云平台性能优化

李玮

2017/8/7

1. 优化原则
2. 监控分析优化体系
3. 监测工具
4. 系统优化举例

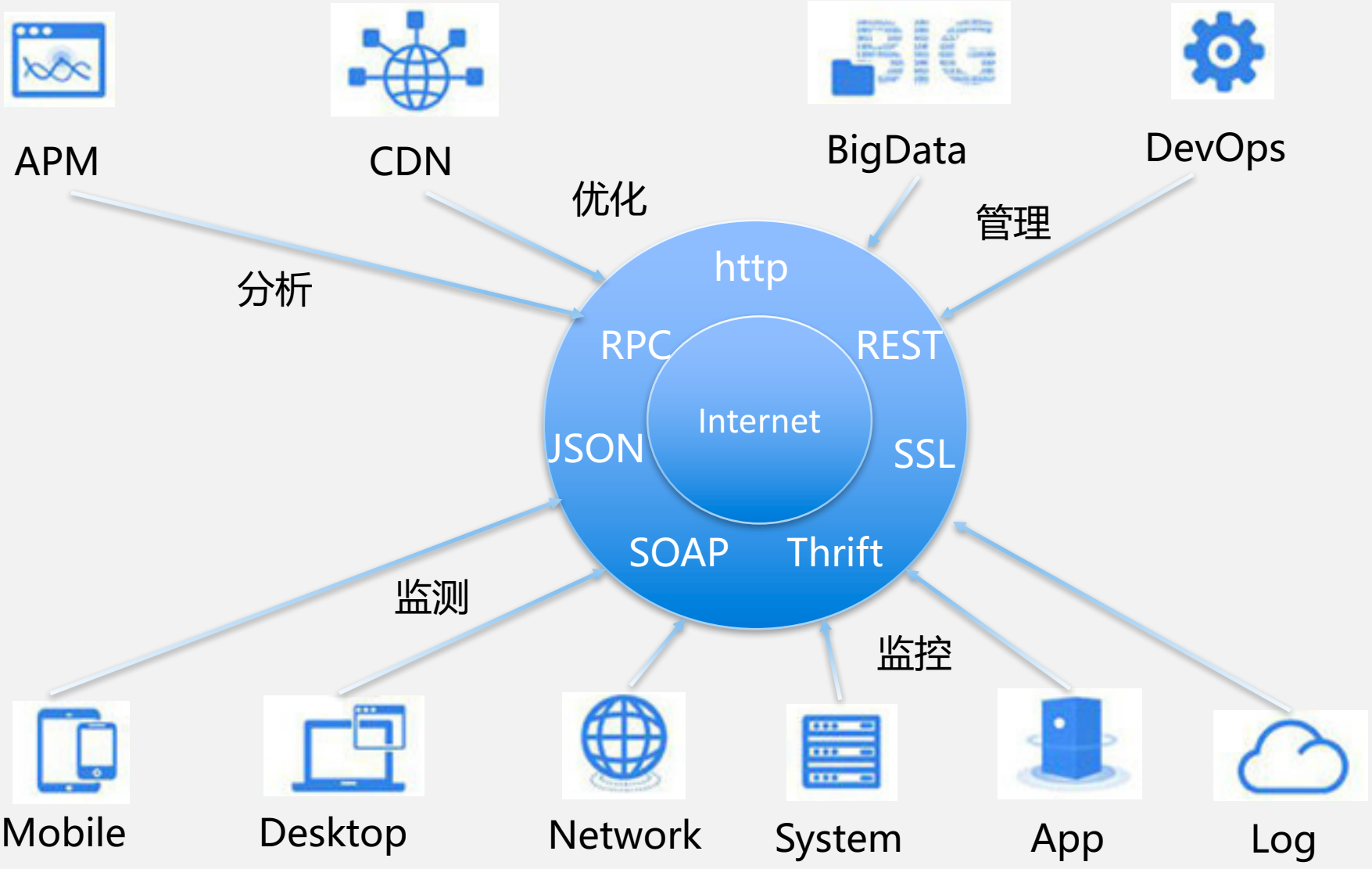
优化的优先级

什么是性能

“性能”其实是一个很宽泛的概念，在不同的语境下都可以有不同的含义，但如果一定要拉出一根墨绳，以我的理解，性能就是一个纯粹的时间概念。可以在更短的时间内完成一项特定的工作，就是性能高；也只有时间短才是性能高的充要条件。

换句话说就是使用更少的资源来完成相同的工作。

性能监测，分析，优化体系


APM
CDN
BigData
DevOps
优化
分析
管理
http
RPC
REST
Internet
JSON
SSL
SOAP
Thrift
监测
监控
Mobile
Desktop
Network
System
App
Log

影响性能的因素

代码及应用

开发语言瓶颈，研发底蕴，代码质量，迭代进程，团队成熟度...

服务器及云环境

硬件老化，硬件故障，配置差，不稳定，性能弱，成本因素...

移动

厂商和机型丰富，操作系统高度定制，网络复杂，信号变化，跨网交互普遍，移动性能优化意识和技术缺乏...

产品逻辑和用户行为

产品逻辑复杂，用户秒杀，大规模推广，高峰期访问

PC

用户端硬件配置，系统环境干扰，接入网复杂，恶意竞争

基础网络

多网割裂，南北互通，国际互通，用户分布明显，黑带宽，数据中心分布局限

性能木桶理论

PC、Mobile
Web App\Native App

服务器及云环境
运维、硬件、系统、日志

产品逻辑及用户行为
设计，结构，内容

基础网络
IDC、CDN、BGP、GSLB

代码及应用
前端、后端、移动端、安全

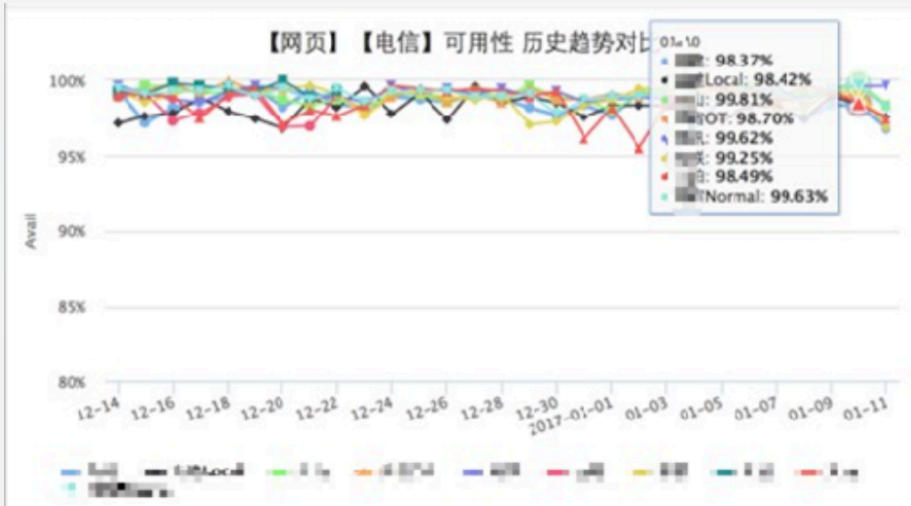
性能优化要遵循的

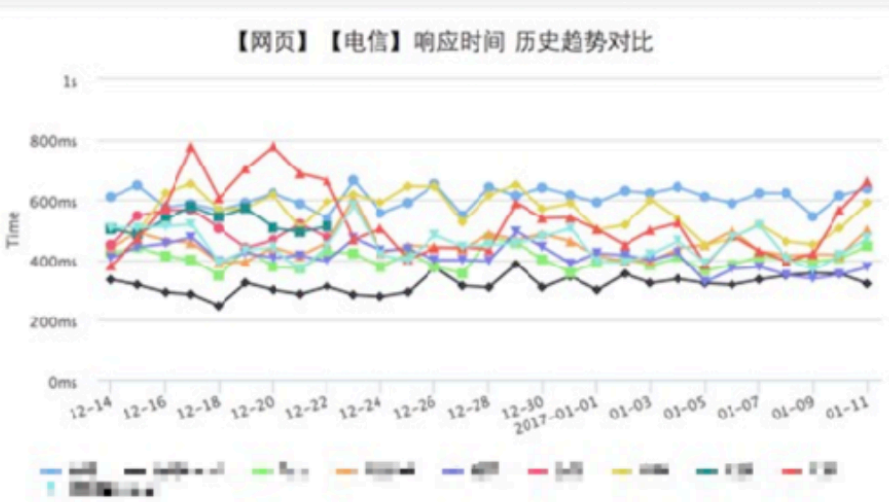
- 过早优化是万恶之源
- 优化顺序是从高层到底层
- 谨遵阿姆达尔定律（Amdahl ‘s law）

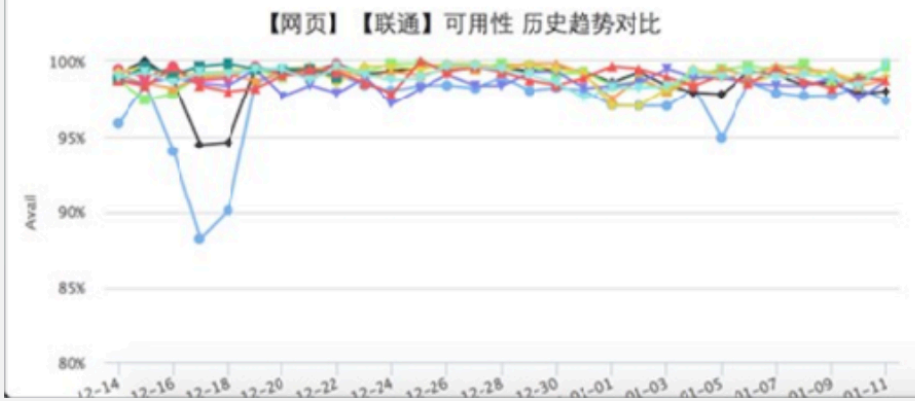
性能监测

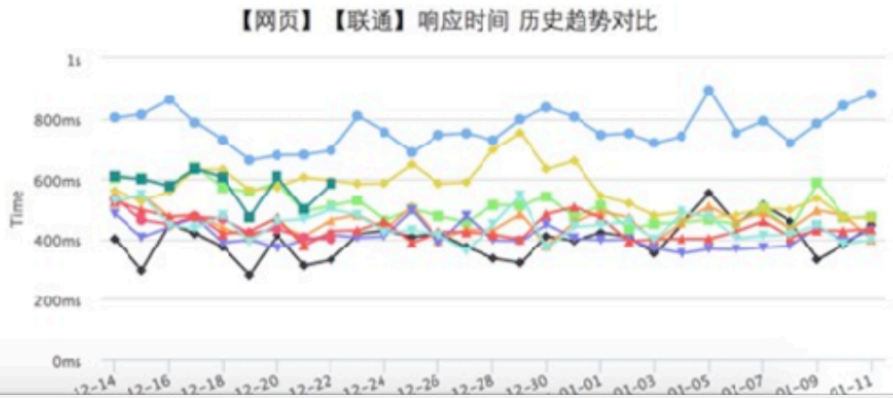
- 系统监测
- 日志监测
- 网络监测
- 存储监测

听云


【网页】【电信】可用性 历史趋势对比
Avail
100%
95%
90%
85%
80%
Local: 98.42%
Normal: 99.63%


【网页】【电信】响应时间 历史趋势对比
Time
1s
800ms
600ms
400ms
200ms
0ms


【网页】【联通】可用性 历史趋势对比
Avail
100%
95%
90%
85%
80%


【网页】【联通】响应时间 历史趋势对比
Time
1s
800ms
600ms
400ms
200ms
0ms

可复现

重复多次可以给出非常近似的测试结果。这意味着需要考虑尽量减少系统等外界因素的干扰，比如同时运行的某个资源占用大户，或者阻塞式IO，以及各种系统的限制，例如NUMA和CPU核数等。基准测试的目的并非是求出一个“平均”或者“最大最小”的性能指标，而是为性能测试工具提供一个良好的观察对象，以便发现和诊断性能瓶颈所在。

覆盖典型执行路径

无论什么类型的测试，其实都是对被测对象输入输出的管理。有些测试可以将被测对象看作黑盒子，但性能基准测试，必须要构建能够覆盖典型执行路径的输入。因为作为在使用环境中大量调用的代码，它们的性能是我们主要关心的内容。同时也是给测试工具发挥最大效用提供必要的条件。

C / C++性能检测工具

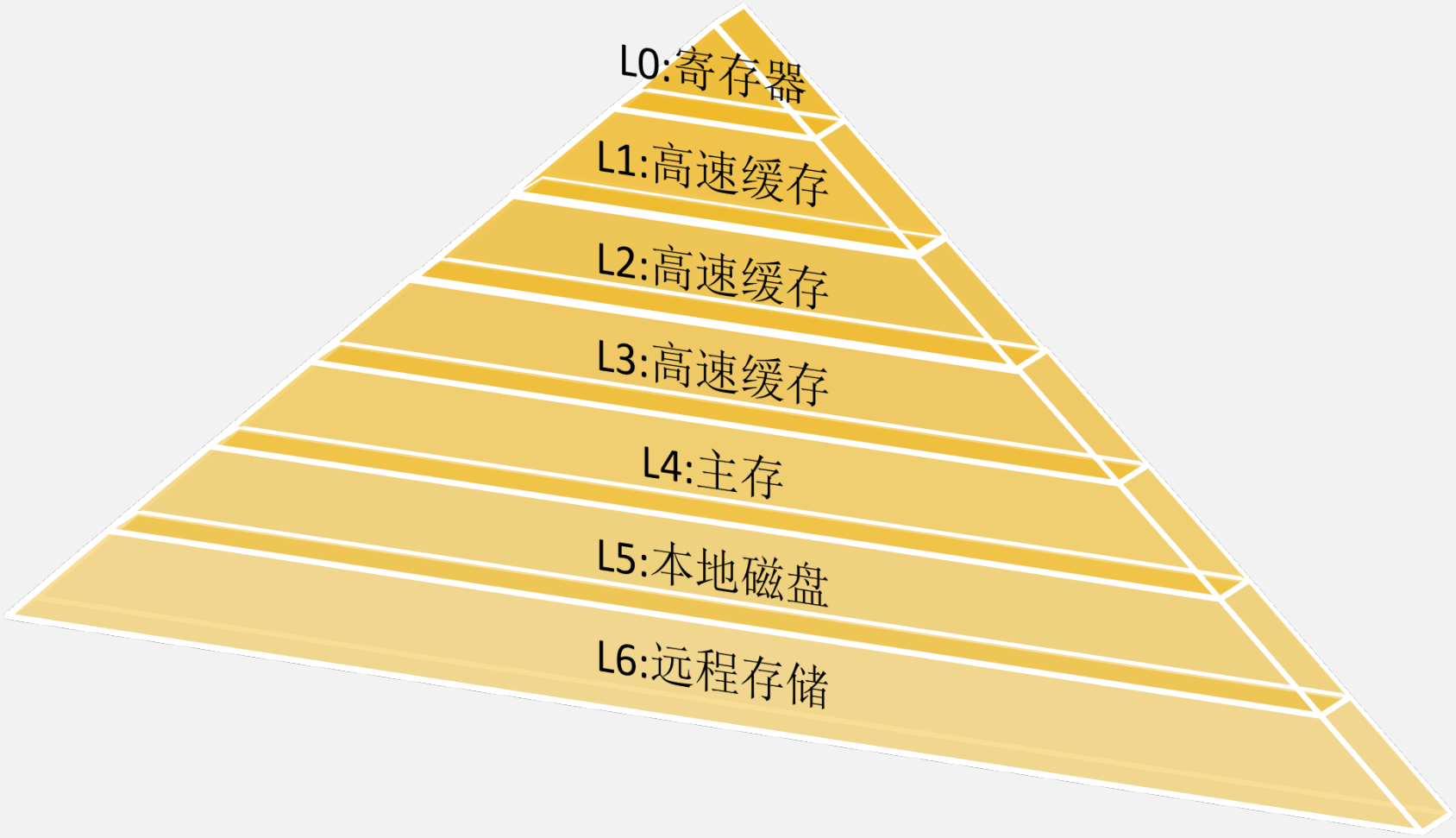
- SystemTap
- Flame Graph
- Perf
- Valgrind
  - Memcheck　　内存检查工具
  - Callgrind　　检查程序中函数调用过程中出现的问题
  - Cachegrind　　主要用来检查程序中缓存使用出现的问题
  - Helgrind/DRD　主要用来检查多线程程序中出现的竞争问题
  - Massif　　堆栈分析器

Intel的一些工具

- Intel® C++ Compiler C/C++编译器
- Intel® VTune™ Amplifier XE 性能调优工具
- Intel® Inspector 内存和线程debug工具
- Intel® Math Kernel Library 数学库
- Intel® Threading Building Blocks 线程库（TBB）
- Intel® Integrated Performance Primitives 高性能库（IPP）
- Intel® Data Analytics Acceleration Library 关于数据挖掘分析的高性能库

- 使用局部性原理
- 指令流水（Instruction pipeline）
- SIMD（单指令流多数据流/Single Instruction Multiple Data）

存储器的层次结构



- 通常，一个编写良好的（优化的）程序具有良好的局部性
  - 时间局部性（Temporal locality）：如果被访问过的存储器地址在较短时间内被再次访问，则程序具有良好的时间局部性。
  - 空间局部性（Spatial locality）：如果程序访问某个存储器地址后，又在较短时间内访问临近的存储器地址，则程序具有良好的空间局部性。两次访问的地址越接近，空间局部性越好。

局部性原理

一个良好的局部性程序

```
int sum(int a[M][N])
{
    int i,j,sum = 0;
    for(i=0;i<M;++i)
        for(j=0;j<N;++j)
            sum += a[i][j];
    return sum;
}
```

一个局部性很差程序

```
int sum(int a[M][N])
{
    int i,j,sum = 0;
    for(j=0;j<N;++j)
        for(i=0;i<M;++i)
            sum += a[i][j];
    return sum;
}
```

指令流水（Instruction pipeline）

指令流水线（英语：Instruction pipeline）是为了让计算机和其它数字电子设备能够加速指令的通过速度（单位时间内被运行的指令数量）而设计的技术。

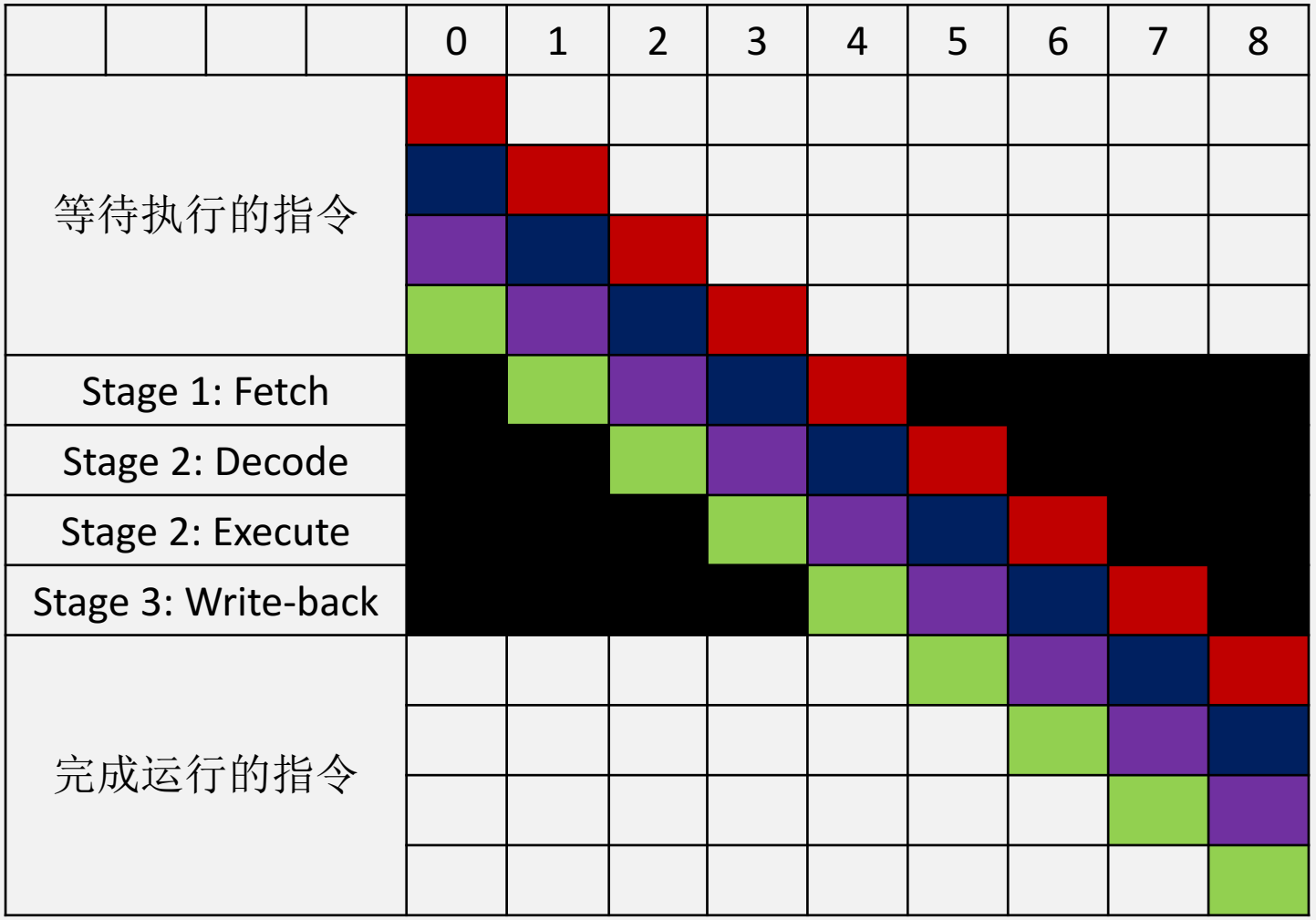
指令流水（Instruction pipeline）

Instr. No.	Pipeline Stage						
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
Clock Cycle	1	2	3	4	5	6	7

RISC机器的五层流水线示意图（IF：读取指令，ID：指令解码，EX：运行，MEM：内存访问，WB：写回寄存器）

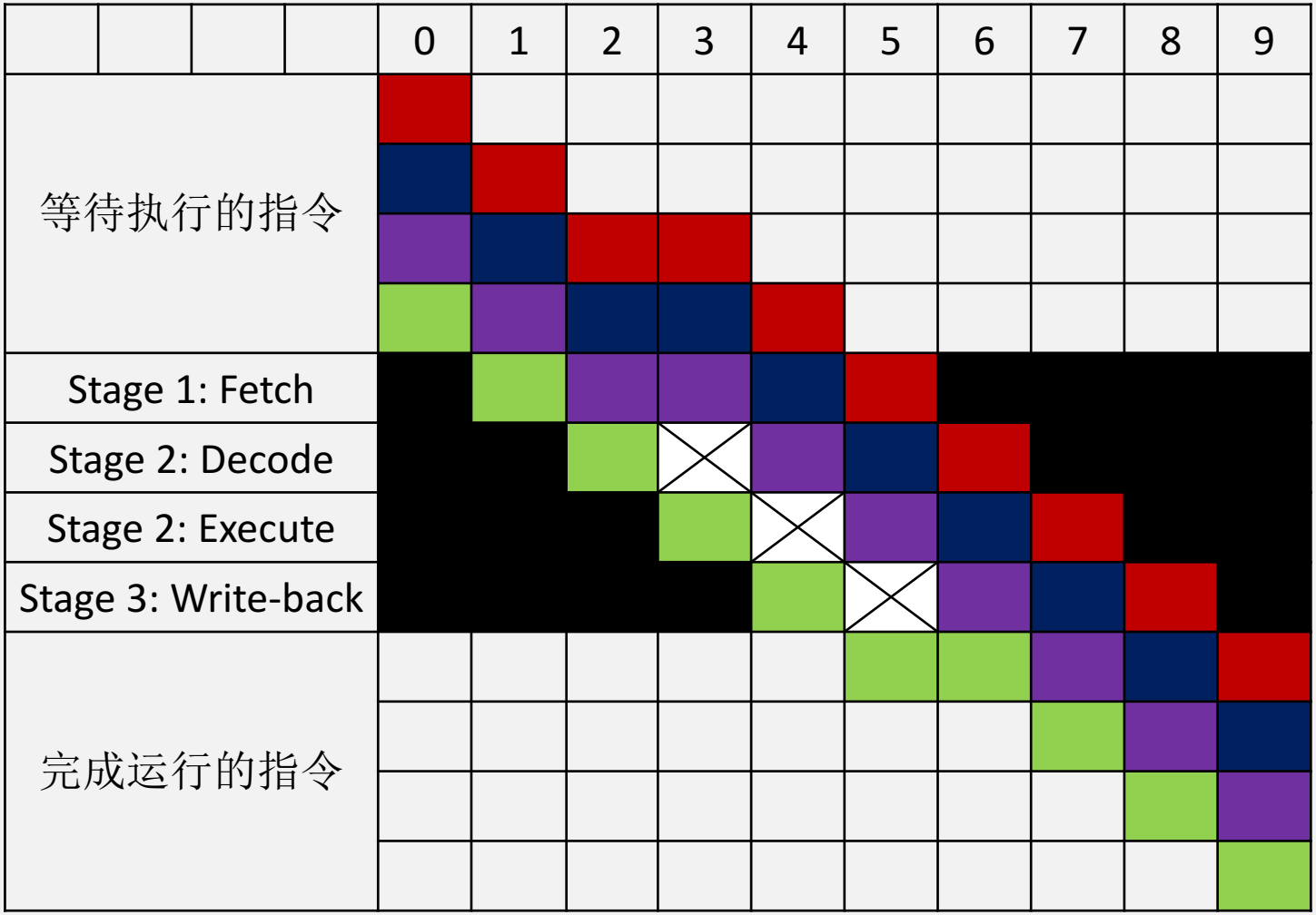
指令流水（Instruction pipeline）

- 一般流水示意图
- 1 绿色 Fetch
- 2 紫色 Decode
- 3 蓝色 Execute
- 4 红色 Write-back



指令流水中断

- 气泡流水示意图
- 1 绿色 Fetch
- 2 紫色 Decode
- 3 蓝色 Execute
- 4 红色 Write-back



指令流水冲突

资源冲突：流水线上的一个指令需要使用已经被另一个指令占据的资源

数据冲突

- 指令层的数据冲突：指令需要的数据还没有计算出来
- 传输层的数据冲突：指令需要的寄存器(register)内容还没有被存入寄存器

控制流冲突：流水线必须等待一个有条件Goto指令是否会被执行

解决指令流水冲突

乱序执行与内存栅栏
之间的关系
以及无锁编程

资源冲突

- 增加功能单位部件
- 指令乱序执行

控制流冲突

- 分支预测，减少流水线清除

解决指令流水冲突

指令层的数据冲突：指令需要的数据还没有计算出来

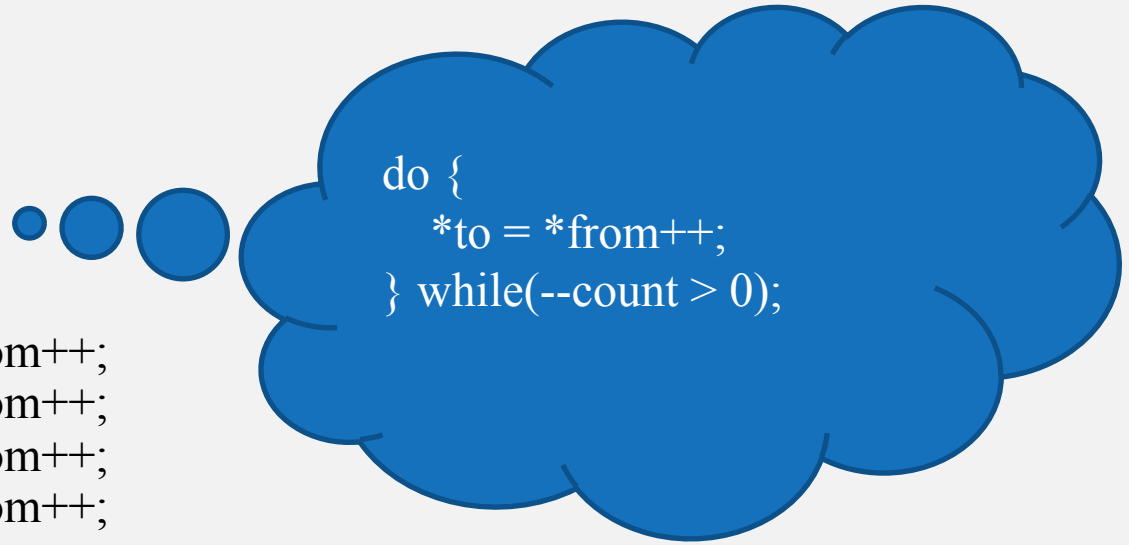
- 避免上下文依赖的计算

传输层的数据冲突：指令需要的暂存器(register)内容还没有被存入暂存器

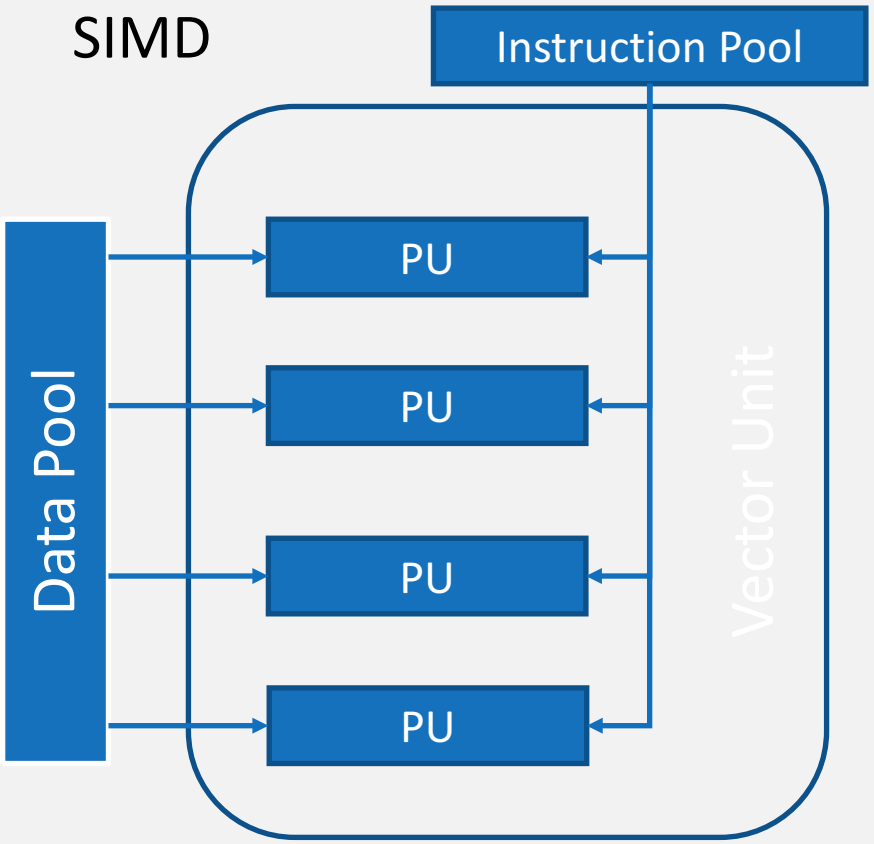
- 局部性原理

达夫设备（Duff’s device)

```
send(to, from, count)
register char *to, *from;
register count;
{
    register n = (count + 7) / 8;
    switch(count % 8) {
        case 0:     do { *to = *from++;
        case 7:          *to = *from++;
        case 6:          *to = *from++;
        case 5:          *to = *from++;
        case 4:          *to = *from++;
        case 3:          *to = *from++;
        case 2:          *to = *from++;
        case 1:          *to = *from++;
                } while(--n > 0);
        }
}
```

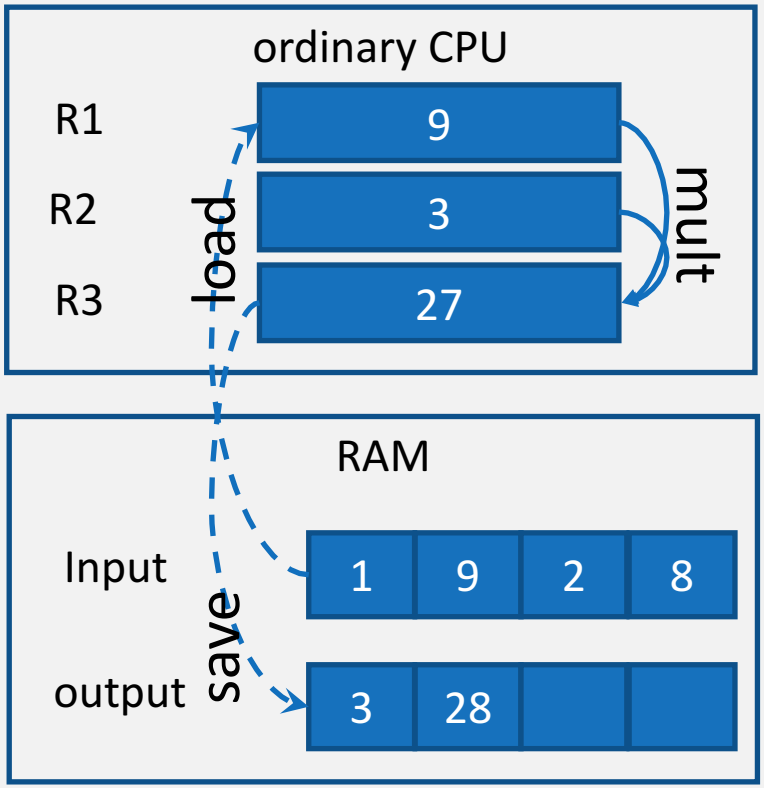


SIMD（单指令流多数据流/Single Instruction Multiple Data）

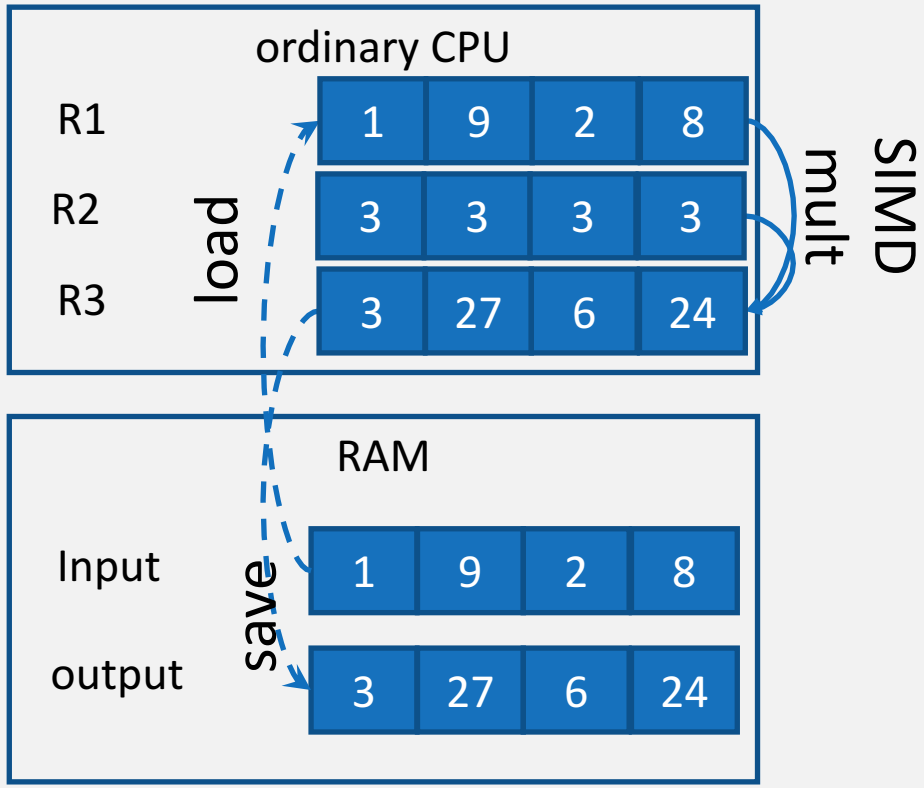


SIMD（单指令流多数据流/Single Instruction Multiple Data）

SISD计算方式



SIMD计算方式



THANK YOU