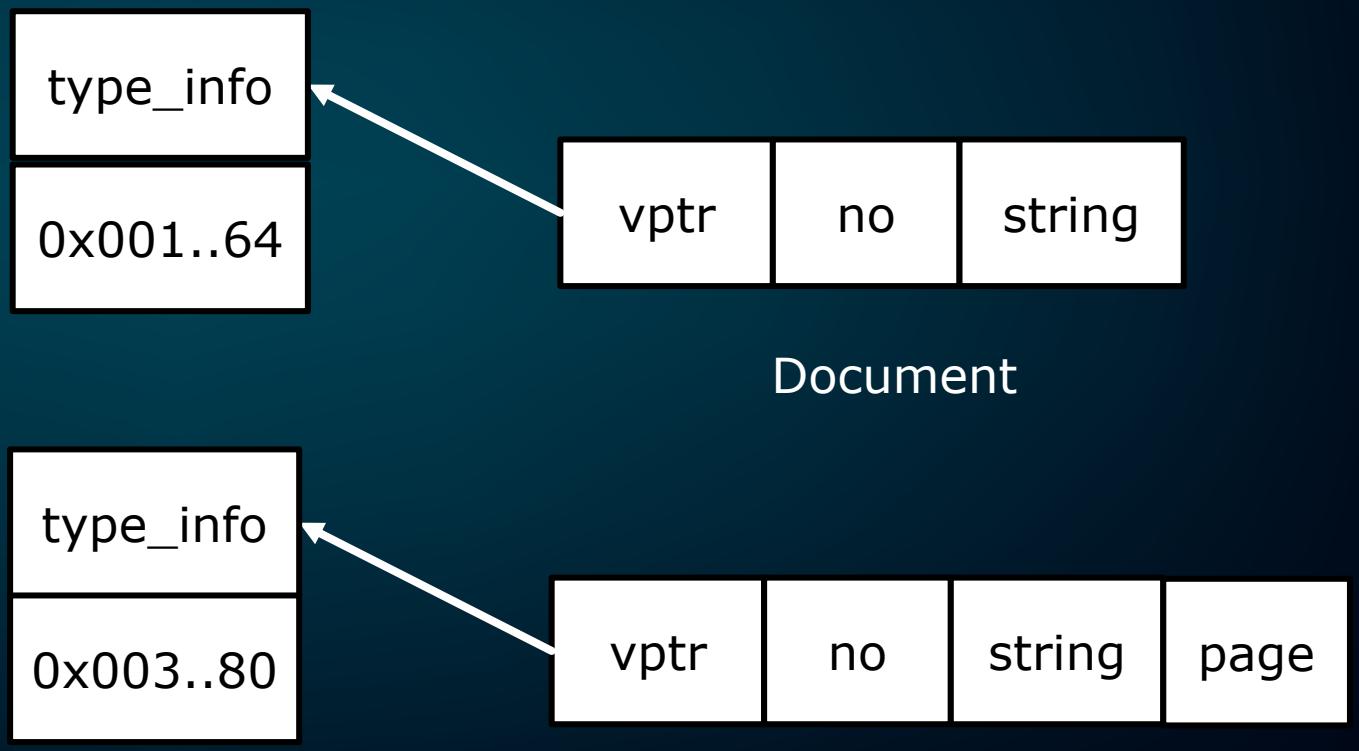
抽象类——内存模型、虚函数表

实现类（子类）内存布局必须 “包含”抽象类（基类）内存布局

抽象类增加了虚函数和RTTI（运行时类型信息）存储成本

```
class Document {
    int no;
    string title;
public:
    virtual void process()=0;
    virtual ~Document()=default;
};

class Markdown: public Document{
    int page;
public:
    void process() override {
        //...
    }
};
```



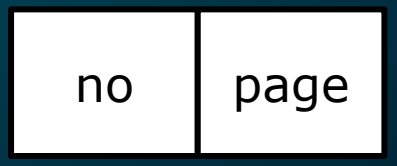
概念——内存模型、虚函数表

概念不定义任何内存布局，对实现类无任何影响

无需支持运行时辨析，不会增加虚表结构和RTTI存储成本

```
template <typename T>
concept Document = std::semiregular<T> &&
requires(T t) {
    t.process() ;
};

class Markdown {
    int no;
    int page;
public:
    void process() {
        ...
    }
};
```



抽象类——多接口重复

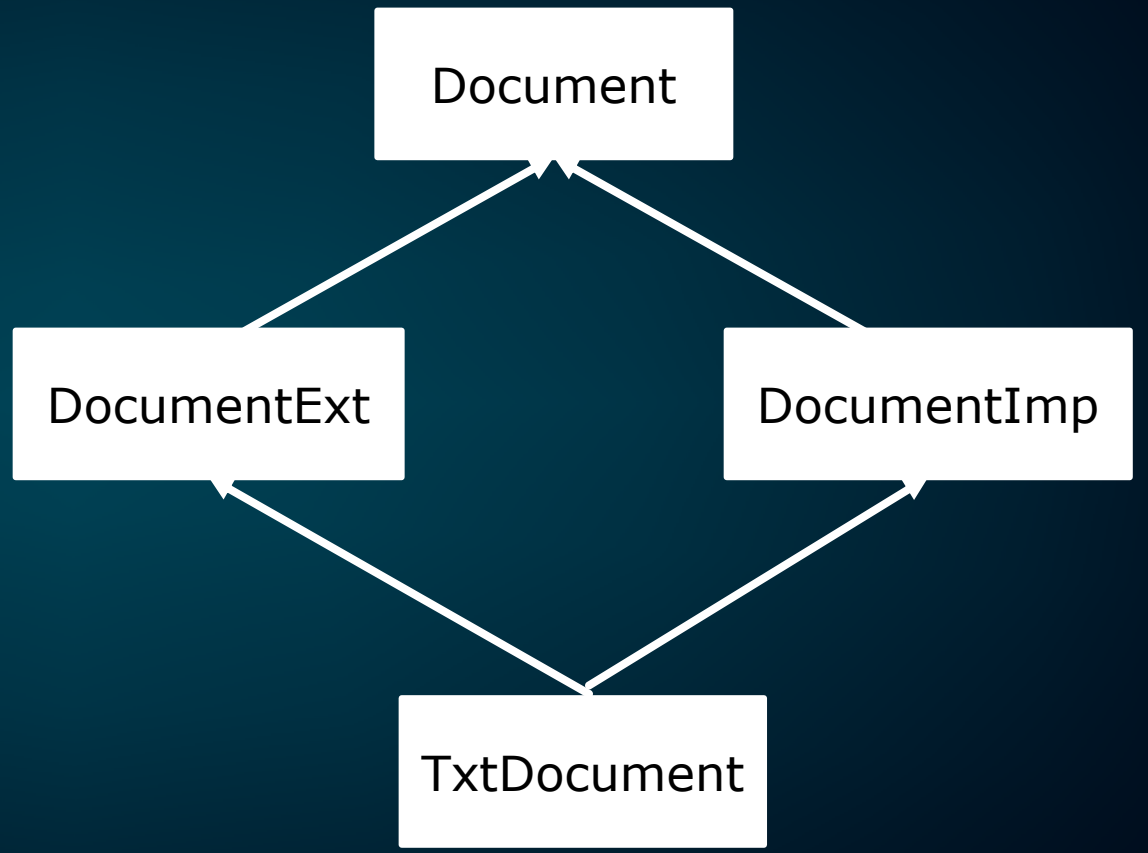
菱形继承带来的重复性内存负担

```
class Document { };

class DocumentExt:
    public virtual Document {
};

class DocumentImp:
    public virtual Document {
};

class TxtDocument:
    public DocumentExt,
    private DocumentImp{
};
```



概念——多接口重复

因为概念不定义内存模型，菱形实现的重复性自然被消除

```
template <typename T>
concept Document = std::semiregular<T> &&
requires(T t) {
    t.process() ;
};

template <typename T>
concept DocumentExt = Document<T> &&
requires(T t) {
    t.extend() ;
};

template <typename T>
concept DocumentImp = Document<T> &&
requires(T t) {
    t.process_imp() ;
};
```

```
template <typename T>
concept TxtDocument = DocumentExt<T> &&
DocumentImp<T> &&
requires(T t) {
    t.read_text() ;
};
```

等价于

```
template <typename T>
concept TxtDocument=std::semiregular<T> &&
requires(T t)
{
    t.process() ;
    t.extend() ;
    t.process_imp() ;
    t.read_text() ;
};
```

抽象类 —— 运行时函数调用

- 接口函数（虚函数）调用时采用运行时绑定，性能差

```
class Document{
public:
    virtual void process()=0;
    virtual ~Document()=default;
};


void invoke(){

    unique_ptr<Document>
        pDoc=make_unique<Markdown>();

    pDoc->process();
}
```

运行时绑定，性能较差：

- 虚表指针进行二次间接运算
- 虚函数无法内联（inline）
- 缓存不友好
- 不利于代码分析和优化器

概念—— 编译时函数调用

- 接口函数调用时采用编译时绑定，性能好

```
template <typename T>
concept Document = std::semiregular<T> &&
requires(T t)
{
    t.process() ;
};


template<Document T>
void invoke(){
    T doc;
    doc.process();
}

invoke<Markdown>();
```

编译时绑定，性能较好：

- 无间接运算，直接绑定
- 函数调用容易内联（inline）
- 缓存友好
- 有利于代码分析和优化器

性能比较 —— 概念 VS. 抽象类

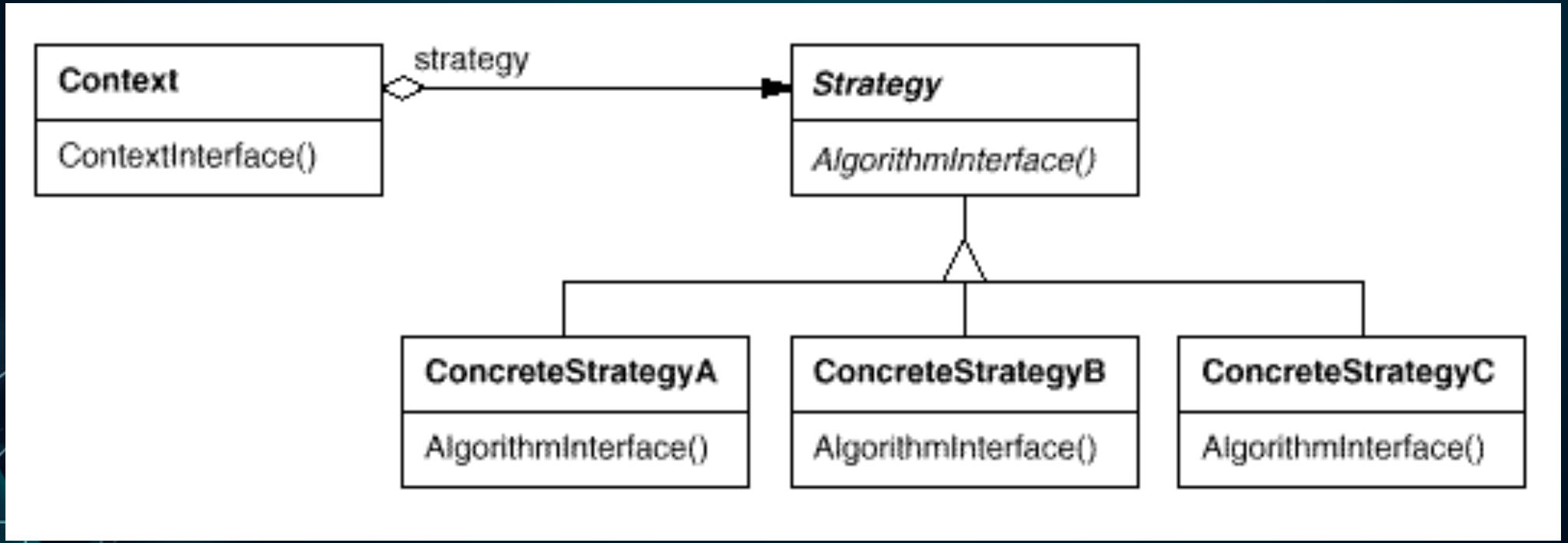
	抽象类	概念
值/引用语义	对象实例采用引用语义，增加间接层，强制对象堆分配，容易共享引用，性能差	对象实例可以采用值语义，无需间接层，对象栈分配，性能好
内存模型	实现类（子类）内存布局必须 “包含” 抽象类（基类）内存布局	概念不定义任何内存布局，对实现类无任何影响
多接口重复	菱形继承带来的重复性内存负担	菱形实现的重复性自然被消除
虚函数表	抽象类增加了虚函数和RTTI（运行时类型信息）存储成本	无需支持运行时辨析，不会增加虚表结构和RTTI存储成本
接口函数调用	运行时绑定：接口函数（虚函数）调用时采用运行时绑定，间接调用，性能差	编译时绑定：接口函数调用时采用编译时绑定，无间接运算，性能好
系统优化支持	接口函数调用无法内联（inline），缓存不友好，不利于代码分析和优化器	接口函数调用容易内联（inline），缓存友好，有利于代码分析和优化器
总结	有一定开销的抽象机制	零开销抽象机制

设计比较 —— 概念 VS. 抽象类

	抽象类	概念
多态机制	运行期多态	编译期多态
类型实现方式	显式实现，必须通过继承，具有侵入性	隐式实现，接口满足即可、无侵入性
操作表达	仅支持虚函数表达操作	支持类型成员、静态成员、操作符等各种
基础特性	不支持	支持构造、拷贝、移动、相等判断等基础特性
关系运算	只能通过继承表达特化关系	支持与（&&）、或（||）逻辑运算
总结	继承和虚函数为接口表达带来很多限制	接口设计具有丰富的粒度和自由，回归本来的含义

使用概念重塑模式案例分析（一）——策略模式

定义一系列算法，把它们一个个封装起来，并且使它们可互相替换。该模式使得算法可独立于使用它的客户程序而变化。



面向对象 Strategy

```
class ListStrategy{
public:
    virtual void list_item(const string& item)=0;
    virtual void start()=0;
    virtual void end() =0;
    virtual ~ListStrategy() = default;
};

class MarkdownList : public ListStrategy{
public:
    void start() override { }
    void end() override{ }
    void list_item(const string& item) override{ }
};

class HtmlList : public ListStrategy{
public:
    void start() override { }
    void end() override{ }
    void list_item(const string& item) override{ }
};
```

```
class TextProcessor
{
public:
    void append_list(
        const vector<string>& items)
    {
        list_strategy->start();
        for (auto& item : items)
            list_strategy->list_item(item);
        list_strategy->end();
    }
    ......
private:
    unique_ptr<ListStrategy> list_strategy;
};
```

基于概念的Strategy

```
template <typename T>
concept ListStrategy =
requires(T t, const string& item) {
    t.list_item(item);
    t.start();
    t.end();
};

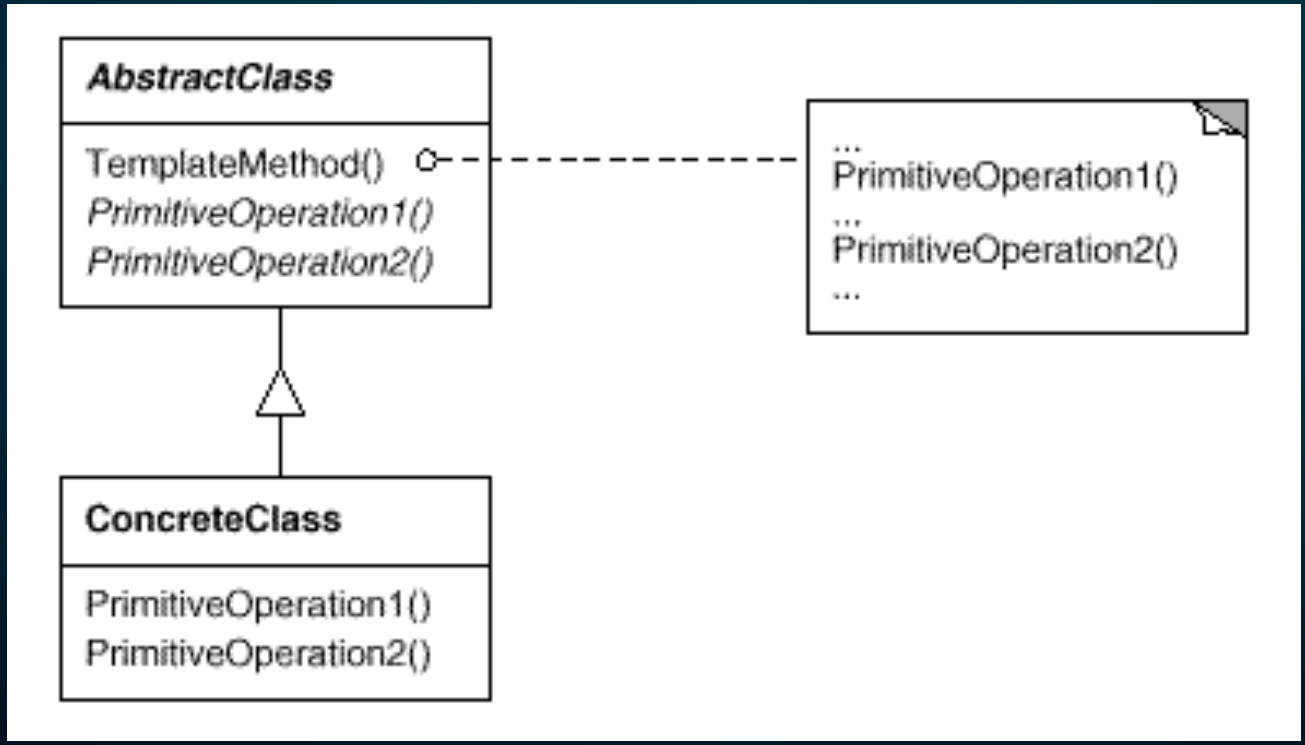
class MarkdownList {
public:
    void start() { }
    void end() { }
    void list_item(const string& item){ }
};

class HtmlList {
public:
    void start() { }
    void end() { }
    void list_item(const string& item){ }
};
```

```
template <ListStrategy T>
class TextProcessor : private T {
public:
    void append_list(
      const vector<string>& items)
    {
        T::start();
        for (auto& item : items)
            T::list_item(item);
        T::end();
    }
    ......
};
```

使用概念重塑模式案例分析（二）——Template Method模式

定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。



面向对象Template Method

```
class Library
{
public:
    void process()
    {
        step1();
        while (!step2())
            step3();
        step4();
        step5();
    }
    virtual ~Library(){}
protected:
    void step1() { ... }
    void step3() { ... }
    virtual bool step2() = 0;
    virtual int step4() = 0;
    virtual void step5()=0;
};
```

```
class App : public Library
{
protected:
    bool step2() override
    { ... }

    int step4() override
    { ... }

    void step5() override
    { ... }
};
```

基于概念的Template Method

```
template <typename T>
concept Component = requires(T t) {
    t.step1();
    {t.step2()} -> std::convertible_to<bool>;
    t.step3();
    {t.step4()} -> std::integral;
    t.step5();
};

class Library {
public:
    static void process(Component auto& component)
    {
        component.step1();
        while (!component.step2())
            component.step3();
        component.step4();
        component.step5();
    }
    void step1() { ... }
    void step3() { ... }
};
```

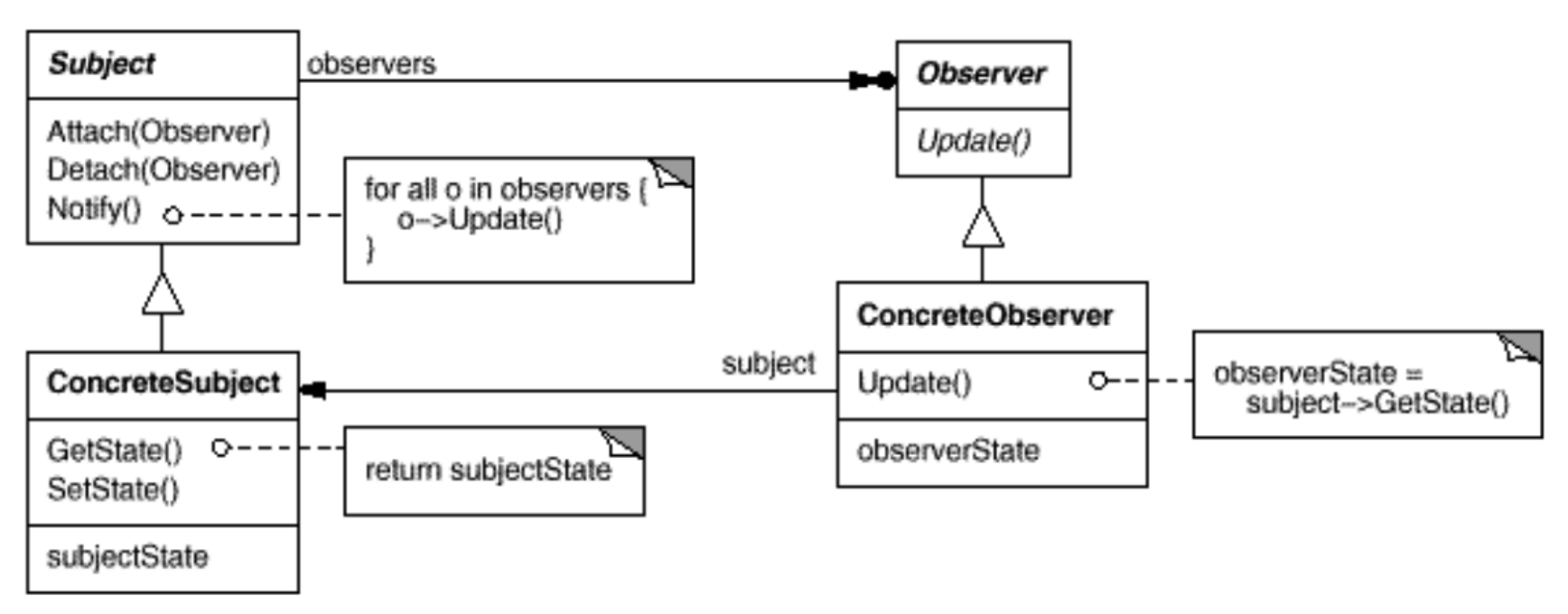
```
class App : public Library
{
public:
    bool step2()
    { ... }

    int step4()
    { ... }

    void step5()
    { ... }
};
```

使用概念重塑模式案例分析（三）——Observer模式

定义对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新。



面向对象Observer

```
class IProgress{
public:
    virtual void update(float value)=0;
    virtual ~IProgress(){}
};

class LogProgress : public IProgress {
public:
    void update(float value) override{
        ...
    }
};
```

```
class Subject
{
private:
    list<shared_ptr<IProgress>> m_progressList;

protected:
    virtual void onUpdate(float value) {
        for (auto& progress : m_progressList){
            progress->update(value);
        }
}

public:
    void addIProgress(const
    shared_ptr<IProgress>& iprogress){
        m_progressList.push_back(iprogress);
    }

    void removeIProgress(const
    shared_ptr<IProgress>& iprogress){
        m_progressList.remove(iprogress);
    }
};
```

基于概念的Observer

```
//标准库定义概念
template< class F, class... Args >
concept invocable =
requires(F&& f, Args&&... args) {
    std::invoke(std::forward<F>(f),
        std::forward<Args>(args)...);
};

struct ProgressObserver{
    void operator()(float value){
        ...
    }
};

using ProgressHandler =
    std::function<void(float)>;

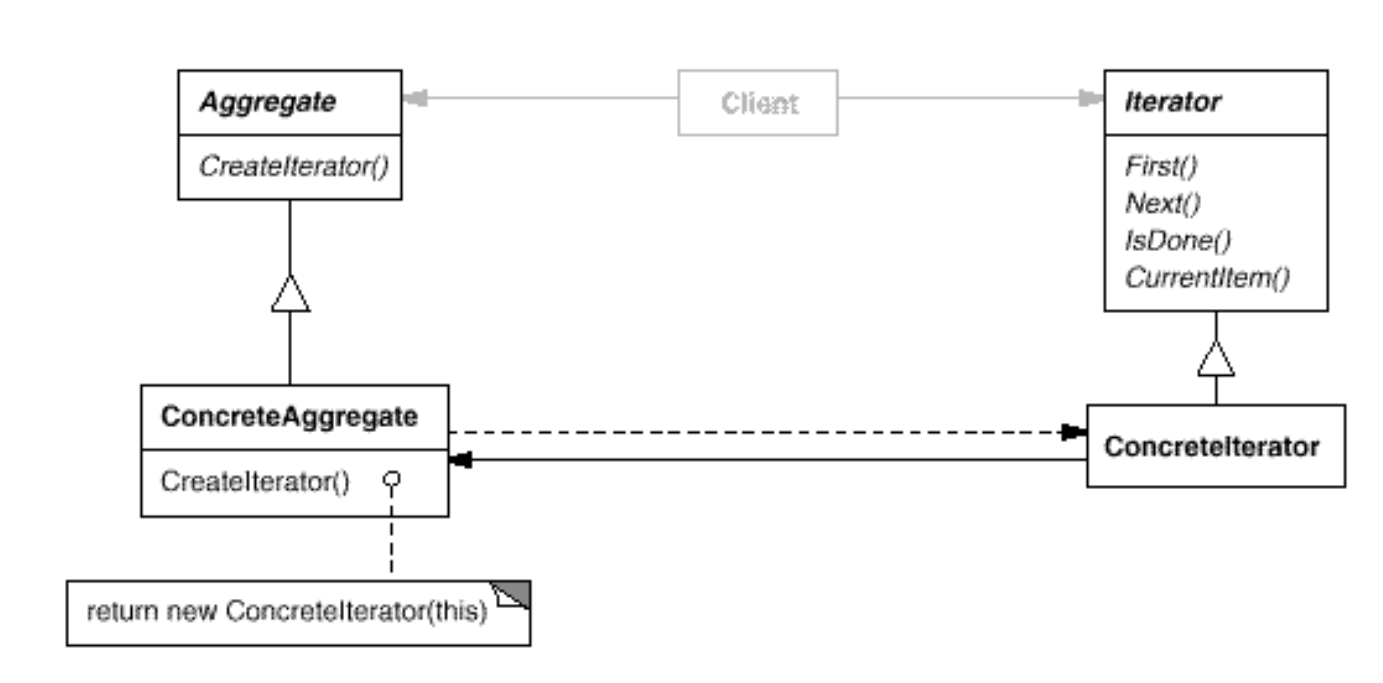
ProgressListener callback=[](float
value){
    ...
};
```

```
template<std::invocable<float> ProgressObserver>
class Subject
{
private:
    list<ProgressObserver> m_progressList;
protected:
    void onUpdate(float value) {
        for (auto& progress : m_iprogressList){
            progress(value);
    }
}

public:
    void addIProgress(
        const ProgressObserver& progress){
        m_progressList.push_back(progress);
    }
    void removeIProgress(
        const ProgressObserver& progress){
        m_progressList.remove(progress);
    }
};
```

使用概念重塑模式案例分析（四）——Iterator模式

提供一种方法顺序访问一个聚合对象中的各个元素，而又不暴露（稳定）该对象的内部表示。



基于概念的Iterator

```
template<class I>
concept forward_iterator =
std::input_iterator<I> &&
std::incrementable<I> &&
std::sentinel_for<I, I>;

template<class I>
concept bidirectional_iterator =
std::forward_iterator<I> &&
requires(I i) {
    { --i } -> std::same_as<I&>;
    { i-- } -> std::same_as<I>;
};
```

```
template<class I>
concept random_access_iterator =
std::bidirectional_iterator<I> &&
std::totally_ordered<I> &&
std::sized_sentinel_for<I, I> &&
requires(I i, const I j,
    const std::iter_difference_t<I> n) {
    { i += n } -> std::same_as<I&>;
    { j + n } -> std::same_as<I>;
    { n + j } -> std::same_as<I>;
    { i -= n } -> std::same_as<I&>;
    { j - n } -> std::same_as<I>;
    { j[n] } ->
    std::same_as<std::iter_reference_t<I>>;
};
```

概念在设计实践中的注意事项

概念与traits、enable_if

概念与类型、类型检查: static_assert

概念、模板与auto变量

语法约束 VS. 语义约束

概念定义的颗粒度

运行时多态存储

```
template <typename T>
concept Drawable = requires(T t) {
    t.draw() ;
};
struct Line {
    void draw() const{
        cout<<"Line draw"<<endl;
    }
};

struct Rect{
    void draw() const{
        cout<<"Rect draw"<<endl;
    }
};

template<Drawable T>
void process(const T& shape){
    shape.draw();
}
```

```
template<class... Ts>
struct overloaded : Ts... {
using Ts::operator()...;
};

template<class... Ts>
overloaded(Ts...) -> overloaded<Ts...>;

using Shape = std::variant<LineRect>;

vector<Shape> vec;
vec.emplace_back(line);
vec.emplace_back(rect);

for_each(vec.begin(), vec.end(), [](auto& elem){
    auto visitor=
       overloaded{[](Drawable auto& d){ d.draw();}};
    std::visit(visitor,elem);
    }
);
```

C++ 20 中的概念库

C++ 20标准库包括大约70个概念，包括基础概念（<concepts>）、迭代器概念（<iterator>)、范围概念（<ranges>）

算数概念：整数、浮点、有无符号....
对象概念：拷贝、移动、缺省初始化、赋值、正则...
类型概念：相同、可转换、继承、构造、赋值、交换...
比较概念：相等、比较、次序、三向比较....
回调概念：一元函数、二元函数、谓词函数、关系函数
迭代器概念：输入、输出、前向、双向、随机、连续
范围概念、视图概念、随机数、仿指针、排序.....

讲座总结

概念具有“零开销抽象”的核心精神，是C++20最重要的特性，将深刻影响C++系统级编程的设计。

各种领域概念正在开源项目和核心类库中快速发展壮大。建设领域概念库将成为C++系统软件设计的重要活动。

概念将大幅度改进泛型编程，将泛型编程带入日常编程活动中。

概念会重塑设计模式，也将催生更多的设计思想和手段。

谢谢，欢迎交流

CPP-Summit 2022



李永顺

中兴通讯5G软件架构总工，资深架构师

中兴通讯5G软件技术总工，公司级资深技术教练。拥有丰富系统架构设计及遗留系统架构改造经验，对大型嵌入式系统使用C++进行领域建模，组合式设计、性能优化有丰富的实战经验。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

整洁DCI架构实践

大规模C++项目软件设计

李永顺
中兴通讯

议程

1 **背景介绍**

通信领域面临的主要问题，面向对象的门派之争：易理解、修改

2 **DDD带来的变化及存在的挑战**

组件化方法论，结构化面向对象；对象协作动态模型，系统状态和行为

3 **DCI架构（宏观+微观）**

宏观DCI，微观DCI

4 **整洁DCI架构**

整洁架构与 DCI架构融合方法；接口、行为、模型分形架构

5 **整洁DCI架构 C++案例**

整洁DCI架构物理设计，宏观DCI，微观DCI 实现模式

01

背景介绍

背景-通信软件挑战

①处理**大量并发**活动
②在**指定时刻**或者时间内执行
③系统部分在**多台计算机**上
④与**硬件**交互密切
⑤软件**规模非常大**
⑥功能**复杂**，特性交叉
⑦**不间断**运行很多年
⑧软件维护**不能停止**系统
⑨苛刻的**质量、可靠性**要求
⑩包括硬件故障和软件错误的**容错**

--Joe Armstrong

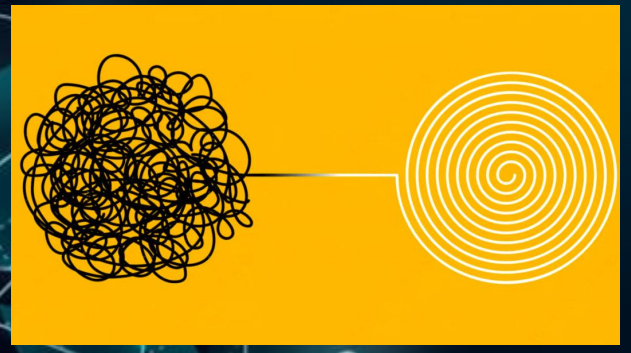


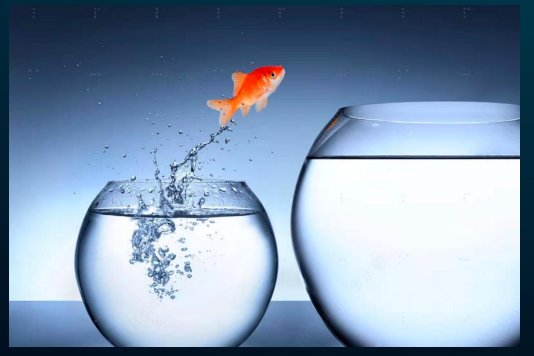
珠峰基站

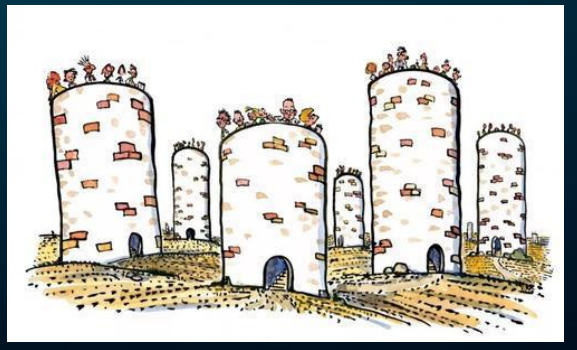
特点：
资源（计算、存储）受限、实时
规模大、复杂度高、高可靠

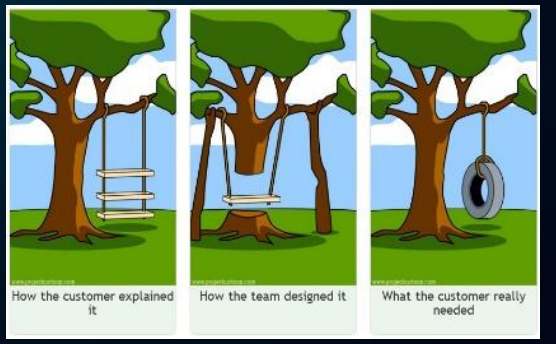
问题提出

- 规模大、场景多，业务复杂度指数级增加，如何**控制复杂度**
- 需求旺盛、硬件多样，如何**响应变化**
- 计算、存储资源受限，如何提升**组件复用性（代码段重复，人力复用）**
- 如何保持业务架构与系统**架构一致性**









设计演进路线

1969	1983	1995	2002	2004	2009	2012
结构化	面向对象	设计模式	设计原则 SOLID	DDD	DCI	整洁架构

一部约束与规范史

面向对象两派之争

- 学院派：强调面向对象对于**现实世界的模拟**
  - 优势： 更容**易理解**
  - 劣势： 引入复杂和耦合，可能产生上帝类，复杂的继承体系
- 工程派：强调**应对变化**
  - 优势： 软件更容**易修改**
  - 劣势： 许多概念是由于设计而创造的概念； 过多的小对象，给对象的生命周期管理与可理解性带来负担

02

DDD带来的变化及存在的挑战

DDD带来的变化

- 统一语言，**降低沟通、交流成本**
- 以领域模型为桥梁，建立**业务架构与系统架构映射**
- 通过限界上下文给出**组件边界划分**方法
- **结构化面向对象**设计
  - 提炼出值对象、实体等核心概念，通过聚合对其封装及组合
  - 通过工厂、仓储管理对象生命周期（CRUD）

DDD面临的挑战

- 领域模型反映了领域对象的结构及关系，缺失了**对象间的协作**
- DDD分层架构未能表达**系统级状态及行为**
- DDD分层架构未能对**高频变化的行为**及**低频变化的领域知识**较好解耦
- 用户心中的协作和交互模型，如何映射到计算机的**内存模型**

03

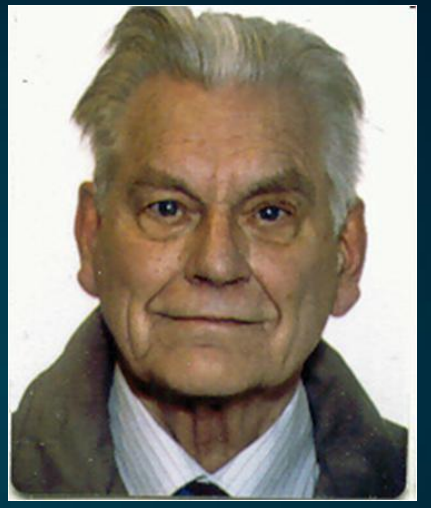
DCI架构

DCI论文

The DCI Architecture: A New Vision of Object-Oriented Programming

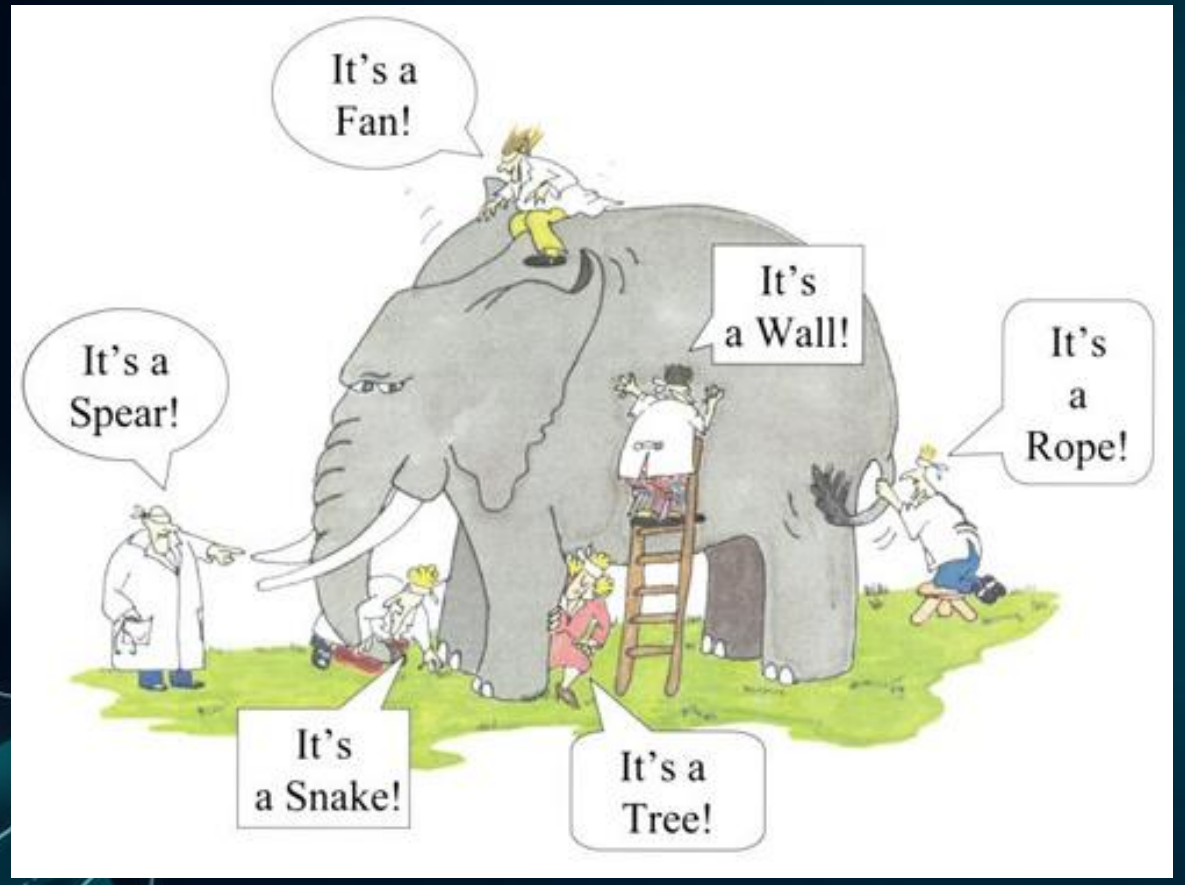
by Trygve Reenskaug(特里夫·雷因斯高) and James O. Coplien

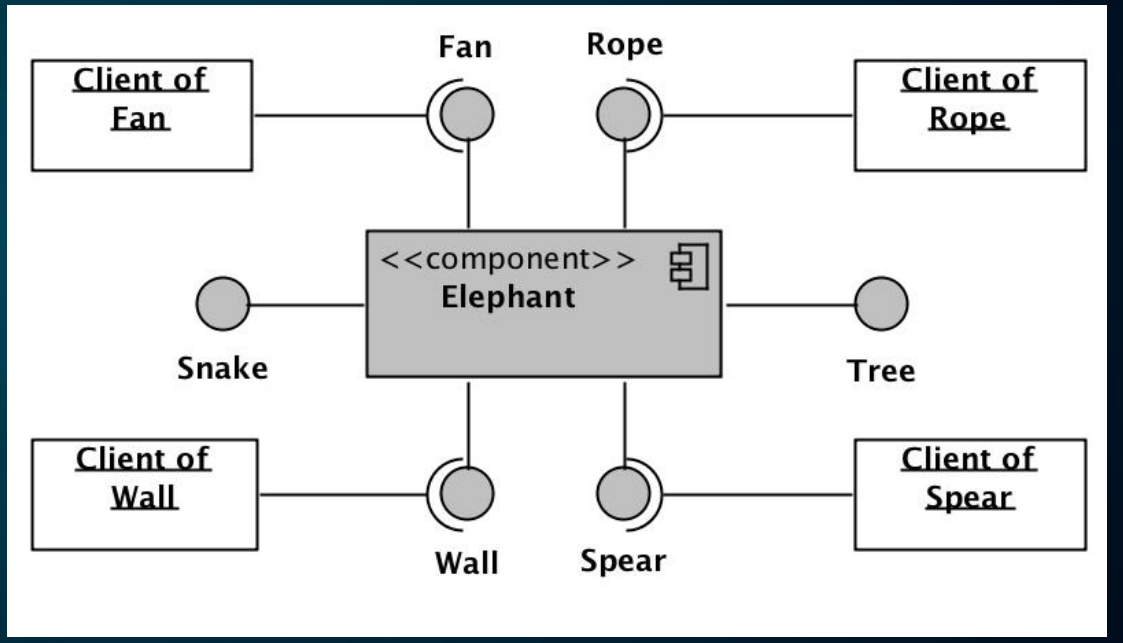
March 20, 2009





多角色对象

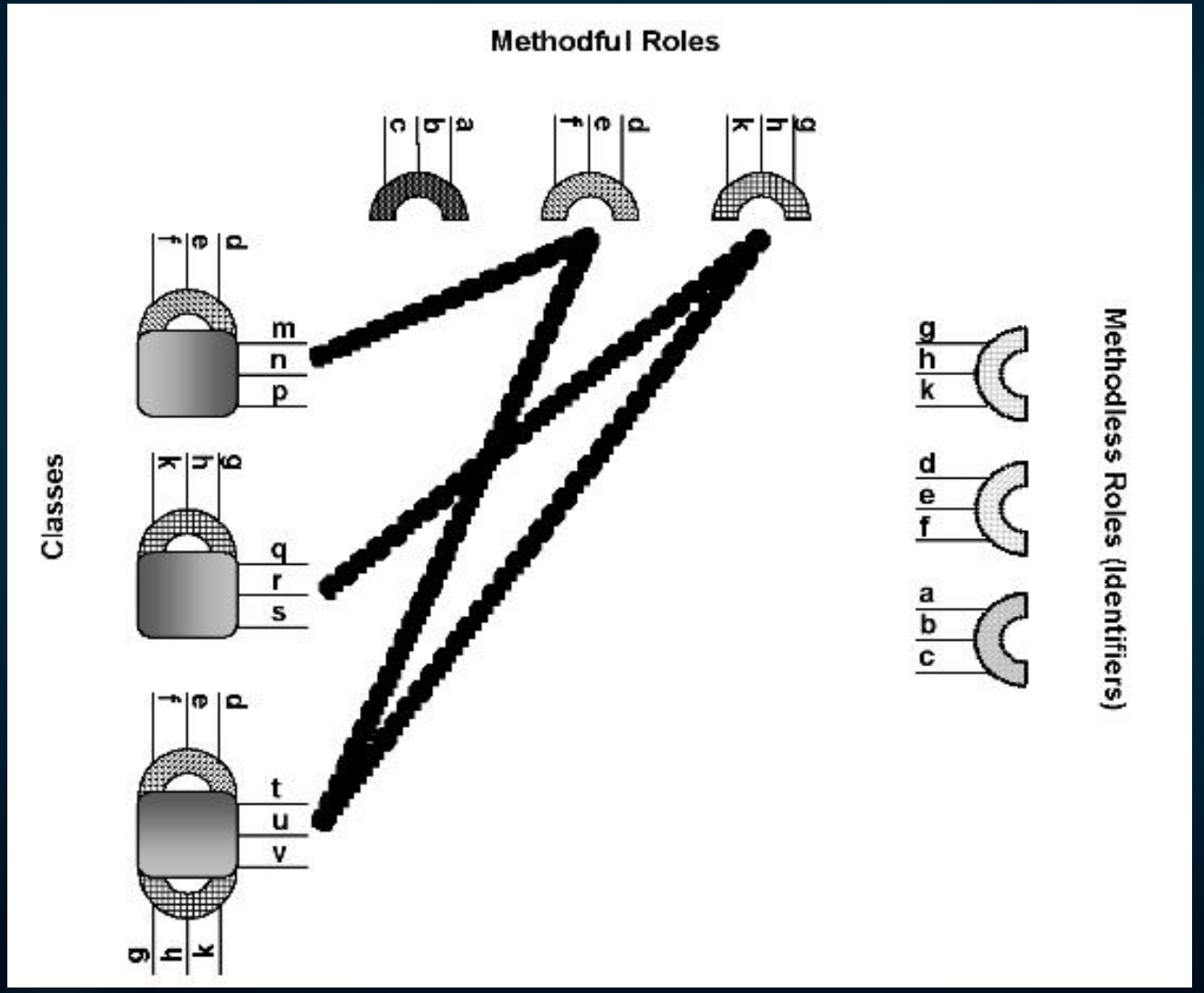

It's a Fan!
It's a Spear!
It's a Wall!
It's a Rope!
It's a Snake!
It's a Tree!


Client of Fan
Fan
Rope
Client of Rope
<<component>>
Elephant
Snake
Tree
Client of Wall
Wall
Spear
Client of Spear

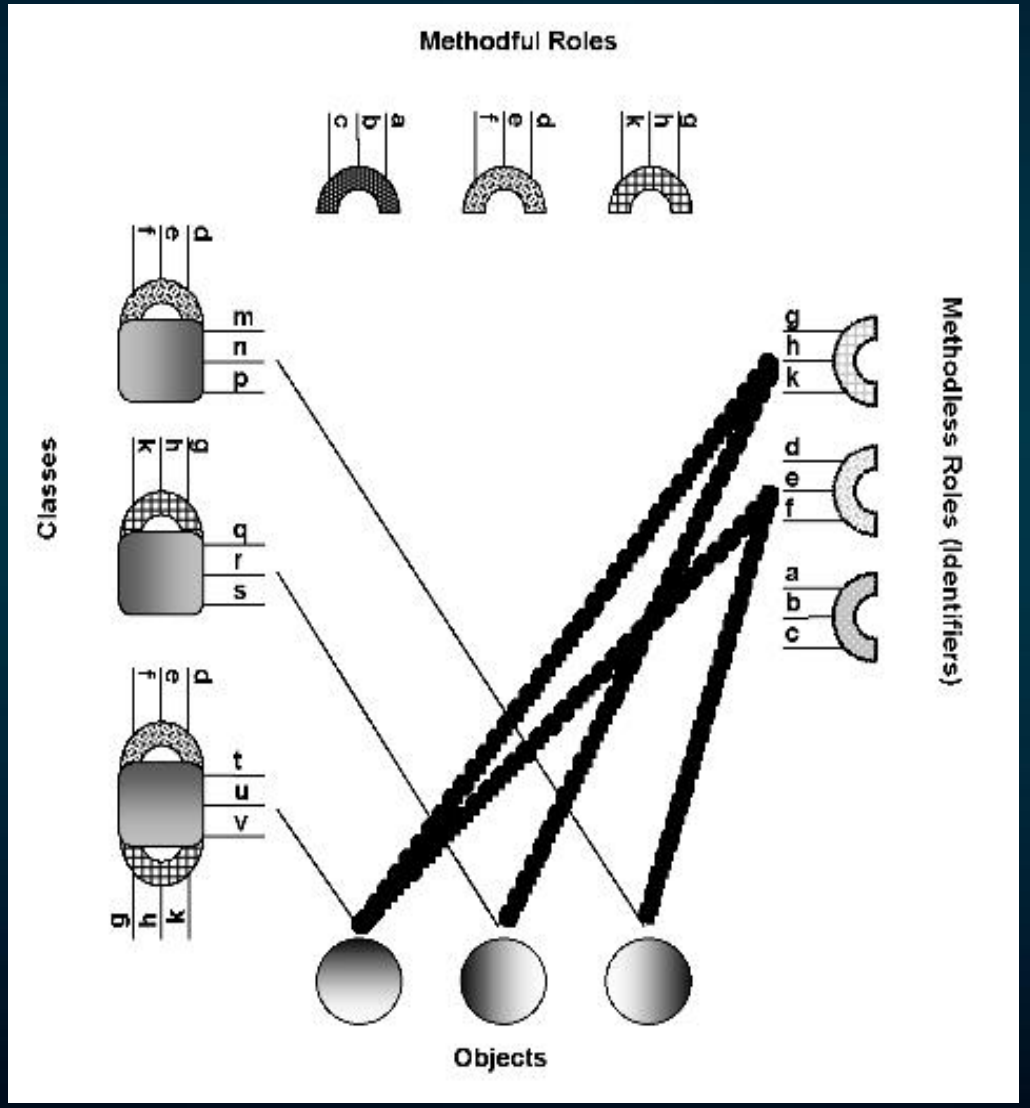
DCI: Data Context Interaction

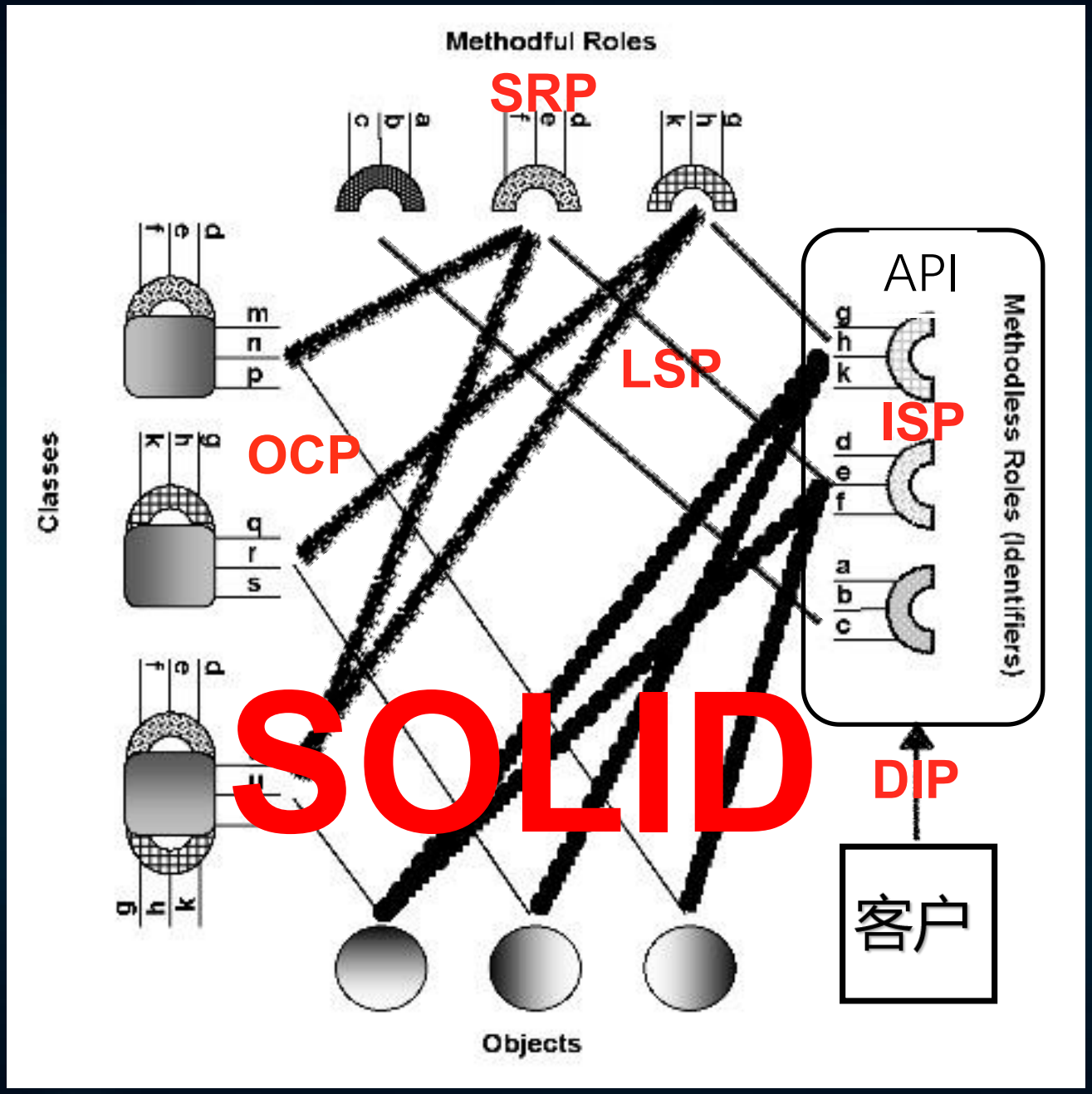
- **DATA**: "what the system *is*."
  - Domain Concepts
  - Relations
  - Constraints
- **INTERACTION**: "what the system *does*."
  - Algorithm
  - Business Logic
- **CONTEXT**: Comprises *use cases* and *algorithms* in which **data** objects are used through specific **roles**.

Classes & Roles : Composite

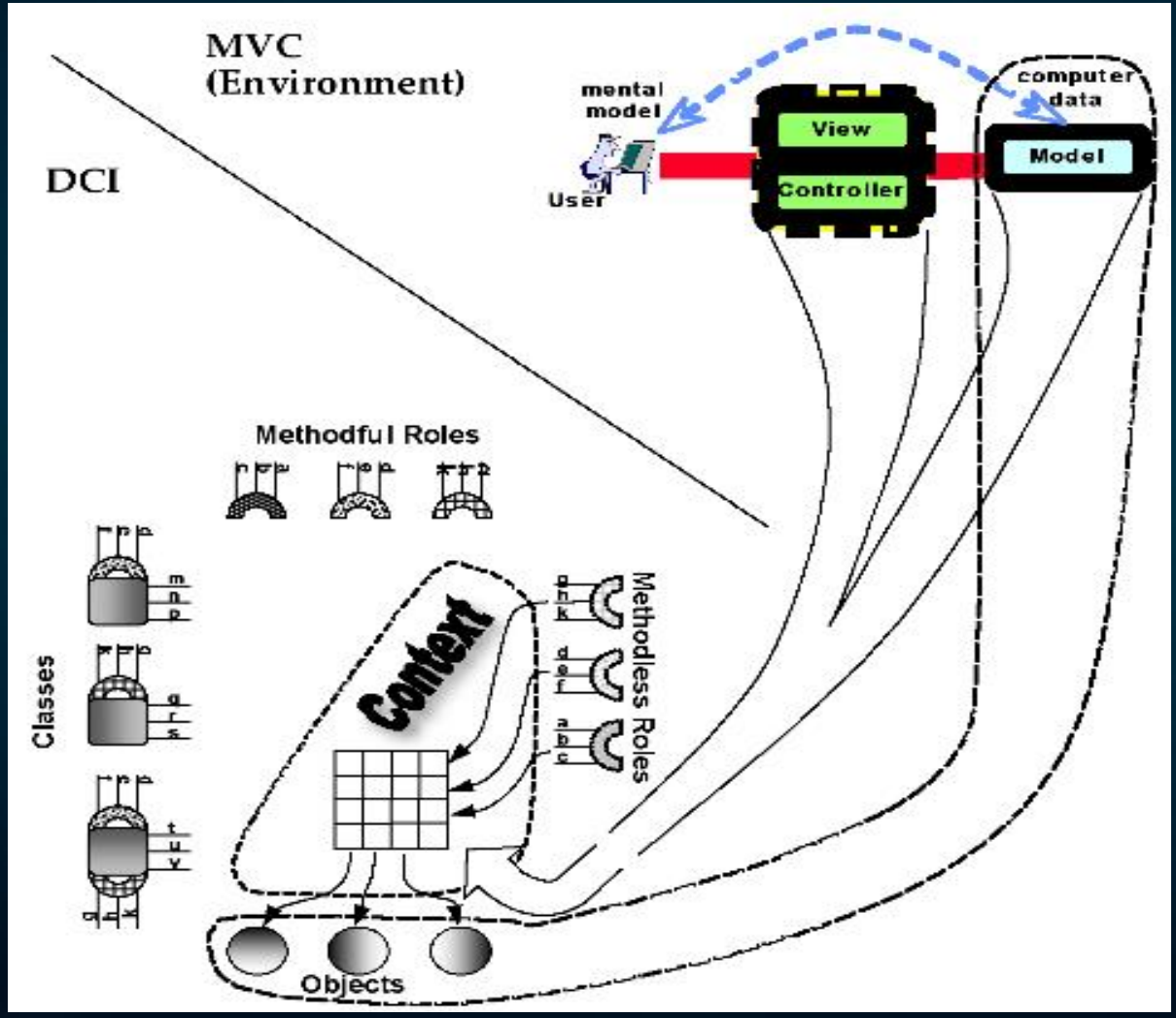


Runtime: Objects, Roles

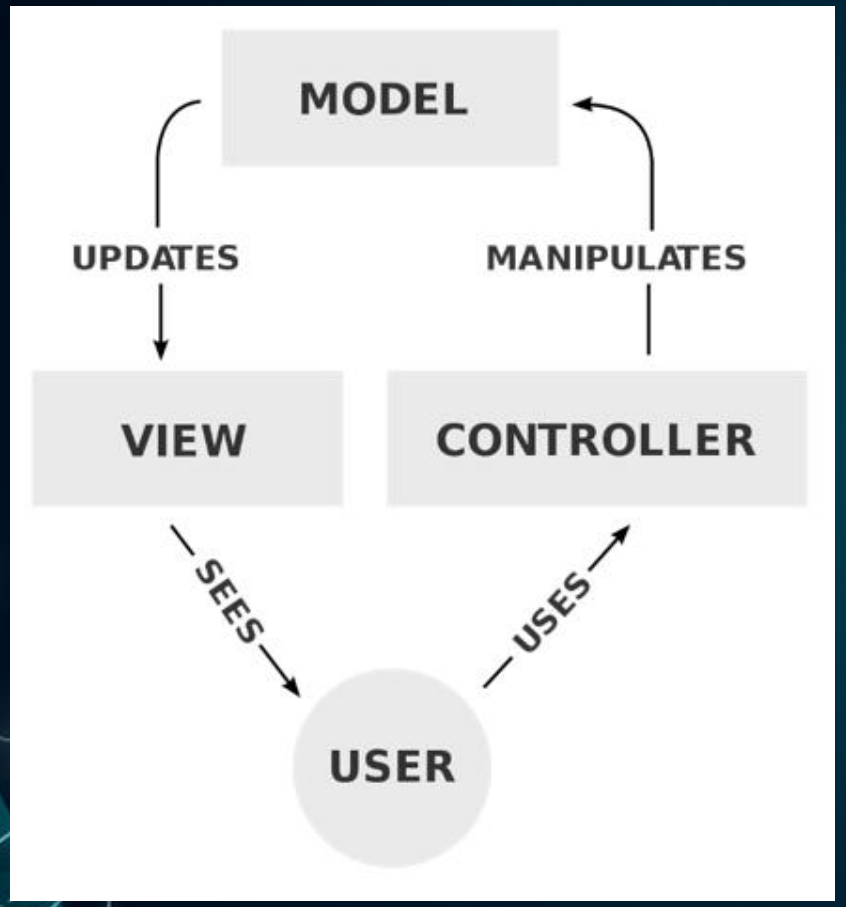



Methodful Roles
SRP
LSP
ISP
OCP
SOLID
DIP
API
Methodless Roles (Identifiers)
Classes
Objects
客户

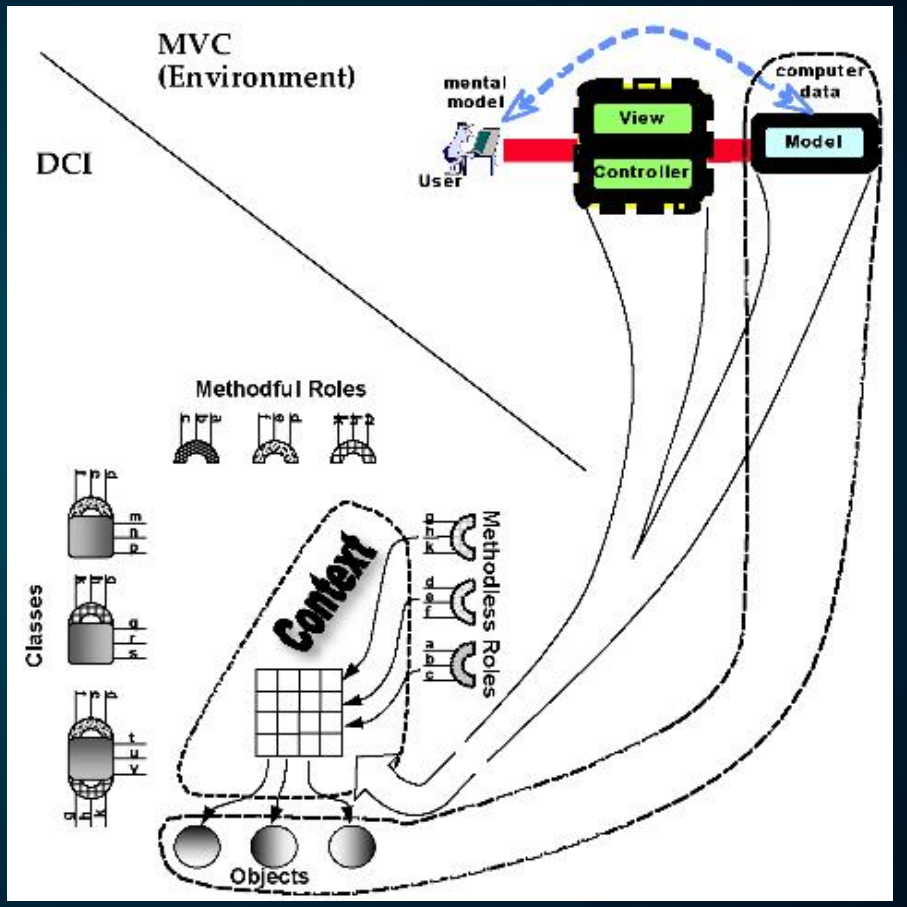
Context : Mapping Roles to Objects



分层架构演进-MVC架构-DCI架构



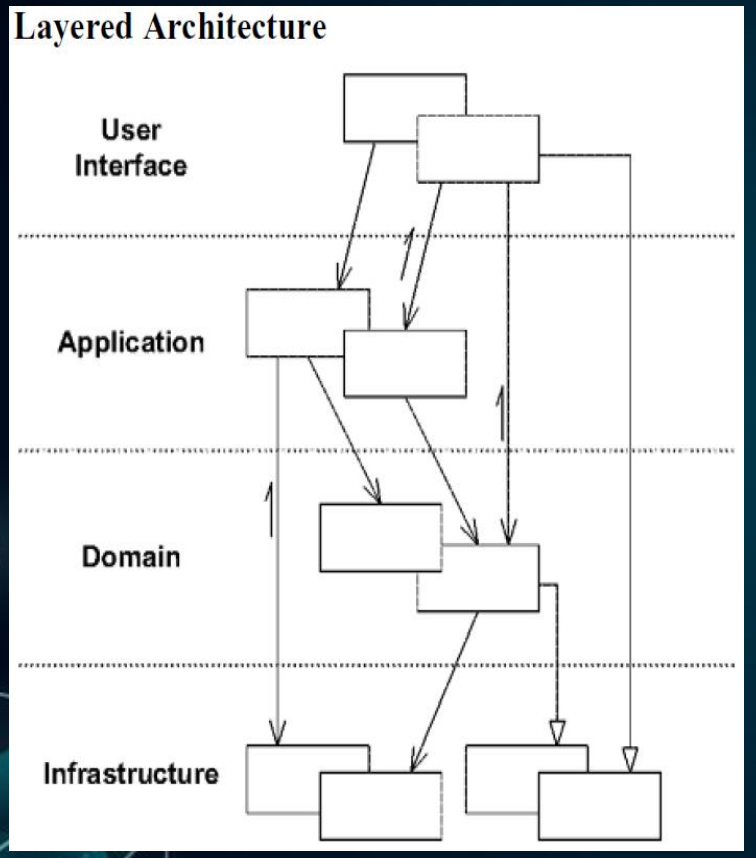
--Trygve Reenskaug 1978



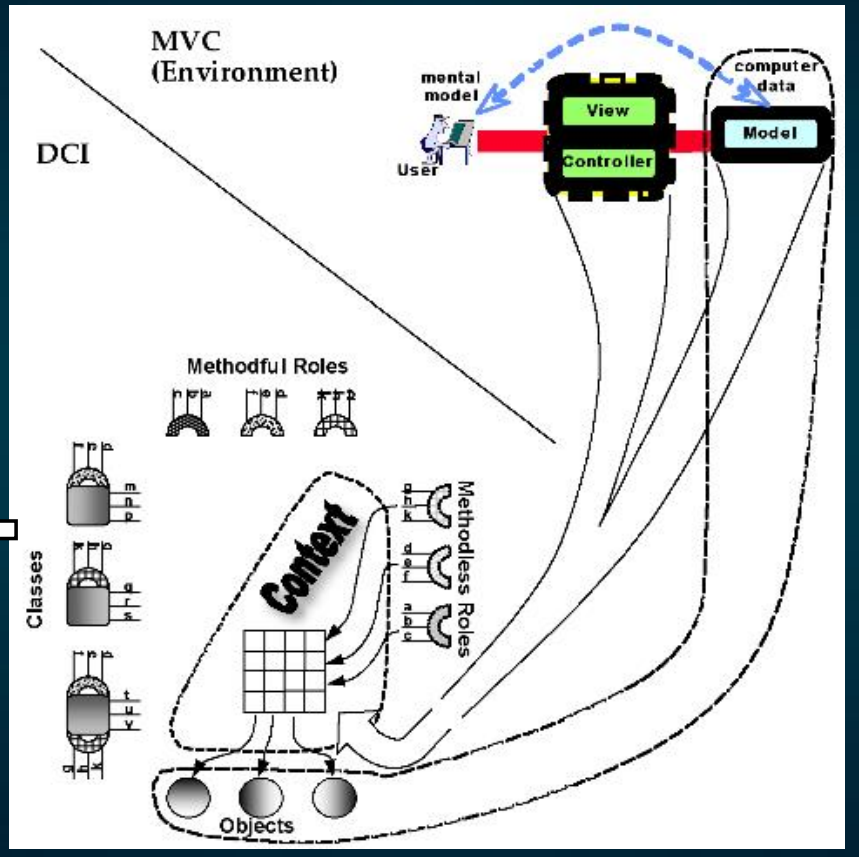
The DCI Architecture

--Trygve Reenskaug.. 2009

DCI（宏观）的分层架构演进



+



=

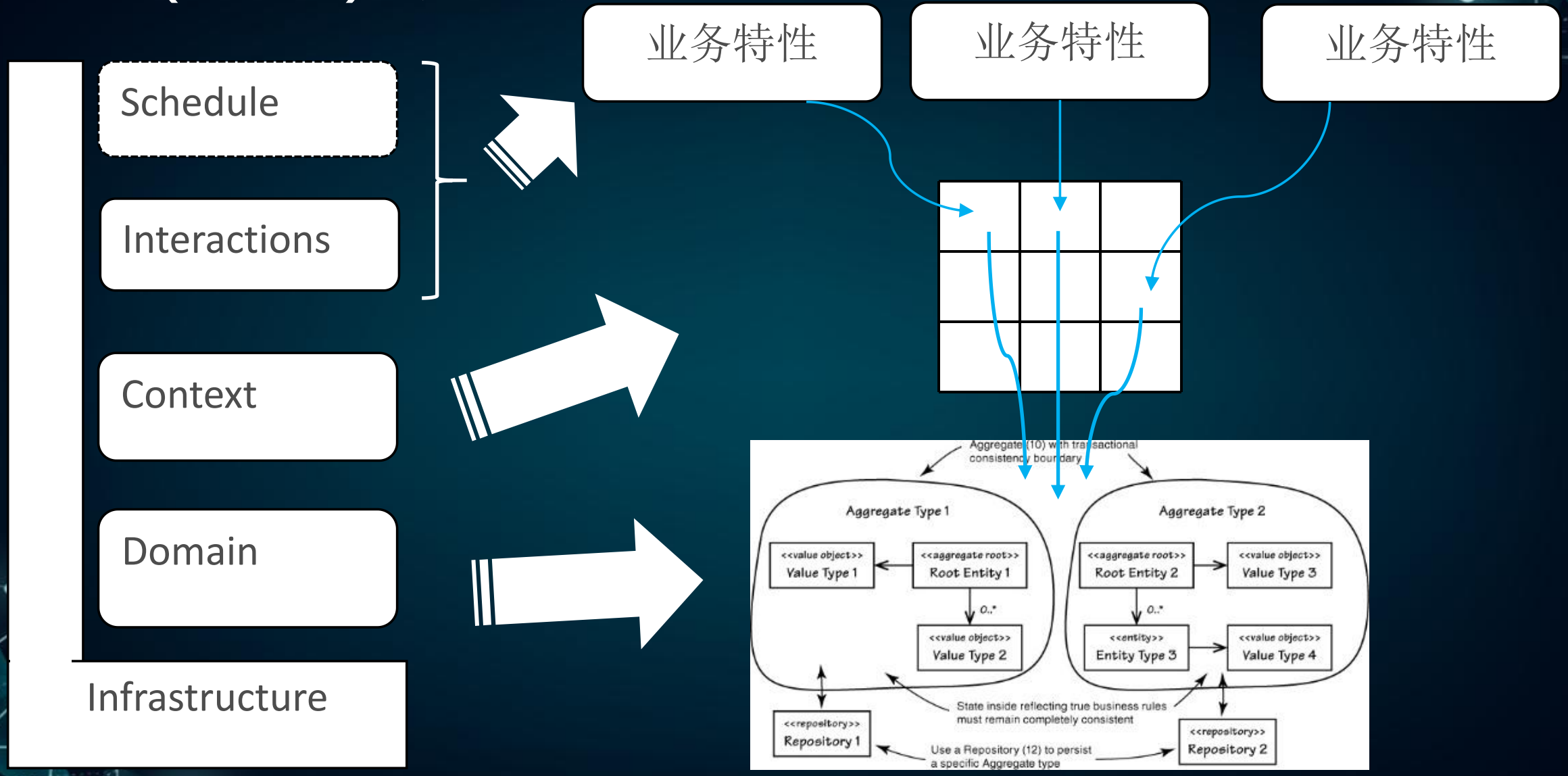
分层架构

- Schedule
- Interactions
- Context
- Domain

Infrastructure

- DDD领域模型缺失了**对象间的协作**
- DDD分层架构未能表达**系统级状态及行为**

DCI（宏观）分层架构


Schedule
Interactions
Context
Domain
Infrastructure
业务特性
业务特性
业务特性
Aggregate (10) with transactional consistency boundary
Aggregate Type 1
<<value object>>
Value Type 1
<<aggregate root>>
Root Entity 1
0..*
<<value object>>
Value Type 2
Aggregate Type 2
<<aggregate root>>
Root Entity 2
<<value object>>
Value Type 3
0..*
<<entity>>
Entity Type 3
<<value object>>
Value Type 4
State inside reflecting true business rules must remain completely consistent
<<repository>>
Repository 1
Use a Repository (12) to persist a specific Aggregate type
<<repository>>
Repository 2

DCI（微观）C++实现

- 多重继承
  - 继承作为一种组合手段
    - 逻辑和数据都作为Role
    - 领域对象组合Role，并注入依赖
  - 没有传统意义多重继承的缺点
- 访问所依赖Role的手段
  - 工厂方法
    - 1对1，多对1
  - 参数
    - 1对多，多对多

收益：

① 简化组合方式
② 降低内存开销
③ 减轻内存管理负担
④ 简化依赖关系管理
⑤ 对象与角色自动转换
⑥ 降低Cache Miss

```
struct Object
    : Role1
    , Role2
    , Role3
    , Role4
{
    // 这些工厂方法即属于工厂职责的实现
    Role2& getRole2() override { return *this; }
    Role3& getRole3() override { return *this; }
    Role4& getRole4() override { return *this; }
};
```

小类，大对象：C++ https://www.jianshu.com/p/6a8d72b3bf48

DCI（微观） In C++

```
DEFINE_ROLE(Spear)
{
    virtual void stab() const;
};

DEFINE_ROLE(Rope)
{
    virtual void tie() const;
};

DEFINE_ROLE(Wall)
{
    virtual void block() const;
};
```

```
DEFINE_ROLE(Hunter)
{
    virtual ~Hunter() = default;
    virtual void hunt() const
    {
        ROLE(Wall).block();
        ROLE(Rope).tie();
        ROLE(Spear).stab();
    }
private:
    USE_ROLE(Wall);
    USE_ROLE(Rope);
    USE_ROLE(Spear);
};
```

```
struct Elephant : Rope
                , Spear
                , Wall
                , Hunter
{
private:
    IMPL_ROLE(Rope);
    IMPL_ROLE(Spear);
    IMPL_ROLE(Wall);
};
```

```
#define ROLE(role) get##role()
#define ROLE_PROTO_TYPE(role) role& ROLE(role) const
#define USE_ROLE(role) virtual ROLE_PROTO_TYPE(role) = 0
#define HAS_ROLE(role) USE_ROLE(role)
#define IMPL_ROLE(role)                                  \
ROLE_PROTO_TYPE(role)                                    \
{                                                        \
    return const_cast<role&>(static_cast<const role&>(*this)); \
}
```

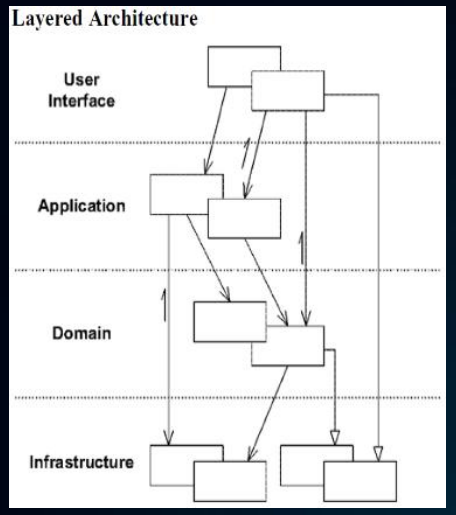
仍旧面临的挑战

- DDD分层架构与DCI架构如何进行**有效结合**，取长补短
- 领域层组件如何**按特性进一步解耦**
- 如何保持**业务架构与系统架构的一致性**
- **存量系统**如何渐进的进行架构演进

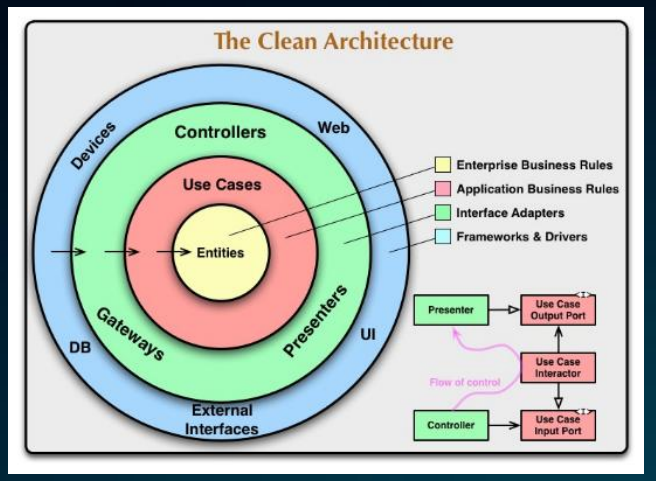
04

整洁DCI架构

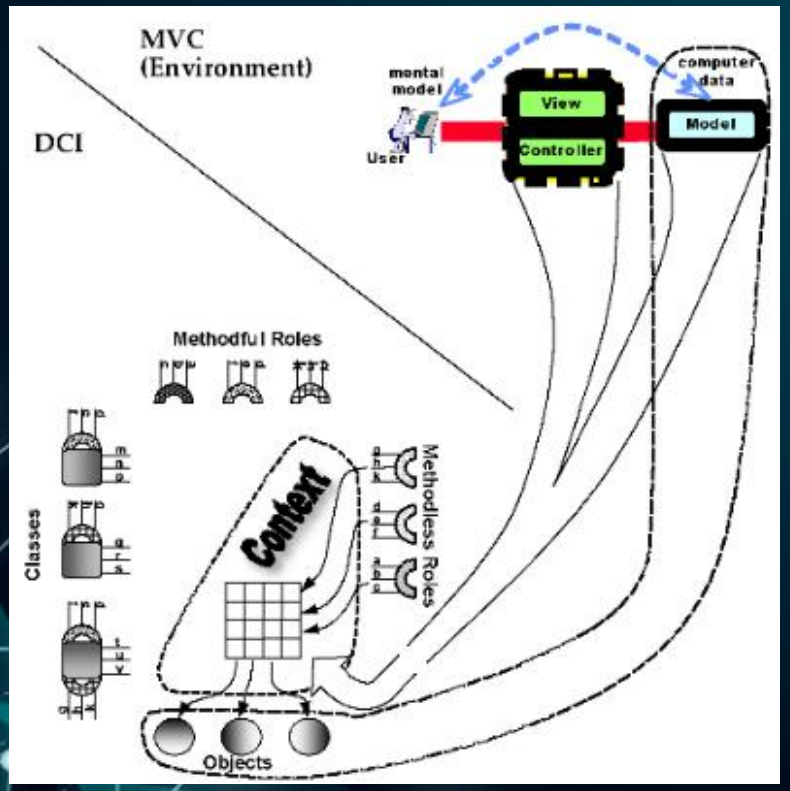
DDD架构演进：整洁DCI架构



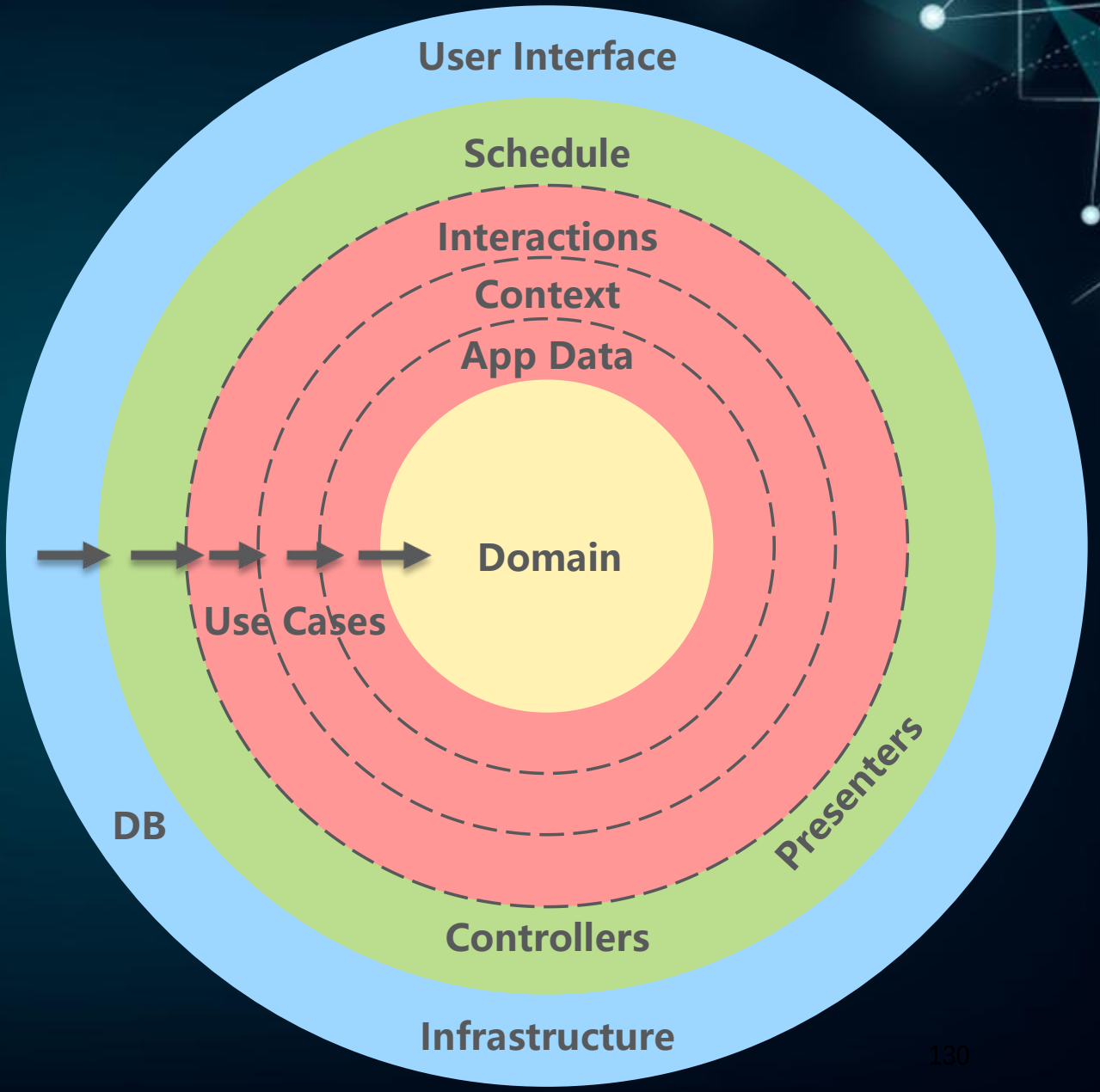
2003



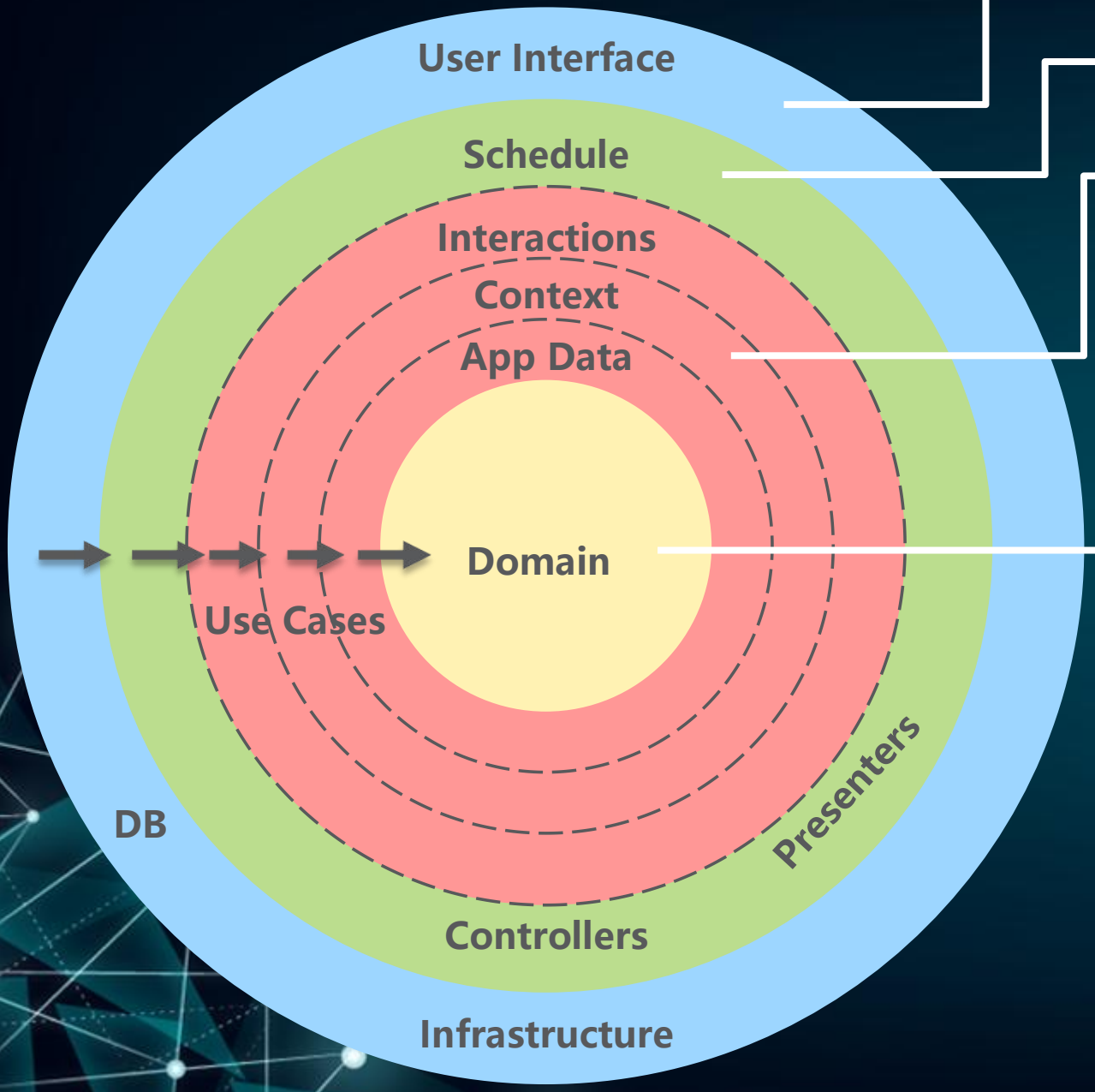
2012



2009



DDD架构演进：整洁DCI架构

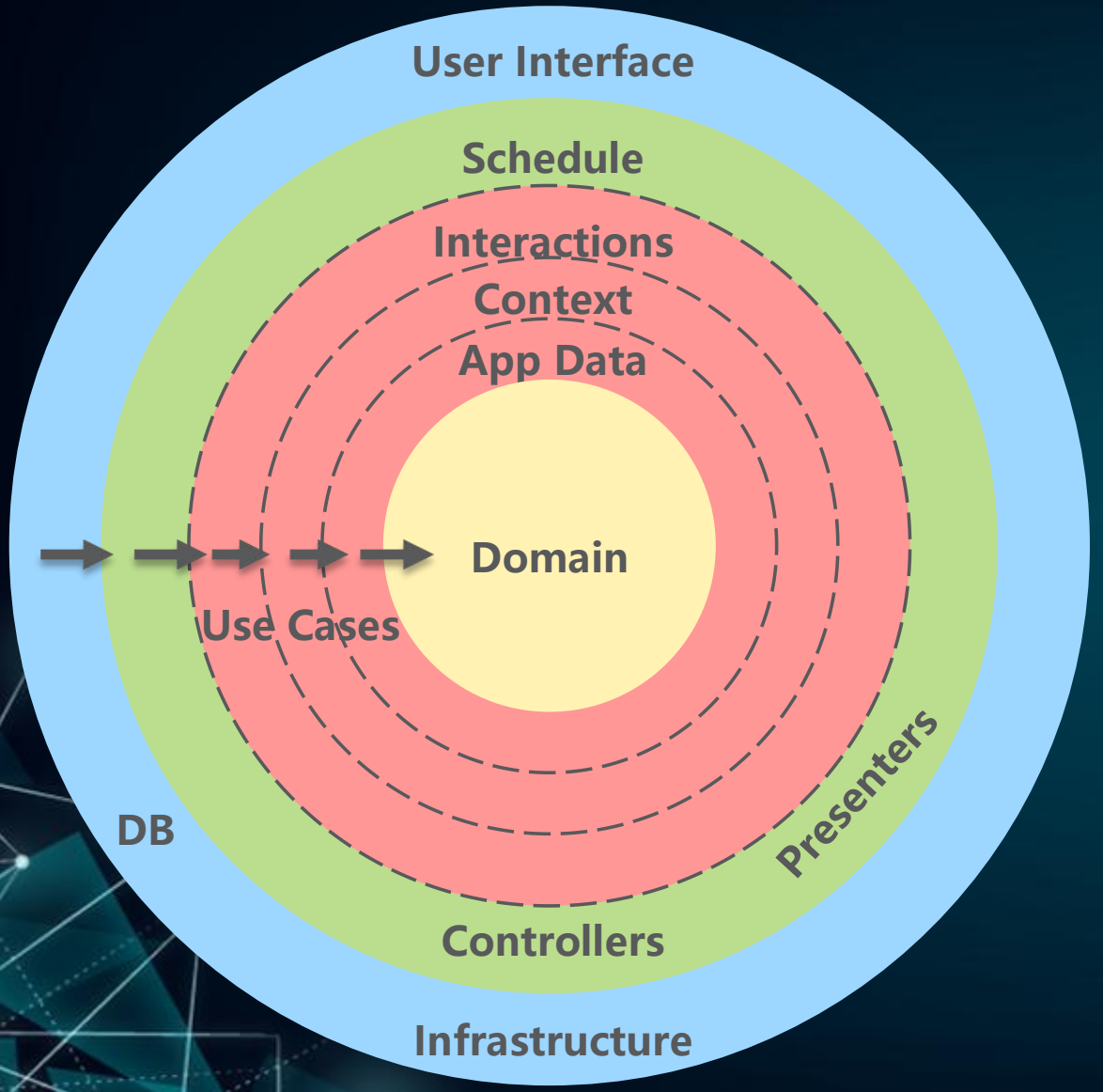


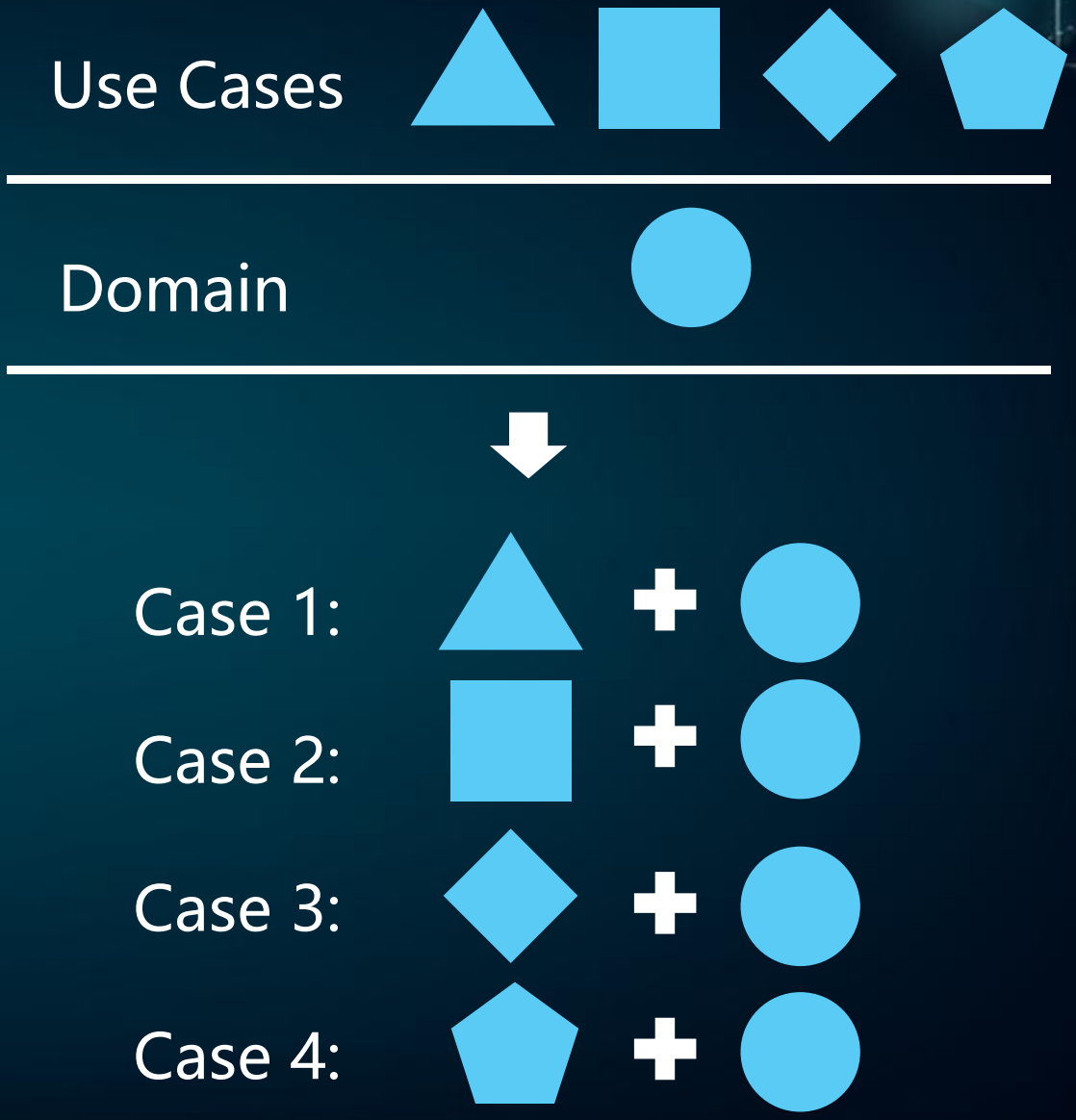
- L4 基础设施层
- L3 接口适配层
- L2 用例层
  - 交互 — Interactions
  - 上下文 — Context
  - 应用数据（App Data）
- L1 领域层
  - 接口（Interface）
  - 行为（Behavior）
  - 模型（Model）

Data

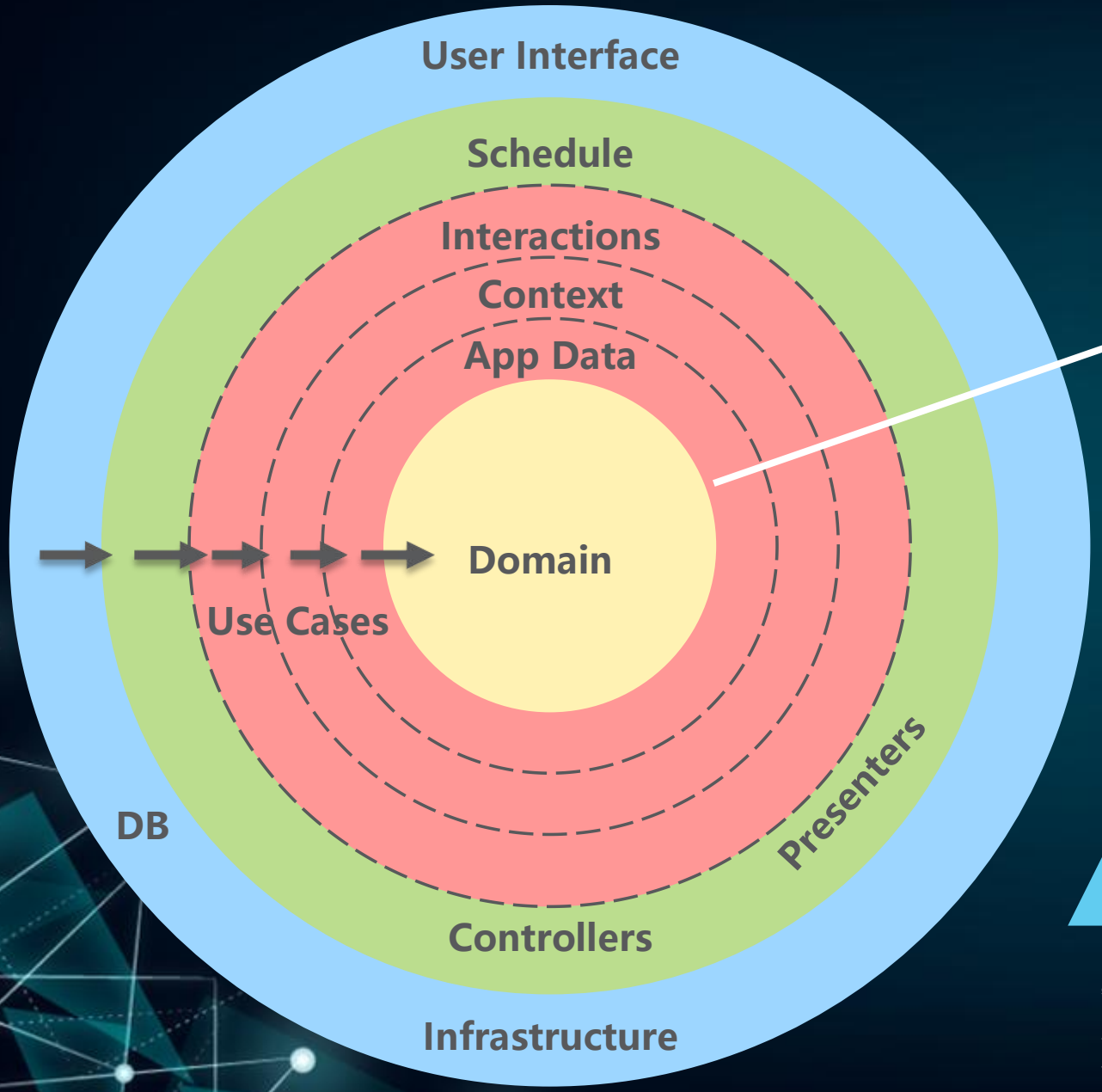
		定义	特征
L4 基础设施层		系统基础设施，包括框架、数据库、驱动程序等	业务无关
L3接口适配层		用例/场景控制、展示，包括用例调度策略、展示形式等	调用用例层
L2 用例层	交互	用例业务逻辑，体现为一系列原子业务能力的交互	用例做什么，事务性，调用上下文
	上下文	用例上下文，完成该用例/场景/算法下领域模型到角色的映射	原子可执行单位，调用应用数据或领域接口
	应用数据	用例数据模型及其关系，即DDD中应用级领域对象，领域服务等	用例是什么，调用领域接口
L1 领域层（分形）	接口	通用业务逻辑中业务能力的封装，体现角色或抽象接口	不是UI，抽象的，稳定的，调用行为
	行为	通用业务逻辑的业务能力，一般表达为角色	无状态，功能内聚，调用模型或其他角色
	模型	通用数据模型及其关系，即DDD中领域模型	有状态，不可调用其他接口或行为

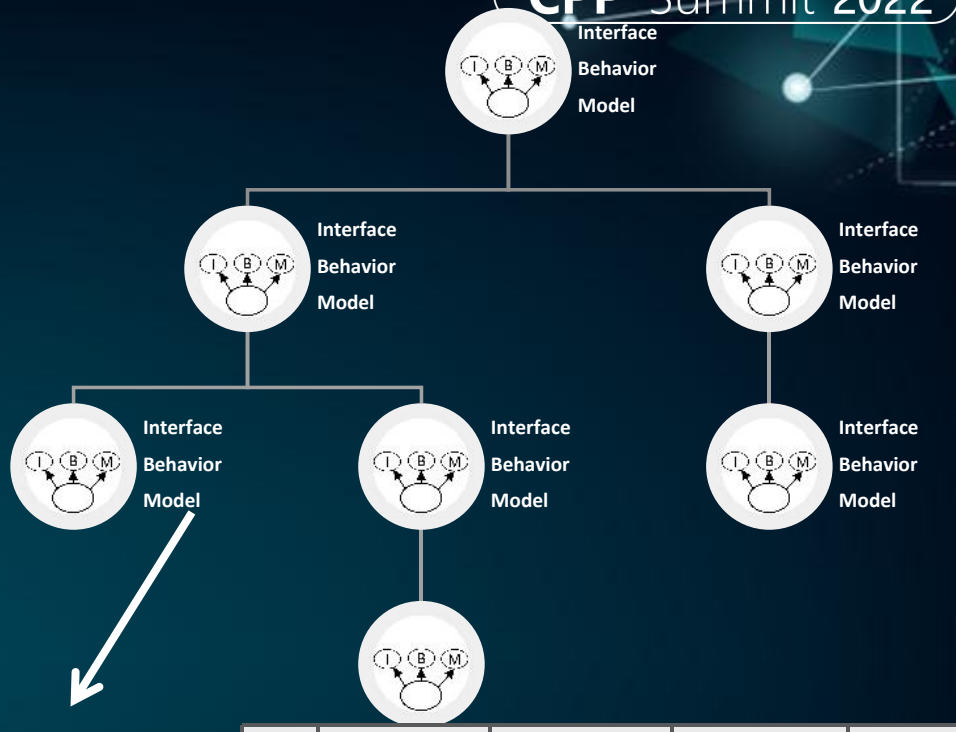
DDD架构演进：整洁DCI架构

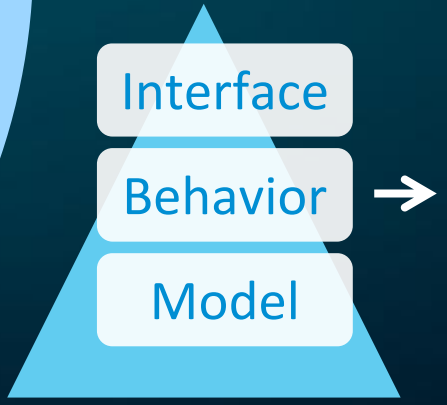




DDD架构演进：整洁DCI架构-分形







	聚合根	聚合	实体/值对象	服务
接口	角色子集	角色子集	方法子集	方法子集
行为	角色集合	角色集合	方法	方法
模型	聚合	聚合/实体/值对象	数据	NA

基础设施层定义接口(API)，封装实现细节，对应的技术组件通过接口(API)供其他层依赖

应用数据并非贫血的数据，指应用级领域模型，可以按照分形范式的接口、行为、模型进行拆分

领域层不依赖其它层，可独立构建

业务架构与系统架构一致性

方案

需求方案 → **组件方案**

特性 波及→ **组件**

组件 拆分→ **模块**

模块 拆分→ **特性**

组件化 **中台化**

DDD **RUP4+1**

目标：

1. 组件拆分成可独立构建的模块
2. 模块编译成动态库形态
3. 可以按特性编排模块
4. 可以按场景编排特性

	C1			C2	
	M1	M2	M3	M4	M5
F1	√		√		
F2	√	√	√		√
F3	√			√	
F4	√	√	√	√	√

F：特性，C：组件，M：模块

程序

构建工程 **Modern CMake** 约束→ **代码结构**

组件 拆分/依赖→ **模块**

组件/模块 编译→ **动态/静态库**

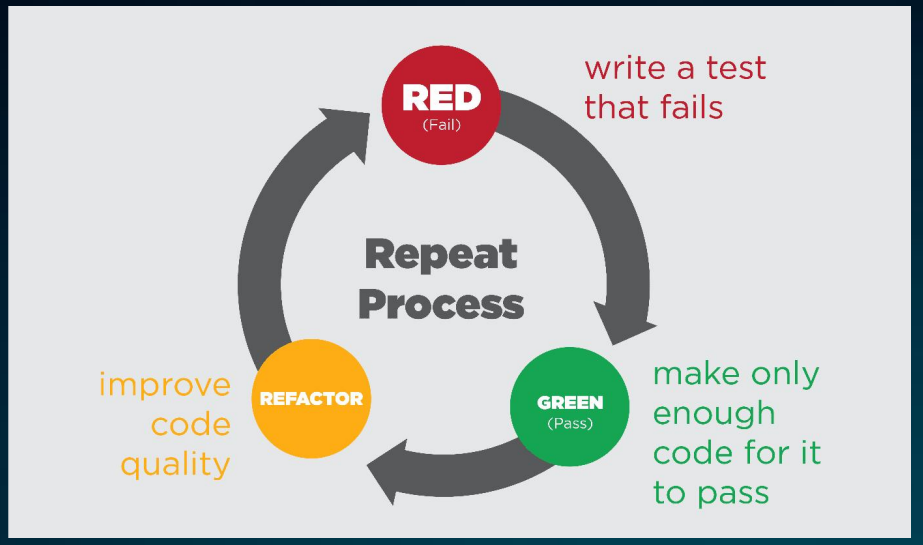
组件/模块 关联→ **特性**

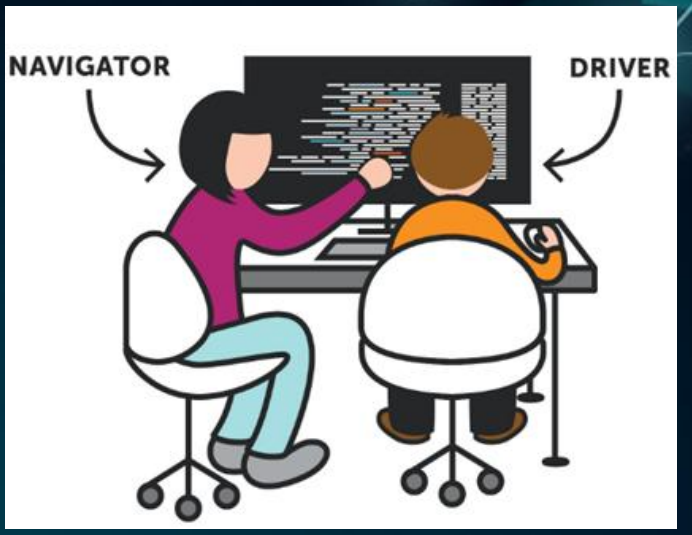
需求	Modern CMake功能支撑
组件、模块分层结构控制	add_subdirectory
组件、模块间依赖关系管理	target_include_directories target_link_directories target_link_libraries
模块封装(接口，私有)特性管理	PUBLIC,PRIVATE,INTERFACE
组件、模块的静态库、动态库	target_sources add_library<STATIC/SHARED>
组件、模块与特性、功能关联管理	include feature_config.cmake target_link_libraries

05

整洁DCI架构 C++ 案例

场景一


RED
(Fail)
write a test
that fails
Repeat
Process
GREEN
(Pass)
make only
enough
code for it
to pass
REFACTOR
improve
code
quality

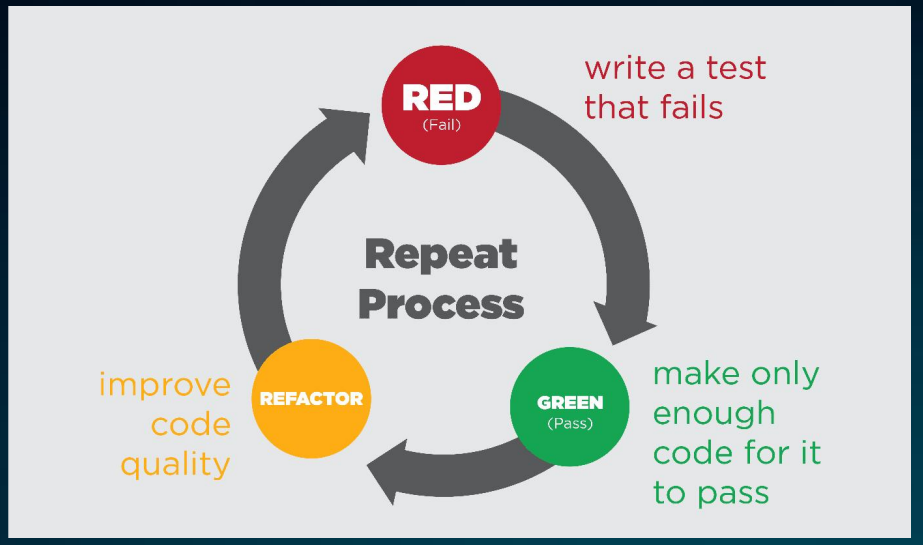

NAVIGATOR
DRIVER

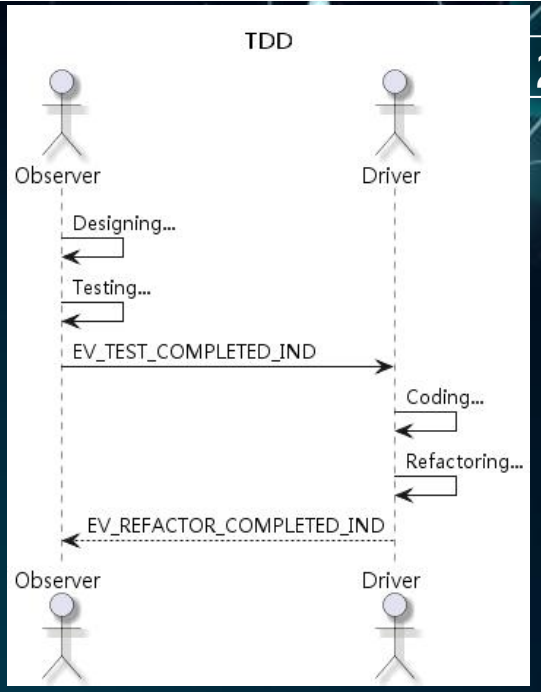
场景二


Time
to go
Home


Turn left
in 1 block.
Ok, that's a
good route.

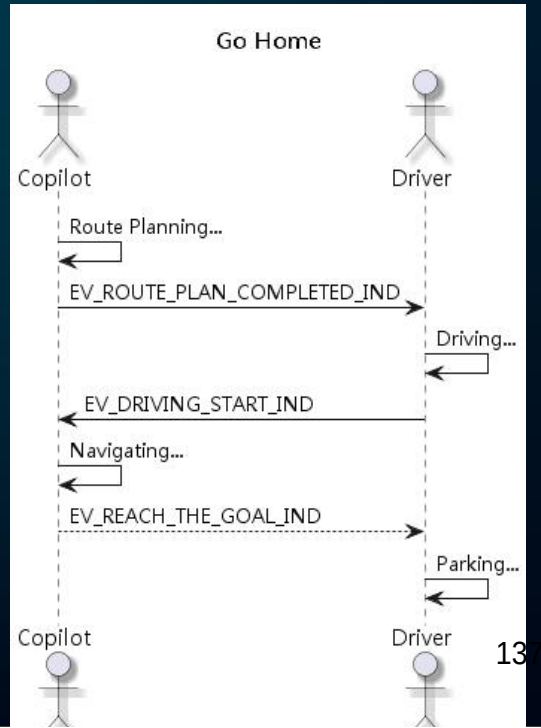
场景一

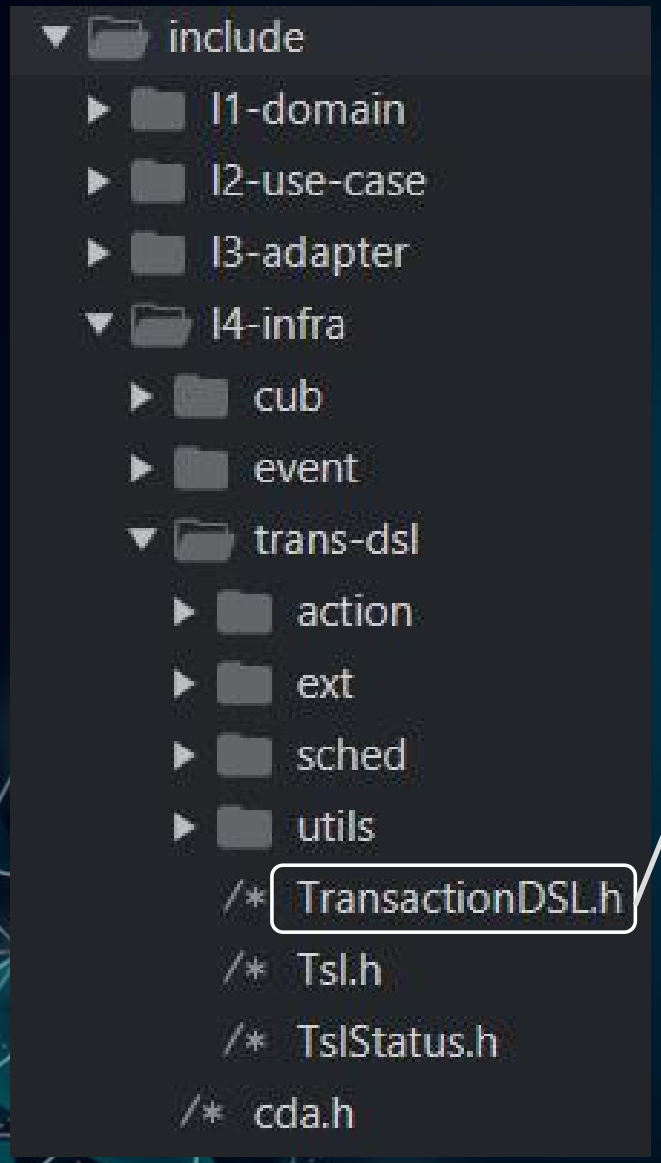

RED
(Fail)
write a test
that fails
Repeat
Process
improve
code
quality
REFACTOR
GREEN
(Pass)
make only
enough
code for it
to pass


TDD
Observer
Driver
Designing...
Testing...
EV_TEST_COMPLETED_IND
Coding...
Refactoring...
EV_REFACTOR_COMPLETED_IND
Observer
Driver

场景二


Time
to go
Home


Go Home
Copilot
Driver
Route Planning...
EV_ROUTE_PLAN_COMPLETED_IND
Driving...
EV_DRIVING_START_IND
Navigating...
EV_REACH_THE_GOAL_IND
Parking...
Copilot
Driver




```
#include <trans-dsl/sched/action/SchedSequentialAction.h>
#include <trans-dsl/sched/helper/LinkedHelper.h>
#include <trans-dsl/sched/helper/VoidHelper.h>

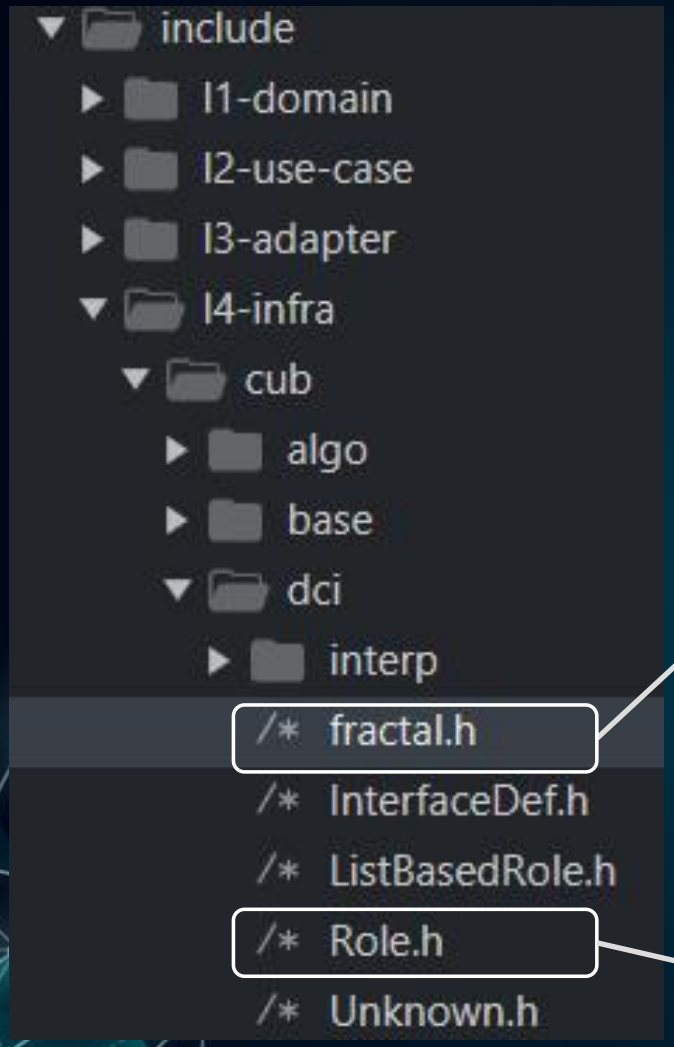
#ifdef __GXX_EXPERIMENTAL_CXX0X__
#include <trans-dsl/sched/helper/11/Sequential11.h>
#else
#include <trans-dsl/sched/helper/98/Sequential98.h>
#endif

////////////////////////////////////////////////////////////////////////

#define __sequential(...)      \
        TSL_NS::details::SEQUENTIAL__< __VA_ARGS__ >

#define __void_sequential(...) \
        TSL_NS::details::VOID_SEQUENTIAL__< __VA_ARGS__ >
```

https://github.com/godsme/**transaction-dsl**

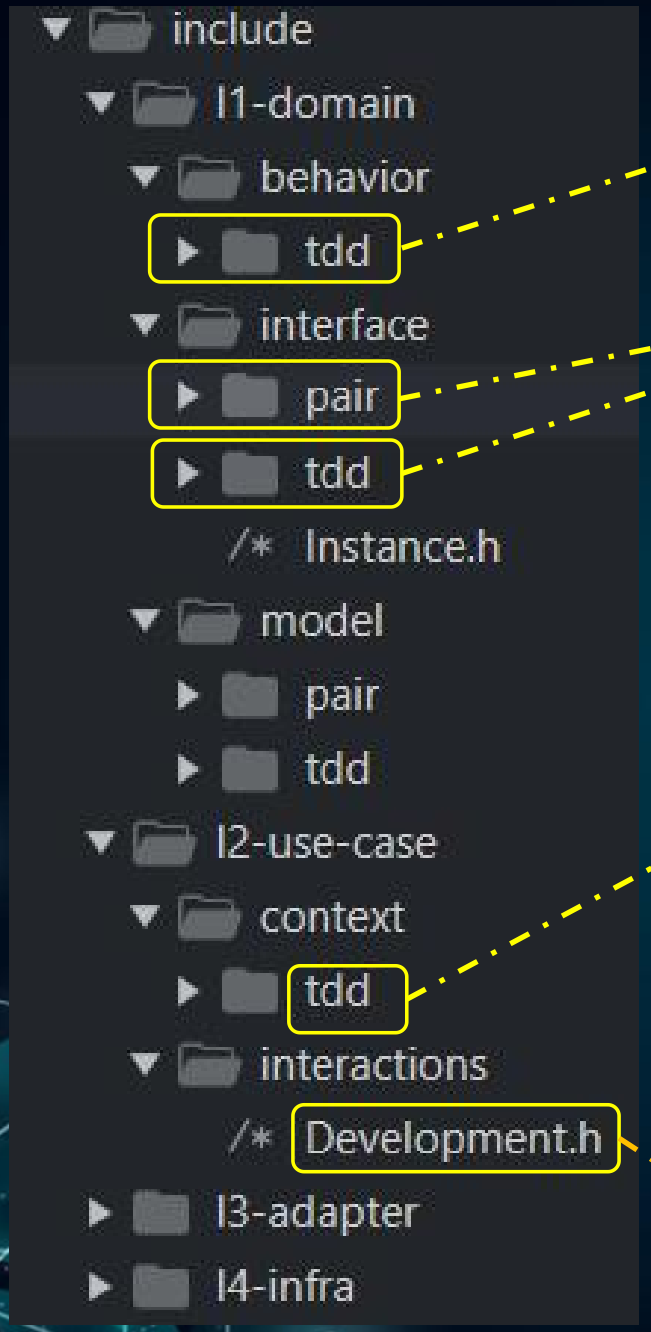



```
#define DEF_I(interface)  struct interface : ::details::Interface
#define IMPL_I(interface) IMPL_ROLE(interface)
#define I(interface) ROLE(interface)
#define USE_I(interface) USE_ROLE(interface)
#define HAS_I(interface) USE_I(interface)

////////////////////////////////////////////////////////////////////
#define DEF_B(behavior)  struct behavior : ::details::Behavior
#define IMPL_B(behavior) IMPL_ROLE(behavior)
#define B(behavior) ROLE(behavior)
#define USE_B(behavior) USE_ROLE(behavior)
#define HAS_B(behavior) USE_I(behavior)

////////////////////////////////////////////////////////////////////
#define DEF_M(model)  struct model : ::details::Model
#define IMPL_M(model) IMPL_ROLE(model)
#define M(model) ROLE(model)
#define USE_M(model) USE_ROLE(model)
#define HAS_M(model) USE_I(model)
```

```
#define ROLE(role) get##role()
#define ROLE_PROTO_TYPE(role) role& ROLE(role) const
#define USE_ROLE(role) virtual ROLE_PROTO_TYPE(role) = 0
#define HAS_ROLE(role) USE_ROLE(role)
#define IMPL_ROLE(role)                                  \
ROLE_PROTO_TYPE(role)                                    \
{                                                        \
    return const_cast<role&>(static_cast<const role&>(*this)); \
}
```




```
Status MakeOnlyEnoughCodeToPassTest::execute()
{
    I(PairInf).I(PairDriver).coding();
}
```

④

```
DEF_I(PairInf)
{
    HAS_I(PairDriver);
    HAS_I(PairObserver);
};
```

⑤

```
DEF_I(PairDriver)
{
    virtual Status coding() = 0;
    virtual Status refactor() = 0;
};
```

```
DEF_I(TddInf)
{
    HAS_B(WriteAFailingTest);
    HAS_B(MakeOnlyEnoughCodeToPassTest);
    HAS_B(ImproveCodeQuality);

    HAS_B(SendTestCompletedInd);
    HAS_B(SendRefactorCompletedInd);
};
```

③

②

```
Status TddCodingAction::exec(const TransactionInfo&)
{
   WAIT_ON(EV_TEST_COMPLETED_IND, handleTestCompletedInd);

   return CUB_CONTINUE;
}

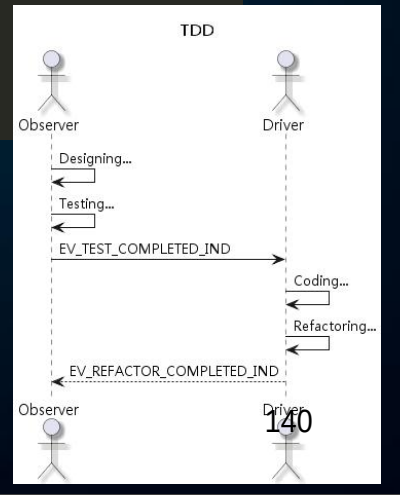
ACTION_SIMPLE_EVENT_HANDLER_DEF(TddCodingAction, handleTestCompletedInd, TestCompletedInd)
{
    std::cout << "Received EV_TEST_COMPLETED_IND " << std::endl;

    TRANS_TO_ROLE(TddInf).B(MakeOnlyEnoughCodeToPassTest).execute();

    return CUB_SUCCESS;
}
```

①

```
__def(TddTrans) __as
( __sequential(__call(TddStartAction)
              ,__call(TddTestAction)
              ,__asyn(TddCodingAction)
              ,__call(TddRefactorAction)
              ,__asyn(TddEndAction)
              )
);
```



```
struct Tdd : TddInf
           , private WriteAFailingTest
           , private MakeOnlyEnoughCodeToPassTest
           , private ImproveCodeQuality
           , private SendTestCompletedInd
           , private SendRefactorCompletedInd
           , private PairProgramming
{
    IMPL_B(WriteAFailingTest);
    IMPL_B(MakeOnlyEnoughCodeToPassTest);
    IMPL_B(ImproveCodeQuality);

    IMPL_B(SendTestCompletedInd);
    IMPL_B(SendRefactorCompletedInd);

private:
    IMPL_I(PairInf);
};
```

```
struct InstanceImpl : Instance
                    , private Tdd
                    , private Driving
{
    IMPL_I(TddInf);
    IMPL_I(DrivingInf);
};
```

```
struct PairProgramming : PairInf
                       , private ConcretePairDriver
                       , private ConcretePairObserver
{
    IMPL_I(PairDriver);
    IMPL_I(PairObserver);
};
```

```
struct ConcretePairDriver : PairDriver
{
    virtual Status coding() override;
    virtual Status refactor() override;
};
```

```
Status ConcretePairDriver::coding()
{
    std::cout << "PairDriver Coding..." << std::endl;

    return CUB_SUCCESS;
}
Status ConcretePairDriver::refactor()
{
    std::cout << "PairDriver Refactoring..." << std::endl;

    return CUB_SUCCESS;
}
```

```
include
  l1-domain
    behavior
      driving
    interface
      driving
      /* Instance.h
    model
      driving
  l2-use-case
    context
      drving
    interactions
      /* GoHome.h
  l3-adapter
    schedule
  l4-infra
```

```
DEF_I(DrivingInf)
{
    HAS_B(PlanTravelRoute);
    HAS_B(SendRoutePlanCompletedInd);
    HAS_B(StartDriving);
    HAS_B(SendDrivingStartInd);
    HAS_B(DrivingNavigation);
    HAS_B(SendReachTheGoalInd);
    HAS_B(ParkingTheCar);
};
```

```
Status StartDriving::execute()
{
    I(CarDriver).drive();

    return CUB_SUCCESS;
}
```

```
DEF_I(CarDriver)
{
    virtual Status drive() = 0;
    virtual Status park() = 0;
};
```

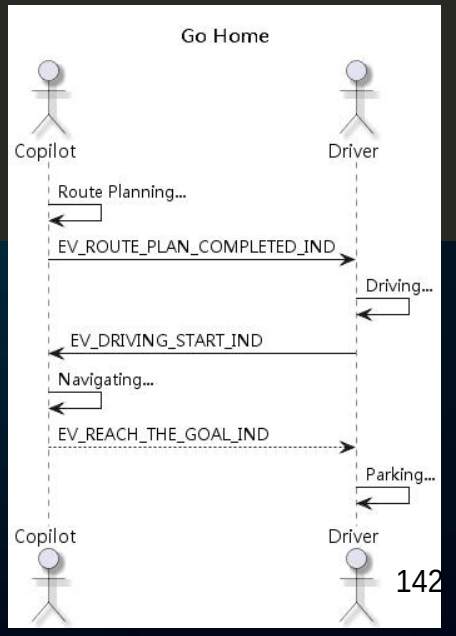
```
Status StartDrvingAction::exec(const TransactionInfo&)
{
    WAIT_ON(EV_ROUTE_PLAN_COMPLETED_IND, handleRoutePlanCompletedInd);

    return CUB_CONTINUE;
}

ACTION_SIMPLE_EVENT_HANDLER_DEF(StartDrvingAction, handleRoutePlanCompletedInd, RoutePlanCompletedInd)
{
    std::cout << "Received EV_ROUTE_PLAN_COMPLETED_IND " << std::endl;
    TRANS_TO_ROLE(DrivingInf).B(StartDriving).execute();
    TRANS_TO_ROLE(DrivingInf).B(SendDrivingStartInd).execute();

    return CUB_SUCCESS;
}
```

```
__def(GoHomeTrans) __as
( __sequential( __call(RoutePlanAction)
              , __asyn(StartDrvingAction)
              , __asyn(NavigateAction)
              , __asyn(ParkingAction)
              )
);
```




include
l1-domain
behavior
driving
tdd
interface
driving
pair
tdd
/* Instance.h
model
driving
pair
tdd
l2-use-case
context
drving
tdd
interactions
/* Development.h
/* GoHome.h
l3-adapter
schedule
l4-infra
=
include
l1-domain
behavior
tdd
interface
pair
tdd
/* Instance.h
model
pair
tdd
l2-use-case
context
tdd
interactions
/* Development.h
l3-adapter
schedule
l4-infra
+
include
l1-domain
behavior
driving
interface
driving
/* Instance.h
model
driving
l2-use-case
context
drving
interactions
/* GoHome.h
l3-adapter
schedule
l4-infra
/* cda.h

回顾

1 **背景介绍**

通信领域面临的主要问题，面向对象的门派之争：易理解、修改

2 **DDD带来的变化及存在的挑战**

组件化方法论，结构化面向对象；对象协作动态模型，系统状态和行为

3 **DCI架构（宏观+微观）**

宏观DCI，微观DCI

4 **整洁DCI架构**

整洁架构与 DCI架构融合方法；接口、行为、模型分形架构

5 **整洁DCI架构 C++案例**

整洁DCI架构物理设计，宏观DCI，微观DCI 实现模式

Q&A



李隆

鉴释科技首席科学家

目前担任鉴释科技核心技术的首席科学家，专注于代码验证基础架构。于2008年在中科大获得计算机软件和理论博士学位，学术研究包括证明基于语言的分析技术在构建高效且可靠的软件方面的应用，并发表了数篇期刊和会议论文。毕业后，李隆博士加入了三星电子，从事高级技术小组的统计机器翻译工作。并于2010年加入HP编译器团队，从事HP Non-Stop编译器后端和SDK。

CPP-Summit

UWASM

– An Enhanced WASM Framework

Long Li

XCALIBYTE

Agenda

1. **Background**
   WASM & UWASM introduction
2. **UWASM Framework**
   Whole framework of UWASM toolchain
3. **UWASM Core**
   UWASM instruction set & binary format
4. **UWASM VM**
   Implementation of UWASM virtual machine
5. **UWASM Future**
   Future works of UWASM

01

Background

WASM-WEBASSEMBLY

- Standard from W3C
- Able to compile Modern type language into Bytecode, C/C++/Rust/Typescript
- Well supported in V8 based browser
- Getting popular in embedded space
  - Modern type language support
  - Compile once run anywhere
  - Reasonable speed depending on the implementation of VM

WASM Safety

- "Describes a memory-safe, sandboxed execution environment that may even be implemented inside existing JavaScript virtual machines."
- Soundness
  - Progress & Preservation
  - "The type system of WebAssembly is *sound*, implying both *type safety* and *memory safety* with respect to the WebAssembly semantics."

WASM Soundness

Theorems

Given the definition of valid configurations, the standard soundness theorems hold. [2]

Theorem (Preservation). If a configuration $S; T$ is valid with result type $[t^*]$ (i.e., $\vdash S; T : [t^*]$), and steps to $S'; T'$ (i.e., $S; T \hookrightarrow S'; T'$), then $S'; T'$ is a valid configuration with the same result type (i.e., $\vdash S'; T' : [t^*]$). Furthermore, S' is an extension of S (i.e., $\vdash S \preceq S'$).

A *terminal* thread is one whose sequence of instructions is a result. A terminal configuration is a configuration whose thread is terminal.

Theorem (Progress). If a configuration $S; T$ is valid (i.e., $\vdash S; T : [t^*]$ for some result type $[t^*]$), then either it is terminal, or it can step to some configuration $S'; T'$ (i.e., $S; T \hookrightarrow S'; T'$).

From Preservation and Progress the soundness of the WebAssembly type system follows directly.

Corollary (Soundness). If a configuration $S; T$ is valid (i.e., $\vdash S; T : [t^*]$ for some result type $[t^*]$), then it either diverges or takes a finite number of steps to reach a terminal configuration $S'; T'$ (i.e., $S; T \hookrightarrow^* S'; T'$) that is valid with the same result type (i.e., $\vdash S'; T' : [t^*]$) and where S' is an extension of S (i.e., $\vdash S \preceq S'$).

In other words, every thread in a valid configuration either runs forever, traps, or terminates with a result that has the expected type. Consequently, given a valid store, no computation defined by instantiation or invocation of a valid module can "crash" or otherwise (mis)behave in ways not covered by the execution semantics given in this specification.

WASM Soundness

- Progress
  - A well-typed program can excution one step forward
- Preservation
  - A well-typed program takes one step forward, then the resulting program is still well-typed

A WELL-TYPED PROGRAM WILL NOT GET STUCK!

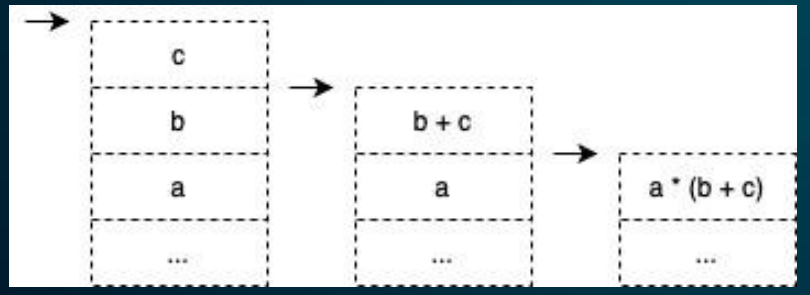
WASM Instructions

nn, *mm*	::=	32 | 64
sx	::=	u | s
instr	::=	*inn*.const *inn* | f*nn*.const *fnn*
	|	*inn*.*iunop* | f*nn*.*funop*
	|	*inn*.*ibinop* | f*nn*.*fbinop*
	|	*inn*.*itestop*
	|	*inn*.*irelop* | f*nn*.*frelop*
	|	*inn*.extend8_s | *inn*.extend16_s | i64.extend32_s
	|	i32.wrap_i64 | i64.extend_i32_*sx* | *inn*.trunc_f*mm*_*sx*
	|	*inn*.trunc_sat_f*mm*_*sx*
	|	f32.demote_f64 | f64.promote_f32 | f*nn*.convert_i*mm*_*sx*
	|	*inn*.reinterpret_f*nn* | f*nn*.reinterpret_*inn*
	|	…
iunop	::=	clz | ctz | popcnt
ibinop	::=	add | sub | mul | div_*sx* | rem_*sx*
	|	and | or | xor | shl | shr_*sx* | rotl | rotr
funop	::=	abs | neg | sqrt | ceil | floor | trunc | nearest
fbinop	::=	add | sub | mul | div | min | max | copysign
itestop	::=	eqz
irelop	::=	eq | ne | lt_*sx* | gt_*sx* | le_*sx* | ge_*sx*
frelop	::=	eq | ne | lt | gt | le | ge

WASM Execution Environment

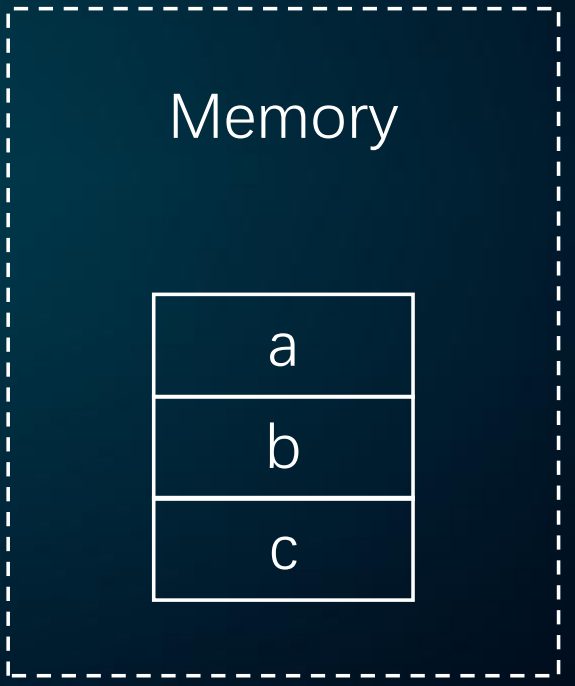
- Stack Machine

```
push a
push b
push c
add
mul
```



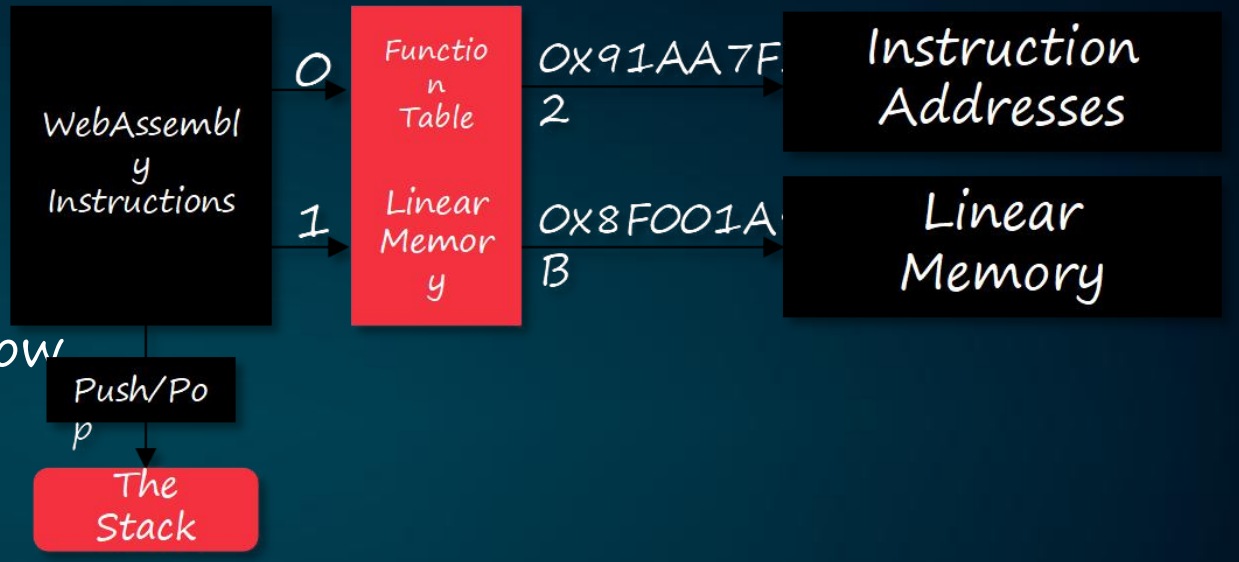
- Linear Memory

```
struct AA {
  int a;
  char b;
  long c;
};
```



WASM Execution Environment

- Stack Machine
  - Instruction operation on stack
    - Type correctness
    - No stack overflow or underflow
- Linear Memory Model
  - Linear buffer allocation
  - Untyped byte data
  - Linear code section
- Function call
  - Function table array
  - Indirect call by index





WASM-WEBASSEMBLY

- Performance upper limit
- WASM is NOT security proven
- Compiling C language into WASM still exposes risk in DOS/XSS

CODE Snippet

Typical protocol structure

- Buffer: msg[32]
- Size: len
- Operation: send

```
/*
 * Copyright 2021 The Xcalibyte Authors.  All rights reserved.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <emscripten.h>

// example to show fundamental flaw of emcripsten and WASM
// linear stack overflow
// WASM function index overwrite
// can be anything, lets have fun

//protocol stack structure
typedef struct XSS_Send {
  char msg[32];
  uint16_t len;
  void (*send)(const char *);
} XSS_Send;

//actual send func
void send(const char *msg){
  printf("%s", msg);
}
```

Overwrite function pointer 'send' via data field 'msg'



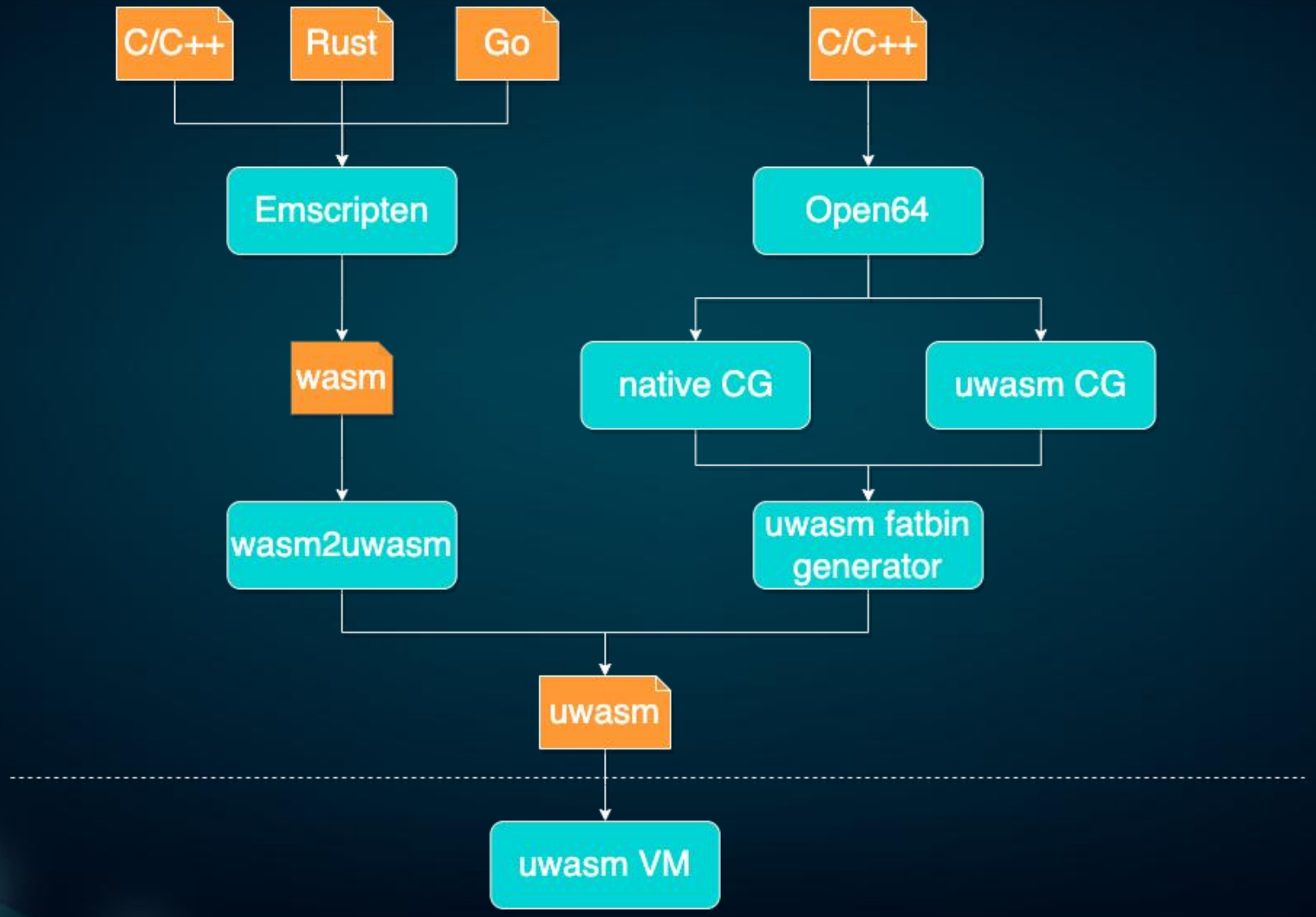
UWASM - MICRO WASM

- Compatible with WASM
- Fix-length bytecode format
- Register based — unlimited register
- Compiler and VM cooperation
  - SAST/DAST framework
  - Compiler optimization to reduce overhead
- Security
  - Data/Code isolation
  - Flexible runtime control

02

UWASM Framework

UWASM Framework



UWASM Toolchain Components

- Compiler - open64
  - Well-defined multilevel IR
  - Automatic parallel partition & Loop optimization
  - Xcalibyte enhanced SAST
  - Instrumentation for runtime check

- UWASM CG
  - Input: Open64 WHIRL IR
  - Output: UWASM binary file (UWASM encoded instructions & data)

UWASM Toolchain Components Cont.

- Native CG
  - Input: Open64 WHIRL IR
  - Output: native binary file (x86/Mips/RISC-V)

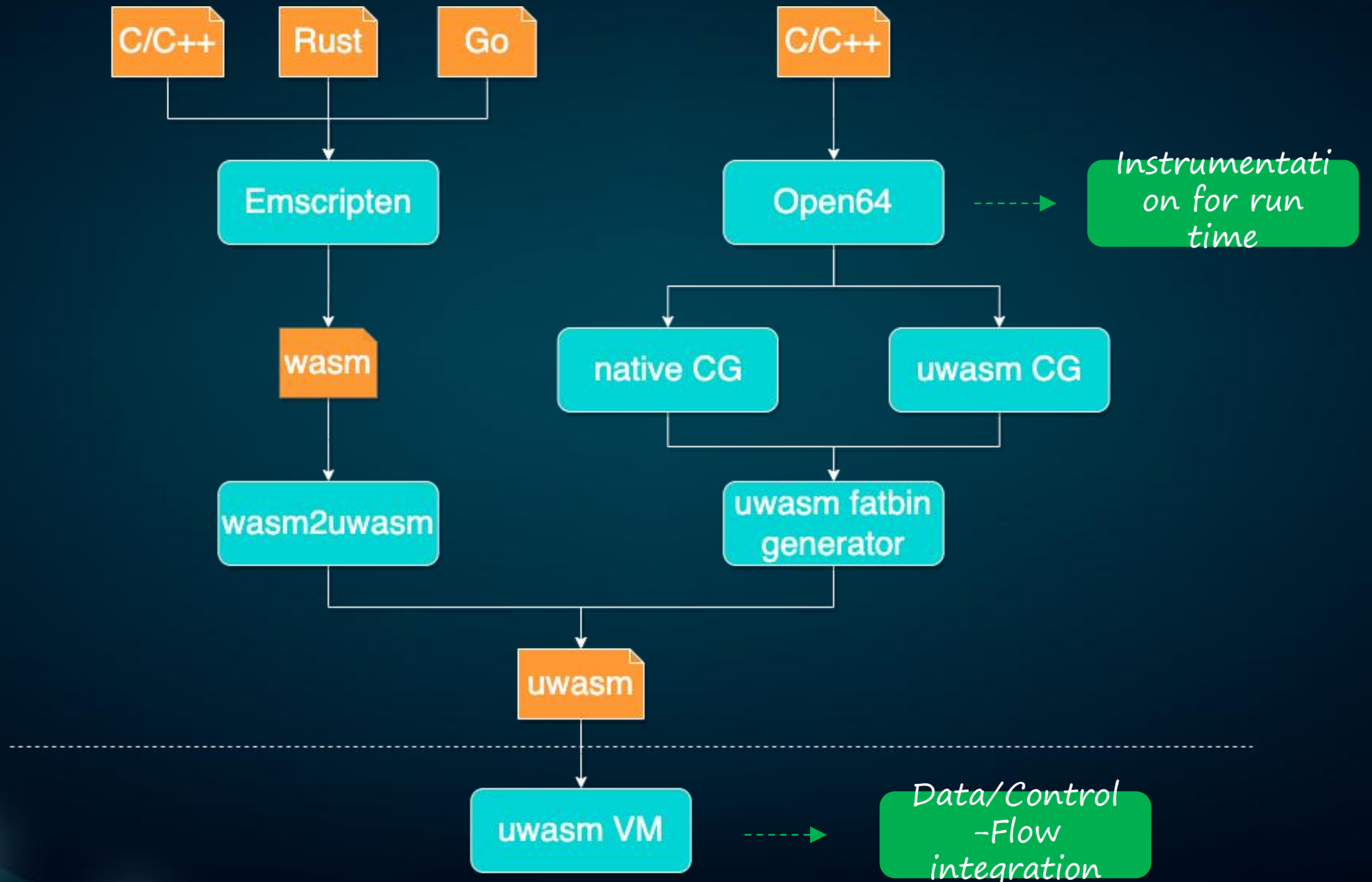
- UWASM Fatbin Generator
  - Merge UWASM file and native binary file

UWASM Toolchain Components Cont.

- UWASM VM
  - Interpreter for UWASM binary file

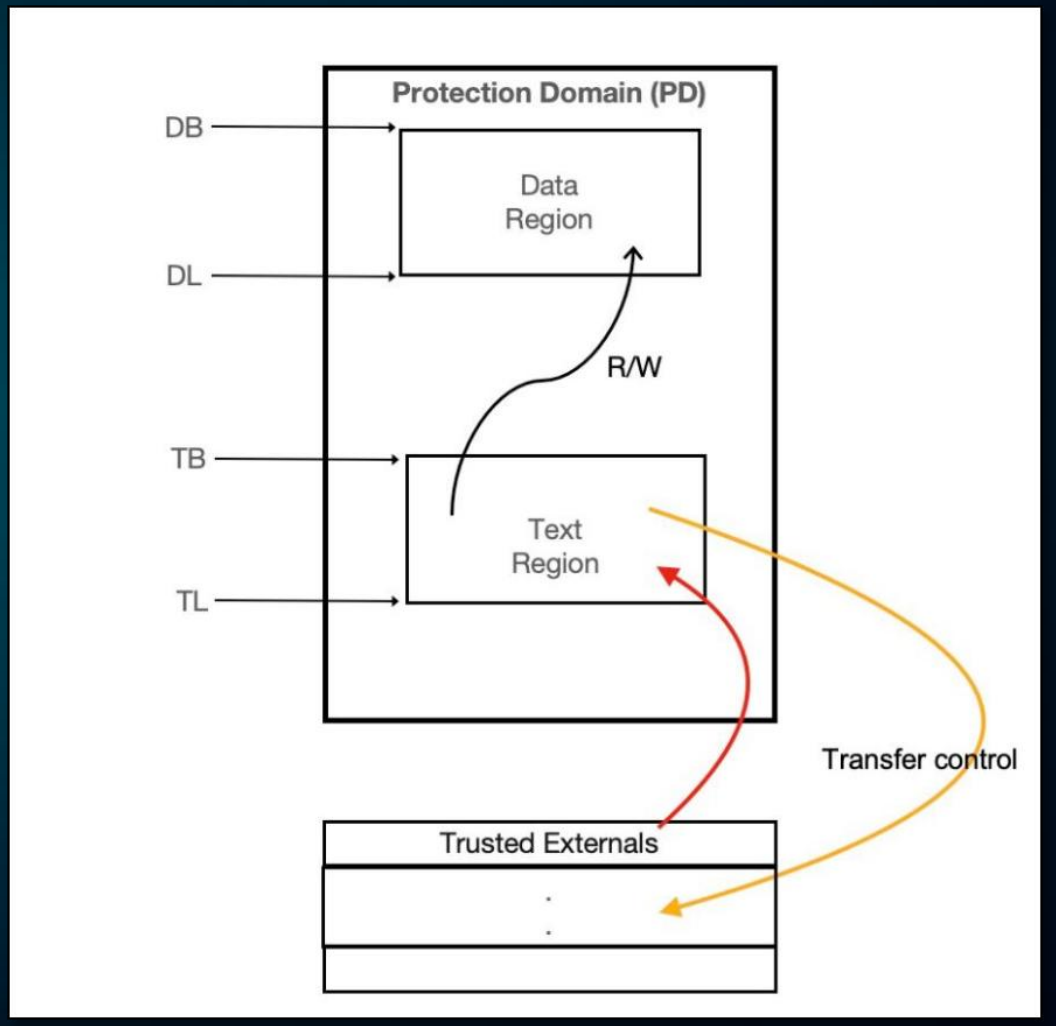
- Wasm2Uwasm
  - A standalone tool to convert WASM file into UWASM file
  - Converting on instruction level
  - No optimizations during conversion

UWASM Security



UWASM Software-based Fault Isolation

- Ensure data integrity
  - All memory reads and writes by the code should be within the data region
- Ensure control-flow integrity
  - A control-flow transfer by the code must stay within the code region or target a safe external address



Data Integrity

$mem(r_1 + 12) := r_2$ ⟶

$r_{10} := r_1 + 12$
if (r_{10} < DB) goto error
if (r_{10} > DL) goto error
$mem(r_{10}) := r_2$

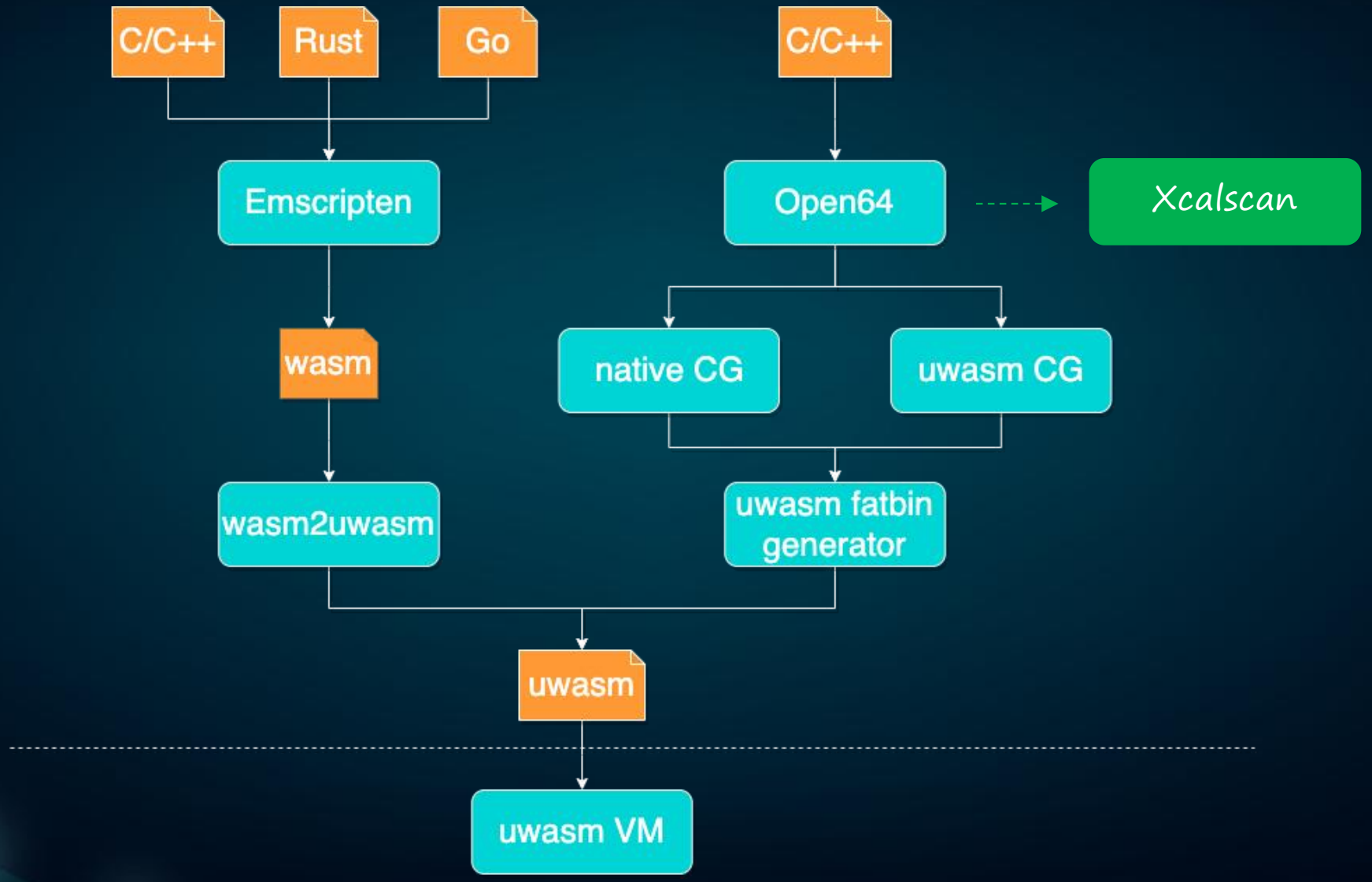
Control-flow Integrity

"All control-flow transfers must target the beginning of a pseudo-instruction in the code region or target a safe external address"

l_1: $r_{10} := r_1 + 12$
l_2: $r_{10} := dGuard(r_{10})$
l_3: $mem(r_{10}) := r_2$ // What if jump to l_3 ?

pseudo-instruction: non-dangerous instruction or a guard followed by a dangerous instruction.

UWASM + SAST Tool



03

UWASM Core

UWASM Register

- Registers – unlimited (R0-RN)
  - Return (R0-R1)
  - Formal parameter
  - Locals
- Data
  - i32: 32-bit integer
  - i64: 64-bit integer
  - f32: 32-bit floating point
  - f64: 64-bit floating point

i32 reg file		i64 reg file		f32 reg file		f64 reg file	
R0	ret	RD0	ret	F0	ret	FD0	ret
	param		param		param		param
RN	locals	RDN	locals	FN	locals	FDN	locals

Address Mode

- Fixed length 32/64 bit

	0	1	2	3 4..7	8..9	10..11	12..15	16	17	18.19	20.21	22.23	24	25	26.27	28.29	30.31
am32s_r_r_imm	0	Opcode (7 bits)			Valtype	Rd (lower 128 numbers)				Rs (source reg)			Signed immediate (8 bits)				
am32s_r_r_ofs	0	Opcode			Valtype	Rd				Rs			Unsigned offset				
am32s_r_imm_ofs	0	Opcode (7 bits)			Valtype	Rs				7 bit signed immediate			Unsigned offset				
am32s_r_r_r	0	Opcode			Valtype	Rd				Rs			nu	Rt			
am32_r_ofs	0	Opcode (7 bits)			Valtype	Reg (by valtype)						Unsigned offset					
am32_r_imm	0	Opcode			Valtype	Rd (dest reg) (store inst has Rs and Rd reversed)						Signed immediate					
am32_r_r	0	Opcode			Valtype	Rd (dest reg)						Rs (source reg)					
am32_imm_ofs	0	Opcode			Valtype	Signed Immediate						Unsigned offset					
am32_ofs	0	Opcode			Valtype	Unsigned offset (22bits)											
am32_r	0	Opcode			Valtype	Unused						Rs (source reg, typically for indirection)					
am32_r_ofs	0	Opcode			Valtype	Rd (dest reg)						Unsigned offset					
am64_r_ofl	1	Opcode (7 bits)			xValtype: vec, sat vec etc	Rd (dest reg)						Unused					
	32 bit offset																
am64_ofs_imm	1	Opcode			xValtype: vec, sat vec etc	20 bit offset											
	32 bit imm																
am64_r_r_ofs	1	Opcode			xValtype: vect, sat vec etc	Rd (dest reg) (store inst has Rs and Rd reversed)						Unused					
	24 bit offset									Unused		Rs (source reg) (typically - ((Rs)+ofs)					
am64_r_r_r	1	Opcode			xValtype: vect, sat vec etc	Rd (dest reg)						Unused					
	Unused				Unused	Rs (source reg 1)				Unused		Rt (source reg 2)					

Instructions

- 1 - 1 map with WASM instruction
- Numeric Instructions
  - arithmetic : add, sub, mul, div, rem, min, max
  - Logic: and, or, xor, shl, shr, rotl, rotr, ...
  - Relational: eq, ne, lt, gt, le, ge, lt_u, gt_u,...
  - conv, const, ...
- Memory Instructions
  - load/store: load, lload, store, lstore, gget, gset
  - memorysize, memorygrow
- Control Instructions
  - nop, unreach, return, call, icall, jmp, ijmp, ...

Instructions - Example

```
int fibonacci(int num) {
  if (num <= 1) return num;

  return fibonacci(num - 1) + fibonacci(num - 2);
}
```

```
(func $fibonacci (type $tfibonacci) (result i32) (param $r1 i32)
    (local $r2 i32) (local $r3 i32) (local $r4 i32) (local $r5 i32)
 # io reg info (length):start-index [r(2):6 rd(0):0 f(0):0 fd(0):0]
  .loc  1 7 0
  i32.const r3 1                    #
  i32.jgt r1 r3 .Lt_1_770           #
  i32.mov r0 r1                     #
  jmp .LBB6_fibonacci               #
.Lt_1_770:
  .loc  1 9 0
  i32.addi r7 r1 -1                 #
  call $fibonacci                   # fibonacci
  i32.mov r4 r6                     #
  i32.addi r7 r1 -2                 #
  call $fibonacci                   # fibonacci
  i32.mov r5 r6                     #
  i32.add r0 r4 r5                  #
.LBB6_fibonacci:
   return                           #
)
```

UWASM binary format

	0..1	2..3	4..5	6..7	8..11	12..15	16..19	20..23	24..27	28..31
uWASM Module header	U (in lower case)				A		S		M	
	Other magic									
	Section header entry size						Number of sections			
Section header table entries	Section id		Generail unused or special use as defined				Number of entries in vector			
...	Index to section vector									
	Entry 2 ···									
	Last entry - end of section header table									
Specific section definition	All indices are relative to beginning of Module header									
Customer section	TBD									
Type section (func type)	Tag: 0x1		Unused				Number of elements in vector			
	Index to vector (corresponding)									
Import section	Tag : 0x2		Number of elements in vector (we do not expect more than 2**12 imports)				Number of elements in vector			
	Index to vector									
Function section	Tag: 0x3		Unused				number of elements in vector			
	Index to functions (local and body are in code section, the function section is only prototype/signature of the function)									
Table section (may occur multiple times)	Tag: 0x4		Unit size (bytes, 4 bytes···)		Index into specific table (e.g. name string table etc)					
	Number of elements in table (unit as in specified in unit size)									
Memory section	Tag: 0x5		Unused				Number of elements in vector			
	Index to vector									
Global section	Tag : 0x6		Type (32/64)		Number of elements in vector					
(may occur multiple times)	Index to vector									
Export section	Tag: 0x7		Unused		Number of elements in vector					
	Index to vector									
Start section (unused)	tag: 0x8									
Element section	TBD									
Code section	Tag: 0xA		Unused				Number of elements in vector			
	index to vector									

UWASM binary format - Example

```
========================================================================
FILE HEADER
========================================================================
[HEADER] magic: uASM, section header size: 824, section number: 11
========================================================================
SECTION HEADERS
========================================================================
[SH] id: 01, tag:  1,          , elem num:    3, index:   12
[SH] id: 02, tag:  7, kind:  0, elem num:    2, index:   24
[SH] id: 03, tag:  2, kind:  1, elem num:    6, index:   32
[SH] id: 04, tag:  3,          , elem num:    8, index:  104
[SH] id: 05, tag: 10,          , elem num:    2, index:  168
[SH] id: 06, tag:  5,          , elem num:   36, index:  324
[SH] id: 07, tag:  6, kind:  0, elem num:    8, index:  360
[SH] id: 08, tag:  4, kind:  2, elem num:   16, index:  488
[SH] id: 09, tag:  4, kind:  0, elem num:  160, index:  504
[SH] id: 10, tag:  4, kind:  1, elem num:  160, index:  664
[SH] id: 11, tag:  8,                , index:    0
========================================================================
TYPE TABLE:
========================================================================
[   0] origin: ret num: 1, param num: 0, raw data: 0x00000000 (00 00000000 00000000 00000000)
[   0] cache:  (i32) ()
[   1] origin: ret num: 1, param num: 1, raw data: 0x00000000 (00 00000000 00000000 00000000)
[   1] cache:  (i32) (i32 p1)
[   2] origin: ret num: 0, param num: 0, raw data: 0x00000000 (00 00000000 00000000 00000000)
[   2] cache:  () ()
========================================================================
IMPORT SECTION
========================================================================
Empty
========================================================================
IMPORT NATIVE SECTION
========================================================================
Import native function:    2, native type offset:    1, no in native section.
Import native function:    3, native type offset:    1, no in native section.
Import native function:    4, native type offset:    1, no in native section.
Import native function:    5, native type offset:    1, no in native section.
Import native function:    6, native type offset:    1, no in native section.
Import native function:    7, native type offset:    1, no in native section.
========================================================================
EXPORT SECTION
========================================================================
Export function: 0
Export function: 1
========================================================================
FUNCTION TABLE:
========================================================================
;; Function index: 0, main  [0] (i32) ()
;; RET and PARAM num:          (i32: 1, i64: 0, f32: 0, f64: 0) (i32: 0, i64: 0, f32: 0, f64: 0)
;; Local register num:  (i32: 3, i64: 0, f32: 0, f64: 0)
;; IO register num:                    (i32: 2, i64: 0, f32: 0, f64: 0)
;; All register num:            (i32: 5, i64: 0, f32: 0, f64: 0)
;; Stack size: 0
;; Code sz: 5, label number: 0

[0000]          const      R4         10
[0001]          call       1
[0002]          mov        R2         R3
[0003]          mov        R0         R2
[0004]          return

[LBL]
empty table
```

```
------------------------------------------------------------------------
;; Function index: 1, fibonacci  [1] (i32) (i32 p1)
;; RET and PARAM num:          (i32: 1, i64: 0, f32: 0, f64: 0) (i32: 1, i64: 0, f32: 0, f64: 0)
;; Local register num:  (i32: 6, i64: 0, f32: 0, f64: 0)
;; IO register num:                    (i32: 2, i64: 0, f32: 0, f64: 0)
;; All register num:            (i32: 8, i64: 0, f32: 0, f64: 0)
;; Stack size: 0
;; Code sz: 12, label number: 2

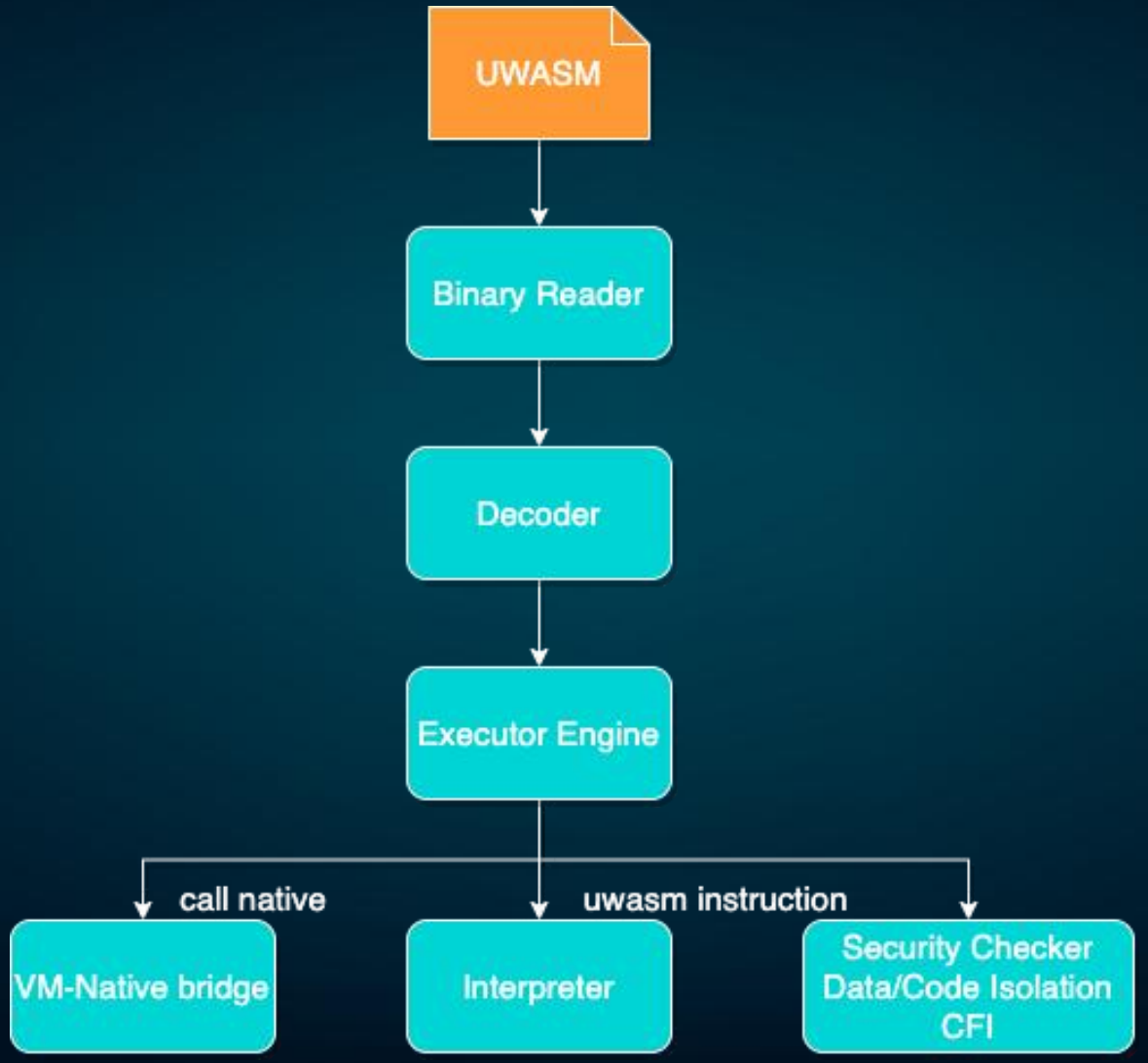
[0000]          const      R3         1
[0001]          igt        R1         R3         0
[0002]          mov        R0         R1
[0003]          jmp        1
[0004]          add        R7         R1         -1
[0005]          call       1
[0006]          mov        R4         R6
[0007]          add        R7         R1         -2
[0008]          call       1
[0009]          mov        R5         R6
[0010]          add        R0         R4         R5
[0011]          return

[LBL]
0000: 0004
0001: 0011
========================================================================
DATA SECTION
========================================================================
[Memory section] length = 36(0x00000024)
646174610000000001000000020000000300000004000000050000000600000007000000
========================================================================
GLOBAL TABLE
========================================================================
[   0] off:    4, mut:    0, type: i32
[   1] off:    8, mut:    0, type: i32
[   2] off:   12, mut:    0, type: i32
[   3] off:   16, mut:    0, type: i32
[   4] off:   20, mut:    0, type: i32
[   5] off:   24, mut:    0, type: i32
[   6] off:   28, mut:    0, type: i32
[   7] off:   32, mut:    0, type: i32
========================================================================
NON NULL TERMINATED STRING TABLE
========================================================================
00000000000000000000000000000000
========================================================================
NATIVE SECTION
========================================================================
No native section found.
========================================================================
========================================================================
```

04

UWASM VM

UWASM VM – Data Flow



UWASM VM Components – Binary Reader

- Binary Reader
  - open uwasm file and mmap into VM memory
  - Read section header
  - Read each section and cache in internal data structure

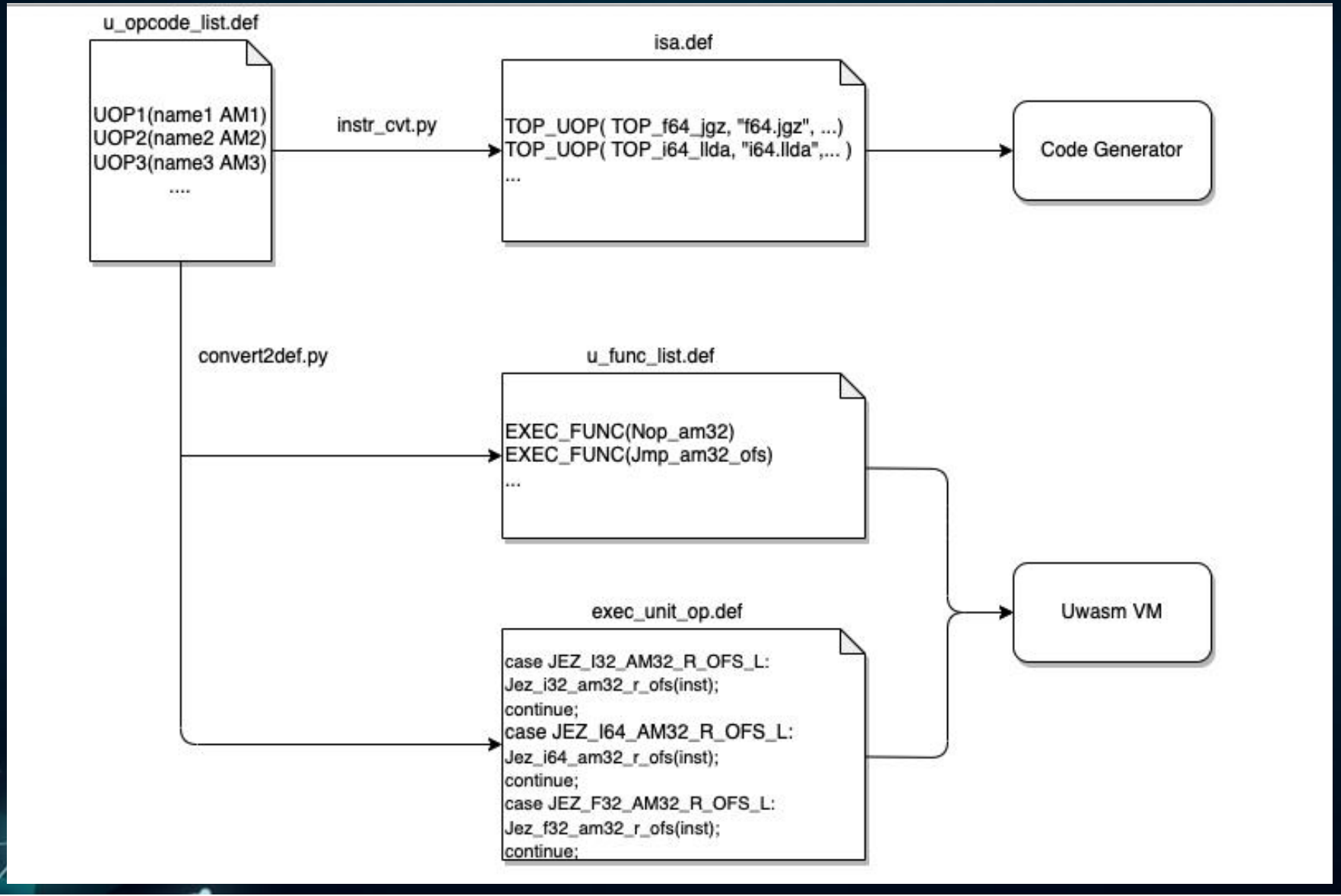
UWASM VM Components - Decoder

- Decoder
  - Expand binary form instructions into expanded internal data-struct
  - Decode once fasten execution

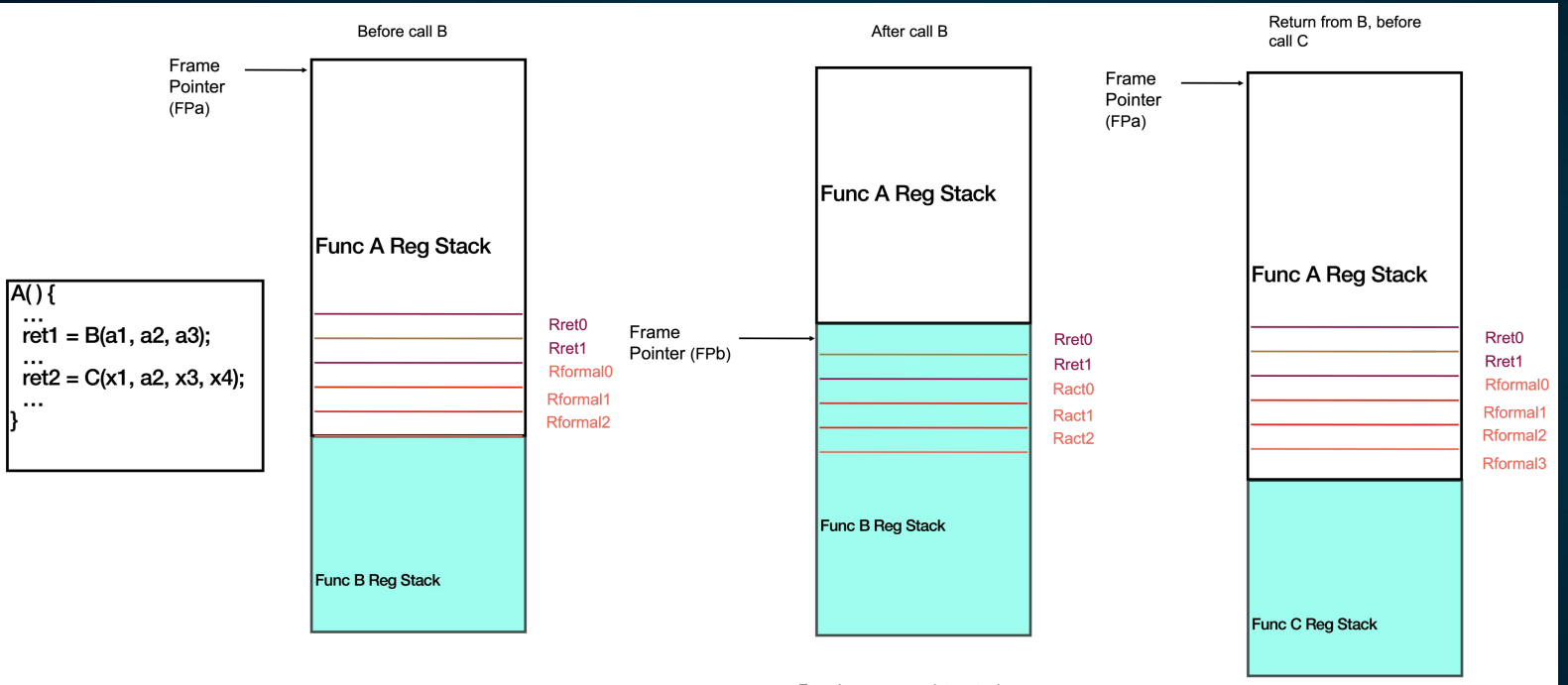
UWASM VM Components - Executor

- Executor Engine
  - Search entry function, set pc to first instruction
  - Call instruction's handler function
  - High efficiency development model by abstraction

UWASM VM Components - Executor



UWASM Call ABI



VM execution engine:

- maintain an infinite size register file
- Set current frame pointer upon entry from call instruction
  - Register number starts from 0 on entry
- keeps a stack of frame pointers and maintain current frame pointer
  - Push caller FP on function call
  - Pop after return from call (restore caller FP)
- R0 and R1 is always return register 0 and return register 1

Function scope register stack:

- Always offset based
- Max 1024 size
- 4 stacks, one per ValType
- Includes: locals + temps + return regs + parameters
- From caller view: return regs + formals at end of reg stack (after local and temps)
- From callee view: return regs + actuals at beginning of reg stack
- Within one function, formal area is max of all calls inside the function
  - Max size of locals + temps = 1024 - (max_formal_area + return reg (2))

UWASM Call Native

- Read native function info
- Create cif for each function
- Prepare data/memory for parameter & return
- Libffi call (foreign function interface library)
- Get return value and save to VM register/memory

```
IMPORT NATIVE SECTION
================================================================================
Import native function:   24, native type offset:   16, native function offset: 6496
Import native function:   25, native type offset:   19, native function offset: 20640
```

```
--------------------------------------------------------------------------------
;; Function index: 24, calc_func  [0] (i32) (i32 p1, i32 p2)
;; RET and PARAM num:           (i32: 1, i64: 0, f32: 0, f64: 0) (i32: 2, i64: 0, f32: 0, f64: 0)
;; Extern function
--------------------------------------------------------------------------------
;; Function index: 25, core_bench_state  [11] (i32) (i32 p1, i32 p2, i32 p3, i32 p4, i32 p5, i32 p6)
;; RET and PARAM num:           (i32: 1, i64: 0, f32: 0, f64: 0) (i32: 6, i64: 0, f32: 0, f64: 0)
;; Extern function
```

UWASM VM Performance

- Benchmark – Coremark
- WASM Engine – Emscripten 1.39.15 (Based on LLVM 11.0.0
- UWASM Engine - Open64

WASM Engine	Opt. Level	Node.js 12.18.1 V8 7.8.279.23	Node.js 14.4.0 V8 8.1.307.31	Node.js 15.10.0 V8 8.6.395.17	UWASM Engine Option	UWASM Score
Interpreter	-O2	43	64	N/A	-O2	896
	-O3	44	67	N/A	-O3	978
Liftoff	-O2	6619	8690	9775	-aot	24315
	-O3	6654	8660	9221		
Turofan	-O2	13708	15420	15640		
	-O3	13843	15860	16024		

05

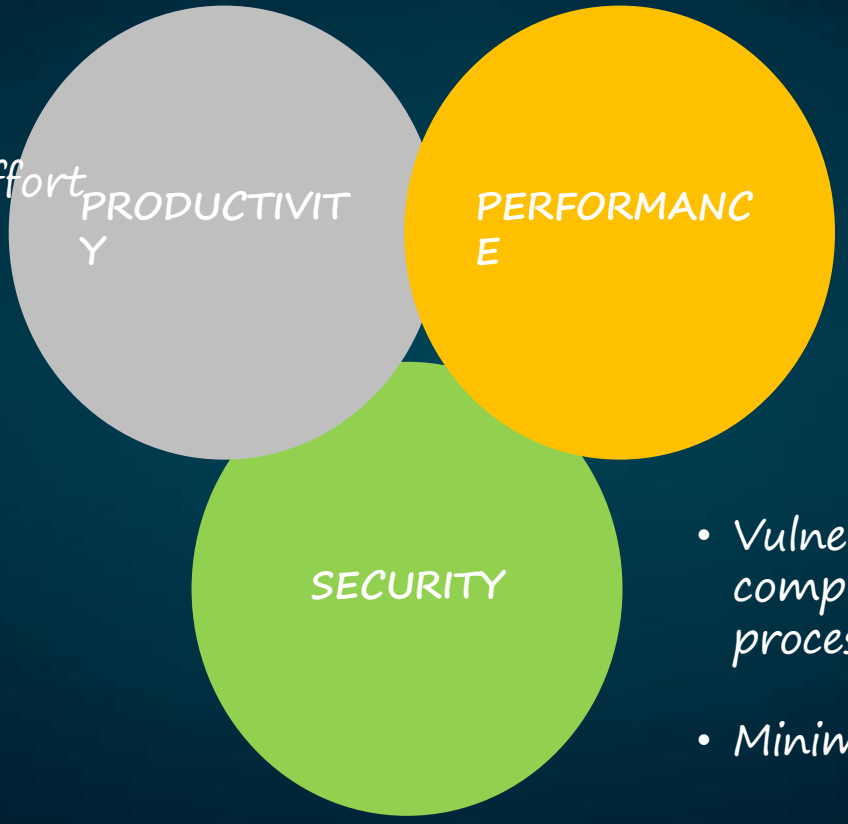
UWASM Future

UWASM Future

Application – Existing apps migration effort

Mix language compilation system

Integrate SAST tool



- Native and parallel execution for compute intensive
- Dynamic and real-time runtime scheduling based on resource available

- Vulnerability checks along compilation, start-up and execution process
- Minimum performance hit

Thanks



赵永刚

Boolan 资深咨询师

16年C/C++系统级软件开发经验，前诺基亚贝尔资深架构师，内部技术教练，是系统软件架构、代码安全、性能优化方面的专家。开发经验涉及通信、嵌入式Linux开发、自动驾驶系统软件产品开发的多个领域。14 年嵌入式系统开发经验，从上层应用开发到底层 硬件适配层开发均有丰富的经验。对系统级软件架构设计、重构与守护、设计模式、开发者测试、性能优化、安全编码等有深入研究和丰富经验。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

C++开发者测试最佳实践

赵永刚
Boolan资深技术顾问

议 程

1. 开发者测试
   定义、分类和主要活动
2. 可测性设计
3. 测试用例设计

01

开发者测试

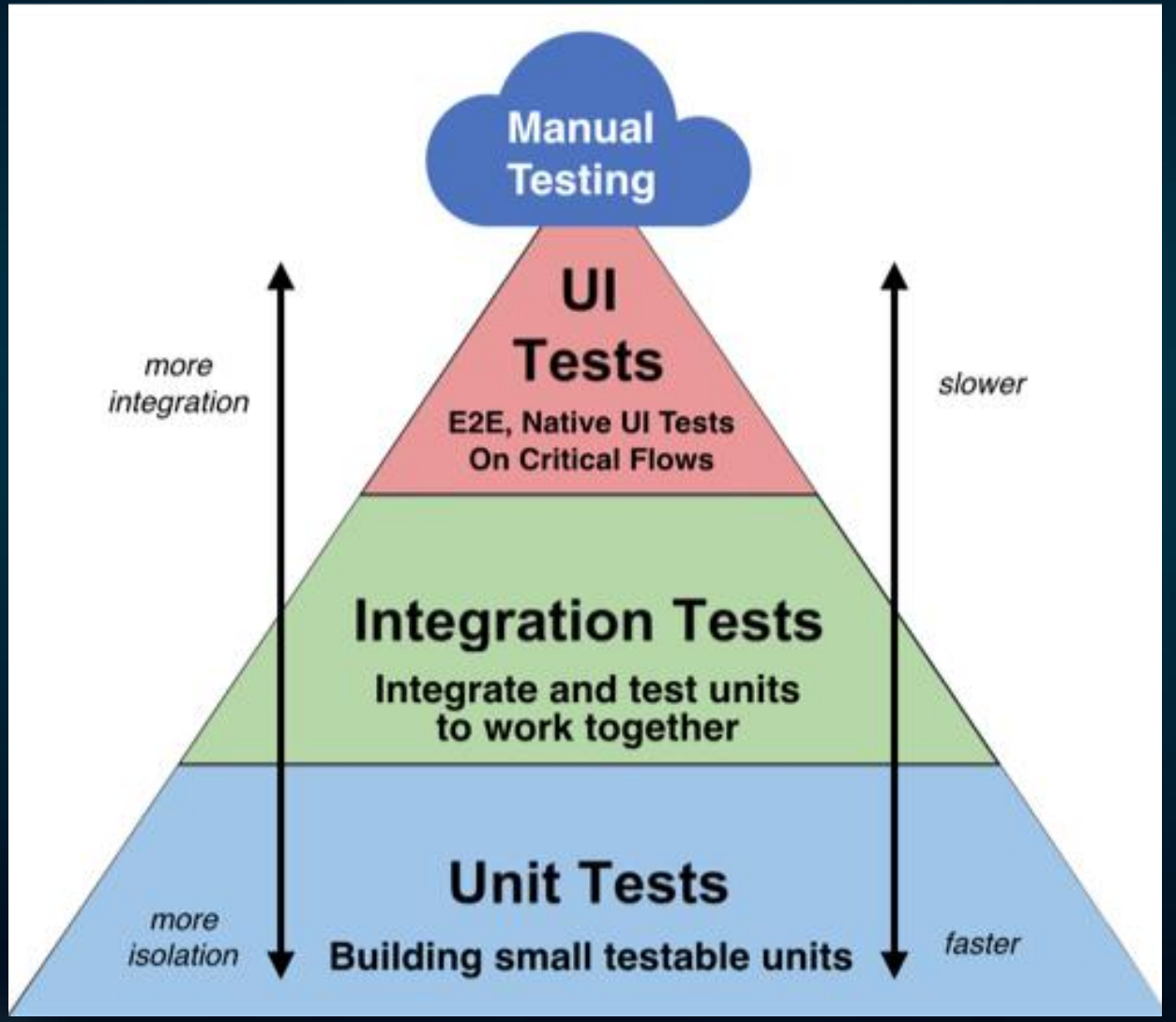
开发者测试

- 开发者做的测试
- 是软件开发工程师
  - **有意和系统地**使用测试工具和技术
  - 创建具有尽可能少的缺陷的软件
- 目标：可测试和可维护的软件。

谁来做测试？



开发者测试


Manual Testing
UI Tests
E2E, Native UI Tests On Critical Flows
Integration Tests
Integrate and test units to work together
Unit Tests
Building small testable units
more integration
slower
more isolation
faster

测试左移

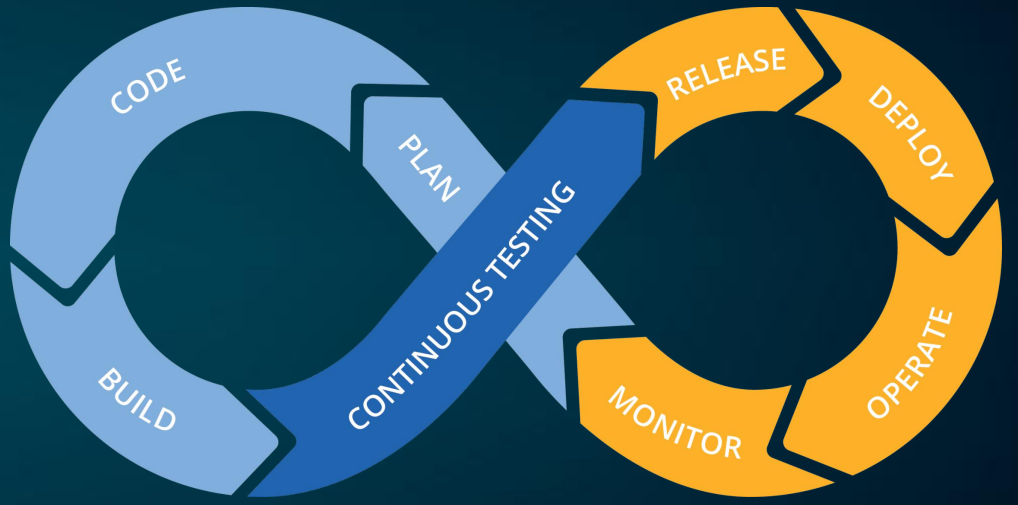




左移 = 降本 + 增效 + 提高质量 + 竞争力

开发者测试

- 是一种开发活动
  - 自动化
  - 帮助设计质量
  - 帮助提升开发效率
- 与CI/CD结合



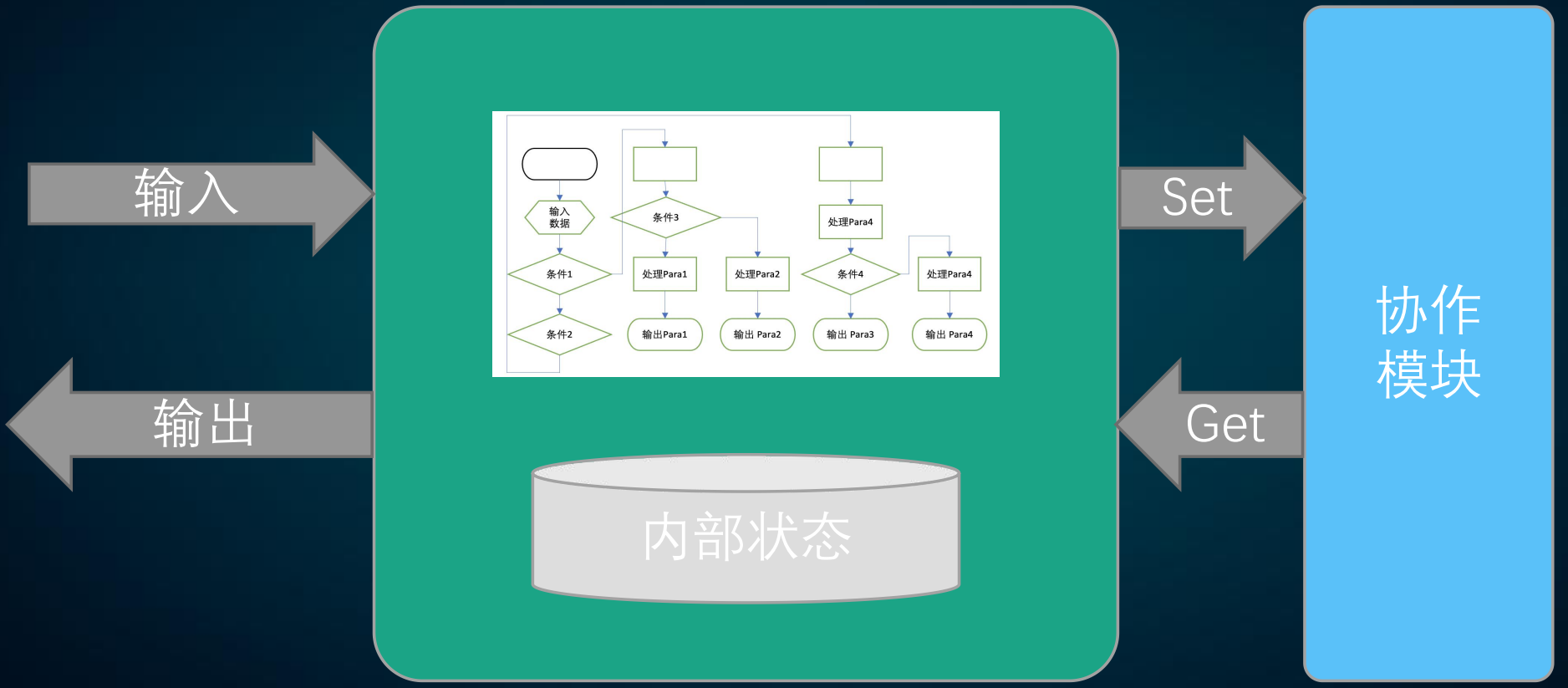
02

可测性设计

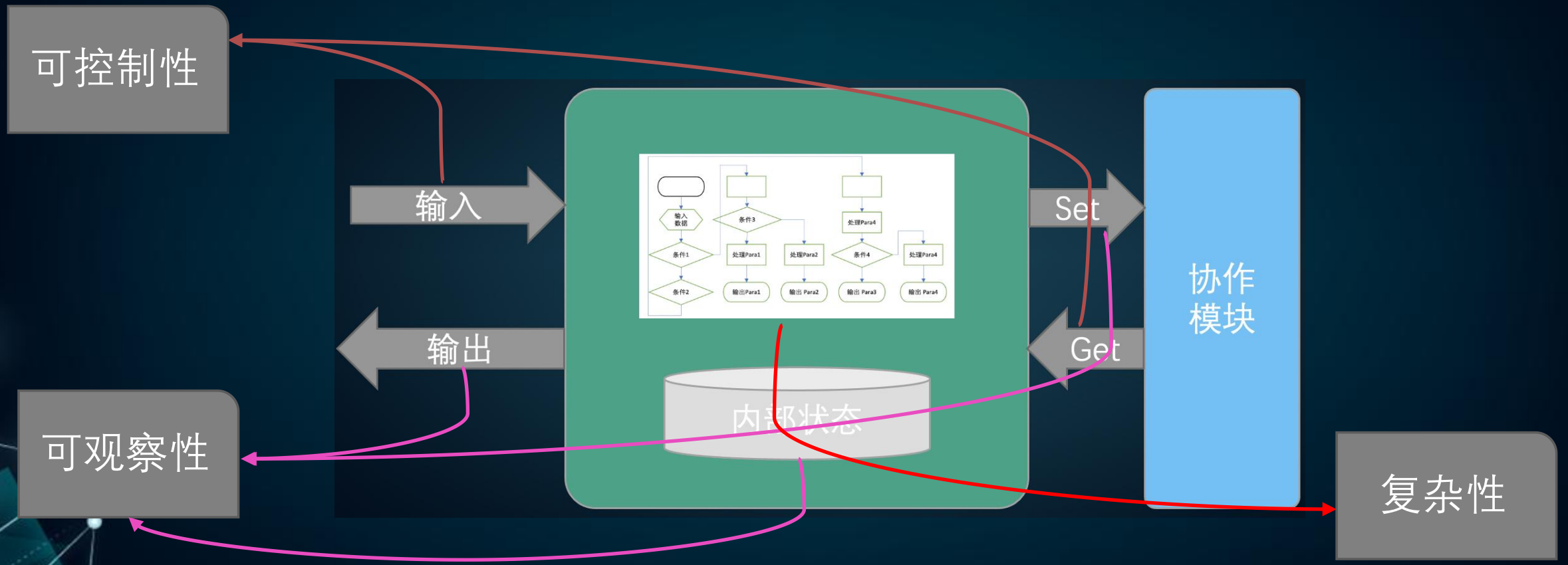
软件的可测性

- 软件质量的一个属性
- 可测性：软件被测试的便利程度
- ISO: Define testability "Attributes of software that bear on the effort needed to validate the software product”

软件测试的模型


输入
输出
输入数据
条件1
条件2
条件3
处理Para1
处理Para2
输出Para1
输出 Para2
处理Para4
条件4
处理Para4
输出 Para3
输出 Para4
内部状态
Set
Get
协作模块

可测性



提高可测性的策略



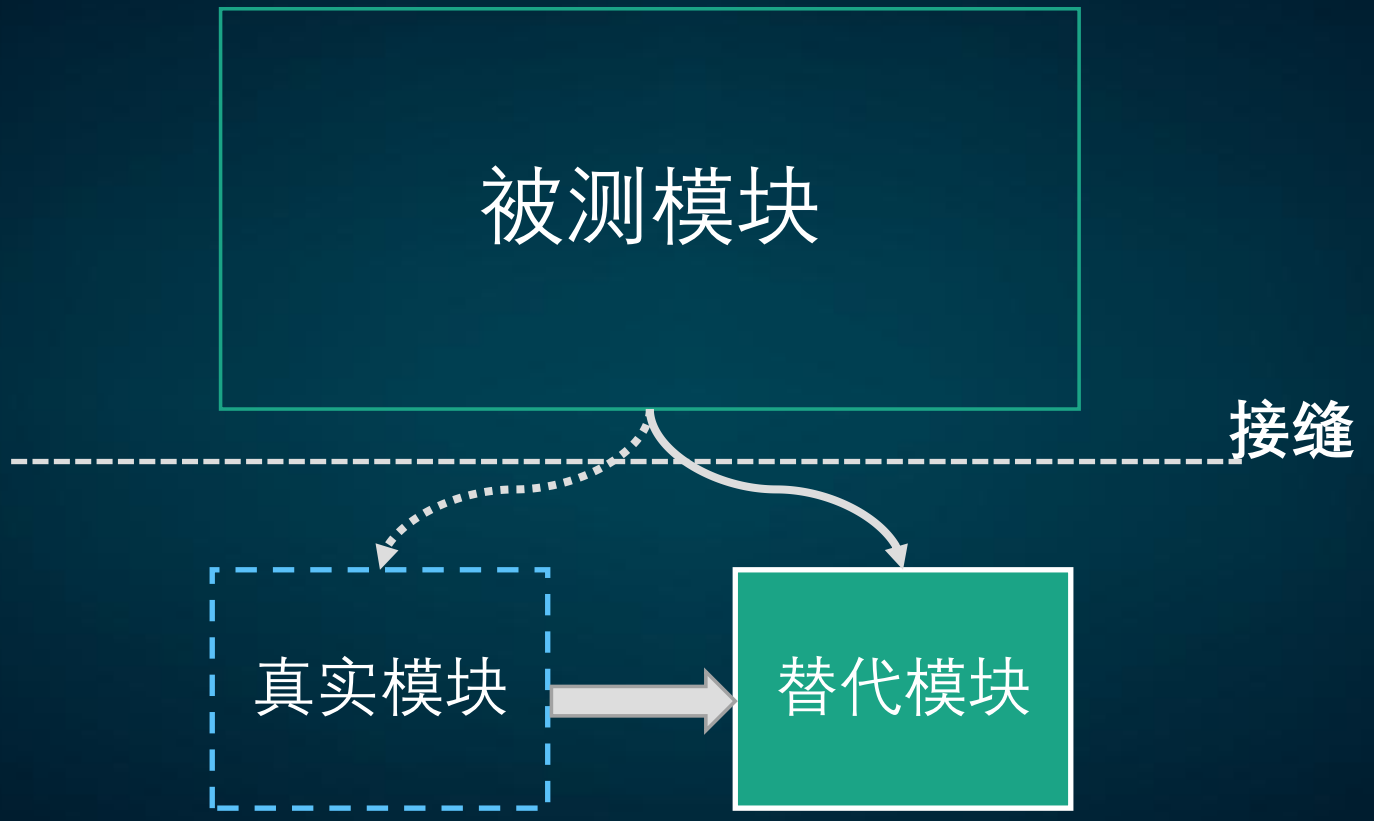
提高可控制性

提高观察性



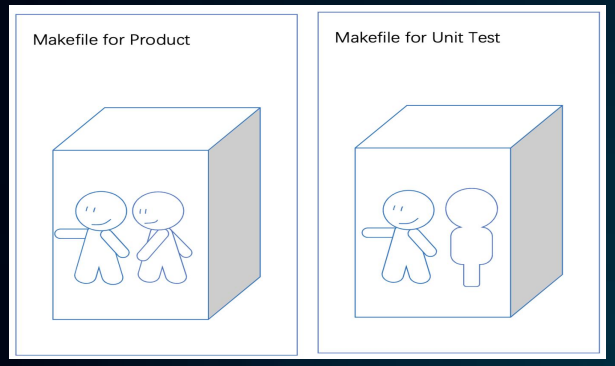
降低复杂度

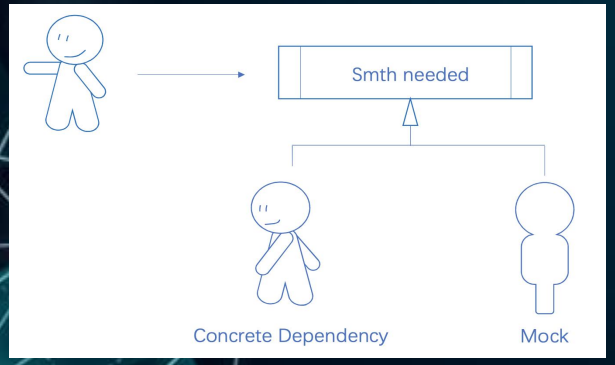
接缝

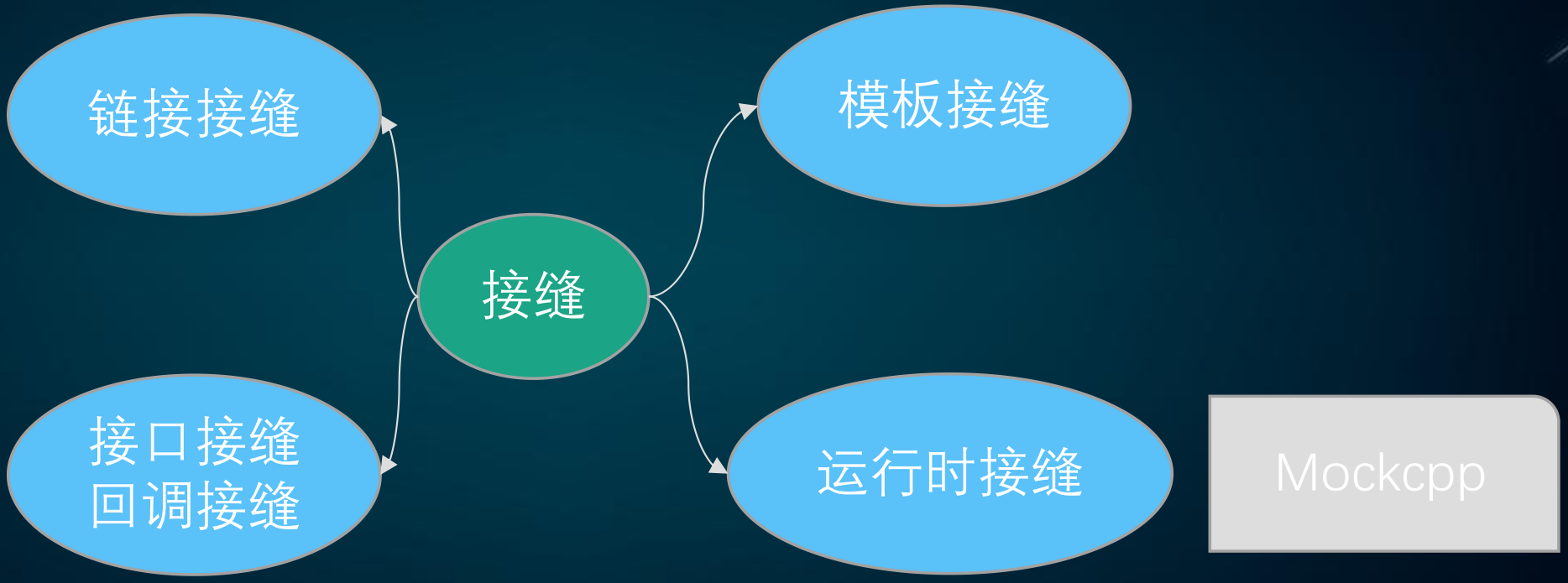


- 接缝是可以控制被测对象的行为而无需修改被测对象的地方。

C++的接缝技术


Makefile for Product
Makefile for Unit Test


Smth needed
Concrete Dependency
Mock


链接接缝
接口接缝
回调接缝
接缝
模板接缝
运行时接缝
Mockcpp

模板接缝

```
// A simple packet stream class.  None of its members is virtual.
class ConcretePacketStream {
 public:
  void AppendPacket(Packet* new_packet);
  const Packet* GetPacket(size_t packet_number) const;
  size_t NumberOfPackets() const;
  ...
};

// A mock packet stream class.  It inherits from no other, but defines
// GetPacket() and NumberOfPackets().
class MockPacketStream {
 public:
  MOCK_METHOD(const Packet*, GetPacket, (size_t packet_number), (const));
  MOCK_METHOD(size_t, NumberOfPackets, (), (const));
```

模板接缝

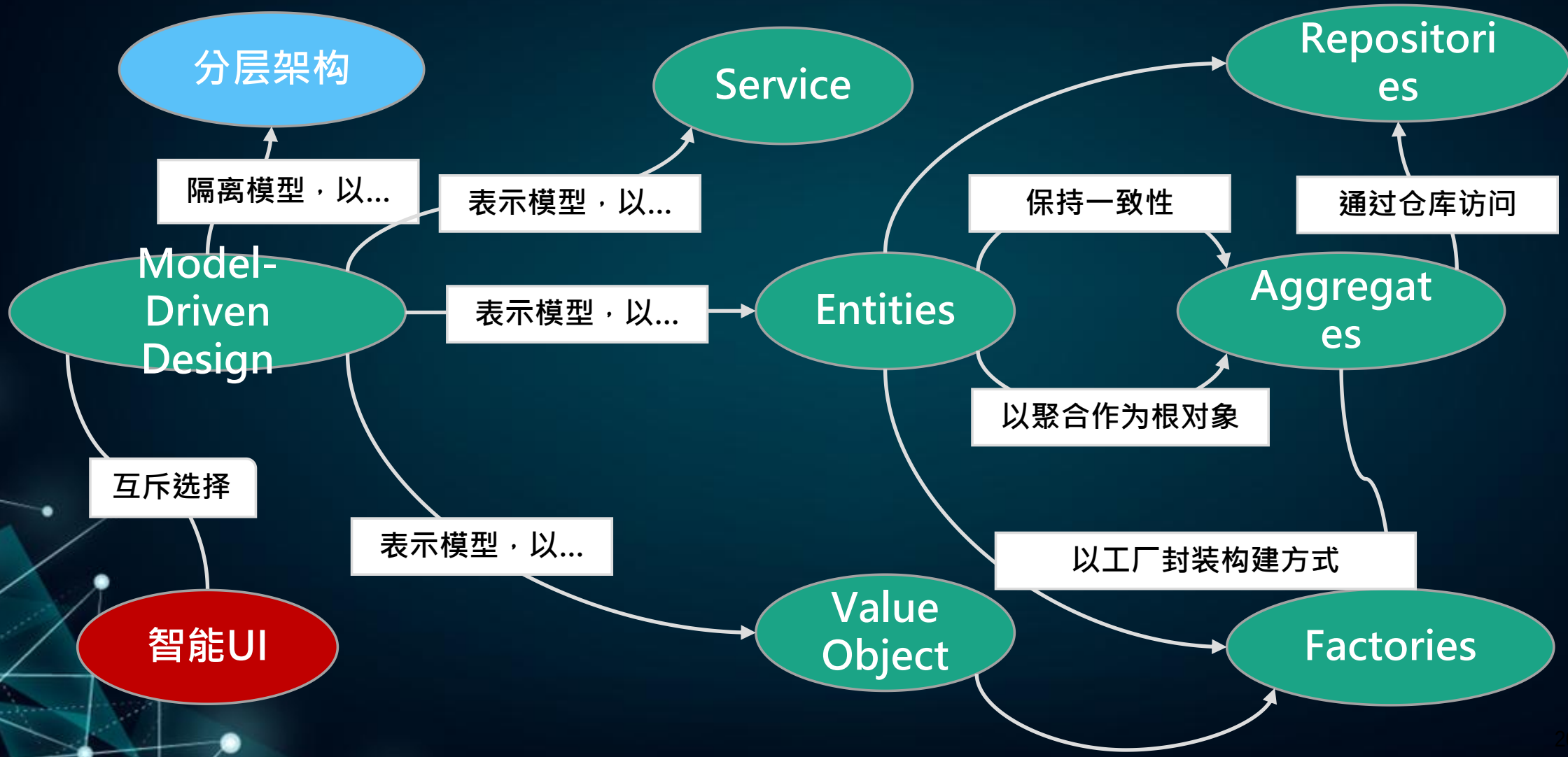
```
template <class PacketStream>
class PacketReader {
 public:
  void ReadPackets(PacketStream* stream, size_t packet_num);
};
```

```
MockPacketStream mock_stream;
EXPECT_CALL(mock_stream, ...)...;
.. set more expectations on mock_stream ...
PacketReader<MockPacketStream> reader(&mock_stream);
... exercise reader ...
```

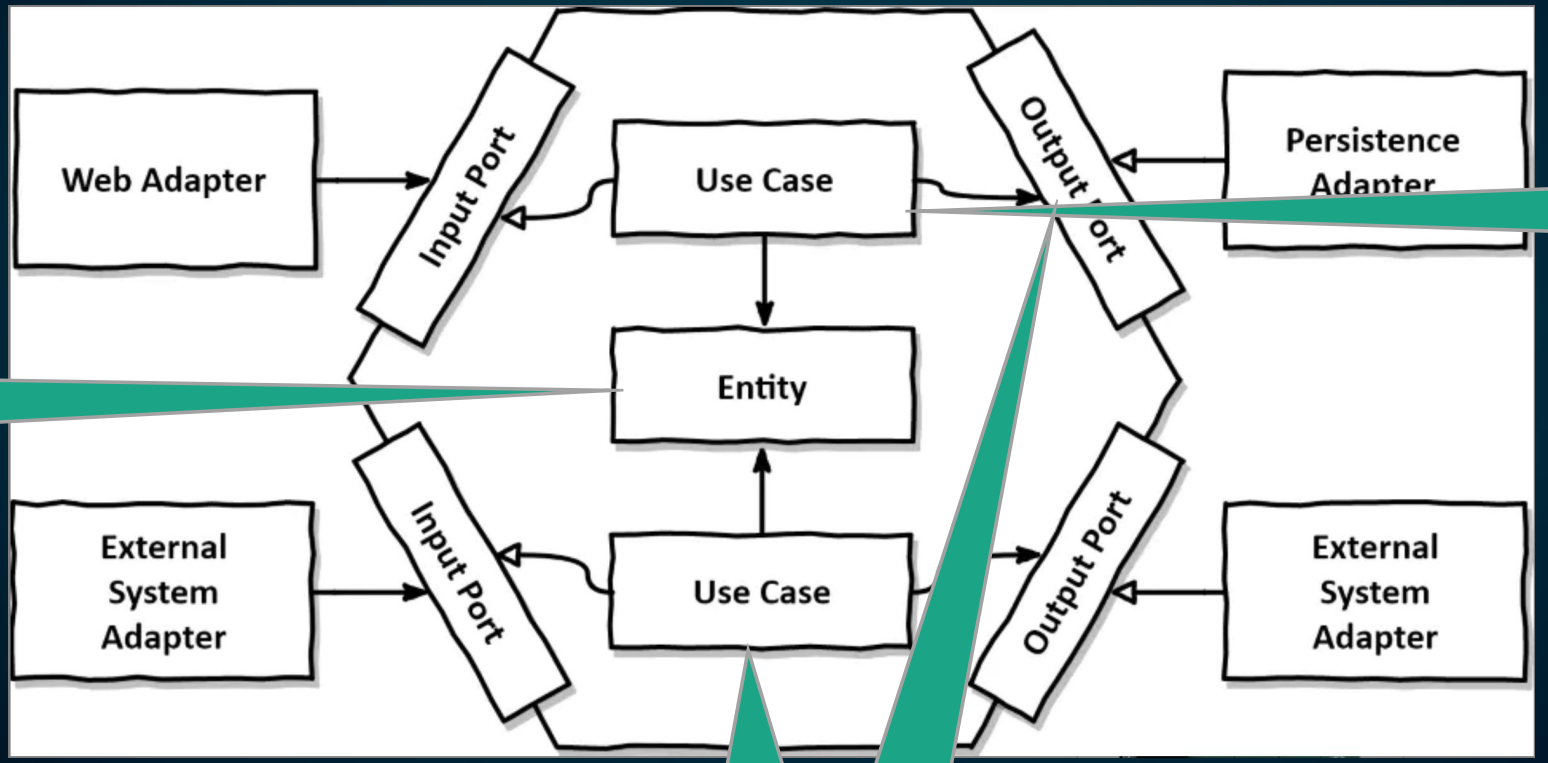
选择合适的接缝

- 模块之间遗留的c接口，可以采用链接接缝，或者使用mockcpp动态替换
- 需要动态修改行为的，使用抽象接口作为接缝
- 可以静态确定行为的、有性能要求的，可以采用concept作为接缝

DDD


分层架构
隔离模型，以...
Service
表示模型，以...
Repositories
保持一致性
通过仓库访问
Model-Driven Design
表示模型，以...
Entities
Aggregates
以聚合作为根对象
互斥选择
表示模型，以...
以工厂封装构建方式
智能UI
Value Object
Factories

DDD与可测性设计



关注点分离

可控性

观察点

可测性的坏味道

```
int foo(int a, int b, char e, string_view f) {
  // 预处理
  // ...
  bar(A{B{C{a, b}, D{e, f}}});
}
```

对象的创建责任和被测模块的
其他行为耦合在一起

提高可测性

```
int foo(const A &a) {
  // 预处理
  // ...
  bar(a);
}
```

将创建责任从被测模块分离出去

可测性的坏味道

- 构造函数太复杂
- 违反了迪米特法则，引入不必要的依赖
- 类做了太多的事，违反单一职责
- 单例模式的使用，到处使用全局状态

单例模式

```
class Publisher {
public:
  static Publisher& Instance();
  void Register(const Notify& notify) ;
  void Broadcast(int msg) ;
 private:
  Publisher() = default;
  std::vector<Notify> notify_list_;
};
```

单例模式的可测性分析

- 每个测试，开始执行前都会有一个预期的状态
- 每个测试执行后，都会影响全局状态
- 测试的执行并不总按照固定的顺序进行
- 对全局状态的测试会影响测试之间的独立性
- 最终总会在单例上引入Reset()之类的函数

```
TEST(PublisherTest, empty_publisher_should_broadcast_nothing){...}

TEST(PublisherTest, publisher_with_1_subscriber_should_send_to_notification_to_this_subscriber){...}

TEST(PublisherTest, publisher_with_2_subscribers_should_broadcast_notifications){...}
```

Special Interface模式

```
class Publisher {
public:
  static Publisher& Instance();
  void Reset();
  std::vector<Notify>& GetStatus()
const;
  void Register(const Notify& notify) ;
  void Broadcast(int msg) ;
 private:
  Publisher() = default;
  std::vector<Notify> notify_list_;
};
```

提高可测性：分离创建责任

```
class Publisher {
 public:
  void Register(const Notify& notify);
  void Broadcast(int msg) ;
private:
  std::vector<Notify> notify_list_;
};

Publisher& GetPublisher() {
  static Publisher publisher{};
  return publisher;
}
```

实体 Entity

工厂 Factory
仓库 Repository

多线程如何测试

- 数据竞争
- 线程调度不确定
- 用例构建复杂

线程控制点

```
void foo() {
    // 预处理
    // ...
    // 第一步
    inject_point("foo first step complete")
    // 后续步骤
}

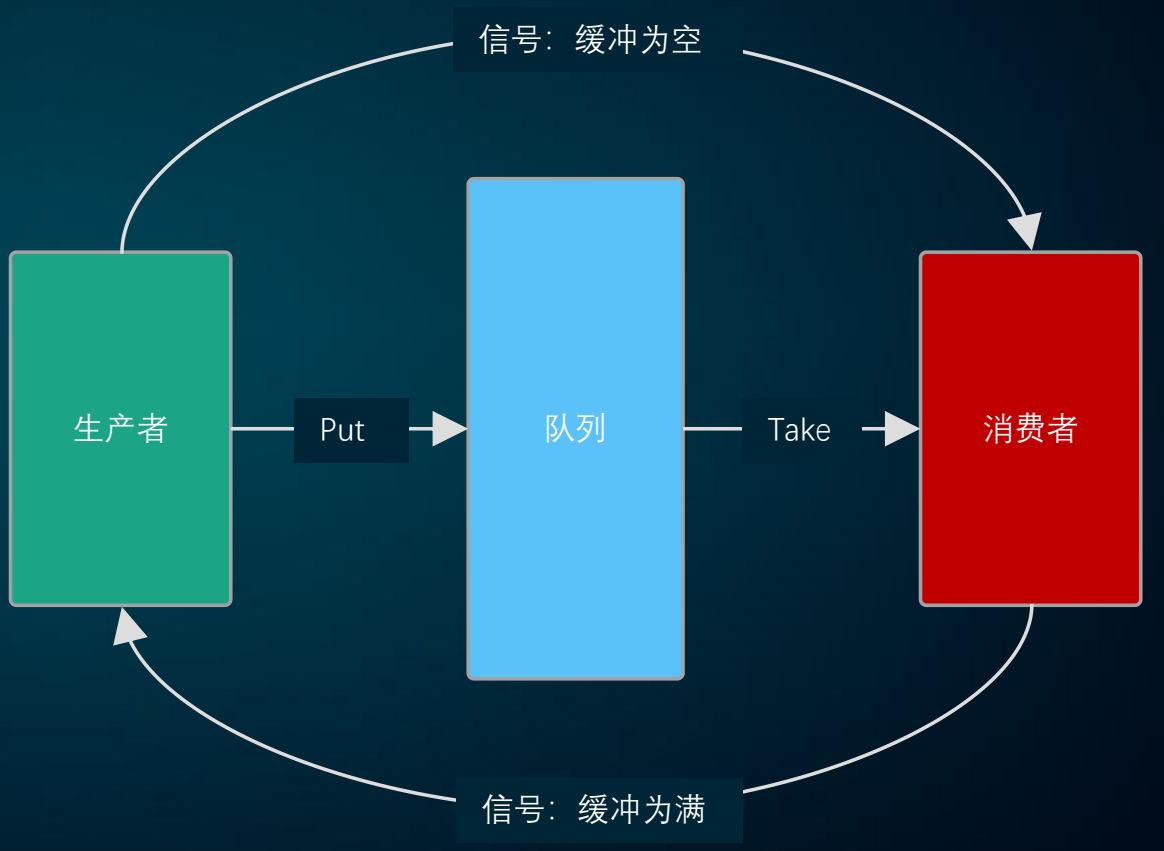
void bar() {
    // 第一步
    // ...
    // 第二步
    inject_point("bar second step start")
}
```

```
TEST(multi_thread_test, 2_thread_synchroize) {
    // arrange & act
    auto foo_thr = std::thread(foo);
    auto bar_thr = std::thread(bar);

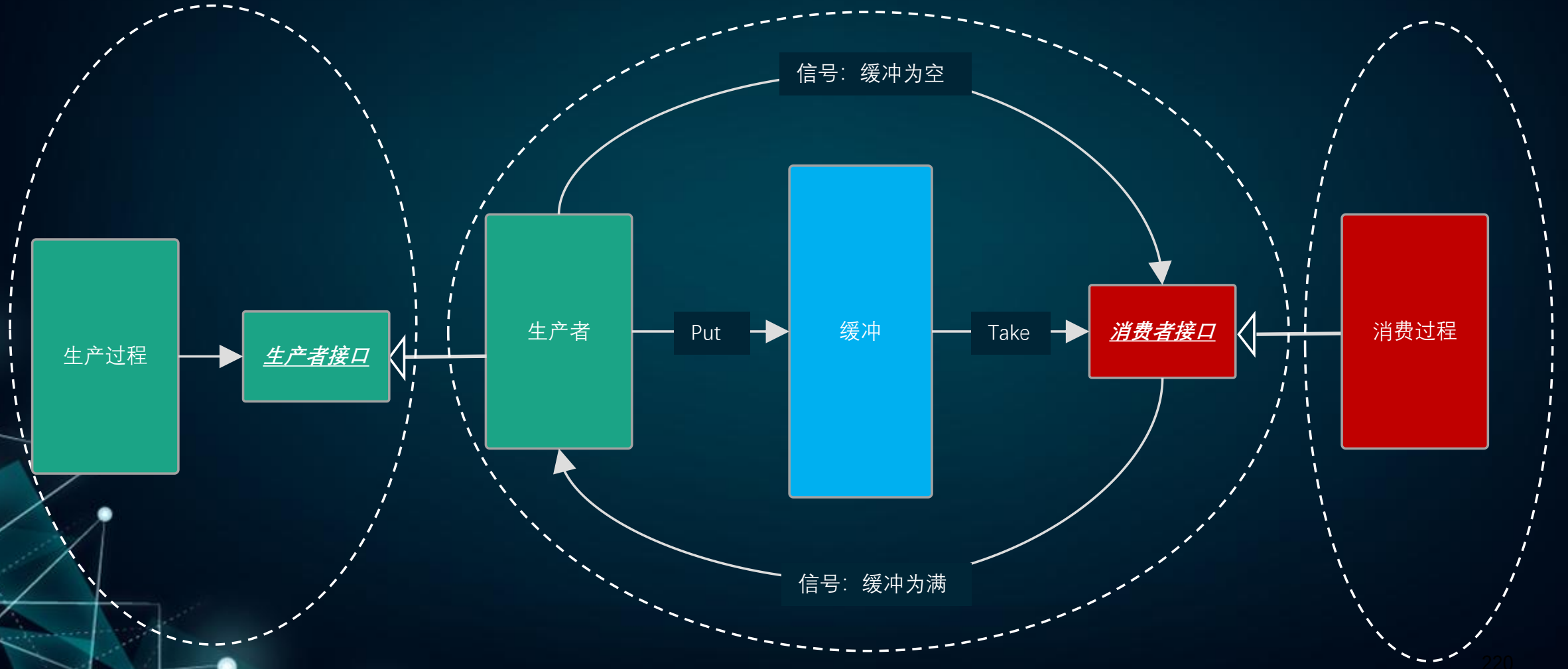
    ThreadController.Sync("foo first step complete", foo_thr.get_id())
                    .With("foo first step complete", bar_thr.get_id());
    // assert
    // ...
}
```

生产消费问题

- 生产者**产生**内容
- 消费者**消费**内容
- 当缓冲满时，生产者**等待**缓冲被消费
- 当缓冲空时，消费者**等待**缓冲被生产



分离关注点



03

测试用例设计

参数化测试

height	width	perimeter
5	8	26
1	4	10
10	30	80
...	...	...

```
TEST_P(RectangleTest,
       perimeter_of_rectangle_should_be_2_X_length_width) {
auto test_data = GetParam();
Rectangle rectangle{get<0>(test_data), get<1>(test_data)};
ASSERT_THAT(rectangle.perimeter(), Eq(get<2>(test_data)));
}

INSTANTIATE_TEST_SUITE_P(perimeter, RectangleTest,
        ::testing::Values(tuple(5, 8, 26),
                          tuple(1, 4, 10),
                          tuple(10, 30, 70)));
```

数据驱动测试

	A	B	C
1	Height	Width	Perimeter
2	5	8	26
3	1	4	10
4	10	30	80
5			
6			
7			
8			

数据 Data

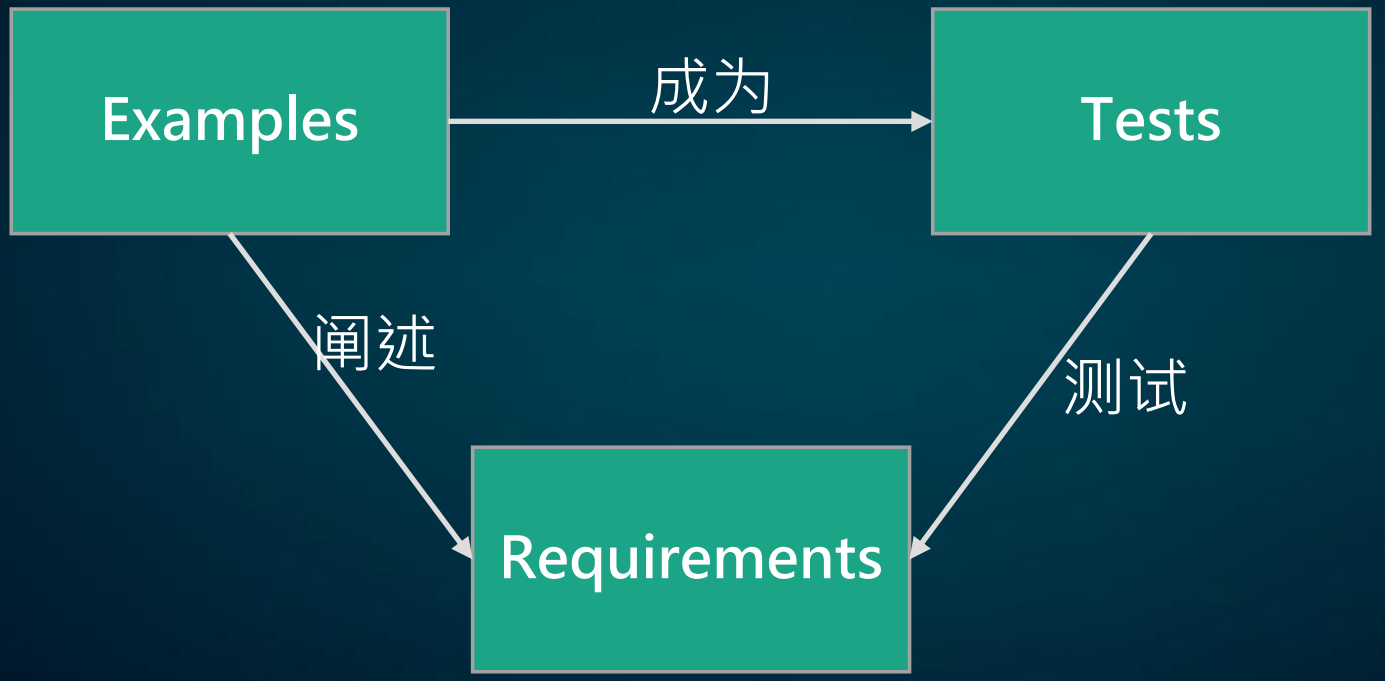
测试 Test

rectangle.param

```
//    width,    height,    perimeter
tuple(  5,      8,         26),
tuple(  1,      4,         10),
tuple(  10,     30,        80)
```

```
INSTANTIATE_TEST_SUITE_P(perimeter, RectangleTest,
        ::testing::Values(
#include "rectangle.param"
                ));
```

C++的实例化需求



一切皆数据

- 火星车：
  - 在地球上的控制中心为每一个漫步者设置初始位置，格式为x y orientation，例如：5 7 N
  - 控制中心向漫步者发送的一系列指令进行探测。
  - 漫步者可以理解的指令为：左转九十度，右转九十度，向前移动一米。

一切皆数据

	A	B	C
1	Origin Orientation	Instruction	Target Orientation
2	Orientation::N()	TurnRight	Orientation::E()
3	Orientation::N()	TurnLeft	Orientation::W()
4	Orientation::E()	TurnRight	Orientation::S()
5	Orientation::E()	TurnRight	Orientation::N()

```
Orientation TurnLeft(Orientation& origin) {
    return origin.turnLeft()
}

Orientation TurnRight(Orientation& origin) {
    return origin.turnRight()
}

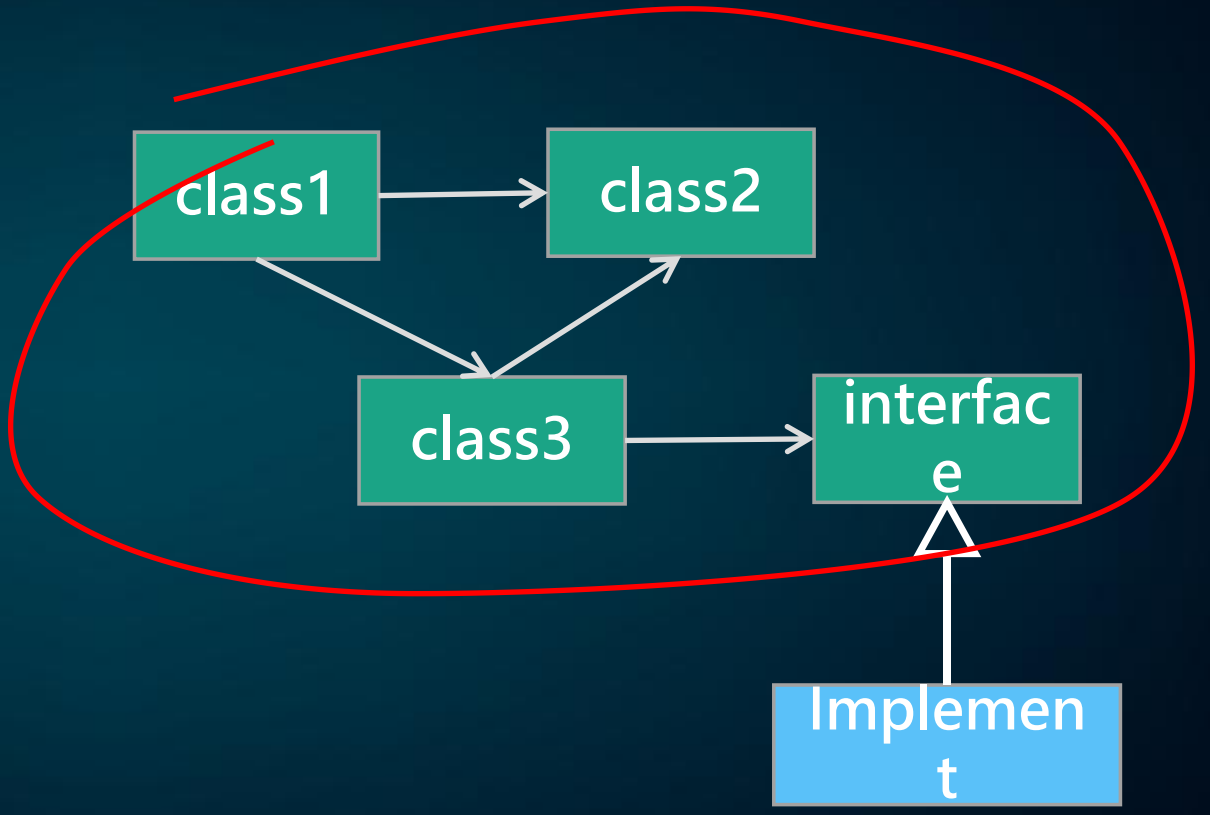
INSTANTIATE_TEST_SUITE_P(left_right_data, OrientationTurnTest,
                         Values(
                                 std::tuple(Orientation::N(), TurnRight, Orientation::E()),
                                 std::tuple(Orientation::E(), TurnLeft,  Orientation::N()),
                                 std::tuple(Orientation::E(), TurnRight, Orientation::S()),
                                 std::tuple(Orientation::S(), TurnLeft, Orientation::E())
                                ));
```

适用于测试数据维度较高的场景

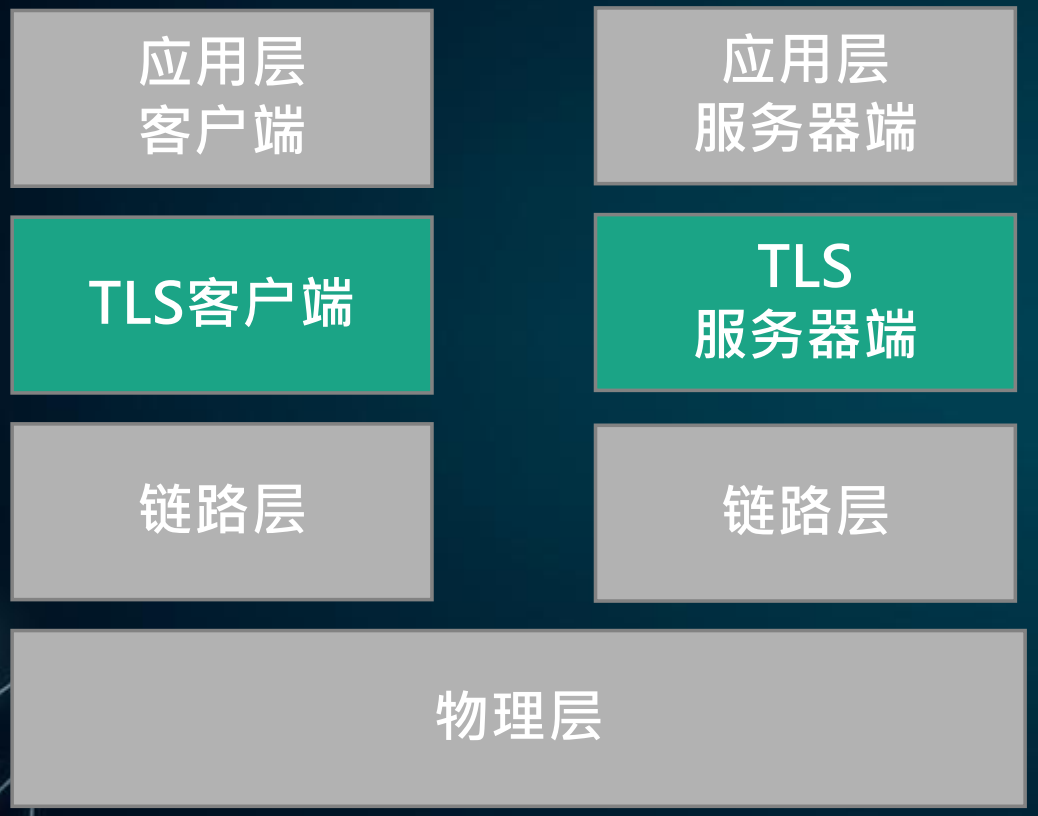
Origin Orientation	Instruction	Target Orientation	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Colu
Orientation::N()	TurnRight	Orientation::E()	Orientation::	Orientation::	Orientation::	Orientation::	Orientation::	E()				
Orientation::N()	TurnLeft	Orientation::W()	Orientation::	Orientation::	Orientation::	Orientation::	Orientation::	W()				
Orientation::E()	TurnRight	Orientation::S()	Orientation::	Orientation::	Orientation::	Orientation::	Orientation::	S()				
Orientation::E()	TurnRight	Orientation::N()	Orientation::	Orientation::	Orientation::	Orientation::	Orientation::	N()				

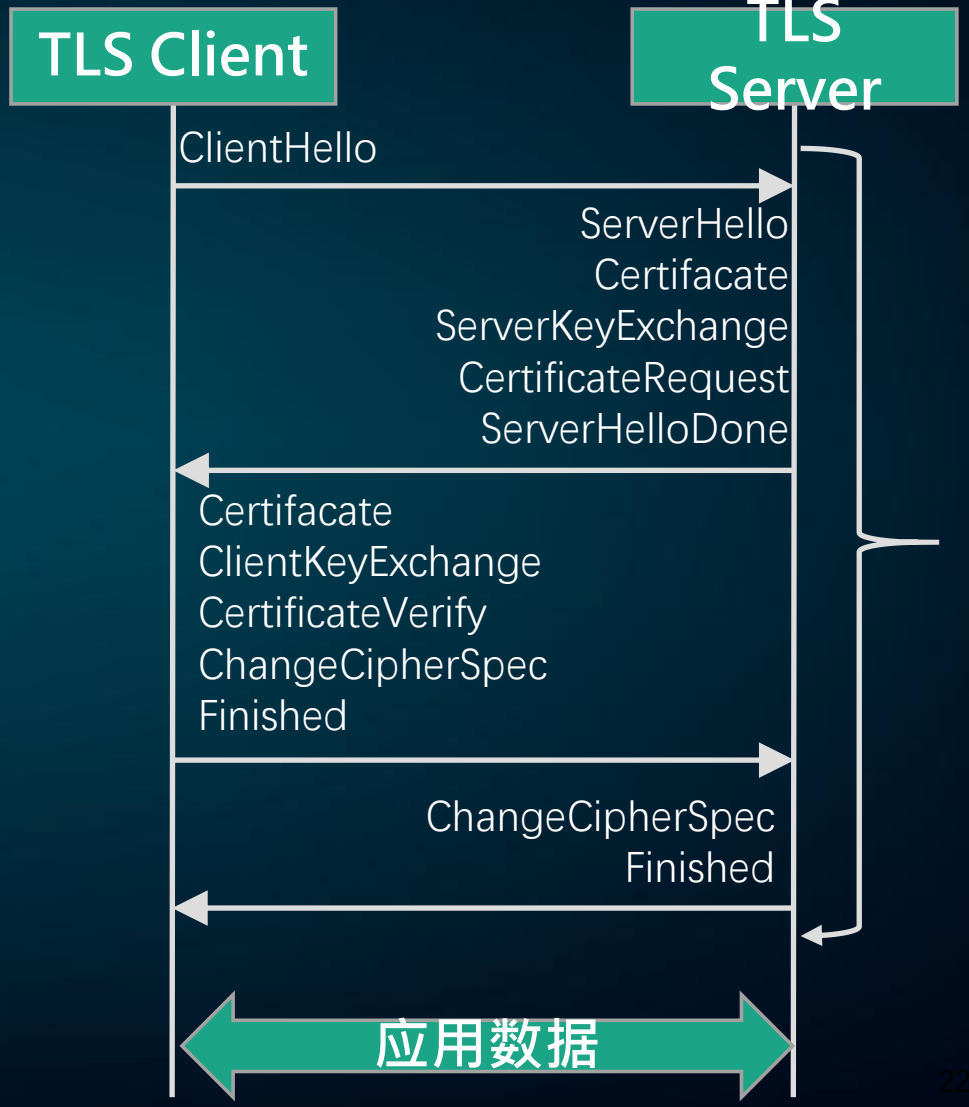
测试对象的选择

- 一般来说，遵循设计原则的系统，最好的测试对象是模块

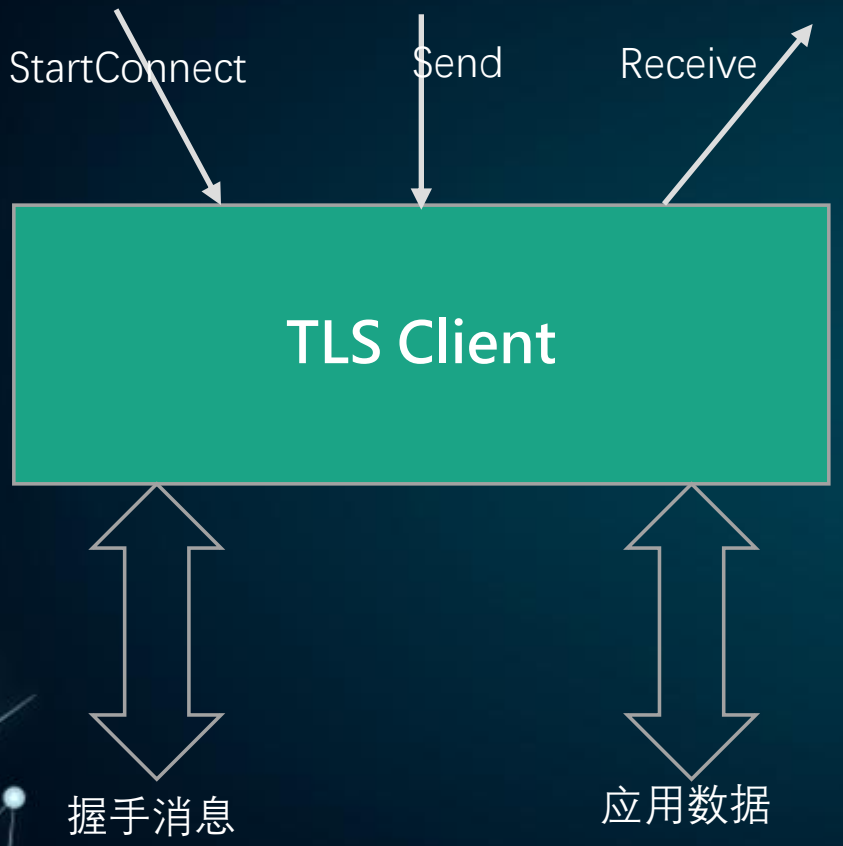


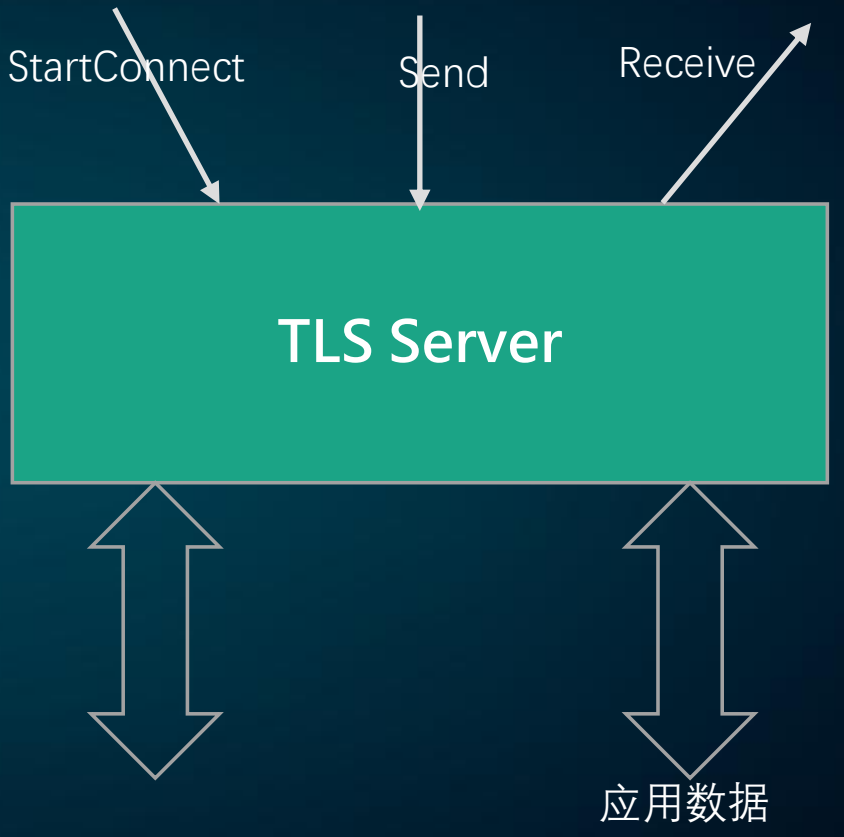
通信设备的例子


应用层
客户端
应用层
服务器端
TLS客户端
TLS
服务器端
链路层
链路层
物理层

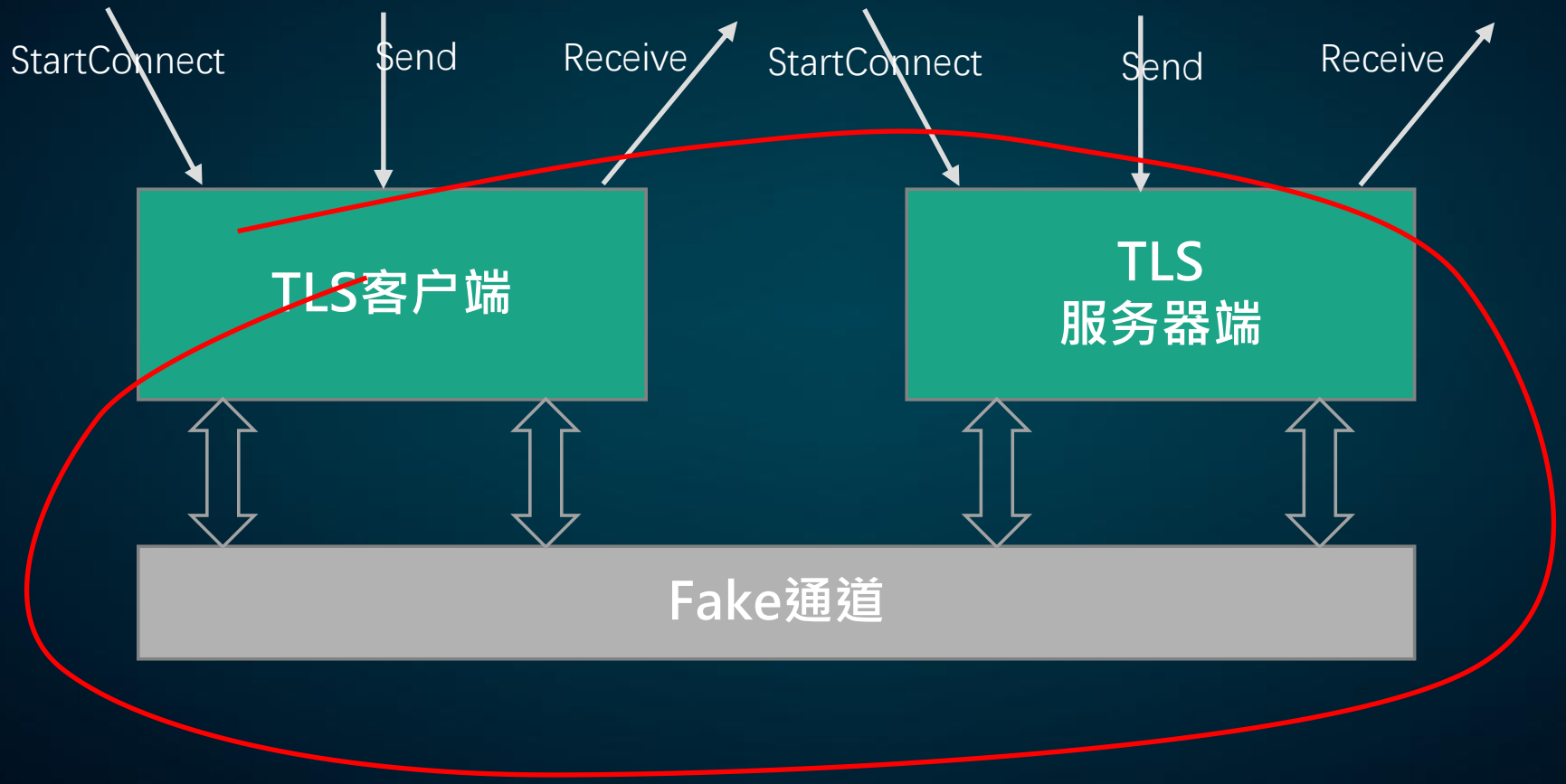

TLS Client
TLS Server
ClientHello
ServerHello
Certifacate
ServerKeyExchange
CertificateRequest
ServerHelloDone
Certifacate
ClientKeyExchange
CertificateVerify
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
应用数据

怎么测试


StartConnect
Send
Receive
TLS Client
握手消息
应用数据


StartConnect
Send
Receive
TLS Server
应用数据

测试对象的选择


StartConnect
Send
Receive
StartConnect
Send
Receive
TLS客户端
TLS
服务器端
Fake通道

谢谢观看

CPP-Summit 2022



吴咏炜

Boolan首席咨询师

国内知名 C++专家，曾任英特尔亚太研发中心资深系统架构师，近 30 年 C/C++系统级软件开发和架构经验。专注于 C/C++ 语言(包括 C++98/C++11/14/17/20)、软件架构、性能优化、设计模式和代码重用。对于精炼、易于维护的代码和架构有着不懈的追求，对开源平台(GNU/Linux)有深入的理解。长期担任资深技术教练，涉及 C++、软件架构、安全软件开发、开源软件等多方面。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

吴咏炜
博览首席咨询师

C++ 编译期编程的过去、现在和未来

议程

1 简介

2 一点点历史

3 模板元编程

4 constexpr

5 变参模板和静态反射

1

简介

什么是编译期编程？

运行期 vs 编译期

普通编程（运行期）

- 代码在**运行**程序的时候执行

编译期编程

- 代码在**编译**程序的时候执行

为什么要使用编译期编程？

奇技淫巧？



制式武器 vs 自定义武器

演讲本身

- 示例为主
  - 有完整可编译的代码
- 展示可能性和方向
- 不讲解的内容
  - 完整可用的库
  - 完整的知识点教授
  - 技巧和陷阱

2

一点点历史

非类型模板参数

```
template<class T, int size> class buffer;
buffer<char*, 1000> glob;
```

(Stroustrup, 1988)

C++ is a powerful enough language—the first such language in our experience—to allow the construction of generic programming components that combine mathematical *precision*, *beauty*, and *abstractness* with the *efficiency* of non-generic hand-crafted code.

—Alex Stepanov

(Stroustrup, 1994)

模板的“滥用”

```
// Prime number computation by Erwin Unruh
template <int i> struct D { D(void*); operator int(); };

template <int p, int i> struct is_prime {
    enum { prim = (p%i) && is_prime<(i > 2 ? p : 0), i -1> :: prim };
    };

template < int i > struct Prime_print {
    Prime_print<i-1> a;
    enum { prim = is_prime<i, i-1>::prim };
    void f() { D<i> d = prim; }
    };

struct is_prime<0,0> { enum {prim=1}; };
struct is_prime<0,1> { enum {prim=1}; };
struct Prime_print<2> { enum {prim = 1}; void f() { D<2> d = prim; } };
#ifndef LAST
#define LAST 10
#endif
main () {
    Prime_print<LAST> a;
    }
```

(Unruh, 1994)

标准 C++ 下可“工作”的版本

```
// Prime number computation by Erwin Unruh

template <int p, int i> struct is_prime {
  enum { prim = (p==2) || (p%i) && is_prime<(i>2?p:0), i-1> :: prim };
};

template<> struct is_prime<0,0> { enum {prim=1}; };
template<> struct is_prime<0,1> { enum {prim=1}; };

template <int i> struct D { D(void*); };

template <int i> struct Prime_print {
  Prime_print<i-1> a;
  enum { prim = is_prime<i, i-1>::prim };
  void f() { D<i> d = prim ? 1 : 0; a.f();}
};

template<> struct Prime_print<1> {
  enum {prim=0};
  void f() { D<1> d = prim ? 1 : 0; };
};

int main() {
  Prime_print<18> a;
  a.f();
}
```

某编译器下的输出

```
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<17>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<13>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<11>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<7>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<5>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<3>'
...
unruh.cpp:15:19: error: no viable conversion from 'int' to 'D<2>'
...
```

https://godbolt.org/z/o9G9M8Mx9

3

模板元编程

阶乘 – 数学定义

$$n! = n \cdot (n-1)!$$
$$0! = 1$$

阶乘 – 模板元编程

```cpp
template <int N>
struct factorial {
    static const int value = N * factorial<N - 1>::value;
};

template <>
struct factorial<0> {
    static const int value = 1;
};
```

“函数”的定义

阶乘 – 模板元编程的展开

```
factorial<3>::value
```



```
3 * factorial<2>::value
```



```
3 * (2 * factorial<1>::value)
```



```
3 * (2 * (1 * factorial<0>::value))
```



```
3 * (2 * (1 * 1))
```

“函数”的调用

C++ 里的函数式编程

- 一个类模板表达了一个“函数”
- 模板的实例化相当于“纯函数调用”
- 一个“变量”（类模板里的静态成员变量）只能赋值一次
- 实例化的结果在下面的编译过程中被记住，相当于记忆化
- ……

编译期条件

```
template <bool Condition, typename Then, typename Else>
struct conditional;

template <typename Then, typename Else>
struct conditional<true, Then, Else> {
    using type = Then;
};

template <typename Then, typename Else>
struct conditional<false, Then, Else> {
    using type = Else;
};
```

编译期循环

```
template <bool Condition, typename Body>
struct loop_result;

template <typename Body>
struct loop_result<true, Body> {
    using type = typename loop_result<Body::next_type::condition,
                                      typename Body::next_type>::type;
};

template <typename Body>
struct loop_result<false, Body> {
    using type = typename Body::type;
};

template <typename Body>
struct loop {
    using type = typename loop_result<Body::condition, Body>::type;
};
```

编译期循环 – 用继承简化

```
template <bool Condition, typename Body>
struct loop_result;

template <typename Body>
struct loop_result<true, Body>
    : loop_result<Body::next_type::condition,
                  typename Body::next_type> {};

template <typename Body>
struct loop_result<false, Body> : Body {};

template <typename Body>
struct loop : loop_result<Body::condition, Body> {};
```

阶乘 – 改用循环

```
template <int N, int Last, int Result>
struct factorial_loop {
    static const bool condition = (N <= Last);
    using type = integral_constant<int, Result>;
    using next_type = factorial_loop<N + 1, Last, Result * N>;
};

template <int N>
struct factorial : loop<factorial_loop<1, N, 1>>::type {};
```

汇编结果

```
printf("%d\n",
       factorial<10>::value);
```

```
.LC0:
        .string "%d\n"
main:
        push    rbp
        mov     rbp, rsp
        mov     esi, 3628800
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

编译期计算的结果

https://godbolt.org/z/j59ErdnTj

https://godbolt.org/z/Ea8xcff7e

模板元编程之质数计算

算法

```
primesTo n = sieve [2..n]
             where
             sieve (x:xs) = x:(sieve $
                               filter (\a -> a `mod` x /= 0)
                                      xs)

             sieve [] = []
```

1. 产生 2..*n* 的列表

2. 列表首项是质数

3. 剩余项里过滤掉首项除得尽的数

4. 再递归筛

5. 空列表即递归终结

工具 – 编译期列表

```
struct nil {};

template <typename Head, typename Tail = nil>
struct list {};
```

```
// 可理解成
struct list {
    any head;
    list* tail{nullptr};
};
```

工具 – 编译期过滤

```
template <template <typename> class Pred, typename List>
struct filter;

template <template <typename> class Pred, typename Head, typename Tail>
struct filter<Pred, list<Head, Tail> > {
    typedef
        typename conditional<Pred<Head>::value,
                             list<Head, typename filter<Pred, Tail>::type>,
                             typename filter<Pred, Tail>::type>::type type;
};

template <template <typename> class Pred>
struct filter<Pred, nil> {
    typedef nil type;
};
```

类模板声明（原型）

条件分支

特化（主要过滤式）

递归调用

特化（终结条件）

工具 – 整数和类型的相互转换

```cpp
template <typename T, T Val>
struct integral_constant {
    static const T value = Val;
    typedef T value_type;
    typedef integral_constant type;
};

// 类型
integral_constant<int, 42>
integral_constant<bool, true>
// 值
integral_constant<int, 42>::value    // 42
integral_constant<bool, true>::value // true
```

工具 – 产生序列

主模板

```
template <int First, int Last>
struct range {
    typedef list<integral_constant<int, First>,
                 typename range<First + 1, Last>::type>
        type;
};
```

特化（终结条件）

```
template <int Last>
struct range<Last, Last> {
    typedef nil type;
};
```

筛法求质数

```
template <typename T>
struct sieve_prime;

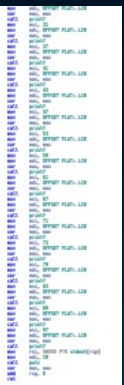
template <typename Head, typename Tail>
struct sieve_prime<list<Head, Tail> > {
    template <typename T>
    struct is_not_divisible
        : integral_constant<bool, (T::value % Head::value) != 0> {};

    typedef list<Head, typename sieve_prime<
                           typename filter<is_not_divisible, Tail>::type>::type>
        type;
};

template <>
struct sieve_prime<nil> {
    typedef nil type;
};

        template <int N>
        struct primes_to : sieve_prime<typename range<2, N + 1>::type>::type {};
```

```
 5        mov     esi, 2
 6        mov     edi, OFFSET FLAT:.LC0
 7        xor     eax, eax
 8        call    printf
 9        mov     esi, 3
10        mov     edi, OFFSET FLAT:.LC0
11        xor     eax, eax
12        call    printf
13        mov     esi, 5
14        mov     edi, OFFSET FLAT:.LC0
15        xor     eax, eax
16        call    printf
```



Output (0/0) x86-64 gcc 11.2 - *1862ms (81029B) ~3668 lines filtered*

Output of x86-64 gcc 11.2 (Compiler #1)

A Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73  79  83  89  97
```

https://godbolt.org/z/xE7MKca4s

C++ 模板是图灵完全的！

求质数的等效 Scheme 代码

```
(define (sieve-prime lst)
  (cond
    [(null? lst) '()]
    [else (let ([n (car lst)])
            (let ([is-not-divisible
                   (lambda (m) (not (= (remainder m n) 0)))])
              (cons n (sieve-prime (filter is-not-divisible
                                           (cdr lst))))))]))

(define (primes-to n) (sieve-prime (range 2 (+ n 1))))
```

模板元编程的问题

难写，“土”

编译慢（慢于 Haskell/Scheme 代码的运行速度）

可能耗费大量内存

容易让编译器崩溃，甚至让系统失去响应

4

constexpr

const 的问题之一

```cpp
template <int N>
struct factorial {
  static const int value =
    N * factorial<N - 1>::value;
};

template <>
struct factorial<0> {
  static const int value = 1;
};
```

```cpp
template <typename T>
void print_value(const T& value)
{
  …
}

print_value(factorial<10>::value);
```

undefined reference to `factorial<10>::value'

https://godbolt.org/z/TW4GTKPYc

constexpr 变量

```cpp
template <int N>
struct factorial {
  static constexpr int value =
    N * factorial<N - 1>::value;
};

template <>
struct factorial<0> {
  static constexpr int value = 1;
};
```

```cpp
template <typename T>
void print_value(const T& value)
{
  …
}

print_value(factorial<10>::value);
```

https://godbolt.org/z/c3Gh13eWb

C++11 的 constexpr 函数

```
constexpr int factorial(int n)
{
    return n == 0 ? 1 : n * factorial(n - 1);
}
```

https://godbolt.org/z/8Gfjxx8fv

C++14 的 constexpr 函数

```
constexpr int factorial(int n)
{
    int result = 1;
    for (int i = 2; i <= n; ++i) {
        result *= i;
    }
    return result;
}
```

https://godbolt.org/z/3PE43PTn4

C++17 的编译期筛子

array 支持 constexpr 方法

```
template <int N>
constexpr auto sieve_prime()
{
    array<bool, N + 1> sieve{};
    for (int i = 2; i <= N; ++i) {
        sieve[i] = true;
    }
    for (int p = 2; p * p <= N; p++) {
        if (sieve[p]) {
            for (int i = p * p; i <= N; i += p) {
                sieve[i] = false;
            }
        }
    }
    return sieve;
}
```

编译期数质数（C++20）

```
template <size_t N>
constexpr size_t prime_count(const array<bool, N>& sieve)
{
    return count(cbegin(sieve), cend(sieve), true);
}
```

count 是 constexpr 函数

编译期数质数（C++17）

```
template <size_t N>
constexpr size_t prime_count(const array<bool, N>& sieve)
{
    size_t count = 0;
    for (size_t i = 2; i < sieve.size(); ++i) {
        if (sieve[i]) {
            ++count;
        }
    }
    return count;
}
```

结果转为 array

```
template <int N>
constexpr auto get_prime_array()
{
    constexpr auto sieve = sieve_prime<N>();
    array<int, prime_count(sieve)> result{};
    for (size_t i = 2, j = 0; i < sieve.size(); ++i) {
        if (sieve[i]) {
            result[j] = i;
            ++j;
        }
    }
    return result;
}
```

```
4           .long   2
5           .long   3
6           .long   5
7           .long   7
8           .long   11
9           .long   13
10          .long   17
```

Output (0/0) x86-64 gcc 11.2 - *956ms (76770B) ~4228 lines filtered*

Output of x86-64 gcc 11.2 (Compiler #1)

A Wrap lines Select all

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67  71  73  79  83  89  97  101
  103  107  109  113  127  131  137  139  149  151  157  163  167  173  179  181  191  193
```

https://godbolt.org/z/hjezWEf7v

编译性能

模板元编程

N = 1000

- GCC：约 1.2 秒
- Clang：编译器崩溃
- MSVC：fatal error C1202

constexpr 函数

N = 10000

- GCC：约 0.7 秒
- Clang：约 0.8 秒
- MSVC：约 1.3 秒

一些限制

- 编译期常量只能作为模板参数传递
  - 函数参数在函数里永远不能用作编译期常量
- C++20 之前 vector/string 完全不能在 constexpr 函数里使用
  - 不允许任何动态内存分配
- C++20 开始 vector/string 可以在 constexpr 函数里临时使用
  - 可以有动态内存分配；但函数返回前必须完全释放
  - 不能传递给运行期；不能声明为 constexpr 变量
  - 编译期用 vector 筛子求质数示例：https://godbolt.org/z/6c833fE4r

5

变参模板和静态反射

变参模板的主要用法

- 转发不定数量的参数到其他函数（通常结合转发引用）
- 通过递归调用或折叠表达式来对参数进行遍历处理

折叠表达式

```
template <typename... Args>
constexpr bool is_any_null(const Args&... args)
{
    return (... || (args == nullptr));
}
```

对反射结构体的编译期遍历

```cpp
template <typename T, typename F, size_t... Is>
constexpr void forEachImpl(T&& obj, F&& f, std::index_sequence<Is...>)
{
    using TDecay = std::decay_t<T>;
    (void(f(typename TDecay::template _field<T, Is>(obj).name(),
            typename TDecay::template _field<T, Is>(obj).value())),
     ...);
}

template <typename T, typename F>
constexpr void forEach(T&& obj, F&& f)
{
    using TDecay = std::decay_t<T>;
    forEachImpl(std::forward<T>(obj), std::forward<F>(f),
                std::make_index_sequence<TDecay::_field_count>{});
}
```

(罗能, 2020)

打印反射结构体

```cpp
template <typename T>
void dumpObj(const T& obj, std::ostream& os = std::cout,
             const char* fieldName = "", int depth = 0)
{
    if constexpr (IsReflected_v<T>) {
        os << indent(depth) << fieldName << (*fieldName ? ": {\n" : "{\n");
        forEach(obj, [depth, &os](const char* fieldName, auto&& value) {
            dumpObj(value, os, fieldName, depth + 1);
        });
        os << indent(depth) << "}" << (depth == 0 ? "\n" : ",\n");
    } else {
        os << indent(depth) << fieldName << ": " << obj << ",\n";
    }
}
```

dumpObj 的输出效果

```
DEFINE_STRUCT(
    Point,
    (double) x,
    (double) y);

DEFINE_STRUCT(
    Rect,
    (Point) p1,
    (Point) p2,
    (uint32_t) color);

dumpObj(Rect{
    {1.2, 3.4},
    {5.6, 7.8},
    12345678,
});
```

```
{
    p1: {
        x: 1.2,
        y: 3.4,
    },
    p2: {
        x: 5.6,
        y: 7.8,
    },
    color: 12345678,
}
```

https://godbolt.org/z/384af5xc9

静态反射

- 标准中尚未支持
- 目前需要通过宏和编译期编程技巧来实现
- 编译时静态展开，高性能！

将来的静态反射？

```
template <typename T>
void dumpObj(const T& obj, std::ostream& os = std::cout,
             const char* fieldName = "", int depth = 0)
{
    if constexpr (std::is_class_v<T>) {
        os << indent(depth) << fieldName << (*fieldName ? ": {\n" : "{\n");
        template for (constexpr meta::info member :
                      meta::members_of(^T, meta::is_nonstatic_data_member)) {
            dumpObj(obj.[:member:], os, meta::name_of(member), depth + 1);
        }
        os << indent(depth) << "}" << (depth == 0 ? "\n" : ",\n");
    } else {
        os << indent(depth) << fieldName << ": " << obj << ",\n";
    }
}
```

反射

编译期遍历

逆反射

https://cppx.godbolt.org/z/88vvchrna

总结

- 编译期编程提供了在编译期进行推理、计算的能力
- 模板元编程在 C++ 发展的早期提供编译期编程的能力
- 现代 C++ 提供了更多编译期编程的特性，更加易用
- 未来 C++ 可能提供更多编译期的支持功能，如静态反射
  - 希望大约会是在 26……

参考资料

参考资料

Wyatt Childers et al. 2022. Scalable Reflection in C++ (P1240R2). http://wg21.link/p1240r2

Bjarne Stroustrup. 1988. Parameterized Types for C++. Proc. USENIX C++ Conference, Denver, CO. Also, USENIX Computer Systems, Vol 2 No 1. Winter 1989. https://www.usenix.org/legacy/publications/compsystems/1989/win_stroustrup.pdf

Bjarne Stroustrup. 1994. *The Design and Evolution of C++*. Addison-Wesley.

Andrew Sutton et al. 2020. Expansion Statements (P1306R1). http://wg21.link/p1306r1

Andrew Sutton et al. 2021. The Syntax of Static Reflection" (P2320R0). http://wg21.link/p2320r0

Erwin Unruh. 1994. Primzahlen. http://www.erwin-unruh.de/primorig.html

Todd L. Veldhuizen. 2003. C++ Templates are Turing Complete. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.3670

罗能. 2020. 如何优雅的实现 C++ 编译期静态反射. https://netcan.github.io/2020/08/01/%E5%A6%82%E4%BD%95%E4%BC%98%E9%9B%85%E7%9A%84%E5%AE%9E%E7%8E%B0C-%E7%BC%96%E8%AF%91%E6%9C%9F%E9%9D%99%E6%80%81%E5%8F%8D%E5%B0%84/

胶片网址

https://github.com/adah1972/cpp_summit_2022

谢谢观看



尉刚强

资深软件技术顾问

拥有十多年大型软件行业从业经验，曾先后在嵌入式平台系统、无线通信系统、大数据、人工智能对话平台、SAAS服务等领域中从事核心系统架构设计，持续性能优化与工程自动化提效工作。全栈软件工程师，精通C/C++、Java、Scala、NodeJS，Ruby和Python等各种前后端技术和语言，并对不同语言的实现模式、性能优化、构建机制有深入研究。

CPP-Summit

尉刚强
资深软件技术顾问

Modern C++
基于性能的重构优化



尉刚强

资深软件技术顾问

全栈软件工程师

咨询领域：

- 智能汽车行业
- OS/芯片相关的系统级软件
- AI平台与引擎

咨询业务：

- 系统级的持续性能优化与业务重构
- 工程构建与自动化测试改进，效率提升

专栏：

- 极客时间 《性能优化高手课》

议程

1 **C++系统软件性能现状与挑战**

性能现状、行业认识、优化方法、优化挑战

2 **性能优化重构工程**

数据获取、性能测试、数据分析

3 **C++性能优化重构实战详解**

OO生命周期、并发设计、ModernC++、标准库等

01

C++软件性能现状与挑战

软件性能优化本质

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	–	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

本质在于减少指令数：

1. Python2执行大约7小时；
2. Java相对Python提高了10.8倍；
3. C相对Java提高了4.4被，比Python提高47倍；

根据硬件架构定制软件实现：

1. 并行化，充分利用18核提升7.8倍
2. 优化并行任务及缓存策略，提升18.4倍
3. 使用矢量化，提升3.5倍
4. 使用Intel AVX指令级加速，提升2.7倍；相对原始版本，提升6.28万倍

开发效率与性能的平衡：

1. V7的C语言实现代码量，是V1的Python实现的20倍；
2. 软件实现与硬件架构耦合更严重，对程序员能力要求很高。
3. C++语言更好的支撑软件开发效率与性能的平衡

C++软件系统性能现状

- 系统业务现状

多核高并发，多IO、业务复杂度不断增加

对性能精度的要求越来越高，ms》us》ns

开发语言C++ 机制复杂，高性能编码难度高

系统业务需求变更节奏越来越快

- 研发团队现状

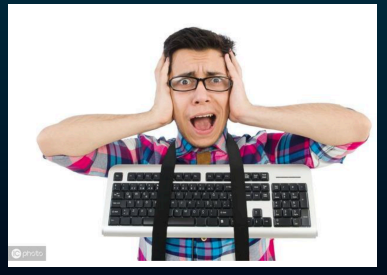
复杂系统的高性能的设计与建模能力不足

只能进行简单、粗放式的获取性能测量

传统C开发，对C++高性能编码经验很有限

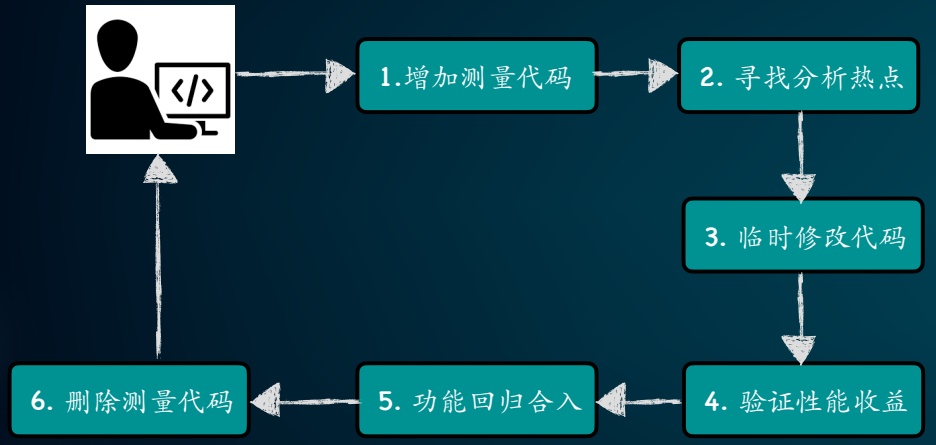
赶进度、堆代码式开发，很快掉入性能泥潭

软件性能太差了！怎么办？



C++ 传统性能优化

- 传统性能优化过程方法



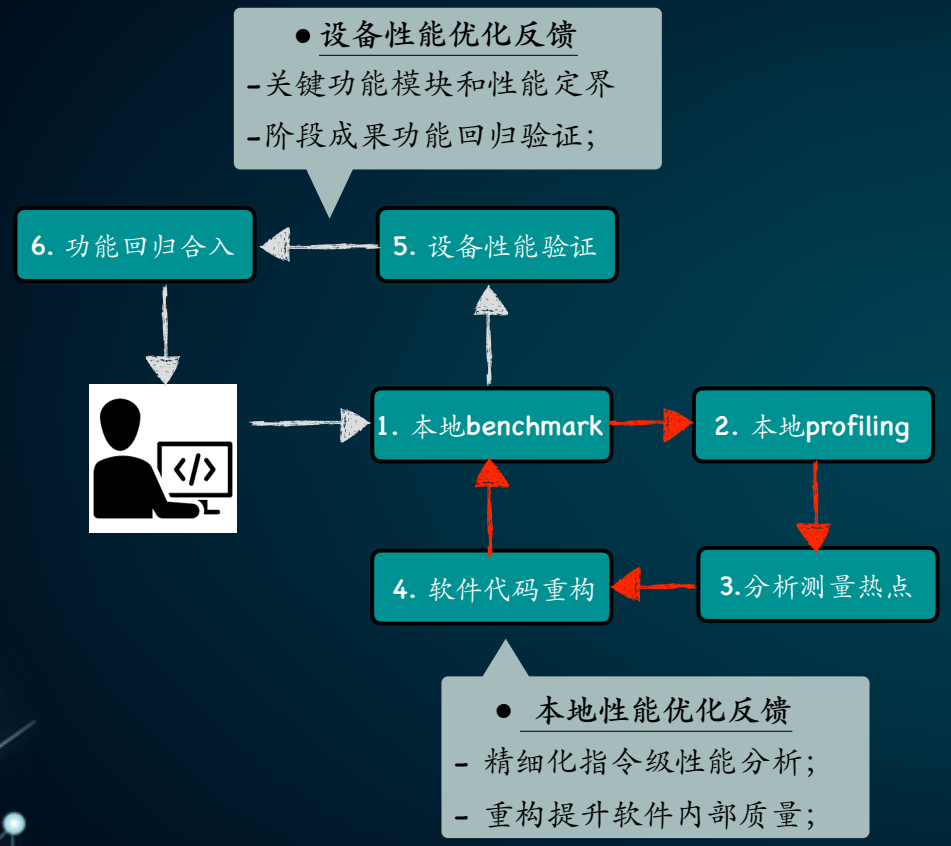
- 问题
  - 打点测量数据不系统，不支持精准性能分析；
  - 中间手动操作繁琐，容易出错；
  - 临时性的侵入式增加打点代码，浪费严重；
  - 全系统运行性能测试，无法剥离外部子系统干扰；
  - 占用系统资源比较多；
  - 性能测试验证反馈周期长，优化效率低下；
  - 性能优化成果守护难，难于持久；
  - 性能优化导致代码内部质量更差；

总结：当C++软件系统面临极致化性能需求，传统性能优化过程存在的弊端会更加的严重；

02

C++性能优化重构工程

C++性能优化重构工程



➤ 价值优势

- 测试反馈周期快，可以支持重构迭代;
- 数据分析，无需侵入增加打点代码;
- 本地化测试全自动化，无需人工干预;
- 测量数据精细化，支撑精准性能分析;
- 打点测量数据不系统，不支持精准性能分析;
- 剥离掉外部依赖，独立分析业务性能

C++性能优化重构工程

- 可行性与约束分析

☑Cpu 架构维度

- 指令系统： x86，arch64，自研指令集系统
- 主频，cache大小等；

☑外部依赖维度

- 与os、platform的接口依赖
- 本地化测试系统中的mock，stub实现等；

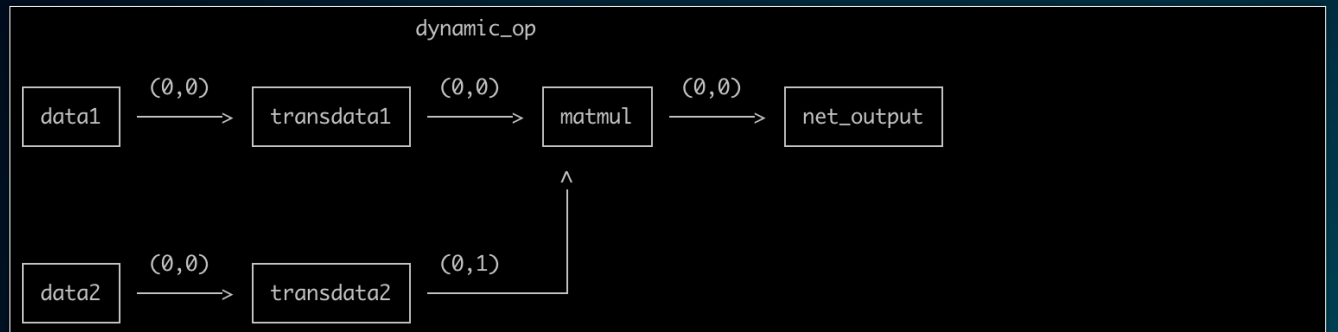
☑硬件访问维度

- 寄存器访问，中断等；

☑业务特性维度

- IO密集型，还是CPU密集型
- 实时性系统还是吞吐量性能系统；

C++性能优化重构工程




```
-------------------------------------------------------------------------------------------------------
Benchmark                                                        Time             CPU   Iterations
-------------------------------------------------------------------------------------------------------
test_hybrid_single_op_exec_performance_when_infer_in_same_thread     467842 ns       165093 ns         4465
---------------- GraphEngine package archive generated ----------------
```

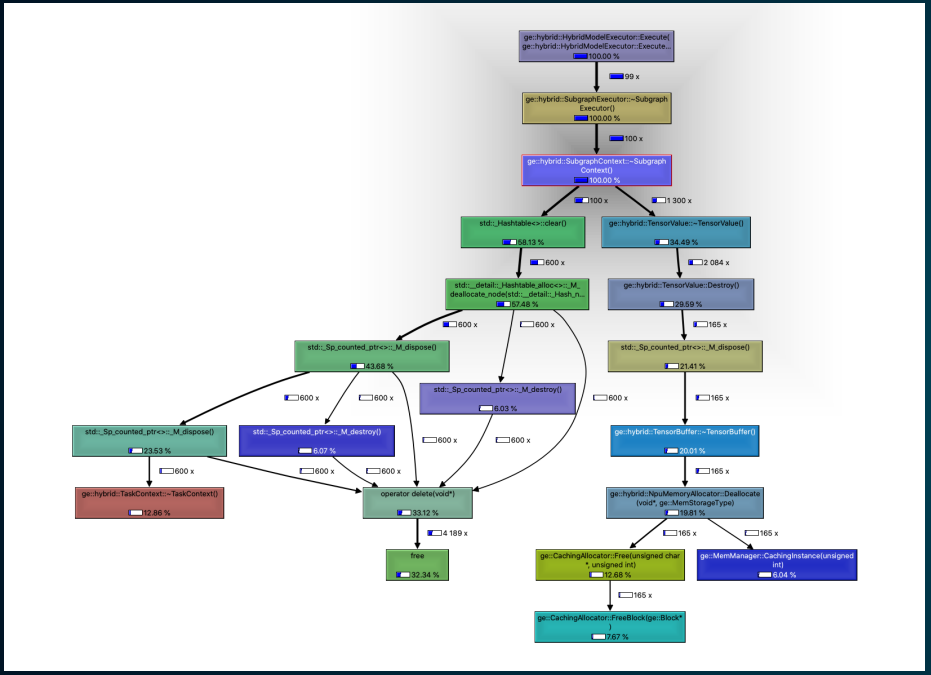
```
Load Average: 0.12, 0.10, 0.11
-------------------------------------------------------------------------------------------------------
Benchmark                                                        Time             CPU   Iterations
-------------------------------------------------------------------------------------------------------
test_thread_pool_commit_performance                              80394 ns        30589 ns        24418
test_hybrid_single_op_exec_performance_when_infer_in_same_thread 433325 ns       154186 ns         4164
test_hybrid_single_op_performance_when_infer_in_thread_pool      469827 ns       181249 ns         3598
test_transdata_op_infer_performance                                6406 ns         6380 ns       114037
test_singel_op_convert_performance                                  163 ns          162 ns      4366725
test_single_shape_refiner_op_performance                           5868 ns         5838 ns       118724
test_single_op_exec_performance                                   14377 ns        14328 ns        49269
---------------- GraphEngine package archive generated ----------------
```

优化前

优化后

- BenchMark 如何发挥最佳效果
  - 有ST级别测试框架支持；
  - 业务执行逻辑与系统保持一致；
  - 支撑各种方案的性能对比测试；
  - 支持细粒度功能接口的性能测试；
  - 本地机器和系统服务器硬件差异，可以通过等比分析；
  - 关注性能提升百分比，而不是绝对值；
  - 一键式命令执行；

C++性能优化重构工程



- ✦ 优化前开销： 11us左右；(10630 IR percall)
- ✦ 精准分析结论：

1. 内存释放占比 33%;
2. TensorBuffer释放20%
3. TaskContext释放 12%
4. 智能指针开销（10% ~20%）

- ✦ 性能优化重构点：

1. 去掉所有动态申请（node_state, frame_state);
2. 根据什么生命周期重构领域对象，删除智能指针;
3. 重构领域对象区分可变部分、非可变部分，支持可重入;

- 备注： 获取分析结果，可以快速修改验证，并获取反馈验证结果；

03

C++性能优化重构实战详解

C++性能优化
重构实战



1. 性能优化开销基数
2. OO 生命周期优化
3. 高效实现并发互斥
4. Modern C++高性能编码
5. C++第三方库使用

性能优化开销基数

1. 线程创建级：
   - 时长： 10us+
   - 场景： 线程创建，进程创建等
2. 线程切换级
   - 时长： 2us+
   - 场景： 互斥锁、读写锁等
3. 内存申请级
   - 时长： 30ns~500ns
   - 场景： new, malloc, free, string copy等
4. CPU流水线停滞
   - 时长： 10ns+
   - 场景： atom变量操作， 内存屏障
5. CPU执行流水线低效
   - 时长： ns 级
   - 场景： 分支逻辑判断， volatile;

OO 生命周期优化

- 使用引用替换指针

```
struct Car{
  Car(Engine *engine, Tyre* tyre): engine(engine),
tyre(tyre){
  }

  bool Run(){
    if(engine == nullptr){
      return false;
    }
  }

  Tyre* GetTyre(){
    return tyre;
  }

private:
  Engine *engine;
  Tyre* tyre;
};
```

使用指针的潜在弊端：

- 冗余的判断；
- 无效的测试；
- 隐藏设计问题；

解决办法：

- 尽量使用引用（组合，聚合）；
- 使用visitor模式访问机会元素；

注意事项：

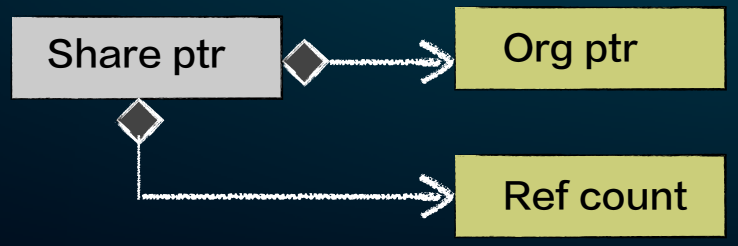
- 记得使用auto& ret接受返回值；

OO 生命周期优化

- 尽量减少使用share ptr

```
  template<typename _Yp>
_Assignable<_Yp>
operator=(const __shared_ptr<_Yp, _Lp>& __r) noexcept
{
  _M_ptr = __r._M_ptr;
  _M_refcount = __r._M_refcount; // __shared_count::op=
doesn't throw
  return *this;
}
```

```
_Atomic_word  _M_use_count;     // #shared
_Atomic_word  _M_weak_count;    // #weak + (#shared != 0)
```



SharePtr潜在弊端：

- 很多场景下两个动态内存;
- 引用计数原子操作;
- 不能使用栈上分配的对象;
- 二级制膨胀

解决办法：

- 合理设计对象生命周期
- 使用裸指针指针或引用替换

OO 生命周期优化

- 合理使用unique_ptr

```
struct Executor{
  Executor();

  void Init(int para){
    context.reset(new Context(para));
  }

  void Run(){
    auto para = context->GetPara();
    ...
  }

private:
  std::unique_ptr<Context> *context;
};
```

使用Unique_ptr潜在弊端：

- 冗余的判断校验；
- 对象构造原子化破坏；
- 执行性能差；
- 二级制膨胀；

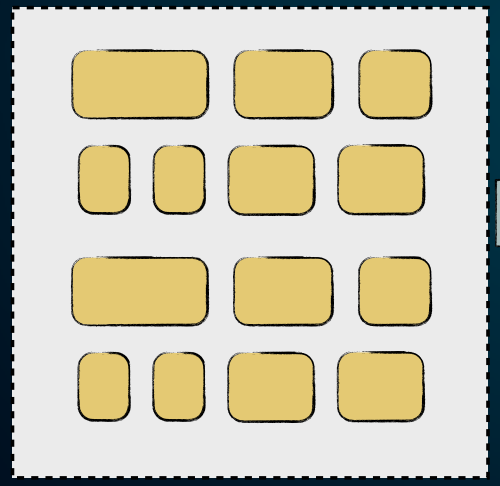
解决办法：

- 调整优化对象生命周期
- 实现对象可重置的机制
- 使用placement延迟初始化

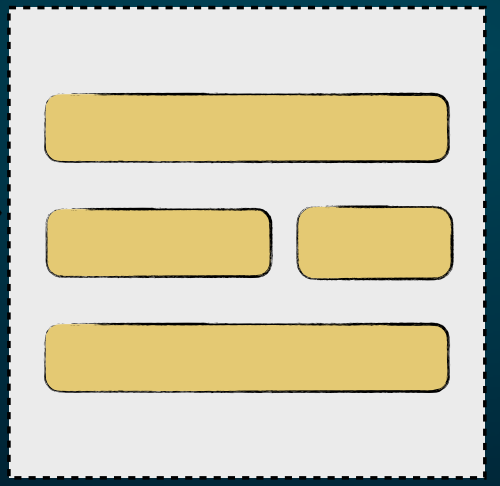
OO 生命周期优化

- 重构优化OO生命周期

●无生命周期管理



● 有效生命周期管理

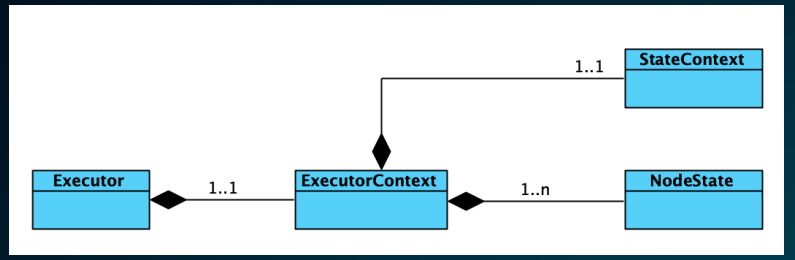


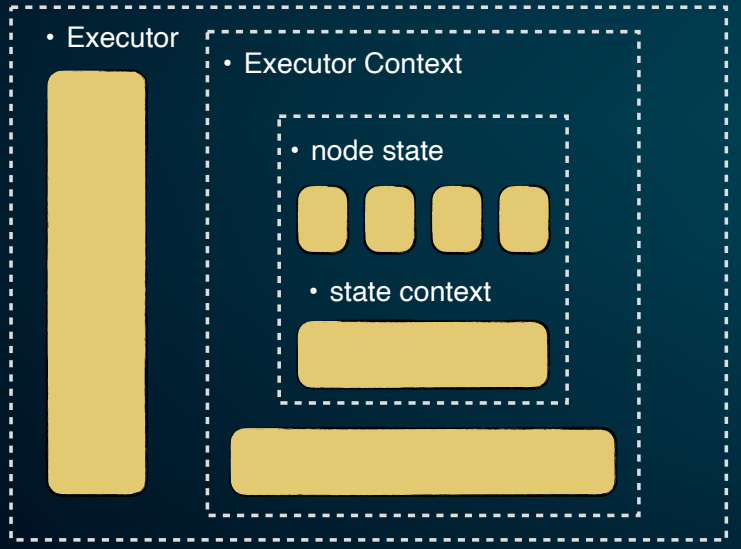
● 高性能内存管理

- 重载new/delete：内存自定义管理；
- placement ：内存预占，延迟初始化，指定内存构造；
- object-pool：静态确定对象规格并预占内存；
- 替代STL库：自行封装智能指针和引用计数，自行封装容器；

OO 生命周期优化-案例

- 优化前

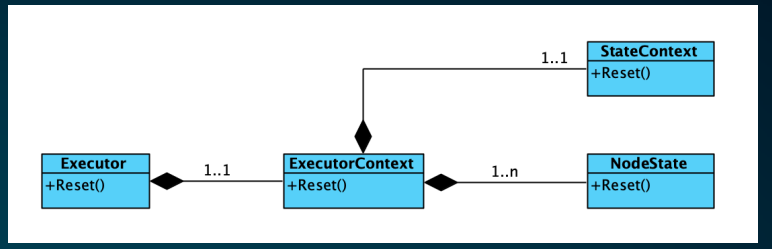


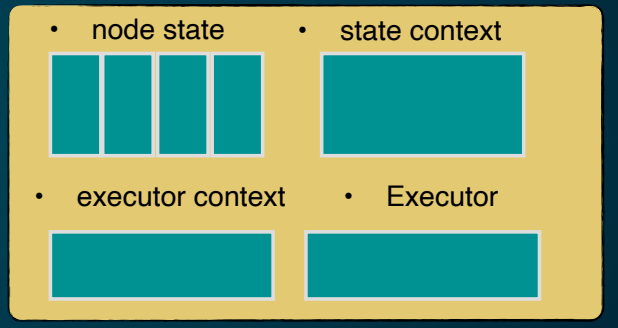


内存块个数：8块

每次创建释放： 10us

- 优化后





内存块个数：1块

每次创建释放： 2us

主要优化点：

- 基于私有继承/成员变量整合对象占用内存块；
- 基于placement方式整合对象数组的内存块；
- 区分不变数据和可变数据， 提供reset方案减少创建释放调用

高效实现并发互斥

```cpp
#include <iostream>
#include <thread>

int number_1 = 0;
int number_2 = 0;

void atom_increase_call()
{
        for (int i = 0; i < 10000; i++)
        {
                number_1++;
                number_2++;
        }
}

void atom_read_call()
{
        int inorder_count = 0;
        for (int i = 0; i < 10000; i++)
        {
                if (number_2 > number_1)
                {
                        inorder_count++;
                }
        }
        std::cout << "thread:3 read inorder_number is " << inorder_count
                        << std::endl;
}

int main()
{
        std::thread threadA(atom_increase_call);
        std::thread threadB(atom_increase_call);
        std::thread threadC(atom_read_call);
        threadA.join();
        threadB.join();
        threadC.join();
        std::cout << "thread:main read number is " << number_1 << std::end
        return 0;
}
```

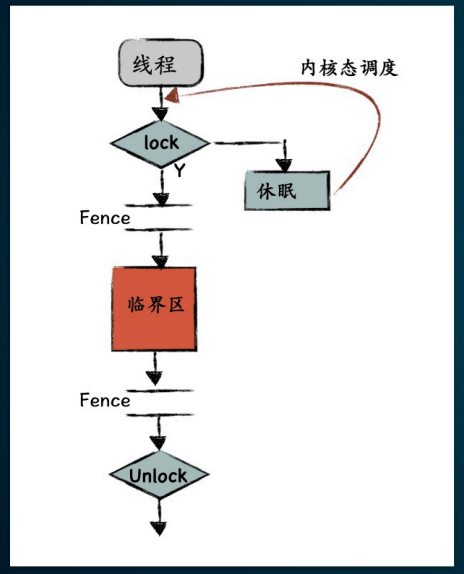
```
| 第一次:
thread:3 read inorder_number is 1
thread:main read number_1 is 15379
thread:main read number_2 is 15378

| 第二次:
thread:3 read inorder_number is 13
thread:main read number_1 is 15822
thread:main read number_2 is 15821
```

- 原子性破坏
- cache失效
- 指令执行乱序

高效实现并发互斥

- 同步互斥开销对比

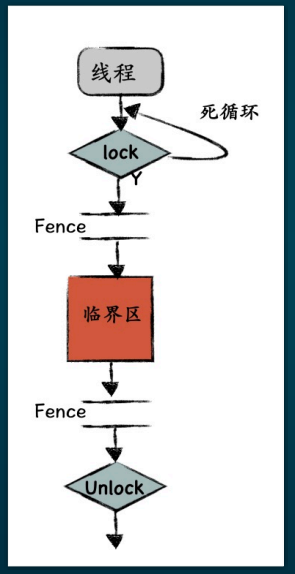


•mutux lock

✓加锁、解锁时间：10ns级别

✓上下文切换： 大于2us；

✓对指令流水线影响： 大

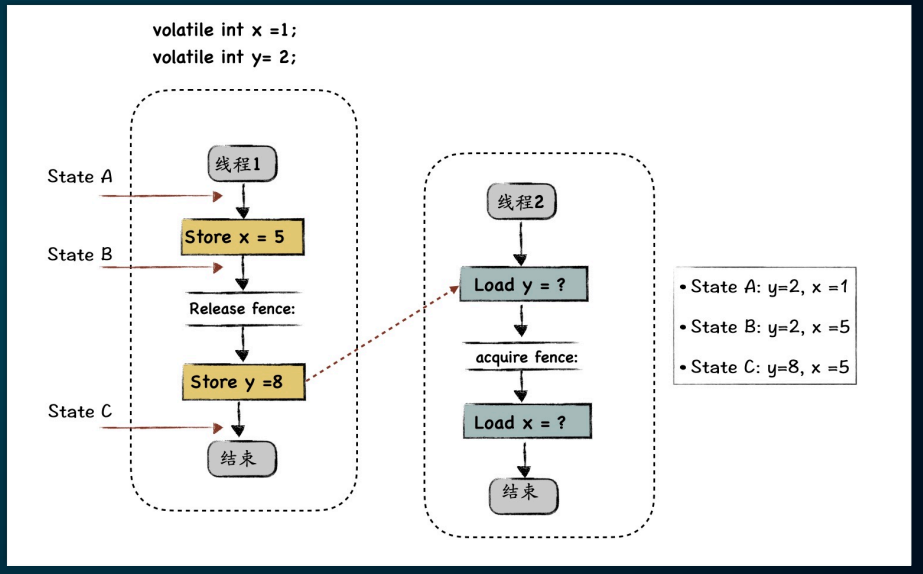


• Spinlock

✓加锁、解锁时间：10ns级别；

✓等待时间：和临界区大小相关；

✓对指令流水线影响： 大



• Volatile + memory fence（或者原子变量）

✓加锁、解锁时间： 无

✓等待时间： 业务逻辑相关；

✓对指令流水线影响： 小

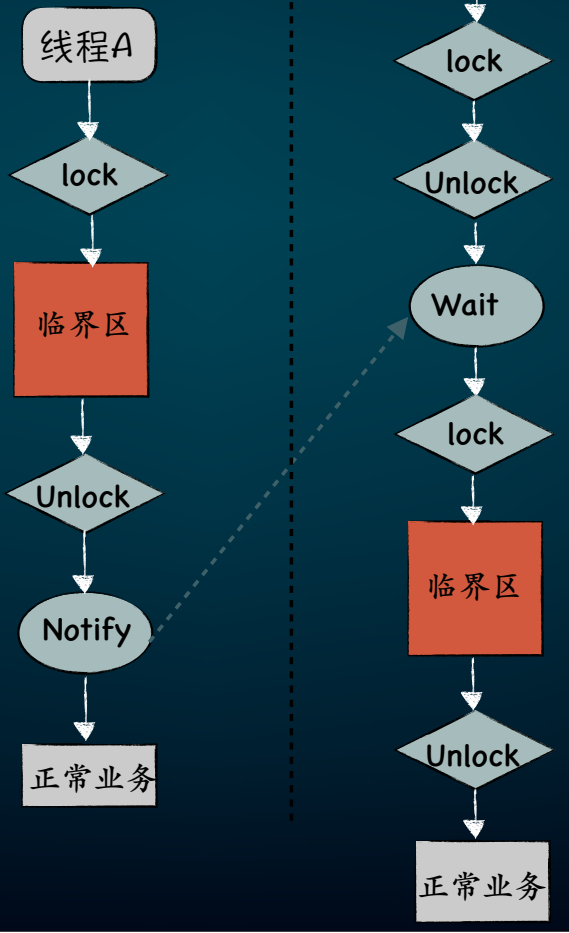
高效实现并发互斥

- 留意条件变量的性能开销

```
std::deque<int> q;
std::mutex mu;
std::condition_variable cond;

void producer() {
    int count = 10;
    while (count > 0) {
        std::unique_lock<std::mutex> locker(mu);
        q.push_front(count);
        locker.unlock();
        cond.notify_one();  // Notify one waiting thread, if there is one.
        std::this_thread::sleep_for(std::chrono::seconds(1));
        count--;
    }
}

void consumer() {
    int data = 0;
    while ( data != 1) {
        std::unique_lock<std::mutex> locker(mu);
        while(q.empty())
            cond.wait(locker); // Unlock mu and wait to be notified
        data = q.back();
        q.pop_back();
        locker.unlock();
        std::cout << "t2 got a value from t1: " << data << std::endl;
    }
}
```



使用的潜在弊端：

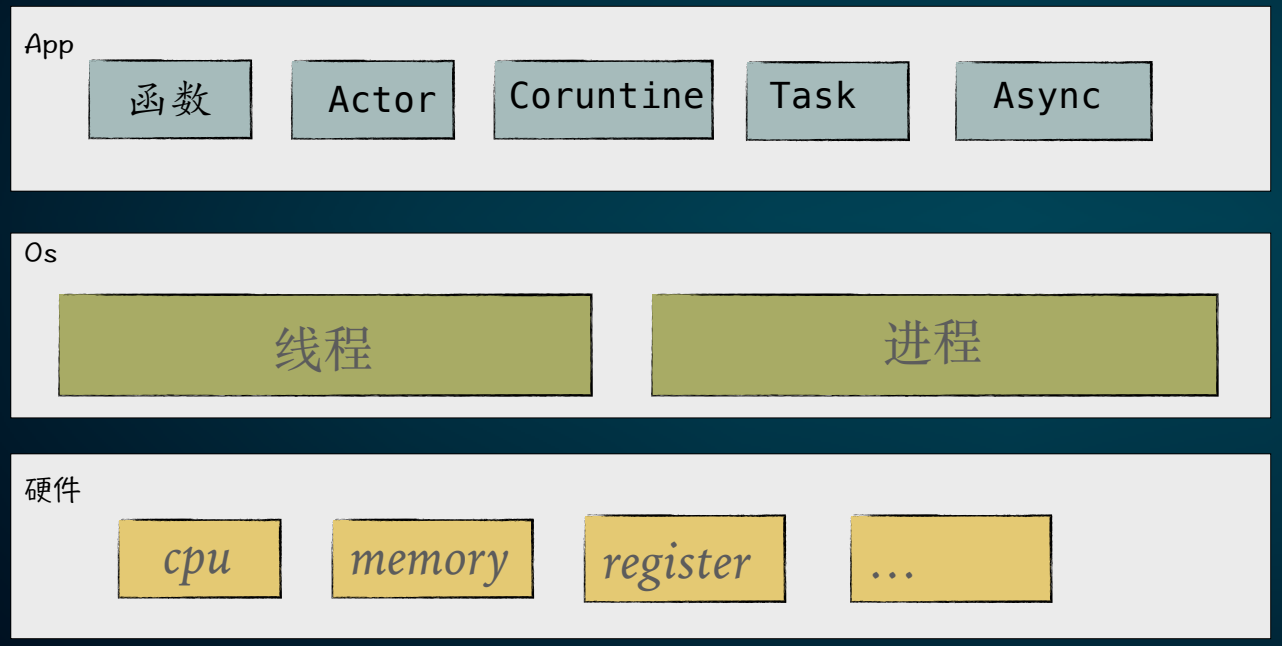
- 同步阻塞过长
- 线程切换性能损耗

解决办法：

- 调整并发设计，减少使用次数

高效实现并发互斥

- 创建线程开销不能忽视



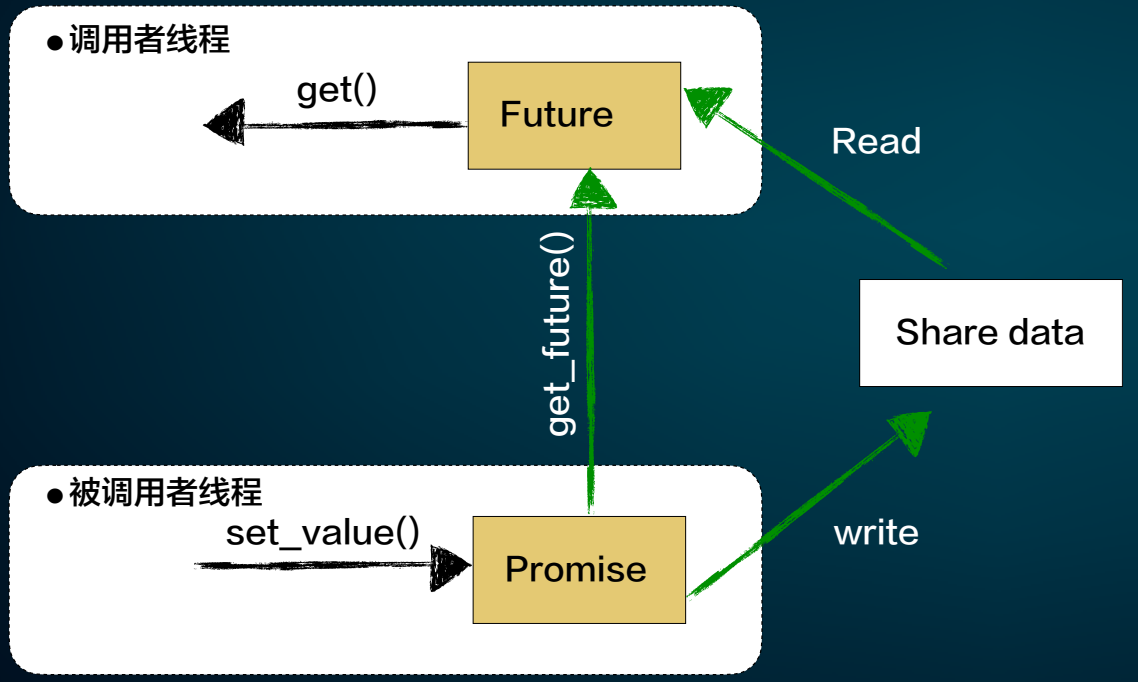
- 类型1：等待性线程
- 类型2：繁忙性线程

创建线程开销：

- 为线程在操作系统表中分配空间
- 初始化寄存器组
- 调度线程开销
- 分配调度量子开销
- 增加内存使用
- cache的抖动
- 流水线停滞
- 线程栈分配内存开销
- 几十微妙

高效实现并发互斥

- 合理使用Async机制



使用的潜在弊端：

- 占用当前线程时间长（10us+，创建线程)
- 小代码段使用async造成CPU浪费

解决办法：

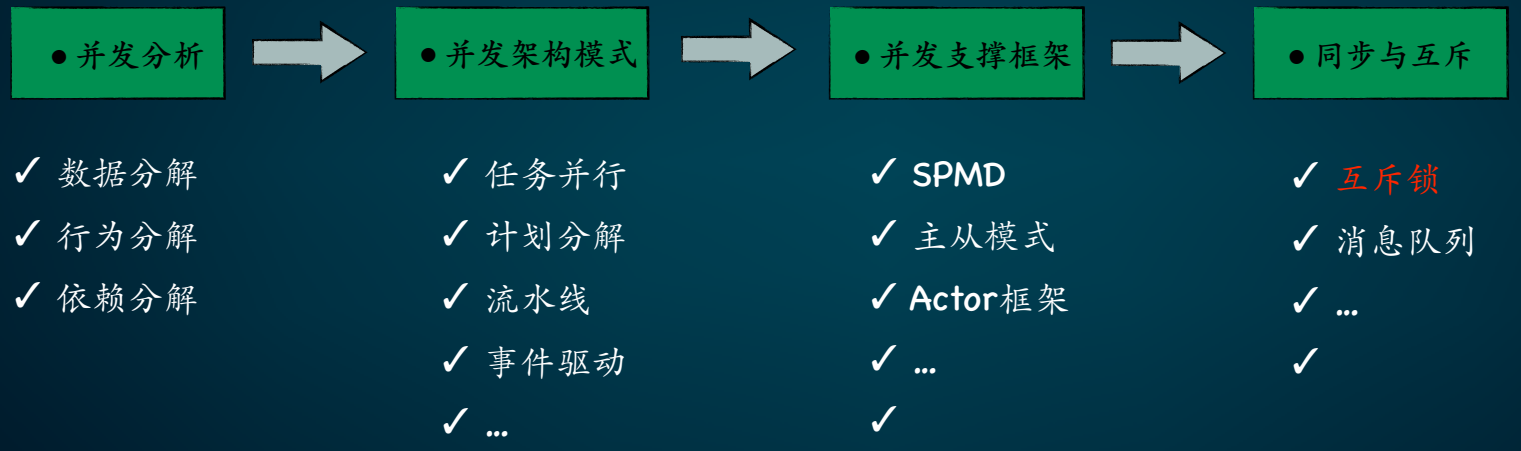
- 使用线程池技术
- 更小的业务代码串行执行；

高效实现并发互斥

- 同步互斥优化策略
  - 减少锁使用频率
  - 减少锁粒度
  - 减少锁持有时间（临界区）
  - 读写锁分离
  - 锁粗化
  - 避免无效等待
  - 无锁化(volatile + fence + CAS)

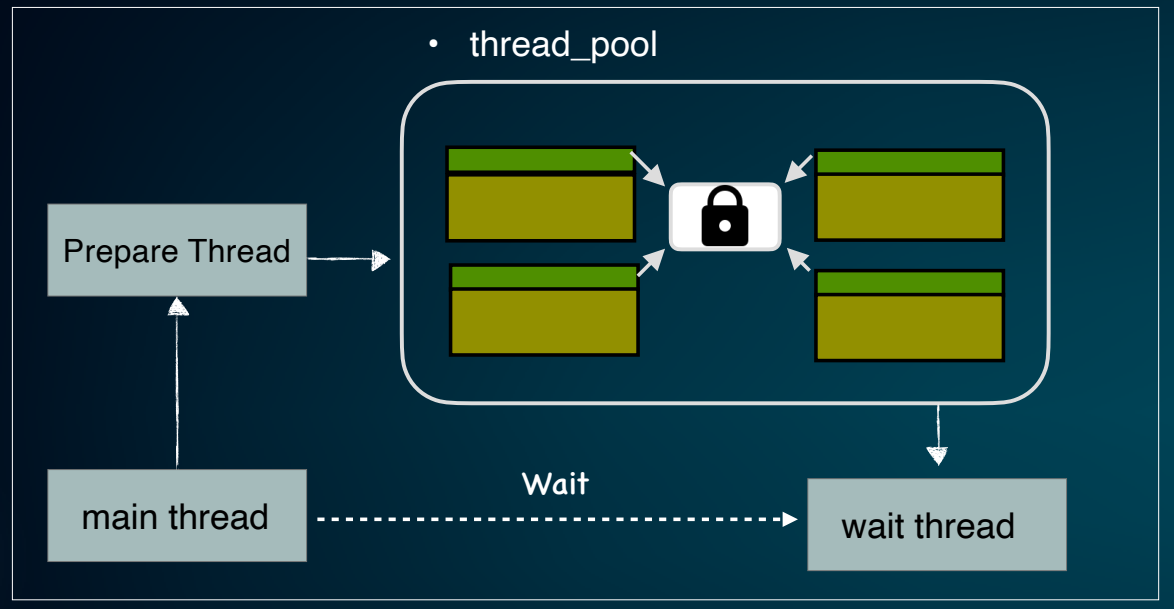
高效实现并发互斥

- 并发分析与设计全景

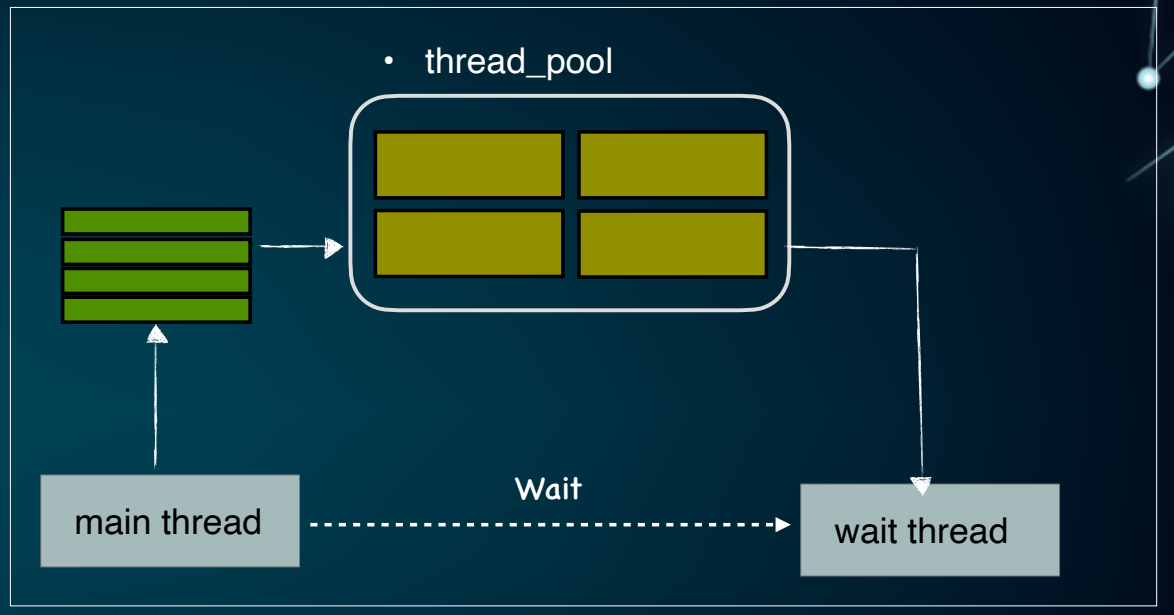


高效实现并发互斥-案例

● 优化前



● 优化后



优化前：

- 线程池内线程之间存在互斥访问
- 创建独立准备线程（**prepareThread**）开销大于业务执行时间；
- 因为线程并发互斥侵入到业务实现，导致业务代码锁泛滥；

优化后：

- 原来准备线程执行开销很小，去除冗余的线程创建；
- 剥离线程池互斥访问的业务，放到独立线程中；
- 优化去除线程池的锁，去除领域对象代码中锁；

Modern C++高性能编码

- 利用ROV优化特性

```
struct Shape{
  std::array<int, 8> dim;
};

Shape GetShape(){
  Shape output;
  output.dim = {0, 2, 3, 4};
  return output;
}

void GetShapeByPara(Shape& output){
  output.dim = {0, 2, 3, 4};
  return input;
}
```

类对象通过返回值返回：

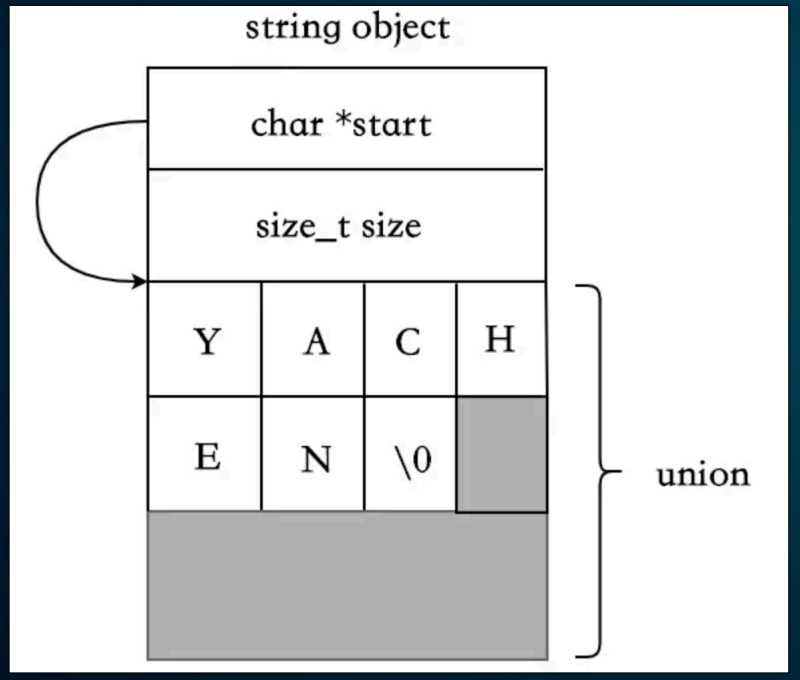
- 接口使用者更友好；
- ROV优化可以减少返回值拷贝，性能有可能更佳

类对象通过参数返回：

- 接口使用者不友好
- 由于需要在外部创建对象，性能很有可能也不佳

Modern C++高性能编码

- 利用SSO机制，设计系统中的string变量；



采用SSO机制后string特点：

- 15个字节内的字符串会采用栈上分配
- 抛弃了COW机制，每次copy都需要复制内容；

使用建议：

- 很多业务场景，可以通过设计控制字符串长度小于15；
- 针对几乎不改动，大量赋值的场景，可以设计实现COW的字符串

- std::string 使用SSO替换COW机制的实现

Modern C++高性能编码

- 利用constexpr 编译器计算优化

```
struct A {
  constexpr A(int xx, int yy): x(xx), y(yy)
{}
  int x, y;
};

constexpr A a(1, 2);
enum {SIZE_X = a.x, SIZE_Y = a.y};
```

constexpr的机制能力：

- 支持constexpr构造函数（c++11）
- 可以使用分支语句（C++14）
- 可以使用循环控制 （C++14）
- ++n (c++14)

使用建议：

- 识别系统中的编译期计算，使用constexpr来编码实现

Modern C++高性能编码

- Inline 模式

```
static void test_shape_inline(benchmark::State &state) {
  for (auto _ : state) {
    auto shape = Shape{2, x, 5};
    auto shape2 = Shape{2, 4, 5};
    auto shape3 = Shape{2, 3, z};
    for(int i = 0; i < 4 ;i ++){
      shape3.SetDim(2, shape3.GetDim(2) * (i+1));
    }
    ret = shape3.GetDim(2) +  shape2.GetDim(2) + shape3.GetDim(1);
  }
}
BENCHMARK(test_shape_inline);
```

- NotInline 模式

```
static void test_shape_not_inline(benchmark::State &state) {
  for (auto _ : state) {
    auto shape = Shape{2, x, 5};
    auto shape2 = Shape{2, 4, 5};
    auto shape3 = Shape{2, 3, z};
    for(int i = 0; i < 4 ;i ++){
      shape3.SetDimNotInline(2, shape3.GetDimNotInline(2) * (i+1));
    }
    ret = shape3.GetDimNotInline(2) +  shape2.GetDimNotInline(2) + shape3.GetDimNotInline(1);
  }
}

BENCHMARK(test_shape_not_inline);
```

```
-------------------------------------------------------------------------------
Benchmark                                     Time             CPU   Iterations
-------------------------------------------------------------------------------
test_transdata_infershape_performance      2.44 ns         2.44 ns    285348981
test_reshape_infershape_performance        46.9 ns         46.8 ns     14393235
test_dynamic_rnnv3_infershape_performance  9.48 ns         9.48 ns     62124113
test_shape_inline                         0.295 ns        0.295 ns   1000000000
test_shape_not_inline                      19.9 ns         19.9 ns     33791130
Generating coverage statistics, please wait
```

初步结论： 是否使用内联方式，有可能导致最终函数执行开销增加近几十倍

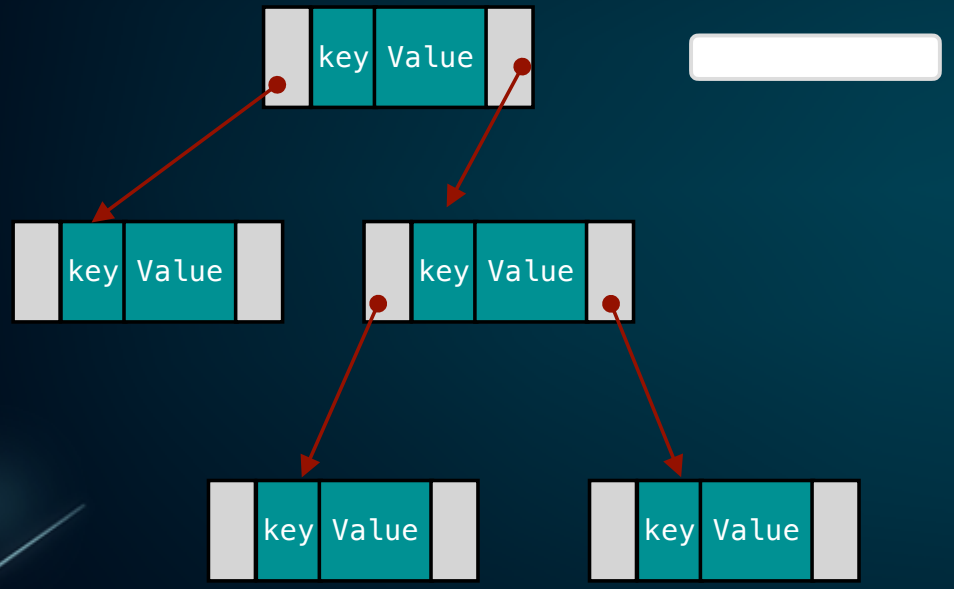
原因分析： 不启用内联，不仅增加了函数调用开销，而且会抑制编译器指令优化、以及CPU的指令发射性能

Modern C++高性能编码

- 更新C++编码
  - 使用move 机制减少内存copy
  - 给函数增加 no except
  - 使用type rich 编程
  - …

C++库使用

- 慎用string 类型的std::map操作



使用潜在弊端：

- String 的比较大小性能开销大；
- 插入map时需要进行String的拷贝；
- String拷贝可能引入额外动态内存申请；
- 插入操作会导致平衡二叉树调整；

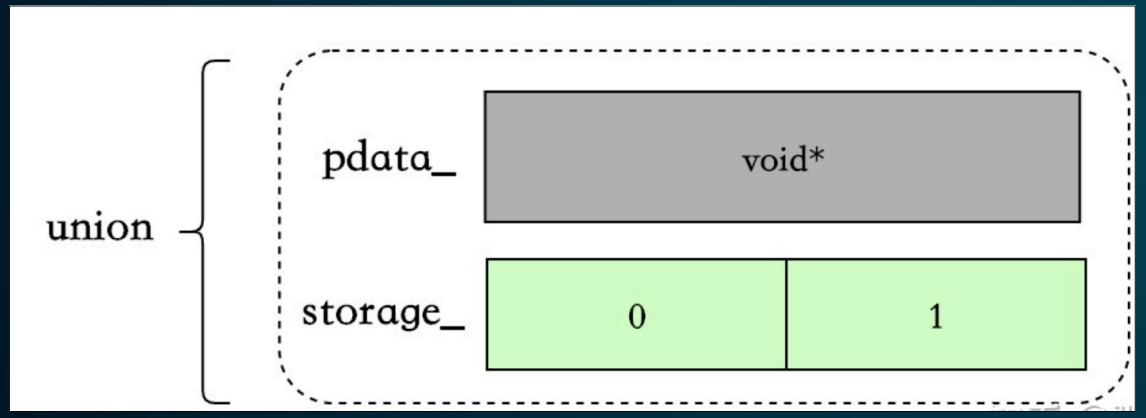
解决办法：

- 尽量不要使用string类型的map；

C++第三方库使用

- 使用智能栈上分配

•small_vector<int, 2> vec;



使用普通vector的弊端：

- 比较小的集合使用动态内存开销大；

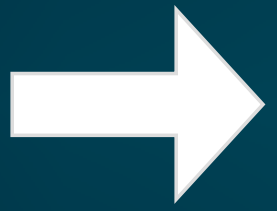
解决办法：

- 使用智能栈上分配替换动态集合；
- 这种技术可以很多场合使用；

C++第三方库使用

- 尽量使用emplace 操作

- std:: vector
  - push_front
  - Insert
  - push_back
- std::map
  - Insert



- std:: vector
  - emplace_front
  - emplace
  - emplace_back
- std::map
  - emplace

说明：针对特定场景，可以实现类似emplace的原地构造方法

C++第三方库使用

- 使用编译器类型替换运行时

```
namespace ge {
template<typename T>
struct __attribute__((visibility("default"))) TypeIdHolder {
constexpr static char id = 0;
};

template<typename T>
constexpr char TypeIdHolder<T>::id;

using TypeId = const void *;
constexpr TypeId kInvalidTypeId = nullptr;

template<typename T>
__attribute__((visibility("default"))) TypeId GetTypeId(const T &obj) {
  using PureT = typename std::remove_cv<typename std::remove_reference<T>::type>::type;
  return &(TypeIdHolder<PureT>::id);
}

template<typename T>
__attribute__((visibility("default"))) TypeId GetTypeId() {
  using PureT = typename std::remove_cv<typename std::remove_reference<T>::type>::type;
  return &(TypeIdHolder<PureT>::id);
}
}
```

动态类型潜在弊端：

- 查询性能开销大；
- 占用空间大；

解决办法：

- 使用编译器类型ID；

注意事项：

- 控制静类型使用范围；

C++第三方库使用

- 使用编译器类型替换运行时

```
[ RUN      ] TypeIdTest.test_so_1_and_so_2
 call in so 1
int type id is0x7fe6b39ae348
double type id is0x7fe6b39ae349
 call in so 2
int type id is0x7fe6b39ae348
double type id is0x7fe6b39ae349
[       OK ] TypeIdTest.test_so_1_and_so_2 (1 ms)
```

文档解释

```
ıring/wei/air# readelf -Ws ./build/tests/st/so_1/libSO_1.so | gr
   1 OBJECT  UNIQUE DEFAULT   23 _ZN2ge12TypeIdHolderIdE2idE
   1 OBJECT  UNIQUE DEFAULT   23 _ZN2ge12TypeIdHolderIiE2idE
   1 OBJECT  UNIQUE DEFAULT   23 _ZN2ge12TypeIdHolderIdE2idE
   1 OBJECT  UNIQUE DEFAULT   23 _ZN2ge12TypeIdHolderIiE2idE
```

C++第三方库使用

- 其他第三方库
  - Folly
    - 全称： Facebook Open-source Library
    - 代码仓：https://github.com/facebook/folly
    - 参考： https://zhuanlan.zhihu.com/p/348596697
  - absl(Abseil) --- google
    - 全称： Facebook Open-source Library
    - 代码仓：https://github.com/facebook/folly
  - boost库
    - 全称： Facebook Open-source Library
    - 官网：https://www.boost.org/

谢谢观看



Paul McKenney

前IBM杰出工程师, 世界级并行编程专家

Paul E. McKenney是世界级并行编程专家，Linux 内核中 RCU 实现和 rcutorture 测试模块的维护者，也是RCU的发明人。对于实时操作系统内核同步机制（例如 Linux 中的实时 RCU）、Linux 和 UNIX 操作系统内核中的 SMP/NUMA 可扩展性和性能、网络性能分析、路由和拥塞控制， 嵌入式实时应用程序有着丰富的经验和研究。 Paul 曾担任IBM Linux 技术中心杰出工程师，发表过 200 多篇论文，拥有 100 多项专利。他也是经典图书《深入理解并行编程》的作者。他的专长包括多线程和多核系统的性能编程、NUMA 架构、开源软件项目。

Paul E. McKenney, Facebook

Linux Foundation Live Mentorship Series, December 7, 2021



Unraveling RCU-Usage Mysteries

(Fundamentals)

© 2021 Facebook Corporation

RCU Usage: Overview

- Respect the Laws of Physics!!!
- Memory Ordering and Memory Fences
- You Are Here
- Example Multithreaded Use Case
  - Reader-writer locking and RCU
- RCU Usage in Linux Kernel
- Debugging Linux-Kernel RCU Code

Respect the Laws of Physics!!!

Respect the Laws of Physics!!!

- Following the footsteps of Admiral Hopper:
  - Light goes 11.803 inches/ns in a vacuum
    - Or, if you prefer, 29.979 cm/ns
  - But over and back: 5.9015 in/ns
  - But 4GHz: 1.4754 in/ns
  - But Cu: ~0.5 in/ns, or Si transistors: ~0.05 in/ns
  - Plus other slowdowns: prototols, electronics, ...

But Laws Have Loopholes...

- 3D integration for smaller systems
- Systolic arrays for unidirectional data flow
  - Obtain similar effects with modern accelerators
- New semiconductor technologies
  - Too bad about the need to hand-place atoms...

But Laws Have Loopholes...

- 3D integration for smaller systems
- Systolic arrays for unidirectional data flow
  - Obtain similar effects with modern accelerators
- New semiconductor technologies
  - Too bad about the need to hand-place atoms...
- **Store buffers and read-only replication**

Memory Ordering & Memory Fences

“Load Buffering” (LB), Relaxed

Thread 0:

```
r1 = x0.load(memory_order_relaxed);
x1.store(1, memory_order_relaxed);
```

Thread 1:

```
r2 = x1.load(memory_order_relaxed);
x2.store(1, memory_order_relaxed);
```

Thread 2:

```
r2 = x2.load(memory_order_relaxed);
x0.store(1, memory_order_relaxed);
```

Outcome of interest: `r1 && r2 && r3`

“Load Buffering” (LB), **Relaxed**

Thread 0:

```
r1 = x0.load(memory_order_relaxed);
x1.store(1, memory_order_relaxed);
```



Thread 1:

```
r2 = x1.load(memory_order_relaxed);
x2.store(1, memory_order_relaxed);
```



Thread 2:

```
r2 = x2.load(memory_order_relaxed);
x0.store(1, memory_order_relaxed);
```



Outcome of interest: `r1 && r2 && r3`

“Load Buffering” (LB), Relaxed

Thread 0:

```
x1.store(1, memory_order_relaxed);
r1 = x0.load(memory_order_relaxed);
```

Thread 1:

```
x2.store(1, memory_order_relaxed);
r2 = x1.load(memory_order_relaxed);
```

Thread 2:

```
x0.store(1, memory_order_relaxed);
r2 = x2.load(memory_order_relaxed);
```

Outcome of interest: `r1 && r2 && r3` **Can happen!**

http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem/

“Load Buffering” (LB), Rel/Acq

“Load Buffering” (LB), Rel/Acq

Thread 0:

```
    r1 = x0.load(memory_order_acquire);
    x1.store(1, memory_order_release);
```

Thread 1:

```
    r2 = x1.load(memory_order_acquire);
    x2.store(1, memory_order_release);
```

Thread 2:

```
    r2 = x2.load(memory_order_acquire);
    x0.store(1, memory_order_release);
```

Outcome of interest: `r1 && r2 && r3`

“Load Buffering” (LB), Rel/Acq

Thread 0:

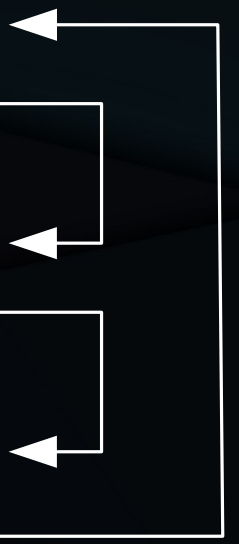
```
r1 = x0.load(memory_order_acquire);
x1.store(1, memory_order_release);
```

Thread 1:

```
r2 = x1.load(memory_order_acquire);
x2.store(1, memory_order_release);
```

Thread 2:

```
r2 = x2.load(memory_order_acquire);
x0.store(1, memory_order_release);
```



Synchronizes with

Outcome of interest: `r1 && r2 && r3`

“Load Buffering” (LB), Rel/Acq

Thread 0:

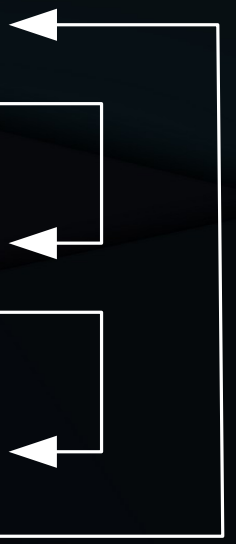
```
r1 = x0.load(memory_order_acquire);
x1.store(1, memory_order_release);
```

Thread 1:

```
r2 = x1.load(memory_order_acquire);
x2.store(1, memory_order_release);
```

Thread 2:

```
r2 = x2.load(memory_order_acquire);
x0.store(1, memory_order_release);
```



Synchronizes with

Outcome of interest: `r1 && r2 && r3` **Cannot happen!**

“Load Buffering” (LB), Rel/Acq/Rlx

“Load Buffering” (LB), Rel/Acq/RIx

Thread 0:

```
r1 = x0.load(memory_order_relaxed);
x1.store(1, memory_order_release);
```

Thread 1:

```
r2 = x1.load(memory_order_acquire);
x2.store(1, memory_order_release);
```

Thread 2:

```
r2 = x2.load(memory_order_acquire);
x0.store(1, memory_order_relaxed);
```

Outcome of interest: `r1 && r2 && r3`

“Load Buffering” (LB), Rel/Acq/Rlx

Thread 0:

```
r1 = x0.load(memory_order_relaxed);
x1.store(1, memory_order_release);
```

Thread 1:

```
r2 = x1.load(memory_order_acquire);
x2.store(1, memory_order_release);
```

Thread 2:

```
r2 = x2.load(memory_order_acquire);
x0.store(1, memory_order_relaxed);
```



Outcome of interest: `r1 && r2 && r3` **Still cannot happen!**

Check with CPPMEM

```
int main() {
  atomic_int x0 = 0;
  atomic_int x1 = 0;
  atomic_int x2 = 0;
  {{{ { r1 = x0.load(memory_order_relaxed).readsvalue(1);
        x1.store(1, memory_order_release); }
  ||| { r2 = x1.load(memory_order_acquire).readsvalue(1);
        x2.store(1, memory_order_release); }
  ||| { r2 = x2.load(memory_order_acquire).readsvalue(1);
        x0.store(1, memory_order_relaxed); }
  }}};
  return 0; }
```

http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem/

Check with CPPMEM

```
int main() {
  atomic_int x0 = 0;
  atomic_int x1 = 0;
  atomic_int x2 = 0;
  {{{ { r1 = x0.load(memory_order_relaxed).readsvalue(1);
        x1.store(1, memory_order_release); }
  ||| { r2 = x1.load(memory_order_acquire).readsvalue(1);
        x2.store(1, memory_order_release); }
  ||| { r2 = x2.load(memory_order_acquire).readsvalue(1);
        x0.store(1, memory_order_relaxed); }
  }}};
  return 0; }
```

8 executions; no consistent

http://svr-pes20-cppmem.cl.cam.ac.uk/cppmem/

A More Ornate Example

A More Ornate Example (W+WRC)

Thread 0:

```
x0.store(1, memory_order_release);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3`

A More Ornate Example (W+WRC)

Thread 0:

```
x0.store(1, memory_order_release);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
r3 = x0.load(memory_order_acquire);
```

No ordering!

Outcome of interest: `r1 && !r2 && !r3` **Can happen!**

A More Ornate Example (W+WRC)

Thread 0:

```
    x0.store(1, memory_order_release);
    x1.store(1, memory_order_release);
```

Thread 1:

```
    r1 = x1.load(memory_order_acquire);
    r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
    x2.store(1, memory_order_relaxed);
    atomic_thread_fence(memory_order_seq_cst);
    r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3`

A More Ornate Example (W+WRC)

Thread 0:

```
    x0.store(1, memory_order_release);
    x1.store(1, memory_order_release);
```

Thread 1:

```
    r1 = x1.load(memory_order_acquire);
    r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
    x2.store(1, memory_order_relaxed);
    atomic_thread_fence(memory_order_seq_cst);
    r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3` **Still can happen!**

A More Ornate Example (W+WRC)

Thread 0:

```
    x0.store(1, memory_order_release);
    x1.store(1, memory_order_release);
```

Thread 1:

```
    r1 = x1.load(memory_order_acquire);
    r2 = x2.load(memory_order_acquire);
```

Thread 2:

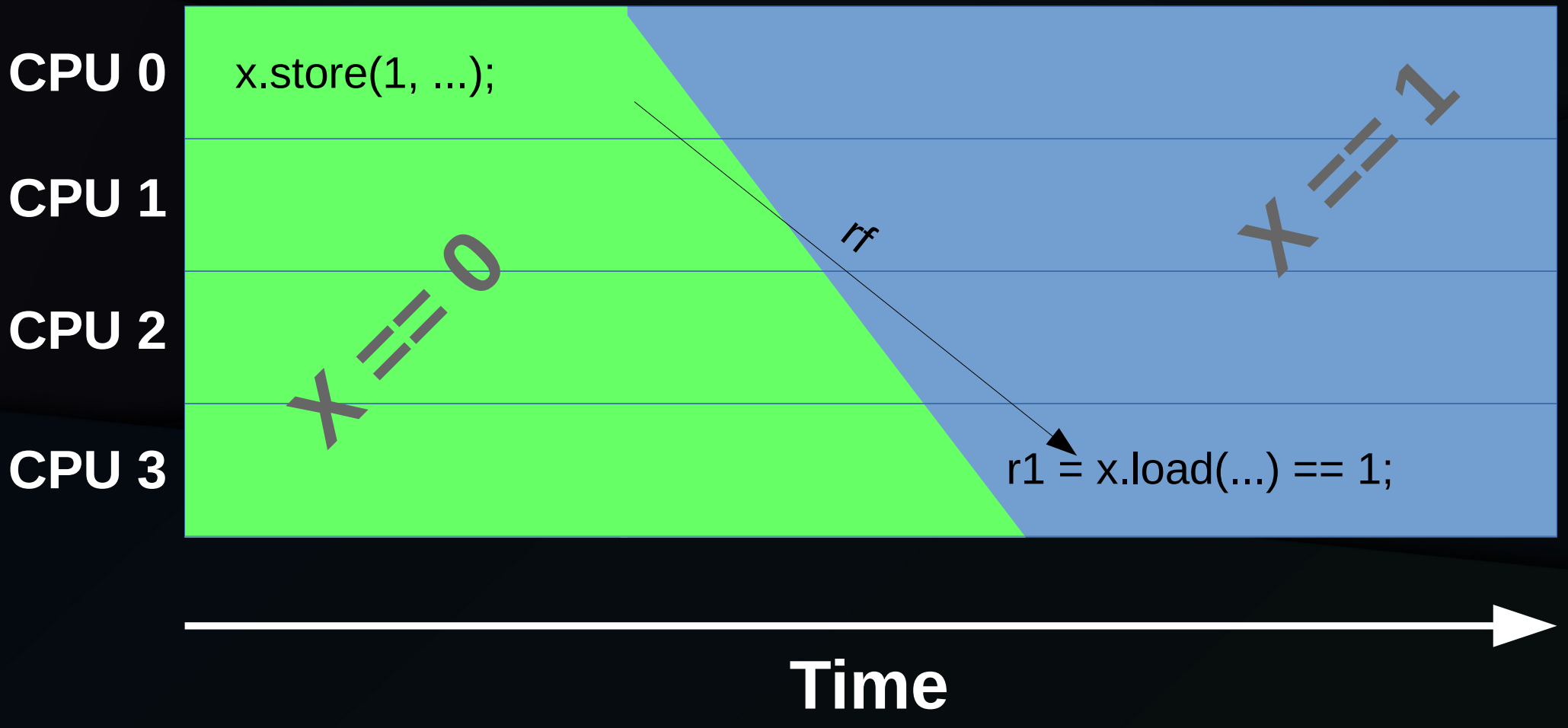
```
    x2.store(1, memory_order_relaxed);
    atomic_thread_fence(memory_order_seq_cst);
    r3 = x0.load(memory_order_acquire);
```

No ordering???

Outcome of interest: `r1 && !r2 && !r3` **Still can happen!**

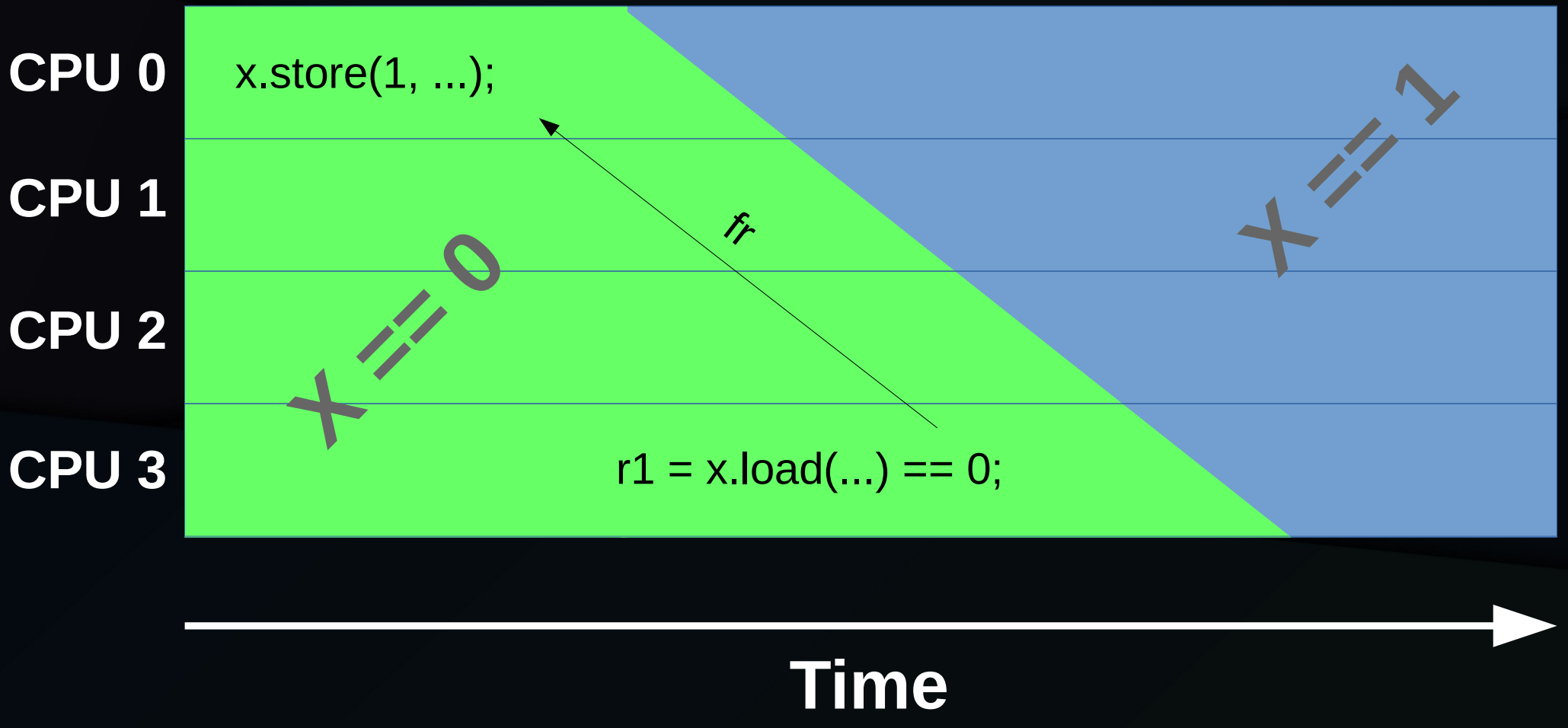
Not All Links Are Created Equal!!!

Not All Links Are Created Equal!!!



rf: “Reads From”, one process wrote the value that the next process read: Temporal

Not All Links Are Created Equal!!!



fr: “From Read”, one process read the value before the next process wrote: ~~Temporal~~

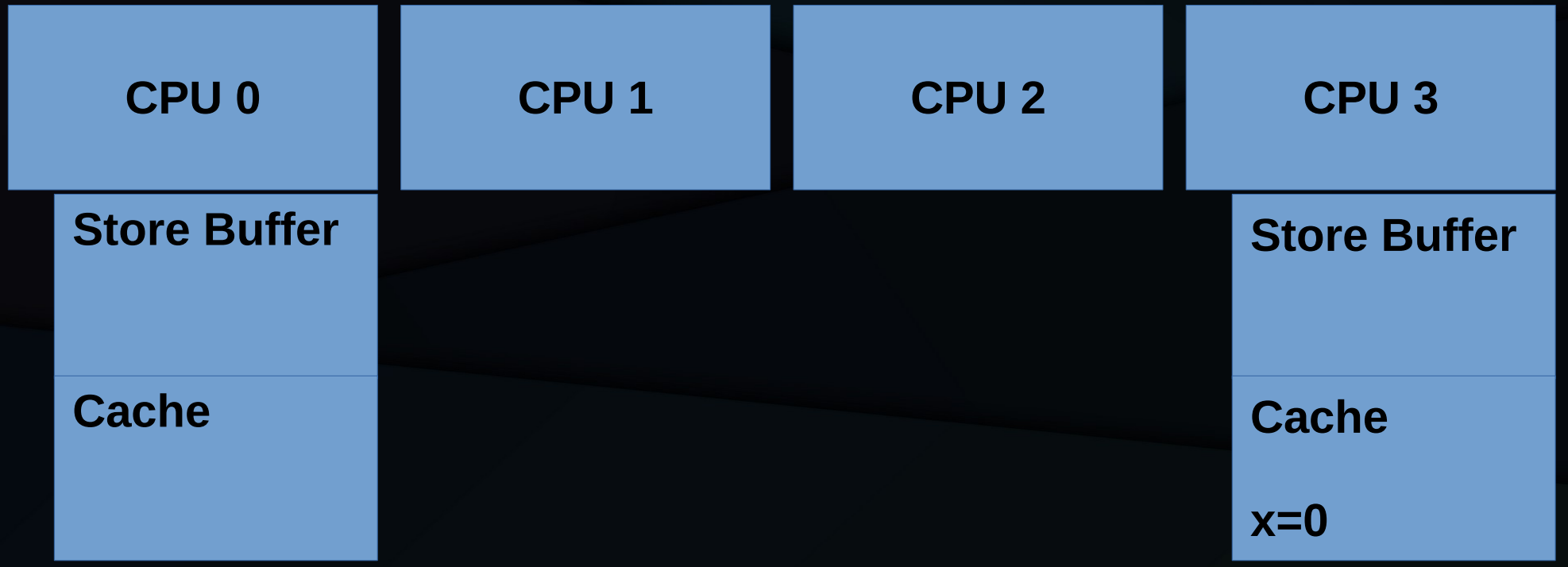
How Can This Happen???

- Cache line for variable x is initially at CPU 3
- CPU 0 writes 1 to x, but doesn't have cacheline
  - So holds the write in CPU 0's store buffer
  - And requests exclusive access to the cacheline (which takes time)
- CPU 3 reads x, obtaining “0” immediately from cacheline
- CPU 0 receive's x's cacheline
  - And CPU 0's write finally gets to the cacheline
  - Overwriting the value that CPU 3 read, despite the write starting earlier
- Once again, writes are not instantaneous!!!

The Role of Store Buffers (1/7)

x.store(1, ..._relaxed)

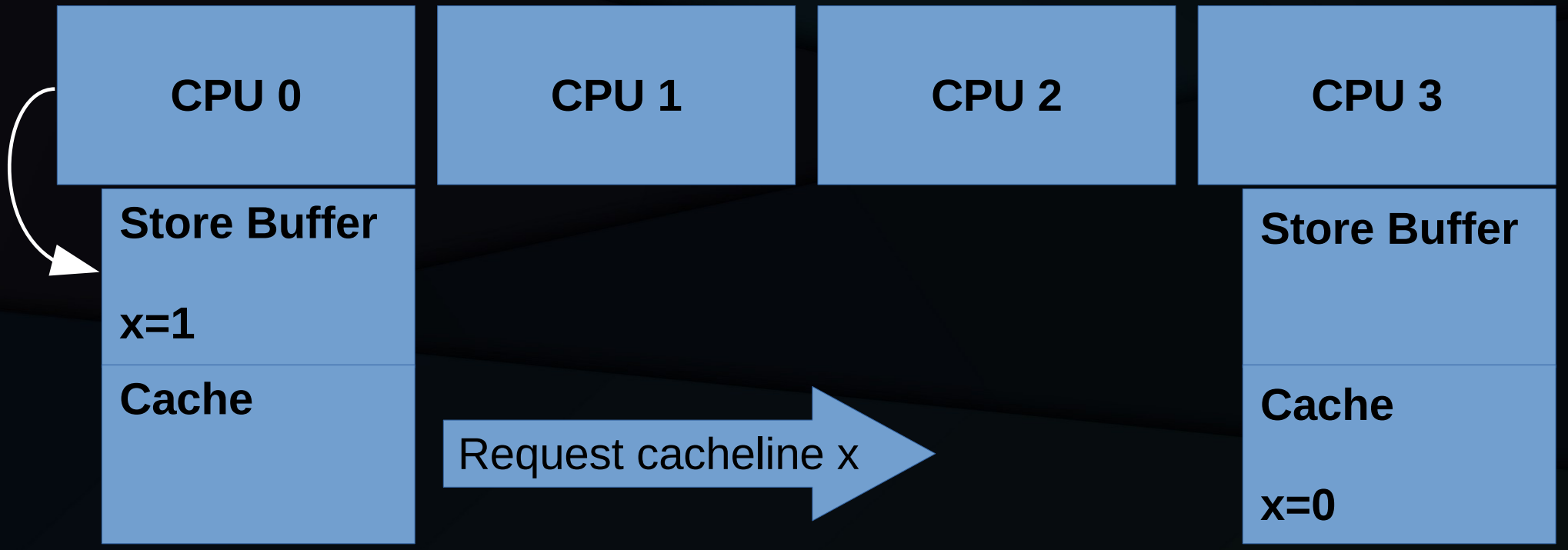
x.load(..._relaxed)



The Role of Store Buffers (2/7)

~~x.store(1, ..._relaxed)~~

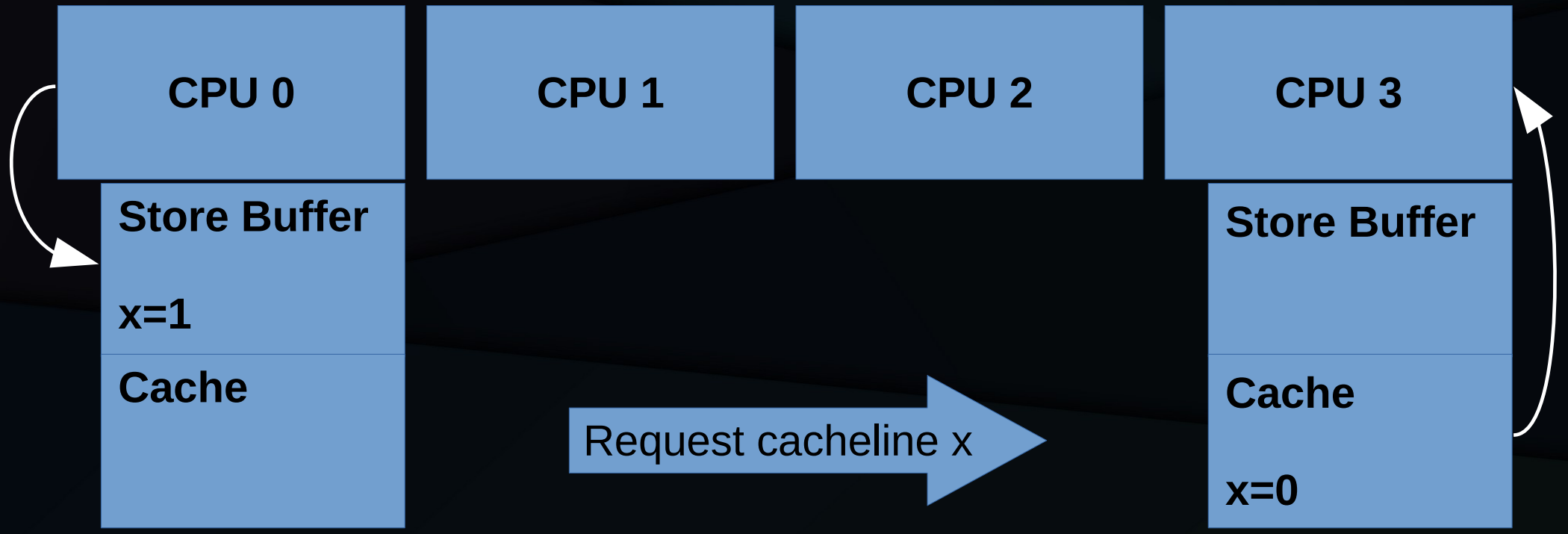
x.load(..._relaxed)



The Role of Store Buffers (3/7)

~~x.store(1, ..._relaxed)~~

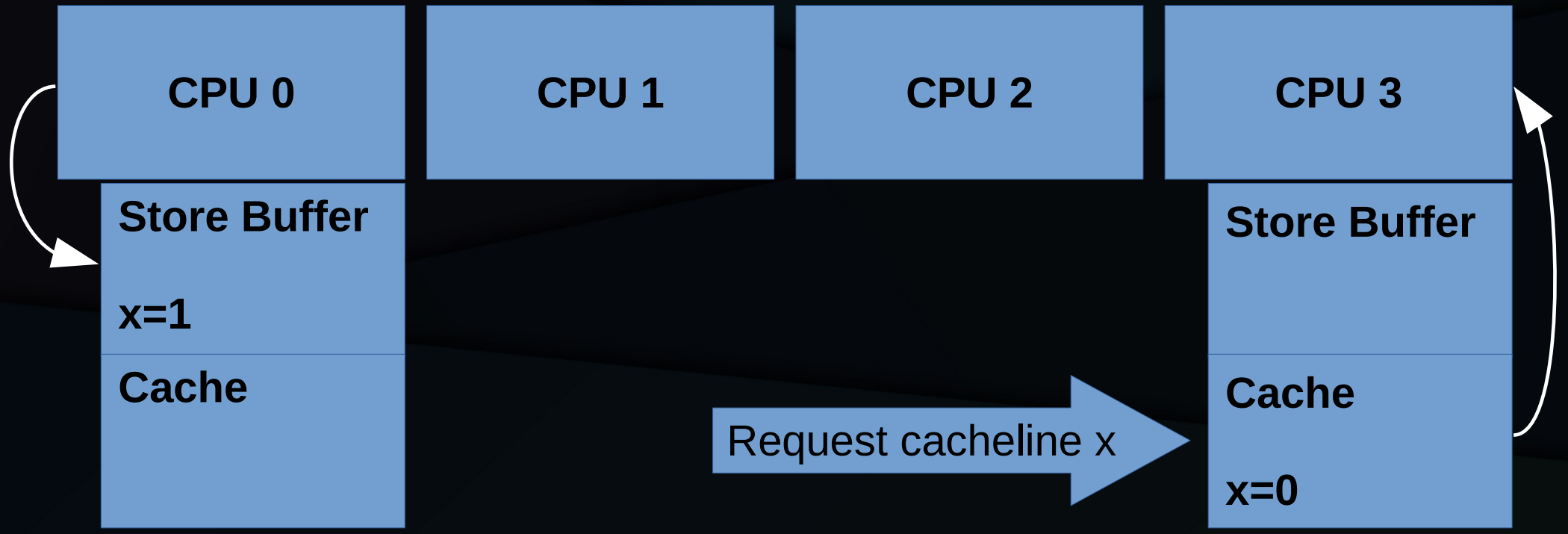
~~x.load(..._relaxed)~~



The Role of Store Buffers (4/7)

~~x.store(1, ..._relaxed)~~

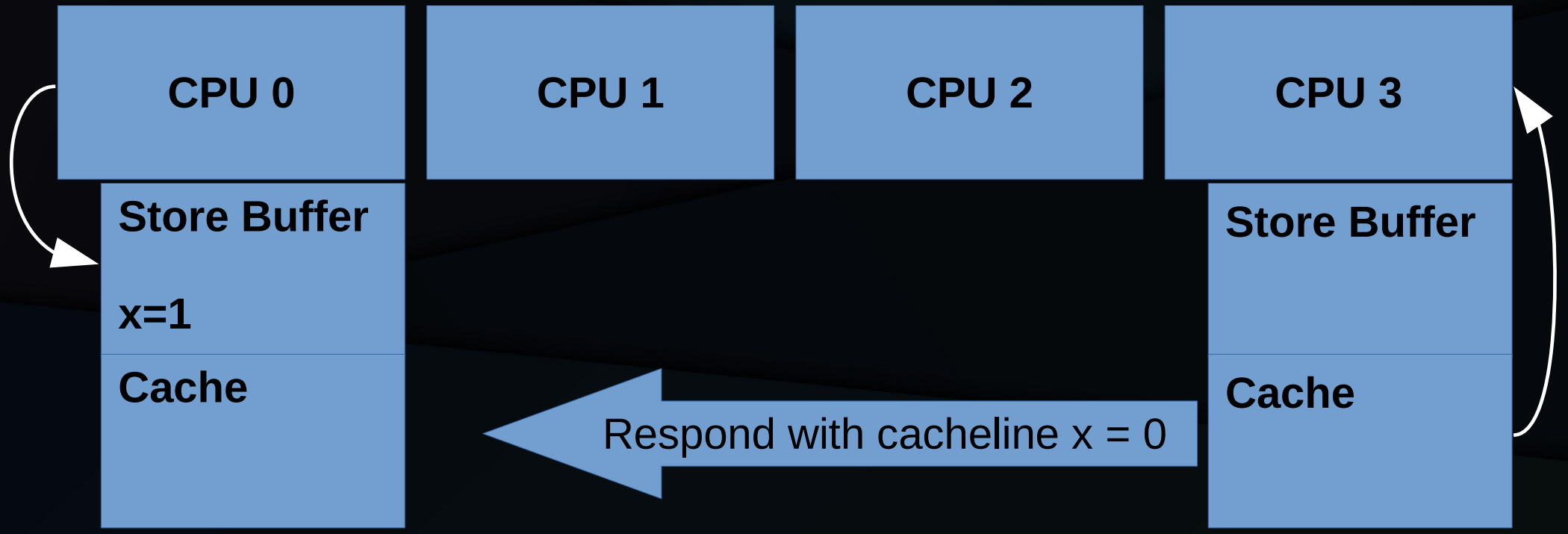
~~x.load(..._relaxed)~~



The Role of Store Buffers (5/7)

~~x.store(1, ..._relaxed)~~

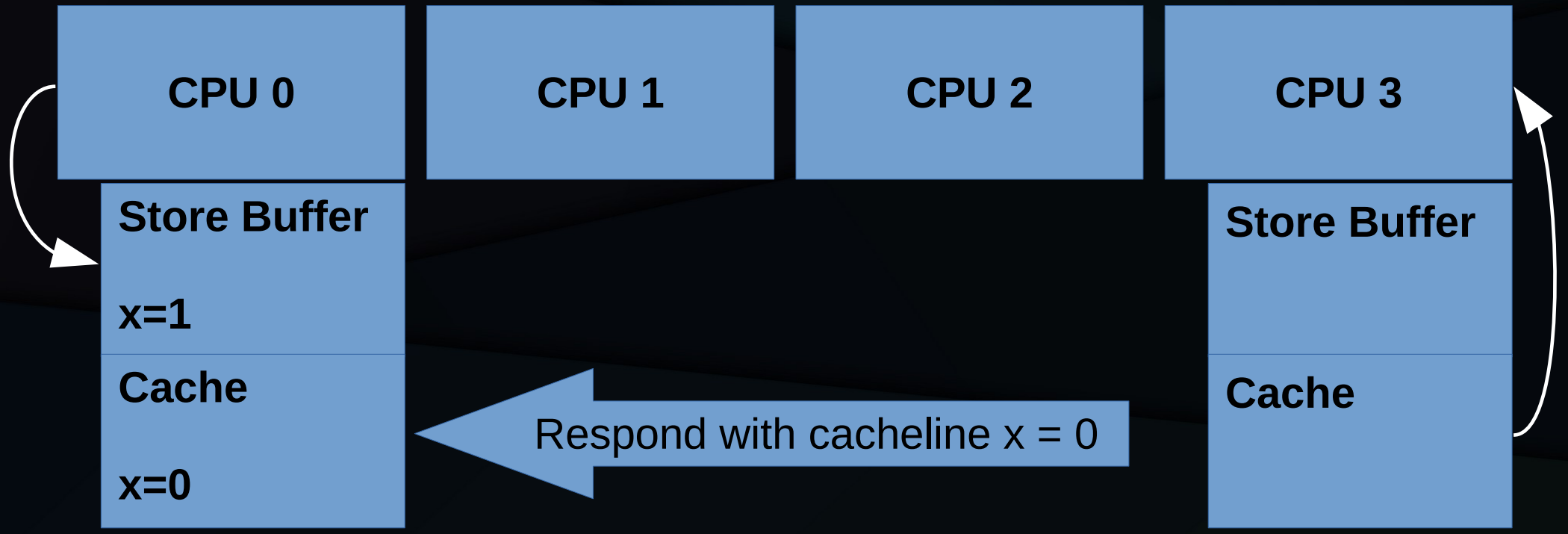
~~x.load(..._relaxed)~~



The Role of Store Buffers (6/7)

~~x.store(1, ..._relaxed)~~

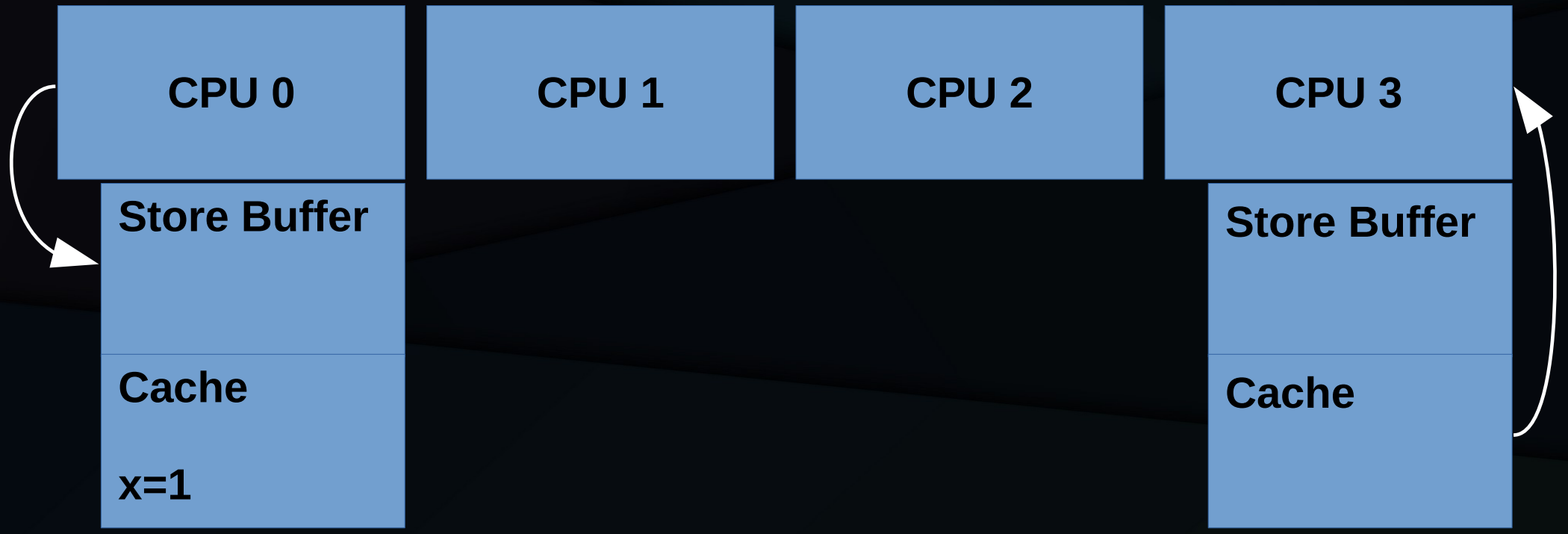
~~x.load(..._relaxed)~~



The Role of Store Buffers (7/7)

~~x.store(1, ..._relaxed)~~

~~x.load(..._relaxed)~~



Again, the speed of light is finite and atoms are of non-zero size!!!

So Add Another Memory Fence!!!

A More Ornate Example (W+WRC)

Thread 0:

```
x0.store(1, memory_order_release);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
atomic_thread_fence(memory_order_seq_cst);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
atomic_thread_fence(memory_order_seq_cst);
r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3`

A More Ornate Example (W+WRC)

Thread 0:

```
    x0.store(1, memory_order_release);
    x1.store(1, memory_order_release);
```

Thread 1:

```
    r1 = x1.load(memory_order_acquire);
    atomic_thread_fence(memory_order_seq_cst);
    r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
    x2.store(1, memory_order_relaxed);
    atomic_thread_fence(memory_order_seq_cst);
    r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3` ***Still*** **can happen!!!**

A More Ornate Example (W+WRC)

Thread 0:

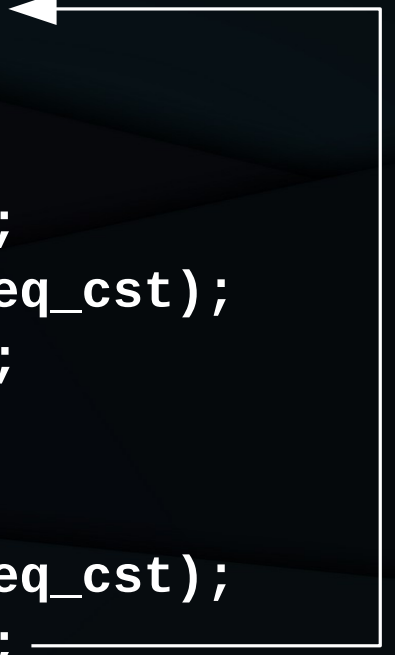
```
x0.store(1, memory_order_release);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
atomic_thread_fence(memory_order_seq_cst);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
atomic_thread_fence(memory_order_seq_cst);
r3 = x0.load(memory_order_acquire);
```



No ordering???

Outcome of interest: `r1 && !r2 && !r3` ***Still* can happen!!!**

OK, No More Mr. Nice Guy...

Thread 0:

```
x0.store(1, memory_order_release);
atomic_thread_fence(memory_order_seq_cst);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
atomic_thread_fence(memory_order_seq_cst);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
atomic_thread_fence(memory_order_seq_cst);
r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3`

OK, No More Mr. Nice Guy...

Thread 0:

```
x0.store(1, memory_order_release);
atomic_thread_fence(memory_order_seq_cst);
x1.store(1, memory_order_release);
```

Thread 1:

```
r1 = x1.load(memory_order_acquire);
atomic_thread_fence(memory_order_seq_cst);
r2 = x2.load(memory_order_acquire);
```

Thread 2:

```
x2.store(1, memory_order_relaxed);
atomic_thread_fence(memory_order_seq_cst);
r3 = x0.load(memory_order_acquire);
```

Outcome of interest: `r1 && !r2 && !r3` **Finally, cannot happen!!!**

But Please Don’t Hack Concurrency

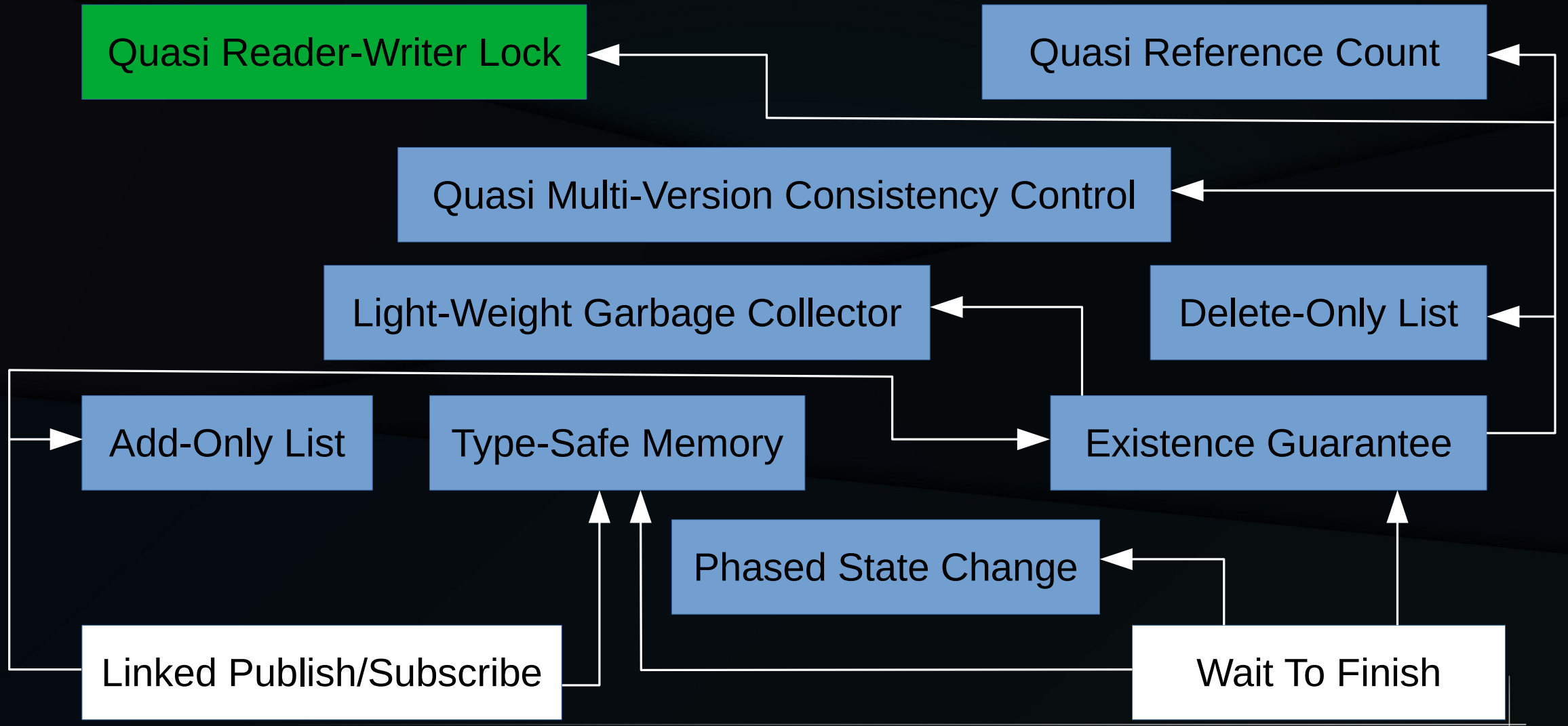
- Have a solid design up front
- Use tools to check your design
- Only then start coding

But Please Don’t Hack Concurrency

- Have a solid design up front
- Use tools to check your design
- Only then start coding
- Concurrency uses your intelligence against you
  - The smarter you are, the deeper a hole you will dig for yourself before you realize you are in trouble

RCU Use Cases

RCU Use Cases



Example Multithreaded Use Case

Hide

Example Multithreaded Use Case

- Configuration information in variables `a` and `b`:
  - `int a, b; // Current configuration values`
  - Infrequently updated based on external inputs
  - Given reader access needs consistent values
- Reading “Oldish” values OK, ***if*** consistent
- Very frequent reader access to `a` and `b`

Reader-Writer Locked Use Case

Reader-Writer Locked Use Case

```
std::shared_mutex myrwlock;
int a, b; // Current configuration values

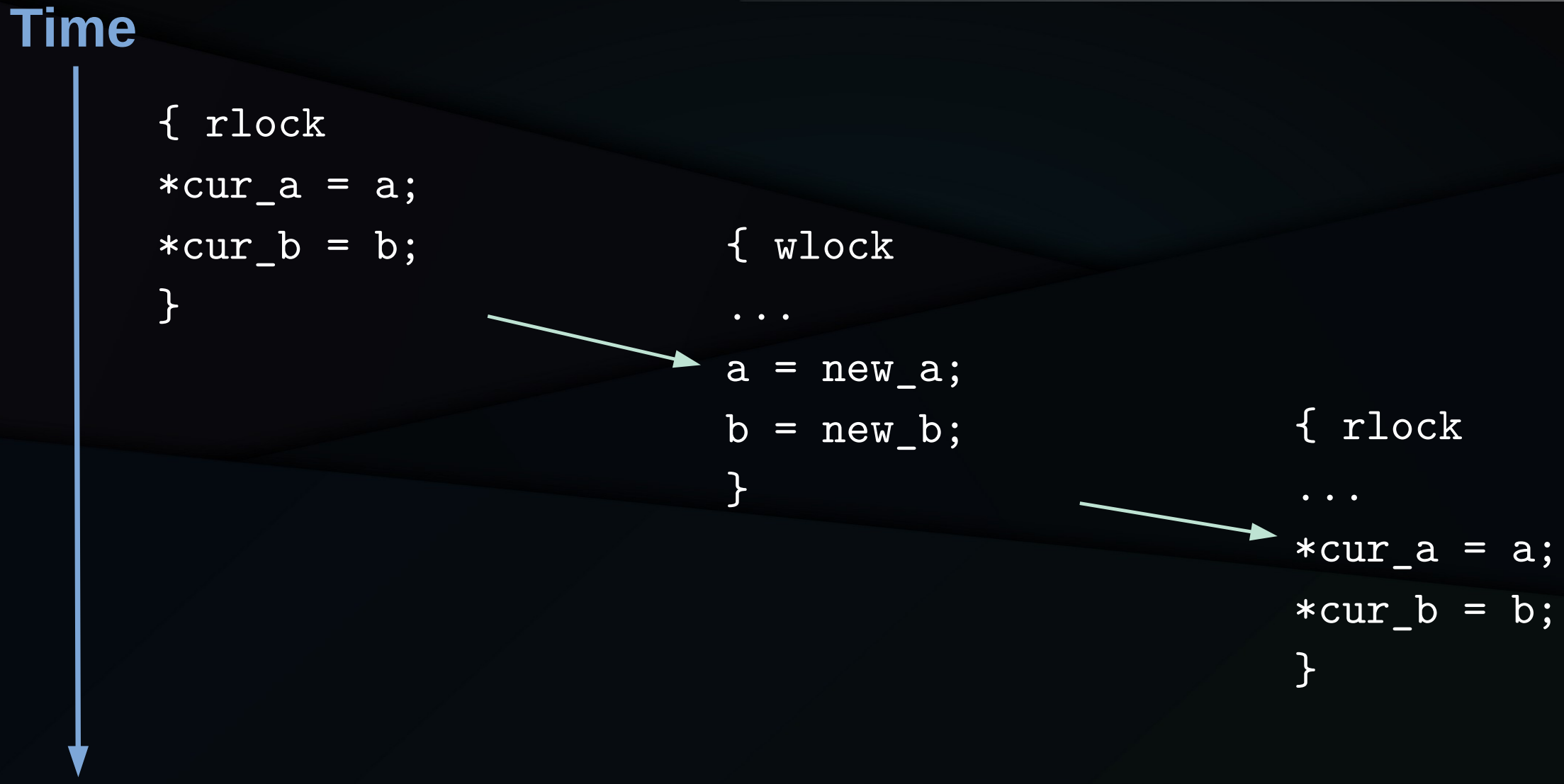
void get(int *cur_a, int *cur_b)
{
    std::shared_lock<std::shared_mutex> rlock(myrwlock);
    *cur_a = a;
    *cur_b = b;
}
```

Reader-Writer Locked Use Case

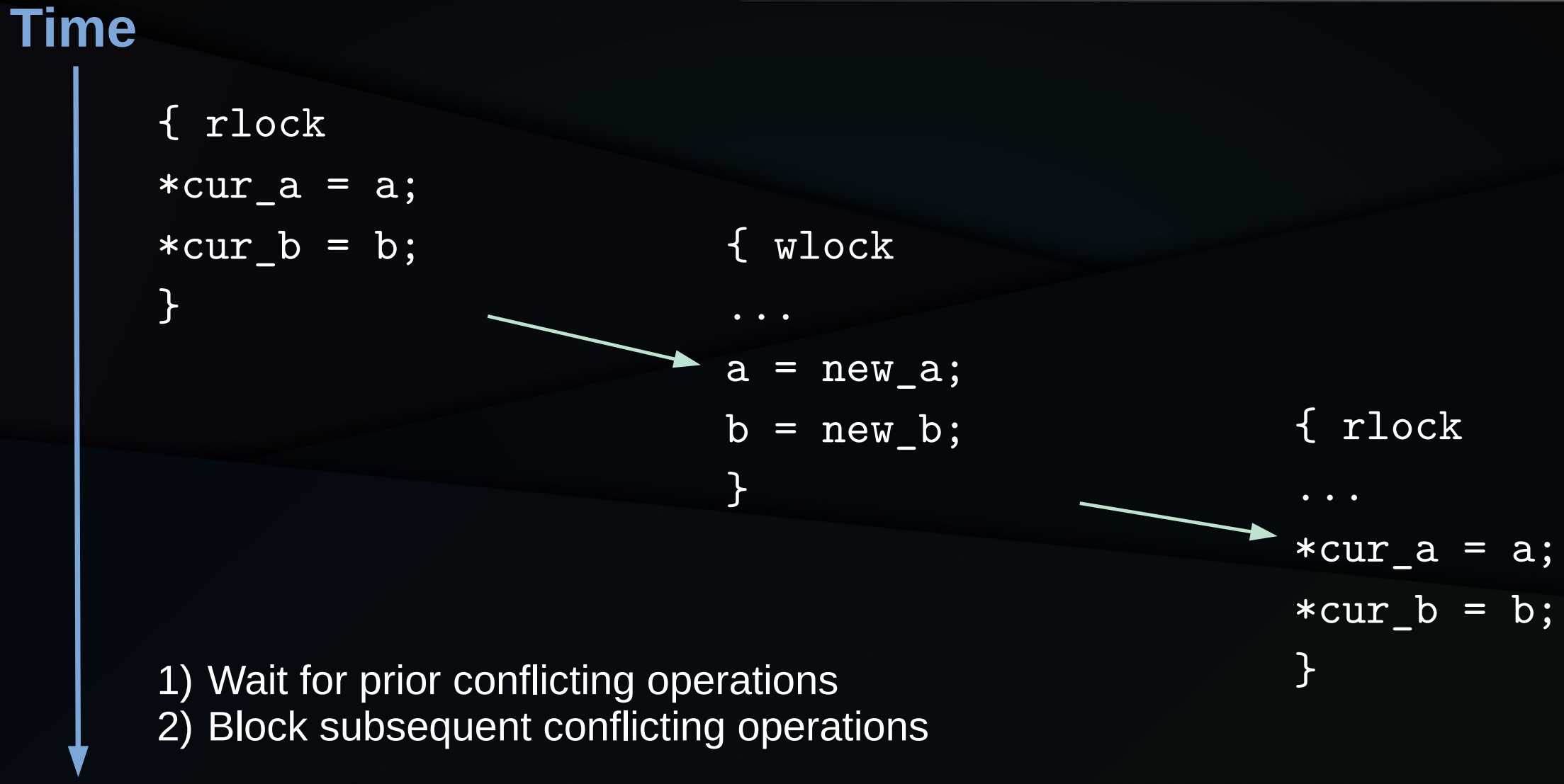
```
void set(int new_a, int new_b)
{
    std::unique_lock<std::shared_mutex> wlock(myrwlock);
    a = new_a;
    b = new_b;
}
```

Reader-Writer Lock Semantics

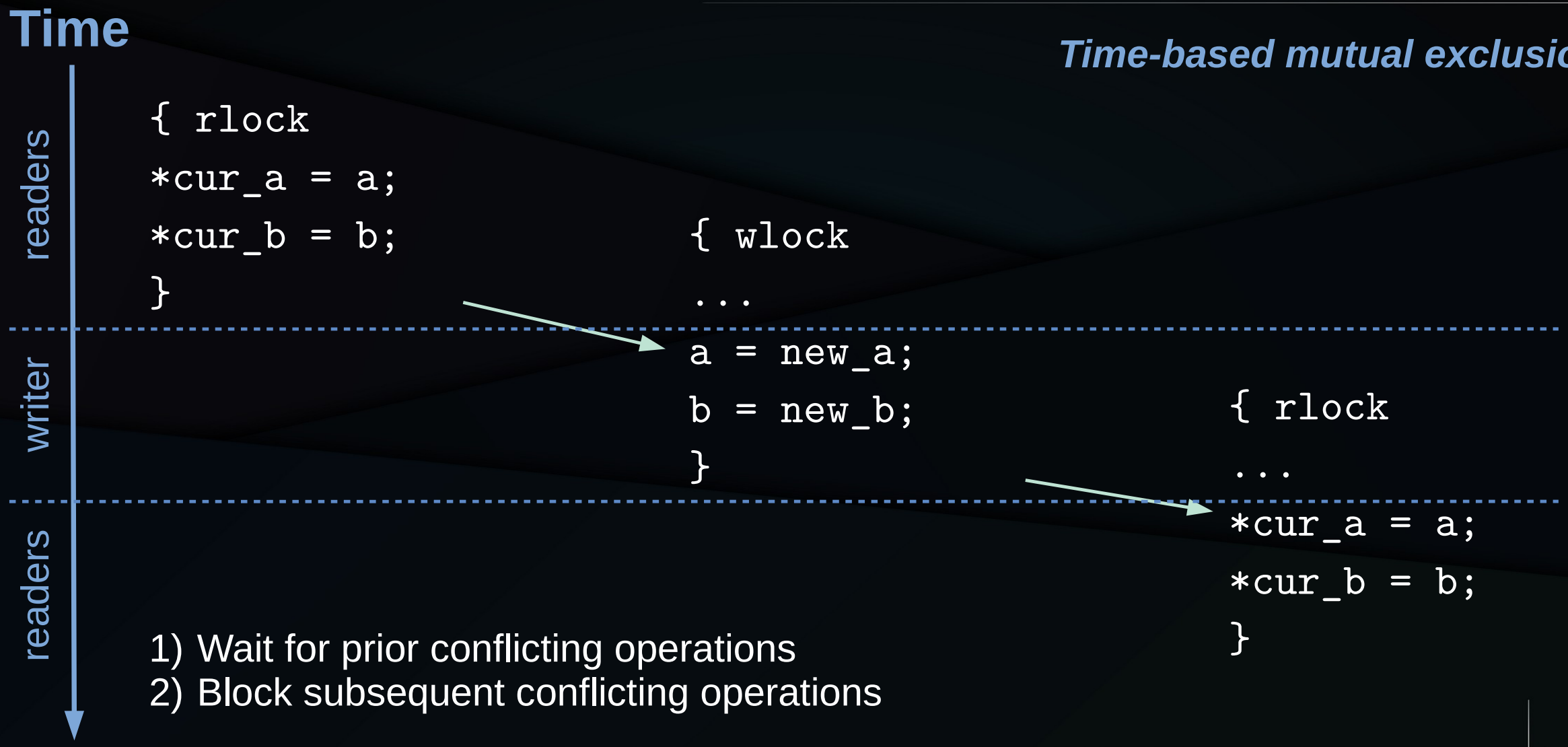
Reader-Writer Lock Semantics



Reader-Writer Lock Semantics



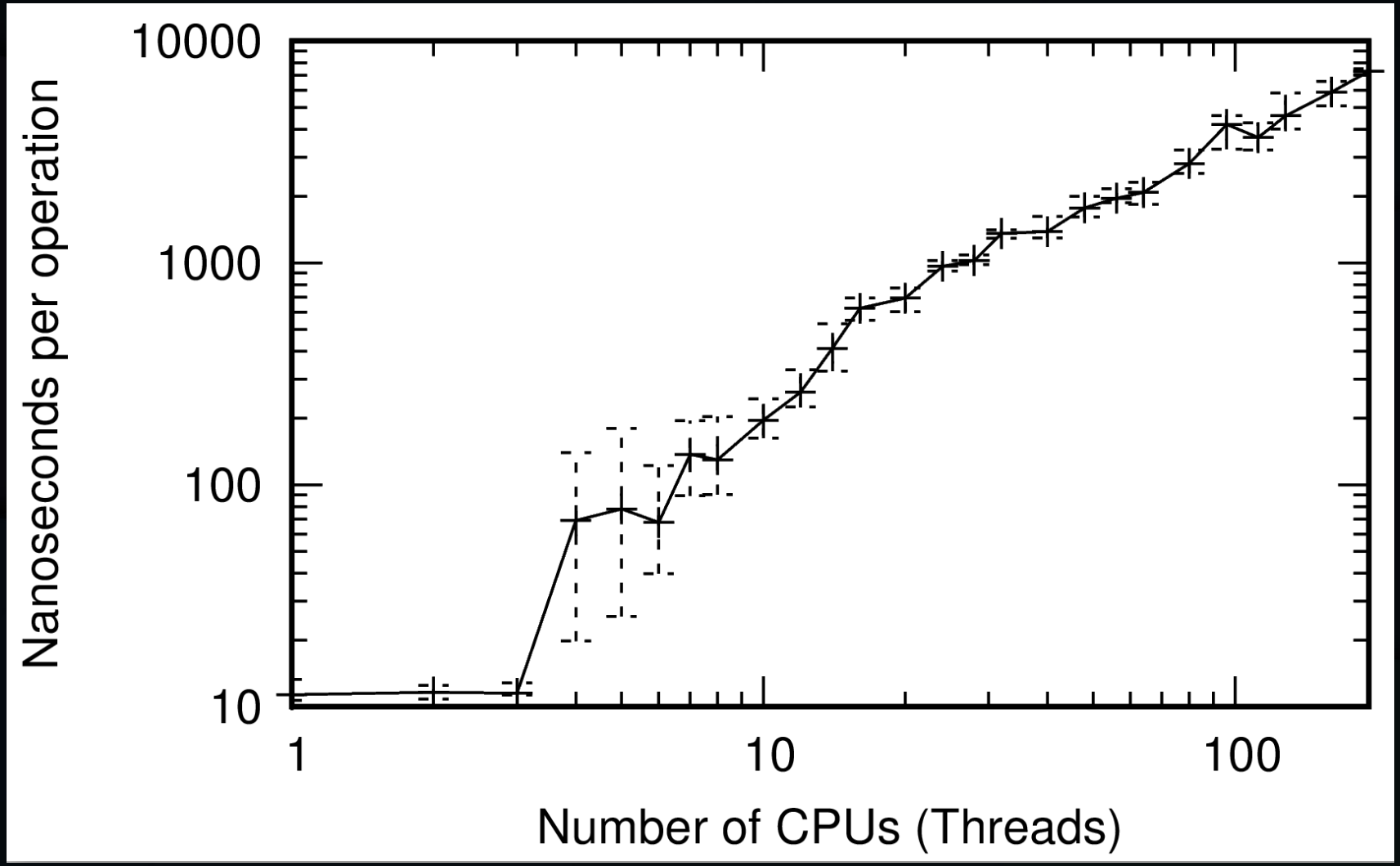
Reader-Writer Lock Semantics



Time-based mutual exclusion

Reader-Writer Lock Performance

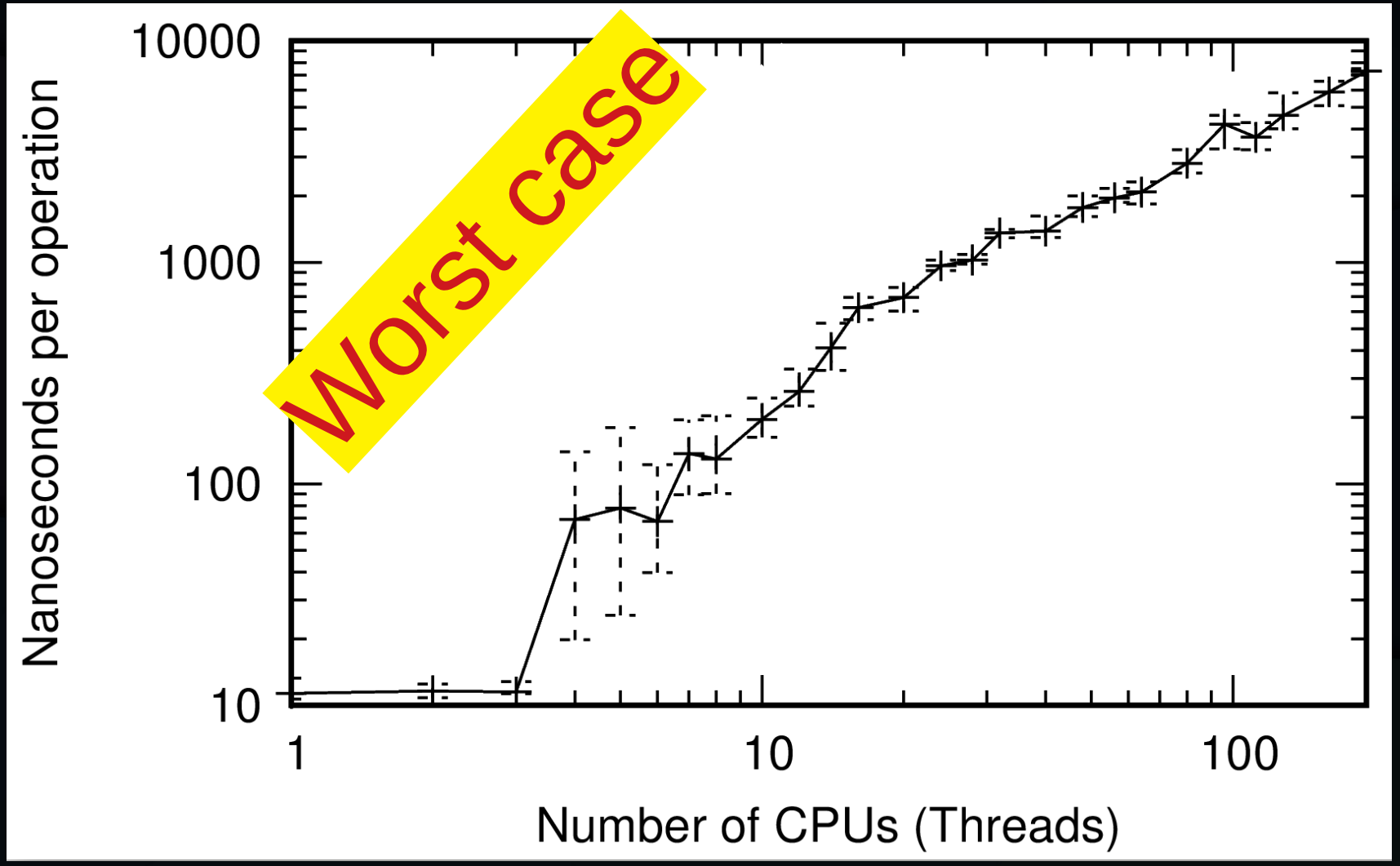
Reader-Writer Lock Performance



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/defer/data/rcuscale.hps.2020.05.28a/rwlockperf.eps

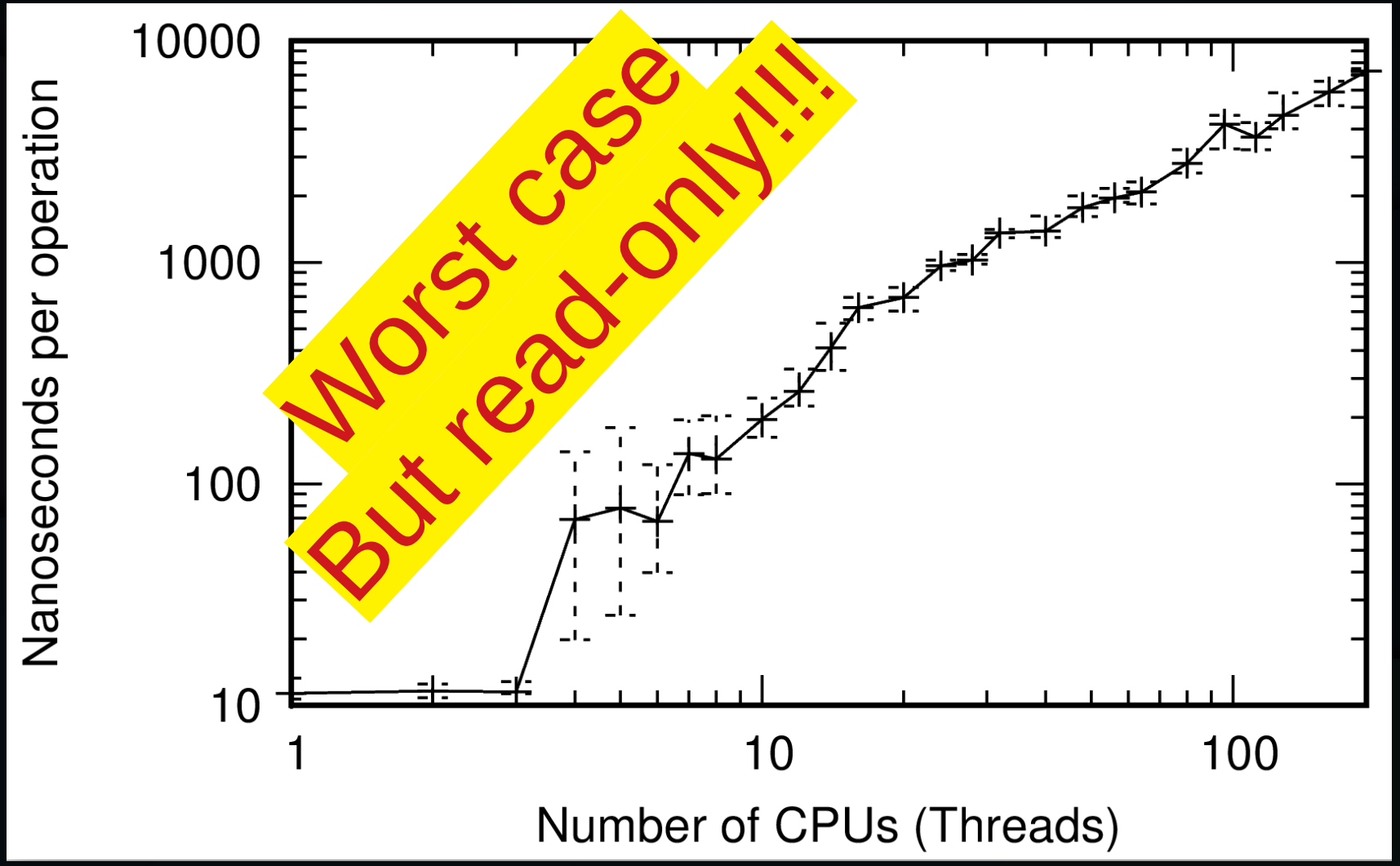
Reader-Writer Lock Performance



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/defer/data/rcuscale.hps.2020.05.28a/rwlockperf.eps

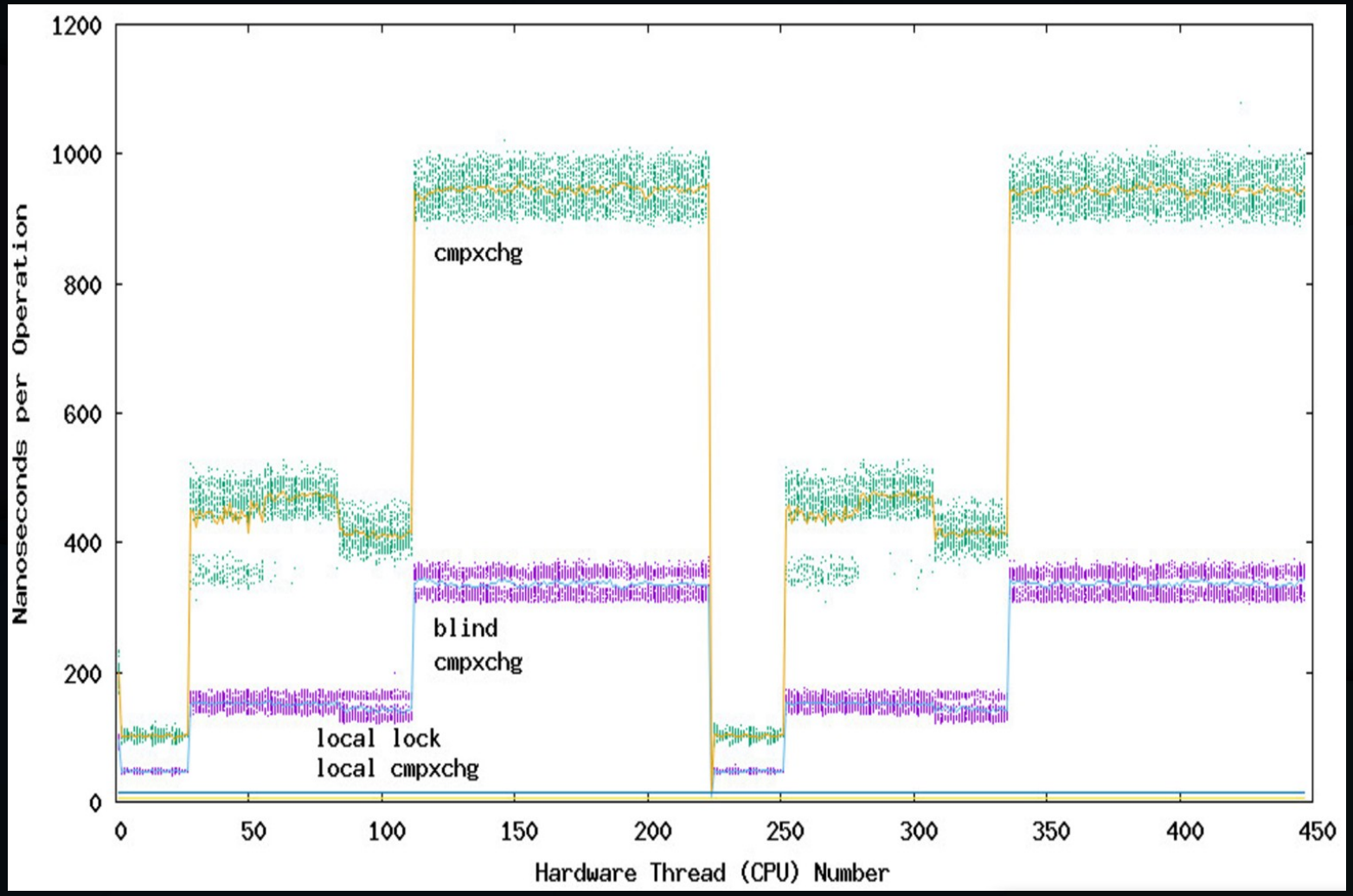
Reader-Writer Lock Performance



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/defer/data/rcuscale.hps.2020.05.28a/rwlockperf.eps

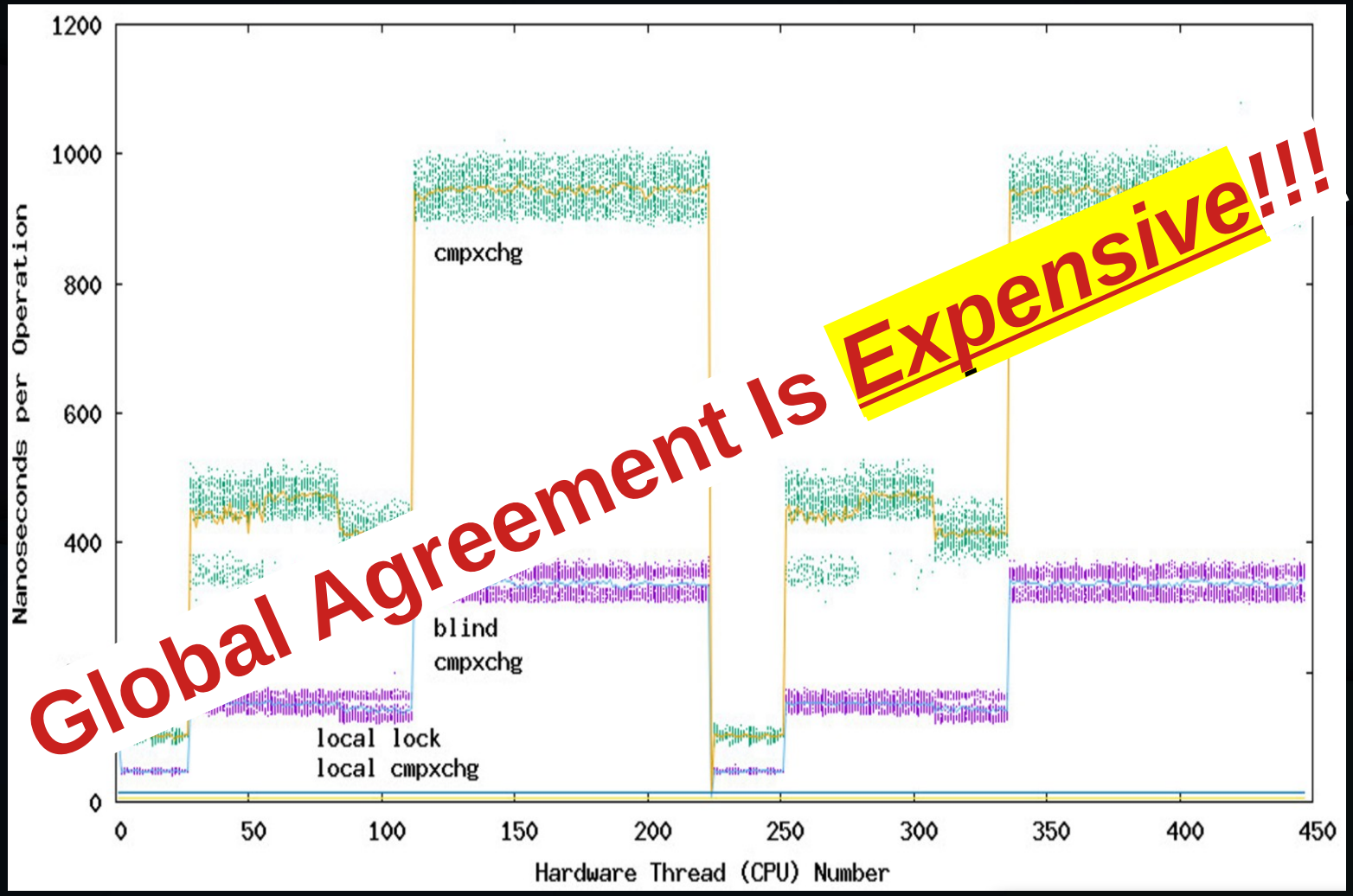
Not Just Contention: HW Latency



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/cpu/data/hps.2020.03.04b/cachetorture-latency-scatter.jpg

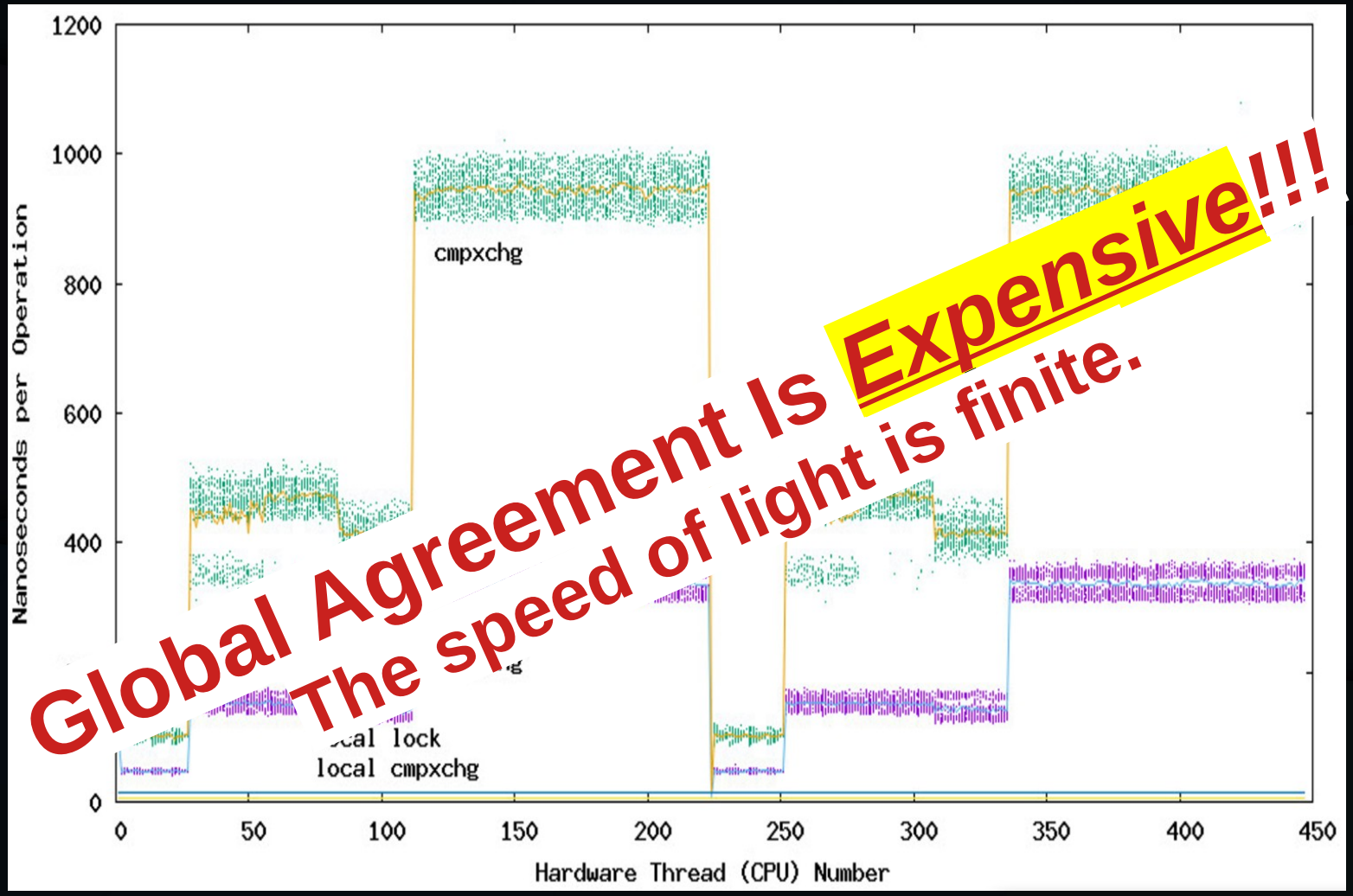
Not Just Contention: HW Latency



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/cpu/data/hps.2020.03.04b/cachetorture-latency-scatter.jpg

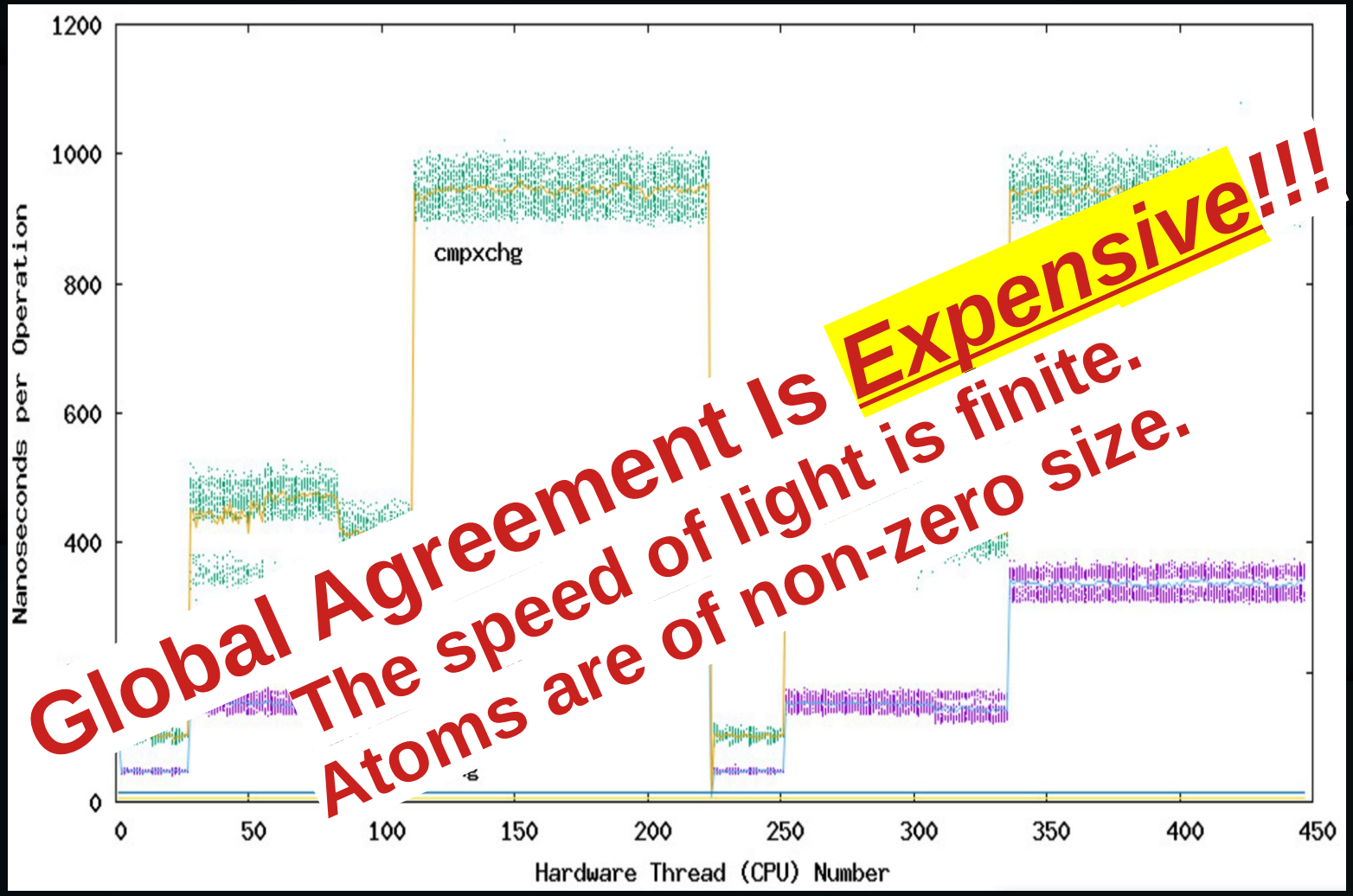
Not Just Contention: HW Latency



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/cpu/data/hps.2020.03.04b/cachetorture-latency-scatter.jpg

Not Just Contention: HW Latency



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

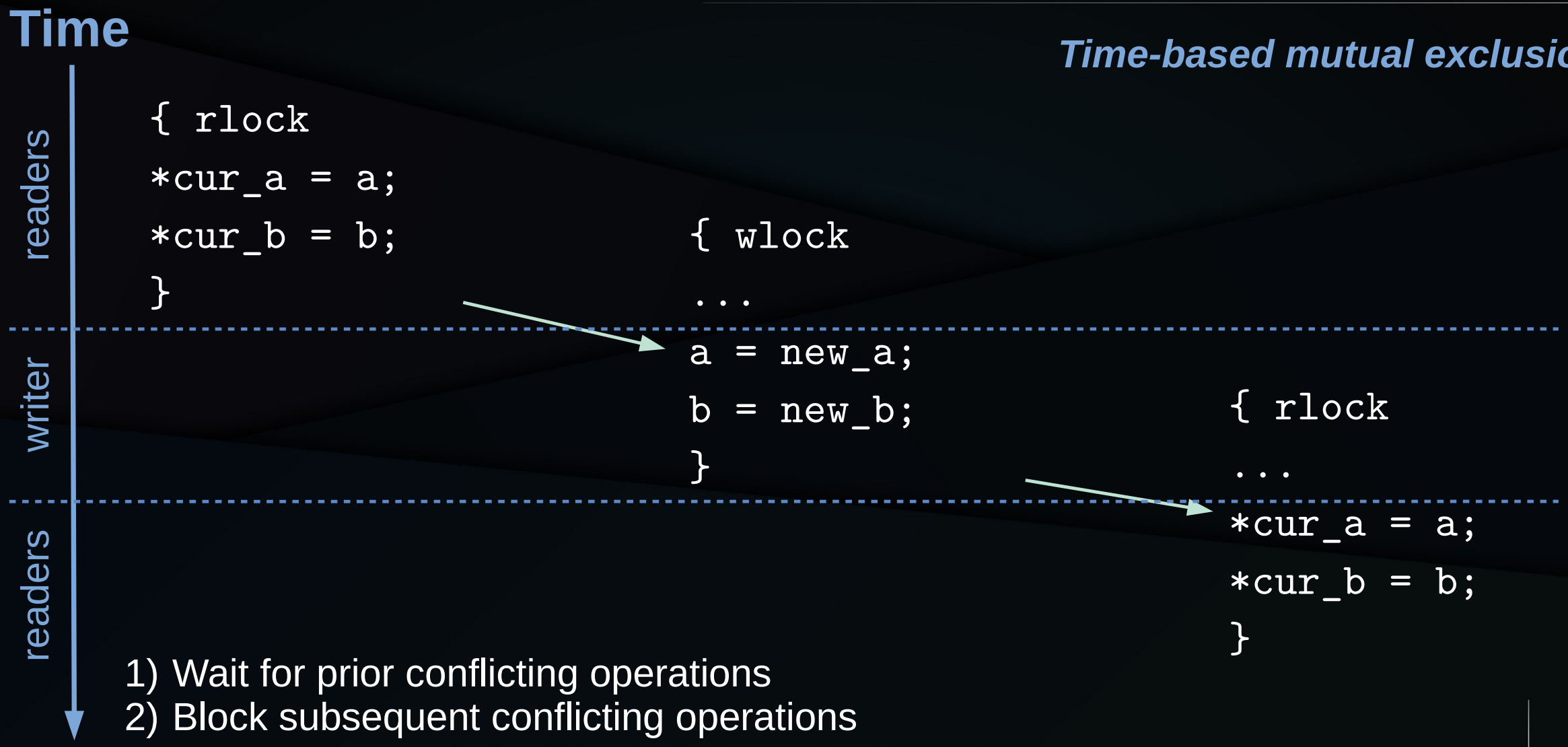
perfbook CodeSamples/cpu/data/hps.2020.03.04b/cachetorture-latency-scatter.jpg

We Must Use Another Loophole...

- 3D integration for smaller systems
- Systolic arrays for unidirectional data flow
  - Obtain similar effects with modern accelerators
- New semiconductor technologies
  - Too bad about the need to hand-place atoms...
- **Store buffers and read-only replication**

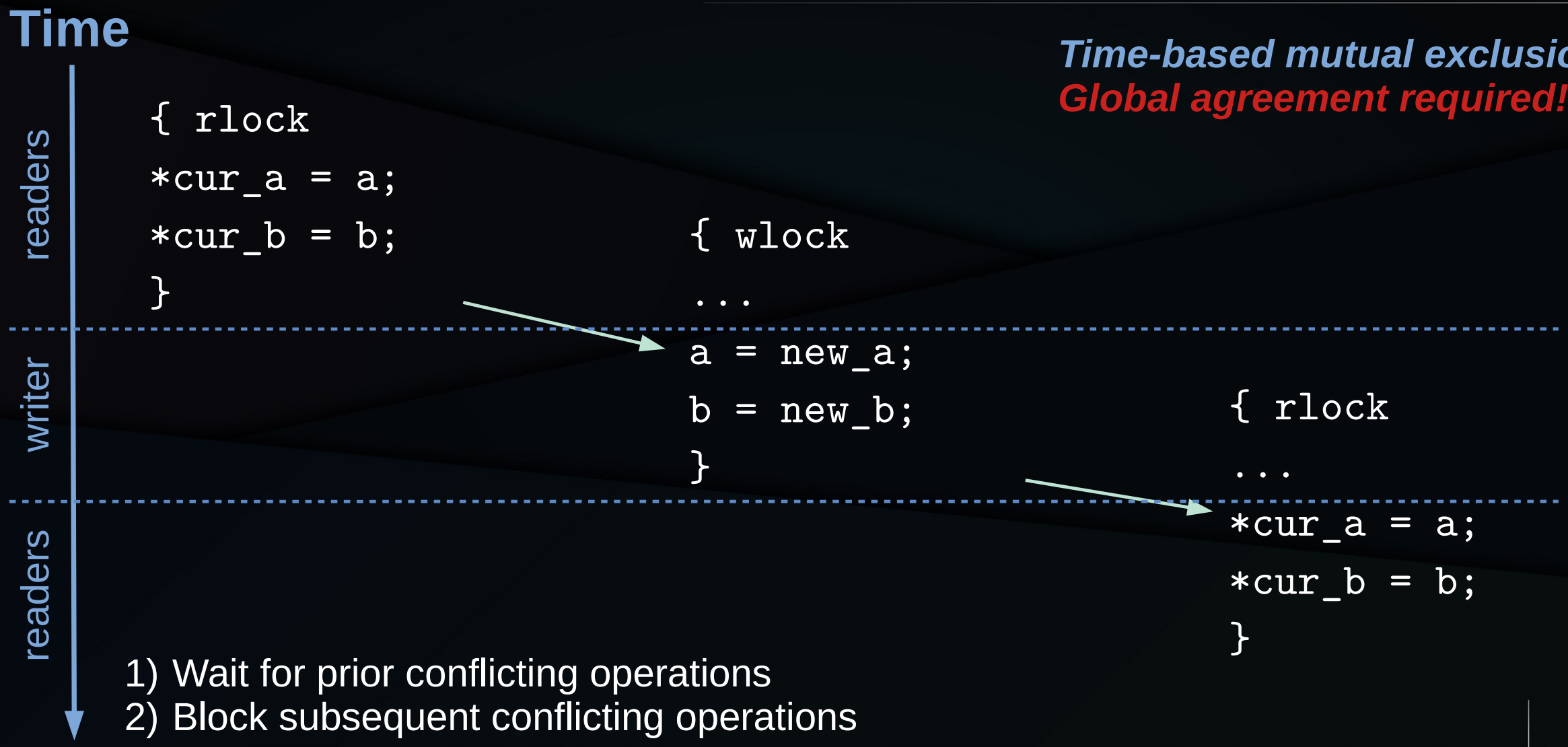
Lighter Weight Semantics

Reader-Writer Lock Semantics (Redux)



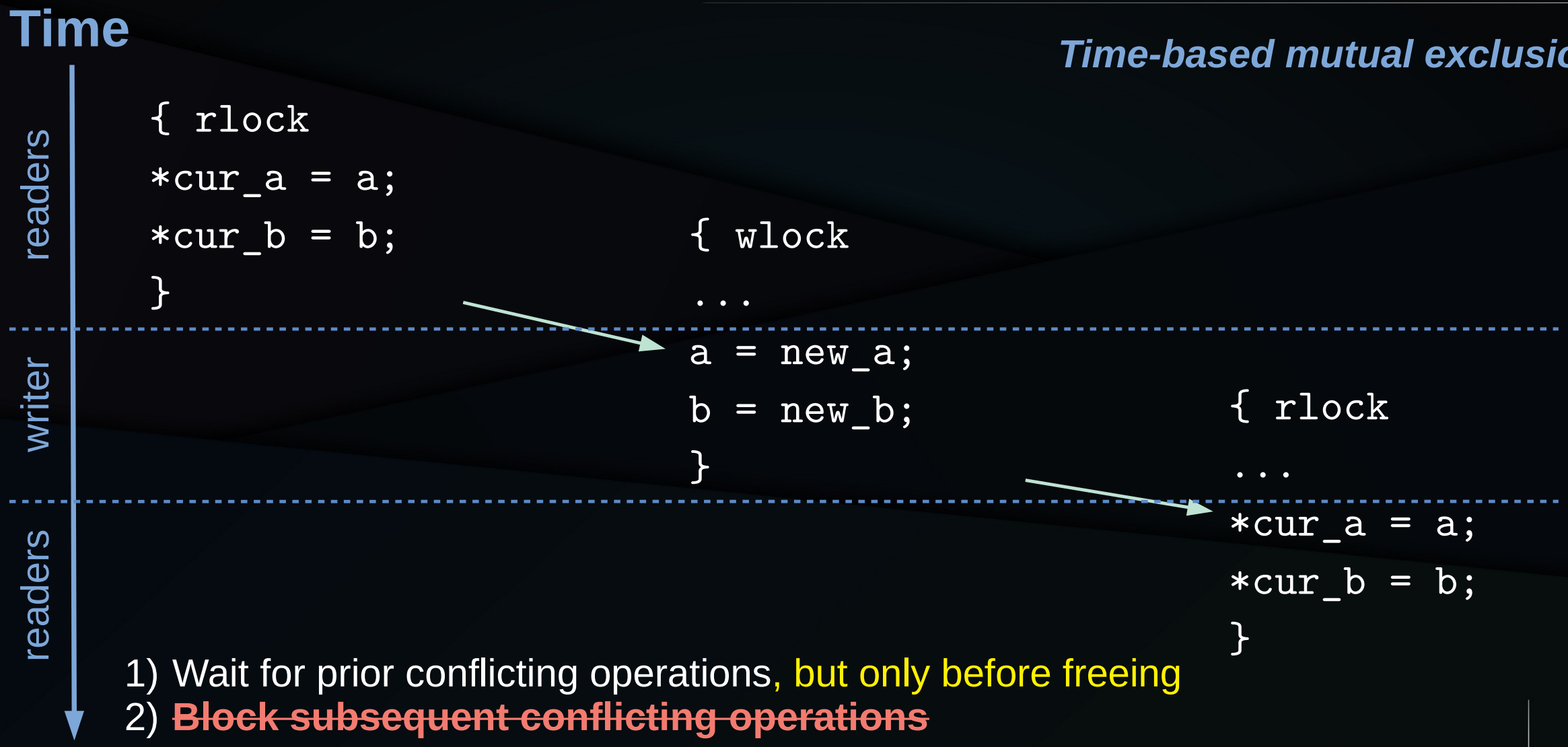
Time-based mutual exclusion

Reader-Writer Lock Semantics (Redux)

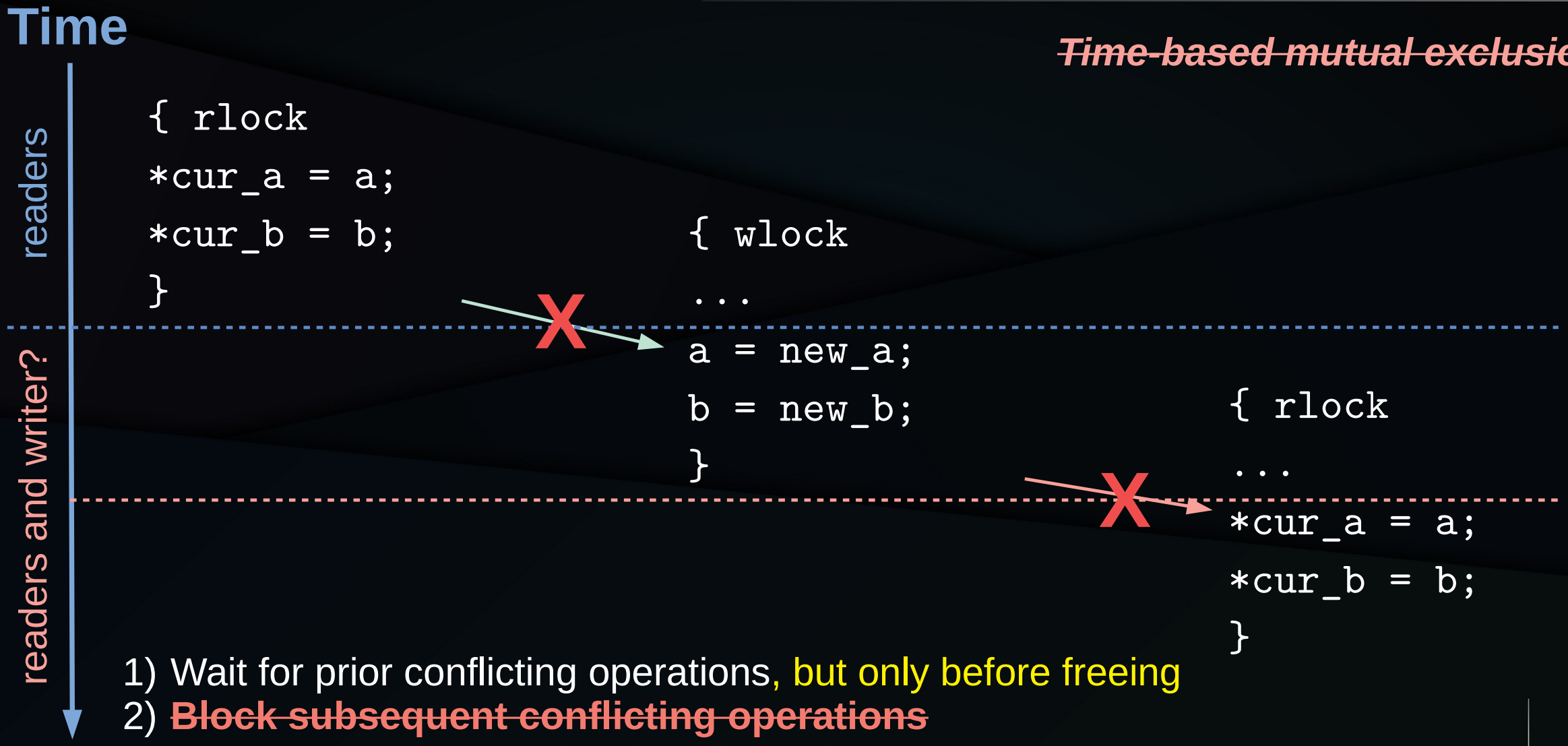


Time-based mutual exclusion:
Global agreement required!!!

Lighter Weight Semantics



Lighter Weight Semantics???



~~*Time-based mutual exclusion*~~

1) Wait for prior conflicting operations, but only before freeing
2) ~~**Block subsequent conflicting operations**~~

Core RCU API

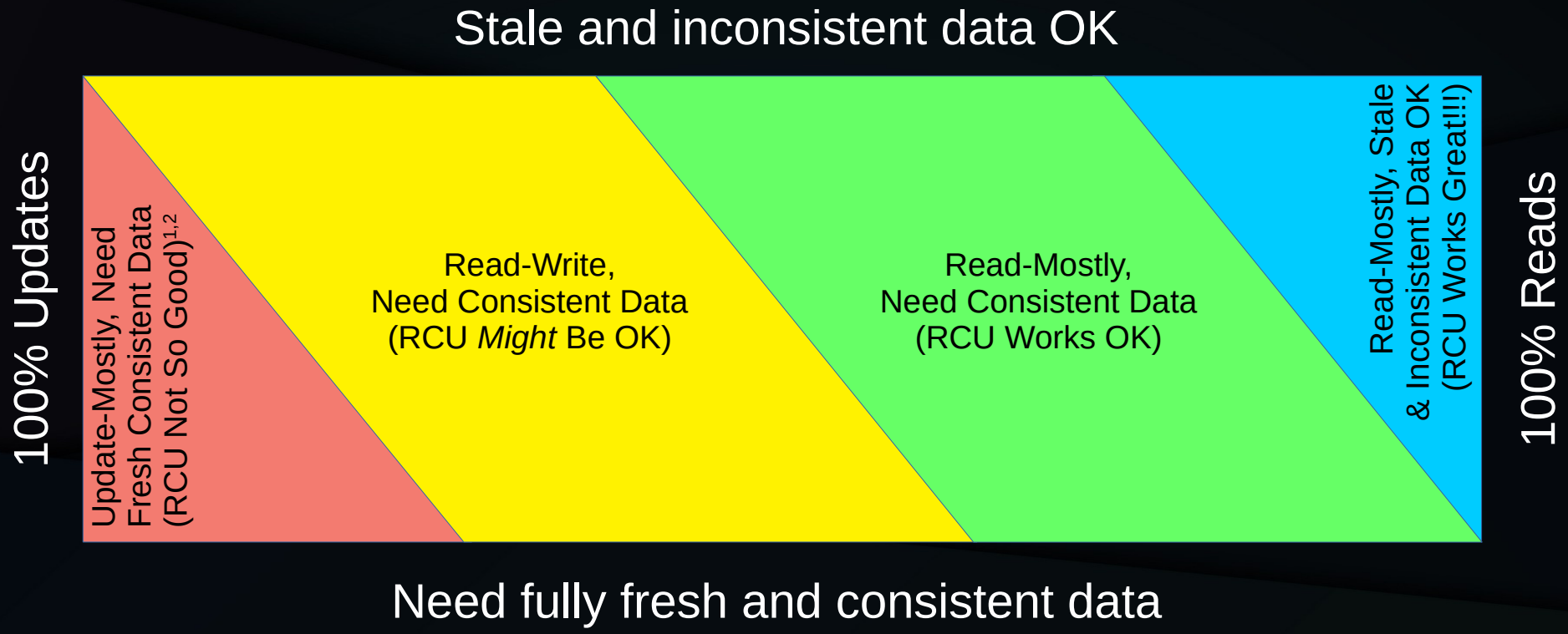
- `rcu_domain::lock`: Begin reader
- `rcu_domain::unlock`: End reader
- `rcu_synchronize()`: Wait for pre-existing readers
- `.retire()`: Invoke deleter after pre-existing readers complete
- `memory_order_consume`: Load RCU-protected pointer
- `memory_order_release`: Update RCU-protected pointer

For the full C++ RCU API as of September 2022: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2545r0.pdf

RCU Semantics (English)

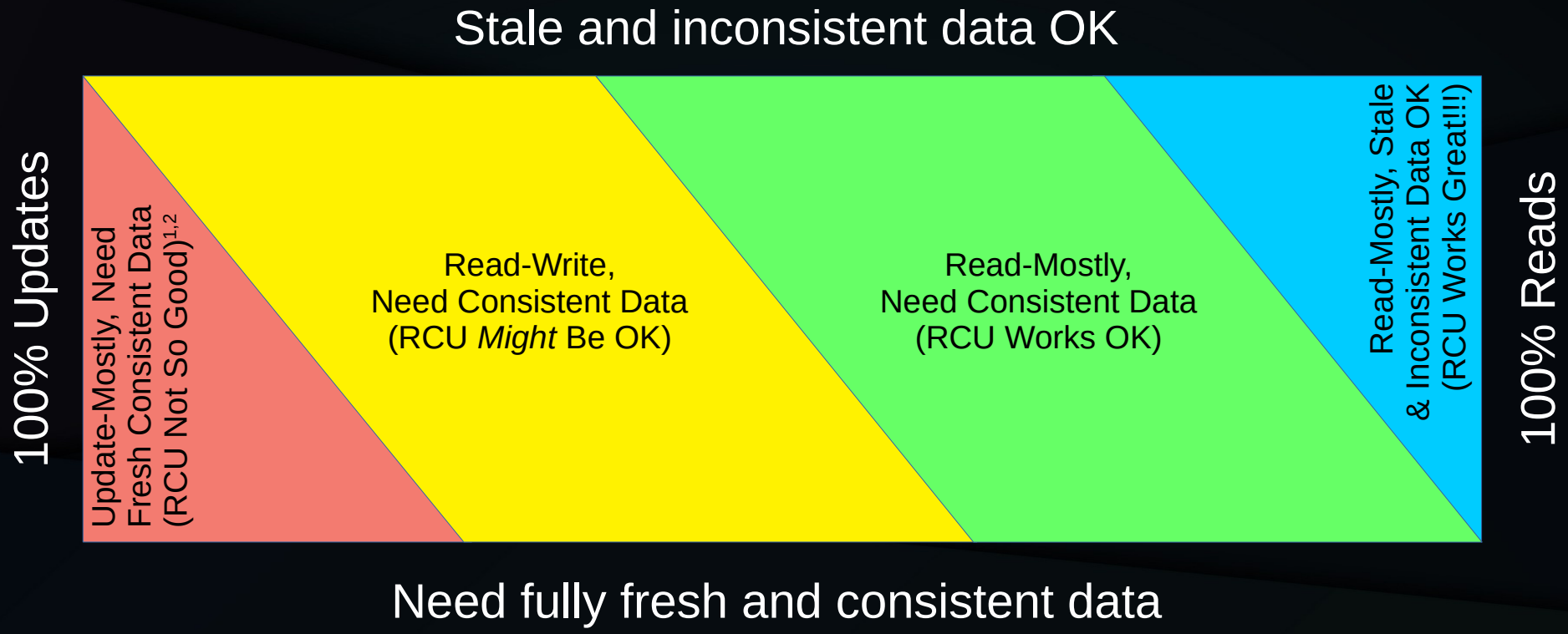
- Semantics weakened from reader-writer locking:
  1) Writers wait for read-holders, but only before freeing
  2) ~~Readers wait for the writer-holders~~
- Compensate for weak temporal semantics by adding *restrictions and spatial semantics* to RCU use cases
- Restated for RCU:
  - If rcu_synchronize() cannot prove that it started before a given rcu_domain::lock(), it must not return to its caller until the matching rcu_domain::unlock() completes

RCU Semantics (Restrictions)



1. RCU provides ABA protection for update-friendly mechanisms
2. RCU provides bounded wait-free read-side primitives for real-time use

RCU Semantics (Restrictions)



1. RCU provides ABA protection for update-friendly mechanisms
2. RCU provides bounded wait-free read-side primitives for real-time use

And RCU is most frequently used for linked data structures.

RCU Semantics (Show Me The Code)

nothin
g

Example Application (Redux)

- Configuration information in variables `a` and `b`:
  - `int a, b; // Current configuration values`
  - Infrequently updated based on external inputs
  - Given reader access needs consistent values
- Reading “Oldish” values OK, ***if*** consistent
- Very frequent reader access to `a` and `b`

Design of RCU Use Case

- Put a & b into a structure to obtain consistency
  - "All problems in computer science can be solved by another level of indirection."

    David Wheeler
- Update: Allocate new structure & update pointer
- Free old structure "when it is safe to do so"

Core RCU API (Redux)

- `rcu_domain::lock`: Begin reader
- `rcu_domain::unlock`: End reader
- `rcu_synchronize()`: Wait for pre-existing readers
- `.retire()`: Invoke deleter after pre-existing readers complete
- `memory_order_consume`: Load RCU-protected pointer
- `memory_order_release`: Update RCU-protected pointer

For the full C++ RCU API as of September 2022: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2545r0.pdf

RCU Use Case: Reader

RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

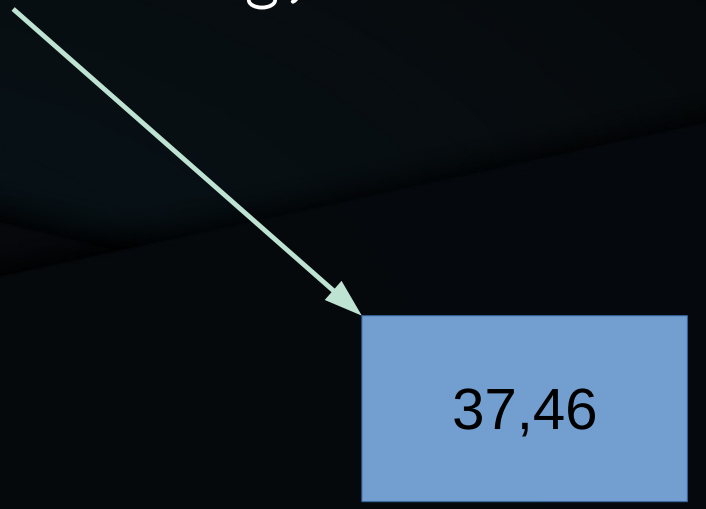
    std::scoped_lock rlock(std::rcu_default_domain());
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a;
    *cur_b = mcp->b;
}
```

RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a;
    *cur_b = mcp->b;
}
```



RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a;
    *cur_b = mcp->b;
}
```

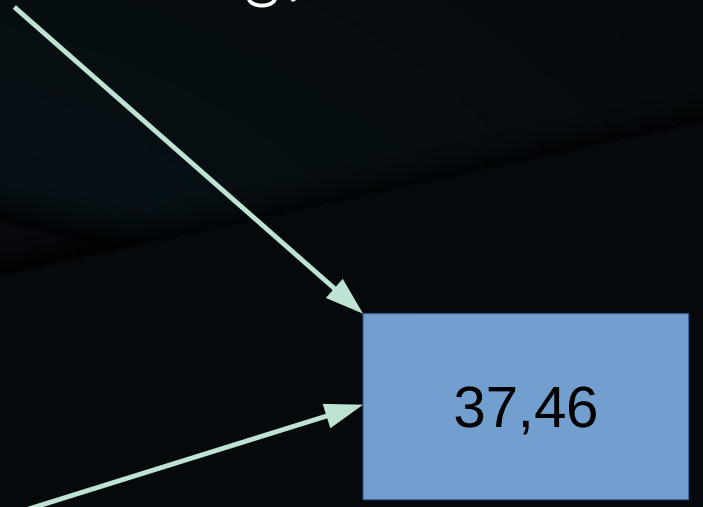


RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b;
}
```

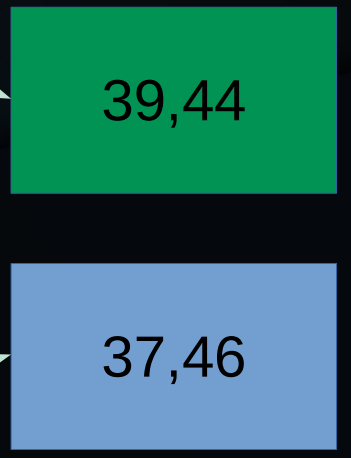


RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b;
}
```

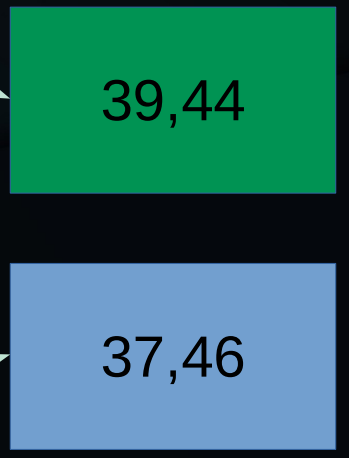


RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b; (46)
}
```

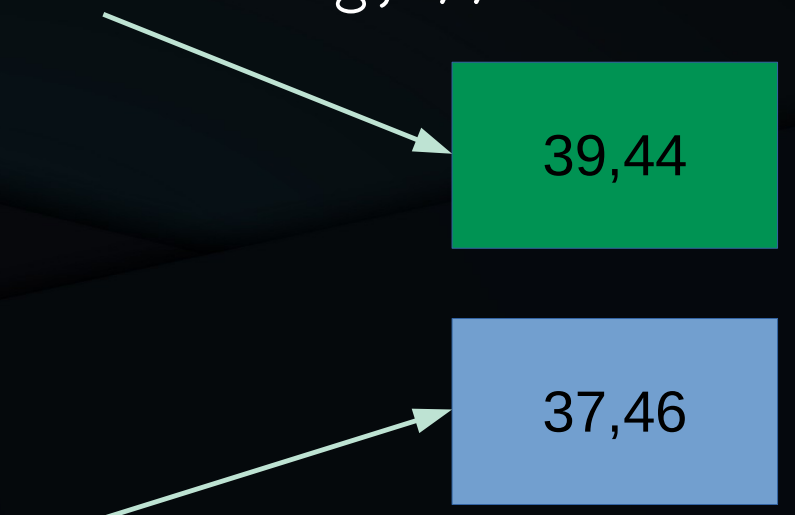


RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b; (46)
}
```

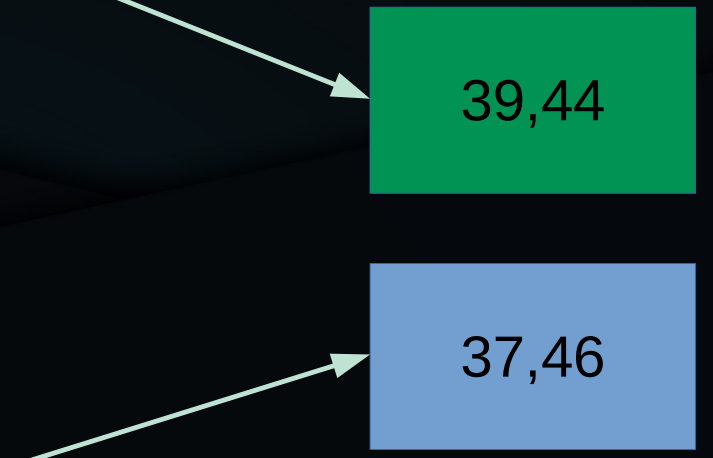


RCU Use Case: Reader

```
atomic<struct myconfig { int a, b; }*> curconfig; // Initialized

void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b; (46)
}
```



Despite change, got consistent values!

RCU Use Case: Writer

RCU Use Case: Atomic Writer

```
void set(int cur_a, int cur_b)
{
    struct myconfig *mcp = new ...;

    mcp->a = cur_a;
    mcp->b = cur_b;
    mcp = curconfig.exchange(mcp);
    rcu_synchronize();
    delete mcp;
}
```

RCU Use Case: Atomic Writer

```
void set(int cur_a, int cur_b)
{
    struct myconfig *mcp = new ...;

    mcp->a = cur_a;
    mcp->b = cur_b;
    mcp = curconfig.exchange(mcp);
    rcu_synchronize();
    delete mcp;
}
```



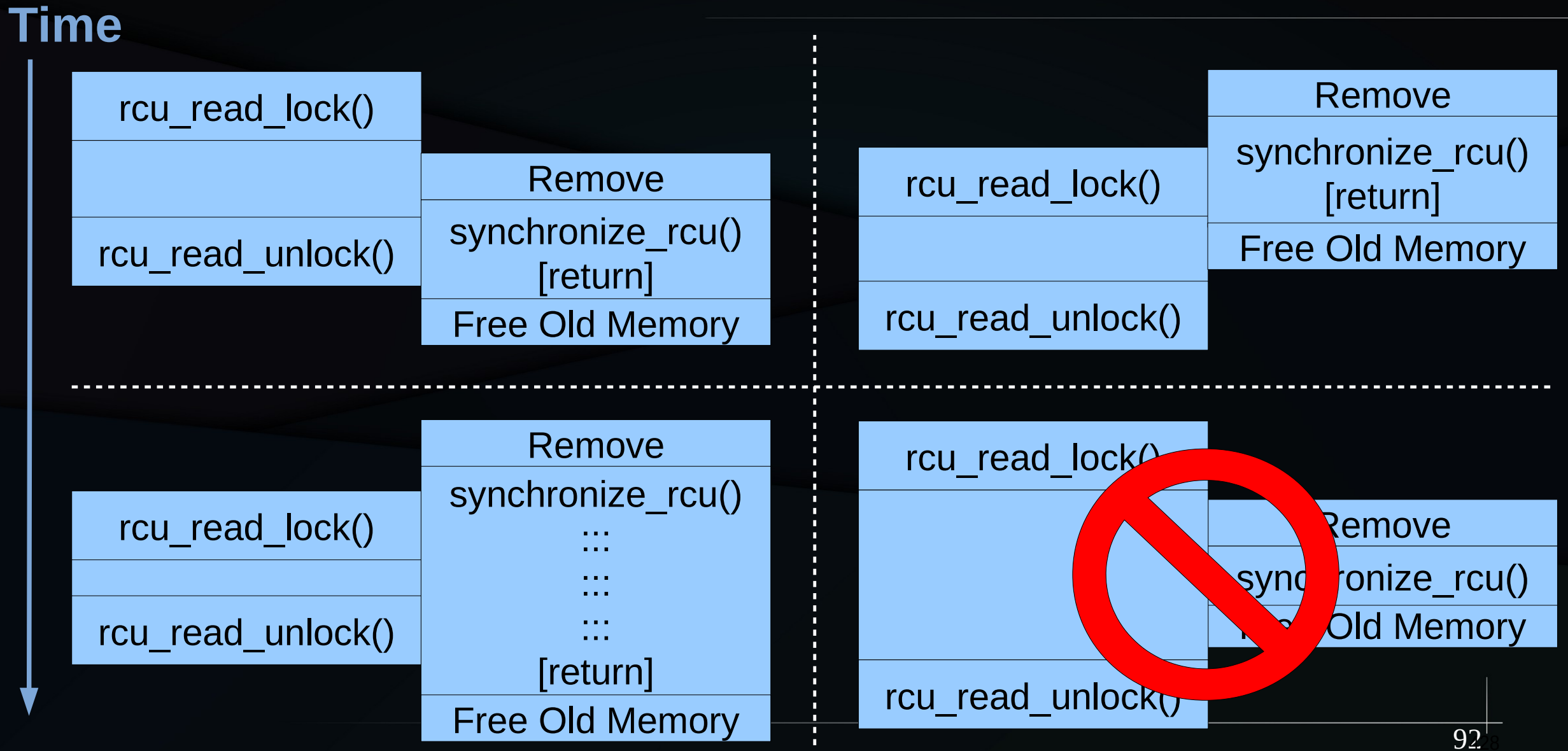
These will represent one writer

RCU Semantics

RCU Semantics (English, Redux)

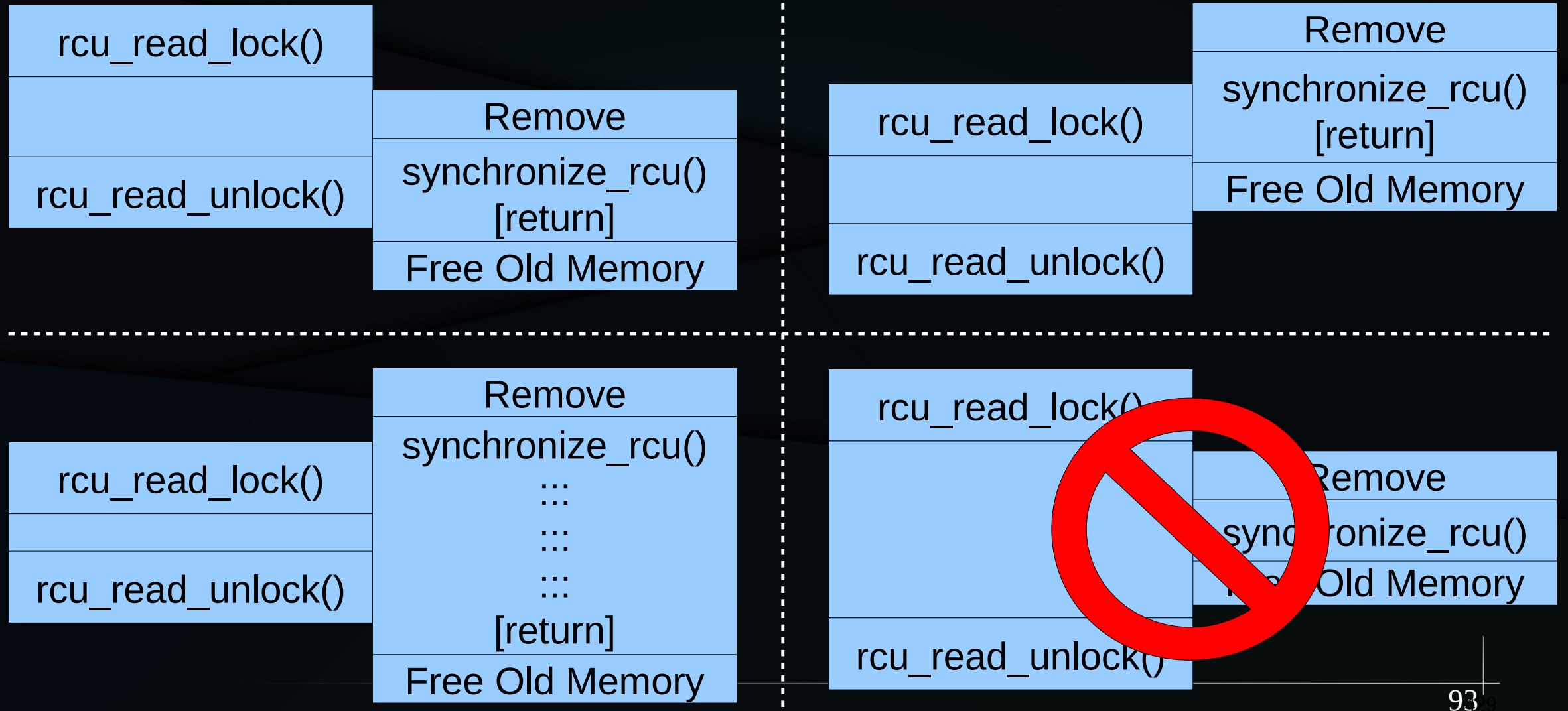
- Semantics weakened from reader-writer locking:
  1) Writers wait for read-holders, but only before freeing
  2) ~~Readers wait for the writer-holders~~
- Compensate for weak temporal semantics by adding *restrictions and spatial semantics* to RCU use cases
- Restated for RCU:
  - If synchronize_rcu() cannot prove that it started before a given rcu_read_lock(), it must not return to its caller until the matching rcu_read_unlock() completes

RCU Semantics (Graphical)



RCU Semantics (Graphical)

Time (really ordering)



RCU Semantics (Exercise The Code)




```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)




```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)

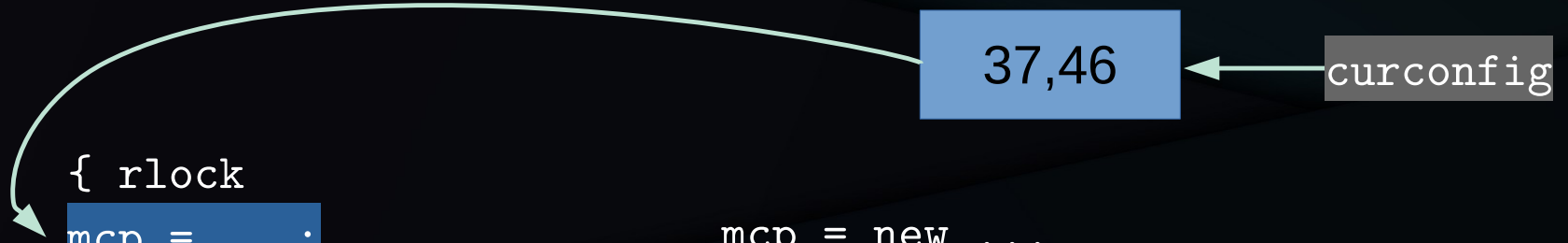



```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)

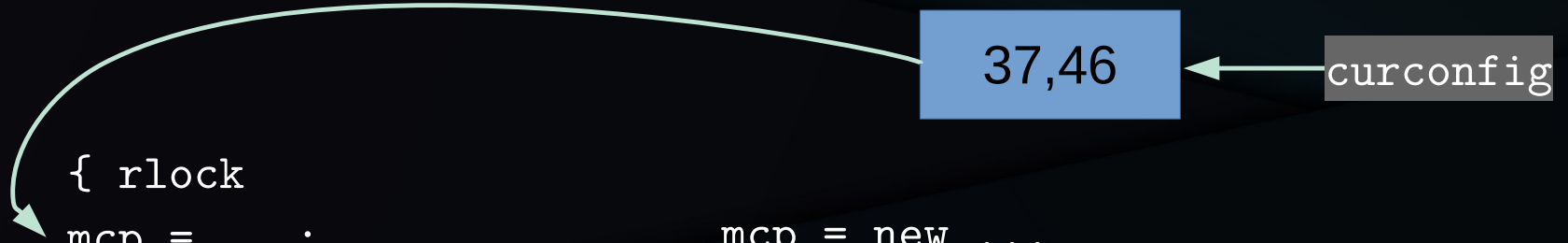



```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)

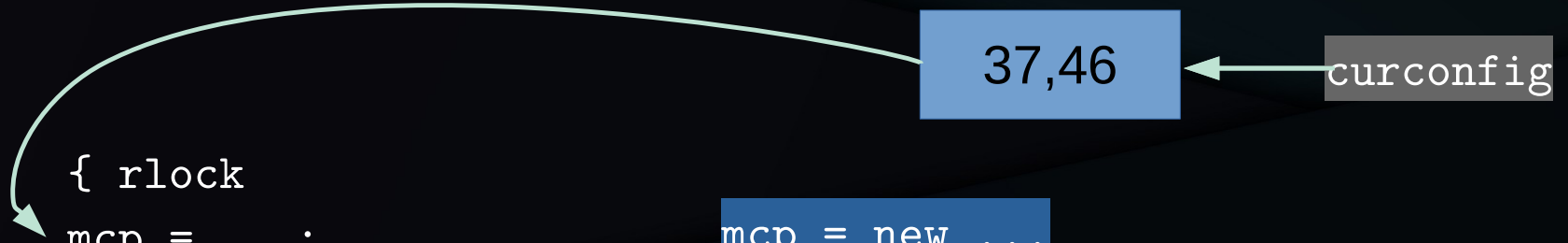



```
{ rlock
mcp = ...;
*cur_a = mcp->a; (37)
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)

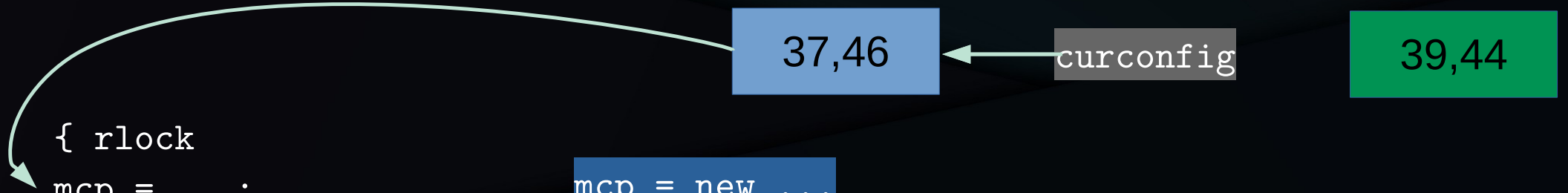



```
{ rlock
mcp = ...;
*cur_a = mcp->a; (37)
*cur_b = mcp->b;
}
```

```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

RCU Semantics (Exercise The Code)

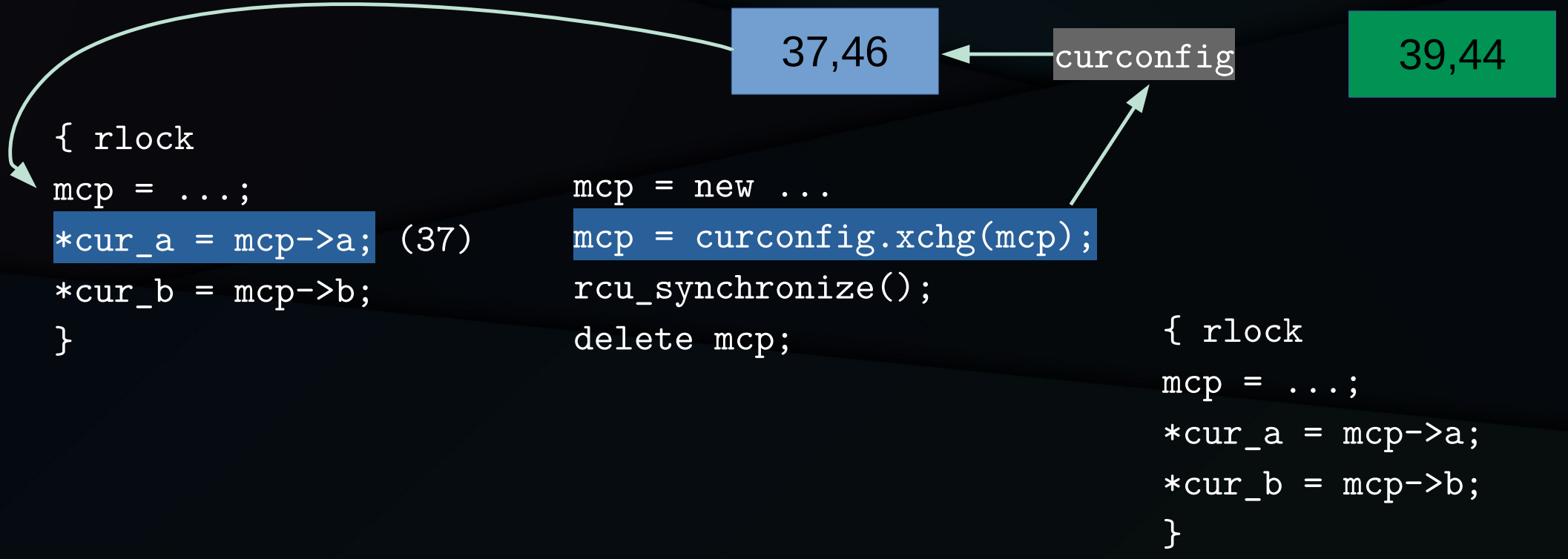



```
{ rlock
mcp = ...;
*cur_a = mcp->a; (37)
*cur_b = mcp->b;
}
```

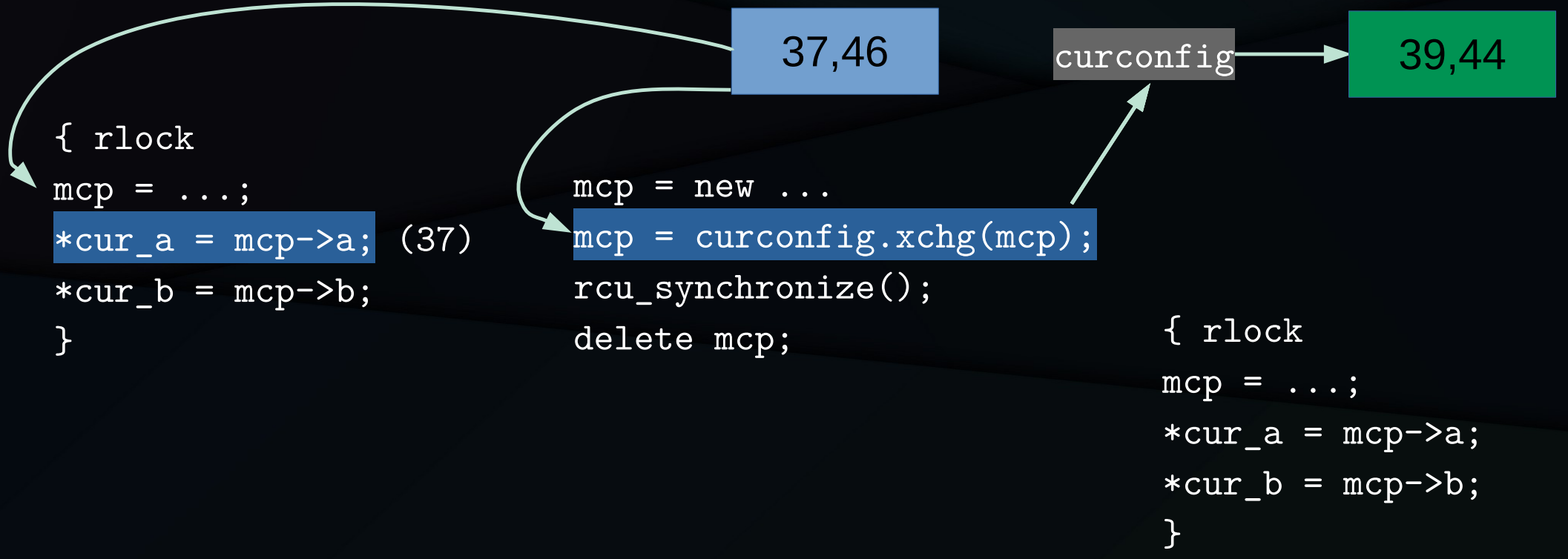
```
mcp = new ...
mcp = curconfig.xchg(mcp);
rcu_synchronize();
delete mcp;
```

```
{ rlock
mcp = ...;
*cur_a = mcp->a;
*cur_b = mcp->b;
}
```

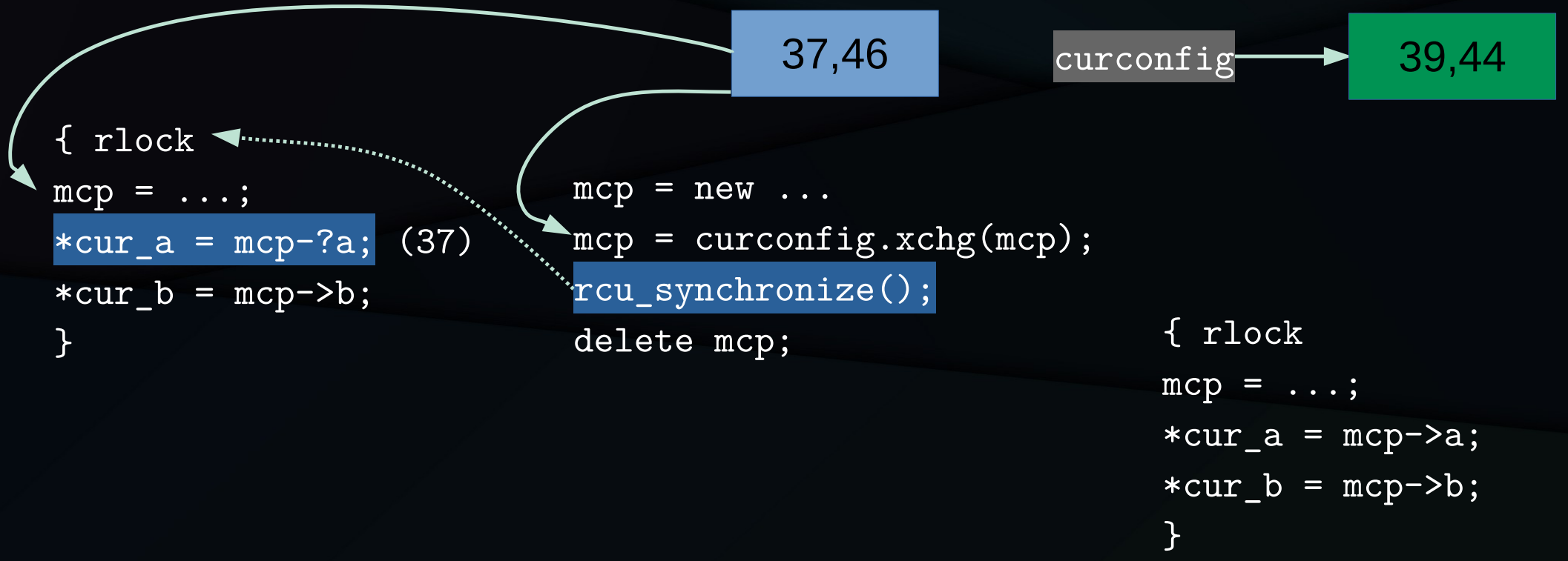
RCU Semantics (Exercise The Code)



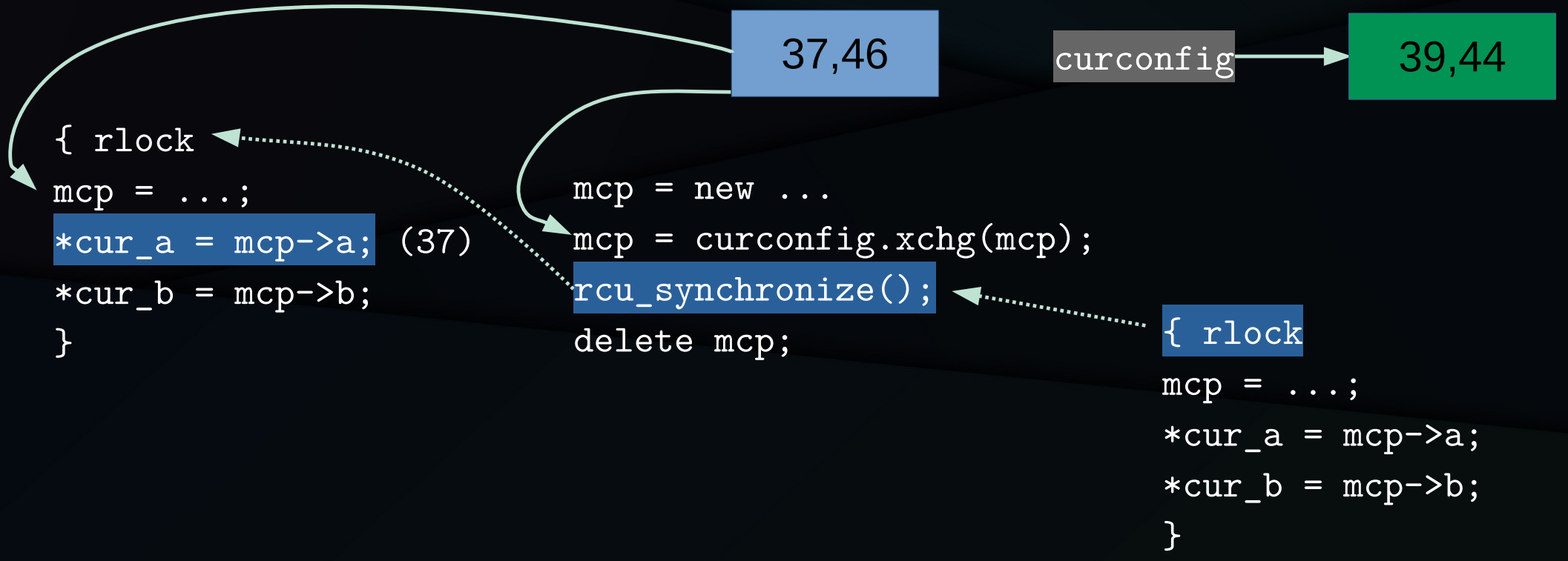
RCU Semantics (Exercise The Code)



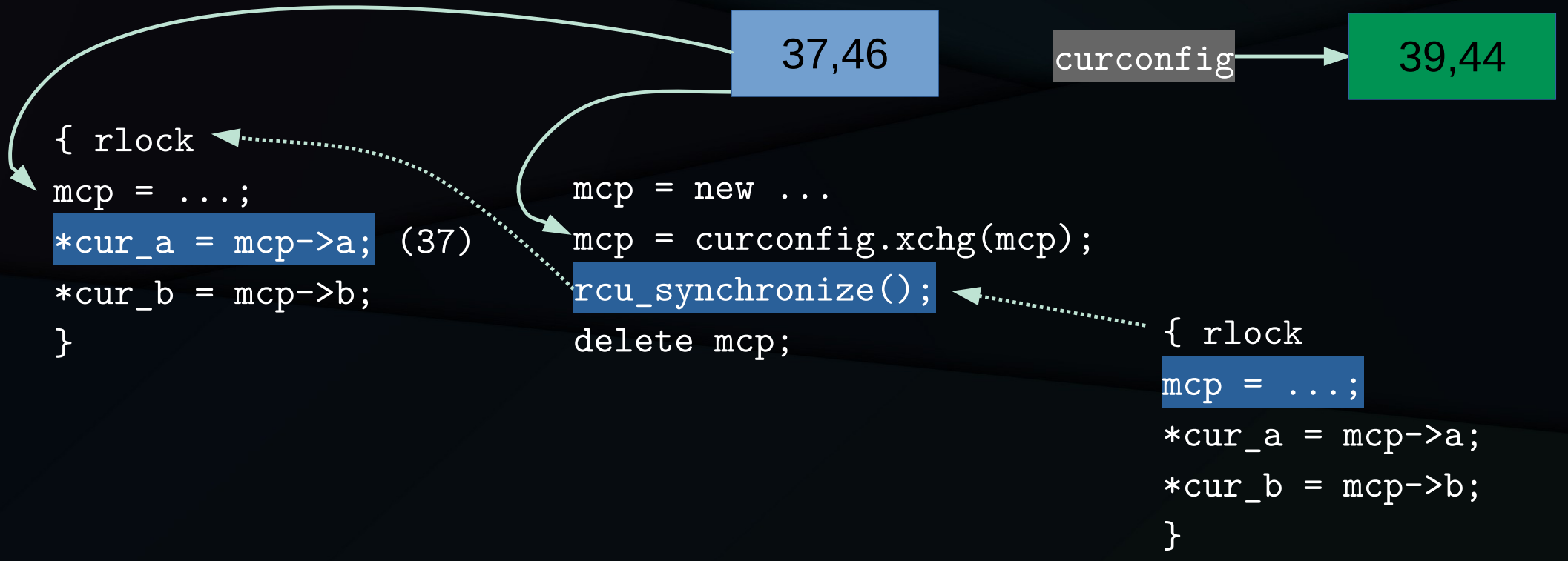
RCU Semantics (Exercise The Code)



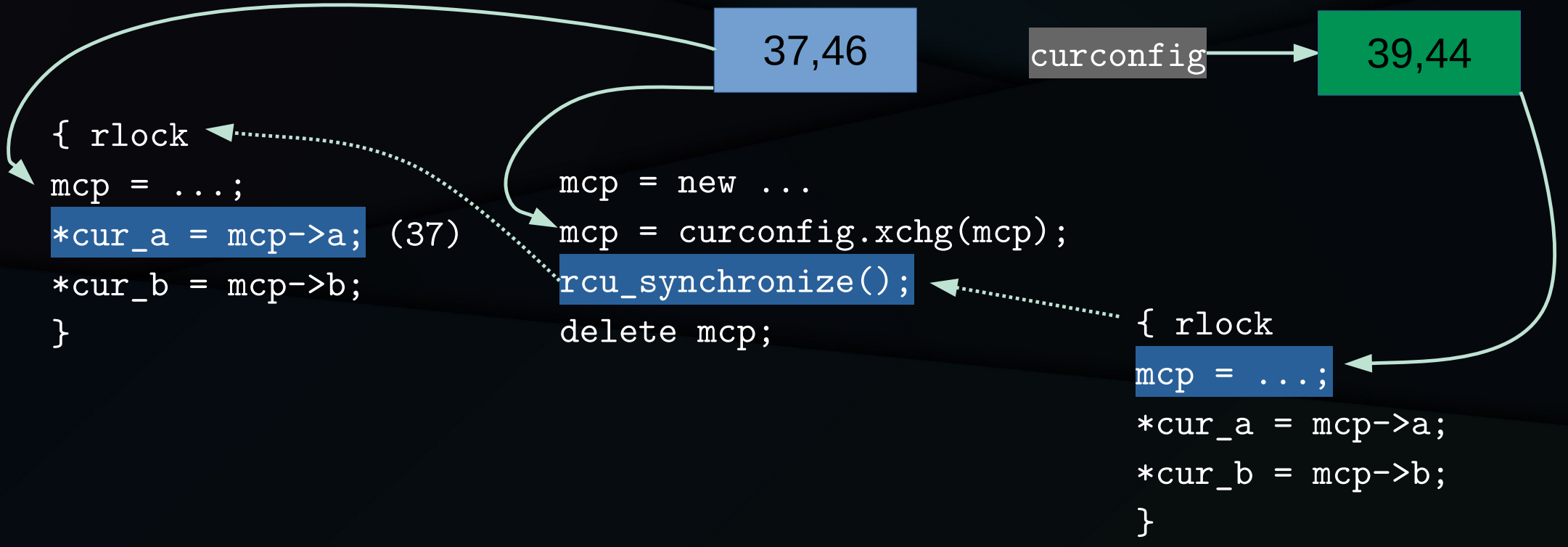
RCU Semantics (Exercise The Code)



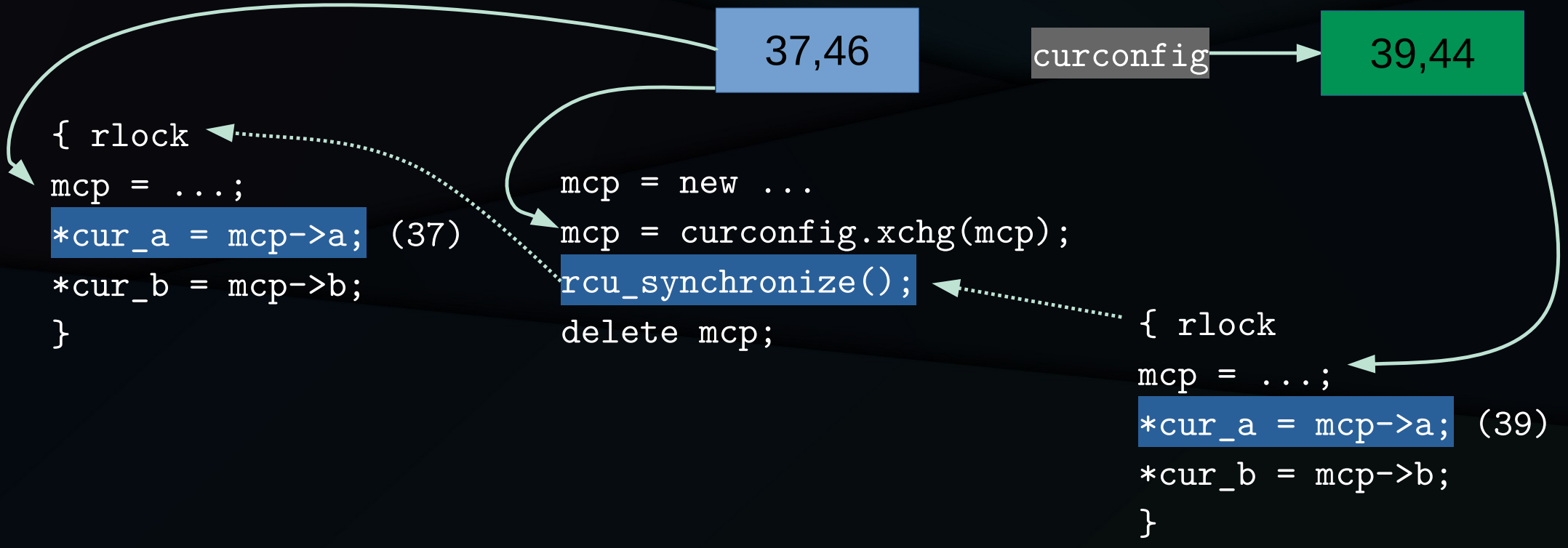
RCU Semantics (Exercise The Code)



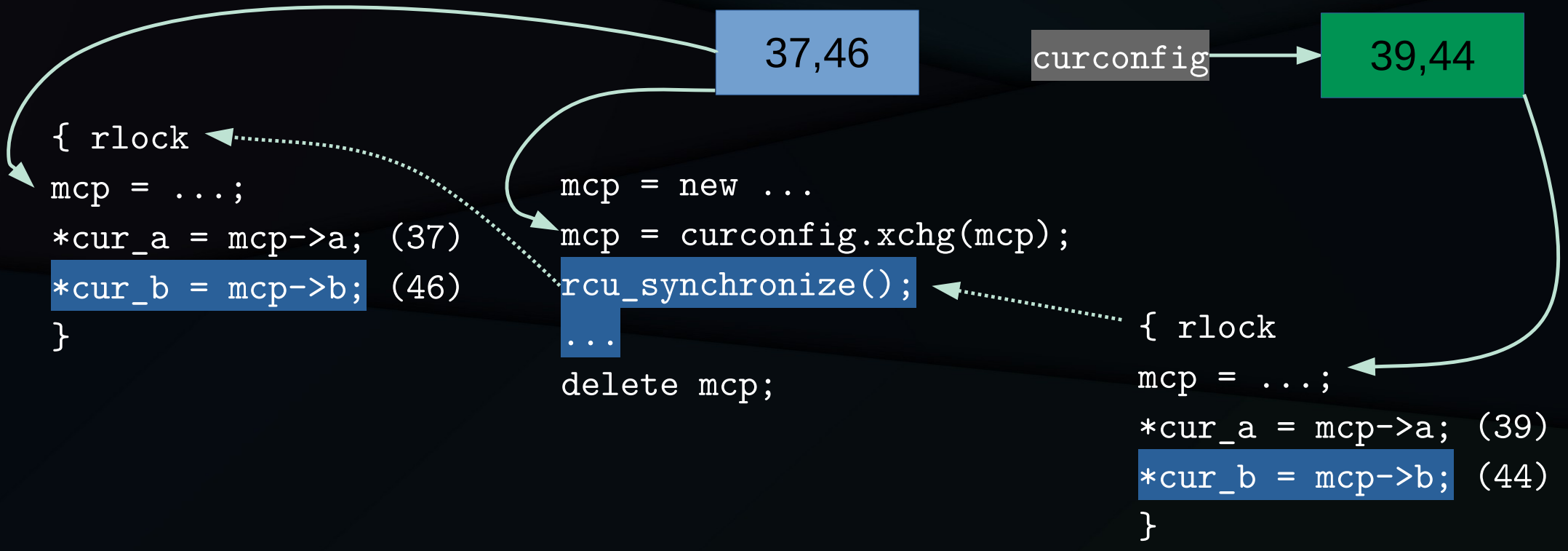
RCU Semantics (Exercise The Code)



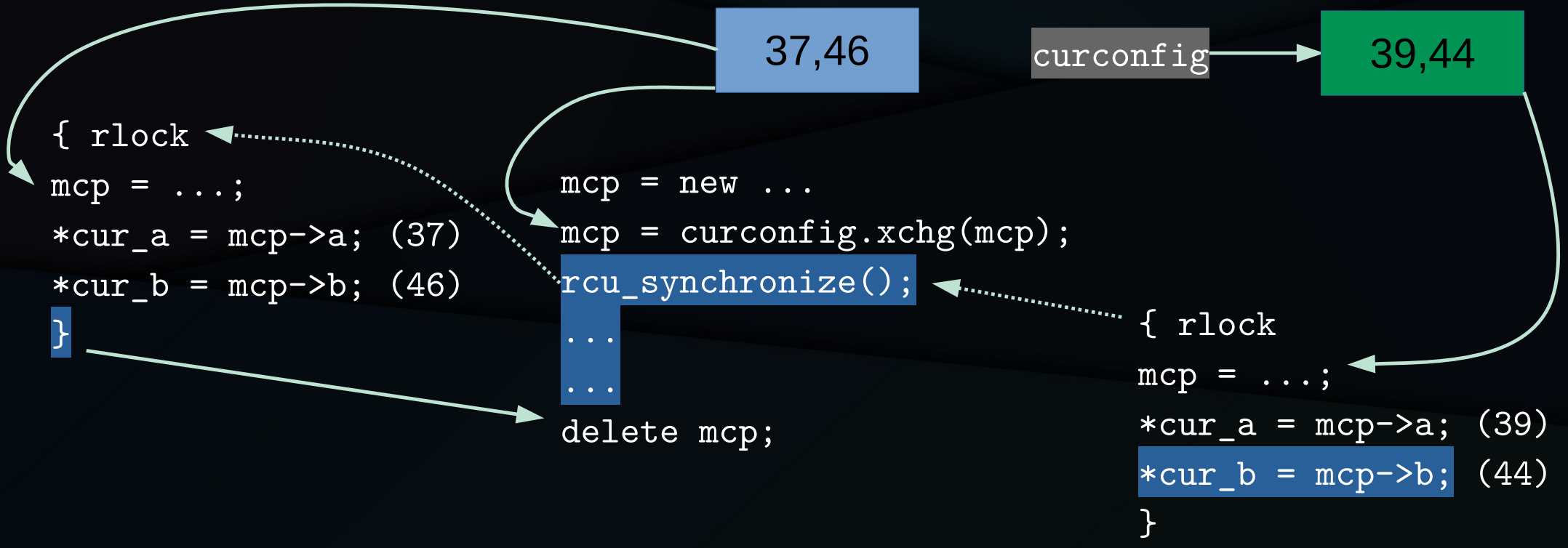
RCU Semantics (Exercise The Code)



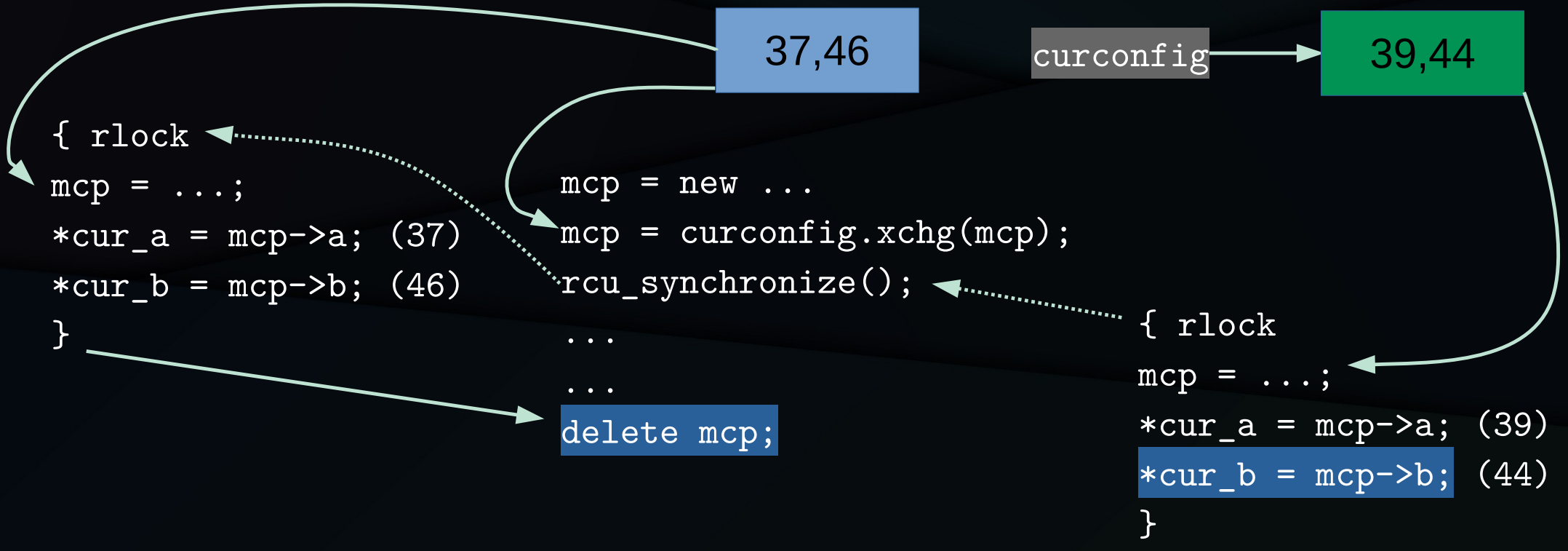
RCU Semantics (Exercise The Code)



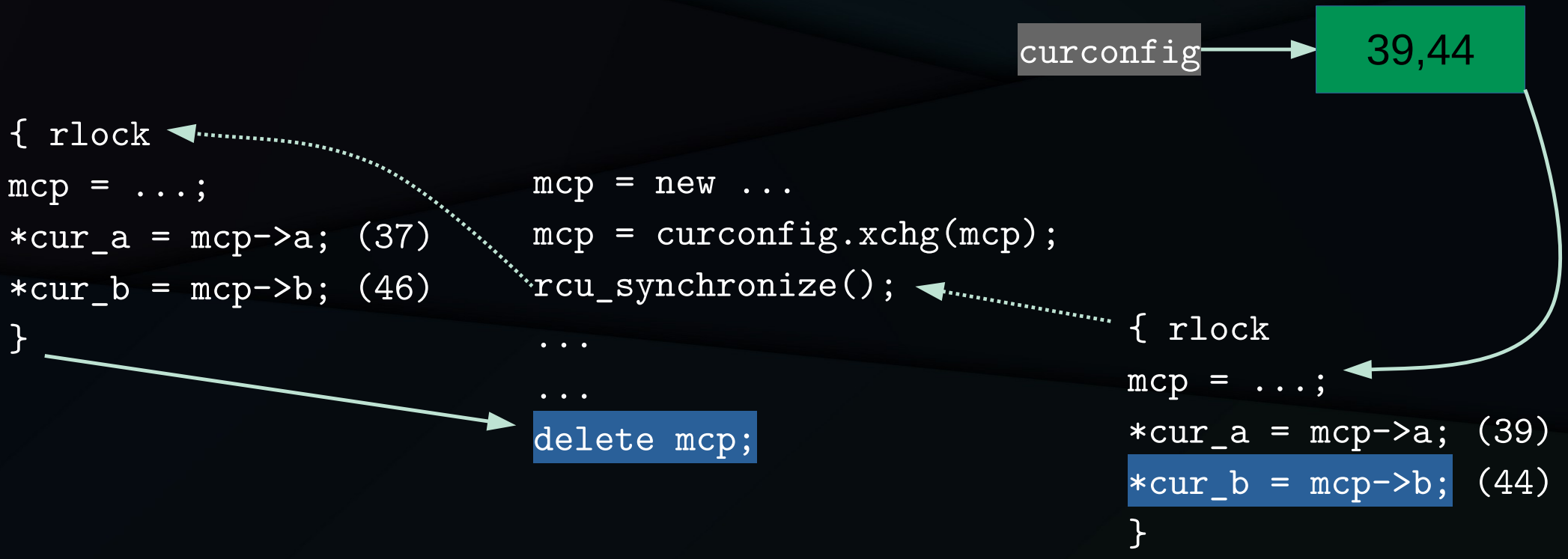
RCU Semantics (Exercise The Code)



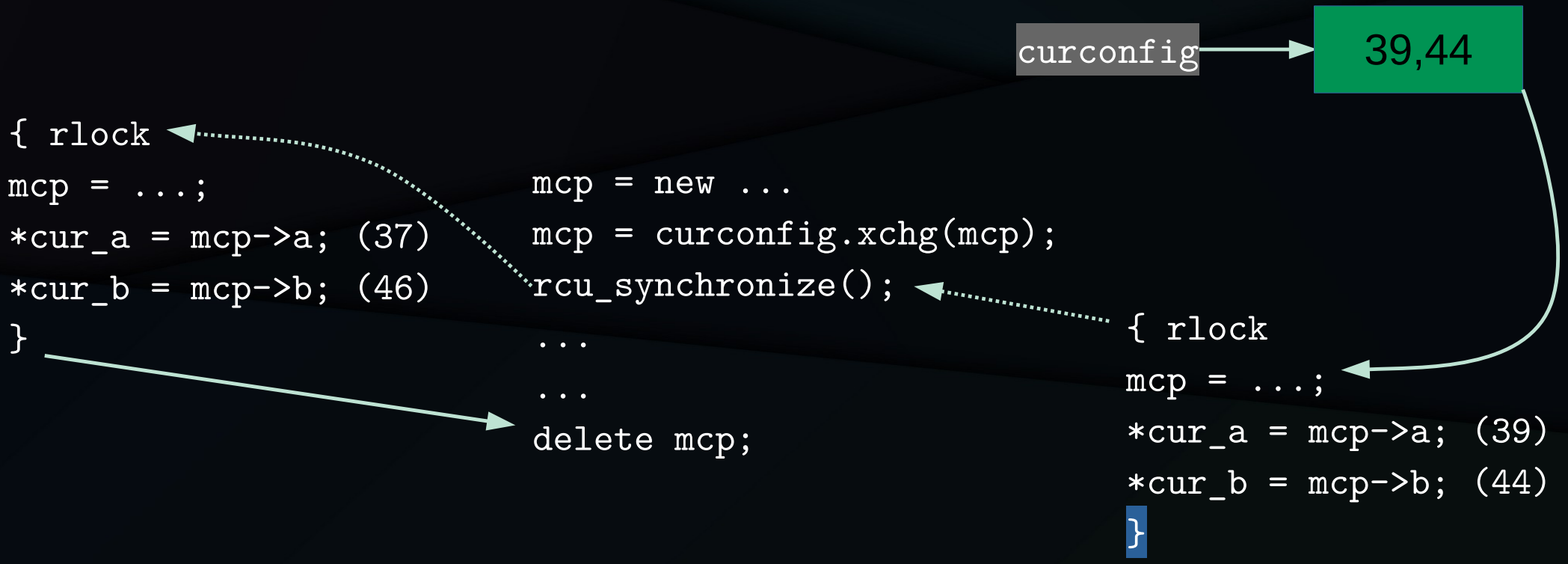
RCU Semantics (Exercise The Code)



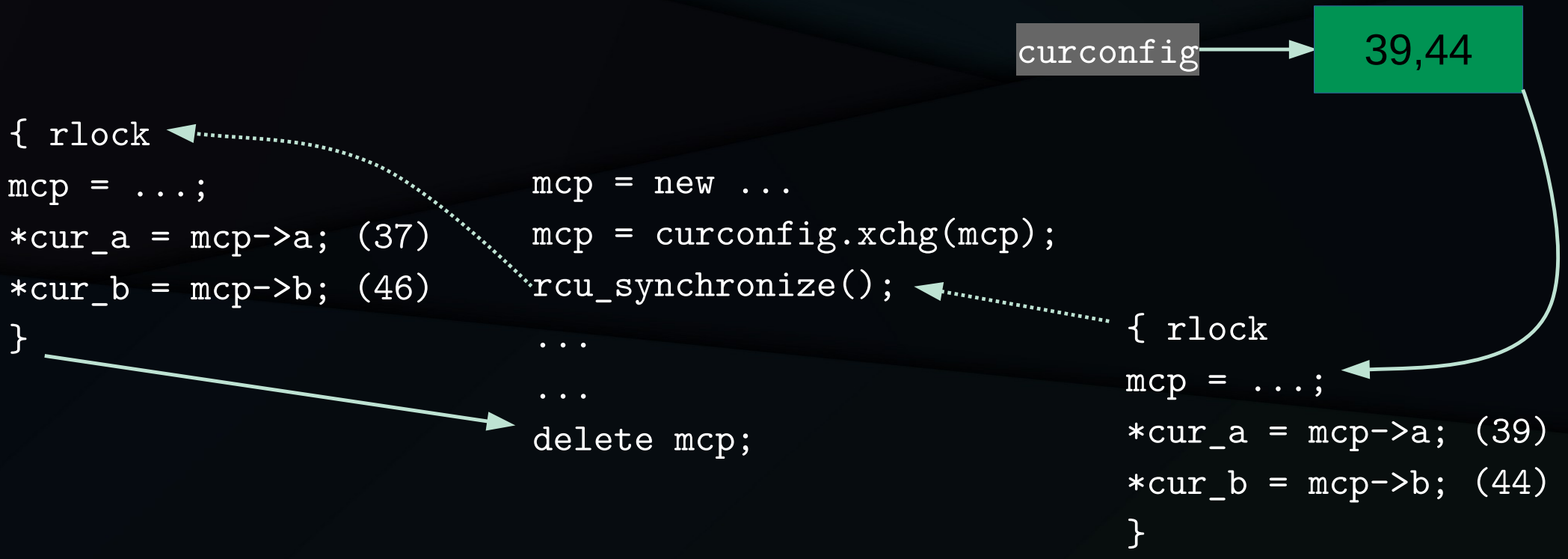
RCU Semantics (Exercise The Code)



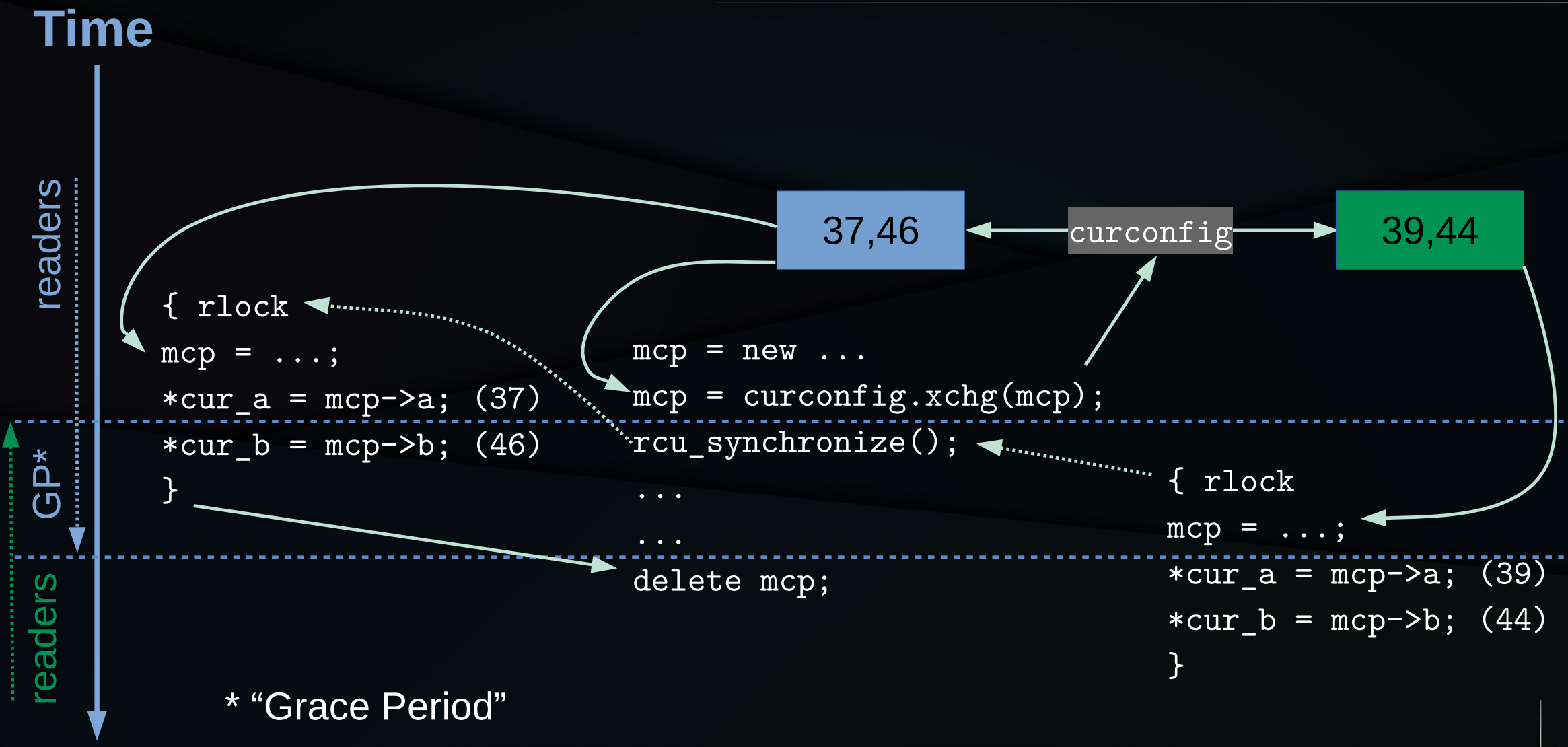
RCU Semantics (Exercise The Code)



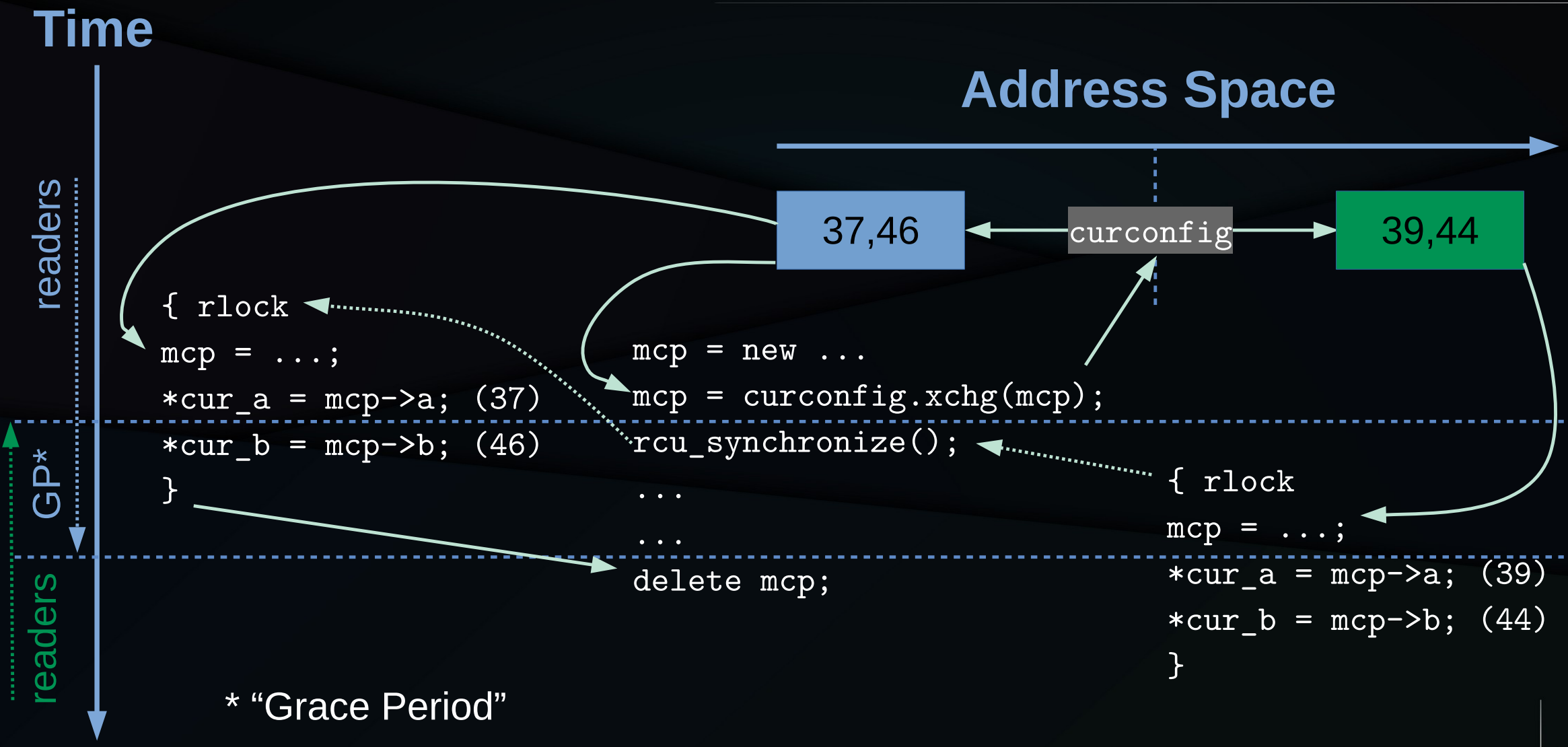
RCU Semantics (Exercise The Code)



RCU Semantics: (Temporal)



RCU Semantics: (Temporal & Spatial)



First space/time articulation for RCU (to the best of my knowledge): Jonathan Walpole and his students Josh Triplett and Phil Howard

Space/Time Synchronization

Core RCU API: Temporal vs. Spatial

- `rcu_domain::lock`: Begin reader
- `rcu_domain::unlock`: End reader
- `rcu_synchronize()`: Wait for pre-existing readers
- `.retire()`: Invoke deleter after pre-existing readers complete
- `memory_order_consume`: Load RCU-protected pointer
- `memory_order_release`: Update RCU-protected pointer

For the full C++ RCU API as of September 2022: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2545r0.pdf

Space/Time Synchronization Outline

- Readers:
  - Time via rcu_domain::lock and rcu_domain::unlock
  - Space via memory_order_consume loads
- Updates are split into reader-visible and not
  - Add: Initialize, then use memory_order_release store
  - Delete: Remove, wait for grace period, then free
- Existence guarantees (as opposed to ownership)

Reader Space/Time Synchronization

```
void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b; (46)
}
```

Temporal synchronization 1

Spatial synchronization

Existence guarantee

Temporal synchronization 2

Reader Space/Time Synchronization

```
void get(int *cur_a, int *cur_b)
{
    struct myconfig *mcp;

    std::scoped_lock rlock(...);
    mcp = curconfig.load(memory_order_consume);
    *cur_a = mcp->a; (37)
    *cur_b = mcp->b; (46)
}
```

Temporal synchronization 1
(Start read-side critical section)

Spatial synchronization
(Get current version)

Existence guarantee

Temporal synchronization 2
(End read-side critical section)

Updater Space/Time Synchronization

```
void set(int cur_a, int cur_b)
{
    struct myconfig *mcp = new ...;

    mcp->a = cur_a;
    mcp->b = cur_b;
    mcp = curconfig.exchange(mcp);
    rcu_synchronize();
    delete mcp;
}
```

Spatial synchronization

Temporal synchronization

Updater Space/Time Synchronization

```
void set(int cur_a, int cur_b)
{
    struct myconfig *mcp = new ...;

    mcp->a = cur_a;
    mcp->b = cur_b;
    mcp = curconfig.exchange(mcp);
    rcu_synchronize();
    delete mcp;
}
```

Spatial synchronization
(Make old data inaccessible and new data accessible to future readers)

Temporal synchronization
(Wait for pre-existing readers)

RCU: Exploiting Both Temporal and Spatial Synchronization for Decades!



Paolo Neo, Public Domain

Who Does Spatial Synchronization?!?

Who Does Spatial Synchronization?!?

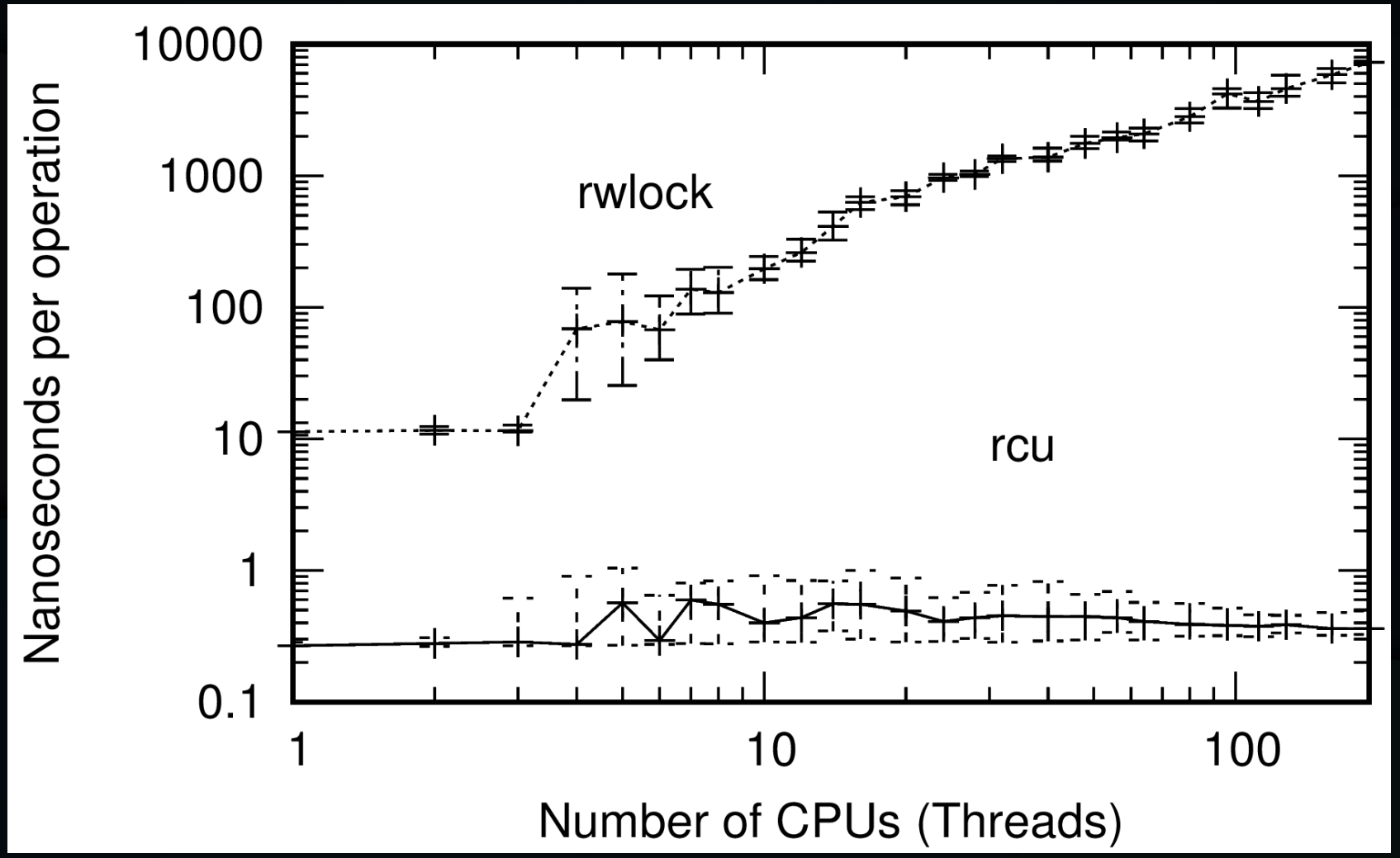
- Per-task stack locations
- Per-CPU/-thread variables
- Hash tables with per-bucket locks
  - And sharding in general (the “data locking” of old)
- Hazard pointers & other deferred reclamation

Who Does Spatial Synchronization?!?

- Per-task stack locations
- Per-CPU/-thread variables
- Hash tables with per-bucket locks
  - And sharding in general (the “data locking” of old)
- Hazard pointers & other deferred reclamation
- **In short, pretty much everybody!!!**

Performance: rwlock vs RCU

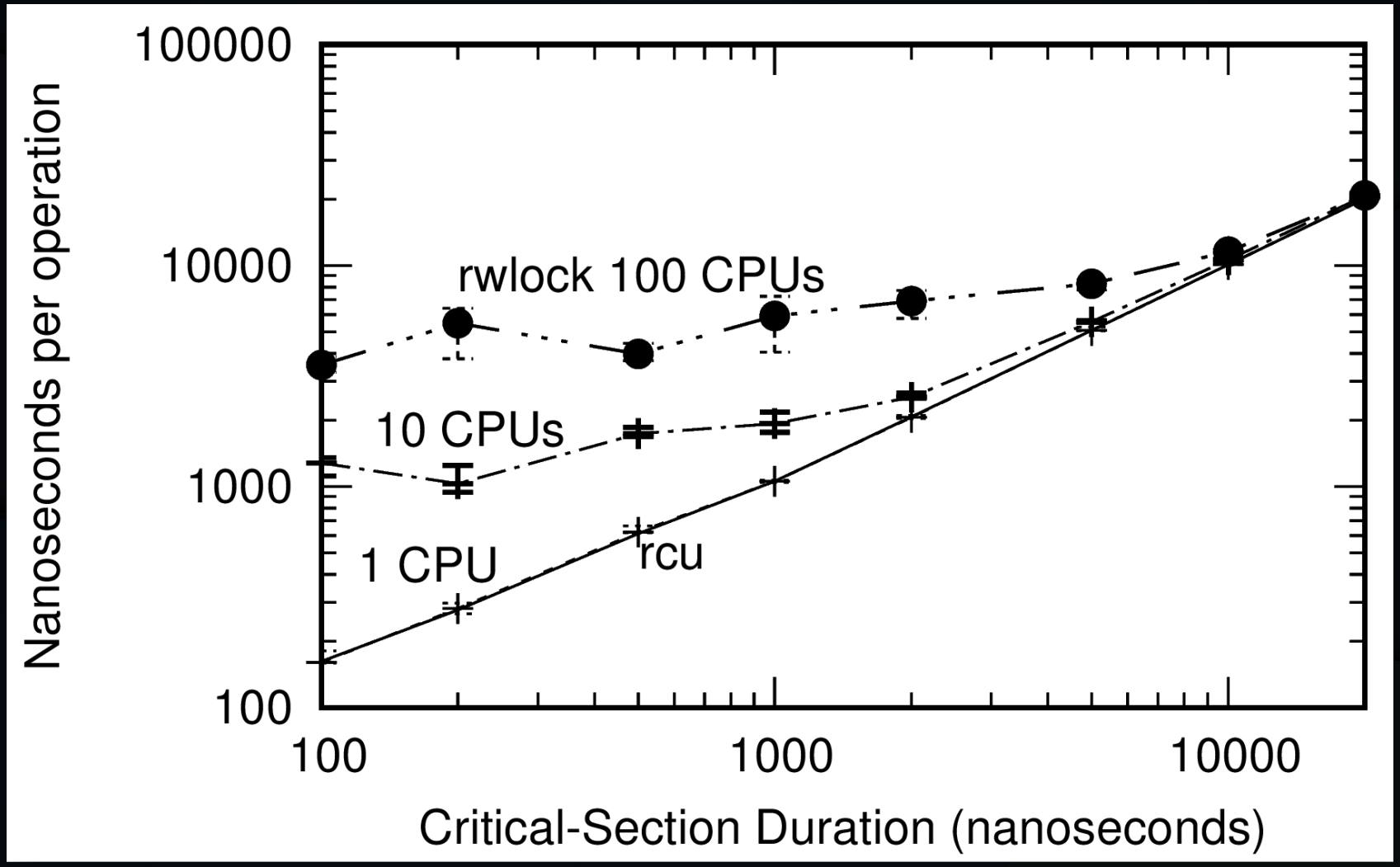
Scalability for Empty Critical Sections



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/defer/data/rcuscale.hps.2020.05.28a/rwlockperf.eps

… And Non-Empty Critical Sections



Intel(R) Xeon(R) Platinum 8176 CPU @ 2.10GHz

perfbook CodeSamples/defer/data/rcuscale.hps.2020.05.28a/rwlockperf.eps

RCU Insert, Delete, and Traversal

RCU Insert, Delete, and Search

- Atomic replacement is unusual
- More typical: Separate insertion and deletion
  - `list_add_rcu()`
  - `list_del_rcu()`
  - `list_for_each_entry_rcu()`
- And asynchronous post-grace-period kfree():
  - `kfree_rcu()`

RCU Insert, Delete, and Search

```
struct mystruct {
    int a;
    int b;
    struct list_head list;
    struct rcu_head rh;
};
```

RCU Insert, Delete, and Search

```
void mystruct_insert(int a, int b)  // Allow duplicates
{
    struct mystruct *msp = kmalloc(...);

    initialize_mystruct(msp, a, b);
    spin_lock(&mylock);
    list_add_rcu(&msp->list, &mylist); // Provides ordering
    spin_unlock(&mylock);
}
```

RCU Insert, Delete, and Search

```
void mystruct_delete(int a)
{
    struct mystruct *msp = NULL;

    spin_lock(&mylock);
    msp = mystruct_search(a);
    if (msp)
        list_del_rcu(&msp->list);
    spin_unlock(&mylock);
    if (msp)
        kfree_rcu(msp, rh); // Free after grace period
}
```

RCU Insert, Delete, and Search

```
// Caller must rcu_read_lock() or hold mylock.
struct mystruct *mystruct_search(int a)
{
    struct mystruct *msp;

    list_for_each_entry_rcu(msp, &mylist, list)
        if (msp->a == a)
            return msp;
    return NULL;
}
```

Would anything need to change to allow `mystruct_search()` to do its own synchronization?

RCU Insert, Delete, and Search

```
int mystruct_lookup(int a)
{
    struct mystruct *msp;
    int ret = -1;

    rcu_read_lock();
    msp = mystruct_search(a);
    if (msp)
        ret = msp->b;
    rcu_read_unlock();
    return ret;
}
```

RCU to Quasi Reader-Writer Lock

RCU to Quasi Reader-Writer Lock

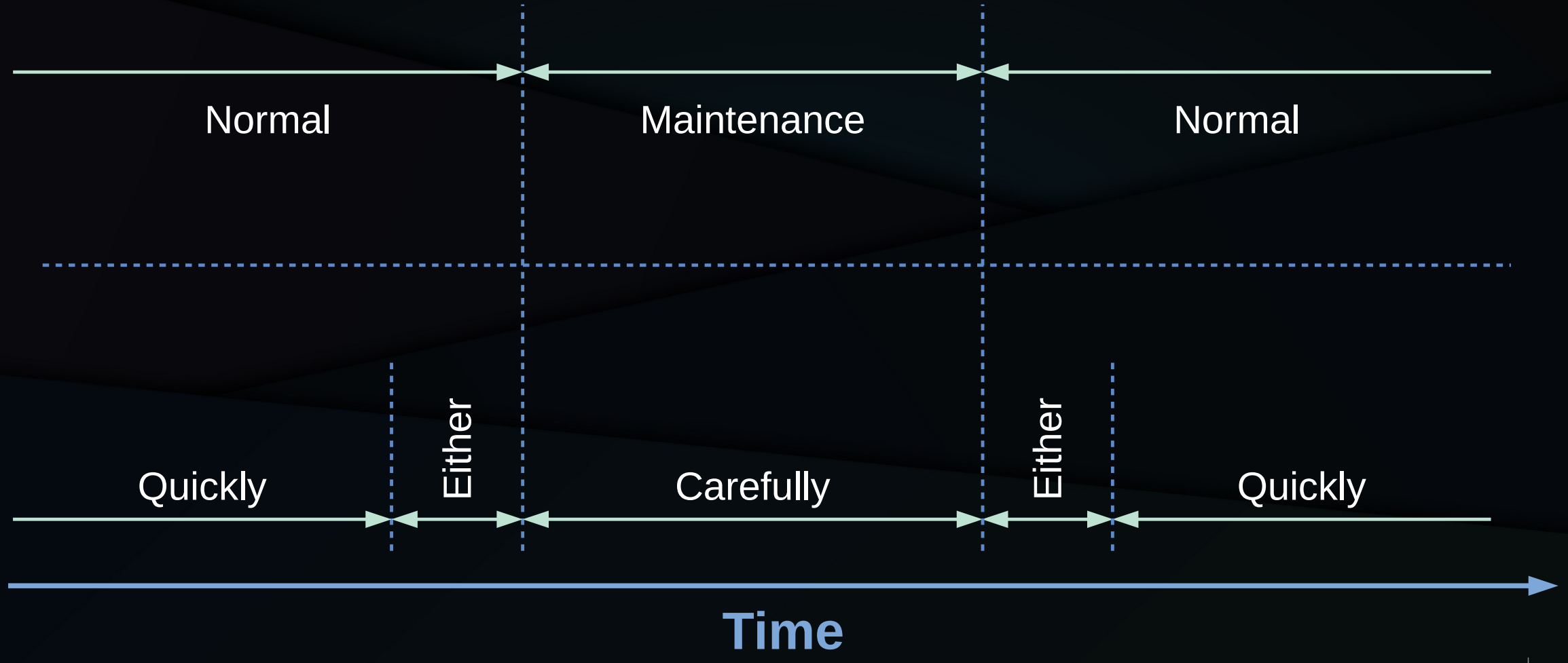
- Add to publish/subscribe and wait-to-finish:
  - Heap-allocated linked structure
  - Deferred reclamation
  - RCU readers as read-held reader-writer lock
  - Both spatial and temporal synchronization

Example: Phased State Change

Example: Phased State Change

- Multithreaded application
  - Common-case operation must be fast
  - But care is required during maintenance
- Use flag to indicate that care is required
  - But how to reliably synchronize?
  - OK to be careful just before/after maintenance

Phased State Change (Graphical)



Common-Case Operation

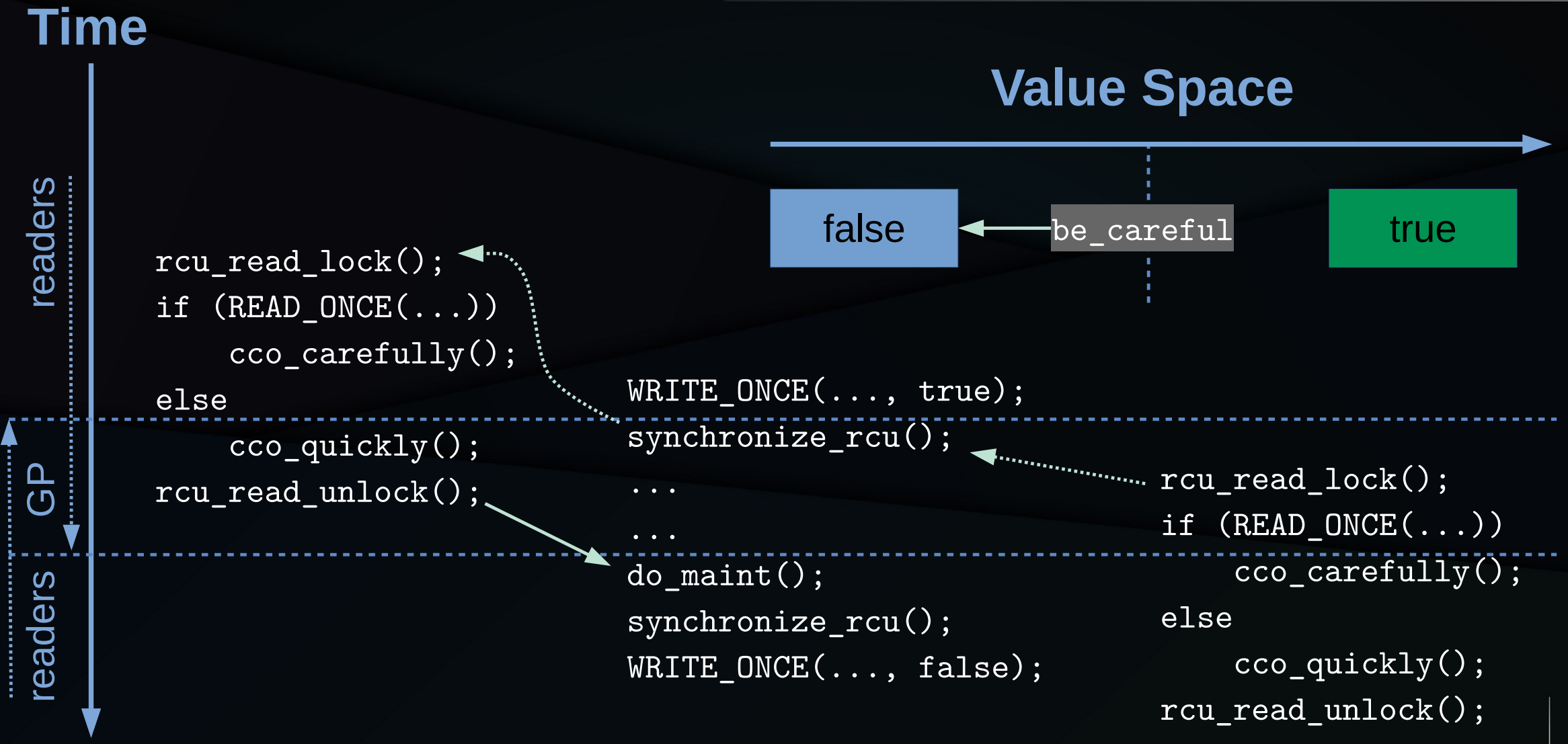
```
bool be_careful;

void cco(void)
{
    rcu_read_lock();
    if (READ_ONCE(be_careful))
        cco_carefully();
    else
        cco_quickly();
    rcu_read_unlock();
}
```

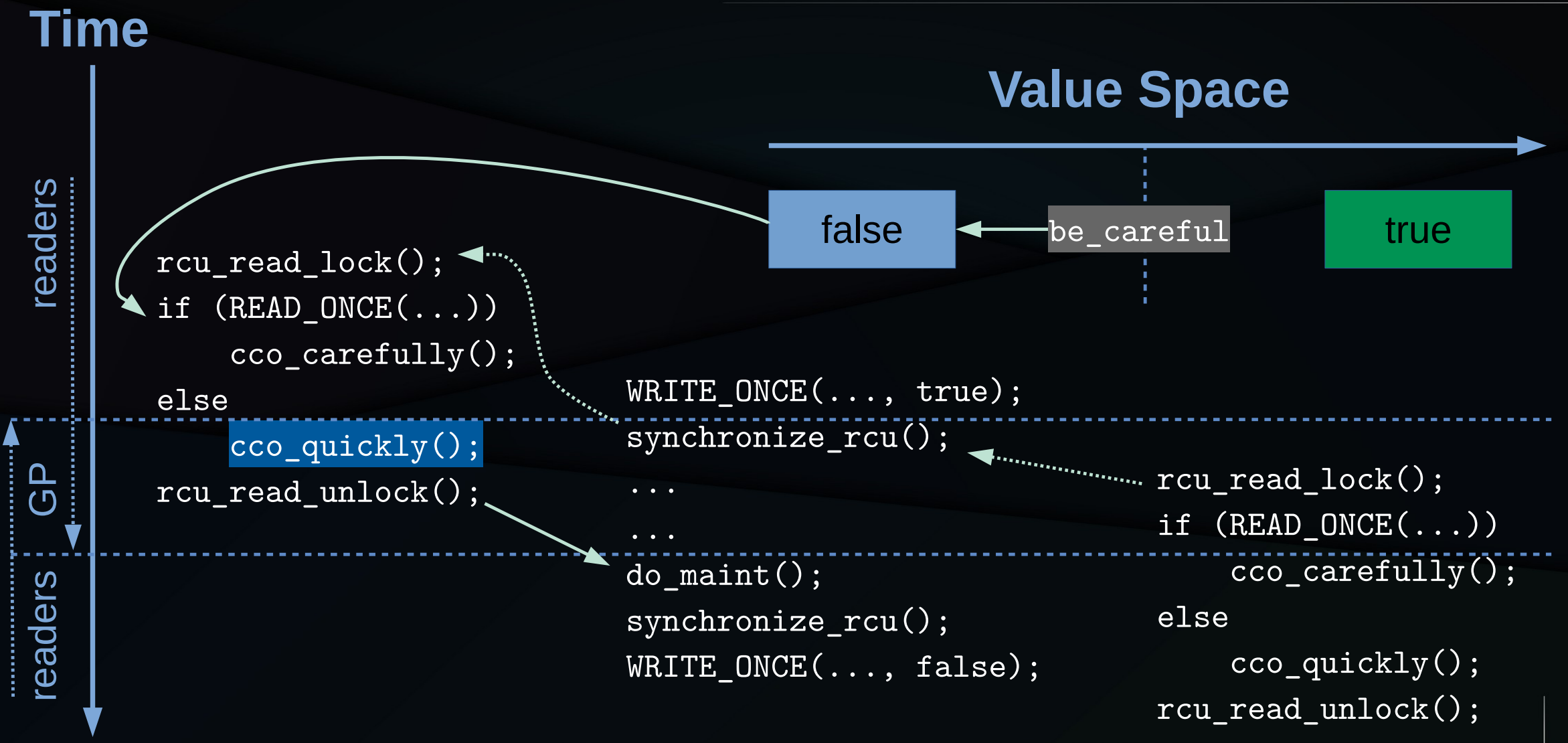
Maintenance Operation

```
void maint(void)
{
    WRITE_ONCE(be_careful, true);
    synchronize_rcu();
    do_maint();
    synchronize_rcu(); // Why is this needed?
    WRITE_ONCE(be_careful, false);
}
```

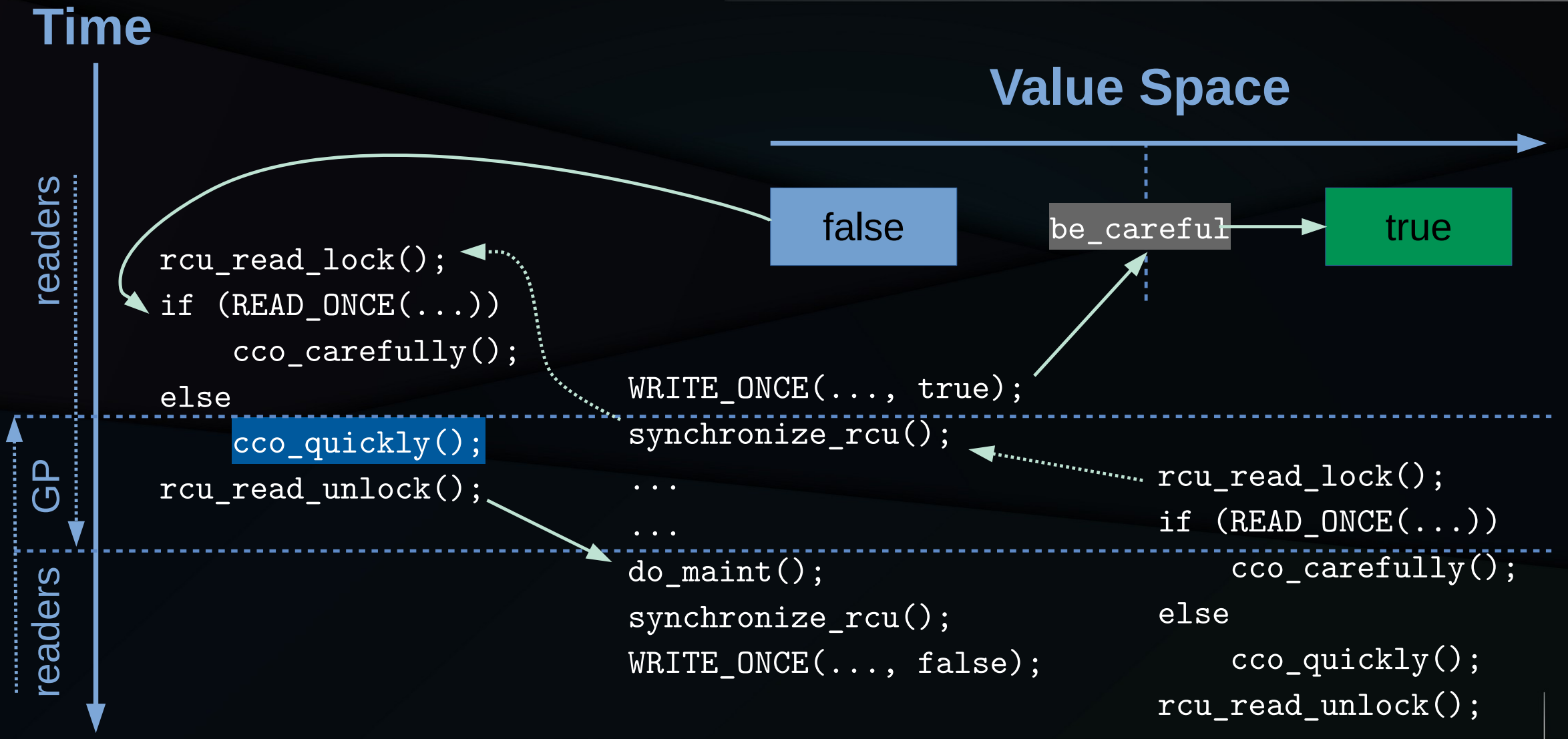
Phased State Change: Time & Space



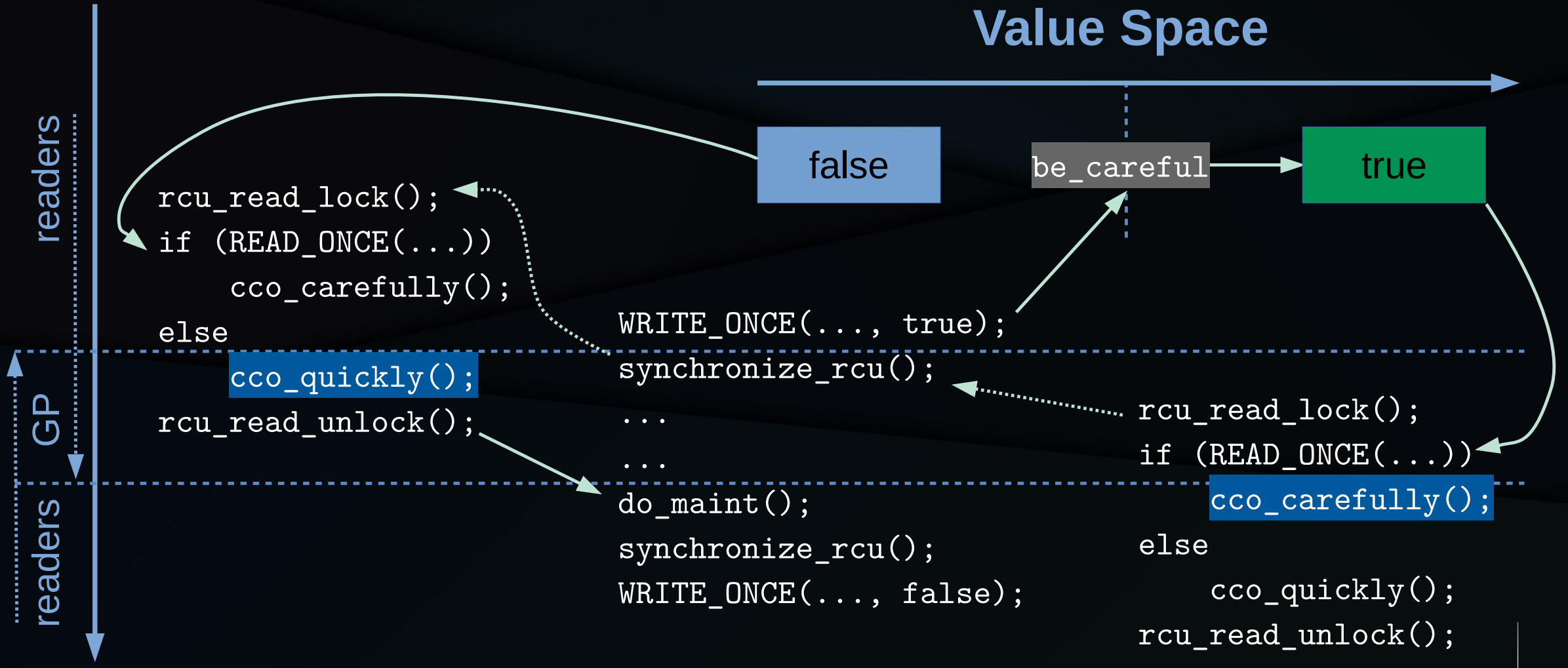
Phased State Change: Time & Space



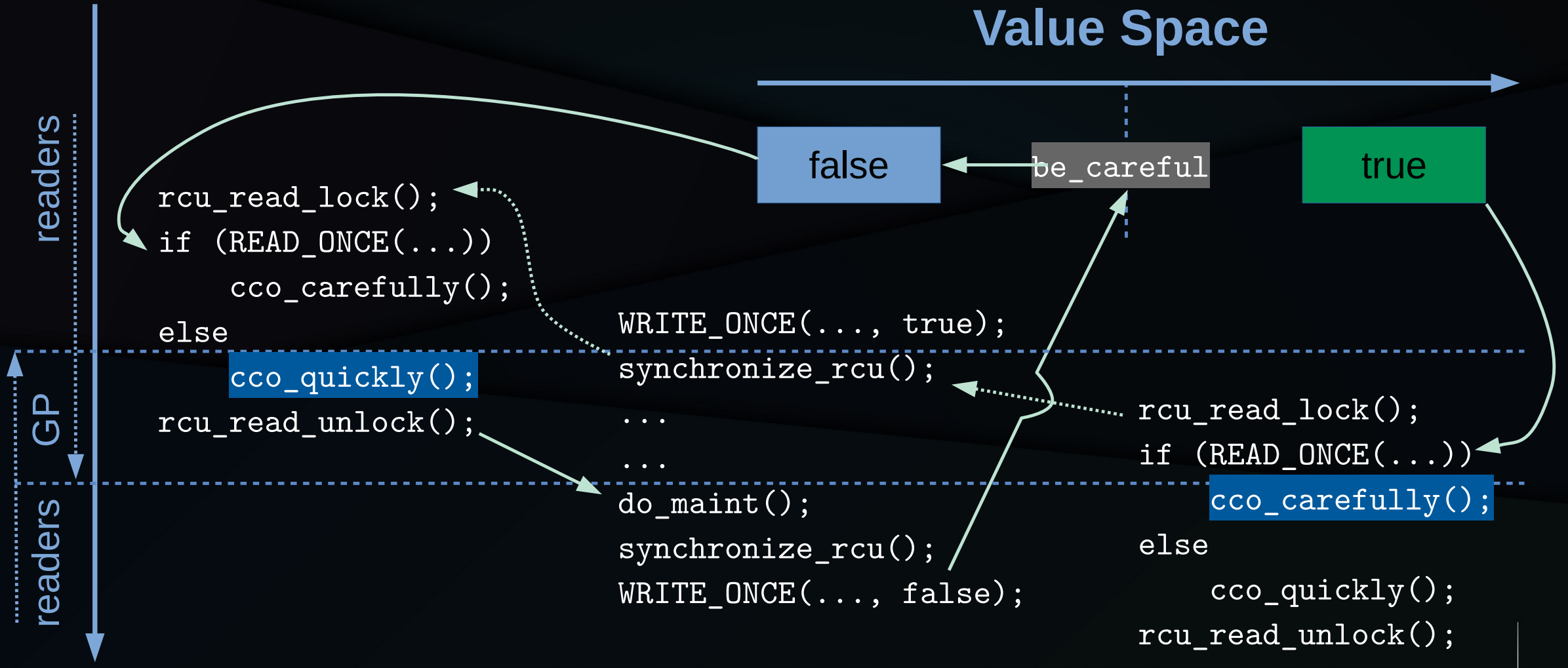
Phased State Change: Time & Space



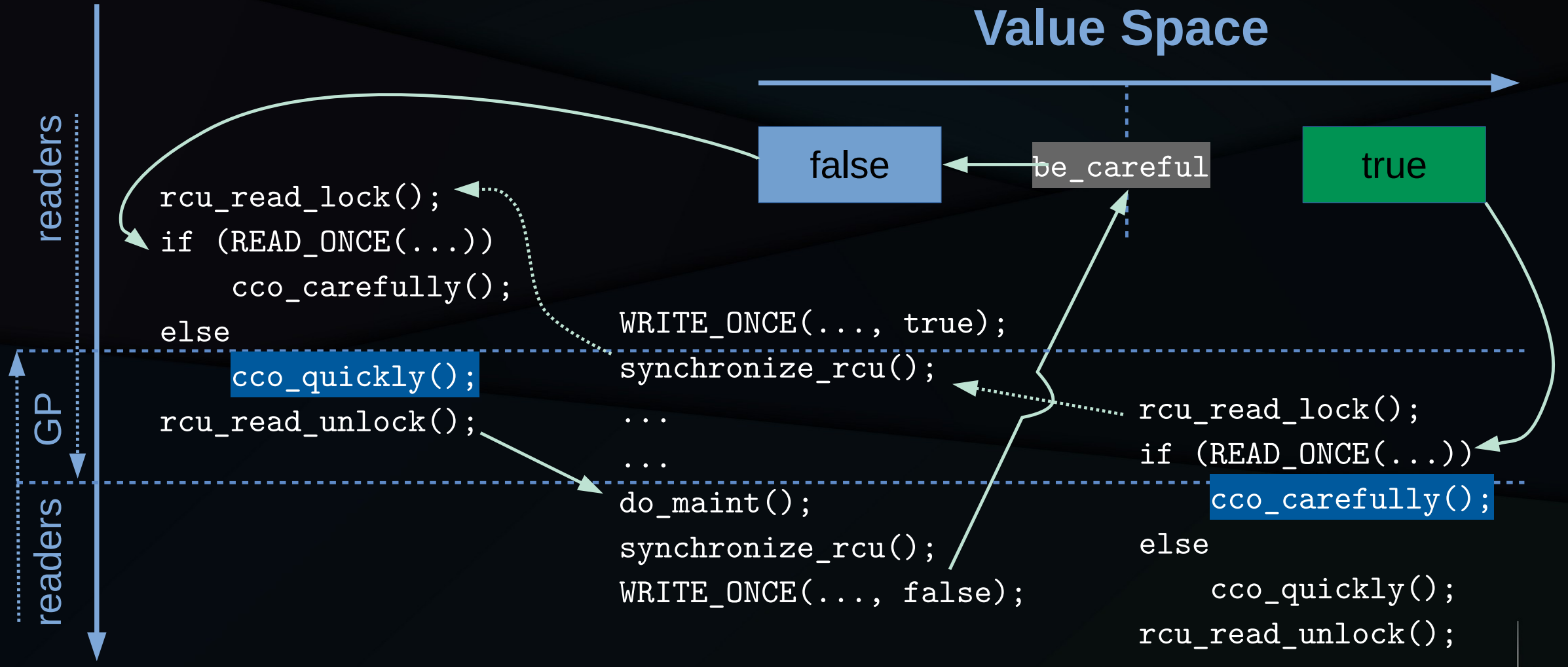
Phased State Change: Time & Space



Phased State Change: Time & Space



Phased State Change: Time & Space



Alternatives for maintenance-end synchronization?

RCU to Phased State Change

RCU to Phased State Change

- Add to wait-to-finish:
  - Checked state variable

Linux Kernel Usage?

Linux Kernel Usage?

- Example RCU-protected data structures:
  - task_struct: parent, cred, sighand, cgroups
  - cgroup: subsys
  - fdtable: fd
  - files_struct: fdt, fd_array
  - inode, super_block: [is]_fsnotify_marks

Linux Kernel Usage?

- include/linux/rcu_sync.h:
  - rcu_sync_init(), rcu_sync_enter_start(), rcu_sync_enter(), rcu_sync_exit()
- Used by per-CPU reader-writer semaphores
  - Light-weight readers in the absence of writers

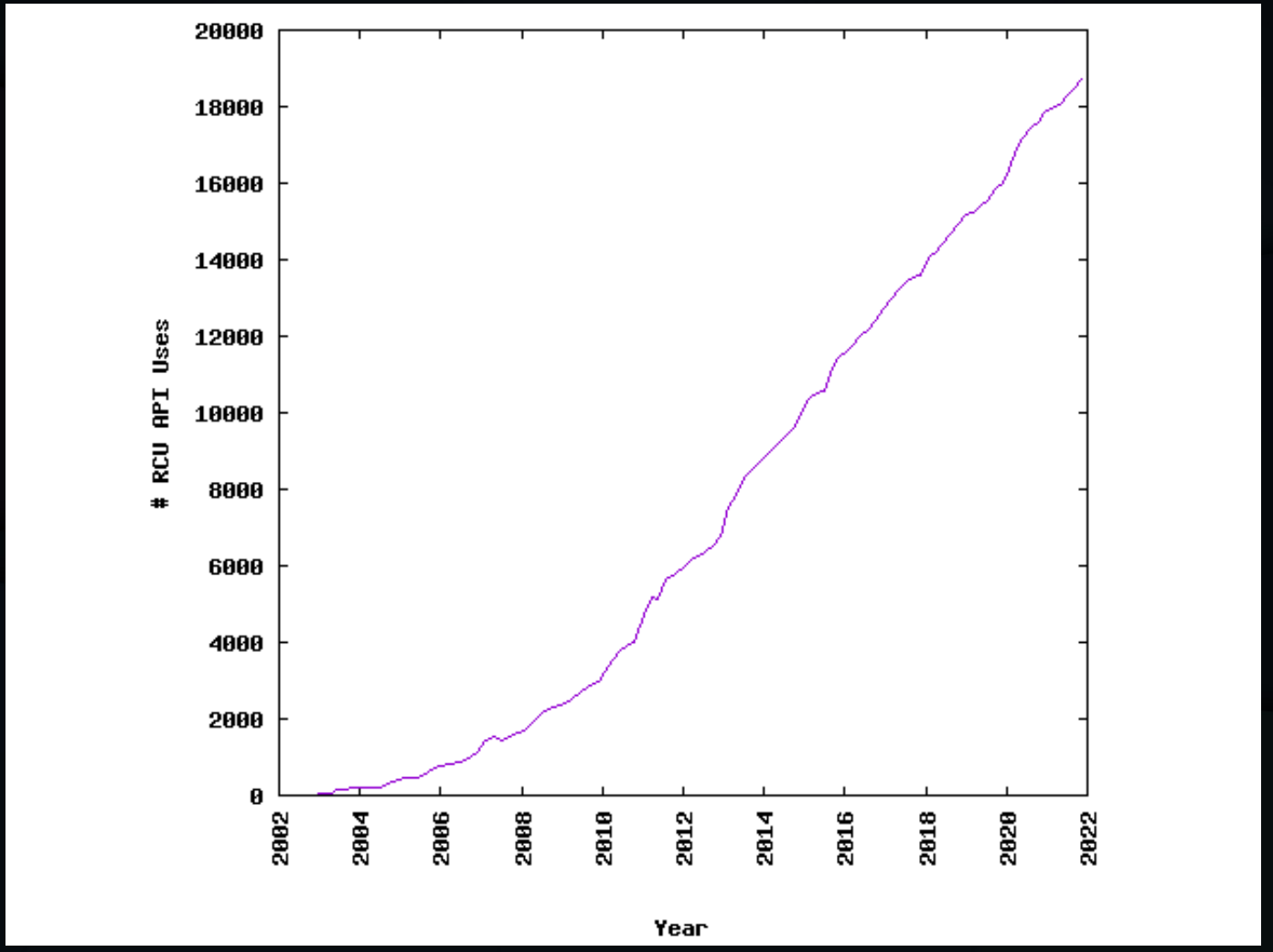
Overall RCU Usage in v5.15?

Overall RCU Usage in v5.15?

Subsystem	Calls to RCU APIs	Lines of Code *	Uses/KLoc
ipc	89	9,649	9.22
virt	65	8,465	7.68
net	7,411	1,192,884	6.21
security	595	104,460	5.70
kernel	1,689	406,868	4.15
mm	319	160,825	1.98
...	...	...	...
drivers	5,328	19,275,468	0.28
...	...	...	...
Total	18,710	27,557,130	0.68

* Only code in .h and .c files is counted, but all lines in such files are counted.

Overall RCU Usage Over Time?



http://www.rdrop.com/~paulmck/RCU/linuxusage.html

Debugging Linux-Kernel RCU Code

Debugging Linux-Kernel RCU Code

- Assertions: lockdep, sparse, KCSAN
- Prototype with userspace RCU
  - Use a real debugger!!!
- RCU CPU stall warnings

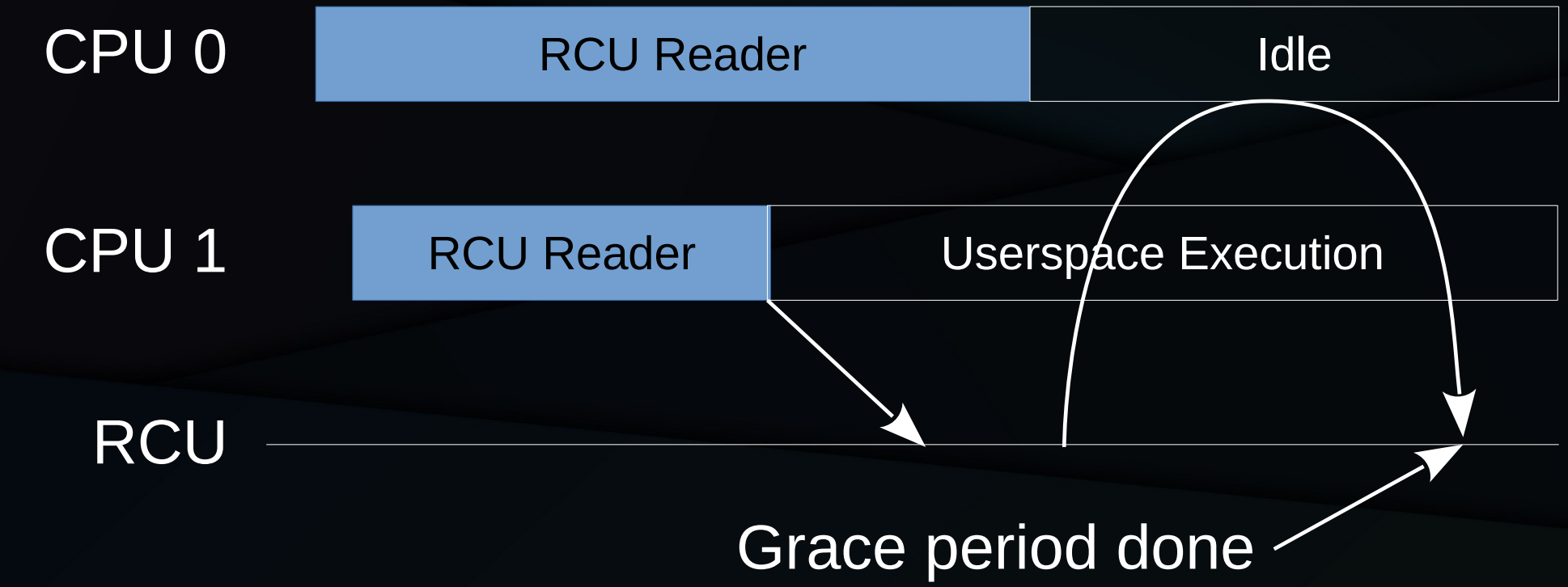
RCU lockdep Assertions

- Code really an RCU reader?
  - `rcu_read_lock_held()`
  - `rcu_read_lock_bh_held()`
  - `rcu_read_lock_sched_held()`
  - `rcu_read_lock_any_held()`
- Code an RCU reader or protected by specified lock(s)?
  - `rcu_dereference_check()`
  - `rcu_dereference_bh_check()`
  - `rcu_dereference_sched_check()`

RCU sparse and KCSAN Assertions

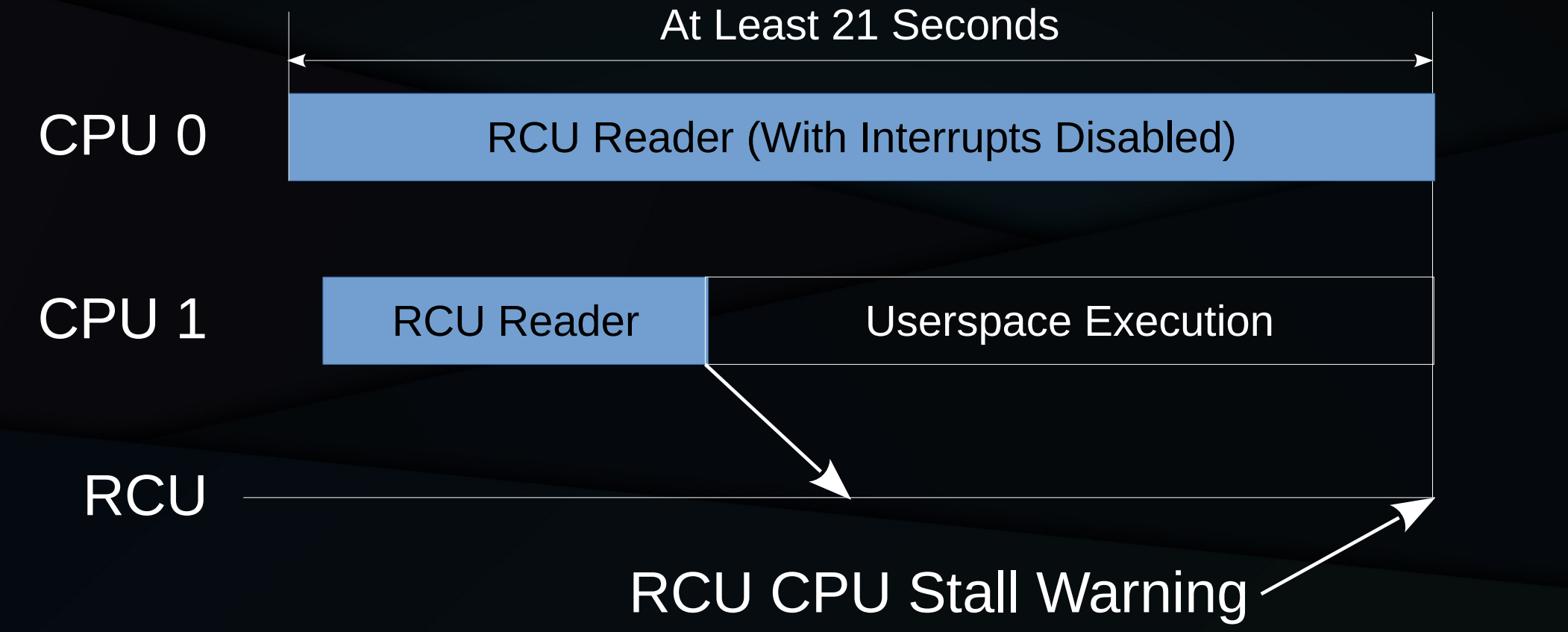
- Mark a pointer as an RCU-protected pointer with `__rcu`:
  - `void __rcu *map`
- KCSAN assertions (not just for RCU!)
  - `ASSERT_EXCLUSIVE_BITS()`
  - `ASSERT_EXCLUSIVE_ACCESS()`
  - `ASSERT_EXCLUSIVE_ACCESS_SCOPED()`
  - `ASSERT_EXCLUSIVE_WRITER()`
  - `ASSERT_EXCLUSIVE_WRITER_SCOPED()`

RCU With No Stall Warning



More complexity for CONFIG_PREEMPT=y kernels

RCU CPU Stall Warning



Normally indicates a problem outside of RCU

But I Don’t Want Stall Warnings!!!

- Boot with rcupdate.rcu_cpu_stall_suppress=1
- Boot with rcupdate.rcu_cpu_stall_timeout=NN (in seconds)
  - Or build with CONFIG_RCU_CPU_STALL_TIMEOUT=NN
  - 3 <= N <= 300, in seconds
- Why would you want to suppress CPU stall warnings?
  - Slow embedded system tested on faster development system
    - Increase timeout on slow embedded system (or decrease during test)
  - Throughput-based rip-and-replace cloud-computing environment
  - Embedded production environment where console output is ignored
    - Suppress warnings entirely
- But if response time matters, you do care about CPU stalls
  - Especially during development and testing: RCU CPU stall warnings find real bugs

Many Causes of RCU CPU Stalls

- For more information:
  - 2018 linux.conf.au presentation
    - Video: https://www.youtube.com/watch?v=23_GOr8Sz-E
    - Slides: https://static.sched.com/hosted_files/ossna2017/2b/stallwarning.2017.09.08a.pdf
  - Documentation in Linux-kernel source tree
    - Documentation/RCU/stallwarn.rst

Summary

Summary

- RCU synchronizes in space as well as time
  - But the time and space aspects are deeply intertwined
  - Enables near-zero-cost read-side synchronization
- Example RCU use case:
  - Quasi reader-writer lock (consistent config)
- RCU's dirty little secret

Summary

- RCU synchronizes in space as well as time
  - But the time and space aspects are deeply intertwined
  - Enables near-zero-cost read-side synchronization
- Example RCU use case:
  - Quasi reader-writer lock (consistent config)
- RCU's dirty little secret:
  - RCU is dead simple

Summary

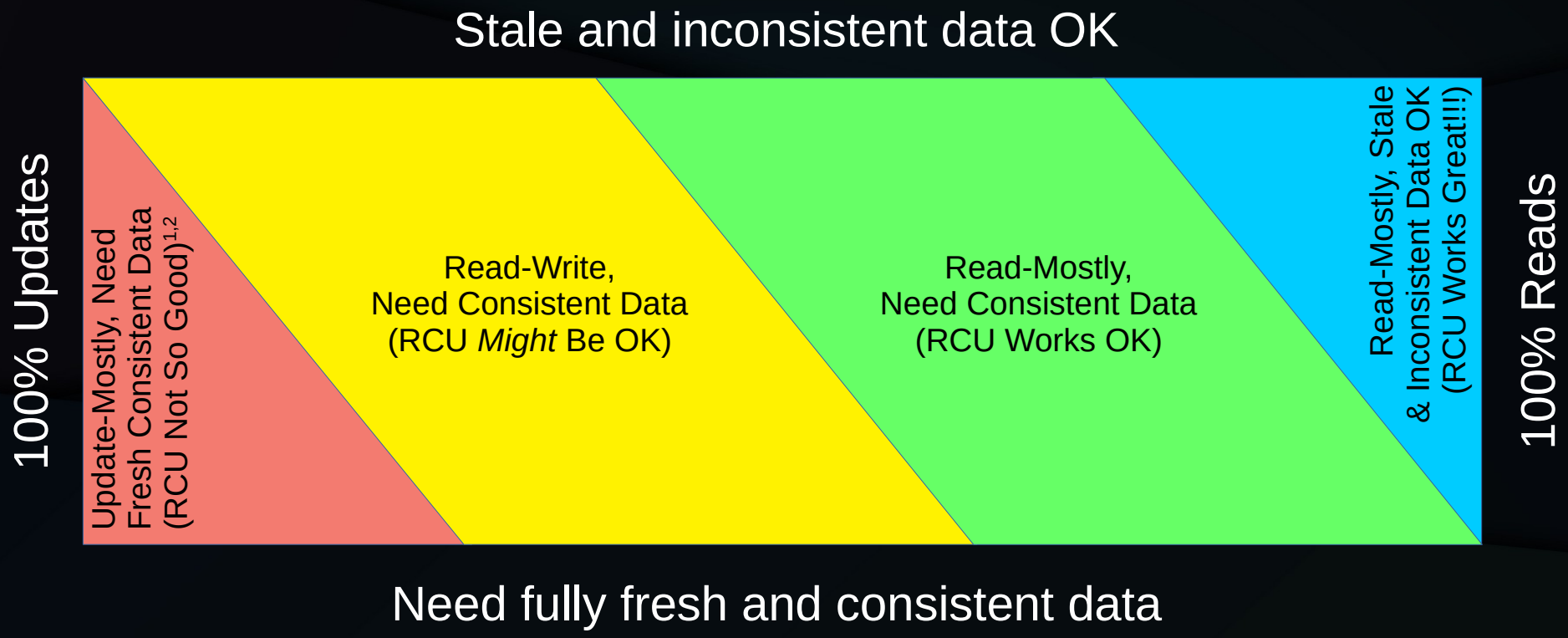
- RCU synchronizes in space as well as time
  - But the time and space aspects are deeply intertwined
  - Enables near-zero-cost read-side synchronization
- Example RCU use case:
  - Quasi reader-writer lock (consistent config)
- RCU's dirty little secret:
  - RCU is dead simple, but in order to make good used of it, you must change the way that you think about your problem

Summary

- “I hear and I forget.”
- “I see and I remember.”
- “I do and I understand.”
- To really understand RCU, play with it.

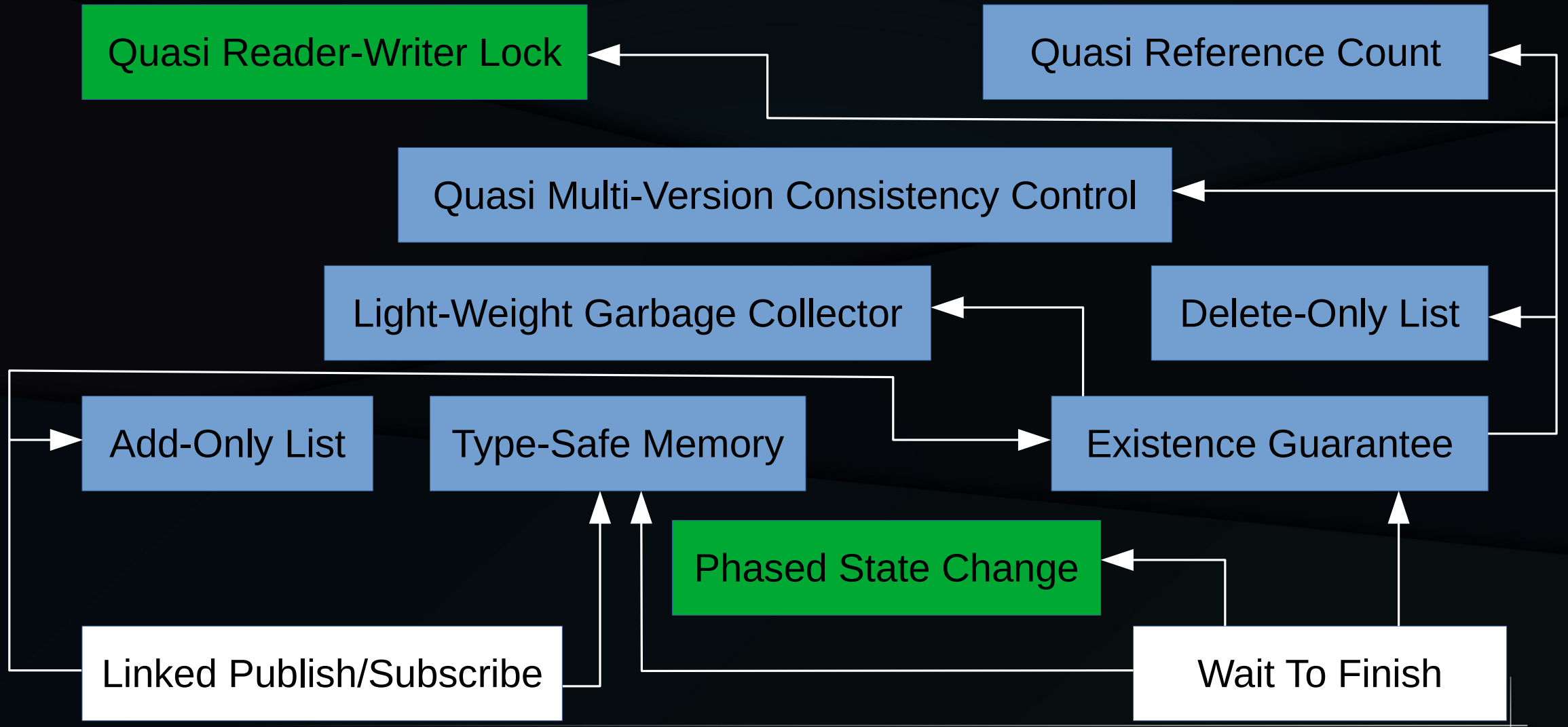
Questions?

RCU Area of Applicability (Redux)



1. RCU provides ABA protection for update-friendly mechanisms
2. RCU provides bounded wait-free read-side primitives for real-time use

You Are Here; More in February



For More Information

- “RCU Usage In the Linux Kernel: One Decade Later”:
  - http://www.rdrop.com/~paulmck/techreports/survey.2012.09.17a.pdf
  - http://www.rdrop.com/~paulmck/techreports/RCUUsage.2013.02.24a.pdf
  - 2020 update: https://dl.acm.org/doi/10.1145/3421473.3421481
- “Structured Deferral: Synchronization via Procrastination”: http://doi.acm.org/10.1145/2488364.2488549
- Linux-kernel RCU API, 2019 Edition: https://lwn.net/Articles/777036/
- “Stupid RCU Tricks: So you want to torture RCU?”: https://paulmck.livejournal.com/61432.html
- Documentation/RCU/* in kernel source
- “Is Parallel Programming Hard, And, If So, What Can You Do About It?”, “Deferred Processing” chapter: https://mirrors.edge.kernel.org/pub/linux/kernel/people/paulmck/perfbook/perfbook.html
- Folly-library RCU implementation (also C-language user-space RCU)
- Large piles of information: http://www.rdrop.com/~paulmck/RCU/



Maged Michael

ACM 杰出科学家，Facebook技术专家

Maged M. Michael是ACM杰出科学家。他是Hazard指针、无锁malloc和几种并发数据结构算法的发明者。他因其关于可扩展的无锁动态内存分配的论文而获得2014年ACM SIGPLAN最具影响力的PLDI论文奖。作为国际知名的计算机科学家，他现在服务于Facebook，在此之前他是IBM沃森研究中心的资深研究员。他从罗切斯特大学获得了计算机科学博士学位。

Hazard Pointer Essentials in C++

Maged Michael

C++ and System Software Summit

September 28-29, 2022

Hazard Pointers

A **hazard pointer** is a **pointer** that can be **written by one thread** at a time.

Why Use Hazard Pointers?

Hazard pointers are used to **protect**
access to dynamic **objects**
that may be **removed concurrently**
from being **reclaimed**.

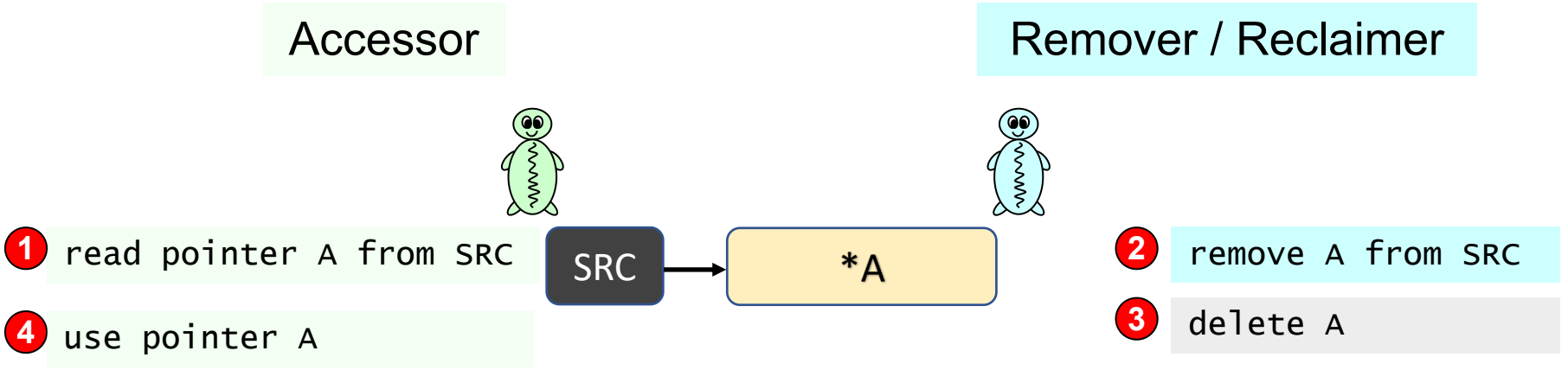
Hazard pointer protection is **fast**, **scalable**, and **non-blocking**.

Hazard Pointer Protection

If a hazard pointer **points to an object**
before its removal,
then as long as the hazard pointer **remains unchanged**
the object will **not be reclaimed**.

.

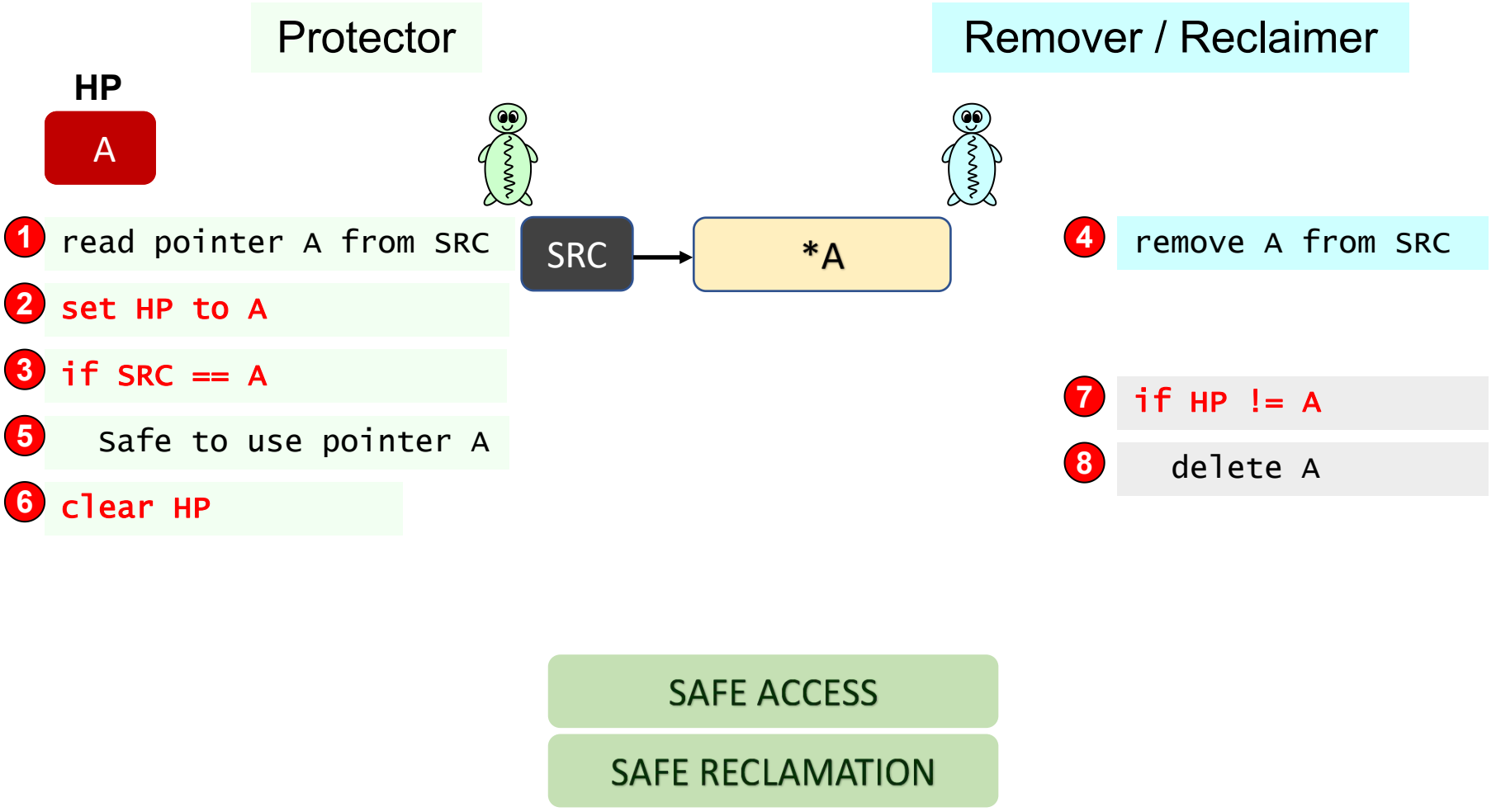
Unsafe Reclamation



ERROR

UNSAFE ACCESS TO RECLAIMED OBJECT

Basic Hazard Pointer Algorithm



Hazard Pointers in the C++ Standard

- Part of Concurrency Technical Specification v2 (expected 2023)
  - Working draft N4895 (wg21.link/n4895).
- Proposal for inclusion in C++26 standard:
  - Subset of Concurrency TS v2.
  - Reviewed by SG1 (Concurrency Study Group). Voted to advance to API.
  - Ongoing review of proposed interface by LEWG (Library Evolution Working Group).
  - Proposal document P2530 (wg21.link/p2530).
- Reference Implementation:
  - Open source: github.com/facebook/folly/blob/main/folly/synchronization/Hazptr.h
  - Superset of the C++26 and Concurrency TS v2 interfaces.
  - Heavily used in production.

C++26 Proposed Interface

Class template for objects protectable by hazard pointers

Class for hazard pointer holders to manage hazard pointers

Free functions that manage hazard pointer holders

Objects Protectable by Hazard Pointers

```
template <typename T, typename D = default_delete<T>>
class hazard_pointer_obj_base {
public:
  void retire(D d = D()) noexcept;
};
```

retire():

- Called after object is removed (no longer reachable).
- Called at most once.
- Retired object is to be reclaimed when no longer protected

Code Example

```
class Foo : public folly::hazard_pointer_obj_base<Foo> { /* Foo members */ };

std::atomic<Foo*> src_;

void update(Foo* newptr) {
  Foo* oldptr = src_.exchange(newptr); // remove *oldptr
  oldptr->retire();
}
```

Hazard Pointer Interface

Class for hazard pointer holders

```
class hazard_pointer {
  hazard_pointer() noexcept; // Empty
  hazard_pointer(hazard_pointer&&) noexcept;
  hazard_pointer& operator=(hazard_pointer&&) noexcept;
  ~hazard_pointer();
  [[nodiscard]] bool empty() const noexcept;
  template <typename T> T* protect(const atomic<T*>& src) noexcept;
  template <typename T> bool try_protect(T*& ptr, const atomic<T*>& src) noexcept;
  template <typename T> void reset_protection(const T* ptr) noexcept;
  void reset_protection(nullptr_t = nullptr) noexcept;
  void swap(hazard_pointer&) noexcept;
};
```

Free functions that manage hazard pointer holders

```
hazard_pointer make_hazard_pointer(); // Construct a non-empty hazard pointer
void swap(hazard_pointer&, hazard_pointer&) noexcept;
```

Basic Hazard Pointer Interface

```
class hazard_pointer {
  hazard_pointer() noexcept; // Empty
  template <typename T> T* protect(const atomic<T*>& src) noexcept;
};

hazard_pointer make_hazard_pointer(); // Construct a non-empty hazard pointer
```

- **Default constructor** constructs **empty** hazard pointer holder that does not own an actual hazard pointer.
- **make_hazard_pointer**() constructs a nonempty hazard pointer holder that owns a hazard pointer.
- Only **nonempty** holders can be used to **protect** objects.
- **protect(src)** returns a pointer to an object that was reachable from src and is protected by the owned hazard pointer as long as the hazard pointer is active and remains unchanged.

Code Example:

```
hazard_pointer h = make_hazard_pointer();
Foo* ptr = h.protect(src_);
// *ptr is protected as long as h is alive and unchanged
```

Motivating Example Using Read-Write Lock

Shared data, frequently-read, infrequently-updated

```
class Foo { /* Foo members */ };
```

```
Foo* ptr_;
Std::shared_mutex mutex_;
```

```
Result read(Func fn) { // Called frequently
  std::shared_lock rlock(mutex_);
  return fn(ptr_);
}
```

```
Void update(Foo* newptr) { // Called infrequently
  Foo* oldptr;
  { std::unique_lock wlock(mutex_);
    oldptr = std::exchange(ptr_, newptr); }
  delete oldptr;
}
```

Read lock executes expensive atomic instructions in the fast path

Readers block writers

Writers block readers

User functions are not allowed to block or to call **update**

Same Example but Using Hazard Pointers

Shared data, frequently-read, infrequently-updated

```
class Foo : public hazard_pointer_obj_base<Foo> { /* Foo members */ };
```

```
std::atomic<Foo*> src_;
```

```
Result read(Func fn) { // Called frequently
 hazard_pointer h = make_hazard_pointer();
 Foo* ptr = h.protect(src_);
 return fn(ptr);
}
```

```
Void update(Foo* newptr) { // Called infrequently
  Foo* oldptr = src_.exchange(newptr);
  oldptr->retire();
}
```

Hazard pointer protection has no expensive atomic instructions in the fast path

Readers do not block writers

Writers do not block readers

User functions are allowed to block and to call **update**

Other Use Examples of C++26 Interface

Hand-over-hand traversal

- Full example: github.com/facebook/folly/blob/main/folly/synchronization/example/HazptrSWMRSet.h
- Uses two hazard pointers and **swaps** them

Iterators

- Open source example: Folly ConcurrentHashMap (github.com/facebook/folly/blob/main/folly/concurrency/ConcurrentHashMap.h)
- Uses default constructor and **move operator** in addition to make_hazard_pointer and protect.
- Allows user code iterators that are fast, scalable, and flexible.

Example:

```
for (auto it = map.begin(); it != map.end(); ++it)
  userOperation(it);
```

Basic Implementation Overview

- Reference implementation:
  github.com/facebook/folly/blob/main/folly/synchronization/Hazptr.h

- Global structures:
  - List of hazard pointers
  - List of retired objects

- Thread local:
  - Cache of hazard pointers

- Main operations:
  - Management of hazard pointers
    - Allocation: push in global list and owner thread's cache.
  - Retirement of objects
    - Push in list of retired objects
  - Protection of objects using hazard pointers
  - Asynchronous reclamation

Protection functions

- Bool try_protect(ptr, src):
  - Write ptr to hazard pointer
  - Return src == ptr
- protect(src):
  - Read ptr from src
  - Call try_protect(ptr, src)
  - Return ptr if try_protect returns true
  - Otherwise start over
- reset_protection(ptr):
  - Write ptr to hazard pointer
  - Caller knows that ptr is not yet retired

Asynchronous Reclamation

- Why? To bound the number of not-yet-reclaimed reclaimable objects.
- Amortization to achieve amortized expected constant time per reclaimed object.
- When the number of retired objects reaches the greater of 1000 and twice the number of hazard pointers, i.e., NumberOfRetired >= max(1000, 2*NumberOfHazardPointers).
- Otherwise, every 2 seconds.
- Asynchronous reclamation steps:
    1. Extract objects from the global list of retired objects.
    2. Read values from the list of hazard pointers and add values to a local hash set.
    3. Match addresses of retired objects with pointers in local hash set (that were read from the hazard pointers).
    4. Push matched objects back into the list of retired objects.
    5. Unmatched objects are safe to reclaim.

Performance

```
================================================================================
Protection of a read from a shared object           1 thread         10 threads
================================================================================
folly::hazard_pointer                                 3.9 ns             4.8 ns
folly::hazptr_local<1>  (optimized destruction)       2.2 ns             3.1 ns
folly::hazard_pointer preconstructed                  < 1 ns             1.1 ns
================================================================================
```

From single socket runs on a dedicated commodity server

Other Safe Reclamation Methods

1. Reader-writer locks: std::shared_mutex
2. Reference counting: std::shared_ptr, std::atomic<std::shared_ptr>
3. RCU (Read-Copy-Update): C++26 proposal P2545 (wg21.link/p2545).

- Rwlocks and RCU protect many objects but are sensitive to blocking.
- Hazard pointers and reference counting are flexible with blocking but need to be specific about protected objects.
- Hazard pointers and RCU defer reclamation. Rwlocks and ref counting reclaim immediately.

Access Protection	Reader-Writer locks	Reference counting	RCU	HP
Speed	-	--	++	++
Scalability	-	--	++	++
Blocking / duration	-	++	-	++
Complex protection	++	-	++	-
Reclamation timing	++	++	-	-
Bound on unreclaimed	++ immediate	++ immediate.	- no	+

Summary

- Hazard pointers safe reclamation method
- Fast, scalable, and non-blocking protection

Useful for ...

- **Speed and scalability.**
- **Indefinite protection, e.g., iterators, arbitrary user callbacks.**

May not be the right tool ...

- **If immediate reclamation is required. Use locking and/or reference counting.**
- **For irregular protection patterns. Use locking or RCU.**

References

- Hazard Pointers: Safe Memory Reclamation for Lock-Free Objects, *IEEE Transactions on Parallel and Distributed Systems.* 15 (8): 491 – 504, June 2004.
- C++26 proposal (wg21.link/p2530).
- Working Draft, Extensions to C++ for Concurrency Version 2 (wg21.link/n4895).
- **github.com/facebook/folly/blob/main/folly/synchronization/Hazptr.h**

THANK YOU!



John Lakos

大规模软件设计世界级权威，彭博集团BDE基础开发部掌门人

John 是 C++大规模组件化开发方法和过程的世界级权威，现任彭博集团（Bloomberg LP）基础开发部（BDE）的掌门人、该部门负责开发大规模、可重用的C++套件。John也负责训练和辅导彭博集团软件基础架构团队的所有技术主管。他也是彭博集团BDE/BSL开源创新库的发起人。 他同时也是C++标准委员会负责标准库工作组的资深委员。他的代表作品《大规模C++软件设计》是大规模软件开发方面的权威著作。

CPP-Summit 2022

CPP-Summit

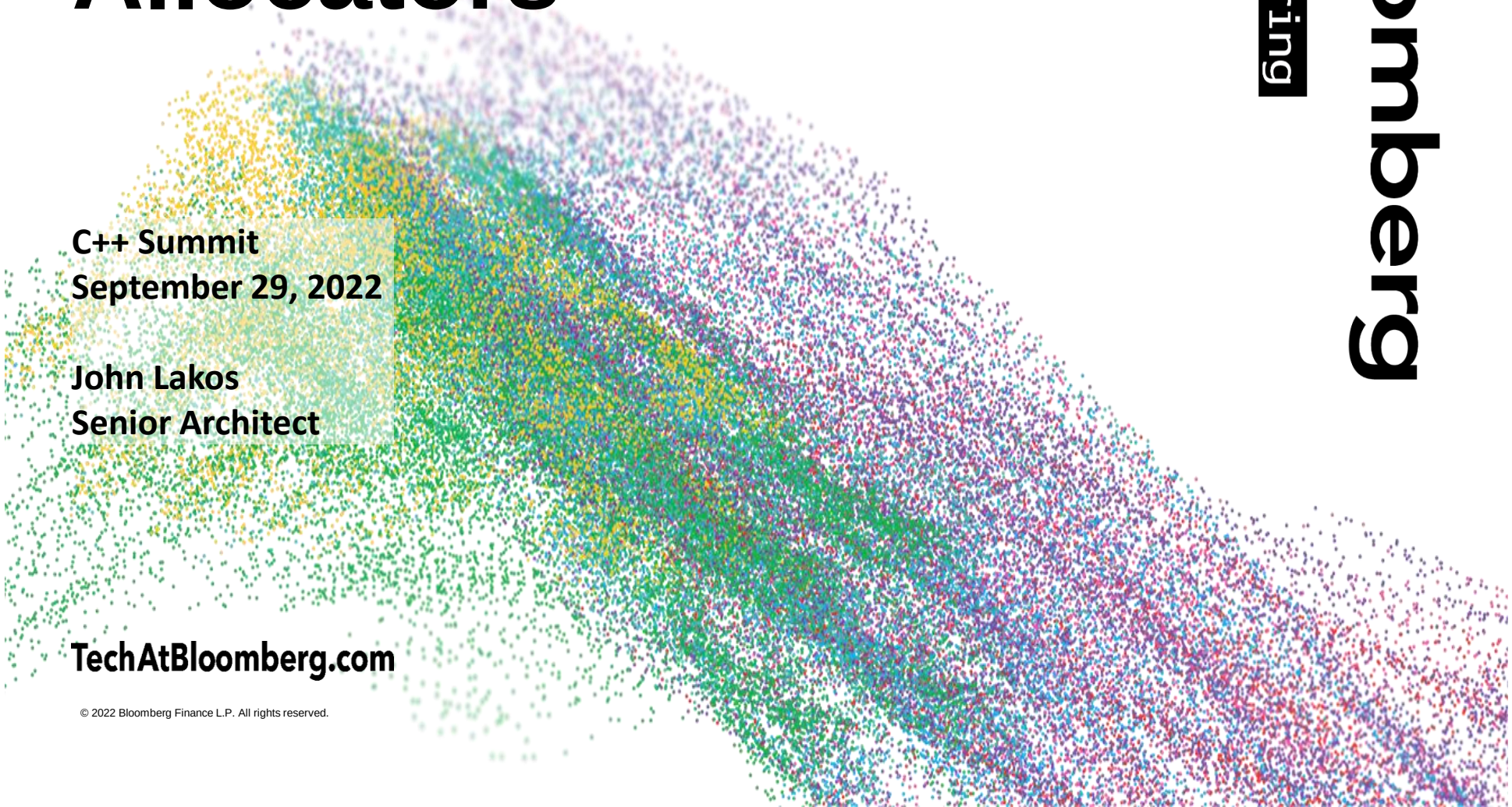
Local (Arena) Memory Allocators

John Lakos
Senior Architect

Local (Arena) Memory Allocators

Bloomberg

Engineering



C++ Summit
September 29, 2022

John Lakos
Senior Architect

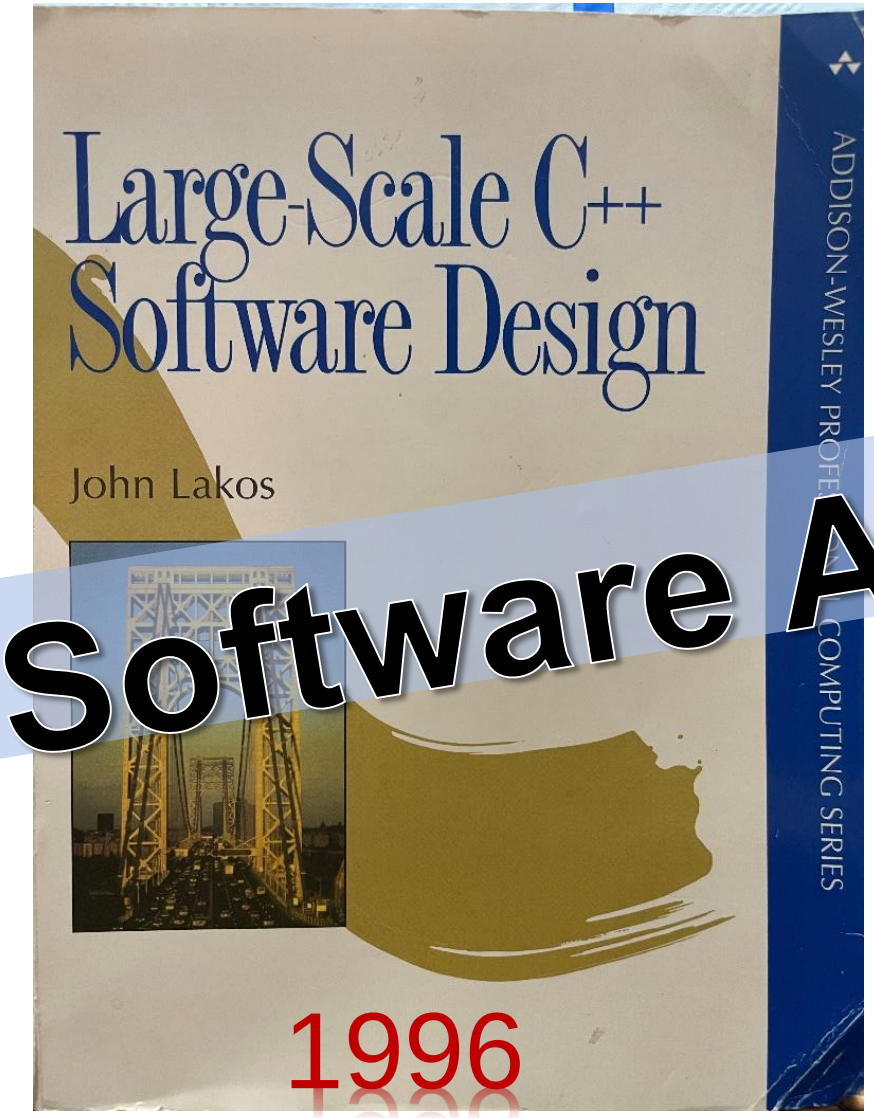
TechAtBloomberg.com

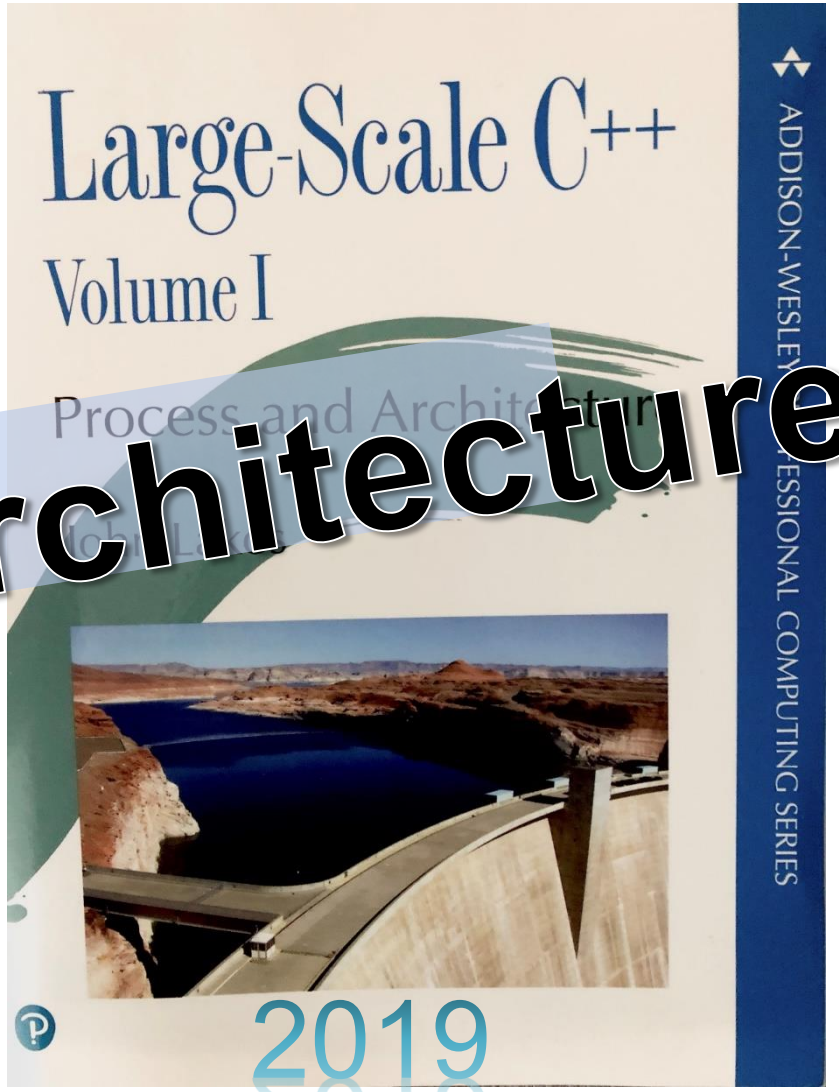
© 2022 Bloomberg Finance L.P. All rights reserved.

Who am I?

John Lakos

- **I work at Bloomberg in New York City** (c. 2001)
  - In the Chief Technology Office (c. 2022)
- **I'm active with the C++ Standards Committee:**
  - Initial engagement (c. 2005)
  - Voting member (c. 2007)
- **I've written a few books:**
  - Large-Scale C++ Software Design (c. 1996)
  - Large-Scale C++ (Volume I): Process and Architecture (c. 2019)
  - Embracing Modern C++ *Safely* (c. 2021)





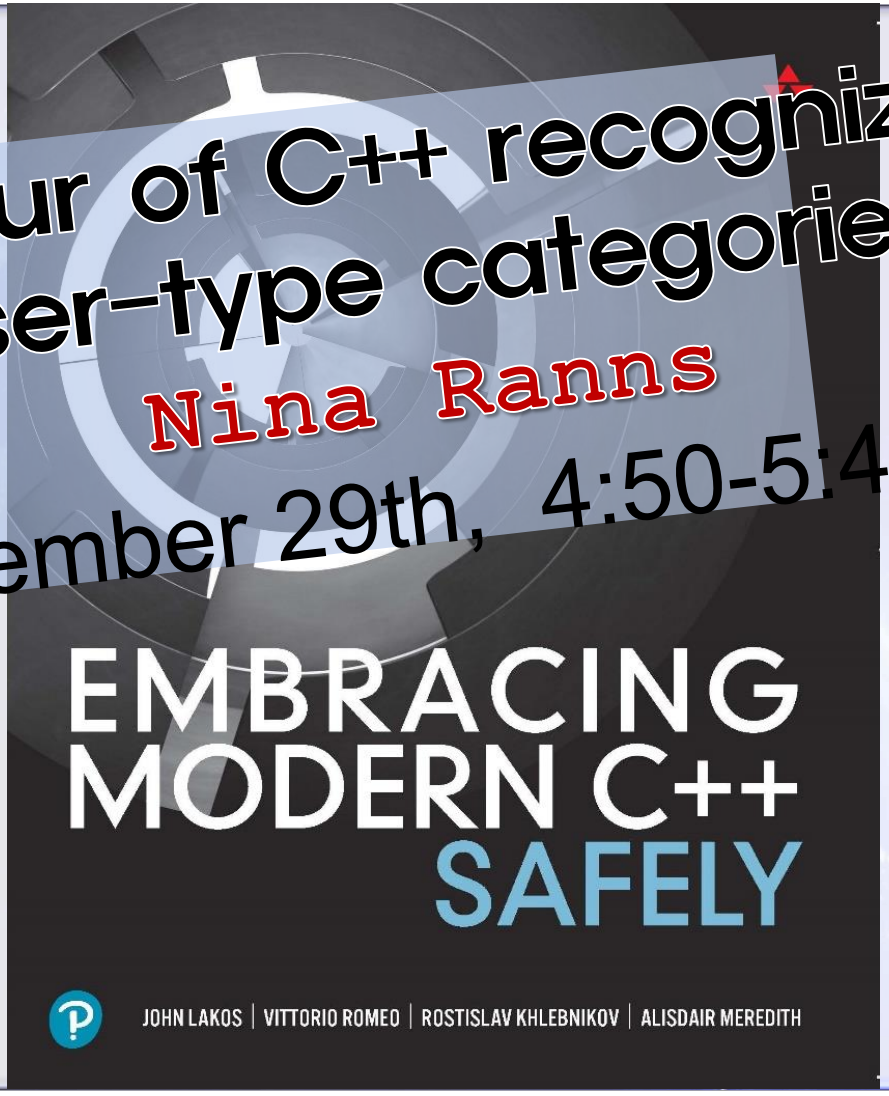
Software Architecture



A tour of C++ recognized user-type categories

Nina Ranns

September 29th, 4:50-5:40 PM



Copyright Notice

© 2017 Bloomberg L.P. Permission is granted to copy, distribute, and display this material, and to make derivative works and commercial use of it. The information in this material is provided "AS IS", without warranty of any kind. Neither Bloomberg nor any employee guarantees the correctness or completeness of such information. Bloomberg, its employees, and its affiliated entities and persons shall not be liable, directly or indirectly, in any way, for any inaccuracies, errors or omissions in such information. Nothing herein should be interpreted as stating the opinions, policies, recommendations, or positions of Bloomberg.

Abstract

The runtime implications of the physical location of allocated memory are sometimes overlooked, even in the most performance-critical code. In this talk, we will examine how the performance of long-running systems can degrade when using just one global allocator (e.g., via `new` and `delete`). We will contrast the use of global allocators with various kinds of *local* allocators, i.e., allocators that allocate memory for a well-defined subset of objects in the system. We will also demonstrate how local allocators can reduce, if not entirely prevent, degradation seen in systems that rely solely on the global allocator. Six dimensions — *fragmentability*, *allocation density*, *variation*, *utilization*, *locality*, and *contention* — will be introduced to help characterize a given subsystem, assess the potential for accelerating its runtime performance, and, where appropriate, aid in determining the best local allocator to do so. *Empirical evidence* will be presented to demonstrate that introducing an appropriate local allocator can often result in substantial reductions in run times (compared with a similar system relying solely on just a single, global allocator).

Important Recurring Questions

Are memory allocators really worth the trouble?

- ❑ What situations merit their use?
- ❑ How are they applied effectively?
- ❑ What's the performance impact?

Outline

1. Introduction and Background
   What are memory allocators, and why are they useful?
2. Understanding the Problem
   What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   When and how do you use which allocator, and why?
4. Conclusions
   What must we remember about memory allocators?

Outline

1. Introduction and Background
   - What are memory allocators, and why are they useful?
2. Understanding the Problem
   - What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   - When and how do you use which allocator, and why?
4. Conclusions
   - What must we remember about memory allocators?

Important Questions

Why do we like the C++ language?

- It enables us to “fine-tune” at a low level when needed.
- It can deliver very high runtime performance.

Important Questions

Why (should) we care about memory allocators?

- They enable us to “fine-tune” at a low level when needed.
- They can help to improve runtime performance.

Important Questions

What are the benefits?

- ❑ Not all memory is alike:
  - ✓ Fast, Shared, Protected, Mapped
- ❑ Other qualitative benefits:
  - ✓ Testing, Debugging, Measuring
- ❑ Enhanced runtime performance:
  - ✓ Better Locality, Less Contention

Important Questions

What are the benefits?

(anecdotal)

Bear Stearns (c. 1997)

System's (coalescing) allocator optimized for allocation, not deallocation.

Important Questions

What are the benefits?

(anecdotal)

Bloomberg (c. 2002)

Process global `static` memory saved/restored via memory-mapped IO.

Important Questions

What are the benefits?

(anecdotal)

Bloomberg (c. 2006)

User interfaces observed to be *zippier* when using local allocator.

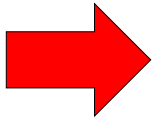
Important Questions

What are the common arguments against?

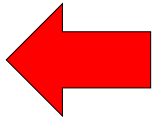
- Requires more up-front design effort
- Complicates user interfaces
- May actually degrade performance:
  - No special allocator needed
  - Poorly chosen allocator supplied

Addressing Allocator Concerns

These are valid concerns!



- Requires more up-front design effort
- Complicates user interfaces
- May actually degrade performance:
  - No special allocator needed
  - Poorly chosen allocator supplied



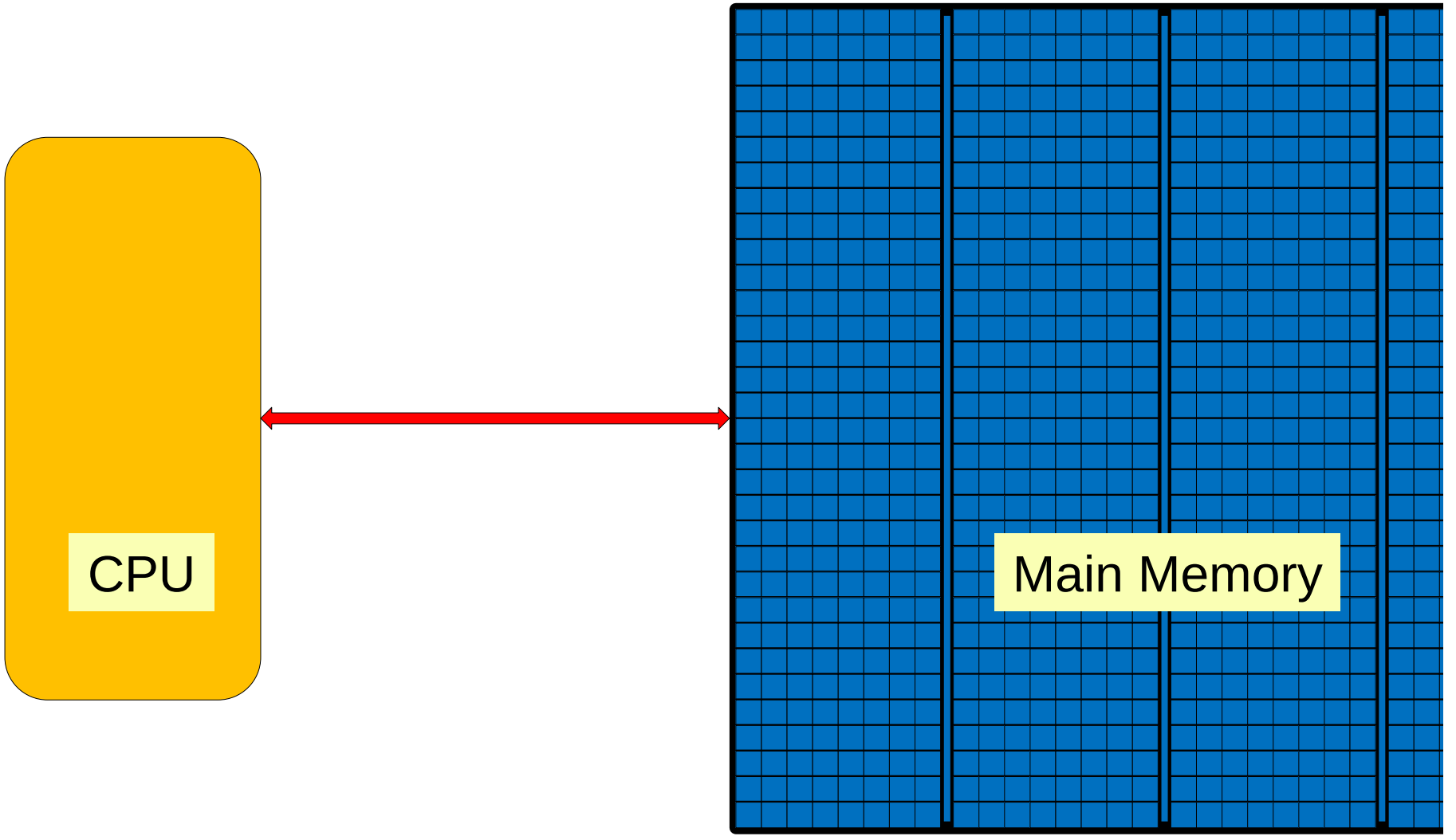
They can be addressed only with:

- **Well-supported facts**
- **Careful measurement**

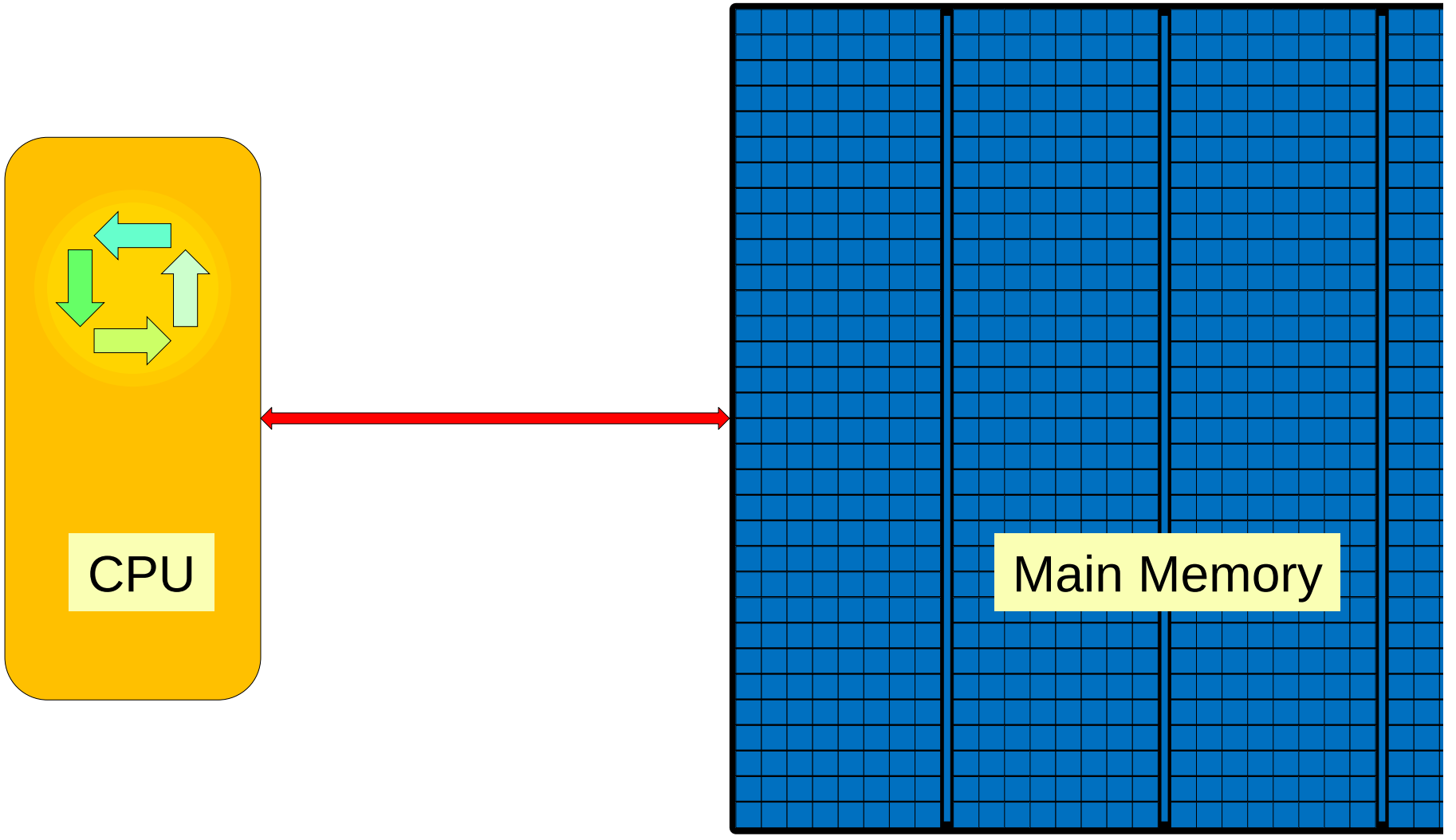
Review of Computer Memory

Main Memory

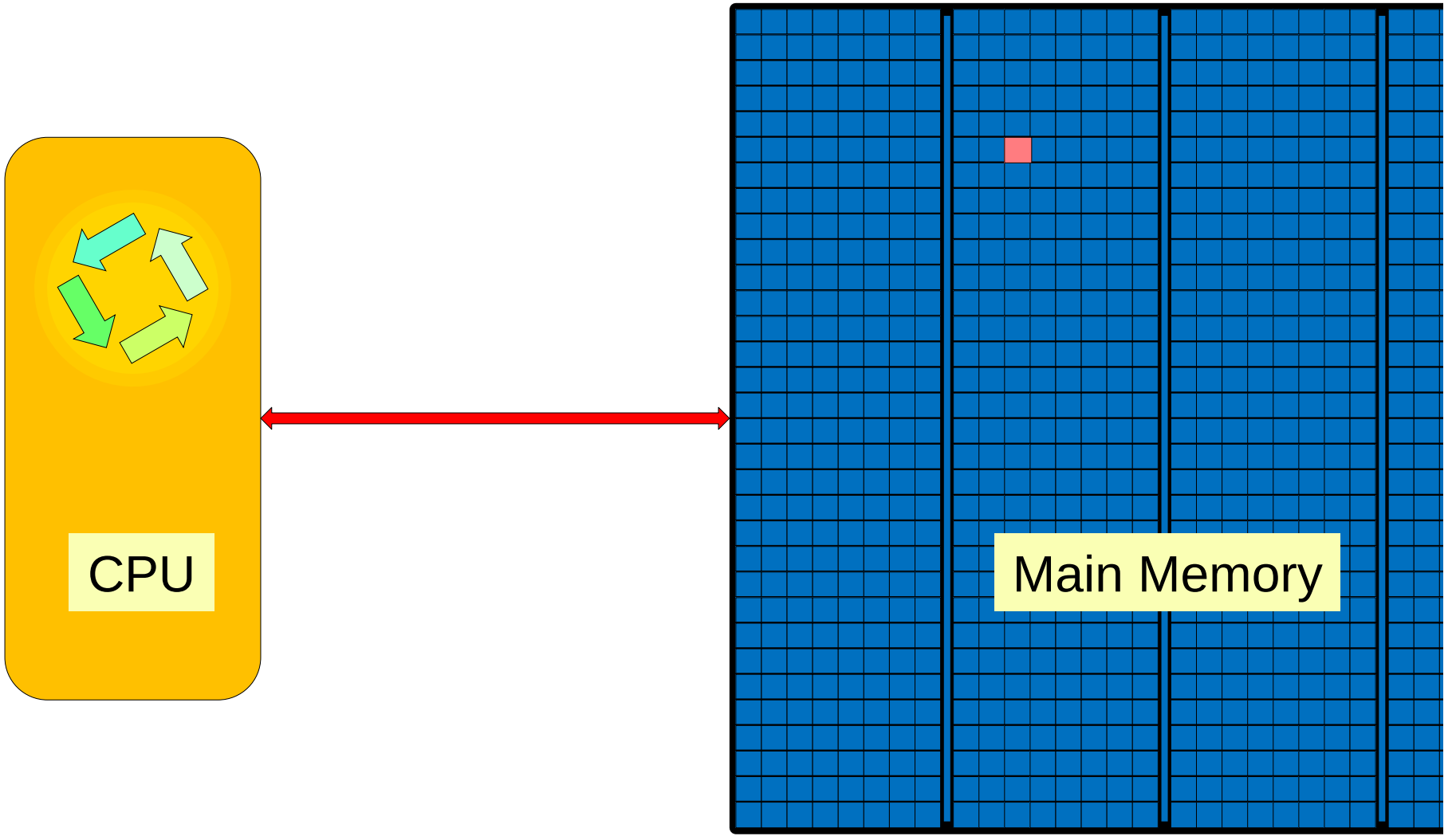
Review of Computer Memory



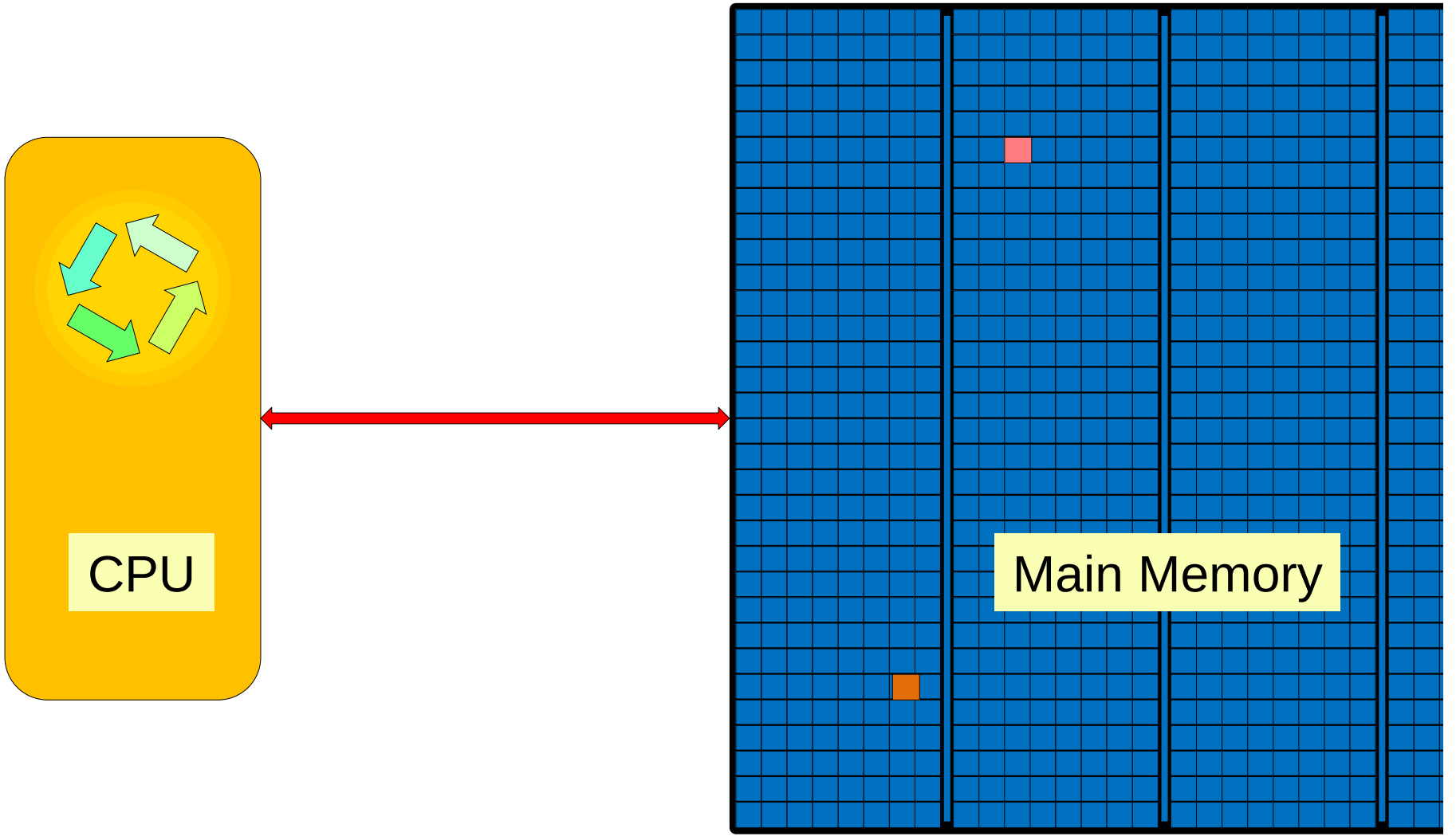
Review of Computer Memory



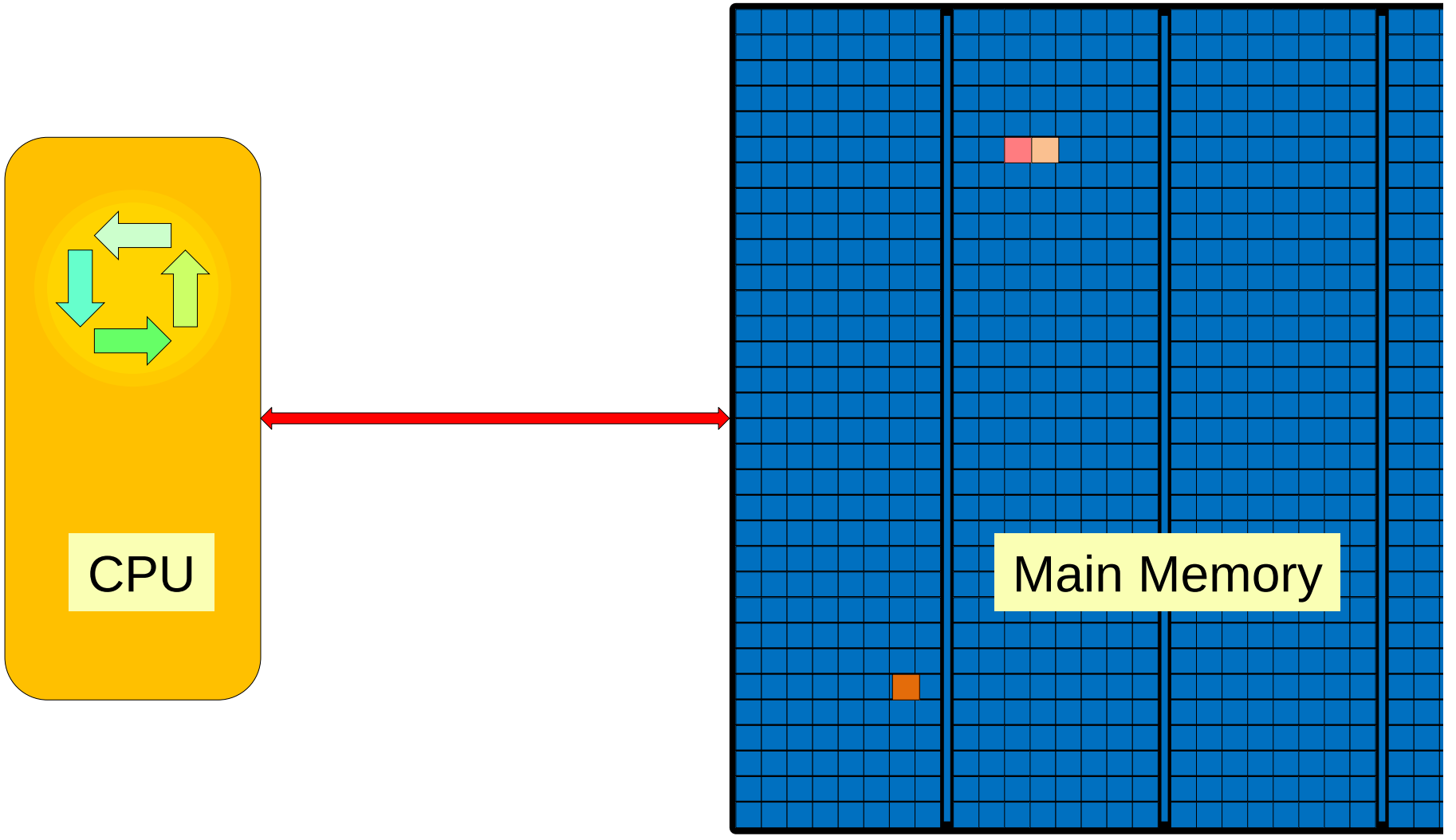
Review of Computer Memory



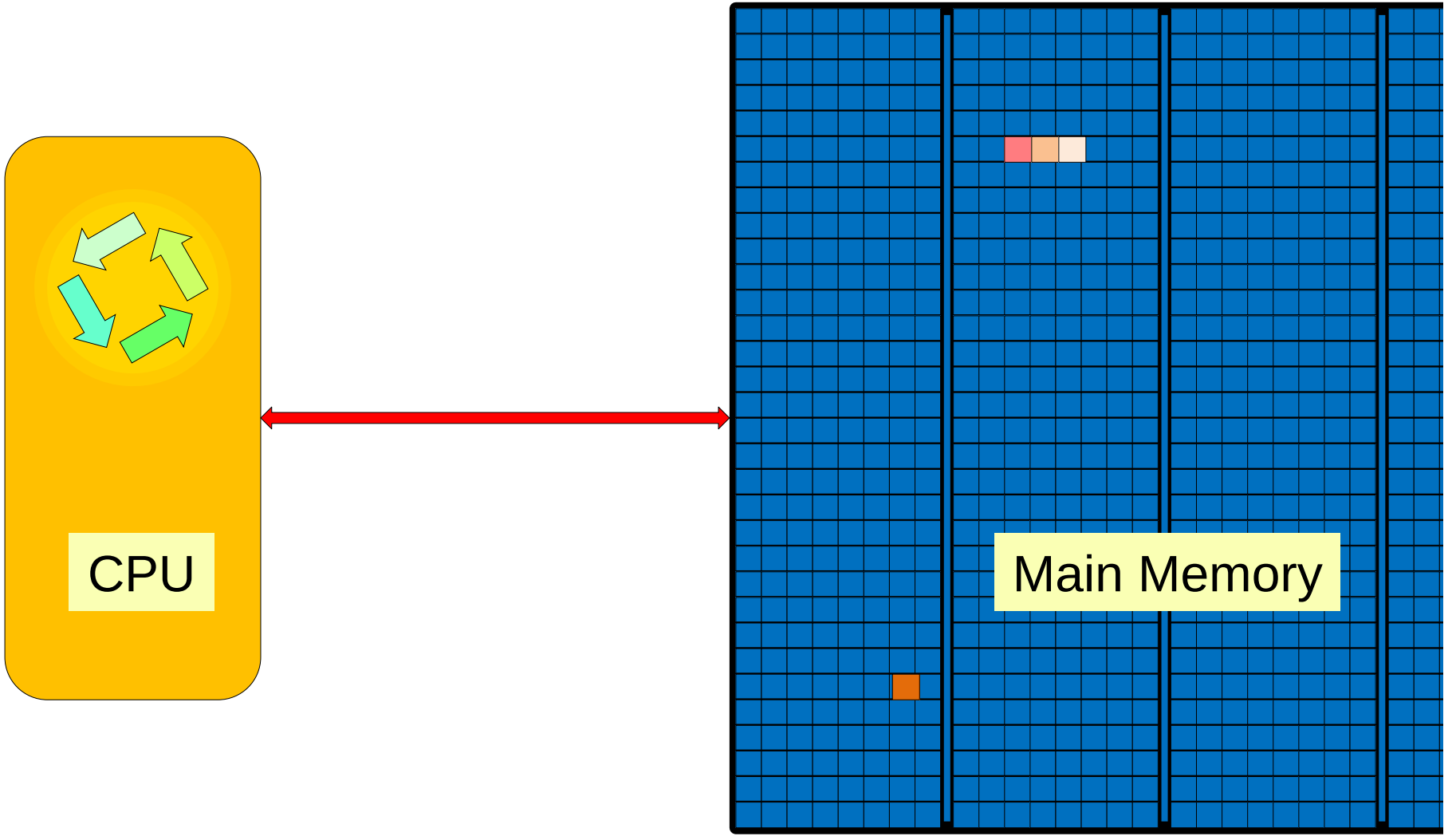
Review of Computer Memory



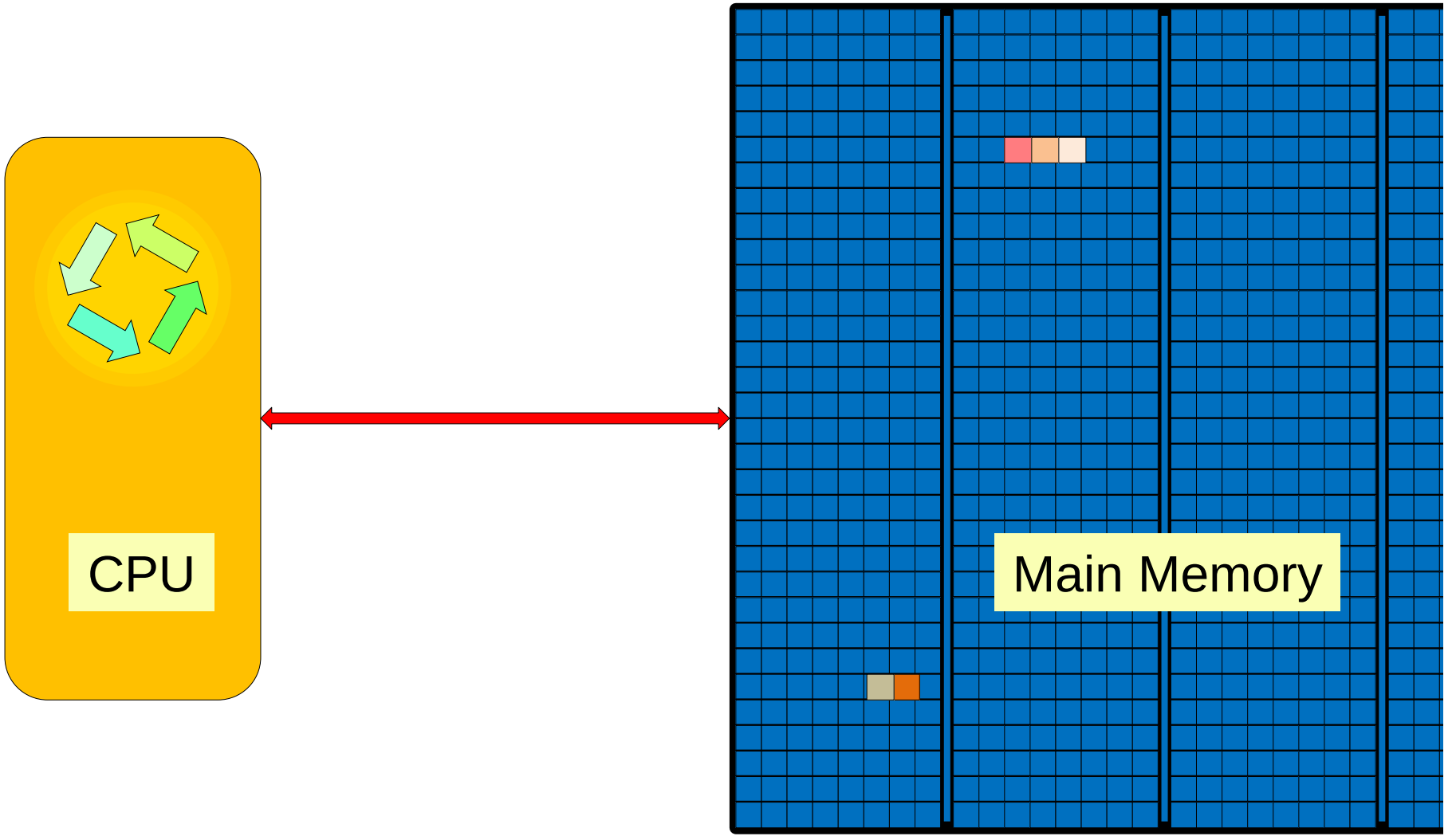
Review of Computer Memory



Review of Computer Memory



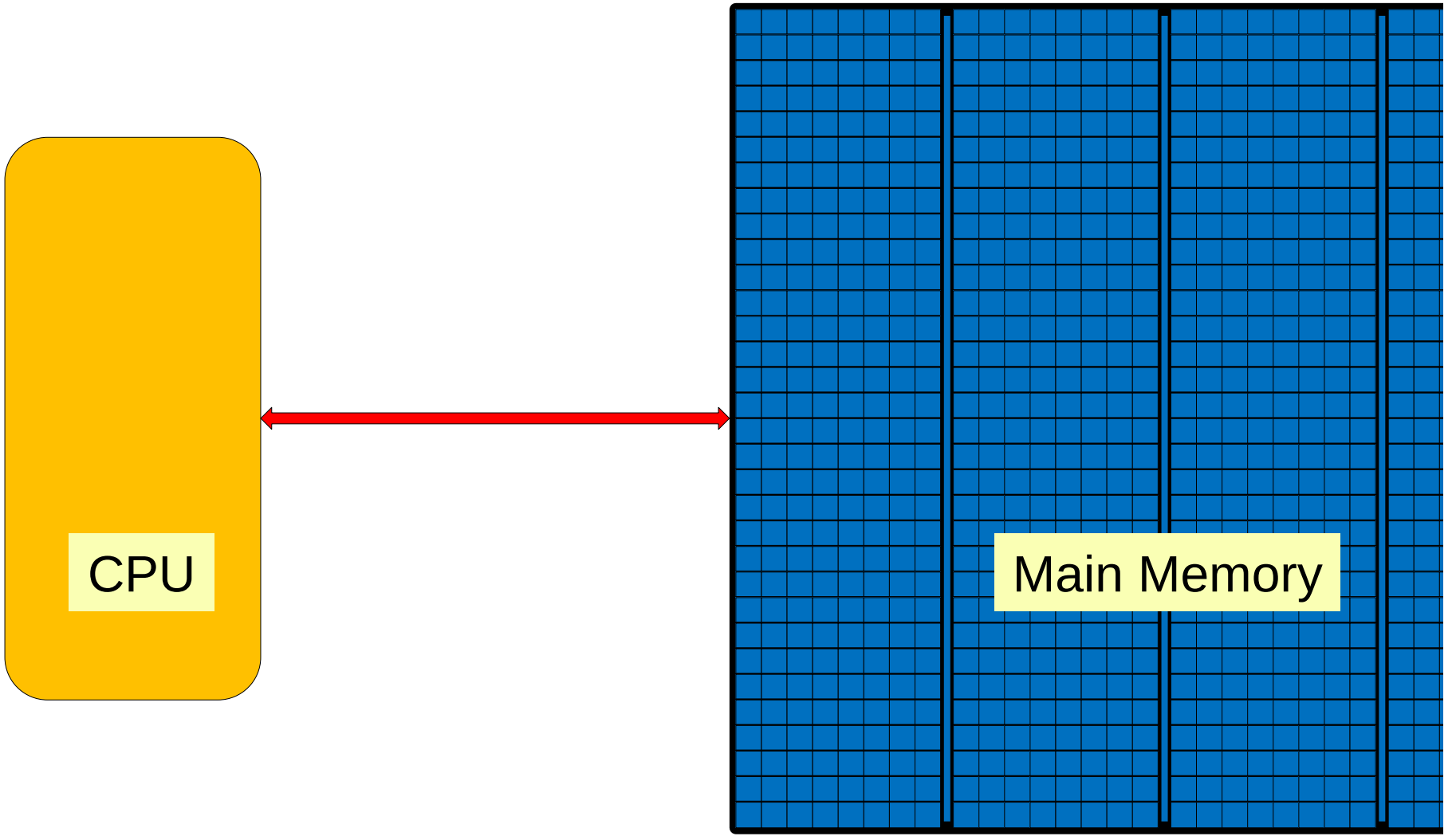
Review of Computer Memory



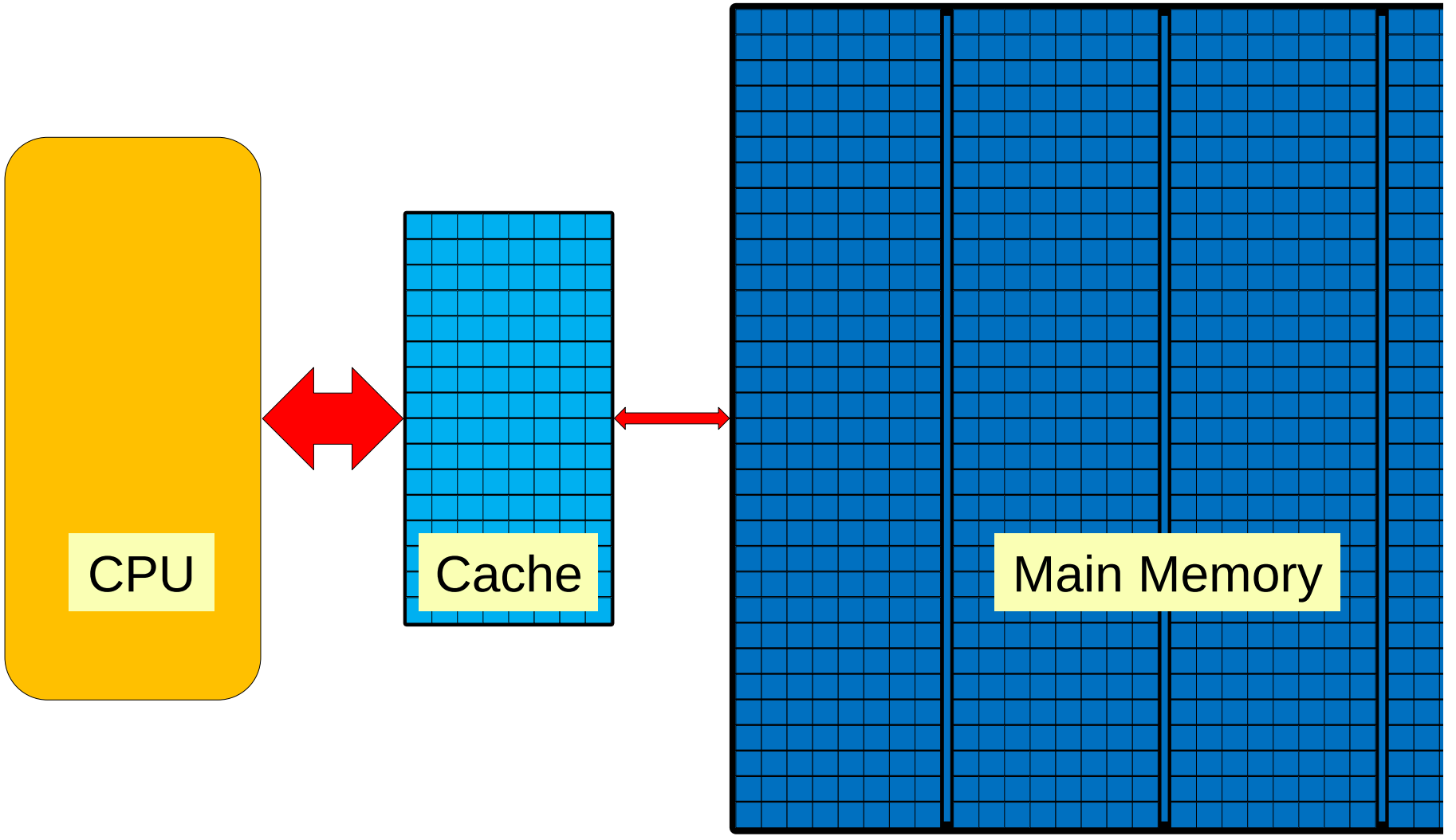
Review of Computer Memory

Cache Memory

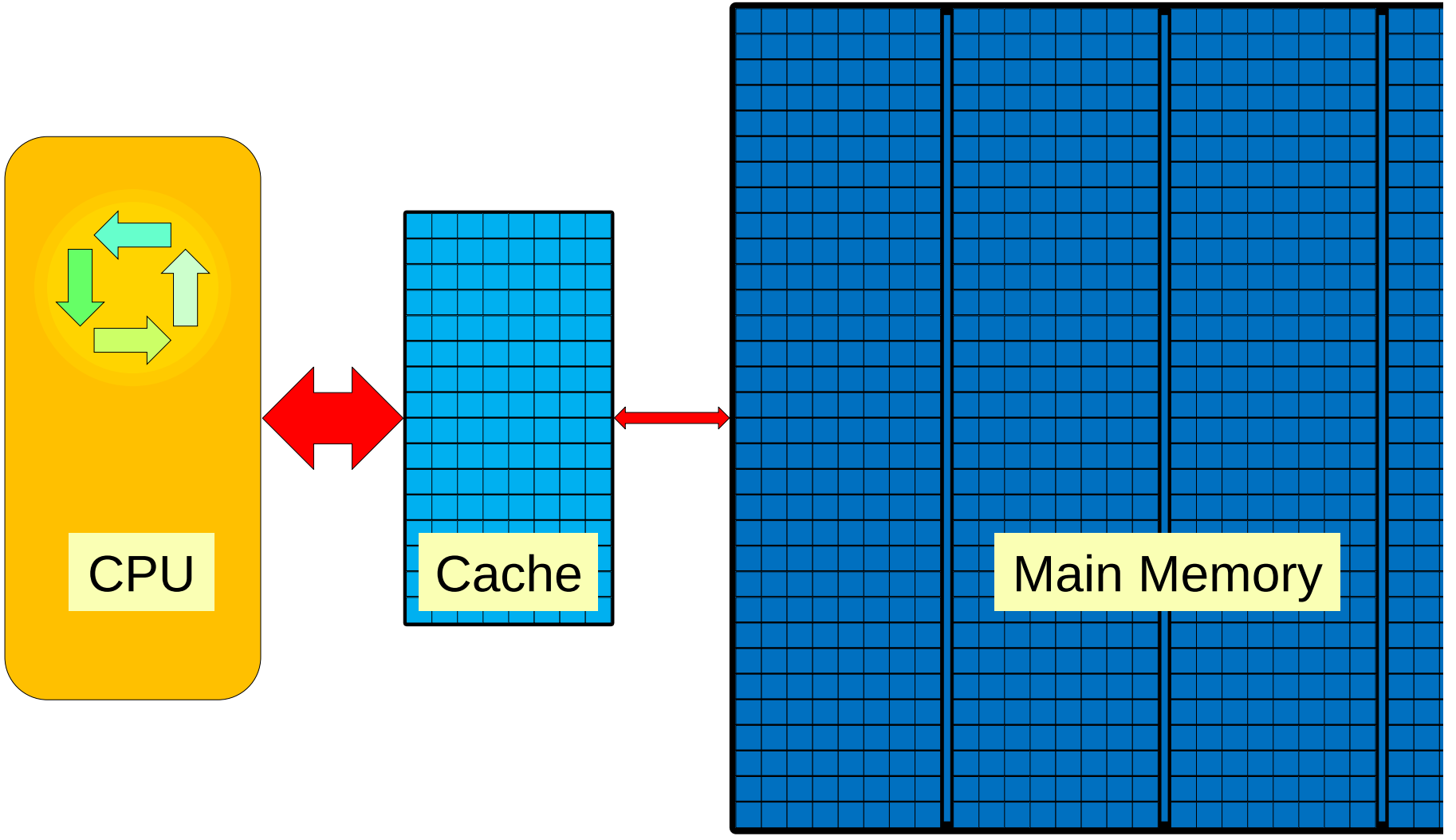
Review of Computer Memory



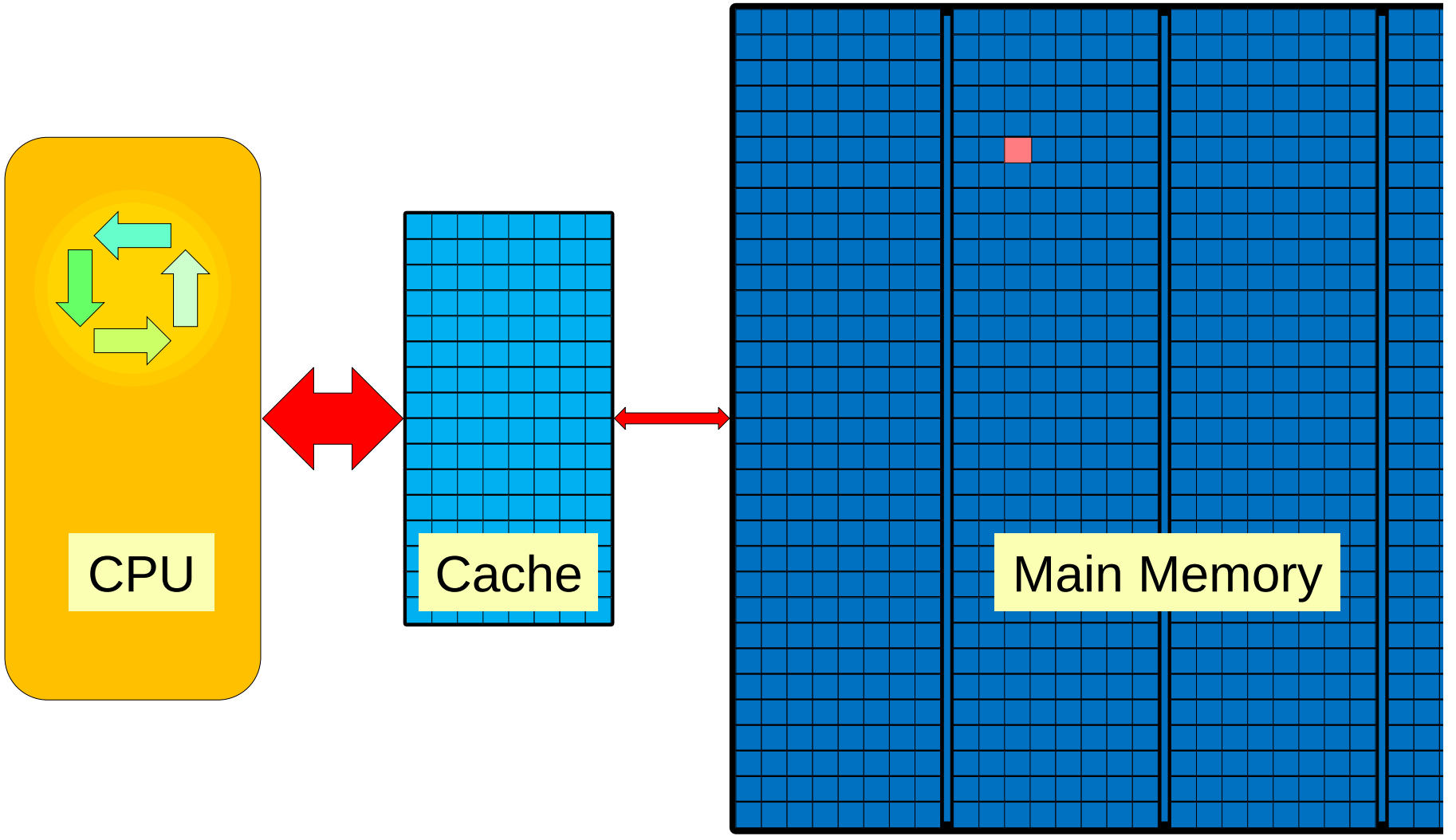
Review of Computer Memory



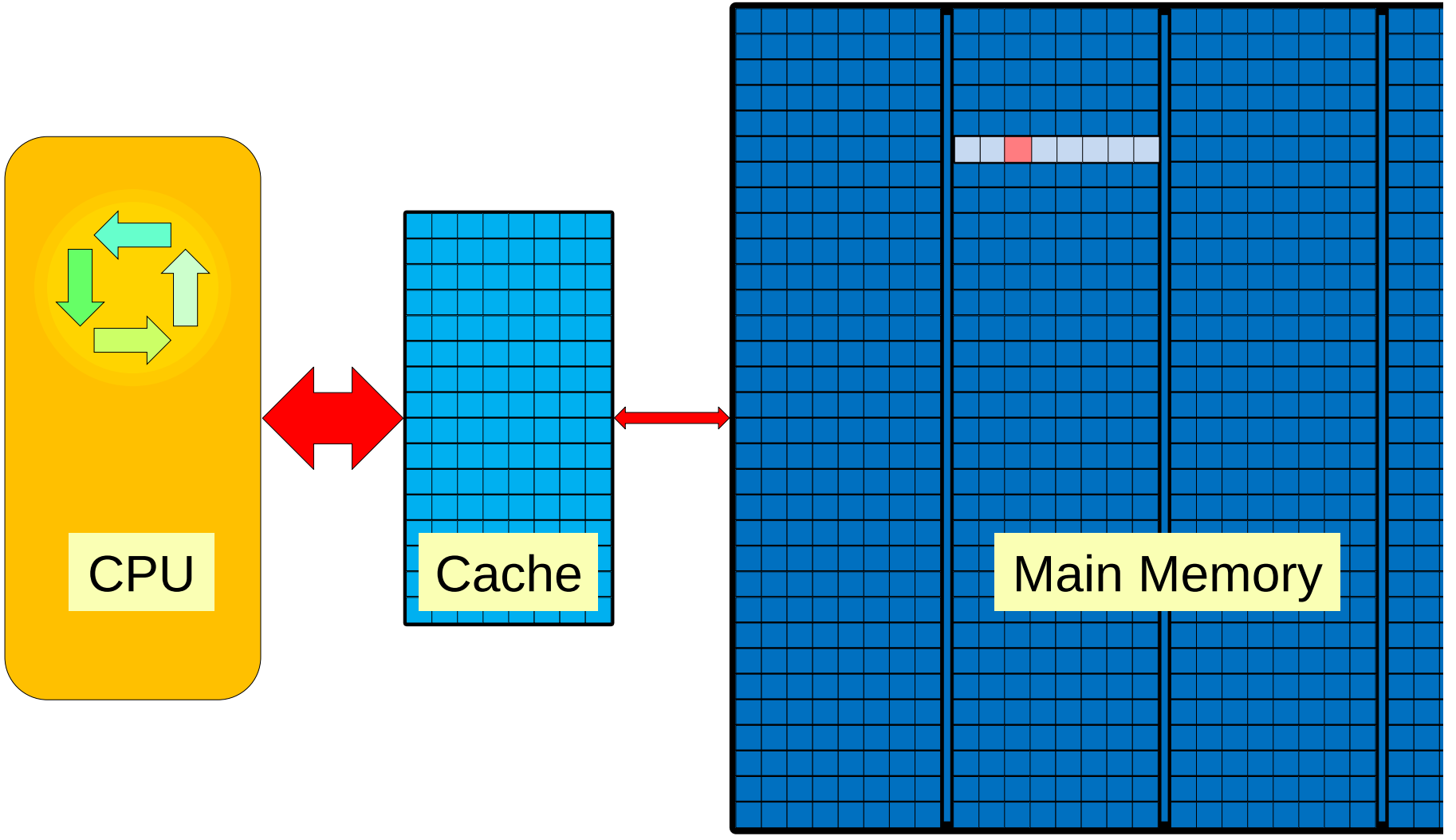
Review of Computer Memory



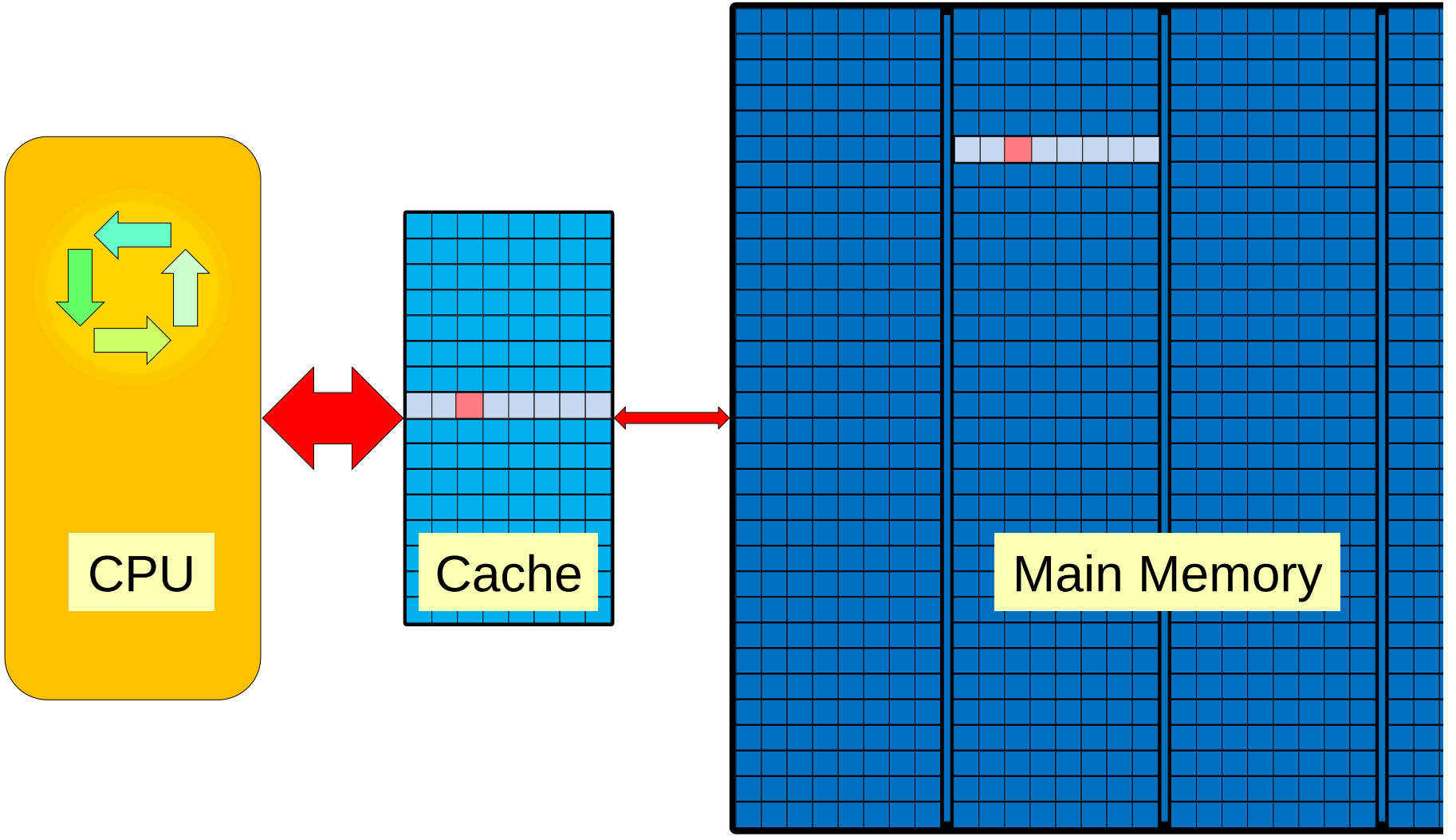
Review of Computer Memory



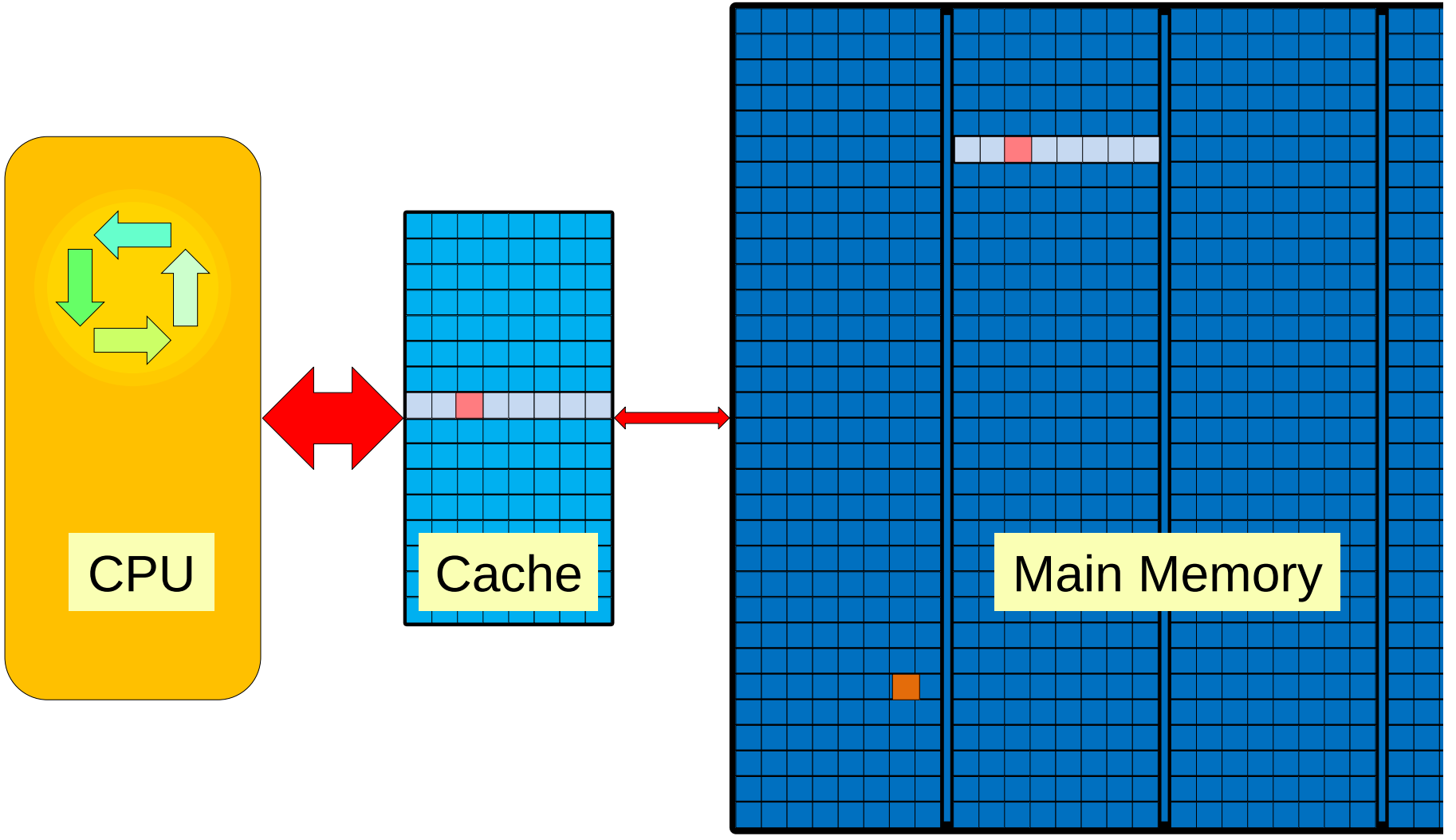
Review of Computer Memory



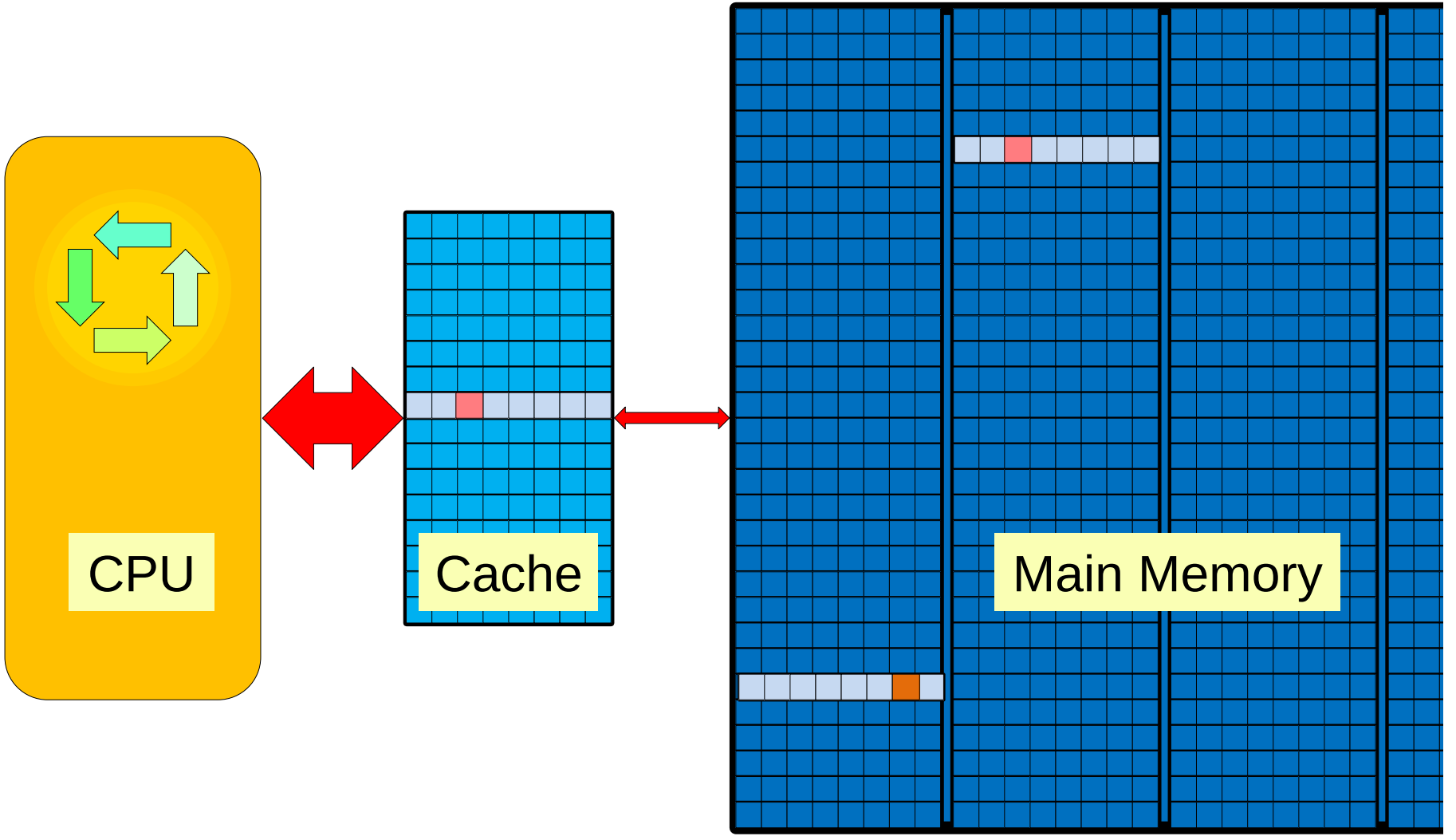
Review of Computer Memory



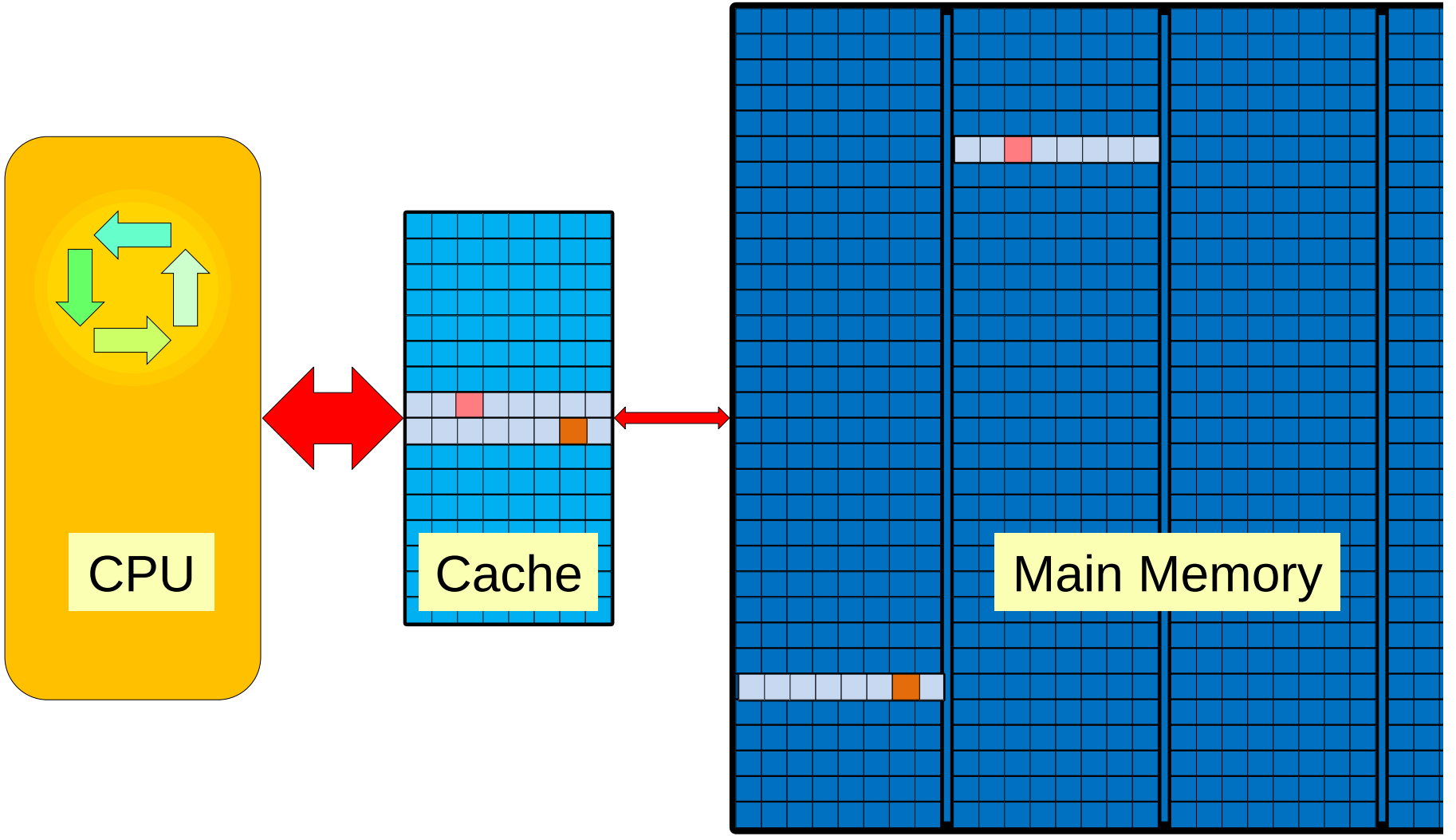
Review of Computer Memory



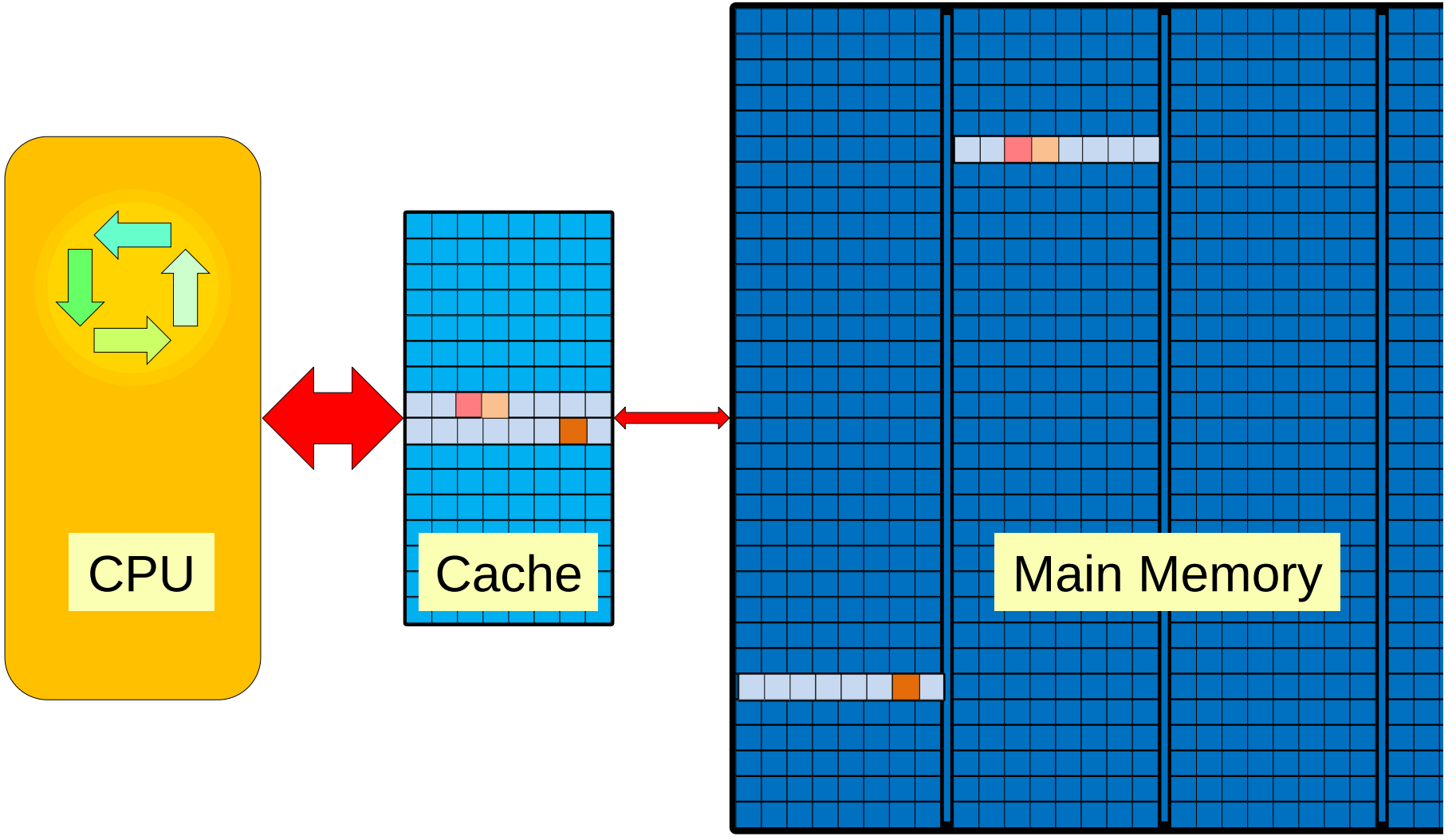
Review of Computer Memory



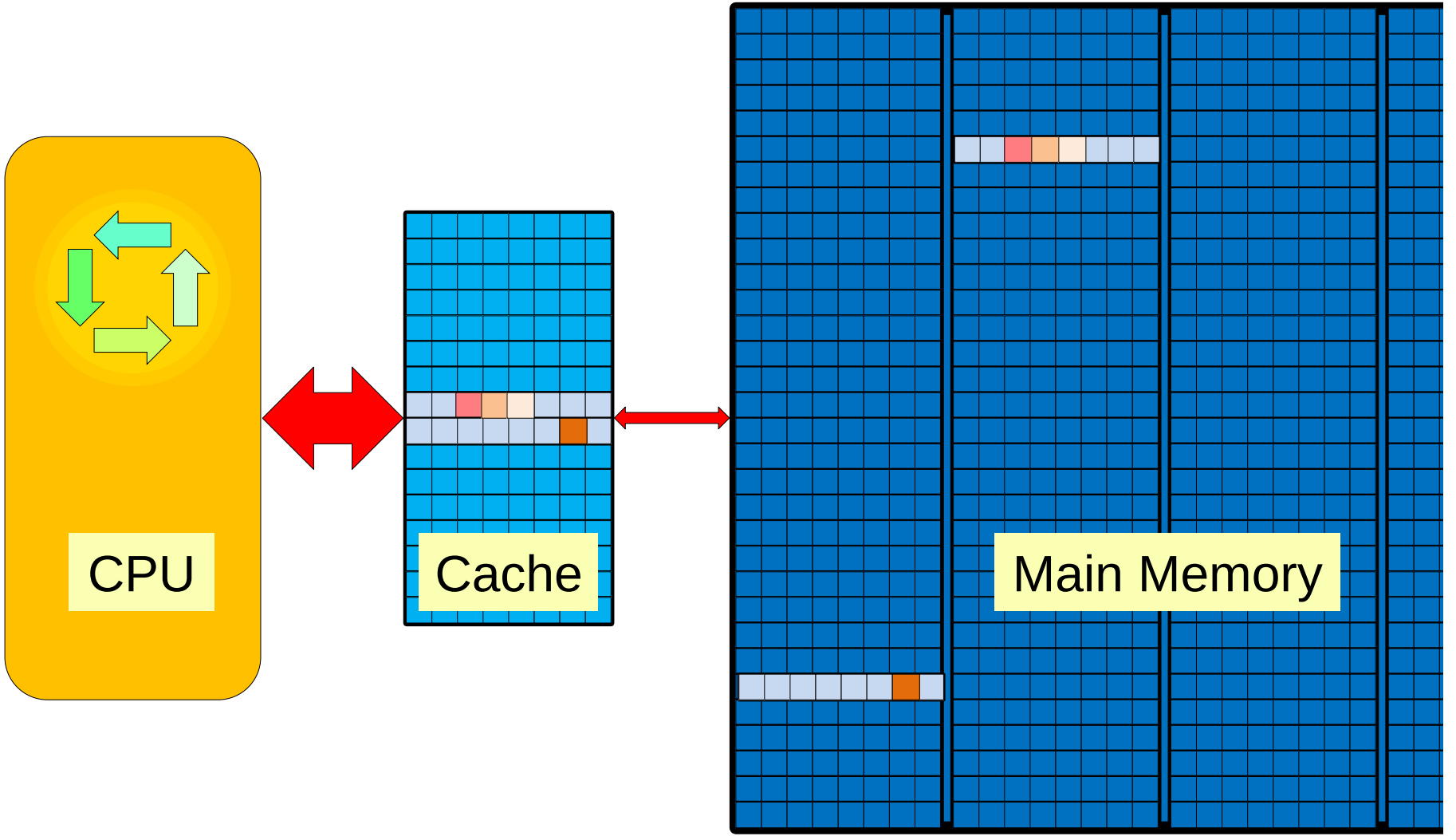
Review of Computer Memory



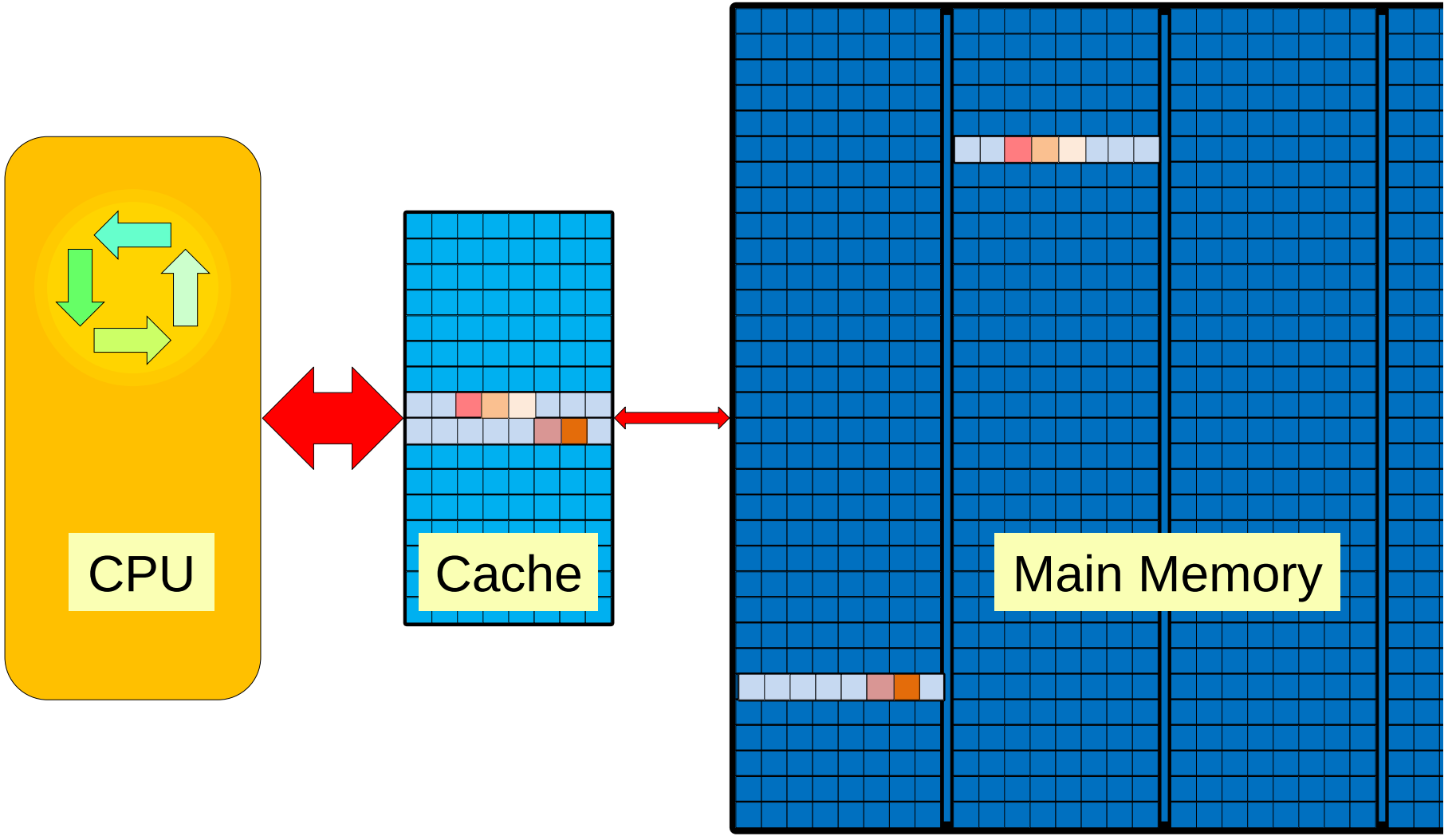
Review of Computer Memory



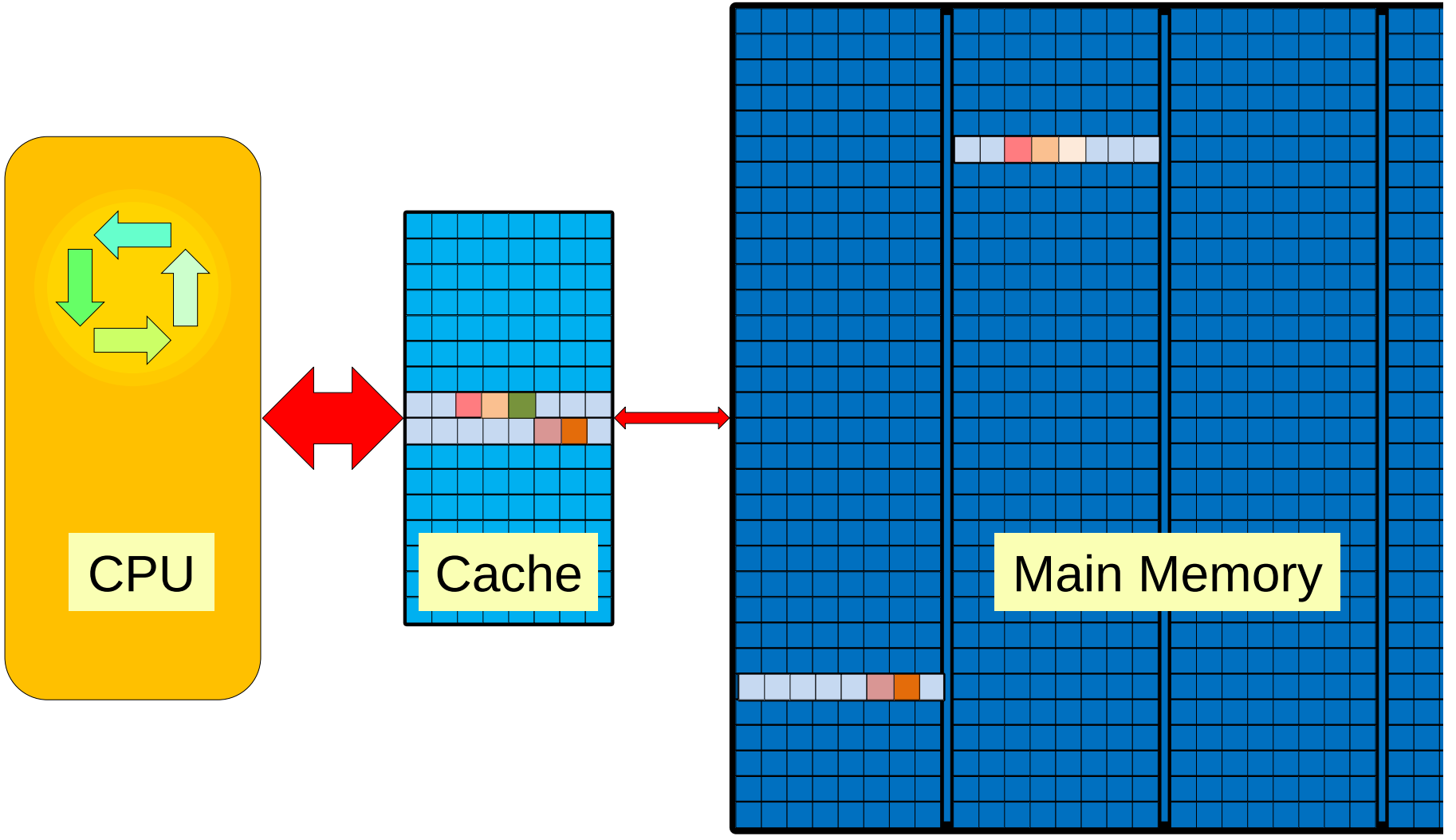
Review of Computer Memory



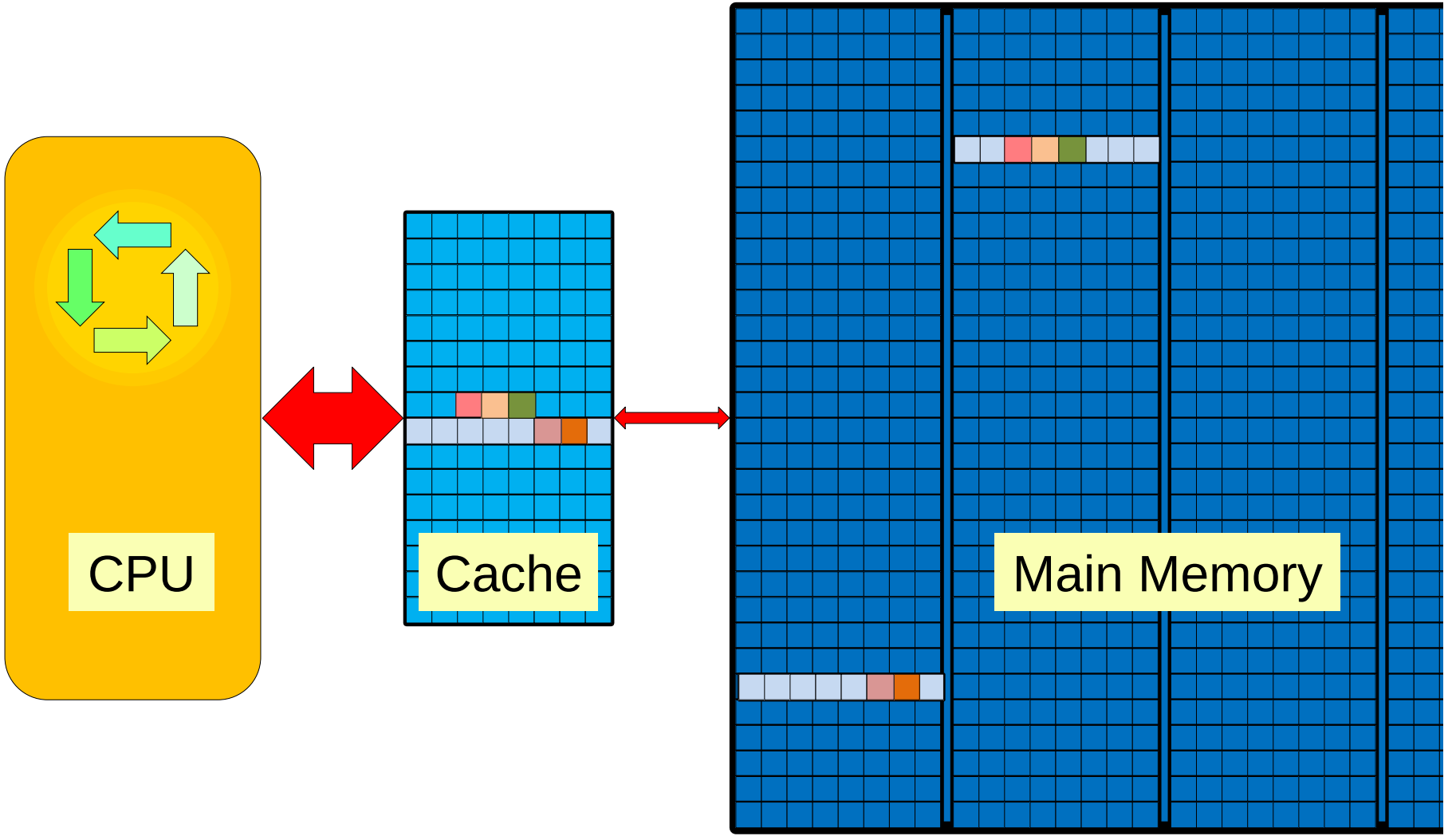
Review of Computer Memory



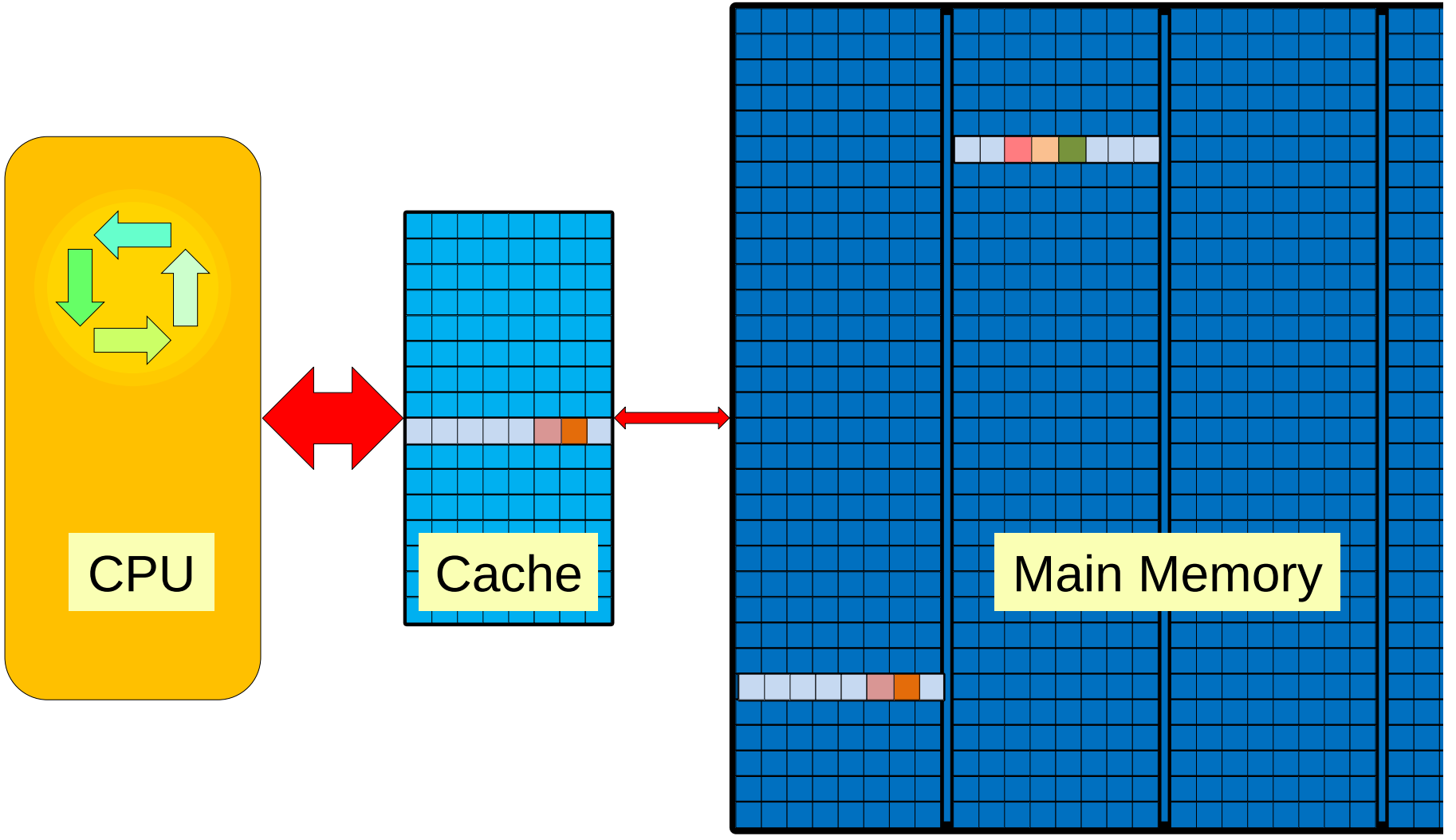
Review of Computer Memory



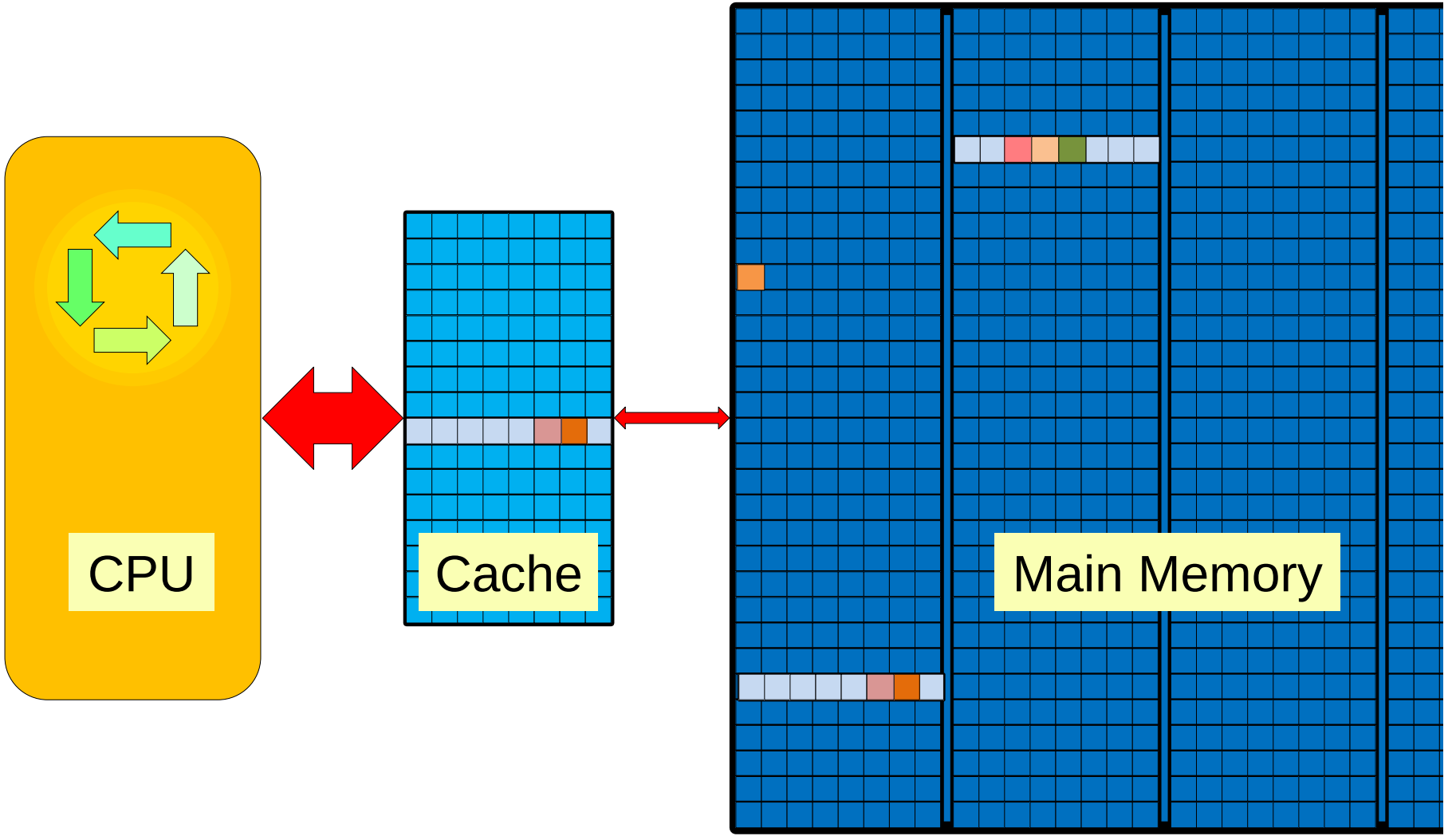
Review of Computer Memory



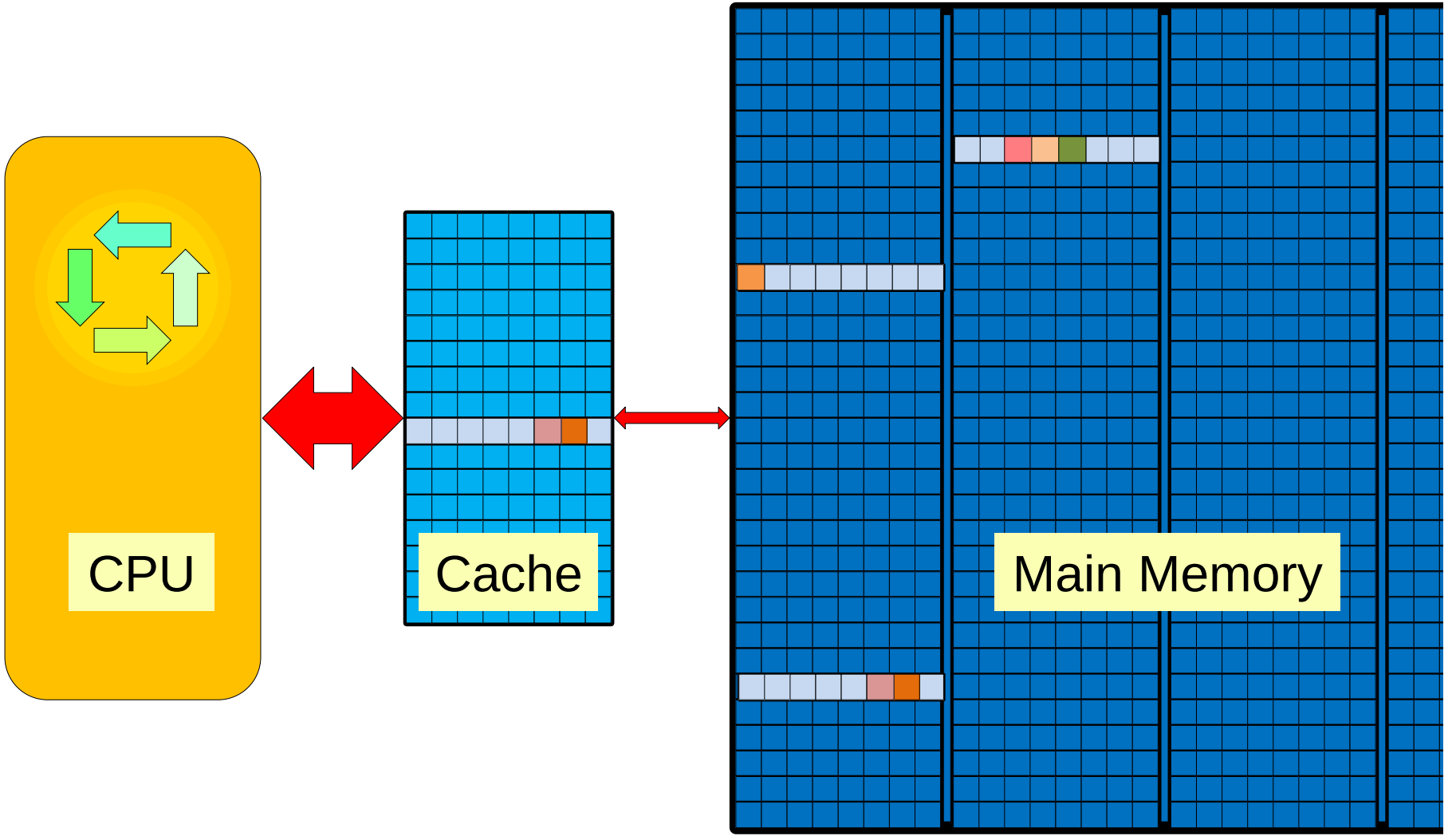
Review of Computer Memory



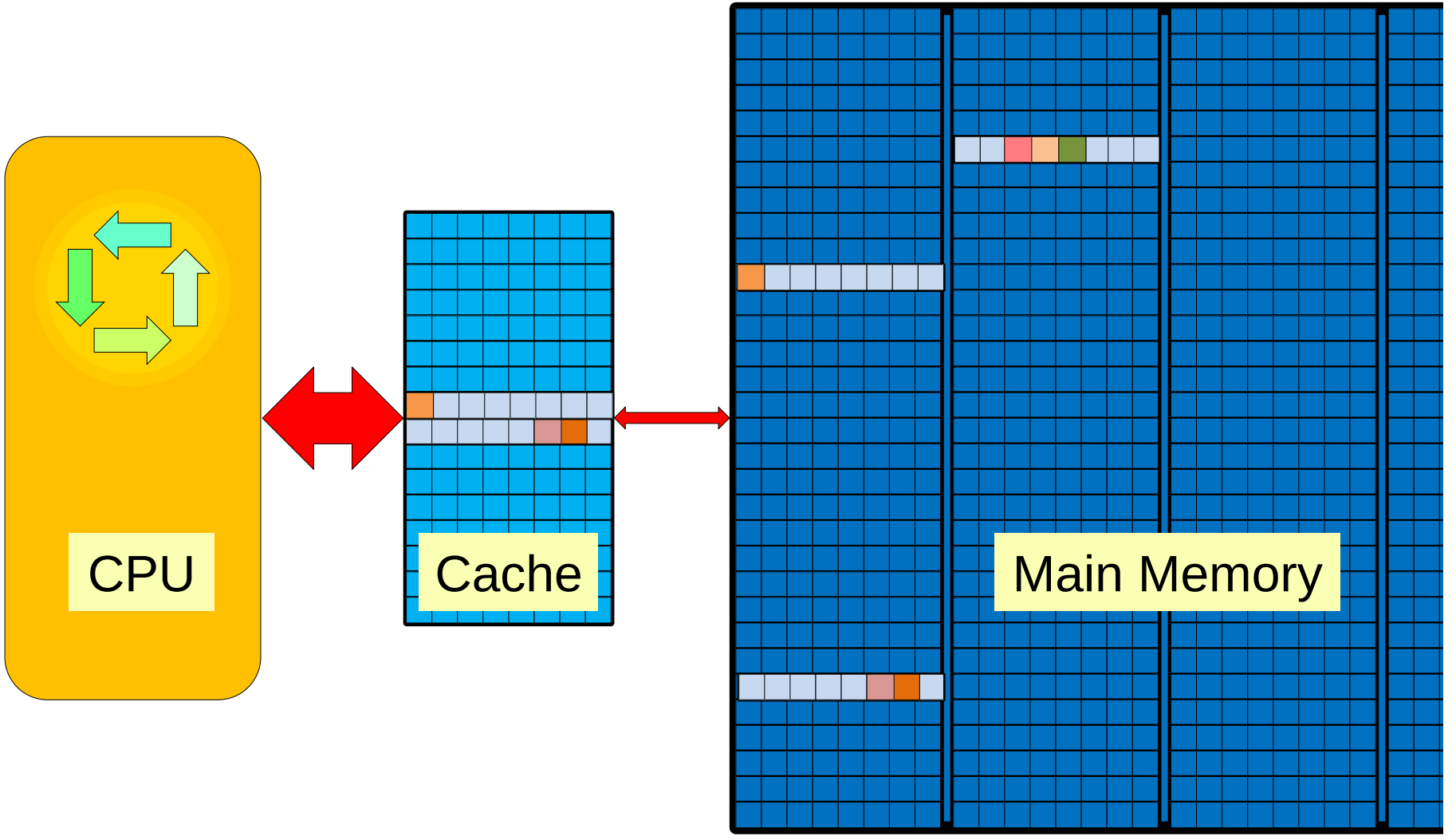
Review of Computer Memory



Review of Computer Memory



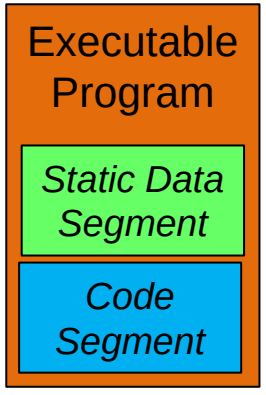
Review of Computer Memory

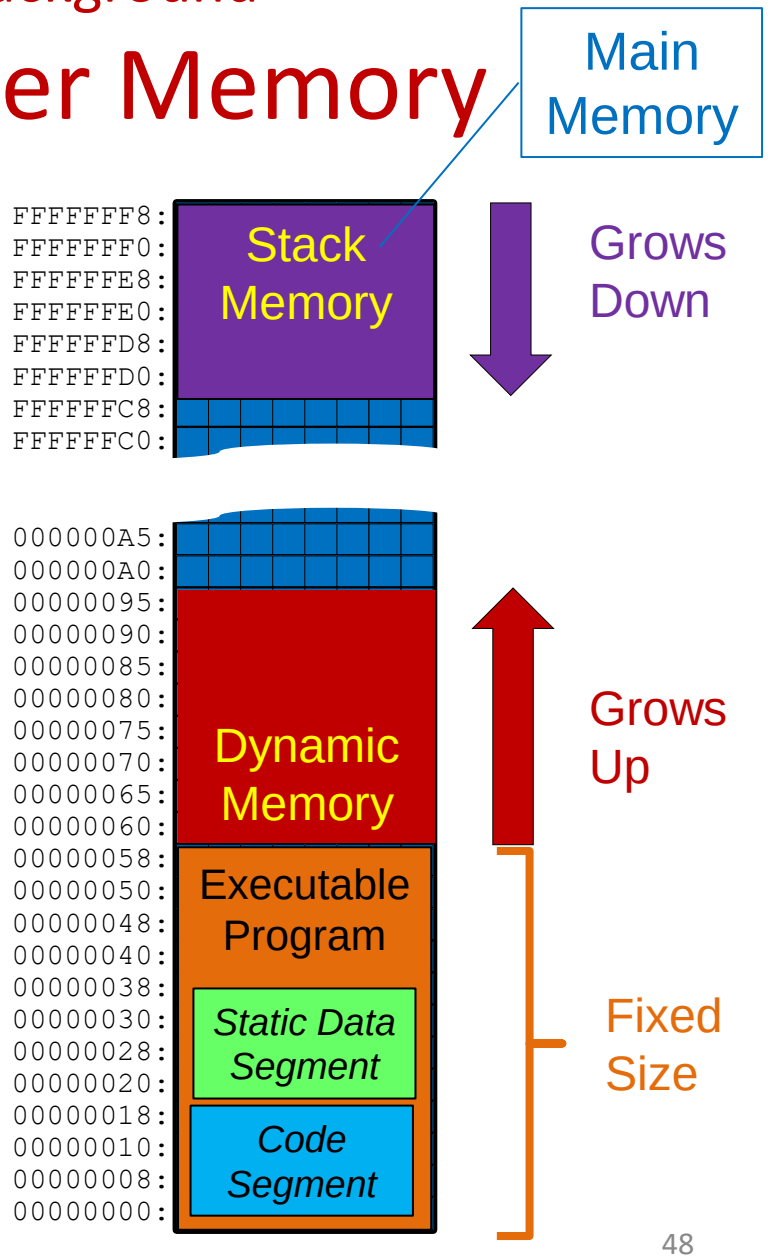


Review of Computer Memory

(Main) Memory Segments

Review of Computer Memory

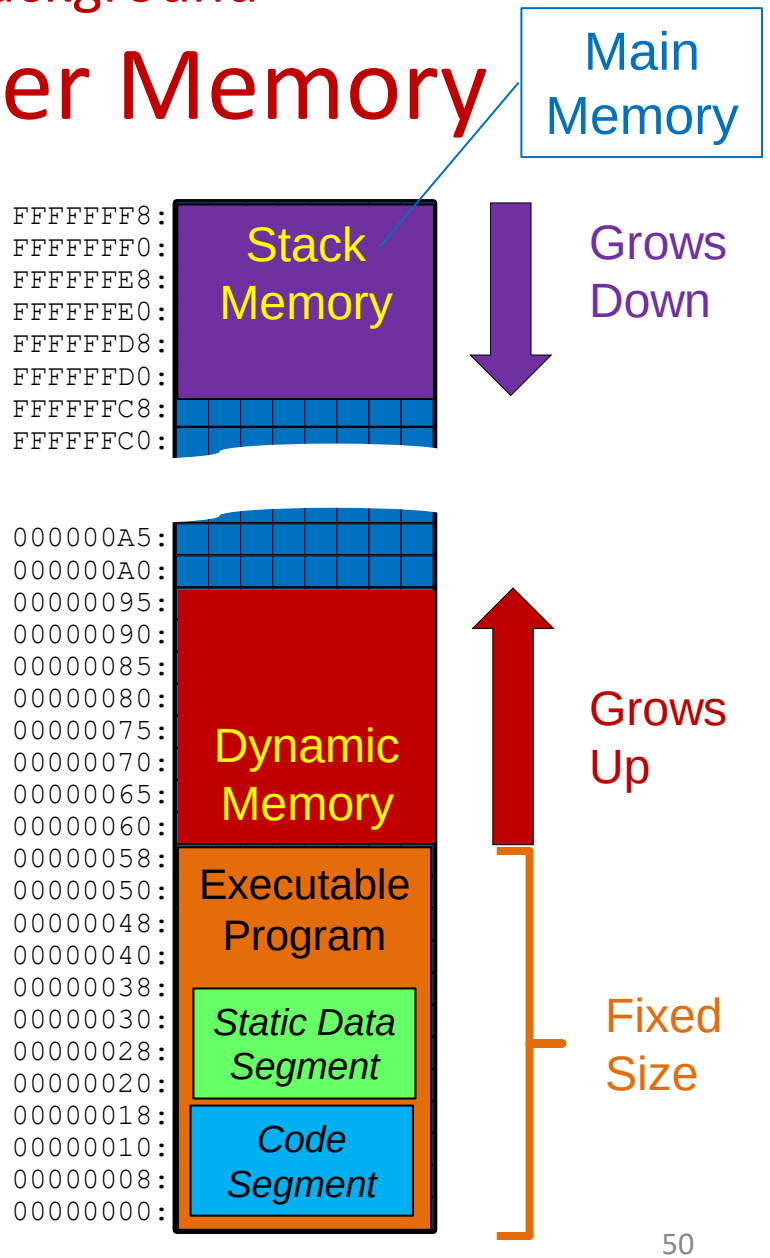




Important Questions

What is a *Memory Allocator*?

Review of Computer Memory



Review of Computer Memory

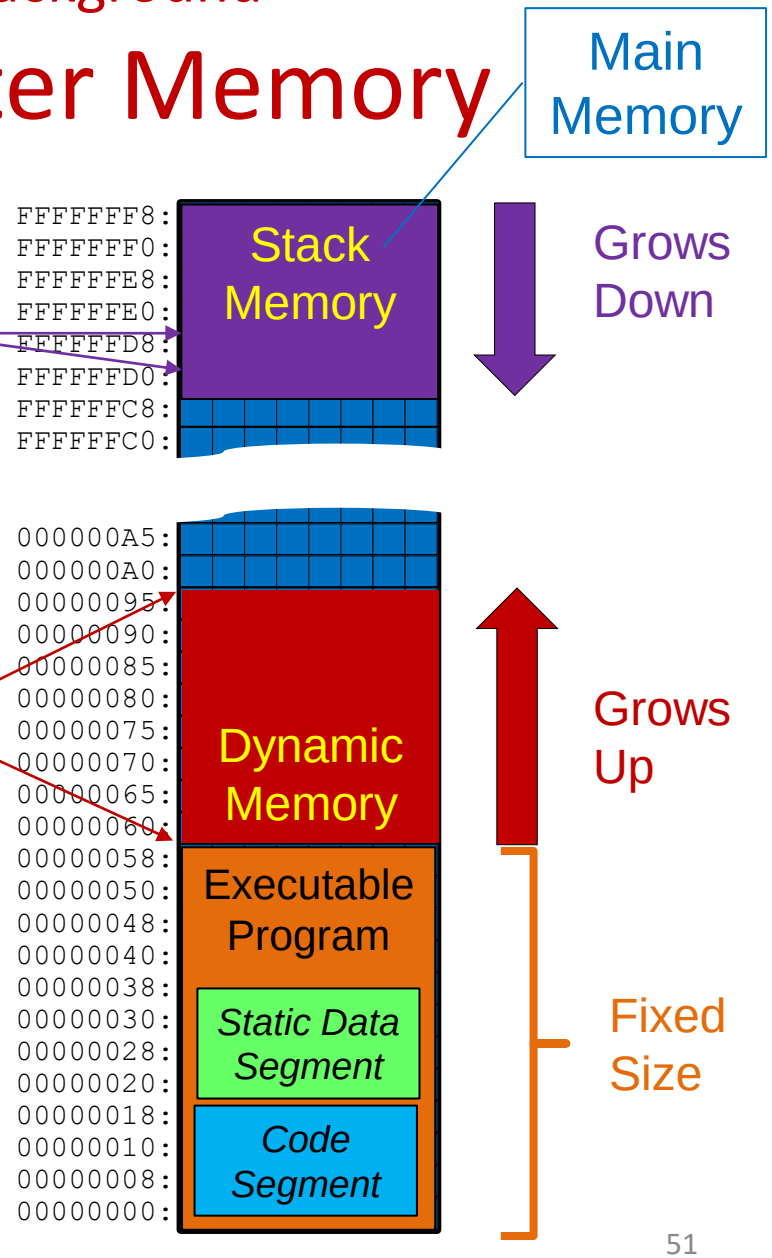
Special-Purpose Local Allocator:

```
// alloca.h
void *alloca(size_t nBytes);
```

General-Purpose Global Allocator:

```
// malloc.h
void *malloc(size_t nBytes);
void free(void *address);
```

C-Language Memory-Allocation Utilities



Memory Allocator Definition (take 1)

A *memory allocator* organizes a* region of computer memory, dispensing and reclaiming authorized access to suitable sub-regions on demand.

*possibly non-contiguous

General versus Special Allocator

A *General-Purpose Allocator*

- Is designed to work reasonably well for *all* use cases.
- Satisfies *all* requirements for memory allocators.

A *Special-Purpose Allocator*

- (Typically) works especially well for *some* use cases
- Need not satisfy *all* requirements for allocators, e.g.:
  - May not be safe to use in a multi-threaded program
  - May not reuse individually freed memory
- Requires specific knowledge of the context of use

Global versus Local Allocator

A *Global Allocator*

- Operates on a single ubiquitous region of memory
- Exists throughout the lifetime of a program
- Is inherently accessible from all parts of a program

A *Local Allocator*

- Operates on a local sub-region (“arena”) of memory
- May exist for less than the lifetime of a program
- Is (typically) supplied for client use via a *reference*
- Can (typically) be used to free memory unilaterally

Global, General Allocator Utility

C:

```
// <malloc.h>
void *malloc(size_t nbytes);
void free(void *address);
```

C++:

```
// <new>
namespace std {
void *operator new(size_t nbytes);
void operator delete(void *address);
…
```

General/Special × Global/Local

	Global	Local

General	`malloc`/`free`
	`new`/`delete`
	tcmalloc
	jemalloc
		`multipool_allocator`
		Any general algorithm applied to a physically (and temporally) local region of memory.

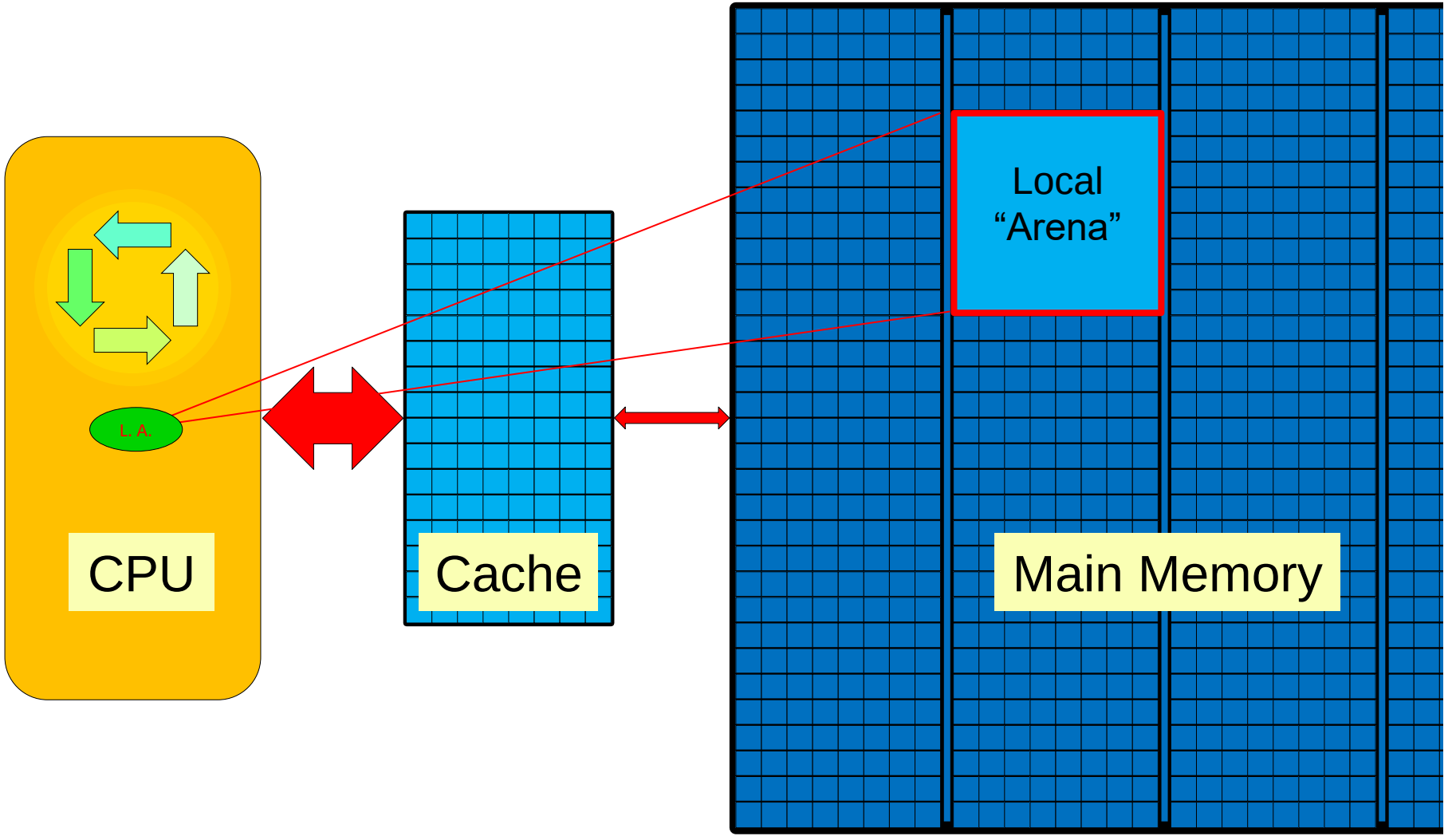
Special	An unsynchronized *tcmalloc* allocator "plugged into" (i.e., used to implement) `malloc`/`free`	`alloca`
		`monotonic_allocator`
		An unsynchronized version of a `multipool_allocator`

Memory Allocator Definition (take 2)

A *memory allocator* is a stateful ***utility*** or ***mechanism*** that organizes a* region of computer memory, dispensing and reclaiming authorized access to suitable sub-regions on demand.

*possibly non-contiguous

What is a local memory allocator?



Local Allocator Mechanism

```
class LocalAllocator {
    // internal data structure
  public:
    LocalAllocator(const LocalAllocator&) = delete;
    LocalAllocator& operator=(const LocalAllocator&)
                                          = delete;
    // CREATORS
    LocalAllocator(/* … */);
    // MANIPULATORS
    void *allocate(std::size_t nBytes);
    void deallocate(void *address);
};
```

Local Allocator Mechanism

```
class LocalAllocator {
    // internal data structure
  public:
    LocalAllocator(const LocalAllocator&) = delete;
    LocalAllocator& operator=(const LocalAllocator&)
                                          = delete;
    // CREATORS
    LocalAllocator(/* … */);
    // MANIPULATORS
    void *allocate(std::size_t nBytes);
    void deallocate(void *address);
};
```

Local Allocator Mechanism

```
class LocalAllocator {
    // internal data structure
  public:
    LocalAllocator(const LocalAllocator&) = delete;
    LocalAllocator& operator=(const LocalAllocator&)
                                          = delete;
    // CREATORS
    LocalAllocator(void *begin, void *end);
    // MANIPULATORS
    void *allocate(std::size_t nBytes);
    void deallocate(void *address);
};
```

Local Allocator Mechanism

```
class LocalAllocator {
    // internal data structure
  public:
    LocalAllocator(const LocalAllocator&) = delete;
    LocalAllocator& operator=(const LocalAllocator&)
                                          = delete;
    // CREATORS
    LocalAllocator(void *begin, void *end);
    // MANIPULATORS
    void *allocate(std::size_t nBytes);
    void deallocate(void *address);
};
```

Local Allocator Mechanism

```
class LocalAllocator {
    // internal data structure
  public:
    LocalAllocator(const LocalAllocator&) = delete;
    LocalAllocator& operator=(const LocalAllocator&)
                                        = delete;
    // CREATORS
    LocalAllocator(void *begin, void *end);
    // MANIPULATORS
    void *allocate(std::size_t nBytes);
    void deallocate(void *address);
    void release(); // local allocators only
};
```

Memory Allocator Definition (take 3)

A *memory allocator* is (the client-facing interface for) a stateful *utility* or *mechanism* that organizes a* region of computer memory, dispensing and reclaiming authorized access to suitable sub-regions on demand.

*possibly non-contiguous

Memory Allocator Interfaces

Allocators can be supplied for use in multiple ways:

1. As (stateful) utility functions.
   - ❖ Doesn't support allocator objects.
2. As a "reference wrapper" template parameter.
   - ✓ Concrete allocator type is available for use by client's compiler.
   - ❖ Forces a client to be a template to hold the allocator reference.
   - ❖ Allocator type affects the C++ type of the client object.
3. As the address of a pure abstract base class.
   - ✓ Allocator can be held via a base-class reference by a non-template class.
   - ✓ The choice of allocator does not affect the C++ type of the client object.
   - ❖ Allocator must be accessed via its virtual-function interface.
   - ❖ Object must somehow hold an extra address, even for the default case.

Outline

1. Introduction and Background
   - What are memory allocators, and why are they useful?
2. Understanding the Problem
   - What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   - When and how do you use which allocator, and why?
4. Conclusions
   - What must we remember about memory allocators?

Outline

1. Introduction and Background

   What are memory allocators, and why are they useful?

2. Understanding the Problem

   What aspects of software affect allocation strategy?

3. Analyzing the Benchmark Data

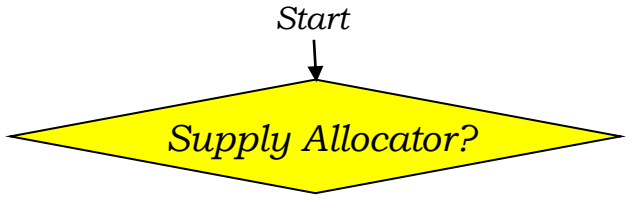
   When and how do you use which allocator, and why?

4. Conclusions

   What must we remember about memory allocators?

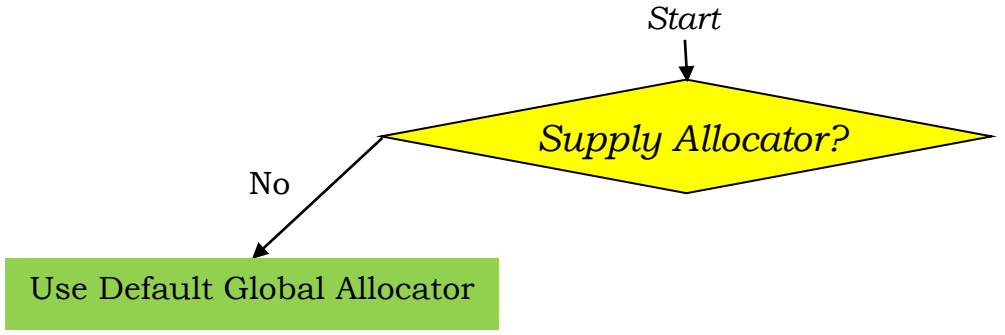
2. Understanding the Problem

Supply a local allocator? Which one?

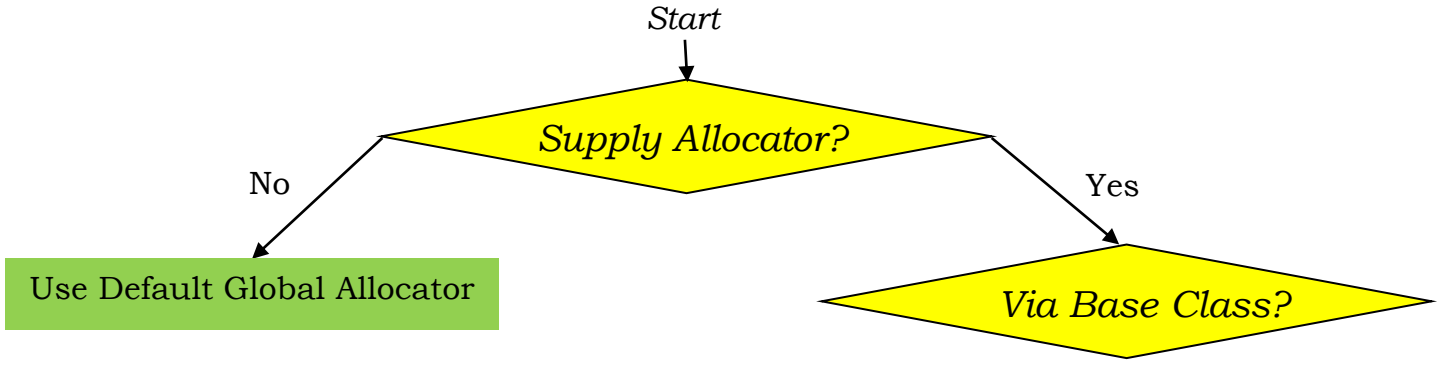


2. Understanding the Problem

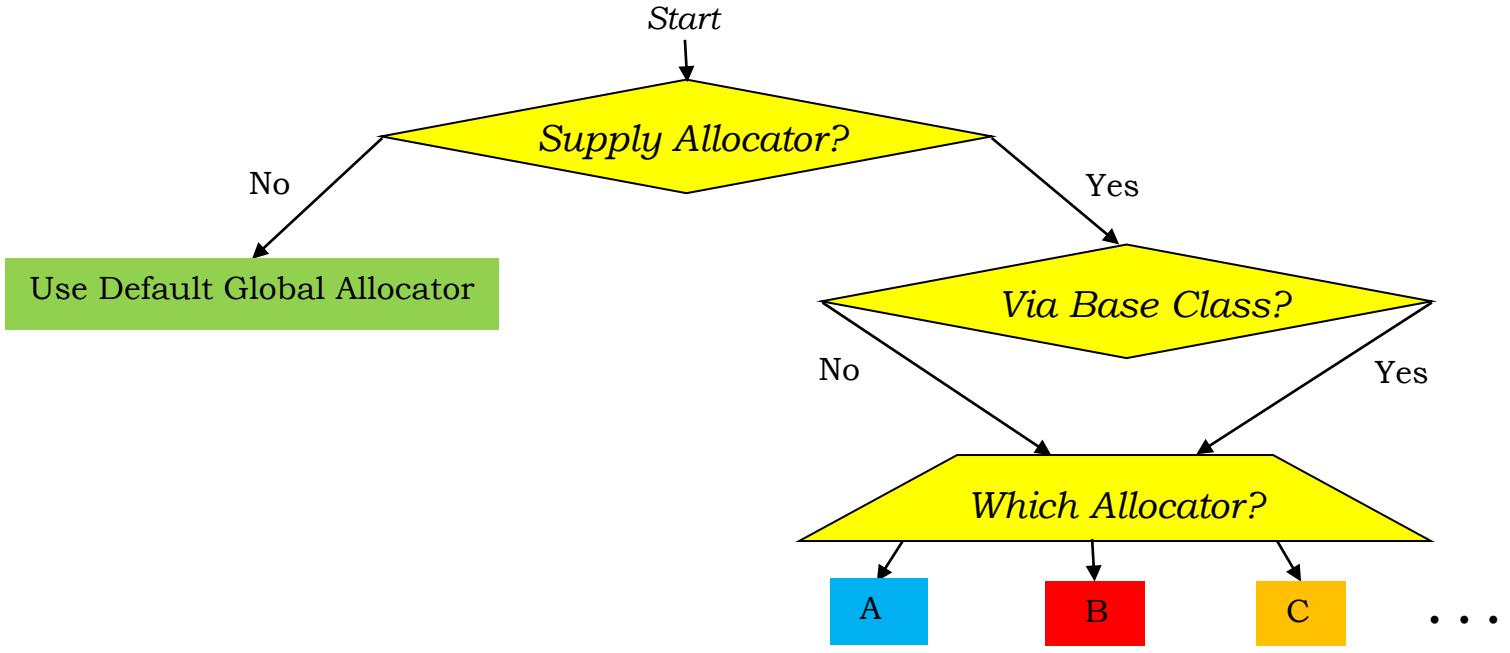
Supply a local allocator? Which one?



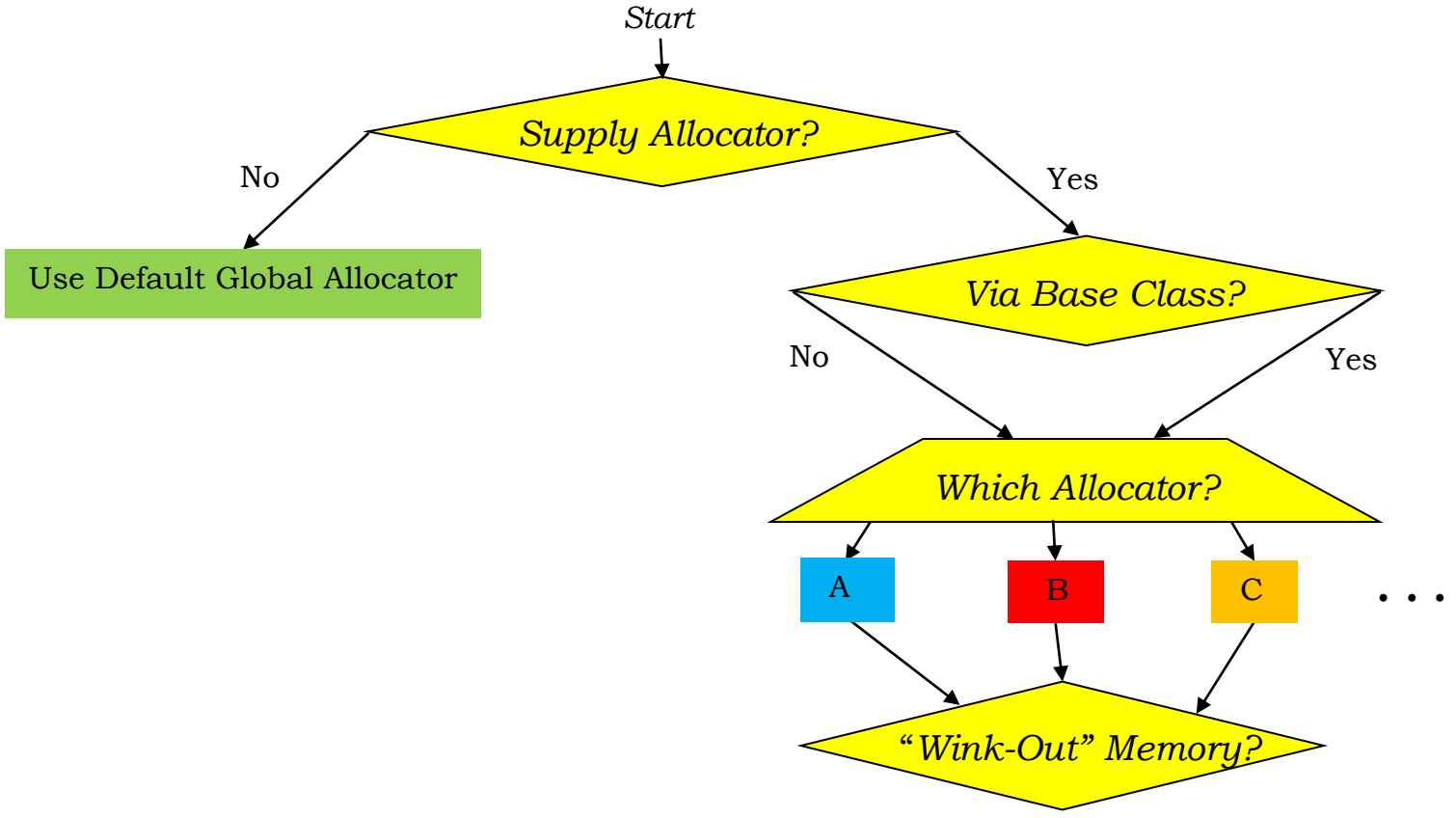
Supply a local allocator? Which one?



Supply a local allocator? Which one?

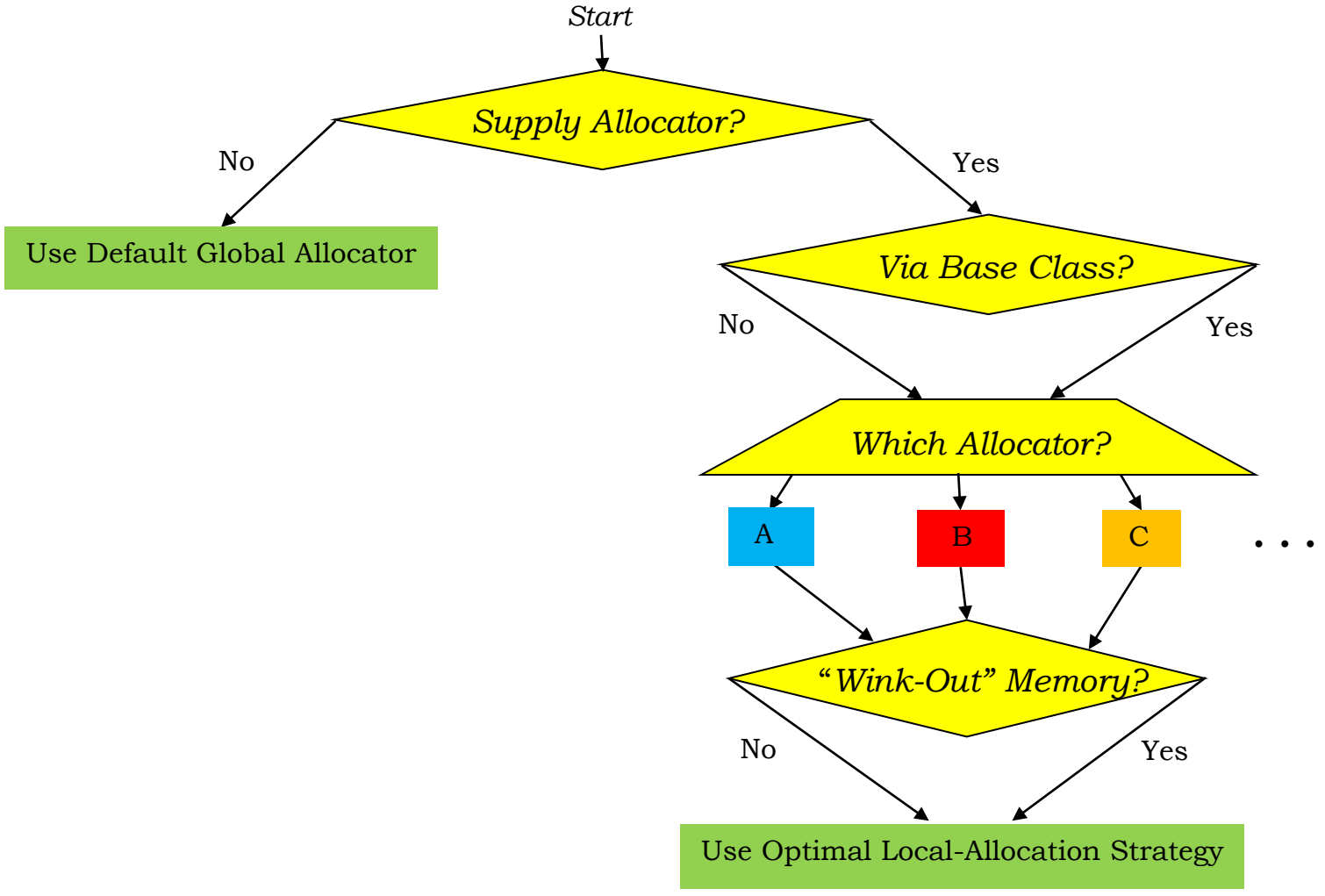


Supply a local allocator? Which one?



2. Understanding the Problem

Supply a local allocator? Which one?



Our Tool Chest of Allocation Strategies

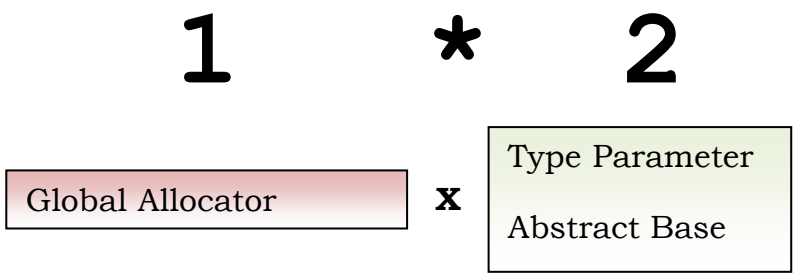
Our Tool Chest of Allocation Strategies

1

Global Allocator

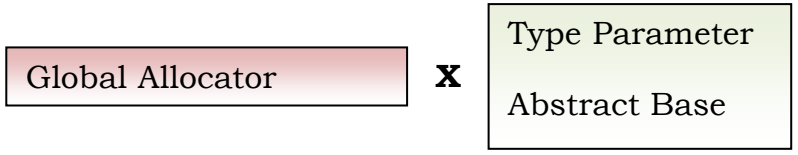
2. Understanding the Problem

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies

2 = 1 * 2



Our Tool Chest of Allocation Strategies

2 = 1 * 2

Global Allocator x Type Parameter / Abstract Base

3

Monotonic

Multipool

Multipool<Monotonic>

Our Tool Chest of Allocation Strategies

2 = 1 * 2

Global Allocator x Type Parameter / Abstract Base

3 * 2

Monotonic / Multipool / Multipool<Monotonic> x Type Parameter / Abstract Base

Our Tool Chest of Allocation Strategies

2 = 1 * 2

Global Allocator x Type Parameter / Abstract Base

3 * 2 * 2

Monotonic / Multipool / Multipool<Monotonic> x Type Parameter / Abstract Base x Normal Destruction / (Magically) “Winked Out”

Our Tool Chest of Allocation Strategies

2 = 1 * 2

Global Allocator x Type Parameter / Abstract Base

12 = 3 * 2 * 2

Monotonic / Multipool / Multipool<Monotonic> x Type Parameter / Abstract Base x Normal Destruction / (Magically) “Winked Out”

Our Tool Chest of Allocation Strategies

2 = 1 * 2

Global Allocator x Type Parameter Abstract Base

12 = 3 * 2 * 2

Monotonic Multipool Multipool<Monotonic> x Type Parameter Abstract Base x Normal Destruction (Magically) "Winked Out"

14 **Allocation Strategies (AS)**

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

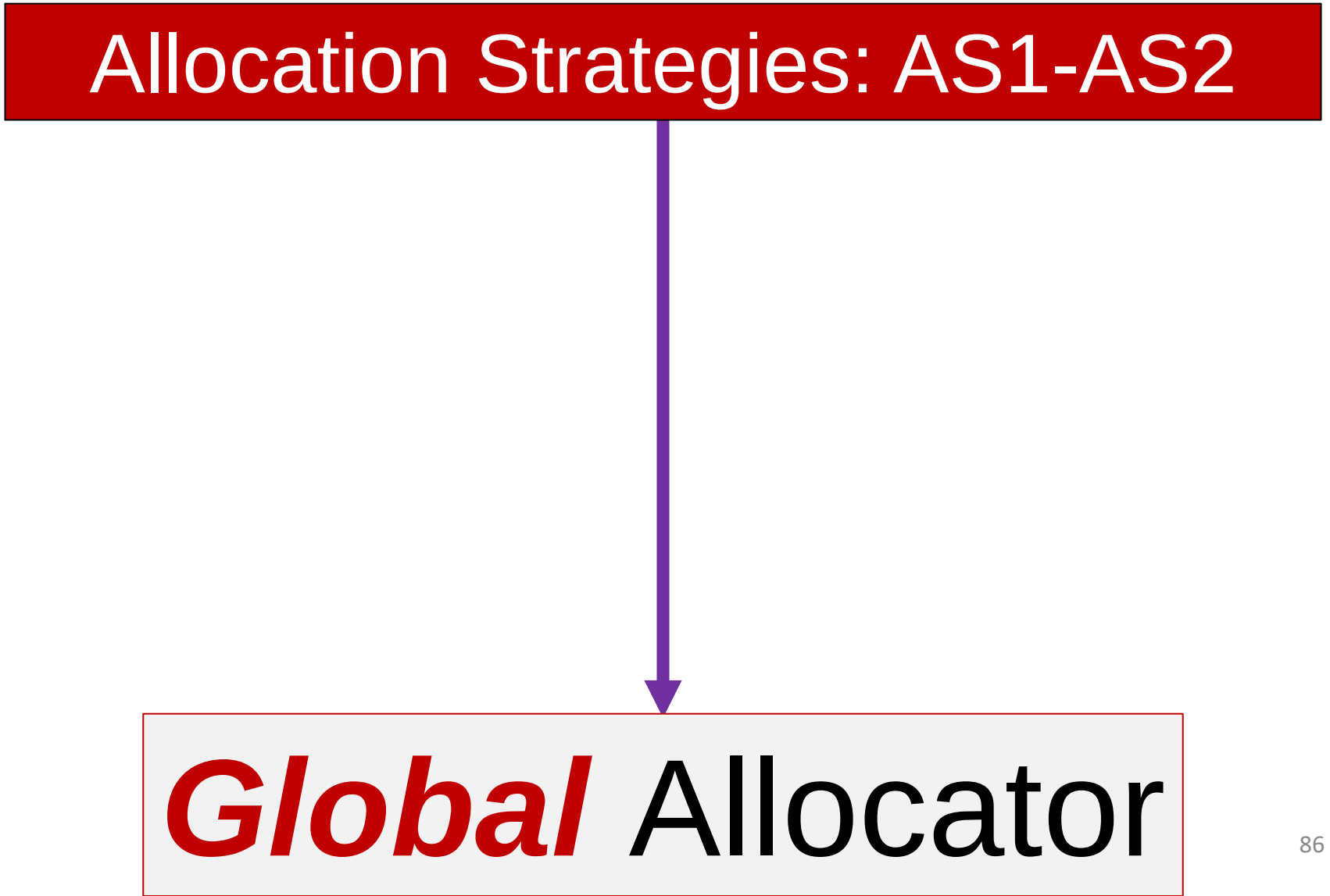
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies

Standard Allocator

```
class allocator {
    // no data members
  public:
    // CREATORS
    allocator() { }
    allocator(const allocator& ) { }
    ~allocator() { }

    // MANIPULATORS
    allocator& operator=() = delete;
    void *allocate(std::size_t nBytes) {
                          return ::operator new(nBytes); }
    void deallocate(void *address) {
                            ::operator delete(address); }
};

// FREE OPERATORS
bool operator==(const allocator&, const allocator&) {
                                                return true; }
```

Our Tool Chest of Allocation Strategies

Standard Allocator

```
class allocator {
    // no data members
  public:
    // CREATORS
    allocator() { }
    allocator(const allocator& ) { }
    ~allocator() { }

    // MANIPULATORS
    allocator& operator=() = delete;
    void *allocate(std::size_t nBytes) {
                          return ::operator new(nBytes); }
    void deallocate(void *address) {
                            ::operator delete(address); }
};

// FREE OPERATORS
bool operator==(const allocator&, const allocator&) {
                                                return true; }
```

Our Tool Chest of Allocation Strategies

AS1: Standard Allocator
Default Global Allocator

```
class allocator {
    // no data members
  public:
    // CREATORS
    allocator() { }
    allocator(const allocator& ) { }
    ~allocator() { }

    // MANIPULATORS
    allocator& operator=() = delete;
    void *allocate(std::size_t nBytes) {
                          return ::operator new(nBytes); }
    void deallocate(void *address) {
                            ::operator delete(address); }
};

// FREE OPERATORS
bool operator==(const allocator&, const allocator&) {
                                                return true; }
```

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myFunction()
{
   std::vector<int> v;
}
```

Same object code generated as

```
myFunction()
{
   std::vector<int, allocator> v;
}
```

Our Tool Chest of Allocation Strategies

```
myFunction()
{
   std::vector<int> v;
}
```

Same object code generated as

```
myFunction()
{
   std::vector<int, allocator> v;
}
```

Type Parameter

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);









} // 'system' goes out of scope (and is destroyed).
```

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);









}  // 'system' goes out of scope (and is destroyed).
```

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);

    for (int i = 0; i < N; ++i) {
        system[i] = new std::list<int>;
        // build up list of elements
    }




}  // 'system' goes out of scope (and is destroyed).
```

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);

    for (int i = 0; i < N; ++i) {
        system[i] = new std::list<int>;
        // build up list of elements
    }
    // Do benchmark (e.g., access links).




}  // 'system' goes out of scope (and is destroyed).
```

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);

    for (int i = 0; i < N; ++i) {
        system[i] = new std::list<int>;
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (int i = 0; i < N; ++i) {
        delete system[i];
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);

    for (int i = 0; i < N; ++i) {
        system[i] = new std::list<int>;
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::list<int> *> system(N);

    for (auto& ptr : system) {
        ptr = new std::list<int>;
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Protocol:
Pure Abstract Base Class

```
class Allocator {
    // no data members
  public:
    // CREATORS
    virtual ~allocator();  // Defined empty in '.cpp' file.

    // MANIPULATORS
    virtual void *allocate(std::size_t nBytes) = 0;
    virtual void deallocate(void *address) = 0;
};
```

Our Tool Chest of Allocation Strategies

Protocol:
Pure Abstract Base Class

```
class Allocator {
    // no data members
  public:
    // CREATORS
    virtual ~allocator();  // Defined empty in '.cpp' file.

    // MANIPULATORS
    virtual void *allocate(std::size_t nBytes) = 0;
I   virtual void deallocate(void *address) = 0;
};
```

Our Tool Chest of Allocation Strategies

```
class NewDeleteAllocator : public Allocator {
    // no data members
  public:
    // CREATORS
    NewDeleteAllocator() = default;
    ~NewDeleteAllocator() = default;
    NewDeleteAllocator(const NewDeleteAllocator&) = delete;

    // MANIPULATORS
    void operator=(const NewDeleteAllocator&) = delete;

    inline void *allocate(std::size_t nBytes) override {
                                return ::operator new(nBytes); }

    inline void deallocate(void *address) override {
                                  ::operator delete(address); }
};
```

Concrete Derived Class

Our Tool Chest of Allocation Strategies

AS2

Concrete Derived Class

```
class NewDeleteAllocator : public Allocator {
    // no data members
  public:
    // CREATORS
    NewDeleteAllocator() = default;
    ~NewDeleteAllocator() = default;
    NewDeleteAllocator(const NewDeleteAllocator&) = delete;

    // MANIPULATORS
    void operator=(const NewDeleteAllocator&) = delete;

    inline void *allocate(std::size_t nBytes) override {
                                return ::operator new(nBytes); }

    inline void deallocate(void *address) override {
                                  ::operator delete(address); }
};
```

Our Tool Chest of Allocation Strategies

AS2

Concrete Derived Class

```
class NewDeleteAllocator : public Allocator {
    // no data members
  public:
    // CREATORS
    NewDeleteAllocator() = default;
    ~NewDeleteAllocator() = default;
    NewDeleteAllocator(const NewDeleteAllocator&) = delete;

    // MANIPULATORS
    void operator=(const NewDeleteAllocator&) = delete;

    inline void *allocate(std::size_t nBytes) override {
                                  return ::operator new(nBytes); }

    inline void deallocate(void *address) override {
                                 ::operator delete(address); }
};
```

Yes these are **inline** and for good reason!

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myFunction()
{
   NewDeleteAllocator a;
   std::pmr::vector<int> v(&a);

   // …
}
```

Via Protocol

Virtual functions can (sometimes) be bound at compile time!

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
} // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
} // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark()
{
    const int N = 1000;

    std::vector<std::pmr::list<int> *> system(N);

    NewDeleteAllocator a;

    for (int i = 0; i < N; ++i) {
        system[i] = new std::pmr::list<int>(&a);
        // build up list of elements
    }

    // Do benchmark (e.g., access links).

    for (auto ptr : system) {
        delete ptr;
    }
}  // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

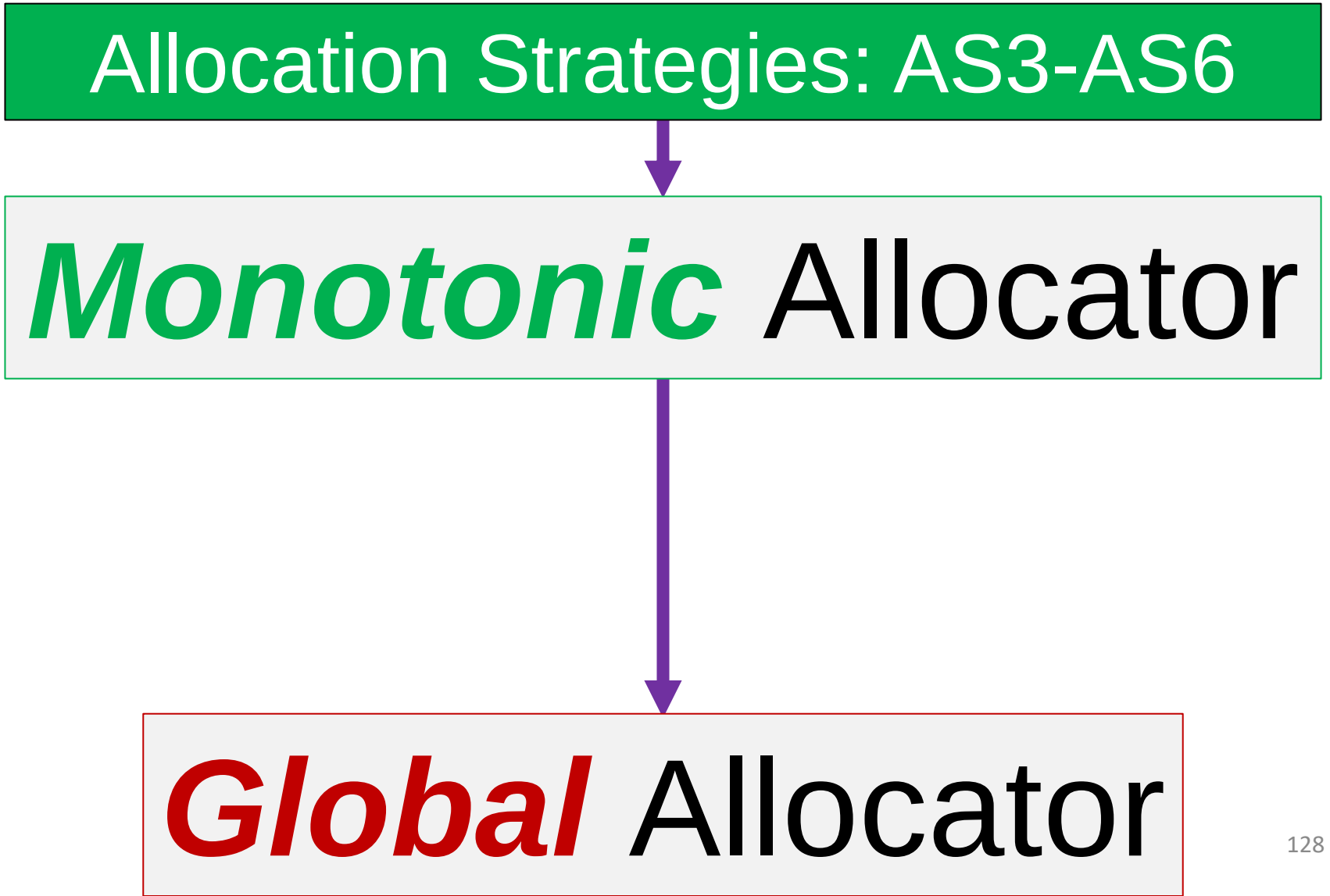
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

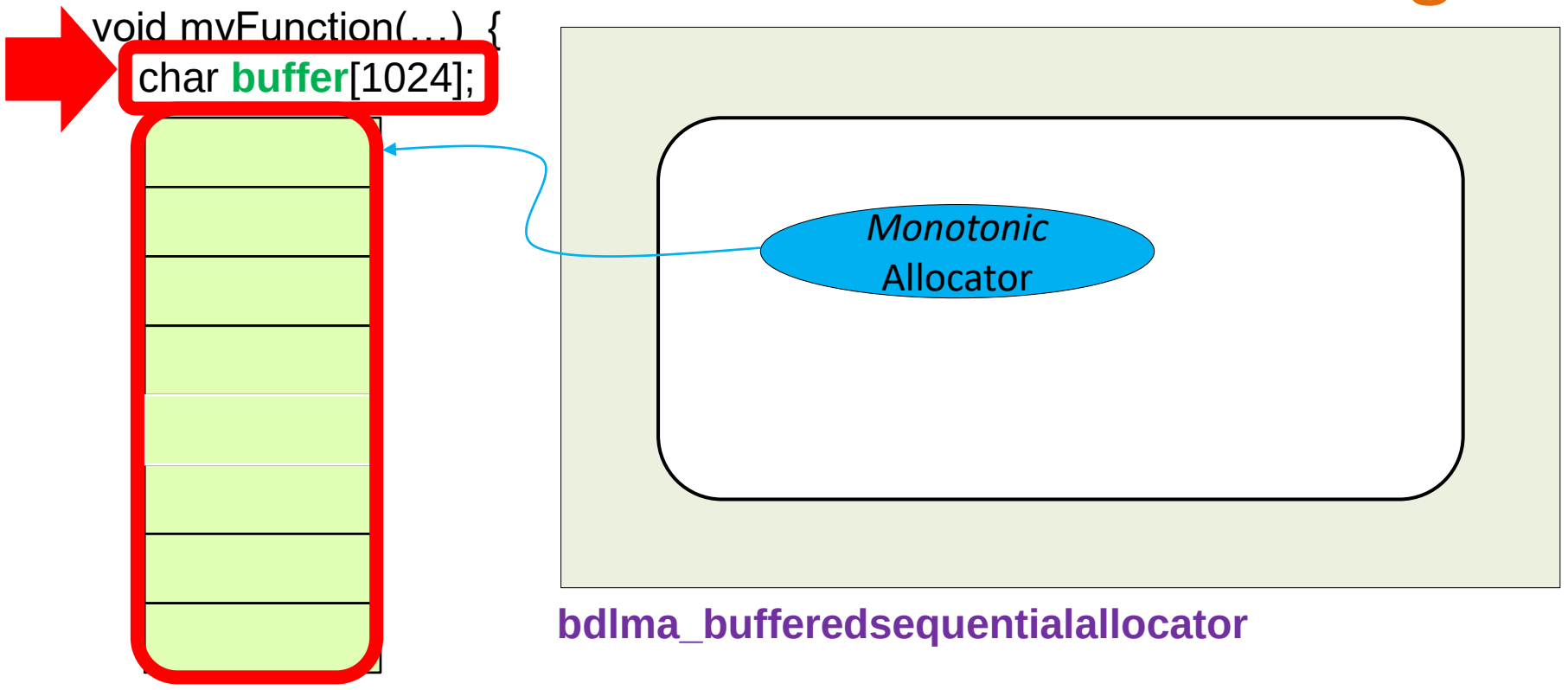
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies

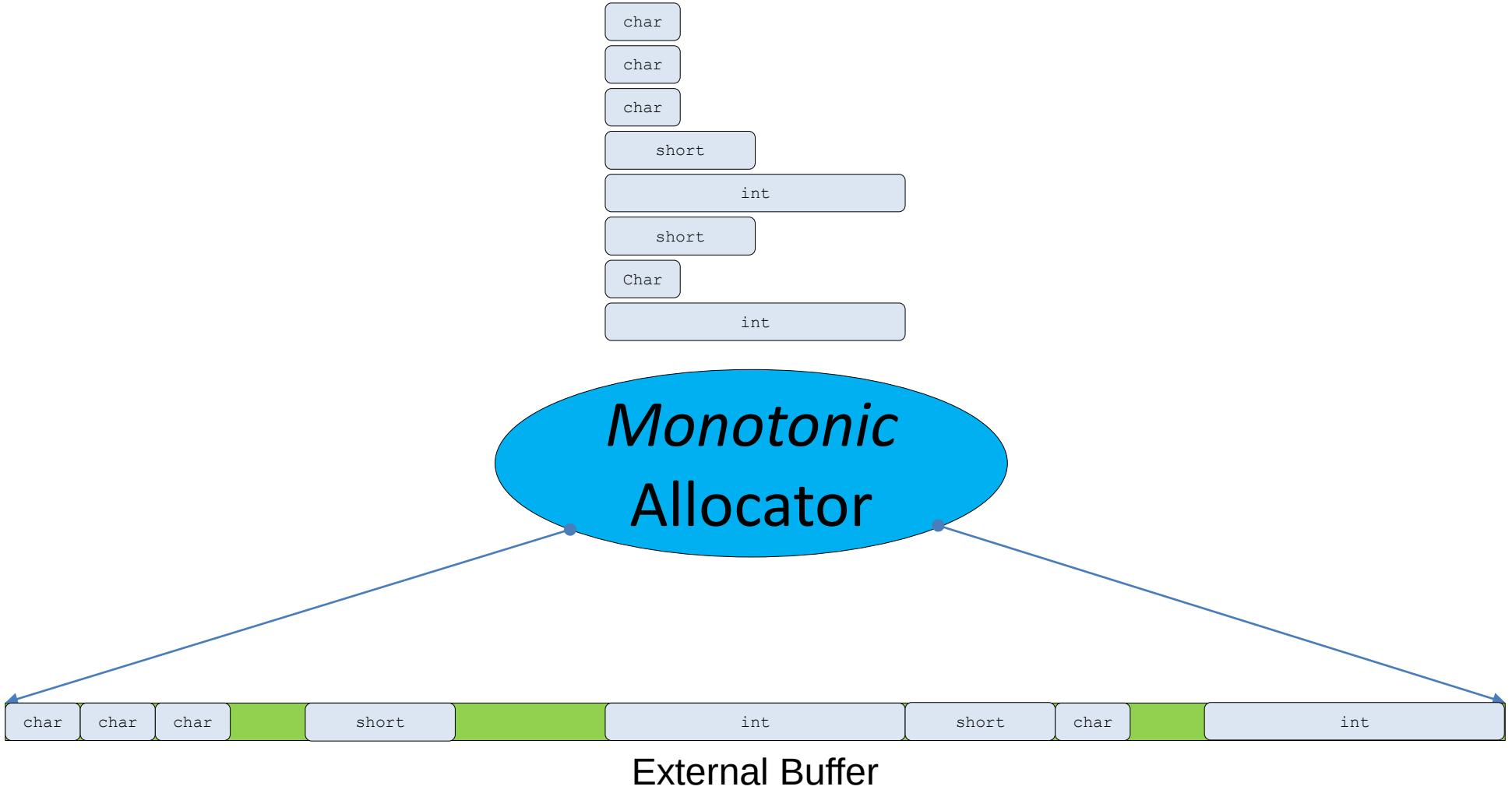


bdlma_bufferedsequentialallocator

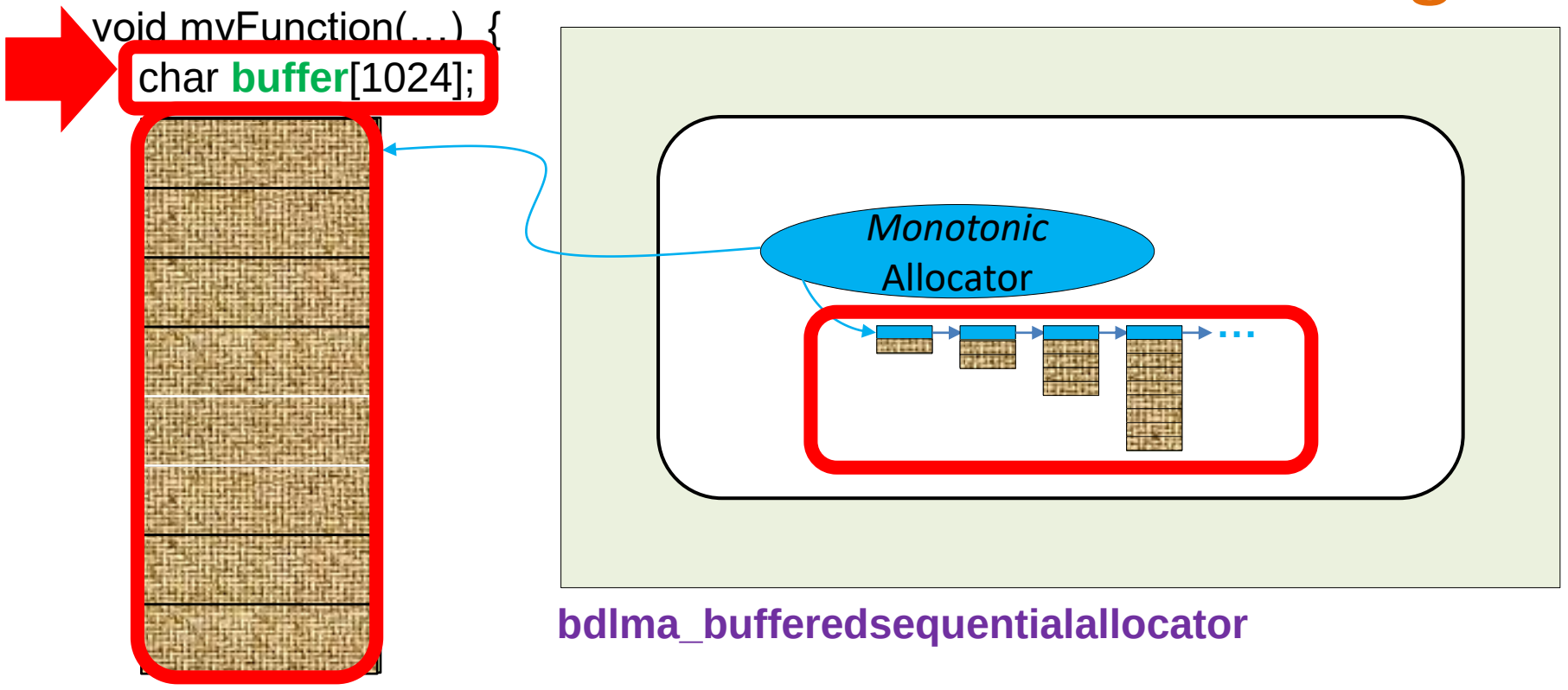
```
    bdlma::BufferedSequentialAllocator localAllocator(buffer, sizeof buffer);
    bsl::vector(&localAllocator);
     // …
}
```

2. Understanding the Problem

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies



bdlma_bufferedsequentialallocator

```
    bdlma::BufferedSequentialAllocator localAllocator(buffer, sizeof buffer);
    bsl::vector(&localAllocator);
    // …
}
```

Note that deallocate is a No-Op!

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Can also supply buffer on the stack!

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Normal Destruction

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

(magically "Winked Out")

Our Tool Chest of Allocation Strategies

```
myBenchmark() {
    const int N = 1000;
    std::vector<std::list<int, wrapper> *> system(N);
    bdlma::BufferedSequentialAllocator a; // monotonic
    for (int i = 0; i < N; ++i) {
        void *p = a.allocate(sizeof std::list<int, wrapper>);
        system[i] = new(p) std::list<int, wrapper>(&a);
        // build up list of elements
    }
    // Do benchmark (e.g., access links).
    for (auto ptr : system) {
        ptr->~std::list<int, wrapper>();
        a.deallocate(ptr);
    }
}    // 'a' goes out of scope (and is destroyed).
     // 'system' goes out of scope (and is destroyed).
```

Legal C++!

(magically "Winked Out")

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

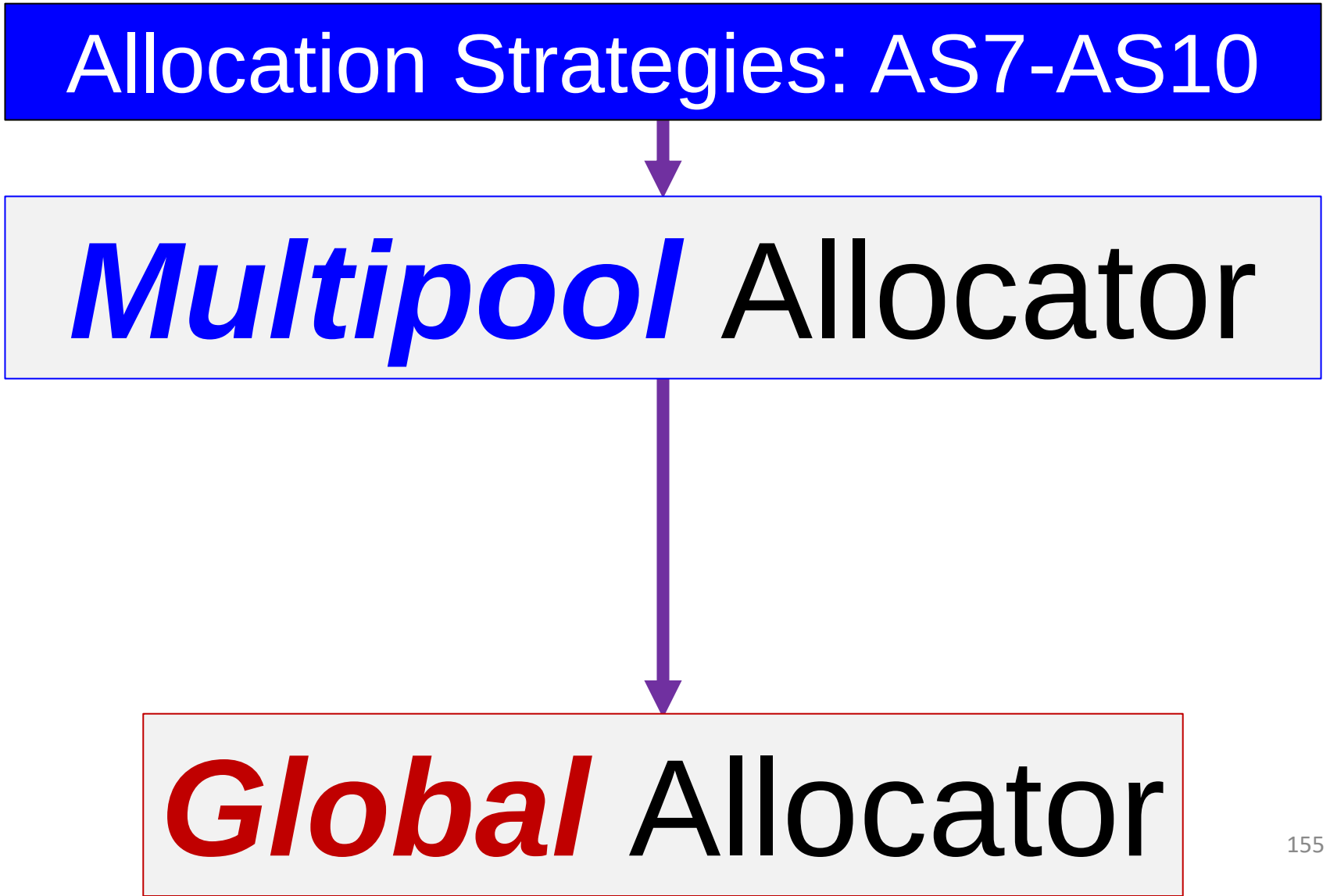
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

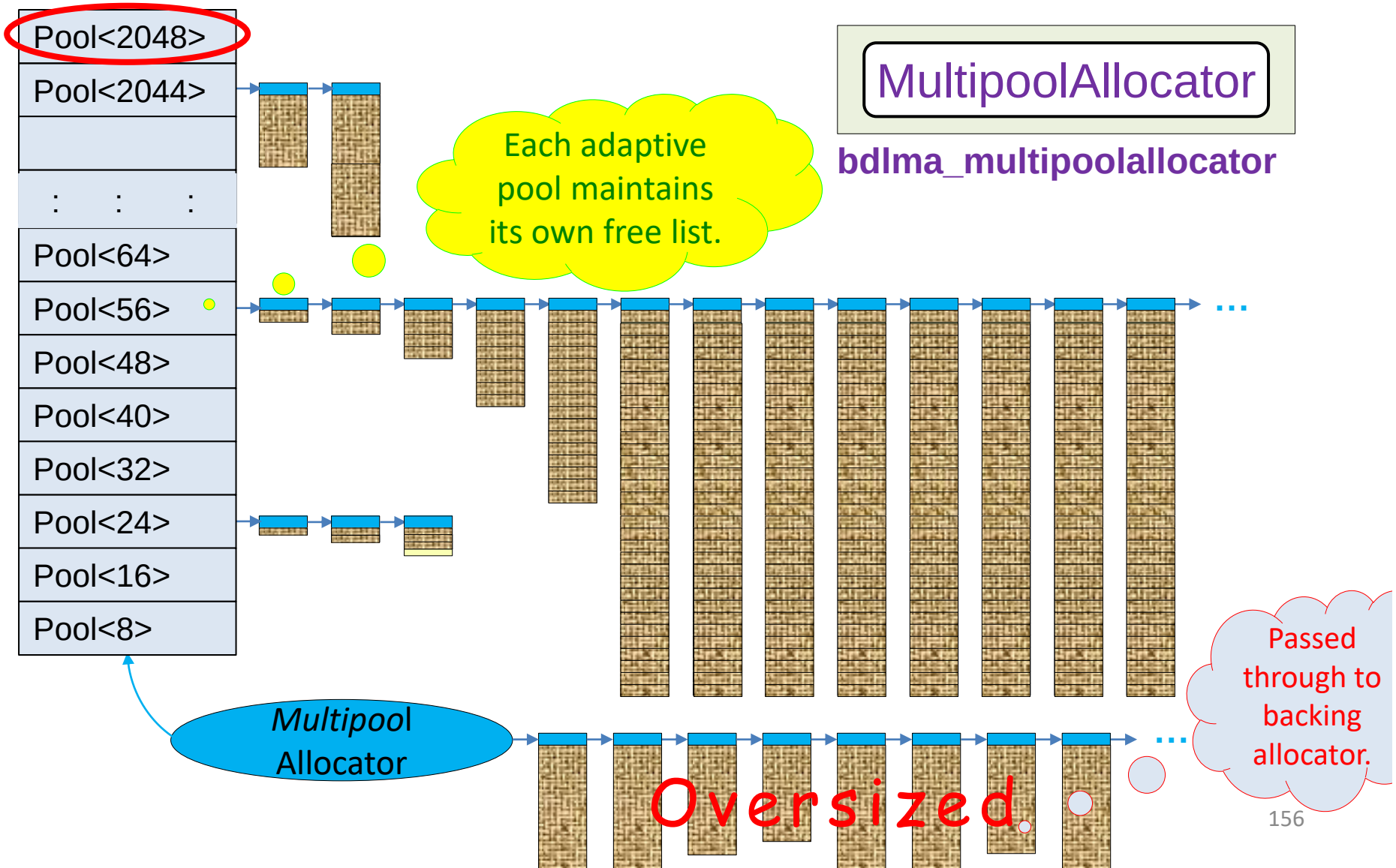
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

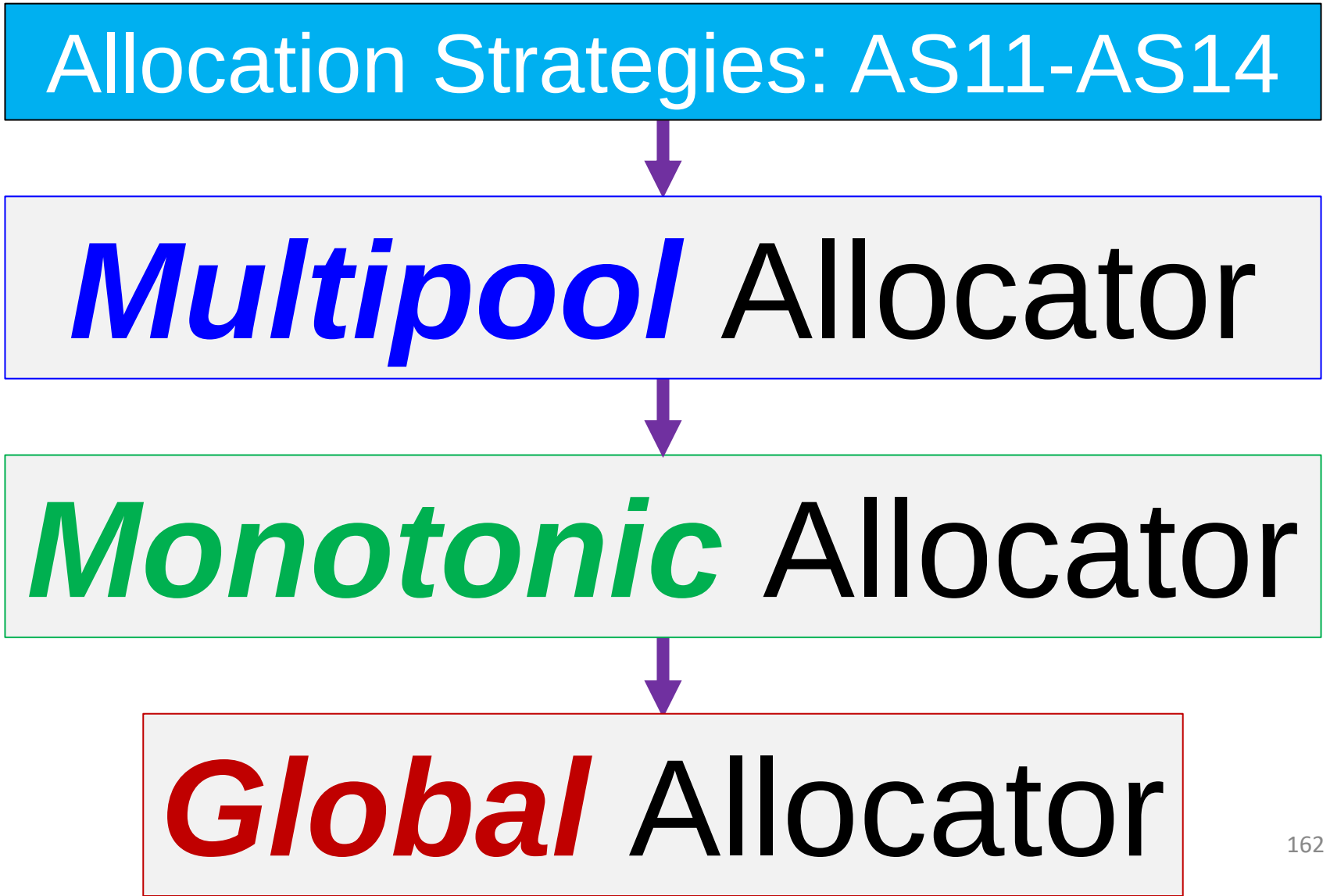
Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies



Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Our Tool Chest of Allocation Strategies

Label	Allocator Type	Allocator Binding	Destruction of Allocated Objects
AS1	Default Global Allocator	Type Parameter	Normal Destruction
AS2	New/Delete Allocator	Abstract Base	Normal Destruction
AS3	Monotonic	Type Parameter	Normal Destruction
AS4	Monotonic	Type Parameter	(magically) “Winked Out”
AS5	Monotonic	Abstract Base	Normal Destruction
AS6	Monotonic	Abstract Base	(magically) “Winked Out”
AS7	Multipool	Type Parameter	Normal Destruction
AS8	Multipool	Type Parameter	(magically) “Winked Out”
AS9	Multipool	Abstract Base	Normal Destruction
AS10	Multipool	Abstract Base	(magically) “Winked Out”
AS11	Multipool<Monotonic>	Type Parameter	Normal Destruction
AS12	Multipool<Monotonic>	Type Parameter	(magically) “Winked Out”
AS13	Multipool<Monotonic>	Abstract Base	Normal Destruction
AS14	Multipool<Monotonic>	Abstract Base	(magically) “Winked Out”

Characterizing Usage Scenarios

What basic *"size" parameters* characterize *software usage*?

Characterizing Usage Scenarios

Two fundamental (architectural) *"size" parameters* spring to mind:

Characterizing Usage Scenarios

Two fundamental (architectural) *"size" parameters* spring to mind:

N: number of **instructions** executed (per thread)

Characterizing Usage Scenarios

Two fundamental (architectural) *"size" parameters* spring to mind:

N: number of **instructions** executed (per thread)

W: number of active (worker) **threads**

Characterizing Usage Scenarios

Two fundamental (architectural*) *"size" parameters* spring to mind:

N: number of **instructions** executed (per thread)

W: number of active (worker) **threads**

*Note that these parameters are deliberately chosen to be platform independent.

Characterizing Usage Scenarios

What *"aspects"* of software affect optimal *allocation strategy*?

Characterizing Usage Scenarios

We initially proposed five *"dimensions"* to help us characterize *aspects* of software usage:



Characterizing Usage Scenarios

We initially proposed five *"dimensions"* to help us characterize *aspects* of software usage:

- **D** DENSITY of *allocation operations*
- **V** VARIATION of *allocated sizes*
- **L** LOCALITY of *accessed memory*
- **U** UTILIZATION of *allocated memory*
- **C** CONTENTION of *concurrent allocations*

Characterizing Usage Scenarios

These "*dimensions*" are intended to be rough indications, not precise measures:

Characterizing Usage Scenarios

These "*dimensions*" are intended to be rough indications, not precise measures:

- They are intended to be scaled (somehow) to fit the range [0 .. 1].

Characterizing Usage Scenarios

These "*dimensions*" are intended to be rough indications, not precise measures:

- They are intended to be scaled (somehow) to fit the range [0 .. 1].
- 0 implies the *minimum* for the *aspect*, whereas 1 implies the *maximum*.

Characterizing Usage Scenarios

These "*dimensions*" are intended to be rough indications, not precise measures:

- They are intended to be scaled (somehow) to fit the range [0 .. 1].
- 0 implies the *minimum* for the *aspect*, whereas 1 implies the *maximum*.
- Note that these *"dimensions"* are far from independent.

Characterizing Usage Scenarios

DENSITY of *allocation operations*

$$\mathbf{D} = \frac{numAllocDeallocOps}{\mathbf{N}}$$

Characterizing Usage Scenarios

DENSITY of *allocation operations*

$$\mathbf{D} = \frac{numAllocDeallocOps}{\mathbf{N}}$$

0.0: No allocation instructions occur.

1.0: Every instruction is either `allocate` or `deallocate`.

Characterizing Usage Scenarios

DENSITY of *allocation operations*

$$\mathbf{D} = \frac{numAllocDeallocOps}{\mathbf{N}}$$

0.0: No allocation instructions occur.

1.0: Every instruction is either `allocate` or `deallocate`.

Consider populating `vector<int>` using `push_back`.

2. Understanding the Problem

Characterizing Usage Scenarios

VARIATION of *allocated sizes*

$$\mathbf{v} = \frac{numUniqueAllocSizes}{numAllocOps}$$

2. Understanding the Problem

Characterizing Usage Scenarios

VARIATION of *allocated sizes*

$$\mathbf{v} = \frac{numUniqueAllocSizes}{numAllocOps}$$

0.0: All allocated sizes are the same.

1.0: Every allocated size is different.

Characterizing Usage Scenarios

VARIATION of *allocated sizes*

$$\mathbf{v} = \frac{numUniqueAllocSizes}{numAllocOps}$$

0.0: All allocated sizes are the same.

1.0: Every allocated size is different.

Consider: `std::set<int>`

`std::string`

Characterizing Usage Scenarios

LOCALITY of *accessed memory*

I = num access instructions on subregion over duration

M = num bytes of memory in subregion

T = num transitions out of subregion over duration

2. Understanding the Problem

Characterizing Usage Scenarios

*Note that **LOCALITY** (**L**) can play a critical role in long-running programs even when the allocation **DENSITY** (**D**) is negligible!*

M =

T = num[illegible] over duration

$$\mathbf{L}_{physical} = \frac{\mathbf{I}}{\mathbf{M} * \mathbf{T}} \qquad \mathbf{L} = \frac{\mathbf{I}}{\mathbf{M} * \mathbf{T}} \qquad \mathbf{L}_{temporal} = \frac{\mathbf{I}}{\mathbf{M} * \mathbf{T}}$$

0.0: Subregion is large or not accessed repeatedly.

1.0: Subregion is small and accessed repeatedly.

2. Understanding the Problem

Characterizing Usage Scenarios

UTILIZATION of *allocated memory*

$$\mathbf{U} = \frac{maxMemoryInUse}{totalMemoryAllocated}$$

Characterizing Usage Scenarios

UTILIZATION of *allocated memory*

$$\mathbf{U} = \frac{maxMemoryInUse}{totalMemoryAllocated}$$

0.0: Memory is repeatedly freed/reallocated.

1.0: All allocations precede deallocations.

Characterizing Usage Scenarios

UTILIZATION of *allocated memory*

$$\mathbf{U} = \frac{maxMemoryInUse}{totalMemoryAllocated}$$

0.0: Memory is repeatedly freed/reallocated.

1.0: All allocations precede deallocations.

Consider: `vector<int>`

Assume we reserve capacity up front.

`vector<string>`

Assume strings are long.

2. Understanding the Problem

Characterizing Usage Scenarios

CONTENTION of *concurrent allocations*

$$\mathbf{c} = \frac{expectedNumConcurrentAllocations}{numberOf\,Active\,Threads\,(\mathbf{W})}$$

Characterizing Usage Scenarios

CONTENTION of *concurrent allocations*

$$\mathbf{C} = \frac{expectedNumConcurrentAllocations}{numberOf\,Active\,Threads\,(\mathbf{W})}$$

0.0: At most one thread has non-zero (**D**).

1.0: The allocation **D**ENSITY (**D**) per thread is 1.

Characterizing Usage Scenarios

CONTENTION of *concurrent allocations*

$$\mathbf{C} = \frac{expectedNumConcurrentAllocations}{numberOf\,Active\,Threads\,(\mathbf{W})}$$

0.0: At most one thread has non-zero (**D**).

1.0: The allocation **D**ENSITY (**D**) per thread is 1.

*Note that thread **C**ONTENTION (**C**) is strongly correlated with allocation **D**ENSITY (**D**).*

Characterizing Usage Scenarios

Summary

D DENSITY of *allocation operations*

V VARIATION of *allocated sizes*

L LOCALITY of *accessed memory*

U UTILIZATION of *allocated memory*

C CONTENTION of *concurrent allocations*

2. Understanding the Problem

Characterizing Usage Scenarios

Mnemonic: ?

D DENSITY of *allocation operations*

V VARIATION of *allocated sizes*

L LOCALITY of *accessed memory*

U UTILIZATION of *allocated memory*

C CONTENTION of *concurrent allocations*

Characterizing Usage Scenarios

Mnemonic: **D.V.L.U.C.** the **DUCK**

- **D** DENSITY of *allocation operations*
- **V** VARIATION of *allocated sizes*
- **L** LOCALITY of *accessed memory*
- **U** UTILIZATION of *allocated memory*
- **C** CONTENTION of *concurrent allocations*

Outline

1. Introduction and Background
   What are memory allocators, and why are they useful?
2. Understanding the Problem
   What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   When and how do you use which allocator, and why?
4. Conclusions
   What must we remember about memory allocators?

Outline

1. Introduction and Background
   What are memory allocators, and why are they useful?
2. Understanding the Problem
   What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   When and how do you use which allocator, and why?
4. Conclusions
   What must we remember about memory allocators?

Roadmapping the Benchmarks

Considerations:

- We wanted to explore each dimension to observe its effects on optimal memory allocation.
- Our first thought was to create a single benchmark that spanned all five dimensions.
  - Find the centroid.
  - Vary the arguments along each dimension separately
  - Such a single benchmark is not at all easy.
- We finally settled on four separate benchmarks.
  - Benchmark I addresses the first two dimensions.

Tested Across Many Problem Sizes

Considerations:

- We tried not to assume the answers we expected.
  - Explored a wide range of problem sizes, **N**.
- Used successive *powers of two*.
  - We often show <u>just</u> the exponent in tables: 5 versus 2^5.
- Contrasted results across disparate physical sizes.
  - E.g., by holding overall problem size **N** constant.
- Traded-off comparable parameters, e.g.,
  - *Subsystem size* versus *number of subsystems*
  - *Subsystem iterations* versus *experiment repetitions*

Platforms Used For These Benchmarks

All of results presented here are from a server having ***dual*** Intel Xeon E5-2620v2 processors.

Each processor:

- Ivy Bridge EP – "Sandy Bridge" architecture (c. 2013)
- 6 cores (for a total of 12 cores)
- 15 MB of L3 cache
- running at a fixed clock rate of 2.1 GHz
- 16GB of DDR3-1600 RAM (13G available to processes)

See: http://ark.intel.com/products/75789

Platforms Used For These Benchmarks

All benchmark programs were

- compiled using gcc-5.1,
- using optimizing "-O3 –march=native",
- and run under Linux 3.18.

All experiments used only one core at a time

Except, that is, for Benchmark IV, which measures CONTENTION (**C**) and used more of the available cores.

Alternative Global Allocators

We investigate alternative global allocators:

- *tcmalloc*
- *jemalloc*

> "We determined that the **native allocators** (e.g., the one currently shipped with GCC on Linux) **performed as well or better**."

Benchmark Road Map

I. **Short Running:** Build Up, Use, Tear Down
   - Allocation **D**ENSITY and **V**ARIATION in Allocated Sizes

II. **Long Running:** Time-Multiplexed Subsystems
   - Access **L**OCALITY – both *Physica*l and *Temporal*

III. **Short Running:** Varying Memory Reusability
   - Memory **U**TILIZATION

IV. **Multithreaded:** Varying Numbers of Threads
   - Allocator **C**ONTENTION

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

I. **Short Running:** Build Up, Use, Tear Down
   - Allocation DENSITY and VARIATION in Allocated Sizes

II. **Long Running:** Time-Multiplexed Subsystems
   - Access LOCALITY – both *Physica*l and *Temporal*

III. **Short Running:** Varying Memory Reusability
   - Memory UTILIZATION

IV. **Multithreaded:** Varying Numbers of Threads
   - Allocator CONTENTION

Benchmark I: DENSITY, VARIATION



Benchmark I: DENSITY, VARIATION

Considerations:

- Initially we wanted to investigate allocation **D**ENSITY.
  - Focused on allocation/deallocation costs themselves.
- Chose a variety of common data structures.
  - Used `int`, `string`, `vector`, and `unordered_set`.
- Didn't want access **L**OCALITY to dominate results.
  - Wrote to *just* the first byte of each newly allocated element.
- Later Incorporated **V**ARIATION into allocated memory.
  - `vector` objects' capacities were reserved up front.
  - `string` lengths were 33*-1000 (uniformly distributed).
    - * Our *short-string-optimization* threshold is up to 32 bytes.

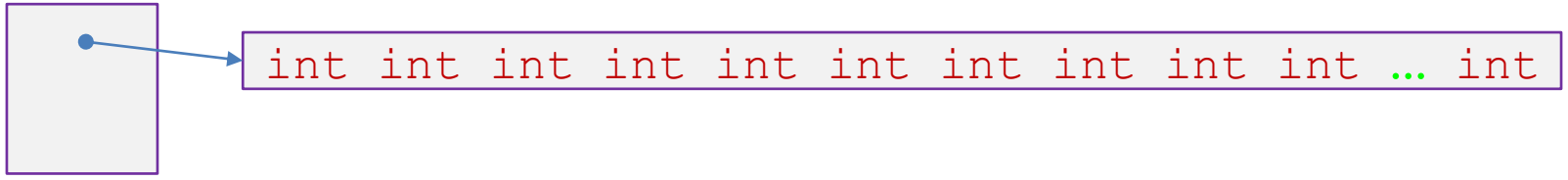
Benchmark I: DENSITY, VARIATION

Simple Data Structures

Label	Data Structure
DS1	`vector<int>`
DS2	`vector<string>`
DS3	`unordered_set<int>`
DS4	`unordered_set<string>`

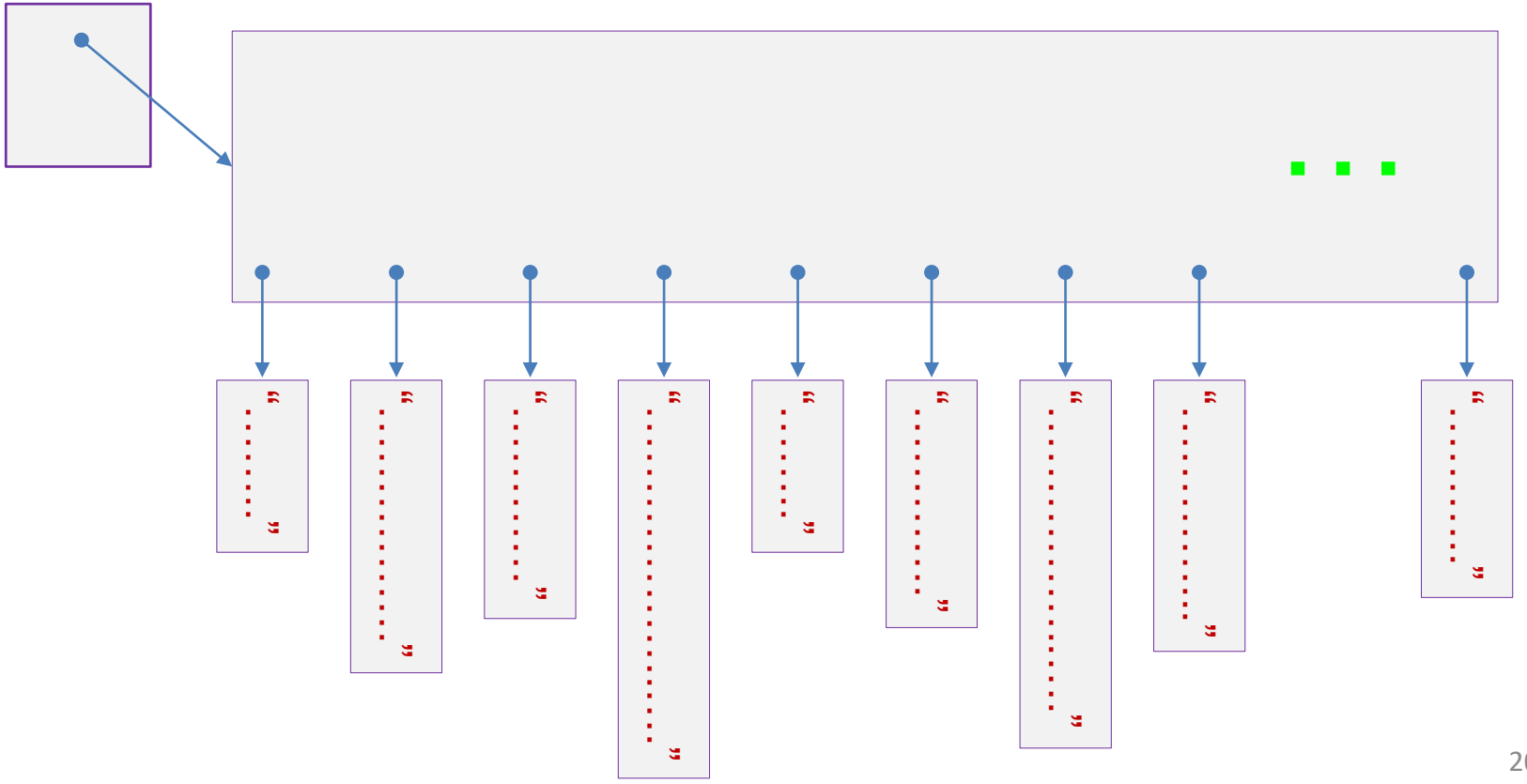
Benchmark I: DENSITY, VARIATION

DS1	vector<int>



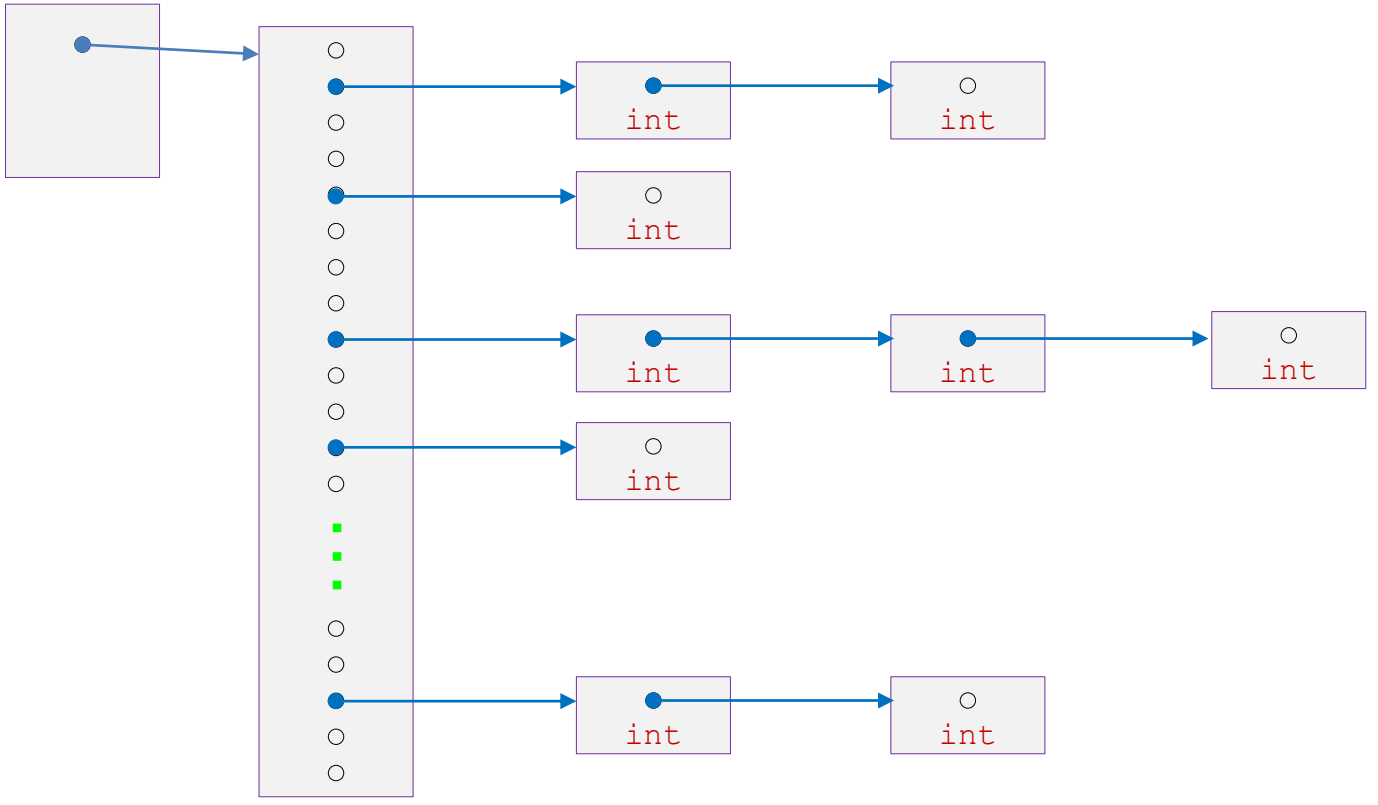
Benchmark I: DENSITY, VARIATION

DS2 vector<string>



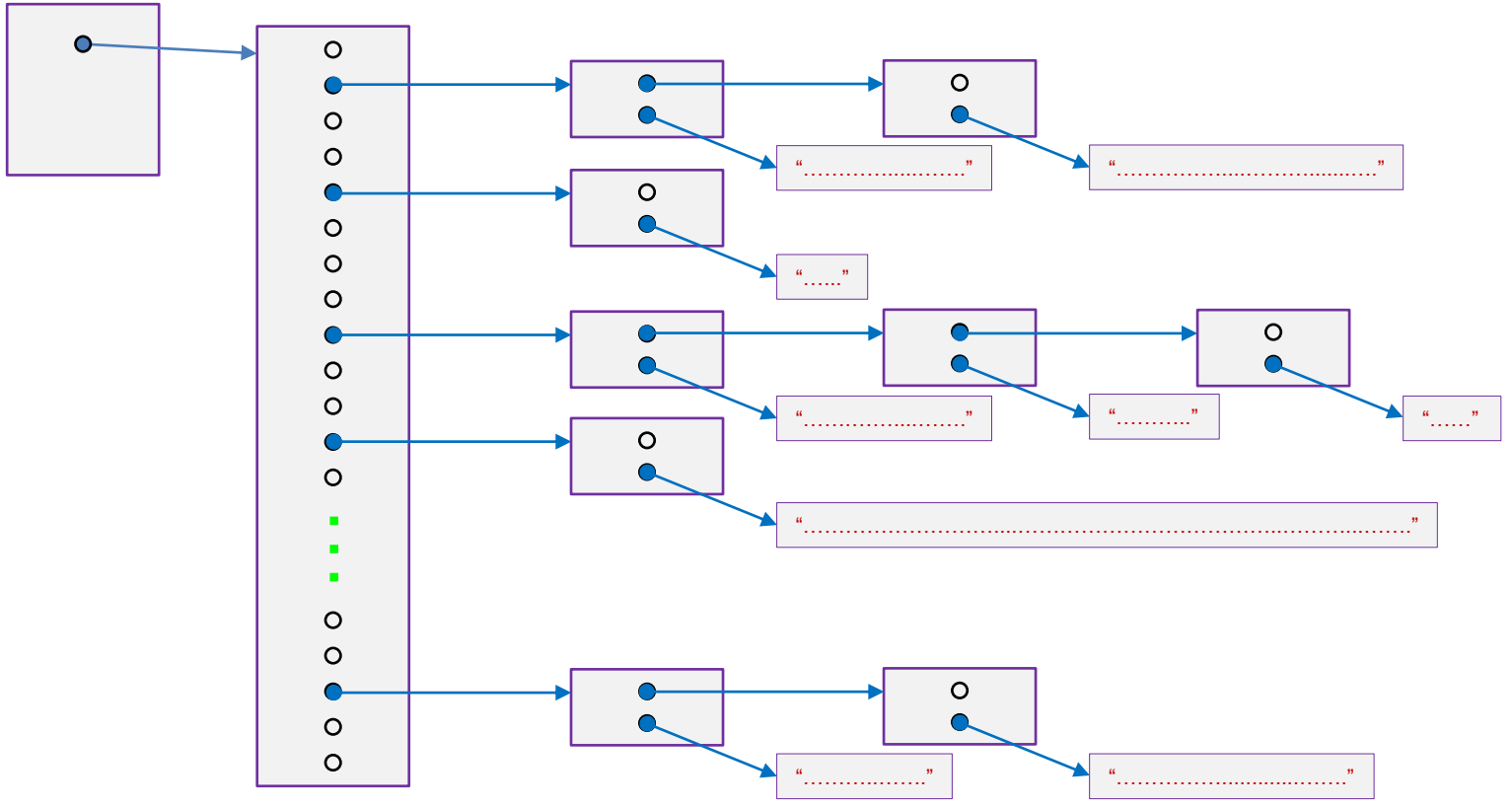
Benchmark I: DENSITY, VARIATION

DS3 unordered_set<int>



Benchmark I: DENSITY, VARIATION

DS4	unordered_set<string>



Benchmark I: DENSITY, VARIATION

Plan:

Benchmark I: DENSITY, VARIATION

Plan:

- For each data structure in a thoughtfully chosen set:
  - Create the data structure.
  - Access it lightly.
  - Destroy it.
  - Repeat (until the problem size *N* is reached).

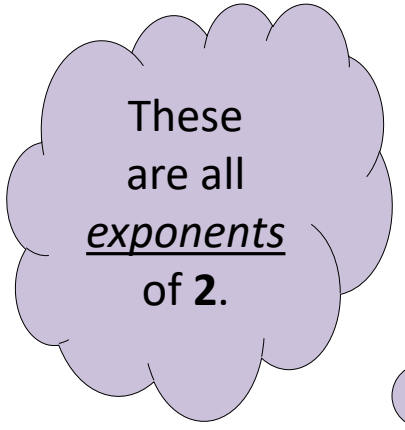
Benchmark I: DENSITY, VARIATION

Plan:

- For each data structure in a thoughtfully chosen set:
  - Create the data structure.
  - Access it lightly.
  - Destroy it.
  - Repeat (until the problem size *N* is reached).
- We chose the overall problem size to be $N = 2^{27}$.
  - The container *size (S)* varies from 2^8 to 2^{16}.
  - The number of experiment repetitions *(R)* = *N*/*S*.

Benchmark I: DENSITY, VARIATION

Contrasting access times across (sub)system sizes



Overall Problem Size = 2^{27}

$\log_2 \mathbf{N} = 27$

Container Size (S)	Experiment Repetitions (R)
8	19
9	18
10	17
11	16
12	15
13	14
14	13
15	12
16	11

3. Analyzing the Benchmark Data

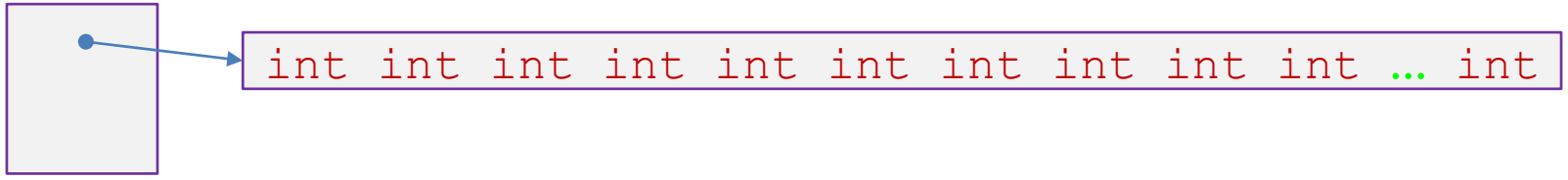
Benchmark I: DENSITY, VARIATION

Plan:

- For each data structure in a thoughtfully chosen set:
  - Create the data structure.
  - Access it lightly.
  - Destroy it.
  - Repeat (until the problem size *N* is reached).
- We chose the overall problem size to be $N = 2^{27}$.
  - The container *size (S)* varies from 2^8 to 2^{16}.
  - The number of repetitions *(R) = N/S.*
- Each result entry is absolute **RUN TIME** (in seconds).

Benchmark I: DENSITY, VARIATION

DS1	vector<int>



3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

DS1

Each result entry represents absolute runtime in seconds.

`vector<int>`

Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

Benchmark I: DENSITY, VARIATION

DS1 vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual			Virtual			Virtual				Virtual		
				(wink)		(wink)		(wink)		(wink)		(wink)		(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

DS1 vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

DS1 vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

220

750

Benchmark I: DENSITY, VARIATION

DS1 vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

3. Analyzing the Benchmark Data

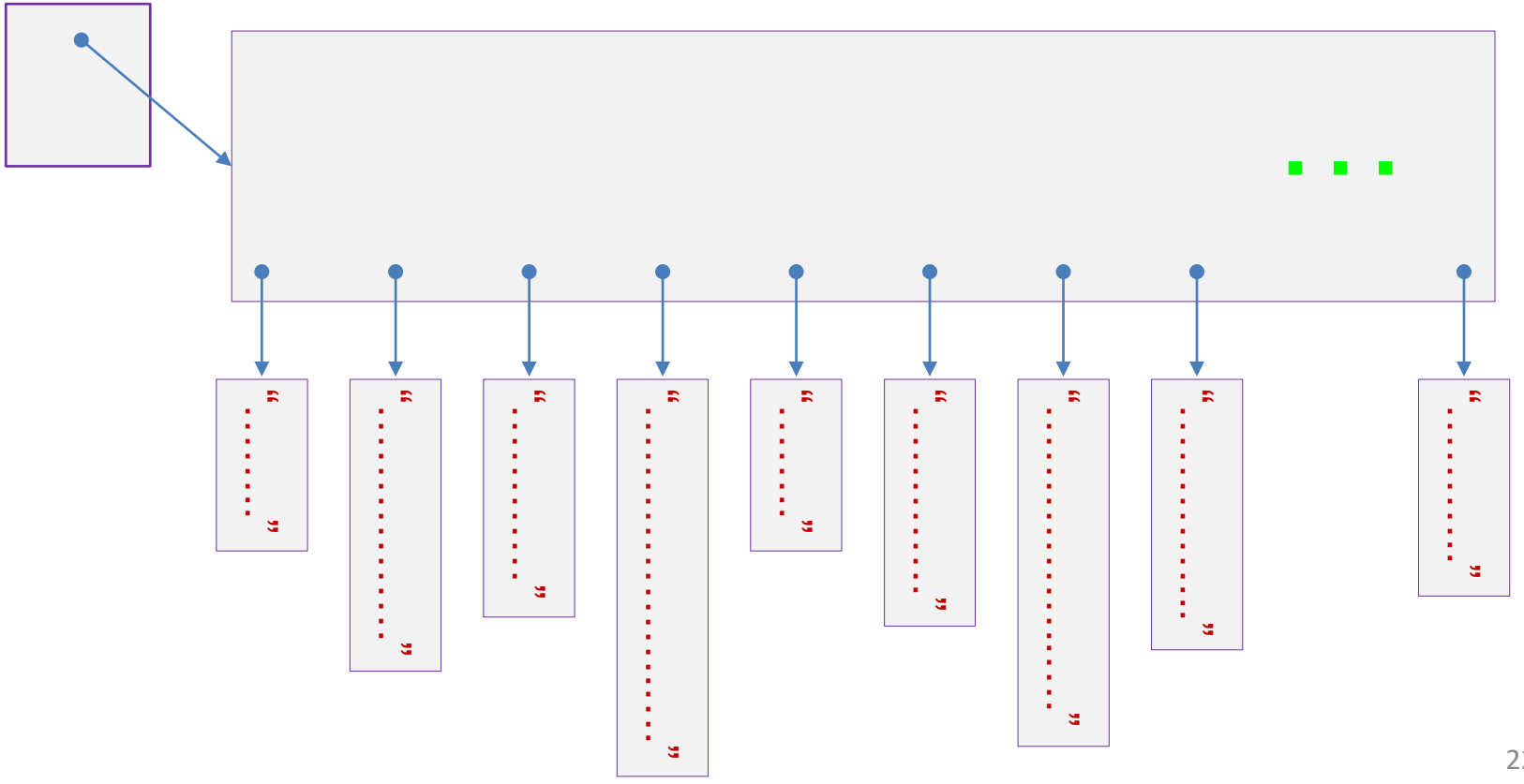
Benchmark I: DENSITY, VARIATION

DS1 vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

Benchmark I: DENSITY, VARIATION

DS2	vector<string>



3. Analyzing the Benchmark Data

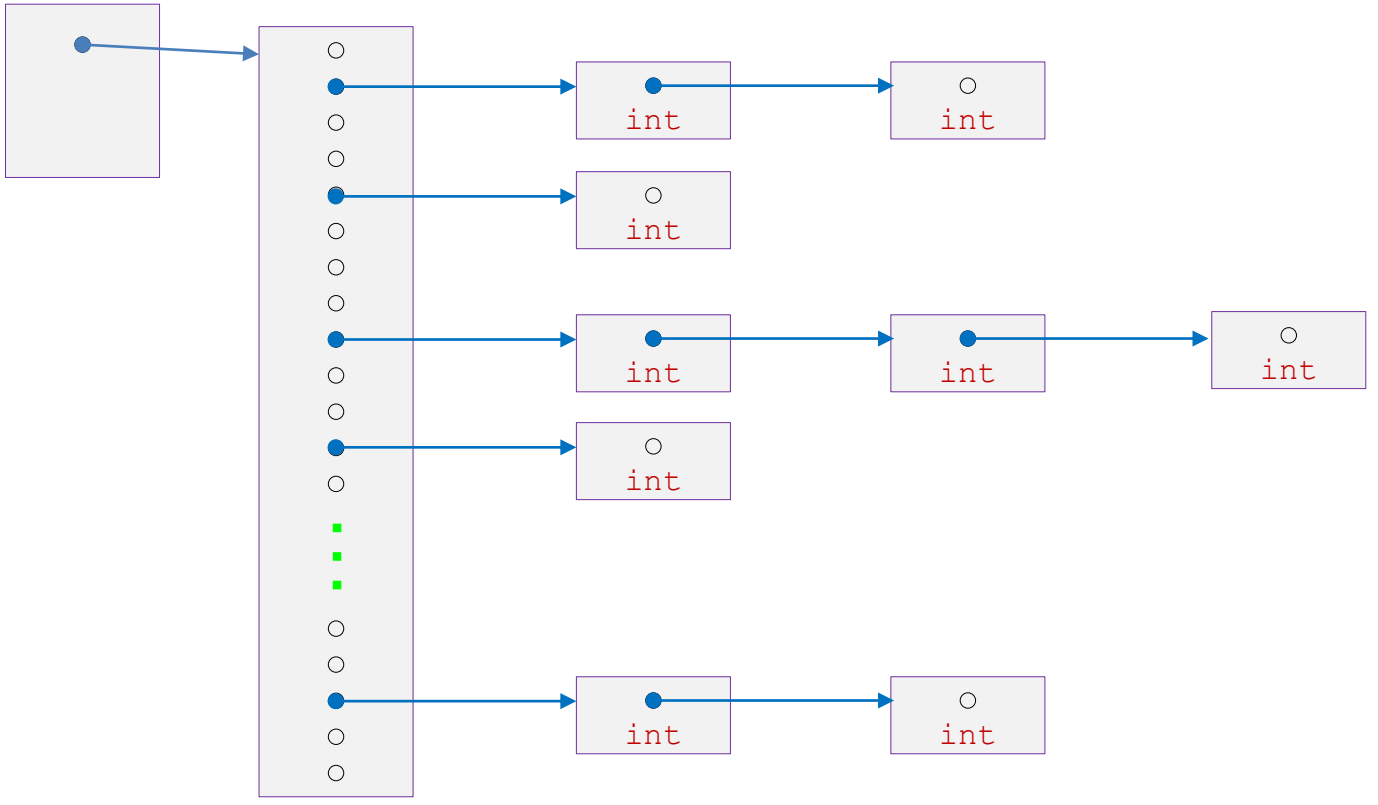
Benchmark I: DENSITY, VARIATION

DS2 vector<string>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	68.9	67.3	12.9	12.8	13.3	12.9	18.1	17.8	18.2	17.7	15.5	14.8	15.6	14.8
2^7	68.8	68.2	12.8	12.9	13.2	12.9	20.6	20.2	20.6	20.4	15.1	14.3	15.0	14.4
2^8	70.8	68.9	13.2	12.8	13.6	12.9	30.8	30.4	30.7	30.3	15.3	14.6	15.4	14.7
2^9	73.1	71.2	13.5	13.5	13.9	13.5	38.2	37.6	38.0	37.3	15.9	15.1	15.9	15.1
2^{10}	75.4	74.3	13.6	13.5	14.0	13.7	41.1	40.3	41.6	40.9	16.0	15.1	15.9	15.0
2^{11}	76.9	74.5	13.6	13.5	14.1	13.6	43.9	43.2	43.7	42.6	16.0	15.0	16.0	15.1
2^{12}	76.1	74.8	13.7	13.5	14.0	13.6	41.2	38.8	40.6	39.4	15.9	14.9	15.8	15.0
2^{13}	76.1	74.8	13.6	13.6	14.0	13.6	41.4	39.2	41.3	39.9	15.9	15.0	15.8	14.9
2^{14}	78.3	76.5	13.6	13.6	14.0	13.6	45.8	42.3	44.8	44.0	16.1	15.2	16.2	15.4
2^{15}	90.4	91.0	20.2	20.1	20.5	20.1	62.2	58.7	62.2	58.2	26.0	25.0	26.0	24.9
2^{16}	103.0	103.0	21.5	21.3	21.8	21.3	66.5	59.2	65.1	59.9	27.0	25.3	27.1	25.2

Benchmark I: DENSITY, VARIATION

DS3	unordered_set<int>



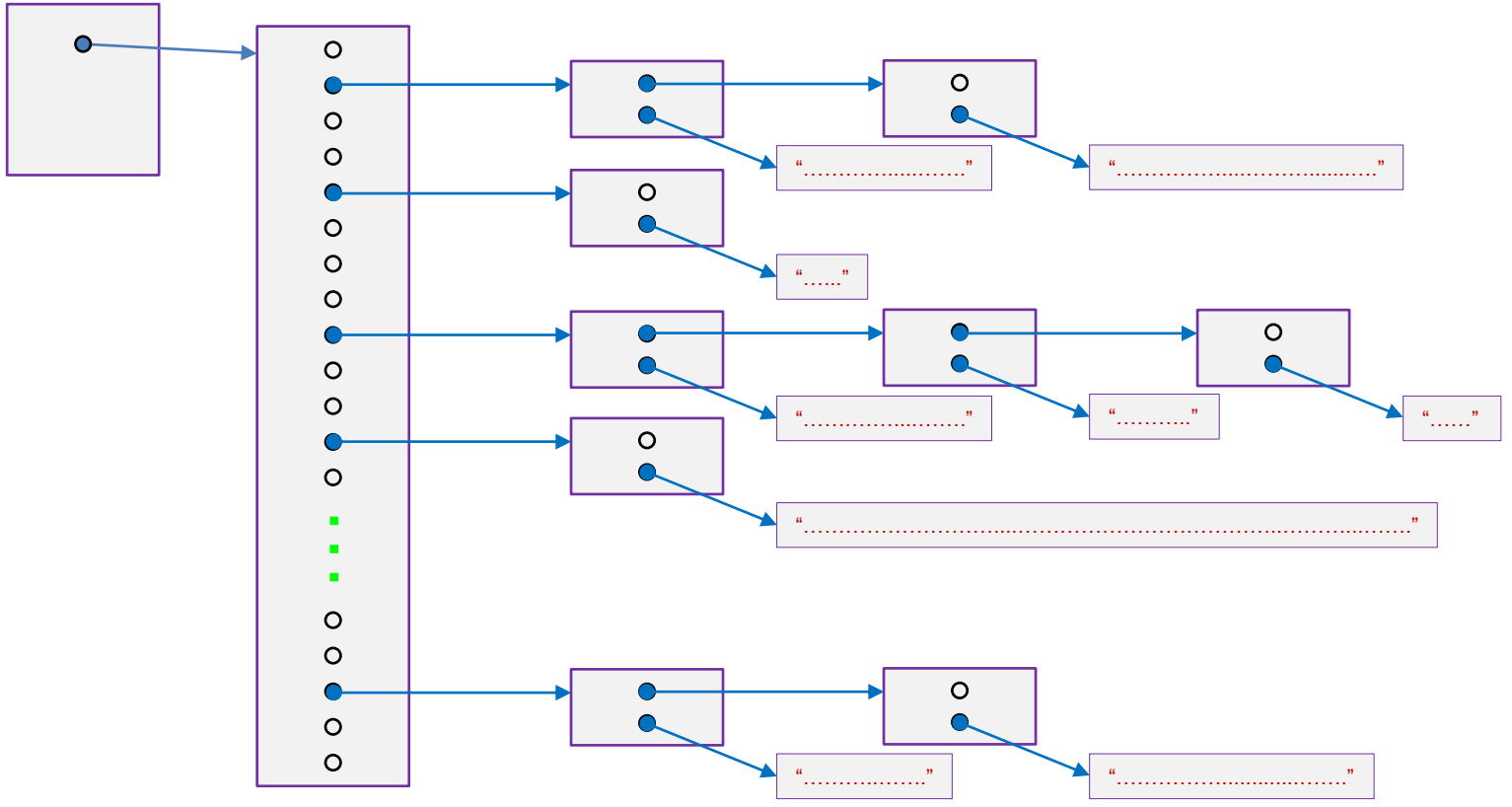
Benchmark I: DENSITY, VARIATION

DS3 unordered_set<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	10.20	11.00	5.08	4.88	5.62	5.34	7.16	7.12	7.50	7.20	6.19	5.73	6.40	5.81
2^7	12.50	13.30	5.04	4.81	5.68	5.24	6.37	6.22	6.71	6.31	5.80	5.46	6.08	5.50
2^8	15.80	16.40	4.99	4.79	5.54	5.22	5.95	5.81	6.21	5.92	5.65	5.32	5.82	5.40
2^9	18.30	19.00	5.01	4.80	5.53	5.18	5.78	5.56	6.01	5.70	5.56	5.20	5.76	5.21
2^{10}	21.40	22.30	4.99	4.83	5.55	5.20	5.72	5.46	5.95	5.55	5.52	5.27	5.68	5.24
2^{11}	25.50	26.10	4.98	4.81	5.56	5.16	5.67	5.44	5.86	5.65	5.53	5.23	5.69	5.26
2^{12}	27.10	28.00	5.02	4.81	5.55	5.20	6.42	6.10	6.57	6.25	5.51	5.12	5.68	5.27
2^{13}	27.90	28.80	5.03	4.81	5.59	5.21	7.34	6.91	7.46	7.03	5.61	5.16	5.71	5.24
2^{14}	28.50	29.00	5.03	4.80	5.58	5.26	7.03	6.59	7.18	6.68	5.64	5.19	5.80	5.34
2^{15}	28.30	29.20	5.03	4.78	5.56	5.28	7.11	6.65	7.20	6.83	5.68	5.17	5.78	5.24
2^{16}	31.60	31.80	5.02	4.76	5.60	5.22	6.79	6.37	6.93	6.46	5.68	5.17	5.79	5.24

Benchmark I: DENSITY, VARIATION

DS4 unordered_set<string>



Benchmark I: DENSITY, VARIATION

DS4 unordered_set<string>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	103.0	120.0	52.2	51.9	52.4	51.2	58.4	57.6	59.7	58.9	55.1	54.1	56.9	55.3
2^7	103.0	122.0	52.5	52.1	52.9	51.8	63.3	61.9	64.4	63.8	55.3	54.0	56.8	55.7
2^8	109.0	128.0	53.6	53.0	53.7	52.6	76.3	74.7	77.4	75.9	56.5	54.9	57.9	56.7
2^9	113.0	134.0	54.5	53.4	54.9	53.0	83.1	81.7	82.8	81.4	57.3	56.7	58.0	56.4
2^{10}	119.0	143.0	56.6	54.9	56.9	54.6	87.6	85.9	88.1	86.5	58.8	56.9	59.2	57.3
2^{11}	122.0	144.0	57.0	55.3	57.7	54.9	90.7	89.2	90.7	88.4	59.4	57.6	60.0	57.8
2^{12}	122.0	146.0	57.9	55.9	58.4	55.7	93.2	90.7	93.2	90.7	60.5	58.3	60.7	58.4
2^{13}	124.0	148.0	58.2	56.3	58.5	55.9	95.1	91.5	94.3	92.0	60.5	58.2	60.7	58.7
2^{14}	139.0	166.0	59.1	57.3	59.6	56.8	98.5	94.1	97.8	95.8	61.8	59.6	62.2	60.0
2^{15}	176.0	211.0	66.0	62.7	66.2	62.4	121.0	115.0	122.0	115.0	76.5	73.3	76.8	74.0
2^{16}	196.0	232.0	78.5	72.0	79.1	71.0	137.0	127.0	136.0	127.0	87.1	82.4	87.8	82.9

Benchmark I: **D**ENSITY, **V**ARIATION

Questions and/or Discussion?

Benchmark I: DENSITY, VARIATION

Composite Data Structures

Label	Data Structure
DS5	`vector<vector<int>>`
DS6	`vector<vector<string>>`
DS7	`vector<unordered_set<int>>`
DS8	`vector<unordered_set<string>>`
DS9	`unordered_set<vector<int>>`
DS10	`unordered_set<vector<string>>`
DS11	`unordered_set<unordered_set<int>>`
DS12	`unordered_set<unordered_set<string>>`

Benchmark I: DENSITY, VARIATION

Composite Data Structures:

- The composite data elements were much larger.
- We wanted run times to be roughly comparable.
- We kept the overall problem size $N = 2^{27}$.
  - The outer container *size (S) still* varies from 2^8 to 2^{16}.
  - The inner container size *(K)* was fixed at 2^7.
  - Now, the number of repetitions *(R)* = $N/(K \cdot S)$.

Benchmark I: DENSITY, VARIATION

Contrasting access times across (sub)system sizes

These are all *exponents* of **2**.

Overall Problem Size = 2^{27}

$\log_2 \mathbf{N} = 27$

Outer Container Size (S)	(fixed) Inner Container Size (K)	Experiment Repetitions (R)
8	7	12
9	7	11
10	7	10
11	7	9
12	7	8
13	7	7
14	7	6
15	7	5
16	7	4

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

DS5

`vector<vector<int>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
$_2^6$	0.97	1.00	0.19	0.13	0.20	0.17	0.24	0.20	0.20	0.21	0.21	0.19	0.20	0.21
$_2^7$	0.96	0.96	0.22	0.16	0.18	0.14	0.21	0.20	0.19	0.20	0.16	0.20	0.21	0.19
$_2^8$	0.99	1.00	0.19	0.13	0.18	0.17	0.27	0.30	0.27	0.29	0.19	0.19	0.20	0.21
$_2^9$	0.99	1.02	0.19	0.13	0.18	0.14	0.36	0.33	0.33	0.36	0.19	0.15	0.20	0.20
$_2^{10}$	1.01	1.04	0.19	0.18	0.19	0.14	0.37	0.36	0.36	0.38	0.22	0.19	0.20	0.22
$_2^{11}$	1.02	1.05	0.19	0.13	0.19	0.14	0.36	0.35	0.36	0.36	0.20	0.15	0.20	0.22
$_2^{12}$	1.03	1.05	0.19	0.19	0.22	0.18	0.33	0.36	0.32	0.32	0.20	0.21	0.20	0.19
$_2^{13}$	1.02	1.05	0.19	0.13	0.22	0.19	0.35	0.35	0.34	0.33	0.20	0.21	0.22	0.19
$_2^{14}$	1.05	1.10	0.19	0.17	0.19	0.16	0.38	0.36	0.38	0.37	0.17	0.19	0.20	0.19
$_2^{15}$	1.13	1.18	0.22	0.19	0.19	0.16	0.50	0.45	0.47	0.45	0.21	0.21	0.17	0.18
$_2^{16}$	1.29	1.32	0.22	0.19	0.20	0.17	0.54	0.47	0.52	0.50	0.22	0.21	0.22	0.21

Benchmark I: DENSITY, VARIATION

DS6

Little Virtual-Function Overhead

vector<vector<string>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	72.60	72.70	9.06	9.06	9.36	8.98	41.70	40.00	41.20	39.20	11.20	10.30	11.20	10.30
2^7	74.90	76.00	8.92	8.98	9.29	8.89	46.50	44.80	46.00	43.00	11.40	11.00	12.70	10.30
2^8	85.50	85.20	17.10	17.40	17.30	16.90	62.90	58.40	61.30	58.40	22.80	22.50	23.30	22.00
2^9	96.40	96.30	18.40	18.70	19.00	18.40	66.20	59.00	64.70	59.30	24.20	22.70	24.50	22.30
2^{10}	102.0	102.0	18.70	18.60	19.10	18.60	67.00	59.60	65.90	59.00	24.80	22.50	24.80	22.50
2^{11}	102.0	101.0	18.40	18.70	19.20	18.20	62.40	55.00	61.30	54.20	24.80	22.60	25.10	22.30
2^{12}	104.0	103.0	18.50	18.70	19.40	18.30	61.60	54.20	60.50	53.40	24.90	22.70	25.10	22.30
2^{13}	103.0	104.0	18.80	18.40	19.00	18.60	61.80	53.40	59.90	53.50	25.30	22.60	25.10	22.60
2^{14}	97.10	96.30	19.20	19.60	20.10	19.20	60.60	53.70	60.20	52.90	29.00	26.70	29.20	26.30
2^{15}	88.10	88.70	23.40	23.20	23.70	23.40	62.60	54.40	60.90	53.90	33.40	30.60	33.20	30.70
2^{16}	76.70	76.70	25.00	25.30	25.80	25.00	63.40	54.80	62.90	54.30	35.00	32.80	35.50	32.40

Benchmark I: DENSITY, VARIATION

DS7

More Virtual-
Function Overhead

`vector<unordered_set<int>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	28.80	28.70	2.97	2.69	3.43	2.98	4.89	4.37	5.33	4.73	3.21	2.65	3.64	3.05
2^7	28.30	28.50	2.97	2.66	3.36	2.95	4.99	4.44	5.43	4.91	3.20	2.62	3.61	2.97
2^8	28.20	28.10	2.94	2.62	3.33	2.92	5.02	4.53	5.53	4.97	3.23	2.60	3.60	3.01
2^9	31.80	31.70	2.92	2.61	3.33	2.93	5.08	4.54	5.52	4.92	3.16	2.58	3.58	2.96
2^{10}	46.60	47.20	2.92	2.61	3.33	2.89	5.07	4.49	5.48	4.93	3.15	2.58	3.57	2.98
2^{11}	54.30	54.10	2.92	2.61	3.33	2.89	5.63	4.75	5.88	5.37	3.16	2.60	3.61	2.98
2^{12}	54.70	54.80	2.96	2.66	3.34	2.91	6.90	5.79	7.28	6.23	4.15	3.05	4.58	3.40
2^{13}	55.10	56.00	3.51	2.95	3.77	3.21	7.01	6.03	7.47	6.35	4.27	3.08	4.65	3.48
2^{14}	51.00	50.90	3.53	2.99	3.81	3.25	7.08	6.00	7.47	6.46	4.29	3.14	4.71	3.47
2^{15}	44.80	45.40	3.58	3.01	3.83	3.26	7.07	6.04	7.55	6.52	4.35	3.14	4.75	3.53
2^{16}	38.20	38.20	3.58	3.06	3.86	3.30	7.14	6.11	7.58	6.47	4.37	3.18	4.80	3.54

235

Benchmark I: DENSITY, VARIATION

Winking Matters!

DS8 vector<unordered_set<string>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	114.0	116.0	26.0	23.8	26.3	24.0	56.2	54.7	56.9	54.6	27.5	25.8	27.9	26.0
2^7	123.0	130.0	26.5	24.4	25.7	23.5	62.7	60.1	62.7	60.5	27.5	26.3	28.2	26.1
2^8	162.0	171.0	31.7	27.3	32.2	27.8	78.0	74.2	79.2	73.9	35.0	32.0	35.5	32.5
2^9	175.0	181.0	36.8	28.0	38.1	28.0	81.7	74.1	81.2	74.9	36.3	32.1	37.2	32.1
2^{10}	176.0	183.0	40.0	28.9	37.4	28.2	82.1	74.5	82.1	74.7	36.9	32.0	37.4	32.2
2^{11}	176.0	183.0	39.3	28.0	37.3	28.0	81.4	74.4	82.0	74.3	36.9	32.1	37.8	32.1
2^{12}	179.0	185.0	39.4	28.0	37.1	28.0	81.8	74.1	81.6	74.4	37.0	32.0	37.8	32.2
2^{13}	173.0	178.0	39.6	27.9	36.9	28.2	81.8	73.6	81.5	74.3	37.2	32.0	37.8	32.4
2^{14}	157.0	160.0	41.0	29.9	38.8	29.9	81.5	74.1	82.2	74.0	44.0	39.3	45.1	39.2
2^{15}	122.0	131.0	47.6	35.8	44.8	36.2	85.2	75.5	83.7	76.1	50.5	45.2	51.0	45.5
2^{16}	95.4	106.0	51.4	40.5	48.1	38.9	84.8	76.2	88.7	75.9	53.1	48.5	54.8	48.2

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

DS9 `unordered_set<vector<int>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	0.97	0.94	0.23	0.19	0.24	0.21	0.26	0.27	0.30	0.26	0.25	0.26	0.25	0.24
2^7	1.40	1.43	0.22	0.21	0.22	0.19	0.24	0.26	0.25	0.27	0.24	0.26	0.24	0.24
2^8	1.35	1.39	0.25	0.22	0.24	0.23	0.30	0.35	0.34	0.33	0.24	0.23	0.25	0.24
2^9	1.29	1.32	0.22	0.18	0.22	0.17	0.37	0.38	0.37	0.36	0.23	0.22	0.19	0.22
2^{10}	1.32	1.38	0.24	0.22	0.22	0.19	0.41	0.39	0.42	0.39	0.23	0.24	0.23	0.22
2^{11}	1.34	1.36	0.23	0.21	0.22	0.17	0.44	0.42	0.43	0.41	0.23	0.23	0.25	0.22
2^{12}	1.34	1.41	0.22	0.20	0.22	0.16	0.46	0.42	0.45	0.43	0.23	0.17	0.27	0.22
2^{13}	1.46	1.54	0.22	0.18	0.22	0.16	0.48	0.49	0.49	0.48	0.23	0.21	0.25	0.21
2^{14}	1.53	1.61	0.22	0.17	0.22	0.18	0.43	0.42	0.45	0.41	0.24	0.22	0.24	0.22
2^{15}	1.61	1.76	0.25	0.21	0.24	0.19	0.50	0.49	0.50	0.49	0.24	0.18	0.23	0.21
2^{16}	1.79	1.92	0.28	0.25	0.29	0.24	0.55	0.51	0.56	0.55	0.30	0.23	0.32	0.24

Benchmark I: DENSITY, VARIATION

DS10 unordered_set<vector<string>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
$_2^6$	73.00	73.20	9.41	9.39	9.34	8.97	41.70	39.70	41.10	39.30	11.20	10.40	11.20	10.30
$_2^7$	74.70	75.30	9.32	9.34	9.24	8.87	46.20	43.70	45.30	44.20	12.70	10.60	11.40	10.80
$_2^8$	83.10	85.40	18.00	17.30	16.90	17.20	62.20	58.90	61.90	57.60	23.20	22.30	23.10	22.40
$_2^9$	91.40	94.90	19.00	19.00	18.80	18.60	65.00	59.90	64.40	58.90	24.30	22.60	24.10	22.60
$_2^{10}$	98.20	101.0	19.20	18.90	19.10	18.60	66.50	59.70	65.40	59.10	24.80	22.60	24.60	22.70
$_2^{11}$	99.50	101.0	19.00	19.10	19.30	18.40	66.90	59.50	66.10	58.70	24.90	22.70	25.10	22.50
$_2^{12}$	102.0	105.0	19.40	19.00	19.20	18.80	67.00	58.90	65.80	59.40	25.30	22.60	25.10	22.70
$_2^{13}$	103.0	104.0	19.00	19.20	19.40	18.40	66.70	59.20	66.20	58.20	25.30	22.90	25.50	22.60
$_2^{14}$	95.80	97.20	19.80	20.00	20.30	19.30	62.80	55.60	61.90	54.30	29.20	26.80	29.60	26.50
$_2^{15}$	87.10	89.80	24.00	23.70	24.00	23.50	64.30	55.00	61.90	54.90	33.60	30.80	33.50	31.00
$_2^{16}$	77.10	78.20	25.60	25.70	26.00	25.10	63.90	55.50	63.30	54.50	35.30	33.00	35.70	32.60

Benchmark I: DENSITY, VARIATION

DS11 `unordered_set<unordered_set<int>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
$_2^6$	28.70	29.10	3.06	2.75	3.55	3.14	4.96	4.40	5.41	4.84	3.24	2.73	3.73	3.15
$_2^7$	29.10	29.00	3.02	2.71	3.47	3.06	5.03	4.52	5.49	4.89	3.23	2.66	3.68	3.08
$_2^8$	28.80	29.10	3.00	2.68	3.45	3.04	5.18	4.55	5.57	4.98	3.24	2.66	3.65	3.06
$_2^9$	31.80	32.30	2.99	2.64	3.43	2.98	5.12	4.54	5.55	4.95	3.22	2.60	3.65	2.99
$_2^{10}$	46.50	47.10	2.95	2.65	3.40	2.99	5.13	4.57	5.62	4.96	3.21	2.58	3.62	2.97
$_2^{11}$	53.30	53.50	2.94	2.64	3.43	2.96	5.58	4.84	5.75	5.39	3.20	2.63	3.67	3.01
$_2^{12}$	54.60	55.00	3.02	2.66	3.43	2.98	6.47	5.94	6.99	6.28	3.83	3.00	4.21	3.38
$_2^{13}$	56.50	56.50	3.38	2.98	3.72	3.26	7.04	6.04	7.48	6.45	4.15	3.03	4.58	3.39
$_2^{14}$	52.10	52.20	3.50	2.99	3.88	3.25	7.35	6.07	7.83	6.59	4.33	3.05	4.76	3.38
$_2^{15}$	45.70	46.20	3.62	2.99	3.95	3.27	7.70	6.39	8.11	6.83	4.43	3.06	4.81	3.44
$_2^{16}$	39.30	39.30	3.72	3.05	4.03	3.31	7.57	6.30	8.09	6.61	4.52	3.10	4.92	3.45

Benchmark I: DENSITY, VARIATION

DS12 `unordered_set<unordered_set<string>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	121.0	125.0	25.9	23.7	26.1	23.9	56.3	54.5	56.7	54.7	27.4	25.8	27.8	26.0
2^7	141.0	145.0	26.4	24.3	25.6	23.4	62.1	59.6	62.5	60.0	27.9	25.8	28.3	25.8
2^8	165.0	173.0	31.5	27.3	32.2	27.7	77.4	73.7	77.8	74.2	34.8	31.9	35.6	32.2
2^9	171.0	178.0	35.9	27.6	34.4	27.8	80.0	73.7	79.7	74.6	35.7	32.0	36.5	31.9
2^{10}	177.0	182.0	38.7	28.6	35.6	27.9	81.1	74.3	81.3	74.3	36.7	31.8	37.1	32.0
2^{11}	177.0	183.0	38.2	27.6	36.2	27.7	81.3	74.3	82.2	74.1	37.0	32.0	37.8	31.9
2^{12}	179.0	186.0	39.1	27.7	36.5	28.0	81.6	73.5	81.5	74.1	37.3	31.8	37.9	32.1
2^{13}	165.0	169.0	39.0	27.8	36.7	27.8	81.3	73.9	82.8	73.5	37.3	32.1	38.3	32.1
2^{14}	153.0	156.0	40.9	29.6	38.7	29.6	81.5	74.1	82.4	73.7	44.4	39.2	45.4	39.1
2^{15}	122.0	131.0	47.6	35.7	44.8	36.1	85.7	75.2	83.9	75.4	51.0	45.1	51.4	45.5
2^{16}	100.0	111.0	51.4	40.4	48.0	38.8	85.1	75.5	86.2	75.6	53.6	48.4	54.6	48.2

Benchmark I: DENSITY, VARIATION

Observation:

Global Allocators Are *Always* Inferior!

Question:

Which Local Allocation Strategy is best?

Plan:

Look Again, Ignoring Global Strategies.

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS1

vector<int>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	1.2	1.9	0.3	0.4	0.4	0.4	0.8	1.0	0.9	1.1	0.6	0.7	0.8	0.7
2^7	0.9	1.6	0.3	0.4	0.4	0.4	0.5	0.7	0.6	0.7	0.5	0.5	0.6	0.5
2^8	0.8	1.0	0.2	0.4	0.4	0.3	0.4	0.6	0.5	0.6	0.3	0.5	0.5	0.5
2^9	0.8	1.0	0.2	0.4	0.4	0.4	0.3	0.5	0.5	0.5	0.3	0.4	0.4	0.4
2^{10}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{11}	0.7	0.9	0.2	0.3	0.4	0.3	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{12}	0.7	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{13}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{14}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{15}	0.8	0.9	0.2	0.3	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4
2^{16}	0.8	0.9	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4	0.2	0.4	0.4	0.4

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS2 vector<string>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	68.9	67.3	12.9	12.8	13.3	12.9	18.1	17.8	18.2	17.7	15.5	14.8	15.6	14.8
2^7	68.8	68.2	12.8	12.9	13.2	12.9	20.6	20.2	20.6	20.4	15.1	14.3	15.0	14.4
2^8	70.8	68.9	13.2	12.8	13.6	12.9	30.8	30.4	30.7	30.3	15.3	14.6	15.4	14.7
2^9	73.1	71.2	13.5	13.5	13.9	13.5	38.2	37.6	38.0	37.3	15.9	15.1	15.9	15.1
2^{10}	75.4	74.3	13.6	13.5	14.0	13.7	41.1	40.3	41.6	40.9	16.0	15.1	15.9	15.0
2^{11}	76.9	74.5	13.6	13.5	14.1	13.6	43.9	43.2	43.7	42.6	16.0	15.0	16.0	15.1
2^{12}	76.1	74.8	13.7	13.5	14.0	13.6	41.2	38.8	40.6	39.4	15.9	14.9	15.8	15.0
2^{13}	76.1	74.8	13.6	13.6	14.0	13.6	41.4	39.2	41.3	39.9	15.9	15.0	15.8	14.9
2^{14}	78.3	76.5	13.6	13.6	14.0	13.6	45.8	42.3	44.8	44.0	16.1	15.2	16.2	15.4
2^{15}	90.4	91.0	20.2	20.1	20.5	20.1	62.2	58.7	62.2	58.2	26.0	25.0	26.0	24.9
2^{16}	103.0	103.0	21.5	21.3	21.8	21.3	66.5	59.2	65.1	59.9	27.0	25.3	27.1	25.2

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS3 `unordered_set<int>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	10.20	11.00	5.08	4.88	5.62	5.34	7.16	7.12	7.50	7.20	6.19	5.73	6.40	5.81
2^7	12.50	13.30	5.04	4.81	5.68	5.24	6.37	6.22	6.71	6.31	5.80	5.46	6.08	5.50
2^8	15.80	16.40	4.99	4.79	5.54	5.22	5.95	5.81	6.21	5.92	5.65	5.32	5.82	5.40
2^9	18.30	19.00	5.01	4.80	5.53	5.18	5.78	5.56	6.01	5.70	5.56	5.20	5.76	5.21
2^{10}	21.40	22.30	4.99	4.83	5.55	5.20	5.72	5.46	5.95	5.55	5.52	5.27	5.68	5.24
2^{11}	25.50	26.10	4.98	4.81	5.56	5.16	5.67	5.44	5.86	5.65	5.53	5.23	5.69	5.26
2^{12}	27.10	28.00	5.02	4.81	5.55	5.20	6.42	6.10	6.57	6.25	5.51	5.12	5.68	5.27
2^{13}	27.90	28.80	5.03	4.81	5.59	5.21	7.34	6.91	7.46	7.03	5.61	5.16	5.71	5.24
2^{14}	28.50	29.00	5.03	4.80	5.58	5.26	7.03	6.59	7.18	6.68	5.64	5.19	5.80	5.34
2^{15}	28.30	29.20	5.03	4.78	5.56	5.28	7.11	6.65	7.20	6.83	5.68	5.17	5.78	5.24
2^{16}	31.60	31.80	5.02	4.76	5.60	5.22	6.79	6.37	6.93	6.46	5.68	5.17	5.79	5.24

244

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS4 unordered_set<string>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	103.0	120.0	52.2	51.9	52.4	51.2	58.4	57.6	59.7	58.9	55.1	54.1	56.9	55.3
2^7	103.0	122.0	52.5	52.1	52.9	51.8	63.3	61.9	64.4	63.8	55.3	54.0	56.8	55.7
2^8	109.0	128.0	53.6	53.0	53.7	52.6	76.3	74.7	77.4	75.9	56.5	54.9	57.9	56.7
2^9	113.0	134.0	54.5	53.4	54.9	53.0	83.1	81.7	82.8	81.4	57.3	56.7	58.0	56.4
2^{10}	119.0	143.0	56.6	54.9	56.9	54.6	87.6	85.9	88.1	86.5	58.8	56.9	59.2	57.3
2^{11}	122.0	144.0	57.0	55.3	57.7	54.9	90.7	89.2	90.7	88.4	59.4	57.6	60.0	57.8
2^{12}	122.0	146.0	57.9	55.9	58.4	55.7	93.2	90.7	93.2	90.7	60.5	58.3	60.7	58.4
2^{13}	124.0	148.0	58.2	56.3	58.5	55.9	95.1	91.5	94.3	92.0	60.5	58.2	60.7	58.7
2^{14}	139.0	166.0	59.1	57.3	59.6	56.8	98.5	94.1	97.8	95.8	61.8	59.6	62.2	60.0
2^{15}	176.0	211.0	66.0	62.7	66.2	62.4	121.0	115.0	122.0	115.0	76.5	73.3	76.8	74.0
2^{16}	196.0	232.0	78.5	72.0	79.1	71.0	137.0	127.0	136.0	127.0	87.1	82.4	87.8	82.9

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS5

vector<vector<int>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	0.97	1.00	0.19	0.13	0.20	0.17	0.24	0.20	0.20	0.21	0.21	0.19	0.20	0.21
2^7	0.96	0.96	0.22	0.16	0.18	0.14	0.21	0.20	0.19	0.20	0.16	0.20	0.21	0.19
2^8	0.99	1.00	0.19	0.13	0.18	0.17	0.27	0.30	0.27	0.29	0.19	0.19	0.20	0.21
2^9	0.99	1.02	0.19	0.13	0.18	0.14	0.36	0.33	0.33	0.36	0.19	0.15	0.20	0.20
2^{10}	1.01	1.04	0.19	0.18	0.19	0.14	0.37	0.36	0.36	0.38	0.22	0.19	0.20	0.22
2^{11}	1.02	1.05	0.19	0.13	0.19	0.14	0.36	0.35	0.36	0.36	0.20	0.15	0.20	0.22
2^{12}	1.03	1.05	0.19	0.19	0.22	0.18	0.33	0.36	0.32	0.32	0.20	0.21	0.20	0.19
2^{13}	1.02	1.05	0.19	0.13	0.22	0.19	0.35	0.35	0.34	0.33	0.20	0.21	0.22	0.19
2^{14}	1.05	1.10	0.19	0.17	0.19	0.16	0.38	0.36	0.38	0.37	0.17	0.19	0.20	0.19
2^{15}	1.13	1.18	0.22	0.19	0.19	0.16	0.50	0.45	0.47	0.45	0.21	0.21	0.17	0.18
2^{16}	1.29	1.32	0.22	0.19	0.20	0.17	0.54	0.47	0.52	0.50	0.22	0.21	0.22	0.21

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS6

vector<vector<string>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	72.60	72.70	9.06	9.06	9.36	8.98	41.70	40.00	41.20	39.20	11.20	10.30	11.20	10.30
2^7	74.90	76.00	8.92	8.98	9.29	8.89	46.50	44.80	46.00	43.00	11.40	11.00	12.70	10.30
2^8	85.50	85.20	17.10	17.40	17.30	16.90	62.90	58.40	61.30	58.40	22.80	22.50	23.30	22.00
2^9	96.40	96.30	18.40	18.70	19.00	18.40	66.20	59.00	64.70	59.30	24.20	22.70	24.50	22.30
2^{10}	102.0	102.0	18.70	18.60	19.10	18.60	67.00	59.60	65.90	59.00	24.80	22.50	24.80	22.50
2^{11}	102.0	101.0	18.40	18.70	19.20	18.20	62.40	55.00	61.30	54.20	24.80	22.60	25.10	22.30
2^{12}	104.0	103.0	18.50	18.70	19.40	18.30	61.60	54.20	60.50	53.40	24.90	22.70	25.10	22.30
2^{13}	103.0	104.0	18.80	18.40	19.00	18.60	61.80	53.40	59.90	53.50	25.30	22.60	25.10	22.60
2^{14}	97.10	96.30	19.20	19.60	20.10	19.20	60.60	53.70	60.20	52.90	29.00	26.70	29.20	26.30
2^{15}	88.10	88.70	23.40	23.20	23.70	23.40	62.60	54.40	60.90	53.90	33.40	30.60	33.20	30.70
2^{16}	76.70	76.70	25.00	25.30	25.80	25.00	63.40	54.80	62.90	54.30	35.00	32.80	35.50	32.40

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS7

vector<unordered_set<int>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	28.80	28.70	2.97	2.69	3.43	2.98	4.89	4.37	5.33	4.73	3.21	2.65	3.64	3.05
2^7	28.30	28.50	2.97	2.66	3.36	2.95	4.99	4.44	5.43	4.91	3.20	2.62	3.61	2.97
2^8	28.20	28.10	2.94	2.62	3.33	2.92	5.02	4.53	5.53	4.97	3.23	2.60	3.60	3.01
2^9	31.80	31.70	2.92	2.61	3.33	2.93	5.08	4.54	5.52	4.92	3.16	2.58	3.58	2.96
2^{10}	46.60	47.20	2.92	2.61	3.33	2.89	5.07	4.49	5.48	4.93	3.15	2.58	3.57	2.98
2^{11}	54.30	54.10	2.92	2.61	3.33	2.89	5.63	4.75	5.88	5.37	3.16	2.60	3.61	2.98
2^{12}	54.70	54.80	2.96	2.66	3.34	2.91	6.90	5.79	7.28	6.23	4.15	3.05	4.58	3.40
2^{13}	55.10	56.00	3.51	2.95	3.77	3.21	7.01	6.03	7.47	6.35	4.27	3.08	4.65	3.48
2^{14}	51.00	50.90	3.53	2.99	3.81	3.25	7.08	6.00	7.47	6.46	4.29	3.14	4.71	3.47
2^{15}	44.80	45.40	3.58	3.01	3.83	3.26	7.07	6.04	7.55	6.52	4.35	3.14	4.75	3.53
2^{16}	38.20	38.20	3.58	3.06	3.86	3.30	7.14	6.11	7.58	6.47	4.37	3.18	4.80	3.54

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS8

vector<unordered_set<string>>

Size	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	114.0	116.0	26.0	23.8	26.3	24.0	56.2	54.7	56.9	54.6	27.5	25.8	27.9	26.0
2^7	123.0	130.0	26.5	24.4	25.7	23.5	62.7	60.1	62.7	60.5	27.5	26.3	28.2	26.1
2^8	162.0	171.0	31.7	27.3	32.2	27.8	78.0	74.2	79.2	73.9	35.0	32.0	35.5	32.5
2^9	175.0	181.0	36.8	28.0	38.1	28.0	81.7	74.1	81.2	74.9	36.3	32.1	37.2	32.1
2^{10}	176.0	183.0	40.0	28.9	37.4	28.2	82.1	74.5	82.1	74.7	36.9	32.0	37.4	32.2
2^{11}	176.0	183.0	39.3	28.0	37.3	28.0	81.4	74.4	82.0	74.3	36.9	32.1	37.8	32.1
2^{12}	179.0	185.0	39.4	28.0	37.1	28.0	81.8	74.1	81.6	74.4	37.0	32.0	37.8	32.2
2^{13}	173.0	178.0	39.6	27.9	36.9	28.2	81.8	73.6	81.5	74.3	37.2	32.0	37.8	32.4
2^{14}	157.0	160.0	41.0	29.9	38.8	29.9	81.5	74.1	82.2	74.0	44.0	39.3	45.1	39.2
2^{15}	122.0	131.0	47.6	35.8	44.8	36.2	85.2	75.5	83.7	76.1	50.5	45.2	51.0	45.5
2^{16}	95.4	106.0	51.4	40.5	48.1	38.9	84.8	76.2	88.7	75.9	53.1	48.5	54.8	48.2

3. Analyzing the Benchmark Data

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS9

unordered_set<vector<int>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	0.97	0.94	0.23	0.19	0.24	0.21	0.26	0.27	0.30	0.26	0.25	0.26	0.25	0.24
2^7	1.40	1.43	0.22	0.21	0.22	0.19	0.24	0.26	0.25	0.27	0.24	0.26	0.24	0.24
2^8	1.35	1.39	0.25	0.22	0.24	0.23	0.30	0.35	0.34	0.33	0.24	0.23	0.25	0.24
2^9	1.29	1.32	0.22	0.18	0.22	0.17	0.37	0.38	0.37	0.36	0.23	0.22	0.19	0.22
2^{10}	1.32	1.38	0.24	0.22	0.22	0.19	0.41	0.39	0.42	0.39	0.23	0.24	0.23	0.22
2^{11}	1.34	1.36	0.23	0.21	0.22	0.17	0.44	0.42	0.43	0.41	0.23	0.23	0.25	0.22
2^{12}	1.34	1.41	0.22	0.20	0.22	0.16	0.46	0.42	0.45	0.43	0.23	0.17	0.27	0.22
2^{13}	1.46	1.54	0.22	0.18	0.22	0.16	0.48	0.49	0.49	0.48	0.23	0.21	0.25	0.21
2^{14}	1.53	1.61	0.22	0.17	0.22	0.18	0.43	0.42	0.45	0.41	0.24	0.22	0.24	0.22
2^{15}	1.61	1.76	0.25	0.21	0.24	0.19	0.50	0.49	0.50	0.49	0.24	0.18	0.23	0.21
2^{16}	1.79	1.92	0.28	0.25	0.29	0.24	0.55	0.51	0.56	0.55	0.30	0.23	0.32	0.24

250

780

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS10

unordered_set<vector<string>>

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	73.00	73.20	9.41	9.39	9.34	8.97	41.70	39.70	41.10	39.30	11.20	10.40	11.20	10.30
2^7	74.70	75.30	9.32	9.34	9.24	8.87	46.20	43.70	45.30	44.20	12.70	10.60	11.40	10.80
2^8	83.10	85.40	18.00	17.30	16.90	17.20	62.20	58.90	61.90	57.60	23.20	22.30	23.10	22.40
2^9	91.40	94.90	19.00	19.00	18.80	18.60	65.00	59.90	64.40	58.90	24.30	22.60	24.10	22.60
2^{10}	98.20	101.0	19.20	18.90	19.10	18.60	66.50	59.70	65.40	59.10	24.80	22.60	24.60	22.70
2^{11}	99.50	101.0	19.00	19.10	19.30	18.40	66.90	59.50	66.10	58.70	24.90	22.70	25.10	22.50
2^{12}	102.0	105.0	19.40	19.00	19.20	18.80	67.00	58.90	65.80	59.40	25.30	22.60	25.10	22.70
2^{13}	103.0	104.0	19.00	19.20	19.40	18.40	66.70	59.20	66.20	58.20	25.30	22.90	25.50	22.60
2^{14}	95.80	97.20	19.80	20.00	20.30	19.30	62.80	55.60	61.90	54.30	29.20	26.80	29.60	26.50
2^{15}	87.10	89.80	24.00	23.70	24.00	23.50	64.30	55.00	61.90	54.90	33.60	30.80	33.50	31.00
2^{16}	77.10	78.20	25.60	25.70	26.00	25.10	63.90	55.50	63.30	54.50	35.30	33.00	35.70	32.60

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS11 `unordered_set<unordered_set<int>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	28.70	29.10	3.06	2.75	3.55	3.14	4.96	4.40	5.41	4.84	3.24	2.73	3.73	3.15
2^7	29.10	29.00	3.02	2.71	3.47	3.06	5.03	4.52	5.49	4.89	3.23	2.66	3.68	3.08
2^8	28.80	29.10	3.00	2.68	3.45	3.04	5.18	4.55	5.57	4.98	3.24	2.66	3.65	3.06
2^9	31.80	32.30	2.99	2.64	3.43	2.98	5.12	4.54	5.55	4.95	3.22	2.60	3.65	2.99
2^{10}	46.50	47.10	2.95	2.65	3.40	2.99	5.13	4.57	5.62	4.96	3.21	2.58	3.62	2.97
2^{11}	53.30	53.50	2.94	2.64	3.43	2.96	5.58	4.84	5.75	5.39	3.20	2.63	3.67	3.01
2^{12}	54.60	55.00	3.02	2.66	3.43	2.98	6.47	5.94	6.99	6.28	3.83	3.00	4.21	3.38
2^{13}	56.50	56.50	3.38	2.98	3.72	3.26	7.04	6.04	7.48	6.45	4.15	3.03	4.58	3.39
2^{14}	52.10	52.20	3.50	2.99	3.88	3.25	7.35	6.07	7.83	6.59	4.33	3.05	4.76	3.38
2^{15}	45.70	46.20	3.62	2.99	3.95	3.27	7.70	6.39	8.11	6.83	4.43	3.06	4.81	3.44
2^{16}	39.30	39.30	3.72	3.05	4.03	3.31	7.57	6.30	8.09	6.61	4.52	3.10	4.92	3.45

252

Benchmark I: DENSITY, VARIATION

Poor Choice!

DS12 `unordered_set<unordered_set<string>>`

	Global		Monotonic				Multipool				Multipool<Mono>			
		Virtual		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)		(wink)	Virtual	(wink)
Size	AS1	AS2	AS3	AS4	AS5	AS6	AS7	AS8	AS9	AS10	AS11	AS12	AS13	AS14
2^6	121.0	125.0	25.9	23.7	26.1	23.9	56.3	54.5	56.7	54.7	27.4	25.8	27.8	26.0
2^7	141.0	145.0	26.4	24.3	25.6	23.4	62.1	59.6	62.5	60.0	27.9	25.8	28.3	25.8
2^8	165.0	173.0	31.5	27.3	32.2	27.7	77.4	73.7	77.8	74.2	34.8	31.9	35.6	32.2
2^9	171.0	178.0	35.9	27.6	34.4	27.8	80.0	73.7	79.7	74.6	35.7	32.0	36.5	31.9
2^{10}	177.0	182.0	38.7	28.6	35.6	27.9	81.1	74.3	81.3	74.3	36.7	31.8	37.1	32.0
2^{11}	177.0	183.0	38.2	27.6	36.2	27.7	81.3	74.3	82.2	74.1	37.0	32.0	37.8	31.9
2^{12}	179.0	186.0	39.1	27.7	36.5	28.0	81.6	73.5	81.5	74.1	37.3	31.8	37.9	32.1
2^{13}	165.0	169.0	39.0	27.8	36.7	27.8	81.3	73.9	82.8	73.5	37.3	32.1	38.3	32.1
2^{14}	153.0	156.0	40.9	29.6	38.7	29.6	81.5	74.1	82.4	73.7	44.4	39.2	45.4	39.1
2^{15}	122.0	131.0	47.6	35.7	44.8	36.1	85.7	75.2	83.9	75.4	51.0	45.1	51.4	45.5
2^{16}	100.0	111.0	51.4	40.4	48.0	38.8	85.1	75.5	86.2	75.6	53.6	48.4	54.6	48.2

Benchmark I: DENSITY, VARIATION

Dimensional Characterization

Allocation DENSITY (**D**):	?	(means: "being investigated")
"Chunk"-Size VARIATION (**V**):	?	
Access LOCALITY (**L**):	LOW	
Memory UTILIZATION (**U**):	1	
Allocator CONTENTION (**C**):	0	

Benchmark I: DENSITY, VARIATION

Take-Away Tips

- Global allocators are consistently inferior when dealing with systems having a short, focused duration irrespective of **C**ONTENTION (**C**).
- For systems having high **U**TILIZATION (**U**), such as those modeled in this benchmark, a *monotonic allocator* is always superior, especially when applied to a static buffer residing on the program stack (similar in effect to `alloca` in C).

Benchmark I: **D**ENSITY, **V**ARIATION

Questions and/or Discussion?

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

I. **Short Running:** Build Up, Use, Tear Down
   - Allocation DENSITY and VARIATION in Allocated Sizes

II. **Long Running:** Time-Multiplexed Subsystems
   - Access LOCALITY – both *Physica*l and *Temporal*

III. **Short Running:** Varying Memory Reusability
   - Memory UTILIZATION

IV. **Multithreaded:** Varying Numbers of Threads
   - Allocator CONTENTION

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

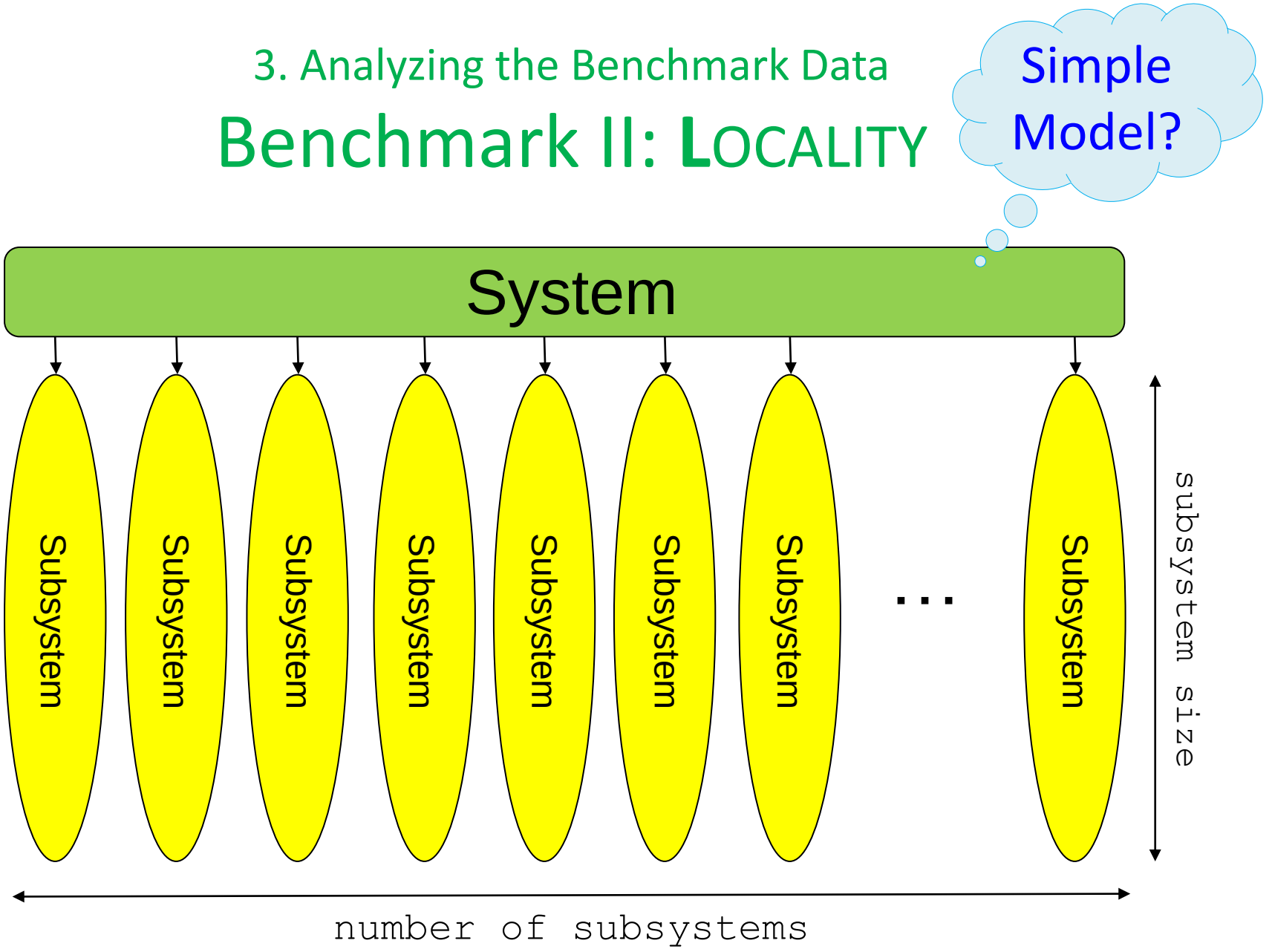


Benchmark II: LOCALITY

Considerations:

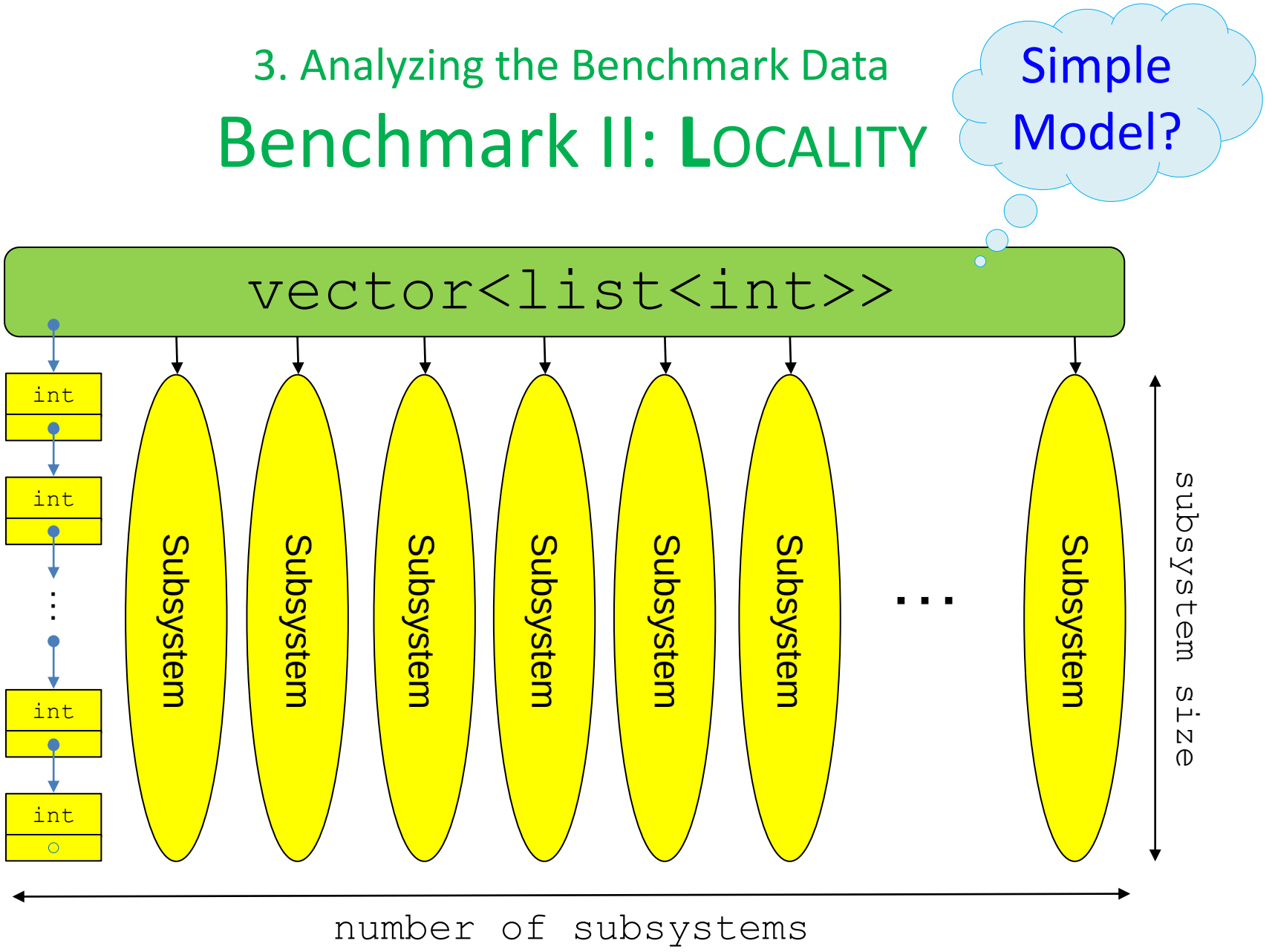
- Investigate access LOCALITY.
  - Observe its effects both *physically* and *temporally*.
- Simulate concurrent subsystems.
  - Vary both their *sizes* and *time slices* independently.
- Access LOCALITY *should* dominate results.
  - Time accessing data should be substantial (hours).
  - Time to set-up/tear-down should be negligible (seconds).

Benchmark II: LOCALITY



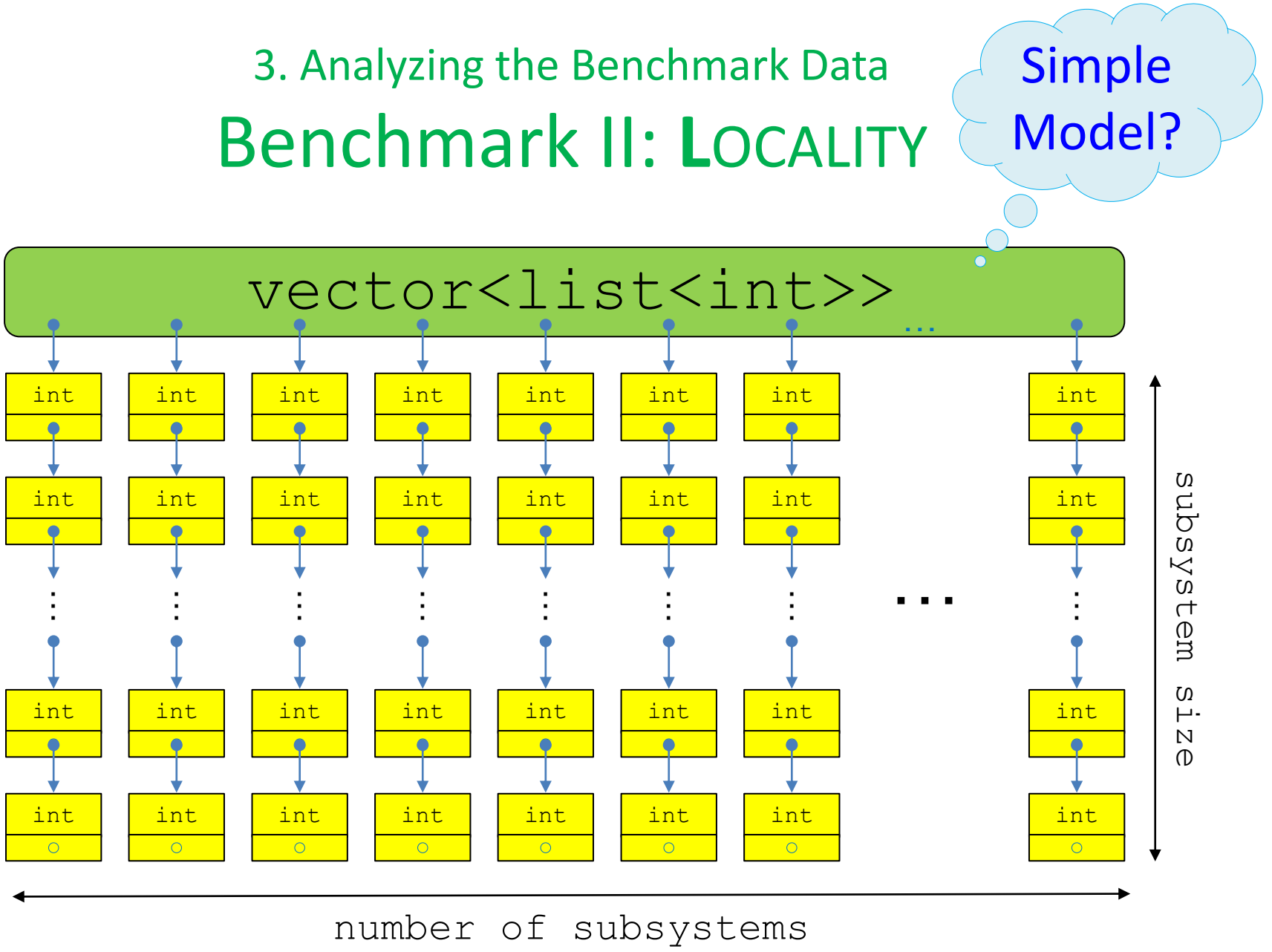
3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



Benchmark II: LOCALITY

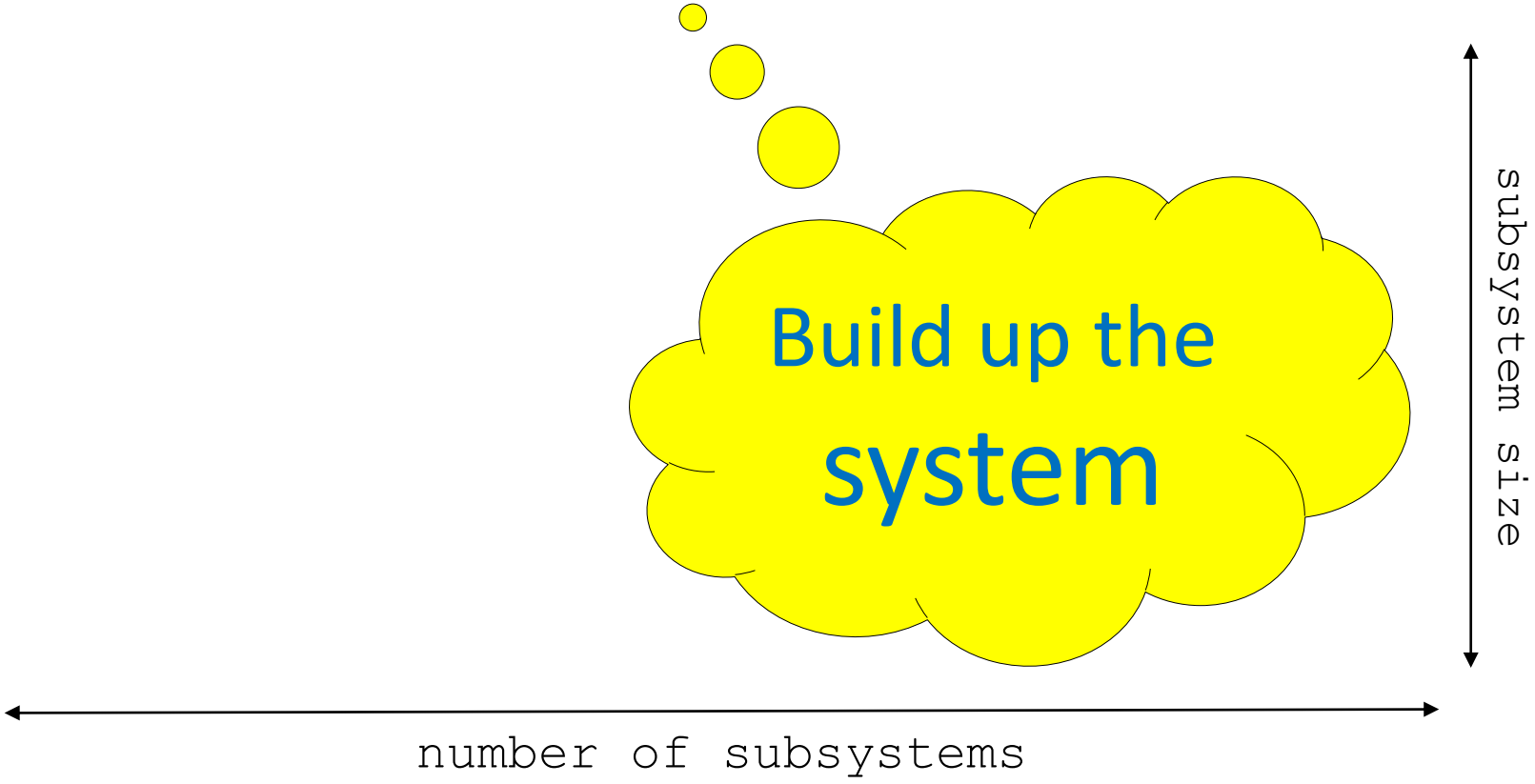
Creation Plan:

- Build up a data structure:

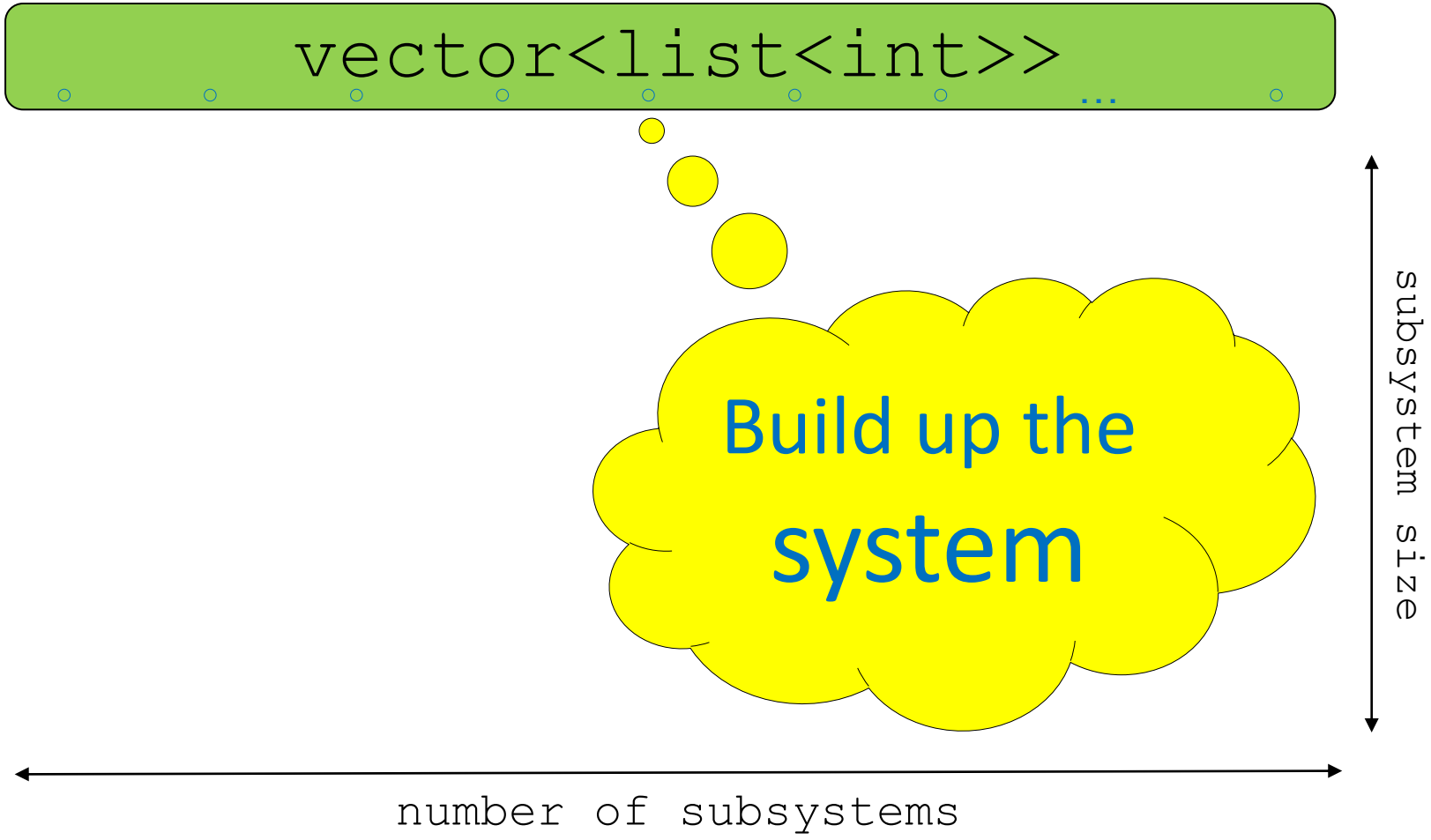
```
vector<list<int>> system
```

  - Having `system.size()` subsystems
  - With each i^{th} subsystem of size `system[0].size()`
- Tear down the data structure
  - Occurs automatically at the end of the program.
- The result is an initialized data structure, `system`.
  - Can also be used separately to measure Build-Up + Tear-Down run times.

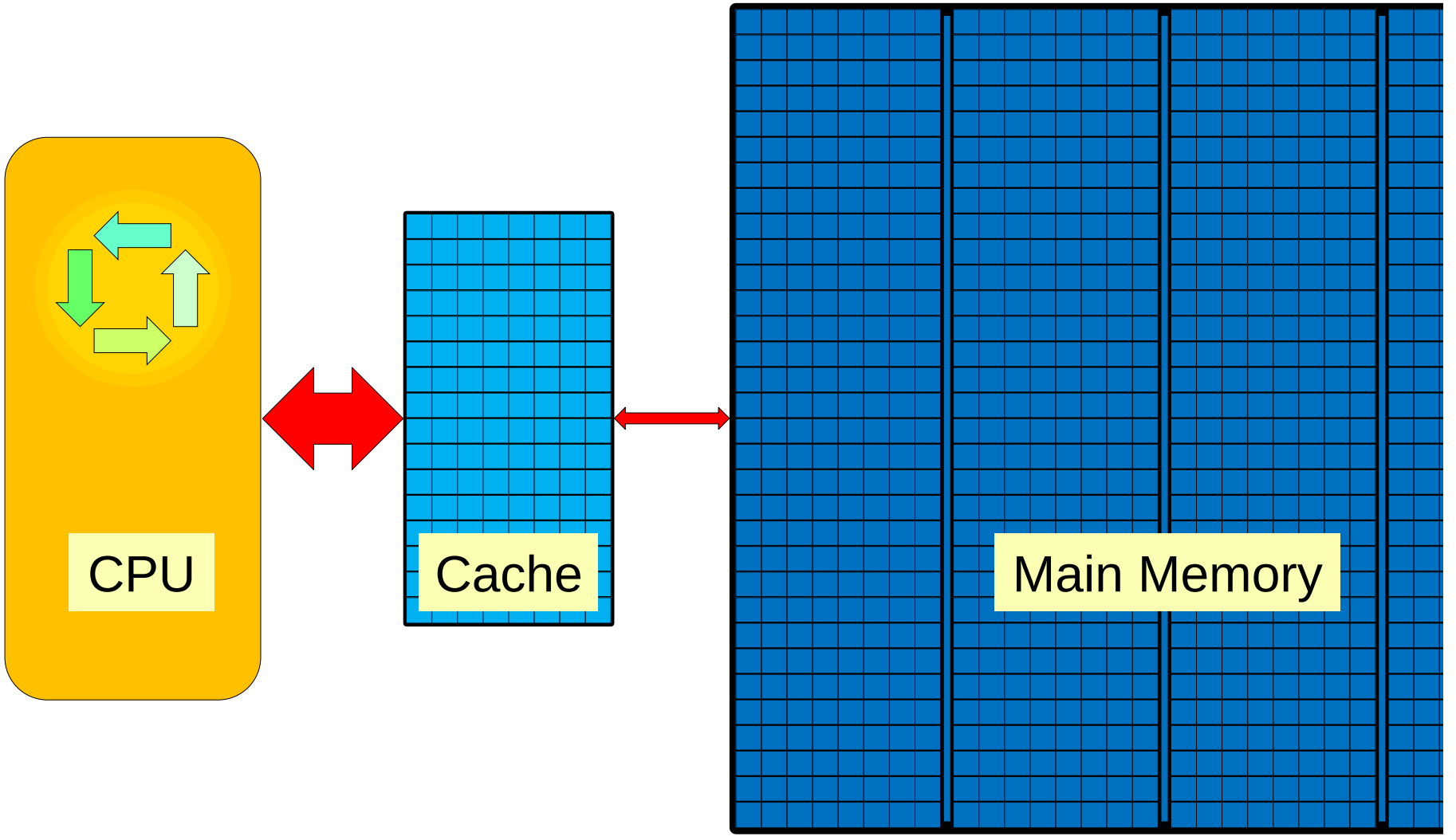
Benchmark II: LOCALITY



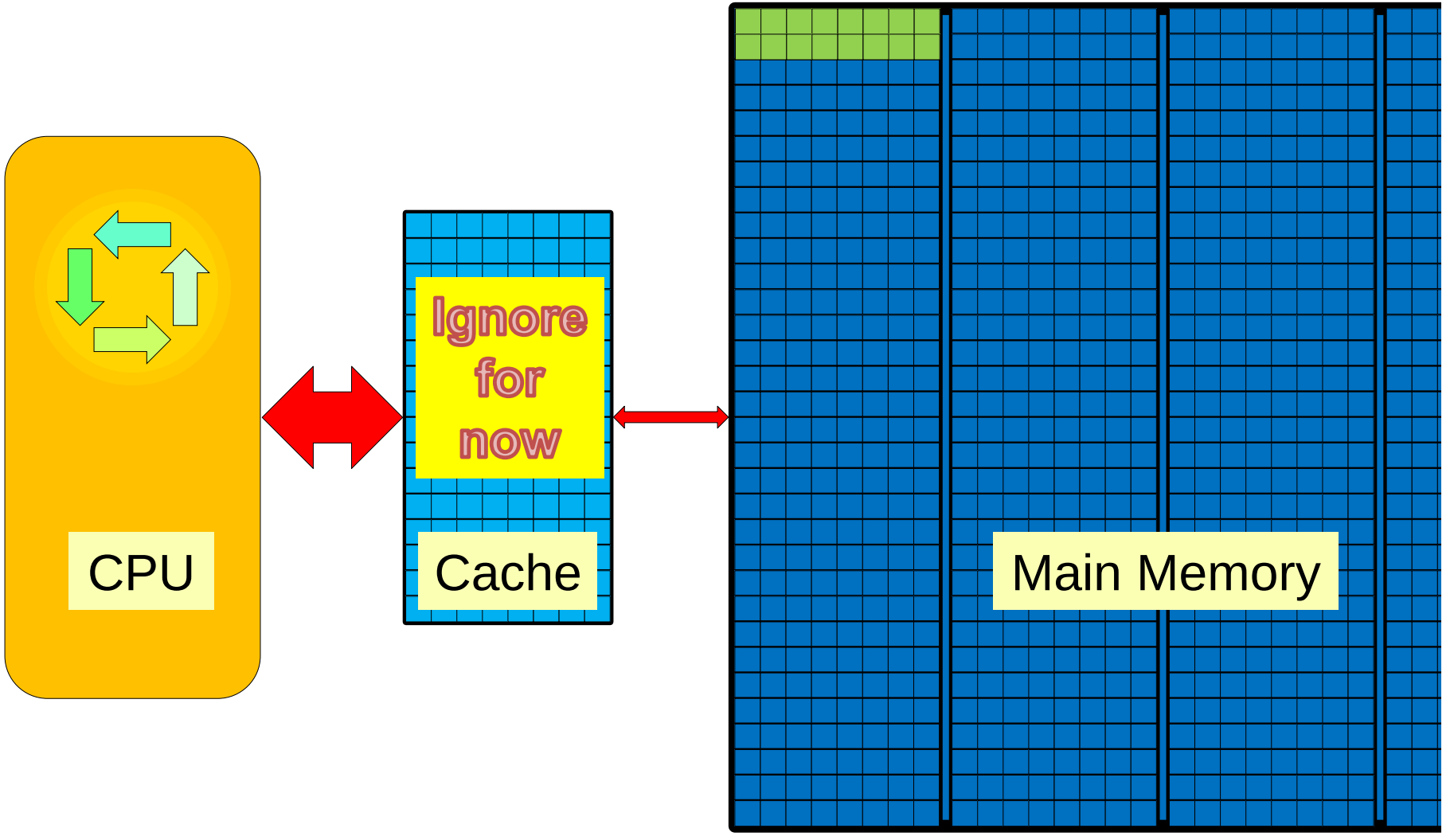
Benchmark II: LOCALITY



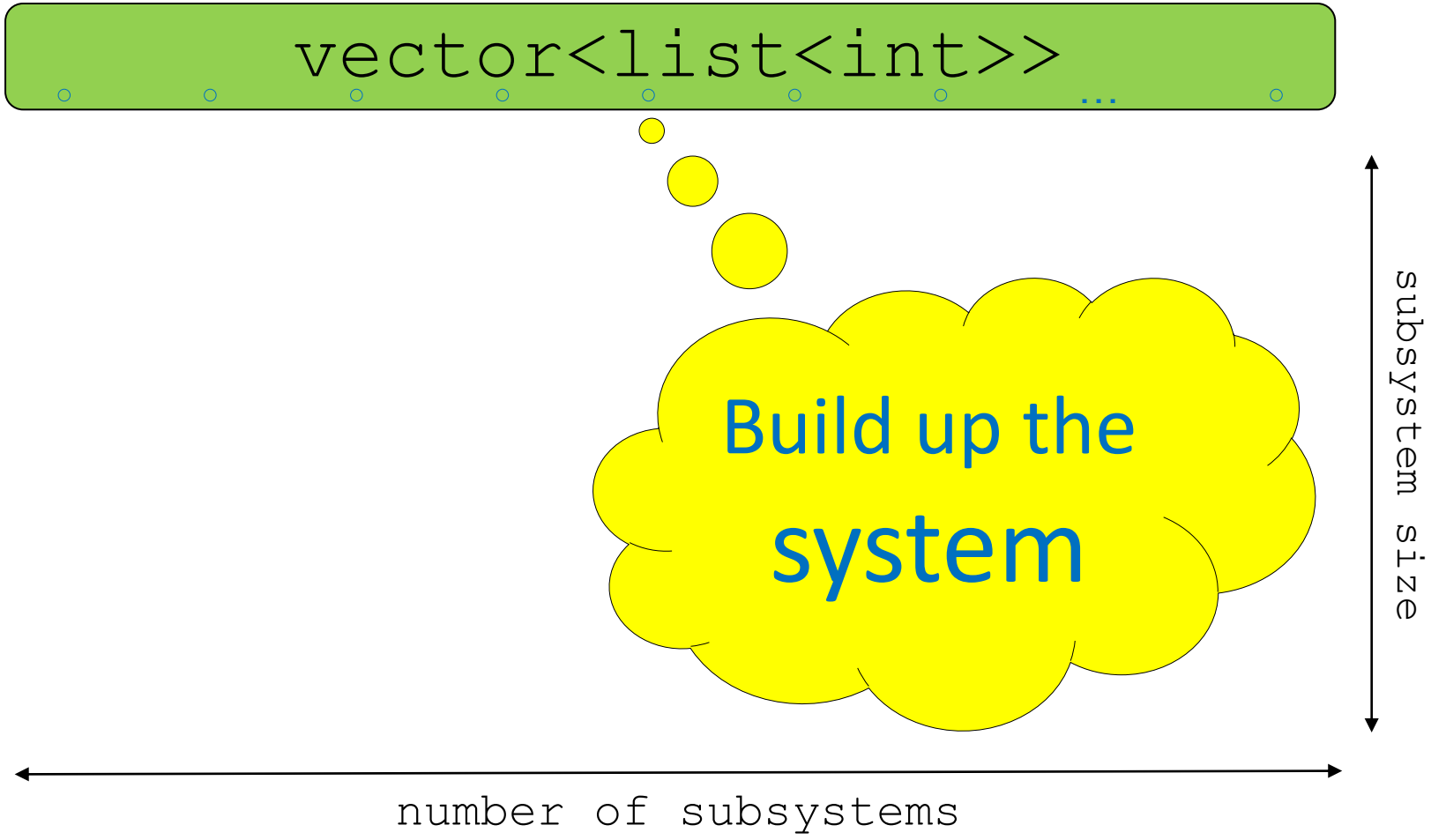
Benchmark II: LOCALITY



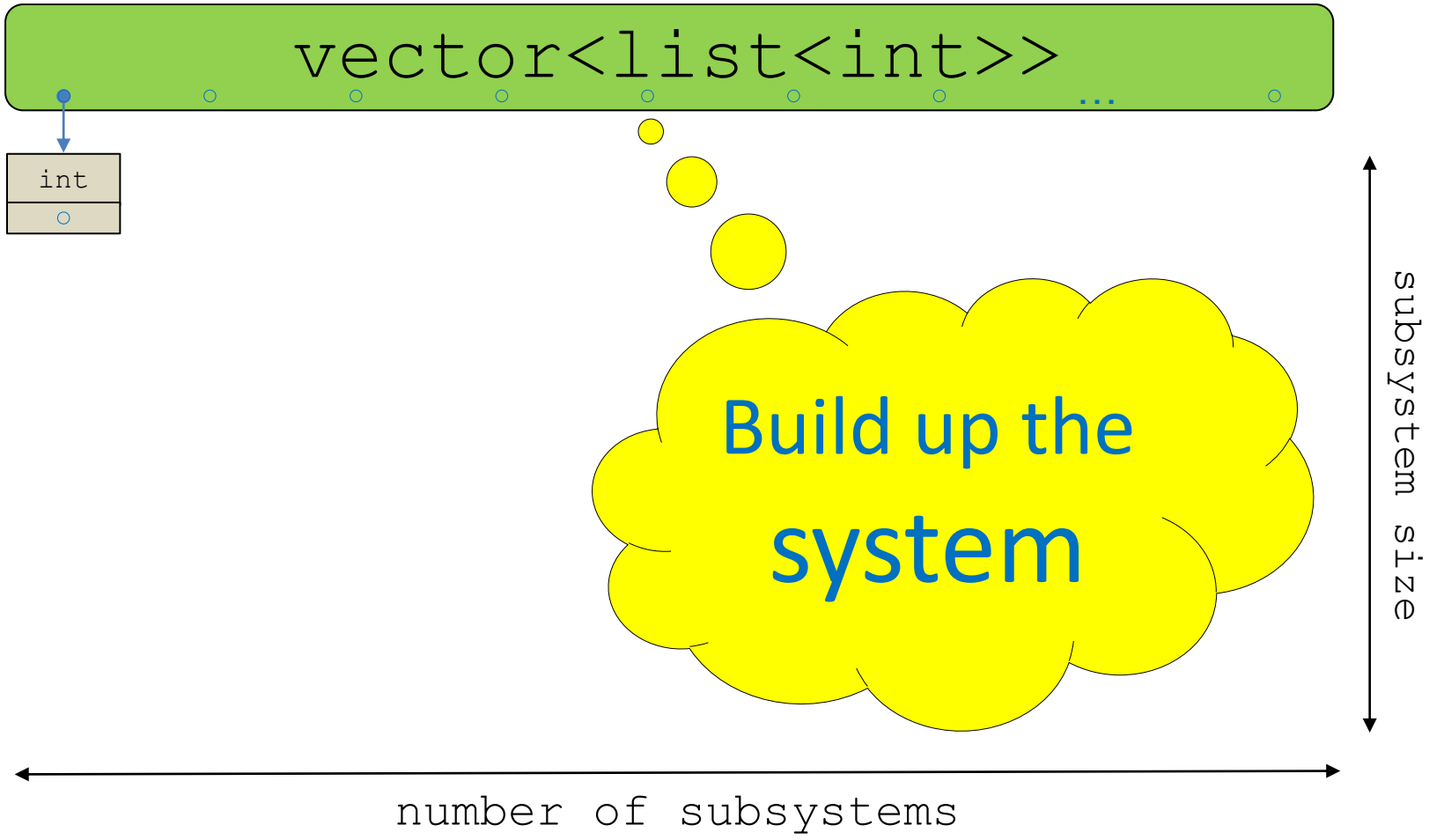
Benchmark II: LOCALITY



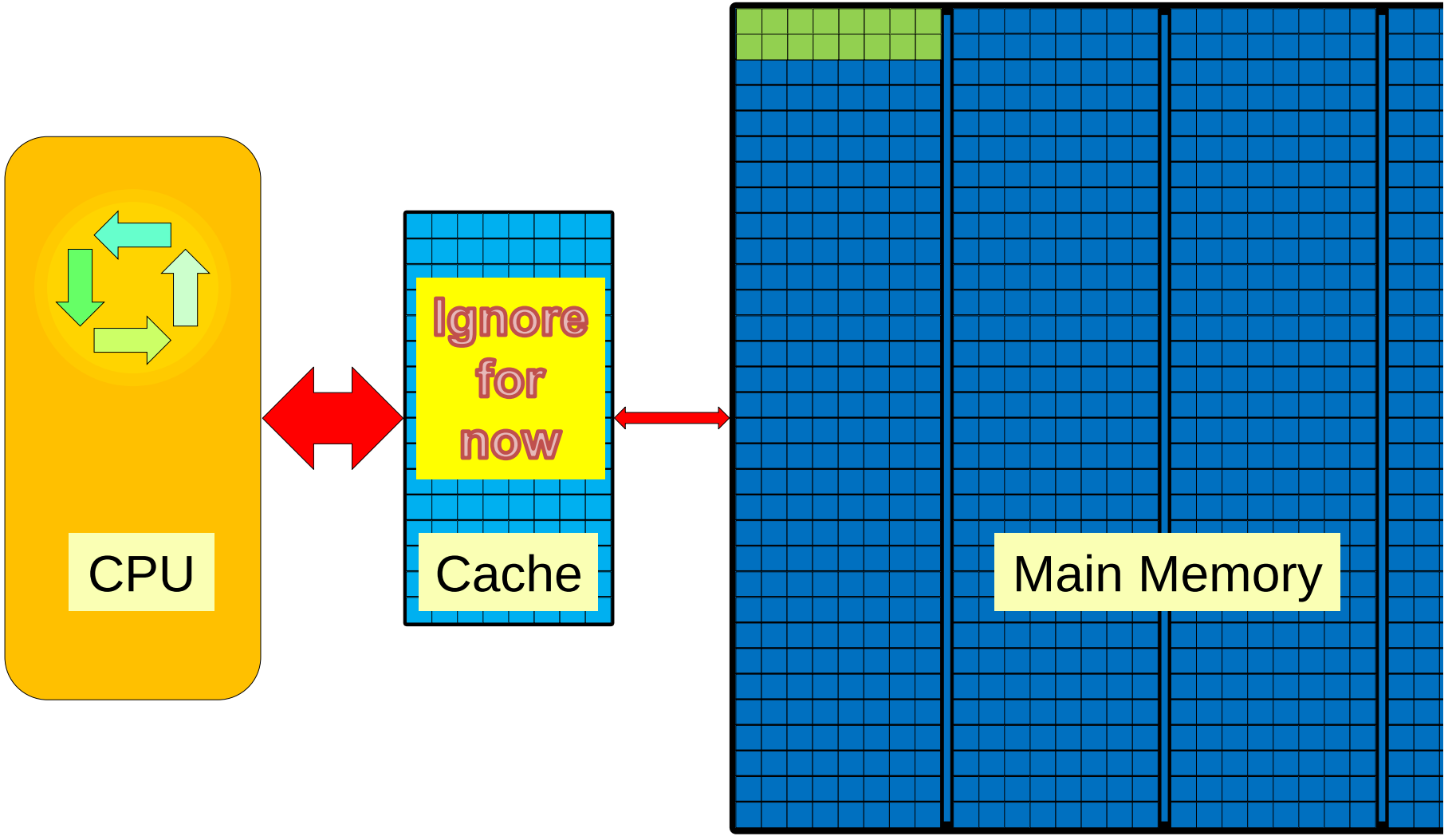
Benchmark II: LOCALITY



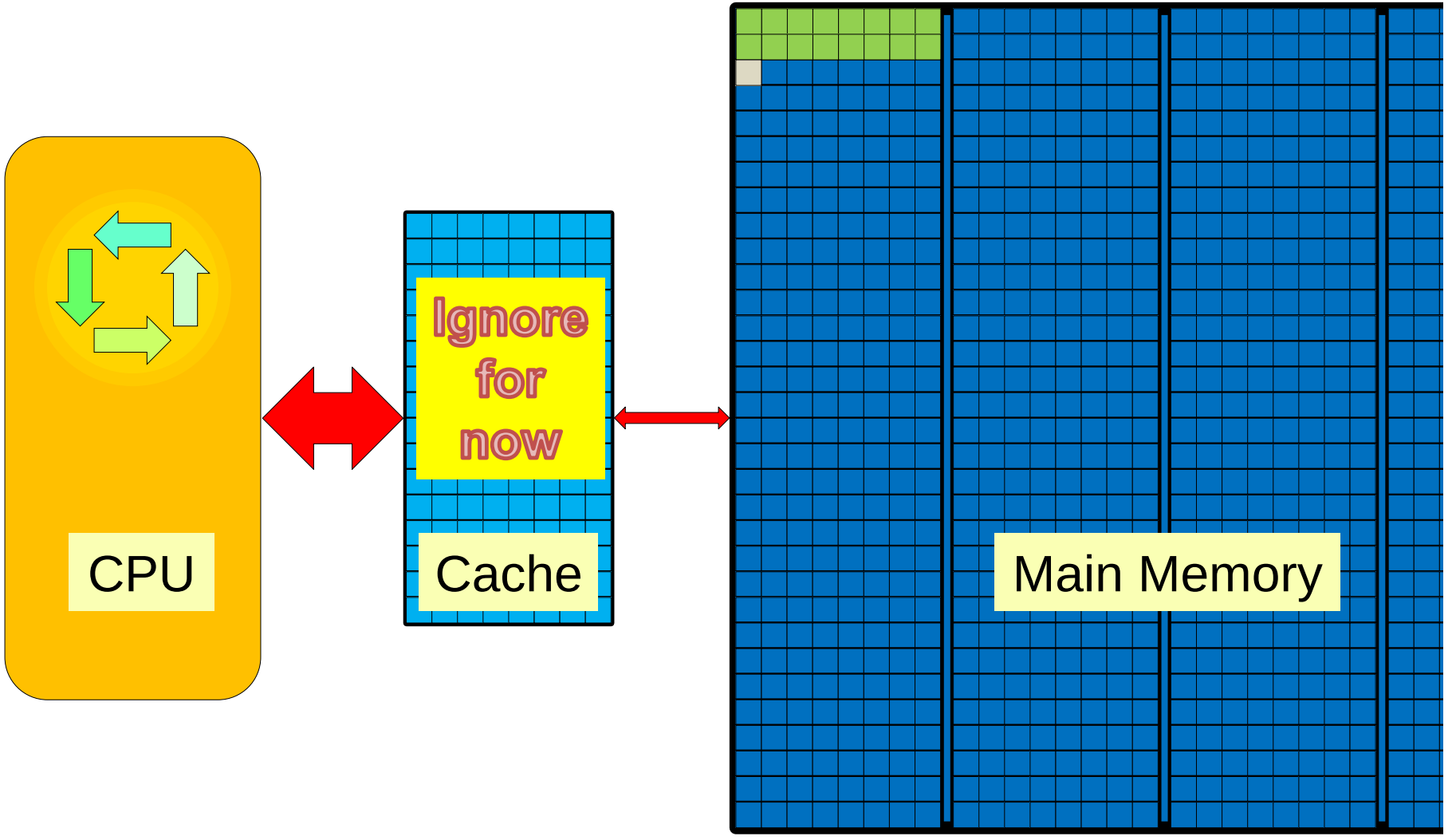
Benchmark II: LOCALITY



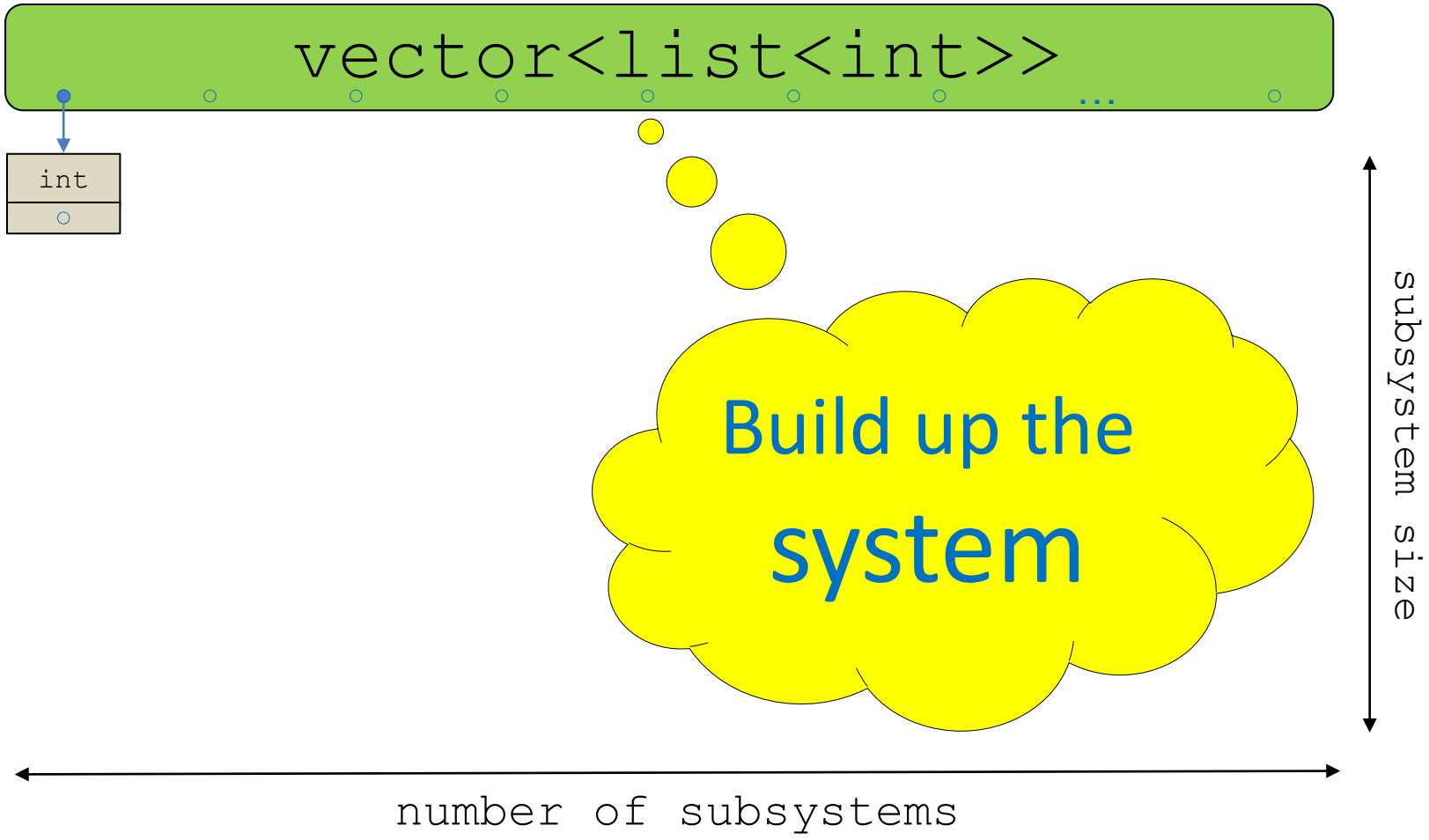
Benchmark II: LOCALITY



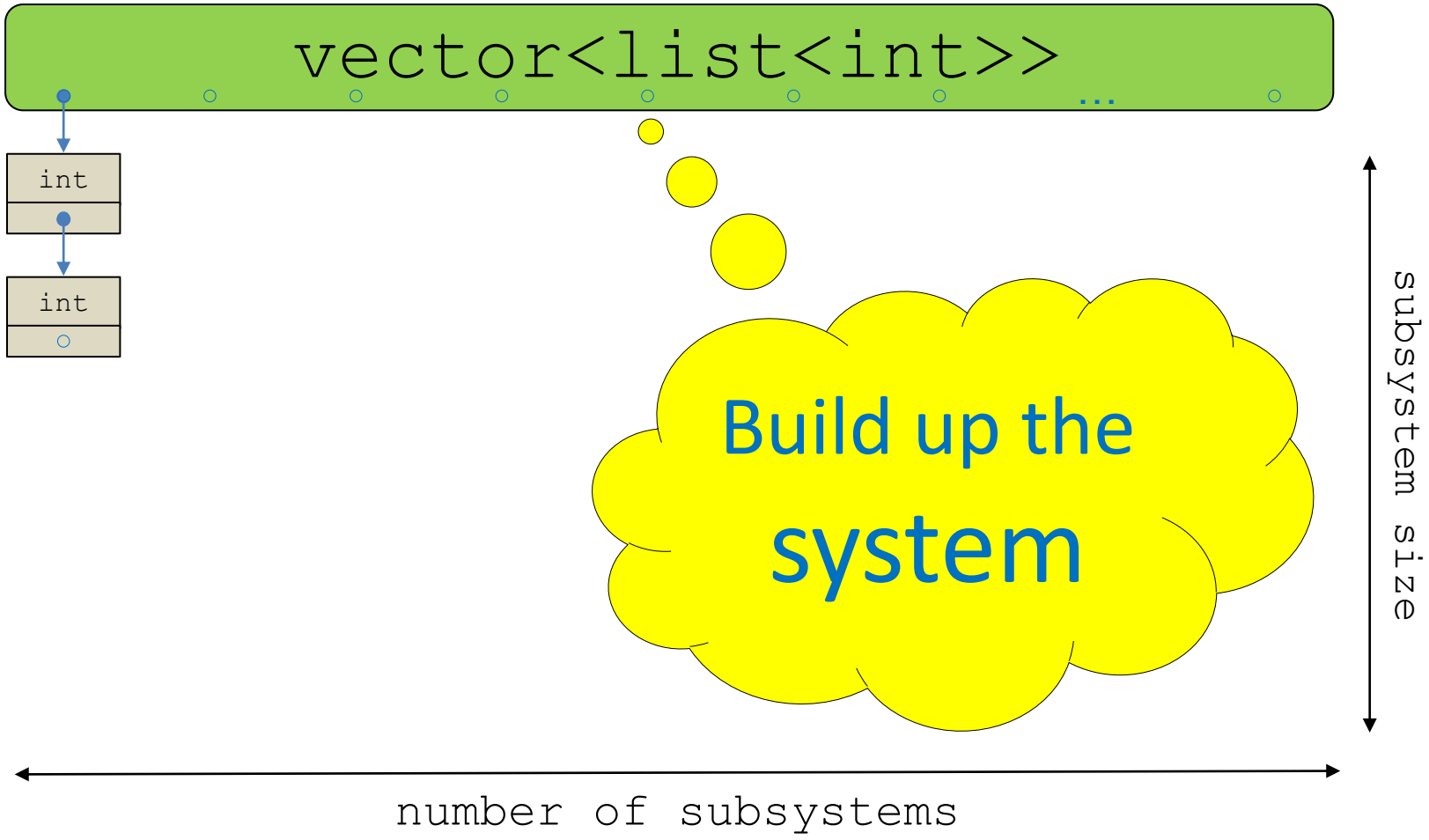
Benchmark II: LOCALITY



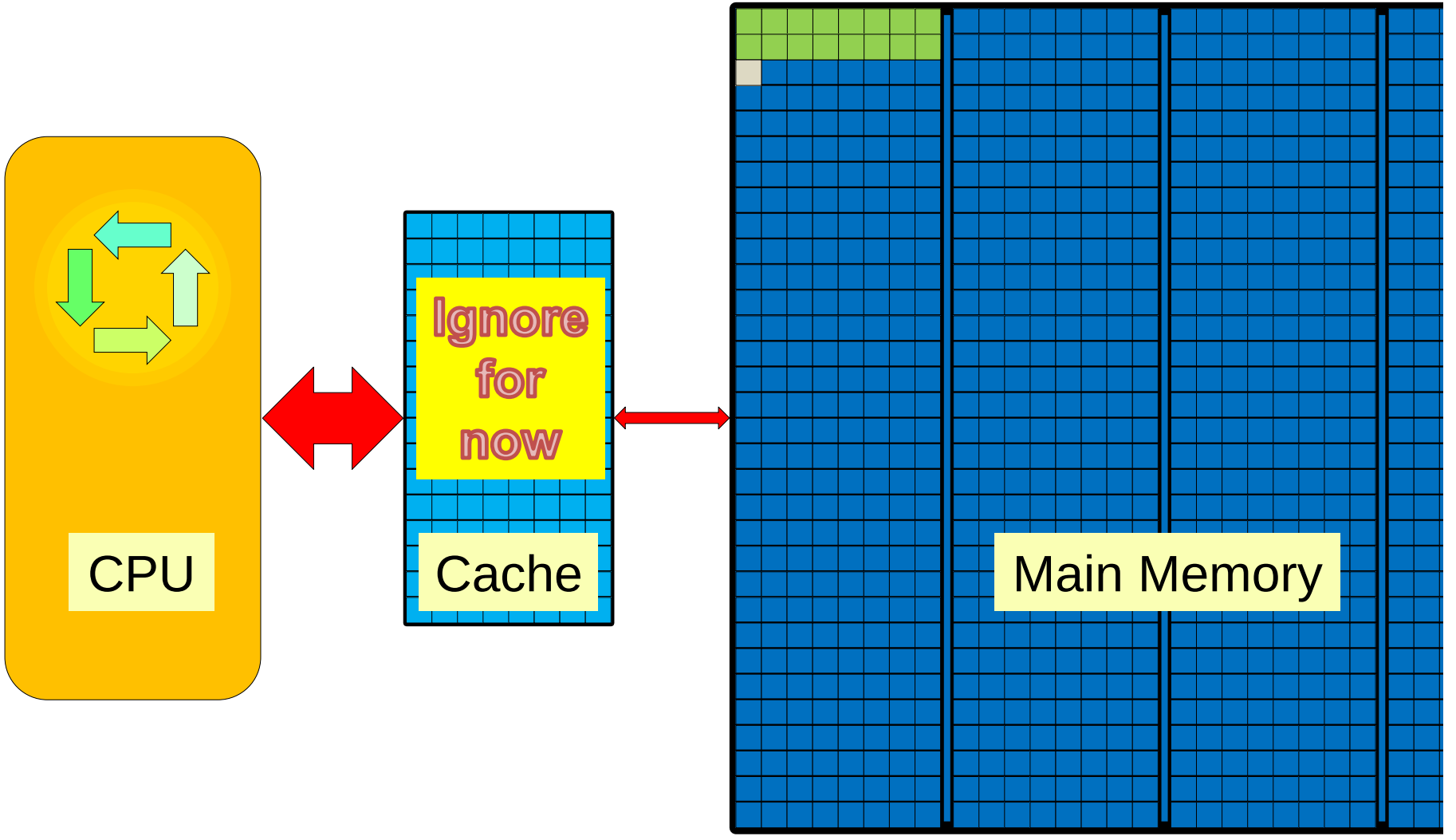
Benchmark II: LOCALITY



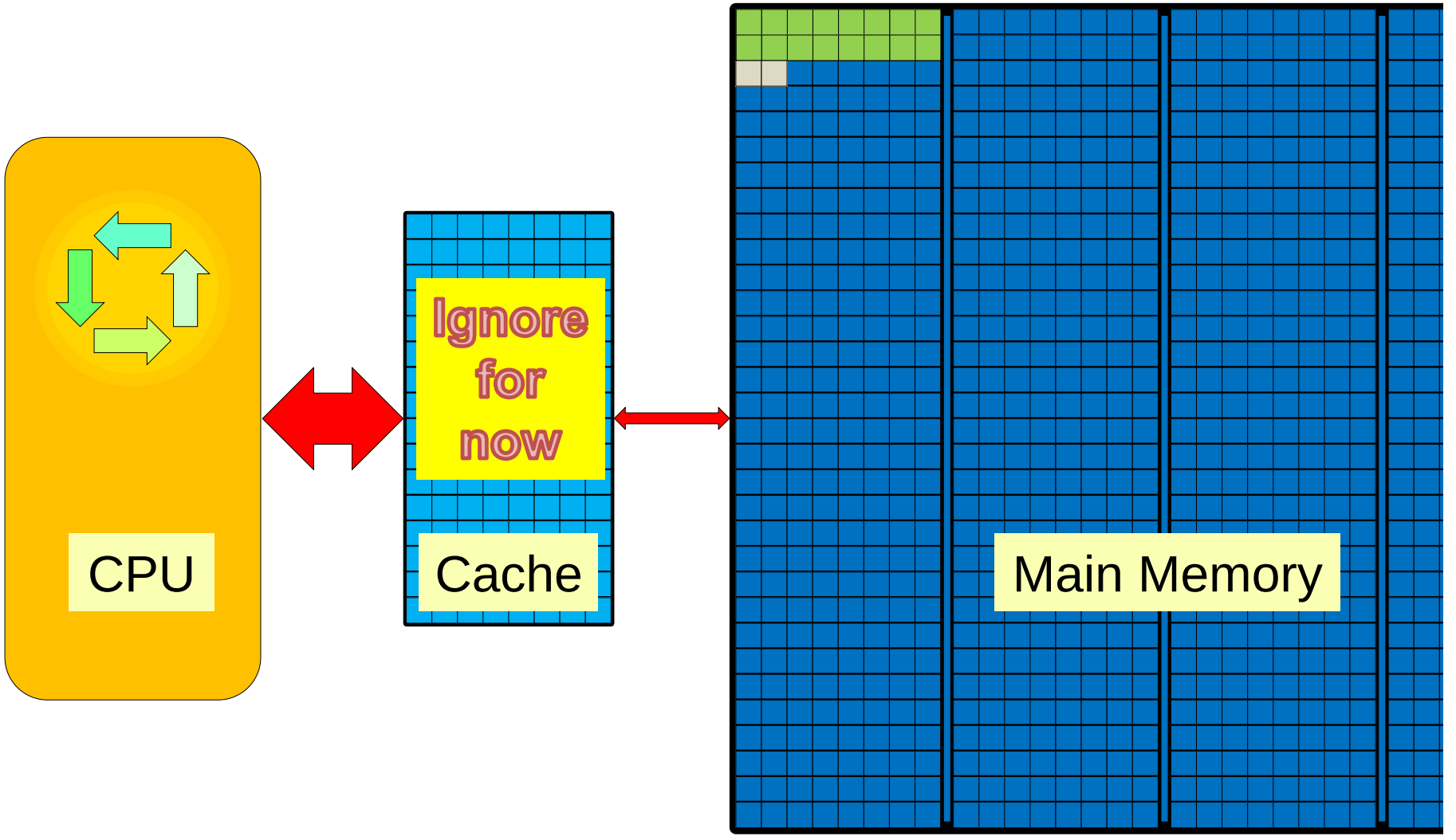
Benchmark II: LOCALITY



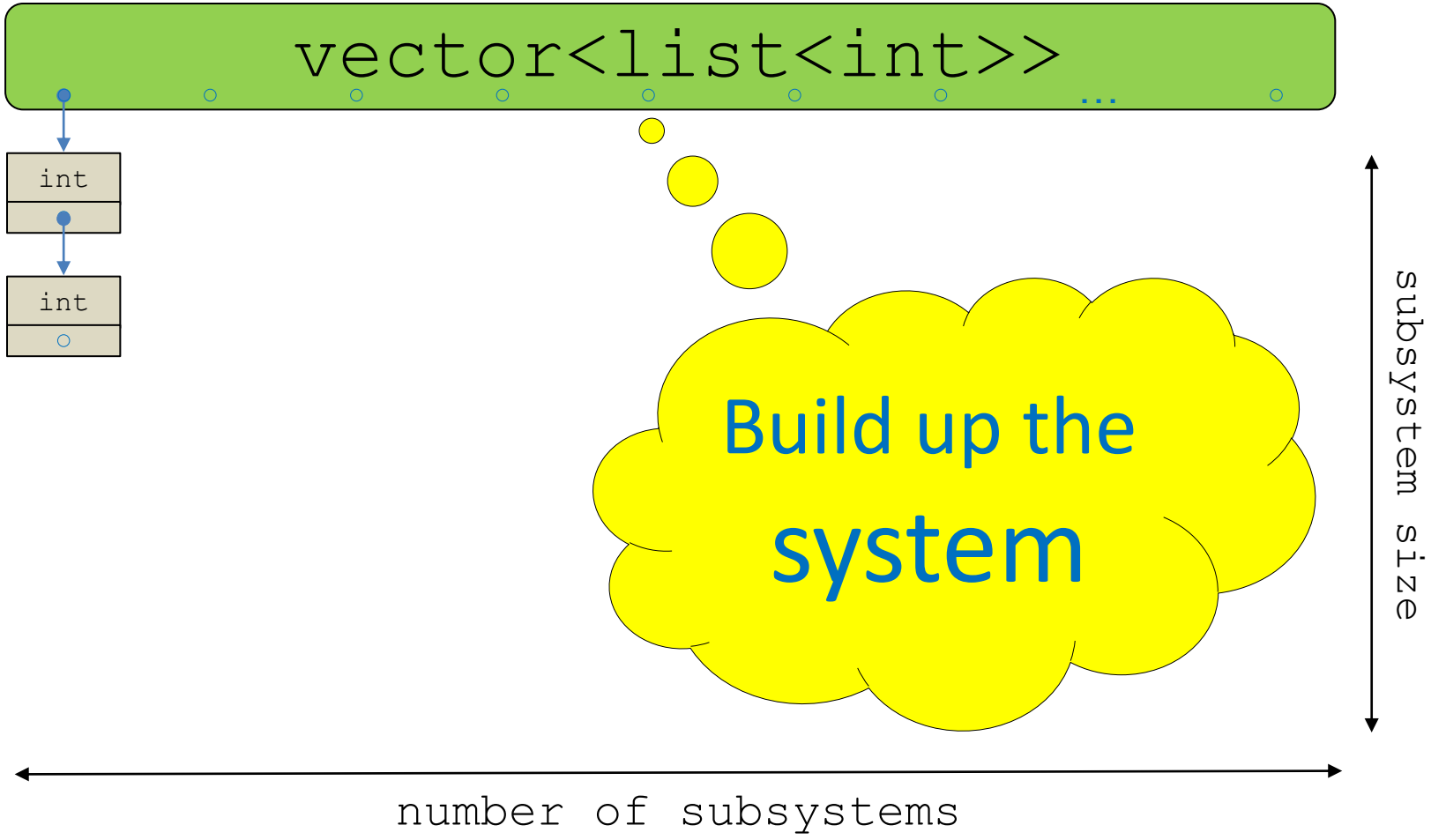
Benchmark II: LOCALITY



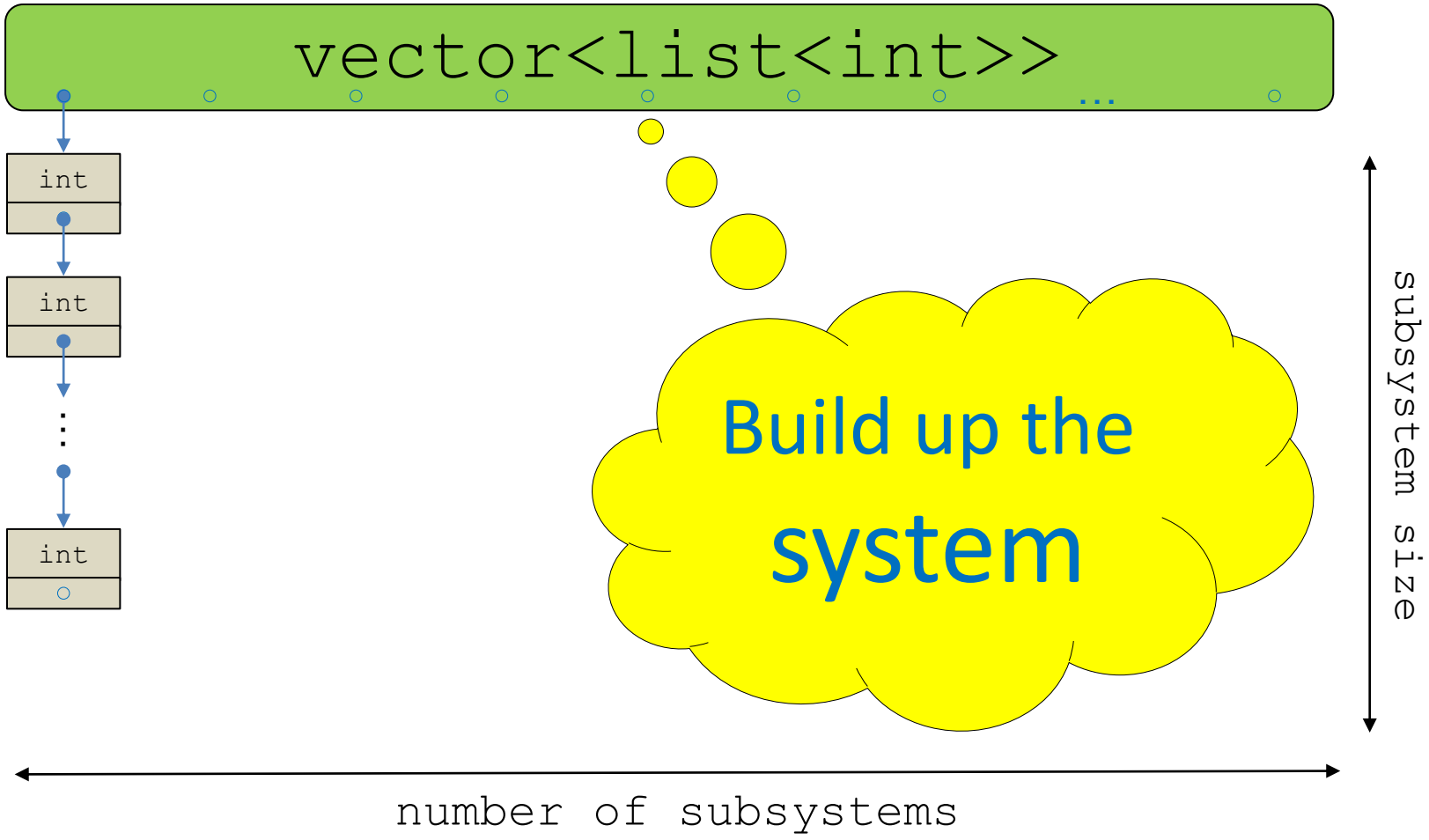
Benchmark II: LOCALITY



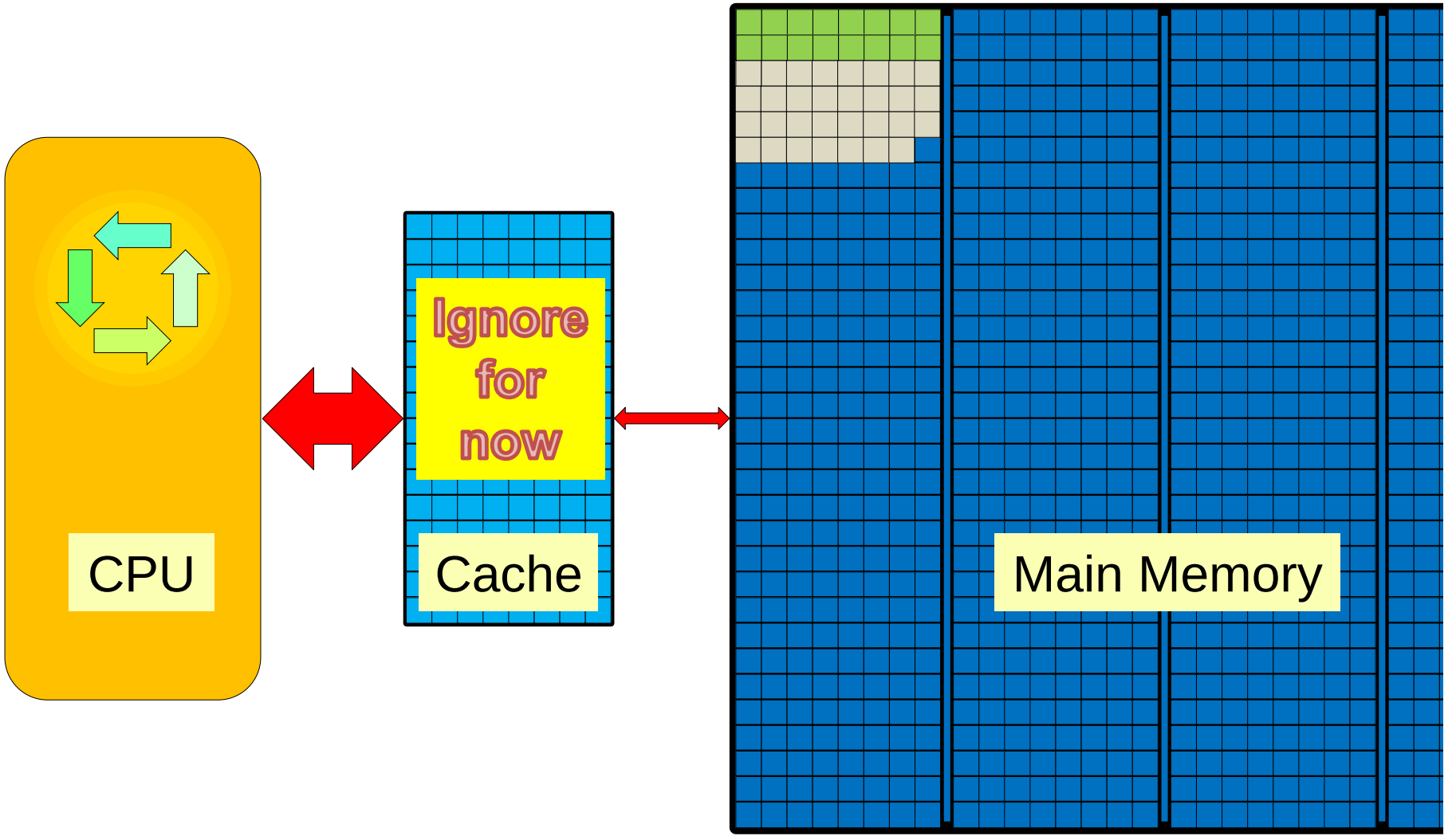
Benchmark II: LOCALITY



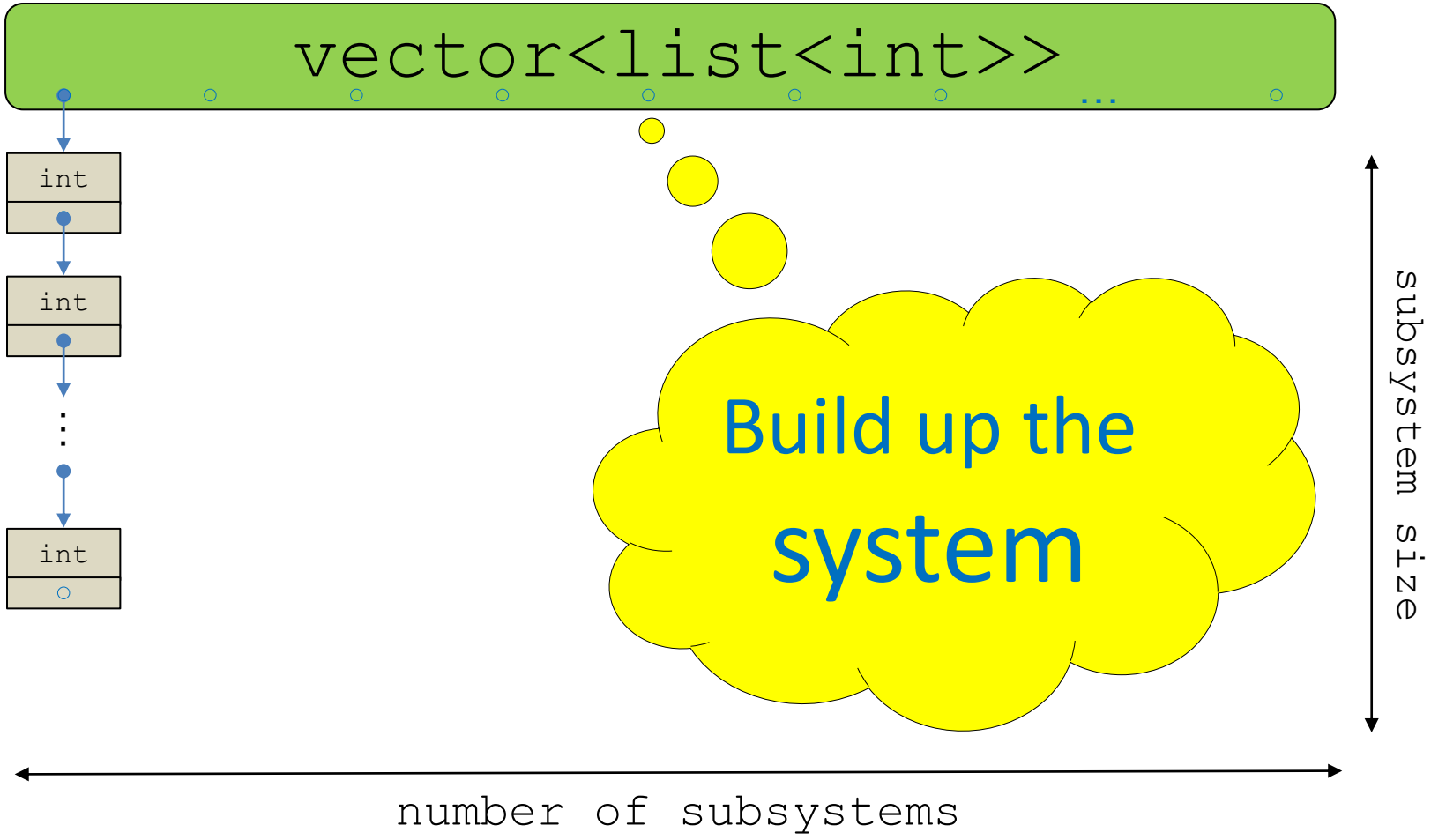
Benchmark II: LOCALITY



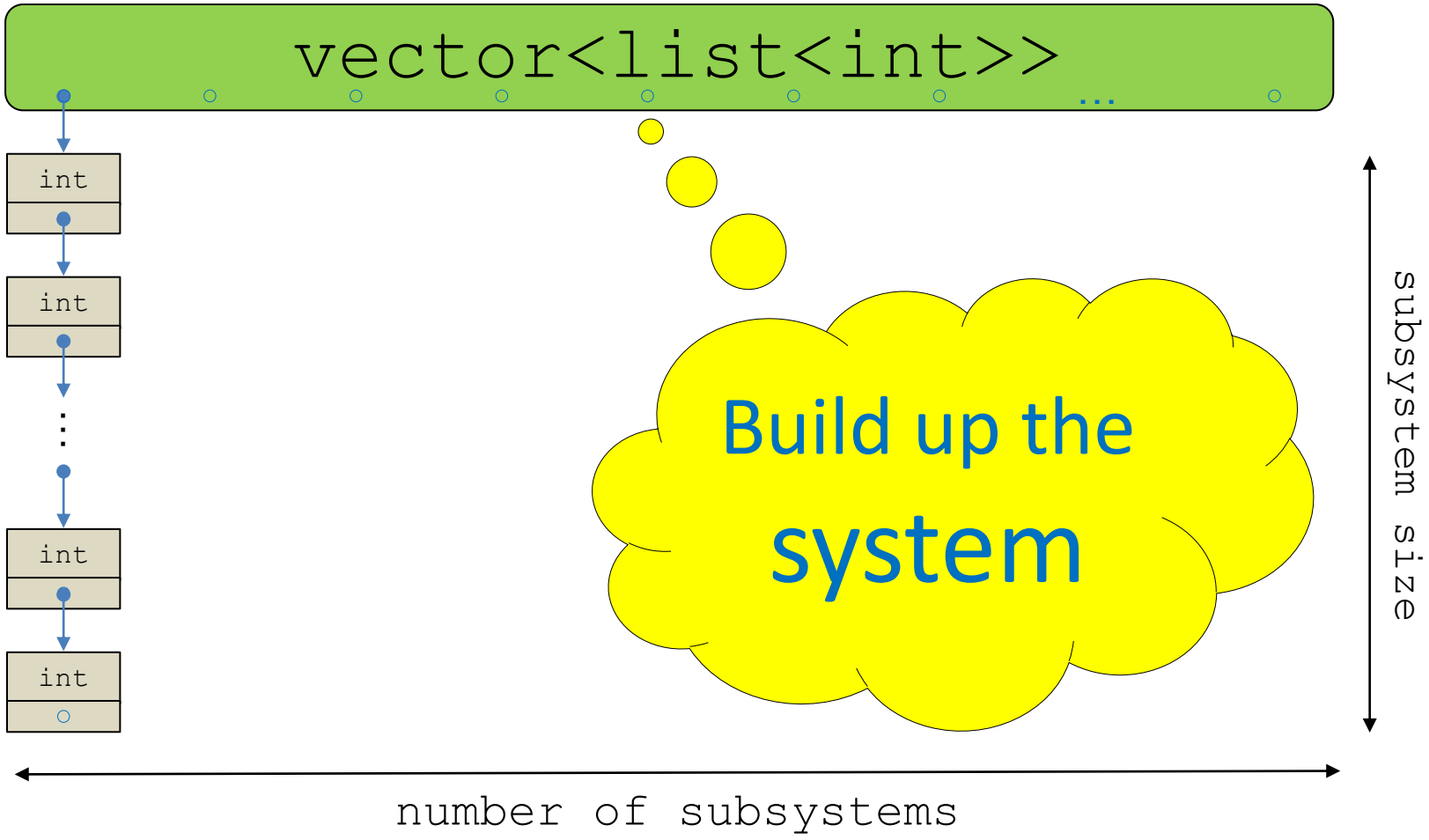
Benchmark II: LOCALITY



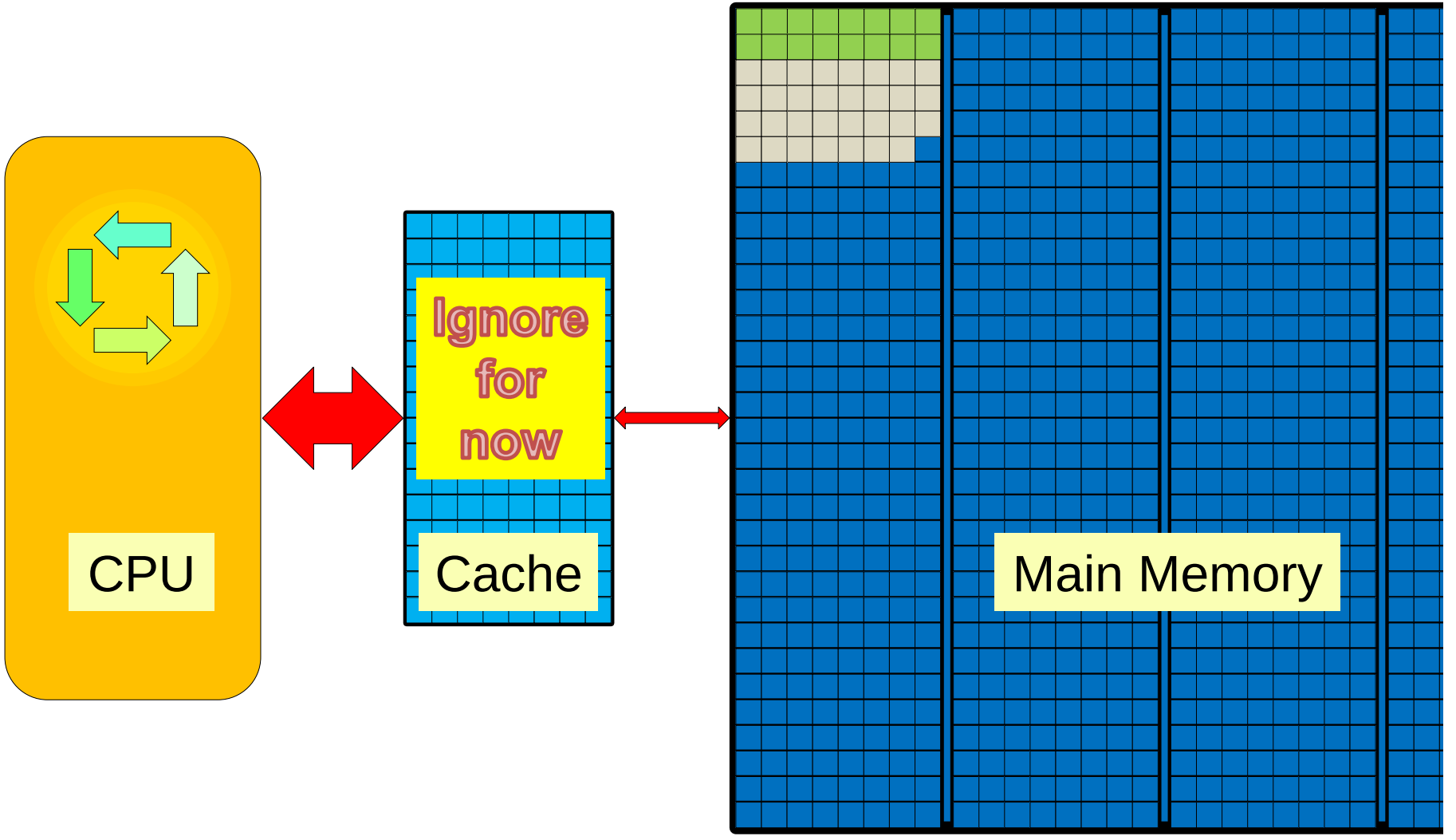
Benchmark II: LOCALITY



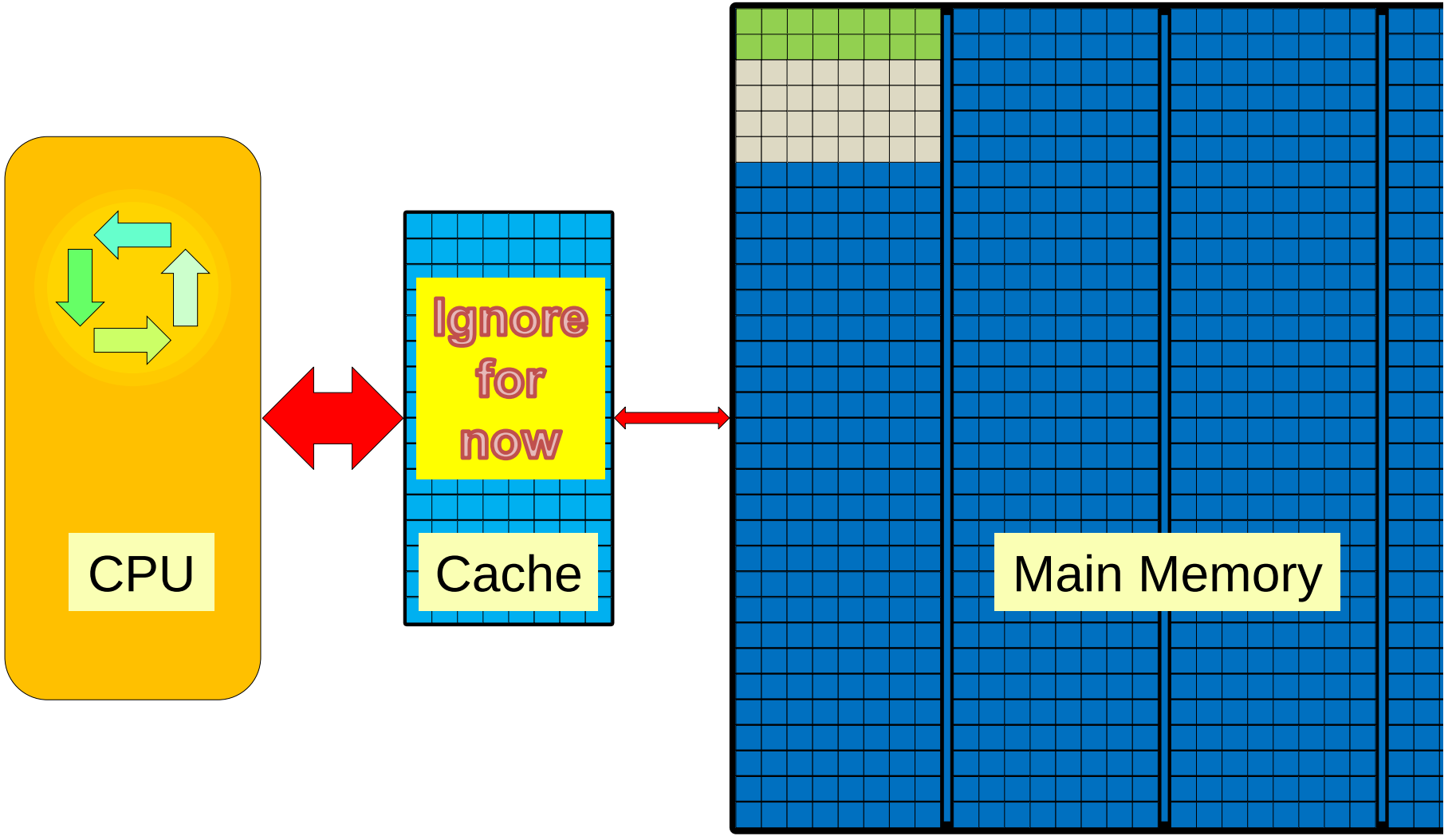
Benchmark II: LOCALITY



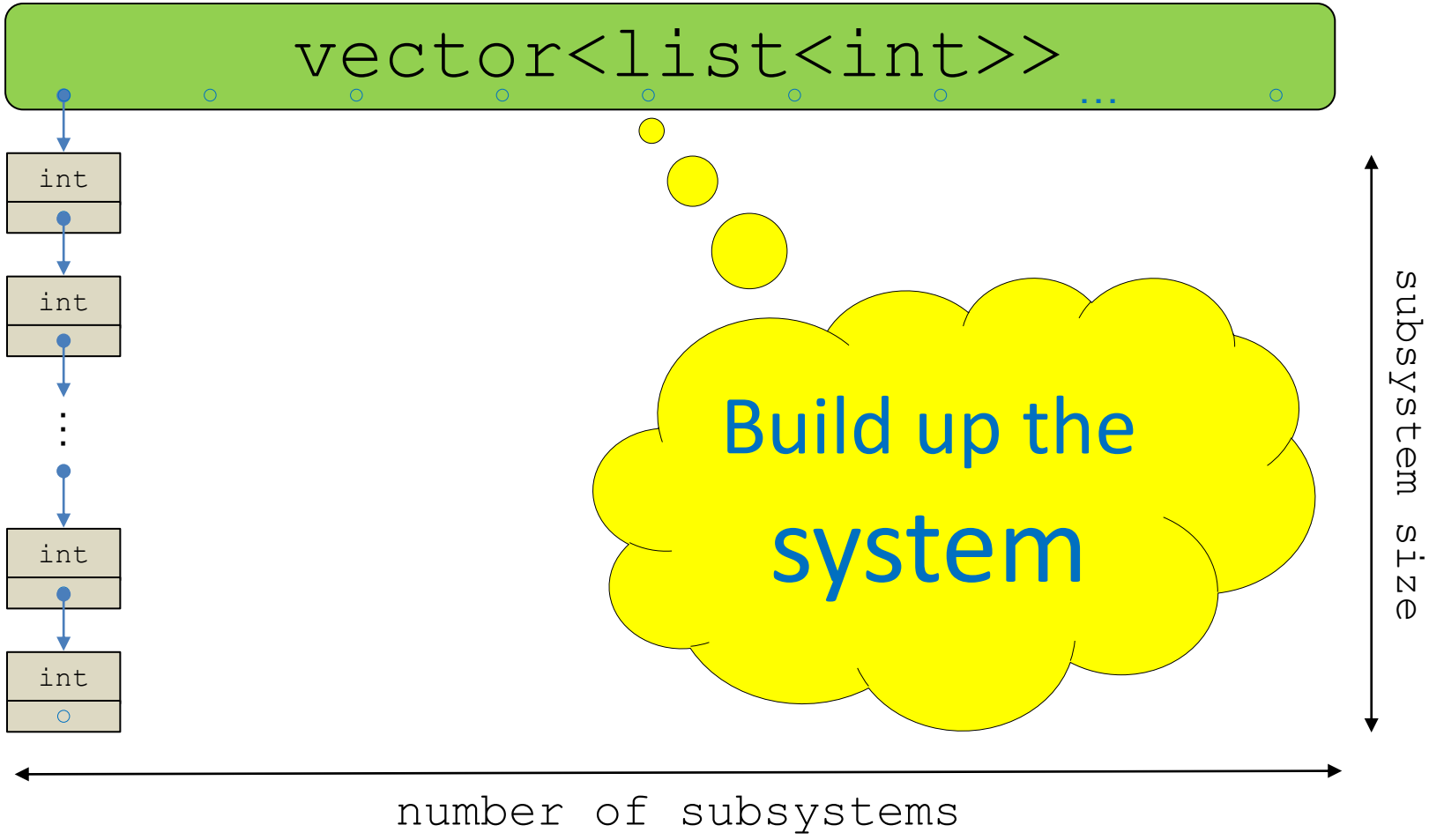
Benchmark II: LOCALITY



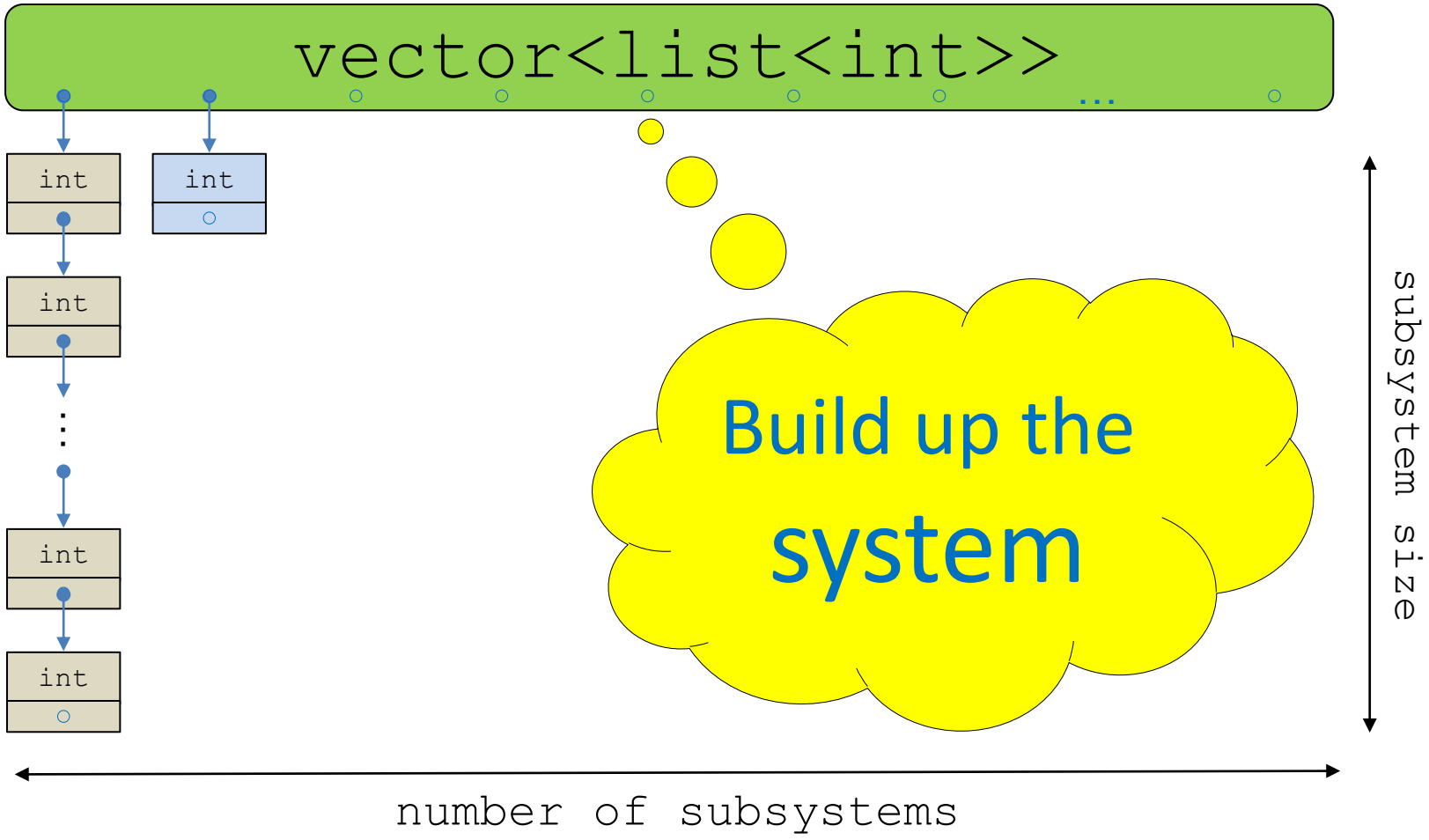
Benchmark II: LOCALITY



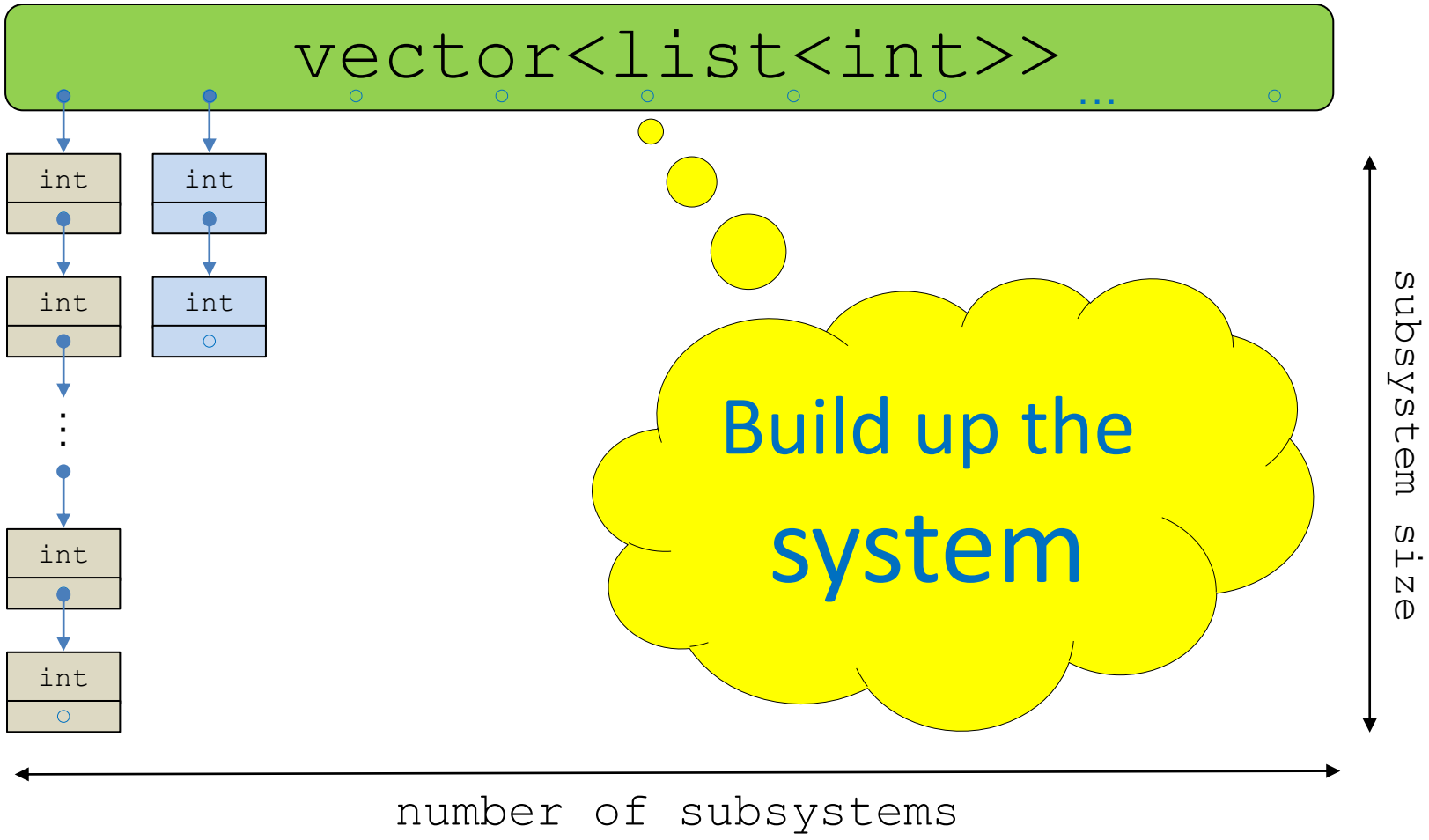
Benchmark II: LOCALITY



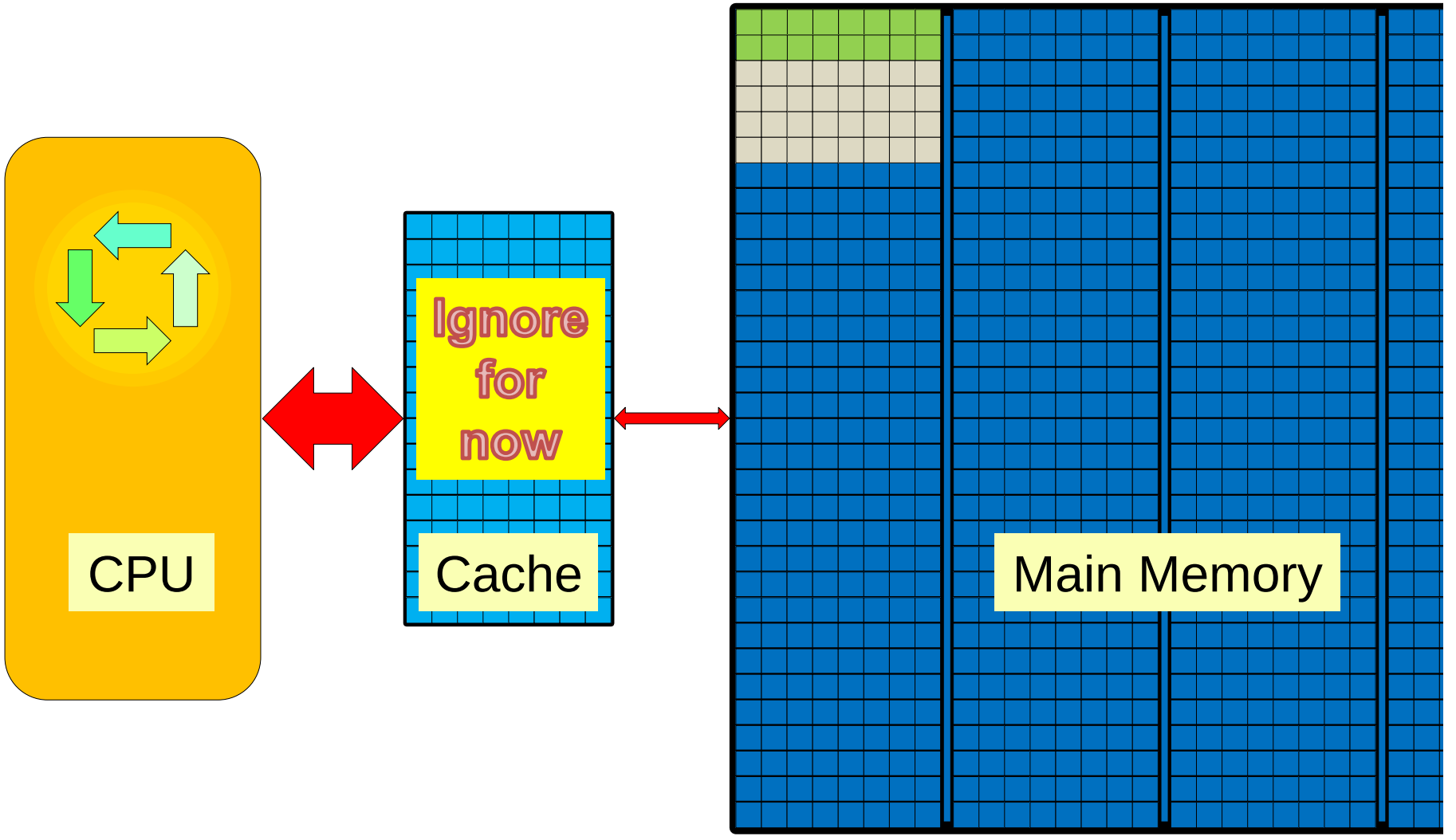
Benchmark II: LOCALITY



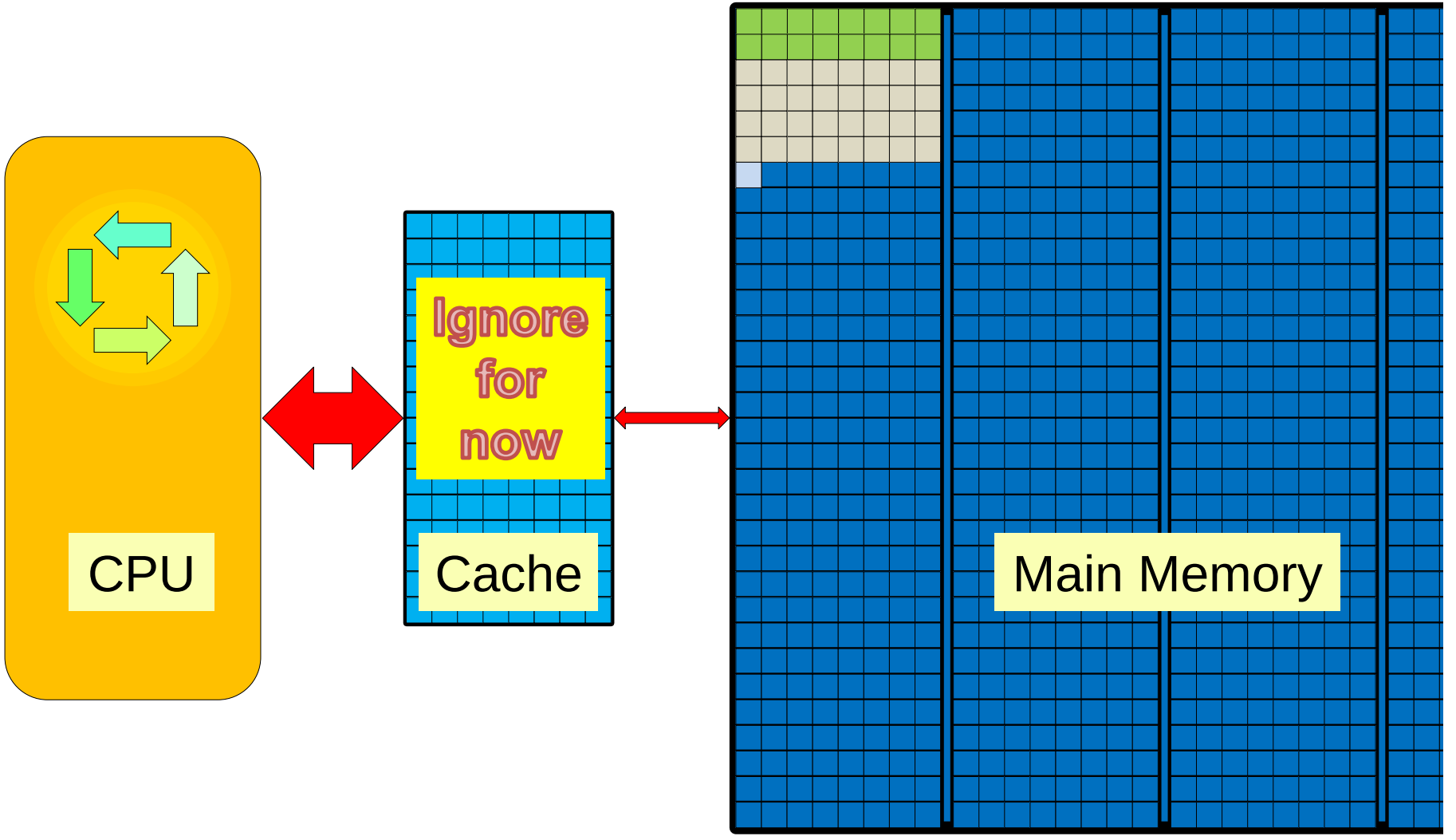
Benchmark II: LOCALITY



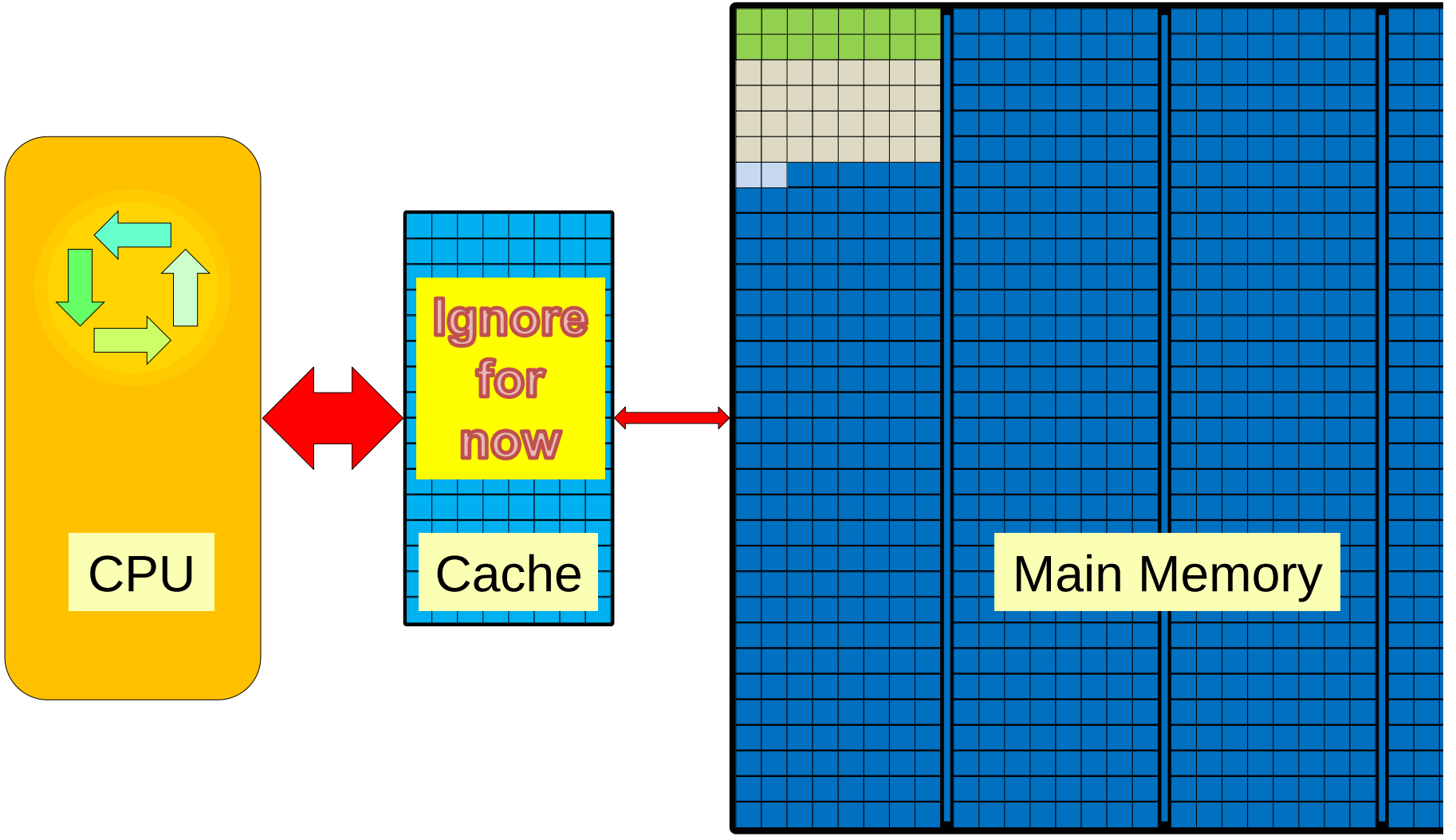
Benchmark II: LOCALITY



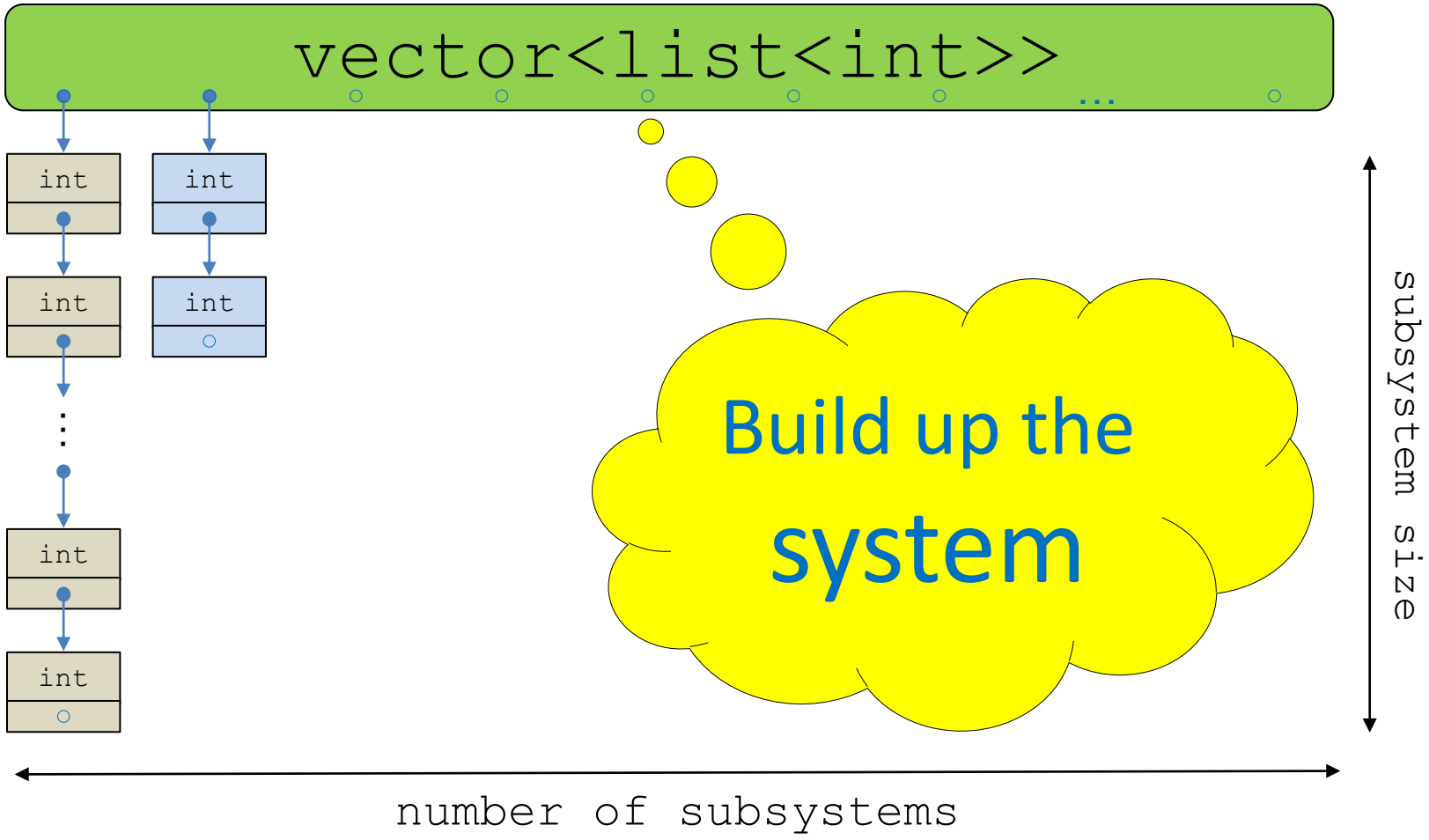
Benchmark II: LOCALITY



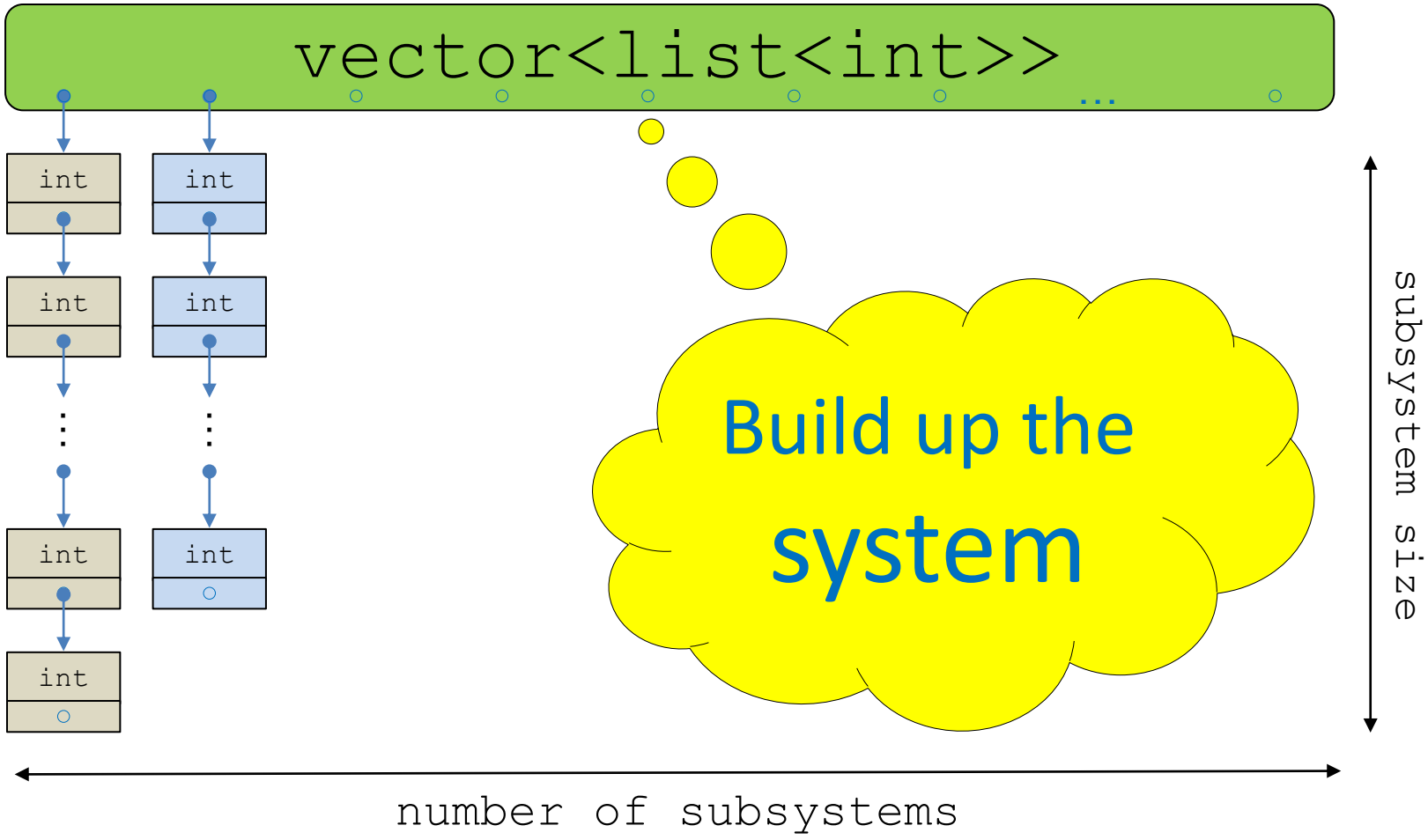
Benchmark II: LOCALITY



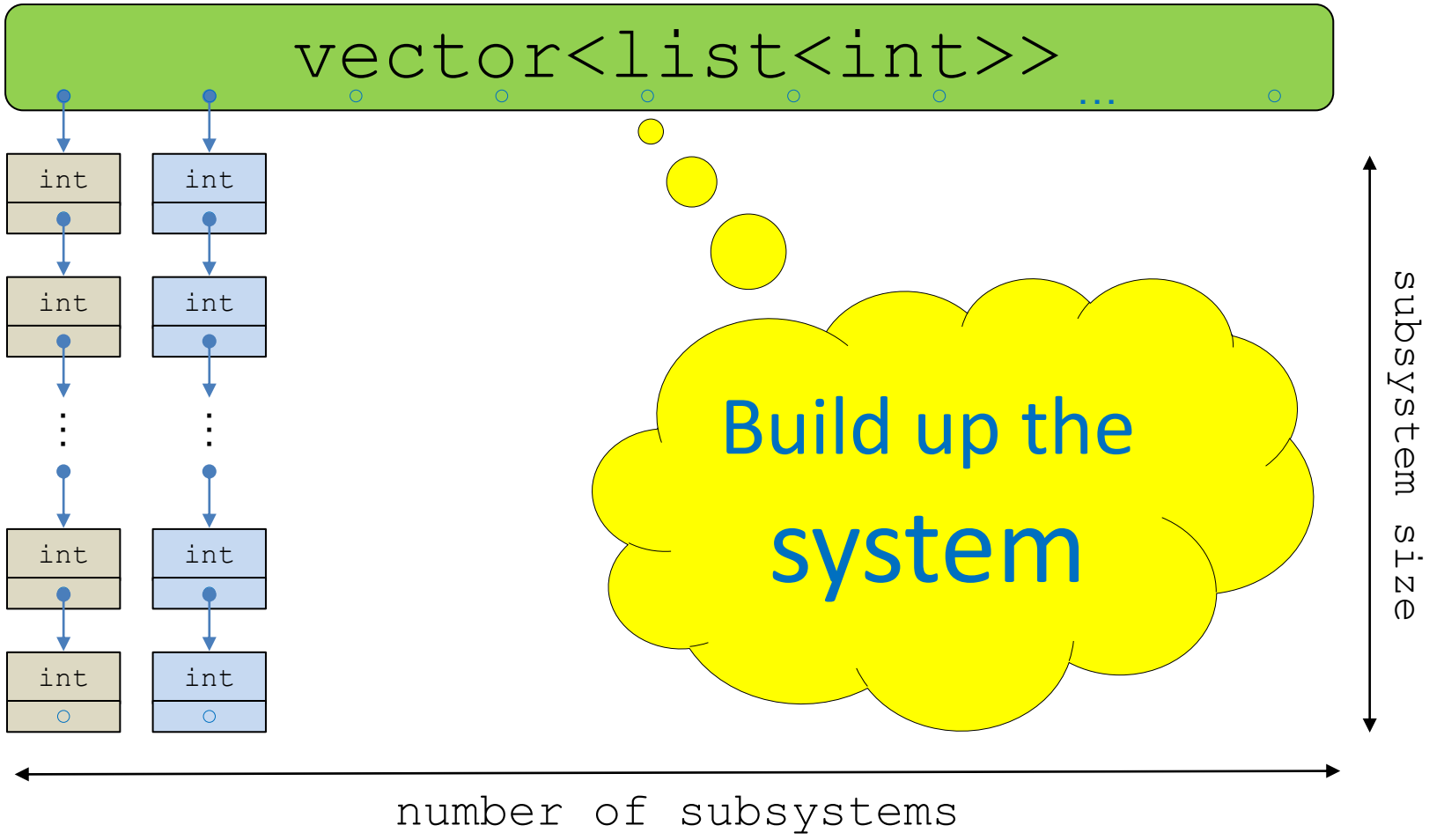
Benchmark II: LOCALITY



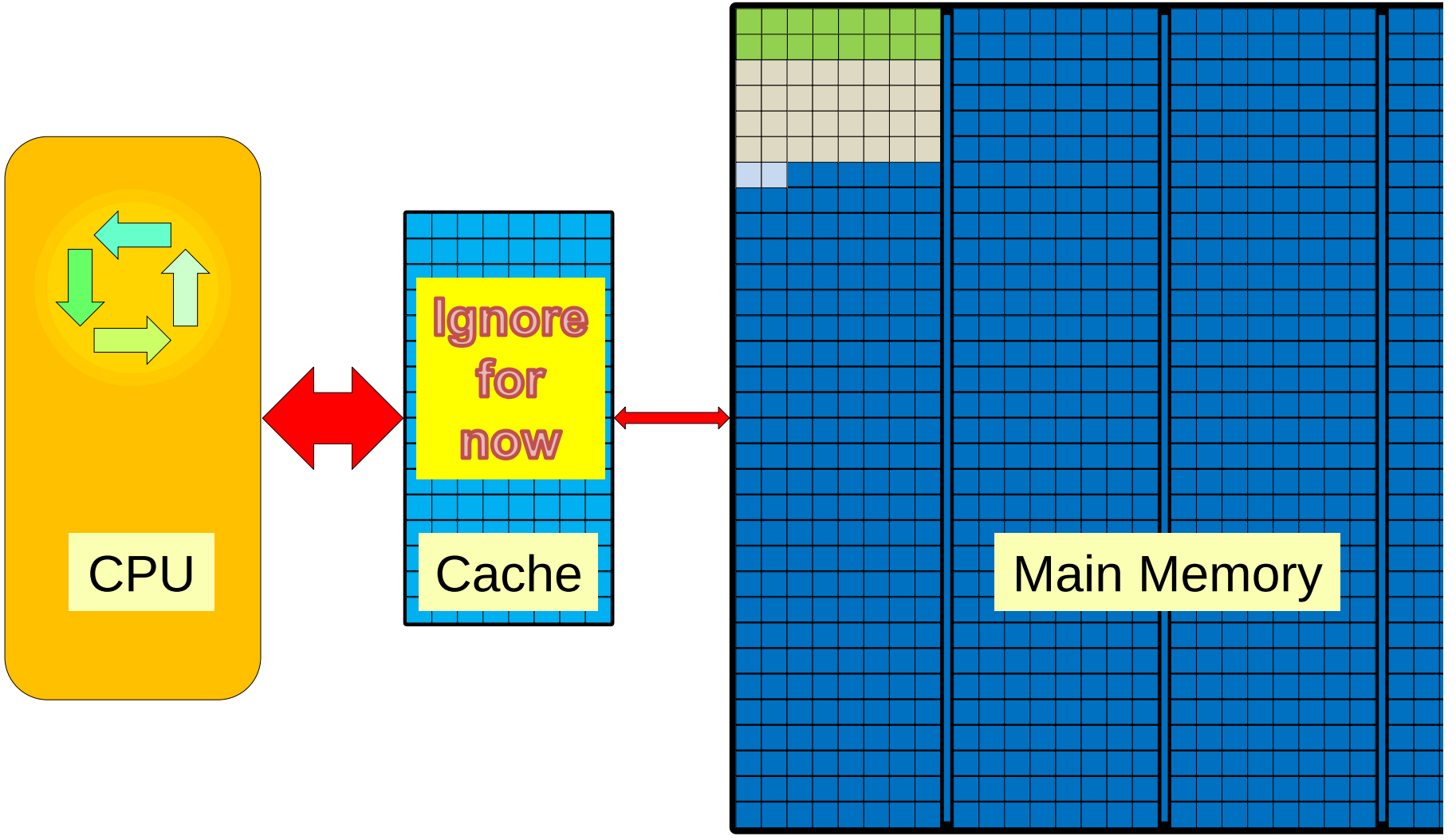
Benchmark II: LOCALITY



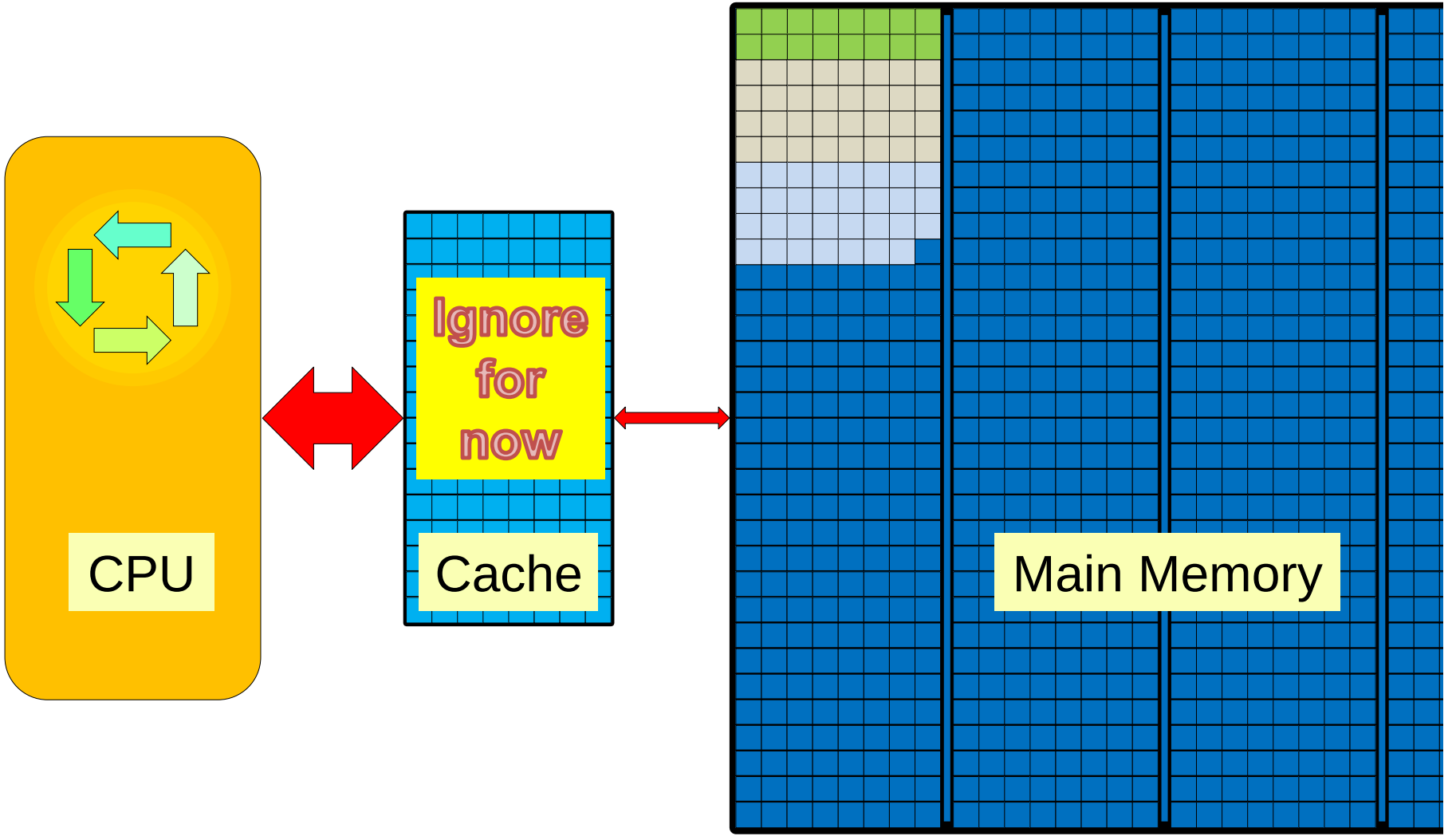
Benchmark II: LOCALITY



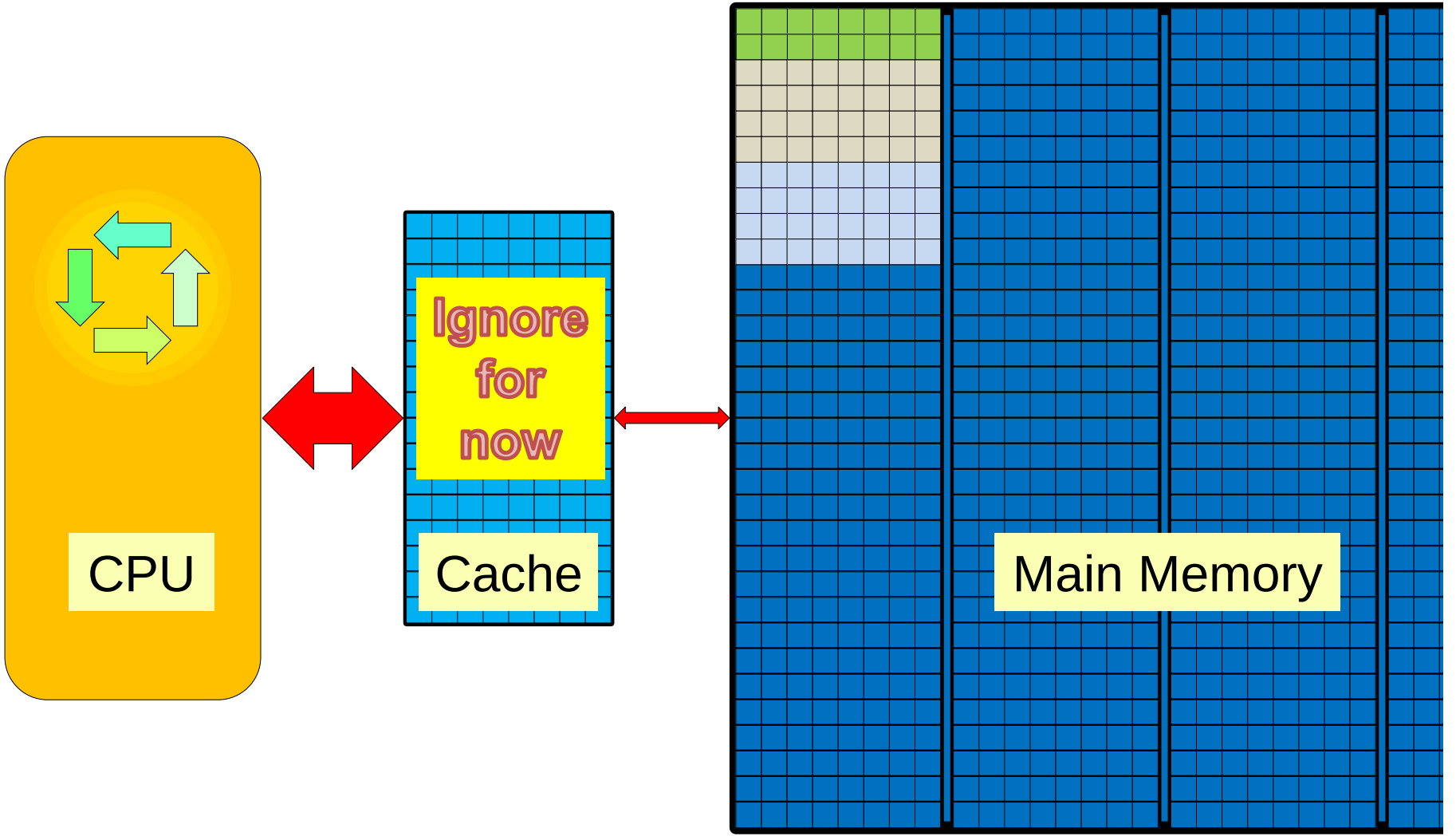
Benchmark II: LOCALITY



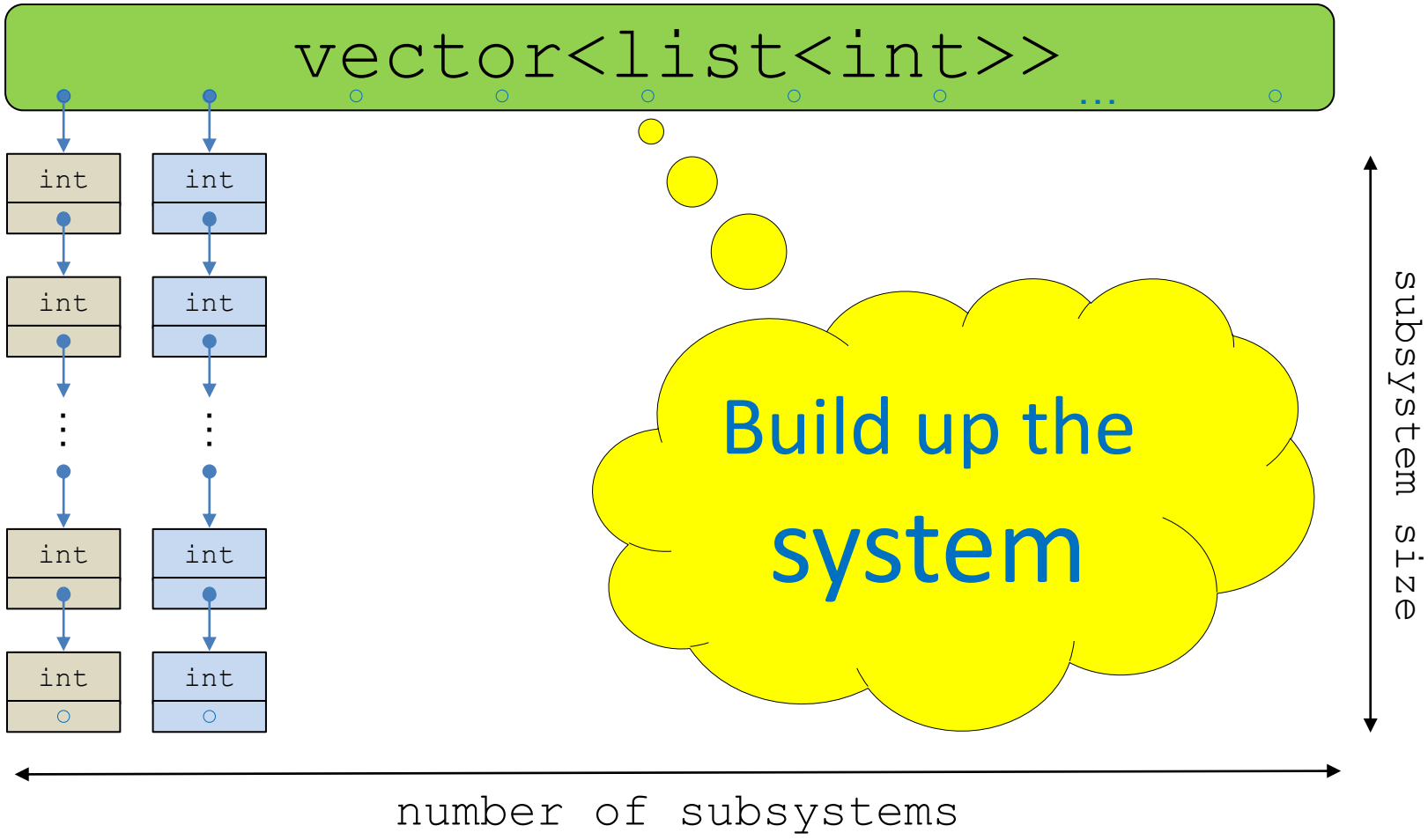
Benchmark II: LOCALITY



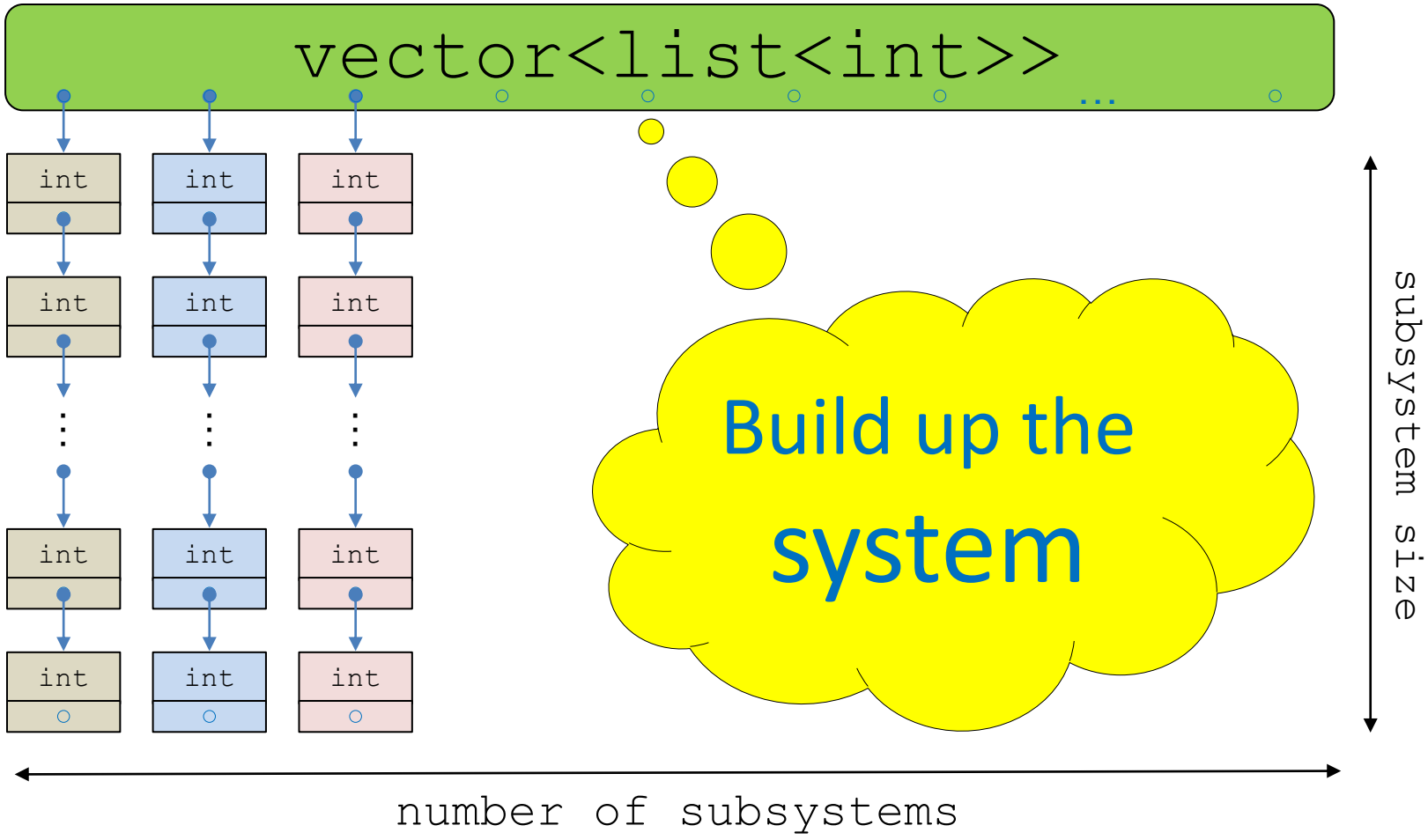
Benchmark II: LOCALITY



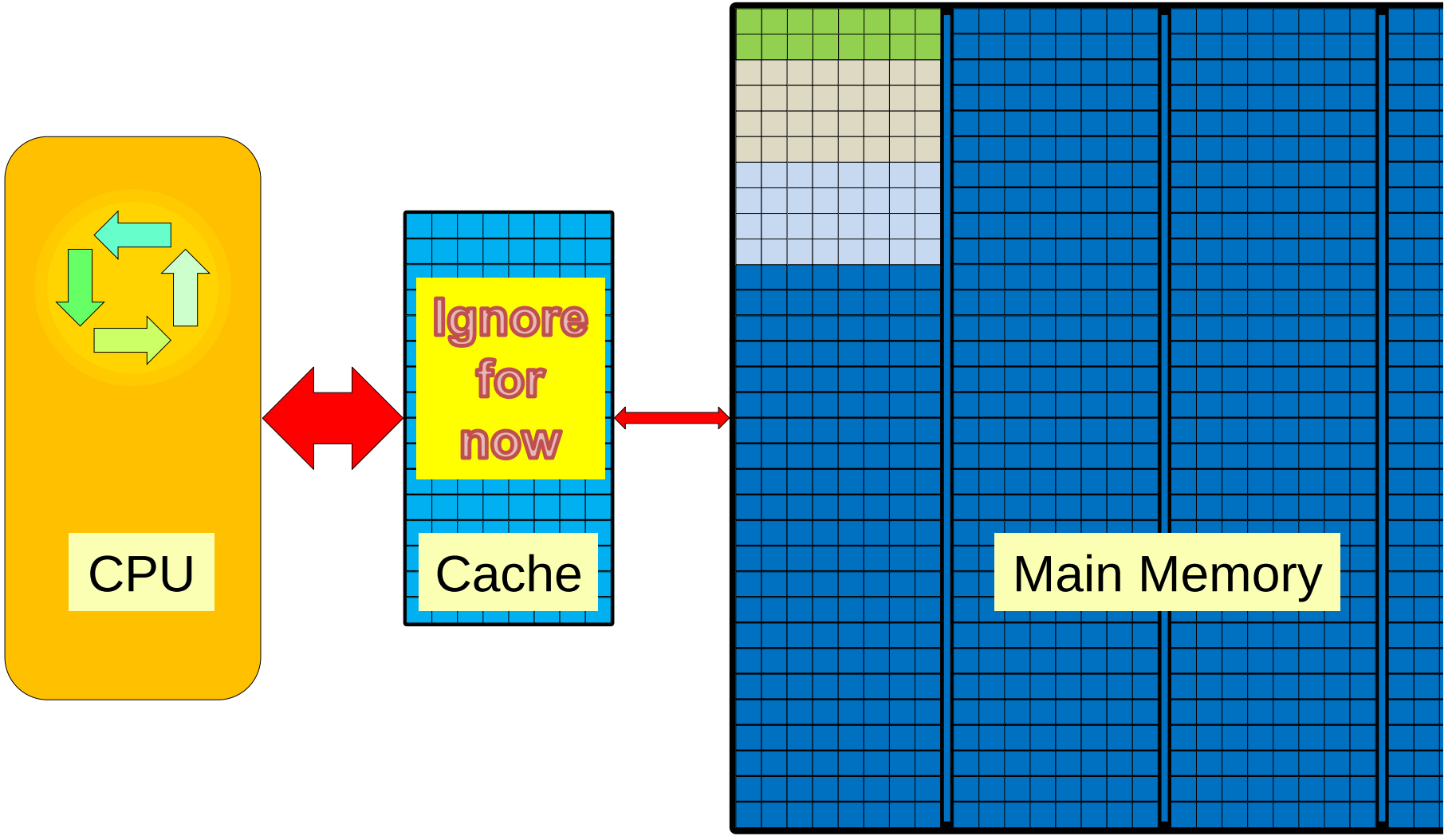
Benchmark II: LOCALITY



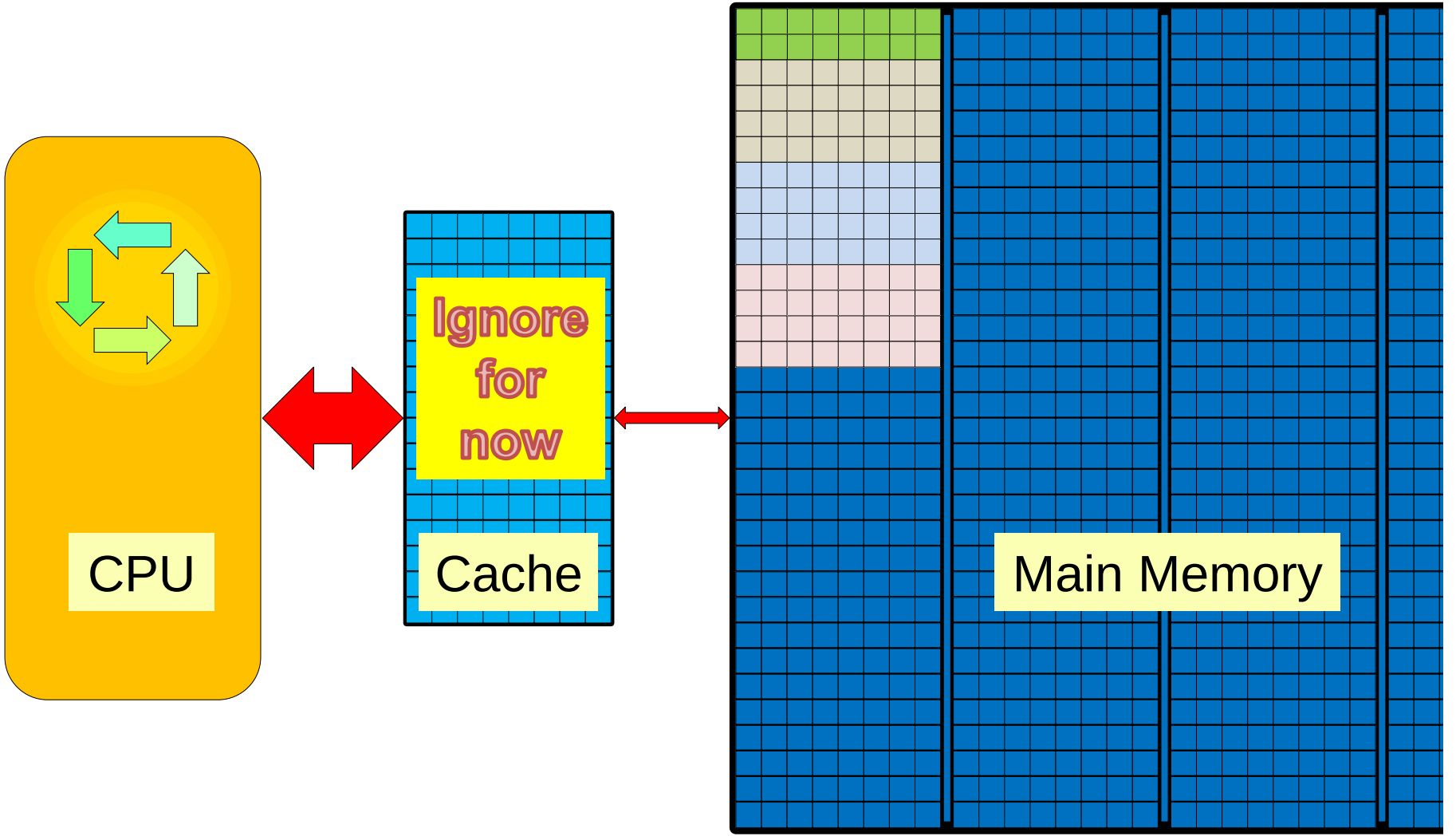
Benchmark II: LOCALITY



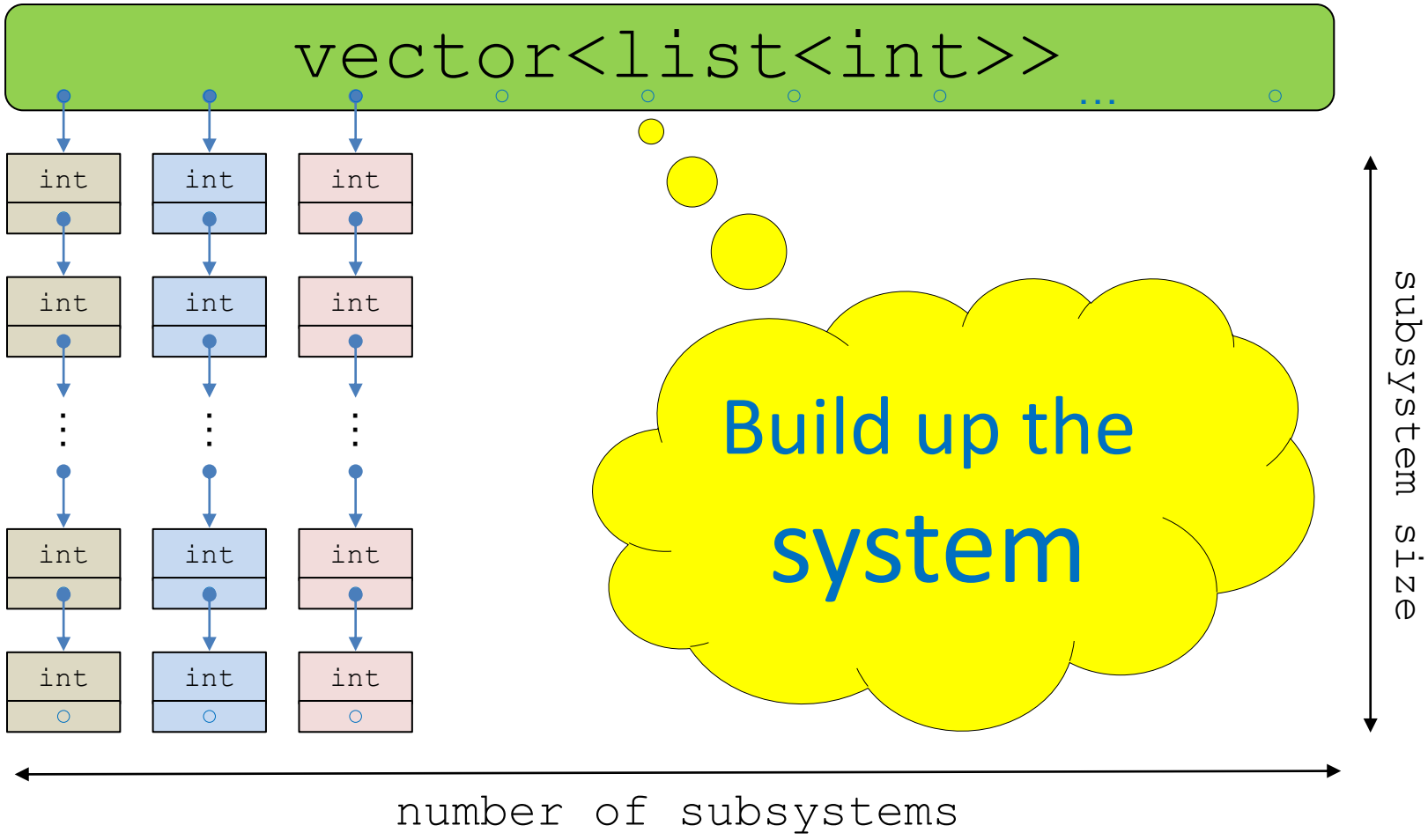
Benchmark II: LOCALITY



Benchmark II: LOCALITY

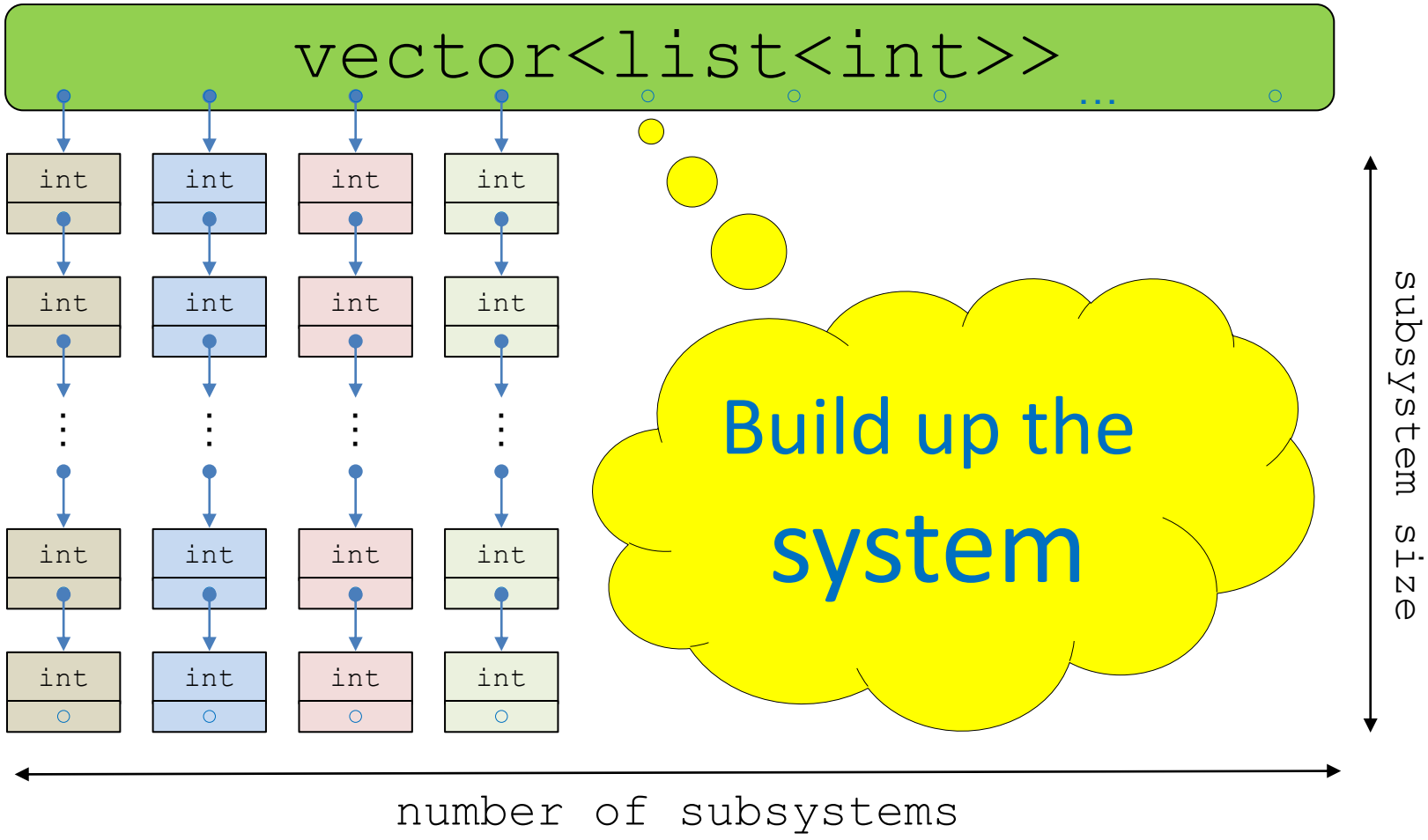


Benchmark II: LOCALITY

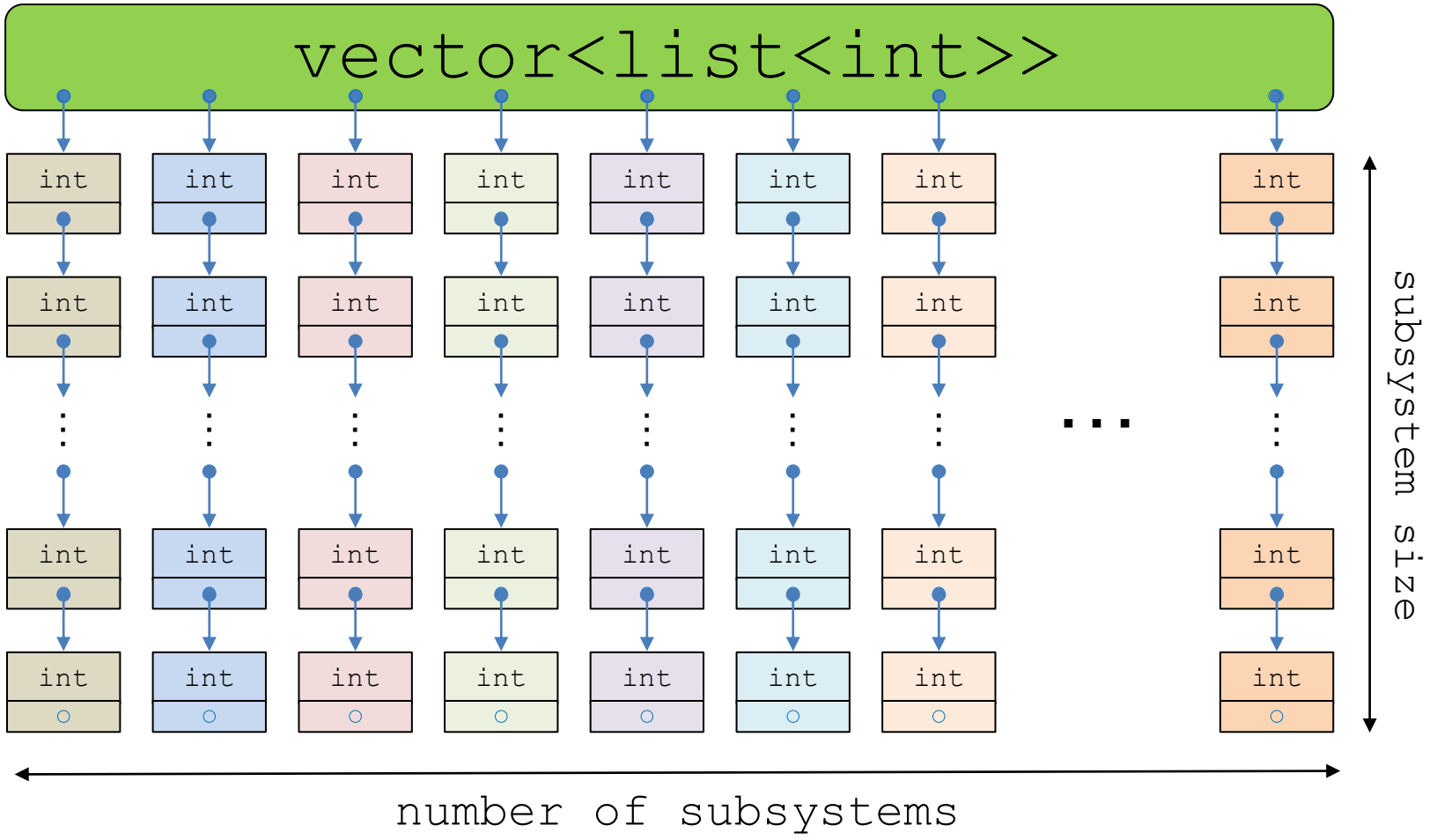


3. Analyzing the Benchmark Data

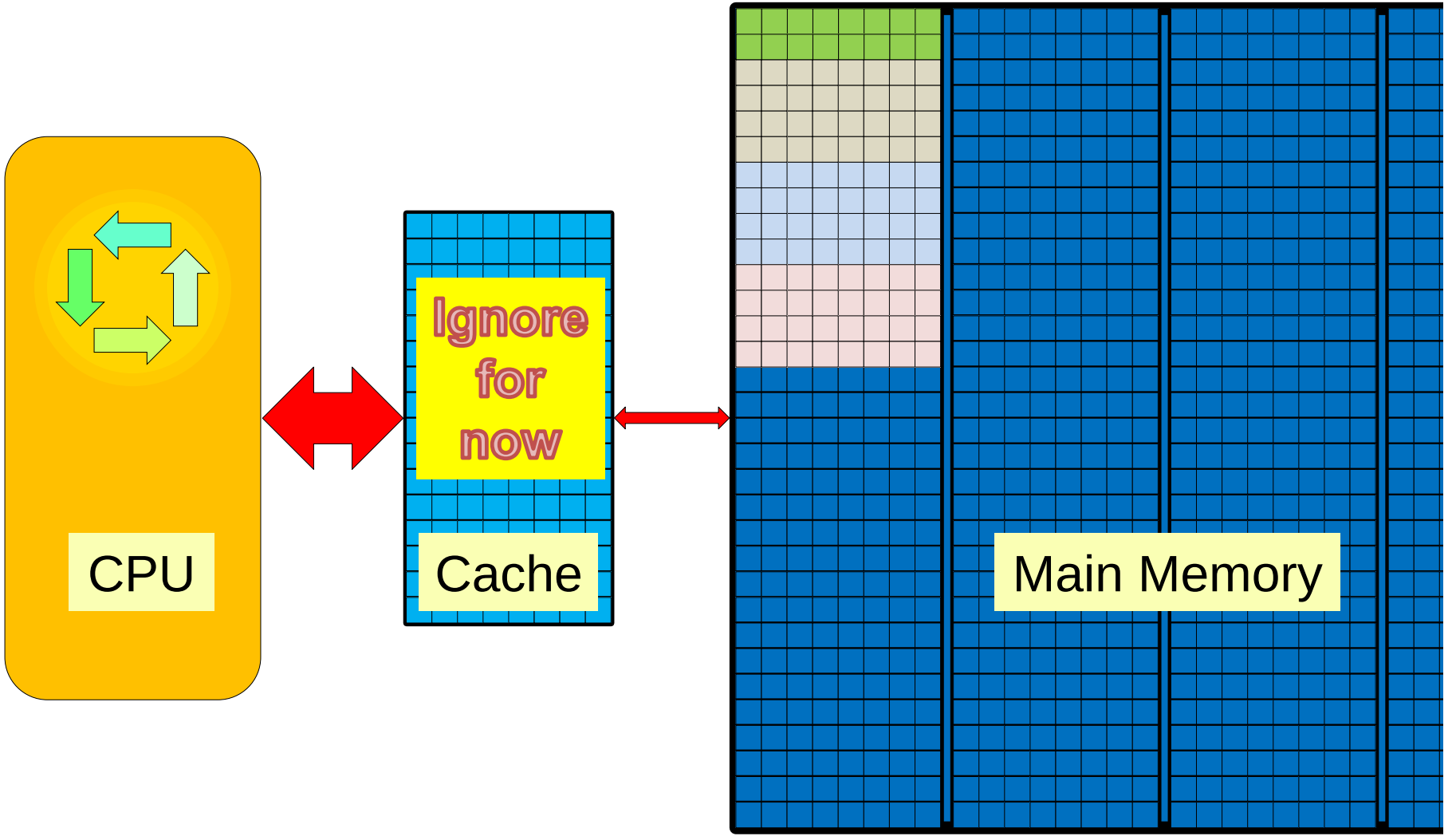
Benchmark II: LOCALITY



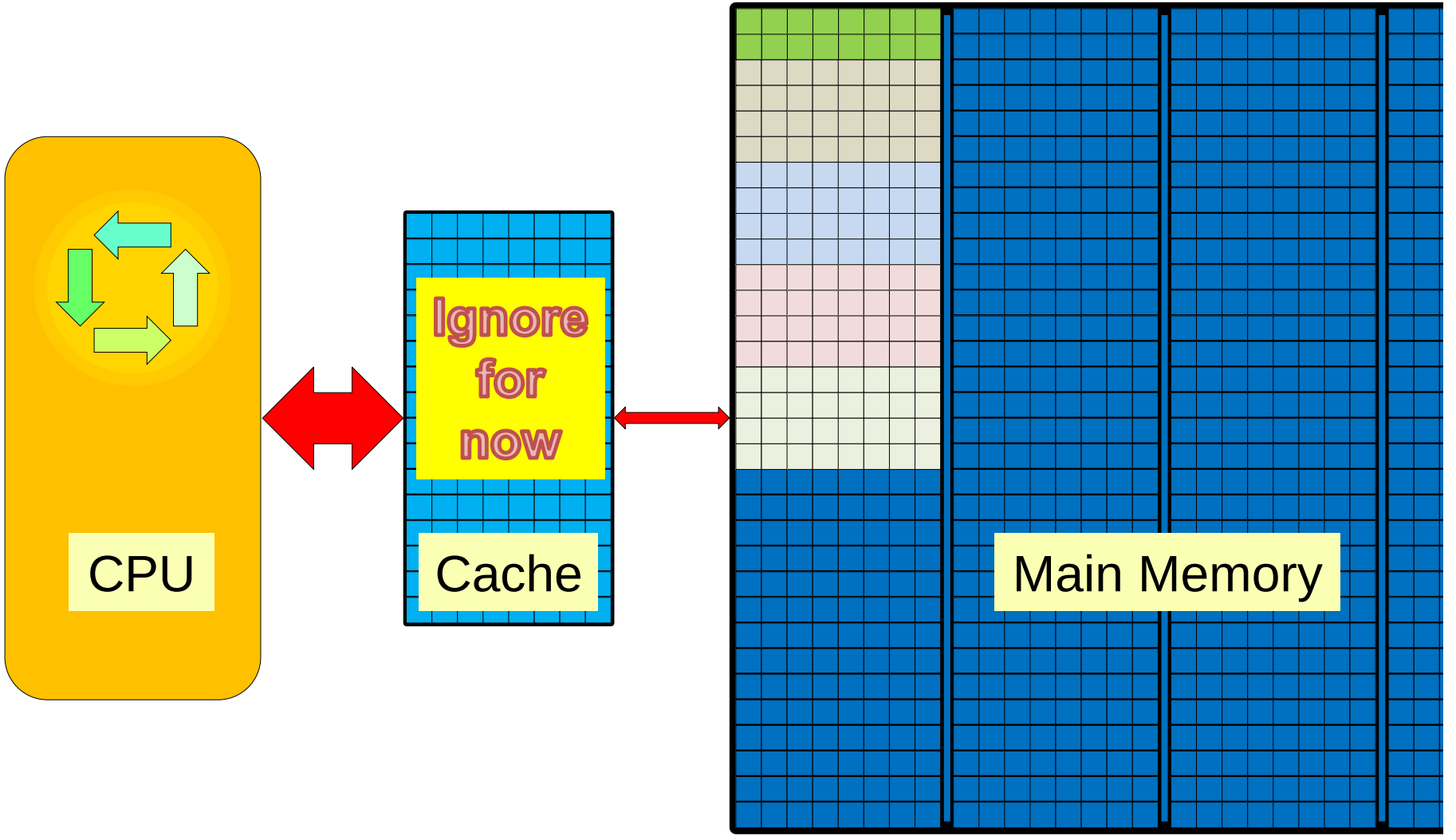
Benchmark II: LOCALITY



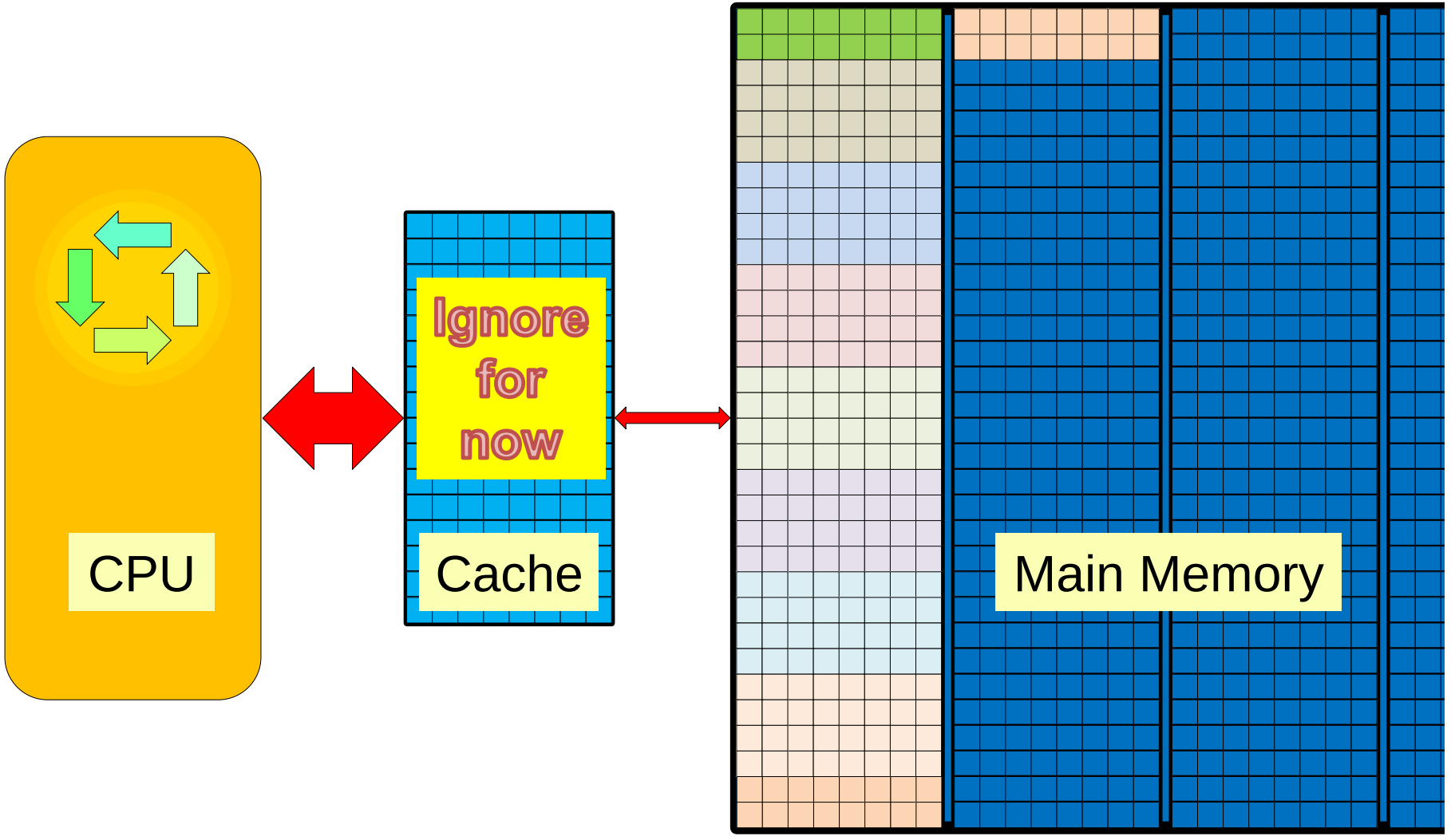
Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY

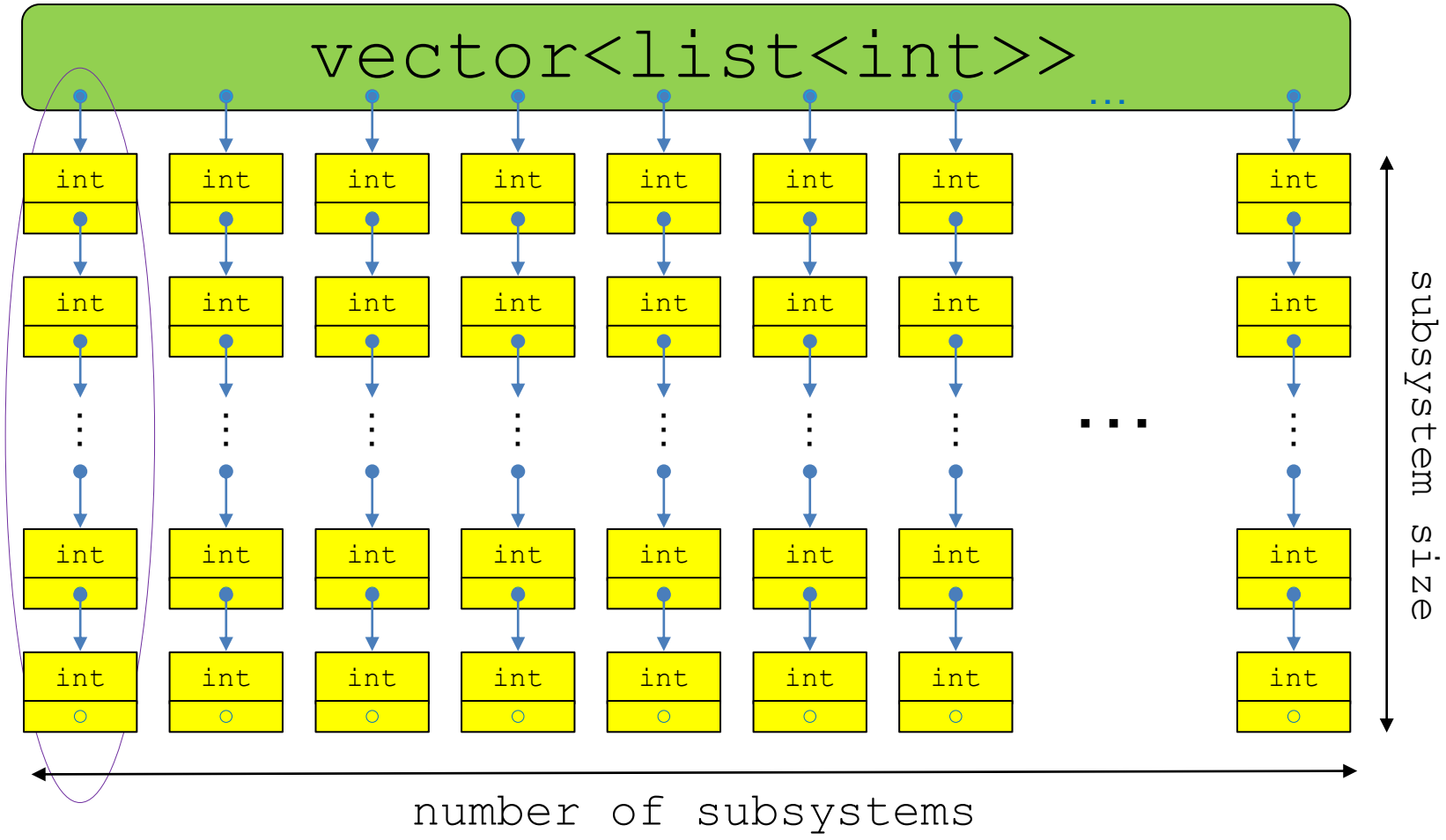


Benchmark II: LOCALITY

Access Plan:

- Visit each subsystem in turn.
- For each subsystem
  - Write to each of the `int` values in order.
  - ***Repeat*** the sequence of writes for a total of **I** iterations.
  - Advance to the next subsystem.
- ***Repeat*** the entire process for a total of **R** repetitions.
- The result of this experiment is (wall-clock) **RUN TIME.**
  - No memory is allocated or deallocated.

Benchmark II: LOCALITY



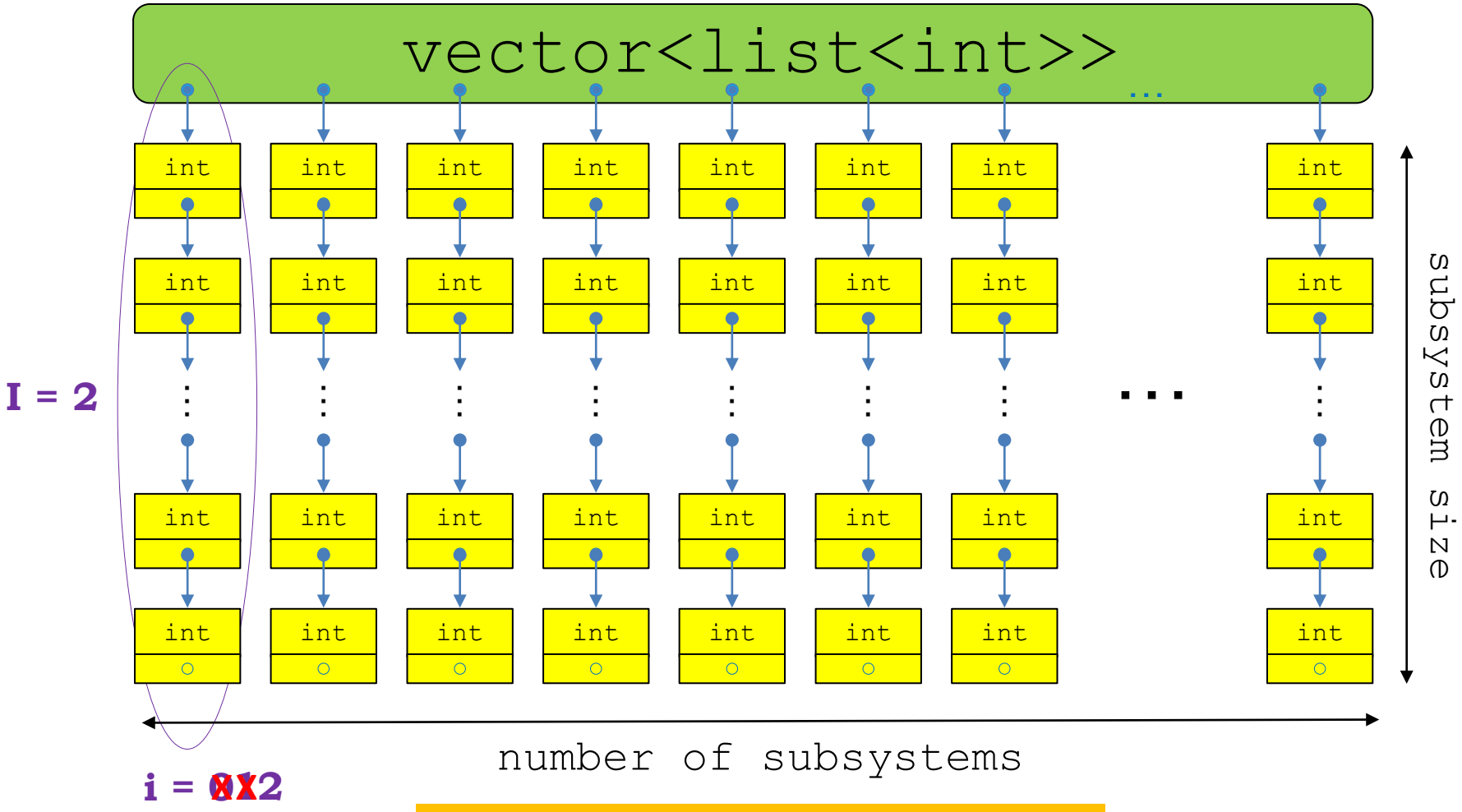
Single Iteration of First Subsystem Access Complete

Benchmark II: LOCALITY

Access Plan:

- Visit each subsystem in turn.
- For each subsystem
  - Write to each of the `int` values in order.
  - ***Repeat*** the sequence of writes for a total of **I** iterations.
  - Advance to the next subsystem.
- ***Repeat*** the entire process for a total of **R** repetitions.
- The result of this experiment is (wall-clock) **RUN TIME.**
  - No memory is allocated or deallocated.

Benchmark II: LOCALITY



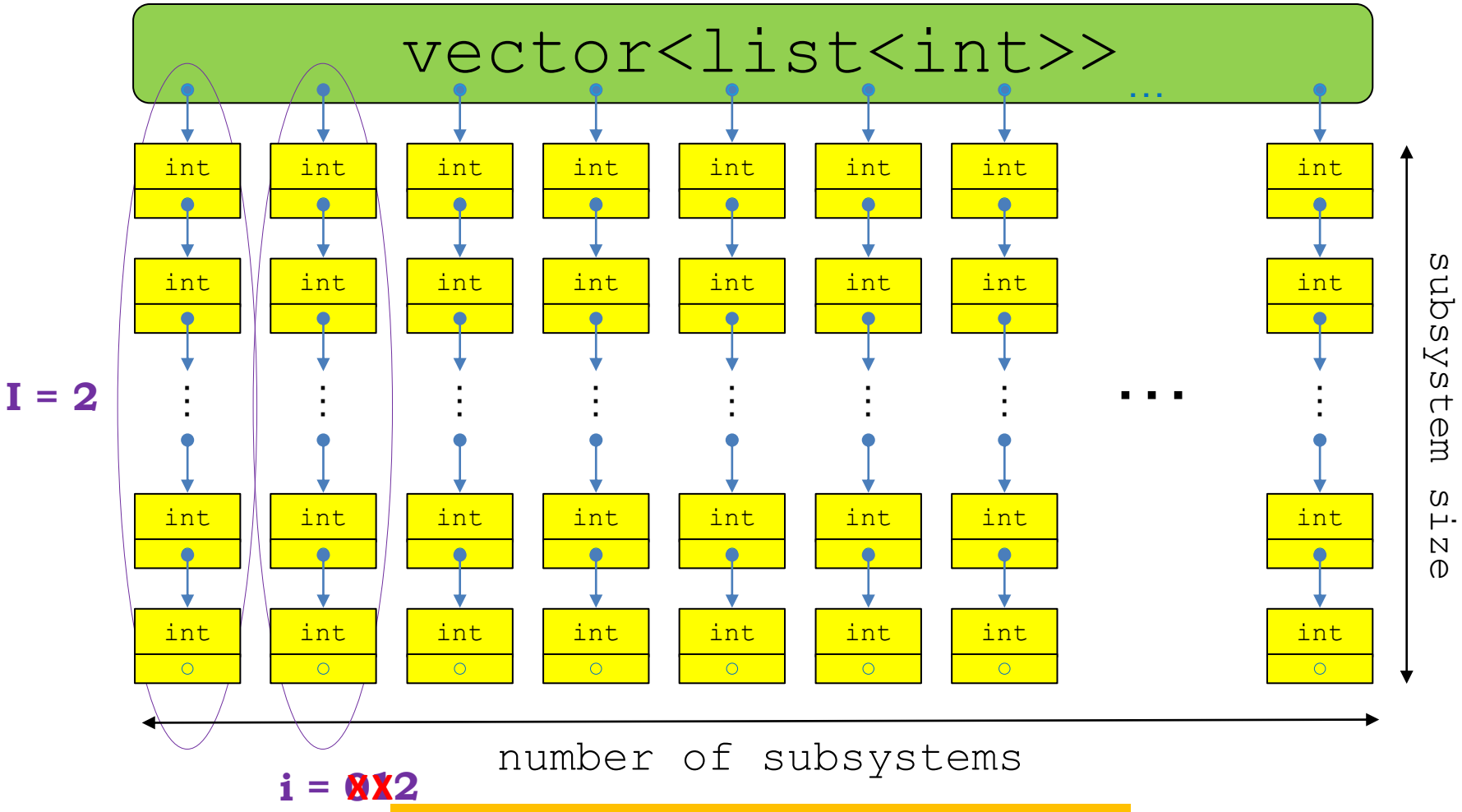
First Subsystem Access Complete

Benchmark II: LOCALITY

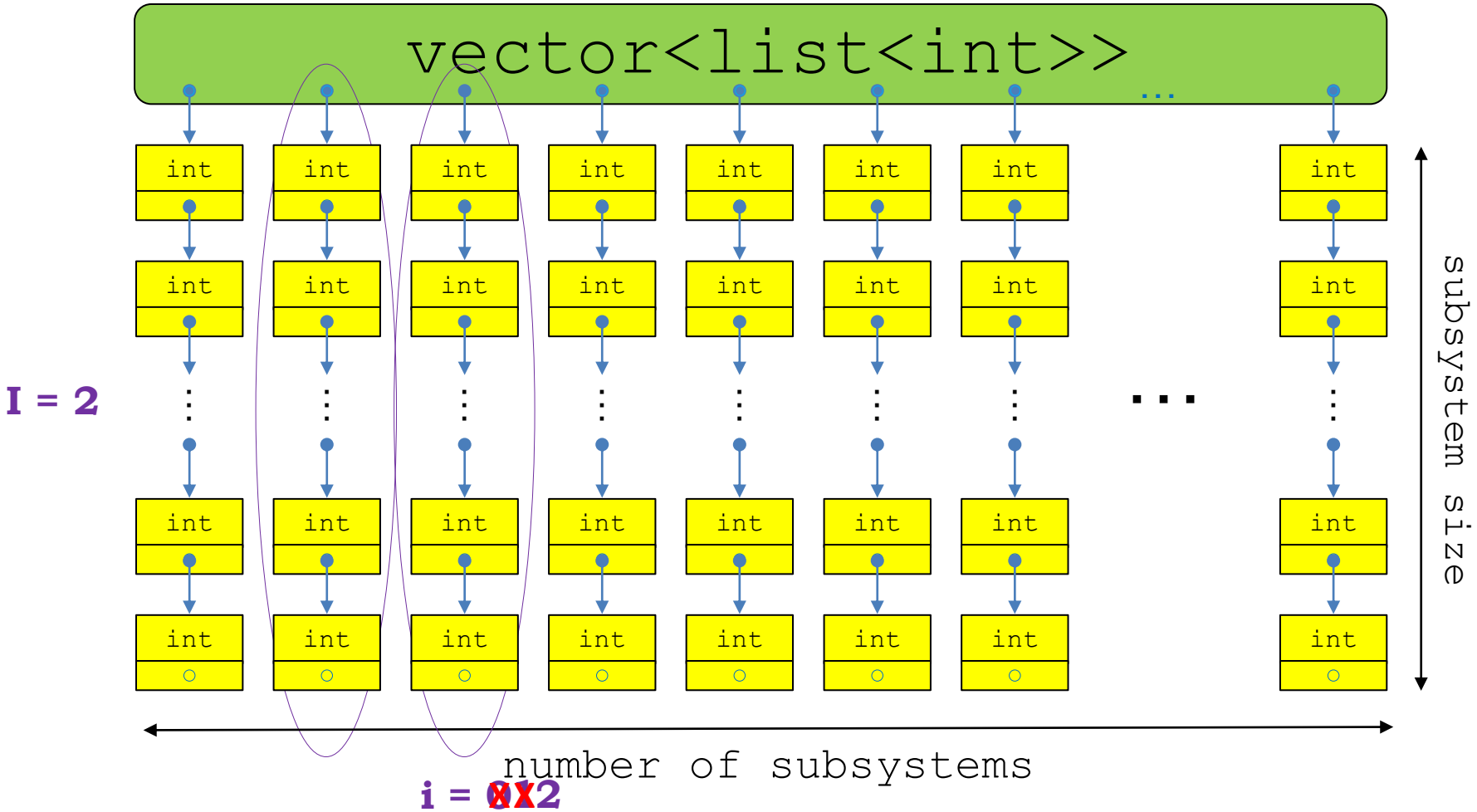
Access Plan:

- Visit each subsystem in turn.
- For each subsystem
  - Write to each of the `int` values in order.
  - ***Repeat*** the sequence of writes for a total of **I** iterations.
  - Advance to the next subsystem.
- ***Repeat*** the entire process for a total of **R** repetitions.
- The result of this experiment is (wall-clock) **RUN TIME.**
  - No memory is allocated or deallocated.

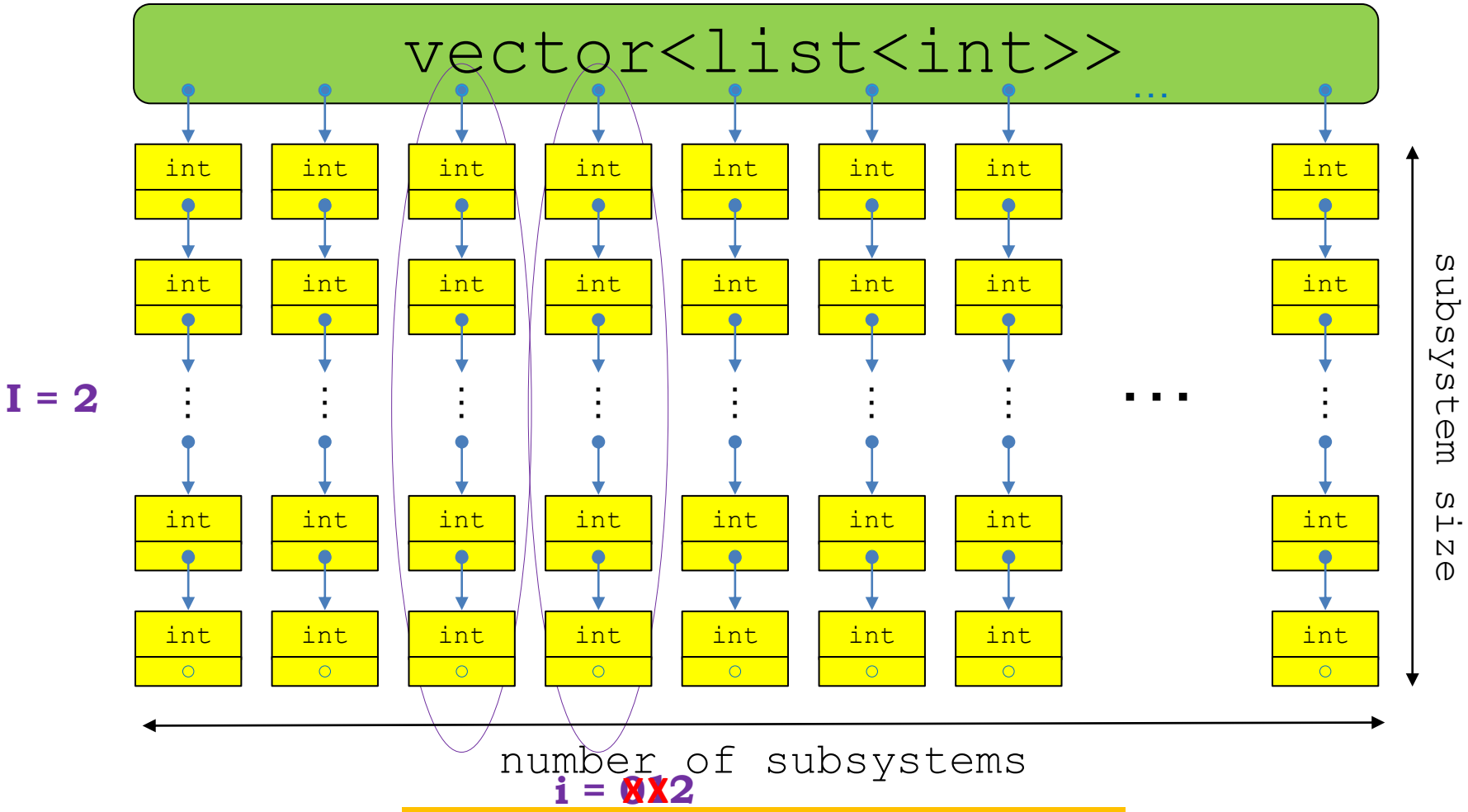
Benchmark II: LOCALITY



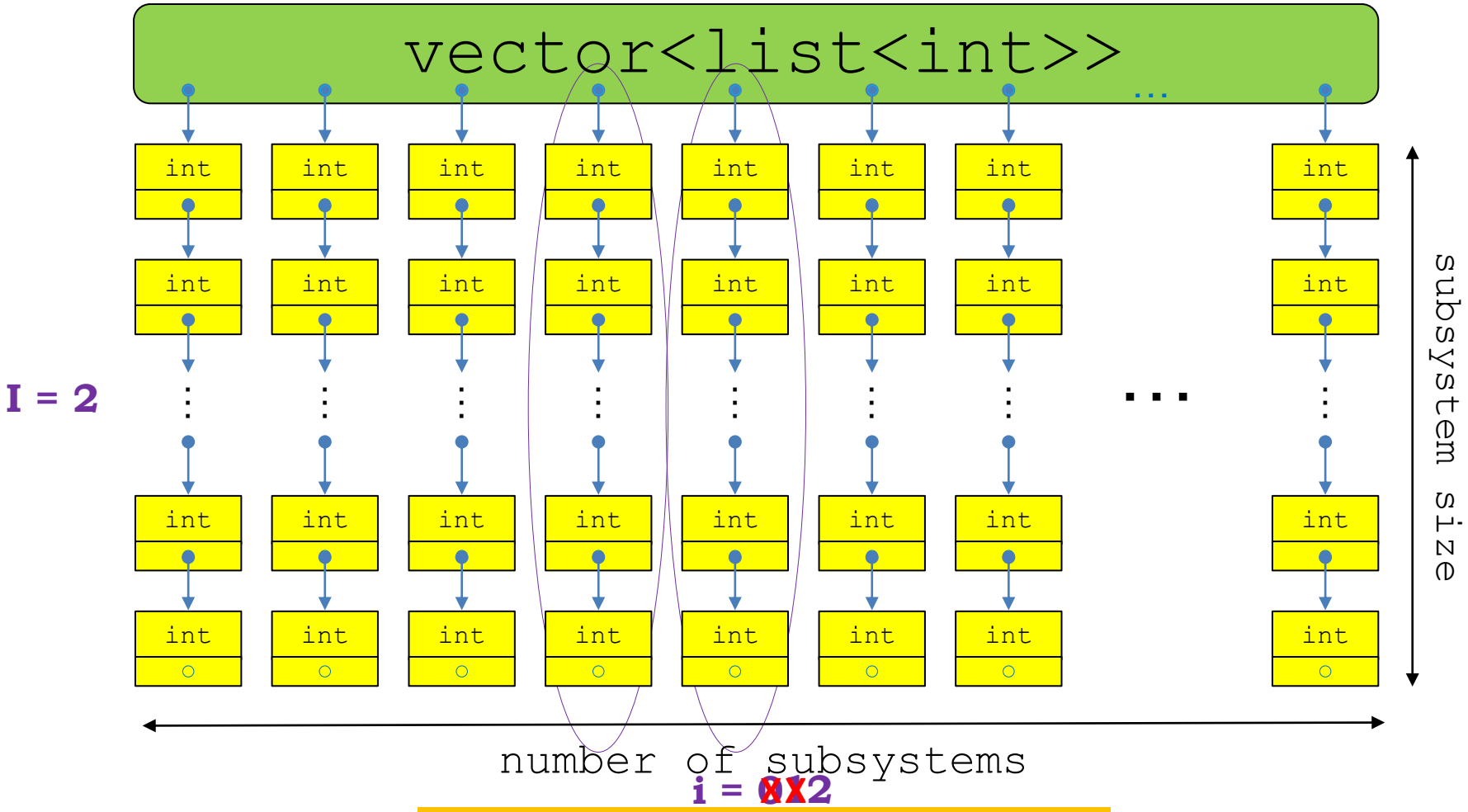
Benchmark II: LOCALITY



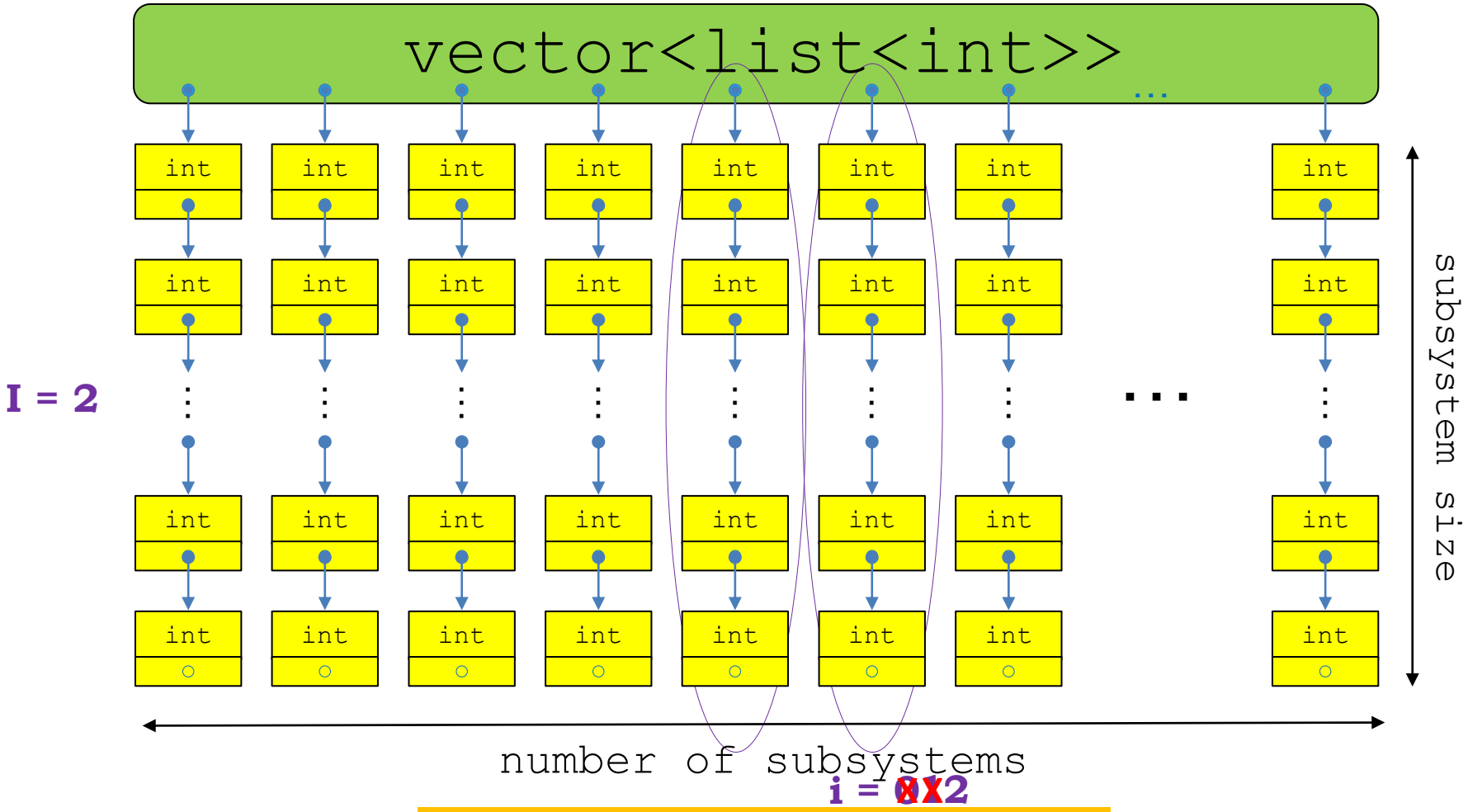
Benchmark II: LOCALITY



Benchmark II: LOCALITY

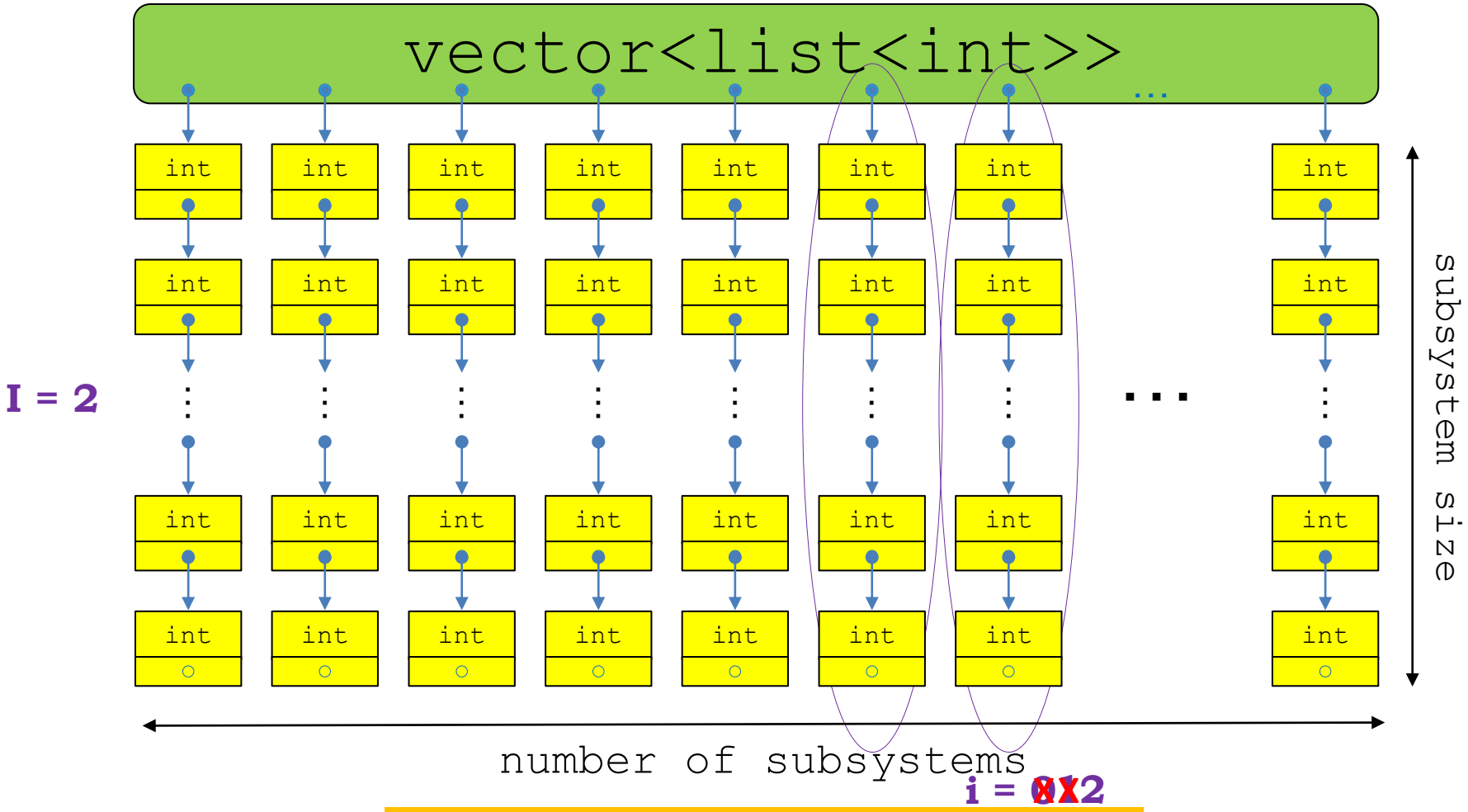


Benchmark II: LOCALITY



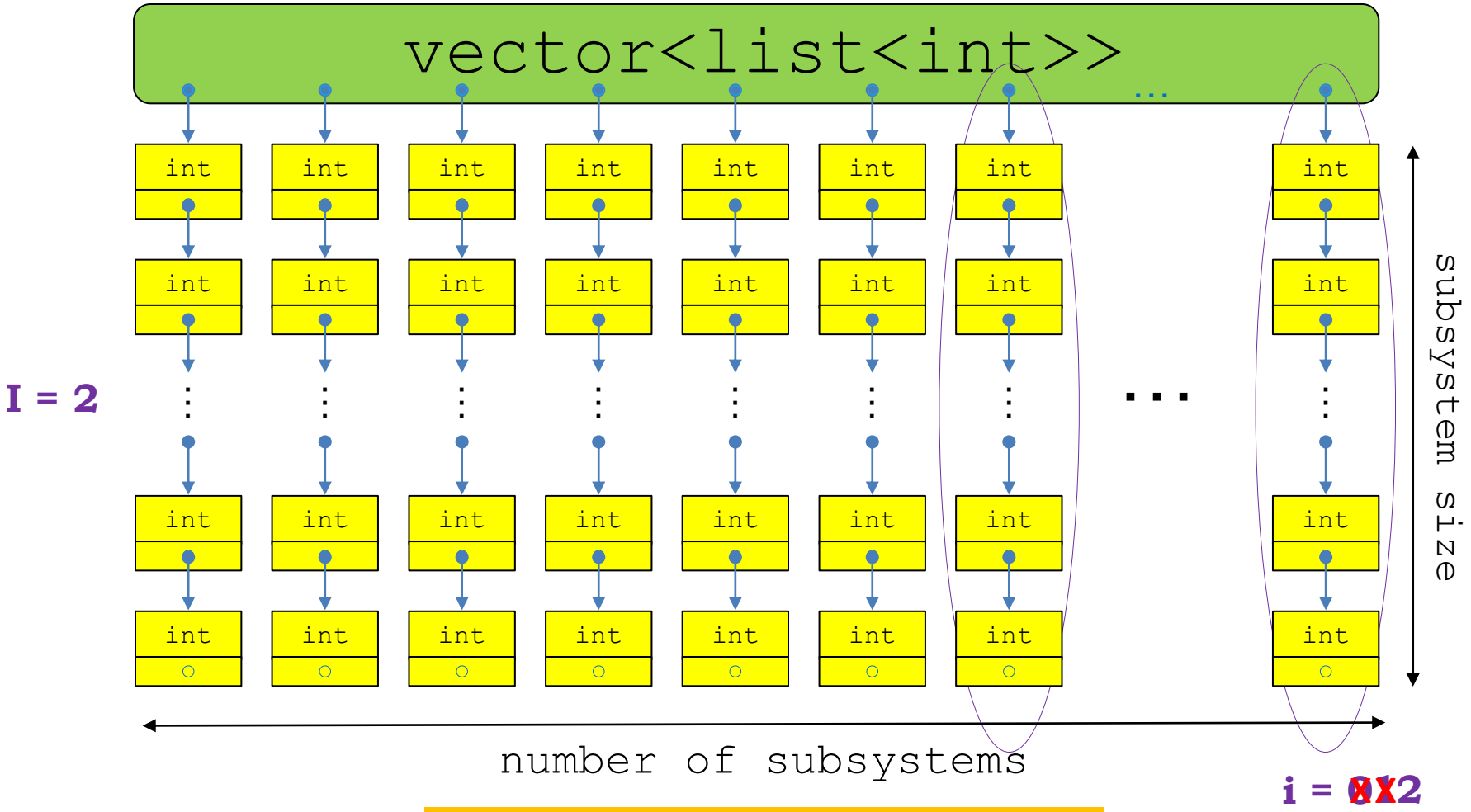
Sixth Subsystem Access Complete

Benchmark II: LOCALITY



Seventh Subsystem Access Complete

Benchmark II: LOCALITY



Last Subsystem Access Complete

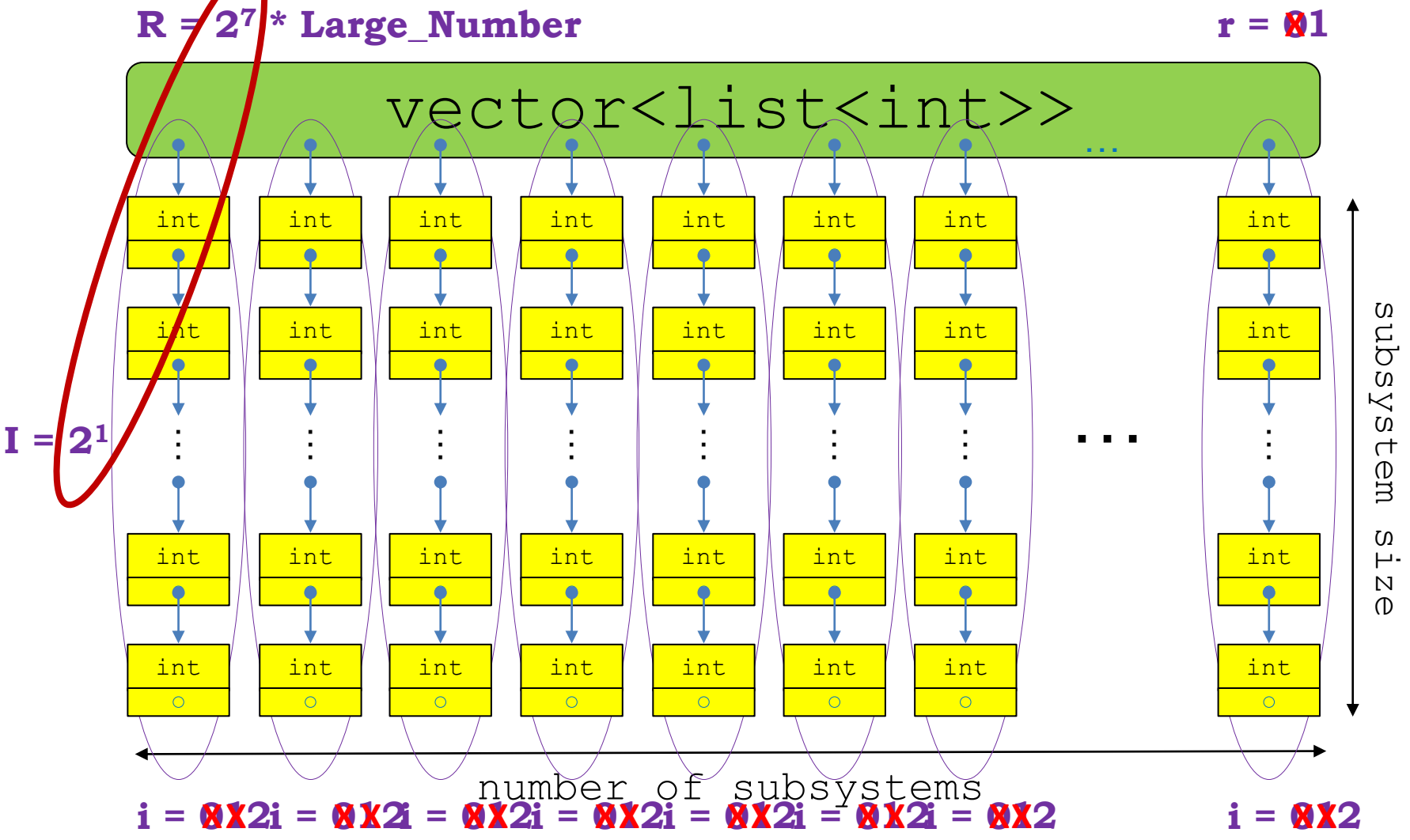
Benchmark II: LOCALITY

Access Plan:

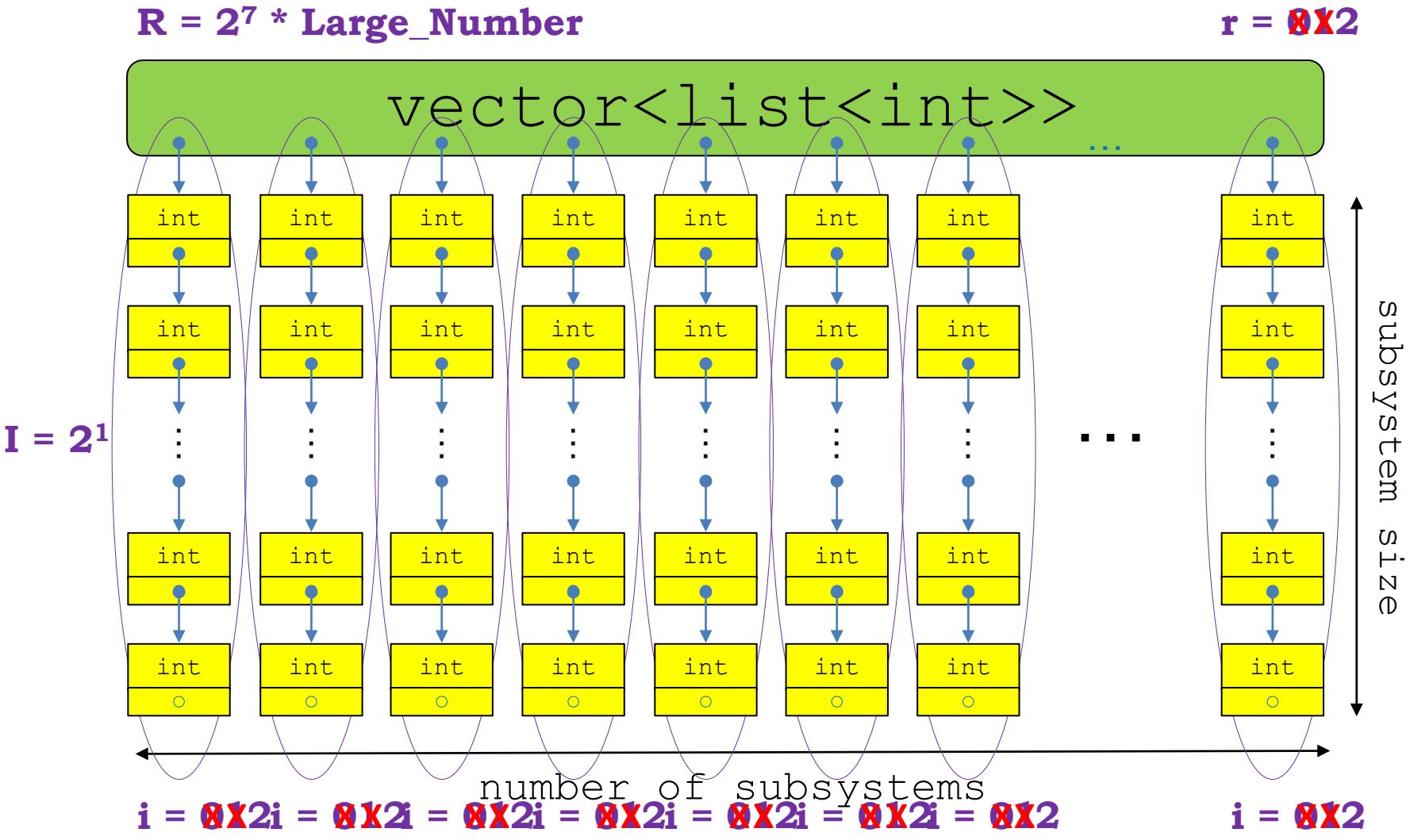
- Visit each subsystem in turn.
- For each subsystem
  - Write to each of the `int` values in order.
  - ***Repeat*** the sequence of writes for a total of **I** iterations.
  - Advance to the next subsystem.
- ***Repeat*** the entire process for a total of **R** repetitions.
- The result of this experiment is (wall-clock) **RUN TIME.**
  - No memory is allocated or deallocated.

Benchmark II: LOCALITY

$2^1 * 2^7 = 2^8$

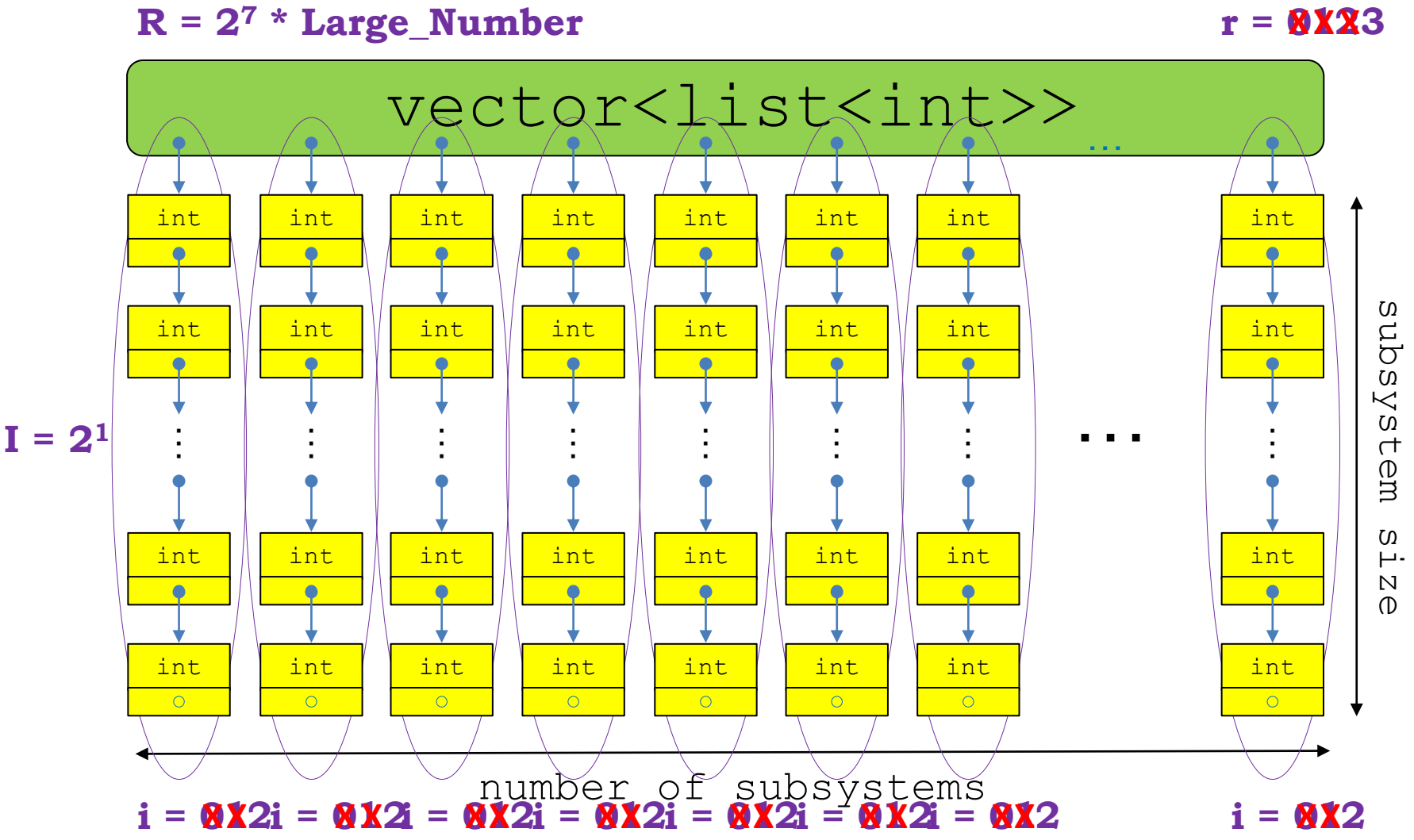


Benchmark II: LOCALITY



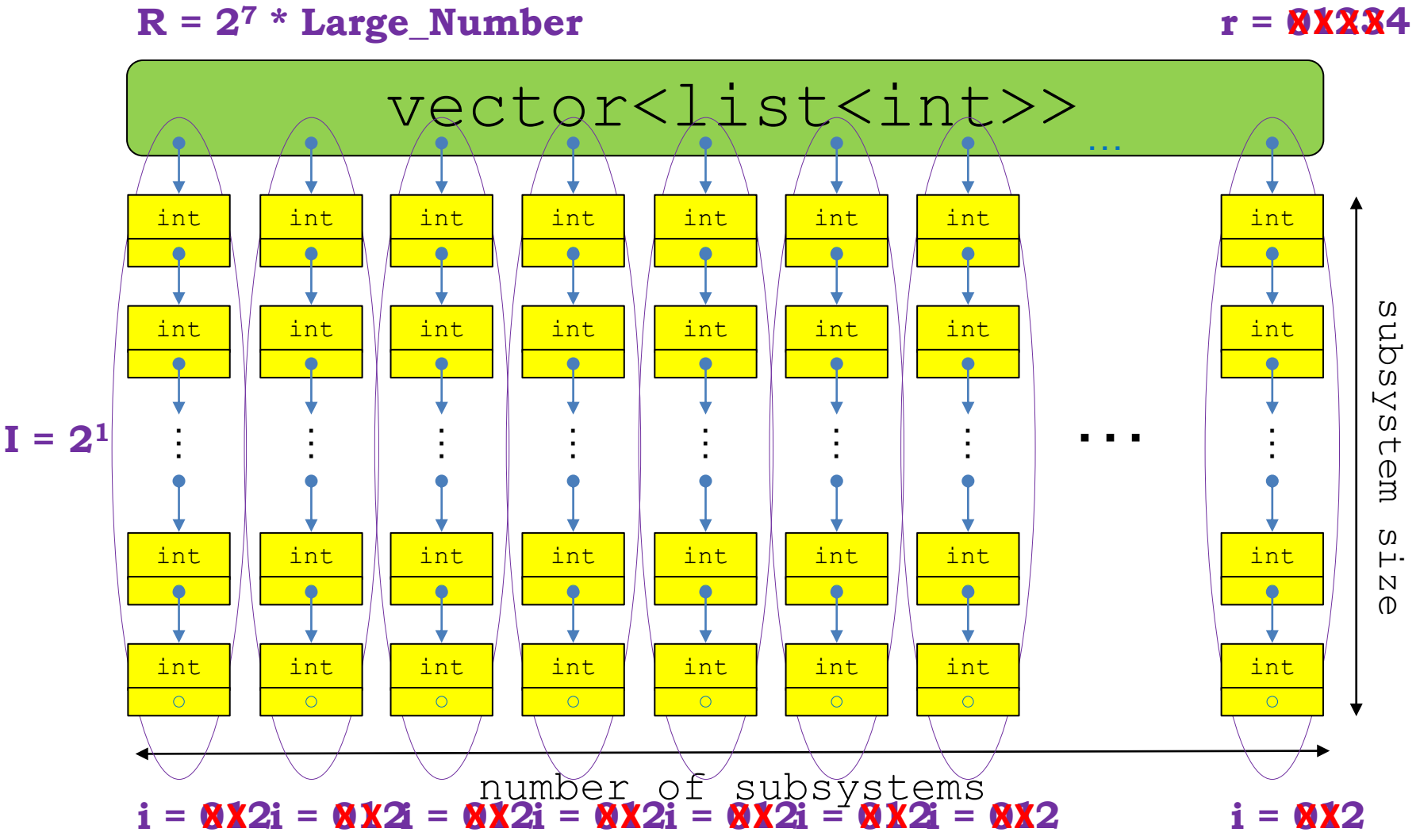
3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



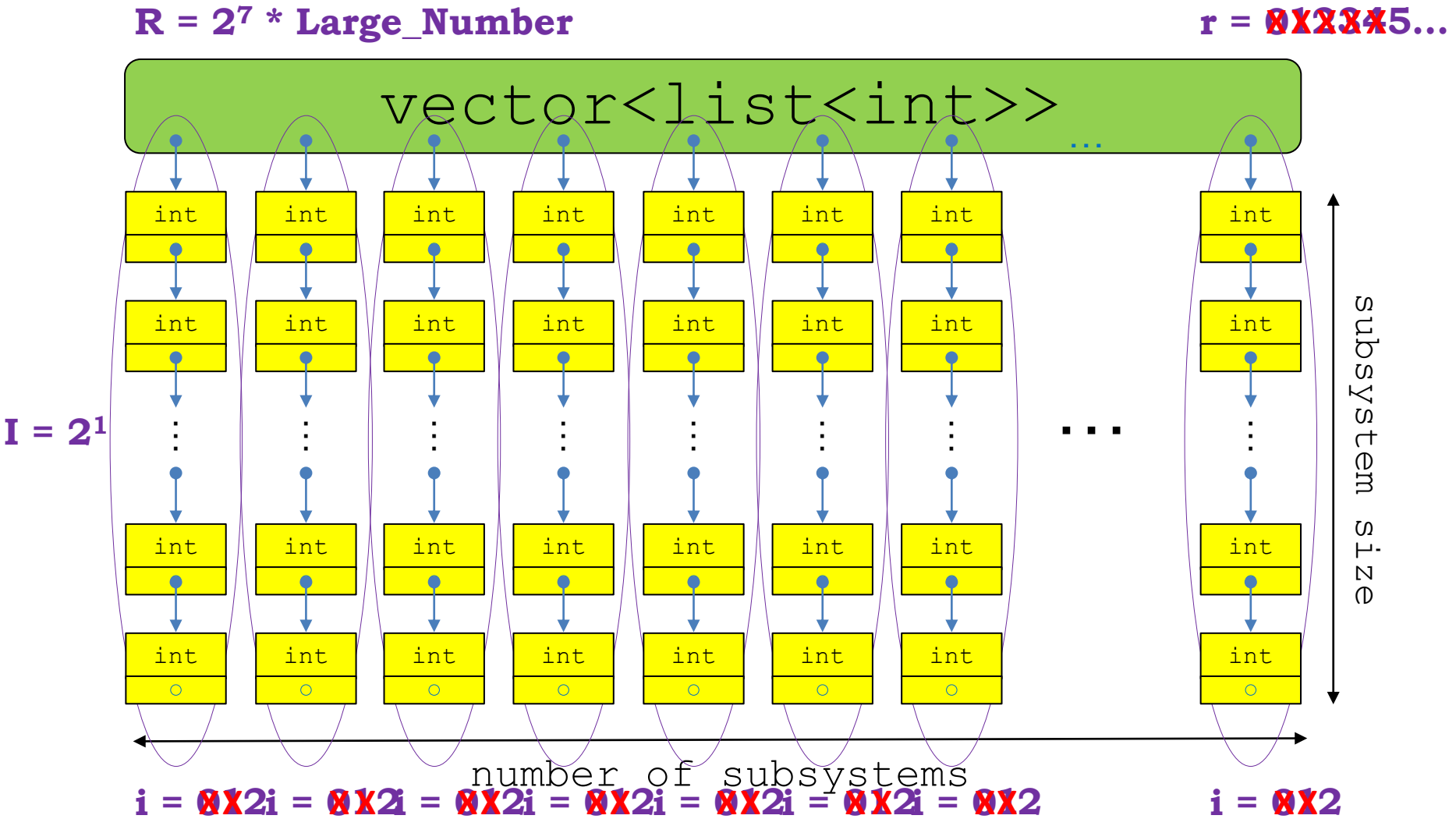
3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Access Plan:

- Visit each subsystem in turn.
- For each subsystem
  - Write to each of the `int` values in order.
  - ***Repeat*** the sequence of writes for a total of **I** iterations.
  - Advance to the next subsystem.
- ***Repeat*** the entire process for a total of **R** repetitions.
- The result of this experiment is (wall-clock) **RUN TIME.**
  - No memory is allocated or deallocated.

Benchmark II: LOCALITY

Pseudo Code

```
// using namespace std;

vector<list<int>> system(Num_Subsystems)
```

Benchmark II: LOCALITY

```
vector<list<int>>
```

Benchmark II: LOCALITY

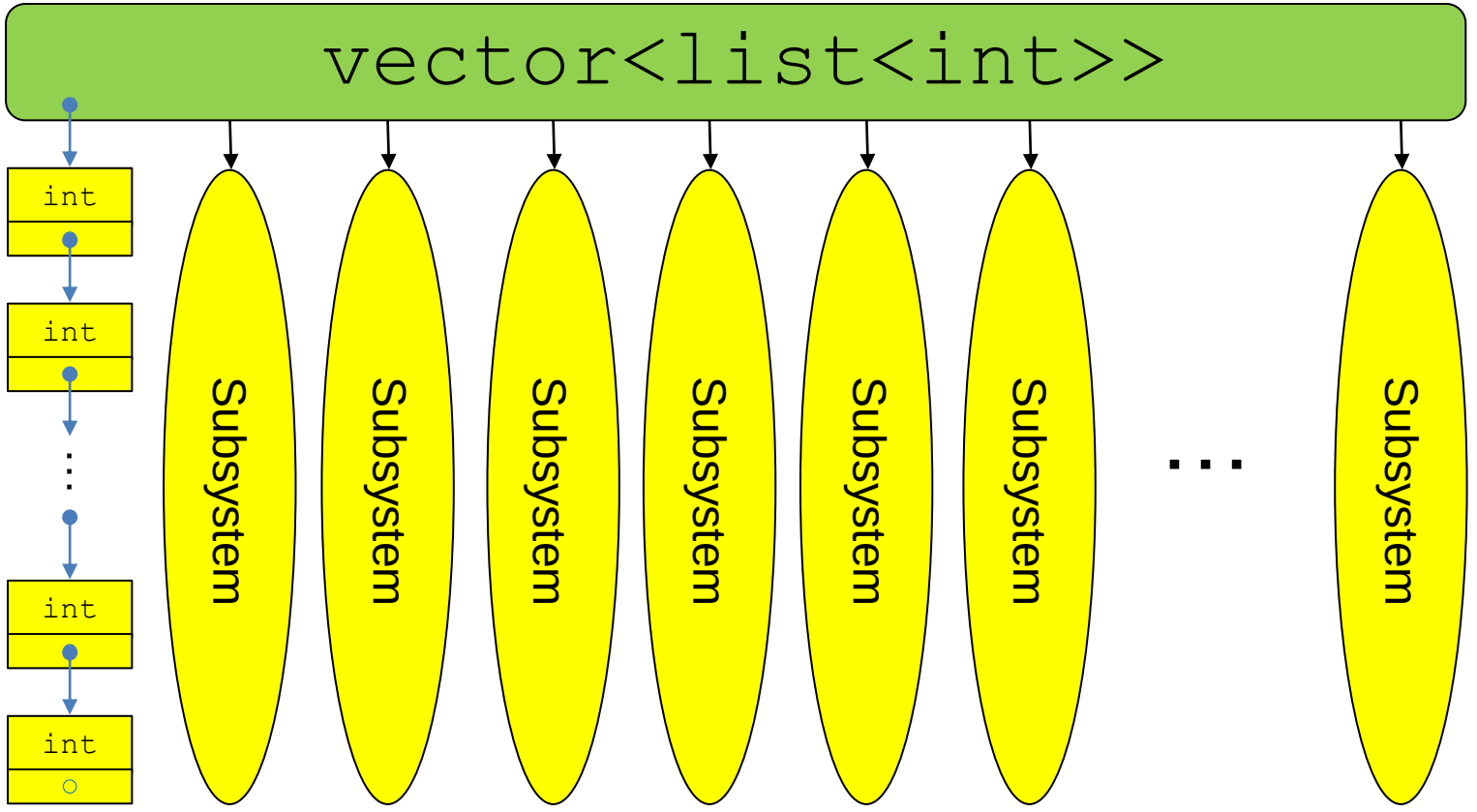
Pseudo Code

```
// using namespace std;

vector<list<int>> system(Num_Subsystems)

// Build up each list to have Subsystem_Size links.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Pseudo Code

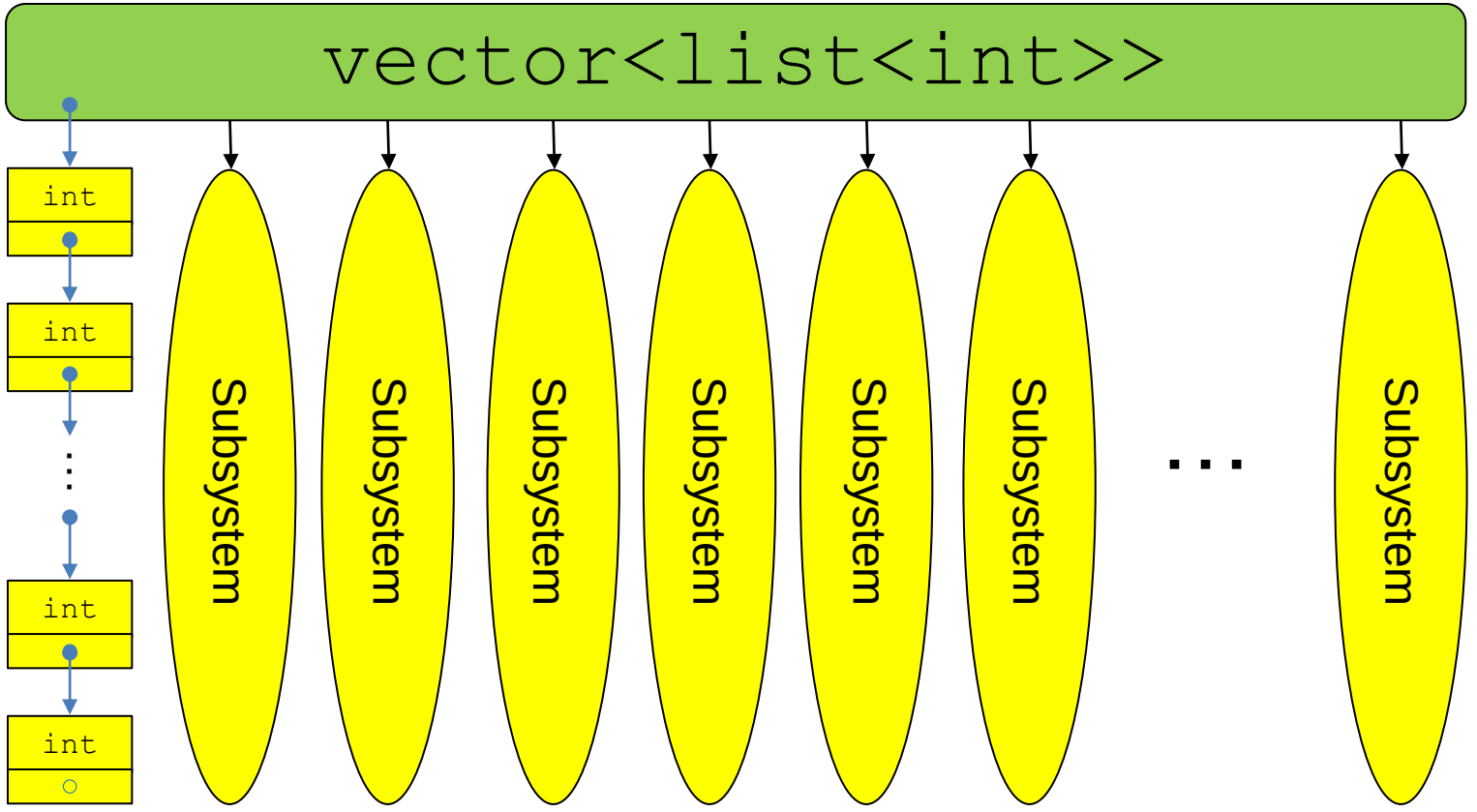
```
// using namespace std;

vector<list<int>> system(Num_Subsystems)

// Build up each list to have Subsystem_Size links.
```

```
++*e;  // Read/write current subsystem element.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Pseudo Code

```
// using namespace std;

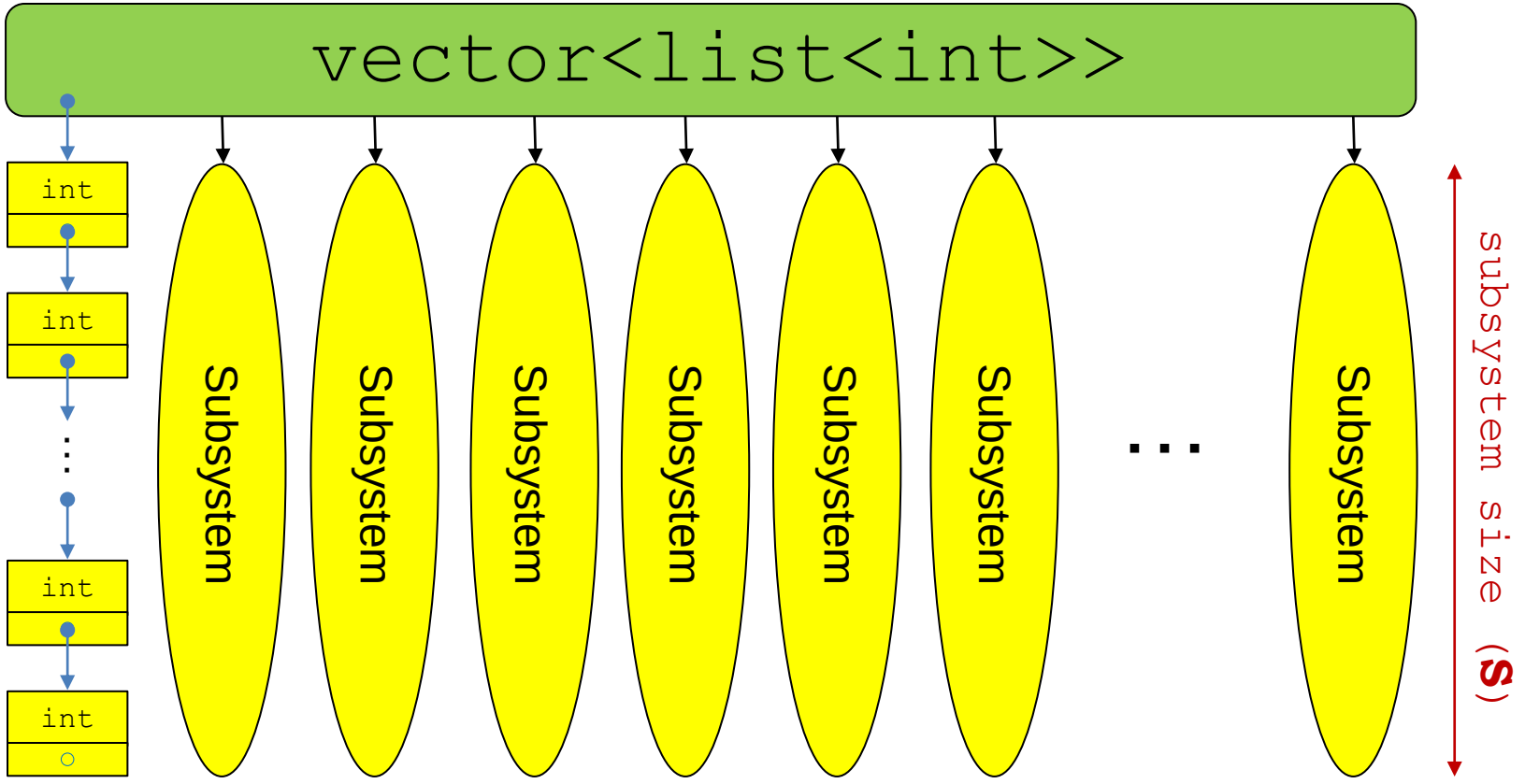
vector<list<int>> system(Num_Subsystems)

// Build up each list to have Subsystem_Size links.
```

```
for (int& e : system[k])

  ++*e;  // Read/write current subsystem element.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Pseudo Code

```
// using namespace std;

vector<list<int>> system(Num_Subsystems)

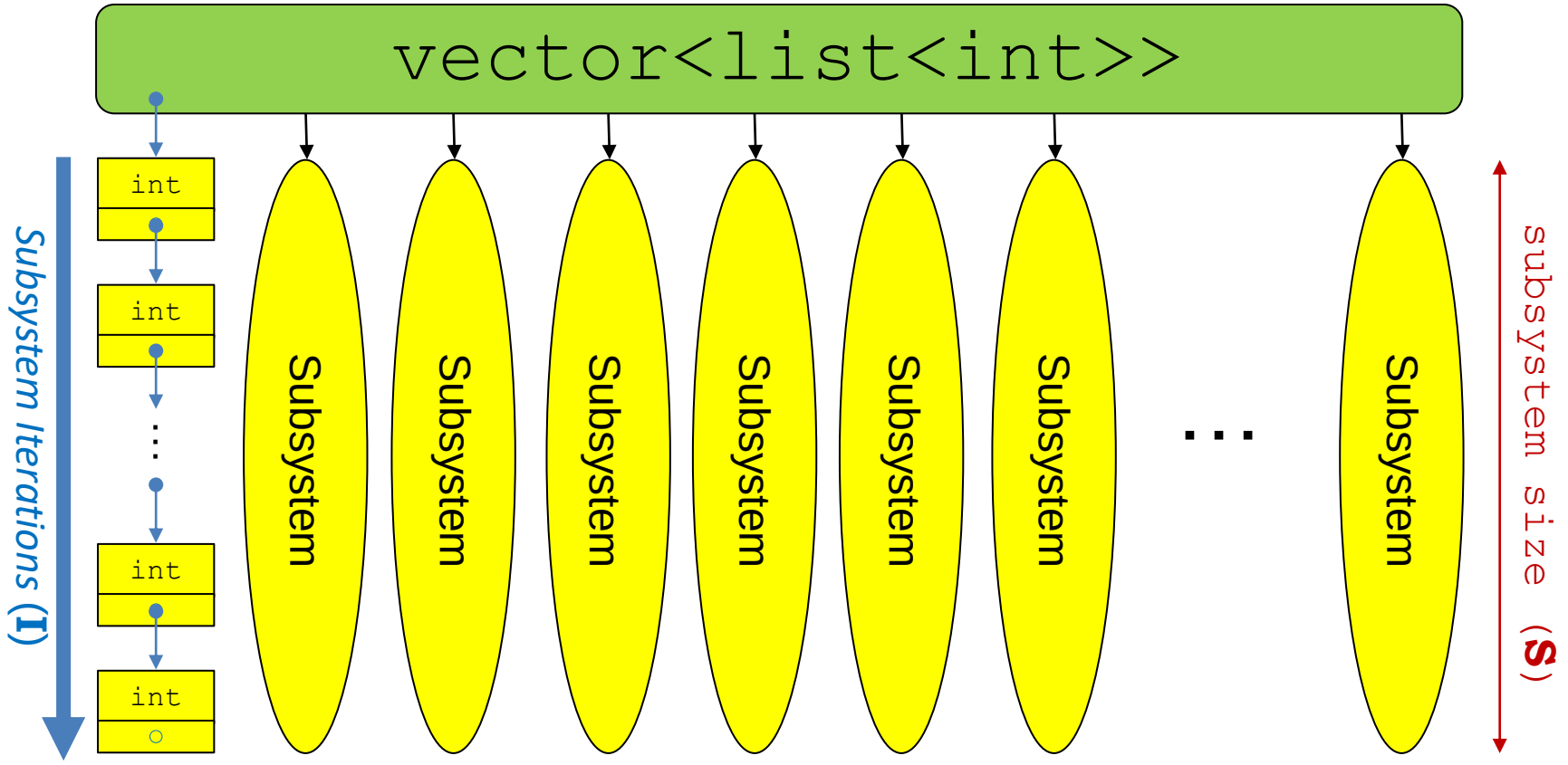
// Build up each list to have Subsystem_Size links.
```

```
    for (int i = 0; i < Num_Subsystem_Iterations; ++i)

      for (int& e : system[k])

        ++*e;  // Read/write current subsystem element.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Pseudo Code

```
// using namespace std;

vector<list<int>> system(Num_Subsystems)

// Build up each list to have Subsystem_Size links.

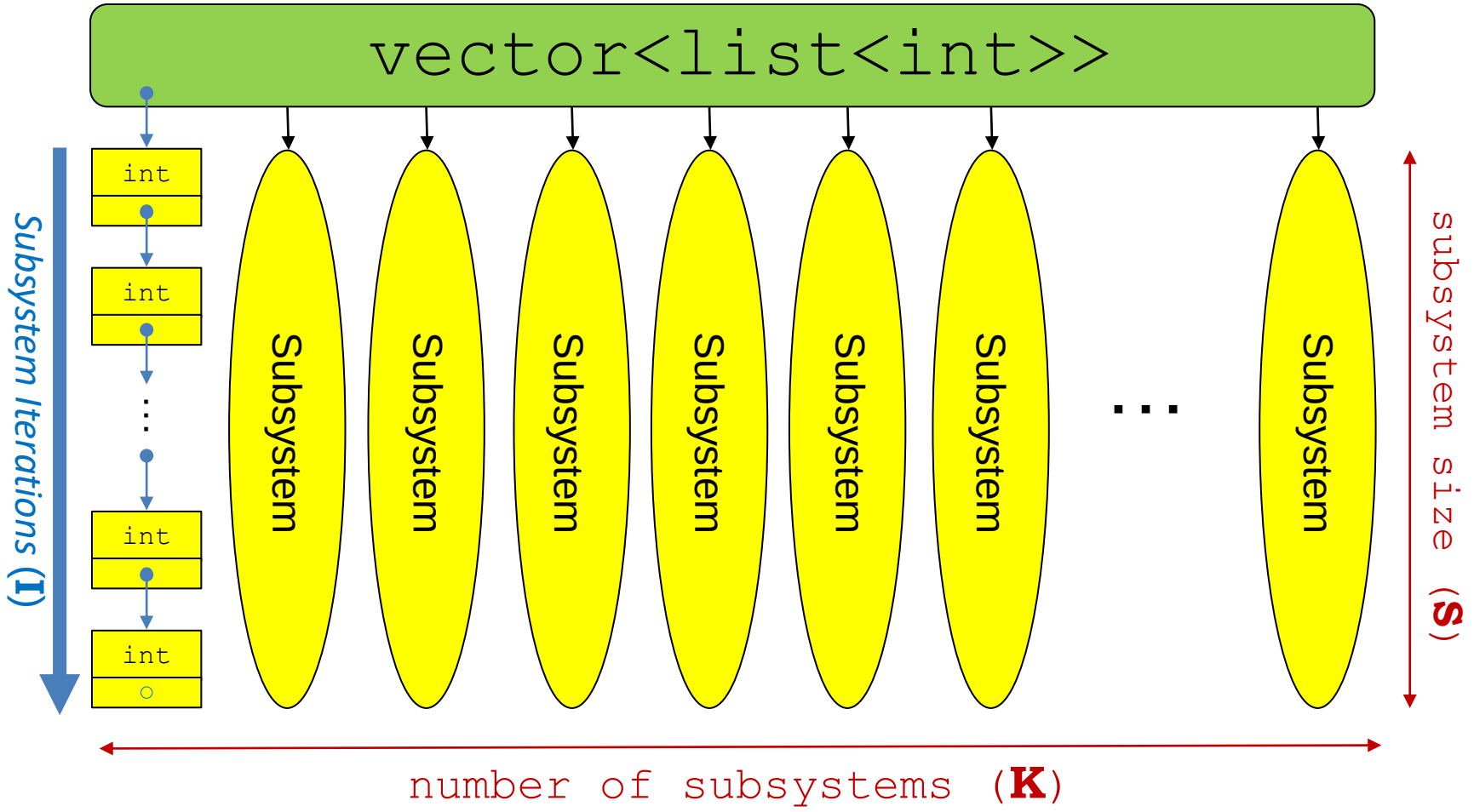

  for (int k = 0; k < Num_Subsystems; ++k)

    for (int i = 0; i < Num_Subsystem_Iterations < ++i)

      for (int& e : system[k])

        ++*e;  // Read/write current subsystem element.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY

Pseudo Code

```
// using namespace std;

vector<list<int>> system(Num_Subsystems)

// Build up each list to have Subsystem_Size links.

for (int r = 0; r < Num_System_Repetitions; ++r)

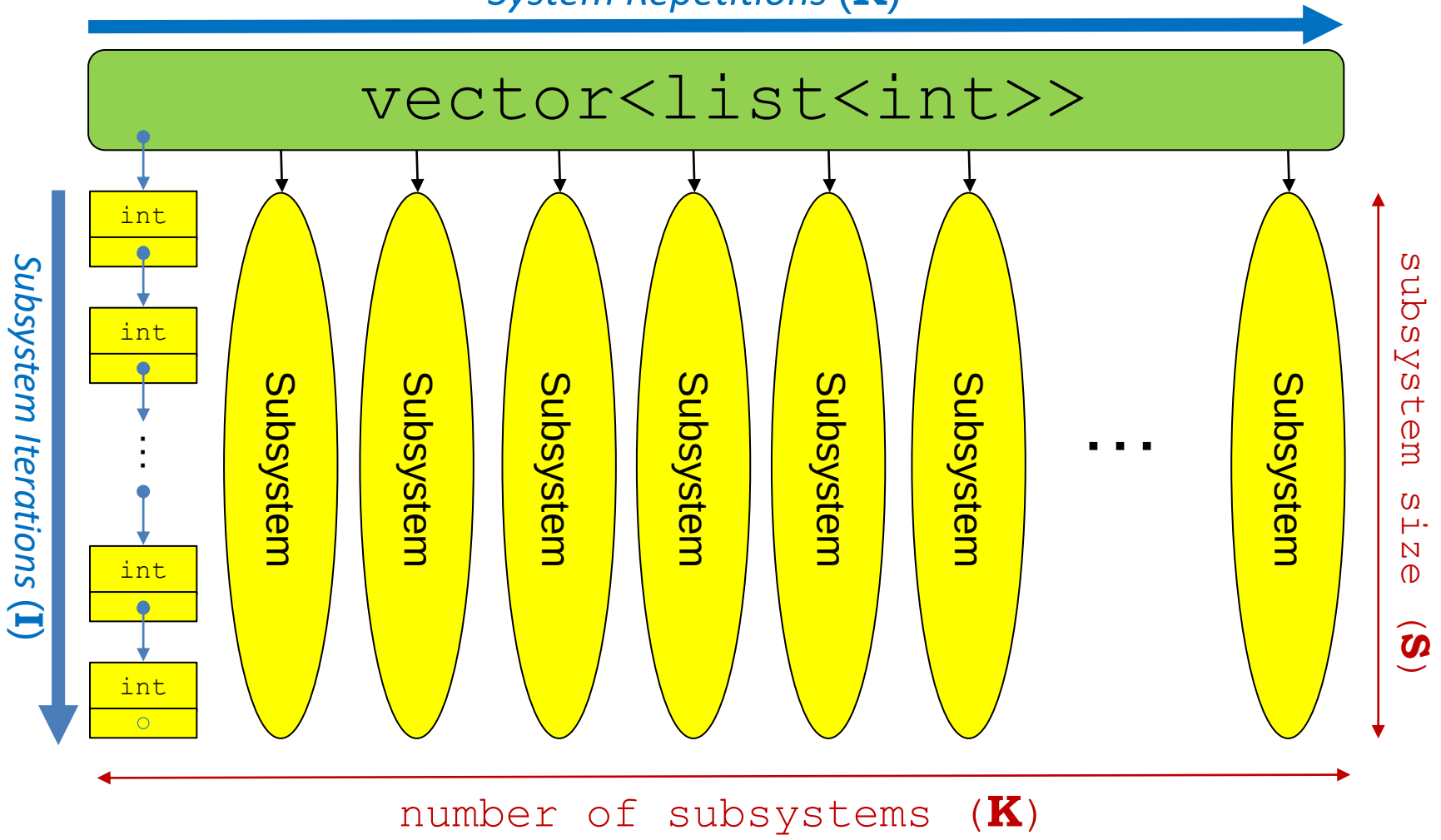
  for (int k = 0; k < Num_Subsystems; ++k)

    for (int i = 0; i < Num_Subsystem_Iterations < ++i)

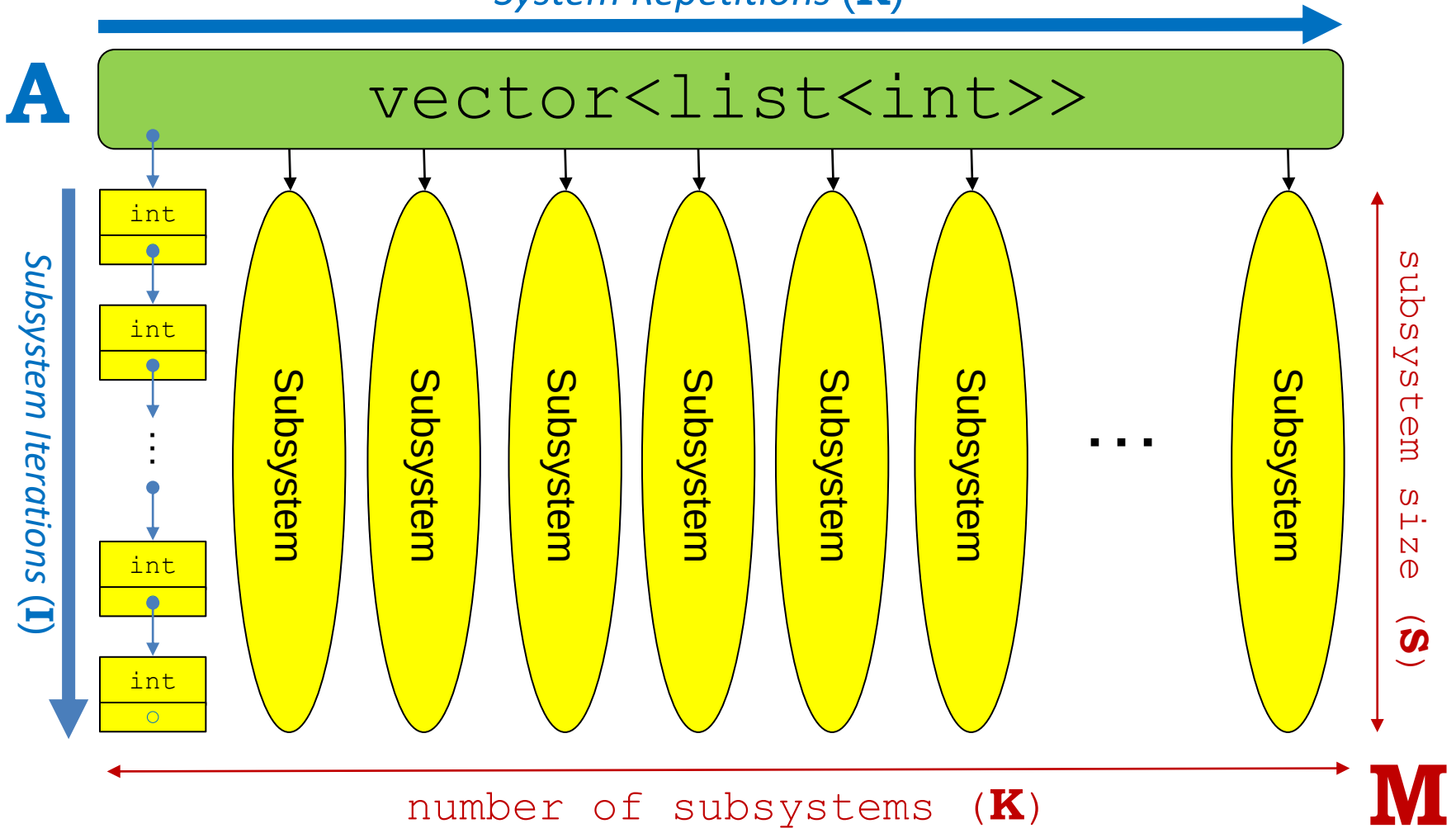
      for (int& e : system[k])

        ++*e;  // Read/write current subsystem element.
```

Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY

System Parameters

- ***Total*** *Number of Instructions Executed* (**N**)
  - **N** = *System Memory Size* (**M**) * *Access Factor* (**A**)
    - *(number of system elements * accesses per element)*
- ***Physical*** *System Memory Size* (**M**)
  - **M** = *subsystem size* (**S**) * *number of subsystems* (**K**)
- ***Temporal*** *Access Factor* (**A**)
  - **A** = *subsystem iterations* (**I**) * *system repetitions* (**R**)

Benchmark II: LOCALITY

Trading off *physical* (**M**) and *temporal* (**A**) values.

(N = 45) = **(M = 21)** * **(A = 24)**

Implies 2^{45} (more than 32 trillion) *ACCESS* instructions executed.

Subsystem Size (**S**)	*Number of Subsystems* (**K**)
0	21
1	20
2	19
3	18
4	17
:	:
20	1
21	0

X

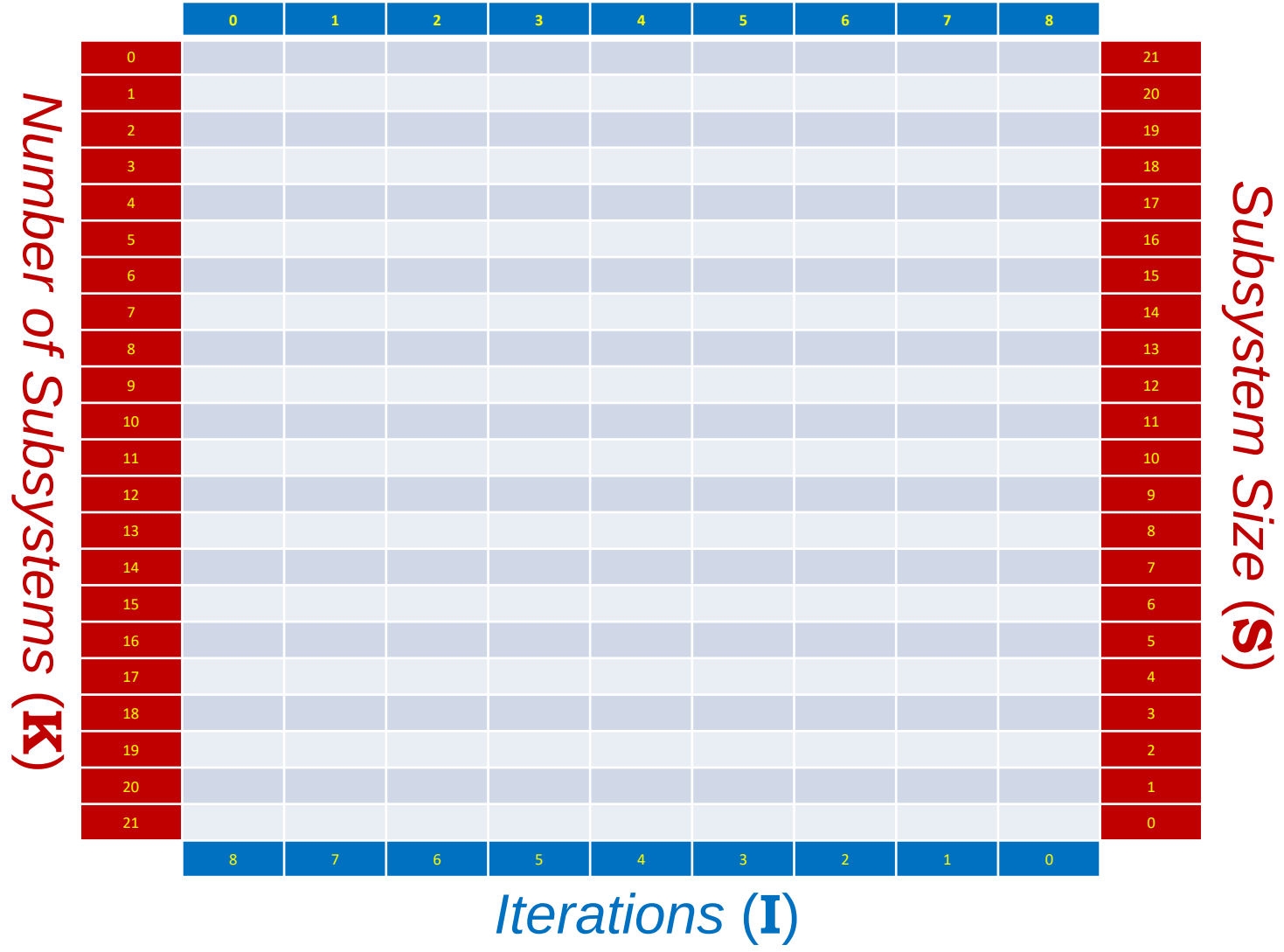
Number of Iterations (**I**)	*Number of Repetitions* (**R**)
8	16
7	17
6	18
5	19
4	20
3	21
2	22
1	23
0	24

340

3. Analyzing the Benchmark Data

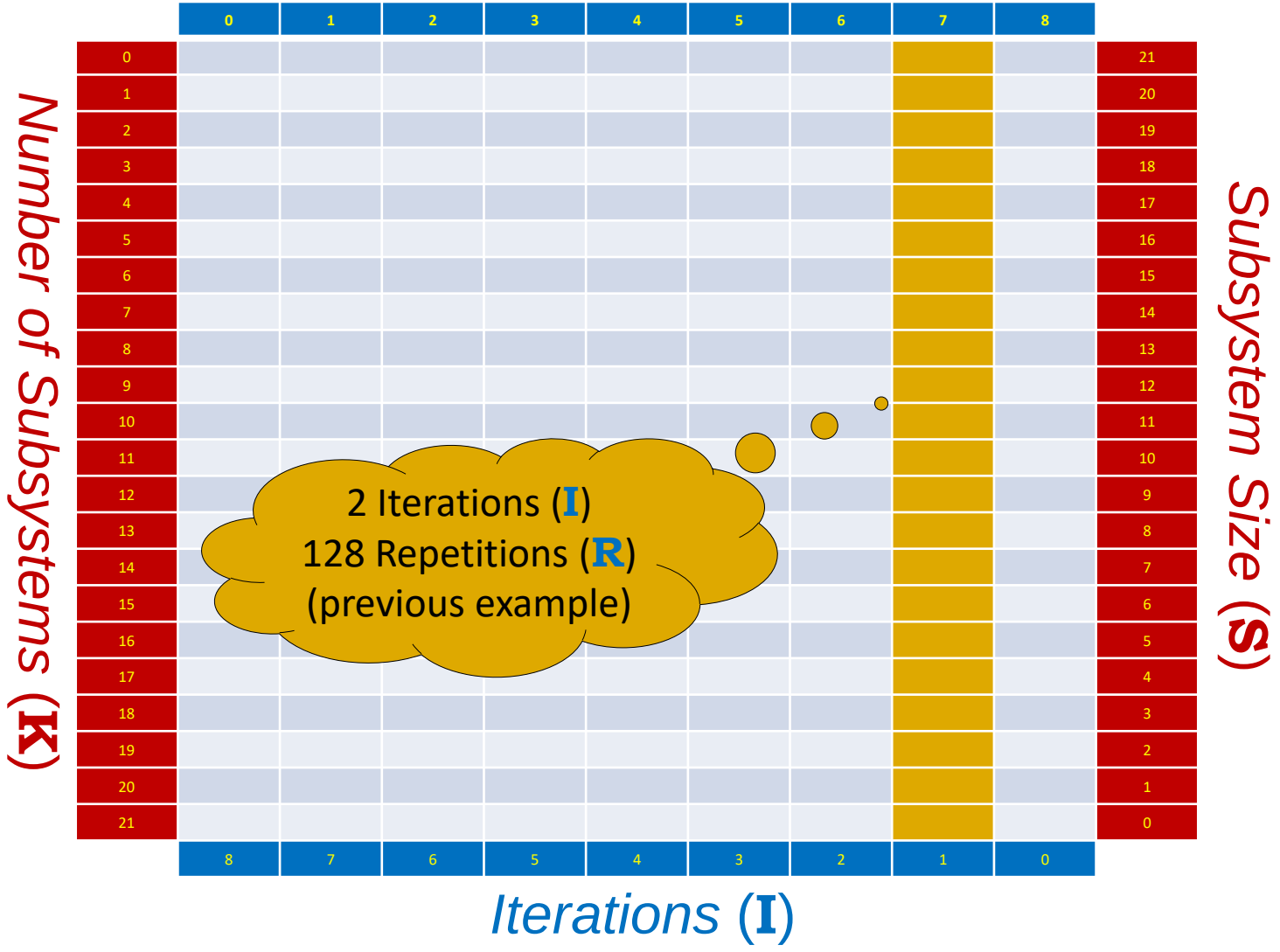
Benchmark II: LOCALITY

Large_Number * *Repetitions* (**R**)



Benchmark II: LOCALITY

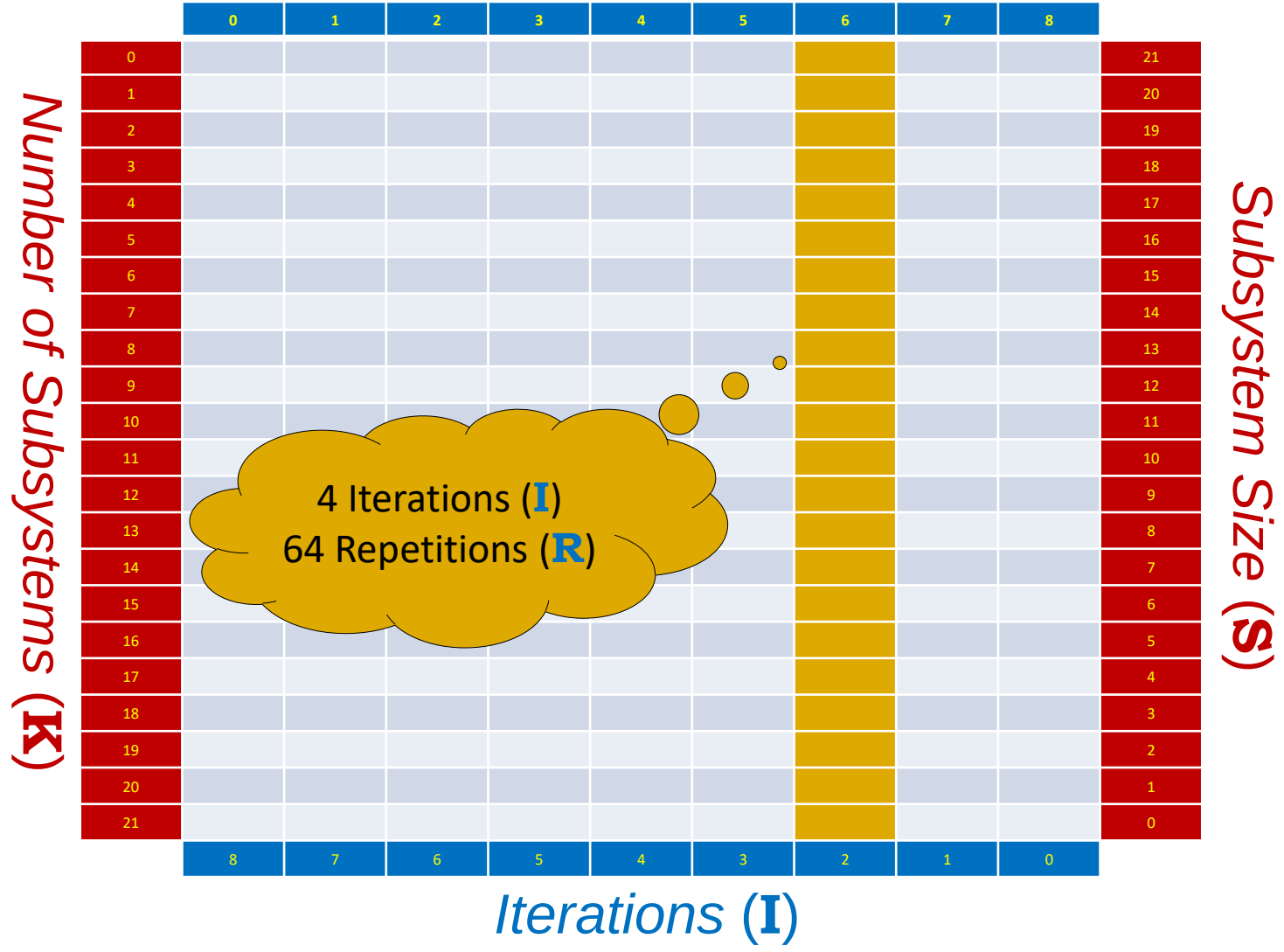
Large_Number * *Repetitions* (**R**)



3. Analyzing the Benchmark Data

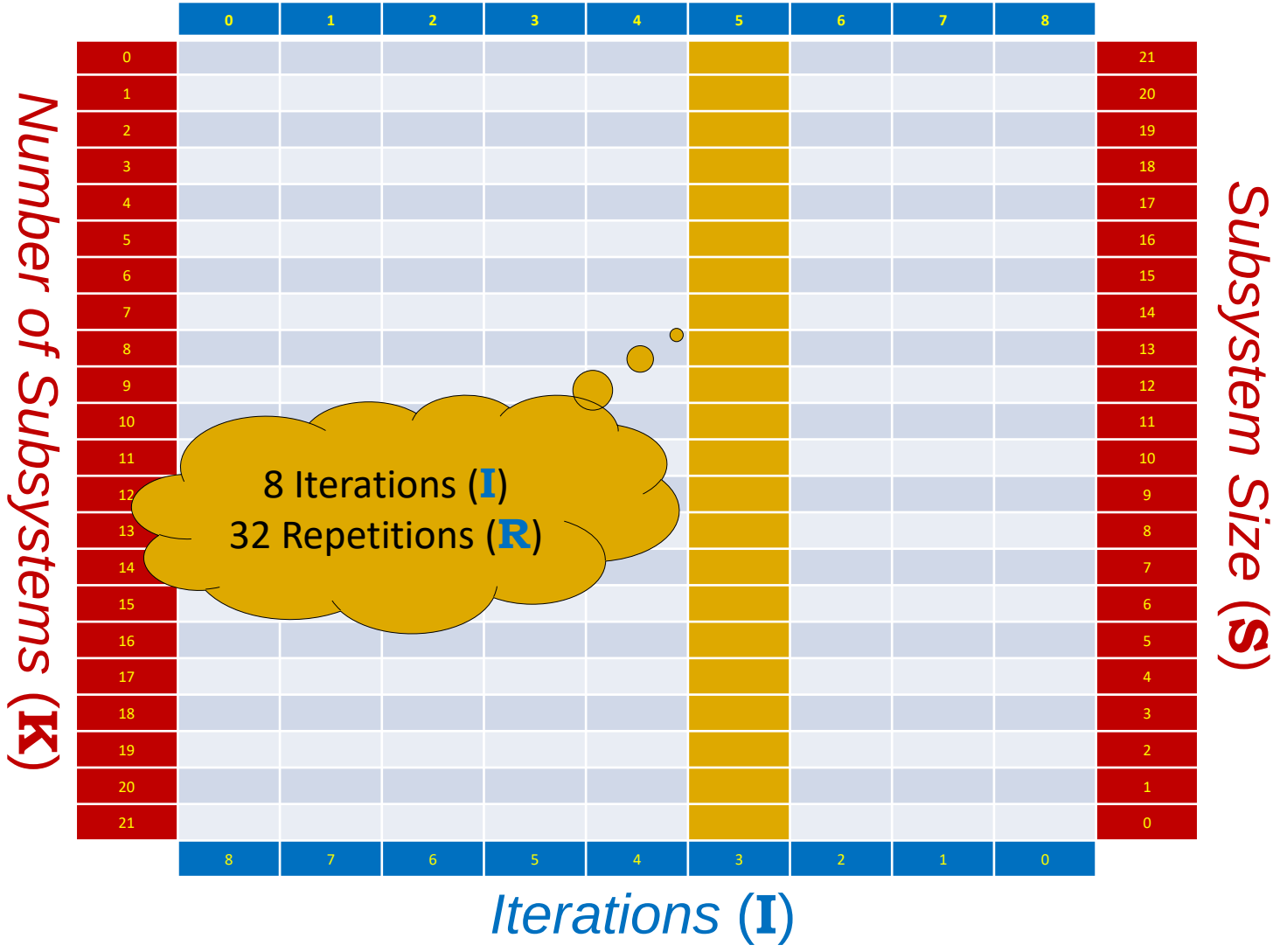
Benchmark II: LOCALITY

Large_Number * *Repetitions* (**R**)



Benchmark II: LOCALITY

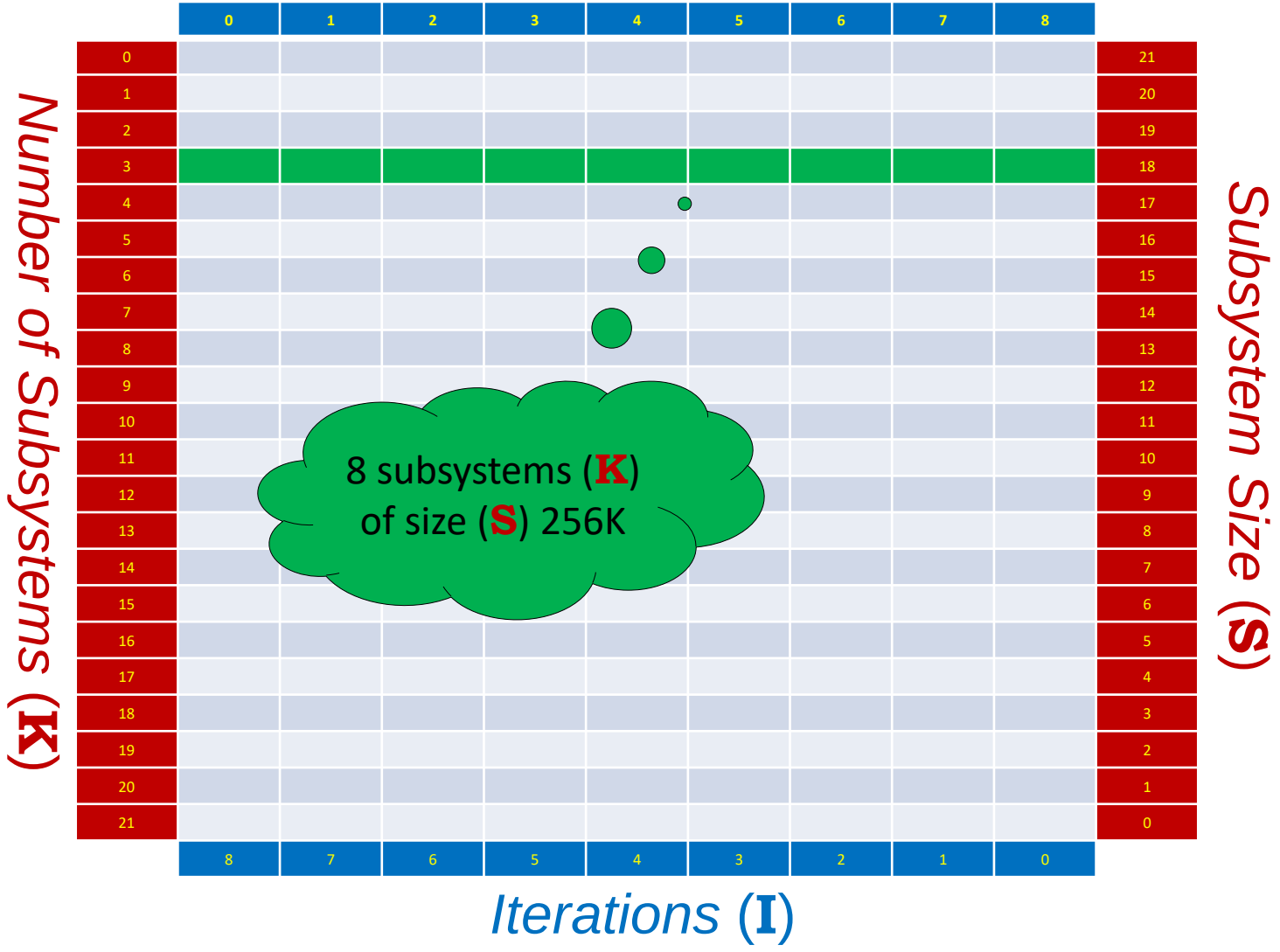
Large_Number * *Repetitions* (**R**)



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

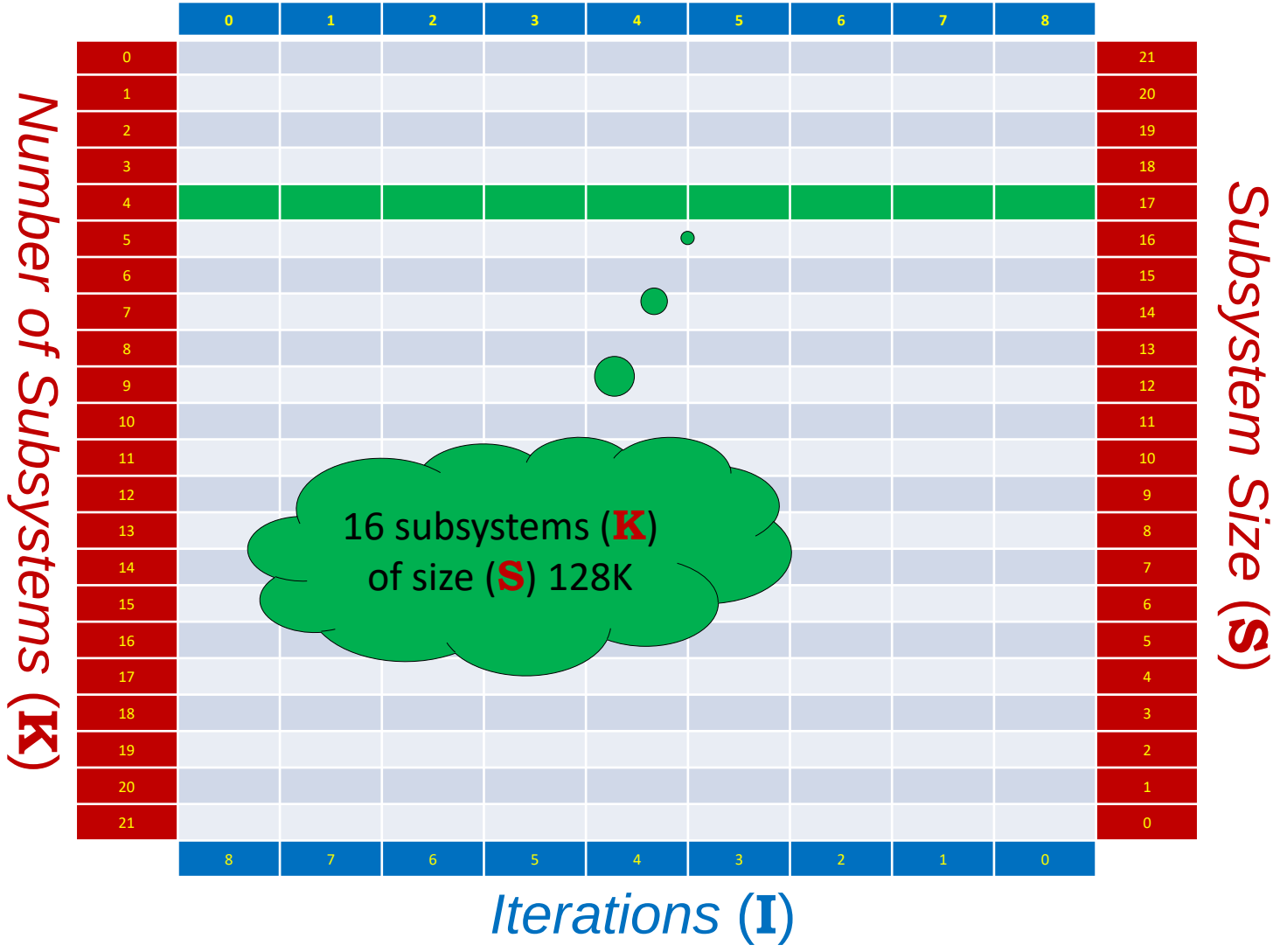
Large_Number * *Repetitions* (**R**)



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

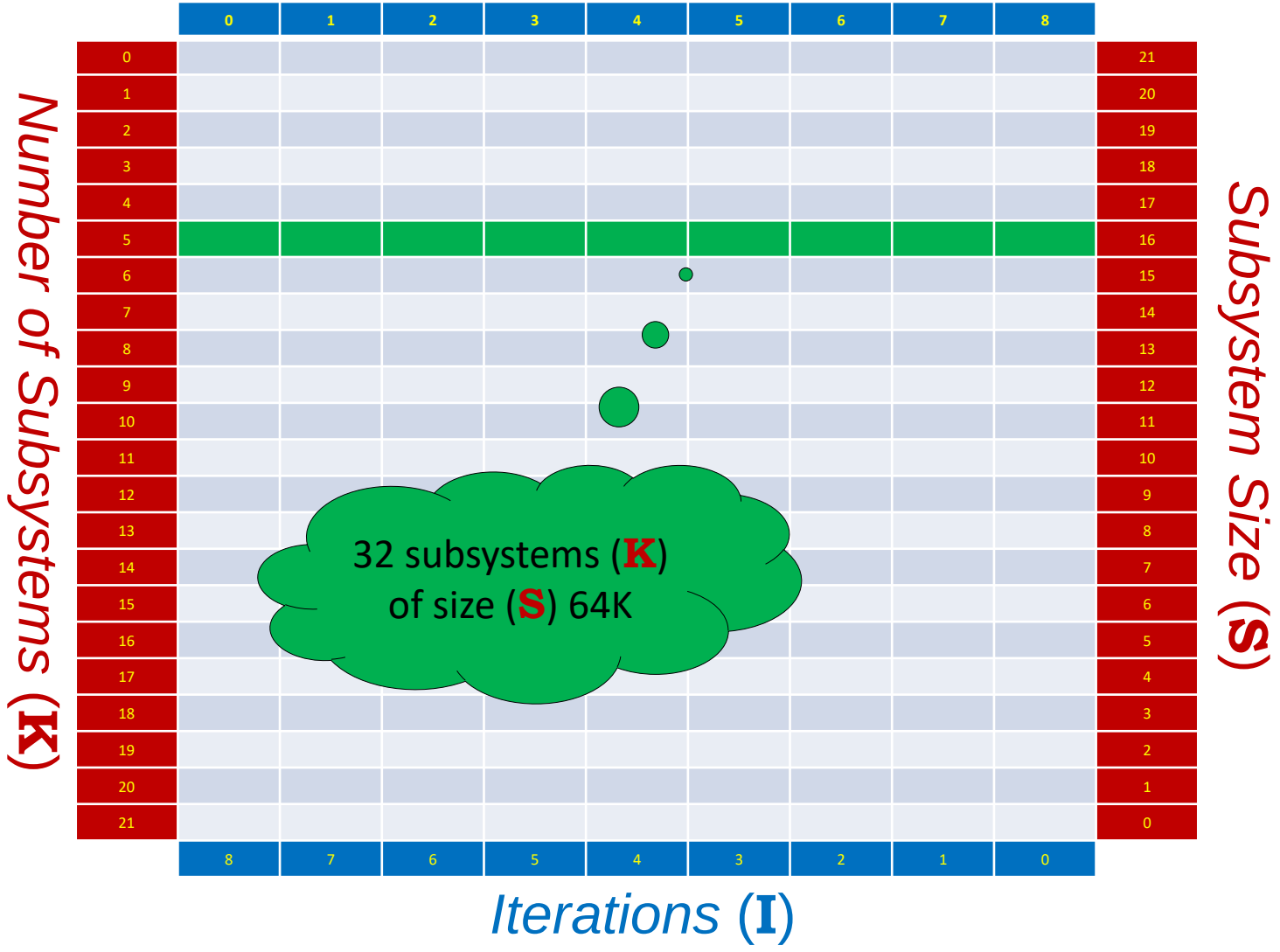
Large_Number * *Repetitions* (**R**)



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

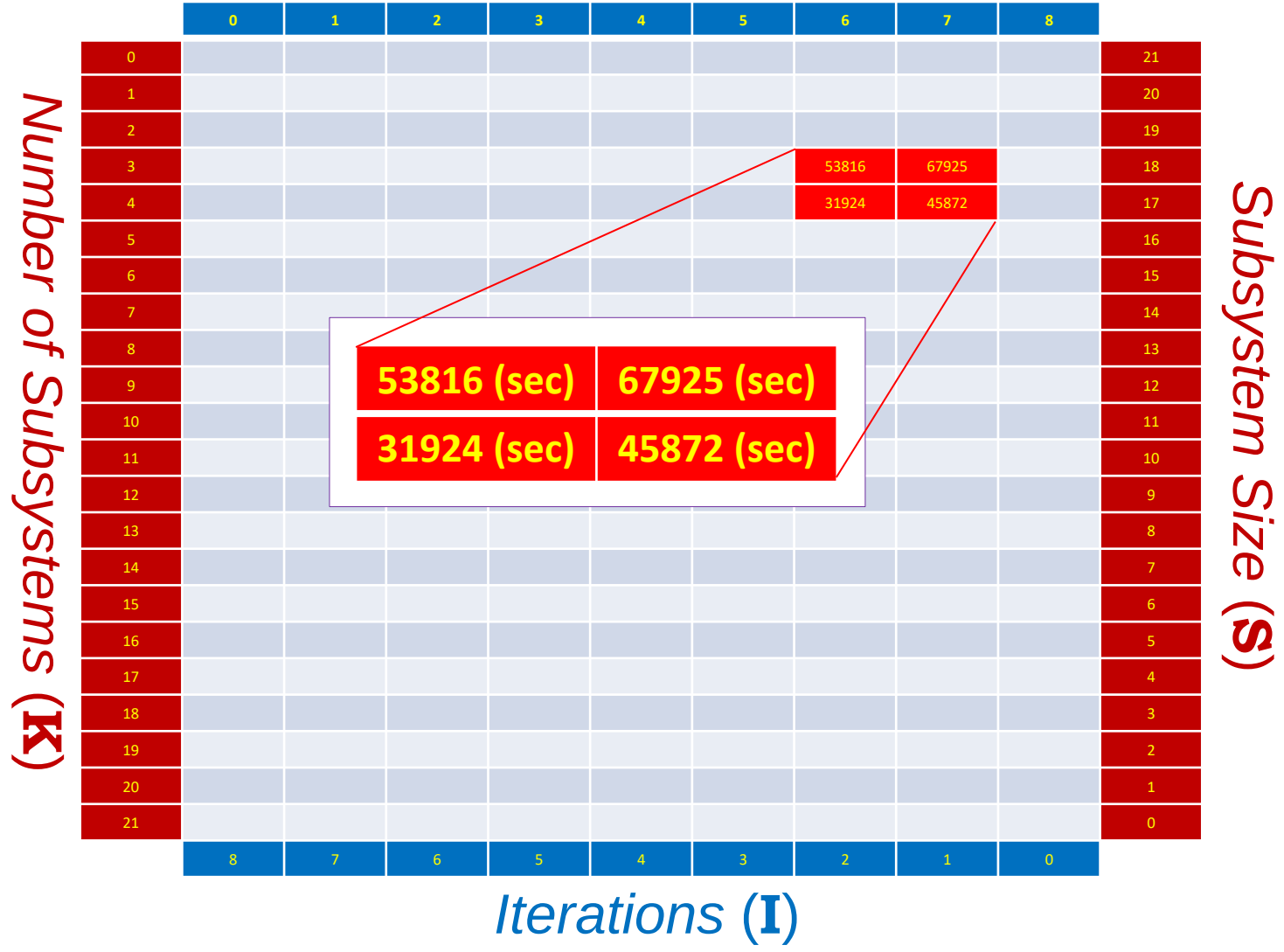
Large_Number * *Repetitions* (**R**)



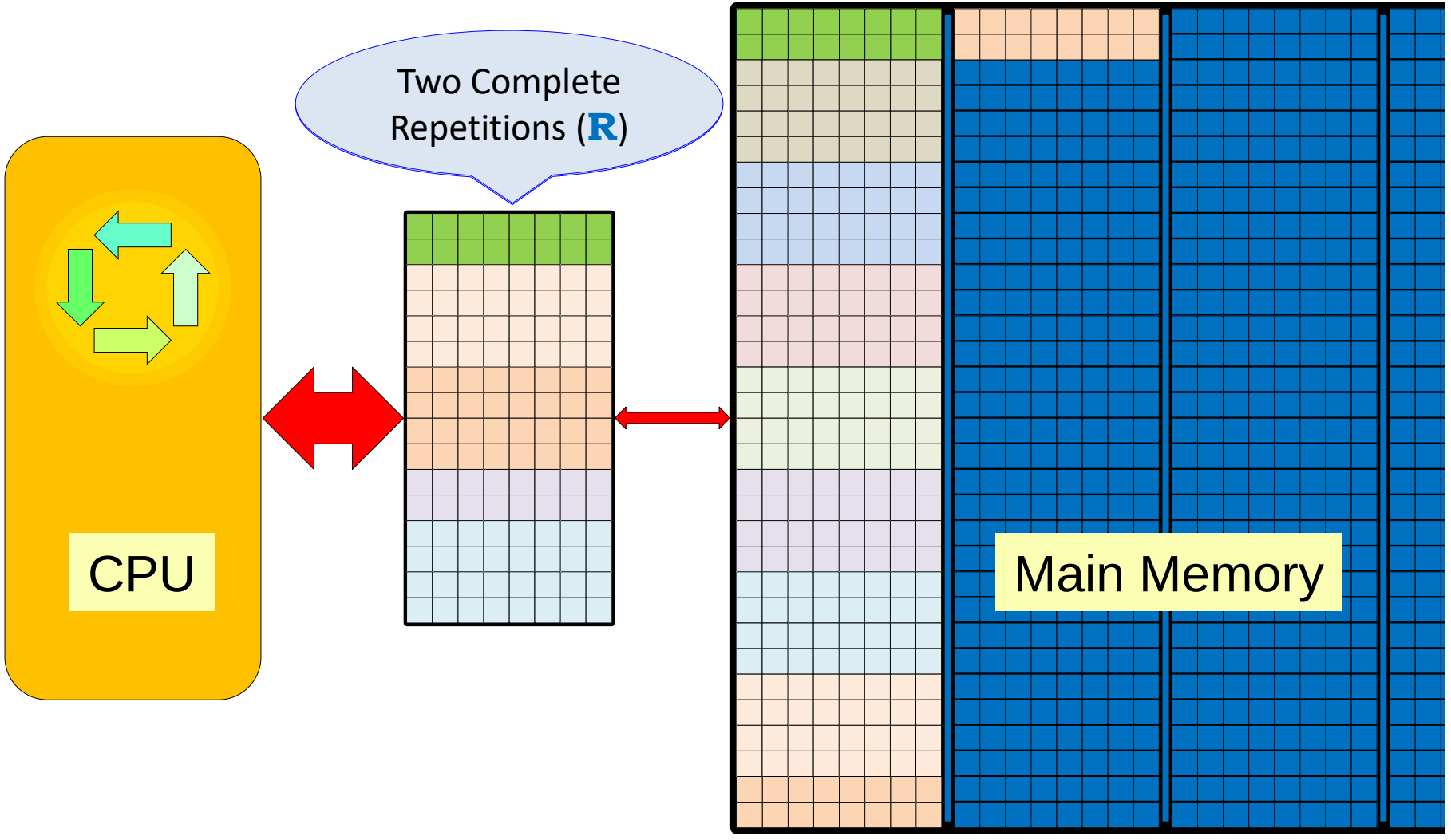
3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

Large_Number * *Repetitions* (**R**)



Benchmark II: LOCALITY



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY



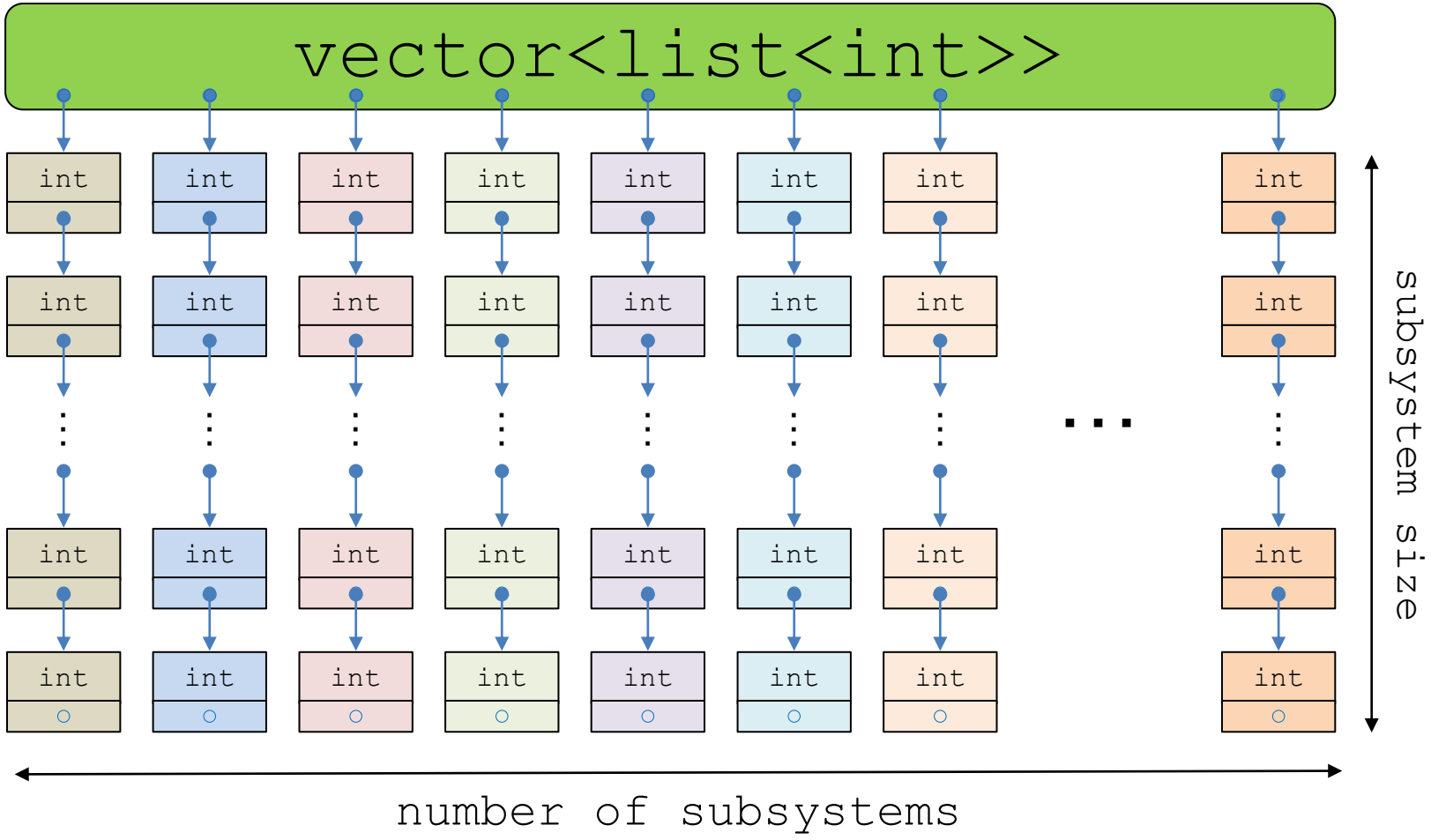
Benchmark II: LOCALITY

Shuffle Plan:

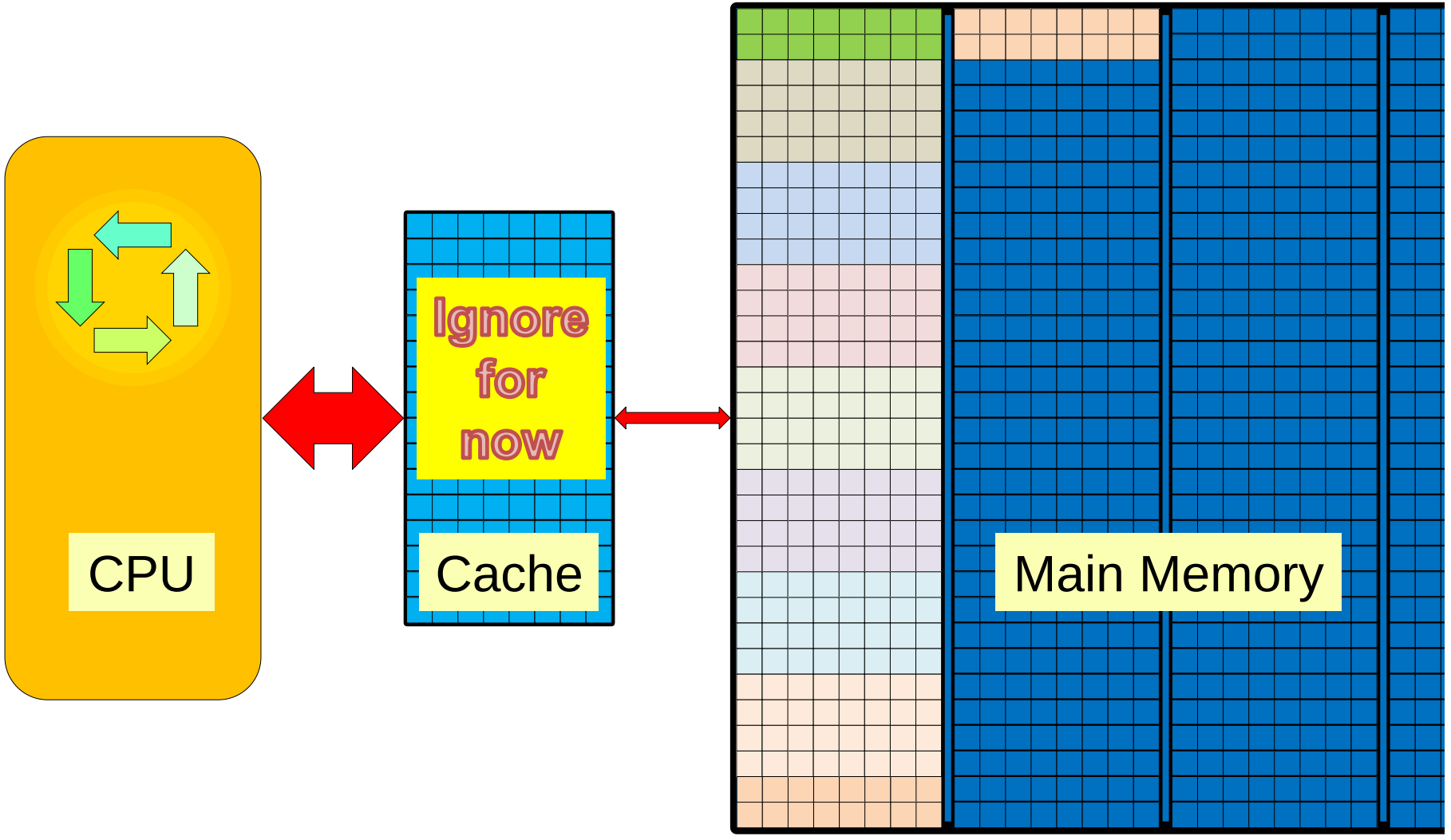
- Visit each subsystem in turn (just one element).
- For each subsystem
  - **Pop** the front element (after saving its `int` value).
  - **Push** that value onto the back of a random subsystem.
  - **Advance** to the next subsystem.
- Repeat until all elements are sufficiently shuffled.
- The result of this experiment is a shuffled system.
  - There is also the added run time do to the shuffling.

3. Analyzing the Benchmark Data

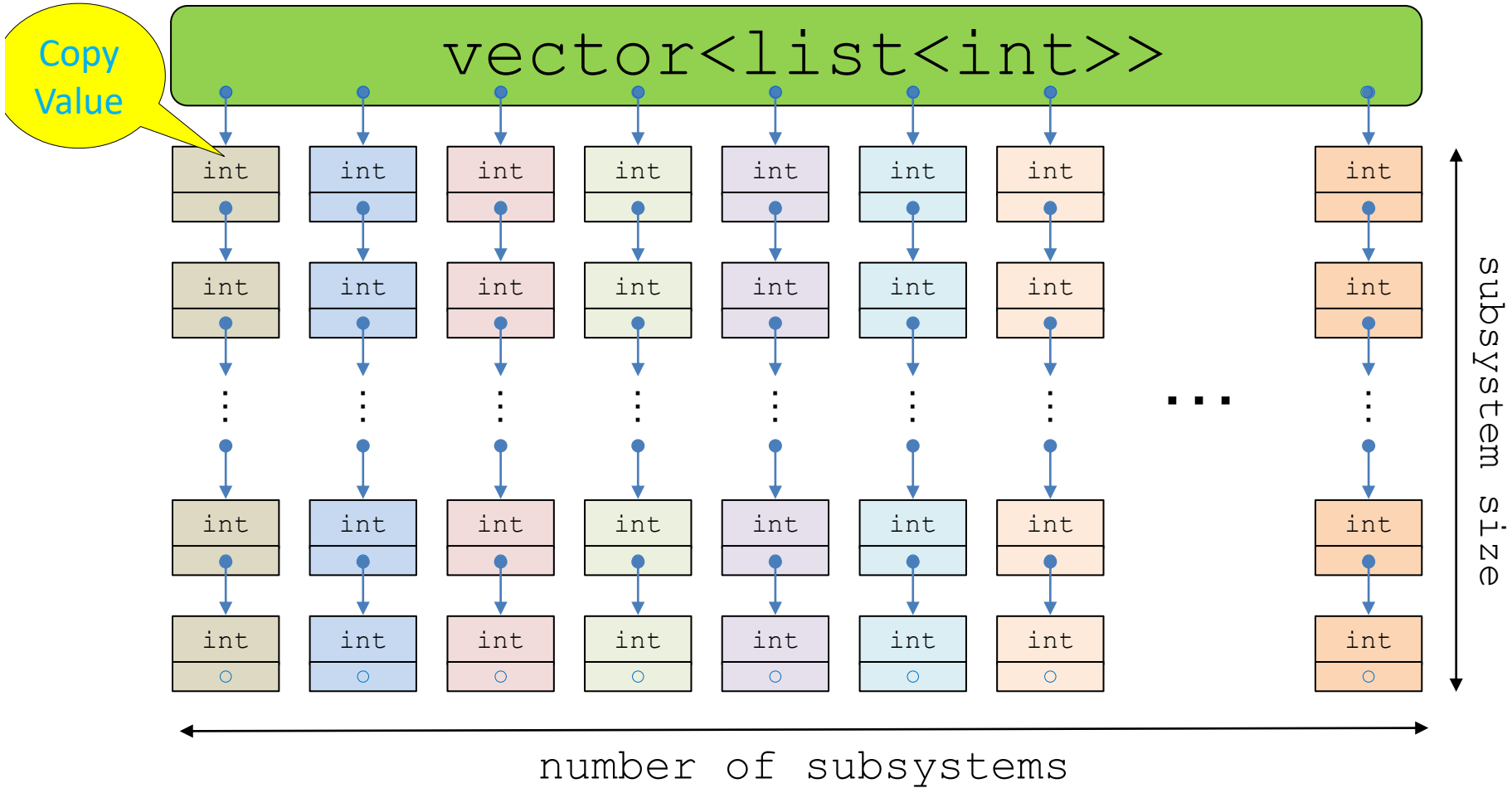
Benchmark II: LOCALITY

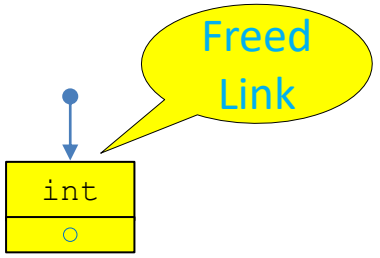


Benchmark II: LOCALITY



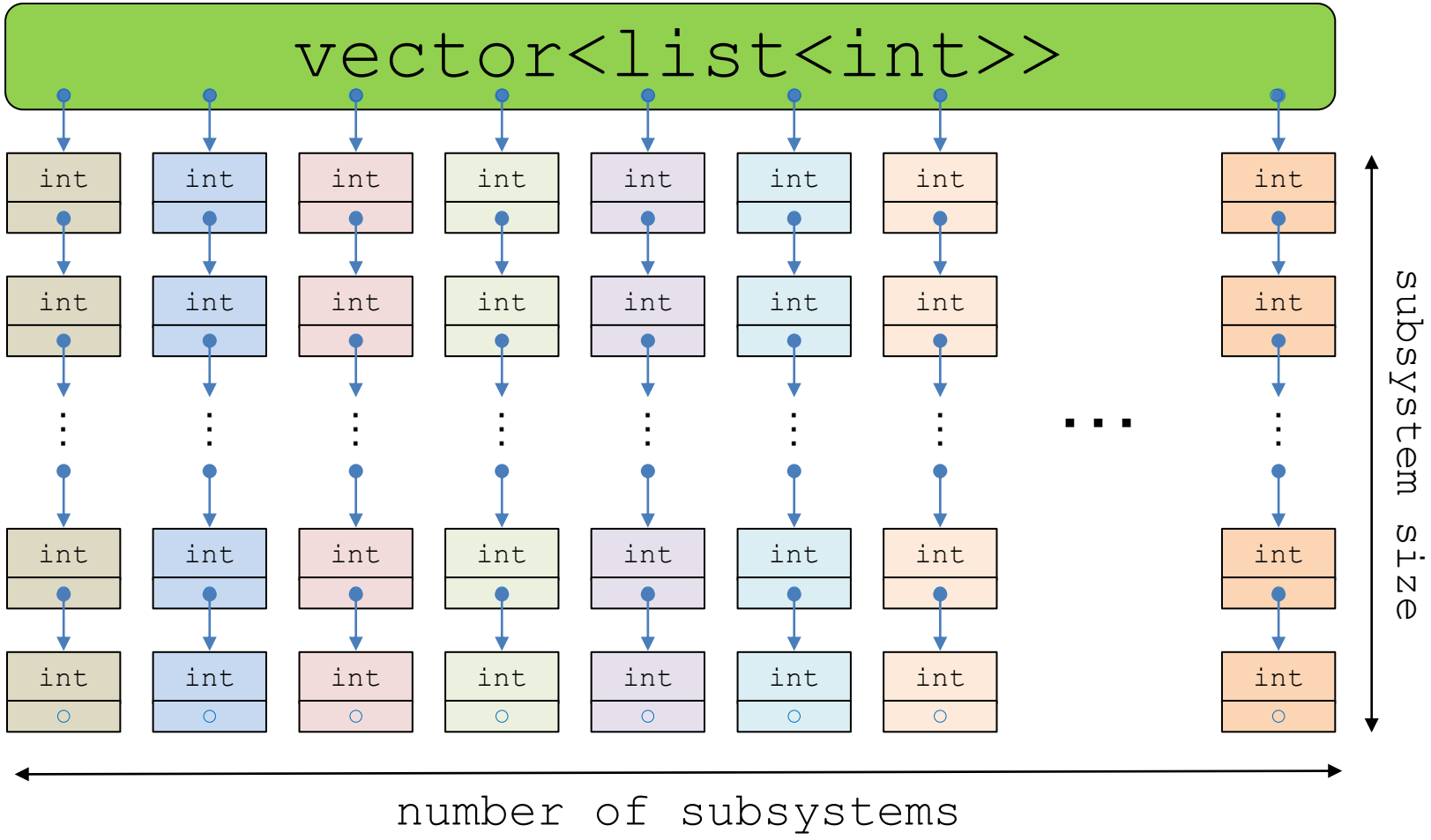
Benchmark II: LOCALITY



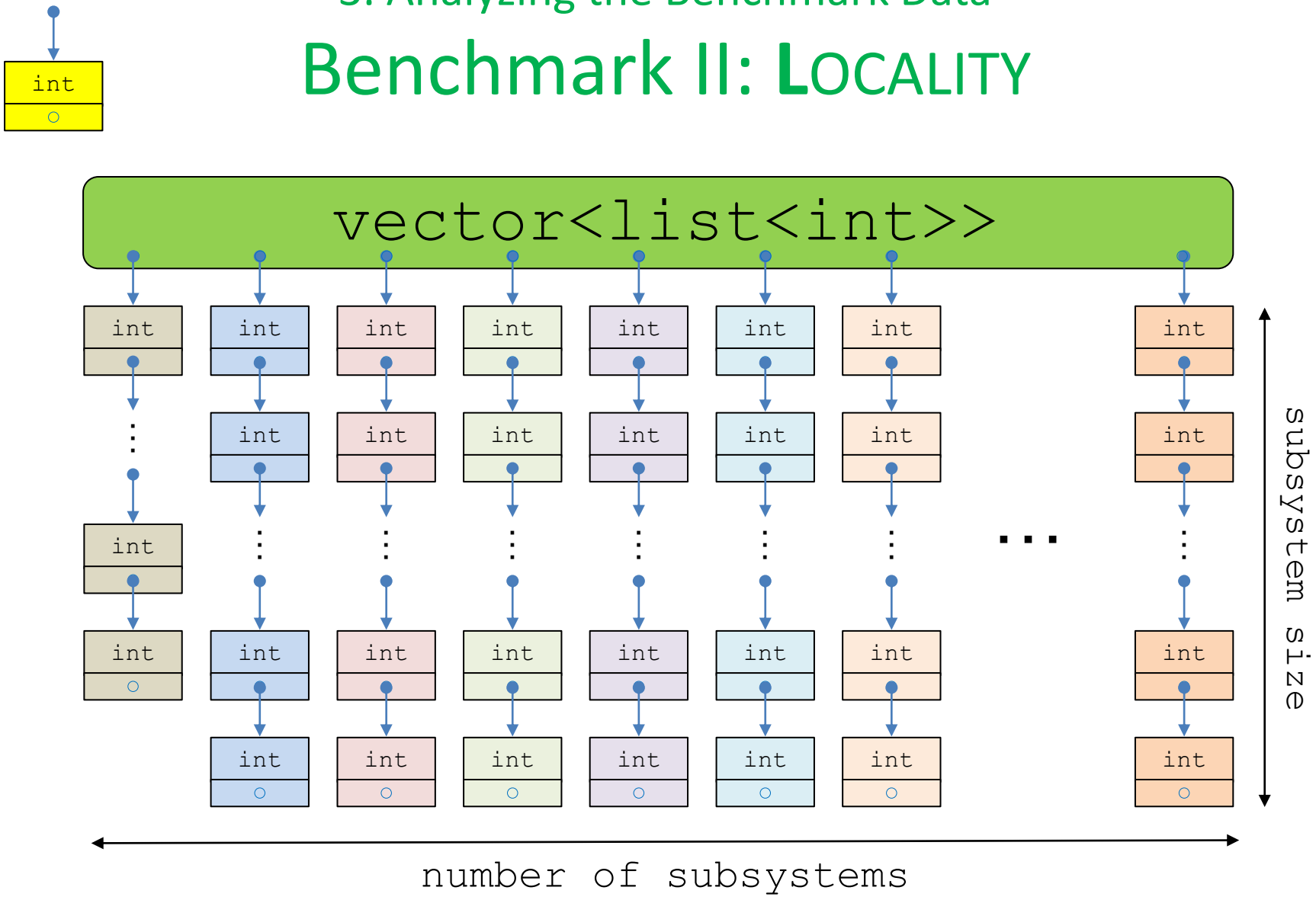


3. Analyzing the Benchmark Data

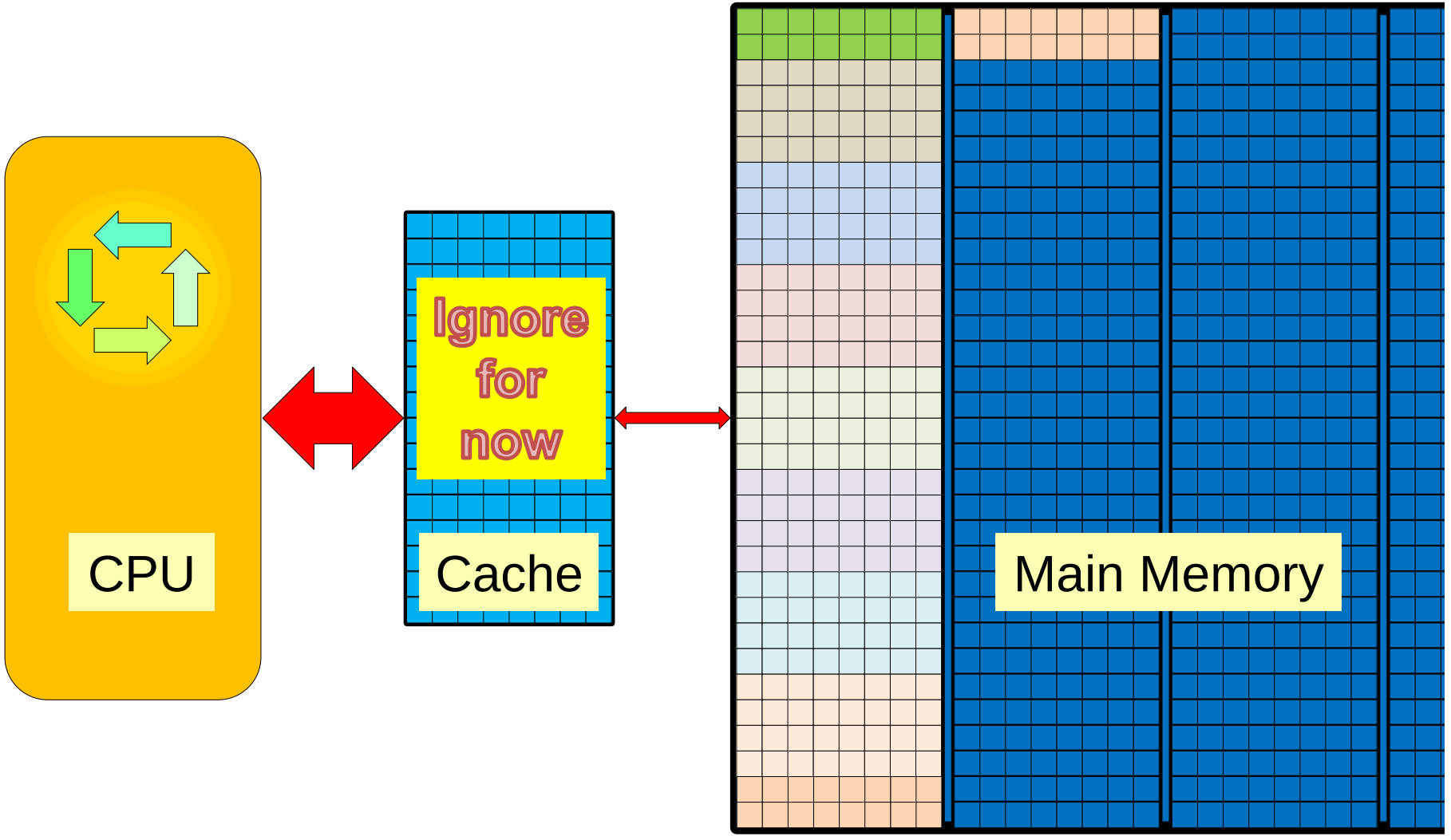
Benchmark II: LOCALITY



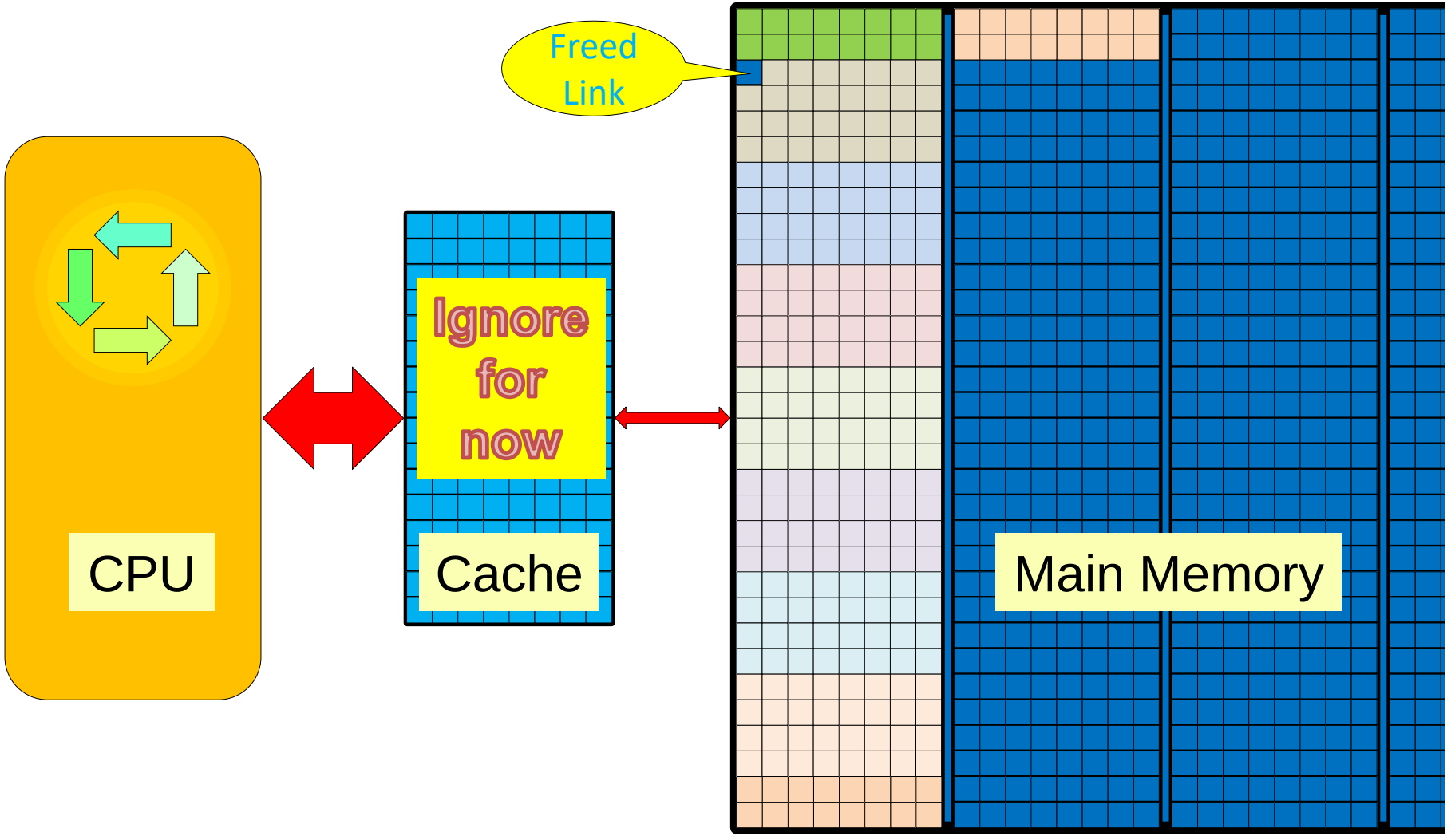
Benchmark II: LOCALITY



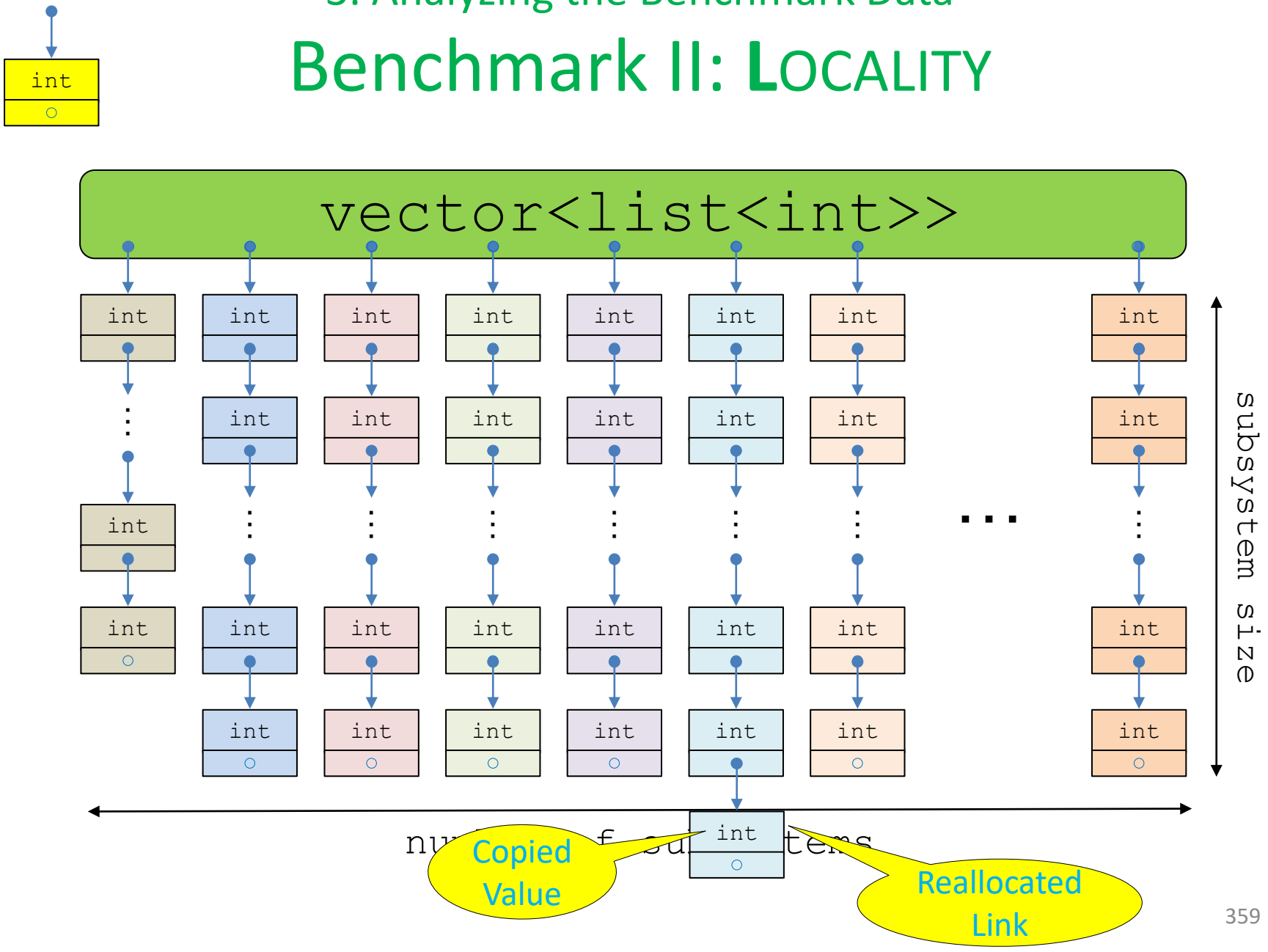
Benchmark II: LOCALITY



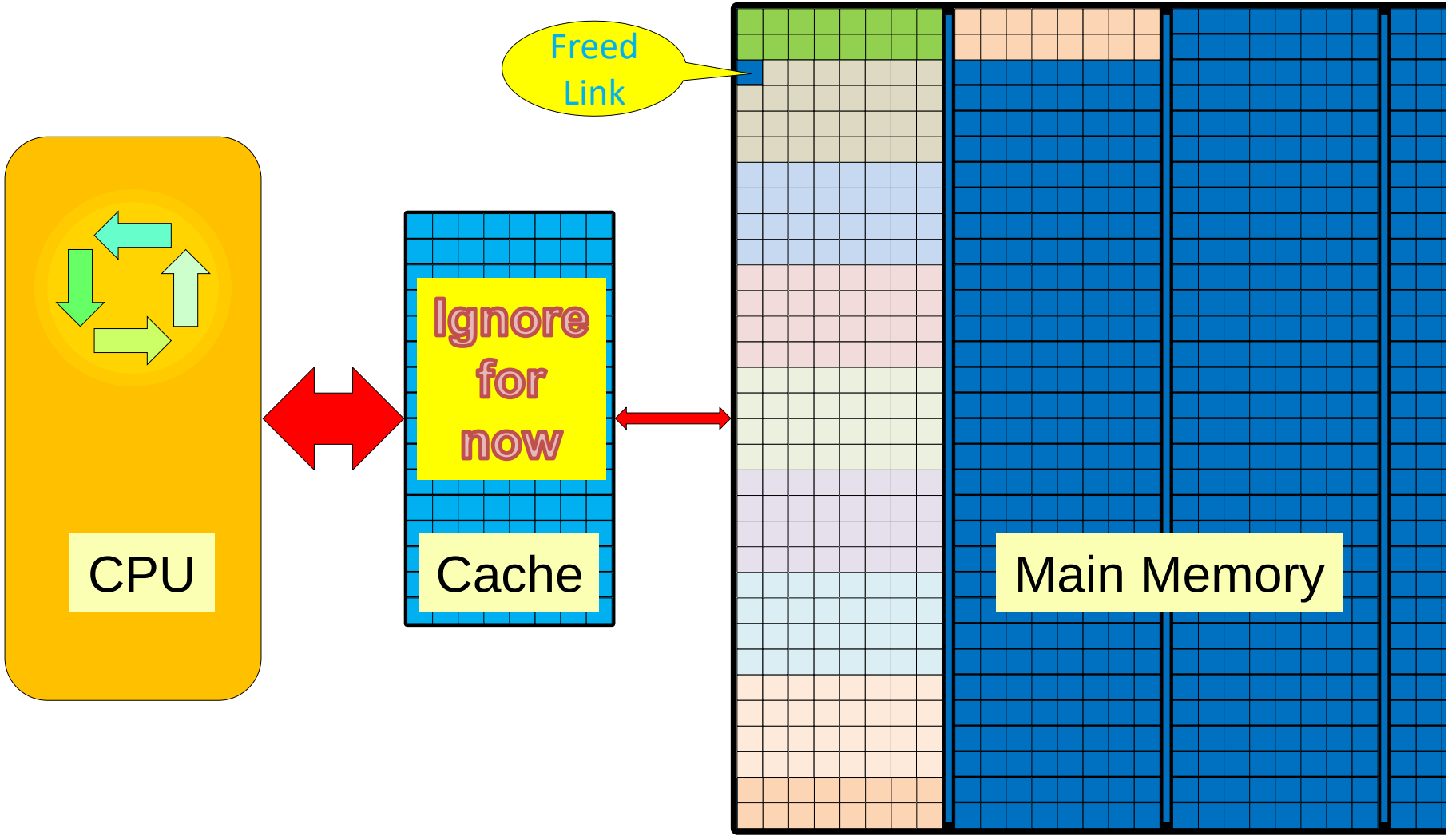
Benchmark II: LOCALITY



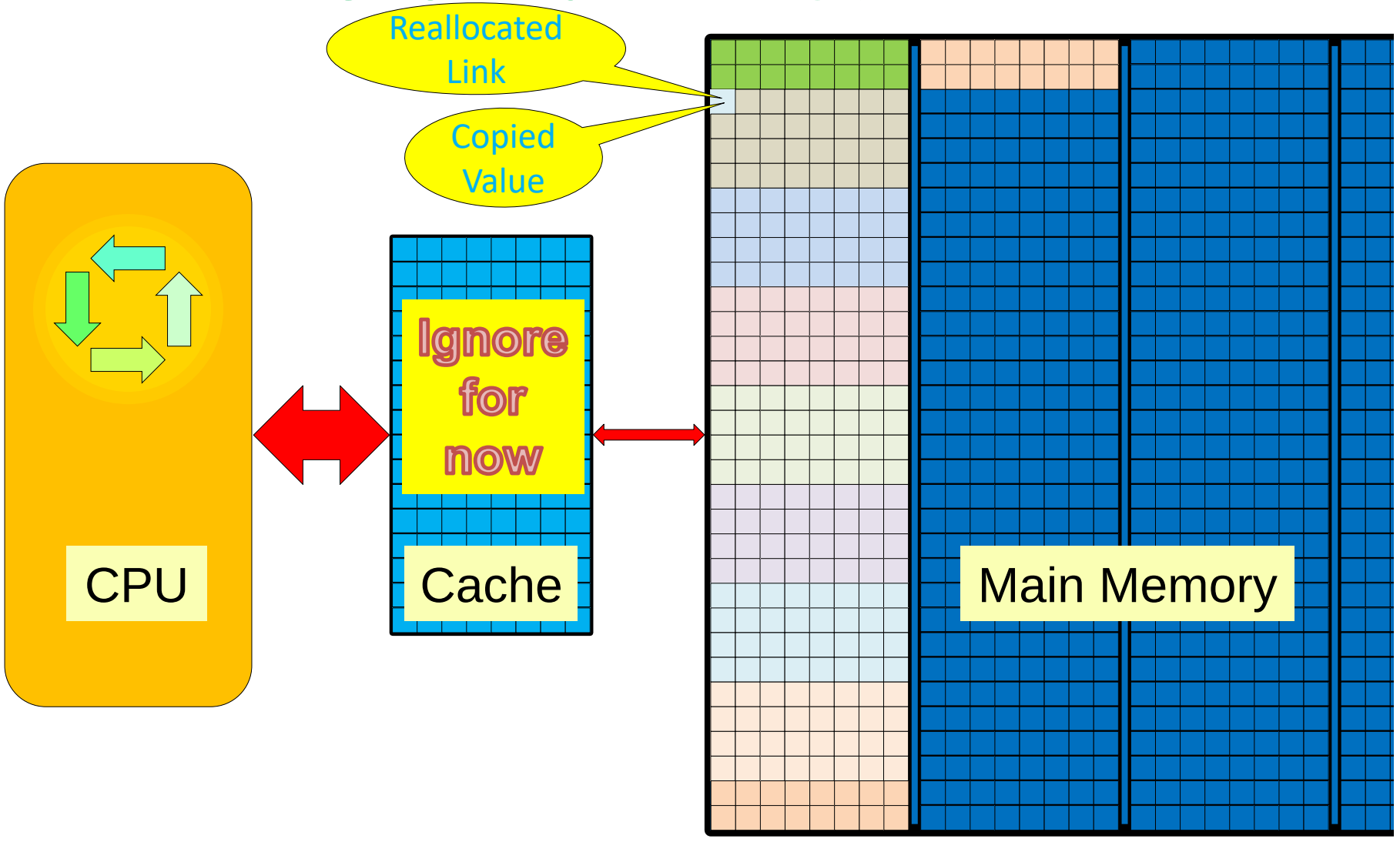
Benchmark II: LOCALITY

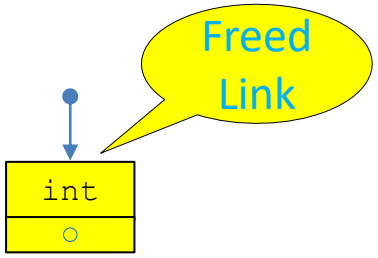


Benchmark II: LOCALITY



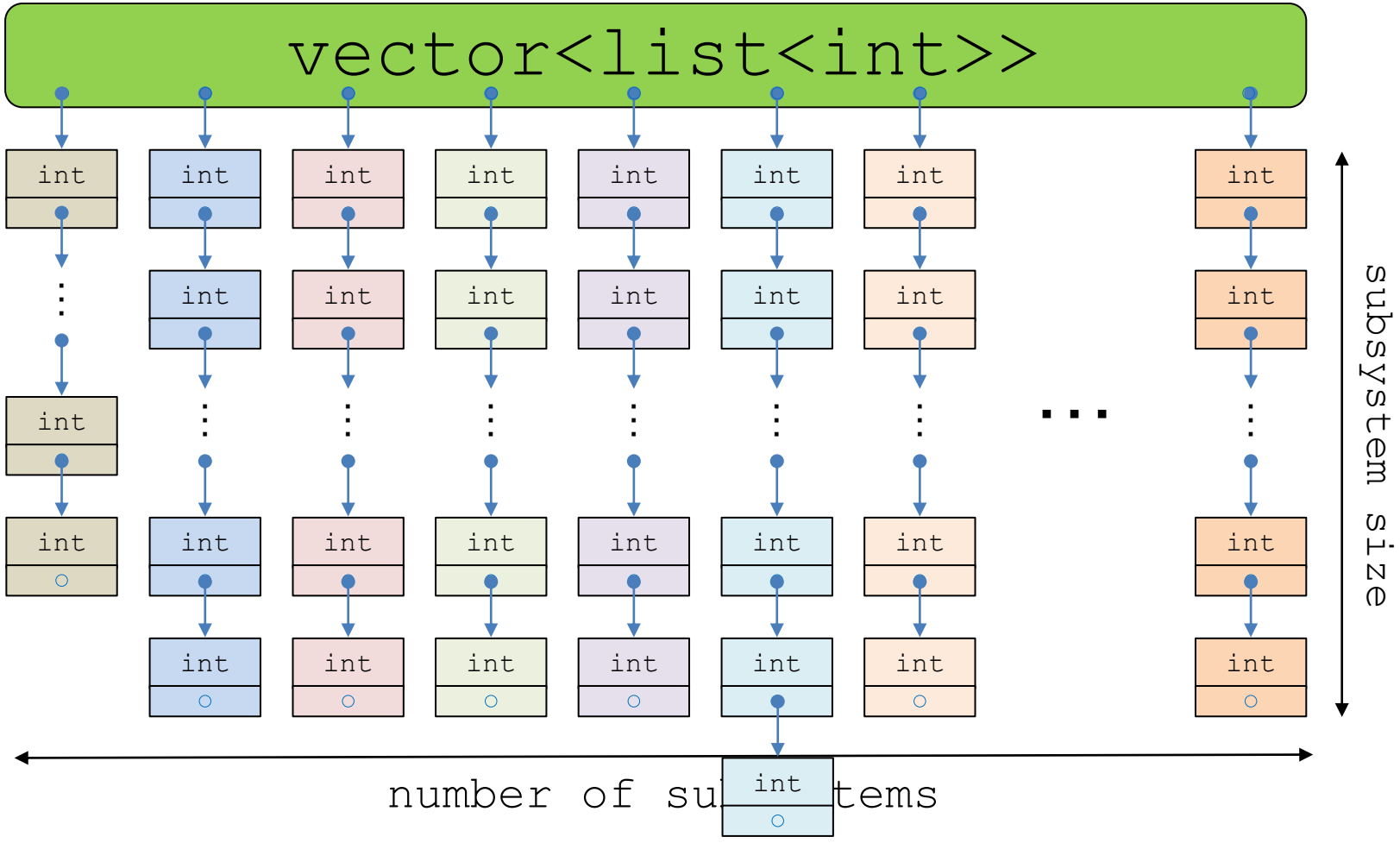
Benchmark II: LOCALITY



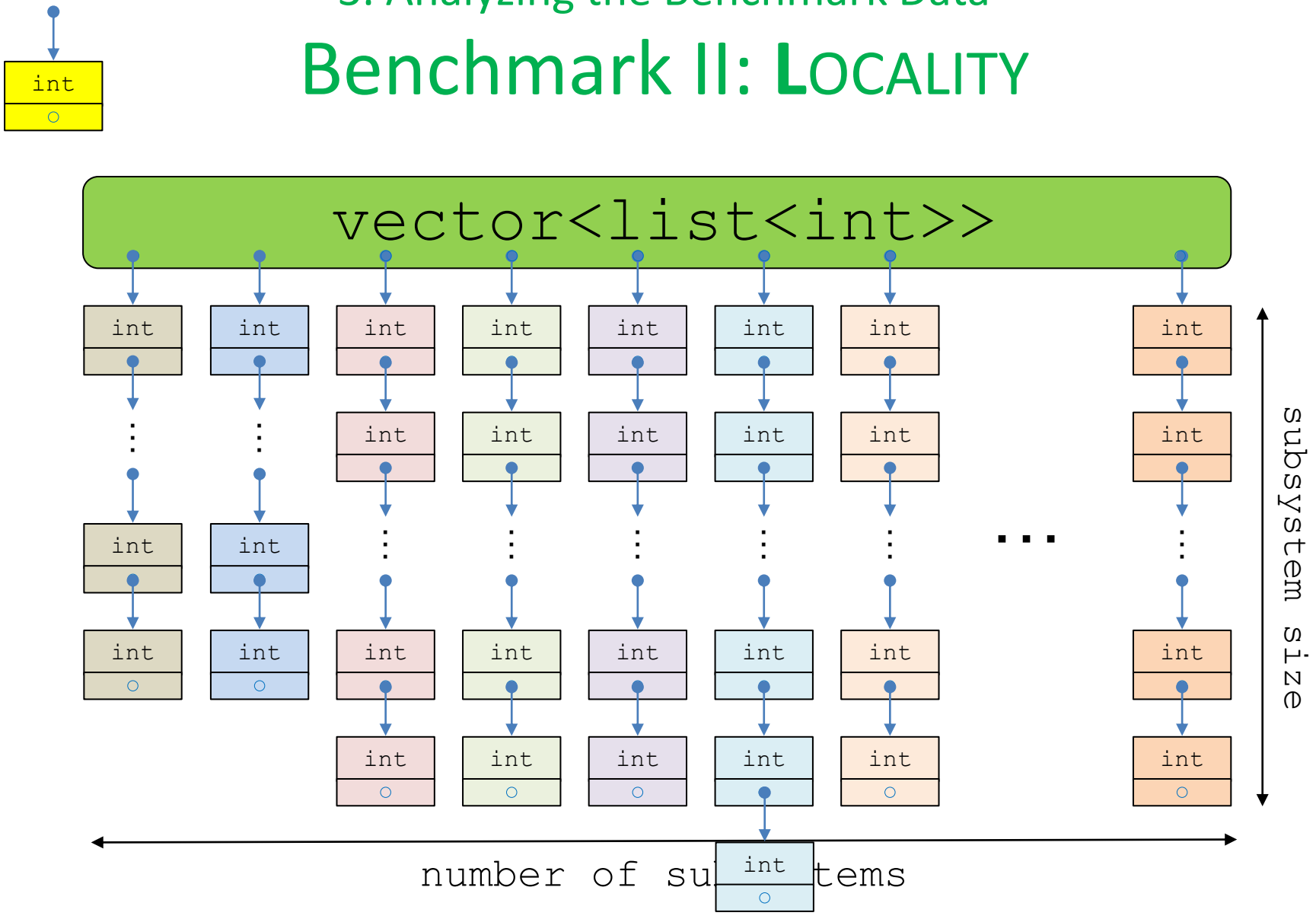


3. Analyzing the Benchmark Data

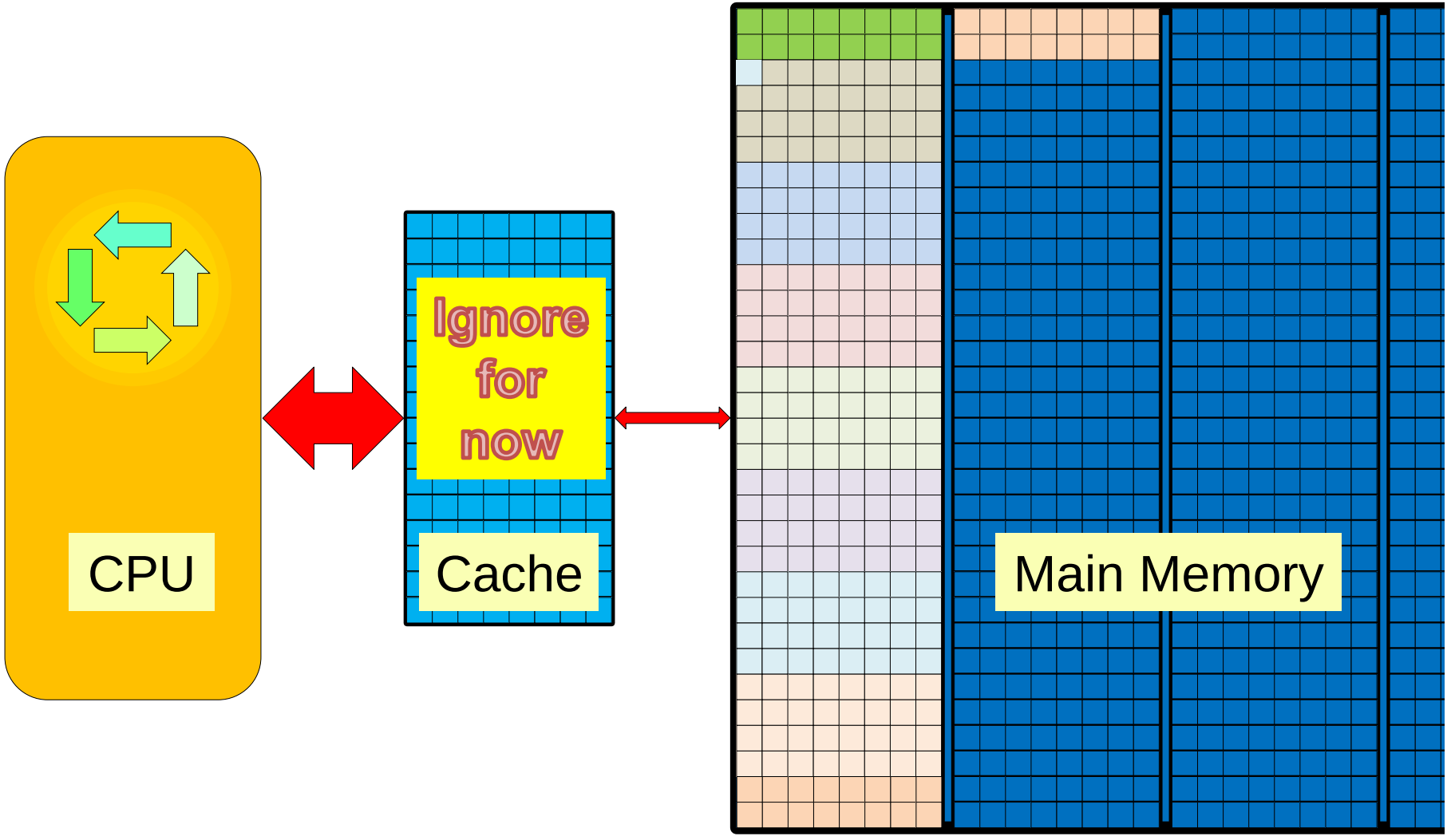
Benchmark II: LOCALITY



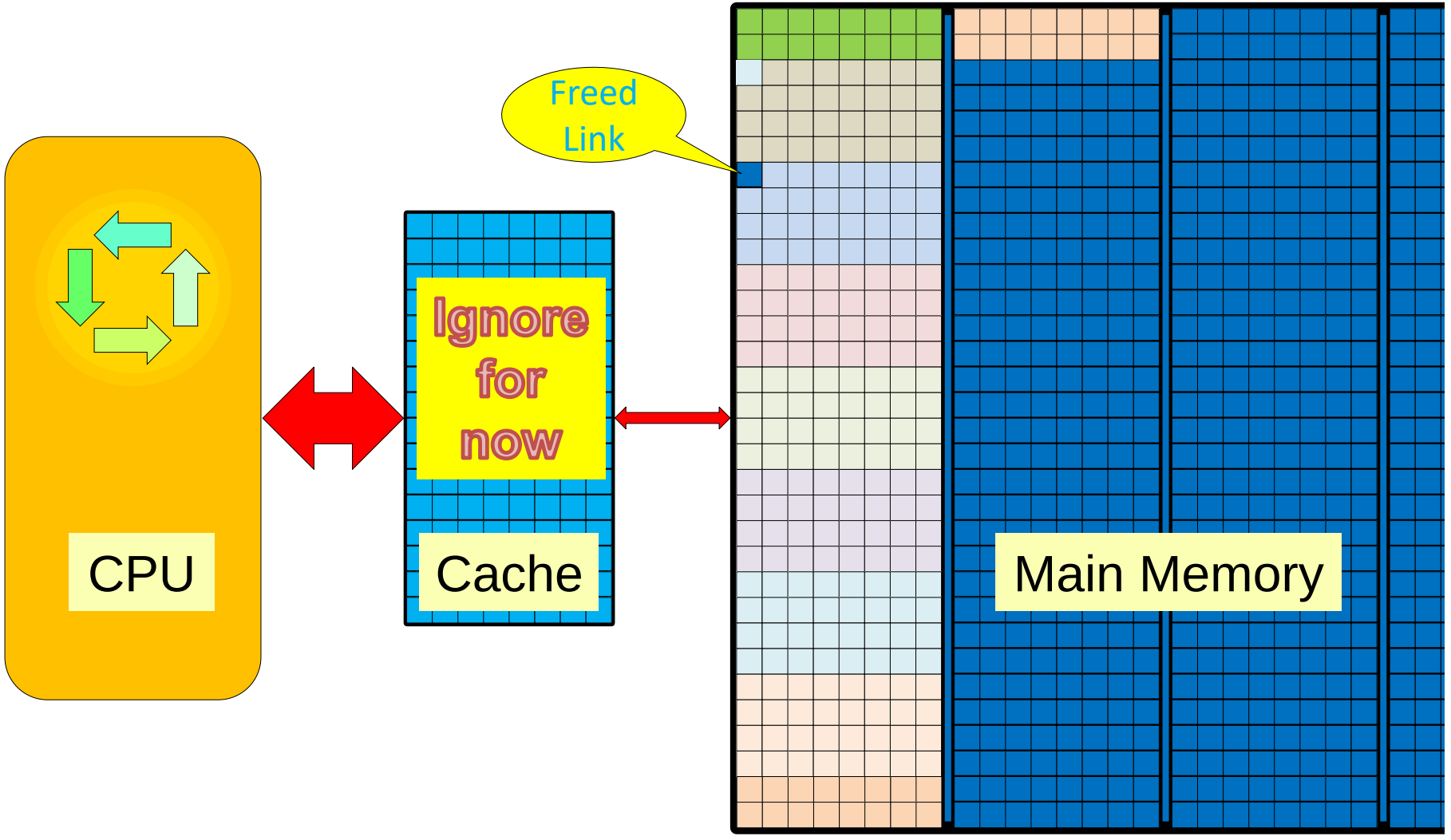
Benchmark II: LOCALITY



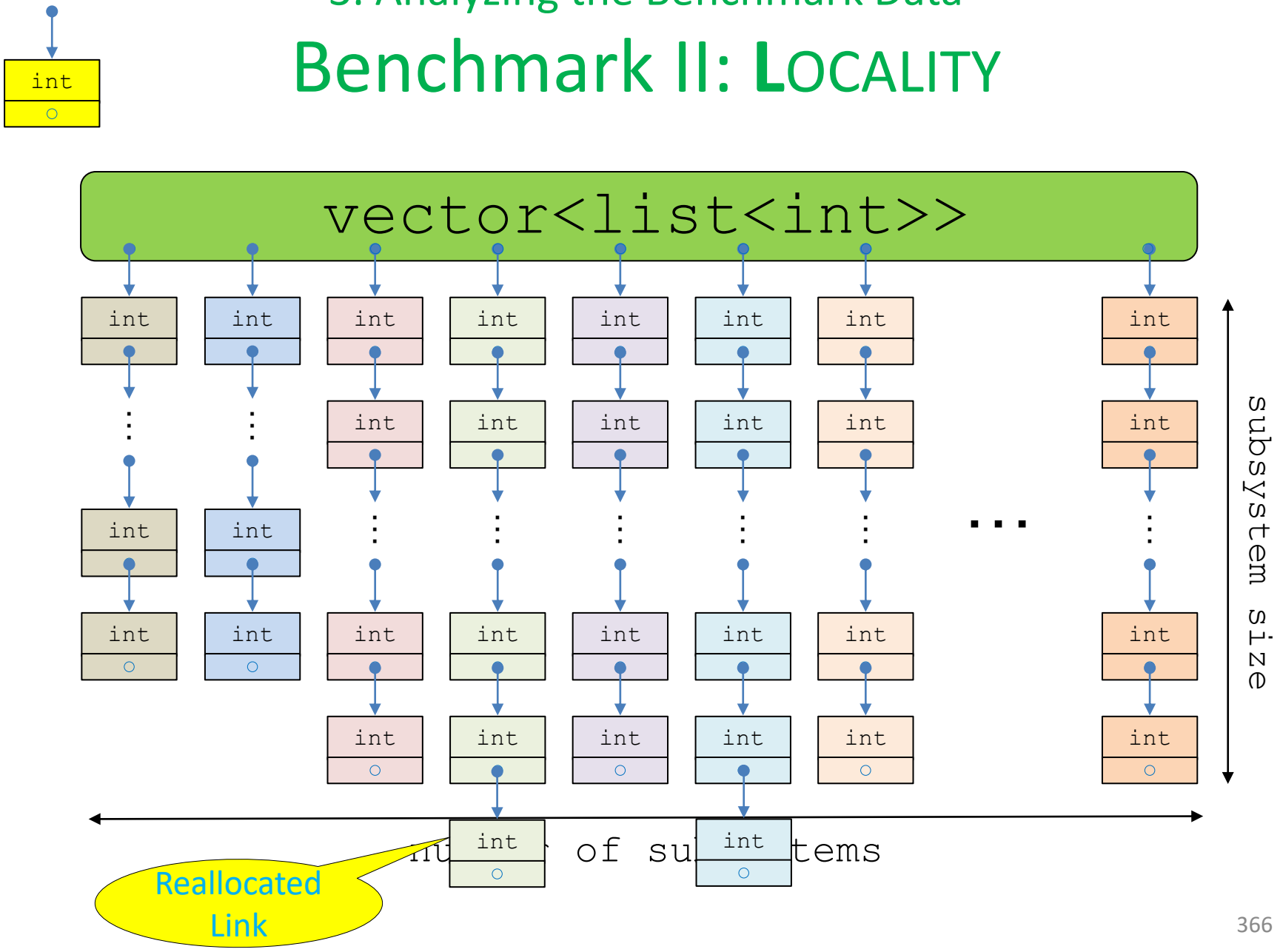
Benchmark II: LOCALITY



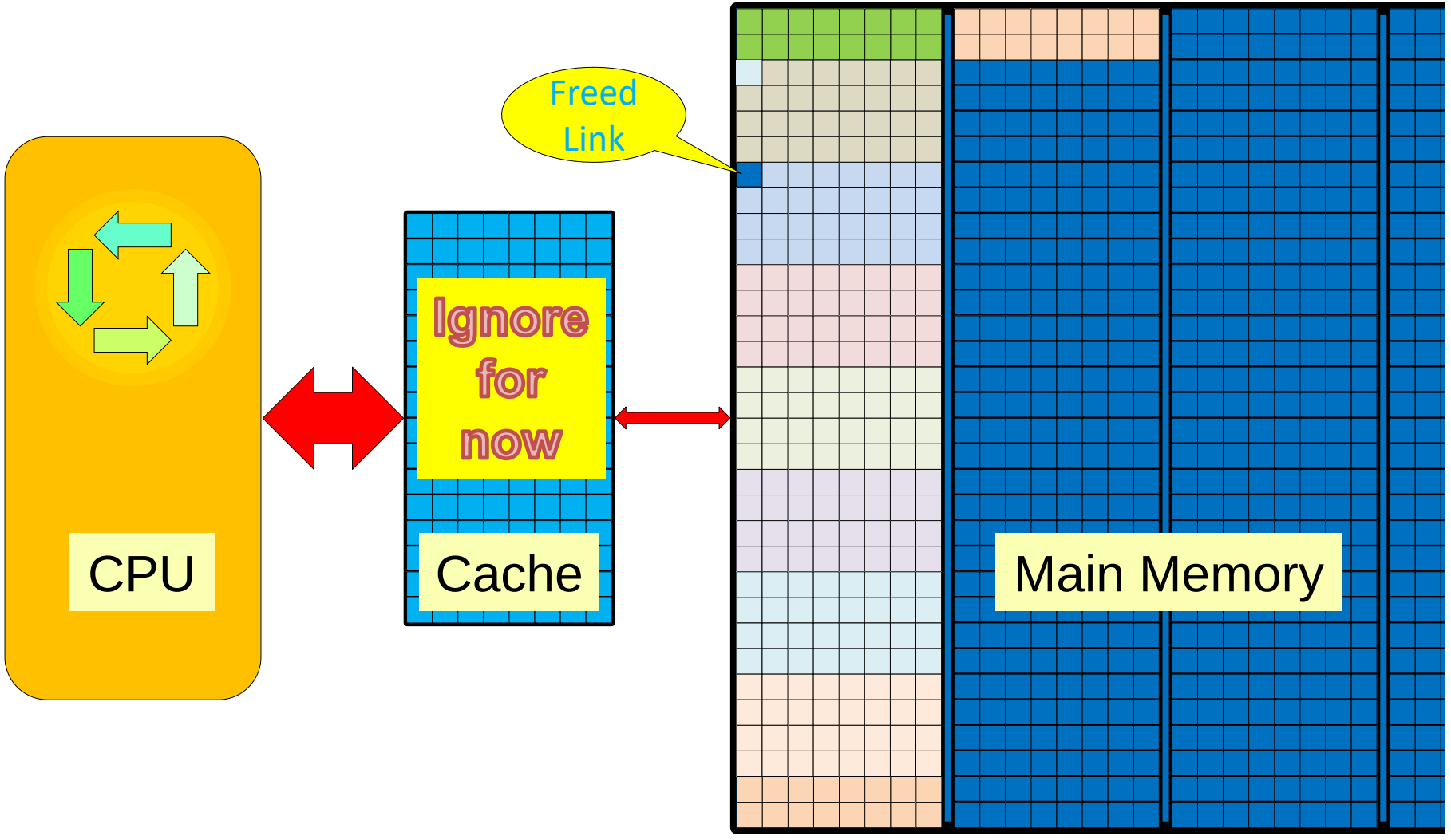
Benchmark II: LOCALITY



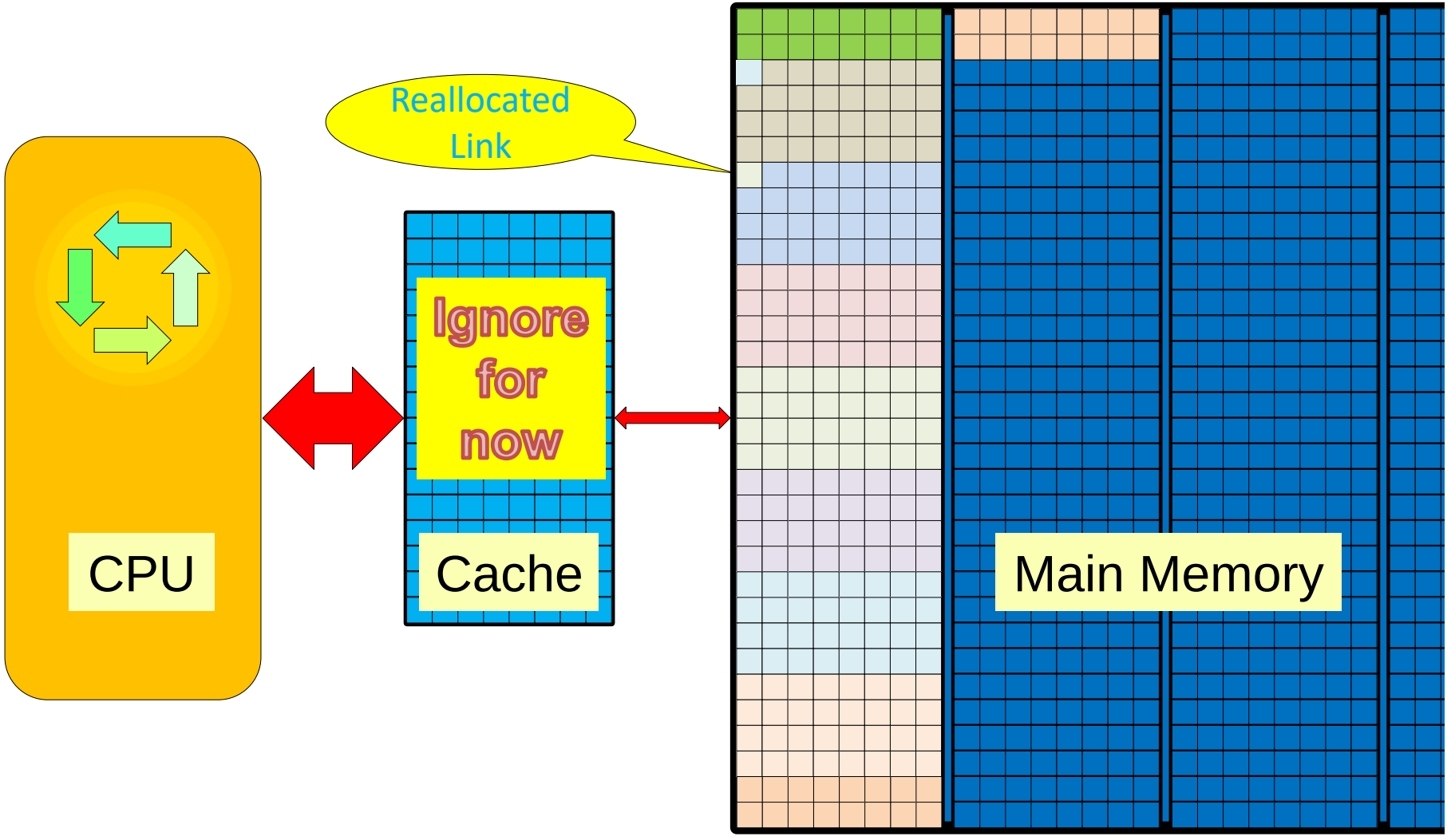
Benchmark II: LOCALITY



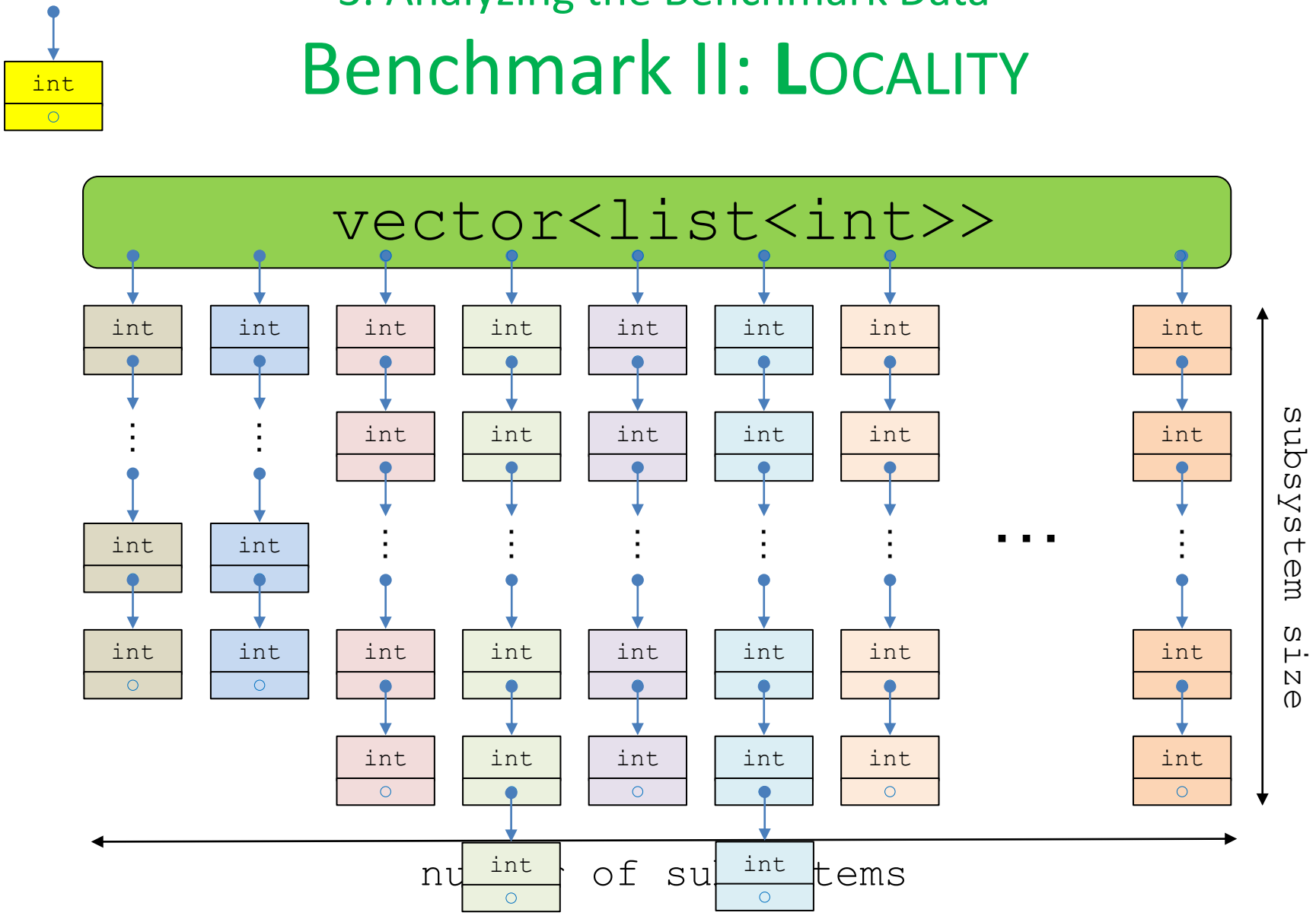
Benchmark II: LOCALITY



Benchmark II: LOCALITY

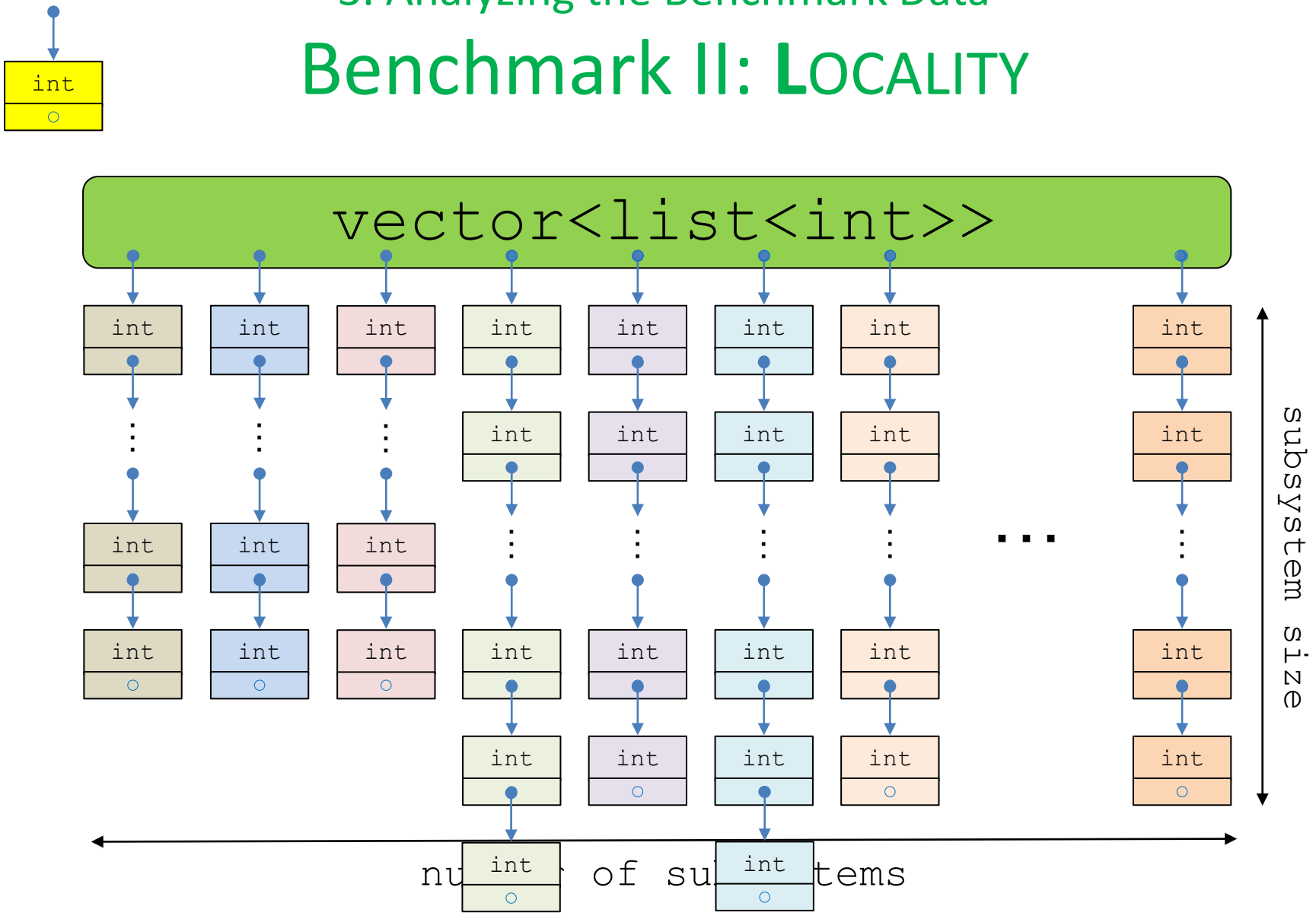


Benchmark II: LOCALITY

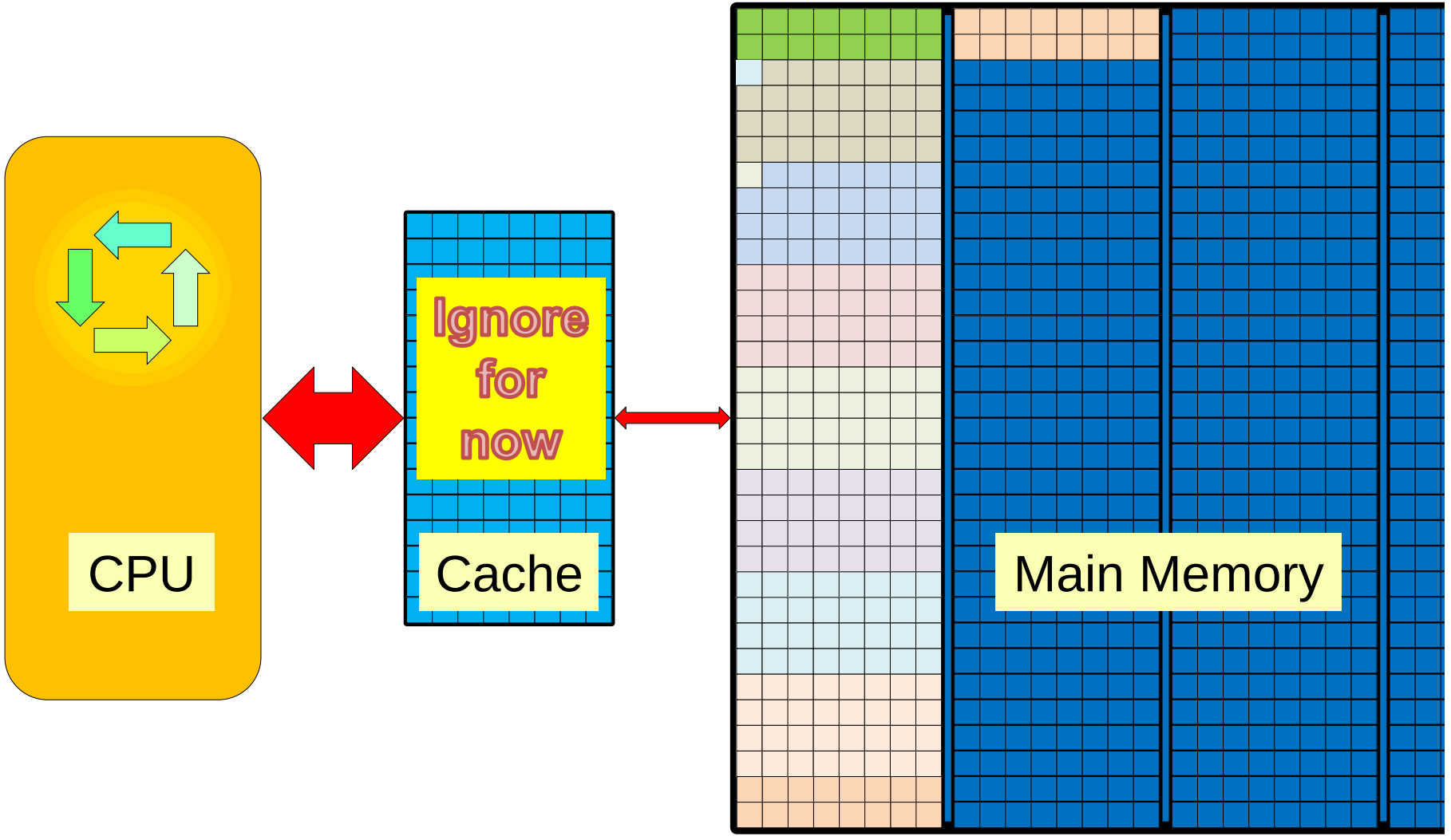


3. Analyzing the Benchmark Data

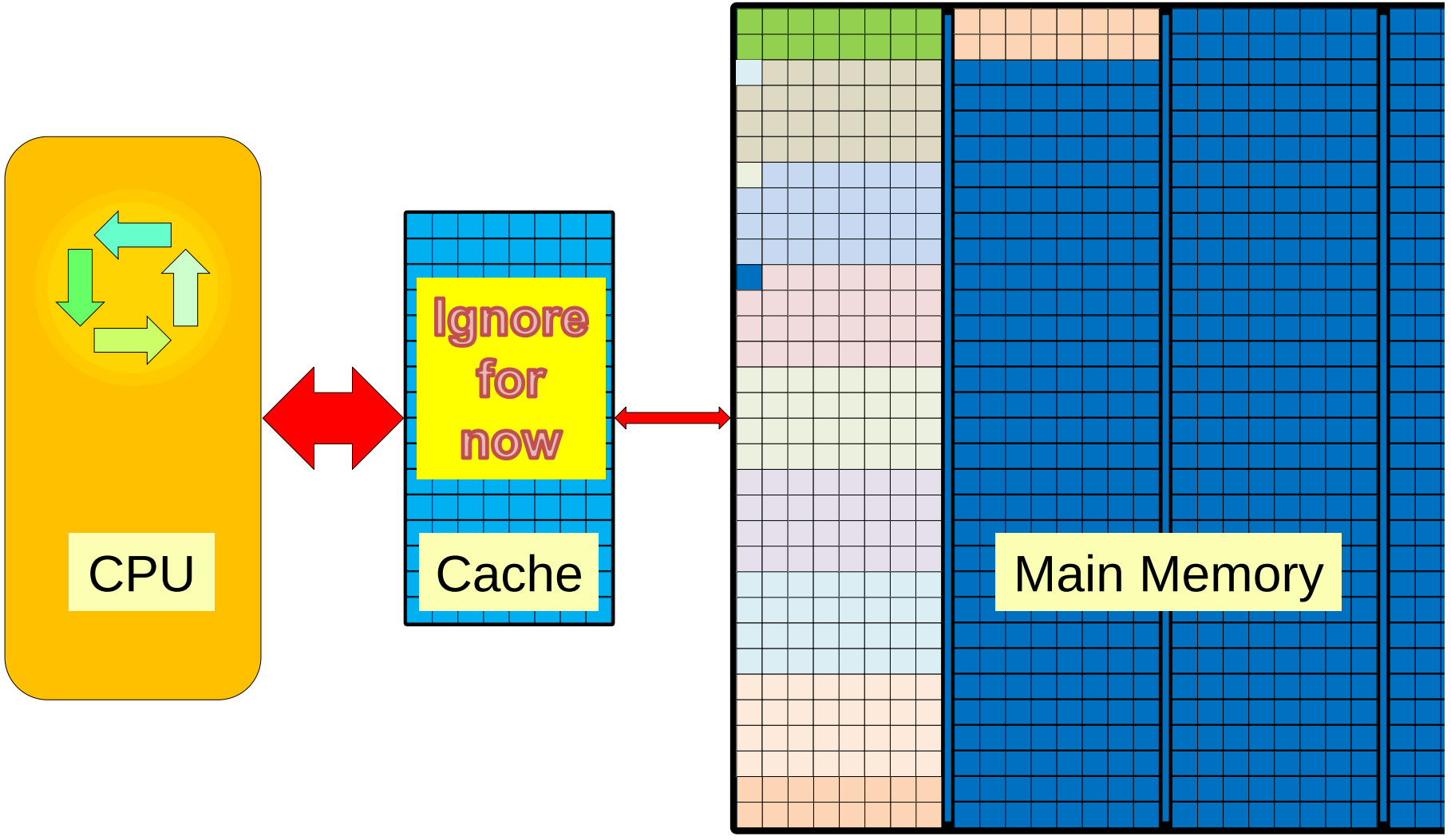
Benchmark II: LOCALITY



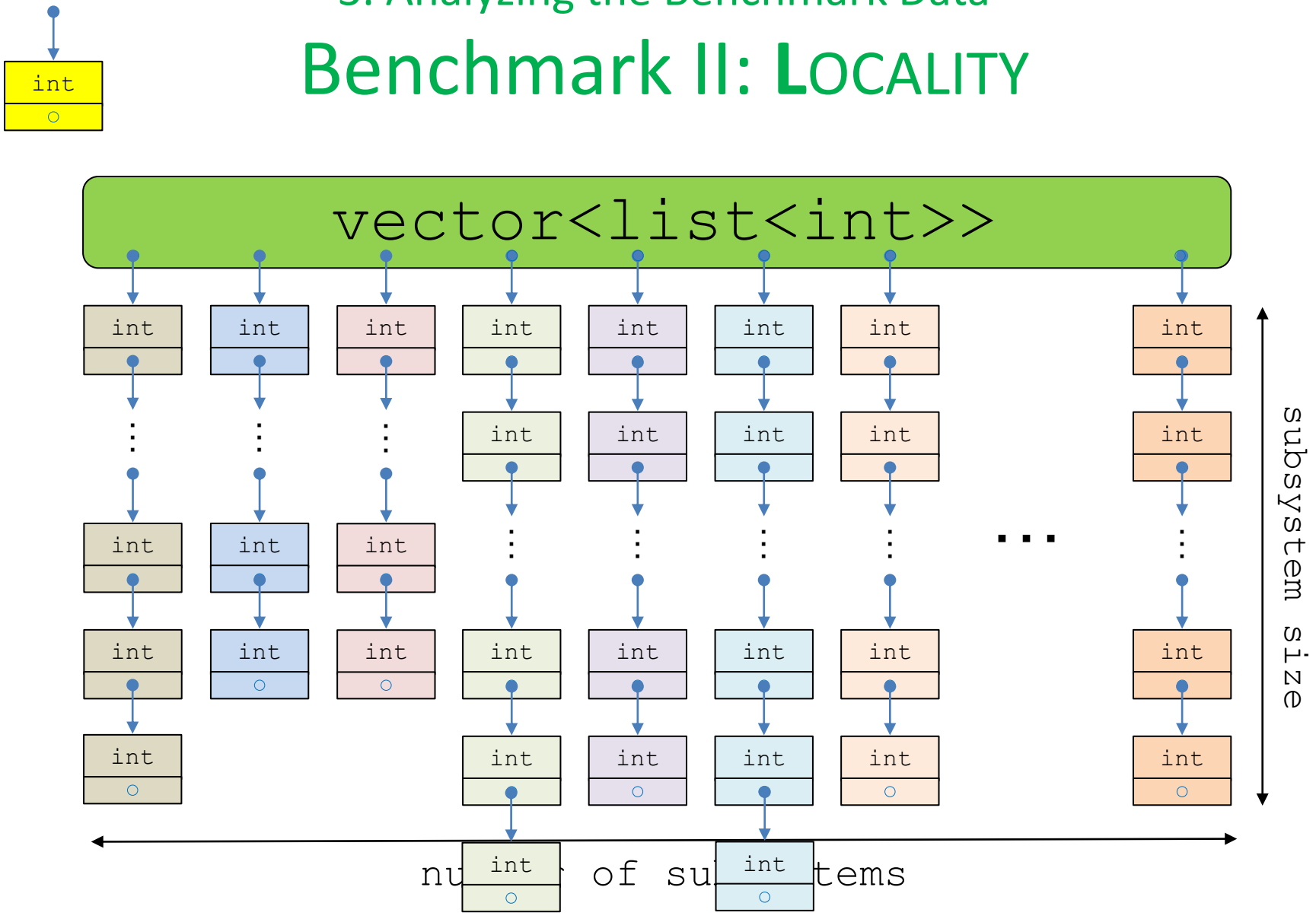
Benchmark II: LOCALITY



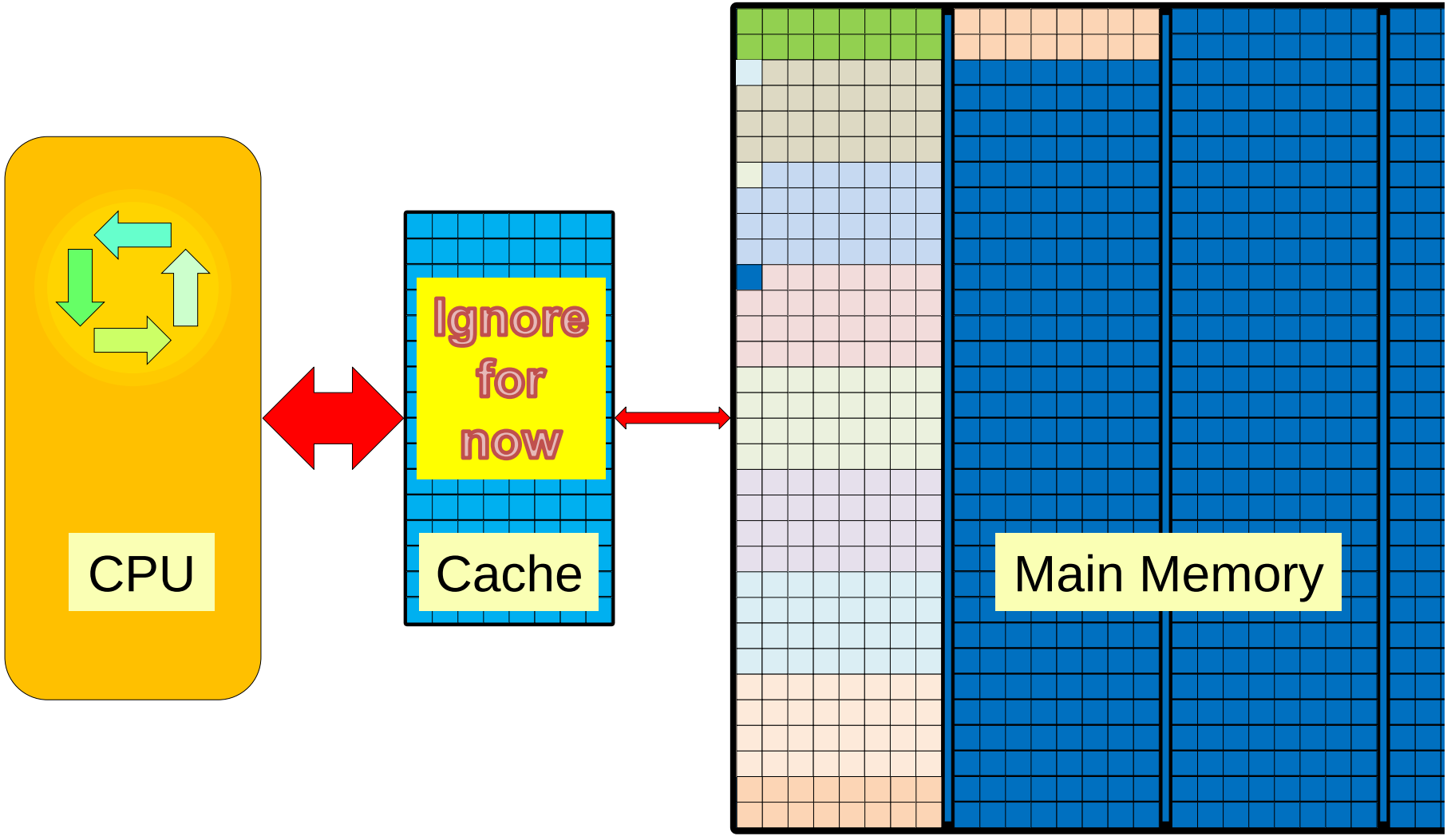
Benchmark II: LOCALITY



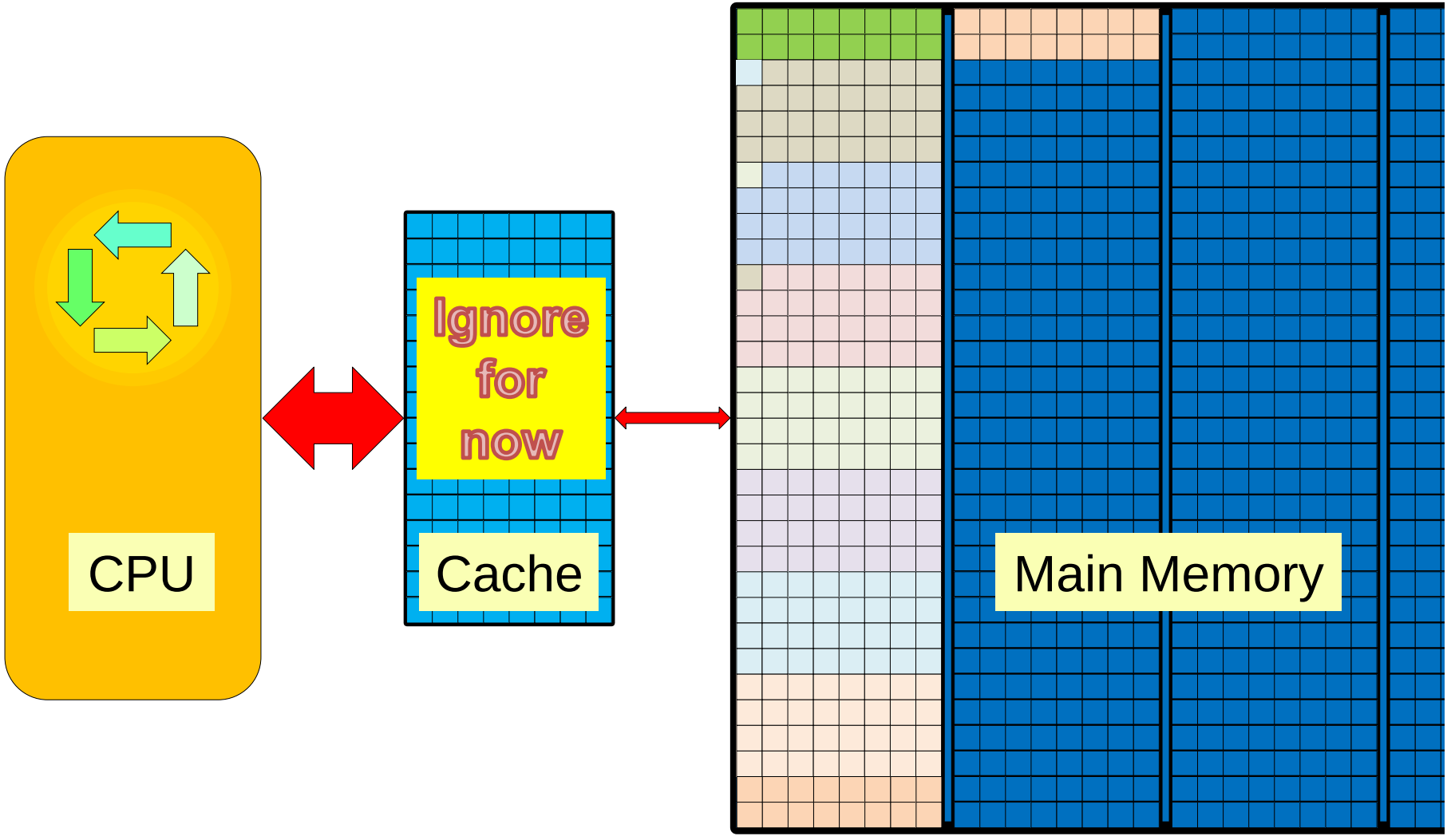
Benchmark II: LOCALITY



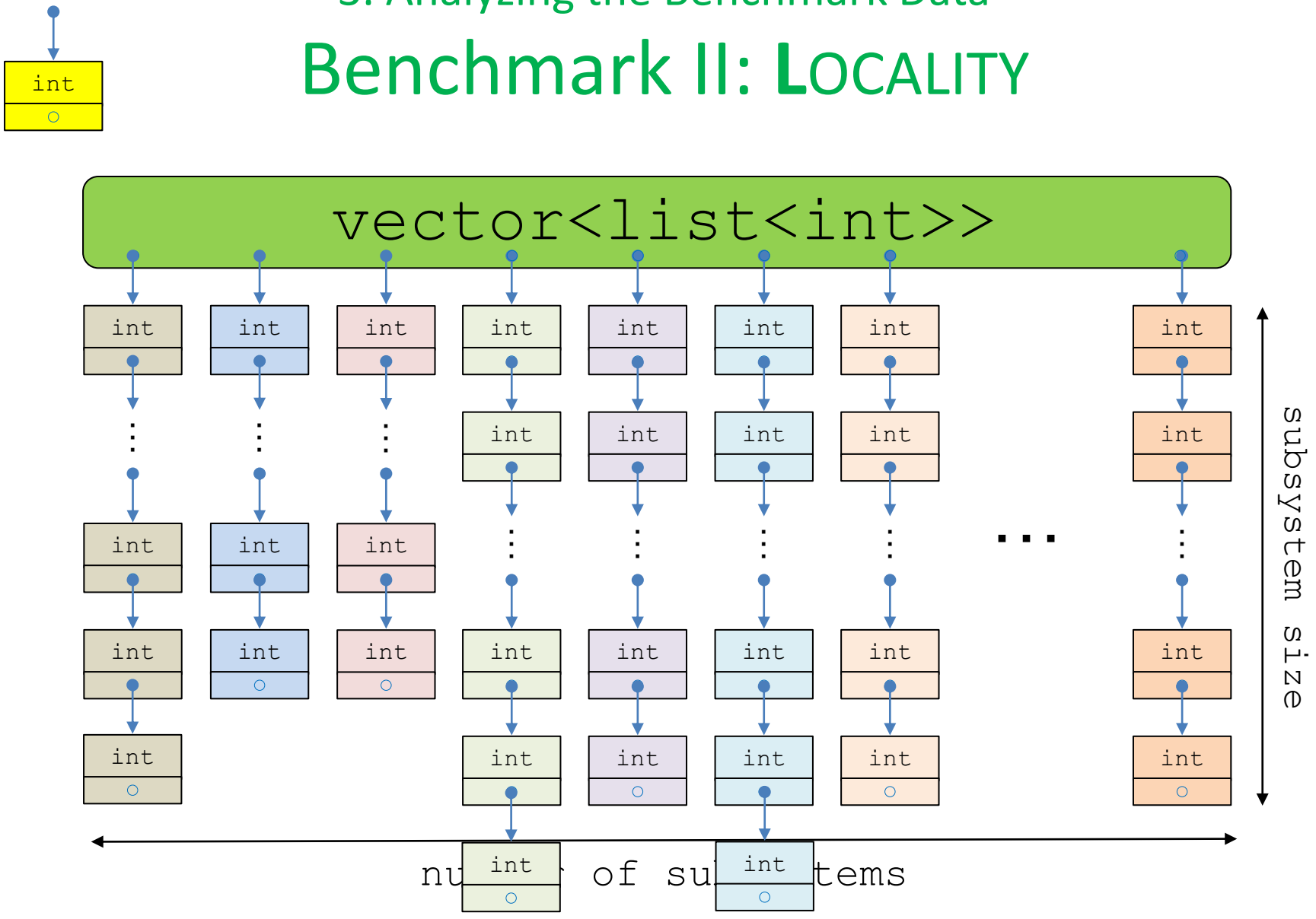
Benchmark II: LOCALITY



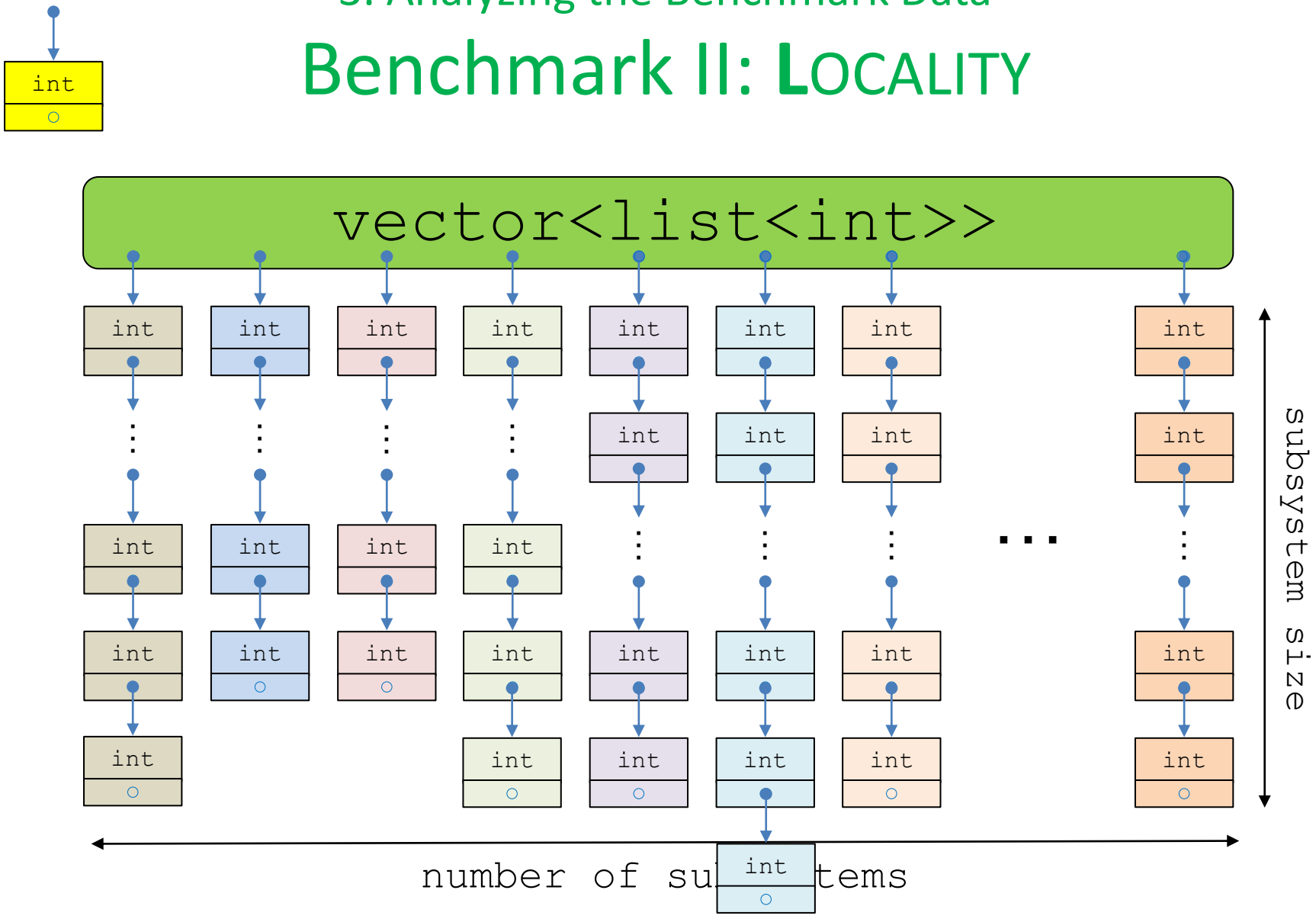
Benchmark II: LOCALITY



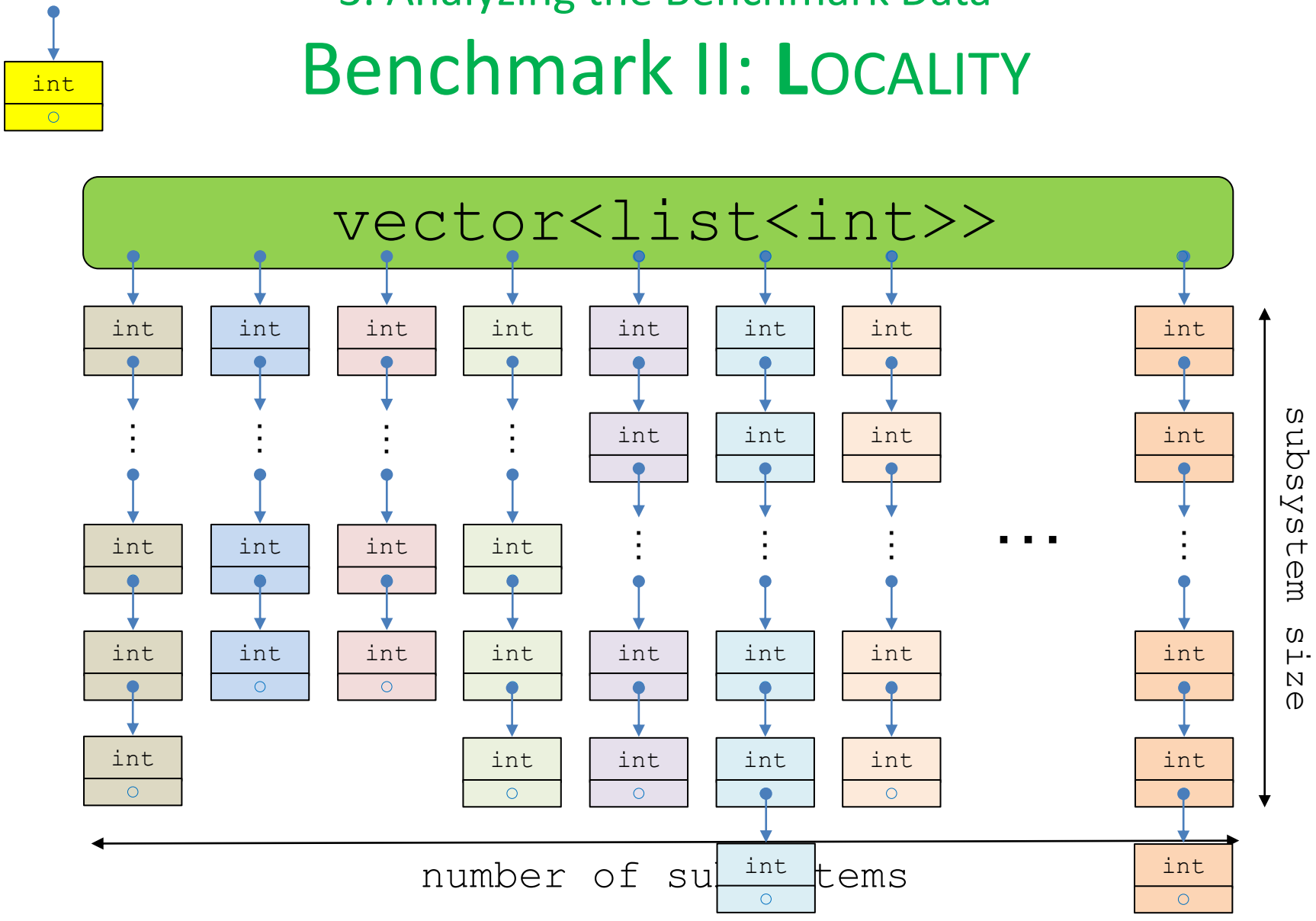
Benchmark II: LOCALITY



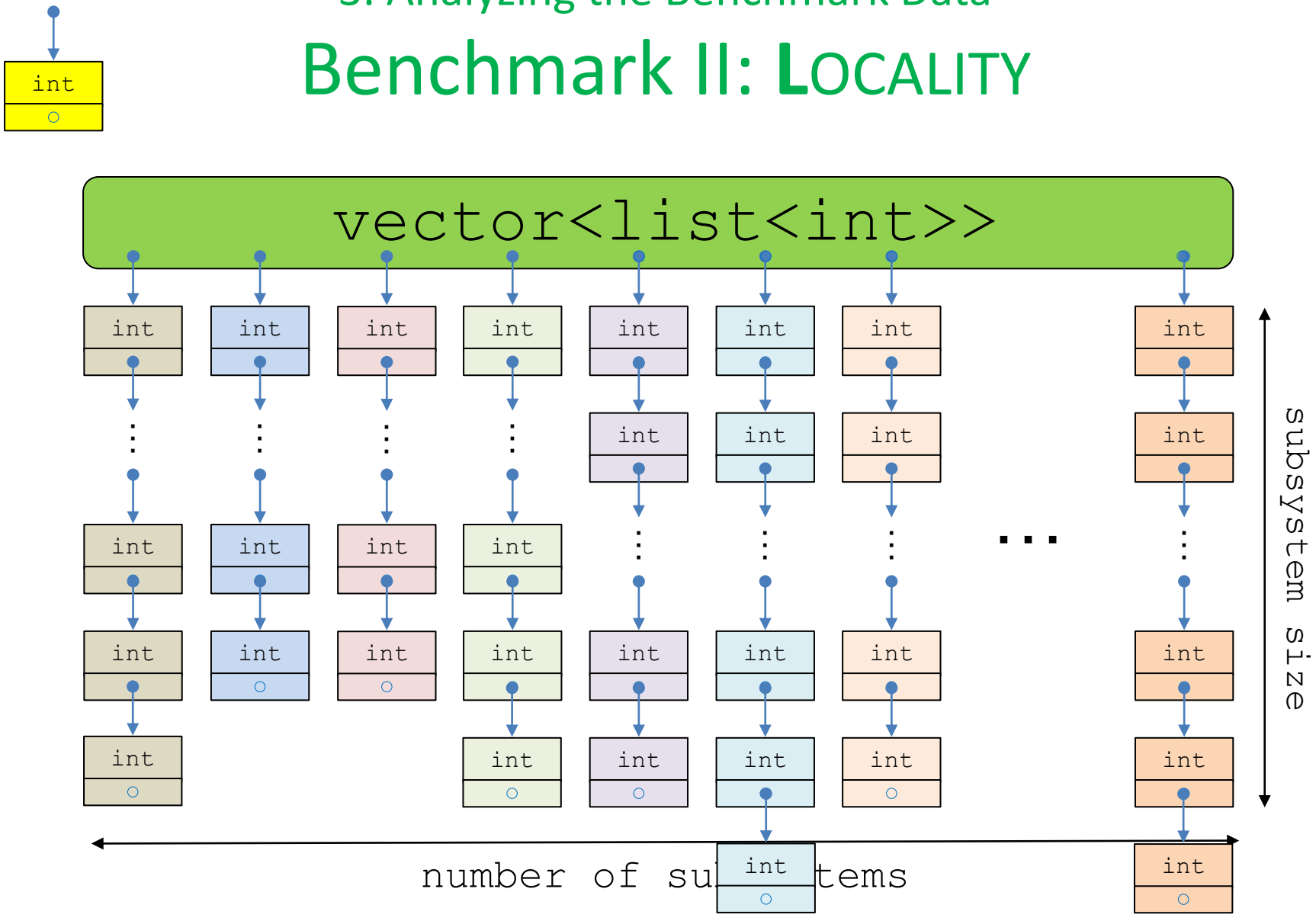
Benchmark II: LOCALITY



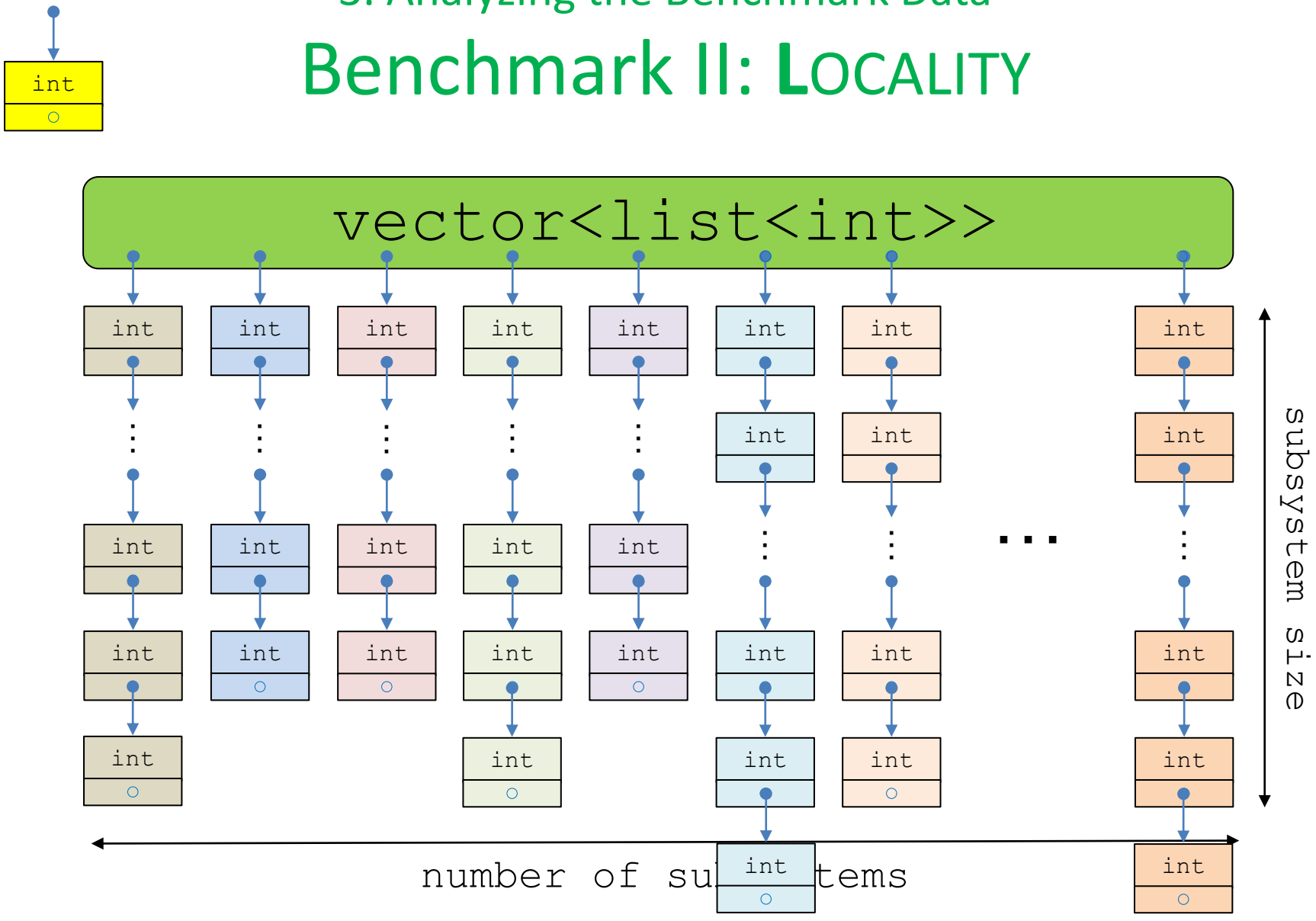
Benchmark II: LOCALITY



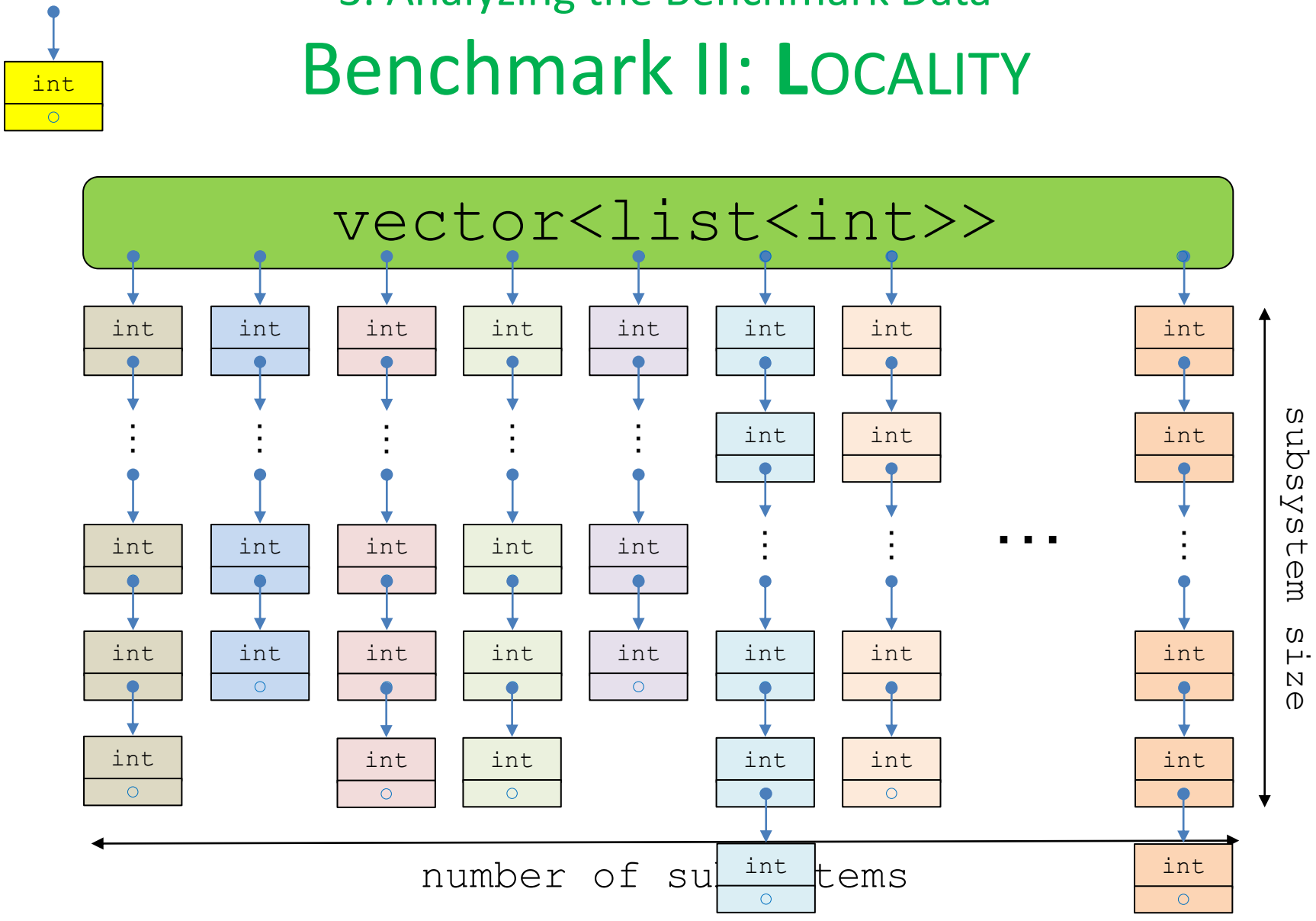
Benchmark II: LOCALITY



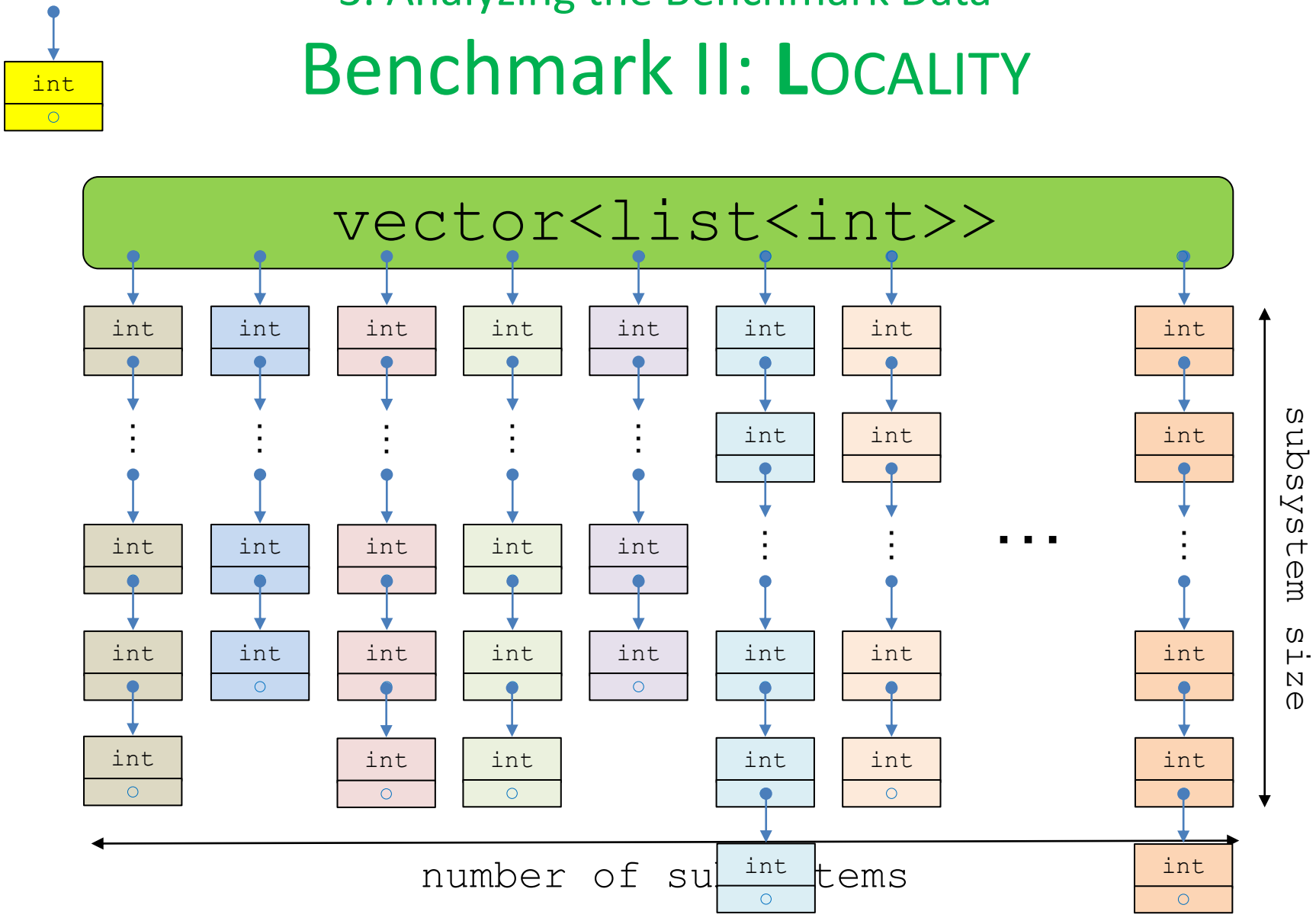
Benchmark II: LOCALITY



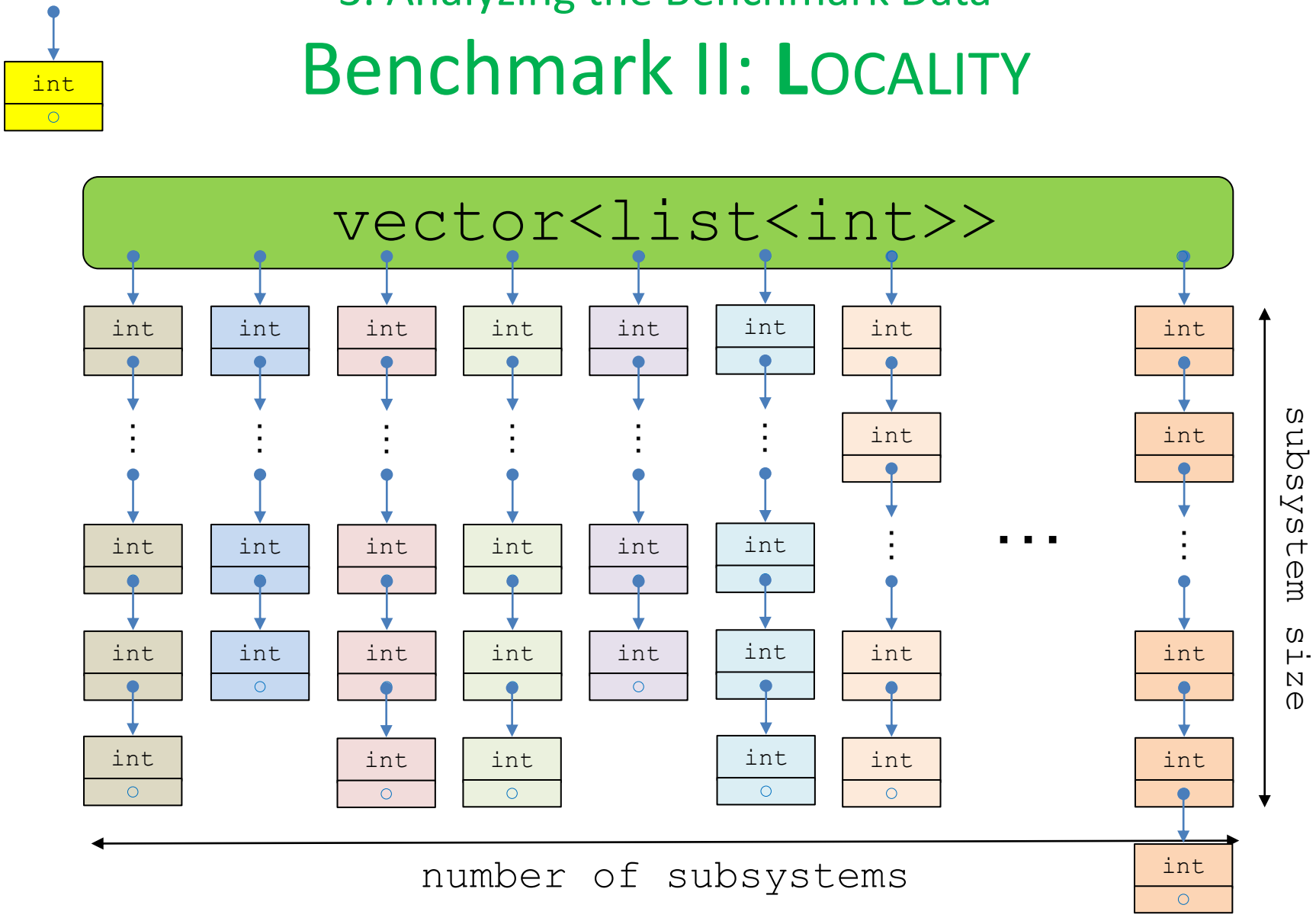
Benchmark II: LOCALITY



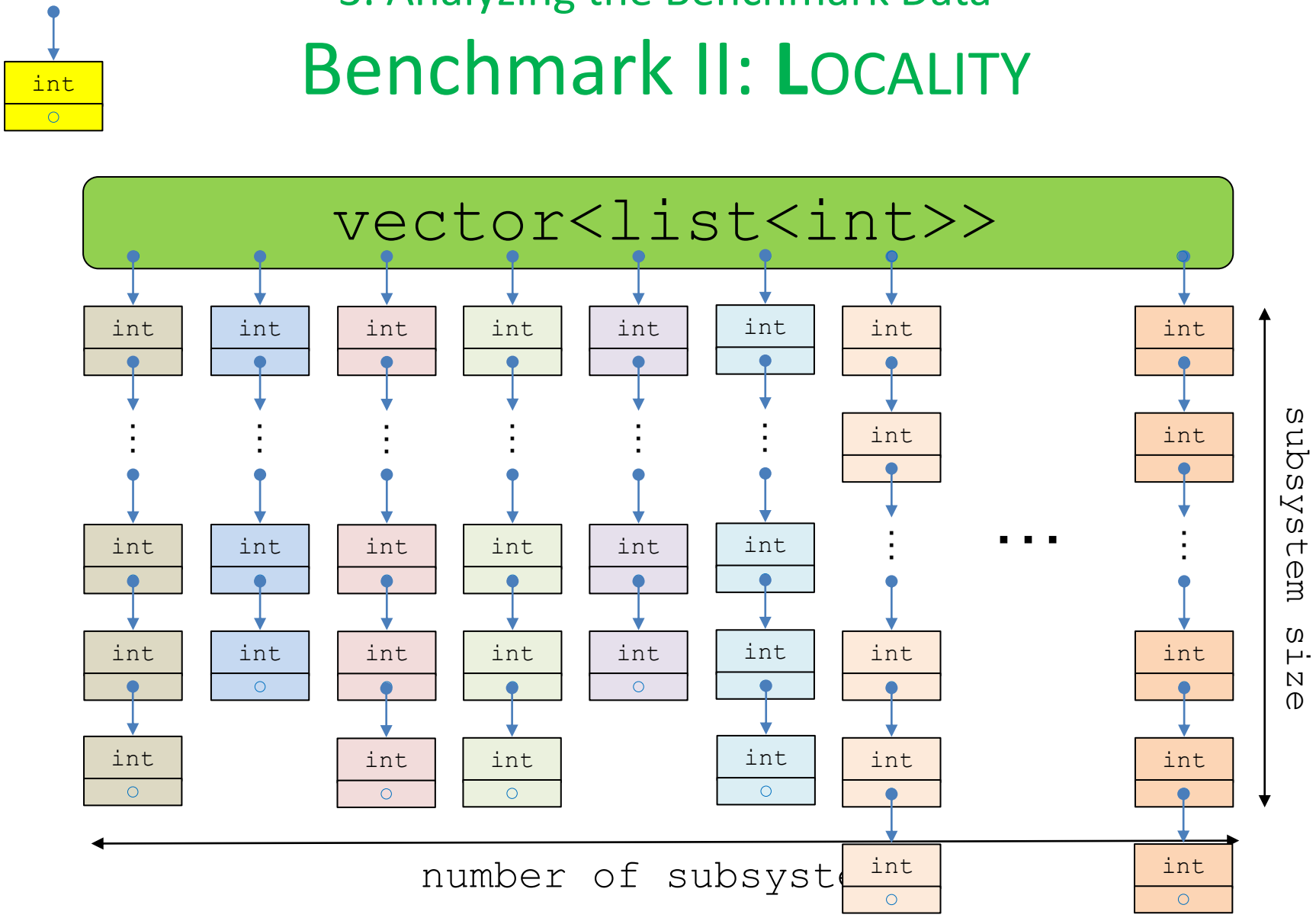
Benchmark II: LOCALITY



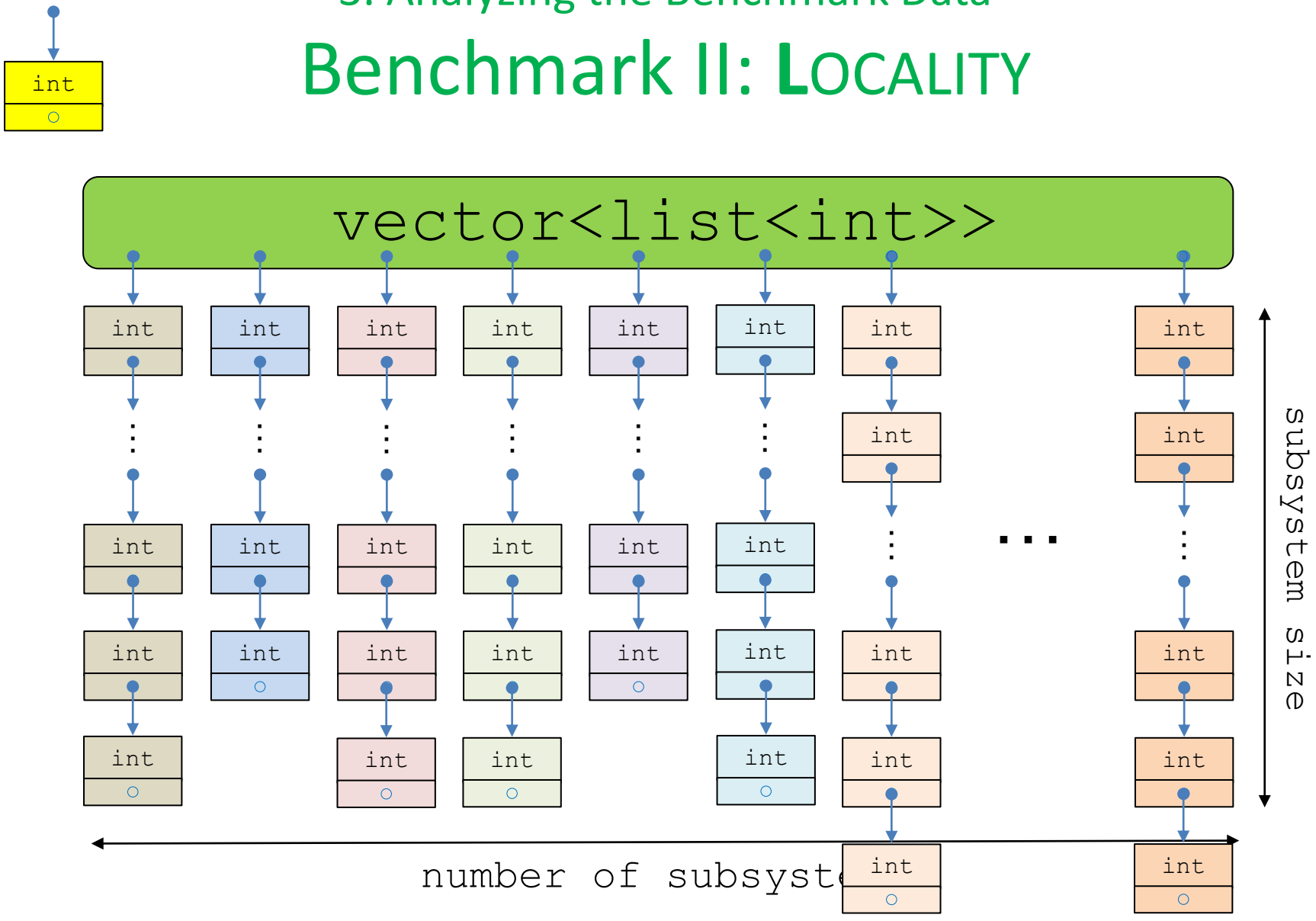
Benchmark II: LOCALITY



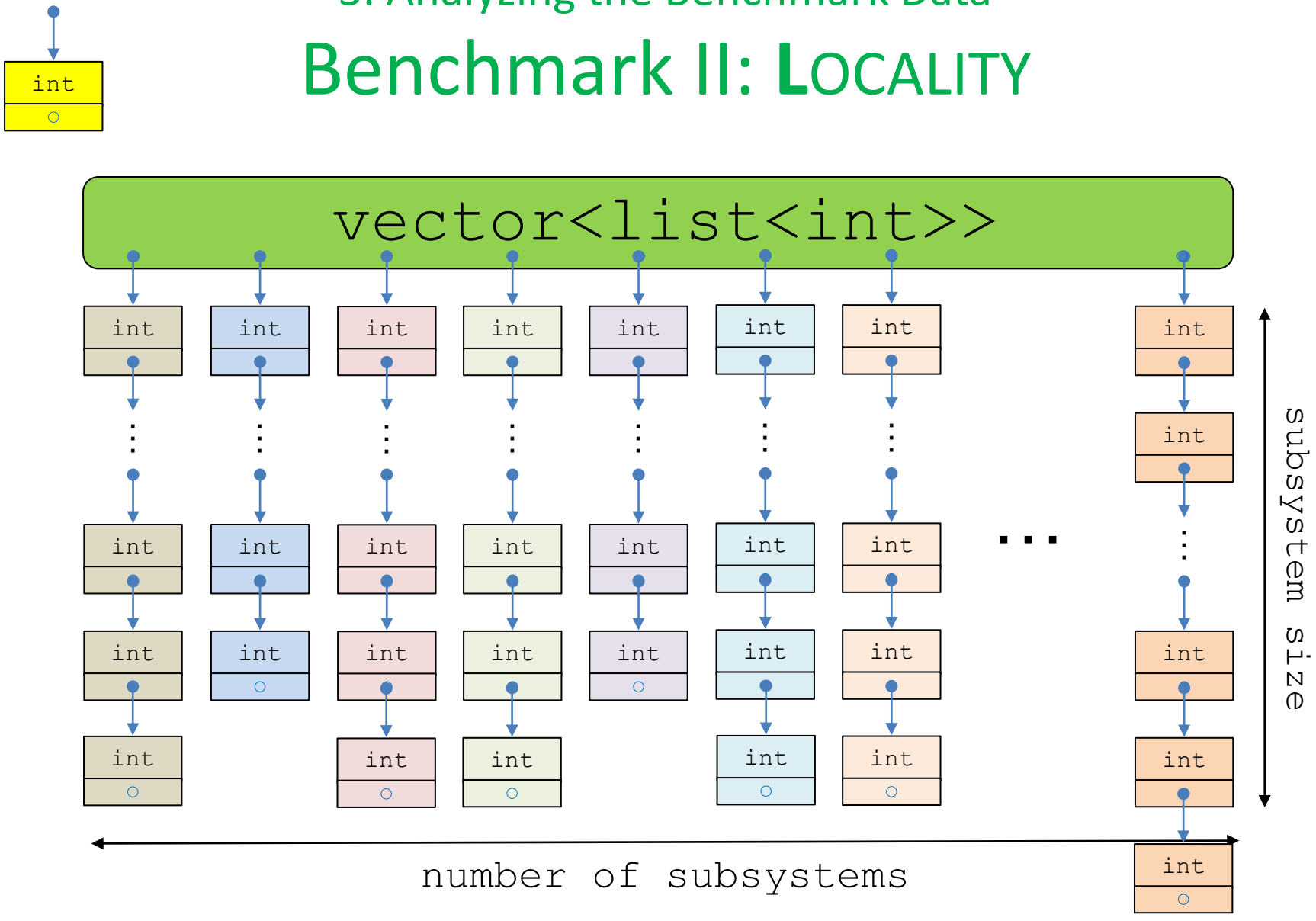
Benchmark II: LOCALITY



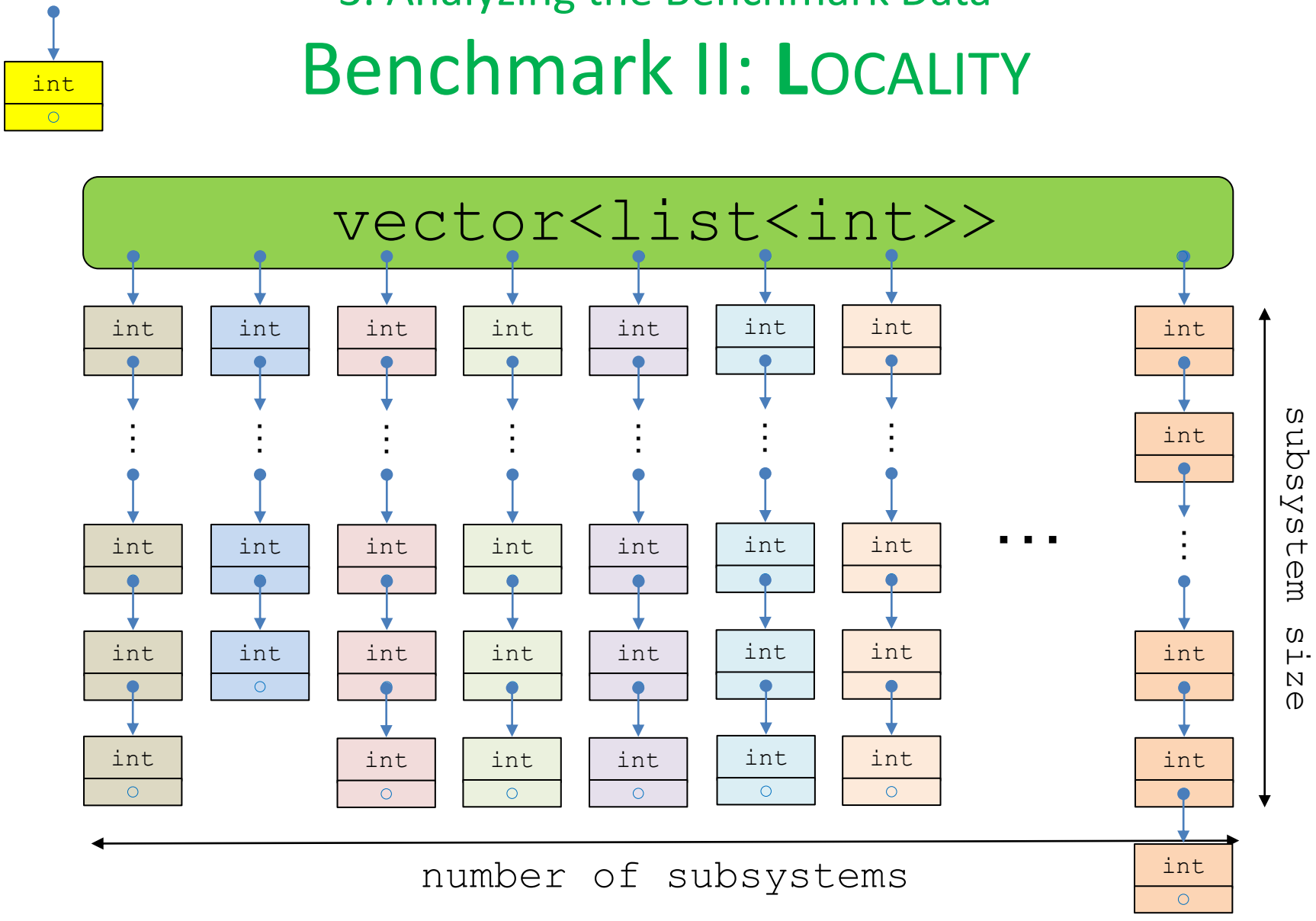
Benchmark II: LOCALITY



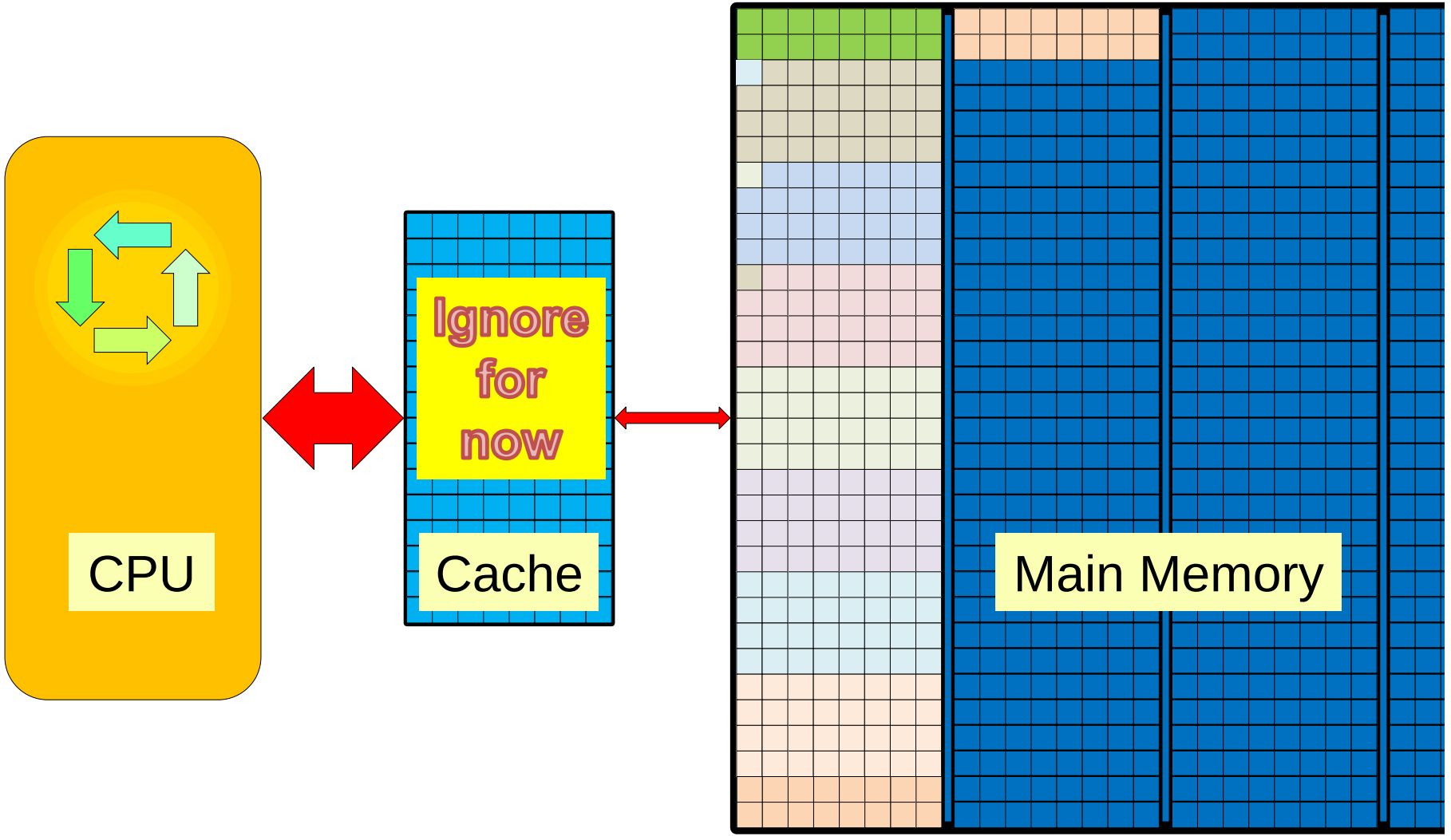
Benchmark II: LOCALITY



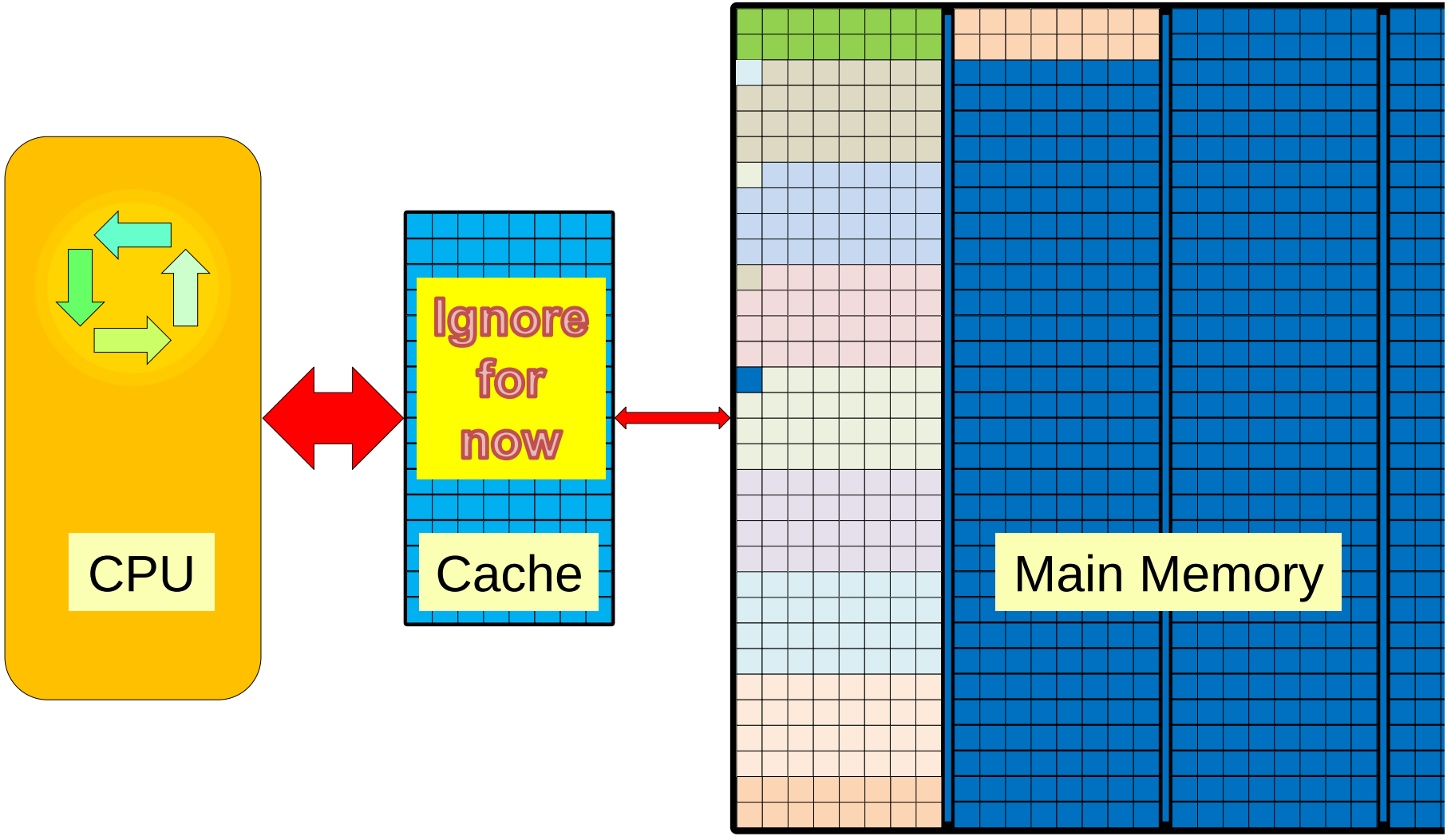
Benchmark II: LOCALITY



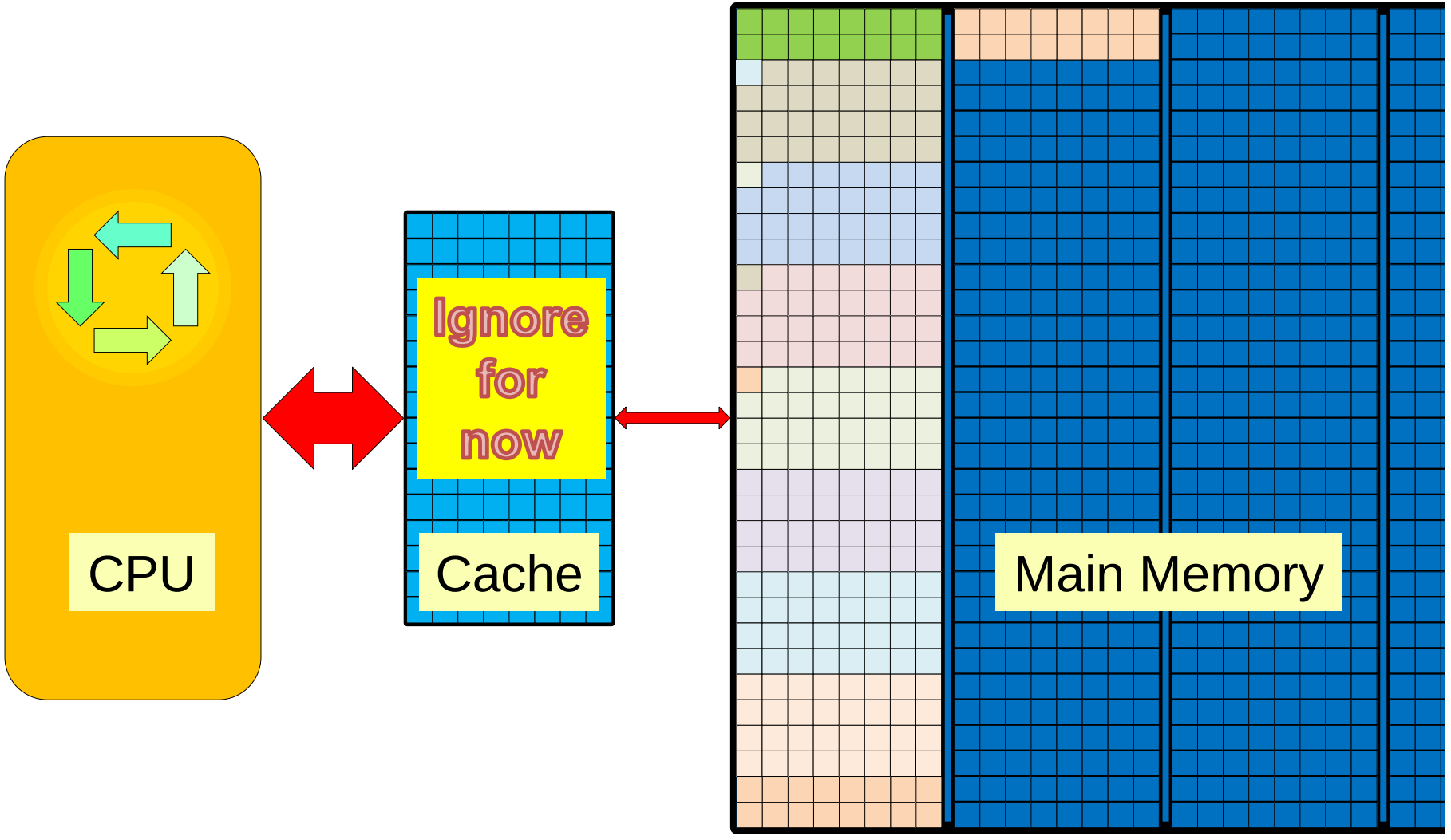
Benchmark II: LOCALITY



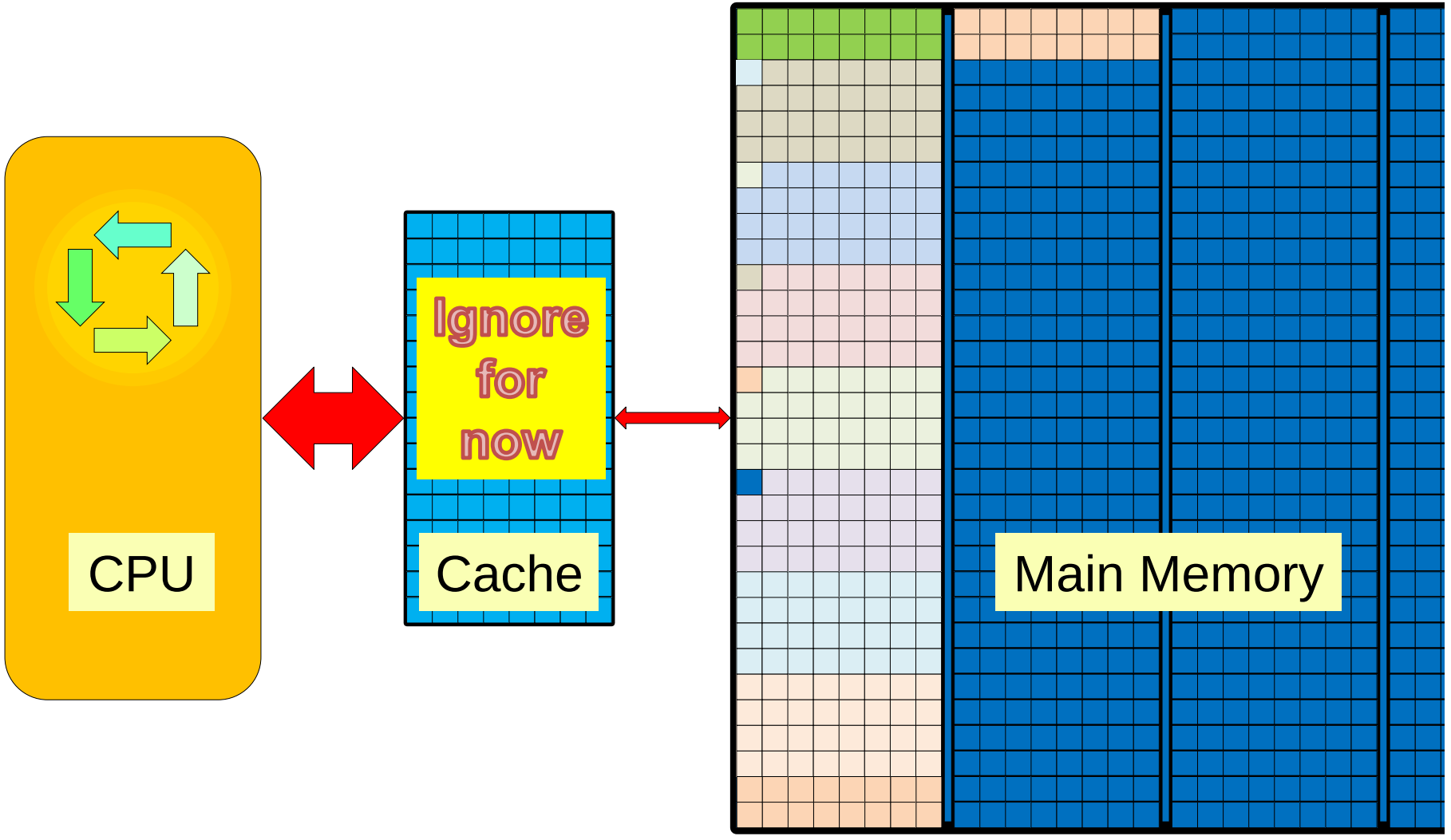
Benchmark II: LOCALITY



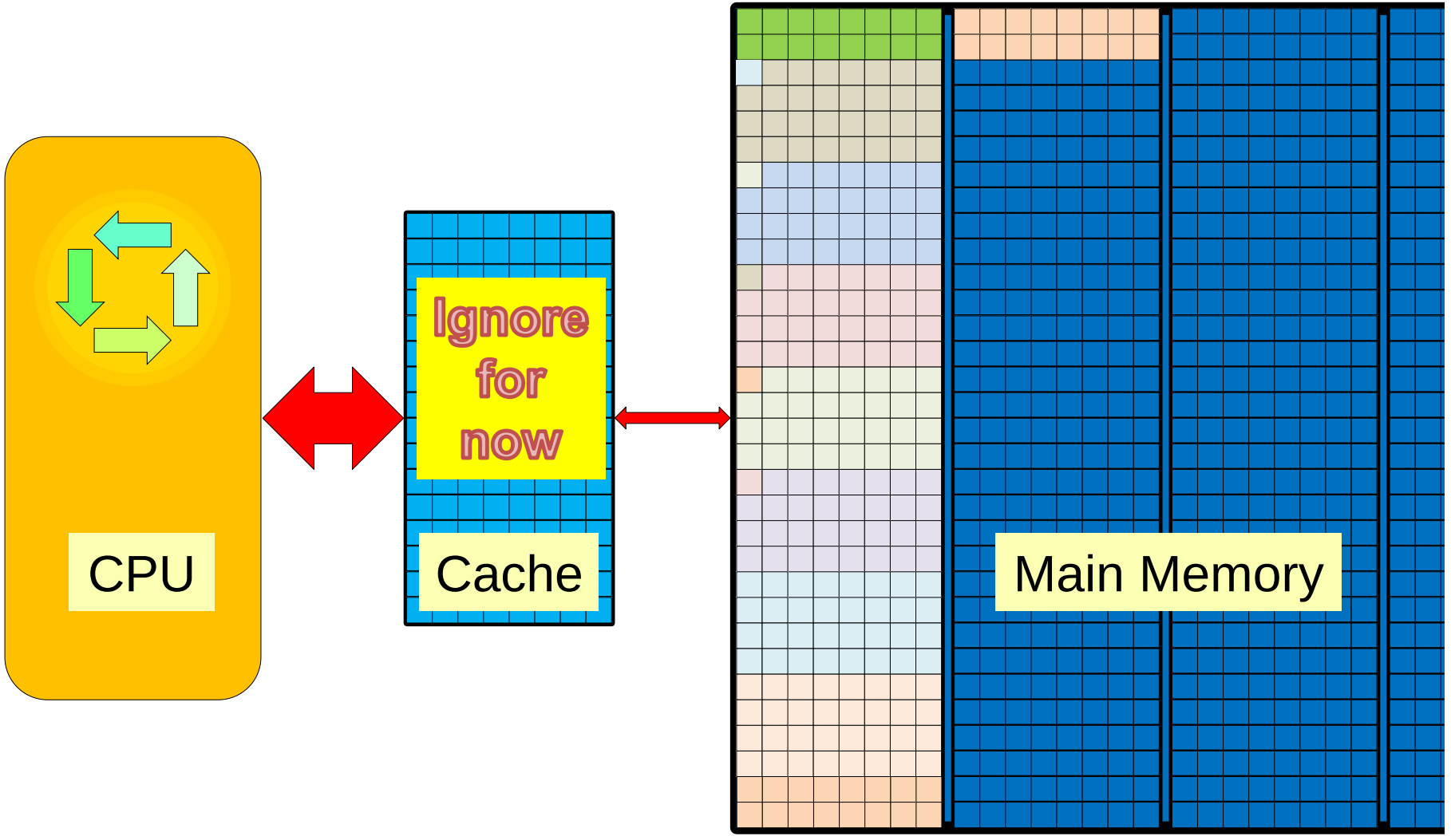
Benchmark II: LOCALITY



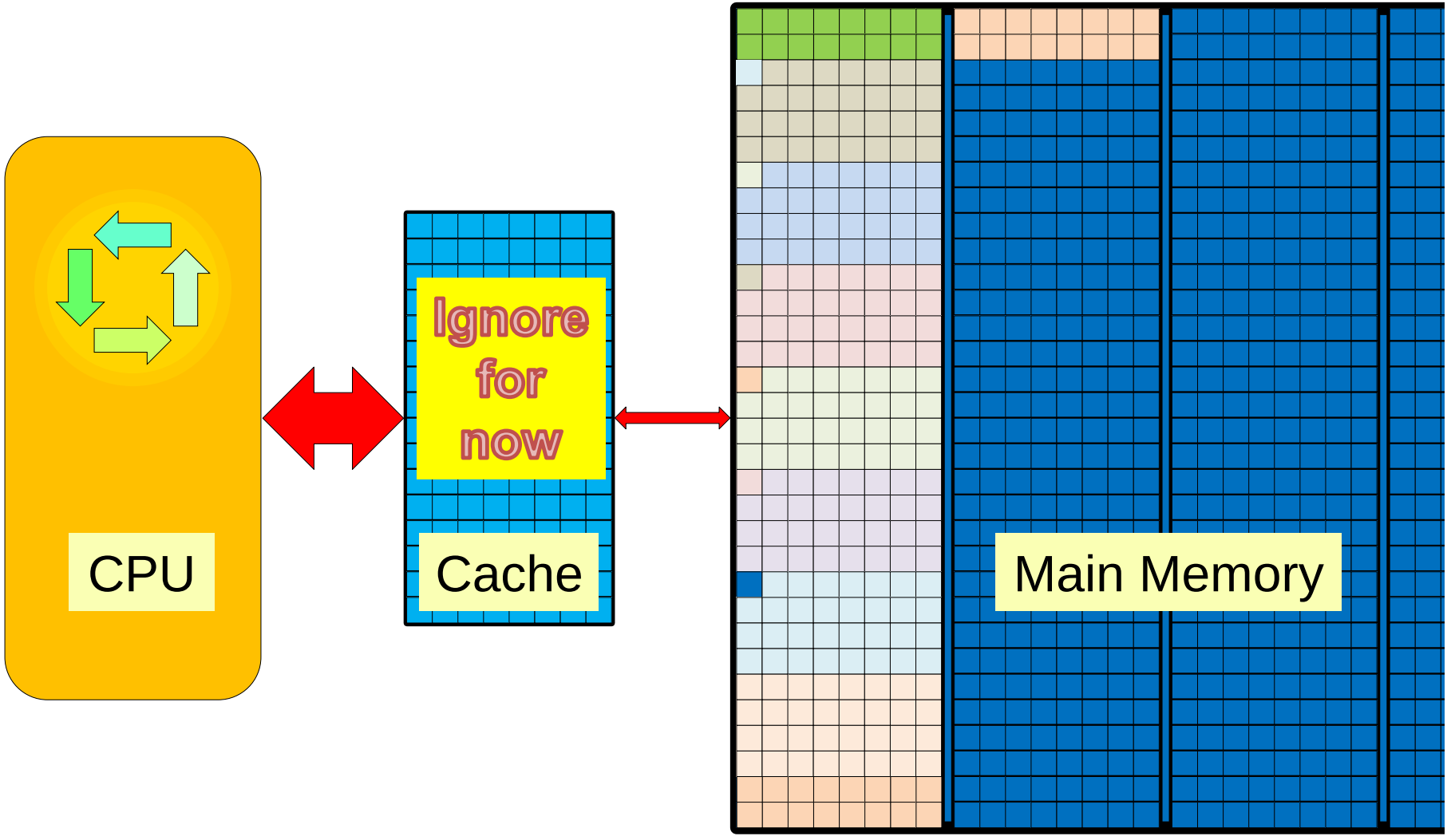
Benchmark II: LOCALITY



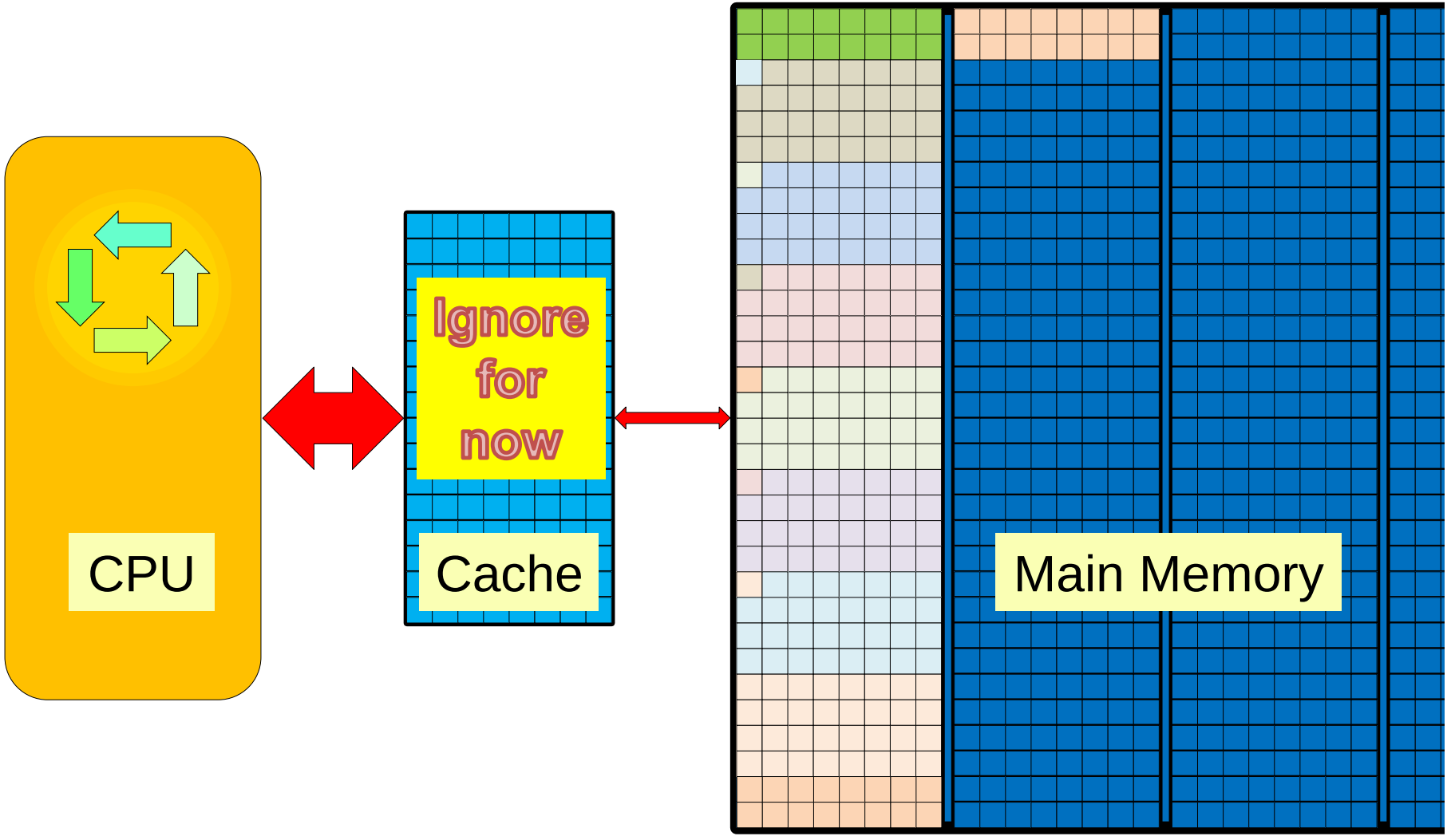
Benchmark II: LOCALITY



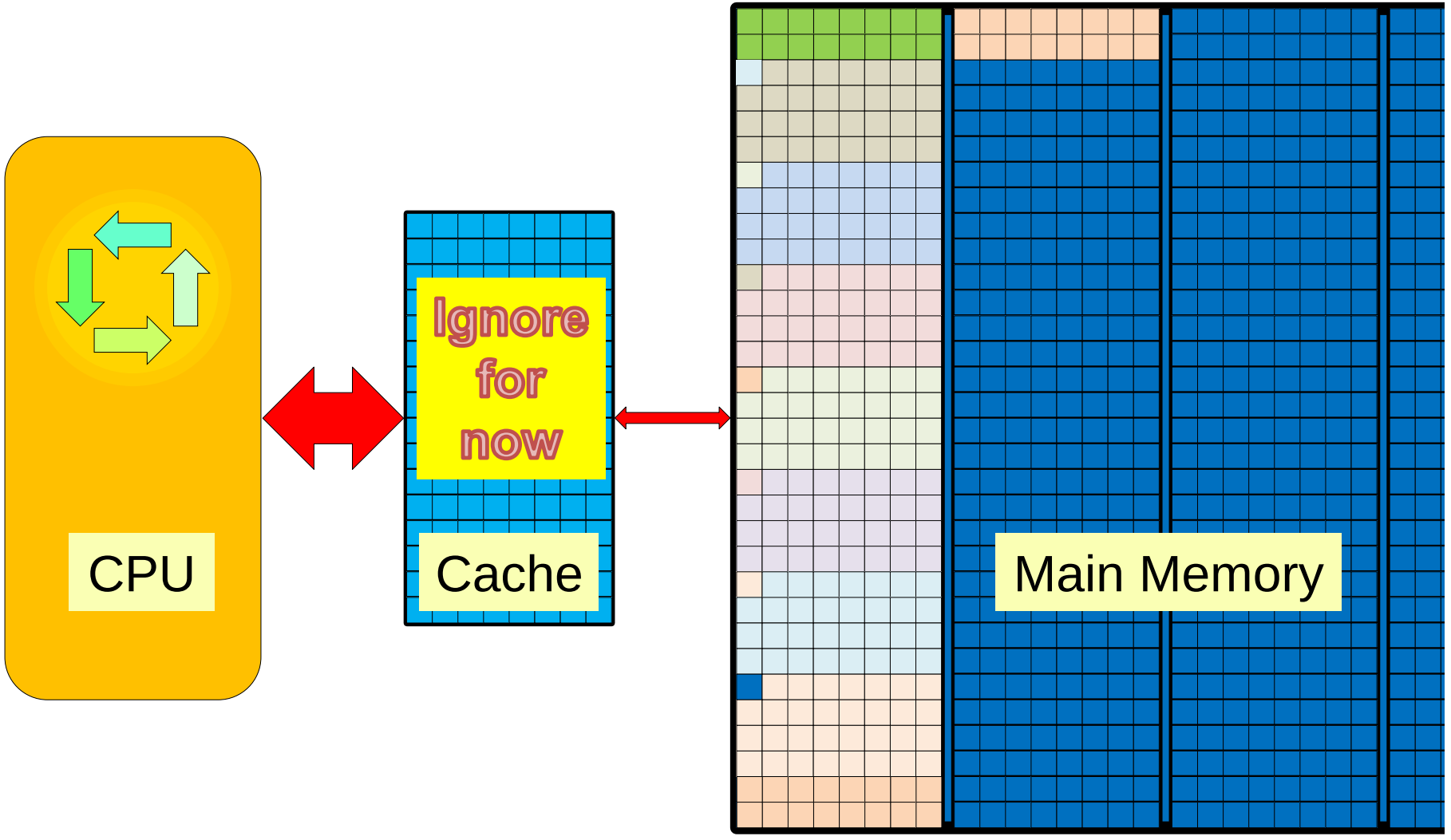
Benchmark II: LOCALITY



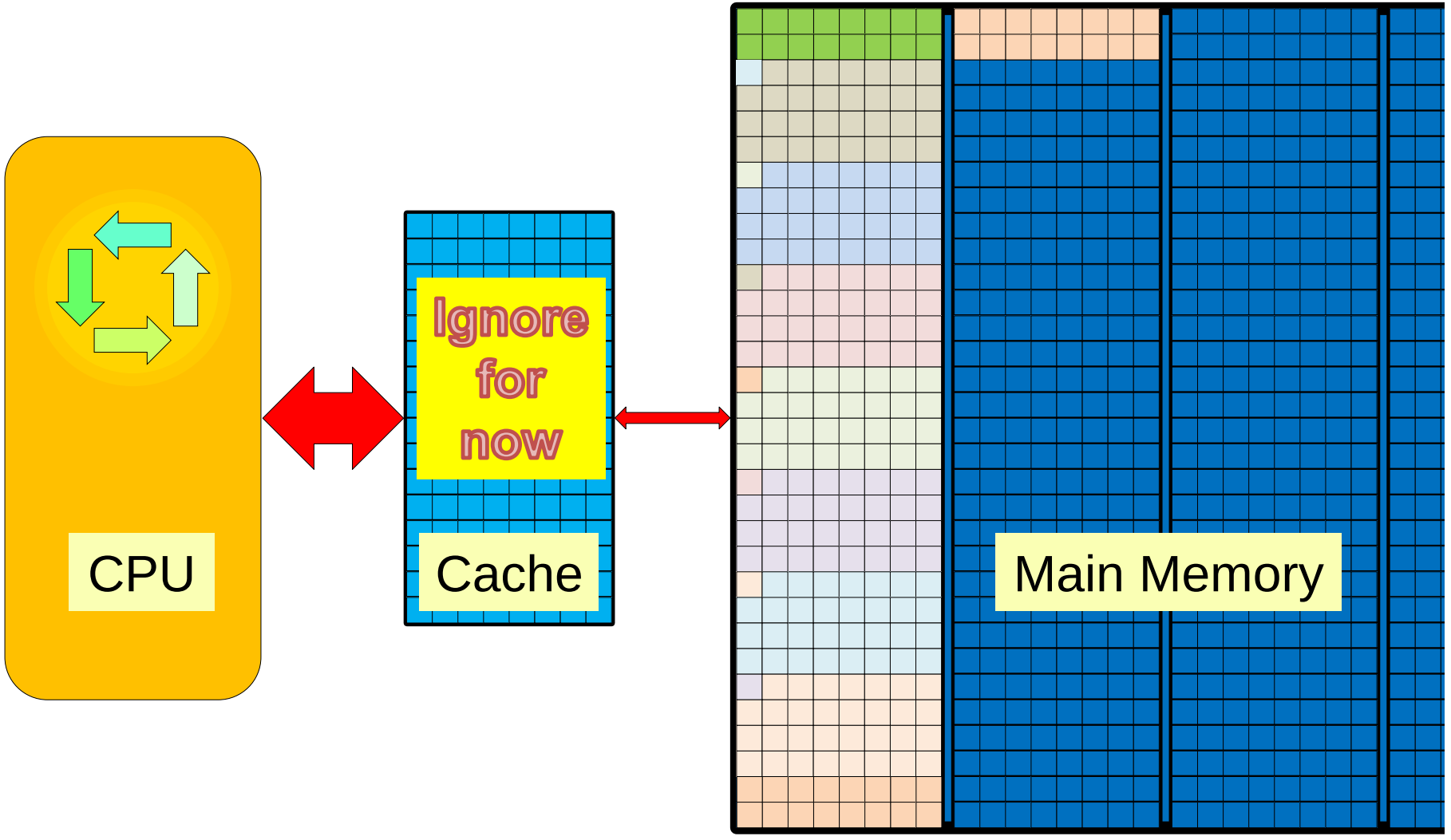
Benchmark II: LOCALITY

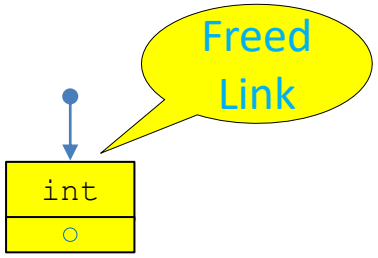


Benchmark II: LOCALITY



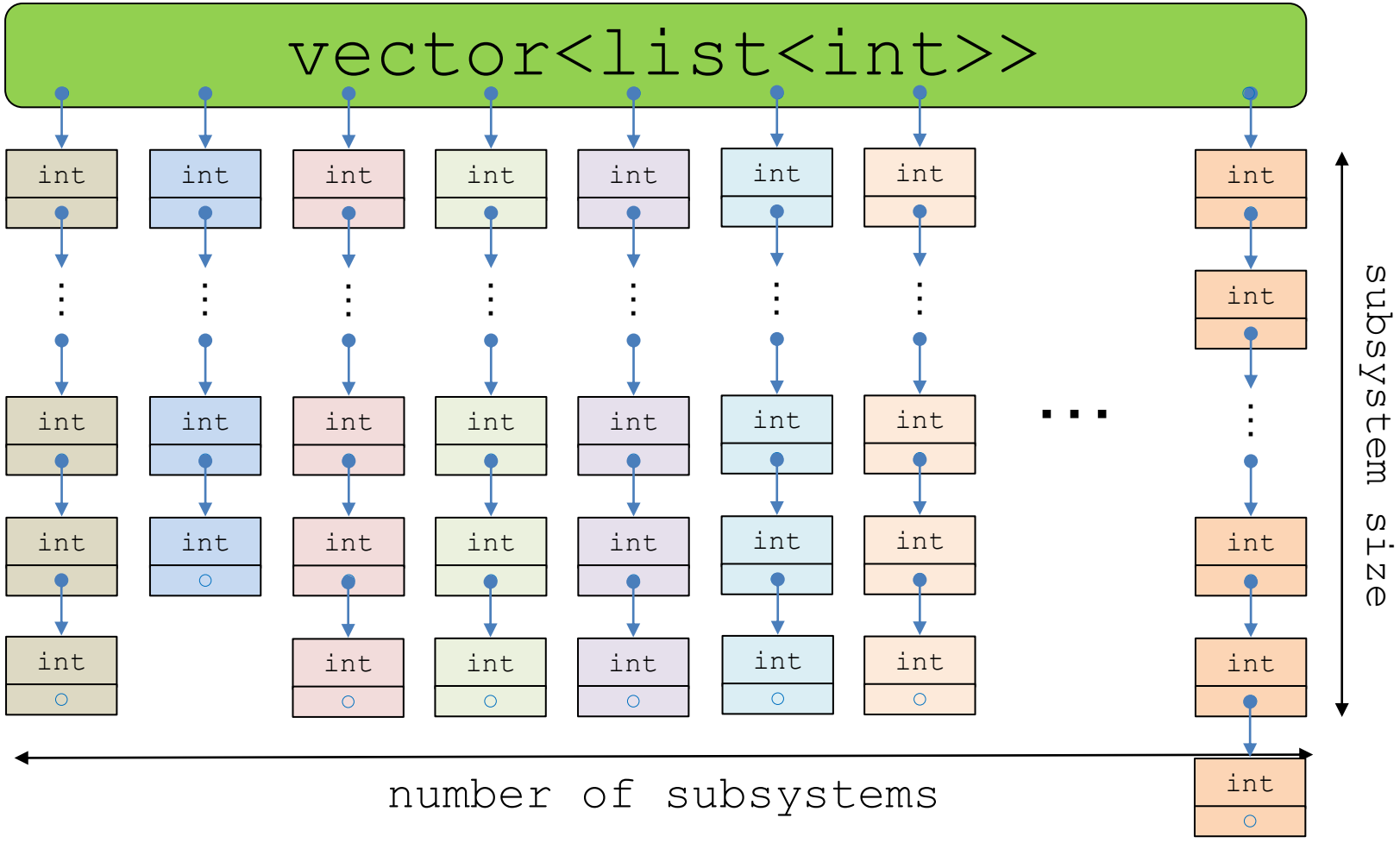
Benchmark II: LOCALITY



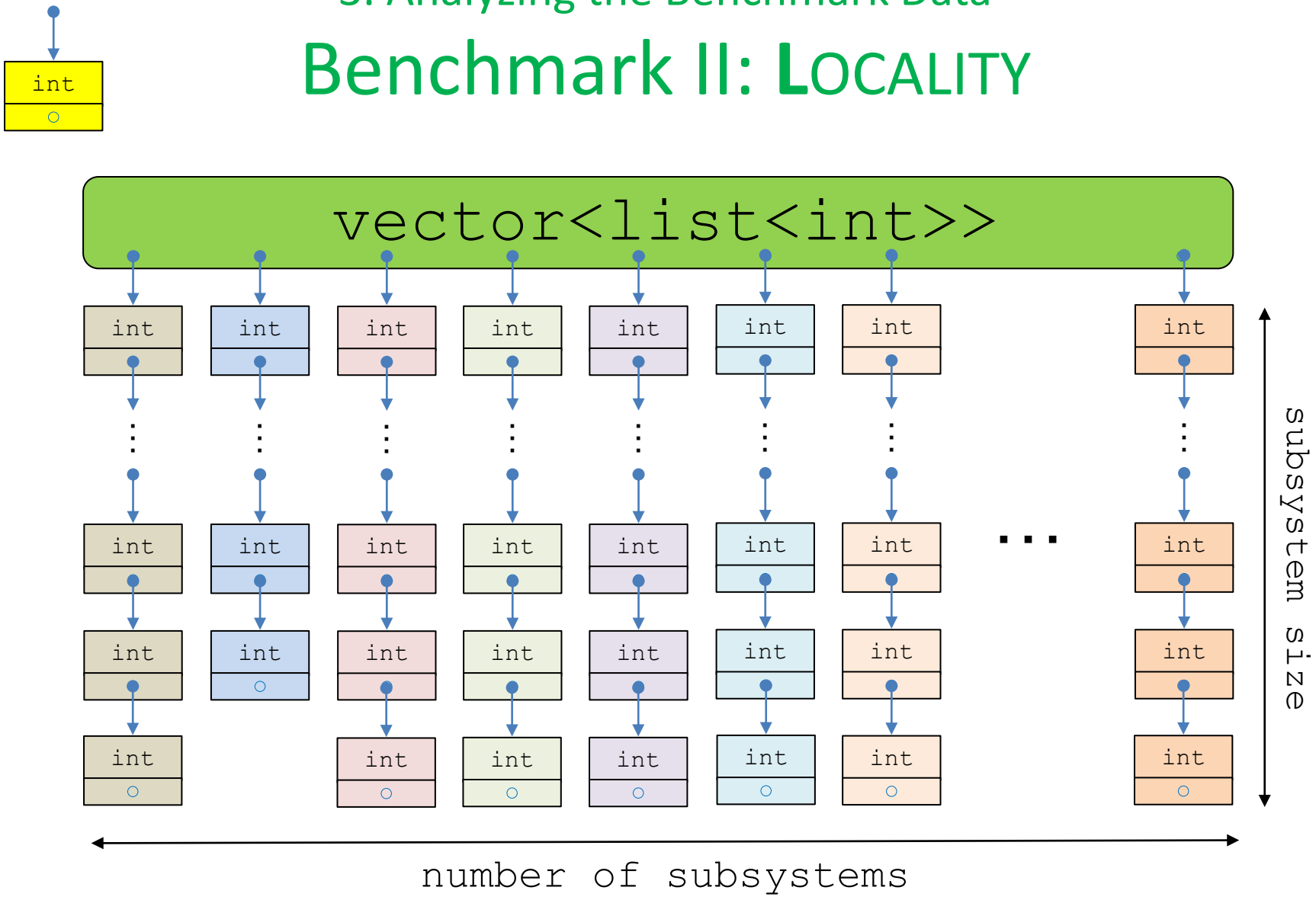


3. Analyzing the Benchmark Data

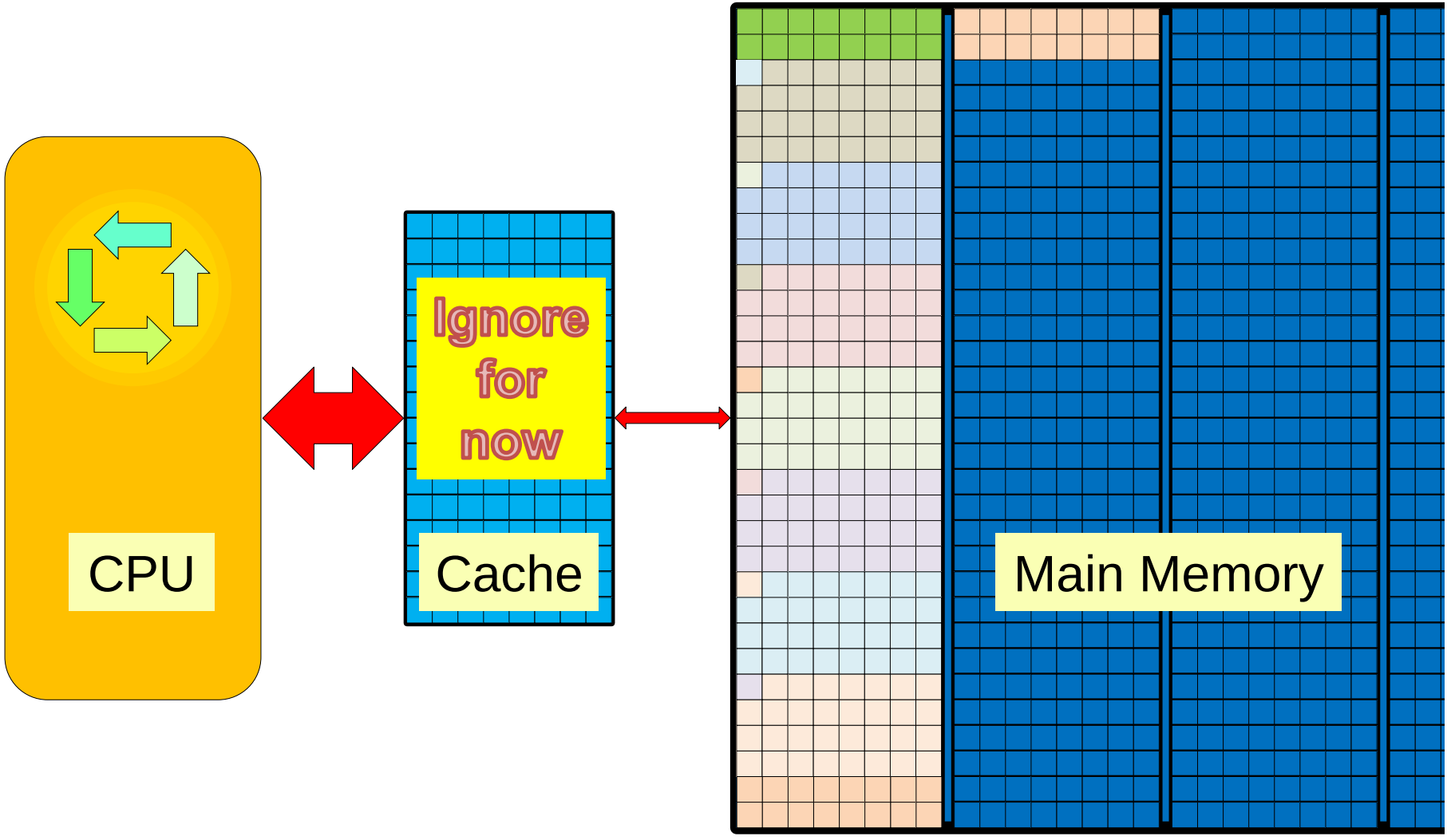
Benchmark II: LOCALITY



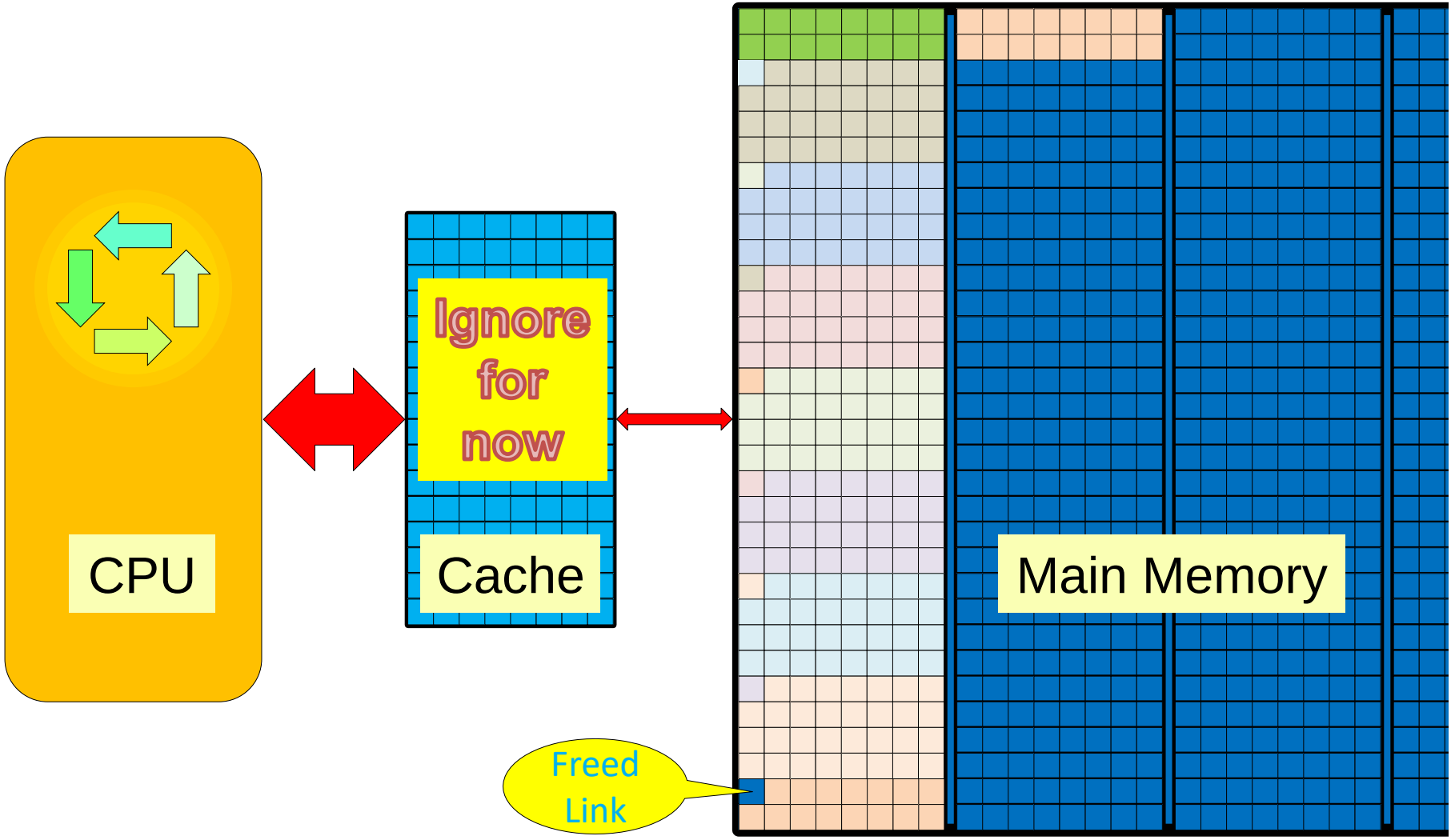
Benchmark II: LOCALITY



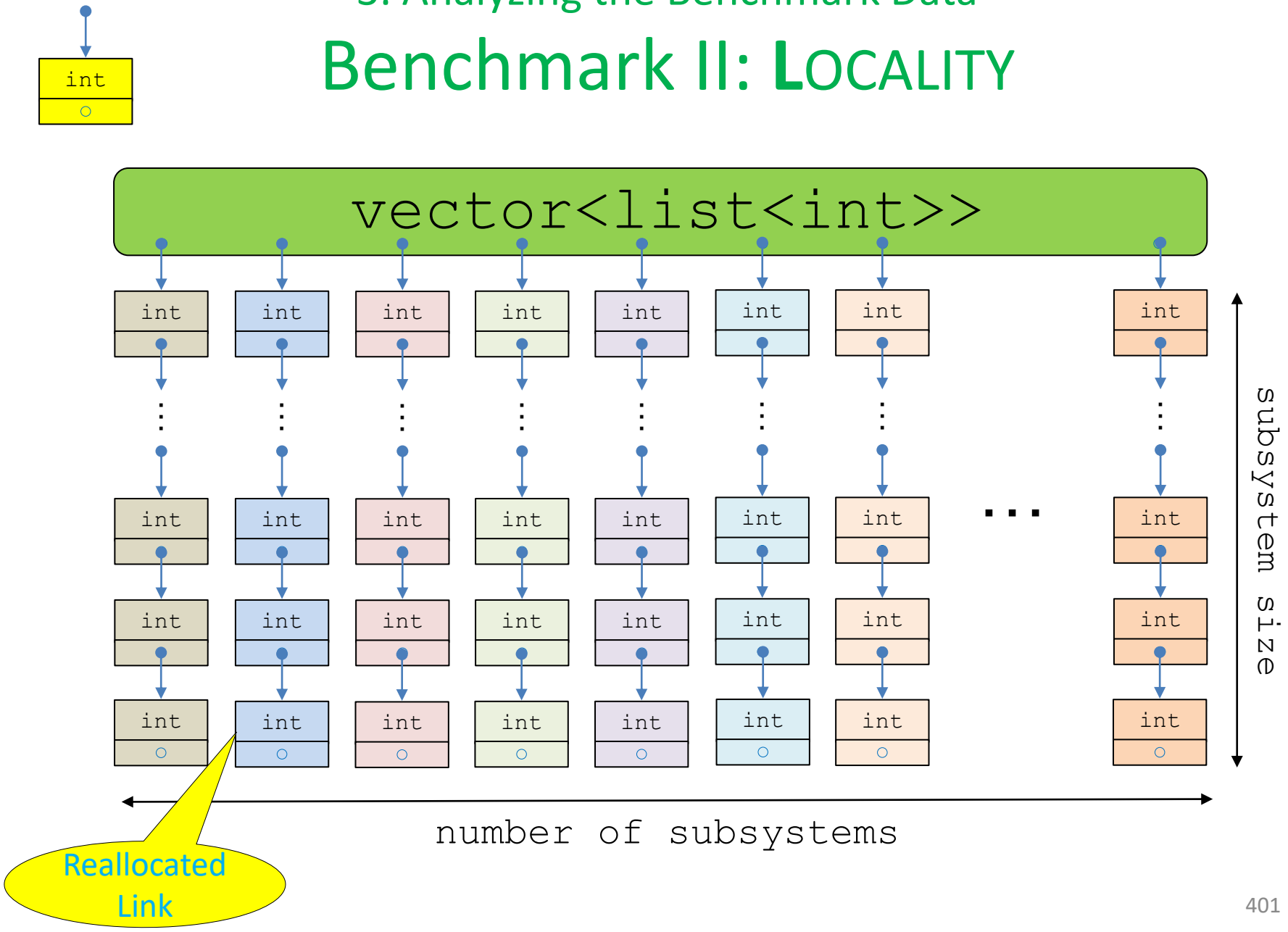
Benchmark II: LOCALITY



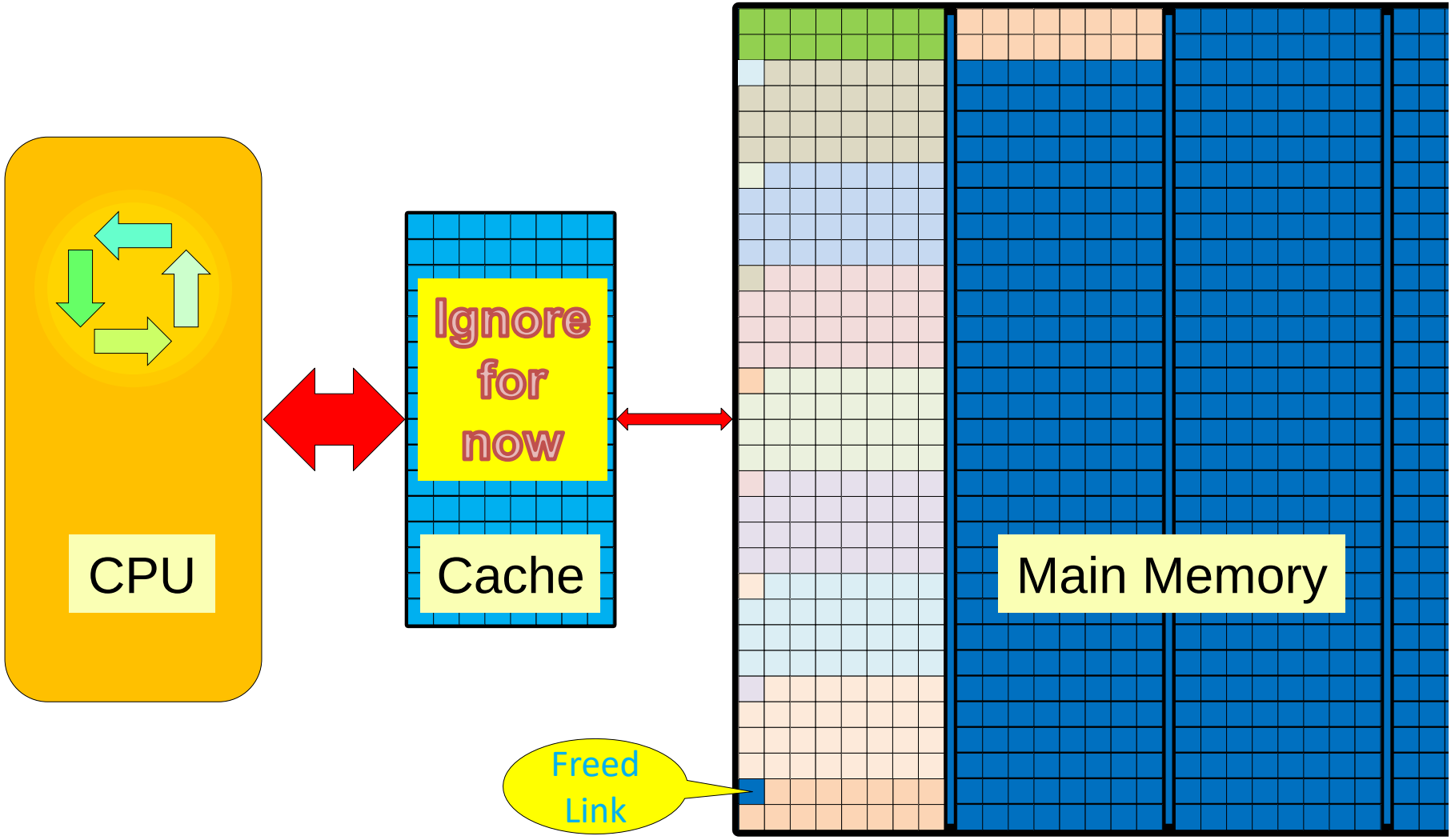
Benchmark II: LOCALITY



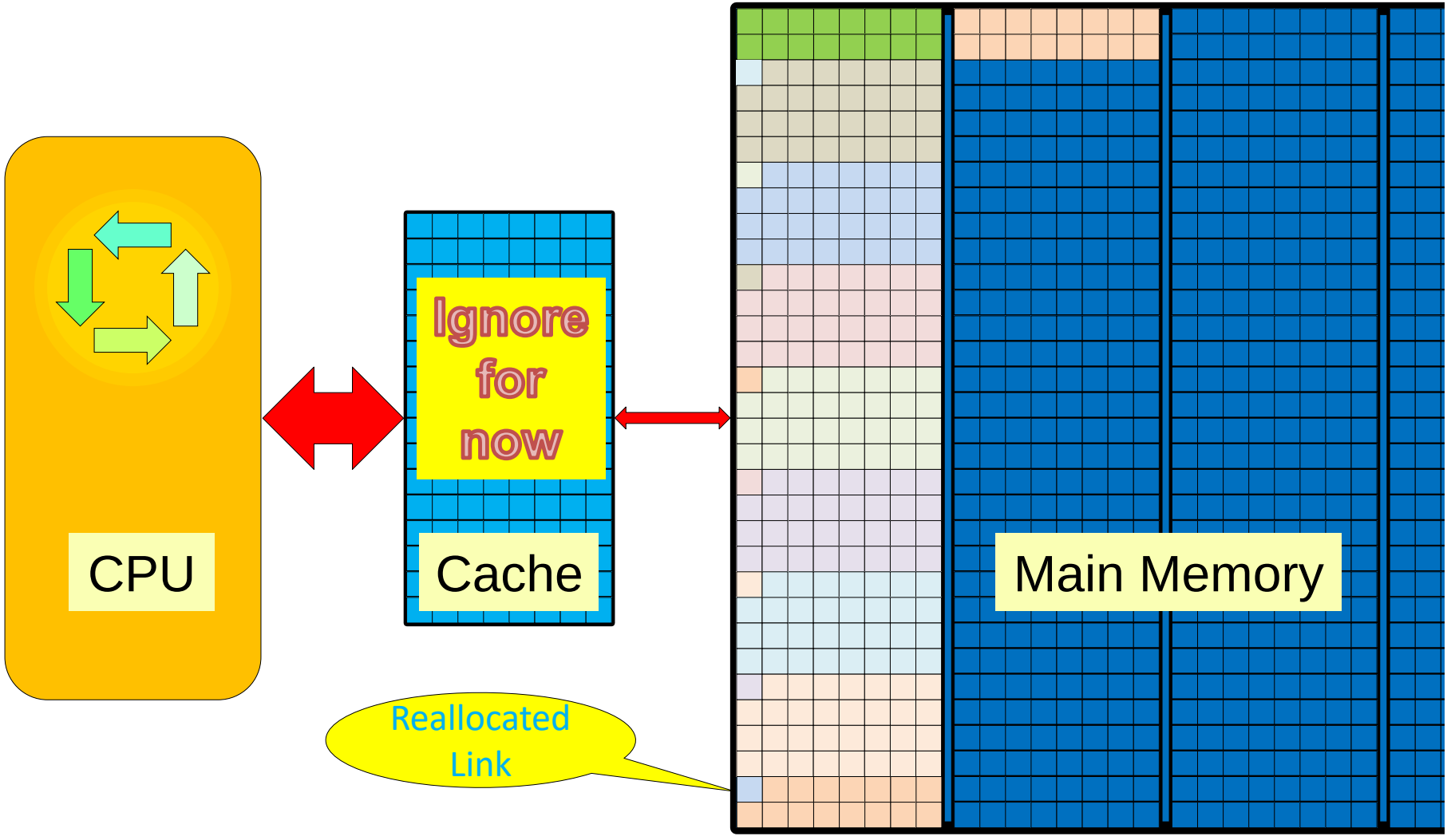
Benchmark II: LOCALITY



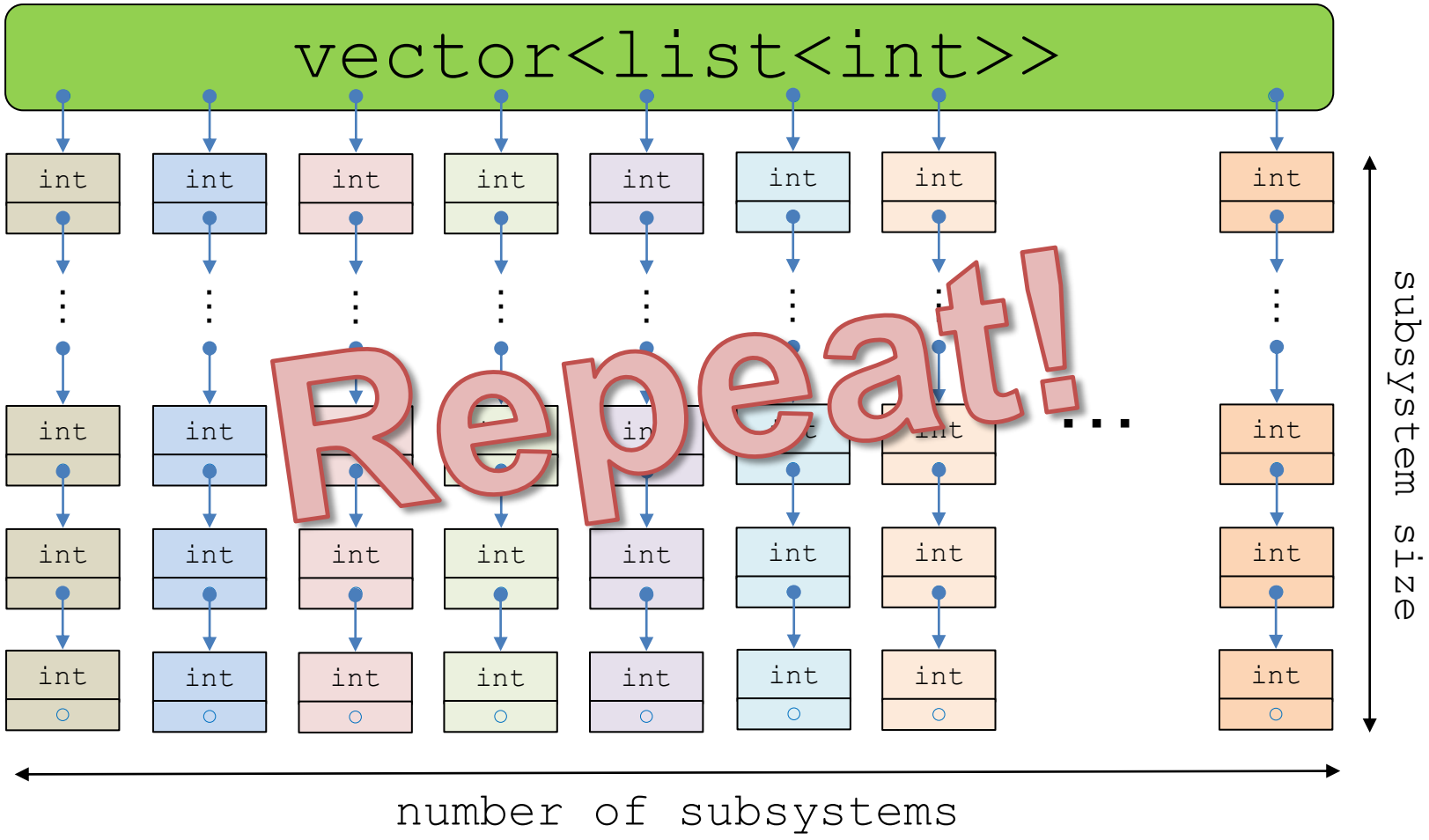
Benchmark II: LOCALITY



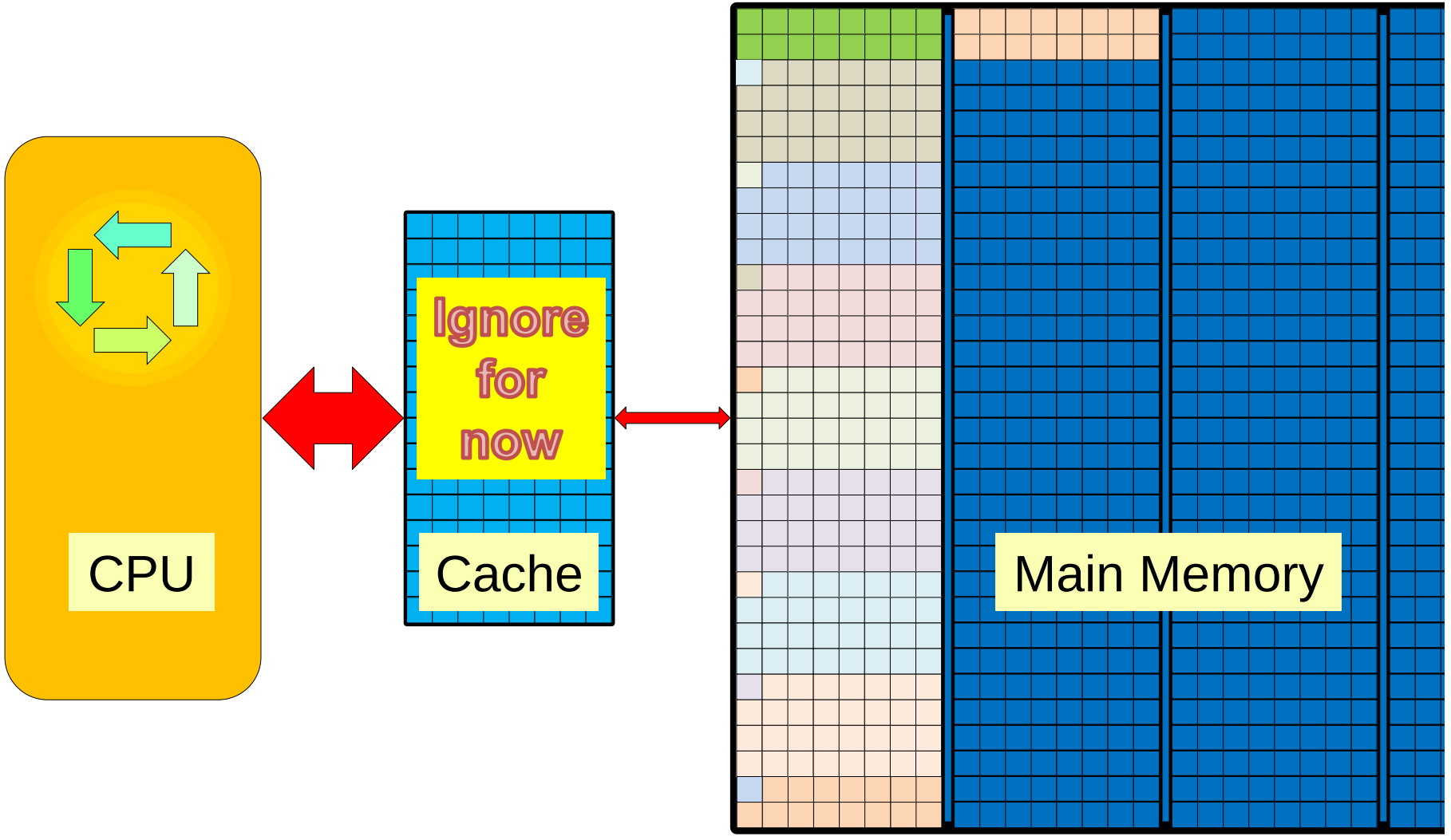
Benchmark II: LOCALITY



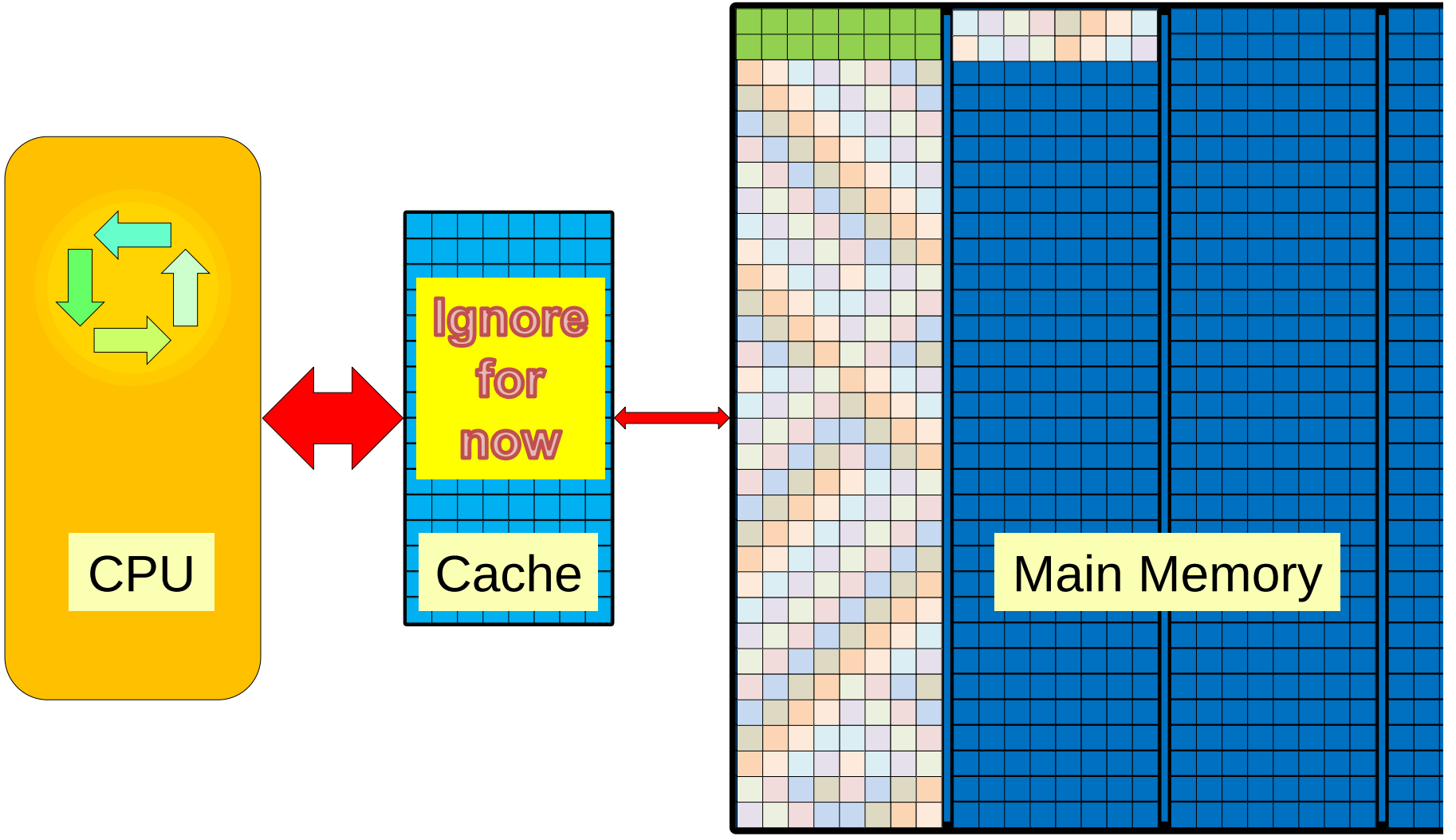
Benchmark II: LOCALITY



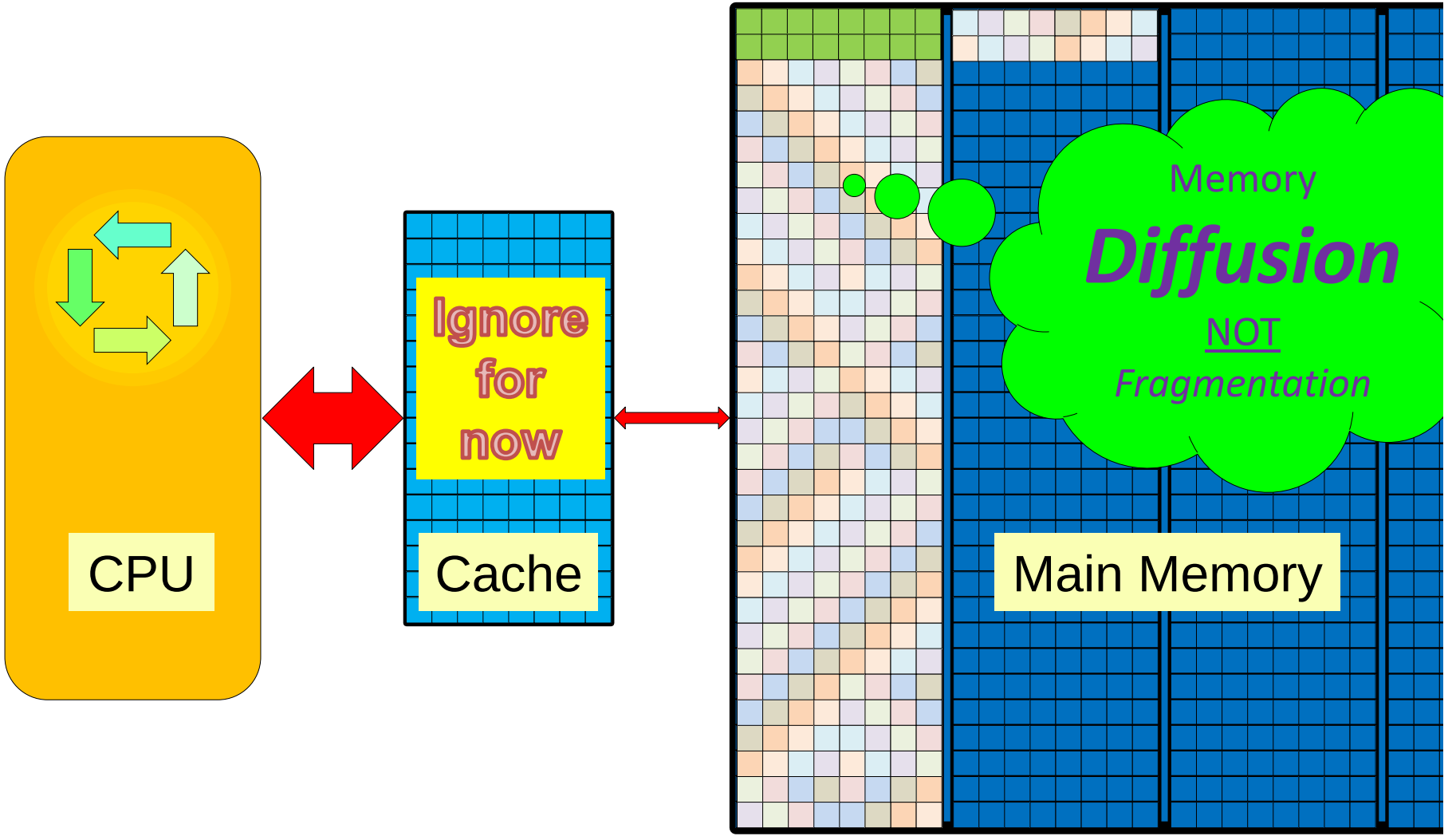
Benchmark II: LOCALITY



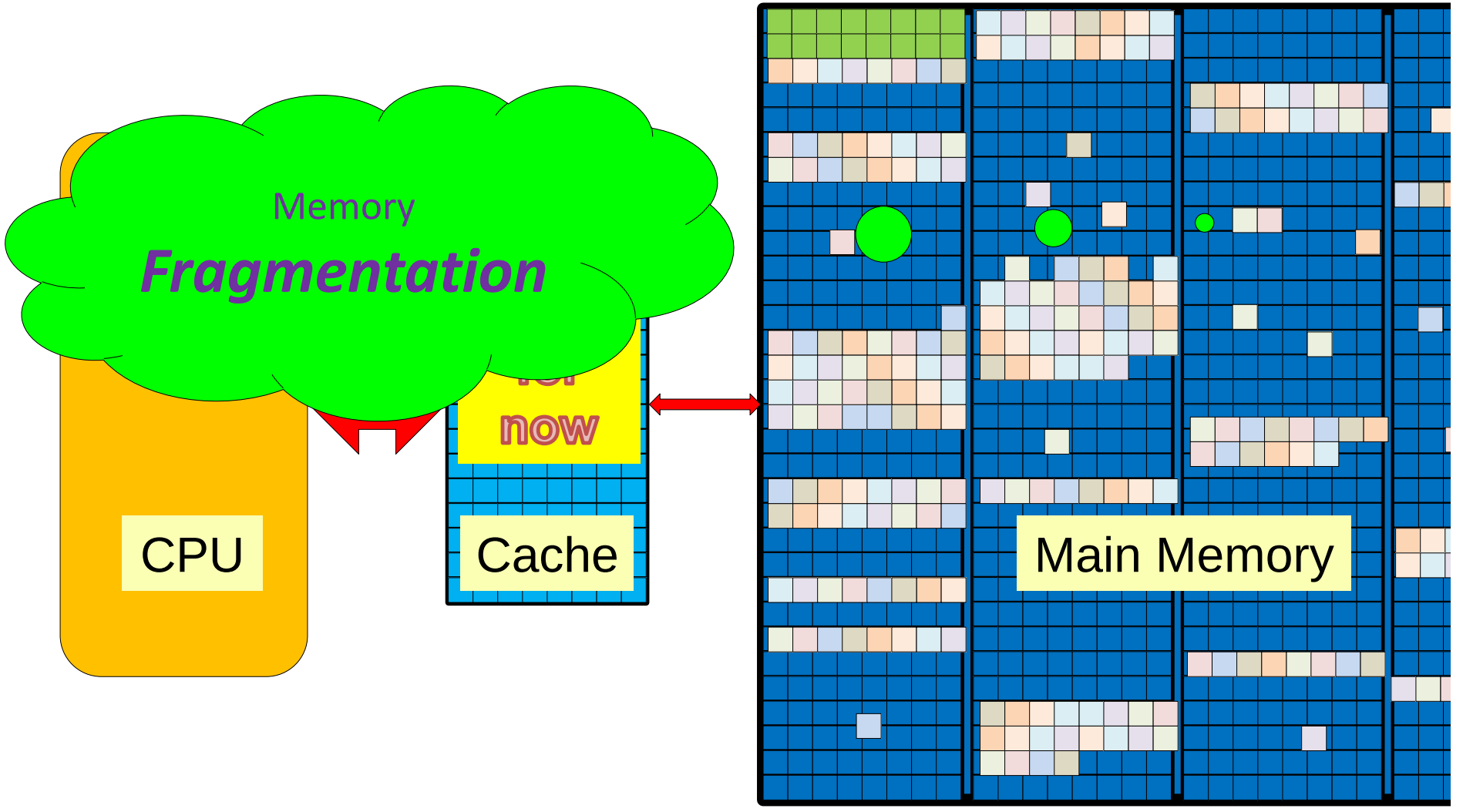
Benchmark II: LOCALITY



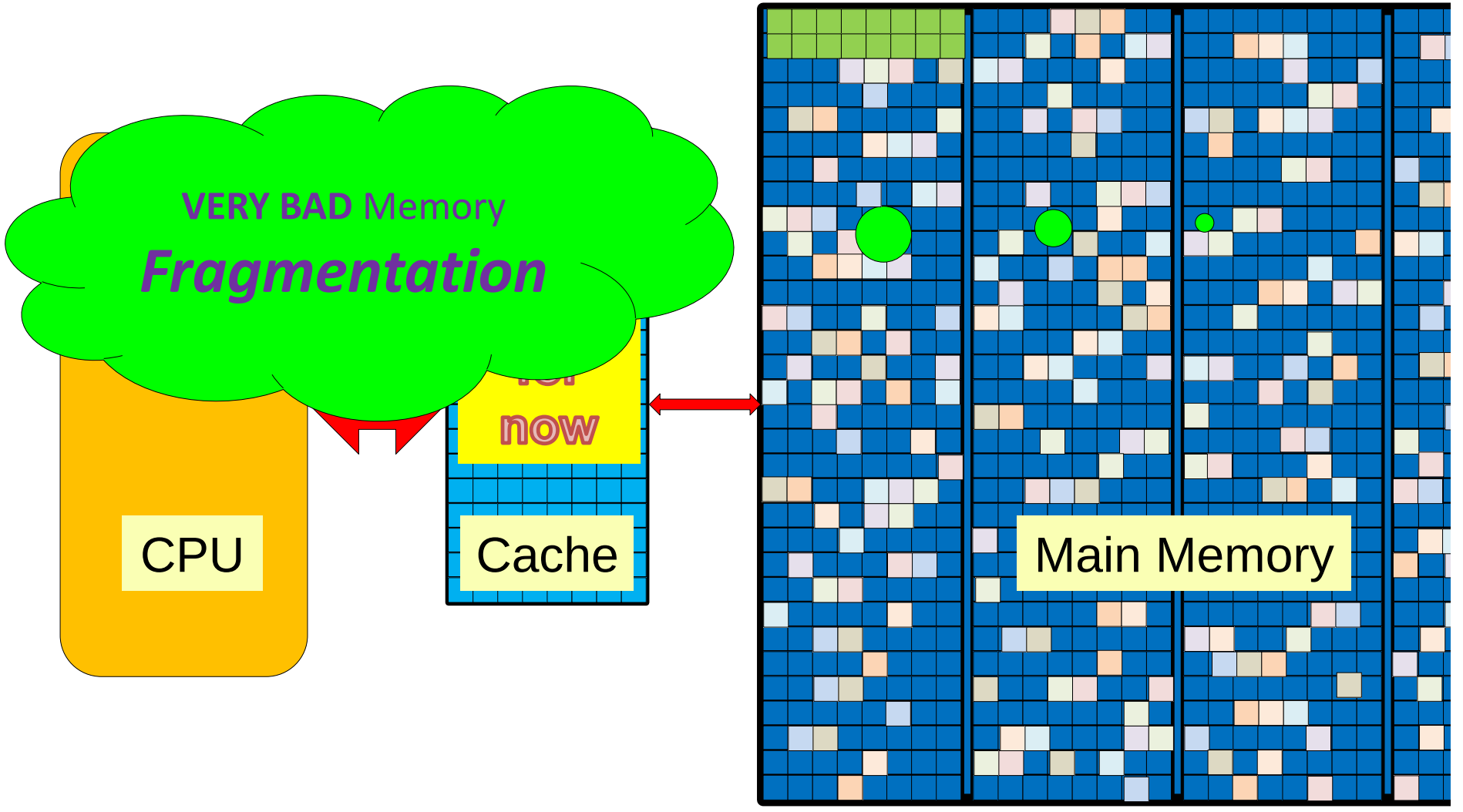
Benchmark II: LOCALITY



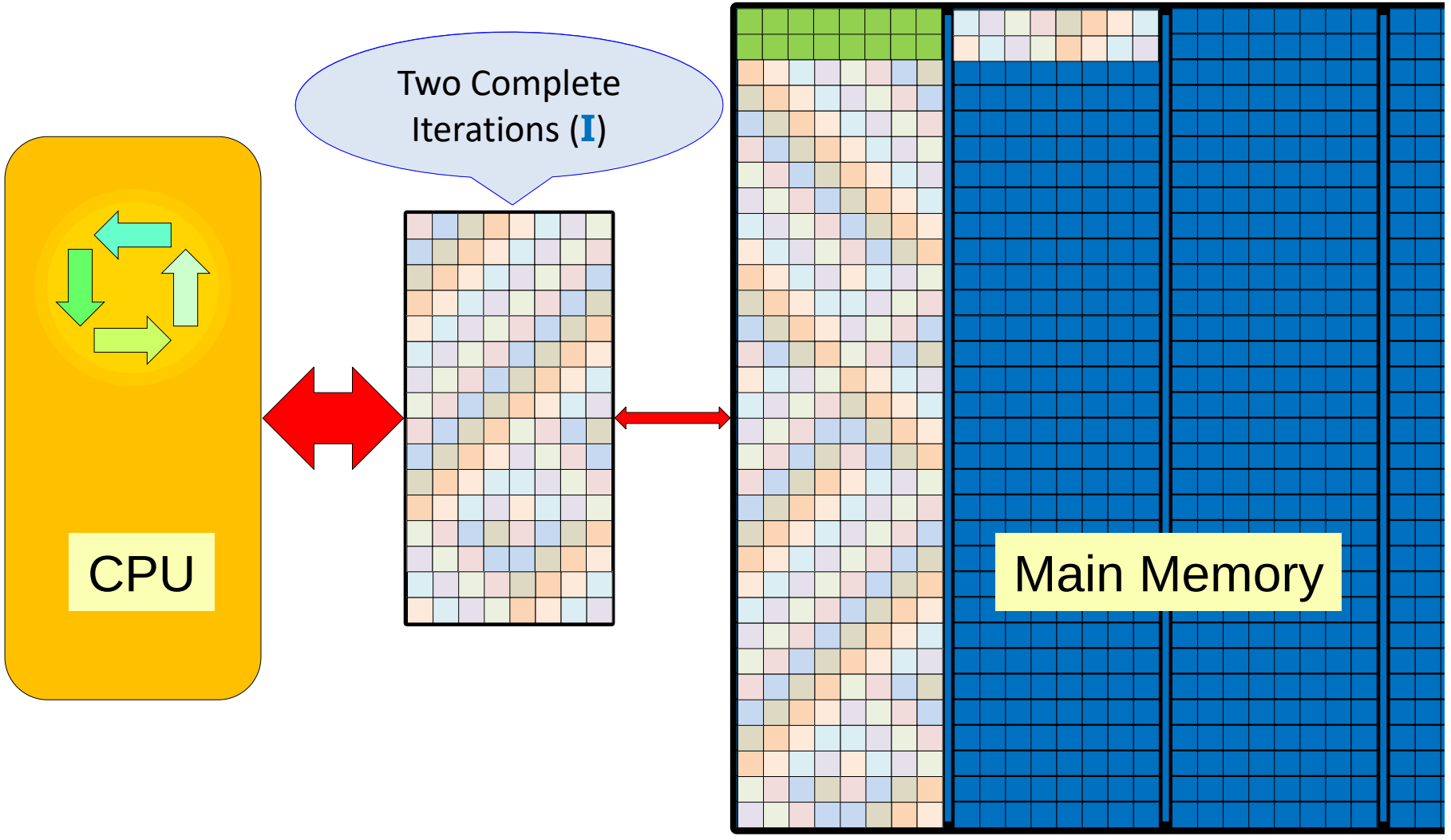
Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY

Overall (Multiprogram) Plan:

A. Create and Access **After** Shuffle:

B. Create and Access **Before** Shuffle.

C. Create and Shuffle **Only**

The result of each program is its (wall-clock) **RUN TIME.**

- **Shuffled**-Memory Access Time: $A - C$
- **Unshuffled**-Memory Access Time: $B - C$
- **Degradation Ratio** due to *memory diffusion*:

$$\textbf{Degradation Ratio } (DR) = \frac{A - C}{B - C}$$

Negligible!

411

Benchmark II: LOCALITY

Degradation Ratio (DR)

$$DR = \frac{A}{B} = \frac{access\ (A)fter}{access\ (B)efore}$$

$$DR = \frac{create \rightarrow shuffle \rightarrow access}{create \rightarrow access \rightarrow shuffle}$$

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

No Shuffling

Degradation Ratio (DR)

Iterations per Subsystem

I = 10

Temporal Locality

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

One subsystem of size 10^7

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

1 Complete Shuffle

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

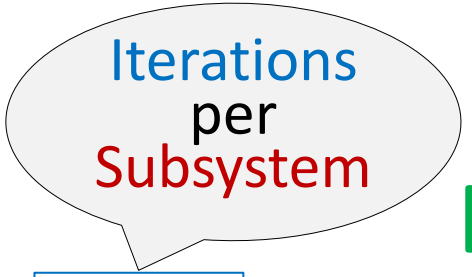
Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)



3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

I = 10

Temporal Locality

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

We'll use **5** complete shuffles from now on.

Number of Complete Shuffles (10^7 mo

Subsystem Size	0	1	2	3	4	5	6			
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)

Iterations per Subsystem

I = 10

Temporal Locality

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

Ten subsystems of size 10^6

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8	7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1	8.1	8.1	7.9	8.2	8.2	8.3	8.2
10^3	1.0	5.4	5.4	5.4	5.4	5.7	5.3	5.4	5.3	5.4
10^2	1.0	3.8	4.0	4.1	4.2	4.1	4.2	4.3	4.2	4.2
10^1	1.0	3.4	3.6	3.6	3.6	3.6	3.6	3.9	3.6	3.6
10^0	1.0	4.7	5.7	5.8	5.7	5.8	5.8	5.7	5.5	5.6

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Ten subsystems of size 10^6

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7		7.8	7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1							8.2
10^3	1.0	5.4								4
10^2	1.0	3.8								
10^1	1.0	3.4								.6
10^0	1.0	4.7	5.7	5.8					5.5	5.6

After one shuffle:

$$DR = \frac{A}{B} = 11.4$$

Degradation Ratio (DR)

Benchmark II: LOCALITY

Iterations per Subsystem

I = 10

Temporal Locality

Ten subsystems of size 10^6

Physical System Size **M** = **K** · **S** = 10^7

Diffusion

Number of Complete Shuffles (10^7 move operations)

Subsystem Size	0	1	2	3	4	5	6	7	8	9
10^7	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
10^6	1.0	11.4	15.6	16.2	16.0	15.8	15.6	15.8	15.7	16.2
10^5	1.0	7.7	7.8		7.8	8.0	8.0	8.0	8.0	8.2
10^4	1.0	8.0	8.1							8.2
10^3	1.0	5.4								4
10^2	1.0	3.8								
10^1	1.0	3.4								.6
10^0	1.0	4.7	5.7	5.8					5.5	5.6

After two shuffles:

$$DR = \frac{A}{B} = 15.6$$

Degradation Ratio (DR)

Benchmark II: LOCALITY

Questions and/or Discussion?

Benchmark II: LOCALITY

Trading off *physical* (**M**) and *temporal* (**A**) values.

(**N** = 45) = (**M** = 21) * (**A** = 24)

2^{45} accesses

Subsystem Size (**S**)	*Number of Subsystems* (**K**)
0	21
1	20
2	19
3	18
:	:
:	:
20	2
21	1

X

Number of Iterations (**I**)	*Number of Repetitions* (**R**)
8	16
7	17
6	18
5	19
4	20
3	21
2	22
1	23
0	24

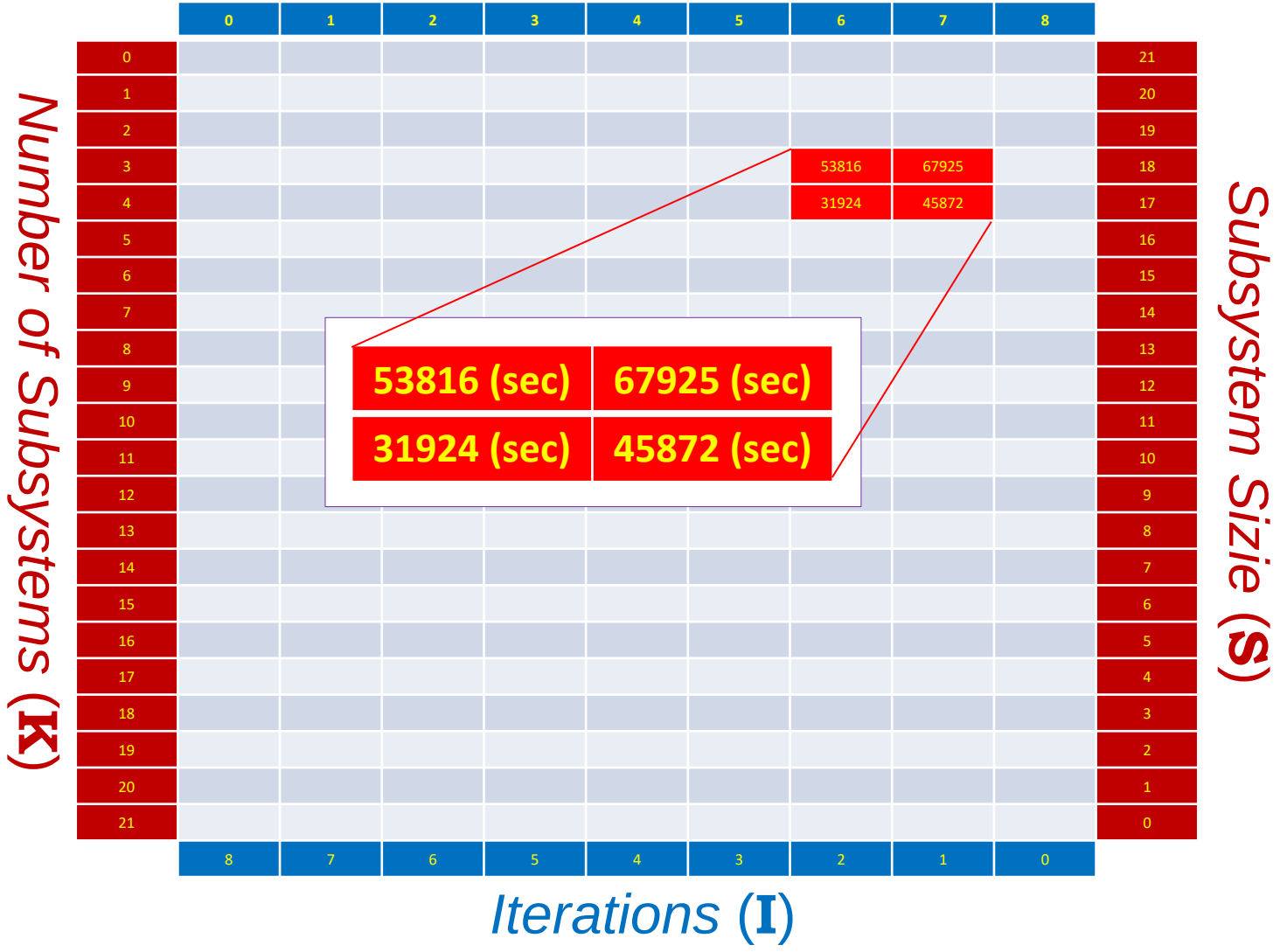
The larger the subsystem (**S**), the fewer there are (**K**).

The fewer the iterations (I), the more repetitions (R).

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

Large_Number * *Repetitions* (**R**)



Benchmark II: LOCALITY

System Size = 2^{21}

Without Local Allocators

shuffle = 5

(Diffusion)

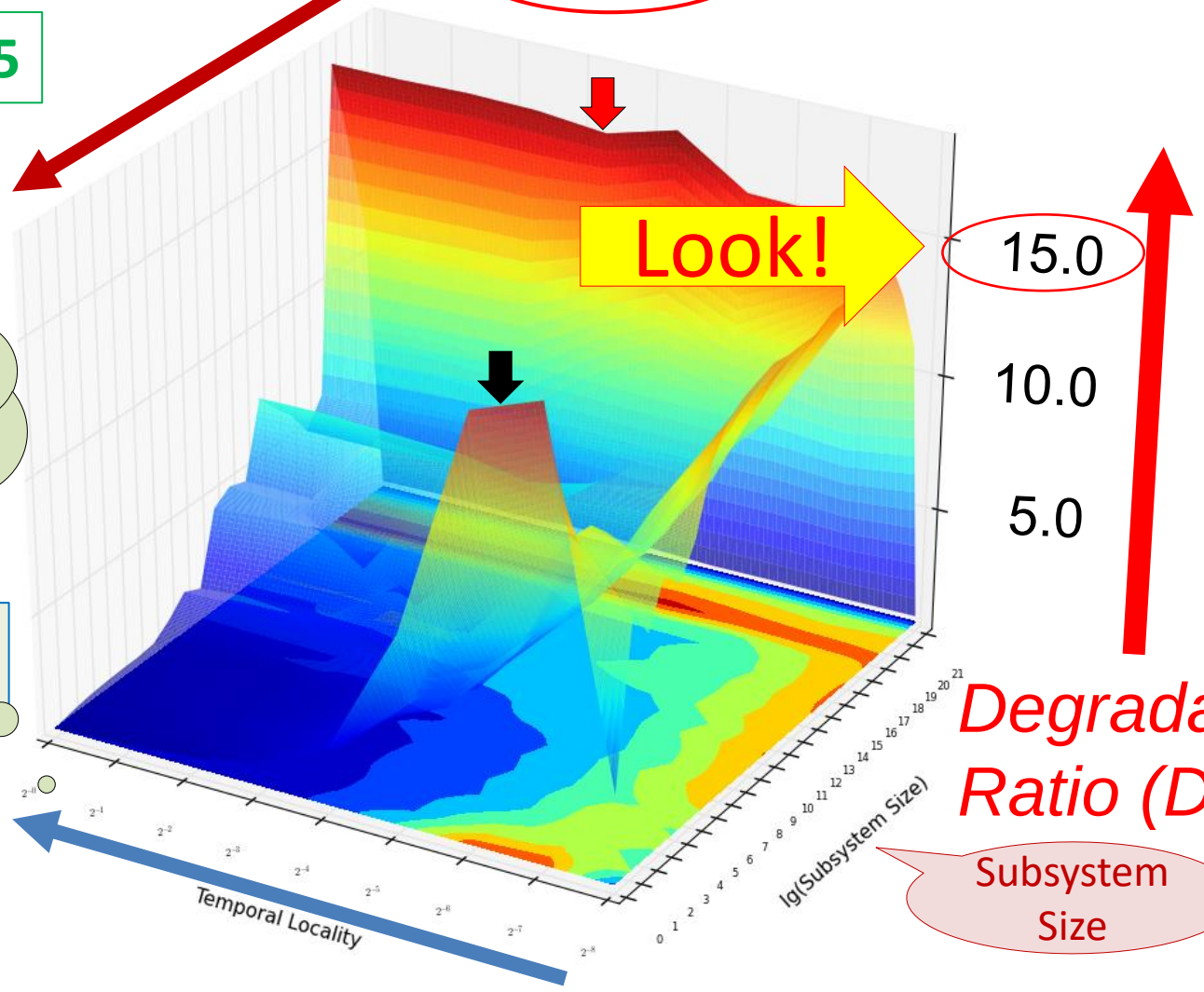
Physical Locality

I · R is fixed.

I ranges from [2^8 .. 2^0].

Max *Temporal Locality* = 256

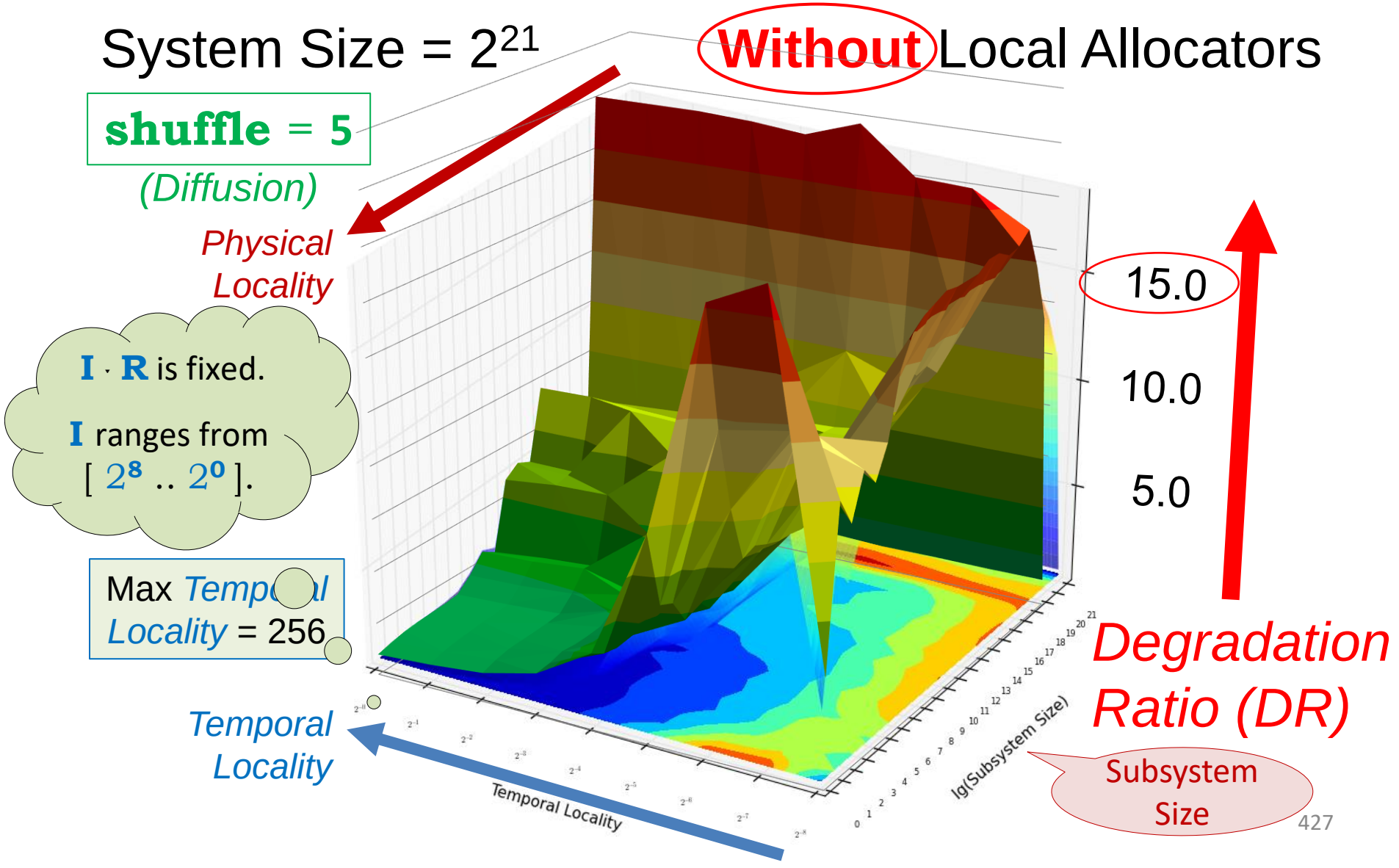
Temporal Locality



Degradation Ratio (DR)

Subsystem Size

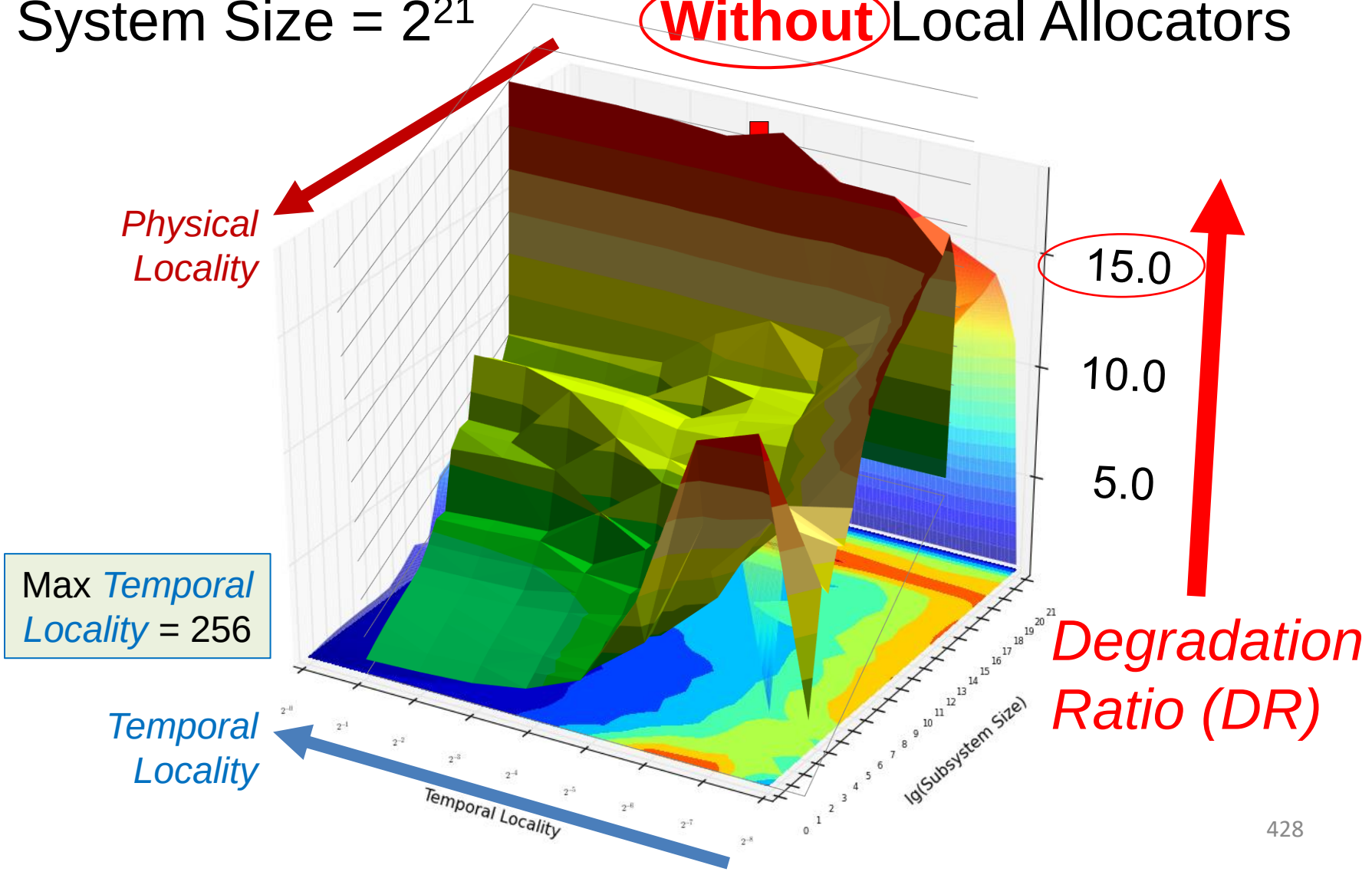
Benchmark II: LOCALITY



Benchmark II: LOCALITY

System Size = 2^{21}

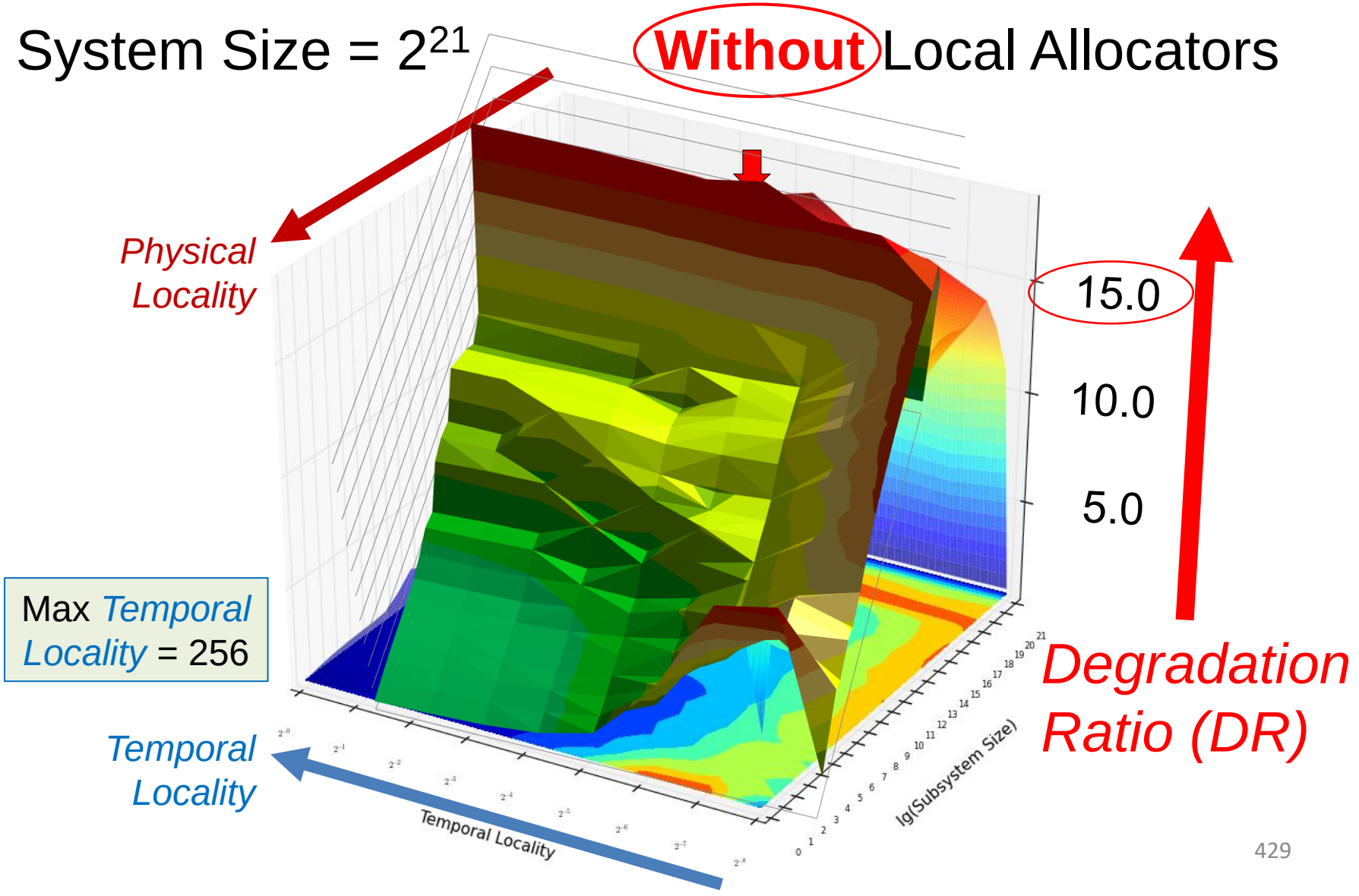
Without Local Allocators



Benchmark II: LOCALITY

System Size = 2^{21}

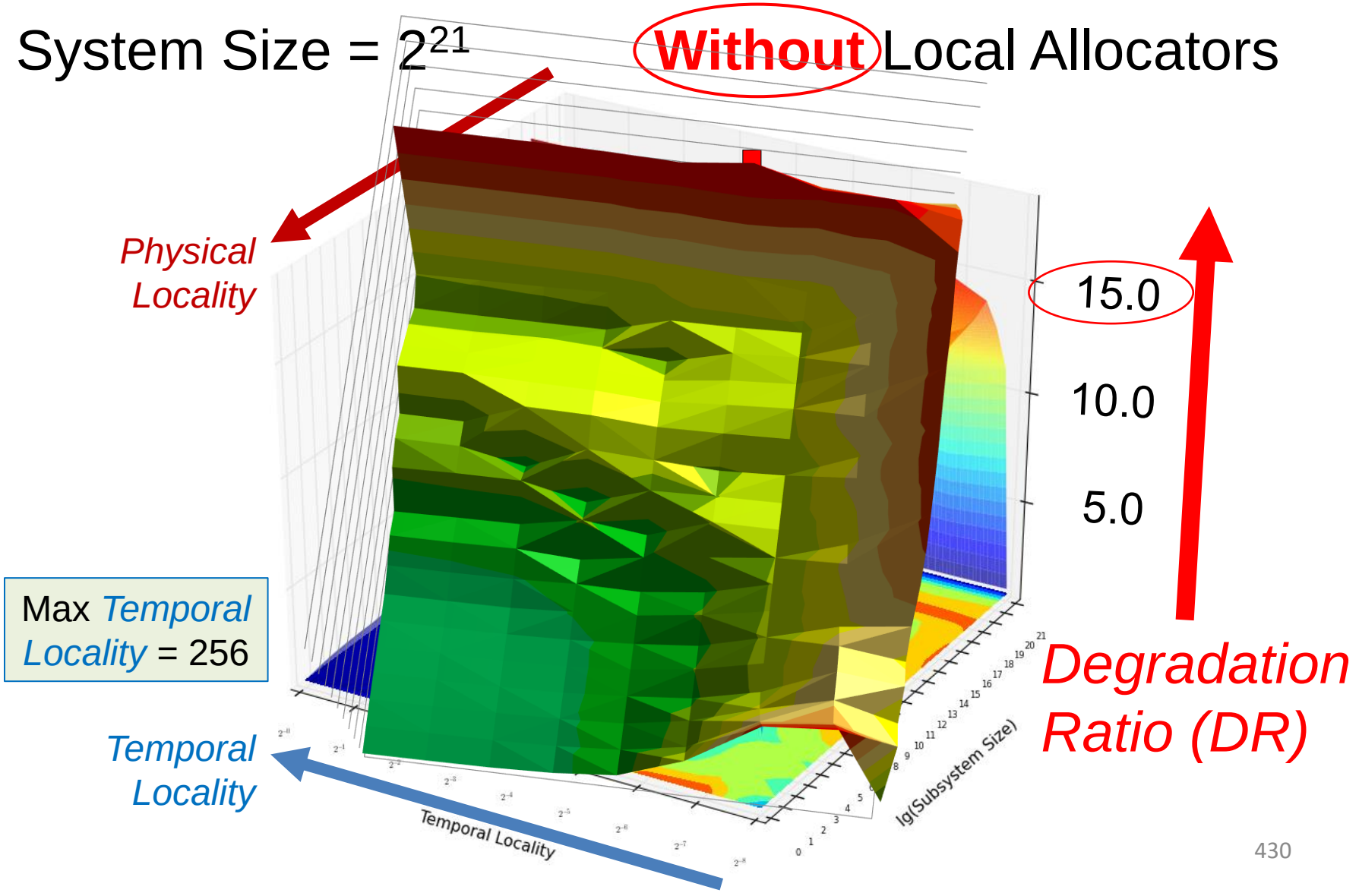
Without Local Allocators



Benchmark II: LOCALITY

System Size = 2^{21}

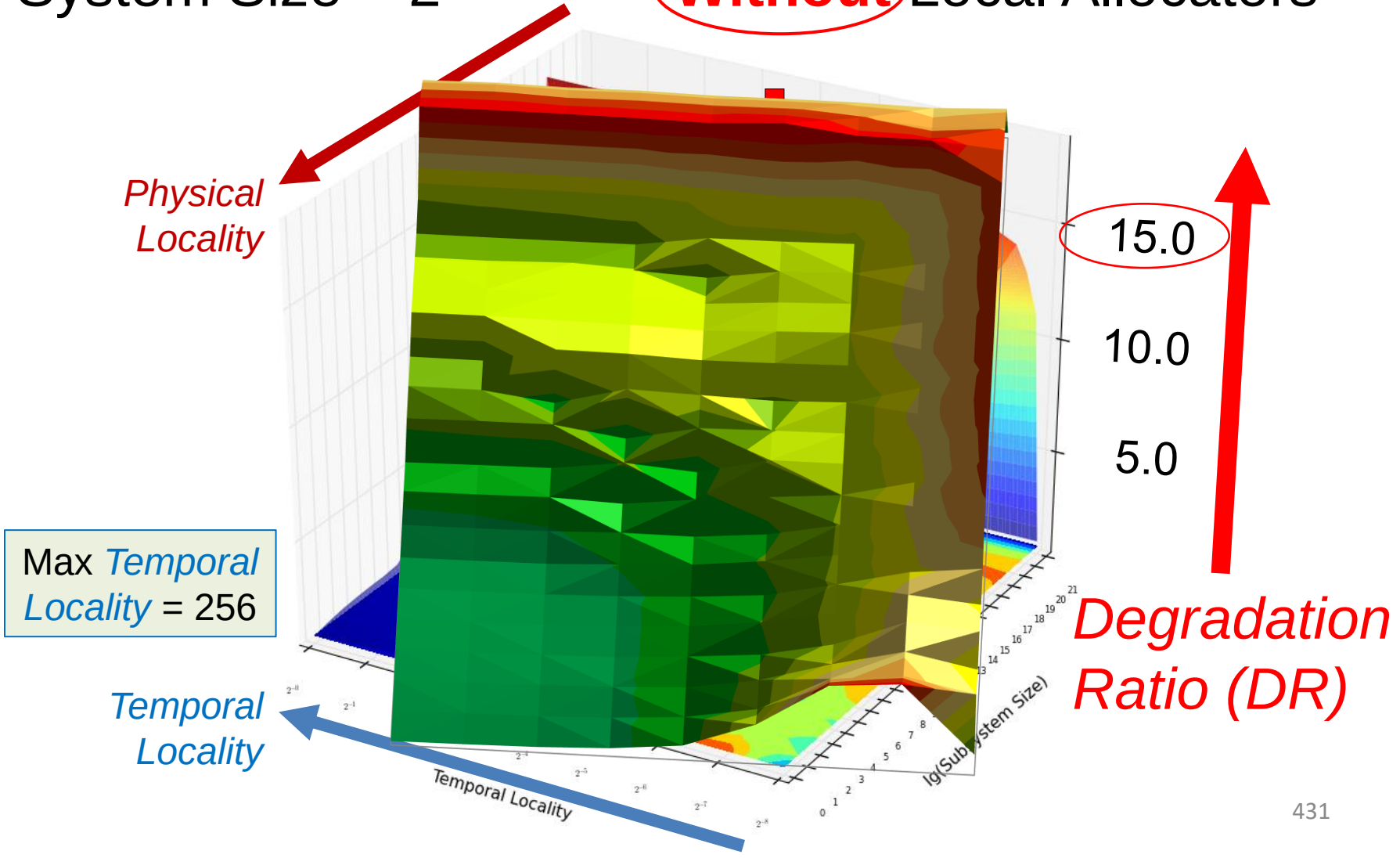
Without Local Allocators



Benchmark II: LOCALITY

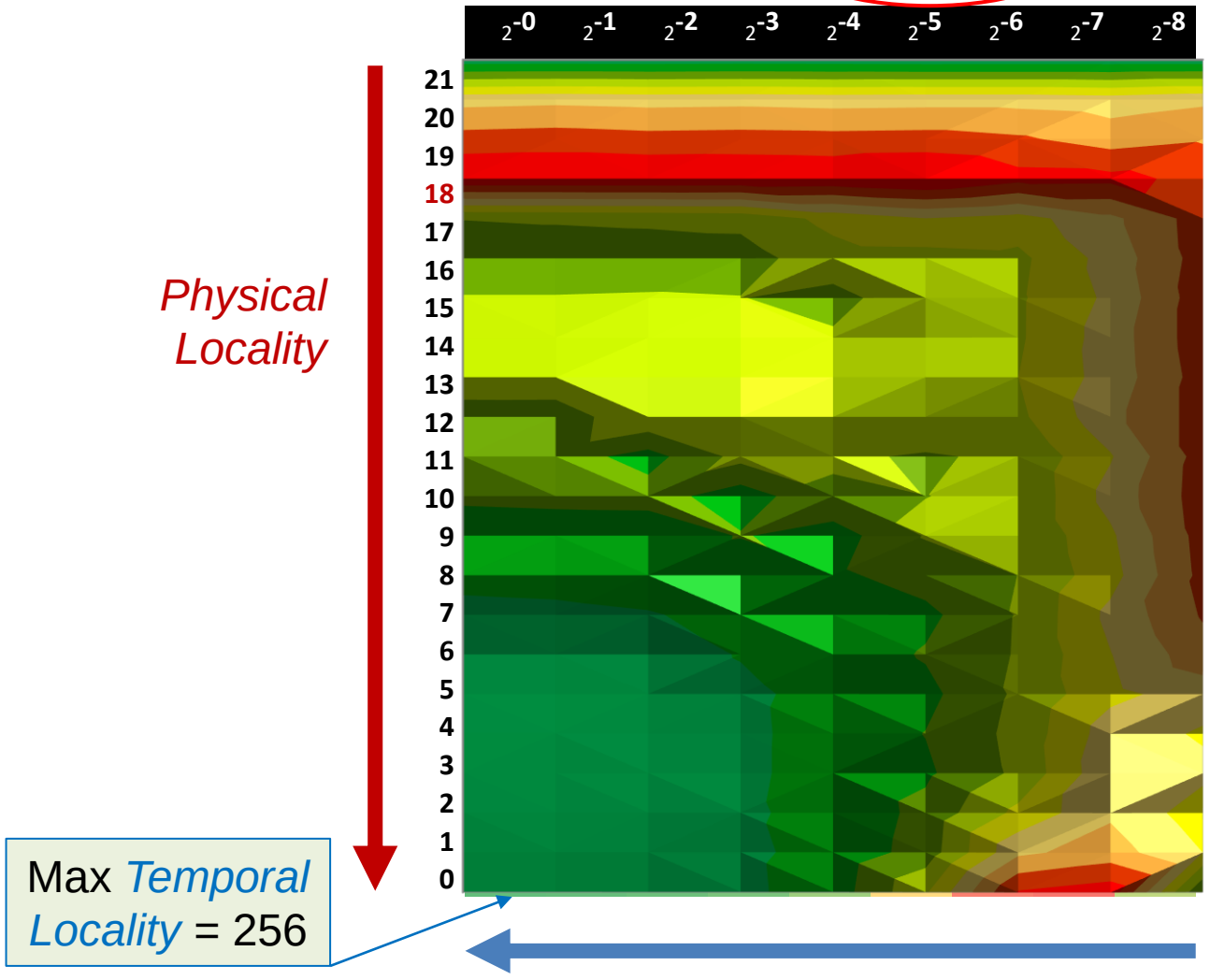
System Size = 2^{21}

Without Local Allocators



Benchmark II: LOCALITY

System Size = 2^{21} **Without** Local Allocators



Degradation Ratio (DR)

Benchmark II: LOCALITY

System Size = 2^{21} **Without** Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
21	1.00	1.06	1.01	1.09	1.04	0.96	0.98	0.97	0.99
20	11.30	11.50	11.30	11.40	11.20	11.30	11.30	11.00	11.60
19	14.80	15.00	14.70	14.80	14.70	14.80	14.30	13.10	13.80
18	18.00	18.00	18.00	17.90	17.70	18.40	16.70	16.60	15.40
17	6.04	6.17	6.30	6.51	8.64	9.95	9.17	11.50	15.00
16	5.07	5.07	5.13	5.19	7.16	7.24	7.52	10.80	14.90
15	6.08	6.08	6.15	6.05	5.37	7.30	7.72	10.40	15.20
14	6.77	6.81	6.78	6.67	6.25	7.23	7.73	10.90	15.20
13	7.55	7.59	7.46	7.36	6.92	7.51	7.99	10.80	14.90
12	4.82	4.79	7.70	7.60	7.08	7.26	7.55	11.40	14.90
11	5.05	4.99	3.21	6.66	6.23	5.85	6.27	9.83	14.90
10	4.65	4.87	4.93	2.92	5.71	5.99	6.15	10.70	15.00
9	2.01	2.23	2.38	4.15	3.03	6.14	6.18	9.67	14.80
8	2.32	2.40	2.60	2.08	3.63	4.86	6.01	9.25	14.60
7	1.68	1.75	1.92	2.36	2.30	3.51	6.12	10.50	14.20
6	1.22	1.31	1.44	2.06	2.76	4.18	6.16	9.93	13.20
5	1.15	1.24	1.39	1.75	2.40	3.45	6.35	9.50	10.90
4	1.13	1.23	1.37	1.72	2.53	4.05	6.60	11.00	9.77
3	1.10	1.19	1.37	1.72	2.55	3.66	6.42	11.60	10.50
2	1.04	1.14	1.36	1.79	2.43	4.61	8.51	11.70	8.91
1	0.93	1.06	1.26	1.66	2.55	4.86	11.60	12.90	10.00
0	0.78	0.90	1.10	1.61	2.88	7.75	16.20	17.20	4.06

Degradation Ratio (DR)

Physical Locality

More, Smaller Subsystems

Subsystems Reaccessed More Frequently

Max *Temporal Locality* = 256

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{21} **Without** Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
21	1.00	1.06	1.01	1.09	1.04	0.96	0.98	0.97	0.99
20	11.30	11.50	11.30	11.40	11.20	11.30	11.30	11.00	11.60
19	14.80	15.00	14.70	14.80	14.70	14.80	14.30	13.10	13.80
18	18.00	18.00	18.00	17.90	17.70	18.40	16.70	16.60	15.40
17	6.04	6.17	6.30	6.51	8.64	9.95	9.17	11.50	15.00
16	5.07	5.07	5.13	5.19	7.16	7.24	7.52	10.80	14.90
15	6.08	6.08	6.15	6.05	5.37	7.30	7.72	10.40	15.20
14	6.77	6.81	6.78	6.67	6.25	7.23	7.73	10.90	15.20
13	7.55	7.59	7.46	7.36	6.92	7.51	7.99	10.80	14.90
12	4.82	4.79	7.70	7.60	7.08	7.26	7.55	11.40	14.90
11	5.05	4.99	3.21	6.66	6.23	5.85	6.27	9.83	14.90
10	4.65	4.87	4.93	2.92	5.71	5.99	6.15	10.70	15.00
9	2.01	2.23	2.38	4.15	3.03	6.14	6.18	9.67	14.80
8	2.32	2.40	2.60	2.08	3.63	4.86	6.01	9.25	14.60
7	1.68	1.75	1.92	2.36	2.30	3.51	6.12	10.50	14.20
6	1.22	1.31	1.44	2.06	2.76	4.18	6.16	9.93	13.20
5	1.15	1.24	1.39	1.75	2.40	3.45	6.35	9.50	10.90
4	1.13	1.23	1.37	1.72	2.53	4.05	6.60	11.00	9.77
3	1.10	1.19	1.37	1.72	2.55	3.66	6.42	11.60	10.50
2	1.04	1.14	1.36	1.79	2.43	4.61	8.51	11.70	8.91
1	0.93	1.06	1.26	1.66	2.55	4.86	mask		10.00
0	0.78	0.90	1.10	1.61	2.88	7.75			4.06

Degradation Ratio (DR)

Physical Locality

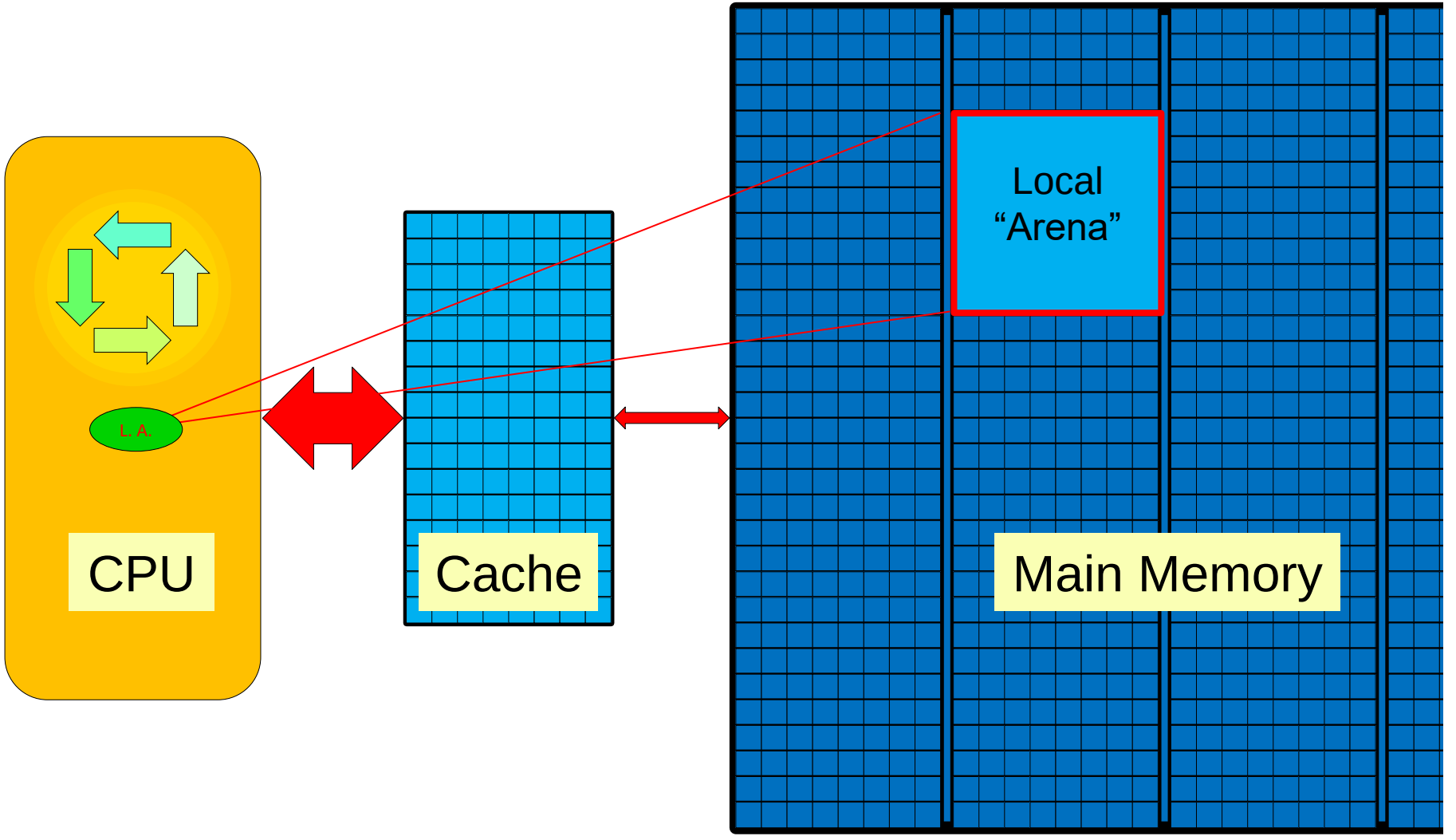
More Smaller Subsystems

Subsystems Reaccessed More Frequently

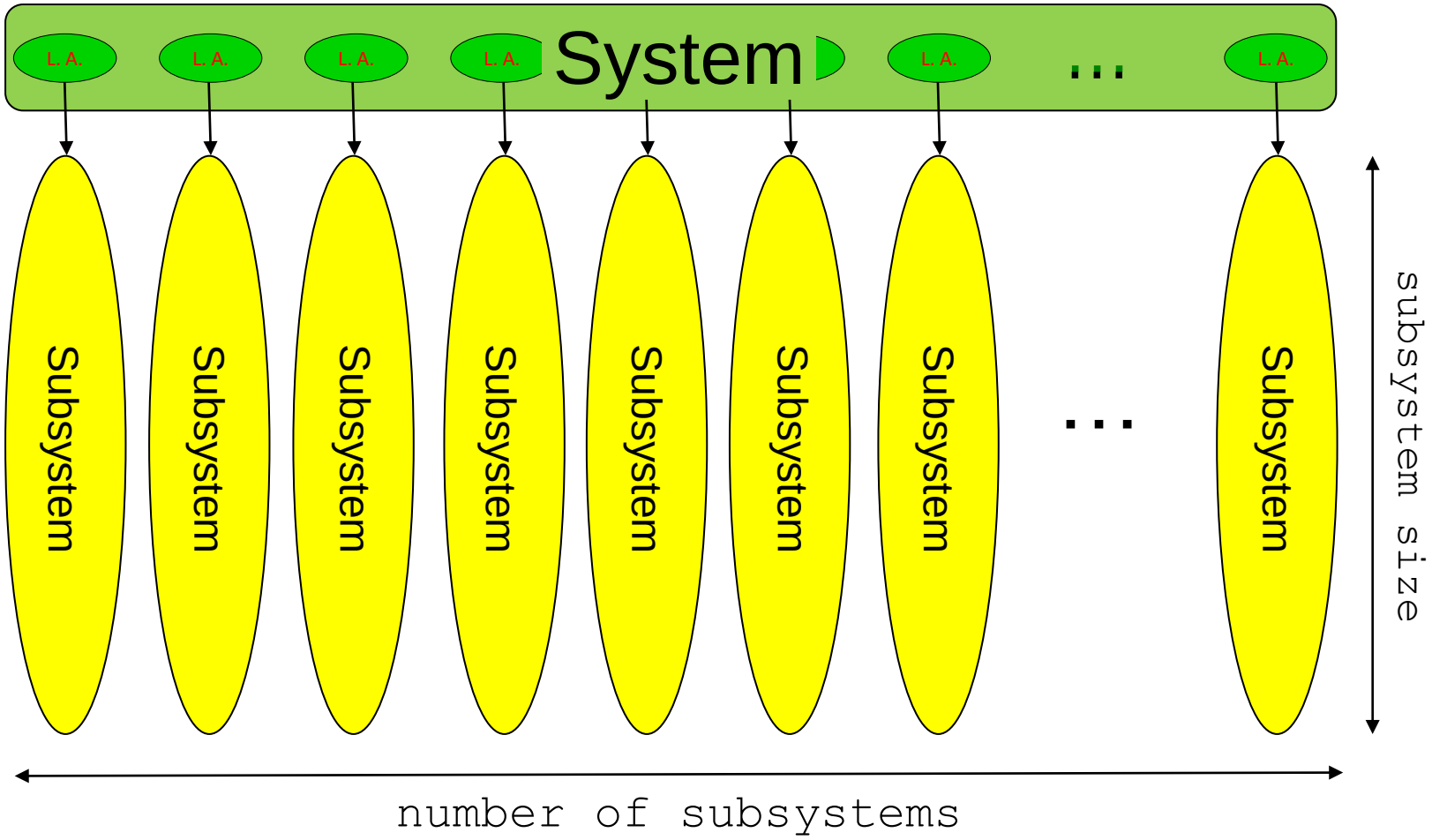
Max *Temporal Locality* = 256

Temporal Locality

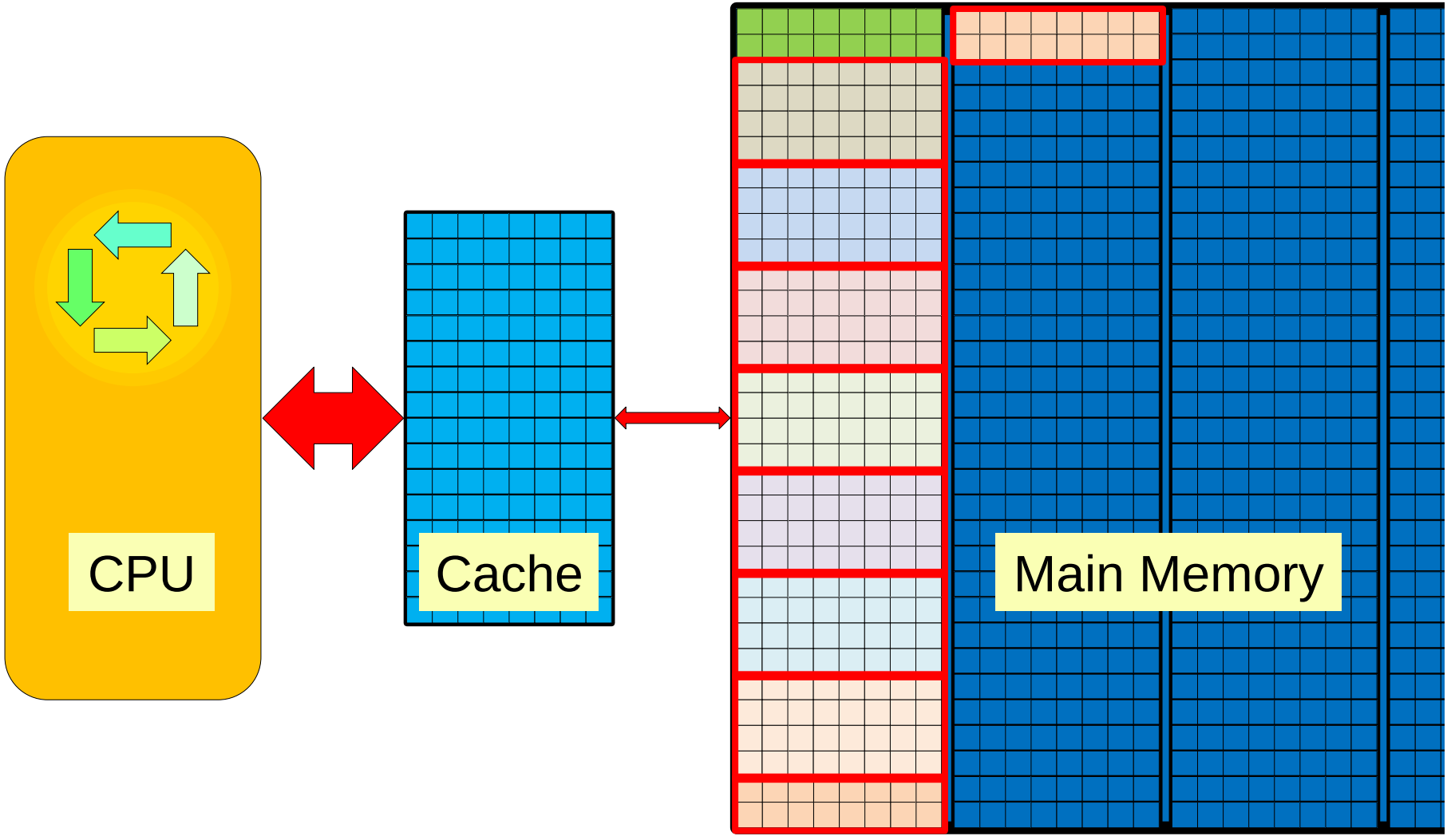
Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY



Benchmark II: LOCALITY

Which local Allocation Strategy should we use?

Benchmark II: LOCALITY

Which local Allocation Strategy should we use?

AS7, AS9, AS11, or AS13

Why?

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

Which local Allocation Strategy should we use?

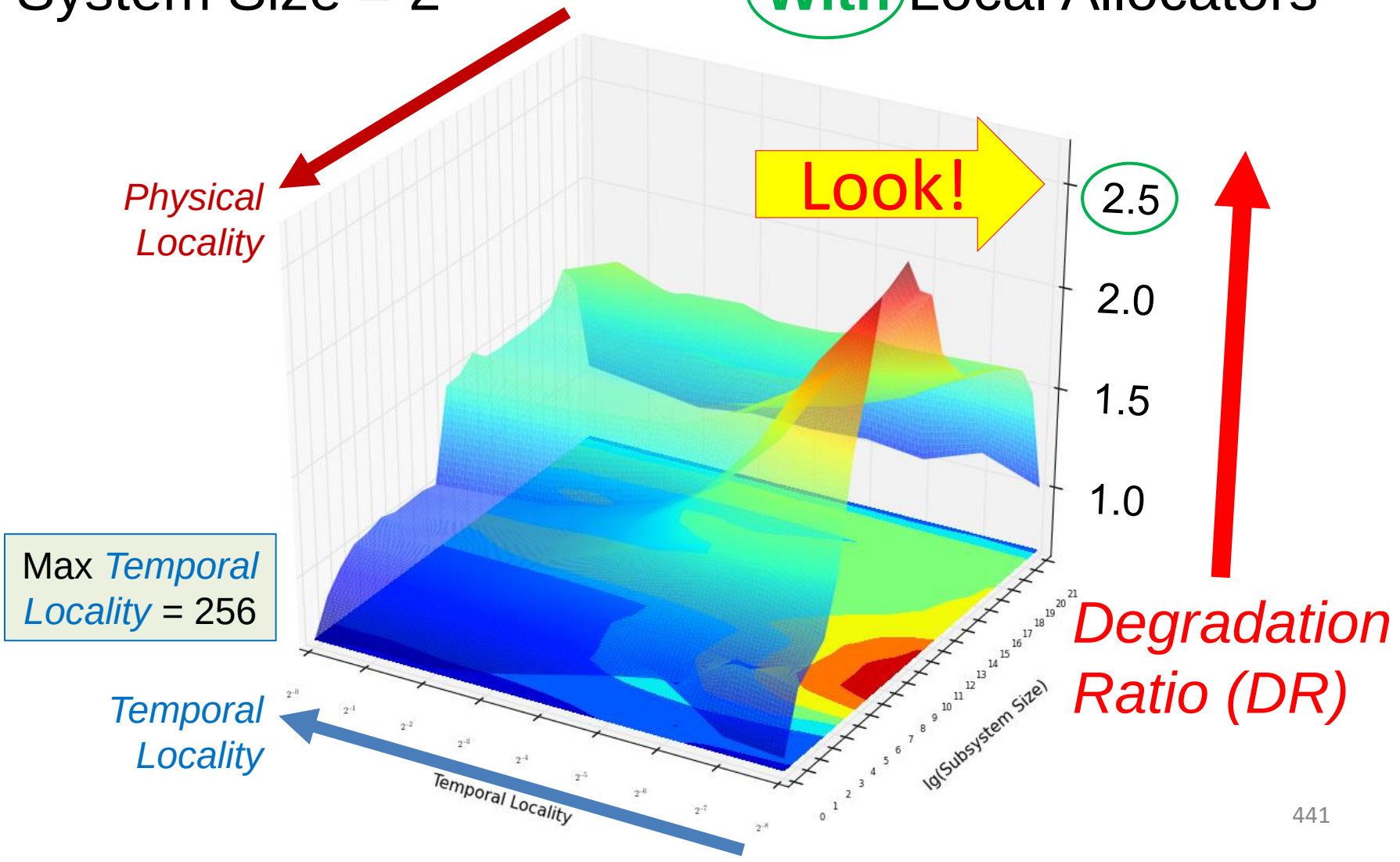
AS7, AS9, AS11, or AS13

Due to shuffle, which causes low **U**TILIZATION (**U**).

Benchmark II: LOCALITY

System Size = 2^{21}

With Local Allocators



Benchmark II: LOCALITY

System Size = 2^{21}

With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
21	1.06	1.02	0.98	1.02	1.02	1.04	1.00	1.11	1.01
20	1.53	1.52	1.62	1.63	1.55	1.63	1.58	1.54	1.53
19	1.65	1.75	1.65	1.65	1.65	1.66	1.66	1.68	1.69
18	1.51	1.49	1.46	1.42	1.43	1.47	1.52	1.66	1.75
17	1.48	1.48	1.48	1.51	1.48	1.54	1.59	1.65	1.81
16	1.48	1.52	1.49	1.50	1.55	1.56	1.56	1.67	1.82
15	1.48	1.48	1.48	1.49	1.51	1.55	1.60	1.69	1.88
14	1.47	1.48	1.48	1.49	1.50	1.54	1.61	1.72	1.90
13	1.48	1.49	1.50	1.50	1.53	1.58	1.66	1.79	1.99
12	1.54	1.51	1.54	1.55	1.57	1.65	1.72	1.91	2.11
11	1.48	1.53	1.53	1.55	1.60	1.65	1.82	2.00	2.42
10	1.47	1.49	1.51	1.54	1.57	1.70	1.88	2.11	2.49
9	1.02	1.04	1.06	1.13	1.22	1.39	1.69	2.14	2.67
8	1.03	1.05	1.08	1.13	1.22	1.42	1.73	2.18	2.62
7	1.03	1.05	1.09	1.14	1.24	1.43	1.75	2.22	2.59
6	1.03	1.06	1.09	1.12	1.24	1.44	1.72	2.08	2.23
5	1.05	1.03	1.08	1.13	1.22	1.38	1.61	1.84	1.94
4	1.02	1.04	1.06	1.11	1.21	1.35	1.53	1.63	1.43
3	1.01	1.01	1.03	1.07	1.15	1.21	1.29	1.25	1.17
2	0.95	0.95	0.97	1.01	1.00	1.04	1.04	0.99	1.10
1	0.85	0.86	0.89	0.90	0.93	0.97	0.89	1.10	1.04
0	0.68	0.71	0.71	0.75	0.83	0.91	0.97	0.77	0.74

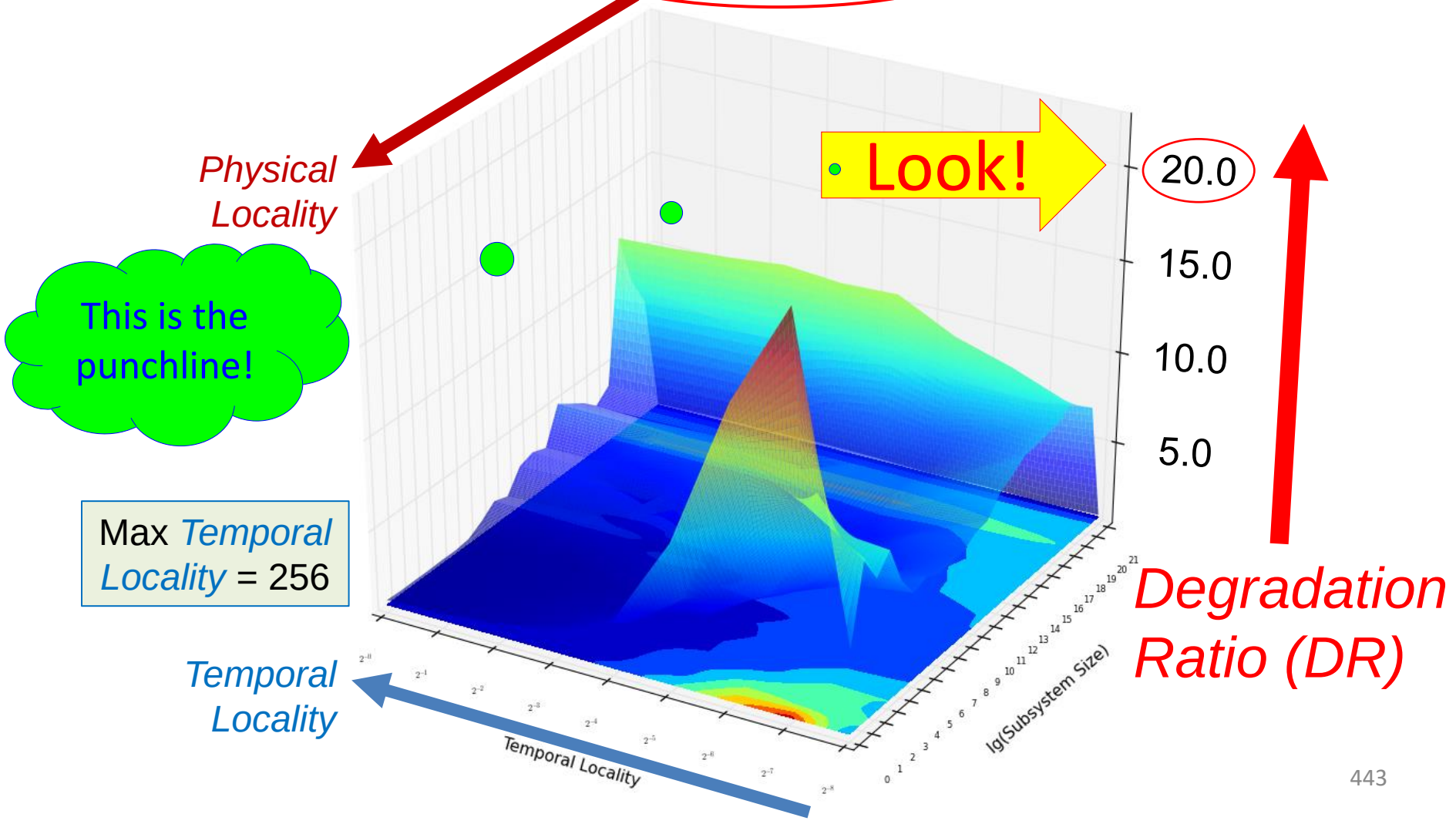
Physical Locality

Degradation Ratio (DR)

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{21} **Without**/With Local Allocators



Benchmark II: LOCALITY

System Size = 2^{21} **Without**/With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
21	0.95	1.03	1.03	1.07	1.02	0.92	0.97	0.87	0.98
20	7.40	7.56	6.97	6.96	7.27	6.94	7.15	7.10	7.56
19	9.02	8.56	8.95	8.99	8.92	8.92	8.61	7.82	8.17
18	11.90	12.10	12.30	12.60	12.40	12.50	10.90	10.00	8.80
17	4.07	4.16	4.26	4.31	5.85	6.48	5.78	6.97	8.27
16	3.43	3.34	3.45	3.45	4.61	4.63	4.82	6.49	8.19
15	4.11	4.11	4.16	4.06	3.56	4.71	4.84	6.14	8.09
14	4.59	4.61	4.59	4.47	4.15	4.69	4.81	6.34	8.02
13	5.11	5.08	4.96	4.90	4.52	4.76	4.82	6.03	7.47
12	3.12	3.16	5.01	4.89	4.51	4.41	4.40	5.96	7.08
11	3.41	3.27	2.09	4.31	3.90	3.54	3.45	4.91	6.15
10	3.16	3.26	3.27	1.89	3.63	3.53	3.28	5.08	6.04
9	1.98	2.15	2.24	3.68	2.48	4.43	3.66	4.51	5.56
8	2.24	2.29	2.42	1.83	2.97	3.43	3.48	4.24	5.59
7	1.64	1.67	1.77	2.07	1.85	2.45	3.49	4.70	5.49
6	1.19	1.24	1.32	1.83	2.23	2.91	3.59	4.78	5.91
5	1.10	1.20	1.29	1.55	1.97	2.50	3.96	5.17	5.61
4	1.11	1.18	1.30	1.55	2.09	3.00	4.31	6.73	6.82
3	1.09	1.18	1.33	1.60	2.23	3.02	4.99	9.30	9.03
2	1.10	1.21	1.40	1.76	2.42	4.43	8.19	11.80	8.09
1	1.09	1.24	1.41	1.85	2.73	5.00	13.00	11.70	9.70
0	1.14	1.26	1.56	2.14	3.47	8.54	16.70	22.40	5.46

Physical Locality

Degradation Ratio (DR)

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{21} **Without**/With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
21	0.95	1.03	1.03	1.07	1.02	0.92	0.97	0.87	0.98
20	7.40	7.56	6.97	6.96	7.27	6.94	7.15	7.10	7.56
19	9.02	8.56	8.95	8.99	8.92	8.92	8.61	7.82	8.17
18	11.90	12.10	12.30	12.60	12.40	12.50	10.90	10.00	8.80
17	4.07	4.16	4.26	4.31	5.85	6.48	5.78	6.97	8.27
16	3.43	3.34	3.45	3.45	4.61	4.63	4.82	6.49	8.19
15	4.11	4.11	4.16	4.06	3.56	4.71	4.84	6.14	8.09
14	4.59	4.61	4.59	4.47	4.15	4.69	4.81	6.34	8.02
13	5.11	5.08	4.96	4.90	4.52	4.76	4.82	6.03	7.47
12	3.12	3.16	5.01	4.89	4.51	4.41	4.40	5.96	7.08
11	3.41	3.27	2.09	4.31	3.90	3.54	3.45	4.91	6.15
10	3.16	3.26	3.27	1.89	3.63	3.53	3.28	5.08	6.04
9	1.98	2.15	2.24	3.68	2.48	4.43	3.66	4.51	5.56
8	2.24	2.29	2.42	1.83	2.97	3.43	3.48	4.24	5.59
7	1.64	1.67	1.77	2.07	1.85	2.45	3.49	4.70	5.49
6	1.19	1.24	1.32	1.83	2.23	2.91	3.59	4.78	5.91
5	1.10	1.20	1.29	1.55	1.97	2.50	3.96	5.17	5.61
4	1.11	1.18	1.30	1.55	2.09	3.00	4.31	6.73	6.82
3	1.09	1.18	1.33	1.60	2.23	3.02	4.99	9.30	9.03
2	1.10	1.21	1.40	1.76	2.42	4.43	8.19	11.80	8.09
1	1.09	1.24	1.41	1.85	2.73	5.00	mask		9.70
0	1.14	1.26	1.56	2.14	3.47	8.54			5.46

Physical Locality

Degradation Ratio (DR)

Order Of Magnitude Speed-Up!

Temporal Locality

Benchmark II: LOCALITY

Repeat with System Size

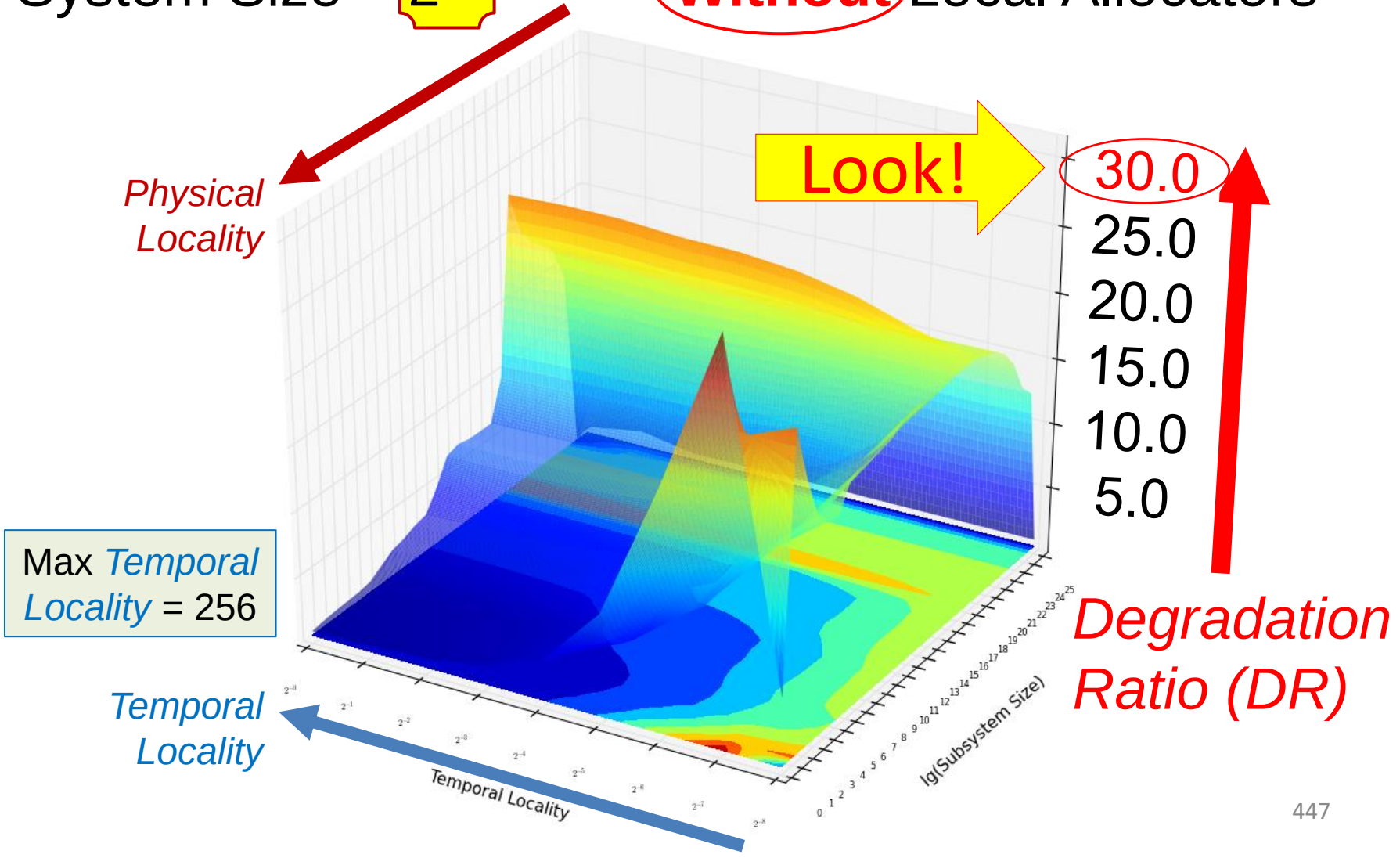
$\mathbf{M} = 2^{25}$

(16 times larger)

Benchmark II: LOCALITY

System Size = 2^{25}

Without Local Allocators



Benchmark II: LOCALITY

System Size = 2^{25} **Without** Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
25	0.97	1.72	0.98	1.02	1.04	1.00	1.00	1.00	1.01
24	0.50	12.71	12.81	13.02	13.08	13.02	13.01	12.95	13.11
23	14.20	14.03	14.10	14.07	14.14	14.05	14.07	14.13	14.14
22	16.70	16.71	16.50	16.55	16.49	16.62	16.55	16.56	16.53
21	17.80	17.90	17.87	17.72	17.85	17.83	17.83	17.76	17.89
20	18.60	18.60	18.52	18.54	18.45	18.64	18.43	18.62	18.66
19	20.10	20.16	19.96	19.85	19.80	19.64	19.25	19.12	18.93
18	23.40	23.54	23.51	23.20	23.13	22.72	21.85	20.45	19.34
17	9.81	10.00	10.06	10.29	10.69	11.53	13.01	15.53	19.30
16	6.81	6.87	6.98	7.21	7.70	8.64	10.41	13.75	19.33
15	6.80	6.88	7.00	7.17	7.66	8.63	10.38	13.66	19.32
14	6.82	6.86	7.00	7.20	7.66	8.56	10.39	13.66	19.18
13	7.03	7.08	7.19	7.39	7.85	8.79	10.56	13.77	19.15
12	6.72	6.75	6.90	7.12	7.62	8.52	10.21	13.62	19.30
11	4.86	4.92	5.07	5.35	5.88	6.92	9.01	13.00	19.16
10	3.36	3.49	3.71	4.07	4.69	5.91	8.25	12.32	18.43
9	3.15	3.29	3.51	3.86	4.54	5.87	8.25	12.37	18.28
8	2.76	2.89	3.13	3.54	4.33	5.66	8.12	12.43	18.16
7	2.52	2.66	2.96	3.45	4.25	5.67	8.16	12.65	18.10
6	1.94	2.14	2.49	3.03	3.91	5.45	8.02	12.72	17.64
5	1.34	1.49	1.78	2.33	3.24	4.79	7.54	12.41	15.84
4	1.17	1.28	1.51	1.96	2.83	4.39	7.39	13.22	16.25
3	1.12	1.24	1.45	1.89	2.73	4.30	7.85	14.74	17.32
2	1.06	1.19	1.43	1.90	2.71	4.70	9.98	18.32	20.54
1	0.97	1.11	1.36	1.78	2.91	5.68	13.74	22.91	24.73
0	0.82	0.97	1.22	1.88	3.50	8.30	18.92	30.99	5.68

Degradation Ratio (DR)

Physical Locality

Max *Temporal Locality* = 256

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{25} **Without** Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
25	0.97	1.72	0.98	1.02	1.04	1.00	1.00	1.00	1.01
24	0.50	12.71	12.81	13.02	13.08	13.02	13.01	12.95	13.11
23	14.20	14.03	14.10	14.07	14.14	14.05	14.07	14.13	14.14
22	16.70	16.71	16.50	16.55	16.49	16.62	16.55	16.56	16.53
21	17.80	17.90	17.87	17.72	17.85	17.83	17.83	17.76	17.89
20	18.60	18.60	18.52	18.54	18.45	18.64	18.43	18.62	18.66
19	20.10	20.16	19.96	19.85	19.80	19.64	19.25	19.12	18.93
18	23.40	23.54	23.51	23.20	23.13	22.72	21.85	20.45	19.34
17	9.81	10.00	10.06	10.29	10.69	11.53	13.01	15.53	19.30
16	6.81	6.87	6.98	7.21	7.70	8.64	10.41	13.75	19.33
15	6.80	6.88	7.00	7.17	7.66	8.63	10.38	13.66	19.32
14	6.82	6.86	7.00	7.20	7.66	8.56	10.39	13.66	19.18
13	7.03	7.08	7.19	7.39	7.85	8.79	10.56	13.77	19.15
12	6.72	6.75	6.90	7.12	7.62	8.52	10.21	13.62	19.30
11	4.86	4.92	5.07	5.35	5.88	6.92	9.01	13.00	19.16
10	3.36	3.49	3.71	4.07	4.69	5.91	8.25	12.32	18.43
9	3.15	3.29	3.51	3.86	4.54	5.87	8.25	12.37	18.28
8	2.76	2.89	3.13	3.54	4.33	5.66	8.12	12.43	18.16
7	2.52	2.66	2.96	3.45	4.25	5.67	8.16	12.65	18.10
6	1.94	2.14	2.49	3.03	3.91	5.45	8.02	12.72	17.64
5	1.34	1.49	1.78	2.33	3.24	4.79	7.54	12.41	15.84
4	1.17	1.28	1.51	1.96	2.83	4.39	7.39	13.22	16.25
3	1.12	1.24	1.45	1.89	2.73	4.30	7.85	14.74	17.32
2	1.06	1.19	1.43	1.90	2.71	4.70	9.98	18.32	20.54
1	0.97	1.11	1.36	1.78	2.91	5.68	mask		24.73
0	0.82	0.97	1.22	1.88	3.50	8.30			5.68

Physical Locality

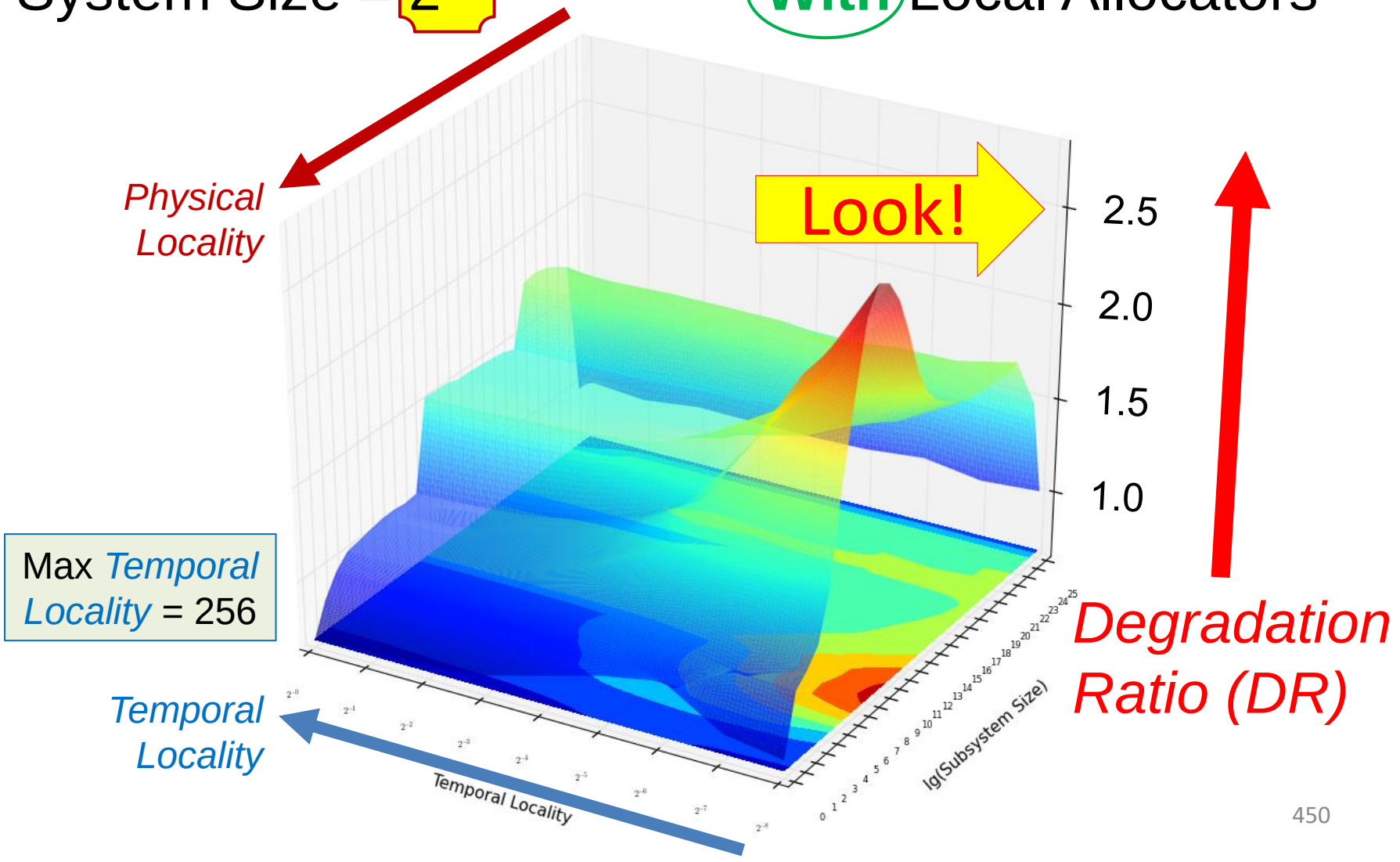
Degradation Ratio (DR)

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{25}

With Local Allocators



Benchmark II: LOCALITY

System Size = 2^{25}

With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
25	1.00	0.97	1.01	1.00	1.00	1.01	1.04	0.99	1.01
24	1.00	1.54	1.57	1.55	1.55	1.53	1.55	1.56	1.53
23	1.71	1.67	1.70	1.69	1.68	1.68	1.68	1.69	1.67
22	1.75	1.75	1.76	1.76	1.75	1.72	1.76	1.76	1.83
21	1.79	1.78	1.78	1.80	1.79	1.74	1.80	1.80	1.80
20	1.80	1.80	1.80	1.81	1.81	1.82	1.81	1.81	1.82
19	1.79	1.78	1.79	1.80	1.79	1.80	1.80	1.82	1.82
18	1.47	1.47	1.47	1.49	1.50	1.53	1.58	1.67	1.83
17	1.49	1.49	1.49	1.50	1.51	1.54	1.59	1.67	1.84
16	1.50	1.50	1.53	1.51	1.53	1.55	1.61	1.70	1.88
15	1.51	1.51	1.51	1.52	1.53	1.56	1.63	1.74	1.92
14	1.51	1.51	1.52	1.52	1.54	1.58	1.65	1.78	1.97
13	1.51	1.52	1.52	1.53	1.55	1.60	1.67	1.82	2.05
12	1.53	1.54	1.54	1.56	1.59	1.64	1.74	1.92	2.20
11	1.54	1.54	1.55	1.57	1.60	1.67	1.84	2.08	2.43
10	1.54	1.55	1.56	1.58	1.61	1.74	1.93	2.25	2.63
9	1.07	1.08	1.11	1.16	1.26	1.44	1.87	2.22	2.76
8	1.06	1.10	1.12	1.18	1.27	1.47	1.85	2.29	2.80
7	1.07	1.06	1.12	1.17	1.28	1.48	1.82	2.32	2.67
6	1.07	1.08	1.10	1.16	1.26	1.46	1.75	2.13	2.31
5	1.05	1.06	1.09	1.14	1.23	1.40	1.62	1.86	1.93
4	1.04	1.05	1.07	1.12	1.22	1.38	1.54	1.65	1.44
3	1.02	1.03	1.05	1.08	1.15	1.23	1.30	1.29	1.20
2	0.96	0.97	0.99	1.02	1.02	1.04	1.05	1.00	1.12
1	0.85	0.86	0.89	0.90	0.94	0.99	0.90	1.08	1.06
0	0.69	0.70	0.72	0.75	0.84	0.92	0.98	0.79	0.74

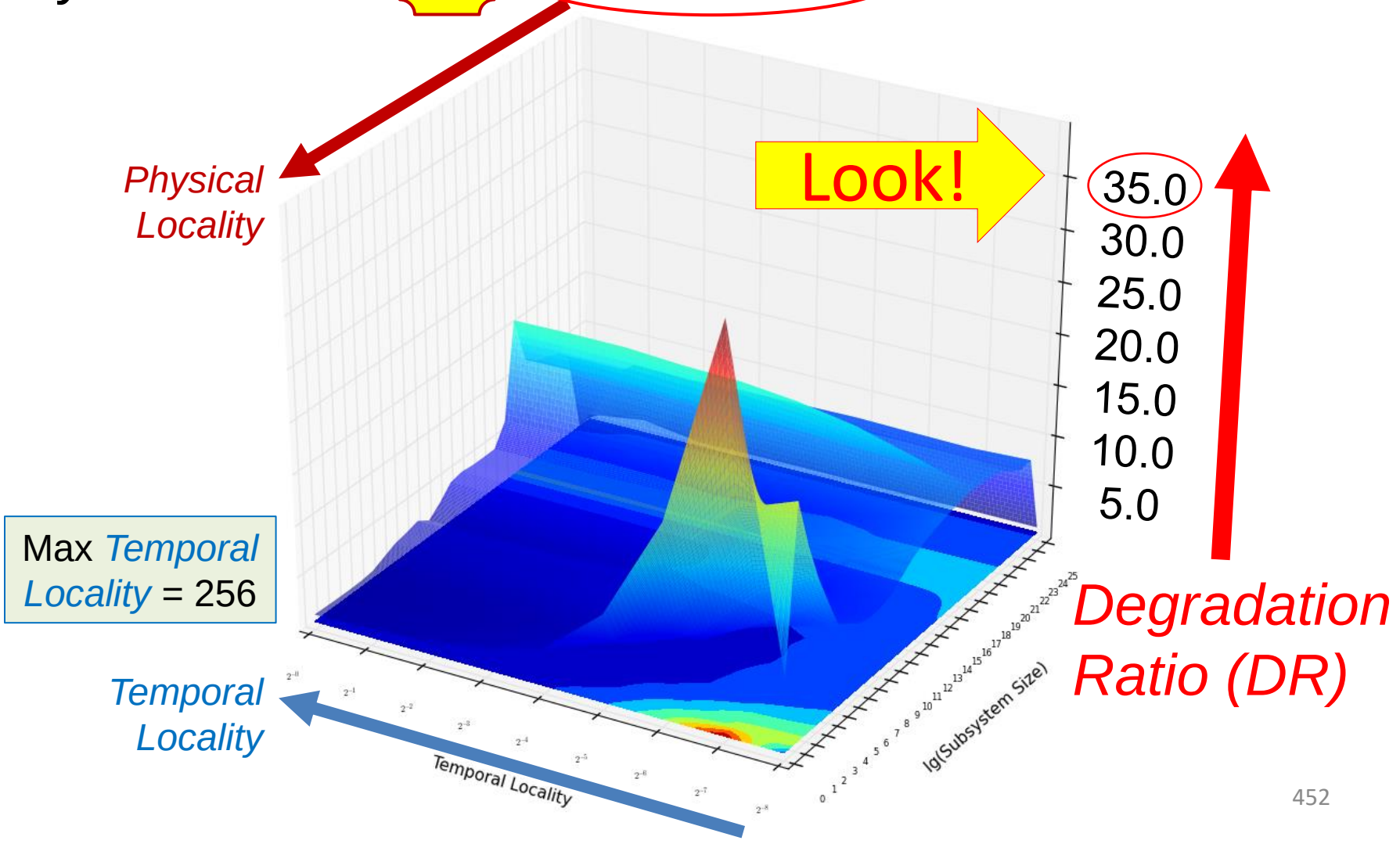
Degradation Ratio (DR)

Physical Locality

Temporal Locality

Benchmark II: LOCALITY

System Size = 2^{25} **Without**/With Local Allocators



Benchmark II: LOCALITY

System Size = 2^{25} **Without**/With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
25	0.97	1.77	0.97	1.02	1.04	0.99	0.96	1.01	1.00
24	0.50	8.26	8.17	8.38	8.43	8.52	8.40	8.30	8.57
23	8.30	8.41	8.30	8.35	8.40	8.36	8.36	8.37	8.44
22	9.53	9.57	9.40	9.43	9.41	9.65	9.42	9.41	9.04
21	9.99	10.00	10.00	9.87	9.98	10.20	9.90	9.88	9.93
20	10.40	10.30	10.30	10.20	10.20	10.20	10.20	10.30	10.30
19	11.20	11.30	11.20	11.00	11.00	10.90	10.70	10.50	10.40
18	15.90	16.00	16.00	15.60	15.40	14.80	13.90	12.30	10.60
17	6.58	6.70	6.73	6.87	7.06	7.50	8.19	9.30	10.50
16	4.53	4.57	4.56	4.76	5.04	5.56	6.47	8.11	10.30
15	4.51	4.56	4.62	4.73	4.99	5.52	6.38	7.86	10.10
14	4.51	4.54	4.62	4.72	4.98	5.43	6.28	7.70	9.75
13	4.64	4.67	4.73	4.82	5.07	5.49	6.31	7.56	9.34
12	4.38	4.38	4.48	4.58	4.80	5.18	5.86	7.10	8.78
11	3.16	3.19	3.27	3.42	3.67	4.15	4.90	6.25	7.88
10	2.18	2.25	2.37	2.58	2.91	3.39	4.28	5.48	7.00
9	2.95	3.04	3.17	3.32	3.59	4.08	4.42	5.57	6.63
8	2.60	2.62	2.80	3.00	3.40	3.84	4.40	5.42	6.50
7	2.36	2.51	2.65	2.95	3.31	3.83	4.49	5.45	6.79
6	1.82	1.99	2.26	2.62	3.10	3.74	4.59	5.97	7.62
5	1.27	1.40	1.64	2.05	2.64	3.42	4.65	6.67	8.21
4	1.13	1.22	1.42	1.76	2.32	3.18	4.79	8.02	11.30
3	1.09	1.21	1.39	1.75	2.37	3.49	6.03	11.40	14.50
2	1.11	1.22	1.45	1.86	2.66	4.51	9.49	18.30	18.40
1	1.14	1.29	1.54	1.98	3.10	5.76	15.30	21.20	23.40
0	1.19	1.38	1.70	2.50	4.18	8.99	19.30	39.20	7.66

Physical Locality

Degradation Ratio (DR)

Temporal Locality

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

System Size = 2^{25} **Without**/With Local Allocators

	2^{-0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
25	0.97	1.77	0.97	1.02	1.04	0.99	0.96	1.01	1.00
24	0.50	8.26	8.17	8.38	8.43	8.52	8.40	8.30	8.57
23	8.30	8.41	8.30	8.35	8.40	8.36	8.36	8.37	8.44
22	9.53	9.57	9.40	9.43	9.41	9.65	9.42	9.41	9.04
21	9.99	10.00	10.00	9.87	9.98	10.20	9.90	9.88	9.93
20	10.40	10.30	10.30	10.20	10.20	10.20	10.20	10.30	10.30
19	11.20	11.30	11.20	11.00	11.00	10.90	10.70	10.50	10.40
18	15.9	16.00	16.00	15.60	15.40	14.80	13.90	12.30	10.60
17	6.58	6.70	6.73	6.87	7.06	7.50	8.19	9.30	10.50
16	4.53	4.57	4.56	4.76	5	5.56	6.47	8.11	10.30
15	4.51	4.56	4.62	4.73	4.99	5.52	6.38	7.86	10.10
14	4.51	4.54	4.62	4.72	4.98	5.43	6.28		9.75
13	4.64	4.67	4.73	4.82	5.07	5.49	6.31		9.34
12	4.38	4.38	4.48	4.58	4.80	5.18	5.86	7.10	8.78
11	3.16	3.19	3.27	3.42	3.67	4.15	4.90	6.25	7.88
10	2.18	2.25	2.37	2.58	2.91	3.39	4.28	5.48	7.00
9	2.95	3.04	3.17	3.32	3.59	4.08	4.42	5.57	6.63
8	2.60	2.62	2.80	3.00	3.40	3.84	4.40	5.42	6.50
7	2.36	2.51	2.65	2.95	3.31	3.83	4.49	5.45	6.79
6	1.82	1.99	2.26	2.62	3.10	3.74	4.59	5.97	7.62
5	1.27	1.40	1.64	2.05	2.64	3.42	4.65	6.67	8.21
4	1.13	1.22	1.42	1.76	2.32	3.18	4.79	8.02	11.30
3	1.09	1.21	1.39	1.75	2.37	3.49	6.03	11.40	14.50
2	1.11	1.22	1.45	1.86	2.66	4.51	9.49	18.30	18.40
1	1.14	1.29	1.54	1.98	3.10	5.76	mask		23.40
0	1.19	1.38	1.70	2.50	4.18	8.99			7.66

Physical Locality

Degradation Ratio (DR)

16 X Speed-Up!

Temporal Locality

3. Analyzing the Benchmark Data

Benchmark II: LOCALITY

Dimensional Characterization

Allocation DENSITY (**D**):	NIL
"Chunk"-Size VARIATION (**V**):	NIL
Access LOCALITY (**L**):	?
Memory UTILIZATION (**U**):	MEDIUM (1/6 due to shuffling 5 times)
Allocator CONTENTION (**C**):	0

Benchmark II: LOCALITY

Take-Away Tips

- For long-running systems, the speed of *allocation/deallocation* operations themselves may be entirely irrelevant to overall runtime performance.
- Partitioning memory corresponding to subsystems having *physical* and *temporal locality* can nonetheless make an enormous difference in overall runtime performance.

Benchmark II: LOCALITY

Questions and/or Discussion?

Benchmark III: UTILIZATION

I. **Short Running:** Build Up, Use, Tear Down
   - Allocation DENSITY and VARIATION in Allocated Sizes

II. **Long Running:** Time-Multiplexed Subsystems
   - Access LOCALITY – both *Physica*l and *Temporal*

III. **Short Running:** Varying Memory Reusability
   - Memory UTILIZATION

IV. **Multithreaded:** Varying Numbers of Threads
   - Allocator CONTENTION

3. Analyzing the Benchmark Data

Benchmark III: **U**TILIZATION



Benchmark III: UTILIZATION

Considerations:

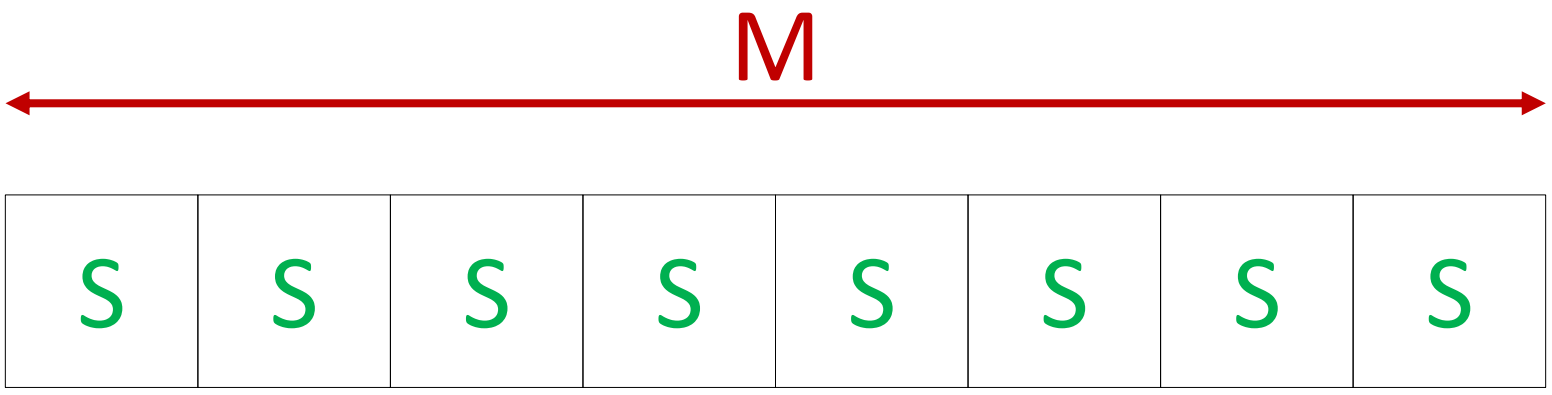
- Investigate memory UTILIZATION (**U**).
  - Focus on allocation/deallocation costs themselves.
- Access LOCALITY (**L**) should *NOT* dominate results.
  - Write to <u>just</u> first byte of each newly allocated element.
- Analyze sub-dimension **S** ("chunk" size) of **U** = **M**/**T**:
  - **T** is the total memory allocated.
  - **M** is the maximum concurrently active memory.
  - **S** is the (atomic) memory "chunk" size.

Benchmark III: UTILIZATION

Plan:

- Allocate “chunks” of memory of size ***S*** until memory of size ***M*** is concurrently allocated.

Benchmark III: UTILIZATION



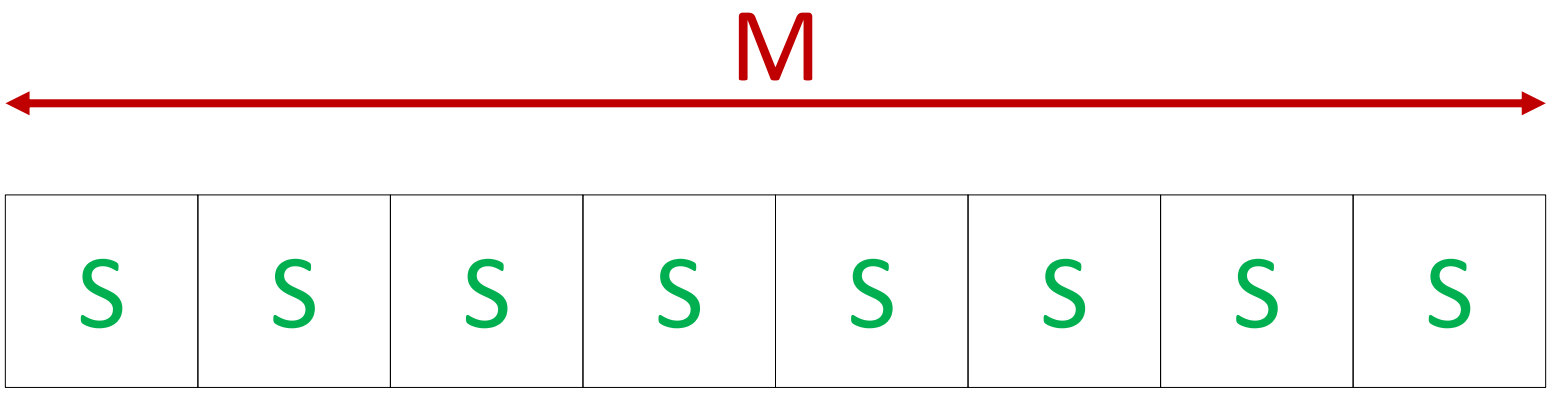
$T = 8S$

Benchmark III: UTILIZATION

Plan:

- Allocate “chunks” of memory of size ***S*** until memory of size ***M*** is concurrently allocated.
- Deallocate **L**east **R**ecently **U**sed chunk and reallocate it until the total memory of size ***T*** has been allocated.

Benchmark III: UTILIZATION



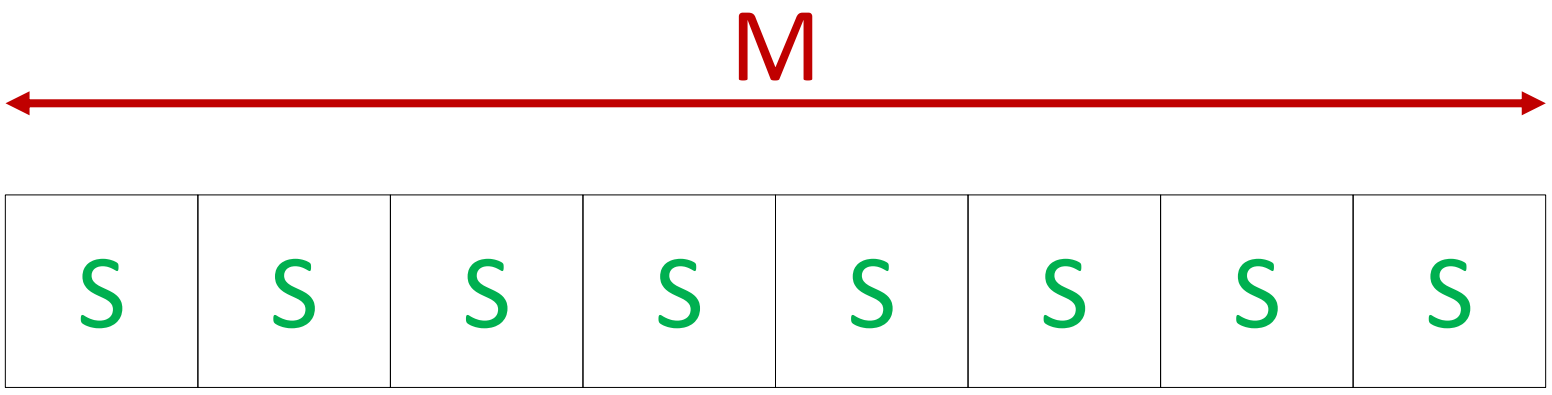
$$T = 65536S$$

Benchmark III: UTILIZATION

Plan:

- Allocate “chunks” of memory of size ***S*** until memory of size ***M*** is concurrently allocated.
- Deallocate Least Recently Used chunk and reallocate it until the total memory of size ***T*** has been allocated.
- Deallocate the remaining allocated memory (of size ***M***), one “chunk” (of size ***S***) at a time.

Benchmark III: UTILIZATION



$$T = 65536S$$

Benchmark III: UTILIZATION

Plan:

- Allocate "chunks" of memory of size ***S*** until memory of size ***M*** is concurrently allocated.
- Deallocate **L**east **R**ecently **U**sed chunk and reallocate it until the total memory of size ***T*** has been allocated.
- Deallocate the remaining allocated memory (of size ***M***), one "chunk" (of size ***S***) at a time.
- The results of this experiment are absolute run times.
  - The entries in each row, other than AS1, are relative to AS1.

Benchmark III: UTILIZATION

Three more considerations:

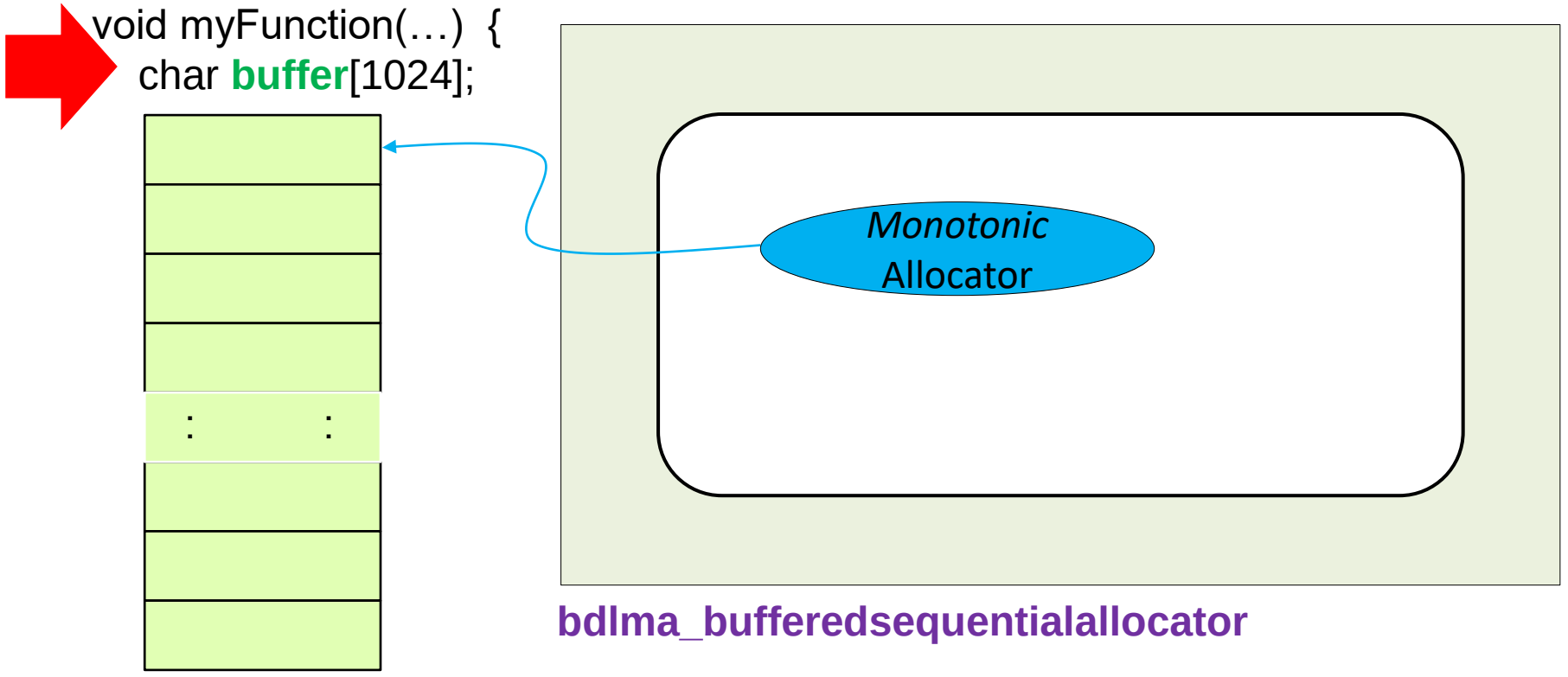
- We chose to omit **AS**s that "wink out" memory.
  - I.e., we omitted **AS4**, **AS6**, **AS8**, **AS10**, **AS12**, and **AS14**.
  - Virtually all of the memory deallocated individually.
  - (No mathematical possibility of any noticeable effect.)

Benchmark III: UTILIZATION

Three more considerations:

- We chose to omit **AS**s that "wink out" memory.
  - I.e., we omitted **AS4**, **AS6**, **AS8**, **AS10**, **AS12**, and **AS14**.
  - Virtually all of the memory deallocated individually.
  - (No mathematical possibility of any noticeable effect.)
- We anticipated trouble with *any* use of **monotonic**.
  - Total memory allocated (**T**) is large (2^{35} bytes).

Benchmark III: UTILIZATION



Note that deallocate is a No-Op!

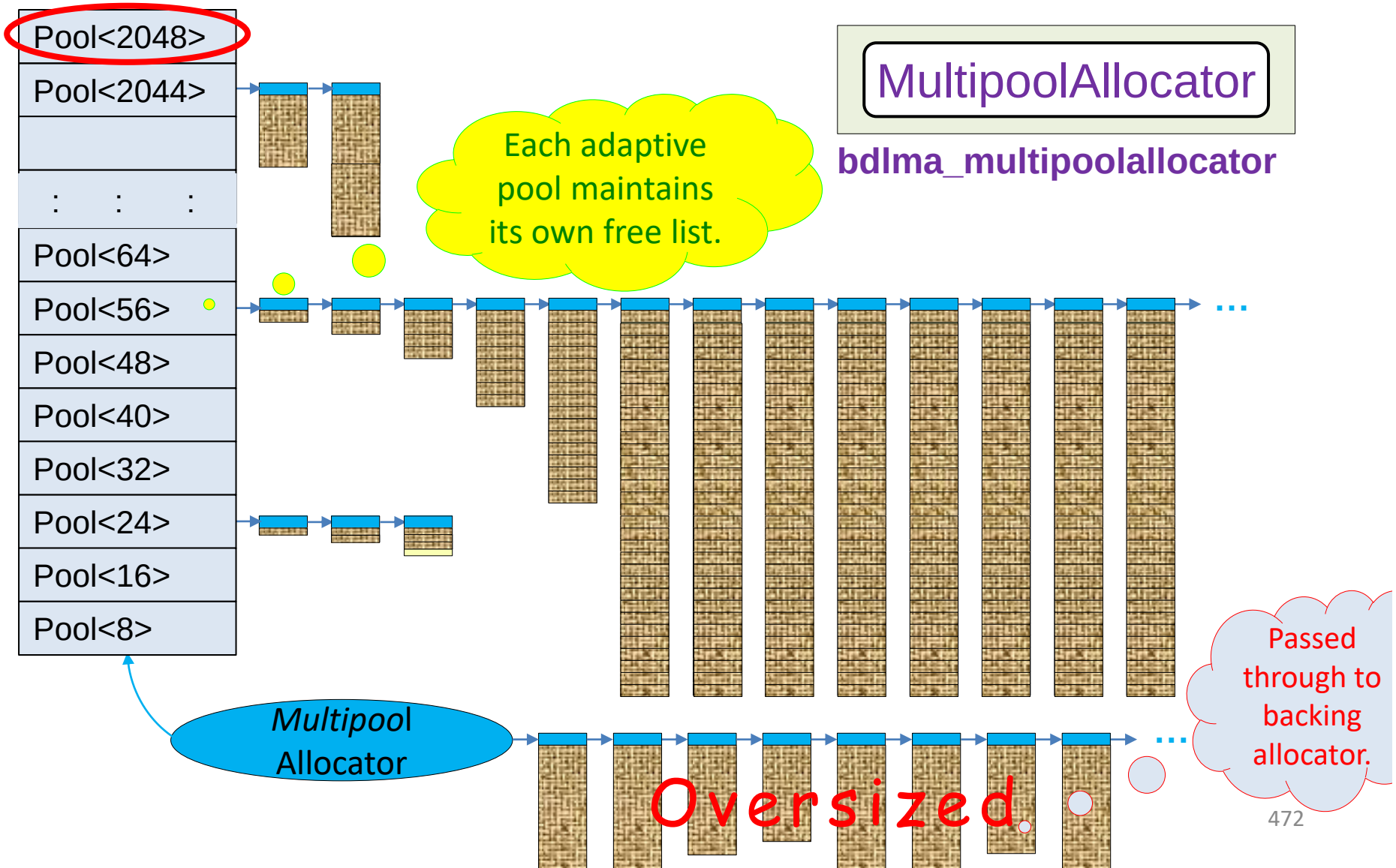
1000

Benchmark III: UTILIZATION

Three more considerations:

- We chose to omit **AS**s that "wink out" memory.
  - I.e., we omitted **AS4**, **AS6**, **AS8**, **AS10**, **AS12**, and **AS14**.
  - Virtually all of the memory deallocated individually.
  - (No mathematical possibility of any noticeable effect.)
- We anticipated trouble with *any* use of **monotonic**.
  - Total memory allocated (**T**) is large (2^{35} bytes).
- Artificial limit of $\mathbf{2^{12}}$ for pooled "chunks" (**AS7**-**AS14**).
  - Any **S** larger than $\mathbf{2^{12}}$ pass through to backing allocator.

Our Tool Chest of Allocation Strategies



These are all *exponents* of **2**.

3. Analyzing the Benchmark Data

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{30}

Inputs			Global	Virtual	Monotonic	Virtual	Multipool	Virtual	Multi<Mono>	Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
30	15	10	0.063s	103	440	435	46	43	46	47
30	16	10	0.069s	102	401	395	42	42	41	45
30	17	10	0.064s	110	435	428	46	44	47	46
30	18	10	0.063s	102	440	434	46	39	54	47
30	19	10	0.063s	104	439	434	51	46	47	47
30	20	10	0.064s	110	433	430	46	42	46	52
30	20	11	0.035s	125	758	747	54	37	49	37
30	20	12	0.022s	101	1216	1206	51	31	52	32
30	20	13	0.013s	60	1985	1961	110	67	1996	1979
30	20	14	0.008s	77	3356	3304	110	58	3276	3314
30	20	15	0.004s	74	5985	6288	60	111	6016	6057

Maximum Active Memory (M)

Size of Each Allocation (S)

Absolute Times

Relative Percentages

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{31}

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
31	15	10	0.127s	104	428	434	39	38	39	41
31	16	10	0.123s	102	442	446	42	42	41	40
31	17	10	0.124s	102	439	442	45	45	42	45
31	18	10	0.123s	102	442	447	47	46	41	42
31	19	10	0.123s	107	441	446	42	41	46	43
31	20	10	0.127s	99	431	434	44	42	41	41
31	20	11	0.064s	102	815	824	48	40	52	48
31	20	12	0.038s	93	1369	1387	57	51	47	54
31	20	13	0.021s	102	2368	2392	108	80	2376	2401
31	20	14	0.013s	61	3787	3833	109	67	3797	3844
31	20	15	0.007s	54	6621	6706	112	59	6651	6708

Maximum Active Memory (M)

Size of Each Allocation (S)

3. Analyzing the Benchmark Data

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{32}

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
32	15	10	0.248s	103	FAIL	FAIL	38	39	38	41
32	16	10	0.248s	102	FAIL	FAIL	38	41	38	39
32	17	10	0.246s	102	FAIL	FAIL	40	39	39	39
32	18	10	0.246s	102	FAIL	FAIL	40	40	39	40
32	19	10	0.246s	102	FAIL	FAIL	40	42	40	40
32	20	10	0.246s	102	FAIL	FAIL	40	41	41	41
32	20	11	0.124s	102	FAIL	FAIL	46	44	41	47
32	20	12	0.062s	102	FAIL	FAIL	44	45	46	56
32	20	13	0.034s	108	FAIL	FAIL	127	110	FAIL	FAIL
32	20	14	0.022s	72	FAIL	FAIL	105	78	FAIL	FAIL
32	20	15	0.015s	87	FAIL	FAIL	99	60	FAIL	FAIL

Maximum Active Memory (M)

Size of Each Allocation (S)

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{33}

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
33	15	10	0.495s	102	FAIL	FAIL	41	39	39	39
33	16	10	0.493s	102	FAIL	FAIL	38	39	38	41
33	17	10	0.492s	102	FAIL	FAIL	38	41	38	40
33	18	10	0.492s	102	FAIL	FAIL	40	41	39	40
33	19	10	0.492s	102	FAIL	FAIL	40	41	40	41
33	20	10	0.492s	102	FAIL	FAIL	40	40	40	41
33	20	11	0.248s	102	FAIL	FAIL	42	43	41	42
33	20	12	0.122s	101	FAIL	FAIL	43	47	45	47
33	20	13	0.062s	102	FAIL	FAIL	112	112	FAIL	FAIL
33	20	14	0.040s	89	FAIL	FAIL	96	88	FAIL	FAIL
33	20	15	0.022s	102	FAIL	FAIL	107	80	FAIL	FAIL

Maximum Active Memory (M)

Size of Each Allocation (S)

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{34}

Inputs			Global	Virtual	Monotonic	Virtual	Multipool	Virtual	Multi<Mono>	Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
34	15	10	0.990s	103	FAIL	FAIL	41	39	41	39
34	16	10	0.986s	102	FAIL	FAIL	38	39	38	40
34	17	10	0.985s	102	FAIL	FAIL	38	39	39	40
34	18	10	0.984s	102	FAIL	FAIL	40	40	39	40
34	19	10	0.983s	102	FAIL	FAIL	40	41	40	40
34	20	10	0.984s	102	FAIL	FAIL	40	41	40	41
34	20	11	0.494s	102	FAIL	FAIL	42	42	41	42
34	20	12	0.241s	102	FAIL	FAIL	43	42	47	44
34	20	13	0.120s	107	FAIL	FAIL	114	113	FAIL	FAIL
34	20	14	0.064s	102	FAIL	FAIL	117	112	FAIL	FAIL
34	20	15	0.038s	96	FAIL	FAIL	103	95	FAIL	FAIL

Maximum Active Memory (M)

Size of Each Allocation (S)

Benchmark III: UTILIZATION

Total Allocated Memory (T) = 2^{35}

Inputs			Global	Virtual	Monotonic	Virtual	Multipool	Virtual	Multi<Mono>	Virtual
T	M	S	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
35	15	10	1.981s	102	FAIL	FAIL	38	41	39	39
35	16	10	1.975s	102	FAIL	FAIL	39	40	38	39
35	17	10	1.970s	102	FAIL	FAIL	39	40	39	40
35	18	10	1.967s	102	FAIL	FAIL	39	39	39	40
35	19	10	1.967s	102	FAIL	FAIL	39	41	40	41
35	20	10	1.968s	102	FAIL	FAIL	40	41	40	40
35	20	11	0.988s	102	FAIL	FAIL	41	42	41	41
35	20	12	0.481s	102	FAIL	FAIL	42	42	44	44
35	20	13	0.240s	102	FAIL	FAIL	113	113	FAIL	FAIL
35	20	14	0.125s	102	FAIL	FAIL	113	112	FAIL	FAIL
35	20	15	0.070s	94	FAIL	FAIL	103	110	FAIL	FAIL

Maximum Active Memory (M)

Size of Each Allocation (S)

Benchmark III: UTILIZATION

Dimensional Characterization

Allocation DENSITY (**D**):	HIGH (2/3 due to single read op)
"Chunk"-Size VARIATION (**V**):	0
Access LOCALITY (**L**):	LOW
Memory UTILIZATION (**U**):	? (LOW)
Allocator CONTENTION (**C**):	0

Benchmark III: UTILIZATION

Take-Away Tips

- **Never** use a *monotonic* allocator by itself when the UTILIZATION (**U**) is low and the total number of bytes allocated is large (e.g., in a loop).
- **Beware** that a *multipool* allocator passes its request to its backing allocator when the "chunk" size of memory being allocated exceeds the maximum pool size (e.g., 2048).

Benchmark III: **U**TILIZATION

Questions and/or Discussion?

Benchmark IV: CONTENTION

I. **Short Running:** Build Up, Use, Tear Down
   - Allocation DENSITY and VARIATION in Allocated Sizes

II. **Long Running:** Time-Multiplexed Subsystems
   - Access LOCALITY – both *Physica*l and *Temporal*

III. **Short Running:** Varying Memory Reusability
   - Memory UTILIZATION

IV. **Multithreaded:** Varying Numbers of Threads
   - Allocator CONTENTION

Benchmark IV: CONTENTION



Benchmark IV: CONTENTION

Considerations:

- Investigate allocator **C**ONTENTION (**C**) .
  - Focus on allocation/deallocation costs themselves.
- Access **L**OCALITY (**L**) should *NOT* dominate results.
  - Increment just first byte of each newly allocated element.
- Identify sub-dimensions of **C**:
  - **C**: Expected number of concurrent allocations per thread.
  - **I (N/3)**: Num. alloc/access/dealloc sequences per thread.
  - **S**: Atomic memory "chunk" size (in bytes).
  - **W**: Number of active (worker) threads.

Benchmark IV: CONTENTION

Plan:

- For each of **W** (worker) threads:
  - Start the thread.
- For a total of **I** iterations
  - Allocate a chunk of memory of size ***S*** bytes.
  - Increment the first byte of the allocated memory.
  - Deallocate the chunk of memory (of size ***S***).
  - Join all threads.
- The results of this experiment are absolute run times.
  - The entries in each row, other than AS1, are relative to AS1.

Benchmark IV: CONTENTION

Additional Considerations:

- The allocation DENSITY (**D**) is HIGH.
- For A3-14, each thread had access to its own *private unsynchronized* local allocator.
- No contention occurs unless the local allocator goes to its backing (e.g., global) allocator.
- Unlike other targeted benchmarks, this experiment doesn't vary CONTENTION (**C**) over the range [0 .. 1]
  - CONTENTION is HIGH for **W** > 1 (0 for all other benchmarks).
- The # processors was greater than the max **W** = **8**.

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^{6}

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	6	1	0.041s	91	40	39	26	26	24	24
15	6	2	0.037s	100	42	43	27	26	26	29
15	6	3	0.038s	105	41	43	15	16	17	16
15	6	4	0.032s	93	56	58	31	32	25	24
15	6	5	0.032s	91	46	52	26	23	22	24
15	6	6	0.030s	95	51	53	24	27	26	27
15	6	7	0.033s	96	47	49	23	28	21	26
15	6	8	0.029s	96	71	63	33	30	31	25

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^6

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	6	1	0.041s	91	40	39	26	26	24	24
15	6	2	0.037s	100	42	43	27	26	26	29
15	6	3	0.038s	105	41	43	15	16	17	16
15	6	4	0.032s	93	56	58	31	32	25	24
15	6	5	0.032s	91	46	52	26	23	22	24
15	6	6	0.030s	95	51	53	24	27	26	27
15	6	7	0.033s	96	47	49	23	28	21	26
15	6	8	0.029s	96	71	63	33	30	31	25

Number of Active Threads (W)

3. Analyzing the Benchmark Data

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^{6}

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	6	1	0.041s	91	40	39	26	26	24	24
15	6	2	0.037s	100	42	43	27	26	26	29
15	6	3	0.038s	105	41	43	15	16	17	16
15	6	4	0.032s	93	56	58	31	32	25	24
15	6	5	0.032s	91	46	52	26	23	22	24
15	6	6	0.030s	95	51	53	24	27	26	27
15	6	7	0.033s	96	47	49	23	28	21	26
15	6	8	0.029s	96	71	63	33	30	31	25

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^6

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	6	1	0.041s	91	40	39	26	26	24	24
15	6	2	0.037s	100	42	43	27	26	26	29
15	6	3	0.038s	105	41	43	15	16	17	16
15	6	4	0.032s	93	56	58	31	32	25	24
15	6	5	0.032s	91	46	52	26	23	22	24
15	6	6	0.030s	95	51	53	24	27	26	27
15	6	7	0.033s	96	47	49	23	28	21	26
15	6	8	0.029s	96	71	63	33	30	31	25

Number of Active Threads (W)

490

1020

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^{6}

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	6	1	0.041s	91	40	39	26	26	24	24
15	6	2	0.037s	100	42	43	27	26	26	29
15	6	3	0.038s	105	41	43	15	16	17	16
15	6	4	0.032s	93	56	58	31	32	25	24
15	6	5	0.032s	91	46	52	26	23	22	24
15	6	6	0.030s	95	51	53	24	27	26	27
15	6	7	0.033s	96	47	49	23	28	21	26
15	6	8	0.029s	96	71	63	33	30	31	25

Number of Active Threads (W)

3. Analyzing the Benchmark Data

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^7

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	7	1	0.023s	100	114	116	44	47	47	48
15	7	2	0.043s	101	46	69	26	26	26	26
15	7	3	0.041s	103	51	68	25	25	22	25
15	7	4	0.033s	121	78	95	26	19	20	23
15	7	5	0.031s	102	81	86	20	26	26	25
15	7	6	0.032s	99	84	84	18	23	19	25
15	7	7	0.029s	114	111	110	23	27	21	31
15	7	8	0.029s	117	114	120	27	35	31	29

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^8

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	8	1	0.043s	101	87	89	23	23	22	23
15	8	2	0.042s	102	61	59	23	23	27	26
15	8	3	0.046s	90	85	111	23	25	24	25
15	8	4	0.040s	84	100	98	18	18	19	22
15	8	5	0.028s	136	190	200	30	30	30	38
15	8	6	0.024s	125	209	201	33	33	31	29
15	8	7	0.033s	108	162	162	24	29	26	26
15	8	8	0.031s	114	184	188	34	33	36	42

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{15}

Allocation Size (S) = 2^{8}

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
15	8	1	0.043s	101	87	89	23	23	22	23
15	8	2	0.042s	102	61	59	23	23	27	26
15	8	3	0.046s	90	85	111	23	25	24	25
15	8	4	0.040s	84	100	98	18	18	19	22
15	8	5	0.028s	136	190	200	30	30	30	38
15	8	6	0.024s	125	209	201	33	33	31	29
15	8	7	0.033s	108	162	162	24	29	26	26
15	8	8	0.031s	114	184	188	34	33	36	42

Number of Active Threads (W)

1024

3. Analyzing the Benchmark Data

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{16}

Allocation Size (S) = 2^{8}

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
16	8	1	0.085s	97	109	107	23	23	23	23
16	8	2	0.091s	101	104	106	22	21	22	21
16	8	3	0.093s	100	105	104	22	21	21	21
16	8	4	0.097s	94	93	121	20	20	18	17
16	8	5	0.078s	118	108	130	24	18	17	18
16	8	6	0.059s	87	138	136	21	26	22	26
16	8	7	0.063s	93	137	135	17	27	21	20
16	8	8	0.057s	109	162	164	29	28	28	26

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{17}

Allocation Size (S) = 2^{8}

Poor Choice!

Inputs			Global	Virtual	Monotonic	Virtual	Multipool	Virtual	Multi<Mono>	Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
17	8	1	0.090s	100	206	206	45	42	42	42
17	8	2	0.179s	101	107	106	22	22	22	22
17	8	3	0.179s	101	104	104	22	23	22	22
17	8	4	0.209s	109	89	70	16	15	11	11
17	8	5	0.177s	100	85	78	12	15	15	15
17	8	6	0.108s	142	147	178	27	28	25	25
17	8	7	0.140s	85	116	132	24	22	22	22
17	8	8	0.118s	100	142	150	22	21	25	26

Number of Active Threads (W)

3. Analyzing the Benchmark Data

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{18}

Allocation Size (S) = 2^{8}

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
18	8	1	0.177s	109	177	177	45	45	45	46
18	8	2	0.339s	100	95	95	24	24	24	24
18	8	3	0.333s	102	99	95	24	25	24	25
18	8	4	0.304s	98	93	93	24	21	26	21
18	8	5	0.311s	94	97	86	22	24	25	20
18	8	6	0.276s	95	118	122	16	17	18	17
18	8	7	0.297s	79	109	108	18	18	21	18
18	8	8	0.219s	114	176	186	26	21	21	23

Number of Active Threads (W)

Benchmark IV: CONTENTION

Number of Iterations (I) = 2^{19}

Allocation Size (S) = 2^{8}

Poor Choice!

Inputs			Global		Monotonic		Multipool		Multi<Mono>	
				Virtual		Virtual		Virtual		Virtual
I	S	W	AS1	AS2	AS3	AS5	AS7	AS9	AS11	AS13
19	8	1	0.421s	89	134	134	28	23	21	25
19	8	2	0.615s	101	93	93	25	26	26	26
19	8	3	0.631s	99	93	93	25	25	25	25
19	8	4	0.565s	107	95	103	28	28	29	28
19	8	5	0.575s	119	106	101	27	28	27	27
19	8	6	0.499s	114	126	113	17	25	28	22
19	8	7	0.558s	100	113	115	18	18	15	16
19	8	8	0.460s	105	149	148	19	21	18	21

Number of Active Threads (W)

3. Analyzing the Benchmark Data

Benchmark IV: CONTENTION

Dimensional Characterization

Allocation DENSITY (**D**):	HIGH
"Chunk"-Size VARIATION (**V**):	0
Access LOCALITY (**L**):	LOW
Memory UTILIZATION (**U**):	NIL
Allocator CONTENTION (**C**):	? (0 or HIGH)

Benchmark IV: CONTENTION

Take-Away Tips

- Make subsystems' memory explicit using local allocators to obviate *affinity* in thread pools as well as thread-aware global memory allocators.
- Use a (composite) `Multi<Mono>` allocation strategy (AS11-AS14) when **U**TILIZATION (**U**) is low – especially when **V**ARIATION (**V**) is high – so long as the sizes of all the individual "chunks" are smaller than the *pooling threshold* (e.g., 2K bytes).

Benchmark IV: **C**ONTENTION

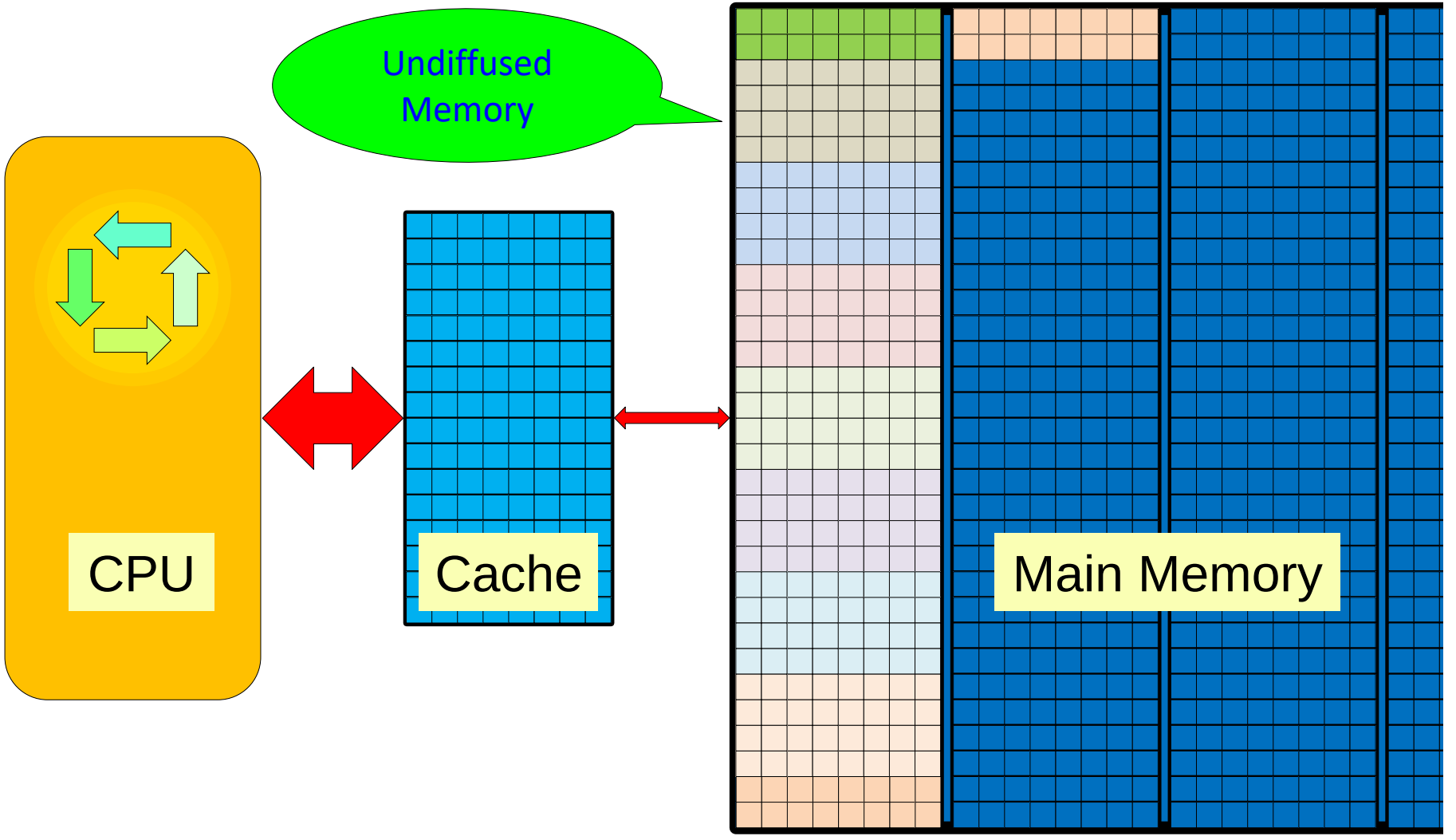
Questions and/or Discussion?

Another Use For Local Allocators

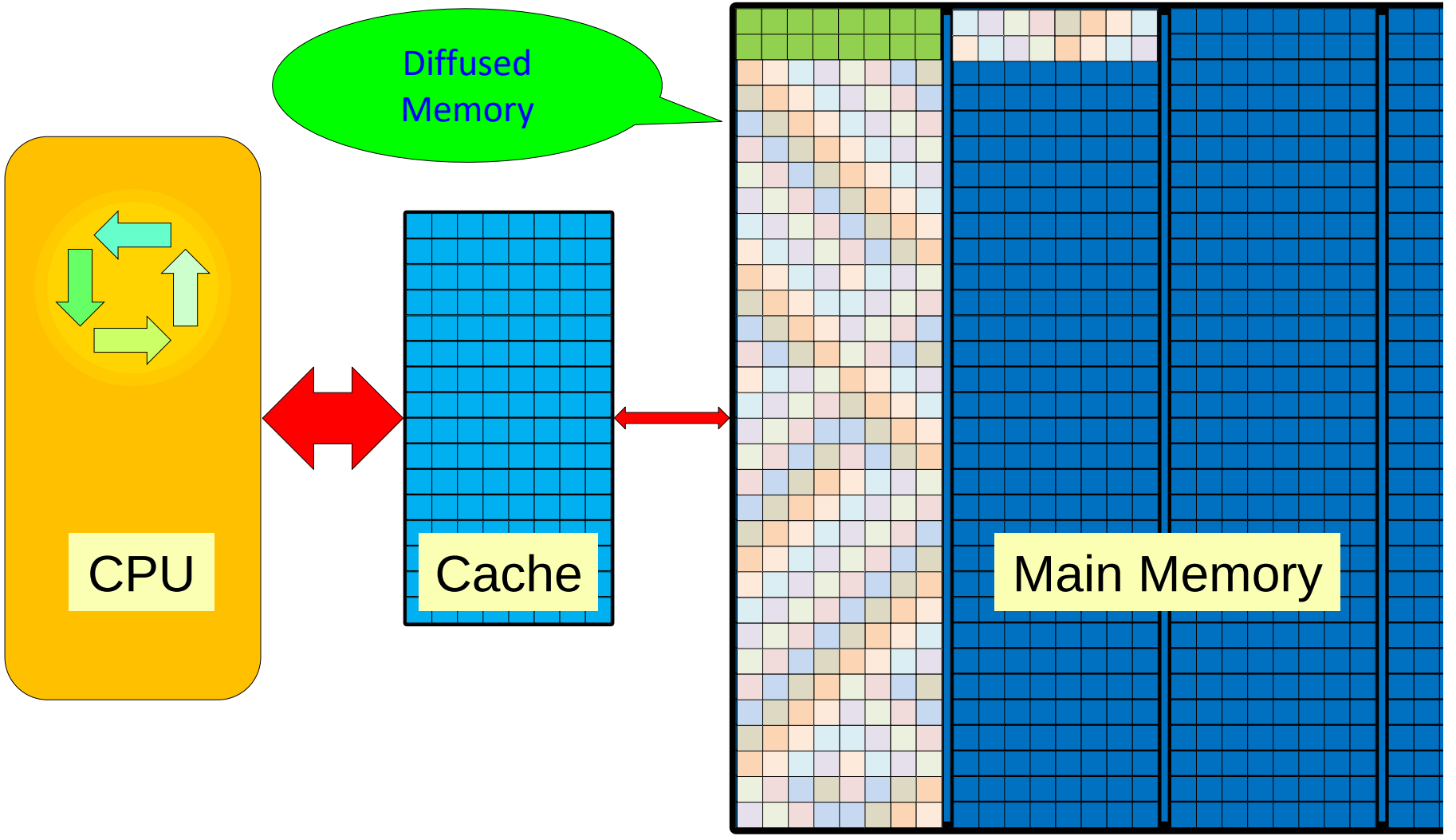
Preventing "*False Sharing*"

Two or more concurrent threads
contending for independent data
(unfortunately)
located on the same cache line.

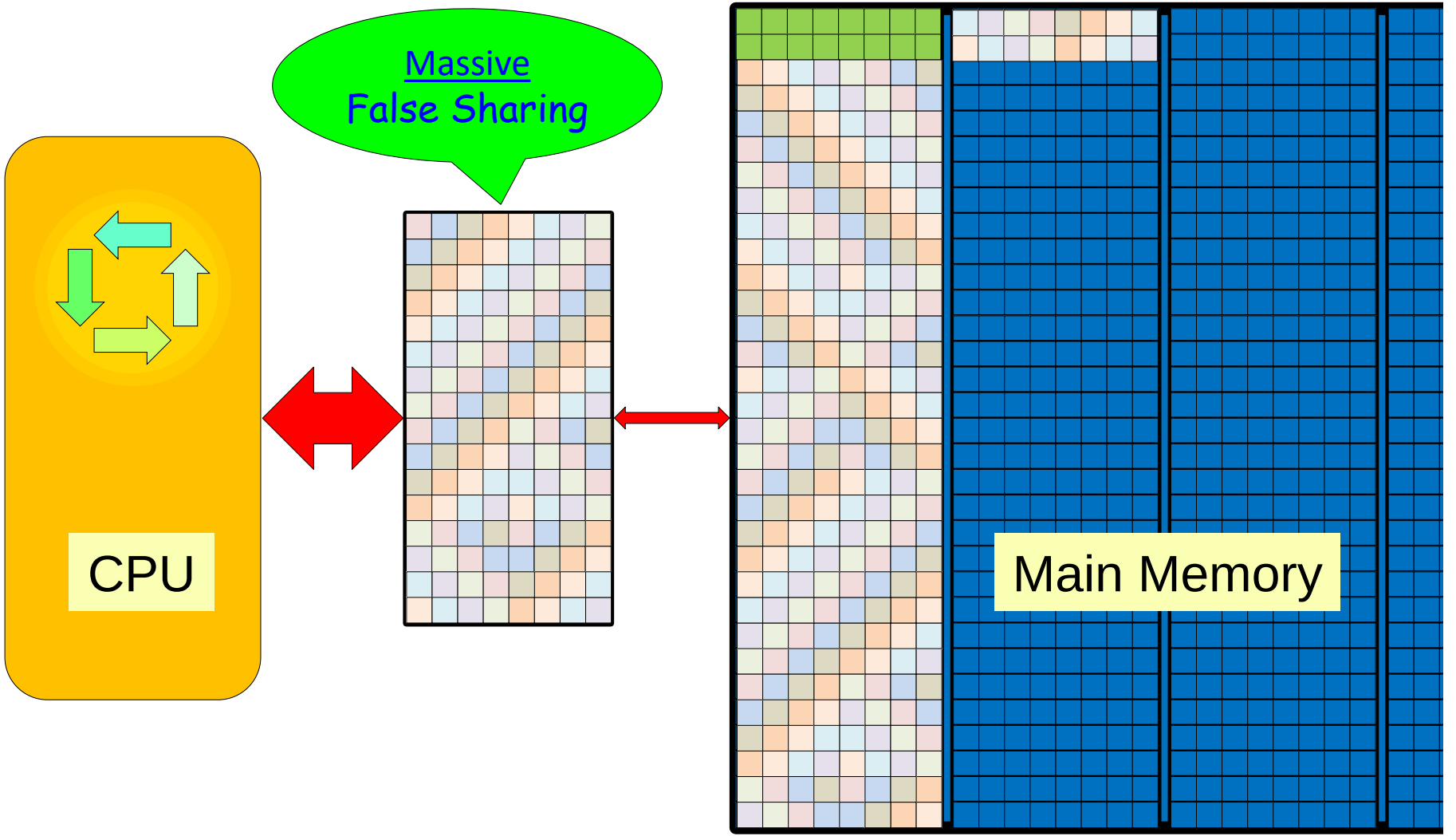
Another Use For Local Allocators



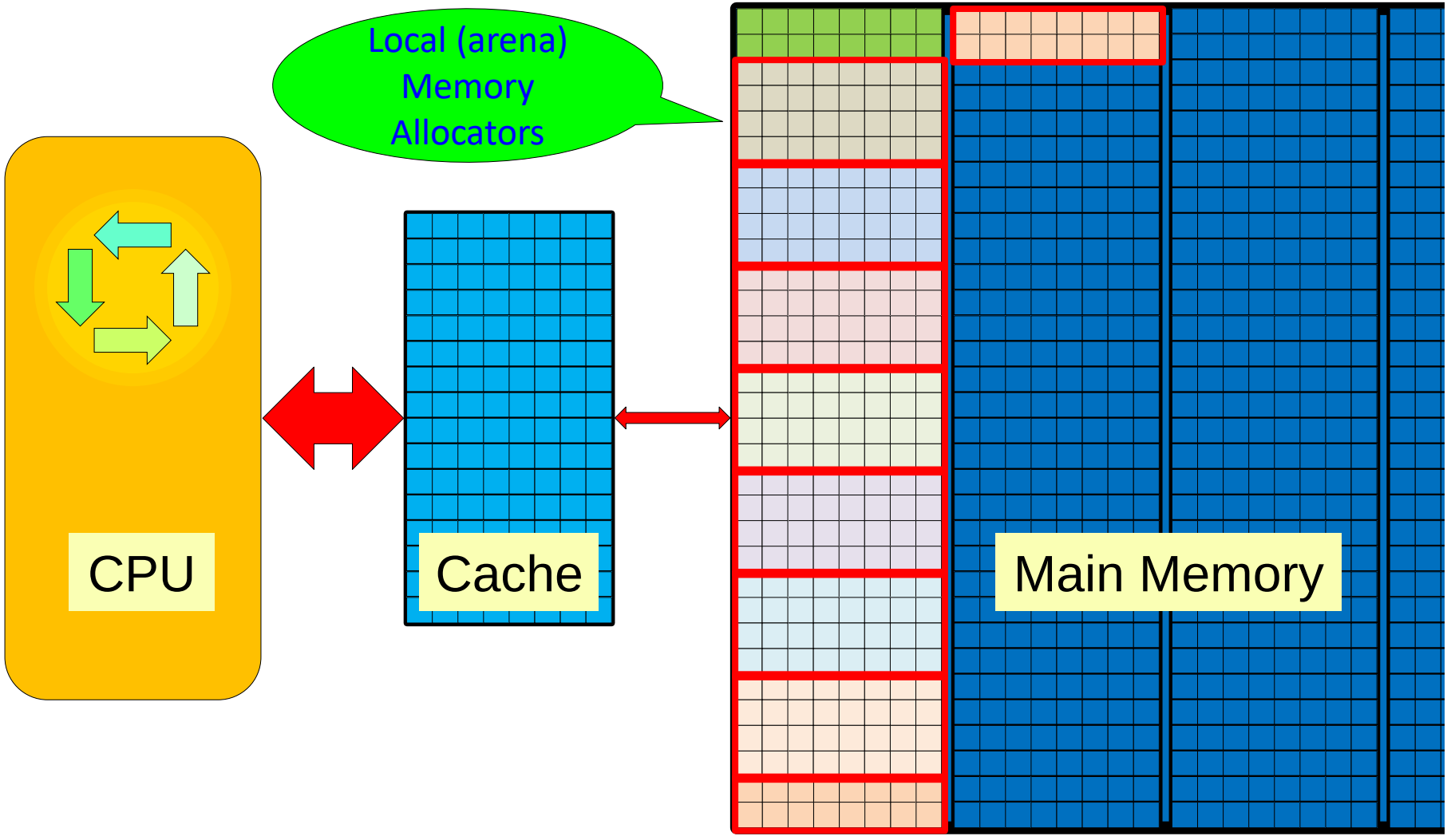
Another Use For Local Allocators



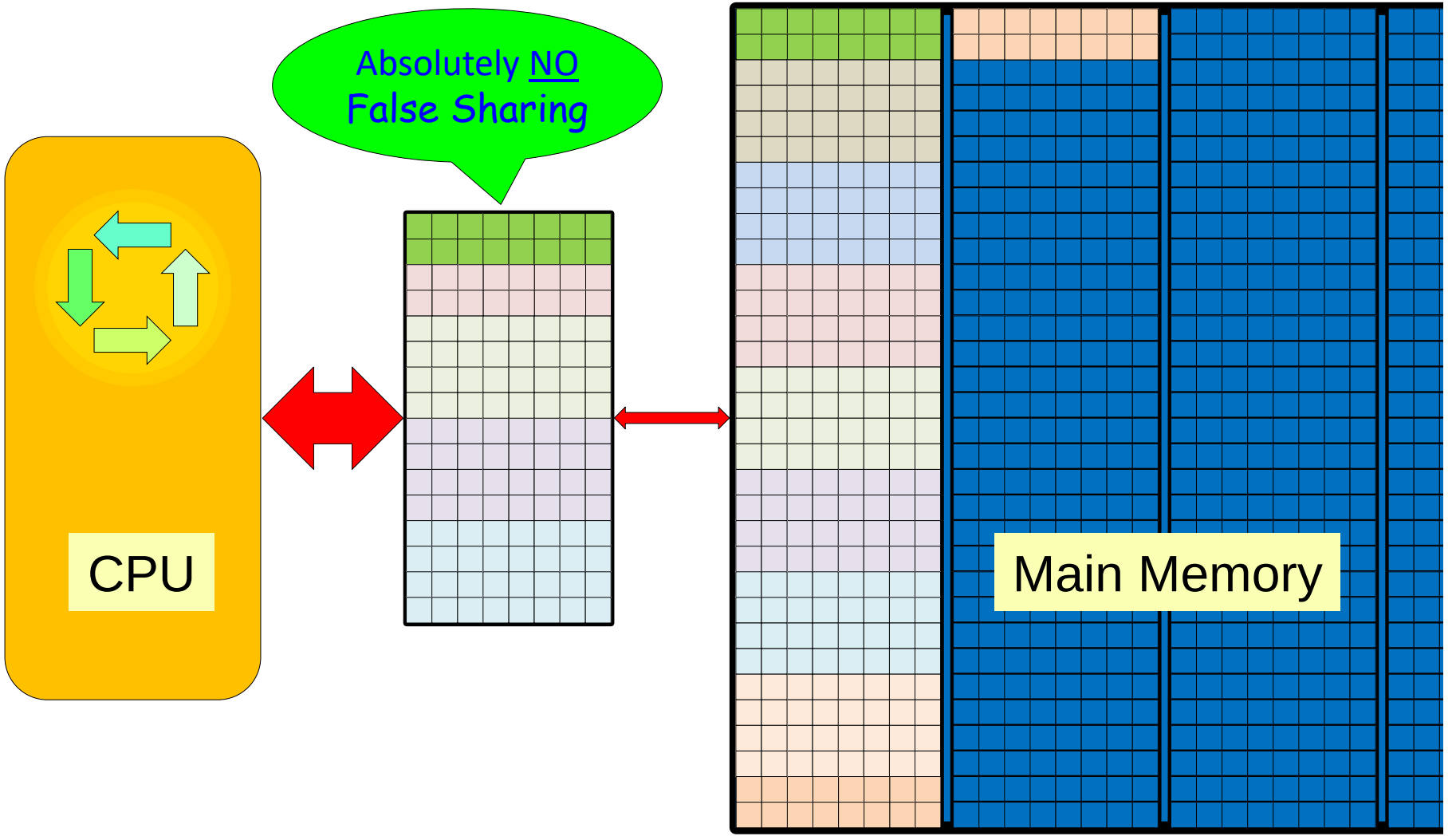
Another Use For Local Allocators



Another Use For Local Allocators



Another Use For Local Allocators



Houston, we have a problem!

But there was a problem…

- Some of the original data was unexplainable.
- We needed a strategy to understand why.
- More work was needed.
- We needed to “confirm” the data!
- How we solved the problem (next slide):
  - A revised paper: **Doc No:** P0089R1
  - A new paper: **Doc No:** P0213R0

The Solution: Graham Bleaney

Enter Graham as Co-Op at Bloomberg (May, 2014).

The Solution: Graham Bleaney

Enter Graham as Co-Op at Bloomberg (May, 2014).

- From P0089R1:

The Solution: Graham Bleaney

Enter Graham as Co-Op at Bloomberg (May, 2014).

- From P0089R1:

  "… a separate effort has recently been made to recreate our experiments in order to confirm these results (P0213 by Graham Bleaney). We anticipate that paper will appear at approximately the same time as this revision."

The Solution: Graham Bleaney

Enter Graham as Co-Op at Bloomberg (May, 2014).

- From P0089R1:

  "… a separate effort has recently been made to recreate our experiments in order to confirm these results (P0213 by Graham Bleaney). We anticipate that paper will appear at approximately the same time as this revision."

- New *allocator-use dimension,* **FRAGMENTABILITY** (**F**):

The Solution: Graham Bleaney

Enter Graham as Co-Op at Bloomberg (May, 2014).

- From P0089R1: Joins us as FTE (Summer, 2017)!

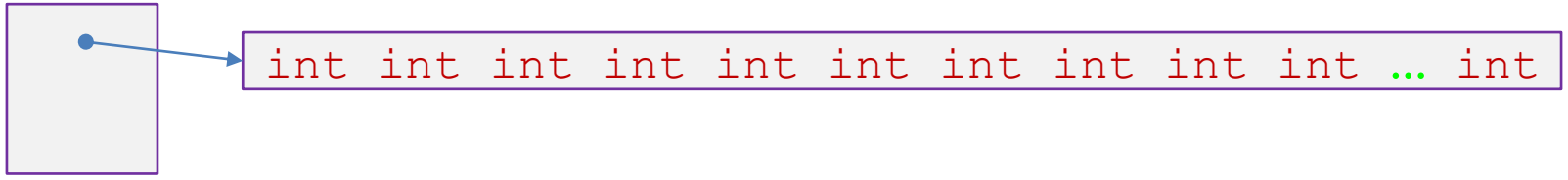
  "… a separate effort has recently been made to recreate our experiments in order to confirm these results (P0213 by Graham Bleaney). We anticipate that paper will appear at approximately the same time as this revision."

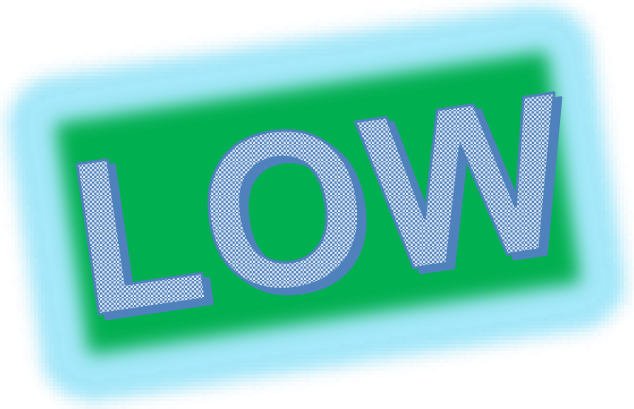
- New *allocator-use dimension,* **FRAGMENTABILITY** (**F**):
  "A measure of the potential of a subsystem's allocated memory to become diffused throughout physical memory, as a result of the interference of other subsystems' memory allocation. If a subsystem is ***fragmentable*** (i.e., other subsystems are present in the process and the subsystem allocates more than one chunk of memory), (**F**) is greater than zero."

3. Analyzing the Benchmark Data

FRAGMENTABILITY (F)

DS1	vector<int>

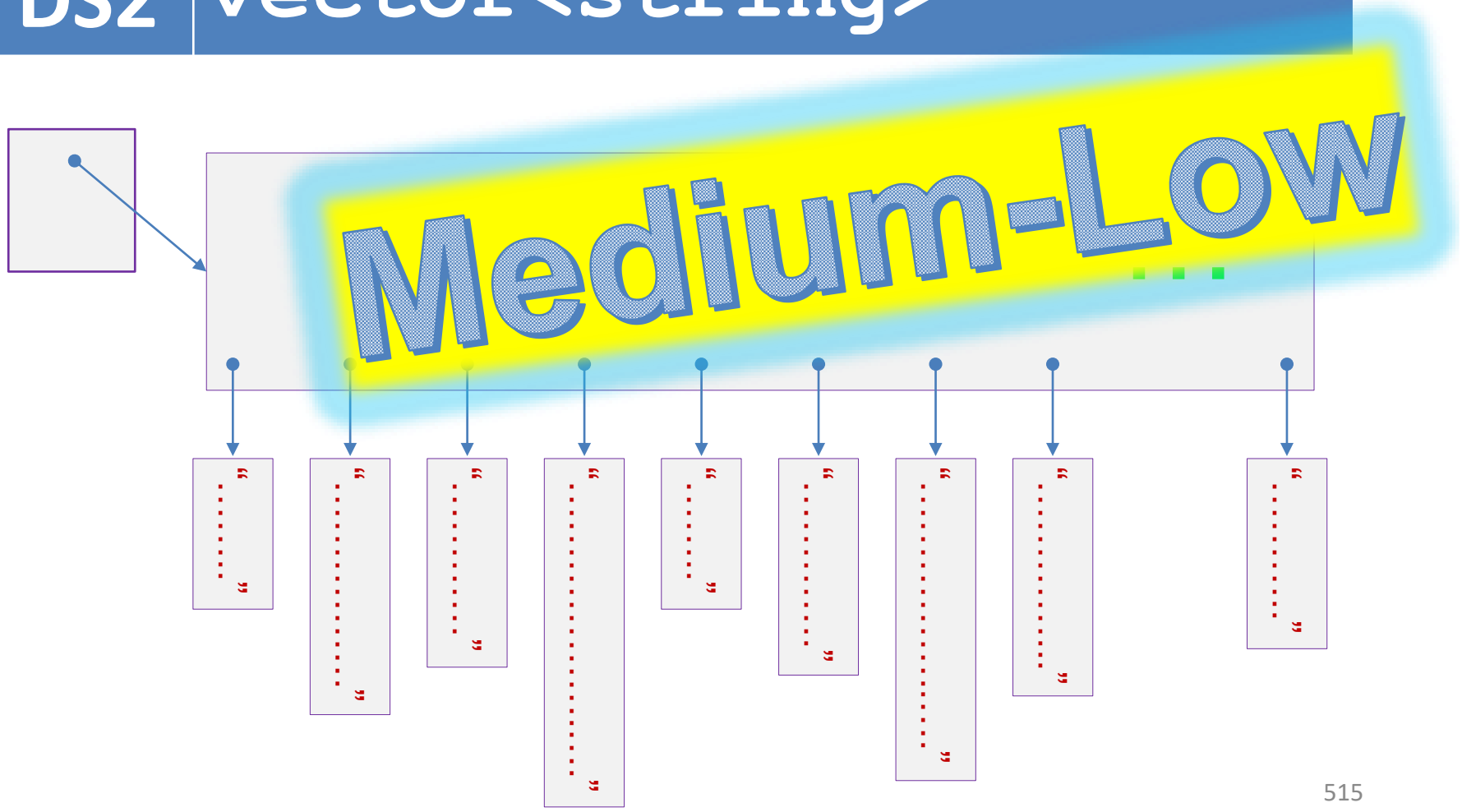




3. Analyzing the Benchmark Data

FRAGMENTABILITY (F)

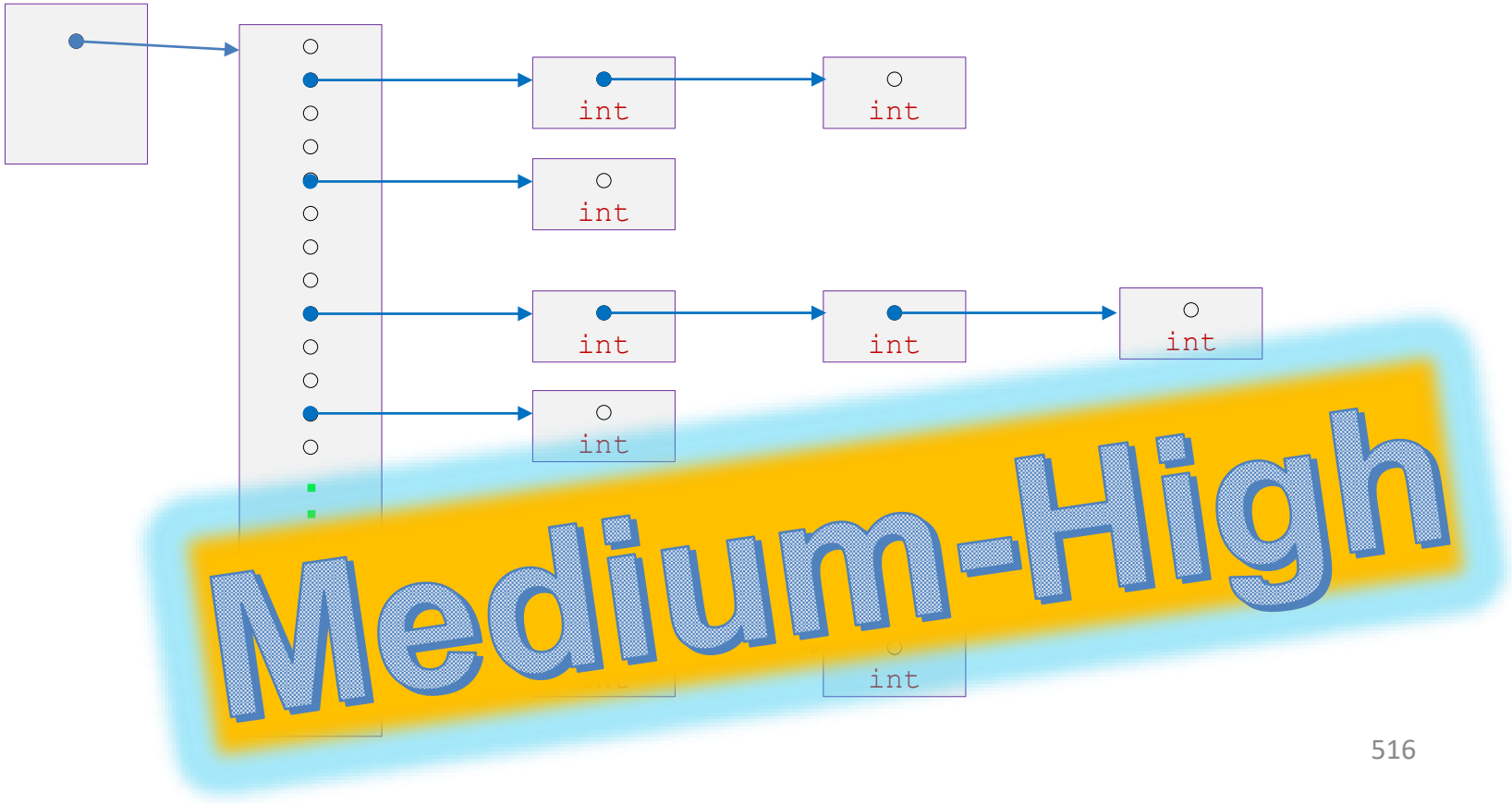
DS2	vector<string>



3. Analyzing the Benchmark Data

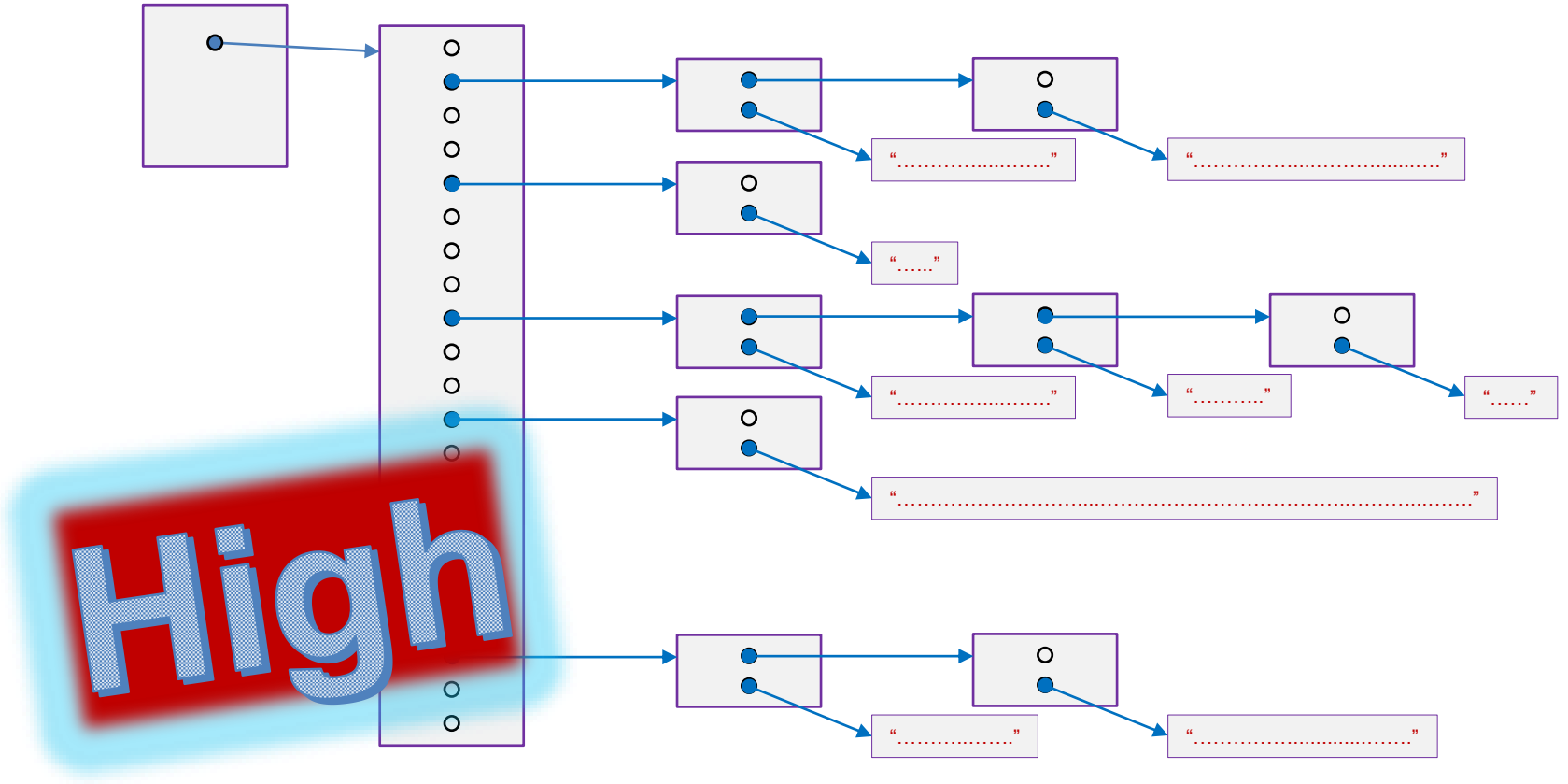
FRAGMENTABILITY (F)

DS3	unordered_set<int>



FRAGMENTABILITY (F)

DS4 unordered_set<string>



The C++ Standardization Process

C++ Standards Committee Meeting:

- Jacksonville, Florida.
  - February 29 thru March 5, 2016.
  - Graham presents his findings to the LEWG.
- **Polymorphic Memory Resources (PMR) was adopted into C++17 on March 5, 2016!**
  - Along with ***both*** of ***our*** local ("arena") allocators:
    - **Monotonic**
    - **Multipool**

References

- **References**
- [1] The Bloomberg BDE Library open-source distribution, https://github.com/bloomberg/bde
- [2] John Lakos, *Large Scale C++ Software Design*, Addison-Wesley, 1996.
- [3] Pablo Halpern, *N3916: Polymorphic Memory Resources*.
- [4] Memory Allocator Benchmark Data, https://github.com/bloomberg/bde-allocator-benchmarks
- [5] Graham Bleaney, *P0213R0: Reexamining the Performance of Memory-Allocation Strategies.*
- [6] John Lakos, Jeffrey Mendelsohn, Alisdair Meredith, Nathan Myers, *P0089R1: On Quantifying Memory-Allocation Strategies (Revision 2).*

Outline

1. Introduction and Background
   - What are memory allocators, and why are they useful?
2. Understanding the Problem
   - What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   - When and how do you use which allocator, and why?
4. Conclusions
   - What must we remember about memory allocators?

Outline

1. Introduction and Background
   What are memory allocators, and why are they useful?
2. Understanding the Problem
   What aspects of software affect allocation strategy?
3. Analyzing the Benchmark Data
   When and how do you use which allocator, and why?
4. Conclusions
   What must we remember about memory allocators?

Important Recurring Questions

Are memory allocators really worth the trouble?

Yes.

What situations merit their use?

There are a few qualitatively different use cases:

A. To improve and/or preserve performance:

- Ensure physical locality of allocated memory.
- Avoid memory diffusion in long-running systems.
- Obviate deallocation of individual objects.
- Sidestep contention during concurrent allocations.
- Separate unrelated data to avoid false sharing.
- Compose effective allocation strategies.

What situations merit their use?

There are a few qualitatively different use cases:

B. To place objects in a specific *kind* of memory:

- Static memory
- Memory-mapped memory
- Read/write protectable memory
- Fast memory (special architectures)
- Shared memory (special allocators)

What situations merit their use?

There are a few qualitatively different use cases:

C. To measure, test, control, or debug memory:

- Counting (auditing) allocator
- Test allocator
- Limit allocator
- Read/write protectable memory allocator
- Alarm allocator (if *monotonic* allocator overflows).

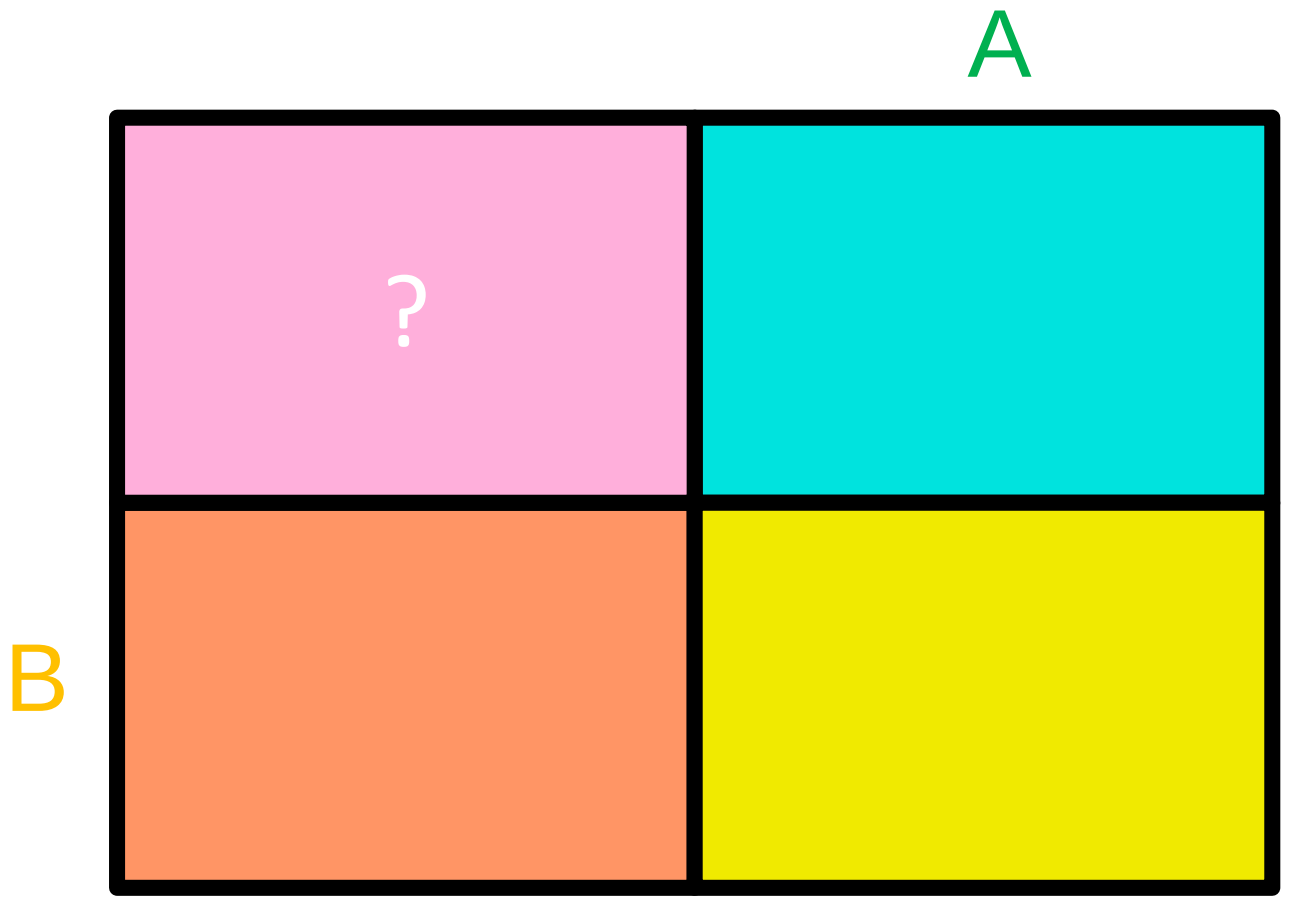
4. Conclusion

How are they applied effectively?

How are they applied effectively?

A: Program is large, long running.

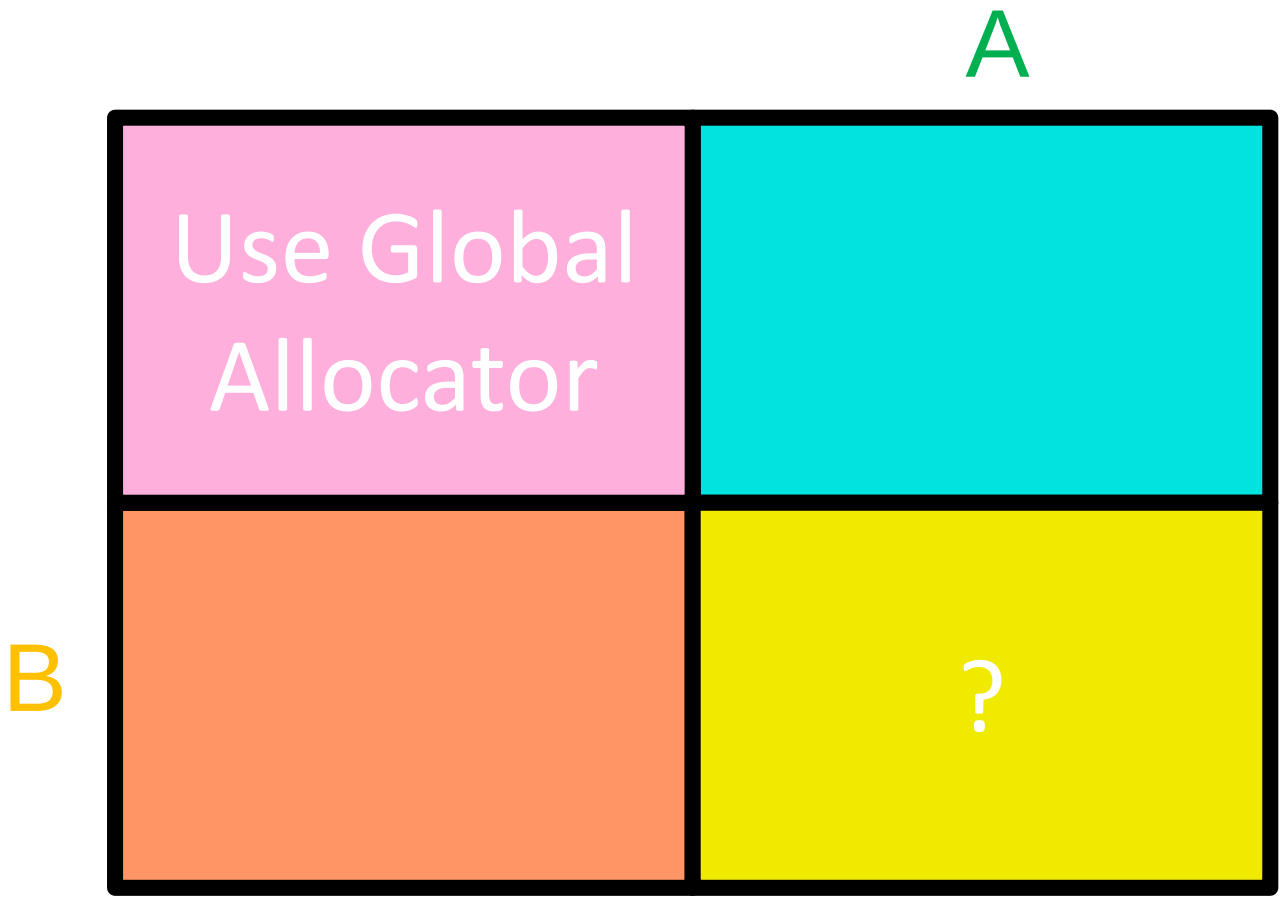
B: Subsystems are accessed disproportionally often (L).



How are they applied effectively?

A: Program is large, long running.

B: Subsystems are accessed disproportionally often (L).

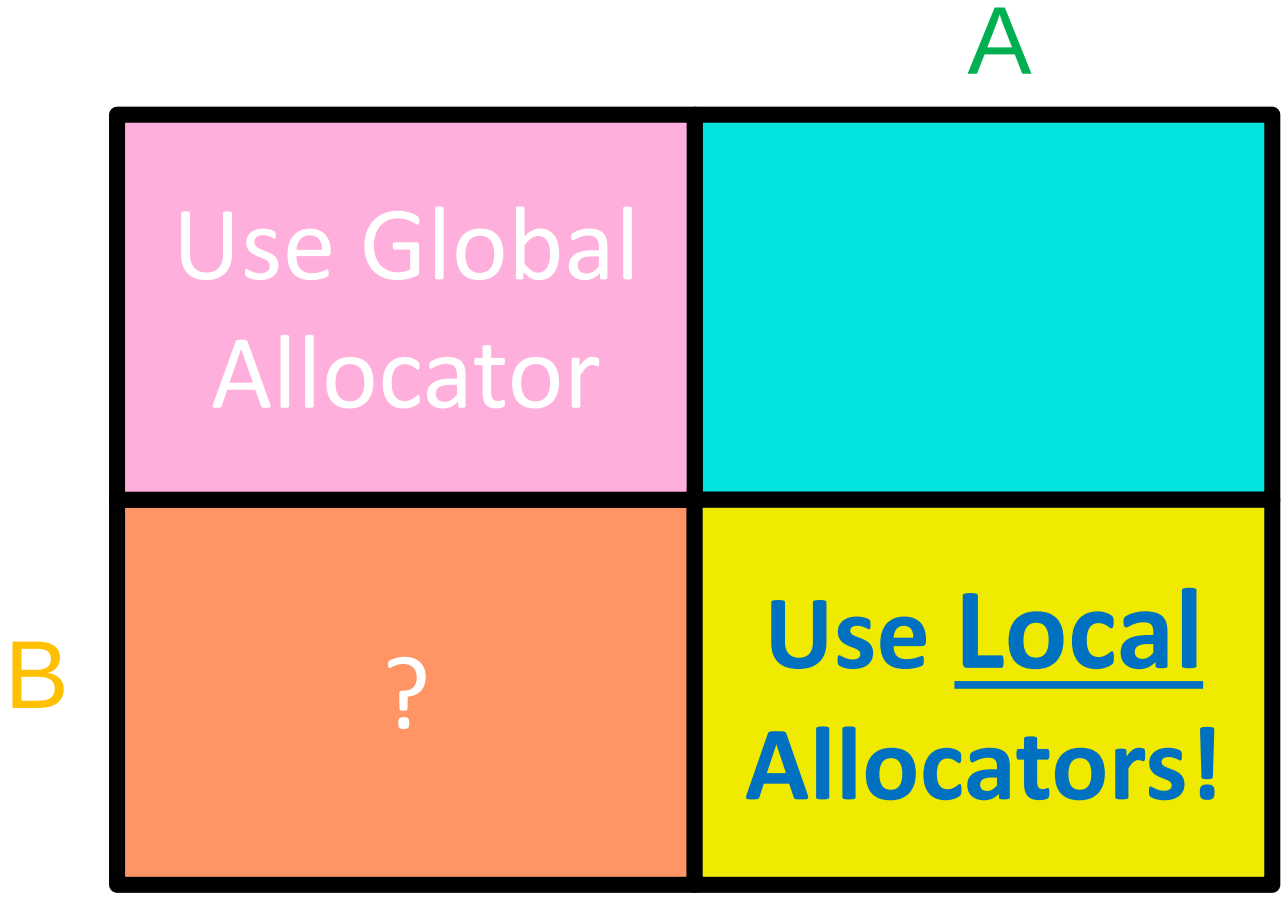


4. Conclusion

How are they applied effectively?

A: Program is large, long running.

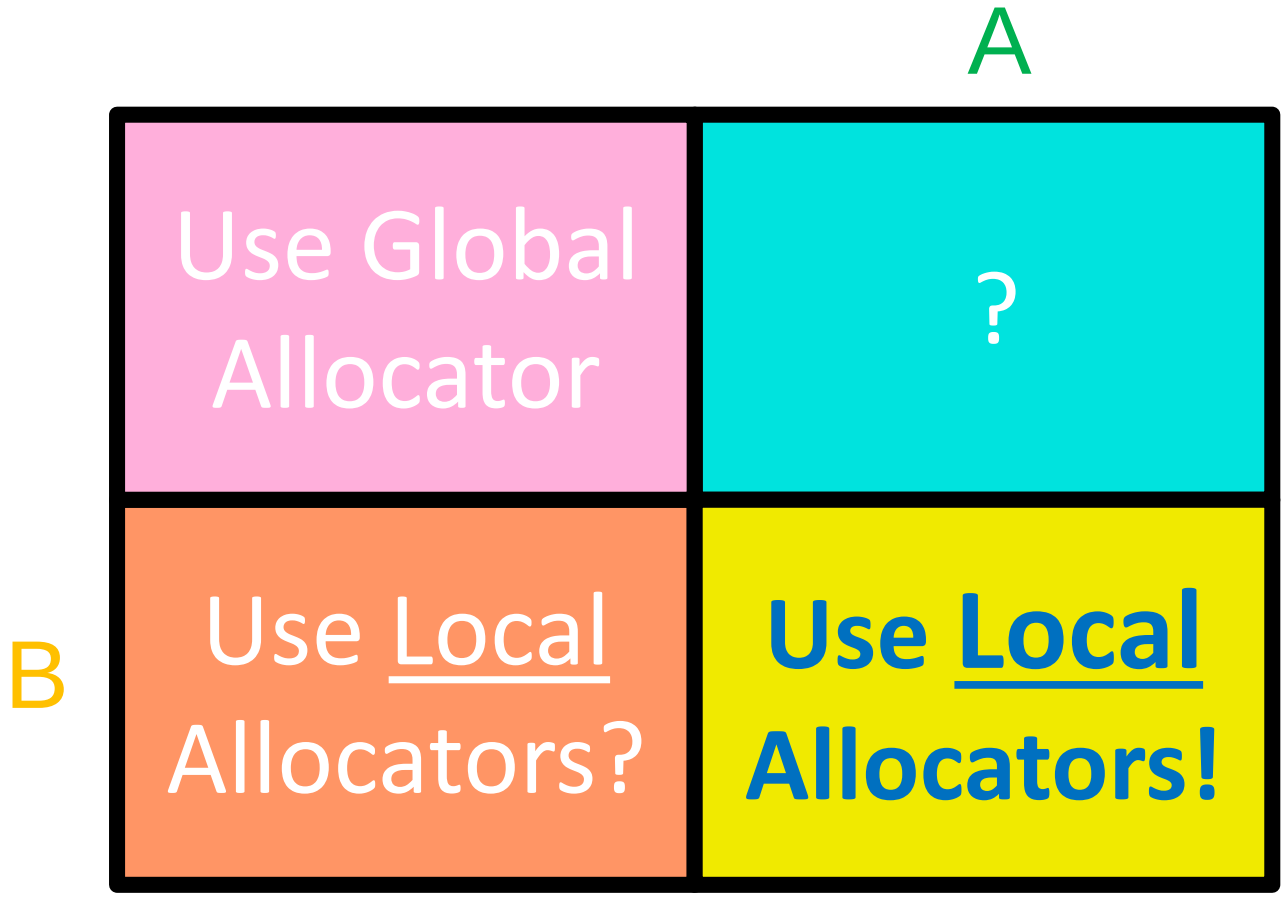
B: Subsystems are accessed disproportionally often (L).



How are they applied effectively?

A: Program is large, long running.

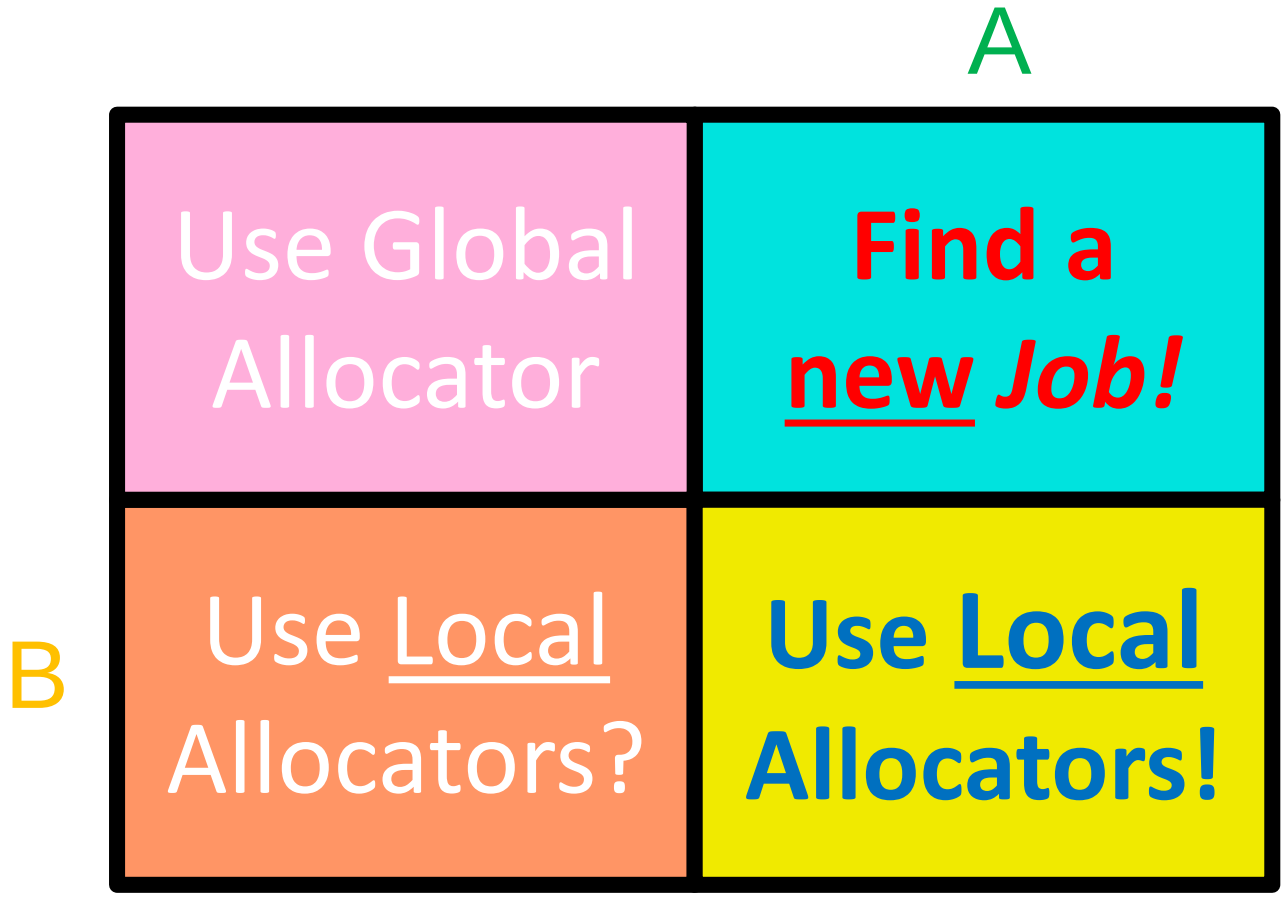
B: Subsystems are accessed disproportionally often (L).



How are they applied effectively?

A: Program is large, long running.

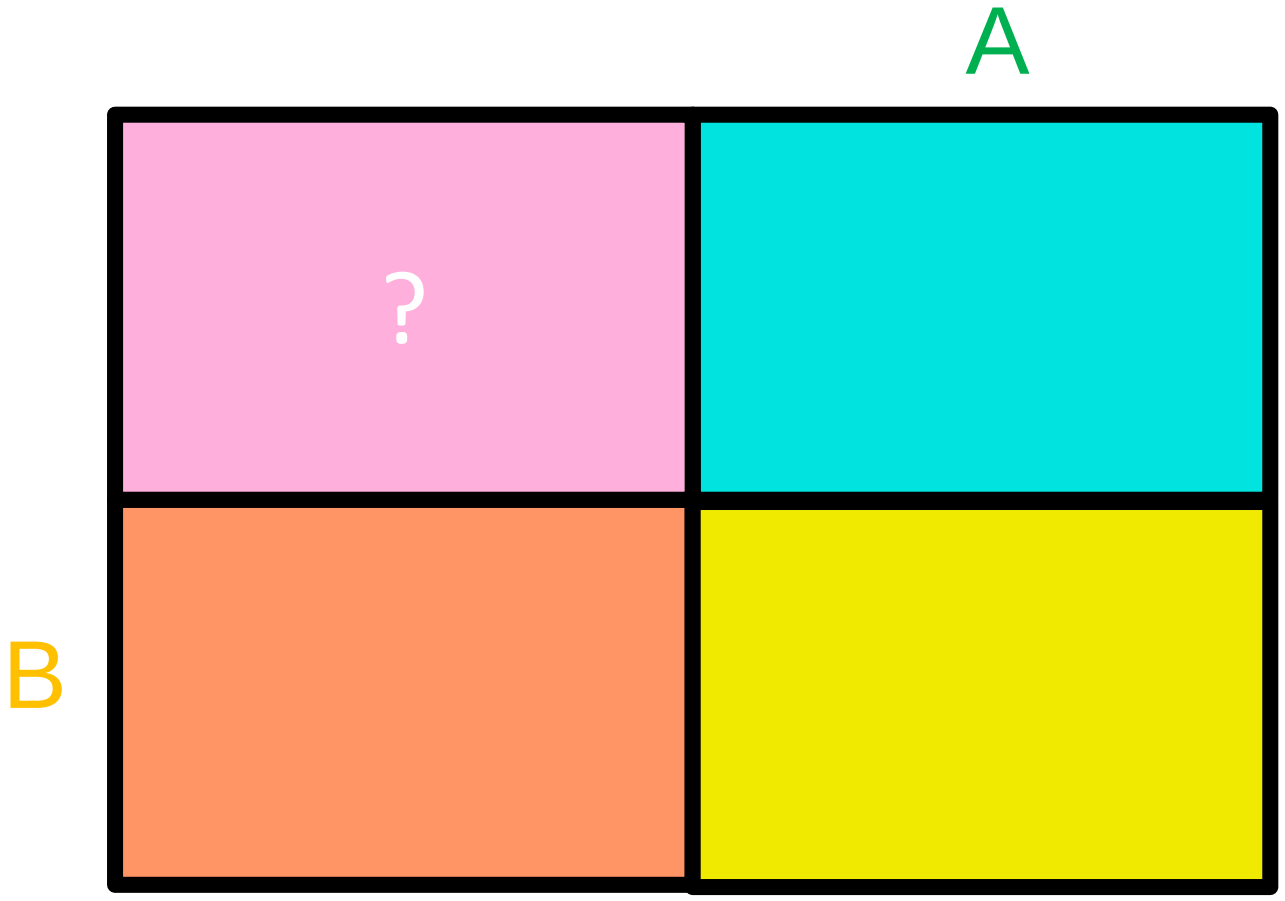
B: Subsystems are accessed disproportionally often (L).



How are they applied effectively?

A: Program is large, long running.

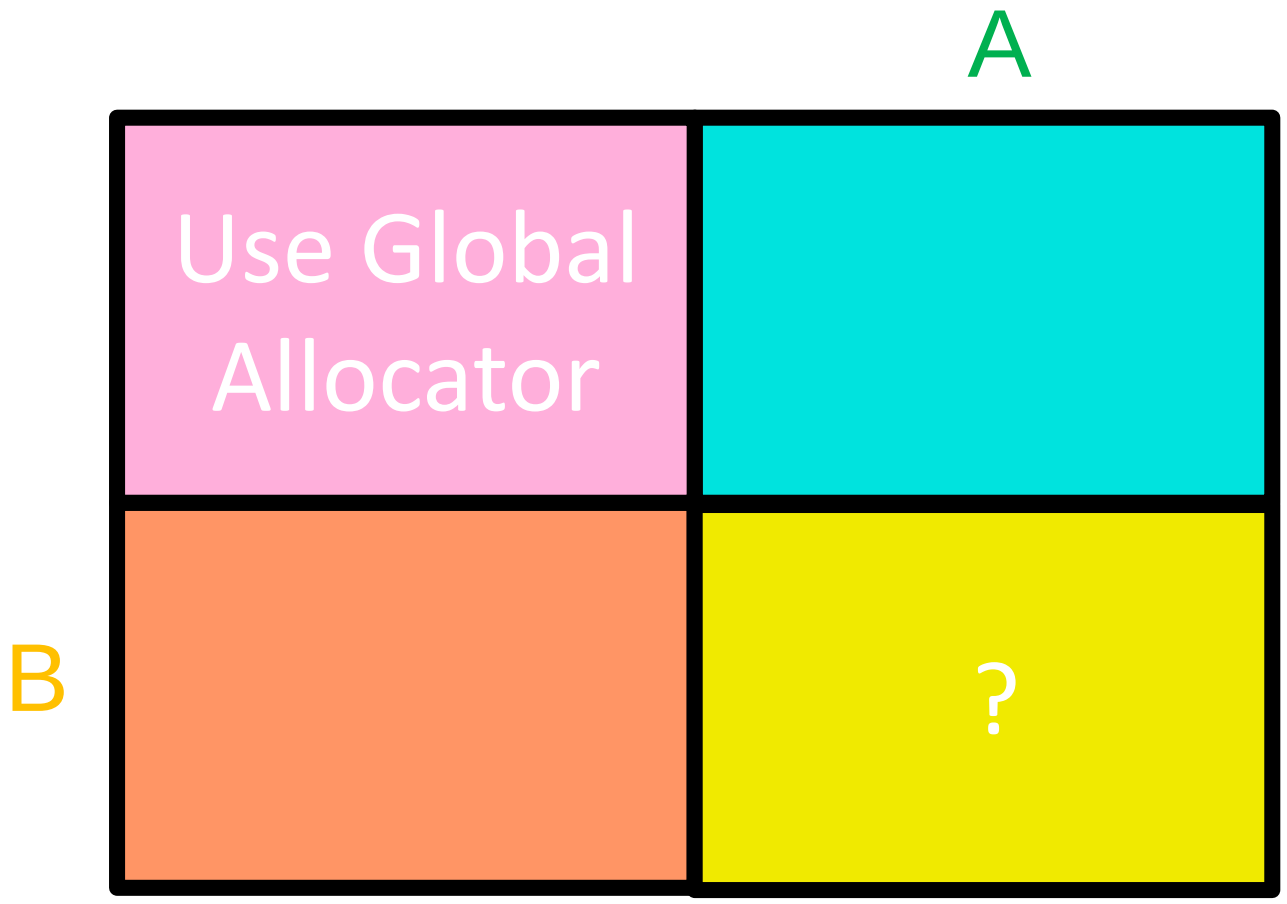
B: Subsystems exhibit high memory utilization (U).



How are they applied effectively?

A: Program is large, long running.

B: Subsystems exhibit high memory utilization (U).

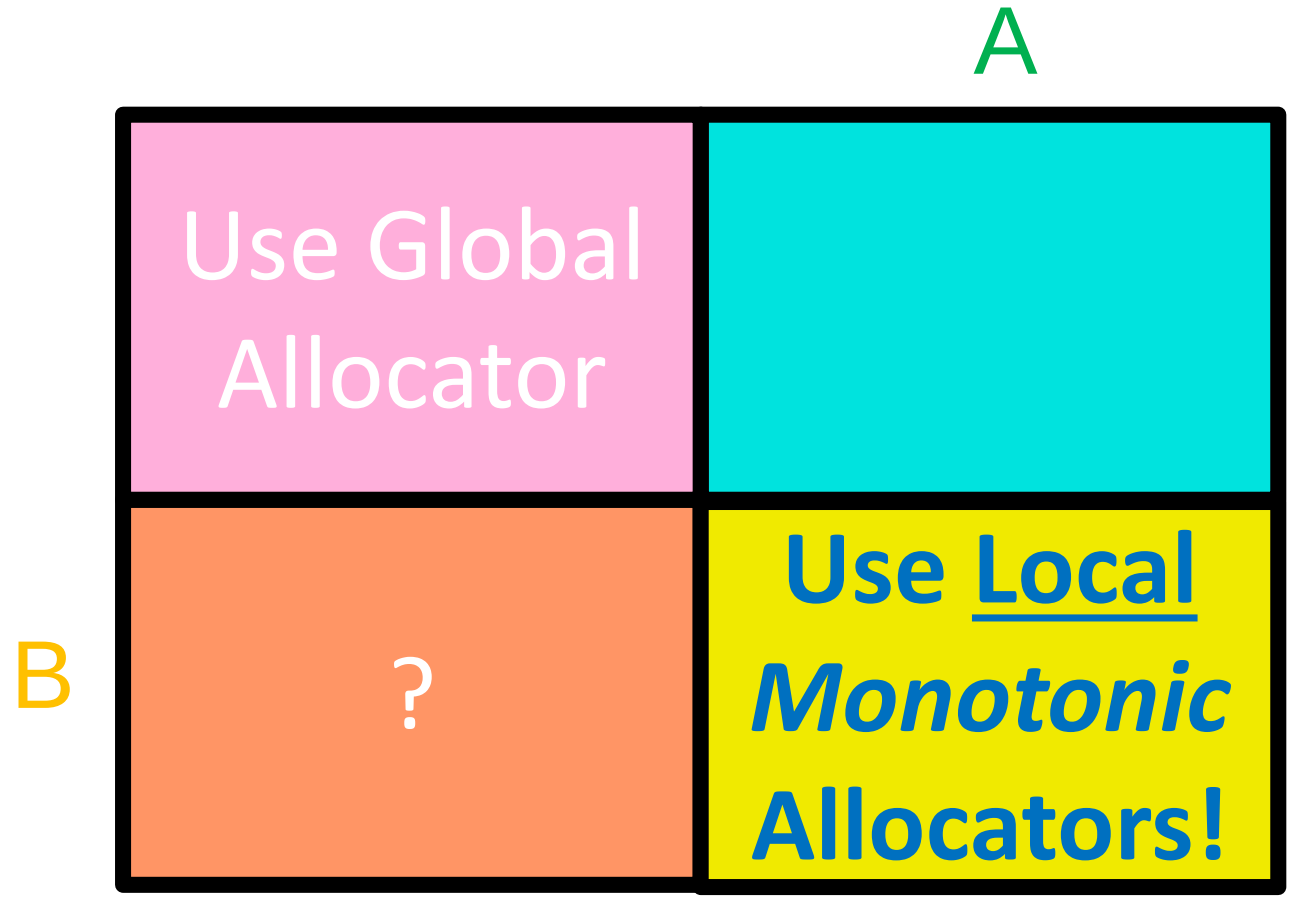


4. Conclusion

How are they applied effectively?

A: Program is large, long running.

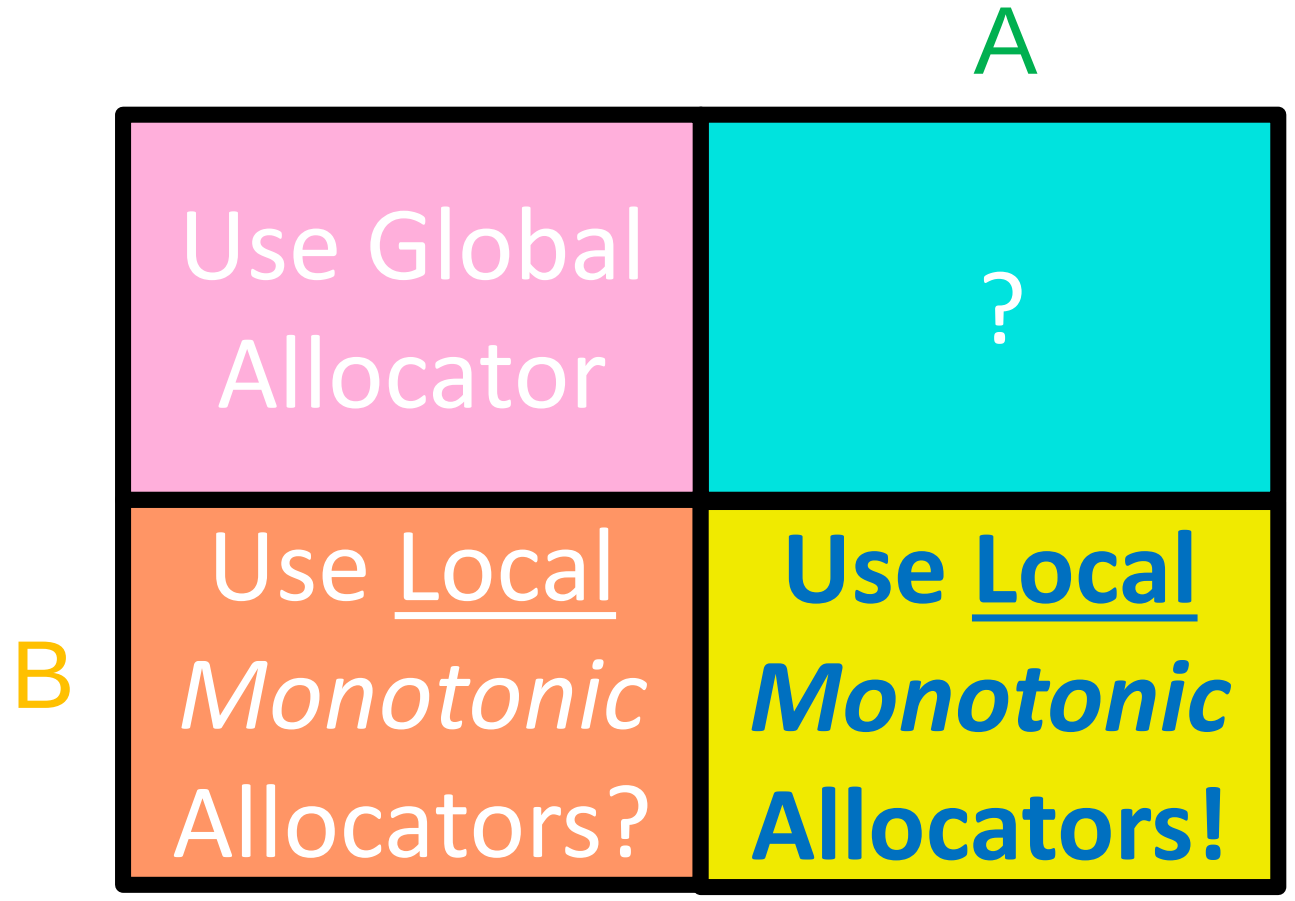
B: Subsystems exhibit high memory utilization (U).



How are they applied effectively?

A: Program is large, long running.

B: Subsystems exhibit high memory utilization (U).

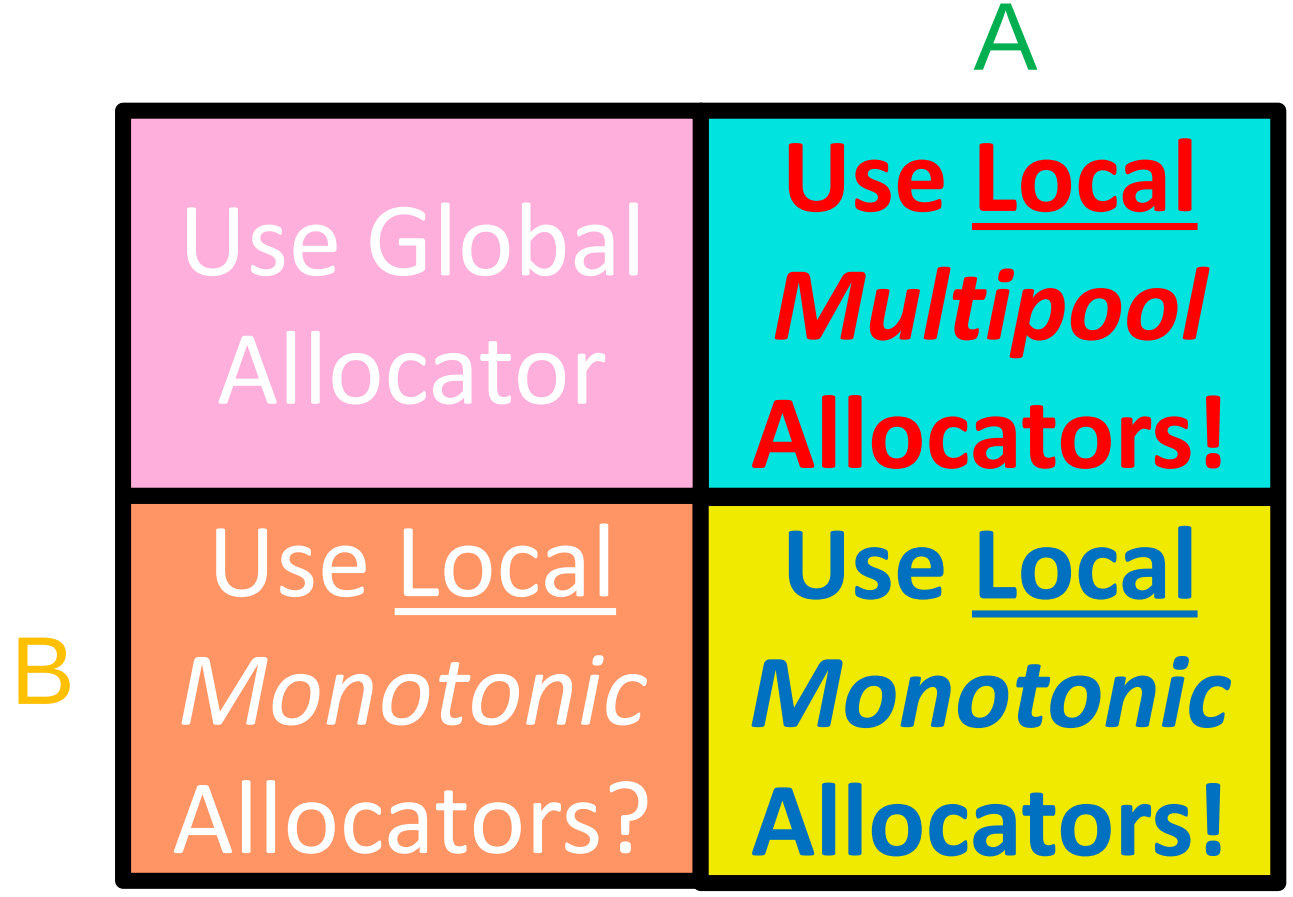


4. Conclusion

How are they applied effectively?

A: Program is large, long running.

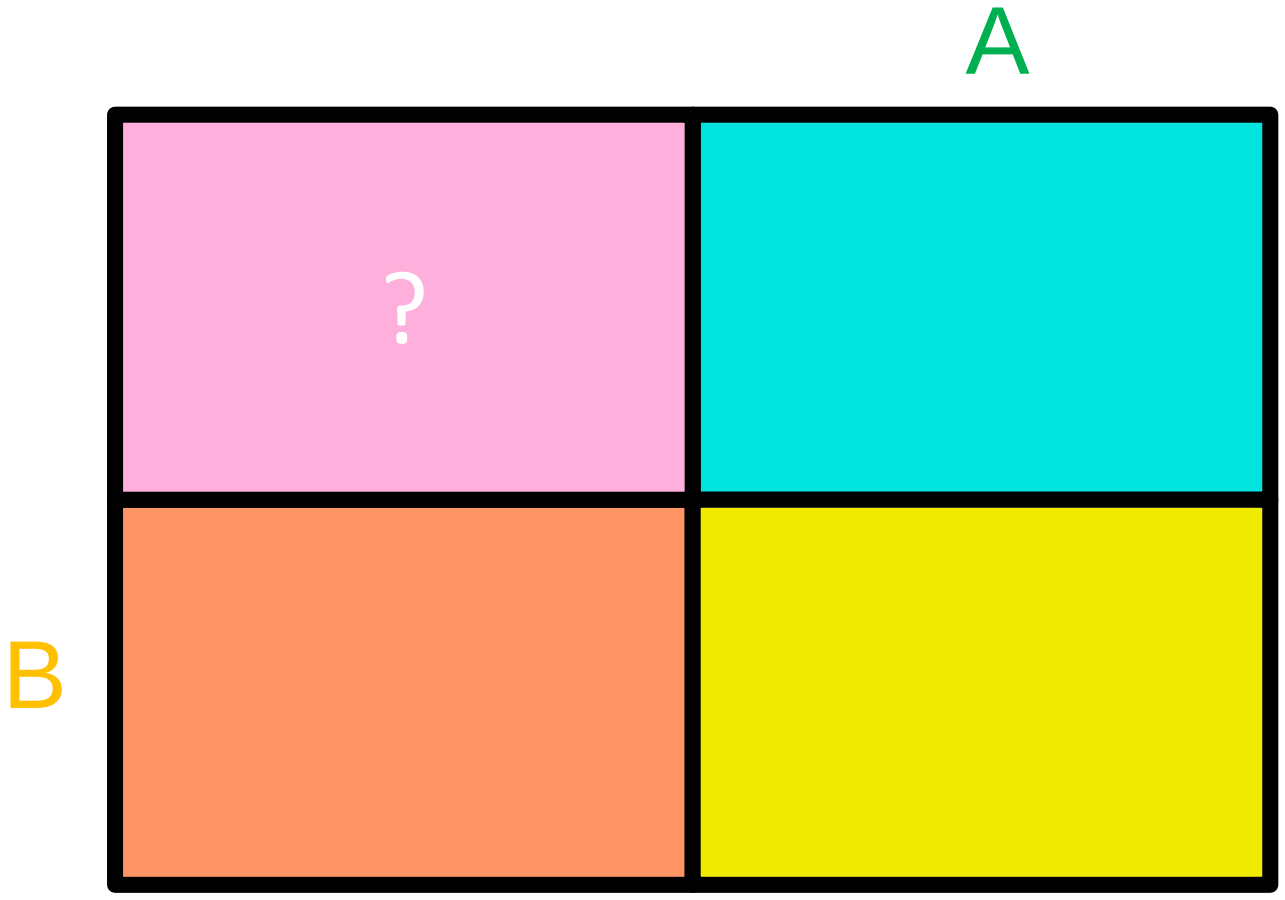
B: Subsystems exhibit high memory utilization (U).



How are they applied effectively?

A: Program is large, long running.

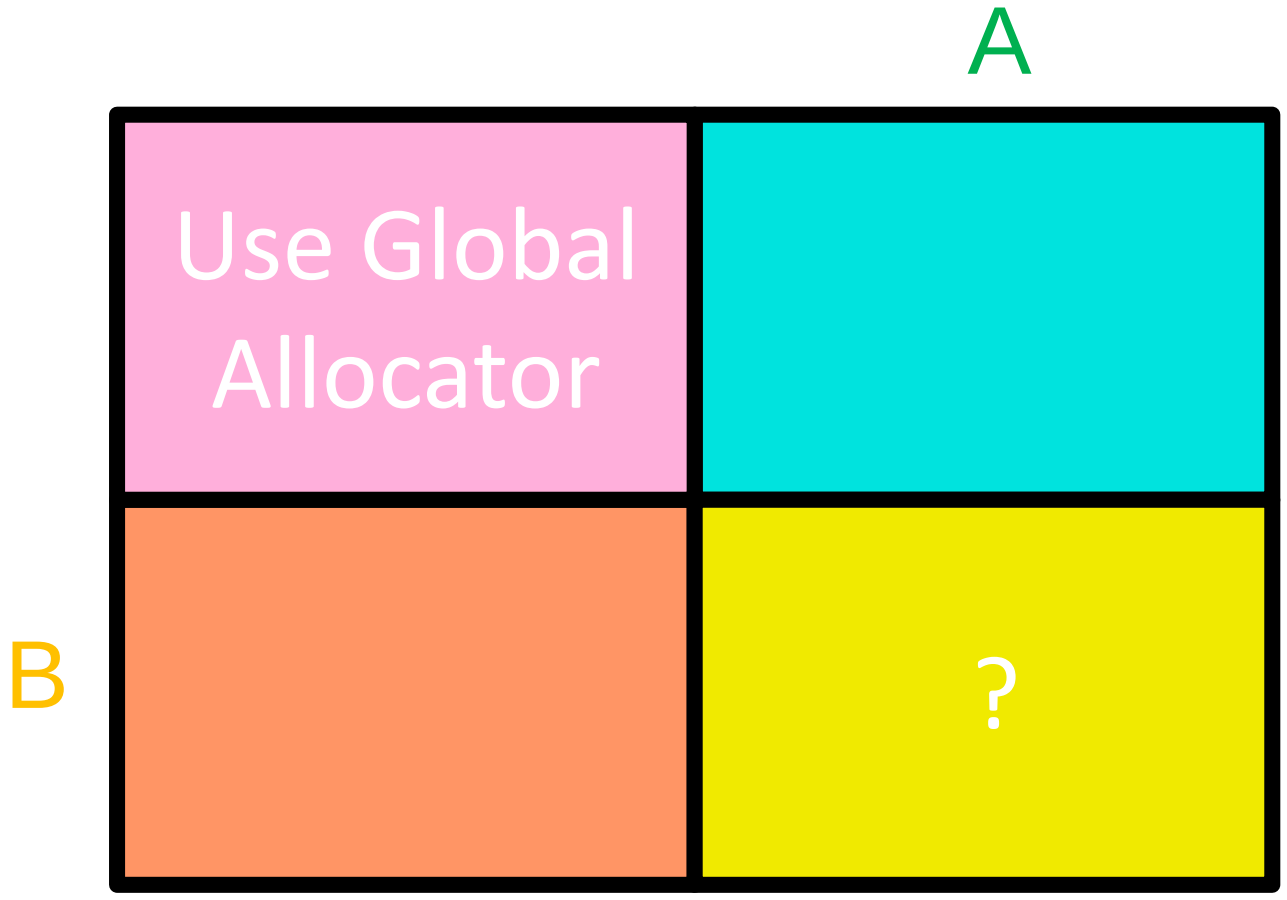
B: Subsystems run concurrently (C).



How are they applied effectively?

A: Program is large, long running.

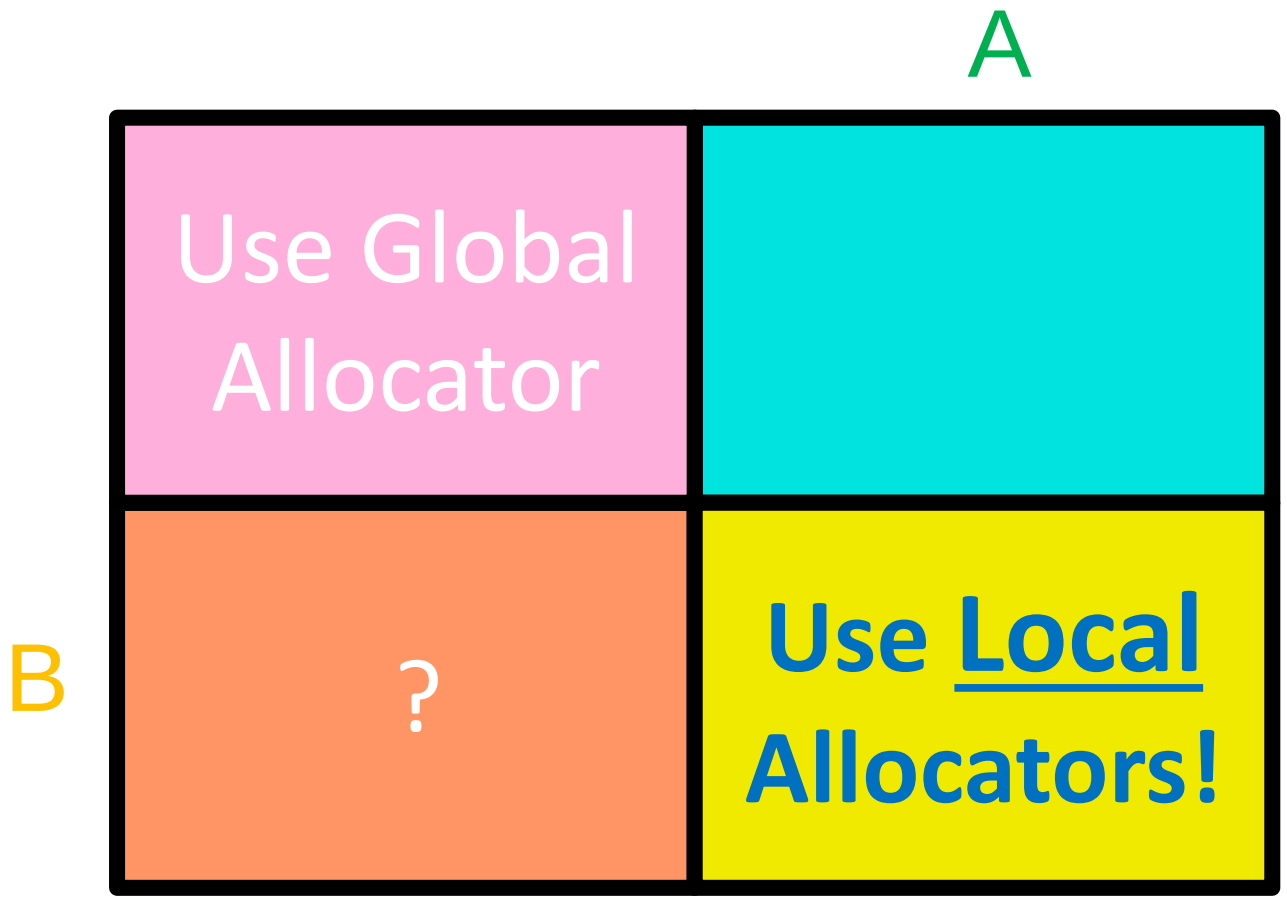
B: Subsystems run concurrently (C).



How are they applied effectively?

A: Program is large, long running.

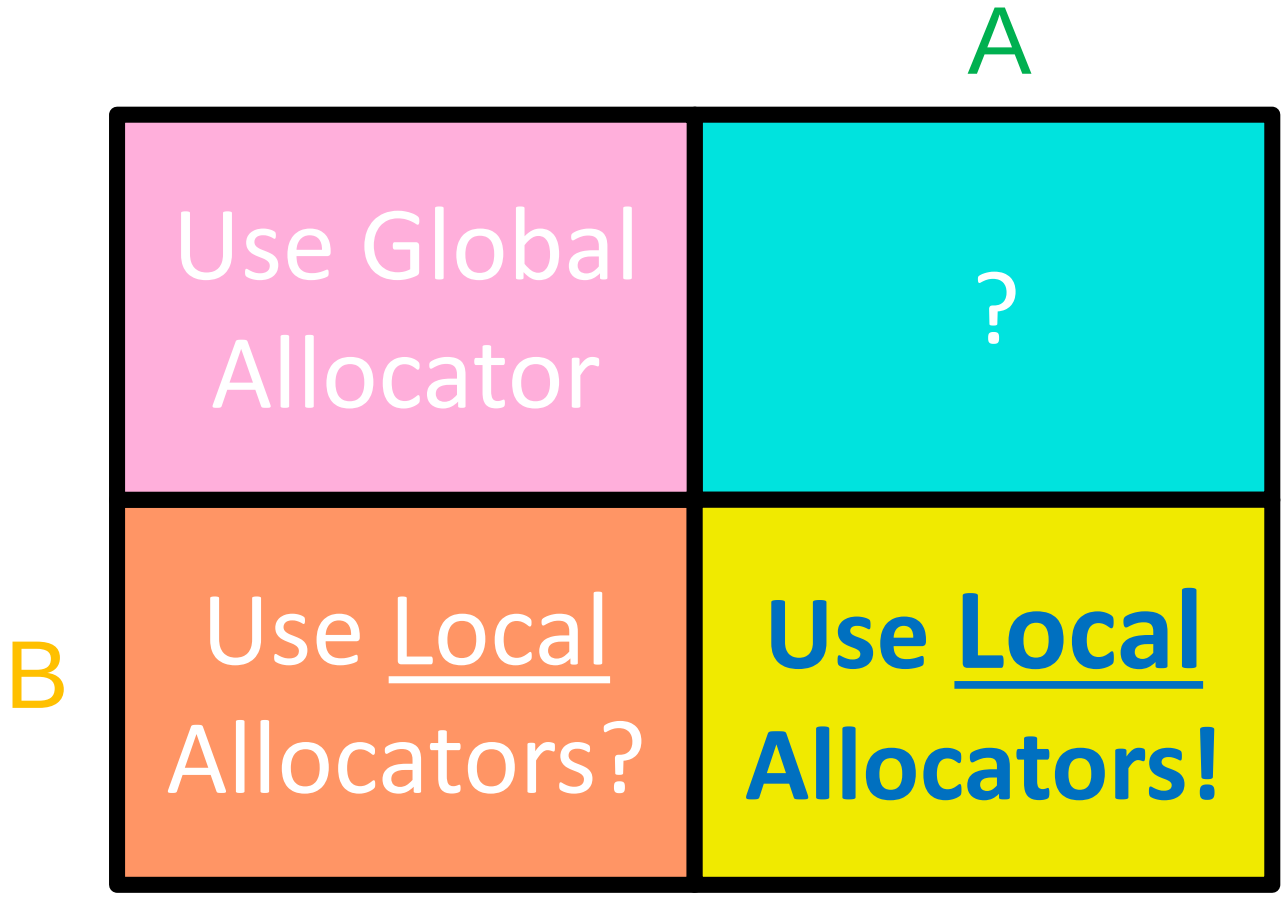
B: Subsystems run concurrently (C).



How are they applied effectively?

A: Program is large, long running.

B: Subsystems run concurrently (C).

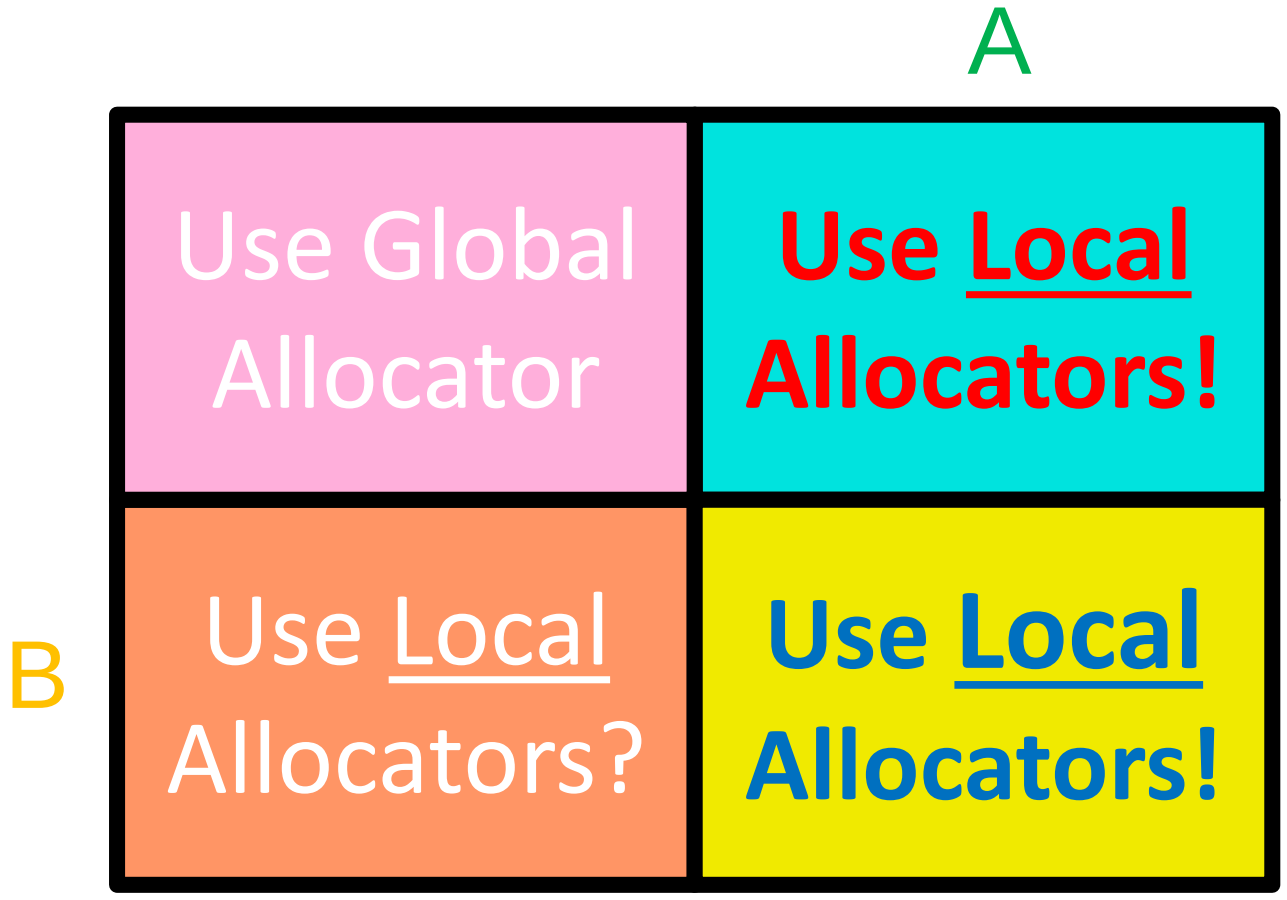


4. Conclusion

How are they applied effectively?

A: Program is large, long running.

B: Subsystems run concurrently (C).



4. Conclusion

What's the performance impact?

The performance impact can be substantial:

1. In every benchmark, use of some (if not all) of the local allocators performed *no worse*, and typically *much better* than the default allocator.
2. For long-running programs, improvements of as much as an *order of magnitude* were observed.
3. The overhead of using the virtual-function interface and (in the default case) holding an extra address was *minimal to non-existent*.

4. Conclusion

Final Thoughts

Object-level control over memory allocation is intrinsic to C++ and must always be so if we hope to maintain this language's supremacy as the best-performing high-level "systems" language.

Final Thoughts

Object-level control over memory allocation is intrinsic to C++, and must always be so if we hope to maintain this language's supremacy as the best-performing high-level "systems" language.

Supporting object-specific memory allocation is admittedly an added burden, exacerbated by an initially difficult-to-use model, which is finally being addressed in C++17 by *Polymorphic Memory Resources*.

4. Conclusion

Final Thoughts

Object-level control over memory allocation is intrinsic to C++, and must always be so if we hope to maintain this language's supremacy as the best-performing high-level "systems" language.

Supporting object-specific memory allocation is admittedly an added burden – exacerbated by an initially difficult-to-use model – which is finally being addressed in C++17 by *Polymorphic Memory Resources*. **Any future incarnation of STL should incorporate the lessons elucidated here.**

4. Conclusion

The End



4. Conclusion

The End



* Graham Bleaney

4. Conclusion

The End



* Graham Bleaney

But don't mess with the duck!

Thank you!

Visit www.TechAtBloomberg.com/cplusplus to learn more about Bloomberg's involvement with the C++ community.

Coming in late 2023:

C++ Allocators for the Working Programmer

ISBN: 9780138060725

Bloomberg Engineering

TechAtBloomberg.com

© 2022 Bloomberg Finance L.P. All rights reserved.


CPP-Summit 2022
Thank you

CPP-Summit 2022



温昱

Boolan 首席咨询师

国内最早研究软件架构的人之一，20 年以上软件架构和设计咨询与研发经验，代表著作《软件架构设计》《一线架构师实践指南》，累计印刷 40 余次。在架构设计、领域建模、OO 设计、质量设计等多方面的 原创方法论或技巧，包括:VSP 分析法，ARCT 设计重构方法论，嵌入式 ProductFamily 需求框架，嵌入式设备平台方法论，ADMEMS 架构实践体系等。长期担任软件咨询师，拥有丰富的军工电子、航电、电信、软件等从业经验。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

嵌入式软件产品化架构

温 昱
Boolan顾问

议程

嵌软产品化研究

1.1 产品化的诱惑

1.2 产品化的难点

1.3 可修改性设计

嵌软产品化案例

2.1 功能与技术分离

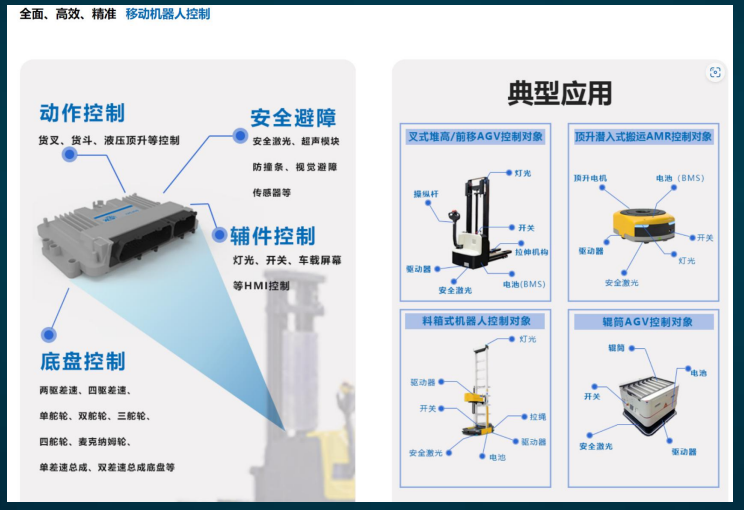
2.2 隔离硬件变化

2.3 隔离领域变化

2.4 框架化平台化（可选）

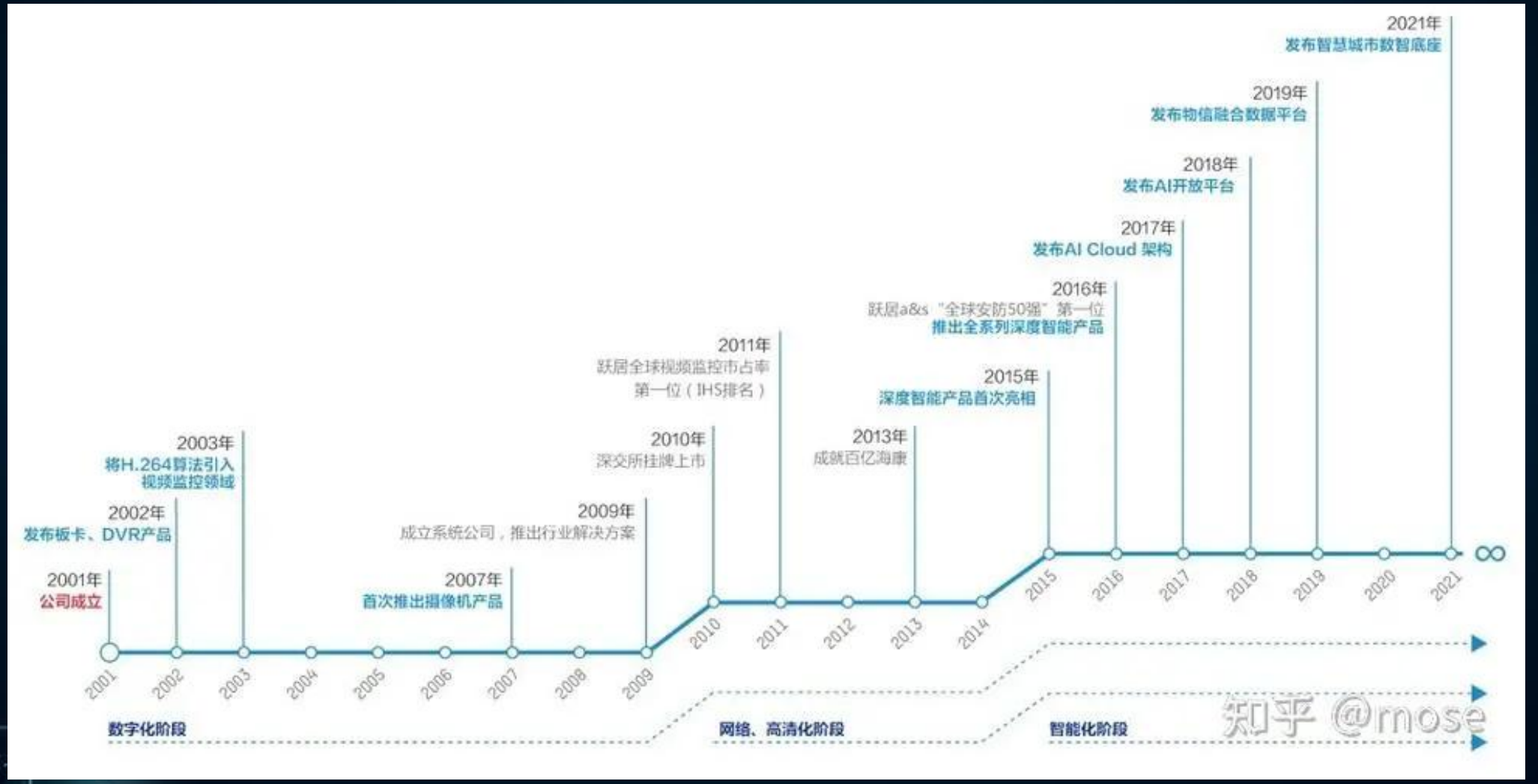
产品化成就头部企业


性能
H3U
高性能型
H2U
通用型
H1U
经济型
H0U
简单逻辑型
点数
14
32
40
64
80
128
256
1024
www.inovance.cn

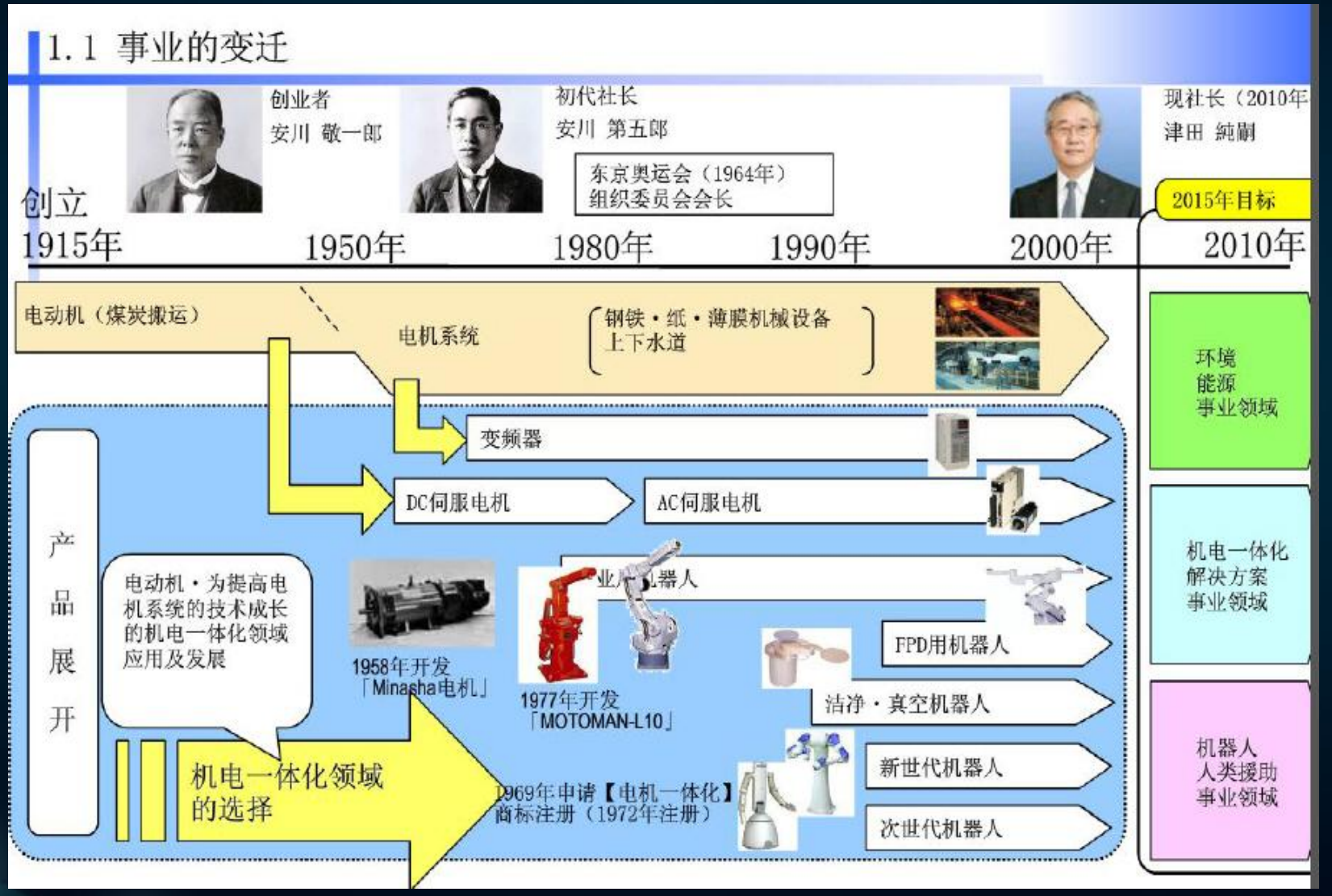

全面、高效、精准 移动机器人控制
动作控制
货叉、货斗、液压顶升等控制
安全避障
安全激光、超声模块
防撞条、视觉避障
传感器等
辅件控制
灯光、开关、车载屏幕
等HMI控制
底盘控制
两驱差速、四驱差速、
单舵轮、双舵轮、三舵轮、
四舵轮、麦克纳姆轮、
单差速总成、双差速总成底盘等
典型应用
叉式堆高/前移AGV控制对象
顶升潜入式搬运AMR控制对象
料箱式机器人控制对象
辊筒AGV控制对象


十五管（双出线）
十五管（单出线）
十二管（单出线）
十二管（单出线）
九 管
大六管
中六管
小蜜蜂
小锂电
3650
2430
双系统平衡车

海康：做器件→做整机→做总包



安川：做器件→做整机→做总包



议程

嵌软产品化研究

1.1 产品化的诱惑

1.2 产品化的难点

1.3 可修改性设计

嵌软产品化案例

2.1 功能与技术分离

2.2 隔离硬件变化

2.3 隔离领域变化

2.4 框架化平台化（可选）

产品化架构：包容两种变化

产品化架构是为“相同或相近领域内一组产品”提供的统一架构设计：

1）目的上，是为推出“硬件配置与功能特性有较大差异”的系列产品，覆盖不同目标市场；

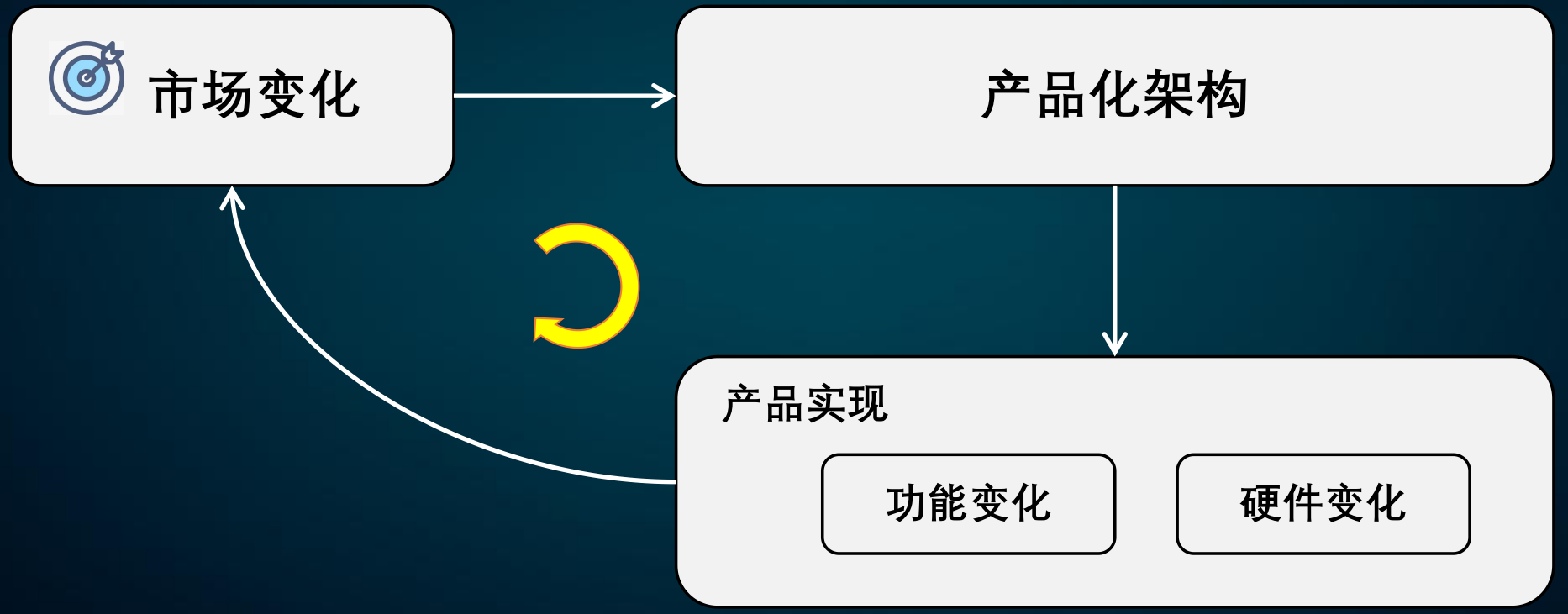
2）目的上，也支持基于产品基线版本，进行特性差异化定制，满足不同客户的个性化需求；

3）特点上，产品化架构具有可重用、可配置、可插拔、可扩展等较高要求；

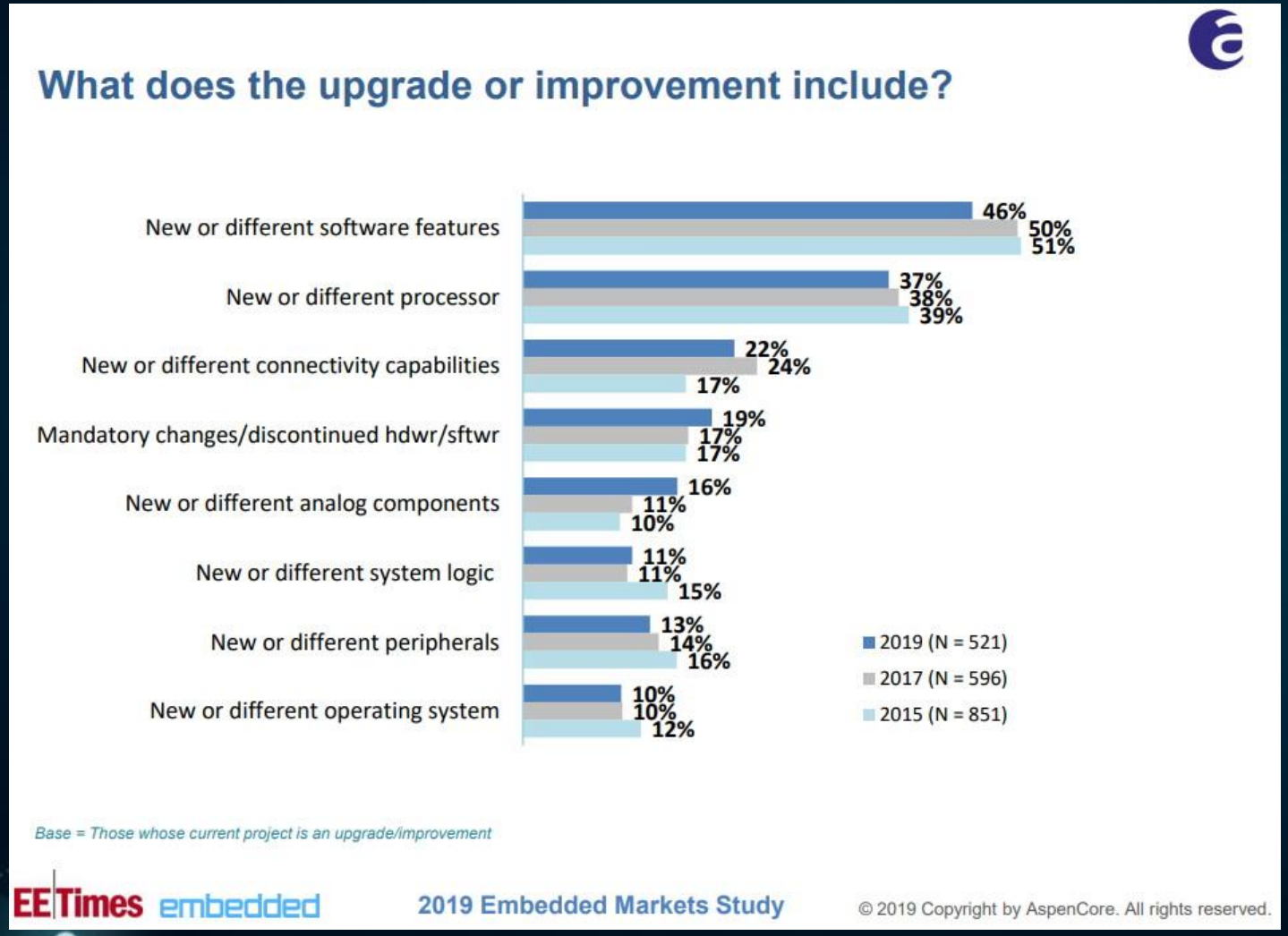
4）战略上，要支撑中长期的市场战略规划、产品定位；

5）管理上，要适应中短期的产品路线图、产品规划。

产品化架构：包容两种变化



需求变化压力



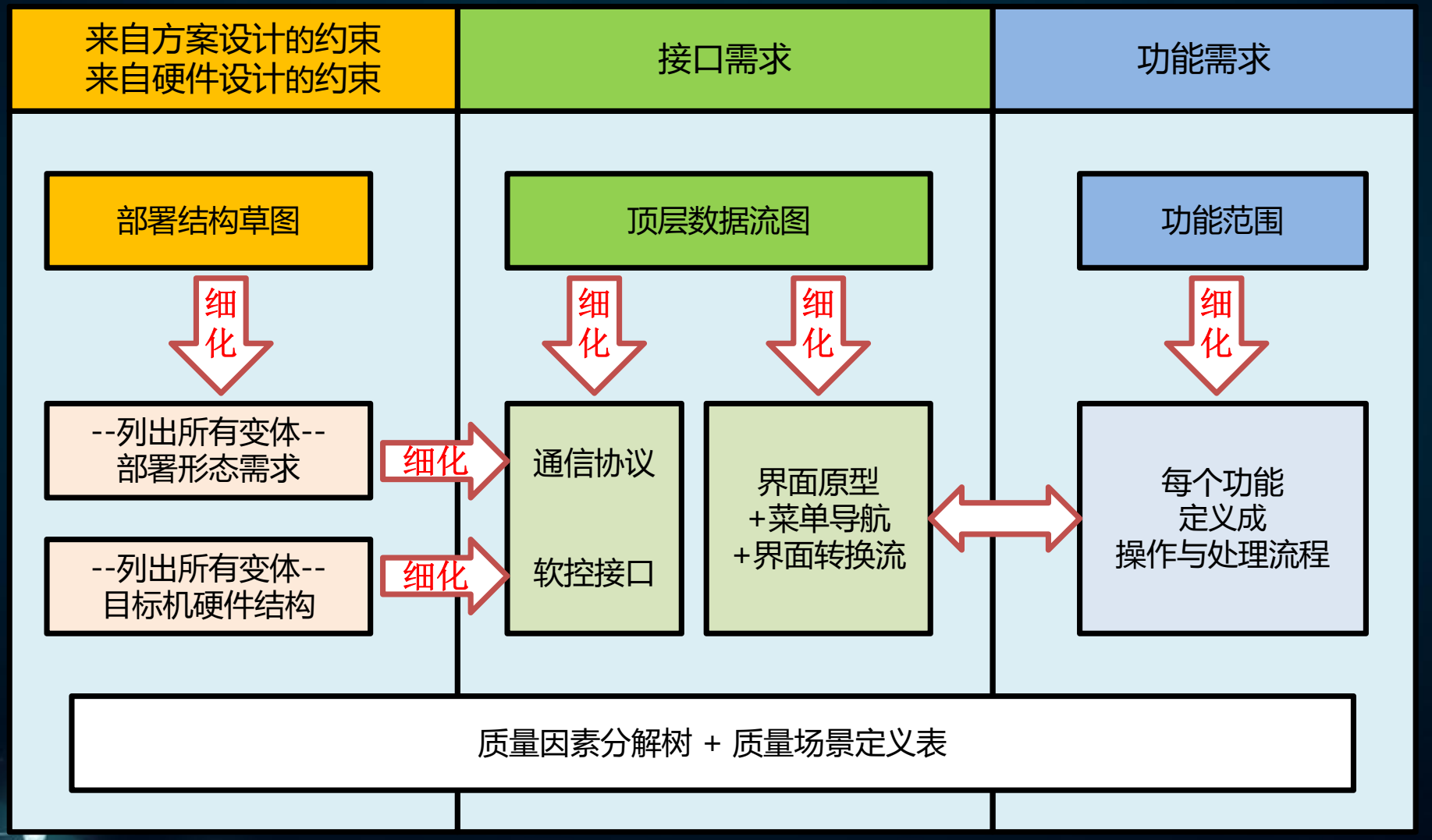
【版权】

EE Times 电子行业全球著名媒体

2019年报告

需求变化模型

（温昱版权所有）



议程

嵌软产品化研究

1.1 产品化的诱惑

1.2 产品化的难点

1.3 可修改性设计

嵌软产品化案例

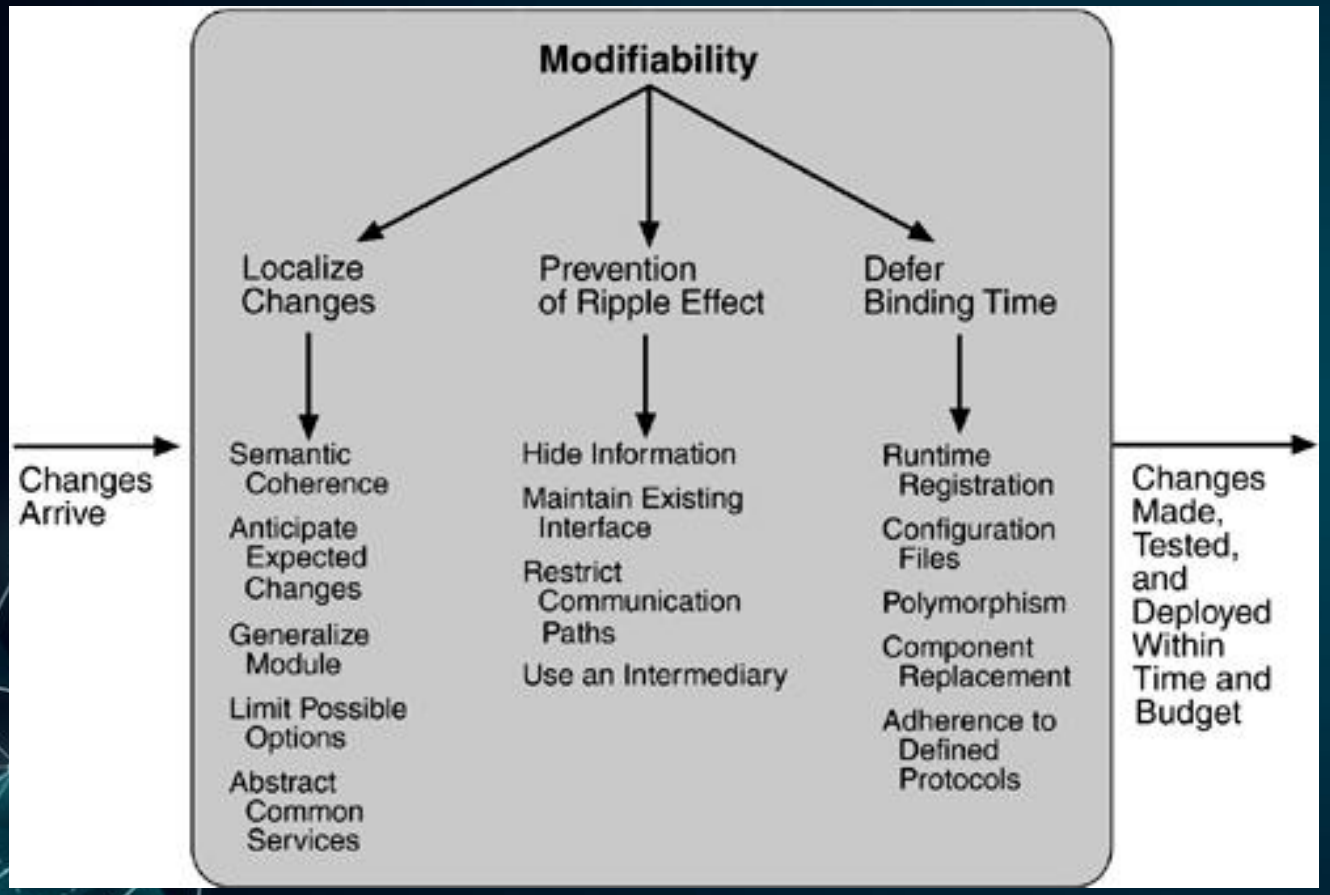
2.1 功能与技术分离

2.2 隔离硬件变化

2.3 隔离领域变化

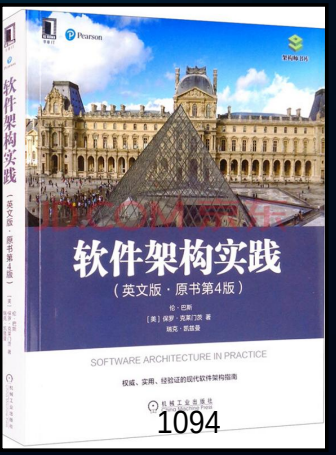
2.4 框架化平台化（可选）

可修改性设计



Many tactics are intended to have impact at loadtime or runtime, such as the following.

- *Runtime registration* supports plug-and-play operation at the cost of additional overhead to manage the registration. Publish/subscribe registration, for example, can be implemented at either runtime or load time.
- *Configuration files* are intended to set parameters at startup.
- *Polymorphism* allows late binding of method calls.
- *Component replacement* allows load time binding.
- *Adherence to defined protocols* allows runtime binding of independent processes.



议程

嵌软产品化研究

1.1 产品化的诱惑

1.2 产品化的难点

1.3 可修改性设计

嵌软产品化案例

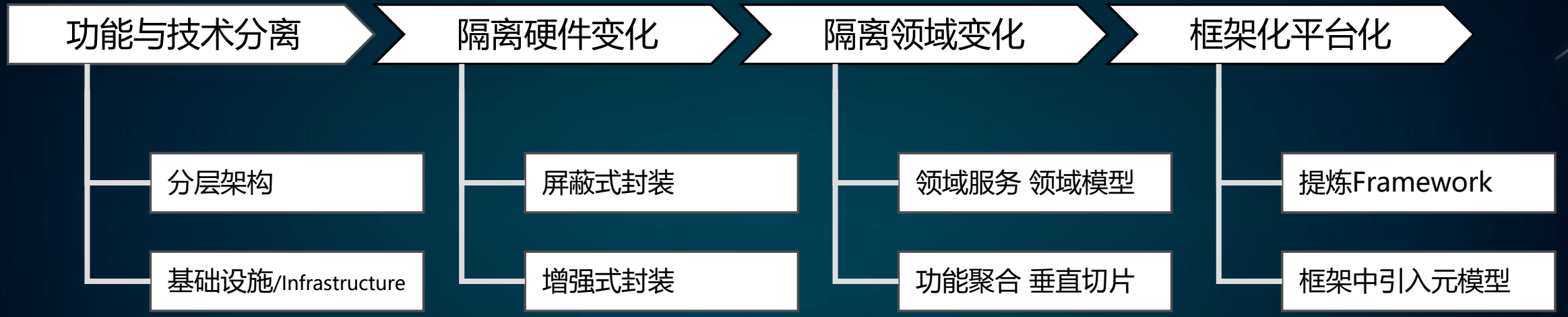
2.1 功能与技术分离

2.2 隔离硬件变化

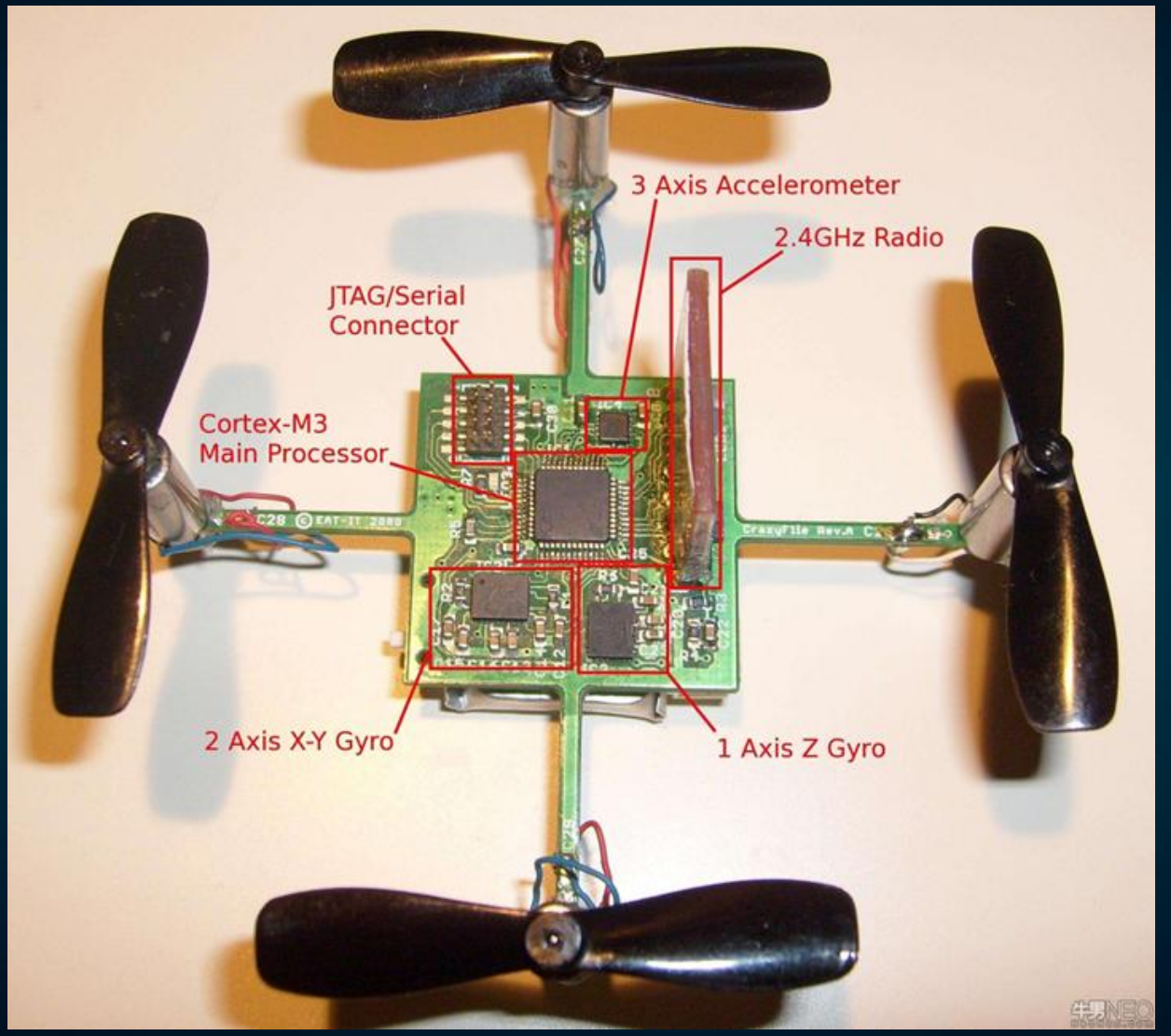
2.3 隔离领域变化

2.4 框架化平台化（可选）

四原则、八技巧、注意节奏


功能与技术分离
隔离硬件变化
隔离领域变化
框架化平台化
分层架构
基础设施/Infrastructure
屏蔽式封装
增强式封装
领域服务 领域模型
功能聚合 垂直切片
提炼Framework
框架中引入元模型

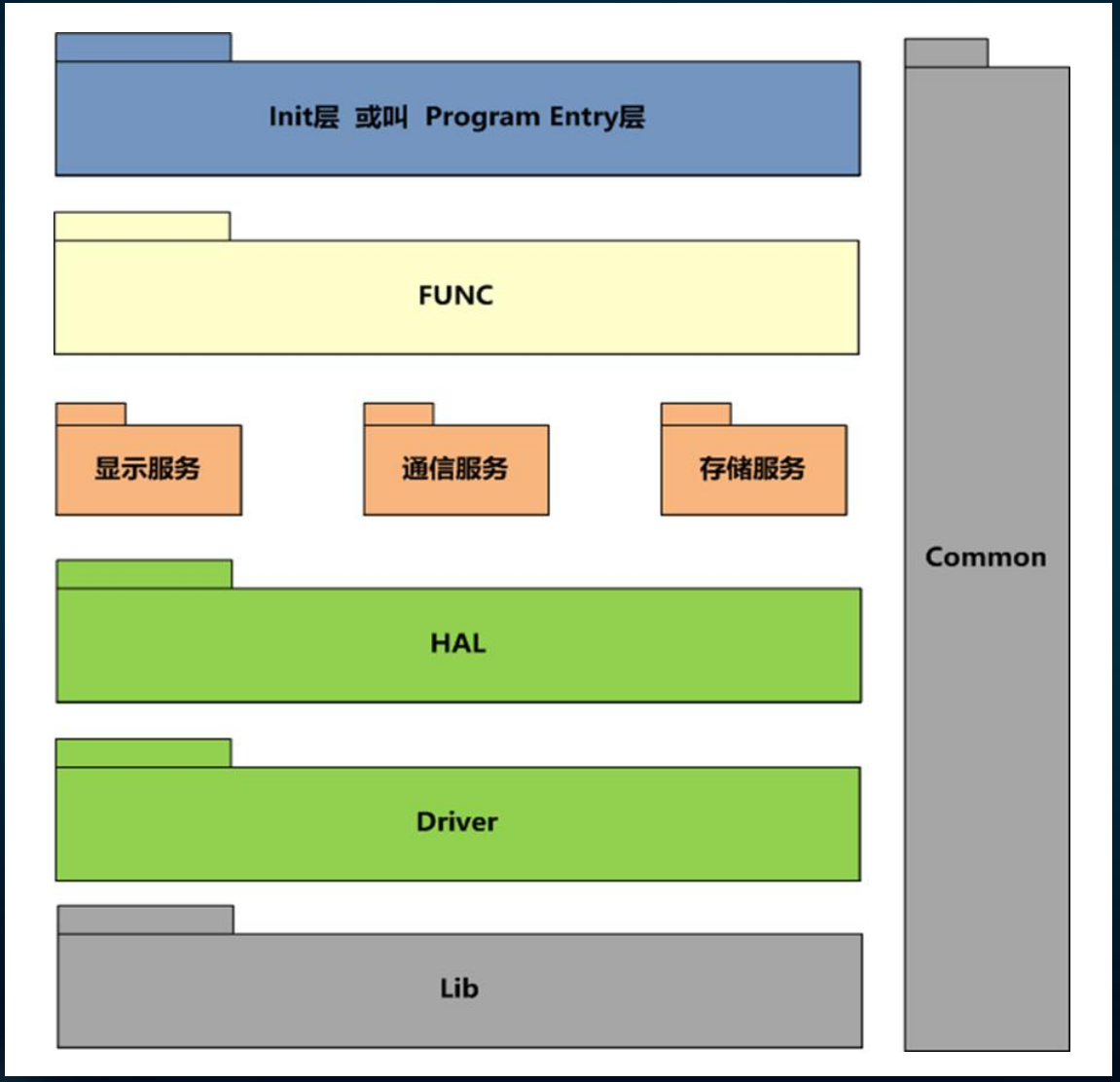
案例 1


3 Axis Accelerometer
2.4GHz Radio
JTAG/Serial
Connector
Cortex-M3
Main Processor
2 Axis X-Y Gyro
1 Axis Z Gyro

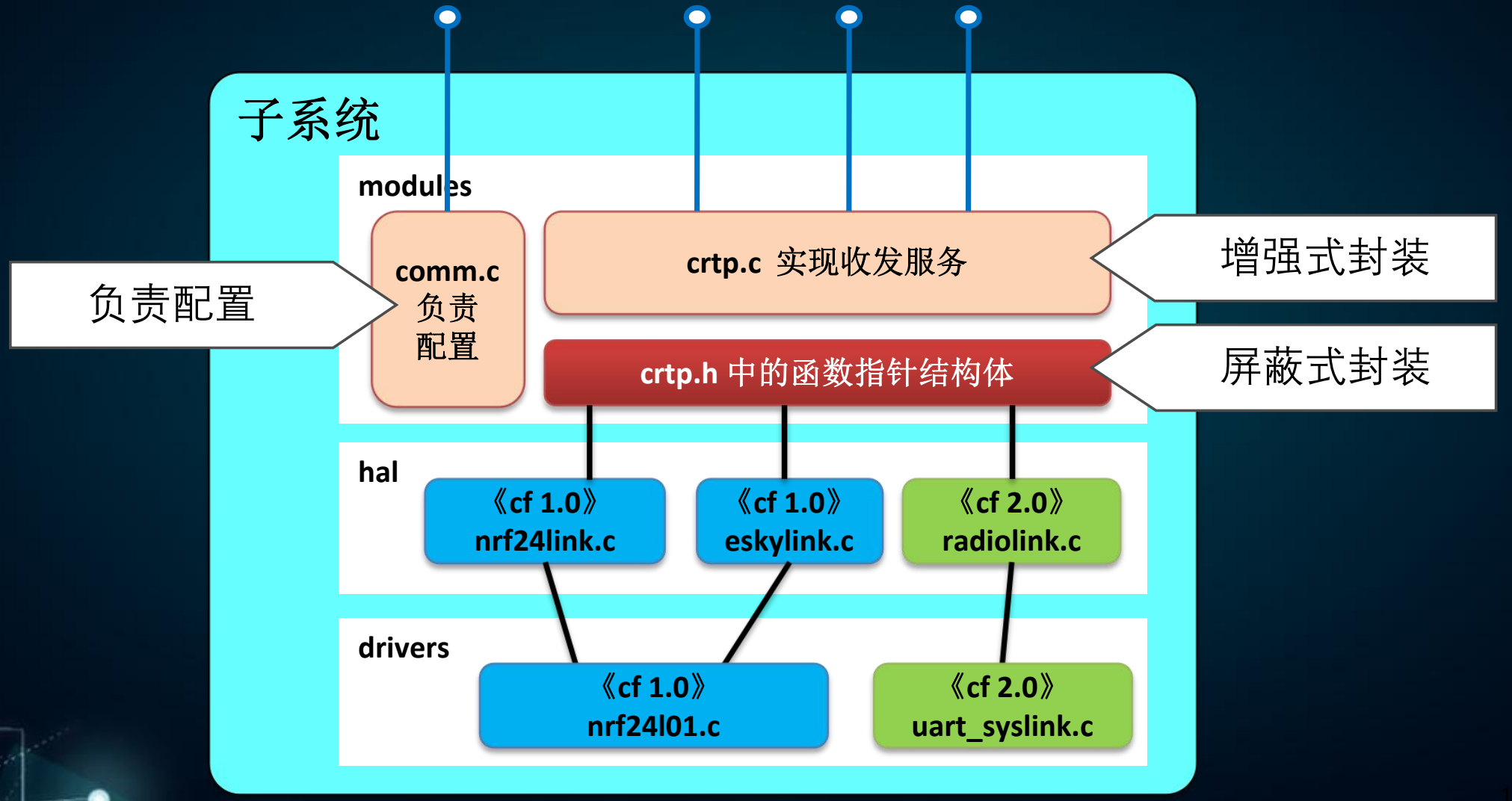
嵌软架构大类别

	1	2	3
架构	Time-Slice Polling + ISR	多任务 + ISR	多进程+多线程
OS	无OS	RTOS	Linux、Windows
硬件	MCU	MUC	CPU/工控机/专用机
语言	C	C	C++
代码量	几万~十几万行代码	几万~几十万行代码	几十万~几千万行代码

功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化



功能与技术分离 | 隔离硬件变化 | 隔离领域变化 | 框架化平台化



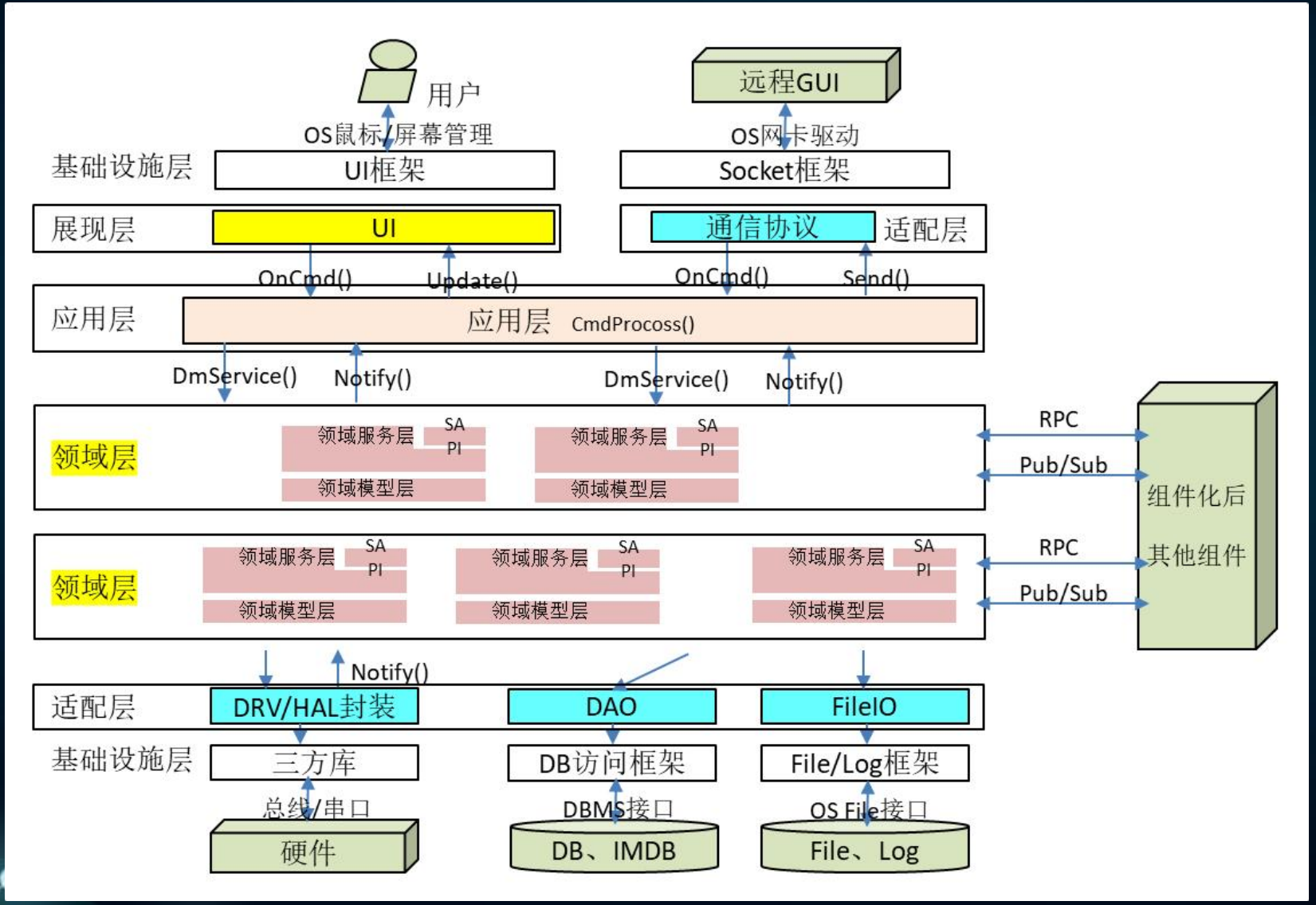
案例 2


送板
錫膏印刷
SPI
貼片
人工目檢
AOI
Reflow
知乎 @深圳优勤电子

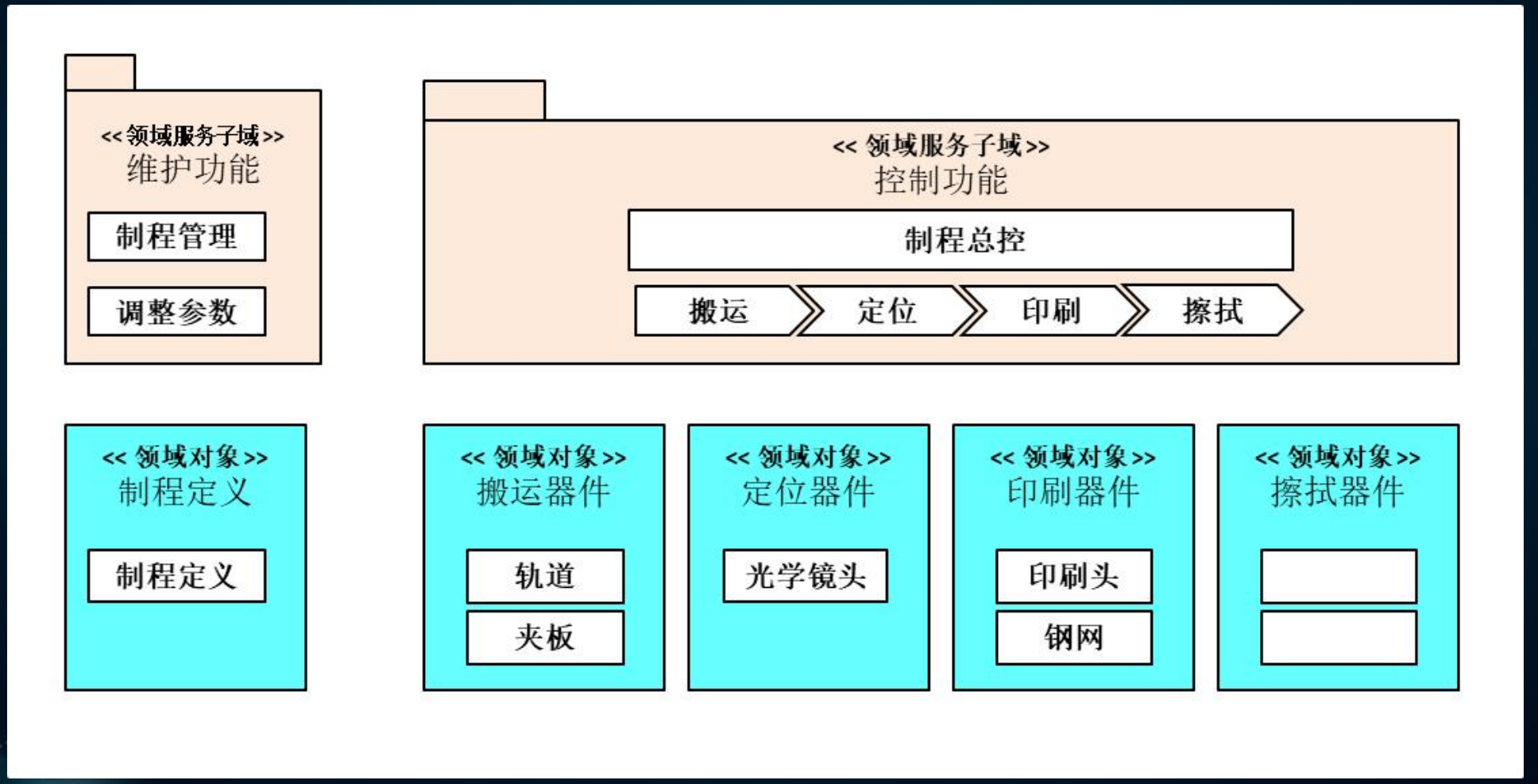
嵌软架构大类别

	1	2	3
架构	Time-Slice Polling + ISR	多任务 + ISR	多进程+多线程
OS	无OS	RTOS	Linux、Windows
硬件	MCU	MUC	CPU/工控机/专用机
语言	C	C	C++
代码量	几万~十几万行代码	几万~几十万行代码	几十万~几千万行代码

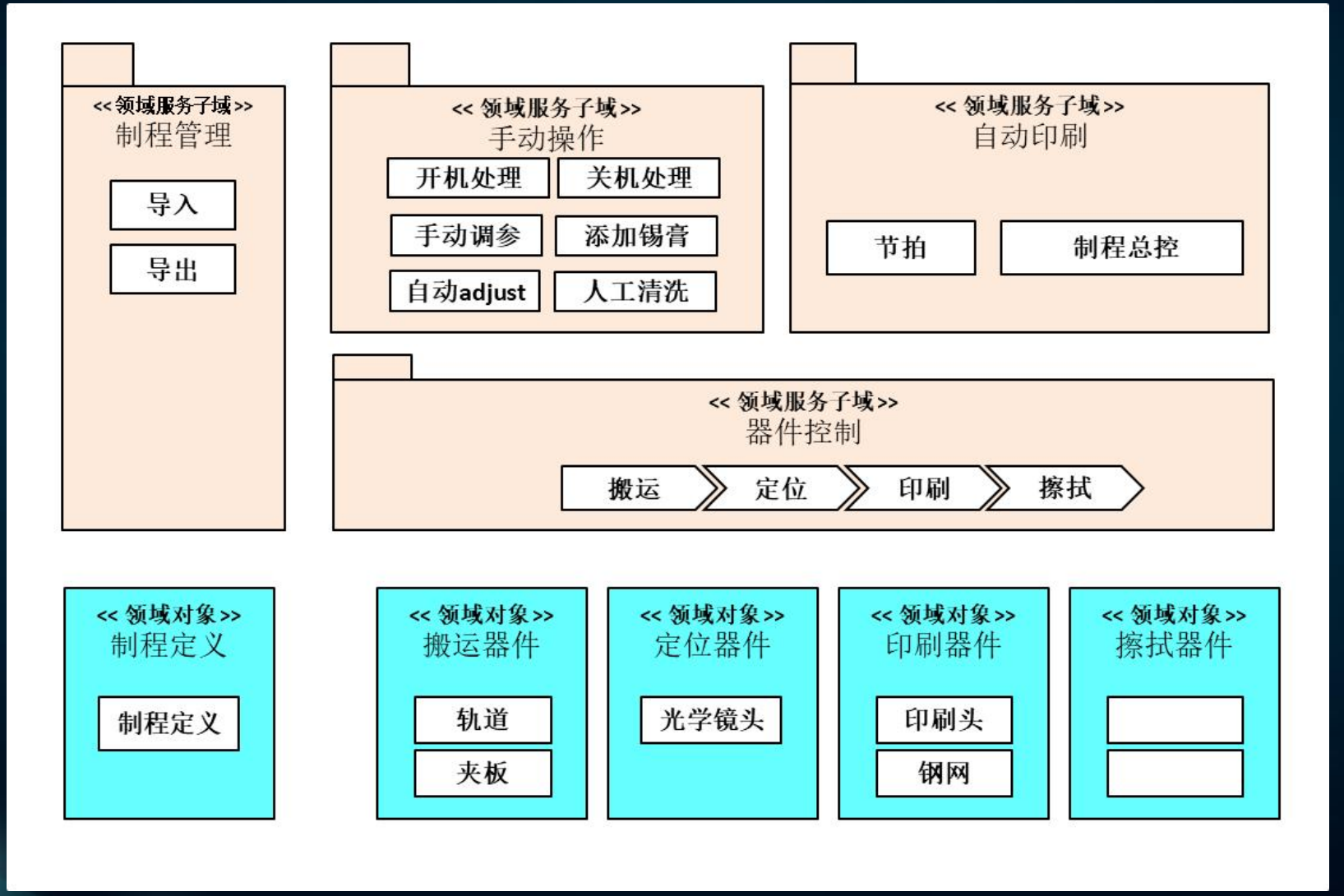
功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化



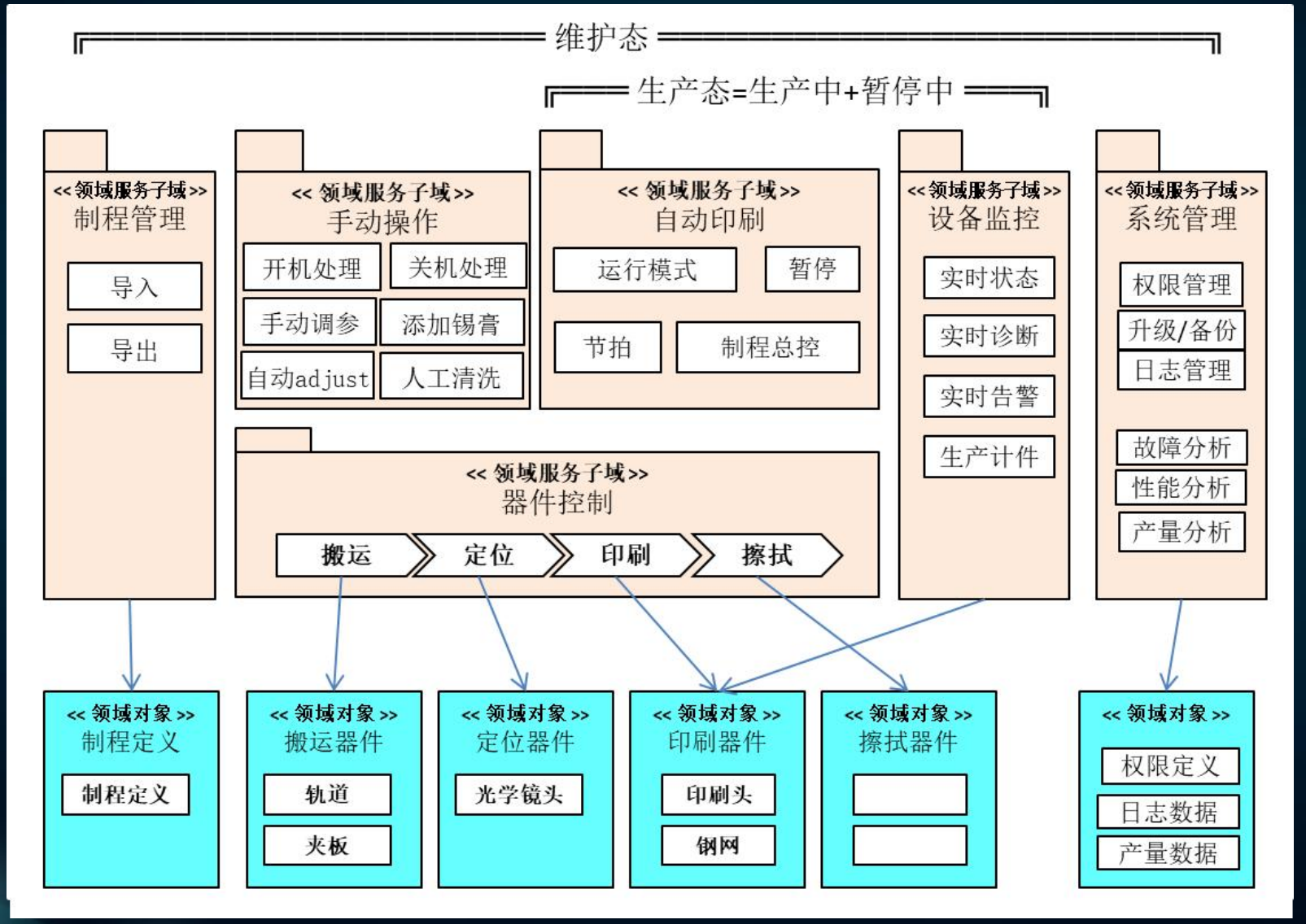
功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化



功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化

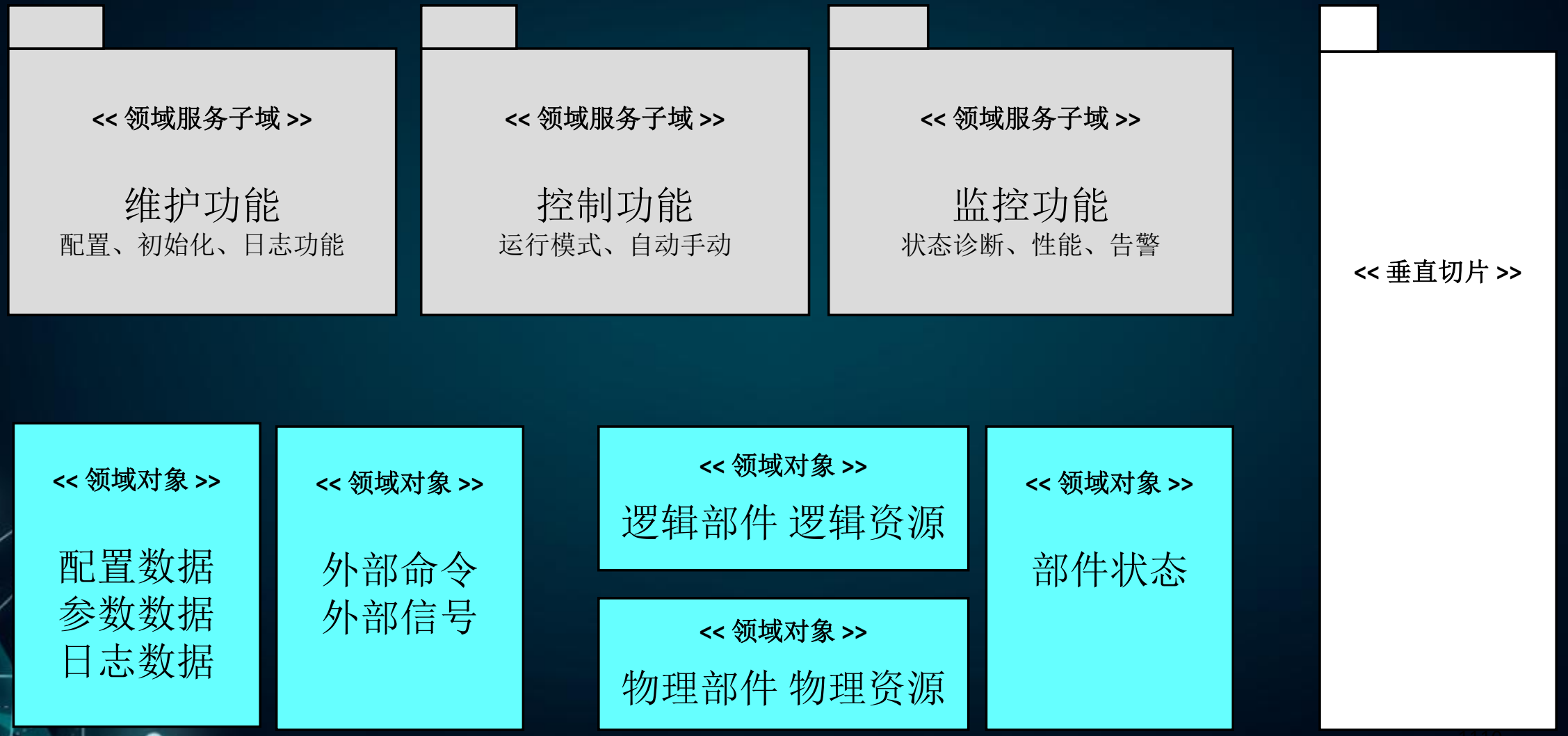


功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化

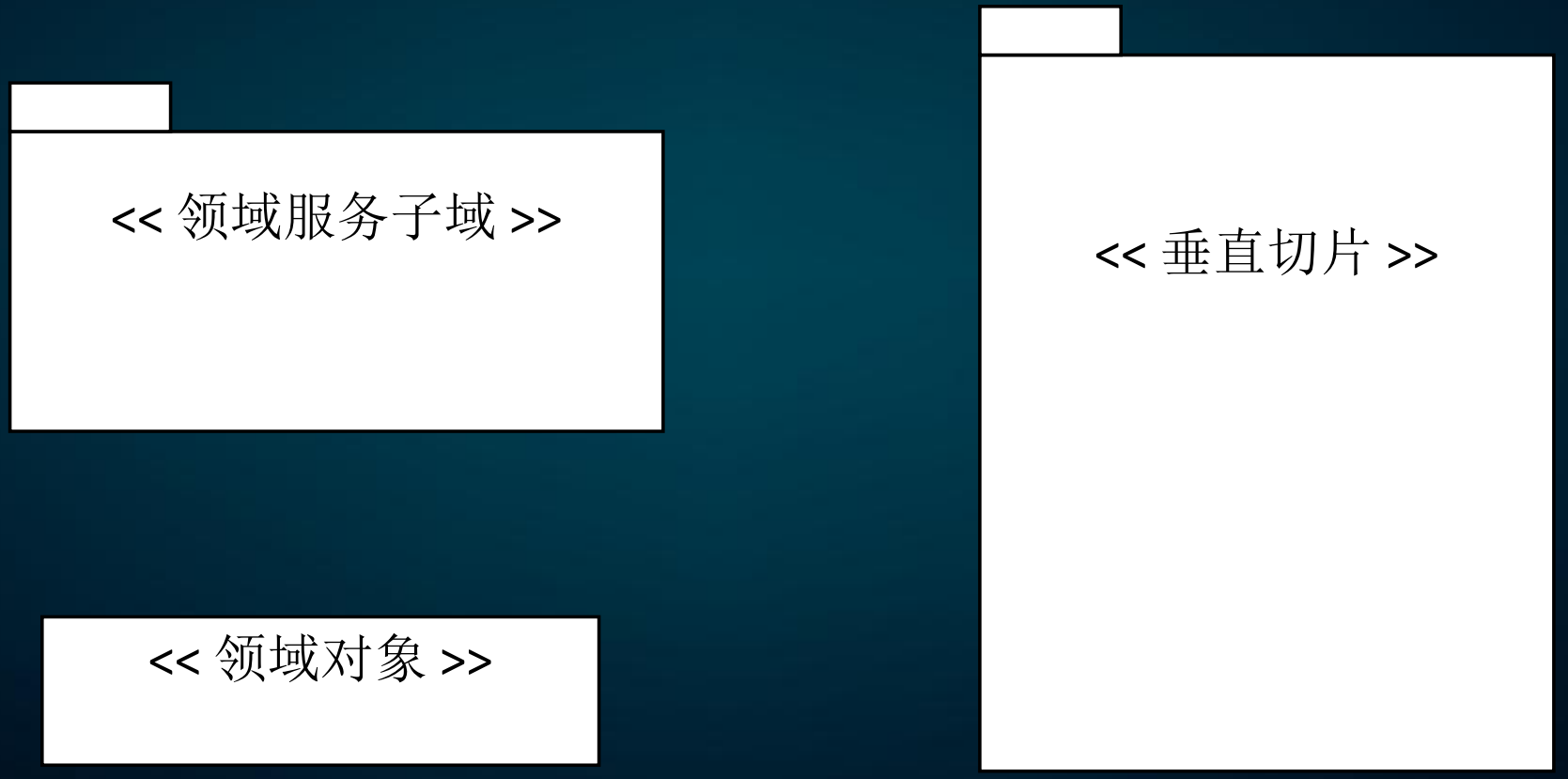


案例 3

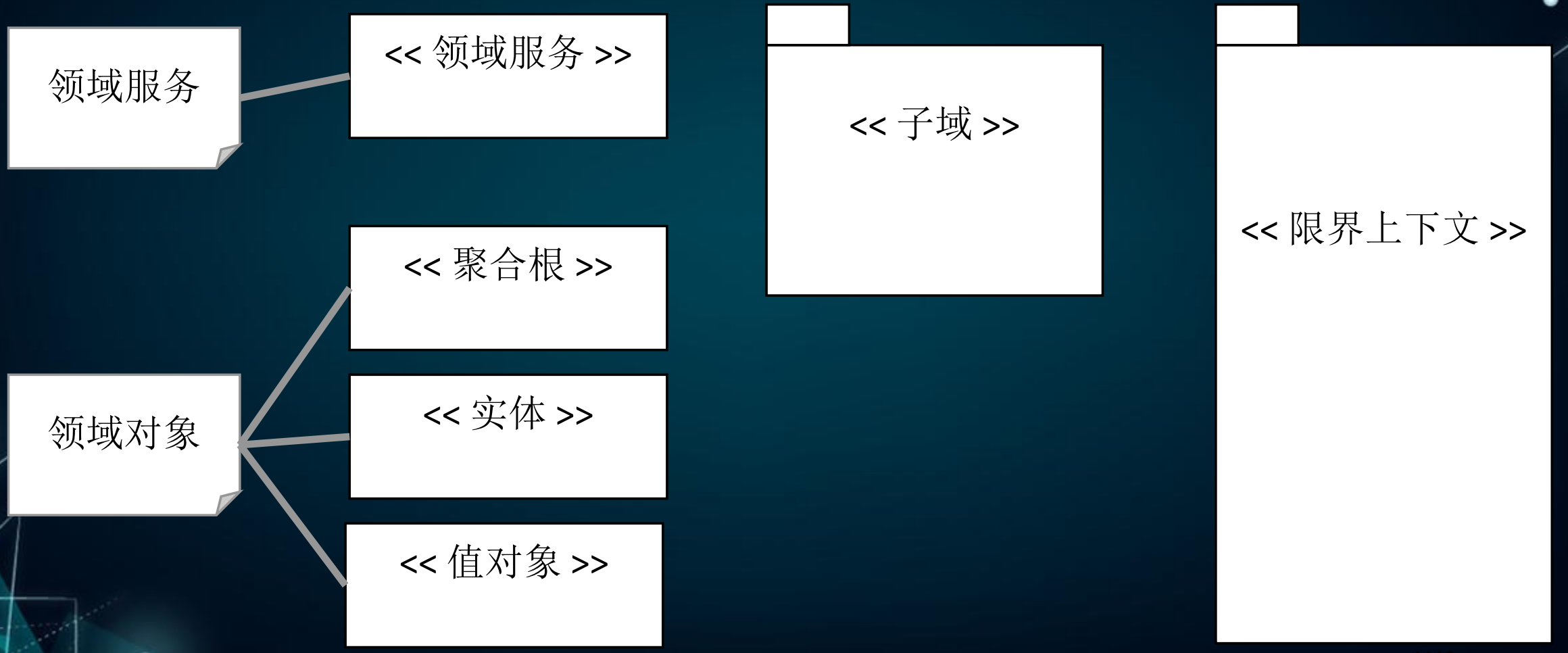
温昱嵌入式DDD规范 带参考模型

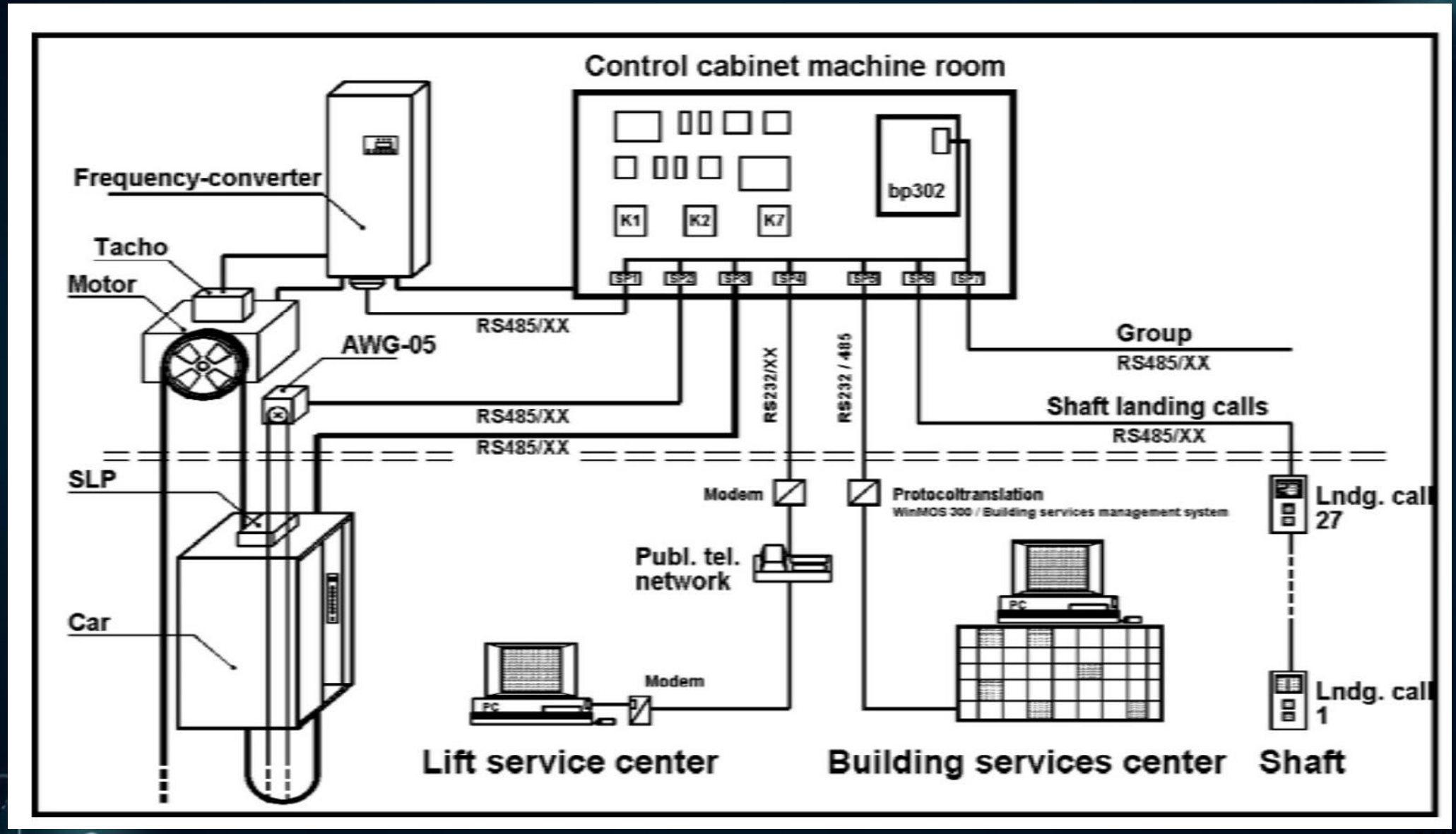


温昱DDD规范

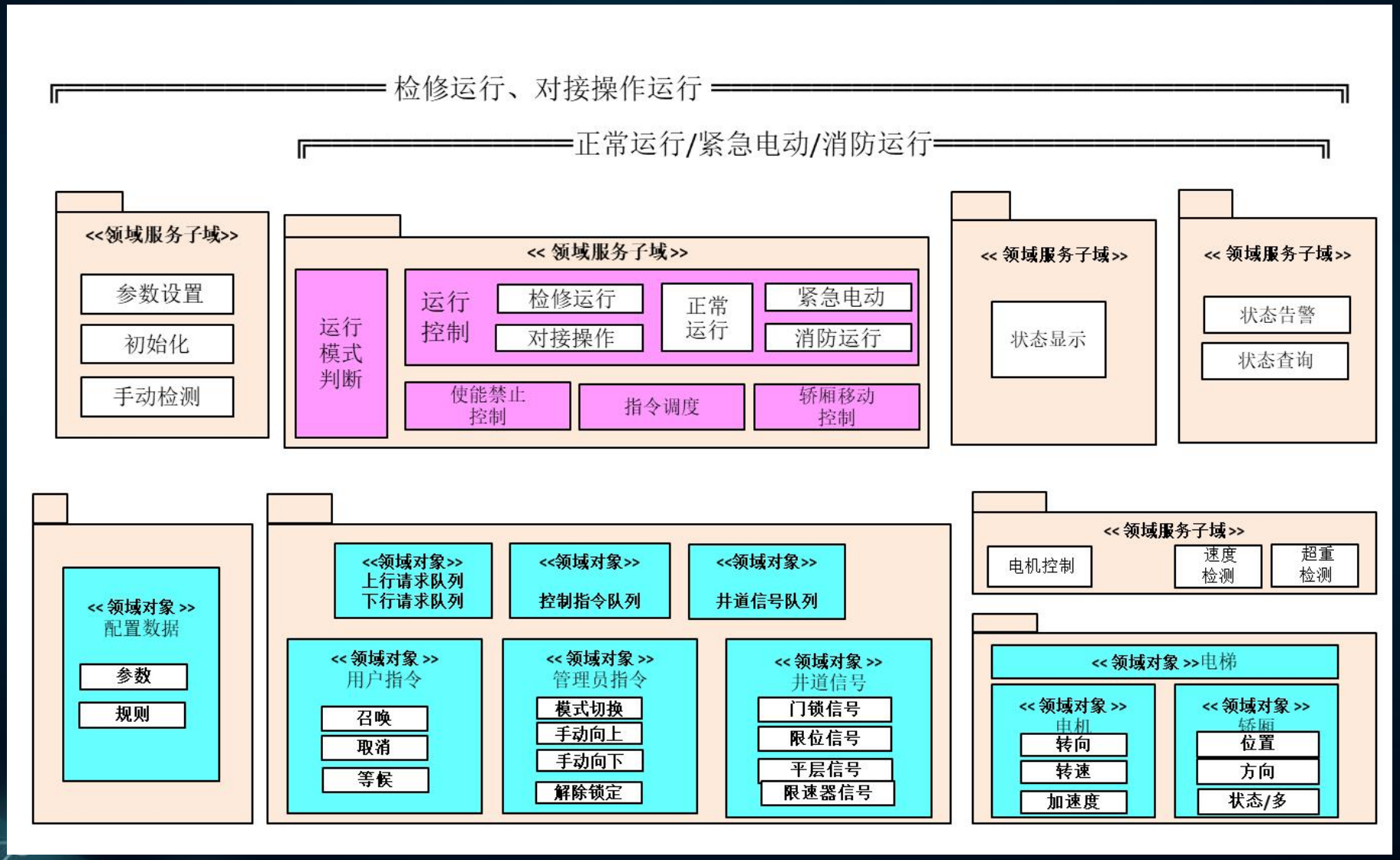


原装DDD规范

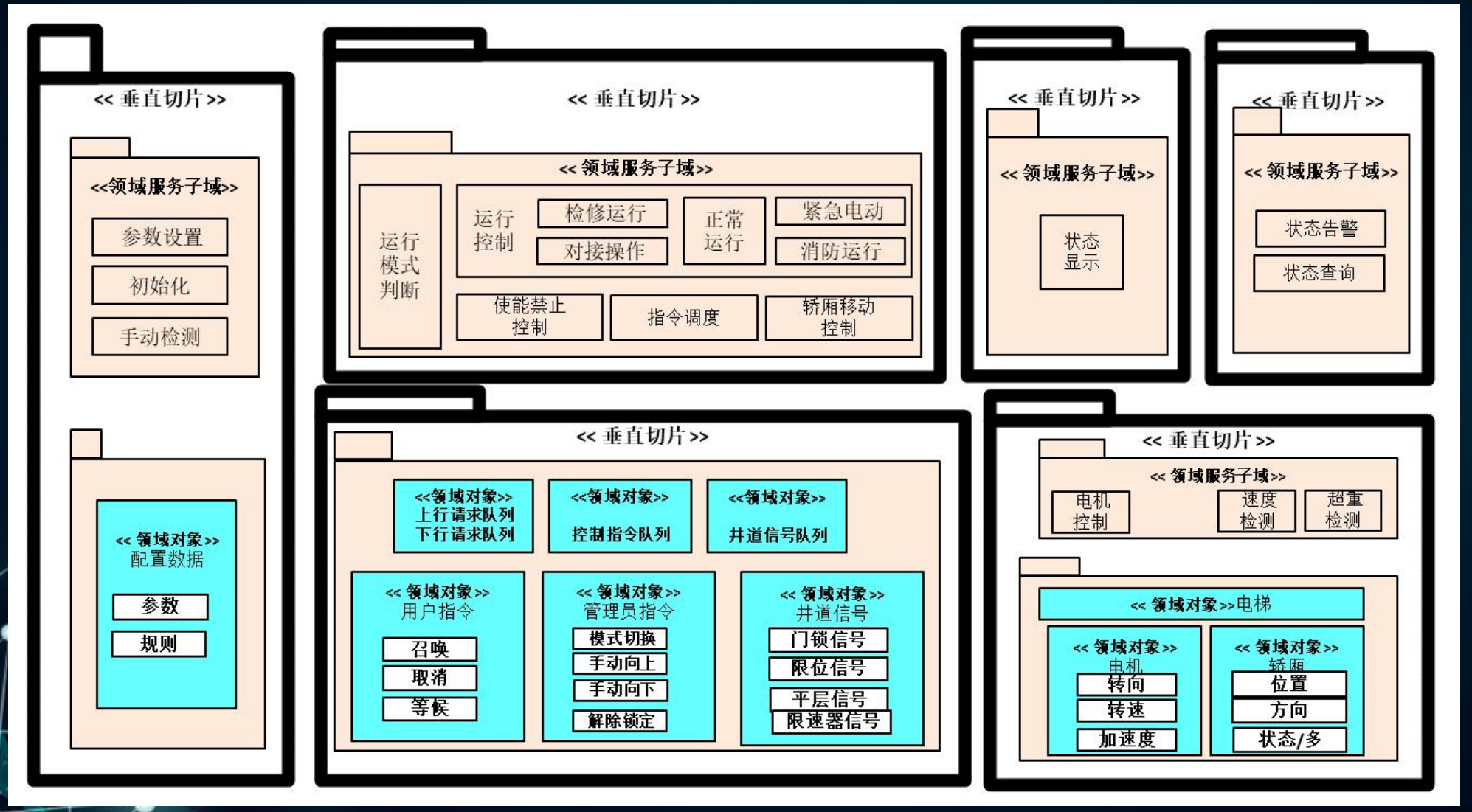



Control cabinet machine room
Frequency-converter
Tacho
Motor
AWG-05
SLP
Car
bp302
K1
K2
K7
SP1
SP2
SP3
SP4
SP5
SP6
SP7
RS485/XX
RS485/XX
RS485/XX
RS232/XX
RS232 / 485
Group
RS485/XX
Shaft landing calls
RS485/XX
Modem
Protocoltranslation
WinMOS 300 / Building services management system
Publ. tel. network
Modem
PC
PC
Lndg. call 27
Lndg. call 1
Lift service center
Building services center
Shaft

功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化



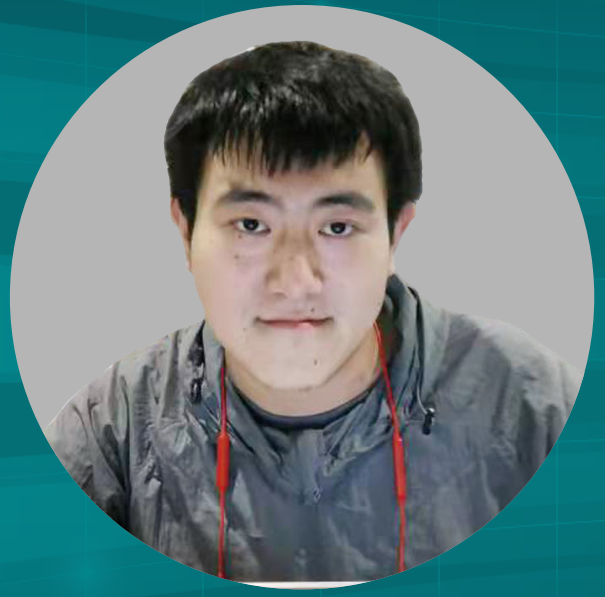
功能与技术分离 → 隔离硬件变化 → 隔离领域变化 → 框架化平台化



谢谢观看

CPP-Summit 2022

许传奇



阿里巴巴开发工程师， LLVM Committer， C++ 标准委员会成员

阿里云程序语言与编译器团队的开发工程师，负责编译工具链维护和推广与 C++20 协程的编译器支持、协程库开发及应用落地，对 C++20 协程与编译器优化有深刻的理解。LLVM 社区 Committer，致力于 C++20 协程在 Clang/LLVM 中的实现、完善及优化。C++ 标准委员会成员，目前致力于将 C++20 协程中大规模应用过程中发现的语言问题及解决方案反馈到语言标准层面。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

现代化工具链在大规模 C++项目中的实践

许传奇
阿里巴巴开发工程师

议程

1. 编译器升级/切换
2. 编译优化简介
3. C++20 协程简介
4. C++20 Modules 简介
5. 总结

01

编译器升级/切换

编译器升级/切换 – 现状

- 大规模C++项目的困境
  - 项目高度自治 – 自主决定编译器版本、语言标准
  - 高度业务导向 – 少关注、不关注编译器和语言标准
  - 先发劣势 – 丧失应用新技术、新特性的能力
  - 沉疴难起 – 编译器版本、语言标准、库依赖被锁死
- C++编译器开发的困境
  - 过往现状 – 投入大、周期长、欠账多
  - 基础设施（编译器）– 投入较少、缺乏支持
  - 开发困境 – 不知道做什么，做出来没人用

编译器升级/切换 – 挑战

- 修复更严格编译器检查的问题
- 修复不同编译器行为差异的问题
- 修复语言标准、编译器行为改变的问题 – 完善测试
- 二进制依赖、ABI兼容问题 – 全源码编译/服务化
- 性能压测、调优

编译器升级/切换 – 成本

- 业务1（规模5M）
  - 二进制依赖、ABI兼容问题 – 0.5人月； 编译器切换、CI、CD – 1.5人月；性能分析调优 – 1人月；
- 业务2（规模7M）
  - 编译器切换改造 – 2人月； 性能压测调优 – 1人月；
- 业务3（规模3M）
  - 二进制依赖、ABI兼容问题 – 预估2人年

编译器升级/切换 – 风险控制

- 由专业的编译器团队负责
- 绝大多数问题在构建、运行、压测阶段暴露并得到修复
- 少数编译器造成的程序行为变化问题需要完善的测试集
- 极少数编译器切换造成的问题在产线上暴露出来 – 本质是业务/库代码的bug

编译器升级/切换 – 收益

- 更短的编译时间
- 更好的运行时性能
- 更好的编译、静态、运行时检查
- 更多优化技术 – ThinLTO、AutoFDO、Bolt等
- 更新的语言特性支持 – C++20协程、C++20 Module等
- 持续性更新升级 – 良性循环

02

编译优化简介

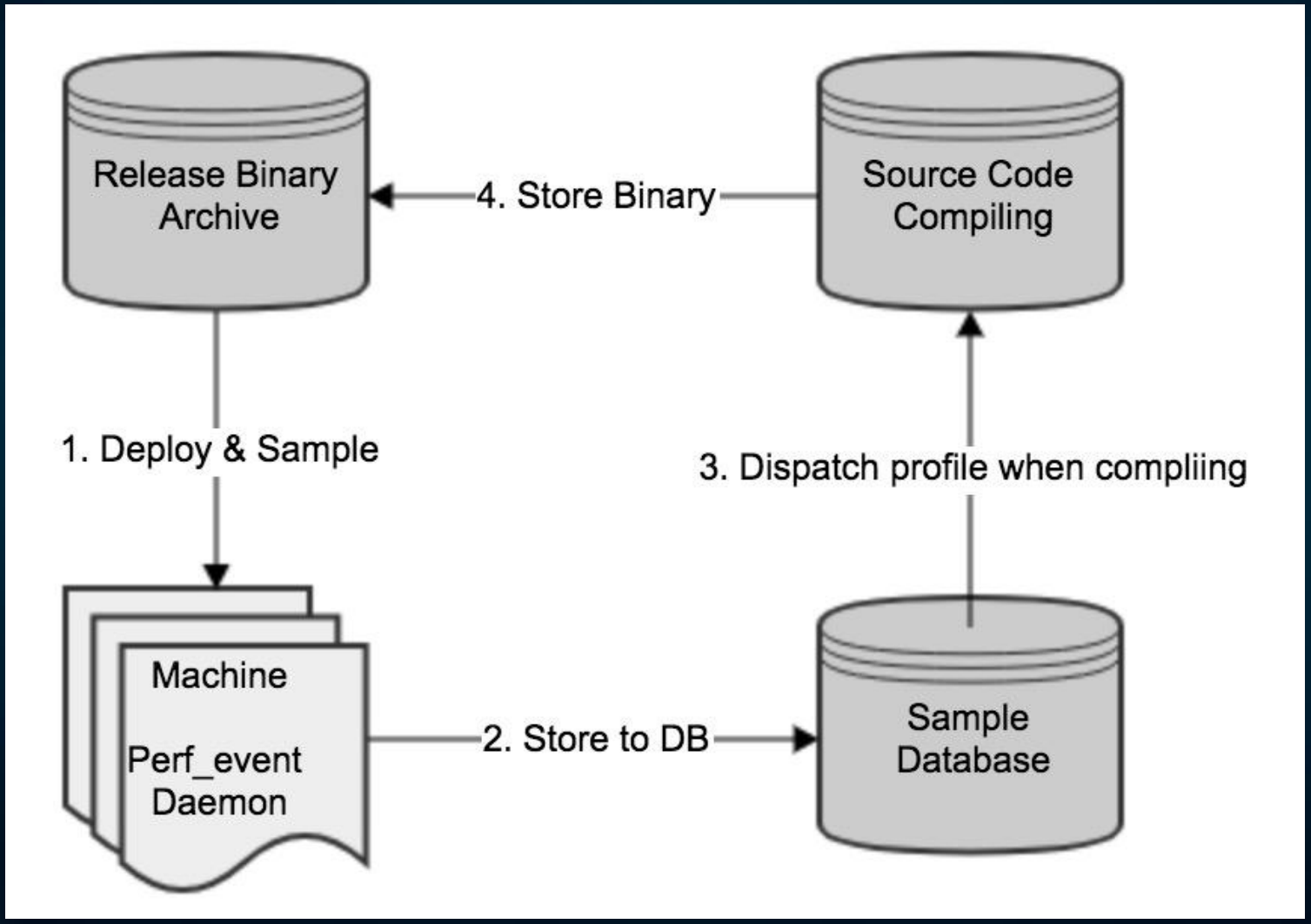
编译优化简介 – ThinLTO

- 传统编译优化只能根据当前 TU 中的信息进行优化。
- LTO (Link Time Optimization) 可使编译器根据所有 TU 中的信息进行全局优化。
  - 可大幅增加程序运行性能。
  - 大幅增加编译时间与内存使用。
- ThinLTO 同样能根据所有 TU 的信息进行优化。
  - 编译时间与内存使用相比于 LTO 大幅下降。
  - 性能与 LTO 相比差大。
- 在实践中能拿到 5%~9% 的性能收益

编译优化简介 – AutoFDO

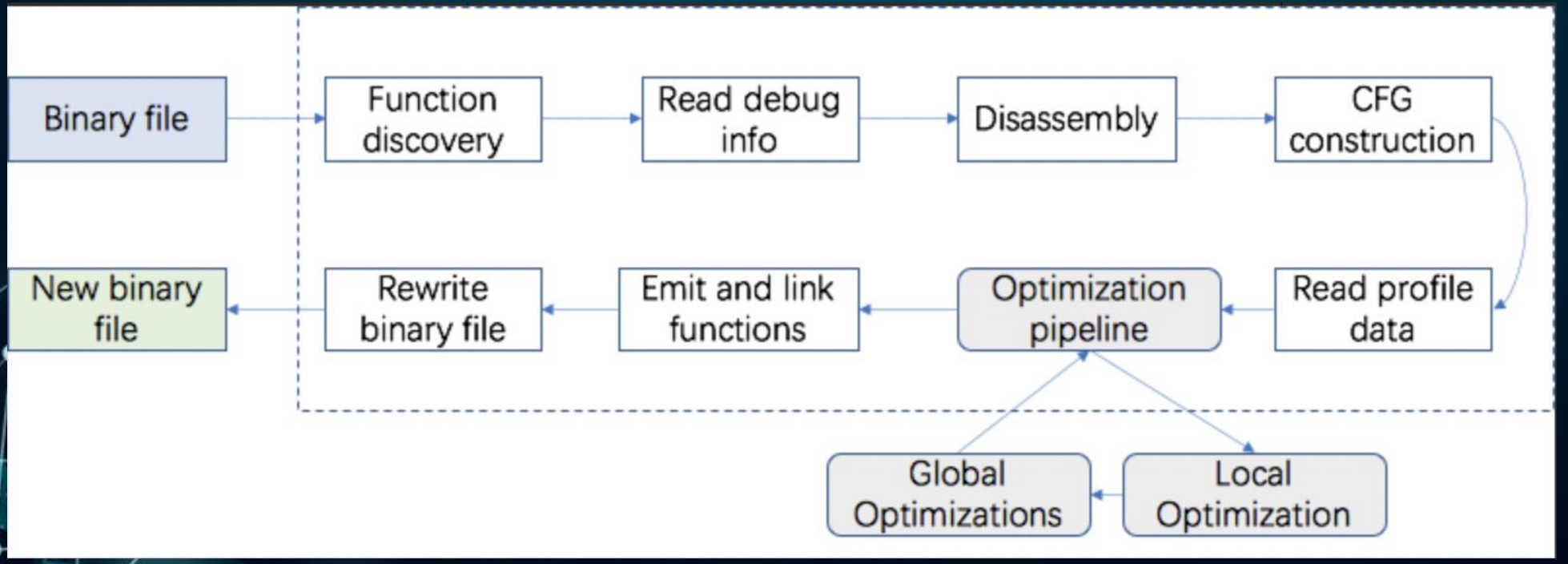
- 传统编译优化时无法获得运行时信息，只能通过建模的方式“猜”运行时状态进行优化。
- FDO (Feedback Directed Optimization) 通过 Profiling 获取运行时信息以供优化。
  - Profiling 时不能开启优化。编译产物不一致。
  - 需要使用 benchmark 进行信息采集。场景不一致。
- AutoFDO 通过 PMU 采集数据，开销小，可以获取真实场景下的数据进行优化。
- 在实践中能拿到 2% ~ 5% 的性能收益

编译优化简介 – AutoFDO



编译优化简介 – BOLT

- BOLT 是针对 Binary 的优化工具。支持对 Binary 的代码重排，降低 Instruction Cache Miss。在大规模程序中效果显著。
- 在实践中能拿到 7% ~ 8% 的性能收益。



03

C++20 协程简介

C++20协程简介

- 协程是一个可挂起的函数。
  - 支持以同步方式写异步代码。
- C++20 协程是无栈协程。在语义层面不保存调用上下文信息。
  - 对比有栈协程
    - 更好的执行 & 切换效率。
  - 对比 Callback
    - 更简洁的编程模式，避免 Callback hell。

C++20协程 Example - Sync

```
uint64_t ReadSync(std::vector<File> Inputs) {
    uint64_t read_size = 0;
    for (auto &&Input : Inputs)
            read_size += ReadImplSync(Input);
    return read_size;
}
```

C++20协程 Example - Async

```
template <RangeT Range, Callable Lambda>
future<void> do_for_each(Range, Lambda);                  // We need introduce another API.
future<uint64_t> ReadAsync(vector<File> Inputs) {
    auto read_size = std::make_shared<uint64_t>(0);       // We need introduce shared_ptr.
    return do_for_each(Inputs,                            // Otherwise read_size would be
                       [read_size] (auto &&Input){        // released after ReadAsync ends.
                         return ReadImplAsync(Input).then([read_size](auto &&size){
                                *read_size += size;
                                return make_ready_future();
                           });
                       })
      .then([read_size] { return make_ready_future<uint64_t>(*read_size); });
}
```

C++20协程 Example - Coro

```
Lazy<uint64_t> ReadCoro(std::vector<File> Inputs) {
    uint64_t read_size = 0;
    for (auto &&Input : Inputs)
        read_size += co_await ReadImplCoro(Input);
    co_return read_size;
}
```

- 使用同步方式写异步代码。
- 兼具开发效率与运行效率。

C++20协程开发

- C++20 协程是无栈协程，需要编译器介入才能实现。
  - 判定协程并搜索相关组件。（Frontend Semantic Analysis）
  - 生成代码。（Frontend Code Generation）
  - 生成、优化、维护协程桢。（Middle-end）
- C++20 协程只设计了基本语法，并没有加入协程库。
  - C++20 协程的目标用户是协程库作者。
  - 其他用户应通过协程库使用协程。

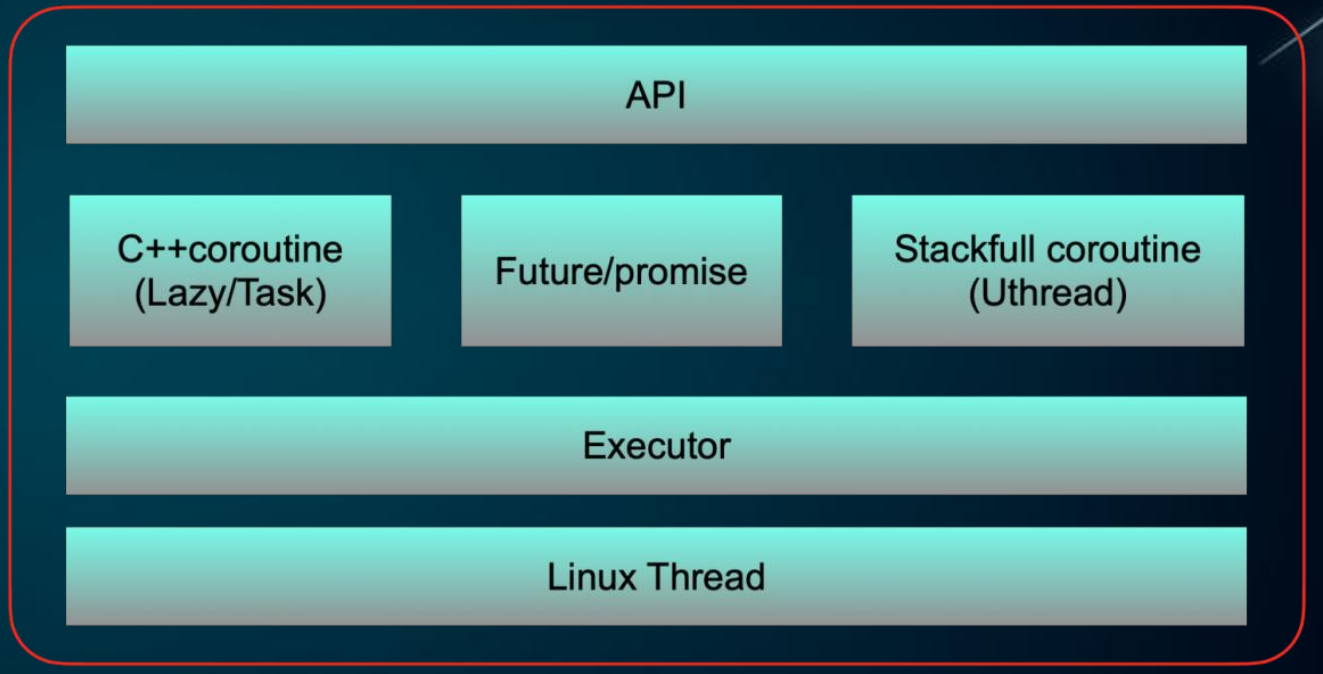
C++20协程开发 – 编译器

- GCC
  - 与社区合作进行协程的支持。
  - GCC-10是第一个支持C++协程特性的GCC编译器。
  - 仅支持，无优化。

- Clang/LLVM
  - 与 Clang/LLVM 社区合作完善C++协程。
  - 改善&优化：对称变换、协程逃逸分析和CoroElide优化，协程帧优化（Frame reduction），完善协程调试能力、尾调用优化、Coro Return Value Optimization等。
  - 在 Clang/LLVM14 中，coroutine 移出了 experimental namespace。
  - Maintaining

C++20协程开发 – 协程库

- async_simple
  - 设计借鉴了 folly 库协程模块。
  - 轻量级。
  - 包含有栈协程、无栈协程以及 Future/Promise 等异步组件。
  - 从真实需求出发。
  - 与调度器解藕，用户可以选择合适自己的调度器。
  - 经受了工业级 Workload 的考验。
  - 开源于：https://github.com/alibaba/async_simple



C++20协程应用

- 业务1（1M Loc、35w core）
  - 原先为同步逻辑
  - 协程化后 Latency 下降 30%
  - 超时查询数量大幅下降甚至清零
- 业务2（7M Loc）
  - 原先为异步逻辑
  - 协程化后 Latency 下降 8%
- 业务3（100K Loc、2.7w core）
  - 原先为同步逻辑
  - 协程化后 qps 提升 10 倍以上性能

04

C++20 Modules 简介

C++20 Modules

- 降低复杂度与出错的机会
- 更好的封装性
- 更快的编译速度

C++20 Modules – 降低复杂度

```
#include "a.h"
#include "b.h"
```



```
#include "b.h"
#include "a.h"
```

C++20 Modules – 降低复杂度

import a;
import b;



import b;
import a;

C++20 Modules – 更好的封装性

```
// a.h
namespace detail {
    void not_use_directly() {...}
}
...
```

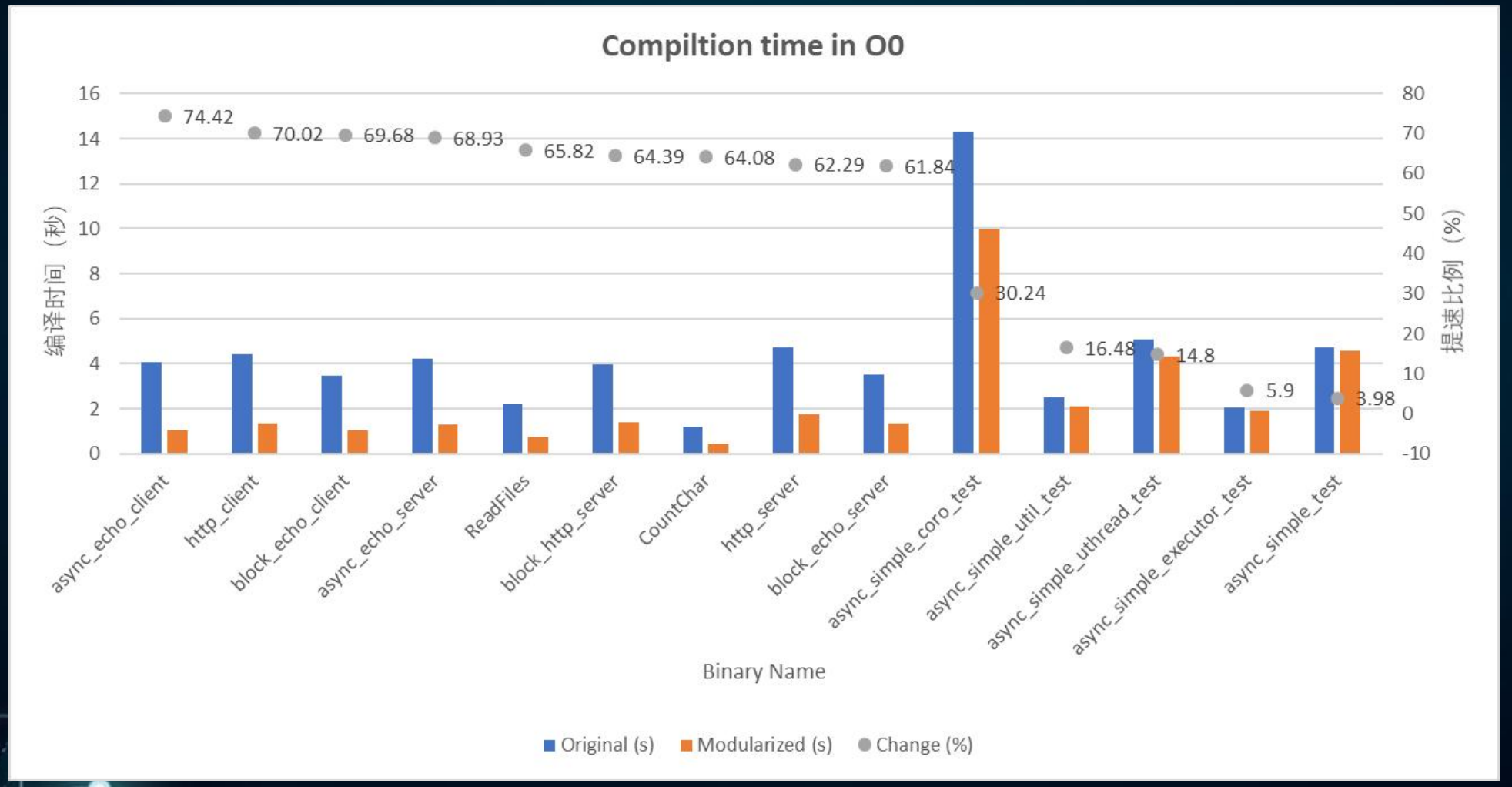
```
// Use.cpp
#include "a.h"
void foo() {
    detail::not_use_directly(); //  It shouldn't!
}
```

C++20 Modules – 更好的封装性

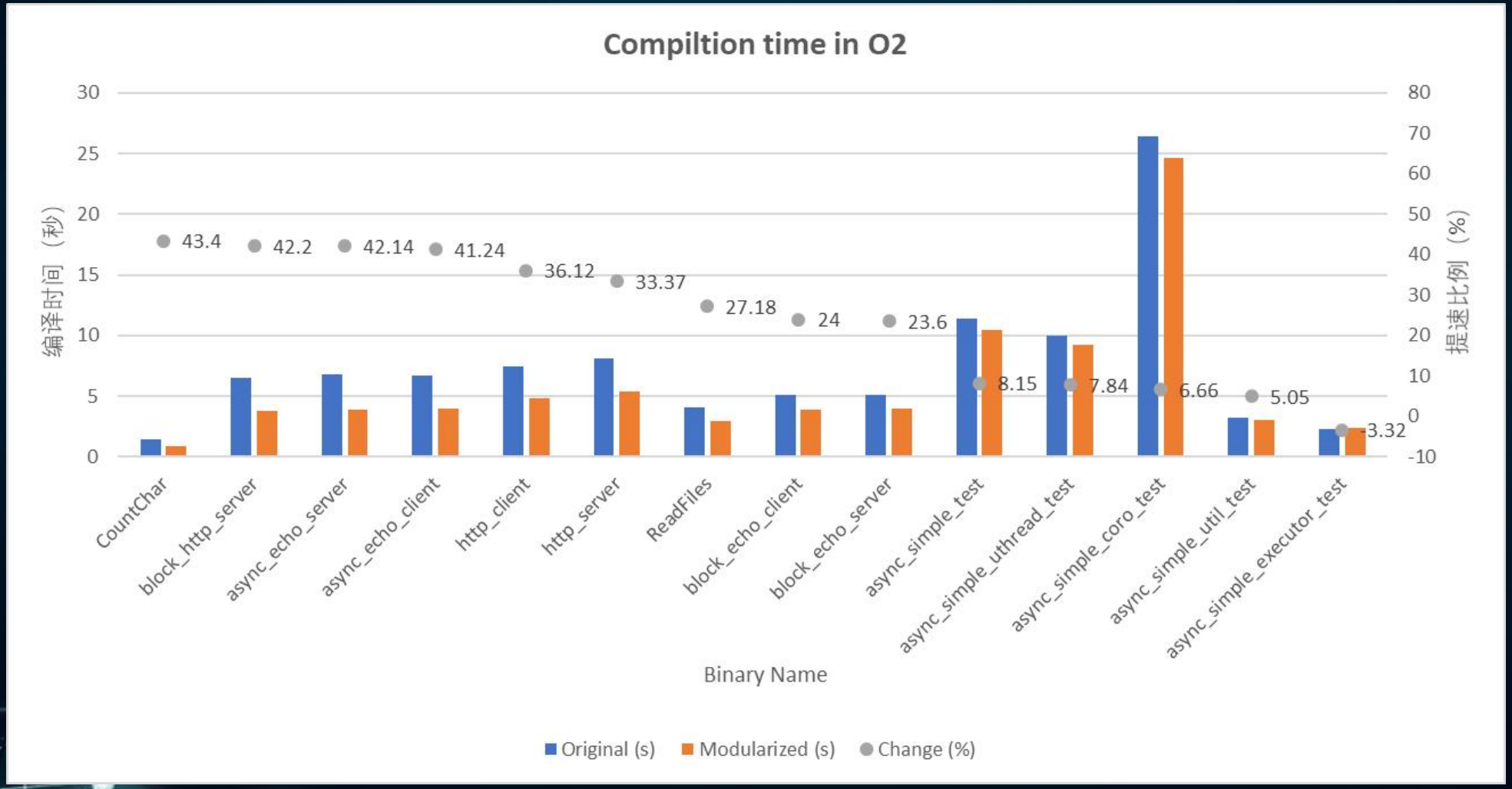
```
// a.cppm
namespace detail {
    void not_use_directly() {...}
}
export void use_directly() {…}
…
```

```
// Use.cpp
import a;
void foo() {
     // Compilation Error
    detail::not_use_directly();
    use_directly(); // OK
}
```

C++20 Modules – 更快的编译速度



C++20 Modules – 更快的编译速度



C++20 Modules – 完全改造真实世界项目

```
16  #include <cstdlib>
17  #include <iostream>
18  #include <thread>
19  #include "asio_util.hpp"
20
21  using asio::ip::tcp;
22
23  async_simple::coro::Lazy<void> session(tcp::socket sock) {
24      int msg_index = 0;
25  >   for (;;) {…
42
43      std::error_code ec;
44      sock.shutdown(asio::ip::tcp::socket::shutdown_both, ec);
45      sock.close(ec);
46
47      std::cout << "Finished echo message, total: " << msg_index - 1 << ".\n";
48  }
```

C++20 Modules – 完全改造真实世界项目

```
16  import std;
17  import asio_util;
18
19  using asio::ip::tcp;
20
21  async_simple::coro::Lazy<void> session(tcp::socket sock) {
22      int msg_index = 0;
23      for (;;) {…
40
41      std::error_code ec;
42      sock.shutdown(asio::ip::tcp::socket::shutdown_both, ec);
43      sock.close(ec);
44
45      std::cout << "Finished echo message, total: " << msg_index - 1 << ".\n";
46  }
```

C++20 Modules – 完全改造真实世界项目

```
16  import std;
17  import asio_util;
18
19  using asio::ip::tcp;
20
21  async_simple::coro::Lazy<void> session(tcp::socket sock) {
22      int msg_index = 0;
23      for (;;) {…
40
41      std::error_code ec;
42      sock.shutdown(asio::ip::tcp::socket::shutdown_both, ec);
43      sock.close(ec);
44
45      std::cout << "Finished echo message, total: " << msg_index - 1 << ".\n";
46  }
```

More examples: https://github.com/alibaba/async_simple/tree/CXX20Modules

C++20 Modules – 进展

- 编译器初步开发完成
- 支持 std modules
- 优先内部应用
- 已在 Clang15 中发布
- 探索编译器与构建系统交互

05

总结

总结 - 使用高版本编译器的收益

- 更短的编译时间
- 更好的运行时性能
- 更好的编译、静态、运行时检查
- 更多优化技术 – ThinLTO、AutoFDO、Bolt等
- 更新的语言特性支持 – C++20协程、C++20 Module等
- 持续性更新升级 – 良性循环

谢谢观看

CPP-Summit 2022



向东

网易杭州研究院架构师

向东目前担任网易杭州研究院架构师，有10年的存储系统开发经验，专注于对象存储/块存储/视频云等方向，熟悉分布式存储系统的架构与设计。对于分布式系统的一致性协议、性能瓶颈分析与优化有深入的理解。目前主要参与开源分布式存储CURVE(opencurve.io)项目的相关工作。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

Curve 分布式存储架构与设计

向东
网易数帆架构师

议程

1. 自研存储的初衷
2. Curve的设计目标
3. Curve块存储设计场景及应用
4. Curve块存储核心设计
5. Curve文件系统设计目标及场景

01

自研存储的初衷

自研存储的初衷

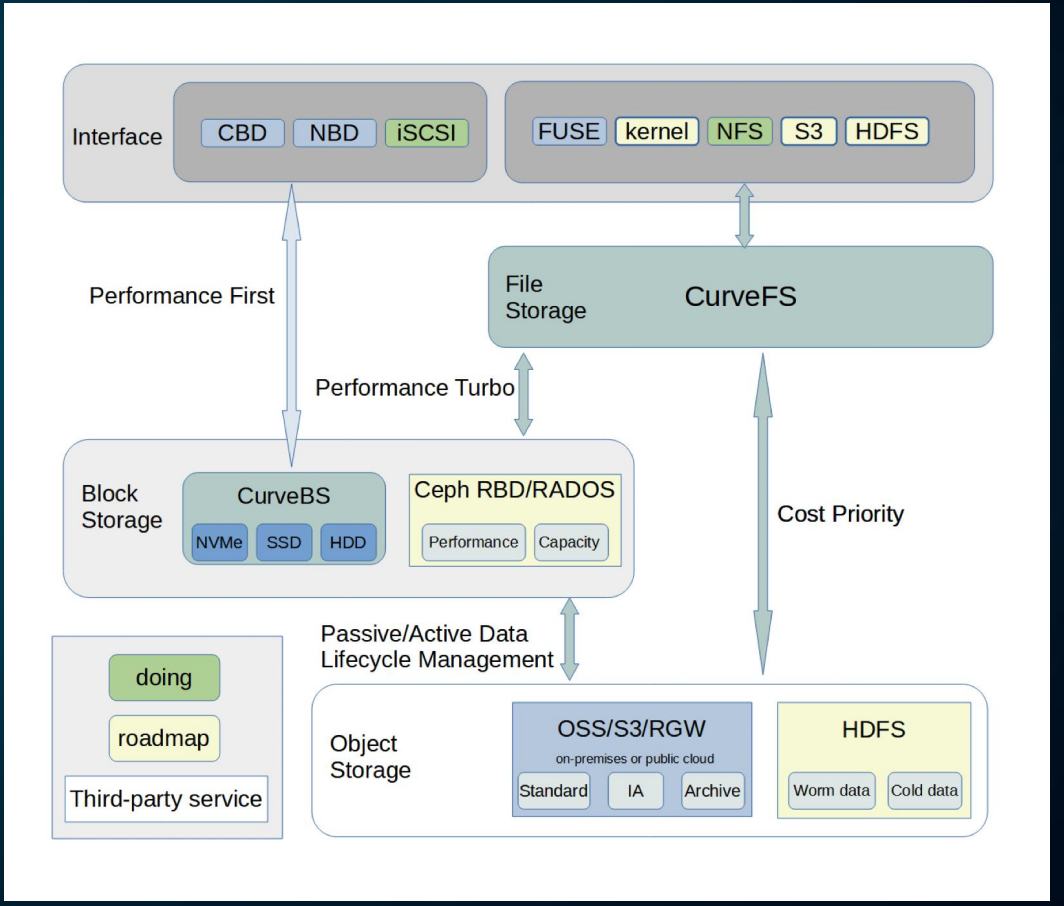
- 代码复杂/代码量大
- 运维难度高
- 无法满足高性能要求

02

Curve的设计目标

Curve的设计目标

- Curve云原生定义存储
- Curve分布式块存储
- Curve分布式文件存储
- 云原生、高性能、易运维

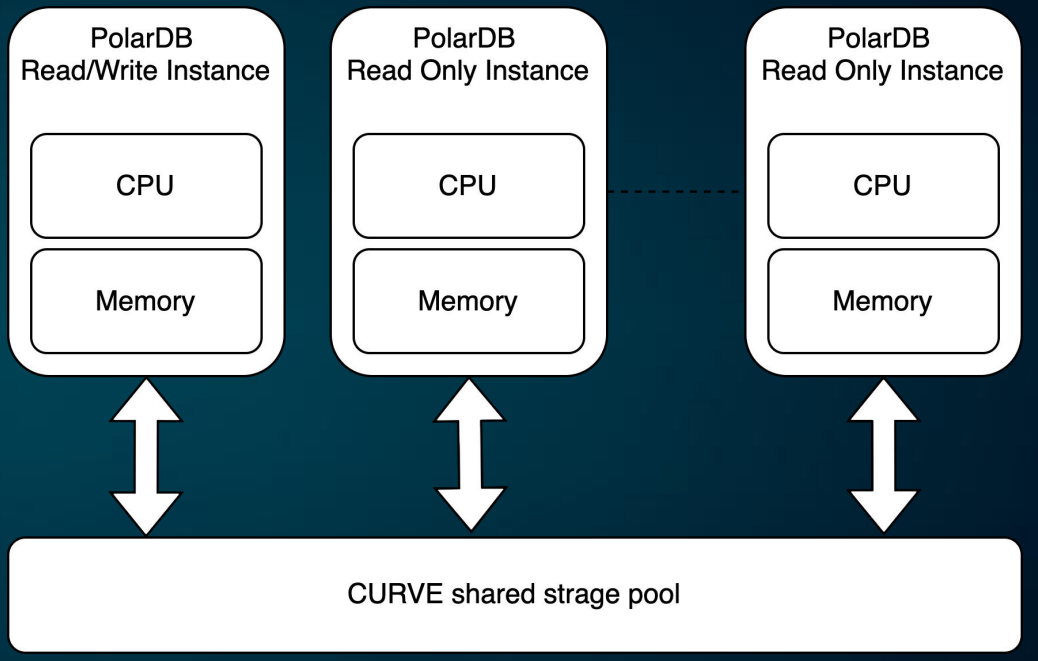


03

Curve块存储设计场景及应用

Curve块存储现状及应用场景

- 高性能分布式共享数据库场景
- Curve提供底层分布式共享存储
- Polardb for PostgreSQL提供上层高性能数据库服务
- 性能测试
  - benchmarkSQL 每分钟事务数提升39%
  - pgbench 延迟降低21% TPS提升26%

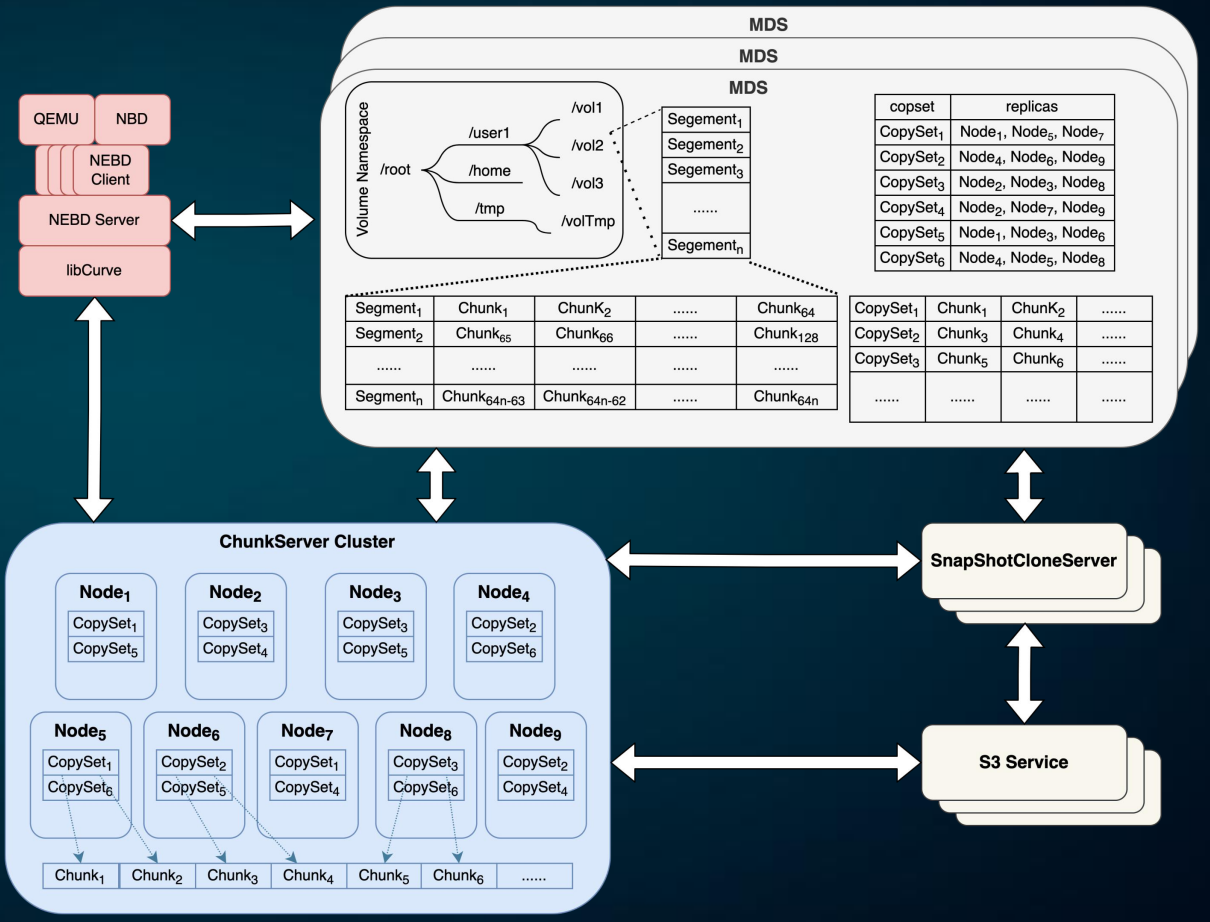


04

Curve块存储核心设计

Curve块核心设计

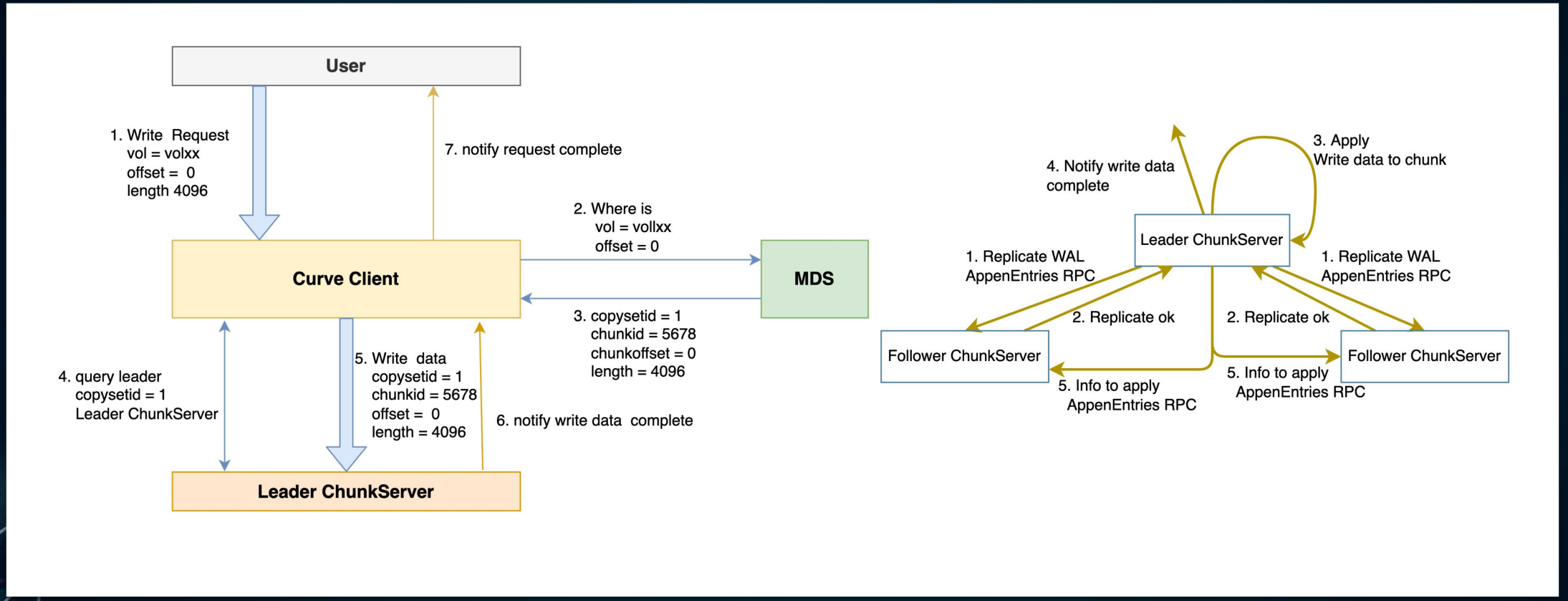
- 数据块的映射和分布
  - 集中管理
- 数据副本的一致性
  - RAFT一致性协议
- 高可用性/高可靠性
  - CopySet分配算法
  - 拓扑结构
- 数据存储引擎
  - chunkfilepool (降低写放大)
  - data stripe (增大并发)
  - zerocopy
- 云原生



4.1

数据副本一致性协议

数据副本的一致性 raft协议

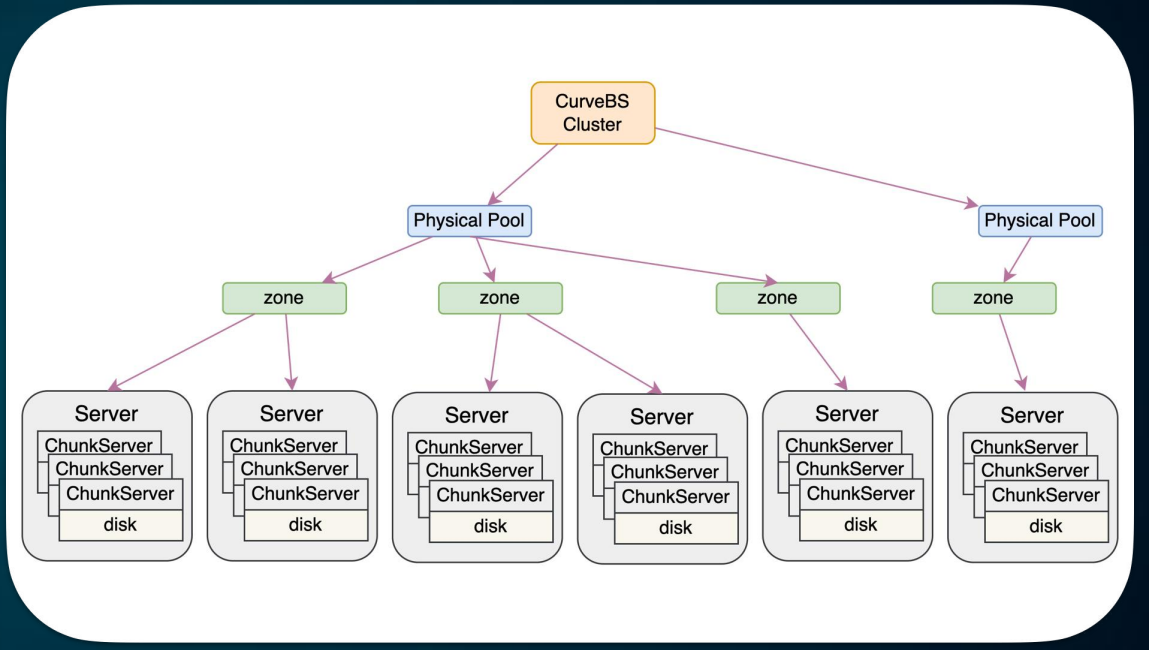


4.2

数据可靠性

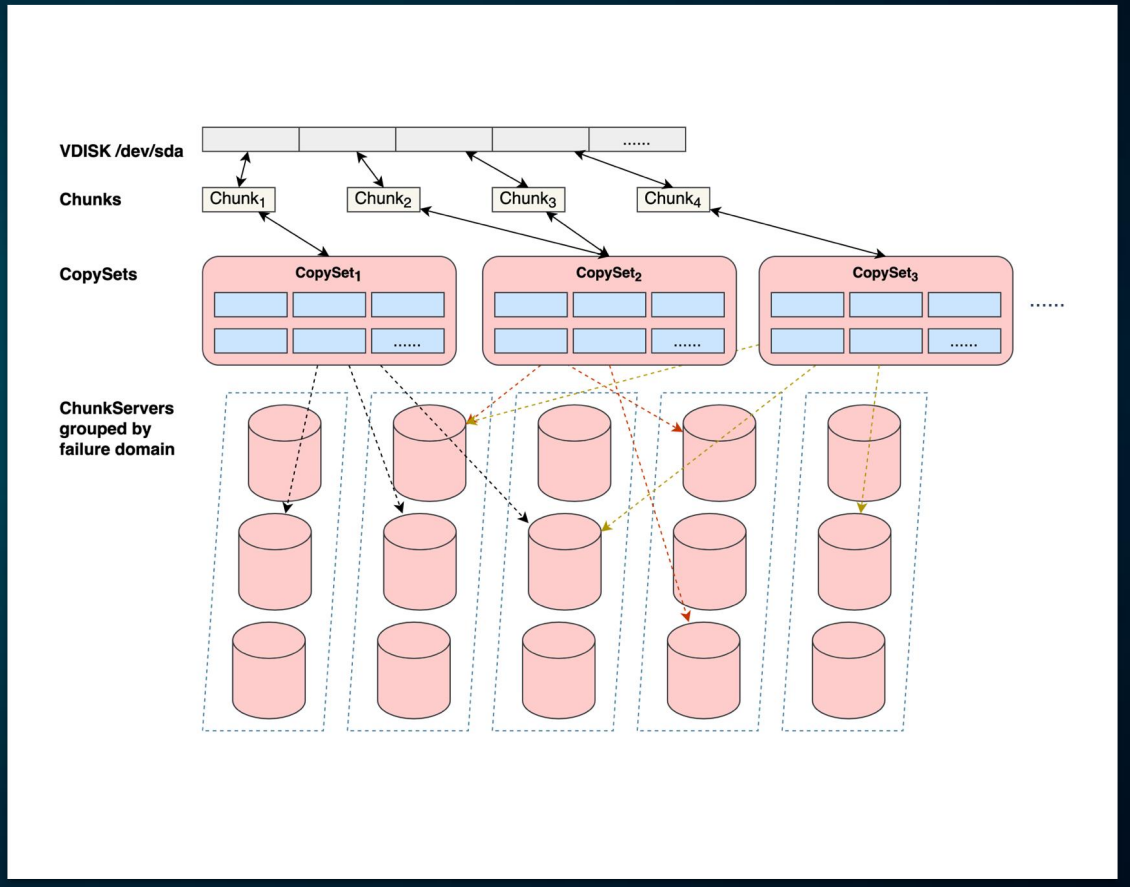
可靠性 Curve块的拓扑结构

- physical pool用于实现对机器资源物理隔离
- zone故障隔离的基本单元
- server表示物理服务器
- chunkserver物理服务器上的服务实例



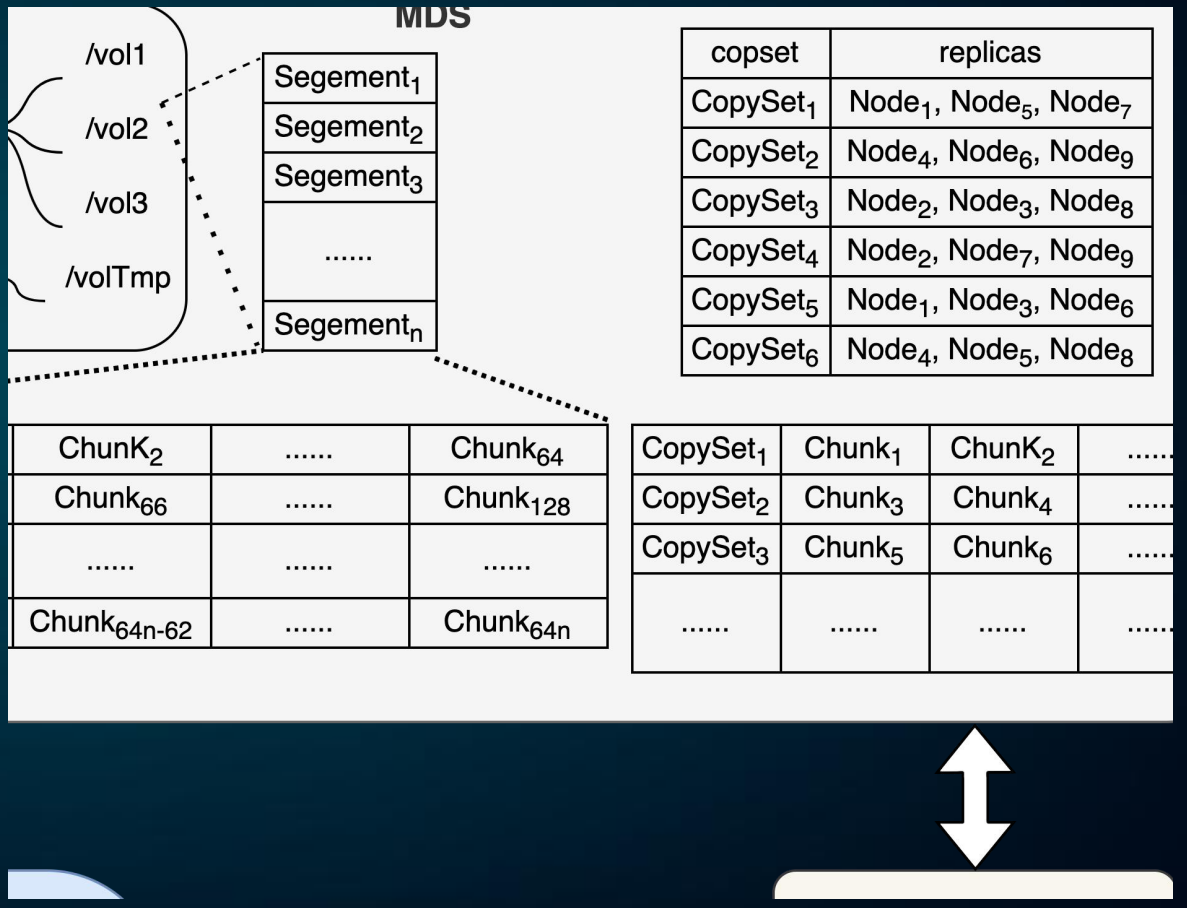
可靠性 Curve块数据组织

- Curve将虚拟块设备映射到文件
- 每个文件包含的chunk分散在集群的存储节点
- chunkserver按照故障域分组
- copyset中的节点属于不同的故障域



可靠性 copyset分配算法

- copyset分配算法
- 1200块盘组成的集群 (每块盘的MTBF为120万小时)
- 3副本
- 1块盘的恢复时间为5分钟
- 数据可用性为6个9

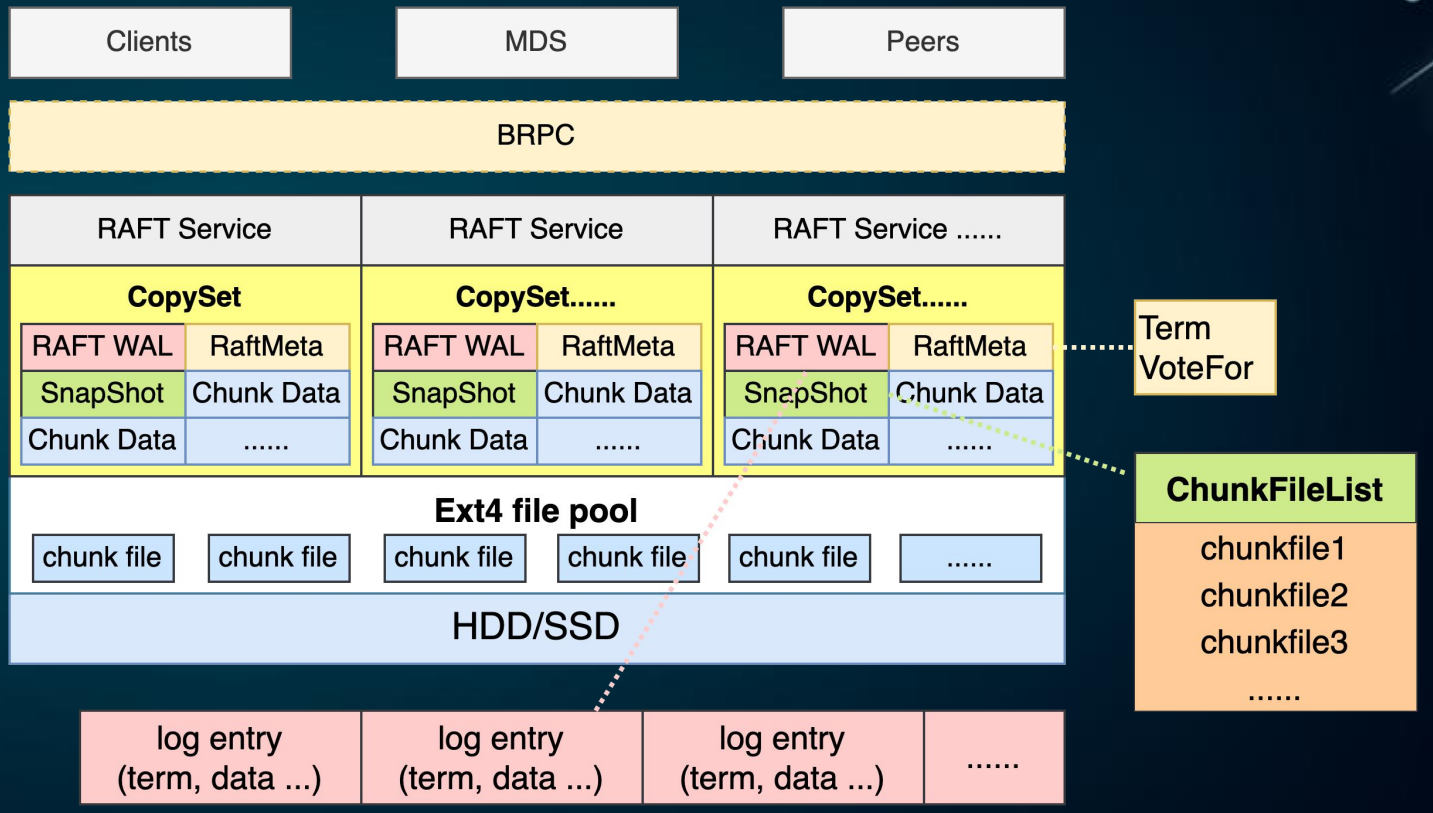


4.3

数据存储引擎

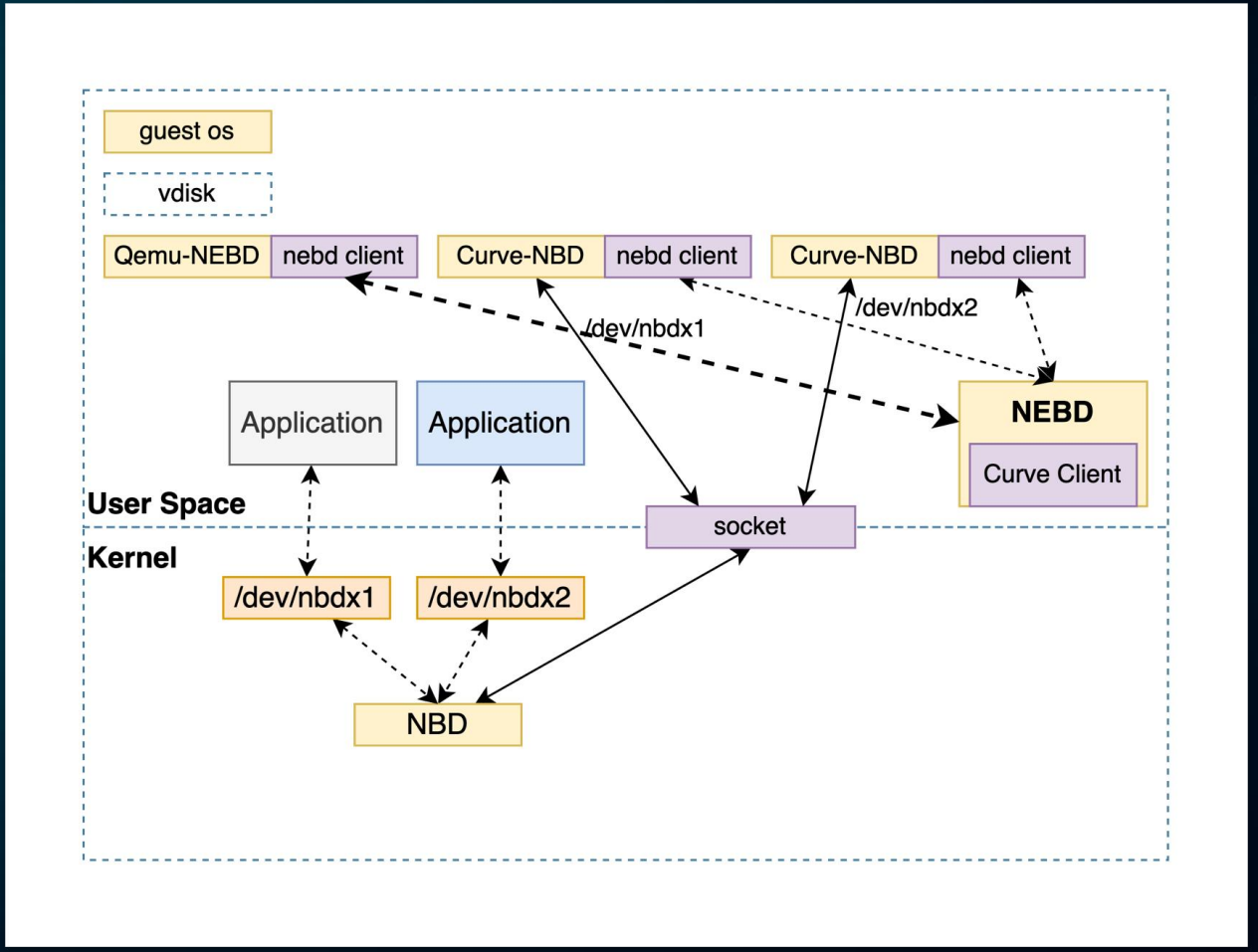
存储引擎 chunkserver

- chunkserver负责数据的存储
- RAFT协议保持数据的一致性
- chunkfile pool降低元数据开销



关于客户端的在线升级

- 客户端分成NebdClient与NebdServer两部分
- NebdClient只做简单的转发
- NebdServer实现大部分的客户端逻辑



一些性能方面的考虑

yanjiangxu commented on 10 May 2017 • edited

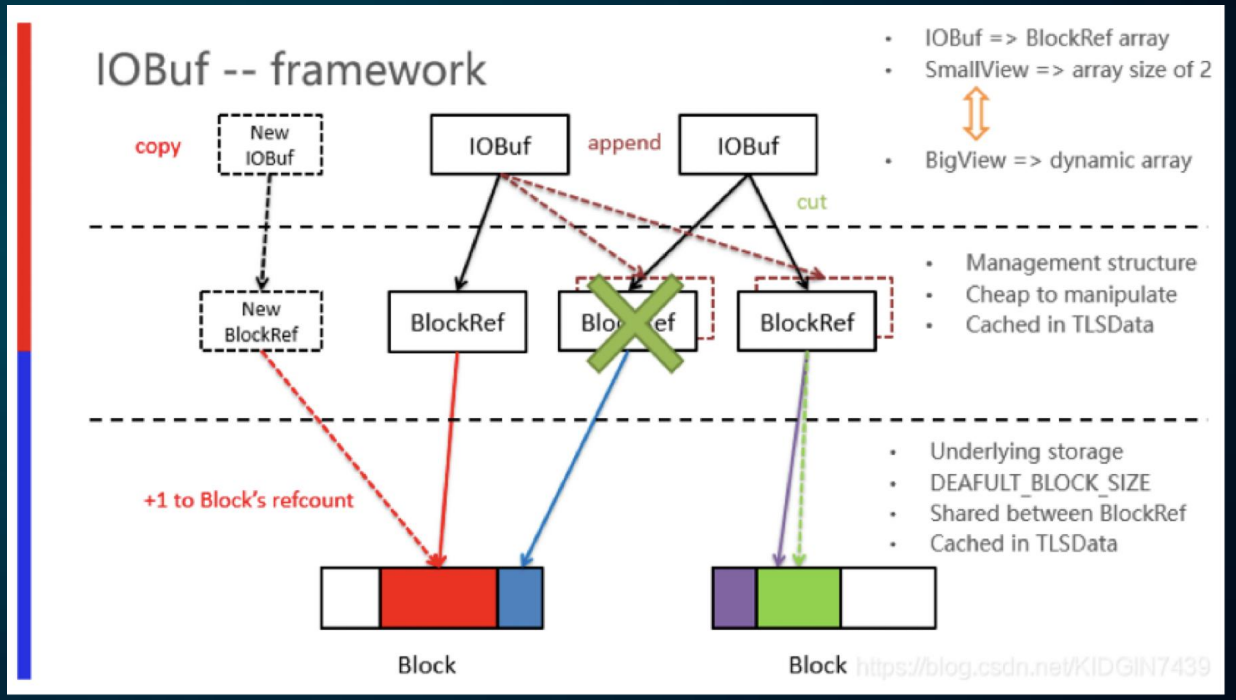
The behavior In original flow:
1) In the case of 3 copies, the client sends the write to the primary, and then the primary forwards to the other two replicas. The primary must wait for the commit until all the replicas from completion. The real write IO latency is the longest latency among three copies.
2) When the data distribution is not uniform, the osd node which owns hottest data will become the bottleneck of total latency.

The behavior In my enhancement:
commit_majority function is not required to wait for all copies of commit to complete: as long as the majority of the commit is completed, the cluster would notify the client to complete.

As in my implementation:
We also follow the scenario: When the number of copies is less than min_size, ceph does not provide read and write capabilities. In order to ensure security, you must ensure that the number of commit to majority.
And I defined two important variables:

```
allow_uncommitted_replicas = pool_min_size - 1,
majority = pool_default_size - allow_uncommitted_replicas.
```

Test case:
1) 3 copies, pool_default_size = 3, pool_min_size = 2, majority = 2,allow_uncommitted_replicas=1
2) The average latency decreased by **11.5%**;
3) Long tail (99.9%) latency decreased by **71.59%**;
4) IOPS increased by **13%**.



4.4

Curve设计效果验证

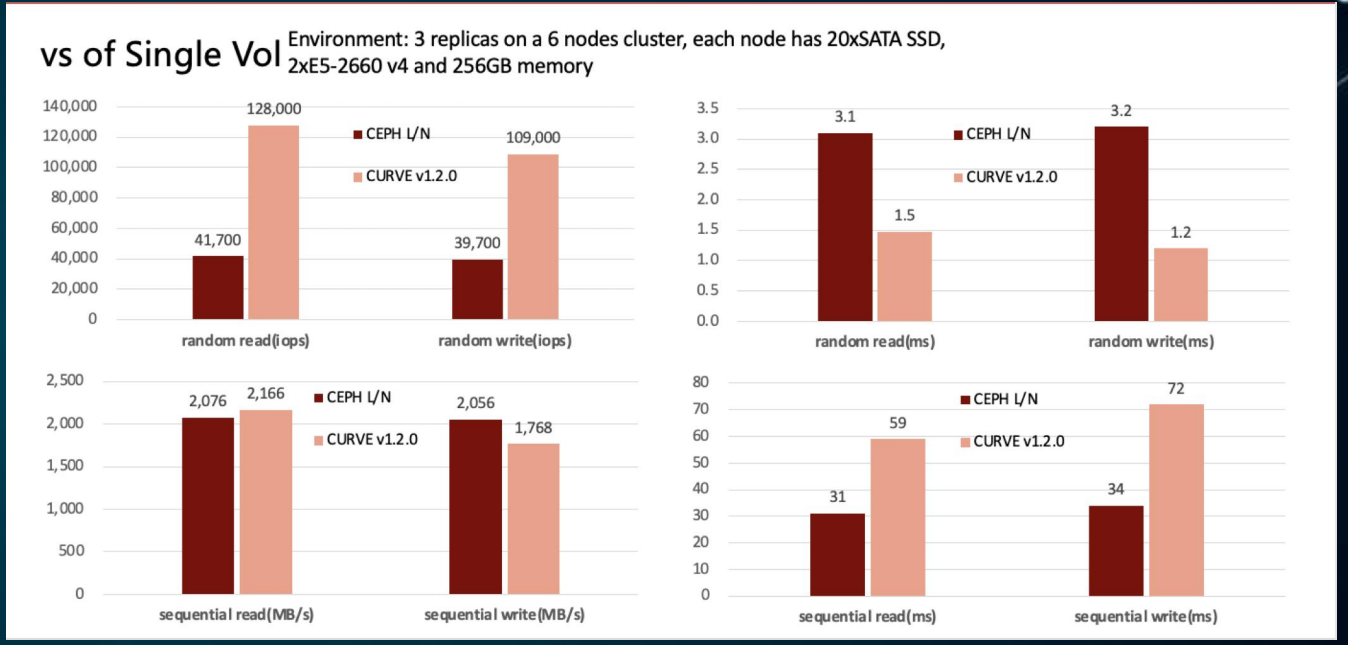
不同故障对Curve的影响

- 故障对I/O抖动延迟的影响

Faults case	curve i/o 抖动
磁盘失效	4s
服务器故障	4s
服务器假死或反应慢	4s
网络延迟高	1s frequently

Curve的性能对比

- 商用服务器同等硬件条件
- 测试环境
- 6接点集群，每个节点20 x SSD，256GB内存，2 x E5-2660 cpu



支持RDMA

- 测试方法
- echo服务测试，发送4KB大小的包并等待返回
- 延迟低于50us

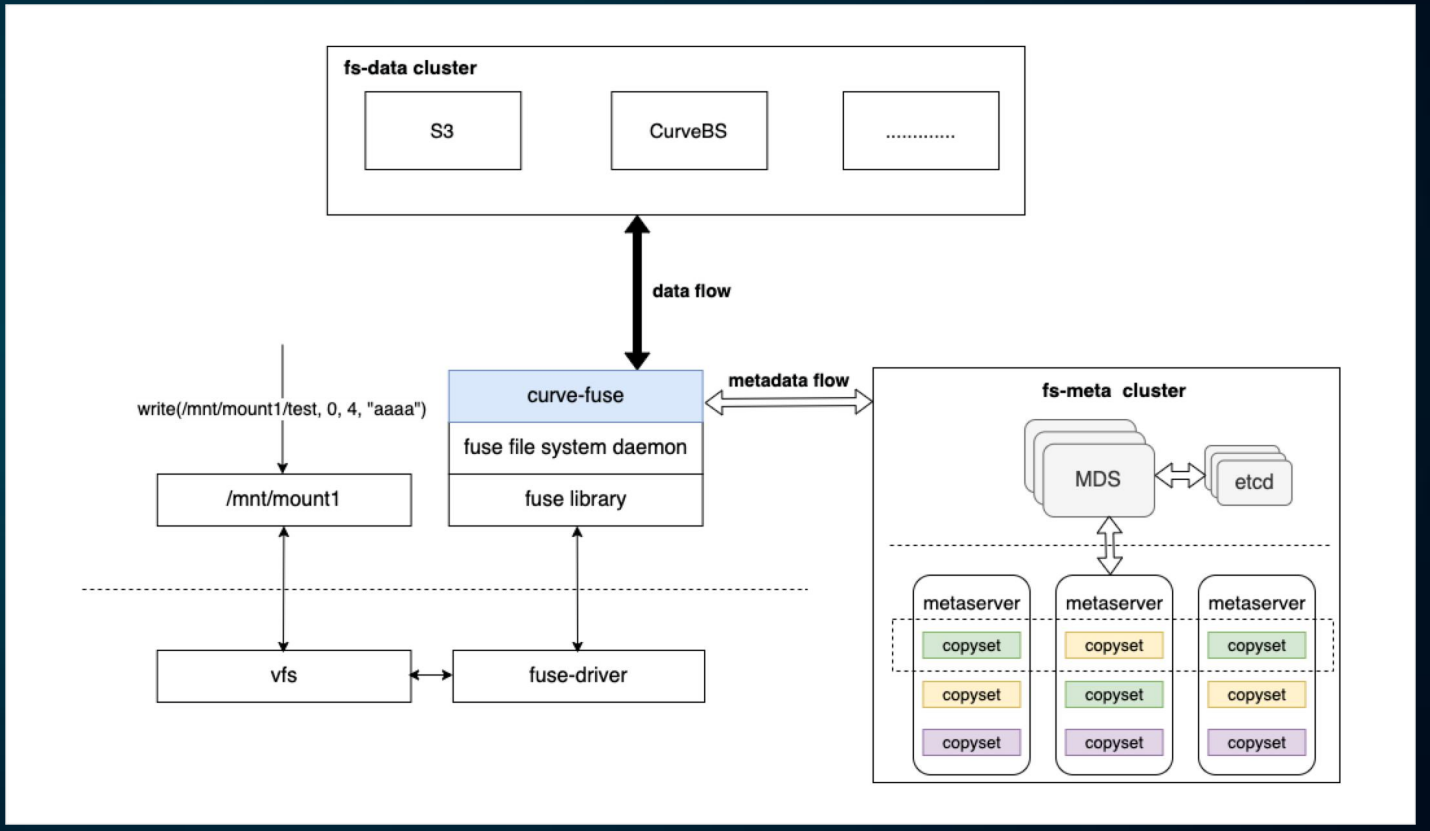
```
no_client --server=10.187.0.91:13339 --use_ucp=true --thread_num=1 --brpc_ucp_worker_busy_poll=true --attachment_size=4096
me/incubator-brpc/src/brpc/ucp_ctx.cpp:35] Running with ucp library version: 1.14.0
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=19186 latency=50
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20026 latency=47
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=18512 latency=50
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=18512 latency=50
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=18406 latency=51
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20643 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20873 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20907 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20901 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20894 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20967 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20923 latency=46
me/incubator-brpc/example/multi_threaded_echo_c++/client.cpp:144] Sending EchoRequest at qps=20923 latency=46
```

05

Curve文件系统设计目标及场景

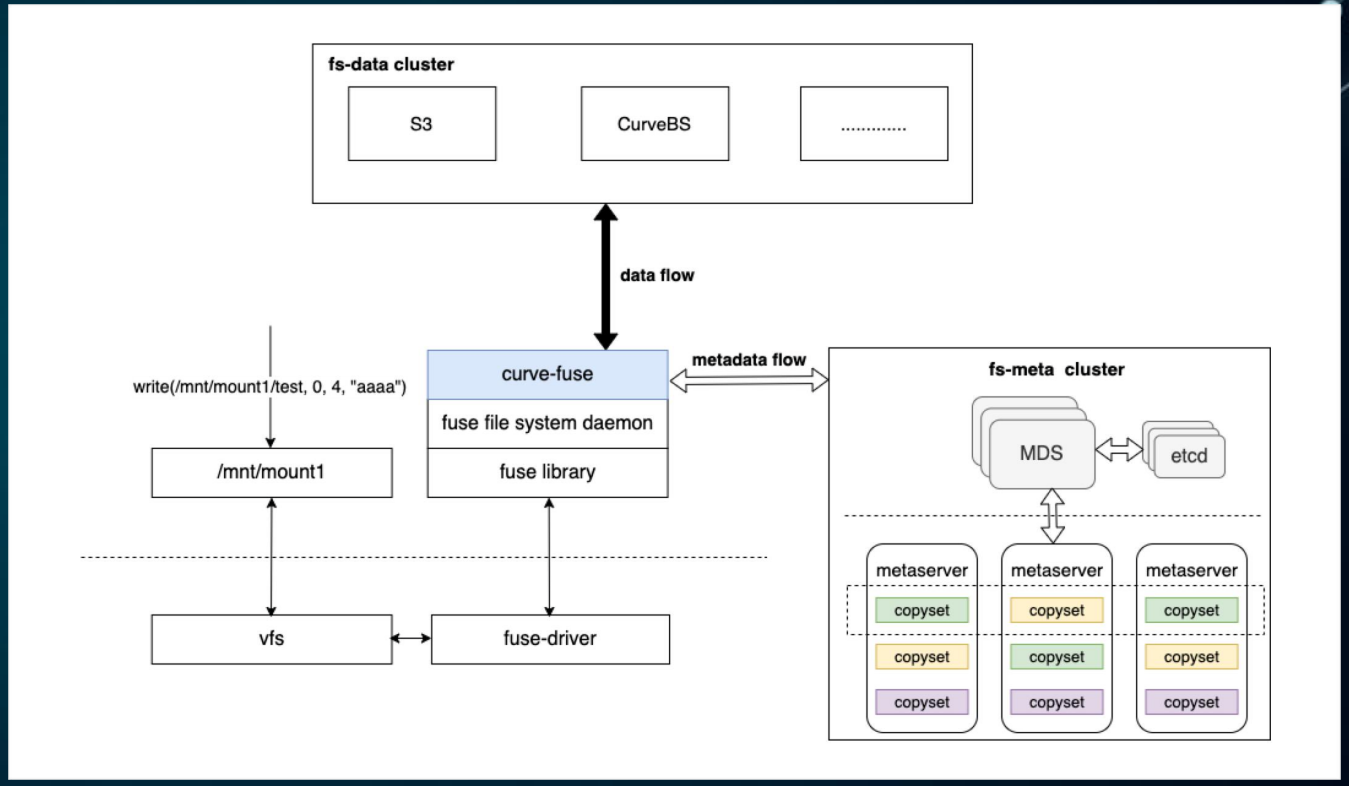
Curve文件系统设计目标

- 元数据服务
- 高性能
- 可扩展易运维
- 云原生



面临的主要场景及挑战

- 兼顾性能与容量的机器学习场景
- 快速跨云弹性发布的业务
- 低成本大容量需求的业务
- 中间件冷热数据自动分离
- S3和POSIX统一访问需求



Curve文件系统进展

元数据操作性能对比	Directory Creation	Directory Stat	Directory Remove	File Creation	File Stat	File Remove	Tree Creation	Tree Remove
CurveFS vs CephFS	10%	19%	0%	0%	20%	29%	-20%	100%

You can take away

- Curve块存储挑战
  - 高可用性
    - RAFT协议; copyset分配算法配合基于拓扑结构的故障域划分
  - 性能方面 (降低cpu利用率/降低写放大/降低延迟)
    - RAFT协议；降低写放大提升吞吐数据并发
  - 易运维
  - 云原生支持
- Curve文件存储挑战
  - 场景多样，读写模式复杂；典型的小文件多的场景，海量客户端共享场景
  - Cache模块的可扩展性，数据一致性
  - 元数据模块的可扩展与数据一致性

谢谢观看

CPP-Summit 2022

谢茂培



金山办公研发中台事业部office产研架构师

任金山办公研发中台事业部架构师，2001-2005年曾任金山词霸、金山游侠开发工程师、WPS系统架构组WPP开发组组长，在软件设计软件工程管理上具有多年实践经验。曾任基塔科技联合创始人CTC、极简桌面联合创始人CTO，曾参与开发听书网、CCIPTV等

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit 2022

超大规模C++项目构建优化

金山办公 KINGSOFT OFFICE | 谢茂培

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建优化的方法

为什么要做构建优化

超大规模C++项目
构建优化

为什么要做构建优化
我们面对的困境
构建优化的直接收益
构建性能问题分析
构建优化的方法

我们面对的困境



C++编译天生就慢



项目规模快速增长
构建耗时越来越长



构建机资源不足，
构建任务经常需要排队等待，
不能立即开始



构建若意外失败，极大
影响工作节奏



我们需要在多平台、多版本、
多分支上开展构建



我们需要有持续构建
快速发布的平台能力

超大规模C++项目
构建优化

为什么要做构建优化
我们面对的困境
构建优化的直接收益

构建性能问题分析
构建优化的方法

构建优化的直接收益



缩短
工程师等待时间
提升研发效率



更少的机器
完成更多的工作
降低硬件成本



应用更多
自动化工具的前提
提高项目质量

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建优化的方法

金山办公 KINGSOFT OFFICE | 谢茂培

构建性能问题分析

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建过程的典型构成
影响构建性能的主要因素
构建优化的努力方向

构建优化的方法

构建过程的典型构成

拉取代码

执行编译

打包上传

自动化测试

.....

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建过程的典型构成
影响构建性能的主要因素
构建优化的努力方向

构建优化的方法

影响构建性能的主要因素

硬件资源

CPU
内存
硬盘
网络

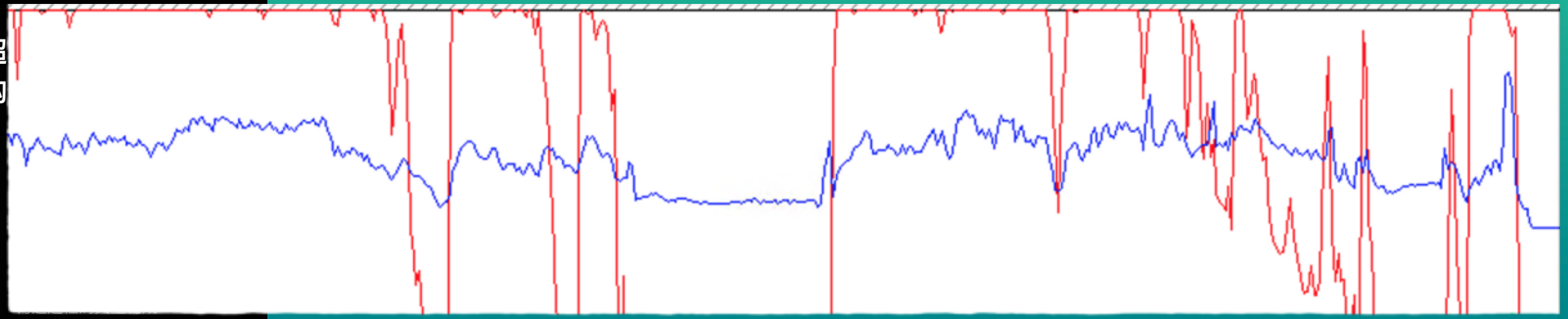
项目自身

工程结构
代码特性

构建过程并发度

一个构建（编译）过程的资源使用情况样例

超
构



CPU　内存　磁盘

影响构建性能的主要因素

构建优化的努力方向

构建优化的方法

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建过程的典型构成
影响构建性能的主要因素
构建优化的努力方向

构建优化的方法

构建优化的努力方向

降低构建任务压力

- 清理无用项目和代码
- 版本控制+构建结果复用
- 使用编译加速工具，如：ccache、UB

提高构建并发度

- 跑满多核
- IO 与 CPU 并行
- 分布式

升级基础设施

- 升级工具链
- 升级硬件

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析
构建优化的方法

金山办公 KINGSOFT OFFICE | 谢茂培

构建优化的方法

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

拉取代码、打包上传等环节（非编译环节）

去除冗余

- git shallow clone
- 检查是否有多余的数据上传、下载
- 检查是否执行了多余的命令
- ……

提高并发度

- 多进程（或多线程）执行命令
- 重叠流水线
- 降低 IO 压力，如数据压缩传输和存储
- ……

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

编译环节优化的方法归类



超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

去除冗余

- 清理无用项目和代码
- 清理重复代码
- 清理重复 include
- 避免 cpp 文件被多个项目包含和重复编译

超大规模C++项目构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

降低编译压力

避免编译期计算
使用常量，或改为运行时计算

合理使用 PCH
· 该用则用，不该用则不用 | · 共享pch

谨慎使用复杂 Template 库

使用 C++20 Module 特性

针对接口编程（以及PImpl）

优化编译、链接选项
· 减少输出 | · INCREMENTAL

使用预声明代替 include

提高并发度 | 构建过程的并发调度机制

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

Makefile、Ninja

文件级并发调度
（make –j8）

MSVC

项目级并发调度
（msbuild –m:8）
项目内文件编辑并发调度
（cl /MP8）

另：VS2022对调度能力有大幅增强，主要是全局文件并发控制、及重叠流水线

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

提高并发度 | 删减项目依赖关系

去除多余的依赖关系

大的动态库用LoadLibrary，避免项目成为构建瓶颈

link过程即慢且难以并发，容易成为瓶颈点
或可配套提供Proxy项目，以简化动态库的使用

静态库项目通常无需直接依赖子项目

将依赖关系上移至使用者dll/exe项目
cmake target_link_libraries INTERFACE

避免依赖跨项目预编译结果导致产生项目依赖

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

提高并发度 | 干预项目构建调度时序

尽早开始瓶颈项目构建，以期尽早完成，减轻瓶颈效应

甚至可考虑将大项目拆解，以并发构建这些拆解出的小项目

避免 IO 密集型任务过度并发，使 CPU 无法跑满

尽量让 IO 密集型任务与 CPU 密集型任务重叠执行

提高预处理过程过程的并发度

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

提高并发度 | 让CPU满负荷运行

高速 SSD（若已可充分并发调度，则 IO 将会成为瓶颈）

关闭防火墙之类后台服务（极大影响 IO 性能）

找出瓶颈项目，针对性优化（比如调高 MP ）

另外：使用msvc时，在极多核的构建机上，我们通常不得不过度并发（m*MP），但要注意避免oom！

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

其他手段 | 对工程进行模块化分解

将单一大工程拆解为多个模块

各模块单独进行版本管理、构建和发布

模块间仅依赖发布（就像依赖 3rd ）

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

其他手段 | 分布式构建

自主进行分布式调度

直接使用分布式编译工具 | 如：IB

前提条件：项目具备超高可并发度，
拥有超高速局域网络环境

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

其他手段 | 复用构建结果或缓存

基于版本管理，使用 3rd 库、以及模块二进制包

使用编译缓存工具 | 如：ccache（对msvc支持不够完善）

超大规模C++项目
构建优化

为什么要做构建优化
构建性能问题分析

构建优化的方法
拉取代码、打包上传等环节
编译环节优化的方法归类
去除冗余
降低编译压力
提高并发度
其他手段

其他手段 | 升级基础设施

采用新的构建工具链

VS2022，xmake，
ninja，clang，
ccache，UB，IB，.....
交叉编译

升级硬件

更高主频，更多的核，更大的内存，更快的磁盘和网络
更高性能的构建机集群

注：用一点硬件成本，换来整体研发效率上的提升！

谢谢

金山办公 KINGSOFT OFFICE | 谢茂培

CPP-Summit 2022

张银奎



Boolan 首席软件专家，著名系统内核专家

著名系统内核专家，《软件调试》作者，在软件产业工作20余年，一多半时间任职于Intel公司的上海研发中心，先后在PASD、DEG、CPG、PCCG、VPG等部门工作。业余时间喜欢写作和参与各类技术会议，发文数百万字，探讨各类软件问题，其中《在调试器里看阿里的软件兵团》等文章广为流传。多次获微软全球最有价值技术专家（MVP）奖励。在多家跨国公司历任开发工程师、软件架构师、开发经理、项目经理等职务，对 IA-32 架构、操作系统内核、驱动程序、虚拟化技术、云计算、软件调优、尤其是软件调试有较深入研究。著有《软件调试》和《格蠹汇编》二书，曾经主笔《程序员》杂志调试之剑专栏 。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

异构计算与SYCL

张银奎
格蠹科技创始人

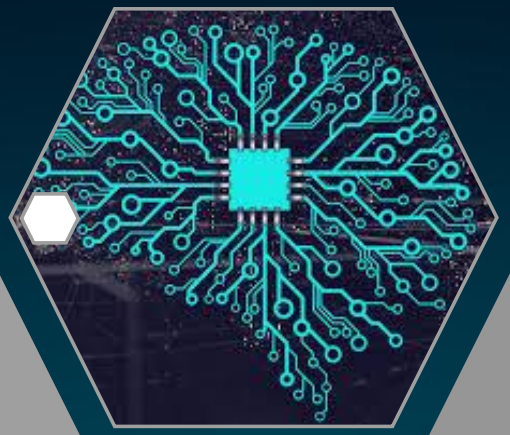
“闻道有先后，术业有专攻”

——*韩愈《师说》*

何谓异构计算?

使用不同种类的处理器或者核心一起计算





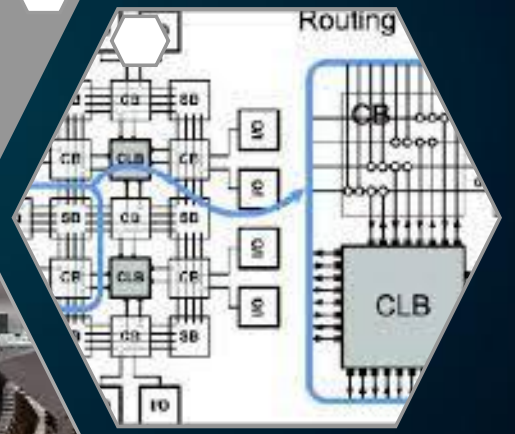
ASIC



通用处理器(GPP)

GPU

FPGA





异构计算



张银奎

2013年11月8日

庐山白鹿洞书院

必要性 → 指令转发 → 数据并行 → SYCL


intel
i386
A80386DX-33 IV
SX366
L1333356
INTEL©'85
intel
i387™DX™
A80387DX 16-33
ADS
HLDA
RDY
U28
U29
U17
U16
RP17
RP16
RP18
RP15
C62
C63
C65
C66
C67
C69
C72
C73
C78
C79
C85
C86
C87
C88
R19
R22
R23
R24
R25
D8
D7
M/IO

```
499:      int nMonths = rand();
00401C64   call        dword ptr [__imp__rand (00403240)]
00401C6A   mov         dword ptr [esp+4],eax
500:      double fIncome, fSalary = 876543.22;
501:
502:      fIncome = fSalary * nMonths;
503:
504:      D4D("Income %f for %d", fIncome, nMonths);
00401C6E   push        eax
00401C6F   fild        dword ptr [esp+8]
00401C73   sub         esp,8
00401C76   fmul        qword ptr [__real@8@4012d5fff3851eb85000 (004037d0)]
00401C7C   fstp        qword ptr [esp]
00401C7F   push        offset string "Income %f for %d" (00404300)
00401C84   push        esi
00401C85   call        CBadBoyDlg::D4D (00401990)
00401C8A   add         esp,14h
00401C8D   pop         esi
505:  }
```

5.2 X87 FPU INSTRUCTIONS

The x87 FPU instructions are executed by the processor's x87 FPU. These instructions operate on floating-point, integer, and binary-coded decimal (BCD) operands. For more detail on x87 FPU instructions, see Chapter 8, "Programming with the x87 FPU."

These instructions are divided into the following subgroups: data transfer, load constants, and FPU control instructions. The sections that follow introduce each subgroup.

5.2.1 x87 FPU Data Transfer Instructions

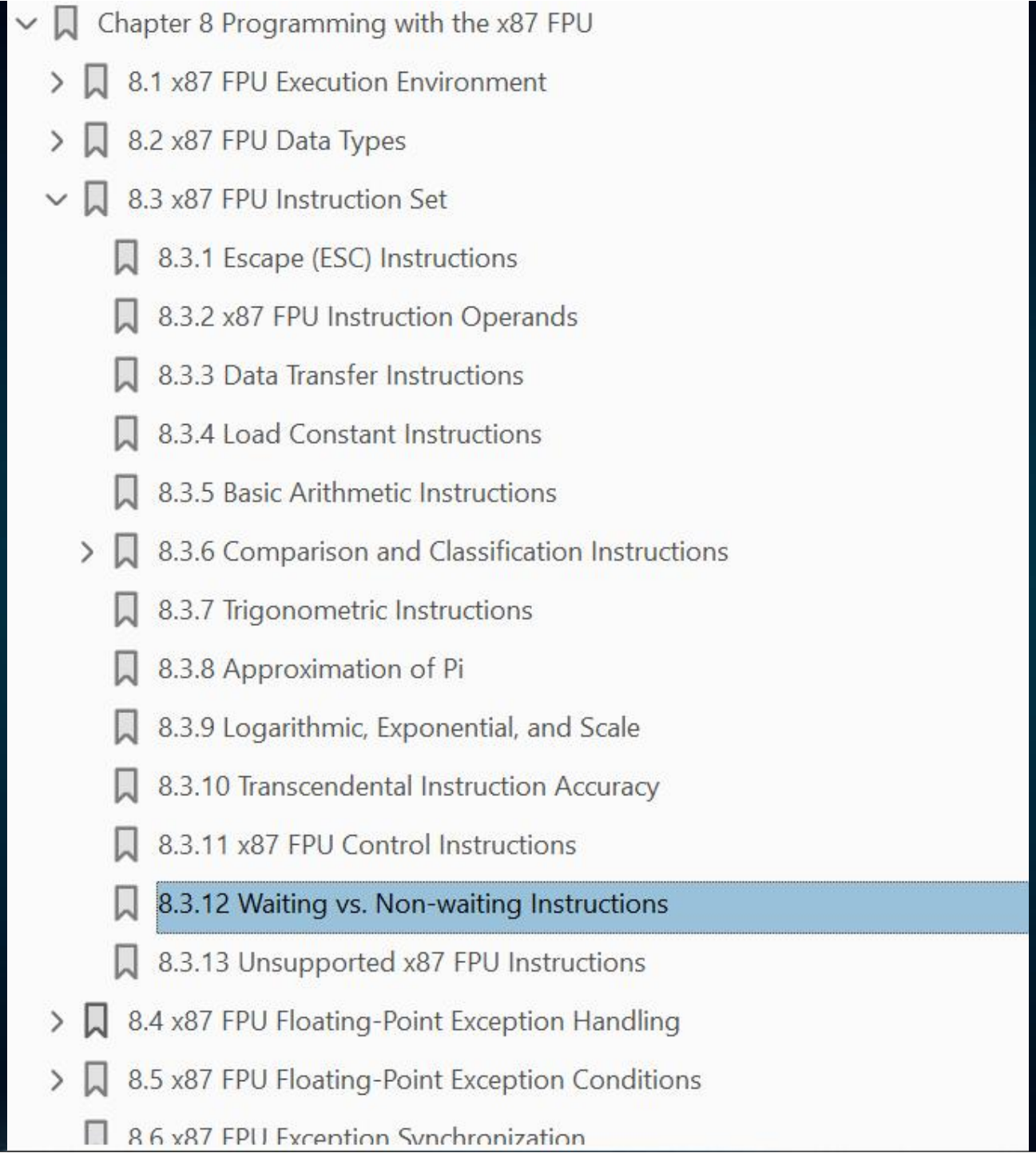
The data transfer instructions move floating-point, integer, and BCD values between memory and the x87 FPU registers. They also perform conditional move operations on floating-point operands.

FLD	Load floating-point value.
FST	Store floating-point value.
FSTP	Store floating-point value and pop.
FILD	Load integer.
FIST	Store integer.
FISTP[1]	Store integer and pop.
FBLD	Load BCD.
FBSTP	Store BCD and pop.
FXCH	Exchange registers.
FCMOVE	Floating-point conditional move if equal.
FCMOVNE	Floating-point conditional move if not equal.
FCMOVB	Floating-point conditional move if below.
FCMOVBE	Floating-point conditional move if below or equal.
FCMOVNB	Floating-point conditional move if not below.
FCMOVNBE	Floating-point conditional move if not below or equal.
FCMOVU	Floating-point conditional move if unordered.
FCMOVNU	Floating-point conditional move if not unordered.

```
499:      int nMonths = rand();
00401C64   call        dword ptr [__imp__rand (00403240)]
00401C6A   mov         dword ptr [esp+4],eax
500:      double fIncome, fSalary = 876543.22;
501:
502:      fIncome = fSalary * nMonths;
503:
504:      D4D("Income %f for %d", fIncome, nMonths);
00401C6E   push        eax
00401C6F   fild        dword ptr [esp+8]
00401C73   sub         esp,8
00401C76   fmul        qword ptr [__real@8@4012d5fff3851eb85000 (004037d0)]
00401C7C   fstp        qword ptr [esp]
00401C7F   push        offset string "Income %f for %d" (00404300)
00401C84   push        esi
00401C85   call        CBadBoyDlg::D4D (00401990)
00401C8A   add         esp,14h
00401C8D   pop         esi
505:  }
```

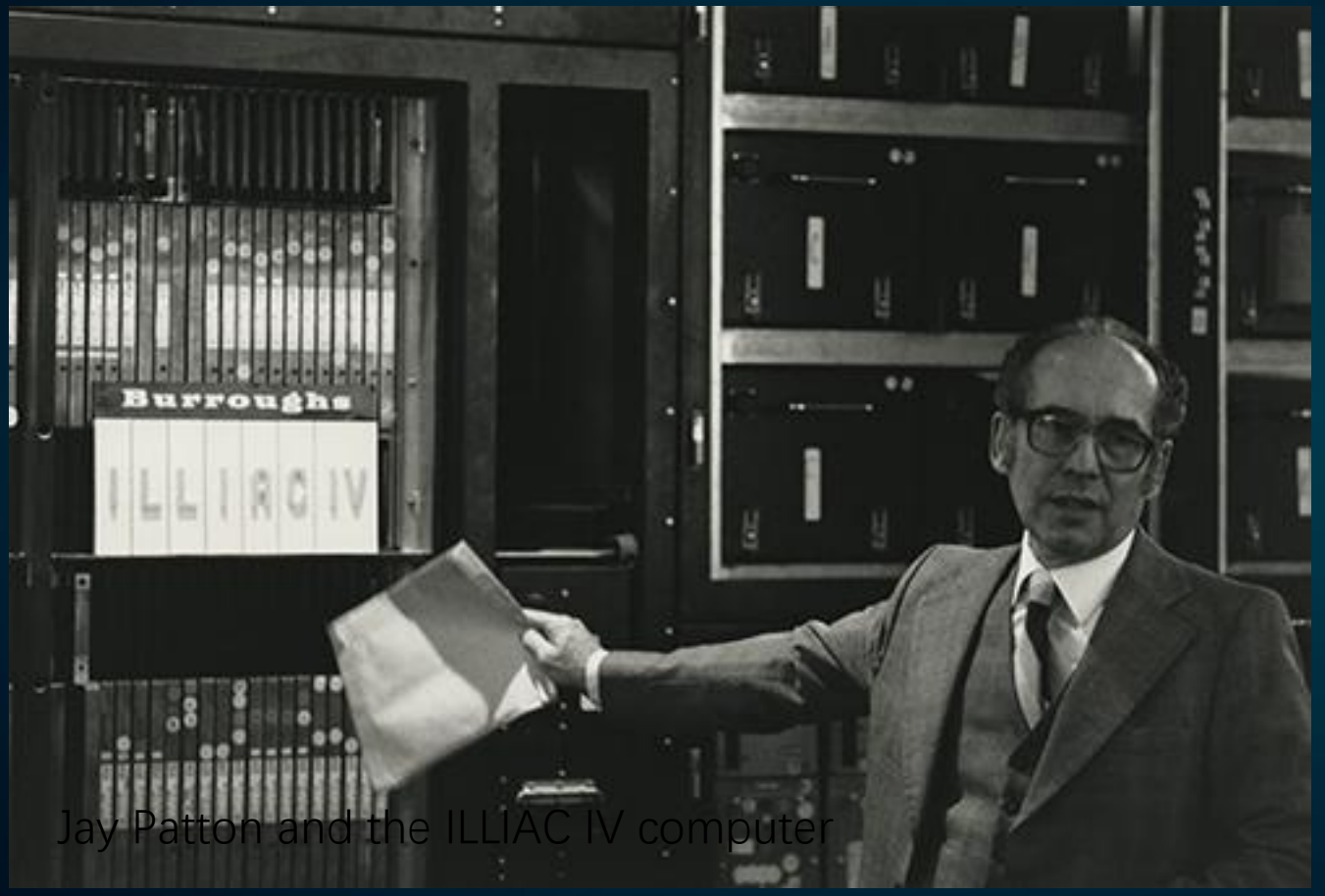


对于程序员来说，不需要感知，浑然一体


Chapter 8 Programming with the x87 FPU
8.1 x87 FPU Execution Environment
8.2 x87 FPU Data Types
8.3 x87 FPU Instruction Set
8.3.1 Escape (ESC) Instructions
8.3.2 x87 FPU Instruction Operands
8.3.3 Data Transfer Instructions
8.3.4 Load Constant Instructions
8.3.5 Basic Arithmetic Instructions
8.3.6 Comparison and Classification Instructions
8.3.7 Trigonometric Instructions
8.3.8 Approximation of Pi
8.3.9 Logarithmic, Exponential, and Scale
8.3.10 Transcendental Instruction Accuracy
8.3.11 x87 FPU Control Instructions
8.3.12 Waiting vs. Non-waiting Instructions
8.3.13 Unsupported x87 FPU Instructions
8.4 x87 FPU Floating-Point Exception Handling
8.5 x87 FPU Floating-Point Exception Conditions
8.6 x87 FPU Exception Synchronization

必要性 → 指令转发 → 数据并行 → SYCL

ILLIAC IV

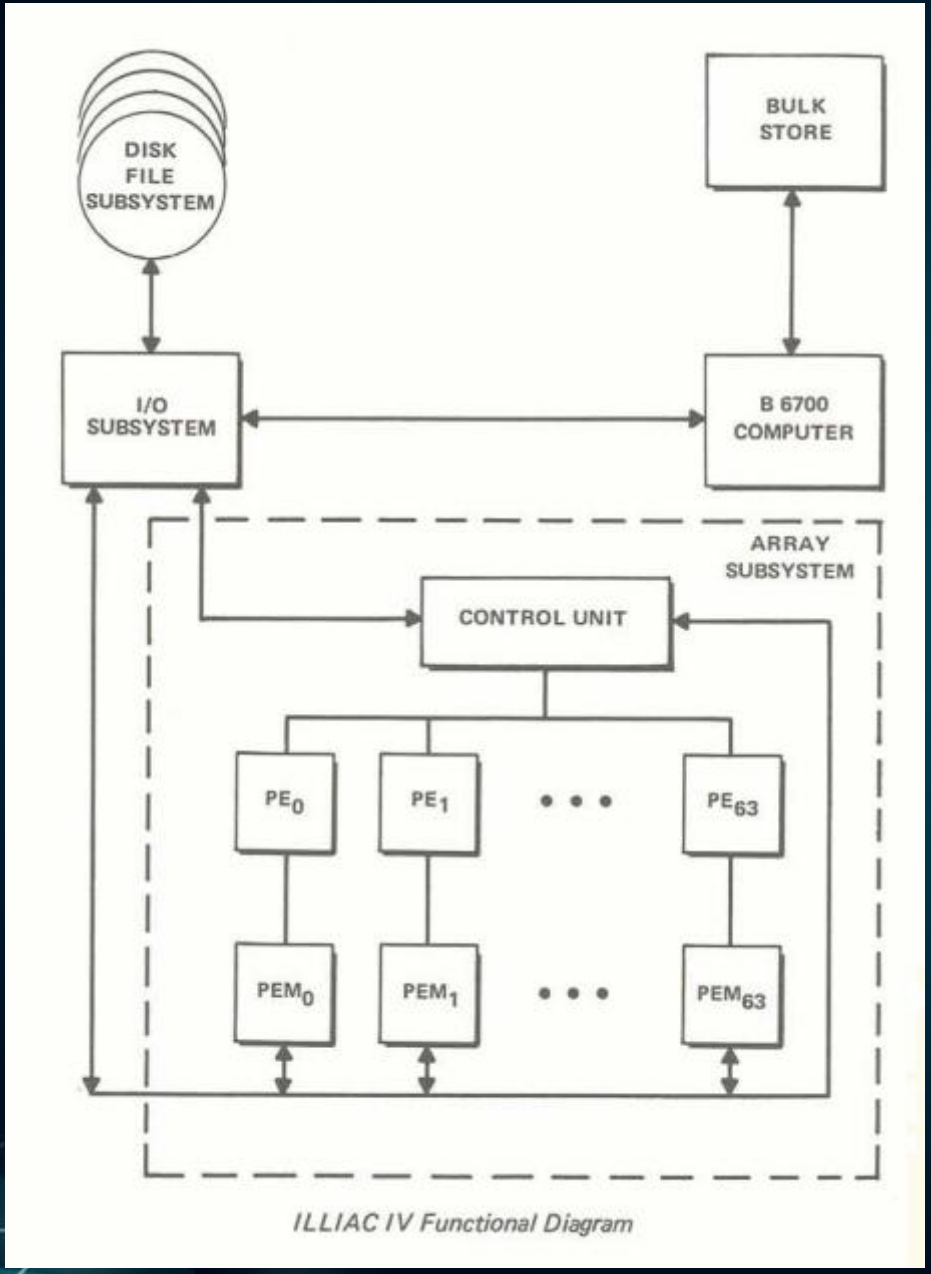


Jay Patton and the ILLIAC IV computer

- Pioneer of centrally controlled parallel computers
- Designed at the University of Illinois and built by Burroughs started in 1965, the computer took six years to complete at a cost of $40 million
- Installed at NASA, serving for 8 years until its shut-down in 1981
- It was the fastest machine then in use, but ahead of its time

Computer History Museum

Processor Elements



ILLIAC IV Functional Diagram

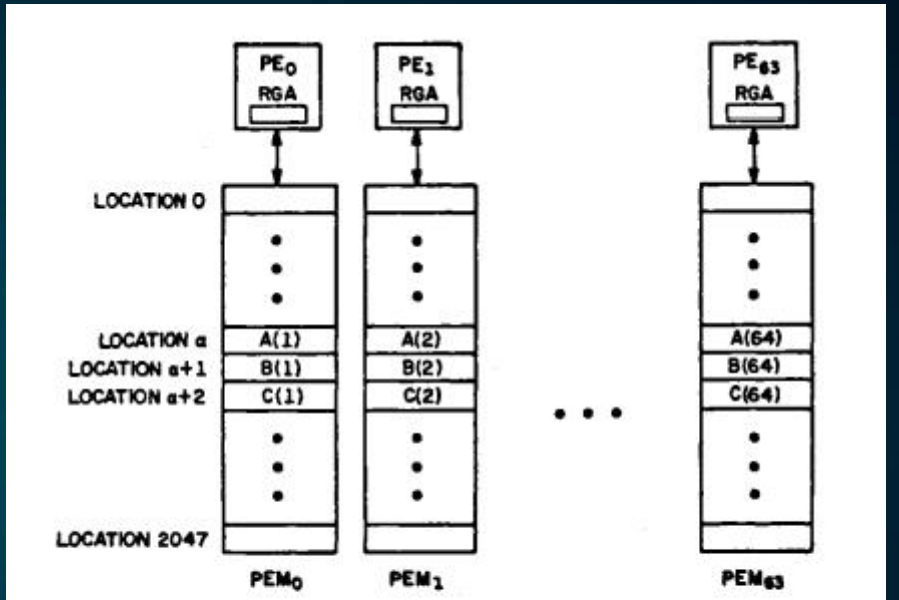


Fig. 7. Arrangement of data in PEM to accomplish

DO 10 $I = 1, 64$

10 $A(I) = B(I) + C(I)$.

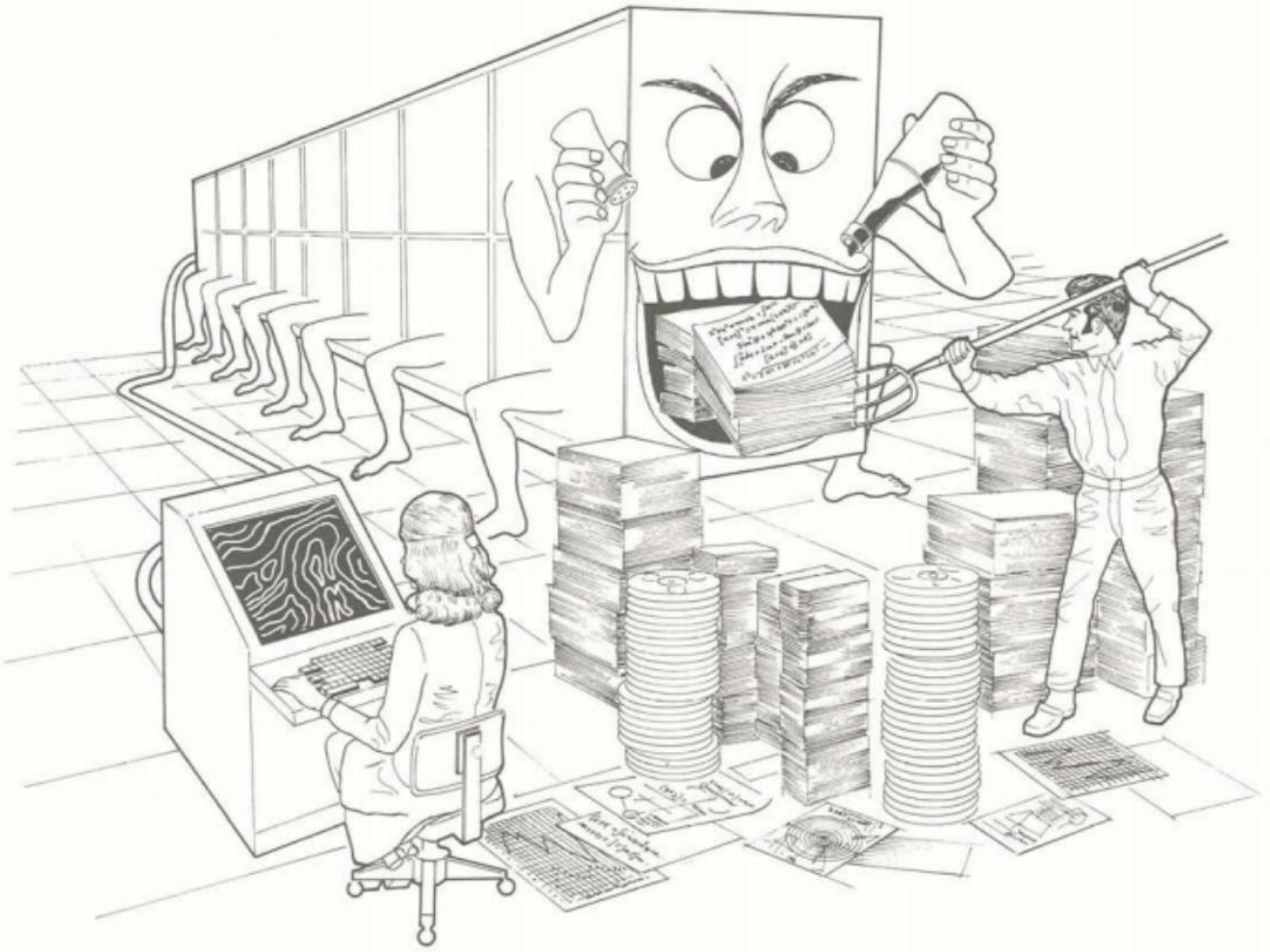
计划最多支持256个PE

ILLIAC IV实现64个PE

PEM - Processing Element Memory

nanoseconds, and on executing one multiply instruction simultaneously on 128 pairs of 32-bit operands in 625 nanoseconds.

These add and multiply times assume that the operands are already in place in the registers of the PE's. This was true for 50 percent of the add instructions and for over one third of the multiply instructions in the codes surveyed. For the other 50 percent add instructions and the more than 60 percent multiply instructions, a memory access time of 375 nanoseconds must be added. Overlap in the Control Unit will hide an undetermined fraction of this memory access time.



ILLUSTRATIVE KERNELS

To illustrate ILLIAC IV's speed of operation the times required to evaluate some simple arithmetic expressions are given.

For 64 variables,

A_0 to A_{63}, find A_j^n for all j - - - - 0.563n microsecond

Find $\sum_{k=1}^{n} A_j^k$ for all j - - - - - - - - - - - - - 0.875n microsecond
(Method of nested polynomials)

数据并行 – Single Instruction, Multiple Data



vpaddb ymm3,ymm1,ymm2

```
YMM1I0 = 0FA00BB80014000A YMM1I1 = 0FA007D017707530 YMM1I2 = B1E017700E100FA0 YMM1I3 = C56800031F409E58
YMM2I0 = 0028001E07D003E8 YMM2I1 = 8AD086E87D000BB8 YMM2I2 = B1E02328FB50F63C YMM2I3 = 5DC08002701C9688
YMM3I0 = 0FC80BD607E403F2 YMM3I1 = 99708DB8947080E8 YMM3I2 = 62C03A98096005DC YMM3I3 = 222880058F5C34E0
```

AVX packed integer add in byte

一次完成32个整数加法

Quad Word

IA上的SIMD实现



MMX

- 1997年基于P5的奔腾处理器引入
- 8个64位寄存器MM0 – MM7, 与X87复用
- 数据类型：整数



SSE

- 1999年随奔腾III引入
- 8个独立的128位寄存器XMM0 – XMM7，不依赖FPU
- 70条新指令
- 数据类型：组合和标量单双精度浮点数，组合整数(SSE2)，组合字符串



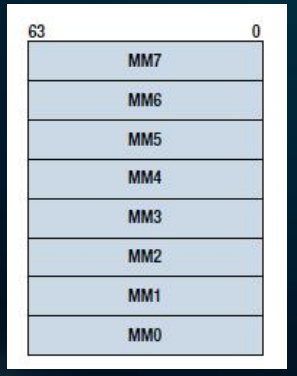
AVX

- 2008年提出，2011年发布的Sandy Bridge处理器开始支持
- 8个可以直接寻址的256位寄存器YMM0-YMM7，低128位与XMM寄存器重叠
- 三目运算符

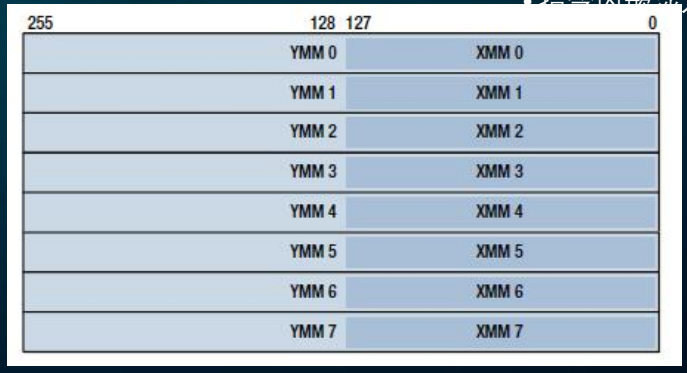


AVX-512

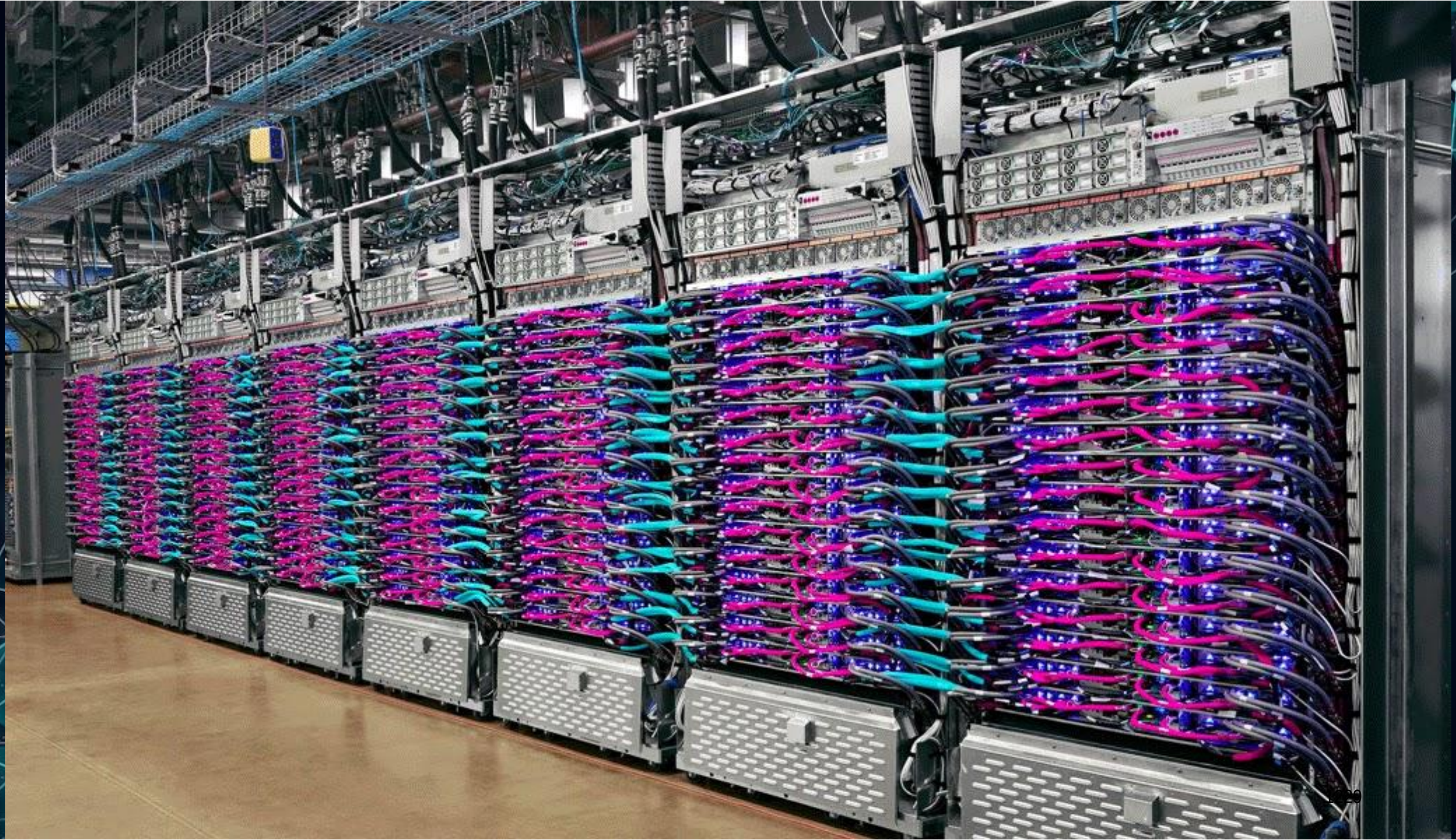
- 2013年宣布，2015年发布产品
- Many Integrated Core(MIC)
- Xeon Phi, Knights Landing
- 32个512位宽寄存器ZMM0-ZMM31
- 8个新的掩码寄存器
- 指令内嵌进入控制





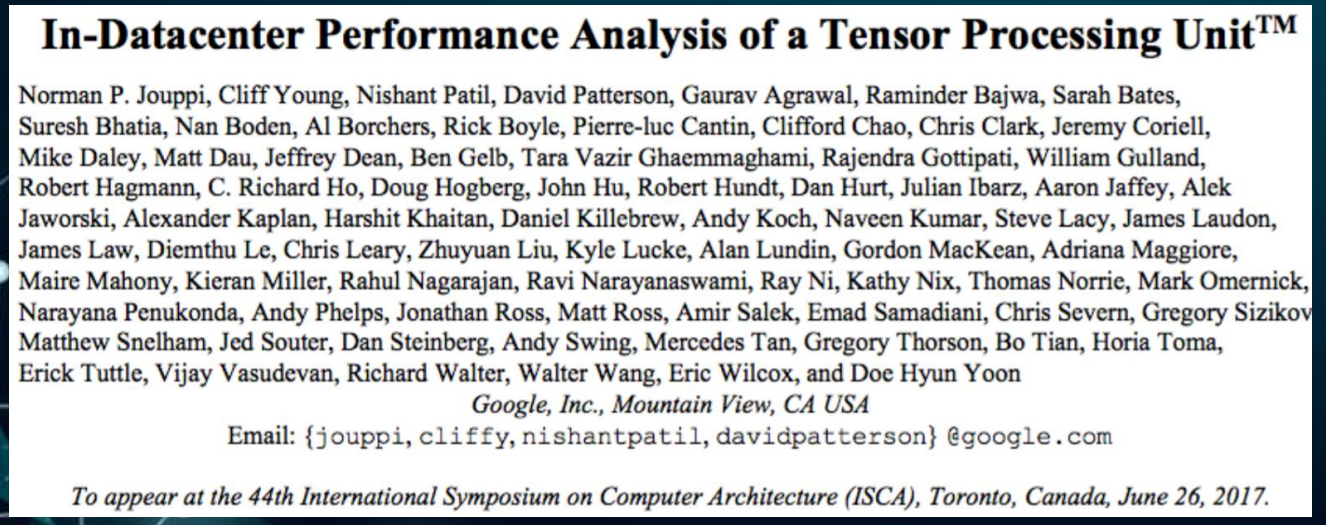


```
    for (Uint32 i = 0; i < num_pixels; i += 16)
00EDFB2B  mov         dword ptr [ebp-0BCh],0
00EDFB35  jmp         CImgWorkerIntrinsic::Threshold+176h (0EDFB46h)
00EDFB37  mov         eax,dword ptr [ebp-0BCh]
00EDFB3D  add         eax,10h
00EDFB40  mov         dword ptr [ebp-0BCh],eax
00EDFB46  mov         eax,dword ptr [ebp-0BCh]
00EDFB4C  cmp         eax,dword ptr [num_pixels]
00EDFB4F  jae         CImgWorkerIntrinsic::Threshold+1E7h (0EDFBB7h)
    {
        pipixels = *(__m128i*)(pb_src+i);
00EDFB51  mov         eax,dword ptr [pb_src]
00EDFB54  add         eax,dword ptr [ebp-0BCh]
00EDFB5A  movdqu      xmm0,xmmword ptr [eax]
00EDFB5E  movdqu      xmmword ptr [pipixels],xmm0
        pipixels = _mm_sub_epi8(pipixels, piscale);
00EDFB66  movdqa      xmm0,xmmword ptr [pipixels]
00EDFB6E  psubb       xmm0,xmmword ptr [piscale]
00EDFB73  movdqa      xmmword ptr [ebp-1C0h],xmm0
00EDFB7B  movdqa      xmm0,xmmword ptr [ebp-1C0h]
00EDFB83  movdqa      xmmword ptr [pipixels],xmm0
        *((__m128i*)(pb_mask+i)) = _mm_cmpgt_epi8(pipixels, pithreshold);
00EDFB8B  movdqa      xmm0,xmmword ptr [pipixels]
00EDFB93  pcmpgtb     xmm0,xmmword ptr [pithreshold]
00EDFB98  movdqa      xmmword ptr [ebp-1A0h],xmm0
00EDFBA0  mov         eax,dword ptr [pb_mask]
00EDFBA3  add         eax,dword ptr [ebp-0BCh]
00EDFBA9  movdqu      xmm0,xmmword ptr [ebp-1A0h]
00EDFBB1  movdqu      xmmword ptr [eax],xmm0
    }
00EDFBB5  jmp         CImgWorkerIntrinsic::Threshold+167h (0EDFB37h)
```



谷歌TPU

第一代：2016年5月（I/O大会）宣布
第二代：2017年5月宣布
第三代：2018年5月宣布
第四代：2021年5月宣布



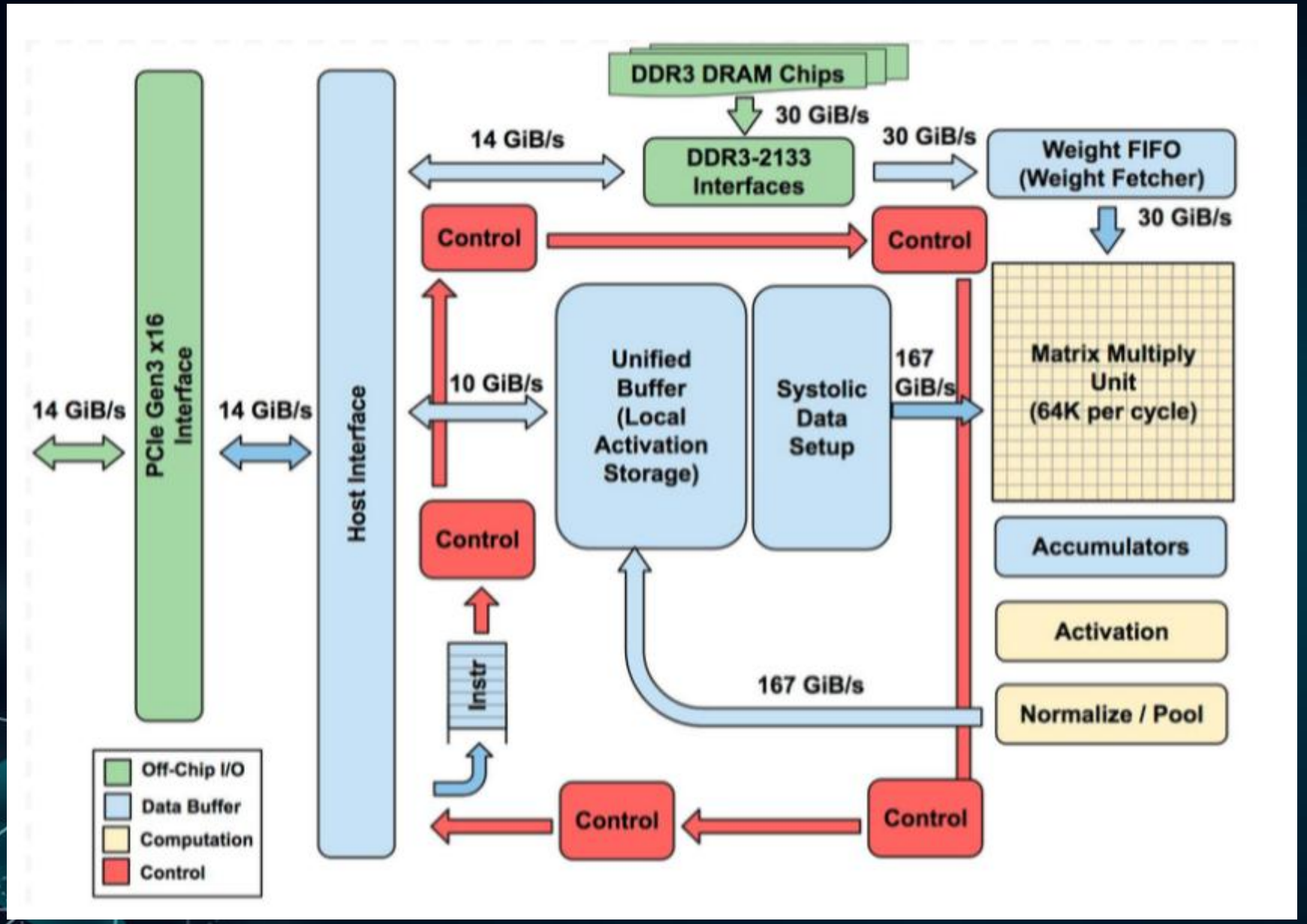
In-Datacenter Performance Analysis of a Tensor Processing Unit™

Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon

Google, Inc., Mountain View, CA USA

Email: {jouppi, cliffy, nishantpatil, davidpatterson}@google.com

To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.

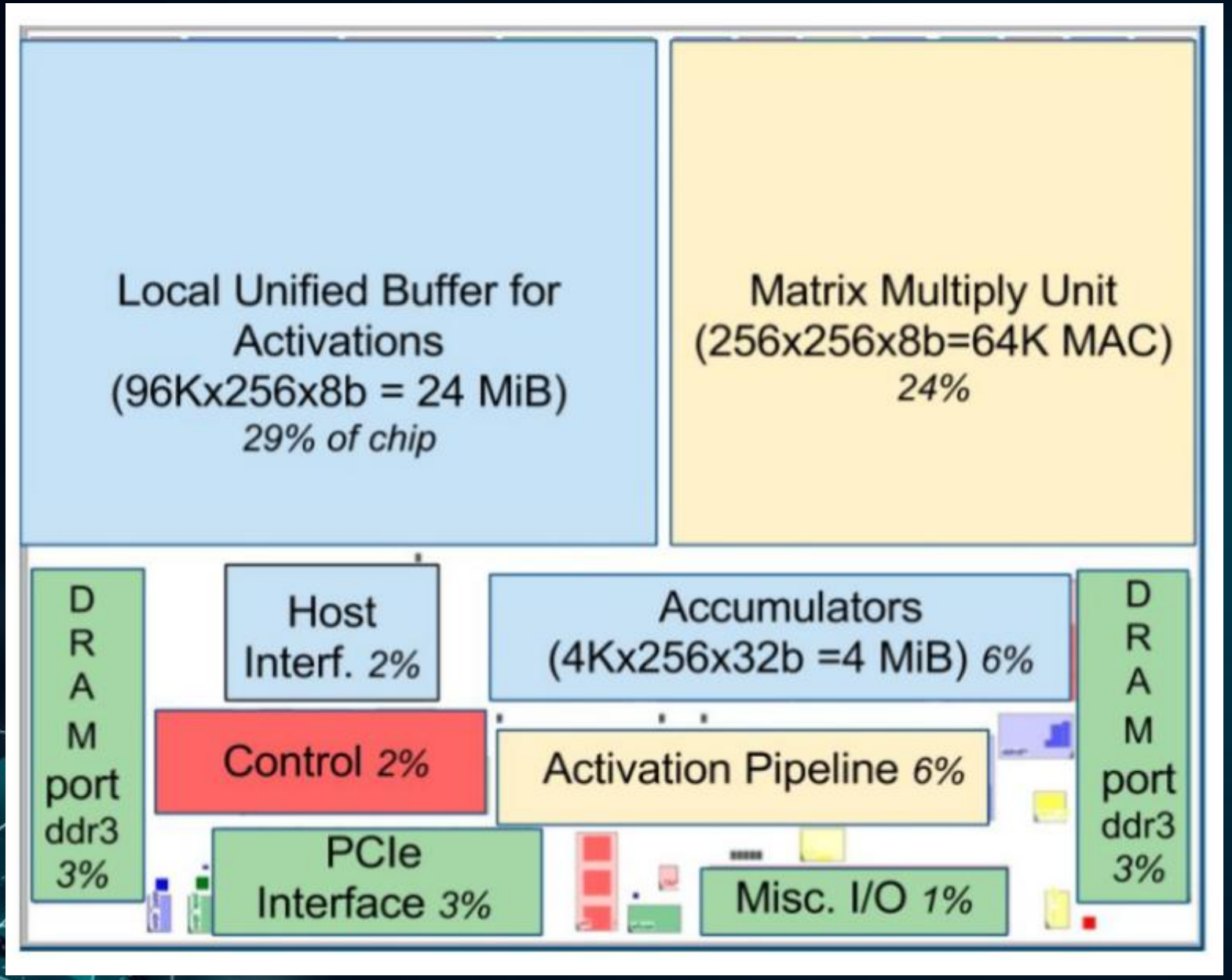


TPU逻辑框图

矩阵乘法单元是计算核心

输入为权值FIFO和Unified Buffer中的数据

输出送给激活单元和池化单元，再把数据送给UB，进行下一层计算

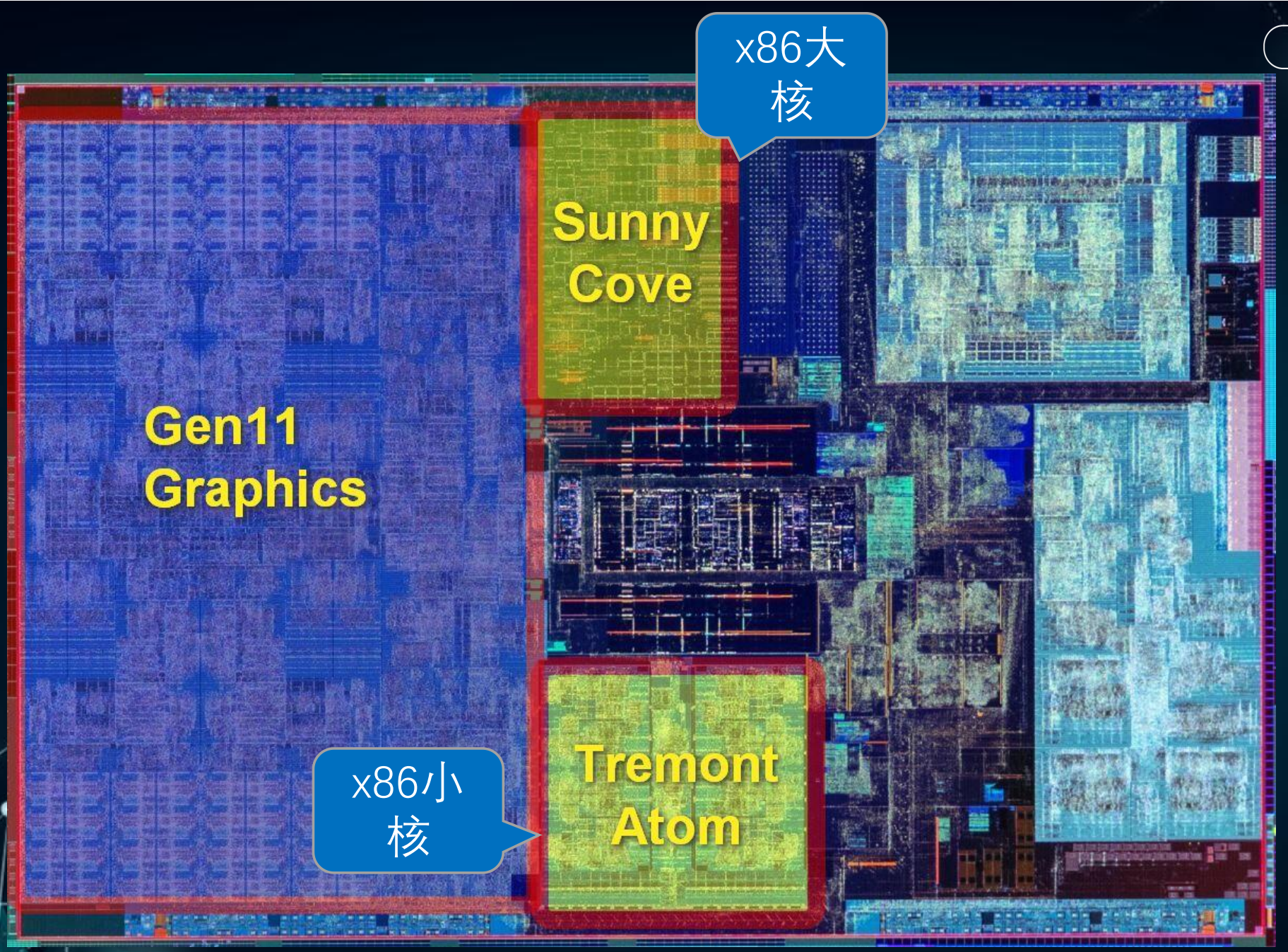


TPU的芯片平面分配 (Floor Plan)

蓝色：数据缓冲区37%
黄色：计算30%
绿色：I/O10%
红色：控制2%


2019 FOVEROS PRODUCT
POP Memory
P1274 Compute
P1222 IO/SoC
Package
Ultra-Mobile formfactor (12x12x1mm)
2mW Standby
P1222 – low cost, low leakage process for base die
P1274 – 10nm compute die
PoP memory for small form factor
Intel® Confidential. Under embargo.
2018 ARCHITECTURE DAY
intel
40

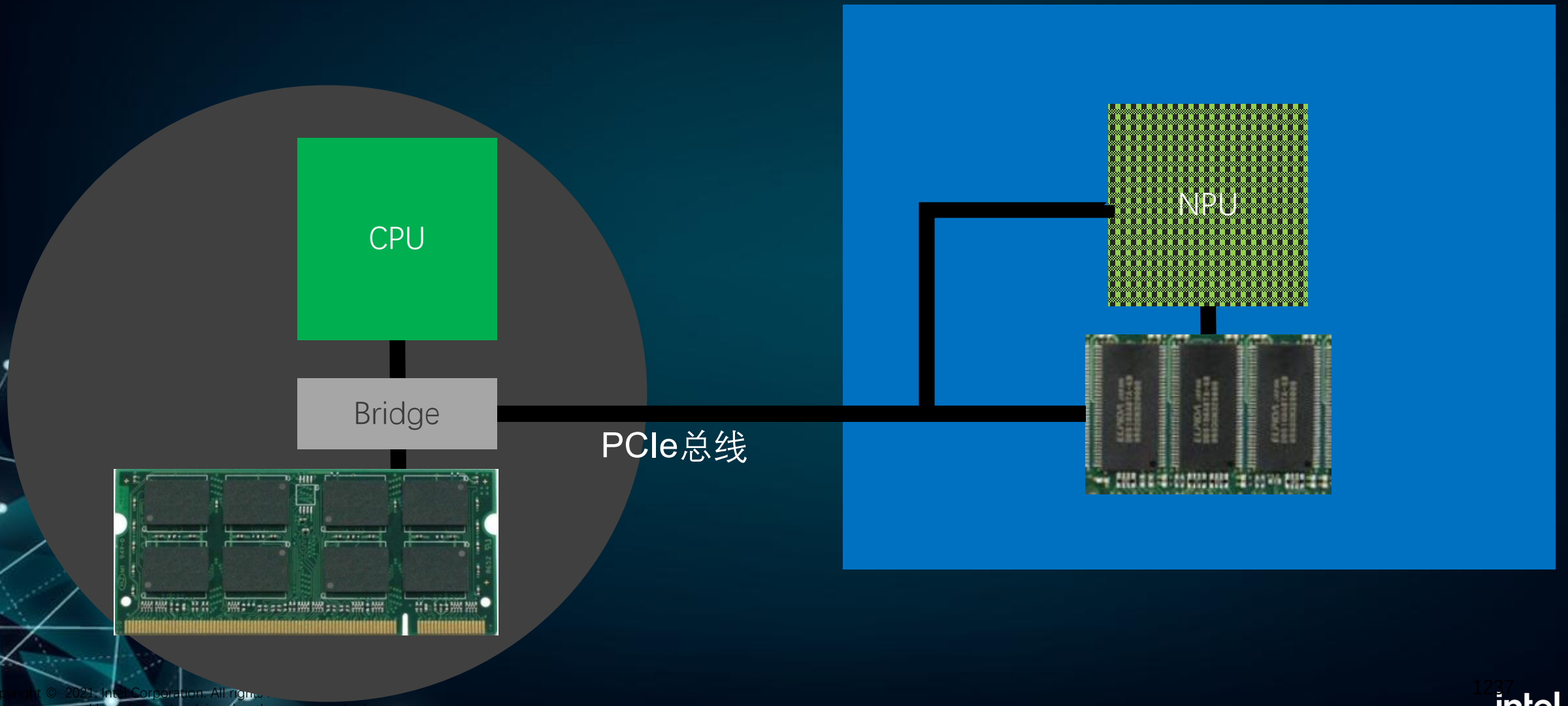
x86家族的异构平台



https://www.anandtech.com/show/15877/intel-hybrid-cpu-lakefield-all-you-need-to-know/3

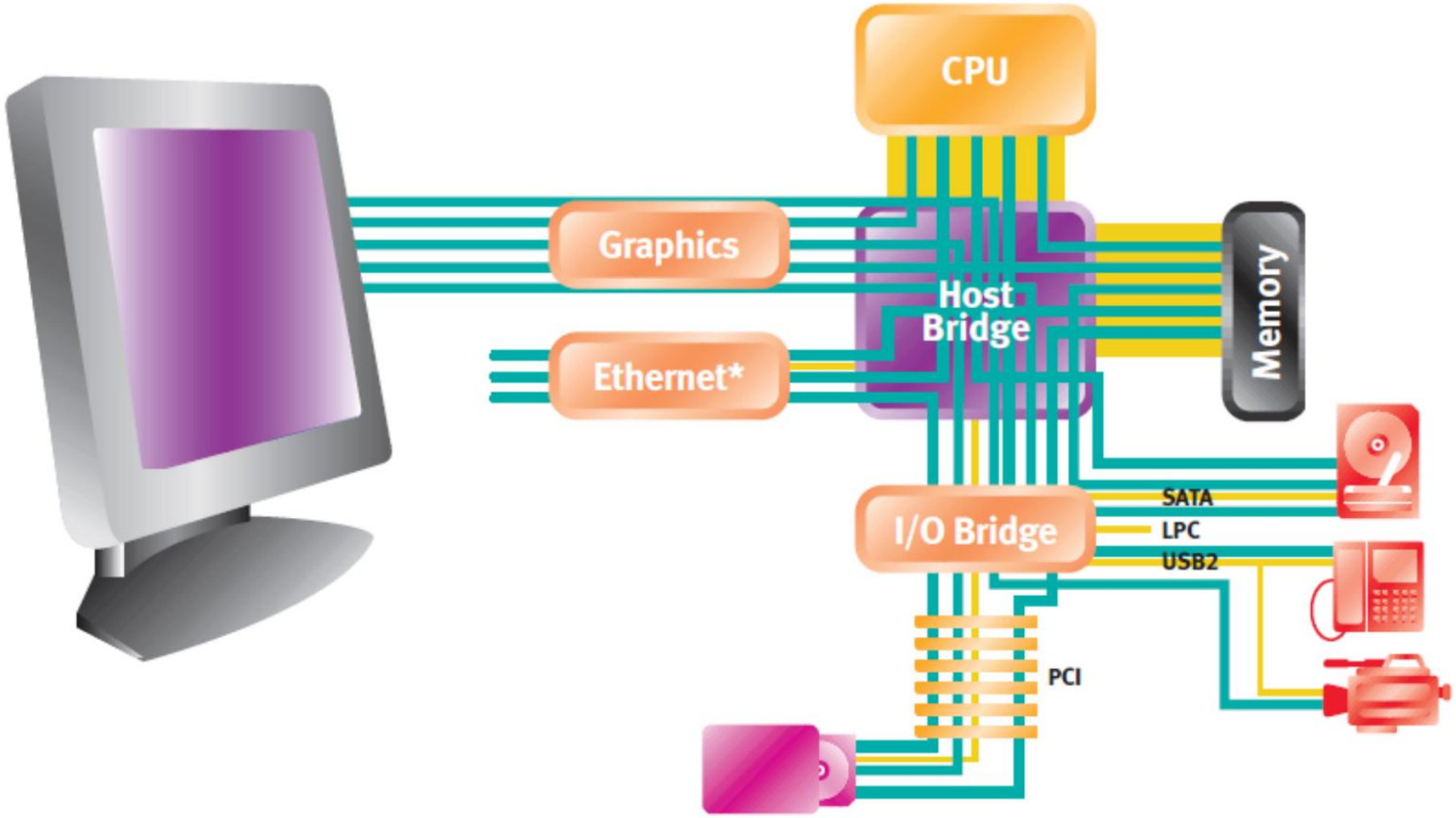
必要性 → 指令转发 → 数据并行 → oneAPI → SYCL

宿主内存与设备内存



Copyright © 2021 Intel Corporation. All rights.
*Other names and brands may be claimed as the property of others.

系统概览



Copyright © 2021, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

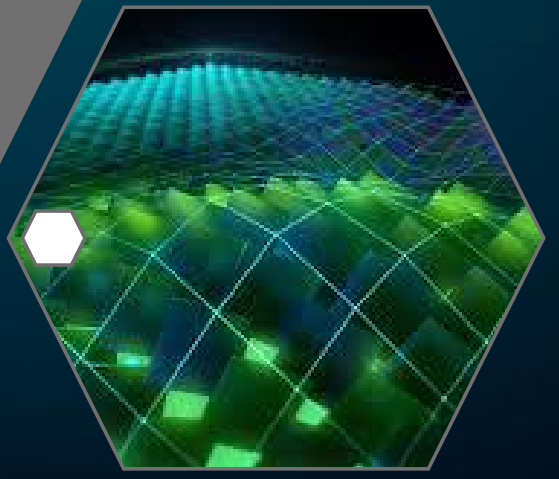


OpenCL

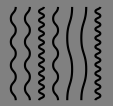


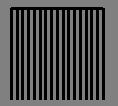
CUDA

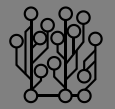
HLSL

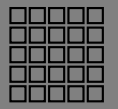


应用工作负载需采用各种不同的硬件

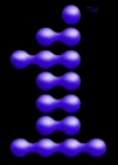

标量


矢量


空间


矩阵

中间件和框架



XPU

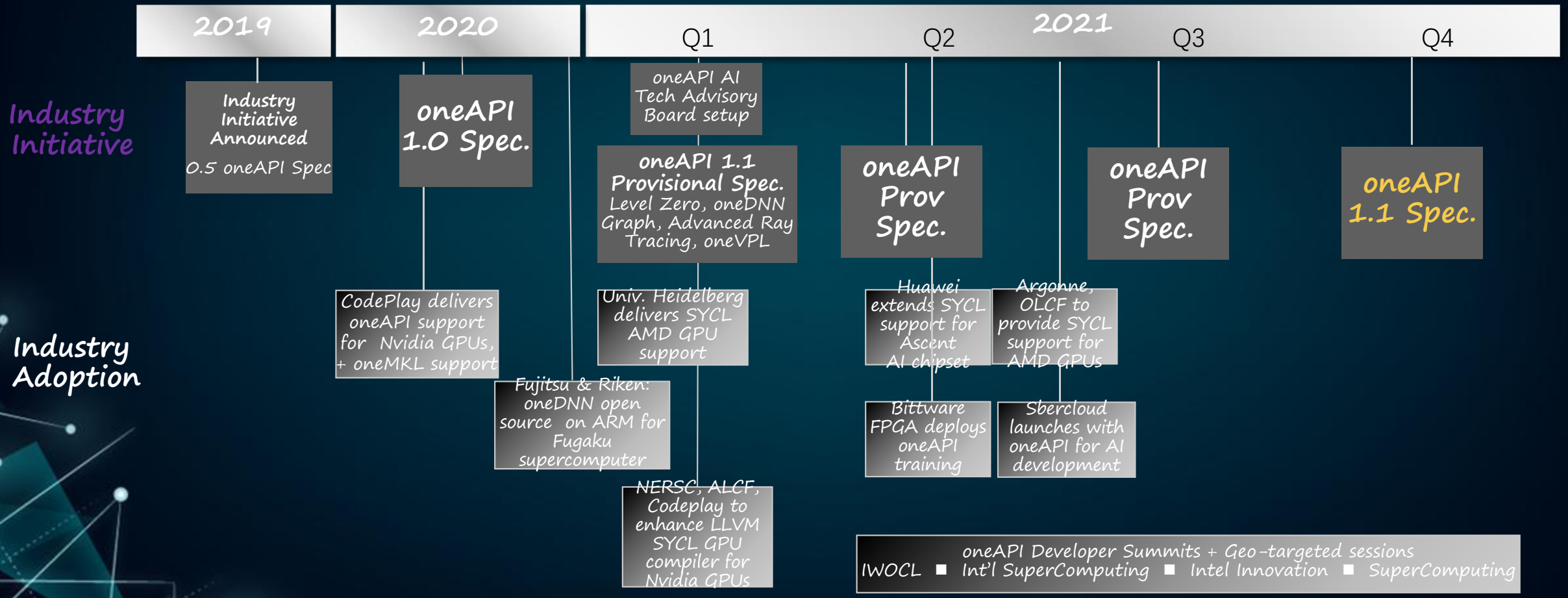

CPU


GPU


FPGA


其他加速器

oneAPI简史


2019
2020
2021
Q1
Q2
Q3
Q4
Industry Initiative
Industry Initiative Announced
0.5 oneAPI Spec
oneAPI 1.0 Spec.
oneAPI AI Tech Advisory Board setup
oneAPI 1.1 Provisional Spec.
Level Zero, oneDNN Graph, Advanced Ray Tracing, oneVPL
oneAPI Prov Spec.
oneAPI Prov Spec.
oneAPI 1.1 Spec.
Industry Adoption
CodePlay delivers oneAPI support for Nvidia GPUs, + oneMKL support
Fujitsu & Riken: oneDNN open source on ARM for Fugaku supercomputer
Univ. Heidelberg delivers SYCL AMD GPU support
NERSC, ALCF, Codeplay to enhance LLVM SYCL GPU compiler for Nvidia GPUs
Huawei extends SYCL support for Ascent AI chipset
Argonne, OLCF to provide SYCL support for AMD GPUs
Bittware FPGA deploys oneAPI training
Sbercloud launches with oneAPI for AI development
oneAPI Developer Summits + Geo-targeted sessions
IWOCL ■ Int'l SuperComputing ■ Intel Innovation ■ SuperComputing

oneAPI 生态系统

End Users

BioDataAnalysis GmbH, BENTLEY, Accrad Accelerated Radiology, archanan, LAIKA, EURECOM, GE, Brightskies, PHILIPS, 大势智慧 DASPATIAL, SAMSUNG MEDISON, TANGENTSTUDIOS, allegro.ai, ILLUMINATION MACGUFF

National Labs

Argonne NATIONAL LABORATORY, CINECA, CERN openlab, Laboratório Nacional de Computação Científica

ISVs & OSVs

AI SINGAPORE, codeplay, CANONICAL, CERTIFACE, CGG, SAP, ANSYS, SAS, FOUNDRY, mercenaries engineering Guerilla, SENAI CIMATEC FIEB, CHAOSGROUP, MAXON, GIGASPACES innovate with confidence, MEGH COMPUTING, KFBIO, Tech Mahindra, AsiaInfo 亚信科技, SUSE, UNITED IMAGING

OEMs & SIs

DELL Technologies, Atos, Hewlett Packard Enterprise, HCL, BittWare a molex company, Lenovo, RENIAC, MEGWARE

Universities & Research Institutes

Ben-Gurion University of the Negev, UNIVERSITY OF OREGON, TÉCNICO LISBOA, inesc id lisboa, ZUSE INSTITUTE BERLIN, Stockholm University, Leibniz-Rechenzentrum, LOBACHEVSKY UNIVERSITY, सी डैक CDAC, UNIVERSIDAD DE MÁLAGA, UNIVERSITY OF CAMBRIDGE, University College London, ILLINOIS Theoretical and computational Biophysics Group, KTH

Indian Institutes of Technology Delhi / Kharagpur / Roorkee

Indian Institute of Science Bangalore

IISER PUNE Indian Institute of Science Education & Research Pune

CSPs & Frameworks

Microsoft Azure, Alibaba Cloud, Google Cloud, TensorFlow, Taboola, Tencent 腾讯, SBER BANK, PyTorch

这些组织支持 oneAPI 计划提出的统一的编程模型“概念”，以便顺利地进行跨架构开发。
它并不表示任何购买或使用英特尔产品的协议。* 其他的名称和品牌可能是其他所有者的资产。

英特尔® oneAPI套件

基于英特尔丰富的 CPU工具传承并扩展到 XPU 架构

一套完整的高级编译器、库以及移植、分析和调试器工具

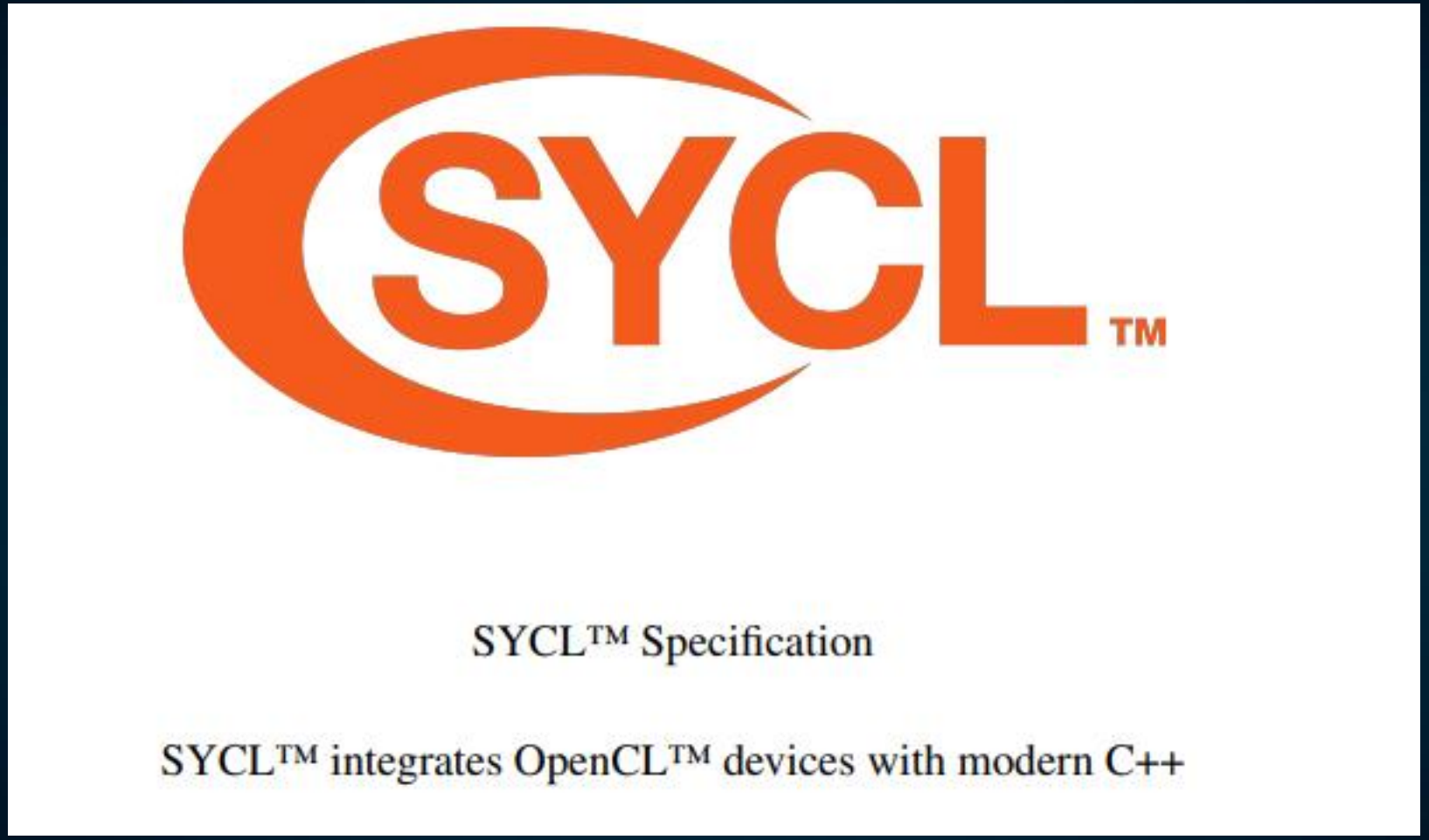
- 利用最先进的硬件功能加速计算
- 可与现有的编程模型和代码库（C++、Fortran、Python、OpenMP 等）互操作，开发人员可放心将现有应用与 oneAPI 无缝协作
- 易于过渡到新系统和加速器 --使用单一代码库让开发人员有更多时间投入创新

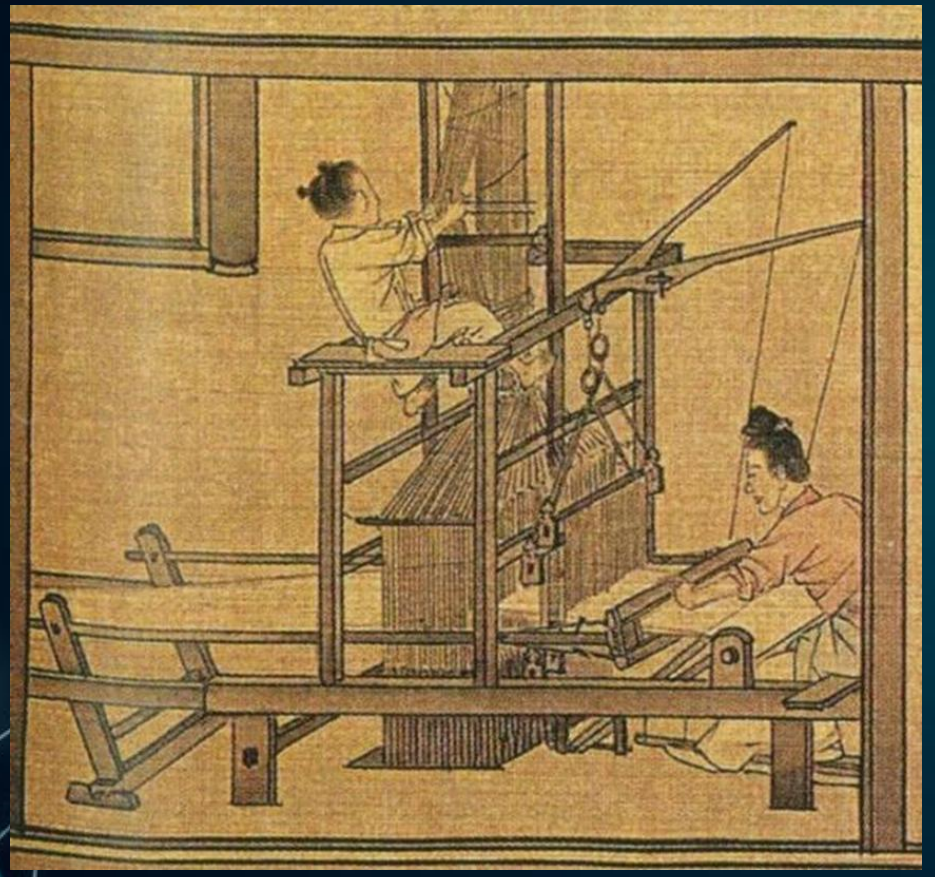


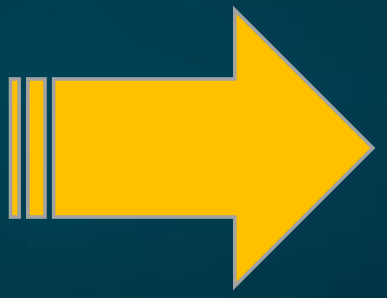
现已上市

更多详情请访问 software.intel.com/oneapi
部分功能可能因架构不同而有所区别，需要进行自定义调试。未来将支持其他加速器。

必要性 → 指令转发 → 数据并行 → oneAPI → SYCL


SYCL™
SYCL™ Specification
SYCL™ integrates OpenCL™ devices with modern C++





SYCL™

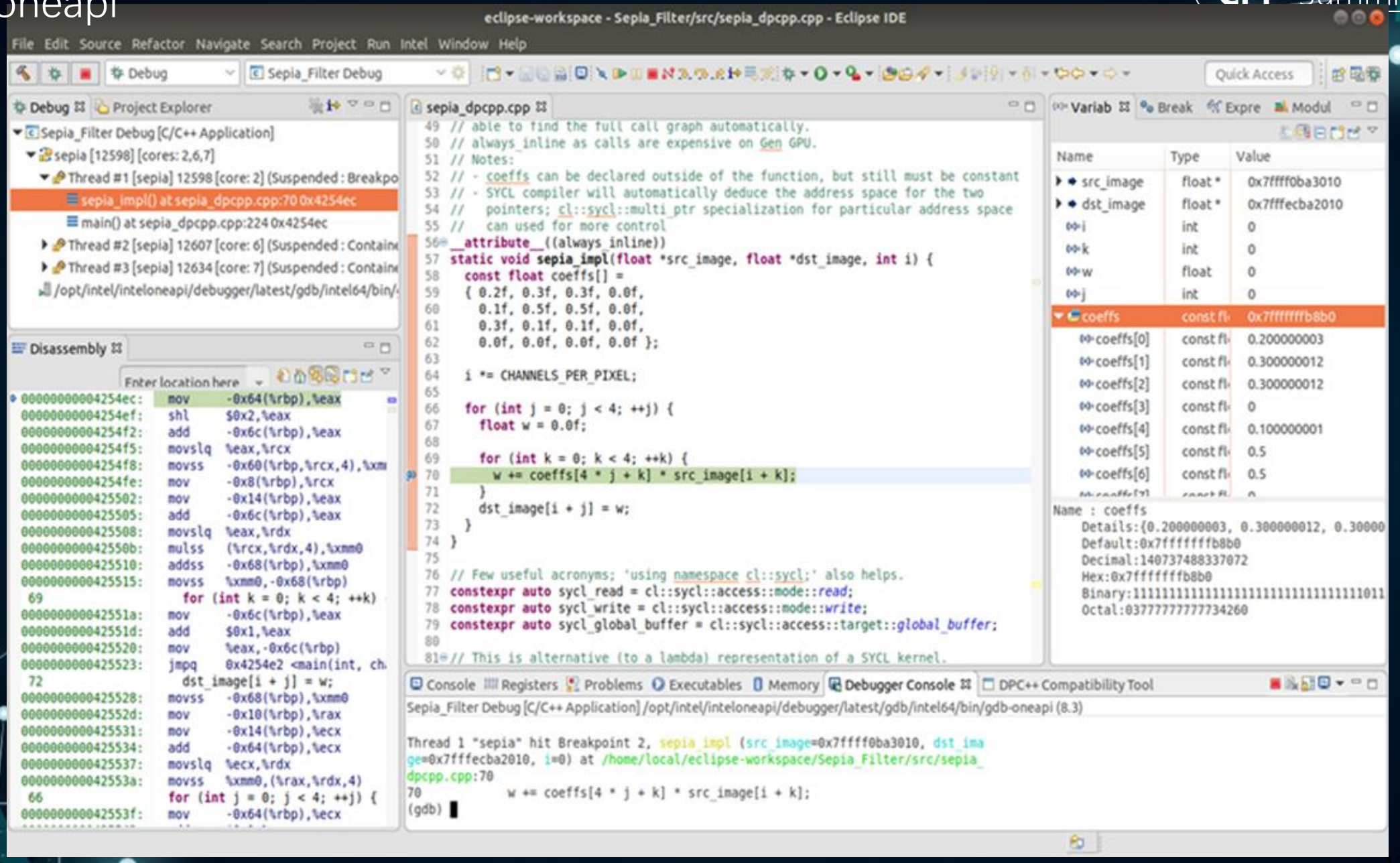
SYCL™ Specification

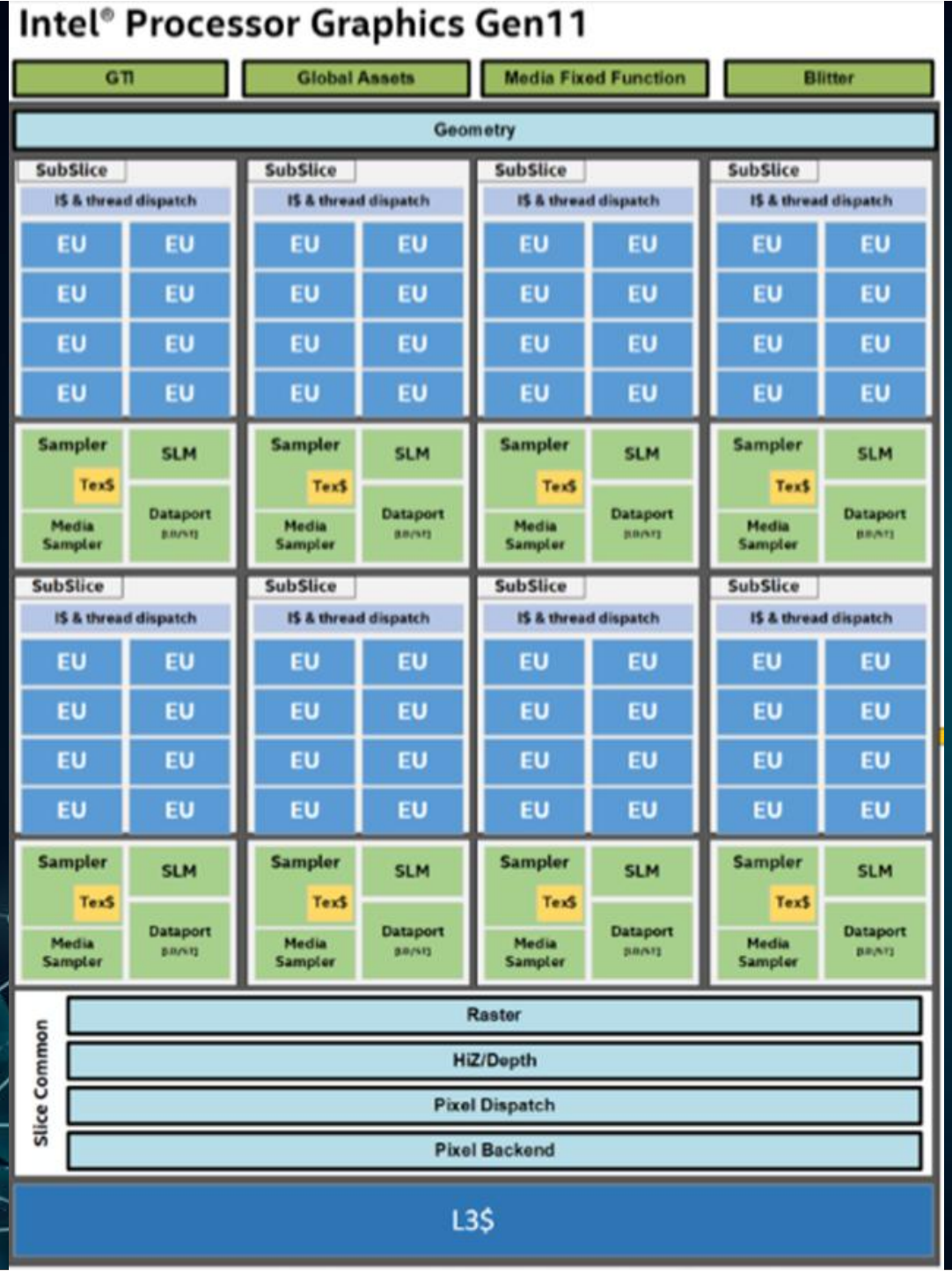
SYCL™ integrates OpenCL™ devices with modern C++

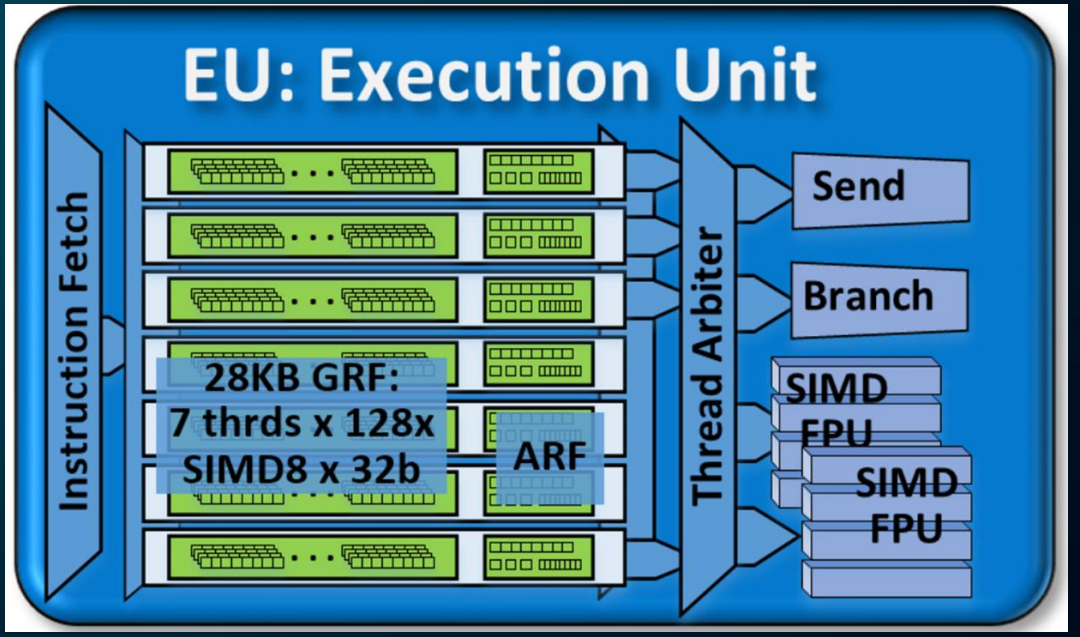
SYCL 2020 (1.2)



Gdb-oneapi


eclipse-workspace - Sepia_Filter/src/sepia_dpcpp.cpp - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Intel Window Help
Debug
Sepia_Filter Debug
Quick Access
Debug
Project Explorer
Sepia_Filter Debug [C/C++ Application]
sepia [12598] [cores: 2,6,7]
Thread #1 [sepia] 12598 [core: 2] (Suspended : Breakpo
sepia_impl() at sepia_dpcpp.cpp:70 0x4254ec
main() at sepia_dpcpp.cpp:224 0x4254ec
Thread #2 [sepia] 12607 [core: 6] (Suspended : Containe
Thread #3 [sepia] 12634 [core: 7] (Suspended : Containe
/opt/intel/inteloneapi/debugger/latest/gdb/intel64/bin/
Disassembly
Enter location here
0000000004254ec: mov -0x64(%rbp),%eax
0000000004254ef: shl $0x2,%eax
0000000004254f2: add -0x6c(%rbp),%eax
0000000004254f5: movslq %eax,%rcx
0000000004254f8: movss -0x60(%rbp,%rcx,4),%xm
0000000004254fe: mov -0x8(%rbp),%rcx
0000000000425502: mov -0x14(%rbp),%eax
0000000000425505: add -0x6c(%rbp),%eax
0000000000425508: movslq %eax,%rdx
000000000042550b: mulss (%rcx,%rdx,4),%xmm0
0000000000425510: addss -0x68(%rbp),%xmm0
0000000000425515: movss %xmm0,-0x68(%rbp)
69 for (int k = 0; k < 4; ++k)
000000000042551a: mov -0x6c(%rbp),%eax
000000000042551d: add $0x1,%eax
0000000000425520: mov %eax,-0x6c(%rbp)
0000000000425523: jmpq 0x4254e2 <main(int, ch
72 dst_image[i + j] = w;
0000000000425528: movss -0x68(%rbp),%xmm0
000000000042552d: mov -0x10(%rbp),%rax
0000000000425531: mov -0x14(%rbp),%ecx
0000000000425534: add -0x64(%rbp),%ecx
0000000000425537: movslq %ecx,%rdx
000000000042553a: movss %xmm0,(%rax,%rdx,4)
66 for (int j = 0; j < 4; ++j) {
000000000042553f: mov -0x64(%rbp),%ecx
sepia_dpcpp.cpp
49 // able to find the full call graph automatically.
50 // always_inline as calls are expensive on Gen GPU.
51 // Notes:
52 // - coeffs can be declared outside of the function, but still must be constant
53 // - SYCL compiler will automatically deduce the address space for the two
54 // pointers; cl::sycl::multi_ptr specialization for particular address space
55 // can used for more control
56 __attribute__((always_inline))
57 static void sepia_impl(float *src_image, float *dst_image, int i) {
58 const float coeffs[] =
59 { 0.2f, 0.3f, 0.3f, 0.0f,
60 0.1f, 0.5f, 0.5f, 0.0f,
61 0.3f, 0.1f, 0.1f, 0.0f,
62 0.0f, 0.0f, 0.0f, 0.0f };
63
64 i *= CHANNELS_PER_PIXEL;
65
66 for (int j = 0; j < 4; ++j) {
67 float w = 0.0f;
68
69 for (int k = 0; k < 4; ++k) {
70 w += coeffs[4 * j + k] * src_image[i + k];
71 }
72 dst_image[i + j] = w;
73 }
74 }
75
76 // Few useful acronyms; 'using namespace cl::sycl;' also helps.
77 constexpr auto sycl_read = cl::sycl::access::mode::read;
78 constexpr auto sycl_write = cl::sycl::access::mode::write;
79 constexpr auto sycl_global_buffer = cl::sycl::access::target::global_buffer;
80
81 // This is alternative (to a lambda) representation of a SYCL kernel.
Variab
Break
Expre
Modul
Name
Type
Value
src_image
float *
0x7ffff0ba3010
dst_image
float *
0x7fffecba2010
i
int
0
k
int
0
w
float
0
j
int
0
coeffs
const fl
0x7fffffffb8b0
coeffs[0]
const fl
0.200000003
coeffs[1]
const fl
0.300000012
coeffs[2]
const fl
0.300000012
coeffs[3]
const fl
0
coeffs[4]
const fl
0.100000001
coeffs[5]
const fl
0.5
coeffs[6]
const fl
0.5
Name : coeffs
Details:{0.200000003, 0.300000012, 0.30000
Default:0x7fffffffb8b0
Decimal:140737488337072
Hex:0x7fffffffb8b0
Binary:11111111111111111111111111111011
Octal:03777777777734260
Console
Registers
Problems
Executables
Memory
Debugger Console
DPC++ Compatibility Tool
Sepia_Filter Debug [C/C++ Application] /opt/intel/inteloneapi/debugger/latest/gdb/intel64/bin/gdb-oneapi (8.3)
Thread 1 "sepia" hit Breakpoint 2, sepia_impl (src_image=0x7ffff0ba3010, dst_ima
ge=0x7fffecba2010, i=0) at /home/local/eclipse-workspace/Sepia_Filter/src/sepia_
dpcpp.cpp:70
70 w += coeffs[4 * j + k] * src_image[i + k];
(gdb)


Intel® Processor Graphics Gen11
GTI
Global Assets
Media Fixed Function
Blitter
Geometry
SubSlice
I$ & thread dispatch
EU
Sampler
Tex$
Media Sampler
SLM
Dataport
Slice Common
Raster
HiZ/Depth
Pixel Dispatch
Pixel Backend
L3$


EU: Execution Unit
Instruction Fetch
28KB GRF:
7 thrds x 128x
SIMD8 x 32b
ARF
Thread Arbiter
Send
Branch
SIMD FPU
SIMD FPU

cuda表达的向量加法




```
__global__ void
vectorAdd(const float *A, const float *B, float *C, int numElements)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, numElements);
```

Hello World! with Device Code




```
__global__ void mykernel(void) {
}
```

- CUDA C/C++ keyword `__global__` indicates a function that:
  - Runs on the device
  - Is called from host code

- `nvcc` separates source code into host and device components
  - Device functions (e.g. `mykernel()`) processed by NVIDIA compiler
  - Host functions (e.g. `main()`) processed by standard host compiler
    - `gcc, cl.exe`

虽然可以写在一个.cu文件里，但是其实是当作两种语言来编译的

© NVIDIA Corporation 2011

```
//************************************
// Vector add in SYCL on device: returns sum in 4th parameter "sum".
//************************************
void VectorAdd(queue &q, const int *a, const int *b, int *sum, size_t size) {
        // Create the range object for the arrays.
        range<1> num_items{size};

        // Use parallel_for to run vector addition in parallel on device. This
        // executes the kernel.
        //    1st parameter is the number of work items.
        //    2nd parameter is the kernel, a lambda that specifies what to do per
        //    work item. the parameter of the lambda is the work item id.
        // SYCL supports unnamed lambda kernel by default.
        auto e = q.parallel_for(num_items, [=](auto i) { sum[i] = a[i] + b[i]; });

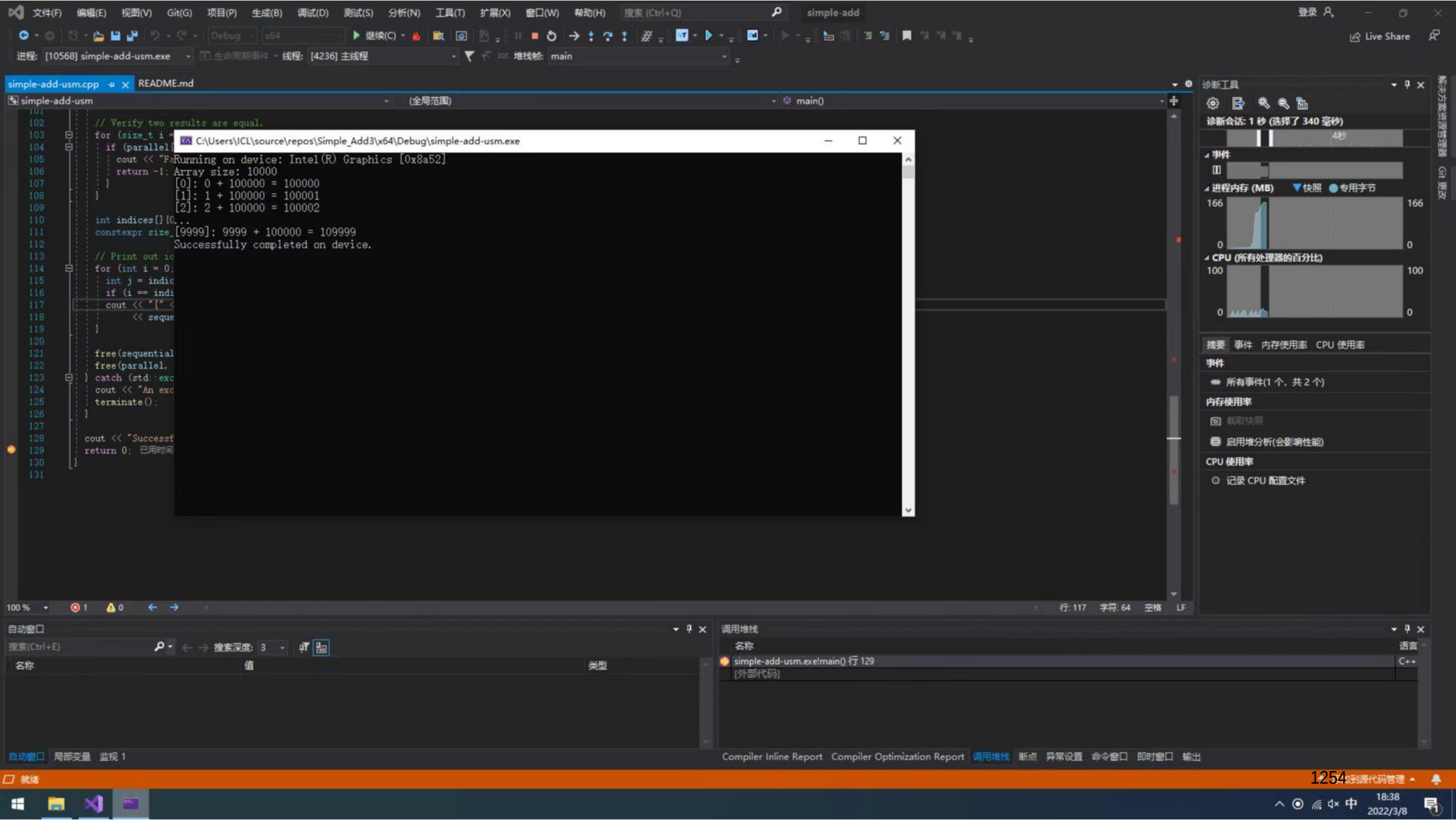
        // q.parallel_for() is an asynchronous call. SYCL runtime enqueues and runs
        // the kernel asynchronously. Wait for the asynchronous call to complete.
        e.wait();
}
```

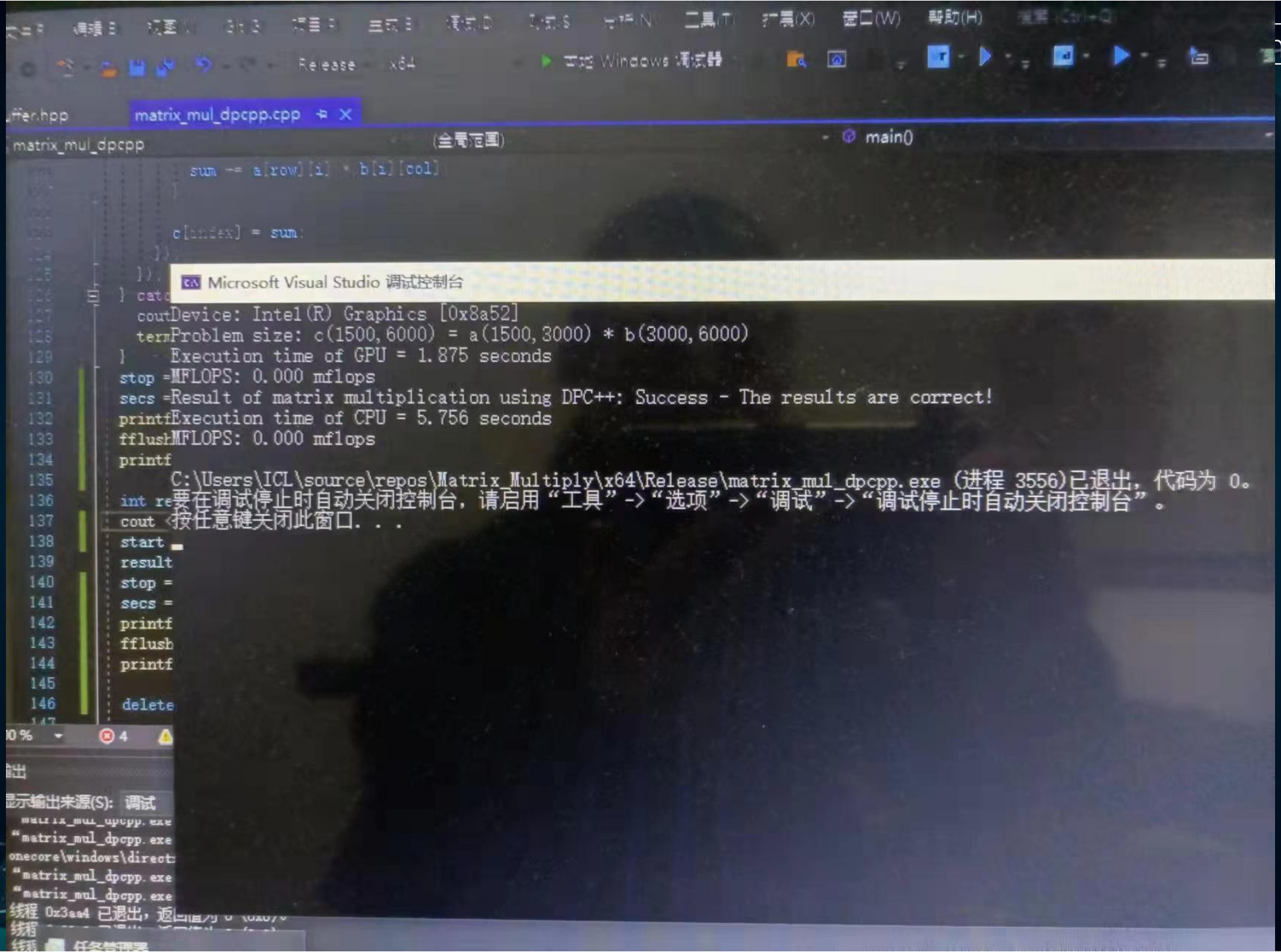
去掉注释后

```
void VectorAdd(queue &q, const int *a, const int *b, int *sum, size_t size) {
        range<1> num_items{size};

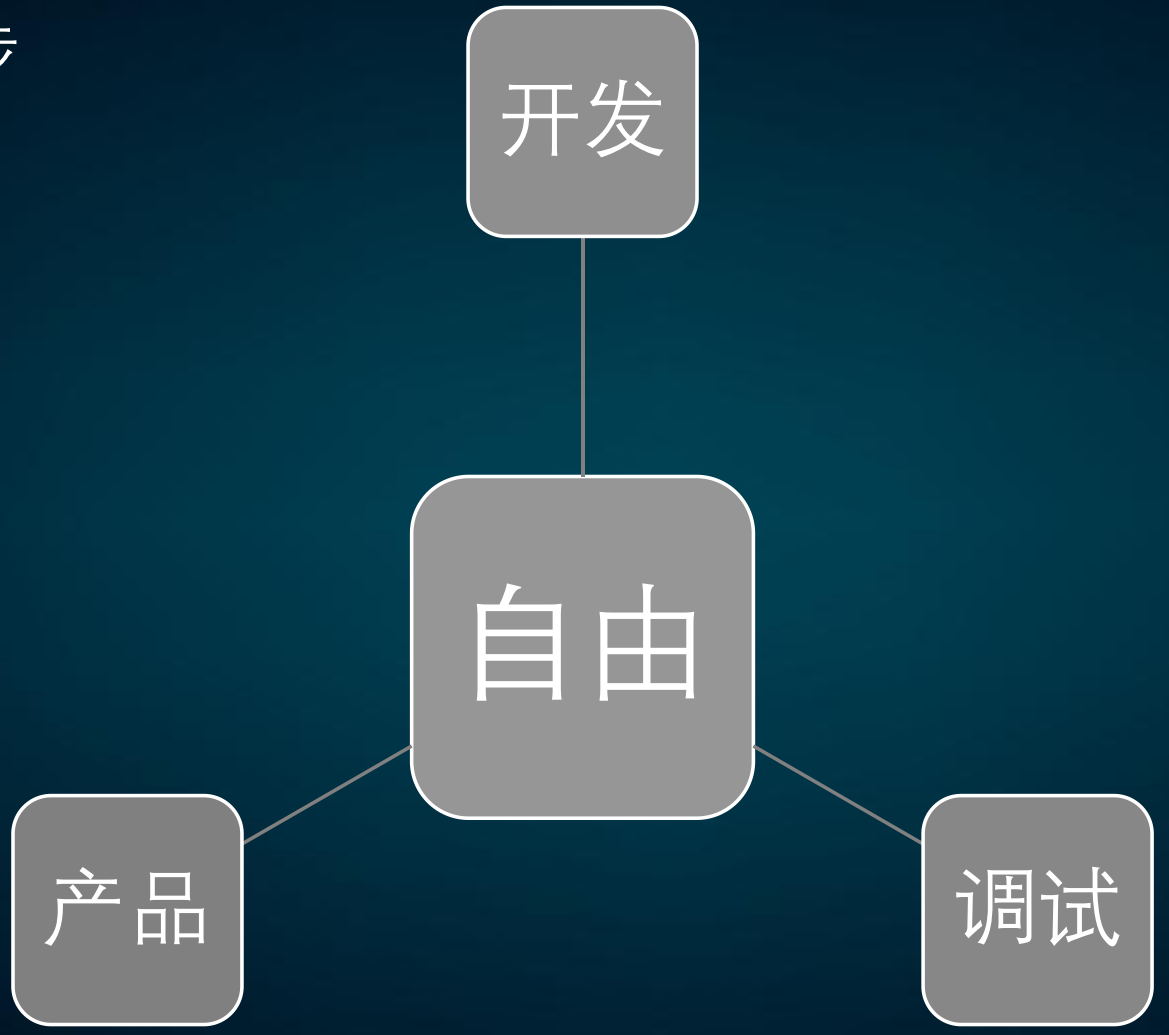
        auto e = q.parallel_for(num_items, [=](auto i) { sum[i] = a[i] + b[i]; });

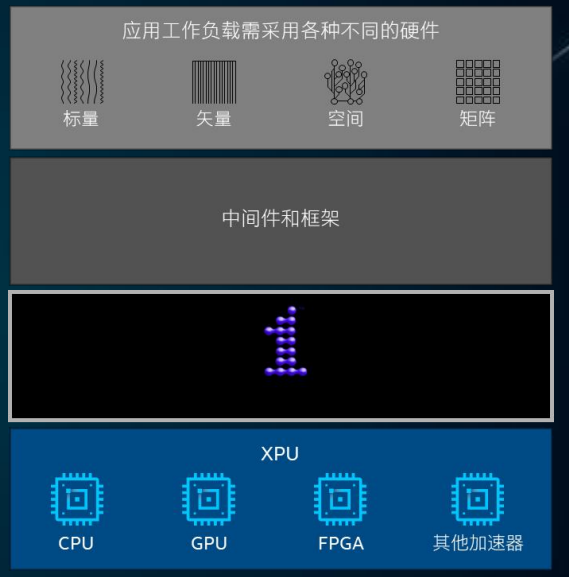
        e.wait();
}
```


文件(F) 编辑(E) 视图(V) Git(G) 项目(P) 生成(B) 调试(D) 测试(S) 分析(N) 工具(T) 扩展(X) 窗口(W) 帮助(H)
simple-add
登录
Live Share
进程: [10568] simple-add-usm.exe
线程: [4236] 主线程
堆栈帧: main
simple-add-usm.cpp
README.md
simple-add-usm
(全局范围)
main()
// Verify two results are equal.
for (size_t i =
if (parallel[
cout << "Fa
return -1;
int indices[](0...
constexpr size_
// Print out ic
for (int i = 0;
int j = indic
if (i == indi
cout << "[" <
<< seque
free(sequential
free(parallel,
} catch (std::exc
cout << "An exc
terminate();
cout << "Successf
return 0; 已用时间
C:\Users\ICL\source\repos\Simple_Add3\x64\Debug\simple-add-usm.exe
Running on device: Intel(R) Graphics [0x8a52]
Array size: 10000
[0]: 0 + 100000 = 100000
[1]: 1 + 100000 = 100001
[2]: 2 + 100000 = 100002
[9999]: 9999 + 100000 = 109999
Successfully completed on device.
诊断工具
诊断会话: 1 秒 (选择了 340 毫秒)
4秒
事件
进程内存 (MB)
快照
专用字节
166
0
CPU (所有处理器的百分比)
100
0
摘要
事件
内存使用率
CPU 使用率
所有事件(1 个，共 2 个)
截取快照
启用堆分析(会影响性能)
记录 CPU 配置文件
100 %
行: 117
字符: 64
空格
LF
自动窗口
搜索(Ctrl+E)
搜索深度: 3
名称
值
类型
调用堆栈
名称
语言
simple-add-usm.exe!main() 行 129
C++
[外部代码]
自动窗口
局部变量
监视 1
Compiler Inline Report
Compiler Optimization Report
调用堆栈
断点
异常设置
命令窗口
即时窗口
输出
就绪
添加到源代码管理
1254
18:38
2022/3/8


matrix_mul_dpcpp.cpp
main()
sum += a[row][i] * b[i][col];
c[index] = sum;
Microsoft Visual Studio 调试控制台
Device: Intel(R) Graphics [0x8a52]
Problem size: c(1500,6000) = a(1500,3000) * b(3000,6000)
Execution time of GPU = 1.875 seconds
MFLOPS: 0.000 mflops
Result of matrix multiplication using DPC++: Success - The results are correct!
Execution time of CPU = 5.756 seconds
MFLOPS: 0.000 mflops
C:\Users\ICL\source\repos\Matrix_Multiply\x64\Release\matrix_mul_dpcpp.exe (进程 3556)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .

oneAPI的进步


开发
自由
产品
调试


应用工作负载需采用各种不同的硬件
标量
矢量
空间
矩阵
中间件和框架
XPU
CPU
GPU
FPGA
其他加速器

Q&A

yinkui.zhang@nanocode.cn

```
foo[] = {4,8,12,16};

A: v = foo[tid.x];

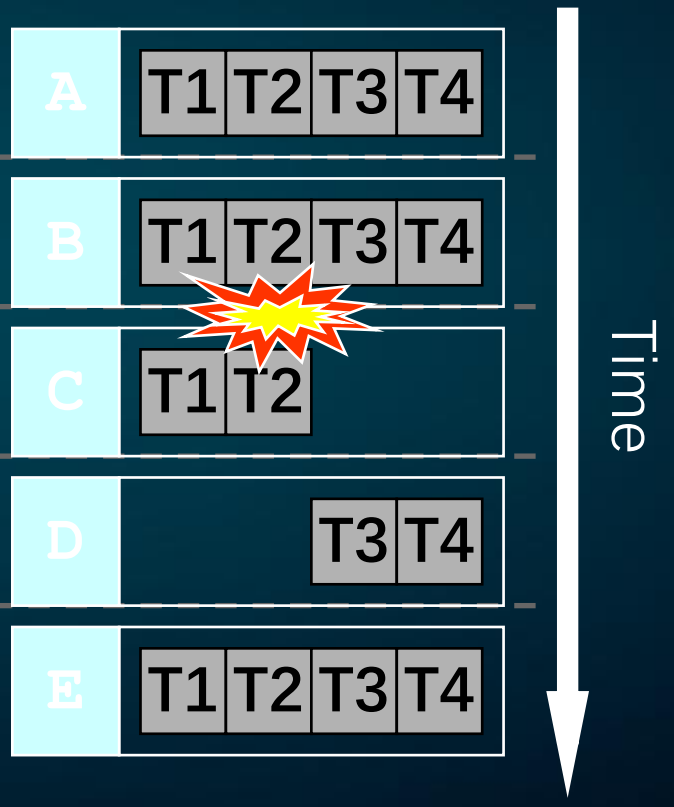
B: if (v < 10)

C:     v = 0;

   else

D:     v = 10;

E: w = bar[tid.x]+v;
```



CPP-Summit 2022



李颖欣

搜狗高级架构师

搜狗公司搜索事业部-系统架构部的高级架构师，拥有8年多的C++架构研发的经验，从事搜狗自研分布式Key Value存储系统，负责公司内部上千台规模、十多个集群集群、横跨多地的分布式kv离线/在线存储解决方案。近两年参与C++异步调度编程范式Workflow的开发，也是此上生态RPC框架SRPC的主要作者。自从项目开源以来，一方面在公司内外同步推进Workflow和SRPC的开发与落地，另一方面推动生态项目的调研和开发，此外也持续构建项目的开源社区，维护开发者生态。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

Workflow的性能优化实践

李颖欣
搜狗 高级架构师

议程

1 **背景介绍**

场景 / 系统层次 / 性能优化 /

2 **架构设计**

资源 / 管理 / 任务 / 任务流 / 复合任务 /

3 **编程范式**

范式 / 结构化并发 / 内存与生命周期 /

4 **案例分享**

案例分享 / 思路总结 / 开源收获 /

01

背景介绍

在什么样的场景下演化出Workflow？涉及什么方面的性能优化？

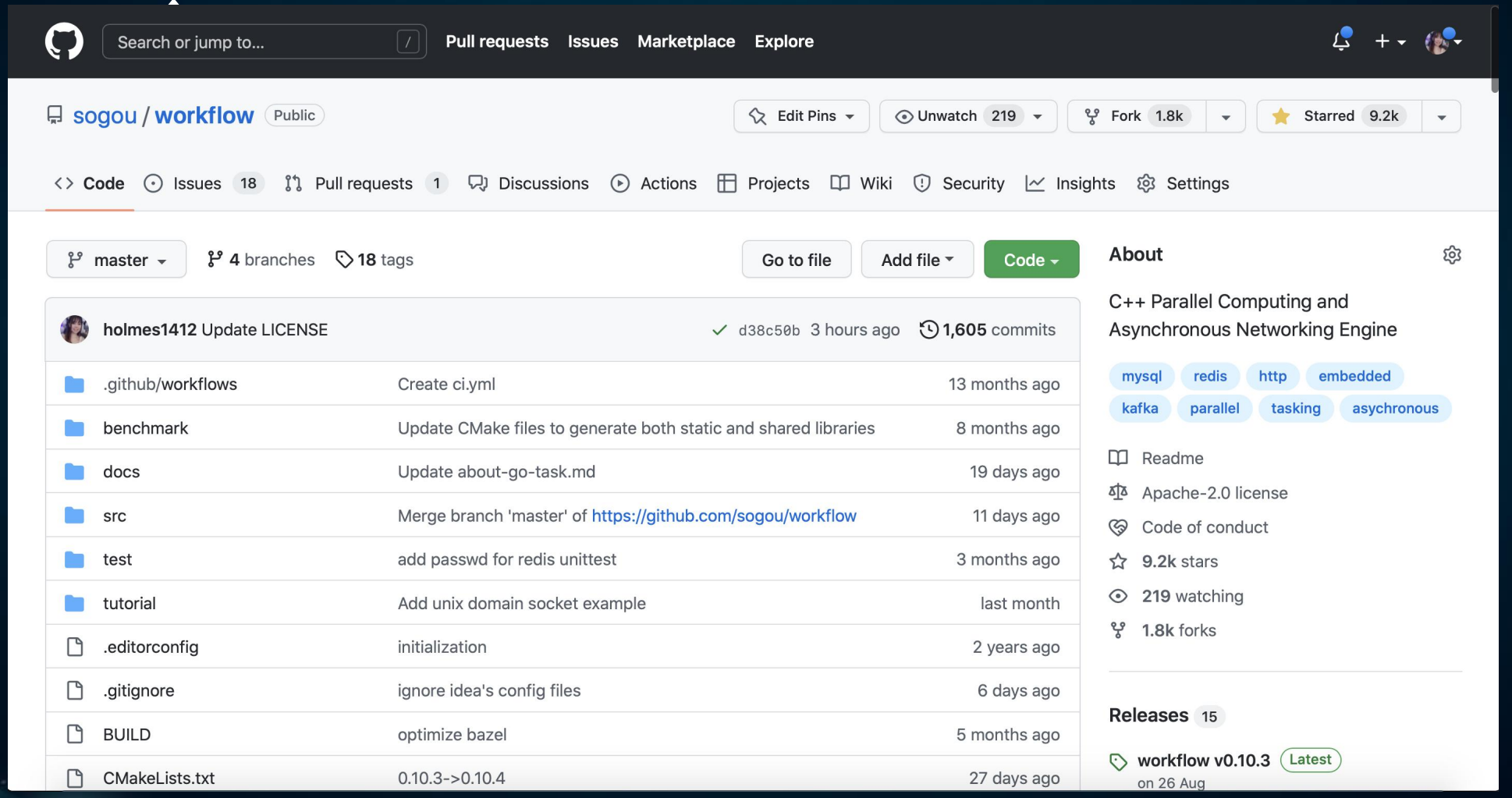
业务背景

搜索引擎

- 重计算服务：C++是唯一选择
- 模块多：调用复杂
- 烧钱：烧机器，也烧研发投入，技术投入高
- 兼容企业内部技术栈

架构演化有基因。这样的背景下，需要的是一个企业级 、 轻量级、高性能的解决方案。

https://github.com/sogou/workflow



个人背景



李颖欣

百度 – 搜狗 – 腾讯

搜索事业部 • 系统架构部 • 高级架构师

- 10年左右C++架构研发经验
- 从事自研分布式Key Value存储系统，负责公司内部上千台规模、二十多个集群、横跨多地的分布式KV离线/在线存储解决方案
- 近三年：参与C++异步调度编程范式Workflow的开发，并在此上建立了生态RPC项目srpc
- 推动开源社区，公司内外同步推进Workflow和srpc的开发与落地

项目背景

企业级 → 通用 ← 开源

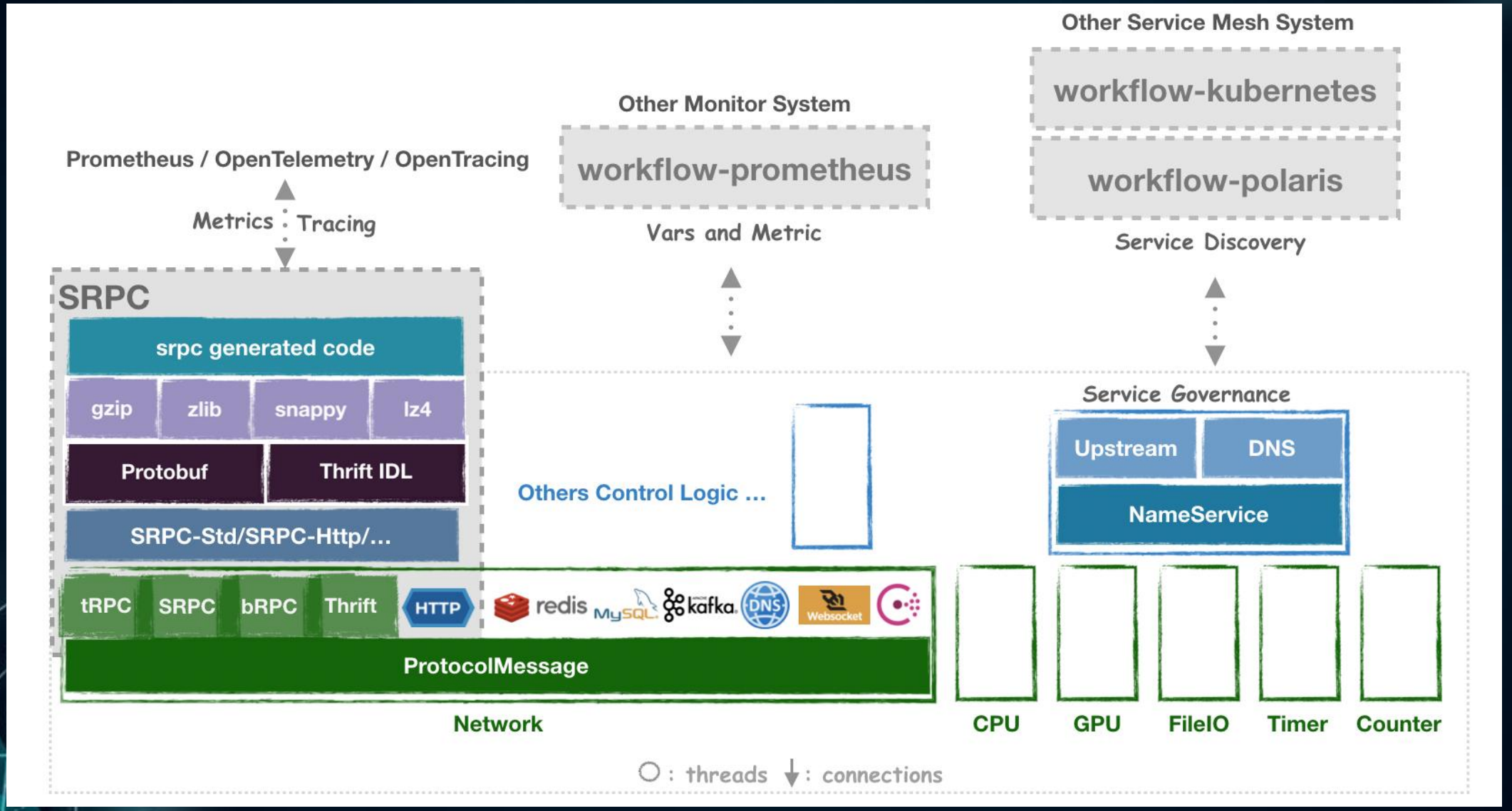
项目背景

Workflow是起源于搜狗的C++服务器引擎，支撑公司内的绝大部分后端C++在线服务，包括所有搜索服务，云输入法，在线广告等，每日处理数百亿请求：

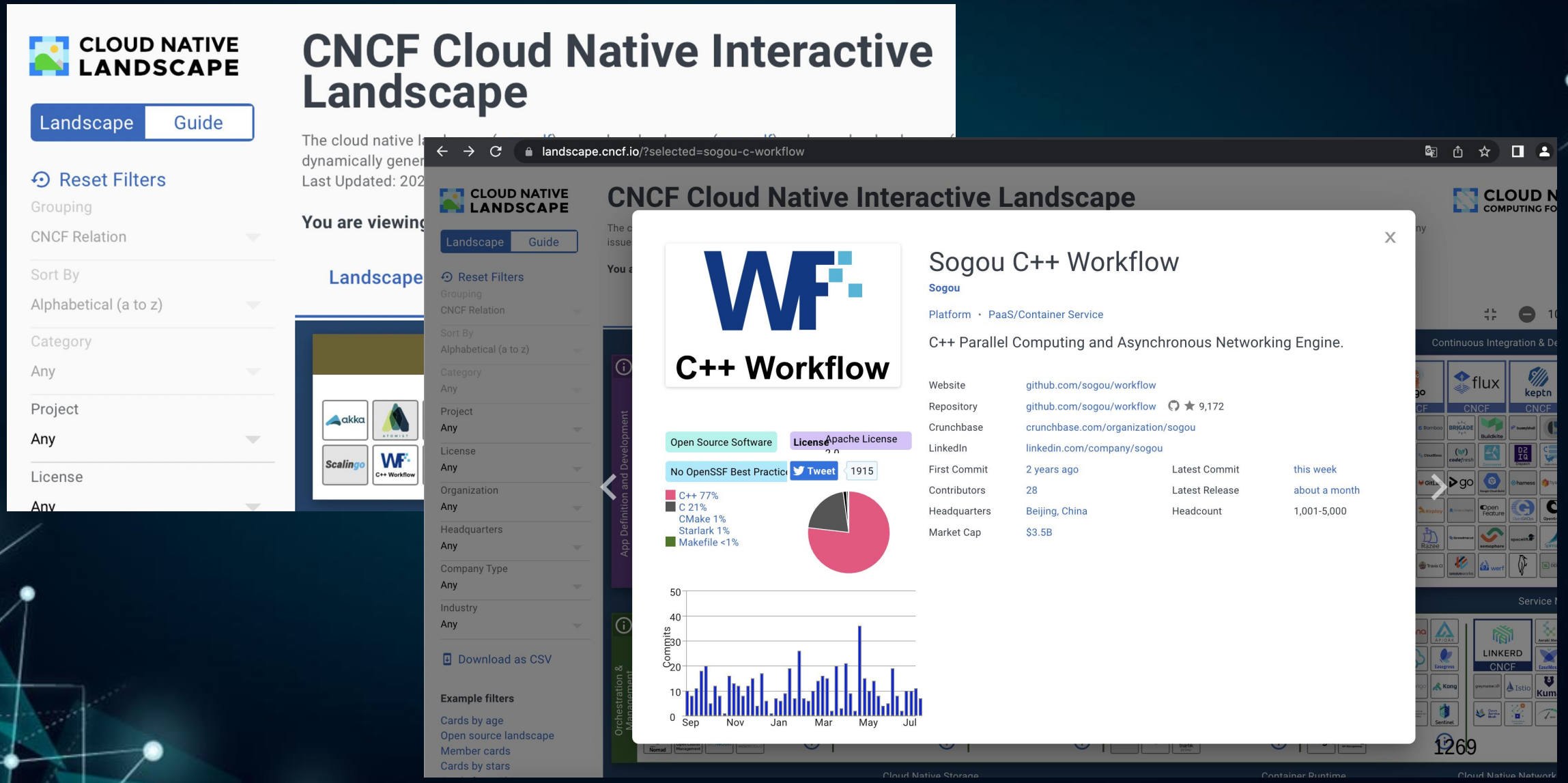
- 包括通信、计算、文件IO、定时器、计数器等异步资源
- 创新性引入的任务流概念，各种异步任务都可以被对等地组装起来实现复杂的业务逻辑
- 内部自带多种通用协议
- 跨平台、支持多种操作系统
- 自带命名服务，包括进程内服务治理与负载均衡

开源 2 年，设计轻盈优雅，是一个并行计算与异步通信融为一体的编程范式。

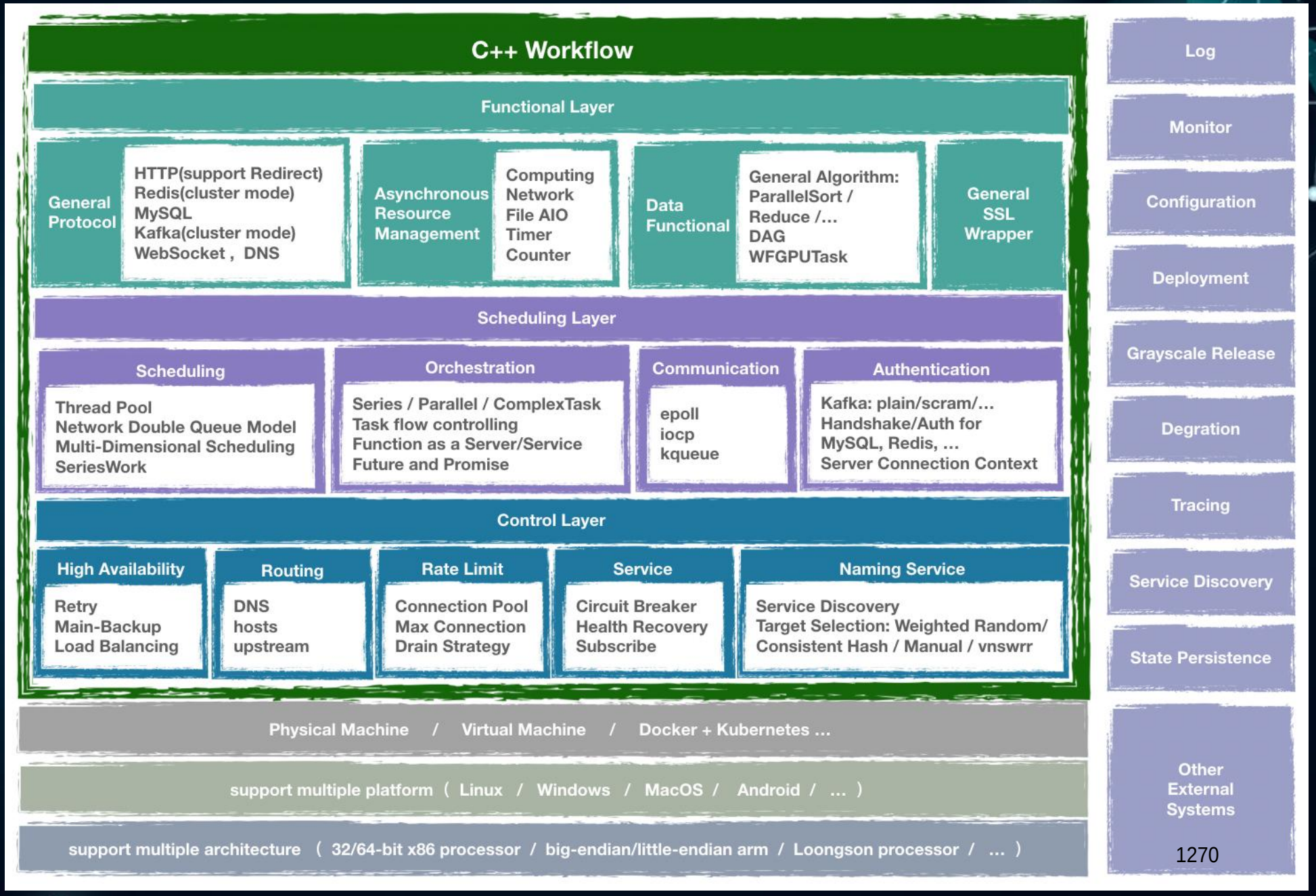
项目生态


Other Service Mesh System
workflow-kubernetes
workflow-polaris
Service Discovery
Other Monitor System
workflow-prometheus
Vars and Metric
Prometheus / OpenTelemetry / OpenTracing
Metrics
Tracing
SRPC
srpc generated code
gzip
zlib
snappy
lz4
Protobuf
Thrift IDL
SRPC-Std/SRPC-Http/...
Others Control Logic ...
Service Governance
Upstream
DNS
NameService
tRPC
SRPC
bRPC
Thrift
HTTP
redis
MySQL
kafka
DNS
Websocket
ProtocolMessage
Network
CPU
GPU
FileIO
Timer
Counter
: threads
: connections

CNCF Landscape - Platform - PaaS



Workflow PaaS层次图



项目演进



起源于搜狗内部自研的分布式KV存储系统的异步通信模块，并且把各种资源同等对待地封装，以统一的任务方式提供给用户。

由于公司内大部分业务都属于搜索相关的重计算，许多模块每日请求量上十亿级别，对性能要求非常高，Workflow解决了实际业务痛点，耗费半年时间被公司内开发者自行选择成为公司内C++编程标准，目前还是一个相对年轻、快速演进中的项目。

性能优化

Workflow中的性能优化包括了许多方面：

1. 连接的管理
2. 计算调度的算法
3. 队列的实现
4. ...

这次会涵盖的：从架构设计的理念出发达到的性能优化。

主要特点

1. **计算与通信融为一体**
   特别适合需要网络通信的重计算模块。不单网络、磁盘IO或CPU计算，而是让所有异步资源都尽可能被全部调起;

2. **轻量级**
   无依赖、编译快、体积小、插件化;

3. **架构层次适合组装与复用**
   提出算法、协议、业务逻辑分离的编程思想，统一的开发模式让一切都以任务形式组装;

4. **用户体验友好**
   接口简洁、提供常用协议、使用方式符合人类思维;

```cpp
#include <stdio.h>
#include "workflow/WFHttpServer.h"

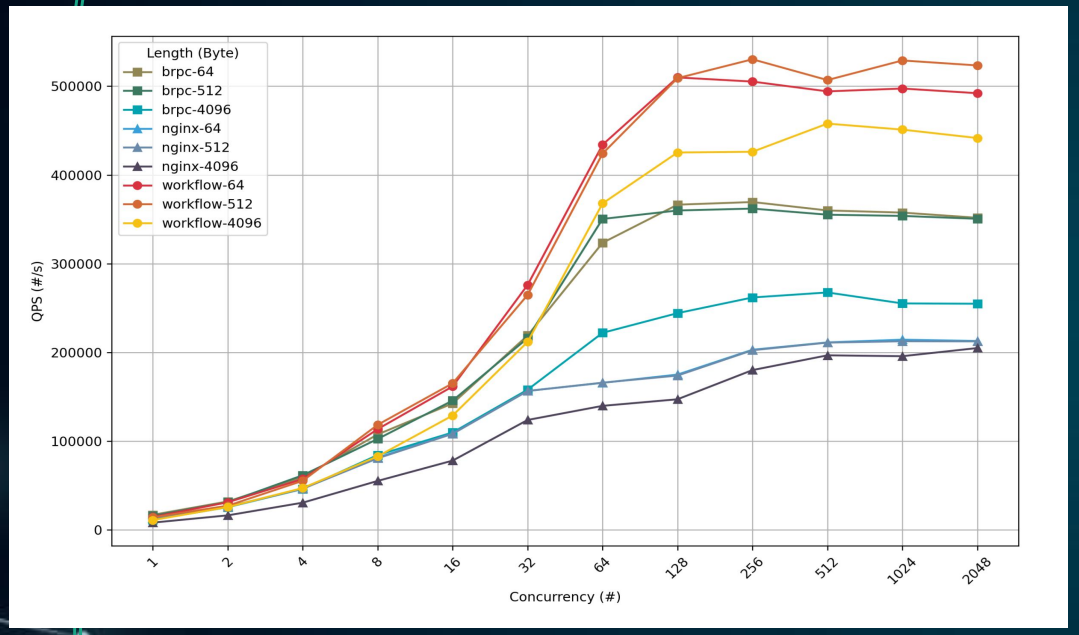
int main()
{
    WFHttpServer server([](WFHttpTask *task) {
        task->get_resp()->append_output_body("<html>Hello World!</html>");
    });

    if (server.start(8888) == 0) { // start server on port 8888
        getchar(); // press "Enter" to end.
        server.stop();
    }

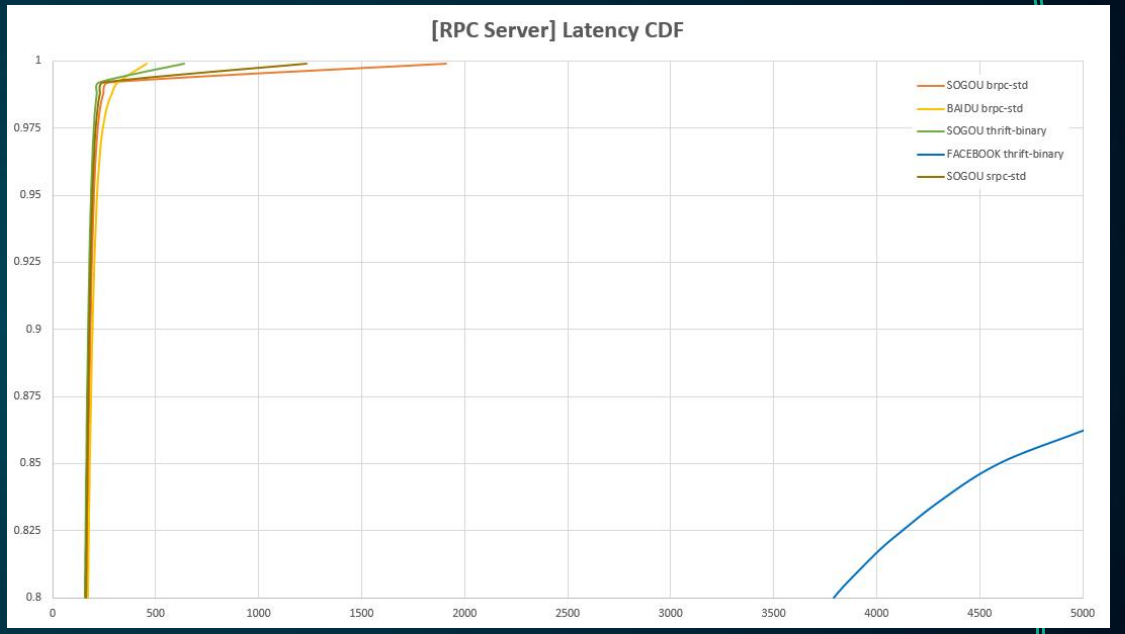
    return 0;
}
```

基础的性能数据

吞吐 与 长尾



Workflow：固定请求大小下，QPS与并发的关系



生态项目srpc：延迟的累计分布函数统计图

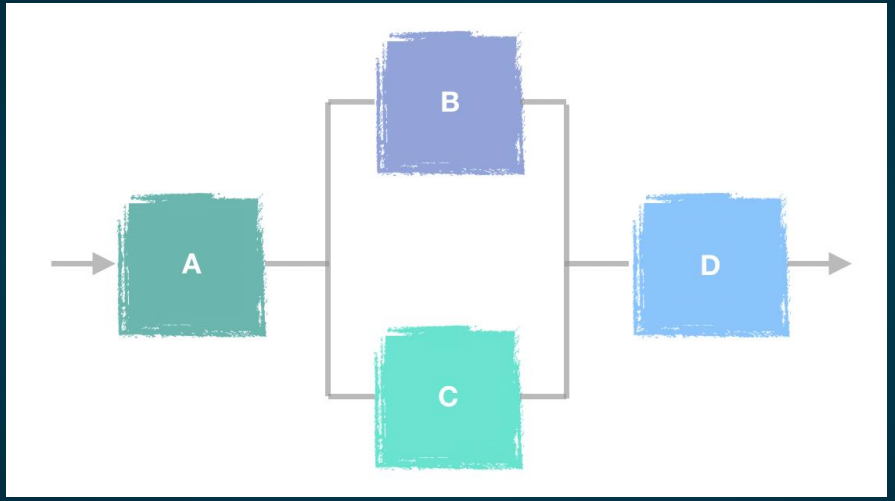
理念：通用

02

架构设计

为了做到无损耗的任务流，从底向上都像搭积木一样去编写每一个模块。

先来一个例子



资源

并行计算与异步通信融为一体

既不是RPC框架+线程调度，也不是任务调度框架+网络通信器。

项目主页的介绍：

C++ Parallel Computing and Asynchronous Networking Engine

可以被调度的，应该是任何一种资源，而资源之间是平等的。

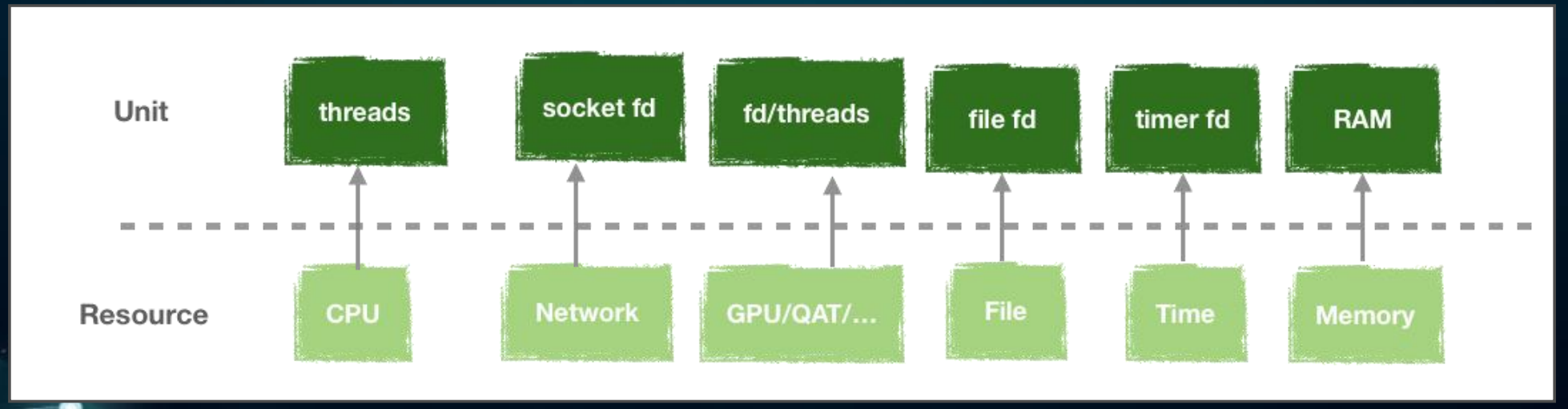
这样才能做到：

底层不同的资源，在上层的调度接口都可以打通，用统一的模式去调度。

资源

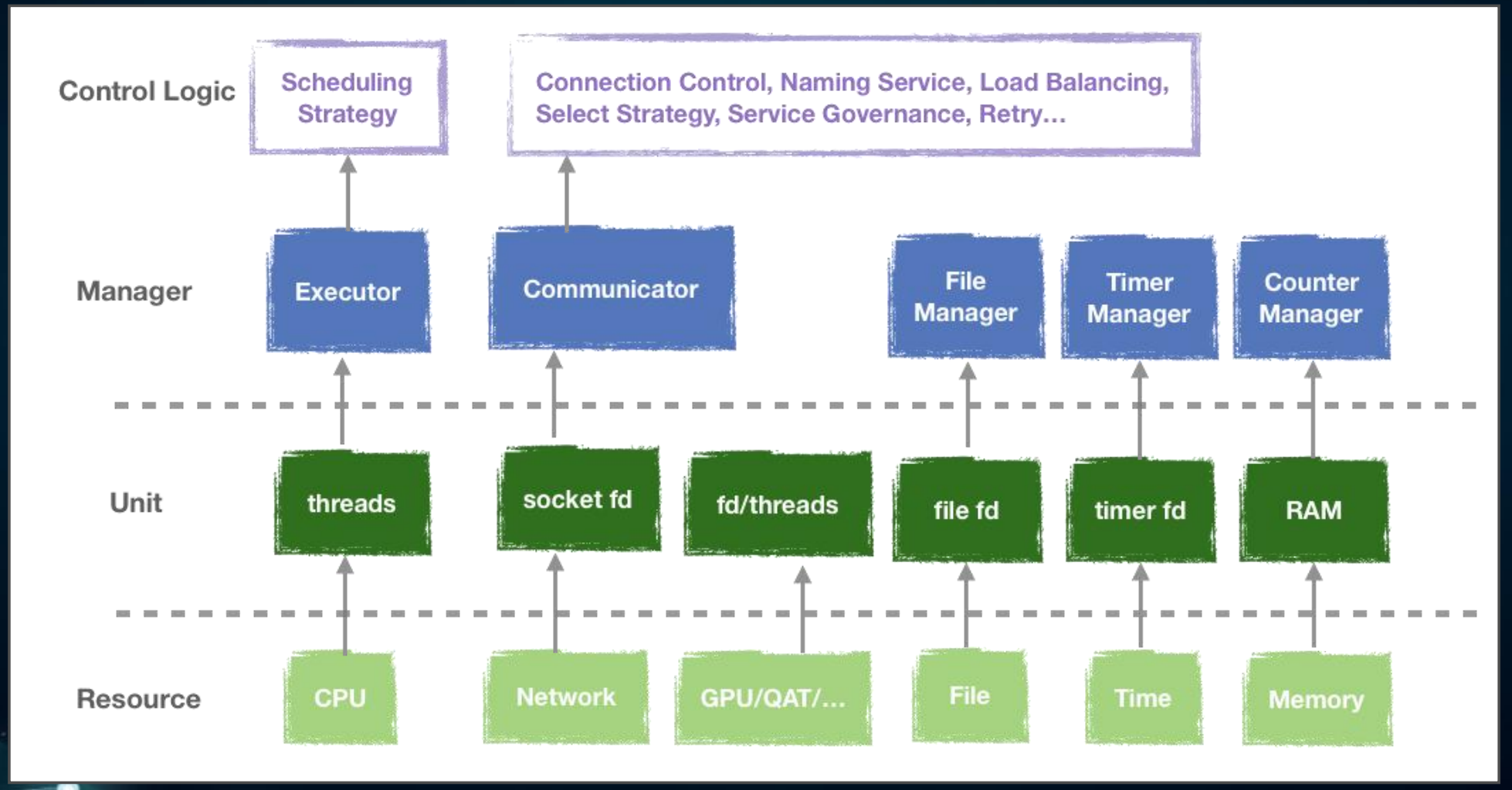
：框架如何接入？

不同的系统资源有不同的调度单位以及不同的调度方式。



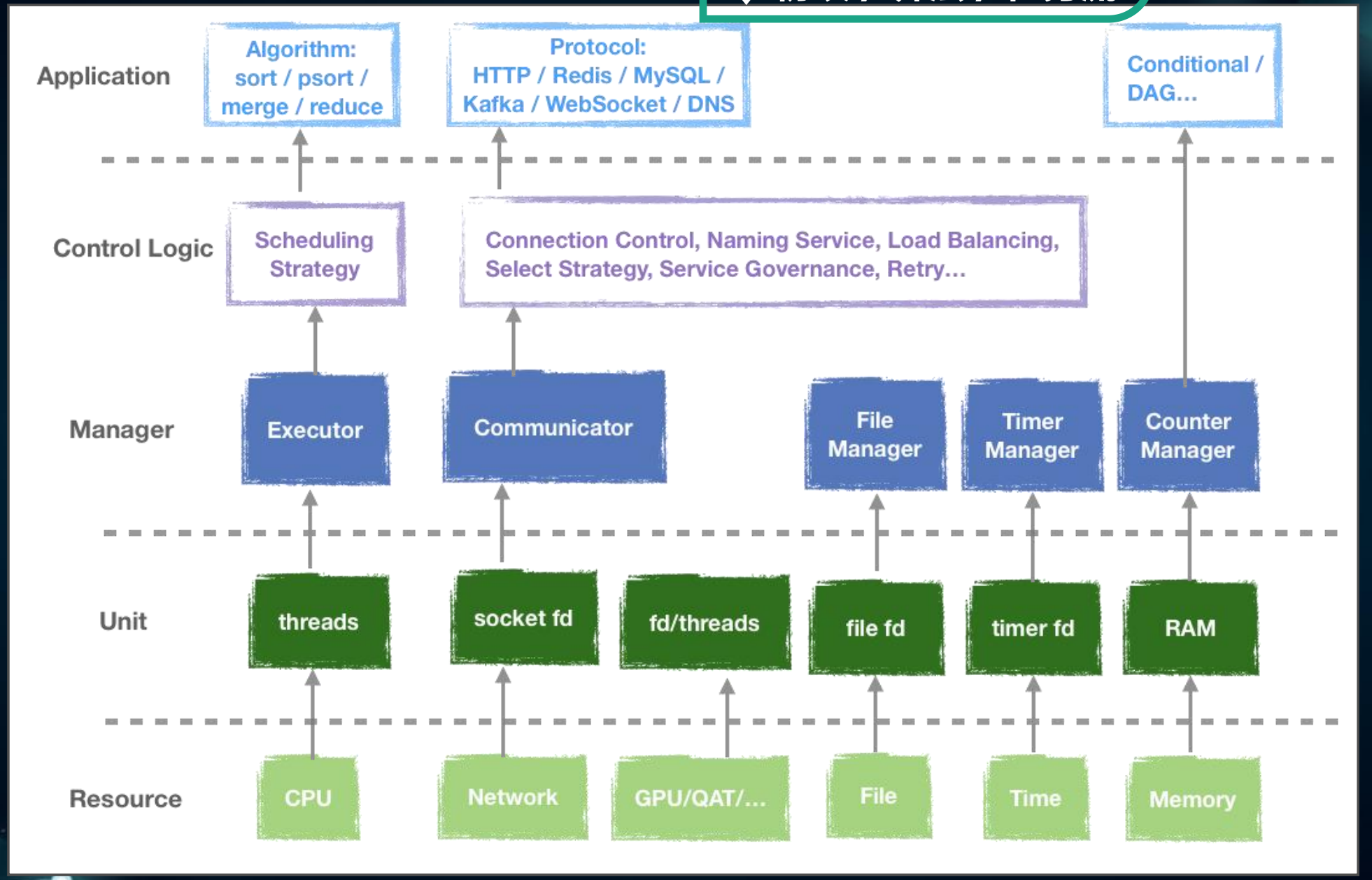
管理

- 机制：每种资源都需要它的调度管理器
- 策略：面对场景与需求的控制逻辑



任务

协议 / 算法 / 任务流


Application
Algorithm: sort / psort / merge / reduce
Protocol: HTTP / Redis / MySQL / Kafka / WebSocket / DNS
Conditional / DAG...
Control Logic
Scheduling Strategy
Connection Control, Naming Service, Load Balancing, Select Strategy, Service Governance, Retry...
Manager
Executor
Communicator
File Manager
Timer Manager
Counter Manager
Unit
threads
socket fd
fd/threads
file fd
timer fd
RAM
Resource
CPU
Network
GPU/QAT/...
File
Time
Memory

任务

用std::function<void (TASK *)>做回调的载体

工厂模式，行为派生，不同资源的任务其继承路径相似

统一的接口和用法：server与client对称，计算与网络对称，定位资源的方式对称

用法的统一

```
WFGoTask *task;
task = WFTaskFactory::create_go_task("queue_name",
                                     add, a, b);
```

```
WFHttpTask *task;
task = WFTaskFactory::create_http_task("http://cpp-summit.org",
                                       REDIRECT_MAX, RETRY_MAX,
                                       callback);
```

```
WFFileIOTask *task;
task = WFTaskFactory::create_pread_task("/home/.vimrc",
                                        buf, size, 0,
                                        callback);
```

接口的统一

计算

```
276 public:
277     template<class FUNC, class... ARGS>
278     static WFGoTask *create_go_task(const std::string& queue_name,
279                                     FUNC&& func, ARGS&&... args);
280
```

网络

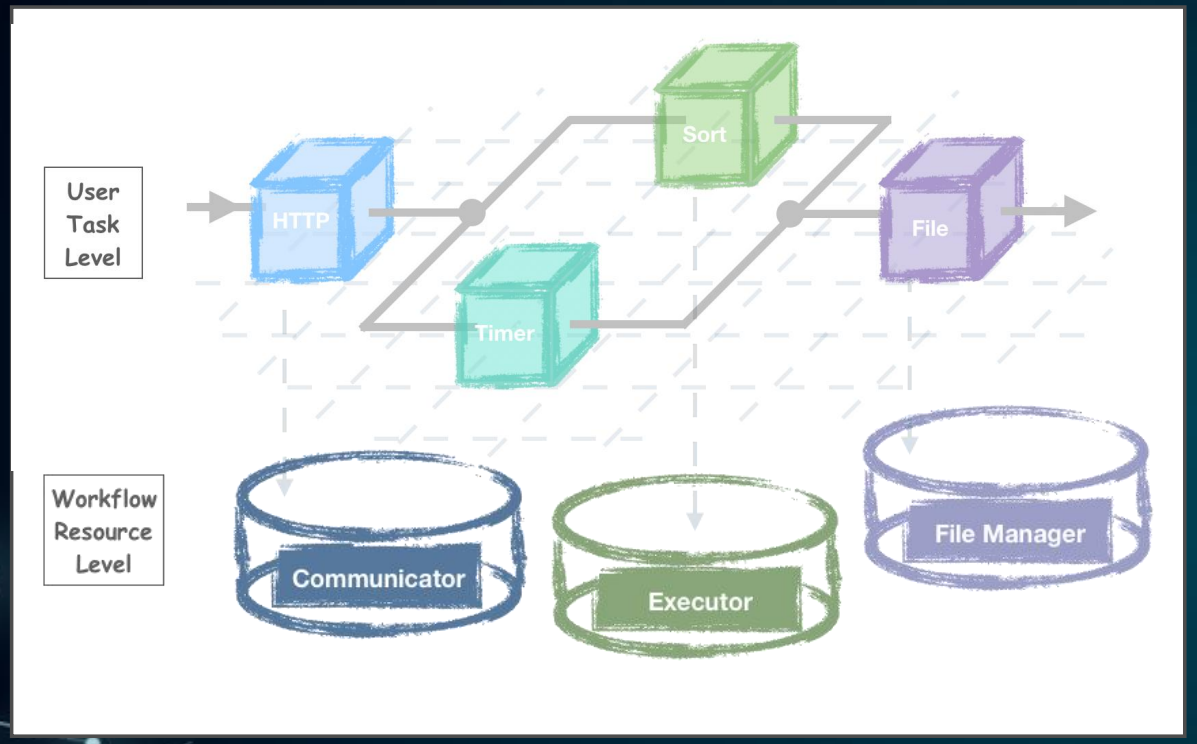
```
42 using WFHttpTask = WFNetworkTask<protocol::HttpRequest,
43                                  protocol::HttpResponse>;
44 using http_callback_t = std::function<void (WFHttpTask *)>;
45
46 using WFRedisTask = WFNetworkTask<protocol::RedisRequest,
47                                   protocol::RedisResponse>;
48 using redis_callback_t = std::function<void (WFRedisTask *)>;
```

文件

```
81 using WFFileIOTask = WFFileTask<struct FileIOArgs>;
82 using fio_callback_t = std::function<void (WFFileIOTask *)>;
```

任务流

新的结构化并发单位，解决异步调度的问题




```
auto *a = WFTaskFactory::create_http_task("https://...", ...);
auto *b = WFTaskFactory::create_go_task("go", sort, ...);
auto *c = WFTaskFactory::create_timer_task(1000, ...);
auto *d = WFTaskFactory::create_pwrite_task("/dev/...", ...);

(*a > ( (*b) * (*c) ) > d).start();
```

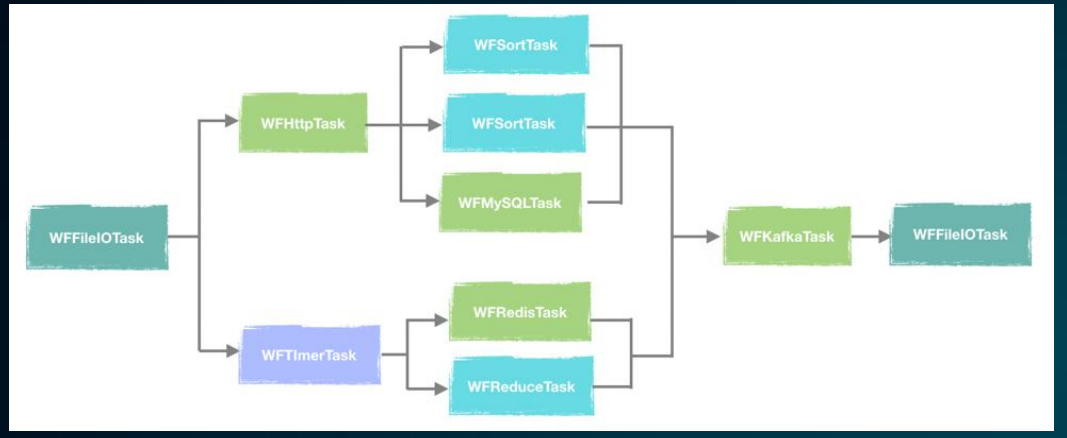
```
WFGraphNode& a = graph->create_graph_node(http_task);
WFGraphNode& b = graph->create_graph_node(go_task);
WFGraphNode& c = graph->create_graph_node(timer_task);
WFGraphNode& d = graph->create_graph_node(write_task);

a-->b;
a-->c;
b-->d;
c-->d;
```

```
series_of(task1)->push_back(task2);
```

- 业务逻辑的调度条件受限于**上层任务流**的依赖关系；
- 具体资源的调度受限于**资源本身**与管理器和控制逻辑；
- 所有**类型**的任务都可以放到同一个任务流里；

任务流

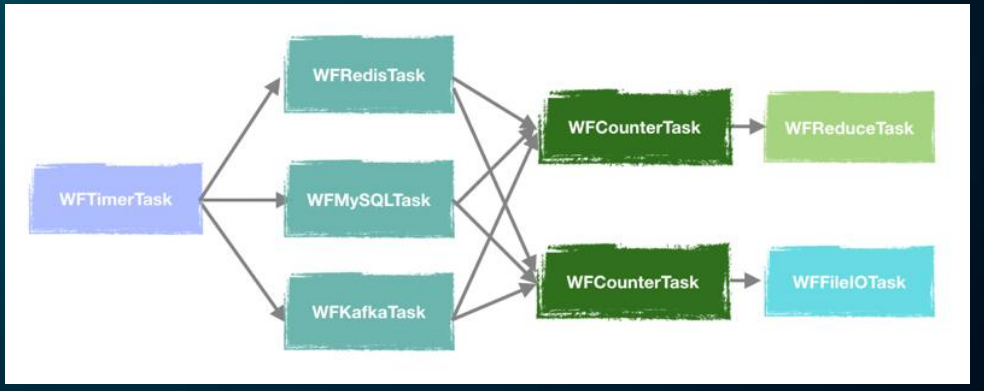


Series：串行任务流；
Parallel：并行任务流；

互相拼装，组成复杂的业务逻辑；

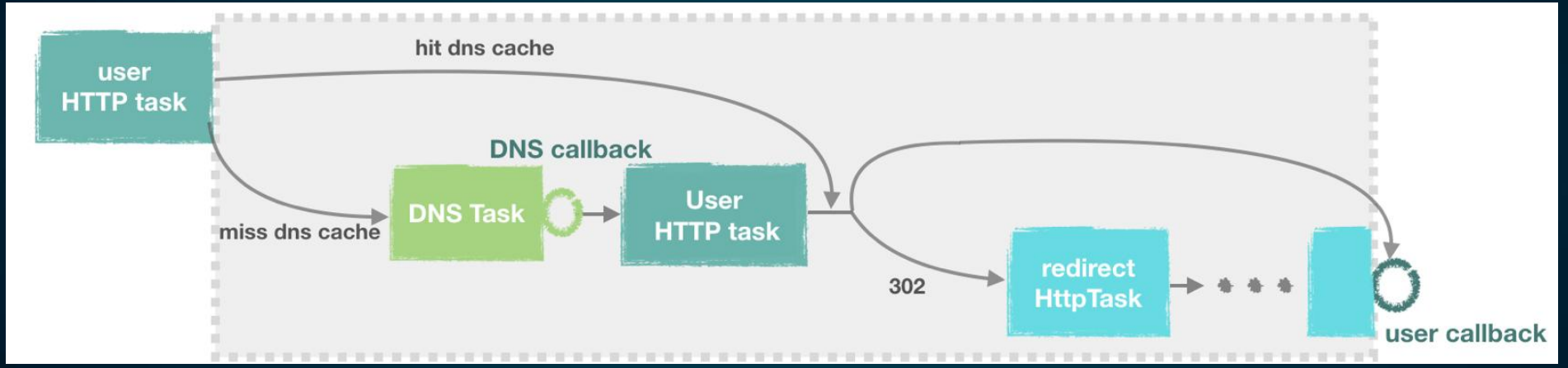
支持更加复杂的DAG结构；

支持动态 / 静态创建任务流；



复合任务

任务可以组装的意义在于：我们得以对复杂的任务进行隐藏，解决回调函数模式的碎片化问题；



HTTP Complex Task Internal Flow Graph

- 一次http请求，可能包含许多次异步过程（DNS，重定向），但对用户来讲，就是一次通信任务；
- 文件排序，包括复杂的文件IO与CPU计算的交互过程，对外可以封装出一个完整的文件排序算法；
- 如果把业务逻辑想象成用设计好的电子元件搭建电路，那么每个电子元件内部可能又是一个复杂电路。

03

编程范式

程序 = 协议 + 算法 + 任务流

编程范式

程序 = 协议 + 算法 + 任务流

一个典型的后端程序由这三个部分组成，并且完全独立开发。

Workflow的编程范式遵循进程代数中的顺序、归约、隐藏、递归等演算语义。

其中包括三个约定：

- 串行是由任务组成的；
- 并行是由串行组成的；
- 并行是一种任务；

协议

大多数情况下，用户使用的是内置的通用网络协议，用户可以方便的自定义网络协议。

算法

与协议对称，表示任何一次边界清晰的复杂计算。如果说协议的调用是rpc，算法的调用就是一次apc (Async Procedure Call)。

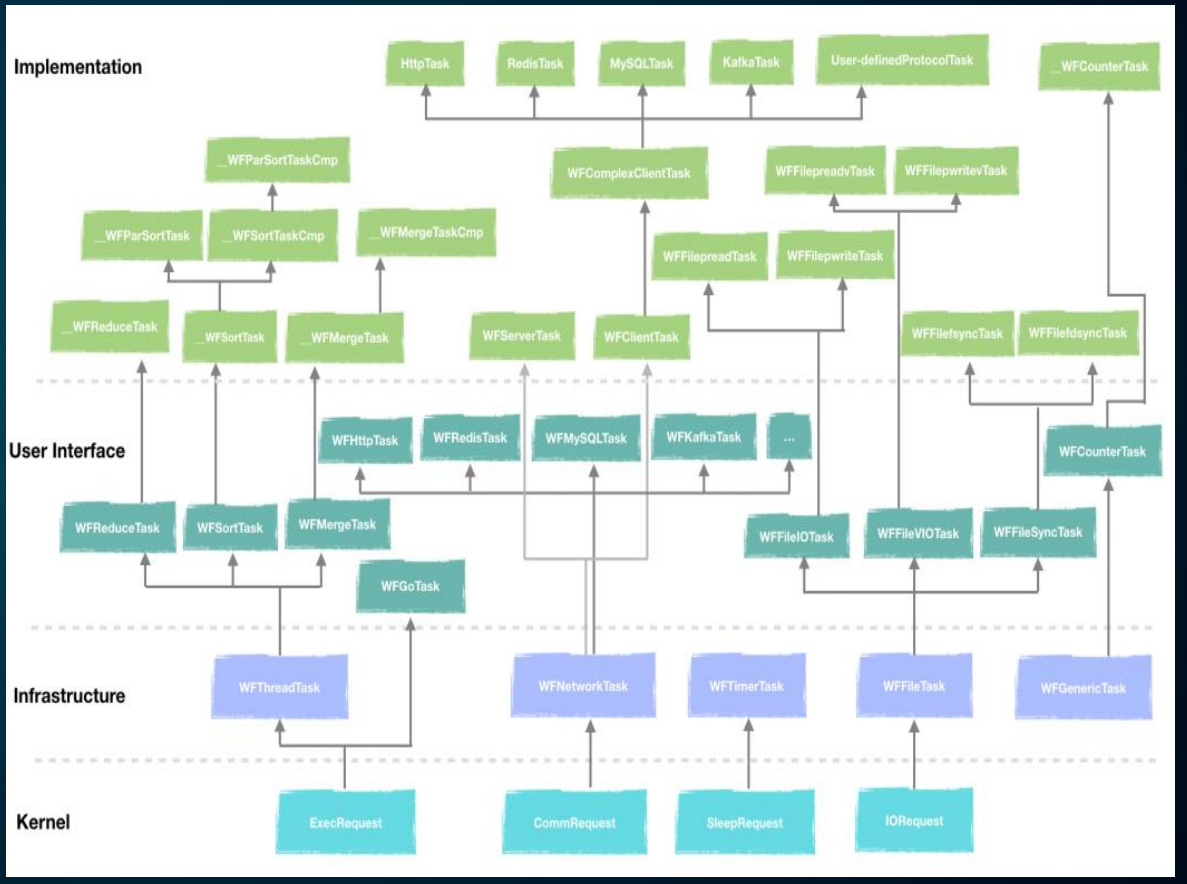
任务流

实际的业务逻辑，可以静态/动态构建。所有任务都是异步执行的。

编程范式

结构化并发

- 基础任务，任务工厂，复合任务
- 包含多种基础任务：通讯，文件IO，CPU，GPU，定时器，计数器等
- 一切任务都由任务工厂产生，并且在callback之后自动回收
- server任务是一种特殊的通讯任务，由框架调用任务工厂产生，通过process函数交给用户
- 用户通过工厂产生的极有可能是复合任务，但对用户透明



Workflow内部派生结构

结构化并发

```
WFHttpServer server([](WFHttpTask *proxy_task) {
    WFHttpTask *http_task = WFTaskFactory::create_http_task(...);
    // fill req and body
    series_of(proxy_task)->push_back(http_task);
});
```

一个不阻塞任何当前server函数线程、异步发起后端请求、自动回复的proxy

回调与内存回收机制

- **一切调用都是异步执行，几乎不存在占着线程等待的操作：**
  显式的回调机制。用户清楚自己在写异步程序。

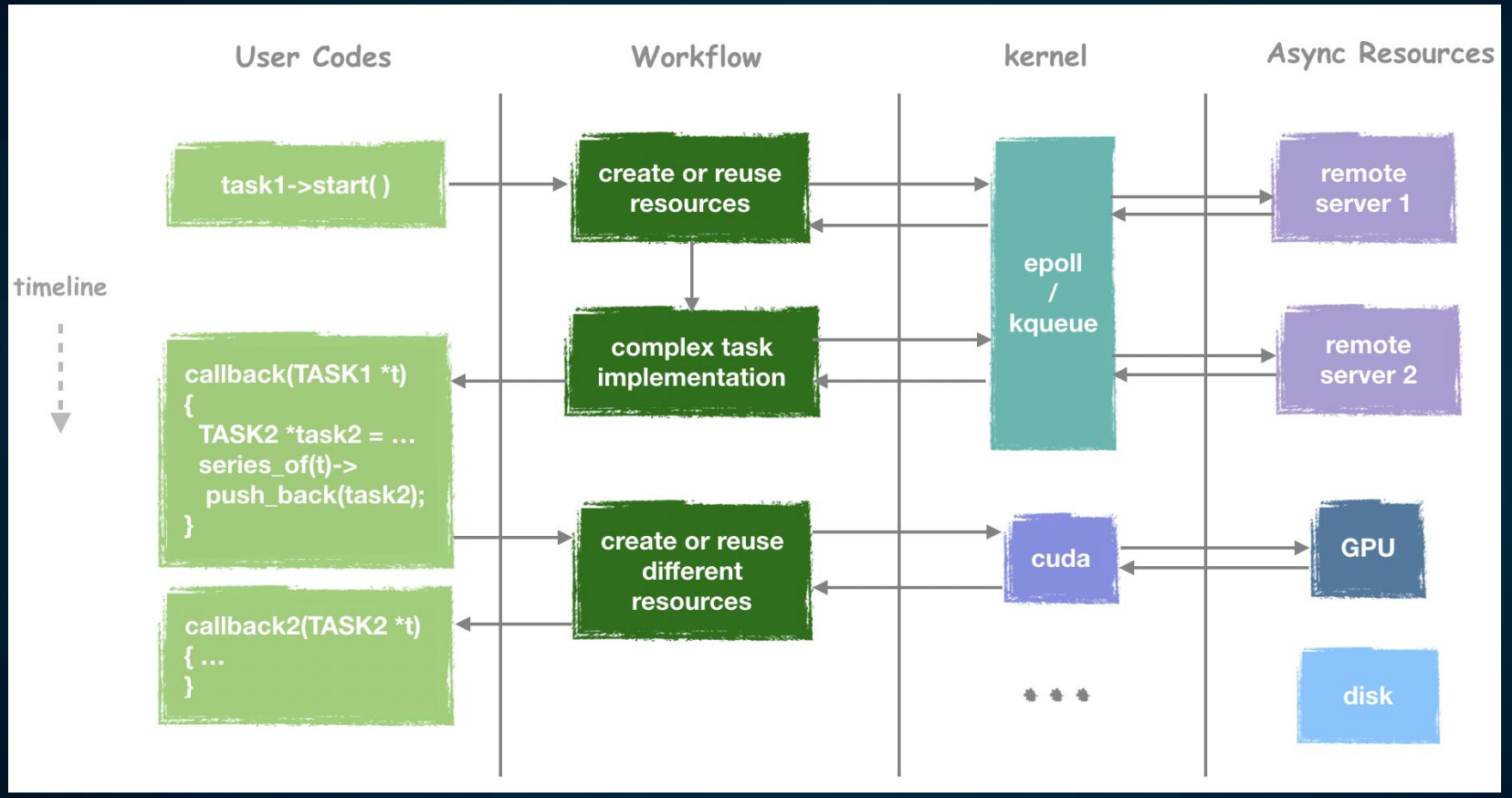
- **通过一套对象生命周期机制，大幅简化异步程序的内存管理：**
  任何框架创建的任务，生命周期都是从创建到callback函数运行结束为止。没有泄漏风险。
  如果创建了任务之后不想运行，则需要通过dismiss()接口删除。

- **任务中的数据对称：**
  计算 = 输入 + 输出 + 算法
  网络 = 请求 + 回复 + 协议
  例如网络请求的resp，也会随着任务被回收。此时用户可通过std::move( )把需要的数据移走。

- **任务中的用法对称：**
  尽量避免用户级别派生，而是以std::function封装用户行为。
  包括任何任务的callback、任何server的process，符合FaaS(Function as a Service)思想。
  一个算法的实现，简单来讲也是一个std::function。

编程范式

回调与内存回收机制



- 内部模块每一层都要需要有严格的语义
- 如果深入使用，又会发现一切皆可派生

04

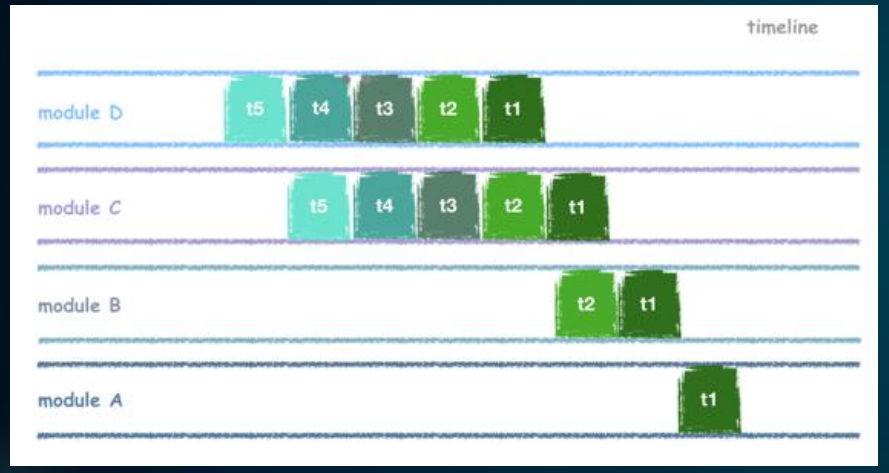
性能优化案例

案例分享

Cache流水线模式

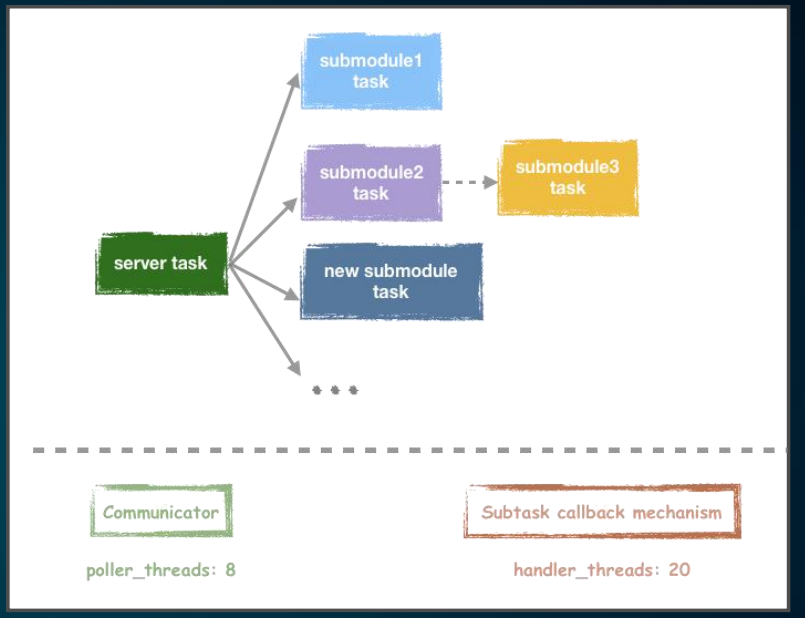
网页cache模块作为检索系统的核心，最重要的两个功能：

- 请求多路下游数据源，并对结果进行整合
- 使用复杂的计算进行精排



需要解决：

1. 不让任何一个异步环节空占资源
2. 资源配比灵活，应对耗时增长和新增数据源
3. 错误情况下的动态流程调整

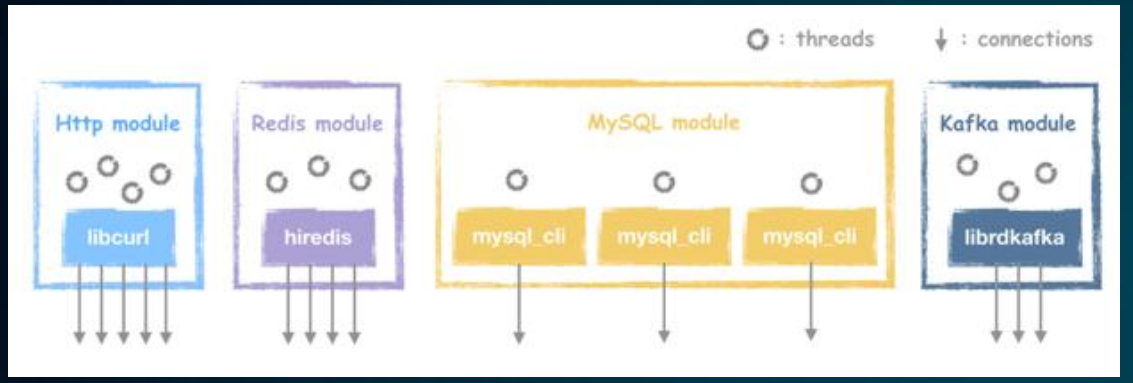


易用性：统一的接口
灵活性：无需提前规划，只要有系统资源即可立刻调起
扩展性：对等添加新加数据源
性能收益：延迟降低15ms，调度无损耗，固定延迟下QPS可打到理论极限值
额外收益：内置服务治理，支持下游多种选取策略

案例分享

抓取中台

多网络协议混合，常常需要自行管理连接复用和线程复用，如果客户端本身的复用做的不好，往往会成为系统的瓶颈。



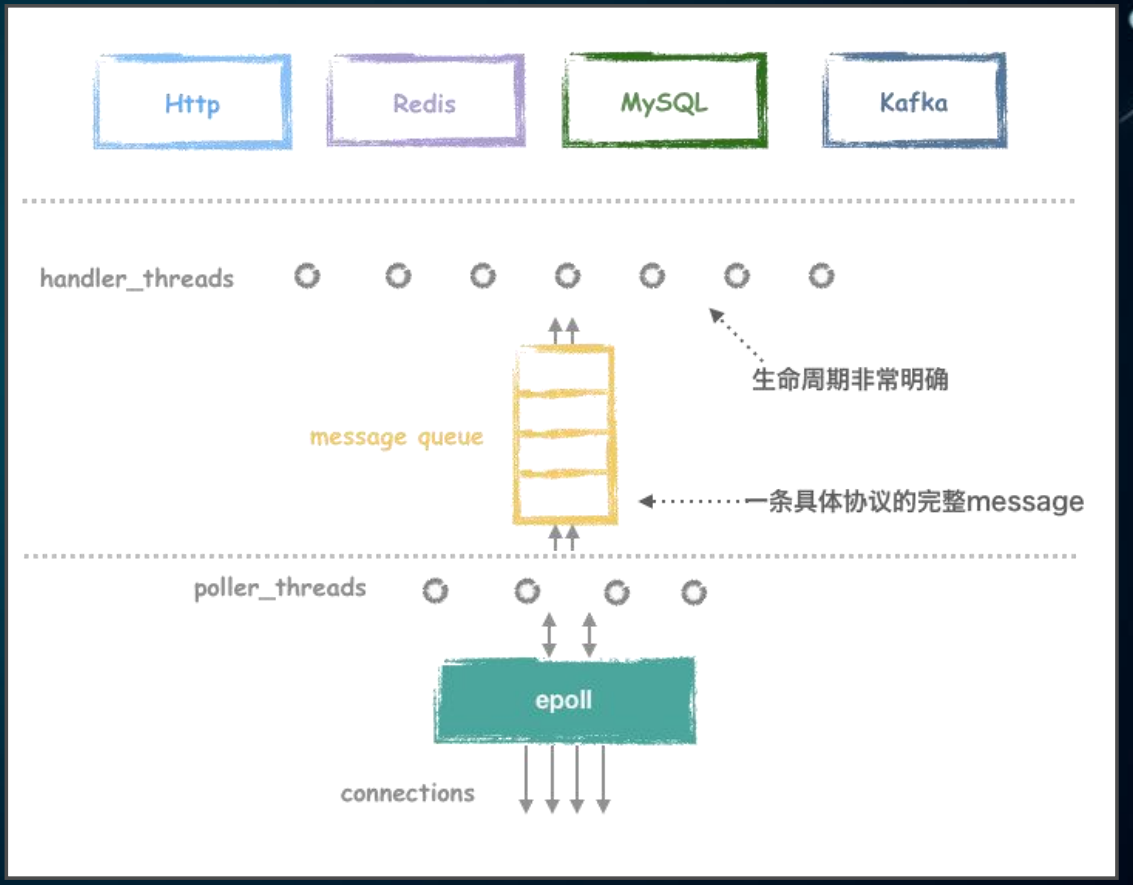
所有协议使用统一的编程范式才能做到连接与线程的资源共享，统一的使用接口也降低开发成本。Workflow内置的Http、Redis、MySQL、Kafka等协议全部都是自解析，特别适合实现异步分布式协议。

性能收益：

抓取速度提升一倍；

连接数降低一半，并发提升4倍；

并发控制更容易；

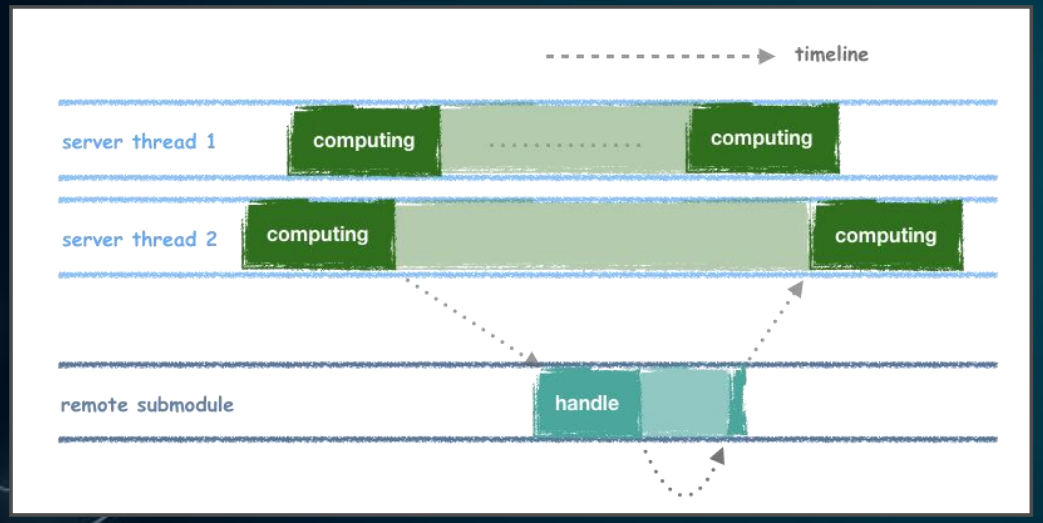


Workflow网络模型

案例分享

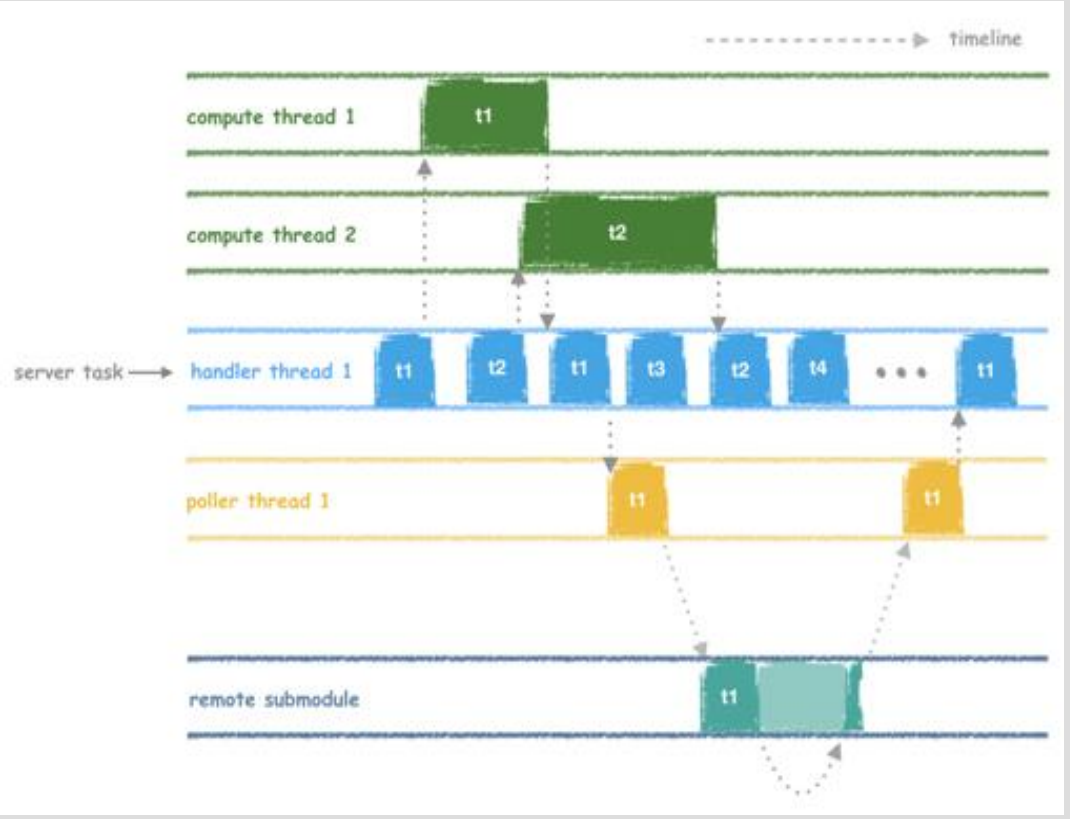
云输入法

每个请求到来之后都需要进行许多繁重的计算，包括引入其他硬件设备，部分计算因为本地内存 / 算力不足而被挪到远端，远端子请求回来后还要进行一部分计算，最后才回复用户。



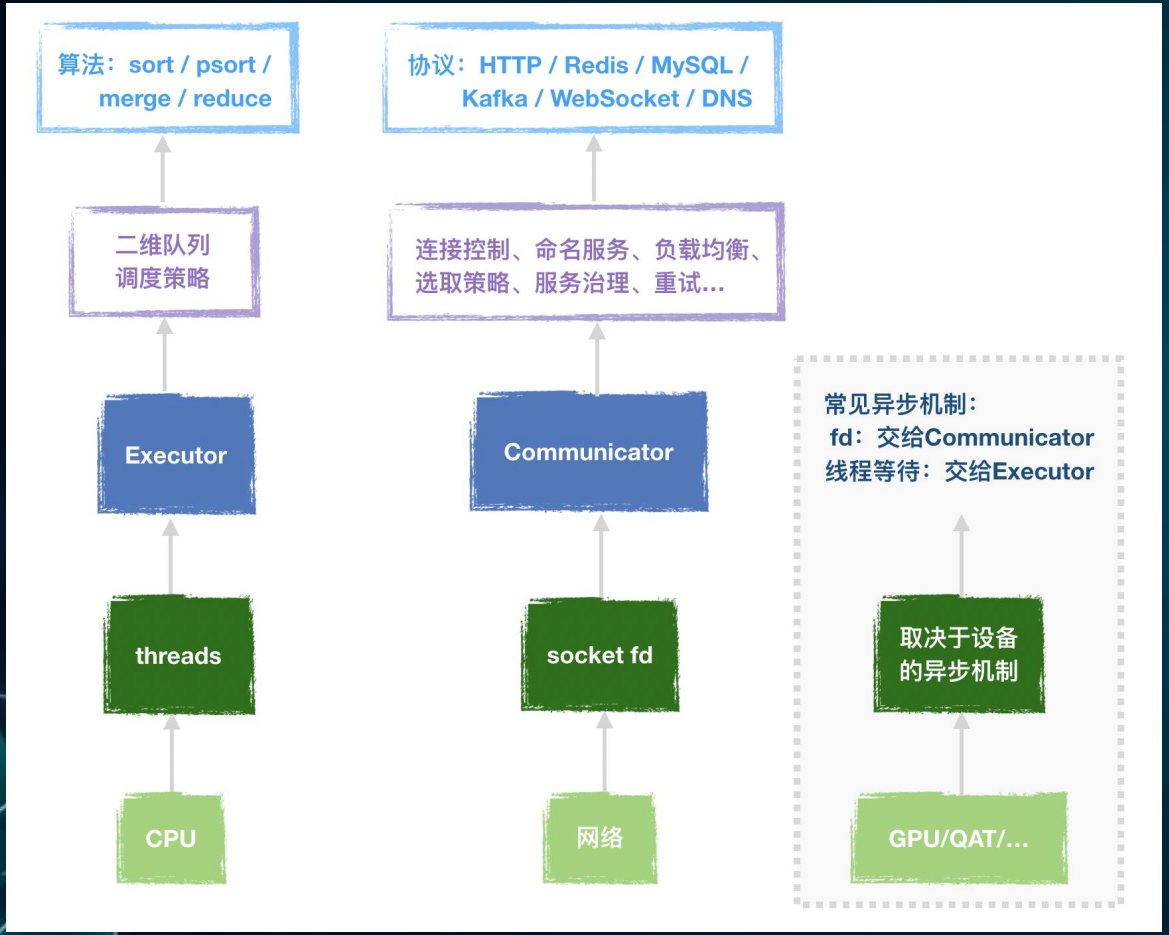
这样会导致的问题：

计算堵死通信线程，新来的网络请求无人响应；
通信影响计算，导致没法跑满CPU；
算法同学和工程同学的工作互相影响；



案例分享

云输入法 – QAT加速卡



具体收益：

- **QPS提升2.5倍，CPU打满；**
- **算法与服务调度互不影响；**
- **易于接入新硬件：**

  比nginx模块支持QAT需要增加模块、改nginx主框架支持用户增删fd的做法要简单，而且实测平响从37ms降低到18ms，P99从500ms降低到200ms；

来自开源社区的场景分享

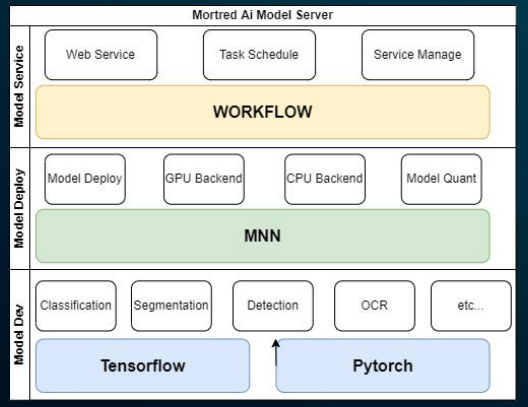
1. 小红书： mysql + kafka
2. 量化交易公司：http server与抓取
3. 嵌入式领域：树莓派与龙芯等
4. 安卓交叉编译：应用于电视机上
5. 部署深度学习服务

开源生态项目

1、 srpc：基于Workflow的高性能RPC，支持多IDL，多协议
https://github.com/sogou/srpc

2、 wfrest：外部生态项目，提供Web框架RESTful接口
https://github.com/wfrest/wfrest

3、 Mortred AI Model Server：充分发挥CPU和GPU性能的DL服务器
https://github.com/MaybeShewill-CV/mortred_model_server



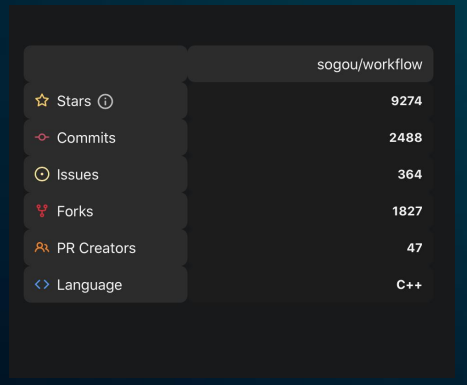
workflow + mnn + tensorflow

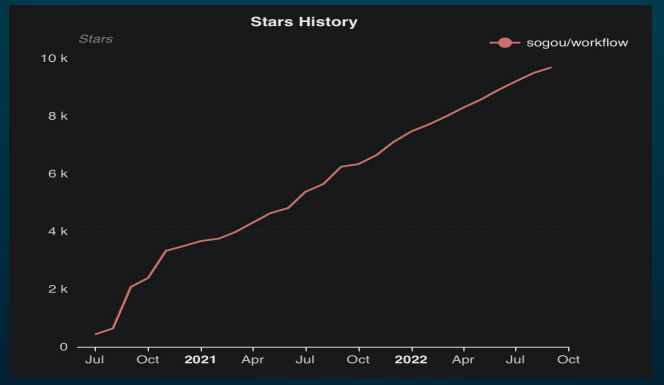
开源收获

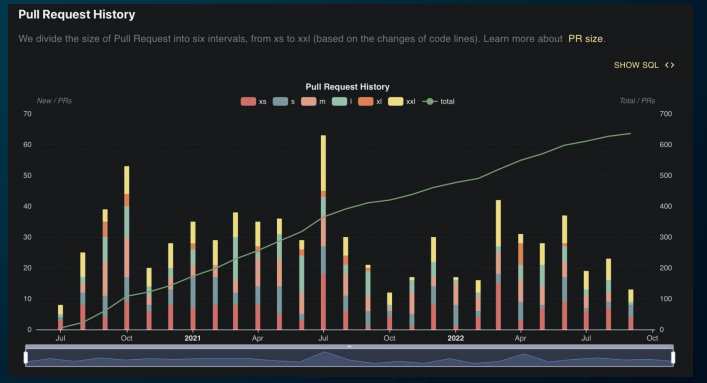
1. 已成为Debian Linux / Ubuntu 22.04自动安装包

```
[WORKFLOW IN UBUNTU 22.04] apt search libworkflow-dev
Sorting... Done
Full Text Search... Done
libworkflow-dev/jammy 0.9.10-1 amd64
  Parallel computing and asynchronous web server engine
```

2. GitHub增长迅速，其他平台上的技术积累也较受欢迎







3. 陆续有更多公司内外用户，windows开发者很多，进入高校课程内容

CPP-Summit 2022

Fedor Pikus



Mentor 首席工程科学家

现任职Mentor公司负责著名大型EDA产品Calibre的长期技术方向，架构设计，软件技术研究，以及工程师团队的指导和培训。Fedor 1998年加入Mentor Graphics. 曾任相关EDA产品Calibre PERC,LVS, DFM 的首席软件架构师。他还担任过谷歌高级软件工程师。Fedor是公认的高性能计算和 C++ 专家。经常在CPPCon等技术会议和软件开发期刊上发表过演讲和文章。Fedor 拥有超过 25 项专利和 100 多篇关于物理、EDA、软件设计和 C++ 的论文和演讲。

主办方：Boolan 高端IT咨询与教育平台

Design for Performance

Fedor G Pikus
Technical Fellow, Siemens Digital Software Industries
September 2022 – CppSummit

SIEMENS

Background

- I design Electronic Design Automation (EDA) tools
  - Used by designers and manufacturers of integrated circuits (IC)
  - Apple, Qualcomm, Intel, TSMC, Samsung, ATI, Nvidia, etc
- Performance is the most important metric after feature capabilities
  - More important than … anything else?

Customers On Correctness

Customer: We want your software to do this and this.

EDA Vendor: We could do it this way but it’s not always the right result.
C: OK, we will test it...

... 2 days later ...

C: We ran it on all our designs, it works. Do it!

EV: How many designs is that?

C: Three.

EV: What if it doesn’t work on another design?

C: This never happens.

Customers On Performance

Customer: How fast is your tool?

EDA Vendor: 12 hours on your largest design, less on others.

C: How many designs is that?

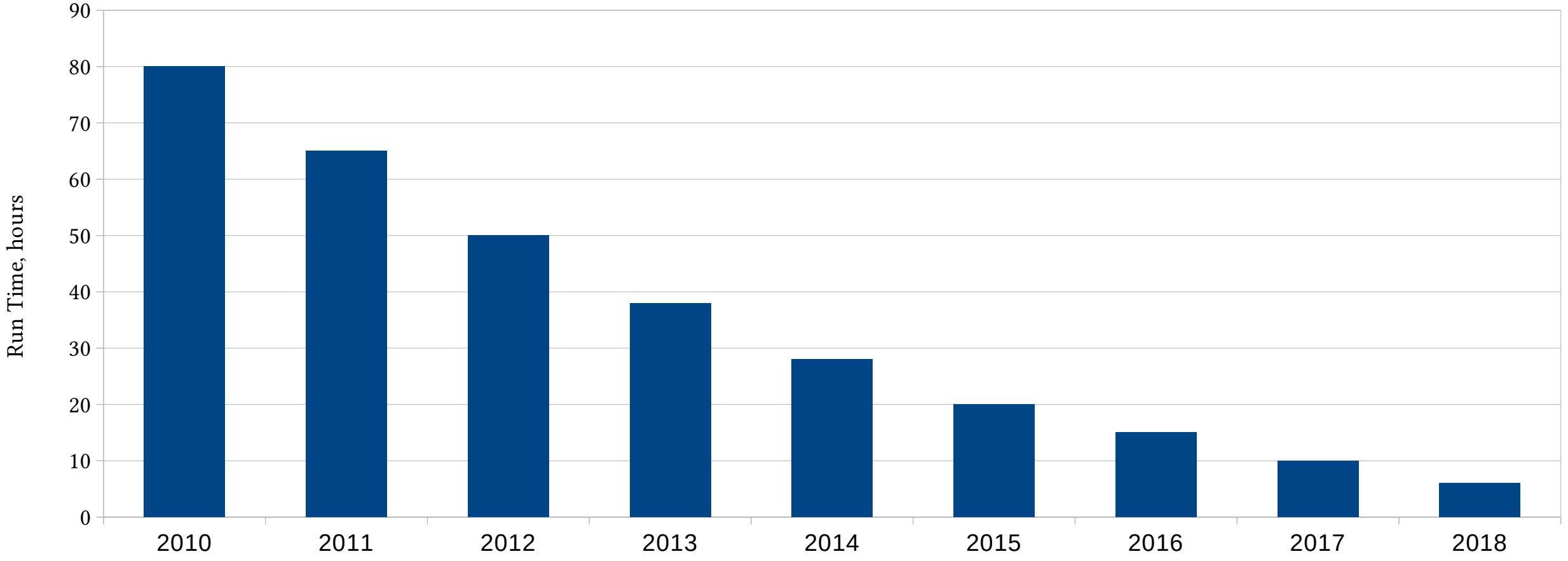
EV: 10. And 20 more from your competitors.

C: What if we get an even larger design?

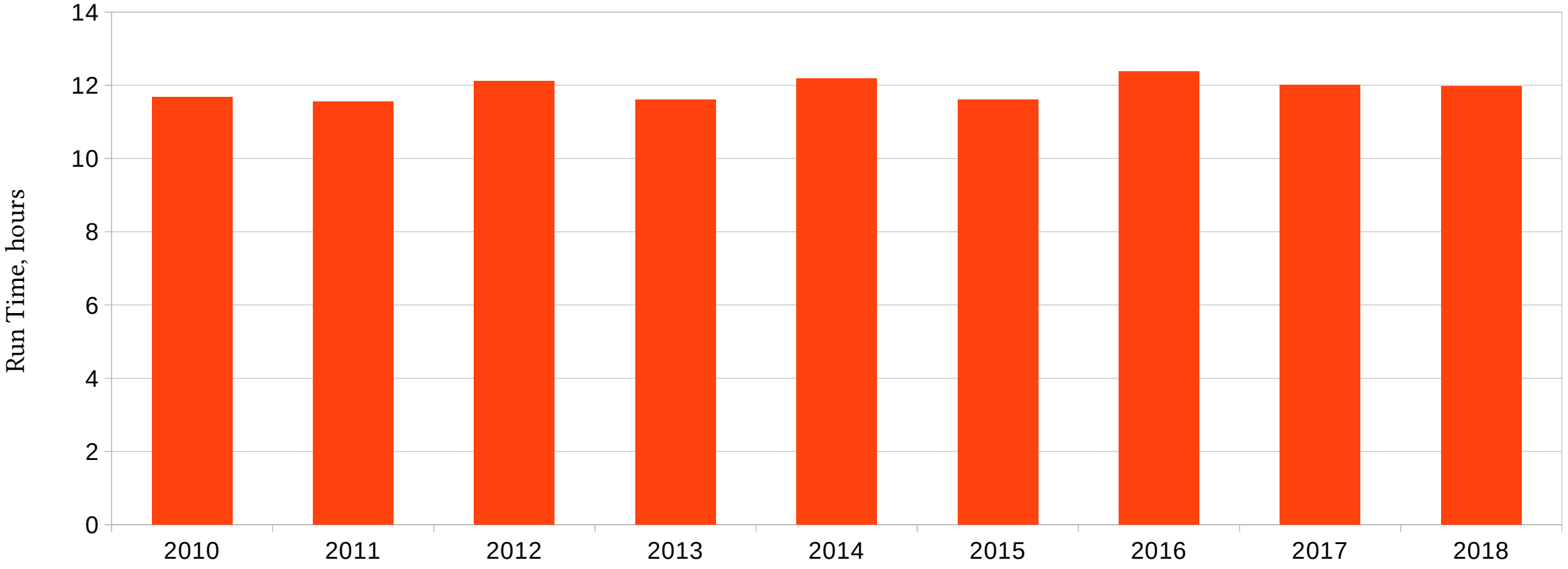
EV: On all your designs and other designs it runs in under 12 hours

C: But what if some day we get an even larger or a very unusual design?
Write us another tool that can handle even that in 12 hours!

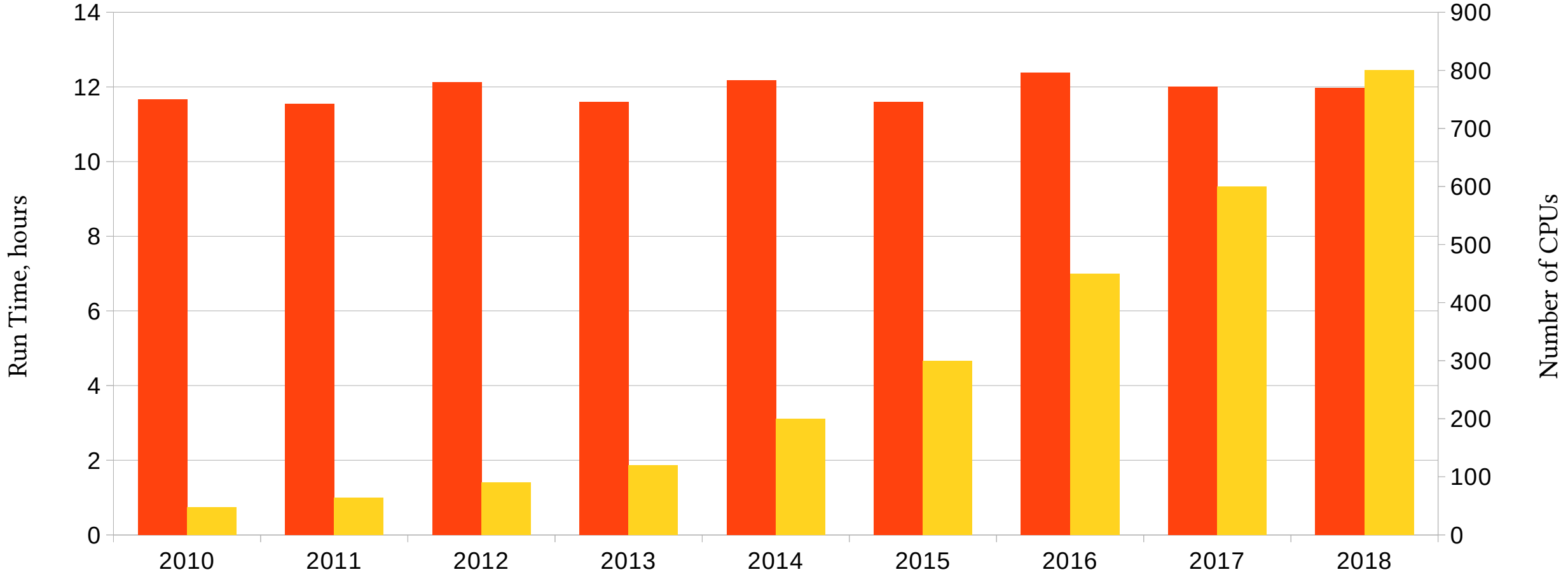
EDA Performance Progress



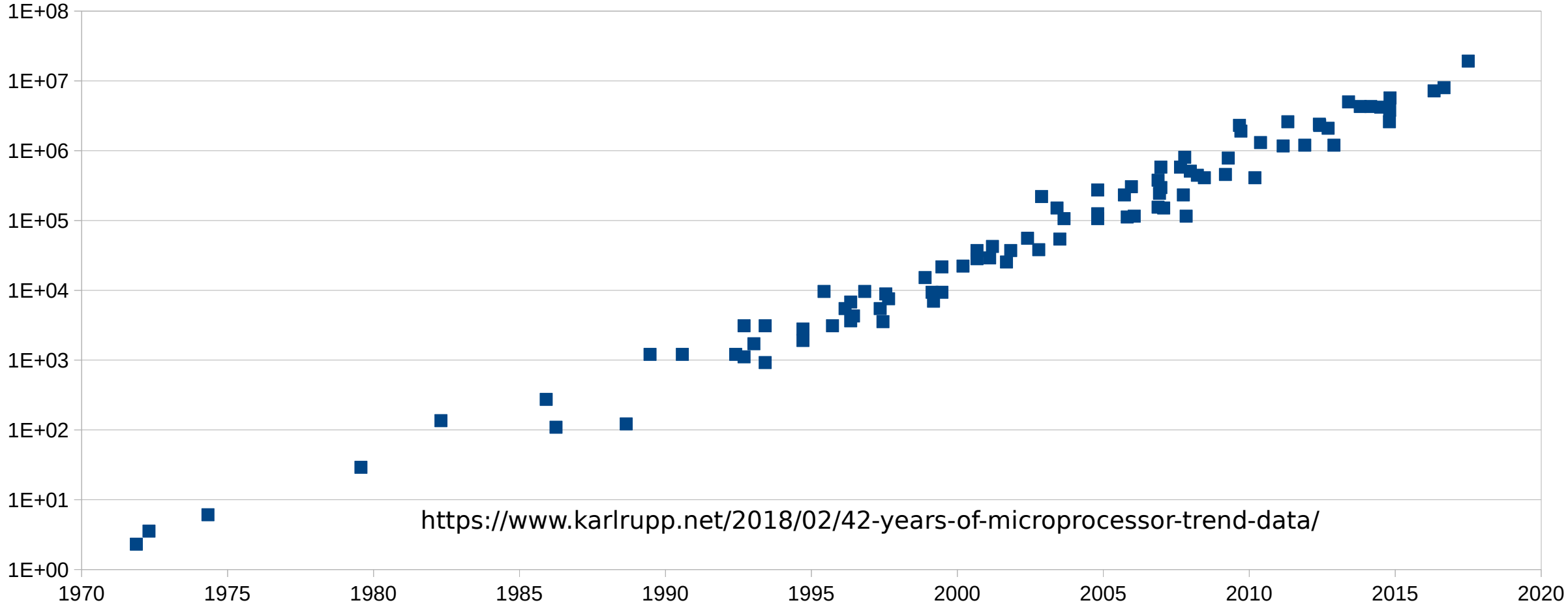
EDA Performance On The Largest Design



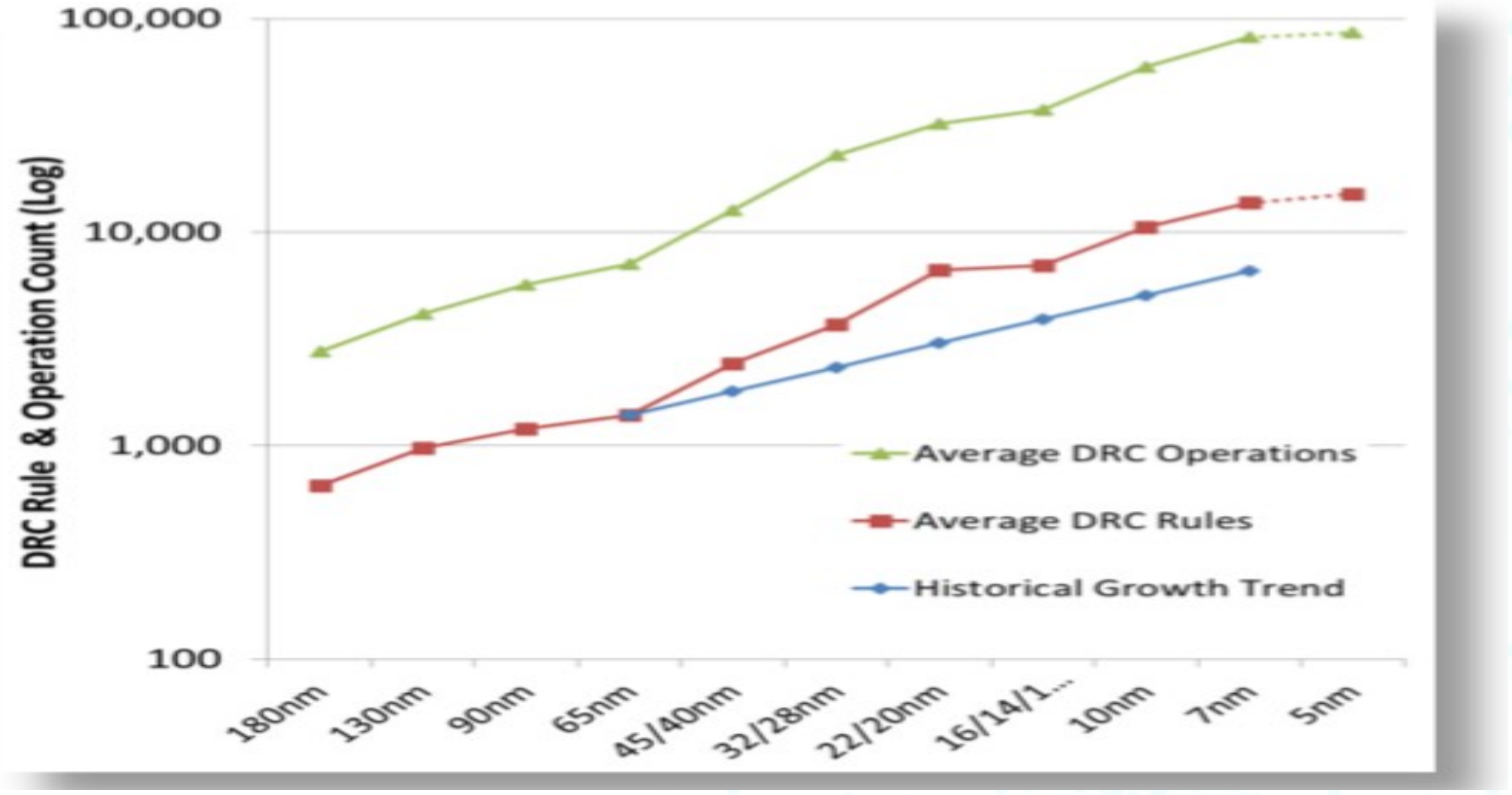
EDA Performance On The Largest Design



Largest Design - Transistor Count



Largest Design – Complexity



Long-term Development And Support

Technologies		LVS, DevX, RCX, Rcred	DSPF, Proc	
Mentor Graphics	Calibre	DRC, ERC, LVS, DevX	GDSII, Spice, SVRF, Proc	GDSII, text
	XCalibre	RCX, DevX, RCred, Tim	GDSII, SVRF, Proc	Spice, SPF, SDF, Text

1996

TSMC Adopts Mentor Graphics Calibre Physical Verification Tool Into Its SoC Design Flow

IP News

2000

TSMC Adopts Mentor Graphics Calibre Physical Verificatio

2018

Mentor launches Calibre RealTime Digital to help cut weeks off of IC design signoff

Mentor, a Siemens business, today announced Calibre® RealTime Digital – a new physical verification tool that works in concert with popular commercial place-and-route environments to ensure "Correct-by-Calibre" routing, and help design teams cut weeks off of IC signoff.

Software Design == Survival

- Current crisis can be handled by skillful implementation
  - Often a skillful hack
- Solving immediate problems as they arise accumulates technical debt that must be paid off later
  - Continued development becomes progressively more expensive
  - Maintenance becomes more difficult and uncertain
- Software evolves over time, but usually gradually
  - Complete rewrite takes the tool off market for the duration

Design Is The Foundation Of Software

- Design that lasts
- Adaptable to changes
  - Beyond what was expected
- Architecture of some products is the same after 10-15 years
  - A lot of features added
  - Some implementations rewritten
  - Components and their interactions, interfaces, data flows remain



Design Is The Foundation Of Software

- Not necessarily **good** software
- Adaptable to changes
  - Not always organized
- Architecture of some products is the same after 10 years
  - Just as chaotic as it was then
- Usually a crisis is needed to justify the necessary rewrite



Design Is The Foundation Of Software

- Software is NOT like a house
  - You can build it without a foundation
- After a while nobody can change anything
- Best case: one day it’ll break completely
  - Then we can rewrite it



Efficiency VS Performance

- Efficiency relates to not wasting resources
  - Not making unneeded computations
  - Not leaving hardware idle
- Performance relates to specific metrics

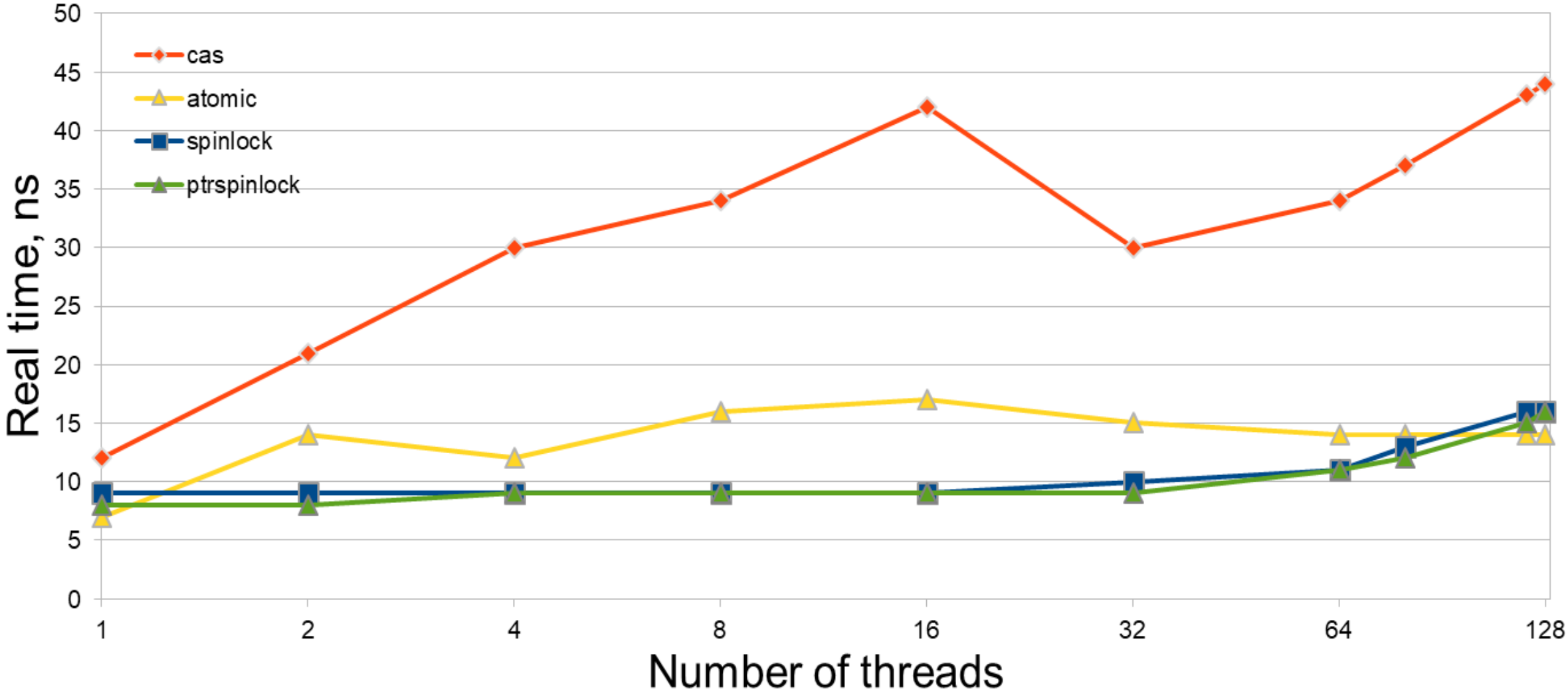
What Makes A Program Efficient?

- Good algorithm – this is most important
  - You cannot fix bad algorithm by optimization
- Using as much of the hardware as possible (in space and in time)
  - Processors have a lot of computing resources, and a lot of area dedicated to computing
  - Using as much hardware as possible, all the time
  - Computing resources are of no use if they are idle
- Not doing anything you don’t absolutely need to
  - Not as obvious as it sounds, there are a lot of subtle ways your program could be doing computations you do not need

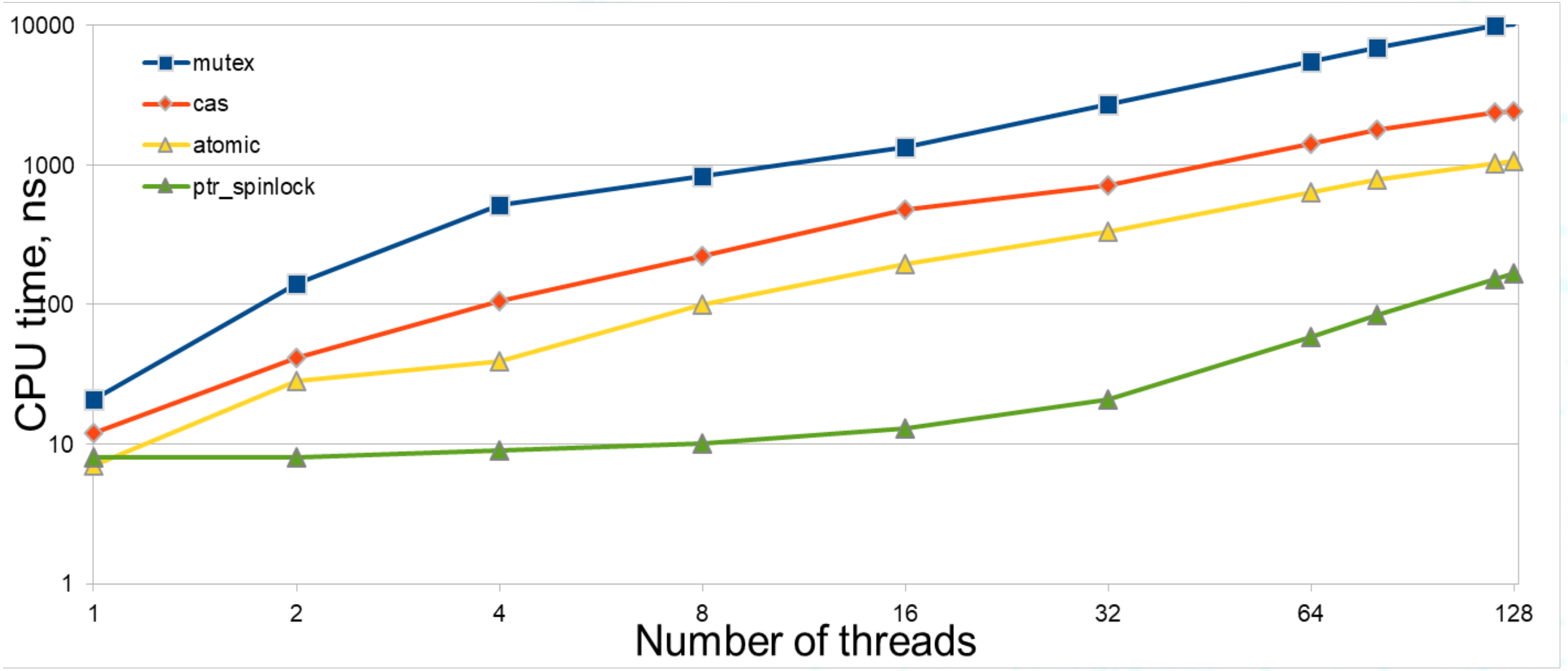
What Is High Performance?

- High performance depends on the application and requirements
- In HPC, usually refers to high throughput:
  - When will my results be ready?
- Often refers to power efficiency:
  - How much power is needed to get my results?
- In real-time or embedded systems, latency is more important than throughput
  - Often maximum latency

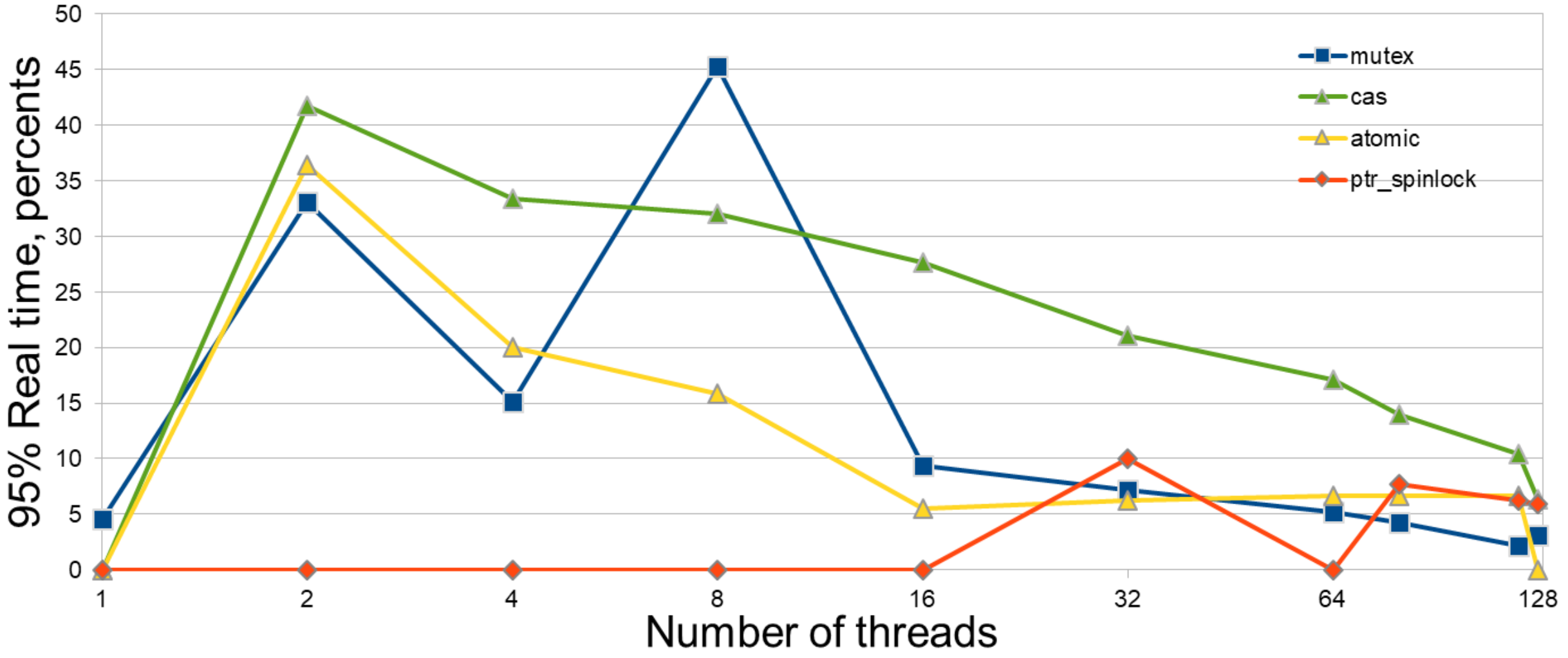
Multi-core Latency – HPC View



Multi-core Latency – Mobile View



Multi-core Latency – Real-Time View



Does Good Design Help Performance?

- We know the guiding principles of a good design
  - Encapsulation, modularity, separation of concerns…
- Do they help to write fast code?

Does Good Design Help Performance?

- We know the guiding principles of a good design
  - Encapsulation, modularity, interfaces, separation of concerns…
- Do they help to write fast code?
- Sort of : good design makes it easier to improve implementation
- A very good design can also perform poorly and resist optimization
  - Encapsulation, modularity, interfaces, etc must be designed with the understanding of their effect on performance
  - With a good design, the concern is to not lock in bad performance

Good Design Vs Design For Performance

- Performance must be one of the initial design objectives
- Not all otherwise good designs are high-performing
- Not all high-performance designs are otherwise good
- How to design for performance?

Good Design Vs Design For Performance

- Performance must be one of the initial design objectives
- Not all otherwise good designs are high-performing
- Not all high-performance designs are otherwise good
- How to design for performance?



How to avoid boring talks?

Design: Descriptive Or Prescriptive?

- Designs are like stories





Good Design Vs Design For Performance

- Let’s see some code...
- ... and try to make it faster ...
- ... and learn about benchmarking ...
- ... and profiling
- ... and what makes code fast ...
- ... or slow ...
- Along the way, we may discover few things abo performance



What Can Code Teach Us About Design?

- How can a specific chunk of code help with design?

First Rule Of Performance

- Never guess about performance
- Performance must be measured
- Program execution can be timed
  - Timer system calls, std::chrono in C++
- Programs can be profiled using profiler tools
- Performance of code fragments can be measured in microbenchmarks
  - Convenient: measures performance of the exact code fragment you need
  - But: performance may be affected by the context (other threads, prior memory allocations, compiler optimizations)
- All of these tools require code – but we are still working on the design

What Can Code Teach Us About Design?

- You need to change an old design to improve performance…
  Where to start?
- New design is influenced by knowledge about performance
  - A good new design will someday become an old design
- Knowledge about performance has to be gained or confirmed by measurements

What Limits Performance?

- Different factors can limit performance of a program
- Memory access: memory-bound programs
- CPU speed: computation-bound programs
- Memory use: program performance is limited by the amount of available memory, not by the speed of computation
- Input and output: I/O-bound programs
- Scaling and locking: concurrent programs

What Limits Performance?

- The distinction is not always clear:
- I/O-bound program that reads and writes compressed data
  - Real limit is the compression/decompression speed
- CPU-bound program with a lot of redundant computations
  - Could have saved intermediate results if we had a lot more memory
- Better scaling means more computations at the same time
  - More computations use more temporary memory, can run out

Code: Matrix-vector Product

```
void mv_prod(size_t N, const float* m, const float* v, float* res) {
    const float* r = m;
    for (size_t i = 0; i < N; ++i) {
        float& res1 = res[i];
        res1 = 0;
        for (size_t j = 0; j < N; ++j) {
            res1 += r[j]*v[j];
        }
        r += N;
    }
}
```

Code: Matrix-vector Product

- Is it memory-bound or compute-bound?
- How can we tell?

Code: Matrix-vector Product

```
inline float prod8(const float* a, const float* b) {
    union { float r[8]; __m256 rv; };
    rv = _mm256_dp_ps(_mm256_load_ps(a),
                                        _mm256_load_ps(b), 0xf1);
    return r[0] + r[4];
}
```

Code: Matrix-vector Product

```
float vv_prod(size_t N, const float* a, const float* b) {
    float res = 0;
    for (size_t i = 0; i < N; i += 8) { res += prod8(a + i, b + i); }
    return res;
}
```

Design Consideration

- Prefer high-granularity API
  - Restrictive: process one element at a time (make no assumptions that there will be more)
  - Permissive: process all these elements
- Alternative – process one element at a time but finalize explicitly
  - Ewww...

Design Consideration

- Aligned memory is needed!
  - Depending on how memory is allocated, could need a major rewrite
  - In general, memory management is a constraint in many designs, needs to be considered early

Code: Matrix-vector Product

```
inline float prod8(const float* a, const float* b) {
    union { float r[8]; __m256 rv; };
    rv = _mm256_dp_ps(_mm256_load_ps(a),
                                        _mm256_load_ps(b), 0xf1);
    return r[0] + r[4];
}
```

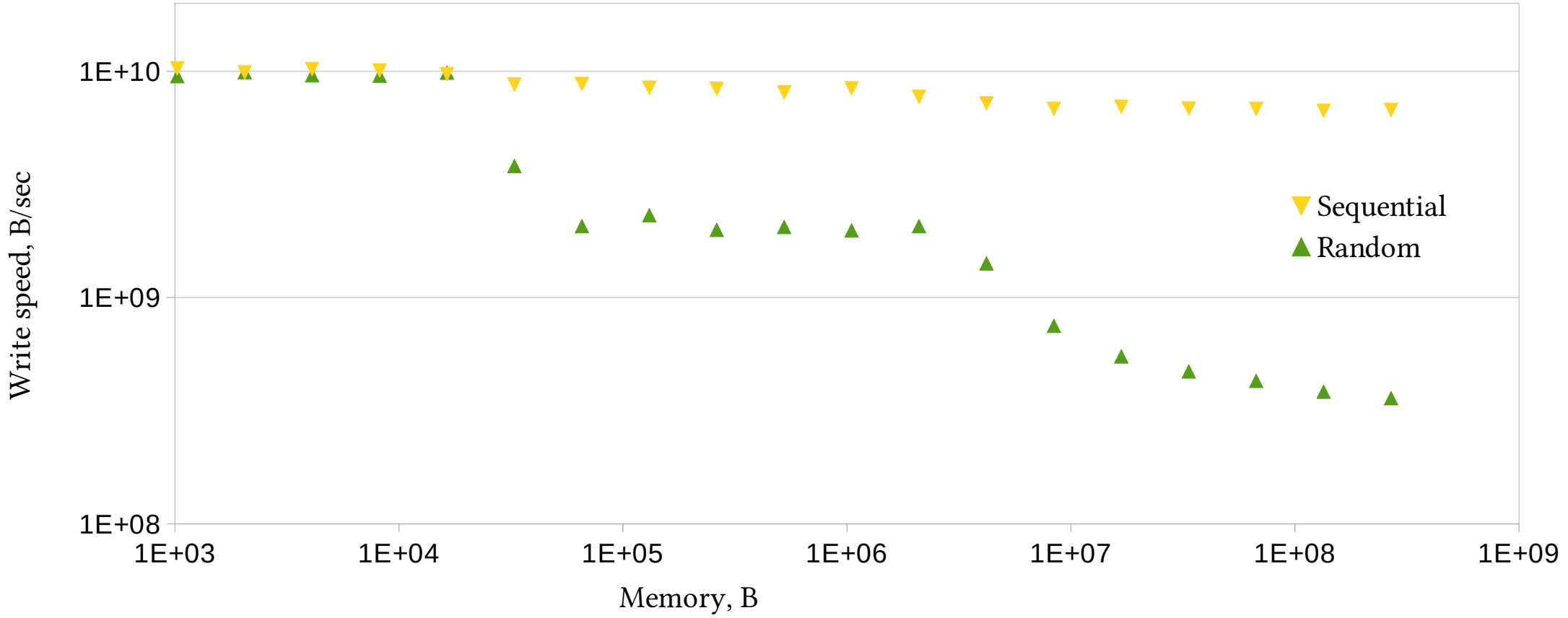
Design Consideration

- High-performance code is often low-level
  - Sometimes non-portable
  - Sometimes abuses the language
- Good design contains such C++ code
  - Esoteric knowledge and arcane code contained in a small part of the code
  - Higher level abstractions provided for the rest of the code
  - Take care: abstractions and APIs must not over-constrain the implementation or force overhead

Design Consideration

- Memory access speed depends on how memory is accessed
  - Is the accelerated code fast because it computes faster, or because it reads memory faster? Some of both
  - Memory access speed can be a bottleneck, needs more thought

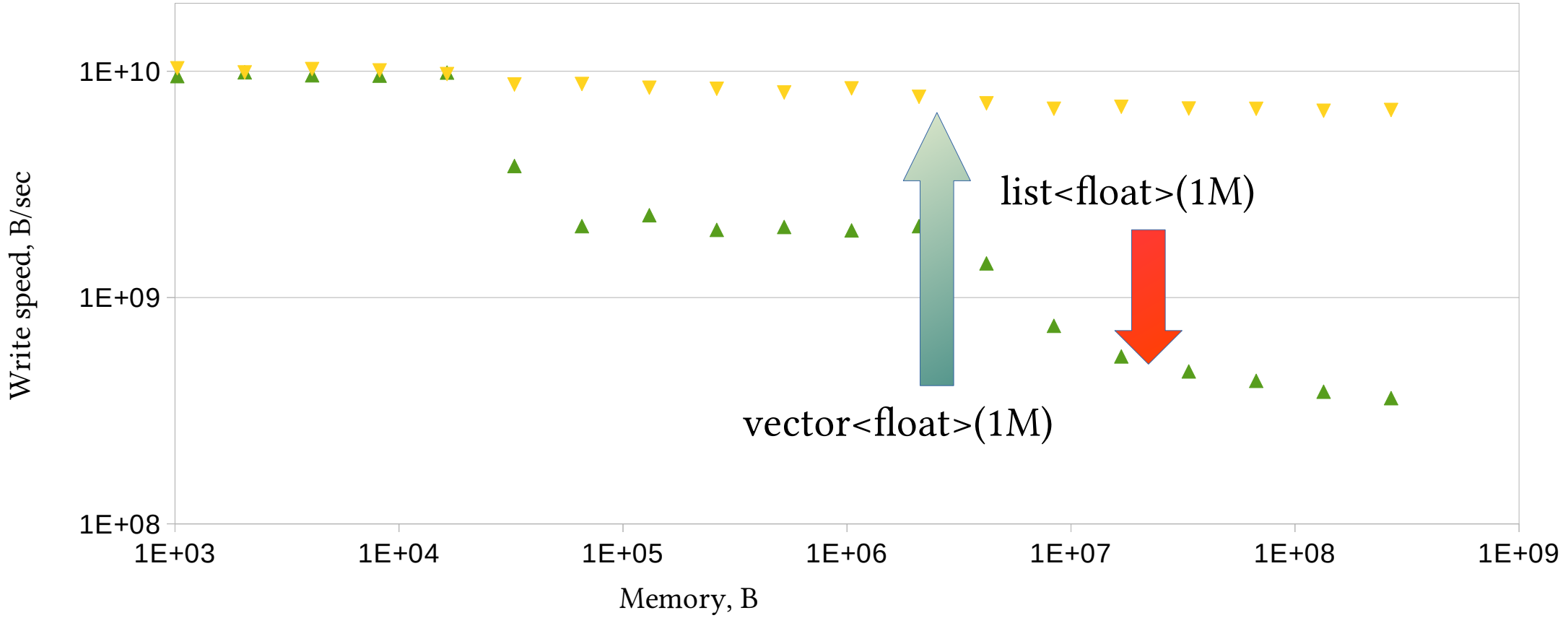
Memory Speed



Design Consideration

- Data organization is chosen during design
  - Data organization and data structures determine possible ways to access the data, and their efficiency
  - Ways to access the data determine which algorithms can be used

Memory Speed



More Code

- Can it really be that bad?

More Code

- Can it really be that bad?
- It’s worse.

Performance Made Simple

1) Run the profiler
2) Find the step that takes most time
3) Optimize the code
4) Run the profiler again



Performance Made Hard Again

1) Run the profiler
2) Find the step that takes most time
3) Optimize the code
4) Run the profiler again
5) Until nothing stands out
   - But nothing is very good either
6) Can it really happen?
7) What does it mean?



What To Optimize When

there Is Nothing To Optimize?

- When there is no hot code, there is hot data!
- Sometimes a single data access function stands out
  - But not its callers, calls are scattered all over
  - The access function itself may be impossible to optimize
  - At least within its current interface contract
- Two options: access less data or organize data differently
  - Both are design issues

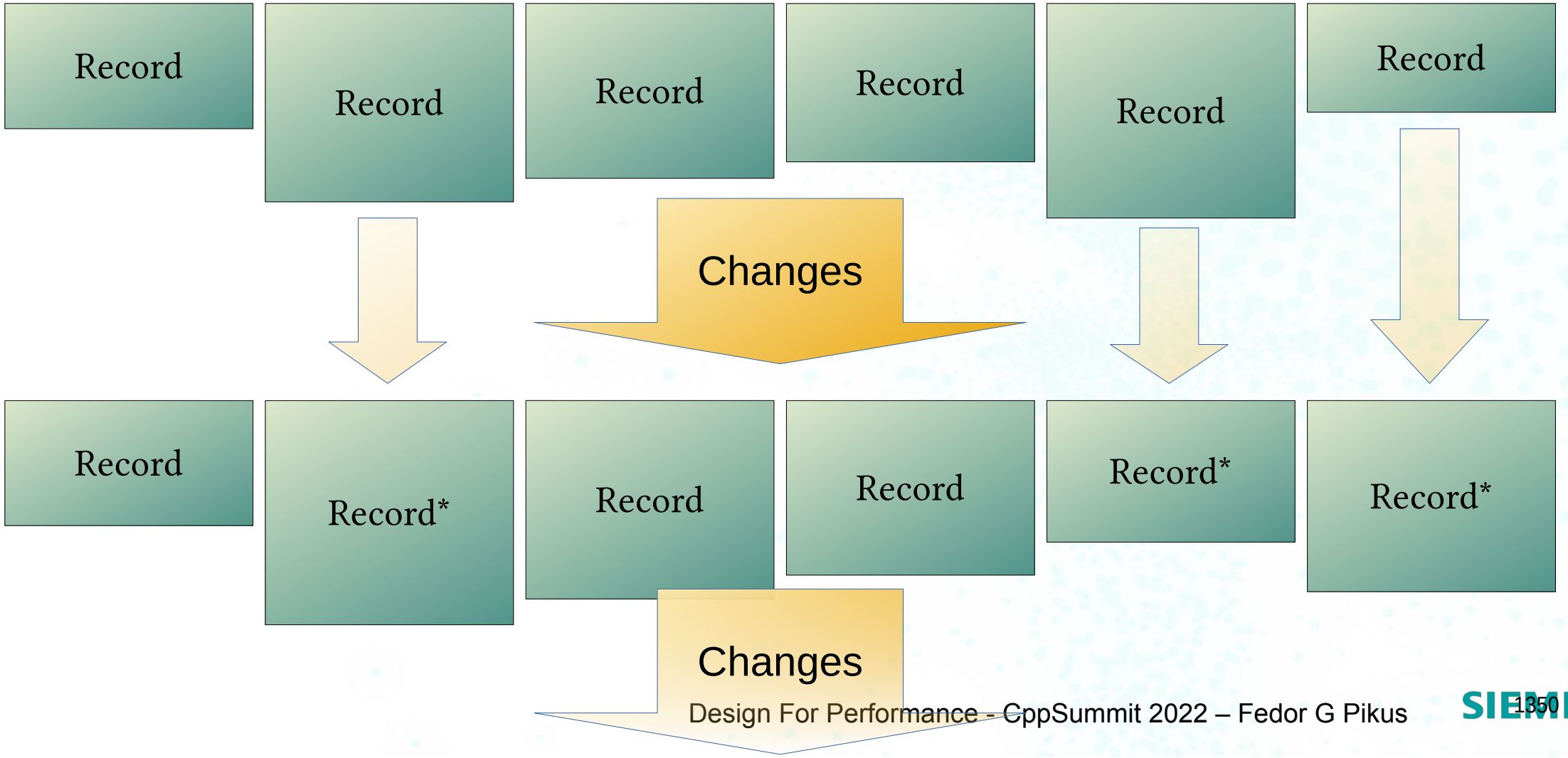
Design Considerations

- Design determines the data flow at the high level
  - What are the components of the system?
  - Who has what information, and who needs it?
  - What are the data flows?
- Data organization determines what information can be accessed
- Data access requirements limit performance
  - API which exposes implementation can limit performance

Design Considerations - Example

- `std::unordered_map` – a hash table
  - It has hashing functions
- `insert(), find(), erase()` – normal hash table stuff
- Bucket iterators…
  - `begin(n), end(n)`
  - It's a list!
- All I wanted was a hash table, the API restricted the implementation
  - There are implications for performance

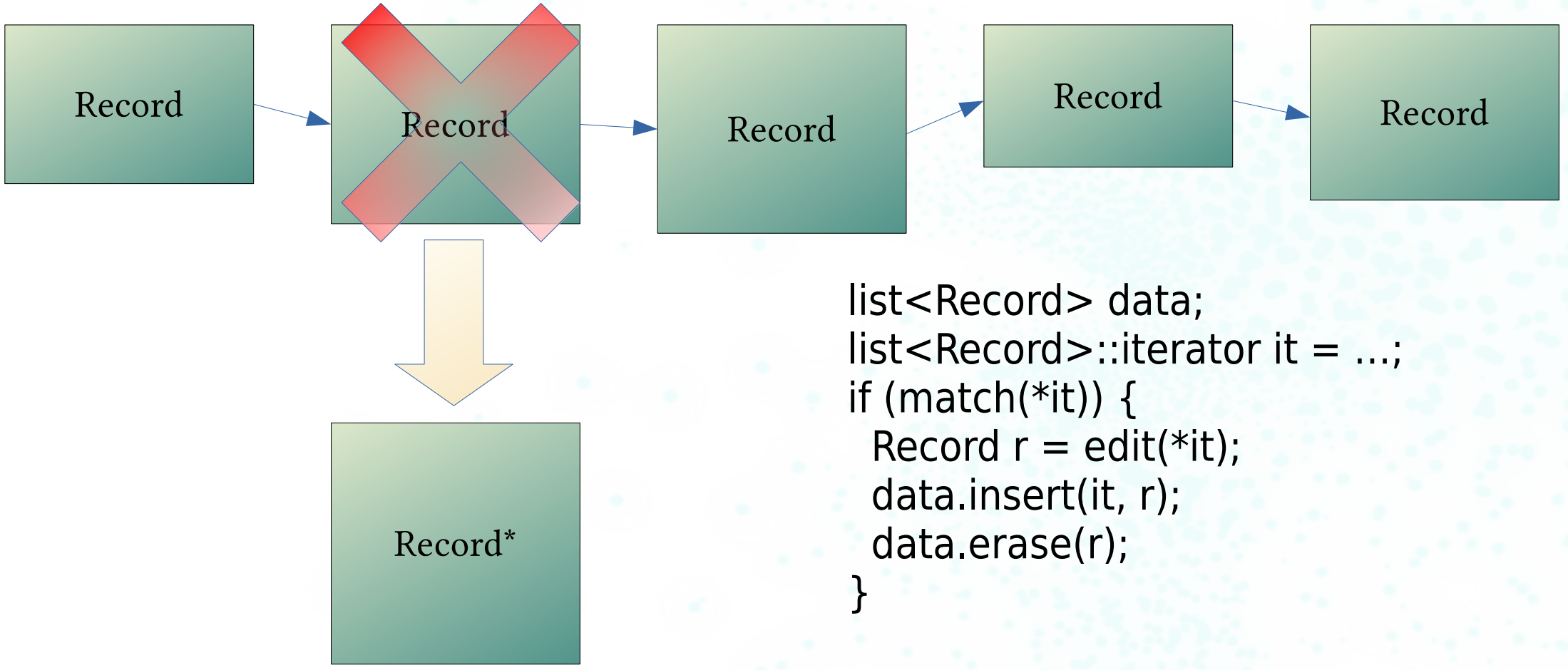
Design Example



Design Example – Random Access



Design Example – Random Access

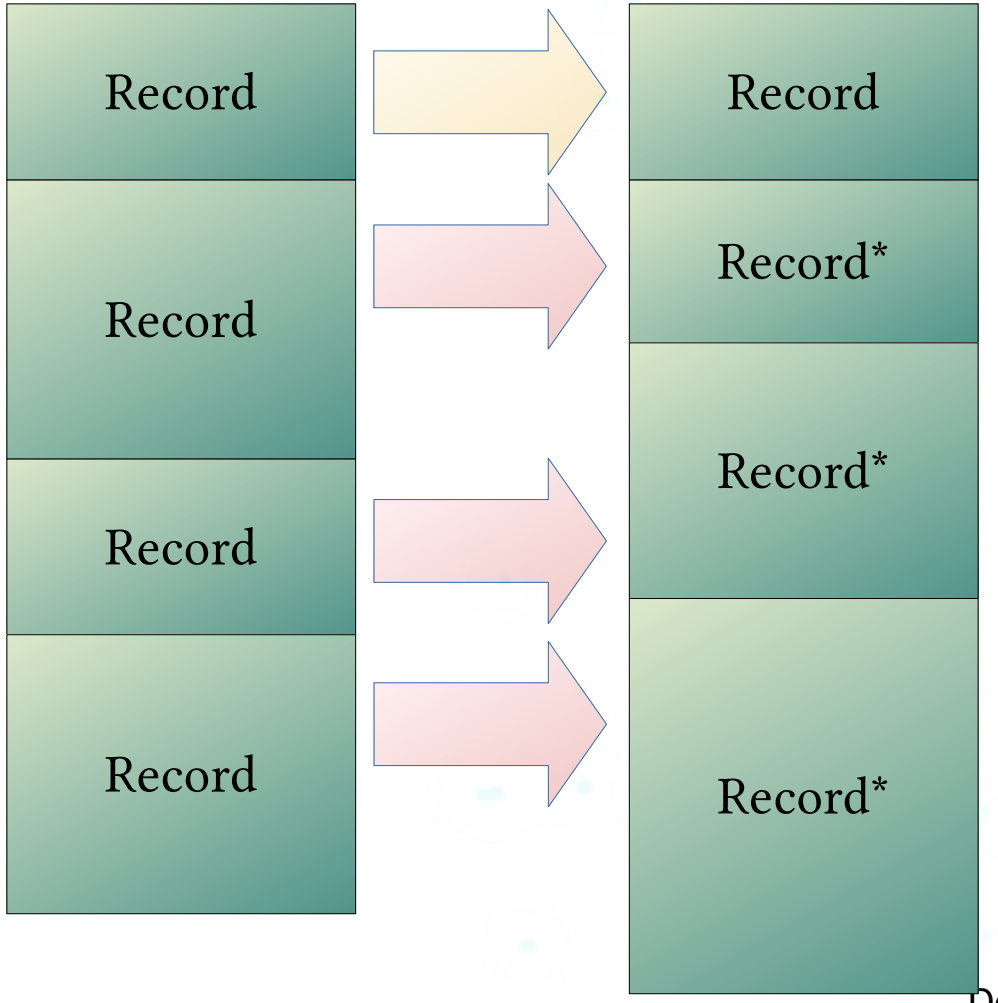



```
list<Record> data;
list<Record>::iterator it = ...;
if (match(*it)) {
  Record r = edit(*it);
  data.insert(it, r);
  data.erase(r);
}
```

Design Example – Random Access

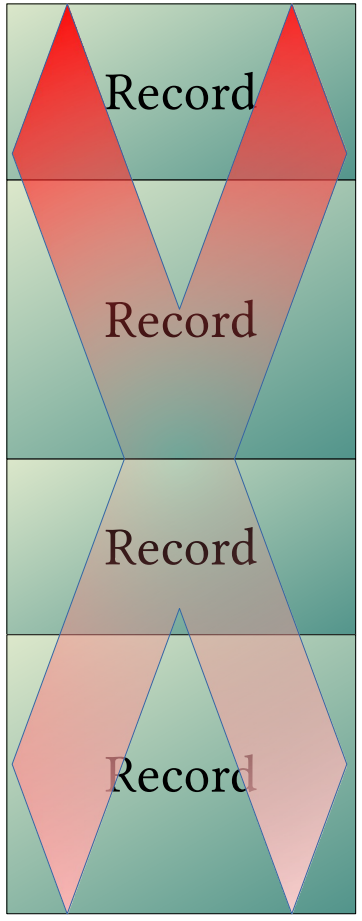


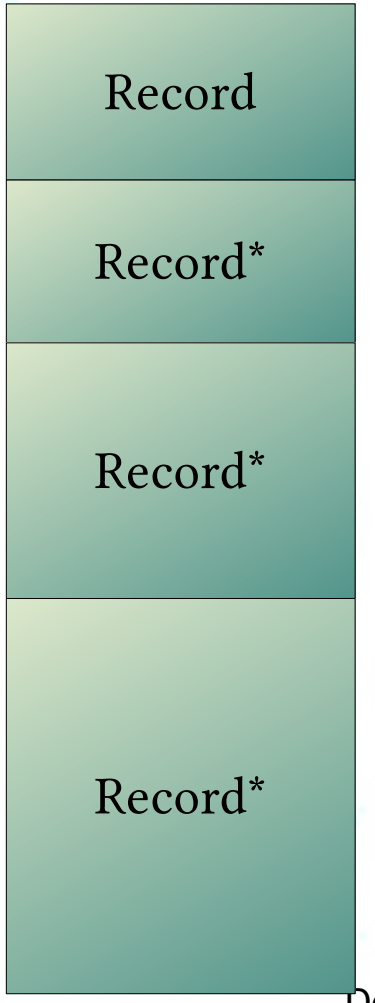
Design Example – Streaming Access




```
Record* data;
Record* new_data;
if (match(data)) {
  *new_data = edit(data);
} else {
  *new_data = *data;
}
```

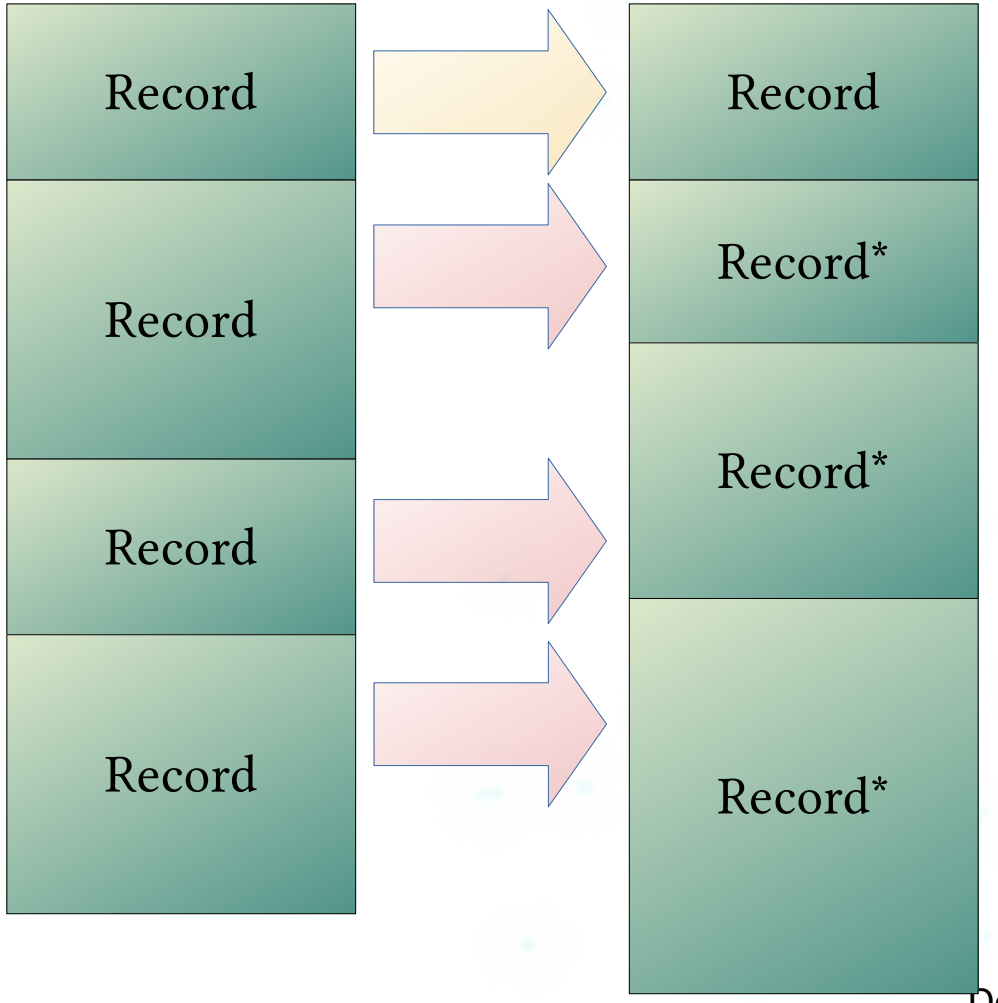
Design Example – Streaming Access





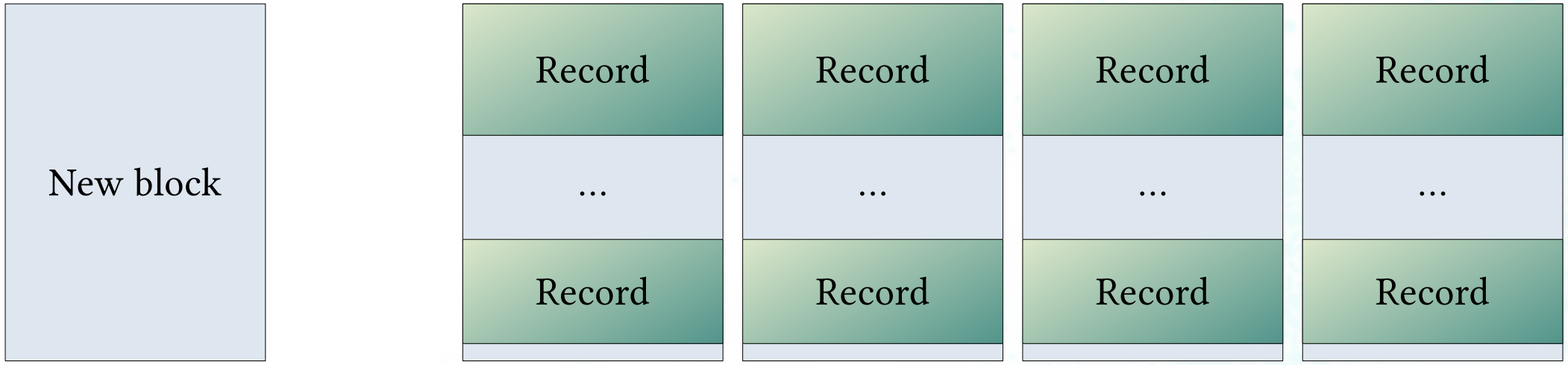

```
Record* data;
Record* new_data;
if (match(data)) {
  *new_data = edit(data);
} else {
  *new_data = *data;
}
deallocate(data);
data = new_data;
```

Design Example – Streaming Access

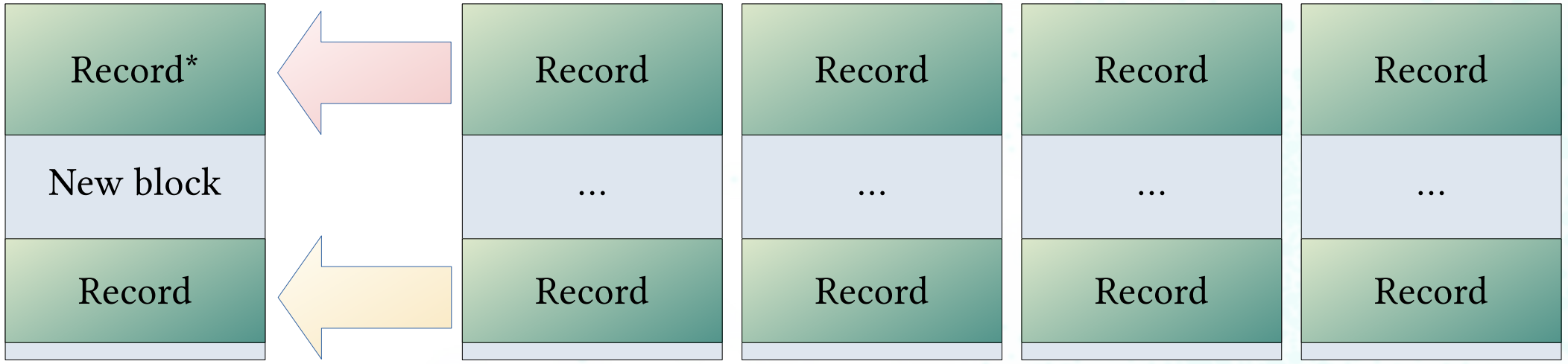


- Consumes twice as much memory at the peak usage
- Unless we can recycle it…

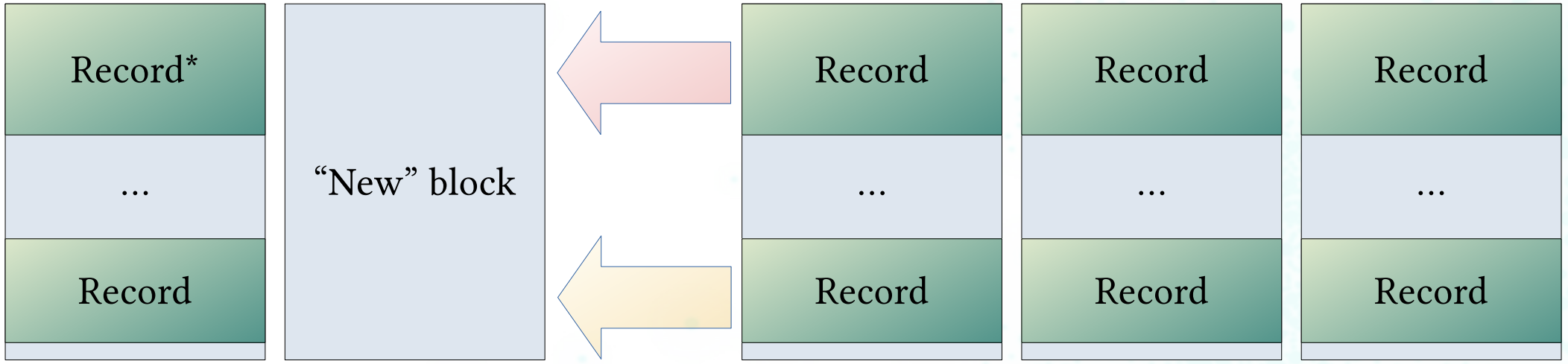
Design Example – Block Streaming Access



Design Example – Block Streaming Access



Design Example – Block Streaming Access



- Extra memory is limited to one block
- Most recently used memory is reused (hot in cache)

Design Considerations

- Restricted data access allows for a more efficient implementation
  - Data can be organized for optimal access performance
- Restricting data access also restricts available operations
  - Some operations will need additional memory accesses or copying
  - Performance difference between optimal and random access is huge
- Temporary data conversion to a different format may be required
  - If these conversions are too frequent, a different design is needed

Design Considerations

- Design for performance must be influenced by measurements
  - You cannot make good decisions without good knowledge
- Measuring performance requires a working program
  - Implementing a program requires design
- Solution: prototypes, mocks, and experiments
  - Example: memory write speed (just need to know access patterns and data volume, does not need to be real data or real use of data)

Design For Performance - Examples

- Design for performance must be influenced by measurements
  - Saving temporary data to disk: what is the minimum time to write and read the expected volume of data?
  - Compressing data in memory: how long does it take to decompress?
  - Extensible hierarchy via run-time polymorphism: what is the overhead of the virtual function call?
  - Data organization allows limited access: what is the cost of an extra pass over all data (extra copy, compression/decompression, etc)

Design For Concurrency



SIEMENS

Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use undefined behavior to improve performance!

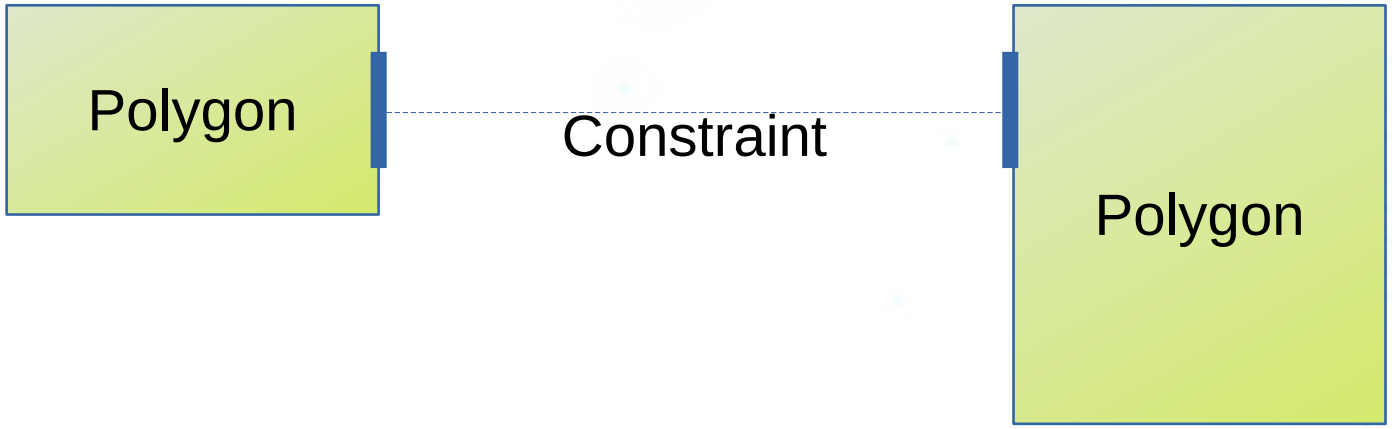
Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use undefined behavior to improve performance!

```
a[i++] = f(i=10, a=new int[++i]);
```

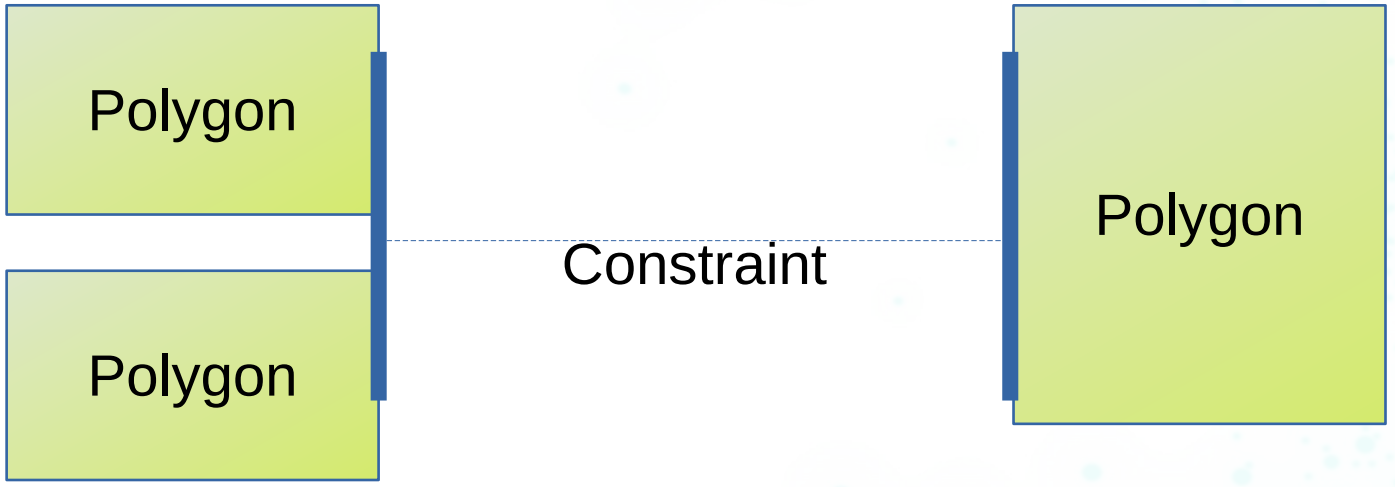
Not this kind of UB!
Ansel&Barbara

Undefined Behavior Example



- Operation: build a constraint graph where polygons are nodes and constraints are edges

Undefined Behavior Example



- Edge connects 2 nodes. Not 3. Undefined behavior!
- Expensive to handle consistently. What really happens?
  - An edge can connect any two nodes. Or none at all.

Undefined Behavior

- Undefined behavior is really a narrow contract
- If the inputs are in contract, correct behavior is guaranteed
- If the inputs are out of contract, anything can happen
- Use narrow contracts to improve performance!
  - Narrow contracts (undefined behavior) allows the implementation to be inconsistent where unnecessary consistency would be expensive
- Provide a way to detect undefined behavior

Design For Performance

- Design for performance must be influenced by measurements
- Data organization and information flow limit peak performance
- APIs should not expose or mandate the implementation
- APIs should present logical tasks or transactions
  - Preferably with large granularity
- Contain exotic code in few encapsulated areas of the system
- Good design practices are (mostly) good for performance too
  - Encapsulation, minimizing interactions, modularity
- Use narrow contracts (undefined behavior) to your advantage

Branch Prediction Optimizations Are Advanced And Must Be Justified

```
v3[i] = 0:
```

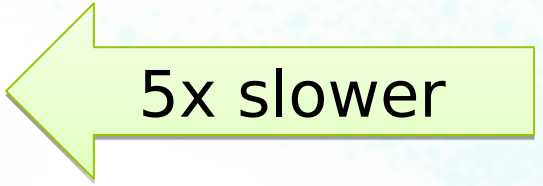
```
$ perf stat ./branch_predictions
```

- 0.05% branch misses

```
v[3] = rand():
```

```
$ perf stat ./branch_predictions
```

- 9% branch misses




```
$ perf record -e branches,branch-misses ./branch_predictions && perf report
```

- Optimizations to eliminate conditionals are usually invasive and may use more memory
- Branch predictors are quite complex; do not optimize until misprediction is confirmed by a profiler

Data Structures And Algorithms Determine Memory Access Patterns

```
std::vector<unsigned long> v1,v2;
for(size_t i=0;i<N;++i)v1.pushback(i);
for(size_t i=0;i<N;++i)v2[i]=v1[i];
std::list<unsigned long> l1,l2;
for(size_t i=0;i<N;++i)l1.pushback(i);
for(size_t i=0;i<N;++i)l2[i]=l1[i];
```

contiguous allocation

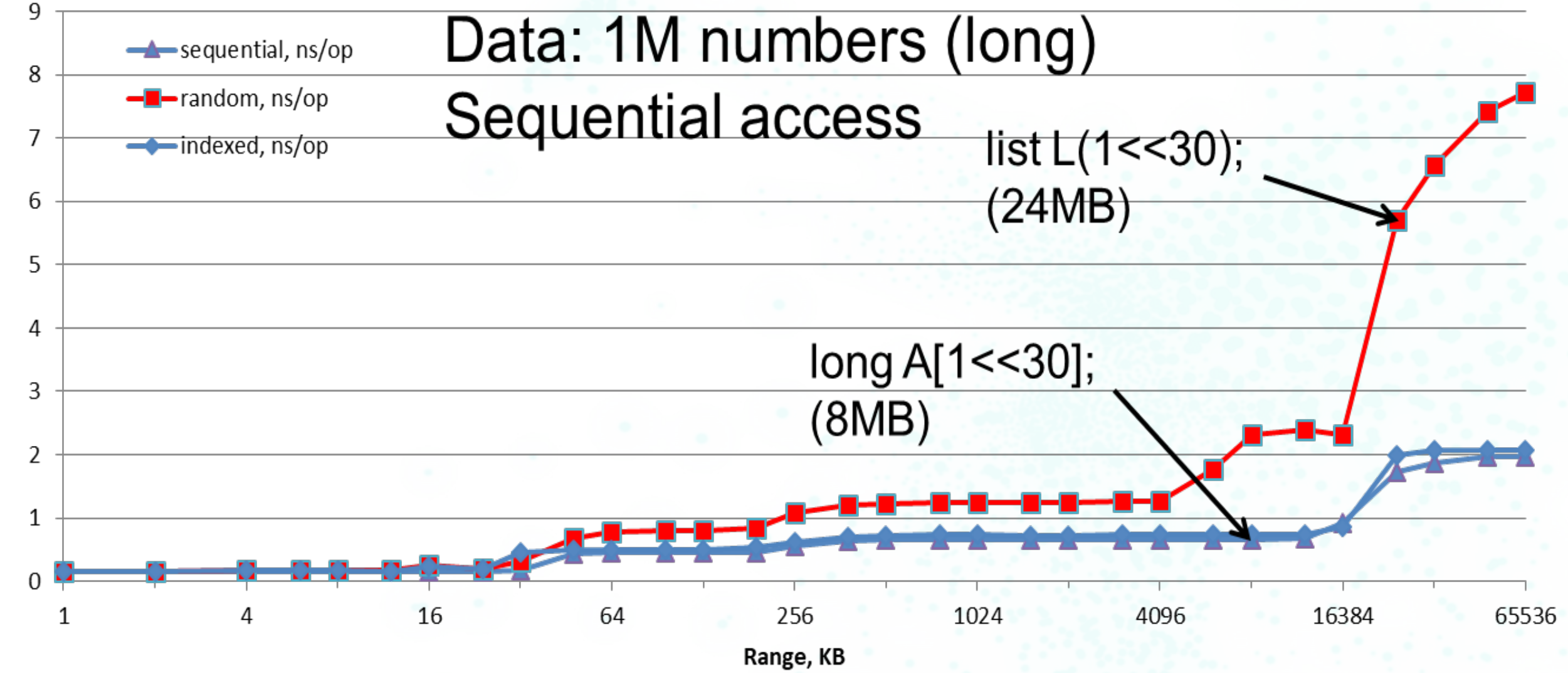
sequential access

random allocation

random access

- Choice of algorithms is often constrained (most sorts will have random access) but memory access should be considered
- Implementation of data structures is often determined by memory layout and memory access costs

Data Structures == Access Patterns



Set-associative Caches In Modern CPUs

- Most CPUs have N-way associative L1 and L2 caches with N=2, 4, or 8
  - L3 caches are more commonly directly mapped (1-way)
- Types of cache misses:
- First miss: first access of a random memory address
- Capacity miss: to cache this access, another address must be evicted from cache
  - Prefetch is intended to counter this situation
- Conflict miss: N+1st access to an address in the same N-way set
  - Cache may be under capacity but something must be evicted before new address is cached
- Conflict misses show up in sequential accesses with stride
  - Some strides are much worse than others
  - The worst stride is where set bits are the same and tag bits change on every access
  - Accessing addresses what differ only in offset (same set and tag) is very fast

What To Optimize When There Is Nothing To Optimize?

- When there is no hot code, there is hot data!
- Sometimes a single data access function stands out
  - But not its callers, calls are scattered all over
  - The access function itself may be impossible to optimize
  - At least within its current interface contract
- Two options: access less data or organize data differently
  - Both are design issues

Memory Access Is Important

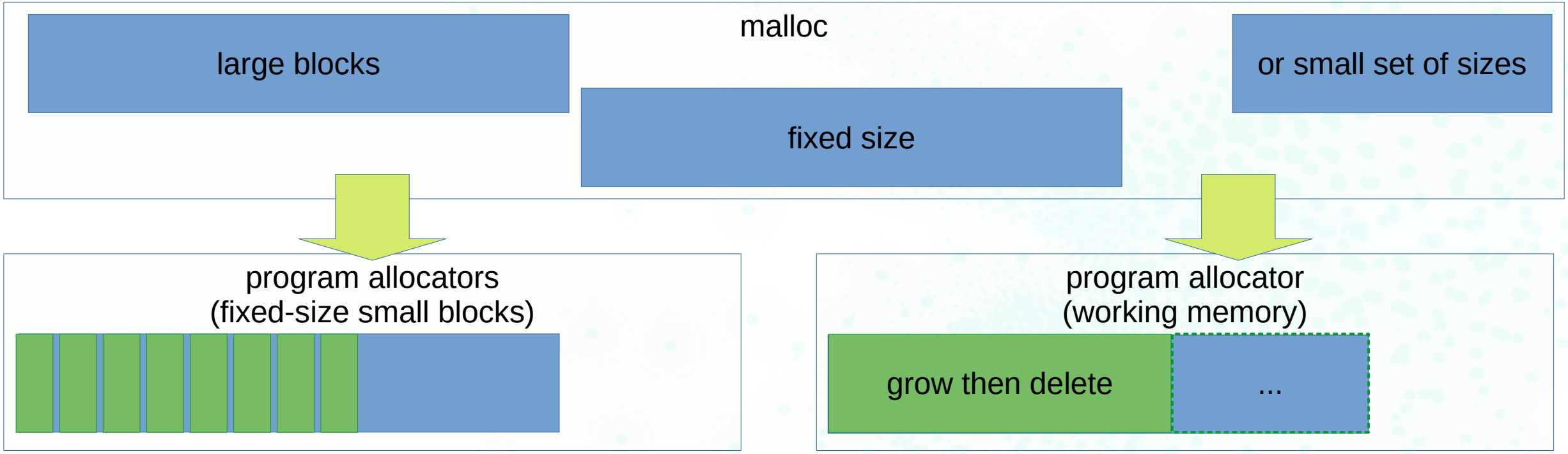
- The cost of accessing memory may be the most important factor influencing the data structure design (and even algorithms)
- Do as much processing as possible on a small data set
  - Random accesses have no overhead if data fits into L1 cache
  - Prefer to collect local computations together
- Operate on as much data as possible per instruction
  - A lot of the advantage of SSE/AVX stems from reading/writing longer words
- In memory-bound programs, memory size == performance. Reducing memory use, even by adding computations, translates into performance gains.

Memory Access Is Important

- The cost of accessing memory may be the most important factor influencing the data structure design (and even algorithms)
- Prefetch really works
  - Scanning memory forwards, backwards, or with a constant stride all work the same
  - Changing access patterns takes time for prefetch to adapt
- Organize data to process it sequentially
- Usually even at the cost of an extra read and write
- Almost always at the cost of additional computations
- Indirection and indirect indexing have non-trivial cost
  - Reading a value to access another value indirectly uses two cache locations
  - Indexes are frequently accessed and take up cache space

Program-specific Memory Management

- To avoid memory fragmentation, you have to limit your interactions with the system allocator
  - also helps to control memory locality and thread-memory affinity

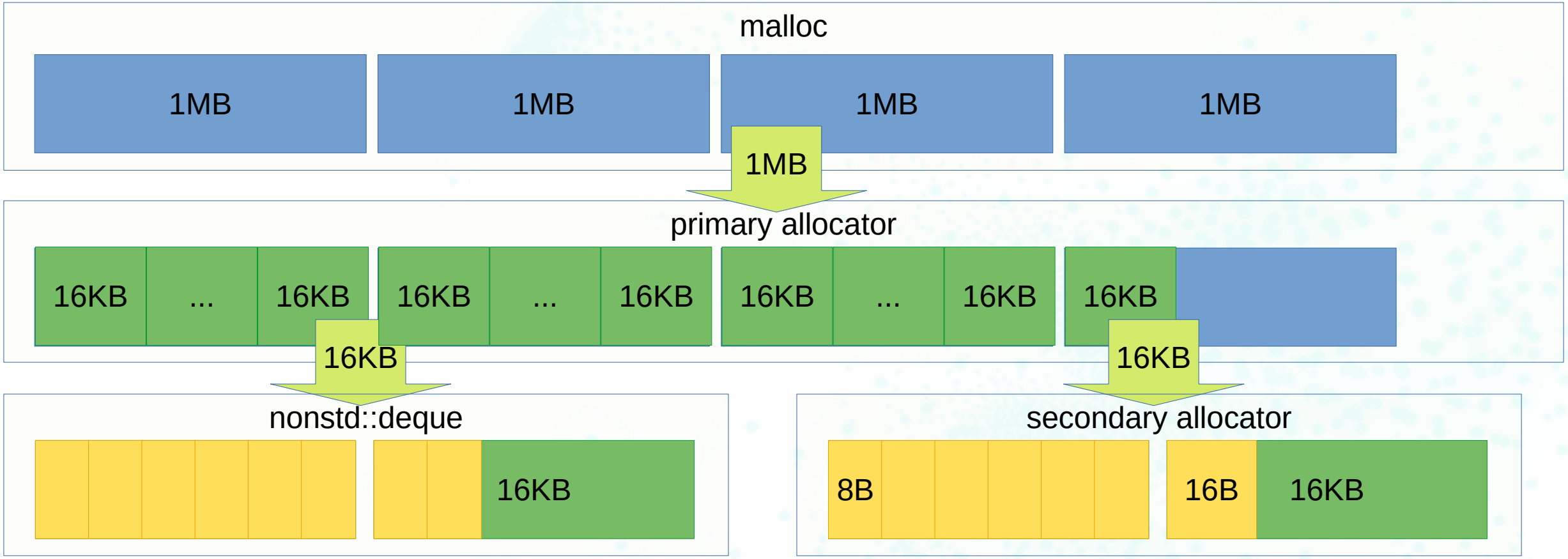


- Custom allocators are optimized for specific applications
- Design limitations are the trade-off for using simpler limited allocators

Program-specific Memory Management – Memory Pools

- Grow-only allocator, contiguous or block-contiguous memory
  - Allocates any size but interacts with malloc in a limited set of sizes
- The entire memory is released when the allocator is deleted
  - Object destructors still need to be called
- Particularly fast if used within one thread – no need for locking
- Useful for persistent data – never deallocate until the end of the program
- Useful for short-term working memory – allocate many temporary objects then free everything
  - User-driven garbage collection
- Specialized memory allocators are faster and more efficient because they are less generic
  - Speed comes at the expense of design flexibility

Program-specific Memory Management – Block Alocator



- No memory fragmentation, good memory reuse, great memory locality
- Limits on allocation size for the clients
  - Slab allocators are a generalization of this idea

Function Calls

- Regular function calls
  - Including static functions and member functions
- Inlined calls
  - Including member functions
- Virtual functions
- Indirect function calls via a function pointer

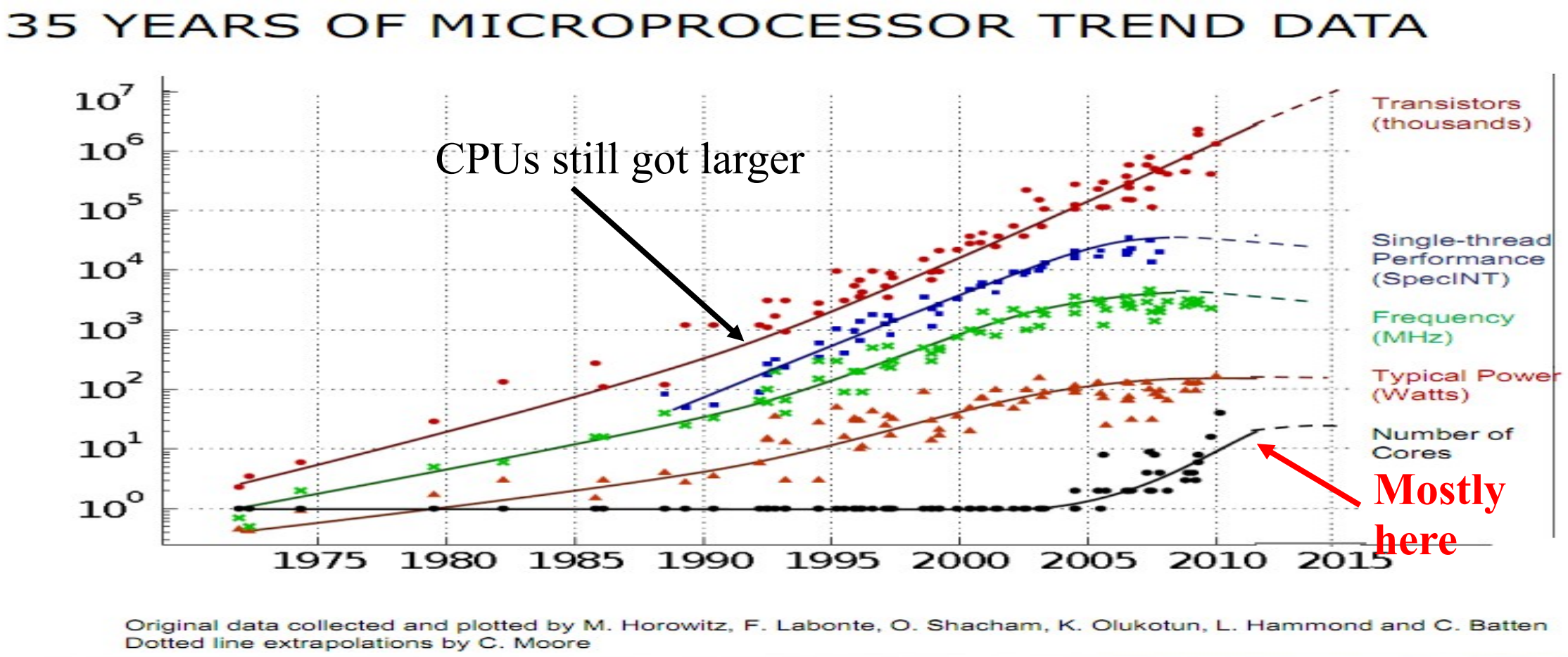
How Expensive Is A Virtual Function?

- 1.5x-2x slower than a regular function call
- 10x or more slower than an inline function call
- ~1x as `if-else`
  - Why are you using a virtual function if you don't need run-time dispatch?
- Faster than `if-else if-else if-else` (depends on the length of the chain)
- ~1x as `switch`
- `std::variant` – depends on the implementation
- Run-time vs compiler-time polymorphism is often dictated by the design
  - Not by a specific implementation

Memory And Caches And Concurrency

- Memory and caches in multi-core systems
- Cache hierarchy and memory bandwidth in multi-core systems
- Data races and synchronization
- Cost of data sharing and synchronization
- Concurrency and memory model

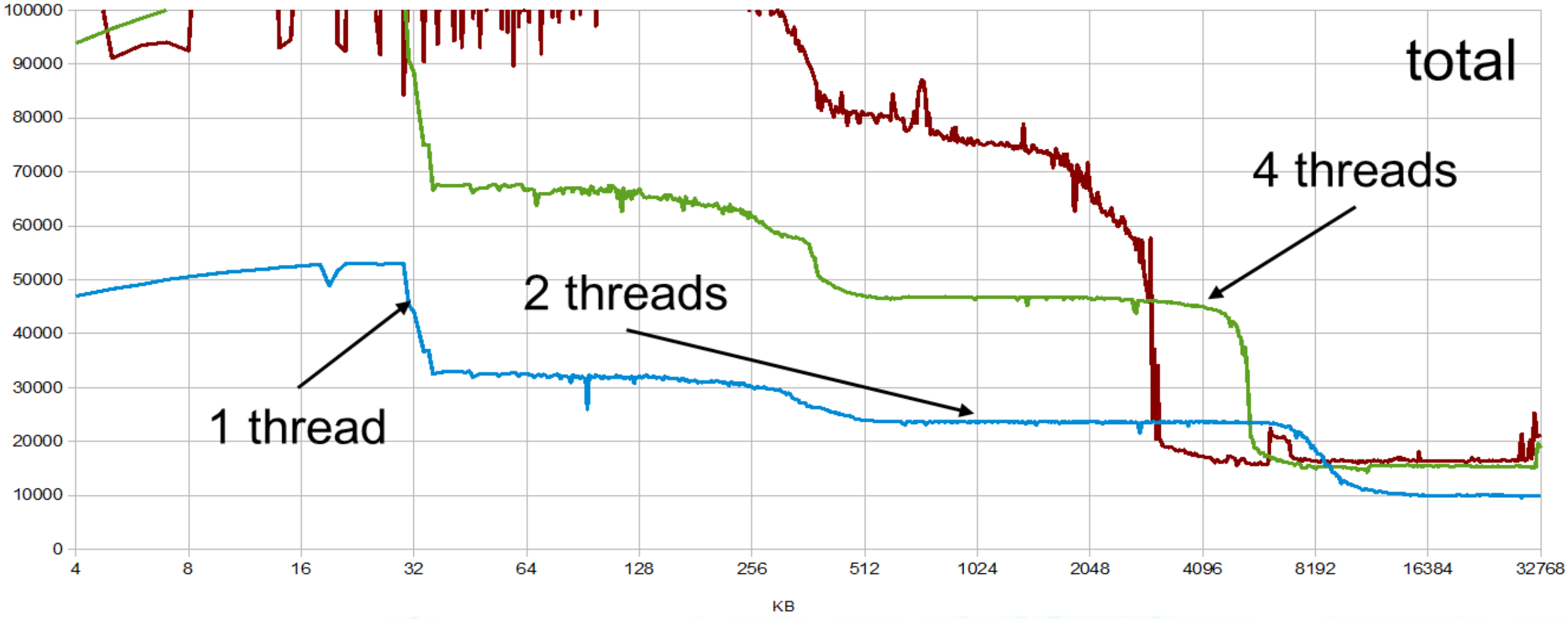
Then What Happened?



Most Advances In Cpus Are In Parallelism: Concurrency And Vectorization

- Vectorization is processing more data per instruction
  - MMX, SSE, AVX
- Concurrency is running multiple threads of execution
  - Multiple cores and Simultaneous Multi-threading (SMT)
- Multiple threads must be programmed
- Threads interact through memory
- Interactions of threads through memory is defined by the memory model
- Understanding the memory model is essential to be able to write correct and efficient concurrent programs

Aggregate Memory Bandwidth



Concurrent Programs Are More Sensitive To Memory Access

- There is not enough memory bandwidth to supply each core with data at full rate
- Caches are shared between multiple cores
- So far, we have only considered the case of perfect parallelism: thread computations are completely independent

Correct Code

```
unsigned long sum = 0;

std::mutex M;

void thread_work() {

    std::lock_guard<std::mutex> L(M);

    ++sum;

}
```

- No data races
- Does not scale at all:
  - Time to do N increments on one thread: t1
  - Time to do N increments on K threads: t2 > t1

Better Code – Algorithmic Solution

```
unsigned long sum = 0;

std::mutex M;

void thread_work() {
    unsigned long local_sum = 0;
    ++local_sum;        // ... as many times as needed ...
    std::lock_guard<std::mutex> L(M);
    sum += local_sum;
}
```

- Do not do any unnecessary work (updating shared data is work)
- Hard work is not always unnecessary: other threads may need to see the updated value more often than once at the end

Lessons On Sharing Memory

- Concurrent programs are more likely to suffer from memory bandwidth constraints
- Sharing data between threads can lead to data races
- Avoiding data races can reduce performance
- Avoiding shared data improves performance!
- There is significant cost to data sharing itself
- Be aware of false sharing and cache line alignment

When Does Performance Matter?

- High performance is a constraint on the implementation
- It comes at a cost: usually a trade-off against other constraints (code complexity, development time)
- A library of high-performing data structures and algorithms can give “free” performance if used right
- Most demanding portions of the code often require individual attention and one-off solutions

So, What Can You Do?

- Performance begins at design stage:
- Design efficient data structures
  - Think about how much data fits in cache
  - Minimize indirect memory accesses
- Evaluate synchronization schemes early
- This violates the rule “do not optimize prematurely”!
  - Do not pessimize prematurely either
  - Do optimize prematurely when fixing problems is hard

Use Measurement Tools

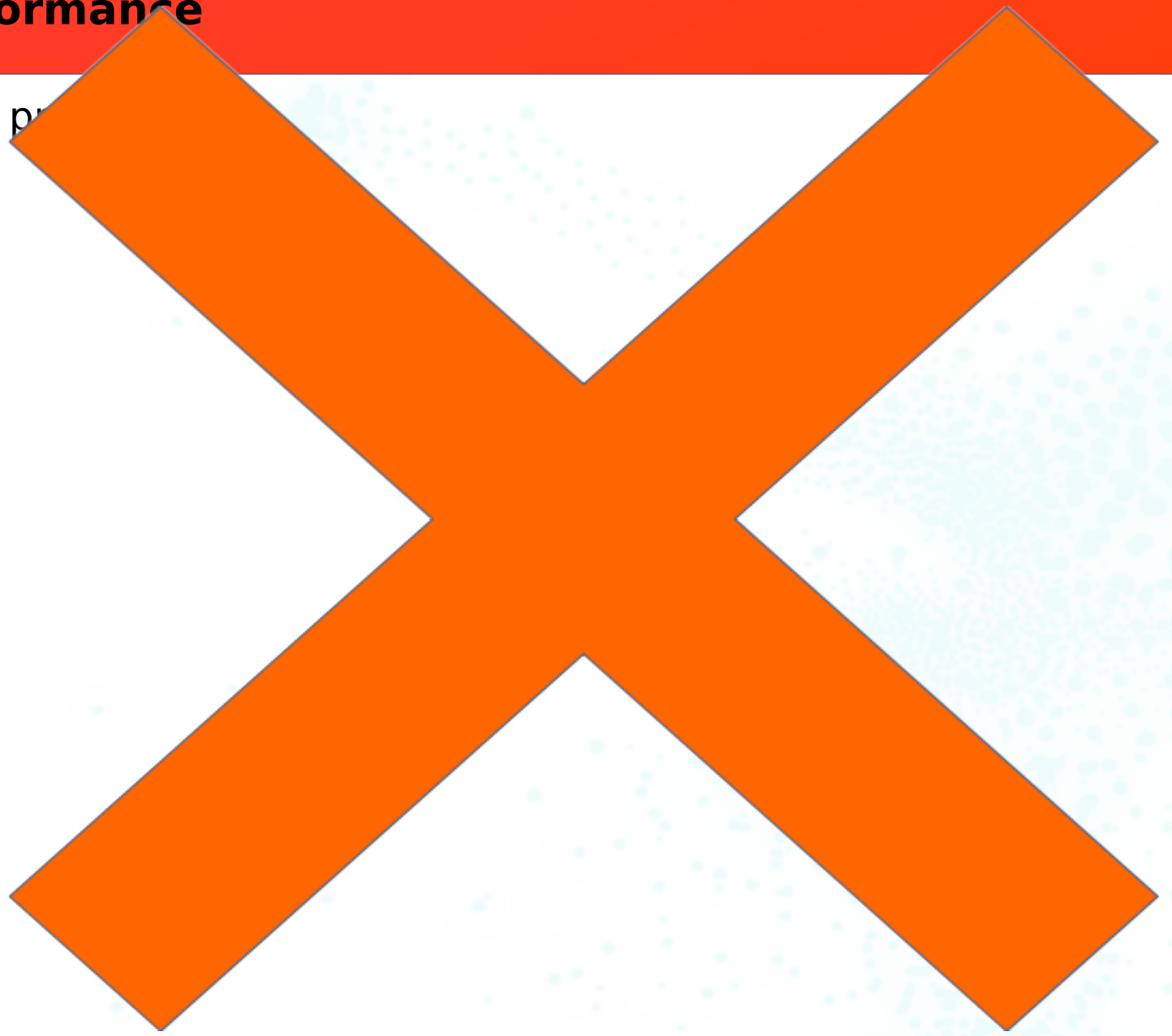
- Even partial information may lead to a critical insight
- Coarse-level timers have low overhead and will guide you to the problem areas
- Use profiling tools effectively: they can measure more than the CPU time (cache misses, pipeline stalls, etc)
- If it appears that the runtime is spread evenly over large part of your code, take a data-centric view instead

Design For Performance

- Do not optimize prematurely

Design For Performance

- Do not optimize pr



Design For Performance

- Do optimize prematurely, but only the important stuff
- When designing API, avoid constraining the implementation
  - Do not expose implementation details, data layout, etc
- Do not provide any more access than necessary
  - If the clients need sequential access to data, do not provide random access “just in case”
- If concurrency or thread-safety may be desired, offer only APIs that can be made thread-safe
  - Bad API for a queue: if (!q.empty()) return q.front();
  - Good API: q.front() returns pair of bool (empty) and value
- Use profilers and timers to find performance bottlenecks

Design For Performance

- Do optimize prematurely, but only the important stuff
- When designing data structures, pay attention to memory locality, cache efficiency, indirections, and memory access patterns
- Concealing implementation behind a well-designed API helps

Design For Performance

- Do optimize prematurely, but only the important stuff
- When designing data structures, pay attention to memory locality, cache efficiency, indirections, and memory access patterns
- Conceal implementation behind a well-designed API
- API may also force implementation
  - std::shared_ptr has API that requires that dereferencing the pointer is double indirection (pointer object to control block to data)
  - Short strings can be stored in the string object itself, avoiding indirection, but then reference-counted strings cannot be supported

Design For Performance

- Do optimize prematurely, but only the important stuff
- When designing data structures, pay attention to memory locality, cache efficiency, indirections, and memory access patterns
- Concealing implementation behind a well-designed API may help
- API may also force implementation
- For concurrent programs, consider data sharing and synchronization, minimize the amount of shared data and the frequency of sharing
- Profilers will usually highlight performance problems but not if the problem is in accessing a data structure that’s used everywhere

Design for Performance

Fedor G Pikus
Technical Fellow, Siemens Digital Software Industries
September 2022 – CppSummit

Restricted | © Siemens 20XX | Author | Department | YYYY-MM-DD

SIEMENS

CPP-Summit 2022

Michael Spertus



Amazon 首席工程师、C++标准委员会资深委员

Michael Spertus是世界级C++技术权威。作为ISO C++标准委员会资深成员，Michael 曾递交过50多项标准提案，是内存管理与性能调优方面公认的技术权威。他目前是Symantec的技术院士与首席科学家，负责云端安全服务。同时在芝加哥大学任教。Michael 自1980开始沉迷软件设计，是IBM PC第一个商用C语言编译器的作者，并曾创办 Geodesic，后被VERITAS收购。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

Mike Spertus

Amazon
University of Chicago

For better performance
Use the cache

THE CONTENT

1. **What is a cache? Why do I care?**
   Do I really need "yet another thing" to worry about?
2. **Learn by doing**
   Explore the question with an example program
3. **Best practices**
   Wow, this is complicated. Can I just have a few simple rules?

01

What is a cache?

Why do I care?

Compute is fast
Memory is slow

- Processors typically run one instruction every clock cycle
    - Billions of computations per second
- A memory access typically takes over one hundred cycles

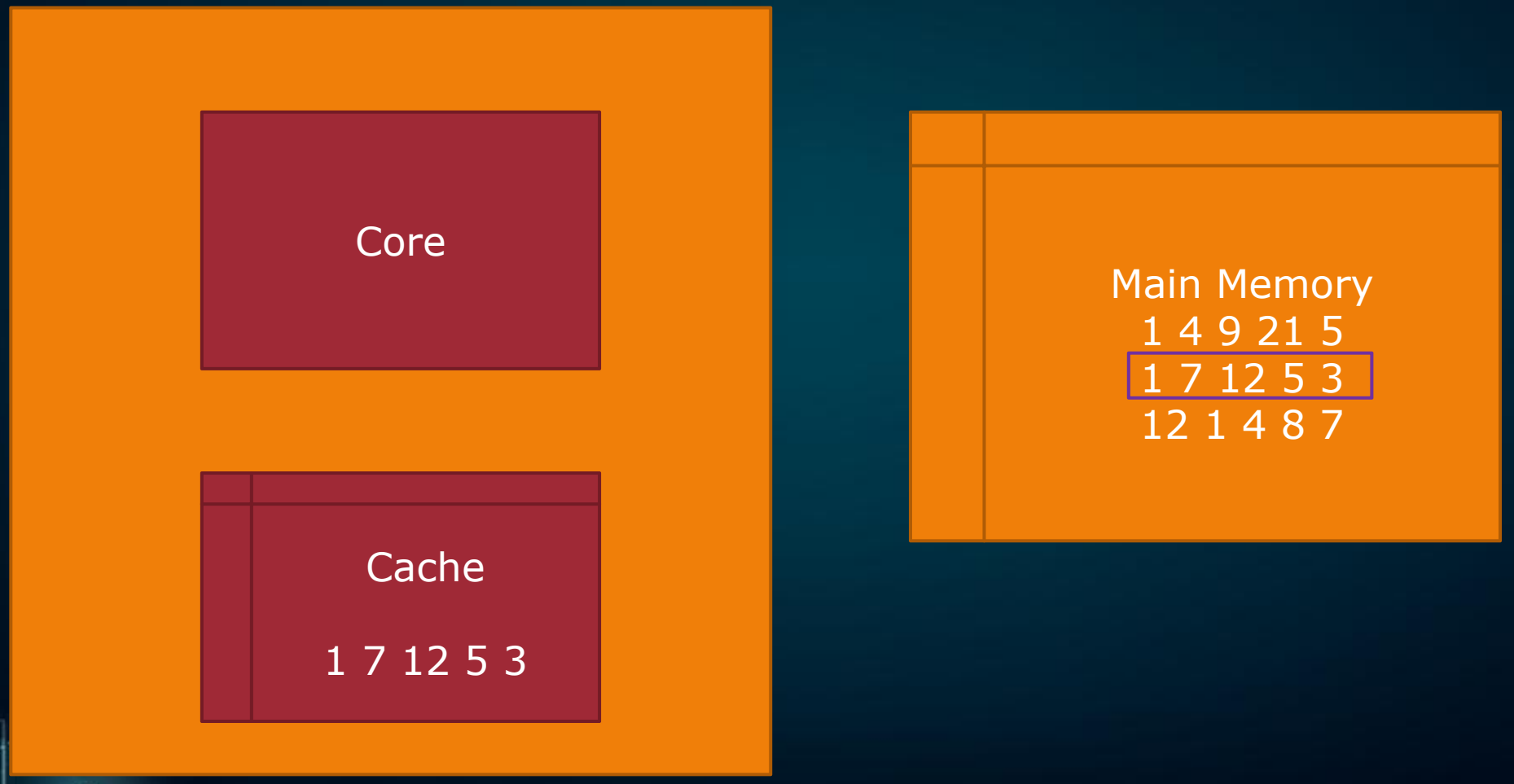
- Since every operation accesses memory (if only to read the instruction)
    - If computers didn't address this issue
    - They would run a hundred times slower than they are capable of processing

100x Slower!

A cache is what bridges this gap

- Processors have some very fast memory on chip
- That can be accessed in as little as one clock cycle
- This memory is used to "cache" a copy of a section of memory for fast access

The basic idea



In reality, there isn't just one cache

- Real CPUs have many caches on chip
  - A fast but small (kilobytes) L1 cache on each core for instructions
  - A fast but small (kilobytes) L1 cache on each core for data
  - A L2 cache on each core that is larger (hundreds of kilobytes) but not quite as fast
  - A L3 cache on each CPU chip that is even larger (megabytes) but a little less fast than L2
    - Still much faster than main memory
- But it doesn't really affect the big picture
  - I'll ignore this

How does a computer decide what memory to cache?

- When the memory that a processor reads is already in the cache, that is called a cache hit
- If it needs to go to slow main memory to read the contents, that is called a cache miss
- The caching algorithm's goal is to maximize this "hit rate"
- Because reading main memory is so slow, even a small improvement to the hit rate can mean a big difference to your program
- A program with a 95% cache hit rate will run *almost twice as fast* as a program with a 90% cache hit rate because the cache misses will be cut in half
- **Note:** In addition to hit rate, there are some optimizations to help the processor quickly access the cached data
    - We will come back to this later when we talk about direct-mapped caches

How do computers maximize the hit rate?

- The challenge is to predict what data we think will be accessed in the future
- This is a hard problem, but some simple algorithms do surprisingly well
- First, whenever we read memory we store it in the cache because that shows the data at that location is of interest to the process and is likely to be read again
- The second techniques is that we cache all of the data near the memory we read because programs often read nearby memory locations
    - Array elements
    - Characters in a string
    - Fields in a struct
    - Consecutive machine instructions

Cache lines

- To bring in the nearby data
- Whenever a computer reads memory
- It doesn't just cache the memory it read
- But a whole region, called a *cache line*, that contains the memory
- The actual size of a cache line varies by processor, but is typically 32-128 bytes
- As we will see, C++ gives us tools for working with cache lines

- By combining cache lines with large caches, programs can often get cache hit rates well over 90%
- Cache misses will still be a big deal, but at least your program won't be 100x slower

Are computers getting faster?

Of course they are!

Are you sure?

Perhaps the biggest secret in computer progress is that computer cores have not gotten any faster in 15 years

- 2005's Pentium 4 HT 571 ran at 3.8GHz, which is better than many high-end CPUs today
- The problem with increasing clock speeds is heat
  - A high end CPU dissipates over 100 watts in about 1 cubic centimeter
  - An incandescent light bulb dissipates 100 watts in about 75 cubic centimeters

Computers are fast
Cores are slow

- Besides memory, the other big bottleneck to performance that we will consider is concurrency
- One of the biggest secrets in computing is that even though the number of transistors on a chip keeps going up (Moore's law), individual cores stopped getting faster about a decade ago
- Let's take a look
- https://semiwiki.com/ip/312695-white-paper-scaling-is-falling/

To be fast, you have to use threads

- Since cores stopped getting faster, if you want your program to run faster than it did a decade ago, you have to use concurrency
- Fortunately, because of Moore's law, even a cellphone can have 4 or more cores, and a server can have over 100
- But just because we have 100 cores doesn't mean that our program will be 100 times faster
- We have to make sure that each core can efficiently do a share of the work
- This is the topic of multithreaded programming
- Let's have a quick reminder

Hello, Threads

```
#include <iostream>
#include <thread>

void hello_threads() {
    std::cout<<"Hello Concurrent World\n";
}

int
main(){
      // Print in a different thread
    std::thread t(hello_threads);
    t.join(); // Wait for that thread to complete
}
```

Threads and Caches

- Unfortunately, while both threads and caches make your programs much faster
- They don't work well together
- If thread 1 is running on core A and is working on an object that it has cached
- Thread 2 running on core B will not see the cached version of the object but instead the stale version in main memory
    - This is an example of a *race condition* or merely a *data race*
- **Note:** Sometimes the hardware will force core A to flush its contents back to main memory so that core B will read the correct data values. This is called a *cache coherency protocol*, but of course writing to main memory and back is very slow, so the performance cost is very high even if a data race is avoided

Race conditions

- Race conditions occur when multiple threads use the same memory where at least one thread is modifying the memory

- If a program has a race condition, it's behavior is



Simple Rule

Do not allow race conditions in your programs

How to prevent race conditions

- C++ provides a variety of tools for writing code without race conditions
- We will illustrate these by working through an example and learning how to write data race-free code that doesn't
- Let's take a look

PLEASE ADD THE TITLE



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

02

Learn by doing

Our example: A counter

- To explore how to write safe, high-performance code, we will implement a counter
- Our counter can be incremented or read by any thread
- While this example seems simple, it is amazing how much can be learned from it
- In effect, it is the worst case for sharing data among threads since all threads are rapidly modifying the counter
    - We have to prevent a data race in the increment
    - And we have to make effective use of the cache even though the count has to be shared by different processors that can't see each other's caches

First attempt: Use a lock

- The most basic way to prevent race conditions is to use a lock or a mutex
- By putting a lock around the counter, only one thread will be able to access it at a time, and the new value will be written back to main memory after each increment
- This is a safe and correct implementation
- Any guesses what the performance will look like?

- Let's take a look at the code

Locked counter

```
class DistributedCounter {
public:
  typedef long long value_type;
  static std::string name() { return "locked counter"; };
private:
  value_type count;
  std::shared_mutex mutable mtx;
public:
  DistributedCounter() : count(0) {}
  void operator++() {
    std::unique_lock lock(mtx);
    ++count;
  }

  value_type get() const {
    std::shared_lock lock(mtx);
    return count;
  }
};
```

Locked counter performance

- Running the counter in a program with one hundred threads each counting to ten million takes 113 seconds
- Even though the computer has 128 cores
- We probably aren't getting much parallelism because they are all taking turns to get the lock each time they increment the counter
- We also are probably getting bad memory performance because the different processors need to update main memory after each increment
- Can we do better?

Second attempt: Locked array

- Instead of just storing a single value in our counter
- We will store an array of values in the counter
- Each thread will use a hash function to decide which value to increment and only lock that value
- This should allow one thread to increment its value at the same time as another thread is incrementing its value

Locked array code

```
class DistributedCounter {
  typedef size_t value_type;
  struct bucket {
    std::shared_mutex sm;
    value_type count;
  };
  static size_t const buckets{ 128 };
  std::vector<bucket> counts{ buckets };
public:
  static std::string name() { return "locked array"; };
  void operator++() {
    size_t index = std::hash<std::thread::id>()(std::this_thread::get_id()) % buckets;
    std::unique_lock ul(counts[index].sm);
    counts[index].count++;
  }
  value_type get() {
    return std::accumulate(counts.begin(), counts.end(), (value_type)0,
        [](auto acc, auto &x) { std::shared_lock sl(x.sm); return acc + x.count; });
  }
};
```

Locked array performance

- Running the same test now takes 13.86 seconds
- Ten times faster!
- Are we set?
- Let's think more about the cache

Third attempt: Locked padded array

- Remember when I mentioned that the cache system caches an entire cache line to take advantage of locality?
- This means that when a processor caches a counter from the array, it will actually cache several adjacent counters that belong to the same cache line
- As soon as it modifies its own counter, the entire cache line enters a modified state, which means that any other counters in the cache line that are being used by other threads need to be resync'ed with main memory in order to avoid cache coherency problems
- This is called a "false sharing" problem where objects that aren't shared in the code still are connected in the cache
- To address this, let's pad the individual counters in the array to 256 bytes, which will certainly be longer than a cache line
- Let's see the code

Locked padded array code

```
class DistributedCounter {
  typedef size_t value_type;

  struct bucket {
    std::shared_mutex sm;
    value_type count;
    char padding[256];
  };

  static size_t const buckets{ 128 };
  std::vector<bucket> counts{ buckets };
public:
  static std::string name() { return "locked padded array"; };
  void operator++() {
    size_t index = std::hash<std::thread::id>()(std::this_thread::get_id()) % buckets;
    std::unique_lock ul(counts[index].sm);
    counts[index].count++;
  }
  value_type get() {
    return std::accumulate(counts.begin(), counts.end(), (value_type)0,
      [](auto acc, auto &x) { std::shared_lock sl(x.sm); return acc + x.count; });
  }
};
```

Locked padded array performance

- 6.31s
- Eliminating the false sharing doubled the performance from the last version!
- Can we do better?

Fourth attempt: Atomic padded array

- Locks are not the only way to avoid race conditions
- C++ also includes atomics
- The idea is that computer hardware often has built in operations like increment that can be done "atomically" (i.e., without a race condition) without need for acquiring a lock in the program code
- Let's modify the previous code to use atomics

Atomic padded array code

```
class DistributedCounter {
  typedef size_t value_type;

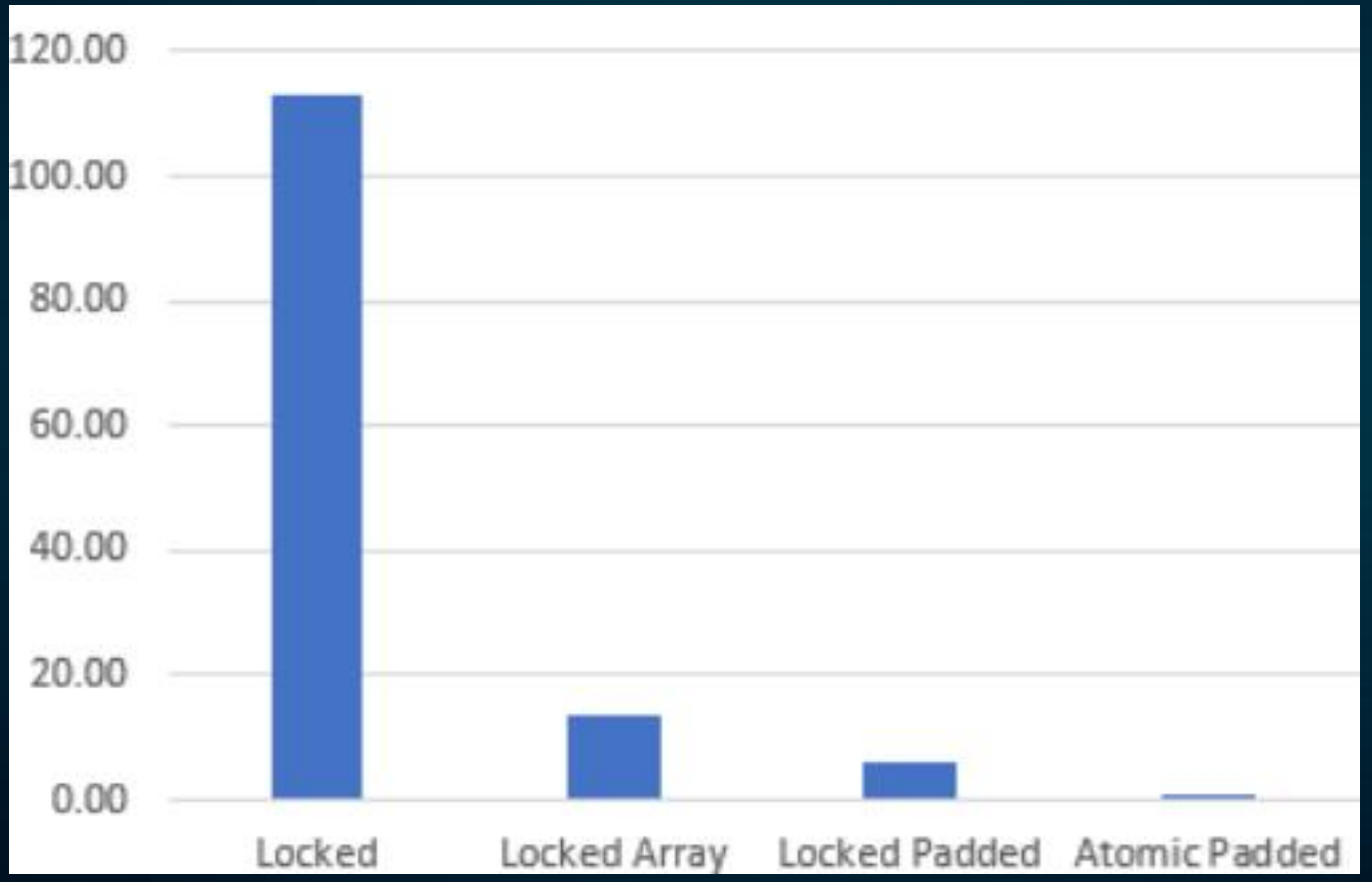
  struct bucket {
    std::atomic<value_type> count;
    char padding[256];
  };

  static size_t const buckets{ 128 };
  std::vector<bucket> counts{ buckets };
public:
  static std::string name() { return "locked padded array"; };
  void operator++() {
    size_t index = std::hash<std::thread::id>()(std::this_thread::get_id()) % buckets;
    counts[index].count++;
  }
  value_type get() {
    return std::accumulate(counts.begin(), counts.end(), (value_type)0,
      [](auto acc, auto &x) {return acc + x.count.load(); });
  }
};
```

Atomic padded array performance

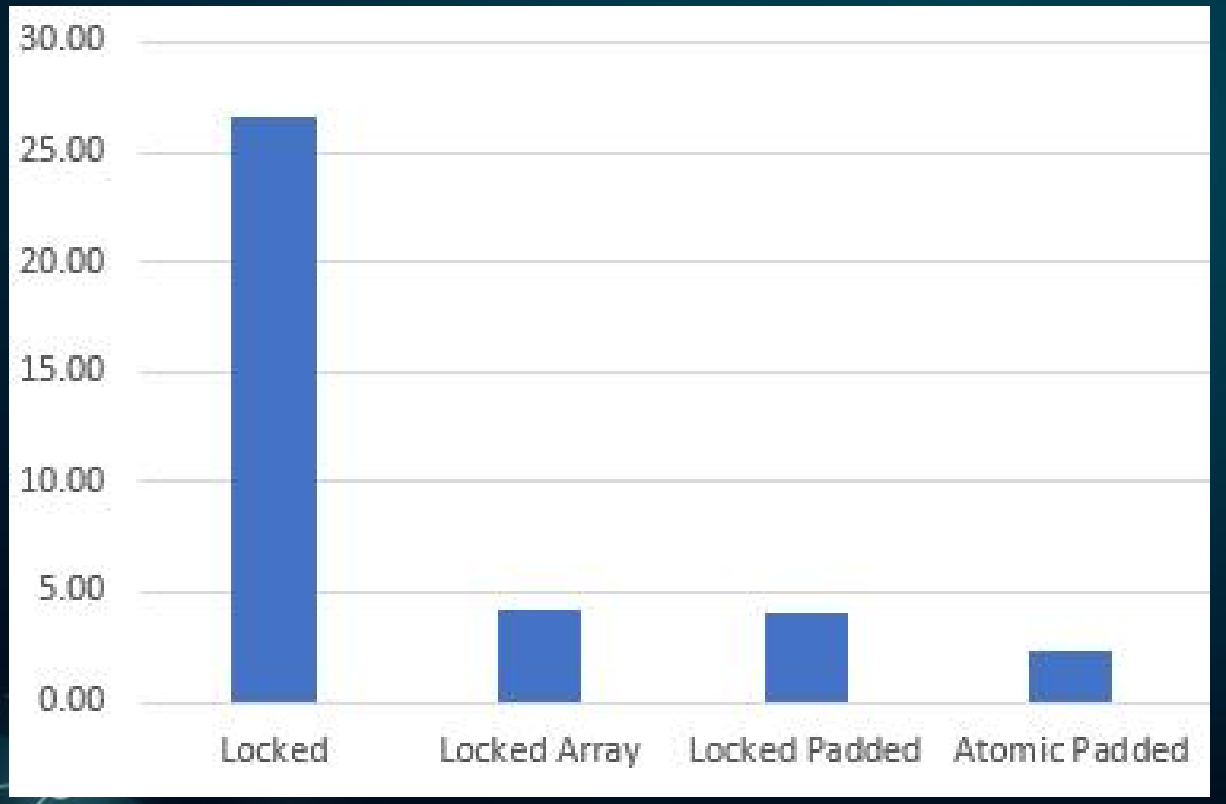
- Now our benchmark takes only 1.15s
- About one hundred times faster than our original version!
- This is a good counter implementation

Let's recap


120.00
100.00
80.00
60.00
40.00
20.00
0.00
Locked
Locked Array
Locked Padded
Atomic Padded

Are we done? Not so fast

- Here's the numbers for the same tests run on my laptop
  - I adjusted the total counts and numbers of threads but the code is the same



Can different machines give different behaviors?

- Yes!
- On my laptop, the padding has no effect!
- Since my laptop only has one CPU socket, the impact of false sharing is much less than on my production server which has two CPU sockets that can only communicate via the motherboard and main memory
- If I had developed my code only on my development laptop, I would have unknowingly paid a 100% performance penalty on my production server!
- This machine dependence makes it hard to develop code you know will perform well everywhere

Another machine dependency

- The ghostscript postscript renderer is used by many millions of people for their document rendering
- In Ben Zorn's PhD thesis, he discovered that supposedly "high performance" custom memory managers had made Ghostscript significantly slower
- To understand why, we need to look more deeply at the cache

Direct-Mapped Caches

Since a processor needs to be able to access cached memory quickly

Finding the cache line that holds it needs to be quick

In a direct-mapped cache, a single memory location can only be mapped to one or two possible cache lines

- See https://courses.cs.washington.edu/courses/cse378/09wi/lectures/lec15.pdf

Not understanding direct-mapped caches had dire consequences for Ghostscript

What went wrong?

- The custom allocator maintained a pool of free-pages in a linked list, with the first word of each free page as a pointer to the next free page.
- This code was developed on a machine without a direct-mapped cache, so it ran fine
- However, on machines with direct-mapped caches, all of the freelist pointers mapped to the same cache line because they were all at the beginning of a page
- This resulted in a cache miss on each step of walking through the freelist, effectively making the cache worthless
- Ghostscript was spending as much as a third of its time in cache misses from walking through the page freelist.
- **Note**: You don’t need to understand this as long as you understand the moral
    - "You can't trust your own benchmarks where caches are concerned"

03

Best practices

Cache-conscious programming

We need best practices

- Since caching effects have a huge impact on program performance
- And they vary wildly from machine to machine
- It isn't enough to just tune the code on your own machine
- Instead, we need a set of consistent best practices that are known to give robust performance across a wide range of architectures
- To this end, Maurice Herlihy and Nir Shavit developed a set of Cache-Conscious programming best practices that can be used to program with confidence

Cache-conscious programming (Adapted from Herlihy&Shavit p. 477)

- Objects or fields that are accessed independently should be aligned and padded so they end up on different cache lines.
- Keep read-only data separate from data that is modified frequently.
- When possible, split an object into thread-local pieces. For example, a counter used for statistics could be split into an array of counters, one per thread, each one residing on a different cache line. While a shared counter would cause invalidation traffic, the split counter allows each thread to update its own replica without causing cache coherence traffic.
- If a lock protects data that is frequently modified, then keep the lock and the data on distinct cache lines, so that threads trying to acquire the lock do not interfere with the lock-holder's access to the data.
- If a lock protects data that is frequently uncontended, then try to keep the lock and the data on the same cache lines, so that acquiring the lock will also load some of the data into the cache.
- If a class or struct contains a large chunk of data whose size is divisible by a high power of two, consider separating it out of the class and holding it with an `unique_ptr` to avoid the Ghostscript problem from above
- Use a profiling tool like VTune to identify where your cache bottlenecks are

Portable cache-conscious programming

The rules for cache-conscious programming, had a lot of phrases like “put on the same/different cache line"

But how do we know whether two objects are on the same or different cache lines?

How do we do that in C++?

We could make an educated guess, like we did with adding 256 bytes to our padded counter

- However, such “magic numbers” are always brittle

C++17 introduced a portable approach

- If you keep objects further than `std::hardware_destructive_interference_size`, they “should” (i.e., implementation-defined) end up on different cache lines
- If you keep objects closer than `std::hardware_constructive_interference_size`, they “should” end up on the same cache line

Thank you

CPP-Summit 2022



Amir Kirsh

Incredibuild开发布道师

Amir Kirsh既是一名讲师也是一名从业者。在特拉维夫大学(Tel-Aviv-yaffo)和特拉维夫大学(Tel-Aviv University)任教，同时在Incredibuild担任开发布道师，之前在Comverse担任首席开发者，主要研究C++。Amir也是特拉维夫CoreCpp会议和CoreCpp meetup小组的组织者之一，以色列ISO C++国家机构的成员。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

Amir Kirsh

Lecturer, Dev Advocate

Implementing max

a walk through API design, dangling references and C++20 constraints

About me

Lecturer
Academic College of Tel-Aviv-Yaffo
Tel-Aviv University

Developer Advocate at

INCREDIBUILD

Member of the Israeli ISO C++ NB

Co-Organizer of the **CoreCpp** conference and meetup group



INCREDIBUILD

Suffering from slow builds?

It’s not just waste of time

It affects your dev cycles
and productivity

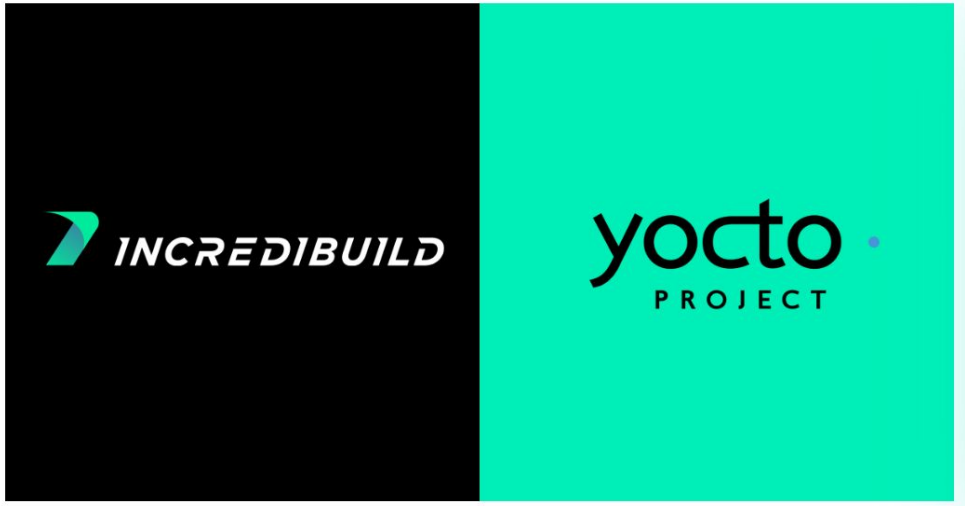


We also accelerate Yocto builds!

Our recent talks at Yocto Project Summit:

https://bit.ly/YPS-2022_IB_bitbake

https://bit.ly/YPS-2022_IB_Cache



Incredibuild + Yocto:

https://www.incredibuild.com/blog/announcing-incredibuild-support-for-yocto
https://www.incredibuild.com/lp/yocto

Implementing max

(is it that difficult?)

Things to consider

- Efficiency (avoid redundant copies, allow move if possible)
- Avoid dangling references
- Support different types (that are comparable and share a common type, e.g. int and double)

Not in the scope for today:

- Allow changing the returned value if possible
- Support returning a ref for different lvalue-ref types – if one inherits from the other
- Work on variadic packs

The initial challenge

```
// case 1 - lvalues
std::string s1 = "hello", s2 = "hi";
const std::string& max1 = my_max(s1, s2);

// case 2 - rvalues - ?
const std::string& max2 = my_max("hello"s, "world"s);
```

Let’s try things together…

Live Coding!

Reference

Main resource:

C++ passing by const ref vs universal ref - Stack Overflow

Additional related resources:

C++11 rvalues and move semantics confusion (return statement) - Stack Overflow

Implementing variadic min / max functions - c++

Reflections on C++ Variadic Templated Versions of std::min() and std::max() - Stack Overflow

Variadic template max function trouble - Stack Overflow

Old one, and related, Dr. Dobbs, Andrei Alexandrescu, April 01, 2001: https://erdani.org/publications/cuj-04-2001.php.html

Thank you



熊谱翔

RT-Thread 创始人

熊谱翔 RT-Thread创始人，上海睿赛德电子科技有限公司CEO。2000年毕业于重庆邮电学院，先后任职上海贝尔阿尔卡特、宇梦通信、Marvell，拥有丰富的嵌入式实时系统和3G/4G手机基带系统软件开发经验。于2006年创立了RT-Thread开源操作系统，并以开源社区的形式，带领中国国内优秀的工程师，融入实时面向对象设计开发了RT-Thread自主操作系统内核和微内核架构操作系统RT-Thread Smart，大量周边成熟、稳定软件组件，如文件系统，图形用户界面等。



底层操作系统上的C++

熊谱翔

个人简单介绍

- Bernard Xiong | 午夜熊
  - 近50岁、迈向知天命码农
  - C/PHP/C++/Java/Python…
  - 嵌入式系统
    - 图形系统
  - 实时系统
  - 2006年写了一个嵌入式操作系统：RT-Thread，并开源出来



目录

OS

嵌入式操作系统那些事

- 操作系统，C vs C++



系统底层如何支持到C++

- 基础操作
- 异常
- C++11



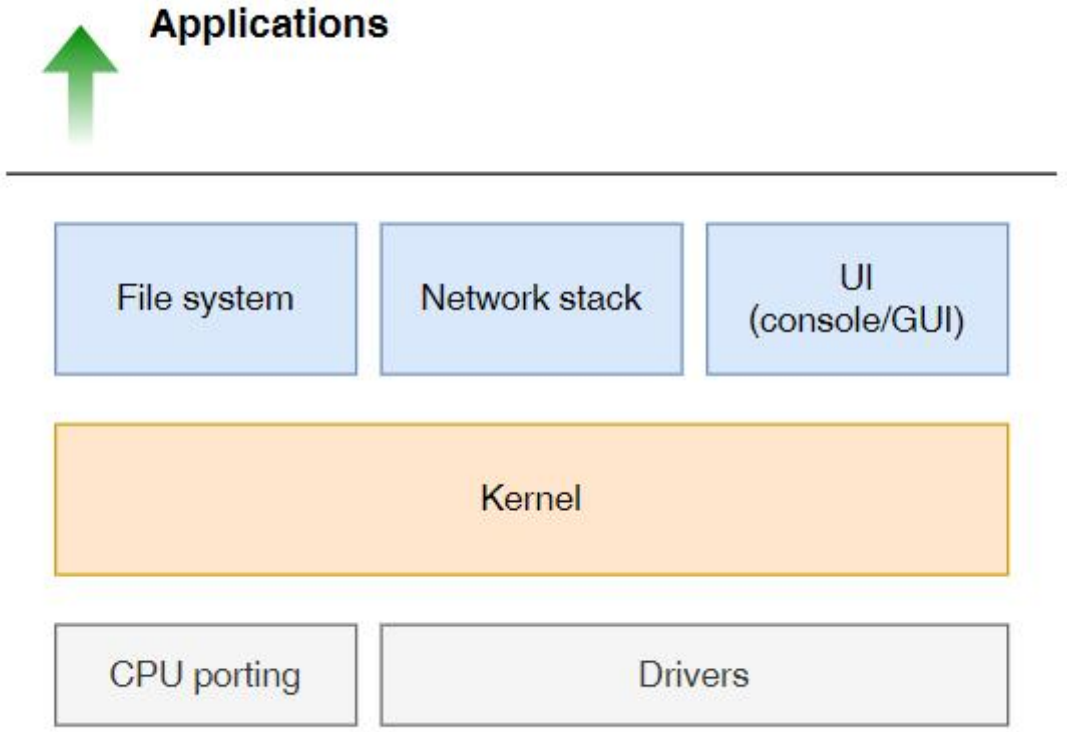
UI上的C++

- 现代场合下的UI 和 C++

操作系统

- 经典操作系统范畴（狭义操作系统）

  核心：为上层应用服务



嵌入式操作系统

- RT-Thread嵌入式操作系统
  - 2006年诞生

  - 完整的操作系统内核，面向嵌入式设备的嵌入式实时操作系统

    原则：社区，开放，中立

10+亿装机量

- 工业
- 电力
- 航天
- 汽车
- 消费类电子
- 物联网终端

10+万开发者

100所高校
开设RT-Thread相关课程
或实验室

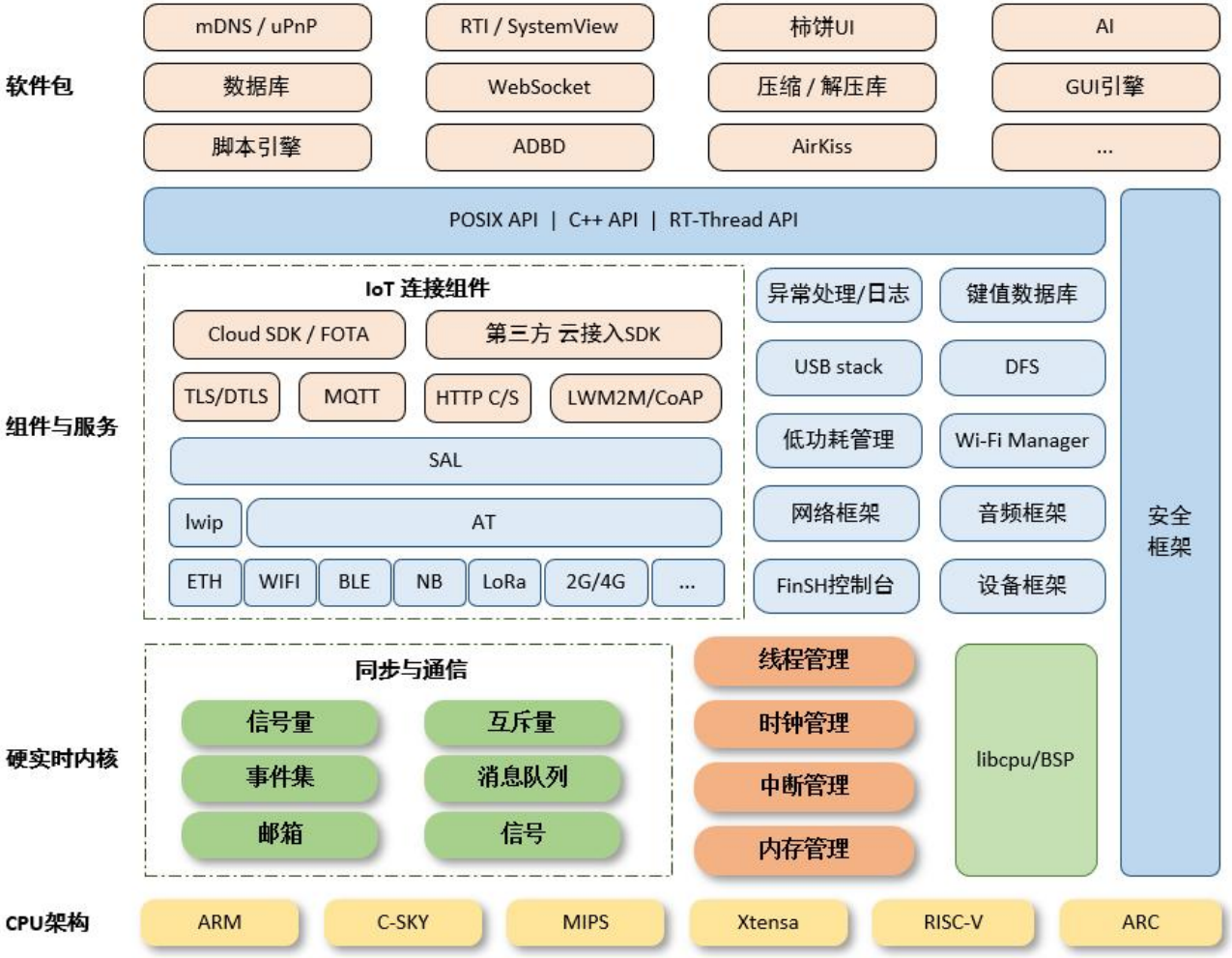


《嵌入式实时操作系统:RT-Thread设计与实现》
《RT-Thread设备驱动开发指南》（新）
机械工业出版社出版

近100家
合作芯片厂商

- 覆盖主流芯片架构
- 近千款芯片

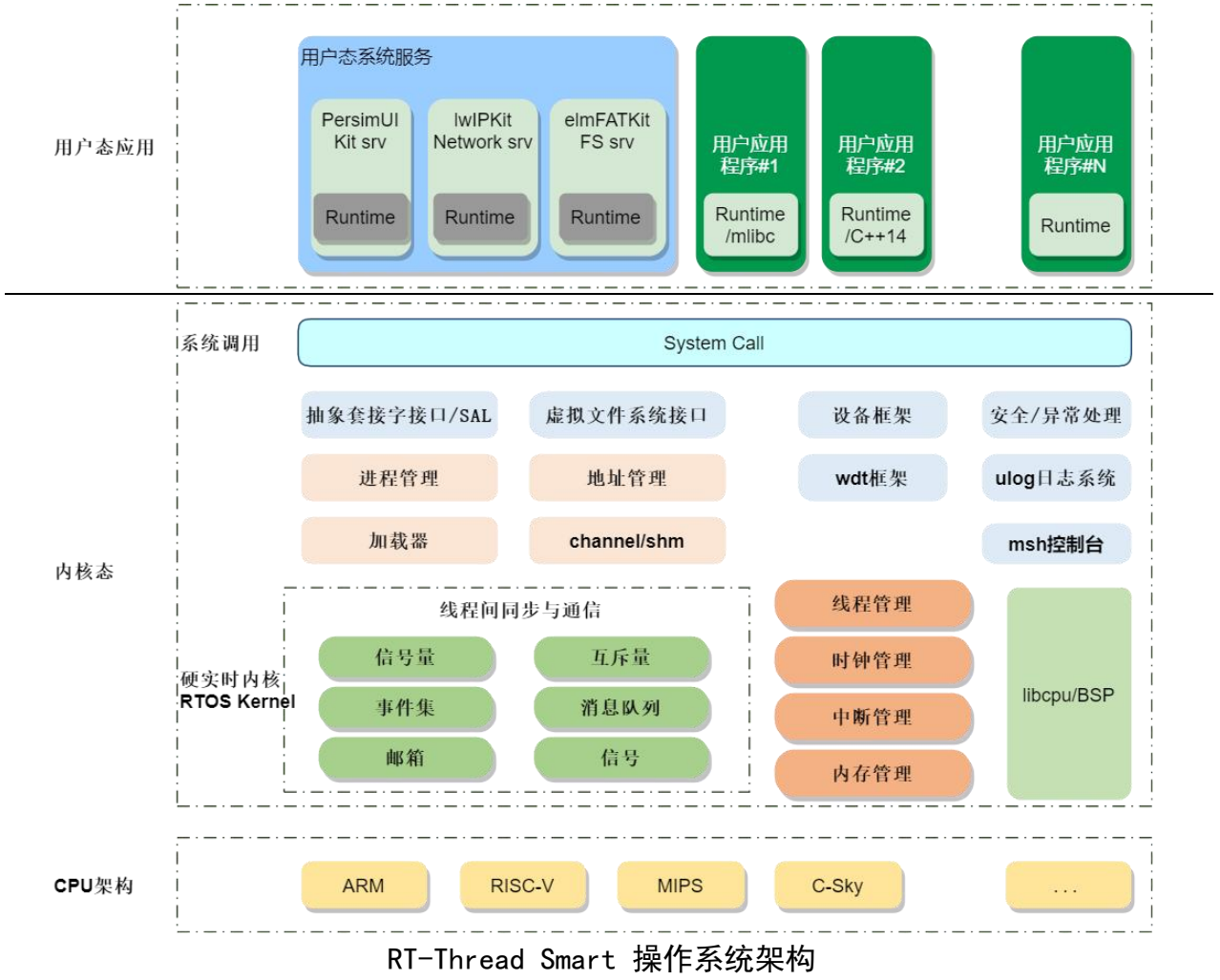
RT-Thread操作系统的架构



分层的架构，在应用层提供系列的支撑，包括对不同语言的支持。

- 系统：运行在线性内存地址空间，采用C语言和汇编编程
- 柿饼UI：C++
- 脚本引擎：JavaScript，MicroPython，PikaScript，Lua等

内核态和用户态分离的操作系统



RT-Thread Smart 操作系统架构

系统整体分成内核态和用户态

- 内核是系统的核心，提供服务；
- 用户态运行应用程序：
  - 用户态具备完整的地址空间，相互隔离进行保护；
  - 需要硬件MMU支持；

⚠ RT-Thread Smart是一个混合内核的操作系统，一些系统服务可以放置于用户态，内核小型化；应用和系统服务可通过IPC消息进行交互。

底层操作系统

- C 或 C++
  - 为什么不使用C++来编写操作系统？
    - C++名字粉碎化，汇编中调用C++并不方便（extern “C”，当成C时才可以）；
    - C++隐藏的机制
      - **对象**，虚表，拷贝构造
      - 模板，多态，异常
    - 对人员能力的要求会变高；
  - C语言：简单纯粹
    - 函数：入参，返回值
    - 汇编级

底层操作系统

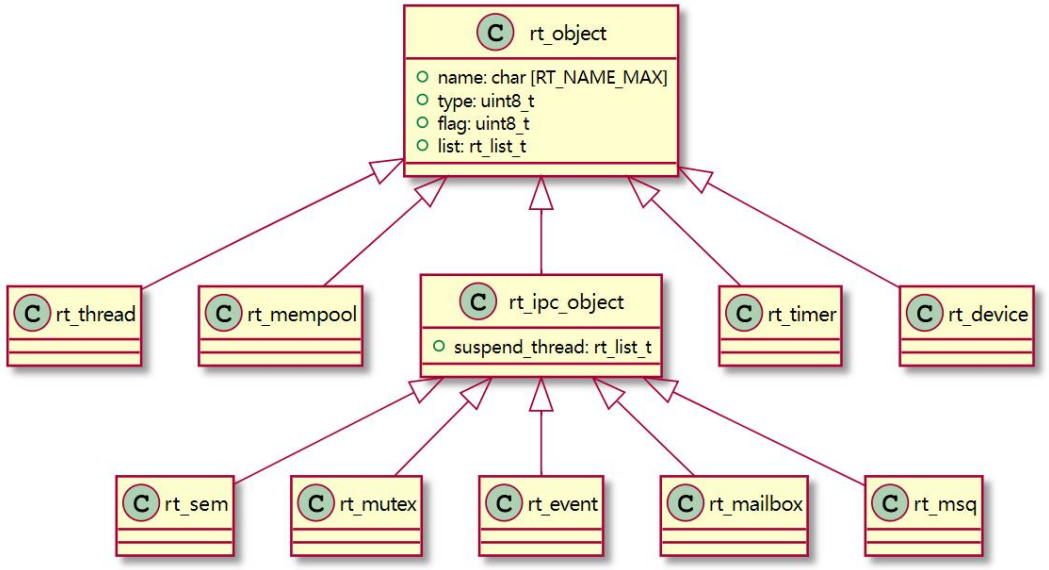
- 面向过程和面向对象
  - 过程 – 看到的是一个个的**动作**
    - 做了什么
    - 逻辑关系
    - 循环关系
  - 对象 – 看到的是一个个的**实体**
    - 模块化编程
    - 模块的属性
    - 模块的方法

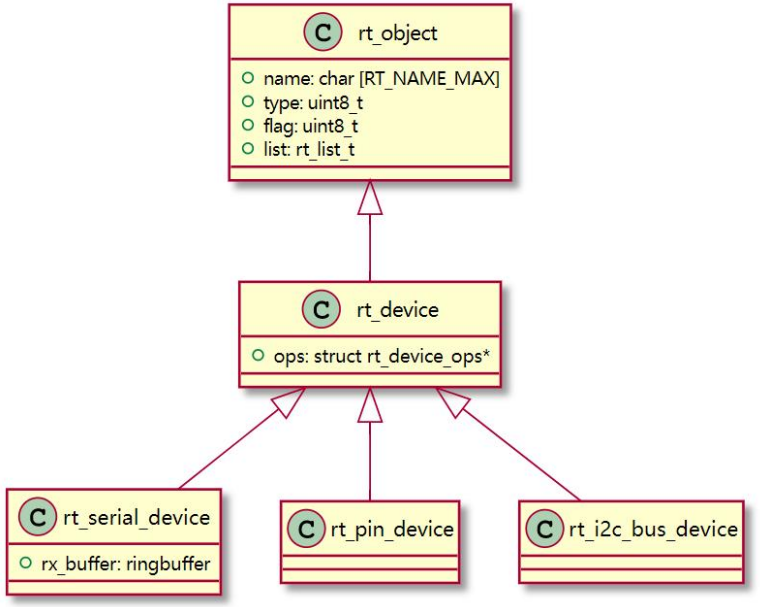
底层操作系统

- 面向对象设计
  - C语言中依然可以应用面向对象设计 （过程 → 模块）
    - 划分成一个个类（结构体），并匹配方法
    - 嵌套结构体及类型转换的继承；
    - 结构体，静态函数方式的封装；
    - 函数指针形式的多态；
- RT-Thread中的设计原则
  - 简单 - 用简单、易懂的方式进行描述，也包括命名方式
  - 清晰 – 需要有条理性
  - 模块化 - 相互之间接口清晰，依赖清晰，松耦合
  - **面向对象**原则……

RT-Thread操作系统的设计方法

- 内核对象 和 设备对象





- 对象可复用、可遍历

底层的基础C++支持

- new/delete操作

```
void *operator new(size_t size)
{
    return rt_malloc(size);
}

void *operator new[](size_t size)
{
    return rt_malloc(size);
}
```

```
void operator delete(void *ptr)
{
    rt_free(ptr);
}

void operator delete[](void *ptr)
{
    return rt_free(ptr);
}
```

底层的基础C++支持

- 对象构造和析构
  - ctors → __ctors_start__/__ctors_end__
  - .init_array

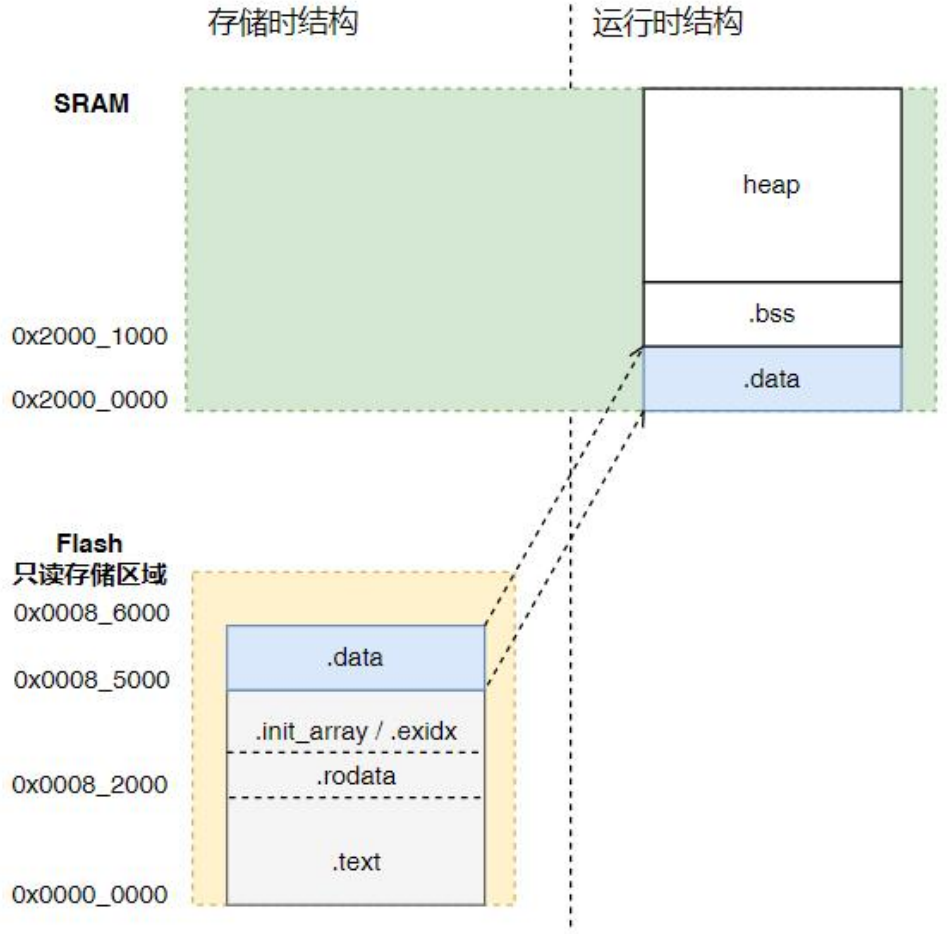
```
RT_WEAK int cplusplus_system_init(void)
{
    typedef void(*pfunc)();
    extern pfunc __ctors_start__[];
    extern pfunc __ctors_end__[];
    pfunc *p;

    for (p = __ctors_start__; p < __ctors_end__; p++)
        (*p)();

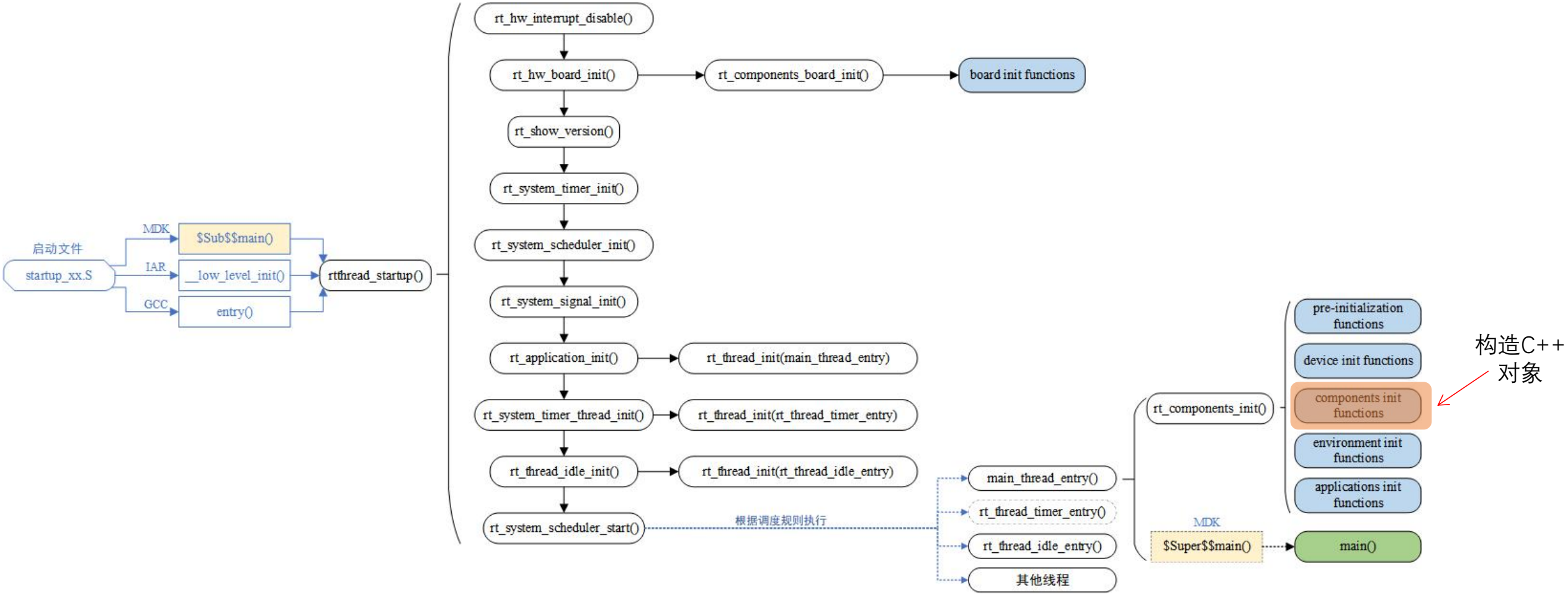
    return 0;
}
INIT_COMPONENT_EXPORT(cplusplus_system_init);
```

嵌入式操作系统内存布局

- 存储时结构、运行时结构



系统启动和C++对象构造


启动文件
startup_xx.S
MDK
IAR
GCC
$Sub$$main()
__low_level_init()
entry()
rtthread_startup()
rt_hw_interrupt_disable()
rt_hw_board_init()
rt_components_board_init()
board init functions
rt_show_version()
rt_system_timer_init()
rt_system_scheduler_init()
rt_system_signal_init()
rt_application_init()
rt_thread_init(main_thread_entry)
rt_system_timer_thread_init()
rt_thread_init(rt_thread_timer_entry)
rt_thread_idle_init()
rt_thread_init(rt_thread_idle_entry)
rt_system_scheduler_start()
根据调度规则执行
main_thread_entry()
rt_thread_timer_entry()
rt_thread_idle_entry()
其他线程
rt_components_init()
pre-initialization functions
device init functions
components init functions
environment init functions
applications init functions
MDK
$Super$$main()
main()
构造C++对象

异常

- 工具链中C++ ABI使用__cxa等函数进行处理：
  - __cxa_allocate_exception
  - __cxa_throw
  - __cxa_begin_catch
  - 等

- 编译参数中使用 -fno-exceptions/-fexceptions关闭或开启异常支持；
- 链接脚本中加入

```
__exidx_start = .;
.ARM.exidx   : { *(.ARM.exidx* .gnu.linkonce.armexidx.*) }
__exidx_end = .;
```

异常过程

```
int throw_exceptions(int argc, char** argv)
{
    try {
        throw_A(100);
    }
    catch(int value)
    {
        rt_kprintf("throw exception: %d\n", value);
    }

    return 0;
}
```

```
int throw_A(int value)
{
    throw value;
    return 0;
}
```

```
00000024 <throw_exceptions>:
  24: e92d4010  push  {r4, lr}
  28: e3a00064  mov r0, #100  ; 0x64
  2c: ebffffe  bl  0 <throw_A>
  30: e3510001  cmp r1, #1
  34: 1a000008  bne 5c <throw_exceptions+0x38>
  38: ebffffe  bl  0 <__cxa_begin_catch>
  3c: e5901000  ldr r1, [r0]
  40: e3000000  movw  r0, #0
  44: e3400000  movt  r0, #0
  48: ebffffe  bl  0 <rt_kprintf>
  4c: ebffffe  bl  0 <__cxa_end_catch>
  50: e3a00000  mov r0, #0
  54: e8bd8010  pop {r4, pc}
  58: ebffffe  bl  0 <__cxa_end_catch>
  5c: ebffffe  bl  0 <__cxa_end_cleanup>
```

```
00000000 <throw_A>:
   0: e92d4010  push  {r4, lr}
   4: e1a04000  mov r4, r0
   8: e3a00004  mov r0, #4
   c: ebffffe  bl  0 <__cxa_allocate_exception>
  10: e3001000  movw  r1, #0
  14: e3401000  movt  r1, #0
  18: e3a02000  mov r2, #0
  1c: e5804000  str r4, [r0]
  20: ebffffe  bl  0 <__cxa_throw>
```

- 过程
  - __cxa_allocate_exception分配exception object和header
  - 转入__cxa_throw抛出异常
    - 执行search过程，匹配catch的类型；执行cleanup过程，跳转到catch后的处理部分；
    - 调用__cxa_begin_catch；执行catch代码；调用__cxa_end_catch。
    - 否则调用std::terminate()；

支持C++11线程库

- Threads - 多线程管理
  - class id;
  - void join();
  - void detach();
  - 需要对接到native thread，特别是id，id.__cpp_thread_t
  - typedef rt_cpp_thread_t native_handle_type;
- Atomic - 原子操作
- Mutual exclusion
  - std::mutex and std::timed_mutex
  - typedef rt_cpp_mutex_t __native_type;
  - void lock();
  - bool try_lock() noexcept;
  - void unlock() noexcept;
- Condition variables
  - typedef rt_cpp_cond_var *native_handle_type;
- Futures
- 其他
  - class tick_clock;
  - class real_time_clock;

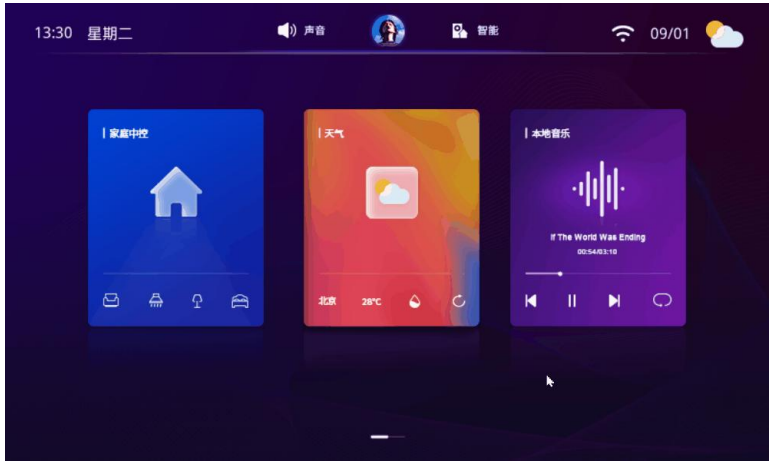
```
#define rt_cpp_thread_t pthread_t
#define rt_cpp_mutex_t  pthread_mutex_t
#define rt_cpp_cond_var pthread_cond_t
```



```
#define rt_cpp_thread_t rt_thread_t
#define rt_cpp_mutex_t  rt_mutex_t
#define rt_cpp_cond_var cond_var_t
```

C++ on PersimUI

- Persimmon UI 是一款现代风格，类Android的轻量级图形用户界面库：
  - 轻量级的GUI，运行在MCU芯片平台
    - 100+MHz ~ 芯片主频，包括ARM Cortex-M/A，RISCV
    - 数百kB ~ 数十MB RAM
  - 底层GUI采用C++实现，具备丰富的控件类；



PersimUI最初的演化：RTGUI

- 极小资源占用的GUI
  - 10 - 20kB级别RAM资源占用的GUI
  - 以C代码方式实现的面向对象编程

```
#define RTGUI_TYPE(type)            (_rtgui_##type##_get_type())

#define DECLARE_CLASS_TYPE(type)     \
    const rtgui_type_t *_rtgui_##type##_get_type(void); \
    extern const struct rtgui_type _rtgui_##type

#define DEFINE_CLASS_TYPE(type, name, parent, constructor, destructor, size) \
    const struct rtgui_type _rtgui_##type = { \
    name, \
    parent, \
    RTGUI_CONSTRUCTOR(constructor), \
    RTGUI_DESTRUCTOR(destructor), \
    size }; \
    const rtgui_type_t *_rtgui_##type##_get_type(void) { return &_rtgui_##type; }
```

基础的对象系统

头文件中进行声明

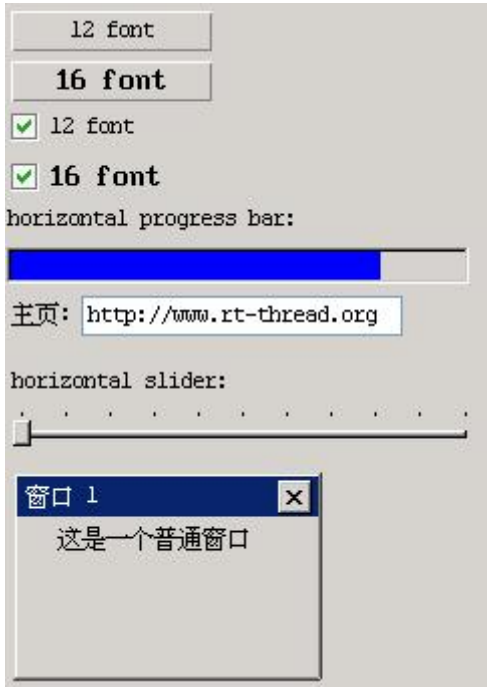
```
DECLARE_CLASS_TYPE(widget);

/** Gets the type of a widget */
#define RTGUI_WIDGET_TYPE       (RTGUI_TYPE(widget))
/** Casts the object to a rtgui_widget */
#define RTGUI_WIDGET(obj)       (RTGUI_OBJECT_CAST((obj), RTGUI_WIDGET_TYPE, rtgui_widget_t))
/** Check if the object is a rtgui_widget */
#define RTGUI_IS_WIDGET(obj)    (RTGUI_OBJECT_CHECK_TYPE((obj), RTGUI_WIDGET_TYPE))

/*
 * the base widget object
 */
struct rtgui_widget
{
    /* inherit from rtgui_object */
    struct rtgui_object object;
    ...
};
```

C文件中进行实现

```
DEFINE_CLASS_TYPE(widget, "widget",
                  RTGUI_PARENT_TYPE(object),
                  _rtgui_widget_constructor,
                  _rtgui_widget_destructor,
                  sizeof(struct rtgui_widget));
```



UI控件的烦恼

- PersimUI，现代化的UI
  - 底层使用C++语言进行编程（时间点2013年）
  - 丰富的控件库

  - C++中的自定义动作
    - C语言中可以灵活的使用函数指针，指针是万能的！

    - 以模板的方式实现
      - Signal<>, Signal<typename Tp1>, Signal<typename Tp1, typename Tp2>
      - Slot0, Slot1, Slot2

      - 分别支持0/1/2参数的signal / slot

信号发送及信号槽接收

- 信号成员及发送端

```
class ValueButton : public Button
{
public:
    ValueButton(const char* text);

    virtual rt_bool_t onClickEvent(struct rtgui_event* event)
    {
        clicked(10, text); /* 调用信号槽函数，传递10，text参数 */
    }

    Signal <int, char*> clicked;
};
```

- 信号槽接收端

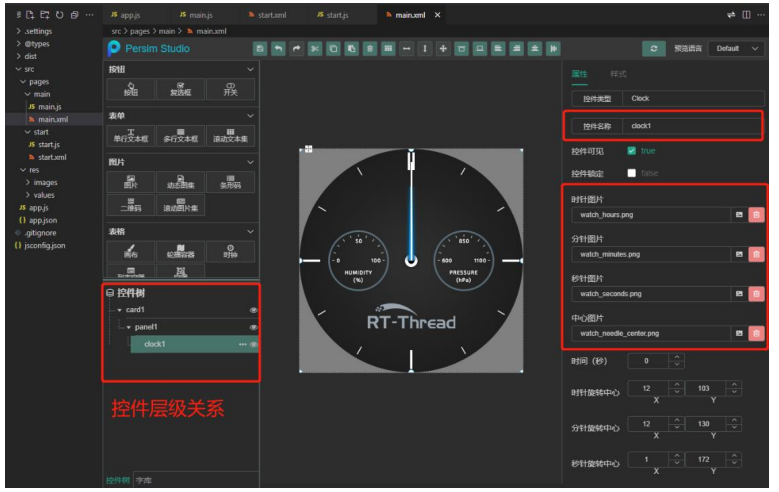
```
class A
{
public:
    A()
    {
        ValueButton *btn = new ValueButton("Hello");
        btn->clicked.connect(this, &A::onButton);
    }

    void onButton(int value, char *str)
    {
        printf("clicked: value = %d, str=%s\n", value, str);
    }
};
```

PersimUI 全貌

- 应用于各类带屏的智能终端，如穿戴手表，智能家居；

- 完整工具支持：
  - 拖拽式 UI 设计器，可快速完成界面布局；
  - 模拟器环境运行，无需真实硬件，即可进行应用开发；
- 丰富的控件。具备动画引擎，很方便进行组合，实现各种动画效果及转场动画；
- 高效的图形渲染，包括半透明、 Alpha 混合、抗锯齿和浮雕字体等特效；
- 可进行脚本JavaScript编程，屏幕自适应，具备系列的第三方轻应用程序；
- 可适配嵌入式 GPU ，具备 GUI 硬件加速能力
  - 流畅的 2.5D/3D 特效





总结



底层系统软件对C++的支持

- 内存操作，构造函数，异常
- C++并发操作，对接到系统多任务



面向对象设计

- 可以非常好的描述一个系统：封装、继承、多态
- 底层系统软件需要面向对象设计



RT-Thread操作系统

- 小而美的操作系统

https://github.com/RT-Thread/rt-thread



谢谢

CPP-Summit 2022



陈硕

Google高级工程师，muduo开源库作者

2012 年加入 Google 美国硅谷总部，先后供职于广告基础架构和数据仓库等团队。在大规模分布式系统的可靠性工程、性能剖析与优化、排解线上故障等领域有独到的经验与见解。在此之前曾在摩根士丹利 IT 部门工作 5 年，从事实时外汇交易系统开发。编写了知名开源 C++ 网络库 muduo，著有《Linux 多线程服务端编程》，参与翻译了《代码大全（第 2 版）》和《C++ 编程规范（繁体版）》。

主办方：Boolan 高端IT咨询与教育平台

C++ 性能、工具、库

陈硕
giantchen@gmail.com
http://chenshuo.com

《2022 全球 C++ 及系统软件技术大会》

Disclaimer

The opinions expressed in this talk are my own and not necessarily those of my employer.

All data presented here are from published sources, with links when available.

"... almost no C++ programmers are *totally* sure how C++ works, and the ones which are entirely confident seem to be mostly the ones who are also demonstrably wrong about it." --- a tweet from @mcclure111 on Dec 26, 2018

陈硕



2006



2022

标签“人设”：C++、网络编程、Linux 服务器

普通程序员，做了几点微小的工作：

- 码了一个库 muduo 2010
- 写了一本书《Linux 多线程服务端编程》 2012 24 刷 33k 册
- 录了一门课《网络编程实战》 2014~2016

实际：尝过一点猪肉，见过很多猪跑。

今天，借花献佛，分享过去五年的所见所闻。

信息密度估计比较大，大家多包涵。

2005 上海 C++ 技术大会



2009 上海 C++ 技术大会



Agenda / 议程 11:00 ~ 11:50

《2022 全球 C++ 及系统软件技术大会》

性能 / performance　　25 min

- 链接较多，可从 chenshuo.com 下载 PDF 版讲义。

C++ 代码工具、库　　15 min

- Clang as a library
- Abseil, new TCMalloc

Q/A　　10 min

每一张 PPT 右下角有页码，可以记下来以便提问。

CPU performance trend / CPU 性能趋势

End of Moore's law.

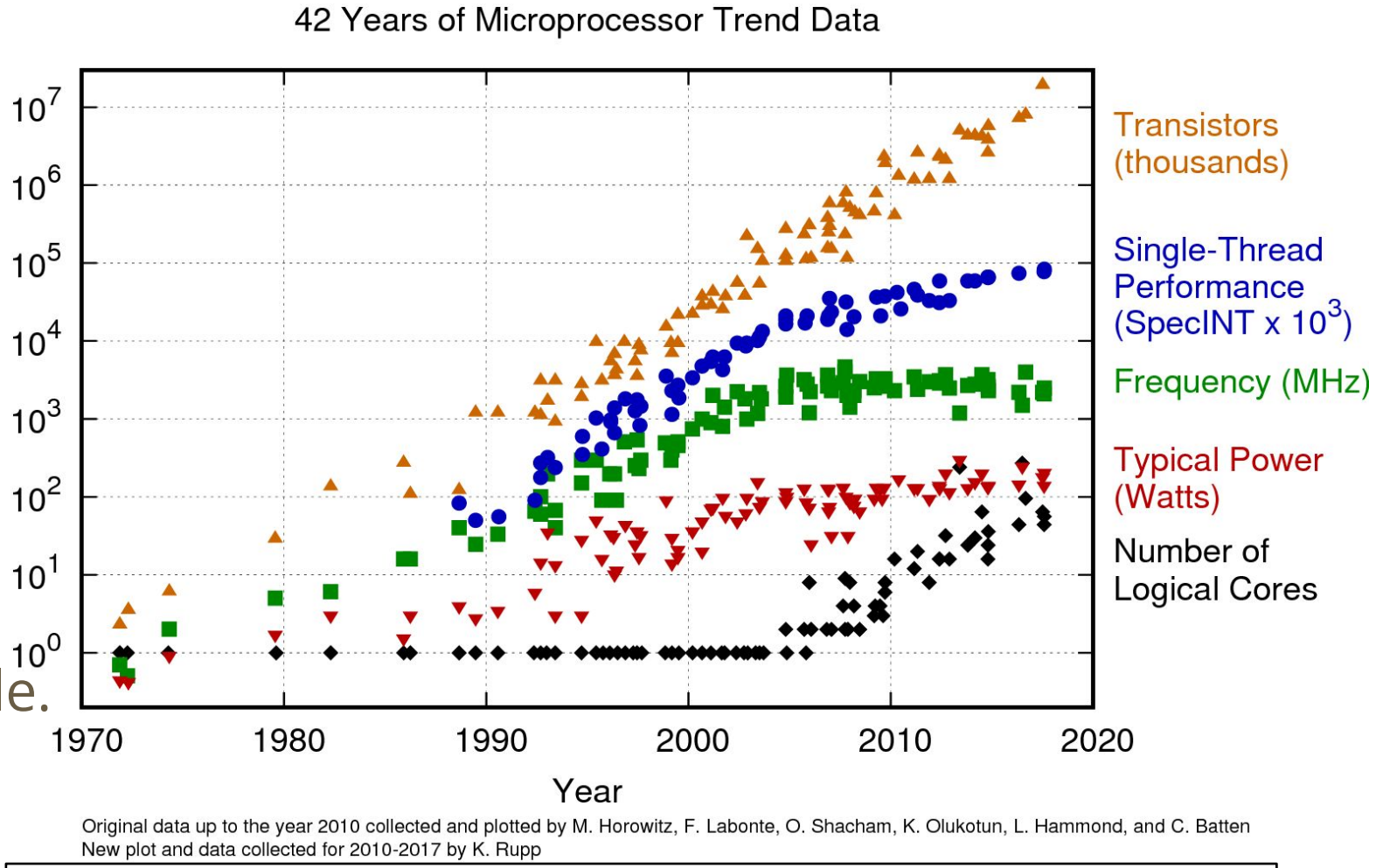
The free lunch is over.

– Herb Sutter, 2005

Single-thread IPC saturation

Instead of getting faster cores, you get (many) more cores.

128c256t per server is feasible.

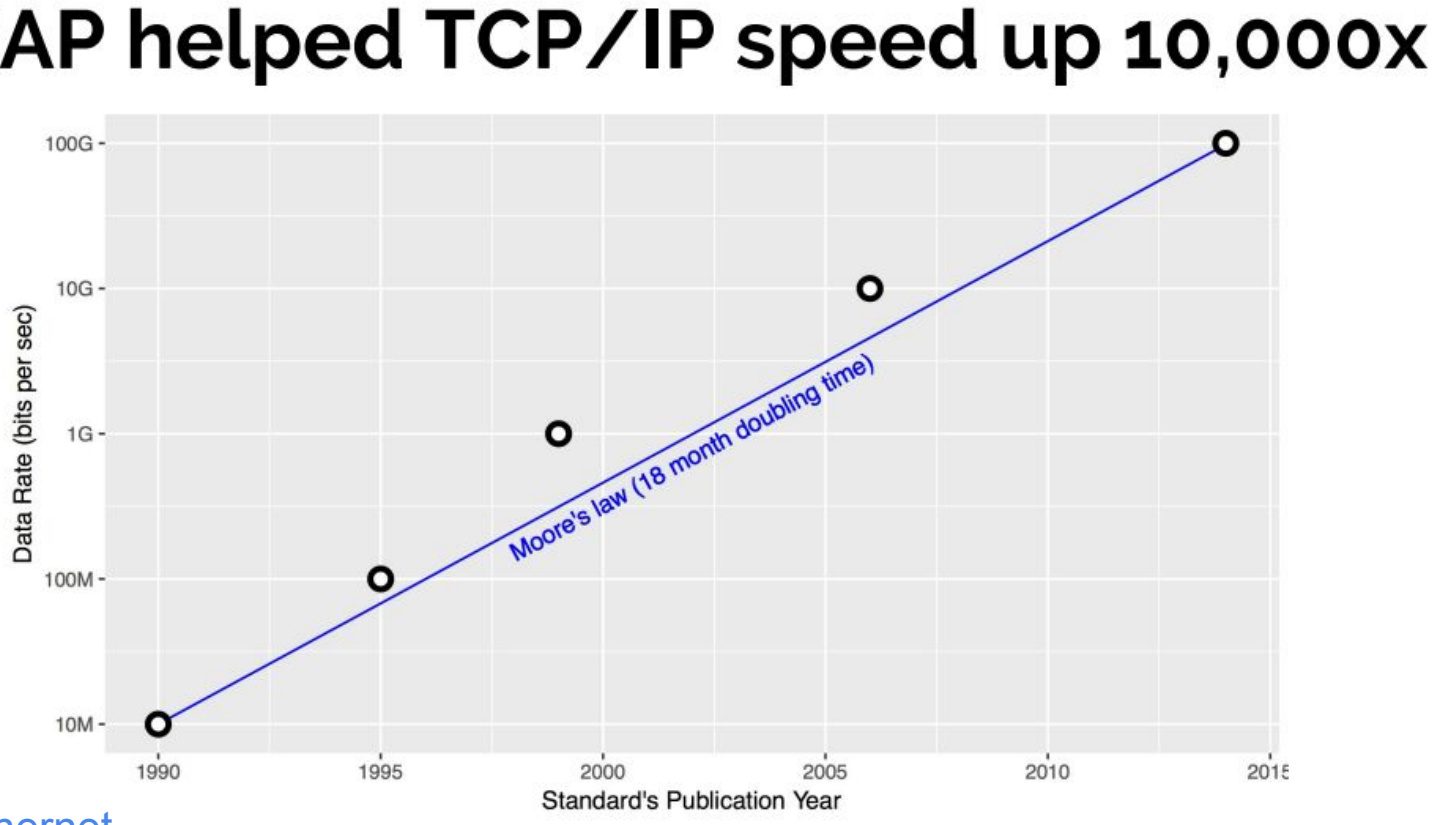


https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Meanwhile, Ethernet keeps getting faster

Von Jacobson: Evolving from AFAP: Teaching NICs about time, Netdev 2018

10Mbps in 1990
100Mbps in 1995
1000Mbps in 1999
10Gbps in 2006
100Gbps in 2014
400GbE 标准化 in 2017



https://en.wikipedia.org/wiki/Terabit_Ethernet

(25 years of Ethernet evolution)

To fill the gap: offload / 工作负载下放

download	下载	例：下载软件
upload	上载、上传	例：上传文件
unload	卸载	例：卸载 app
offload	下放	例：Kernel TLS offload
		Hardware offload

My take: 程序/代码语句级的性能优化已经没多少油水，硬件加速才是王道。

Offload to hardware 1: TCP/IP header checksums

Early (1980s) example of offloading.

cksum: not only waste of CPU cycles, but also double memory bandwidth.

Similarly, even entry level NICs can do CRC calculation and verification.

TCP/IP Performance on VAX by Bill Joy, 1981-11-11

As an experiment to investigate the performance of the resulting TCP/IP implementation, we **transmitted 4 Megabytes** of data between two user processes on different machines. ...
Sending the data from our 11/750 to our 11/780 through TCP/IP **takes 28 seconds**. This includes all the time needed to set up and tear down the connections, for an **user-user throughput of 1.2 Megabaud**. During this time the 11/750 is CPU saturated, but the 11/780 has about 30% idle time. The time spent in the system processing the data is spread out among handling for the Ethernet (20%), IP packet processing (10%), TCP processing (30%), **checksumming (25%)**, and user system call handling (15%), with no single part of the handling dominating the time in the system.

... With **checksum calculation support from the interface hardware** the user-user bandwidth would rise to about 3.5 Megabaud.

http://cdn.preterhuman.net/texts/computing/gopher-archive/gopher.quux.org/Archives/usenet-a-news/FA.tcp-ip/81.11.18_ucbvax.5236_fa.tcp-ip.txt

Offload to hardware 2: New CPU instructions

XML 2008 https://en.wikipedia.org/wiki/SSE4.2

Intel SSE4.2 introduces four instructions for “string and text processing”

AES-NI 2010 https://en.wikipedia.org/wiki/AES_instruction_set

Increase throughput from approx. 28.0 cycles/byte to 3.5 cycles/byte (8x*)

SHA 2013 https://en.wikipedia.org/wiki/Intel_SHA_extensions

On my AMD zen3 5900x, SHA1 2.6GB/s, AES-128-CBC 1.8GB/s, NIC 40Gbps

BMI2 2013 Haswell https://en.wikipedia.org/wiki/BMI2 POPCNT, LZCNT, MULX

My take: 降维打击，就算手写 SIMD intrinsics 也敌不过 CPU 加几条新指令。

Offload to kernel, by adding new syscalls

Early examples:

sendfile(2) 1998 FreeBSD 3.0, 1999 Linux 2.2

recv**m**msg() 2010 Linux 2.6.33 send**m**msg() 2011 Linux 3.0 UDP/QUIC

Recent example: https://docs.kernel.org/networking/tls-offload.html

kTLS 2015 FreeBSD, 2017 Linux 4.13 send, 2018 Linux 4.17 receive

Future example:*

switchto(2) 2020 Paul Turner https://news.ycombinator.com/item?id=24688225

Kernel supports user-space threading https://lkml.org/lkml/2020/7/22/1202

My take: 如果 Linux 内核支持这个，就没 coroutines 什么事儿了。

TLS offload to kernel / NIC (网卡)

TLS - Transport Layer Security, https://letsencrypt.org/, R.I.P. Peter Eckersley

Serving Netflix video traffic: offload to kernel, save memory b/w, like sendfile().

2015 Optimizing TLS for High–Bandwidth Applications in FreeBSD
2016 36Gb/s Improving High–Bandwidth TLS in the FreeBSD kernel

2019 90Gb/s Kernel TLS and **hardware** TLS offload in FreeBSD 13 2019-Oct
2019 200Gb/s https://people.freebsd.org/~gallatin/talks/euro2019.pdf
2021 400Gb/s Serving Netflix Video at 400Gb/s on FreeBSD
2022 800Gb/s Serving Netflix Video Traffic at 400Gb/s **and Beyond**

My take: 买块好网卡，胜过调代码。Chelsio T6 and Mellanox ConnectX-6 Dx

Hardware accelerators 以桌面 CPU/GPU 为例

Perf/$, 好比：手机购置费/初装费
e.g. **Peak** FP32 ops in 2021, with 500$

CPU: AMD 5900X, AVX2, 12c24t, 4.8GHz
32*24*4.8 = 3.69Tflops / 500$
7.37 Gflops/$
https://en.wikipedia.org/wiki/FLOPS

GPU: RTX 3070, 5888 CUDA cores, 1.5GHz
2 * 5888 * 1.5 = 17.66 TFlops / 500$
35.33 Gflops/$
https://en.wikipedia.org/wiki/GeForce_30_series

DSP: TI C6678, 22.4 Gflops/core @ 1.4GHz, 150$
8 * 22.4 = 180 Gflops / 150$, 1.2 Gflops/$

My take: 用 CPU 算 32-bit 浮点数(DL)是浪费电/时间。

Perf/Watt, TDP　好比：每月套餐费/月租费

1 W * 1 years = 8.76 kWh, 0.12$/kWh, ~ 1$
5 W * 5 years ~= 25$，省电 = 省钱

CPU: AMD 5900X, 3.69 TFlops/105W
35.1 Gflops/W, 4.38W/logical core

GPU: RTX 3070, 17.66 TFlops/220W
80.3 Gflops/W, 37.4mW/CUDA core (117x)

TPUv2: 45 Tflops/280W (is it FP32 or BF16?)
160.7 Gflops/W, 2017
https://en.wikipedia.org/wiki/Tensor_Processing_Unit

Performance tips

Kernighan & Pike《程序设计实践》第 7 章:

Use a better algorithm or data structure.

E.g. searching, *O*(*N*) for linked-list, *O*(log *N*) for binary search tree.

TCP reassemble queue, 2016 Linux 4.9 changed from linked-list to rb-tree

`out_of_order_queue` is now a tree, for receiving segments.

FreeBSD 12 rewrote `tcp_reass()` in 2018, in response to CVE-2018-6922, but still uses linked-list, not BST. Same in FreeBSD 13.

TCP retransmit queue, 2018 Linux 4.15 changed from linked-list to rb-tree

Less critical, optimize for SACK, according to Eric Dumazet.

Performance links

John Ousterhout: Always Measure One Level Deeper, CACM 2018 July

https://cacm.acm.org/magazines/2018/7/229031-always-measure-one-level-deeper/fulltext

Brendan Gregg: LISA21 Computing Performance 2021: What's on the Horizon

https://www.brendangregg.com/Slides/YOW2021_ComputingPerformance/

Producing Wrong Data Without Doing Anything Obviously Wrong, 2009 SIGPLAN

https://users.cs.northwestern.edu/~robby/courses/322-2013-spring/mytkowicz-wrong-data.pdf

Brendan Gregg《性能之巅》第 2 版

推荐语 https://www.cnblogs.com/Solstice/p/gregg.html

Back to C++

Modern C++ Tooling

Use clang as a library

Parsing C++ used to be hard, if not possible

C++ syntax is not context-free, but rather context-sensitive.

Parsing C++ is a multi engineer year effort, 100k+ LOC.
clangParse, clangAST, clangAST did the heavy-lifting for us

C is almost context-free, one well-known conflict: typedef-name, is T a typedef?

Statement: T * p; T multiples p, or define p as a pointer to T.
Expression: (T)*p T multiples p, or cast *p to type T. foo((T)*p);
Statement: T (p); call function T on parameter p, or define p as a variable
https://en.wikipedia.org/wiki/Lexer_hack

Clang is both a C++ compiler and a library for building source code tools.

Ordinarily programmer like yours truly can build source tool too.

Indexing and navigation of large codebase

Find definition of a function/field.

List all callers of a particular overload of a function. Find all usage of a field.

Regex matching is insufficient as name resolution is so complex

Kythe as the indexer: https://kythe.io

Indexing C++ code with lots of macros and templates are painful

Example UI:

https://source.chromium.org/chromium/chromium/src/+/master:base/files/file.h

Name resolution in C++ is complex

Rename the parameter x or data member Point::x, use clang-rename

```
struct Point {
  int x, y;

  Point(int x, int y)
    : x(x), y(y) {}
};
```

```
// Another example of confusion names
struct stat stat; // stat 结构体 与 stat 对象
stat("somefile", &stat);  // stat() 函数
print(stat.st_size);  // Which stat above?
```

clangParse, clangAST, clangSema did the heavy-lifting, thankfully.

```
// Find employee, current or former. Overload resolution 有时反直觉。
Lookup(string_view firstName, string_view lastName, bool former=false);
Lookup(string_view fullName, bool former=false);
employee = Lookup("Shuo", "Chen");  // char* ⇒ bool takes precedence of UDC
```

Source code tools

Prerequisite: compile your code with clang (-c only). Linux kernel since 5.7, 2020.

clang-format Five coding styles: LLVM, Google, Chromium, Mozilla, WebKit

clang-tidy Linter with fixes

clang-rename, clang-refactor, ClangMR

C++ refactoring made possible, customizable and scalable
scoped_ptr ⇒ std::unique_ptr

clangd Intelligent code completion in your editor (VIM/Emacs)

Thread Safety Annotations in Clang

1. Annotate your mutex and lock_guard
2. Annotate thread safety assumptions
3. Compile with Clang -Wthread-safety

```
class Counter {
 public:
  void increase();
  int get() const;
 private:
  mutable MutexLock mutex_;
  int count_ GUARDED_BY(mutex_);
};
```

```
void Counter::increase() {
  MutexLockGuard L(mutex_);
  ++count_;  // Good
}

int Counter::get() const {
  // warning: reading variable'count_'
  // requires holding mutex 'mutex_'.
  return count_;  // Data race!
}
```

Sanitizers: Address, Memory, Thread, etc.

Modern Valgrind memcheck, 2~3x slowdown, instead of 10~50x.

- First added to gcc 4.8, mainstream maintained in clang/llvm.

Common errors caught:

- AddressSanitizer: use-after-free, out-of-bound read/write, double-free
- MemorySanitizer: use-of-uninitialized-values
- ThreadSanitizer: data race
- UndefinedBehaviorSanitizer: integer overflow (could cause security bugs)

One sanitizer at a time, build multiple sanitized binaries for (unit-)testing

AddressSanitizer (asan), accessing invalid address

1. Compile with -fsanitize=address
2. Run your binary

```
struct Node {
  Node* next;
  int value;
};
void freeListBad(Node* head) {
  for (; head != nullptr;
       head = head->next) {
    delete head;
  }
}
```

```
$ g++ -fsanitize=address -g asan.cc
$ ./a.out

==4600==ERROR: AddressSanitizer:
heap-use-after-free on address 0xXXXXXX

READ of size 8 at 0x602000000010 thread T0
  #0 0x55f589cd61c9 in freeListBad(Node*)
                                  asan.cc:10
  #1 0x55f589cd6227 in main asan.cc:18
```

MemorySanitizer (msan): uninitialized read

Sample: branch on uninitialized data member.

```
class Foo {
 public:
  void print() {
    if (initialized_) {
      printf("Good!\n");
    }
  }
 private:
  bool initialized_;
};
```

```
$ clang++ -fsanitize=memory \
  -fsanitize-memory-track-origins -g msan.cc
$ ./a.out
==5209==WARNING: MemorySanitizer:
use-of-uninitialized-value
    #0 0x49a76a in Foo::print() msan.cc:7:9
    #1 0x49a6e7 in main msan.cc:18:5
Uninitialized value was created by an allocation
of 'f' in the stack frame of function 'main'
    #0 0x49a6c0 in main msan.cc:16
```

Disable ObjectPool / MemoryPool when sanitizing

MemoryPool/ObjectPool might hide errors. Use new/delete directly then.

The memory is never return to MemoryPool, but ~MemoryPool() frees all memory.

- In long running service, it's memory leak. But it won't surface in unittests.

A memory pool could also hide buffer overrun caught by asan.

- It may allocate a large chunk of memory and gives small chunks to clients

A stateful object is reused, but its reset() function didn't clear all state.

- You may read stale values from previous usage, instead of caught by msan.

Use and extend Clang-Tidy, modern C++ linter

Keep your codebase in a consistent style, save learning and maintenance cost.

Examples:

- std::string_view should be pass-by-value, not by const reference.
- Passing-by-non-const-reference was disallowed by Google C++ style*.
- Enforce your coding styles: ClassName, data_member_, local_variable.

Pick a subset of checks that fits your codebase

- Don't turn on all checks, styles conflict, some checks are too idealistic
- https://clang.llvm.org/extra/clang-tidy/checks/list.html

Some checks also come with suggested fixes

```
$ clang-tidy --checks=-*,bugprone-misplaced-operator-in-strlen-in-alloc -fix

tidy0.c:5:15: warning: addition operator is applied to the argument of strlen
instead of its result [bugprone-misplaced-operator-in-strlen-in-alloc]
  char* out = malloc(strlen(str + 1));
                     ^      ~~~~~~~~~~~~~~~
                            strlen(str) + 1
```

https://clang.llvm.org/extra/clang-tidy/checks/abseil/string-find-startswith.html

if (s.find("Hello World") == 0) ⇒ if (absl::StartsWith(s, "Hello World"))

https://clang.llvm.org/extra/clang-tidy/checks/google/explicit-constructor.html

https://clang.llvm.org/extra/clang-tidy/checks/modernize/avoid-bind.html

Write your own compile warnings

How to find similar misuses in your codebase? Regexp or machine learning?

```
const char* str = GetName();

// O(N^2) algorithm by accident if compiler can't hoist strlen()
// out of the loop
for (int i = 0; i < strlen(str); ++i) {
    if (...) {
        // pass 'str' to another function
    }
}
```

Dump Clang AST (Abstract Syntax Tree)

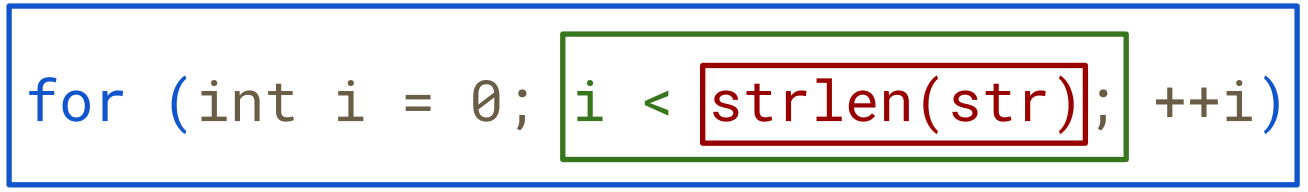
```
clang++ -fsyntax-only -Xclang -ast-dump slowloop.cc
```

```
-ForStmt 0xbea6d0 <line:7:3, line:9:9>
 |-DeclStmt 0xbea398 <line:7:8, col:17>
 | `-VarDecl 0xbea310 <col:8, col:16> col:12 used i 'int' cinit
 |   `-IntegerLiteral 0xbea378 <col:16> 'int' 0
 |-<<<NULL>>>
 |-BinaryOperator 0xbea4f8 <col:19, col:33> 'bool' '<'
 | |-ImplicitCastExpr 0xbea4e0 <col:19> 'int' <LValueToRValue>
 | | `-DeclRefExpr 0xbea3b0 <col:19> 'int' lvalue Var 0xbea310 'i' 'int'
 | `-CallExpr 0xbea4a0 <col:23, col:33> 'int'
 |   |-ImplicitCastExpr 0xbea488 <col:23> 'int (*)(const char *)' <FunctionToPointerDecay>
 |   | `-DeclRefExpr 0xbea438 <col:23> 'int (const char *)' lvalue Function 0xbe9f30 'strlen' 'int
 |   `-ImplicitCastExpr 0xbea4c8 <col:30> 'const char *' <LValueToRValue>
 |     `-DeclRefExpr 0xbea418 <col:30> 'const char *' lvalue ParmVar 0xbea038 'str' 'const char *'
 |-UnaryOperator 0xbea538 <col:36, col:38> 'int' lvalue prefix '++'
 | `-DeclRefExpr 0xbea518 <col:38> 'int' lvalue Var 0xbea310 'i' 'int'
 `-IfStmt 0xbea6b8 <line:8:5, line:9:9>
```

```
for (
  int i = 0;
  i < strlen(str);
  ++i) {
    if (...) {
```

Craft an AST matcher with clang-query

ASTMatchers is a domain specific language to create predicates on Clang's AST




```
forStmt(hasCondition(hasDescendant(
    callExpr(callee(functionDecl(hasName("strlen")))))))
```

Exercise: How about while-loop

Auto-fix recurring mistakes in your codebase

```
// What's wrong with follow code of thread-safe counter?
class Counter {
 public:
  void Increase() {
    std::unique_lock<std::mutex> (mu_);  // WRONG, no locking at all
    ++count_;                           L(mu_);  // Correct
  }                                     // Why unique_lock has default-ctor?
 private:
  mutable std::mutex mu_;
  int count_ = 0;
};
```

Warnings with -Wall (-Wparentheses) or -Wshadow

AST with or without variable name

```
lock_guard<mutex> L(mu_);
```

```
-DeclStmt 0xad4648 <line:12:5, col:39>
 `-VarDecl 0xad3f68 <col:5, col:38> col:33 L 'std::lock_guard<std::mutex>':'std::lock_guard<std::mutex>' callinit
  `-CXXConstructExpr 0xad4618 <col:33, col:38> 'std::lock_guard<std::mutex>':'std::lock_guard<std::mutex>' 'void (std::mutex &)'
   `-MemberExpr 0xad3ef8 <col:35> 'std::mutex':'std::mutex' lvalue ->mu_ 0xad05a0
    `-CXXThisExpr 0xad3ee8 <col:35> 'const Counter *' implicit this
```

Calls 1-arg ctor `lock_guard::lock_guard(mutex&)`

```
unique_lock<mutex> (mu_);
```

```
-DeclStmt 0x2f78250 <line:14:5, col:39>
 `-VarDecl 0x2f750c0 <col:5, col:38> col:35 mu_ 'std::unique_lock<std::mutex>':'std::unique_lock<std::mutex>' callinit
  `-CXXConstructExpr 0x2f78228 <col:35> 'std::unique_lock<std::mutex>':'std::unique_lock<std::mutex>' 'void () noexcept'
```

Calls default constructor `unique_lock::unique_lock()`

A possible matcher

```
std::unique_lock<std::mutex> (mu_);
```

```
-DeclStmt 0x2f78250 <line:14:5, col:39>
 `-VarDecl 0x2f750c0 <col:5, col:38> col:35 mu_ 'std::unique_lock<std::mutex>':'
   `-CXXConstructExpr 0x2f78228 <col:35> 'std::unique_lock<std::mutex>':'std::ur
```

```
declStmt(has(varDecl(hasType(asString("std::unique_lock<std::mutex>")),
                hasInitializer(cxxConstructExpr(argumentCountIs(0))))))
```

Variable declaration of `"std::unique_lock<std::mutex>"` that calls default ctor

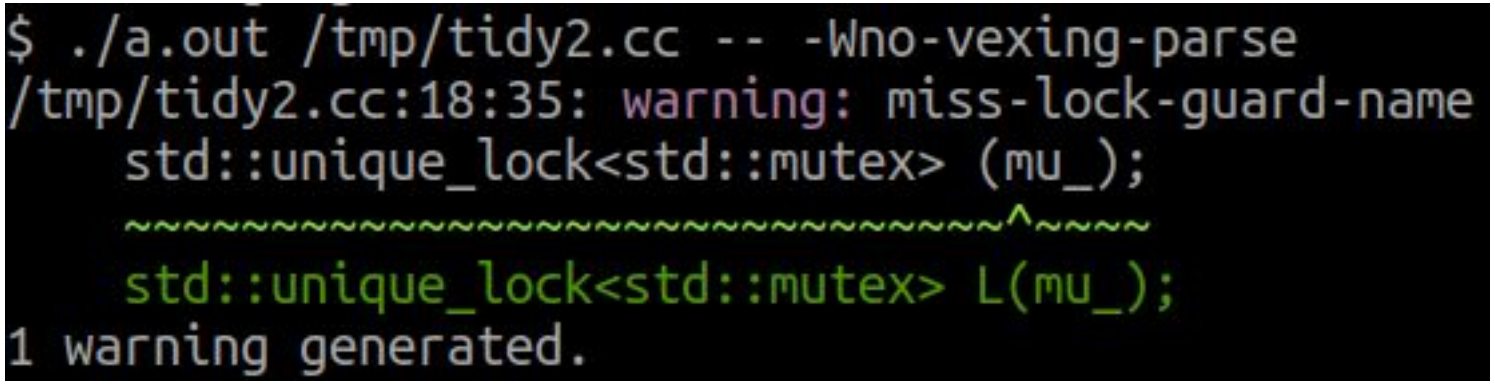
This matcher is no ideal, false positives and false negatives:

- A local unique_lock that doesn't hide mutex member? (Is this ever useful?)
- Hard coded type name, what about `typedef`s, `unique_lock<timed_mutex>`?

Suggest a fix: Inserting a variable name

```
auto fix = clang::FixItHint::CreateReplacement(
    stmt->getSourceRange(), typeName + " L(" + varName + ");");

clang::DiagnosticBuilder diag = getDiagnostics(...);
diag << fix;
```



Build a refactor tool to clean-up codebase

Three ways to build the tool

- A standalone tool with its own main() function
- A plugin to clang, a shared library loaded at runtime
- Your own branch of clang-extra-tools source tree, built as part of clang-tidy

Two APIs

- RecursiveASTVisitor
- ASTMatcher

Exercise: Build a tool to check usage of LockGuard class for your codebase.

- Then upstream to clang-extra-tools, let it be part of official clang-tidy.

Static Analysis using Control Flow Graph

ASTMatcher can't explores all possible branches in code, hard to catch bugs like use-after-move, value not always initialized, etc.

CFG represents a block of statements (inside a function), their logical sequences.

Beyond CFG: translation unit or whole program analysis, which is too advanced.

Maybe put in Clang's static analyzer checks, instead of clang-tidy.

Sorry I don't have a good demo of CFG in this talk.

Using clang as a library

Non-experts can build their own source-code tool to:

Warn wrong code pattern specific to their codebase / library

    As programmers don't read warnings in code comments

Detect (and fix) common mistakes, maybe run as pre-submit checks

Refactor the codebase, migrate deprecated usage to recommended one.

    Reduce the burden of maintenance

Once fixed a bug, add a unittest, and maybe a clang-tidy check if it's common.

Google’s open-source C++ libraries

Google's Abseil C++ library abseil.io/docs/cpp/

Swiss-tables, `flat_hash_map`, `flat_hash_set`, etc.
https://abseil.io/blog/20180927-swisstables

`absl::StrSplit()` for Splitting Strings. strtok(3) 嘛，这有何难？
https://abseil.io/docs/cpp/guides/strings#absl strsplit-for-splitting-strings

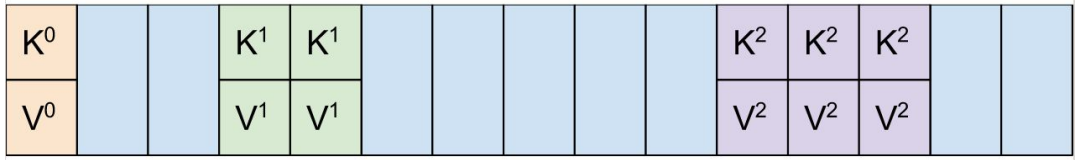
Adaptation to Returned Types: vector<string_view>, set<string>, pair<>

Random library, because rand() Considered Harmful by Stephan T. Lavavej, 2013

```
absl::BitGen bitgen;   // Harmful: x = elems[rand() % elems.size()];
size_t index = absl::Uniform(bitgen, 0u, elems.size());
auto x = elems[index];
```

absl::flat_hash_map

Use `flat_hash_map` by default.



Memory locality, avoid cache misses.

Pointer chasing is harmful.

```
treemap - std::map
hashmap - std::unordered_map
nodemap - absl::node_hash_map
flatmap - absl::flat_hash_map
```

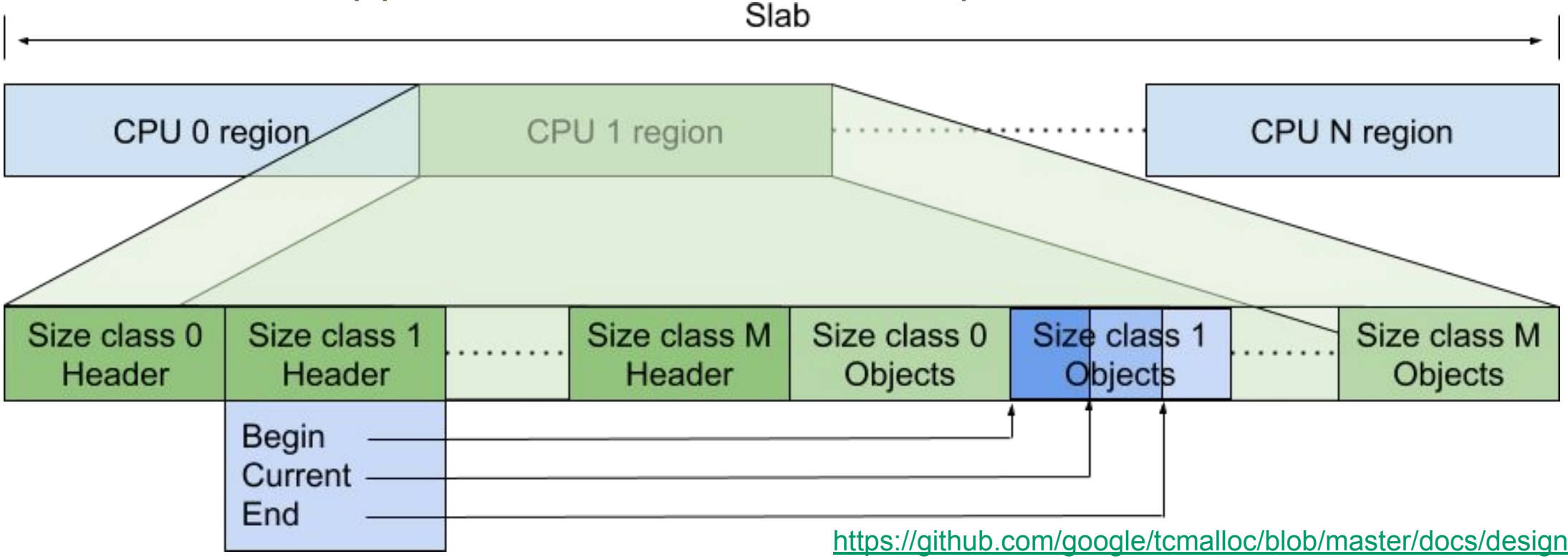
https://github.com/chenshuo/recipes/blob/master/benchmark/bm_containers.cc#L26

Benchmark	Time	CPU
BM_wordcount<treemap>/10	0.001 ms	0.001 ms
BM_wordcount<treemap>/100	0.015 ms	0.015 ms
BM_wordcount<treemap>/1000	0.250 ms	0.250 ms
BM_wordcount<treemap>/10000	3.88 ms	3.88 ms
BM_wordcount<treemap>/100000	70.0 ms	70.0 ms
BM_wordcount<treemap>/1000000	1112 ms	1112 ms
BM_wordcount<treemap>/10000000	16941 ms	16940 ms
BM_wordcount<hashmap>/10	0.001 ms	0.001 ms
BM_wordcount<hashmap>/100	0.011 ms	0.011 ms
BM_wordcount<hashmap>/1000	0.147 ms	0.147 ms
BM_wordcount<hashmap>/10000	1.75 ms	1.75 ms
BM_wordcount<hashmap>/100000	26.0 ms	26.0 ms
BM_wordcount<hashmap>/1000000	823 ms	823 ms
BM_wordcount<hashmap>/10000000	6860 ms	6859 ms
BM_wordcount<nodemap>/10	0.001 ms	0.001 ms
BM_wordcount<nodemap>/100	0.012 ms	0.012 ms
BM_wordcount<nodemap>/1000	0.153 ms	0.153 ms
BM_wordcount<nodemap>/10000	1.75 ms	1.75 ms
BM_wordcount<nodemap>/100000	24.3 ms	24.3 ms
BM_wordcount<nodemap>/1000000	894 ms	894 ms
BM_wordcount<nodemap>/10000000	10681 ms	10681 ms
BM_wordcount<flatmap>/10	0.001 ms	0.001 ms
BM_wordcount<flatmap>/100	0.010 ms	0.010 ms
BM_wordcount<flatmap>/1000	0.120 ms	0.120 ms
BM_wordcount<flatmap>/10000	1.25 ms	1.25 ms
BM_wordcount<flatmap>/100000	16.3 ms	16.3 ms
BM_wordcount<flatmap>/1000000	512 ms	512 ms
BM_wordcount<flatmap>/10000000	4632 ms	4632 ms

New TCMalloc with per-CPU cache

Original Thread Caching Malloc, now **per-CPU cache** since Linux 4.18, 2018.

Kernel must support **RSEQ** - Restartable sequences



https://github.com/google/tcmalloc/blob/master/docs/design.md

C++ Tips of the Week, Google's *Effective C++*

https://abseil.io/tips/

- **April 06, 2020**

  Tip of the Week #163: Passing `absl::optional` parameters
  Tip of the Week #171: Avoid Sentinel Values
  Tip of the Week #172: Designated Initializers
  Tip of the Week #173: Wrapping Arguments in Option Structs
  Tip of the Week #175: Changes to Literal Constants in C++14 and C++17.
  Tip of the Week #176: Prefer Return Values to Output Parameters
  Tip of the Week #177: Assignability vs. Data Member Types

- **December 19, 2019**

  Tip of the Week #108: Avoid `std::bind`

Summary: 开阔思路、逢山开路、遇水搭桥

Compiler is customizable

- Thread Safety Annotations

```
Mutex mu_;
int value_ GUARDED_BY(mu_);
```

Kernels are customizable

- Willem de Bruijn added epoll_pwait2(2) to Linux 5.11 in 2021, 纳秒 timeout

CPUs can add new features

Heterogeneous computing (accelerators)

- GPU is an accelerator for floating point
- TPU is an accelerator for matrix multiplication
- NIC can do TCP segmentation, also can be an accelerator for encryption (TLS)

C++ 吐槽大会

C++ Move Semantics - The Complete Guide
by **Nicolai M. Josuttis**, 260 pages.
http://www.cppmove.com/

The Nightmare of Initialization in C++,
CppCon 2018, Seattle
http://www.josuttis.com/cpp/c++initialization.pdf

Link



You Retweeted

Ólafur Waage @olafurw · Sep 4

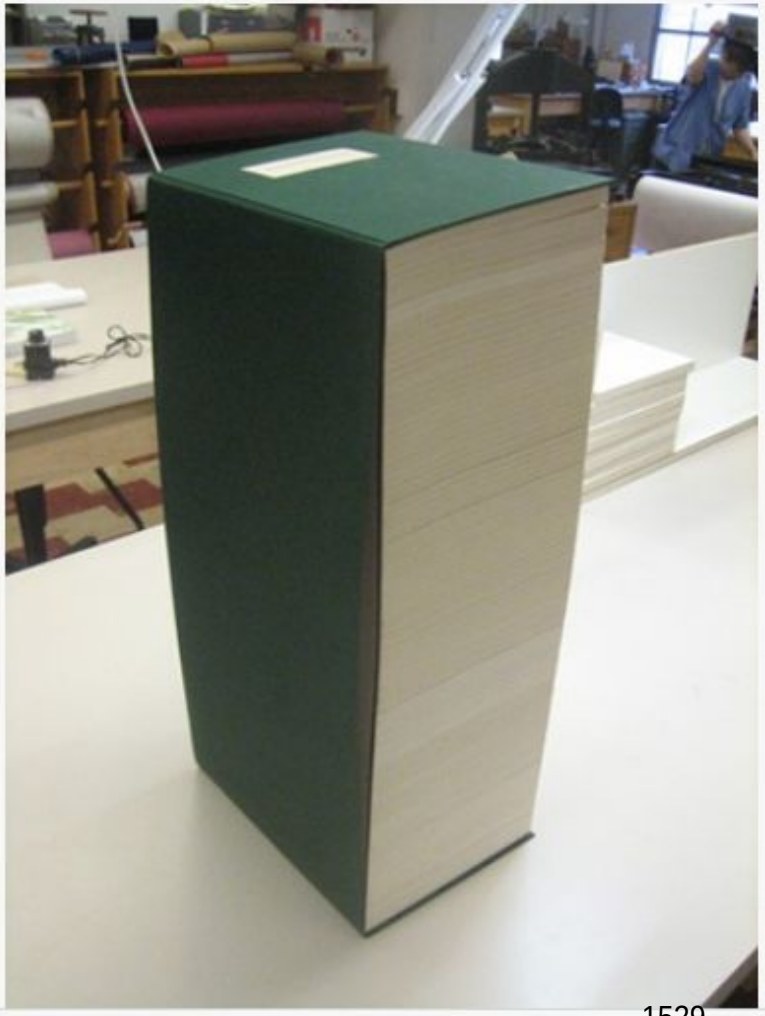
C++ variable initialization.

Link

Madza @madzadev · Sep 3

Give this book a coding-related title

Show this thread



14 13 184

C++: Rules for Different Ways of Initialization

		always has defined value	narrowing is error	works for `initializer_list<>`	`explicit` conversion supported	works for aggregates	works for `auto`	works for members
	Type `i;`	no	-	no	-	✓ (no init)	no	✓
	Type `i{};`	✓	-	✓	-	✓	no	✓
	Type `i();`	**function declaration**						
direct initialization	*Type* `i{x};`	✓	✓[1]	✓	✓	✓	✓[2]	✓
direct initialization	*Type* `i(x);`	✓	no	no	✓	since C++20, not nested	✓	no
direct initialization	*Type* `i(x, y);`	✓ (2 args)	no	no	✓	since C++20, not nested	✓	no
copy initialization	*Type* `i = x;`	✓	no	no	no	no	✓	✓
copy initialization	*Type* `i = {x};`	✓	✓[1]	✓	no	✓	✓ init-list	✓
copy initialization	*Type* `i = (x);`	✓ (1 arg)	no	no	no	since C++20, not nested	✓ (1 arg)	✓ (1 arg)
copy initialization	*Type* `i = (x, y);`	✓ (last arg)	no	no	no	since C++20, not nested	✓ (last arg)	✓ (last arg)

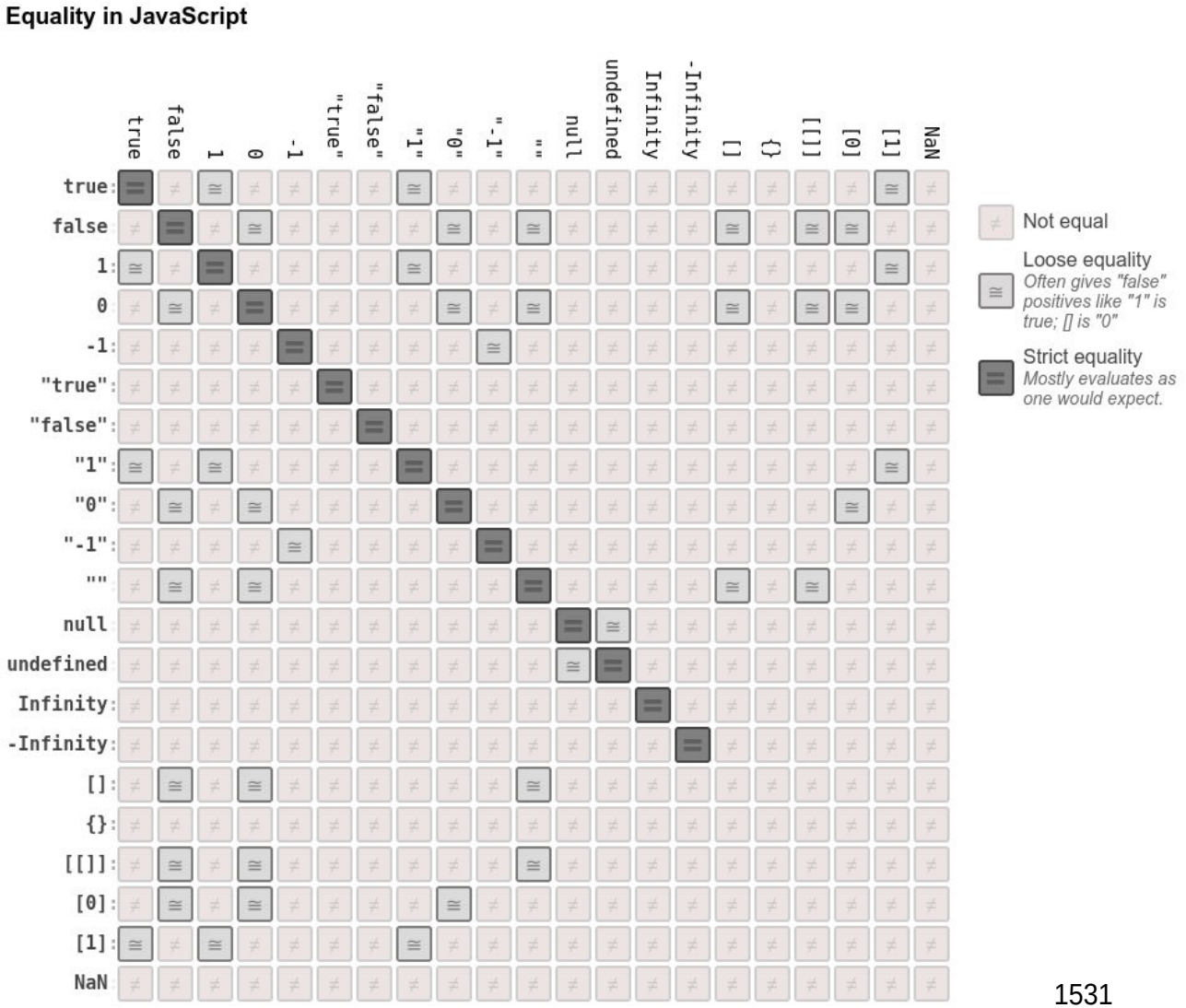
[1]: **g++** needs `-pedantic-errors` or `-Werror=narrowing` to detect narrowing errors

[2]: `std::initializer_list<>` before g++ 5, clang 3.8, and Visual Studio 2015

josuttis.com/cpp/c++initialization.pdf
© creativecommons.org/licenses/by/4.0/

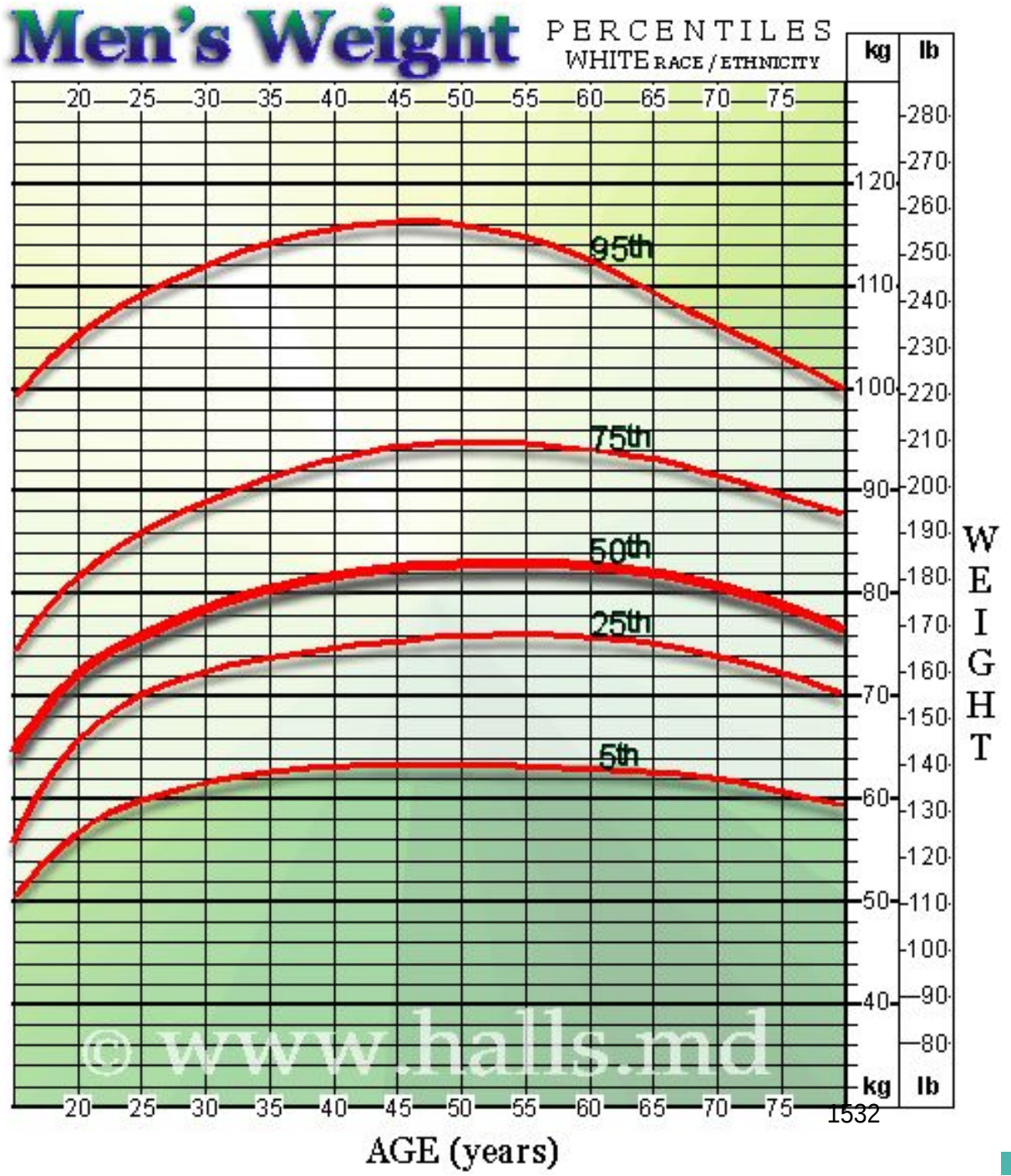
JavaScript 也不过如此吧？

https://dorey.github.io/JavaScript-Equality-Table/unified/



Off topic / 题外话

https://halls.md/average-weight-men/



Q & A
10 minutes



宋顾杨

蚂蚁集团技术专家

蚂蚁集团计算智能技术部技术专家，Ray开源社区committer。2017年以来主要参与和推动了Ray在蚂蚁集团从0到1再到万台服务器规模的落地。 近两年活跃在Ray开源社区，Ray C++开发框架的发起人，致力于解决C++分布式系统的通用问题，推动C++在分布式领域的创新。长期从事的工作还包括分布式系统的运行时环境构建、可观察性能力建设和易用性优化等。

CPP-Summit

宋顾杨
蚂蚁集团技术专家

利用Ray快速构建C++分布式系统

自我介绍

2020-今　　Ray开源社区Committer

- C++分布式框架
- Runtime Environments



2017-今　　蚂蚁集团-新计算引擎团队

- 参与和推动了Ray在蚂蚁集团从0到1再到万台服务器规模的落地

2015-2016　蚂蚁集团-应用网络团队

- 7层网关，支付宝钱包网络自研协议

议程

1 分布式系统复杂性

2 Ray简介

3 利用Ray快速构建分布式系统

4 技术内幕

5 Ray在蚂蚁的应用

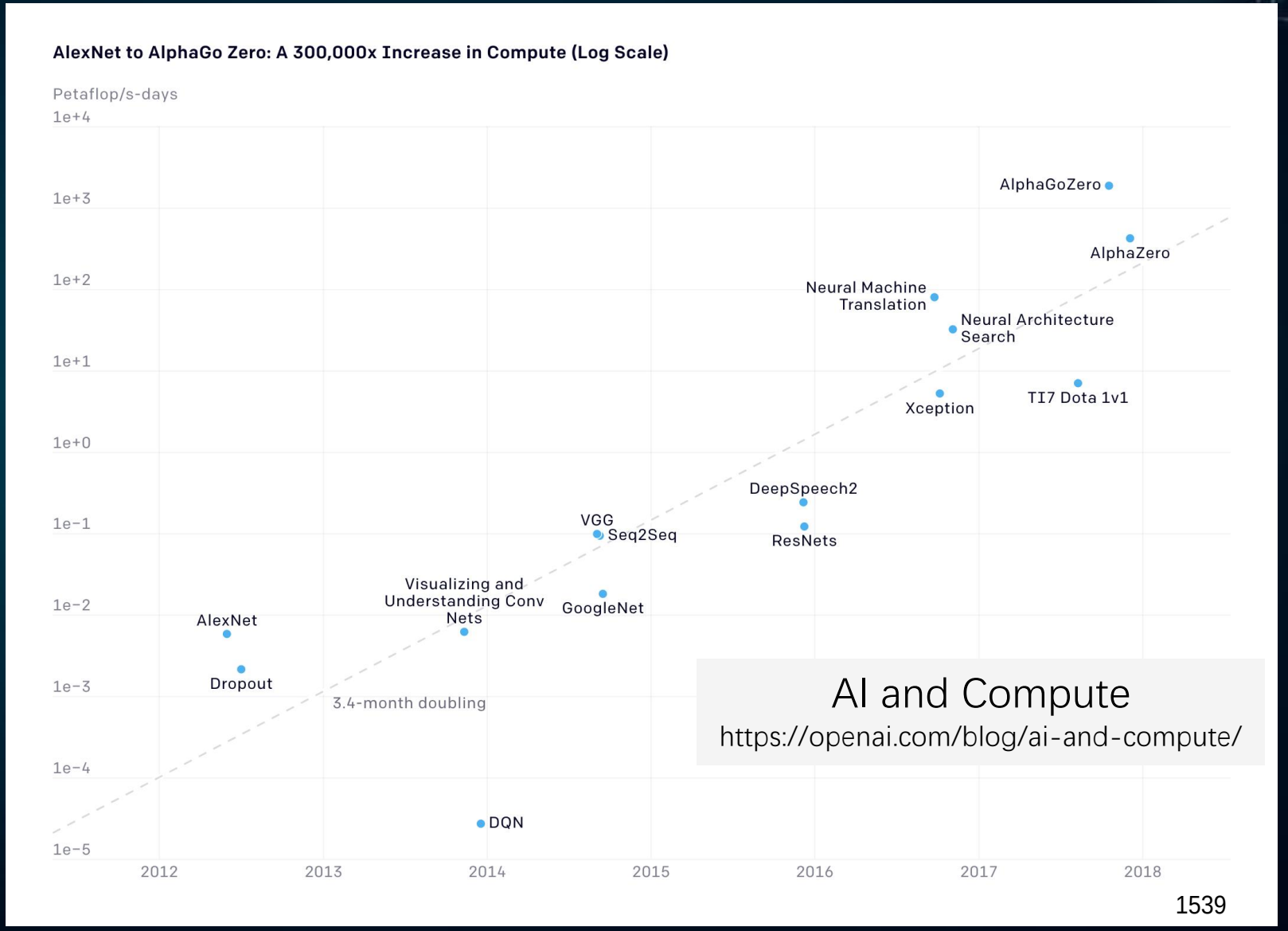
6 Ray开源现状

01

分布式系统复杂性

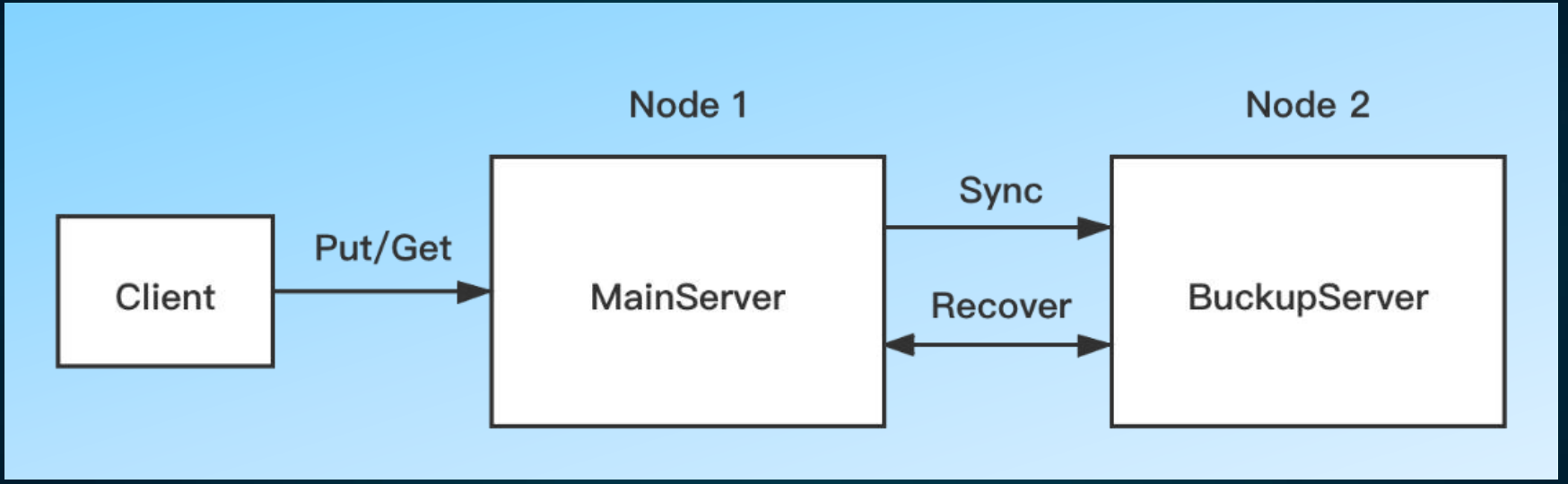
分布式系统发展

- 大数据
- 机器学习
- 推荐系统
- 搜索引擎
- ……

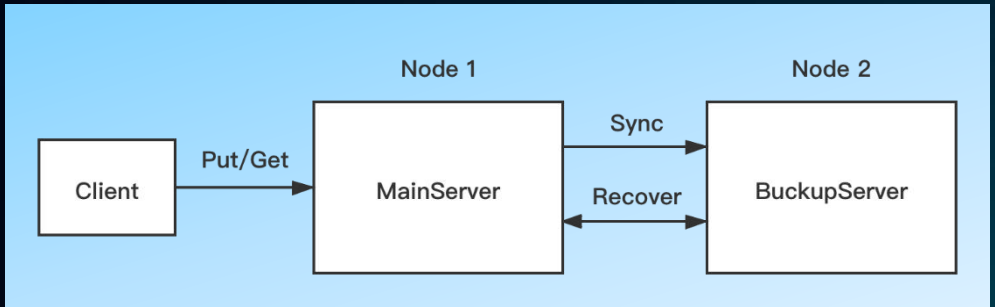


1539

问题：构建分布式系统需要解决哪些问题？



分布式系统通用问题



- 组件通信
- 数据存储
- 部署
- 资源调度
- 故障恢复
- 运维与监控

研发运维成本



02

Ray简介

Ray是什么

- 由加州大学伯克利分校RISELab发起并开源
- 通用分布式计算引擎
- 新一代计算基础设施



Ray通用分布式编程API

Key idea：不绑定计算模式，
把单机编程中的基本概念分布式化

Function -> Task
Class -> Actor

Ray通用分布式编程API——无状态计算单元Task

单机(function)

```
int heavy_compute(int value) {
  # Heavy computation here.
  # return ...
}

std::vector<int> results;
for (int i = 0; i < 10000; i++) {
  results.push_back(heavy_compute(i));
}

for (auto result: results) {
  std::cout << result << std::endl;
}
```

分布式(task)

```
int heavy_compute(int value) {
  # Heavy computation here.
  # return ...
}

RAY_REMOTE(heavy_compute);

std::vector<ray::ObjectRef<int>> results;
for (int i = 0; i < 10000; i++) {
  results.push_back(ray::Task(heavy_compute).Remote(i));
}

for (auto result: ray::Get(results)) {
  std::cout << *result << std::endl;
}
```

把普通function变成remote task

异步调用，远程执行

获取结果

编写分布式程序像写单机程序一样简单!

Ray Task实现基本原理——分布式Object Store

进程级future —— std::future

```
1   // future from an std::async
2   std::future<int> future =
3       std::async(std::launch::async,
4                  heavy_compute, 1);
5   int result = future.get();
```

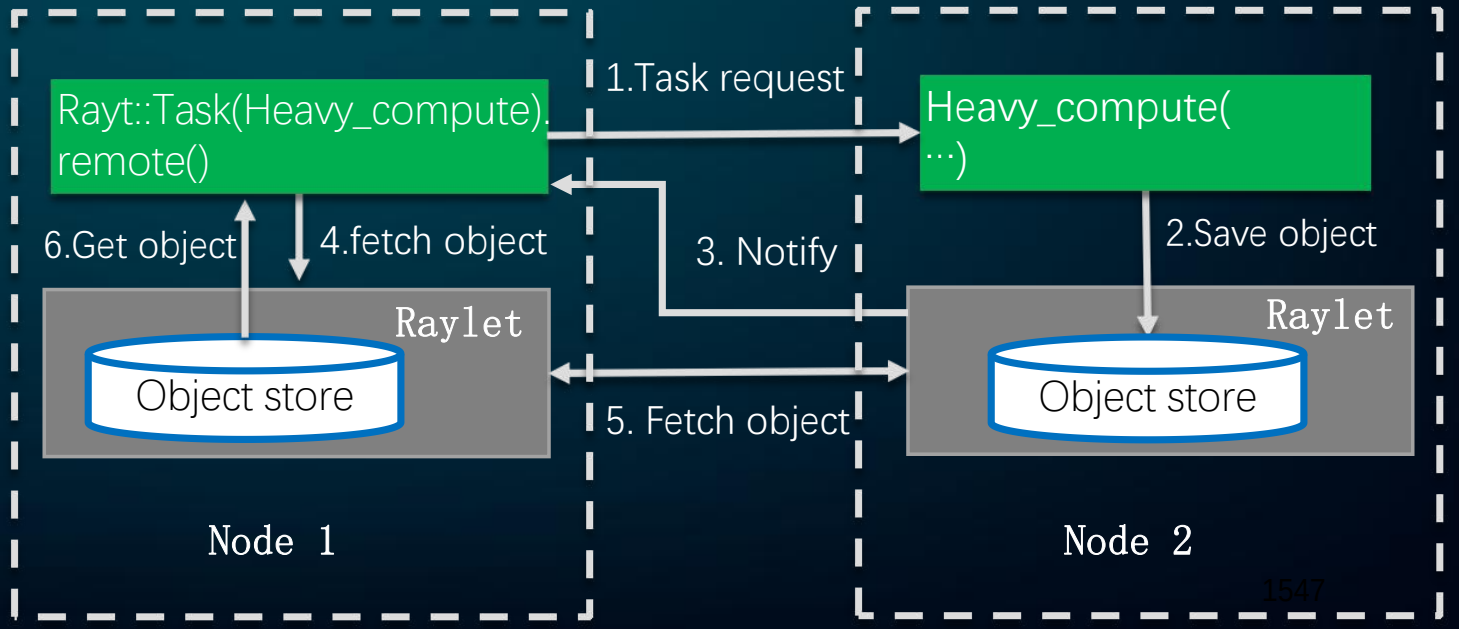
跨节点future —— Ray object

```
1   // object from an ray::Task
2   ray::ObjectRef<int> object =
3       ray::Task(heavy_compute).Remote(1);
4   int result = *object.Get();
```

Object引用

分布式Object store

- 多节点之间Object传输
- 同节点内shared memory
- 自动垃圾回收
- Object spilling



Ray通用分布式编程API——有状态计算单元Actor

单机(class)

```
1  class Counter {
2    int count;
3  public:
4    Counter(int init) {count = init;}
5    int Add(int x) {return x + 1;}
6  }
7
8  Counter *CreateCounter(int init) {
9  return new Counter(init);
10 }
```

分布式(actor)

把普通Class变成remote Actor

```
1  RAY_REMOTE(CreateCounter, &Counter::Add);
2
3  auto actor = ray::Actor(CreateCounter).Remote(0);
4
5  auto result = actor.Task(&Counter::Add).Remote(1);
6
7  EXPECT_EQ(1, *(ray::Get(result)));
```

创建远程Actor对象

调用远程Actor的方法

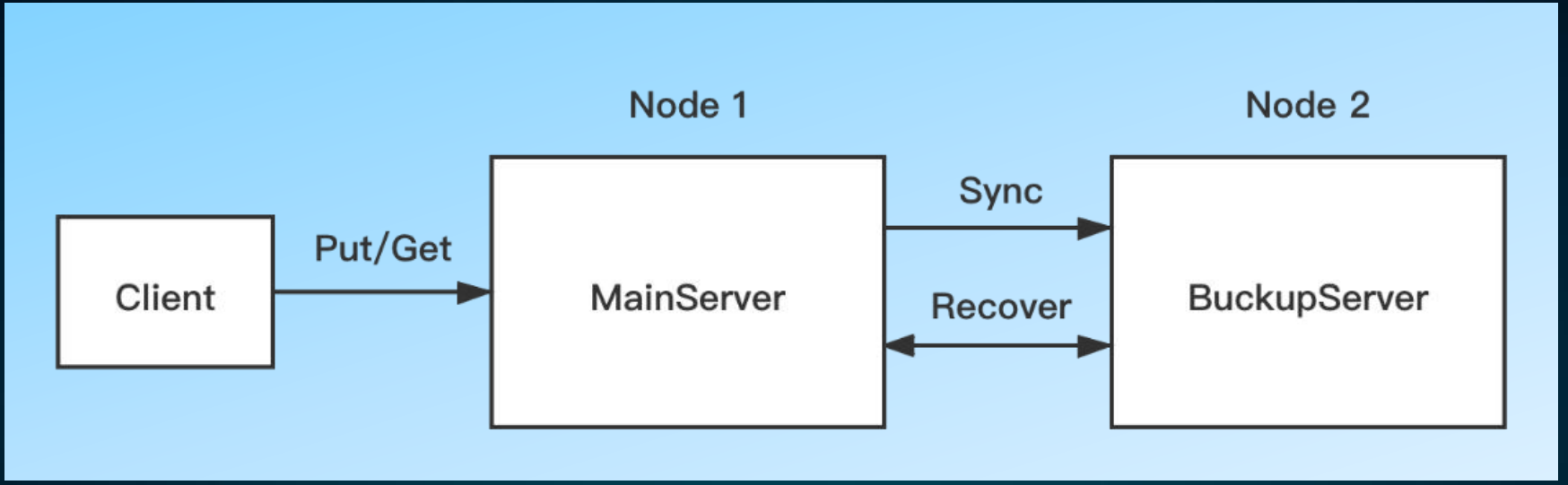
小结

- Ray提供了一套基础的分布式编程API：Task、Actor、Object。
- 基础API可以解决分布式系统中的组件间通信、数据存储和传输等问题。

03

利用Ray
构建分布式系统

Demo说明



- 服务端一个main server一个backup server
- 仅main server提供服务，backup只做备份数据
- main server挂掉后都可以实现自动恢复

demo：200行代码实现一个KV存储系统

main server实现

```
1  class MainServer {
2   public:
3    MainServer();
4
5    void Put(const std::string &key, const std::string &val) {
6        // First synchronize the data to the backup server
7        ...
8
9        // Update the local kv
10       data_[key] = val;
11   }
12
13   std::pair<bool, std::string> Get(const std::string &key) {
14       auto it = data_.find(key);
15       if (it == data_.end()) {
16         return std::pair<bool, std::string>{};
17       }
18       return {true, it->second};
19   }
20
21  private:
22   std::unordered_map<std::string, std::string> data_;
23 };
```

Put接口

Get接口

数据存储在本地内存的map中

backup server实现

SyncData接口同步main的数据

```cpp
class BackupServer {
 public:
  BackupServer();

  // When the master node writes data, it will call SyncData
  // to synchronize the data to this backup node first
  void SyncData(const std::string &key, const std::string &val) {
    data_[key] = val;
  }

  // When the master node restarts, it will call GetAllData
  // for data recovery
  std::unordered_map<std::string, std::string> GetAllData() {
    return data_;
  }

 private:
  std::unordered_map<std::string, std::string> data_;
};
```

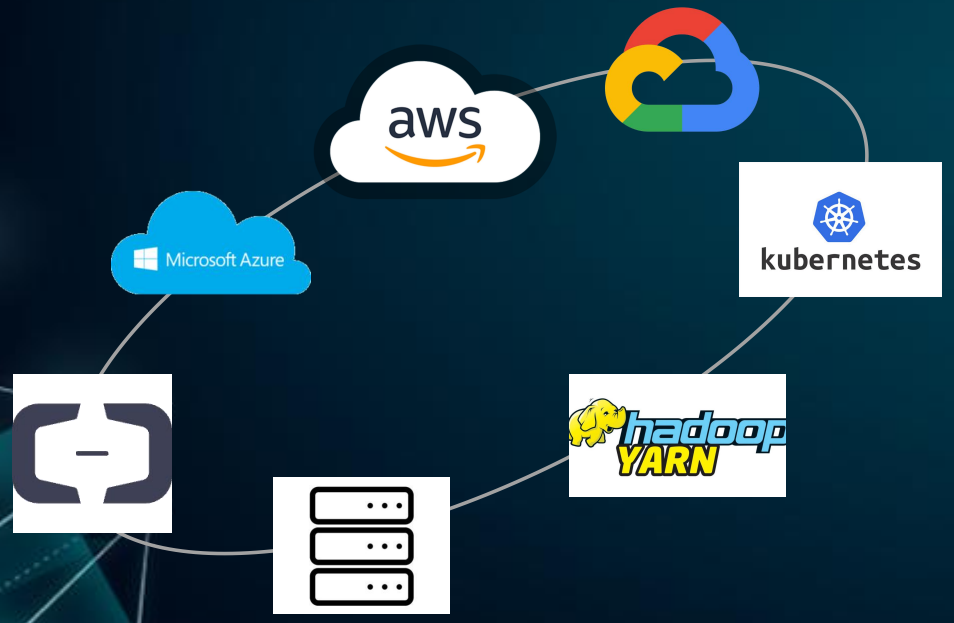
GetAllData接口帮main恢复数据

数据存储在本地内存的map中

部署——Ray集群部署

一键云上部署

```
$ ray up -y config.yaml
```



自定义部署

部署head节点：

```
$ ray start --head
```

部署worker节点：

```
$ ray start –address${HRAD_ADDRESS}
```

部署——server部署

利用ray actor
实现main server
和
backup server的
部署

```
1   // Deploy the actor instances to
2   // the Ray cluster by ray::Actor
3   void StartServer() {
4     ray::Actor(CreateMainServer)
5         .SetName("main_actor")
6         .Remote();
7
8     ray::Actor(CreateBackupServer)
9         .SetName("backup_actor")
10        .Remote();
11  }
```

通过
setName设置
actor name

调度——资源

1个CPU + 16G内存

```
1  const std::unordered_map<std::string, double> RESOUECES{
2    {"CPU", 1.0},
3    {"memory", 16 * 1024.0 * 1024.0 * 1024.0}
4  };
5
6  void StartServer() {
7    ray::Actor(CreateMainServer)
8      ...
9      .SetResources(RESOUECES)
10     .Remote();
11
12   ray::Actor(CreateBackupServer)
13     ...
14     .SetResources(RESOUECES)
15     .Remote();
16 }
```

给actor设置资源

调度——策略

利用PlacementGroup申请两个节点的资源

```
1  std::vector<std::unordered_map<std::string, double>> bundles{
2    RESOUECES, RESOUECES};
3
4  ray::PlacementGroupCreationOptions options{
5    false, name, bundles, ray::PlacementStrategy::SPREAD};
6  auto placement_group = ray::CreatePlacementGroup(options);
7
8  assert(placement_group.Wait(10));
9
10 void StartServer() {
11   ray::Actor(CreateMainServer)
12     ...
13     .SetPlacementGroup(placement_group, 0)
14     .Remote();
15
16   ray::Actor(CreateBackupServer)
17     ...
18     .SetPlacementGroup(placement_group, 1)
19     .Remote();
20 }
```

调度策略为SPREAD（反亲和性）

调度到不同的物理节点

设置运行时环境

设置依赖的libs

```
1   ray::RuntimeEnv runtime_env;
2   runtime_env.Set("native_libraries",
3     std::vector<std::string>{"https://my_bucket/kv_libs.zip"});
4   runtime_env.Set("Container",
5     R"({"image": "ray/ray-test:kv-demo"})");
6
7   void StartServer() {
8     ray::Actor(CreateMainServer)
9       ...
10      .SetRuntimeEnv(runtime_env)
11      .Remote();
12
13    ray::Actor(CreateBackupServer)
14      ...
15      .SetRuntimeEnv(runtime_env)
16      .Remote();
17  }
```

设置容器环境

为actor设置运行时环境

服务发现和组件通信

Client与main server通信

client端通过actor name实现对main server的服务发现

Task通信写数据

```
1   main_actor_ = ray::GetActor<MainServer>("main_actor");
2
3   (*main_actor_).Task(&MainServer::Put).Remote(
4     "first_key", "first_value").Get();
5
6   auto result = *(*main_actor_).Task(
7     &MainServer::Get).Remote("first_key").Get();
```

Task通信读数据

故障恢复——进程

```
1  void StartServer() {
2      ray::Actor(CreateMainServer)
3          ...
4          .SetMaxRestarts(100)
5          .Remote();
6
7      ray::Actor(CreateBackupServer)
8          ...
9          .SetMaxRestarts(100)
10         .Remote();
11 }
```

设置进程自动
重启/重调度次数

故障恢复——状态

Main server被重启后，通过WasCurrentActorRestarted判断是否发生了failover

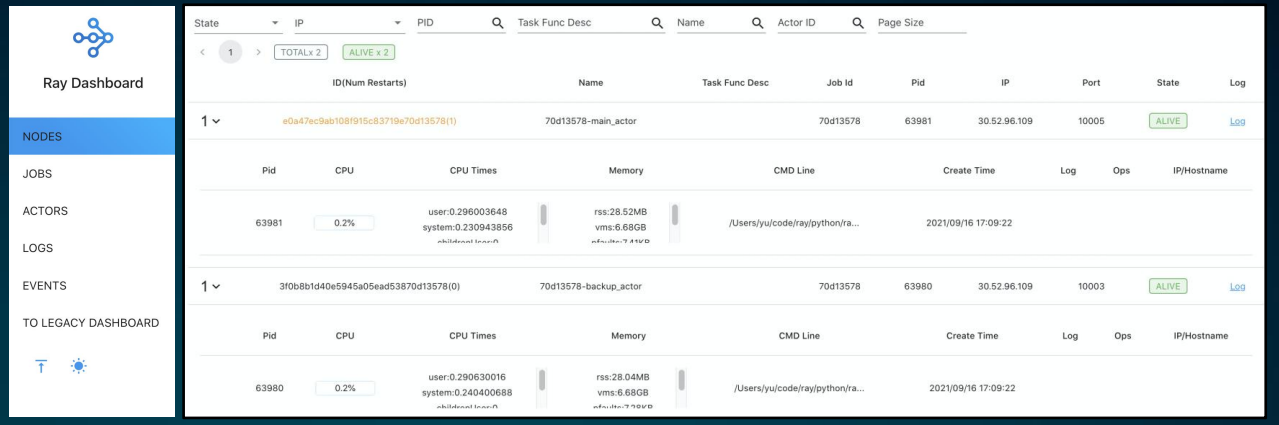
通过actor name对backup server服务发现

```
1  MainServer::MainServer(){
2    if (ray::WasCurrentActorRestarted()) {
3      backup_actor_ = *ray::GetActor<BackupServer>(
4          "backup_actor");
5
6      data_ = *backup_actor_.Task(
7          &BackupServer::GetAllData).Remote().Get();
8    }
9  }
```

通过Task从buckup server恢复数据

运维与监控

- Ray Dashboard
  - Nodes
  - Actors
  - Logs、events
  - profiling
- State Client
- Metrics,Tracing



CLI | Python SDK

```
ray list actors
```

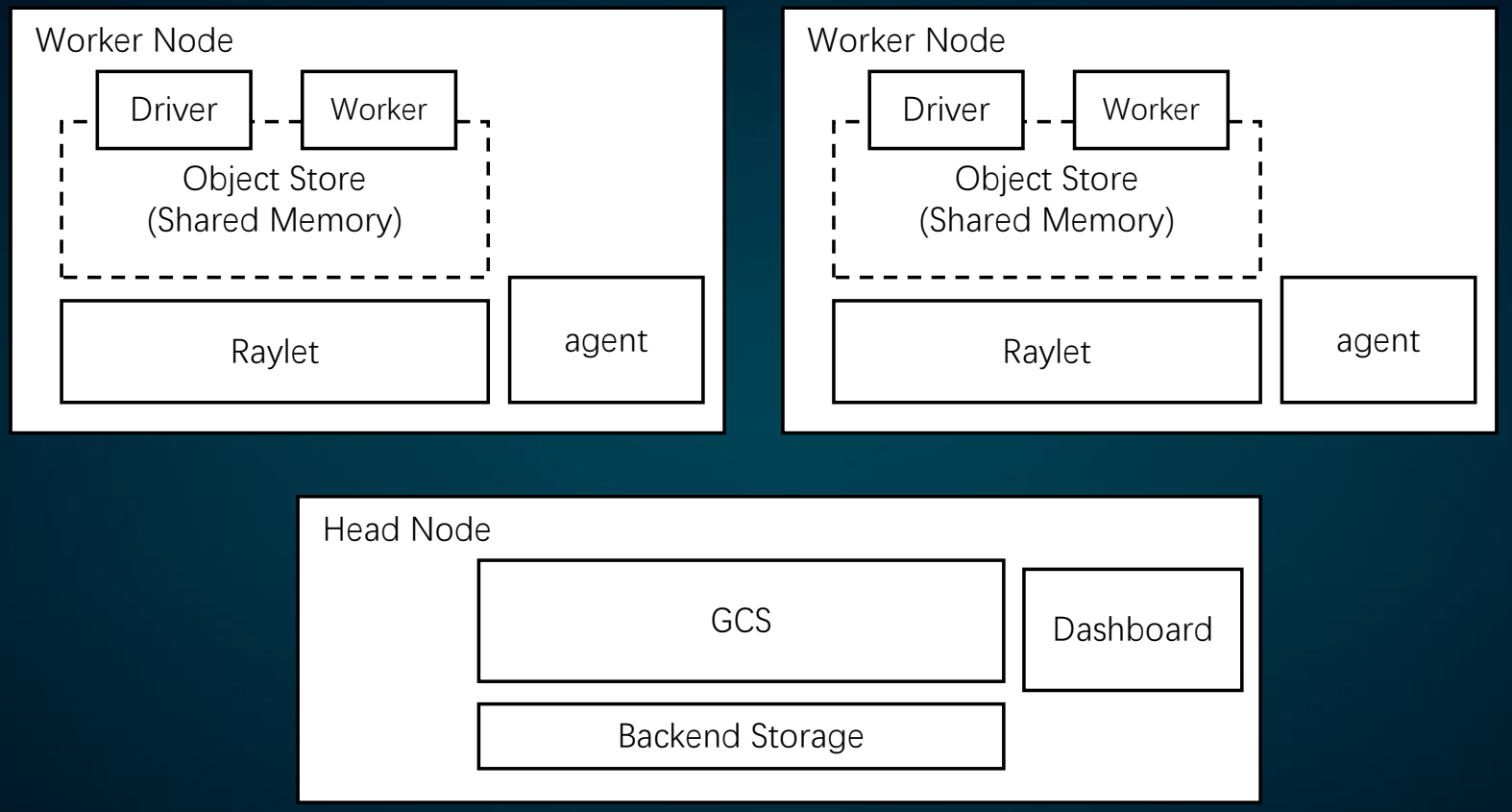
```
======== List: 2022-07-23 21:29:39.323925 ========
Stats:
------------------------------
Total: 2

Table:
------------------------------
    ACTOR_ID                          CLASS_NAME    NAME      PID  STATE
0  31405554844820381c2f0f8501000000  Actor                  96956  ALIVE
1  f36758a9f8871a9ca993b1d201000000  Actor                  96955  ALIVE
```

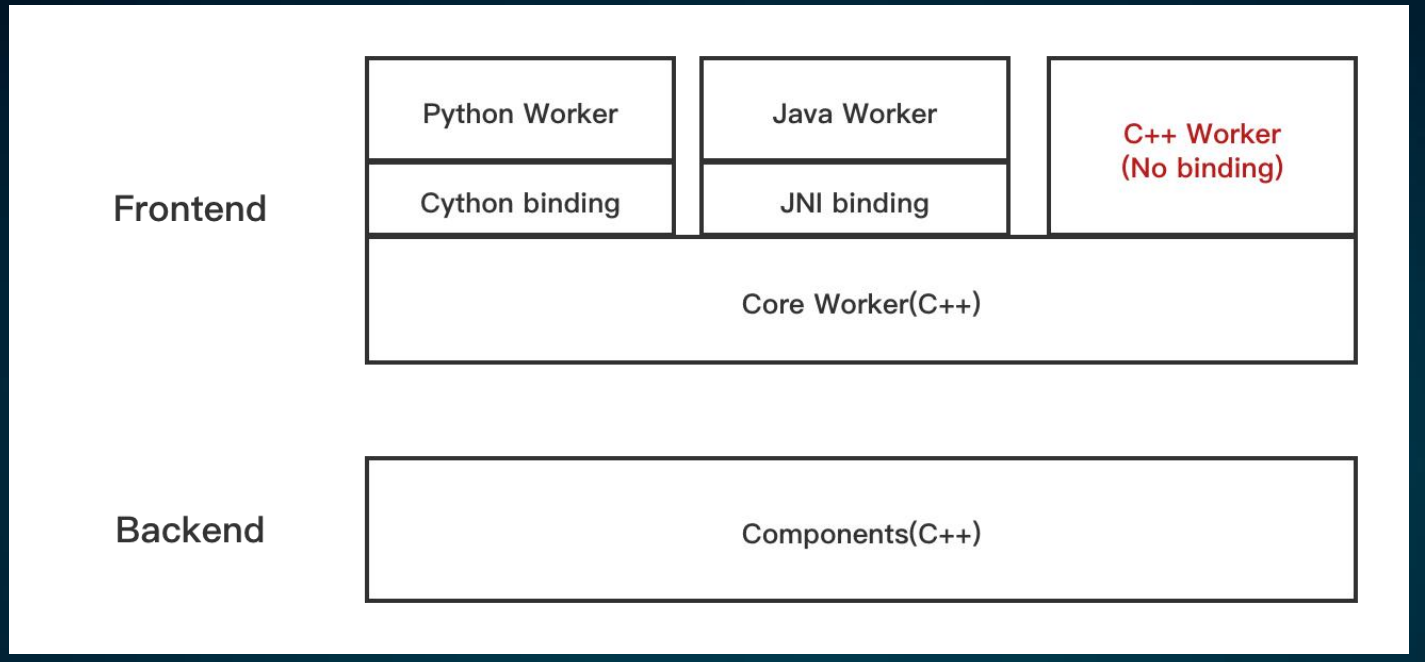
You can get the state of a single task using the get API.



Ray集群


Worker Node
Driver
Worker
Object Store
(Shared Memory)
Raylet
agent
Worker Node
Driver
Worker
Object Store
(Shared Memory)
Raylet
agent
Head Node
GCS
Dashboard
Backend Storage

编程语言



- Backend纯C++
- C++ Core Worker
- Python/Java/C++ Worker

通用分布式编程API：多语言和跨语言

支持Python/Java/C++

```python
@ray.remote
class Counter:
    # ...

counter = Counter.remote()
obj = counter.increment.remote()
res = ray.get(obj)
```

```java
// Java Actor Example
ActorHandle<Counter> actor = Ray.actor(
    Counter::new).remote();
ObjectRef<Integer> obj = actor.task(
    Counter::increment).remote();
int res = Ray.get(obj);
```

跨语言调用Task/Actor

```python
actor = ray.cross_language.cpp_actor_class(
    "CreateCounter", "Counter"
).remote()

obj = actor.increment.remote()
assert 1 == ray.get(obj)
```

小结

- 利用Ray的通用分布式编程API，构建了一个完整的分布式KV store
- 基于Ray的集群和Actor实现了部署
- 基于Ray的Task实现了组件通信
- 基于Ray的调度实现了定制资源和反亲和性
- 基于Ray的Runtime Env解决了环境依赖问题
- 基于Ray的named actor实现了服务发现
- 基于Ray的restart能力实现了进程和状态的故障恢复
- 基于Ray的Dashboard和State API获取了基本的运维与监控能力

完整Demo地址 https://github.com/SongGuyang/ray-cpp-example

04

技术内幕

Ray C++框架实现的核心挑战

无运行时反射

Python &Java worker依靠运行时反射实现远程调用，C++语言特性导致实现难度系数增加



无实现先例

现有计算引擎普遍无C++ API

Ray API更low level

序列化

如何给用户更友好的序列化体验

更高质量

极致性能

Zero copy等

Ray C++框架实现

调用端

执行端

Java

```
Class Demo {
        public static String
        heavyCompute(int value) {
          return value;
        }
}
Ray.Task(Demo:: heavyCompute, 1);
```

远程方法描述
className
functionName
signature

args(binary)

demo-SNAPSHOT.jar

Reflect
Deserialize

classloader

```
Method method;
Object[] args;

method().invoke(null, args);
```

C++

```
class Demo {
        public:
          static string
          heavyCompute(int value) {
            return value;
          }
}
Ray:: Task(Demo::heavyCompute, 1);
```

className
functionName
signature

args(binary)

demo.so

Deserialize

dlopen(demo.so)

```
typedef int(* Function)(int);
Function func = dlsym( handle,
“heavyCompute”);

func(…);
```

如何找到 function?
Function如何executable?

Ray C++框架实现

核心问题：

- function寻址
- function可执行



RAY_REMOTE(heavy_compute)

Ray C++框架实现——RAY_REMOTE

```
1   #define CONCATENATE(s1, s2) s1##s2
2   #define ANONYMOUS_VARIABLE(str) CONCATENATE(str, __LINE__)
3
4   #define RAY_REMOTE(...)                    \
5     static auto ANONYMOUS_VARIABLE(var) = \
6       ray::internal::RegisterRemoteFunctions(#__VA_ARGS__, __VA_ARGS__);
```

全局变量

程序加载过程中注册remote函数

Ray C++框架实现——RAY_REMOTE

在Task调用端和执行端分别生成一对双向map：

```
1   template <typename Function>
2   RegisterRemoteFunction(std::string const &name, const Function &f) {
3     func_ptr_to_name_map_.emplace(
4         GetAddress(f), name);
5     name_to_executable_func_map_.emplace(name,
6       std::bind(TemplateFunc<Function>, std::move(f), std::placeholders::_1));
7   }
```

正向map

反向map

编译期生成可执行函数，固定参数列表

```
1   using ExecutableFunc = std::function<msgpack::sbuffer(const ArgsBufferList &)>;
2
3   std::unordered_map<std::string, std::string> func_ptr_to_name_map_;
4   std::unordered_map<std::string, ExecutableFunc> name_to_executable_func_map_;
```

Ray C++框架实现——RAY_REMOTE

ExecutableFunc 生成细节：



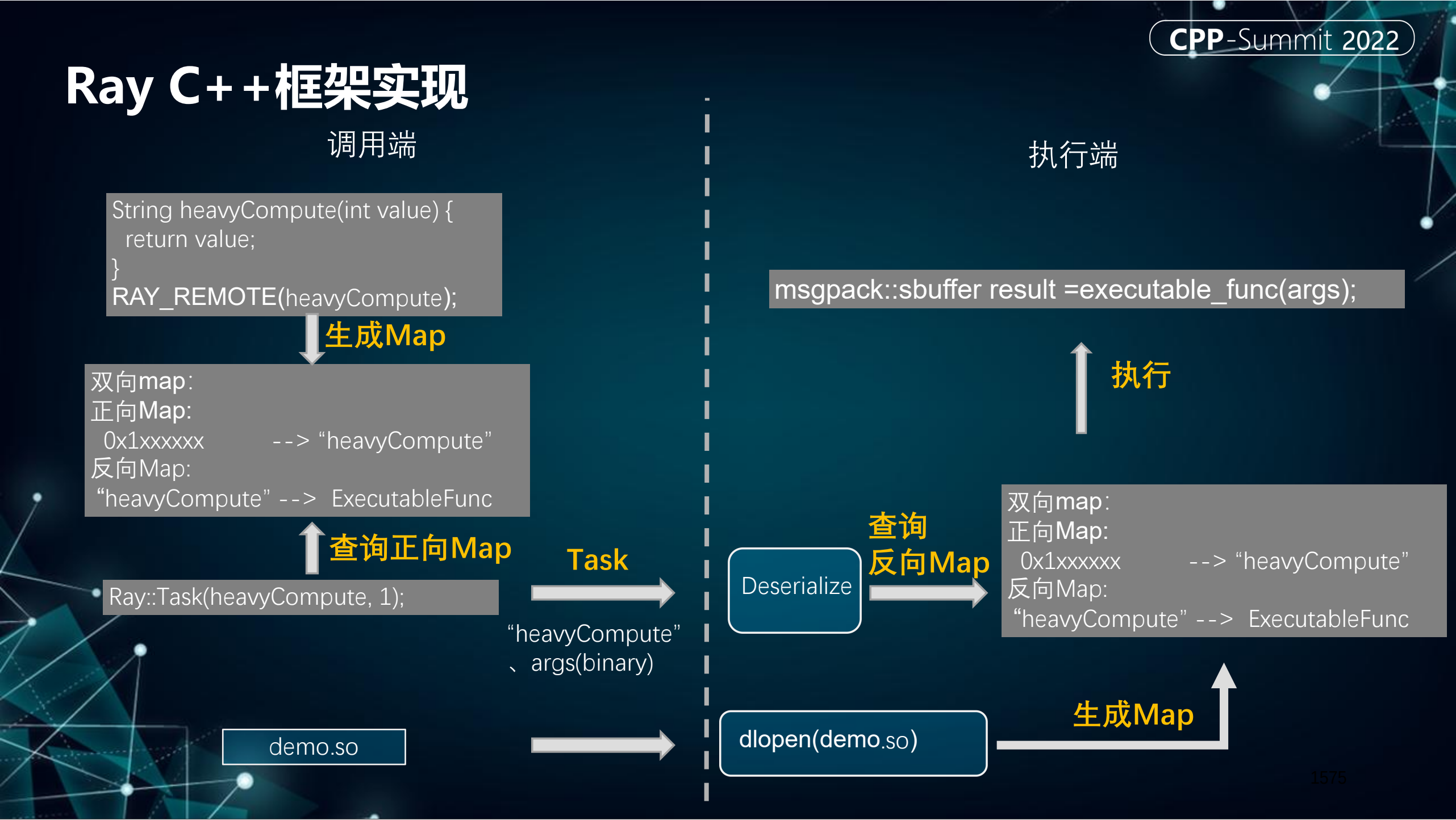

```cpp
template <typename Function>
msgpack::sbuffer TemplateFunc(const Function &func,
                              const ArgsBufferList &args_buffer) {
    using RetrunType = boost::callable_traits::return_type_t<Function>;
    using ArgsTuple = boost::callable_traits::args_t<Function>;
    ArgsTuple tp{};
    bool is_ok = GetArgsTuple(
        tp,
        args_buffer,
        std::make_index_sequence<std::tuple_size<ArgsTuple>::value>{});

    ...

    return Invoker<Function>::Call<RetrunType>(func, std::move(tp));
  }
```

```cpp
using ExecutableFunc = std::function<msgpack::sbuffer(const ArgsBufferList &)>;
```

Ray C++框架实现


调用端
执行端
String heavyCompute(int value) {
return value;
}
RAY_REMOTE(heavyCompute);
生成Map
双向map:
正向Map:
0x1xxxxxx --> “heavyCompute”
反向Map:
“heavyCompute” --> ExecutableFunc
查询正向Map
Ray::Task(heavyCompute, 1);
Task
“heavyCompute”
、args(binary)
demo.so
Deserialize
查询
反向Map
双向map:
正向Map:
0x1xxxxxx --> “heavyCompute”
反向Map:
“heavyCompute” --> ExecutableFunc
dlopen(demo.so)
生成Map
msgpack::sbuffer result =executable_func(args);
执行

05

Ray在蚂蚁的应用

为什么用Ray

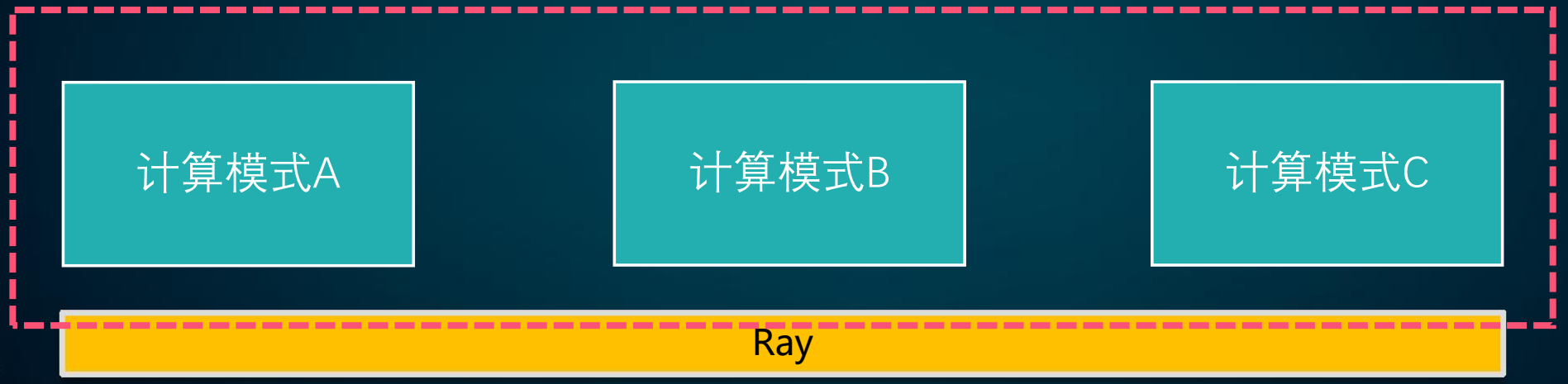
- 通用Ray分布式编程API
- 可以支持任何计算模式



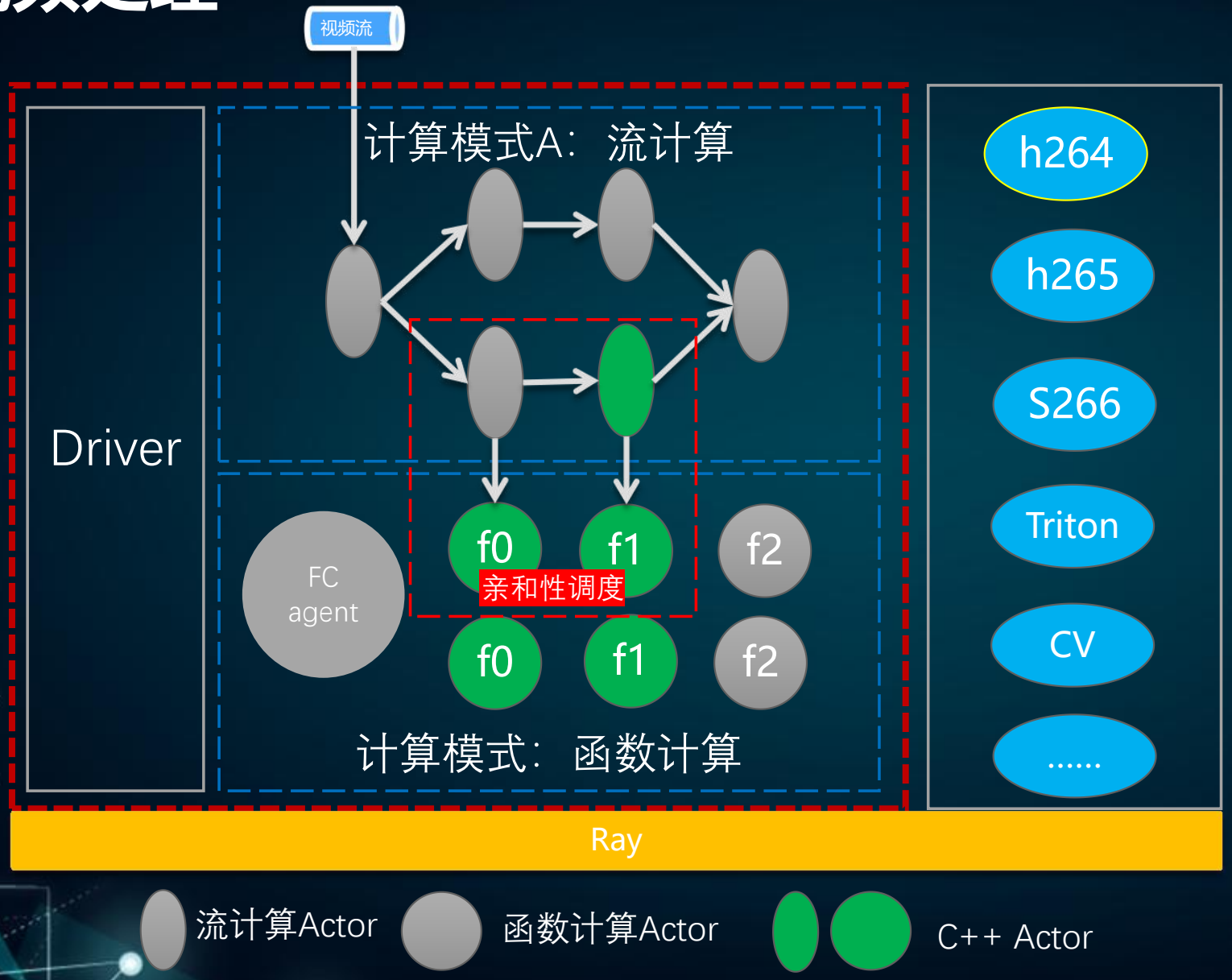
统一分布式底盘



融合计算

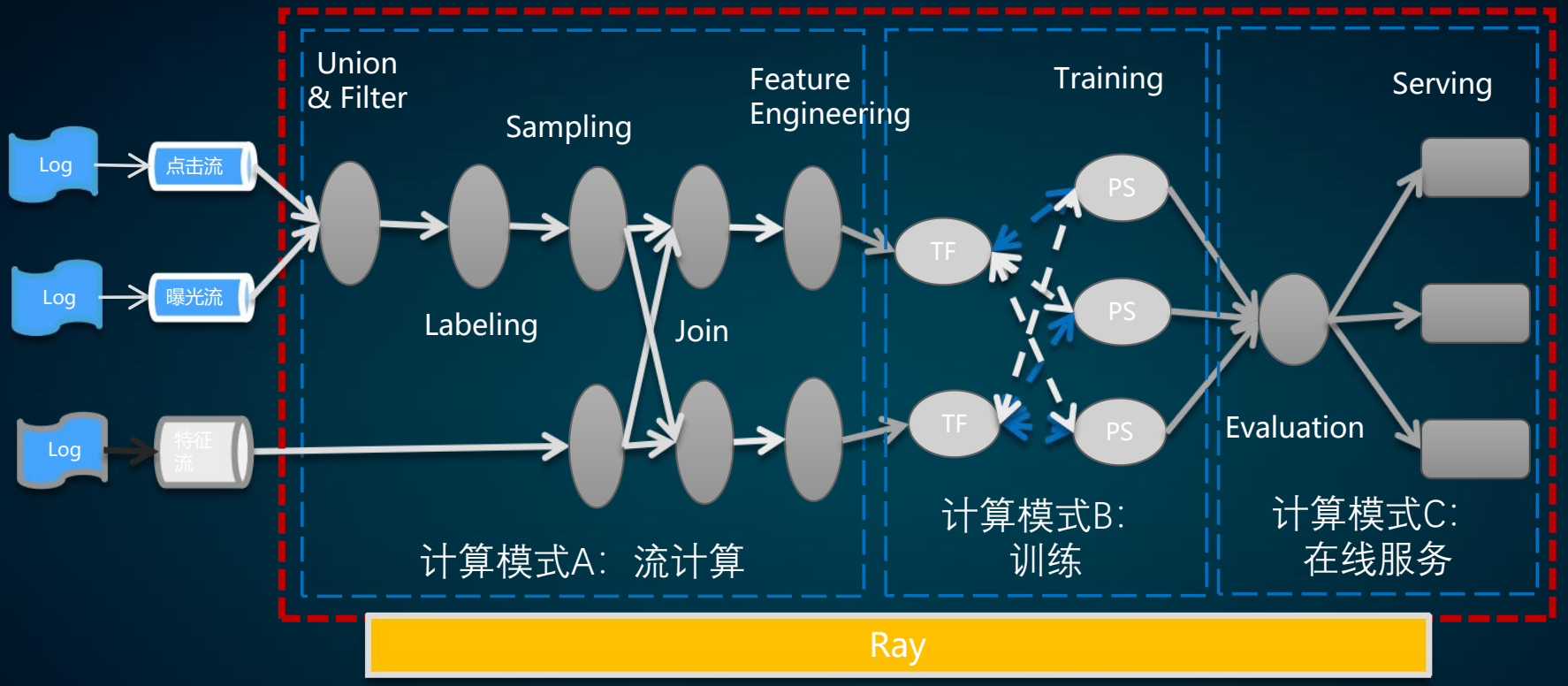


音视频处理



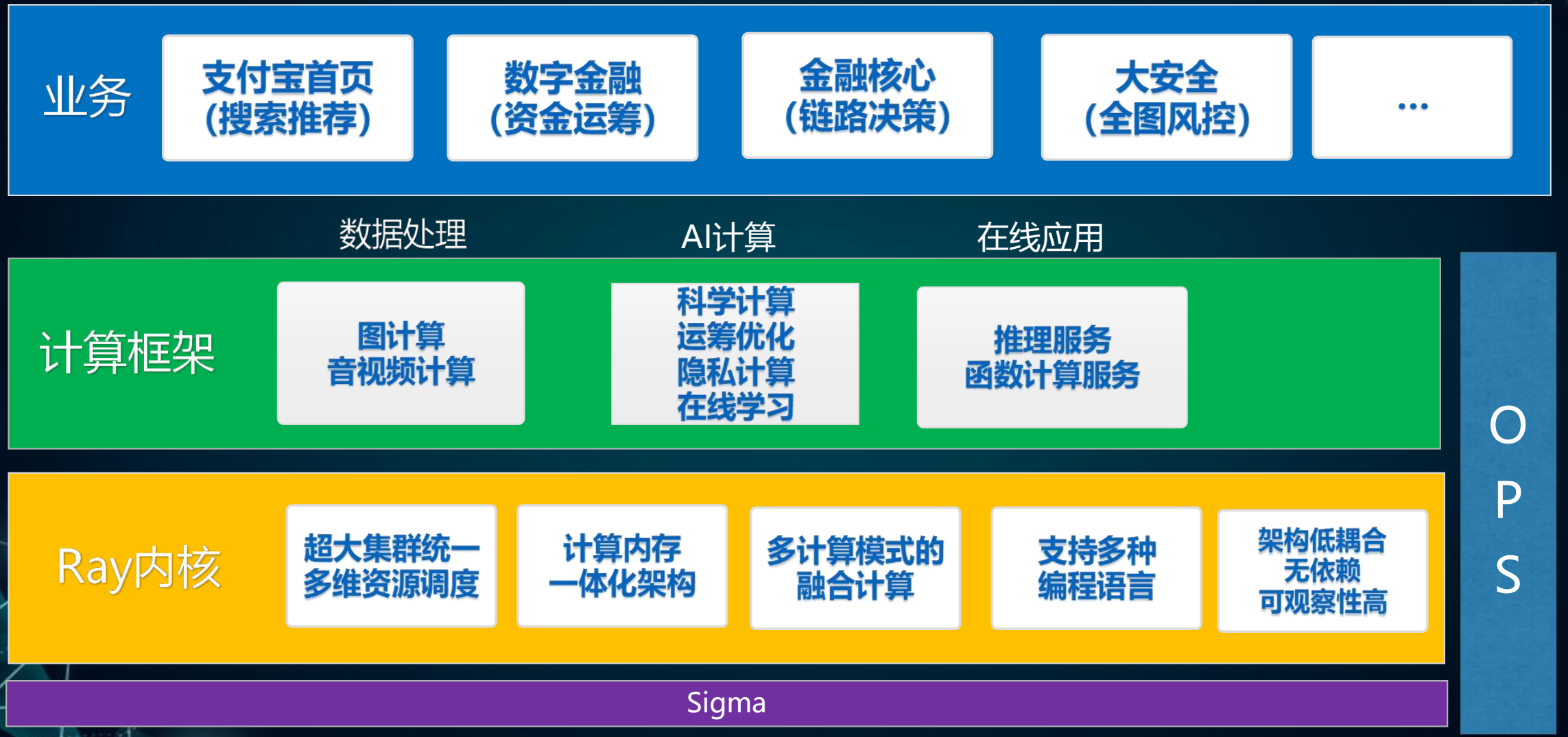
- 一个作业
- Collocation调度 + Shared Memory提高数据传输效率
- 全链路动态扩缩容。
- C++/Java跨语言。

在线学习



- 一个作业涵盖整个在线学习链路。
- Collocation调度 + Shared Memory提高数据传输效率。
- 一体化Checkpoint，联动Failover。
- 全链路数据流反压&动态扩缩容。
- Python/Java/C++跨语言。

蚂蚁集团新计算架构



超过 40w core 的规模，同时支持：南网、阿里集团、达摩院、菜鸟、网商、外部企业等。

06

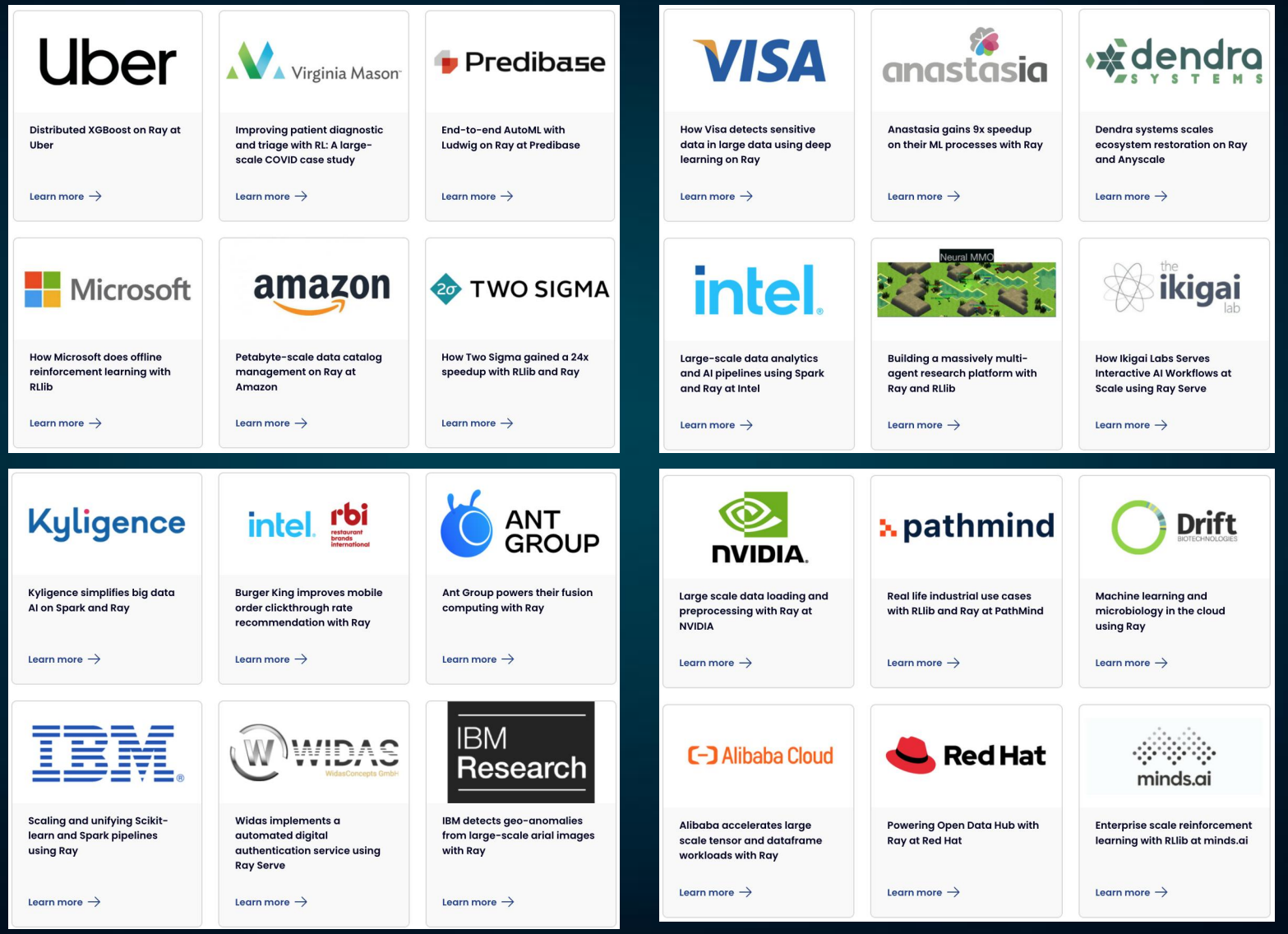
Ray开源现状

活跃度

https://github.com/ray-project/ray 17年开源至今



企业应用



www.anyscale.com/user-stories

Ray forward 2022



主办方 | RAY中文社区　　合作方 | 示说

Ray Forward Meetup 2022

共创Ray中文社区

活动时间：

2022年8月6日下午1:30-6:00

活动地点：

线上: 示说网直播 Ray中文社区视频号直播

Ray Forward Meetup已经举办了四年，今年主要联合国内各个企业级应用的友商共建Ray中文社区，共同推进Ray在国内的企业级规模化落地，打造具有特色的Ray中文社区生态。

会议议程

1:30-2:05 **Ray 2.0前瞻**

在今年的RaySummit上，我们准备发布全新的Ray2.0。这个版本在性能,易用性,和可靠性上都有很重要的突破。新的特性包括RayAI Runtime（AIR）, KubeRay,高性能shuffle,以及其他很多重要改进。

张喆 Anyscale 开源团队负责人

2:05-2:40 **Ray在蚂蚁的规模化落地介绍**

这次分享，我会介绍过去四年，蚂蚁集团为什么基于Ray构建的分布式计算、AI系统，以及这些计算、AI引擎在蚂蚁落地的情况。同时我会介绍蚂蚁在Ray 开源社区的贡献。

若夷 蚂蚁集团-资深技术专家

2:40-3:15 **Ray在分布式训练容错和数据缓存上的实践**

深度学习模型训练周期长，消耗计算资源多；分布式训练的稳定性和耗时性能是模型迭代的关键指标。我们利用ray actor的状态持有特性，监控用户训练状态和计算节点GPU健康状态，自动伸缩训练任务至健康的计算节点，提高训练过程的容错性。为了解决分布式训练数据加载的性能瓶颈，我们利用ray object缓存训练所需的数据集，并且利用数据分布的局部性特征优化数据加载的务分发，在生产应用中获得了预期的性能收益。

邱爽 大疆创新-高性能计算高级工程师

3:30-4:05 **华为云Fathom数据科学智能化框架**

数据和AI正在成为现代企业数智化转型的核心，可是如何利用海量数据与AI的澎湃算力帮助业务实现快速智能驱动的建模与洞察，帮助核心业务实现数据与AI双轮驱动的新增长成为了一个技术难题。华为云Fathom数据科学智能化框架，结合华为多年数字化治理经验与Ray社区丰富的分布式计算能力，实现AI与数据无缝流转与融合，帮助用户快速上手基于数据与AI的智能计算，通过提供稳固高效的分布式底座，赋能金融、制造、交通等领域企业级用户。本议题将介绍华为云Fathom的设计与业界思考，展示在重复利用Ray的基础上如何提供易用的大规模分布式数据科学智能化体验。

曹宇 华为云高级工程师 英国爱丁堡大学博士后

4:05 -4:40 **Veloce：基于Ray的异构神经网络训练的低代码工具库**

该工具库可以凭借 Ray 即时地、低延迟地在笔记本或云上启动分布式工作负载，并自动将用户提供的单机模型转换为数据并行的多机模型；该库还可以在数据稀疏性的定义下，提供灵活的异构计算策略执行复杂的训练模式。

翟晓宇 奇虎360-机器学习资深工程师

4:40-5:15 **IDP：基于Ray的AI开发生产平台**

AI平台对于企业的核心价值在于降本增效，低成本加速企业AI创新。实现降本增效的核心之一是高性能的任务调度和计算，提升资源利用率。白海科技新一代AI开发生产平台IDP，基于Ray的底层分布式计算框架来构建高性能的任务调度引擎，实现task级别精细化任务调度。本次分享将主要介绍：IDP选择基于Ray来自研任务调度引擎的原因，IDP任务调度引擎的具体设计与实现，以及使用Ray构建AI平台过程中的经验总结等。

卢亿雷 白海科技创始人兼CEO

5:15-5:50 **Ray 助力 Byzer 实现分布式 Python Runtime**

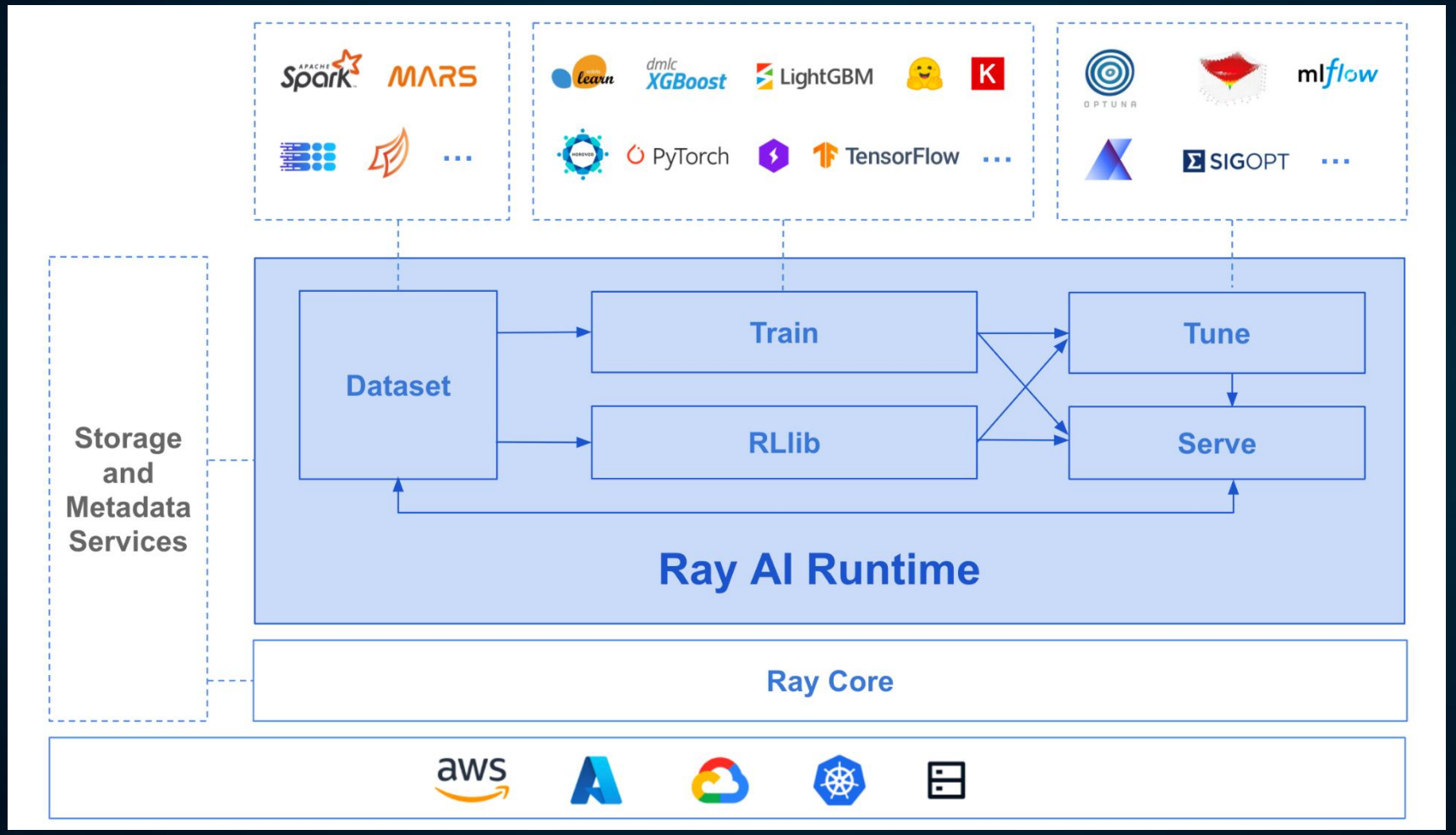
这次分享，我们会介绍 Byzer 如何打通 Spark 和 Ray 之间的数据通路，在 SQL 和 Python 之间实现数据协作，同时介绍相比 PySpark 的实现的一些优缺点。现场我们会演示用户如何通过 Byzer 完成利用 SQL 抽取各种数据源的数据，然后转化为 分布式 Pandas 数据集合(Dask on Ray)进行操作，最后重新返回成表继续使用SQL处理的流程。本文全程使用第三代 Notebook 产品, Byzer Notebook 来演示。

祝威廉 Byzer社区PMC
资深数据架构师
Kyligence技术合伙人

生态系统

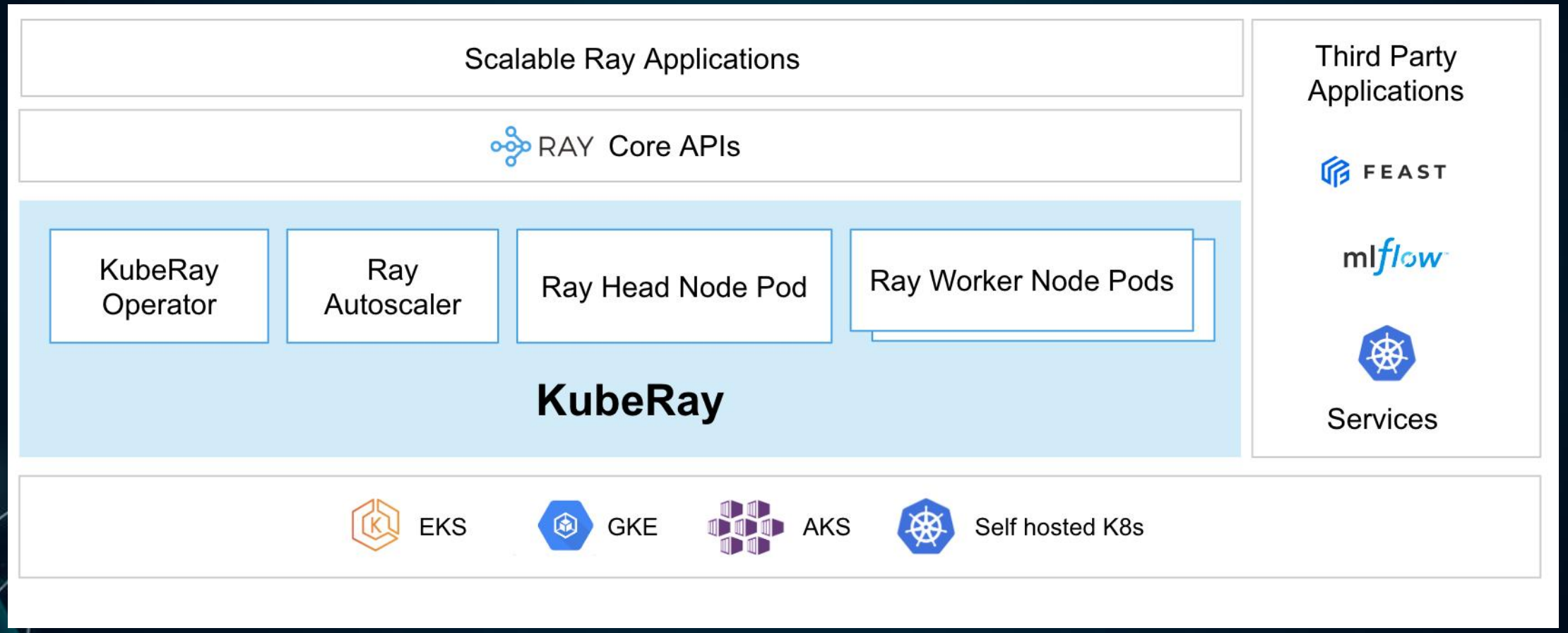

Native Libraries
Third Party Libraries
rllib
tune
RAY SERVE
raysgd
HOROVOD
PyTorch
Azure Machine Learning
Weights & Biases
spaCy
Amazon SageMaker
ANALYTICS ZOO
Joblib
HYPEROPT
OPTUNA
DASK
ModelArts
SELDON
MODIN
MARS
RAY

Ray 2.0——Ray AIR



一个可扩展的、统一的机器学习工具集。

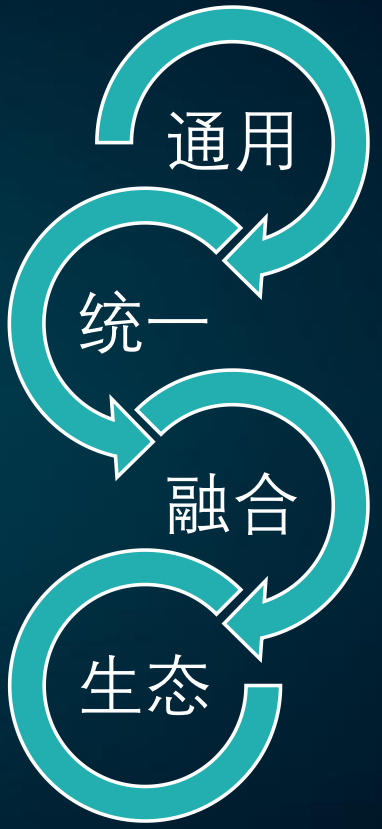
Ray 2.0——KubeRay



Ray on Kubenetes 工具集。

总结

- Ray提供一套**通用**的分布式编程API
- Ray可以作为**统一**的分布式计算底盘
- 利用Ray可以**融合**不同的计算模式
- Ray有着丰富的开源**生态**并快速发展



CPP-Summit 2022



左春伟

美团基础研发平台-数据平台架构师

美团-基础研发平台-数据平台架构师（研究员），前华为20级软件教练、淘宝高级技术专家、微信高级工程师，拥有超十年的C++开发和架构经验。2008年北京邮电大学硕士毕业后加入搜狐从事MMORPG游戏后端开发，之后进入互联网，参与过搜索、直播服务器、机器学习等领域和方向的研发，目前专注于高性能计算和分布式系统架构，在系统开发和性能优化方面有着丰富的经验。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

左春伟

美团数据平台
架构师

性能优化的路有多远?

议程

1 **概述**
微架构 | 存储层次 | 工具 | 方法

2 **宏观架构**
架构决定性能上限

3 **关键组件**
内存池 | 0-Copy | 缓存

4 **实现细节**
细节决定性能极限

01

概述

何为性能优化?

不影响正确性的前提下，使程序运行得更快、
吞吐更大、延迟更低、并发更高、服务能力更强...

不同程序有不同的性能关注点

- 科学计算关注运算速度
- 射击游戏时延敏感
- 图形引擎注重渲染效率
- 服务器关注吞吐能力

01

概述



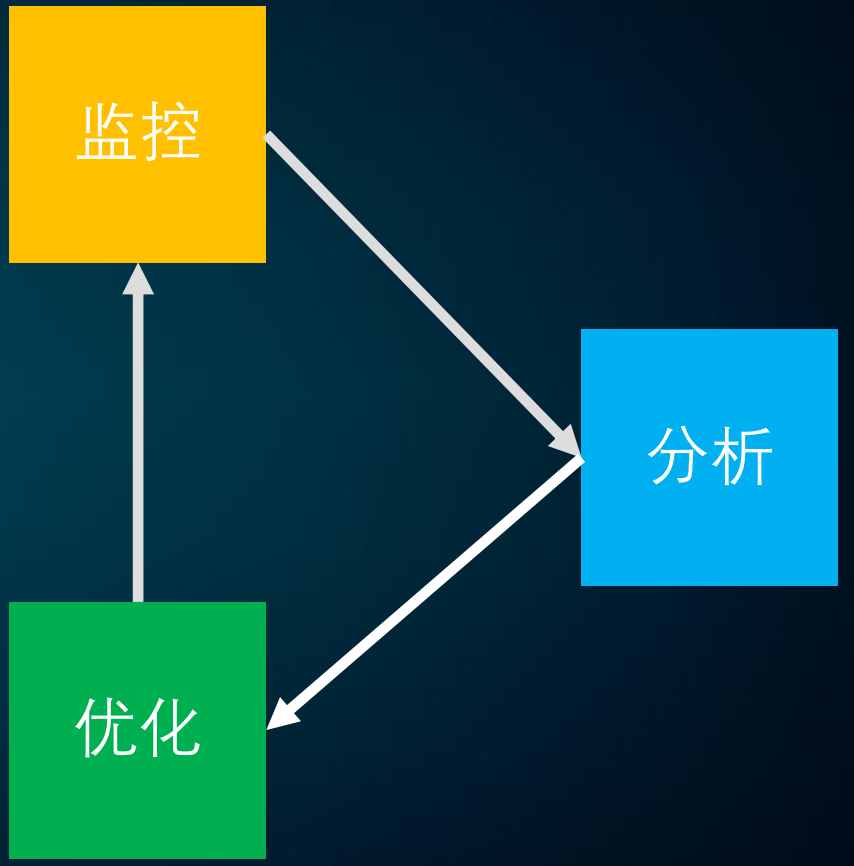
01

概述 监控先行

建立度量：你度量什么，你得到什么！

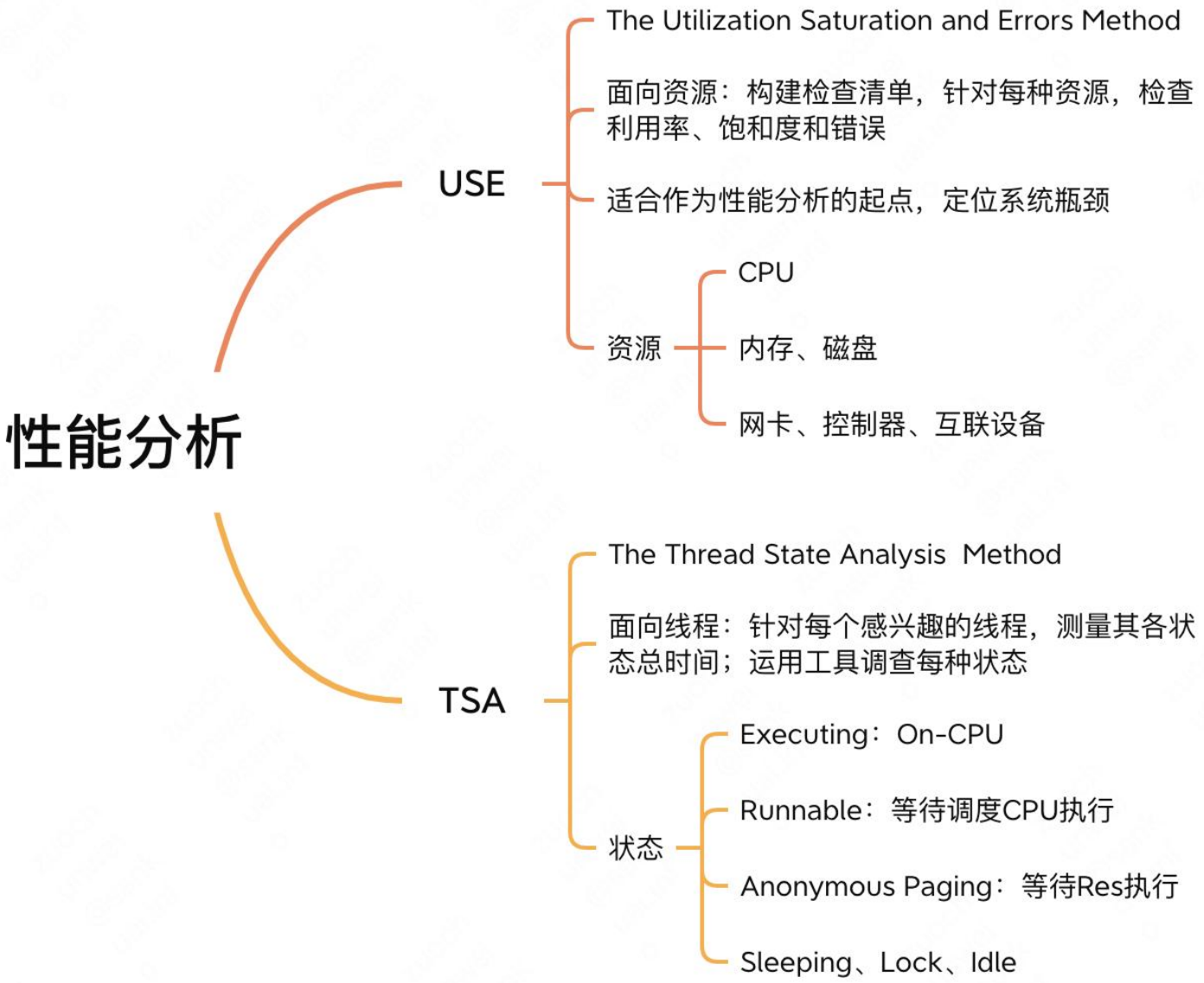
基本原则：依赖数据而非凭空假设

搭建真实的压测环境，或者逼近真实环境，有时候是困难的，也可能非常耗费时间，但它依然是值得的



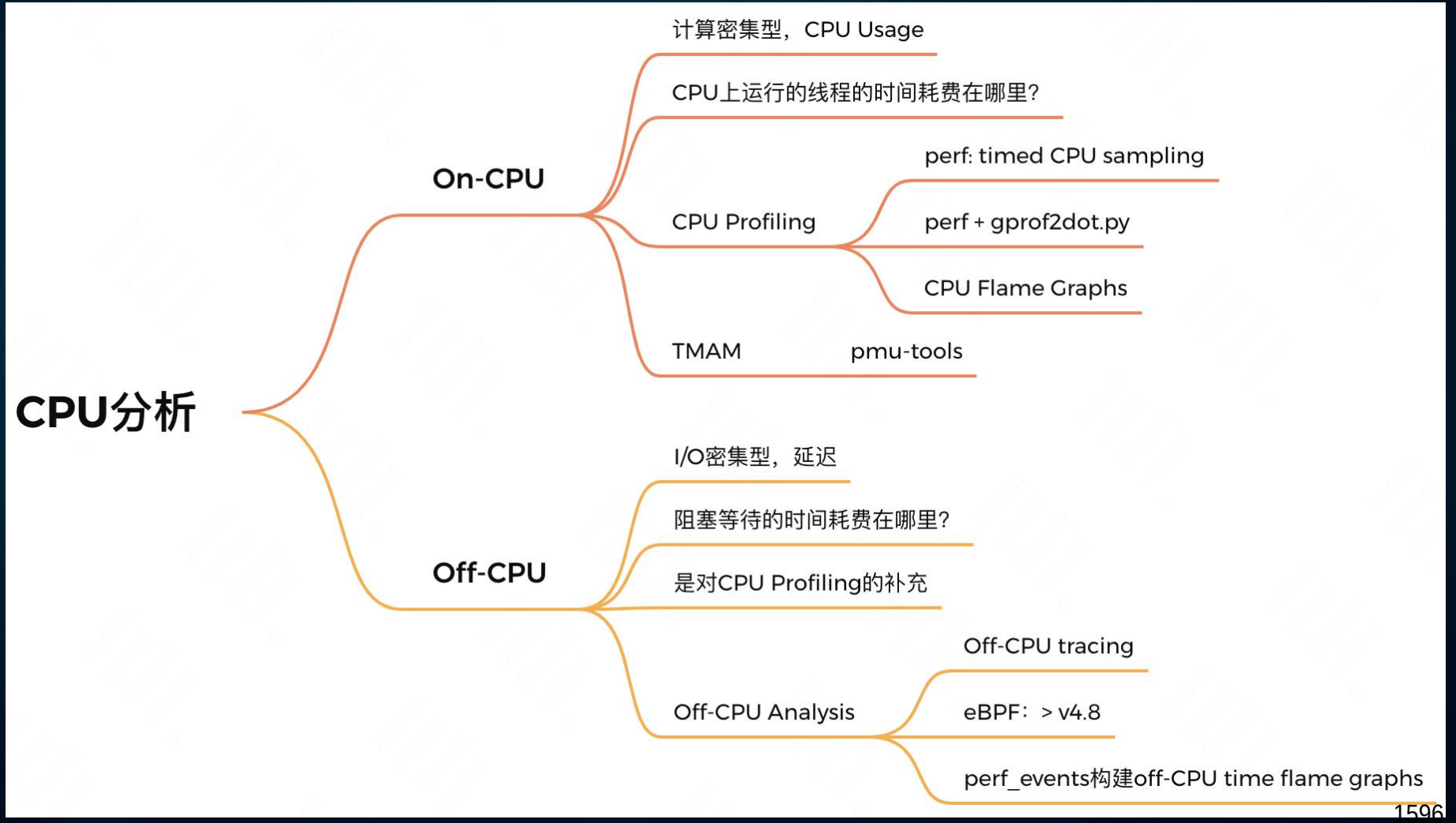
01

概述


性能分析
USE
The Utilization Saturation and Errors Method
面向资源：构建检查清单，针对每种资源，检查利用率、饱和度和错误
适合作为性能分析的起点，定位系统瓶颈
资源
CPU
内存、磁盘
网卡、控制器、互联设备
TSA
The Thread State Analysis Method
面向线程：针对每个感兴趣的线程，测量其各状态总时间；运用工具调查每种状态
状态
Executing：On-CPU
Runnable：等待调度CPU执行
Anonymous Paging：等待Res执行
Sleeping、Lock、Idle

01

概述


CPU分析
On-CPU
计算密集型，CPU Usage
CPU上运行的线程的时间耗费在哪里?
CPU Profiling
perf: timed CPU sampling
perf + gprof2dot.py
CPU Flame Graphs
TMAM
pmu-tools
Off-CPU
I/O密集型，延迟
阻塞等待的时间耗费在哪里?
是对CPU Profiling的补充
Off-CPU Analysis
Off-CPU tracing
eBPF：> v4.8
perf_events构建off-CPU time flame graphs

01 perf：查找函数/指令CPU热点

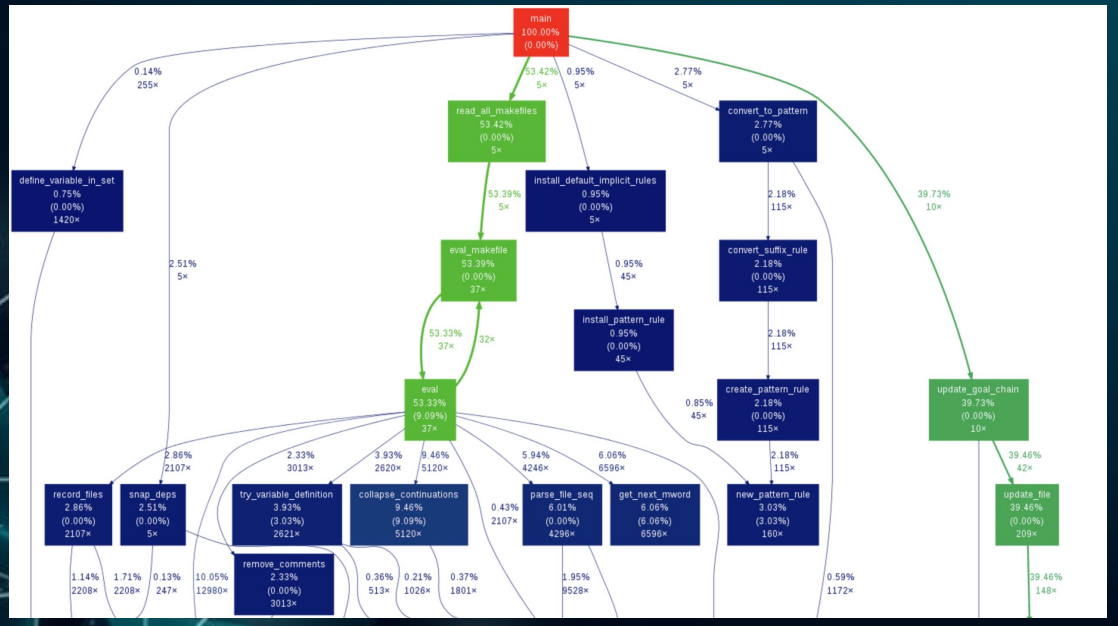
概述 工具

perf + gprof2dot.py

```
usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

The most commonly used perf commands are:
  annotate        Read perf.data (created by perf record) and display annotated code
  archive         Create archive with object files with build-ids found in perf.data file
  bench           General framework for benchmark suites
  buildid-cache   Manage build-id cache.
  buildid-list    List the buildids in a perf.data file
  c2c             Shared Data C2C/HITM Analyzer.
  config          Get and set variables in a configuration file.
  data            Data file related processing
  diff            Read perf.data files and display the differential profile
  evlist          List the event names in a perf.data file
  ftrace          simple wrapper for kernel's ftrace functionality
  inject          Filter to augment the events stream with additional information
  kallsyms        Searches running kernel for symbols
  kmem            Tool to trace/measure kernel memory properties
  kvm             Tool to trace/measure kvm guest os
  list            List all symbolic event types
  lock            Analyze lock events
  mem             Profile memory accesses
  record          Run a command and record its profile into perf.data
  report          Read perf.data (created by perf record) and display the profile
  sched           Tool to trace/measure scheduler properties (latencies)
  script          Read perf.data (created by perf record) and display trace output
  stat            Run a command and gather performance counter statistics
  test            Runs sanity tests.
  timechart       Tool to visualize total system behavior during a workload
  top             System profiling tool.
  version         display the version of perf binary
  probe           Define new dynamic tracepoints
  trace           strace inspired tool

See 'perf help COMMAND' for more information on a specific command.
```



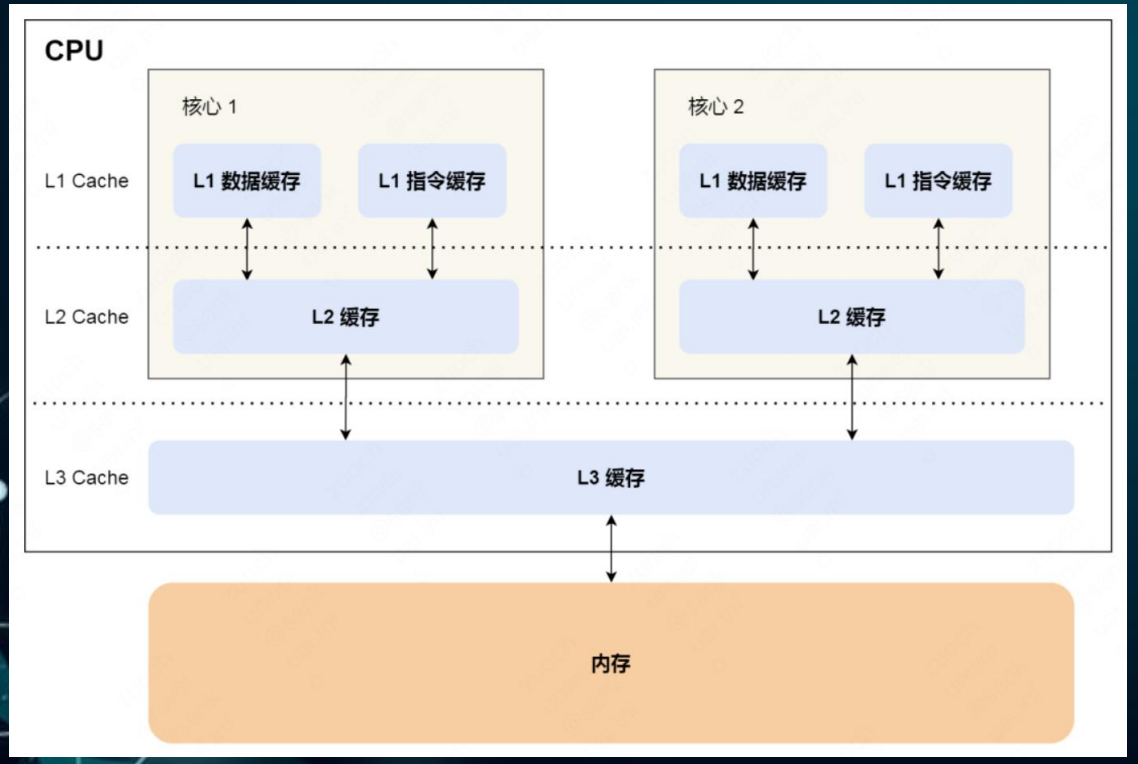

```
[root@ ~]# perf list

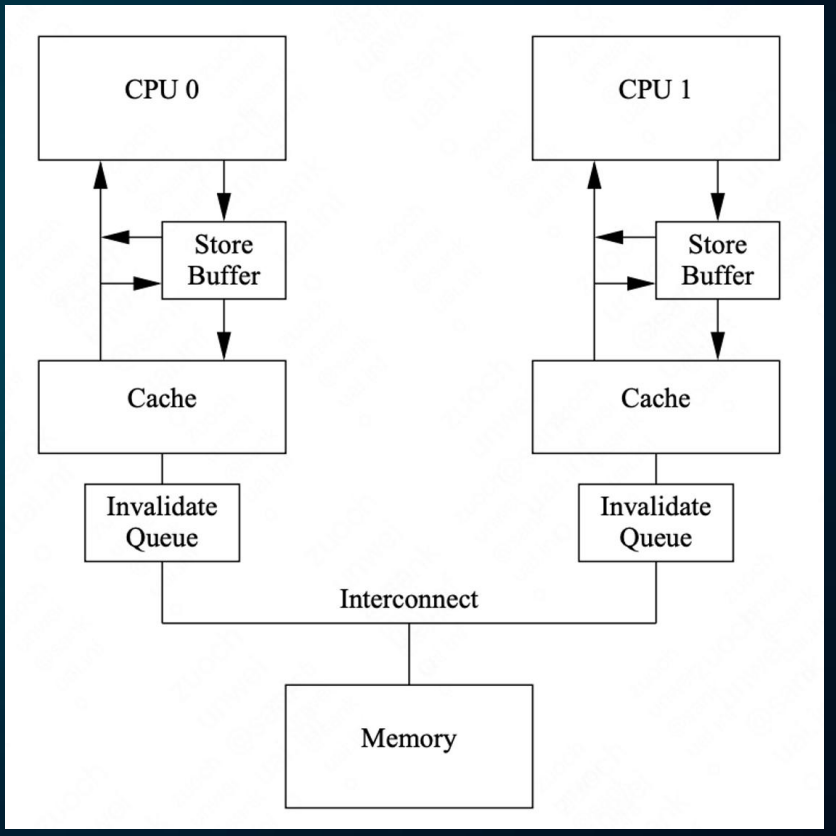
List of pre-defined events (to be used in -e):

  alignment-faults                                   [Software event]
  bpf-output                                         [Software event]
  context-switches OR cs                             [Software event]
  cpu-clock                                          [Software event]
  cpu-migrations OR migrations                       [Software event]
  dummy                                              [Software event]
  emulation-faults                                   [Software event]
  major-faults                                       [Software event]
  minor-faults                                       [Software event]
  page-faults OR faults                              [Software event]
  task-clock                                         [Software event]
```

01 从CPU微架构视角审视程序瓶颈

概述 微架构



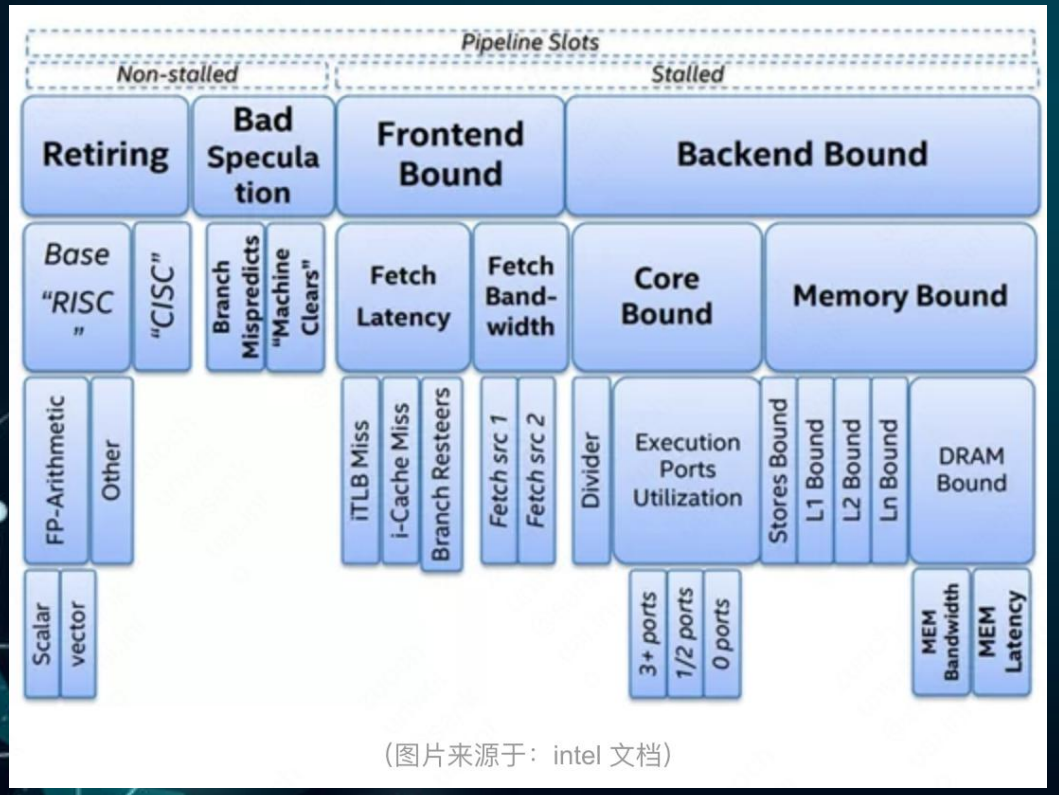


https://www.scss.tcd.ie/Jeremy.Jones/VivioJS/caches/MESIHelp.htm

01

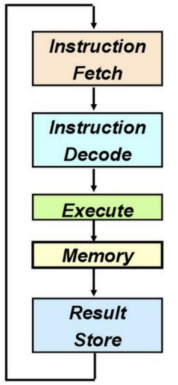
跟函数无关
开源工具：pmu-tools

概述 TMAM Top-down Micro-architecture Analysis Methodology



(图片来源于：intel 文档)

- **Fetch**
  - Fetch instruction pointed by PC from I-Cache
- **Decode**
  - Decode instruction (generate control signals)
  - Fetch operands from register file
- **Execute**
  - For a memory access: calculate effective address
  - For an ALU operation: execute operation in ALU
  - For a branch: calculate condition and target
- **Memory Access**
  - For load: read data from memory
  - For store: write data into memory
- **Write Back**
  - Write result back to register file
  - update program counter



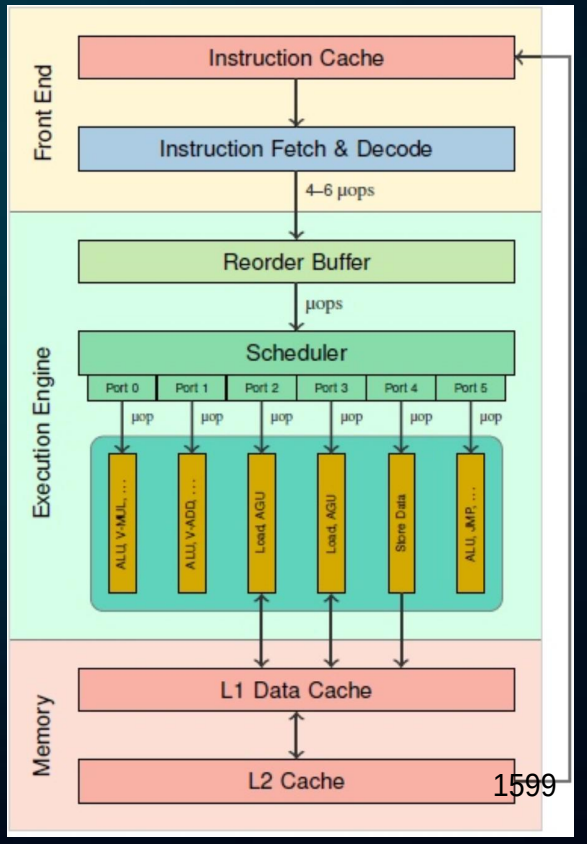
Basic five-stage pipeline

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

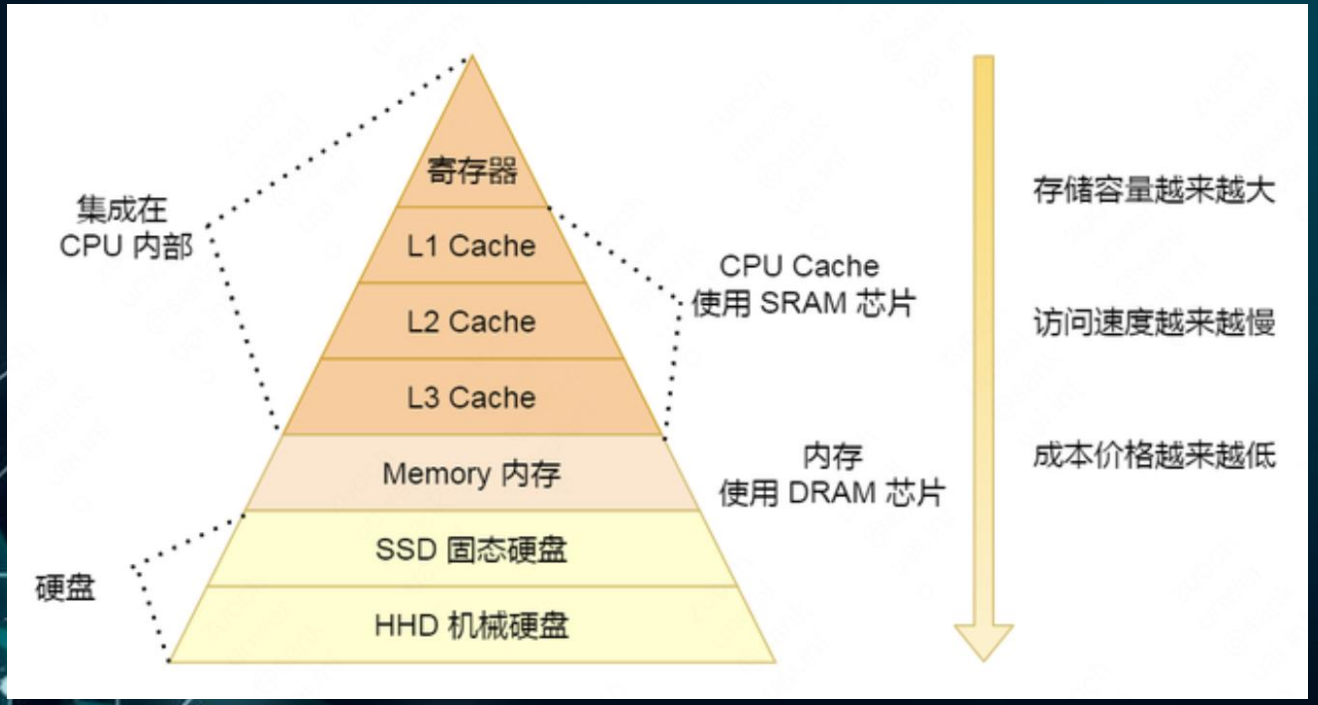
In the fourth clock cycle (the green column), the earliest instruction is in MEM stage, and the latest instruction has not yet entered the pipeline.

(图片来源：intel 官方文档)



01

概述 存储层次结构



存储器	硬件介质	单位成本 美元/MB	随机访问 延时
L1 Cache	SRAM	7	1ns
L3 Cache	SRAM	7	4ns
Memory	DRAM	0.015	100ns
Disk	SSD	0.0004	150us
Disk	HDD	0.00004	10ms

01

概述 CPU的时间哪去了?

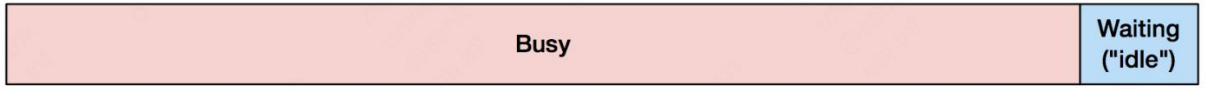
Computation
CPU Stall
- memory stalls
- resource stalls

Branch mis-predication

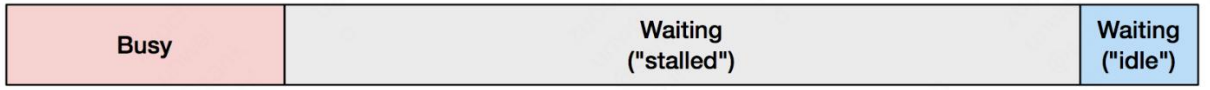
关注CPU有效使用率

怎么让程序跑的更快?
不要等待，要的东西要立即有

What you may think 90% CPU utilization means:



What it might really mean:



Stalled means the processor was not making forward progress with instructions, and usually happens because it is waiting on memory I/O. The ratio I drew above (between busy and stalled) is what I typically see in production. Chances are, you're mostly stalled, but don't know it.

What does this mean for you? Understanding how much your CPUs are stalled can direct performance tuning efforts between reducing code or reducing memory I/O. Anyone looking at CPU performance, especially on clouds that auto scale based on CPU, would benefit from knowing the stalled component of their %CPU.

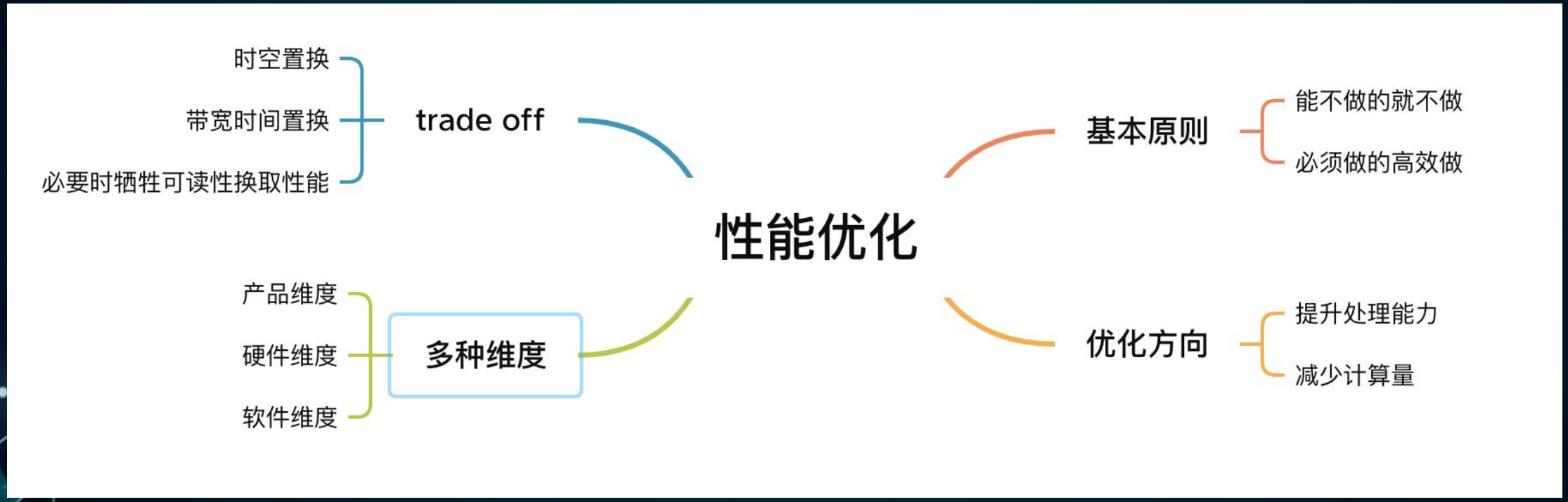
```
# perf stat -a -- sleep 10

 Performance counter stats for 'system wide':

     641398.723351      task-clock (msec)         #   64.116 CPUs utilized            (100.00%)
          379,651      context-switches          #    0.592 K/sec                    (100.00%)
           51,546      cpu-migrations            #    0.080 K/sec                    (100.00%)
       13,423,039      page-faults               #    0.021 M/sec
1,433,972,173,374      cycles                    #    2.236 GHz                      (75.02%)
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
1,118,336,816,068      instructions              #    0.78  insns per cycle          (75.01%)
  249,644,142,804      branches                  #  389.218 M/sec                    (75.01%)
    7,791,449,769      branch-misses             #    3.12% of all branches          (75.01%)

      10.003794539 seconds time elapsed
```

优化思路


性能优化
trade off
时空置换
带宽时间置换
必要时牺牲可读性换取性能
基本原则
能不做的就不做
必须做的高效做
多种维度
产品维度
硬件维度
软件维度
优化方向
提升处理能力
减少计算量

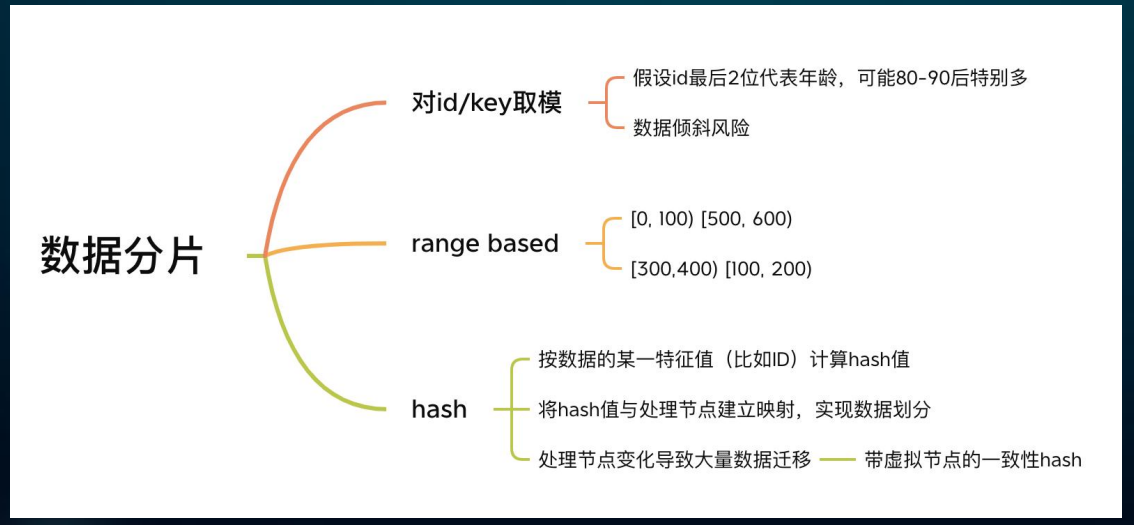
02

宏观架构 通过并行提高性能

- 数据并行
- 任务并行

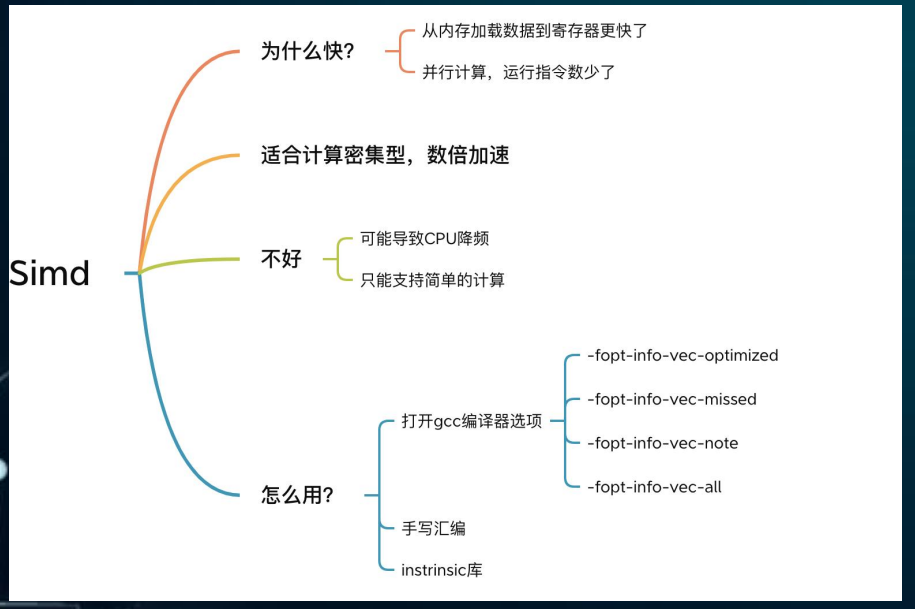
大问题通常能划分为被并行执行的小问题

- 多机器
- 多进程
- 多线程
- SIMD

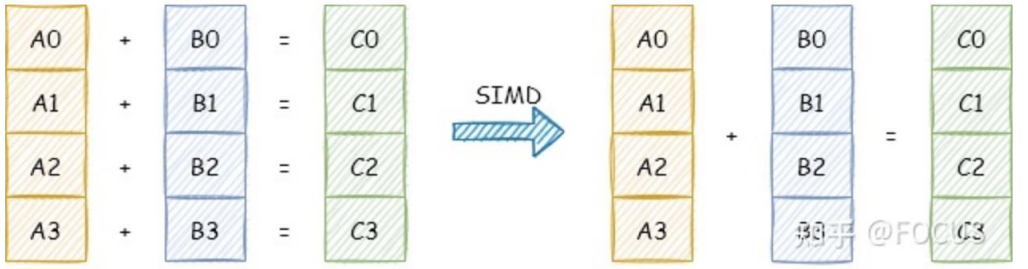


02

宏观架构　通过SIMD提高性能



SIMD 的全称是 Single Instruction Multiple Data，中文名"单指令多数据"。顾名思义，一条指令处理多个数据。



如上图所示：

- 一个普通加法指令，一次只能对两个数执行一个加法操作。
- 一个 SIMD 加法指令，一次可以对两个数组（向量）执行加法操作。

02

宏观架构 通过流水线提高性能

- 以面积换速度
- 用更多的资源换取高吞吐

02

宏观架构 通过批处理提高性能

- 指令被紧凑执行，i-cache局部性好
- 一段代码里的（相邻）数据被多次访问，提高d-cache命中

03

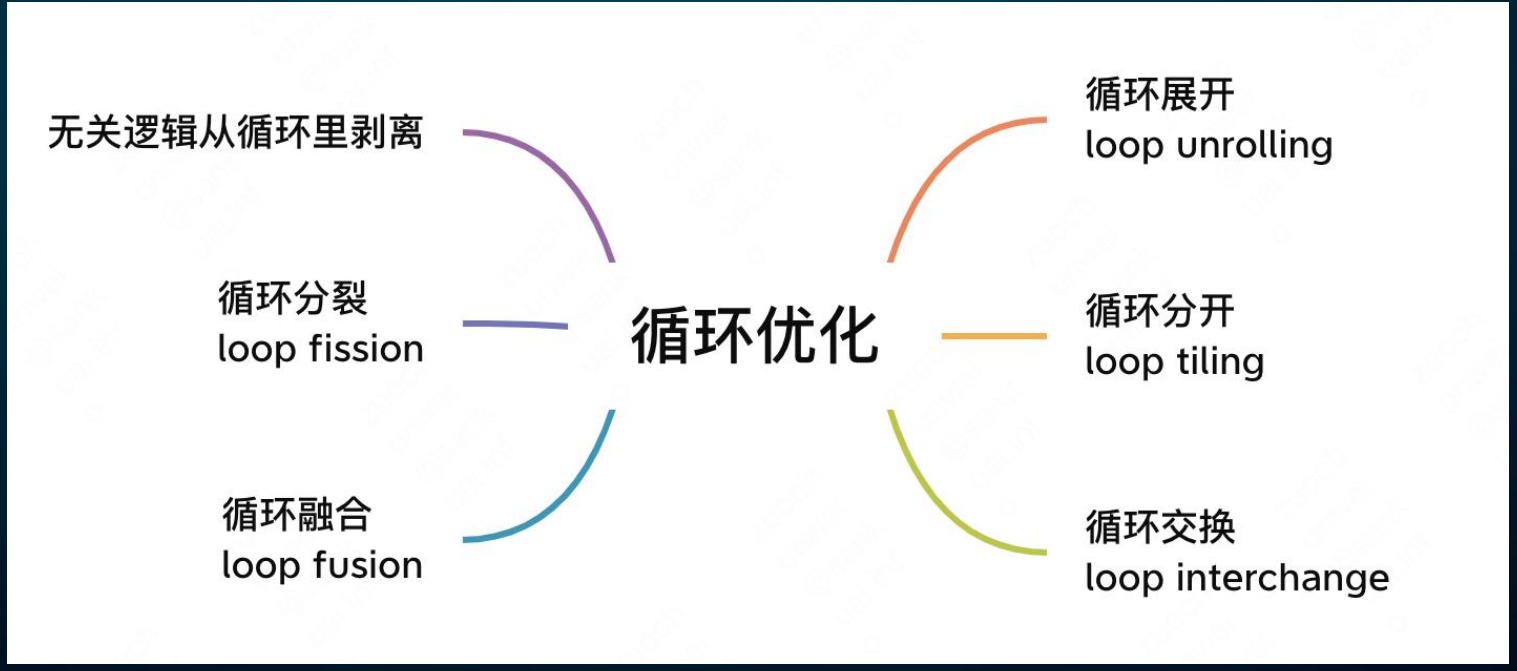
关键组件

CPU offload：0-Copy
内存池（复用）
消息队列（削峰填谷）
缓存（局部性）

04

实现细节

循环经常是程序的瓶颈，优化循环收益高，需重点关注



04

实现细节　循环优化：unrolling

```
4  const int N = 1024 * 1024 * 128;
5  uint32_t x = 0;
6
7  int main(int argc, char* argv[])
8  {
9      test_timer timer;
10
11     if (*argv[1] == '1') {
12         for (int i = 0; i < N; ++i) {
13             x += i;
14         }
15     } else {
16         for (int i = 0; i < N; i+=4) {
17             x += i;
18             x += i+1;
19             x += i+2;
20             x += i+3;
21         }
22     }
23
24     std::cout << x << ": time:" << timer.get_micro() << std::endl;
25
26     return 0;
27 }
```

循环展开

循环经常是程序的瓶颈，需要重点关注

```
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ loop.cpp -O3
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 0
4227858432: time:6122
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 0
4227858432: time:8868
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 1
4227858432: time:16953
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 1
4227858432: time:16971
[sankuai@xr-data-olap-experiment-staging05 test]$
```

展开循环后更快

04

实现细节　循环优化：fusion

```
DO I = 1, N
  A(I) = B(I) + C1
ENDDO
DO I = 1, N
  D(I) = A(I) + C2
ENDDO
```



```
DO I = 1, N
  A(I) = B(I) + C1
  D(I) = A(I) + C2
ENDDO
```

04

实现细节 循环优化

```
for (int i = 0; i < 1000; ++i) {
    if (flag) {
        do();
    }
}
```



```
int local_flag = flag
for (int i = 0; i < 1000; ++i) {
    if (local_flag) {
        do();
    }
}
```

04

实现细节　循环优化：fission

How do we optimize the following?

```
for (int32_t i = 0; i < batchSize; ++i) {
  int32_t bucket = hash(keyCol[i]) % ht->size;

  if (ht[bucket].key == keyCol[i]) {
    ht[bucket].value += valueCol[i];
  }
}
```



Break the big loop into smaller loops

```
for (int32_t i = 0; i < batchSize; ++i) {
  buckets[i] = hash(keyCol[i]) % ht->size;
}

for (int32_t i = 0; i < batchSize; ++i) {
  keys[i] = ht[buckets[i]].key;
}

for (int32_t i = 0; i < batchSize; ++i) {
  if (keys[i] == keyCol[i]) {
    ht[buckets[i]].value += valueCol[i];
  }
```

04

实现细节　循环优化：interchange

二维数组，先行后列 or 先列后行?
开启优化选项（-floop-interchange），编译器能自动识别

```
[sankuai@xr-data-olap-experiment-staging05 test]$ vim loop.cpp
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ loop.cpp -O3
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 0 && ./a.out 1
way:0
loop:, micro cost:35
way:1
loop:, micro cost:32
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ loop.cpp -O0
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 0 && ./a.out 1
way:0
loop:, micro cost:42304
way:1
loop:, micro cost:14044
[sankuai@xr-data-olap-experiment-staging05 test]$ _
```

O3会开启优化

O0不会优化

```
7  const int m = 1024*4, n = 1024;
8  struct Foo {
9      int64_t a[m][n];
10 };
11
12 void test_loop(char way)
13 {
14     Foo* f = new Foo;
15     for (int i = 0; i < m; ++i) {
16         for (int j = 0; j < n; ++j) {
17             f->a[i][j] = i + j;
18         }
19     }
20
21     int64_t sum = 0;
22     test_timer timer;
23     if (way == '1') {
24         for (int i = 0; i < m; ++i) {
25             for (int j = 0; j < n; ++j) {
26                 sum += f->a[i][j];
27             }
28         }
29     } else {
30         for (int i = 0; i < n; ++i) {
31             for (int j = 0; j < m; ++j) {
32                 sum += f->a[j][i];
33             }
34         }
35     }
36     std::cout << "way:" << way << std::endl;
37     timer.print_micro("loop:");
38 }
```

04

实现细节 通过预取遮掩访存时延

1. 指令预取是自动完成的
2. 数据预取分硬件预取和软件预取
   - gcc有选项，开启后会在编译阶段自动插入预取代码（适合连续性数据访存）
   - 对分散的数据预取，需要手动添加代码，需验证效果，时机把握很重要
   - 预取语句执行有开销，只有收益大于预取，且CACHE-MISS很高才是值得做
   - 建议手动加预取放到最后一步

04

实现细节 查找

基于有序数组的二分查找比std::map更快
尽量少用字符串比较，多用整型比较

```
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ find.cpp -O3
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out
345
vector, micro cost:99
345
map, micro cost:127
[sankuai@xr-data-olap-experiment-staging05 test]$
```

```cpp
const int N = 1024*64;

std::vector<int> keys;
std::map<int, int> map;
std::vector<int> vector;

void init(int n) {
    while (map.size() < N) {
        int key = rand() % 102400;
        map[key] = rand();
    }

    for (auto b = map.begin(); b != map.end(); ++b) vector.push_back(b->first);
    for (int i = 0; i < n; ++i) keys.push_back(rand() % 102400);
}

int find_map() {
    int found = 0;
    for (auto x : keys) {
        auto y = map.find(x);
        if (y != map.end()) ++found;
    }
    return found;
}

int find_array() {
    int found = 0;
    for (auto x : keys) {
        if (std::binary_search(vector.begin(), vector.end(), x)) ++found;
    }
    return found;
}

int main(int argc, char* argv[]) {
    srand(time(0));
    //初始化
    init(512);
    {
        test_timer t1;
        std::cout << find_array() << std::endl;
        t1.print_micro("vector");
    } {
        test_timer t1;
        std::cout << find_map() << std::endl;
        t1.print_micro("map");
    }
    return 0;
}
```

04

实现细节 空间换时间（查表）

```cpp
enum {
    NUMBER = 0, SPACE, U_CHAR, L_CHAR,
    ADD, SUB, MUL, DIV, OTHER, NUM,
};

int type2count[NUM] = {};
int *charmap[256];

void init() {
    for (int i = 0; i < 256; ++i)
    {
        if (i >= '0' && i <= '9') {
            charmap[i] = type2count + NUMBER;
        } else if (i == '+') {
            charmap[i] = type2count + ADD;
        } else if (i == '-') {
            charmap[i] = type2count + SUB;
        } else if (i == '*') {
            charmap[i] = type2count + MUL;
        } else if (i == '/') {
            charmap[i] = type2count + DIV;
        } else if (i >= 'a' && i <= 'z') {
            charmap[i] = type2count + L_CHAR;
        } else if (i >= 'A' && i <= 'Z') {
            charmap[i] = type2count + U_CHAR;
        } else if (i == ' ' || i <= '\t') {
            charmap[i] = type2count + SPACE;
        } else {
            charmap[i] = type2count + OTHER;
        }
    }
}
```

```cpp
void stat(char* s)
{
    for (int i = 0; s[i]; ++i) {
        if (s[i] >= '0' && s[i] <= '9') {
            type2count[NUMBER]++;
        } else if (s[i] == '+') {
            type2count[ADD]++;
        } else if (s[i] == '-') {
            type2count[SUB]++;
        } else if (s[i] == '*') {
            type2count[MUL]++;
        } else if (s[i] == '/') {
            type2count[DIV]++;
        }  else if (s[i] >= 'A' && s[i] <= 'Z') {
            type2count[U_CHAR]++;
        } else if (s[i] == ' ' || s[i] <= '\t') {
            type2count[SPACE]++;
        } else if (s[i] >= 'a' && s[i] <= 'z') {
            type2count[L_CHAR]++;
        } else {
            type2count[OTHER]++;
        }
    }
}

void stat_findtab(char* s)
{
    while (*s) {
        ++(*charmap[*s]);
        ++s;
    }
}
```

```
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out 1
not find tab stat, micro cost:280
106000
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out o
find tab stat, micro cost:174
106000
```

查表更快

04

实现细节 增量更新

只做增量，只传输差值，不做全量，这个思想有很多应用场景

举例：游戏服务器每间隔1秒都需要把玩家属性打包发送给客户端

1. 简单的做法：把所有属性打包（整个属性数组）一次性发给客户端，耗费带宽
2. 改进：给属性编号，发送的时候，只发送变化的id-value pair，节约带宽

04

实现细节 延迟计算

将计算尽可能往后推，从而达到降低计算开销的目的

拿游戏服务器举例：

1. 玩家的战斗力属性是通过装备、血量、等级等属性计算出来的
2. 装备、血量变化需要重算战力，血量频繁变化会导致战力频繁重算
3. 为战力增加一个脏标记，在装备、血量等变化的时候置脏
4. 为战力增加一个get接口，在脏的时候，重算战力，并清标记，再非脏时候，直接返回

延迟计算跟Copy-On-Write思路上类似

1. fork进程的时候，子进程共享父进程的地址空间，提高fork效率
2. 子进程修改地址空间内容的时候，才clone出一份子进程的独立内容

04

实现细节

绑核

- 规避core0
- lscpu：命令查看cpu信息
- C代码查核数
  - int get_nprocs_conf () //配置核数
  - int get_nprocs () //可用核数
- taskset
  - taskset -p pid //查进程核亲和性
  - taskset -pc core pid //指定进程运行在某个核上
  - taskset -c 1 ./a.out //启动时绑核
- 通过C代码绑核
  - sched_setaffinity / sched_getaffinity
  - pthread_setaffinity_np / pthread_getaffinity_np

04

实现细节　线程切换

结论：我测试的结果是线程切换单核单次3.5微秒，差不多一秒钟能切换30万次

开2个进程，进程a和b通过xsi消息队列通信。

进程a b的执行逻辑都是执行一个1千万次循环。

进程a循环内发送一个msgtype为a2b的消息，然后接收一个msgtype为b2a的消息，进程b先收后发，send和recv都不带nowait flag，这样的话，recv的动作会block。

把a b进程都绑定到core1上执行，启动命令
以taskset -c 1./a启动进程a
以taskset -c 1./b启动进程b

进程a在发送消息后，recv的时候会block，切换到进程b，b收消息，ok；再发消息，ok；再下一轮循环，收消息，block，被调度走；换a执行，依次交替执行。

协程：更低的切换成本

04

如果一个线程被暂停，那么其他线程应能继续前进，它需要有系统级（system-wide）的吞吐。

实现细节 lock-free

lock-free允许单个线程饥饿但保证系统级吞吐。如果一个程序的线程执行足够长的时间，那么至少一个线程会往前推进，那么这个算法就是lock-free的。

lock-free描述的是代码逻辑的属性，不使用锁的代码，大部分具有这种属性。

lock-free是对代码（算法）性质的描述，是属性，而无锁是说代码如何实现，是手段。

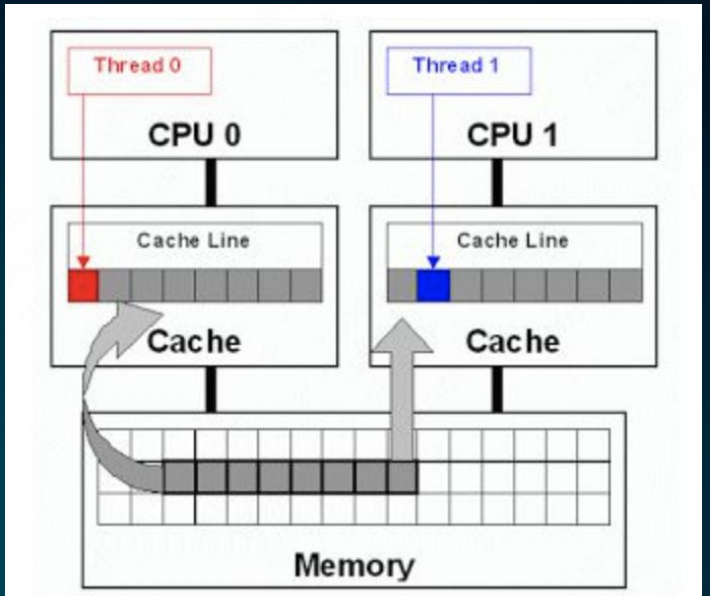
lock-free栈 下面是C++ atomic compare_exchange_weak()实现的一个lock-free堆栈（来自CppReference）

```
template <typename T>
struct node {
    T data;
    node* next;
    node(const T& data) : data(data), next(nullptr) {}
};

template <typename T>
class stack {
    std::atomic<node<T>*> head;
public:
    void push(const T& data) {
      node<T>* new_node = new node<T>(data);
      new_node->next = head.load(std::memory_order_relaxed);
      while (!head.compare_exchange_weak(new_node->next, new_node,
                                         std::memory_order_release,
                                         std::memory_order_relaxed));
    }
};
```

04

实现细节 false sharing(伪共享)



Intel提供的FalseSharing的示意图

```
const size_t shm_size = 16*1024*1024; //16M
static char shm[shm_size];
std::atomic<size_t> shm_offset{0};

void f() {
    for (;;) {
        auto off = shm_offset.fetch_add(sizeof(long));
        if (off >= shm_size) break;
        *(long*)(shm + off) = off;
    }
}
```

```
void f_fast() {
    for (;;) {
        const long inner_loop = 16;
        auto off = shm_offset.fetch_add(sizeof(long) * inner_loop);
        for (long j = 0; j < inner_loop; ++j) {
            if (off >= shm_size) return;
            *(long*)(shm + off) = j;
            off += sizeof(long);
        }
    }
}
```

04

实现细节 inline

函数调用单次开销大约2纳秒
inline能减少函数调用开销
inline是对编译器的建议，是否内联决定权在编译器
inline函数一般需要定义在头文件，LTO可以跨编译单元内联
inline缺点：可执行二进制文件可能变大

04

实现细节 restrict

```cpp
int add1(int* a, int* b)
{
    *a = 10;
    *b = 12;
    return *a + *b;
}
```

```
0000000000400a10 <_Z4add1PiS_>:
  400a10:   c7 07 0a 00 00 00       movl   $0xa,(%rdi) ; *a = 10
  400a16:   c7 06 0c 00 00 00       movl   $0xc,(%rsi) ; *b = 10
  400a1c:   8b 07                   mov    (%rdi),%eax ; 结果 = *a
  400a1e:   83 c0 0c                add    $0xc,%eax   ; 结果 += 12
  400a21:   c3                      retq
```

```cpp
int add2(int* __restrict  a, int* __restrict b)
{
    *a = 10;
    *b = 12;
    return *a + *b ;
}
```

```
0000000000400a30 <_Z4add2PiS_>:
  400a30:   c7 07 0a 00 00 00       movl   $0xa,(%rdi) ; *a = 10
  400a36:   b8 16 00 00 00          mov    $0x16,%eax  ; 结果 = 22
  400a3b:   c7 06 0c 00 00 00       movl   $0xc,(%rsi) ; *b = 12
  400a41:   c3                      retq
```

04

实现细节 restrict与SIMD

```
void add_restrict(int * __restrict a, const int * b, const int * c) {
    for (int i = 0; i < 4; i++)
        a[i] = b[i] + c[i];
}

void add_norestrict(int * a, const int * b, const int * c) {
    for (int i = 0; i < 4; i++)
        a[i] = b[i] + c[i];
}
```

restrict告诉编译器指针a不会与其他指针指向的内存区域重叠

编译器基于该信息自动做SIMD优化

```
add_restrict(int*, int const*, int const*):
        movdqu  xmm0, xmmword ptr [rsi]
        movdqu  xmm1, xmmword ptr [rdx]
        paddd   xmm1, xmm0
        movdqu  xmmword ptr [rdi], xmm1
        ret
add_norestrict(int*, int const*, int const*):
        mov     eax, dword ptr [rdx]
        add     eax, dword ptr [rsi]
        mov     dword ptr [rdi], eax
        mov     eax, dword ptr [rdx + 4]
        add     eax, dword ptr [rsi + 4]
        mov     dword ptr [rdi + 4], eax
        mov     eax, dword ptr [rdx + 8]
        add     eax, dword ptr [rsi + 8]
        mov     dword ptr [rdi + 8], eax
        mov     eax, dword ptr [rdx + 12]
        add     eax, dword ptr [rsi + 12]
        mov     dword ptr [rdi + 12], eax
        ret
```

04

实现细节 restrict与SIMD

```
for (int i = 0; i < n; i++) {
    b[i] = 0;
    for (int j = 0; j < n; j++) {
        b[i] += a[i][j];
    }
}
```



```
for (int i = 0; i < n; i++) {
    double sum = 0;
    double* __restrict__ a_row = a[i];
    for (int j = 0; j < n; j++) {
        sum += a_row[j];
    }
    b[i] = sum;
}
```

04

实现细节 constexpr if

constexpr if没有运行时开销

```
5  template<typename T> __attribute__ ((noinline))
6  auto test_constexpr(T t)
7  {
8      if constexpr(std::is_integral<T>::value)
9          return 1;
10     else
11         return 0;
12 }
13
14 int main(int argc, char* argv[]) {
15     auto x = test_constexpr(100);
16     std::cout << x << std::endl;
17     return 0;
18 }
```

```
0000000000014c80 <auto test_constexpr<int>(int) (.constprop.0)>:
 14c80: b8 01 00 00 00                  movl    $1, %eax
 14c85: c3                              retq
 14c86: 66 2e 0f 1f 84 00 00 00 00 00   nopw    %cs:(%rax,%rax)
```

04

实现细节 对齐

误用aligned带来性能下降

```
experiment-staging05 test]$ g++ cache_test.cpp -O3 -o noncache
experiment-staging05 test]$ vim cache_test.cpp
experiment-staging05 test]$ g++ cache_test.cpp -O3 -o cached
experiment-staging05 test]$ taskset -c 1 ./noncache
5 microseconds:176314
experiment-staging05 test]$ taskset -c 1 ./cached
5 microseconds:778845
```

```
2
3  #define CACHE_LINE __attribute__((aligned(64)))
4  //#define CACHE_LINE
5
6  struct S
7  {
8    int r1;
9    int r2;
10   S ():r1 (1), r2 (2) {}
11 } CACHE_LINE;
12
13 void add(const S a[],int members,long &total) {
14     int idx = members;
15     do {
16           total += a[idx].r1;
17           total += a[idx].r2;
18     } while(--idx);
19 }
20
21 int main (int argc, char *argv[])
22 {
23   const int SIZE = 1024 * 1024 * 64;
24   S *a = (S*)malloc(sizeof(S) * SIZE);
25   std::iota((int64_t*)a, (int64_t*)(a+SIZE), 1);
26   long total = 0L;
27   int loop = 5;
28   test_timer timer;
29   while (loop--) {  // 方便对比测试
30       add(a, SIZE, total);
31   }
32   std::cout << "total:" << total << " microseconds:" << timer.get_micro() << std::endl;
33   return 0;
34 }
```

04

实现细节

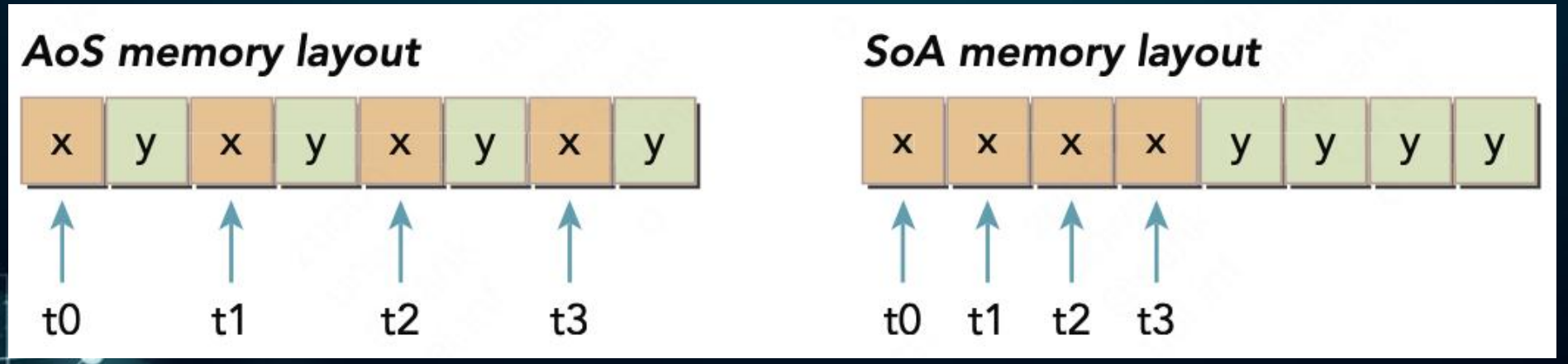
减少复制：C++11移动语义

1. 移动构造函数
2. 移动赋值操作符
3. 带右值引用的emplace_back等接口

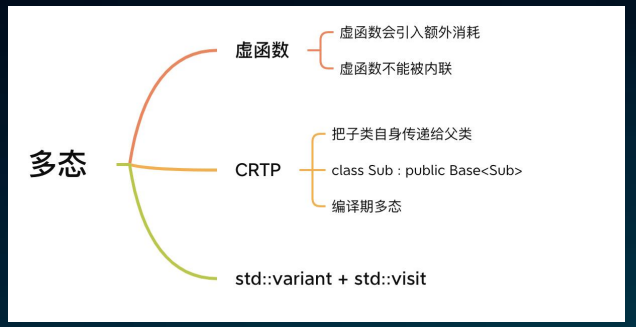
04

实现细节 数据结构

- 结构体的数组 vs 数组的结构体
- 同时考虑逻辑结构和存储结构



04



实现细节　减少虚函数调用

```
vf call cost:2.01487 nsec
[sankuai@xr-data-olap-experiment-staging05 test]$ vim vf_test.cpp
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ vf_test.cpp -O3 -g -std=c++14
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out
x:10485761
virtual call cost:2.42192 nsec
[sankuai@xr-data-olap-experiment-staging05 test]$ vim vf_test.cpp
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ vf_test.cpp -O3 -g -std=c++14
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out
x:-5242879
non-virtual call cost:0.128473 nsec
[sankuai@xr-data-olap-experiment-staging05 test]$ vim vf_test.cpp
[sankuai@xr-data-olap-experiment-staging05 test]$ g++ vf_test.cpp -O3 -g -std=c++14
[sankuai@xr-data-olap-experiment-staging05 test]$ ./a.out
x:-5242879
non-virtual (noinline) call cost:2.02149 nsec
[sankuai@xr-data-olap-experiment-staging05 test]$
```

```cpp
#include "gettime.h"

const unsigned int N = 10 * 1024 * 1024;

struct B {
    virtual ~B() {}
    __attribute__ ((noinline))
    /*virtual*/ int vf(int x) { return x; }
};

struct D1 : B {
    //__attribute__ ((noinline))
    int vf(int x) /*override final*/ { return x + 1; }
};

struct D2 : B {
    //__attribute__ ((noinline))
    int vf(int x) /*override final*/ { return x + 2; }
};

//void test(B* pb)
int test(B* pb[])
{
    int x = 1;
    for (unsigned int i = 0; i < N; ++i)
    {
        if (i & 1)
            x += pb[0]->vf(i);
        else
            x += pb[1]->vf(i);
    }
    return x;
}

int main()
{
    B* pb[2] = {new D1, new D2};
    test_timer timer;
    int x = test(pb);
    std::cout << "x:" << x << std::endl;
    std::cout << "non-virtual (noinline) call cost:" << (timer.get() * 1.0 / N) << " nsec" << std::endl;
    return 0;
}
```

04 奇异递归模板模式

实现细节 CRTP

```
template <class T>
struct Base
{
    void interface()
    {
        // ...
        static_cast<T*>(this)->implementation();
        // ...
    }

    static void static_func()
    {
        // ...
        T::static_sub_func();
        // ...
    }
};

struct Derived : Base<Derived>
{
    void implementation();

    static void static_sub_func();
};
```

那同样是实现多态，是用继承好呢，还是用variant好呢？可以看这个图：

Inheritance	std::variant
Need not know all the derived types upfront (open-world assumption)	Must know all the cases upfront (closed-world assumption)
Dynamic Allocation (usually)	No dynamic allocation
Intrusive (must inherit from the base class)	Non-intrusive (third-party classes can participate)
Reference semantics (think how you copy a vector of pointers to base class?)	Value semantics (copying is trivial)
Algorithm scattered into classes	Algorithm in one place
Language supported (Clear errors if pure-virtual is not implemented)	Library supported (poor error messages)
Creates a first-class abstraction	It's just a container
Keeps fluent interfaces	Disables fluent interfaces. Repeated std::visit
Supports recursive types (Composite)	Must use recursive_wrapper and dynamic allocation. Not in the C++17 standard.

04

实现细节 编译器优化

- LTO
- PGO
- 分支预测

小结

- 理解系统底层对做优化有好处
- 相信并提醒编译器做优化

thanks

CPP-Summit 2022

谢丙堃



《现代C++语言核心特性解析》作者

从事C++开发工作十余年，先后在数家知名互联网公司担任C++高级开发工程师、技术专家和技术经理。是C++语言的爱好者，热衷于研究C++语言的新特性以及C++模板元编程技术。著有《现代C++语言核心特性解析》一书，被评为异步社区2021年影响力作者，也是Intel DPC++认证讲师。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

谢丙堃
《现代C++语言核心特性解析》作者

SYCL 编码实践与编译设计浅析

议程

1. **SYCL & DPC++基础介绍**
   异构计算的背景，DPC++的优势
2. **SYCL & DPC++编程基础**
   理解DPC++中设备、队列、统一共享内存
3. **SYCL & DPC++编译设计简介**
   理解DPC++编译设备代码的方法
4. **DevCloud中学习SYCL & DPC++**
   介绍使用DevCloud的方法

01

SYCL & DPC++ 基础介绍

计算时代的变化

CPU 单核
单线程



CPU 多核
多线程



多种计算芯片
异构计算

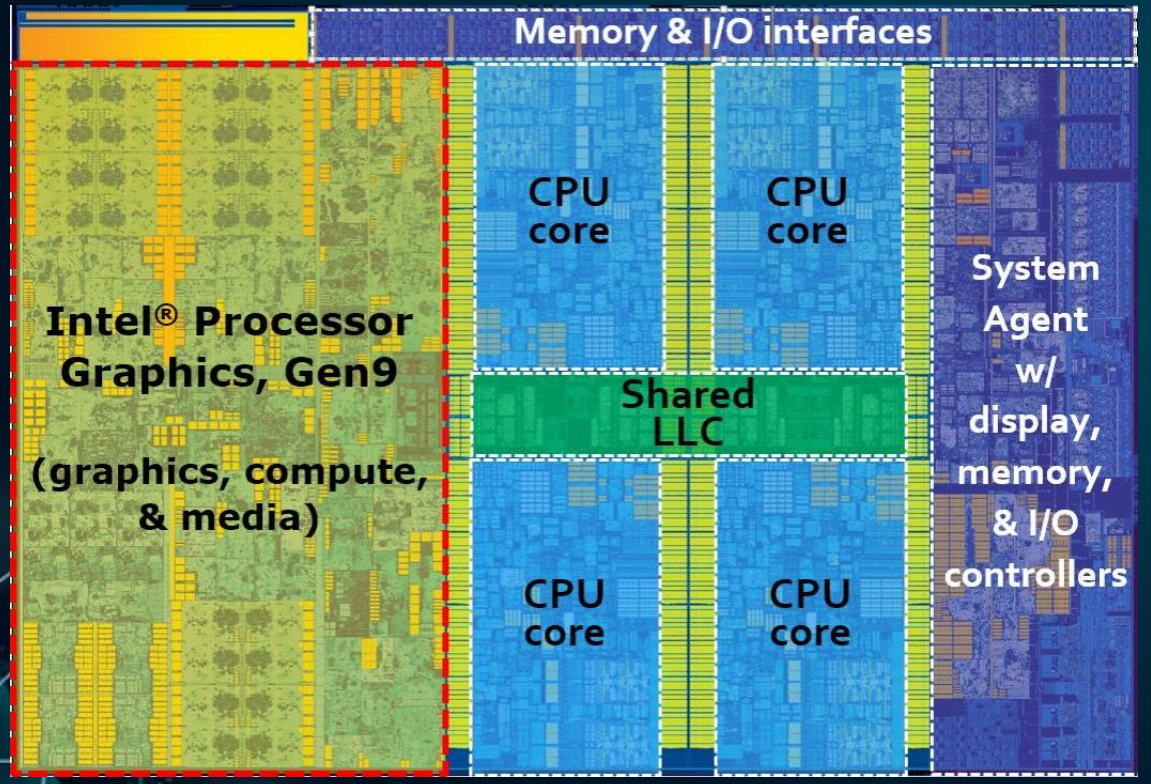
什么是异构计算

- 异构计算指使用一种以上处理器或内核的系统。
  - 通过添加不同的协处理器；
  - 集成专门的处理功能来处理特定任务。

- 这些系统的性能或能效可显著提升。

现代编程任务中的挑战

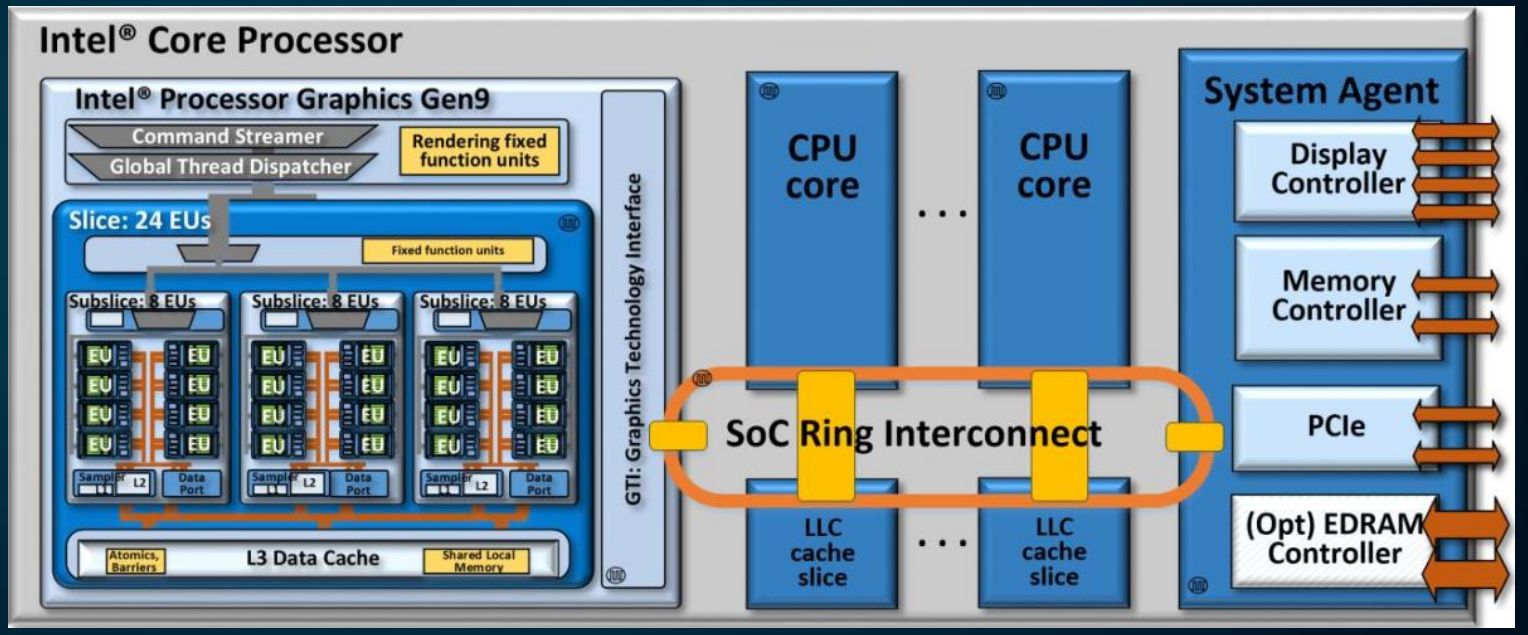
- 英特尔 酷睿 i7 处理器 6700K





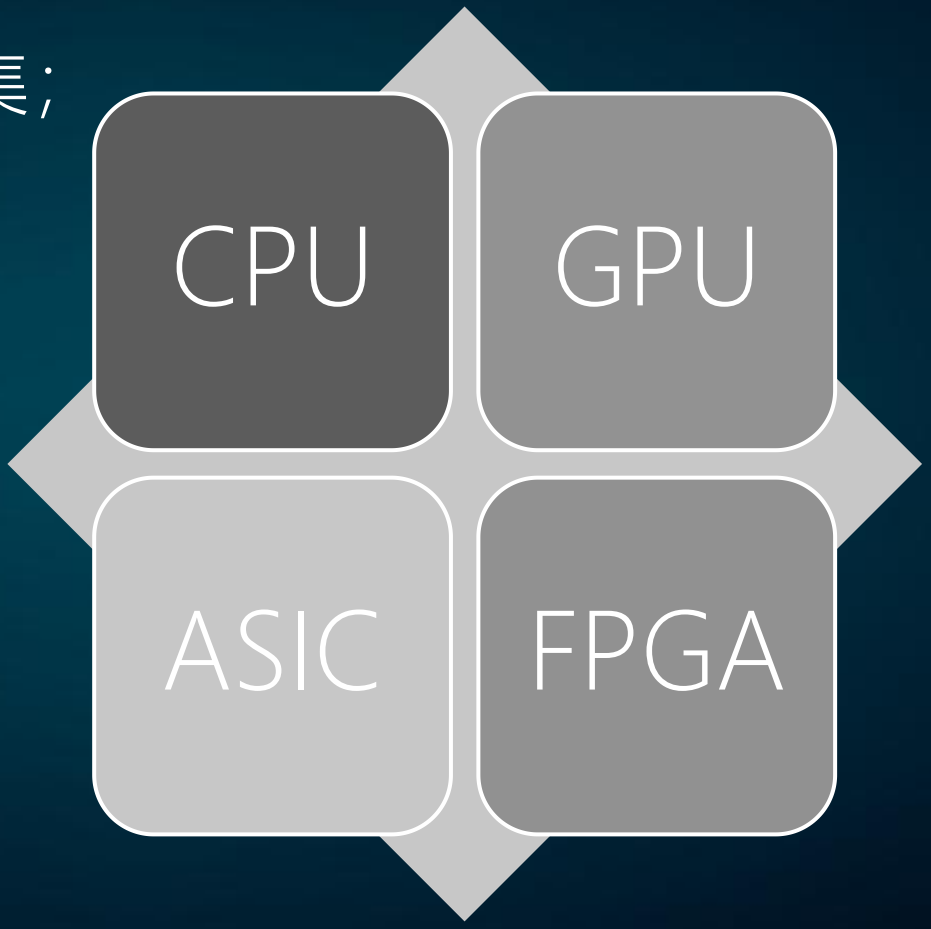
现代编程任务中的挑战

- Intel HD Graphics 530
  - Shading Units 192
  - TMUs 24
  - ROPs 3
  - Execution Units 24



现代编程任务中的挑战

- 每种处理器拥有不同的指令集；
- 使用不同的编程方法和规范；
- 需要不同的工具链。

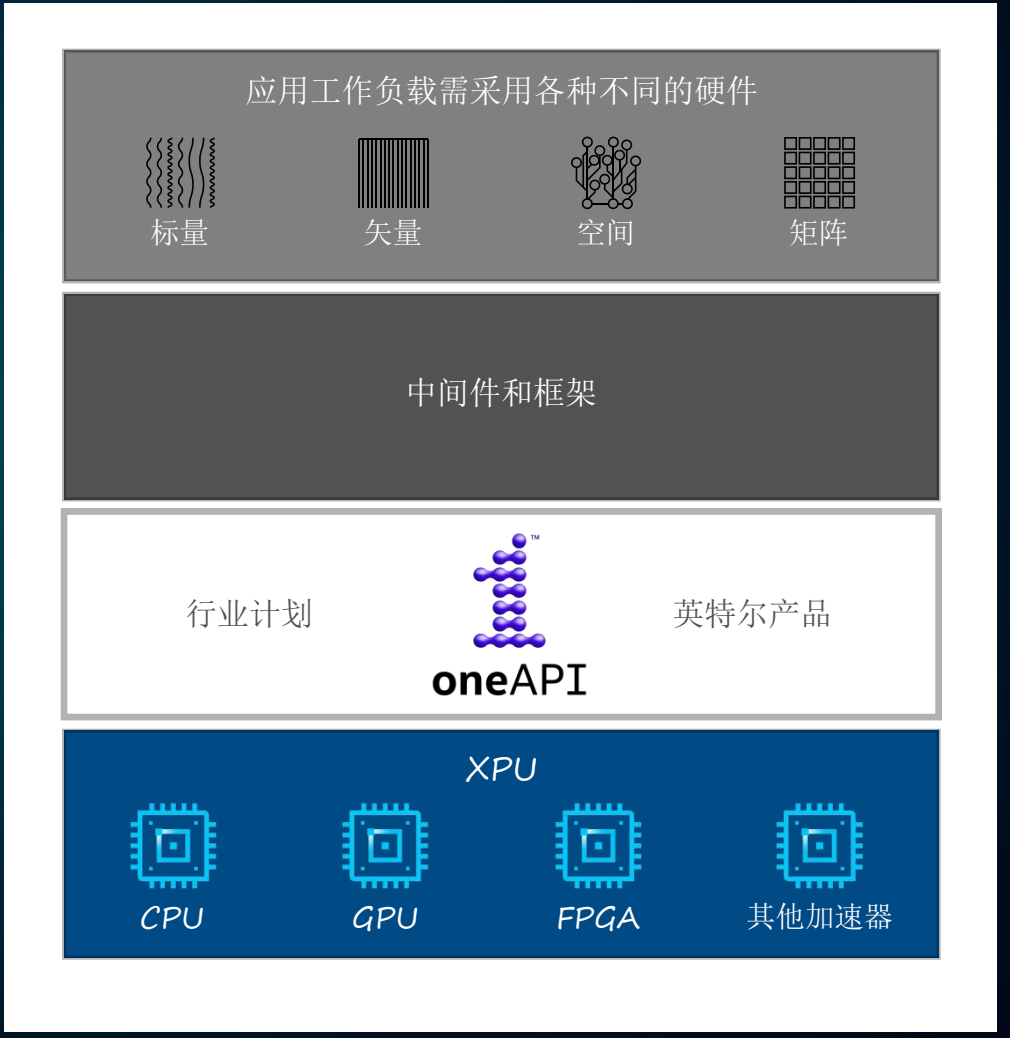


oneAPI - 用于多种架构编程模型

- oneAPI 是 Intel 为了推动加速计算的新时代，摆脱专有编程模型的经济和技术负担而推出的一套开发框架。

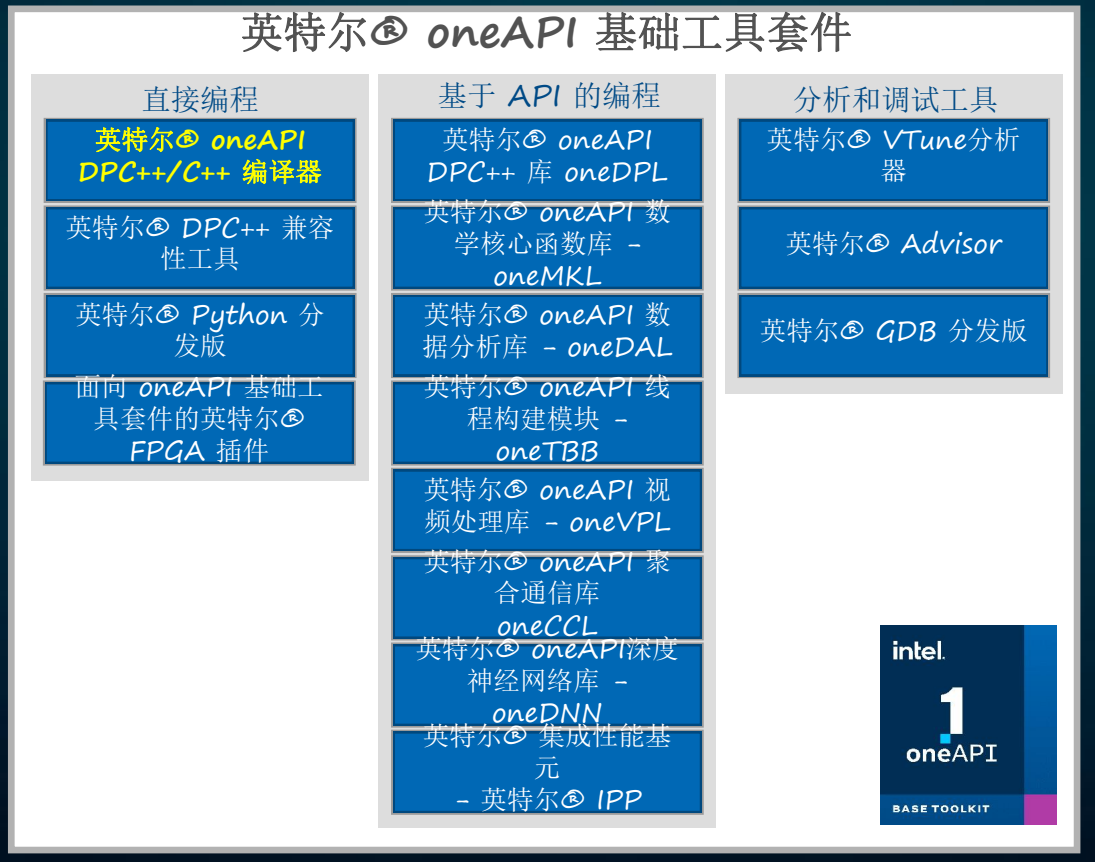
- 硬件和框架直接的抽象层
  - 让应用开发者不必担心硬件架构
  - 专注于任务本身
  - 兼容性、移植性强

- 开发套件
  - 软件工具众多
  - 适用范围广
    - HPC IOT AI ...



oneAPI – SYCL & DPC++

- DPC++是Intel oneAPI开发套件中的一个组件。



DPC++(Data Parallel C++)

- DPC++是SYCL标准的一种实现：
  - 使用标准现代C++（C++17标准）；
  - 集成了 SYCL 标准，支持数据并行性和异构编程；
  - 扩展 SYCL 标准：增强效率，提升性能；
  - 基于LLVM的开源编译器。

02

SYCL & DPC++ 编程基础

一个最简单的例子

单一源代码：host和device代码混合编程。

使用标准C++代码。

queue：使用DPC++ 队列来调度和执行设备上的命令队列

malloc_shared：使用Unified Shared Memory (USM) 来进行数据管理

parallel_for：使用加速设备并行执行lambda表达式计算

```
#include <CL/sycl.hpp>
constexpr int N=16;
using namespace sycl;
int main() {
  queue q;
  int *data = malloc_shared<int>(N, q);
  q.parallel_for(N, [=](auto i) {
    data[i] = i;
  }).wait();
  for (int i=0; i<N; i++) std::cout << data[i] << "\n";
  free(data, q);
  return 0;
}
```

Host code

Device code

Host code

free：使用sycl::free释放分配的USM

队列(queue)

- 队列用于提交命令组(command group)到SYCL 运行时执行；
  - 成员函数submit：将命令组功能对象提交到队列，以便安排在设备上执行；
  - 成语函数parallel_for是submit的一种简化写法。

- 队列是一种将工作(work)提交到设备的机制；
- 一个队列映射(map)到一个设备，多个队列(multiple queue)可以映射到同一设备。

使用submit提交命令

```
#include <CL/sycl.hpp>
constexpr int N=16;
using namespace sycl;
int main() {
  queue q;
  int *data = malloc_shared<int>(N, q);
  q.submit([&] (handler &h){
    h.parallel_for(N, [=](auto i) {
    data[i] = i;
  });
  }).wait();

  for (int i=0; i<N; i++) std::cout << data[i] << "\n";
  free(data, q);
  return 0;
}
```

handler：命令组句柄对象，提供一系列的调度函数

设备选择

- 队列可以通过设备选择器(device selector)选择指定设备：

```
default_selector selector;
// host_selector selector;
// cpu_selector selector;
// gpu_selector selector;
queue q(selector);
std::cout << "Device: " << q.get_device().get_info<info::device::name>() << std::endl;
```

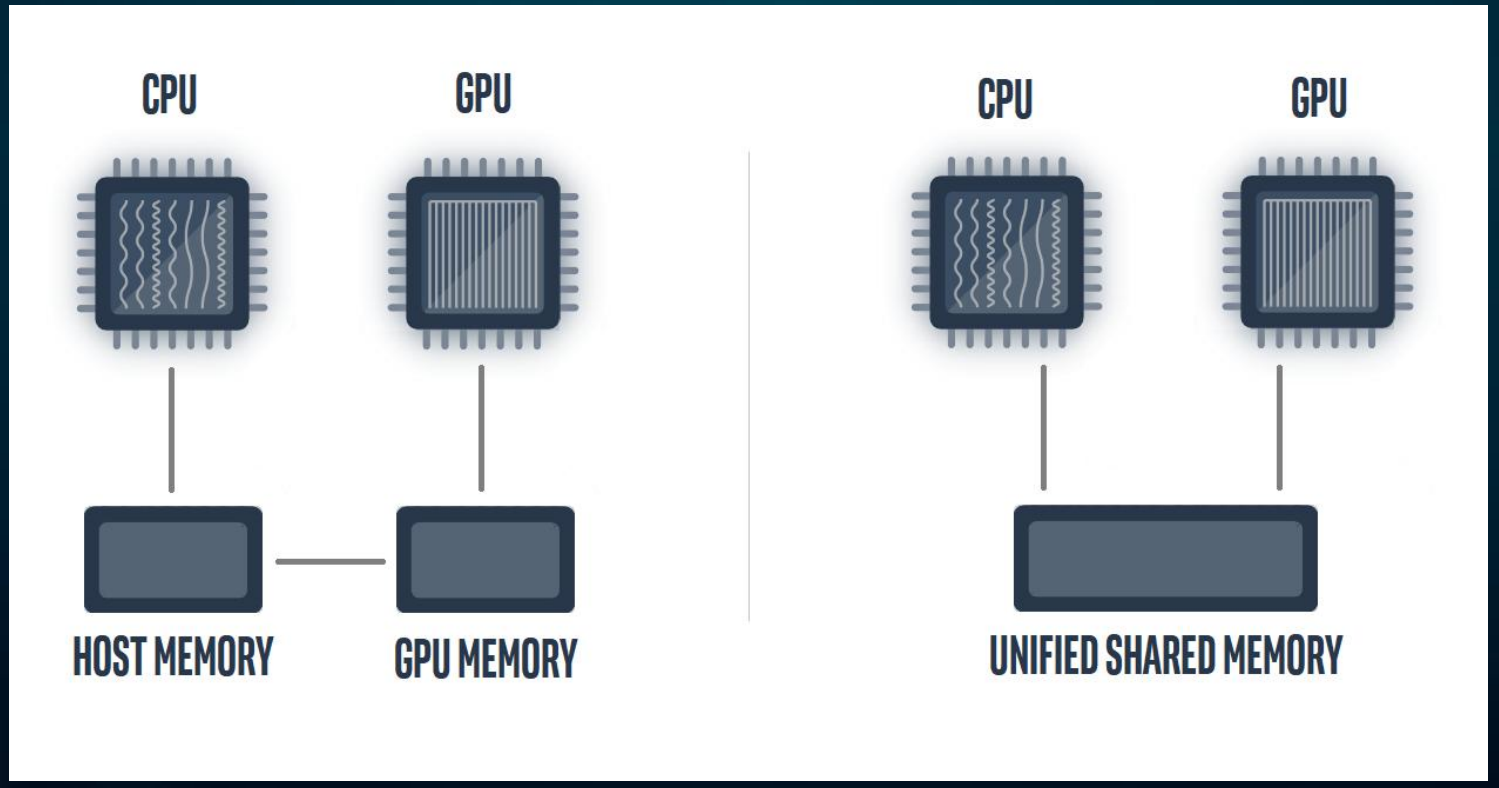
- 自定义设备选择：

```
class IntelGPUSelector : public device_selector {
 public:
  int operator()(const device& Device) const override {
    const std::string DeviceName = Device.get_info<info::device::name>();

    return Device.is_gpu() && (DeviceName.find("Intel") != std::string::npos) ? 100 : 0;
  }
};
```

统一共享内存 (USM)

- 统一共享内存(Unified Shared Memory)是一种基于指针的内存模型方法，适用于异构编程。



SYCL缓冲器(buffer) 方法

- 缓冲器(buffer)
- 访问器(accessor)

- 优点:
  - 非常简洁地表达了数据依赖关系
- 缺点:
  - 使用buffer不如直接使用指针和数组方便

```
#include <CL/sycl.hpp>
#include <vector>
constexpr int N = 16;
using namespace std;
using namespace sycl;
int main() {
  queue q;
  vector<int> v(N);
  {
    buffer buf(v);
    q.submit([&](handler &h) {
       accessor a(buf, h, write_only);
       h.parallel_for(N, [=](auto i) { a[i] = i; });
     }).wait();
  }

  for (int i = 0; i < N; i++)
    std::cout << v[i] << "\n";
  return 0;
}
```

DPC++ 并行内核

- 并行内核允许一个操作的多个实例并行执行
- 并行内核中没有循环和迭代器，每个操作都是完全独立的，并且不分顺序
- 并行内核使用 parallel_for 函数表示

CPU 应用中的 ***for*** 循环 （能否并行要看编译器）

```
for(int i=0; i < 1024; i++){
    a[i] = b[i] + c[i];
});
```

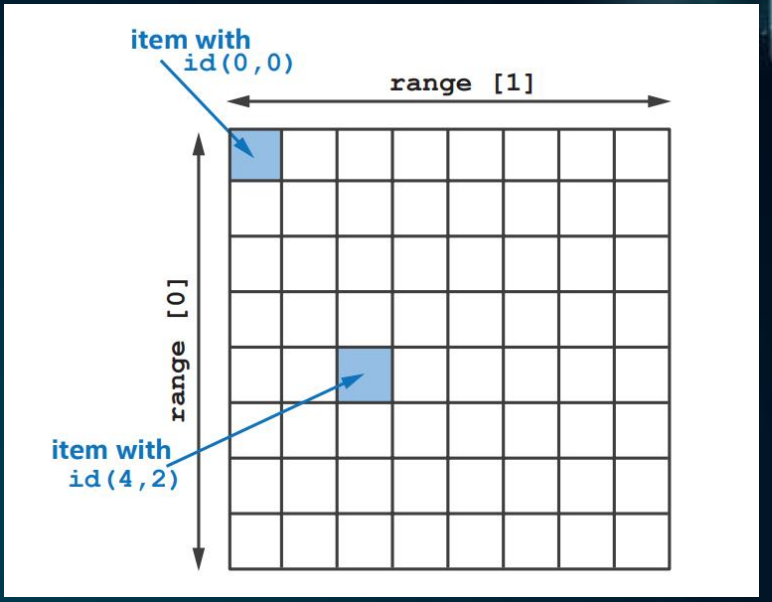


使用 ***parallel_for*** 卸载到加速器

```
h.parallel_for(range<1>(1024), [=](id<1> i){
    A[i] =  B[i] + C[i];
});
```

DPC++ 基础数据并行内核

- 基本并行内核的功能通过 range、id 和 item 类提供。

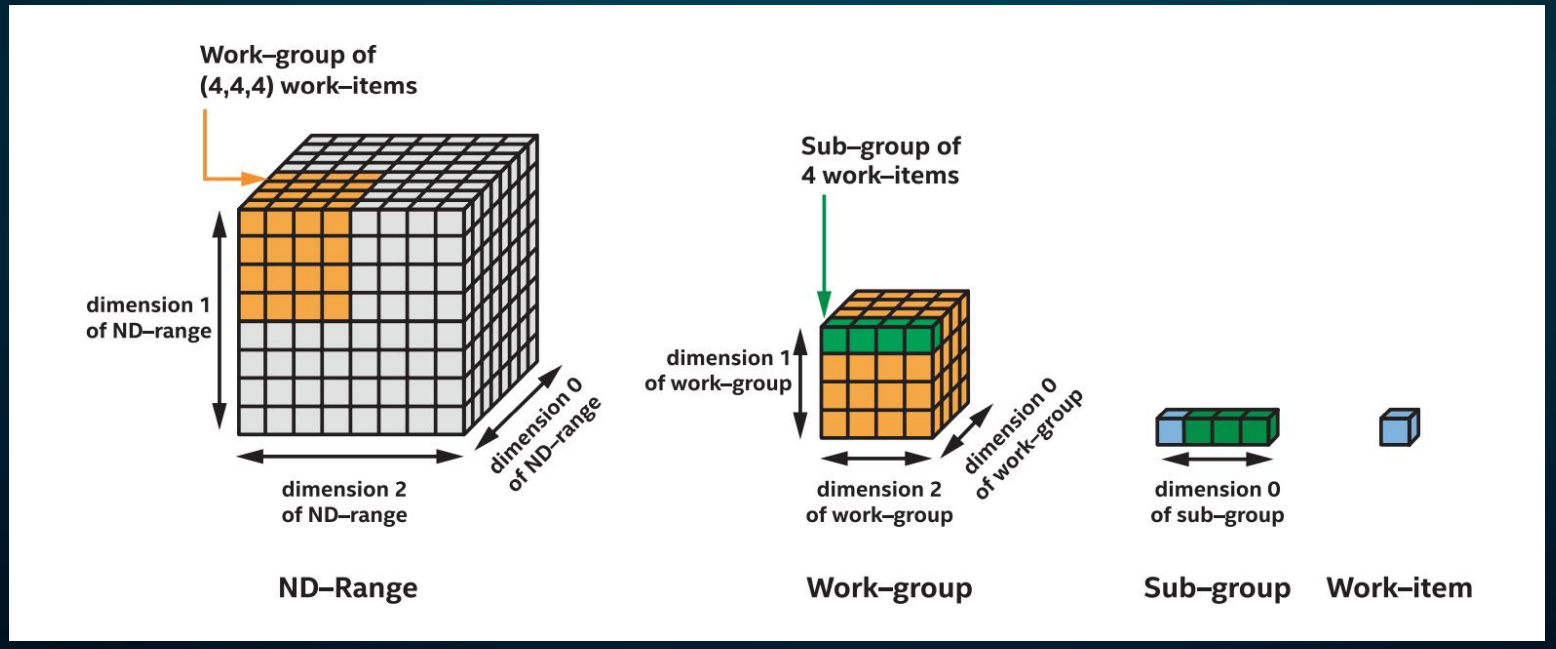


- range 类用于描述并行执行维度和大小
  - 可以表示1、2、3维
  - 维度需要在编译时确定
  - 每个维度的大小可以是运行时指定
- id 类用于表示range空间中的索引
  - 同样可以表示1、2、3维
  - 维度需要在编译时确定
  - 索引一个并行运行的实例
  - buffer的偏移
- item 类代表内核函数的单个实例
  - 封装内核的执行范围和该范围内的实例索引（分别使用 get_id 和 get_range）
  - 与 range 和 id 一样，它的维度必须在编译时确定

```
h.parallel_for(range<1>(1024), [=](item<1> item){
    auto idx = item.get_id();
    auto R = item.get_range();
    // CODE THAT RUNS ON DEVICE
});
```

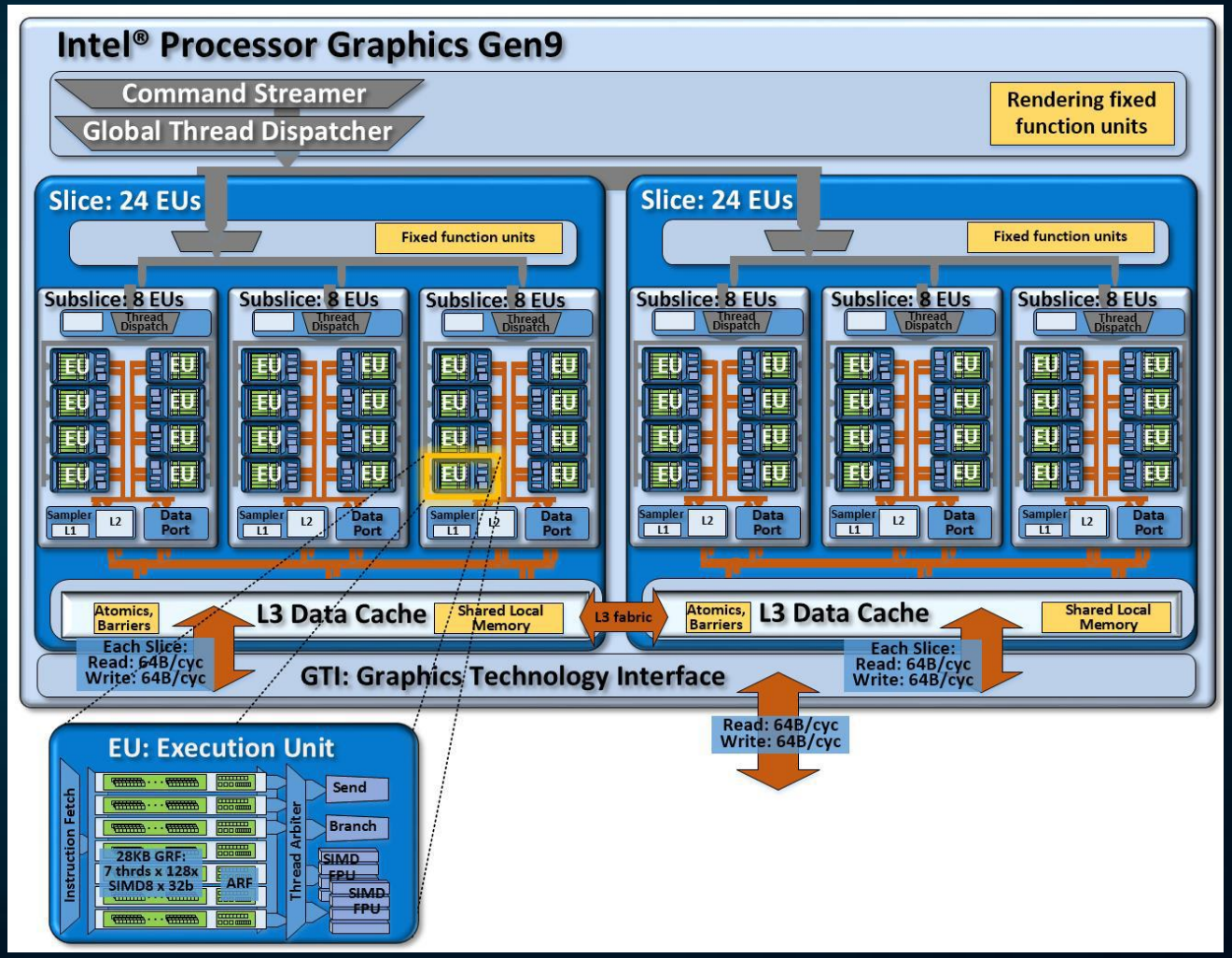
DPC++ 显式 ND-Range 内核

- 基础并行内核虽然使用方便，但是无法根据硬件架构进行优化
- ND-Range 内核可以将实例分为不同类型的分组，并且将它们精确的映射到硬件平台上
- 正确的使用 ND-Range 内核可以充分的发挥出硬件性能潜力，包括内存访问和计算单元分配等



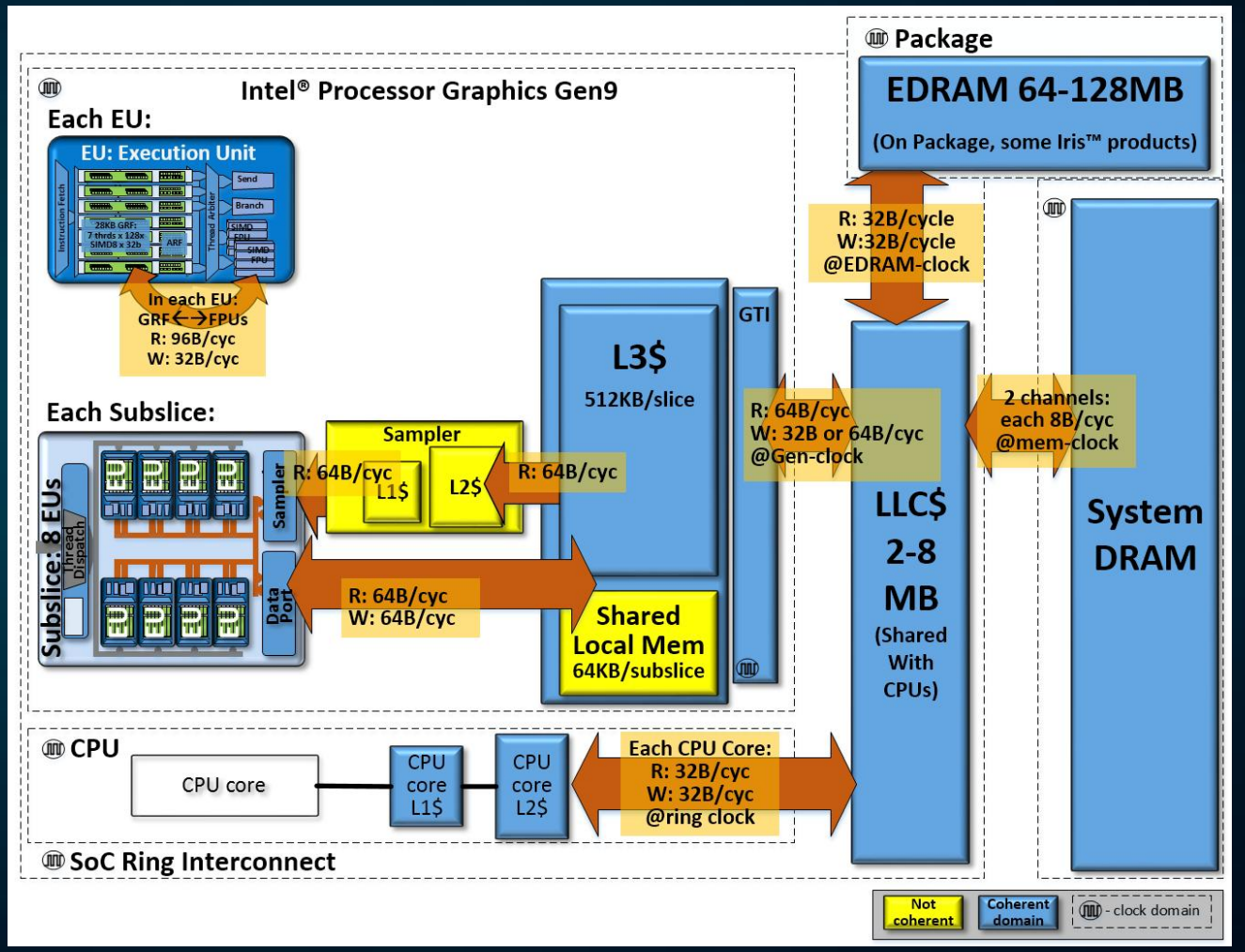
DPC++ ND-Range 发挥硬件性能原理

- Intel Skylake GT3/e Graphics
- With 48 EUs and eDRAM



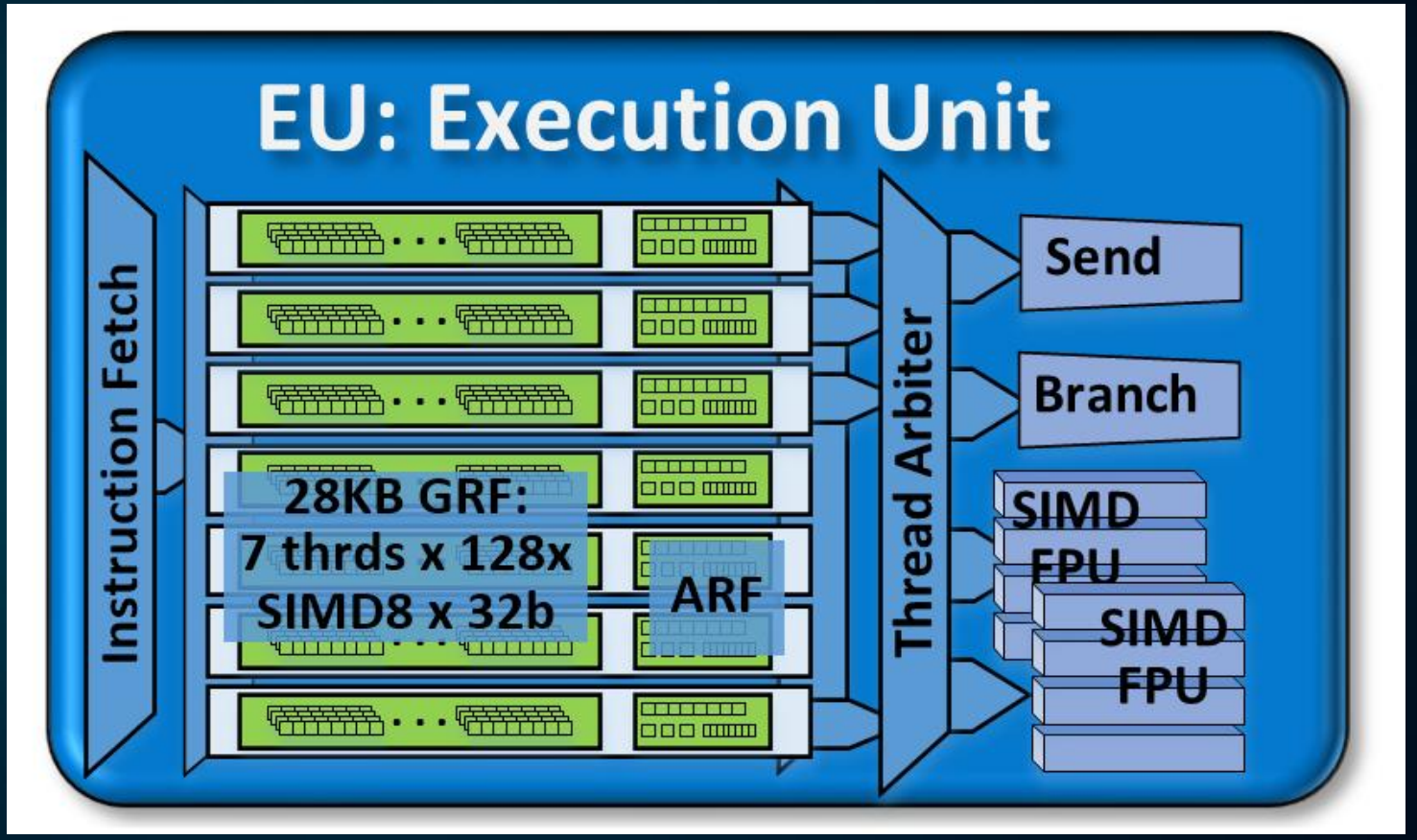
DPC++ ND-Range 发挥硬件性能原理

- Intel Skylake GT3/e Graphics
- With 48 EUs and eDRAM



DPC++ ND-Range 发挥硬件性能原理

- Intel Skylake GT3/e Graphics
- With 48 EUs and eDRAM



DPC++ 中的范围

- DPC++ 语言和运行时包含一系列 C++ 类、模板和库，但并不是所有C++特性都能在任意范围使用

- 应用范围和命令组范围：
  - 在主机上执行的代码
  - 应用和命令组范围内均支持 C++ 的所有功能
- 内核范围：
  - 在设备上执行的代码
  - 内核范围支持的 C++ 有部分限制
    - 更广泛的设备支持和大规模并行性
    - 不支持C++特性包括：动态多态性、动态内存分配（因此不使用 new 或 delete 运算符进行对象管理）、静态变量、函数指针、运行时类型信息 (RTTI) 和异常处理。
    - 不允许从内核代码调用任何虚拟成员函数和可变参数函数。
    - 内核代码中不允许递归。

03

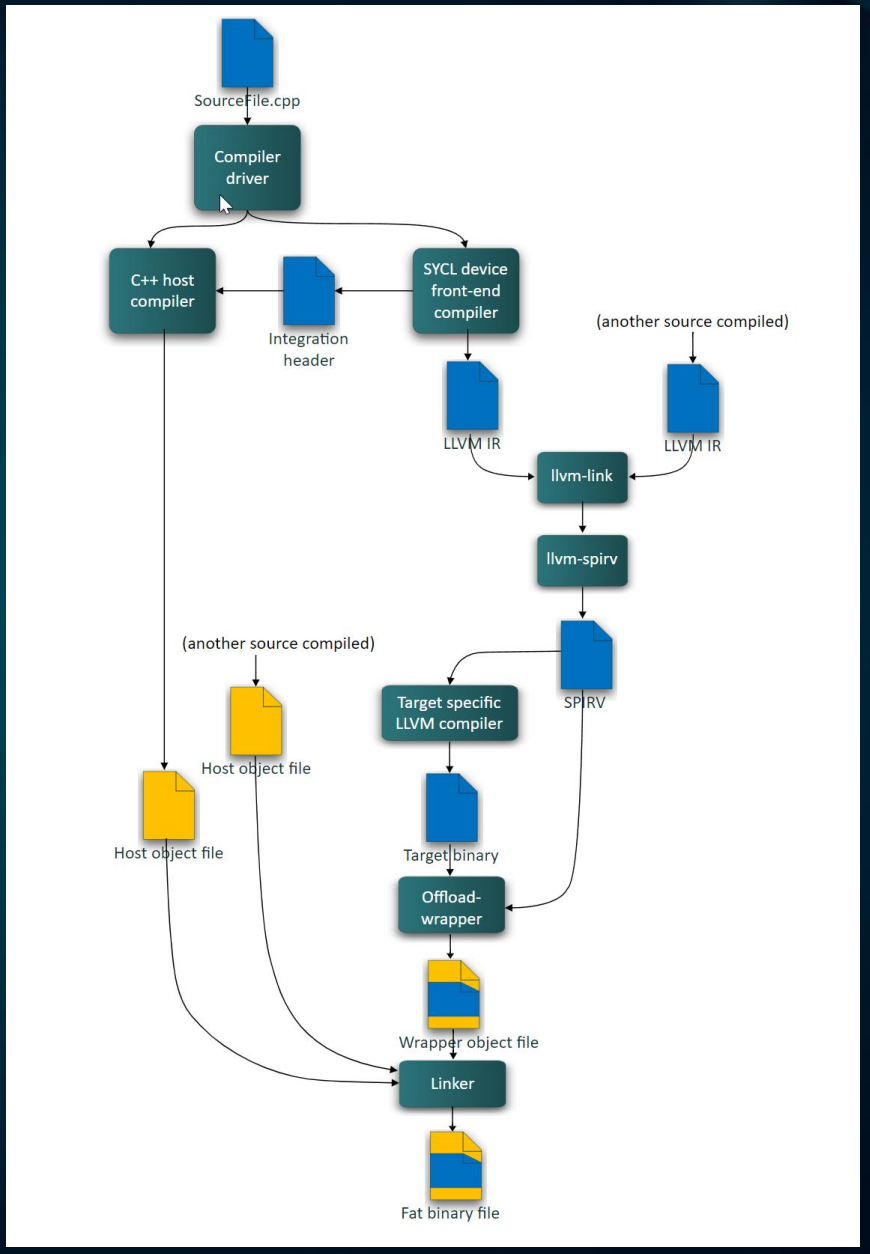
SYCL & DPC++ 编译器设计简介

Intel DPC++ Compiler

Compiler	Notes	Linux* Driver	Windows* Driver
Intel® DPC++ Compiler	A C++ and Khronos SYCL* compiler with a Clang front-end.	`dpcpp`	`dpcpp` (`clang` compatible) `dpcpp-cl` (`clang-cl` compatible)
Intel® C++ Compiler	A C++ compiler with a Clang front-end, supporting OpenMP* offload.	`icx` for C `icpx` for C++	`icx`

应用程序构建流程

- 前端
  - 解析输入源，“概述”代码的设备部分，对设备代码应用额外的限制；
  - 为设备代码生成 LLVM IR 和提供运行时库的内核名称、参数顺序和数据类型的“集成标头”。
- 中端
  - 转换初始 LLVM IR 以供后端使用。
  - 例如： LLVM IR → SPIR-V 转换器
- 后端
  - 生成本机“设备”代码。
  - 编译时（在提前编译场景中）或在运行时（在即时编译场景中）被调用。



转存SPIR-V

Offset	Name	Virtual Size	Virtual Address	RVA	Pointer
00000208	.text	0x00011961 (70.3 KB)	0x0000000040001000	0x00001000	0x0000
00000230	.text.st	0x00001043 (4.6 KB)	0x0000000040013000	0x00013000	0x0001
00000280	.rdata	0x0000571C (21.7 KB)	0x0000000040015000	0x00015000	0x0001
000002F8	.data	0x00002971 (10.3 KB)	0x0000000040001B000	0x0001B000	0x0001
00000398	.pdata	0x00000B04 (2.7 KB)	0x0000000040001E000	0x0001E000	0x0001
00000460	.idata	0x00002274 (8.6 KB)	0x0000000040001F000	0x0001F000	0x0001
00000550	.tgtsym	0x000001C2 (450 bytes)	0x0000000040022000	0x00022000	0x0001
00000668	__CLANG_	0x0001B8F3 (110.2 KB)	0x0000000040023000	0x00023000	0x0001
000007A8	.00cfg	0x00000151 (337 bytes)	0x0000000040003F000	0x0003F000	0x0003
00000910	_RDATA	0x00000953 (2.3 KB)	0x0000000040040000	0x00040000	0x0003
00000AA0	.reloc	0x00000681 (1.6 KB)	0x0000000040041000	0x00041000	0x0003

3.1. Magic Number

Magic number for a SPIR-V module.

Tip **Endianness:** A module is defined as a stream of words, not a stream of bytes. However, if stored as a stream of bytes (e.g., in a file), the magic number can be used to deduce what endianness to apply to convert the byte stream back to a word stream.

Magic Number
0x07230203

Hex Dump View

```
0001CBE0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0001CBF0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0001CC00  0302 2307 0001 0100 0E00 0600 4509 0000  ..#.........E...
0001CC10  0000 0000 1100 0200 0400 0000 1100 0200  ................
0001CC20  0500 0000 1100 0200 0600 0000 1100 0200  ................
0001CC30  0B00 0000 1100 0200 2600 0000 1100 0200  ........&.......
0001CC40  FD15 0000 0A00 0900 5350 565F 494E 5445  ........SPV_INTE
0001CC50  4C5F 6F70 7469 6D69 7A61 7469 6F6E 5F68  L_optimization_h
0001CC60  696E 7473 0000 0000 0A00 0A00 5350 565F  ints........SPV_
0001CC70  4B48 525F 6E6F 5F69 6E74 6567 6572 5F77  KHR_no_integer_w
0001CC80  7261 705F 6465 636F 7261 7469 6F6E 0000  rap_decoration..
0001CC90  0B00 0500 0100 0000 4F70 656E 434C 2E73  ........OpenCL.s
0001CCA0  7464 0000 0B00 0500 0200 0000 5350 4952  td..........SPIR
0001CCB0  562E 6465 6275 6700 0E00 0300 0200 0000  V.debug.........
0001CCC0  0200 0000 0F00 1300 0600 0000 1200 0000  ................
0001CCD0  5F5A 5453 4E32 636C 3473 7963 6C36 6465  _ZTSN2cl4sycl6de
0001CCE0  7461 696C 3139 5F5F 7066 5F6B 6572 6E65  tail19__pf_kerne
0001CCF0  6C5F 7772 6170 7065 7249 5A34 6D61 696E  l_wrapperIZ4main
0001CD00  4555 6C54 5F45 3232 5F32 3145 4500 0000  EUlT_E22_21EE...
0001CD10  0F00 0900 0600 0000 4200 0000 5F5A 5453  ........B..._ZTS
0001CD20  5A34 6D61 696E 4555 6C54 5F45 3232 5F32  Z4mainEUlT_E22_2
```

Clang 前端中的 SYCL 支持

- 设备代码概述
  - 负责识别和概述单一来源中的“设备代码”。
- SYCL 内核函数对象（仿函数或 lambda）底层化
  - 为 SYCL 内核创建了一个 OpenCL 内核函数接口。
- 设备代码诊断
  - 对设备代码实施语言限制。
- 集成标头生成
  - 发出的信息是使用 OpenCL API 绑定SYCL代码的主机和设备部分所需要的。

Clang 前端中的 SYCL 支持

```
int foo(int x) { return ++x; }
int bar(int x) { throw std::exception{"CPU code only!"}; }
...
using namespace cl::sycl;
queue Q;
buffer<int, 1> a{range<1>{1024}};
Q.submit([&](handler& cgh) {
      auto A = a.get_access<access::mode::write>(cgh);
      cgh.parallel_for<init_a>(range<1>{1024}, [=](id<1> index) {
        A[index] = index[0] * 2 + foo(42);
      });
    }
...
```

Clang 前端中的 SYCL 支持

```
// SYCL kernel is defined in SYCL headers:
template <typename KernelName, typename KernelType/*, ...*/>
__attribute__((sycl_kernel)) void sycl_kernel_function(KernelType KernelFuncObj) {
  // ...
  KernelFuncObj();
}

// Generated OpenCL kernel function
__kernel KernelName(global int* a) {
  KernelType KernelFuncObj; // Actually kernel function object declaration
  // doesn't have a name in AST.
  // Let the kernel function object have one captured field - accessor A.
  // We need to init it with global pointer from arguments:
  KernelFuncObj.A.__init(a);
  // Body of the SYCL kernel from SYCL headers:
  {
    KernelFuncObj();
  }
}
```

设备代码单独链接

- 用户将源代码分成四个文件：dev_a.cpp、dev_b.cpp、host_a.cpp和host_b.cpp，其中只有dev_a.cpp和dev_b.cpp包含设备代码。

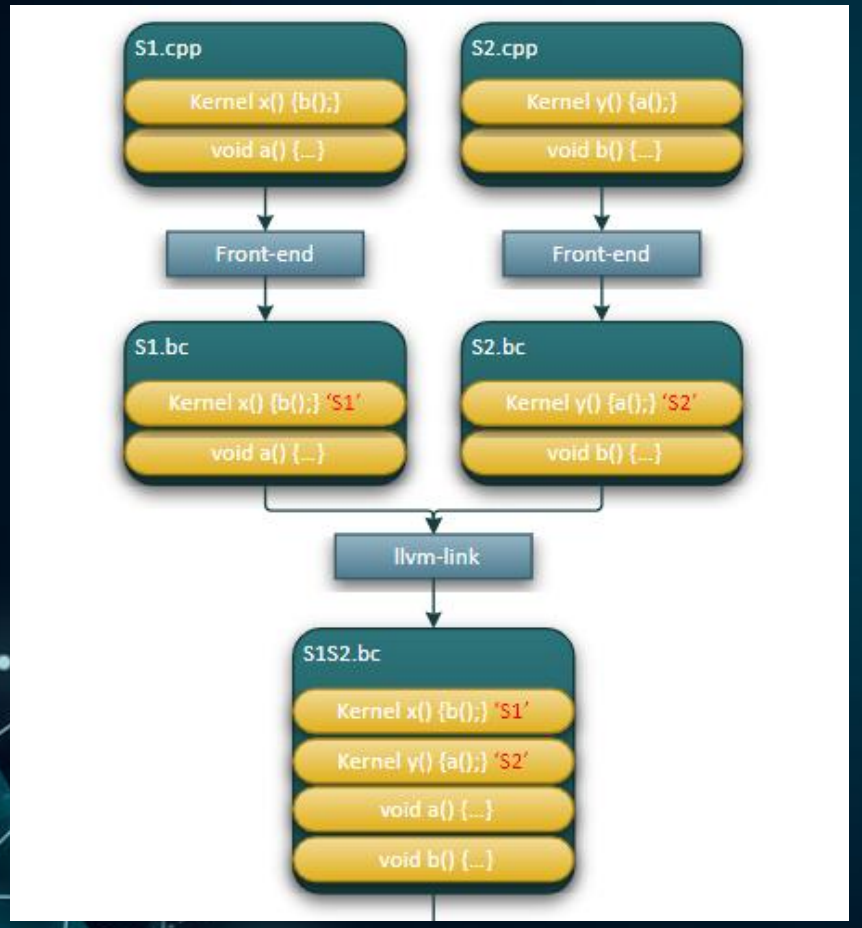


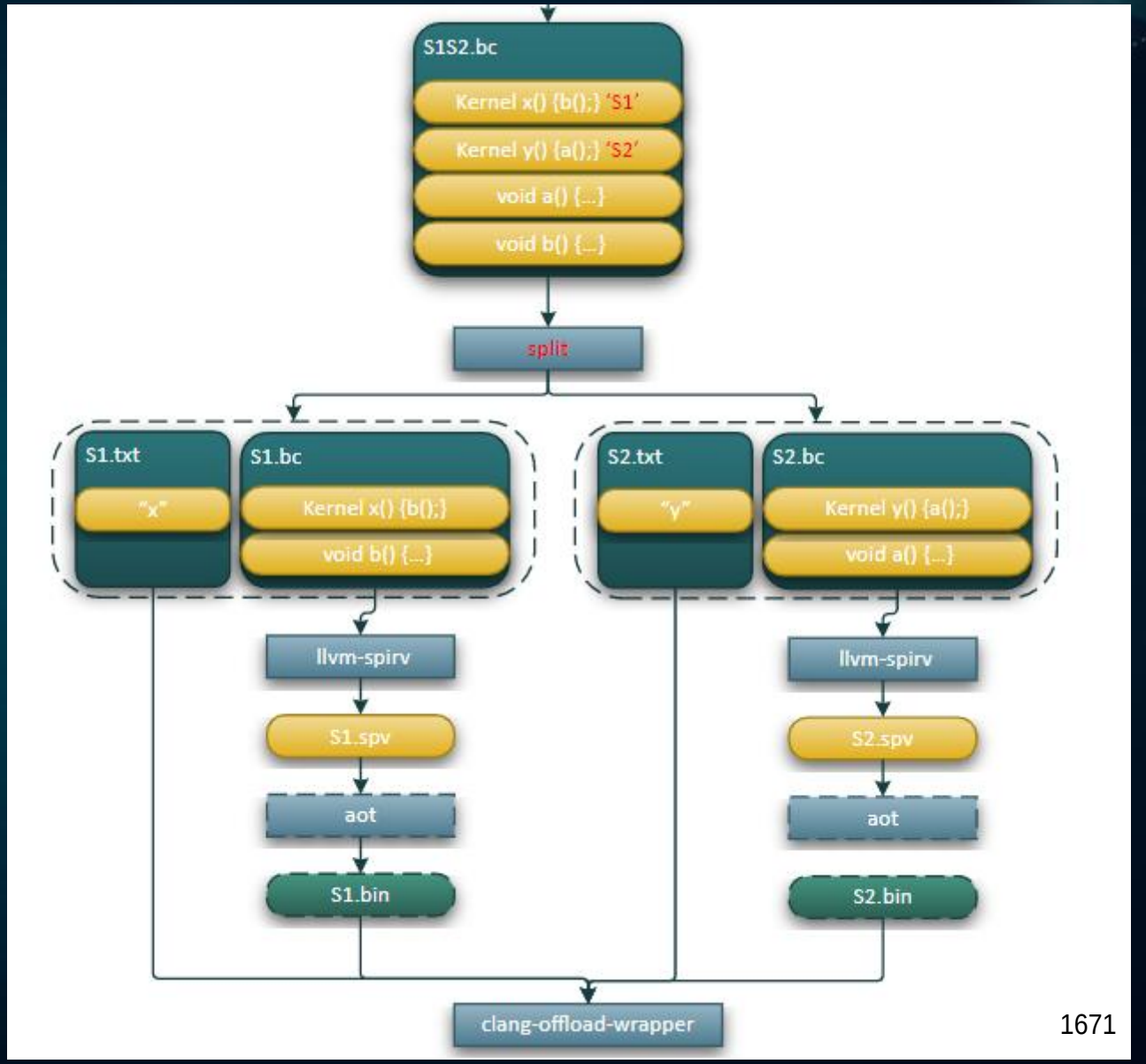
设备代码拆分

- 将所有设备代码放入单个 SPIR-V 模块在以下情况下效果不佳：
  - 定义了数千个内核，其中只有一小部分在运行时使用。将它们全部放在一个 SPIR-V 模块中会显著增加 JIT 时间。
  - 设备代码可以专门用于不同的设备。例如，只有在 FPGA 上才能执行的内核应该仅能够适用于 FPGA 的扩展。即使从未在其他设备上调用此特定内核，这也会导致其他设备上的 JIT 编译失败。

- 为了解决这些问题，编译器可以将单个模块拆分为更小的模块：
  - 根据源代码生成单独的模块 （翻译单元）
  - 根据单个内核生成单独的模块

设备代码拆分


S1.cpp
Kernel x() {b();}
void a() {...}
S2.cpp
Kernel y() {a();}
void b() {...}
Front-end
Front-end
S1.bc
Kernel x() {b();} 'S1'
void a() {...}
S2.bc
Kernel y() {a();} 'S2'
void b() {...}
llvm-link
S1S2.bc
Kernel x() {b();} 'S1'
Kernel y() {a();} 'S2'
void a() {...}
void b() {...}

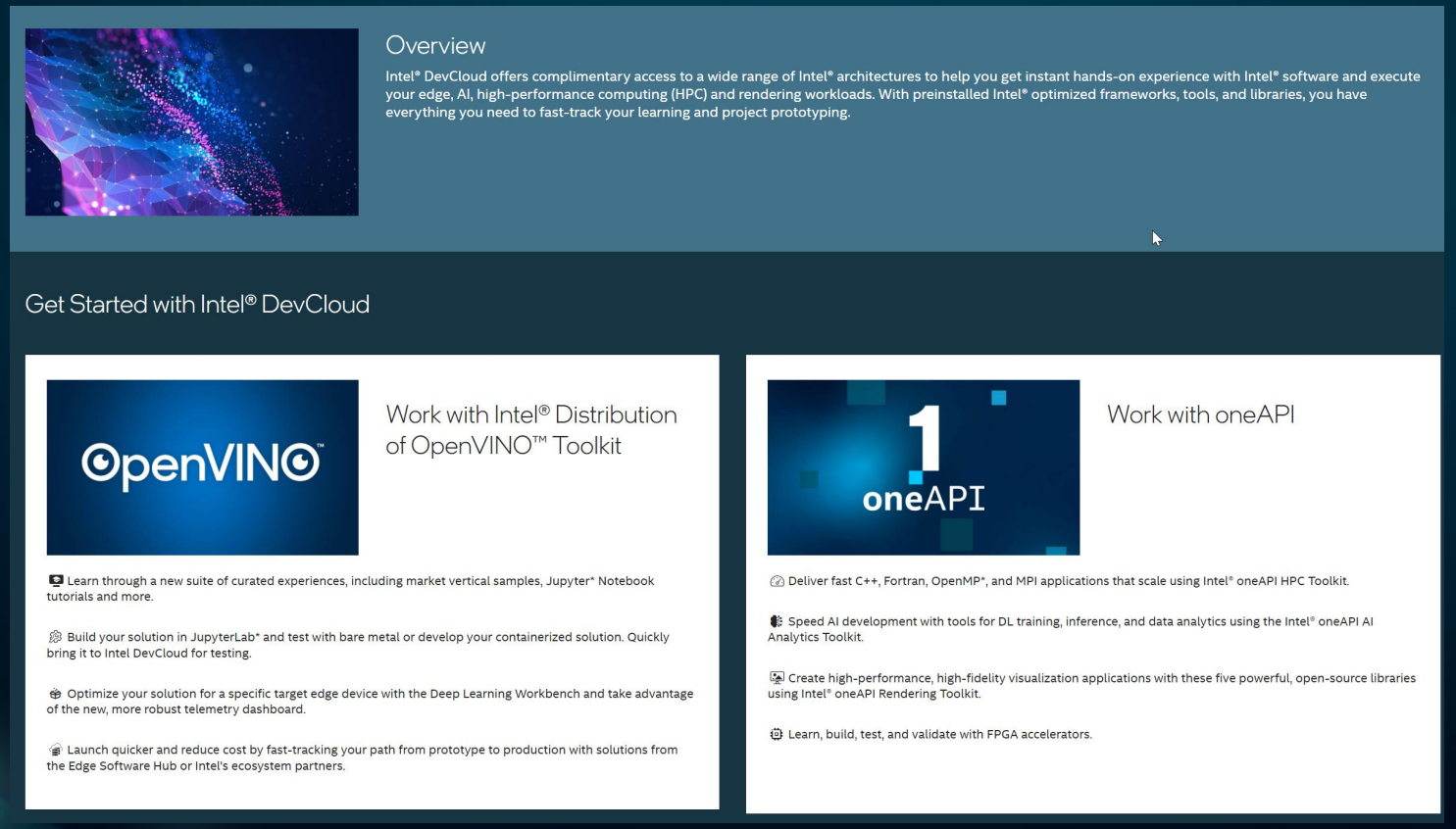

S1S2.bc
Kernel x() {b();} 'S1'
Kernel y() {a();} 'S2'
void a() {...}
void b() {...}
split
S1.txt
"x"
S1.bc
Kernel x() {b();}
void b() {...}
S2.txt
"y"
S2.bc
Kernel y() {a();}
void a() {...}
llvm-spirv
llvm-spirv
S1.spv
S2.spv
aot
aot
S1.bin
S2.bin
clang-offload-wrapper

04

DevCloud中学习SYCL & DPC++

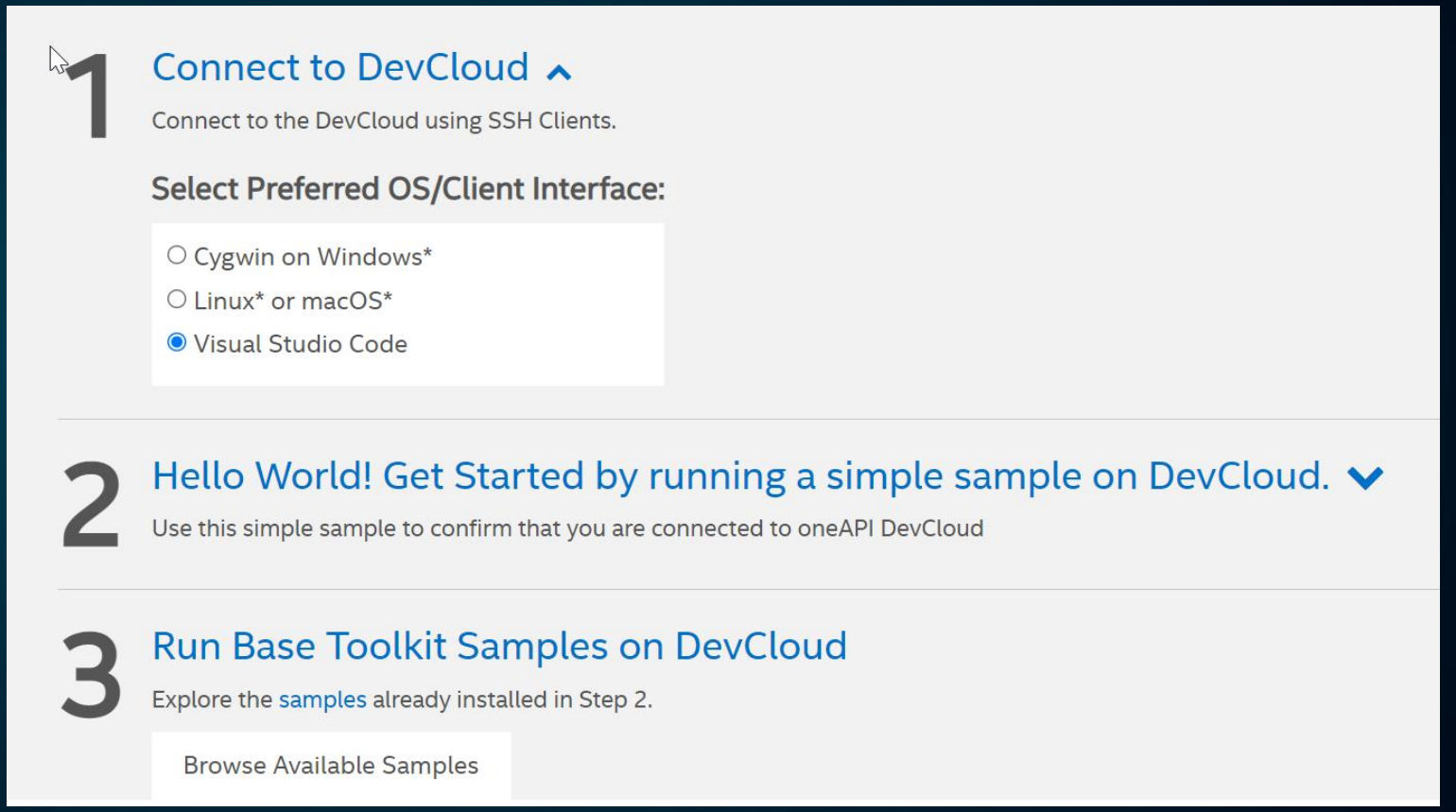
DevCloud简介

- DevCloud是Intel提供的免费访问各种Intel架构的云平台，它可以让使用者快速动手体验和学习各种优化框架、工具和库。
  - 访问链接：https://www.intel.com/content/www/us/en/developer/tools/devcloud/overview.html



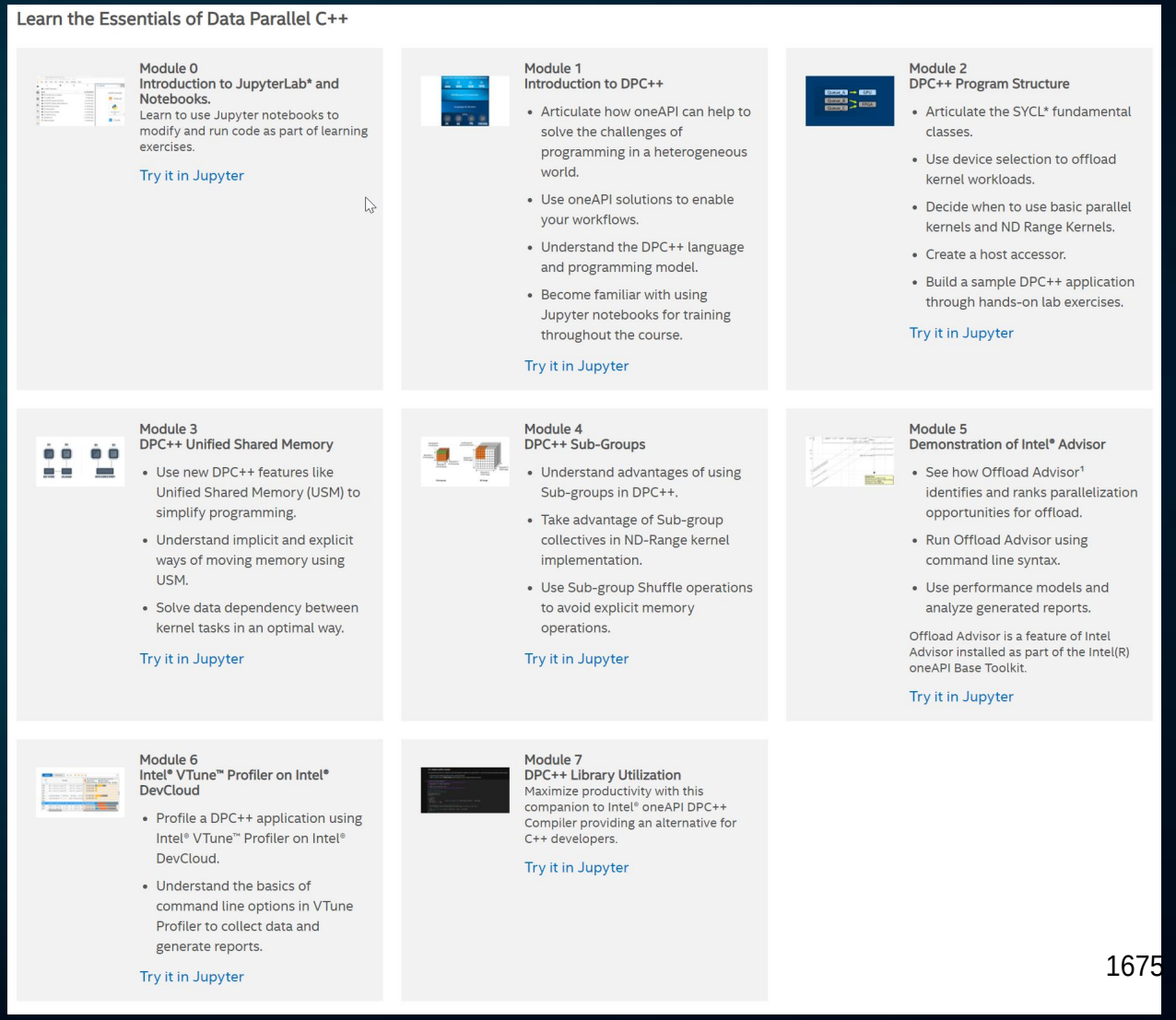
通过ssh连接DevCloud

- 连接教程详细；
- 操作过程操作简单；
- 编程自由度高；
- 推荐经过学习后实验使用。



通过jupyter notebook使用DevCloud

- 非常详尽的学习教程：
  - 教学循序渐进；
  - 文字和图片描述DPC++方方面面；
  - 附带可修改和运行代码示例。
- 强烈推荐初学者使用。



通过jupyter notebook使用DevCloud

Launcher | Unified_Shared_Memory.ipyr

Markdown

1. Inspect the code cell below and click run ▶ to save the code to file.

2. Next run ▶ the cell in the **Build and Run** section below the code to compile and execute the code.

[1]:
```
%%writefile lab/usm.cpp
//==============================================================
// Copyright © 2020 Intel Corporation
//
// SPDX-License-Identifier: MIT
// =============================================================
#include <CL/sycl.hpp>
using namespace sycl;

static const int N = 16;

int main() {
  queue q;
  std::cout << "Device : " << q.get_device().get_info<info::device::name>()
            << std::endl;

  //# USM allocation using malloc_shared
  int *data = malloc_shared<int>(N, q);

  //# Initialize data array
  for (int i = 0; i < N; i++) data[i] = i;

  //# Modify data array on device
  q.parallel_for(range<1>(N), [=](id<1> i) { data[i] *= 2; }).wait();

  //# print output
  for (int i = 0; i < N; i++) std::cout << data[i] << std::endl;
  free(data, q);
  return 0;
}
```

```
Overwriting lab/usm.cpp
```

Build and Run

Select the cell below and click run ▶ to compile and execute the code:

```
! chmod 755 q; chmod 755 run_usm.sh; if [ -x "$(command -v qsub)" ]; then ./q run_usm.sh; else ./run_usm.sh; fi
```

```
Job has been submitted to Intel(R) DevCloud and will execute soon.

 If you do not see result in 60 seconds, please restart the Jupyter kernel:
 Kernel -> 'Restart Kernel and Clear All Outputs...' and then try again

Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
1849943.v-qsvr-1           ...ub-singleuser u123998         00:00:12 R jupyterhub
1849944.v-qsvr-1           run_usm.sh       u123998                0 Q batch

Waiting for Output ████████████████████████████████████████████████

TimeOut 60 seconds: Job is still queued for execution, check for output file later (run_usm.sh.o1849944)

 Done!

################################################################################
#      Date:           Sat 12 Feb 2022 04:24:19 AM PST
#    Job ID:           1849944.v-qsvr-1.aidevcloud
#      User:           u123998
# Resources:           neednodes=1:gpu:ppn=2,nodes=1:gpu:ppn=2,walltime=06:00:00
################################################################################

## u123998 is compiling DPCPP_Essentials Module3 -- DPCPP Unified Shared Memory - 1 of 4 usm.cpp
Device : Intel(R) UHD Graphics P630 [0x3e96]
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30

################################################################################
# End of output for job 1849944.v-qsvr-1.aidevcloud
# Date: Sat 12 Feb 2022 04:24:40 AM PST
################################################################################

Job Completed in 60 seconds.
```

谢谢聆听

CPP-Summit 2022



杨文波

Boolan 资深咨询师

近20年软件研发经验，历任飞利浦研发中心、NXP恩智浦半导体、Shure舒尔中国研发中心嵌入式软件研发工程师、架构师、开发团队主管，多个产品获得多项技术和专业奖项，在世界各地以极高的可靠性和性能服务各类型高端客户。有丰富的从驱动到内核、一直到上层应用的嵌入式软件研发经验，对现代C++、设计模式、软件设计、安全编码有深入研究。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

杨文波

使用模板元编程优化嵌入式设计

Improve Your Embedded Design with Affordable Template Meta Programming

议程

1 **嵌入式设计与负担得起的模板元编程**
听众、嵌入式设计的特点、模板元编程、负担得起

2 **从硬件到抽象**
底层驱动、类型安全、物理量建模、常见问题

3 **总结与展望**
标准、库、未来

01

嵌入式设计与模板元编程

路线图

你在哪?

- 0. rename .c to .cpp, recompile using C++ compiler
- 1. C++ OOP
- 2. C++ TMP
- 3. C++ MP

Effective C++…

- 0. C
- 1. OOP
- 2. Template
- 3. The STL
- 4. constexpr
- …

Effective C++…

- 0. C
- 1. OOP
- 2. Template
- 3. The STL
- 4. constexpr
- …

不学操缦 不能安弦
不学博依 不能安《诗》
不学杂服 不能安礼
不兴其艺 不能乐学

2.1

逃离 #ifdef 地狱

#ifdef 地狱

```
#if MBED_CONF_CELLULAR_PRESENT
void CellularInterface::set_default_parameters()
{
    /* CellularInterface is expected to attempt to work without any parameters - we
     * will try, at least.
     */
#ifdef MBED_CONF_NSAPI_DEFAULT_CELLULAR_APN
#ifndef MBED_CONF_NSAPI_DEFAULT_CELLULAR_USERNAME
#define MBED_CONF_NSAPI_DEFAULT_CELLULAR_USERNAME NULL
#endif
#ifndef MBED_CONF_NSAPI_DEFAULT_CELLULAR_PASSWORD
#define MBED_CONF_NSAPI_DEFAULT_CELLULAR_PASSWORD NULL
#endif
    set_credentials(MBED_CONF_NSAPI_DEFAULT_CELLULAR_APN, MBED_CONF_NSAPI_DEFAULT_CELLULAR_USERNAME,
MBED_CONF_NSAPI_DEFAULT_CELLULAR_PASSWORD);
#endif
#ifdef MBED_CONF_NSAPI_DEFAULT_CELLULAR_SIM_PIN
    set_sim_pin(MBED_CONF_NSAPI_DEFAULT_CELLULAR_SIM_PIN);
#endif
#ifdef MBED_CONF_NSAPI_DEFAULT_CELLULAR_PLMN
    set_plmn(MBED_CONF_NSAPI_DEFAULT_CELLULAR_PLMN);
#endif
}
#endif // MBED_CONF_CELLULAR_PRESENT
```

#ifdef 地狱

```
void LlInitBbInit(void)
{
 //.....;
 #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_0)
    #ifdef INIT_CENTRAL
      LlExtScanMasterInit();
      LlExtInitMasterInit();
      LlPhyMasterInit();
      #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_2)
        LlCisMasterInit();
        LlBisMasterInit();
        LlPowerControlInit();
      #endif
    #else
      #ifdef INIT_OBSERVER
        LlExtScanMasterInit();
        #if (BT_VER >= LL_VER_BT_CORE_SPEC_5_2)
          LlBisMasterInit();
        #endif
      #endif
    #endif
  //......;
```

>=

```
void LlInitBbInit(void)
{
  LlExtScanMasterInit();
  LlExtInitMasterInit();
  LlPhyMasterInit();
}

void LlInitBbInit(void)
{
  LlExtScanMasterInit();
  LlExtInitMasterInit();
  LlPhyMasterInit();

  LlExtScanMasterInit();
  LlExtInitMasterInit();
  LlPhyMasterInit();
}

void LlInitBbInit(void)
{
  LlExtScanMasterInit();
  LlBisMasterInit();
}

void LlInitBbInit(void)
{
  LlExtScanMasterInit();
  LlBisMasterInit();
}
```

改进思路

Macros are almost never necessary in C++. Use const (§5.4) or enum (§4.8) to define manifest constants, inline (§7.1.1) to avoid function-calling overhead, templates (Chapter 13) to specify families of functions and types, and namespaces (§8.2) to avoid name clashes.

— TCPPPL 3rd Chapter 1.

基于策略

```
// TODO: 强类型枚举
constexpr int Central = 0;
constexpr int Observer = 1;

template <int BT_VER, int INIT_ROLE> struct BleInitDriverChoice;

struct DeviceVer50_Central {
  void init() {
     std::cout << "50 Center";
  }
};
struct DeviceVer50_Observer {
  void init() { std::cout << "50 Observer"; }
};
struct DeviceVer52_Central {
  void init() { std::cout << "52 Central"; }
};
struct DeviceVer52_Observer {
  void init() { std::cout << "52 Observer";}
};
```

```
// 全特化/显式实例化可以放到 .cpp 文件中，解耦 + 避免膨胀
template <> struct BleInitDriverChoice< 50, Central > {
  typedef DeviceVer50_Central type;
};

template <> struct BleInitDriverChoice< 50, Observer > {
  typedef DeviceVer50_Observer type;
};

template <> struct BleInitDriverChoice< 52, Central > {
  typedef DeviceVer52_Central  type;
};

template <> struct BleInitDriverChoice< 52, Observer > {
  typedef DeviceVer52_Observer type;
};

struct BleDriver {
  typedef BleInitDriverChoice< 52, Central >::type type;
};


int main()
{
  BleDriver::type bleDriver;
  bleDriver.init();
}
```

2.2

深藏 void*

void*

Try to avoid `void*`, ···, except deep within the implementation of some function or class....

— TCPPPL 3rd Chapter 1.

...every object is accessed according to the type with which it was defined

— Type-and-resource safety in modern C++

I.4: Make interfaces precisely and strongly typed

— C++ CG

```
/**
 * Brief       Set instance owner for the SPI/I2C peripheral.
 *
 * Parameter   instance  The instance.
 * Parameter   object    The object.
 */
void object_owner_spi2c_set(int instance, void *object)
{
    if (instance < SPI2C_INSTANCES) {
        nordic_spi2c_owners[instance] = object;
    }
}

/**
 * Brief       Get instance owner for the SPI/I2C peripheral.
 *
 * Parameter   instance  The instance.
 *
 * Return      Pointer to the object owning the instance.
 */
void * object_owner_spi2c_get(int instance)
{
    void *object = (void *) 0xFFFFFFFF;

    if (instance < SPI2C_INSTANCES) {
        object = nordic_spi2c_owners[instance];
    }

    return object;
}
```

不用 void*?

The only reasons we don't claim complete type safety is

Our rules have not yet been completely and systematically enforced on large code bases; our tools needs to mature.

We need a few (isolated and explicit) type violations to effectively deal with hardware (e.g.,casts to turn machine addresses into typed pointers and the ability to input into uninitialized memory)

- *— A brief introduction to C++'s model for type- and resource-safety*

模板 + void*

- 对外类型安全
- 对内泛型和设计复用
- 内联

```
class GenericPtrSpi2c {
   protected:
      GenericPtrSpi2c();
      ~GenericPtrSpi2c();
      void object_owner_spi2c_set(int instance, void *object);
      void* object_owner_spi2c_get(int instance);
   private:
      //...;
};

template<typename T>
class Spi2c<T*>: private GenericPtrSpi2c
{
public:
  void object_owner_spi2c_set(int instance, T *objectPtr) {
      GenericPtrSpi2c::object_owner_spi2c_set(instance, objectPtr);
  }

  T*  object_owner_spi2c_get() {
      return static_cast<T*>(
              GenericPtrSpi2c::object_owner_spi2c_get(int instance)
              );
  }
};
```

模板 + void*

I.4

… …

Alternative: Often, a template parameter can eliminate the void turning it into a T* or T&. For generic code these Ts can be general or concept constrained template parameters.*

— C++ CG

```
class GenericPtrSpi2c {
   protected:
      GenericPtrSpi2c();
      ~GenericPtrSpi2c();
      void object_owner_spi2c_set(int instance, void *object);
      void* object_owner_spi2c_get(int instance);
   private:
      //...;
};

template<typename T>
class Spi2c<T*>: private GenericPtrSpi2c
{
public:
  void object_owner_spi2c_set(int instance, T *objectPtr) {
      GenericPtrSpi2c::object_owner_spi2c_set(instance, objectPtr);
  }

  T*  object_owner_spi2c_get() {
      return static_cast<T*>(
              GenericPtrSpi2c::object_owner_spi2c_get(int instance)
              );
  }
};
```

模板 + void*

… that compile-time type checking (i.e., the attempt to discover a correct type assignment) obviates the need to carry types at run-time, with an obvious gain in the efficiency in implementation

— Robin Milner, 1977

```
class GenericPtrSpi2c {
   protected:
      GenericPtrSpi2c();
      ~GenericPtrSpi2c();
      void object_owner_spi2c_set(int instance, void *object);
      void* object_owner_spi2c_get(int instance);
   private:
      //...;
};

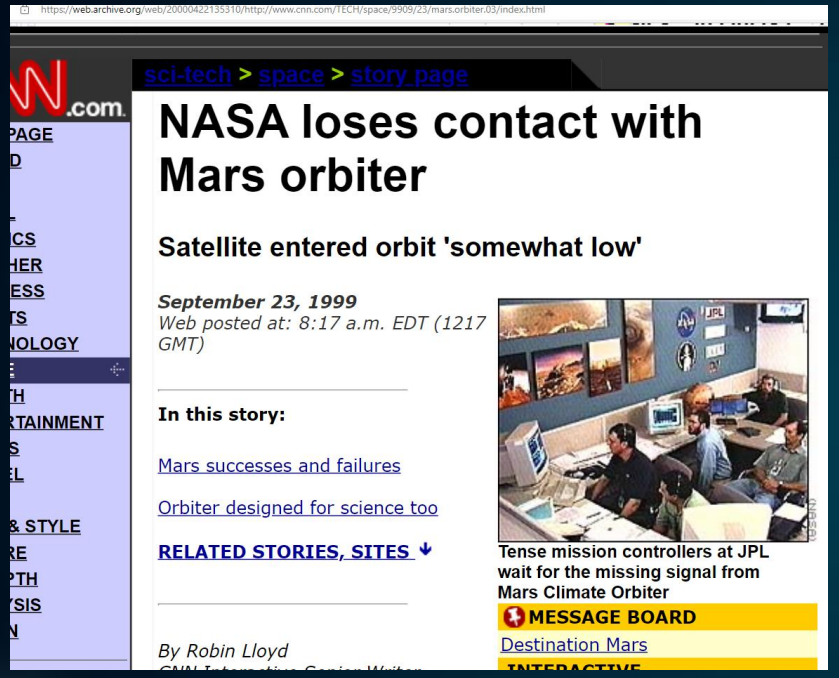
template<typename T>
class Spi2c<T*>: private GenericPtrSpi2c
{
public:
  void object_owner_spi2c_set(int instance, T *objectPtr) {
      GenericPtrSpi2c::object_owner_spi2c_set(instance, objectPtr);
  }

  T*  object_owner_spi2c_get() {
      return static_cast<T*>(
              GenericPtrSpi2c::object_owner_spi2c_get(int instance)
              );
  }
};
```

2.3

量纲安全

量纲安全



sci-tech > space > story page

NASA loses contact with Mars orbiter

Satellite entered orbit 'somewhat low'

September 23, 1999
Web posted at: 8:17 a.m. EDT (1217 GMT)

In this story:

Mars successes and failures

Orbiter designed for science too

RELATED STORIES, SITES

Tense mission controllers at JPL wait for the missing signal from Mars Climate Orbiter

MESSAGE BOARD

Destination Mars

By Robin Lloyd

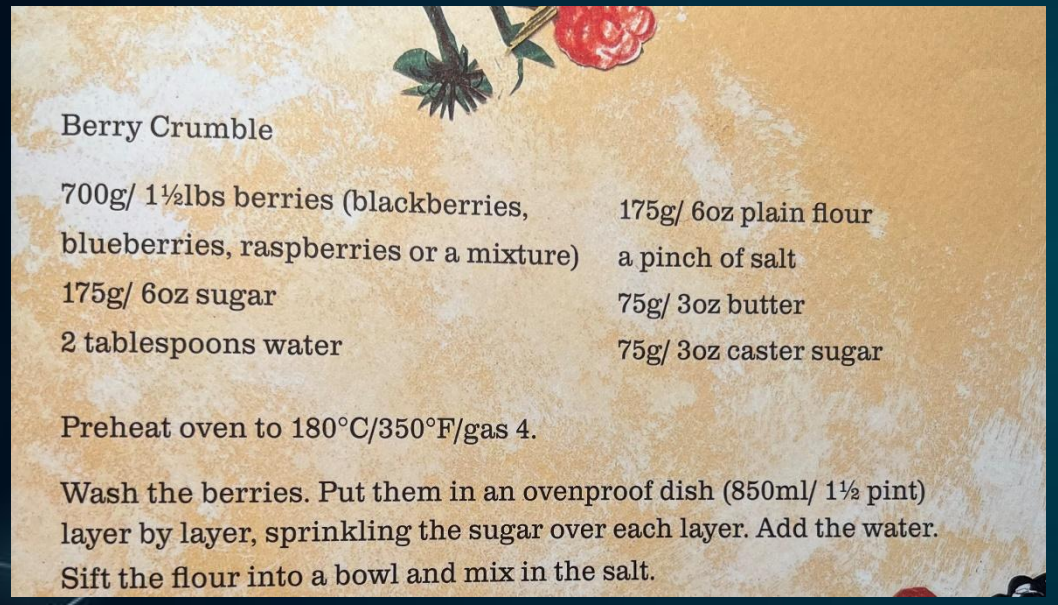
unfortunately, no signals were received from the spacecraft.

An investigation indicated that the failure resulted from a navigational error due to commands from Earth being sent in English units (in this case, pound-seconds) without being converted into the metric standard (Newton-seconds).

The error caused the orbiter to miss its intended orbit (87 to 93 miles or 140 to 50 kilometers) and to fall into the Martian atmosphere at approximately 35 miles (57 kilometers) in altitude and to disintegrate due to atmospheric stresses.

```cpp
class MPCController : public Controller {
 public:
  // ...;
  /**
   * @brief compute steering target and throttle/ brake based on current vehicle
   * status and target trajectory
   * @param localization vehicle location
   * @param chassis vehicle status e.g., speed, acceleration
   * @param trajectory trajectory generated by planning
   * @param cmd control command
   * @return Status computation status
   */
  common::Status ComputeControlCommand(
      const localization::LocalizationEstimate *localization,
      const canbus::Chassis *chassis, const planning::ADCTrajectory *trajectory,
      ControlCommand *cmd) override;
  // ...;
 protected:
  // the following parameters are vehicle physics related.
  // control time interval
  double ts_ = 0.0;
  // corner stiffness; front
  double cf_ = 0.0;
  // corner stiffness; rear
  double cr_ = 0.0;
  // distance between front and rear wheel center
  double wheelbase_ = 0.0;
  // mass of the vehicle
  double mass_ = 0.0;
  // ...;
  // limit steering to maximum theoretical lateral acceleration
  double max_lat_acc_ = 0.0;
  // ...;
```

量纲安全



Berry Crumble

700g/ 1½lbs berries (blackberries, blueberries, raspberries or a mixture)
175g/ 6oz sugar
2 tablespoons water
175g/ 6oz plain flour
a pinch of salt
75g/ 3oz butter
75g/ 3oz caster sugar

Preheat oven to 180°C/350°F/gas 4.

Wash the berries. Put them in an ovenproof dish (850ml/ 1½ pint) layer by layer, sprinkling the sugar over each layer. Add the water. Sift the flour into a bowl and mix in the salt.

```cpp
class MPCController : public Controller {
 public:
  // ...;
  /**
   * @brief compute steering target and throttle/ brake based on current vehicle
   * status and target trajectory
   * @param localization vehicle location
   * @param chassis vehicle status e.g., speed, acceleration
   * @param trajectory trajectory generated by planning
   * @param cmd control command
   * @return Status computation status
   */
  common::Status ComputeControlCommand(
      const localization::LocalizationEstimate *localization,
      const canbus::Chassis *chassis, const planning::ADCTrajectory *trajectory,
      ControlCommand *cmd) override;
  // ...;
 protected:
  // the following parameters are vehicle physics related.
  // control time interval
  double ts_ = 0.0;
  // corner stiffness; front
  double cf_ = 0.0;
  // corner stiffness; rear
  double cr_ = 0.0;
  // distance between front and rear wheel center
  double wheelbase_ = 0.0;
  // mass of the vehicle
  double mass_ = 0.0;
  // ...;
  // limit steering to maximum theoretical lateral acceleration
  double max_lat_acc_ = 0.0;
  // ...;
```

量纲安全

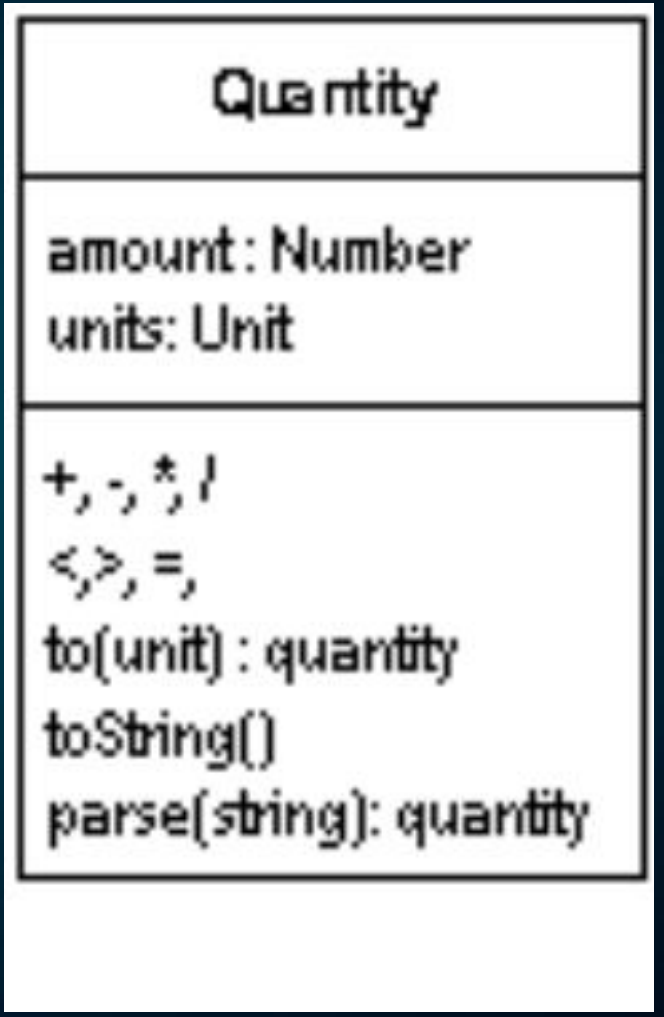
多数软件（仍然）忽略量纲：

```
double t;                       // 时间 – 秒
double a;                       // 加速度 – 米/秒²
double d;                       // 距离 – 米
...
std::cout << d/(t*t) - a;       // 可以
std::cout << d/t - a;           // 无意义，但编译可以通过
```

```
class MPCController : public Controller {
 public:
  // ...;
  /**
   * @brief compute steering target and throttle/ brake based on current vehicle
   * status and target trajectory
   * @param localization vehicle location
   * @param chassis vehicle status e.g., speed, acceleration
   * @param trajectory trajectory generated by planning
   * @param cmd control command
   * @return Status computation status
   */
  common::Status ComputeControlCommand(
      const localization::LocalizationEstimate *localization,
      const canbus::Chassis *chassis, const planning::ADCTrajectory *trajectory,
      ControlCommand *cmd) override;
  // ...;
 protected:
  // the following parameters are vehicle physics related.
  // control time interval
  double ts_ = 0.0;
  // corner stiffness; front
  double cf_ = 0.0;
  // corner stiffness; rear
  double cr_ = 0.0;
  // distance between front and rear wheel center
  double wheelbase_ = 0.0;
  // mass of the vehicle
  double mass_ = 0.0;
  // ...;
  // limit steering to maximum theoretical lateral acceleration
  double max_lat_acc_ = 0.0;
  // ...;
```

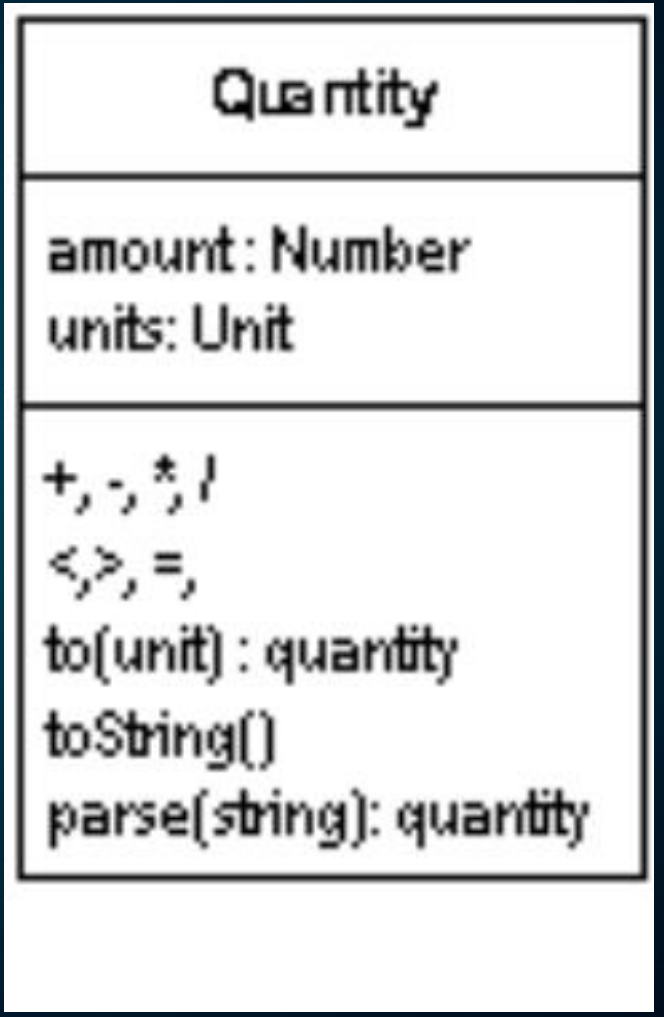
量纲安全

- Quantity Pattern
  - 数量+单位表示有量纲的值
  - OO
  - 动态绑定
  - 转换
  - 开销
  - 错误代价
  - 负担
  - 场景（人为规则）



物理量纲安全

- Quantity Pattern?
  - 数量+单位表示有量纲的值
  - OO
  - 动态绑定
  - 转换
  - 开销
  - 错误代价
  - 心智负担
  - 场景（自然规律）



物理量纲安全

观察

量纲类型由量纲指数决定：

$v = distance^{1}/time^{1} = distance^{1} * time^{-1}$

$a = distance^{1}/time^{2} = distance^{1} * time^{-2}$

$t = distance^{0} * time^{1}$

指数的每种组合都应当是一种不同的类型。

原则上，组合的数量是不受限的。

模板生成类型

物理量纲安全

观察

量纲类型由量纲指数决定：

$v = distance^1/time^1 = distance^1 * time^{-1}$

$a = distance^1/time^2 = distance^1 * time^{-2}$

$t = distance^0 * time^1$

指数的每种组合都应当是一种不同的类型

原则上，组合的数量是不受限的。

模板生成类型

```
template<int m, int d, int t>
class Units {
public:
    explicit Units(double initVal = 0):val(initVal){}
    double value() const{ return val;}
    // ...;
private:
    double val;
};

Units<1,0,0> m;
Units<1,0,0> n;
Units<0,1,0>d;
Units<0,0,1>t;

int main()
{
    m = n; // OK
    m = t; // 编译错误
}
```

物理量纲安全

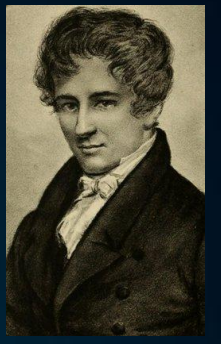
```
template<int m, int d, int t>
class Units {
public:
    explicit Units(double initVal = 0):val(initVal){}
    double value() const{ return val;}
    // ...;
private:
    double val;
};

Units<1,0,0> m;
Units<1,0,0> n;
Units<0,1,0>d;
Units<0,0,1>t;
```

```
template<int m, int d, int t>
Units<m, d, t> operator+(const Units<m, d, t>& lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(lhs);
    return result += rhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(double lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(rhs);
    return result *= lhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(const Units<m, d, t>& lhs,
                         double rhs)
{
    Units<m, d, t> result(lhs);
    return result *= rhs;
}
```



物理量纲安全

```
const Mass kilogram(1);
const Distance meter(1);
const Time second(1);

const Mass pound(kilogram/2.2);
const Time minute(60 * second);
const Distance inch(0.0254*meter);

Distance length(42 * inch);
```

```
template<int m, int d, int t>
Units<m, d, t> operator+(const Units<m, d, t>& lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(lhs);
    return result += rhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(double lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(rhs);
    return result *= lhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(const Units<m, d, t>& lhs,
                         double rhs)
{
    Units<m, d, t> result(lhs);
    return result *= rhs;
}
```

物理量纲安全

```
template<int m1, int d1, int t1,
         int m2, int d2, int t2>
Units<m1+m2, d1+d2, t1+t2>
operator*(const Units<m1, d1, t1>& lhs,
          const Units<m2, d2, t2>& rhs)
{
    typedef Units<m1+m2, d1+d2, t1+t2> ResultType;
    return ResultType(lhs.value() * rhs.value());
}

template<int m1, int d1, int t1,
         int m2, int d2, int t2>
Units<m1-m2, d1-d2, t1-t2>
operator/(const Units<m1, d1, t1>& lhs,
          const Units<m2, d2, t2>& rhs)
{
    typedef Units<m1-m2, d1-d2, t1-t2> ResultType;
    return ResultType(lhs.value() / rhs.value());
}
```

```
template<int m, int d, int t>
Units<m, d, t> operator+(const Units<m, d, t>& lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(lhs);
    return result += rhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(double lhs,
                         const Units<m, d, t>& rhs)
{
    Units<m, d, t> result(rhs);
    return result *= lhs;
}

template<int m, int d, int t>
Units<m, d, t> operator*(const Units<m, d, t>& lhs,
                         double rhs)
{
    Units<m, d, t> result(lhs);
    return result *= rhs;
}
```

物理量纲安全

- SI
  - 质量 M
  - 长度 D
  - 时间 T
  - 电流 C
  - 温度 K
  - 发光强度 I
  - 物质的量 A
- SI 派生
  - $N = kgms^{-2}$
  - $\Omega = m^2kgs^{-3}A^{-2}$

```
template<class T, int m, int d, int t, int q, int k, int i, int a>
class Units {
public:
    explicit Units(T initVal = 0) : val(initVal) {}
    T& value() { return val; }
    const T& value() const { return val; }
    ...
private:
        T val;
};

template<class T, int m1, int d1, int t1, int q1, int k1, int i1, int a1,
                  int m2, int d2, int t2, int q2, int k2, int i2, int a2>
Units<T, m1+m2, d1+d2, t1+t2, q1+q2, k1+k2, i1+i2, a1+a2>
operator*(const Units<T, m1, d1, t1, q1, k1, i1, a1>& lhs,
          const Units<T, m2, d2, t2, q2, k2, i2, a2>& rhs)
{
    typedef           Units<T, m1+m2, d1+d2, t1+t2, q1+q2, k1+k2, i1+i2, a1+a2>
    ResultType;
    return ResultType(lhs.value() * rhs.value());
}
```

物理量纲安全

无量纲 quantity

- 统一
- 类型兼容
- 偏特化

```
template<typename T>
class Units<T, 0, 0, 0, 0, 0, 0, 0> {
public:
//...
    Units(T initVal = 0): val(initVal) {}  // 允许隐式转换
    operator T() const { return val; }     // 到/从 类型为 T 的值

    Units& operator=(T newVal)             // 允许赋值
    { val = newVal; return *this; }        // 到/从类型为 T 的值
//…
Private:
    T val;
};
```

物理量纲安全

语言、库

- 支持分数指数
- 支持多单位系统
- 量纲生成
- 领域知识、公理

it is possible to define systems of units that have no base quantities at all

Pi theorem, independent quantities, parameters and return values of a transcendental function (e.g., exponential, logarithmic, or trigonometric functions) should be unitless

SIUNITS

<chrono>

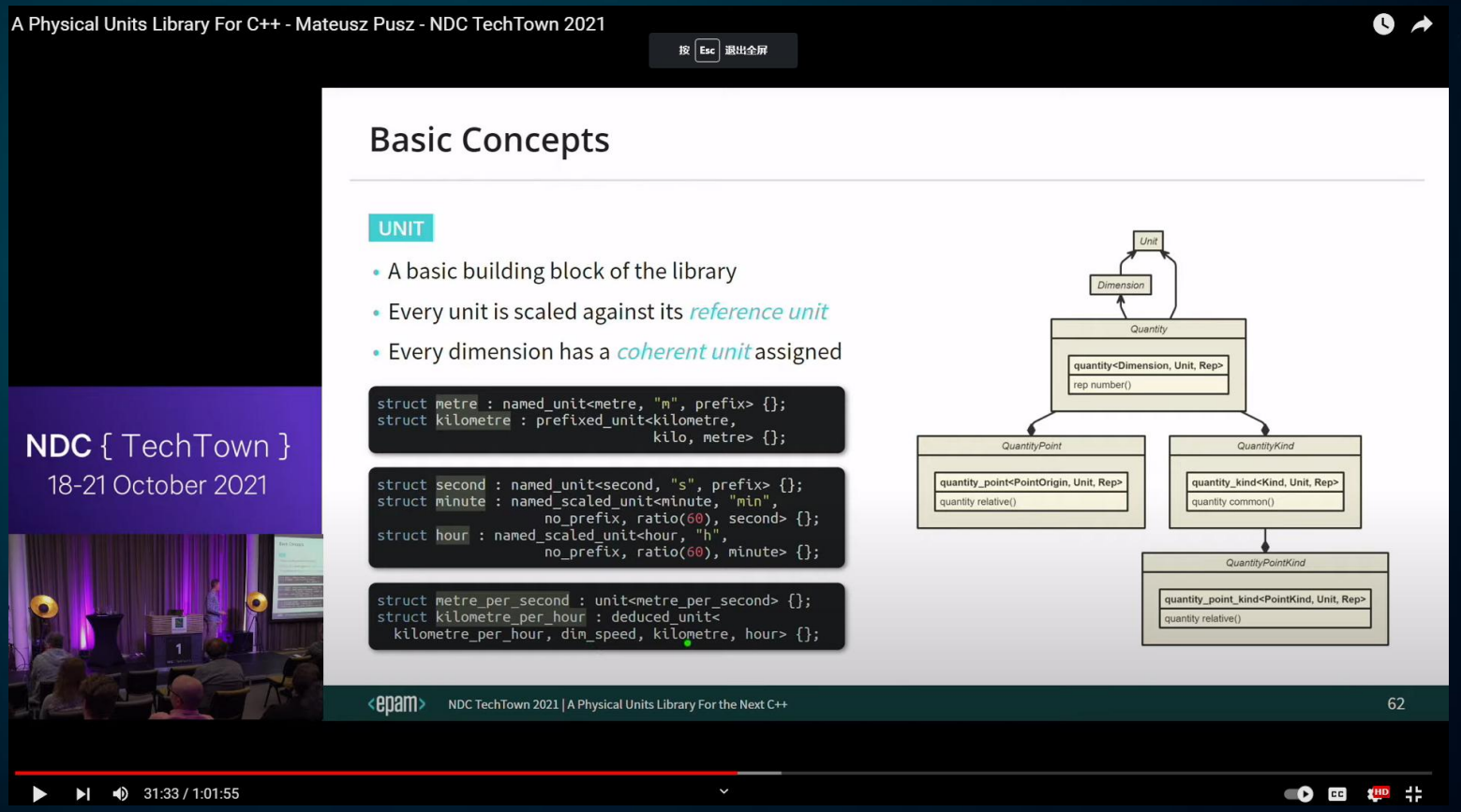
Boost.Units

P1935 mp-units *概念、CTTP、NTTP …*

物理量纲安全

语言、库

- 分数指数
- 多单位系统
- 量纲生成
- 领域知识、公理
- 推理



2.4

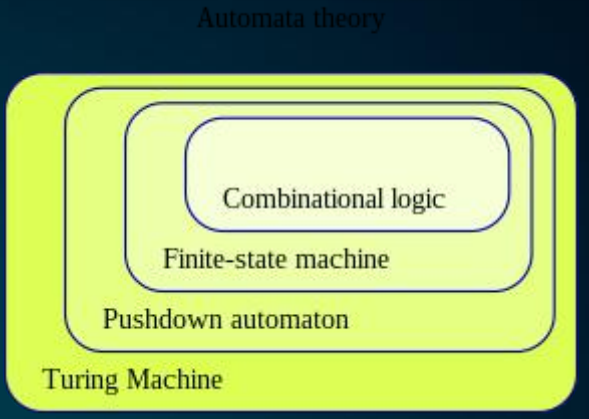
DSL 有限状态机

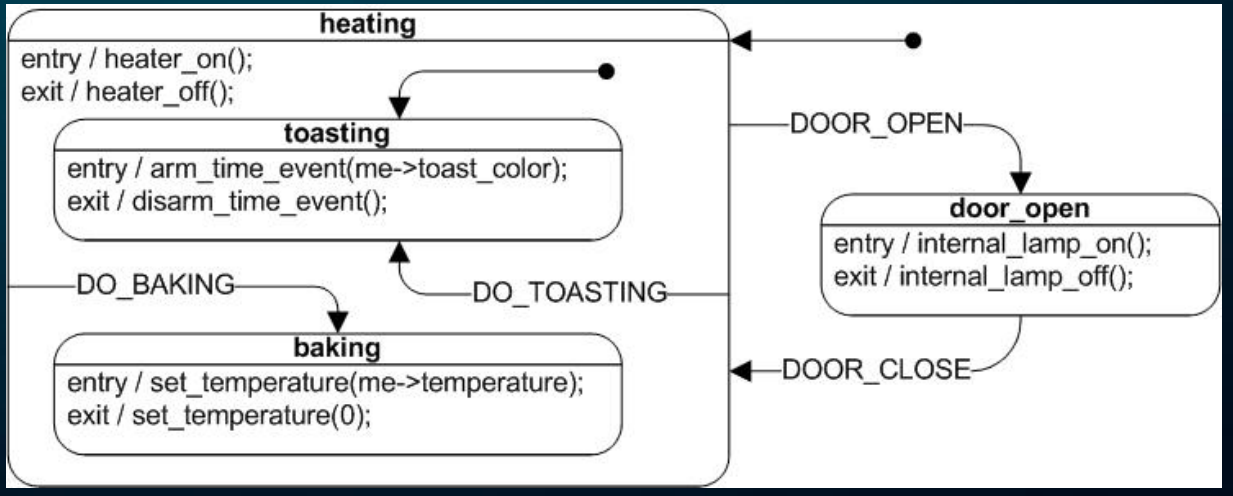
有限状态机

- 需求，领域建模
- 定义和理论(Moor & Mealy)
- 直观
  - 状态
  - 事件
  - 转换
  - 行动

State-transition table

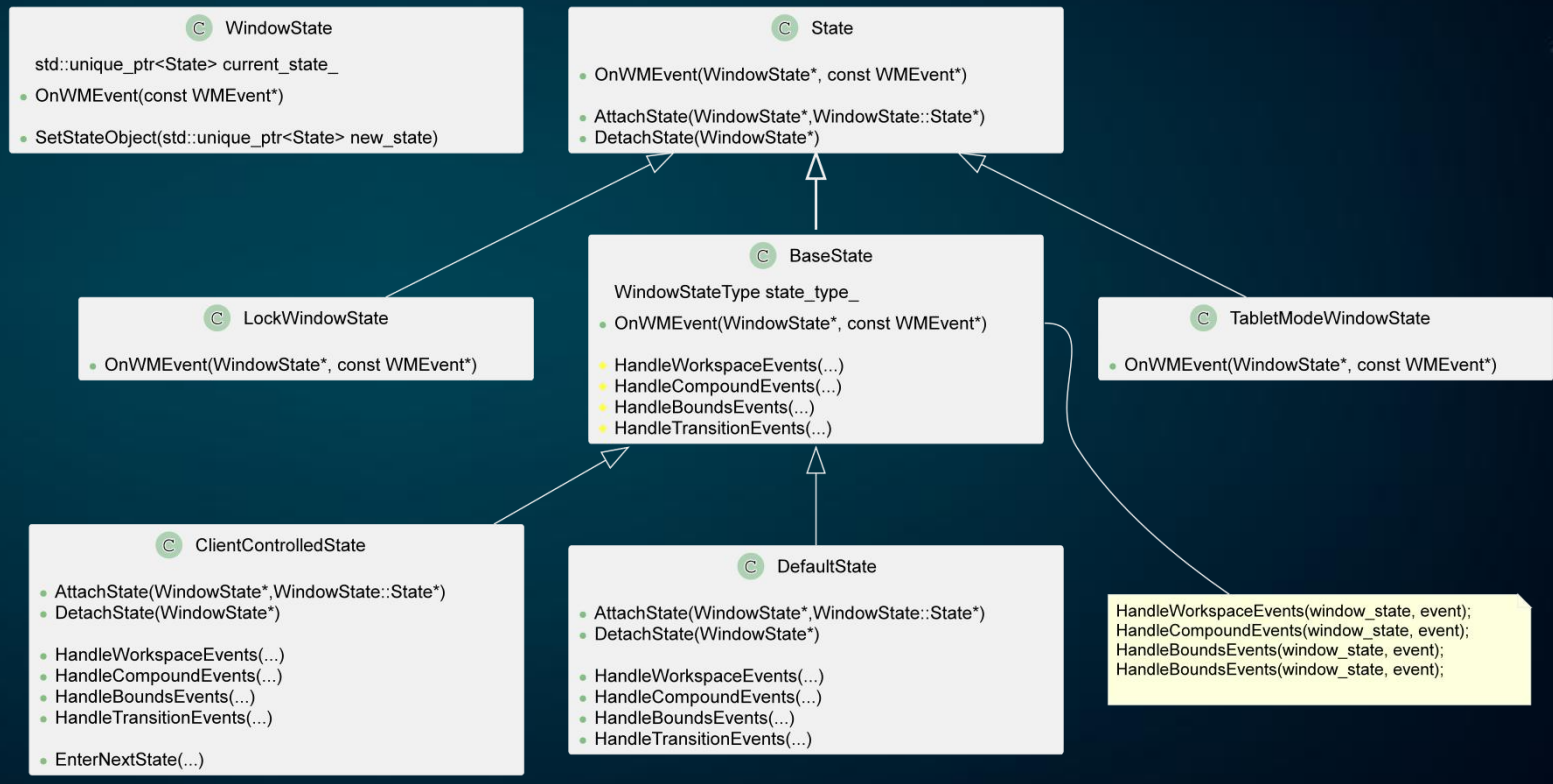
Input \ Current state	State A	State B	State C
Input X	...	...	...
Input Y	...	State C	...
Input Z	...	...	...





有限状态机

- 命令式 if/else or switch/case
  - 性能 开销 耦合 维护
- 面向对象 State Pattern
  - 性能 开销 耦合 维护
- std::variant, Visitor Pattern
  - 性能 开销 耦合 维护
  - C++17
- Boost: MSM, Statechart, SML
- 图灵完备…

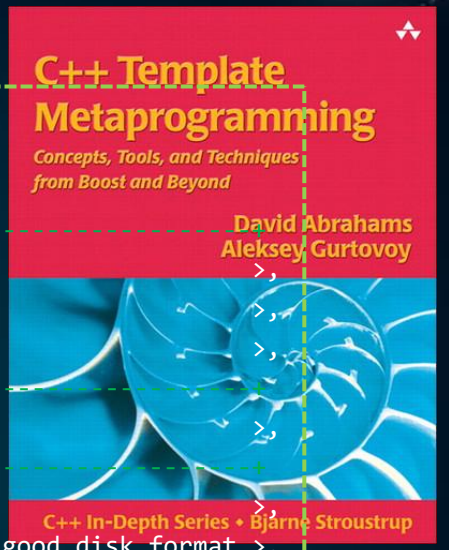


有限状态机

Boost.MSM

- 填表
- DSL
- 生成式

```
struct transition_table : mpl::vector<
//    Start     Event         Target      Action                      Guard
//  +---------+-------------+-----------+---------------------------+----------------------------+
a_row< Stopped , play       , Playing   , &player_::start_playback                                >,
a_row< Stopped , open_close , Open      , &player_::open_drawer                                   >,
 _row< Stopped , stop       , Stopped                                                            >,
//  +---------+-------------+-----------+---------------------------+----------------------------+
a_row< Open    , open_close , Empty     , &player_::close_drawer                                  >,
//  +---------+-------------+-----------+---------------------------+----------------------------+
a_row< Empty   , open_close , Open      , &player_::open_drawer                                   >,
  row< Empty   , cd_detected, Stopped   , &player_::store_cd_info   , &player_::good_disk_format >,
  row< Empty   , cd_detected, Playing   , &player_::store_cd_info   , &player_::auto_start       >,
//  +---------+-------------+-----------+---------------------------+----------------------------+
a_row< Playing , stop       , Stopped   , &player_::stop_playback                                 >,
a_row< Playing , pause      , Paused    , &player_::pause_playback                                >,
a_row< Playing , open_close , Open      , &player_::stop_and_open                                 >,
//  +---------+-------------+-----------+---------------------------+----------------------------+
a_row< Paused  , end_pause  , Playing   , &player_::resume_playback                               >,
a_row< Paused  , stop       , Stopped   , &player_::stop_playback                                 >,
a_row< Paused  , open_close , Open      , &player_::stop_and_open                                 >
//  +---------+-------------+-----------+---------------------------+----------------------------+
> {};
```



有限状态机

- 表现力
  - （图灵）完备
  - 层级和并发状态
  - UML 状态图 + Petri 网
- 静态 vs 动态
- 源代码的类型安全
- 编译期模板支持程度
- 可执行代码的大小和速度
- 堆内存
- 多线程
- 调试
- 状态和转移之间的耦合程度
- 维护

```
void test()
{
    player p;
    p.start();
    p.process_event(open_close());
    p.process_event(open_close());
    p.process_event(play());

    p.process_event(NextSong());pstate(p); //2nd song active
    p.process_event(PreviousSong());pstate(p);//2nd song active
    … …
}
```

有限状态机

- CRTP

```
// front-end: define the FSM structure
    struct player_ : public msm::front::state_machine_def<player_>
    {
        // The list of FSM states
        struct Empty : public msm::front::state<>
        {
            // every (optional) entry/exit methods get the event passed.
            template <class Event,class FSM>
            void on_entry(Event const&,FSM& ) {std::cout << "entering: Empty" << std::endl;}
            template <class Event,class FSM>
            void on_exit(Event const&,FSM& ) {std::cout << "leaving: Empty" << std::endl;}
        };


         struct Open : public msm::front::state<>
        {
            template <class Event,class FSM>
            void on_entry(Event const&,FSM& ) {std::cout << "entering: Open" << std::endl;}
            template <class Event,class FSM>
            void on_exit(Event const&,FSM& ) {std::cout << "leaving: Open" << std::endl;}
        };
```

有限状态机

- CRTP 奇异递归模板模式

```
template<class Derived>
class state_machine
{
    //...
protected:
    template< int CurrentState, class Event,
              int NextState, void (Derived::*action)(Event const&) >
    struct row
    {
    // for later use by our metaprogram
        static int const current_state = CurrentState;
        static int const next_state = NextState;
        typedef Event event;
        typedef Derived fsm_t;
        // do the transition action
        static void execute(Derived& fsm, Event const& e)
        {
            (fsm.*action) (e);
        }
     };
    // ...;
 };
```

```
// front-end: define the FSM structure
    struct player_ : public msm::front::state_machine_def<player_>
    {
        // The list of FSM states
        struct Empty : public msm::front::state<>
        {
            // every (optional) entry/exit methods get the event passed.
            template <class Event,class FSM>
            void on_entry(Event const&,FSM& ) {std::cout << "entering: Empty" << std::endl;}
            template <class Event,class FSM>
            void on_exit(Event const&,FSM& ) {std::cout << "leaving: Empty" << std::endl;}
        };


         struct Open : public msm::front::state<>
        {
            template <class Event,class FSM>
            void on_entry(Event const&,FSM& ) {std::cout << "entering: Open" << std::endl;}
            template <class Event,class FSM>
            void on_exit(Event const&,FSM& ) {std::cout << "leaving: Open" << std::endl;}
        };
```

有限状态机

编译期 if-else

```
void process_event(play const& e)
{
  switch (this->state)
  {
   case Stopped:
      this->start_playback(e);
      this->state = Playing;
      break;
   case Paused:
      this->resume_playback(e);
      this->state = Playing;
      break;
   default:
      this->state = no_transition(this->state, e);
  }
}

// "stop" event processor
void process_event(stop const& e)
{
    ...
}
```

```
// "play" event processor
void process_event(play const& e)
{
    this->state = case_Stopped(e);
}

int case_Stopped(play const& e)
{
    if (this->state == Stopped)
    {
       this->start_playback(e);
       return Playing;
    }
    else return this->case_Paused(e);
}

int case_Paused(play const& e)
{    //…;
}
```

有限状态机

编译期 if-else

```
template<class Transition,  // row
         class Next >       //
struct event_dispatcher
{
    typedef typename Transition::fsm_t fsm_t;
    typedef typename Transition::event event;

    static int dispatch(
    fsm_t& fsm, int state, event const& e)
    {
        if (state == Transition::current_state)
        {
            Transition::execute(fsm, e);
            return Transition::next_state;
        }

        else // move on to the next node in the chain.
        {
            return Next::dispatch(fsm, state, e);
        }
    };
 //...;
}
```

```
event_dispatcher<row<Stopped, play, Playing, &player::start_playback>,
event_dispatcher<row<Paused, play, Playing, &player::resume_playback>,
//...
default_event_dispatcher>
>
```

有限状态机

编译期 if-else 生成器，生成模板的模板

```
template<class Table, class Event>
struct generate_dispatcher
  : mpl::fold<
        mpl::filter_view<   // select rows triggered by Event
            Table,
          , boost::is_same<Event, transition_event<_1< >
        >
      , default_event_dispatcher
      , event_dispatcher<_2,_1>
    >
{};
```

```
template<class Event>
int process_event(Event const& evt)
{
    // 元函数，生成需要的派发器
    typedef typename generate_dispatcher<typename Derived::transition_table,
                                         Event>::type dispatcher;

    // 派发事件
    this->state = dispatcher::dispatch(
        *static_cast<Derived*>(this),       // CRTP
        this->state,
        evt
    );

    // 返回新状态
    return this->state;
}
```

有限状态机

- 表现力
  - （图灵）完备
  - 层级和并发状态
  - UML 状态图 + Petri 网
- 静态 vs 动态
- 源代码的类型安全
- 编译期模板支持程度
- 可执行代码的大小和速度
- 堆内存
- 多线程
- 调试
- 状态和转移之间的耦合程度
- 维护

```
void test()
{
    player p;
    p.start();
    p.process_event(open_close());
    p.process_event(open_close());
    p.process_event(play());

    p.process_event(NextSong());pstate(p); //2nd song active
    p.process_event(PreviousSong());pstate(p);//2nd song active
    … …
}
```

总结

标准、库、未来

嵌入式 C++ 与 TMP

- 14, 20(concept), 23(contract)
- Freestanding
- Boost
- 社区
  - Kvasir etc.
- 芯片
- 自己

```
/************************************************************************
 * This file contains the Kvasir Register Abstraction DSL (Domain Specific Language)
 * which provide an extra layer between Hardware SFRs
 * (Special Function Registers) and code accessing them.
 * Copyright 2015 Odin Holmes
 * Aditional contribution from Stephan Bökelmann
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
************************************************************************/

// ...;

template <typename TAddress, unsigned Mask, typename Access, typename FieldType>
struct RegisterExec<
Register::Action<FieldLocation<TAddress, Mask, Access, FieldType>, ReadAction>>
{
    DEBUG_OPTIMIZE unsigned operator()(unsigned in = 0)
    {
        return GetAddress<TAddress>::read();
    }
};

// ...;
namespace GpioPortB0{    ///<Byte pin registers port 0/1; pins PIO0/1_0 to PIO0/1_31
    using Addr = Register::Address<0x50000000,0xfffffffe,0x00000000,unsigned char>;
    ///Read: state of the pin P0/1_n, regardless of direction, masking, or alternate
    // function, except that pins configured as analog I/O always read as 0. Write: loads the pin's output
bit.
    constexpr Register::FieldLocation<Addr,Register::maskFromRange(0,0),Register::ReadWriteAccess,unsigned>
pbyte{};
}

// ...;
{
    Kvasir::Register::reads_.push({0x4003b010, 0x55});
    auto i = apply(read(Kvasir::Adc0Ra::d));
    if (i != 0x55)
    {
        return 1;
    }
    Kvasir::Register::reads_.push({0x4003b010, 0x555555});
    i = apply(read(Kvasir::Adc0Ra::d));
    if (i != 0x5555)
    {
        return 1;
    }
}
```

问答

CPP-Summit 2022

Kris Jusiak



Quantlab Financial资深架构师

Kris是一位热衷于编程的软件架构师，多年来在电信、游戏金融等不同的行业工作。他专注于现代C++开发，注重性能和质量；同时是一个开源爱好者，拥有多个开源库，使用模板元编程技术来支持C++规则--"不要为你不使用的东西付费"，同时在特定领域语言的帮助下，尽量做到声明性。Kris也是极端编程技术、测试/行为驱动开发的热心倡导者，并真正相信 "走得快的唯一方法就是走得好！"。

主办方：Boolan 高端IT咨询与教育平台

CppSummit / September 28-29, Shanghai

Embracing C++20 Meta-Programming

KRIS@JUSIAK.NET | @KRISJUSIAK | LINKEDIN.COM/IN/KRIS-JUSIAK

Agenda

1727

Agenda

- Motivation/Goal

1728

Agenda

- Motivation/Goal
- C++20 Meta-Programming

1729

Agenda

- Motivation/Goal
- C++20 Meta-Programming
  - Design By Introspection

1730

Agenda

- Motivation/Goal
- C++20 Meta-Programming
  - Design By Introspection
  - Immediately Invoked Function Expression

1731

Agenda

- Motivation/Goal
- C++20 Meta-Programming
  - Design By Introspection
  - Immediately Invoked Function Expression
  - std::fixed_string

Agenda

- Motivation/Goal
- C++20 Meta-Programming
  - Design By Introspection
  - Immediately Invoked Function Expression
  - std::fixed_string
  - constexpr std::vector

Agenda

1734

Agenda

- Showcase

1735

Agenda

- Showcase
  - State Machine Language

1736

Agenda

- Showcase
  - State Machine Language
- Summary

Agenda

- Showcase
  - State Machine Language
- Summary

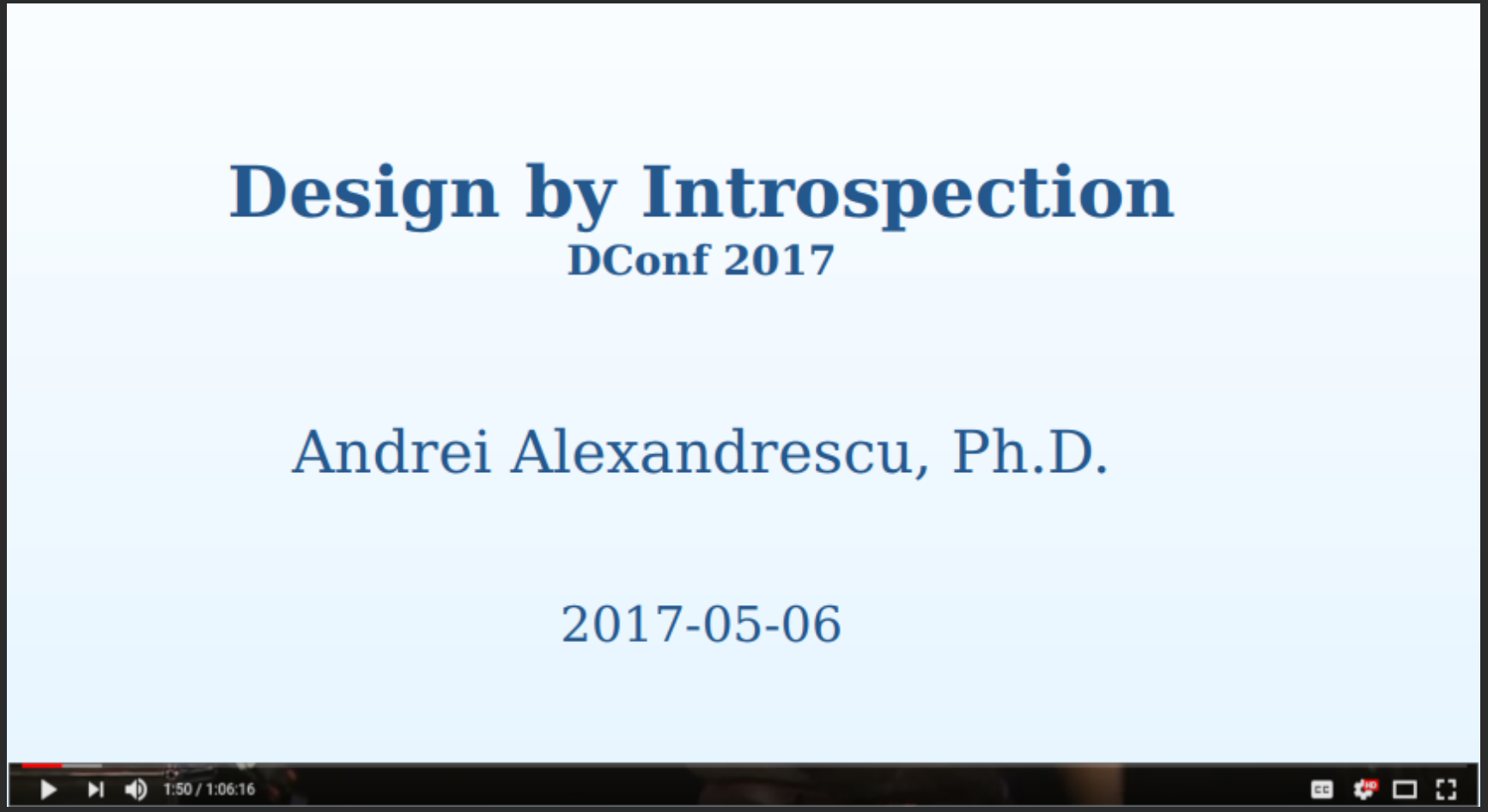
POWERED BY COMPILER EXPLORER

1738

C++20 - Design By Introspection

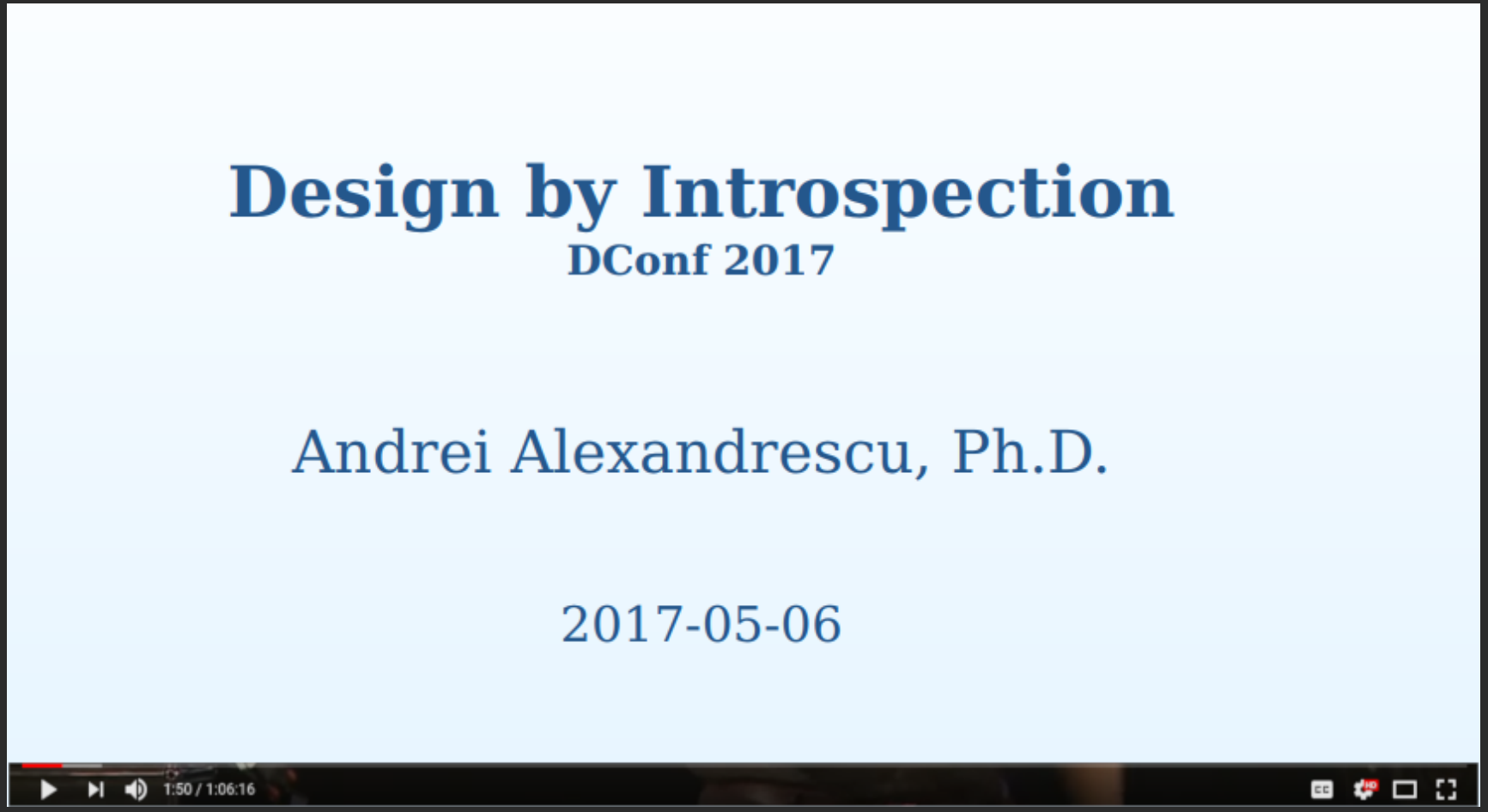
1739

C++20 - Design By Introspection



1740

C++20 - Design By Introspection



https://www.youtube.com/watch?v=29h6jGtZD-U

C++20 - Design By Introspection

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  return t.value; // error: bar doesn't have value
}
```


1743

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  return t.value; // error: bar doesn't have value
}
```

```
consteval auto get(auto t) {
  if (t.value) { // error: bar doesn't have value
    return t.value;
  } else {
    return 0;
  }
}
```

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  return t.value; // error: bar doesn't have value
}
```

```
consteval auto get(auto t) {
  if (t.value) { // error: bar doesn't have value
    return t.value;
  } else {
    return 0;
  }
}
```

```
...
```

C++20 - Design By Introspection

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  if constexpr (requires{ t.value; }) { // ok 👍
    return t.value;
  } else {
    return 0; // base case
  }
}
```

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  if constexpr (requires{ t.value; }) { // ok 👍
    return t.value;
  } else {
    return 0; // base case
  }
}
```

```
struct {
  int value{42};
} f;

static_assert(42 == foo(f));
```

C++20 - Design By Introspection

```
consteval auto get(auto t) {
  if constexpr (requires{ t.value; }) { // ok 👍
    return t.value;
  } else {
    return 0; // base case
  }
}
```

```
struct {
  int value{42};
} f;

static_assert(42 == foo(f));
```

```
struct { } b;

static_assert(0 == foo(b));
```

C++20 - Immediately-Invoked Function Expression (IIFE)

< C++20

>= C++20

1750

C++20 - Immediately-Invoked Function Expression (IIFE)

< C++20

```
template<auto N>
constexpr auto foo = [] {
  // expand over N... // before C++20 requires
                      /// an external function call
};
```

>= C++20

1751

C++20 - Immediately-Invoked Function Expression (IIFE)

< C++20

```
template<auto N>
constexpr auto foo = [] {
  // expand over N... // before C++20 requires
                      /// an external function call
};
```

>= C++20

```
template<auto N>
constexpr auto unroll = [](auto expr) {
  [expr]<auto ...Is>(std::index_sequence<Is...>) {
    ((expr(), void(Is)), ...);
  }(std::make_index_sequence<N>{});
};
```

C++2X - P1061 - Structured Bindings can introduce a Pack

C++2X - P1061 - Structured Bindings can introduce a Pack

```
template<auto N>
constexpr auto unroll = [](auto expr) {
  auto [... Is] = std::make_index_sequence<N>{};
  ((expr(), void(Is)), ...);
};
```

C++20 - Immediately-Invoked Function Expression (`IIFE`)

1755

C++20 - Immediately-Invoked Function Expression (IIFE)

```
int main() {
  unroll<2>([]{ std::puts("CppSummit 2022!"); });
}
```

1756

C++20 - Immediately-Invoked Function Expression (IIFE)

```cpp
int main() {
  unroll<2>([]{ std::puts("CppSummit 2022!"); });
}
```

```asm
.LC0: .string "CppSummit 2022!"
main:
  sub     rsp, 8
  mov     edi, OFFSET FLAT:.LC0
  call    puts
  mov     edi, OFFSET FLAT:.LC0
  call    puts
  xor     eax, eax
  add     rsp, 8
  ret
```

C++20 - Fixed String

C++20 - Fixed String

```
template<class T> auto foo();
```


1759

C++20 - Fixed String

```
template<class T> auto foo();
```

```
foo<"bar">; // error: invalid explicitly-specified
            // argument for template parameter T
```


1760

GNU++17 - Wgnu-string-literal-operator-template

GNU++17 - Wgnu-string-literal-operator-template

```
template<char... Cs>
struct ct_string {
  static constexpr char str[] = {Cs..., 0};
  static constexpr auto name() -> std::string_view {
      return std::string_view{str, sizeof...(Cs)};
  }
};
```

GNU++17 - Wgnu-string-literal-operator-template

```
template<char... Cs>
struct ct_string {
  static constexpr char str[] = {Cs..., 0};
  static constexpr auto name() -> std::string_view {
      return std::string_view{str, sizeof...(Cs)};
  }
};
```

```
template <class T, T... Cs>
[[nodiscard]] constexpr auto operator""_cs() {
    return ct_string<Cs...>{};
}
```

GNU++17 - Wgnu-string-literal-operator-template

GNU++17 - Wgnu-string-literal-operator-template

```
constexpr auto foo = "foo"_cs;
static_assert(typeid(ct_string<'f', 'o', 'o'>) == foo);
```

GNU++17 - Wgnu-string-literal-operator-template

```
constexpr auto foo = "foo"_cs;
static_assert(typeid(ct_string<'f', 'o', 'o'>) == foo);
```

```
template<class T> auto foo;
```

GNU++17 - Wgnu-string-literal-operator-template

```
constexpr auto foo = "foo"_cs;
static_assert(typeid(ct_string<'f', 'o', 'o'>) == foo);
```

```
template<class T> auto foo;
```

```
foo<"foo"_cs>              // error: invalid explicitly-specified
                           // argument for template parameter T
```

GNU++17 - Wgnu-string-literal-operator-template

```
constexpr auto foo = "foo"_cs;
static_assert(typeid(ct_string<'f', 'o', 'o'>) == foo);
```

```
template<class T> auto foo;
```

```
foo<"foo"_cs>                // error: invalid explicitly-specified
                             // argument for template parameter T
```

```
foo<decltype("foo"_cs)>; // okay
```

C++20 - fixed_string

1769

C++20 - fixed_string

```
template <std::size_t N> struct fixed_string {
```

```
};
```


1770

C++20 - fixed_string

```
template <std::size_t N> struct fixed_string {
  std::array<char, N + 1> data{};
  static constexpr auto size = N;
};
```

1771

C++20 - fixed_string

```
template <std::size_t N> struct fixed_string {
  std::array<char, N + 1> data{};
  static constexpr auto size = N;

  constexpr explicit(true) fixed_string(const auto... cs)
    : data{cs...} { }

  constexpr explicit(false) fixed_string(const char (&str)[N + 1]) {
    std::copy_n(str, N + 1, std::data(data));
  }
};
```

1772

C++20 - fixed_string

```
template <std::size_t N> struct fixed_string {
```

```
  std::array<char, N + 1> data{};
  static constexpr auto size = N;
```

```
  constexpr explicit(true) fixed_string(const auto... cs)
    : data{cs...} { }

  constexpr explicit(false) fixed_string(const char (&str)[N + 1]) {
    std::copy_n(str, N + 1, std::data(data));
  }
```

```
  [[nodiscard]]
  constexpr auto operator<=>(const fixed_string&) const = default;

  [[nodiscard]]
  constexpr explicit(false) operator std::string_view() const {
    return {std::data(data), N};
  }
```

```
};
```

1773

C++20 - fixed_string

C++20 - fixed_string

```
template<fixed_string> auto foo();
foo<"bar">(); // okay
```

1775

C++20 - fixed_string

```
template<fixed_string> auto foo();
foo<"bar">(); // okay
```

```
template <stdext::fixed_string Str>
static auto to_string = []<auto... Ns>(std::index_sequence<Ns...>) {
  return string<Str.data[Ns]...>{};
}(std::make_index_sequence<Str.size()>{});
```

1776

C++20 - fixed_string

```
template<fixed_string> auto foo();
foo<"bar">(); // okay
```

```
template <stdext::fixed_string Str>
static auto to_string = []<auto... Ns>(std::index_sequence<Ns...>) {
  return string<Str.data[Ns]...>{};
}(std::make_index_sequence<Str.size()>{});
```

```
static_assert(string<'f', 'o', 'o'> == to_string<"foo">)
```

1777

C++20 - constexpr std::vector

1778

C++20 - constexpr std::vector

```
constexpr std::vector v{4, 2, 1, 3};
std::ranges::sort(v);
static_assert(std::vector{1, 2, 3, 4} == v);
```


1779

C++20 - constexpr std::vector

```
constexpr std::vector v{4, 2, 1, 3};
std::ranges::sort(v);
static_assert(std::vector{1, 2, 3, 4} == v);
```

Hmm, almost, but...

1780

C++20 - https://github.com/boost-ext/mp

1781

C++20 - https://github.com/boost-ext/mp

```
#include <ranges>

auto hello_world = [](auto list, auto add_const, auto has_value){
 return list
   | std::views::drop(1_c)
   | std::views::reverse
   | std::views::take(2_c)
   | std::views::transform(add_const)
   | std::views::filter(has_value)
   ;
};
```

1782

C++20 - https://github.com/boost-ext/mp

1783

C++20 - https://github.com/boost-ext/mp

```
auto add_const = []<class T> -> T const {};
auto has_value = []<class T> { return requires(T t) { t.value; }; };
```

1784

C++20 - https://github.com/boost-ext/mp

```
auto add_const = []<class T> -> T const {};
auto has_value = []<class T> { return requires(T t) { t.value; }; };
```

```
struct bar {}; struct foo { int value; }; struct val { int value; };
```


1785

C++20 - https://github.com/boost-ext/mp

1786

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                              // int, foo, val, bar
```

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                                  // int, foo, val, bar
```

```
   | std::views::drop(1_c)                    // foo, val, bar
```

1789

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                              // int, foo, val, bar
```

```
   | std::views::drop(1_c)                // foo, val, bar
```

```
   | std::views::reverse                  // bar, val, foo
```

1790

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                              // int, foo, val, bar
```

```
   | std::views::drop(1_c)                // foo, val, bar
```

```
   | std::views::reverse                  // bar, val, foo
```

```
   | std::views::take(2_c)                // bar, val
```

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                              // int, foo, val, bar
```

```
   | std::views::drop(1_c)                // foo, val, bar
```

```
   | std::views::reverse                  // bar, val, foo
```

```
   | std::views::take(2_c)                // bar, val
```

```
   | std::views::transform(add_const)     // bar const, val const
```

C++20 - https://github.com/boost-ext/mp

```
static_assert(mp::list<val const> ==
  hello_world(mp::list<int, foo, val, bar>, add_const, has_value)
);
```

```
 return list                              // int, foo, val, bar

  | std::views::drop(1_c)                 // foo, val, bar

  | std::views::reverse                   // bar, val, foo

  | std::views::take(2_c)                 // bar, val

  | std::views::transform(add_const)      // bar const, val const

  | std::views::filter(has_value)         // val const
  ;
```

1793

C++20 - https://github.com/boost-ext/mp

1794

C++20 - https://github.com/boost-ext/mp

```cpp
#include <ranges>

auto slice = [](auto list, auto begin, auto end) {
  return list
    | std::views::drop(begin)
    | std::views::take(end - 1_c);
};
```


1795

C++20 - https://github.com/boost-ext/mp

1796

C++20 - https://github.com/boost-ext/mp

```
// type_list
static_assert(slice(mp::list<int, double, float, short>, 1_c, 3_c) ==
                    mp::list<double, float>);
```

C++20 - https://github.com/boost-ext/mp

```
// type_list
static_assert(slice(mp::list<int, double, float, short>, 1_c, 3_c) ==
                    mp::list<double, float>);
```

```
// variant.type
static_assert(std::is_same_v<
  mp::typeof<slice, std::variant<int, double, float, short>, 1_c, 3_c>,
                    std::variant<double, float>>
);
```

C++20 - https://github.com/boost-ext/mp

1799

C++20 - https://github.com/boost-ext/mp

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c) ==
                    mp::list<2, 3>);
```

C++20 - https://github.com/boost-ext/mp

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c) ==
                    mp::list<2, 3>);
```

```
// fixed_string
static_assert(slice(mp::list<"foobar">, 1_c, 3_c) ==
                    mp::list<"oo">);
```

C++20 - https://github.com/boost-ext/mp

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c) ==
                    mp::list<2, 3>);
```

```
// fixed_string
static_assert(slice(mp::list<"foobar">, 1_c, 3_c) ==
                    mp::list<"oo">);
```

```
// tuple of values
static_assert(slice(std::tuple{1, 2, 3, 4}, 1_c, 3_c) ==
                    std::tuple{2, 3});
```

1802

C++20 - https://github.com/boost-ext/mp

```
// value_list
static_assert(slice(mp::list<1, 2, 3, 4>, 1_c, 3_c) ==
                    mp::list<2, 3>);
```

```
// fixed_string
static_assert(slice(mp::list<"foobar">, 1_c, 3_c) ==
                    mp::list<"oo">);
```

```
// tuple of values
static_assert(slice(std::tuple{1, 2, 3, 4}, 1_c, 3_c) ==
                    std::tuple{2, 3});
```

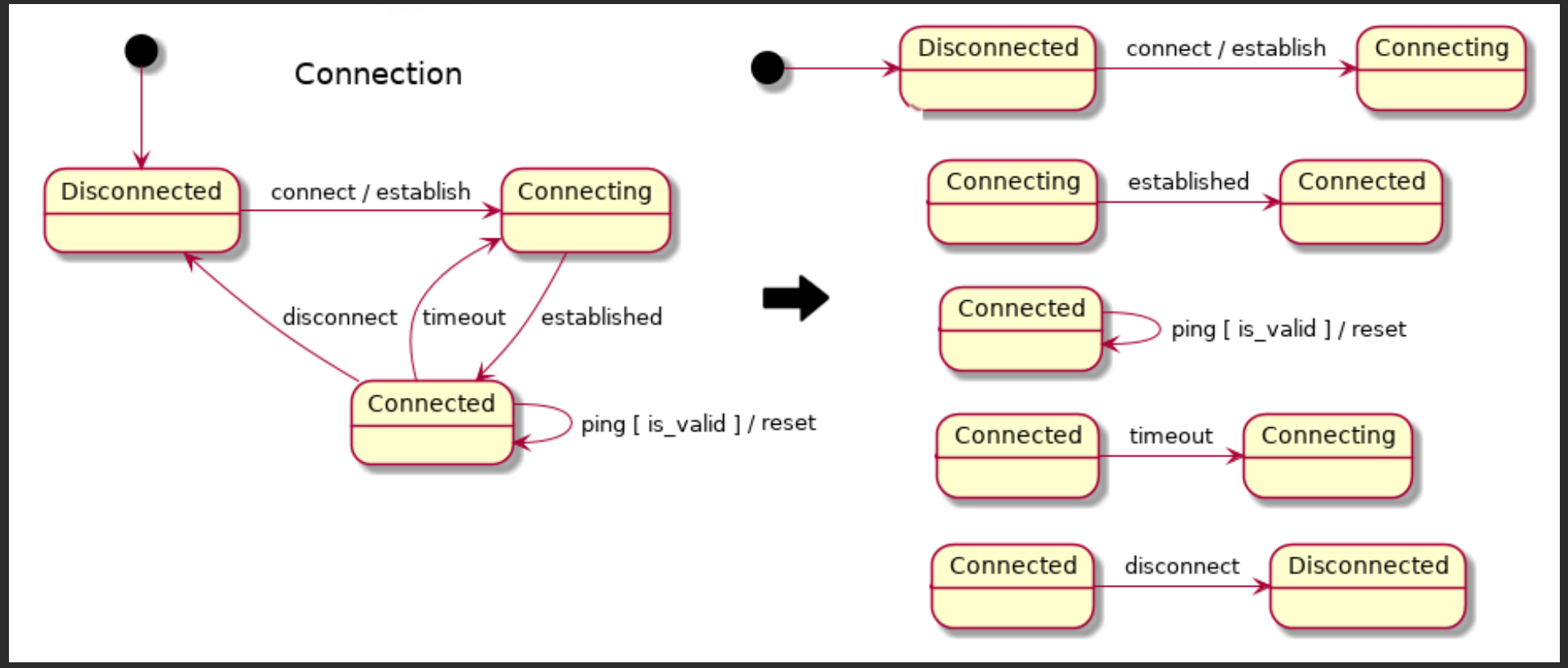
```
#include <cassert>

int main(int argc, const char**) {
  // run-time tuple of values
  assert((slice(std::tuple{1, argc, 3, 4}, 1_c, 3_c) ==
                std::tuple{argc, 3}));
}
```

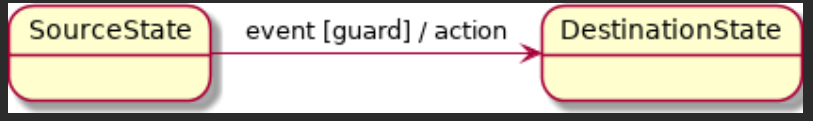
1803

C++20 - Showcase

State Machine Language (UML 2.5)



Transition



SML - Domain Specific Language (DSL)

SML - Domain Specific Language (DSL)

```
* -> Disconnected : connect / establish                -> Connecting
```

SML - Domain Specific Language (DSL)

```
* -> Disconnected : connect / establish              -> Connecting

     Connecting   : established                      -> Connected
```

SML - Domain Specific Language (DSL)

```
* -> Disconnected : connect / establish                -> Connecting

    Connecting   : established                        -> Connected

    Connected    : ping [ is_valid ] / reset
```

SML - Domain Specific Language (DSL)

```
* -> Disconnected : connect / establish                -> Connecting

    Connecting   : established                         -> Connected

    Connected    : ping [ is_valid ] / reset

    Connected    : timeout                             -> Connecting
```

SML - Domain Specific Language (DSL)

```
* -> Disconnected : connect / establish              -> Connecting

    Connecting   : established                       -> Connected

    Connected    : ping [ is_valid ] / reset

    Connected    : timeout                           -> Connecting

    Connected    : disconnect                        -> Disconnected
```

SML - Domain Specific Language (DSL)

1812

SML - Domain Specific Language (DSL)

```
// events
struct connect {};
struct established{};
struct ping { bool valid{}; };
struct timeout {};
struct disconnect {};
```

SML - Domain Specific Language (DSL)

```
// events
struct connect {};
struct established{};
struct ping { bool valid{}; };
struct timeout {};
struct disconnect {};
```

```
// guards/actions
auto establish = []{ std::puts("establish"); };
auto close     = []{ std::puts("close"); };
auto is_valid  = [](const auto& event) { return event.valid; };
auto setup     = []{ std::puts("setup"); };
```

1814

SML - Domain Specific Language (DSL)

1815

SML - Domain Specific Language (DSL)

```
sm connection = [] {
  /**
   * src_state + event [ guard ] / action = dst_state
   */
  return transition_table{
   * "Disconnected"_s + event<connect> / establish  = "Connecting"_s,
     "Connecting"_s   + event<established>          = "Connected"_s,
     "Connected"_s    + event<ping> [ is_valid ] / reset,
     "Connected"_s    + event<timeout> / establish  = "Connecting"_s,
     "Connected"_s    + event<disconnect> / close   = "Disconnected"_s,
  };
};
```

SML - Domain Specific Language (DSL)

1817

SML - Domain Specific Language (DSL)

```
int main() {
  sm connection{Connection{}};
  assert(connection.is("Disconnected"_s));

  connection.process_event(connect{});
  assert(connection.is("Connecting"_s));

  connection.process_event(established{});
  assert(connection.is("Connected"_s));

  connection.process_event(ping{true});
  assert(connection.is("Connected"_s));

  connection.process_event(disconnect{});
  assert(connection.is("Disconnected"_s));
}
```

C++20 - Performance

x86-64 gcc 12.2 | -std=c++20 -O3

Output... | Filter... | Libraries | + Add new... | Add tool...

```
1  .LC0:
2          .string "establish"
3  .LC1:
4          .string "setup"
5  .LC2:
6          .string "close"
7  main:
8          sub     rsp, 8
9          mov     edi, OFFSET FLAT:.LC0
10         call    puts
11         mov     edi, OFFSET FLAT:.LC1
12         call    puts
13         mov     edi, OFFSET FLAT:.LC2
14         call    puts
15         xor     eax, eax
16         add     rsp, 8
17         ret
```

x86-64 clang (trunk) | -std=c++20 -stdlib=libc++ -O3

Output... | Filter... | Libraries | + Add new... | Add tool...

```
1  main:                                   # @main
2          pushq   %rax
3          leaq    .L.str.4(%rip), %rdi
4          callq   puts@PLT
5          leaq    .L.str.5(%rip), %rdi
6          callq   puts@PLT
7          leaq    .L.str.6(%rip), %rdi
8          callq   puts@PLT
9          xorl    %eax, %eax
10         popq    %rcx
11         retq
12 .L.str.4:
13         .asciz  "establish"
14
15 .L.str.5:
16         .asciz  "setup"
17
```

1819

C++20 - Performance

x86-64 gcc 12.2 | -std=c++20 -O3

Output... | Filter... | Libraries | + Add new... | Add tool...

```
1   .LC0:
2           .string "establish"
3   .LC1:
4           .string "setup"
5   .LC2:
6           .string "close"
7   main:
8           sub     rsp, 8
9           mov     edi, OFFSET FLAT:.LC0
10          call    puts
11          mov     edi, OFFSET FLAT:.LC1
12          call    puts
13          mov     edi, OFFSET FLAT:.LC2
14          call    puts
15          xor     eax, eax
16          add     rsp, 8
17          ret
```

x86-64 clang (trunk) | -std=c++20 -stdlib=libc++ -O3

Output... | Filter... | Libraries | + Add new... | Add tool...

```
1   main:                                   # @main
2           pushq   %rax
3           leaq    .L.str.4(%rip), %rdi
4           callq   puts@PLT
5           leaq    .L.str.5(%rip), %rdi
6           callq   puts@PLT
7           leaq    .L.str.6(%rip), %rdi
8           callq   puts@PLT
9           xorl    %eax, %eax
10          popq    %rcx
11          retq
12  .L.str.4:
13          .asciz  "establish"
14
15  .L.str.5:
16          .asciz  "setup"
17
```

Zero overhead abstraction (or maybe even negative overhead?)

1820

C++20 - SML





1821

C++20 - SML

- Front-End : DSL -> list of transitions
  -
  -

1822

C++20 - SML

- Front-End : DSL -> list of transitions
  - *src + event [guard] / action = dst*
  -

1823

C++20 - SML

- Front-End : DSL -> list of transitions
  - *src + event [guard] / action = dst*
- Back-End : executes list of transitions with events
  -

1824

C++20 - SML

- Front-End : DSL -> list of transitions
  - *src + event [guard] / action = dst*
- Back-End : executes list of transitions with events
  - *process_event*

C++20 - SML.front-end / transition (fixed_string)

1826

C++20 - SML.front-end / transition (fixed_string)

```
template<mp::fixed_string Src = "",
         class TEvent  = none,
         class TGuard  = always,
         class TAction = none,
         mp::fixed_string Dst = "">
struct transition {
```

```
  // ...
};
```

C++20 - SML.front-end / transition (fixed_string)

```
template<mp::fixed_string Src = "",
         class TEvent  = none,
         class TGuard  = always,
         class TAction = none,
         mp::fixed_string Dst = "">
struct transition {
```

```
  static constexpr auto src = Src;
  static constexpr auto dst = Dst;
  using event = TEvent;
```

```
  // ...
};
```

C++20 - SML.front-end / transition (fixed_string)

```
template<mp::fixed_string Src = "",
         class TEvent  = none,
         class TGuard  = always,
         class TAction = none,
         mp::fixed_string Dst = "">
struct transition {
```

```
  static constexpr auto src = Src;
  static constexpr auto dst = Dst;
  using event = TEvent;
```

```
  constexpr auto operator*() const;
  constexpr auto operator+(auto state) const;
  constexpr auto operator[](auto guard) const;
  constexpr auto operator/(auto action) const;
  constexpr auto operator=(auto state) const;
```

```
  // ...
};
```

1829

C++20 - SML.utils (constexpr std::vector + std::ranges)

1830

C++20 - SML.utils (constexpr std::vector + std::ranges)

```
template<auto Fn> constexpr auto unique = [](auto types) {
```

```
  return types;
};
```

C++20 - SML.utils (constexpr std::vector + std::ranges)

```
template<auto Fn> constexpr auto unique = [](auto types) {
  const auto std::array fns = {Fn.template operator()<Ts>()...};
  return types;
};
```

C++20 - SML.utils (constexpr std::vector + std::ranges)

```
template<auto Fn> constexpr auto unique = [](auto types) {

  const auto std::array fns = {Fn.template operator()<Ts>()...};

  std::ranges::sort(types,
    [&](auto lhs, auto rhs) { return fns[lhs] < fns[rhs]; });

  return types;
};
```

C++20 - SML.utils (constexpr std::vector + std::ranges)

```
template<auto Fn> constexpr auto unique = [](auto types) {

  const auto std::array fns = {Fn.template operator()<Ts>()...};

  std::ranges::sort(types,
    [&](auto lhs, auto rhs) { return fns[lhs] < fns[rhs]; });

  auto [first, last] = std::ranges::unique(types,
    [&]( auto lhs, auto rhs) { return fns[lhs] == fns[rhs]; });


  return types;
};
```

C++20 - SML.utils (constexpr std::vector + std::ranges)

```
template<auto Fn> constexpr auto unique = [](auto types) {

  const auto std::array fns = {Fn.template operator()<Ts>()...};

  std::ranges::sort(types,
    [&](auto lhs, auto rhs) { return fns[lhs] < fns[rhs]; });

  auto [first, last] = std::ranges::unique(types,
    [&]( auto lhs, auto rhs) { return fns[lhs] == fns[rhs]; });

  types.erase(first, last);

  return types;
};
```

C++20 - SML.back-end / sm (constexpr std::vector)

1836

C++20 - SML.back-end / sm (constexpr std::vector)

```
template<auto transitions> class sm {
  // ...

 public:
  constexpr explicit(true) sm(auto& transitions);
  constexpr auto process_event(const auto& event) -> void;

 private:
  std::size_t> current_state_{};
};
```

1837

C++20 - SML.back-end / events (constexpr std::vector + std::ranges)

1838

C++20 - SML.back-end / events (constexpr std::vector + std::ranges)

```
constexpr auto events = transitions
  | std::views::transform([]<class T>() -> typename T::event {})
  | unique<by_type>
  ;
```

C++20 - SML.back-end / events (constexpr std::vector + std::ranges)

```
constexpr auto events = transitions
  | std::views::transform([]<class T>() -> typename T::event {})
  | unique<by_type>
  ;
```

```
Example: connect, established, ping, timeout, disconnect
```

C++20 - SML.back-end / states (constexpr std::vector + std::ranges)

```
constexpr auto states = transitions
  | []<class... Ts>() { return mp::list<Ts::src..., Ts::dst...>; }
  | std::views::filter([]<mp::fixed_string State> {
      return not std::empty(std::string_view(State));
    })
  | unique<by_name>
  ;
```

1841

C++20 - SML.back-end / states (constexpr std::vector + std::ranges)

```
constexpr auto states = transitions
  | []<class... Ts>() { return mp::list<Ts::src..., Ts::dst...>; }
  | std::views::filter([]<mp::fixed_string State> {
      return not std::empty(std::string_view(State));
    })
  | unique<by_name>
  ;
```

```
Example: Disconnected, Connecting, Connected, Disconnected
```

1842

C++20 - SML.back-end / mappings (constexpr std::vector + std::ranges)



1844

C++20 - SML.back-end / mappings (constexpr std::vector + std::ranges)

```
static constexpr auto mappings = events
```

```
;
```


1846

C++20 - SML.back-end / mappings (constexpr std::vector + std::ranges)

```
static constexpr auto mappings = events
```

```
  | std::views::transform([]<class TEvent> {
      return std::pair{TEvent{}, states
```

```
      };
    })
```

```
;
```


1848

C++20 - SML.back-end / mappings (constexpr std::vector + std::ranges)

```
static constexpr auto mappings = events
```

```
  | std::views::transform([]<class TEvent> {
      return std::pair{TEvent{}, states
```

```
          | std::views::transform([]<mp::fixed_string State> {
              return transitions
                | std::views::filter([]<class T> {
                    return T::src == State and
                           typeid(typename T::event) == typeid(TEvent);
                  })
```

```
            })
```

```
      };
    })
```

```
;
```


1850

C++20 - SML.back-end / mappings (constexpr std::vector + std::ranges)

```
static constexpr auto mappings = events
```

```
  | std::views::transform([]<class TEvent> {
      return std::pair{TEvent{}, states
```

```
          | std::views::transform([]<mp::fixed_string State> {
              return transitions
                | std::views::filter([]<class T> {
                    return T::src == State and
                           typeid(typename T::event) == typeid(TEvent);
                  })
```

```
           | []<class... Ts> {
               [](const auto& event, auto& current_state) {
                 (... or Ts(event, current_event));
               }
             }
```

```
           })
```

```
      };
    })
```

```
;
```


1852

C++20 - SML.back-end / mappings

C++20 - SML.back-end / mappings

```
Example:
  std::pair<ping,
    list<>,                          // state.Connecting
    list<transition<"Connected",
                    ping,
                    is_valid,
                    reset
                    ""><             // state.Connected
    list<>,                          // state.Disconnected
  >, ...>;
```

C++20 - SML.back-end / process_event (IIFE)

C++20 - SML.back-end / process_event (IIFE)

```
constexpr auto process_event(const auto& event) -> void {
```

```
}
```


1856

C++20 - SML.back-end / process_event (IIFE)

```
constexpr auto process_event(const auto& event) -> void {
```

```
[this, &event]<class T>(const std::pair<TEvent, T>& transitions) {
  auto& [_, ts] = transitions;
```

```
// find transitions for corresponding event
(mappings);
```

```
}
```

C++20 - SML.back-end / process_event (IIFE)

```
constexpr auto process_event(const auto& event) -> void {
```

```
[this, &event]<class T>(const std::pair<TEvent, T>& transitions) {
  auto& [_, ts] = transitions;
```

```
  // find transitions for current state
  mp::for_each(auto index, const auto& transition) {
    if (index == current_state_) {
      transition(event, current_state_); // execute transition
    }
  }, ts);
```

```
// find transitions for corresponding event
(mappings);
```

```
}
```

C++20 - SML.front-end/back-end / transition (fixed_string)

1859

C++20 - SML.front-end/back-end / transition (fixed_string)

```
template<mp::fixed_string Src, class TEvent,
         class TGuard, class TAction, mp::fixed_string Dst>
struct transition {
  // ...

  [[no_unique_address]] TGuard guard;
  [[no_unique_address]] TAction action;
};
```

1860

C++20 - SML.front-end/back-end / transition (fixed_string)

```
template<mp::fixed_string Src, class TEvent,
         class TGuard, class TAction, mp::fixed_string Dst>
struct transition {
  // ...

  [[no_unique_address]] TGuard guard;
  [[no_unique_address]] TAction action;
```

```
  [[nodiscard]] constexpr auto
  operator()(const TEvent& event, auto& current_state) const {
    if (guard(event) {          // execute guard
        action(event);          // execute action
        current_state_ = dst; // update current state
        return true;            // stop processing next transitions
    } else {
      return false;
    }
  }
}
```

```
};
```

1861

SML in action...

---

https://godbolt.org/z/1TEoezq5c

Summary

1863

Summary

- C++20 enables new ways of Template Meta-Programming (TMP)

Summary

- C++20 enables new ways of Template Meta-Programming (TMP)
- constexpr std::vector + std::ranges can is a powerful TMP combination

1865

Summary

- C++20 enables new ways of Template Meta-Programming (TMP)
- constexpr std::vector + std::ranges can is a powerful TMP combination
- https://github.com/boost-ext/mp leverages new C++20 TMP techniques

1866

CppSummit / September 28-29, Shanghai

Let's embrace C++20 Meta-Programming!

KRIS@JUSIAK.NET | @KRISJUSIAK | LINKEDIN.COM/IN/KRIS-JUSIAK

1867

CPP-Summit 2022



毛文安

阿里云操作系统高级技术专家

多年Linux操作系统开发、性能调优经验，曾是华为内核实验室net子系统committer，负责内核版本的维护及网络特性开发，向Linux内核社区贡献50+补丁。现为阿里巴巴操作系统团队高级技术专家，主要负责领域为网络新特性的应用和eBPF tracing技术的探索，挖掘从容器应用到内核及驱动的全链路报文时延信息，为解决疑难的网络抖动问题打开一片天地。主导开发了丢包延迟可视化工具pingtrace，以及针对抖动问题的监控系统netinfo，BPF开发编译平台LCC等。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

毛文安

龙蜥社区eBPF技术探索SIG Maintainer
系统运维SIG Owner

eBPF开发和资源优化在多个内核版本的实践

议程

1 **eBPF概述**
介绍eBPF应用场景和社区发展概况

2 **eBPF的开发方式**
介绍eBPF开发流程和当前经典项目

3 **中高版本内核eBPF实践**
介绍Coolbpf实现简洁开发

4 **低版本内核eBPF实践**
介绍3.10低版本内核的eBPF移植

01

eBPF概述

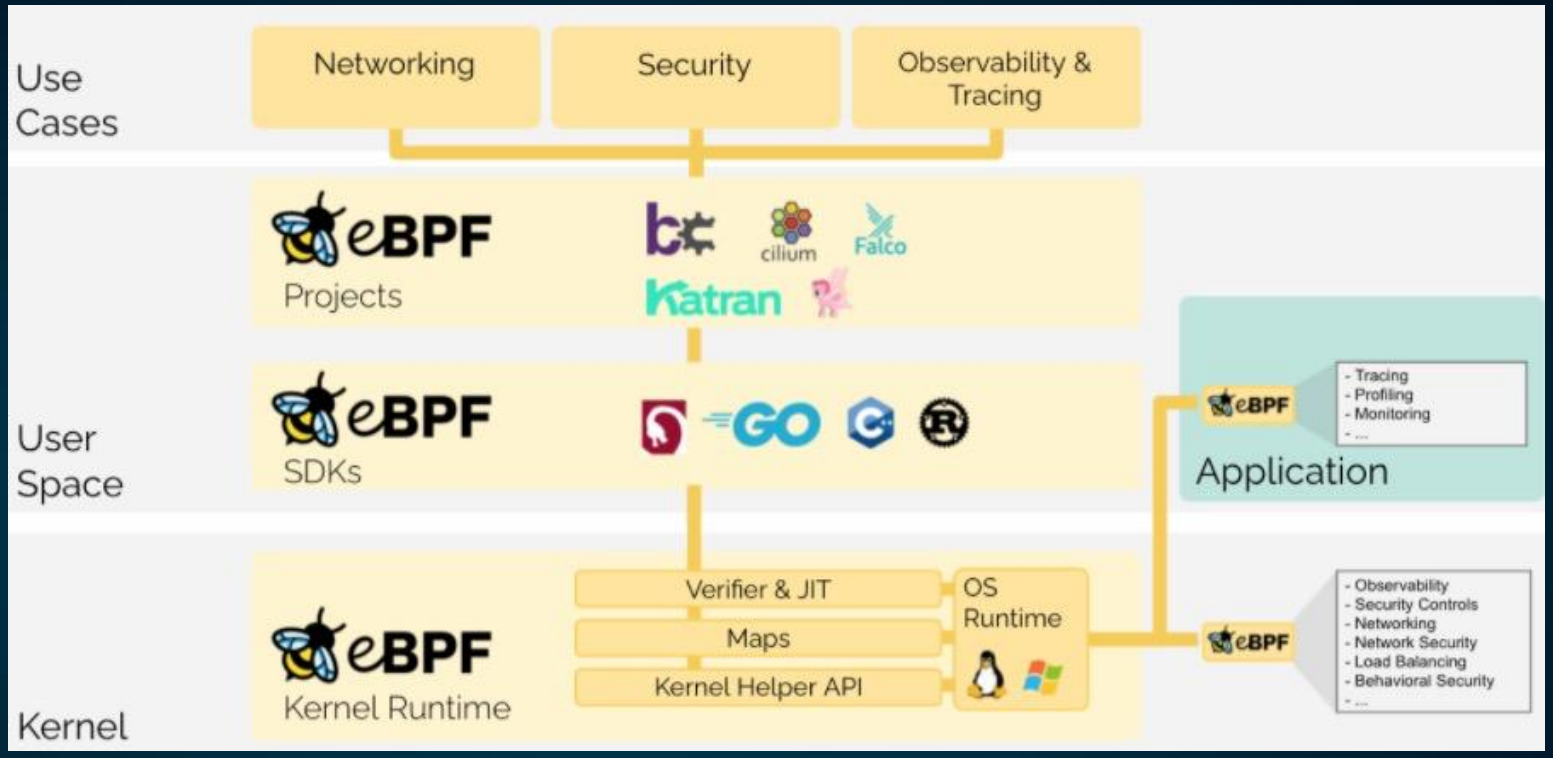
1992年提出Berkely Packet Filter（cBPF），比当时最先进的数据包过滤技术快20倍。主要应用在 tcpdump、seccomp；

2014年提出Extended Berkeley Packet Filter (eBPF)， eBPF指令更接近硬件的ISA，便于提升性能，主要应用在网络优化、系统跟踪等。

eBPF知识点

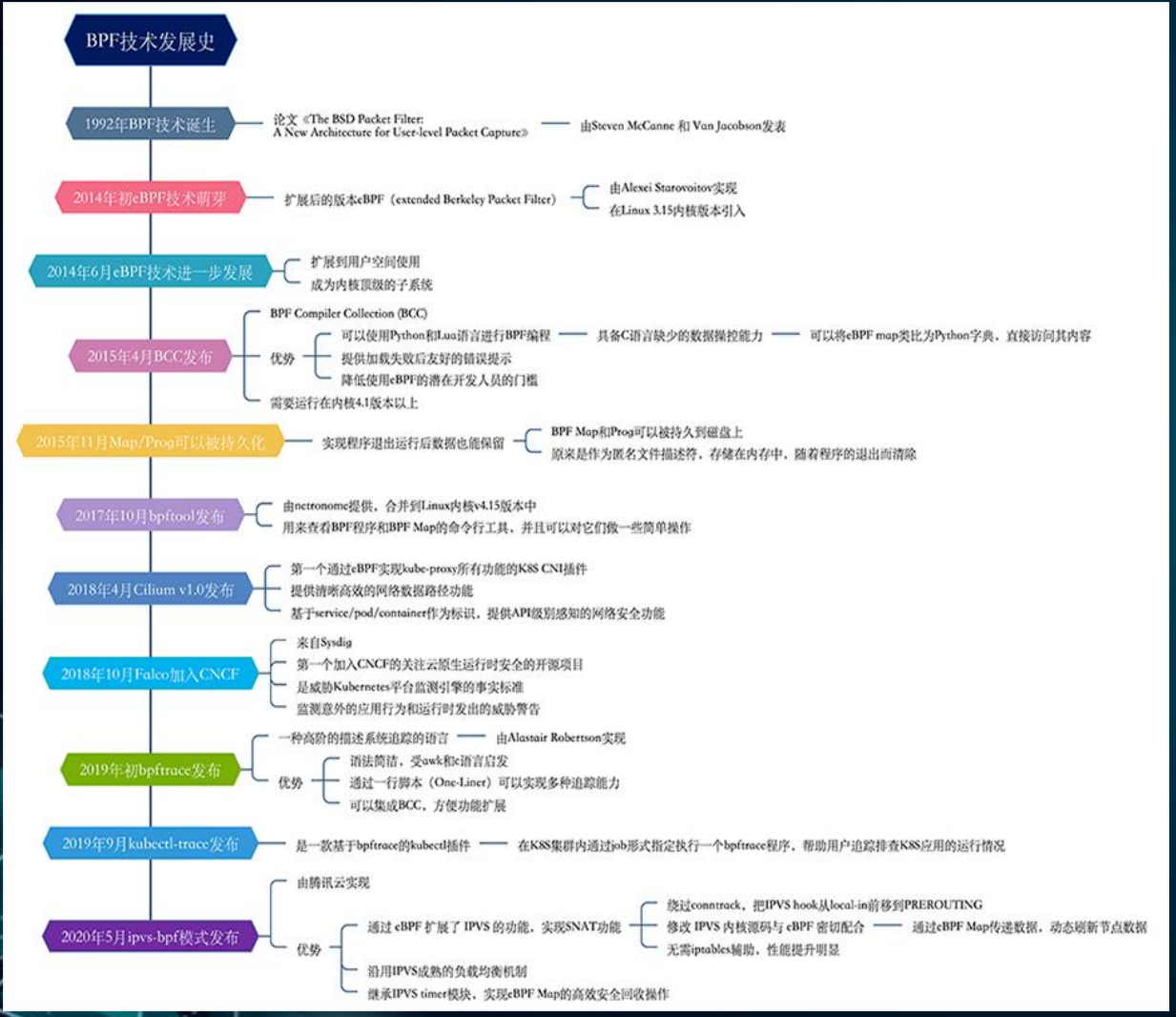
- risc指令集：11 个64位寄存器 (R0 ~ R10)，op:8, dst_reg:4, src_reg:4, off:16, imm:32；
- helpers函数：通过helpers函数访问内核数据，如maps、pid等；
- maps：BPF程序之间或内核及用户态间数据交互；
- object pinning：提供bpf文件系统，延长map和prog的生命周期；
- 尾调用：一个bpf程序可以调用另一个bpf程序，并且调用完成后不用返回到原来的程序；
- jit：将bpf程序的字节码转换成目标机的机器码；
- hardening：保护bpf程序和其二进制程序不被破坏（设置成只读）。

- 性能分析
- 故障诊断
- 网络监控
- 系统安全



BPF优势：它提供了一种在不修改内核代码的情况下，可以灵活修改内核处理策略的方法。

eBPF行业动向



eBPF 特性支持

feature	Kernel version	Commit
eBPF support	3.15	bd4cf0ed331a
x86_64 JIT	3.16	622582786c9e
ARM64 JIT	3.18	e54bcde3d69d
MIPS JIT	4.13	f381bf6d82f0
Kernel helpers	3.15	bd4cf0ed331a
bpf() syscall	3.18	99c55f7d47c0
BPF attached to kprobes	4.1	2541517c32be
XDP	4.8	6a773a15a1e8
BPF attached to perf events	4.9	0515e5999a46
BPF support for sock_ops	4.13	40304b2a1567
BPF support for skbs on sockets	4.14	b005fd189cec
bpftool utility in kernel sources	4.15	71bb428fe2c1
BPF Type Format (BTF)	4.18	69b693f0aefa
AF_XDP	4.18	
BPF 1M insn limit	5.2	c04c0d2b968a
BPF iterator	5.8	180139dca8b3
BPF socket lookup hook	5.9	e9ddbb7707ff
Sleepable BPF programs	5.10	1e6c62a88215

https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

eBPF和内核模块的对比

维度	Linux 内核模块	eBPF
kprobes/tracepoints	支持	支持
安全性	可能引入安全漏洞或导致内核 Panic	通过验证器进行检查，可以保障内核安全
内核函数	可以调用内核函数	只能通过 BPF Helper 函数调用
编译性	需要编译内核	不需要编译内核，引入头文件即可
运行	基于相同内核运行	基于稳定 ABI 的 BPF 程序可以编译一次，各处运行
与应用程序交互	打印日志或文件	通过 perf_event 或 map 结构
数据结构丰富性	一般	丰富
入门门槛	高	低
升级	需要卸载和加载，可能导致处理流程中断	原子替换升级，不会造成处理流程中断
内核内置	视情况而定	内核内置支持

eBPF指令架构

架构对比

维度	cBPF	eBPF
内核版本	Linux 2.1.75（1997年）	Linux 3.18（2014年）[4.x for kprobe/uprobe/tracepoint/perf-event]
寄存器数目	2个：A, X	10个： R0–R9, 另外 R10 是一个只读的帧指针 * R0 - eBPF 中内核函数的返回值和退出值 * R1 - R5 - eBF 程序在内核中的参数值 * R6 - R9 - 内核函数将保存的被调用者callee保存的寄存器 * R10 -一个只读的堆栈帧指针
寄存器宽度	32位	64位
存储	16 个内存位: M[0–15]	512 字节堆栈，无限制大小的 “map” 存储
限制的内核调用	非常有限，仅限于 JIT 特定	有限，通过 bpf_call 指令调用
目标事件	数据包、 seccomp-BPF	数据包、内核函数、用户函数、跟踪点 PMCs 等

```
struct bpf_insn {
        __u8    code;           /* opcode */
        __u8    dst_reg:4;      /* dest register */
        __u8    src_reg:4;      /* source register */
        __s16   off;            /* signed offset */
        __s32   imm;            /* signed immediate constant */
};
```

```
Classic BPF classes:    eBPF classes:

BPF_LD    0x00          BPF_LD    0x00
BPF_LDX   0x01          BPF_LDX   0x01
BPF_ST    0x02          BPF_ST    0x02
BPF_STX   0x03          BPF_STX   0x03
BPF_ALU   0x04          BPF_ALU   0x04
BPF_JMP   0x05          BPF_JMP   0x05
BPF_RET   0x06          [ class 6 unused, for future if needed ]
BPF_MISC  0x07          BPF_ALU64 0x07
```

指令类型

存储加载指令

```
+--------+--------+--------------------+
| 3 bits | 2 bits |   3 bits           |
|  mode  |  size  | instruction class  |
+--------+--------+--------------------+
(MSB)                            (LSB)
```

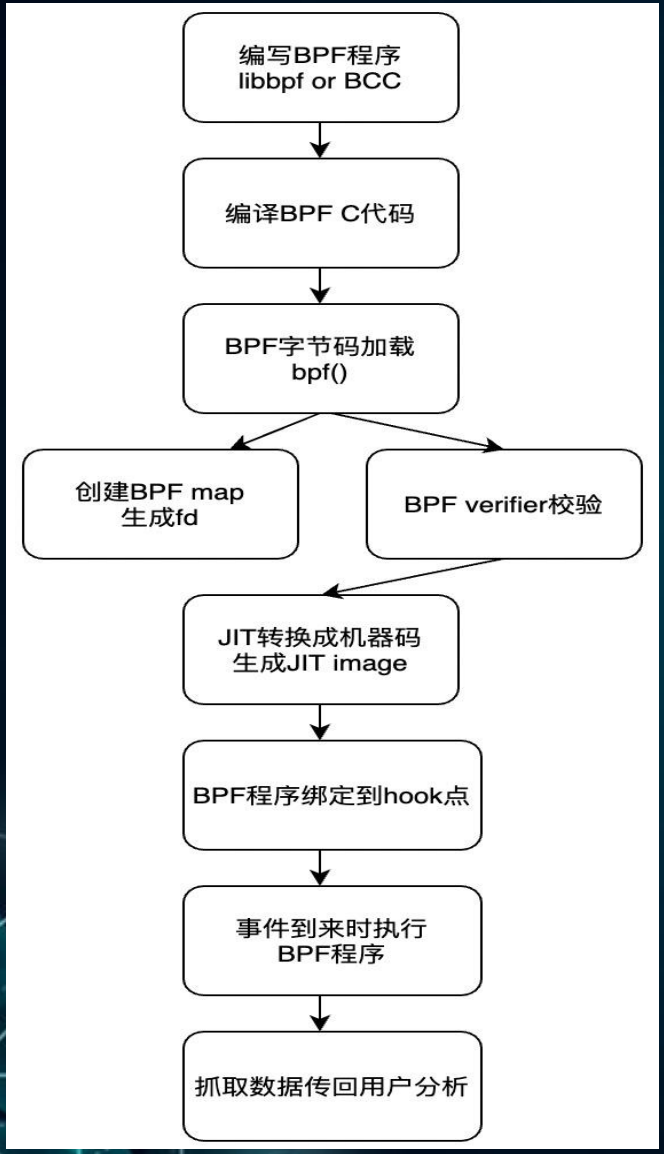
运算跳转指令

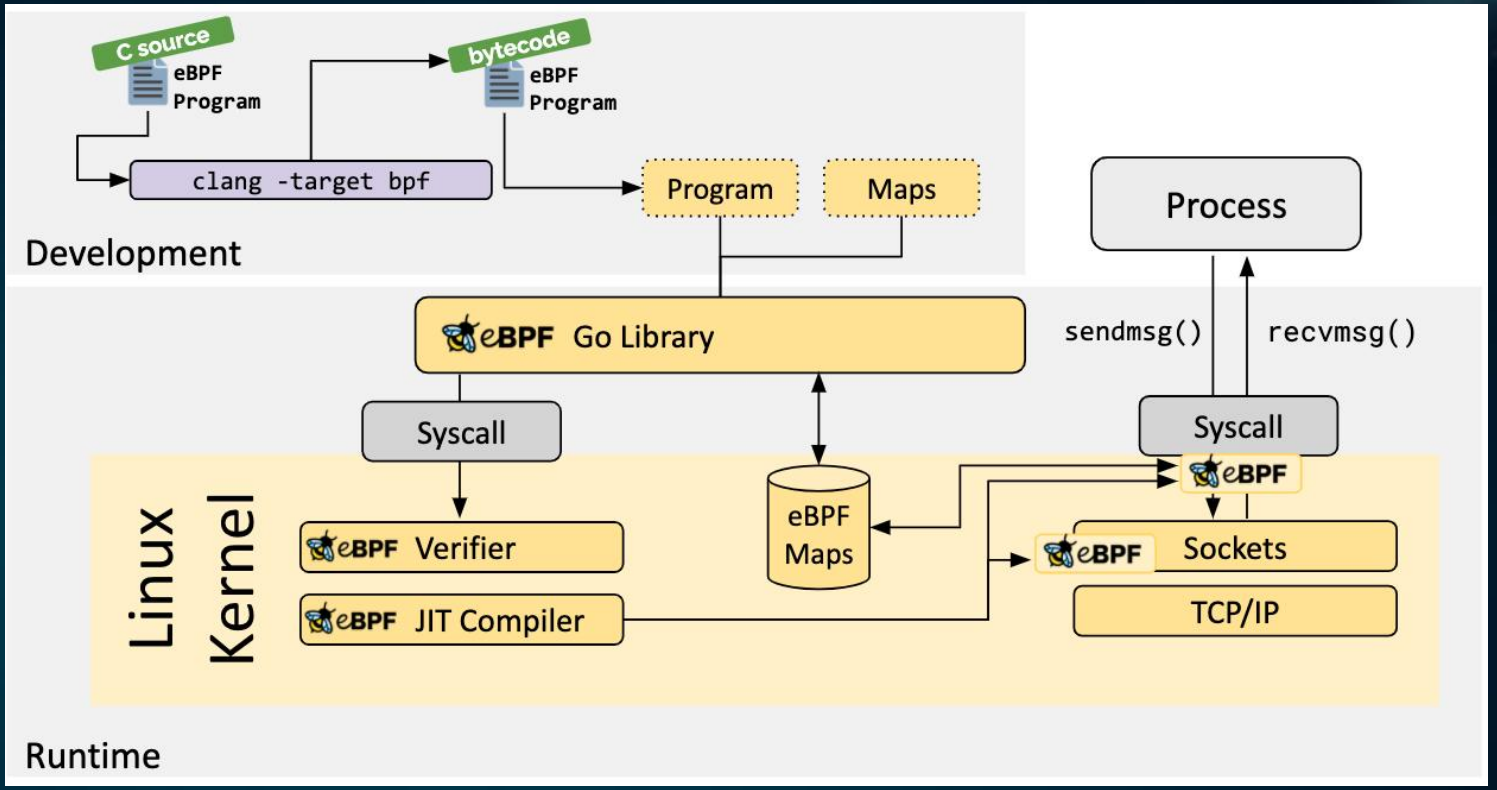
```
+----------------+--------+--------------------+
|   4 bits       | 1 bit  |   3 bits           |
| operation code | source | instruction class  |
+----------------+--------+--------------------+
(MSB)                                    (LSB)
```

02

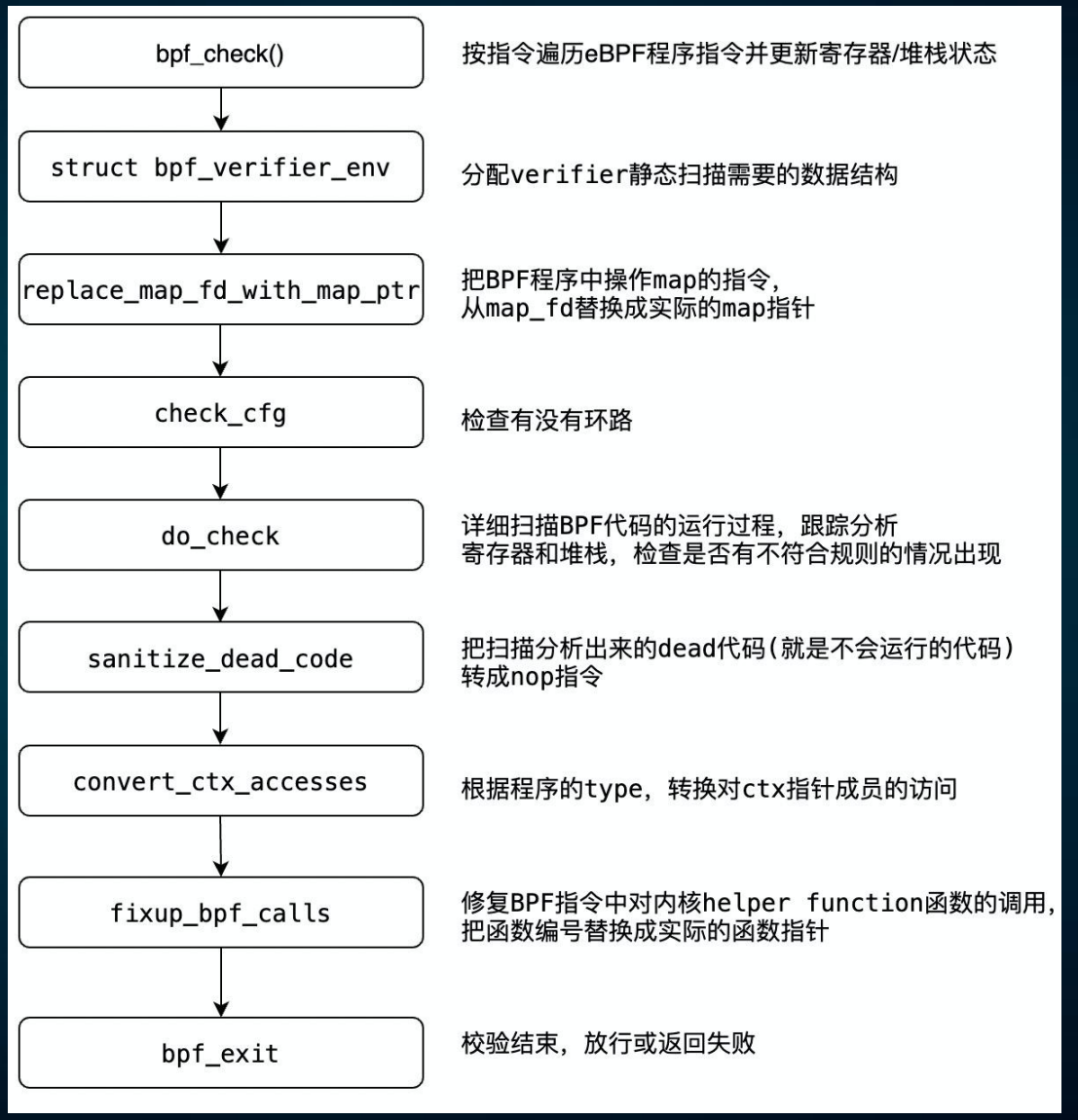
eBPF的开发方式

eBPF如何工作

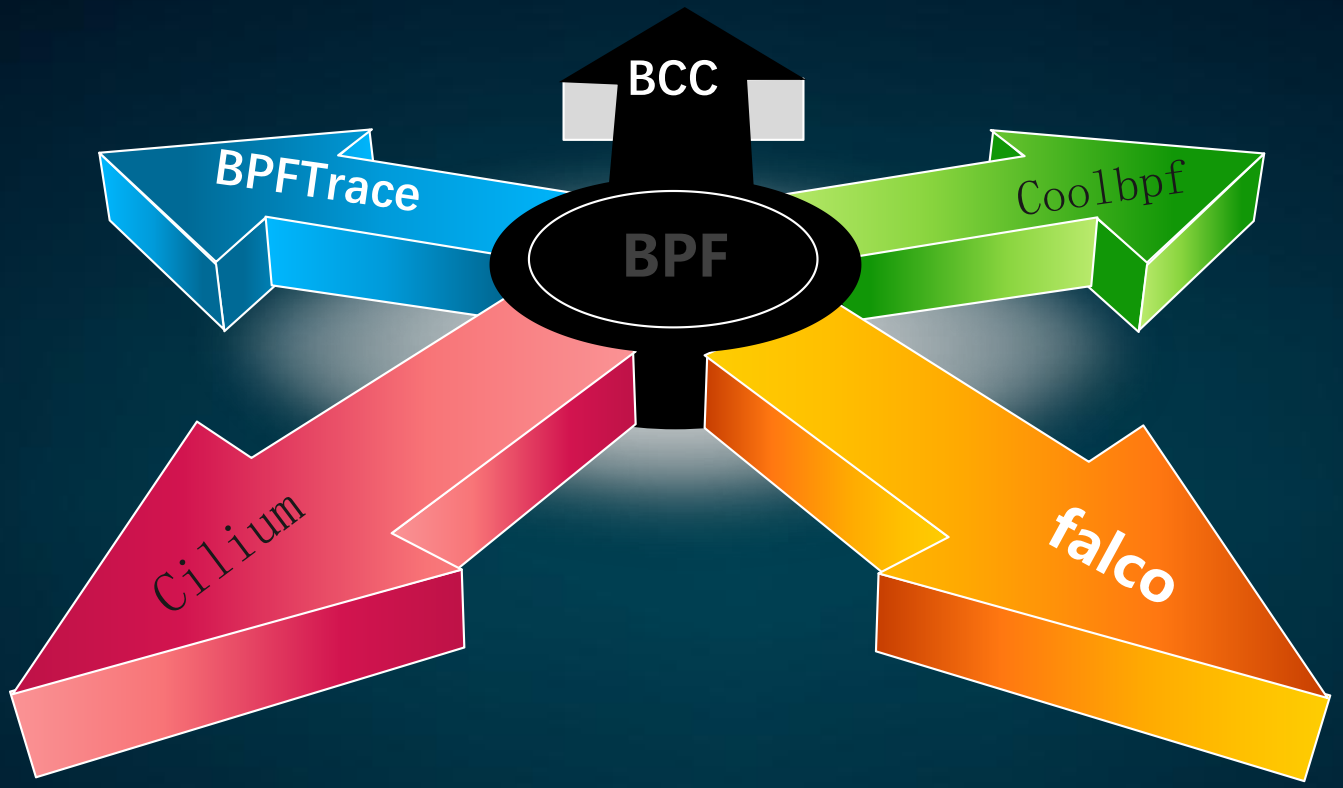

编写BPF程序
libbpf or BCC
编译BPF C代码
BPF字节码加载
bpf()
创建BPF map
生成fd
BPF verifier校验
JIT转换成机器码
生成JIT image
BPF程序绑定到hook点
事件到来时执行
BPF程序
抓取数据传回用户分析

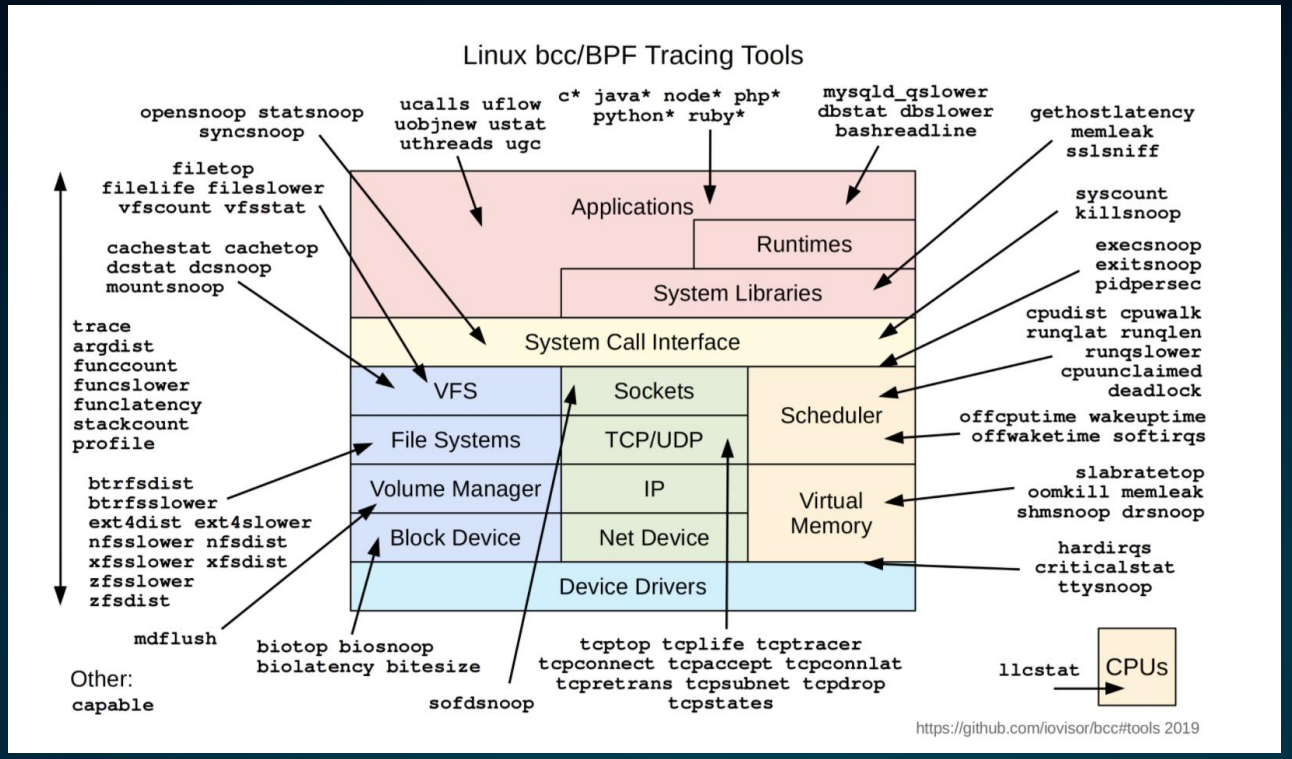

C source
eBPF
Program
bytecode
eBPF
Program
clang -target bpf
Program
Maps
Development
Process
eBPF Go Library
sendmsg()
recvmsg()
Syscall
Syscall
eBPF
eBPF
Maps
Linux
Kernel
eBPF Verifier
eBPF
Sockets
eBPF JIT Compiler
TCP/IP
Runtime

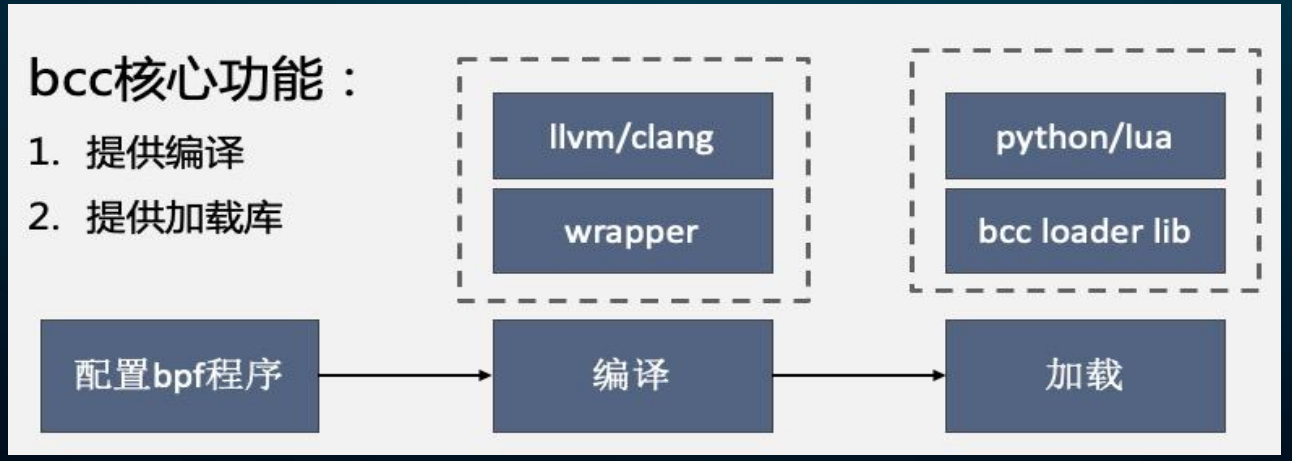
1. 不能包含循环（Direct Acyclic）
2. 检查越界指令，不允许不可达指令
3. 检查所有分支指令，并且仿真运行
4. 寄存器必须先初始化再使用
5. 禁止随机内存访问
6. 总运行步数不能超过32K
7. 分支数量被限制为1k

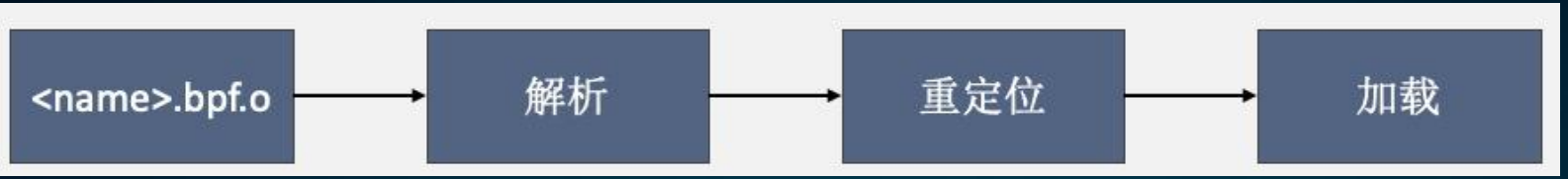


eBPF经典项目




Linux bcc/BPF Tracing Tools
opensnoop statsnoop
syncsnoop
ucalls uflow
uobjnew ustat
uthreads ugc
c* java* node* php*
python* ruby*
mysqld_qslower
dbstat dbslower
bashreadline
gethostlatency
memleak
sslsniff
filetop
filelife fileslower
vfscount vfsstat
cachestat cachetop
dcstat dcsnoop
mountsnoop
syscount
killsnoop
execsnoop
exitsnoop
pidpersec
cpudist cpuwalk
runqlat runqlen
runqslower
cpuunclaimed
deadlock
trace
argdist
funccount
funcslower
funclatency
stackcount
profile
offcputime wakeuptime
offwaketime softirqs
slabratetop
oomkill memleak
shmsnoop drsnoop
btrfsdist
btrfsslower
ext4dist ext4slower
nfsslower nfsdist
xfsslower xfsdist
zfsslower
zfsdist
hardirqs
criticalstat
ttysnoop
mdflush
biotop biosnoop
biolatency bitesize
sofdsnoop
tcptop tcplife tcptracer
tcpconnect tcpaccept tcpconnlat
tcpretrans tcpsubnet tcpdrop
tcpstates
llcstat
CPUs
Other:
capable
Applications
Runtimes
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device
Net Device
Device Drivers
https://github.com/iovisor/bcc#tools 2019


bcc核心功能：
1. 提供编译
2. 提供加载库
llvm/clang
wrapper
python/lua
bcc loader lib
配置bpf程序
编译
加载



1. 二进制解析：解析每个段，包括：

   .btf、.<prog_text>、.map、.data、.rodata、.bss等

2. 重定位：根据bpf.o的btf信息和内核现有的btf信息进行

   重定位、其它段的重定位

3. 加载：主要是map和bpf程序

BPF支持 CO-RE(Compile Once, Run everywhere)

为什么需移植

1. 需要访问原始的内核数据，不同内核版本数据的内存布局不同

2. 需要基于一个内核的两种配置来运行程序

关键组件：

1. BTF: 描述内核镜像，获取内核及BPF程序类型和代码的关键信息

2. Clang释放bpf程序重定位信息到.btf段

3. Libbpf CO-RE根据.btf段重定位bpf程序

重定位信息包括:

1. 结构体成员 & sizeof， Clang通过__builtin_preserve_access_index()记录成员偏移量

```
u64 inode = task->mm->exe_file->f_inode->i_ino;
u64 inode = BPF_CORE_READ(task, mm, exe_file, f_inode, i_ino);
```

2. 数据: map fd填充、全局变量（data、bss、rodata）、extern的变量

```
skel->rodata->my_cfg.feature_enabled = true;

skel->rodata->my_cfg.pid_to_filter = 123;
extern u32 LINUX_KERNEL_VERSION __kconfig;
extern u32 CONFIG_HZ __kconfig;
```

3. 调用地址: CO-RE将所有子程序拷贝到主程序所在区域, 如always_inline函数

1.生成带所有内核类型的头文件vmlinux.h

bpftool btf dump file vmlinux format c > vmlinux.h

2.使用Clang(版本10或更新版本)将BPF程序的源代码编译为.o对象文件;

3.从编译好的BPF对象文件中生成BPF skeleton 头文件 bpftool gen命令生成;

4.在用户空间代码中包含生成的BPF skeleton 头文件;

5.编译用户空间代码，这样会嵌入BPF对象代码，后续就不用发布单独的文件。

生成的BPF skeleton 使用如下函数触发相应的阶段:
<name>__open() – 创建并打开 BPF 应用，之后可以设置skel->rodata 变量
<name>__load() – 初始化，加载和校验BPF 应用部分;
<name>__attach() – 附加所有可以自动附加的BPF程序 (可选，可以直接使用libbpf API作更多控制);
<name>__destroy() – 分离所有的 BPF 程序并使用其使用的所有资源。

```
#include <bcc/proto.h>
#include <uapi/linux/ip.h>
#include <uapi/linux/icmp.h>
#include <uapi/linux/tcp.h>
#include <uapi/linux/udp.h>
#include <net/inet_sock.h>
#include <linux/netfilter/x_tables.h>
#include <linux/virtio.h>
#include <linux/netdevice.h>
```



BCC

需要手动添加所需的内核头文件

```
#include <vmlinux.h>
// 常用的宏定义，比如bpf_printk、offsetof等
#include <bpf/bpf_helpers.h>
// 读取内核地址的宏定义方法，比如BPF_PROBE_READ
#include <bpf/bpf_core_read.h>
// 用于从struct pt_regs获取参数的方法集合
#include <bpf/bpf_tracing.h>
// 其它的自定义头文件
#include "rtrace.h"
```



libbpf

vmlinux.h包含了所有的内核类型，但是不包含#define宏定义

BCC到libbpf的转换 bpf maps定义和访问


```
BPF_HASH(cur_ipt_do_table_args, u32, struct
ipt_do_table_args);

some_map.operation(some, args)
```



BCC

BCC提供了map的默认大小，即10240

```
struct bpf_map_def SEC("maps") cur_ipt_do_table_args = {
        .type = BPF_MAP_TYPE_HASH,
        .key_size = sizeof(u32),
        .value_size = sizeof(struct ipt_do_table_args),
        .max_entries = 10240,
};
    bpf_map_operation_elem(&some_map, some, args);
```



libbpf

需要显示地指定map的大小

tsk->parent->pid



BCC

BCC自动将其转换成两次
bpf_probe_read函数调用

支持CO-RE：

BPF_CORE_READ(tsk, parent, pid)

不支持CO-RE：

struct task_struct pt;

bpf_probe_read(&pt,sizeof(struct task_struct), tsk->parent);

pid_t pid = bpf_probe_read(&pid,sizeof(pid_t),pt->pid)



libbpf

需要显示地指定map的大小

BCC到libbpf的转换 tracepoint和 kprobe


```
TRACEPOINT_PROBE(net, netif_rx)

int kprobe__netif_rx(struct pt_regs *ctx,
struct sk_buff *skb)
```



BCC

```
SEC("tracepoint/net/netif_rx")
int tp_netif_rx(struct trace_event_raw_netif_rx  *args)

SEC("kprobe/netif_rx")
int BPF_KPROBE(kprobe__netif_rx, struct sk_buff
*skb)
```



libbpf

bcc对bpf程序配置主要包括两方面：

1. #if控制函数
2. #if控制函数内语句

```
#if TRACE_L2
int kprobe__netif_rx(struct pt_regs *ctx, struct sk_buff *skb)
{
    return do_trace(ctx, skb, __func__+8, NULL);
}
#endif
```

```
#if TRACE_CPU
    event->cpuid = bpf_get_smp_processor_id();
#endif
```

BCC

1. bpf_object__find_program_by_name和bpf_program__set_autoload来控制函数是否加载
2. 在不支持bpf全局变量的内核版本上可通过map的方式控制函数内语句

```
for (i = 0; i < ENABLE_HOOK_FUNC_NUM; i++)
{
    prog = bpf_object__find_program_by_name(obj->obj, ENABLE_HOOK_FUNC[i]);
    if (prog)
    {
        bpf_program__set_autoload(prog, true);
    }
}
```

```
if (arg->cpuid != 0)
    event->cpuid = bpf_get_smp_processor_id();
```

libbpf

03

中高版本内核 eBPF实践

BPF开发常用方案对比

原生libbpf，无CO-RE（内核samples/bpf示例）
优势：资源占用量低
缺点：
1、需要搭建代码工程、开发效率低；
2、不同内核版本兼容性差；

BCC（bpf compile collection）
优势：开发效率高，可移植性好，支持动态修改内核部分代码
缺点：
1、部署依赖的Clang/LLVM；
2、每次运行都要执行Clang/LLVM编译，争抢内存CPU内存等资源；
3、依赖目标环境头文件；

BPF CO-RE（自己编写的bpf_core_read代码）
优势：不依赖在环境中部署Clang/LLVM，资源占用少
缺点：
1、仍需要搭建编译编译工程；
2、部分代码相对固定，无法动态配置；
3、用户态开发支持信息较少，缺乏高级语言对接；

上述方案，不能很好适配生产环境中，多内核并存、快速批量部署等需求

Coolbpf 技术优势

Coolbpf取BCC和CORE的优势，以CO-RE为基础，融合了资源占用低、可移植性强、动态编译等优点

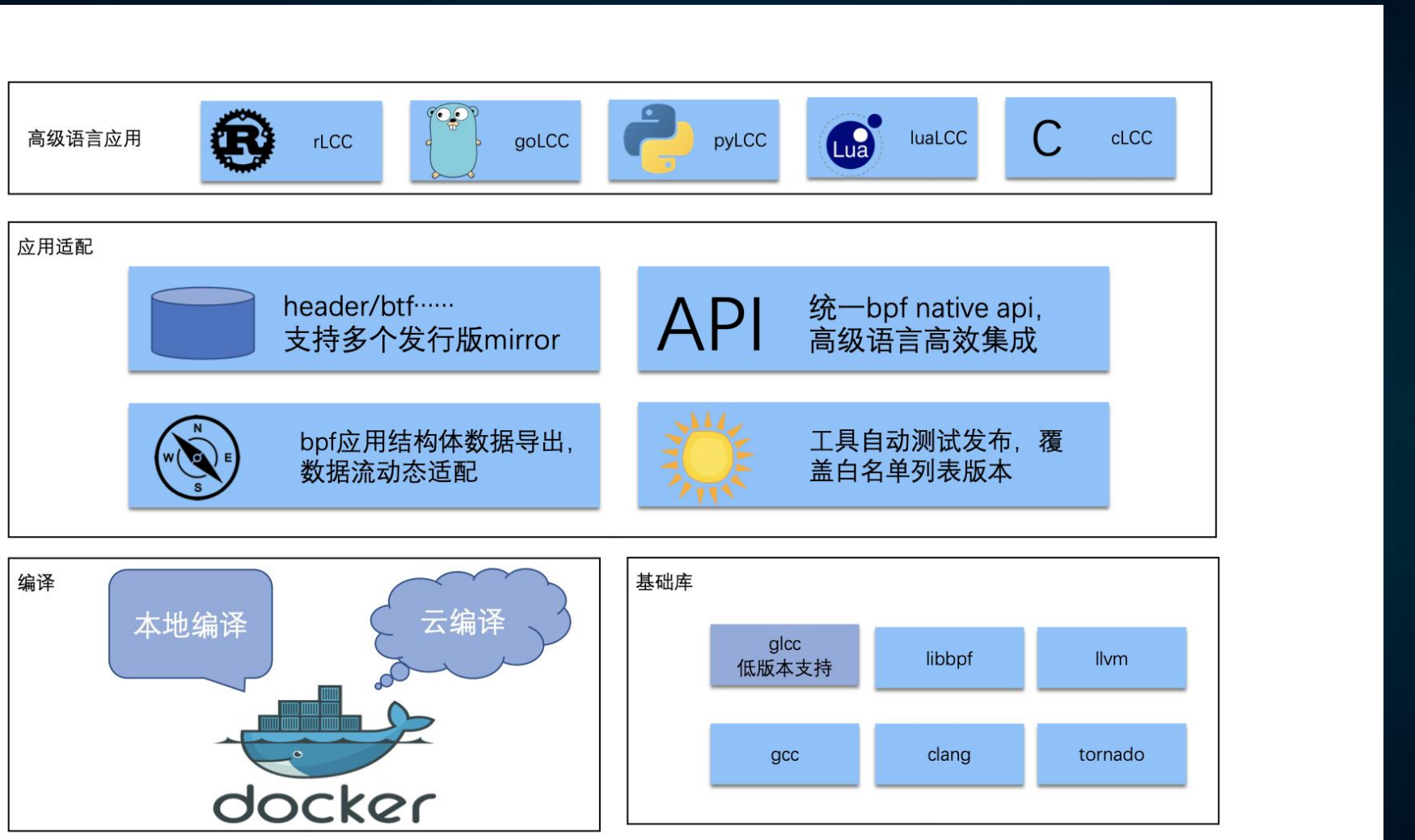


BCC
优势：开发效率高，可移植性好，支持动态修改内核部分代码

BPF CO-RE
优势：不依赖在环境中部署Clang/LLVM，资源占用少

方案优势（pyLCC为例）：
1、开箱即用：内核侧仅提供bpf.c即可，完全剥离出内核编译工程；
2、复用编译成果：本地侧无编译过程，不存在包依赖和CPU/内存等资源消耗问题；
3、自适应不同版本差异：更适合在集群多内核版本共存的场景；

Coolbpf一站式eBPF开发编译平台



Coolbpf六大功能

本地编译服务，高级功能wrap封装

远程编译服务，接收bpf.c，生成bpf.so，提供给高级语言进行加载

高版本特性通过ko方式补齐到低版本

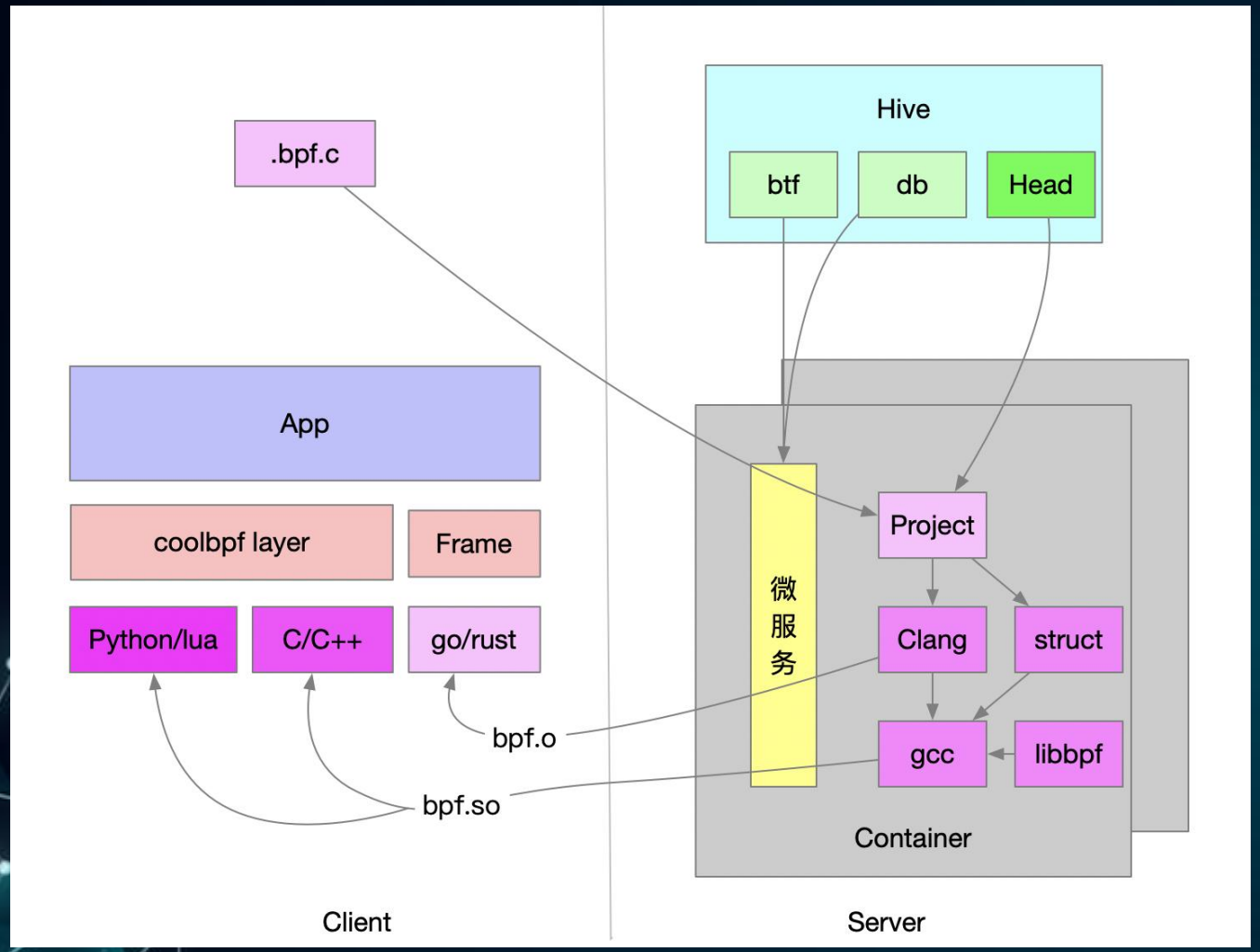
btf的自动生成和最新内核版本爬虫

多内核版本功能测试

python，go，c，rust等高级语言支持

https://lab.openanolis.cn/#/apply/chapters?courseId=94 龙蜥实验室体验教程

Coolbpf一站式eBPF开发编译平台



主要功能点：

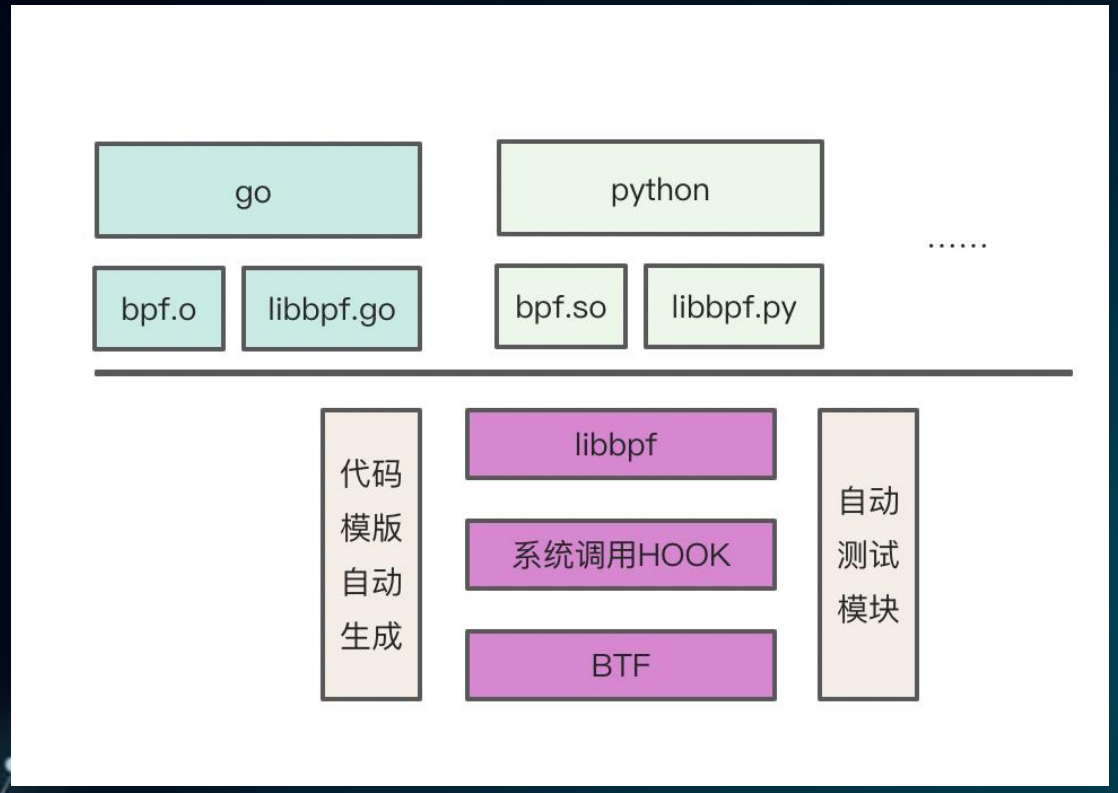
1、标准化bpf编译服务，具备弹性能力

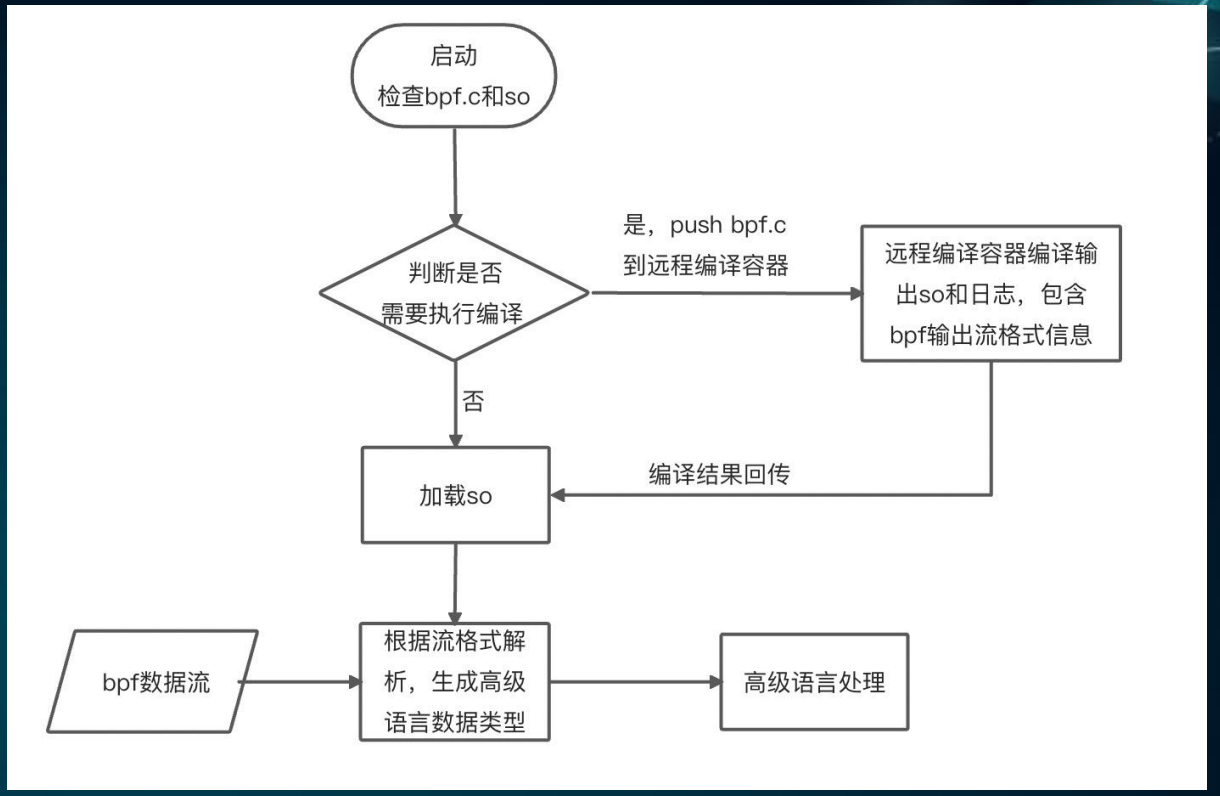
2、Hive服务已支持2000+公开发行版内核

3、输出的so支持CORE，结合BTF可以做到处处运行；

4、客户端Coolbpf中间层提供便捷化的libbpf支持api，加速内核数据流处理

pyLCC（python LCC）远程编译





1. pyLCC在libbpf基础上进行封装，将复杂的编译工程交由容器执行，可选本地编译和远程编译；
2. 本地编译也会在docker容器里运行，无需搭建复杂的环境（安装clang等库）；
3. 对生产环境没有类似BCC py脚本运行瞬时资源冲高的现象，没有资源损耗情况
4. 封装一些基础的执行函数，类似BCC的代码易用度

pyLCC（python LCC）生态支持

支持cpu 架构	x86_64/aarch64
当前支持公开发行版	4
支持公开发行版内核	1200+



1、可从pylcc.openanolis.cn 获取最新的内核支持列表

2、有libbpf和bcc基础可以轻松地从bcc转入pylcc开发

pyLCC event.py示例


```
import ctypes as ct
from pylcc.lbcBase import ClbcBase

bpfPog = r"""
#include "lbc.h"
#define TASK_COMM_LEN 16
struct data_t {
    u32 c_pid;
    u32 p_pid;
    char c_comm[TASK_COMM_LEN];
    char p_comm[TASK_COMM_LEN];
};

LBC_PERF_OUTPUT(e_out, struct data_t, 128);
SEC("kprobe/wake_up_new_task")
int j_wake_up_new_task(struct pt_regs *ctx)
{
    struct task_struct* parent = (struct task_struct *)PT_REGS_PARM1(ctx);
    struct data_t data = {};

    data.c_pid = bpf_get_current_pid_tgid() >> 32;
    bpf_get_current_comm(&data.c_comm, TASK_COMM_LEN);
    data.p_pid = BPF_CORE_READ(parent, pid);
    bpf_core_read(&data.p_comm[0], TASK_COMM_LEN, &parent->comm[0]);

    bpf_perf_event_output(ctx, &e_out, BPF_F_CURRENT_CPU, &data, sizeof(data
    return 0;
}

char _license[] SEC("license") = "GPL";
"""
```

```
class CeventOut(ClbcBase):
    def __init__(self):
        super(CeventOut, self).__init__("eventOut", bpf_str=bpfPog)

    def _cb(self, cpu, data, size):
        stream = ct.string_at(data, size)
        e = self.maps['e_out'].event(stream)
        print("current pid:%d, comm:%s. wake_up_new_task pid: %d, comm: %s"
            e.c_pid, e.c_comm, e.p_pid, e.p_comm
        ))

    def loop(self):
        self.maps['e_out'].open_perf_buffer(self._cb)
        try:
            self.maps['e_out'].perf_buffer_poll()
        except KeyboardInterrupt:
            print("key interrupt.")
            exit()

if __name__ == "__main__":
    e = CeventOut()
    e.loop()
```

1.三步搞定一个BPF应用程序，只要pip install coolbpf
2.init之后，bpf已经加载到内核，只需关注数据处理。
3.不用安装clang，不用安装kernel-dev头文件，资源消耗小
4.常用有内核版本都可以支持

pyLCC 关键技术点

1. 执行pip install coolbpf安装
2. xx.bpf.c编写:

```
bpfPog = r"""
#include "lbc.h"
LBC_PERF_OUTPUT(e_out, struct data_t, 128);
LBC_HASH(pid_cnt, u32, u32, 1024);
LBC_STACK(call_stack,32);
```

3.xx.py编写:

```
import time from pylcc.lbcBase import ClbcBase
class Pingtrace(ClbcBase): def __init__(self):
super(Pingtrace, self).__init__("pingtrace")
```

lbc.h内容:

```
#include "vmlinux.h"
#include <linux/types.h>
#include <bpf/bpf_helpers.h>
#include <bpf/bpf_core_read.h>
#include <bpf/bpf_tracing.h>
```

0. pahole准备

git://git.kernel.org/pub/scm/devel/pahole/pahole.git

1. 安装对应内核 kernel-debuginfo RPM，获得包含 DWARF 的 vmlinux

2. DWARF 生成 BTF section：pahole -J vmlinux

把涉及alios，alinux，anolis，centos以及ubunt主流内核生成vmlinux-btf库，供大家下载

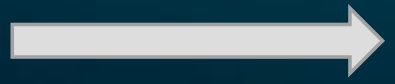
问题：

- 部分内核版本vmlinux btf段过大，如centos 3.10内核vmlinux btf段大小为30MB

问题解决：

- 3.10的BTF段包含大量的近似重复信息。导致重复信息的是类似于__UNIQUE_ID_rh_kabi_hide<num>字样的信息。
- 因为同一个结构体在不同编译单元里面的<num>是不一样的，才导致相同结构体被看成不同的。 那我们只需要去掉<num>这个差异就可以了,加一个选项 kabi-prefix
- 给pahole打了补丁后，3.10的BTF段由原来的30MB精简到3MB，生成btf的时间也大幅提升。

```
real    8m28.912s
user    8m27.271s
sys     0m1.471s
```



```
real    0m19.634s
user    0m18.457s
sys     0m1.169s
```

资源消耗情况

	pylcc	bcc
rss	10352	92288
vmpeak	207444	369672
vmdata	201284	363484

汇总对比如下表，同样的python应用，pylcc在cpu和mem等资源消耗均比bcc有较明显的优势

	pylcc	bcc
启动阶段cpu占用	0%	50%+
运行阶段rss占用	1	9

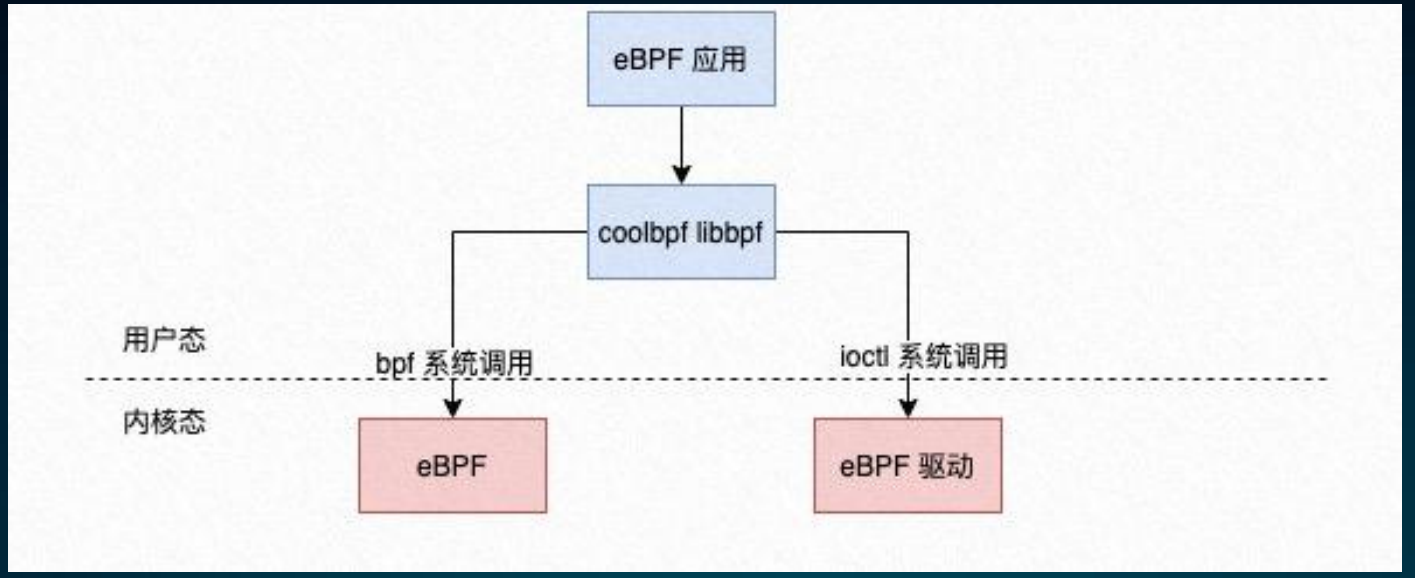
```
top - 09:05:58 up 207 days, 17:16,  2 users,  load average: 1.53, 1.36, 1.26
Tasks: 261 total,   2 running, 258 sleeping,   0 stopped,   1 zombie
%Cpu(s): 10.8 us,  2.2 sy,  0.0 ni, 85.5 id,  1.4 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 24130808 total,  2107008 free, 11896700 used, 10127100 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 11816644 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
2730530 root      20   0  345572  61672  28872 R  44.2  0.3   0:01.33 filelife
```

04

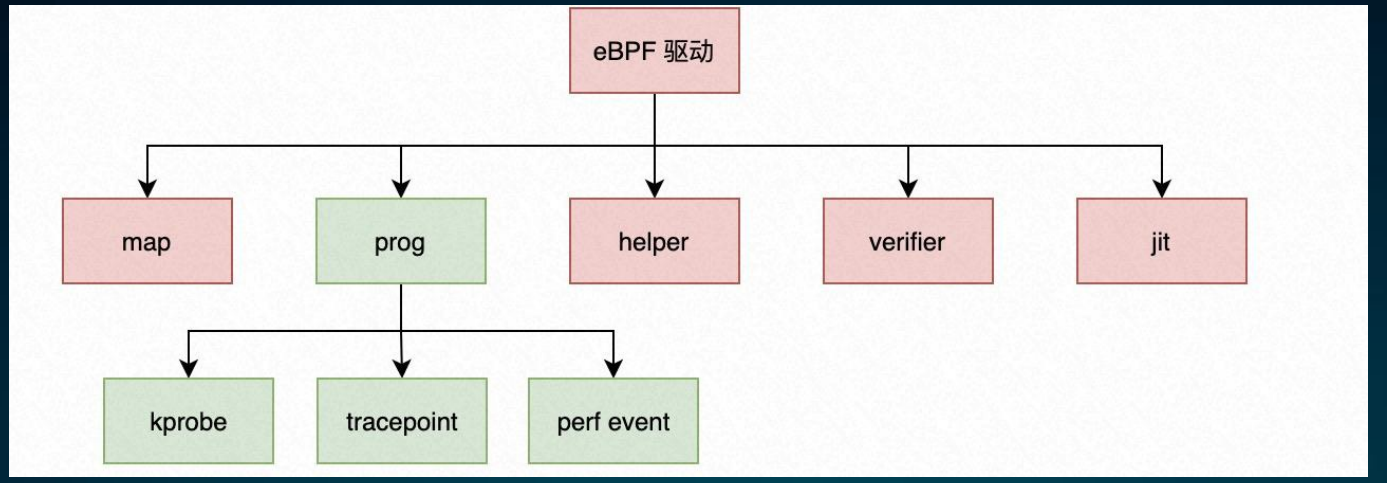
低版本内核 eBPF实践

背景和框架

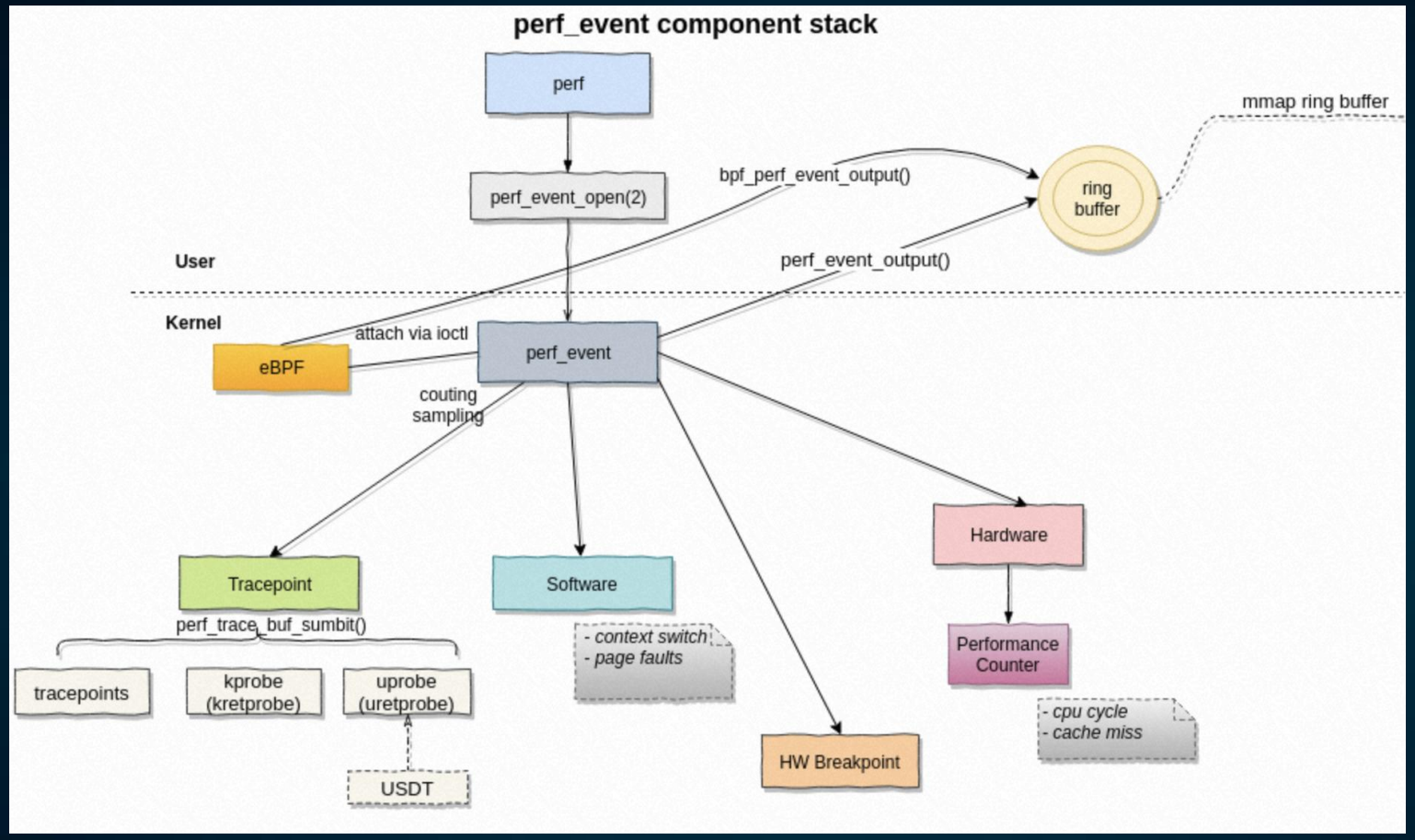


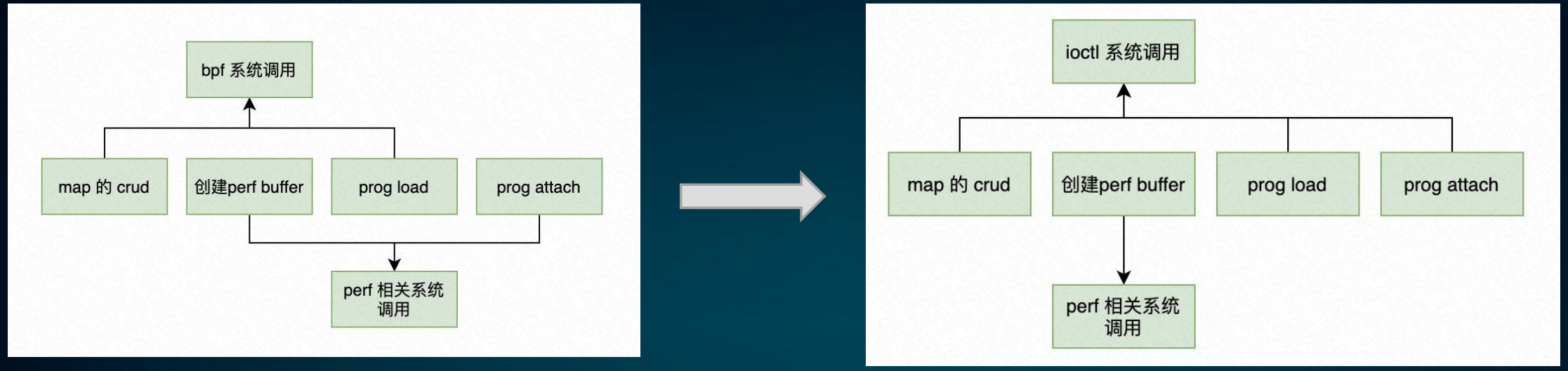
1. 目前基于eBPF编写的程序只能在高版本内核（支持eBPF的内核）上运行，无法在不支持eBPF功能的内核上运行；
2. 线上有很多Alios或者centos低版本内核需要维护
3. 存量BPF工具或项目代码，希望不做修改能跨内核运行

为此我们提出了一种在低版本内核运行eBPF程序的方法，使得二进制程序无需任何修改即可在不支持eBPF功能的内核版本上运行



1. map、helper、verifier 和 jit 和原生的 eBPF 保持一致
2. prog: eBPF 驱动中 prog 的 attach 和原生的 eBPF 略有差异
    - kprobe: perf 框架 -> kprobe 框架
    - tracepoint: perf 框架 -> tracepoint 框架
    - perf event: 均基于 perf 框架


perf_event component stack
perf
perf_event_open(2)
bpf_perf_event_output()
ring buffer
mmap ring buffer
User
perf_event_output()
Kernel
attach via ioctl
eBPF
perf_event
couting
sampling
Tracepoint
Software
Hardware
perf_trace_buf_sumbit()
- context switch
- page faults
Performance Counter
tracepoints
kprobe (kretprobe)
uprobe (uretprobe)
- cpu cycle
- cache miss
HW Breakpoint
USDT



- 通过环境变量来判断是否使能 eBPF 驱动: export ENABLE_BPF_DRV=1
- bpf 系统调用 -> ioctl 系统调用
- perf 相关系统调用（除了创建perf buffer） -> ioctl 系统调用

Coolbpf实践


```
[root@alinux2 ~]#
```

```
[root@iz2zeh94phzrw6jjdg8v54z ~]#
```

低版本内核：从无到有移植eBPF功能，高级eBPF特性移植
中版本内核：4.x 使能vmlinux-btf和libbpf CO-RE
高版本内核：5.x 可配置的btf路径及
自动化CO-RE代码框架生成
容器化编译环境，本地和远程编译

谢谢观看

CPP-Summit 2022



Sebastian Theophil

Think-cell资深软件工程师

Sebastian Theophil在柏林和图卢兹学习计算机科学，并拥有柏林洪堡大学的计算机科学博士学位。自2002年think-cell软件公司成立以来，他一直在该公司工作，最近正致力于将think-cell移植到Mac上。

主办方：Boolan 高端IT咨询与教育平台

Our Starting Point

think-cell

Our Starting Point

think-cell

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded

Our Starting Point

think-cell

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)

1914

Our Starting Point

think-cell

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)
  - Render into application-supplied objects:
    - NSView, CALayer, HWND, DirectX texture
  - Renderer needs to support DirectX and OpenGL/Metal
  - Share the main message loop
  - Support platform-specific features like host application

1915

Our Starting Point

think-cell

- 12 years of development for Windows only
- About 1.000.000 lines of C++
- Pervasive use of Windows platform-specifics through-out code base
- Product is an add-in, dynamically loaded
- Not in control of the main application (nor the computer itself!)
  - Render into application-supplied objects:
    - NSView, CALayer, HWND, DirectX texture
  - Renderer needs to support DirectX and OpenGL/Metal
  - Share the main message loop
  - Support platform-specific features like host application
- Need cross-platform toolkit that hides platform specifics and **behaves identically** on different platforms

think-cell

A Real-World Problem

1917

A Real-World Problem

think-cell

"Download file to cache in a thread-safe way. Several processes write to cache in parallel."

1918

A Real-World Problem

think-cell

"Download file to cache in a thread-safe way. Several processes write to cache in parallel."

Idea:

```
MakeTempFile();

DownloadDataToTempFile();

RenameTempToTargetIfNotExists();
```

1919

A Real-World Problem

think-cell

1. Create a temporary file

The standard offers

```
std::FILE* f = std::tmpfile();
```

https://en.cppreference.com/w/cpp/io/c/tmpfile

1920

A Real-World Problem

think-cell

1. Create a temporary file

The standard offers

```
std::FILE* f = std::tmpfile();
```

https://en.cppreference.com/w/cpp/io/c/tmpfile

If the program closes the file, ..., the file is automatically deleted.
If the program terminates normally [... the file is] also automatically deleted.



1921

A Real-World Problem

think-cell

1. Create a temporary file

The standard offers

```
std::FILE* f = std::tmpfile();
```

https://en.cppreference.com/w/cpp/io/c/tmpfile

If the program closes the file, ..., the file is automatically deleted.
If the program terminates normally [... the file is] also automatically deleted.



If the program terminates abnormally, it is implementation-defined if these temporary files are deleted.

1922

A Real-World Problem

think-cell

2. Download data to a temporary file

No standard solution

- Build something with boost.beast?
- Use libcurl?

1923

A Real-World Problem

think-cell

2. Download data to a temporary file

No standard solution

- Build something with boost.beast?
- Use libcurl?

But: We want to use system network and proxy settings!

A Real-World Problem

think-cell

2. Download data to a temporary file

No standard solution

- Build something with boost.beast?
- Use libcurl?

But: We want to use system network and proxy settings!

Lesson 1: A cross-platform application must be a good application on each platform.

A Real-World Problem

think-cell

2. Download data to a temporary file

https://developer.apple.com/documentation/foundation/nsurlsession

"The NSURLSession class natively supports the data, file, ftp, http, and https URL schemes, with transparent support for proxy servers and SOCKS gateways, as configured in the user's system preferences."

1926

A Real-World Problem

think-cell

2. Download data to a temporary file

```
std::FILE* f = std::tmpfile();
http_get_request(strUrl, [&](auto&& rngbyte) {
    std::fwrite(
        std::ranges::data(rngbyte),
        1,
        std::ranges::size(rngbyte),
        f
    );
});
```

1927

A Real-World Problem

think-cell

2. Download data to a temporary file

```
std::FILE* f = std::tmpfile();
http_get_request(strUrl, [&](auto&& rngbyte) {
    std::fwrite(
        std::ranges::data(rngbyte),
        1,
        std::ranges::size(rngbyte),
        f
    );
});
```

Lesson 2: We can treat http_get_request as a black box. The data going in and data going out are clearly defined. That is the easiest case.

A Real-World Problem

think-cell

3. Rename the temporary file to the target file name

```
void rename(const std::filesystem::path& old_p,
            const std::filesystem::path& new_p);
```

1929

A Real-World Problem

think-cell

3. Rename the temporary file to the target file name

```
void rename(const std::filesystem::path& old_p,
            const std::filesystem::path& new_p);
```

- Does not work with `std::FILE*`
- Temp file still has to exist on disk
- Does not specify access rights, atomicity
- Always overwrites target location

Can we write a better cross-platform rename?

1930

A Real-World Problem

think-cell

```
BOOL MoveFileExW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, DWORD dwFlags);
```

`MOVEFILE_COPY_ALLOWED` Allow copy when moving to different volume

`MOVEFILE_REPLACE_EXISTING` Overwrite target

`MOVEFILE_DELAY_UNTIL_REBOOT` Move when system is rebooted

1931

A Real-World Problem

think-cell

```
BOOL MoveFileExW(LPCWSTR lpExistingFileName, LPCWSTR lpNewFileName, DWORD dwFlags);
```

`MOVEFILE_COPY_ALLOWED` Allow copy when moving to different volume

`MOVEFILE_REPLACE_EXISTING` Overwrite target

`MOVEFILE_DELAY_UNTIL_REBOOT` Move when system is rebooted

```
int renamex_np(char const* from, char const* to, unsigned int flags); // darwin
int renameat2(int olddirfd, const char *oldpath,
              int newdirfd, const char *newpath, unsigned int flags); // Linux
```

`RENAME_SWAP/RENAME_EXCHANGE` Swap source and target atomically on supported file systems

`RENAME_EXCL/RENAME_NOREPLACE` Fail if destination already exists on supported file systems

1932

A Real-World Problem

think-cell

```
int copyfile(char const* from, char const* to, copyfile_state_t state,
             copyfile_flags_t flags)
```

`COPYFILE_ACL` Copy the source file's access control lists.

`COPYFILE_DATA` Copy the source file's data.

`COPYFILE_EXCL` Fail if the to file already exists

`COPYFILE_MOVE` Unlink source

1933

A Real-World Problem

think-cell

```
int copyfile(char const* from, char const* to, copyfile_state_t state,
             copyfile_flags_t flags)
```

`COPYFILE_ACL` Copy the source file's access control lists.

`COPYFILE_DATA` Copy the source file's data.

`COPYFILE_EXCL` Fail if the to file already exists

`COPYFILE_MOVE` Unlink source

```
@class NSFileManager
- (BOOL)replaceItemAtURL:(NSURL *)originalItemURL
           withItemAtURL:(NSURL *)newItemURL
          backupItemName:(NSString *)backupItemName
                 options:(NSFileManagerItemReplacementOptions)options
        resultingItemURL:(NSURL * _Nullable *)resultingURL
                   error:(NSError * _Nullable *)error;
```

1934

A Real-World Problem

think-cell

What is their common denominator?

- and is it more powerful than `std::filesystem::rename`?
- does it really do the same thing on each platform?
- does it really do a useful thing on each platform?

1935

A Real-World Problem

think-cell

What is their common denominator?

- and is it more powerful than `std::filesystem::rename`?
- does it really do the same thing on each platform?
- does it really do a useful thing on each platform?
- What about ACLs?
- What about locked files on Windows and `MOVEFILE_DELAY_UNTIL_REBOOT`?

1936

A Real-World Problem

think-cell

What is their common denominator?

- and is it more powerful than `std::filesystem::rename`?
- does it really do the same thing on each platform?
- does it really do a useful thing on each platform?
- What about ACLs?
- What about locked files on Windows and `MOVEFILE_DELAY_UNTIL_REBOOT`?

Lesson 3: For a good cross-platform interface both, syntax and semantics have to match!

1937

A Real-World Problem

think-cell

bool QFile::setPermissions(QFileDevice::Permissions *permissions*) `[override virtual]`

Reimplements: QFileDevice::setPermissions(QFileDevice::Permissions permissions).

Sets the permissions for the file to the *permissions* specified. Returns `true` if successful, or `false` if the permissions cannot be modified.

> **Warning:** This function does not manipulate ACLs, which may limit its effectiveness.

1938

A Real-World Problem

think-cell

What to do?

1. **Change our algorithm slightly**

```
MakeTempFileInTargetDir(); // ACL correct

DownloadDataToTempFile();

RenameTempToTargetIfNotExists();
```

1939

A Real-World Problem

think-cell

What to do?

1. **Change our algorithm slightly**

```
MakeTempFileInTargetDir(); // ACL correct

DownloadDataToTempFile();

RenameTempToTargetIfNotExists();
```

2. **Change the abstraction level**

```
std::FILE* CacheAndOpenFile(std::string const& strUrl);
```

- Semantics are clear and high-level
- Exploit operating system guarantees
- Increased degrees of freedom for the implementor

1940

A Real-World Problem

think-cell

Lesson 4: Write high-level cross-platform interface abstractions with strong semantics!

- These can share code again where this makes sense (e.g. `http_get_request`)
- Identical low-level semantics are often impossible to define

Examples

- download a file to a cache in thread-safe way
- create a user application settings file
- create a temporary file that is automatically deleted but which can be opened by another application
- create a document in a user-specified folder,
  open system-specific “Save File" dialog,
  and create sandbox exception
- Functions with **strong** and **identical** semantics

1941

think-cell

What About Cross-Platform Libraries?

1942

Cross-Platform Libraries

think-cell

- We use shared memory to implement inter-process communication
- Boost.Interprocess (like Qt) offers a common API for shared memory on Windows and Posix



gettyimages®
OAuth 2.0









1943

Cross-Platform Libraries

think-cell

- We use shared memory to implement inter-process communication
- Boost.Interprocess (like Qt) offers a common API for shared memory on Windows and Posix
- named memory objects mappable into different processes
  - `boost::interprocess::managed_shared_memory`
- pointers stored in shared memory
  - `boost::interprocess::offset_ptr`
  - store offset to their own this pointer
  - shared memory can be mapped at different addresses
- named synchronization objects
  - `boost::interprocess::named_mutex`

1944

Cross-Platform Libraries

think-cell

Server process:

```
#include <boost/interprocess/managed_shared_memory.hpp>
...
using namespace boost::interprocess;
using T = std::pair<double, int>;

// Name specific to build version, host,  security identifier
char const szName[] = { ... };
managed_shared_memory seg(create_only, szName, /* size */ 65536);
T* p = seg.construct<T>("A") // name of the object
    (10.0, 0); // ctor first argument
```

Child process:

```
// Open managed shared memory
managed_shared_memory seg(open_only, szName);

// Find object
T* p = seg.find<T>("A").first;
assert(p->first == 10.0);
```

1945

Cross-Platform Libraries

think-cell

Server process:

```
#include <boost/interprocess/managed_shared_memory.hpp>
...
using namespace boost::interprocess;
using T = std::pair<double, int>;

// Name specific to build version, host,  security identifier
char const szName[] = { ... };
managed_shared_memory seg(create_only, szName, /* size */ 65536);
T* p = seg.construct<T>("A") // name of the object
    (10.0, 0); // ctor first argument
```

Child process:

```
// Open managed shared memory
managed_shared_memory seg(open_only, szName);

// Find object
T* p = seg.find<T>("A").first;
assert(p->first == 10.0);
```

1946

Cross-Platform Libraries

think-cell

On Posix, shared memory is either

1. File-backed:
   backing file still exists if processes crash, even after reboot
2. Posix shared memory (`shm_open`/`shm_unlink`):
   backing memory persists until reboot

1947

Cross-Platform Libraries

think-cell

On Posix, shared memory is either

1. File-backed:
   backing file still exists if processes crash, even after reboot
2. Posix shared memory (`shm_open`/`shm_unlink`):
   backing memory persists until reboot

On Windows, if last process accessing the named memory segment dies, memory is freed

```
HANDLE hMapFile = CreateFileMapping(
    /* use page file */ INVALID_HANDLE_VALUE,
    /* security attributes */ NULL,
    PAGE_READWRITE,
    0,
    /* size */ 256,
    /* object name */ szName);
```

1948

Cross-Platform Libraries

think-cell

On Posix, shared memory is either

1. File-backed:
   backing file still exists if processes crash, even after reboot
2. Posix shared memory (`shm_open`/`shm_unlink`):
   backing memory persists until reboot

On Windows, if last process accessing the named memory segment dies, memory is freed

```
HANDLE hMapFile = CreateFileMapping(
    /* use page file */ INVALID_HANDLE_VALUE,
    /* security attributes */ NULL,
    PAGE_READWRITE,
    0,
    /* size */ 256,
    /* object name */ szName);
```

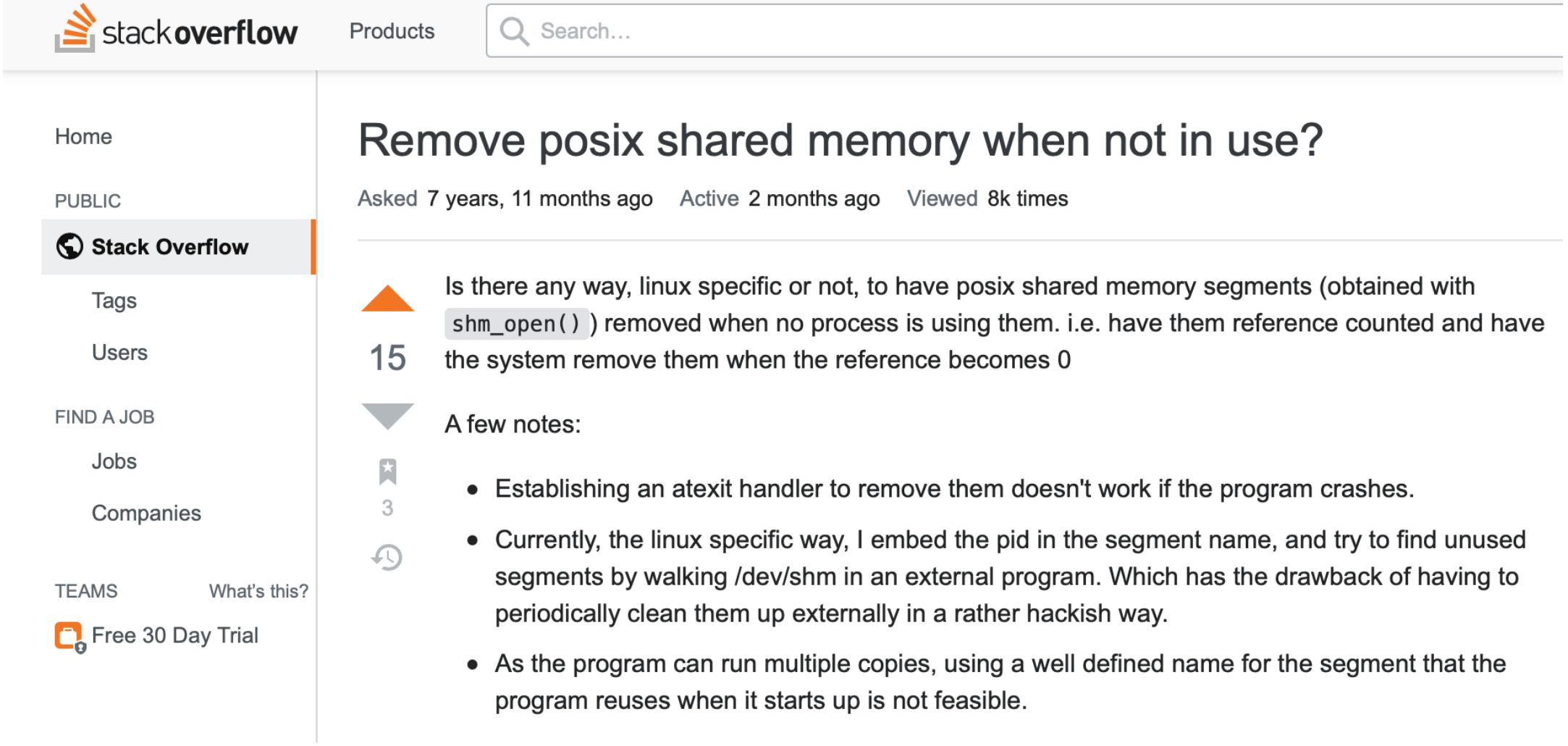
This is a guarantee you want ... but it is not provided by Posix systems nor by boost.interprocess

1949

Cross-Platform Libraries

think-cell

- This seems to cause problems occasionally:

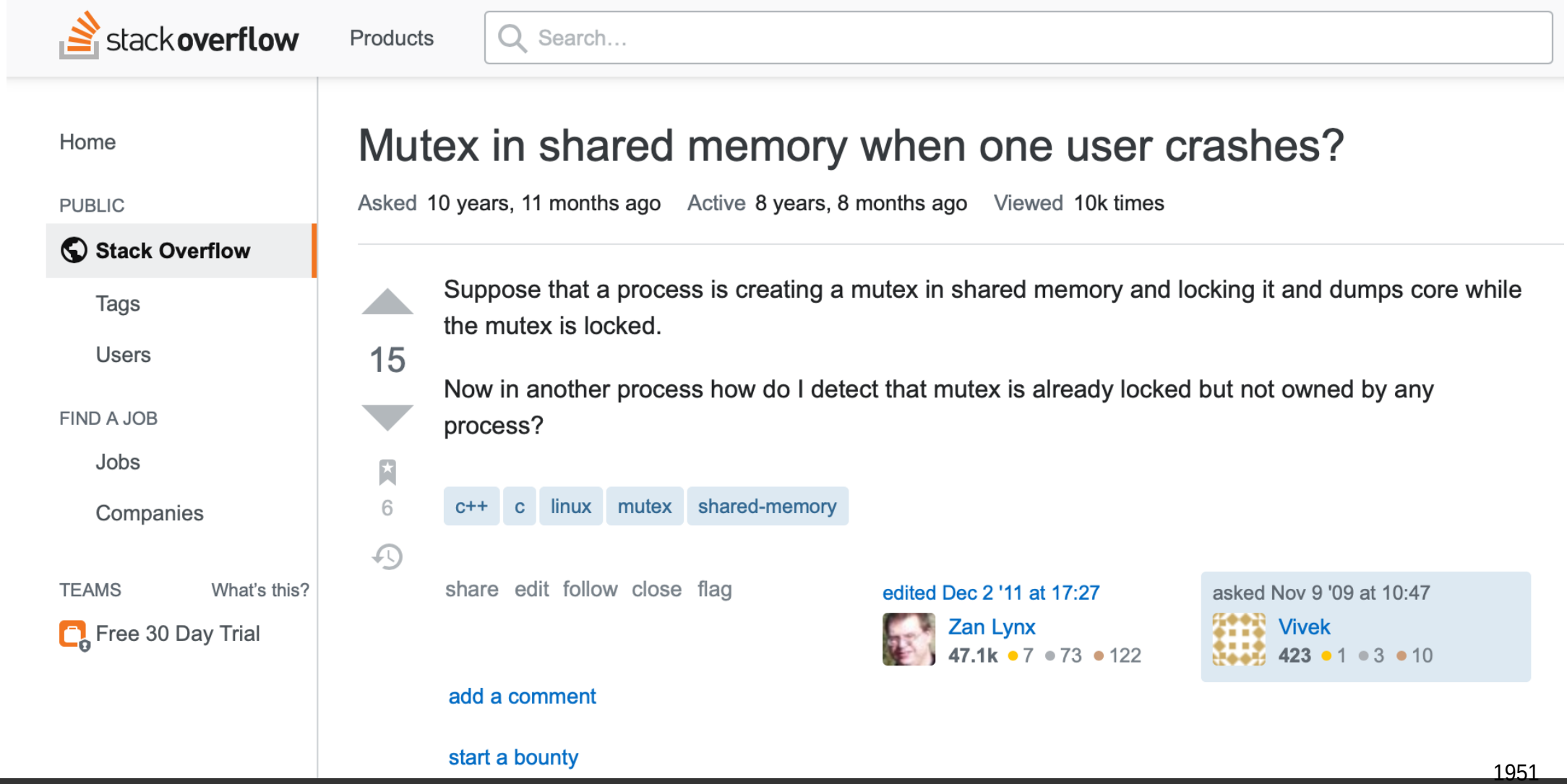


1950

Cross-Platform Libraries

think-cell

- This seems to cause problems occasionally:



stackoverflow Products Search…

Home

PUBLIC

Stack Overflow

Tags

Users

FIND A JOB

Jobs

Companies

TEAMS What's this?

Free 30 Day Trial

Mutex in shared memory when one user crashes?

Asked 10 years, 11 months ago Active 8 years, 8 months ago Viewed 10k times

15

Suppose that a process is creating a mutex in shared memory and locking it and dumps core while the mutex is locked.

Now in another process how do I detect that mutex is already locked but not owned by any process?

6

c++ c linux mutex shared-memory

share edit follow close flag

edited Dec 2 '11 at 17:27

Zan Lynx

47.1k ● 7 ● 73 ● 122

asked Nov 9 '09 at 10:47

Vivek

423 ● 1 ● 3 ● 10

add a comment

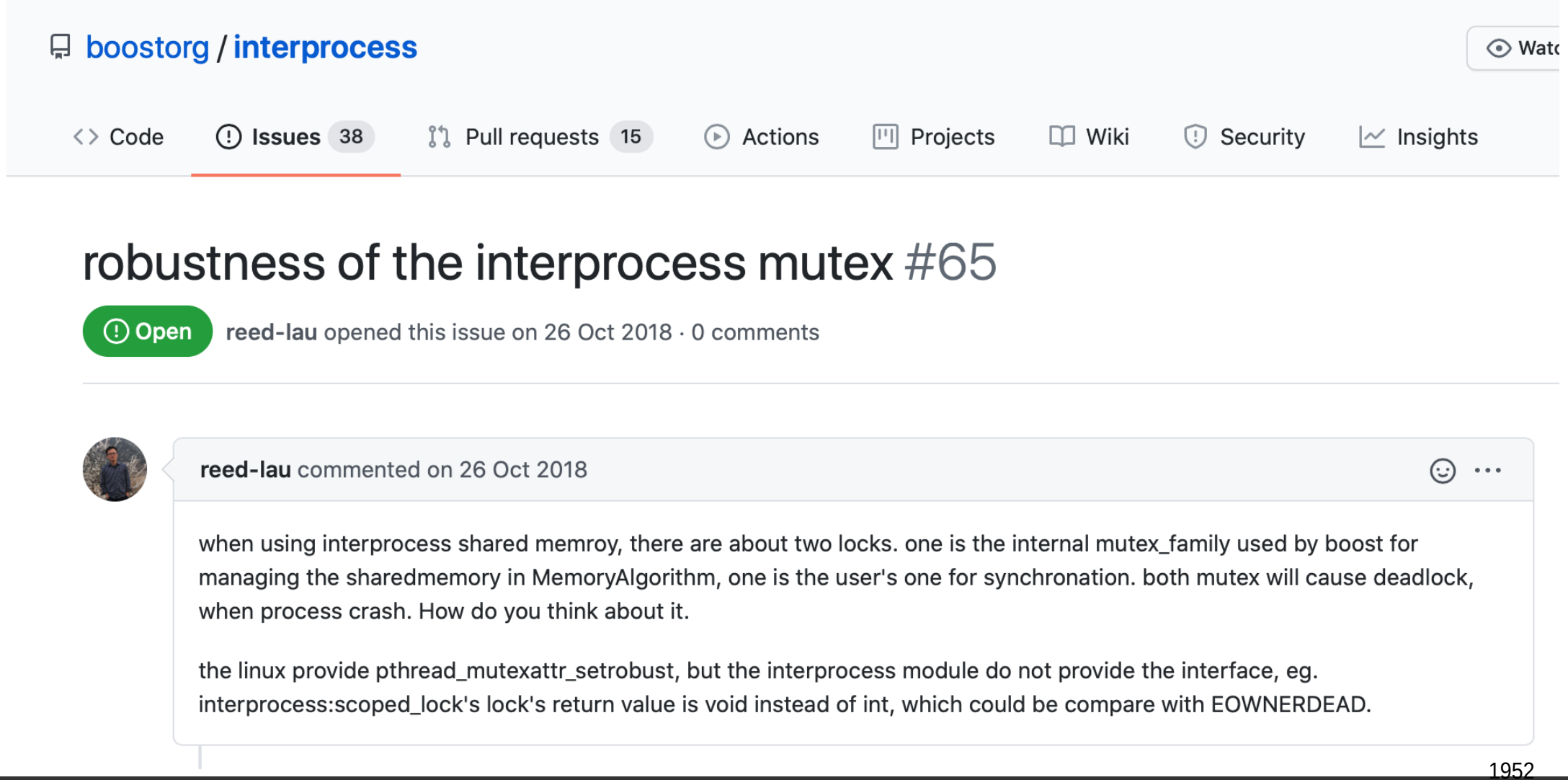
start a bounty

1951

Cross-Platform Libraries

think-cell

- This seems to cause problems occasionally:



boostorg / interprocess

Code | Issues 38 | Pull requests 15 | Actions | Projects | Wiki | Security | Insights

robustness of the interprocess mutex #65

Open **reed-lau** opened this issue on 26 Oct 2018 · 0 comments

reed-lau commented on 26 Oct 2018

when using interprocess shared memroy, there are about two locks. one is the internal mutex_family used by boost for managing the sharedmemory in MemoryAlgorithm, one is the user's one for synchronation. both mutex will cause deadlock, when process crash. How do you think about it.

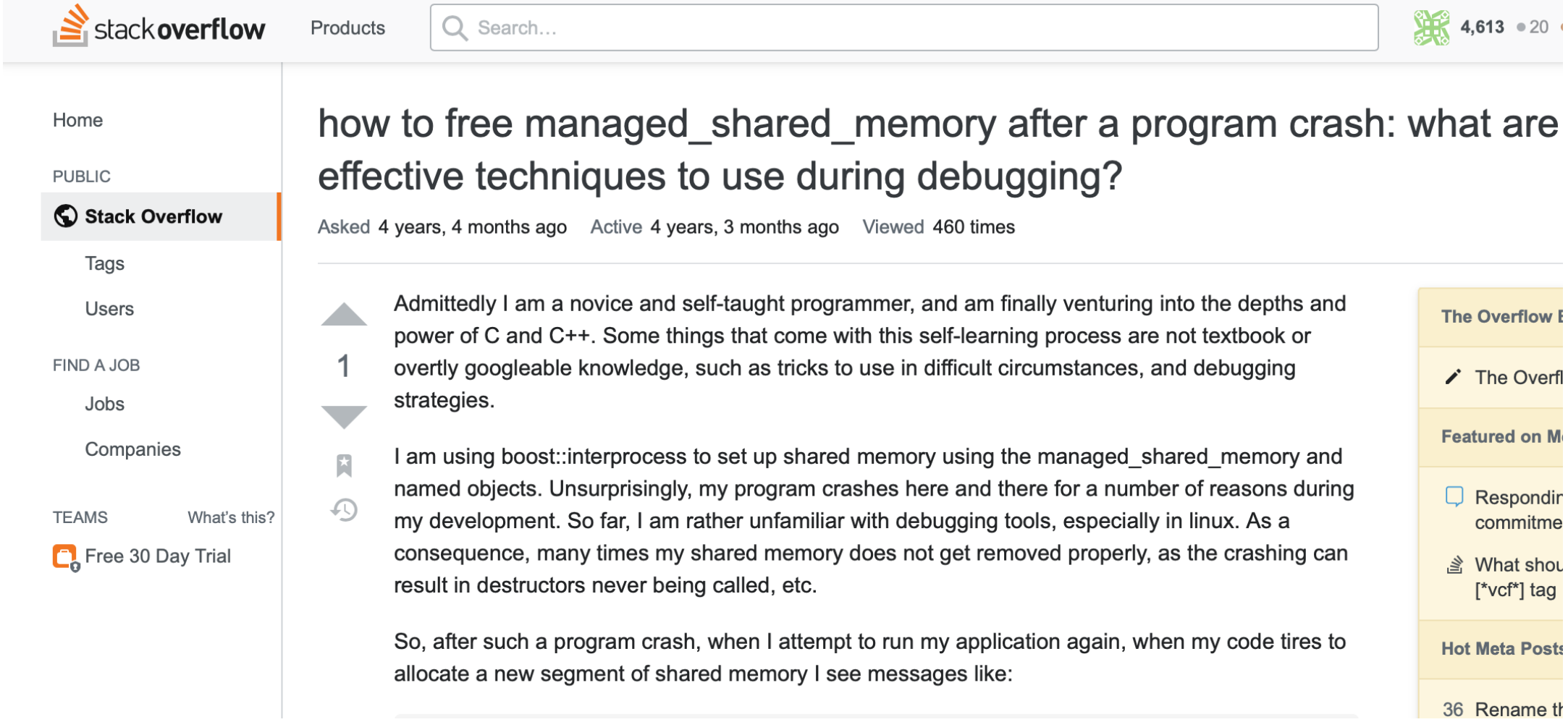
the linux provide pthread_mutexattr_setrobust, but the interprocess module do not provide the interface, eg. interprocess:scoped_lock's lock's return value is void instead of int, which could be compare with EOWNERDEAD.

1952

Cross-Platform Libraries

think-cell

- This seems to cause problems occasionally:



1953

Cross-Platform Libraries

think-cell

- This seems to cause problems occasionally:

[Boost-users] [interprocess] named mutex clean up 90 views



Chard

to boost...@lists.boost.org

Does calling named_mutex::remove() have consistent cross-platform behaviour?

The reason I ask is that I am trying to use a named mutex for an "I'm the only process" check.

That is, the process takes a shared lock on the named mutex (at start up), then, at points within the program, it attempts to get an exclusive lock in order to perform the check/actions. When finished, the mutex is returned to a shared lock.

1954

Cross-Platform Libraries

think-cell

- On Windows, shared memory and mutexes are reference counted,
  i.e., when the last user dies or crashes, resource is freed
- boost.interprocess reimplements weak Posix semantics on Windows because that is easy

Lesson 6: Exploit strong operating system guarantees!

1955

Cross-Platform Libraries

think-cell

- On Windows, shared memory and mutexes are reference counted,
  i.e., when the last user dies or crashes, resource is freed
- boost.interprocess reimplements weak Posix semantics on Windows because that is easy

Lesson 6: Exploit strong operating system guarantees!

Lesson 7: Cross-platform toolkits cannot magically fix huge semantic differences between platforms. But you might!

1956

Cross-Platform Libraries

think-cell

- On Windows, shared memory and mutexes are reference counted,
  i.e., when the last user dies or crashes, resource is freed
- boost.interprocess reimplements weak Posix semantics on Windows because that is easy

Lesson 6: Exploit strong operating system guarantees!

Lesson 7: Cross-platform toolkits cannot magically fix huge semantic differences between platforms. But you might!

Can we do better under our specific constraints?

1957

Cross-Platform Libraries

think-cell

File Locking

- the only resource that has process-lifetime are file locks
- **Attention!** File locks are weird on Posix:
  "Everything you never wanted to know about file locking" https://apenwarr.ca/log/20101213

1958

Cross-Platform Libraries

think-cell

Emulating Windows shared memory behavior on Posix systems:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, O_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request https://github.com/boostorg/interprocess/pull/132

1959

Cross-Platform Libraries

think-cell

Emulating Windows shared memory behavior on Posix systems:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, O_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request https://github.com/boostorg/interprocess/pull/132

1960

Cross-Platform Libraries

think-cell

Emulating Windows shared memory behavior on Posix systems:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, O_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request https://github.com/boostorg/interprocess/pull/132

1961

Cross-Platform Libraries

think-cell

Emulating Windows shared memory behavior on Posix systems:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, O_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request https://github.com/boostorg/interprocess/pull/132

1962

Cross-Platform Libraries

think-cell

Emulating Windows shared memory behavior on Posix systems:

```
handle = ::open(strFile, O_CREAT | O_EXLOCK | O_NONBLOCK, perm);
if(ipcdetail::invalid_file()==handle) {
    ...
    // We are not the first to open backing file
    // block until we get shared lock instead
    handle = ::open(strFile, O_SHLOCK);
} else {
    ipcdetail::truncate_file(handle, 0);
    // Degrade lock to shared lock when we have truncated file
    // Not actually atomic?
    // What about NFS?
    // Not thread-safe
    flock(m_handle, LOCK_SH);
    ...
}
```

Pull request https://github.com/boostorg/interprocess/pull/132

1963

Cross-Platform Libraries

think-cell

Lesson 6: Exploit strong operating system guarantees!

... guarantees are good!
... strong guarantees give you strong semantics
... don't sacrifice operating system guarantees for identical API!

Lesson 7: Cross-platform toolkits cannot magically fix huge semantic differences between platforms. But you might!

... sometimes you can do better than existing toolkits!
... because you know your constraints

1964

Simplifying State Machines

1965

Simplifying State Machines

think-cell

- Superficially, Windows and macOS have a similar event handling architecture
- Windows sends messages to windows:

```
struct CMyWindow: ATL::CWindowImpl<CMyWindow>
{
    // window message map
    BEGIN_MSG_MAP(CMyWindow)
        MESSAGE_HANDLER(WM_MOUSEMOVE, OnMouseMove)
        MESSAGE_HANDLER(WM_LBUTTONDOWN, OnButtonDown)
        MESSAGE_HANDLER(WM_LBUTTONDBLCLK, OnDoubleClick)
    END_MSG_MAP()
    ...
    LRESULT OnMouseMove(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
    LRESULT OnButtonDown(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
    LRESULT OnDoubleClick(UINT nMsg, WPARAM wparam, LPARAM lparam,
        BOOL& bHandled);
};
```

1966

Simplifying State Machines

think-cell

- Superficially, Windows and macOS have a similar event handling architecture
- macOS handles the messages and calls event handlers directly:

```
@interface MyView : NSView {
    - (void)mouseMoved:(NSEvent*)nsevent;
    - (void)mouseDragged:(NSEvent*)nsevent;
    - (void)mouseDown:(NSEvent*)nsevent;
}
```

1967

Simplifying State Machines

think-cell

- Superficially, Windows and macOS have a similar event handling architecture
- macOS handles the messages and calls event handlers directly:

```
@interface MyView : NSView {
    - (void)mouseMoved:(NSEvent*)nsevent;
    - (void)mouseDragged:(NSEvent*)nsevent;
    - (void)mouseDown:(NSEvent*)nsevent;
}
```

- The semantical differences are large though

1968

Simplifying State Machines

think-cell

Windows	macOS
Coordinates relative to client-area of the window/component	Coordinates relative to top-level window
Single or double click	`-[NSEvent clickCount]`
`WM_MOUSELEAVE`, `WM_MOUSEENTERED` are opt-in events.	`-[NSResponder mouseEntered]` `-[NSResponder mouseExited]`
No distinction between `WM_MOUSEMOVE` and drag events.	`-[NSResponder mouseMoved]` `-[NSResponder mouseDragged]`

1969

Simplifying State Machines

think-cell

Windows	macOS
`SetCapture/ReleaseCapture` to receive mouse messages after mouse has exited the window. `WM_CAPTURECHANGED` `WM_CANCELMODE`	Capture is automatic
Mouse message order can be "surprising", see `QWindowsMouseHandler::translateMouseEvent`	`-[NSResponder mouseMoved]` `-[NSResponder mouseDown]` `-[NSResponder mouseDragged]` `-[NSResponder mouseUp]`

1970

Simplifying State Machines

think-cell

Windows	macOS
`SetCapture/ReleaseCapture` to receive mouse messages after mouse has exited the window. `WM_CAPTURECHANGED` `WM_CANCELMODE`	Capture is automatic
Mouse message order can be "surprising", see `QWindowsMouseHandler::translateMouseEvent`	`-[NSResponder mouseMoved]` `-[NSResponder mouseDown]` `-[NSResponder mouseDragged]` `-[NSResponder mouseUp]`

The macOS model is much saner, offers strong guarantees.

Strong guarantees are good!

1971

Simplifying State Machines

think-cell

```
// Windows message map
BEGIN_MSG_MAP(CMyWindow)
MESSAGE_HANDLER(WM_LBUTTONDOWN, OnBtnDown)
END_MSG_MAP()
```

```
// macOS Cocoa callbacks
- (void)mouseMoved:(NSEvent*)nsevent;
- (void)mouseDragged:(NSEvent*)nsevent;
- (void)mouseDown:(NSEvent*)nsevent;
```

Windows Event State Machine

```
LRESULT OnBtnDown(
  UINT nMsg, WPARAM wparam,
  LPARAM lparam, BOOL& bHandled) noexcept {
  // filter button down events
  // capture mouse if necessary
}
```

Common UI Code

```
bool CRenderWindow::OnButtonDown(EMouseButton emousebtn, point const& ptn) noexcept;
```

1972

Simplifying State Machines

think-cell

Lesson 8: Unify application states across platforms.

- Cross-platform code should satisfy identical invariants on all platforms.
- Reduces complexity or moves platform-specific complexity to platform-specific code

1973

think-cell

constexpr all the things

1974

constexpr all the things

think-cell

- Runtime code is error-prone
- Everything that can fail, will fail
- Even more so in cross-platform development

Everything that can go wrong, will go wrong, but in slightly different ways.

1975

constexpr all the things

think-cell

- Runtime code is error-prone
- Everything that can fail, will fail
- Even more so in cross-platform development

Everything that can go wrong, will go wrong, but in slightly different ways.

Lesson 9: Code that runs at compile time cannot fail at runtime.

1976

constexpr all the things

think-cell

- Internationalization is more than translation, but focus on that today
- Important translation features:
  1. Annotate translatable text in code

```
translate("Do not ask for user name/password")
```

  2. Translation context

```
translate_with_context("Proxy Authentication",
"Proxy: http://en.wikipedia.org/wiki/Proxy_server.")
```

  3. There are arbitrary number of plural forms

     0 weeks - 0 недель

     1 week - 1 неделя

     4 weeks - 4 недели

     5 weeks - 5 недель

1977

constexpr all the things

think-cell

- What is the general flow for i18n:
  1. Annotation in source code
  2. Extraction of translatable text
  3. Send to translators
  4. Get `xliff` (XML Localization Interchange File Format) file
  5. Import into project as resource
  6. At program runtime, lookup text/language pair
- Supporting native mechanisms would suck
  - You want the same markup in code
  - You want a single text extraction run, preferably platform independent
  - You want a uniform access mechanism for translatable strings, no lifetime issues (`char const*`!)
- And you want compile-time lookups

1978

constexpr all the things

think-cell

https://godbolt.org/z/eYPjhPx6G

1979

constexpr all the things

think-cell

https://godbolt.org/z/eYPjhPx6G

Lesson 9: Code that runs at compile time cannot fail at runtime.

- Uniform translation lookup across platforms
- Possibly increased compile-time
- Eliminated runtime error-handling

1980

think-cell

Build Your Own Tools

1981

Build Your Own Tools

think-cell

- think-cell ships with Google Chrome extension and web app
- What language to use?

1982

Build Your Own Tools

think-cell

- think-cell ships with Google Chrome extension and web app
- What language to use?

1. No JavaScript
2. TypeScript better but no code sharing with C++
3. Emscripten looked interesting
   - WebAssembly is standardized
   - Convenient way to deliver cross-platform application
   - Interfacing with JavaScript loses type-safety again
   - Lacking wrappers for standard APIs

```
auto xhr = emscripten::val::global("XMLHttpRequest").new_();
xhr["open"]("GET", "http://google.com");
```

1983

Build Your Own Tools

think-cell

- think-cell ships with Google Chrome extension and web app
- What language to use?

1. No JavaScript
2. TypeScript better but no code sharing with C++
3. Emscripten looked interesting
   - WebAssembly is standardized
   - Convenient way to deliver cross-platform application
   - Interfacing with JavaScript loses type-safety again
   - Lacking wrappers for standard APIs

```
auto xhr = emscripten::val::global("XMLHttpRequest").new_();
xhr["open"]("GET", "http://google.com");
```

... so we build our own "compiler"

1984

Build Your Own Tools

think-cell

Type definition libraries

```
interface Document extends Node, NonElementParentNode, DocumentOrShadowRoot {
    readonly URL: string;

    readonly activeElement: Element | null;

    readonly anchors: HTMLCollectionOf<HTMLAnchorElement>;

    title: string;

    createElement<K extends keyof HTMLElementTagNameMap>(
        tagName: K, options?: ElementCreationOptions
    ): HTMLElementTagNameMap[K];
}
```

https://github.com/DefinitelyTyped/DefinitelyTyped

Available for over 7000 JavaScript libraries, including standard DOM APIs

1985

Build Your Own Tools

think-cell

Powerful TypeScript compiler API:

```
function transform(file: string) : void {
    let program = ts.createProgram([file]);
    const sourceFile = program.getSourceFile(file);

    ts.forEachChild(sourceFile, node => {
        if (ts.isFunctionDeclaration(node)) {
            // do something
        } else if (ts.isVariableStatement(node)) {
            // do something else
        }
    });
}
```

1986

Build Your Own Tools

think-cell

typescripten — https://github.com/think-cell/typescripten

- Compiles typescript interface declarations to C++ interfaces
- **i.e. type-safe, ideomatic calls to JavaScript libraries via emscripten**

JavaScript:

```
    document.title = "Hello World from C++";
```

C++:

```
    using namespace tc;
    js::document()->title(js::string("Hello World from C++!"));
```

1987

Build Your Own Tools

think-cell

```
namespace js {
    struct JsObject {
        emscripten::val m_emval;
    };

    struct Document : virtual Node, ... {
        auto URL() noexcept;
        auto activeElement() noexcept;
        auto title() noexcept;
        void title(string v) noexcept;
        // ...
    };

    inline auto Document::title() noexcept { return m_emval["title"].template as<string>(); }
    inline void Document::title(string v) noexcept { m_emval.set("title", v); }

    inline auto document() noexcept {
        return emscripten::val::global("document").template as<Document>();
    }
}
```

1988

Build Your Own Tools

think-cell

typescripten is self-hosting, i.e., can compile typescript compiler interface definition. Compiles standard JS libraries.

```
void transform(js::string const& file)  {
    js::Array<js::string> arr(jst::create_js_object);
    arr->push(file);

    auto const program = js::ts::createProgram(arr, ...);
    auto const sourceFile = program->getSourceFile(file);

    js::ts::forEachChild(sourceFile,
        js::lambda(
            [](js::ts::Node jnodeChild) noexcept -> js::unknown {
                if(js::ts::isFunctionDeclaration(jnodeChild)) {
                }
            }
        )
    );
}
```

1989

Build Your Own Tools

think-cell

Declaration order does not matter in typescript

```
type FooBar = test.Foo | test.Bar;

declare namespace test {
    export interface Foo {
        a: string;
    }

    export interface Bar {
        b: number;
    }
}
```

1990

Build Your Own Tools

think-cell

String enums with custom marshaling

```
export enum FunnyEnum {
    foo = "foo",
    bar = "bar"
}
```

1991

Build Your Own Tools

think-cell

String enums with custom marshaling

```
export enum FunnyEnum {
    foo = "foo",
    bar = "bar"
}
```

```
enum class FunnyEnum { foo, bar };

template<> struct MarshalEnum<FunnyEnum> {
    static inline auto const& Values() {
        static tc::unordered_map<FunnyEnum, unknown> vals{
            {FunnyEnum::foo, string("foo")},
            {FunnyEnum::bar, string("bar")}
        };
        return vals;
    }
};
```

1992

Build Your Own Tools

think-cell

Function-callbacks as ref-counted objects

This is what we want:

```
document.onclick(OnClickHandler)
```

This is what we have:

```
struct S {
    JS_MEMBER_FUNCTION(S, OnClick, void, (js::MouseEvent ev)) {
        js::document()->title(js::string("Hello World from C++!"));
    }

    void AddHandler() {
        js::document()->onclick(OnClick);
    }

    ~S() {
        // OnClick is destroyed, disconnects automatically
    }
};
```

1993

Build Your Own Tools

think-cell

Generics

```
interface HTMLCollectionOf<T extends Element> extends HTMLCollectionBase {
    item(index: number): T | null;
}
```

Indexed access types

```
interface DocumentEventMap  {
    "click": MouseEvent;
    "keydown": KeyboardEvent;
}
addEventListener<K extends keyof DocumentEventMap>(
    type: K, listener: (this: Document, ev: DocumentEventMap[K]) => any, ...
): void;
```

1994

Build Your Own Tools

think-cell

typescripten — https://github.com/think-cell/typescripten

1995

Thank you!

And yes, we are recruiting: `hr@think-cell.com`



1996

CPP-Summit 2022



袁鹏

腾讯专家工程师

拥有十余年编译器/分布式系统/AI计算框架经验，熟悉ARM64和NPU芯片架构，在GCC/Open64开源编译社区贡献多个关键模块，是GCC-FDO优化的核心设计者之一；曾担任自研AI计算框架架构师，经历从0到1诞生的整个架构设计过程。目前关注编译基础设施和AI大规模分布式计算。

主办方：Boolan 高端IT咨询与教育平台

AI for Compiler 探讨

yuanpeng

本次讲座目标

- 了解编译器中一些优化策略的思路
- 系统软件引入AI方法的动机和问题
- 希望这些思路能够启发大家的工作







GCC Initial release: 1987 LOC: 15 million	C++ The first edition: 1985 C++23 / 17 March 2022	LLVM Initial release: 2003 LOC: ~5 million

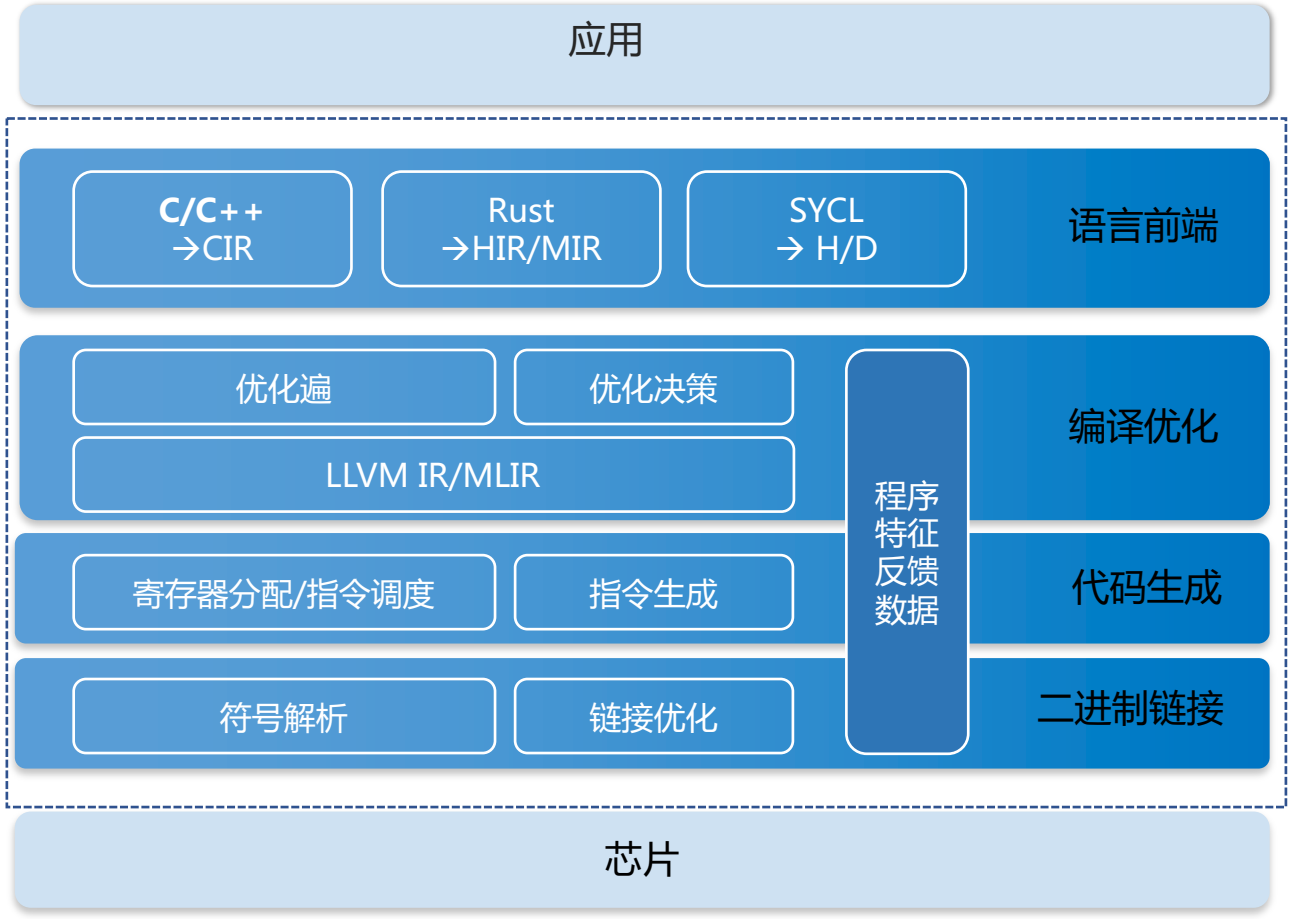
目录

01 编译器介绍

02 AI for Compiler 进展

03 一些案例

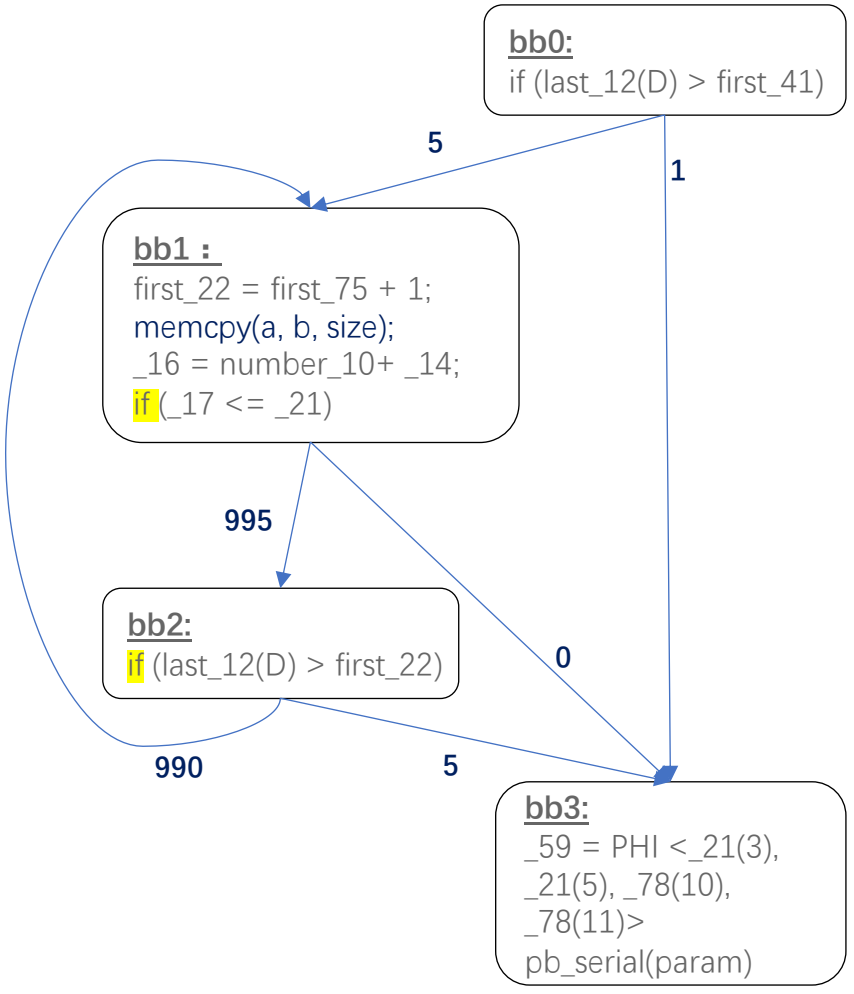
编译器的基本架构



- 前端的趋势是每个语言设计自己的IR以及优化，解决语言特定问题。比如C++引入CIR解决coroutine安全。
- 中端是在IR上进行多种优化，比如常量传播，冗余消除，循环优化等，上百种。
- 后端与芯片体系结构相关的优化，比如指令调度，寄存器分配，指令选择等。
- 链接创造了全视野的机会，可以进行跨函数跨文件乃至二进制级别优化，比如LLVM Bolt。

```
$g++ -O2 quick.cpp  -fdump-tree-all
quick.cpp.004t.original quick.cpp.005t.gimple
quick.cpp.044t.profile_estimate quick.cpp.107t.dce2 ··· quick.cpp.234t.nrv
```

以反馈优化为例：数据驱动的优化



编译器反馈优化可以显著提升一些程序的性能。

GCC –fprofile-generate/-fprofile-use

LLVM -fprofile-instr-generate/-fprofile-instr-use

编译器中的程序反馈数据包括：

① 跳转边执行频率：决定函数中代码热点，变量热点。这些数据有效指导如下优化，原则是“资源优先分配给高频执行的路径”。

- 寄存器分配：高频变量优先分配寄存器。
- Cache感知的代码布局：高频执行路径代码顺序排布，增加ICache局部性，比如bb0->bb1->bb2->bb3。
- 内联：调用频繁的函数优先考虑内联。
- 循环展开：频繁执行的循环考虑展开。

② 关键变量值的变化范围，比如memcpy中size参数。

- 用于代码特化，比如size=8高频发生时，调用高性能memcpy_8函数。

静态估计策略

没有准确的运行时数据，基于规则的静态猜测。

- Branch Prediction for Free; Ball and Larus; PLDI ‘93.
- Static Branch Frequency and Program Profile Analysis; Wu and Larus; MICRO-27.

Heuristics	Probability of taking branch
Loop branch (LBH)	88 %
Pointer (PH)	60 %
Call (CH)	78 %
Opcode (OH)	84 %
Loop exit (LEH)	80 %
Return (RH)	72 %
Store (SH)	55 %
Loop header (LHH)	75 %
Guard (GH)	62 %

- Ph: 指针和null/指针比较
- OH：<0 <=0 =constant
- CH：跳转到函数调用且不后支配。

GCC的规则描述：gcc/predict.def

#define likely(x) __builtin_expect(!!(x), 1)

- "hot label", HITRATE (90)

/* Branch ending with return negative constant is probably not taken. */

- DEF_PREDICTOR (PRED_NEGATIVE_RETURN, "negative return", HITRATE (98), 0)

```
$ g++ -O2 quick.cpp -fdump-tree-profile_estimate

Predictions for bb 3
  DS theory heuristics: 20.24%
  combined heuristics: 20.24%
  opcode values nonequal (on trees) heuristics of
edge 3->4: 34.00%
  call heuristics of edge 3->4: 33.00%

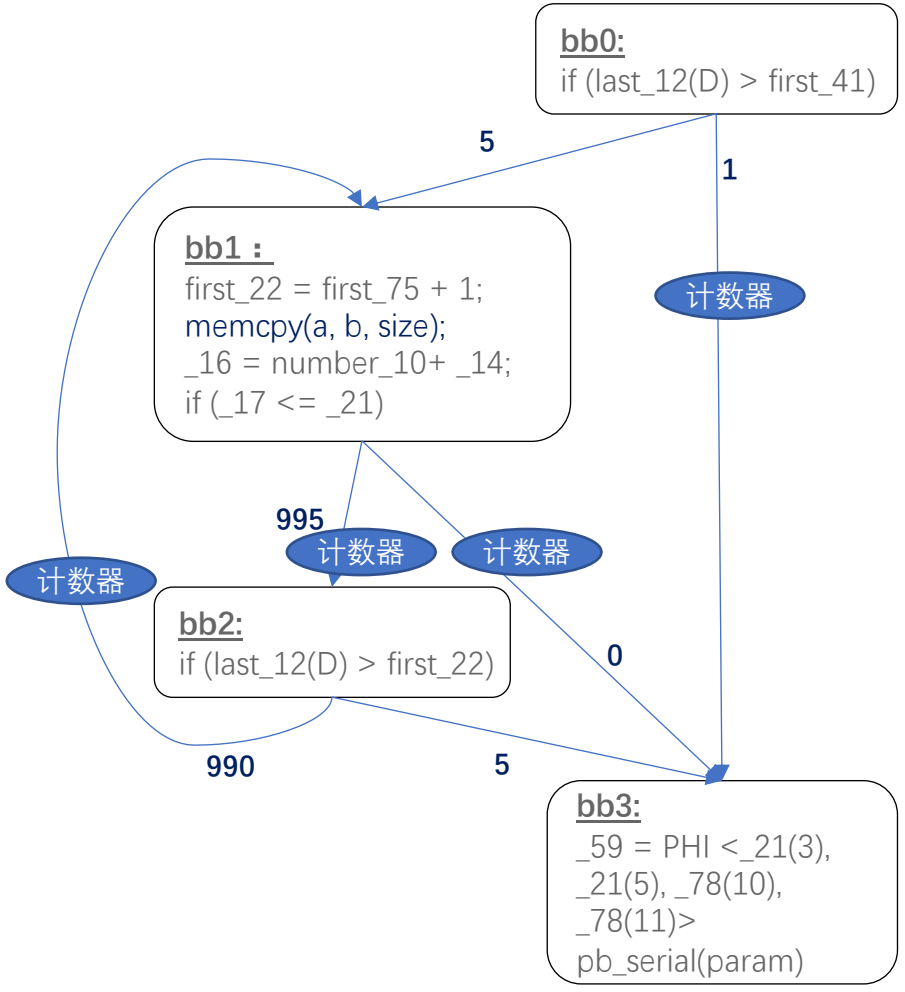
<bb 3> [local count: 365072224]:
  if (__priority_4(D) == 65535)
    goto <bb 4>; [20.24%]
  else
    goto <bb 5>; [79.76%]

  <bb 4> [local count: 73890618]:
  std::ios_base::Init::Init (&__ioinit);
  __cxa_atexit (__dt_comp , &__ioinit, &__dso_handle);
```

准确率 60-70%

2003

运行插桩策略




```
$ g++ -O2 -fprofile-generate quick.cpp
./a.out
$ ls
a.out  quick.cpp  quick.cpp.044t.profile_estimate
quick.gcda
$ gcov-dump -l quick.gcda
3:FUNCTION ident=681322217,
lineno_checksum=0x1ac96cde
8:COUNTERS arcs 5 counts
0: 1 5 995  990 5

$ g++ -O2 -fprofile-use quick.cpp
```

插桩运行：

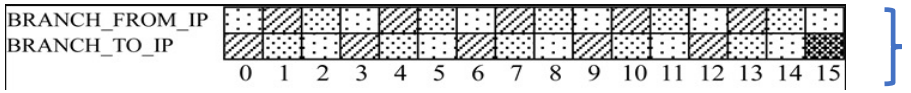
- 在程序的边上插入计数器代码，通过典型输入运行程序获得执行次数。
- 开销大：2-10倍降低，适合benchmark或小程序，但无法用于在线业务。
- 准确率：100%。

PMU取样策略（1）：低开销获得数据

```
40066e:  mov    (%r8),%esi
400671:   cmp    %edx,%ebp
400673:   jle    400694 <quicksort+0x54>
400675:   lea    0x4(%r12,%rax,1),%rax
40067a:   jmp    400690 <quicksort+0x50>
40067c:   nopl   0x0(%rax)
400680:   add    $0x1,%edx
...
400688:   add    $0x4,%rax
40068c:    cmp    %edx,%ebp
40068e:    jle    400694 <quicksort+0x54>
400690:   cmp    %ecx,%esi
400692:    jle    400680 <quicksort+0x40>
400694:    movslq %ebx,%rax
```

运行中的二进制指令

Perf



LBR (Last Branch Record) 数据：

PMU 寄存器中记录应用中最近N次branch执行的地址对：FROM源地址–>TO跳转地址。

{0x400673–>0x400694,
0x40067a–>0x400690,
0x40068e–>0x400694, …}

Perf裸数据中对应多次采样的地址对。跳转被采样多，意味着这个跳转发生“频繁”。

编译器在二进制中记录指令地址和源码行对应关系：

0x40066e –>{quicksort, 2}

0x400671 –>{quicksort, 3}

0x 400673–>{quicksort, 3}

…

数据处理：

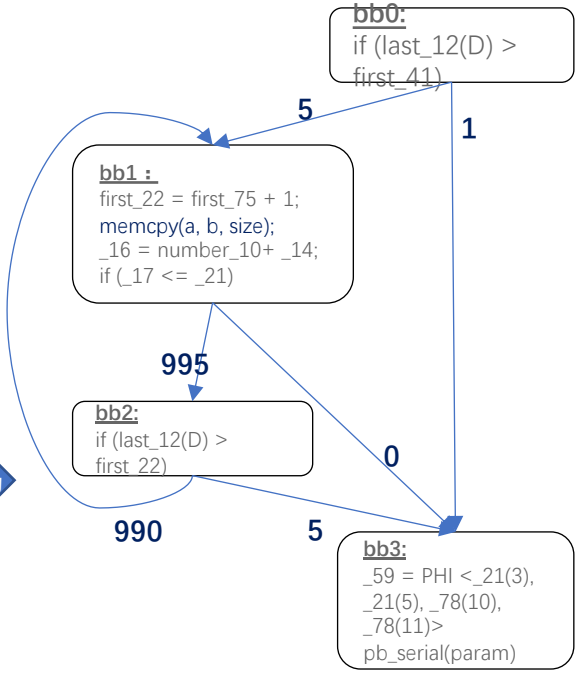
多次的LBR Event合并，综合成每个跳转的采样总次数，并映射回源码。。

0x400673–>0x400694: 995 {quicksort, 3}

0x40067a–>0x400690：990:{quicksort, 20}

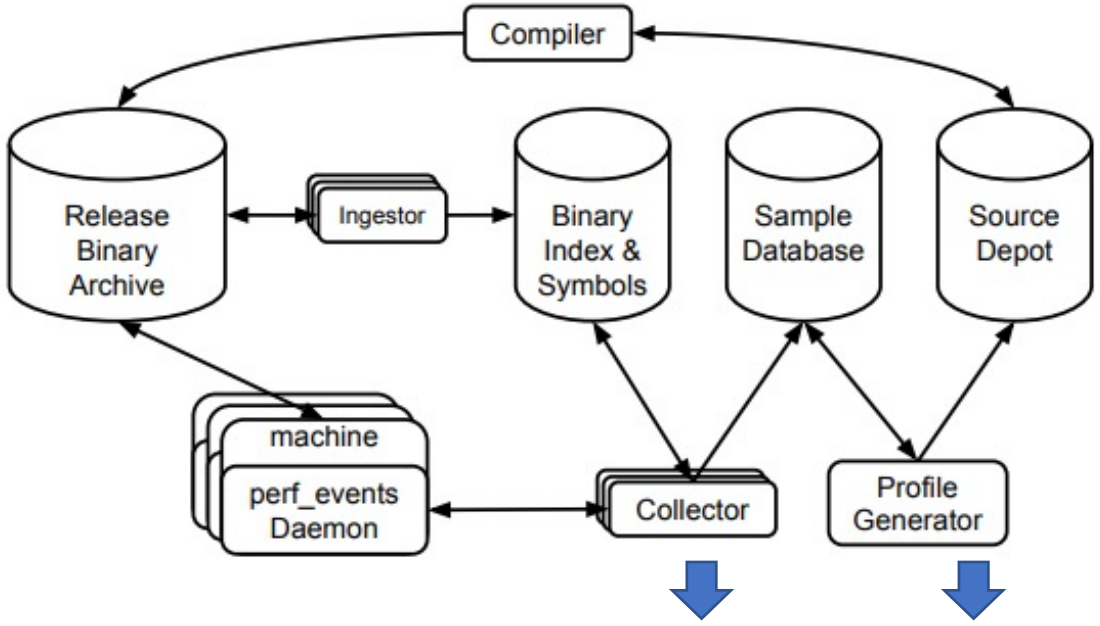
0x40068e–>0x400694:5: {quicksort, 23}

标记到源码



2005

PMU取样策略（2）：部署，以autoFDO为例



服务器部署收集器，定期采集C++应用的perf数据，并保存到数据库。

为每个代码仓生成关联代码的采样数据。

Application	FDO	AutoFDO	Ratio
server	17.61%	15.89%	90.23%
graph1	14.68%	14.04%	95.65%
graph2	7.16%	6.27%	87.50%
machine learning1	8.92%	8.46%	94.85%
machine learning2	7.09%	6.60%	93.06%
encoder	8.63%	3.31%	38.37%
protobuf	16.96%	14.40%	84.94%
artificial intelligence1	10.12%	10.12%	100.00%
artificial intelligence2	13.24%	11.33%	85.61%
data mining	20.48%	15.54%	75.86%
mean	12.40%	10.52%	84.84%

性能接近运行插桩（FDO）。

Google/Meta主要以此种方法为主优化C++应用。

实际应用问题：为数据而战->为行为而战

静态估计	• 编译器O2默认开启。
运行插桩	• SPEC程序比分用：SPEC2017，SPEC 2006. • 能抽象代表性输入的应用可以用，比如数据库。 • 一些AI编译器也用，比如PyTorch TorchScript。
PMU取样	• 一般用于在线应用， • *局限：AMD处理器高版本kernel支持LBR，VM中没有打开PMU选项。*

AutoFDO (2016): 应用PMU数据的持续反馈优化	ThinLTO （2017）： 可扩展及增量链接优化	Propeller （2020）： 二进制链接优化	MLGO (2022): AI For Compiler，基于GWP数据预测新程序运行特征。

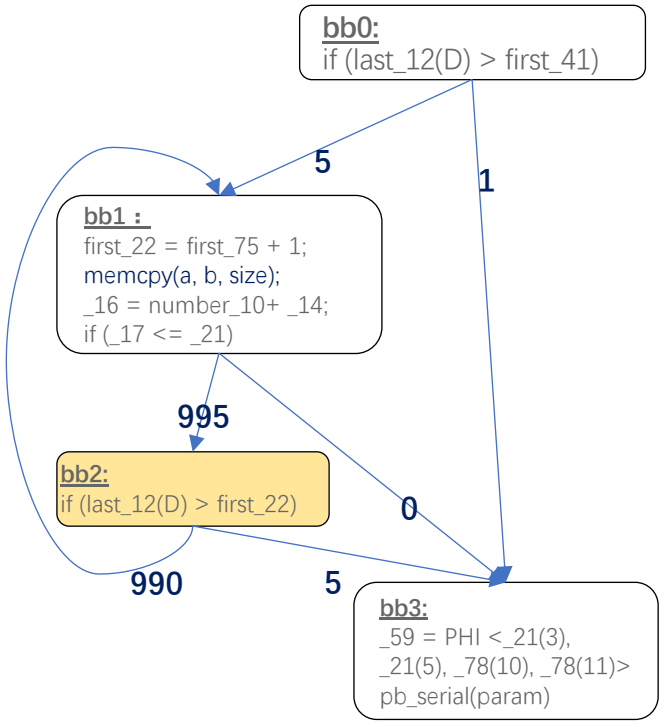
目录

01 编译器介绍

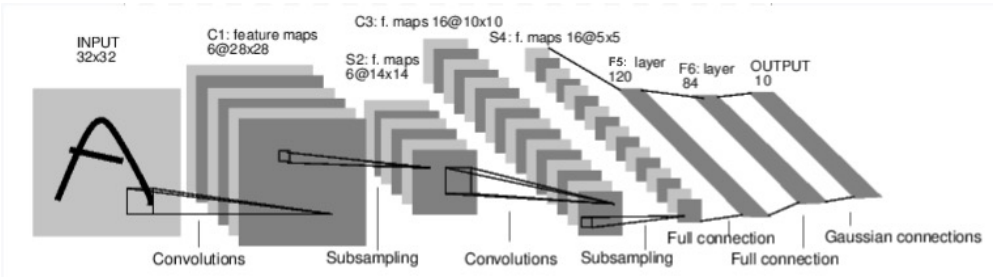
02 AI for Compiler 行业进展

03 一些案例

用深度学习方法预测边概率



输入是32x32像素矩阵，
值范围[0,255]。



手写数字识别LeNet

https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

输出是10概率矩阵，
值范围[0,1]。

0:0
1:0
.....
6: 0.31
7: 0.69
...

- 数字特征：循环深度，循环中基本块数，退出边数。
  [1, 2, 3]
- 类别特征：opcode，边的属性（backedge，inner）。
  [1, 1]
- 控制流特征：目标基本块中的函数。
  [“memcpy”，“pb_serial”]
- 函数特征：指令IR数，边的个数，基本块个数。
  [9, 6, 4]

NN

- *概率*

- 在一个应用上大约80%。
- Google：67%正确，相比静态提升25-50%。 @40 benchmark。

关键问题：

- 抽象问题域，定义输入和输出；
- 选取数据集；
- 选取合适的模型尝试。

注意：

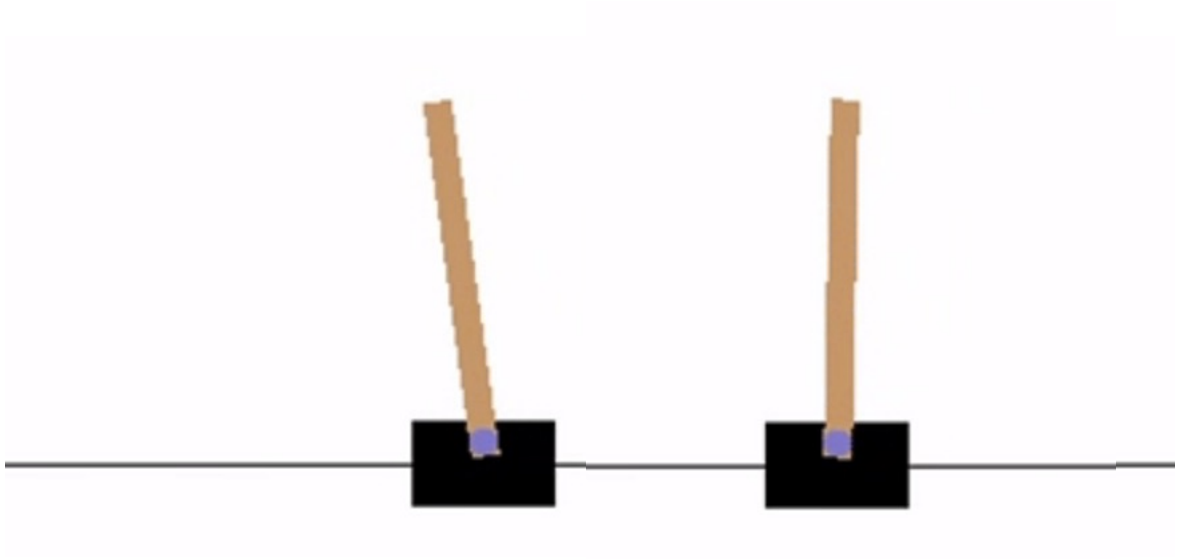
- 跳转上下文关联：更好的模型选择来隐含关联，比如attention。
- 数据集的泛化：一个应用的代码，一个部门的代码，全公司的代码。

能否构建一个AI for Compiler的平台用于验证？

- 程序特征的灵活抽取：预置业界程序分析的最新成果。
- 结果数据评估方便：branch正确率，执行时间，代码大小。
- 训练数据benchmark的多样性。

还有一些问题难以获得训练数据集，比如性能。。。

强化学习介绍：倒立摆Cartpole游戏



Cartpole在重力作用下会倒下，游戏目的是通过左右控制小车（施加左右的力）来避免杆的倒下。

状态：

- 小车在轨道上的位置
- 杆子与竖直方向的夹角
- 小车速度
- 角度变化率

输出：

一系列的动作（方向+作用力），是一个持续的过程。

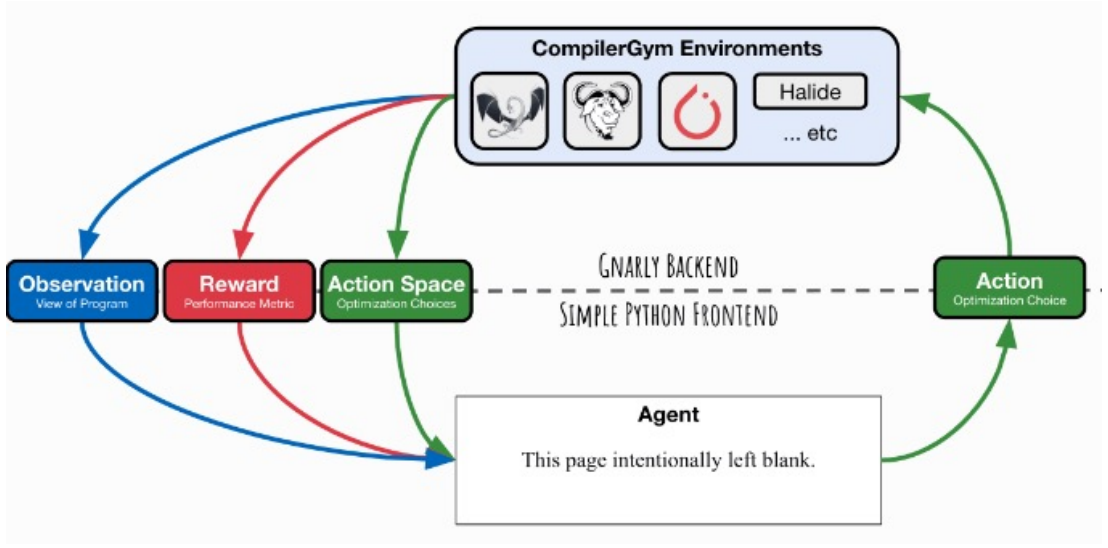
具体参考：

REINFORCEMENT LEARNING (DQN) TUTORIAL

https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html

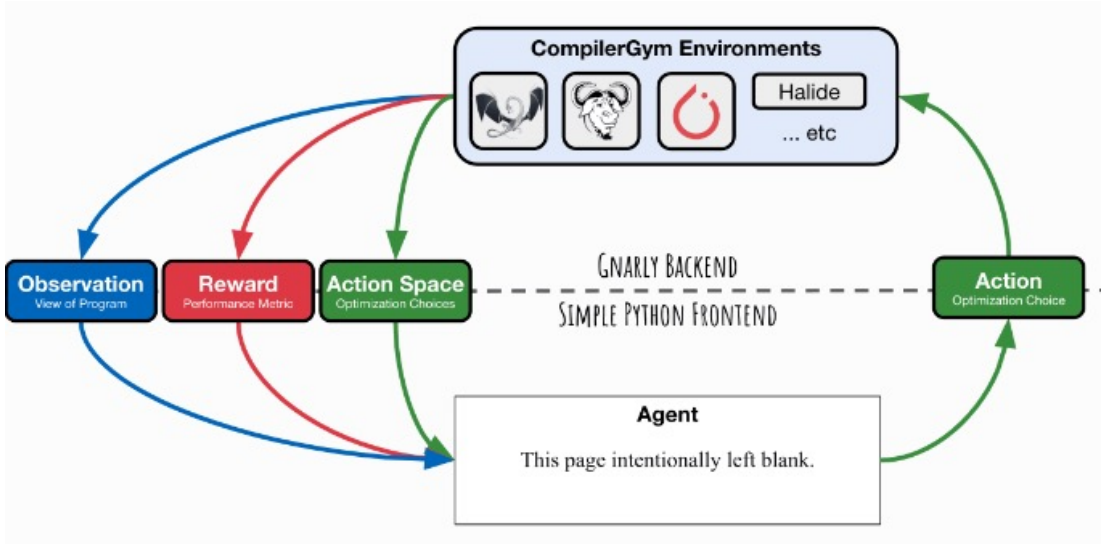
CompilerGym：基于强化学习的编译优化基础设施

Build the “ImageNet for compilers”: a high-quality, open-source OpenAI gym environment for experimenting with compiler optimizations using real-world software and datasets and get it into the hands of AI researchers.



- 与多种编译器集成：GCC, LLVM, Halide。
- 大量benchmark数据集：anghabench，blas，opencv等共一百多万程序，特征分化（访存，跳转，计算）。
- 内置丰富的程序特征(Observation space)：InstCount, Autophase。
- 多种策略空间：优化选项组合。
- 多种目标代价模型：执行时间，代码大小。

CompilerGym示例



Autophase特征是一个56维特征向量，刻画程序的IR特征。比如：

15 BranchCount Number of branches
18 NumEdges Number of edges
32 NumBrInst Number of Br instructions
37 NumLoadInst Number of Load instructions

```
//基于LLVM编译器构建一个验证环境，使用autophase特征，目标代价为优化代码大小。
env = gym.make("llvm-autophase-ic-v0")

//基于qsort程序验证。
env.reset(benchmark="benchmark://qsort")

//执行一个action编译：action就是一个优化选项，比如-O2。
observation, reward, done, info = env.step(0)

// 获得程序特征autophase
>>> observation
array([ 0, 0, 26, 25, 1, 26, 10, 1, 8, 10, 0, 0, 0, 37, 0, 36, 0, 2,
46, 175, 1664, 1212, 263, 26, 193, 0, 59, 6, 0, 3, 32, 0, 36,
10, 1058, 10, 0, 840, 0, 0, 0, 1, 416, 0, 0, 148, 60, 0, 0, 0, 37,
3008, 2062, 9, 0, 1262])

// 从代码大小维度评估本次action效果
>>> reward
0.3151595744680851
```

从优化的视角看，是要寻找一个针对特征的“好模型”，给出action，确保reward好。

MLGO：基于机器学习的编译优化框架

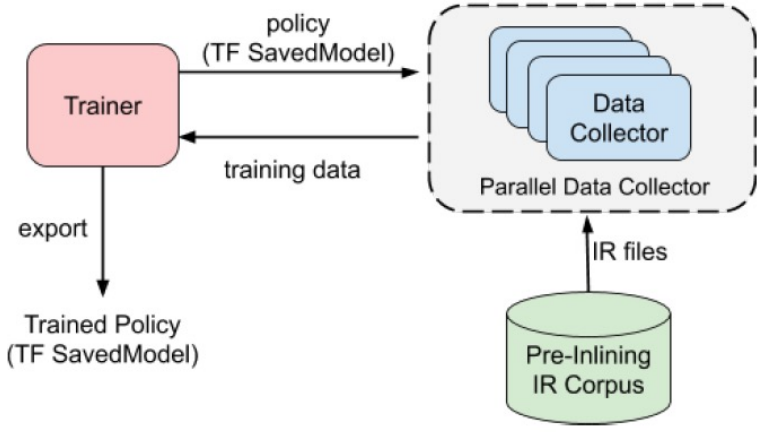
https://github.com/google/ml-compiler-opt

强化学习的方法：
- Deep Q–Network
- Policy Gradient
- Evolution Strategies

对已有编译优化的重构：
- [llvm-dev] RFC: a practical mechanism for applying Machine Learning for optimization policies in LLVM *up to 7% size reduction*
- [llvm-dev] [RFC] MLGO Regalloc: learned eviction policy for regalloc

Type	Feature
caller feature	caller_basic_block_count caller_conditionally_executed_blocks caller_users
callee feature	callee_basic_block_count callee_conditionally_executed_blocks callee_users
call site feature	callsite_height cost_estimate number_constant_params
call graph feature	edge_count node_count

解决inline大小问题的特征



	PG	ES	ES (L)
Size Reduction	4.95%	3.74%	5.94%
Parallelism in Data Collection	100	488	488
Training Time	~12h	~60h	~150h

2014

目录

01 编译器介绍

02 AI for Compiler 行业进展

03 一些案例

利用编译器已有优化改进二进制大小：基于policy_critic_network

问题定义

- 给定一个程序，clang –fx –fy –fz … big.cpp，使得生成的代码二进制“小”。

问题映射

- 训练一个模型，输入是程序特征（state），选择优化选项（action），输出基于大小的reward。
- 强化学习方法（policy critic）

问题实现

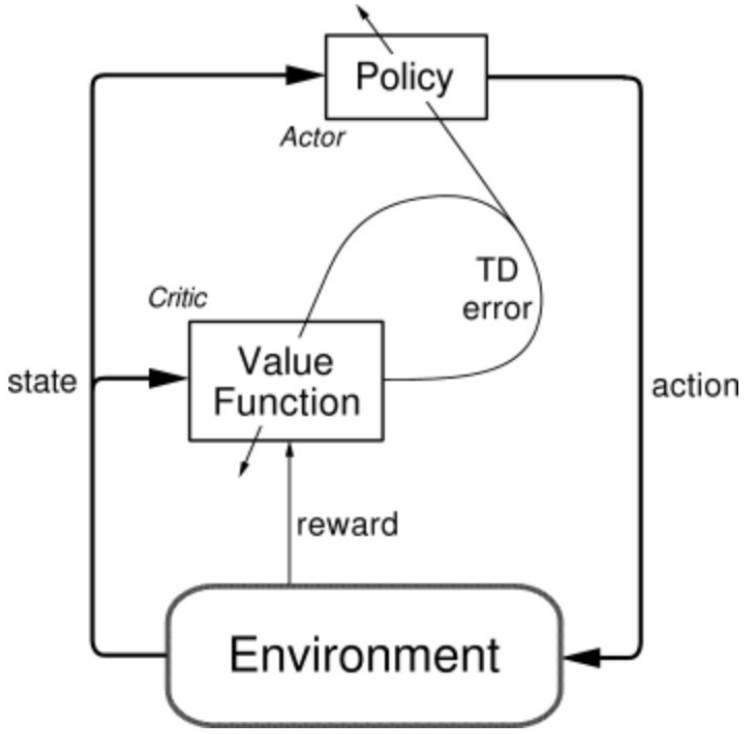
- 选择benchmark
- *模型的训练-验证-调试*

big.cpp

0: -O2->run
1: state->network->-fx
2: “-O2 –fx” -> run
3: state->network->-fy
4: “-O2 –fx -fy” -> run
……

设定步数阈值，最终得到一个选项组合。

Policy和Value两张网络训练模型




```
env = llvm_wrapper(benchmarks, max_episode_steps=200,
steps_in_observation=True)
ppo_training = PPO(env, name="autophase1", EPOCHS=80)
    action, logprob = self.actor_critic.act(obs)
    logprobs, state_values, dist_entropies =
self.actor_critic.evaluate(sampled_states, sampled_actions)
    self.optimizer.zero_grad()
    loss.mean().backward()
    self.optimizer.step()
```

网络选择：简单MLP。
特征：autophase。

在未知应用上，可以获得~6%的下降。

```
self.input_dim = nom_observations
self.output_dim = nom_actions

self.features = nn.Sequential(
    nn.Linear(nom_observations, 128),
    nn.Tanh(),
    nn.Linear(128, 128),
    nn.Tanh(),
    nn.Linear(128, 128),
    nn.Tanh()
)

self.action_lin = nn.Linear(128, nom_actions)
self.v_lin = nn.Linear(128, 1)
```

尚待解决的问题

- 训练时间长，特别是性能目标。强化学习平台需要分布式。
- 数据集：无法适应在线应用的训练。开源软件是否有助于改进特征数据。
- 特征选择难，业界还没有在此有一致的方法。

->**瞄准问题，定义问题，不要为AI而AI，多种方法不冲突。**

感谢2022全球C++及系统软件技术大会！

Q & A

CPP-Summit 2022

刘四维



阅文集团智能业务中心后端技术负责人

曾任职百度架构师，做后端开发8年时间，目前就职于阅文集团智能业务中心，任后端技术负责人。有10年以上的C++开发经验，熟悉后端系统的在线，离线架构，有丰富的搭建推荐和搜索系统的经验。了解各类NoSQL，尤其是对Redis非常熟悉。开源的Redis C++客户端redis-plus-plus已经成为Redis官方推荐的C++客户端，开源的redis-protobuf帮助用户高效实现Redis存储复杂结构体。

主办方：Boolan 高端IT咨询与教育平台

2020

CPP-Summit

当C++遇见Redis





刘四维

阅文智能中心工程负责人

me::about

```
constexpr auto me::about() const noexcept {

    auto name = "刘四维";

    auto experience = { "阅文", "百度", "NI" };

    auto skills = { "C++", "推荐搜索后端", "NoSQL" };

    auto pages = {
            "https://github.com/sewenew",
            "https://stackoverflow.com/users/5384363/for-stack"
    };
}
```

议程

1. **Redis简介**
   In-memory data structure store
2. **redis-plus-plus**
   Redis C++客户端
3. **redis-protobuf**
   Redis as an object store

01

Redis简介

Redis简介



In-memory

data structure

store

Redis简介

缓存

string, hash

集合

set, sorted set

hash, bitmap

计数

integer, double

set, hyperloglog



扩展

Lua scripting

Redis Module

Message Broker

queue (list), pubsub

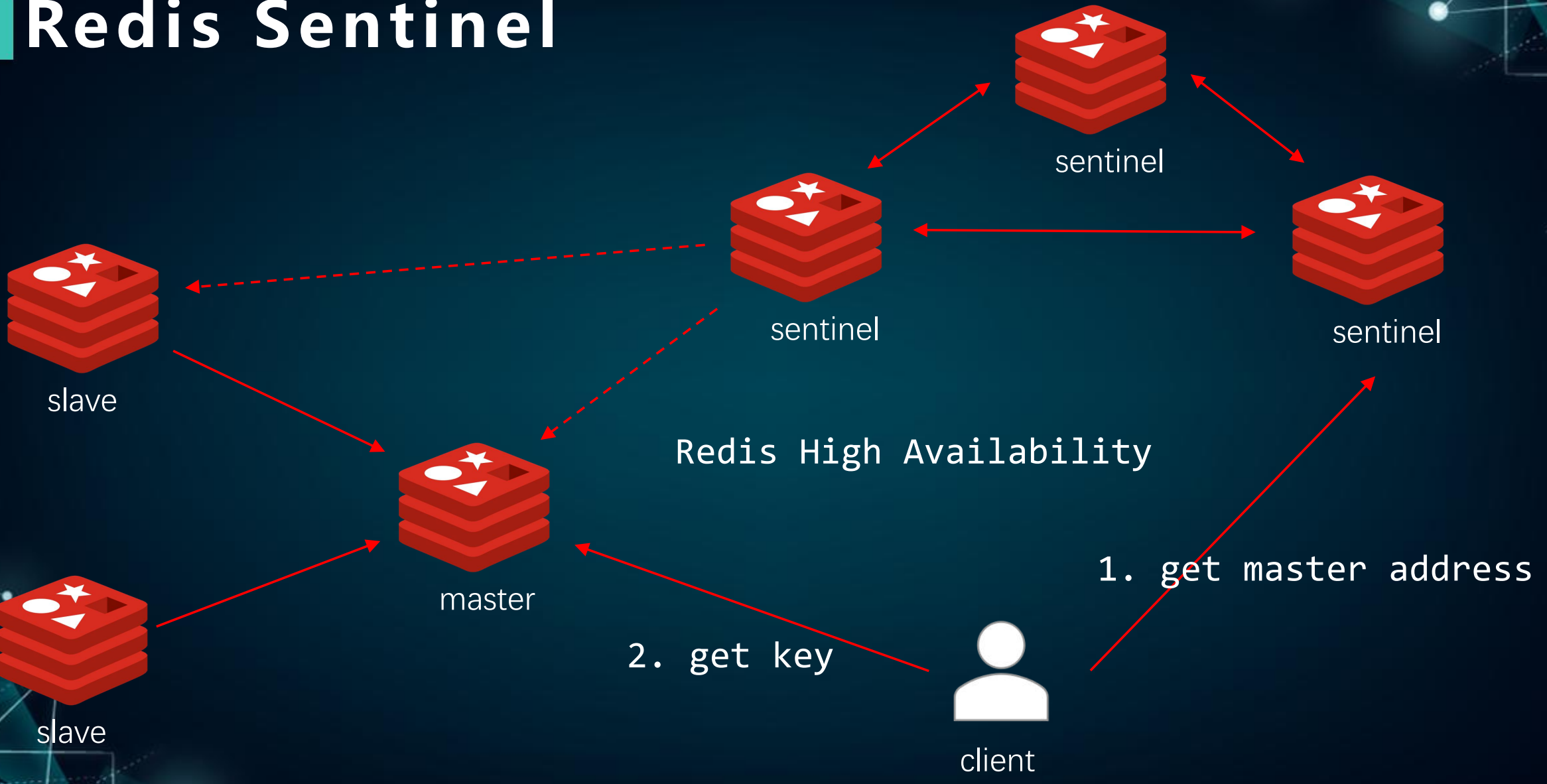
Redis Stream

多模

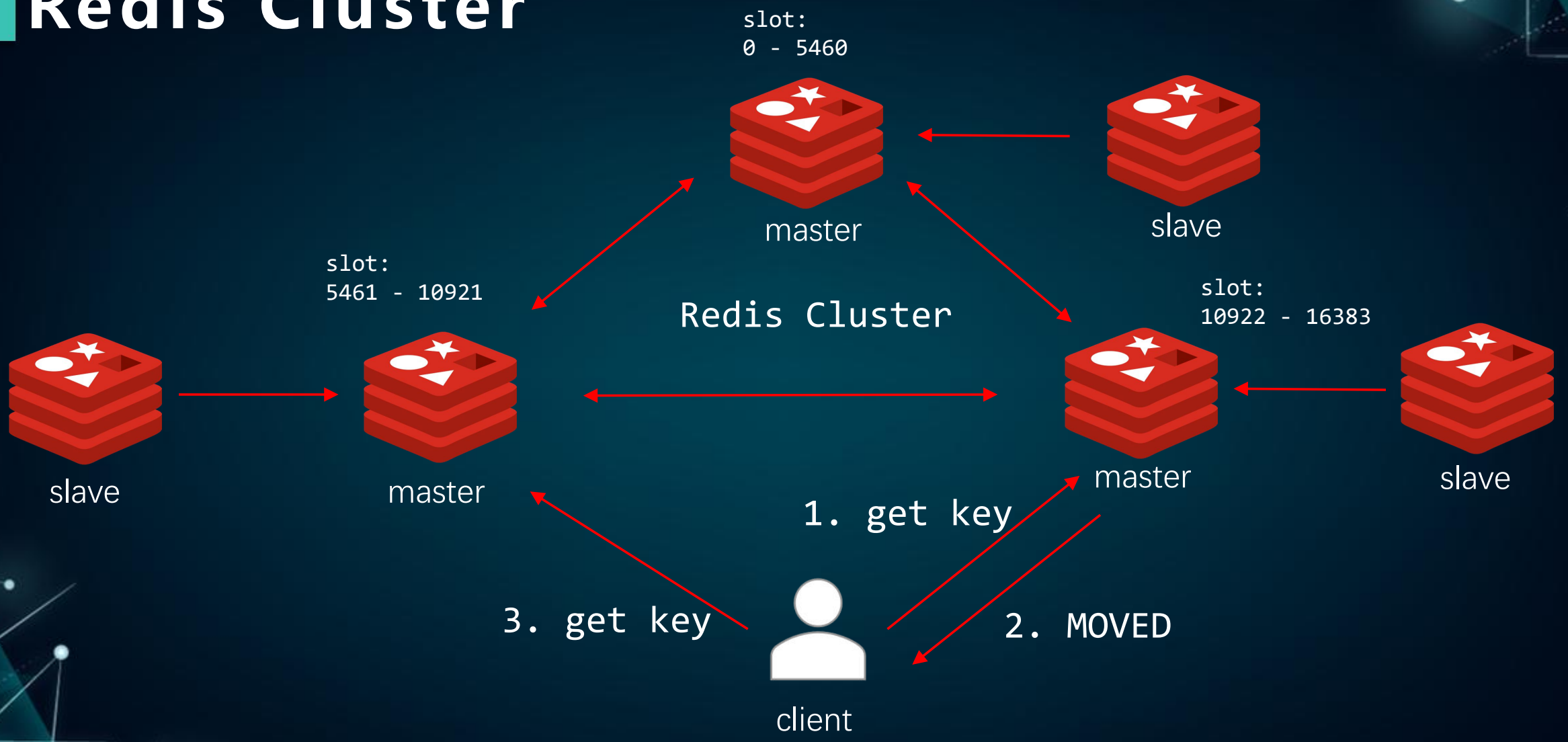
spatial, time series, graph

search engine, document

Redis Sentinel



Redis Cluster



Redis简介



In-memory

数据受限于内存



单线程

无法最大化利用多核优势



集群

Redis Cluster发布较晚



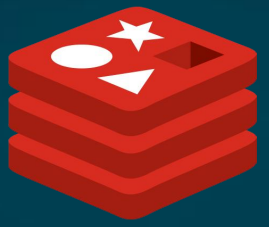
主从异步同步

数据可能丢失

Redis Protocol





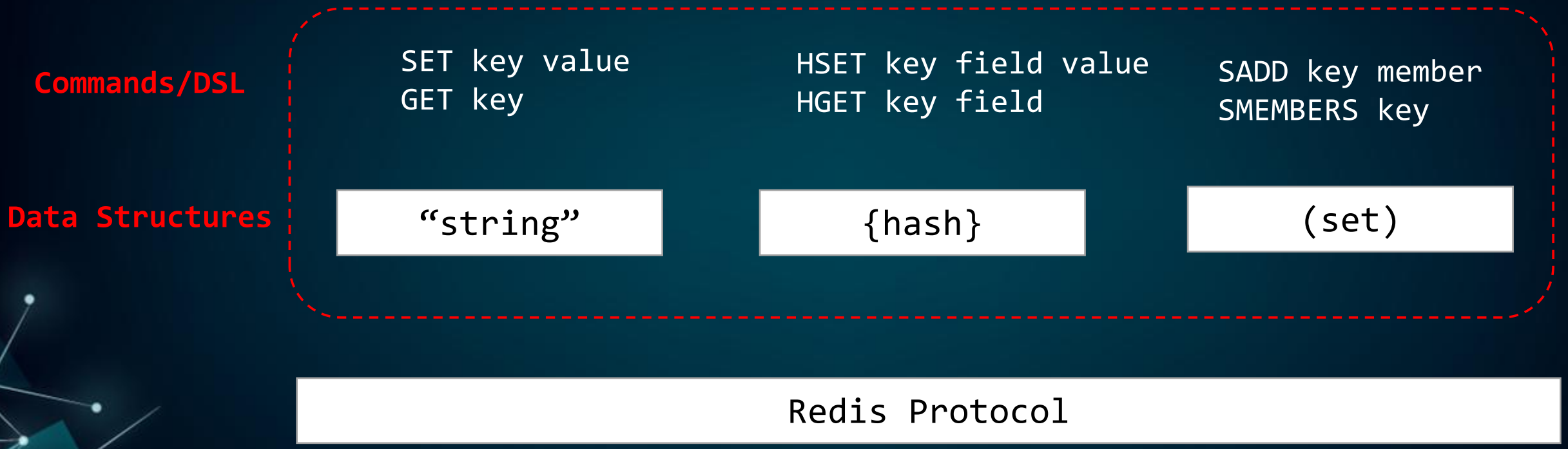


Redis Protocol





Redis Protocol



Redis Protocol

RESP2: REdis Serialization Protocol V2

```
127.0.0.1:6379> SET key value
OK
```

Simple String Reply

```
127.0.0.1:6379> LPUSH list a b c
(integer) 4
```

Integer Reply

```
127.0.0.1:6379> GET key
"value"
```

Bulk String Reply

```
127.0.0.1:6379> SMEMBERS NoSql
1)  "Redis"
2) "MongoDB"
```

Array Reply

Redis Protocol

```
127.0.0.1:6379> HGETALL langs
1) "Redis"
2) "C"
3) "MongoDB"
4) "C++"
5) "ElasticSearch"
6) "Java"
7) "Cassandra"   <-- field
8) "Java"        <-- value
```

Array Reply

```
127.0.0.1:6379> MEMORY STATS
1) "peak.allocated"
2) (integer) 1135312
……
17) "db.0"
18) 1) "overhead.hashtable.main"
       2) (integer) 152
       3) "overhead.hashtable.expires"
       4) (integer) 0
……
45) "rss-overhead.ratio"
46) "1.0348072052001953"
```

Nested Array Reply

Redis Protocol

"string"

123

3.1415926

{hash}

[list]

(set)

<sorted set>

{nested data structure}

MISMATCH

Simple String

Bulk String

Integer

Array

Redis In-memory Data Structures

Redis Protocol

Redis Protocol

"string"

123

3.1415926

{hash}

[list]

(set)

<sorted set>

{nested data structure}

MISMATCH

Simple String

Bulk String

Integer

Array

Null

Double

Boolean

map

set

Redis In-memory Data Structures

Redis Protocol V3

Redis Protocol

"string"

123

3.1415926

{hash}

[list]

(set)

<sorted set>

{nested data structure}

MISMATCH

Simple String

Bulk String

Integer

Array

Null

Double

Boolean

map

set

MISMATCH

std::string

std::vector

std::list

std::unordered_map

std::map

std::set

std::unordered_set

std::tuple

{nested data structure}

Redis In-memory Data Structures

Redis Protocol V3

C++ In-memory Data Structures

Redis Protocol

"string"

123

3.1415926

{hash}

[list]

(set)

<sorted set>

{nested data structure}

Redis Client

MATCH

std::string

std::vector

std::list

std::unordered_map

std::map

std::set

std::unordered_set

std::tuple

{nested data structure}

Redis In-memory Data Structures

C++ In-memory Data Structures

02

redis-plus-plus

Redis客户端

client	语言	易用性 Match?	最近更新	Redis Cluster	Redis Sentinel	连接池	线程安全	Stars
hiredis	C	NO	及时	NO	NO	NO	NO	5k+
hiredis-vip	C	NO	5年前	YES	NO	NO	NO	~300
cpp_redis	C++	NO	Archived 3年前	NO	YES	NO	YES	900+
cpp_redis(*)	C++	NO	2021年以来2次更新	NO	YES	NO	YES	400+

hiredis

```
127.0.0.1:6379> SMEMBERS NoSQL
1) "Redis"
2) "Mongodb"
3) "ElasticSearch"
4) "Cassandra"
```

hiredis

```
auto ctx = std::unique_ptr<redisContext, decltype(&redisFree)>(
                          redisConnect("127.0.0.1", 6379),  // create connection
                          redisFree);
if (!ctx)
    throw std::runtime_error("failed to connect to redis");

auto reply = std::unique_ptr<redisReply, decltype(&freeReplyObject)>(
                                    static_cast<redisReply*>(
                                                redisCommand(ctx.get(), // send command
                                                "SMEMBERS %b", key.data(), key.size())),
                                    freeReplyObject);
if (!reply)
    throw std::runtime_error("failed to call smembers");

std::vector<std::string> members = parse_reply(*reply);
```

hiredis

```
typedef struct redisReply {
    int type;

    long long integer;

    char* str;    // String, double

    size_t len;

    size_t elements;

    struct redisReply** element;  // Nested array
} redisReply;
```

hiredis

```
std::vector<std::string> parse_reply(redisReply& reply) {
    if (reply.type != REDIS_REPLY_ARRAY)
        throw std::runtime_error("expect array reply from for smembers");

    if (reply.element == nullptr)
        return {};

    std::vector<std::string> members;
    for (auto idx = 0U; idx != reply.elements; ++idx) {
        auto *str_reply = (reply.element[idx]);
        if (str_reply == nullptr ||
                            str_reply->type != REDIS_REPLY_STRING ||
                            str_reply->str == nullptr)
            throw std::runtime_error("invalid sub string reply");

        members.emplace_back(str_reply->str, str_reply->len);
    }

    return members;
}
```

hiredis

```
127.0.0.1:6379> MEMORY STATS
1) "peak.allocated"
2) (integer) 1135312
……
17) "db.0"
18) 1) "overhead.hashtable.main"
       2) (integer) 152
       3) "overhead.hashtable.expires"
       4) (integer) 0
……
45) "rss-overhead.ratio"
46) "1.0348072052001953"
```

redis-plus-plus

功能齐全

支持Redis命令，redis cluster, redis sentinel

pipeline，transaction，Lua scripting，pubsub

连接池，线程安全，同步/异步/协程接口

C++
客户端

Modern C++

C++11以上版本，STL风格接口

C++17特性（optional，string_view, variant）

future/callback异步接口，C++20 Coroutine

稳定可靠

线上大量使用

问题及时回复，bug及时修复

github ISSUE，Stackoverflow

更新及时

Based on hiredis

Redis 5: REDIS STRAEM

Redis 6: ACL, TLS

redis-plus-plus

```
127.0.0.1:6379> SMEMBERS NoSQL
1) "Redis"
2) "Mongodb"
3) "ElasticSearch"
4) "Cassandra"
```

redis-plus-plus

```
// tcp://user:pass@127.0.0.1:6379/2?socket_timeout=100ms"
auto redis = sw::redis::Redis("tcp://127.0.0.1:6379");

std::vector<std::string> nosqls;

redis.smembers("NoSQL", std::back_inserter(nosqls));
```

redis-plus-plus

```
redis.set("key", "value"); // std::string_view

std::optional<std::string> val = redis.get("key");
if (val) {
    std::cout << *val << std::endl;
} // else key does not exist
```

redis-plus-plus

Redis List <-> std::vector

```
redis.lpush("list", vec.begin(), vec.end());
redis.lpush("list", { "a", "b", "c" });

std::vector<std::string> items;
redis.lrange("list", 0, -1, std::back_inserter(items));
```

Redis Hash <-> std::unordered_map

```
std::unordered_map<std::string, std::string> hash = { {"k1", "v1"}, {"k2", "v2"} };
redis.hmset("hash", hash.begin(), hash.end());

std::unordered_map<std::string, std::string> langs;
redis.hgetall("langs", std::inserter(langs, langs.begin()));
```

redis-plus-plus

Generic Interface

```
auto val = redis.command<std::optional<std::string>>("get", "key");

auto set2set = redis.command<std::set<std::string>>("smembers", "set");

auto set2vec = redis.command<std::vector<std::string>>("smembers", "set");

auto list2vec = redis.command<std::vector<std::string>>("lrange", "list", 0, -1);

auto list2set = redis.command<std::set<std::string>>("lrange", "list", 0, -1);

auto hash = redis.command<std::unordered_map<std::string, std::string>>("hgetall", "hash");

auto hash2map = redis.command<std::map<std::string, std::string>>("hgetall", "hash");
```

redis-plus-plus

```
127.0.0.1:6379> MEMORY STATS
1) "peak.allocated"
2) (integer) 1135312
……
17) "db.0"
18) 1) "overhead.hashtable.main"
       2) (integer) 152
       3) "overhead.hashtable.expires"
       4) (integer) 0
……
45) "rss-overhead.ratio"
46) "1.0348072052001953"
```

redis-plus-plus

```
using Stat = std::variant<double,
                          long long,
                          std::string,
                          std::map<std::string, long long>>;

using MemoryStats = std::unordered_map<std::string, Stat>;

auto stats = redis.command<MemoryStats>("memory", "stats");
```

redis-plus-plus

Tag Dispatching

```
template <typename T>
struct ParseTag {};

template <typename T>
T parse(redisReply& reply) {
    return parse(ParseTag<T>(), reply);
}
```

redis-plus-plus

```
std::string parse(ParseTag<std::string>, redisReply& reply);

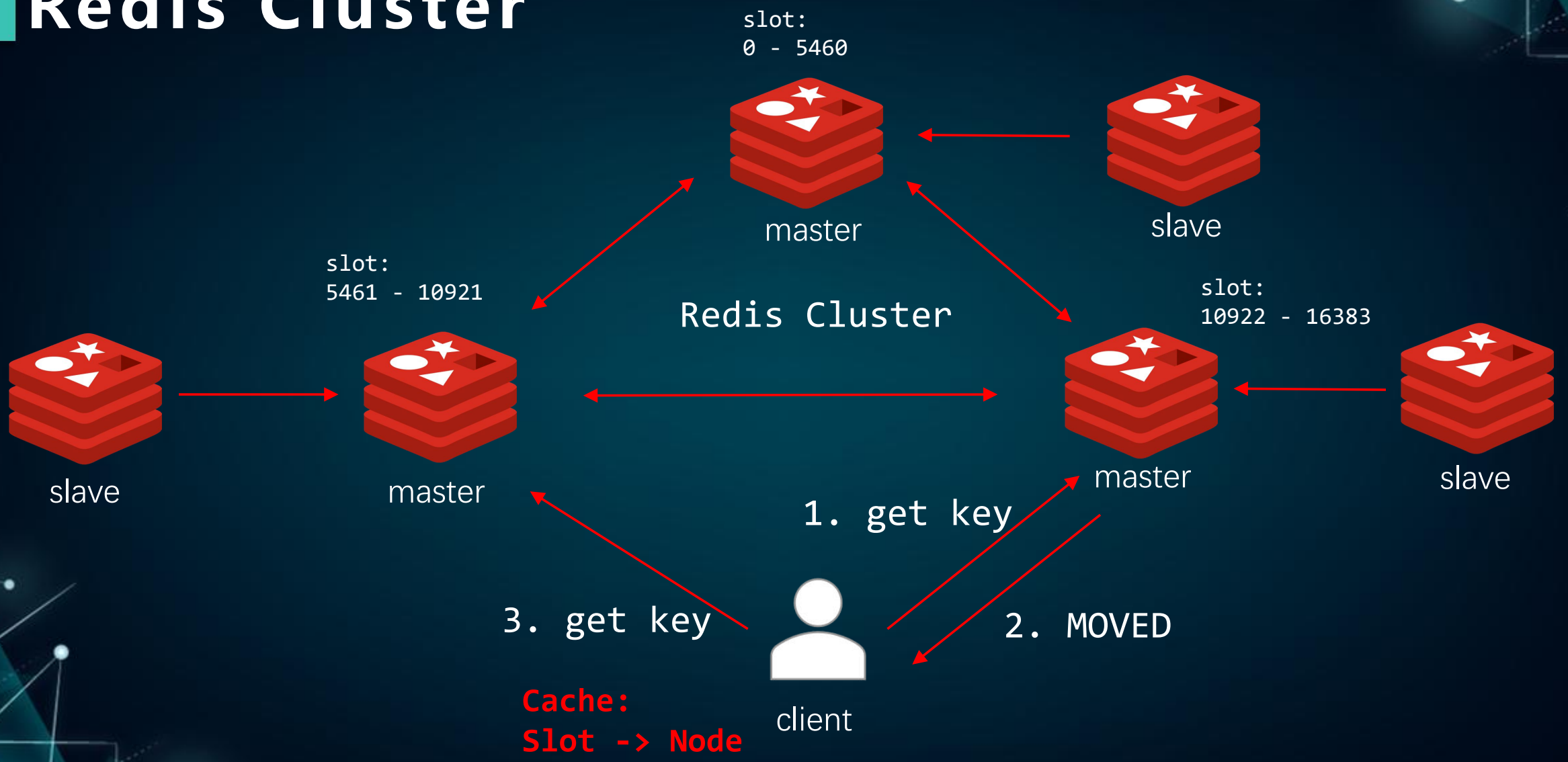
long long parse(ParseTag<long long>, redisReply& reply);

template <typename T,
          typename std::enable_if<IsSequenceContainer<T>::value, int>::type = 0>
T parse(ParseTag<T>, redisReply& reply);

template <typename T,
          typename std::enable_if<IsAssociativeContainer<T>::value, int>::type = 0>
T parse(ParseTag<T>, redisReply& reply);

template <typename ...Args>
std::variant<Args...> parse(ParseTag<std::variant<Args...>>, redisReply& reply);
```

Redis Cluster



Redis Cluster

```
auto cluster = RedisCluster("tcp://127.0.0.1:7000");

cluster.set("key", "value");

auto val = cluster.get("key");

cluster.lpush("list", { "a", "b", "c" });

std::vector<std::string> items;
cluster.lrange("list", 0, -1, std::back_inserter(items));

std::unordered_map<std::string, std::string> hash = { {"k1", "v1"}, {"k2", "v2"} };
cluster.hmset("hash", hash.begin(), hash.end());

std::unordered_map<std::string, std::string> langs;
cluster.hgetall("langs", std::inserter(langs, langs.begin()));
```

Redis Sentinel

```
SentinelOptions sentinel_opts;
sentinel_opts.nodes = { {"127.0.0.1", 9000},
                        {"127.0.0.1", 9001},
                        {"127.0.0.1", 9002} };

auto sentinel = std::make_shared<Sentinel>(sentinel_opts);

ConnectionOptions connection_opts;
connection_opts.connect_timeout = std::chrono::milliseconds(100);
connection_opts.socket_timeout = std::chrono::milliseconds(100);

ConnectionPoolOptions pool_opts;
pool_opts.size = 3;

auto redis = Redis(sentinel, "master_name",
                   Role::MASTER, connection_opts, pool_opts);
```

03

redis-protobuf

redis-protobuf

```
struct Person {
    std::string name;

    int age;

    std::vector<std::string> experience;

    std::vector<std::string> skills;

    std::unordered_map<std::string, std::string> pages;
};
```

redis-protobuf

name

experience



age

pages

skills

Nested Data Structures

MISMATCH

"string"

123

3.1415926

{hash}

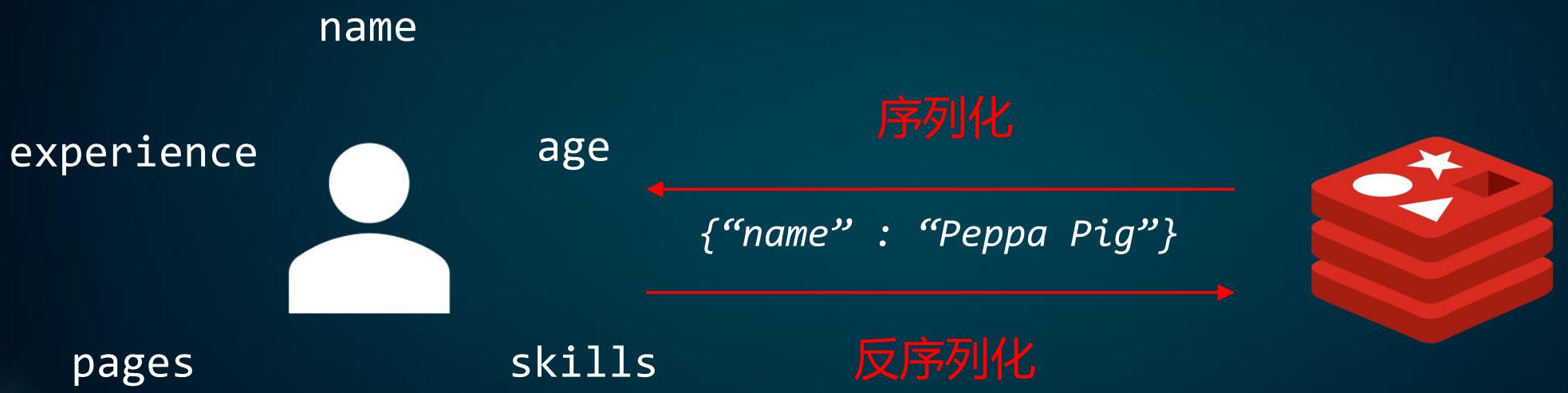
[list]

(set)

<sorted set>

Built-in Data Structures

redis-protobuf



Nested Data Structures

Built-in Data Structures

Redis Module



Redis Module

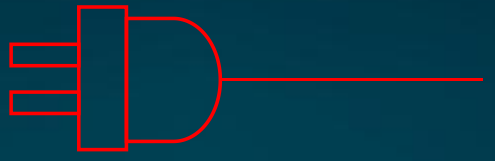
Redis Module

command

name arg1 arg2 …



RedisModule_ReplyWithXXX



RedisModule_CreateCommand



module.so

RedisModule_CreateDataType

name

callback(ctx, argc, argv)

attributes



Redis Module



RediSearch



RedisJSON



RedisAI

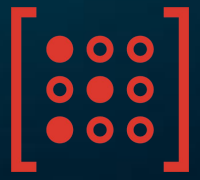




RedisBloom



RedisTimeSeries



RedisGraph

RedisJSON



RedisJSON

```
{
        "name": "Peppa",
        "age": 3,
        "experience": ["kindergarten"],
        "skills": ["jump in muddy puddles", "hide and seek"],
        "pages" : {
                "github": "github.com/peppa",
                "stackoverflow": "stackoverflow.com/peppa“
        }
}
```

Not fast enough & memory inefficient

redis-protobuf

Protocol Buffer

```
message Person {
    string name = 1;

    int32 age = 2;

    repeated string experience = 3;

    repeated string skills = 4;

    map<string, string> pages = 5;
}
```

faster & more memory efficient

redis-protobuf

```
127.0.0.1:6379> PB.SET peppa Person '{"name" : "Peppa", … }'
(integer) 1

127.0.0.1:6379> PB.GET peppa --FORMAT JSON Person
"{\"name\":\"Peppa Pig\", … }"

127.0.0.1:6379> PB.GET peppa --FORMAT BINARY Person
"\n\tPeppa Pig\x10$\x1a\..."
```

redis-protobuf

```
127.0.0.1:6379> PB.GET peppa Person /name
"Peppa"

127.0.0.1:6379> PB.GET peppa Person /skills
1)  "jump in muddy puddles"
2)  "hide and seek"

127.0.0.1:6379> PB.SET peppa Person /age 4
(integer) 1

127.0.0.1:6379> PB.APPEND peppa Person /skills "fly a kite"
(integer) 3

127.0.0.1:6379> PB.GET peppa Person /skills/2
"fly a kite"
```

redis-protobuf

```
PB.SET peppa Person /name "peppa"
```

反射

DescriptorPool::FindMessageTypeByName

"Person"

Descriptor

```
message Person {
    string name = 1;
    int32 age = 2;
    repeated string experience = 3;
    repeated string skills = 4;
    map<string, string> pages = 5;
}
```

DynamicMessageFactory::GetPrototype

DynamicMessage::New

Person → GetReflection → Reflection

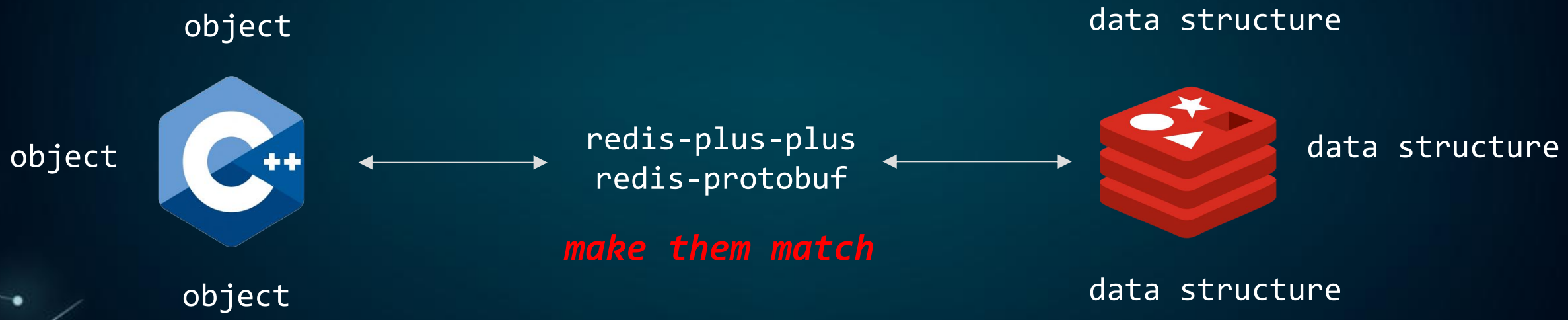
Descriptor::FindFieldByName

"name"

FieldDescriptor

SetString(person, field_desc, val)

Redis As An Object Store



谢谢观看


when
harry
met
sally

CPP-Summit 2022



王铭鑫

微软软件工程师，C++标准委员会成员

现任微软Windows团队软件工程师。曾递交过多项ISO C++标准提案，2019年作为微软公司代表及中国代表加入标准委员会。在OOP理论基础上完成创新性突破并实现了proxy库，现已应用于微软公司多个产品中并开源。作为技术负责人主导过多个服务端和桌面软件项目研发，对优秀的软件架构设计和前沿程序设计理论有不懈的追求。

主办方：Boolan 高端IT咨询与教育平台

CPP-Summit

Mingxin Wang
王铭鑫
mingxwa@microsoft.com

proxy:
Mastering Polymorphism
驾驭多态，无惧规模

About the presenter

Mingxin Wang
王铭鑫

Software Engineer @ Microsoft OS team
受雇于微软公司

ISO C++ committee member (Representing Microsoft and China, China national body chair)
C++标准委员会成员（代表微软公司和中国，兼任中国成员国主席）

Overview

1. **History**
   多态的发展史
2. **Motivation**
   proxy库的设计动机
3. **Getting started**
   proxy库初体验
4. **Mastering polymorphism**
   使用proxy库轻松驾驭多态

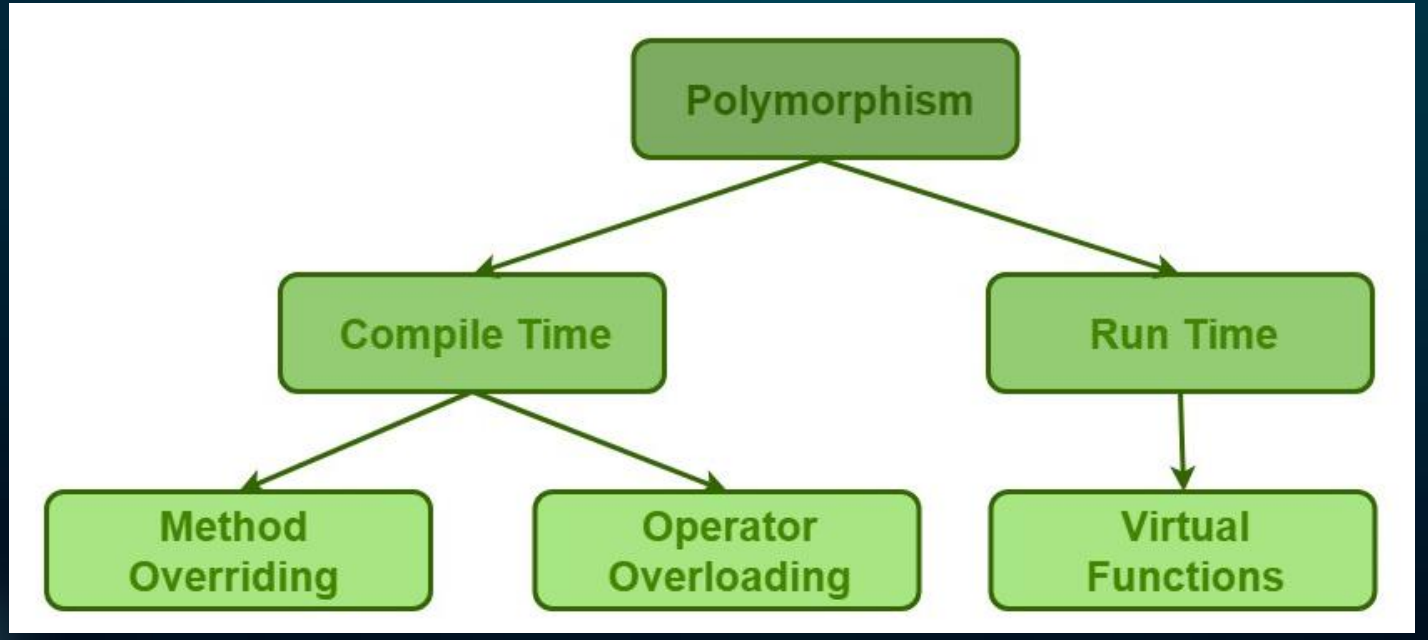
01

History
多态发展史

What is “polymorphism”
什么是多态

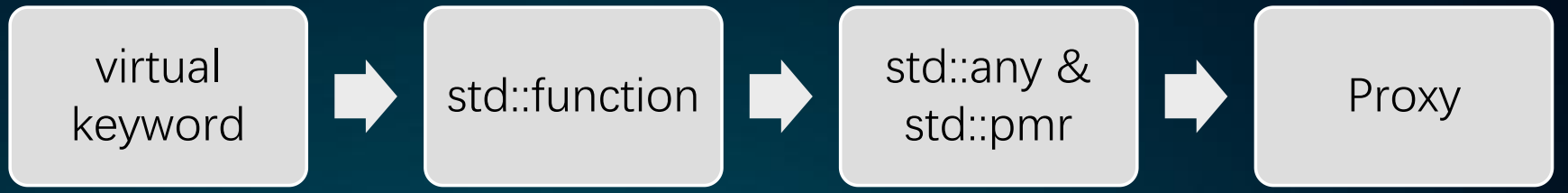
Polymorphism is the provision of a single interface to entities of different types.
多态是为不同类型的实体提供的一个单一接口。

– Bjarne Stroustrup



Polymorphism in different languages
不同语言中的多态

C++



1985 (1982) 2011 2017 2022

C#

virtual & interface keywords

2000

Go

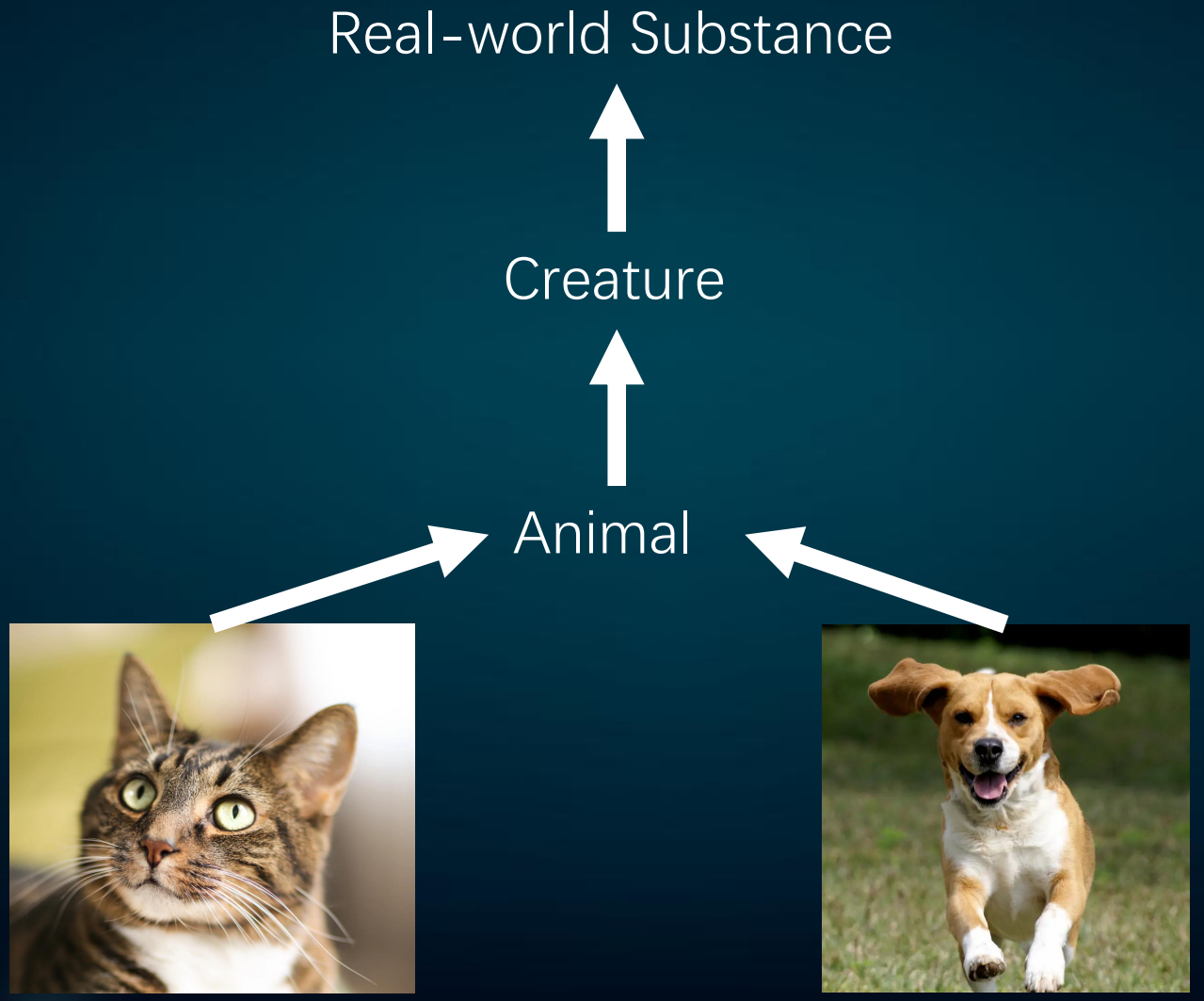
interface keyword

2012

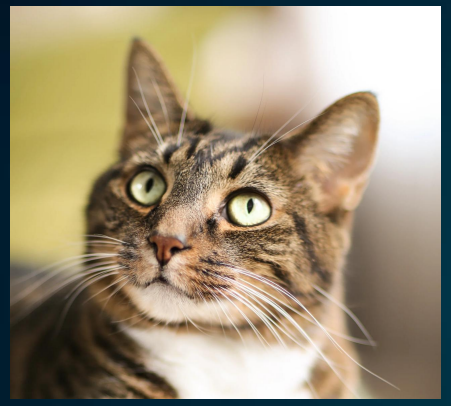
Polymorphism in different languages
不同语言中的多态

	Intrusive 入侵式	Non-intrusive 非入侵式
With GC 带有GC	Java, C#	Go
Without GC 不带有GC	C++	Rust*, ?

What is “intrusive polymorphism”
什么是入侵式多态

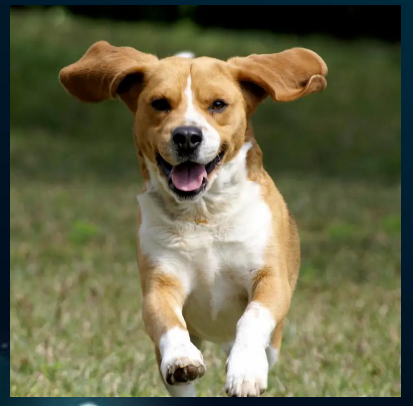


What is “non-intrusive polymorphism”
什么是非入侵式多态



Regard as

Pet



02

Motivation
动机

The power of “proxy”
proxy库的能耐

- Facilitating architecture design and matainance by using duck typing in C++ as easily as in Rust or Golang!
  像Rust或Golang一样在C++中使用鸭子类型，从而让软件架构设计更简单!
- Facilitating lifetime management of polymorphic objects as easily as in languages with runtime Garbage Collection (GC, like Java or C#), without compromising performance!
  像Java或C#等带有运行时GC的语言一样轻松管理多态对象生命周期，但无需牺牲性能!
- Supporting compile-time reflection and expert performance turning per case!
  在特定需求下集成定制化的编译时反射和极致的性能调优!

The 2 core competencies
两大核心竞争力

- Being non-intrusive
  实现非入侵式多态
- Lifetime management per object
  对象粒度的生命周期管理

Core competencies 1: Being non-intrusive
核心竞争力之一：非入侵式多态

Goal: Make abstraction of a resource dictionary

Abstraction:

```
struct at : pro::dispatch<std::string(int)> {
  template <class T>
  auto operator()(T& self, int key) { return self.at(key); }
};
struct ResourceDictionaryFacade : pro::facade<at> {};
```

Library:

```
void DoSomethingWithResourceDictionary(pro::proxy<ResourceDictionaryFacade> p);
```

Client:

```
std::map<int, std::string> var1{{1, "Hello"}};
std::vector<std::string> var2{"I", "love", "Proxy", "!"};
DoSomethingWithResourceDictionary(&var1);  // Prints "Hello"
DoSomethingWithResourceDictionary(&var2);  // Prints "love"
DoSomethingWithResourceDictionary(std::make_shared<std::unordered_map<int, std::string>>());
```

Core competencies 1: Being non-intrusive
核心竞争力之一：非入侵式多态

- The requirements of a polymorphic type, by definition, comes from its use, and there are no polymorphic types, only polymorphic use of similar types. Inheritance is the base class of evil.
  根据定义，多态类型的需求来自于其使用。没有多态对象，只有对于类似对象的多态用法。继承是万恶之源。

- Sean Parent

More talks around non-intrusive polymorphism
关于非入侵多态的更多演讲

- Better Code: Runtime Polymorphism
- Sean Parent
- NDC 2017
- https://www.youtube.com/watch?v=QGcVXgEVMJg&t=1069s

- Polymorphism != Virtual: Easy, Flexible Runtime Polymorphism Without Inheritance
- John Bandela
- CppCon 2019
- https://www.youtube.com/watch?v=PSxo85L2IC0

- Runtime Polymorphism: Back to the Basics
- Louis Dionne
- CppCon 2017
- https://www.youtube.com/watch?v=gVGtNFg4ay0&t=337s

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

Goal: Implement a simple factory:

```
?? MakeDrawableFromCommand(const std::string& s)
```

```
IDrawable* MakeDrawableFromCommand(
    const std::string& s) ?

std::unique_ptr<IDrawable> MakeDrawableFromCommand(
    const std::string& s) ?

std::shared_ptr<IDrawable> MakeDrawableFromCommand(
    const std::string& s) ?

some_gc_ptr<IDrawable> MakeDrawableFromCommand(
    const std::string& s) ?
```

Before

```
std::proxy<FDrawable> MakeDrawableFromCommand(
    const std::string& s);
```

After

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

```
std::proxy<FDrawable> MakeDrawableFromCommand(const std::string& s) {
  std::vector<std::string> parsed = ParseCommand(s);
  if (!parsed.empty()) {
    if (parsed[0u] == "Rectangle") {
      if (parsed.size() == 3u) {
        static std::pmr::unsynchronized_pool_resource rectangle_memory_pool;
        std::pmr::polymorphic_allocator<> alloc{&rectangle_memory_pool};
        auto deleter = [alloc](Rectangle* ptr) mutable
            { alloc.delete_object<Rectangle>(ptr); };
        Rectangle* instance = alloc.new_object<Rectangle>();
        std::unique_ptr<Rectangle, decltype(deleter)> p{instance, deleter};
        p->SetWidth(std::stod(parsed[1u]));
        p->SetHeight(std::stod(parsed[2u]));
        return p;
      }
    } else if (parsed[0u] == "Circle") {
      if (parsed.size() == 2u) {
        Circle circle;
        circle.SetRadius(std::stod(parsed[1u]));
        return std::make_proxy<FDrawable>(circle);
      }
    } else if (parsed[0u] == "Point") {
      if (parsed.size() == 1u) {
        static Point instance;
        return &instance;
      }
    }
  }
  throw std::runtime_error{"Invalid command"};
}
```

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

```
std::proxy<FDrawable> MakeDrawableFromCommand(const std::string& s) {
  std::vector<std::string> parsed = ParseCommand(s);
  if (!parsed.empty()) {
    if (parsed[0u] == "Rectangle") {
      if (parsed.size() == 3u) {
        static std::pmr::unsynchronized_pool_resource rectangle_memory_pool;
        std::pmr::polymorphic_allocator<> alloc{&rectangle_memory_pool};
        auto deleter = [alloc](Rectangle* ptr) mutable
            { alloc.delete_object<Rectangle>(ptr); };
        Rectangle* instance = alloc.new_object<Rectangle>();
        std::unique_ptr<Rectangle, decltype(deleter)> p{instance, deleter};
        p->SetWidth(std::stod(parsed[1u]));
        p->SetHeight(std::stod(parsed[2u]));
        return p;
      }
    } else if (parsed[0u] == "Circle") {
      if (parsed.size() == 2u) {
        Circle circle;
        circle.SetRadius(std::stod(parsed[1u]));
        return std::make_proxy<FDrawable>(circle);
      }
    } else if (parsed[0u] == "Point") {
      if (parsed.size() == 1u) {
        static Point instance;
        return &instance;
      }
    }
  }
  throw std::runtime_error{"Invalid command"};
}
```

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

```
std::proxy<FDrawable> MakeDrawableFromCommand(const std::string& s) {
  std::vector<std::string> parsed = ParseCommand(s);
  if (!parsed.empty()) {
    if (parsed[0u] == "Rectangle") {
      if (parsed.size() == 3u) {
        static std::pmr::unsynchronized_pool_resource rectangle_memory_pool;
        std::pmr::polymorphic_allocator<> alloc{&rectangle_memory_pool};
        auto deleter = [alloc](Rectangle* ptr) mutable
            { alloc.delete_object<Rectangle>(ptr); };
        Rectangle* instance = alloc.new_object<Rectangle>();
        std::unique_ptr<Rectangle, decltype(deleter)> p{instance, deleter};
        p->SetWidth(std::stod(parsed[1u]));
        p->SetHeight(std::stod(parsed[2u]));
        return p;
      }
    } else if (parsed[0u] == "Circle") {
      if (parsed.size() == 2u) {
        Circle circle;
        circle.SetRadius(std::stod(parsed[1u]));
        return std::make_proxy<FDrawable>(circle);
      }
    } else if (parsed[0u] == "Point") {
      if (parsed.size() == 1u) {
        static Point instance;
        return &instance;
      }
    }
  }
  throw std::runtime_error{"Invalid command"};
}
```

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

```
std::proxy<FDrawable> MakeDrawableFromCommand(const std::string& s) {
  std::vector<std::string> parsed = ParseCommand(s);
  if (!parsed.empty()) {
    if (parsed[0u] == "Rectangle") {
      if (parsed.size() == 3u) {
        static std::pmr::unsynchronized_pool_resource rectangle_memory_pool;
        std::pmr::polymorphic_allocator<> alloc{&rectangle_memory_pool};
        auto deleter = [alloc](Rectangle* ptr) mutable
            { alloc.delete_object<Rectangle>(ptr); };
        Rectangle* instance = alloc.new_object<Rectangle>();
        std::unique_ptr<Rectangle, decltype(deleter)> p{instance, deleter};
        p->SetWidth(std::stod(parsed[1u]));
        p->SetHeight(std::stod(parsed[2u]));
        return p;
      }
    } else if (parsed[0u] == "Circle") {
      if (parsed.size() == 2u) {
        Circle circle;
        circle.SetRadius(std::stod(parsed[1u]));
        return std::make_proxy<FDrawable>(circle);
      }
    } else if (parsed[0u] == "Point") {
      if (parsed.size() == 1u) {
        static Point instance;
        return &instance;
      }
    }
  }
  throw std::runtime_error{"Invalid command"};
}
```

Core competencies 2: Lifetime management
核心竞争力之二：生命周期管理

- **The top-secret making proxy both easy-to-use and fast is that it allows lifetime management *per object***, which is a breakthrough innovation of Object-oriented Programming (OOP) theory in recent years.
  proxy保持易用性和高性能的最大的秘密是，它允许每个对象的生命周期管理，这是我们近年来对于传统的OOP理论（基于继承的多态性）的突破性创新。

03

Getting started
proxy库初体验

Live demo

04

Mastering polymorphism
使用proxy库轻松驾驭多态

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Code generation
代码生成

	Inheritance-based polymorphism	The "proxy"

Abstraction	`struct IDrawable {`
	`  virtual void Draw() const = 0;`
	`  virtual double Area() const = 0;`
	`  virtual ~IDrawable() {}`
	`};`
		`struct Draw : std::dispatch<void()> {`
		`  template <class T>`
		`  void operator()(const T& self) { self.Draw(); }`
		`};`
		`struct Area : std::dispatch<double()> {`
		`  template <class T>`
		`  double operator()(const T& self) { return self.Area(); }`
		`};`
		`struct FDrawable : std::facade<Draw, Area> {};`

Library	`class Rectangle : public IDrawable {`
	` public:`
	`  void Draw() const override {`
	`  printf("{Rectangle: width = %f, height = %f}",`
	`  width_, height_);`
	`  }`
	`  double Area() const override`
	`  { return width_ * height_; }`
	
	` private:`
	`  double width_;`
	`  double height_;`
	`};`
		`class Rectangle {`
		` public:`
		`  void Draw() const {`
		`  printf("{Rectangle: width = %f, height = %f}",`
		`  width_, height_);`
		`  }`
		`  double Area() const { return width_ * height_; }`
		
		` private:`
		`  double width_;`
		`  double height_;`
		`};`

Client	`void DoSomethingWithDrawable(std::unique_ptr<IDrawable> p) {`
	`  p->Draw();`
	`}`
		`void DoSomethingWithDrawable(std::proxy<FDrawable> p) {`
		`  p.invoke<op::Draw>();`
		`}`

Code generation
代码生成

	Inheritance-based polymorphism	The "proxy"

Library	`mov rdi, qword ptr [rdi]`
	`mov rax, qword ptr [rdi]`
	`jmp qword ptr [rax]`
		`mov rax, qword ptr [rdi]`
		`add rdi, 8`
		`jmp qword ptr [rax + 24]`

Client	`movsd xmm0, qword ptr [rdi + 8]`
	`movsd xmm1, qword ptr [rdi + 16]`
	`mov edi, offset .L.str`
	`mov al, 2`
	`jmp printf`
		`mov rax, qword ptr [rdi + 8]`
		`movsd xmm0, qword ptr [rax]`
		`movsd xmm1, qword ptr [rax + 8]`
		`mov edi, offset .L.str.18`
		`mov al, 2`
		`jmp printf`

Generated assembly from clang 13.0.0 (x86-64) *

	Inheritance-based polymorphism	The "proxy"

Library	`ldr x0, [x0]`
	`ldr x1, [x0]`
	`ldr x1, [x1]`
	`mov x16, x1`
	`br x16`
		`ldr x1, [x0], 8`
		`ldr x1, [x1, 24]`
		`mov x16, x1`
		`br x16`

Client	`mov x1, x0`
	`adrp x2, .LC0`
	`add x0, x2, :lo12:.LC0`
	`ldp d0, d1, [x1, 8]`
	`b printf`
		`mov x1, x0`
		`adrp x0, .LC3`
		`add x0, x0, :lo12:.LC3`
		`ldr d0, [x1]`
		`b printf`

Generated assembly from gcc 11.2 (ARM64) *

* Compiler flags: -std=c++20 -O3

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Flexible composition of dispatch hierarchies
组合不同抽象定义的支持

```
struct D1; struct D2; struct D3;
struct FA : std::facade<D1, D2, D3> {};
struct FB : std::facade<D1, std::tuple<D3, D2>> {};
struct FC : std::facade<std::tuple<D1, D2, D3>, D1, std::tuple<D2, D3>> {};
```

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Compatible with general-purpose compile-time reflection
支持通用编译时（静态）反射

```
class MyReflectionInfo {
 public:
  template <class P>
  constexpr explicit MyReflectionInfo(std::in_place_type_t<P>)
      : type_(typeid(P)) {}

  const char* GetName() const noexcept { return type_.name(); }

 private:
  const std::type_info& type_;
};

struct FResourceDictionary : std::facade<at> {
  using reflection_type = MyReflectionInfo;
};

std::proxy<FResourceDictionary> p;
p.reflect().GetName();
```

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Using with CPO (Customization Point Object) and modules
结合定制点对象（CPO）和modules的使用语法

```
struct GetSize : pro::dispatch<std::size_t(), std::ranges::size> {};

template <class T>
struct ForEach : pro::dispatch<void(pro::proxy<CallableFacade<void(T&)>>)> {
  template <class U>
  void operator()(U& self, pro::proxy<CallableFacade<void(T&)>>&& func) {
    for (auto& value : self) {
      func.invoke(value);
    }
  }
};
template <class T>
struct IterableFacade : pro::facade<ForEach<T>, GetSize> {};
```

Using with CPO (Customization Point Object) and modules
结合定制点对象（CPO）和modules的使用语法

```
struct GetSize : pro::dispatch<std::size_t(),
    [](auto& value) { return value.size(); }> {};
```

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Expert performance tuning
面向专家的性能调优的支持

Constraints	Defaults
Maximum size *	No less than the size of two pointers
Maximum alignment *	No less than the alignment of a pointer
Minimum copyability **	None
Minimum relocatability **	Nothrow
Minimum destructibility **	Nothrow

```
struct FResourceDictionary : std::facade<at> {
  static constexpr std::size_t maximum_size = sizeof(std::optional<std::vector<std::string>>);
  static constexpr std::constraint_level minimum_copyability = std::constraint_level::nontrivial;
};
```

* It is encouraged (for implementation) to set the default maximum size and maximum alignment greater than or equal to the implementation of raw pointers, **`std::unique_ptr`** with default deleters, **`std::unique_ptr`** with any one-pointer-size of deleters (for pooling) and **`std::shared_ptr`** of any type.

** Applicable constraint levels: **None, Non-trivial, Nothrow, Trivial**

Expert performance tuning
面向专家的性能调优的支持

```
// Abstraction
struct MyFacade : pro::facade</* Omitted */> {
  static constexpr auto minimum_copyability = pro::constraint_level::nontrivial;
};

// Client
pro::proxy<MyFacade> p0 = /* Omitted */;
auto p1 = p0;  // Calls the constructor of the underlying pointer type
```

*

Expert performance tuning
面向专家的性能调优的支持

```
// Abstraction
struct MyFacade : pro::facade</* Omitted */> {
  static constexpr auto minimum_copyability = pro::constraint_level::trivial;
  static constexpr auto minimum_relocatability = pro::constraint_level::trivial;
  static constexpr auto minimum_destructibility = pro::constraint_level::trivial;
  static constexpr auto maximum_size = sizeof(void*);
  static constexpr auto maximum_alignment = alignof(void*);
};

// Client
static_assert(std::is_trivially_copy_constructible_v<pro::proxy<MyFacade>>);
static_assert(std::is_trivially_destructible_v<pro::proxy<MyFacade>>);
```

*

Understanding “proxy”
理解proxy

- High-quality code generation with best-in-class performance
  高质量代码生成保证了高性能
- Flexible composition of dispatch hierarchies
  组合不同抽象定义的支持
- Compatible with general-purpose compile-time reflection
  支持通用编译时（静态）反射
- Using with CPO (Customization Point Object) and modules
  结合定制点对象（CPO）和modules的使用语法
- Expert performance tuning
  面向专家的性能调优的支持
- Support for recursive definition of abstraction
  抽象定义递归的支持

Support for recursive definition of abstraction
抽象定义递归的支持

```
struct PushBackInt;

struct ToString : pro::dispatch<std::string()> {
  template <class T>
  std::string operator()(T& self) { return details::ToString(self); }
};

struct Appendable : pro::facade<PushBackInt, ToString> {};

struct PushBackInt : pro::dispatch<pro::proxy<Appendable>(int)> {
  template <class U>
  auto operator()(U& self, int value) {
    self.push_back(value);
    return &self;
  }
};
```

Live demo

Resources
资源

Paper: https://wg21.link/p0957
Blogpost: https://devblogs.microsoft.com/cppblog/proxy-runtime-polymorphism-made-easier-than-ever/

Implementation: https://github.com/microsoft/proxy (526 lines)
Vcpkg port: proxy

Q & A



Phil Nash

开发者测试专家， Catch创始人

Phil Nash 是著名的单元测试工具Catch/Catch2的创始人，作为JetBrains的技术专家，他的工作主要涉及CLion, AppCode 和 ReSharper C++。他是优秀测试实践、测试驱动开发（TDD）的倡导者。他曾长期担任技术顾问和技术教练工作。他同时C++ London 的创办人， C++ on Sea的主席。

ZEN AND THE ART OF SOFTWARE QUALITY



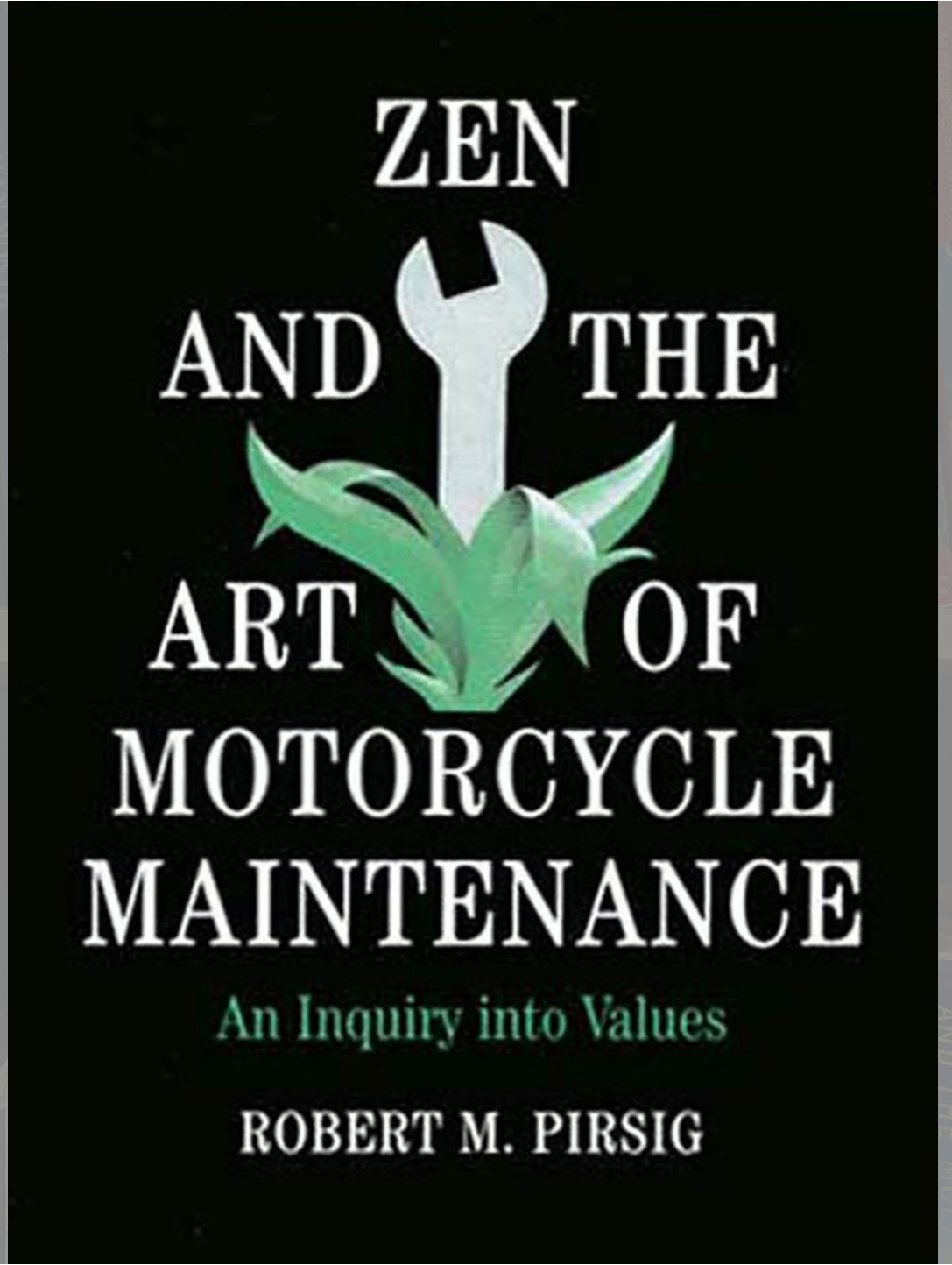
@phil_nash

sonarsource

ZEN AND THE ART OF CODE LIFE-CYCLE MAINTENANCE



@phil_nash
sonarsource


ZEN
AND THE
ART OF
MOTORCYCLE
MAINTENANCE
An Inquiry into Values
ROBERT M. PIRSIG

http://shape-of-code.coding-guidelines.com/2021/03/28/the-aura-of-software-quality/

The aura of software quality

March 28, 2021 Derek Jones Go to comments Leave a comment

Bad money drives out good money, is a financial adage. The corresponding research adage might be "research hyperbole incentivizes more hyperbole".

Software quality appears to be the most commonly studied problem in software engineering. The reason for this is that use of the term software quality imbues what is said with an aura of relevance; all that is needed is a willingness to assert that some measured attribute is a metric for software quality.

Using the term "software quality" to appear relevant is not limited to researchers; consultants, tool vendors and marketers are equally willing to attach "software quality" to whatever they are selling.

When reading a research paper, I usually hit the delete button as soon as the authors start talking about software quality. I get very irritated when what looks like an interesting paper starts spewing "software quality" nonsense.

RSS

Recent Posts

- Evidence-based SE groups doing interesting work, 2021 version
- Looking for a measurable impact from developer social learning
- Two failed software development projects in the High Court
- Electronic Evidence and Electronic Signatures: book
- The Approximate Number System and software estimating

Recent Comments

experiments.

While the experiments were interesting, a multi-site effort and just the kind of thing others should be doing, the results were hardly earth-shattering (the experimental setup was dictated by the practicalities of obtaining the data). I understand why the authors felt the need for some hyperbole (but 12-pages). I hope they continue with this work (with less hyperbole).

Anybody skimming the software engineering research literature will be dazed by the number and range of factors appearing to play a major role in software quality. Once they realize that "software quality" is actually a meaningless marketing term, they are back to knowing nothing. Every paper has to be read to figure out what definition is being used for "software quality"; reading a paper's abstract does not provide the needed information. This is a nightmare for anybody seeking some understanding of what is known about software engineering.

When writing my evidence-based software engineering book I was very careful to stay away from the term "software quality" (one paper on perceptions of software product quality is discussed, and there are around 35 occurrences of the word "quality").

People in industry are very interested in software quality, and sometimes they have the confusing experience of talking to me about it. My first response, on being asked about software quality, is to ask what the questioner means by software quality. After letting them fumble around for 10 seconds or so, trying to articulate an answer, I offer several possibilities (which they are often not happy with). Then I explain how **"software quality" is a meaningless marketing term.** This leaves them confused and unhappy. People have a yearning for software quality which makes them easy prey for the snake-oil salesmen.

Uncategorized marketing, perception, quality, snake oil

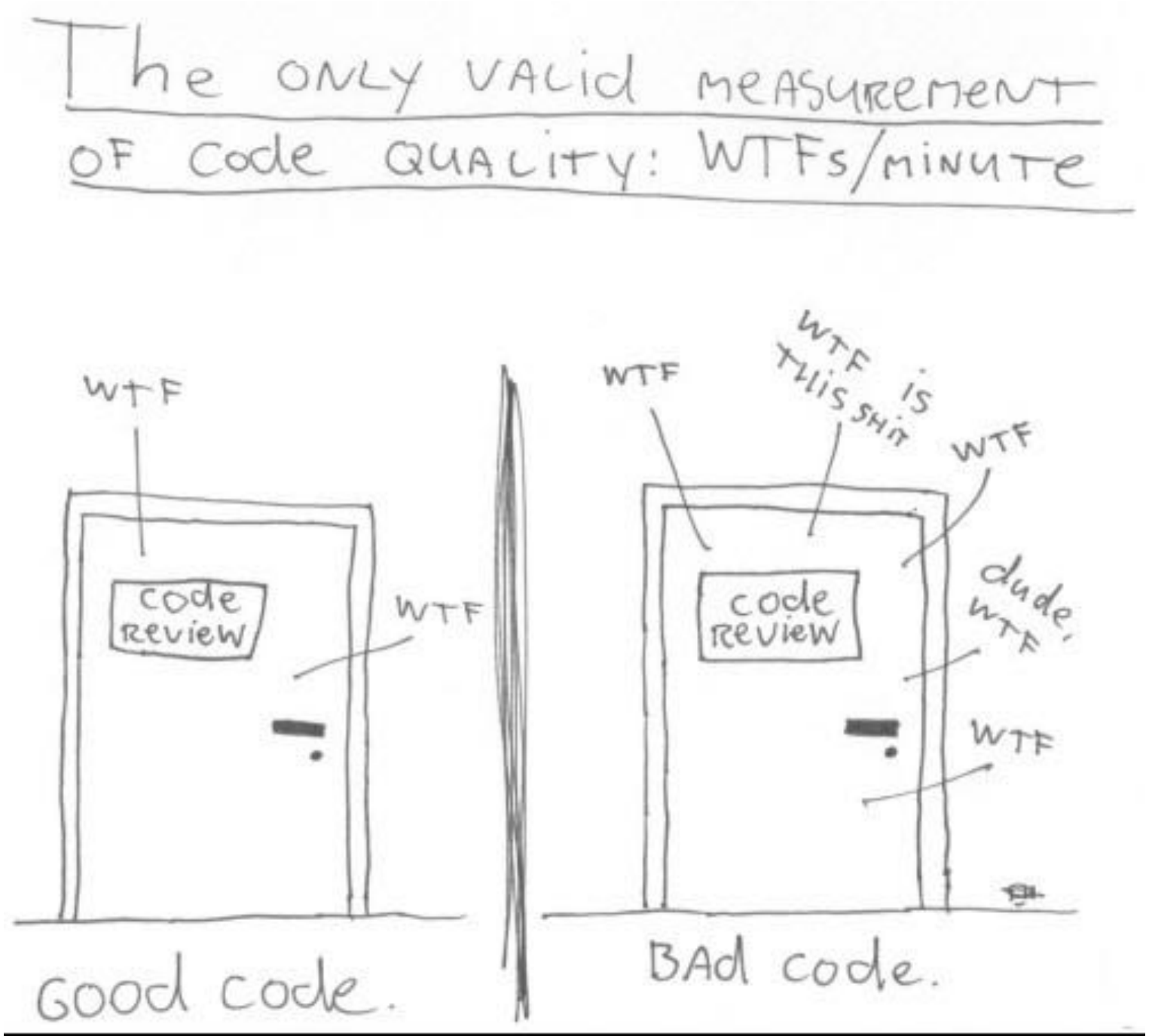
- John Carter on Mutation testing: its days in the limelight are over
- Nemo on Multiple estimates for the same project

Tags

academic Ada benchmark book books C Cobol compiler compiler writer cost/benefit data data analysis economics ecosystems empirical evolution experiment faults floating-point Fortran gcc Hackathon hardware history if statement ISO Standard Java language llvm machine learning optimizing parsing performance programming language projects Python R reliability research source code Standard static analysis testing the future tools

Archives

- October 2021
- September 2021
- August 2021
- July 2021


The ONLY VALID MEASUREMENT OF CODE QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

https://en.wikipedia.org/wiki/Software_quality

Definitions

ISO [edit]

Software quality is "capability of a software product to conform to requirements."[35][36] while for others it can be synonymous with customer- or value-creation[37][38] or even defect level.[39]

ASQ [edit]

ASQ uses the following definition: *Software quality* describes the desirable attributes of software products. There are two main approaches exist: defect management and quality attributes.[40]

The problem inherent in attempts to define the quality of a product, almost any product, were stated by the master Walter A. Shewhart. The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price that the user will pay. This is not easy, and as soon as one feels fairly successful in the endeavor, he finds that the needs of the consumer have changed, competitors have moved in, etc.[49]

— W. Edwards Deming

Tom DeMarco has proposed that "a product's quality is a function of how much it changes the world for the better.

Another definition, coined by Gerald Weinberg in Quality Software Management: Systems Thinking, is

"Quality is value to some person."[52][53]


CISQ
Consortium for Information & Software Quality ™

CISQ
Consortium for Information & Software Quality™

Structural quality refers to the software's security, reliability, performance efficiency and maintainability

CISQ
Consortium for Information & Software Quality™

Structural quality refers to the software's

security

reliability

performance efficiency

maintainability

security	correctness
reliability	reliability
performance efficiency	performance/ efficiency
maintainability	malleability/ evolvability

security

reliability

performance efficiency

maintainability

correctness

reliability

performance/ efficiency

malleability/ evolvability

reasonability

security

reliability

performance efficiency

maintainability

applicability

correctness

reliability

performance/ efficiency

malleability/ evolvability

reasonability

applicability

correctness

reliability

performance/ efficiency

malleability/ evolvability

reasonability

correctness

applicability

reliability

evolvability

efficiency

reasonability

correctness

applicability

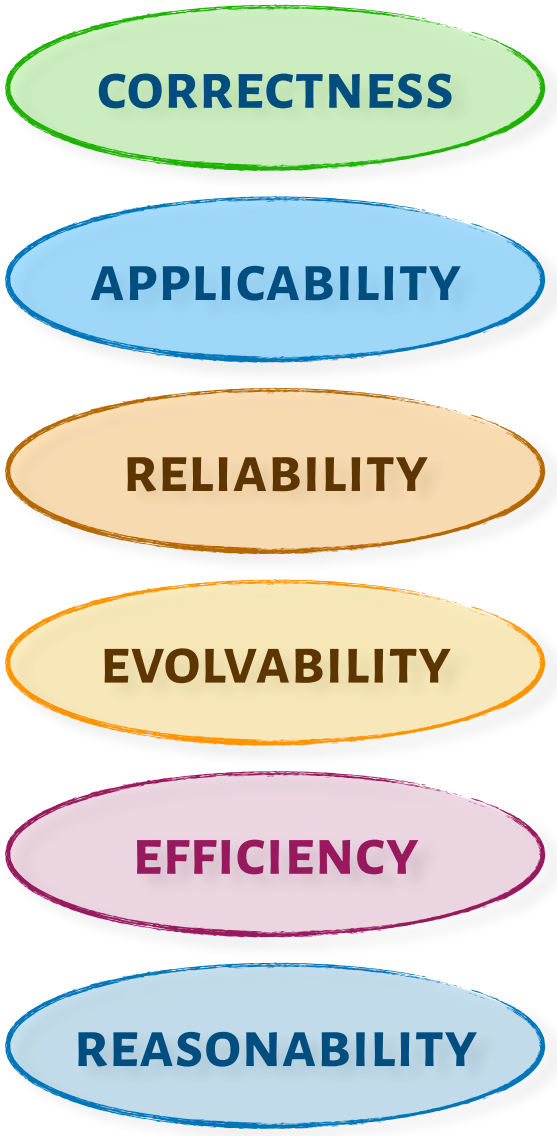
reliability

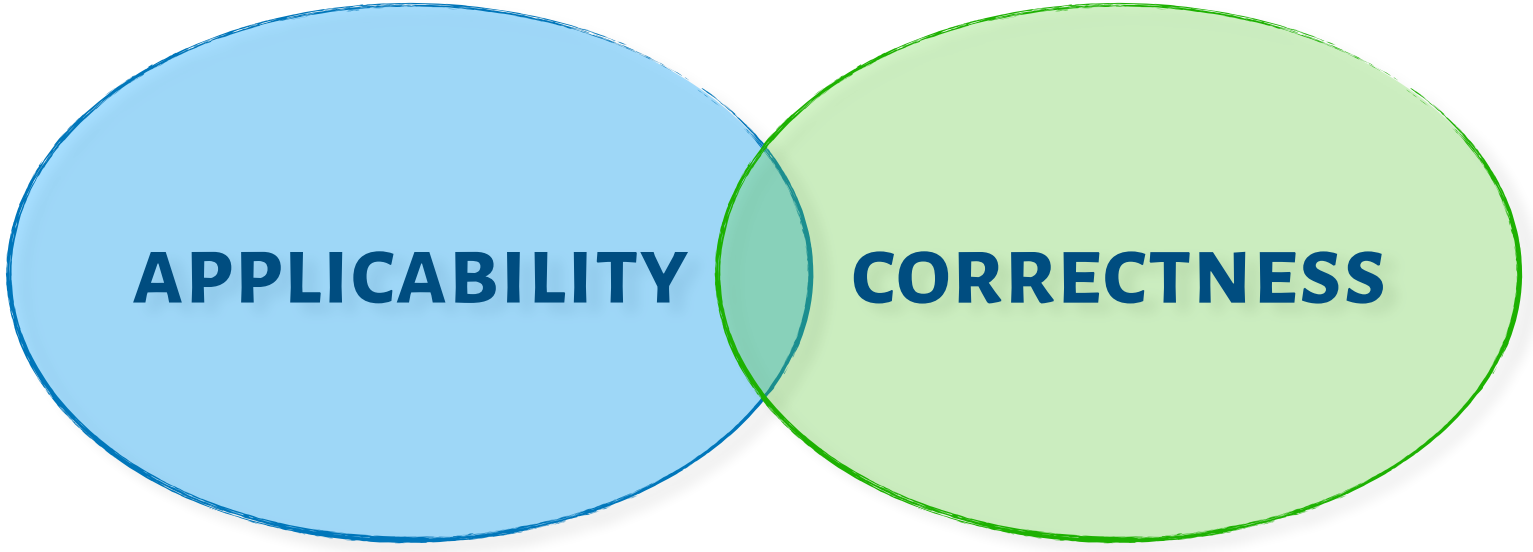
evolvability

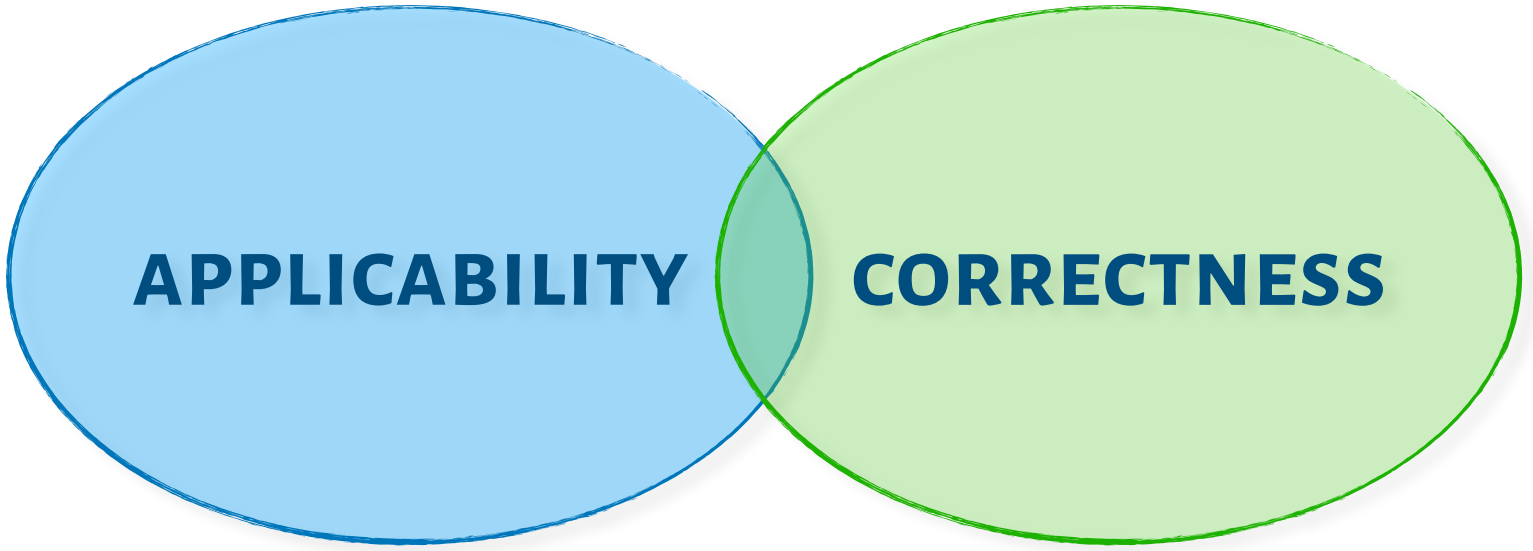
efficiency

reasonability

career

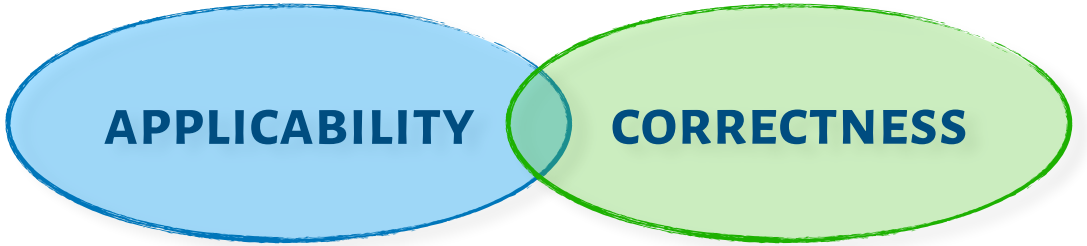

CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY


APPLICABILITY
CORRECTNESS



Do the *right thing*

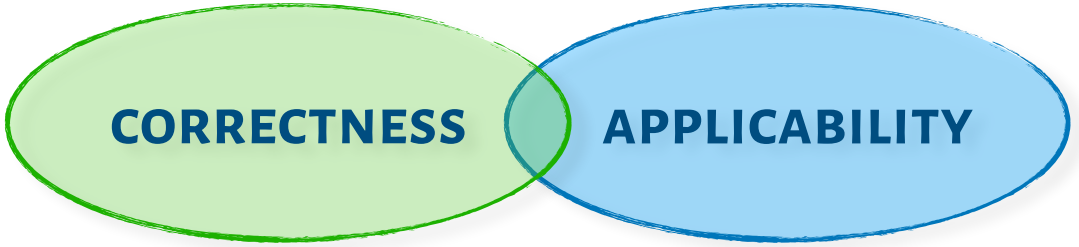
Do the thing *right*



it is better to do the right thing wrong
than the wrong thing right

Russell L. Ackoff





Software Strategy



Do it Right
Then
Do the Right thing

allan kelly
Twitter: @allankellynet
http://www.allankelly.net

Exhibit A - The Alignment Trap

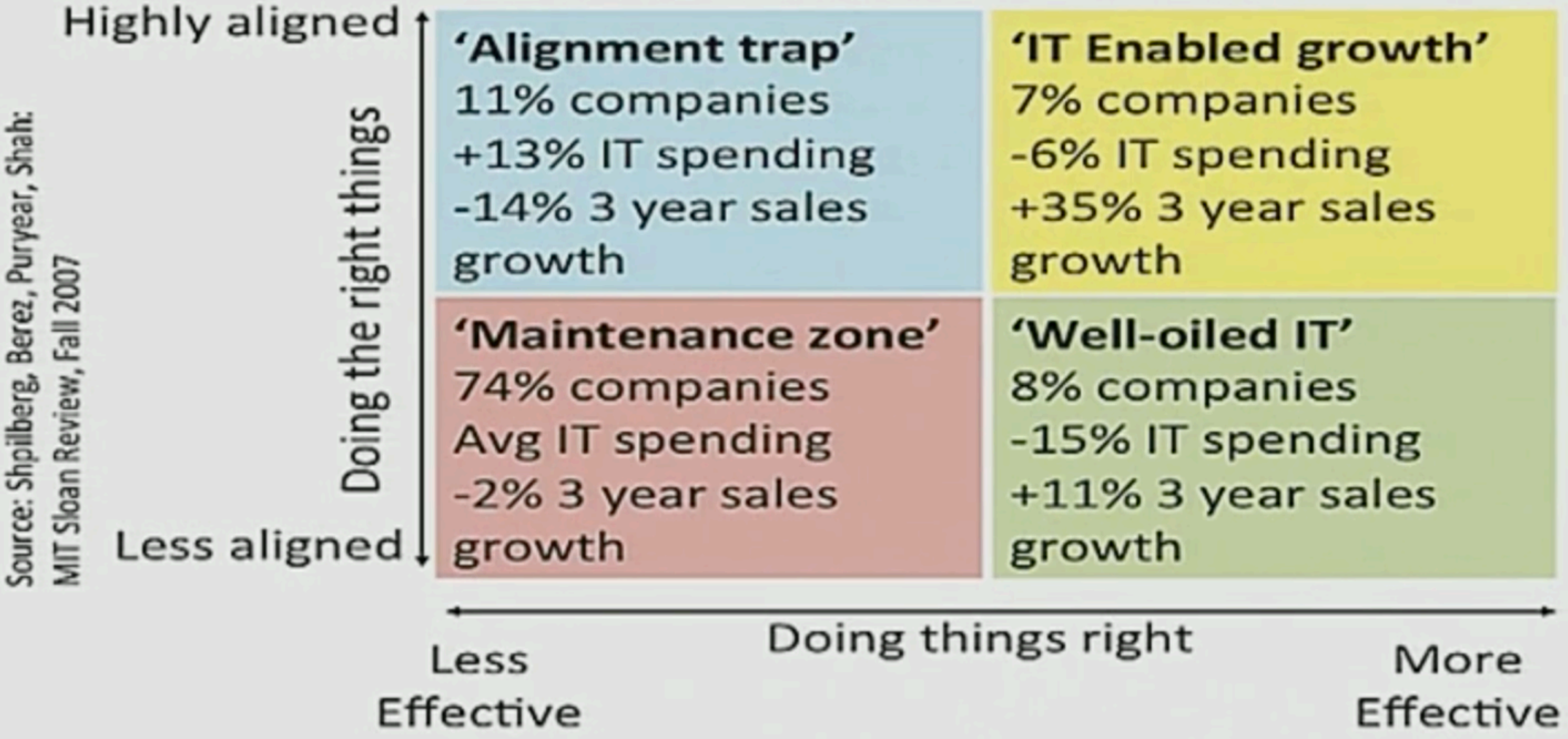


Exhibit A - The Alignment Trap

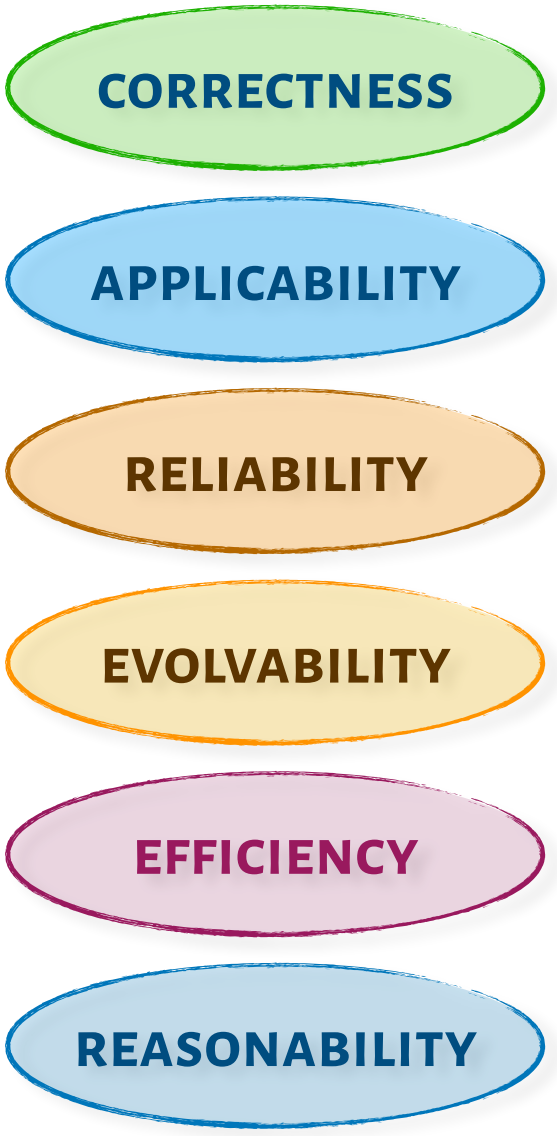


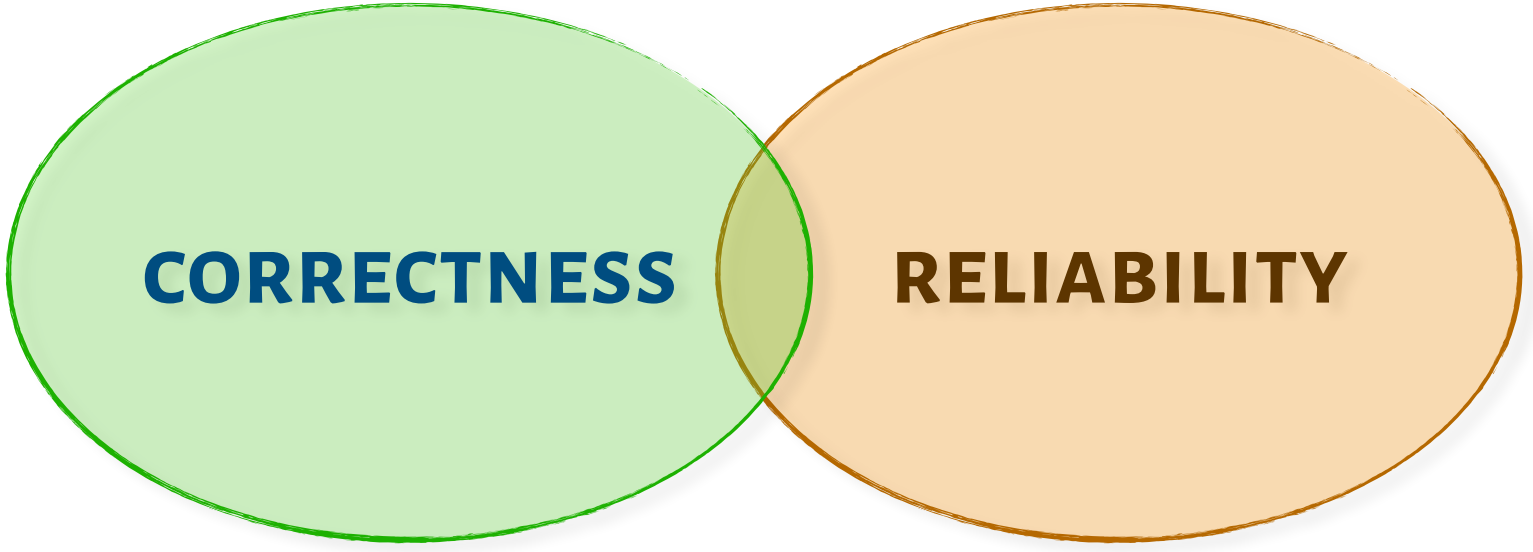
Exhibit A - The Alignment Trap



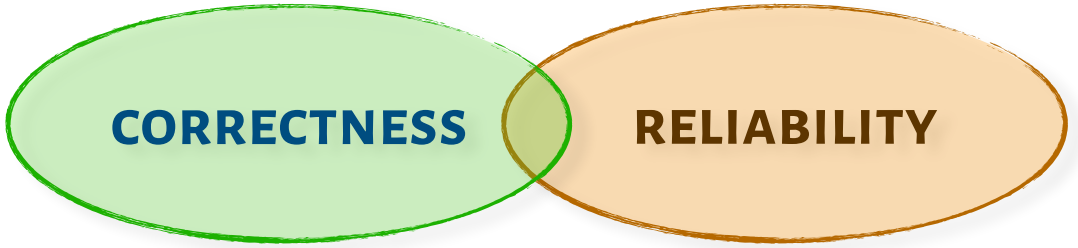
Exhibit A - The Alignment Trap



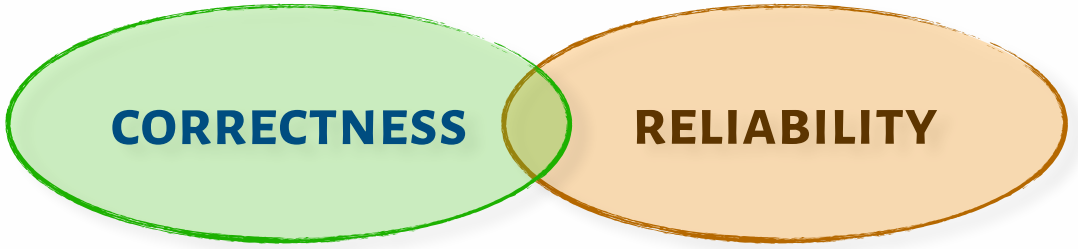

CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY


CORRECTNESS
RELIABILITY

Code coverage

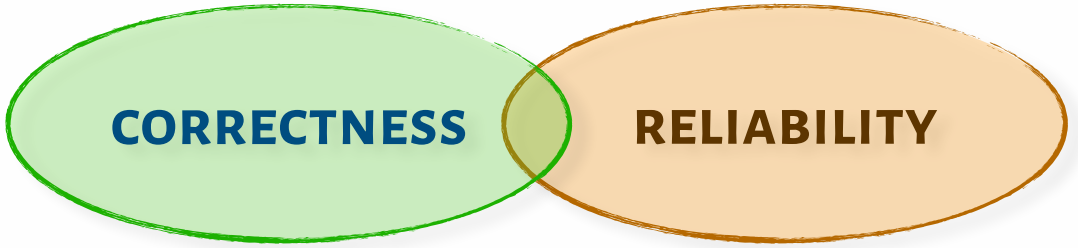






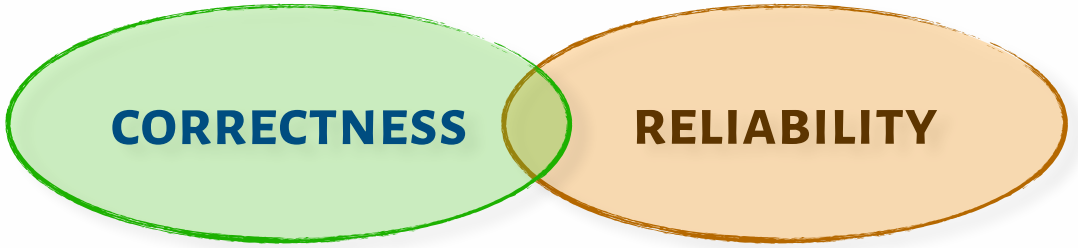
Line/ statement coverage

- Unit testing, esp. TDD



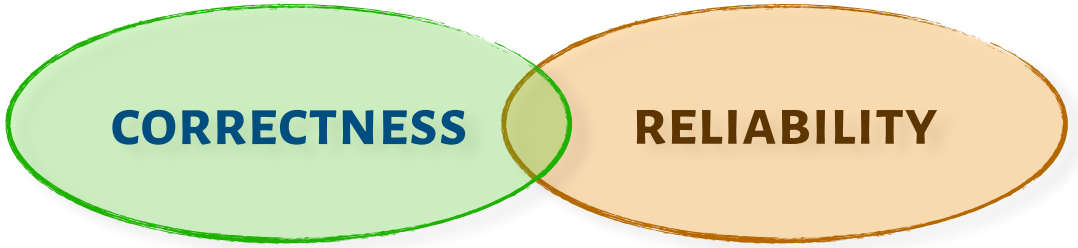
Line/ statement coverage

Data coverage?



Data coverage?

- Property-Based Testing
- Fuzz Testing
- Manual Testing



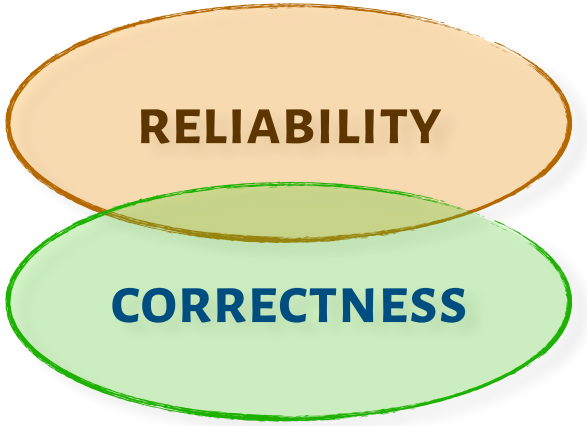
Coverage:

- Line/ statement - TDD
- Data - PBT, Fuzz, manual testing

Error paths (often hidden)

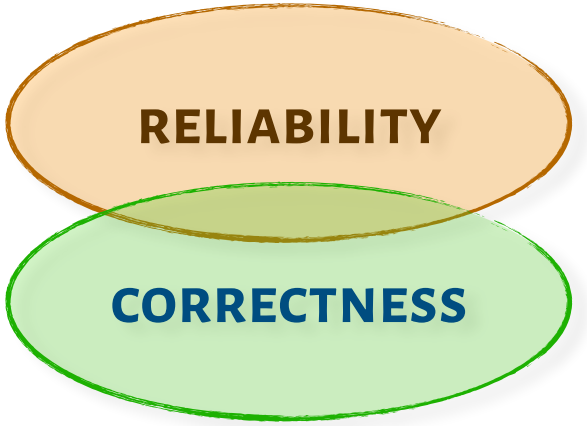
- IO errors/ disappointments
- Logic errors/ contracts

IO errors/ disappointments




```
int str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    return i;
}
```

IO errors/ disappointments

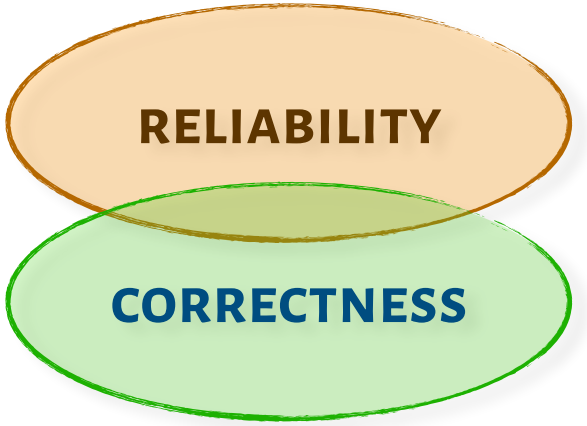



```
int str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    return i;
}
```

```
str_to_int("forty-two")
```

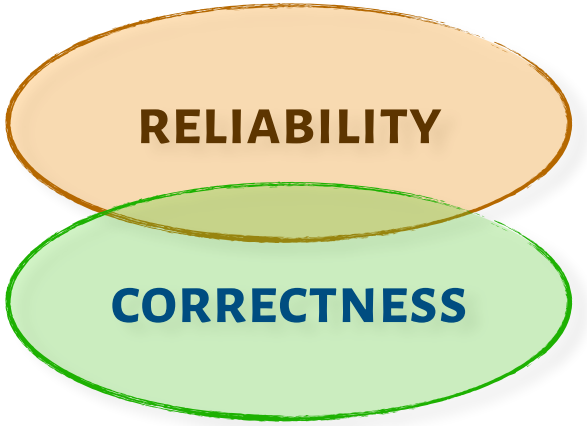
→ **0**

IO errors/ disappointments



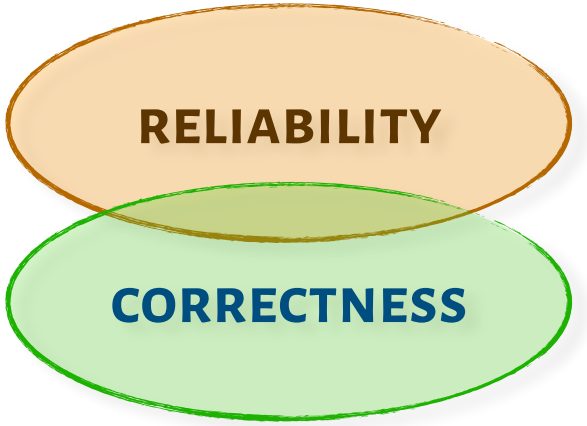

```
int str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    if( ss.good() )
        return i;
    else
        throw std::domain_error("Not a number");
}
```

IO errors/ disappointments




```
std::optional<int> str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    if( ss.good() )
        return i;
    else
        return std::nullopt;
}
```

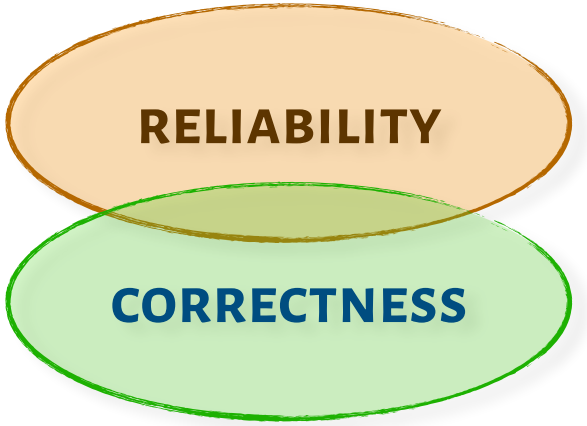
IO errors/ disappointments




```
std::optional<int> str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    if( ss.good() )
        return i;
    else
        return std::nullopt;
}
```

```
if( auto oi = str_to_int("forty-two") )
    std::cout << *oi << "\n";
else
    std::cout << "<not a number>\n";
```

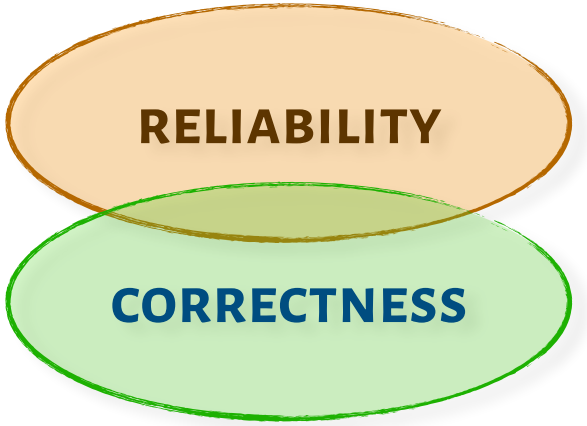
IO errors/ disappointments




```
std::optional<int> str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    if( ss.good() )
        return i;
    else
        return std::nullopt;
}
```

```
if( auto oi = str_to_int("forty-two") )
    std::cout << *oi << "\n";
else
    std::cout << "<not a number>\n";
```

IO errors/ disappointments

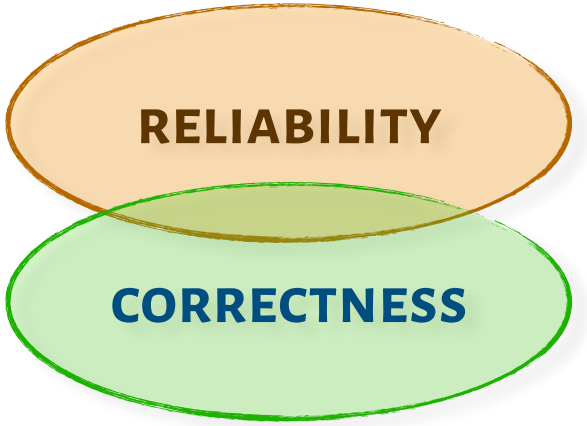



```
std::optional<int> str_to_int( std::string_view str ) {
    int i;
    std::stringstream ss;
    ss << str;
    ss >> i;
    if( ss.good() )
        return i;
    else
        return std::nullopt;
}
```

C++23

```
str_to_int("forty-two")
    .map([](int i){ std::cout << i << "\n"; })
    .or_else([]   { std::cout << "<not a number>\n"; });
```

IO errors/ disappointments

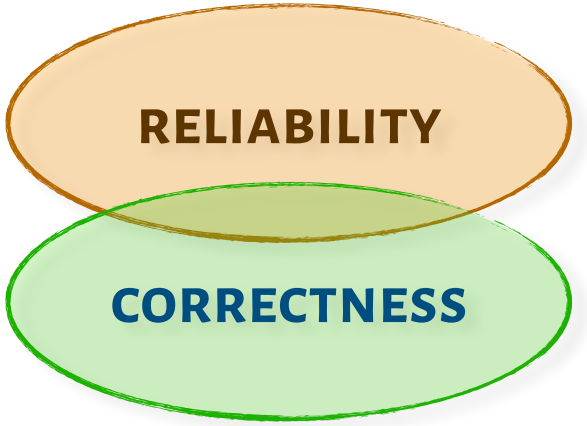


- Exceptions
- Error codes (with [[nodiscard]])
- std::optional
- std::error_code
- boost::outcome



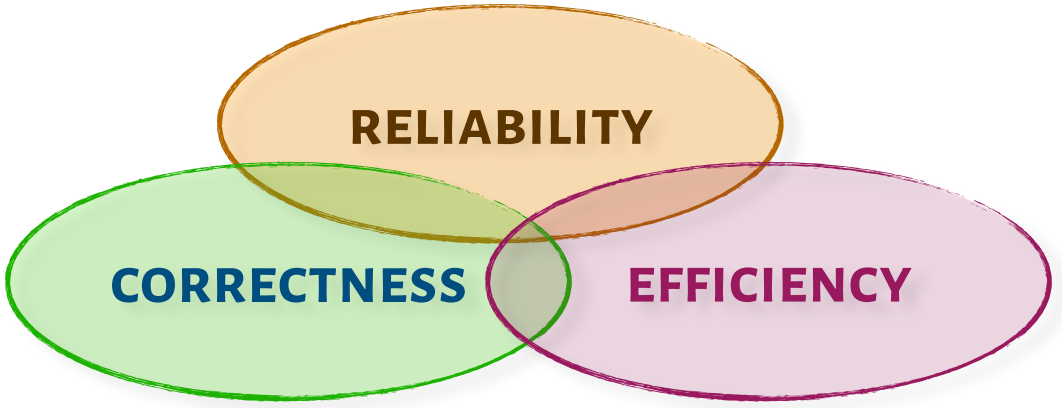
future?

- std::expected
- “static exceptions” (p0709)



Error paths

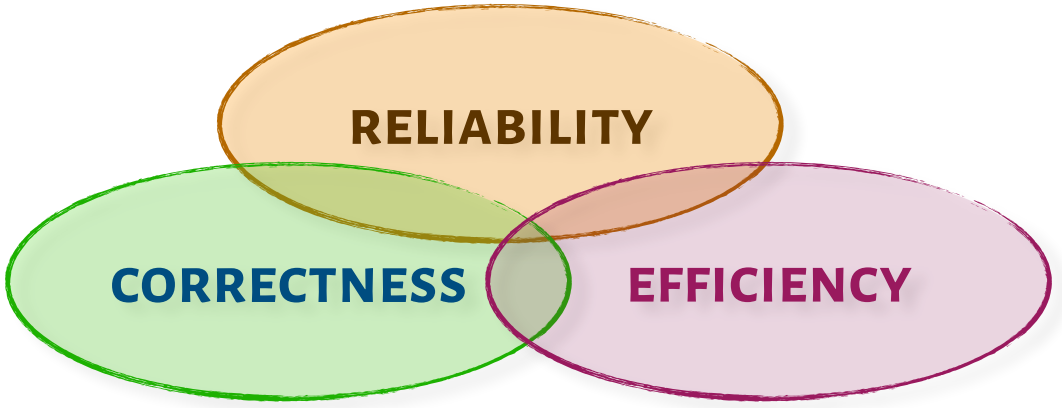
- IO errors/ disappointments
- Logic errors/ contracts



Error paths

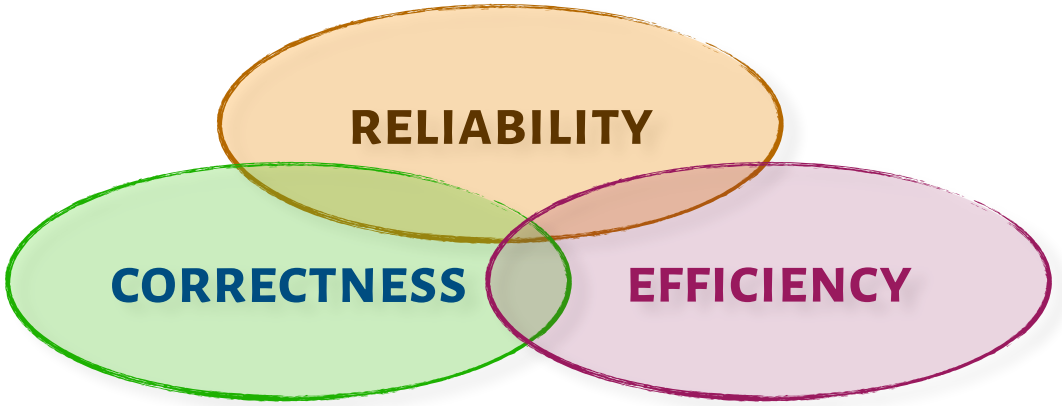
- IO errors/ disappointments
- Logic errors/ contracts

Logic errors/ contracts




```
std::string month_as_string( int month );
```

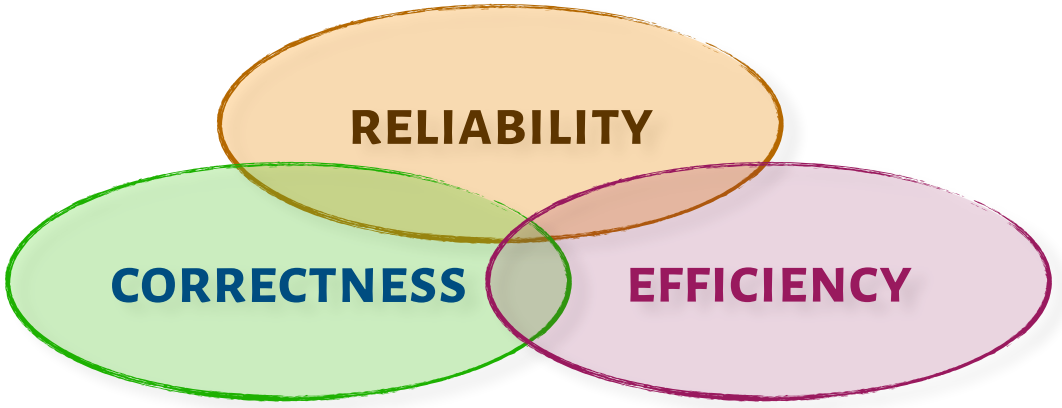
Logic errors/ contracts



range

```
std::string month_as_string( int month );
```

Logic errors/ contracts

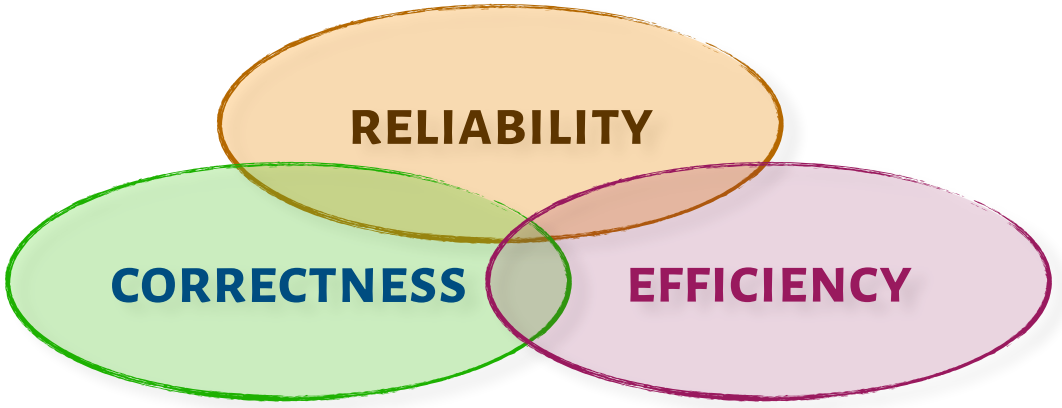


-2,147,483,648 2,147,483,647




```
std::string month_as_string( int month );
```

Logic errors/ contracts



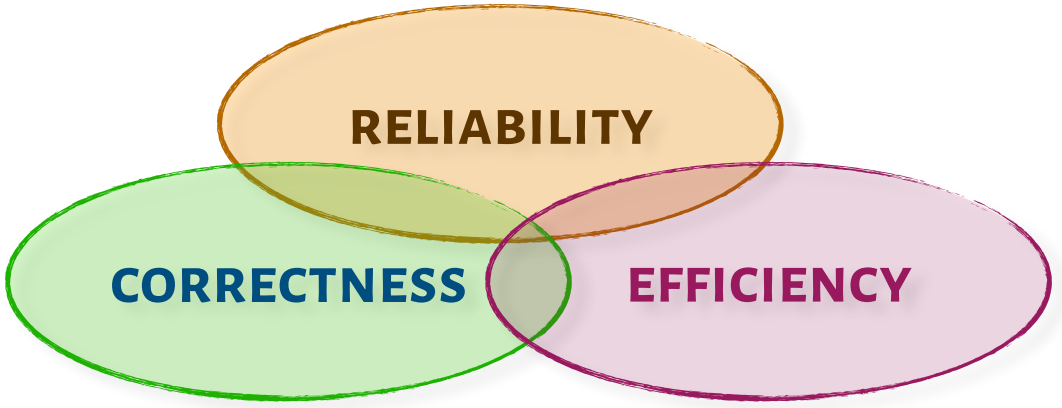
-2,147,483,648 2,147,483,647




```
std::string month_as_string( int month );
```



Logic errors/ contracts



-2,147,483,648 2,147,483,647

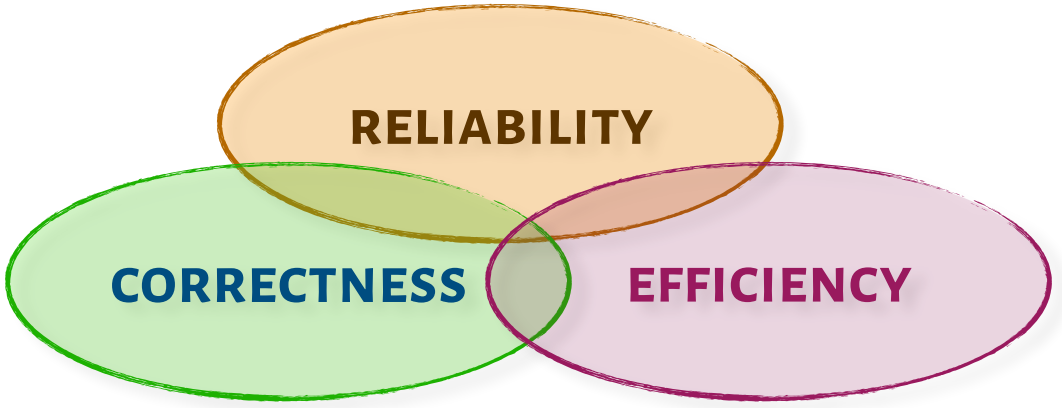
range

```
std::string month_as_string( int month );
```

domain

1 12

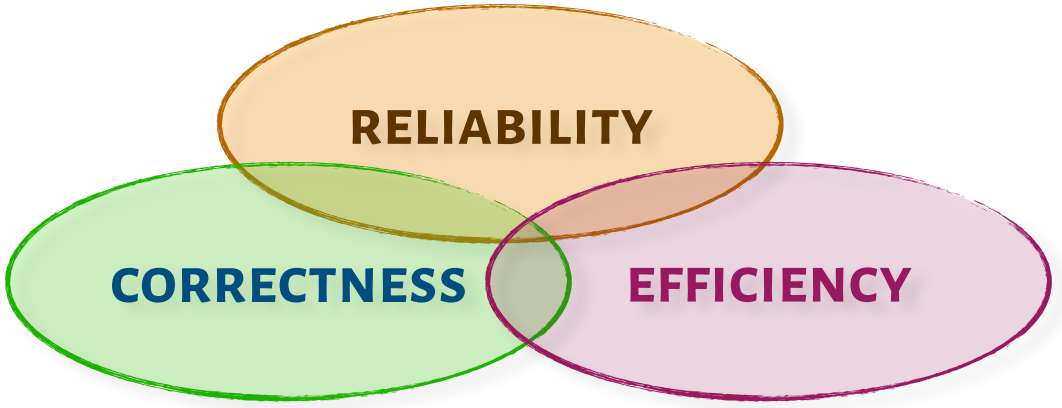
Logic errors/ contracts




```
std::string month_as_string( int month ){
  if( month < 1 || month > 12 )
        throw std::domain_error
          (“Month must be between 1 and 12");

    // ...
}
```

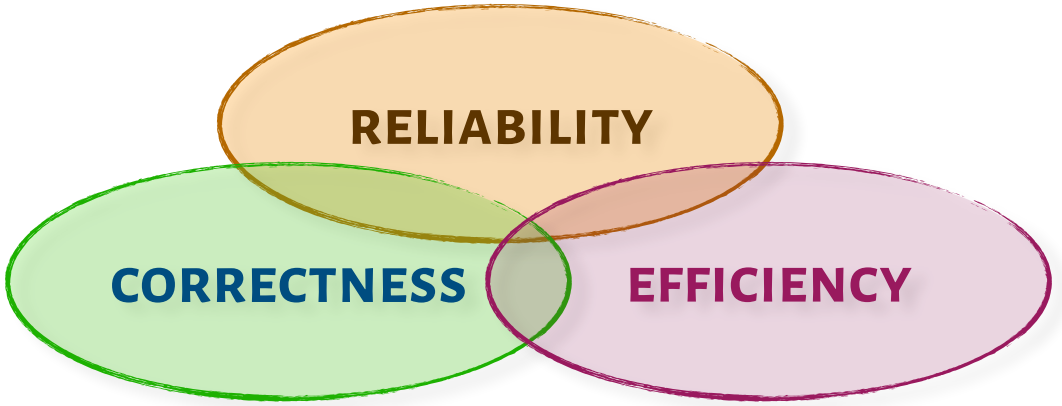
Logic errors/ contracts




```
std::string month_as_string( int month ){
    assert( month >= 1 && month <= 12 );

    // ...
}
```

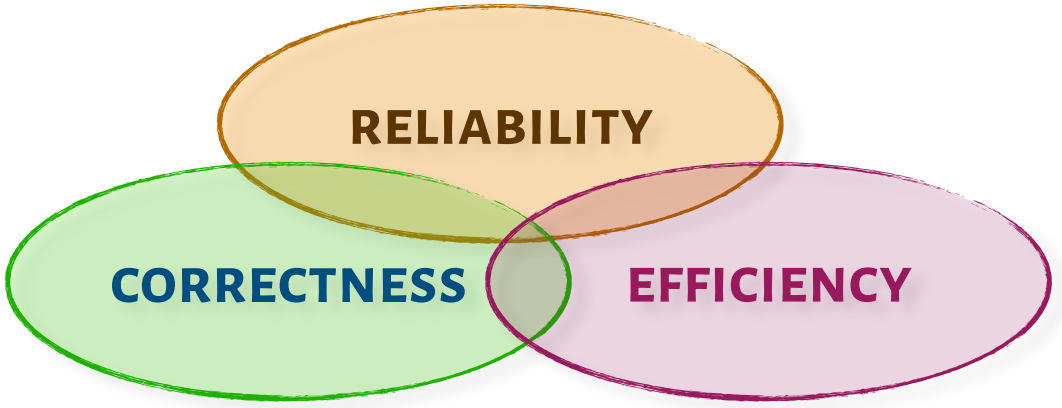
Logic errors/ contracts




```
bool is_valid_month( int month ) {
    return month >= 1 && month <= 12;
}
```

```
std::string month_as_string( int month ){
     assert( is_valid_month(month) );
     // ...
}
```

Logic errors/ contracts

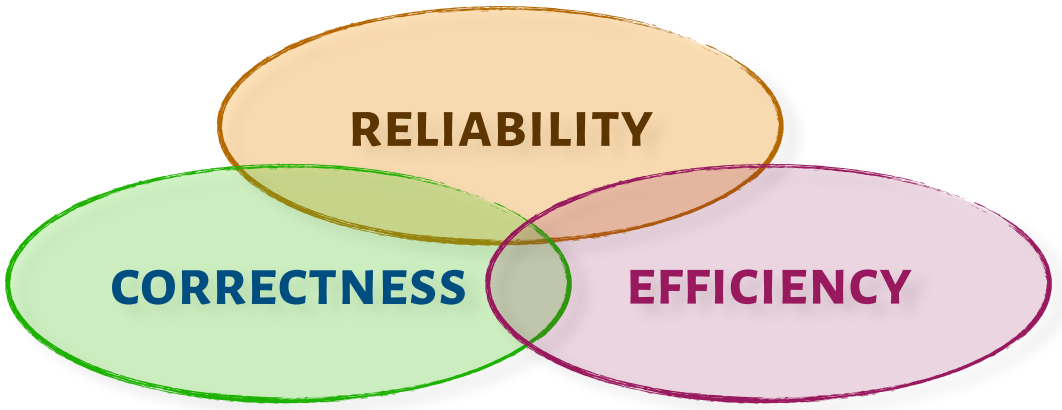



```
bool is_valid_month( int month ) {
    return month >= 1 && month <= 12;
}

std::string month_as_string( int month ){
     assert( is_valid_month(month) );
     // ...
}
```

```
if( is_valid_month(month) ) {
    std::cout << month_as_string(1) << "\n";
}
```

Logic errors/ contracts



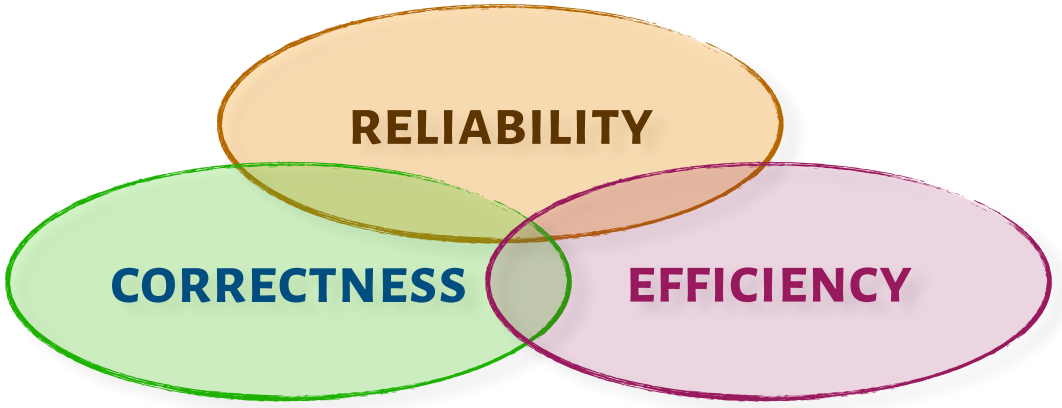

```
bool is_valid_month( int month ) {
    return month >= 1 && month <= 12;
}

std::string month_as_string( int month ){
     assert( is_valid_month(month) );
     // ...
}

 std::string month_as_string_checked( int month ) {
     if( !is_valid_month(month) )
         throw std::domain_error
             ("Month must be between 1 and 12");

     // ...
 }
```

Logic errors/ contracts




```
bool is_valid_month( int month ) {
    return month >= 1 && month <= 12;
}

std::string month_as_string_unchecked( int month ){
     assert( is_valid_month(month) );
     // ...
}

 std::string month_as_string( int month ) {
     if( !is_valid_month(month) )
         throw std::domain_error
             ("Month must be between 1 and 12");

     // ...
}
```

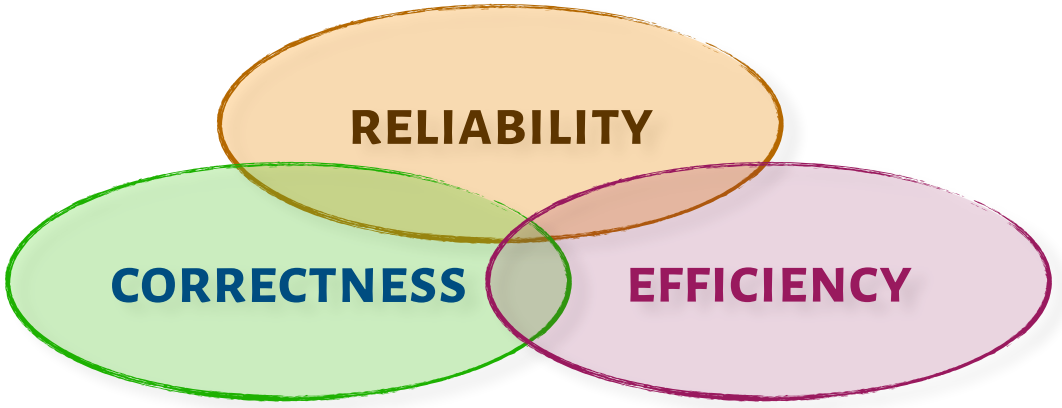
bounded integers in Ada

```
type Day_type   is range    1 ..   31;
type Month_type is range    1 ..   12;
type Year_type  is range 1800 .. 2100;
type Hours is mod 24;
type Weekday is (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday);

type Date is
   record
      Day   : Day_type;
      Month : Month_type;
      Year  : Year_type;
   end record;
```

https://en.wikipedia.org/wiki/Ada_(programming_language)

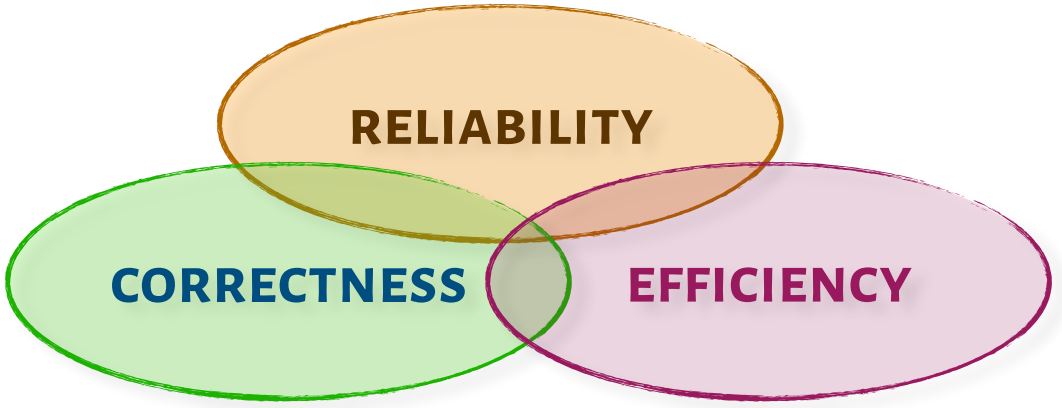
Logic errors/ contracts




```
class Month {
    uint8_t month;
public:
    constexpr Month( uint8_t month ) : month( month ) {
        if( !is_valid_month(month) )
            throw std::domain_error("month must be within 1-12");
    }

    [[nodiscard]] constexpr uint8_t get() const { return month; }
};
```

Logic errors/ contracts

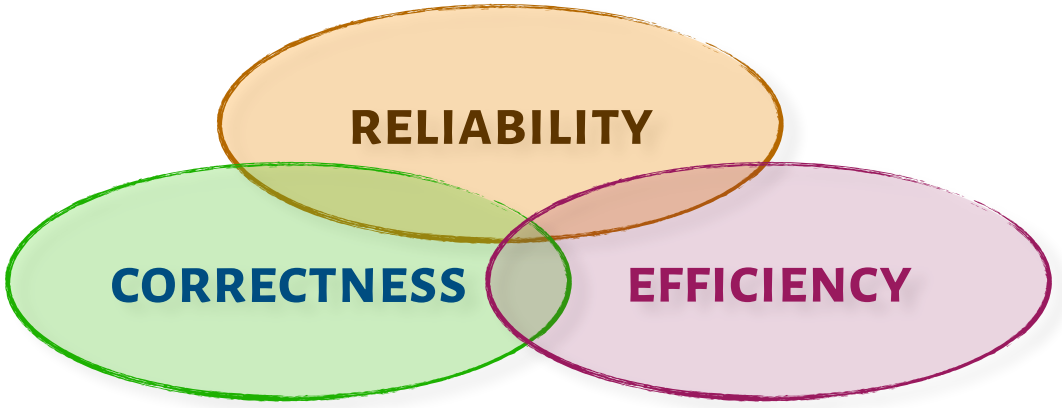



```
class Month {
    uint8_t month;
public:
    constexpr Month( uint8_t month ) : month( month ) {
        if( !is_valid_month(month) )
            throw std::domain_error("month must be within 1-12");
    }

    [[nodiscard]] constexpr uint8_t get() const { return month; }
};


std::string month_as_string( Month month ) {
    // ...
}
```

Logic errors/ contracts

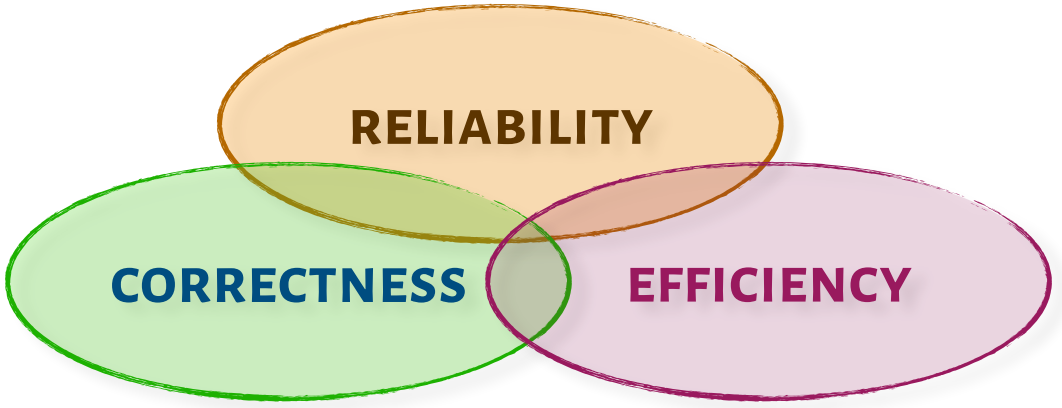



```
class Month {
    uint8_t month;
public:
    constexpr Month( uint8_t month ) : month( month ) {
        if( !is_valid_month(month) )
            throw std::domain_error("month must be within 1-12");
    }

    constexpr Month( uint8_t month, unchecked_tag ) : month( month ) {
        assert( is_valid_month(month) );
    }

    [[nodiscard]] constexpr uint8_t get() const { return month; }
};
```

Logic errors/ contracts

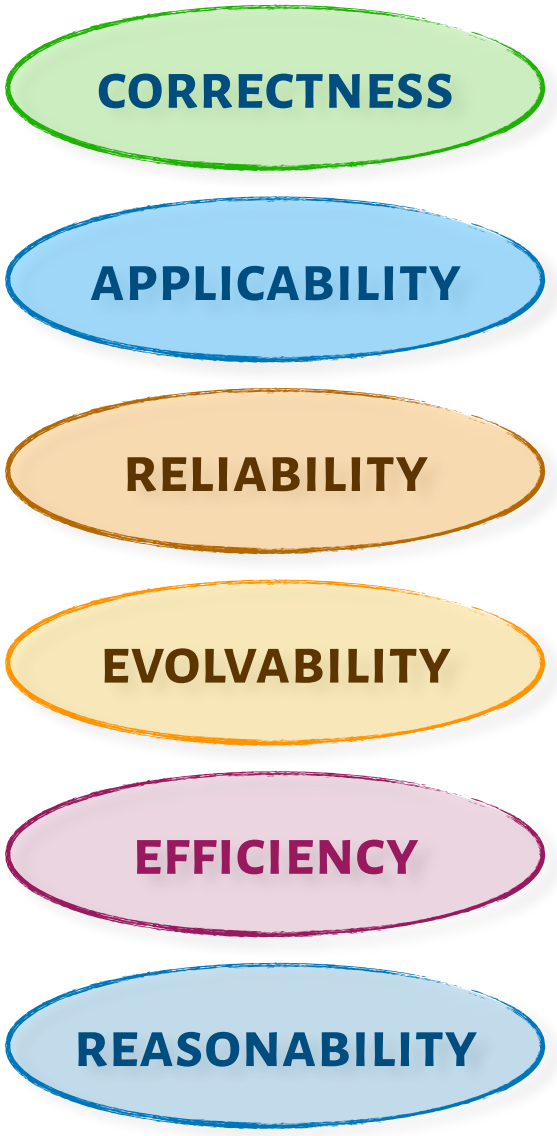


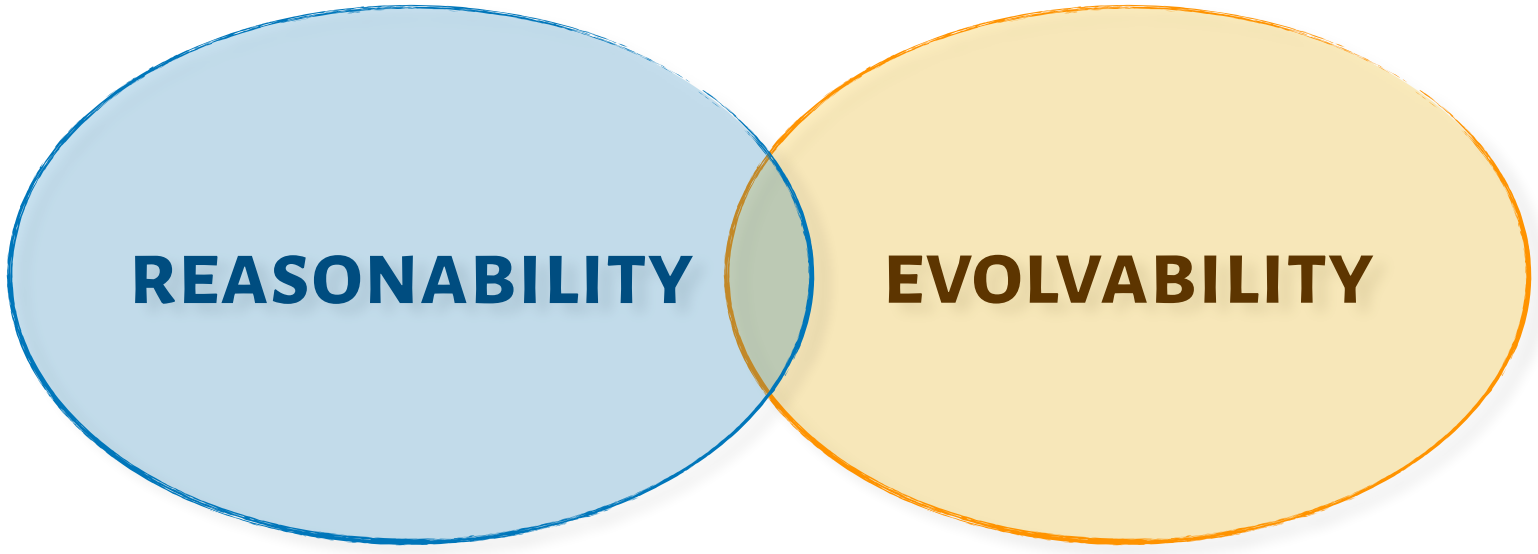

```
class Month {
    uint8_t month;
public:
    constexpr Month( uint8_t month ) : month( month ) {
        if( !is_valid_month(month) )
            throw std::domain_error("month must be within 1-12");
    }

    constexpr Month( uint8_t month, unchecked_tag ) : month( month ) {
        assert( is_valid_month(month) );
    }

    [[nodiscard]] constexpr uint8_t get() const { return month; }
};
```

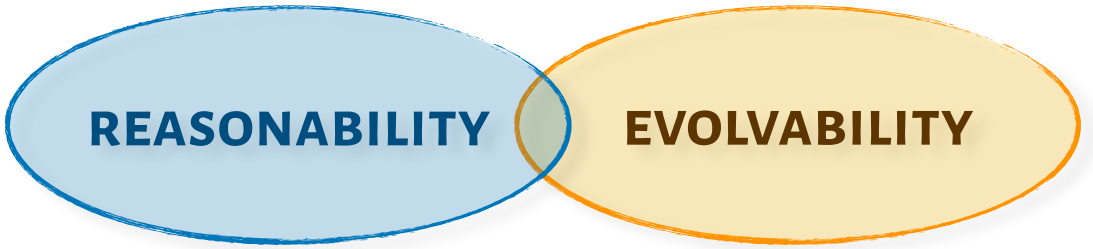
```
Month m1( 13 ); // throws
Month m2( 13, unchecked_tag() ); // asserts
```


CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY


REASONABILITY
EVOLVABILITY

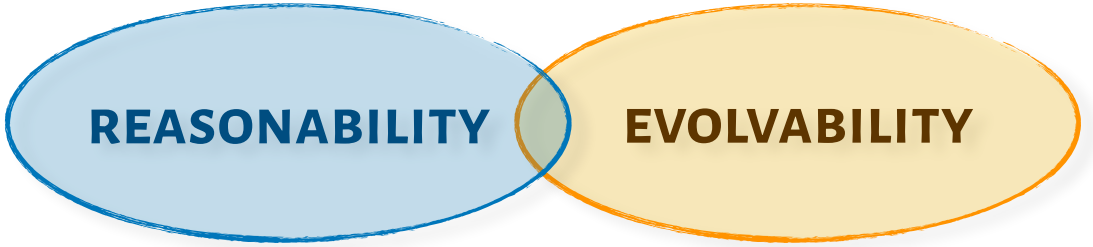
Simplicity

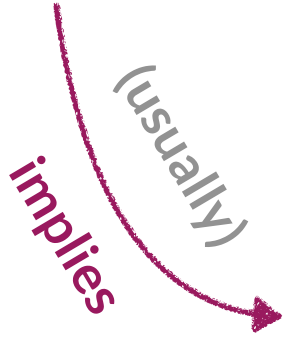

NDC { London }
11-15 January 2016
Inspiring Developers since 2008
seeking
Simplicity


REASONABILITY
EVOLVABILITY

Simplicity







Functional Programming

Immutability **Correct-by-construction**



Simple

One fold/ braid



Complex

Many folds/
braids



Complex

Many folds/
braids



- Threading/ concurrecy
- Globals/ Singletons
- Other shared state
- Mutability
- Cyclomatic complexity
- Leaky abstractions
- Code dependencies
- Module dependencies

lower

Complexity

= easier test

= easier to reason about

= easier to change

TDD + Functional Programming =

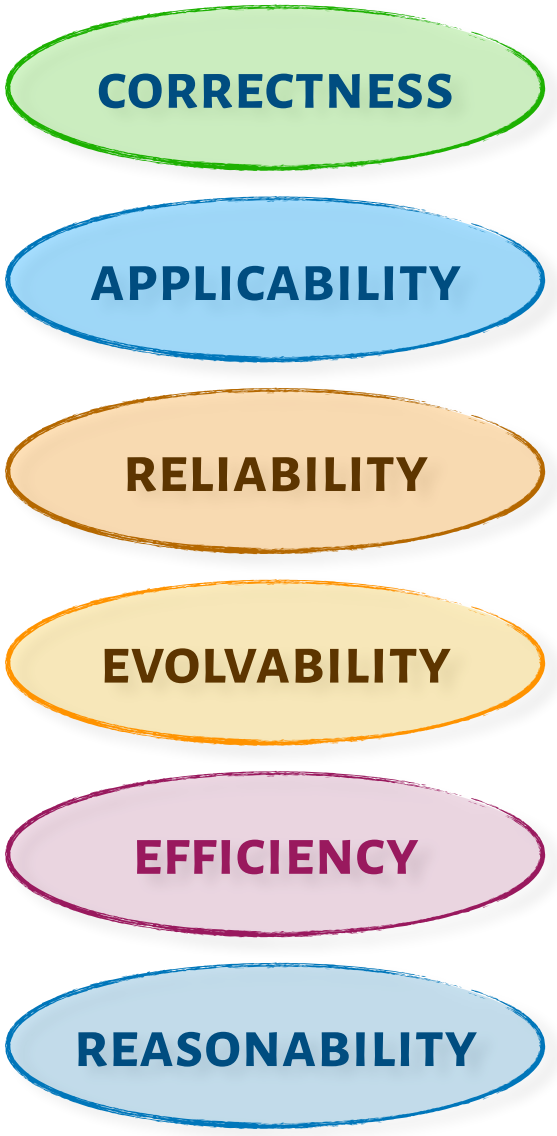
lower

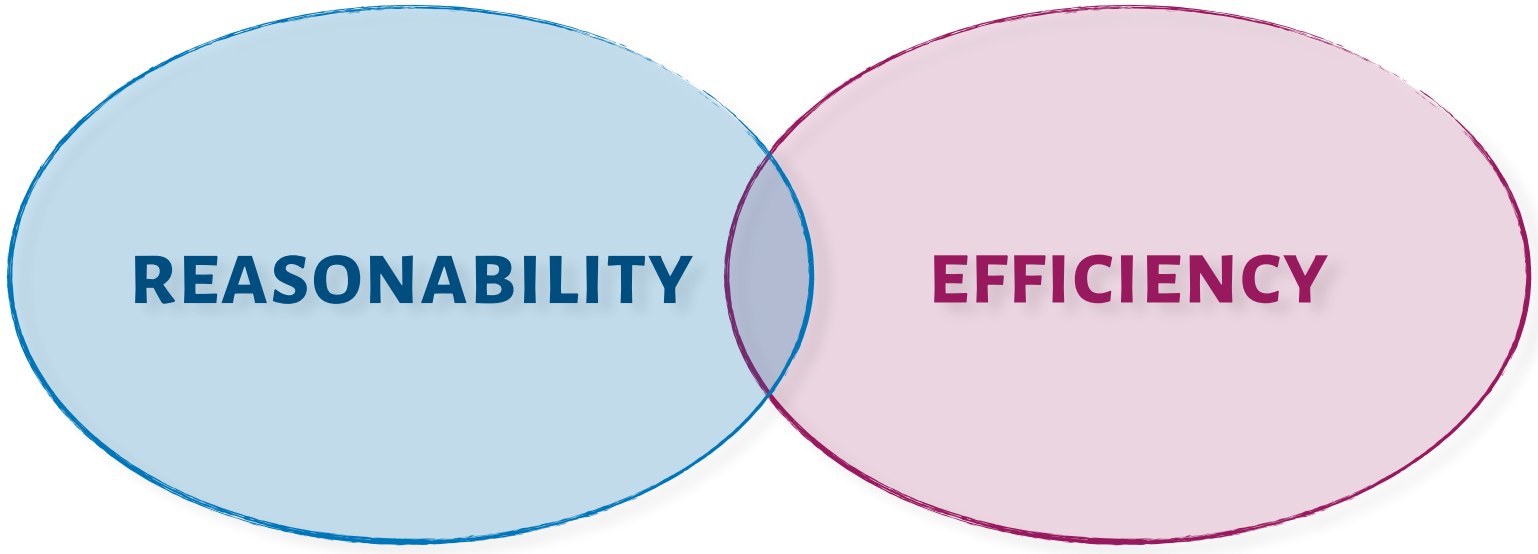
Complexity

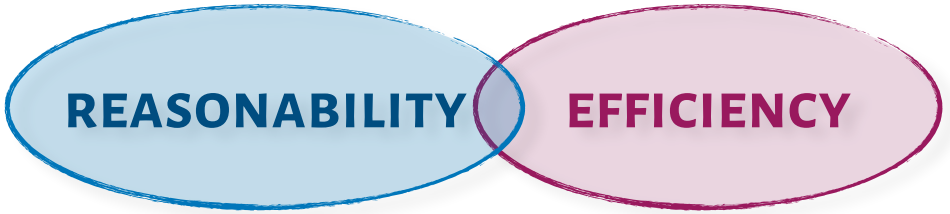
= easier test

= easier to reason about

= easier to change

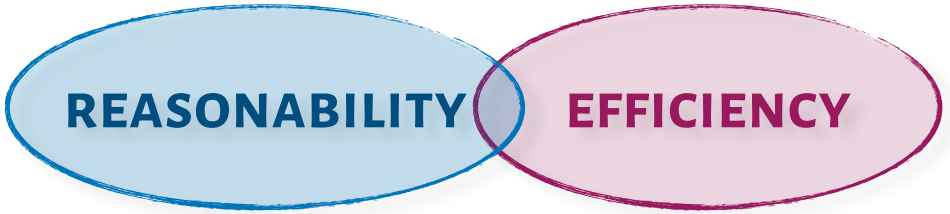

CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY


REASONABILITY
EFFICIENCY



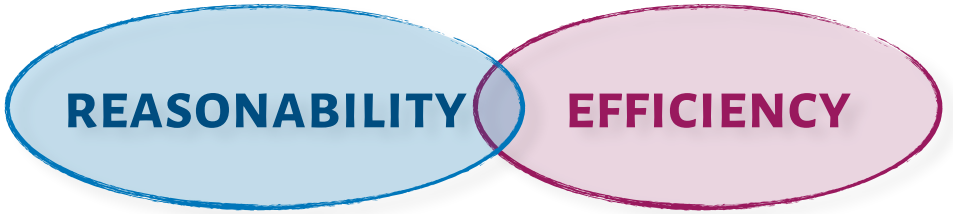

```
Colour colour;
if( sunnySideUp() ) {
    colour = Colour::Yellow;
}
else {
    colour = Colour::White;
}

// ...
```

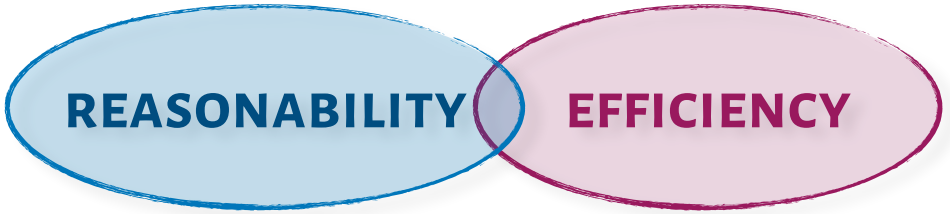



```
const auto colour =
    sunnySideUp()
        ? Colour::Yellow
        : Colour::White;

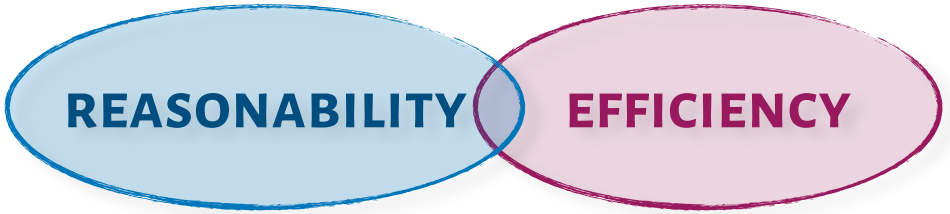
// ...
```




```
State state;
switch( random_colour() ) {
     case Colour::Red:
          state = State::Stopped;
          break;
     case Colour::Amber:
          state = State::GetReady;
          break;
     case Colour::Green:
          state = State::Driving;
          break;
}
```




```
const auto state = [] {
    switch( random_colour() ) {
        case Colour::Red:   return State::Stopped;
        case Colour::Amber: return State::GetReady;
        case Colour::Green: return State::Driving;
    }
}();
```

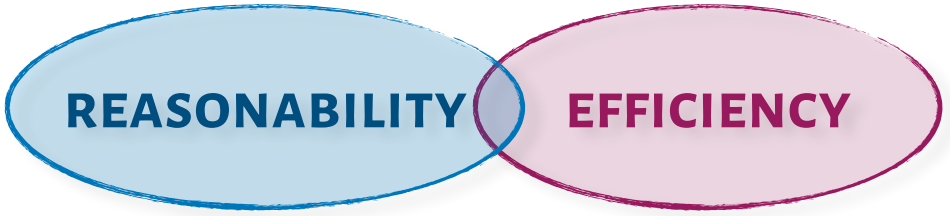



```
class Expensive {
    std::vector<char> data;
public:
    Expensive() { data.reserve(64); }
    Expensive( std::string_view text ) {
        data.reserve( text.length() );
        std::copy( text.begin(),
                   text.end(),
                   std::back_inserter( data ) );
    }

    size_t size() const { return data.size(); }
};
```

```
class Expensive {
    std::vector<char> data;
public:
    Expensive() { data.reserve(64); }
    Expensive( std::string_view text ) {
        data.reserve( text.length() );
        std::copy( text.begin(),
                   text.end(),
                   std::back_inserter( data ) );
    }

    size_t size() const { return data.size(); }
};
```




```
auto const object = []() -> Expensive {
    switch( random_colour() ) {
        case Colour::Red:
            return { "Red traffic light - stop!" };
        case Colour::Amber:
            return { "Amber traffic light - get ready..." };
        case Colour::Green:
            return { "Green traffic light - go!" };
    }
}();
```



Testing

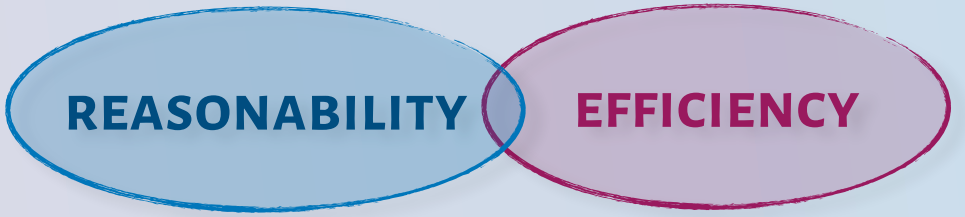




Property-Based Testing



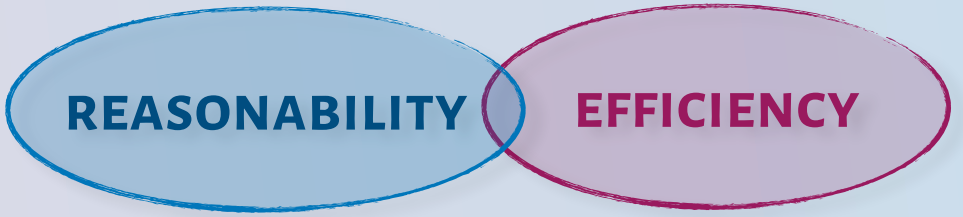
Property-Based Testing




```
int slow_but_easy_algorithm(int i1, int i2);

int fast_but_tricky_algorithm(int i1, int i2);
```

Property-Based Testing

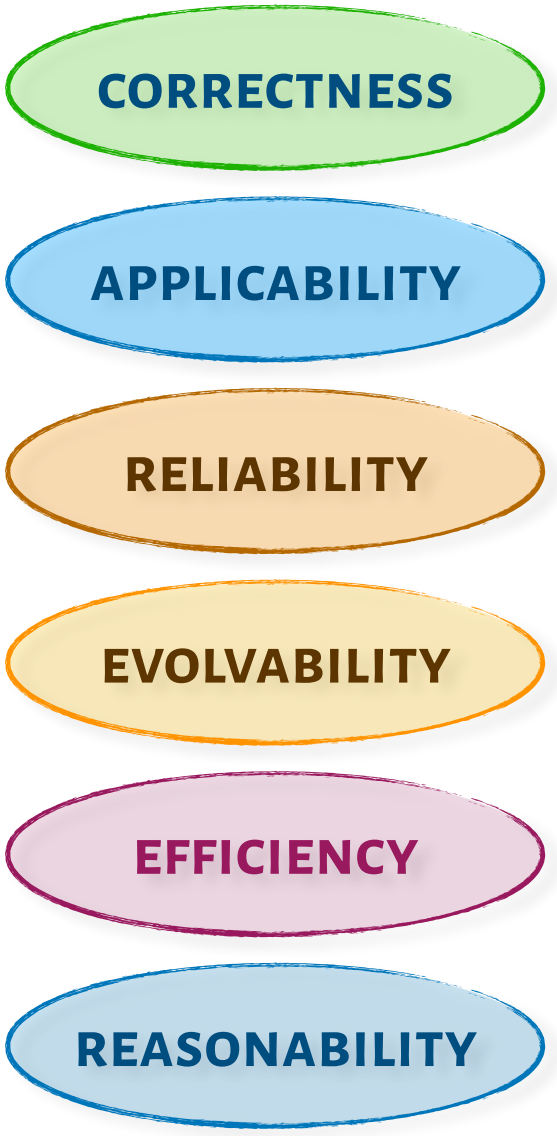


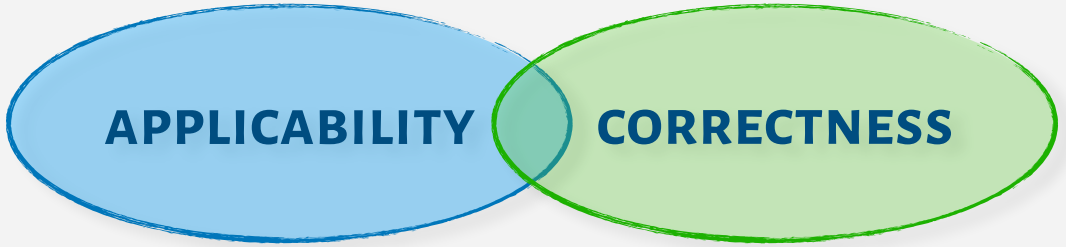

```
int slow_but_easy_algorithm(int i1, int i2);

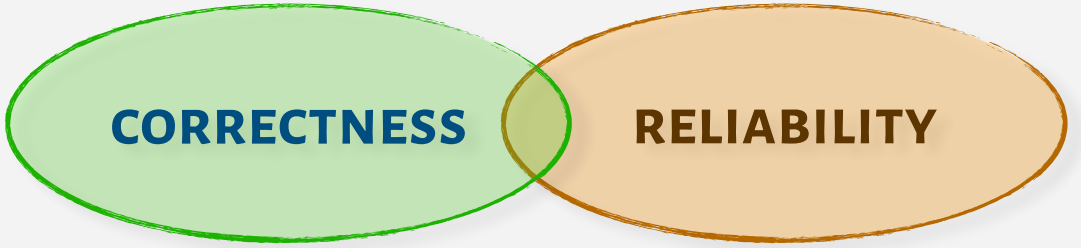
int fast_but_tricky_algorithm(int i1, int i2);
```

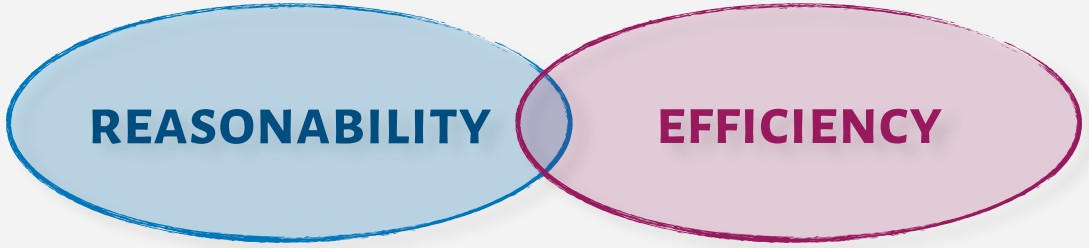
```
int i1 = GENERATE( take(100, random(INT_MIN, INT_MAX )));
int i2 = GENERATE( take(100, random(INT_MIN, INT_MAX )));

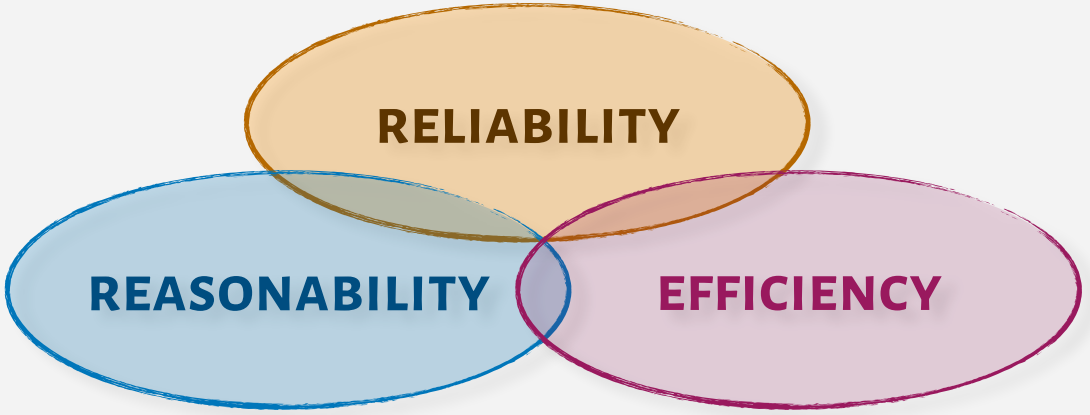
REQUIRE( slow_but_easy_algorithm(i1, i2)
      == fast_but_tricky_algorithm(i1, i2) );
```

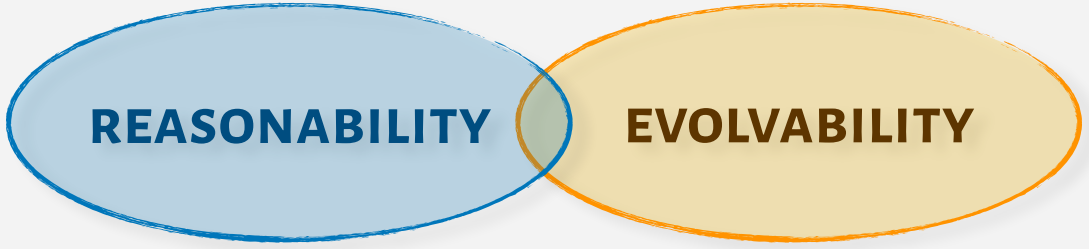

CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY

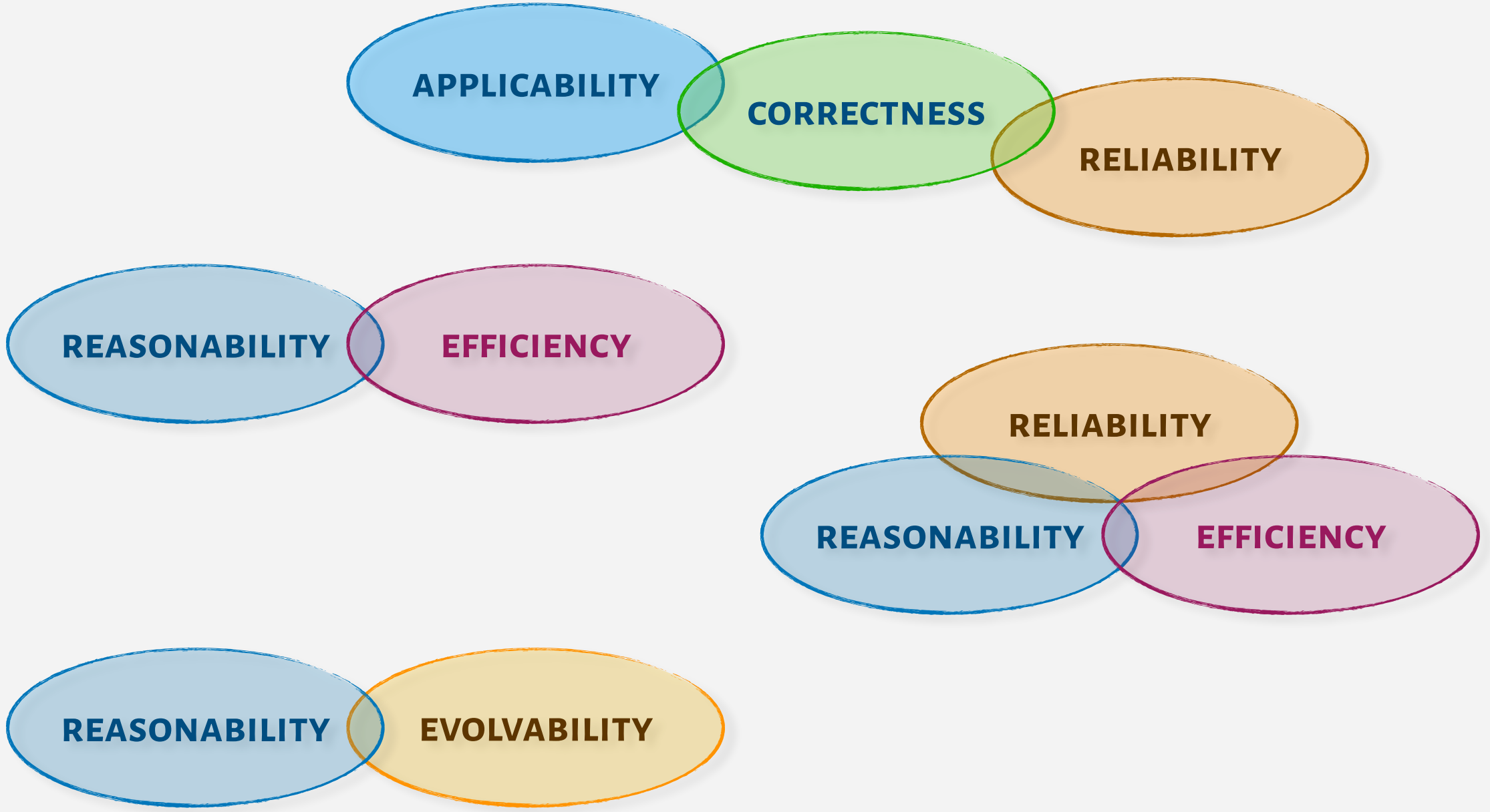

APPLICABILITY
CORRECTNESS

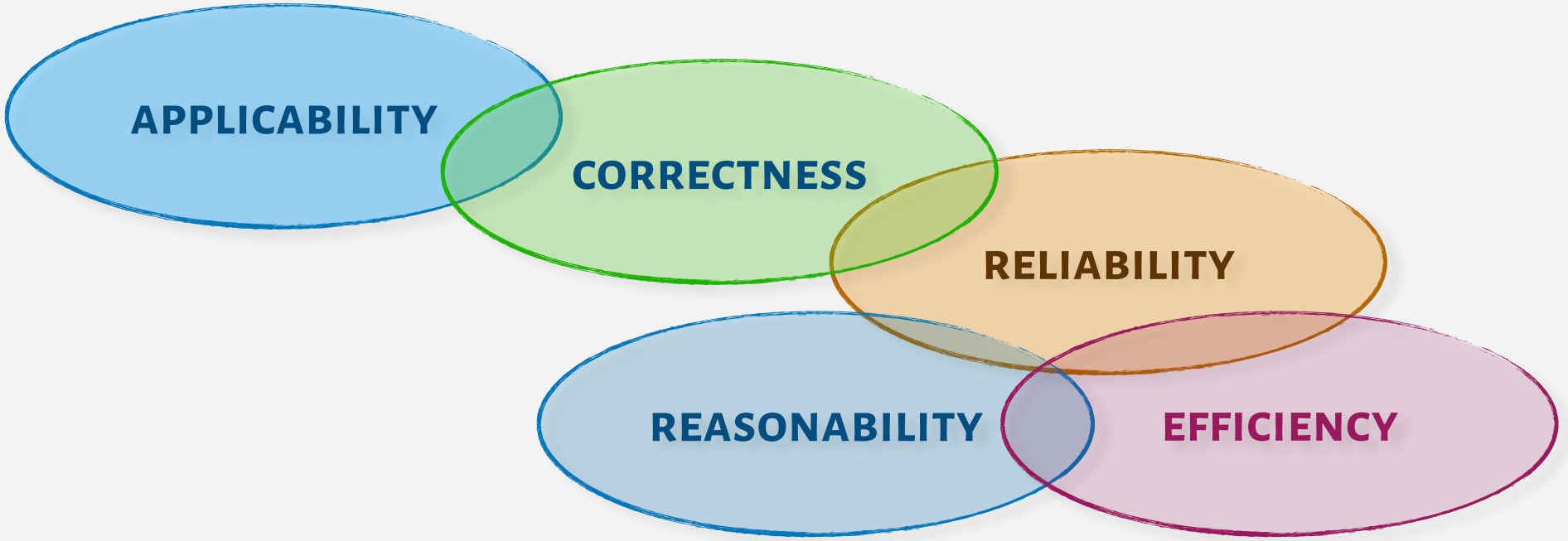

CORRECTNESS
RELIABILITY

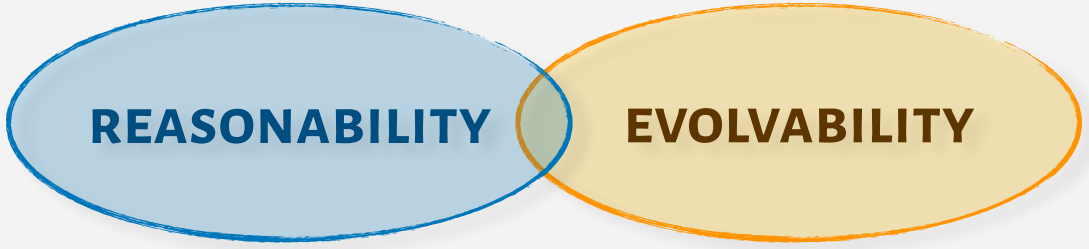

REASONABILITY
EFFICIENCY

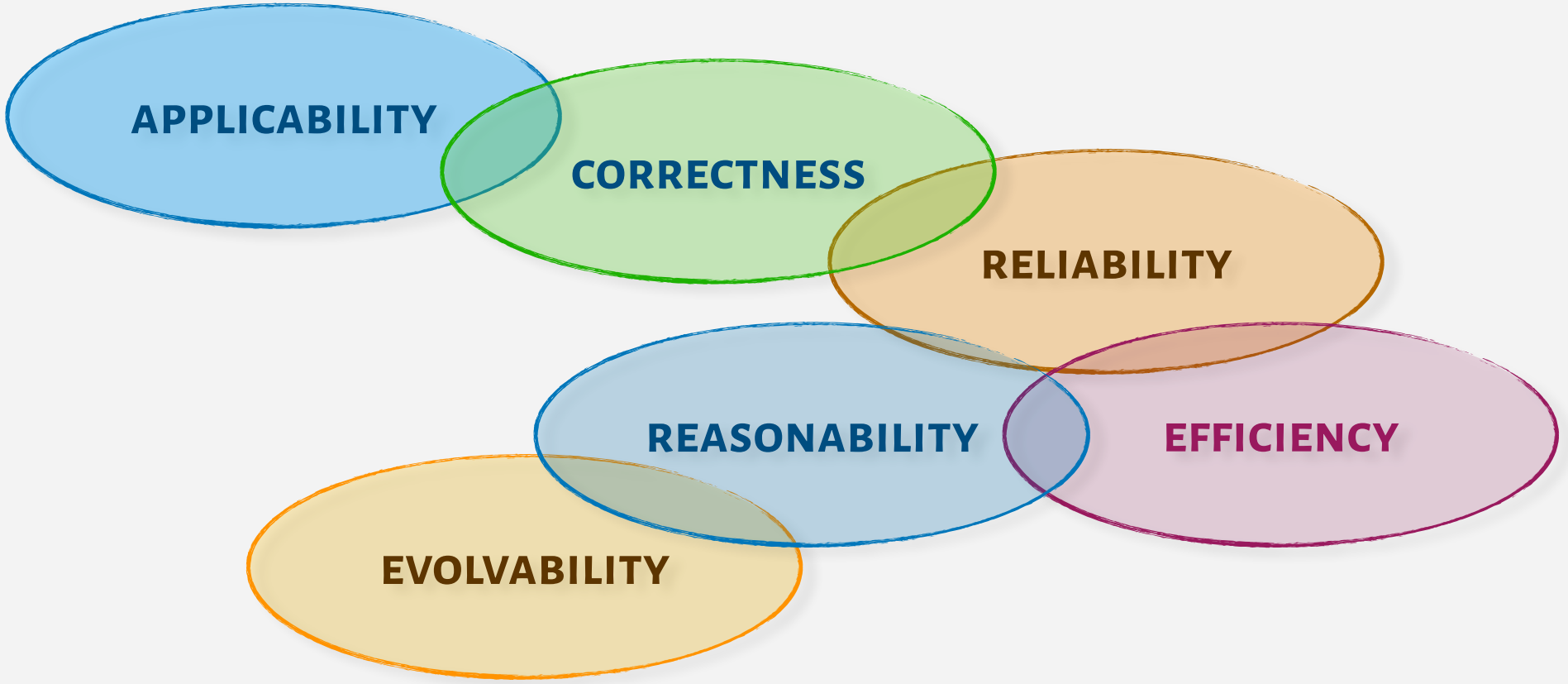

RELIABILITY
REASONABILITY
EFFICIENCY

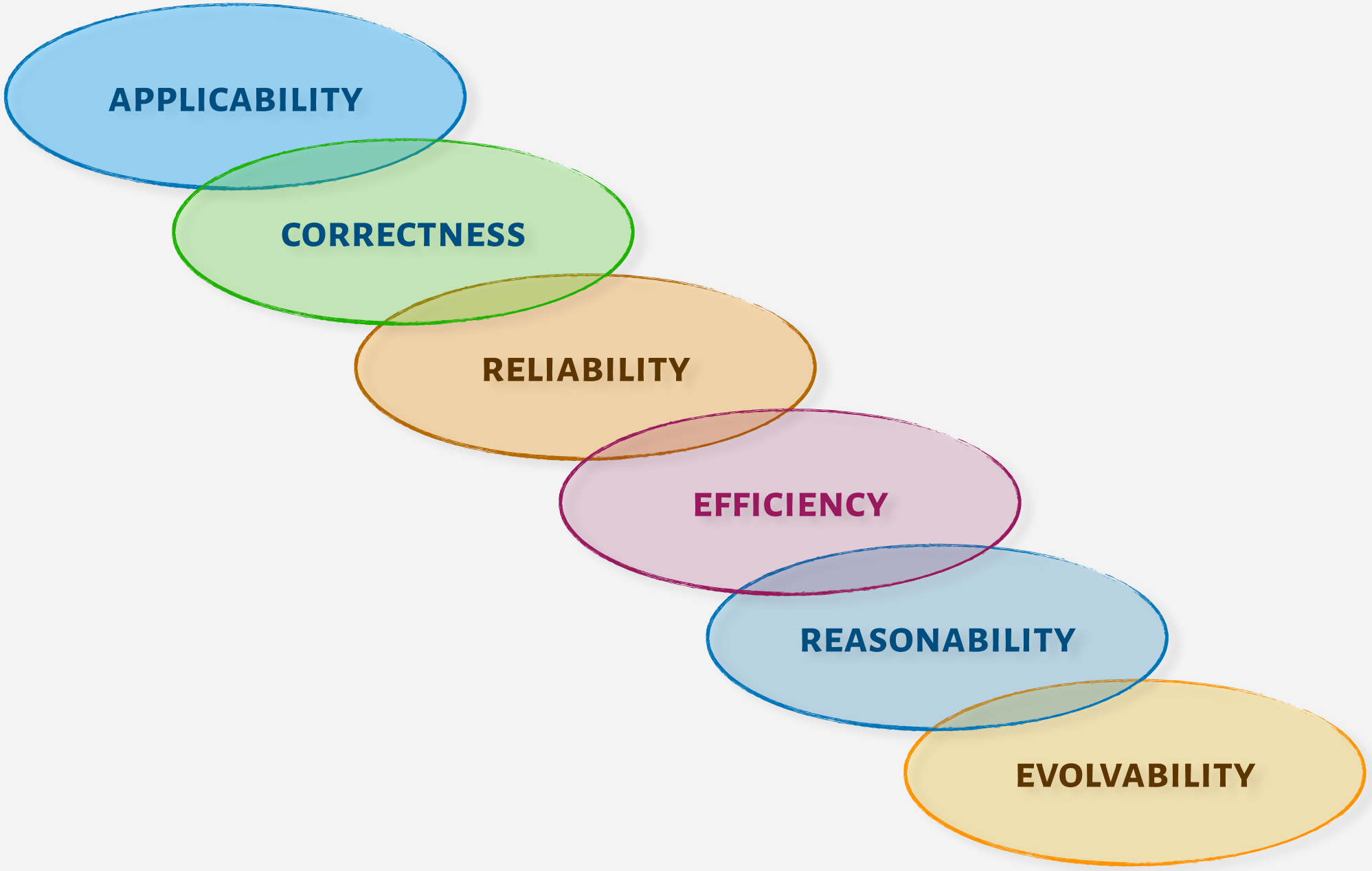

REASONABILITY
EVOLVABILITY

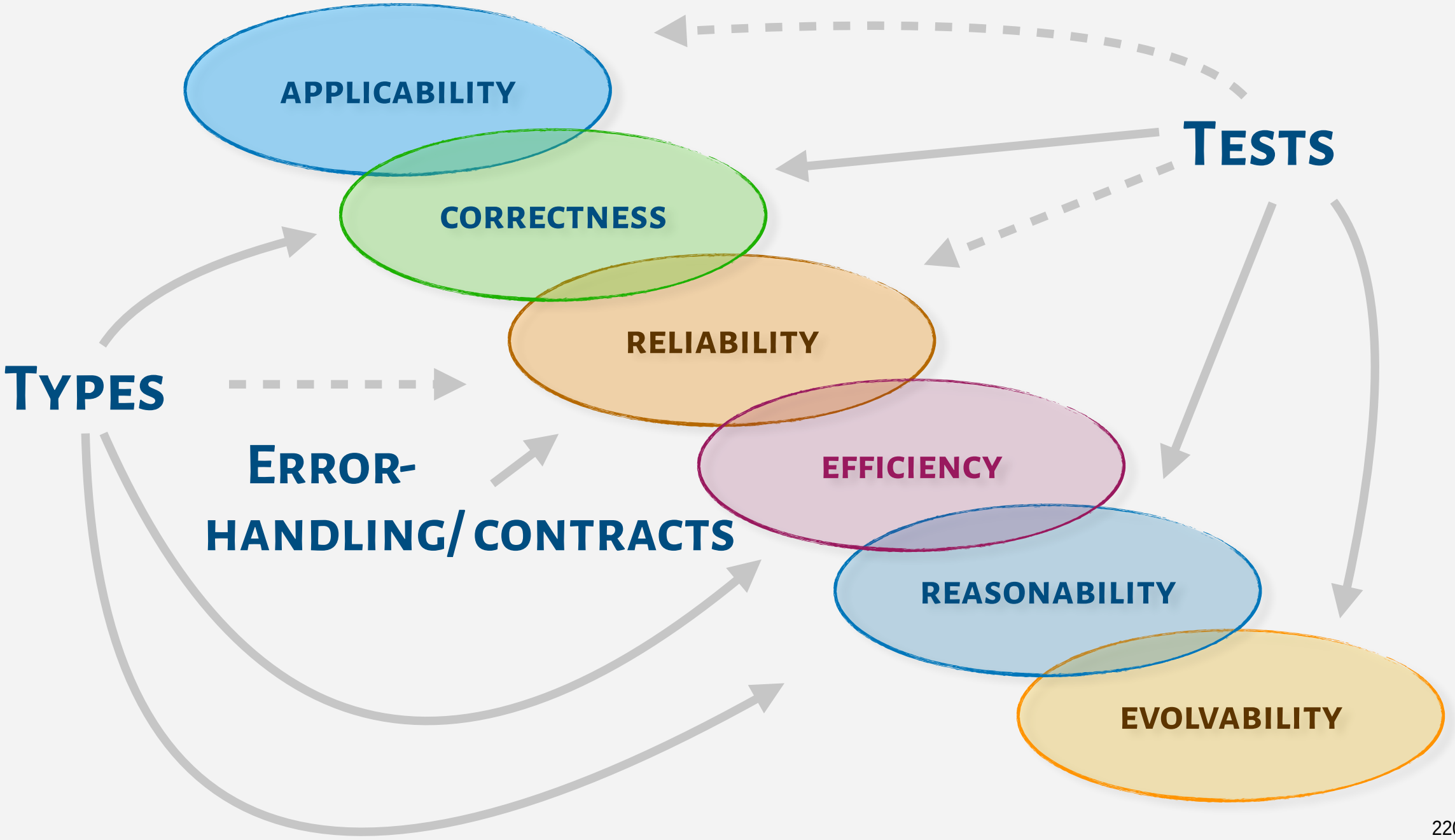

APPLICABILITY
CORRECTNESS
RELIABILITY
REASONABILITY
EFFICIENCY
RELIABILITY
REASONABILITY
EFFICIENCY
REASONABILITY
EVOLVABILITY

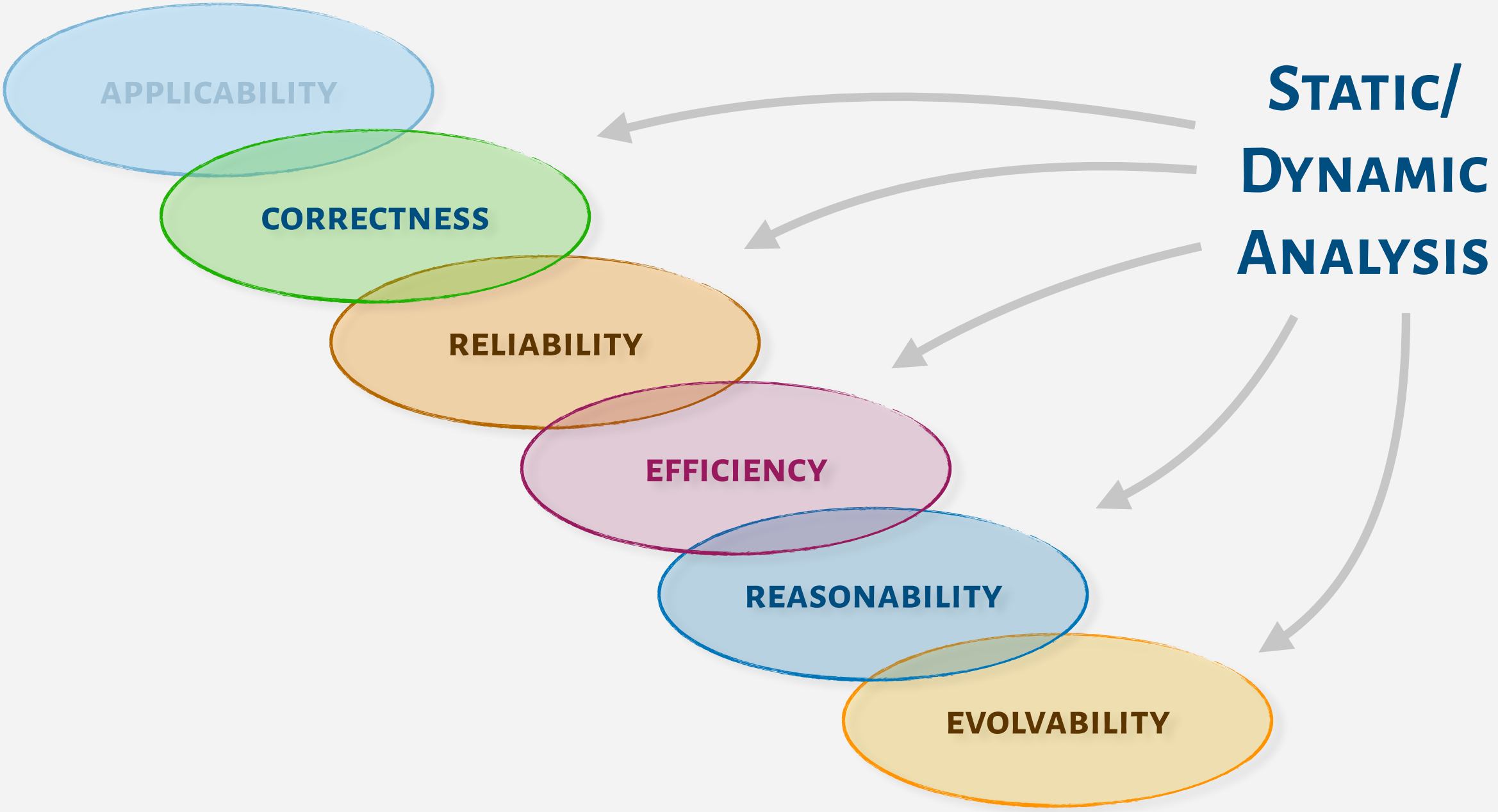

APPLICABILITY
CORRECTNESS
RELIABILITY
REASONABILITY
EFFICIENCY


REASONABILITY
EVOLVABILITY


APPLICABILITY
CORRECTNESS
RELIABILITY
REASONABILITY
EFFICIENCY
EVOLVABILITY


APPLICABILITY
CORRECTNESS
RELIABILITY
EFFICIENCY
REASONABILITY
EVOLVABILITY


APPLICABILITY
CORRECTNESS
RELIABILITY
EFFICIENCY
REASONABILITY
EVOLVABILITY
TESTS
TYPES
ERROR-HANDLING/ CONTRACTS


APPLICABILITY
CORRECTNESS
RELIABILITY
EFFICIENCY
REASONABILITY
EVOLVABILITY
Static/
Dynamic
Analysis

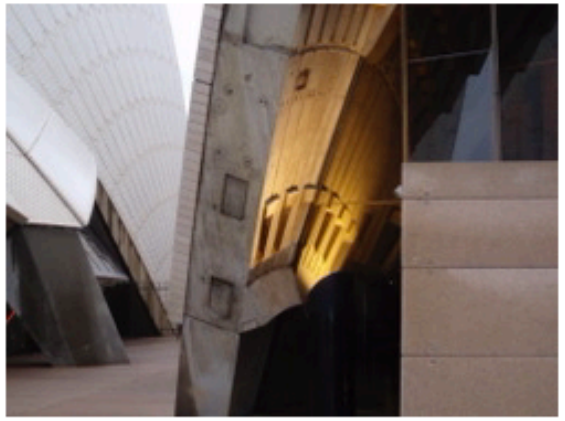
CODA

Michael Feathers

« Are FP and OO Incompatible Syntactic Styles? | Main | Are Nested Classes Really a Good Idea? »

June 12, 2008

The Flawed Theory Behind Unit Testing



I have Google's blogsearch set to give me notifications about unit testing. On an average week, I read dozens of blogs and mailing list discussions about the topic. Occasionally, I read something new, but there's lot of repetition out there. The same arguments crop up often. Of all of them, though, there is one argument about unit testing which really bugs me because it rests on a flawed theory about testing and quality and sadly, it's an argument that I fell for a long time ago and I'd like to lay it to rest. Hopefully, this blog will help, but I have to relate a little history first.

Back in the very early 2000s, I had a conversation with Steve Freeman at a conference. We were talking about Test-Driven Development and Steve had the strong feeling that most of the people who were practicing TDD at the time were doing it wrong - they'd missed something.

Steve was and is part of a close-knit community in London who have been practicing XP and TDD from the very beginning. Among the fruits of their labor was the entire notion of mock objects. Steve Freeman and Tim MacKinnon wrote the paper that introduced the idea to the broader community. The rest is history. There are mock object frameworks out there for nearly every language in common use.

Mock Objects, however, are part of a larger relatively unpublicized approach to TDD. The story I heard was that it was all started by John Nolan, the CTO of a startup named Connextra. John Nolan, gave his developers a challenge: write OO code with no getters. Whenever possible, tell another object to do something rather than ask. In the process of doing this, they noticed that their code became supple and easy to change. They also noticed that the fake objects that they were writing were highly repetitive, so they came up with the idea of a mocking framework that would allow them to set expectations on objects - mock objects.

When Steve told me about this approach, I thought it sounded okay but I there was one thing that I couldn't wrap my head around - Steve, and Tim, and the people who had been on that team were using mocks extensively. In fact, they used mocks whenever they could. This was a bit different from the way that I was practicing TDD. What I did, in general, was use tests to drive a class and then I extracted new classes from the class I was designing as it became

Superficially, this makes sense. Tests can pass or fail and when they fail we learn that we have a problem and we correct it. If you subscribe to this theory, you expect to find fewer integration errors when you do integration testing and fewer "unit" errors when you do unit testing. It's a nice theory, but it's wrong. The best way to see this is to compare unit testing to another way of improving quality - one that has a very dramatic measurable effect.

Back in the 1980s, there was a movement to use something called Clean Room Software Development. The notion behind Clean Room was that you could increase quality by increasing the rigor of development. In Clean Room, you had to write a logical predicate for every little piece of your code and you had to demonstrate, during a review, that your code did no more or less than the predicate described. It was a very serious approach, and it was a bit more radical than what I just described: another tenet of Clean Room was that there was to be no unit testing. None. Zilch. When you wrote your code it was assumed correct after it was reviewed. The only testing that was done was stochastic testing at the functional level.

Amazingly, Clean Room worked. Clean Room teams demonstrated very high quality numbers. When I read about it, I was stunned, but then I came across a passage in a book that I was reading about the process. The author said that many programmers wrote their predicates after writing a section of code, but that experienced programmers often wrote the predicates first.. Gee, that sounds familiar, doesn't it? In TDD, we write the test first and the test is, essentially, a specification of the behavior of the code we are about to write.

In the software industry, we've been chasing quality for years. The interesting thing is there are a number of things that work. Design by Contract works. Test Driven Development works. So do Clean Room, code inspections and the use of higher-level languages.

All of these techniques have been shown to increase quality. And, if we look closely we can see why: all of them force us to reflect on our code.

That's the magic, and it's why unit testing works also. When you write unit tests, TDD-style or after your development, you scrutinize, you think, and often you prevent problems without even encountering a test failure.

compare unit testing to another way of improving quality - one that has a very dramatic measurable effect.

Back in the 1980s, there was a movement to use something called Clean Room Software Development. The notion behind Clean Room was that you could increase quality by increasing the rigor of development. In Clean Room, you had to write a logical predicate for every little piece of your code and you had to demonstrate, during a review, that your code did no more or less than the predicate described. It was a very serious approach, and it was a bit more radical than what I just described: another tenet of Clean Room was that there was to be no unit testing. None. Zilch. When you wrote your code it was assumed correct after it was reviewed. The only testing that was done was stochastic testing at the functional level.

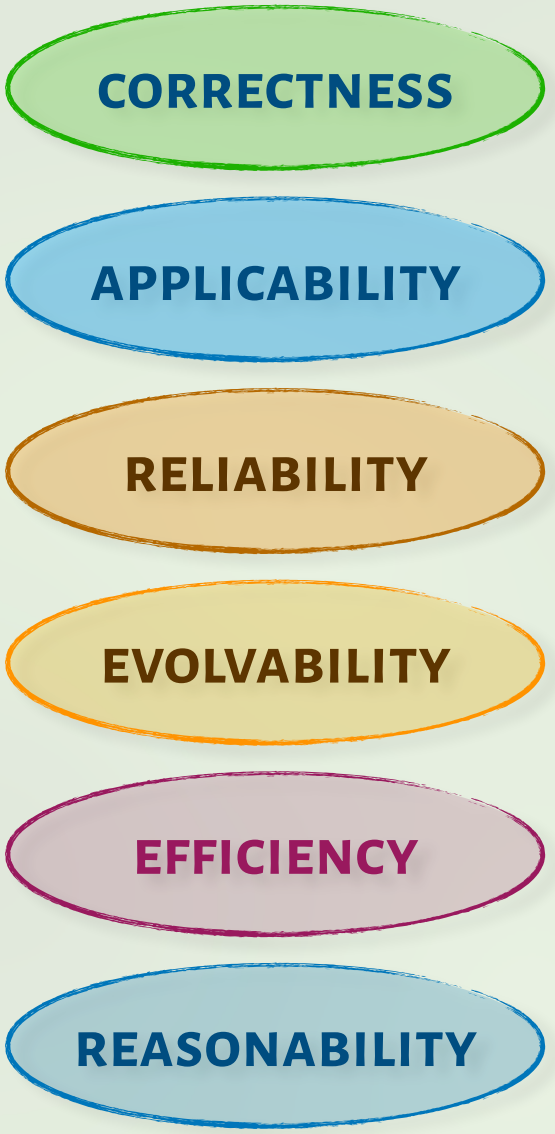
Amazingly, Clean Room worked. Clean Room teams demonstrated very high quality numbers. When I read about it, I was stunned, but then I came across a passage in a book that I was reading about the process. The author said that many programmers wrote their predicates after writing a section of code, but that experienced programmers often wrote the predicates first.. Gee, that sounds familiar, doesn't it? In TDD, we write the test first and the test is, essentially, a specification of the behavior of the code we are about to write.

In the software industry, we've been chasing quality for years. The interesting thing is there are a number of things that work. Design by Contract works. Test Driven Development works. So do Clean Room, code inspections and the use of higher-level languages.

All of these techniques have been shown to increase quality. And, if we look closely we can see why: all of them force us to reflect on our code.

That's the magic, and it's why unit testing works also. When you write unit tests, TDD-style or after your development, you scrutinize, you think, and often you prevent problems without even encountering a test failure.

Now, as you're reading this, you might think that I'm saying that we can get away with doing nothing as long as we sit back in our chairs, rest our chins on our hands, and think about our code. I don't think so. I think that approach may work for short periods for some people, but software development is a long-haul activity. We need practices which help us achieve continuous discipline and a continuous state of reflection. Clean Room and TDD are two practices which


CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY

ZEN


CORRECTNESS
APPLICABILITY
RELIABILITY
EVOLVABILITY
EFFICIENCY
REASONABILITY

ZEN

= Meditation

… is a practice where an individual uses a technique – such as mindfulness, or **focusing the mind on a particular object, thought, or activity – to train attention and awareness, and achieve a mentally clear** and emotionally calm **and stable state**

- wikipedia



ZEN AND THE ART OF SOFTWARE QUALITY



@phil_nash

sonarsource

levelofindirection.com/refs/zen.html



Nina Ranns

ISO C++ Foundation 总监

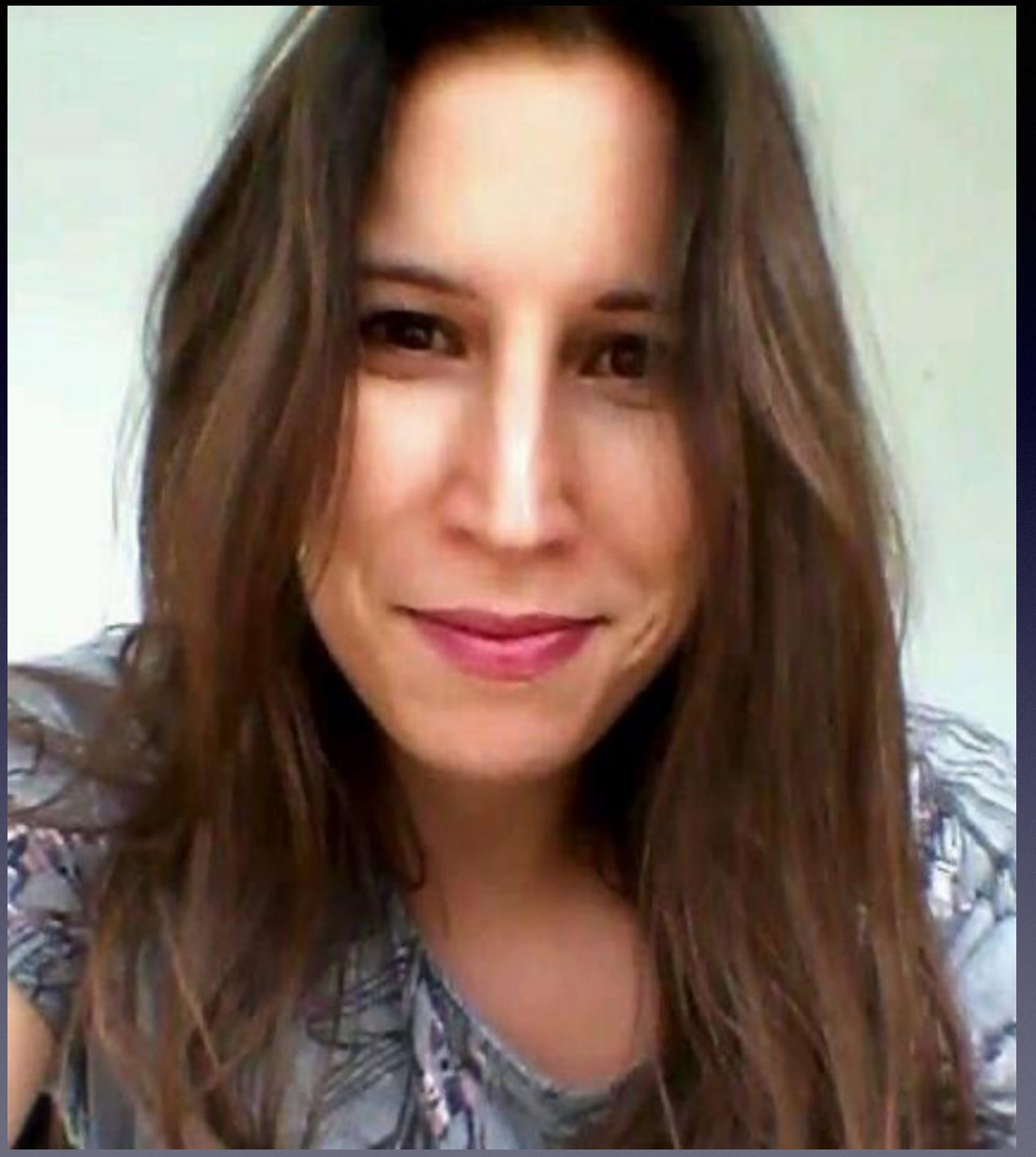
Nina Ranns自2013年以来一直是C++标准委员会的成员，主要关注语言的核心部分，并自2018年起担任委员会秘书。在她的职业生涯中，她曾为西门子、摩托罗拉、Datasift和赛门铁克工作，从UMTS网络的一部分到基于云的防病毒产品。目前是独立顾问，与EDG、QT和最近的Bloomberg签订了合同，她热切地扩展她的库知识，并帮助创建新的多态分配器友好库类型。

A Tour of C++ Recognised User-Type Categories

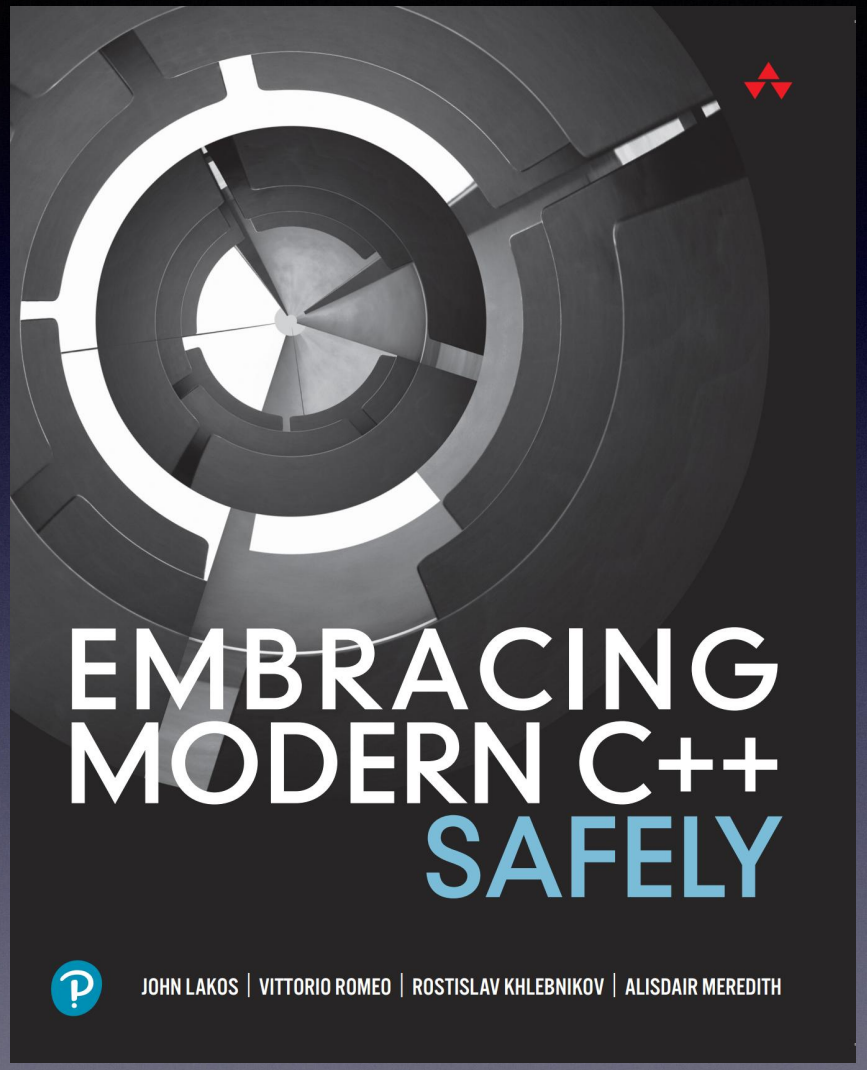
NINA RANNS

Nina Ranns

Committee member since 2013,
Cat owner,
Coffee drinker,
Chocolate lover,
Comma supervisor, and
C++ nerd



2215



- John Lakos
- Vittorio Romeo
- Rostislav Khlebnikov
- Alisdair Meredith
- An extended team of collaborators

2216

What Is The Book About?

- Analyze every new feature of the C++11/14 language
- Categorize risk/reward given the potential for misunderstanding (safe, conditionally safe, unsafe)
- common pitfalls and how to avoid them

Types with special provisions

- treating user defined types as built in types in certain cases
- new features require new definitions (constexpr specifier and literal types)
- “blessing” certain common programming patterns inherited from C code

C++ Object Model

C++ Object Model

- “The constructs in a C++ program create, destroy, refer to, access, and manipulate objects.”

C++ Object Model

- “The constructs in a C++ program create, destroy, refer to, access, and manipulate objects.”
- “An object occupies a region of storage in its period of construction, throughout its lifetime, and in its period of destruction.”

C++ Object Model

- “The *object representation* of an object of type T is the sequence of *N* unsigned char objects taken up by the object of type T, where *N* equals sizeof(T) ”

2222

C++ Object Model

- “The *object representation* of an object of type T is the sequence of *N* unsigned char objects taken up by the object of type T, where *N* equals sizeof(T) ”
- “The *value representation* of an object is the set of bits that hold the value of type T.”

Types of objects

- objects of scalar types (arithmetic types, enumeration types, pointer types, pointer-to-member types, std::nullptr_t)
- objects of user defined types, i.e. class types
- arrays

Types of objects

- objects of scalar types (arithmetic types, enumeration types, pointer types, pointer-to-member types, std::nullptr_t)
- objects of user defined types, i.e. class types
- arrays

Types of objects

- objects of scalar types (arithmetic types, enumeration types, pointer types, pointer-to-member types, std::nullptr_t)
- objects of user defined types, i.e. class types
- arrays

Objects of user defined type

- normally have constructor called when lifetime begins
- normally have destructor called when lifetime ends
- normally have values copied using assignment or copy construction
- normally have unspecified layout
- normally have differing object representation and value representation

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers
- definition incrementally expanded to cover more user defined types

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers
- definition incrementally expanded to cover more user defined types

```
struct X{
    int a,b;
    char c = 'a';
    long l;
};

X x1{1, 2, 'b', 0l}; // x1.a = 1, x1.b = 2, x1.c = 'b' , x1.l = 0l
X x2{1};             // x2.a = 1, x2.b = {}, x2.c = 'a', x2.l = {}
X x3{x2};            // x3.a = 1, x3.b = 0,  x3.c = 'a', x3.l = 0l
```

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers
- definition incrementally expanded to cover more user defined types

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers (since C++20)
- definition incrementally expanded to cover more user defined types

```
struct X {
    int a = 1;
    int b;
    char c = 'a';
    long l = 0;
};

int main()
{
  int  a = 4;
  X x1{.c = 'a'};
  X x2{.b = a, .c = 'c', .l = 31}; // x2.b = 4
  X x3{.a = x2.b, .c = 'a', .l = 31}; // x2.b is uninitialised
}
```

```
struct X {
    int a = 1;
    int b;
    int d;
    char c = 'a';
    long l = 0;
};
int main()
{
  int  a = 4;
  X x1{.c = 'a'};
  X x2{.b = a, .c = 'c', .l = 31}; // x2.b = 4
  X x3{.a = x2.b, .c = 'a', .l = 31}; // x2.b is uninitialised
}
```

```
struct X {
    int a = 1;
    int b;
    int d;
    char c = 'a';
    long l = 0;
};
int main()
{
  int  a = 4;
  X x1{.c = 'a'};
  X x2{.b = a, .c = 'c', .l = 31}; // x2.b = 4
  X x3{.a = x2.b, .c = 'a', .l = 31}; // x2.b is uninitialised
}
```

“The initializations of the elements of the aggregate are evaluated in the element order. “

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers
- definition incrementally expanded to cover more user defined types

Aggregates

- existed in C++98
- can be initialised as a collection of objects (aggregate initialisation)
- allows for designated initialisers
- definition incrementally expanded to cover more user defined types

Aggregates - C++03

- An aggregate is an array or a class with
  - no user-declared constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes

Aggregates - C++03

- An aggregate is an array or a class with
  - no user-declared constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes

Aggregates - C++03

- An aggregate is an array or a class with
  - no user-declared constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes

Aggregates - C++03

- An aggregate is an array or a class with
  - no user-declared constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes

Aggregates - C++11

- An aggregate is an array or a class with
  - no user-provided or explicit constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes
  - no brace-or-equal-initializers for non-static data members

Aggregates - C++14

- An aggregate is an array or a class with
  - no user-provided or explicit constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no base classes

Aggregates - C++17

- An aggregate is an array or a class with
  - no user-provided, explicit, or inherited constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no virtual, private, or protected base classes

ect base classes in declaration order, followed by the direct non-static data members that are n

```
struct Y{
  int i;
};
struct X : public Y{
  char c;
  long l;
};

X x1{{1}, 'c', 2l};
X x2{1, 'c', 2l};
```

Aggregates - C++17

- An aggregate is an array or a class with
  - no user-provided, explicit, or inherited constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no virtual, private, or protected base classes

Aggregates - C++20

- An aggregate is an array or a class with
  - no user-declared, explicit, or inherited constructors,
  - no private or protected direct non-static data members,
  - no virtual functions, and
  - no virtual, private, or protected base classes

```
struct X {
    X() = delete;
};

int main() {
    X x1; // ill-formed - default c'tor is deleted
    X x2{}; // compiles!
}
```

```
struct X {
    int i{1};
    X(int) = delete;
};

int main() {
    X x1(1); // ill-formed - value c'tor is deleted
    X x2{2}; // compiles!
}
```

```
struct X {
    int i{1};
    X(int) = delete;
};

int main() {
    X x1(1); // ill-formed - value c'tor is deleted
    X x2{2}; // compiles!
}
```

See P1008 for more details

Library trait - std::is_aggregate

- Introduced in C++17 to resolve the problem of emplace and construct library idioms

```
template <class U, class... Args&>
    void construct(U* p, Args&&... args);
Effects: ::new((void *)p) U(std::forward<Args>(args)...)

U(std::forward<Args>(args)...) // doesn't work if U is an aggregate
U{std::forward<Args>(args)...} // preffers std::initializer_list constructors
                               // for non aggregates
```

Library trait - std::is_aggregate

- C++20 adds aggregate parens initialisation (see P0960). Library trait obsolete ?

```
struct X{
 int i, j;
};
X x1(1);       // ok in C++20
X x2(1,2);     // ok in C++20
X x3(.i = 3); // designated initialisers not supported
```

Library trait - std::is_aggregate

- C++20 adds aggregate parens initialisation (see P0960). Library trait obsolete ?

```
struct X{
 int i, j;
};
X x1(1);       // ok in C++20
X x2(1,2);     // ok in C++20
X x3(.i = 3); // designated initialisers not supported
```

Note : “… narrowing conversions are permitted, designators are not permitted, a temporary object bound to a reference does not have its lifetime extended, and there is no brace elision. “

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Standard layout types

- Classes with “predictable” layout.
- “Standard-layout classes are useful for communicating with code written in other programming languages.”

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

```
struct X{};  // standard layout, no non static data members

struct Y : public X { // standard layout, static data members first declared
                      // in Y only
    int i, j;
};

struct Z : public Y {}; // standard layout, static data members first declared
                        // in Y only

struct Q : public Z {   // not standard layout, static data members first
                        // declared in Q and in Y
    char c;
};
```

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Distinct address (C++11)

- Two objects that are not bit-fields may have the same address if one is a subobject of the other, or if at least one is a base class subobject of zero size and they are of different types;

Distinct address (C++20)

- Two objects that are not bit-fields may have the same address if one is a subobject of the other, or if at least one is a subobject* of zero size and they are of different types;

Distinct address (C++20)

- Two objects that are not bit-fields may have the same address if one is a subobject of the other, or if at least one is a subobject* of zero size and they are of different types;

*in C++20, [no_unique_address] non-static members could be of zero size

Distinct address (C++20)

- Two objects that are not bit-fields may have the same address if one is a subobject of the other, or if at least one is a subobject* of zero size and they are of different types

*in C++20, [no_unique_address] non-static members could be of zero size

```
struct A {};         // standard layout
struct B : A {};     // standard layout
struct C : A {};     // standard layout

struct D : B,C {}; // not standard layout
struct E : B {       // not standard layout
    A a;
};
```

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Standard Layout Class

- has the same access control for all non-static data members,
- has all non-static data members and bit-fields first declared either in the most derived class or in only one base class,
- has no base classes that require a distinct address,
- has no virtual functions and no virtual base classes,
- all base classes and non-static data members are of standard layout type, no reference members

Implications of Standard Layout

- layout of the object depends only on the non-static data members
- the address of a standard layout class object and its first non-static data member is the same
- the address of a standard layout class object and all of its base classes is the same
- *offsetof* is well defined with standard layout class types*

Implications of Standard Layout

- layout of the object depends only on the non-static data members
- the address of a standard layout class object and its first non-static data member is the same
- the address of a standard layout class object and all of its base classes is the same
- *offsetof* is well defined with standard layout class types*

Implications of Standard Layout

- layout of the object depends only on the non-static data members
- the address of a standard layout class object and its first non-static data member is the same
- the address of a standard layout class object and all of its base classes is the same
- *offsetof* is well defined with standard layout class types*

Implications of Standard Layout

- layout of the object depends only on the non-static data members
- the address of a standard layout class object and its first non-static data member is the same
- the address of a standard layout class object and all of its base classes is the same
- *offsetof* is well defined with standard layout class types*

* and conditionally supported with non standard layout types

Common Initial Sequence

- Two standard layout structs have a common initial sequence if some of their non-static data members and bit-fields in declaration order have layout-compatible types

Common Initial Sequence

- Two standard layout structs have a common initial sequence if some of their non-static data members and bit-fields in declaration order have the same (underlying) type

Common Initial Sequence (C++20)

- Two standard layout structs have a common initial sequence if some of their non-static data members and bit-fields in declaration order have the same (underlying) type, and either both are declared with [[no_unique_address]] or neither is.

Reading from a Union

- If an inactive member of a union has a common initial sequence with the active member, it's possible to read a member of the common initial sequence through that inactive member
- The read must happen through the union object, not through a pointer or a reference to the inactive member

Reading from a Union

```
struct A {
  int i;
  long l;
};
```

```
struct B {
  int j;
  char c;
};
```

```
union U {
  A a;
  B b;
};
```

```
U u{A{}}; // `a` is the active member;
```

```
int k = u.b.j; // ok, `b.j` is a member of the common initial sequence
int l = u.b.c; // UB, `b.c` is not a member of the common initial sequence
```

Reading from a Union

```
struct A {
  int type_id;
  long l;
};
```

```
struct B {
  int type_id;
  char c;
};
```

```
union U {
  A a;
  B b;
};
```

```
U u{A{}}; // `a` is the active member;
```

```
if (u.b.type_id == b_type) {
  /*...*/
}
```

Reading from a Union

```
struct A {
  int type_id;
  long l;
};
```

```
struct B {
  int type_id;
  char c;
};
```

```
union U {
  A a;
  B b;
};
```

```
U u{A{}}; // `a` is the active member;
```

```
B  &bref = u.b; // forms reference to inactive union member
if (b.type_id == b_type) {  // UB: reading through reference to an inactive member
  /*...*/
}
```

Library trait - std::is_standard_layout

- Introduced in C++11
- limited usability (static_assert to ensure correctness of code)
- questionable quality of implementation suggest it isn’t widely used

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Trivial special member functions

Trivial default constructor

- not user-provided
- class has no virtual functions and no virtual base classes
- all the base classes and non-static data members have trivial default constructors
- class has no default member initializers for non-static data members

Trivial default constructor

- not user-provided
- class has no virtual functions and no virtual base classes
- all the base classes and non-static data members have trivial default constructors
- class has no default member initializers for non-static data members

Trivial default constructor

- not user-provided
- class has no virtual functions and no virtual base classes
- all the base classes and non-static data members have trivial default constructors
- class has no default member initializers for non-static data members

Trivial default constructor

- not user-provided
- class has no virtual functions and no virtual base classes
- all the base classes and non-static data members have trivial default constructors
- class has no default member initializers for non-static data members

Trivial copy constructor and assignment operator

- not user-provided
- class has no virtual functions and no virtual base classes
- all the base classes and non-static data members have trivial corresponding special member function

Trivial destructor

- not user-provided
- not virtual
- destructors of all base classes and non-static data members are trivial

Trivially copyable type

- at least one non-deleted* copy operation
- all copy operations are trivial
- has a trivial non-deleted destructor

Trivially copyable type

- at least one non-deleted* copy operation
- all copy operations are trivial
- has a trivial non-deleted destructor

* *C++20 definition takes into account constraints too*

Trivially copyable type

- at least one non-deleted* copy operation
- all copy operations are trivial
- has a trivial non-deleted destructor

* *C++20 definition takes into account constraints too*

Trivially copyable type

- at least one non-deleted* copy operation
- all copy operations are trivial
- has a trivial non-deleted destructor

* *C++20 definition takes into account constraints too*

Trivially copyable type

- at least one non-deleted* copy operation
- all copy operations are trivial
- has a trivial non-deleted destructor

Note that there are no requirements on access control or ambiguity of call

* *C++20 definition takes into account constraints too*

Implications of Trivially Copyable

- the value is contained in the underlying byte representation
- can be *memcpy*-ed to and from another object of same type
- can be *memcpy*-ed to and from an array of *char*, *unsigned char*, or *std::byte*
- compiler can implement the copy (copy/move construction or copy/move assignment) as *memcpy*

Implications of Trivially Copyable

- the value is contained in the underlying byte representation
- can be *memcpy*-ed to and from another object of same type
- can be *memcpy*-ed to and from an array of *char*, *unsigned char*, or *std::byte*
- compiler can implement the copy (copy/move construction or copy/move assignment) as *memcpy*

Implications of Trivially Copyable

- the value is contained in the underlying byte representation
- can be *memcpy*-ed to and from another object of same type
- can be *memcpy*-ed to and from an array of *char*, *unsigned char*, or *std::byte*
- compiler can implement the copy (copy/move construction or copy/move assignment) as *memcpy*

Library trait - std::is_trivially_copyable

- Introduced in C++11
- prerequisite for bitwise copy
- does not ensure copy operation is non-deleted - generic code should also check for availability of the relevant copy operation

```
struct X
{
    X() = default;
    X(const X&) = delete;
    X(X&&) = default;
};

static_assert(std::is_trivially_copyable_v<X>);

template<class T> requires std::is_trivially_copyable_v<T>
void copy(T& dest, const T& source)
{
    std::memcpy(&dest, &source , sizeof(dest)); // ok for X ?
}
```

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Trivial Type

- trivial class type = trivially copyable + non-deleted/eligible trivial default constructor
- remnant from POD days - only used in the library to describe types which used to be required to be of POD type

Library trait - std::is_trivial

- Introduced in C++11
- limited usability - check for POD requirement ?

Library trait - std::is_trivial

- Introduced in C++11
- limited usability - check for POD requirement ?

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

What are PODs

- Defined in C++98 for C compatibility
- their physical representation mimics that of a C type (it doesn't use C++ only features which affect its physical representation)
- behave like C types in terms of construction, copying, and destruction (it doesn't make use of C++ only features that are relevant to constructions, destruction, and copying)

POD: Plain Old Data

- The original definition :
  - an aggregate (no user-declared constructors, no private or protected non-static data members, no base classes, and no virtual functions)
  - no non-static data members of reference type, all non-static data members of POD type, no user-defined copy assignment operator, and no user-defined destructor.

POD in C++11

- properties relevant to object layout - standard layout types
- properties relevant to object construction, copying and destruction - trivial types
- original term no longer needed - deprecated, then removed in C++20

21 Strings library [strings]

21.1 General [strings.general]

1 This Clause describes components for manipulating sequences of any non-array POD (3.9) type. In this Clause such types are called *char-like types* , and objects of char-like types are called *char-like objects* or simply *characters*.

2 The following subclauses describe a character traits class, a string class, and null-terminated sequence utilities, as summarized in Table 61.

23 Strings library [strings]

23.1 General [strings.general]

1 This Clause describes components for manipulating sequences of any non-array trivial standard-layout (6.8.1) type. Such types are called *char-like types*, and objects of char-like types are called *char-like objects* or simply *characters*.

2 The following subclauses describe a character traits class, string classes, and null-terminated sequence utilities, as summarized in Table 74.

Library trait - std::is_pod

- Introduced in C++11
- deprecated in C++20

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Literal types

- A literal type is one for which it might be possible to create an object at compile time.
- It is not a guarantee that it is possible to create such an object, nor is it a guarantee that any object of that type will be usable at compile time.

Literal types

- A literal type is one for which it might be possible to create an object within a constant expression.
- It is not a guarantee that it is possible to create such an object, nor is it a guarantee that any object of that type will usable in a constant expression.

Constant expressions

- requirement for constant initialisation (i.e. initialisation during compile time)
- three new keywords constexpr(C++11), consteval(C++20), and constint(C++20)

Limitations of constant expressions

- things that are not possible/correct at compile time
- things that are difficult to implement
- For more information : C++Now 2019: Daveed Vandevoorde “C++ Constants”, P0992 Andrew Sutton “Translation and evaluation A mental model for compile-time metaprogramming”

Literal types

- a scalar type
- a reference type referring to a literal type
- an array of literal type
- void (C++14)
- a literal class type

Literal class (C++11)

- it is an aggregate type or has at least one constexpr constructor or constructor template that is not a copy or move constructor
- every constructor call and full-expression in the brace-or-equal-initializers for non-static data members is a constant expression
- all of its non-static data members and base classes are of literal types.
- has a trivial destructor

Literal class (C++17)

- it is a closure type, an aggregate type or has at least one constexpr constructor or constructor template that is not a copy or move constructor
- every constructor call and full-expression in the brace-or-equal-initializers for non-static data members is a constant expression
- all of its non-static data members and base classes are of literal types.
- has a trivial destructor

Literal class (C++20)

- it is a closure type, an aggregate type or has at least one constexpr constructor or constructor template that is not a copy or move constructor

- all of its non-static data members and base classes are of literal types.

- has a constexpr destructor

Library trait - std::is_literal_type

- introduced in C++11, deprecated in C++17, removed in C++20
- the real question is - does this initialisation qualify as constant initialisation ?
- will we need the definition of literal type in the future ?

Library trait - std::is_literal_type

- introduced in C++11, deprecated in C++17, removed in C++20
- the real question is - does this initialisation qualify as constant initialisation ?
- will we need the definition of literal type in the future ?

```
struct X
{
    constexpr X() = default;
    X(int){};
    constexpr ~X(){};
};

constinit X x1;     // ok
constinit X x2{1}; // error
```

Library trait - std::is_literal_type

- introduced in C++11, deprecated in C++17, removed in C++20
- the real question is - does this initialisation qualify as constant initialisation ?
- will we need the definition of literal type in the future ?

Library trait - std::is_literal_type

- introduced in C++11, deprecated in C++17, removed in C++20
- the real question is - does this initialisation qualify as constant initialisation ?
- will we need the definition of literal type in the future ?

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Structural types (C++20)

- prerequisite for non-type template parameter
- scalar type
- lvalue reference type
- literal class type with
  - all base classes and non static data members public, non mutable and of structural type

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Types with special provisions

aggregate types

standard-layout types

trivially copyable types

trivial types

POD types

literal types

structural types

implicit-lifetime types

Object lifetime

(before C++20)

- “An object is created by a definition, by a new-expression, when implicitly changing the active member of a union, or when a temporary object is created.“

Object lifetime

(before C++20)

- “An object is created by a definition, by a new-expression, when implicitly changing the active member of a union, or when a temporary object is created.“

```
1 struct X { int a, b; };
2 X *make_x() {
3   X *p = (X*)malloc(sizeof(struct X));
4   p->a = 1;
5   p->b = 2;
6   return p;
7 }
```

Object lifetime

(before C++20)

- “An *object* is created by a definition, by a *new-expression*, when implicitly changing the active member of a union, or when a temporary object is created.“

```
1 struct X { int a, b; };
2 X *make_x() {
3   X *p = (X*)malloc(sizeof(struct X));
4   p->a = 1;
5   p->b = 2;
6   return p;
7 }
```



```
char *buffer = (char*)malloc(n * sizeof(char)));
char *buffer_end = buffer + sizeof(char) * n;
```

```
char *buffer = (char*)malloc(n * sizeof(char)));
char *buffer_end = buffer + sizeof(char) * n;
```

- “When an expression J that has integral type is added to or subtracted from an expression P of pointer type, the result has the type of P.
  — If P evaluates to a null pointer value and J evaluates to 0, the result is a null pointer value.
  — Otherwise, if P points to an array element i of an array object x with n elements the expressions P + J (…) points to the array element i + j of x
  — Otherwise, the behavior is undefined.

```
char *buffer = (char*)malloc(n * sizeof(char)));
char *buffer_end = buffer + sizeof(char) * n;
```

- “When an expression J that has integral type is added to or subtracted from an expression P of pointer type, the result has the type of P.
  — If P evaluates to a null pointer value and J evaluates to 0, the result is a null pointer value.
  — Otherwise, if P points to an array element i of an array object x with n elements the expressions P + J (…) points to the array element i + j of x
  — Otherwise, the behavior is undefined.

```
char *buffer = (char*)malloc(n * sizeof(char)));
char *buffer_end = buffer + sizeof(char) * n;
```



- “When an expression J that has integral type is added to or subtracted from an expression P of pointer type, the result has the type of P.
  — If P evaluates to a null pointer value and J evaluates to 0, the result is a null pointer value.
  — Otherwise, if P points to an array element i of an array object x with n elements the expressions P + J (…) points to the array element i + j of x
  — Otherwise, the behavior is undefined.

```
1 struct X { int a, b; };
2 X *make_x() {
3   X *p = (X*)malloc(sizeof(struct X));
4   p->a = 1;
5   p->b = 2;
6   return p;
7 }
```

X has a trivial constructor.
X has a trivial destructor.

special lifetime rules for types which require no code for construction and no code fo

Implicit lifetime types

- scalar types
- array types
- aggregate classes and classes with a trivial destructor and at least one trivial constructor

Object lifetime

(before C++20)

- “An object is created by a definition, by a new-expression, when implicitly changing the active member of a union, or when a temporary object is created.“

Object lifetime

(after C++20)

- “An object is created by a definition, by a new-expression, by an operation that implicitly creates objects, when implicitly changing the active member of a union, or when a temporary object is created.“

Object lifetime

(after C++20)

- “An object is created by a definition, by a new-expression, by an operation that implicitly creates objects, when implicitly changing the active member of a union, or when a temporary object is created.“
- aligned_alloc, malloc, calloc, realloc, memcpy, memove, std::bit_cast, std::pmr::memory_resource.allocate()

Object lifetime

(after C++20)

- “An object is created by a definition, by a new-expression, by an operation that implicitly creates objects, when implicitly changing the active member of a union, or when a temporary object is created.“
- aligned_alloc, malloc, calloc, realloc, memcpy, memove, std::bit_cast, std::pmr::memory_resource.allocate()
- “An operation that begins the lifetime of an array of char, unsigned char, or std::byte implicitly creates objects within the region of storage occupied by the array.”

Implicit lifetime types

- Created to make C code well defined in C++
- additionally solves the problem of pointer arithmetic
- no library trait

Summary

type	when do you care	trait
aggregate type	member-wise initialisation, ability to use designated initializers	is_aggregate (C++17)
trivially copyable type	copying using memcpy, memove, and bit_cast	std::is_trivially_copyable (C++11)
trivial type	partial check for C code compatibility	std::is_trivial (C++11)
standard layout type	pointer-interconvertibility, union access through common initial sequence, offsetoff	std::is_standard_layout (C++11)
POD type	check for C code compatibility	std::is_pod (C++11, deprecated in C++20)
literal type	requirement for compile time initilisation	std::is_literal_type (C++11, deprecated in C++17, removed in C++20)
structural type	requirement for non-type template parameter	no trait
implicit-lifetime type	makes common programming patterns well defined	no trait

Thank you !