第一章　计算机概述

教　师：　夏文

计算机科学与技术学院

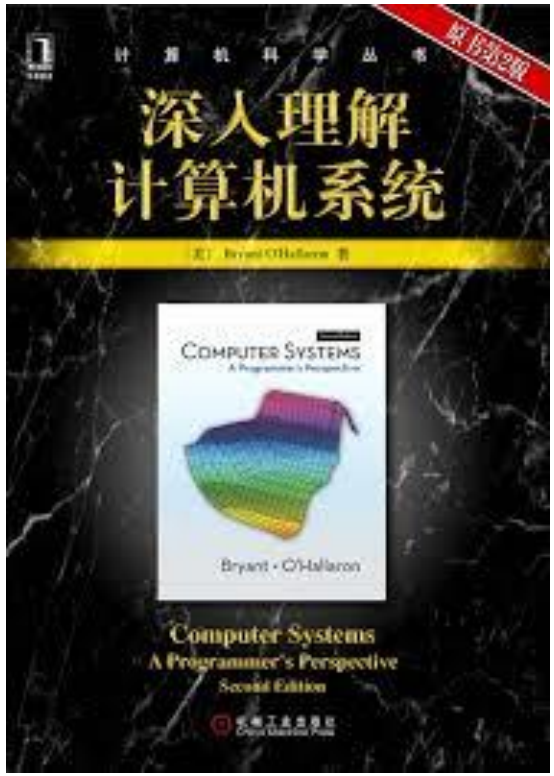
哈尔滨工业大学（深圳）

个人简介

- 主页：http://faculty.hitsz.edu.cn/xiawen
- Tel：18575530070
- 邮箱：xiawen@hit.edu.cn
- 办公室：信息楼L1704

教材

- Randal E. Bryant and David R. O’ Hallaron,
  - *Computer Systems: A Programmer’ s Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016　深入理解计算机系统 3-机械工业出版社
  - http://csapp.cs.cmu.edu
  - 这本书对这门课很重要!
    - 练习题中有典型的考试题目
    - 反复精读，反复巩固



- 计算机系统基础（供参考）
  - 南京大学 袁春风

课件以及交流

- 课程 Web网站: **http://www.cs.cmu.edu/~213**
  - CMU完整的课程资料,包括PPT、实验、课外阅读、视频等

- QQ群：2020夏计算机系统-HITSZ
  - 课件
  - 网上答疑
  - 通知

- 学习要旨
  - 反复学习，反复巩固
  - 触类旁通，学以致用

考研大纲（1）

- 专业课考试时间180分钟，满分150分。包括计算机系统、计算机网络、数据结构与算法共三部分，每部分各70、40、40分。

- 《计算机系统》考试要求
  - 要求考生全面掌握现代计算机系统软硬件各层次的组成结构与工作原理，深入理解程序的机器级表示、代码生成、加载执行、存储与交互等技术。
  - 要求考生从计算机系统的角度进行程序优化、缺陷分析、故障恢复及攻击防范，并设计正确、可靠、高性能的计算机程序，来解决复杂计算机工程问题。

考研大纲（2）

- 1）计算机系统基本概念
  - a: 计算机系统的层次结构
  - b: 程序生成与运行的基本原理与工作过程
  - c: 计算机系统的分析评价方法
- 2）信息表示与处理
  - a: 信息编码与存储
  - b: 整数表示与运算
  - c: 浮点数表示与运算
- 3) 程序的机器级表示
  - a: 机器的结构与运行，指令系统与寻址方式
  - b: 基本数据类型与操作的机器级表示
  - c: 基本程序结构的机器级表示
  - d: 缓冲器溢出原理与漏洞攻防
- 4）处理器体系结构
  - a: ISA的组成与设计
  - b: 顺序结构CPU实现与微操作
  - c: 流水线CPU基本原理与技术

考研大纲（3）

- 5）程序性能优化
  - a: 面向编译器的程序优化方法
  - b: 面向流水线CPU、超标量CPU、向量CPU的程序优化方法
- 6）存储器层次结构
  - a: 存储器层次结构与局部性原理
  - b: 高速缓存技术与基于存储器的程序优化技术
- 7）链接
  - a: 符号解析、静态连接与重定位
  - b: 共享库、动态链接与库打桩
- 8）异常控制流
  - a: 异常与进程
  - b: 信号与处理
- 9）虚拟存储器
  - a: 地址空间映射与虚拟存储器概念
  - b: 虚拟存储器系统构成与实现
  - c: 动态存储器分配技术

本章重点

- 存储器的层次结构
- 初步了解程序的生成和执行过程，在后面的课程会仔细讲解
- 计算机系统的层次模型和抽象表示
- 经典例题

本章目录

1. 课程主题/主旨/目的/目标
2. 五个事实/现实
3. 可执行程序是怎么生成的（程序员的角度）
4. 可执行程序是怎么运行的（程序员的角度）
5. 怎么优化源程序（程序员的角度）
6. 计算机系统层次模型
7. 本课程在CSE/SE课程体系中的地位
8. 课程考核

一、课程目标:

抽象很好但别忘记现实

- **多数计算机科学与计算机工程的课程强调抽象**
  - 抽象数据（类）型
  - 渐进分析Asymptotic analysis
- **抽象是有限制的**
  - 特别是在bug（程序缺陷-故障/错误）面前
  - 需要理解底层实现的细节
- **学完本课程的有用的收获**
  - 成为更加有效地程序员
    - 能够发现并有效地排除bug
    - 能理解并调整程序性能
  - 把计算机系统相关课程串起来
    - 编译原理、操作系统、计算机网络、计算机体系结构、嵌入式系统、存储系统、计算机组成原理、汇编语言、等.

理解盒子里面的东西

- 张晓东：跟热点可能浪费资源，盒子里的东西更重要
  - **计算机学科变化非常大，但所有的变化都建立在核心技术和基础学科之上**
  - 计算机影响着各行各业，没有一个学科有这样大的影响力。比起在学科建设中跟风、追热点，更应注重打好基础。**计算机科学不是空中楼阁，如果基础打得足够好，不用担心怎么变，甚至可以引领学科的进展。**
  - **核心技术的意思是学习计算机‘盒子里的东西’**，而不仅是学怎么用这个盒子”。“国内大学的计算机专业大多数以应用为主，只有几个顶尖大学重视核心和基础学科。”张晓东直言。
  - 对于近来兴起的人工智能本科专业，他看得较为平淡。“人工智能的核心是通过数据分析找到特殊的模式，并快速地做出判断。当今，计算机的计算能力和数据量非常大，人工智能可以做很多的事情，但也有相当的局限性。”“对于计算机学科而言，大学四年能够打好基础就不错了。如果真有资质和能力，什么时候做深入的专科研究都不晚。”

华为事件

没有伤痕累累，哪来皮糙肉厚，英雄自古多磨难

心声社区 今天

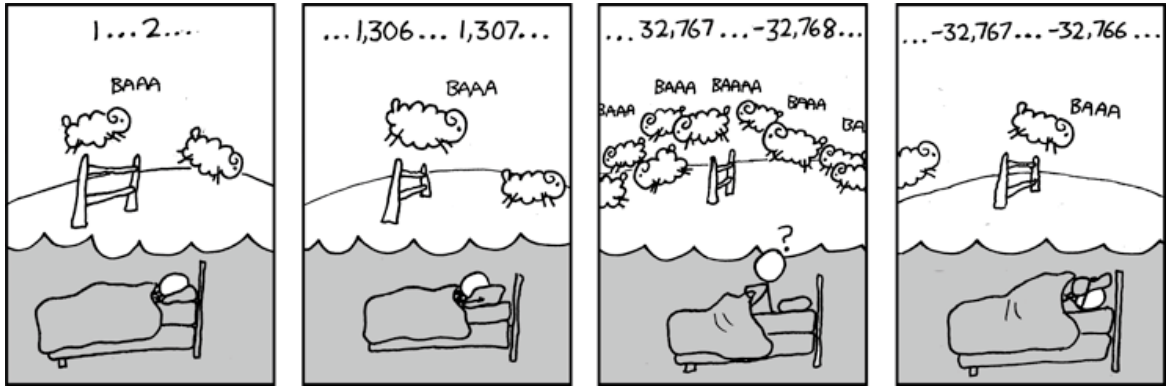
回头看，崎岖坎坷

向前看，永不言弃

二、伟大现实

伟大现实#1: int不是整数, float不是实数

- **例 1: $x^2 \geq 0$?**
  - Float’ s: Yes!



  - Int’ s :
    - 40000 * 40000 = 1600000000
    - 50000 * 50000 = ??
- **例 2: (x + y) + z = x + (y + z)?**
  - 无符号/有符号 Int: Yes!
  - 浮点数Float:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

浮点数的故事

- 为什么要单独拿出来说浮点数?

- 浮点数不等于实数
  - 1991年，美国爱国者导弹误炸28名士兵
  - 1996年，阿丽亚娜5(Ariane5)型火箭发射失败

- 浮点数热门？大数据，HPC，AI

计算机的算法/算术

- **不生成随机值**
  - 算数操作有重要的数学特性
- **不要假设所有的 “通常” 数学特性**
  - 因为数据表示的有限性
  - 整数操作满足 “环ring” 特性
    - 交换律, 结合律 , 分配律
  - 浮点操作满足 “排序ordering” 特性
    - 单调性, 符号值
- **观察**
  - 要理解哪一种抽象应用在哪些上下文中
  - 针对编译器程序员和严肃的应用程序员的重要事项

伟大现实 #2: 你不得不懂汇编

- **有可能是, 你永远不用汇编语言写程序**
  - 编译器比你更好更耐心
- **但是: 要理解汇编是机器级执行模型的关键**
  - Bug面前程序的行为
    - 高级语言模型会失败
  - 调整程序性能
    - 理解由编译器做/不做的优化
    - 理解程序低效的根源
  - 实现系统软件
    - 编译器把机器代码作为目标
    - 操作系统要管理进程状态
  - 创造/对抗恶意软件（malware）
    - x86 汇编是很好的语言选择

伟大现实#3:存储事宜

RAM随机存储器是一个非物理抽象

- **存储器不是无限的**
  - 存储器需要分配与管理
  - 很多应用是存储支配/控制的
- **存储引用错误特别致命**
  - 在时间和空间方面效果都不友好
- **存储器性能是不一致的**
  - Cache与虚拟存储器的效应能大大影响程序性能
  - 针对存储系统的特点，调整程序，能带来速度大幅的提升

例：存储引用Bug

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

fun(0) -> 3.14
fun(1) -> 3.14
fun(2) -> 3.1399998664856
fun(3) -> 2.00000061035156
fun(4) -> 3.14
fun(6) -> Segmentation fault

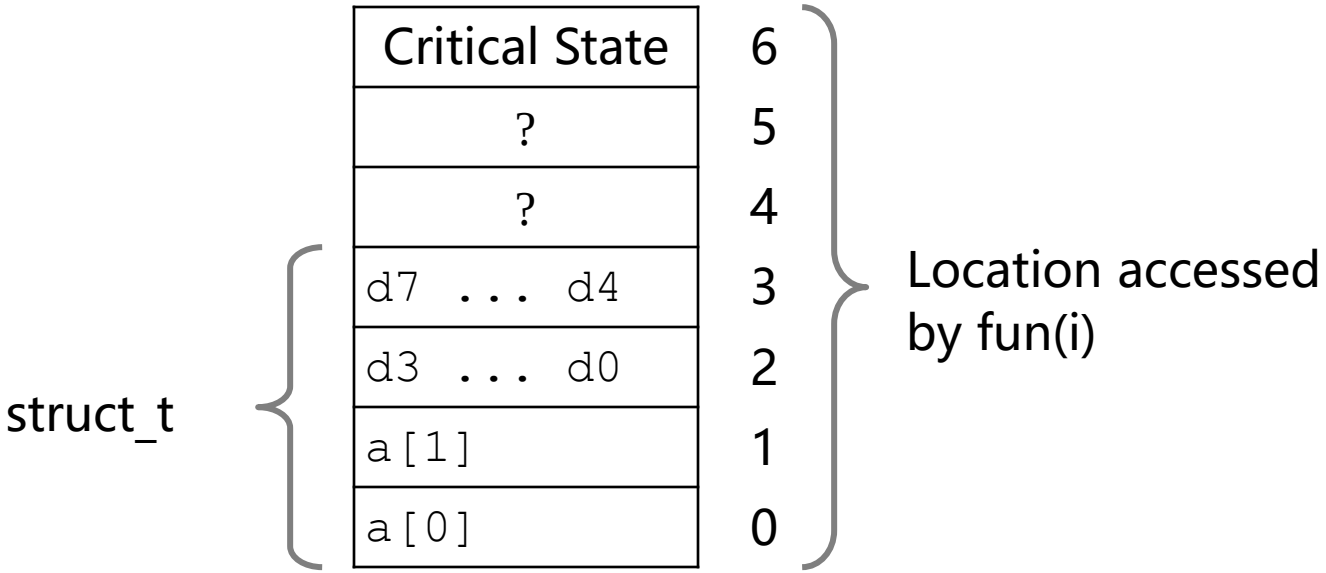
- 结果是特定系统的

例：存储引用Bug（续）

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0) ->        3.14
fun(1) ->        3.14
fun(2) ->        3.1399998664856
fun(3) ->        2.00000061035156
fun(4) ->        3.14
fun(6) ->        Segmentation fault
```

注释:



存储引用错

- **C and C++ 不提供任何存储保护**
- 数组访问的越界
  - 无效指针值
  - 滥用 malloc/free
- **导致险恶/恶意的bug**
  - Bug是否产生效果，依赖于系统或编译器
  - 远距离的行为(Action at a distance)
    - 崩溃的目标逻辑上与你正访问的不相干
    - 可能在bug生成很久才被第一次观察到Bug的影响
- **我能处理啥?**
  - 用 Java, Ruby, Python, ML, …进行编程
  - 理解可能会出现的交互
  - 使用或开发工具来发现引用错 (e.g. Valgrind)

伟大现实#4: 性能比渐进复杂性/时间复杂度更重要!

- **常数因子也有关系!**
- **即使是精确的操作数也无法预测性能**
  - 很容易能看到，代码编写不同，会引起10:1 性能变化
  - 一定要多层次优化: 算法,数据表示, 过程, 循环
- **优化性能一定要理解系统**
  - 程序是怎么编译和执行的
  - 怎样测量系统性能和定位瓶颈
  - 在不破坏代码模块化与整体性下，怎么改进性能

例：内存系统性能

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

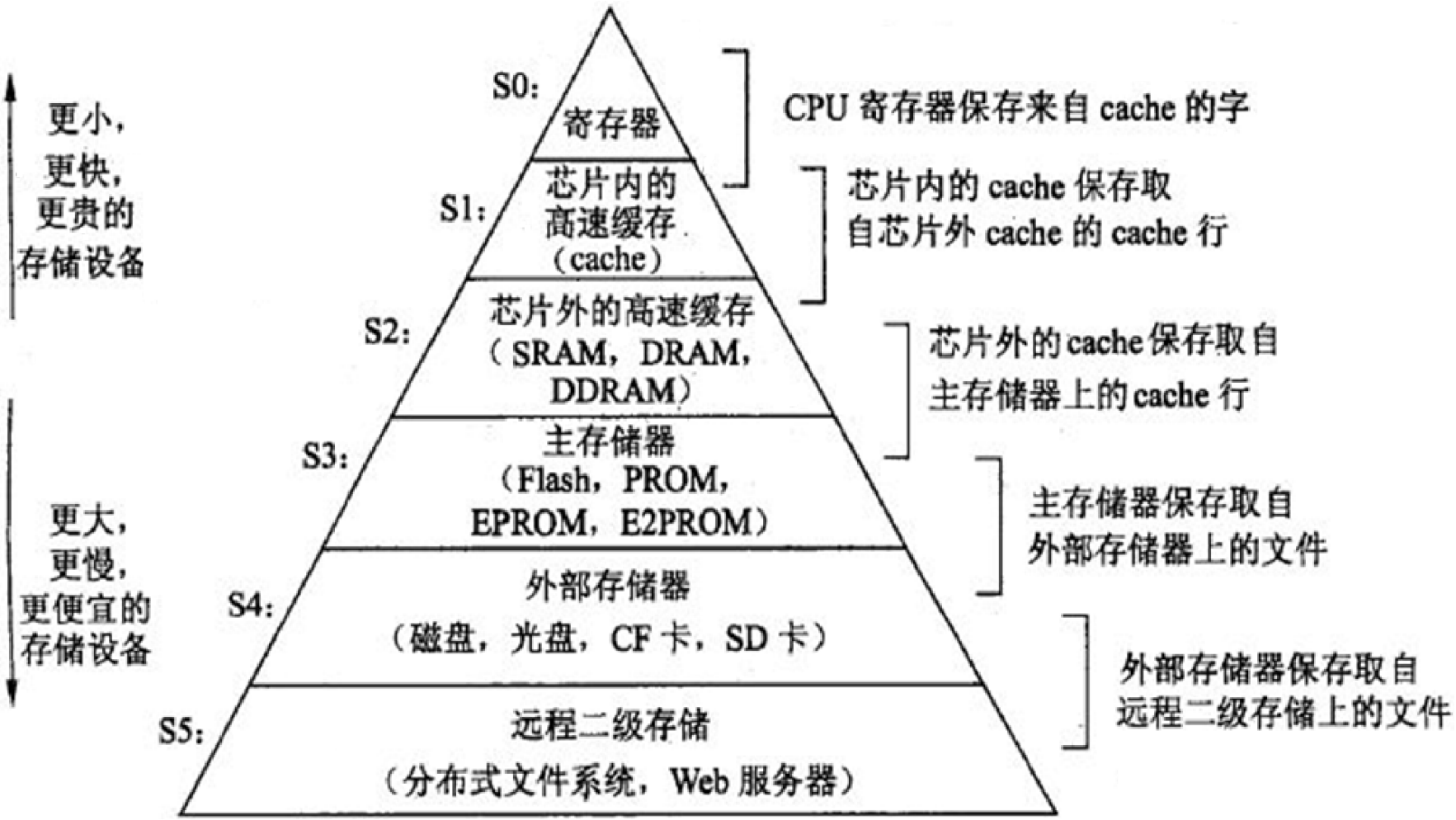
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms 81.8ms

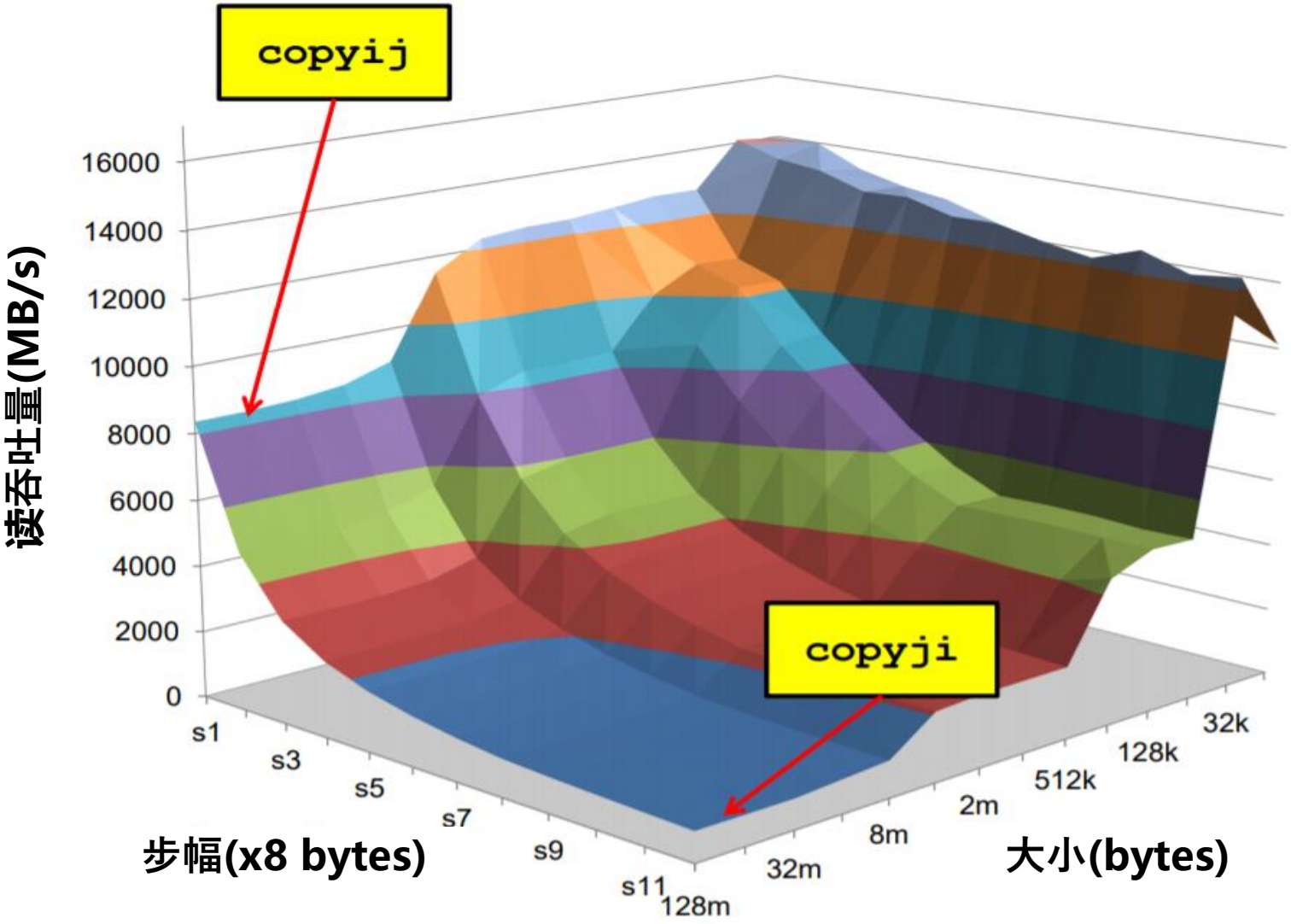
2.0 GHz Intel Core i7 Haswell

- 存储器的层次化组织
- 性能依赖于访问模式
  - 包括怎样遍历多维数组

存储器层次结构

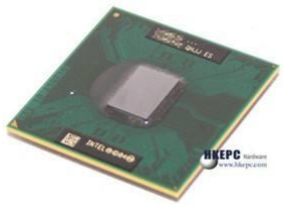

更小，
更快，
更贵的
存储设备
更大，
更慢，
更便宜的
存储设备
S0：
寄存器
S1：
芯片内的
高速缓存
（cache）
S2：
芯片外的高速缓存
（SRAM，DRAM，
DDRAM）
S3：
主存储器
（Flash，PROM，
EPROM，E2PROM）
S4：
外部存储器
（磁盘，光盘，CF 卡，SD 卡）
S5：
远程二级存储
（分布式文件系统，Web 服务器）
CPU 寄存器保存来自 cache 的字
芯片内的 cache 保存取
自芯片外 cache 的 cache 行
芯片外的cache保存取自
主存储器上的cache 行
主存储器保存取自
外部存储器上的文件
外部存储器保存取自
远程二级存储上的文件

为什么性能不同


copyij
copyji
读吞吐量(MB/s)
16000
14000
12000
10000
8000
6000
4000
2000
0
步幅(x8 bytes)
s1
s3
s5
s7
s9
s11
大小(bytes)
32k
128k
512k
2m
8m
32m
128m

存储器示例


numonyx
Omneo P8Q PCM
numonyx
Omneo™ P8P PCM




Kingston
SMS200S3/120G
120GB4000008
S5FAM008 - 1117
Made in TAIWAN
DC+3.3V 0.35A

存储器属性以及发展趋势

- 靠CPU侧：SRAM + DRAM

TABLE I: Characteristics of Different Types of Memory

Category	Read Latency (*ns*)	Write Latency (*ns*)	Endurance (*# of writes per bit*)
SRAM	2-3	2-3	∞
DRAM	15	15	10^{18}
STT-RAM	5-30	10-100	10^{15}
PCM	50-70	150-220	10^{8}-10^{12}
Flash	25,000	200,000-500,000	10^{5}
HDD	3,000,000	3,000,000	∞

伟大现实#5: 计算机比执行程序做的多得多

- **要进行数据的输入输出**
  - I/O系统对程序可靠性与性能很关键
- **要通过网络互相通讯**
  - 网络环境下出现了很多系统级问题
    - 自主进程的并发操作
    - 拷贝不可靠的媒体
    - 跨平台的兼容性
    - 复杂的性能问题

三、可执行程序的生成

经典的“ hello.c ” C-源程序

```
#include <stdio.h>

int main()
{
printf("hello, world\n");
}
```

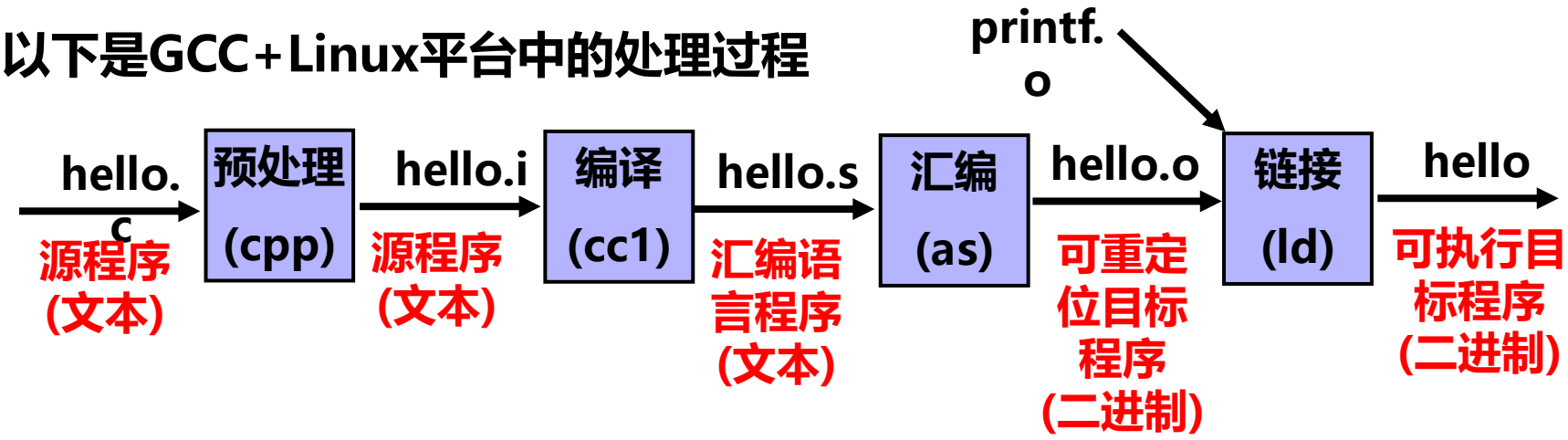
hello.c的ASCII文本表示

#	i	n	c	l	u	d	e	SP	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	SP	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	SP	SP	SP	SP	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	SP	w	o	r	l	d	\	n	"	)	;	\n	SP
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	32
SP	SP	SP	r	e	t	u	r	n	SP	0	;	\n	}	\n	
32	32	32	114	101	116	117	114	110	32	48	59	10	125	10	

计算机不能直接执行hello.c!

功能：输出“hello,world”

以下是GCC+Linux平台中的处理过程

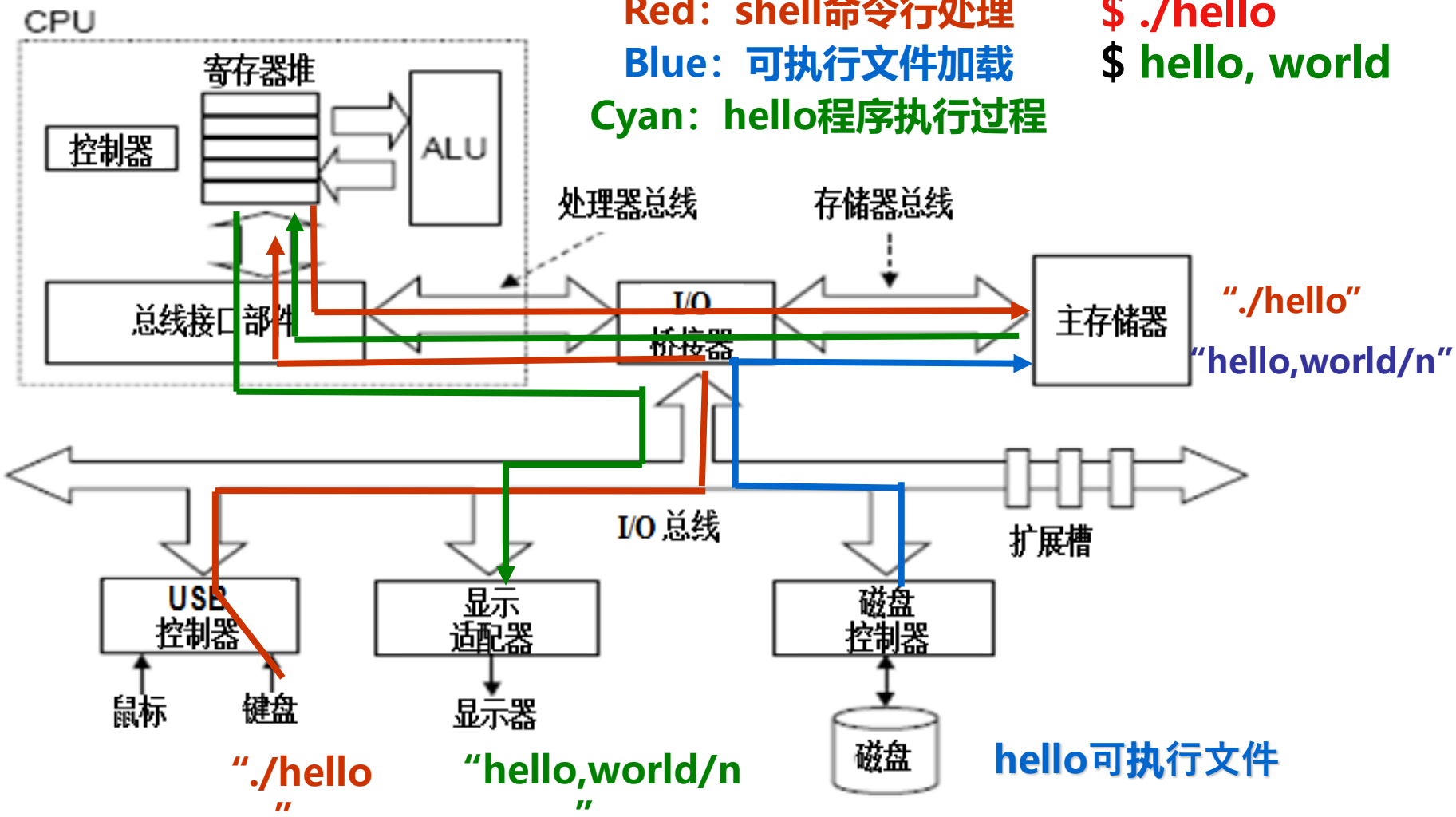


四、可执行程序的执行

Red：shell命令行处理　　$ ./hello

Blue：可执行文件加载　　$ hello, world

Cyan：hello程序执行过程

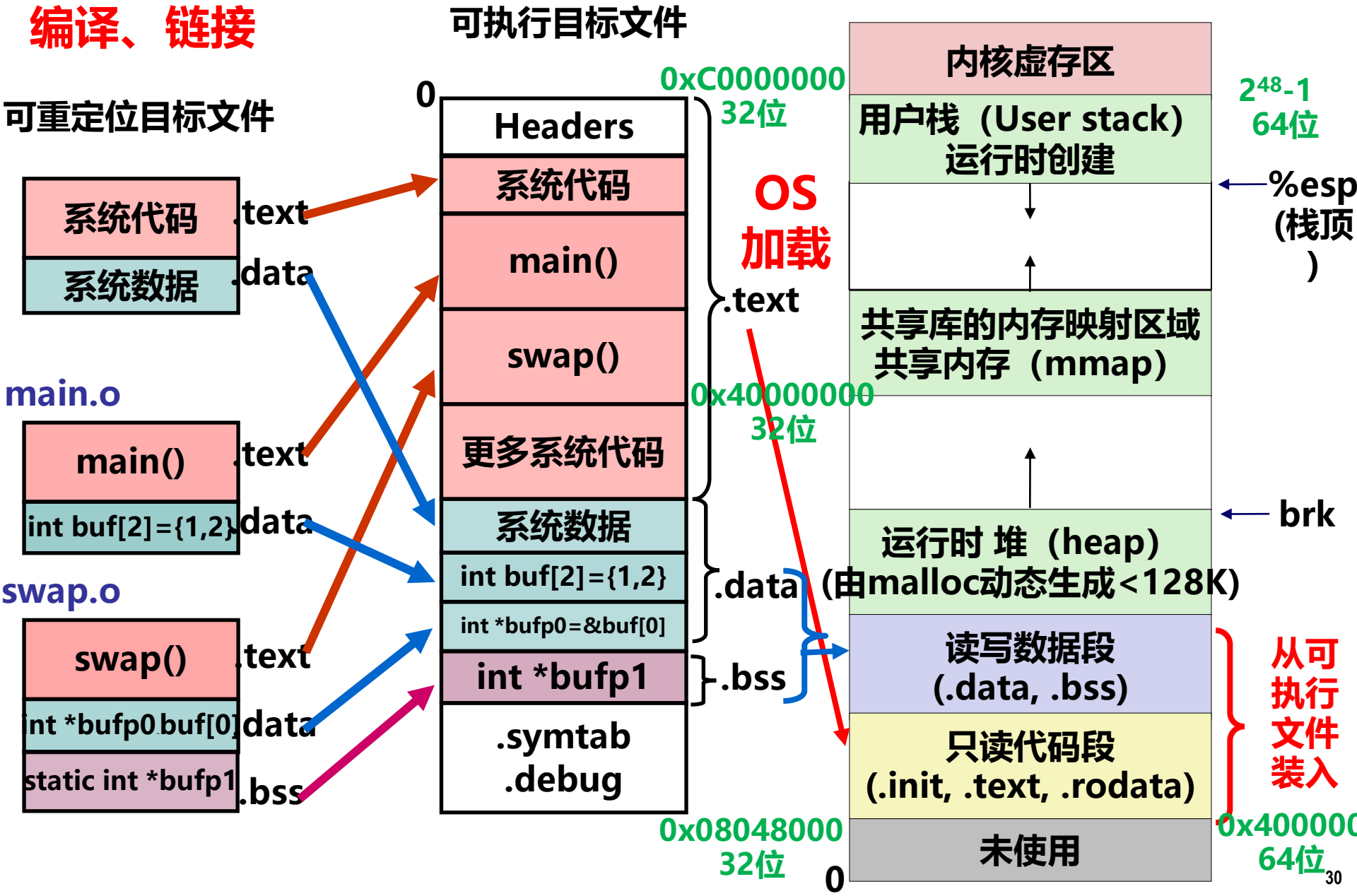


数据经常在各存储部件间传送。故现代计算机大多采用“缓存”技术！

所有过程都是在CPU执行指令所产生的控制信号的作用下进行的。

源程序、目标文件、执行程序、虚拟内存映像

编译、链接



五、怎么优化源程序

1. 更快（本课程重点！）
2. 更省（存储空间、运行空间）
3. 更美（UI 交互）
4. 更正确（本课程重点！各种条件下）
5. 更可靠
6. 可移植
7. 更强大（功能）
8. 更方便（使用）
9. 更规范（格式符合编程规范、接口规范）
10. 更易懂（能读明白、有注释、模块化）

六、计算机系统层次模型

功能**转换**：上层是下层的**抽象**，下层是上层的**实现**

底层为上层提供支撑环境！

程序执行结果

不仅取决于

算法、程序编写

而且取决于

语言处理系统

操作系统

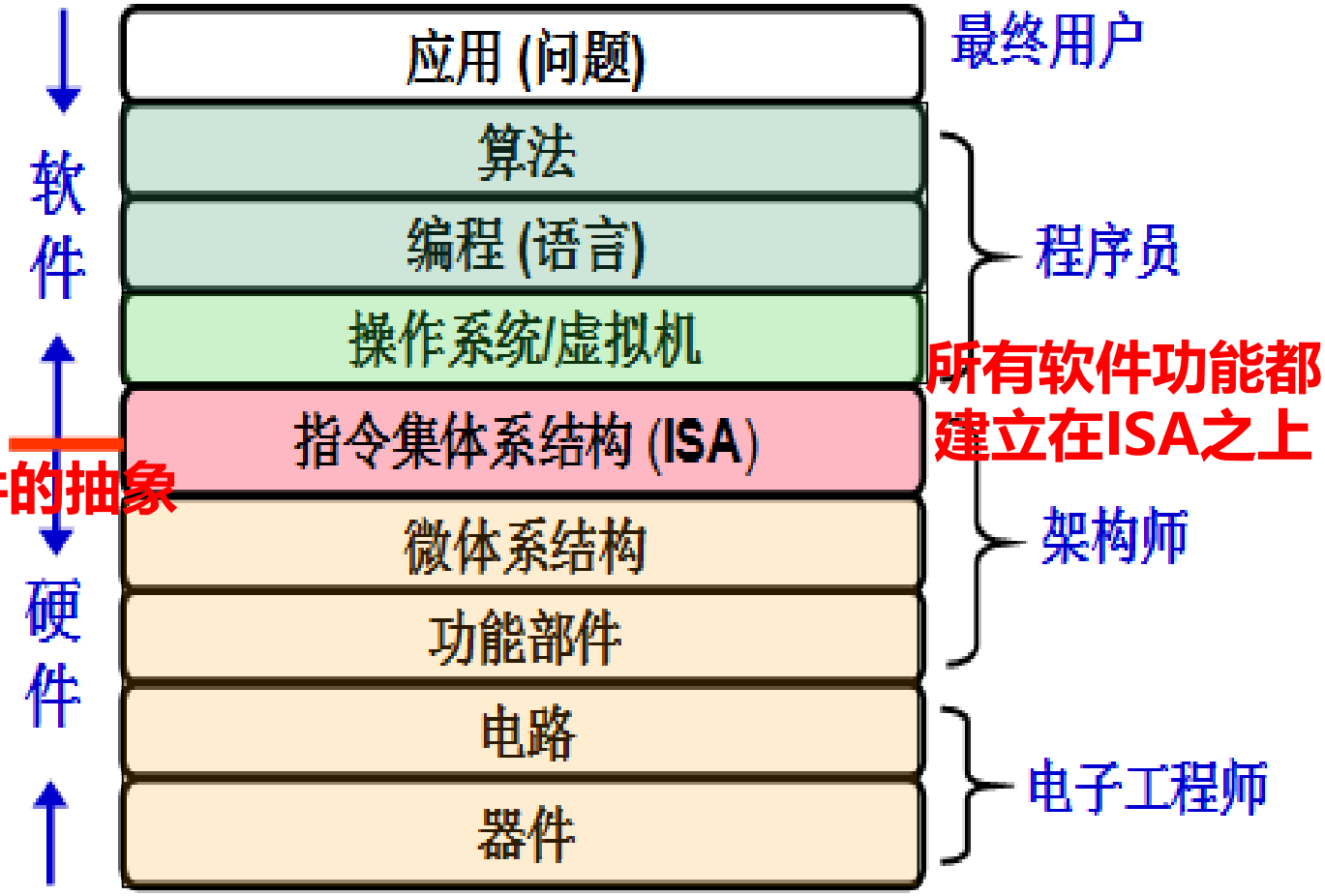
ISA-机器语言

微体系结构

ISA是对硬件的抽象

不同计算机课程处于不同层次

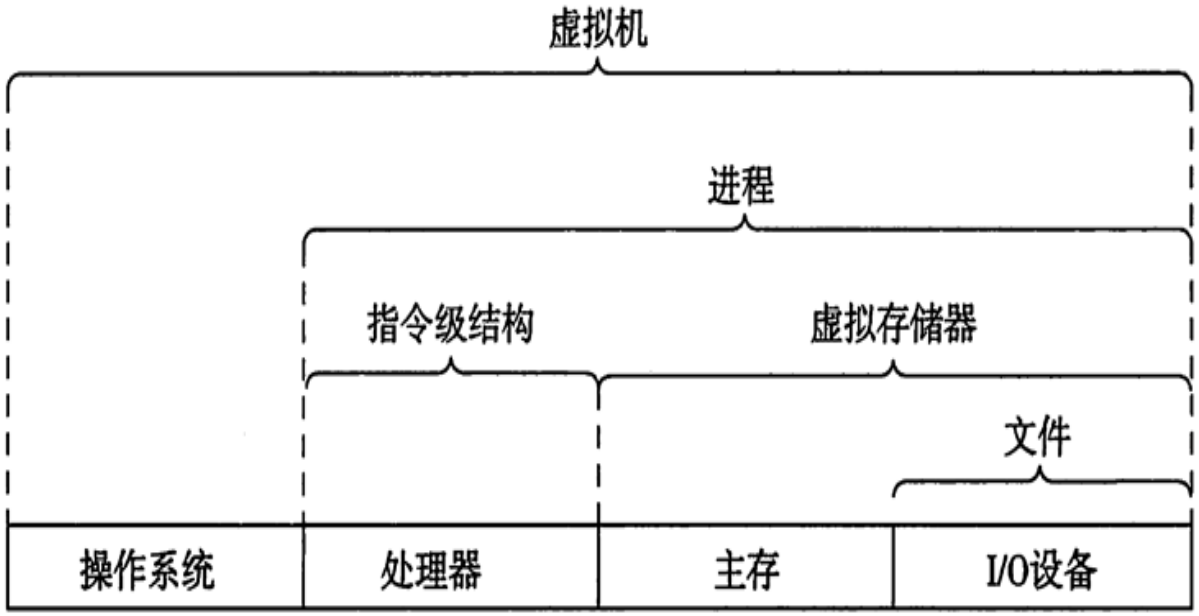
必须将各层次关联起来解决问题



最高层抽象就是点点鼠标、拖拖图标、敲敲键盘，但这背后有多少层转化啊！

计算机系统的抽象表示

-隐藏实际实现的复杂性



中间层语言：MSIL

Java字节码-虚拟机

OS快速迁移部署：虚拟机开发

用户：IO驱动程序、APP不再面向具体CPU，面向IL的虚拟机程序不需移植

微信小程序：都不面向OS了

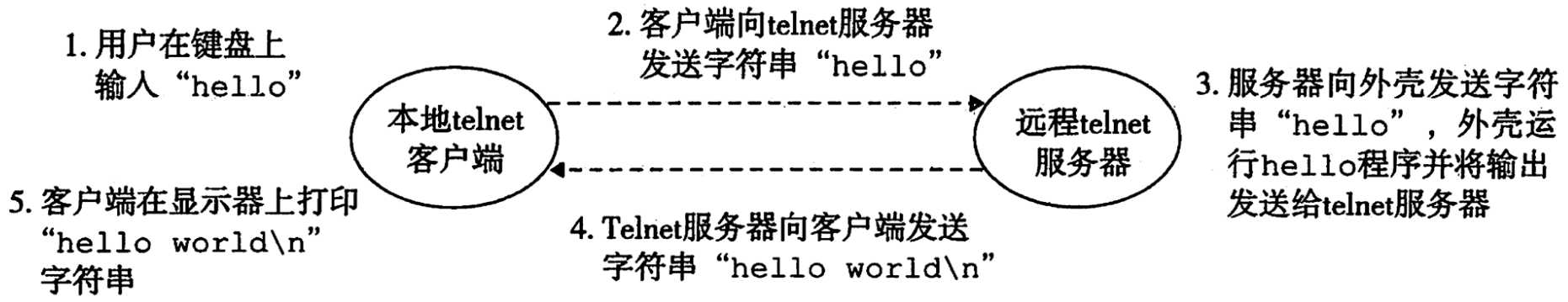
微软：通用语言运行时CLR

SUN：JAVA虚拟机

计算无处不在—普适计算

- CPU无处不在：GPU、NPU、APU
- 存储无处不在
- cache无处不在
- IO无处不在
- 计算机无处不在
- 语言无处不在
- 网络无处不在：网络是一种IO设备
- 计算无处不在：云计算、网格、物联网

- 从低往上，从里到外==简单方便性
- Tradeoff：开销，性价比，内外、上下、软硬、开发/运维



并发与并行-Amdahl定律

- 并发Concurrency：同时具有多个活动的系统
- 并行Parallelism：用并发使一个系统运行的更快

- Linux、Windows等已经是并发的系统，一个进程也可同时执行多个线程（控制流）。
- 单处理器系统下的并发是模拟出来的！

- 多处理器系统：多核、超线程

- 指令级并行：流水线、超标量
- SIMD并行：SSE、AVX

Amdahl定律

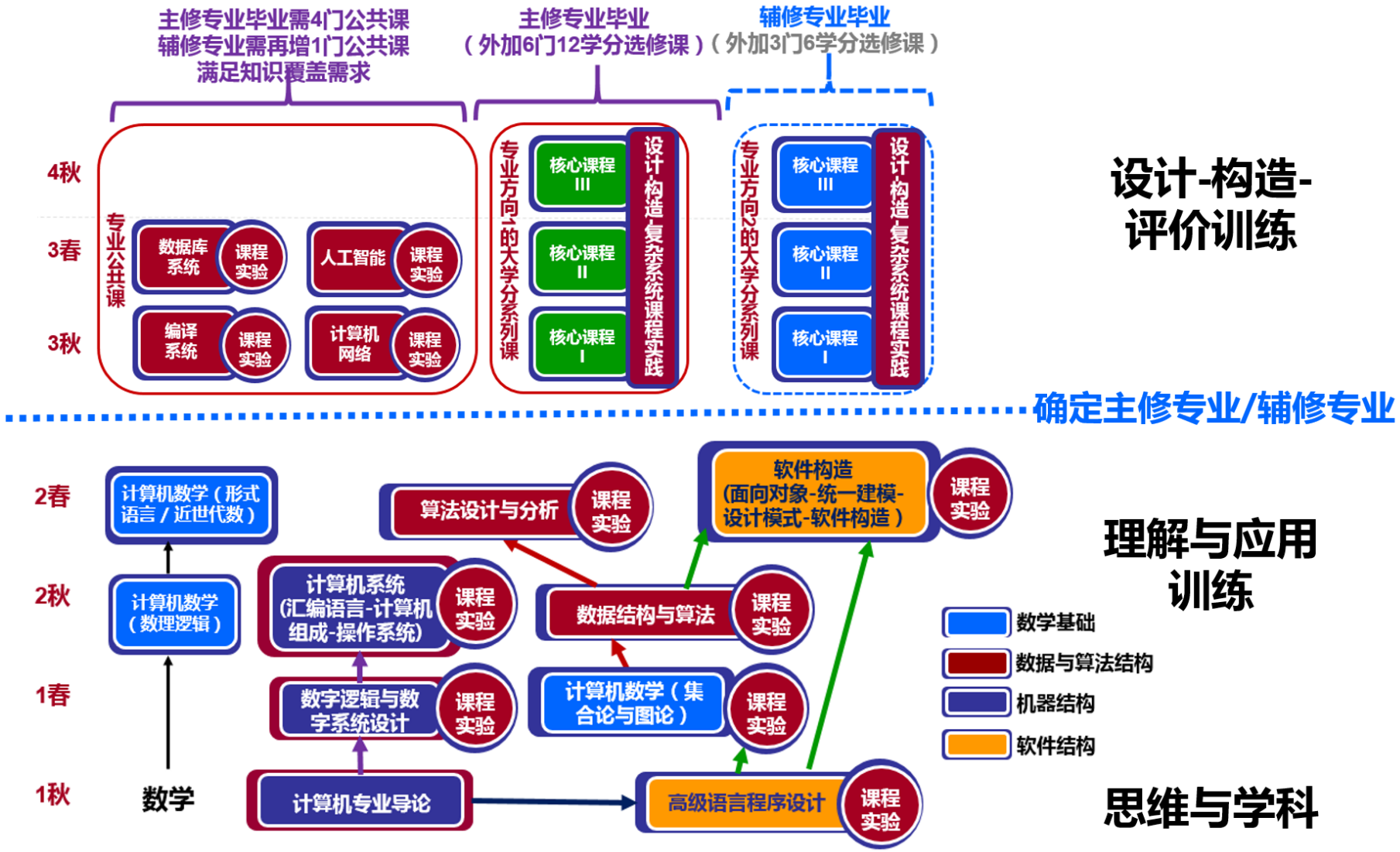
- 系统中对某一部件采用更快执行方式所能获得的系统性能改进程度，取决于这种执行方式被使用的频率，或所占总执行时间的比例。
- 通过更快的处理器来获得加速是由慢的系统组件所限制

- S=1/((1-a)+a/n)
  - 其中，a为并行计算部分所占比例，n为并行处理结点个数。这样，当1-a=0时，(即没有串行，只有并行)最大加速比s=n；当a=0时（即只有串行，没有并行），最小加速比s=1；当n→∞时，极限加速比s→ 1/（1-a），这也就是加速比的上限。例如，若串行代码占整个代码的25%，则并行处理的总体性能不可能超过4。这一公式已被学术界所接受，并被称做“阿姆达尔定律”，也称为“安达尔定理”(Amdahl law)。

七、本课程在CSE/SE课程体系中的地位

- 多数系统课程是建设为中心的
  - 计算机体系结构
    - 用Verilog设计流水线处理器
  - OS
    - 实现OS的示例部分
  - 编译器
    - 编写简单语言的编译器
  - 网络
    - 实现并模拟网络协议

CS/CE 课程体系中的角色



经典例题

1.计算机操作系统抽象表示时(　　)是对处理器、主存和I/O设备的抽象表示。

A.进程　　B. 虚拟存储器　C. 文件　　D. 虚拟机

2. 位于存储器层次结构中的最顶部的是(　)。

A. 寄存器　　B. 主存　　C. 磁盘　　D. 高速缓存

本章主要以选择、填空题考查

成绩组成

- 闭卷考试：
  - 占60%
- 课堂作业5次，每次8分：
  - 占40%
- 题目主要以本部期末试题和课本习题为主

如何学好这门课

- 1、课前预习：看书、PPT
- 2、课堂积极：跟上节奏
- 3、课后消化：反复看书，做题巩固
- 4、学有余力：做做实验，拓展学习

Welcome and Enjoy!