计算机系统基础
Computer System Fundamentals

想听什么？你来定义！

shidinglin@gmail.com
@林仕鼎

欢迎入群，
一起讨论，
共同定义课程！

计算机系统课程

该微信群二维码将在2015年7月19日失效

Class 1

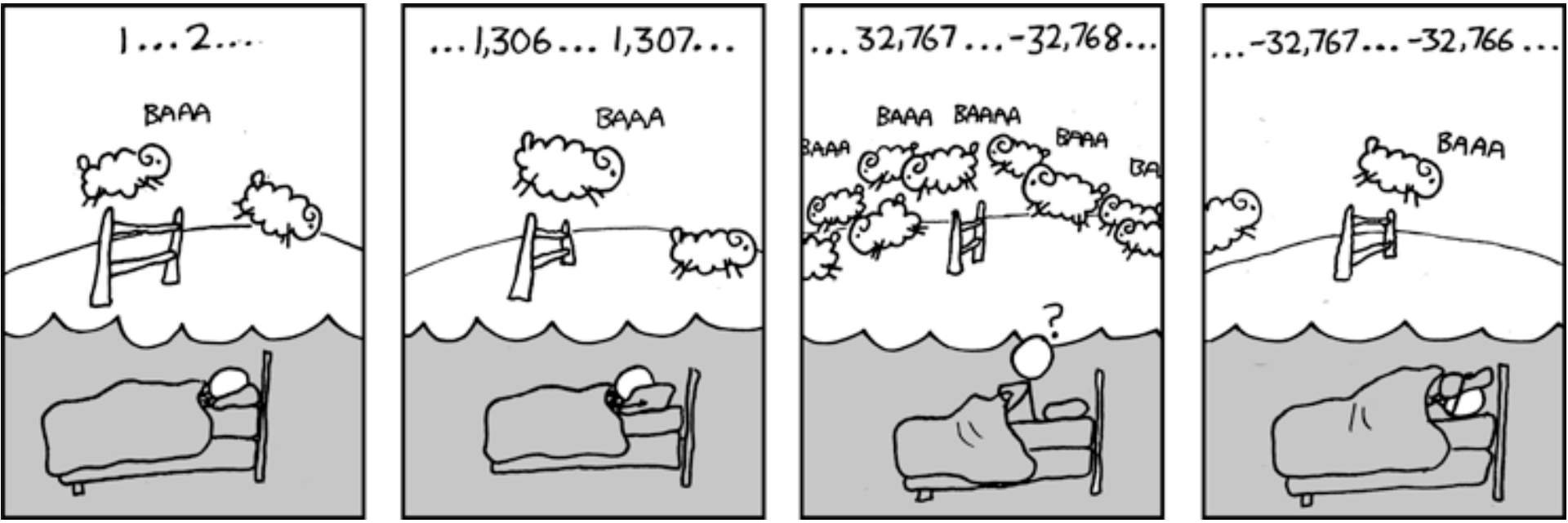
计算机系统概述

@林仕鼎
shidinglin@gmail.com

注: 本节内容大量参考《Computer Systems: A Programmer's Perspective》一书及课件

Agenda

- 为什么要上这门课?
- 课程总体介绍
- 信息的表示



Source: xkcd.com/571

这段代码有什么问题?

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];
size_t memcpy(void*, void*, size_t);

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

如果这么用呢?

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];
size_t memcpy(void*, void*, size_t);

/* Copy at most maxlen bytes from kernel region to user buffer */
int copy_from_kernel(void *user_dest, int maxlen) {
    /* Byte count len is minimum of buffer size and maxlen */
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

了解你的系统 – 1

- Q1: Is $x^2 \geq 0$?
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??

int不是整数，
float不是实数

- Q2: Is (x + y) + z = x + (y + z)?
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 → 3.14
    - 1e20 + (-1e20 + 3.14) → ??

什么返回值?

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

假设Intel x86架构

fun(0) → ?
fun(1) → ?
fun(2) → ?
fun(3) → ?
fun(4) → ?

返回值

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

假设Intel x86架构

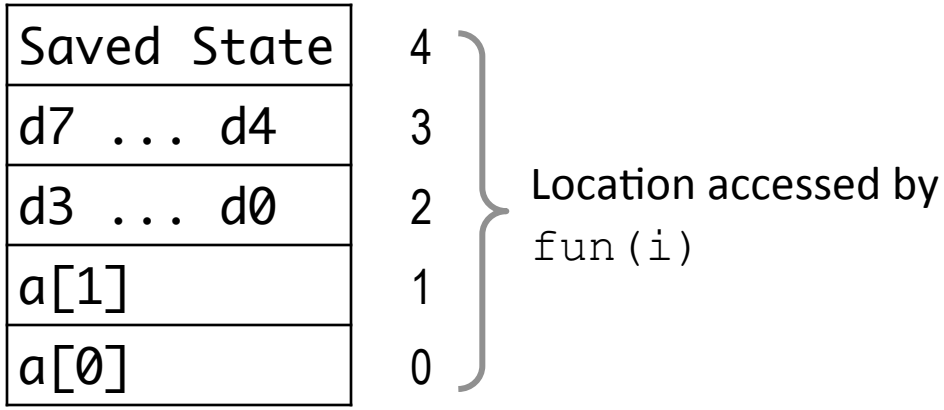
fun(0) → 3.14
fun(1) → 3.14
fun(2) → 3.1399998664856
fun(3) → 2.0000061035156
fun(4) → 3.14, then segmentation fault

原因

```
double fun(int i)
{
  volatile double d[1] = {3.14};
  volatile long int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  → 3.14
fun(1)  → 3.14
fun(2)  → 3.1399998664856
fun(3)  → 2.0000061035156
fun(4)  → 3.14, then segmentation fault
```

Explanation:



了解你的系统 – 2

- 内存需要被管理和分配
  - 堆 vs. 栈
- 在C/C++中内存是不受保护的
  - 数组越界
  - 无效指针
- 内存错误可能导致非常诡异的问题
  - 症状跟系统和编译器的实现相关
  - 可能延迟很久才出现
- 借助内存引用检查工具 (e.g. valgrind)

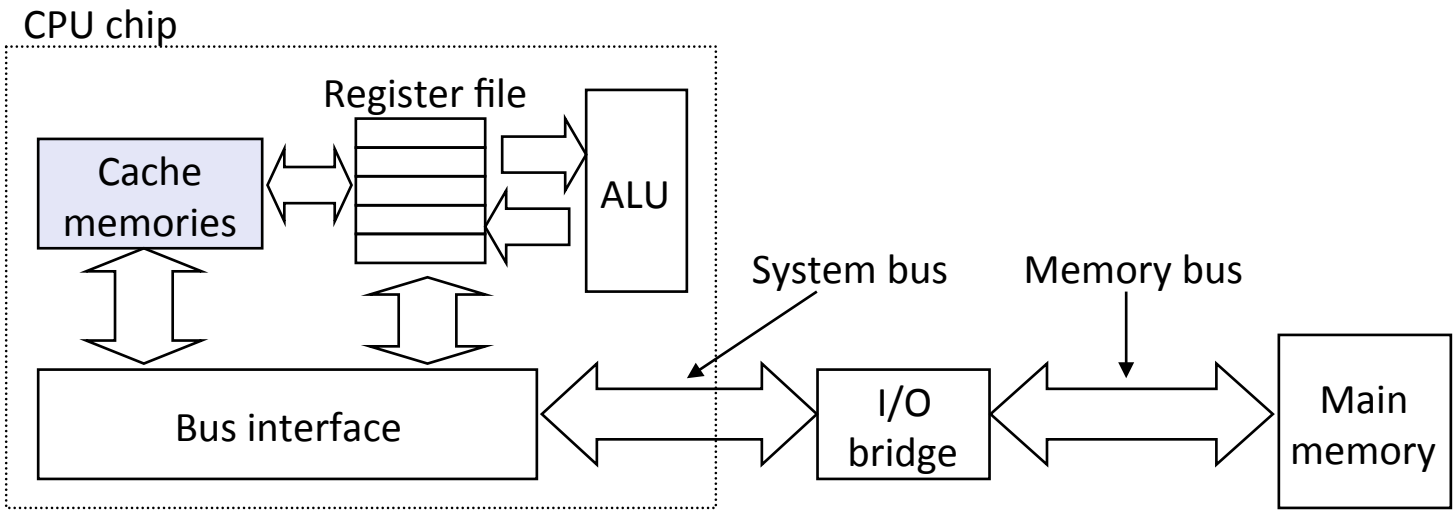
哪个比较快?

```
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];
}
```

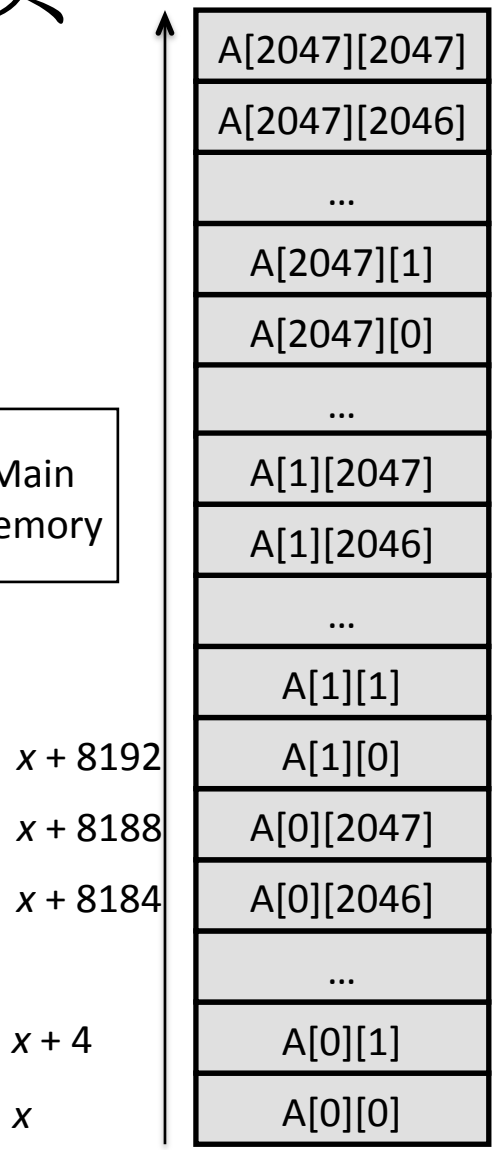
Pentium 4上性能相差21倍!

数组: 从概念到现实



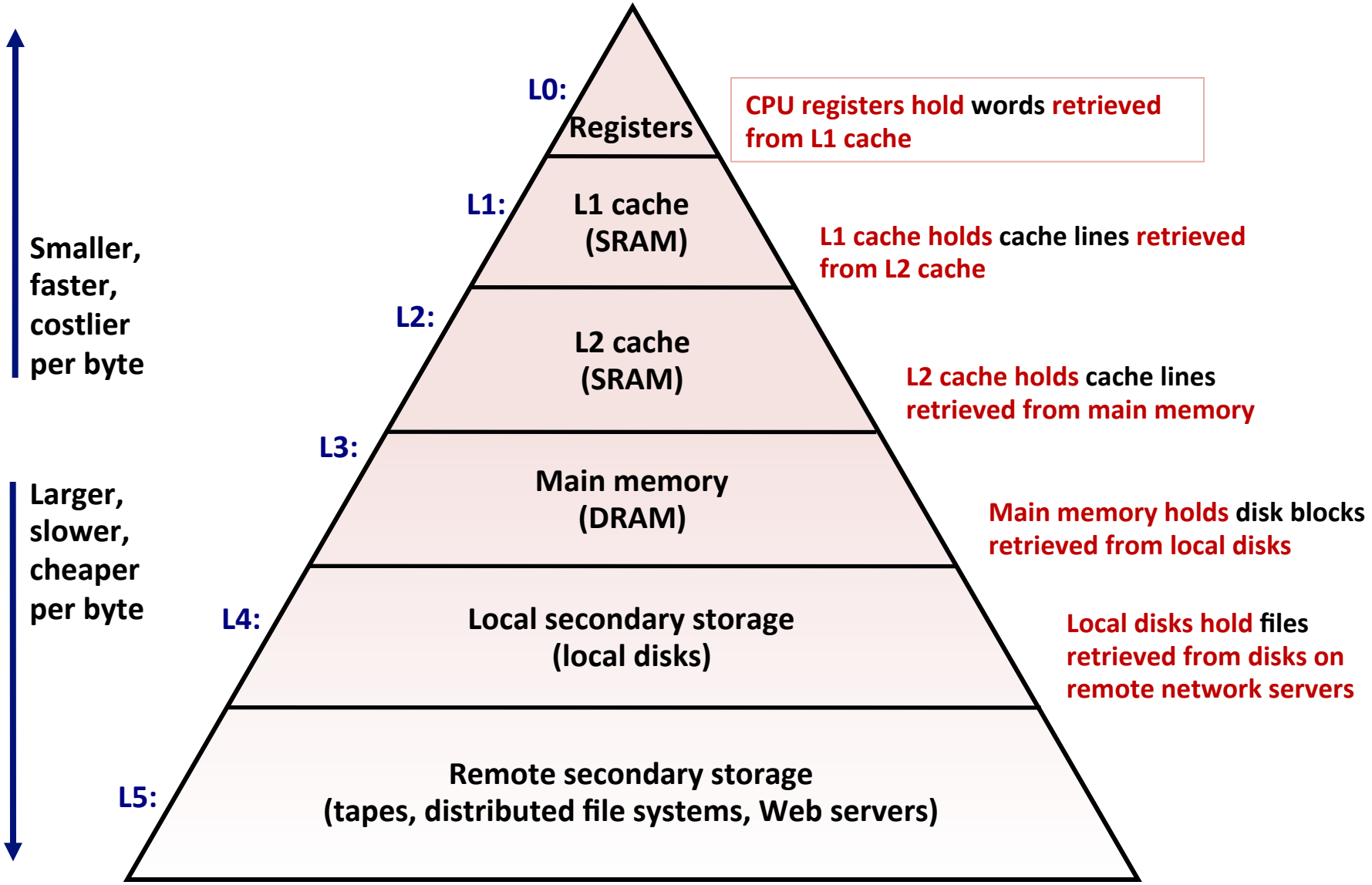
典型的系统结构

位置	延迟(cycles)	介质
L1 Cache	1	SRAM
L2 Cache	10	SRAM
Memory	100	DRAM



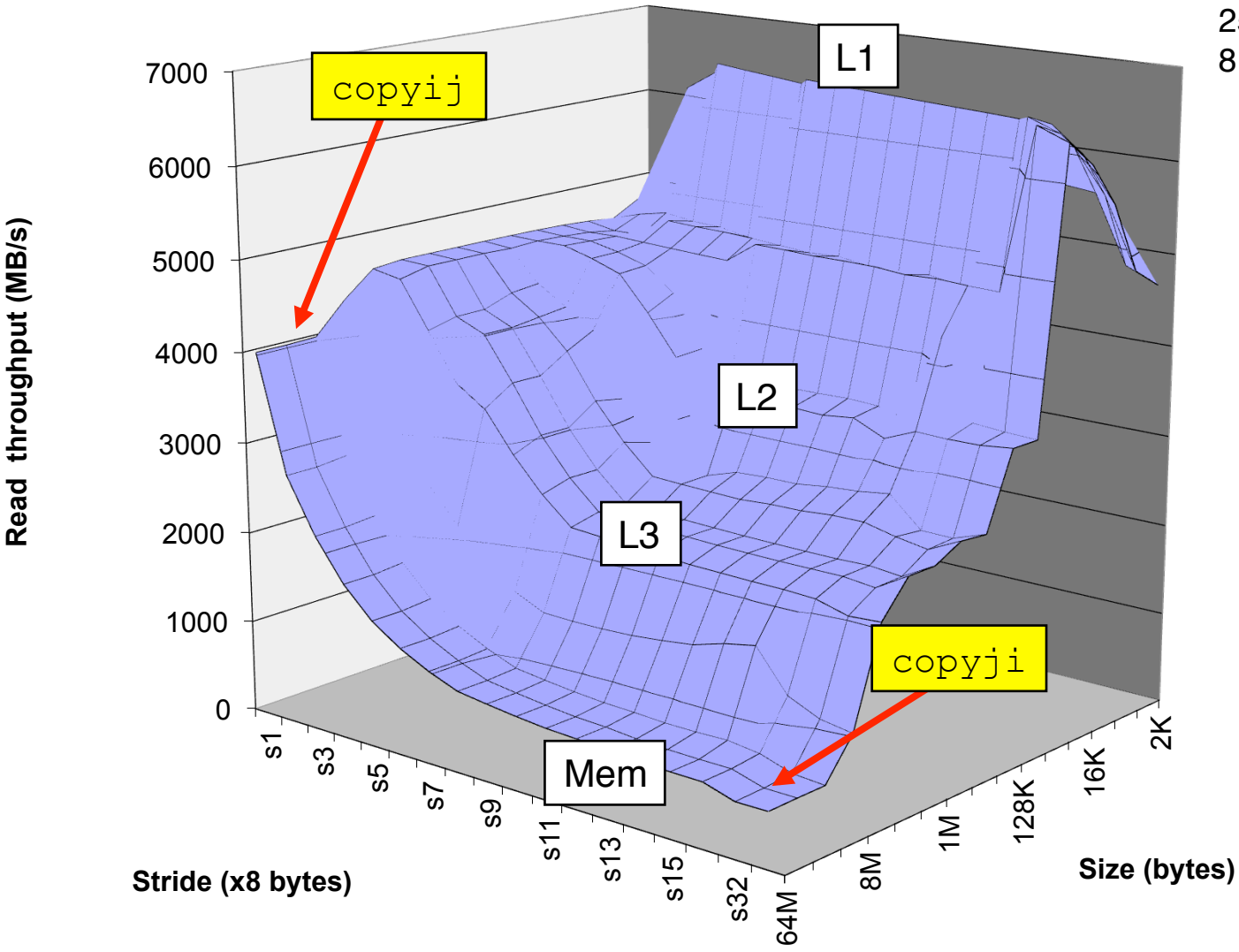
数组的内存布局

典型的Memory Hierarchy



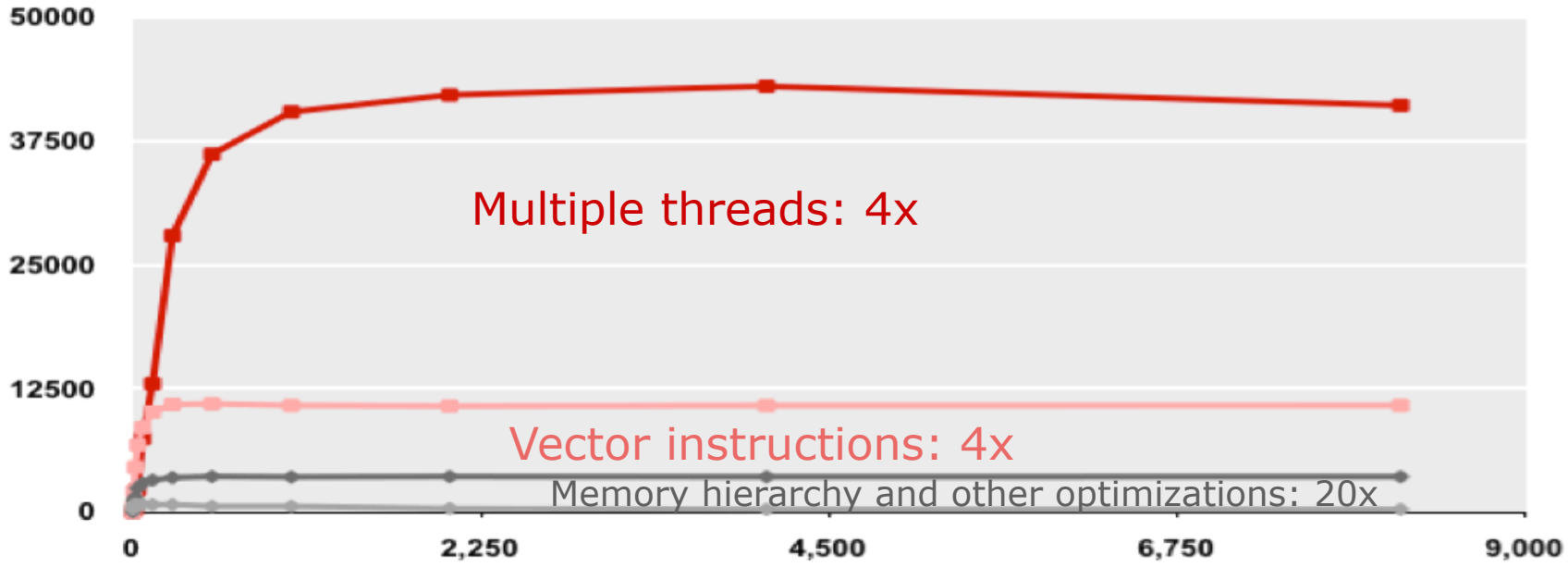
The Memory Mountain

Intel Core i7
2.67 GHz
32 KB L1 d-cache
256 KB L2 cache
8 MB L3 cache



了解你的系统 – 3

- 内存的性能不是均匀的
  - cache和虚拟内存特性对性能有极大的影响
  - 使程序适应于内存的特性
- 即使算法复杂度相同，程序性能可能仍有几十倍的差异



如何精确计时 – 1

- Linux
  - 时钟中断周期: 10ms，#define HZ 100
- 采用x86硬件寄存器Time Stamp Counter
  - 记录CPU执行的cycle数
  - 指令: rdtsc

```
uint64 t1, t2;
t1 = get_counter();
func();
t2 = get_counter();
printf("func required %u64 clock cycles, %f seconds\n",
        t2-t1, (double)(t2-t1)/FREQUENCY);
```

如何精确计时 – 2

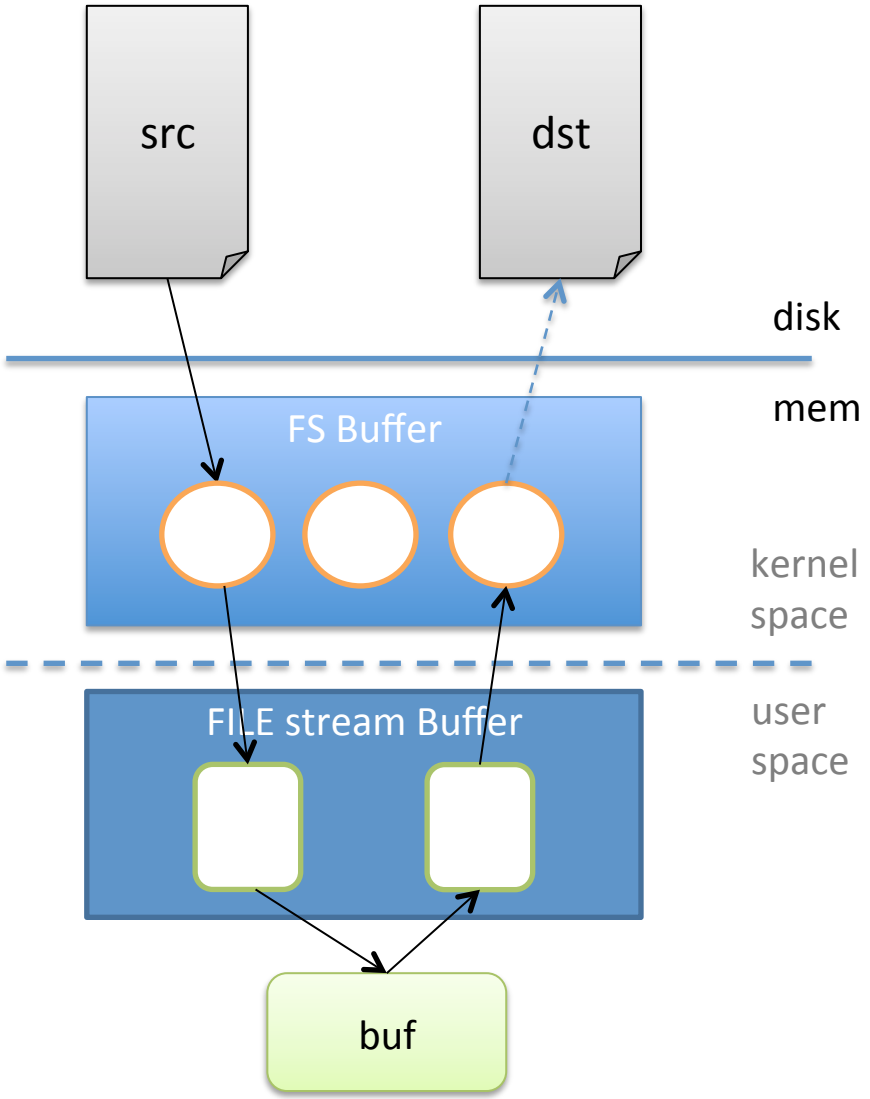
```
/* Set *hi and *lo to the high and low order bits
 *  of the cycle counter.
 */
void access_counter(uint32 *hi, uint32 *lo)
{
    asm("rdtsc; movl %%edx,%0; movl %%eax,%1"
        : "=r" (*hi), "=r" (*lo)
        :
        : "%edx", "%eax");
}

uint64 get_counter()
{
    uint64 t;
    access_counter(((uint32*)&t)+1, (uint32*)&t);
    return t;
}
```

如何高效拷贝文件－1

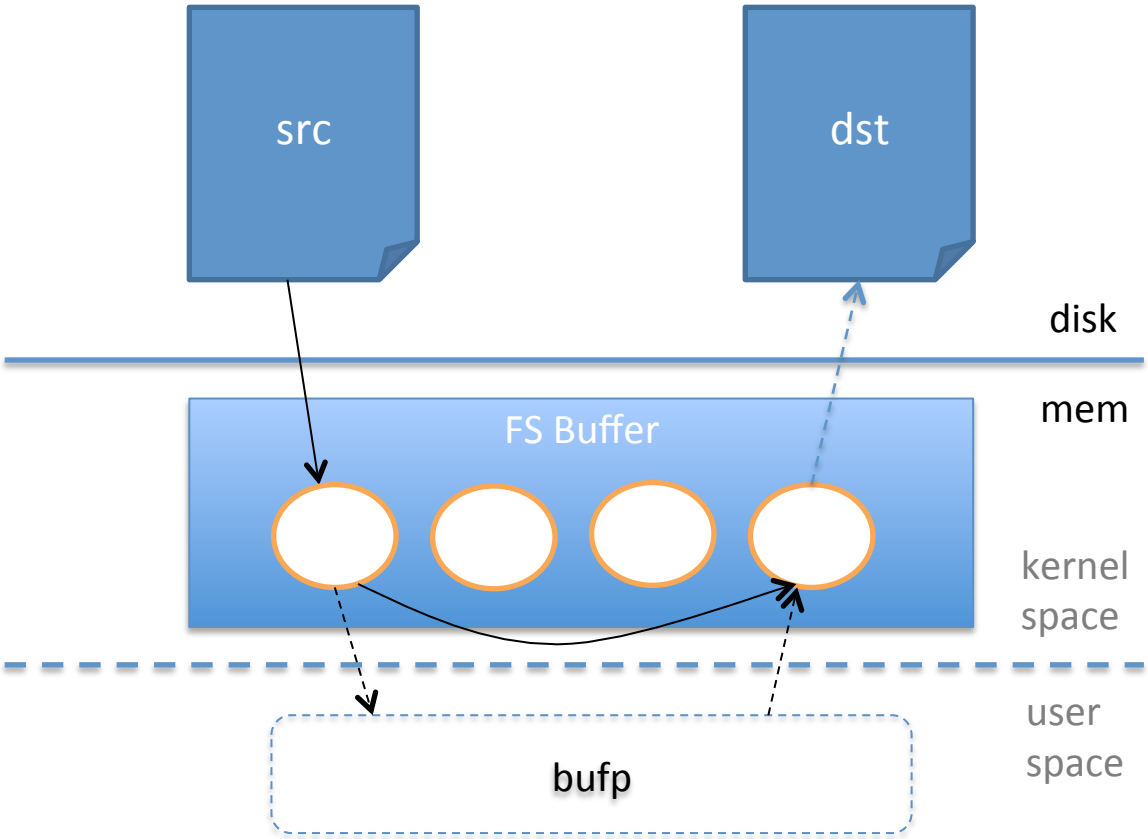
```
/* Note: to simplify the implementation,
 * no failure handling is provided.
 */
void copyfile(FILE *dst_fp, FILE *src_fp,
              int size)
{
    char buf[4096];

    while (size > 4096) {
        fread(buf, 4096, 1, src_fp);
        fwrite(buf, 4096, 1, dst_fp);
        size -= 4096;
    }
    if (size > 0) {
        fread(buf, size, 1, src_fp);
        fwrite(buf, size, 1, dst_fp);
    }
}
```



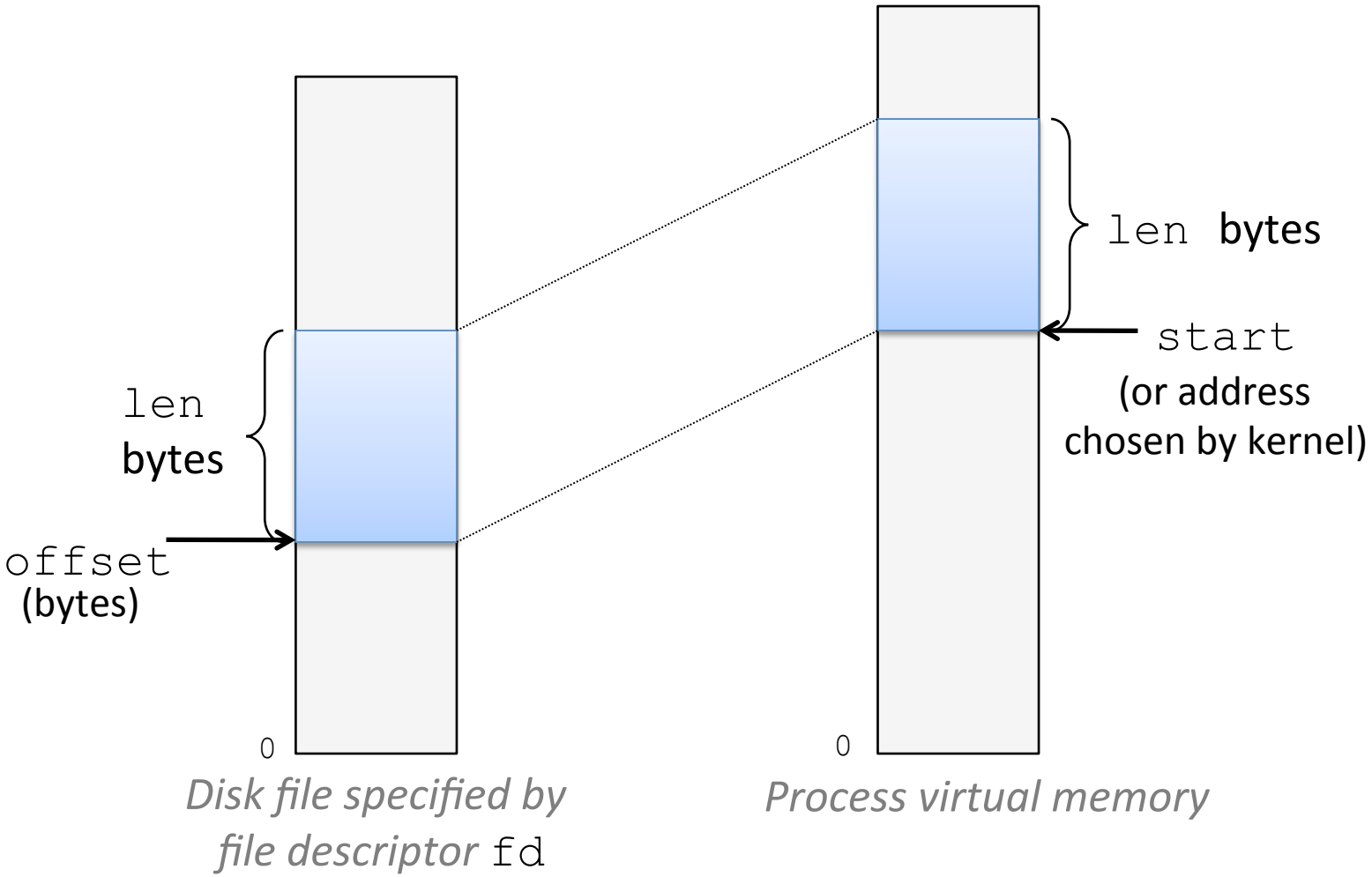
如何高效拷贝文件 – 2

```
void mmapcopy(int dst_fd, int src_fd, int size)
{
    char *bufp;
    bufp = mmap(NULL, size, PROT_READ, MAP_PRIVATE, src_fd, 0);
    write(dst_fd, bufp, size);
}
```



mmap

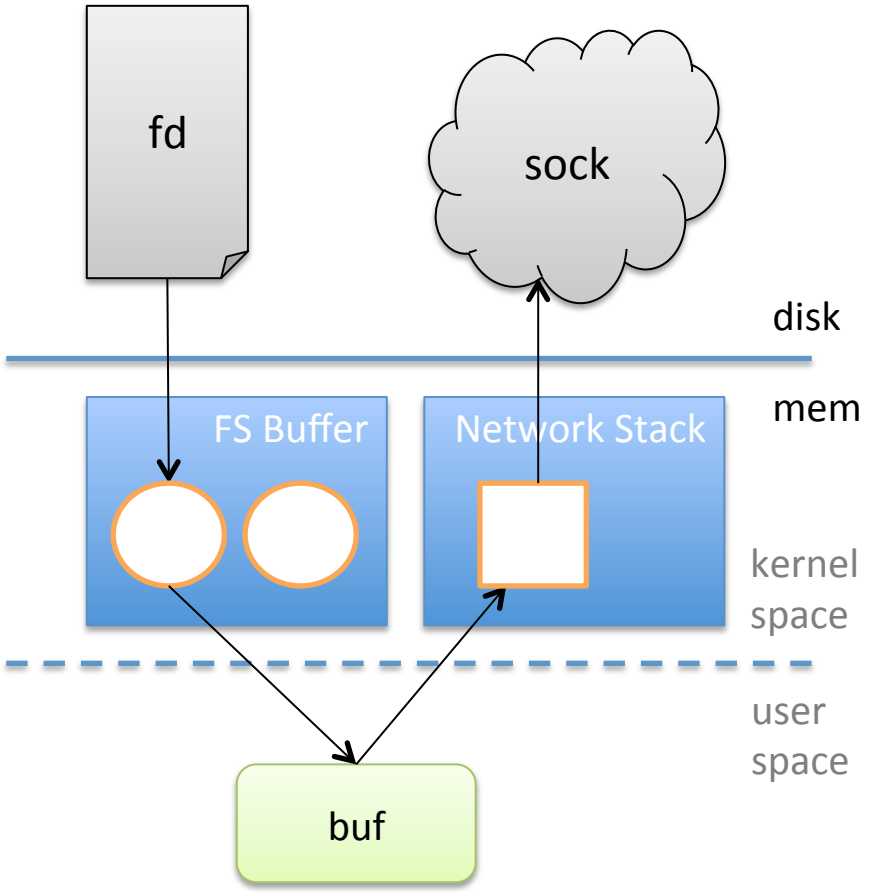
```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset);
```



如何高效传输远程文件－1

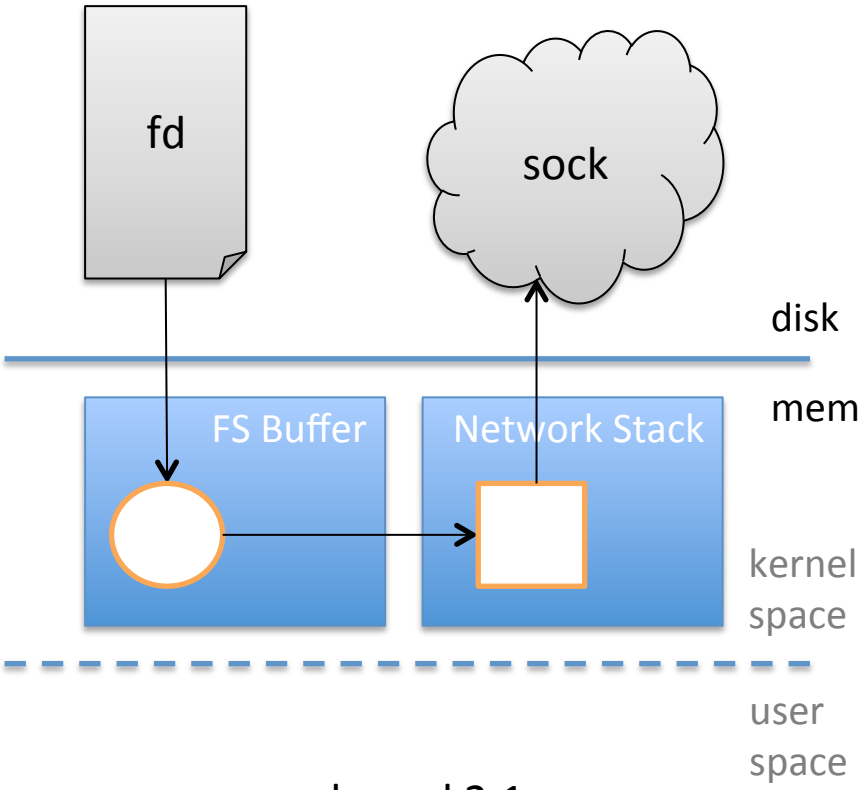
```
/* Note: to simplify the implementation,
 * no failure handling is provided.
 */
void tranfile(int sock, int fd,
              int size)
{
    char buf[4096];

    while (size > 4096) {
        read(fd, buf, 4096);
        write(sock, buf, 4096);
        size -= 4096;
    }
    if (size > 0) {
        read(fd, buf, size);
        write(sock, buf, size);
    }
}
```

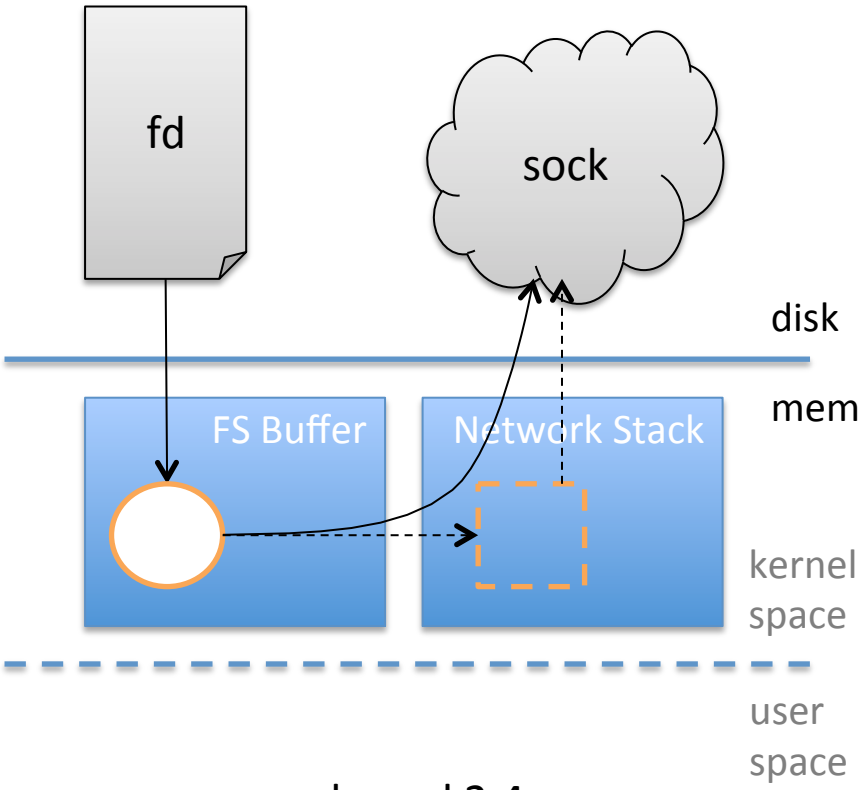


如何高效传输远程文件 – 2

```
ssize_t sendfile(int sock, int fd,
                 off_t *offset, size_t count);
```



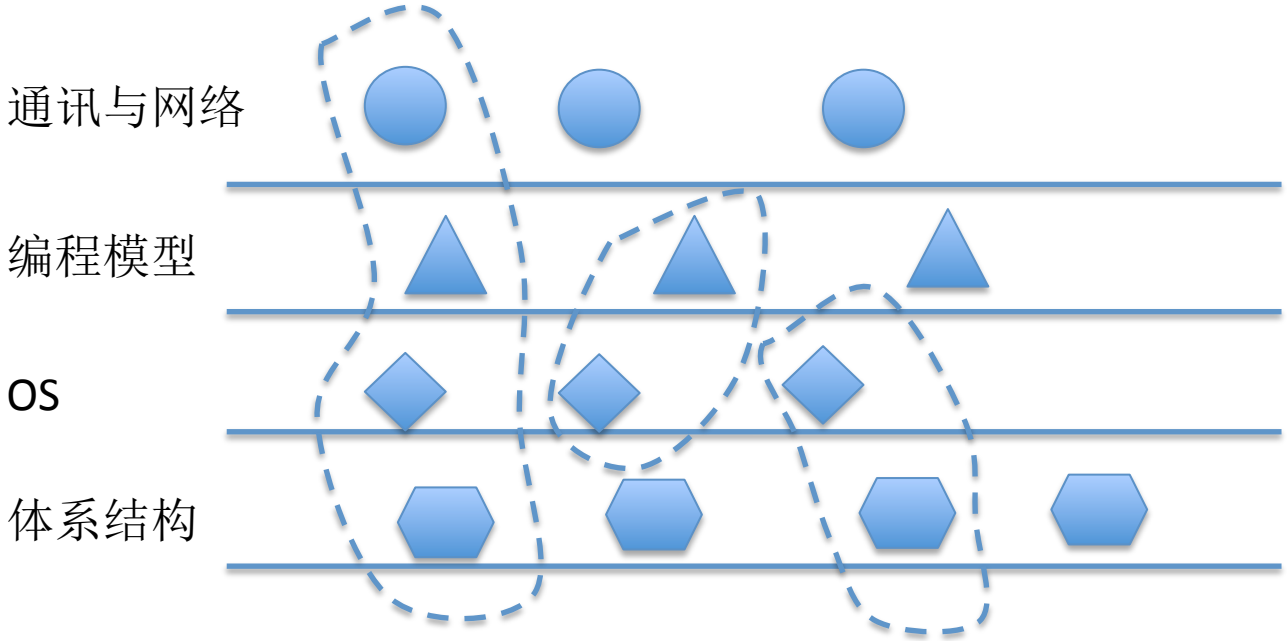
kernel 2.1



kernel 2.4

你需要知道的还有很多

- CPU的工作机理
- 计算机的功能部件与特点
- OS的工作机理
- 编译器的实现
- 软硬件如何配合、各软件层次如何配合



Agenda

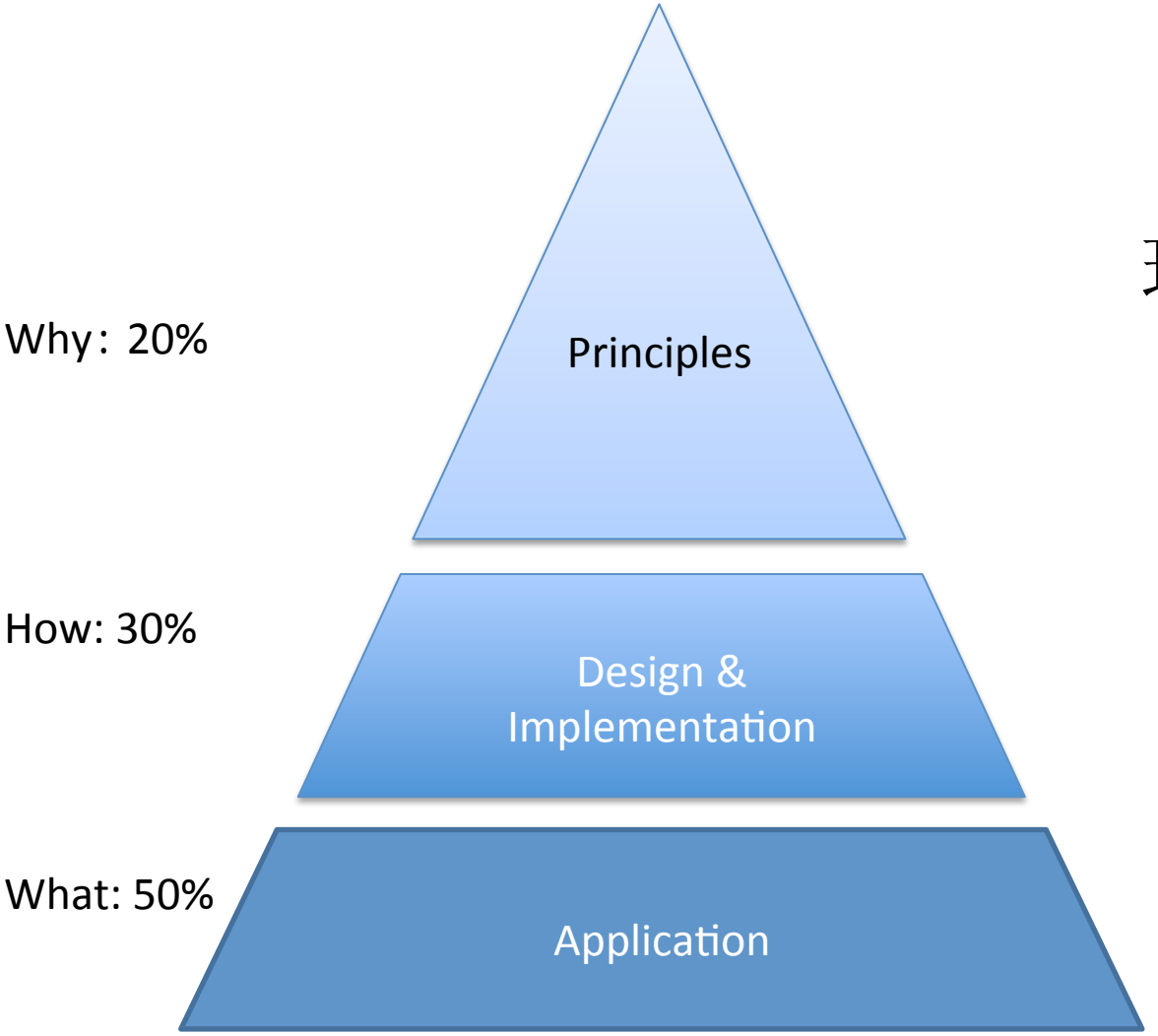
- *为什么要上这门课?*
- **课程总体介绍**
- 信息的表示

课程内容

- 涉及5大基础领域
  - Computer Architecture
  - Operating Systems
  - Programming Model
  - Networking
  - Distributed Systems

- 目标：理解你看到的系统
  - 从场景和实例出发介绍概念
  - 兼顾应用和实现原理 (程序员视角 + 系统工程师视角)
  - 体系化，强调系统全景

内容组织方式



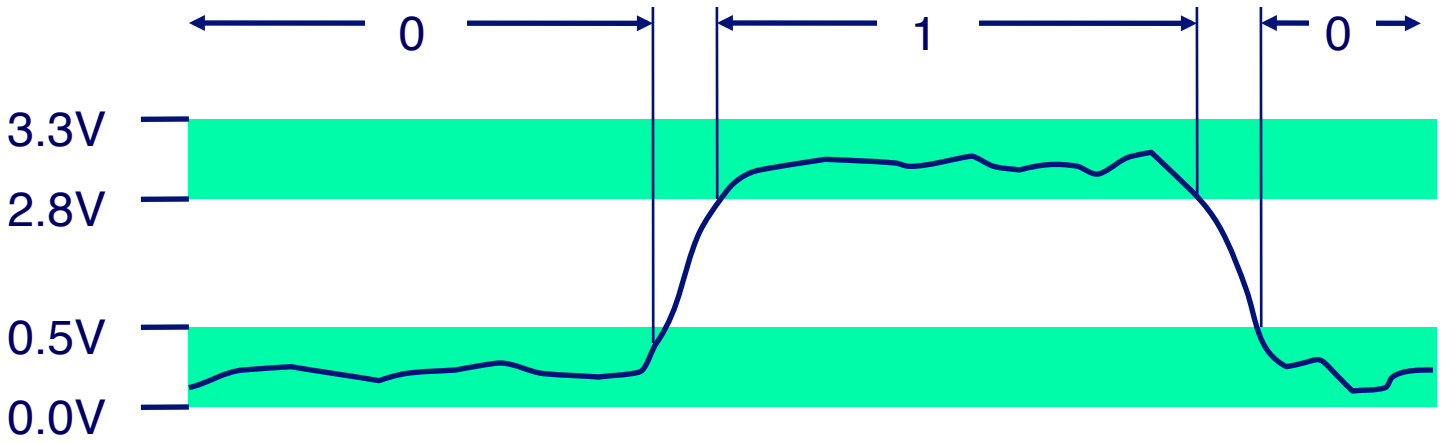
理解你的系统：

1. 实例 + 概念
2. 应用 + 原理
3. 体系化

Agenda

- *为什么要上这门课?*
- *课程总体介绍*
- 信息的表示

二进制的物理表示



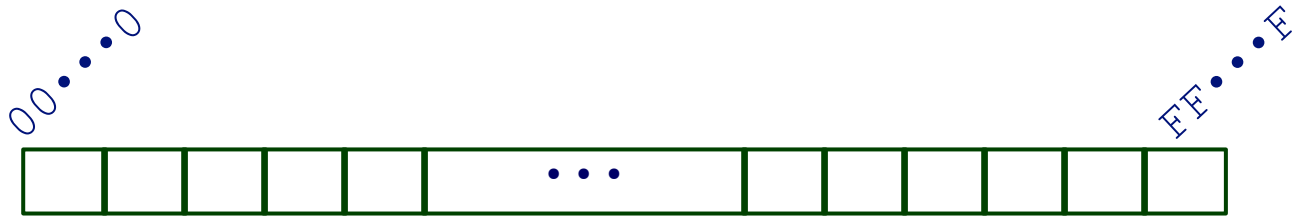
- 简单
- 抗干扰性强

用bit组成byte

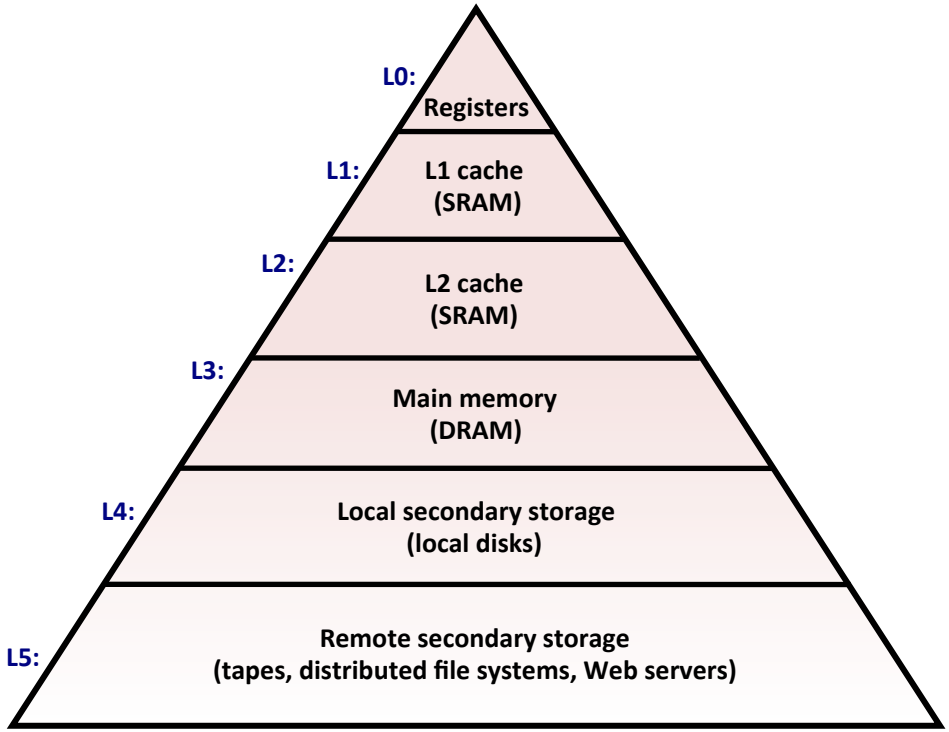
- Byte = 8 bits
  - 2进制: 00000000_2 ~ 11111111_2
  - 10进制: 0_{10} ~ 255_{10}
  - 16进制: 00_{16} ~ FF_{16}
    - '0' ~ '9', 'A' ~ 'F'
    - $FA1D37B_{16}$ 在C语言中的表示
      - 0xFA1D37B
      - 0xfa1d37b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

内存以byte组成



- 逻辑上是一个大“字节数组”
- 虚拟内存
  - 地址空间
    - 进程私有
    - 连续: 0~2^w-1，但有空洞
  - 虚实映射
  - 动态分配: malloc
  - 多种形式: static, stack, heap
- 实际存储
  - Memory Hierarchy



计算机的字 (word)

- 字长 (word size)
  - 操作数宽度，地址宽度
- 一般机器采用32位字长 (4bytes)
  - 地址空间4G
- 高端机器采用64位字长 (8bytes)
  - 地址空间$1.8x10^{19}$
  - x86-64使用48位地址: 256TB
- 多种操作数宽度
  - 1, 2, 4, 8 bytes

数据类型的宽度

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1	1	1
short	2	2	2
int	4	4	4
long	4	4	8
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer	4	4	8

```
printf("%d\n", sizeof(long));
```

字节序 (byte ordering)

- 一个word在内存中如何以byte存放?
  - 在字节流中如何排列?
  - 交换数据时必须一致

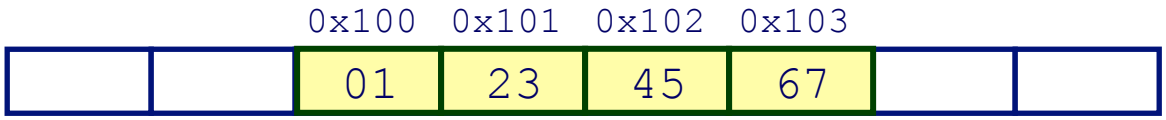
- 两个传统
  - 大尾端 (Big Endian): Sun, PPC Mac, Internet
    - LSB (Least Significant Byte)在高地址
  - 小尾端 (Little Endian): x86
    - LSB在低地址

字节序的例子

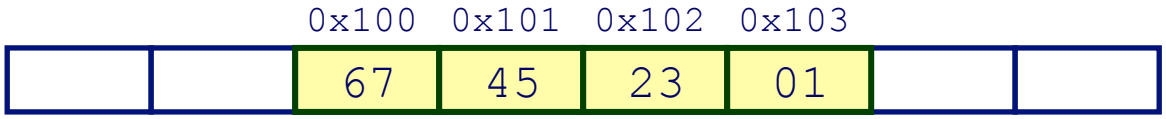
- 假设int x = 0x01234567， &x=0x100

MSB　　LSB　　起始地址，渐增4Bytes

- 大尾端
  - LSB在高地址 (阅读序)



- 小尾端
  - LSB在低地址 (逆阅读序)



看一段汇编代码

- 汇编
  - 二进制机器码的文本表示，语义上等价
  - 可由程序自动翻译, objdump

- 例子

```
地址          机器码                          汇编表示
8048365:    5b                              pop    %ebx
8048366:    81 c3 ab 12 00 00               add    $0x12ab,%ebx
804836c:    83 bb 28 00 00 00 00            cmpl   $0x0,0x28(%ebx)
```

- 如何读数
  - 值: 0x12ab
  - 填满32bits (padding): 0x000012ab
  - 按byte分解: 00 00 12 ab
  - 倒序: ab 12 00 00

从byte到数据类型

- 整数 int
- 浮点数 float
- 字符串 string
- 指针 pointer

- 可把任意数据类型转化为字节数组来观测
  - 用unsigned char*指向其地址

```
typedef unsigned char *pointer;

void show_bytes(pointer start, int len){
  int i;
  for (i = 0; i < len; i++)
    printf(”%p\t0x%.2x\n", start+i, start[i]);
  printf("\n");
}
```

show_bytes例子

```
int a = 15213;
printf("int a = 15213;\n");
show_bytes((pointer)&a, sizeof(int));
```

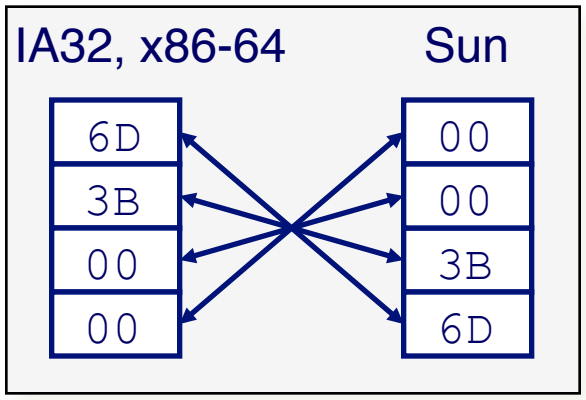
Result (Linux):

```
int a = 15213;
0x11ffffcb8 0x6d
0x11ffffcb9 0x3b
0x11ffffcba 0x00
0x11ffffcbb 0x00
```

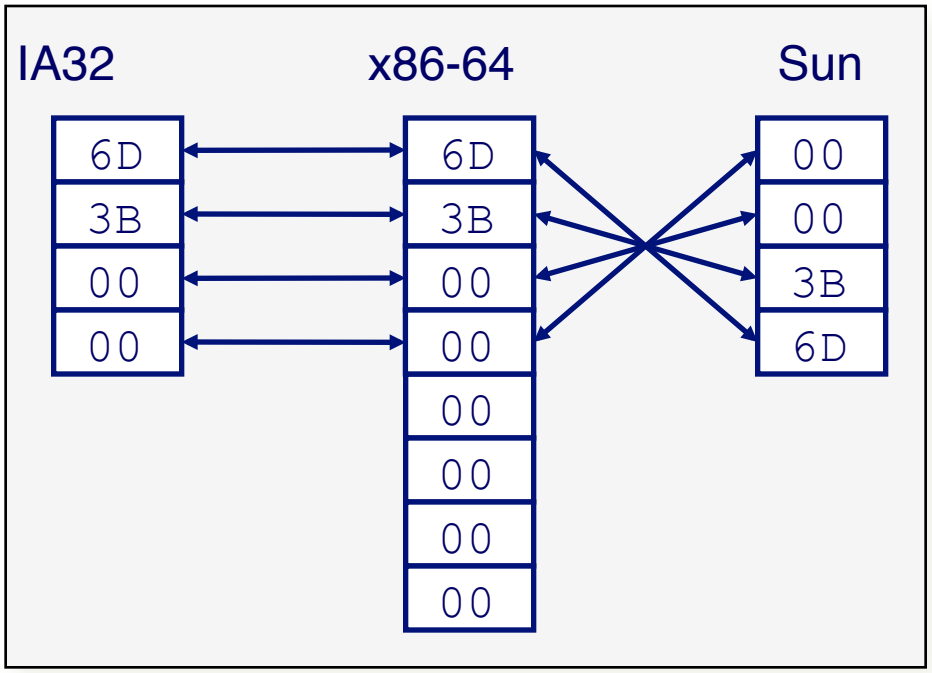
整数的表示

Decimal:	15213
Binary:	0011 1011 0110 1101
Hex:	3 B 6 D

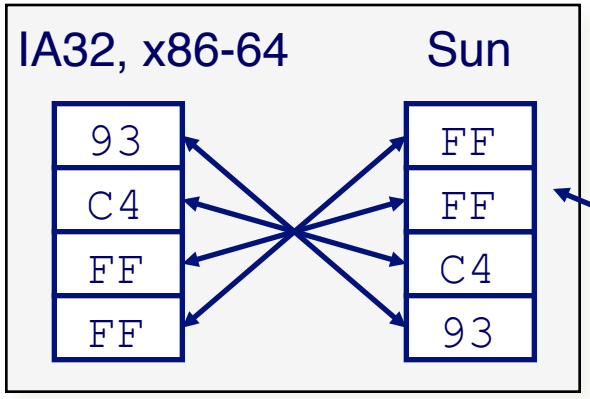
```
int a = 15213;
```




```
long int c = 15213;
```



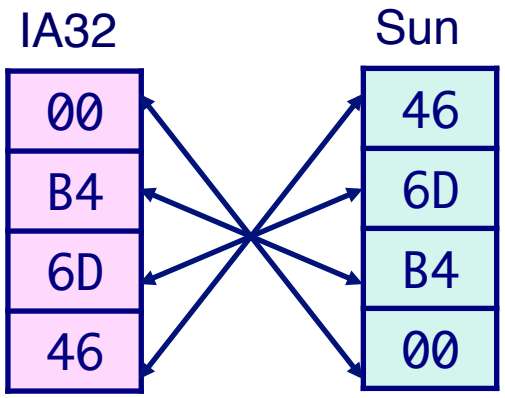

```
int b = -15213;
```



补码表示法

浮点数的表示

```
float f = 15213.0;
```



IEEE单精度浮点数表示法

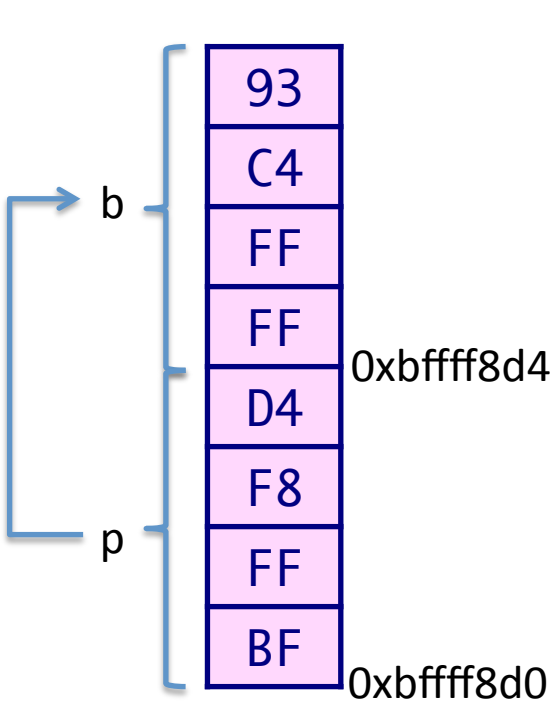
Hex: **4 6 6 D B 4 0 0**

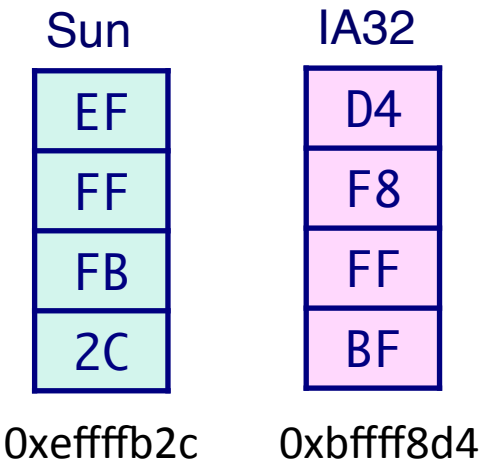
Binary: 0100 0110 0110 1101 1011 0100 0000 0000

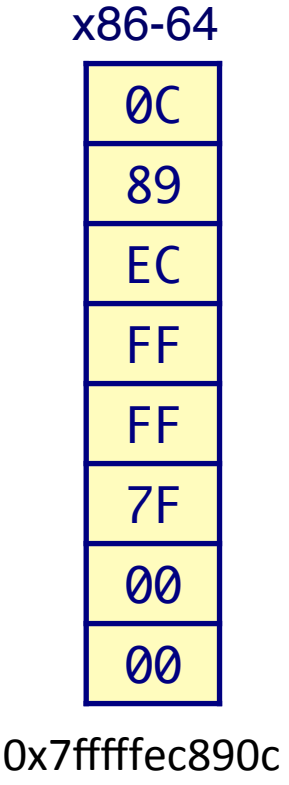
$F = (-1)^s M \ 2^E$

指针的表示

```
int b = -15213;
int *p = &b;
```





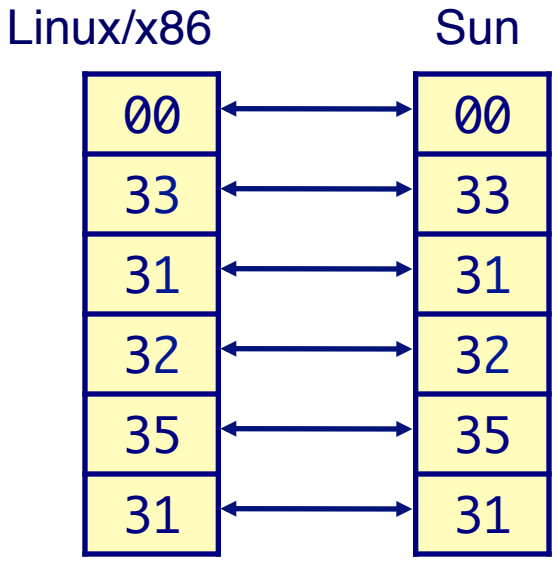


不同的编译器、操作系统和机器会导致地址分配的不同

字符串的表示

```
char S[6] = ”15213";
```

- Strings in C
  - 顾名思义，字符“串”
  - 每个字符以ASCII码表示
    - 7-bit
    - 以0x30表示’0’
      - 数字i = 0x30+i
  - 以0结尾



- 无字节序的问题
  - 文本文件天然跨平台

信息 = bit + presentation

```
int a = 15213;
int b = -15213;
int *p = &b;
float f = 15213.0;
char[6] = “15213”;
void _15213() {…}

*(unsigned int*)&b = ?
*(int*)&f = ?
*(int*)s = ?

p = (int*)(pointer)15123;
*p = ?

p = (int*)(pointer)_15213;
*p = ?
```

作业: 理解补码表示和浮点数表示

云校
yunxiao.com

互联网 + 教育

欢迎加入！

talent@iyunxiao.com