课程概述

CS32122: 计算机系统

教师:吴锐
simple@hit.edu.cn，新技术楼901室

第1讲, Feb. 22, 2022

本节内容提要

- 关于课程
- 五个实例
- 课程视角
- 课程内容
- 课程考核

关于课程

——计算机系统是一门怎样的课程?

以一个简单C程序的生成和执行来说明

可执行程序是怎么生成的？

经典的“ hello.c ”C-源程序

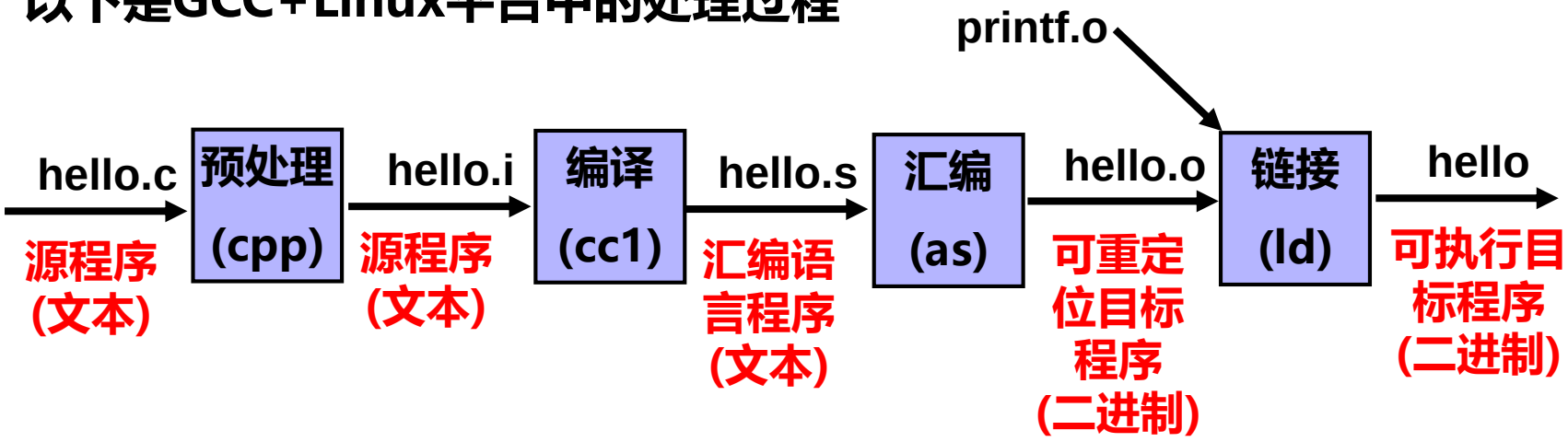
```
#include <stdio.h>

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

hello.c的ASCII文本表示

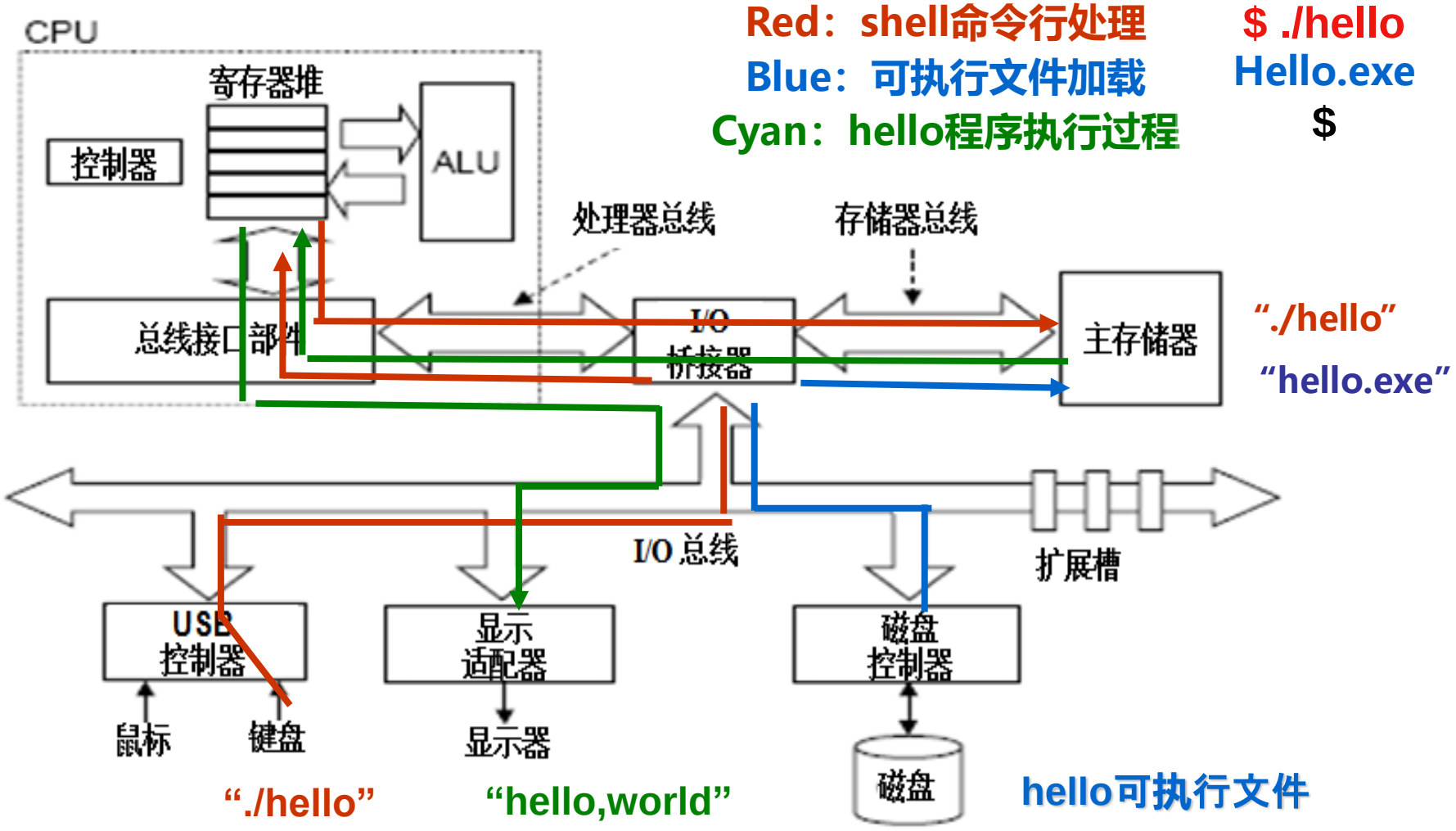
```
# i n c l u d e <sp> < s t d i o .
35 105 110 99 108 117 100 101 32 60 115 116 100 105 111 46
h > \n \n i n t <sp> m a i n ( ) \n {
104 62 10 10 105 110 116 32 109 97 105 110 40 41 10 123
\n <sp> <sp> <sp> <sp> p r i n t f ( " h e l
10 32 32 32 32 112 114 105 110 116 102 40 34 104 101 108
l o , 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

功能：输出“hello,world”　　计算机不能直接执行hello.c!

以下是GCC+Linux平台中的处理过程



可执行程序是怎么执行的？



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。

程序的生成与执行与计算机系统有什么关系呢?

程序的生成与执行涉及计算机多个方面

需要计算机整个系统的支撑

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

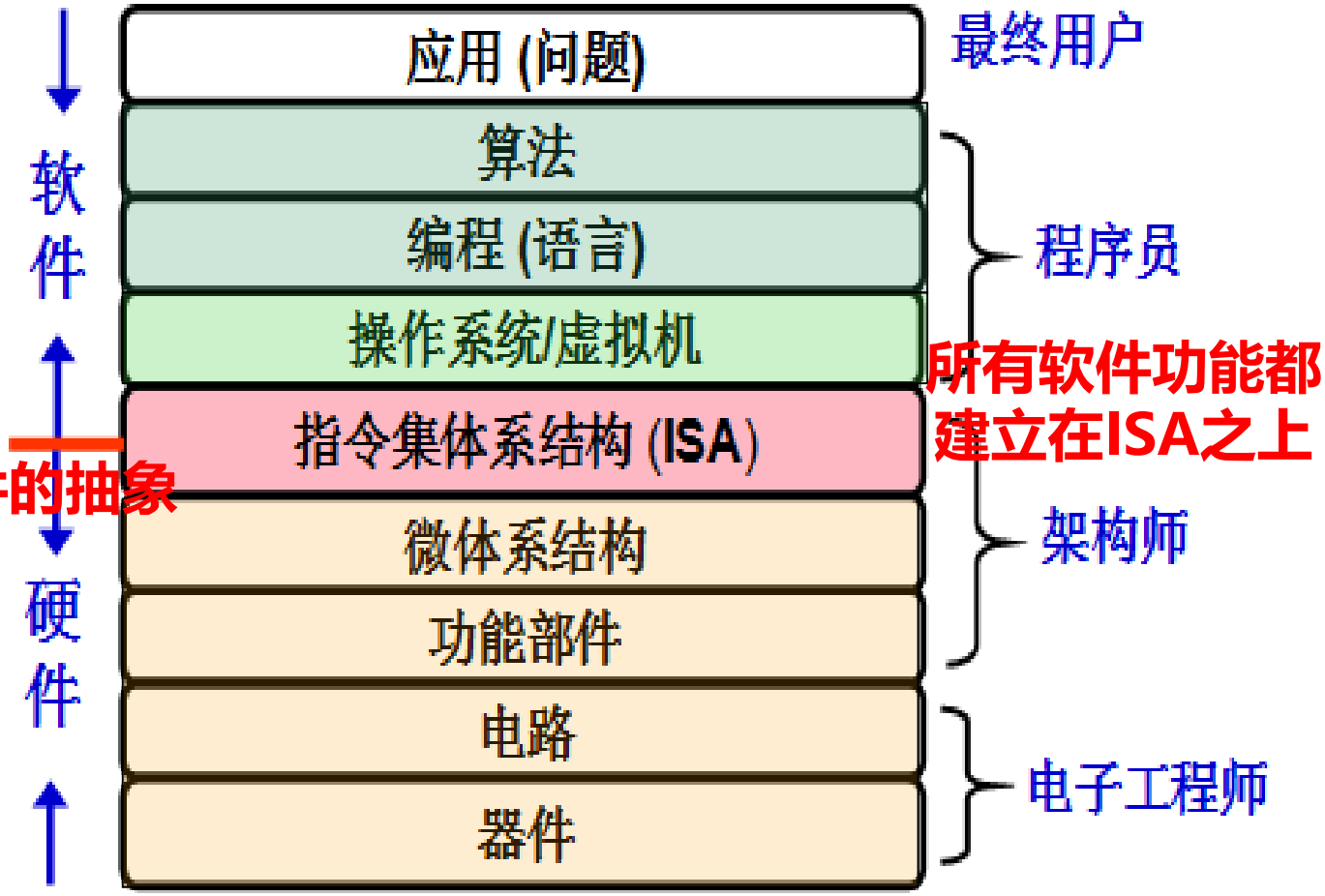
ISA-机器语言

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

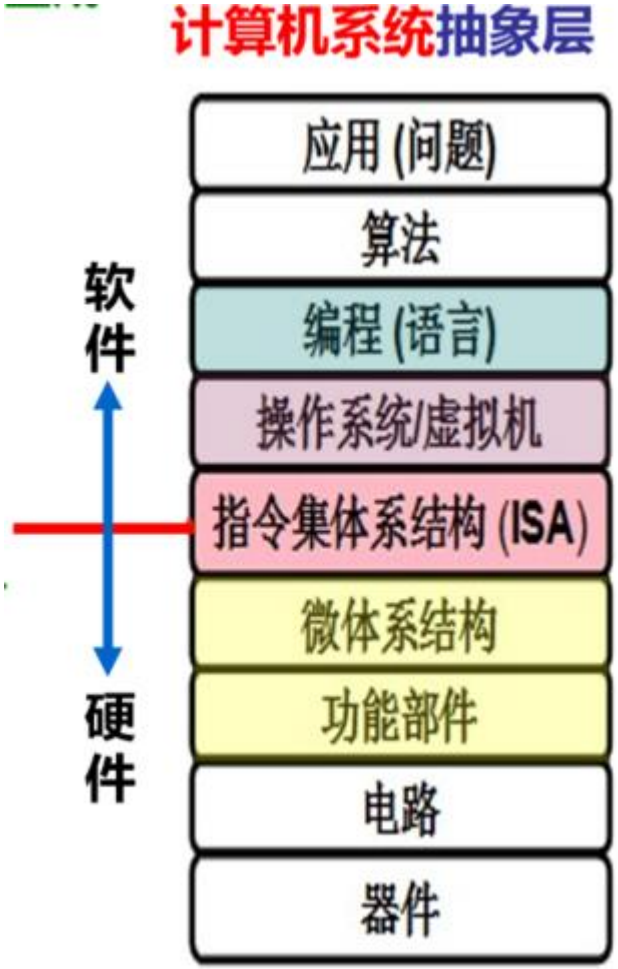


最高层抽象就是点点鼠标、拖拖图标、敲敲键盘，但这背后有多少层转化啊！

计算机系统学习任务

- 课程任务：学习理解计算机是如何生成和运行可执行文件的

- 课程重点学习内容
  - C语言层
    - 数据的机器级表示、运算
    - 语句和过程调用的机器级表示
  - 操作系统、编译和链接的部分内容
  - 指令集体系结构（ISA）层
    - 指令系统、机器代码、汇编语言
  - 微体系结构和硬件层
    - CPU通用结构
    - 层次存储系统



如何学？—几个实际情况

课程学习的一个重要主题:

多数计算机科学/工程的课程都强调抽象

层次模型的抽象、编程语言的抽象

抽象是有局限的！

特别是在出现bug（程序缺陷-故障/错误）时

需要理解底层实现的细节

Abstraction Is Good But Don’t Forget Reality

例子1:
int未必是整数, float未必是实数

程序示例：try/t2.c

- 例 1: $x^2 \geq 0$?
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 = 1600000000
    - 50000 * 50000 ☞?? (2^{31} =2,147,483,648)
- 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) --> ?? 0

理解这个问题需要知道：
机器级数据的表示范围
浮点数的表示与运算规则
高级语言中的运算规则

Source: xkcd.com/571

关于计算机的算术运算

- 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 “循环”特性
    - 交换律, 结合律 , 分配律
  - 浮点操作满足“排序” 特性
    - 单调性, 符号值

例子2:
汇编！汇编！

理解该问题需要知道：
编译器如何优化
机器数如何表示
机器指令的含义与执行
除法错异常的处理

程序示例：try/t2-2.c

- 代码一

```
int a= 0x80000000;
int b = a/-1;
printf("%d\n",b);
```

运行结果： **-2,147,483,648**

通过反汇编得知
除以-1被优化成
取负指令neg，故
未发生除法溢出

- 代码二

```
int a= 0x80000000;
int b = -1;
int c = a/b;
printf("%d\n",c);
```

运行结果为“**floating point exception**”，检测出了溢出

a/b采用除法指令idiv实现，但并不生成OF标志，实际是靠除法前的判断，发现超出表示范围，发出“除法错”异常

不同！**Why**？

So， 你得懂汇编

- 有可能是, 你永远不用汇编语言写程序
  - 编译器比你更擅长更有耐心
- 但是：汇编是机器级执行模型的关键
  - 了解存在Bug程序的行为
  - 调整程序性能
    - 理解编译器所做或不做的优化
    - 理解程序低效的根源
  - 实现系统软件
  - 编写 /对抗 恶意软件（malware）

例子3：存储引用Bug

程序示例：
try/membug

```
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
```

- 结果是面向特定系统的

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

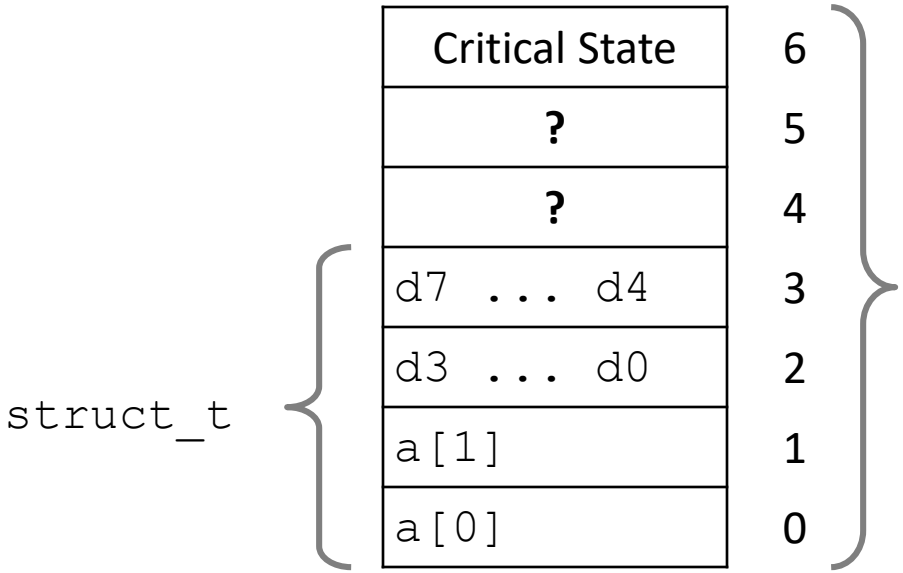
```
fun(0)  ->      3.14
fun(1)  ->      3.14
fun(2)  ->      3.1399998664856
fun(3)  ->      2.0000061035156
fun(4)  ->      3.14
fun(6)  ->      Segmentation fault
```

注释:

理解该问题需要知道:

机器数表示

栈帧中数据的布局



关于存储

RAM并不是一个物理抽象

- 存储器不是无限的
  - 存储器需要分配与管理
  - 很多应用是受存储支配/控制的
- 存储引用错误特别致命
  - 在时间和空间上的影响都是深远的
- 存储器性能是不一致的
  - Cache与虚拟存储器的效率会严重影响程序性能
  - 针对存储系统的特点编写程序，会大幅提升程序运行速度

存储引用错误

- **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
- 导致令人讨厌的**bug**
  - bug造成的任何影响依赖于系统和编译器
  - 可能在bug生成很久才被察觉到
- 怎么办?
  - 用 Java, Ruby, Python, ML, …编程
  - 使用或开发工具来发现地址引用错误 (e.g. Valgrind)

例子4：存储系统的性能

```
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];
}
```

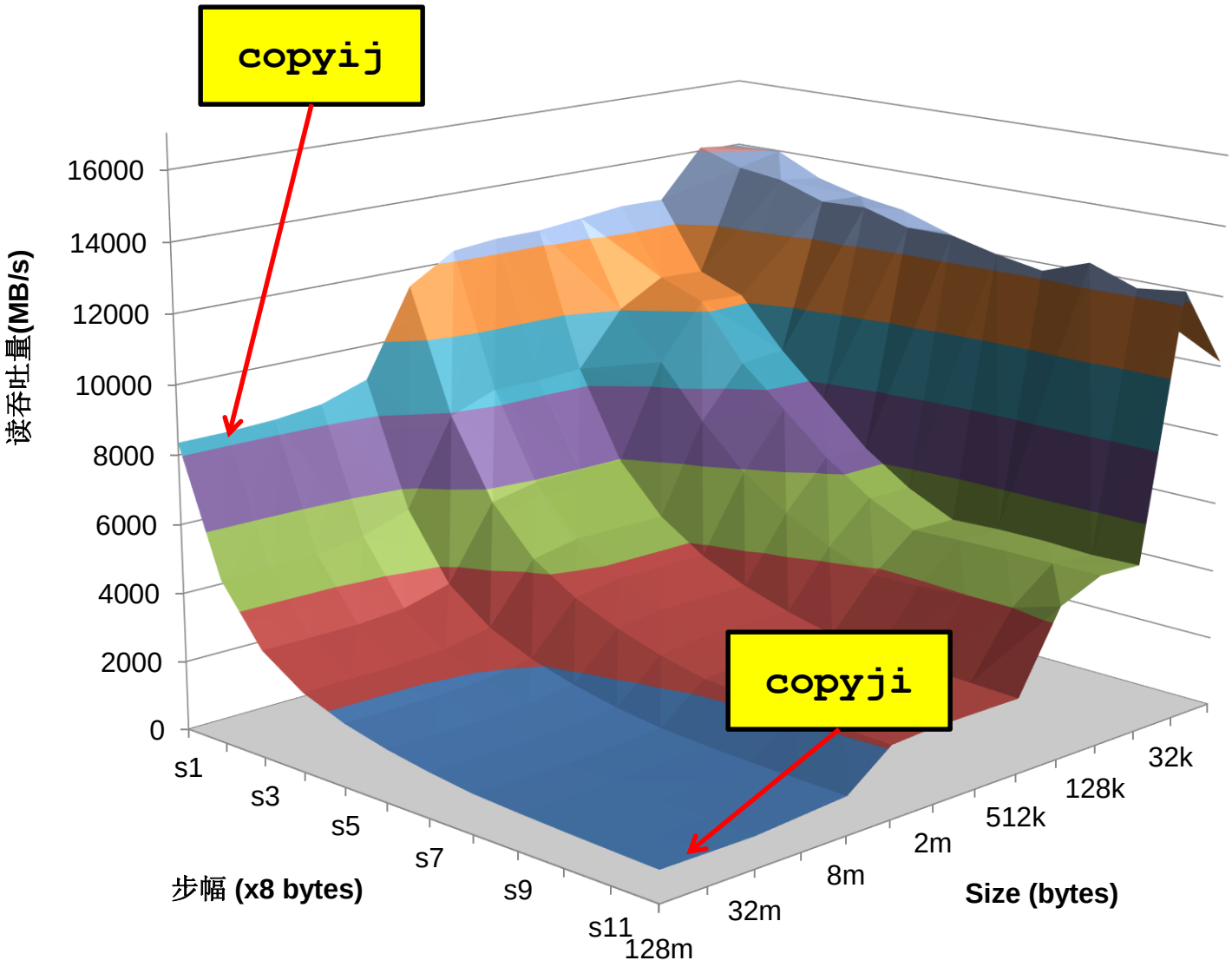
4.3ms **2.0 GHz Intel Core i7 Haswell** **81.8ms**

两个程序功能一样、算法一样，时间空间复杂度一样，执行时间why？

- 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

理解该问题需要知道：
数组的存放方式
Cache机制
访问局部性

为什么性能不同


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

性能比算法渐进复杂度更重要

- 性能无法预测
  - 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
  - 不破坏代码的模块化与整体性，怎么改进性能

例子5:

除了执行程序计算机还要做很多

- 进行数据的输入输出
  - I/O系统对程序的可靠性和性能很关键
- 通过网络互相通讯
  - 网络环境下会出现很多系统级问题
    - 多个进程的并发操作
    - 不可靠媒体的拷贝
    - 交叉平台的兼容性
    - 复杂的性能问题

上述5个实例表明:需要用“系统思维”分析具体问题

课程的视角

- 大多数系统类课程都是以构建为中心
  - 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

本课程的视角

- 本课程是以程序员为中心—程序员的角度认识系统
  - 程序员关心程序如何运行的更快、更稳定、更安全
- 课程目标：
  - 通过更好地理解底层系统，成为更高效的程序员
    - 能够发现并有效地排除bug
    - 能理解并调整程序性能
  - 为CS/SE的后续系统课程打基础
    - 编译、操作系统、计算机网络、计算机体系结构、嵌入式系统、存储系统等.

课程内容：程序与数据

- 主题
  - 位操作,算术运算, 汇编语言程序
  - C控制与数据结构的表示
  - 包括体系结构与编译的方面

- 实验
  - datalab: 位操作
  - bomblab: 拆除一个二进制炸弹
  - attacklab:代码注入攻击的基础知识

课程内容：存储器层次

- 主题
  - 存储技术,存储层次，高速缓冲器, 磁盘, 局部性
  - 包括体系结构与编译的方面

- 实验
  - cachelab: 建立一个 cache模拟器，并为局部性进行优化.
    - 学习如何在你的程序中利用局部性.

课程内容：异常控制流

- 主题
  - 硬件异常，进程，进程控制，Unix信号，非局部跳转
  - 包括体系结构、OS与编译的方面
- 实验
  - tshlab: 编写自己的 Unix 外壳.
    - 第一次引入并发

课程内容：虚拟存储器

- 主题
  - 虚拟存储器, 地址翻译, 动态存储器分配
  - 包括体系结构、OS的方面

- 实验
  - ~~malloclab: 编写你自己的存储器分配程序包~~
    - ~~真实感受下系统底层的编程~~

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!
    - 如何解决实验
    - 练习题中有典型的考试题目

- 参考:
- 袁春风，计算机系统基础，机械工业出版社
  - 南京大学

获得帮助

- 课程 **Web**网站: **http://www.cs.cmu.edu/~213**
  - 完整的课程资料
  - 课件、作业、测验、答案
  - 作业的说明
- **QQ**群
  - （密码：cs2022）
  - 课件
  - 网上答疑
  - 通知

群名称：计算机系统2022春5678
群　号：927653594

评分

考核环节	建议分值比例	考核/评价细则
实验	10%	1.计算机系统漫游及linux下C工具应用； 2.数据与代码的机器表示； 3. 6.程序优化； 4.微壳TinyShell；
作业	0%	平时作业5次，给出作业完成等级，参加考试的必要条件
大作业	10%	百分制，正确性75分、格式10分、条理清晰5分、图文并茂10分
期末考试	80%	一纸开卷模式，客观题60%、主观题40%

Welcome
and Enjoy!