操作系统

Operating Systems

夏文 副教授

xiawen@hit.edu.cn

哈尔滨工业大学（深圳）

2019年12月

第8章 I/O设备管理与存储系统

主要内容

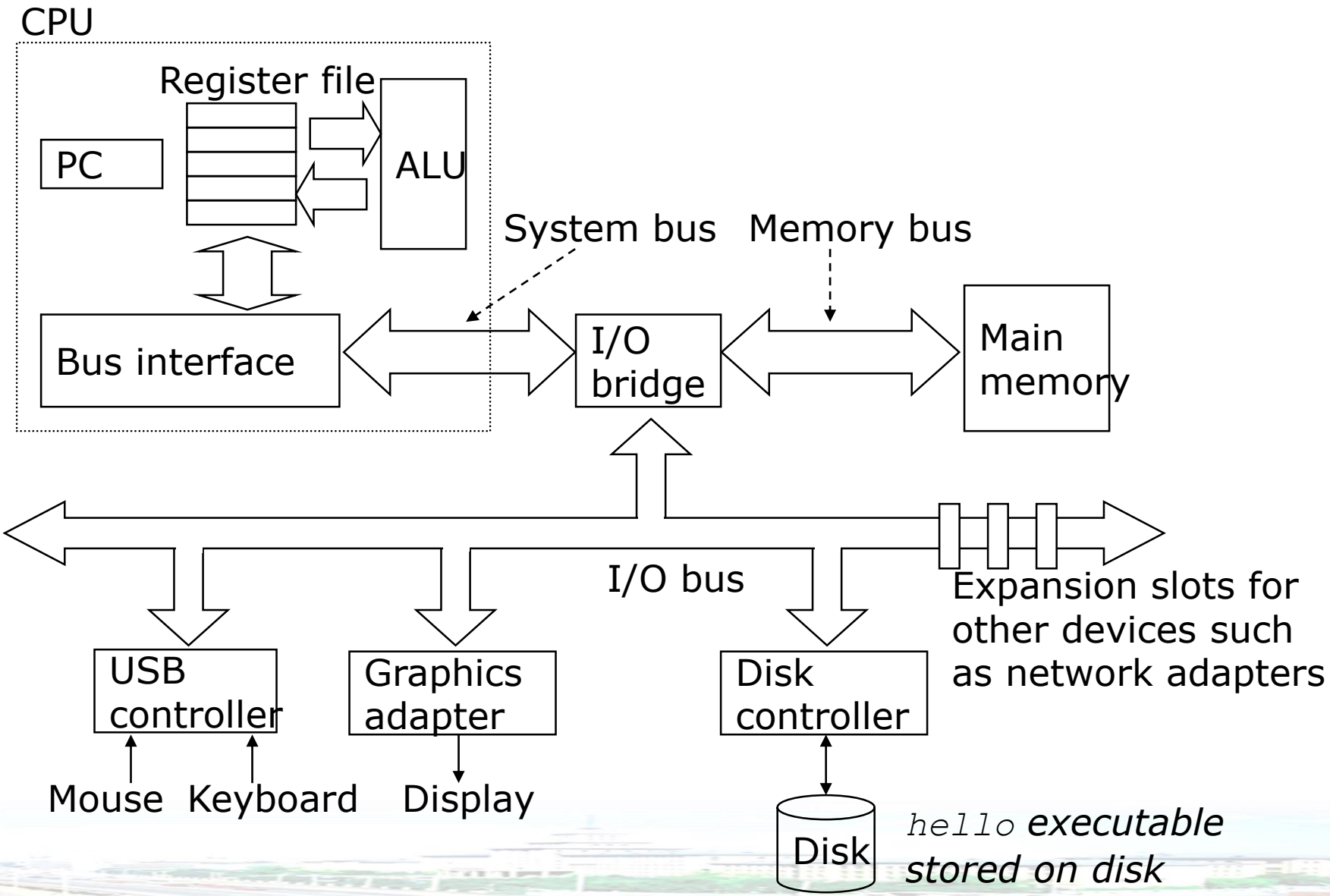
8.1 设备管理概述

8.2 I/O控制方式

8.3 缓冲技术

8.4 磁盘设备

Computer Architecture



计算机系统中的外设

大家为什么要安装设备驱动程序？

计算机包含不同厂商的各种外设

外设不断更新换代

操作系统管理这些外设，为用户提供服务

两端接口规范，中间管理高效

操作系统最关键的生态之一：外设支持

8.1 设备管理概述

- **设备管理是操作系统的重要组成部分**

  同其他管理来说，该部分内容比较复杂凌乱。因为设备种类繁多，各自有着不同的特点，所以需要制定一个通用的、规范的管理框架、方法

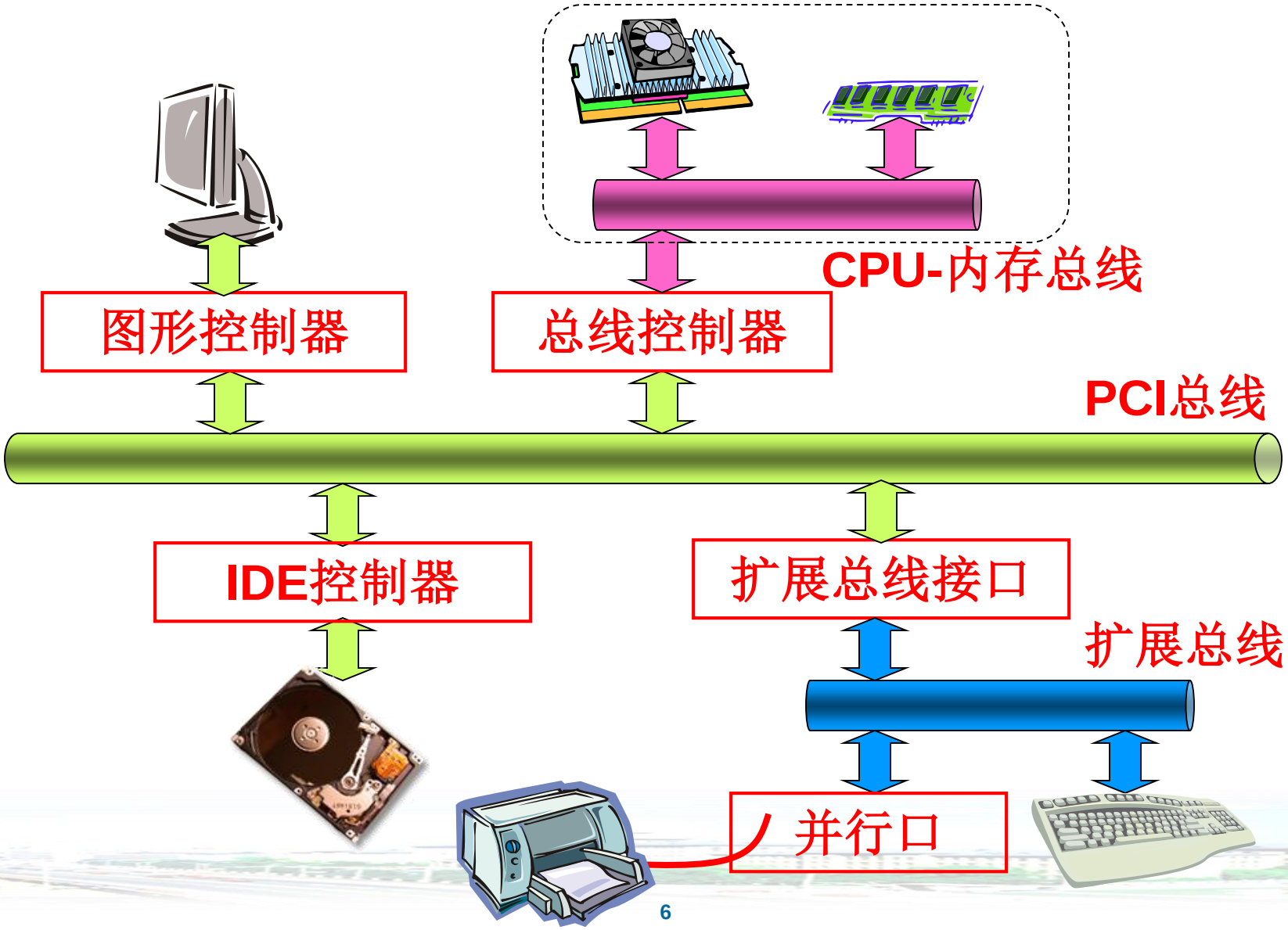
- **设备管理的方法主要有3种：**

（1）操作系统直接操纵设备的运行，例如直接程序控制、中断方式控制

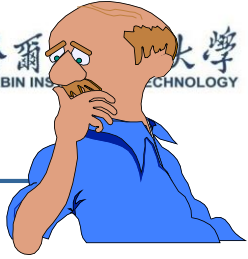
（2）操作系统间接操纵设备的运行，例如DMA和通道方式

（3）操作系统通过使用设备驱动程序，将设备管理工作通过任务（进程）的形式来体现。OS只需制定标准，将具体操纵设备的程序交给不同的制造商去开发

认识计算机外设与计算机!



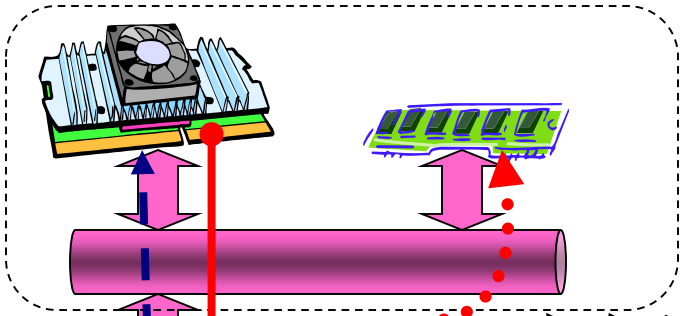
认识计算机外设与计算机!



想一想外设怎么工作?

想让外设工作
并不复杂!

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



CPU-内存总线

总线控制器

PCI总线

IDE控制器

读磁盘

发出一个读命令

将数据送往内存

读完后向CPU发出中断

I/O系统想给用户提供一个什么样的视图?

看一段操纵外设的程序

```
int fd = open("/dev/something");
for (int i = 0; i < 10; i++) {
      fprintf(fd,"Count %d\n",i);
}
close(fd);
```

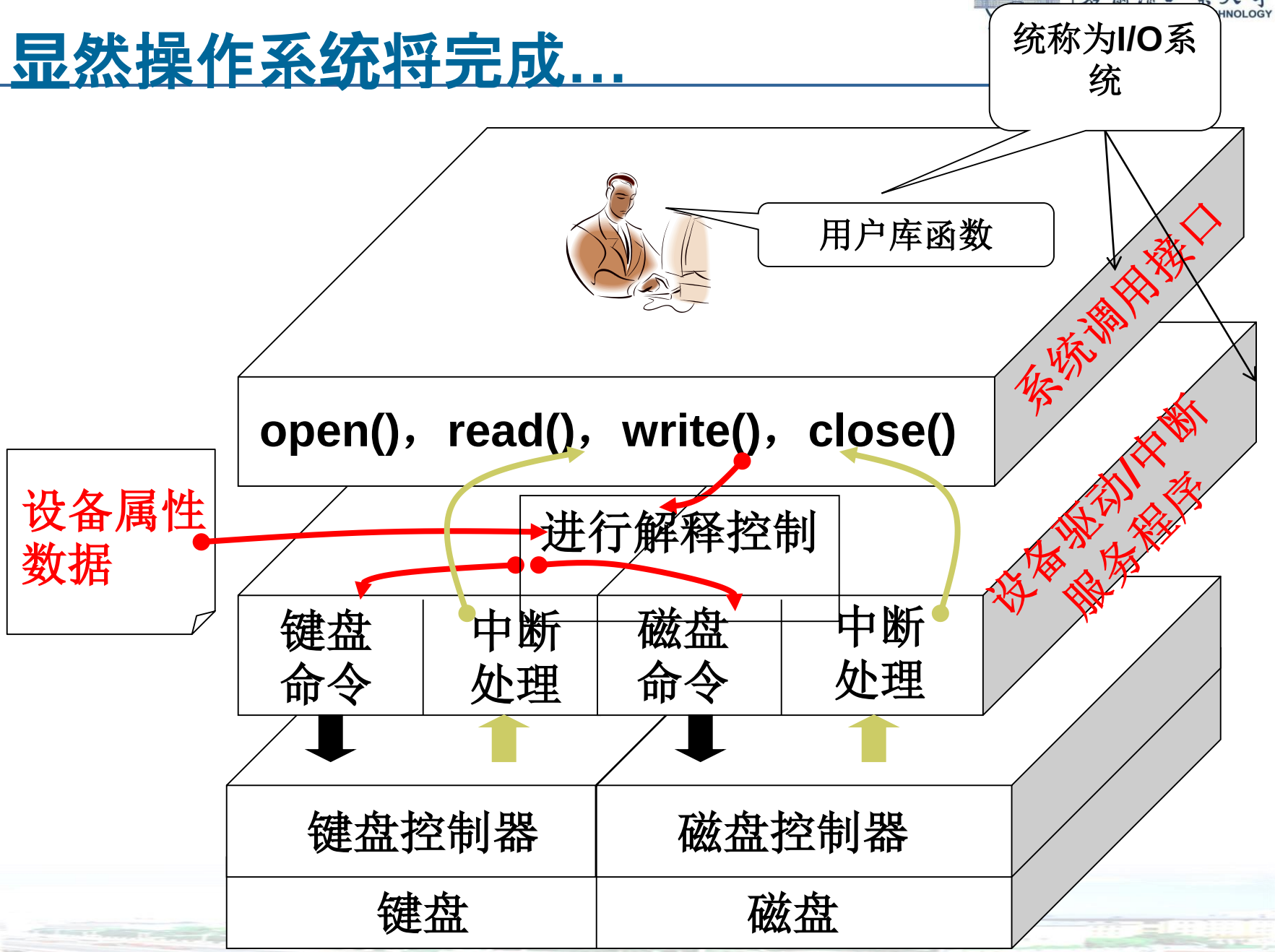
- (1) 不论什么设备都是open, read, write, close

  操作系统为用户提供统一的接口!

- (2) 不同的设备对应不同的文件(设备文件)

  设备文件中存放了设备的属性!

显然操作系统将完成…


统称为I/O系统
用户库函数
系统调用接口
open()，read()，write()，close()
设备属性数据
进行解释控制
设备驱动/中断服务程序
键盘命令
中断处理
磁盘命令
中断处理
键盘控制器
磁盘控制器
键盘
磁盘

IO系统组成

用户库函数接口

IO内核子系统

IO调度（磁盘寻道）、缓冲（文件系统磁盘访问）、错误处理、保护

设备驱动程序（中断服务程序）

隐藏设备差异，为操作系统和设备提供商提供标准的接口

IO硬件

字符设备、块设备；读写；同步异步

IO系统用户接口

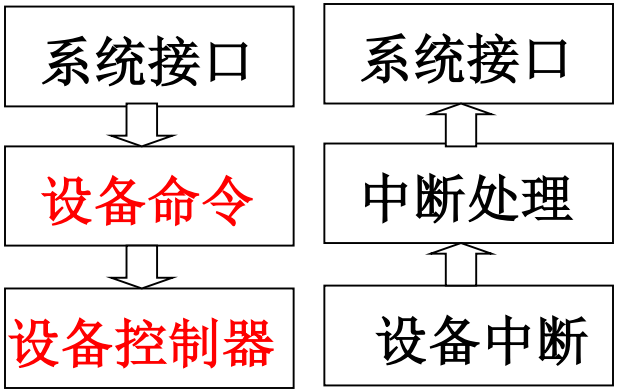
主要接口如下表

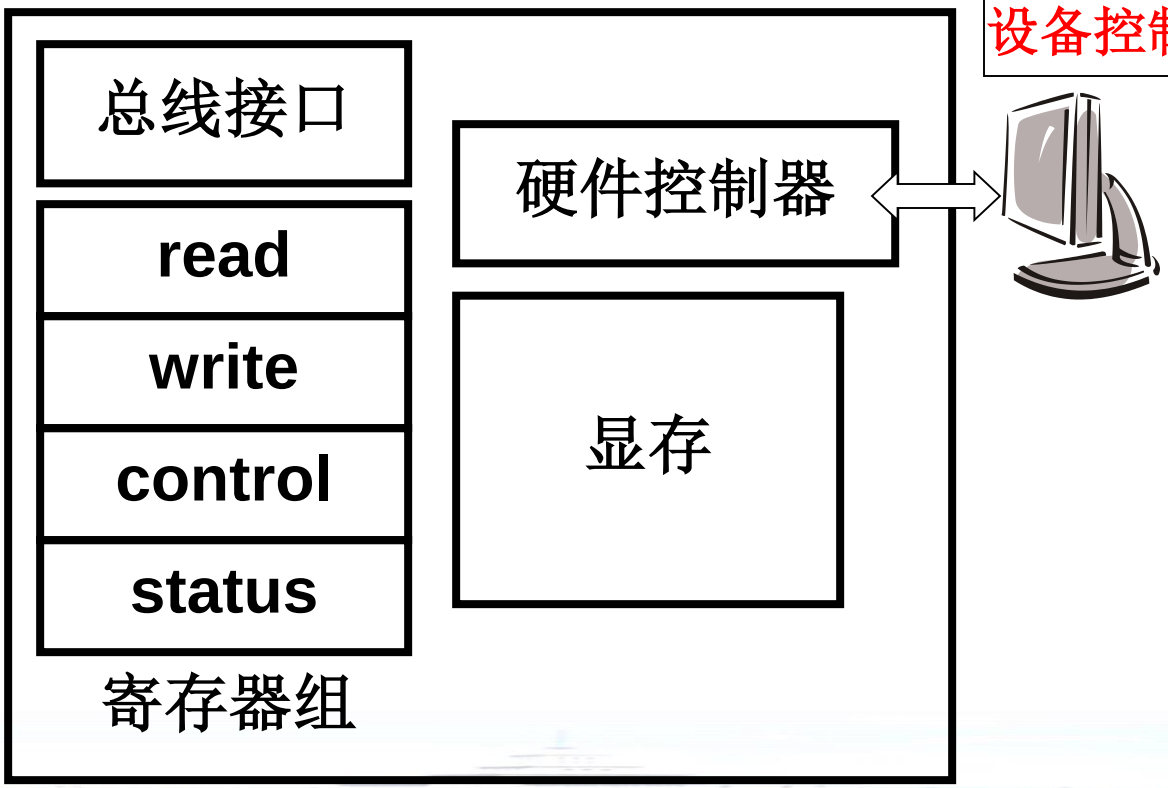
函数名	功能
create()	创建设备
remove()	删除设备
open()	打开设备
close()	关闭设备
read()	从设备中读取数据
write()	向设备中写入数据
ioctl()	控制设备(例如设置波特率等)

Linux I/O: open/close/read/write/lseek

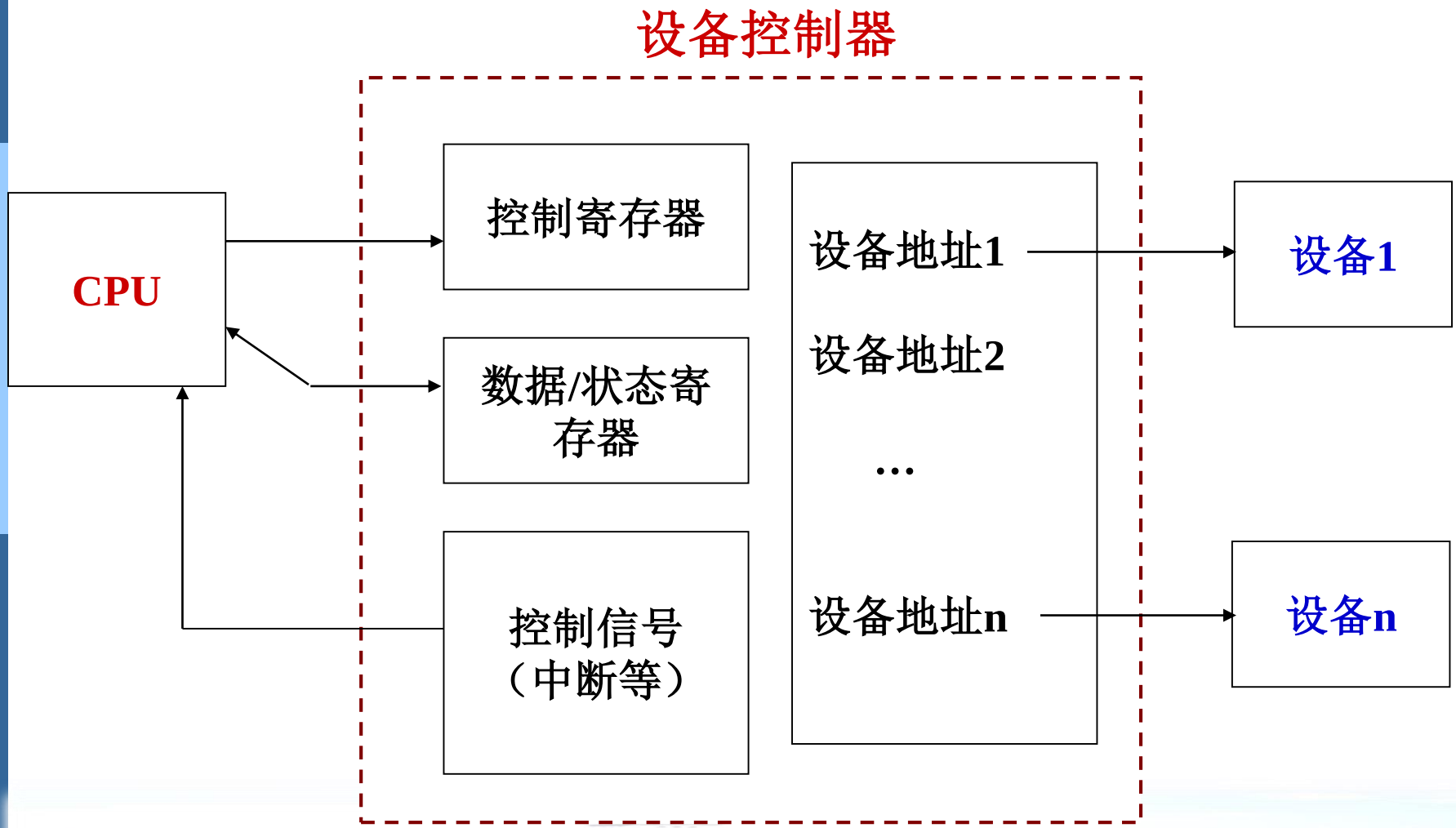
I/O系统如何向设备发命令?

- I/O系统向设备控制器发命令
  - 设备控制器的结构





CPU、设备控制器与设备之间关系

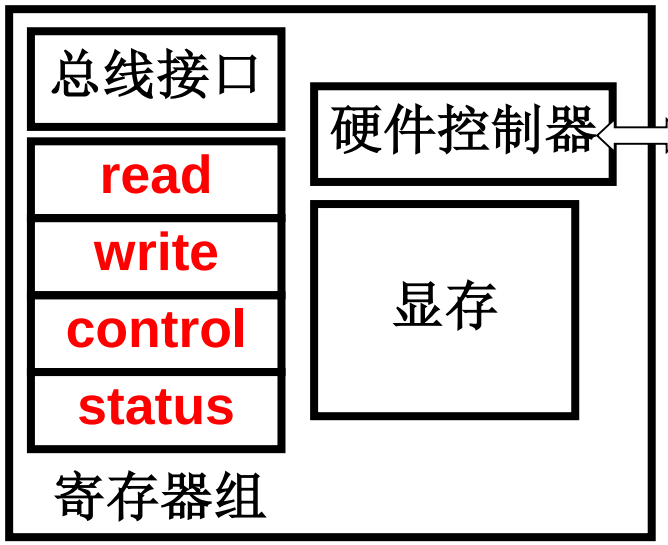


I/O系统向设备发命令方式?

⑩ 读写设备控制器的寄存器!

- 怎么读写? mov [100], ax

关键是地址



- 设备寄存器的编址
- 独立编址: 需要独立的指令。比如X86:in,out;
  只能与DX,AX,AL寄存器结合使用。如out 0x21, AL
- 内存映像编址: 是内存物理地址空间的一部分，使用mov命令，如mov [0x8000f000], AL

查查硬件手册就知道了!

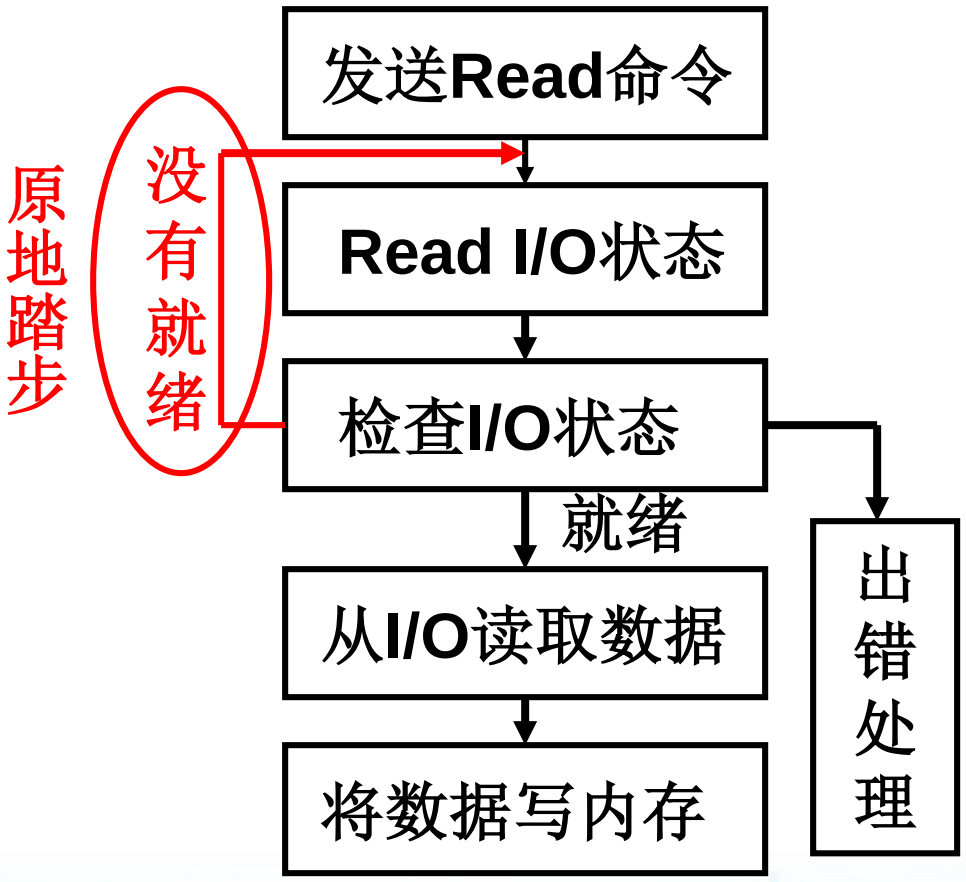
8.2 I/O控制方式

控制I/O硬件的方式?

- 设备管理的主要任务之一是控制设备和内存或CPU之间的数据传送
- I/O控制方式一般有4种：
  - 程序直接控制（查询）方式
  - 中断控制方式
  - 直接内存存取（DMA）方式
  - 通道控制方式（智能设备）

控制I/O硬件的方式

❿ 方案1: 原地踏步等待!



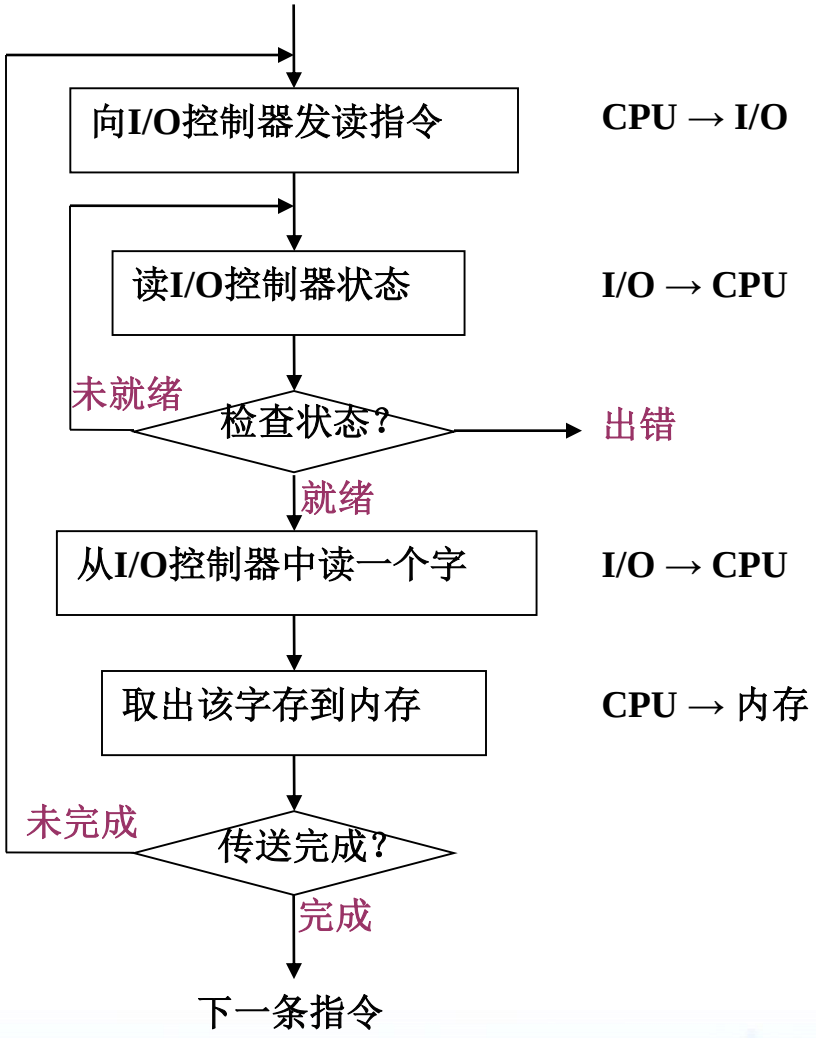
查询（轮询）

```
in AL, 0x??
while(AL!=ready)
{
   in AL, 0x??
}
读数据..
```

轮询!

浪费CPU资源(CPU比外设快太多了)!

例子：程序方法控制I/O设备读入数据流程



例子：程序方法控制I/O设备读入数据流程

程序直接控制(查询)方式 工作步骤小结：

(1) 当某进程需要输入/输出数据时，由CPU向设备控制器发出一条I/O指令启动设备工作(对于输出操作，则CPU还要向数据寄存器中存放输出数据)；

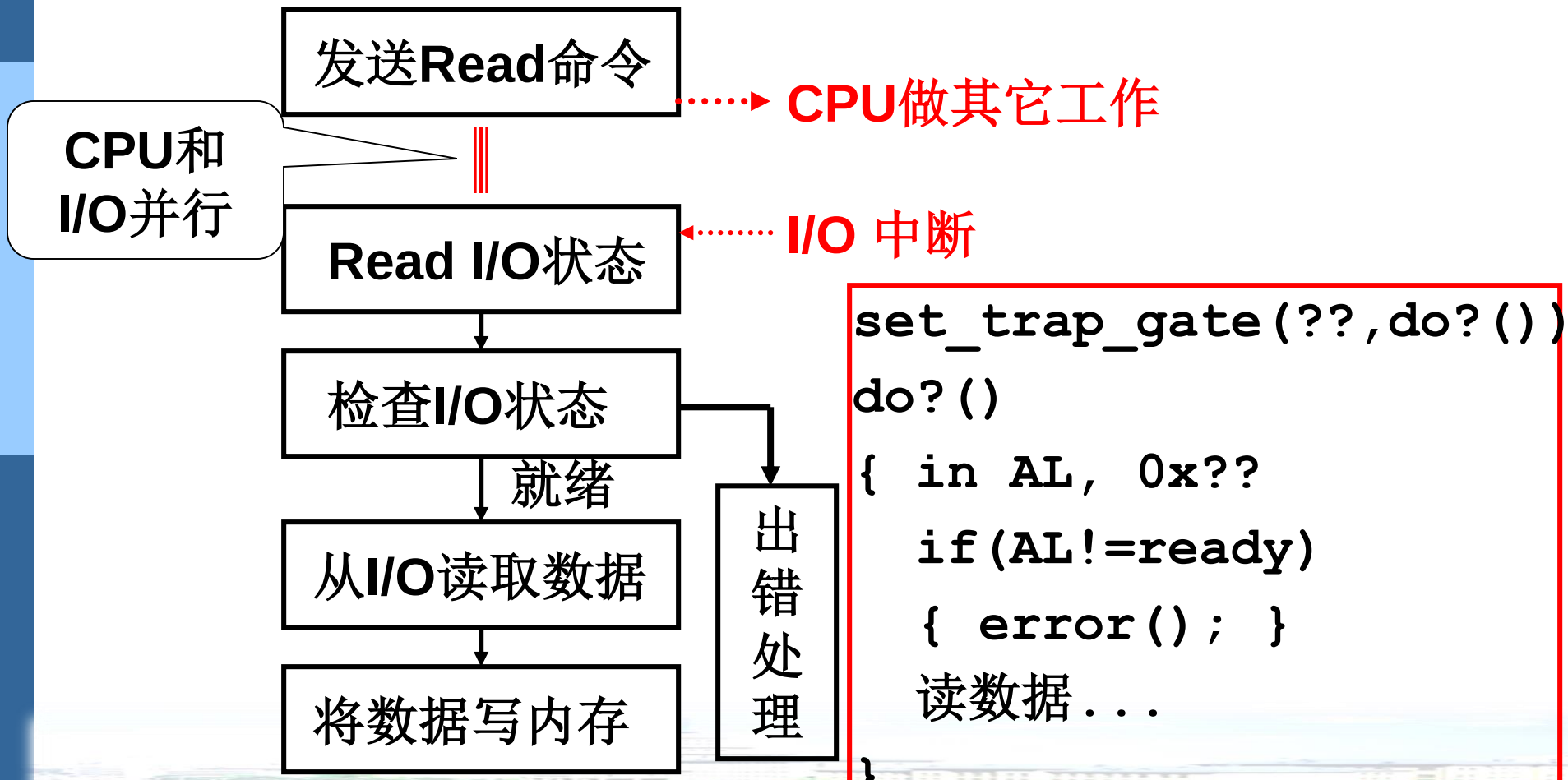
(2) 在设备输入/输出数据期间，CPU不断地循环进行查询设备状态寄存器的值(检查I/O工作是否完成)。

(3) 若完成，对输入操作来说CPU则从数据寄存器中取出数据，然后进行下一次的输入/输出数据或结束。

控制I/O硬件的方式?

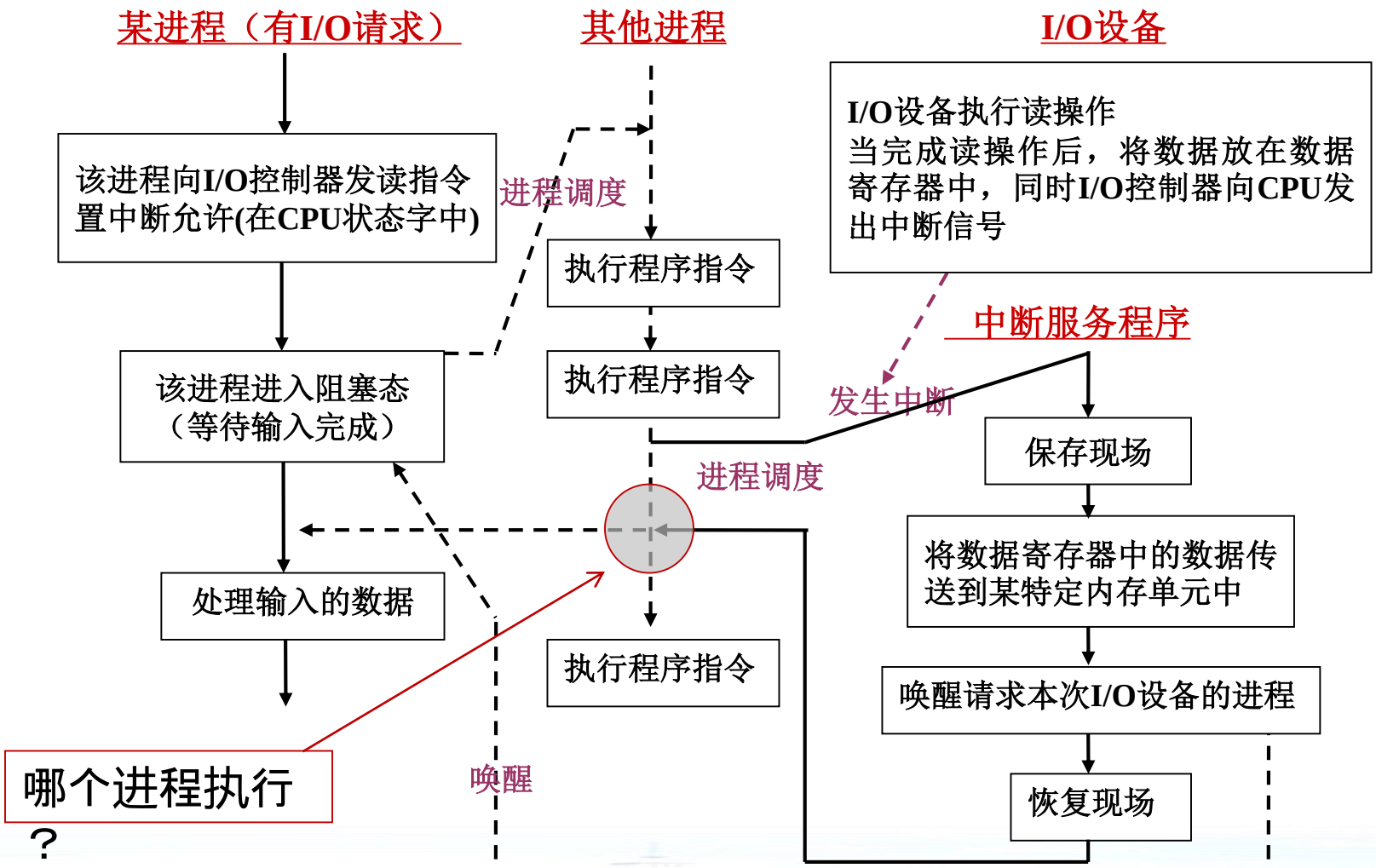
中断是大部分I/O的处理方式!

❿ 方案2: 设备就绪了告诉CPU一声!　　中断



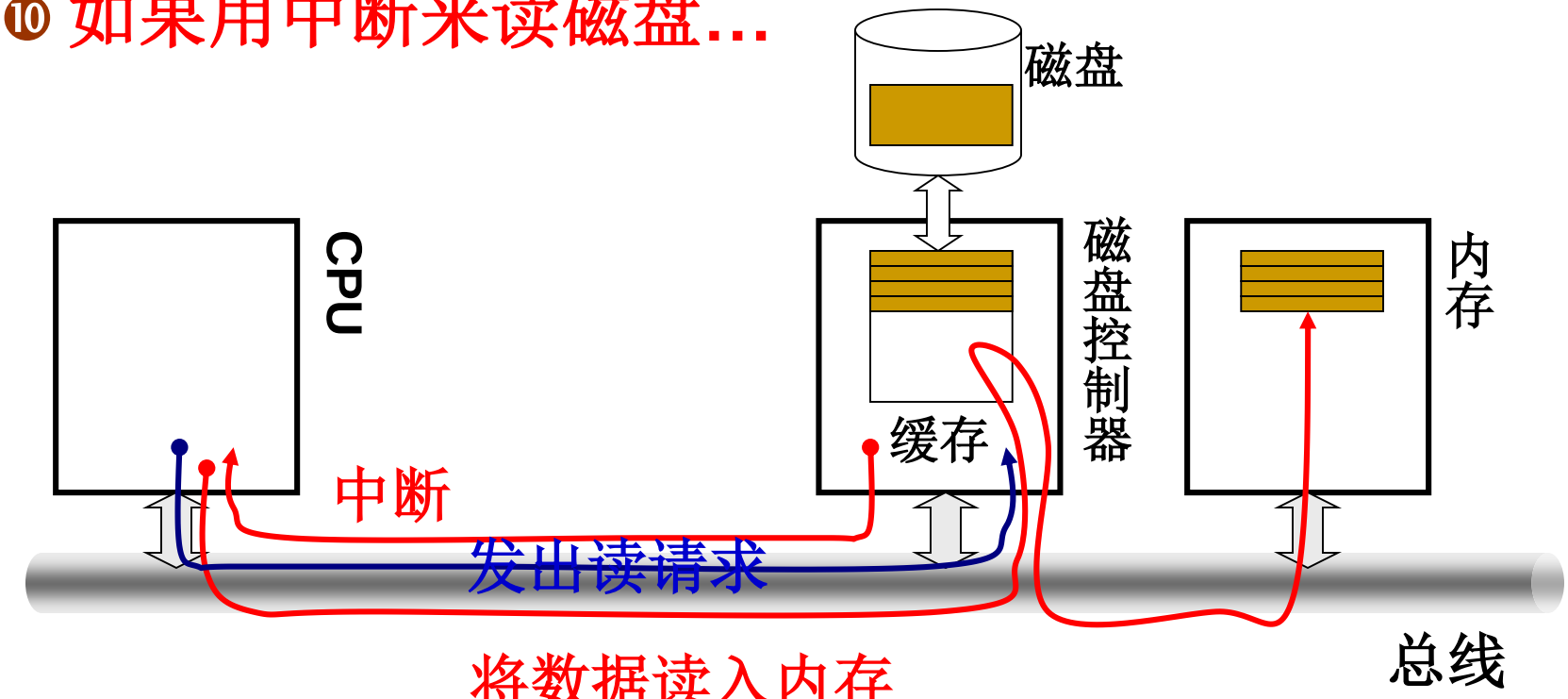

```
set_trap_gate(??,do?())
do?()
{  in AL, 0x??
   if(AL!=ready)
   {  error(); }
   读数据...
}
```

例子：中断方法控制I/O设备读入数据流程



中断在某些场合还不够!

❿ 如果用中断来读磁盘...



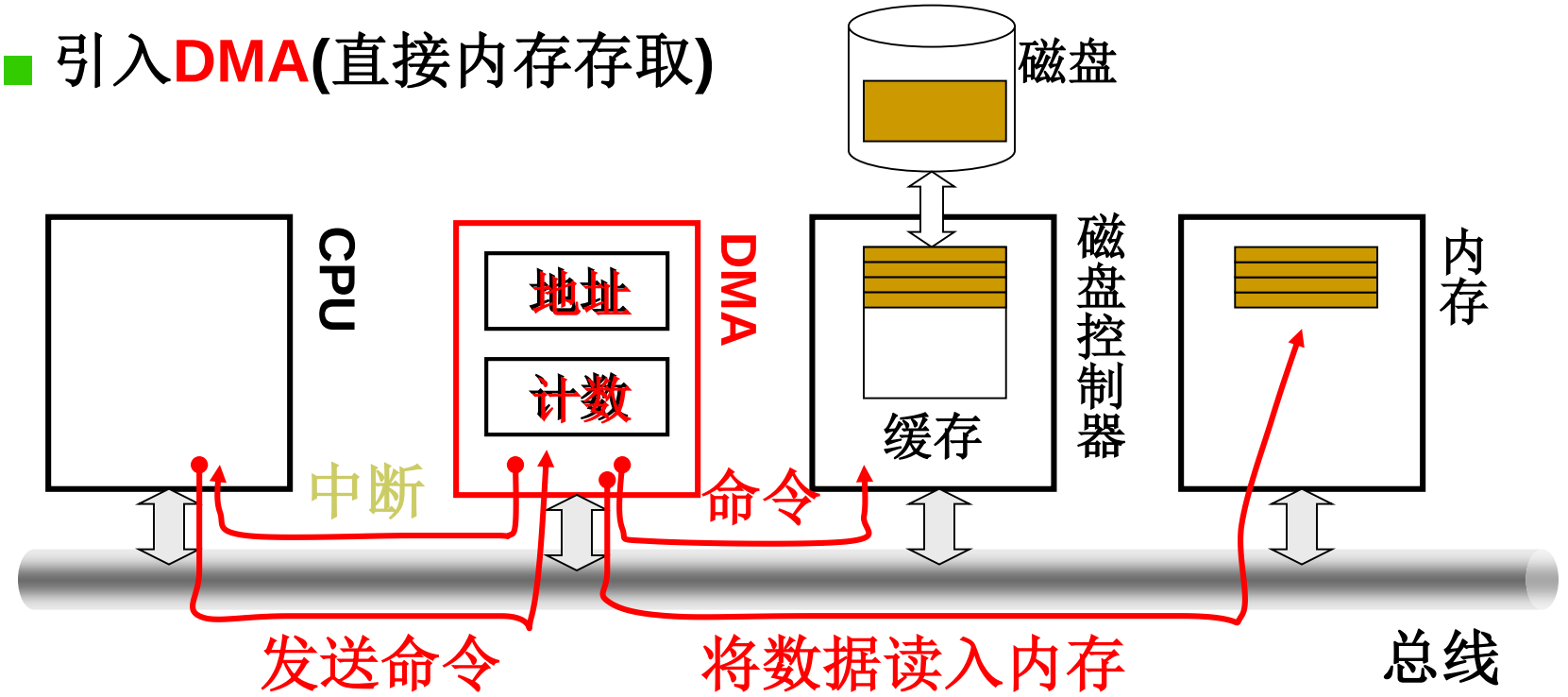
- 每个字节从缓存移动内存都由CPU负责完成

可以设计有一定处理能力的外围设备，
将一些简单任务交给它!

I/O系统发完命令后做什么？

⑩ 方案3: 简单任务自己做，完成了告诉CPU一声!

- 引入DMA(直接内存存取)



- 幸运的是: 该方式的细节由DMA设计者考虑，对于操作系统而言，考虑的仍然只是中断处理

例子:DMA方式数据输入过程

(1)当一个进程要求设备输入数据时，CPU对DMA进行初始化工作:

存放数据的内存起始地址 — DMA控制器的内存地址寄存器;

要输入数据的字节数 — DMA控制器的传送字节数寄存器;

控制字(中断允许、DMA启动位=1) —DMA控制器的控制状态寄存器;

启动位被置1，则启动DMA控制器开始进行数据传输。

(2)该进程放弃CPU，进入阻塞等待状态，等待第一批数据输入完成。

进程调度程序调度其他进程运行。

(3)由DMA控制器控制整个数据的传输。

当输入设备将一个数据送入DMA控制器的数据缓冲寄存器后，DMA控制器立即取代CPU，接管数据地址总线的控制权(CPU工作周期挪用)，将数据送至相应的内存单元;

DMA控制器中的传输字节数寄存器计数减1;

恢复CPU对数据地址总线的控制权;

第(3)步过程循环直到数据传输完毕。

(4)当一批数据输入完成，DMA控制器向CPU发出中断信号，请求中断运行进程并转向执行中断处理程序。

(5)中断程序首先保存被中断进程的现场，唤醒等待输入数据的那个进程，使其变成就绪状态，恢复现场，返回被中断的进程继续执行。

(6)当进程调度程序调度到要求输入数据的那个进程时，该进程就到指定的内存地址中读取数据进行处理。

I/O系统发完命令后做什么？

⑩ 方案4: 可以交办复杂任务，完成后汇报!

- 引入通道（channel）方式
- 通道具有简单的CPU功能，可编程，可管理多个设备同时工作。从而真正实 现了CPU与外部设备的并行工作。

通道控制方式的工作过程：

（1）当一个进程要求输入输出数据时，CPU根据请求形成有关通道程序，然后执行输入输出指令启动通道工作；

（2）申请输入输出数据的进程放弃CPU进入阻塞等待状态，等待数据输入输出工作的完成，于是进程调度程序调度其他进程运行；

（3）通道开始执行CPU放在主存中的通道程序，独立负责外设与主存的数据交换；

（4）当数据交换完成后，通道向CPU发出中断信号，中断正在运行的进程，转向中断处理程序；

（5）中断处理程序首先保护被中断进程的现场，唤醒申请输入输出的那个进程，使其变为就绪状态，关闭通道，然后恢复现场，返回被中断的进程继续运行；

（6）当进程调度程序调度到申请输入输出数据的那个进程时，该进程就到指定的内存地址中进行数据处理。

8.3 缓冲技术

- 缓冲的目的：解决CPU和外设速度不匹配的矛盾，提高CPU与外设之间的并行性，减少对CPU的中断频率
- 缓冲技术的实现方法：硬件缓冲、软件缓冲
  - （1）硬件缓冲：利用专门的硬件寄存器作为缓冲区，一般由外设自带的专用寄存器构成
    例如：Printer、CD-ROM等
  - （2）软件缓冲：借助操作系统的管理，在内存中专门开辟若干单元作为缓冲区

缓冲技术–软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，缓冲池

1.单缓冲：在内存中开辟一个固定大小的区域作为缓冲区

- 外设和CPU交换数据时，先将被交换的数据写入缓冲区，然后再由需要数据的CPU或外设从缓冲区中取出。
- 该方式中，外设与CPU对缓冲区的操作是串行的。

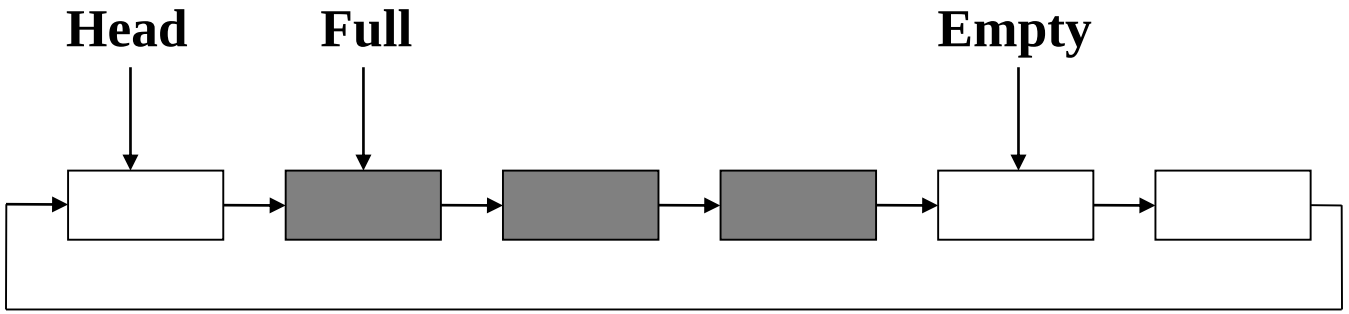
2.双缓冲：在内存中设置2个大小相同的缓冲区。

- 外设和CPU可以交替使用这2个缓冲区，从而在一定程度上实现并行交换数据。

缓冲技术软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，缓冲池

3.环形缓冲：在内存中设置大小相等的多个缓冲区，并将它们链接称为一个环形链表。



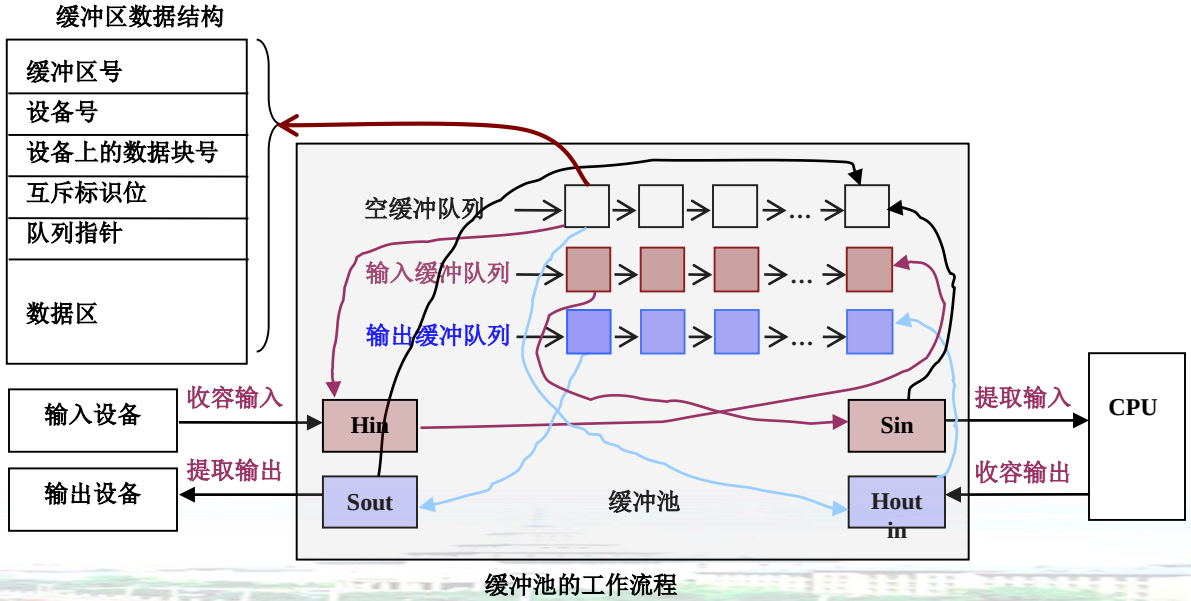
- Head一直指向缓冲区链表的第一个缓冲区；
- Full一直指向缓冲区链表中的第一个存满数据的缓冲区；
- Empty一直指向缓冲区链表中的第一个空白的缓冲区。
- 初始化时：Head=Full=Empty，整个缓冲区链表为空；
- 使用过程中：当Full=Empty <==> 整个缓冲区链表为空。

缓冲技术软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，缓冲池

4.缓冲池： 缓冲池是有多个大小相同的缓冲区组成

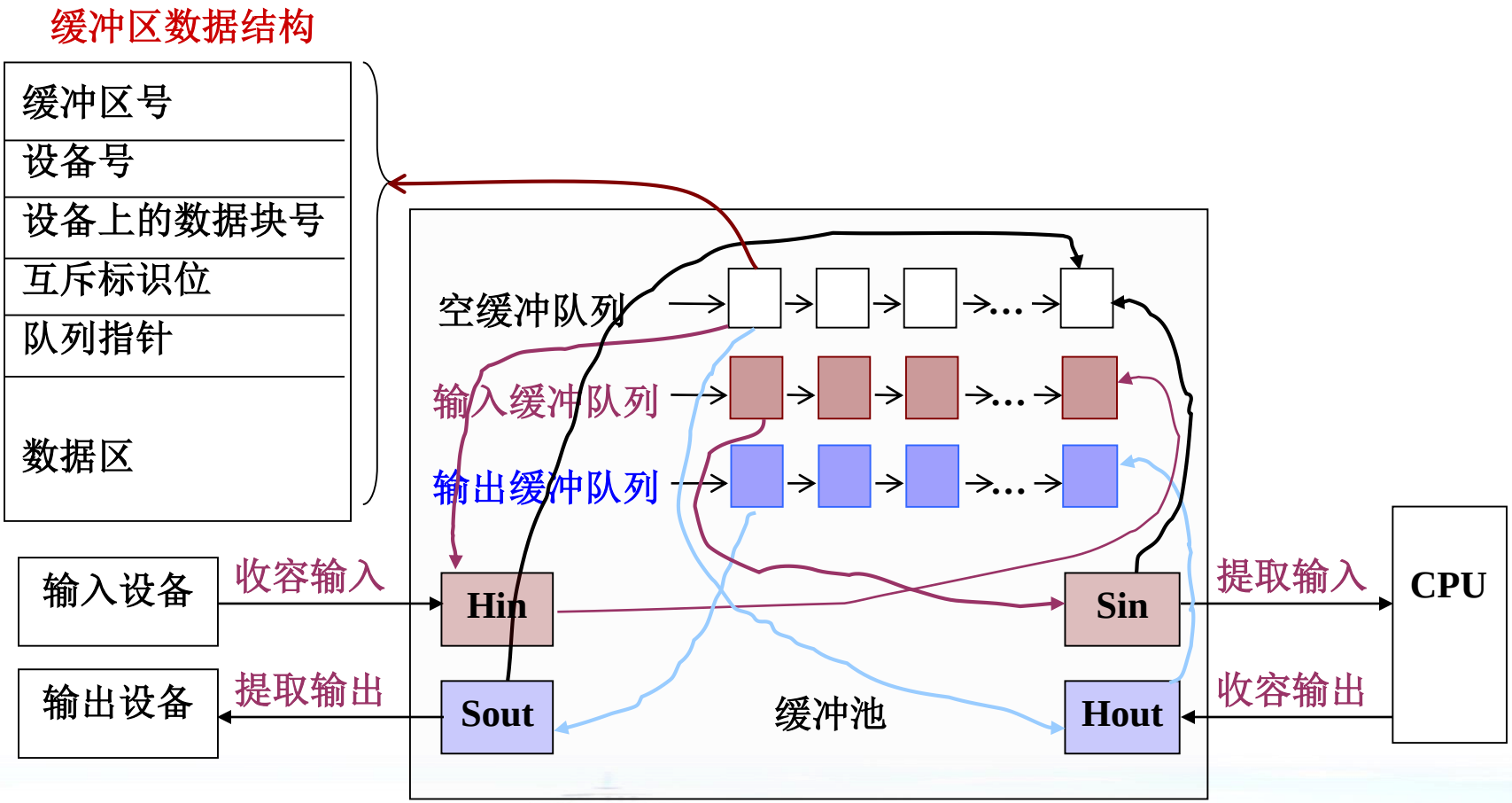
- 池中的缓冲区是系统公共资源，所有进程均可以共享
- 池由系统管理程序统一管理，负责分配、回收工作
- 池中每个缓冲区既可以用于输入数据，也可以用以输出数据



缓冲池的工作流程

缓冲技术软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，缓冲池



缓冲池的工作流程

缓冲技术软件缓冲的4种实现方法

- 单缓冲，双缓冲，环形缓冲，缓冲池

缓冲池的工作流程：

（1）当输入设备需要进行数据输入时，则从空缓冲队列的队首取下一个空缓冲区，将它作为收容输入工作缓冲区，当它被输入装满数据后，则被链接到输入缓冲队列的队尾；

（2）当某进程需要从缓冲池输入数据时，则从输入缓冲队列的队首取一个缓冲区作为提取输入工作缓冲区，该进程从中提取数据，取完后，则将该缓冲区链接到空缓冲区队列的队尾；

（3）当某进程需要输出数据到缓冲池时，则从空缓冲队列的队首取下一个空缓冲区，将它作为收容输出工作缓冲区，该进程向该缓冲区中存放数据，当它被装满数据后，则被链接到输出缓冲队列的队尾；

（4）当输出设备需要进行数据输出时，则从输出缓冲队列的队首取一个缓冲区作为提取输出工作缓冲区，并从中提取数据输出，取完后，则将该缓冲区链接到空缓冲区队列的队尾。

缓冲技术软件缓冲的4种实现方法

- Linux系统为了提高读写磁盘的效率，会先将数据放在一块buffer中。
- 在写磁盘时并不是立即将数据写到磁盘中，而是先写入这块buffer中了。
- 此时如果重启系统，就可能造成数据丢失。
- sync命令用来flush文件系统buffer，这样数据才会真正的写到磁盘中，并且buffer才能够释放出来。
- sync命令会强制将数据写入磁盘中，并释放该数据对应的buffer，
- 所以常常会在写磁盘后输入sync命令来将数据真正的写入磁盘。

缓冲技术软件缓冲的4种实现方法

如果不去手动的输入sync命令来真正的去写磁盘，linux系统也会有两种写磁盘的时机：

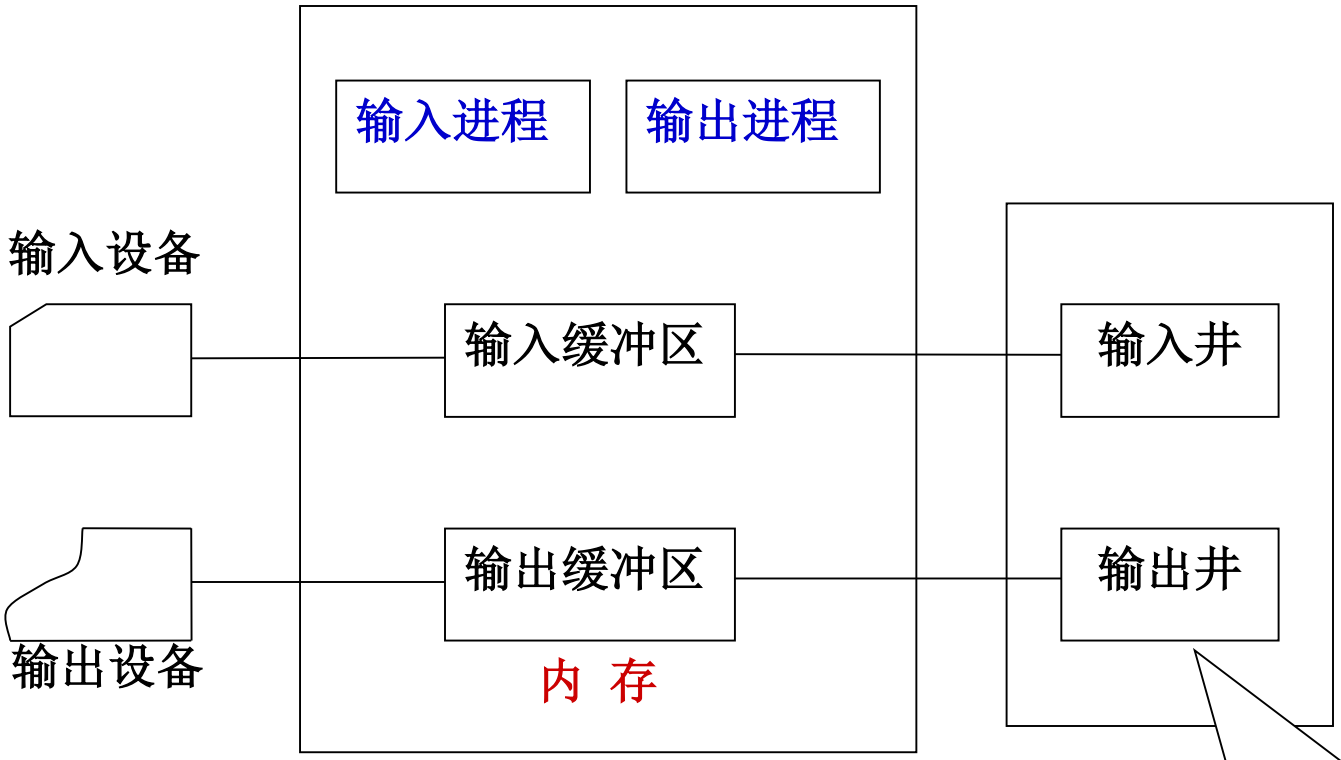
1) kflush内核线程周期性的去写磁盘；

2) buffer已满不得不写。

8.3 缓冲技术 – SPOOLING

- SPOOL – Simultaneous Peripheral Operation On Line外部设备同时联机操作，又称假脱机操作。
- SPOOL是操作系统中采用的一项将独占设备改造成共享设备的技术。
- 实现方法：截获向某独享设备输出的数据，暂时保存到内存缓冲区或磁盘文件中，并进行排队，之后逐个输出到外设上
- 实现这一技术的软、硬件系统称为SPOOL系统，或假脱机系统，或SPOOLING系统。

8.3 缓冲技术 – SPOOLING



SPOOLING系统的组成

输入井和输出井：

- 在磁盘上开辟出来的2个存储区域
- 输入井用于收容I/O设备的输入数据
- 输出井用于收容I/O设备的输出数据

总结一些I/O系统要完成的工作!

```
write(buf, 10);
```

OS需要提供系统调用接口

```
DMA.addr = buf;
DMA.count = 10;
……
sleep_on(Disk);
```

查一下手册就可以找到该写什么命令?该向哪里写?

让出CPU?

需要写中断处理程序!

```
do_write_end()//中断处理
{
    wakeup(Disk);
}
```

- 总的感觉: 很简单

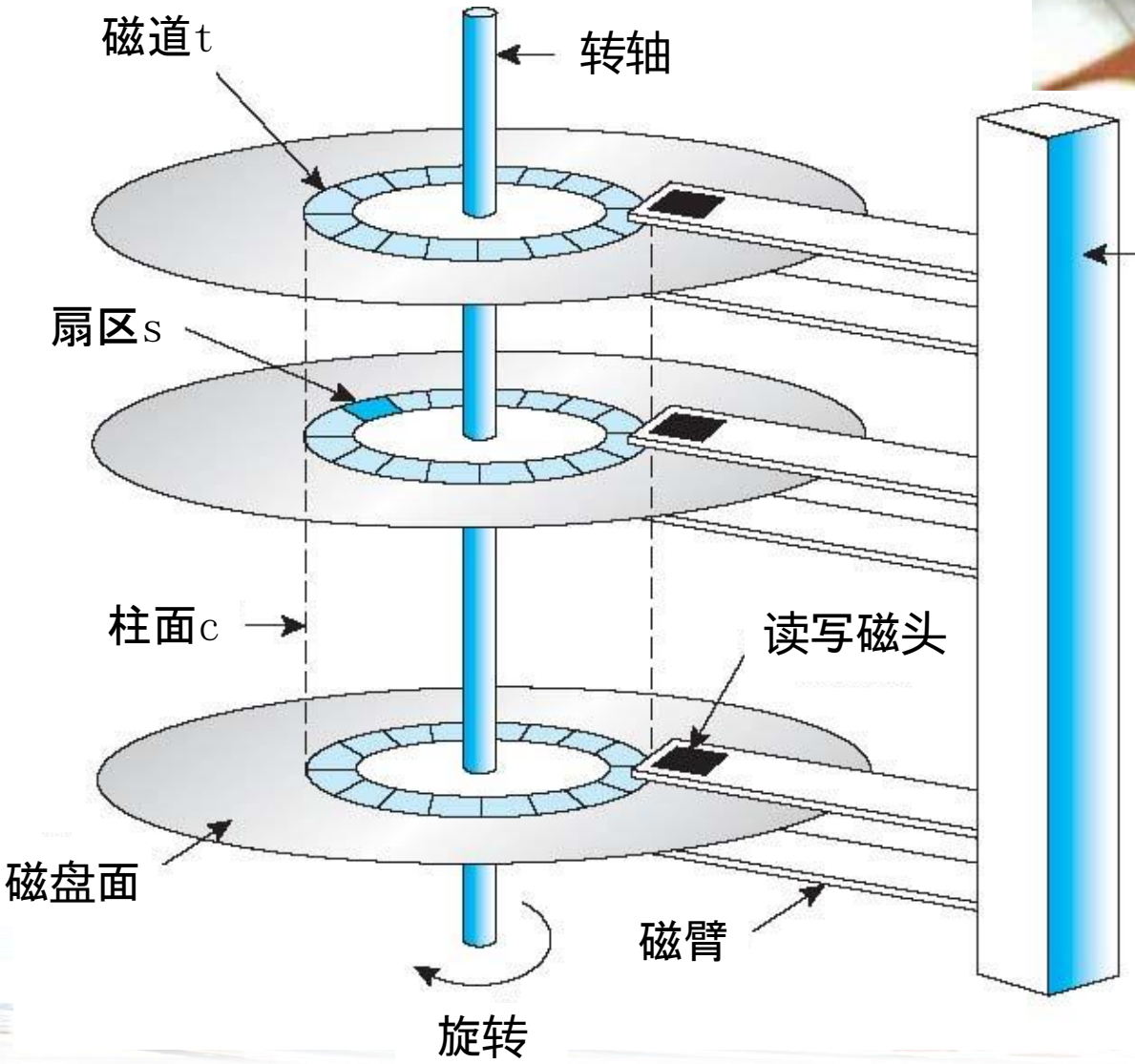
  处理流程是很简单，复杂的是一些细节问题。

I/O设备管理总结

- 如何实现交互? ⇒ 首先需要了解I/O的工作原理
- 从用户如何I/O开始 ⇒ 用户发送一个命令(read)
- 系统调用read ⇒ 被展开成给一些寄存器发送命令的代码
- 发送完命令以后… ⇒ CPU轮询，CPU干其它事情并等中断
- 中断方案最常见 ⇒ 相比其他设备，CPU太快了
- 实现独享设备的共享 ⇒ 假脱机系统（SPOOLING）

认识一下磁盘

认识一下磁盘



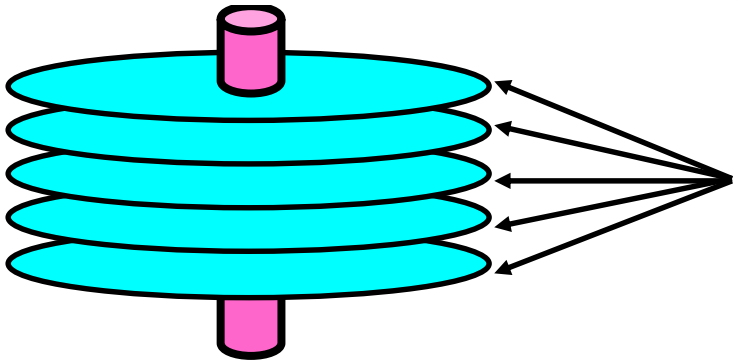
机械臂杆

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。

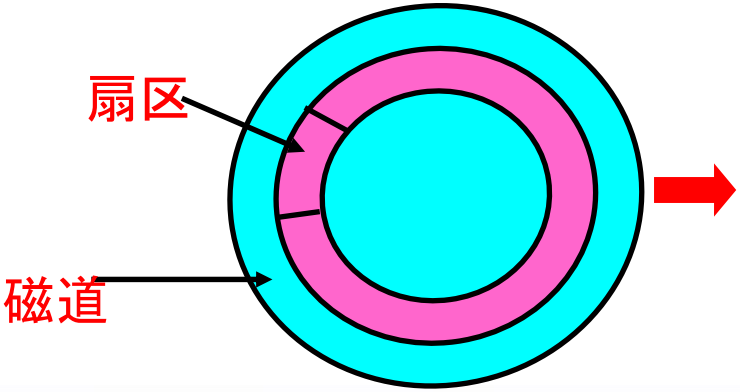
现在的水平已经达到0.005μm～0.01μm，这只是人类头发直径的千分之一。

认识一下磁盘

- 画一个示意图：



- 看看俯视图：



- 所以，磁盘被称为块设备！

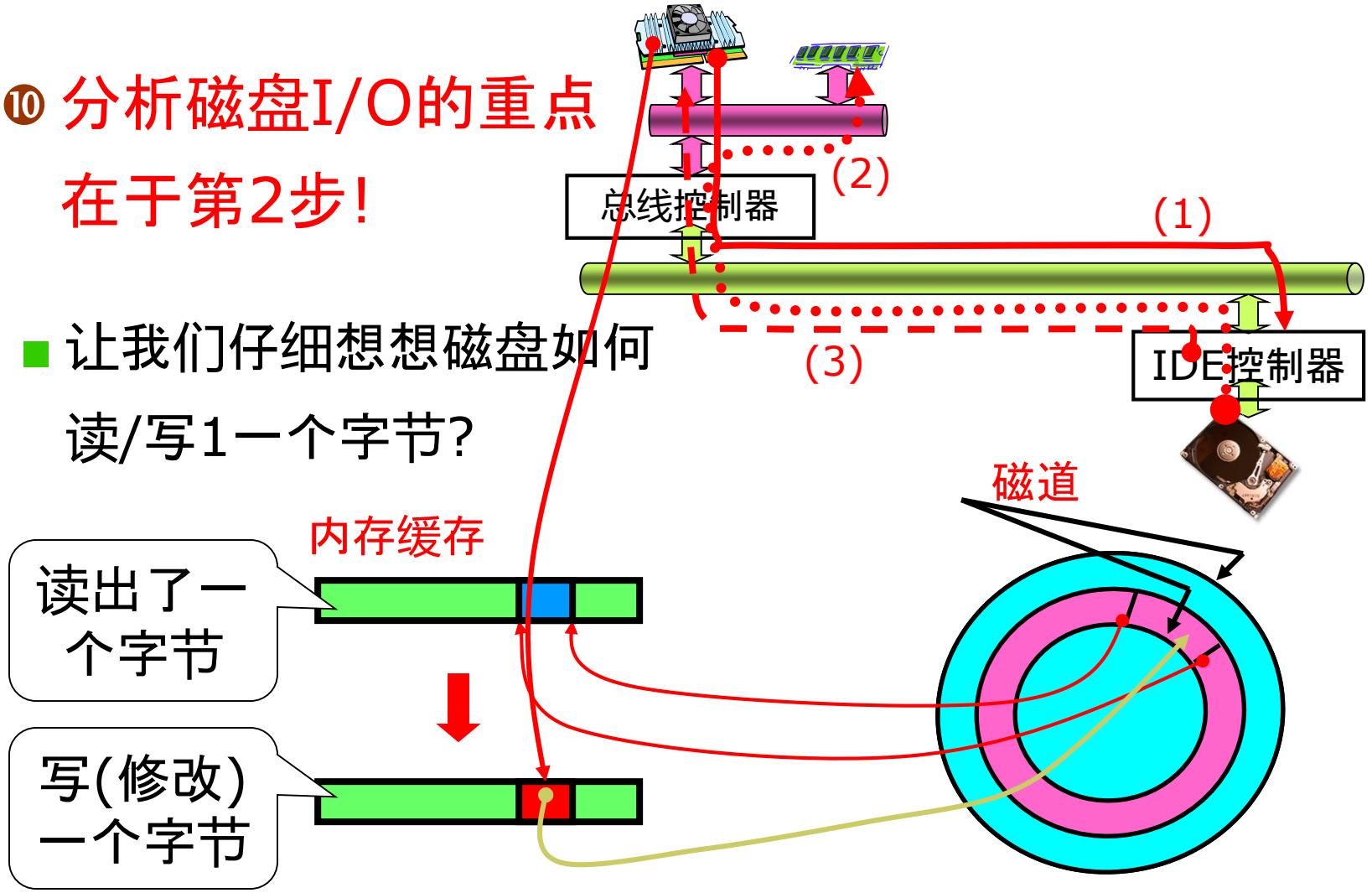
扇区是磁盘的寻址单位、访问单位

- 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

⑩ 分析磁盘I/O的重点在于第2步!

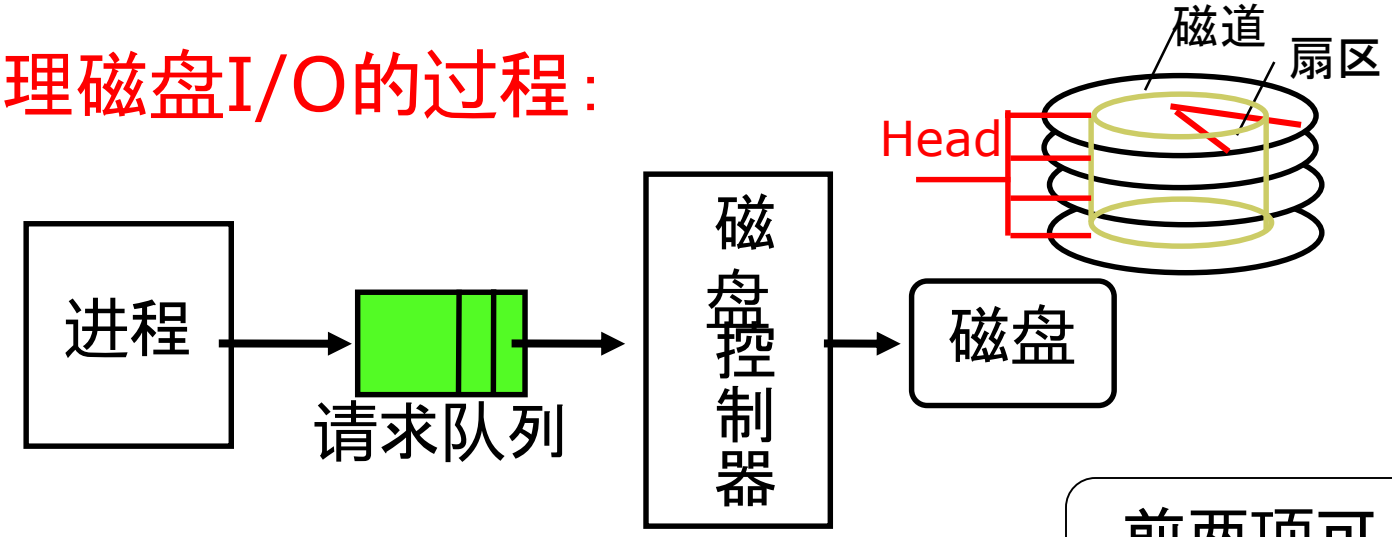
- 让我们仔细想想磁盘如何读/写1一个字节?



- 磁盘I/O: 缓存队列 → 控制器 →寻道 →旋转 → 传输!

磁盘I/O的分析

⑩ 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

8ms to 12 ms

(半周)：4 ms to 8 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

⑩ 磁盘调度：

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

8 ms to 12 ms

4 ms to 8 ms

约0.25ms

- 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！

- 磁盘调度：输入多个磁道请求，给出服务顺序！

随堂习题

- 设一个磁盘的平均寻道时间为12ms，传输速率是200MB/s，控制器开销是0.2ms，转速为每分钟5400转。求读写一个512KB大小数据块的平均磁盘访问时间？

- 平均旋转延时 = 0.5/5400转/分 = 0.0056秒 = 5.6ms
- 平均磁盘访问时间 = 平均寻道时间 + 平均旋转延时 + 传输时间 + 控制器延时 = 12ms + 5.6ms + 512KB/200MB/s + 0.2ms = (12 + 5.6 + 2.5 + 0.2)ms = 20.3ms。

FCFS磁盘调度

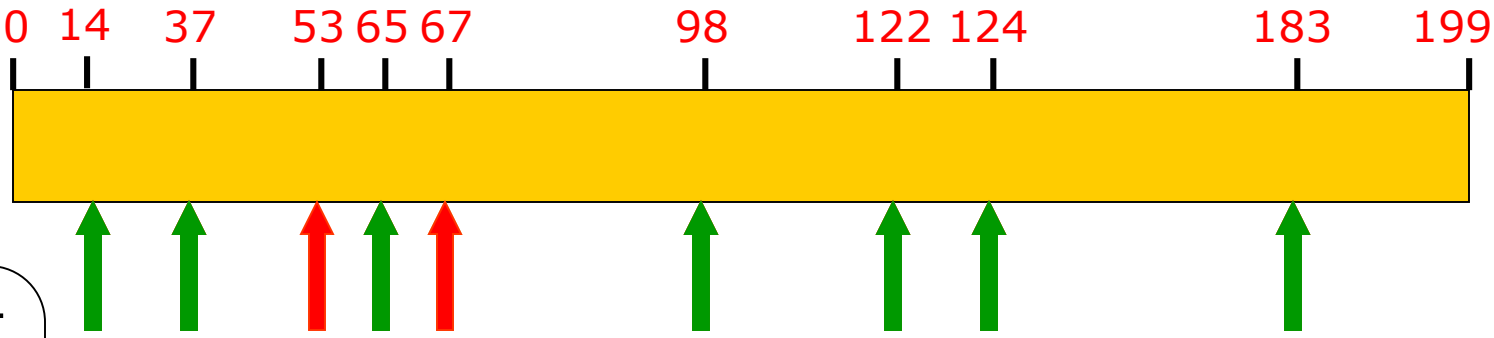
130+146+85+108+110+59+2=640

⑩ 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

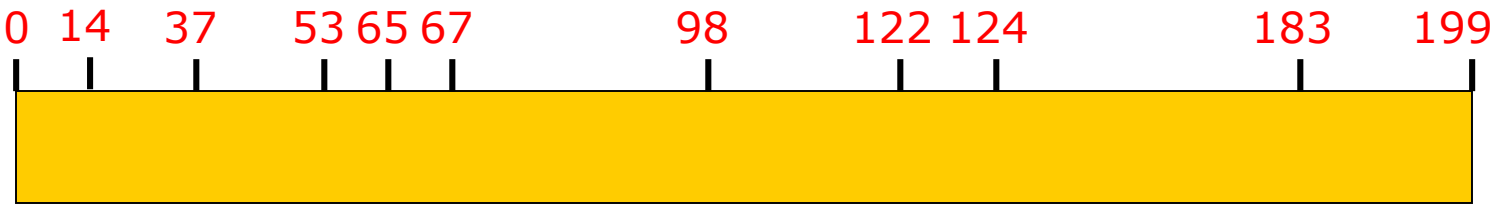
在移动过程中把经过的请求处理了?!

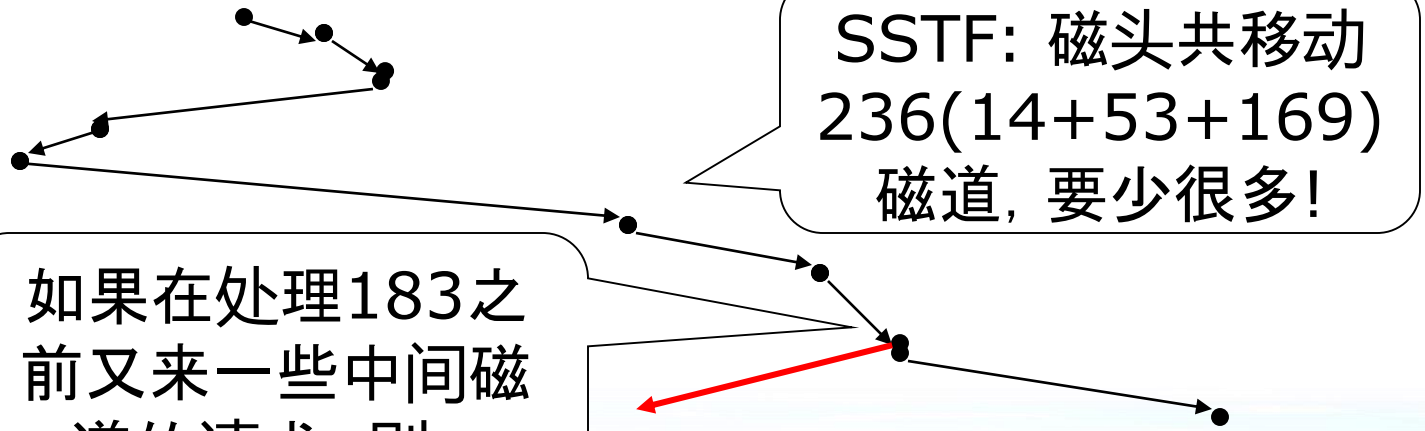
磁头在长途奔袭!

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





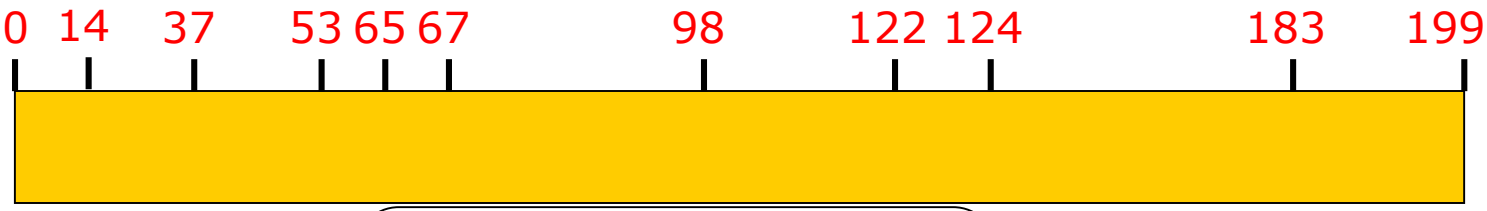
  - SSTF存在饥饿问题

SCAN磁盘调度(扫描/电梯算法)

⑩ 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：磁头共移动53+183=236磁道，和SSTF一样！

这些请求的等待时间较长，只因所在方向不够幸运！

根据其特征，SCAN也被称为电梯算法！

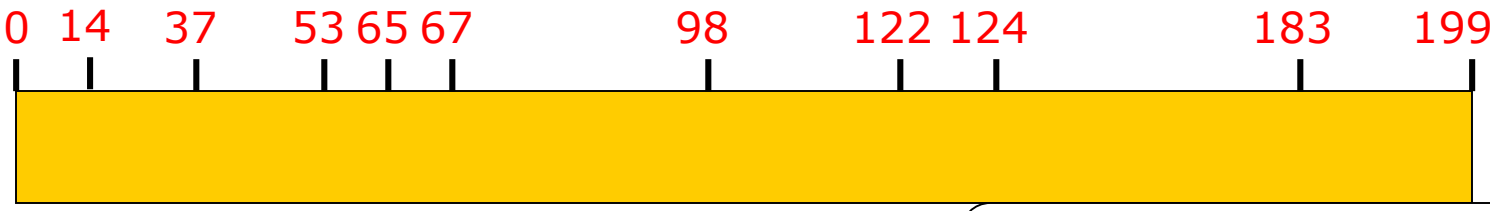
- SCAN导致延迟不均

C-SCAN磁盘调度

⑩ 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！

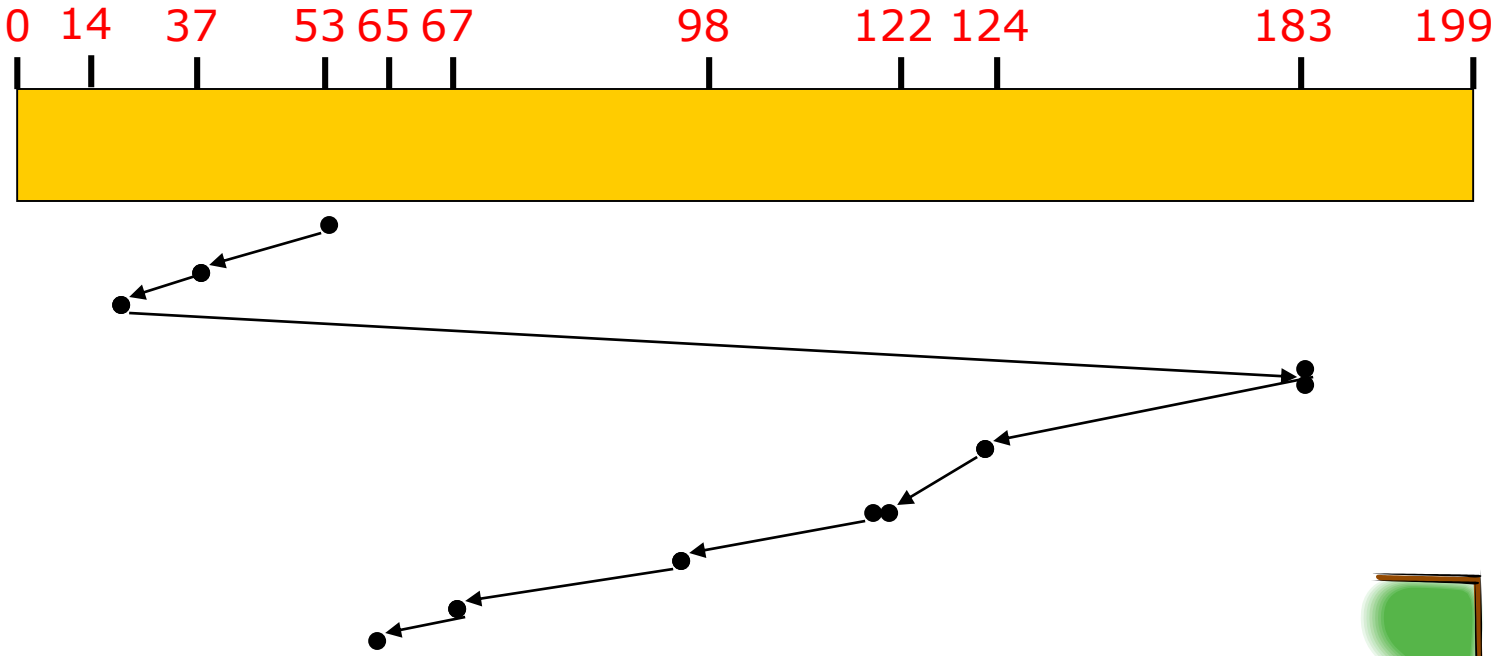
- 14→0(183→199)没有必要

C-LOOK磁盘调度

⑩ 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



- LOOK和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!

磁盘调度算法的比较

	优点	缺点
FCFS算法	公平、简单	平均寻道距离大，仅应用在磁盘I/O较少的场合
SSTF算法	性能比“先来先服务”好	不能保证平均寻道时间最短，可能出现“饥饿”现象
SCAN算法	寻道性能较好，可避免“饥饿”现象	不利于远离磁头一端的访问请求
C-SCAN算法	消除了对两端磁道请求的不公平	--

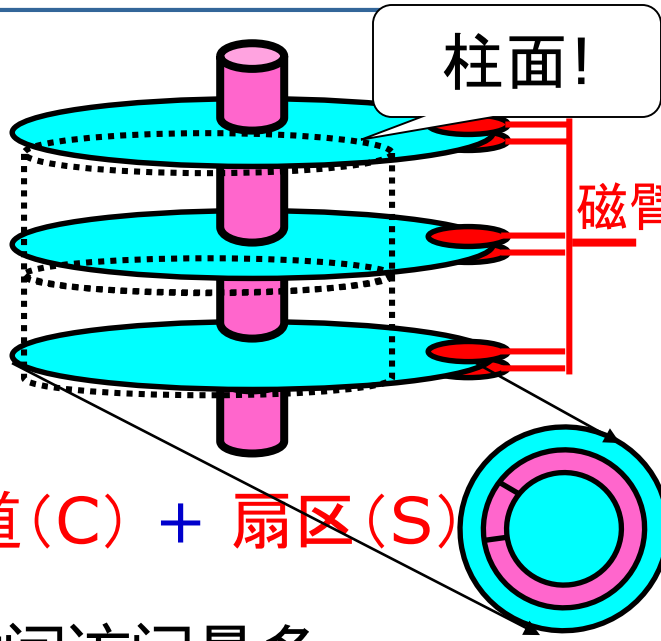
磁盘编址

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扇区号码和纠错码等信息

什么是磁盘的逻辑格式化？
第9章 文件系统！

I/O过程是解开许多磁盘问题的钥匙



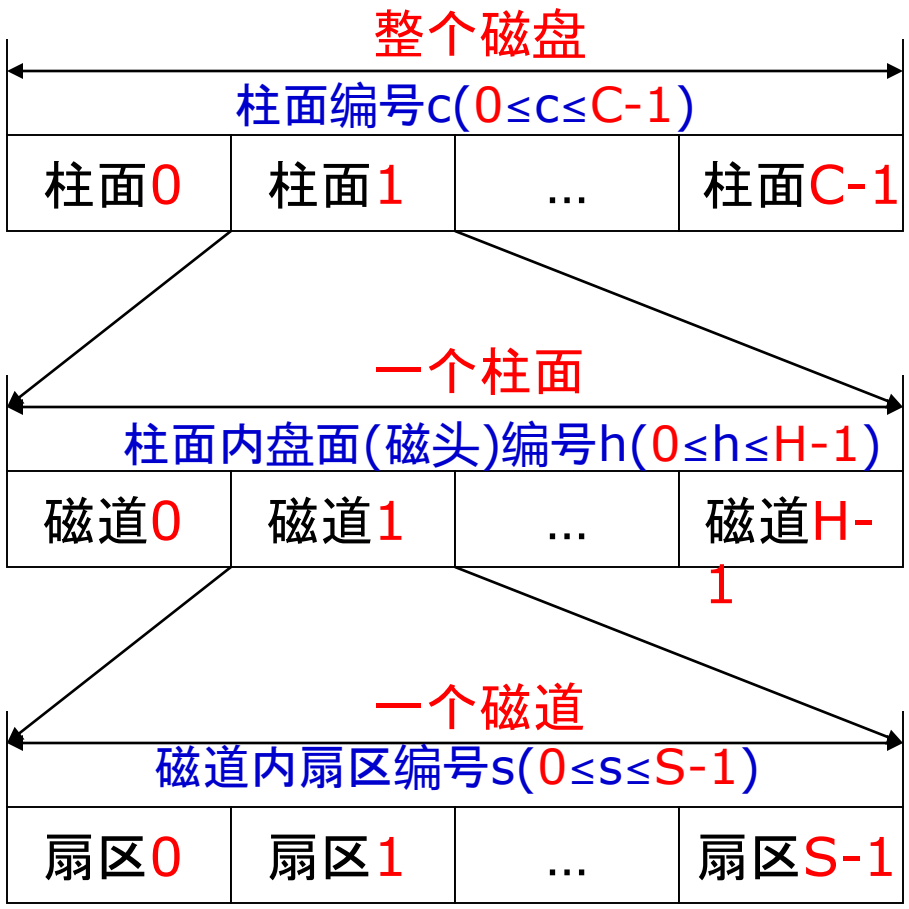
⑩ 磁盘寻址：对于内存，我们往往更关心存放内容的地址

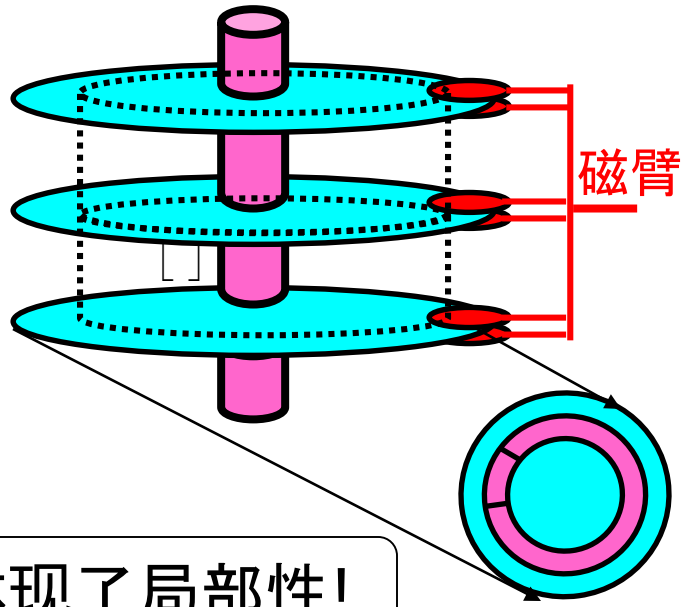
- 实际上就是扇区怎么编址？
- 显然这个地址是(盘面（H） + 磁道（C） + 扇区（S）)
- 寻道和旋转费时多 ⇒ 花最少时间访问最多扇区的方案：磁臂不动、磁盘旋转一周，访问磁头遇到的所有扇区。

让这些扇区的编址邻近：因为局部性!

- 扇区编址(1)：CHS(Cylinder/Head/Sector)
- 扇区编址(2)：扇区编号（Logical Block Addressing LBA）

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.
- 也就是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).：
- 磁柱面数(Cylinders),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，寻址方式也改为线性寻址，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

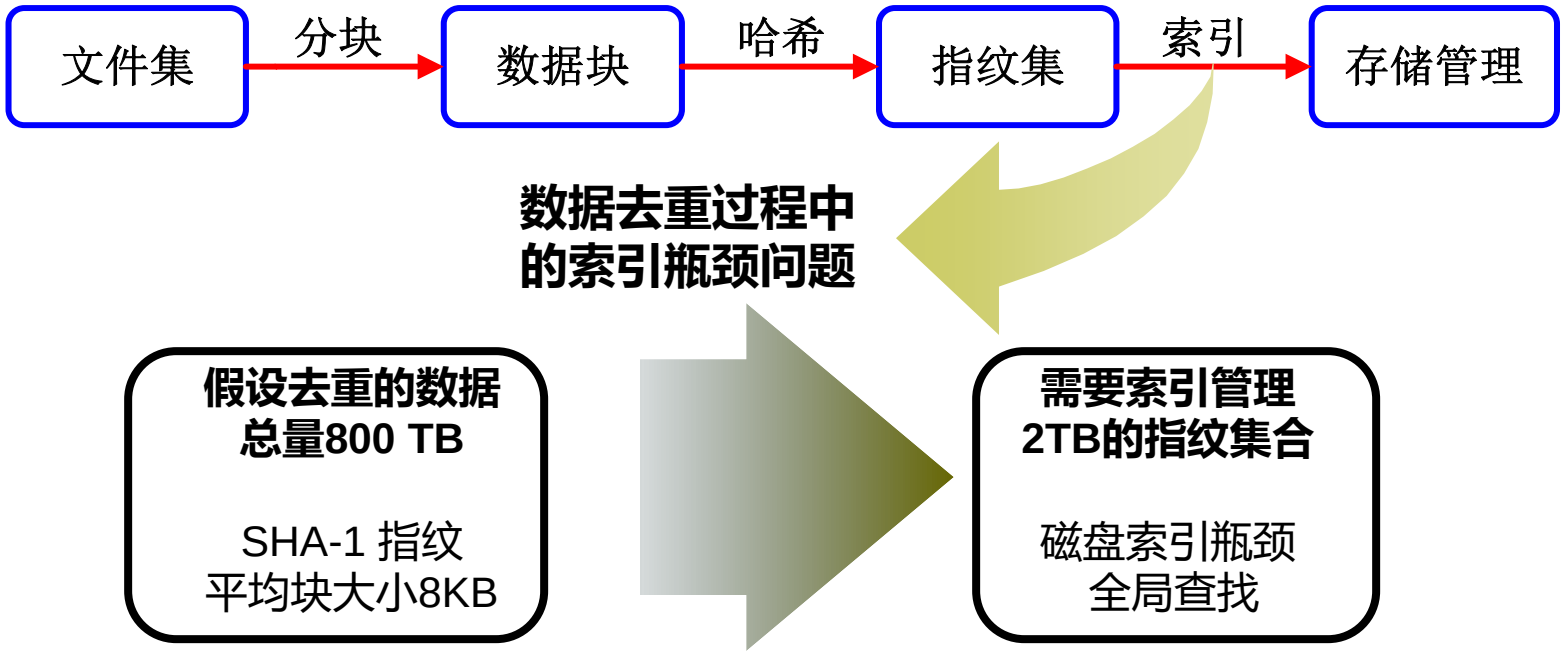
虚拟内存：按需调页与页面置换。

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。

回顾：拓展阅读之磁盘索引瓶颈问题

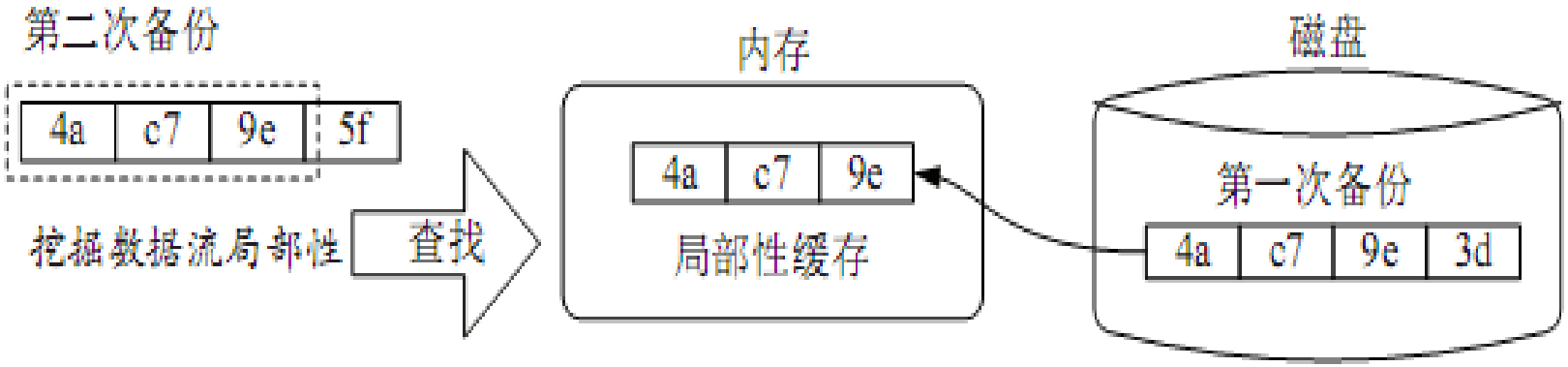


磁盘索引瓶颈问题

Venti使用了磁盘指纹索引方案，其系统冗余数据消除吞吐率仅仅为6.5MB/s。

但是高吞吐率是企业数据保护的关键指标，其备份任务通常得超过100MB/s。

回顾:DDFS解决方案



内存开销(SHA1)？

1G个数据块→8TB数据→BF开销1GB

索引性能？

局部性+BF(布隆过滤器)

减少了99%的磁盘I/O

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

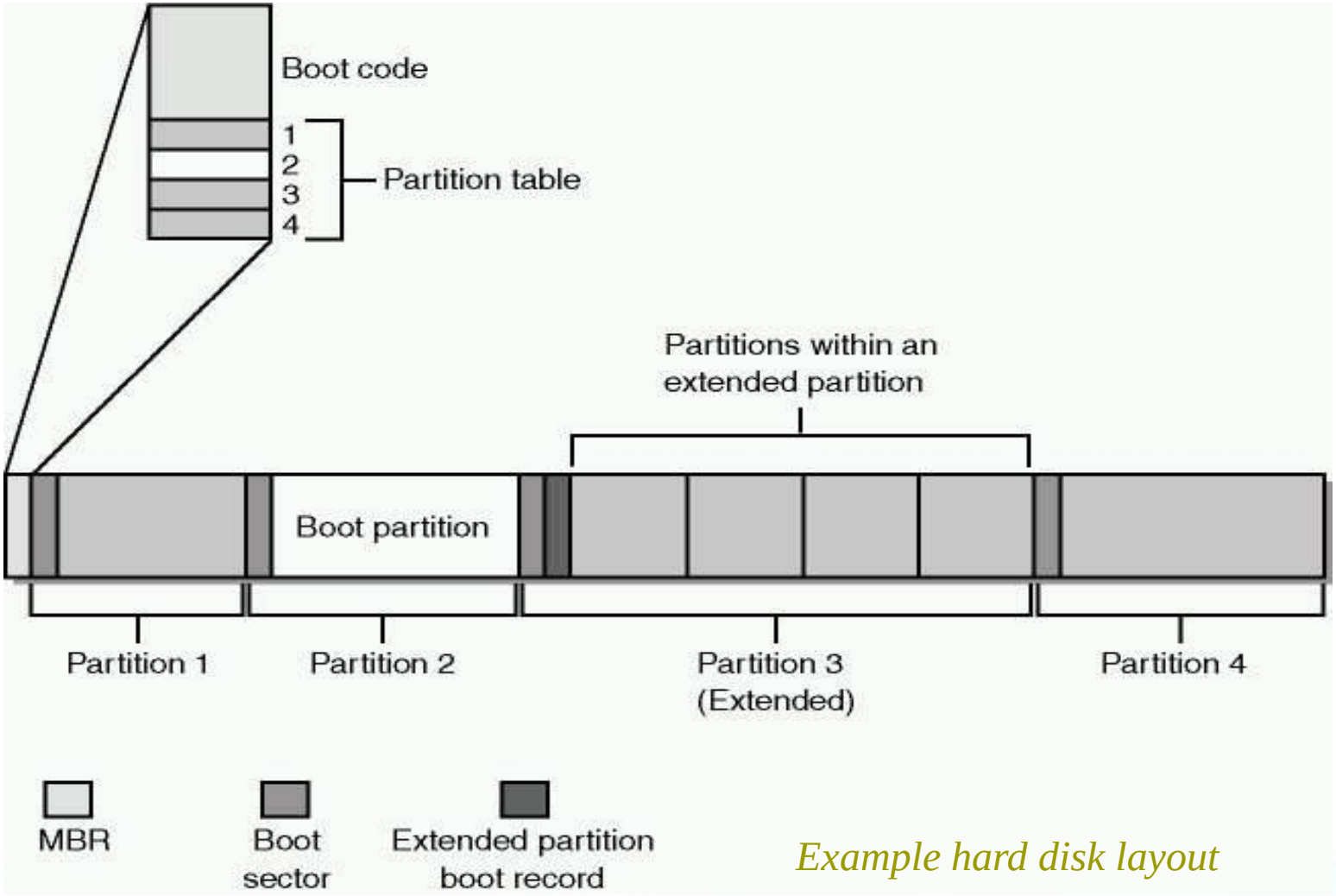
查手册、写端口!

- 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!

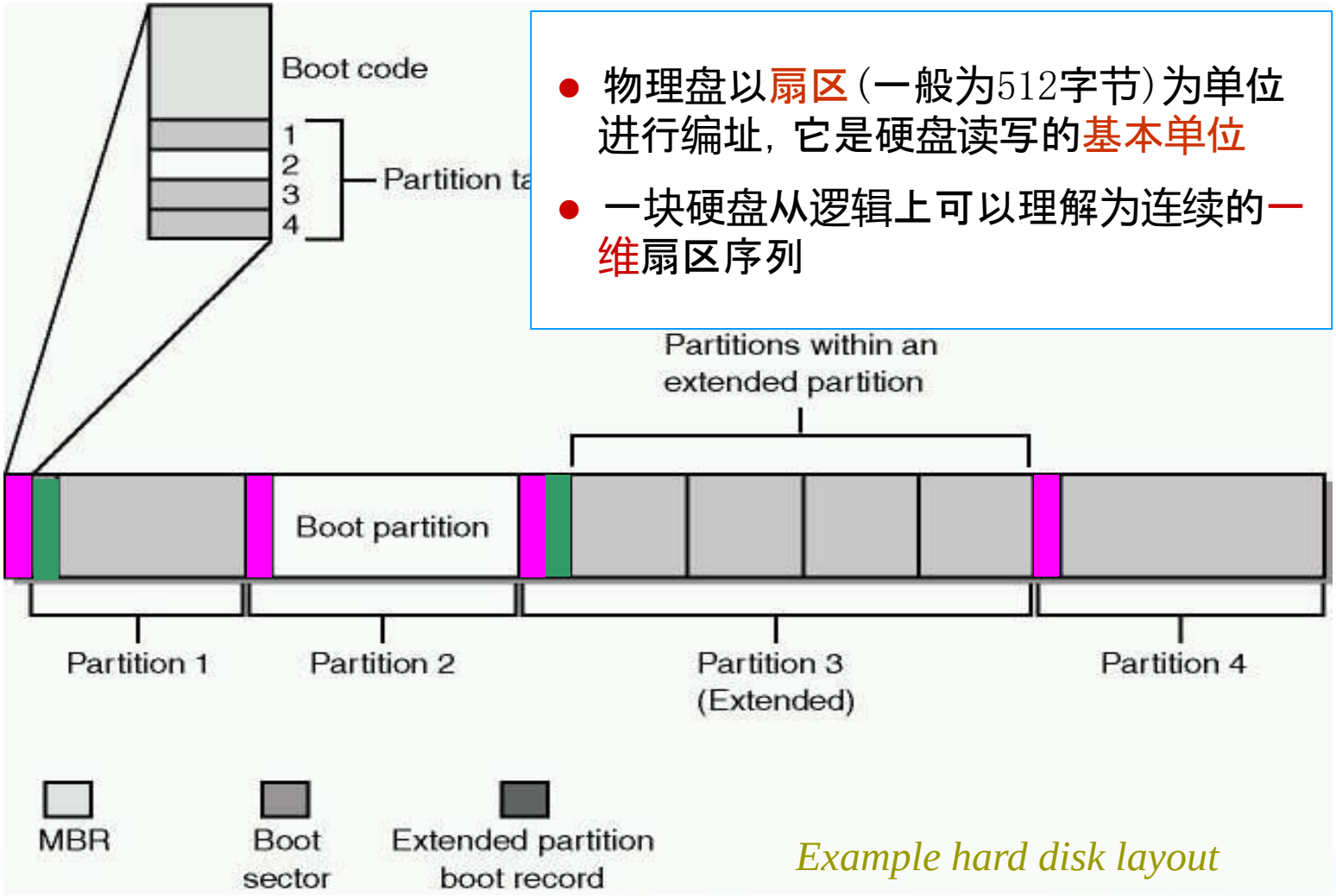
- 第4步：进程继续...

硬盘布局



Example hard disk layout

硬盘布局

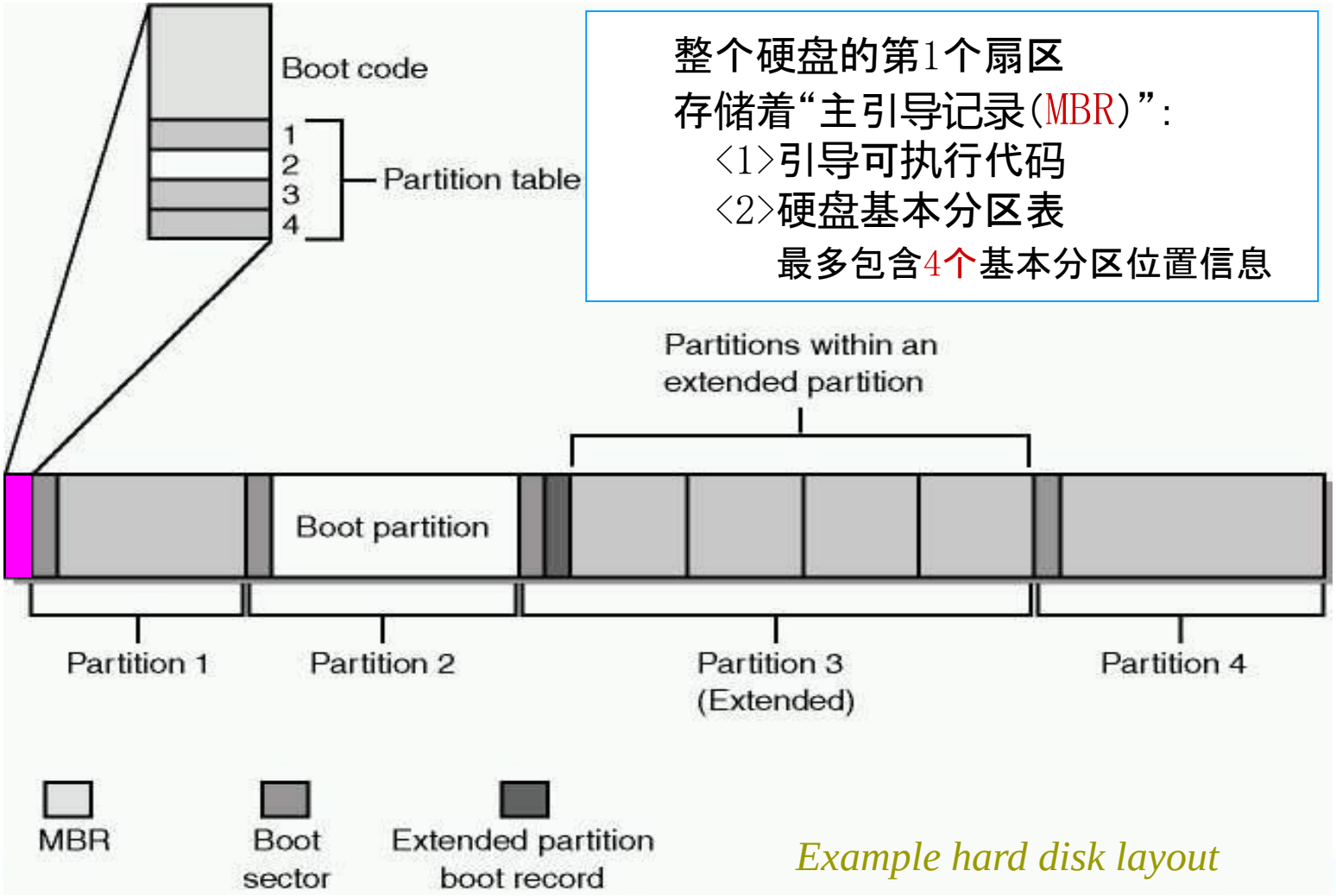


- 物理盘以扇区(一般为512字节)为单位进行编址，它是硬盘读写的基本单位
- 一块硬盘从逻辑上可以理解为连续的一维扇区序列

Example hard disk layout

硬盘布局

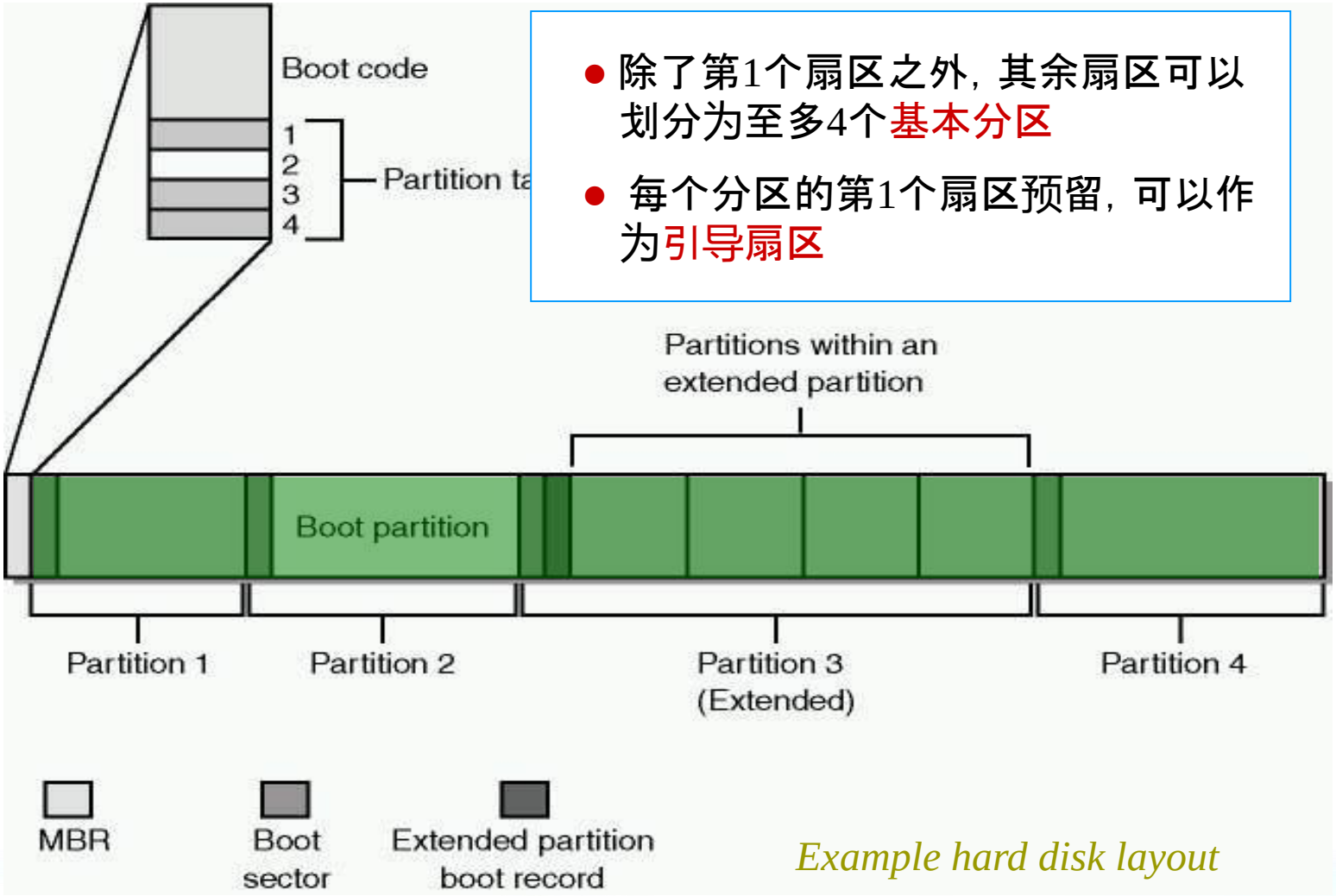
整个硬盘的第1个扇区
存储着“主引导记录(MBR)”:
〈1〉引导可执行代码
〈2〉硬盘基本分区表
最多包含4个基本分区位置信息



Example hard disk layout

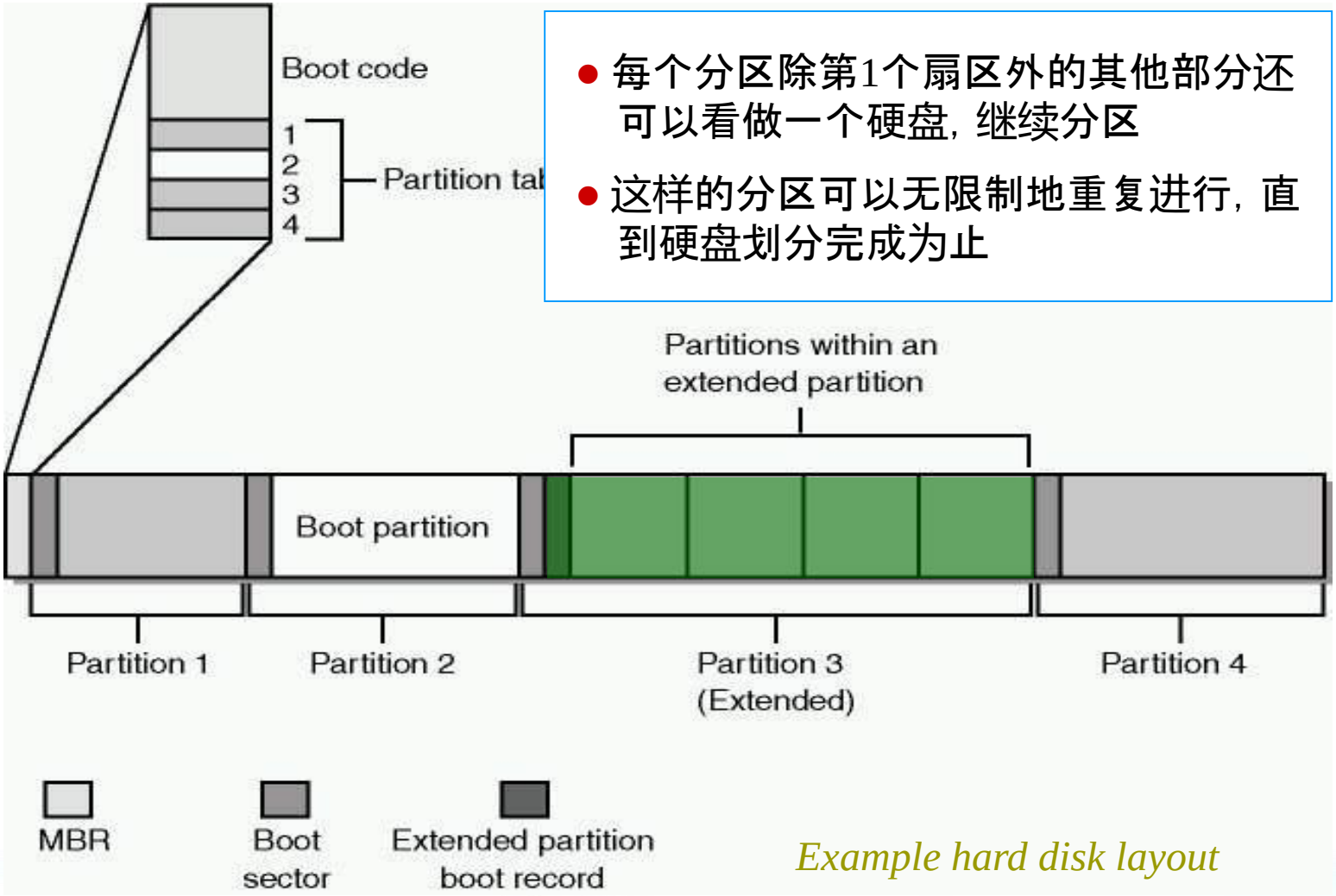
硬盘布局

- 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
- 每个分区的第1个扇区预留，可以作为引导扇区



Example hard disk layout

硬盘布局



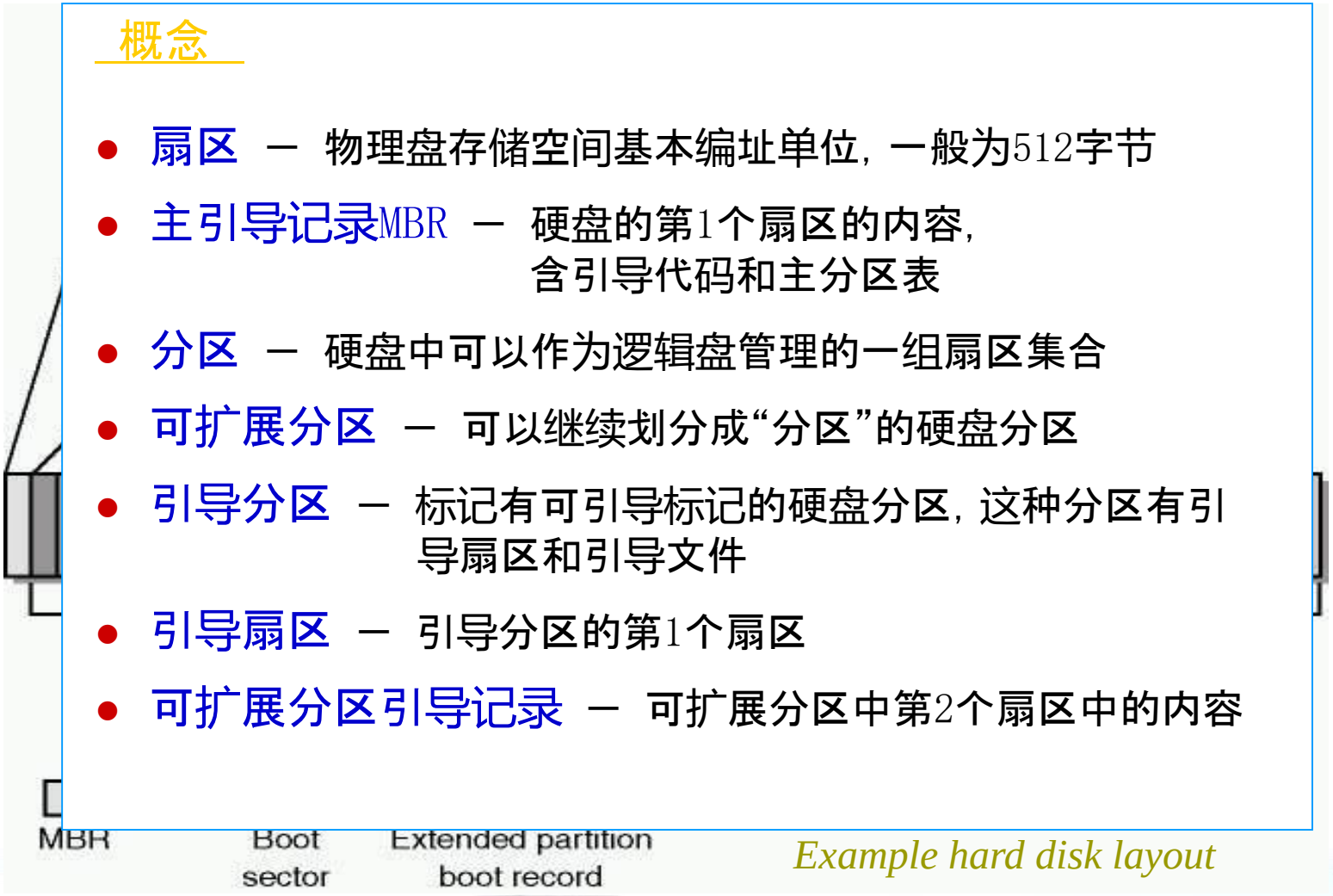
- 每个分区除第1个扇区外的其他部分还可以看做一个硬盘，继续分区
- 这样的分区可以无限制地重复进行，直到硬盘划分完成为止

Example hard disk layout

硬盘布局

概念

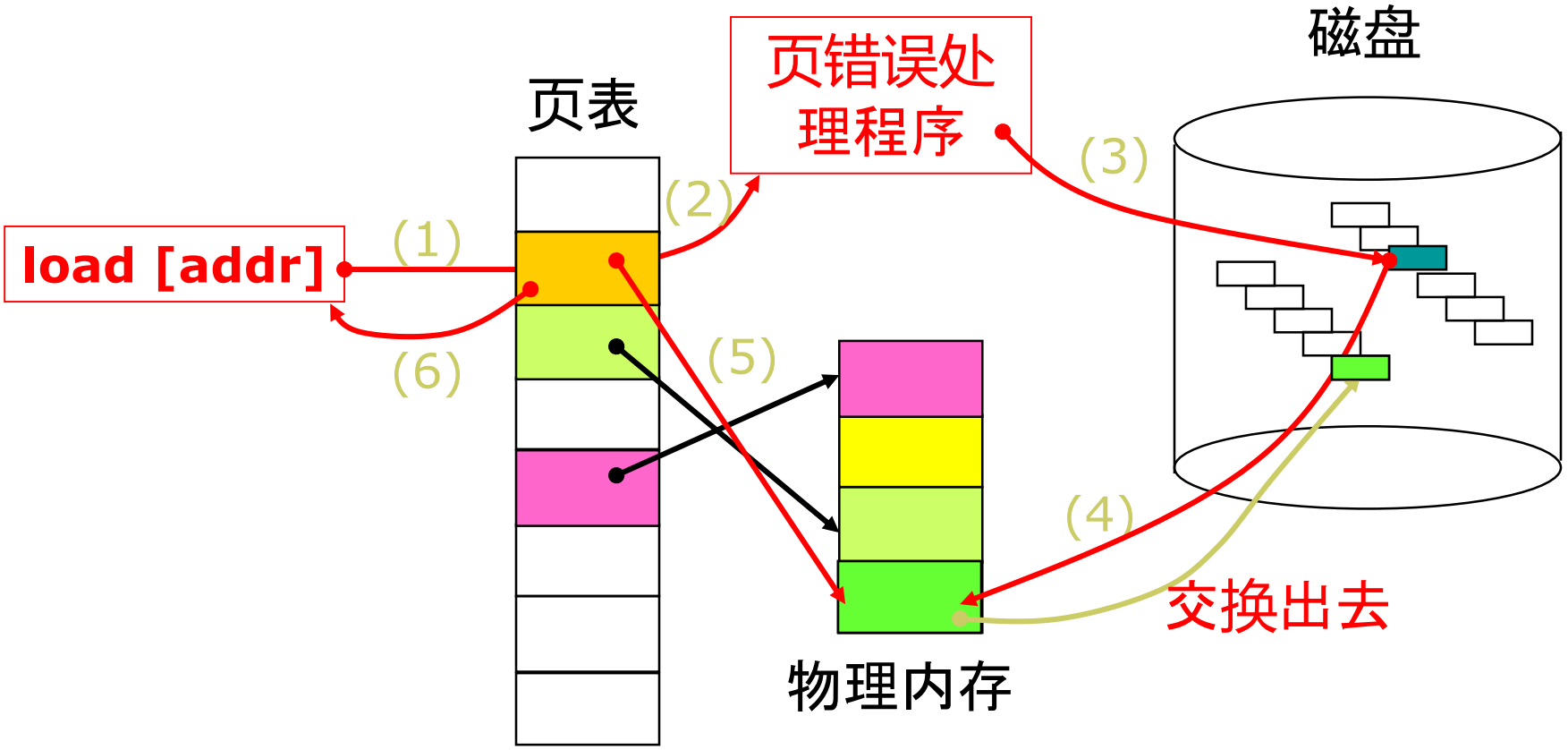
- 扇区 — 物理盘存储空间基本编址单位，一般为512字节
- 主引导记录MBR — 硬盘的第1个扇区的内容，含引导代码和主分区表
- 分区 — 硬盘中可以作为逻辑盘管理的一组扇区集合
- 可扩展分区 — 可以继续划分成“分区”的硬盘分区
- 引导分区 — 标记有可引导标记的硬盘分区，这种分区有引导扇区和引导文件
- 引导扇区 — 引导分区的第1个扇区
- 可扩展分区引导记录 — 可扩展分区中第2个扇区中的内容



Example hard disk layout

页面置换（Swap）？

请求调页—页面置换

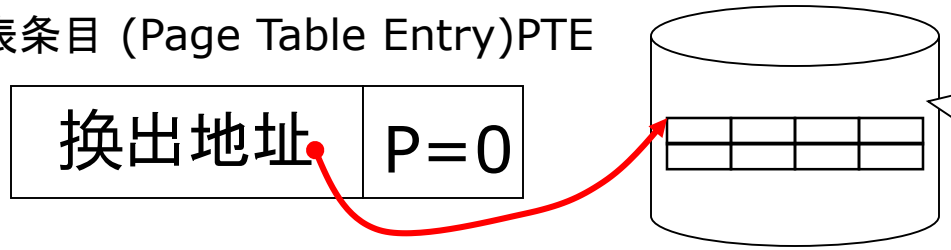


- 交换出去的页面放在哪里?

交换分区

- ⑩ 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

页表条目 (Page Table Entry)PTE



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)

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

Tape is Dead
Disk is Tape
Flash is Disk
RAM Locality is King

Jim Gray

Microsoft

Chunkstash speeding up inline storage deduplication using flash memory

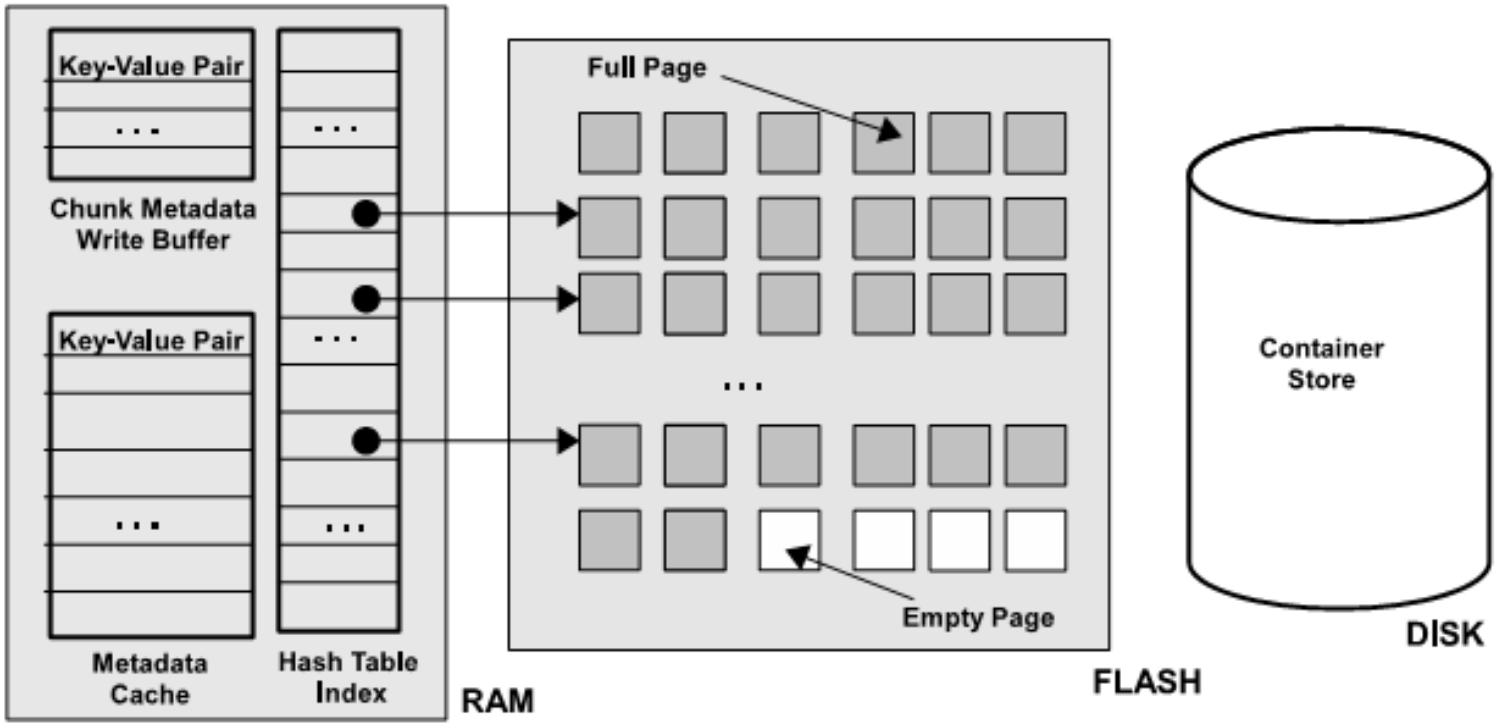
USENIX ATC 2010

ChunkStash

磁盘的性能：IOPS大约100

固态盘性能：IOPS大约100,000

索引组织: cuckoo hash



ChunkStash（续）

测试结果

相关问题

最终的应用？

Windows Server 8（2012）：重复数据删除性能——通过在存储卷中来重复数据删除，以节约磁盘空间，使得存储更加高效，这个新功能可以大大提高重复删除VDI部署里数据的整体效能。附带好处是,重复删除数据技术大大提高了虚拟桌面的启动性能。