操作系统　第五章　外设管理

5.1 外设 和 缓存

同济大学计算机系

Part 1 编程外设硬件

- 计算机的外部设备（External Device） —— CPU 和 内存之外的所有硬件

- CPU 访问内存 和 访问外设 是不一样的

1、计算机与外设的交互 —— 外设控制器

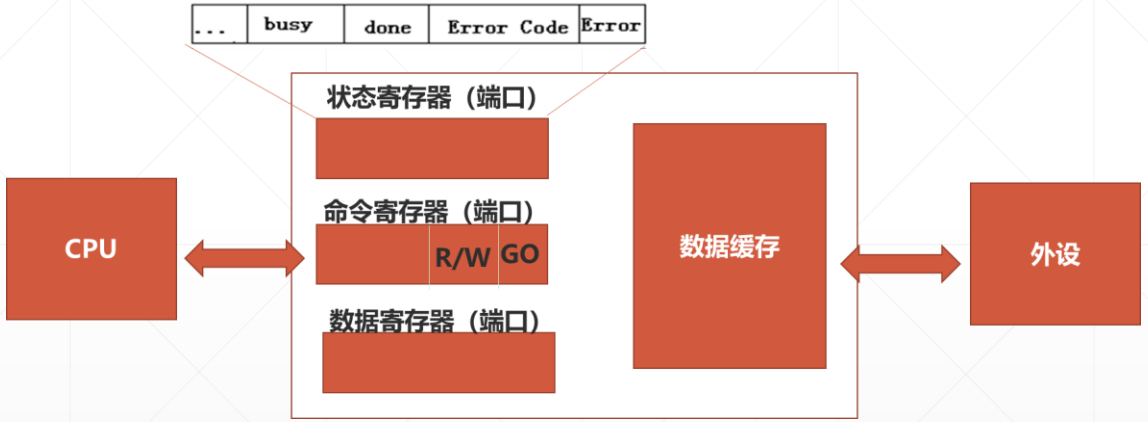
- 外设设备控制器是CPU 与I/O 设备间的接口，其主要职责是控制一个或多个I/O 设备，以实现I/O 设备和计算机之间的数据交换。
  - 外设连外设控制器，外设控制器连计算机。
  - CPU向外设控制器发IO命令，外设控制器控制外设执行输入输出操作。

外设控制器的数据缓存 和 IO操作

读操作（I）：　外设准备的数据放在数据缓存中，供计算机读取

写操作（O）：　计算机写数据缓存，外设将其中的数据转化成物理信号向外界输出

外设控制器的寄存器（端口）



R/W　1是读命令，0是写命令

CPU 写命令端口 向外设发IO命令。

- 读命令　　out 0x00000003, comPort
- 写命令　　out 0x00000001, comPort

CPU 读状态端口（ in 指令），查询外设的工作状态。

- busy标志为1，IO命令执行中，CPU应暂停下一条命令的发送，直至busy标志变为0。
- done 标志为1，IO命令执行完毕。对输入而言，外设输入的信息已经出现在数据缓存。对输出而言，CPU放在数据寄存器中的数据，外设已经输出完毕。
- error 标志为1，IO命令出错，error code 错误的类型。

CPU 读数据端口取外设输入（数据缓存里）。
写数据端口向外设输出（写入数据缓存）。

2、 三种外设控制方式
2.1 程序直接控制IO方式（忙等，读操作）

CPU侧（内核）

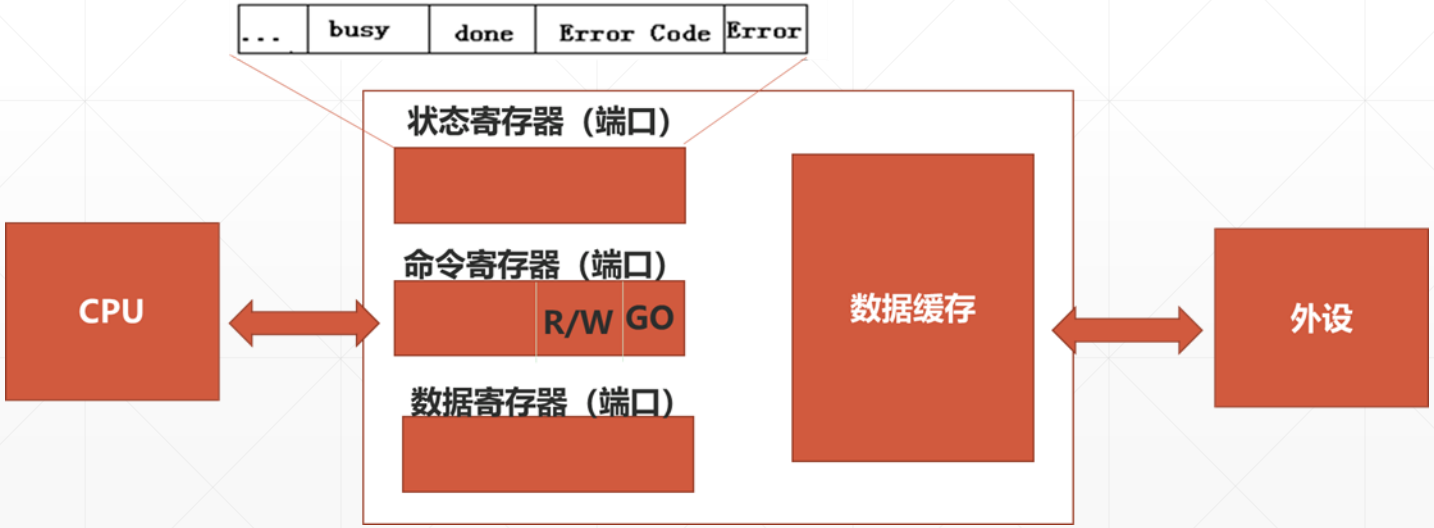
- 忙等外设空闲（busy为0）
- 置外设忙: busy标志赋1，此外done，Error 和 Error Code清0
- 发读命令
- 忙等 IO 完成（done变1）
- 读数据缓存的内容到内存
- busy标志清0

外设控制器

控制外设，实施读操作
IO完成，外设控制器将done标志置1

外设硬件

- 准备好的数据送数据缓存



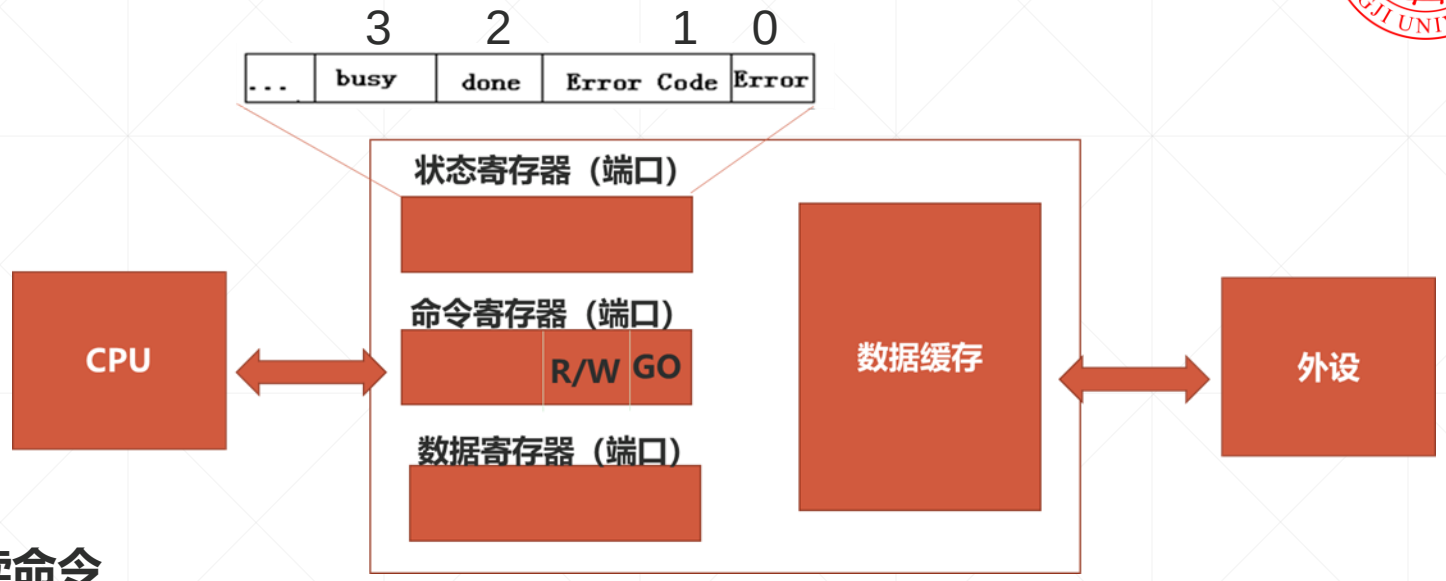
外设编程

1、忙等外设空闲

```
mov 0x0...0001000， flags
do {
    in statusPort， eax
    and flags， eax
} while（eax)
```

2、置外设忙

```
in statusPort， eax,
or 0x0...0001000， eax
out eax，statusPort
```



3.1 发读命令

```
mov 0x0...0000011， eax
out eax， commandPort
```

3.2 发写命令

```
mov 0x0...0000001，eax
out eax， commandPort,
```

4、读数据缓存的内容到内存，硬盘为例

数据缓存：512字节

数据端口：4字节

指令 in dataPort，eax 执行128次

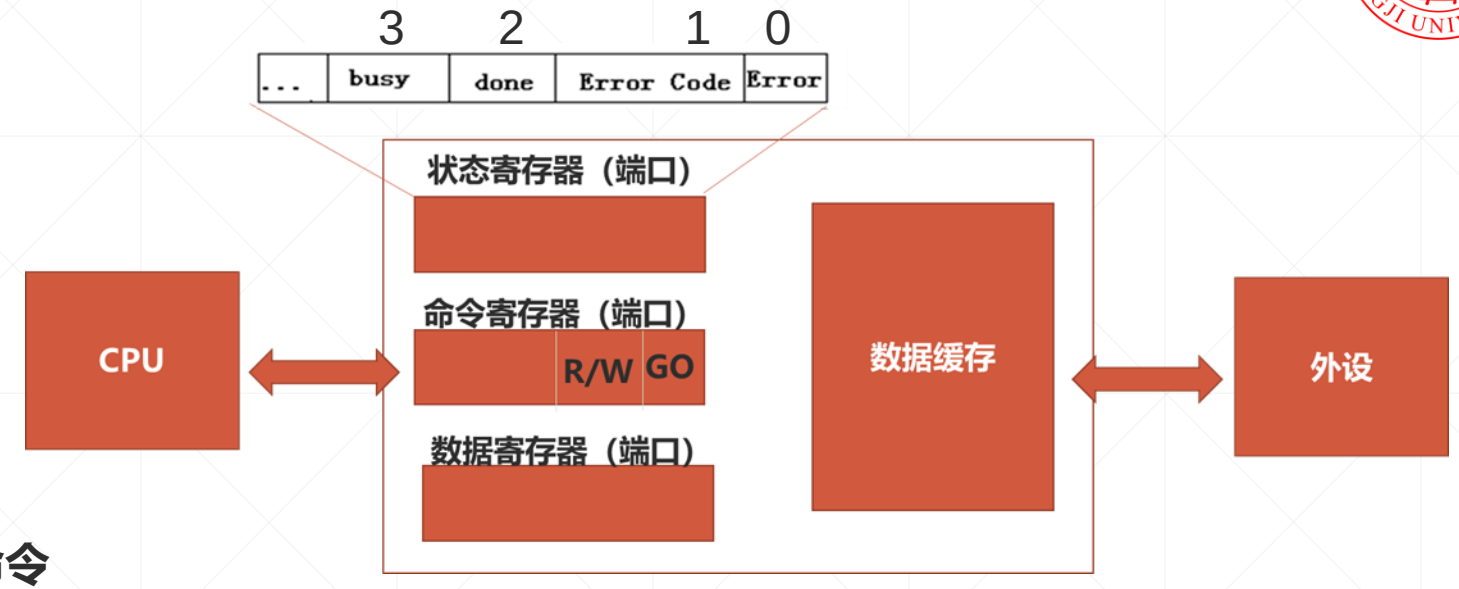
外设编程

1、忙等外设空闲

```
mov flags, 0x0...0001000
do {
    in eax, statusPort
    and eax，flags
} while（eax)
```

2、置外设忙

```
in eax, statusPort
or eax, 0x0...0001000
out statusPort, eax
```



3.1 发读命令

```
mov eax，0x0...0000011
out commandPort，eax
```

3.2 发写命令

```
mov eax，0x0...0000001
out commandPort, eax
```

4、读数据缓存的内容到内存，硬盘为例

数据缓存：512字节

数据端口：4字节

指令 in dataPort eax 执行128次

程序直接控制IO方式（忙等，写操作）

CPU侧（内核）

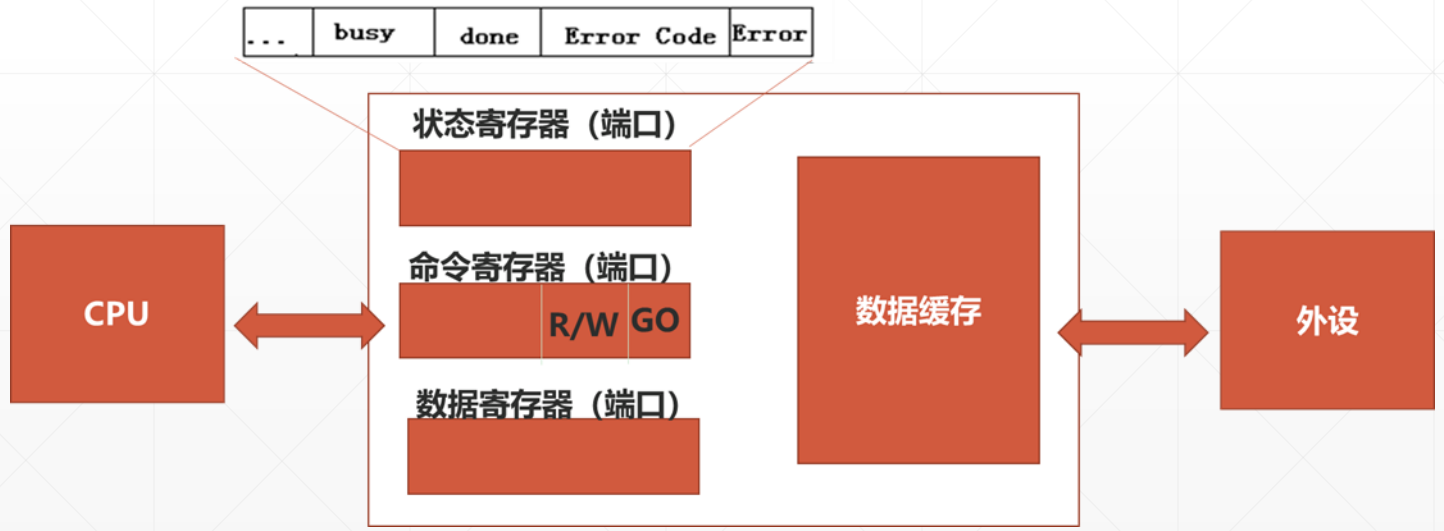
- 忙等外设空闲（busy为0）
- 置外设忙: busy标志赋1，此外done，Error 和 Error Code清0
- 把内存数据送入数据缓存
- 发写命令
- 忙等 IO 完成（done变1）
- busy标志清0

外设控制器

控制外设，实施写操作
IO完成，外设控制器将done标志置1

外设硬件

- 读数据缓存中的数据
- 信号转换送计算机外部



评论：　程序直接控制IO方式　忙等

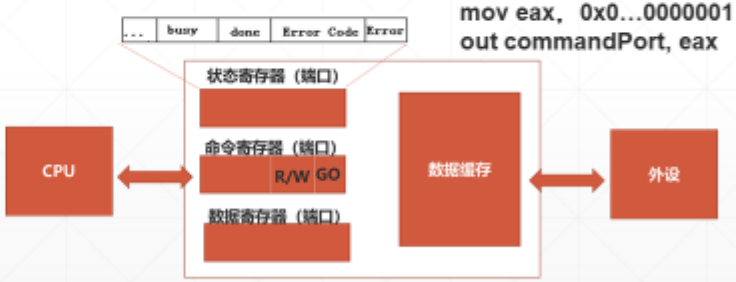
2、 三种外设控制方式
2.1 程序直接控制IO方式（忙等，读操作）

- 忙等外设空闲（busy为0）
- busy标志赋1, done标志清0
- 发读命令
- 忙等 IO 完成（done变1）
- 读数据缓存的内容到内存
- busy标志清0

IO完成，外设控制器将done标志置1。

程序直接控制IO方式（忙等，写操作）

- 忙等外设空闲（busy为0）
- busy标志赋1, done标志清0
- 把内存数据送入数据缓存
- 发写命令
- 忙等 IO 完成（done变1）
- busy标志清0



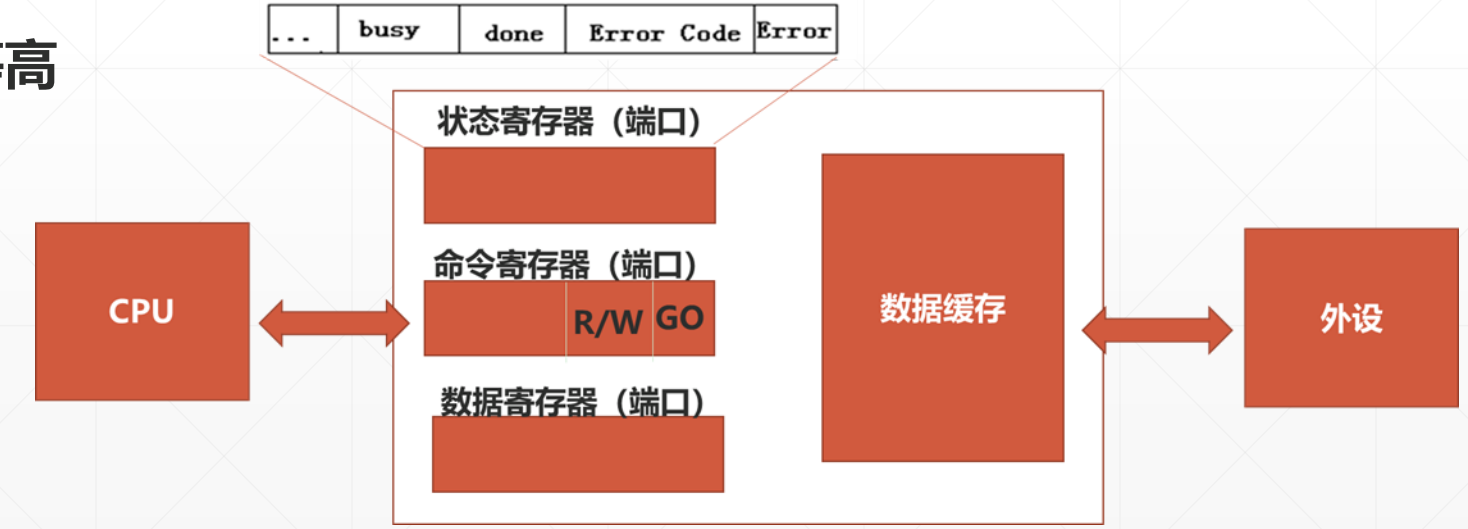
发写命令：
mov eax，0x0...0000001
out commandPort, eax

CPU承担全部IO事务：忙等外设空闲、IO完成。
IO系统效率低下。多个外设无法并行工作，计算和IO无法并行。

程序直接控制IO方式　　轮询（polling）

- 程序在合适的时间点主动查看外设状态
  - 发出请求时，设备忙，离开
  - 发出请求后，IO未完成，离开

- 由于没有忙等，所以轮询的效率比忙等高
- 多个外设可以并行
- 但CPU依然需要承担所有外设事务



2.2 中断驱动的IO方式

读操作（同步读）

CPU侧（外设驱动程序）

1、现运行进程PA**入睡**等IO数据（磁盘的话，发读命令后睡；键盘没读命令的，就直接睡）。

5、PA被选中，使用CPU处理内存中的数据。

外设控制器

2、控制外设，实施读操作

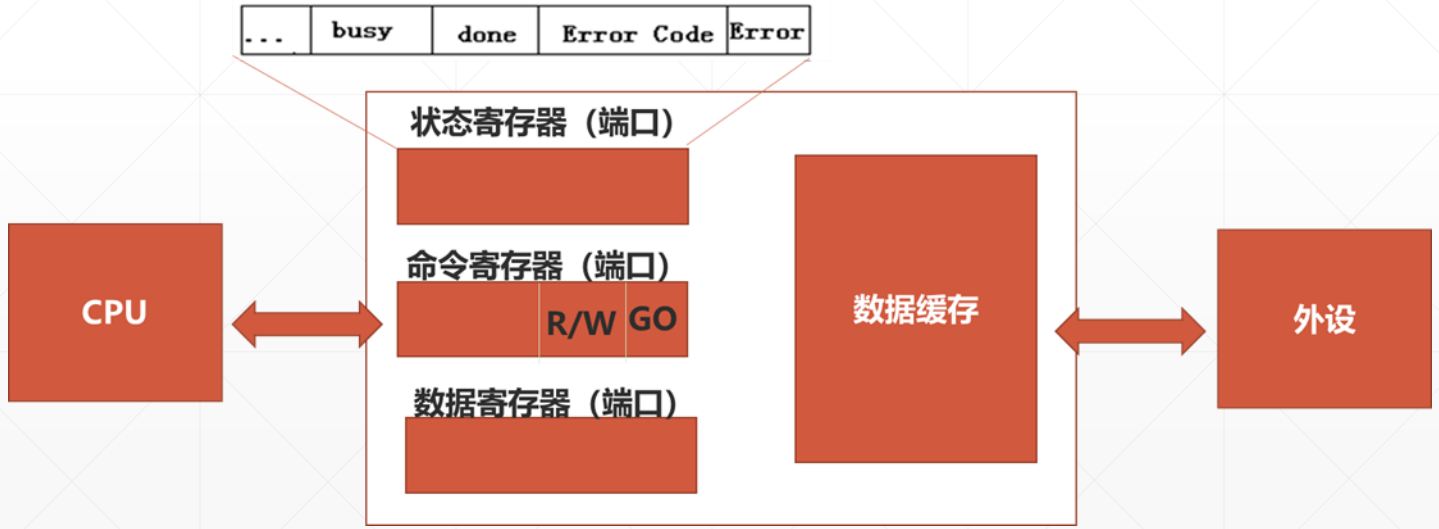
3、IO完成，外设控制器将done标志置1，**发中断信号**

外设硬件

2、准备好的数据送数据缓存

CPU侧（中断处理程序）

4、将数据缓存中的数据复制进内存，**唤醒PA**。



2.2 中断驱动的IO方式 写操作（异步写）

1、PA产生要输出的数据

CPU侧（外设驱动程序）

2、PA将内存数据写入数据缓存。

3、PA发写命令

CPU侧（中断处理程序）

6、none

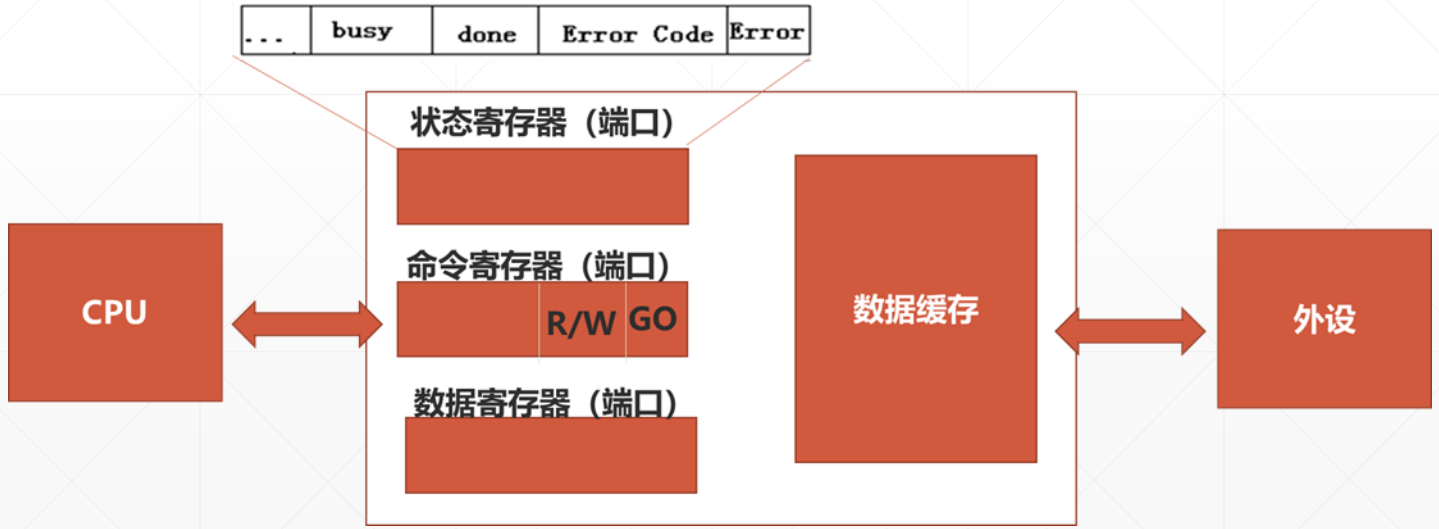
外设控制器

4、控制外设，实施写操作

5、IO完成，外设控制器将done标志置1，发中断信号

外设硬件

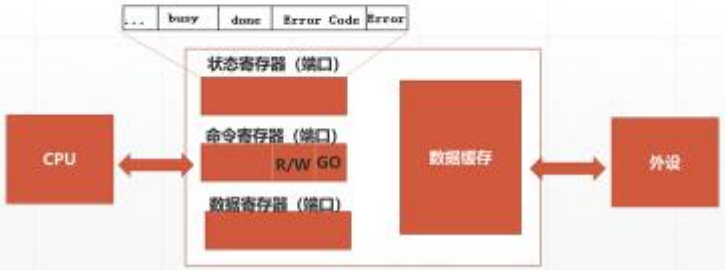
4、数据缓存中的数据送外设



评论　中断驱动的IO方式

2.2 中断驱动的IO方式　读操作

- 现运行进程PA入睡等IO数据（磁盘的话，发读命令后睡；键盘没读命令的，就直接睡）
- IO完成后，外设控制器向CPU发中断信号
- 中断处理程序将数据缓存中的数据复制进内存
- …… PA处理输入数据

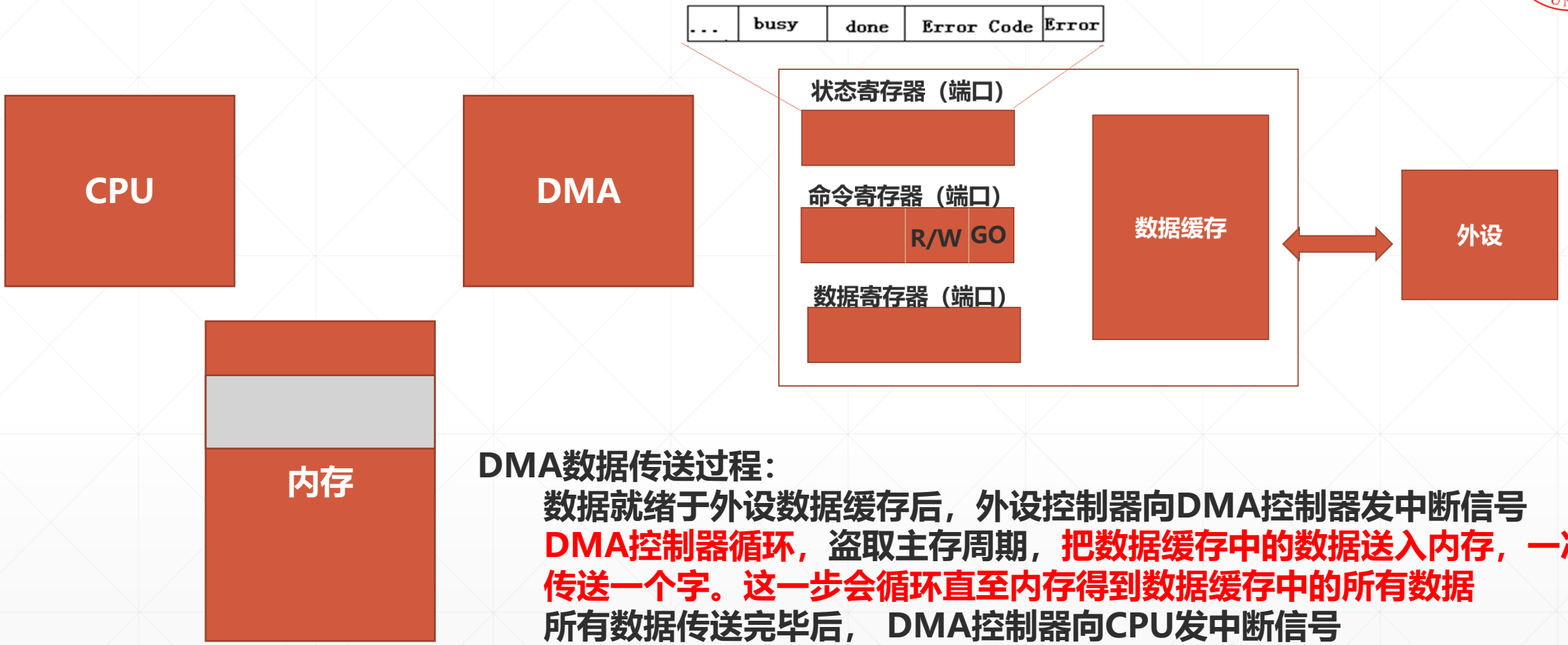


2.2 中断驱动的IO方式　写操作

- PA产生要输出的数据
- PA将内存数据写入数据缓存
- PA发写命令
- IO完成后，外设控制器向CPU发中断信号（本次写操作完成）

- 中断是IO完成信号。CPU不用忙等或轮询 IO操作。　计算和IO可以并行。不同的外设，IO也可以并行。
- IO数据从数据缓存复制到内存的任务由CPU完成。所以，中断驱动的IO方式只适合键盘、鼠标 等数据缓存很小的外设

2.3 DMA（direct memory access)方式（磁盘、网络、GPU……）



DMA数据传送过程：

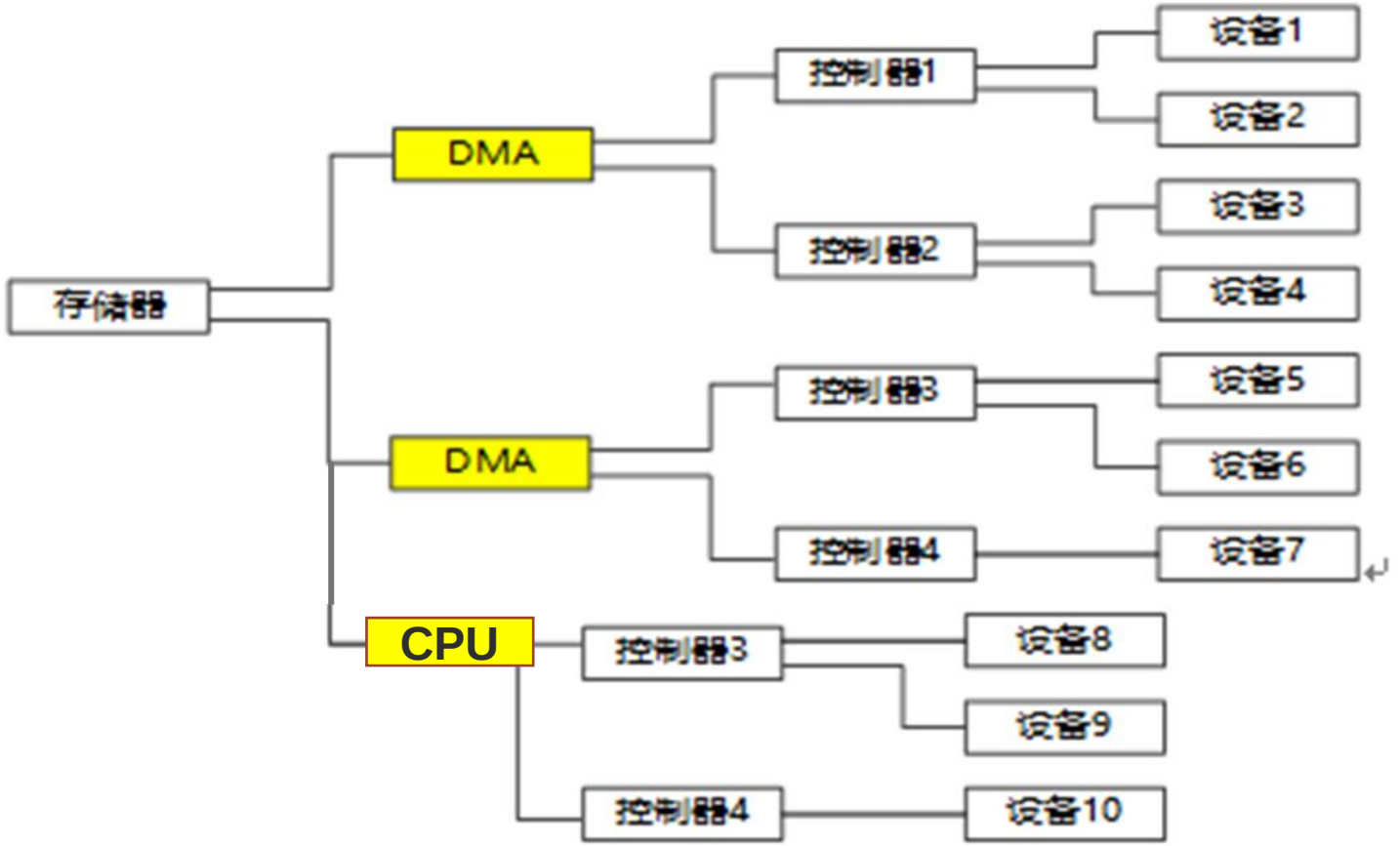
数据就绪于外设数据缓存后，外设控制器向DMA控制器发中断信号

DMA控制器循环，盗取主存周期，把数据缓存中的数据送入内存，一次传送一个字。这一步会循环直至内存得到数据缓存中的所有数据

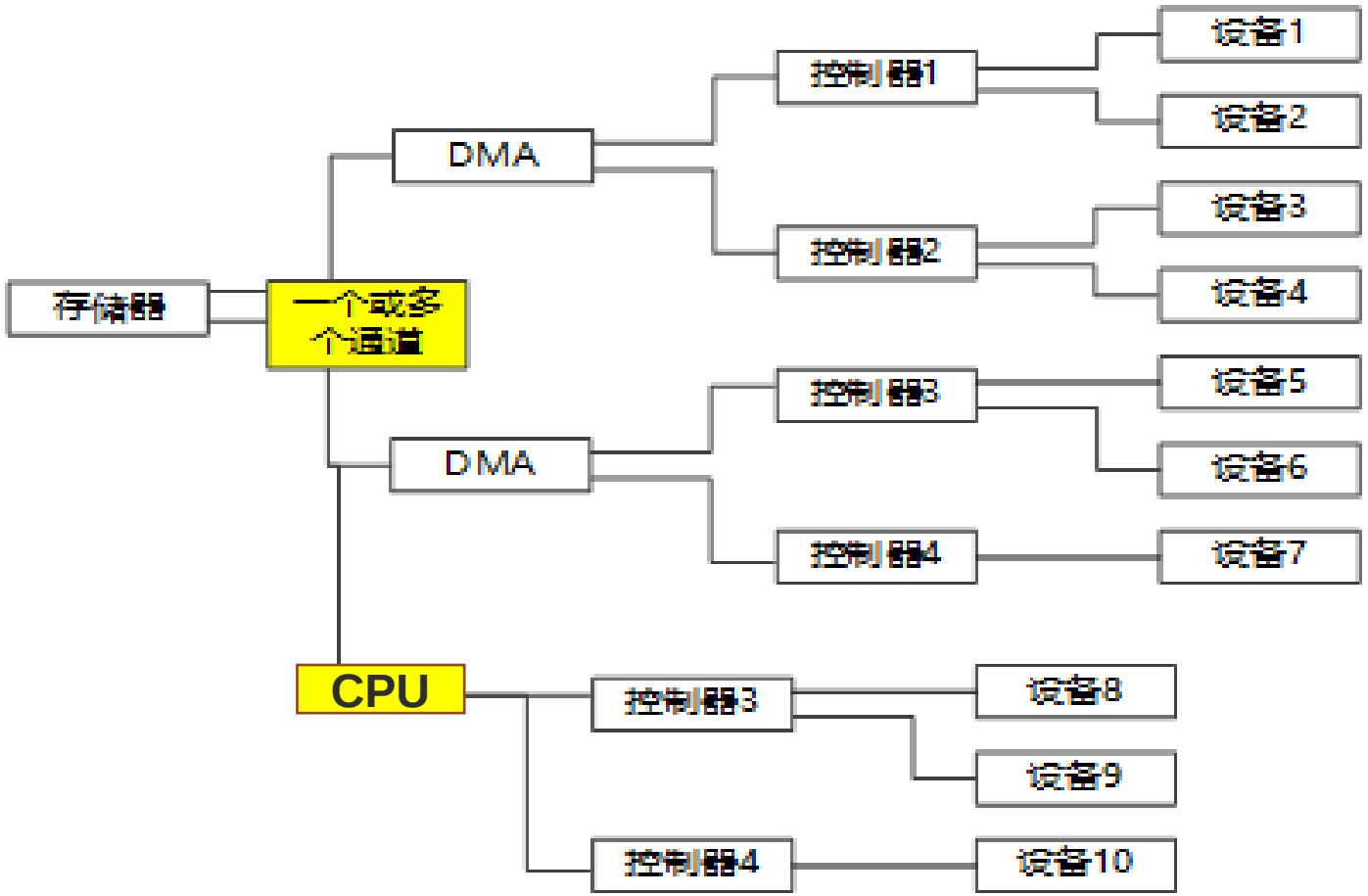
所有数据传送完毕后， DMA控制器向CPU发中断信号

中断处理程序不复制数据。 CPU完全不参与数据传输

3 计算机外设系统硬件架构 PC 架构



计算机外设系统硬件架构　大型机/高档服务器架构



Part 2 缓存

- 内核在核心态数据区为每个外设配缓存，存外设IO数据。

- 缓存的价值
  - 尽量减少 CPU和外设 的交流次数，提高计算和IO的并发度。
  - IO数据重用。

1、 外设的分类

- 存储设备（块设备）
  - 用途　信息存储
  - 举例　硬盘、软盘、U盘、光盘、磁带
  - 特征　数据存储以数据块为单位。

数据块是IO的最小单位。每次IO，读写的数据量是数据块尺寸的整数倍。

比如硬盘，Unix V6++的数据块是一个磁盘扇区，512字节；Linux的数据块是相邻的8个扇区，4096字节。

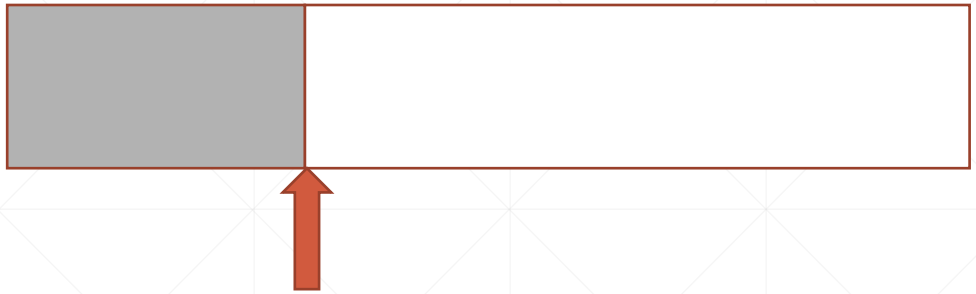
- 字符设备
  - 用途　字符界面的人机接口
  - 举例　终端设备，指键盘、屏幕
  - 特征　与计算机系统每次交互的数据量不定

- 其它设备
  - 网络设备
  - 鼠标
  - 耳麦、扬声器，摄像头
  - 其余传感器

2、终端的 行缓存

每个TTY，一个输入缓存

a



键盘中断处理程序每次送一个**ASCII**字符

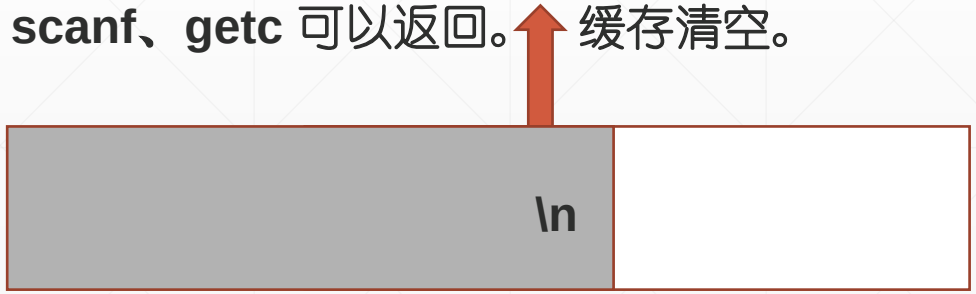
c

每个TTY，一个输出缓存

printf、putc输出的字符串放在这里

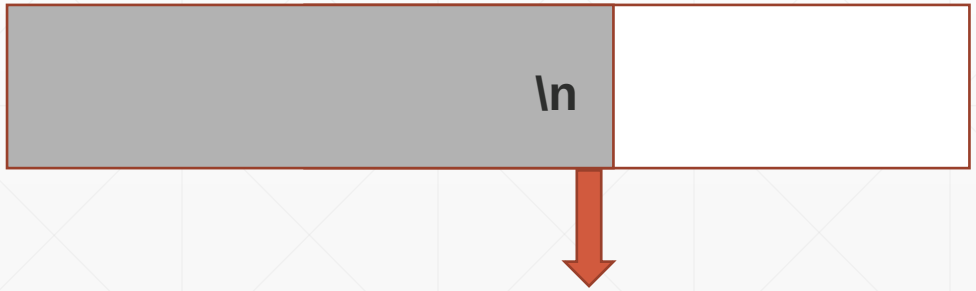
b

积满一行，唤醒等待键盘输入的应用程序。
scanf、getc 可以返回。缓存清空。

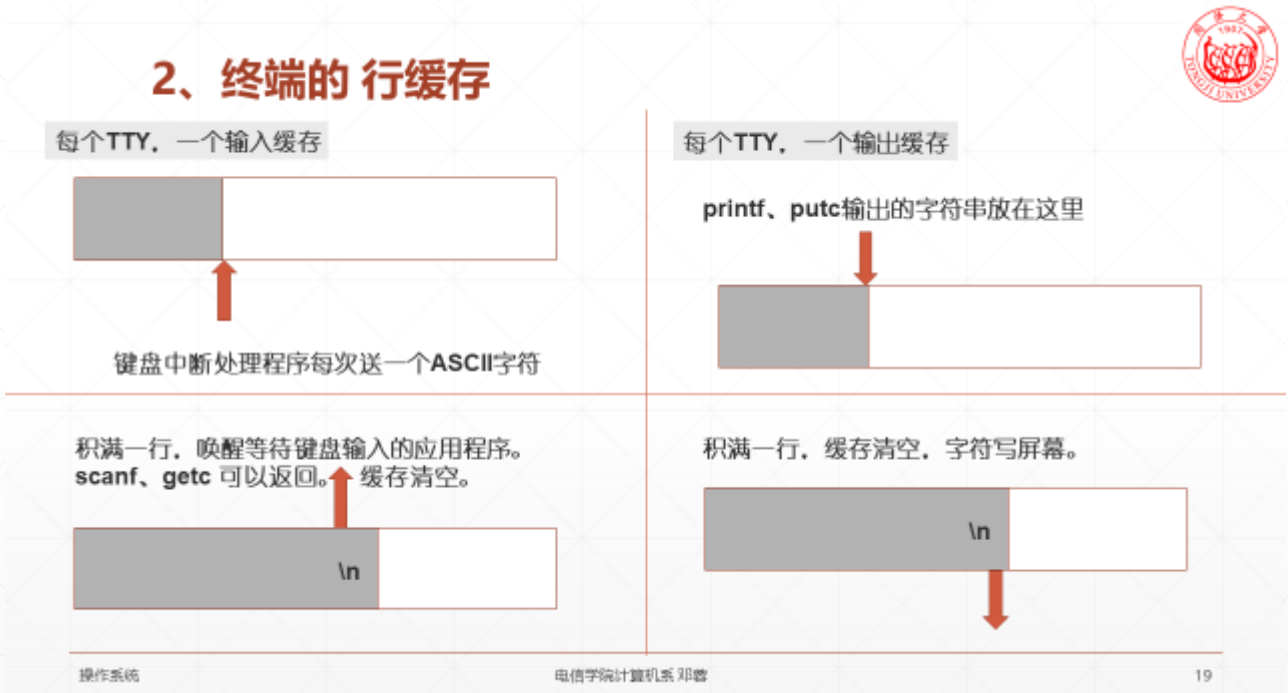


d

积满一行，缓存清空，字符写屏幕。

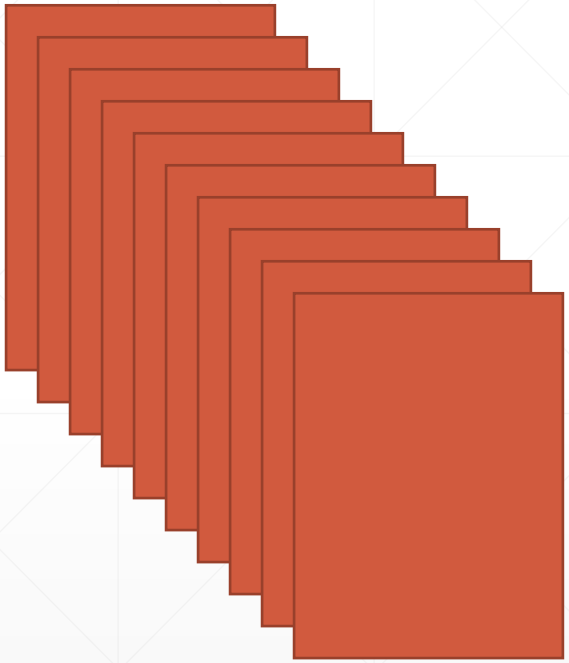


作用



- 集齐一行唤醒等待输入的应用程序。减少进程切换次数。
- 集齐一行写屏幕，提高输出效率。

3、块设备的缓存池



好多等尺寸的缓存块，每个缓存块装一个磁盘数据块，后者是磁盘上的连续多个扇区。

- **UNIX V6++，缓存块512字节，装磁盘上的1个扇区**
- **Linux，缓存块4096字节，装连续的8个扇区**

缓存块，所有进程共享，互斥访问。用前上锁，用后解锁。

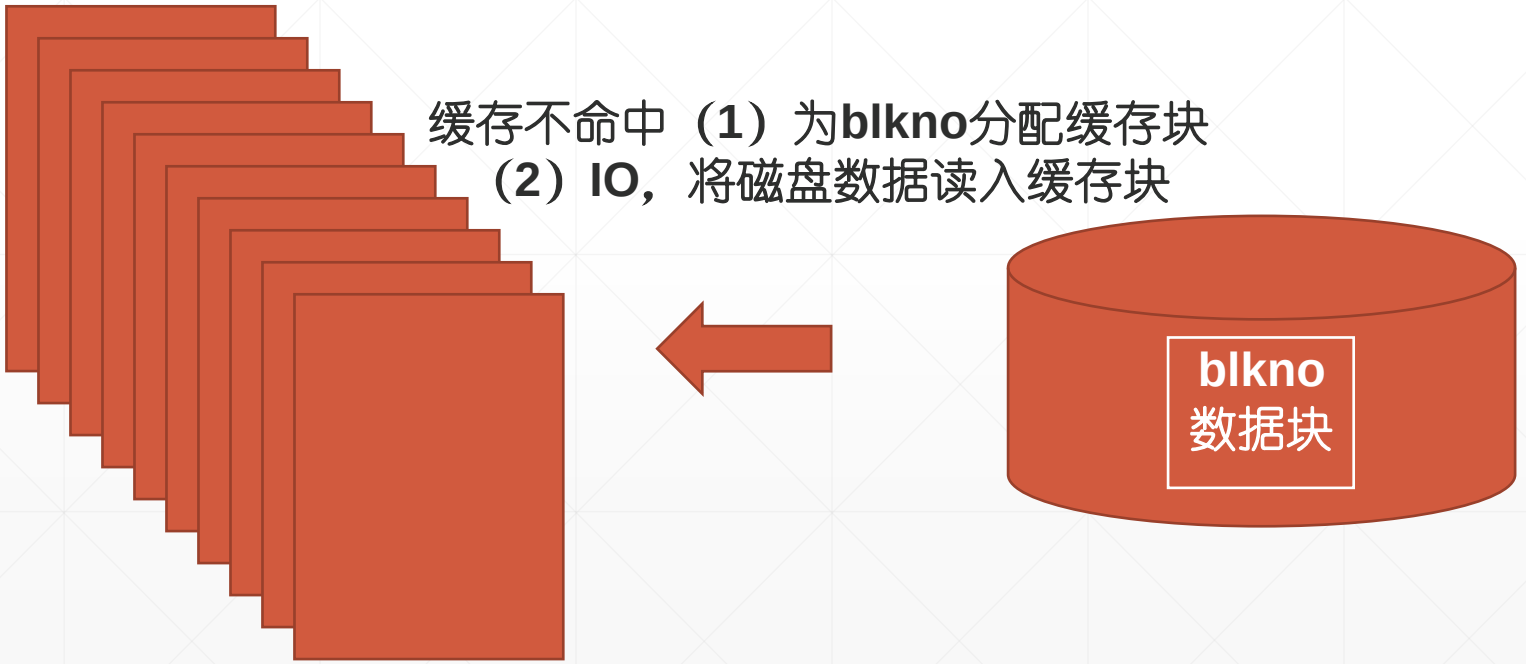
3.1 磁盘读操作（基于缓存的基本读操作）

- **读磁盘数据块 blkno（物理块号）**
- **操作：**

1、缓存池找到分配给 blkno 的缓存块，锁住它

2、读缓存块中的数据

3、 解锁



3.1.1 缓存命中

- 锁缓存块，将缓存块中的数据复制到现运行进程的用户区，解锁缓存块。

- 耗时 = t1 + t2
  - t1，上锁操作的耗时。
    - 读操作执行时，没有其它进程访问数据块blkno
      t1 == 0
    - 否则，需要入睡等其它进程使用完数据块、解锁
      t1 是一个随机量
      进程会睡　sleep(&缓存块，-50)
  - t2，内存数据传输的耗时。假设，进程需要读入的数据量是 n字节

```
src = 缓存块首地址
dst = 用户区中，某个数组的首地址

for( i = 0；i< n；i+=4，dst+=4，src+=4 )
    (int*) dst = (int*) src ；
```

3.1.2 缓存不命中

(1) 为磁盘数据块blkno分配缓存块、上锁

(2) 启动磁盘IO、将磁盘上的blkno数据块读入分配给它的缓存块

(3) 将缓存块中的数据复制到现运行进程的用户区

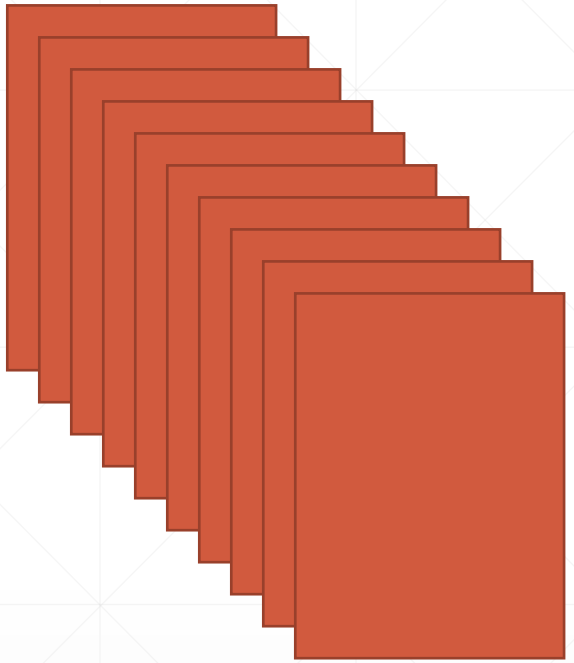
(4) 解锁缓存块。

t2

3.1.2.1 缓存分配操作

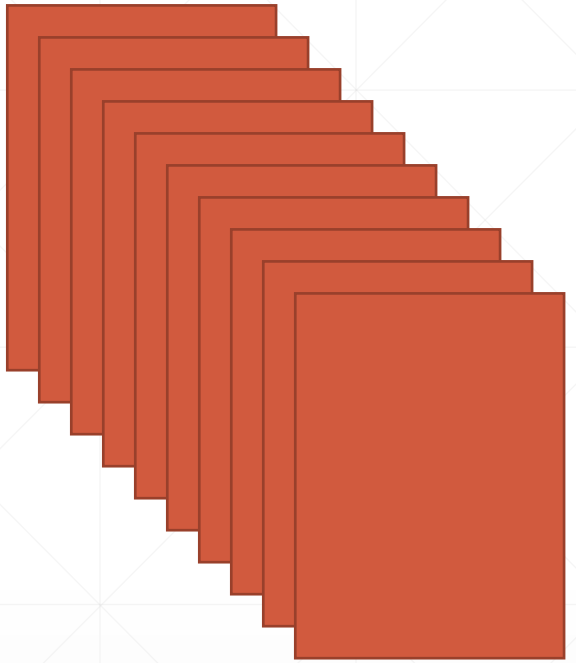
为磁盘数据块blkno分配缓存块、上锁　t3

- 存在空白缓存块，t3==0
- 不存在……，淘汰一个自由缓存块（没有进程正在使用的缓存块）
  - 没有，进程会入睡等待自由缓存块。t3是一个随机量
  - 有，将LRU自由缓存块分配给进程。t3==0

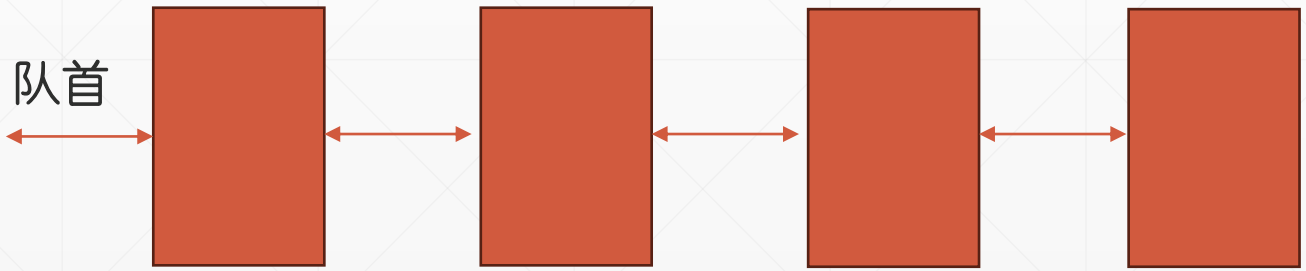


（1）为磁盘数据块blkno分配缓存块、上锁　t3

- 存在空白缓存块，t3==0
- 不存在……，淘汰一个自由缓存块（没有进程正在使用的缓存块）
  - 没有，进程会入睡等待自由缓存块。t3是一个随机量
  - 有，将LRU自由缓存块分配给进程。t3==0



LRU队列，管理自由缓存块（整个系统只有一个）



释放的缓存块，解锁、放队尾
命中的缓存块，上锁，从队列中删除

队首，LRU缓存块。上锁，摘除，分配给新数据块

3.1.2.2 执行磁盘读操作

将磁盘上的数据块读入分配给它的缓存块

- 为缓存块构造一个 IO 请求块
  - src = blkno，dst = 缓存块首地址，IO 数据量 = 数据块大小， 读操作
- 发IO命令
  - 送IO请求块入IO请求队列
    - 先前队列空（磁盘空闲），用IO请求块中的信息写外设命令端口，向磁盘发读命令
    - 否则，IO 请求块入队
- 入睡等待磁盘IO结束　　sleep(&缓存块, -50)

磁盘驱动程序

（1）送IO请求块入IO请求队列

（2）原先前队列空（磁盘空闲）？

空，用IO请求块中的信息写外设命令端口，向磁盘发IO命令

非空，IO 请求块入队

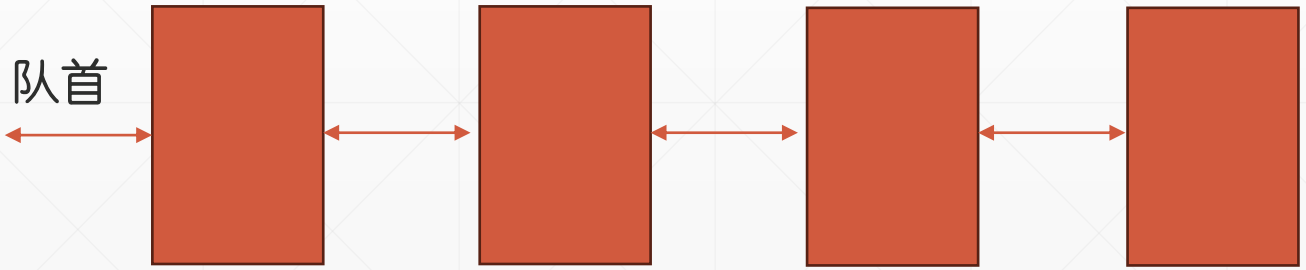
IO完成，**磁盘中断处理程序**

（1）摘除队首缓存块，WakeUpAll(&缓存块)

（2）IO请求队列非空？

用新队首IO请求块中的信息写外设命令端口，向磁盘发IO命令

IO请求队列，管理未完成的IO请求。 Unix V6，FIFS队列



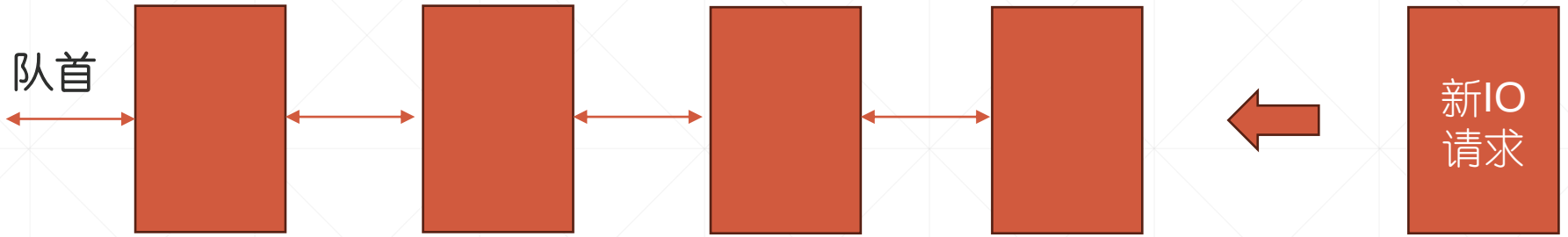
新IO请求的缓存块放队尾

队首缓存块，IO进行中

IO操作的耗时

T = 等待时间 + 服务时间 ≈

从磁盘读取已有数据块的耗时 + 从磁盘读取新数据块的耗时



缓存不命中，读操作的耗时（假设缓存池足够大，不存在没有自由缓存块的情况）

(1) 为磁盘数据块blkno分配缓存块、上锁

(2) 启动磁盘IO、将磁盘上的blkno数据块读入分配给它的缓存块

(3) 将缓存块中的数据复制到现运行进程的用户区

(4) 解锁缓存块。

t2

T + t2 ， T >> t2

例 1：

- 时刻 T1，下述数据块序列缓存不命中。请比较使用缓存池的IO 和 未使用缓存池的IO。性能指标（1）完成IO请求的平均耗时 （2）完成整个IO请求序列系统的总耗时（3）IO总次数

已知：将IO请求放入IO请求队列后至IO操作完成，进程平均需要等待 T。

从缓存复制数据到用户空间（含上锁、解锁操作），平均耗时 t。　　T>>t 。

IO请求序列　556，556，556，600，782，891，900，556，556，556

未使用缓存池，数据从磁盘直接复制到用户空间，IO请求平均耗时 T 。

完成整个IO请求序列系统的总耗时，10 *T。　IO，10次。

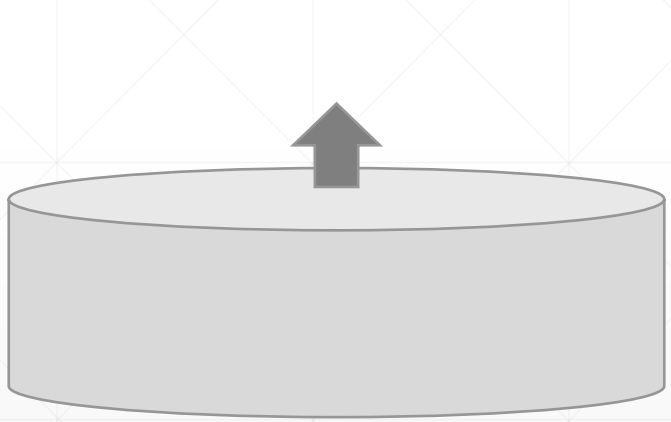
使用缓存池，数据从磁盘复制到缓存。之后进程从缓存中取用数据，复制到自己的用户空间。完成整个序列，IO 5个磁盘数据块。总耗时，5 * T + 10 * t。　IO，5次。

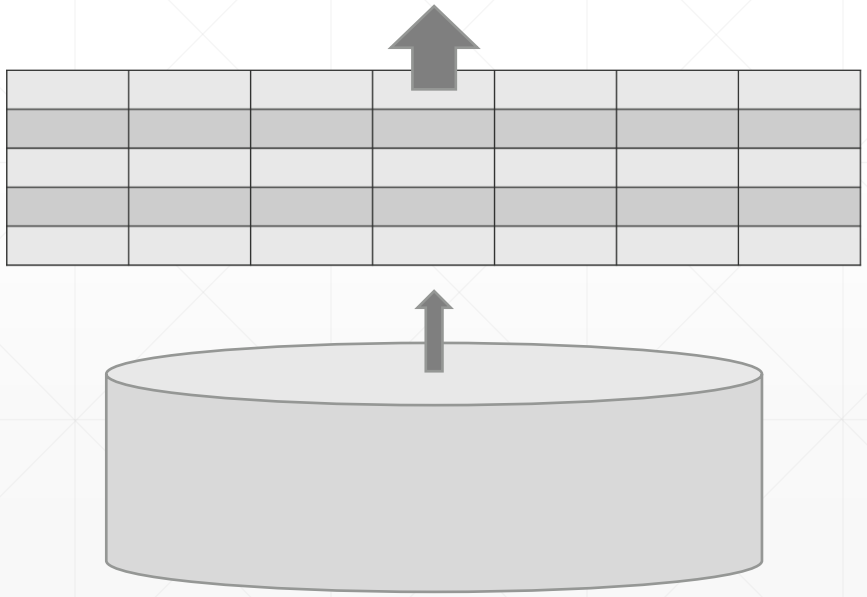
IO请求平均耗时 T/2+t 。

更一般地，借助缓存，访问一个磁盘数据块的平均耗时 T/n+t，n是该数据块缓存命中次数 + 1

对于读操作，缓存的价值

- 若数据块有重用的可能，磁盘高速缓存池可以有效缩短读操作的平均耗时，减少磁盘IO操作的数量，大幅提高磁盘数据服务子系统的吞吐率。





引入缓存的代价

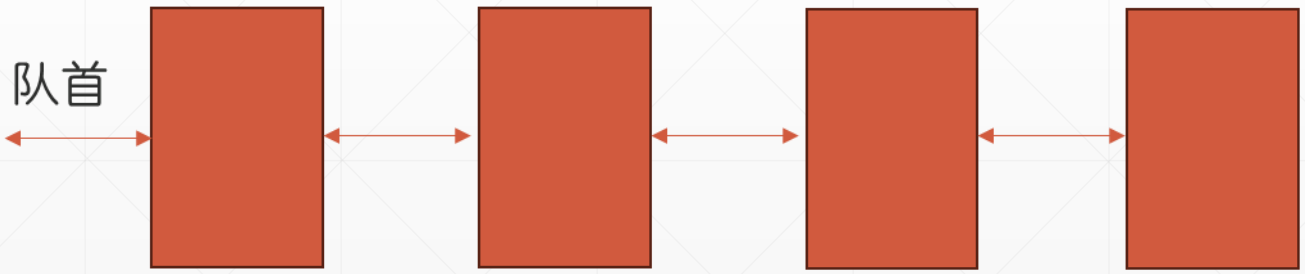
更一般地，借助缓存，访问一个磁盘数据块的平均耗时 T/n+t，n是该数据块缓存命中次数 + 1

原本，数据直接进用户空间。现在先进内核缓存块，再进用户空间。
若数据块没有重用可能，耗时是增加的。

如果我们需要遍历千兆以上的大数据集，每块数据仅使用一次。
关掉磁盘高速缓存管理模块，有利于提升系统性能。例：GFS。

思考题：用什么操作可以废掉LRU队列？
请想办法 保护 LRU 队列。

LRU队列，管理自由缓存块（整个系统只有一个）



释放的缓存块，解锁、放队尾
命中的缓存块，上锁，从队列中删除

队首，**LRU**缓存块。上锁，摘除，分配给新数据块

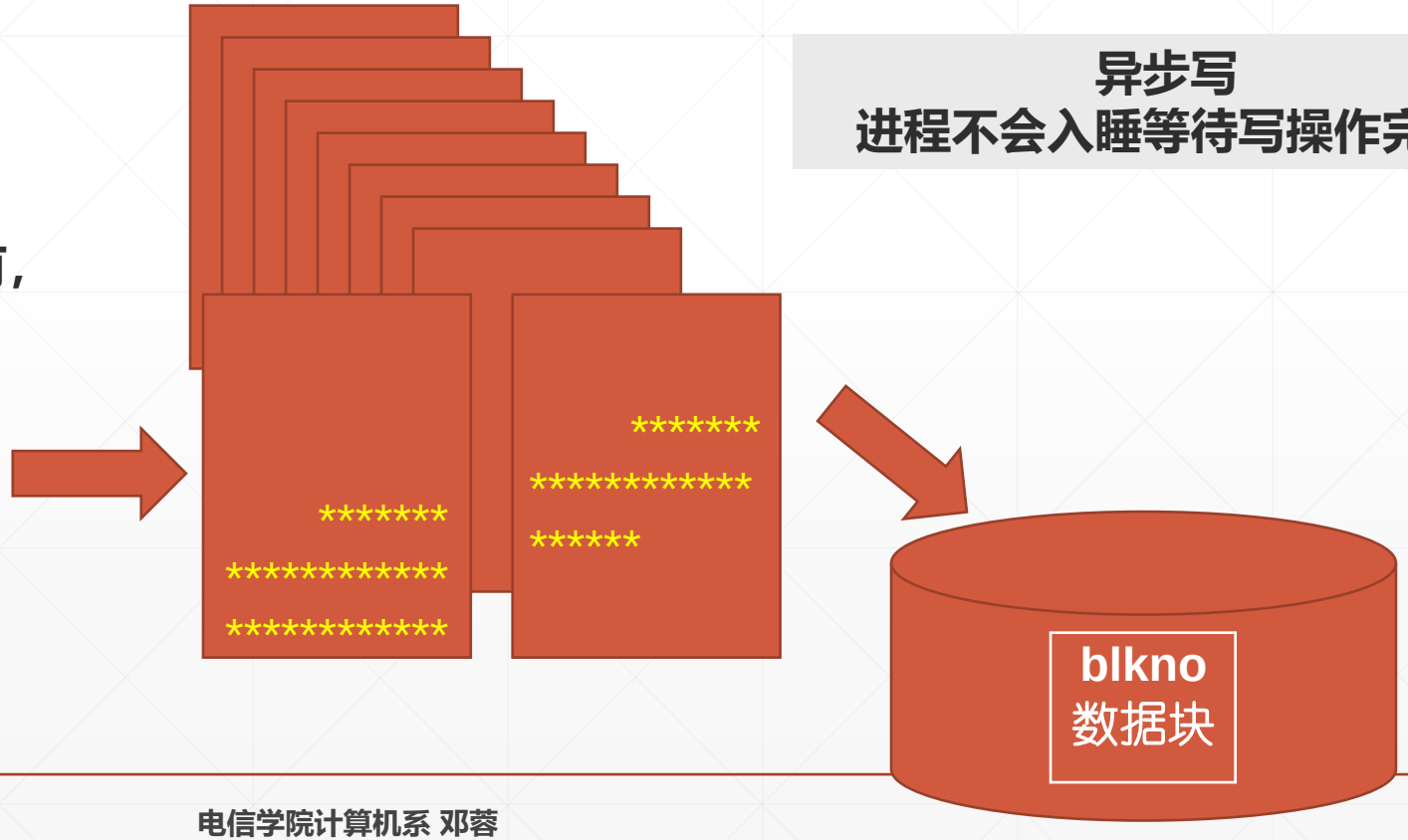
3.2 磁盘写操作（基于缓存的基本写操作）

- 数据写入磁盘数据块 blkno（物理块号）
- 操作：

1、缓存池找到分配给 blkno 的缓存块，锁住它（如果没有，为blkno分配缓存块）

2、数据写入缓存块

3、解锁。写操作返回



异步写
进程不会入睡等待写操作完成

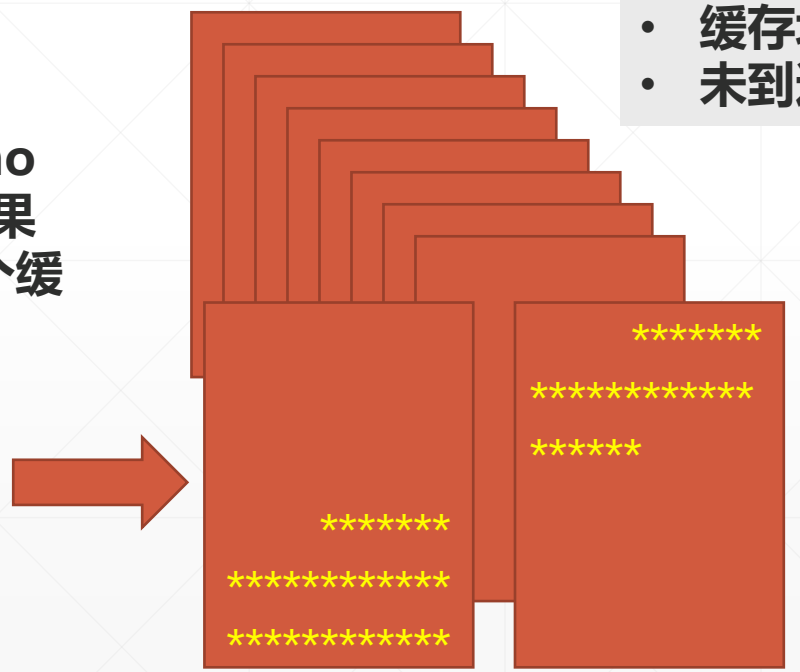
3.2.1 磁盘写操作　延迟写

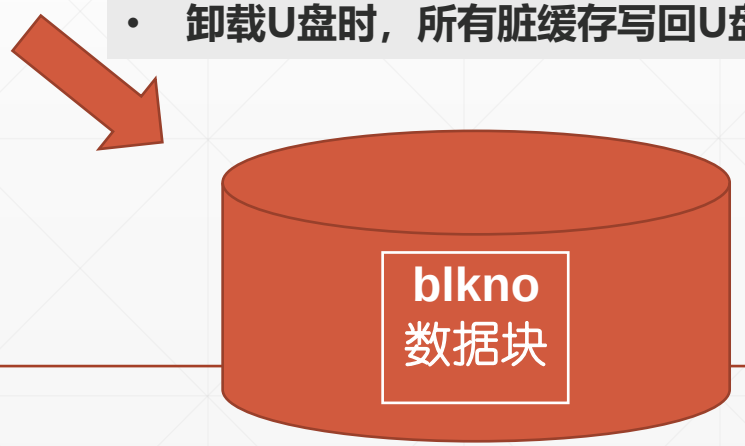
- 数据写入磁盘数据块 blkno（物理块号）
- 操作：

1、缓存池找到分配给 blkno 的缓存块，锁住它（如果没有，为blkno分配一个缓存块）

2、数据写入缓存块

3、 解锁。写操作结束





写过的缓存一定要刷回磁盘。否则，文件的修改就丢失了！

为了提升写操作性能，操作系统广泛采用延迟写技术：

- 缓存块写到尾部，送IO请求队列，启动磁盘写操作。
- 未到达尾部，不写盘。

延迟写产生脏缓存。脏缓存何时写回磁盘？

- 写至尾部，送去IO。
- LRU缓存是脏缓存。装新数据块之前，写回磁盘。
- 关机时，所有脏缓存写回磁盘。
- 卸载U盘时，所有脏缓存写回U盘。

延迟写技术的优缺点

- 优点：合并写操作，减少写操作次数。

缩短写操作的平均响应时间。

- 缺点：丢数据，破坏文件系统的一致性

3.2.2 磁盘写操作　先读后写

- 如果写入缓存块的数据尺寸小于缓存块的容量。需要先读入磁盘数据块，之后向缓存块写入新数据。

- 这是在保护磁盘数据块中，未被新数据覆盖的旧数据。

- 如果写入缓存块的数据尺寸等于缓存块的容量，无需先读后写。

基本的磁盘读写操作：同步读，异步写

- 同步读：若缓存池没有进程所需的磁盘数据，读操作睡眠等待磁盘IO结束。
- 异步写：将数据写入缓存，写操作完成。进程不必睡眠等待磁盘IO结束。