

2025春计算系统23级9~12班

QQ群号：1037857776



# 计算机系统 (CSAPP, ICS)

群名称:顾班-2025春CSAPP

群 号:1037857776

顾崇林

计算机科学与技术学院

[guchonglin@hit.edu.cn](mailto:guchonglin@hit.edu.cn)

# 第一章 计算机概述

教 师： 顾崇林

计算机科学与技术学院

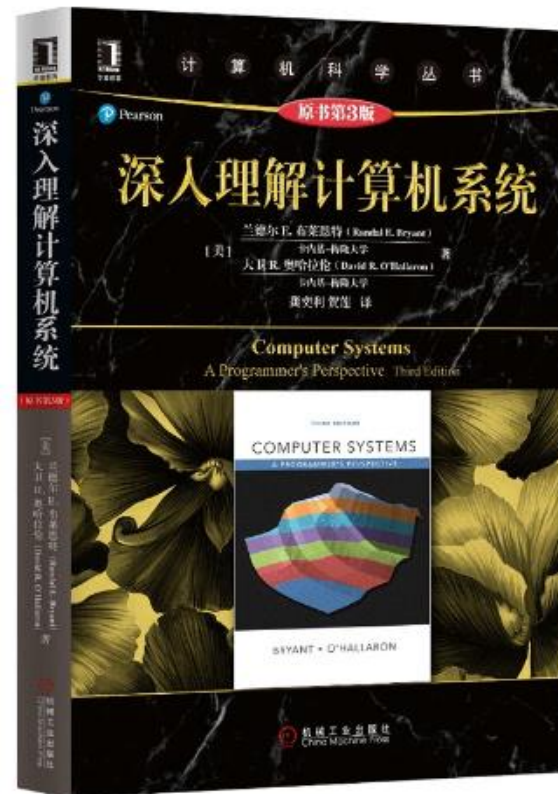
哈尔滨工业大学（深圳）

# 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(原书第3版)-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

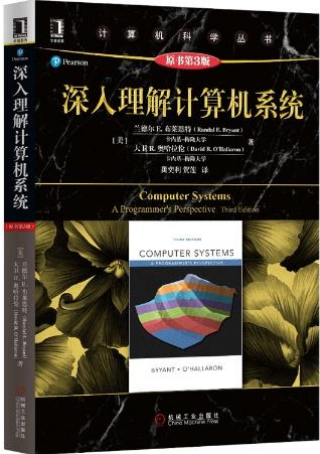



# 课件以及交流

- 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
  - 原版英文PPT下载  
<https://www.cs.cmu.edu/afs/cs/academic/class/15213-s15/www/schedule.html>
- QQ群: **1037857776**
  - 课件、资料
  - 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

# 课程概况 (40学时授课+8学时实验)

- 一. 计算机系统漫游
- 二. 信息表示与处理
- 三. 程序的机器级表示
- 四. 处理器体系结构
- 五. 优化程序性能
- 六. 存储器层次结构
- 七. 链接
- 八. 异常控制流
- 九. 虚拟存储器
- 十. 系统级I/O

学校	课程	网址
CMU	CSAPP 	B站: <a href="https://www.bilibili.com/video/BV1iW411d7hd/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303">https://www.bilibili.com/video/BV1iW411d7hd/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303</a>  <a href="https://www.bilibili.com/video/BV1a54y1k7YE/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303">https://www.bilibili.com/video/BV1a54y1k7YE/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303</a>
九曲阑干		B站: <a href="https://www.bilibili.com/video/BV1cD4y1D7uR/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303">https://www.bilibili.com/video/BV1cD4y1D7uR/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303</a>
北航		B站: <a href="https://www.bilibili.com/video/BV19X4y1P7zW/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303">https://www.bilibili.com/video/BV19X4y1P7zW/?spm_id_from=333.999.0.0&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303</a>
南京大学 (袁春风)	计算机系统基础 	MOOC网址: 计算机系统基础(一): 程序的表示、转换与链接 <a href="https://www.icourse163.org/course/NJU-1001625001?from=searchPage&amp;outVendor=zw_mooc_pcassjg">https://www.icourse163.org/course/NJU-1001625001?from=searchPage&amp;outVendor=zw_mooc_pcassjg</a> 计算机系统基础(二): 程序的执行和存储访问 <a href="https://www.icourse163.org/course/NJU-1001964032?from=searchPage&amp;outVendor=zw_mooc_pcassjg">https://www.icourse163.org/course/NJU-1001964032?from=searchPage&amp;outVendor=zw_mooc_pcassjg</a> 计算机系统基础(三): 异常、中断和输入/输出 <a href="https://www.icourse163.org/course/NJU-1002532004?from=searchPage&amp;outVendor=zw_mooc_pcassjg">https://www.icourse163.org/course/NJU-1002532004?from=searchPage&amp;outVendor=zw_mooc_pcassjg</a> 计算机系统基础(四): 编程与调试实践 <a href="https://www.icourse163.org/course/NJU-1449521162?from=searchPage&amp;outVendor=zw_mooc_pcassjg">https://www.icourse163.org/course/NJU-1449521162?from=searchPage&amp;outVendor=zw_mooc_pcassjg</a> 计算机系统基础(五): x86模拟器编程实践 <a href="https://www.icourse163.org/course/NJU-1464941173?from=searchPage&amp;outVendor=zw_mooc_pcassjg">https://www.icourse163.org/course/NJU-1464941173?from=searchPage&amp;outVendor=zw_mooc_pcassjg</a> B站: 2020 南京大学计算机系统基础习题课 <a href="https://www.bilibili.com/video/BV1qa4y1j7xk?p=7&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303">https://www.bilibili.com/video/BV1qa4y1j7xk?p=7&amp;vd_source=9b3ebd8f0d9db179d1b637e52d4b1303</a>

# 上课时间及地点

- 第3-13周                      周三   第3-4节   T2811教室
- 第3-5, 7-9, 11-13周   周五   第3-4节   T2811教室

# 成绩组成

- 闭卷考试：
  - 占70%
- 作业4~5次：
  - 占10%
- 实验2个：
  - 占20%
  - 共四次课（每次2学时）



# 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. 计算机系统的层次模型**
- 7. 本课程在CSE/SE课程体系中的地位**

# 一、课程目标:

## ■ 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（数据集合及其对应的操作可不依赖于具体的程序设计语言、具体的实现方式，即构成所谓的抽象数据类型 (abstract data type, ADT) ）

## ■ 渐进分析Asymptotic analysis（通过抽象来简化分析过程，忽略每个语句的实际开销对运行时间的增长率影响，只考虑运行时间表达式的最高次数项）

## ■ 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
  - 高级程序语言、汇编语言、计算机组成原理、操作系统、编译原理、计算机网络、计算机体系结构、嵌入式系统、存储系统等

# 理解计算机盒子里的东西

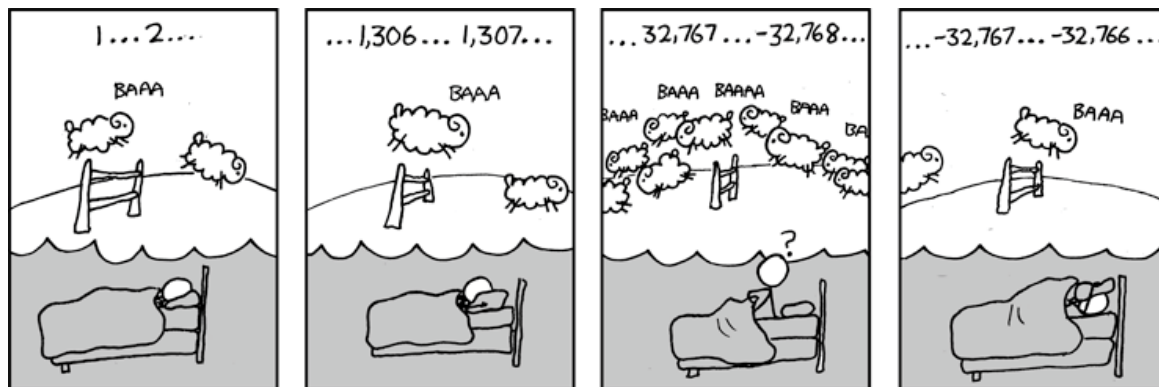
- **张晓东：跟热点可能浪费资源，盒子里的东西更重要**
  - **计算机学科变化非常大，但所有的变化都建立在核心技术和基础学科之上**
  - 计算机影响着各行各业，没有一个学科有这样大的影响力。比起在学科建设中跟风、追热点，更应注重打好基础。**计算机科学不是空中楼阁，如果基础打得足够好，不用担心怎么变，甚至可以引领学科的进展。**
  - **核心技术的意思是学习计算机‘盒子里的东西’，而不仅是学怎么用这个盒子**。“国内大学的计算机专业大多数以应用为主，只有几个顶尖大学重视核心和基础学科。”张晓东直言。
  - 对于近来兴起的人工智能本科专业，他看得较为平淡。“人工智能的核心是通过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学研究都不晚。”**

## 二、伟大现实 (Great Reality)

### 伟大现实#1: int不是整数, float不是实数

#### ■ 例 1: $x^2 \geq 0$ ?

■ Float' s: Yes!



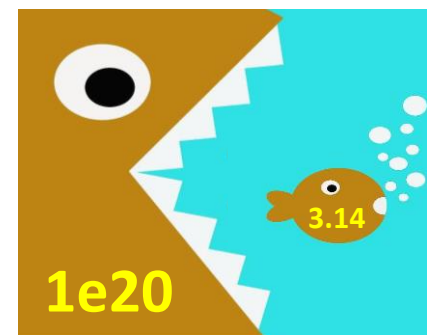
■ Int' s:

- $40000 * 40000 = 16 \text{ } \underline{0000} \text{ } \underline{0000} = 16\text{亿}$
- $50000 * 50000 = ??$

#### ■ 例 2: $(x + y) + z = x + (y + z)$ ?

- 无符号/有符号 Int: Yes!
- 浮点数Float: (注意 $1e20$ 代表 $10^{20}$ )
  - $(1e20 + -1e20) + 3.14 \rightarrow 3.14$
  - $1e20 + (-1e20 + 3.14) \rightarrow ??$

大数吃小数



# 浮点数的故事 1

## ■ 1991年，美国爱国者导弹误炸28名士兵



1991年海湾战争爆发，飞毛腿导弹大战爱国者导弹也拉开了序幕。伊拉克向美军及盟国发射了大量苏制飞毛腿导弹，美军和盟国则用爱国者导弹进行拦截，效果似乎很不错，飞毛腿导弹并未对盟军造成很大威胁。但2月25日，一枚飞毛腿导弹却直接落在了美军位于沙特阿拉伯宰赫兰的军营，导致28名美军士兵死亡，爱国者导弹却罕见地没有做出任何反应。

### 难道是拦截系统失效了？

原来爱国者导弹的拦截计算机是用二进制计算和存储数据的，它会将系统内部时钟测量的时间以 $1/10$ 秒为一个单位，然后转换成二进制存储到一个24位的寄存器里。由于 $1/10$ 的二进制是一个无限循环数 $0.000110011001100110011001100\dots$ ，而寄存器只有24位，后面的就只有舍弃了，这就导致再转回十进制的时候，存在约 $0.000000095$ 秒的误差。本来这点误差也没有什么，但当时爱国者系统已开机100个小时，累积下来的误差就达到了 $0.34$ 秒；而飞毛腿导弹的速度是 $1676$ 米/秒，这导致雷达发现导弹，爱国者启动去寻找目标的时候，已有 $600$ 多米的误差，根本无法在预定区域发现目标，从而无法做出反应。

以色列方面实际在当月中旬就发现了这个问题，并告知了美军，建议每隔一段时间就重启系统，以解决这个问题，制造商也在准备更新软件。然而美军并未引起重视，而且也不明白这每隔一段时间究竟是多长，最终导致了悲剧的发生，第二天所有爱国者导弹系统就全部更新了软件，可28位士兵的生命已经无法挽回了。

# 浮点数的故事 2

## ■ 一行代码引发的“血案”——阿丽亚娜火箭升空40秒爆炸

1996年6月，欧洲航天局计划首次发射新的阿丽亚娜（Ariane）5型火箭。作为经过十年设计、测试和数十亿欧元投入的科技结晶，这枚运载火箭牵动着每位欧洲航天人的心。然而，就在起飞后短短 40 秒，阿丽亚娜501号就在发射区上空炸裂成无数金属残片和燃烧的碎块。对于欧洲航天局来说，这不仅是一次沉重的打击，更是一场令人震惊的灾难。

阿丽亚娜 501 号火箭在脱离发射台后，会按照预定路径平稳加速并飞向太空。在内部，制导系统不断跟踪火箭轨迹并将数据发送至主机载计算机。为了完成数据传输，**制导系统需要将速度读数从 64 位浮点数转换为 16 位带符号整数**。大家可以想想，这个转换过程究竟是怎么回事。使用 16 位无符号整数，我们可以存储 0 到 65535 之间的任意值。而如果把首位用来存放符号（正/负），那么 16 位有符号整数就能涵盖从 -32768 到 +32768 的任意值（实际可用数位只有 15 位）。任何超出这个范围的值都无法正常使用。

因为没有异常处理代码，主计算机将发来的数据解释成了真正的导航数据，认定火箭已经严重偏离航线。为了消解这个根本就不存在的威胁，助推器点燃了全喷嘴偏转，巨大的空气动力压力立即开始撕裂火箭本体。计算机意识到情况到了最危急的关头，于是决定触发自毁机制，把这枚当时造价约 5 亿欧元的火箭当成大炮仗给放了。**也就是说，这场灾难性且耗资巨大的飞行事故，其根源就是一行代码尝试将 64 位浮点数转换成有符号整数，整数溢出结果被直接传递给主计算机，最终被主计算机解释为真实数据。**





# 浮点数的故事

- 为什么要单独拿出来谈浮点数？
- **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

## ■ 不会生成随机值

- 算术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

(浮点加法满足了单调性属性: 如果  $a \geq b$ , 那么对于任何  $a$ 、 $b$  以及  $x$  的值, 除了 NaN, 都有  $x + a \geq x + b$ )

## ■ 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是深远的

### ■ 存储器性能是不一致的

- Cache与虚拟存储器的使用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;  
}
```

**volatile**的作用是作为指令关键字，确保本条指令不会因编译器的优化而省略，且要求每次直接读值。简单地说就是**防止编译器对代码进行优化**。

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

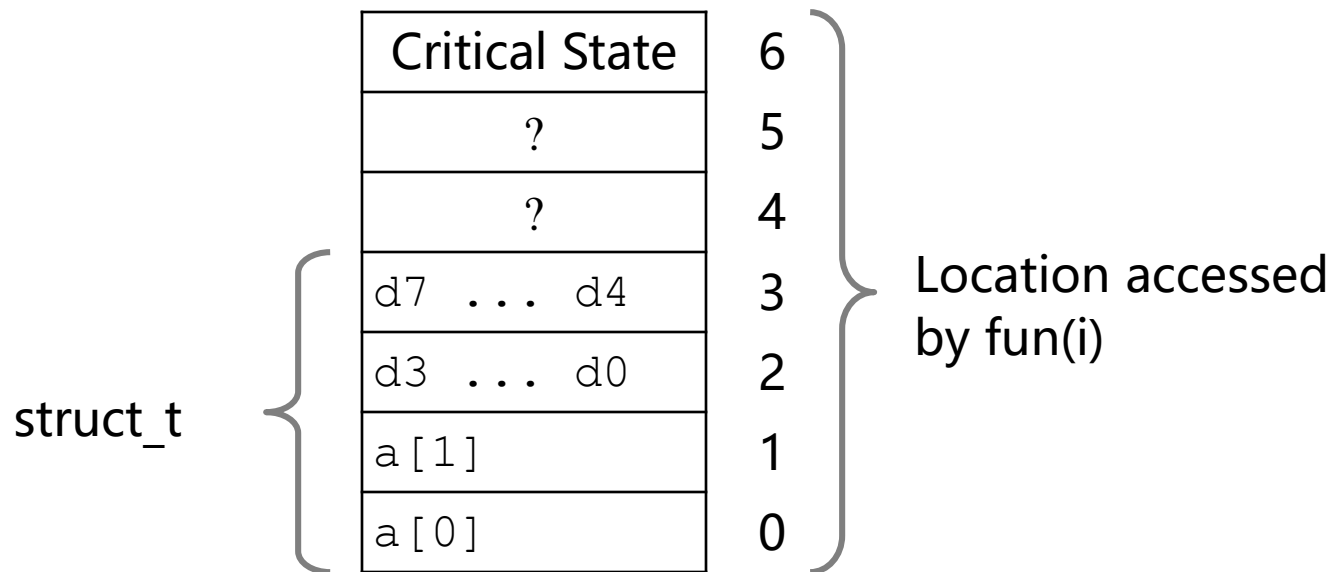
- 结果是特定系统的

# 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
  - 发生在离错误很远的地方(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];
}
```

4.3ms

```
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];
}
```

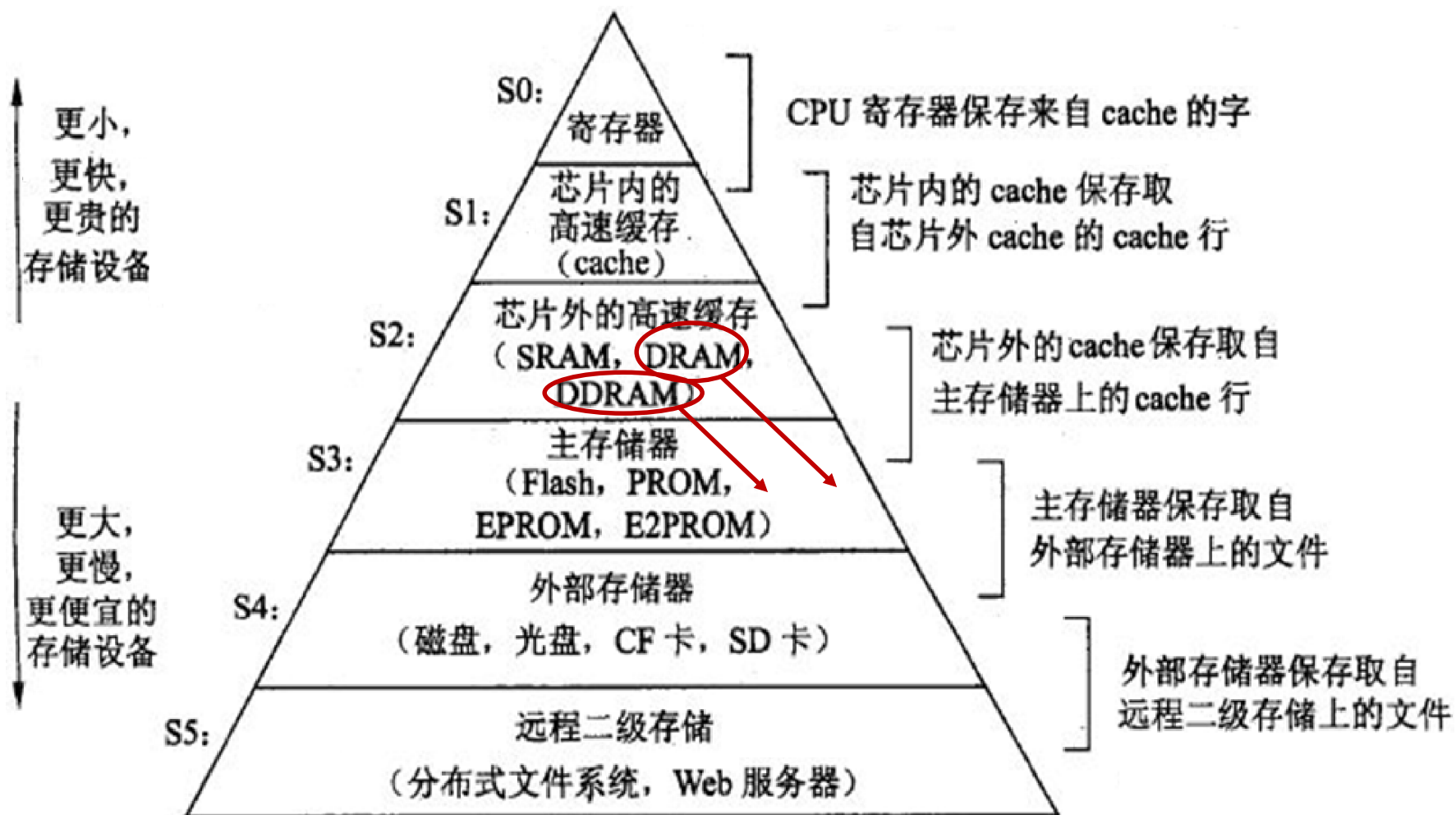
81.8ms

2.0 GHz Intel Core i7 Haswell

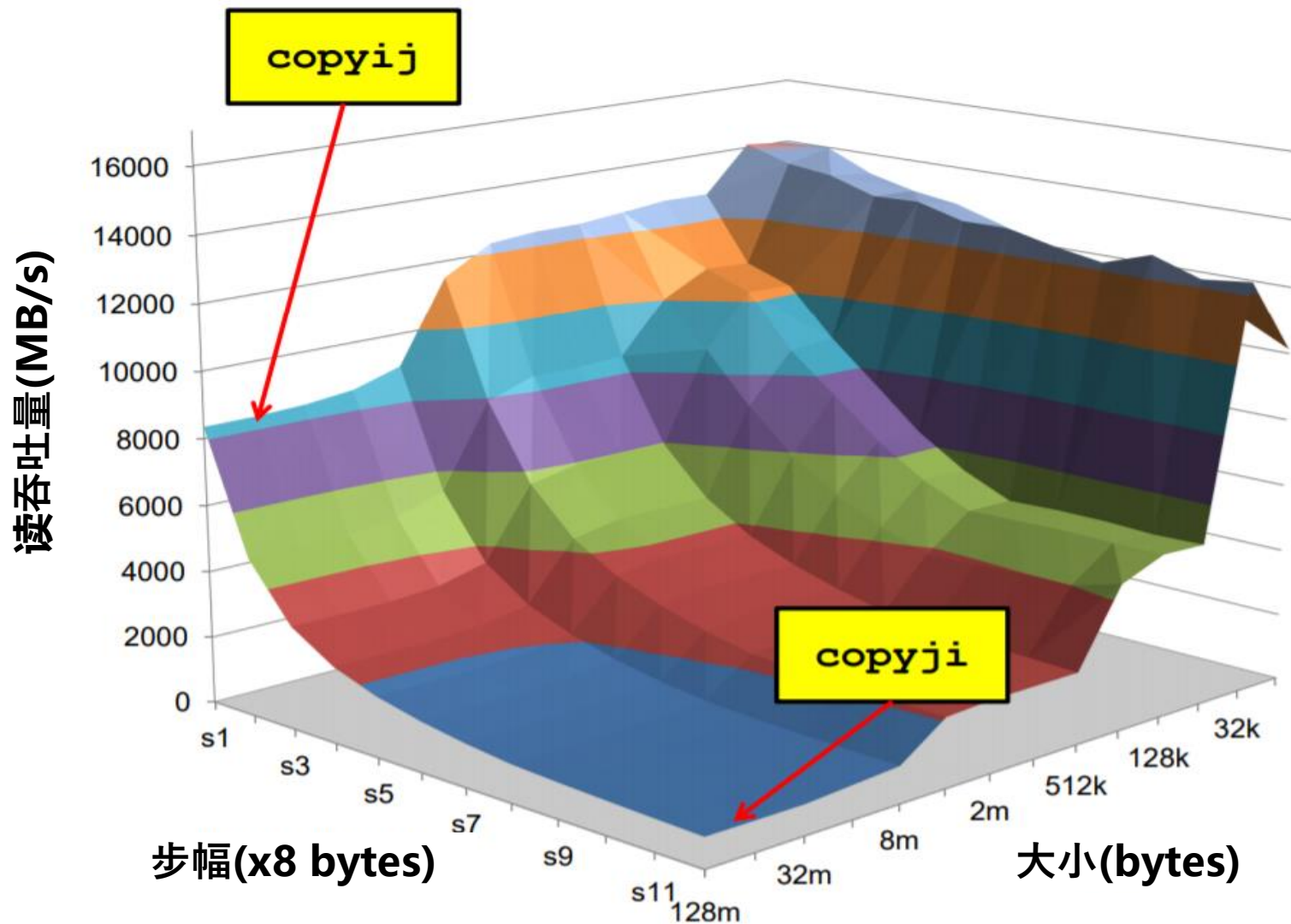
- 存储器的层次化组织
- 性能依赖于访问模式
  - 包括怎样遍历多维数组



# 存储器层次结构 (本页PPT不严谨)



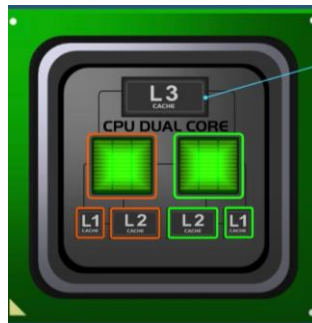
下图是存储山，为什么性能不同 (P445)



# 存储器示例



## DRAM



## SRAM



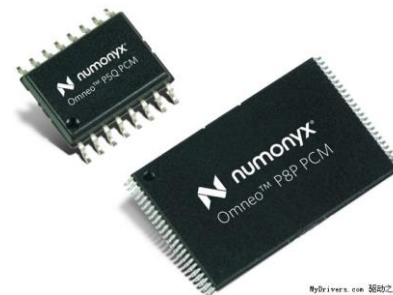
## 磁盘



## M.2接口-SSD



## MSATA-SSD



## PCM

相变存储器：是新一代存储技术,具有非易失性、可字节寻址等特点,其读写速度远高于NAND Flash,可进行数百万次数据擦除与写入,存储的数据可长期保存。

# 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 ( <i>ns</i> )	Write Latency ( <i>ns</i> )	Endurance (# of writes per bit)
SRAM	2-3	2-3	$\infty$
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	$\infty$

# 伟大现实#5: 计算机不只是执行程序

## ■ 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
  - 拷贝不可靠的媒介
  - 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");
}
```

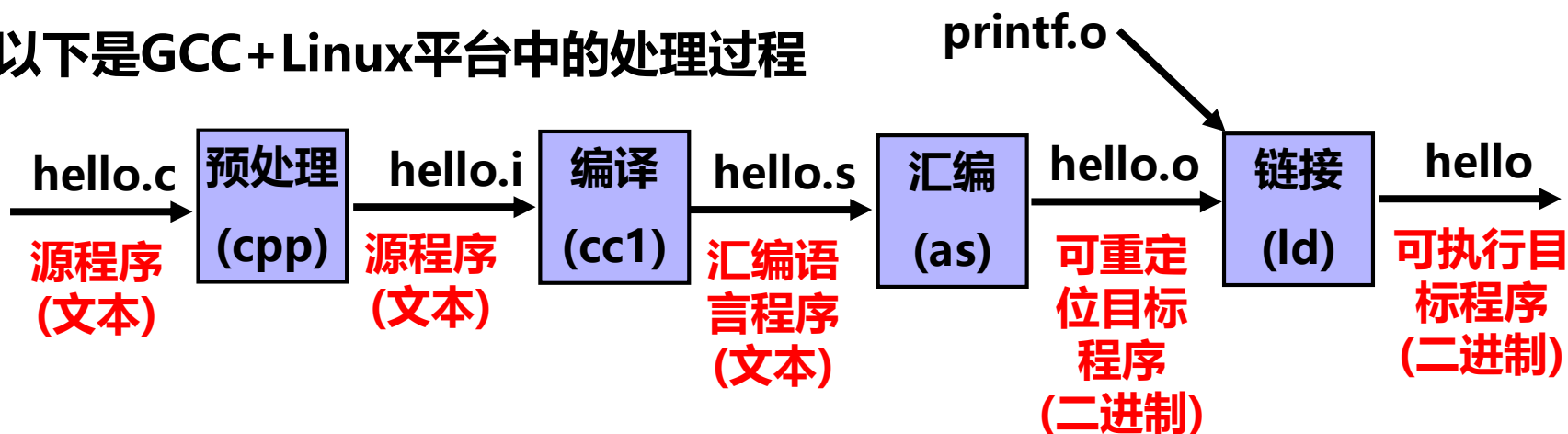
## hello.c的ASCII文本表示

#	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!

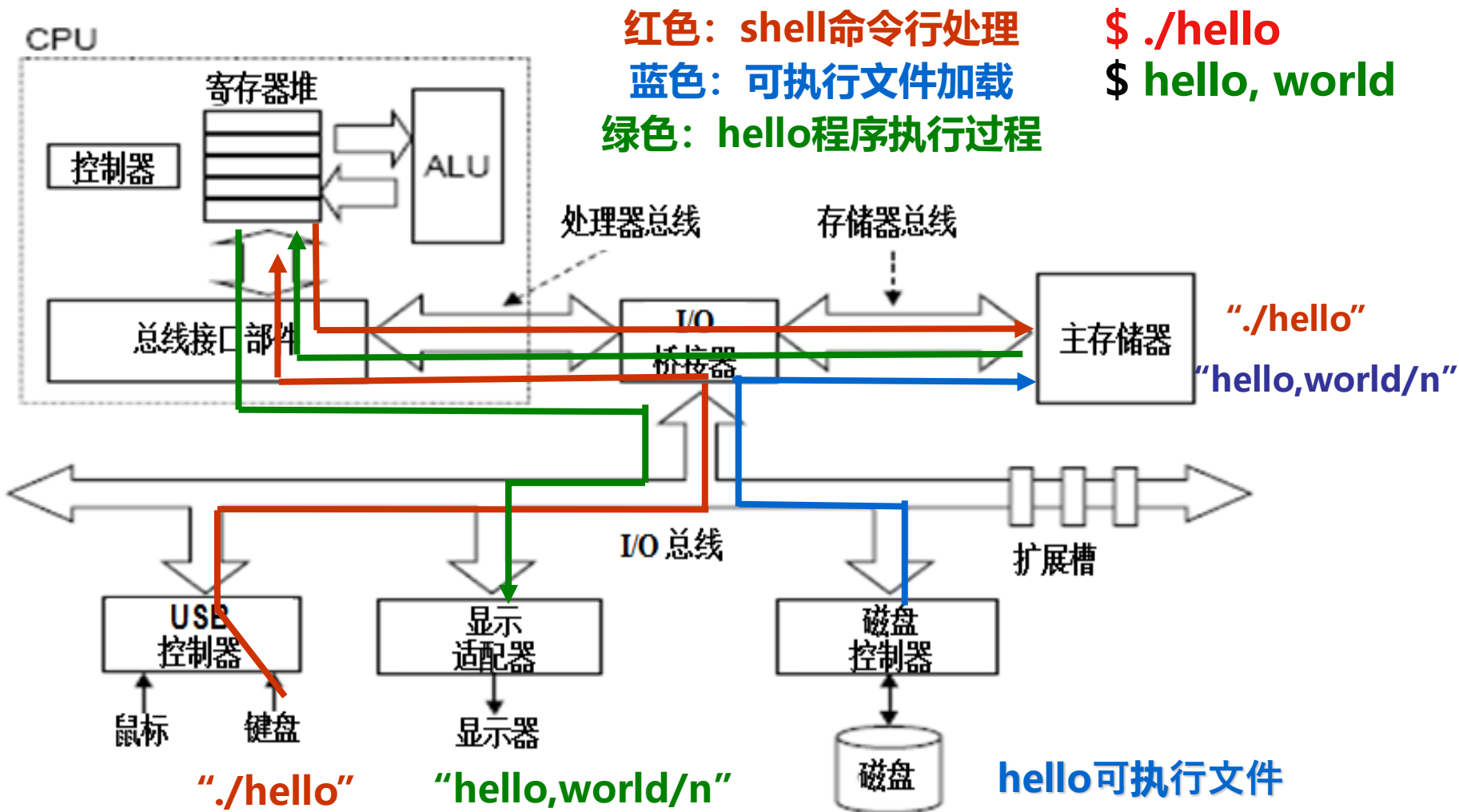
功能：输出 “hello,world”

以下是GCC+Linux平台中的处理过程





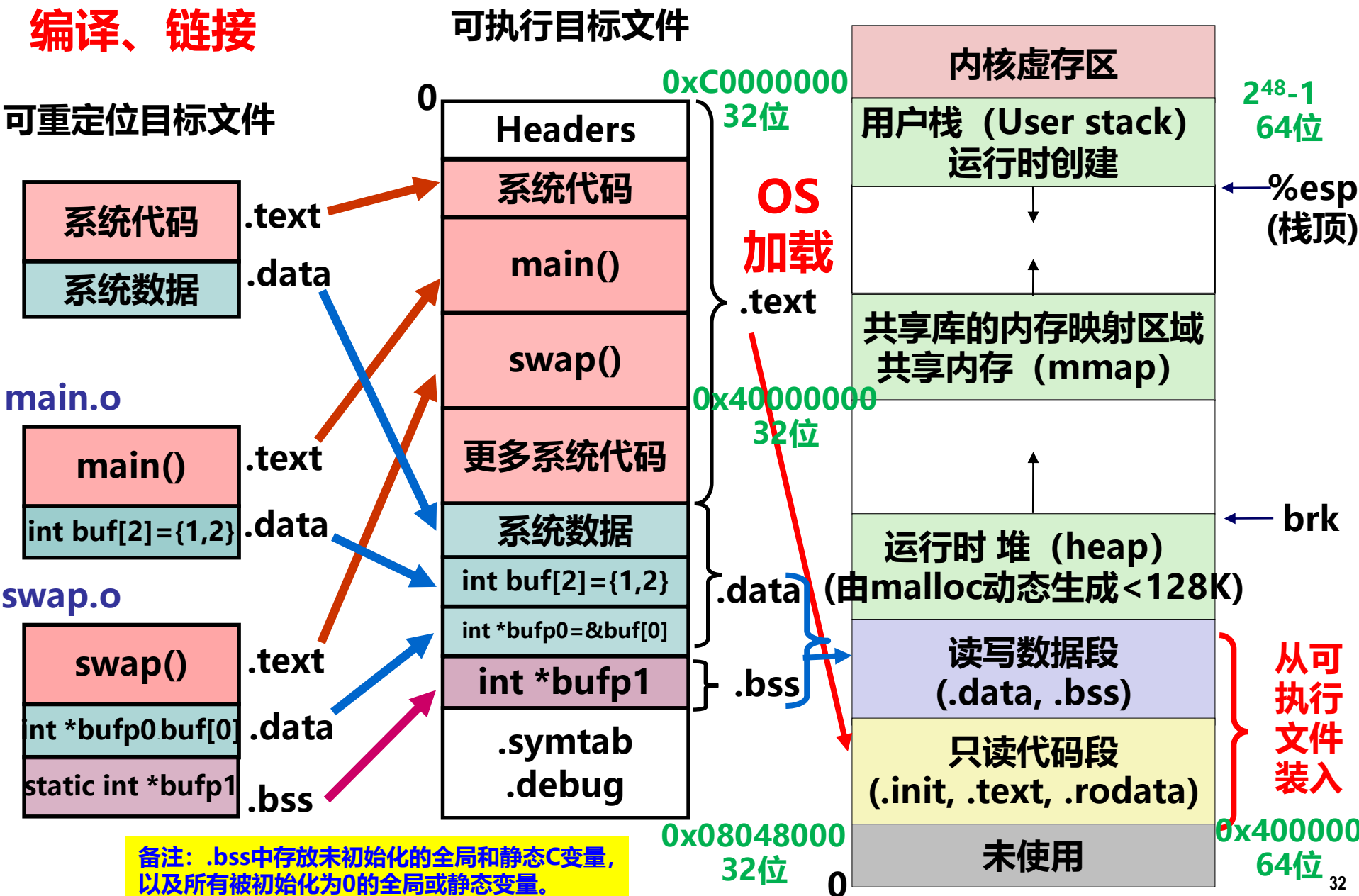
# 四、可执行程序的执行



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

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

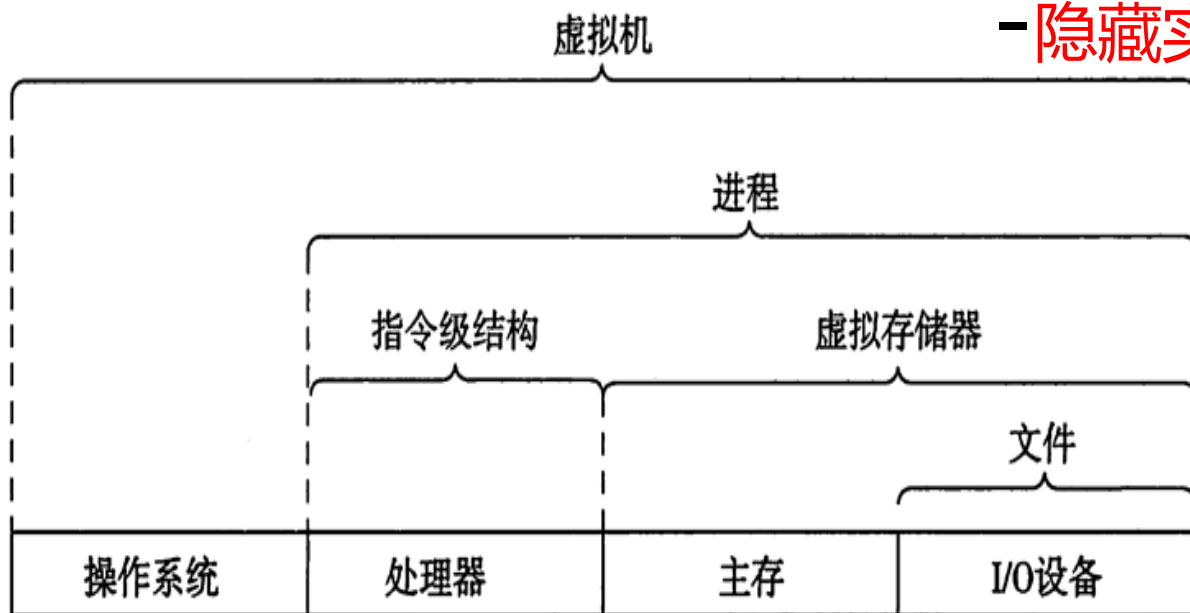
功能转换：上层是下层的抽象，下层是上层的实现  
底层为上层提供支撑环境！



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

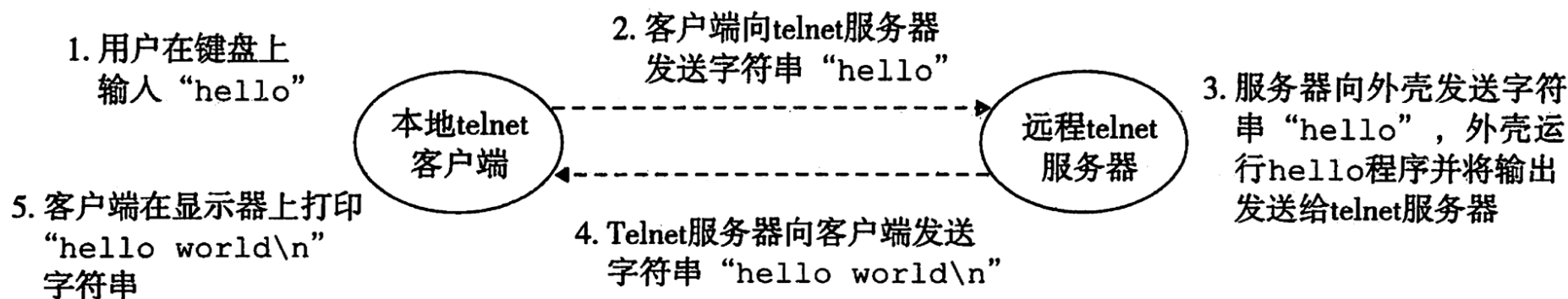


## 计算机系统的几种抽象：

- **指令集架构** 提供了对实际处理器硬件的抽象。
- **文件** 是对I/O设备的抽象。
- **虚拟内存** 是对主存和硬盘的抽象。
- **进程** 是对一个正在运行的程序（或处理器、主存和I/O设备）的抽象。
- **虚拟机** 是对整个计算机的抽象，包括操作系统、处理器和程序。

# 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得更快
- 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
- 加速比： $S = 1 / ((1 - a) + a/n)$ 
  - 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 $n \rightarrow \infty$ 时，极限加速比 $s \rightarrow 1 / (1 - a)$ ，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

- 多数系统课程是从建设者角度讲解
  - 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

# 七、本课程在CSE/SE课程体系中的地位

- 本课程是从程序员角度讲解
  - 目的是了解底层，成为更高效的程序员
  - 使你能够写出更加可靠和高效的程序
    - 包括一些像并发、信号量等特性，与OS底层相关。
  - 涵盖了很多其他课程见不到的细节
- **不仅仅是专为黑客开设的一门课**



# 如何学好这门课

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