ICS 笔记

Part 1：课程概述

一·五个有趣的现实问题

- 1.整型不是整数，浮点型不是实数
  - 【例 1.1】$x^2 \geq 0$永远成立吗
  - 解：
    - a.若x是浮点型，成立
    - b.若x是整型，50000*50000=负数，因为整型有上界溢出
  - 【例 1.2】是否满足加法结合律
  - 解：
    - a.若x,y,z是整型，满足
    - b.若x,y,z是浮点型，(1e20+ -1e20)+3.14=3.14 1e20+(-1e20+3.14)=0，因为浮点数精度不同不满足结合律

    计算机系统的算术≠数学中的算术
  - 【例 1.3】两个整型 a==b，两个浮点型 a==b 不一定成立
  - 解：
    - a.因为精度可能不同
    - b.正确方法：做差取绝对值 abs(a-b)<=epsilon（很小的数）
- 2.了解汇编
  - 【例 2.1】比较 int unsigned int

```
int array[]={1,2,3};
#define TOTAL sizeof(array)/* unsigned int */
void main(){
    int d=-1;
    if(d<=TOTAL)
        printf("small\n");
    else  printf("large\n");// if语句作比较时，编译器认为-1是unsigned int(很大的整数)
}
```

  - 【例 2.2】分析不同的底层汇编代码效率

```
void fun1(int* x,int* y){
    *x+=*y;
    *x+=*y;
}
void fun2(int* x,int* y){
    *x+=2*(*y);
}
```

解：两个程序似乎有相同的行为。但是 fun2 的效率会更高，，fun1 需要 6 次存储器引用，而 fun2 只需 3 次

- 【例 2.3】把小段汇编代码加入 C 代码，来访问硬件（处理器）上的周期计数器（cycle counter）

```
static unsigned cyc_hi=0;
static unsigned cyc_lo=0;
/* Set *hi and *lo to the high and low order bits of the cycle counter */
void access_counter(unsigned* hi,unsigened* lo){
    asm("rdtsc;movl %%edx,%0;movl %%eax,%1"
    : "=r" (*hi),"=r" (*lo))
    :
    : "%edx","%eax");
}
```

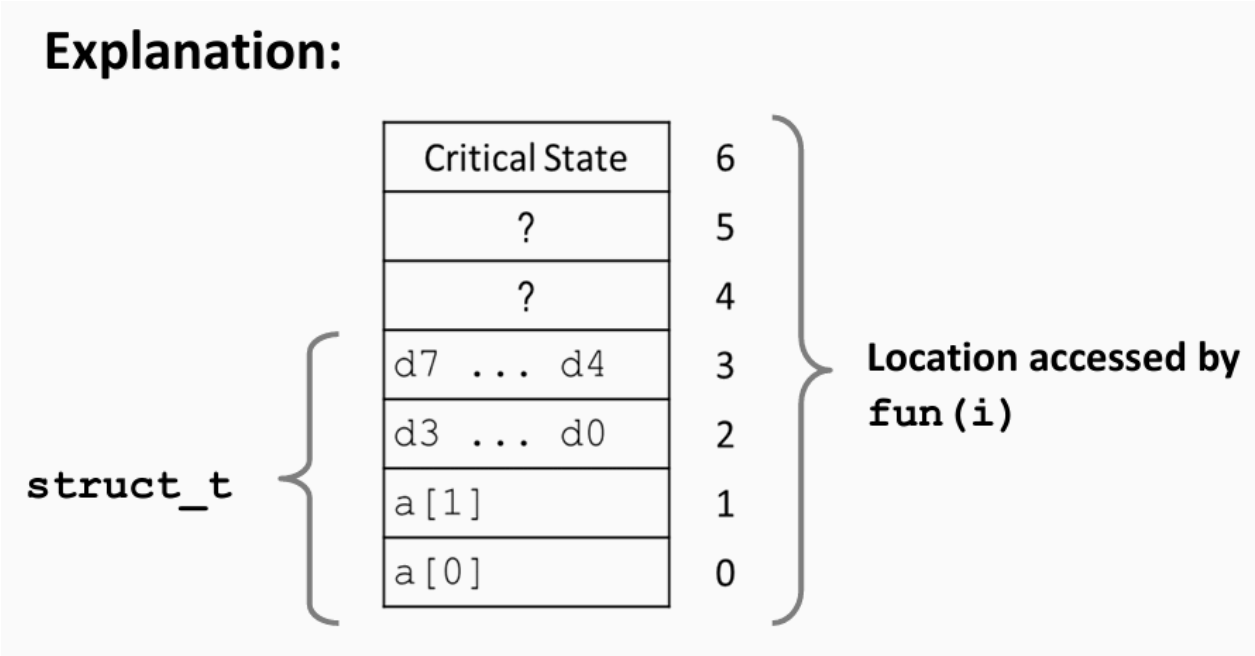
- 防范恶意软件
  - 分析没有源码的软件时，需要反汇编
  - 常见安全漏洞包括：缓冲区溢出，内存泄露，非授权内存写入
  - 对反汇编得到的代码进行静态分析，是一种找到已知安全漏洞代码的有效手段
- 【例 2.4】定位 gets()这样不安全函数对应的汇编代码

```
void main(){
    char buf[1024];
    gets(buf);/* 用户输入不做限制，缓冲区溢出 */
}
#define BUFSIZE 1024
void main(){
    char buf[BUFSIZE];
    fgets(buf,BUFSIZE,stdin);/* 限制输入大小的参数 */
}
```

- 3. 内存对程序性能的影响至关重要
  - 内存是有限的
  - 内存引用（指针）错误尤为严重
  - 内存性能并非始终如一

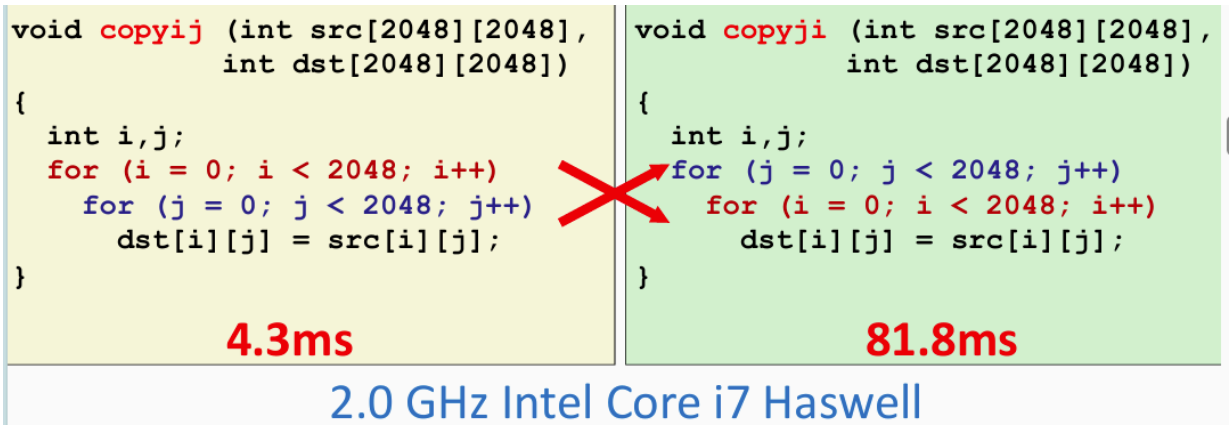
```
typedef struct{
    int a[2];
    double d;
} struct_t;
double fun(int i){
    volatile struct_t s;
    s.d=3.14;
    s.a[i]=1073741824;/* Possible out of bounds */
    return s.d;
}
```

```
fun(0)  -> 3.14
fun(1)  -> 3.14
fun(2)  -> 3.1399998664856
fun(3)  -> 2.00000061035156
fun(4)  -> 3.14
fun(5)  -> 3.14
fun(6)  -> segmentation fault
```

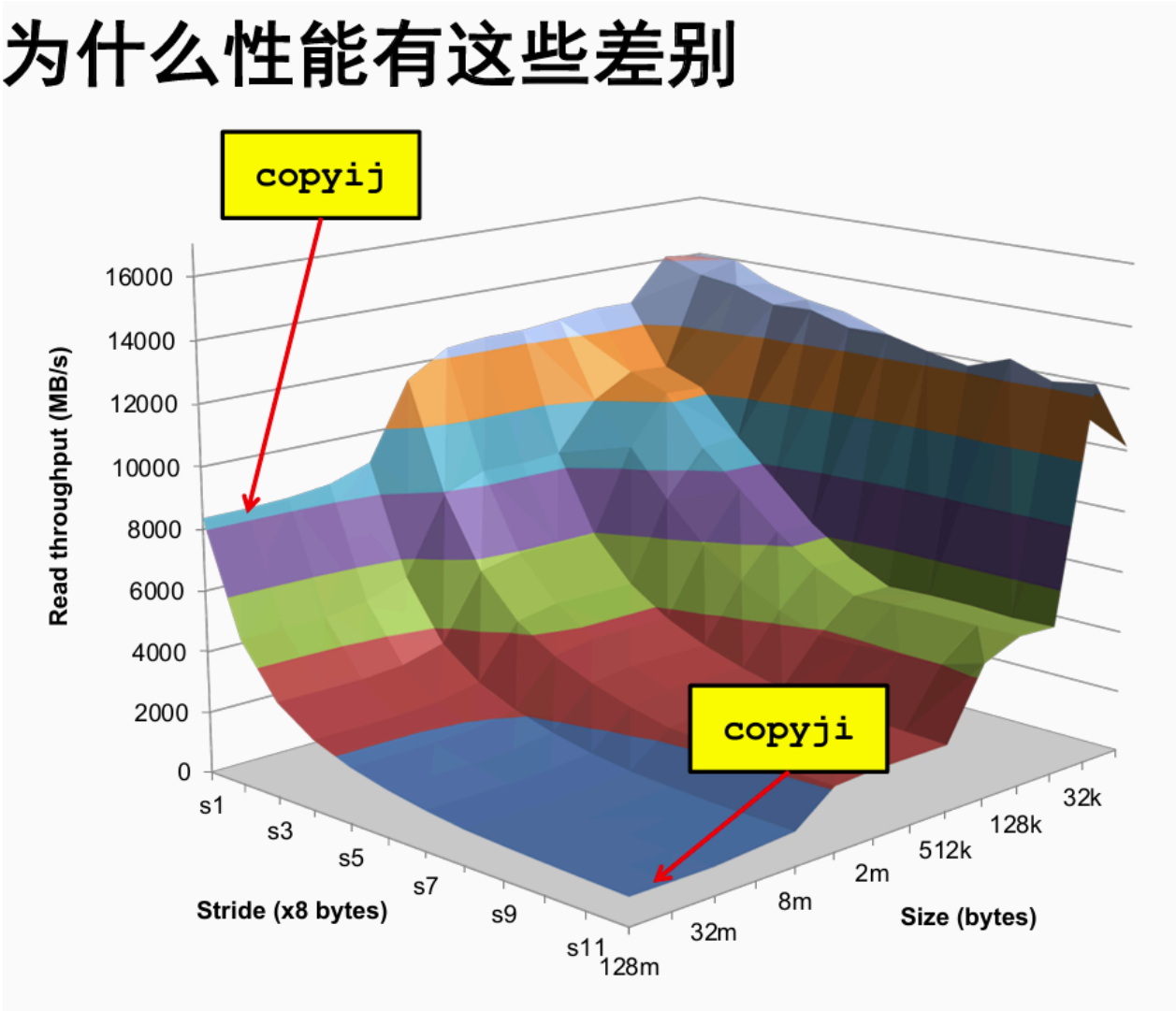


- 内存引用错误
- C 和 C++并未提供对此类错误的防范机制，如
  - 数组越界错误
  - 指针错误
  - 滥用 malloc/free 函数
- 应对措施
  - 其他语言编程
  - 使用工具来检测此类内存错误

- 4. 算法性能分析结果≠实际程序性能
  - 代码写的好坏与否，可能导致程序性能的数量级差别
  - 程序性能优化有多个层面： 算法，数据表达，过程，循环
  - 只有理解了系统实现才能做到有效优化
    - 衡量程序性能的指标：执行时间、内存占用、能耗等
    - 了解程序的编译、执行过程中的细节，如内存访问模式
    - 【例 4.1】内存性能影响程序性能



  - 内存是分层组织的
  - 程序性能取决于内存访问模式，例如访问内存中的多维数组



- 5. 计算机网络环境下的新问题
  - 计算机需要输入与输出数据，在网络环境下，数据输入来源：本地磁盘， 网络中别的计算机。例如，利用上传数据到服务器，利用服务器的超强计算能力做仿真实验
  - I/O 系统对程序稳定性和性能至关重要
  - 如何保证网络中数据的正确性/可靠性（当计算机从网络中别的机器获得数据时）
  - 不同计算机引起的并发操作相互干扰
  - 【例 5.1】网上商店某仓库存量 100，入库业务员读取库存，进货 50，更新库存；出库业务员读取库存，出货 40，更新库存；若上述读取库存操作同时发生，则更新后库存可能为 150 或 60，导致“更新丢失”错误