

设备管理

阅读任务

- 第36节 “The I/O Devices”
- 第38节 “Redundant Arrays of Inexpensive Disks(RAIDS)”

本章教学目标

- 了解I/O子系统的设计目标及实现方式
- 理解I/O子系统的硬件和软件原理
- 掌握磁盘的结构
- 掌握驱动器调度算法
- 理解设备独立性和虚拟设备

I/O工作方式回顾

- 轮询
- 中断
- DMA

轮询

- 工作机制
 - CPU不断去测试状态寄存器中特定位的值
 - 若就绪，则CPU从（向）数据寄存器读取（写入）一个字，并保存到主存储器
 - 若I/O设备忙或未就绪，则CPU处于忙等状态（同步）
- 缺点
 - CPU轮询浪费宝贵的时间
 - CPU需要参与数据的传输，与设备只能串行工作

中断 异步.

- 工作机制
 - 进程发出启动I/O指令，CPU加载控制信息到设备控制器的寄存器，之后进程继续执行或放弃CPU等待I/O操作完成；
 - 设备控制器检查状态寄存器的内容，按照I/O指令的要求，执行I/O操作，一旦传输完成，设备控制器通过中断请求线向CPU发出I/O中断信号；
 - CPU收到并响应I/O中断后，转向设备的中断处理程序执行；
 - 中断处理程序执行操作（如数据读取），并将等待的进程移到就绪队列，结束后退出中断处理程序，返回到中断发生前的状态；
回调
 - 进程调度程序在适当时刻对得到数据的进程恢复执行。
- 优点
 - 克服了轮询I/O中的忙等，实现了CPU和设备的部分并行操作
- 缺点
 - CPU仍需参与数据的传输
 - 每当设备控制器的缓冲满后就要发生中断，发生中断的次数较多
 - 每次中断处理需要保护现场和恢复现场，额外开销大
无法实现并行

中断(cont)

- 在执行关键代码的时候，需要有延迟中断处理的能力；
 - 中断分为可屏蔽中断和不可屏蔽中断
 - 在执行关键代码之前，可以屏蔽中断
- 需要高效的分派中断处理的机制
 - 中断向量：存储了中断处理程序的入口地址
- 需要多重中断处理机制，以支持中断优先级
 - 顺序执行
 - 嵌套执行

内存 \leftrightarrow CPU Register \leftrightarrow 外设 . 内存 \leftrightarrow 外设 .

直接存储器存储 (DMA)

- 工作方式 **不需要CPU**

- CPU对DMA控制器进行设置，给出需要传输的数据的源地址、目的地址、需要传输的数据个数，以及控制信息（如读/写），然后就继续其它工作；
- DMA控制器获得总线控制权（此时，CPU无法获得总线），请求I/O控制器和内存进行数据读/写操作
- I/O控制器与内存通过DAM控制器进行数据读/写操作
- 每传输一个字，DMA控制器将计数器减1，并重复上述操作，直至计数器递减为0
- DMA控制器向CPU发出中断请求信号

- 优点

- 数据传输无需CPU参与，实现了CPU和I/O的并行操作
- 适合大数据量传输

- 缺点

- DMA控制器需要和CPU竞争总线使用权
- 总线使用权的竞争会导致CPU等待，但不会造成其寄存器内容的修改

cpu要往输出口

现场 X

I/O设备的种类

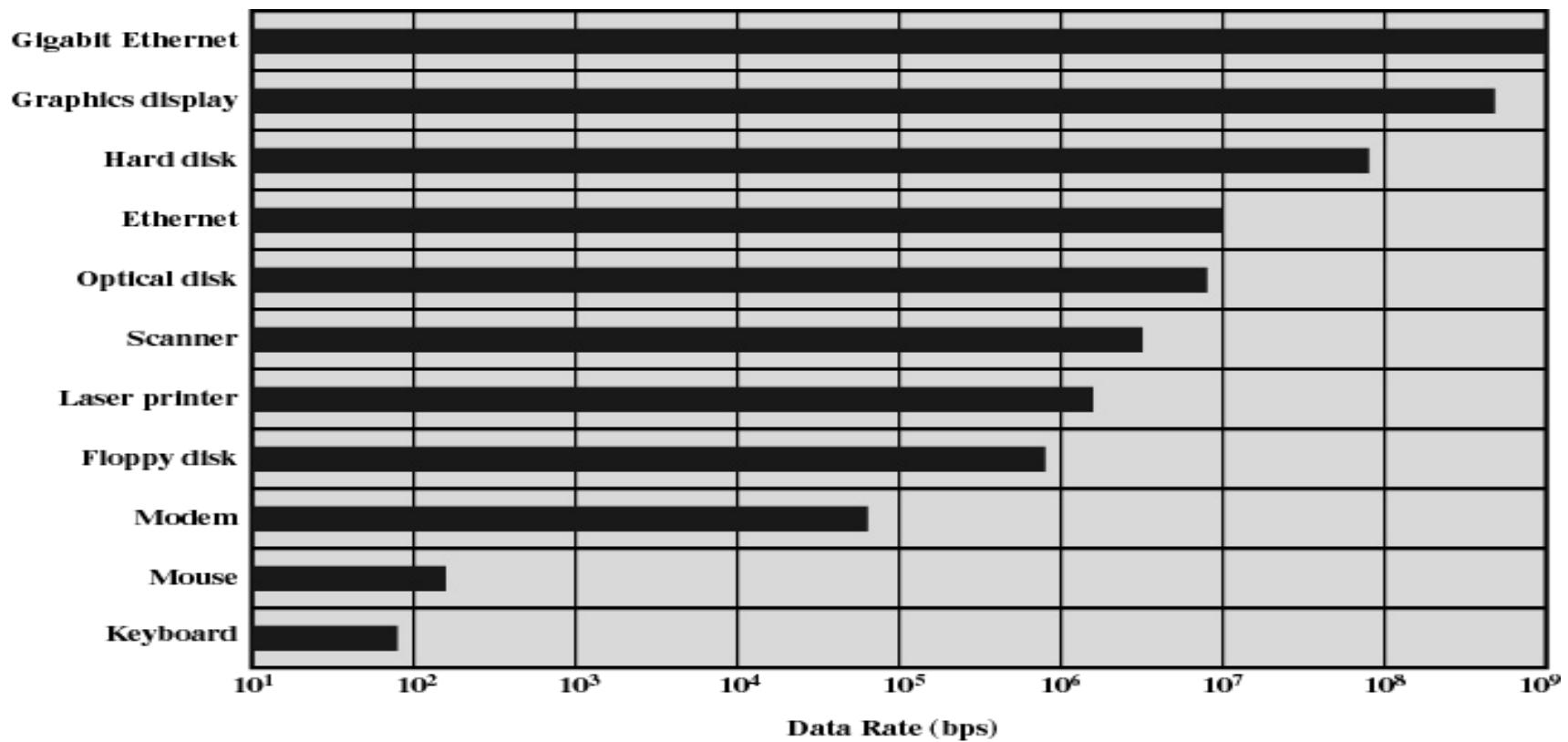
- 用户交互设备
 - 打印机
 - 显示器
 - 键盘
 - 鼠标
- 存储设备
 - 磁盘
 - 磁带
- 通信设备
 - 以太网卡
 - Modem
- 专用设备
 - 飞机操纵杆
 - 踏板
- . . .

调制解调器

I/O设备物理特征

- 数据传输模式
 - 字符设备 vs 块设备
Keyboard block: 磁盘
- 访问模式
 - 随机访问 vs 顺序访问 (-逐从头)
- 传输调度
 - 同步 vs 异步
- 独占性
 - 共享 vs 独占
- 速度
 - 几字节每秒~几G字节每秒
- 读写特性
 - 只读, 只写, 读写

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read/write	CD-ROM graphics controller disk



I/O设备的传输速度

I/O设备特征总结

异构的

- I/O设备的种类和数量越来越多
- I/O设备与主机的联络和信息交换方式各不相同
- I/O设备的传输速度差别很大

限速传输

5G

I/O子系统设计的目的

• 通用性

- I/O设备的多样化，操作方式各不相同，物理特性各异
- 增加新的I/O设备而无需修改操作系统 对上层不需要改变
- 向应用程序和操作系统上层软件隐藏I/O操作的细节

驱动程序：适配器

API

↑
物理接口

• 高效性

- I/O设备的速度远远低于CPU和Memory的速度，是系统的瓶颈所在
- 提高I/O设备的并行性和效率能大大提高系统的整体性能

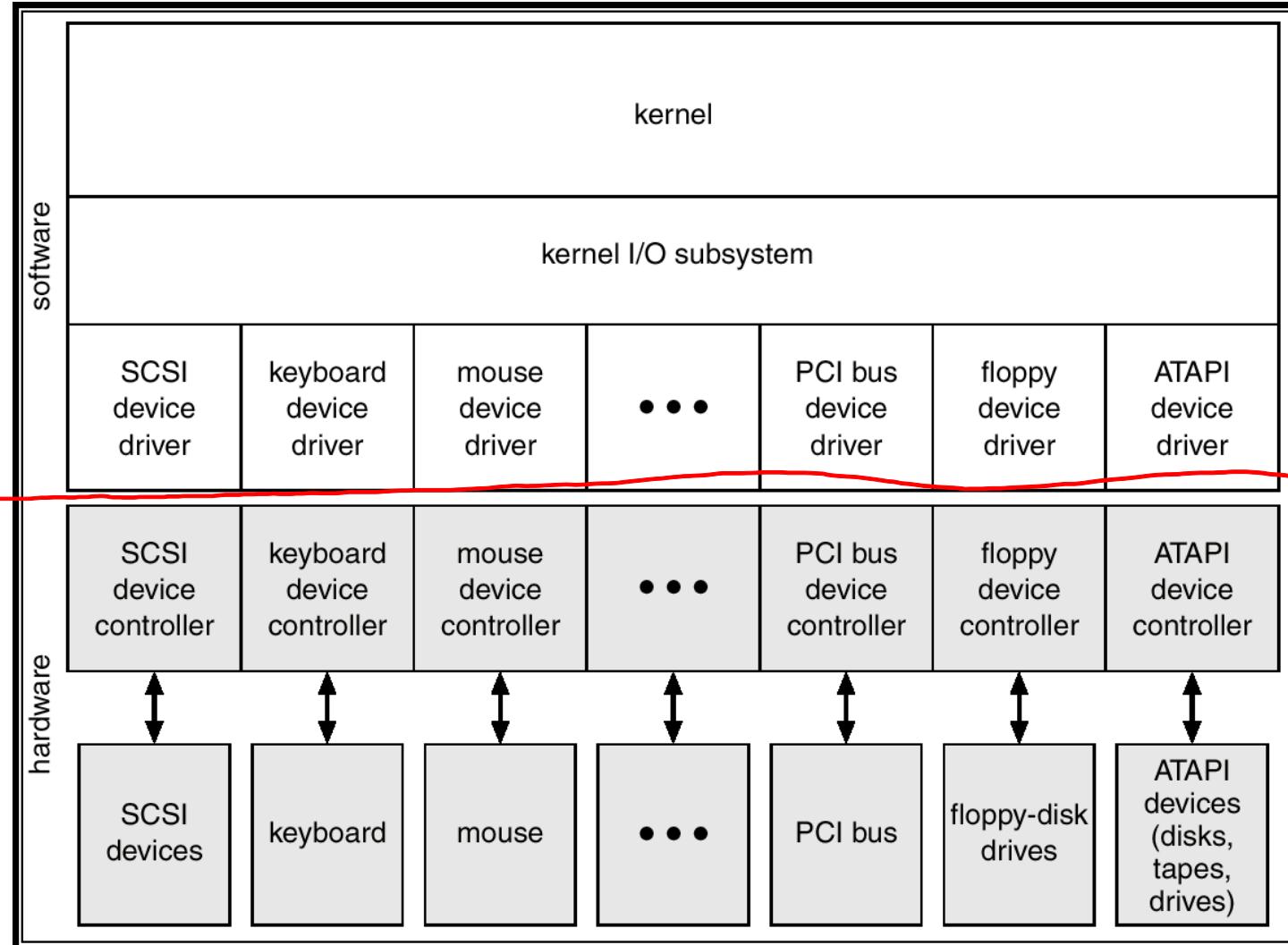
技巧：批量写

I/O子系统应具备的功能

- 设备中断处理
- 缓冲区管理
- 设备分配和去配
- 设备驱动调度
- 虚拟设备及其实现

I/O系统的分层

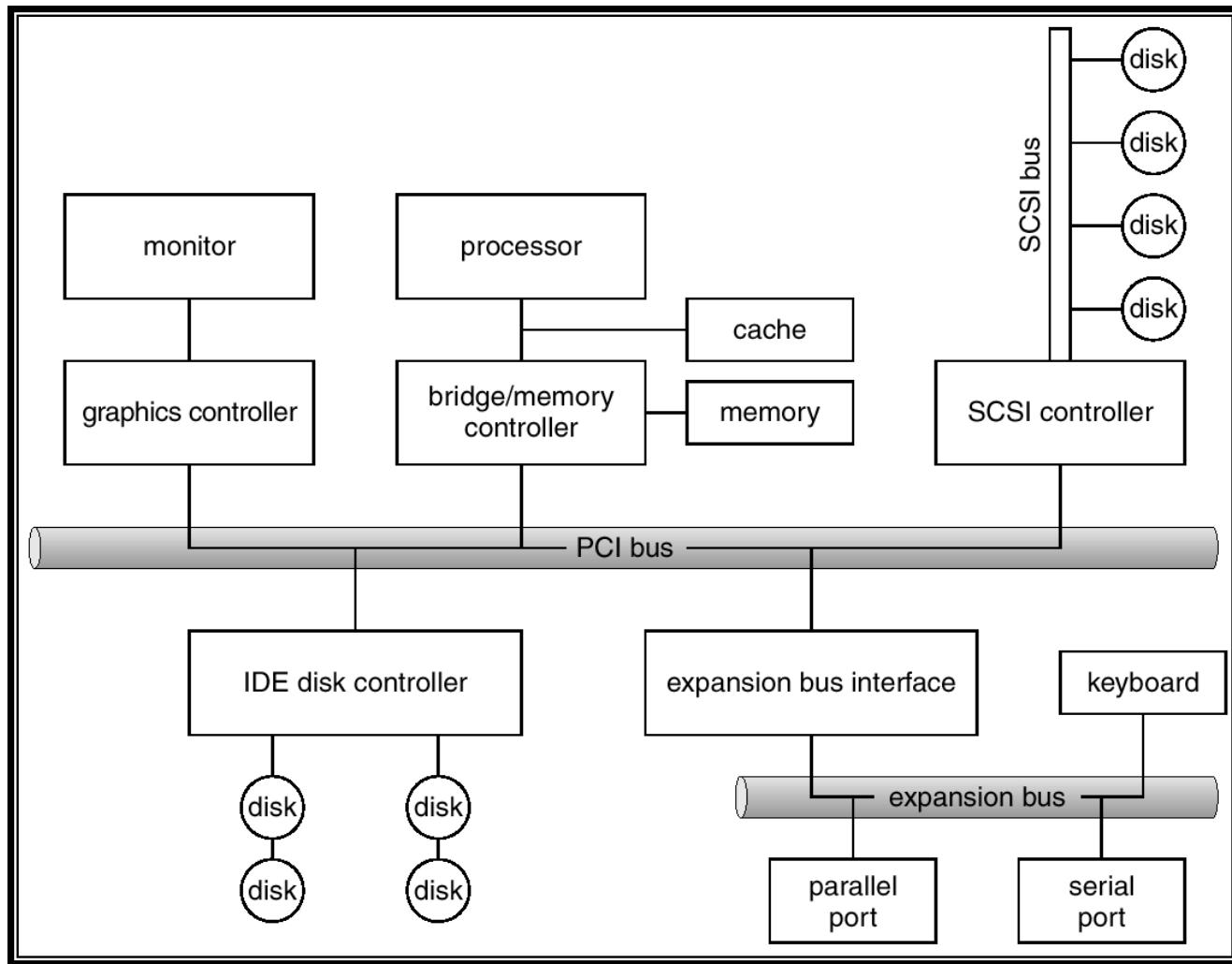
- I/O硬件
 - I/O设备（机械部件）
 - 设备控制器（电子部件），又称适配器
 - 中断装置
 - I/O软件
 - 中断处理程序
 - 设备驱动
 - 与设备无关的内核I/O子系统
- } 与设备相关



I/O硬件的通用概念

- **连接端口 (Port)**
 - I/O设备与机器的连接点
 - 如串行口、并行口
- **总线 (Bus)**
 - 多个设备共用的一组数据线/地址线/控制线
 - 一组严格的**协议**, 规定什么类型的消息可以发送到线上
 - 例如, PC BUS, 扩展总线, SCSI BUS
- **设备控制器 (Controller)**
 - 一组用于操作端口、总线或设备的电子部件
- **I/O端口/寄存器**
 - 用于简化**CPU**和设备通信的一组寄存器, 又称**I/O Port**

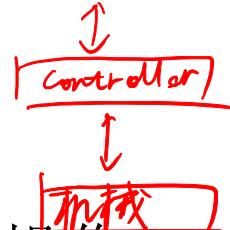
时序
语义
语法



I/O设备、设备控制器、总线以及内存、CPU的关系

设备控制器(controller)

- Controller是操作系统与设备的机械部件之间的接口
- Controller有一组寄存器用于数据传输和信号控制
- 操作系统与Controller的两种交互方式
 - ①通过特殊的I/O指令读/写controller的一组寄存器 学每个I/O的指令open, read, write
 - ②**Memory-mapped I/O**: 将寄存器映射到内存空间，通过内存读写指令来实现 用操作内存的方式操作I/O. OS
- Controller与设备之间的交互
 - 很底层的接口
 - 如磁盘控制器将一串比特流转换成字节块，并执行错误检测
- 引入控制器简化了系统的设计，使CPU从繁杂的设备操作中解放出来



设备控制器

Controller

- 设备控制器的主要功能
 - 接收和识别CPU或通道发来的命令
 - 例如磁盘控制器能接收读、写、查找等各种命令
 - 实现数据交换
 - 设备和控制器之间的数据传输
 - 控制器和主存储器之间的数据传输
 - 发现和记录设备及自身的状态信息，供CPU使用
 - 设备地址识别

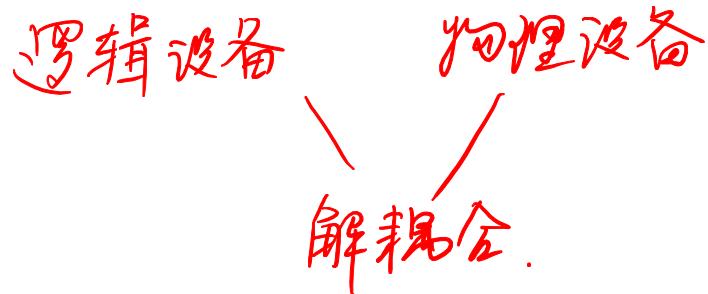
I/O 端口

- 设备控制器的一组寄存器称为I/O端口
- 寄存器的种类
 - 状态寄存器
 - 报告设备的状态，如当前命令是否执行完毕，数据是否可用，设备是否存在错误等
 - 控制寄存器
 - 存放控制命令
 - 输入数据寄存器
 - 存放从设备获得的输入数据
 - 输出数据寄存器
 - 存放从计算机向设备发送的输出数据
- 寄存器的大小通常为1-4字节

I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

I/O 端口地址例子

I/O软件的设计目标



- 设计目标
 - 通用性
 - 高效性
- 设计需考虑的问题
 - 设备无关性 *互独立*
 - 出错处理
 - 同步/异步传输
 - 缓冲技术

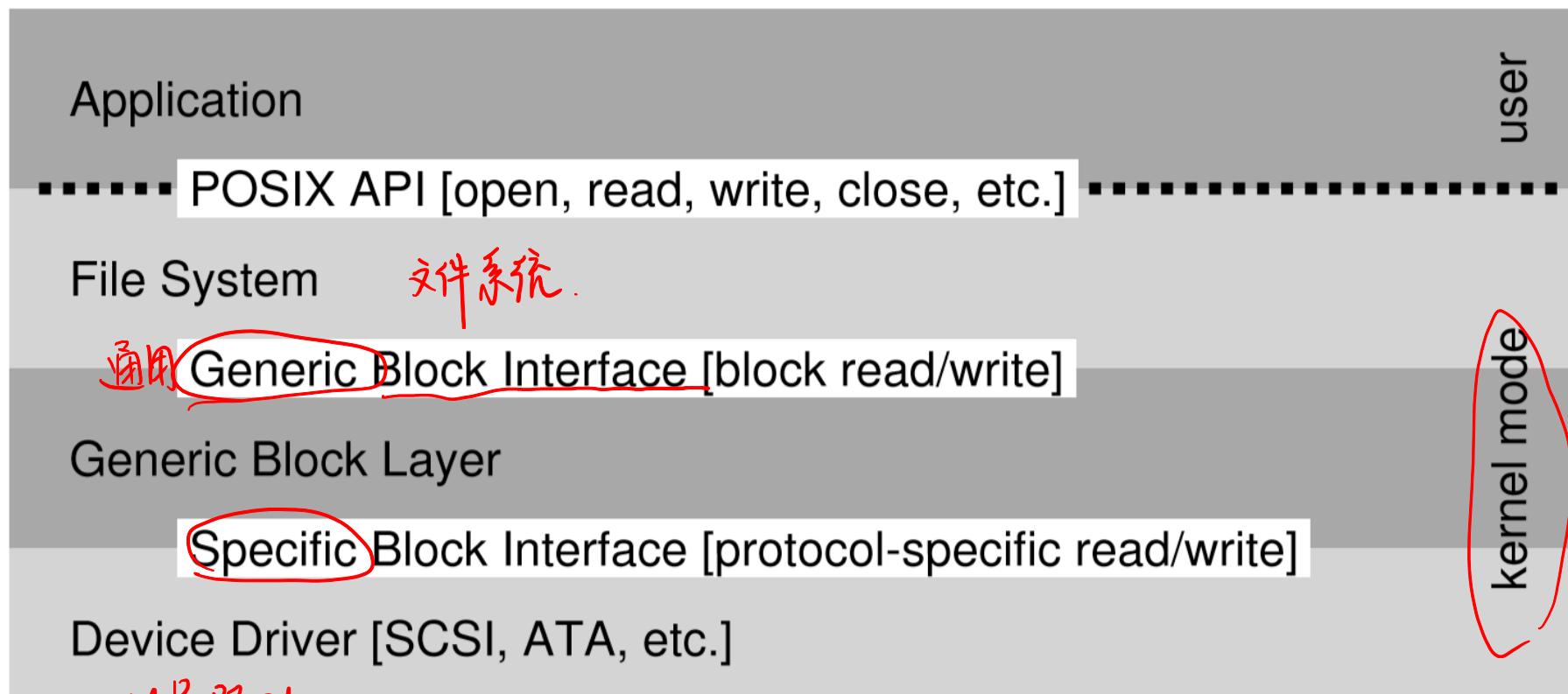
I/O软件

- I/O中断处理程序
 - 设备驱动程序
 - 独立于设备的I/O软件
- } 设备相关

设备驱动程序

- 实现I/O子系统通用性目标的关键技术
- 作为独立于I/O设备的软件和具体I/O硬件细节之间的适配器，对上层软件提供标准的接口
 - API
 - ↑
 - 标准接口
- 实现方法
 - 抽象、分类
 - 封装
 - 软件分层
- 优点
 - 方便了操作系统上层软件开发人员
 - 方便了设备制造商
 - 由于不同操作系统定义的标准不同，设备厂商仍需为不同操作系统提供不同的设备驱动程序

文件系统的层次



设备驱动程序(cont)

OS一大块 { 生僻逻辑 }

内核的一部分

- 向下与I/O设备紧密相关的代码，用于控制I/O设备，通常由设备制造商提供
 - 每个设备控制器都有一组寄存器，用于接收命令、报告状态和传输数据。
 - 但寄存器的个数、命令的种类因设备不同而不同
- 设备驱动程序通常是操作系统内核的一部分
 - 在操作系统执行时动态载入
 - 现在操作系统约70%的代码都是驱动程序
- 每个驱动程序处理
 - 一种设备类型 (如, 鼠标)
 - 一类紧密相关的设备

设备驱动的功能

- 从上层（设备无关层）接收抽象的read, write等请求；
- 初始化设备；
- 检查输入参数的合法性。
- 将合法的输入从抽象的表示转换成具体的表示
 - 例如，将线性的逻辑块号转换成柱面号、磁头号、扇区号
- 检查设备的状态
- 发出一串I/O操作命令来控制设备，设备驱动决定要发出哪些命令。

I/O设备接口类型

分类：

- ① 块设备和字符设备接口 读写操作字节流
- ② 网络设备接口
- ③ 时钟和定时器接口 异步/同步
- ④ 阻塞和非阻塞I/O接口

块和字符设备接口

- 块设备接口对访问磁盘或其它块设备的行为进行了抽象，常用接口包括：
 - `read()`, `write()`
 - `seek()`: 针对支持随机访问的设备 顺序访问 X
 - `raw I/O`
 - 将块设备作为块数组来访问，适用于操作系统自身和数据库管理系统
 - `Memory-mapped I/O` 内存映射 store, load.
 - `Memory map`的系统调用返回文件拷贝的虚拟地址
 - 采用内存访问指令来访问文件（外存）缺页中断
 - 真正的数据传输基于虚拟内存的请求分页机制来实现
- 字符设备接口对字符流方式的设备（如键盘）行为进行了抽象，常用接口包括：
 - `get()`, `put()`: 获取或写一个字符
 - 基于`gets()`/`puts()`的按行读写接口

网络设备接口

- 最常用的是网络套接字接口，对网络I/O的行为进行了抽象，具体包括：
 - `connect()`: 建立连接
 - `Read()`, `write()`: 读写数据 *recv()* *send()*
 - `Listen()`: 监听远程连接请求
 - `Select()`: 用于管理一组socket，当其中有一个状态发生变化时返回

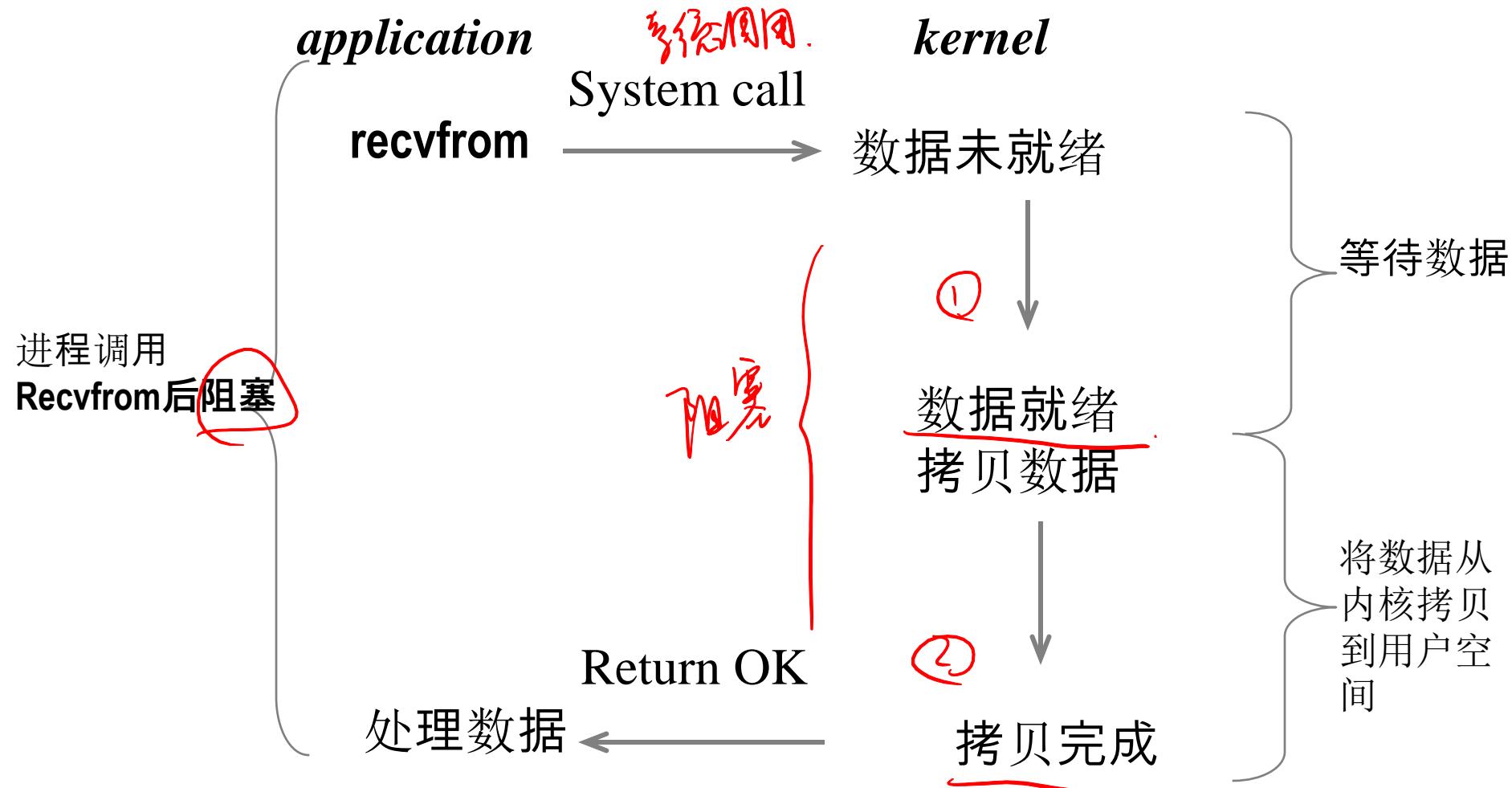
时钟和定时器接口

- 对计算机硬件提供的时钟和定时器行为进行了抽象，一般包括下面的接口：
 - 获取当前时间
 - 获得已经过去的时间（一般以1970年1月1日为基准）
 - 设置定时器，在T时刻触发事件X

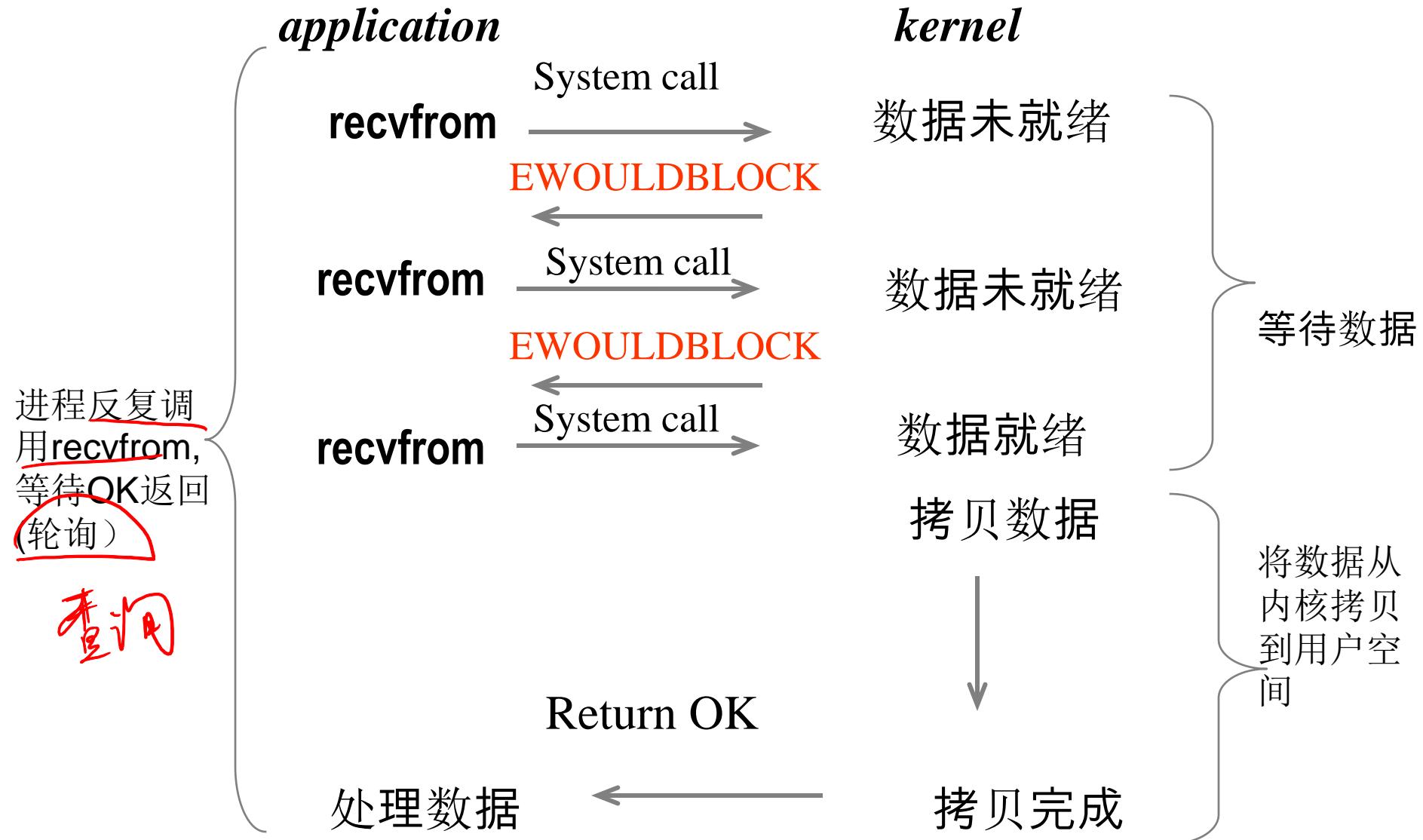
I/O系统调用模式

- 阻塞I/O (blocking I/O)
- 非阻塞型I/O (non-blocking I/O)
- I/O复用 (IP multiplexing)
- 异步I/O (Asynchronous I/O)

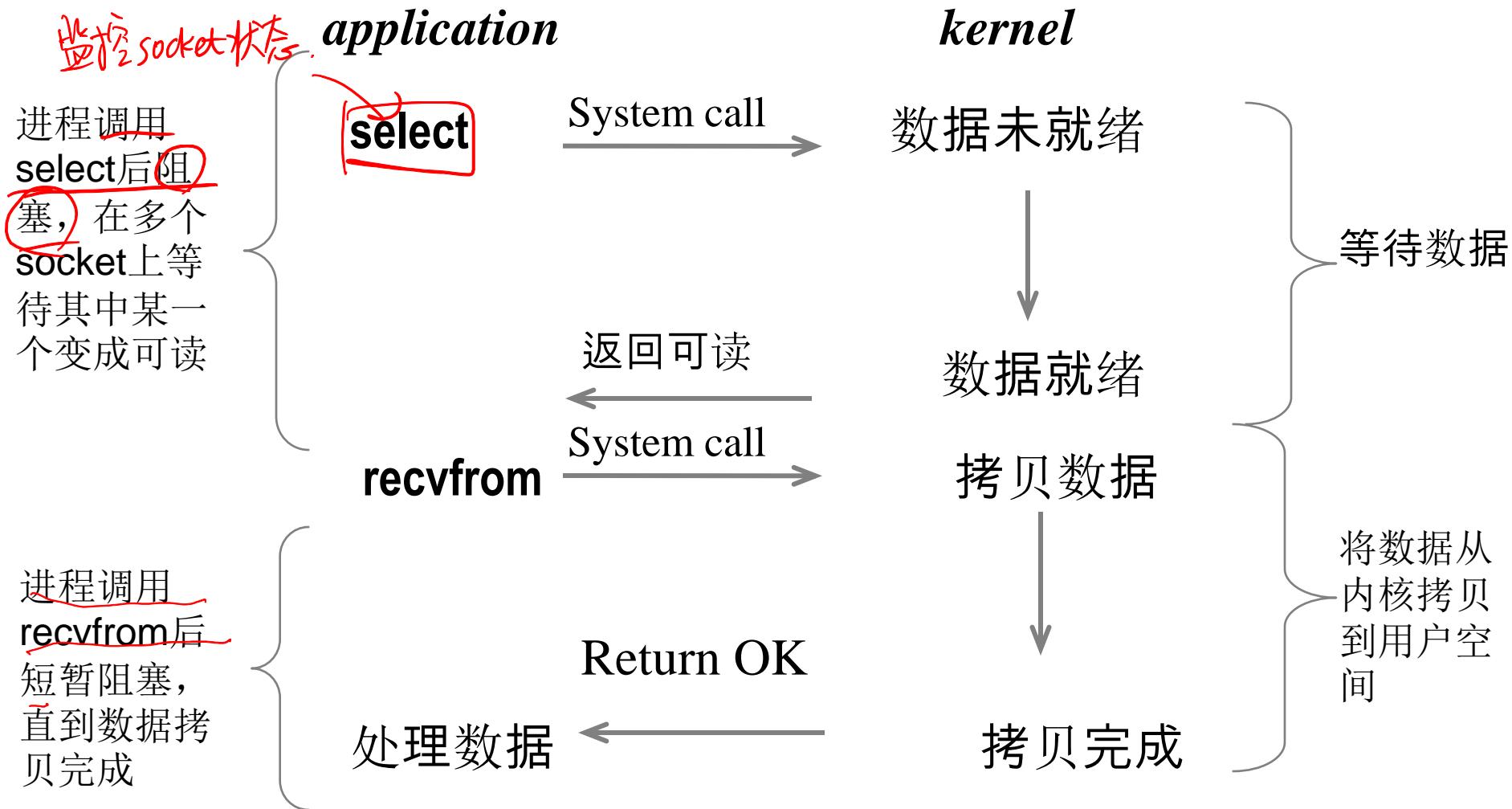
阻塞型I/O



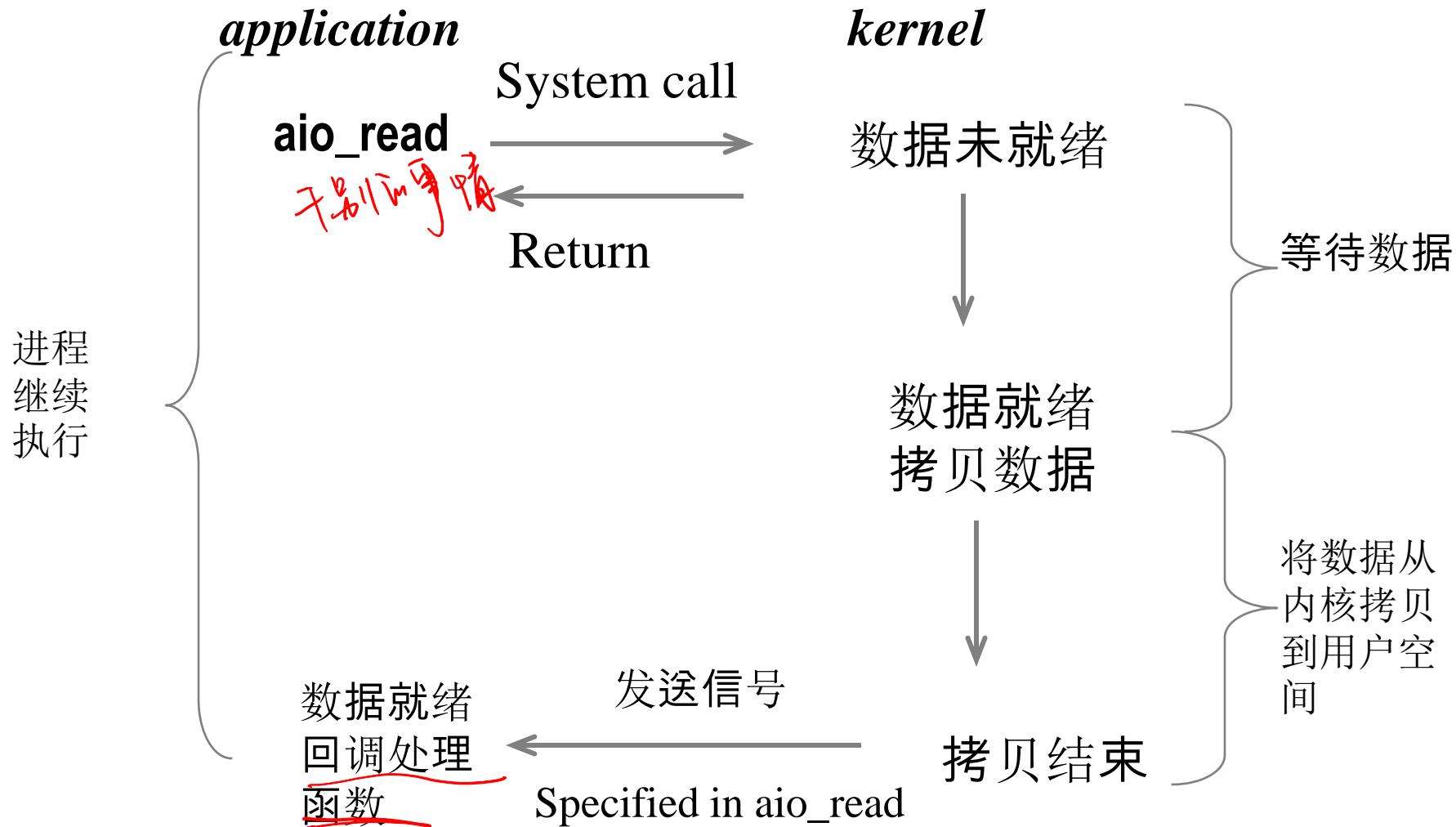
非阻塞I/O



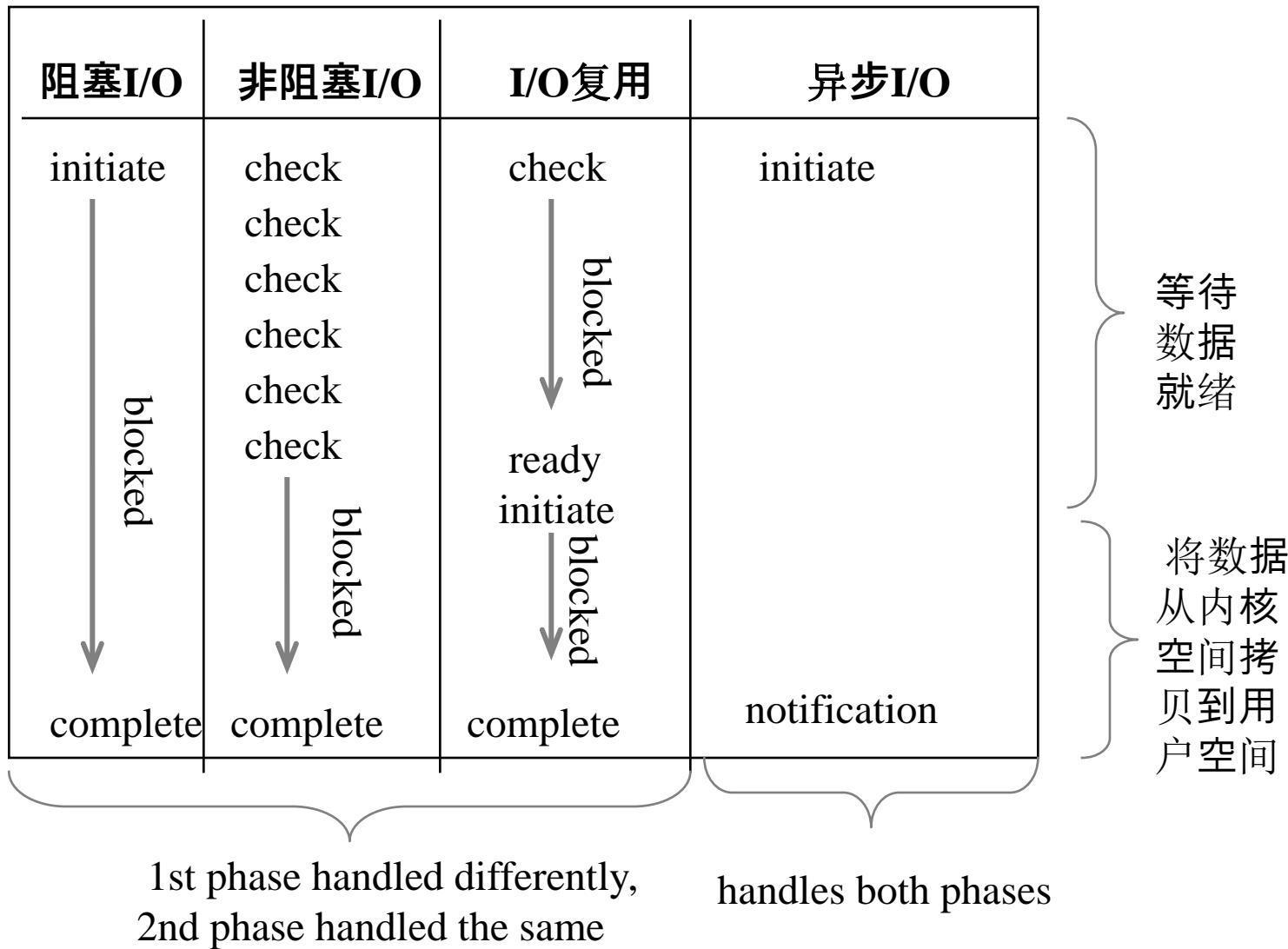
I/O复用(select and poll)



异步I/O



I/O调用模型比较



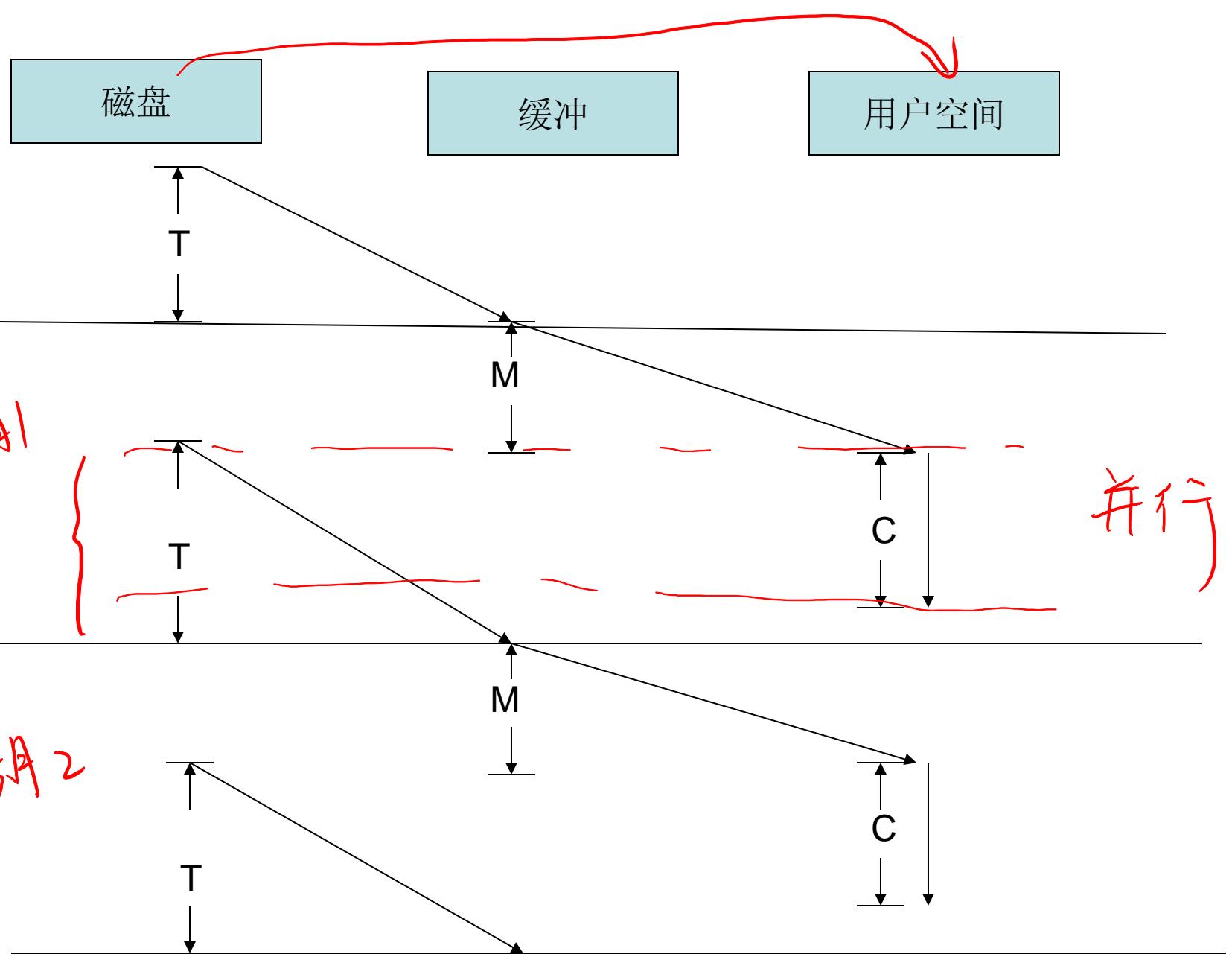
独立于设备的I/O软件

- 缓冲(buffering)
- 缓存(caching)
- 设备虚拟化Spooling
- I/O调度

这些技术均是为提高**I/O系统的性能服务的**

缓冲 buffering

- 缓冲技术主要用于解决 ^{① 振动, 在某瞬间(不长时期)}
– CPU、内存和设备之间的 速度不匹配 问题
 - 如输入缓冲
- 解决 传输数据大小不匹配 问题
 - 如 网络的分包与重组
 - 物理记录大小与逻辑记录大小不一致
- 常用的缓冲技术
 - 单缓冲 数量
 - 双缓冲
 - 多缓冲
- 缓冲是一个典型的消费者-生产者问题



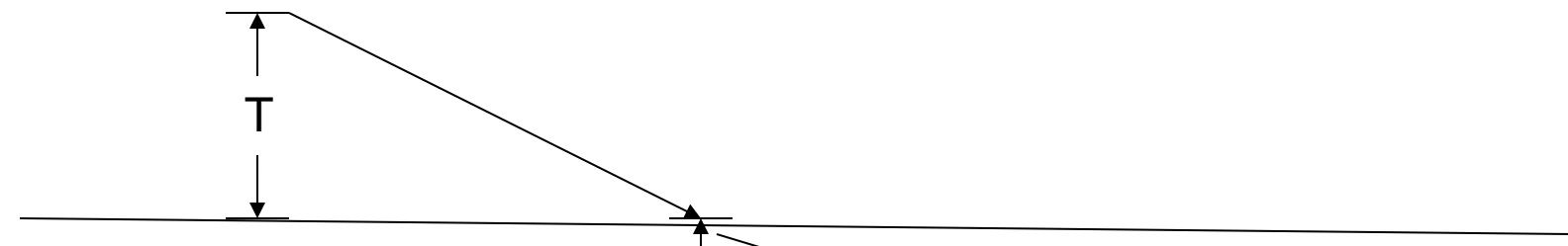
若 T>C, 则每批数据处理时间为T+M

磁盘

缓冲

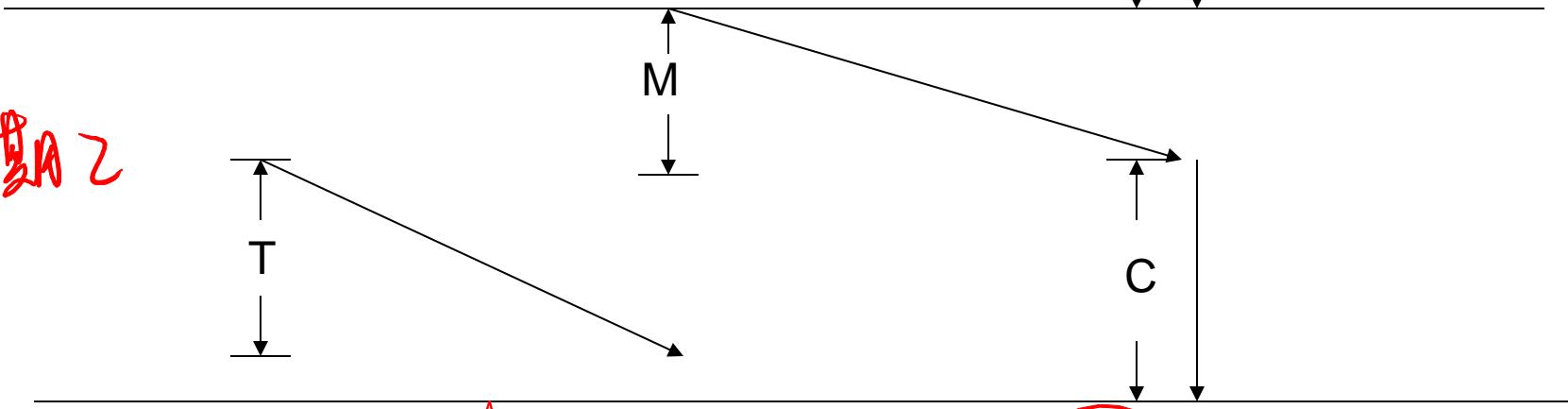
用户空间

周期



读写
过程

周期 2



若 $T < C$, 则每批数据处理时间为 $C+M$

C+T不行

- 采用单缓冲，每批数据处理时间为 $\max(C, \underline{T}) + M$
- 不采用缓冲，每批数据处理时间为 $C+T$
- 一般 $M \ll C$, $M \ll T$,因此采用单缓冲后，处理速度会快许多

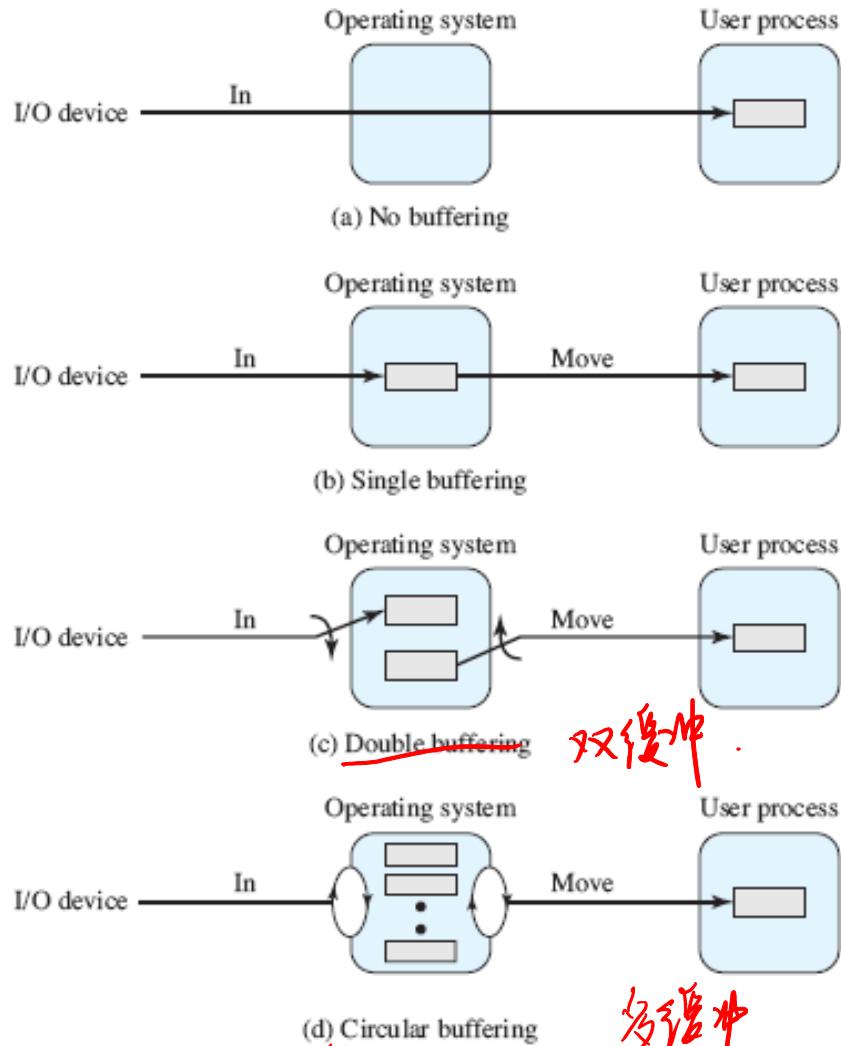


Figure 11.5 I/O Buffering Schemes (input)

缓冲(cont)

- 假设从Modem接收一个文件，存储到硬盘
- Modem 比硬盘速度慢上千倍；
- 可以在内存开辟一块缓冲，收集从Modem接收到的数据；
- 当缓冲满时，一次性写入硬盘；
- 由于硬盘写操作并非瞬间完成，这期间仍需要接收 Modem来的操作，可以采用双缓冲解决，即在将一个缓冲满写入硬盘时，用另一个缓冲接收Modem的数据
- 多缓冲通常用在高速设备向低速设备写数据的场景
 - 但一旦数据产生的平均速率大于低速设备的写入速度，则依然需要产生等待

缓存

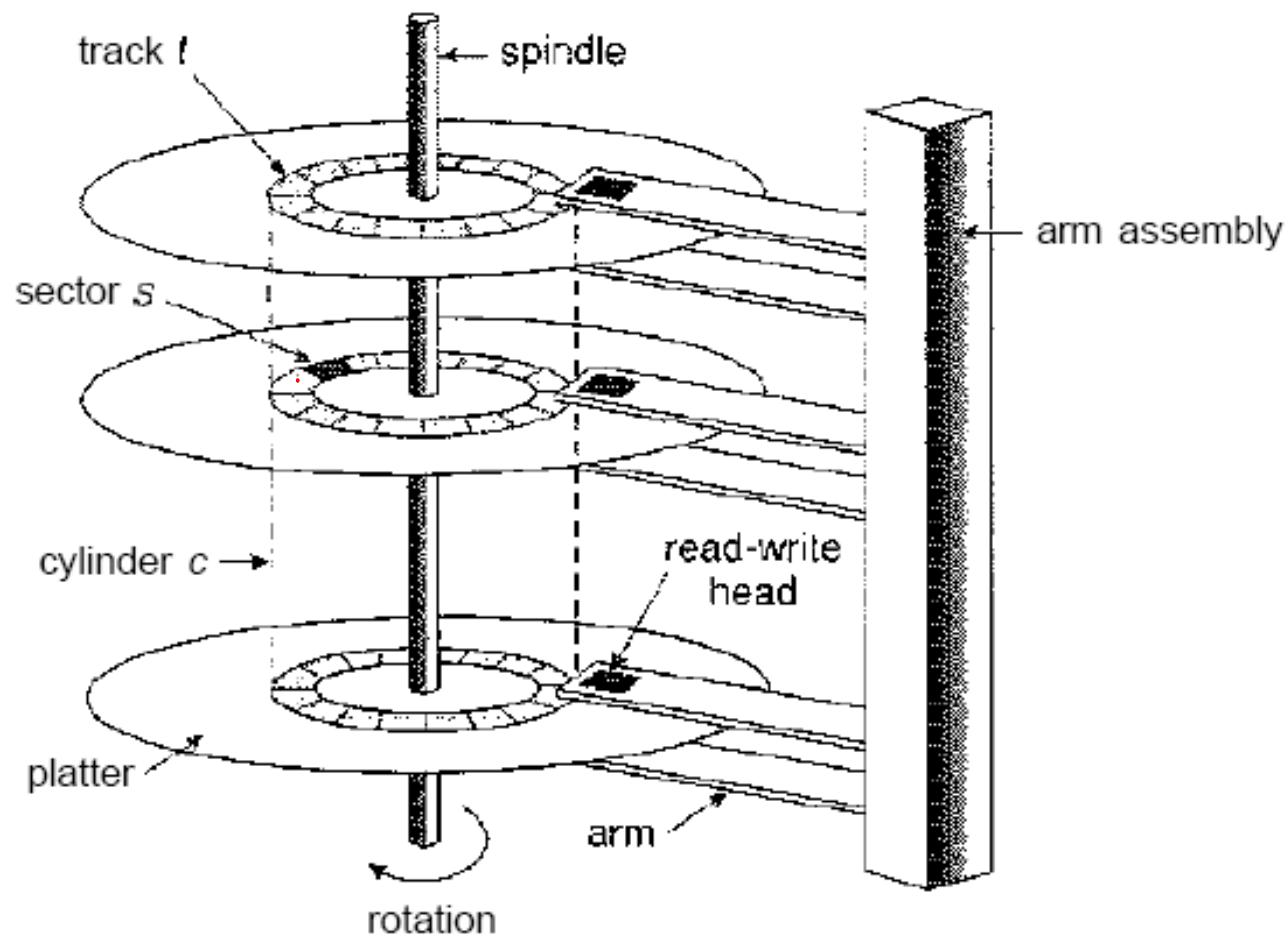
- 缓存是存放数据拷贝的快速存储区域
- 访问缓存要比访问数据原始存放位置更高效
- 缓冲和缓存的区别
 - 缓存的数据一定在其他地方存在备份
 - 缓冲的数据可能在整个系统中仅存在一份 *丢就没了*
- 磁盘缓存
 - 内存中用于存放磁盘扇区数据的一块存储区
 - 当发起I/O操作时，首先检查所请求的扇区是否在磁盘缓存区
 - 磁盘缓存可以采用的替换算法
 - LRU
 - LFU
 - ...

I/O调度

- 应用程序发出的I/O操作请求是随机的，如果每次请求都启动I/O，那么代价会很大
在一定的时间间隔 buffer请求
- 因此，通常会延迟请求的处理，批量处理一批I/O请求
- 操作系统应确定执行这些请求的顺序，优化性能
 - 假设，磁盘的磁头当前处于磁盘最外面的磁道
 - 应用程序1请求最里面的磁道的数据，应用程序2请求靠近外边的磁道的数据，应用程序3请求中间磁道数据
 - 则操作系统可以采用2, 3, 1的调度方式来减少磁头的移动距离
- 每个设备维持一个请求等待队列，应用的请求被置于等待队列，I/O调度器对队列重新排序，排序的准则包括：
 - 系统性能
 - 公平性
 - 响应时间
 - 优先级*性能评价指标*

磁盘调度

- 磁盘的构成
 - 盘片(platter)
 - 磁道(track)
 - 柱面(cylinder)
 - 扇区(sector)
 - 移动臂(arm)
 - 读写磁头(head)
 - 轴(spindle)



盘面数据的分布

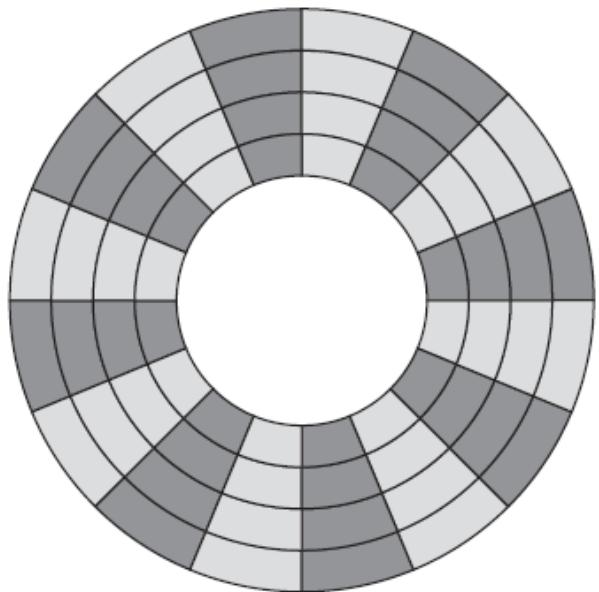
- **CAV: 固定角速度**

- 无论磁头在哪里，轴以固定的角度旋转
 - 每个track包含的扇区数一样，从而数据量一样
 - 内磁道的数据密度比外磁道的高（外磁道空间浪费）
 - 硬盘

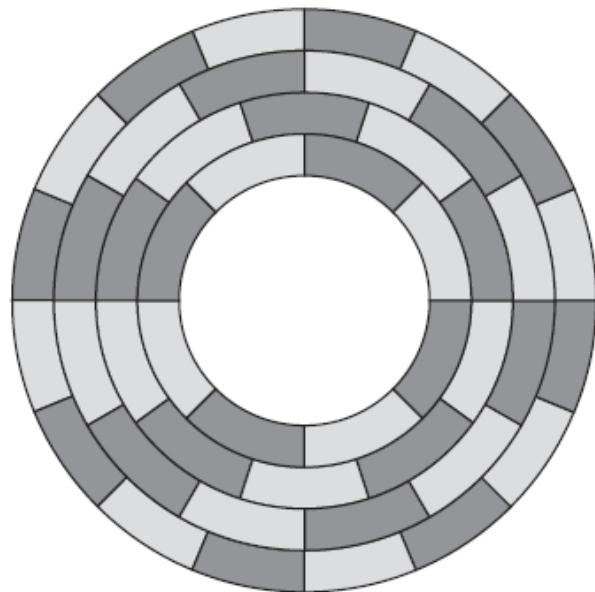
注：现代的高容量磁盘一般把磁盘的磁道分成多个区，每个区包含若干连续的磁道，其中每个磁道的扇区数一样

- **CLV: 固定线速度**

- 磁头靠近内磁道时，旋转角速度快，磁道靠近外磁道时，旋转角速度慢，保持线速度一样。
 - 每个track的数据密度一样，因此外磁道的扇区及存储的数据比内磁道的多
 - CD-ROM、DVD-ROM中使用



CAV



CLV

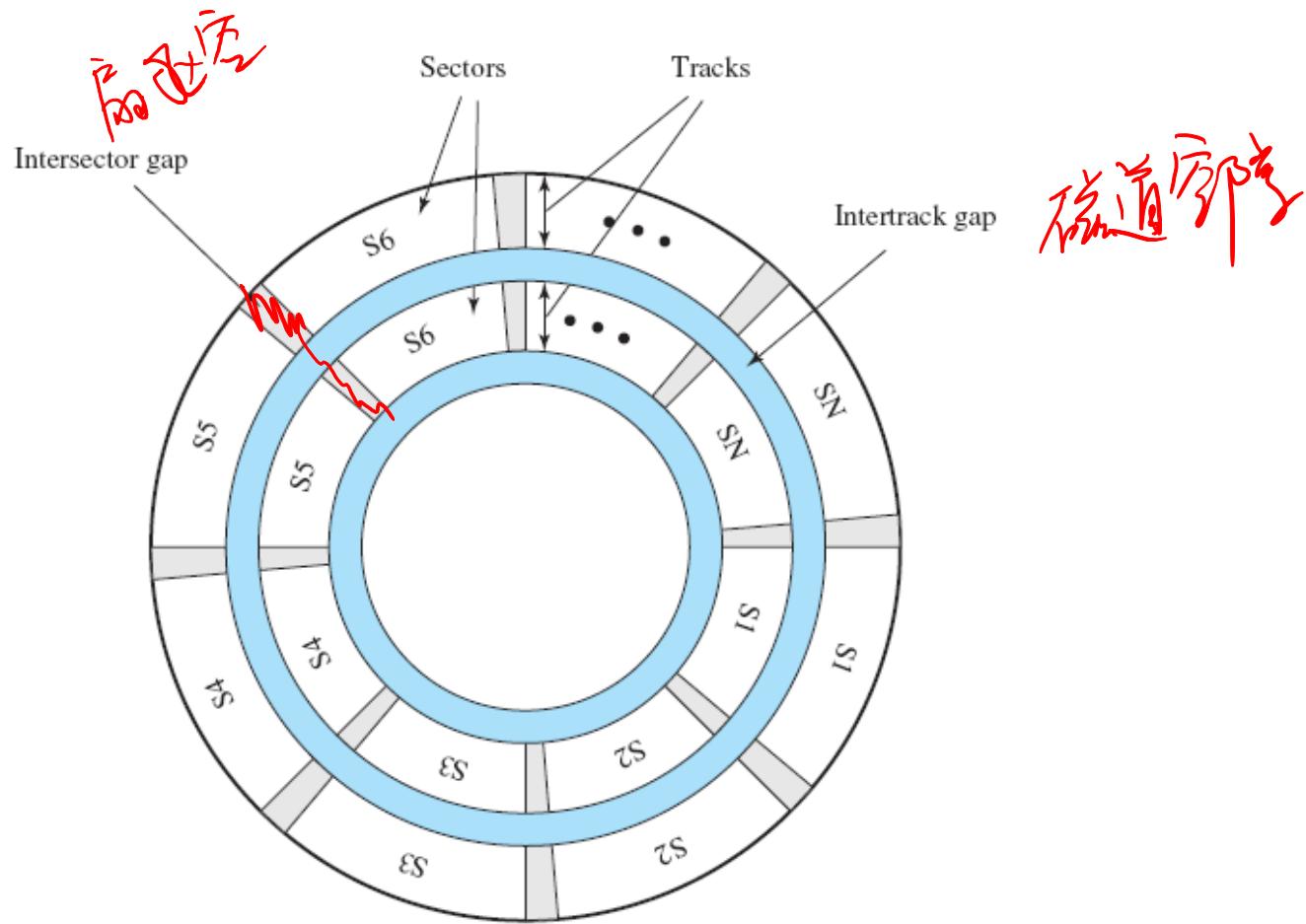


Figure 11.16 Disk Data Layout

扇区与扇区之间有空隙，越外围的磁道，空隙越大。空隙中包含扇区标识的信息，在格式化时写入

磁盘块的编址

柱面号 (圆半径大)
磁头号
扇区号

- 逻辑上看成是一维的磁盘块的数组
- 编址方案
 - 最外道柱面的第一个磁道的第一个扇区为0号块，第二个扇区为1号块
 - 当第一个磁道编完后，接着为同一柱面的第二个磁道编号
防止 arm 向里移动
 - 当一个柱面的所有磁道都编完号后，为下一个柱面编号

磁盘格式化

- 一个新的磁盘是空的，无法存储数据

① 低级格式化/物理格式化

- 将磁盘分成扇区，使得磁盘控制器能读写数据
- 每个扇区是一个特殊数据结构
 - Header
 - data area: 通常是512字节
 - Trailer
- header, trailer包含磁盘控制器所需的信息，如：
 - 扇区号
 - 纠错码(ECC): 每次读写扇区时，设备控制器自动处理ECC

② 分区与逻辑格式化

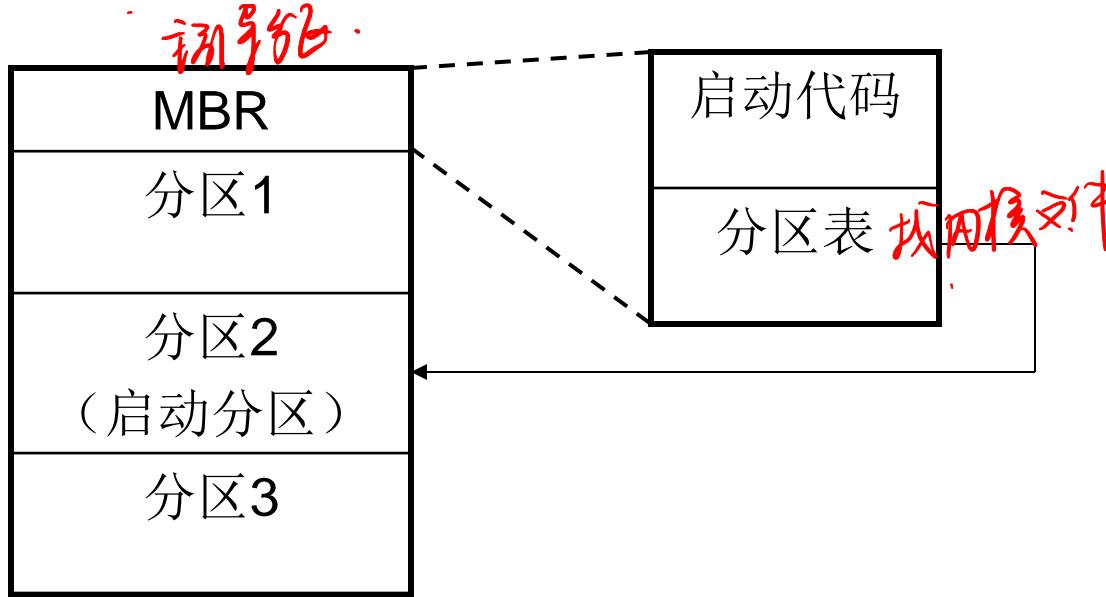
- 将磁盘的柱面分成一组或多组，每一组称为一个分区，操作系统可以将每个分区当作是一个单独的磁盘
- 操作系统将初始的文件系统数据结构存储到磁盘上，如：
 - 空闲和已分配空间表
 - 初始的空目录
- 每个分区可以应用不同的文件系统

NTPX
FAT32 } 文件系统

操作系统的启动流程

- ① 系统加电，执行 BIOS 中的程序
 - 识别可开机的设备（根据用户的设置）
- 执行 [主引导分区 MBR 中的引导加载程序]
 - 引导加载程序的目的是 加载内核文件？^{或31}
 - 引导加载程序可能提供启动菜单，支持 多操作系统
 - 引导加载程序可能只是根据用户的菜单选择，将加载功能转交到其他启动分区的启动扇区来执行
- 内核文件
 - 开始操作系统的功能

操作系统的启动流程



Window 2000的启动流程

- ① 执行ROM中的启动代码，指示系统读取**MBR**中的启动代码
- ② 执行**MBR**中的启动代码，并依据分区表确定从哪个分区启动操作系统
 - **MBR**位于磁盘的第一个扇区
- ③ 读取启动分区的第一个扇区（启动扇区）
- ④ 执行启动扇区的代码，载入操作系统**内核**代码
- ⑤ 执行操作系统代码

磁盘调度

- 磁盘物理记录的表示

- <柱面号, 磁头号, 扇区号>

~~机械操作~~ 磁盘数据访问时间的构成

- 寻道时间/查找时间 ~~机械操作~~

- 磁盘机根据柱面号控制移动臂做横向机械移动，带动读写磁头到达指定柱面所需的时间
 - 寻道时间取决于磁头当前所处柱面的位置与待寻址的数据所位于的柱面位置
 - 典型的平均寻道时间大约在10ms级别

- 旋转等待时间

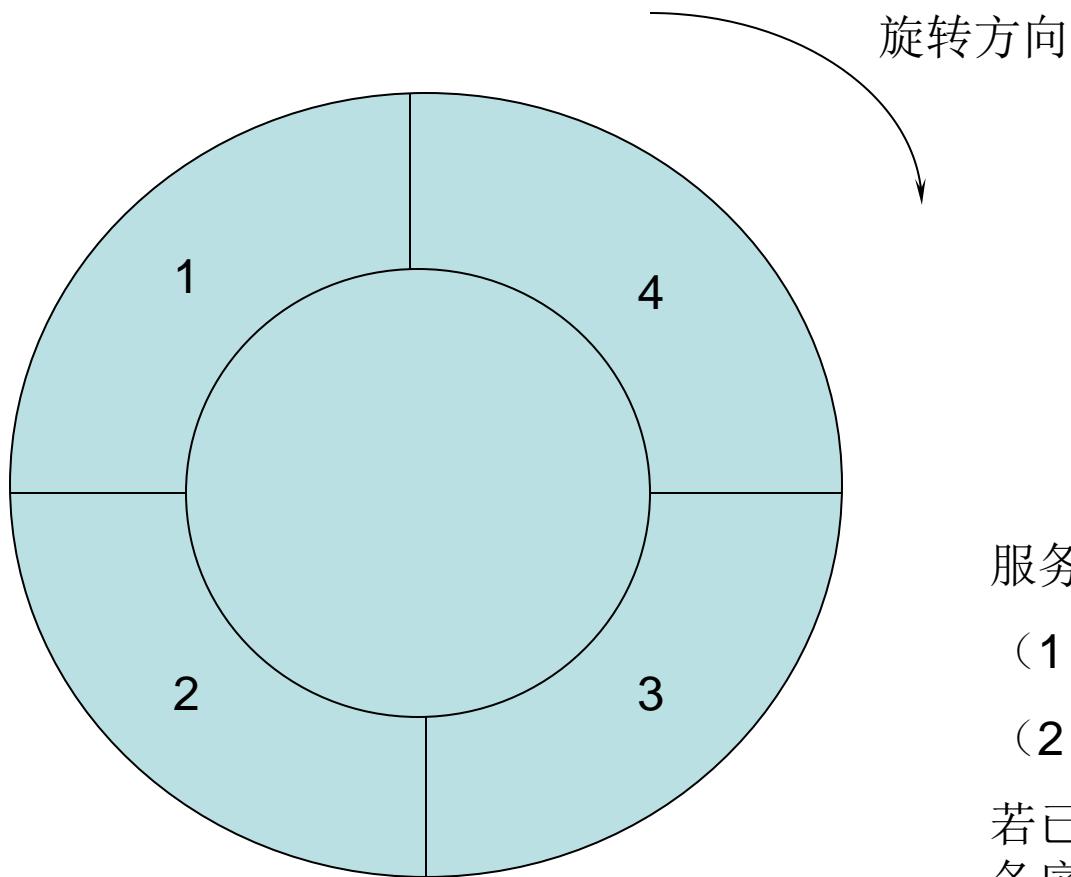
- 等待被访问的扇区旋转到读写磁头下所需的时间
 - 磁盘的旋转速度从3600转每分钟到15000转每分钟
 - 平均等待时间为磁盘旋转半圈的时间，在2ms级别

- 数据传输时间

$$T = \frac{b}{rN}$$

b为需要传输的数据, N为一个磁道的数据总量, r为旋转速度(转/秒)

1/2 圈
1/4 圈
1/8 圈
1/16 圈



请求序列: 4, 3, 2, 1

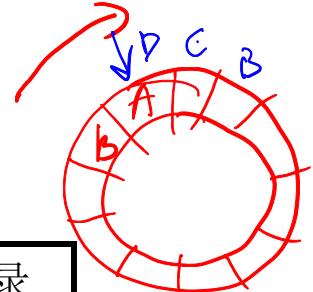
服务序列:

(1) 4, 3, 2, 1

(2) 1, 2, 3, 4

若已知当前读位置为3，则服务序列为4, 1, 2, 3更好

数据优化分布



物理块	逻辑记录
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J

物理块	逻辑记录
1	A
2	H
3	E
4	B
5	I
6	F
7	C
8	J
9	G
10	D

$$20/10 = 2 \text{ ms}$$

若旋转一周时间为20ms，读出每块后的处理时间为4ms，则右边的优化分布能提高信息处理速度

A读完后，B到A的位置

移动臂调度算法

- 对于一组磁盘请求，**寻道时间是优化整体性能的关键**
- 移动臂调度算法的目标
 - 最小化磁头移动的距离
- 移动臂调度算法
 - 先来先服务算法(FIFO)
 - 后来先服务(LIFO)
 - 最短服务时间优先算法(SSTF)
 - 扫描算法(SCAN)
 - 循环扫描算法(C-SCAN)
 - LOOK/ C-LOOK

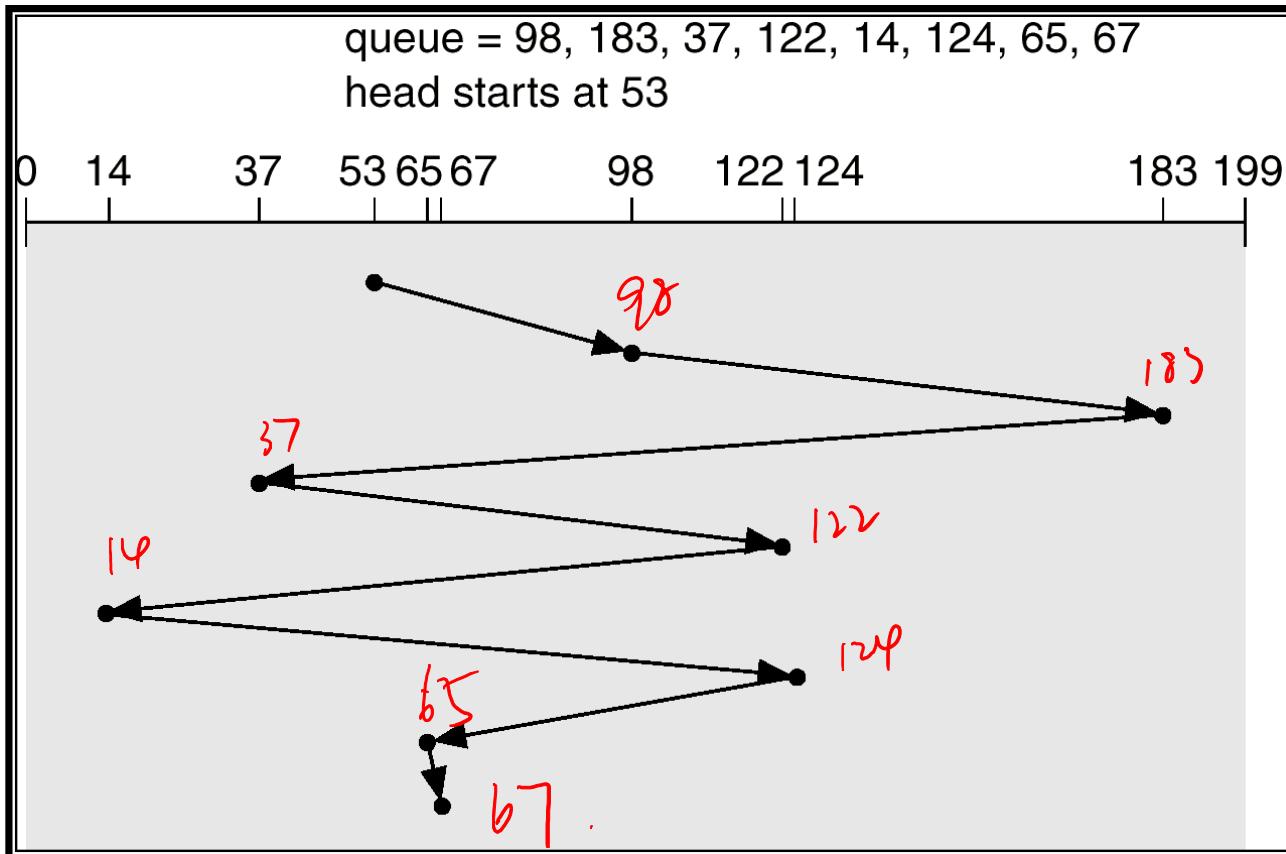
移动臂调度算法 (Cont.)

- 请求柱面号序列 (0-199).

98, 183, 37, 122, 14, 124, 65, 67

当前磁头所处的柱面号**53**

FCFS



磁头总计移动640个柱面

Short Service time First

SSTF (最短服务时间优先)

首次

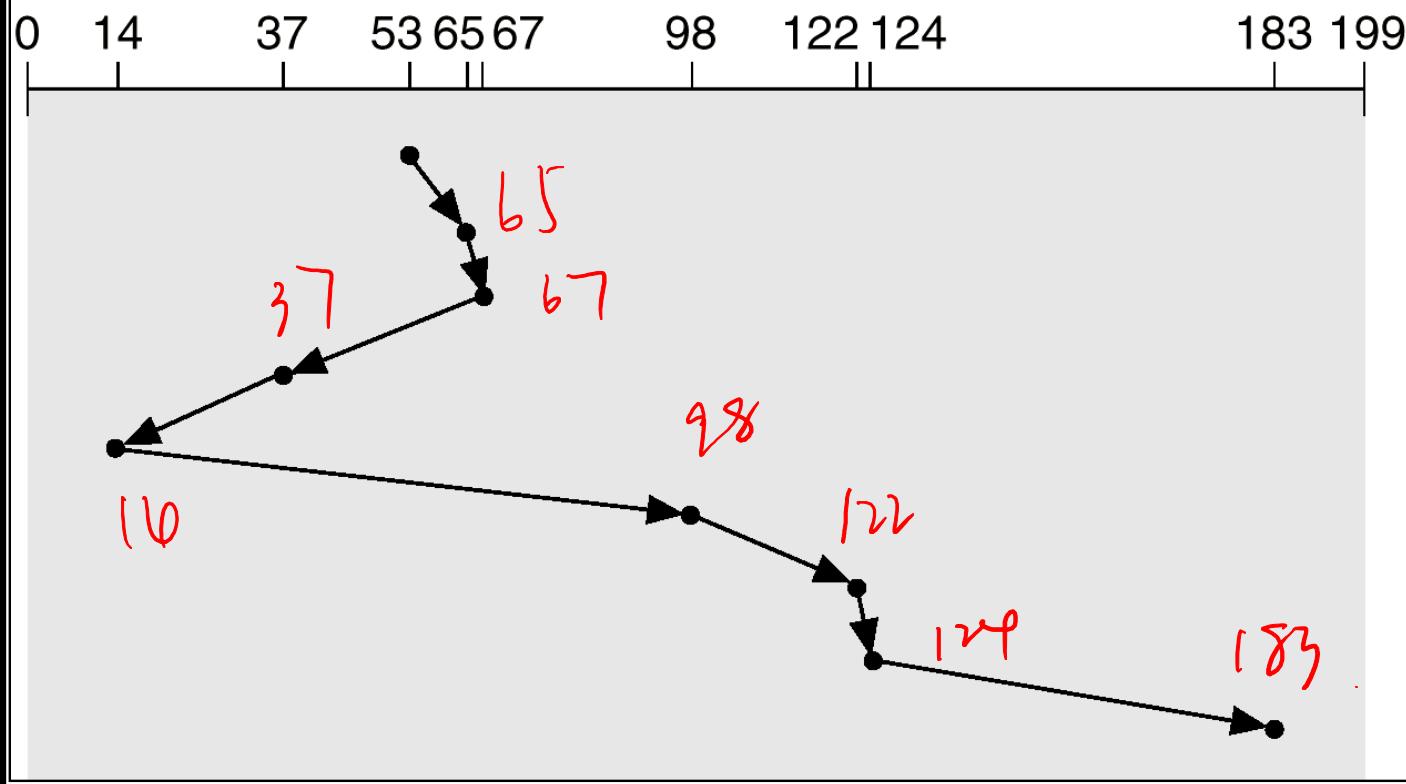
- 选择距离当前磁头位置最近的请求进行服务.
- SSTF是最短作业优先调度的一种表现形式，可能造成某些请求的饥饿

磁头移动不一定最短
局部最优

SSTF (Cont.)

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



磁头总计移动236个柱面

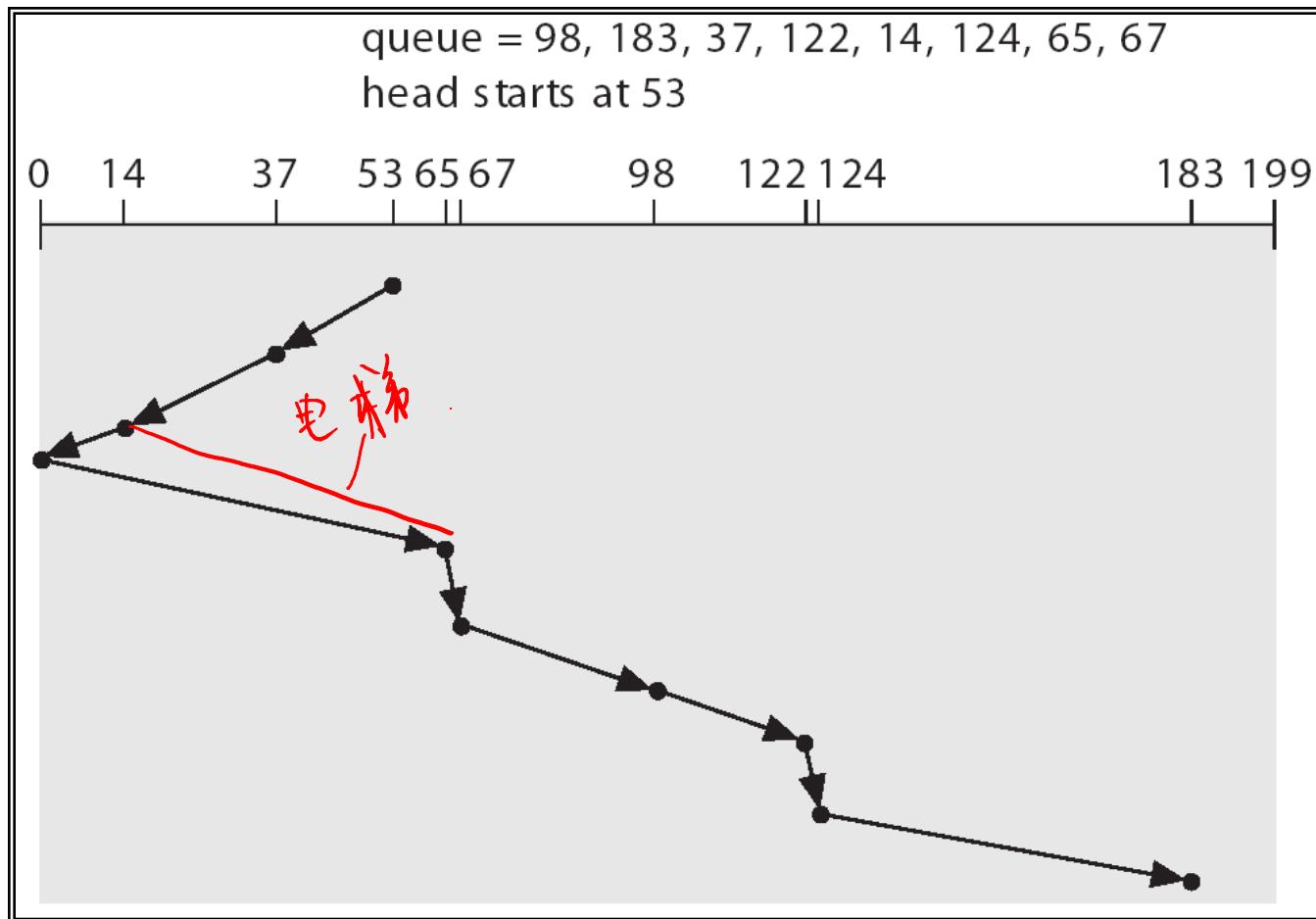
SCAN (扫描算法)

- 磁头总是朝一个方向移动，服务其中遇到的请求
- 当到达一端后，调转方向，向另一个方向移动，并服务遇到的请求.
- SCAN的改进--**电梯调度算法**
 - 并不每次沿着一个方向移动到末端，而是移动到当前移动方向没有请求为止

最差: 2x Total

中: Total

SCAN (Cont.)



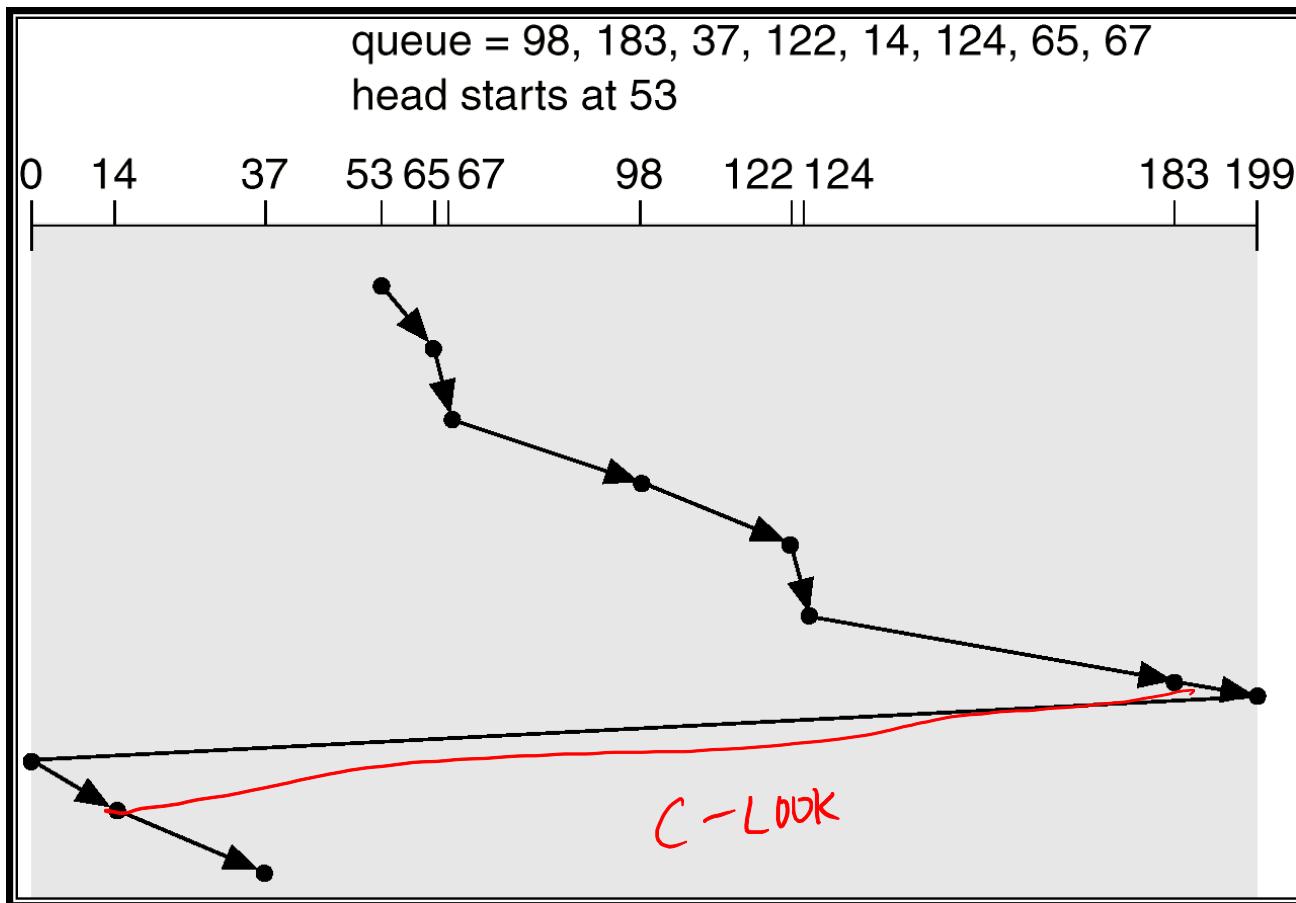
磁头总计移动 $53+183=236$ 个柱面 -28

若是电梯调度算法，则移动 $39+169=\boxed{208}$ 个柱面

C-SCAN (循环扫描算法)

- 磁头从起始位置开始，向一个方向移动，服务遇到的请求
- 当到达一端后，并不调转磁头移动方向，而是直接移动到起始位置，在返回过程中不服务任何请求
从0开始
- 本质上是将柱面看成是一个循环链表.
- 比SCAN算法提供了更均匀的请求等待时间.
均匀

C-SCAN (Cont.)

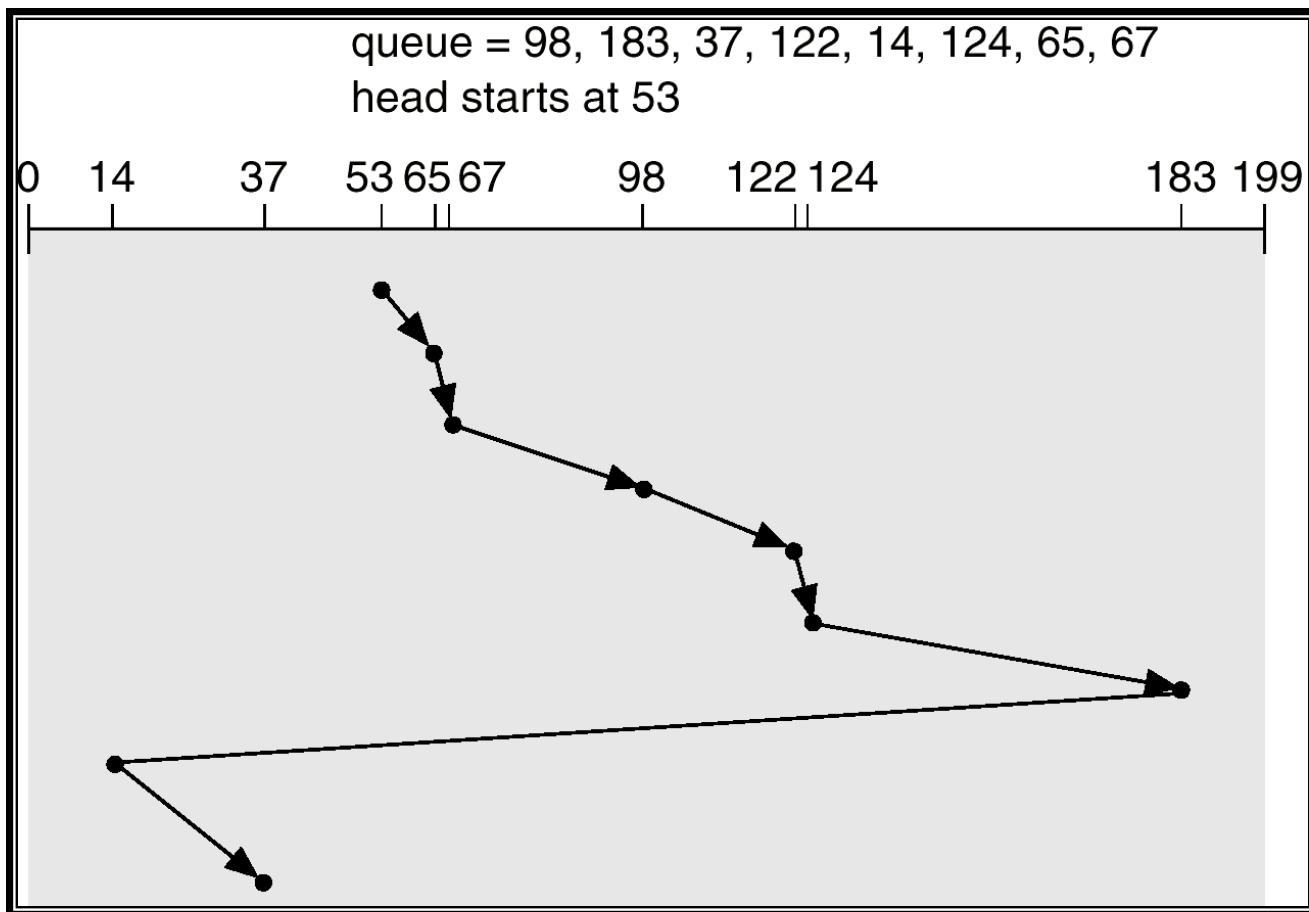


C-LOOK

- C-SCAN算法的优化
- 磁头并不总是移动到最后一个柱面，而是移动到最后一个请求的位置就立即调转方向。

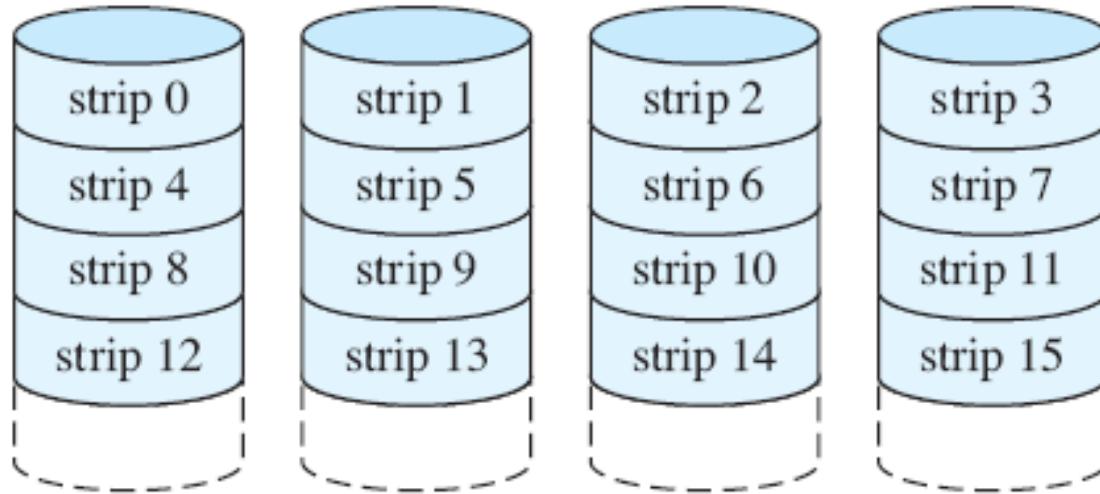
回到第一个请求的位置

C-LOOK (Cont.)



独立磁盘冗余阵列 (RAID)

- RAID的目的
 - 增加容错能力
 - 通过在不同的磁盘上维护冗余数据增加容错能力
 - 获得高性能
 - 通过把数据分布到多个磁盘，采用并行存取以加快数据传输速率来获得高性能
- RAID包含7种，代表了不同的设计架构
- 不同的RAID共有的三个特征
 - 操作系统将RAID的一组物理磁盘看成是一个逻辑磁盘
 - RAID通过数据分段将数据分布在一组物理磁盘上(不同磁盘上)
 - 冗余磁盘容量用于存储校验信息，用于在磁盘失效时恢复数据

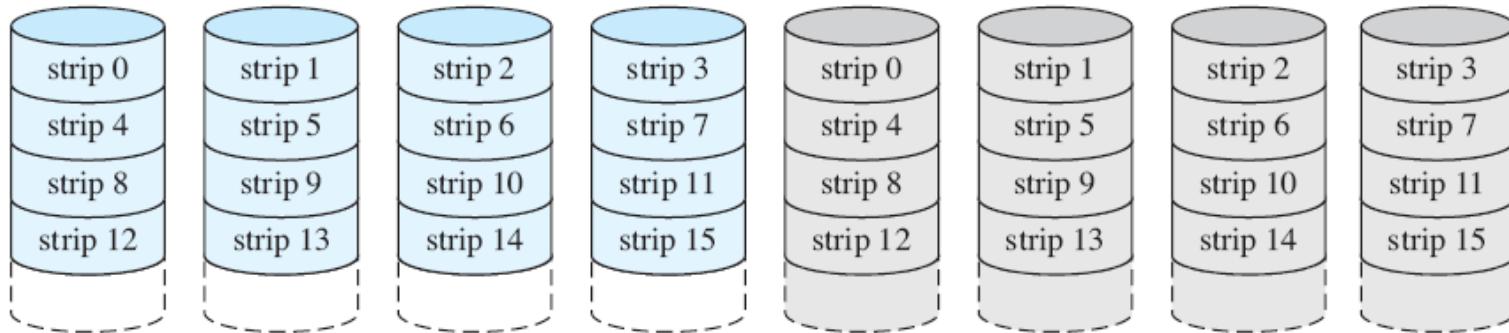


(a) RAID 0 (nonredundant)

(1) **strip**: 数据按固定长度分块分布在不同的物理磁盘上. Strip的粒度可以是block, sector, byte, word

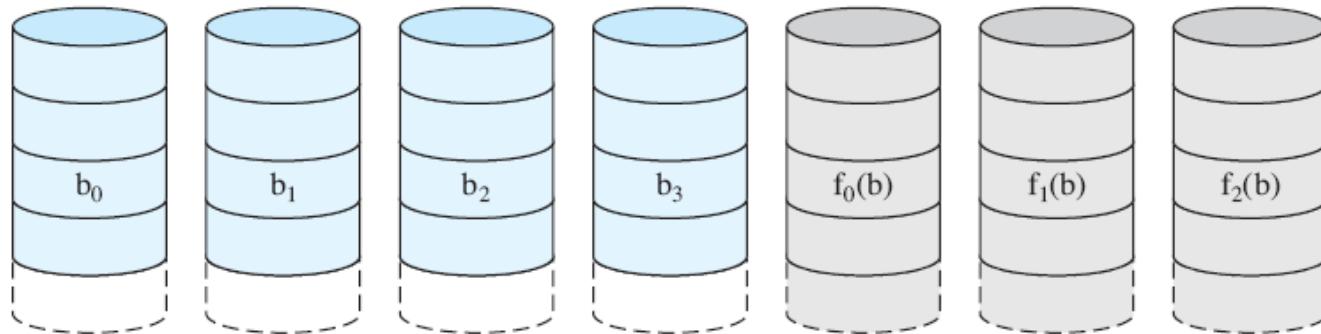
(2) 没有冗余存储

不是真正意义上的RAID



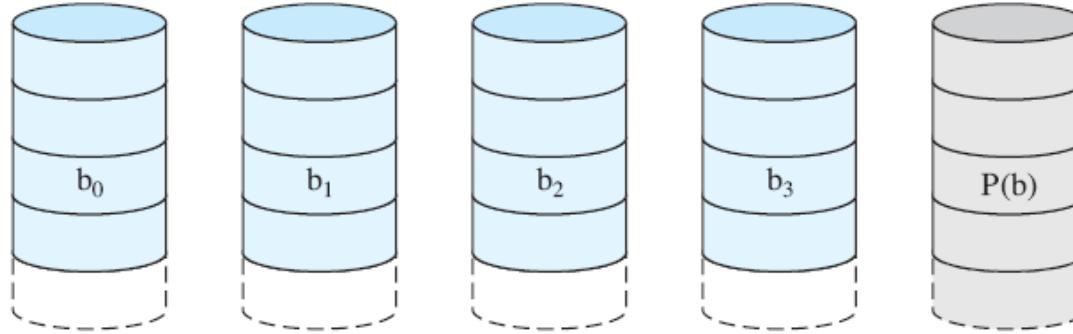
(b) RAID 1 (mirrored)

1. 数据分块分布在不同的物理磁盘上
2. 采用镜像技术实现冗余，容错能力得到大幅提高
 - 假设单个磁盘的平均无故障时间为100000小时，单个盘的平均修复时间为10个小时，则
 - 100个磁盘的平均无故障时间为1000小时，
 - 采用镜像技术后，丢失数据的平均时间间隔变为57000年 两个磁盘同时发生故障
3. 读操作可以由包含数据的两个磁盘中的任一个服务；写操作需要同时更新两个磁盘；从错误中恢复简单
4. 缺点是代价大，需要双倍的磁盘数量



(c) RAID 2 (redundancy through Hamming code)

- (1) 数据按字节或字大小分块分布在不同的磁盘上
- (2) 对应于每个位的纠错码（通常是海明码）存储在多个校验磁盘的相应位置上。
Hamming code
- (3) 所需的冗余磁盘个数是数据盘个数的对数大小
- (5) 采用并行访问技术(不同磁盘的转轴需要同步，保证磁头位于相同的位置)，读写速度高
- (4) 读操作需要访问所有磁盘；写操作需要写所有的磁盘。
- (5) RAID 2适用于磁盘错误频发的场景



(d) RAID 3 (bit-interleaved parity)

(1) 与RAID 2类似，唯一的不同在于使用奇偶校验码而不是纠错码，因此只需要一个冗余磁盘即可

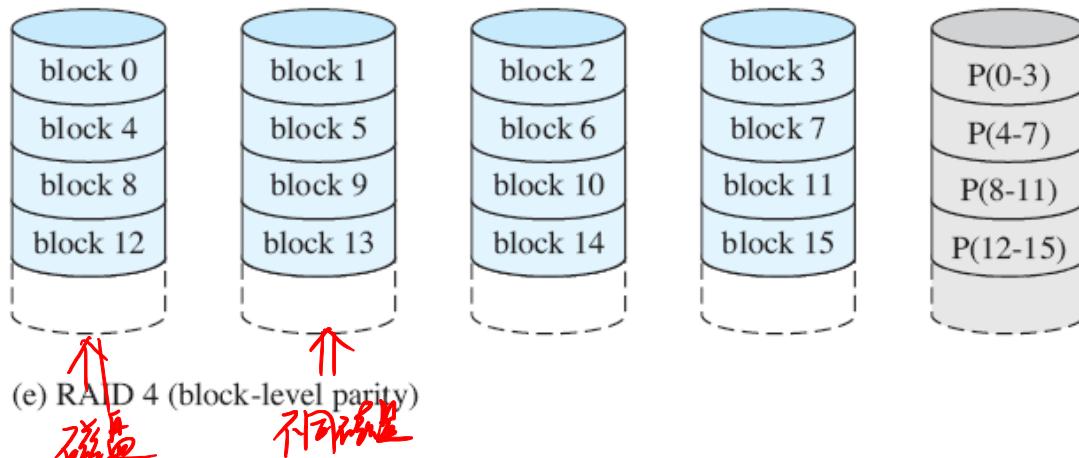
(2) 传统上，奇偶校验码只能用于检测错误，不能纠错，但是磁盘控制器通过扇区的ECC可以判断出哪个磁盘块出错，因此一个奇偶校验盘+ECC可以用于纠错

(2) 能容忍一个磁盘失效

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

若X1失效，则可以通过下面的方式恢复

$$X1(i) = X4(i) \oplus X3(i) \oplus X2(i) \oplus X0(i)$$

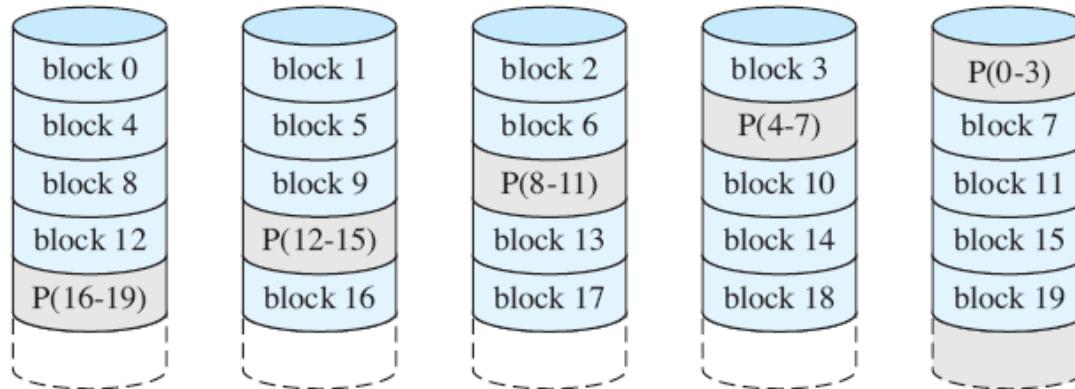


- (1) Strip的粒度较大，以块为单位
- (2) 与RAID 2,3 实行并行访问不同， RAID 4的每个磁盘独立访问
- (3) 适合于大访问量的场景，即多个访问可以同时进行，整体吞吐率高，但单个请求的访问速率没有提高
- (4) 读操作只需要访问数据所在的磁盘；写操作需要更新冗余磁盘的数据，需要两次读，两次写。

 $0 \sim 4$ blocks
data.

$$X4(i) = X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)$$

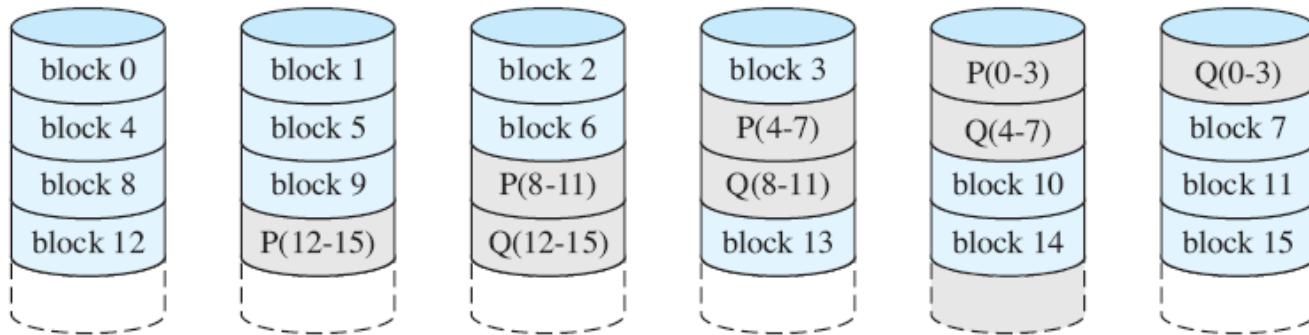
$$\begin{aligned}
 X4'(i) &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \\
 &= X3(i) \oplus X2(i) \oplus X1'(i) \oplus X0(i) \boxed{\oplus X1(i) \oplus X1(i)} \quad \text{不变} \\
 &= \underline{X3(i) \oplus X2(i) \oplus X1(i) \oplus X0(i)} \oplus \underline{X1(i) \oplus X1'(i)} \quad \text{调整顺序} \\
 &= X4(i) \oplus X1(i) \oplus \boxed{X1'(i)} \quad X4(i) \\
 &\quad \text{新旧校验值} \quad \text{原始值} \quad \text{要更新}
 \end{aligned}$$



(f) RAID 5 (block-level distributed parity)

(1) RAID 4中，奇偶校验盘可能成为瓶颈

(2) RAID5将奇偶校验信息分布在不同的磁盘上，有效地避免了该瓶颈。
从而



(g) RAID 6 (dual redundancy)

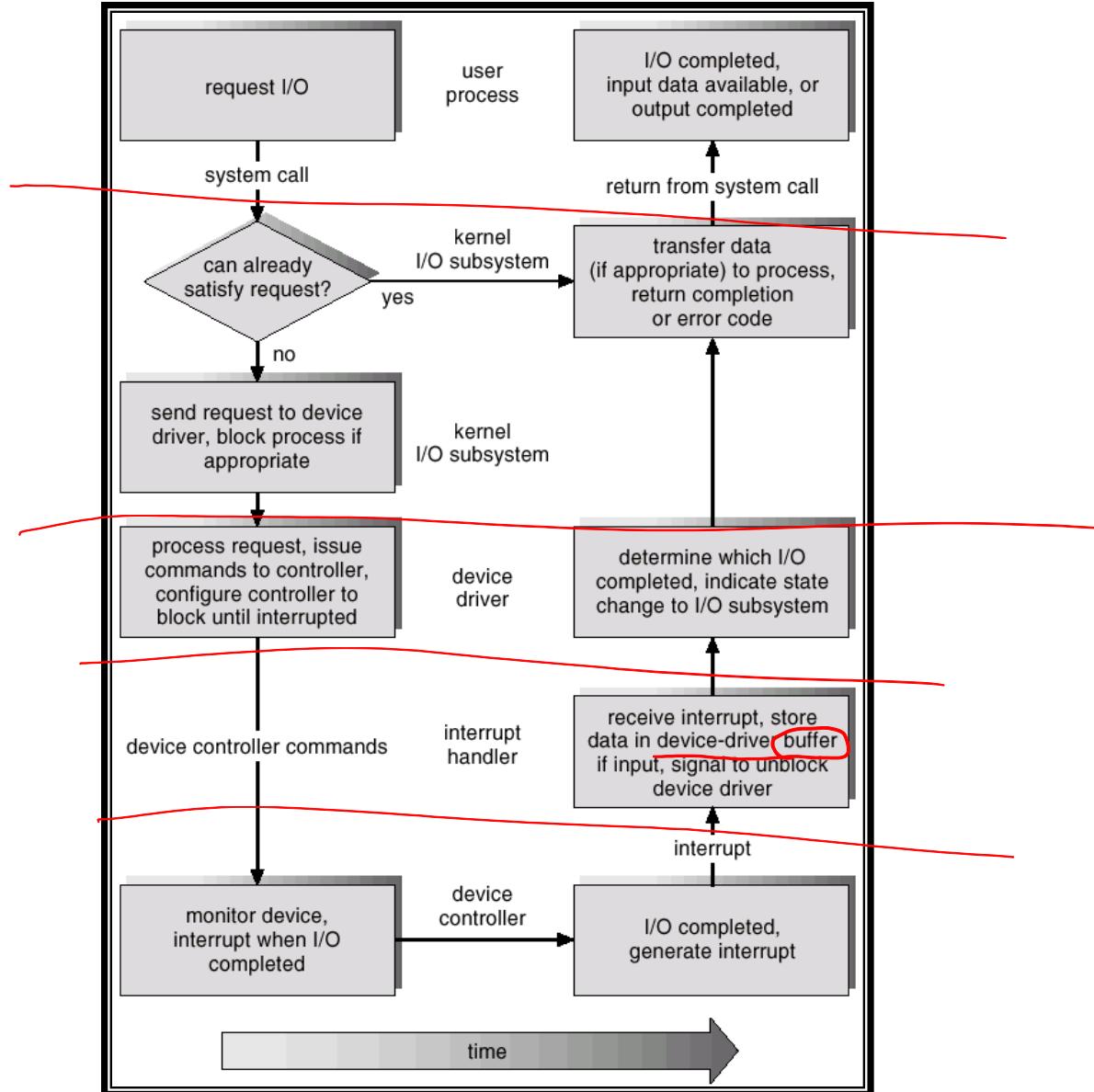
- (1) 与 RAID 5 类似，但采用了 两种不同的校验算法
- (2) 对于 N 个数据盘来说，需要 $N+2$ 个磁盘
- (3) 能容忍 两个盘同时失效
- (4) 写操作的代价高，每次写操作需要写三个盘

其它提高磁盘I/O速度的方法

- 提前读 预测性读取
 - 对采用顺序访问的文件数据，读当前块时已知下次要读出的盘块地址，可以将下一盘块的数据也读入缓冲区
 - 未来要读入数据时可以直接使用缓冲区的数据
- 延迟写
 - 执行写操作时，不立即将缓冲区的数据写盘，而是把它挂到空闲缓冲区队列末尾
 - 当有进程申请缓冲区且分配到上述缓冲区时，才把其中的数据写到磁盘上
 - 当数据未被写回磁盘时，所有对该数据的磁盘访问可以直接从缓冲区取出，而不用访问磁盘

例：读文件I/O请求的过程

- 确定拥有文件的设备
- 将文件名转化到磁盘设备的表示
- 将数据从磁盘读入到内核缓冲区
- 通知请求进程数据可用
- 将控制交还给进程



读文件I/O请求的完整过程

设备分配

- 什么是设备分配?
 - 计算机系统中可能有多种类型的外设，每种外设可能有多台
 - 作业执行前，应对需要的设备提出申请要求，
如何实现不同作业的设备申请？

设备分配

- 设备独立性
 - 用户不指定物理设备，而是指定逻辑设备，使得用户作业和物理设备分离开来，再通过其他途径建立逻辑设备和物理设备之间的映射
- 系统需要提供逻辑设备名到物理设备名的映射表，将逻辑设备名转换成物理设备名
- 设备独立性的优点
 - 应用程序与具体的物理设备无关，系统增减或变更设备时无需修改源程序
 - 易于应对I/O设备故障 换驱动不需要改源程序
 - 能更有效地利用设备资源，提高多道程序设计的能力

设备分配方式

- 静态分配
 - 适合于独占性设备，如卡片机、打印机、磁带机等；
 - 在作业执行前，将所要使用的设备全部分配给它
 - 能够防止死锁，但会降低设备利用率
- 动态分配
 - 在作业执行过程中才将设备分配给作业 无须。
- 虚拟分配
 - 一个物理设备为多个应用分配

虚拟设备

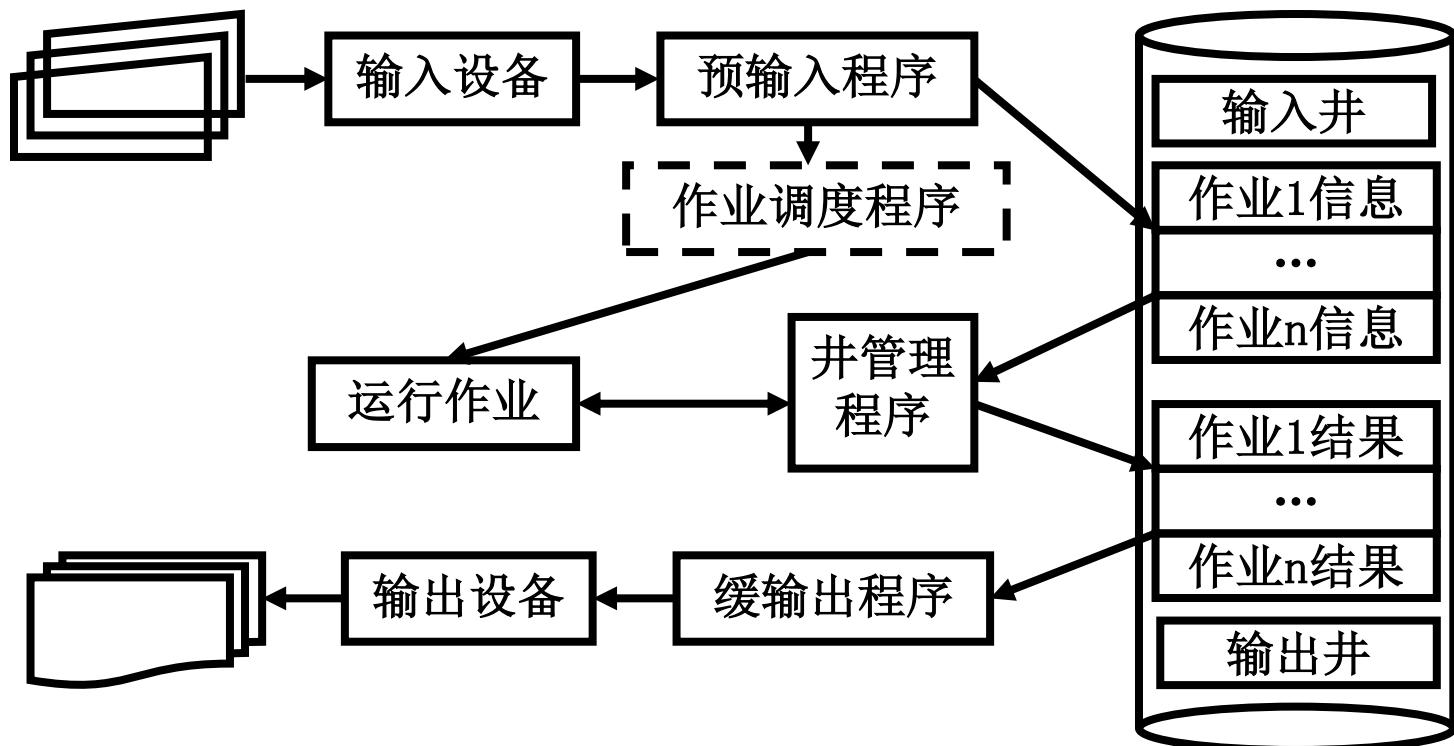
- 为何提出虚拟设备的概念?
 - 对卡片机和打印机等设备，采用静态分配不利于提高系统效率
 - 在作业执行期间，这些设备真正工作的时间很少，大部分时间处于空闲状态
 - 由于设备被占用，则其它申请同样设备的作业将被迫等待

设备虚拟化

- Spooling技术
 - 用一类物理设备模拟另一类物理设备的技术，是使独占型设备变成共享设备的一种技术
 - 通过高速磁盘实现预输入和缓输出，使得作业执行I/O操作时不再需要和低速设备联系，而是和高速磁盘交互
 - 系统构成
 - 输入/输出井
 - 预输入程序
 - 缓输出程序
 - 井管理程序
 - 当执行过程中需要启动某台设备要求进行I/O操作时，操作系统截获该请求并调用井管理程序从相应的输入井读取信息，或将信息送至输出井

虚拟设备

- 联机同时外围设备操作（斯普林系统）



Spooling系统应用

--共享打印机

- 打印机属于独占设备，利用**SPOOLing**技术，可将之改造为一台可供多个用户共享的设备
- 实现机制
 - 系统存在一个特殊的打印机守护进程，是唯一有特权使用打印机设备的进程
 - 在打印一个文件之前，应用程序产生完整的待打印文件，打印机守护进程同意打印输出，但并不真正将打印机分配给该用户进程，而是将待打印文件存放在打印机的spool目录下。
queue
 - 当打印机空闲时，便启动打印机守护进程，打印待输出的文件

本章小结

I/O子系统
特性

通用性和高效性

总

功能性

- I/O子系统的设计目标是提高其**通用性和高效性**
- I/O子系统由硬件和软件组成
 - CPU通过**设备控制器**实现与I/O设备的交互
 - I/O软件包括**中断处理程序**、**设备驱动程序**和**独立于I/O设备的软件**
- 设备驱动程序向上提供**标准的接口**，屏蔽实现细节；向下通过与设备相关的代码控制**设备控制器**；
- 磁盘的基本结构和磁盘驱动调度算法
 - 电梯调度、先来先服务调度、最短查找时间优先调度等；
- 冗余磁盘阵列用于提高系统的**容错能力和性能**；
- **设备独立性**指用户指定逻辑设备而非物理设备，操作系统完成逻辑设备到物理设备的映射
- **Spooling技术**是用一类物理设备模拟另一类物理设备的技术，是使得独占型设备变成共享型设备的一种技术

非易失性内存