

操作系统

Operating Systems

L28 生磁盘的使用

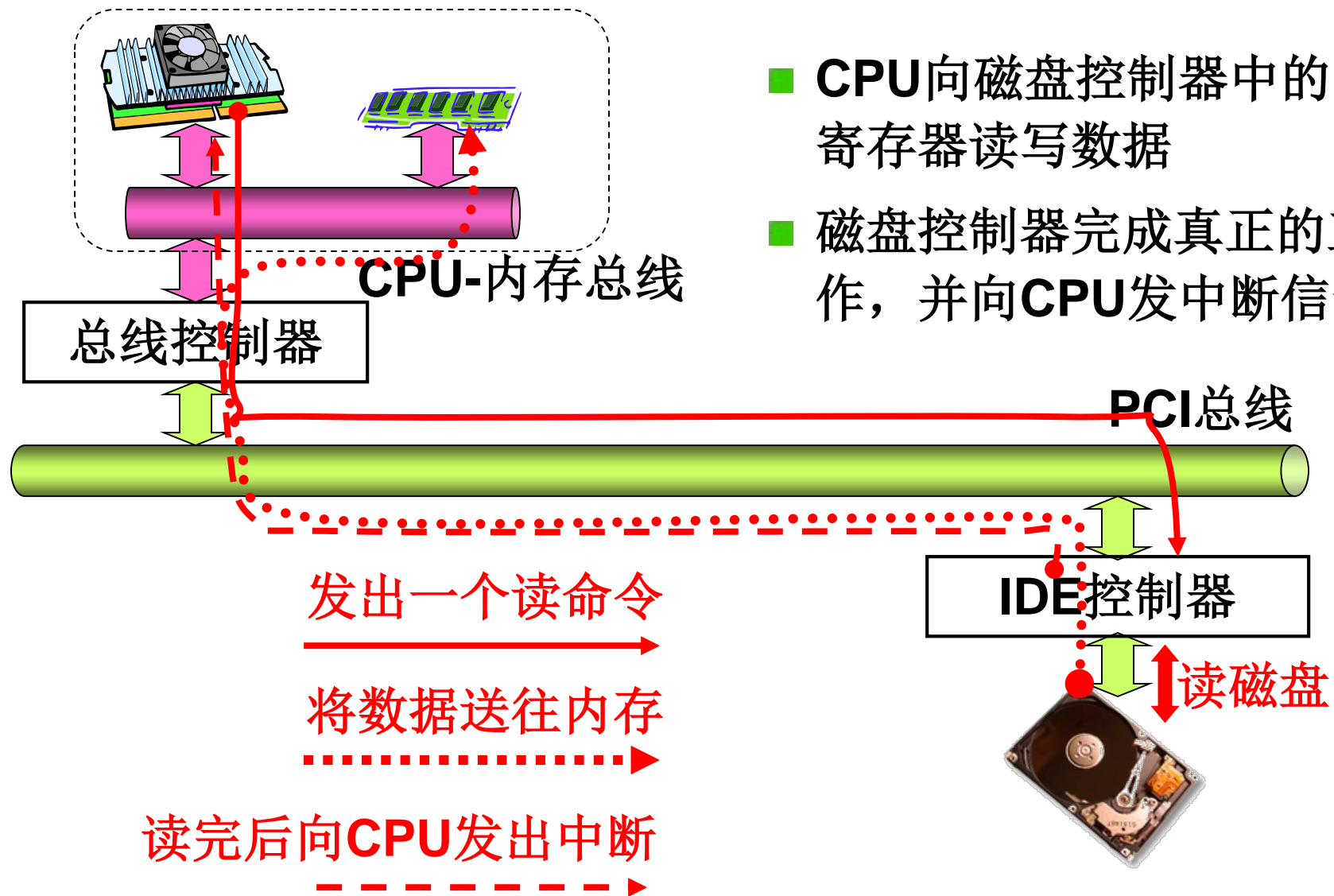
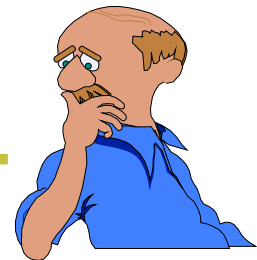
Raw Disks

`lizhijun_os@hit.edu.cn`

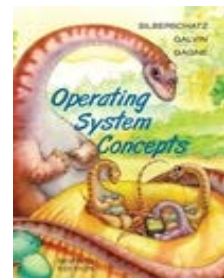
综合楼411室

授课教师：李治军

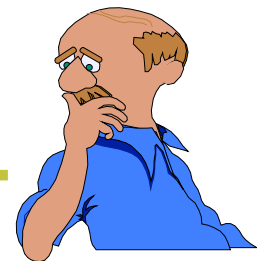
仍然从硬件开始...



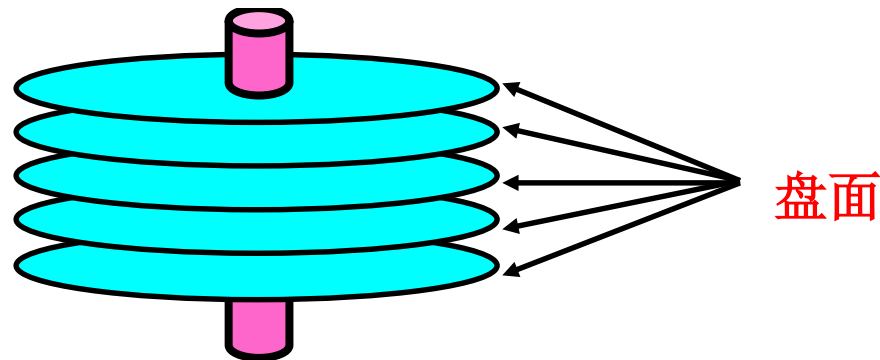
- CPU向磁盘控制器中的寄存器读写数据
- 磁盘控制器完成真正的工作，并向CPU发中断信号



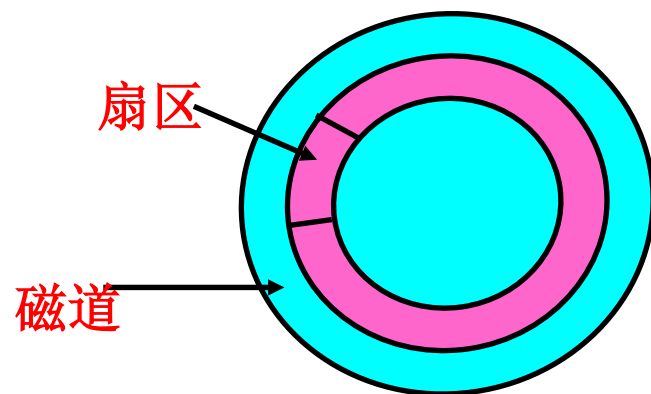
使用磁盘从认识磁盘开始



- 画一个示意图:



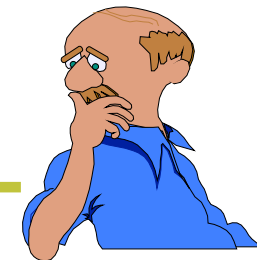
- 看看俯视图:



- 磁盘的访问单位是扇区
- 扇区大小: **512字节**
- 扇区的大小是传输时间和碎片浪费的折衷

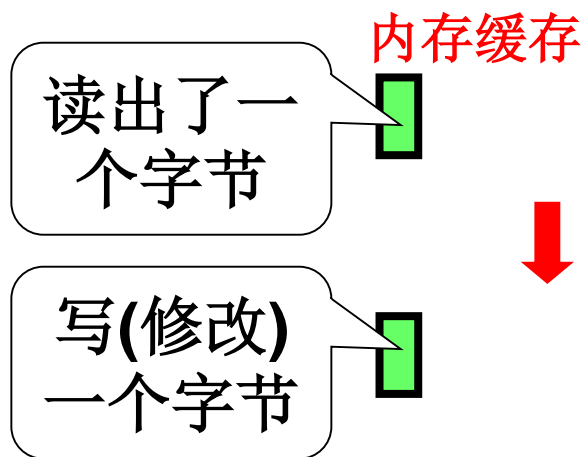


磁盘的I/O过程

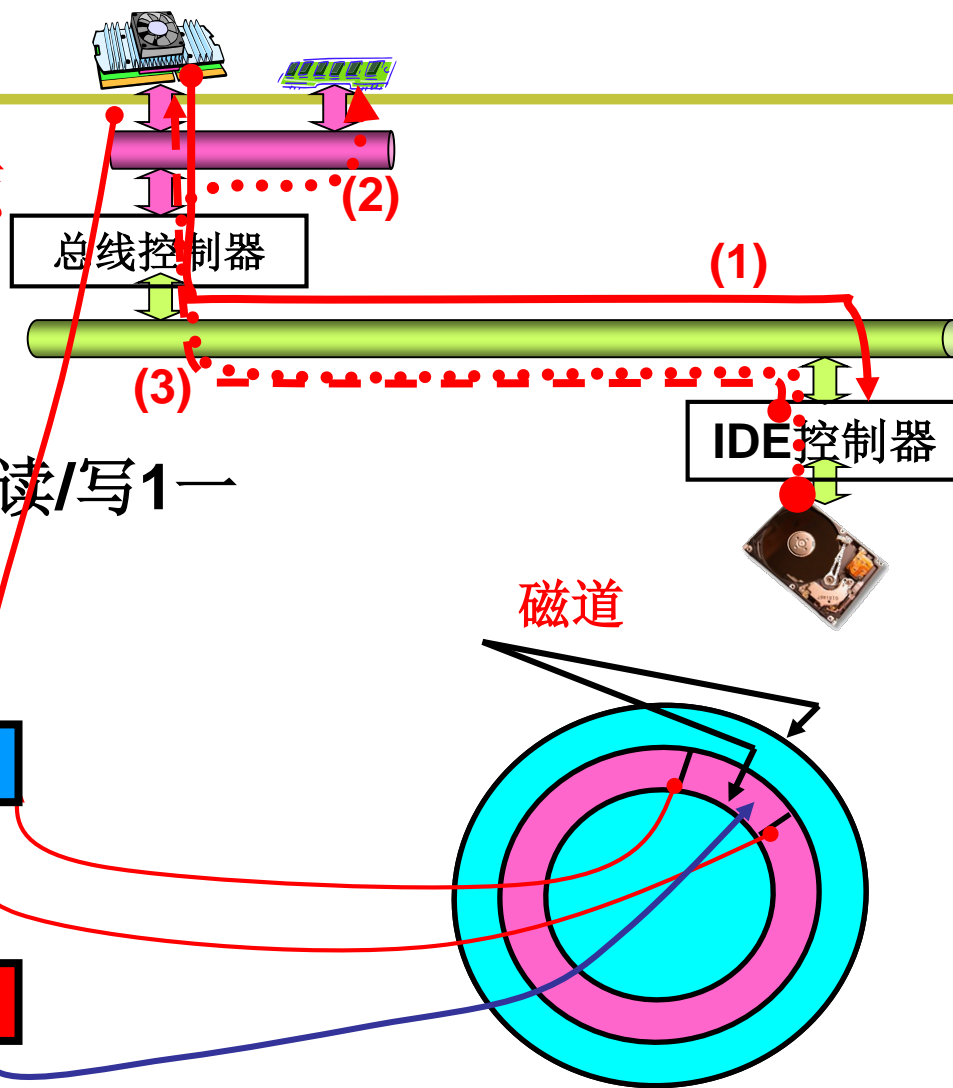


■ 这就要开始使用磁盘了!

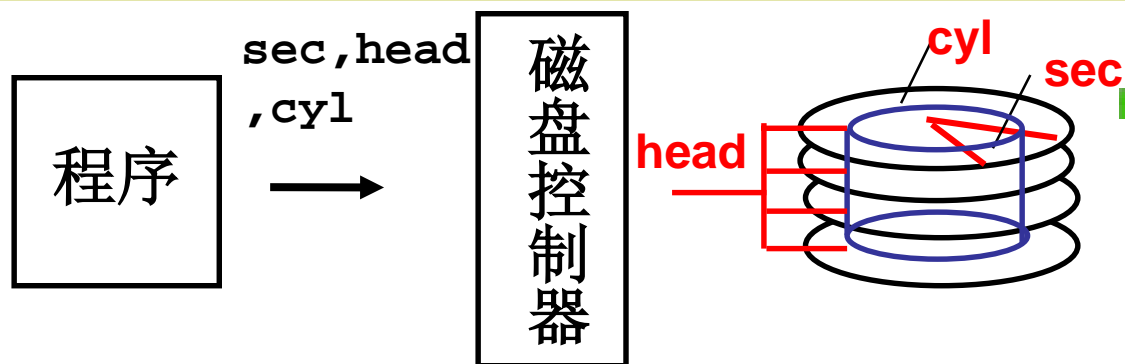
■ 仔细想想磁盘如何读/写1一个字节?



■ 磁盘I/O过程: 控制器→寻道→旋转→传输!



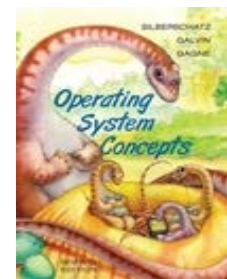
最直接的使用磁盘



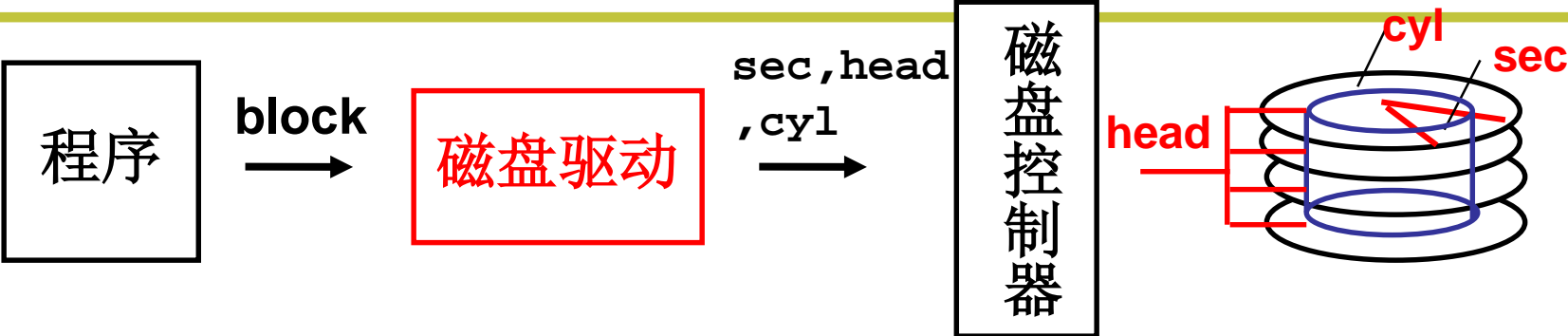
■ 只要往控制器中写柱面、磁头、扇区、缓存位置

```
void do_hd_request(void)
{
    ...hd_out(dev, nsect, sec, head, cyl, WIN_WRITE, ...);
    port_write(HD_DATA, CURRENT->buffer, 256);
}
```

```
void hd_out(drive, nsect, sec, head, cyl, cmd...)
{
    port = HD_DATA; //数据寄存器端口(0x1f0)
    outb_p(nsect, ++port); outb_p(sec, ++port);
    outb_p(cyl, ++port); outb_p(cyl >> 8, ++port);
    outb_p(0xA0 | (drive << 4) | head, ++port);
    outb_p(cmd, ++port);
}
```



通过盘块号读写磁盘(一层抽象)



- 磁盘驱动负责从block计算出cyl, head, sec(CHS)

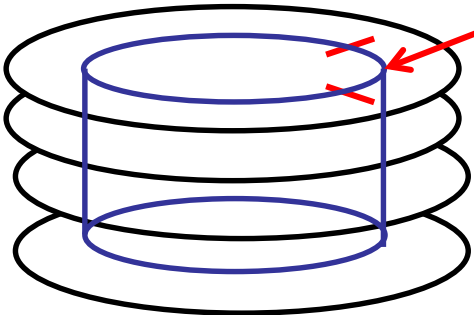
问题: 如何编址? 为什么这样编址? **block相邻的盘块可以快速读出**

磁盘访问时间 = 写入控制器时间 + 寻道时间 + 旋转时间 + 传输时间

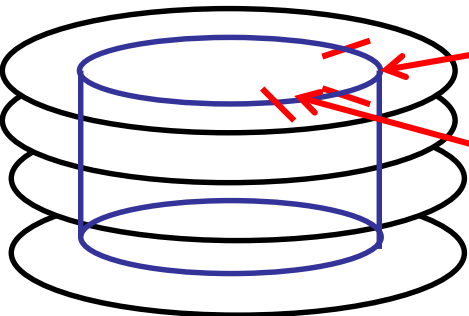
50M/秒, 约0.3ms

12 ms to 8 ms

7200转/分钟: 半周 4 ms



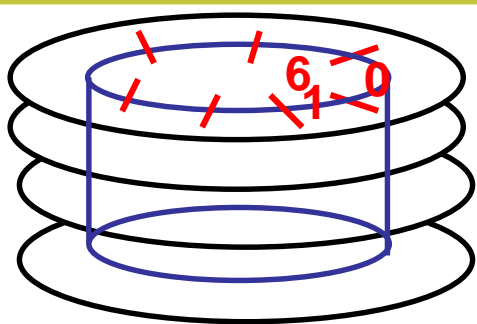
问题: 1号扇区在哪里?



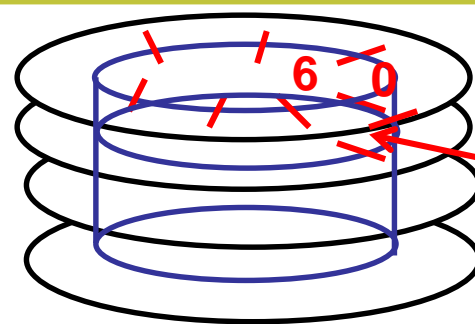
0号扇区
1号扇区



从CHS到扇区号，从扇区到盘块



问题：接下来呢？



7号扇区

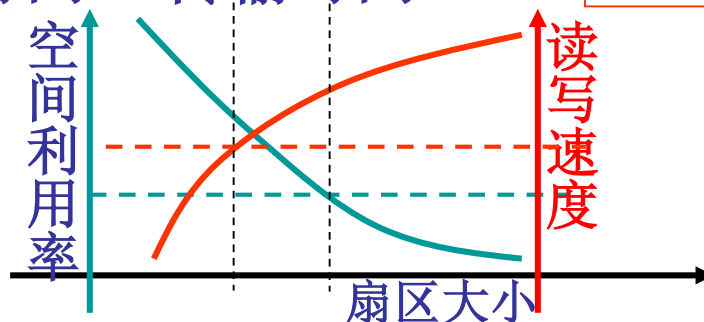
问题：C、H、S得到的扇区号是？ $C \times (\text{Heads} \times \text{Sectors}) + H \times \text{Sectors} + S$

磁盘访问时间 = 写入控制器时间 +
寻道时间 + 旋转时间 + 传输时间

共计约10ms

- 从扇区到盘块：每次读写1 K：
碎片0.5K；读写速度100K/秒；
每次读写1 M：碎片0.5M；
读写速度约40M/秒

扇区大小固定，但操作系统可以
每次读/写连续的几个扇区(盘块)



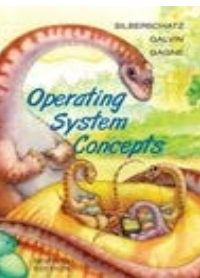
再接着使用磁盘：程序输出block

```
static void make_request()  
{ struct request *req;  
  req=request+NR_REQUEST;  
  req->sector=bh->b_blocknr<<1;  
  add_request(major+blk_dev,req); }
```

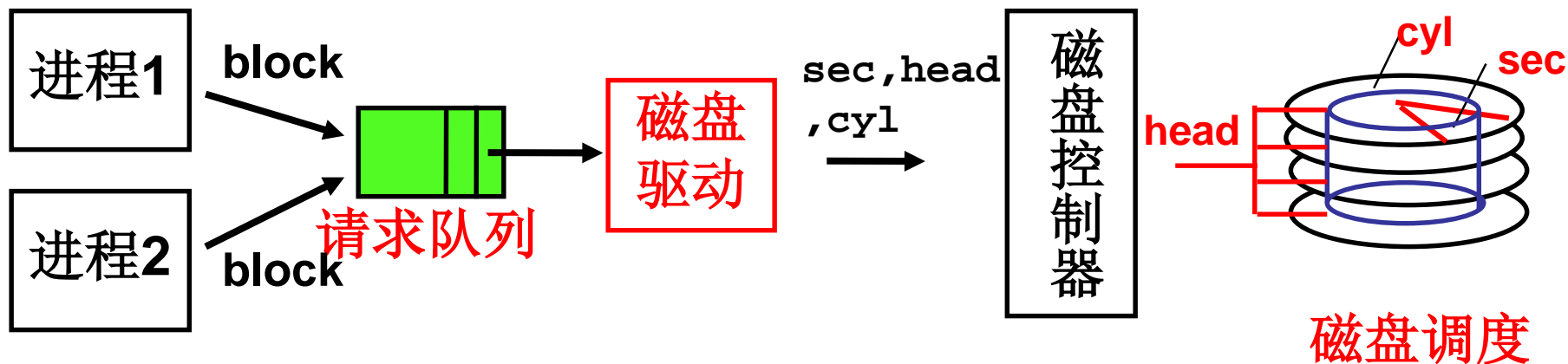
问题：Linux 0.11盘块多大？

$\text{block} = C \times (\text{Heads} \times \text{Sectors}) +$
 $H \times \text{Sectors} + S \rightarrow S = \text{block} \% \text{Sectors}$

```
void do_hd_request(void)  
{ unsigned int block=CURRENT->sector;  
  __asm__("divl %4":"=a"(block),"=d"(sec):"0"(block),  
          "1"(0),"r"(hd_info[dev].sect));  
  __asm__("divl %4":"=a"(cyl),"=d"(head):"0"(block),  
          "1"(0),"r"(hd_info[dev].head));  
  hd_out(dev,nsect,sec,head,cyl,WIN_WRITE,...);  
  ... }
```



多个进程通过队列使用磁盘(第二层抽象)

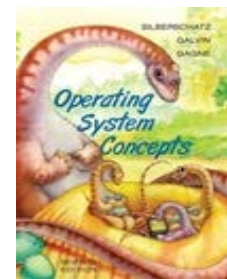


- 多个磁盘访问请求出现在请求队列怎么办？
- 调度的目标是什么？调度时主要考察什么？

目标当然是平均
访问延迟小！

寻道时间是主要
矛盾！

- 给调度算法，仍然从**FCFS**开始...



FCFS磁盘调度算法

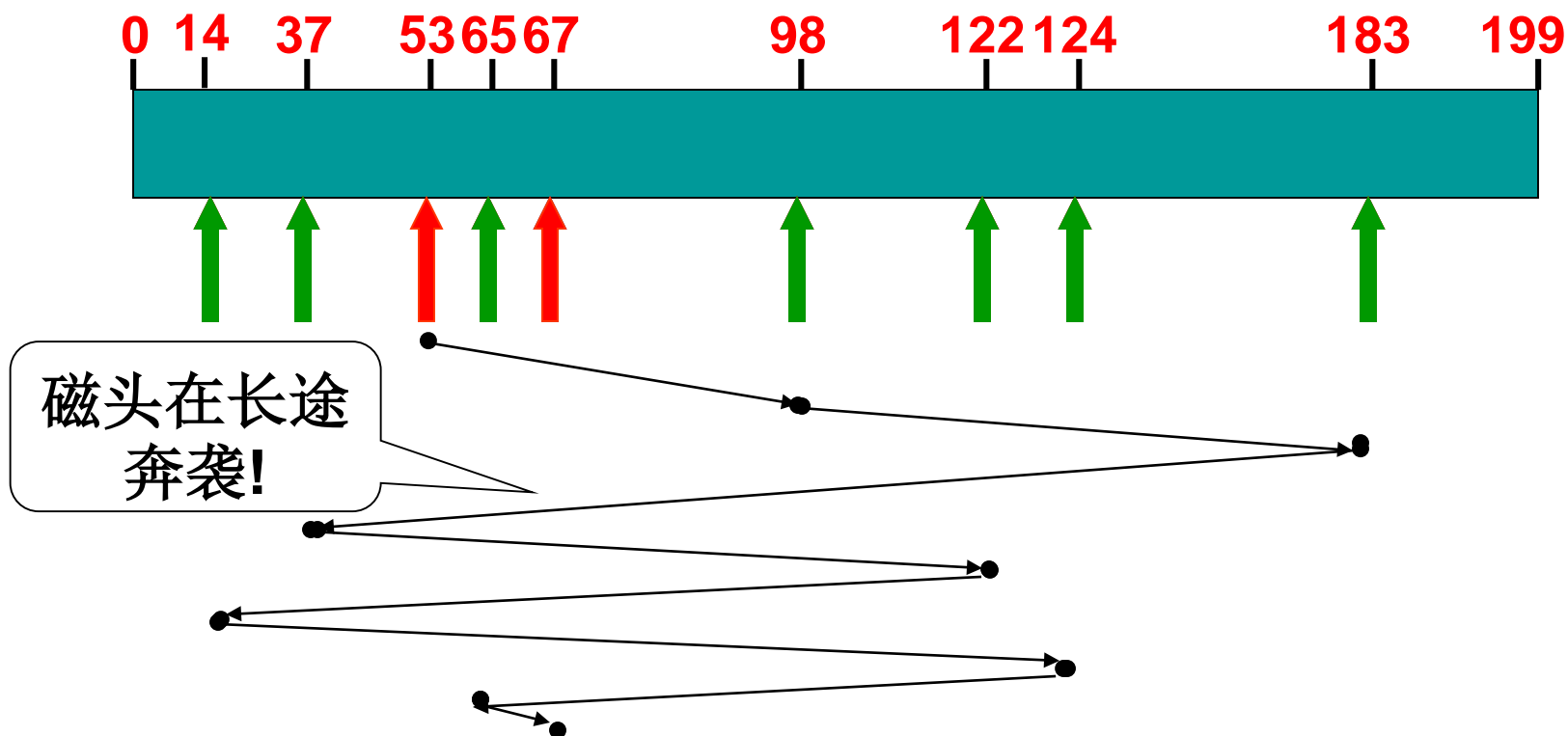
■ 最直观、最公平的调度:

■ 一个实例: 磁头开始位置=53;

请求队列=98, 183, 37, 122, 14, 124, 65, 67

FCFS: 磁头共
移动640磁道!

在移动过程中把经过
的请求处理了!

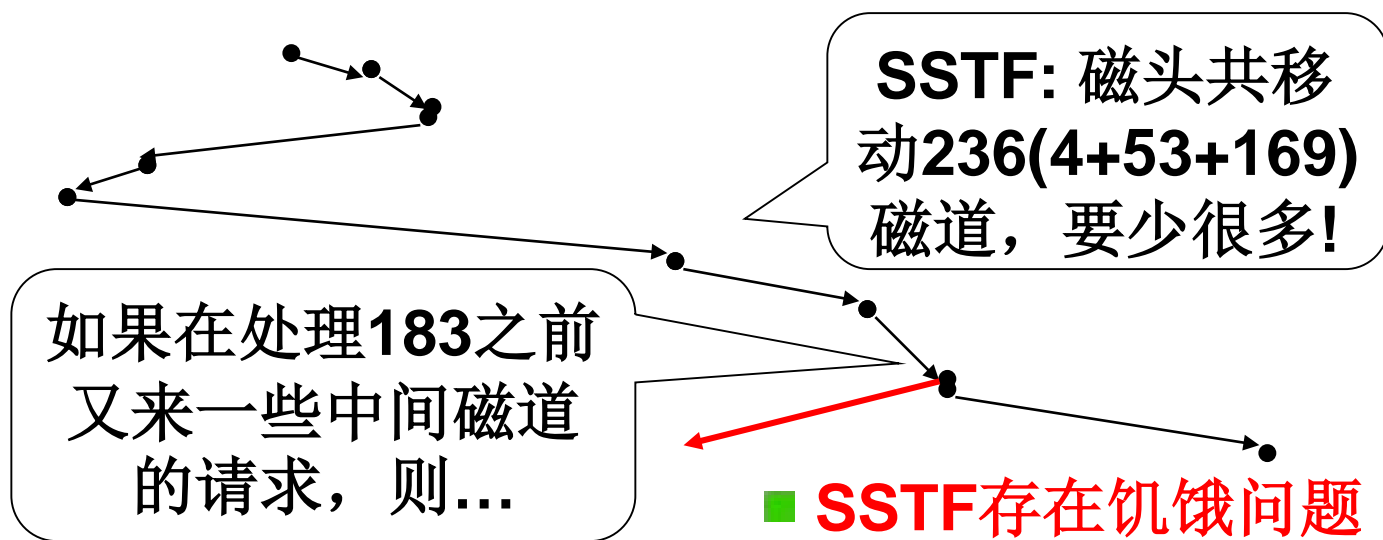
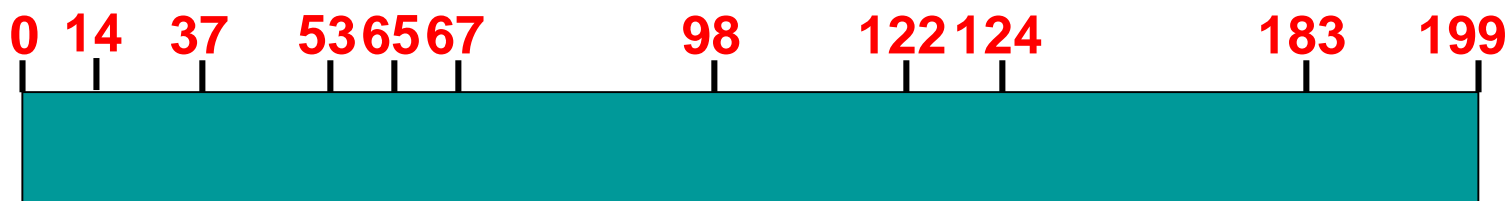


SSTF磁盘调度

■ Shortest-serve-time First:

■ 继续该实例: 磁头开始位置=53;

请求队列=98, 183, 37, 122, 14, 124, 65, 67

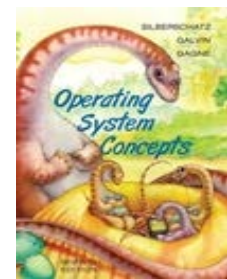
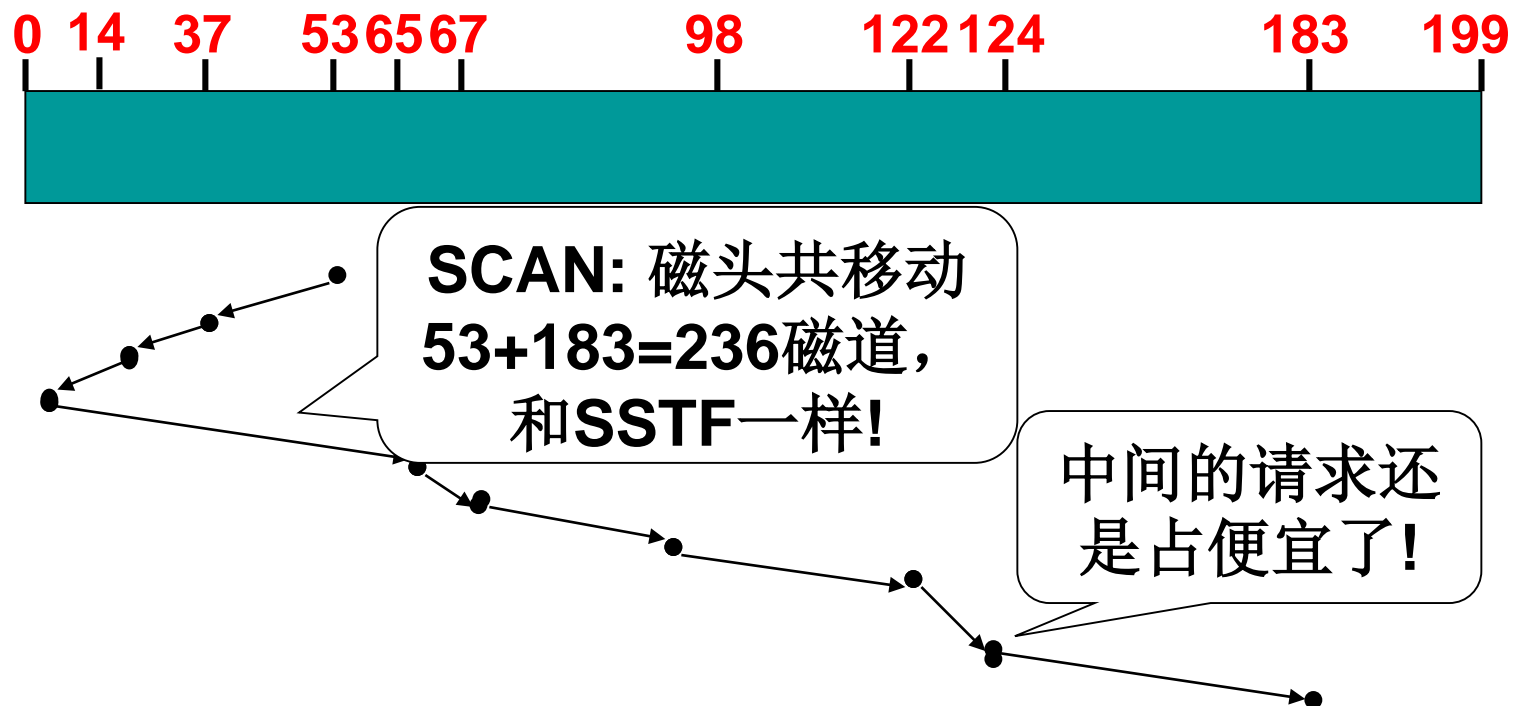


SCAN磁盘调度

■ SSTF+中途不回折：每个请求都有处理机会

■ 继续该实例：磁头开始位置=53；

请求队列=98, 183, 37, 122, 14, 124, 65, 67

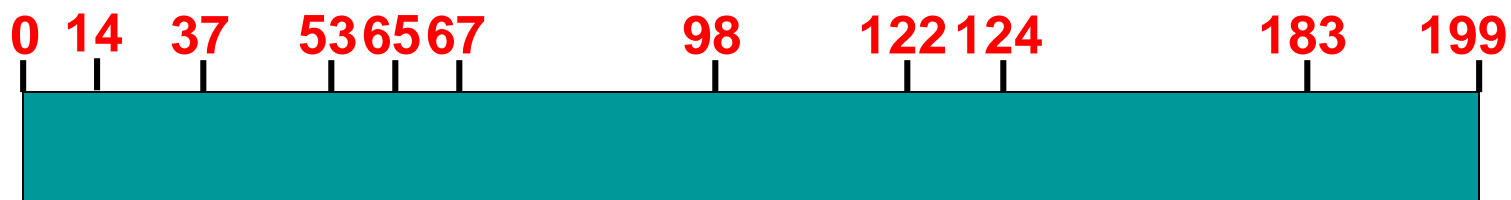


C-SCAN磁盘调度(电梯算法)

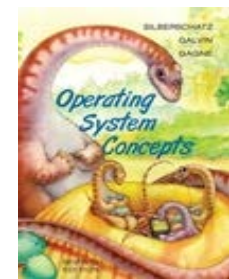
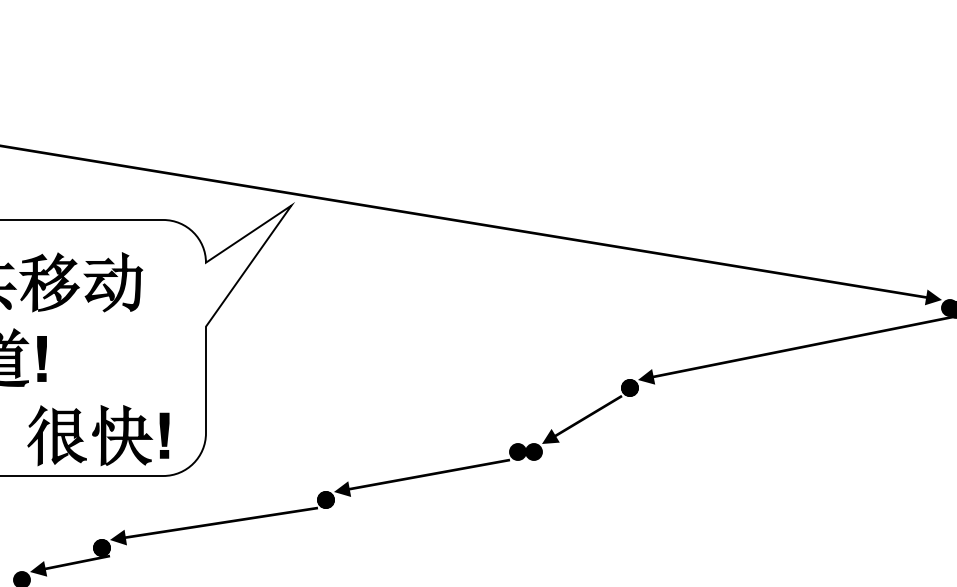
■ SCAN+直接移到另一端：两端请求都能很快处理

■ 继续该实例：磁头开始位置=53；

请求队列=98, 183, 37, 122, 14, 124, 65, 67



CSCAN: 磁头共移动
157+200磁道!
其中**200**是复位, 很快!

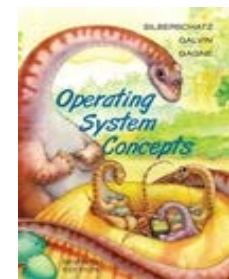


多个进程共同使用磁盘

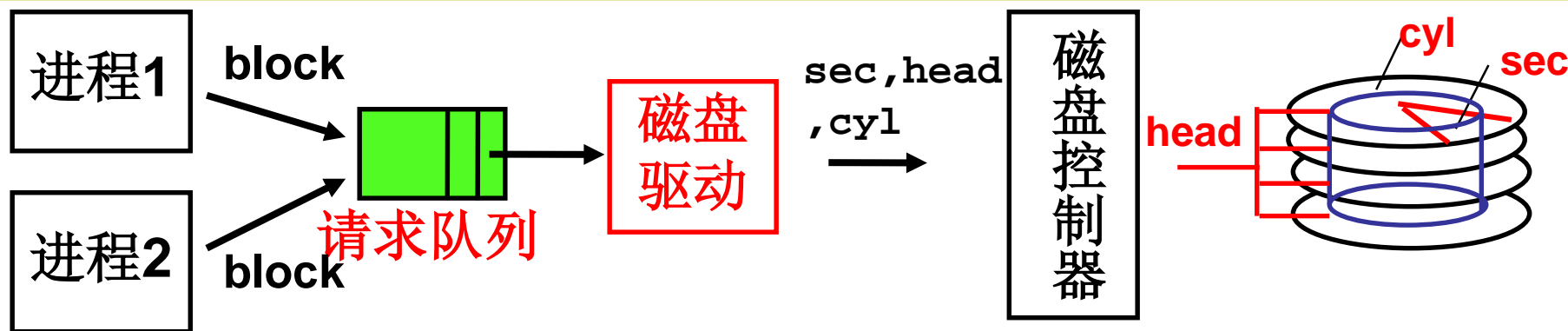
```
static void make_request()  
{  
    ...req->sector=bh->b_blocknr<<1;  
    add_request(major+blk_dev,req);  
}
```

```
static void add_request(struct blk_dev_struct *dev,  
struct request *req)  
{  
    struct request *tmp=dev->current_request;  
    req->next=NULL; cli(); //关中断(互斥)  
    for( ;tmp->next;tmp=tmp->next )  
        if( (IN_ORDER(tmp,req) || !IN_ORDER(tmp,tmp->next))  
            &&IN_ORDER(req,tmp->next) ) break;  
    req->next=tmp->next; tmp->next=req; sti();}
```

```
#define IN_ORDER(s1, s2) \    sector = Cx(Heads×Sectors) +  
                                H×Sectors + S  
((s1)->dev<(s2)->dev) || ((s1)->dev == (s2)->dev\  
    && (s1)->sector<(s2)->sector))
```



生磁盘(raw disk)的使用整理



- (1) 进程“得到盘块号”，算出扇区号(sector)
- (2) 用扇区号make req, 用电梯算法add_request

- (3) 进程sleep_on
 - (4) 磁盘中断处理
- ```
static void read_intr(void)
{
 end_request(1);
 do_hd_request(),
}
```
- 唤醒进程!

- (5) do\_hd\_request算出cyl,head,sector
- (6) hd\_out调用outp(...)完成端口写

