
Parallel Processing

Lecture: **系统评测与基准测试程序**

邵恩

高性能计算机研究中心

Outline

■ 性能指标

- 基本指标：IPC和执行时间
- 多核
- 内存指标

■ 性能的Benchmarks

- 为什么要使用基准？
- 多种有用的基准

■ 实验评测需要注意的地方

性能评估

■ 如何评估性能?

- 指标
 - 单核
 - 多核
 - 内存系统
- 基准测试 (benchmarks)
- 评估的方法
 - 分析模型
 - 模拟器

性能指标

■ 如何衡量一个计算系统的性能？

- 计算的速度？如何定义？
- 物体的速度可以定义为：
 - 米每秒 (m/s)
- 那么，CPU 呢？
 - 百万条指令每秒 (MIPS, Million Instructions Per Second)

■ 合理吗？

- 例如
 - 程序 X (在 CPU A 上, 编译时为 10000 条指令) 在 CPU A 上运行时需要 2 秒
 - 程序 X (在 CPU B 上, 编译时为 5000 条指令) 在 CPU B 上运行时需要 1.5 秒
 - 哪个CPU更快？

性能指标

■ 处理器性能方程

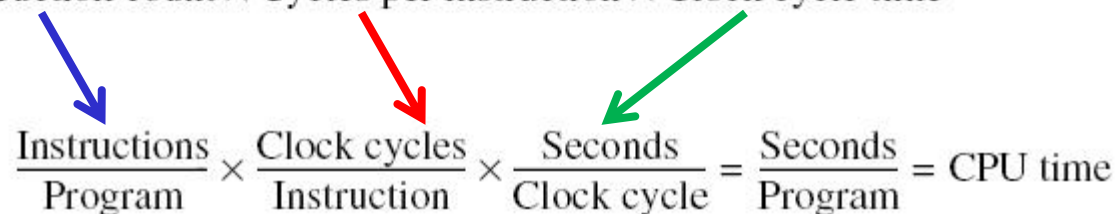
$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{IC(指令计数)} \times \text{CPI(每条指令的时钟周期数)} \times \text{CCT(一个时钟周期时间)}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$



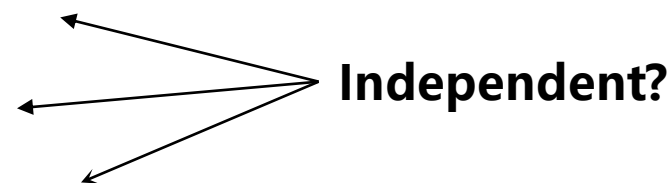
The diagram illustrates the derivation of the CPU time equation by substituting units for the terms in the previous equation. A blue arrow points from 'Instruction count' to 'Instructions / Program'. A red arrow points from 'Cycles per instruction' to 'Clock cycles / Instruction'. A green arrow points from 'Clock cycle time' to 'Seconds / Clock cycle'. The resulting equation is:

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

计算机设计的基本原理

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$

- CPU time = **IC**(指令计数) × **CPI**(每条指令的时钟周期数) × **CCT**(一个时钟周期时间)
- What can the system architecture affect?
 - **CPI**: 内存延迟, IO延迟, ...
 - **CCT**: 高速缓存组织, 功率限制, ...
 - **IC**: 操作系统开销, 编译器选择, ...
- 注: 不同的指令类型有不同的**CPI**(每条指令的时钟周期数)



$$\text{CPU clock cycles} = \sum_{i=1}^n \text{IC}_i \times \text{CPI}_i$$

CPU时钟周期数量

$$\text{CPU time} = \left(\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

CCT(一个时钟周期时间)

Outline

■ 性能指标

- 基本指标：IPC和执行时间
- 多核
- 内存指标

■ 性能的Benchmarks

- 为什么要使用基准？
- 多种有用的基准

■ 实验评测需要注意的地方

基准测试 (Benchmarks)

- 使用一组应用程序来衡量性能
- 类型：
 - 核函数：Kernels
 - 很小
 - 真实应用的关键组成部分
 - Toy 程序
 - 100-行
 - 第一个编程写可运行的程序，例如：快速排序
 - 综合基准测试程序 (Synthetic benchmarks)
 - 为了匹配真实应用程序的配置和计算行为，而发明的“模拟”程序
 - 实际的应用程序

基准测试 (Benchmarks)

■ 基准测试的运行条件是什么？

- 编译器统一标准
 - 对相同编程一语言（例如 C++ 或 C）的所有程序，使用同一编译器和相同编译器的标识选项
- 针对基准测试的源代码由两种评测模式
 - 模式一：不允许修改Benchmarks源代码。
 - 模式二：允许修改源代码，只要更改后的版本产生相同的输出。

基准测试 (Benchmarks)

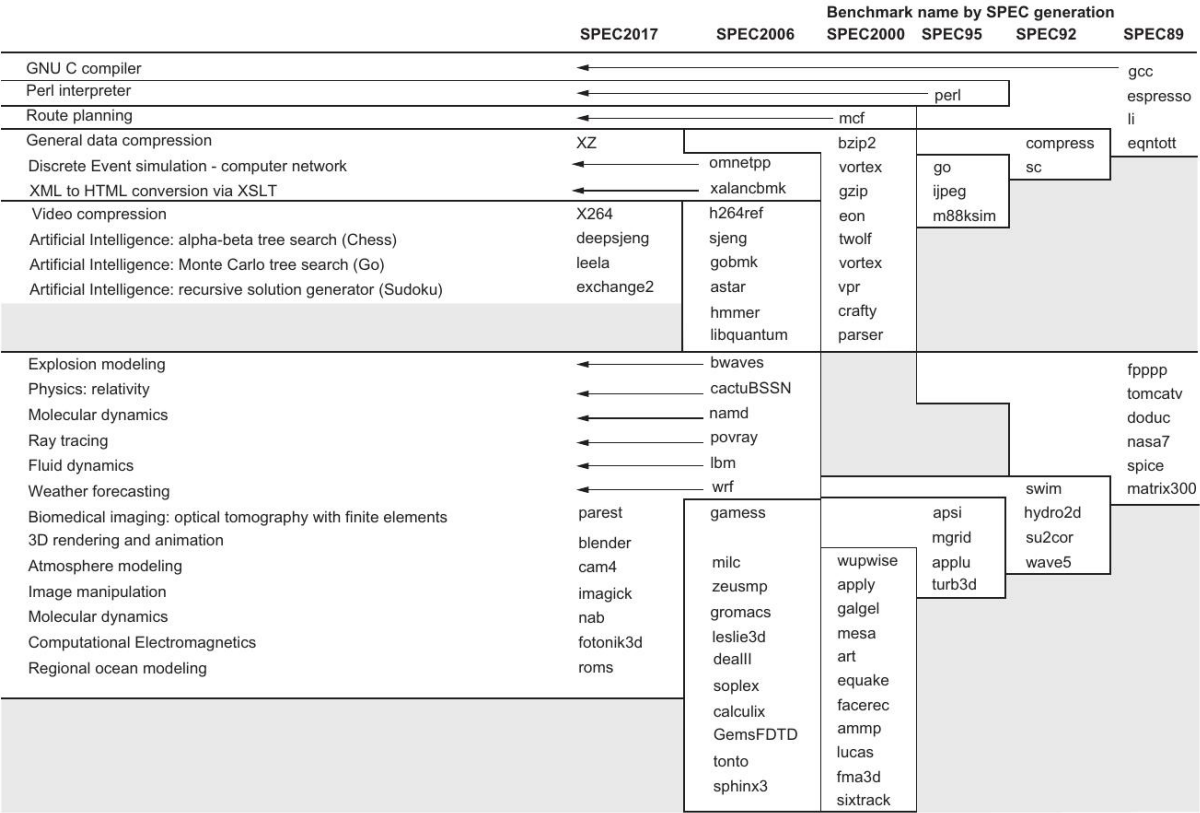
■ 基准测试套件的需求

- 多线程应用程序
 - 共享内存 CMP 已经无处不在
 - 需要额外处理能力的应用程序将需要并行
- 新兴工作负载
 - 未来的处理器将被设计用来满足新兴应用程序的需求 (ChatGPT? 元宇宙?)
 - 基准测试套件应该对新兴应用具有代表性
- 多样性
 - 应用程序日益多样化, 在各种平台上运行并适应不同的使用模式
- 采用最先进技术
 - 基准测试不仅应该代表新兴应用程序, 还应该使用最先进技术。

The PARSEC Benchmark Suite: Characterization and Architectural Implications

标准性能评估公司(SPEC,Standard Performance Evaluation Corporation)

- SPEC是一个非营利性组织，旨在建立、维护和认可计算机系统的标准性能基准测试
 - 它提供了一系列基准测试，用于评估计算机系统的性能



- 例如，SPEC CPU 2017基准测试包含43个基准测试，分为四个套件：
 - SPECspeed 2017整数
 - SPECspeed 2017浮点
 - SPECrate 2017整数
 - SPECrate 2017浮点
- 分别用于比较计算机完成单个任务的时间，以及测量吞吐量（单位时间内的工作量）的性能

标准性能评估公司(SPEC,Standard Performance Evaluation Corporation)

- SPEC提供了多种基准测试，
 - 包括云计算、CPU、图形/工作站、高性能计算、Java客户端/服务器、存储、电源和虚拟化等领域的基准测试

Category	Name	Measures performance of
Cloud	Cloud_IaaS 2016	Cloud using NoSQL database transaction and K-Means clustering using map/reduce
CPU	CPU2017	Compute-intensive integer and floating-point workloads
Graphics and workstation performance	SPECviewperf® 12	3D graphics in systems running OpenGL and Direct X
	SPECwpc V2.0	Workstations running professional apps under the Windows OS
	SPECapcSM for 3ds Max 2015™	3D graphics running the proprietary Autodesk 3ds Max 2015 app
	SPECapcSM for Maya® 2012	3D graphics running the proprietary Autodesk 3ds Max 2012 app
	SPECapcSM for PTC Creo 3.0	3D graphics running the proprietary PTC Creo 3.0 app
	SPECapcSM for Siemens NX 9.0 and 10.0	3D graphics running the proprietary Siemens NX 9.0 or 10.0 app
High performance computing	SPECapcSM for SolidWorks 2015	3D graphics of systems running the proprietary SolidWorks 2015 CAD/CAM app
	ACCEL	Accelerator and host CPU running parallel applications using OpenCL and OpenACC
	MPI2007	MPI-parallel, floating-point, compute-intensive programs running on clusters and SMPs
Java client/server	OMP2012	Parallel apps running OpenMP
	SPECjbb2015	Java servers
Power	SPECpower_ssj2008	Power of volume server class computers running SPECjbb2015
Solution File Server (SFS)	SFS2014	File server throughput and response time
	SPECsfs2008	File servers utilizing the NFSv3 and CIFS protocols
Virtualization	SPECvirt_sc2013	Datacenter servers used in virtualized server consolidation

Figure 1.18 Active benchmarks from SPEC as of 2017.

普林斯顿共享内存计算机应用程序库 (PARSEC)

- 由 9 个应用程序和 3 个内核函数组成
- 针对多处理器芯片的基准测试
- 实验
 - <https://parsec.cs.princeton.edu/documentation.htm>
 - 下载&&运行

Program	Input Set simlarge						Input Set native
	Working Set 1			Working Set 2			Working Set 2
	Data Structure(s)	Size	Growth Rate	Data Structure(s)	Size	Growth Rate	Size Estimate
blackscholes	options	64 KB	C	portfolio data	2 MB	C	same
bodytrack	edge maps	512 KB	const.	input frames	8 MB	const.	same
canneal	elements	64 KB	C	netlist	256 MB	DS	2 GB
dedup	data chunks	2 MB	C	hash table	256 MB	DS	2 GB
facesim	tetrahedra	256 KB	C	face mesh	256 MB	DS	same
ferret	images	128 KB	C	data base	64 MB	DS	128 MB
fluidanimate	cells	128 KB	C	particle data	64 MB	DS	128 MB
frequmine	transactions	256 KB	C	FP-tree	128 MB	DS	1 GB
streamcluster	data points	64 KB	C	data block	16 MB	user-def.	256 MB
swaptions	swaptions	512 KB	C	same as WS1	same	same	same
vips	image data	64 KB	C	image data	16 MB	C	same
x264	macroblocks	128 KB	C	reference frames	16 MB	C	same

针对性能结果的总结

■ 简单的评测方法

- 比较套件中程序执行时间的算术平均值

■ 另一种方法

- 为每个基准添加一个加权因子并使用加权算术平均值

■ 得到与相对计算时间的归一化的执行时间

$$1.25 = \frac{\text{SPECRatio}_A}{\text{SPECRatio}_B} = \frac{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_A}}{\frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

$$\text{Geometric mean} = \sqrt[n]{\prod_{i=1}^n \text{sample}_i}$$

多线程工作负载的指标

- **IPC 是指令数每时钟周期(IPC, Instruction Per Clock)**
 - **IPC 针对单程序，不是多程序工作负载的准确可靠的性能指标（可能会导致误导性或错误的结论）**
 - 不同的执行路径和线程交错
 - 执行的自旋锁循环指令的数量可能会因微架构而异
 - 自旋锁循环指令不会影响整体执行时间
 - 但影响IPC
 - **多程序工作负载的性能指标**
 - 系统吞吐量 (STP, system throughput)
 - 平均归一化周转时间 (ANTT, average normalized turnaround time)
-
- 自旋锁是一种锁，它会导致试图获取它的线程在循环中等待（“自旋”），同时不断检查锁是否可用。
 - 由于线程仍然处于活动状态，但并未执行有用的任务，因此使用这种锁是一种忙等待。
 - 如果线程可能仅被阻塞很短的时间，自旋锁就是高效的，但如果持有时间较长，则可能变得浪费。

多线程工作负载的指标

■ 系统吞吐量 (STP, System Throughput)

- T_i^{SP} : 单程序执行模式 (条件: 程序独立运行) 下的执行时间
- T_i^{MP} : 多程序执行模式 (条件: 程序与其他程序共同运行) 下的执行时间
- 都是一个程序的执行时间, 但执行条件不同

“归一化” 进度 (Normalized Progress) $NP_i = \frac{T_i^{SP}}{T_i^{MP}},$

$$STP = \sum_{i=1}^n NP_i = \sum_{i=1}^n \frac{T_i^{SP}}{T_i^{MP}},$$

多线程工作负载的指标

- 平均归一化周转时间 (ANTT, Average Normalized Turnaround Time)
 - 是一种加权IPC（每周期指令数）指标
 - 用于测量多程序工作负载中程序的平均周转时间，并通过它们的独立执行时间进行归一化
 - NTT是NP的倒数
 - **ANTT 是一个越低越好的指标**，越低说明多程序一起运行与程序单独运行在性能上越接近。

$$NTT_i = \frac{T_i^{MP}}{T_i^{SP}}.$$

$$ANTT = \frac{1}{n} \sum_{i=1}^n NTT_i = \frac{1}{n} \sum_{i=1}^n \frac{T_i^{MP}}{T_i^{SP}}$$

测量访存性能应该用什么指标?

- **独立于CPU，但与 CPU 性能密切相关**
 - 不是浮点运算每秒 (Flop, Floating-point Operations Per Second)
 - 也不是 IPC(每个时钟周期内执行的指令数量)
- **提供内存系统的总体性能,以及内存层次结构中每一层的性能**
- **涵盖现代存储系统的复杂性**
- **简单、易于使用且易于理解**

测量访存性能——APC简介

- Access Per Cycle (APC)
- APC 衡量为每个周期的内存访问次数
 - 测量整体内存系统性能
 - 每个存储的层次，都有自己的 APC 值
 - APC直接会主导整体 CPU 性能
- APC 的优势
 - 将内存评估与 CPU 评估分开
 - 更好地理解整个内存系统
 - 更好地理解计算能力和内存系统性能之间的匹配

APC in Detail

- APC 是在特定内存层次（即 L1、L2、L3、主内存），
总内存访问请求次数除以该级别的内存访问周期
 - 即：在一个时钟周期内的某一个内存层次的访存次数
- $APC = M/T$
- 不同的内存层次有不同的APC
 - 数据缓存： APC_D L1 Data Cache
 - 指令缓存： APC_I L1 Instruction Cache
 - 数据缓存和指令缓存都位于 CPU 的一级缓存（L1 Cache）中，它们通常具有相同的访问速度。
 - 主存： APC_M Main Memory
- APC性能是分层的（每层都不一样）

平均存储器访问时间 (AMAT, Average Memory Access Time)

- AMAT 是计算机科学中用于分析计算机存储器系统性能的常用指标，来衡量存储器性能。
- 越低越好
- 成分：
 - 命中时间 (H, Hit Time)
 - 从缓存中读取数据所需的时间
 - 未命中率 (MR, Miss Rate)
 - 缓存未命中的概率，即：cache miss的概率
 - 平均未命中惩罚 (AMP, Average Miss Penalty)
 - 从下一级存储器中读取数据所需的额外时间

$$AMAT = H + MR \times AMP$$

- 它考虑到命中和未命中对存储器系统性能的不同影响
- 此外，AMAT 可以递归扩展到多层存储器层次结构。它关注局部性和缓存未命中如何影响整体性能，并允许快速分析不同的缓存设计技术。

平均存储器访问时间 (AMAT, Average Memory Access Time)

- 例如，假设缓存的命中时间为 1 个时钟周期，未命中率为 5%，从下一级存储器读取数据所需的时间为 100 个时钟周期
- 按照以下公示计算：

$$AMAT = H + MR \times AMP$$

- 则 $AMAT = 1 + 0.05 * 100 = 6$ 。
- 这意味着平均每次访问存储器需要花费 6 个时钟周期。

并发-AMAT (C-AMAT)

- 是对平均存储器访问时间 (AMAT) 的扩展, 考虑了并发存储器访问
- 由于 C-AMAT 是对 AMAT 的扩展, 因此在没有数据并发时, C-AMAT 会退化为 AMAT。

• 越低越好

• 成分

- H : 命中时间 (H, Hit Time)
 - 从缓存中读取数据所需的时间
- C_H : 命中并发性 (hit concurrency)
 - 在处理多个并发存储器访问请求时, 能够同时命中缓存的请求的数量
- C_M : 未命中并发性 (pure miss concurrency)
 - 在处理多个并发存储器访问请求时, 未能命中缓存且需要从下一级存储器读取数据的请求的数量
- pMR : 纯未命中率 (Pure Miss Rate) 是指纯未命中的概率
- $pAMP$: 纯平均未命中惩罚 (Pure Average Miss Penalty) 是指从下一级存储器读取数据所需的额外时间
- 纯未命中时钟周期: **Pure-miss cycle**

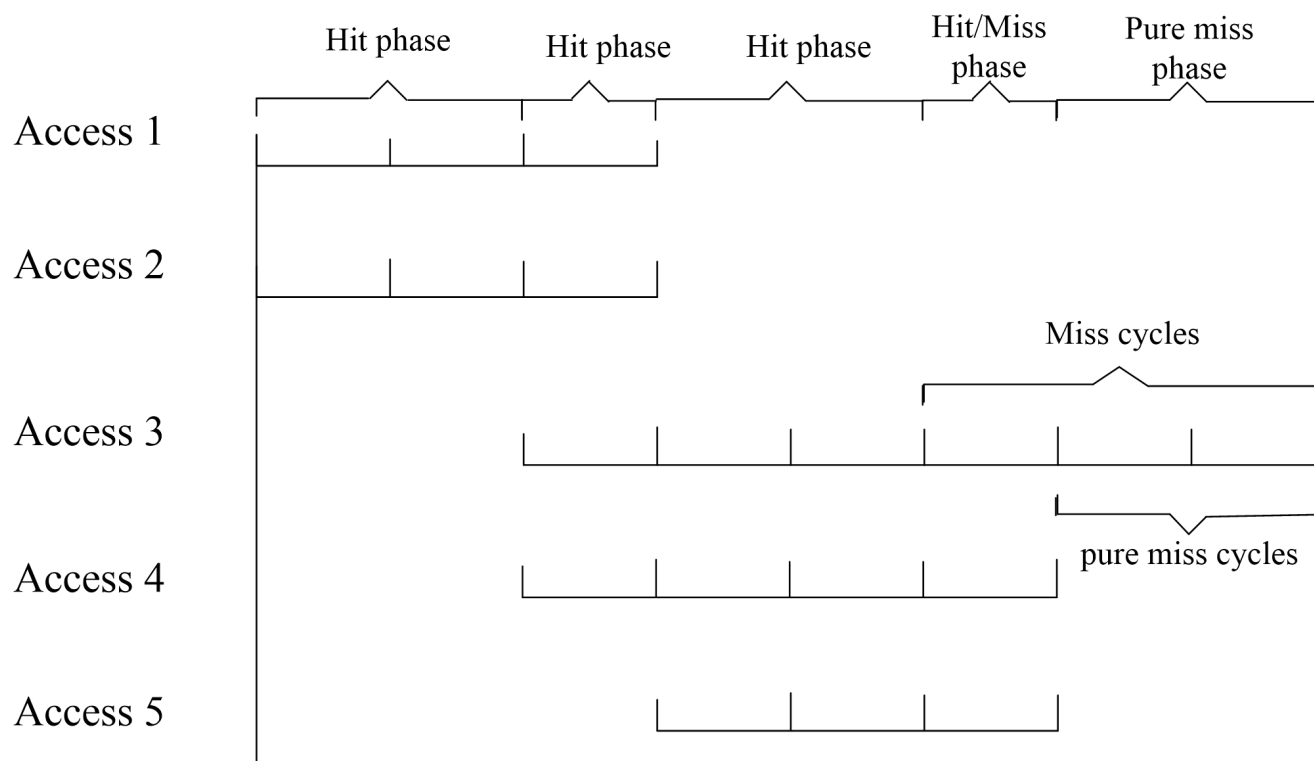
$$C\text{-}AMAT = \frac{H}{C_H} + pMR \times \frac{pAMP}{C_M}$$

$$AMAT = H + MR \times AMP$$

不同的视角

○ 序列视角: AMAT

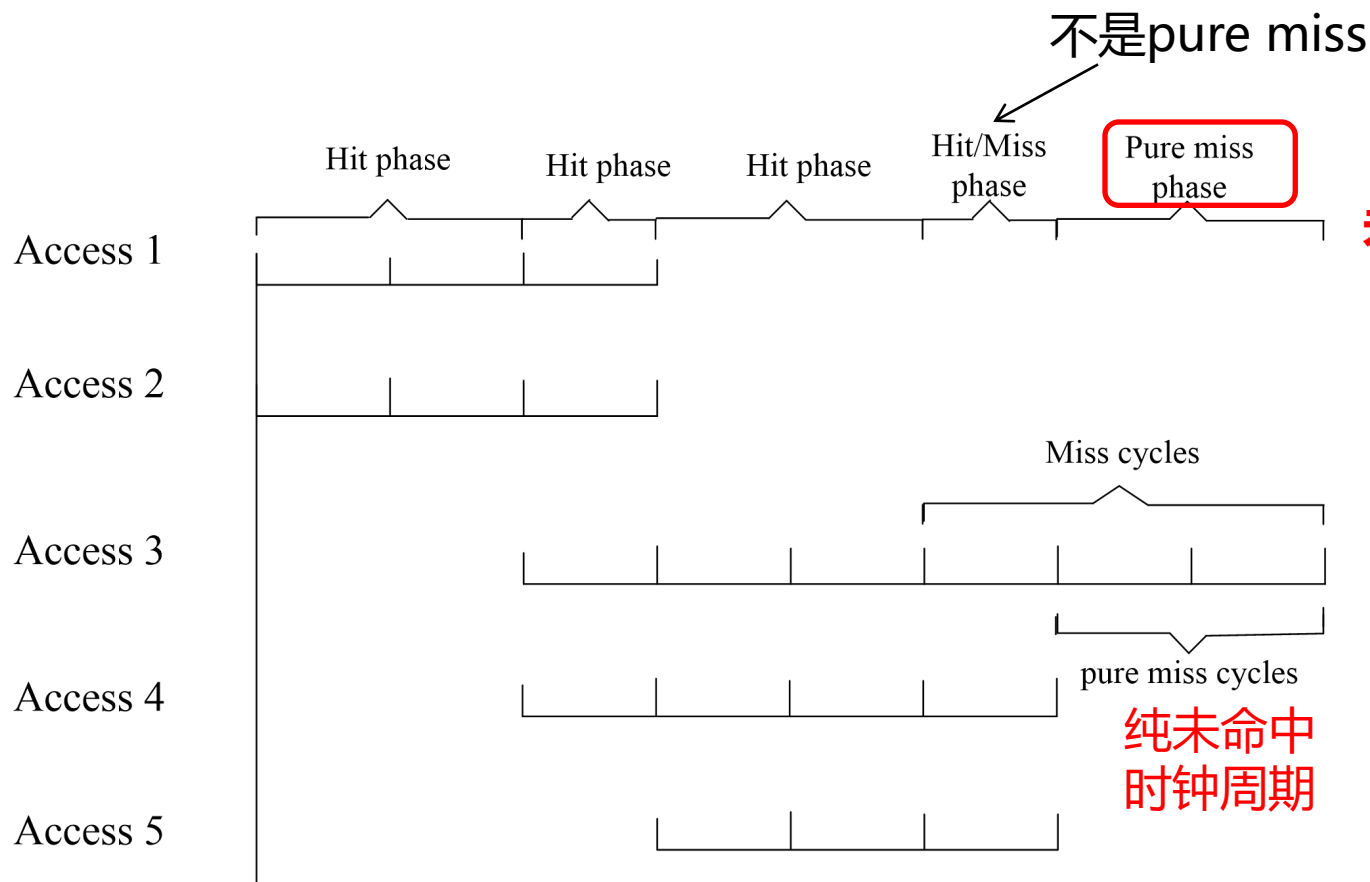
○ 并发视角: C-AMAT



纯未命中: Pure-miss

○ Pure miss 是影响性能的瓶颈

- Pure miss 至少包含一个纯 miss 周期的 miss
- 不与 hit 周期重叠的 miss 周期



未命中率 * 未命中惩罚



$$pMR \times \frac{pAMP}{C_M}$$

纯未命中
时钟周期

C-AMAT 是递归的

- 假设：我们有一个计算机系统，它具有两级缓存（L1 和 L2）和主存储器。我们可以使用 C-AMAT 模型递归地计算每一级存储器的 C-AMAT 值
- 首先，计算 L1 缓存的 C-AMAT 值。假设 L1 缓存的命中时间为 1 个时钟周期，纯未命中率为 5%，纯平均未命中惩罚为 10 个时钟周期，则 L1 缓存的 C-AMAT 值为：
$$\text{C-AMAT(L1)} = 1 + 0.05 * 10 = 1.5$$
- 接下来，计算 L2 缓存的 C-AMAT 值。假设 L2 缓存的命中时间为 10 个时钟周期，纯未命中率为 20%，纯平均未命中惩罚为 100 个时钟周期，则 L2 缓存的 C-AMAT 值为：
$$\text{C-AMAT(L2)} = 10 + 0.2 * 100 = 30$$
- 最后，计算主存储器的 C-AMAT 值。由于主存储器是存储器层次结构的最后一层，因此它没有未命中惩罚。假设主存储器的访问时间为 100 个时钟周期，则主存储器的 C-AMAT 值为：
$$\text{C-AMAT(Memory)} = 100$$
- 综上所述，整个存储器系统的 C-AMAT 值为：
$$\begin{aligned} \text{C-AMAT(System)} &= \text{C-AMAT(L1)} + \text{C-AMAT(L2)} + \text{C-AMAT(Memory)} \\ &= 1.5 + 30 + 100 = 131.5 \end{aligned}$$
- 这意味着平均每次访问存储器需要花费 131.5 个时钟周期。

C-AMAT 是递归的求解纯平均未命中惩罚

$$\frac{pAMP}{C_M}$$

- 在计算 C-AMAT(L1) 时, 需要知道 L1 缓存的命中时间、纯未命中率和纯平均未命中惩罚。
 - 其中, 纯平均未命中惩罚表示从下一级存储器 (即 L2 缓存) 读取数据所需的额外时间。
 - 因此, 在计算 C-AMAT(L1) 时, 需要先计算出 L2 缓存的 C-AMAT 值, 然后将其作为 L1 缓存的纯平均未命中惩罚。
- 同理, 在计算 C-AMAT(L2) 时, 需要知道 L2 缓存的命中时间、纯未命中率和纯平均未命中惩罚。
 - 其中, 纯平均未命中惩罚表示从下一级存储器 (即主存储器) 读取数据所需的额外时间。
 - 因此, 在计算 C-AMAT(L2) 时, 我们需要先计算出主存储器的 C-AMAT 值, 然后将其作为 L2 缓存的纯平均未命中惩罚。
- 综上所述, 在使用 C-AMAT 模型递归地计算每一级存储器的 C-AMAT 值时, 每一级存储器的 C-AMAT 值都依赖于下一级存储器的 C-AMAT 值。
 - 需要按照存储器层次结构从下到上的顺序依次计算每一级存储器的 C-AMAT 值。

体系结构对访存性能的影响因素

- C_H 命中并发性 (hit concurrency) : 多个并发存储器访问请求, 同时命中缓存的请求的数量
 - § 影响因素一: 多端口缓存
 - § 影响因素二: 多组缓存
 - § 影响因素三: 流水线缓存结构
- C_M 未命中并发性 (pure miss concurrency) : 多个并发存储器访问请求, 未能命中缓存且需要从下一级存储器读取数据的请求的数量
 - § 影响因素一: 非阻塞缓存结构
 - § 影响因素二: 预取逻辑

体系结构对访存性能的影响因素

○ 这些技术都可以增加 C_H 和 C_M

§ SMT (Simultaneous Multithreading)

- § 多线程技术，它允许多个线程同时在单个处理器核心上执行。
- § SMT 通过在硬件级别动态共享处理器资源，提高了处理器的利用率和吞吐量
- § 一个典型实现是英特尔的超线程技术 (Hyper-Threading)。

§ CMP (Chip Multiprocessor)

- § 一种多核处理器技术，它在单个芯片上集成了多个处理器核心。
- § 每个核心都可以独立地执行指令，从而提高了处理器的并行性能。
- § 可以有效地提高多线程应用程序的性能。

两个访存指标：APC Versus C-AMAT

○ APC (Accesses Per Cycle) $APC = A/T$

- § 表示处理器每个时钟周期能够从存储器系统中接收到的数据量
- § APC 则可以用于衡量存储器系统的整体吞吐量
- § 反映了存储器系统的服务质量 (QoS) 。

○ APC与传统IPC有很大不同

- § 内存活动周期 (以数据为中心/访问)
- § 重叠模式 (并发数据存取)

○ C-AMAT (Concurrent Average Memory Access Time)

- § 表示平均每次访问 (某层) 存储器需要花费的时钟周期
- C-AMAT 可以用于识别存储器层次结构中的潜在瓶颈
- C-AMAT是一个分析和优化工具
- C-AMAT 的值不依赖于它的五个参数

D. Wang, X.-H. Sun "Memory Access Cycle and the Measurement of Memory Systems", IEEE Transactions on Computers, vol. 63, no. 7, pp. 1626-1639, July.2014

内存级并行 (MLP, Memory Level Parallelism)

○ Memory Level Parallelism (MLP)

- § 在同一时间有多个内存操作（特别是缓存未命中或转换后备缓冲器 (TLB) 未命中）处于等待状态的能力
 - § **即：存储设备能够并行执行访存指令的数量**
- § 在单个处理器中，MLP 可以被视为指令级并行 (ILP) 的一种形式

○ MLP不考虑局部性，仅考虑能够并行执行访存指令的数量

Outline

■ 性能指标

- 基本指标：IPC和执行时间
- 多核
- 内存指标

■ 性能的Benchmarks

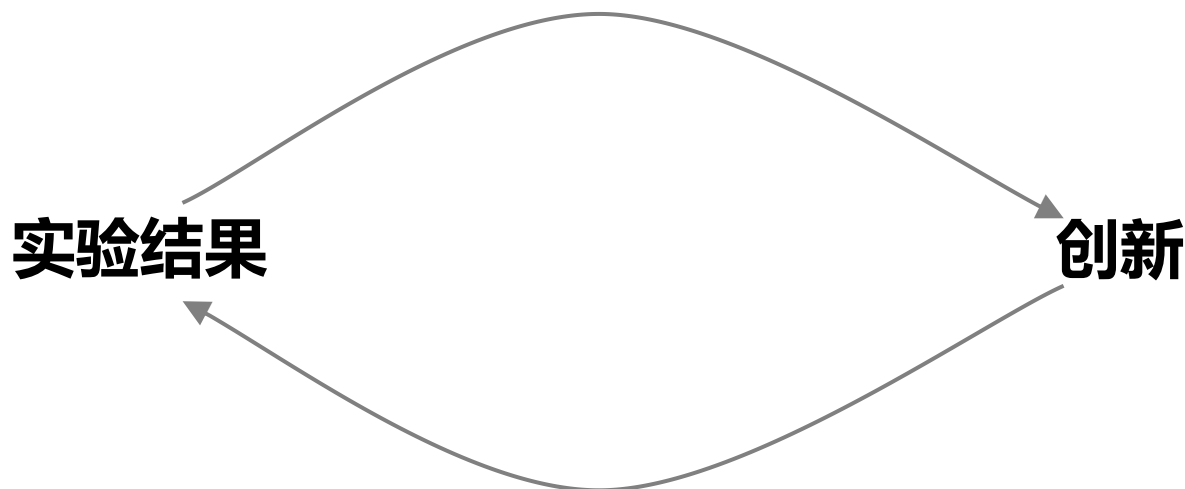
- 为什么要使用基准？
- 多种有用的基准

■ 实验评测需要注意的地方

你应该相信你的实验结果吗？

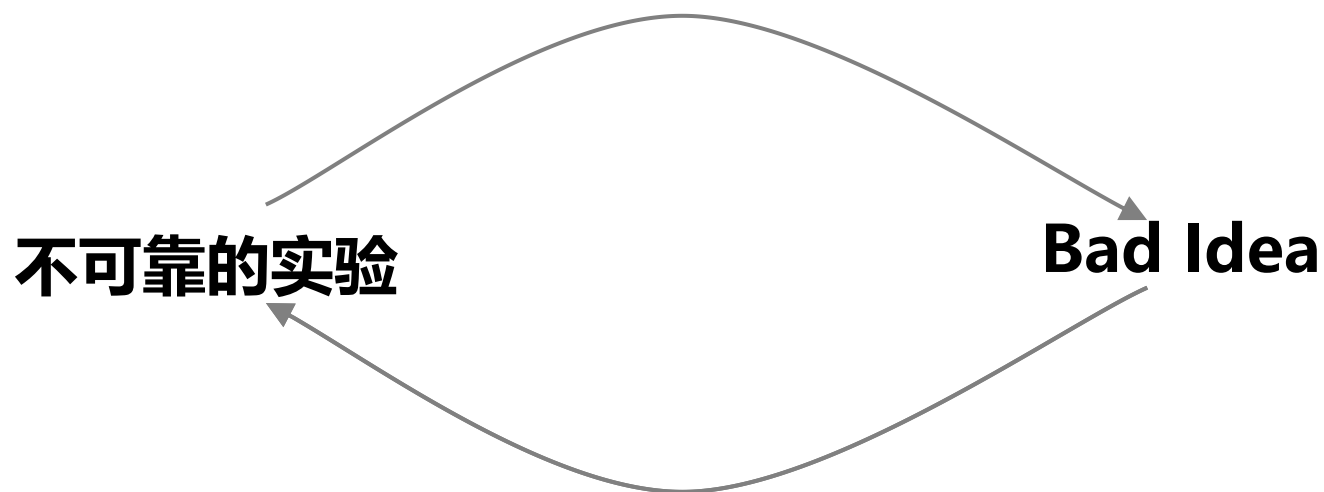
Amer Diwan, Google Stephen M. Blackburn, ANU
Matthias Hauswirth, U. Lugano Peter F. Sweeney, IBM Research
Attendees of Evaluate '11 workshop

为什么会有顾虑？



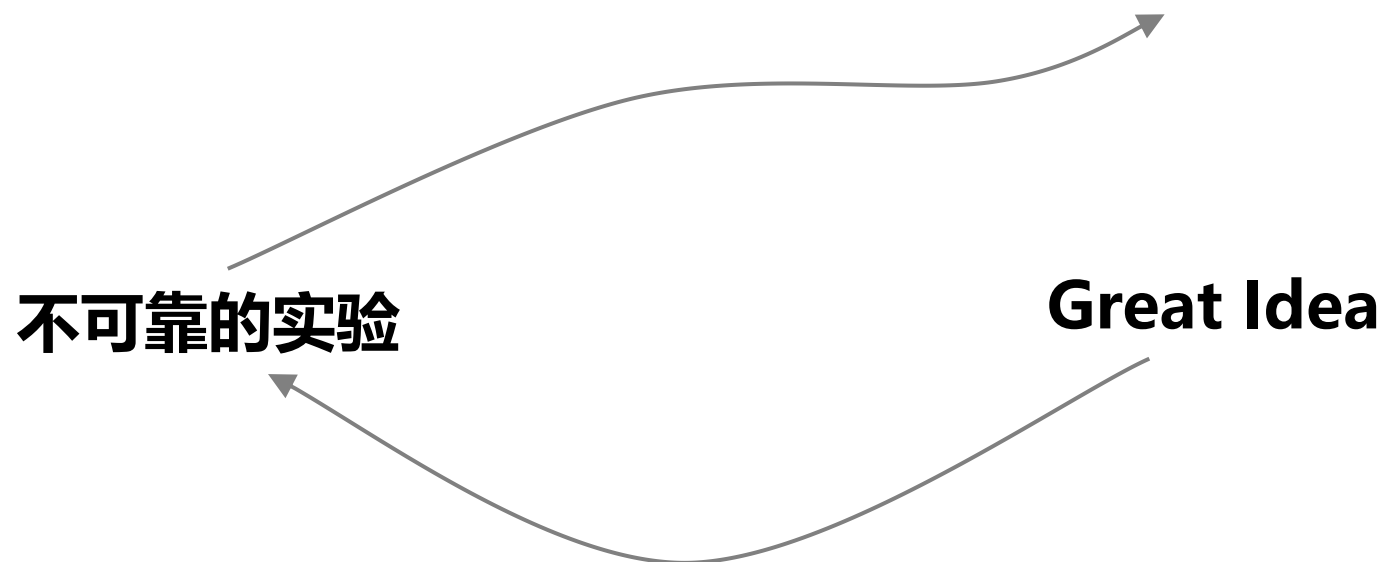
为了科学进步，我们需要可靠的实验。因为实验能够最真实反映创新可行性和效果。

不可靠的实验



不可靠的实验让Bad Idea看起来结果很好

不可靠的实验



让great idea 看起来结果并不理想!

论文

合理的实验很关键，但需要

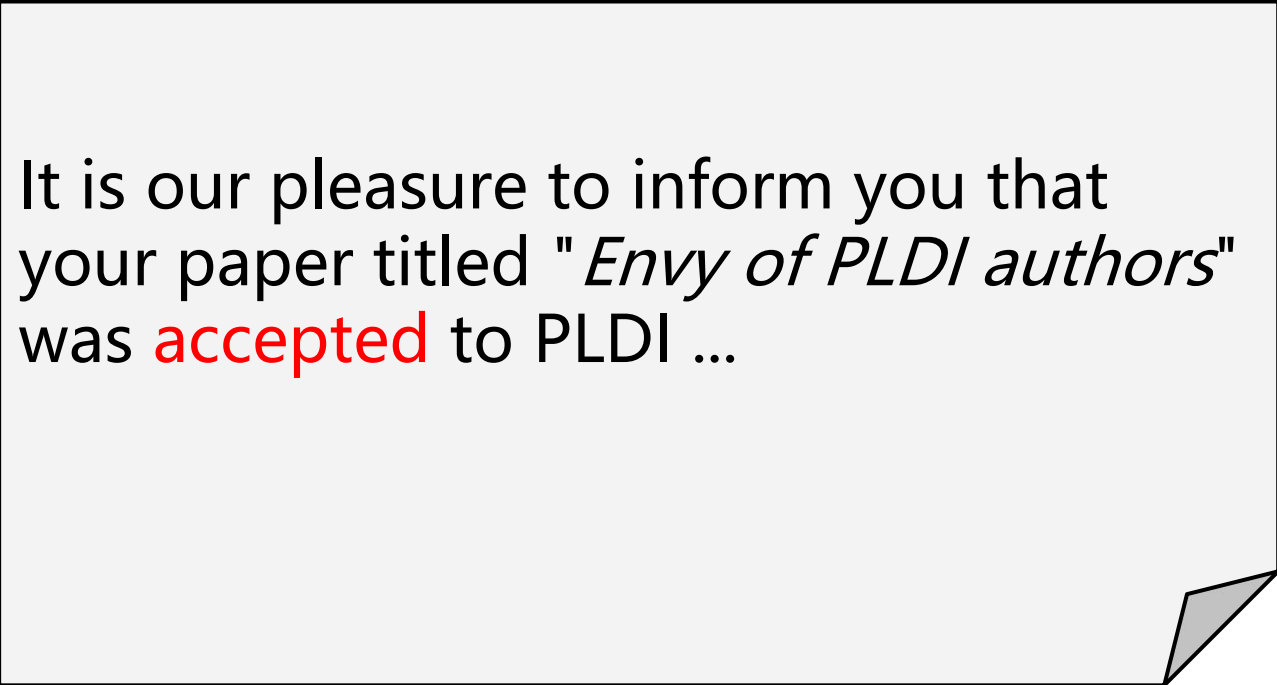
- 创意 (Creativity)
- 勤奋 (Diligence)

作为一个社区，我们必须

- **学习如何设计和进行合理的实验**
- **奖励和鼓励合理的实验**

四大 “罪” (fatal sins)

以下的四大 “罪” 很有可能会妨碍你工作被录用：



It is our pleasure to inform you that
your paper titled "*Envy of PLDI authors*"
was **accepted** to PLDI ...

罪过一：天真

Defn: 忽略声明所需的组件和实验环境

论文中的声明：所有计算机

真正的实验：某台特定的计算机



罪过一：天真

Defn: 忽略声明所需的组件

实验中仅使用了一个
benchmark



但是在论文中声明了整个测试
套件



无知会导致结果出现偏差

“天真”有时候并不明显

A 比 B 性能好!

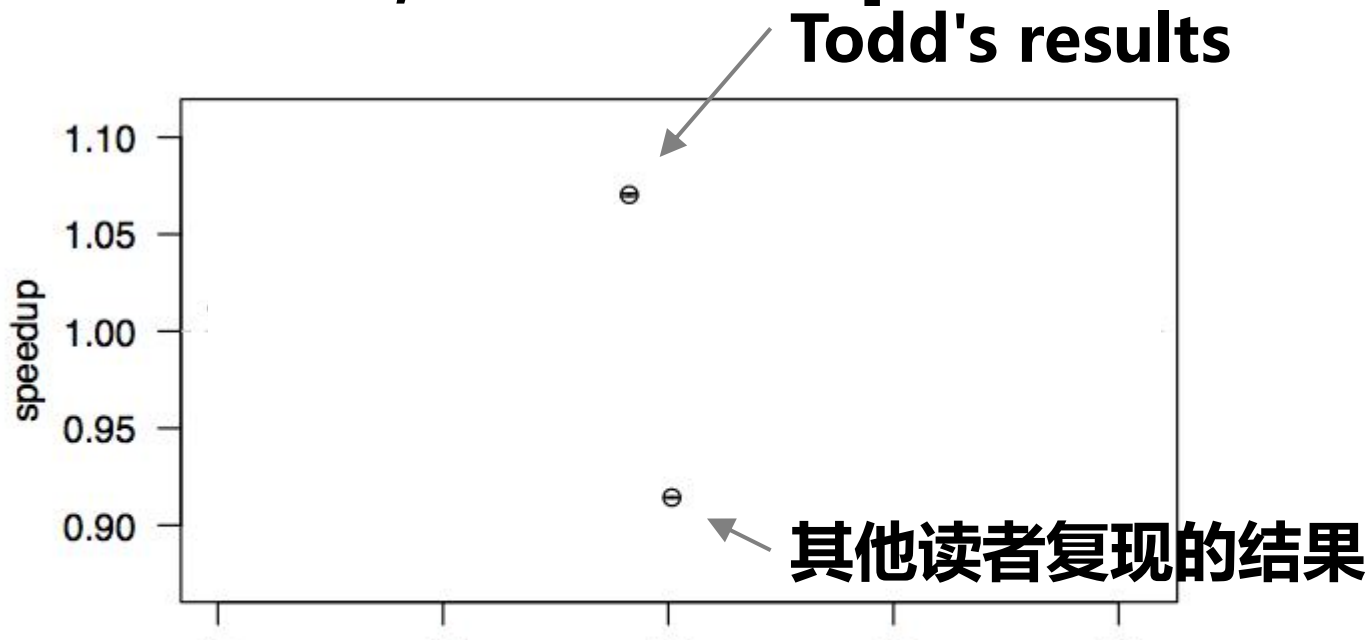
我发现正好相反!



你和合作者有过这样的对话吗?

忽略 Linux 环境变量

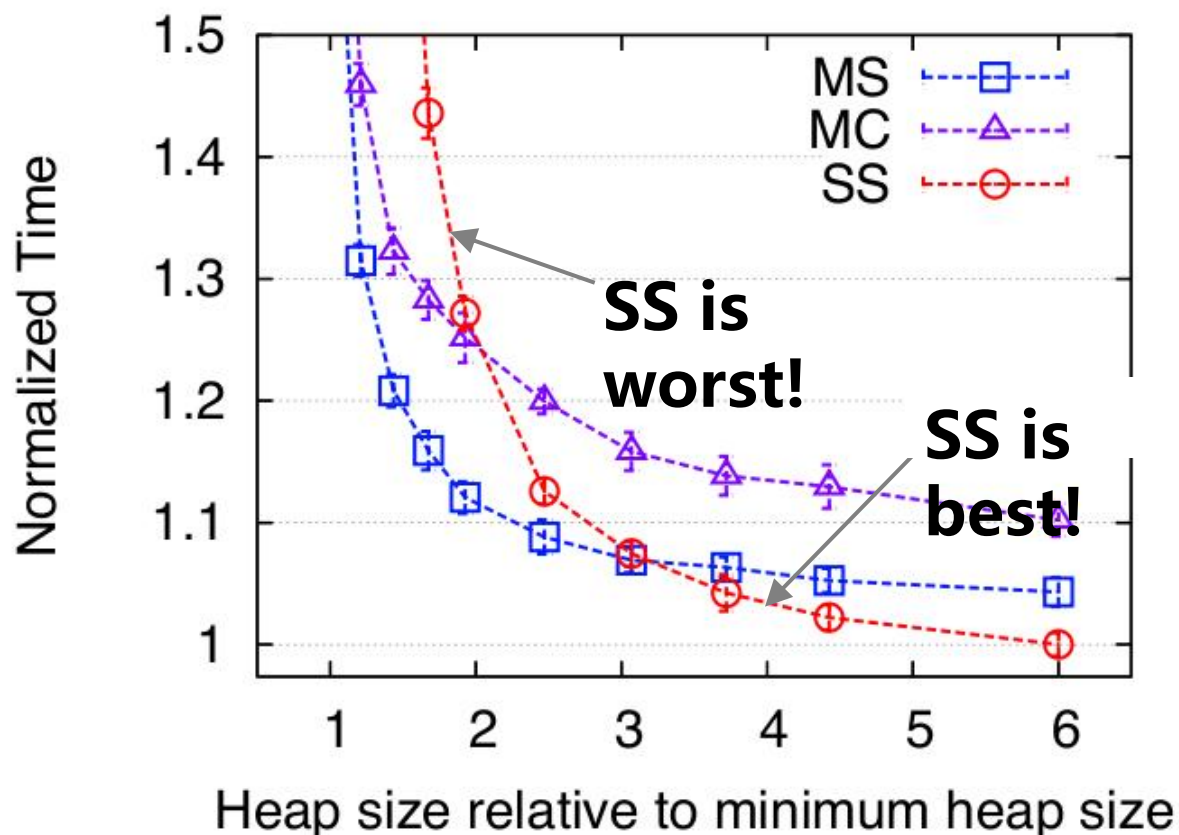
[Mytkowicz et al., ASPLOS 2009]



改变实验环境可以改变你的实验结果!

忽略堆大小

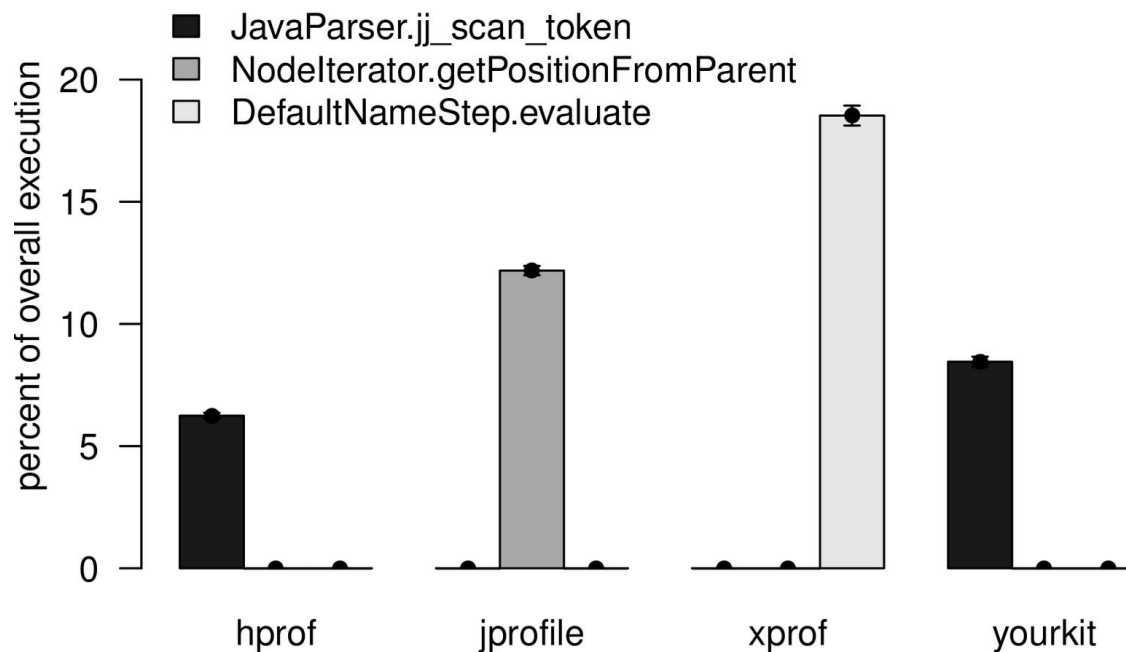
Graph from [Blackburn et al., OOPSLA 2006]



更改堆大小可能会直接改变实验的结果！

忽略硬件探查器的偏差

[Mytkowicz et al., PLDI 2010]



不同的硬件探查器会得出相互矛盾的结论!

罪过二：不当的环境描述

Defn: 实际实验使用与论文描述不一致的组件或计算环境

Experiment:
服务器的计算环境



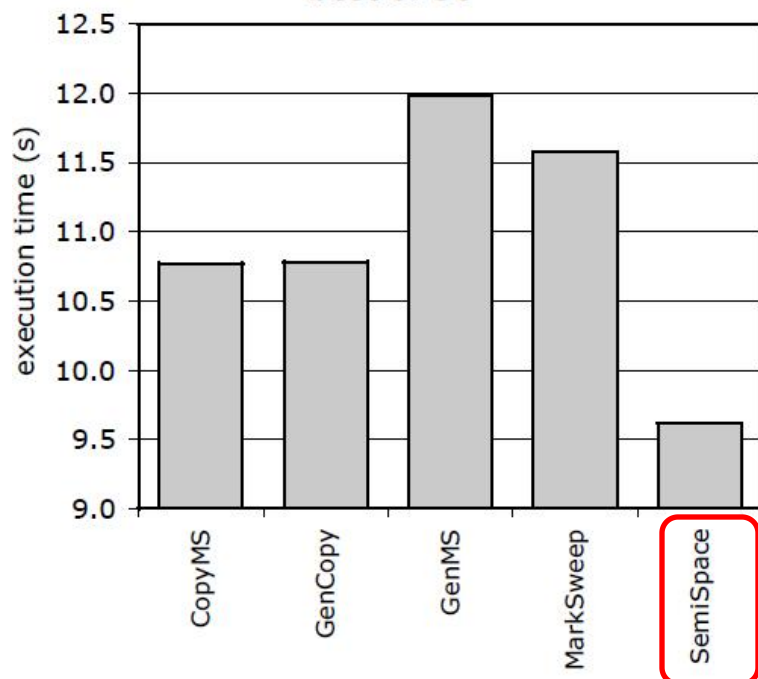
Claim:
移动终端的计算环境



不恰当的统计数据

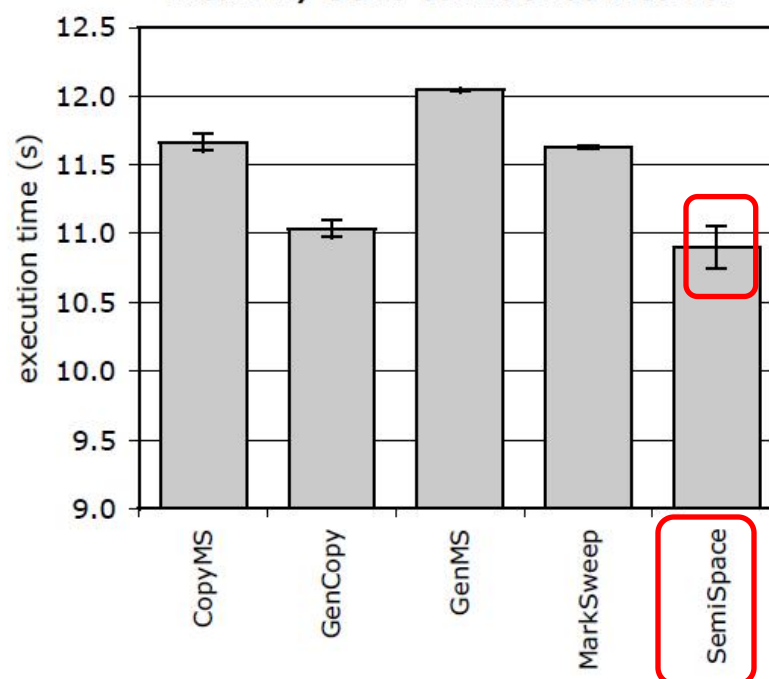
[Georges and Eeckhout, 2007]:

best of 30



(SemiSpace 是迄今为止最好的)

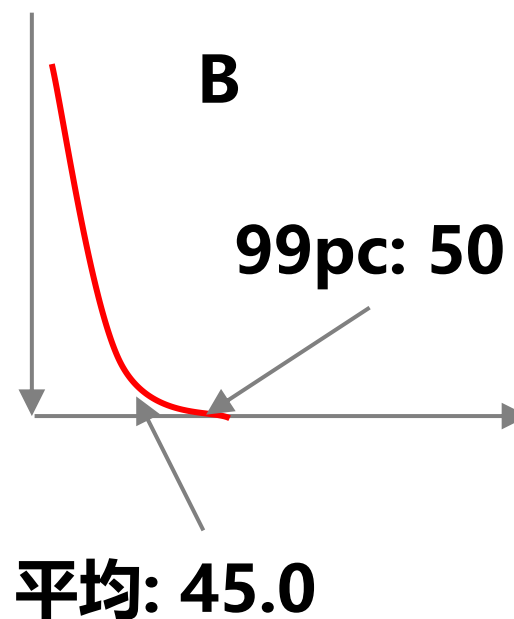
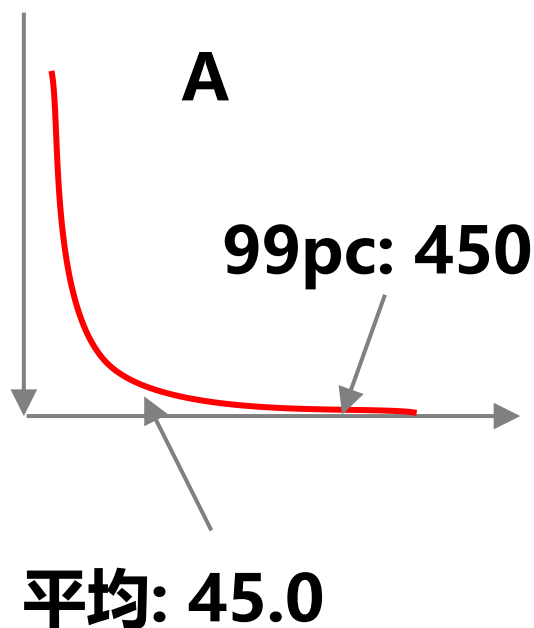
mean w/ 95% confidence interval



(SemiSpace 是最好的之一)

你有没有被一个幸运的异常值愚弄过？

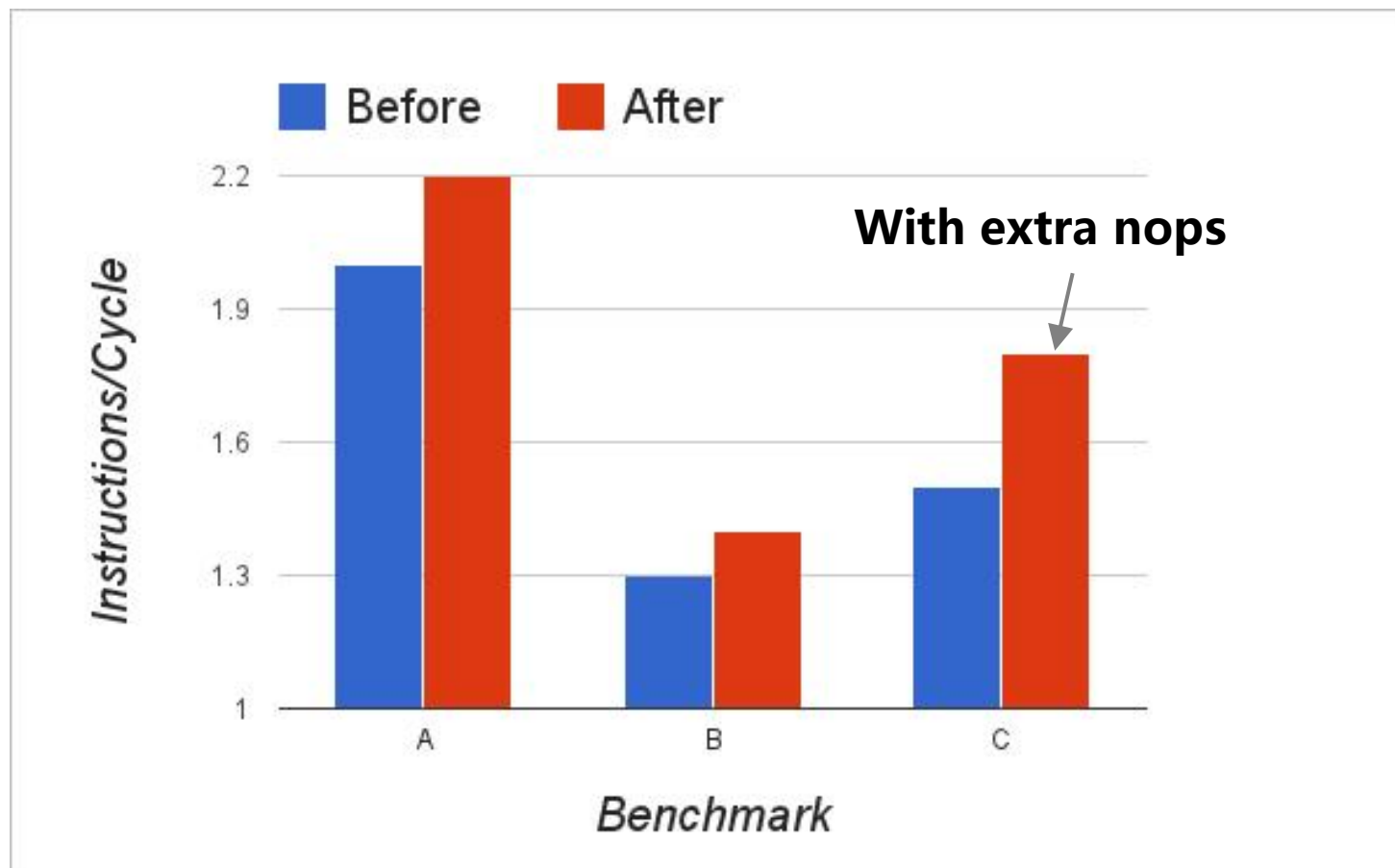
不恰当的数据分析



一次 Google 搜索 = 100 次 RPC
第 99 个百分位影响大多数请求!

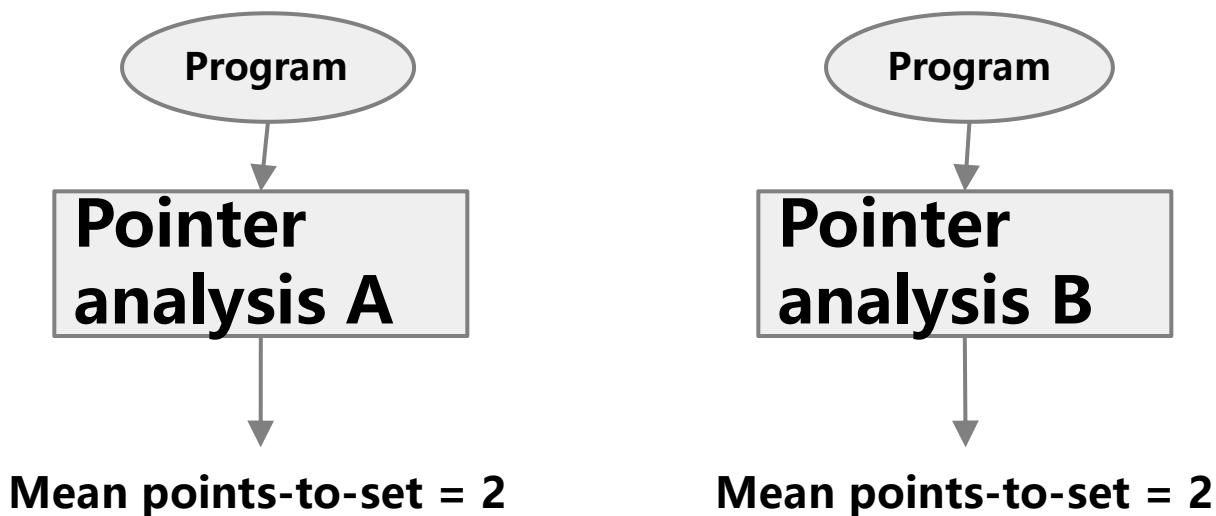
如果长尾延迟很重要，则求平均值 (Mean) 是不合适的!

指标不当



你是否曾经选择一个不是针对你优化目标而设置的度量指标？
(错把微指令当成指令进行统计)

指标不当



文章中的声明：B 比 A 更加简化，但能达到同样精度。



您是否使用过与“更好”不一致的指标？

罪过三：前后矛盾

定义：在不同的实验条件下，比较 A 和 B

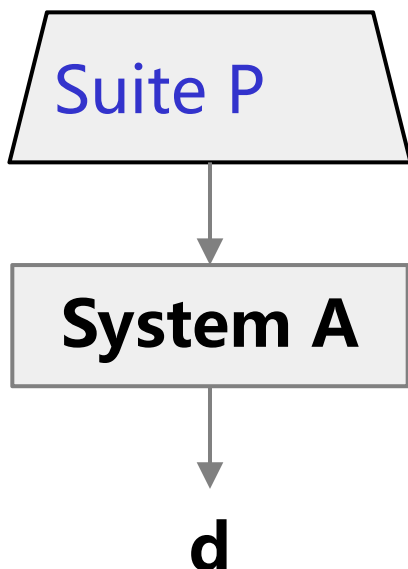
很常见到的拒稿：你这个实验拿苹果跟橙子比。（指代你比较的环境不合理）



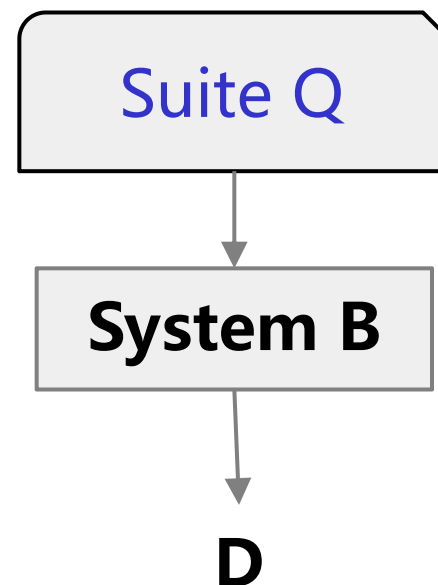
罪过三：前后矛盾

定义：在不同的实验条件下，比较 A 和 B

Experiment:
Baseline方法在P条件下;而我们方法在 Q条件下

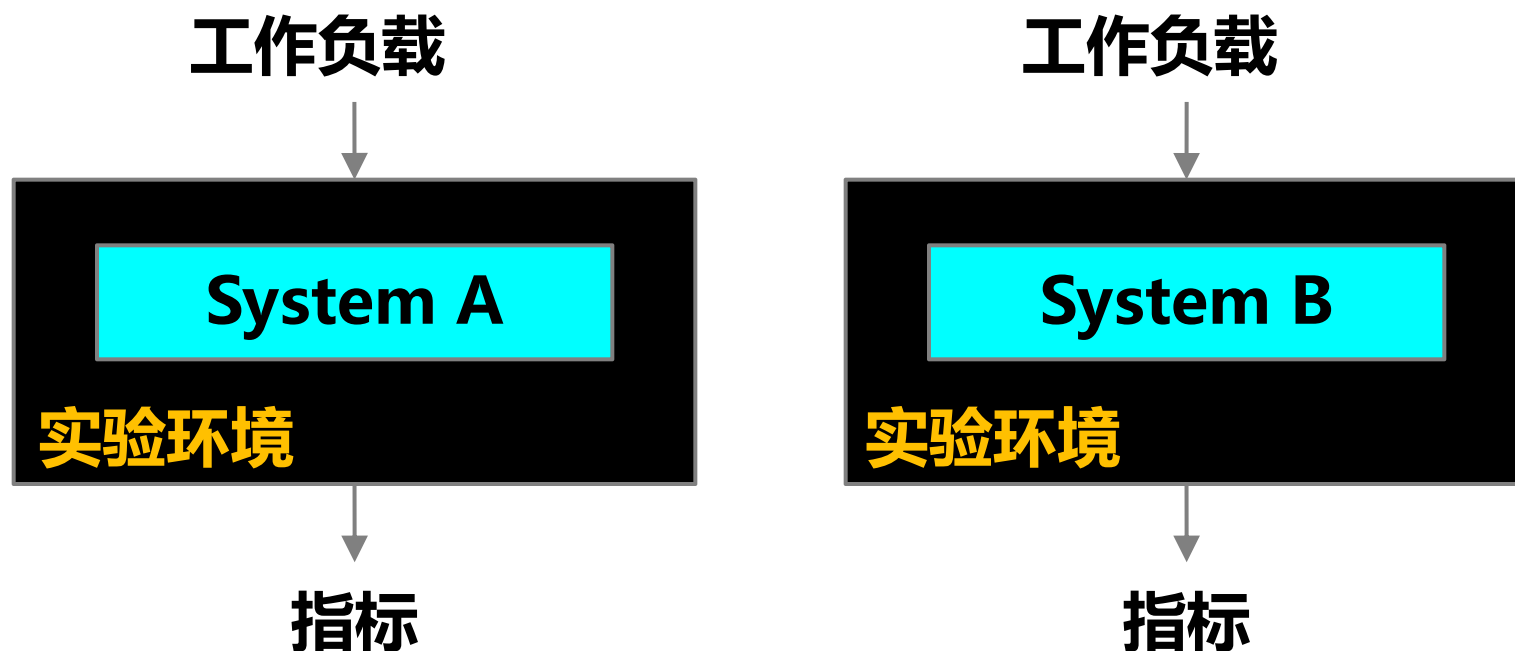


Claim:
由于结果D优于d,所以系统B 优于 A



实验条件的不一致（不完全对等），给出具有误导性的结论！

前后矛盾也并不明显

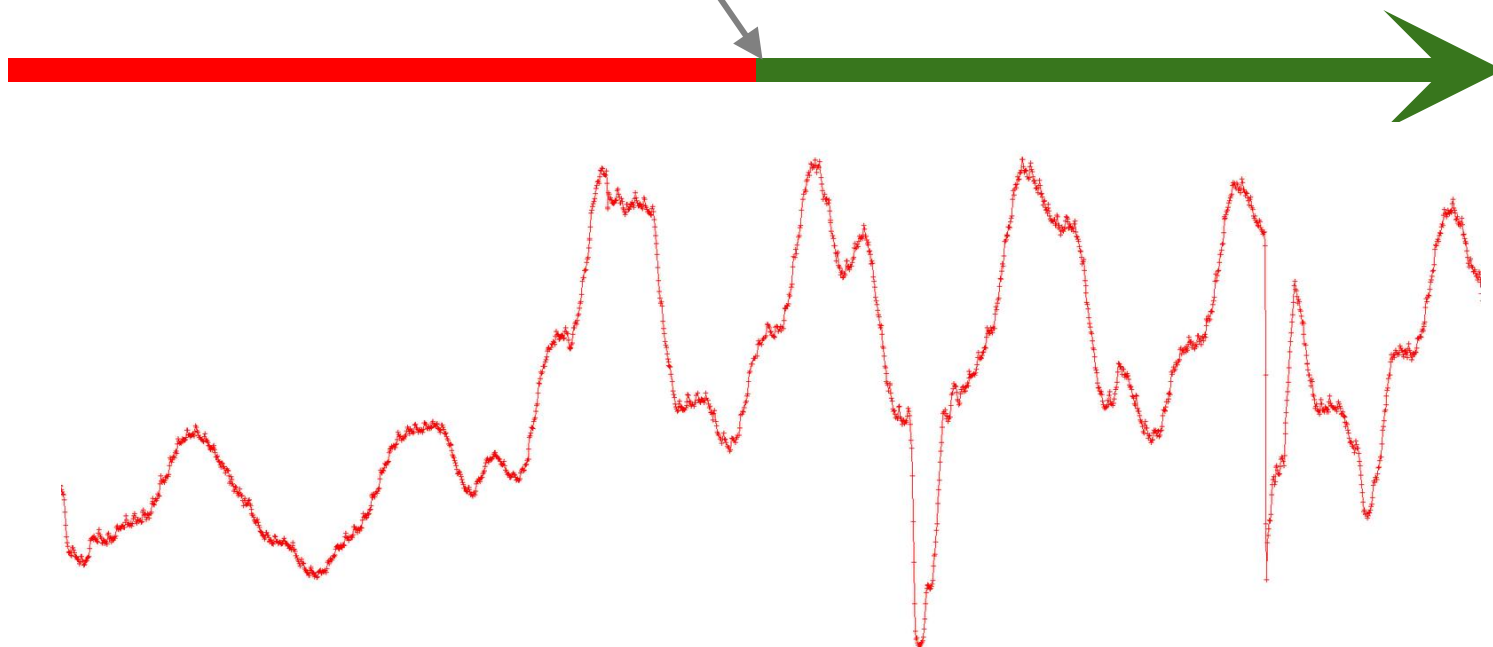


工作负载、实验环境和指标必须相同

不一致的工作负载

我想针对 Gmail 的新优化的效果进行评估

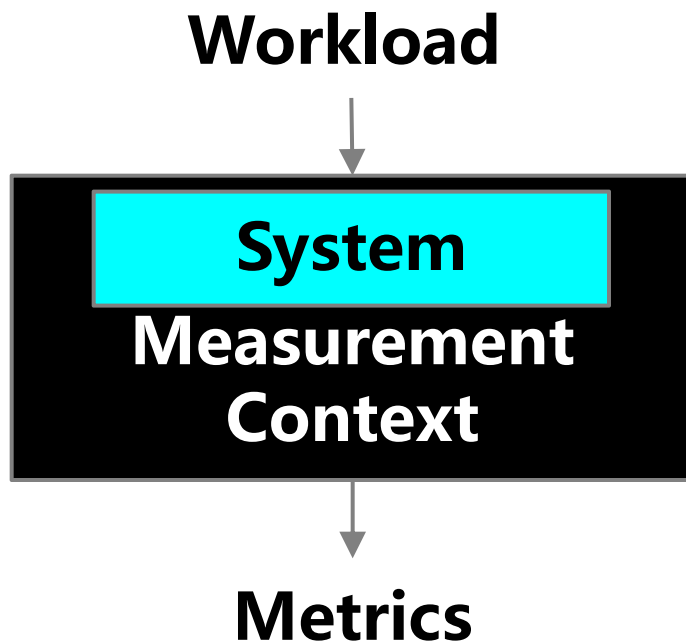
优化已启用



您的工作负载是否在优化前和优化后，发生过变化？

罪恶 4：不可重复性

Defn: 其他人无法重现你的实验



不可重复性使得更难识别所设计的实验是否可靠

忽略任何偏差会使结果无法重现——多多检查，再严谨一点

工作应该多检查几遍，反复重温

致命的四大罪

- 影响实验的各个方面
- 克服这些问题，需要创造力和勤奋！

Thanks Q&A