第11章 磁盘与文件

孙承杰

E-mail： sunchengjie@hit.edu.cn

哈工大计算学部人工智能教研室

2023年秋季学期

主要内容

11.1 磁盘结构

11.2 磁盘调度(IO子系统)

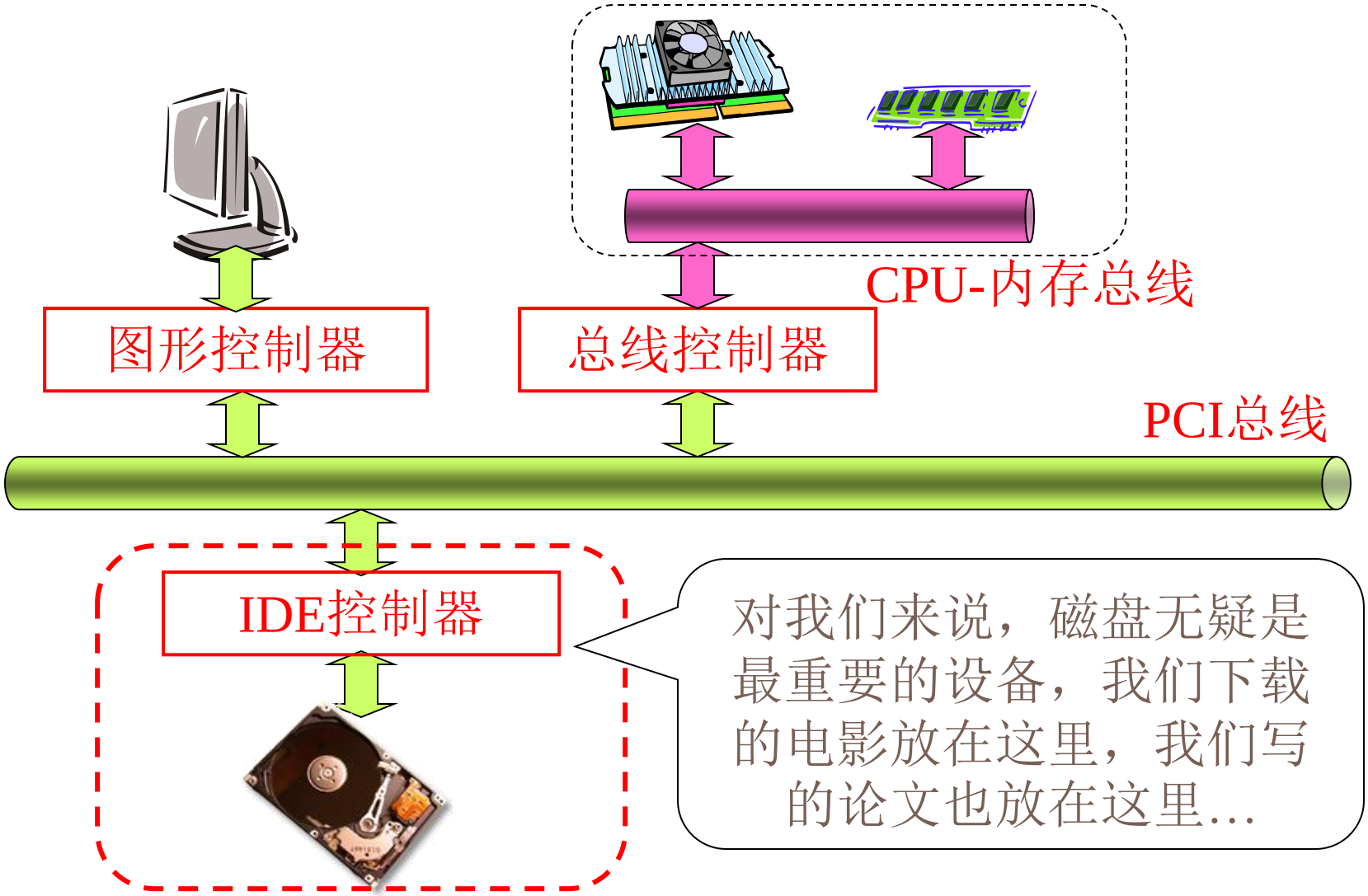
11.3 磁盘编址

11.4 磁盘分区

11.5 磁盘高速缓存与阵列

11.6 文件概念及实现方法

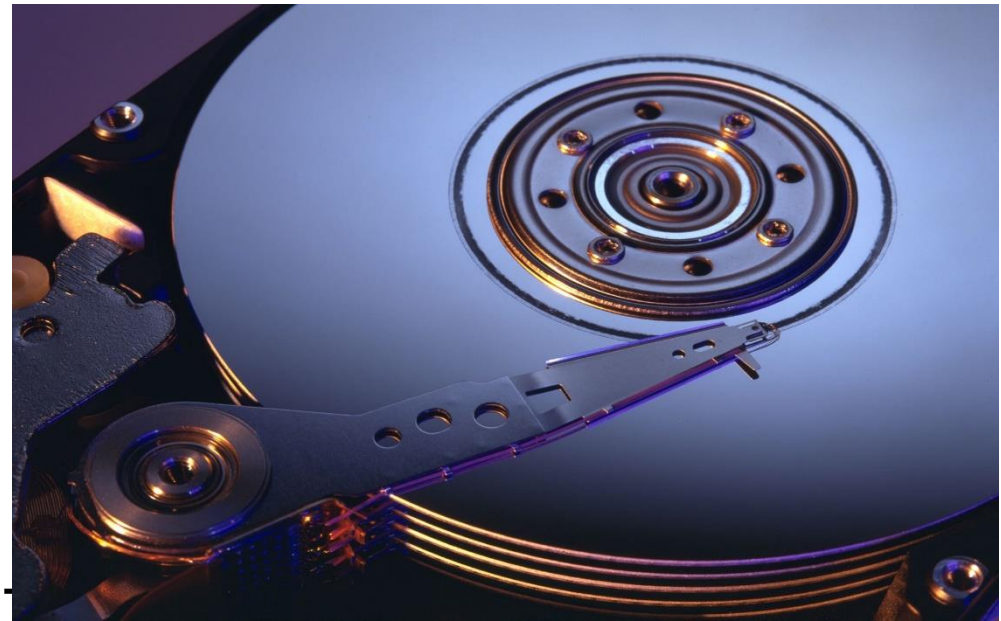
认识计算机外设与计算机!



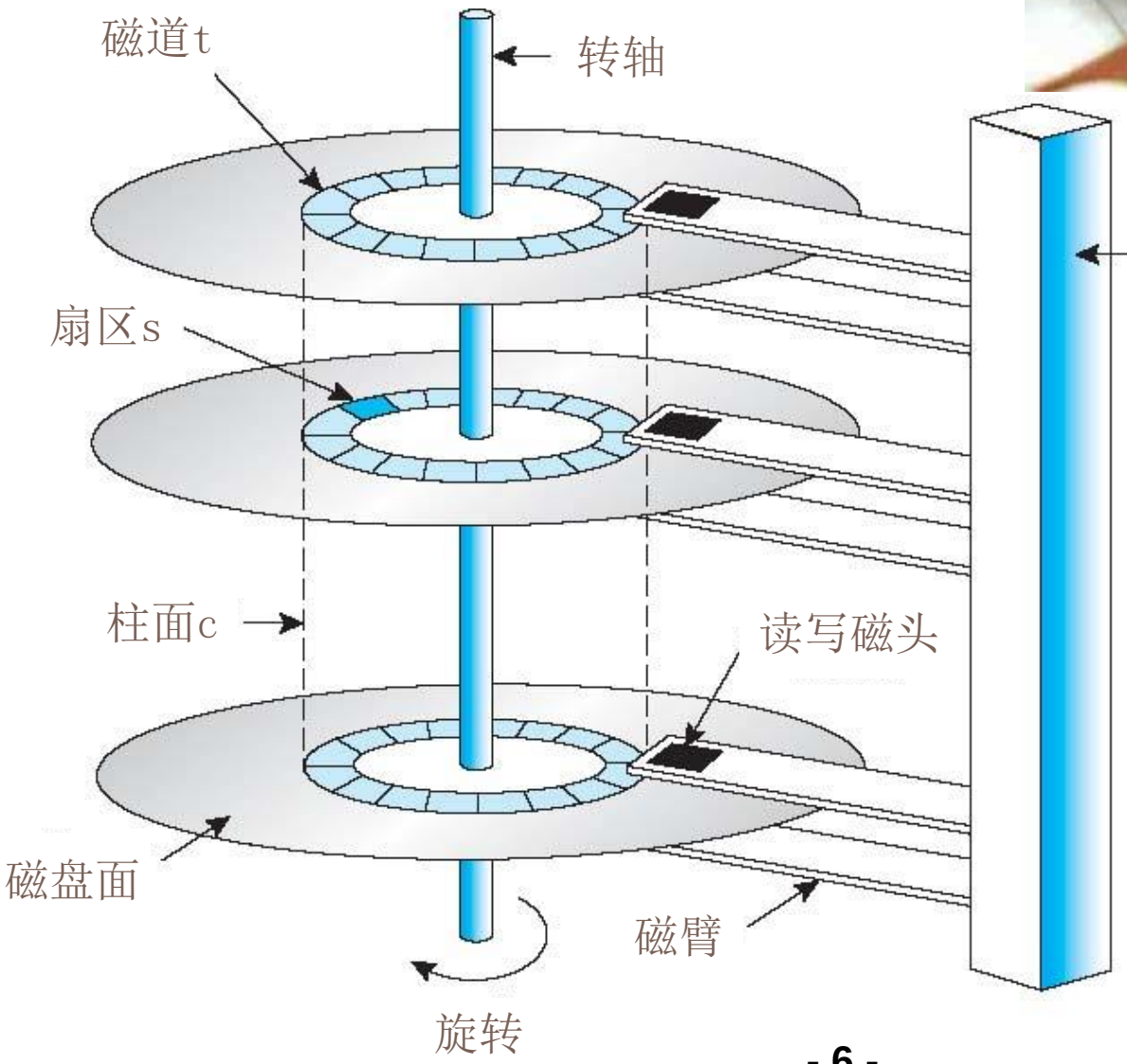
11.1 磁盘结构

首先需要了解磁盘!

认识一下磁盘



认识一下磁盘

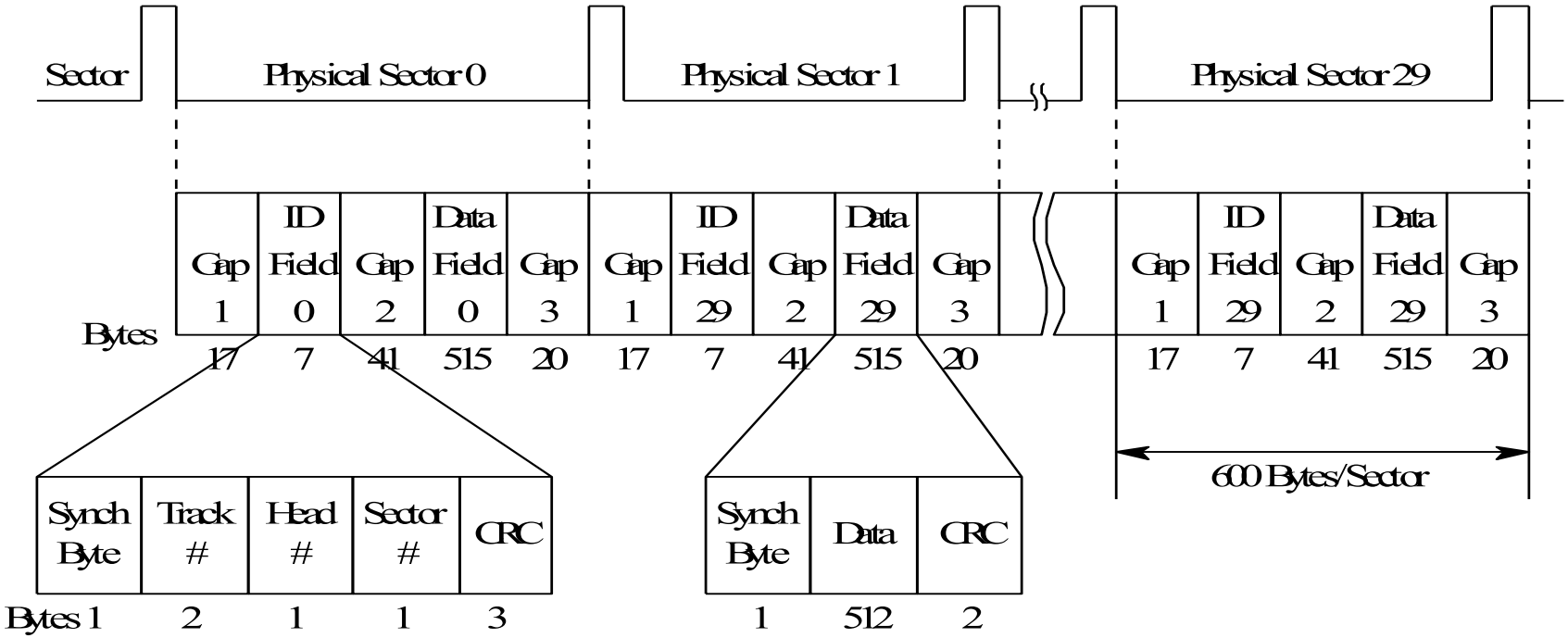


盘片高速旋转产生气流非常强，足以使磁头托起，并与盘面保持一个微小的距离。

现在的水平已经达到 0.005μm～0.01μm，这只是人类头发直径的千分之一。

认识一下磁盘

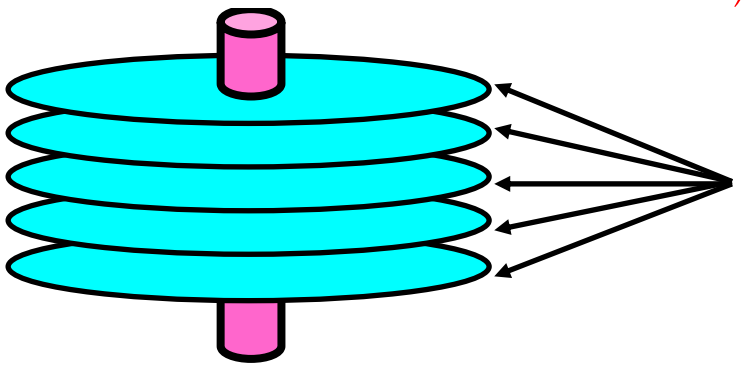
- 数据的格式
  - **Synch：** 具有特定的位图像，作为该字段的分界符；
  - **Track：** 磁道号　　**Head：** 磁头号
  - **Sector：** 扇区号　　**CRC：** 段校验



磁盘的格式化

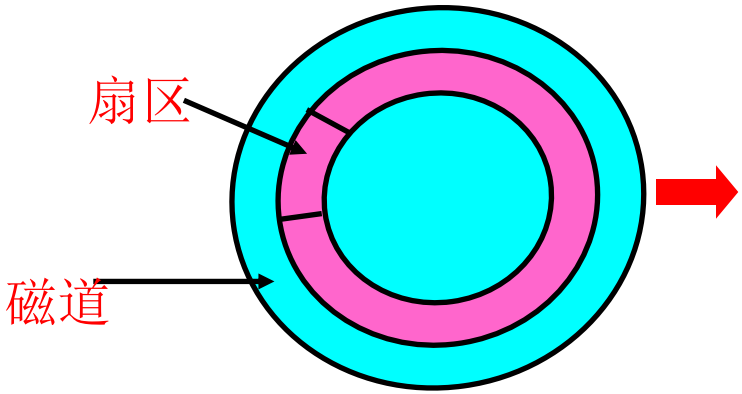
认识一下磁盘

- 画一个示意图:



- 所以，磁盘被称为块设备!

- 看看俯视图:

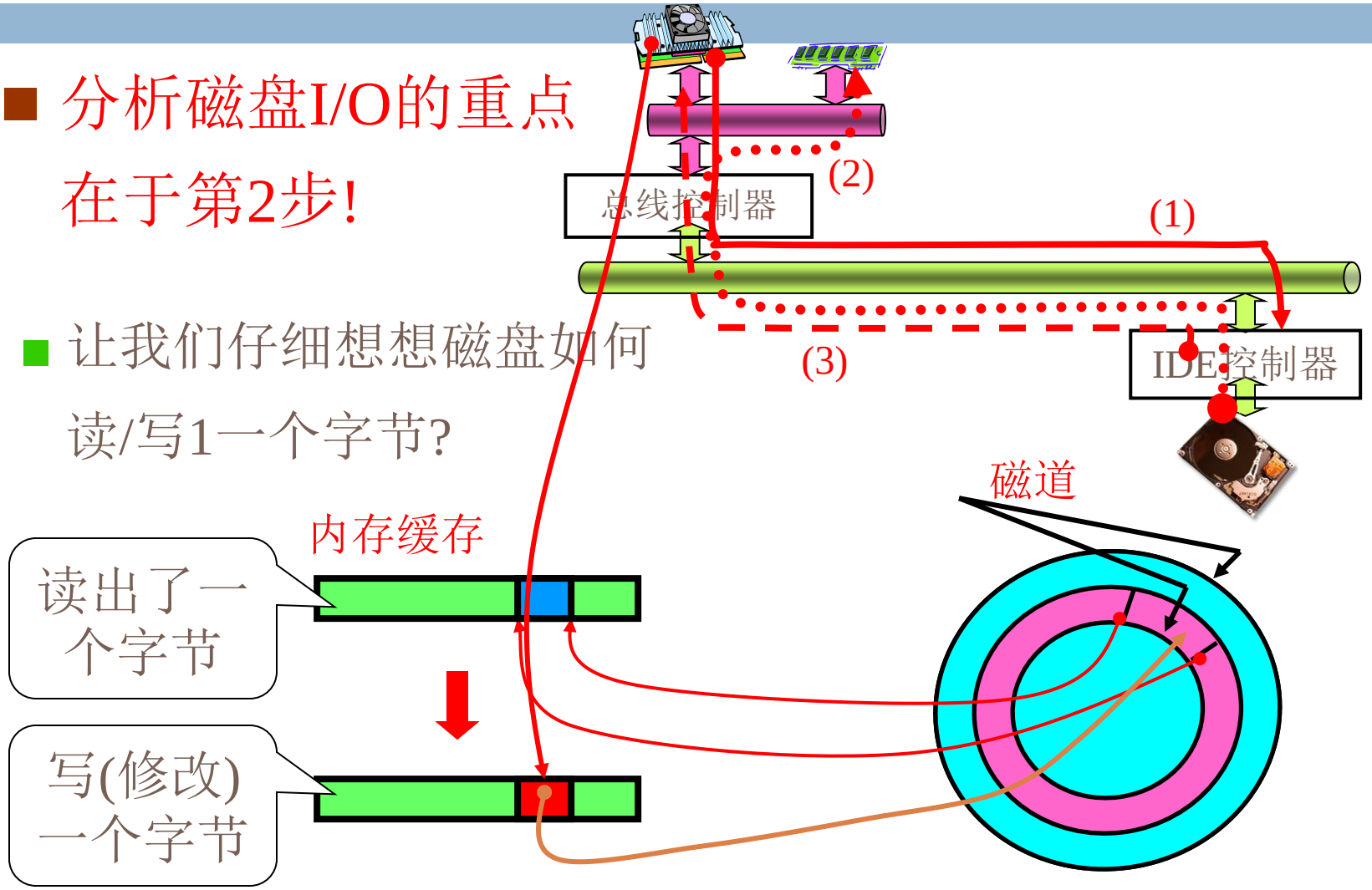


扇区是磁盘的寻址单位、访问单位

- 磁盘的数据单位是扇区
- 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

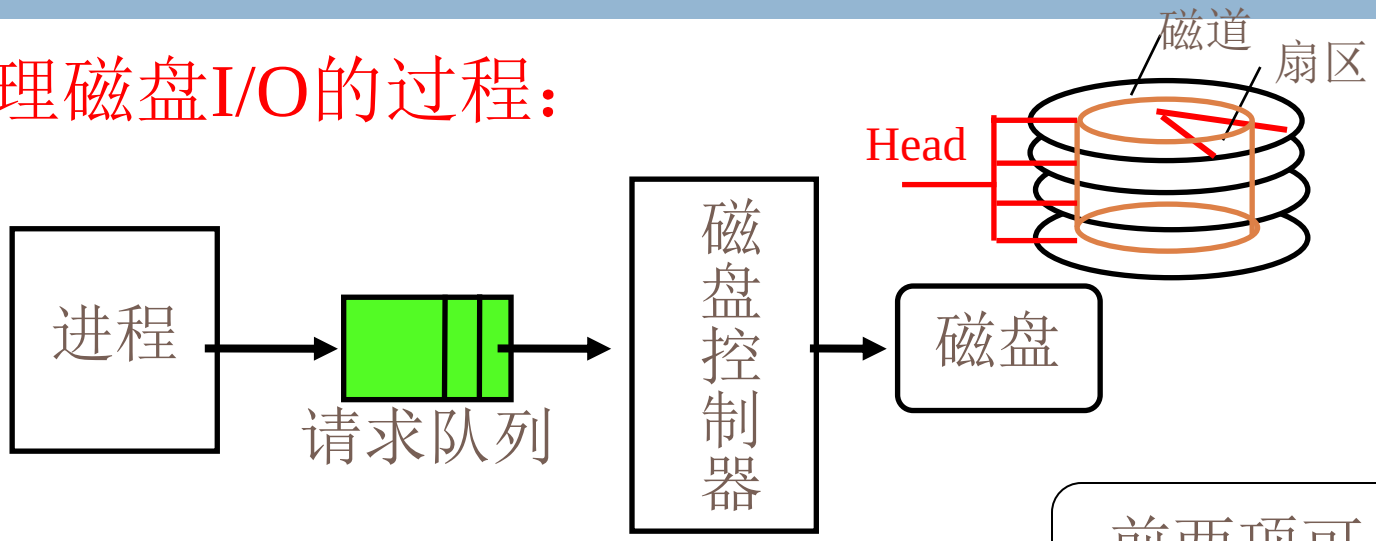
- 分析磁盘I/O的重点在于第2步!
  - 让我们仔细想想磁盘如何读/写1一个字节?



- 磁盘I/O: 缓存队列 → 控制器 → 寻道 → 旋转 → 传输!

磁盘I/O的分析

- 整理磁盘I/O的过程：



- 我们最关心的磁盘什么时候读/写完?

前两项可以忽略!

磁盘访问延迟 = 队列时间 + 控制器时间 + 寻道时间 + 旋转时间 + 传输时间

12 ms to 8 ms

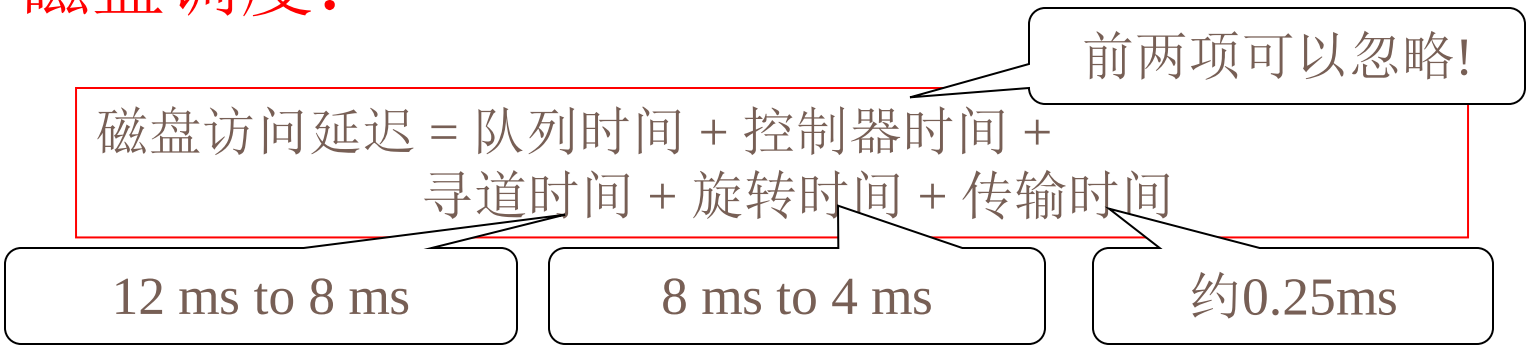
(半周): 8 ms to 4 ms

约0.25ms

- 关键所在: 最小化寻道时间和旋转延迟!

I/O过程是解开许多磁盘问题的钥匙

- 磁盘调度：



  - 多个磁盘访问请求出现在请求队列怎么办？ 调度
  - 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!
一直高速旋转无法控制

  - 磁盘调度：输入多个磁道请求，给出服务顺序!

11.2 磁盘调度

磁盘读写请求频繁发生，
如何尽快响应？

如何优化磁盘读写时间：寻道、旋转两个方面

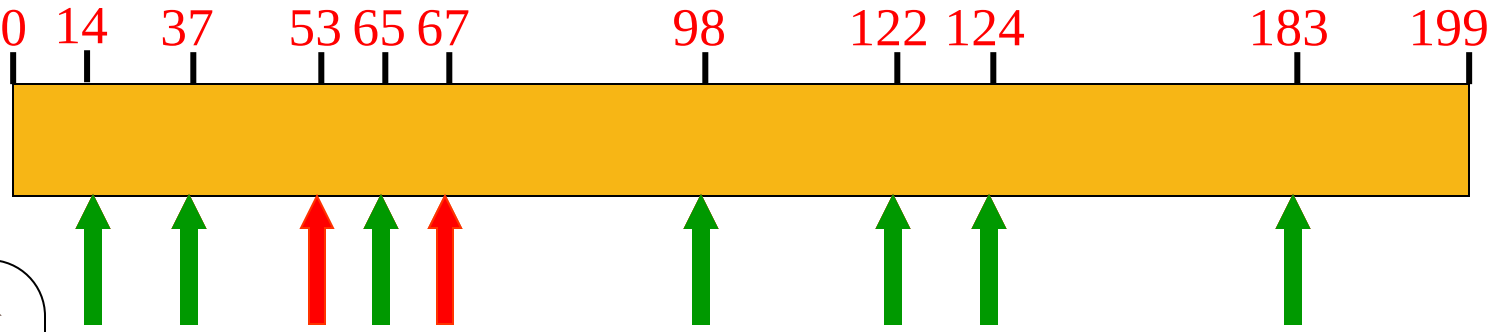
FCFS磁盘调度

130+146+85+108+110+59+2=640

- 最直观、最公平的调度：
  - 一个实例: 磁头开始磁道位置=53;

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

FCFS: 磁头共移动640磁道!



183-53=130

122-37=85

183-37=146

124-14=110

122-14=108

124-65=59

67-65=2

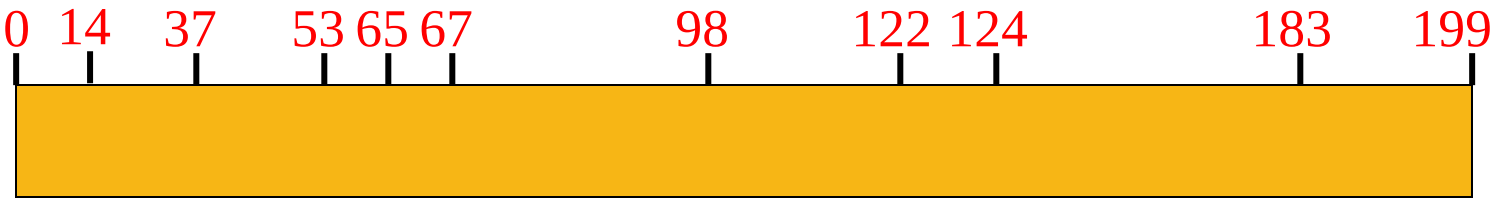
在移动过程中把经过的请求处理了?!

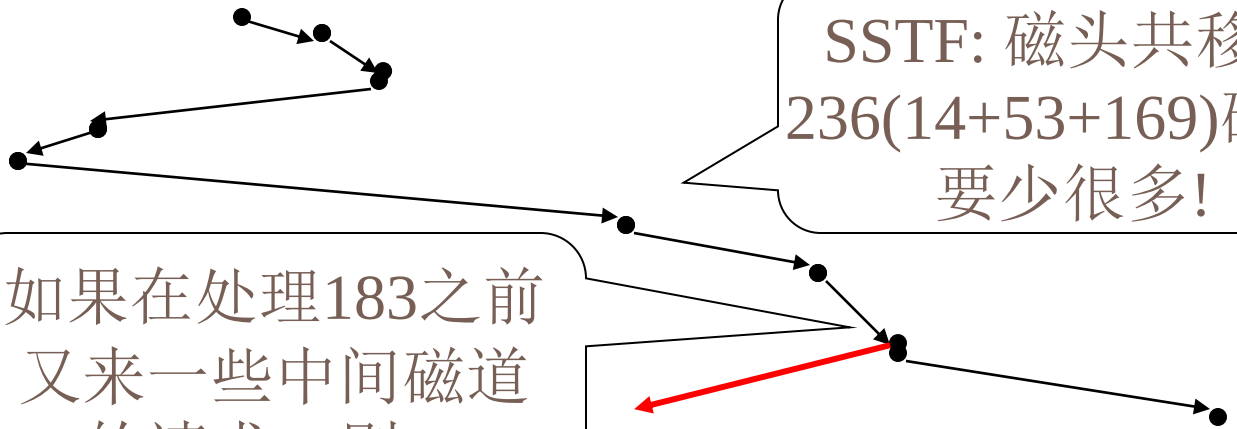
磁头在长途奔袭!

SSTF磁盘调度

- Shortest-seek-time First最短寻道时间优先：
  - 继续该实例: 磁头开始位置=53；

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





SSTF: 磁头共移动236(14+53+169)磁道，要少很多!

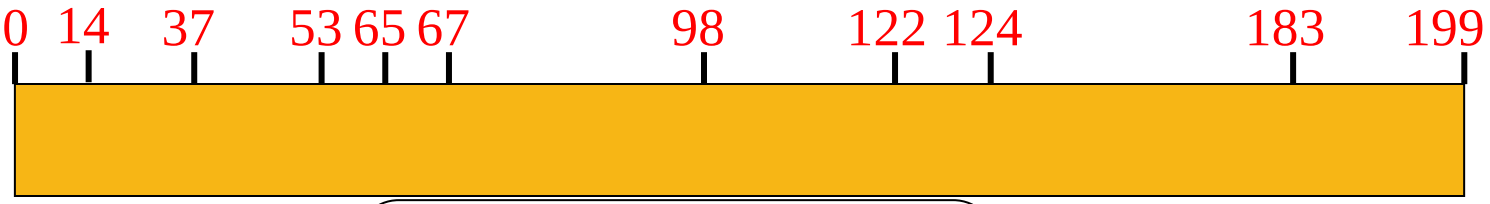
如果在处理183之前又来一些中间磁道的请求，则…

  - SSTF存在饥饿问题

SCAN磁盘调度(扫描/电梯算法)

- 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

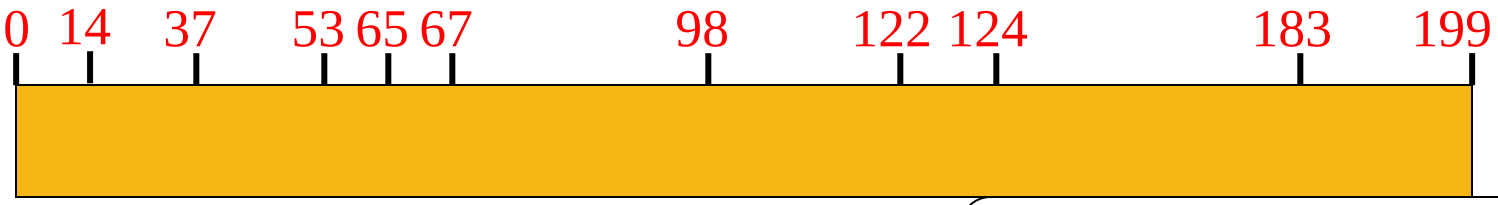


  - SCAN导致延迟不均

C-SCAN磁盘调度

- 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中的Circular
是环的意思!

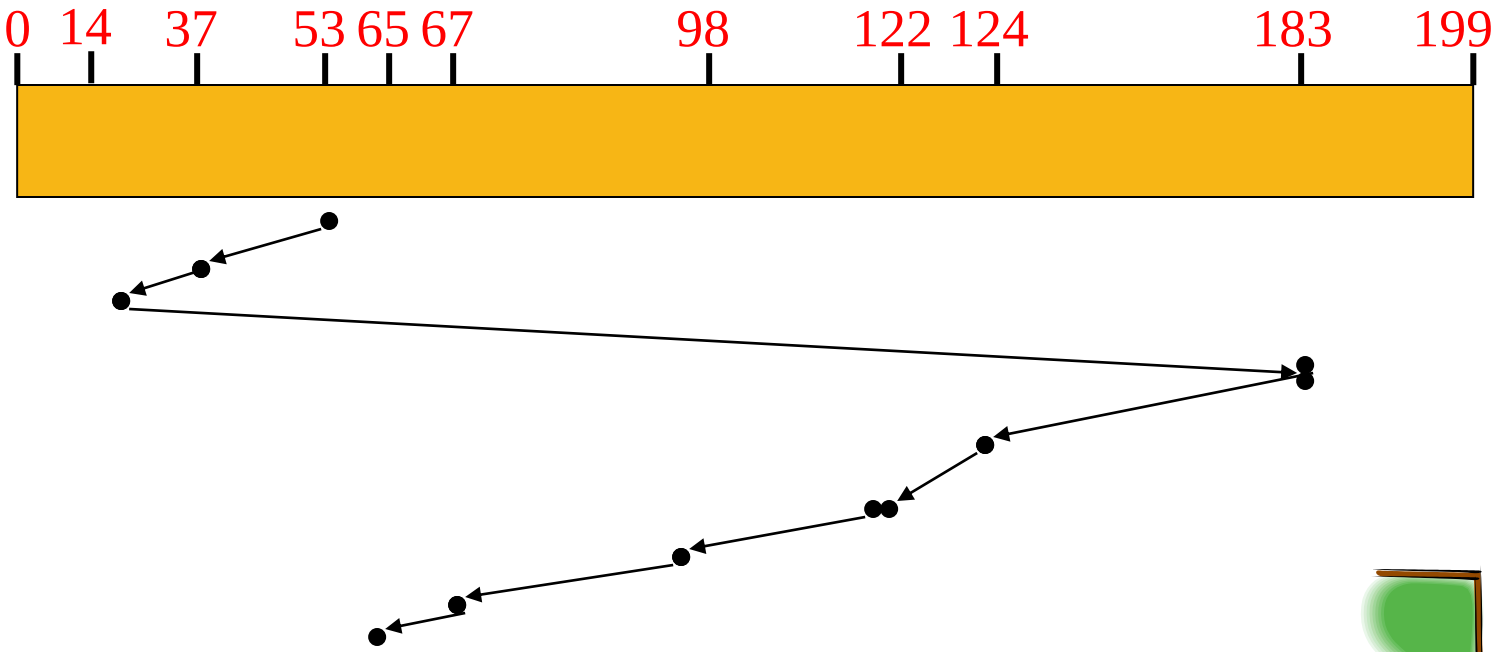
CSCAN: 磁头共移动
53+199+134磁道!
其中199会较快!

- 14→0(183→199)
  没有必要

C-LOOK磁盘调度

- CSCAN+看一看：前面没有请求就回移
  - 继续该实例: 磁头开始位置=53；

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



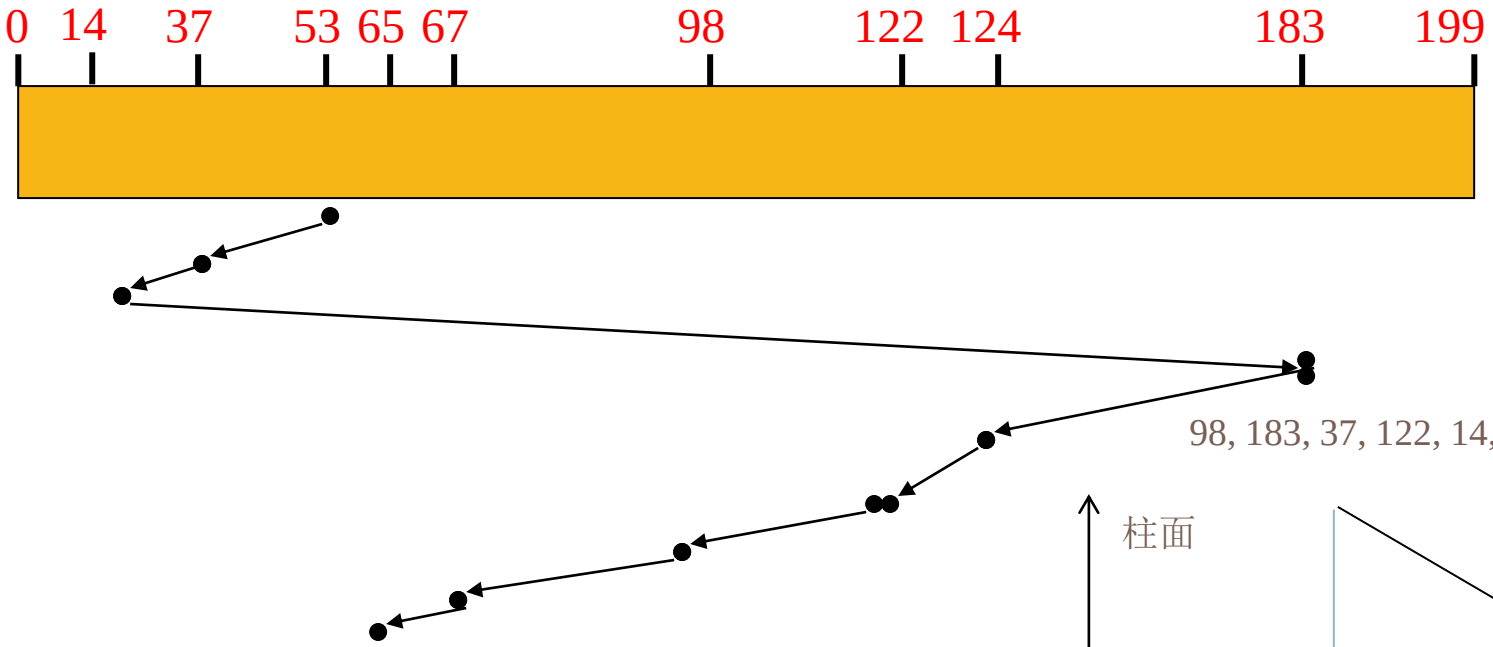
  - C-LOOK是比较合理的缺省算法

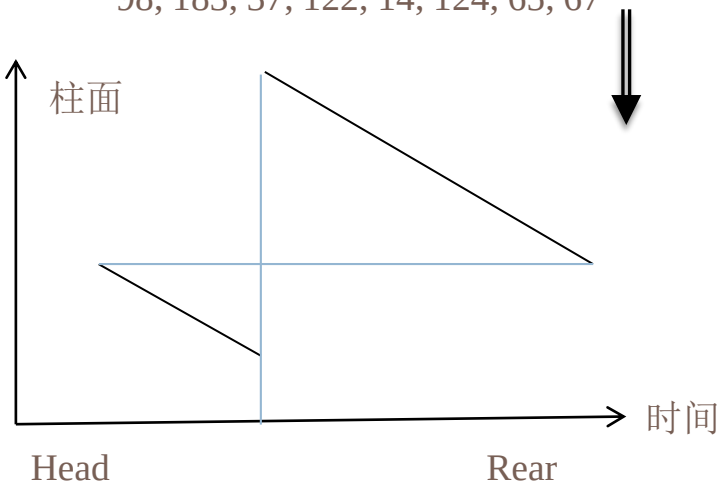
操作系统中所有的算法都要因地制宜!

C-LOOK磁盘调度

继续该实例: 磁头开始位置=53;

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





1）磁道请求队列的的形式

2）新磁道请求如何入队列

C[i+1]<X<c[i]或者X>C[i+1]>c[i]

53-37-14- 183-124-122-98-67-65

11.3 磁盘编址

如何管理磁盘，
首先对磁盘的扇区进行编号！

- 出厂的磁盘需要低级格式化(物理格式化)：

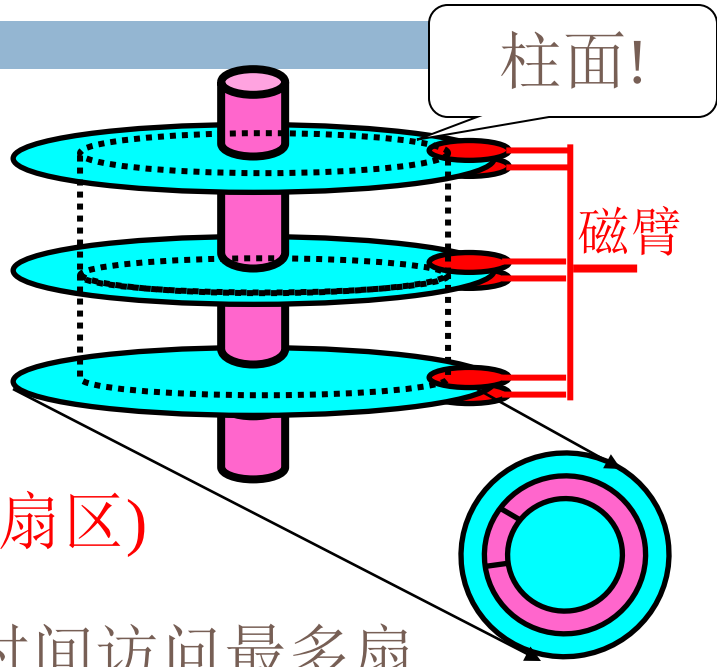
  将连续的磁性记录材料分成物理扇区

- 扇区 = 头 + 数据区 + 尾
- 头、尾中包含只有磁盘控制器能识别的 号码和纠错码等信息

什么是磁盘的逻辑格式化？
第12章 文件系统！

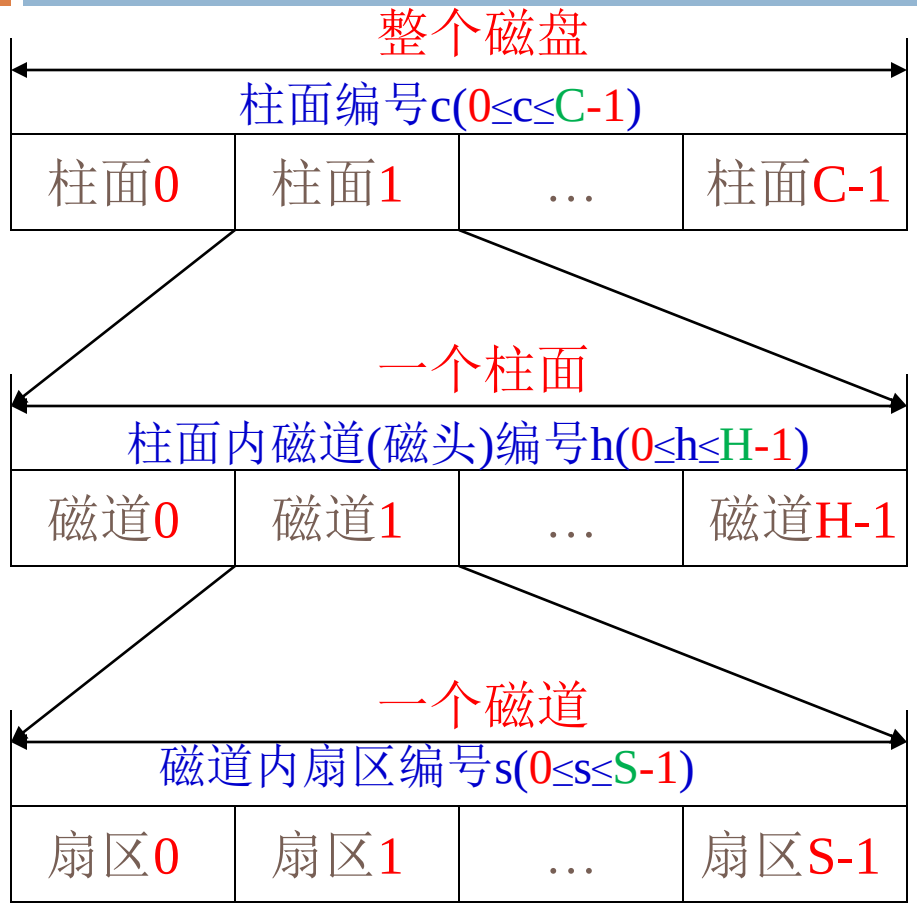
I/O过程是解开许多磁盘问题的钥匙

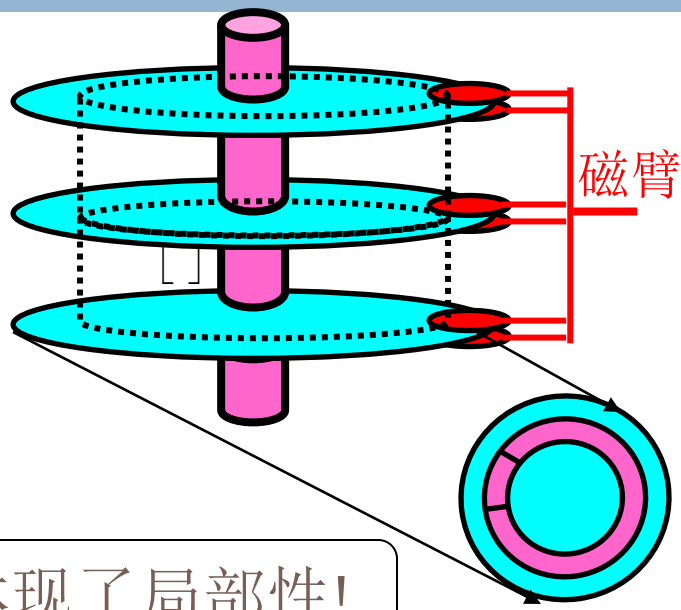
- 磁盘寻址：对于内存，我们往往更关心存放内容的地址
  - 实际上就是扇区怎么编址?
  - 显然这个地址是(盘面 + 磁道 + 扇区)
  - 寻道和旋转费时多 ⇒ 花最少时间访问最多扇区的方案: 磁臂不动、磁盘旋转一周，访问磁头遇到的所有扇区。
  - 扇区编址(1): CHS(Cylinder/Head/Sector)
  - 扇区编址(2): 扇区编号（Logical Block Addressing LBA)



让这些扇区的编址邻近:
因为局部性!

扇区编号—现代磁盘的常见寻址方式





体现了局部性!

- 扇区编号，按照(C,H,S)将扇区形成一维扇区数组，数组索引就是扇区编号

某扇区(c,h,s)编号A = c*H*S + h*S + s　扇区总数 = C*H*S

已知A，则 s = A%S；h = [A/S]%H；c = [A/(H*S)]

扇区编号—现代磁盘的常见寻址方式

- chs(Cylinder/Head/Sector)模式
  - 以前, 硬盘的容量还非常小, 采用与软盘类似的结构生产硬盘
    - 硬盘盘片的每一条磁道都具有相同的扇区数
  - 由此产生了所谓的3D参数 (Disk Geometry)：
    - 柱面数(Cylinders),磁头数(Heads), 扇区数(Sectors per track, 以及相应的寻址方式。

扇区编号—现代磁盘的常见寻址方式

- chs(Cylinder/Head/Sector)模式
  - 磁头数(Heads) 表示硬盘总共有几个磁头,也就是有几面盘片, 最大为 256 (用 8 个二进制位存储);
  - 柱面数(Cylinders) 表示硬盘每一面盘片上有几条磁道,最大为 1024(用 10 个二进制位存储);
  - 扇区数(Sectors per track) 表示每一条磁道上有几个扇区, 最大为63 (用 6 个二进制位存储).
  - 每个扇区一般是 512个字节;
  - 所以采用chs模式的磁盘最大容量为:
    - 256 * 1024 * 63 * 512 / 1048576 = 8064 MB

扇区编号—现代磁盘的常见寻址方式

- chs(Cylinder/Head/Sector)模式
  - 这种方式会浪费很多磁盘空间 (与软盘一样)
  - 为了进一步提高硬盘容量, 产生了等密度结构硬盘，外圈磁道的扇区比内圈磁道多。
  - 采用这种结构后, 硬盘不再具有实际的3D参数, 寻址方式也改为LBA (logical Block Adress)寻址, 即以扇区为单位进行寻址
  - 为了与使用chs寻址的兼容 (如使用BIOS Int13H接口的软件), 在硬盘控制器内部安装了一个地址翻译器, 由它负责将老式3D参数翻译成新的线性参数。

IDE硬盘控制器的寄存器

- 有一组命令寄存器组(Task File Registers)，I/O的端口地址为1F0H~1F7H
  - 1F2H 扇区计数寄存器
    1F3H 扇区号，或LBA块地址0~7
    1F4H 柱面数低8位，或LBA块地址8~15
    1F5H 柱面数高8位，或LBA块地址16~23
    1F6H 驱动器/磁头，或LBA块地址24~27
    1F7H 状态寄存器 命令寄存器
  - CHS或LBA在磁头寄存器中指定

想一想……磁盘驱动应如何实现?

```
do_hd = intr_addr;                          // do_hd 函数会在中断程序中被调用。
outb_p(hd_info[drive].ctl,HD_CMD);          // 向控制寄存器输出控制字节。
port=HD_DATA;                               // 置 dx 为数据寄存器端口(0x1f0)。
outb_p(hd_info[drive].wpcom>>2,++port);     // 参数：写预补偿柱面号(需除 4)。
outb_p(nsect,++port);                       // 参数：读/写扇区总数。
outb_p(sect,++port);                        // 参数：起始扇区。
outb_p(cyl,++port);                         // 参数：柱面号低 8 位。
outb_p(cyl>>8,++port);                      // 参数：柱面号高 8 位。
outb_p(0xA0|(drive<<4)|head,++port);        // 参数：驱动器号+磁头号。
outb(cmd,++port);                           // 命令：硬盘控制命令。
```

Linux 0.11下实现磁盘读写驱动片段

磁盘速度与内存速度的差异

1)磁盘往往不是严格按需读取，而是每次都会预读，即使只需要一个字节，磁盘也会从这个位置开始，顺序向后读取一定扇区长度的数据放入内存。
2)这样做的理论依据是计算机科学中著名的局部性原理：当一个数据被用到时，其附近的数据也通常会马上被使用。

回忆：虚拟内存中程序优化

虚拟内存：按需调页与页面置换。

如何优化提升程序的性能？经常访问的放在一起，被唤出的概率降低，从磁盘读取速度快。

- 对代码来说，紧凑的代码也往往意味着接下来执行的代码更大可能就在相同的页或相邻页。根据时间locality特性，程序90%的时间花在了10%的代码上。如果将这10%的代码尽量紧凑且排在一起，被换出的概率降低，从磁盘读取速度快。

- 对数据来说，尽量将那些会一起访问的数据放在一起。这样当访问这些数据时，因为它们在同一页或相邻页，只需要一次调页操作即可完成；反之，如果这些数据分散在多个页（更糟的情况是这些页还不相邻），那么每次对这些数据的整体访问都会引发大量的缺页错误，从而降低性能。

进程I/O整个过程贯穿

获得编号是使用磁盘的关键!

- 第1步：得到要访问的扇区编号；
  得到读盘的目标(或写盘的源)内存地址

算法输入!

- 第2步：将扇区编号和内存地址写给DMA；
  然后阻塞进程

查手册、写端口!

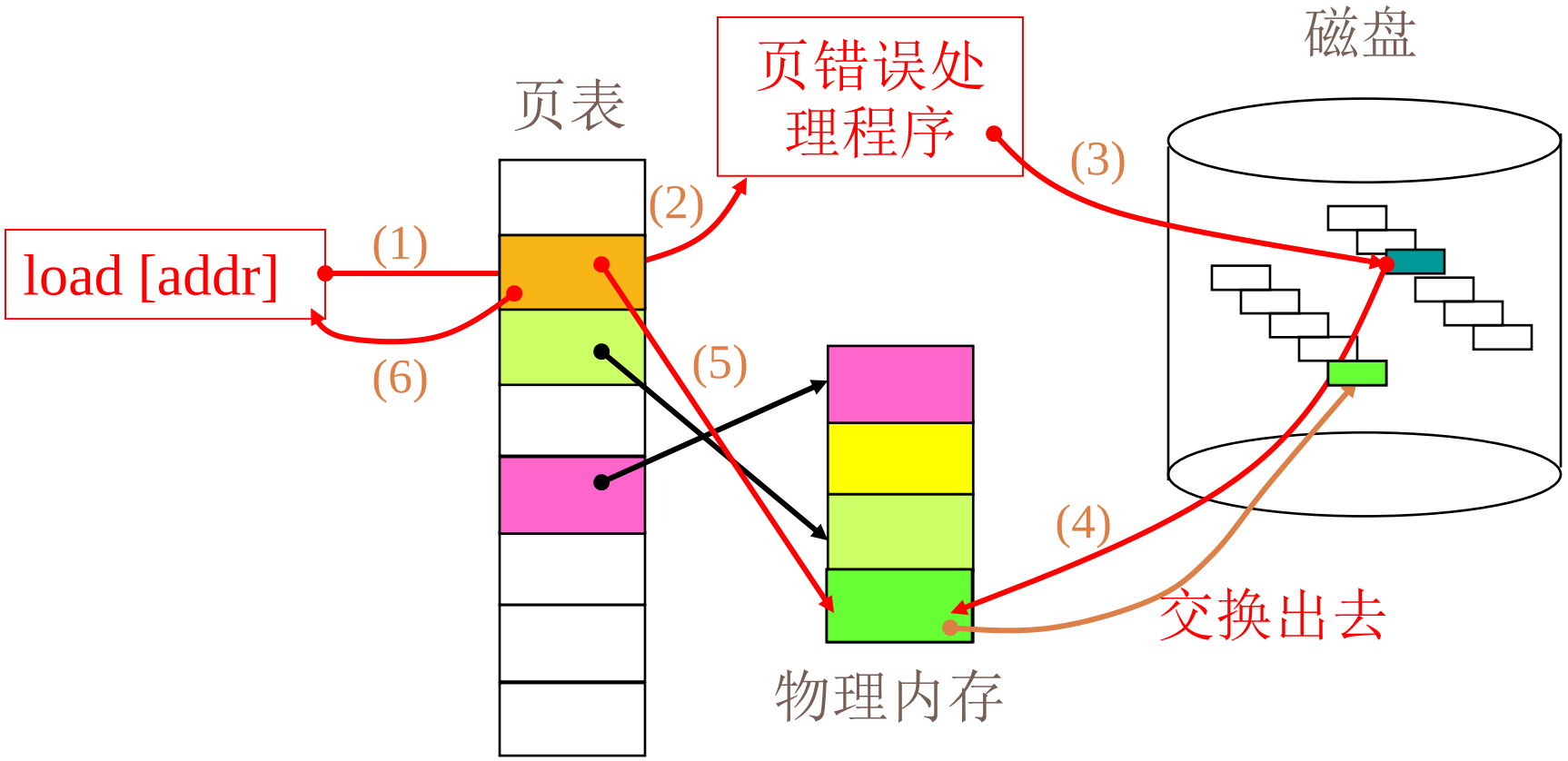
- 第3步：DMA处理完成后中断CPU；中断处理程序唤醒阻塞进程

编写中断处理程序!

- 第4步：进程继续…

页面置换（Swap）？

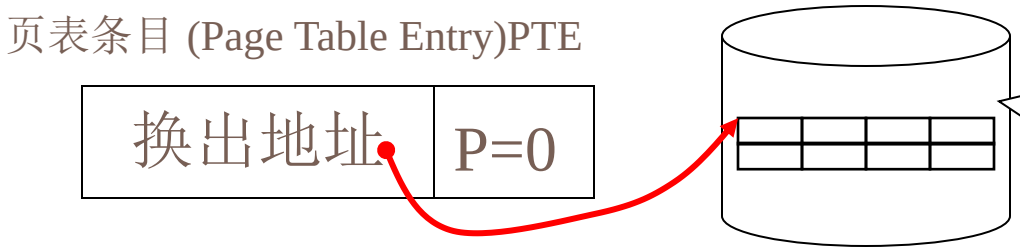
请求调页—页面置换



- 交换出去的页面放在哪里?

交换分区

- 交换出去的页面显然要写到磁盘上
  - 问题的关键是写到磁盘的什么位置?
  - 如果是代码段和数据段，直接写到文件中？
  - 如果是堆栈段呢? 创建一个文件吗?
  - 应该是直接“页面 ↔ 扇区”变成了页面 ↔ 文件 ↔ 扇区映射关系，显然是低效的

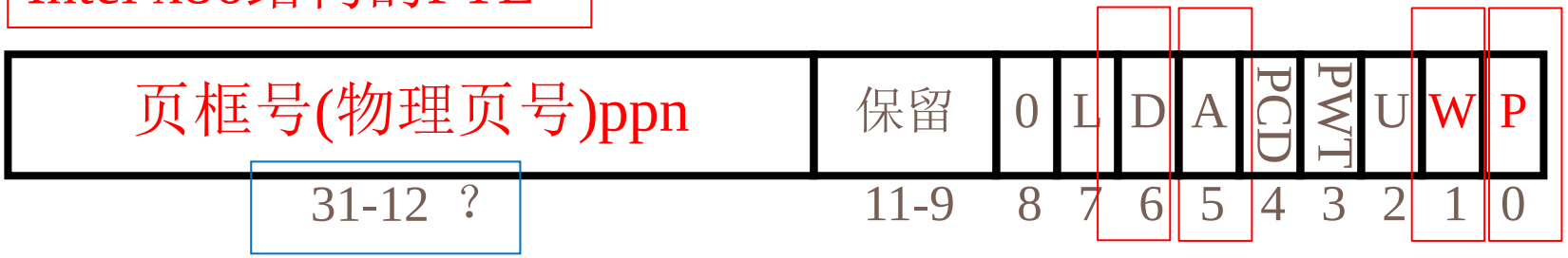


为提高效率，这部分磁盘不存文件，直接用扇区号寻址。(交换分区)

  - 这样使用的磁盘称为生磁盘(raw disk)

回忆：Intel x86的分页硬件

Intel x86结构的PTE



P--位0是存在（Present）标志

A--位5是已访问（Accessed）标志。

D--位6是页面已被修改（Dirty）标志。

R/W--位1是读/写（Read/Write）标志。

换出到SWAP分区后地址存到哪里？ 可以存到页框号的位置

Linux交换分区

- 安装Linux时，需创建一硬盘分区作为交换分区

```
cst:/dev# fdisk -l

Disk /dev/sda: 73.4 GB, 73407820800 bytes
255 heads, 63 sectors/track, 8924 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1        8594    69031273+  83  Linux
/dev/sda2            8595        8924     2650725    5  Extended
/dev/sda5            8595        8924     2650693+  82  Linux swap
```

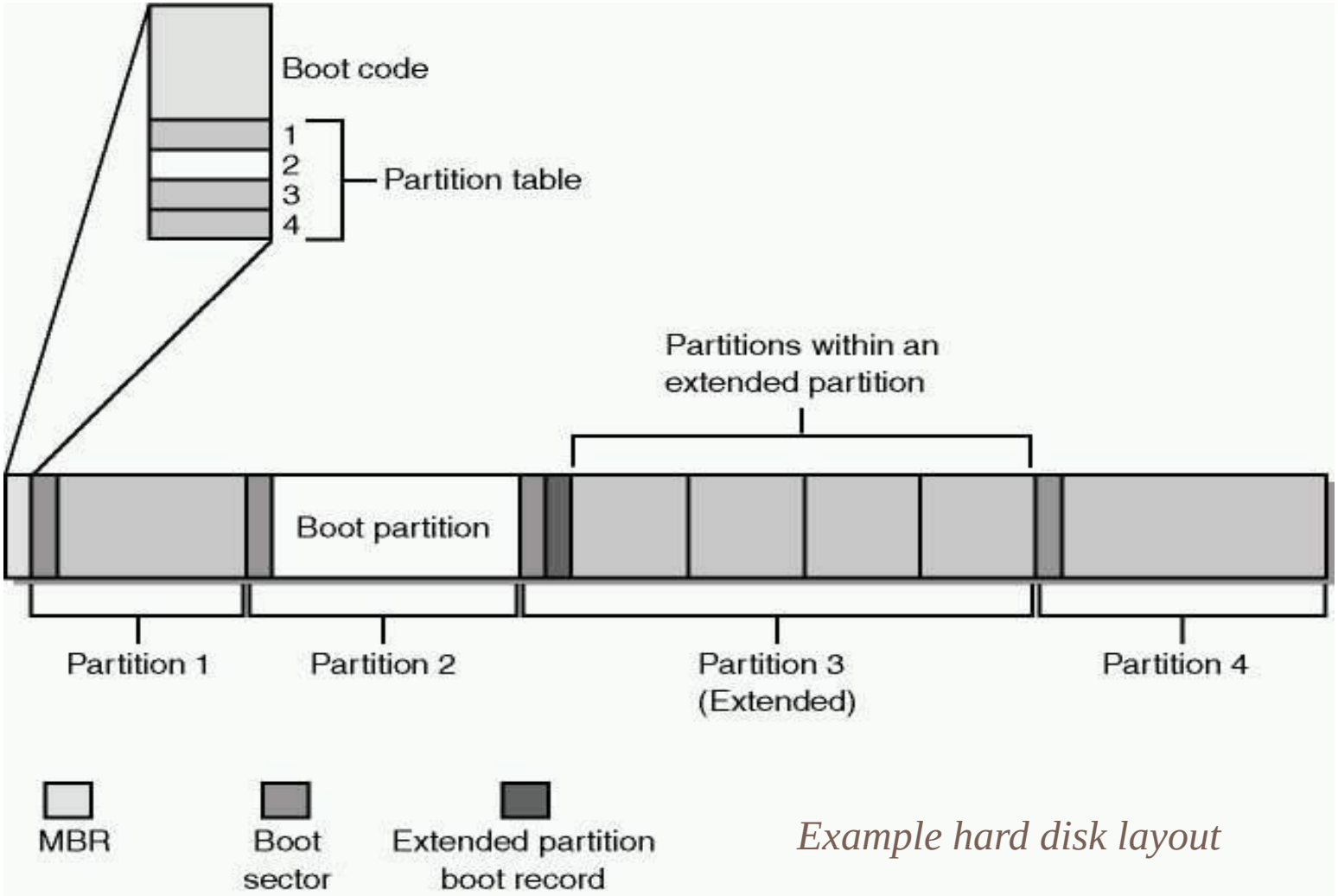
fdisk命令可以查看分区信息

- 因为交换分区要和内存不断交换，所以是动态变化的

```
cst:~# cat /proc/meminfo
MemTotal:      1036564 kB
MemFree:         30584 kB
SwapTotal:     2650684 kB
SwapFree:      2650636 kB
```

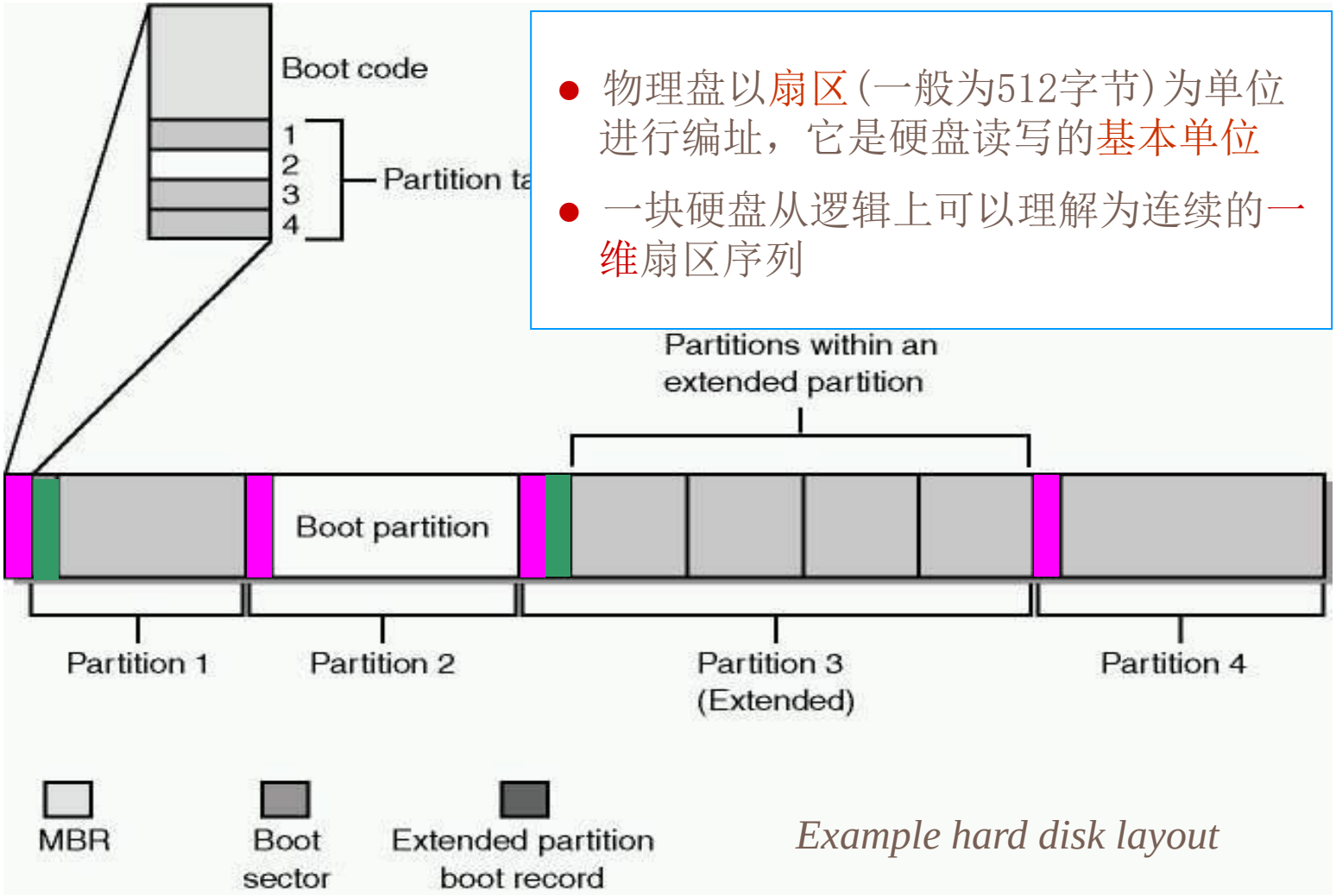
swap分区的大小通常是内存大小的两倍

11.4 磁盘分区



Example hard disk layout

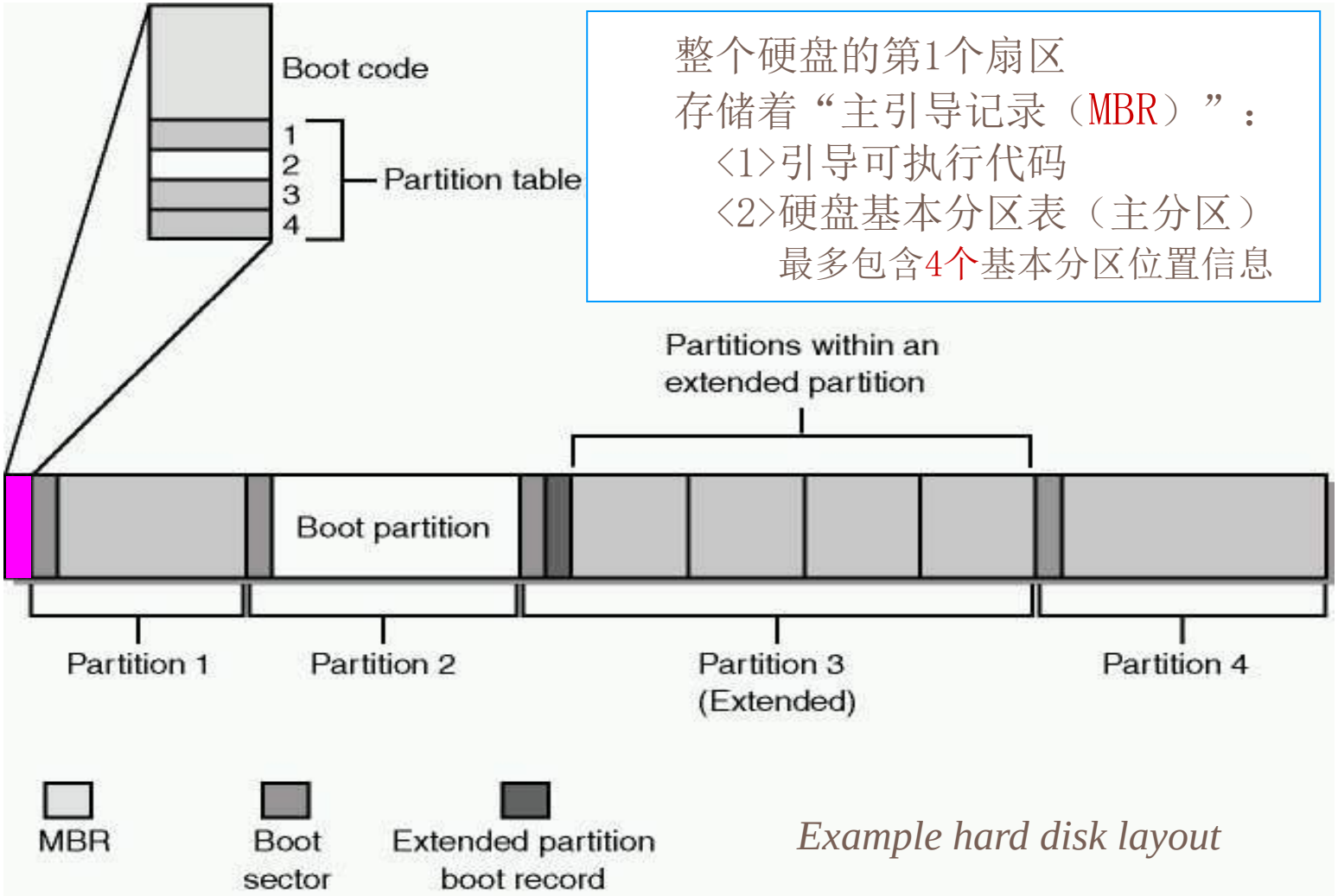
11.4 磁盘分区



- 物理盘以扇区(一般为512字节)为单位进行编址，它是硬盘读写的基本单位
- 一块硬盘从逻辑上可以理解为连续的一维扇区序列

Example hard disk layout

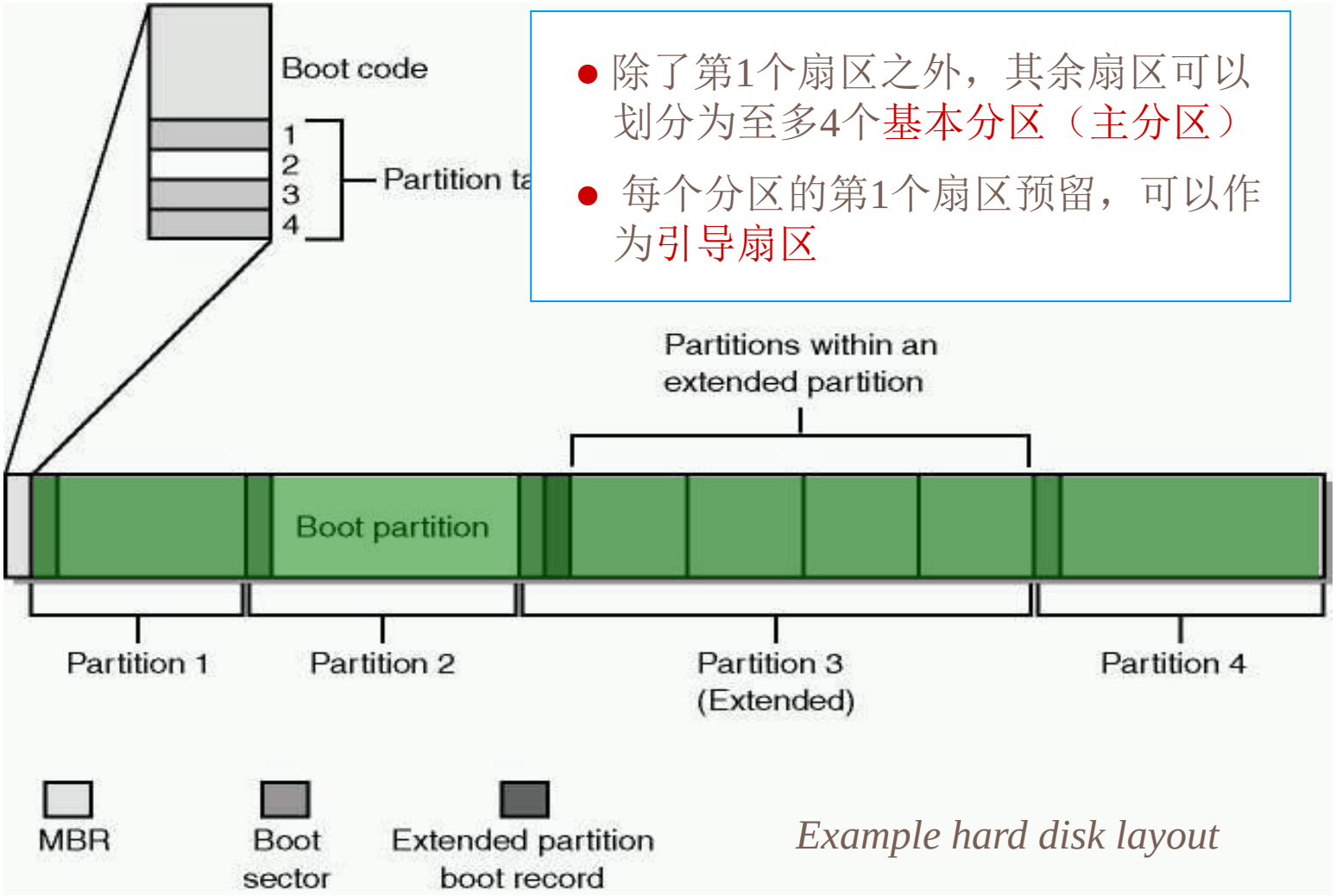
11.4 磁盘分区



整个硬盘的第1个扇区
存储着“主引导记录（MBR）”：
<1>引导可执行代码
<2>硬盘基本分区表（主分区）
最多包含4个基本分区位置信息

Example hard disk layout

11.4 磁盘分区

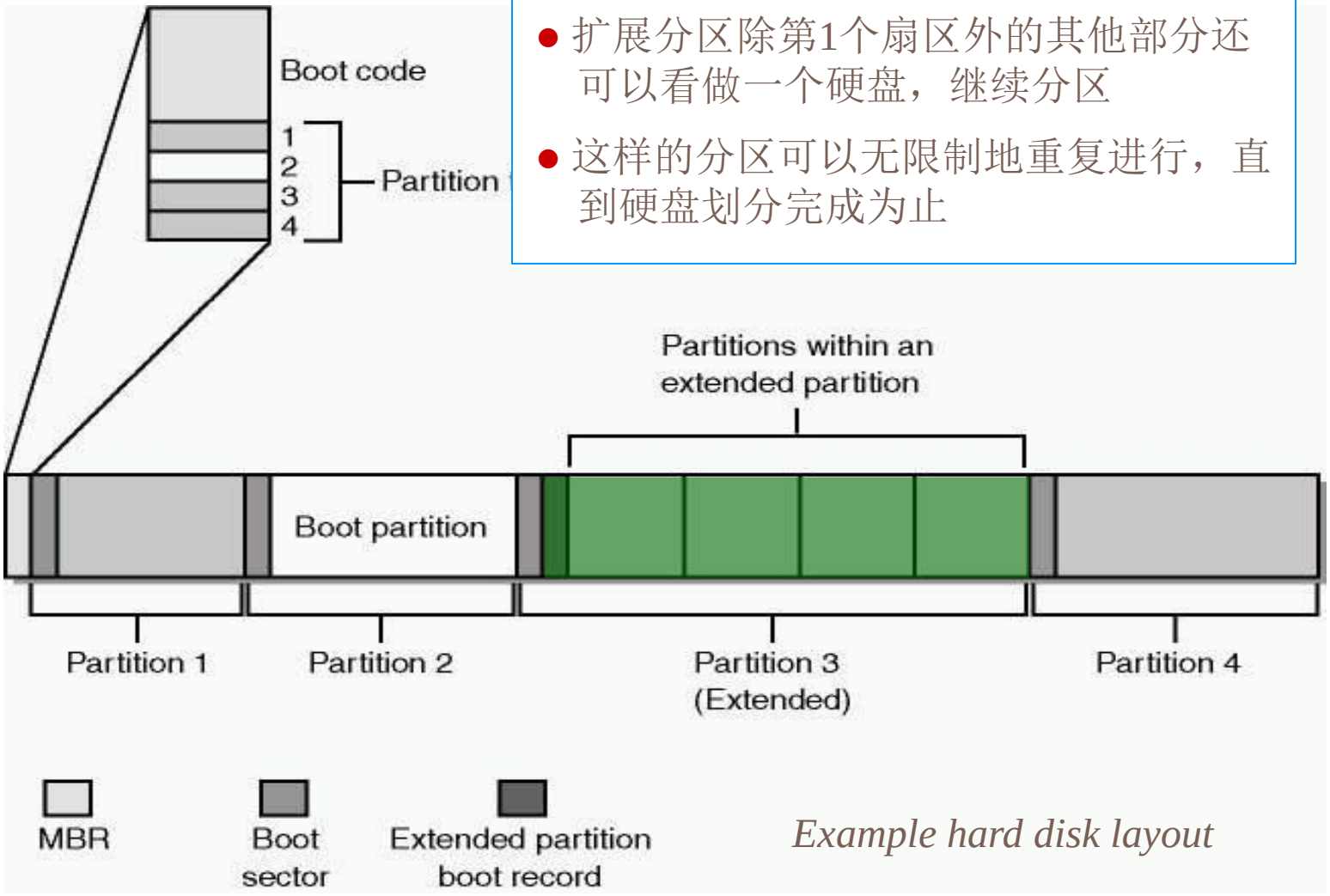


- 除了第1个扇区之外，其余扇区可以划分为至多4个基本分区（主分区）
- 每个分区的第1个扇区预留，可以作为引导扇区

Example hard disk layout

11.4 磁盘分区

- 扩展分区除第1个扇区外的其他部分还可以看做一个硬盘，继续分区
- 这样的分区可以无限制地重复进行，直到硬盘划分完成为止

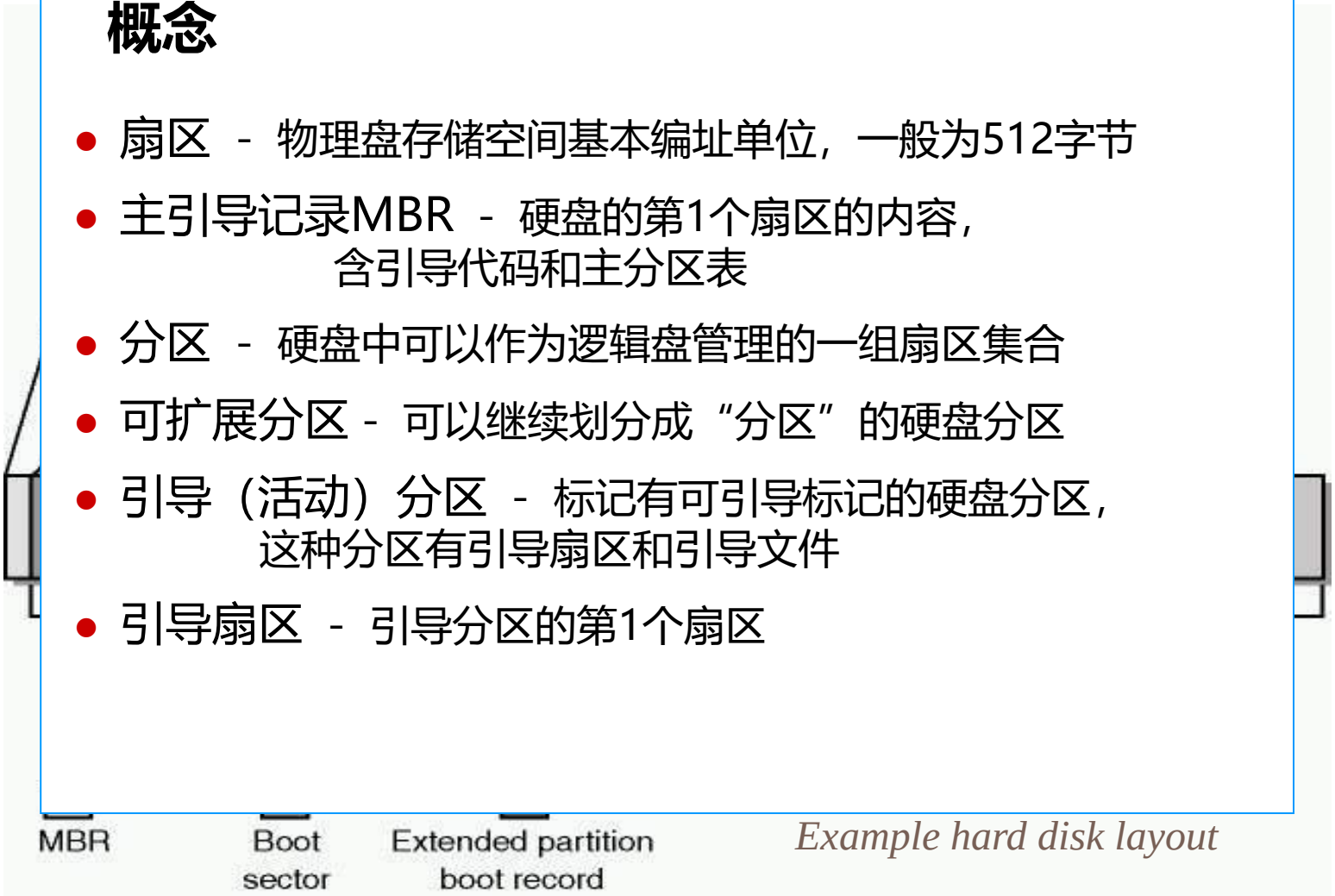


Example hard disk layout

11.4 磁盘分区

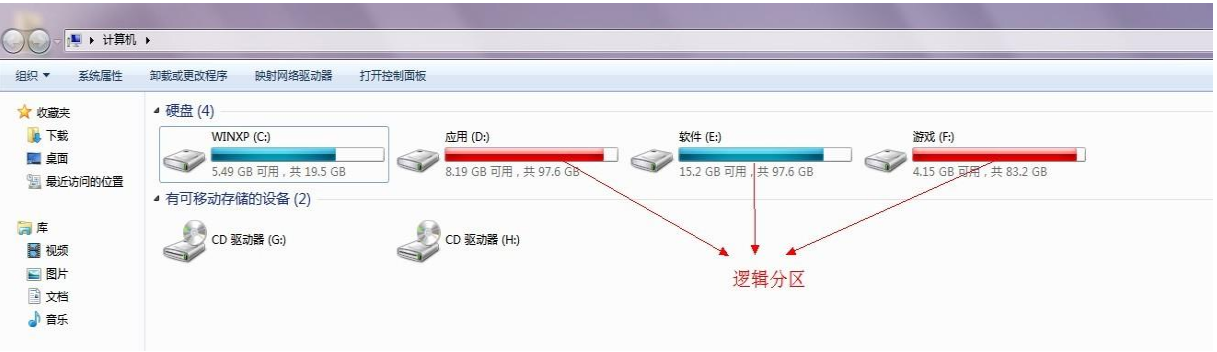
概念

- 扇区 - 物理盘存储空间基本编址单位，一般为512字节
- 主引导记录MBR - 硬盘的第1个扇区的内容，含引导代码和主分区表
- 分区 - 硬盘中可以作为逻辑盘管理的一组扇区集合
- 可扩展分区 - 可以继续划分成“分区”的硬盘分区
- 引导（活动）分区 - 标记有可引导标记的硬盘分区，这种分区有引导扇区和引导文件
- 引导扇区 - 引导分区的第1个扇区



Example hard disk layout

11.4 磁盘分区



在MBR分区表中最多4个主引导程序功能：

1)扫描分区表查找活动分区

2)寻找活动分区的起始扇区

3)将活动分区的引导扇区读到内存

4)执行引导扇区的运行代码

主分区或者3个主分区+1个扩展分区（也就是说扩展分区只能有一个，然后可以再细分为多个逻辑分区。

（安装了操作系统的分区）

MBR由三部分构成：

1．主引导程序代码，占446字节

2．硬盘分区表DPT，占64字节

3．主引导扇区结束标志AA55H

系统在分区时，各分区都不允许跨柱面，即均以柱面为单位，这就是通常所说的分区粒度。

11.5 磁盘高速缓存与阵列

1、磁盘高速缓存(Disk Cache)

- 数据交付指将磁盘高速缓存中的数据传送给请求者进程
- 步骤：先查缓存、后查磁盘并更新缓存

磁盘高速缓存的形式

- 逻辑上是磁盘、物理上是驻留在内存中的盘块
- 固定大小和可变大小

置换算法

- 最近最久
- 访问频率
- …

11.5 磁盘高速缓存与阵列

2、提高磁盘I/O速度的其它方法

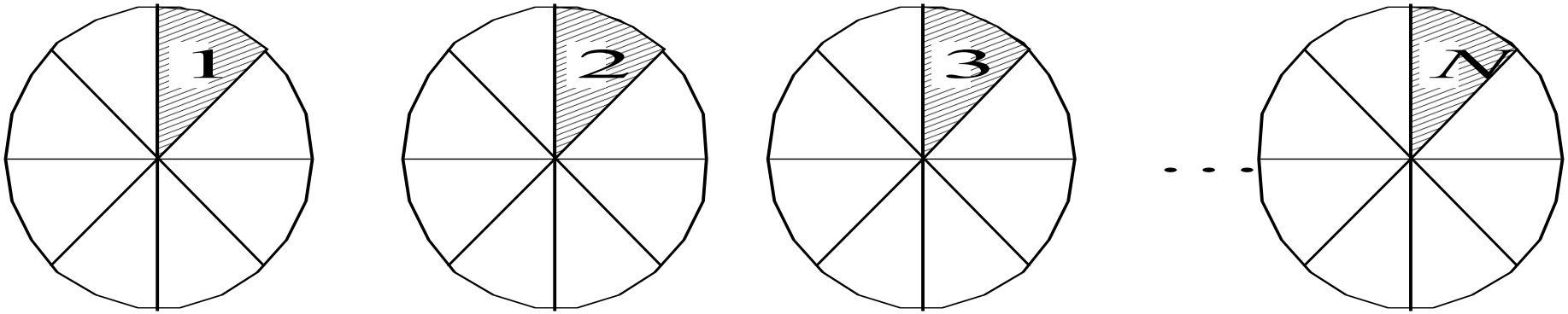
- 提前读
- 延迟写
  - 访问频率高的磁盘块放在替换队列的尾部，减少回写次数
- 优化物理块的分布
  - 目的是减小磁头移动距离
  - 簇分配方式：一个簇为多个连续的块
- 虚拟盘（RAM盘）
  - 和磁盘高速缓存区别：虚拟盘由用户控制；磁盘高速缓存由系统控制。

11.5 磁盘高速缓存与阵列

廉价磁盘冗余阵列

口廉价磁盘冗余阵列RAID（Redundant Arrays of Inexpensive Disk）：是利用一台磁盘阵列控制器，来统一管理和控制一组（几台到几十台）磁盘驱动器，组成一个高度可靠的、快速的大容量磁盘系统。

(1) 并行交叉存取



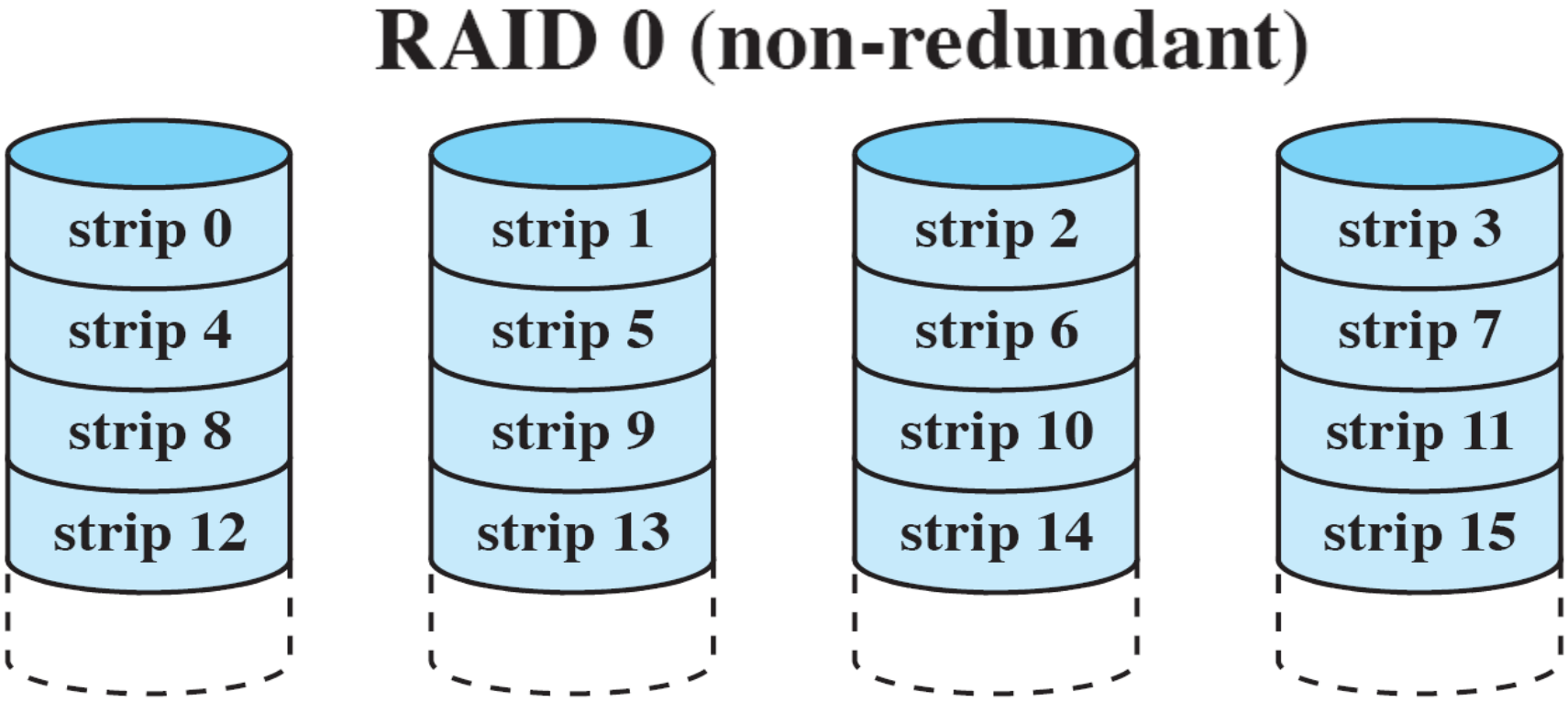
磁盘并行交叉存取方式

(2) RAID技术及分级

- 通过把多个磁盘组织在一起，作为一个逻辑卷提供磁盘跨越功能。
- 通过把数据分成多个数据块，并行写入/读出多个磁盘，以提高访问磁盘的速度。
- 通过镜像或校验操作，提供容错能力。

RAID0	仅提供并行交叉存取。具有并行读写功能，提高了磁盘的I/O速度，但无冗余校验功能。
RAID1	提供磁盘镜像功能。具有并行读写功能，提高了磁盘的I/O速度，但磁盘的利用率仅50%。
RAID3	具有并行传输功能的磁盘阵列。用一台奇偶校验盘容错。
RAID5	具有独立传输功能的磁盘阵列。每个驱动区有自己独立的数据通道。无专门的校验盘，校验信息以螺旋方式分布在每个盘上。
RAID6 RAID7	RAID6设有一个专用的、快速访问的异步校验盘。 RAID7是对RAID6的改进。所有盘都有较高的传输率及有一的性能

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



- 提供并行交叉存取；
- 能有效提高磁盘I/O速度；
- 无冗余校验功能，磁盘系统的可靠性差。

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列

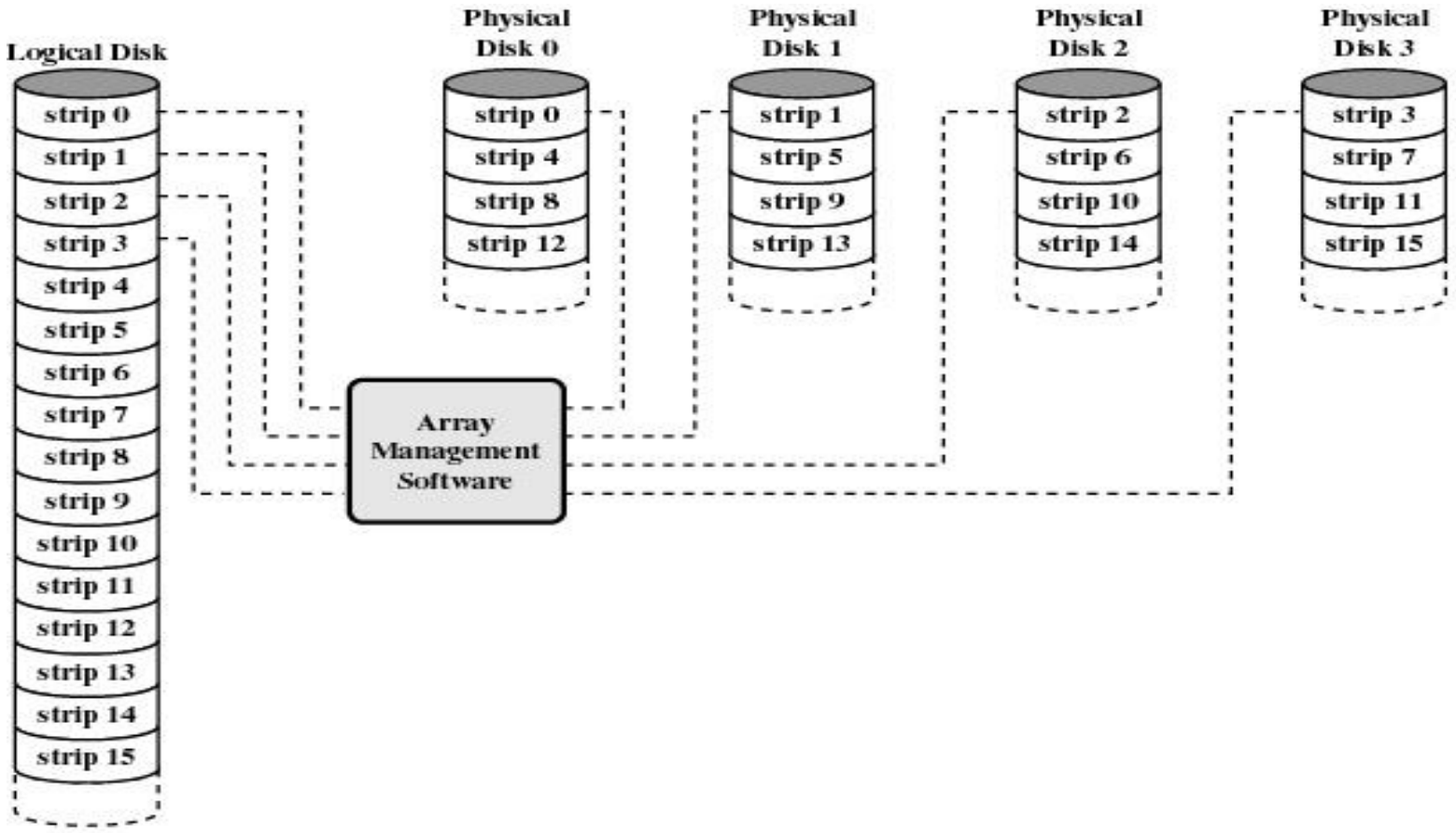
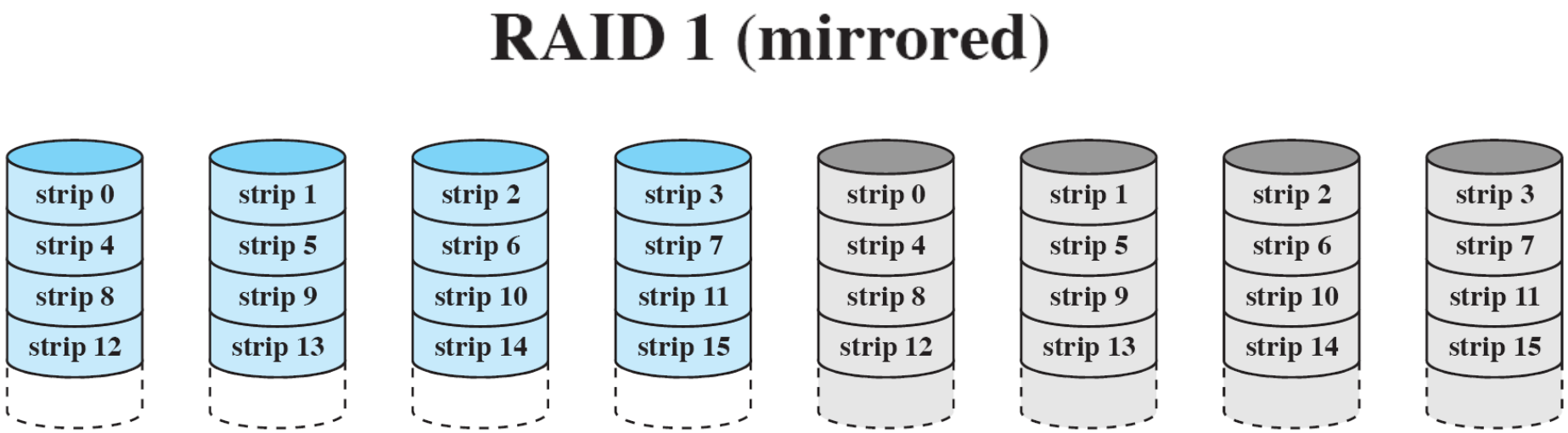


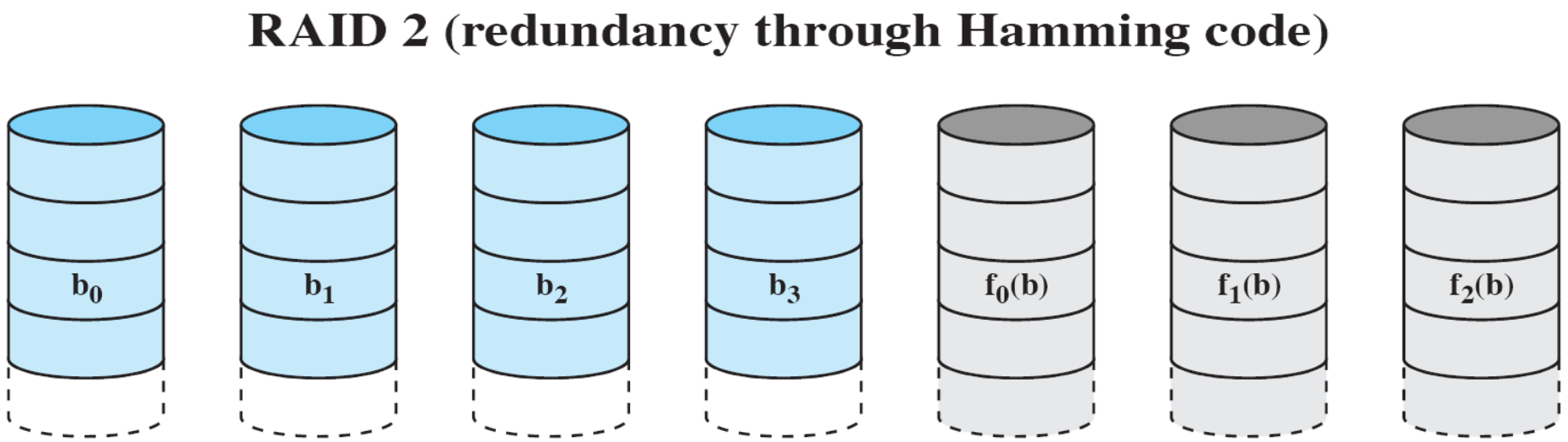
Figure 11.10 Data Mapping for a RAID Level 0 Array [MASS97]

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



- 具有镜像功能；
- 具有并行读写功能，提高了磁盘的I/O速度，但磁盘的利用率仅50%。

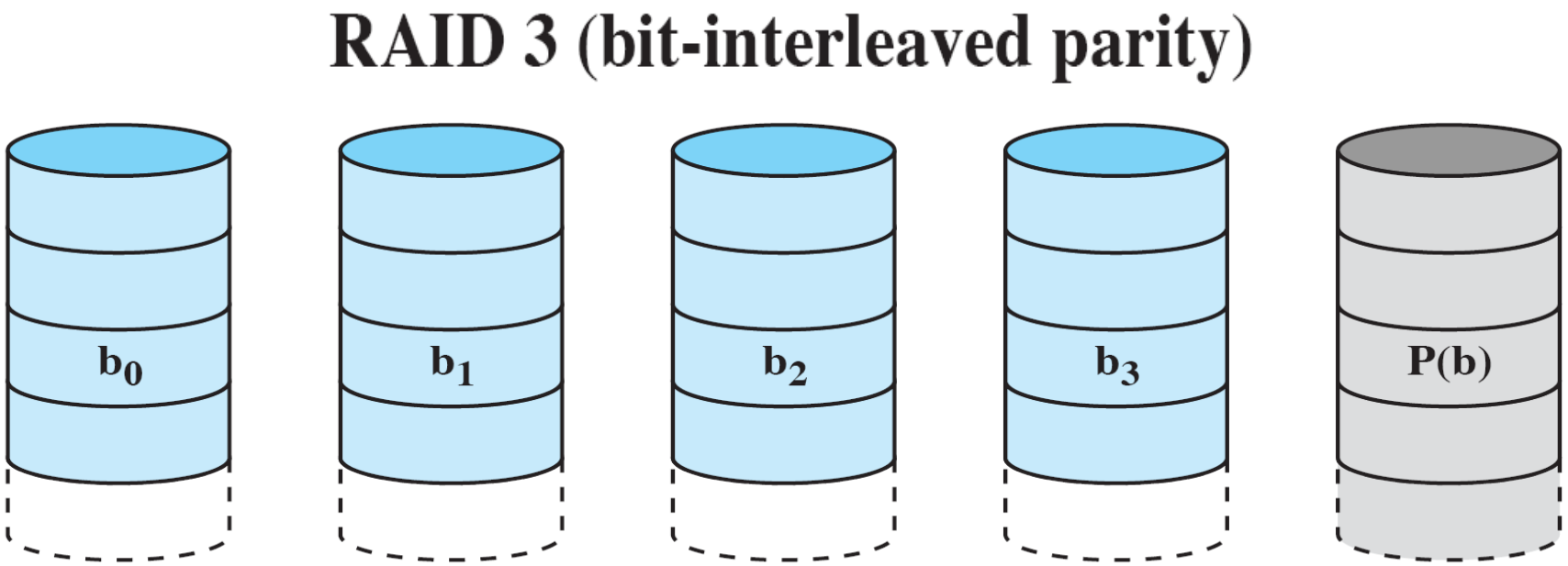
11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



- 采用了早期的错误检测与修正技术----汉明码（Hamming Code ）校验技术进行即时数据校验，冗错性较好；
- 一个硬盘在一个时间只存取一位的信息，但具有极高的数据传输率；
- RAID 2 中的硬盘数量取决于所设定的数据存储宽度；
- 系统成本极高，对冗余的数据传输率要求较高。

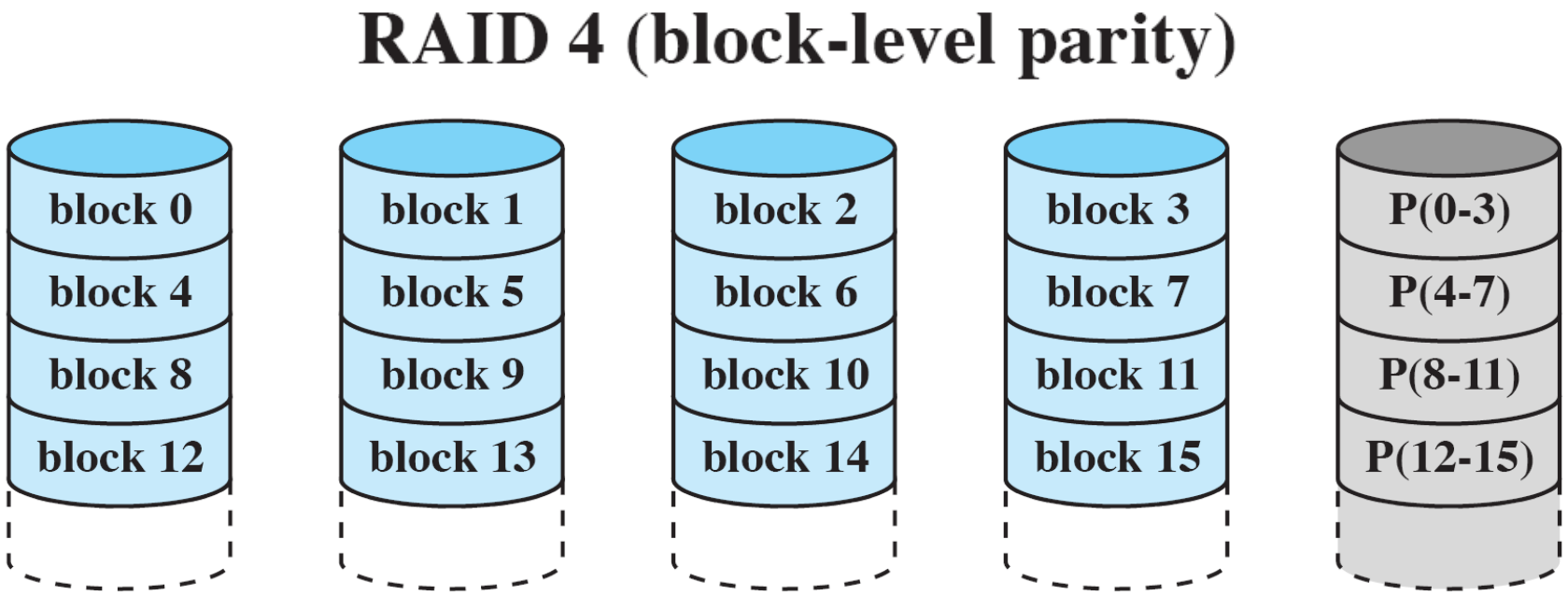
注：汉明码的数量与数据位的数量之间比例公式为，$2^P \geq P+D+1$，P 代表汉明码的个数，D 代表数据位的个数

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



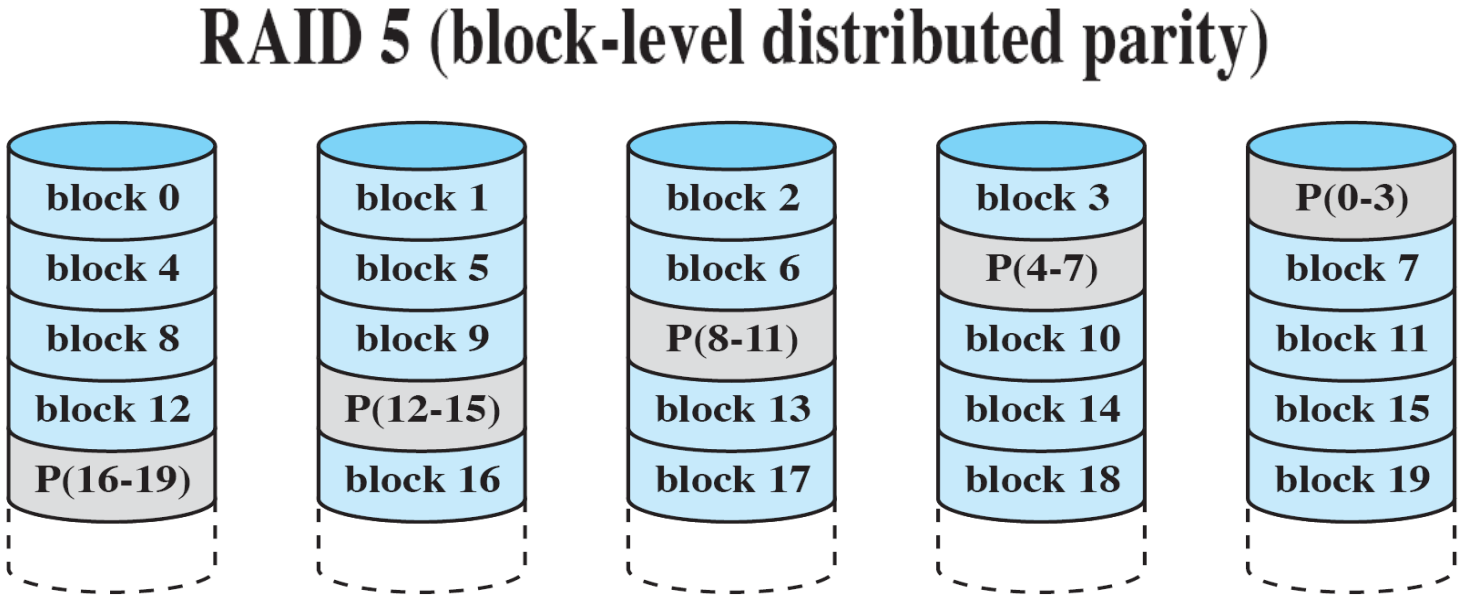
- 具有并行传输功能的磁盘阵列；
- 用一台奇偶校验盘容错；
- 磁盘利用率为（N-1）/N；
- 常用于科学计算和图像处理。

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



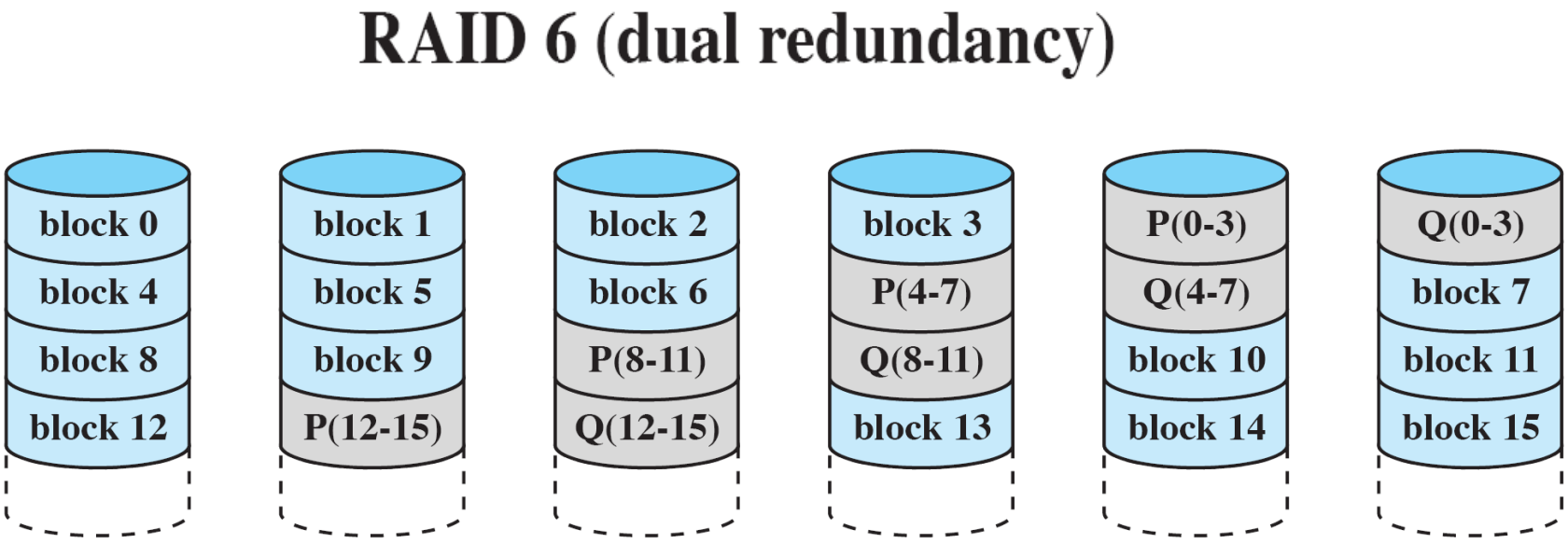
- 独立的数据硬盘与共享的校验硬盘；
- 按数据块为单位进行存储；
- 在不同硬盘上的同级数据块也都通过XOR 进行校验，结果保存在单独的校验盘；
- 相对较高的读取传输率，极差的写入传输率（在写入时要等一个硬盘写完后才能写一下个，并且还要写入校验数据）。

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



- 具有独立传输功能的磁盘阵列；
- 每个驱动区有自己独立的数据通道；
- 无专门的校验盘，校验信息以螺旋方式分布在每个盘上；
- 常用于I/O较频繁的事务处理。

11.5 磁盘高速缓存与阵列 廉价磁盘冗余阵列



- 设置了一个专用的、可快速访问的异步校验盘；
- 具有独立的数据访问通道；
- 具有比RAID 3级及RAID 5级更好的性能，但性能改进很有限；
- 价格昂贵。

RAID的优点

(1) 可靠性高。

(2) 磁盘I/O速度高。

(3) 性能/价格比高。

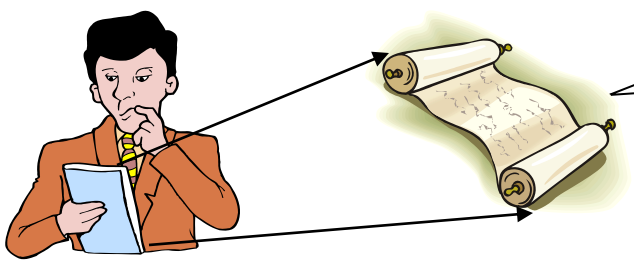
11.6 文件概念及实现方法

为什么引入文件?

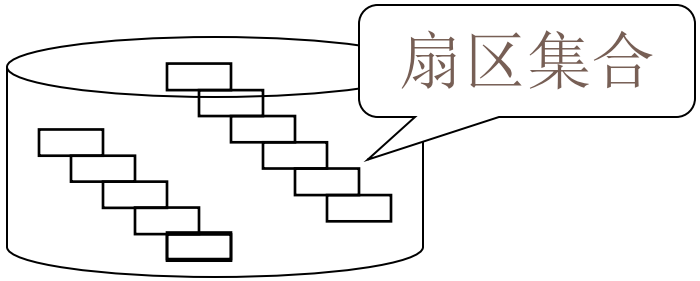
—"烹调”磁盘

为什么引入文件?

- 让普通用户使用raw disk: 许多人连扇区都不知道是什么?要求他们根据扇区编号来访问磁盘…
  - 需要在扇区上引入更高一层次的抽象概念! 文件
  - 首先想一想用户眼里的文件是什么样子?



  - 磁盘上的文件是什么样子?

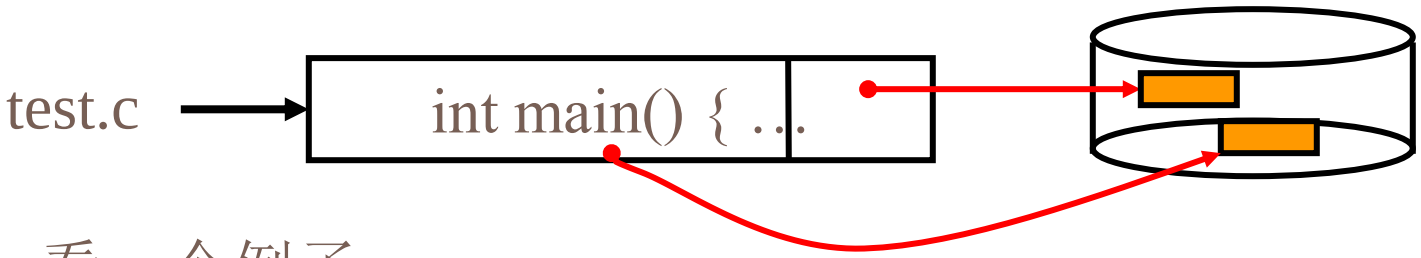


  - 文件: 建立字符流到扇区集合的映射关系

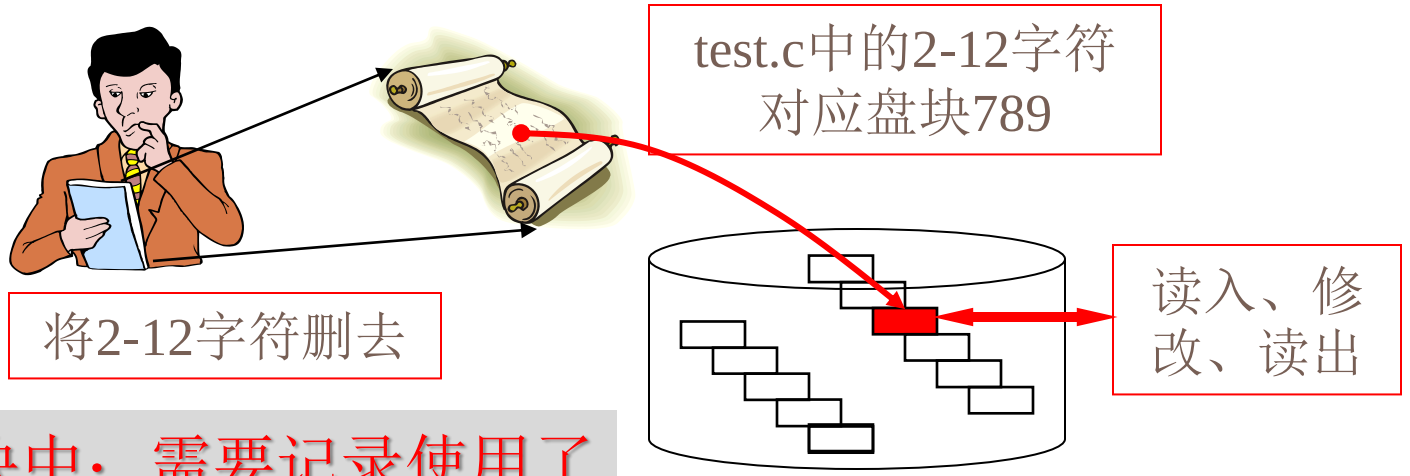
文件概念

为增加灵活性，OS又将多个连续扇区定义为盘块

- 建立字符流到盘块集合的映射关系

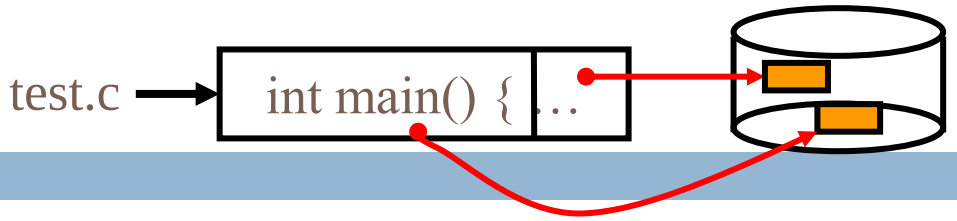


- 看一个例子

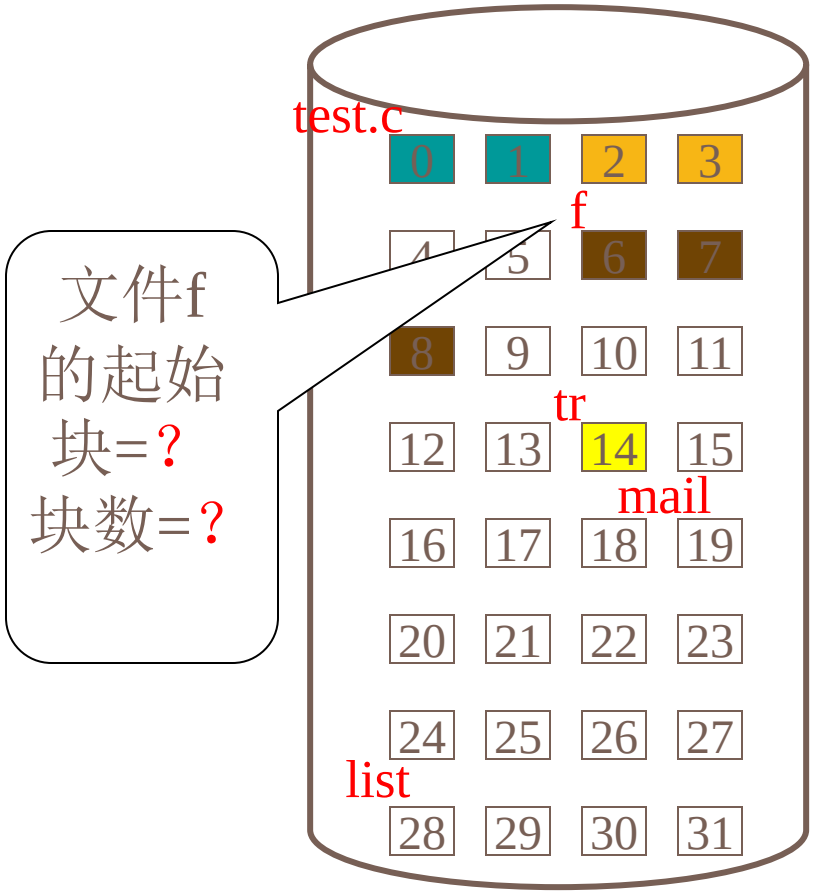


文件存储到盘块中：需要记录使用了哪些盘快，这些盘块的顺序关系

文件的实现



- 文件抽象概念的实现关键：描述这一映射关系
  - 文件实现1: 物理盘块连续分配



- 需存放什么信息?

  起始盘块和盘块个数

- 存放在哪里?

  文件描述信息节点中

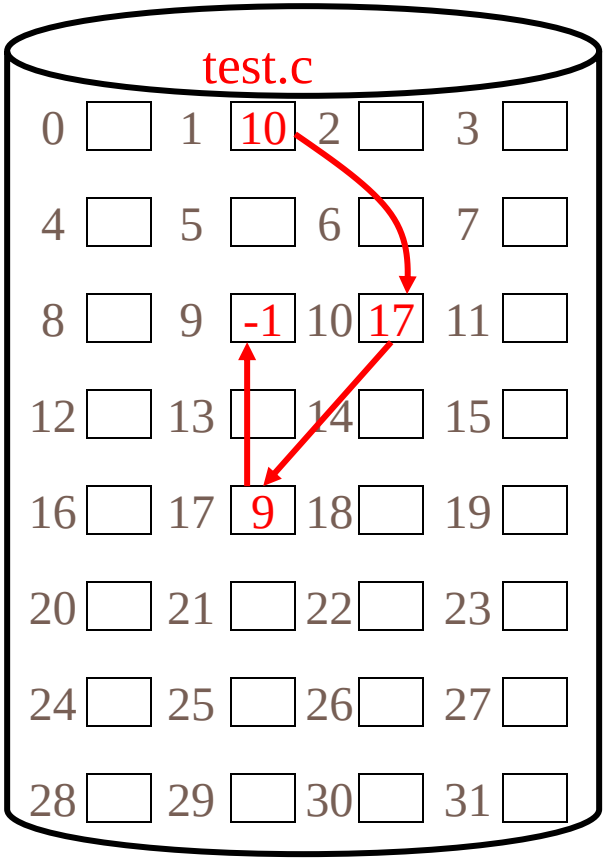
名字很多: FCB, File Header等

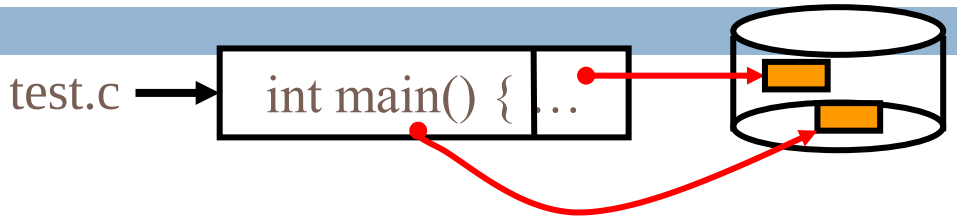
test.c的File Header

文件名	始址	块数
test.c	0	4

- 优点简单快速，缺点？和连续内存分配相比一下！增删改与碎片

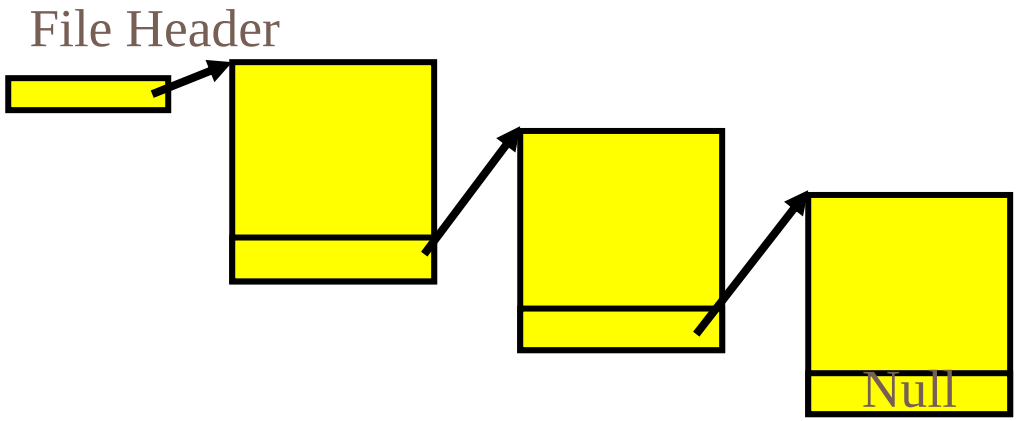
文件实现2: 链式分配





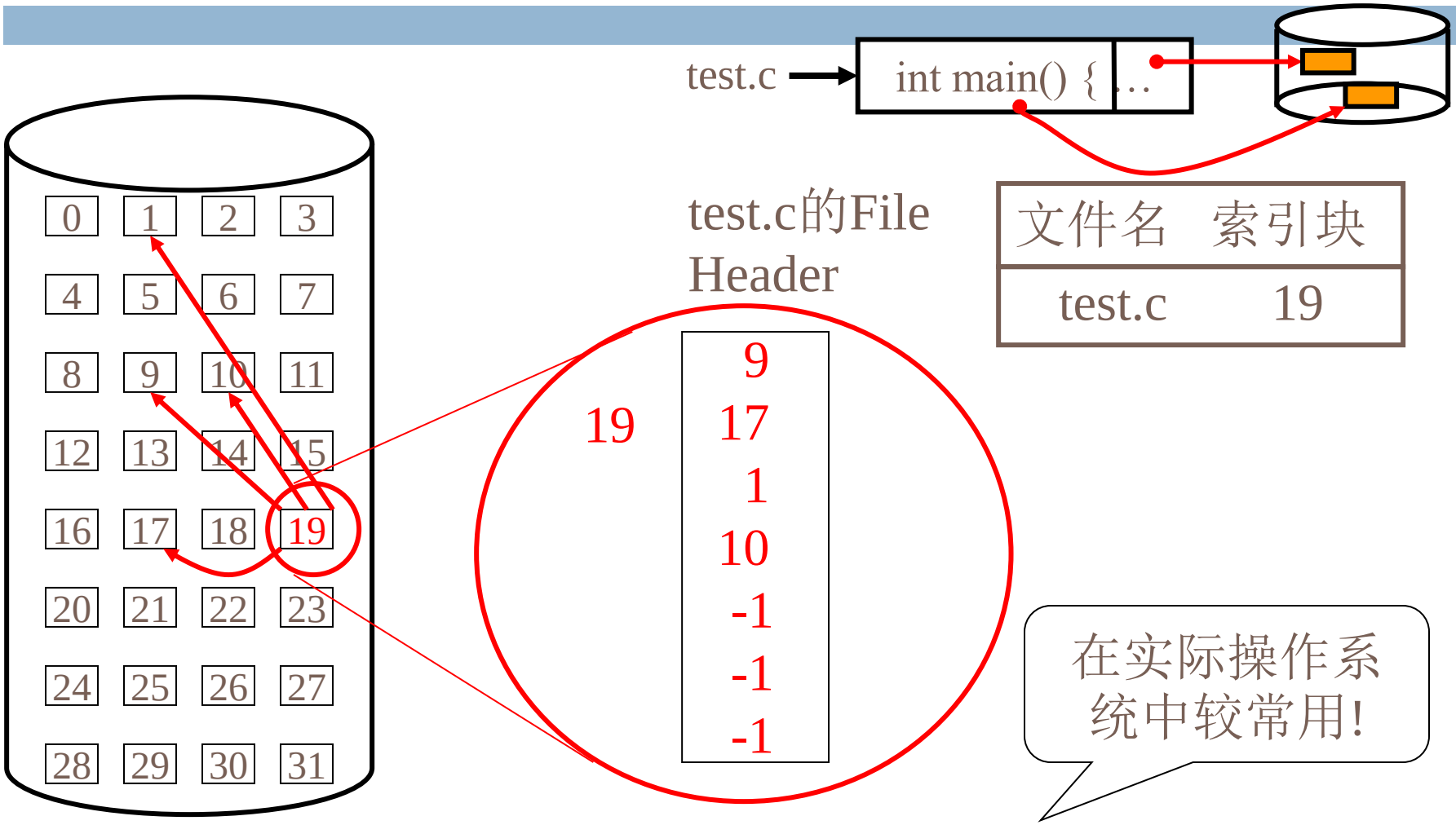
test.c的File Header

文件名	始址
test.c	1



- 优点: 文件长度增减容易
- 缺点: 顺序访问、效率低

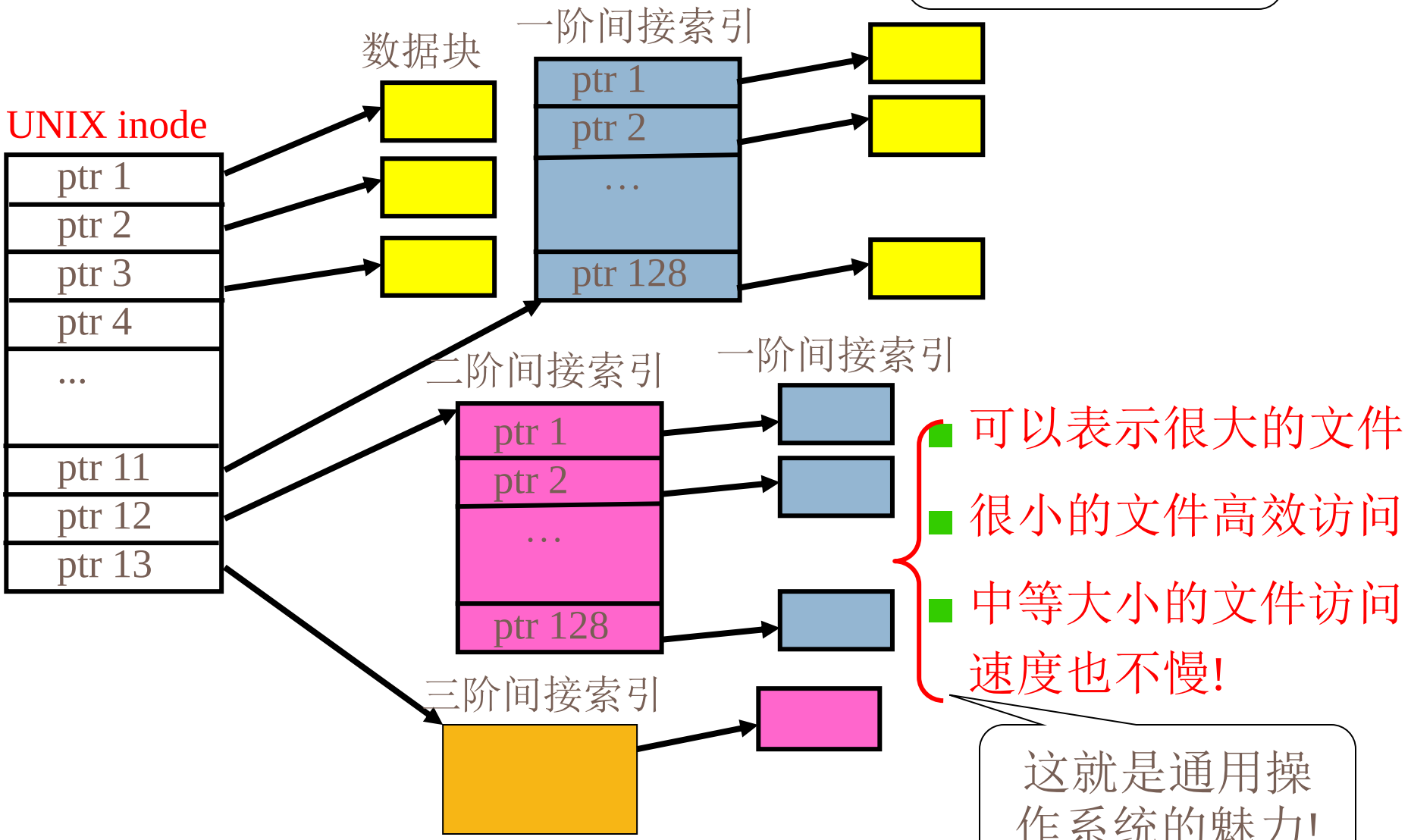
文件实现3: 索引分配



- 优点: 是连续和链式分配的有效折衷

UNIX的索引节点(inode)

根据名字就知道是索引分配!



- 可以表示很大的文件
- 很小的文件高效访问
- 中等大小的文件访问速度也不慢!

这就是通用操作系统的魅力!

磁盘与文件总结

- 磁盘也是一种重要的外设 ⇒ 磁盘提供了大容量存储
- 管理外设首先需要了解外设 ⇒ 认识磁盘结构和读写
- 磁盘结构(读写)⇒柱面(寻道)/磁头(选磁头)/扇区(旋转)⇒传输
- 通过扇区编号来直接访问磁盘 ⇒ 生磁盘 ⇒ 交换分区
- Linux交换分区实现 ⇒ 虚拟内存技术就完整了
- 生磁盘让用户使用是不合适的 ⇒ Cooked Disk
- 熟磁盘: 有意义的扇区集合 ⇒ 文件 ⇒ 字符流到扇区的映射
- 映射方案 ⇒ 连续(象数组) ⇒ 链式(象指针) ⇒ 索引(折衷)