操作系统

L21 大容量存储

胡燕
大连理工大学 软件学院

简介

Introduction to Disk Scheduling

01

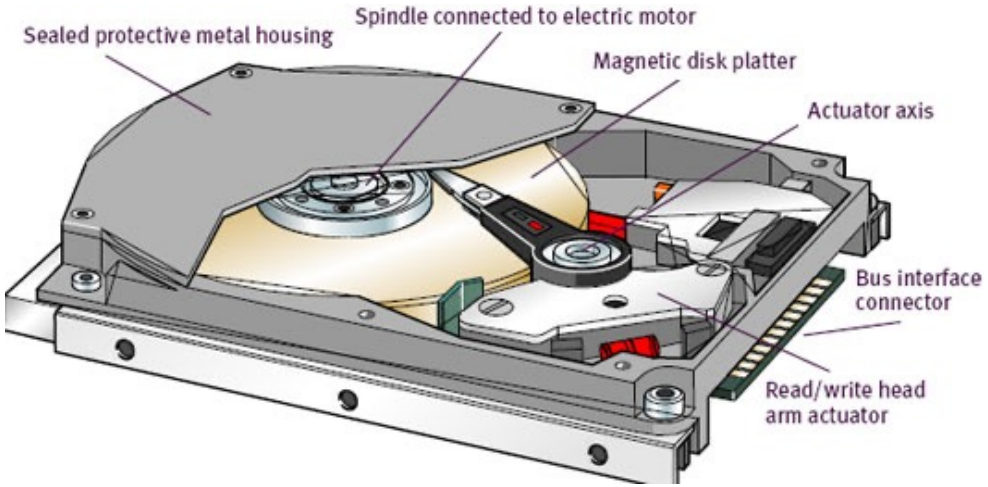
大容量存储

Q: 计算机系统为何需要大容量存储？

大数据/AI时代，数据为王，计算机系统处理的信息的载量越来越大
大容量存储的地位不可撼动。

磁带、机械磁盘是大容量存储的典型代表

磁带

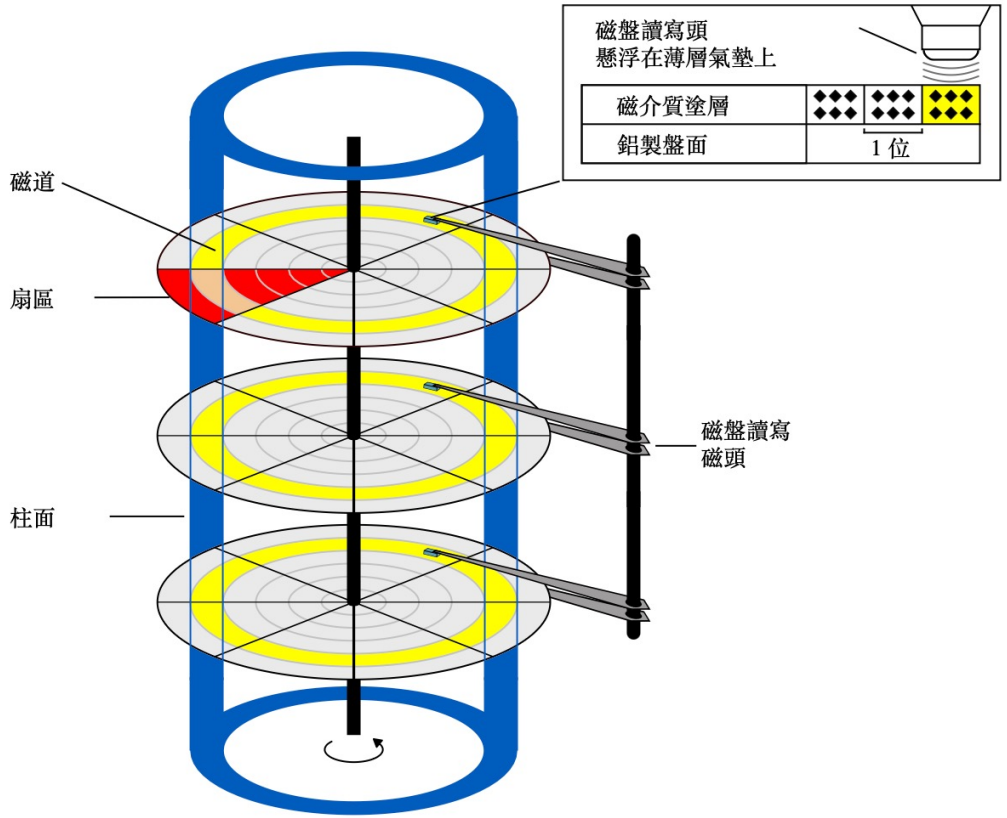


机械硬盘

https://www.ibm.com/cn-zh/it-infrastructure/storage/tape/drives

磁盘物理结构

传统机械磁盘结构：
由柱面、磁道、扇区组成



现代机械磁盘的典型参数值

理论传输速率：6Gb/s
实际（有效）传输速率：1Gb/s

寻道时间：3ms~12ms
旋转延迟：（1/RPM)*60
平均旋转延迟 = 0.5*旋转延迟

Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

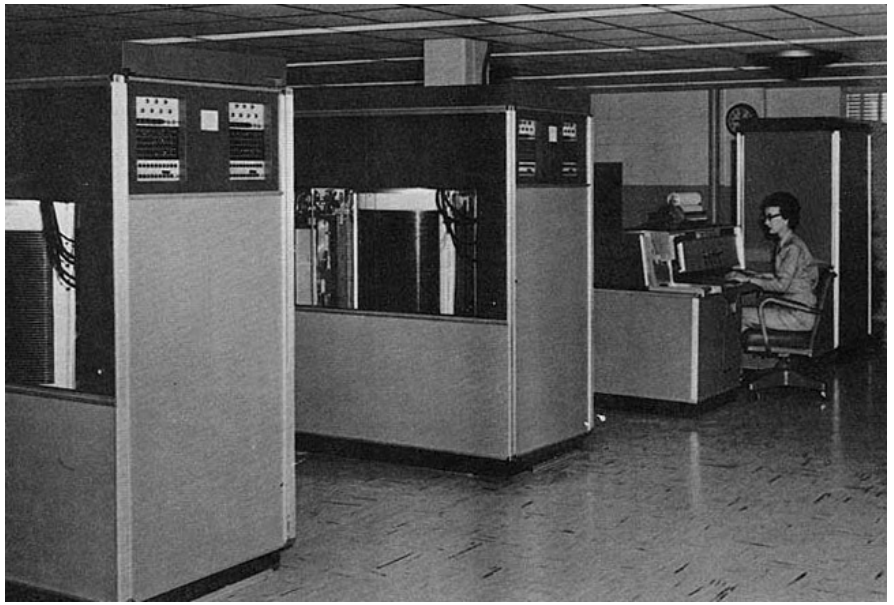
(From Wikipedia)

机械磁盘历史

- **史上最早的商用磁盘**
  - the IBM Model 350 disk storage system， 1956
  - 容量： 5M (7 bit)
  - 50个24英寸磁碟（Platters）
  - 访问时间 < 1 second

机械磁盘历史

- **IBM推动早期硬盘技术的创新**
  - RAMAC：最早使用硬盘存储的商用计算机，上面装载了Model 350 disk storage system
  - 需要整个房间放置该计算机，其硬盘系统有两个冰箱那么大



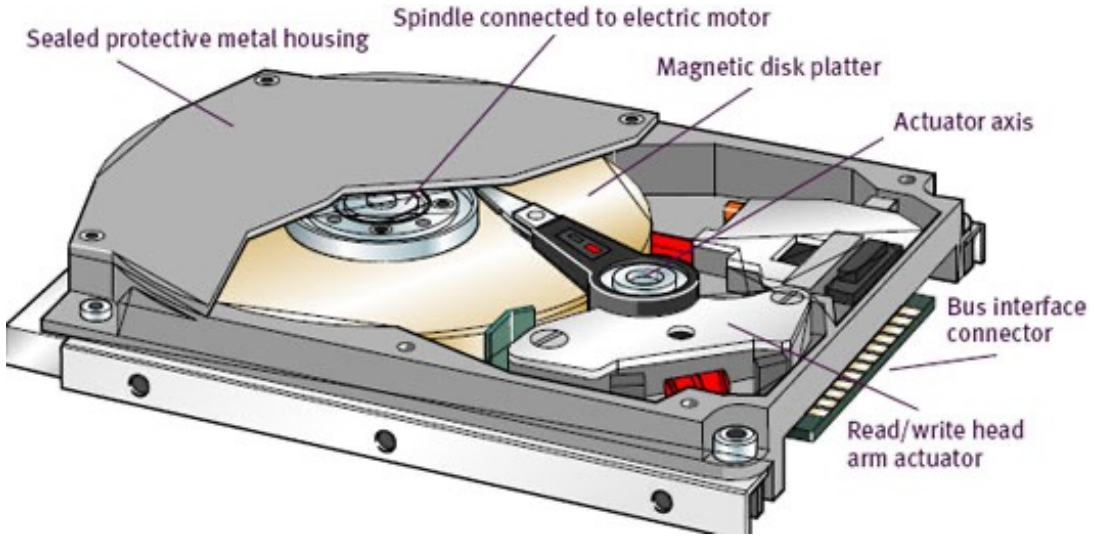
RAMAC：Random Access Method for Accounting and Control

机械磁盘历史

- **IBM早期推出的可移动存储：IBM 1311磁盘**
  - Removable storage

机械磁盘历史

- **现代磁盘样式结构**



机械磁盘历史

- **Advance Format Disk**

扇区大小 = 4Ks

磁盘firmware里不会把4K的物理扇区模拟成512B的逻辑扇区

512e

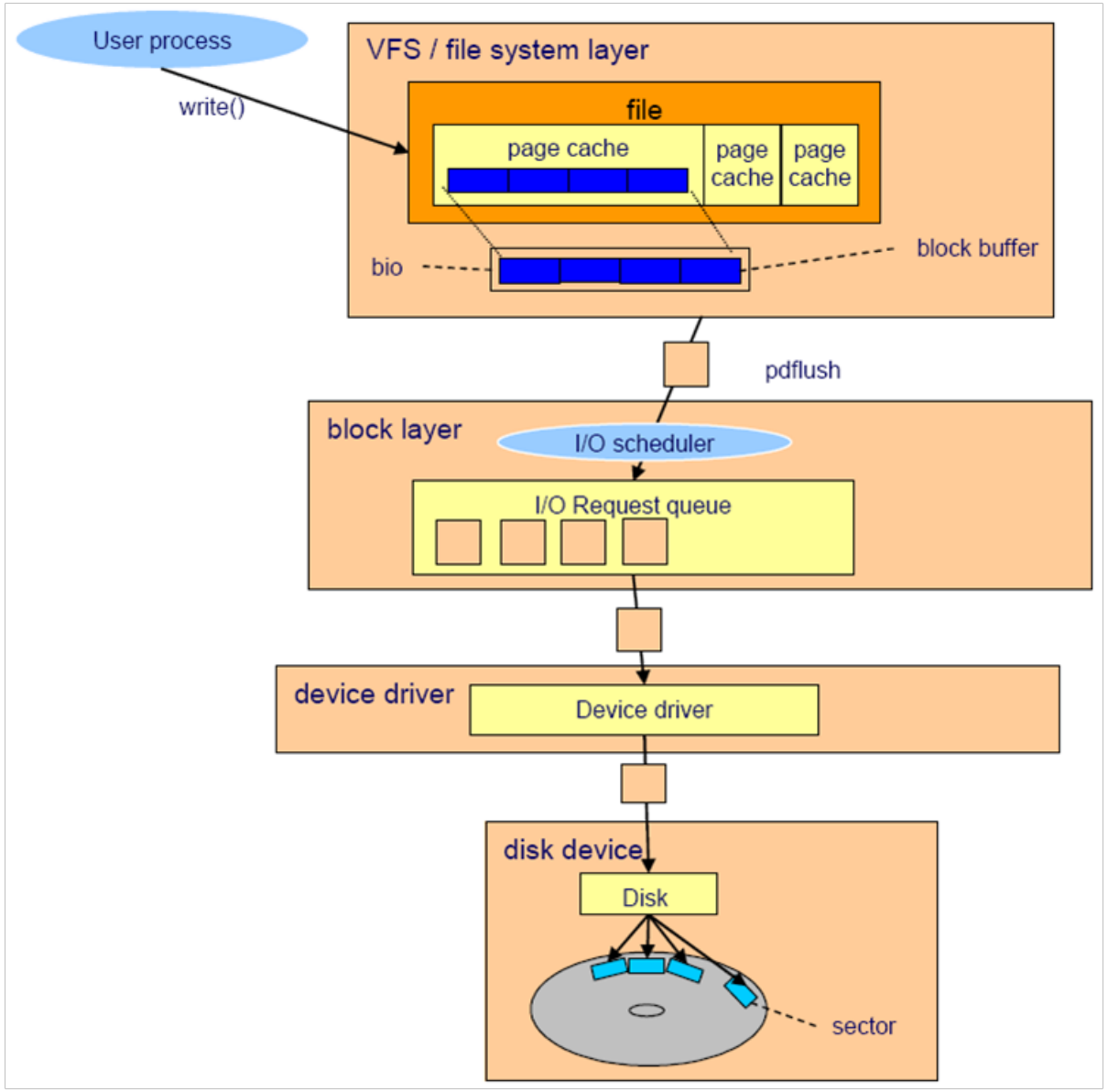
为了向下兼容，磁盘提供商在firmware里模拟512B扇区

磁盘调度算法

Disk Scheduling Algorithms

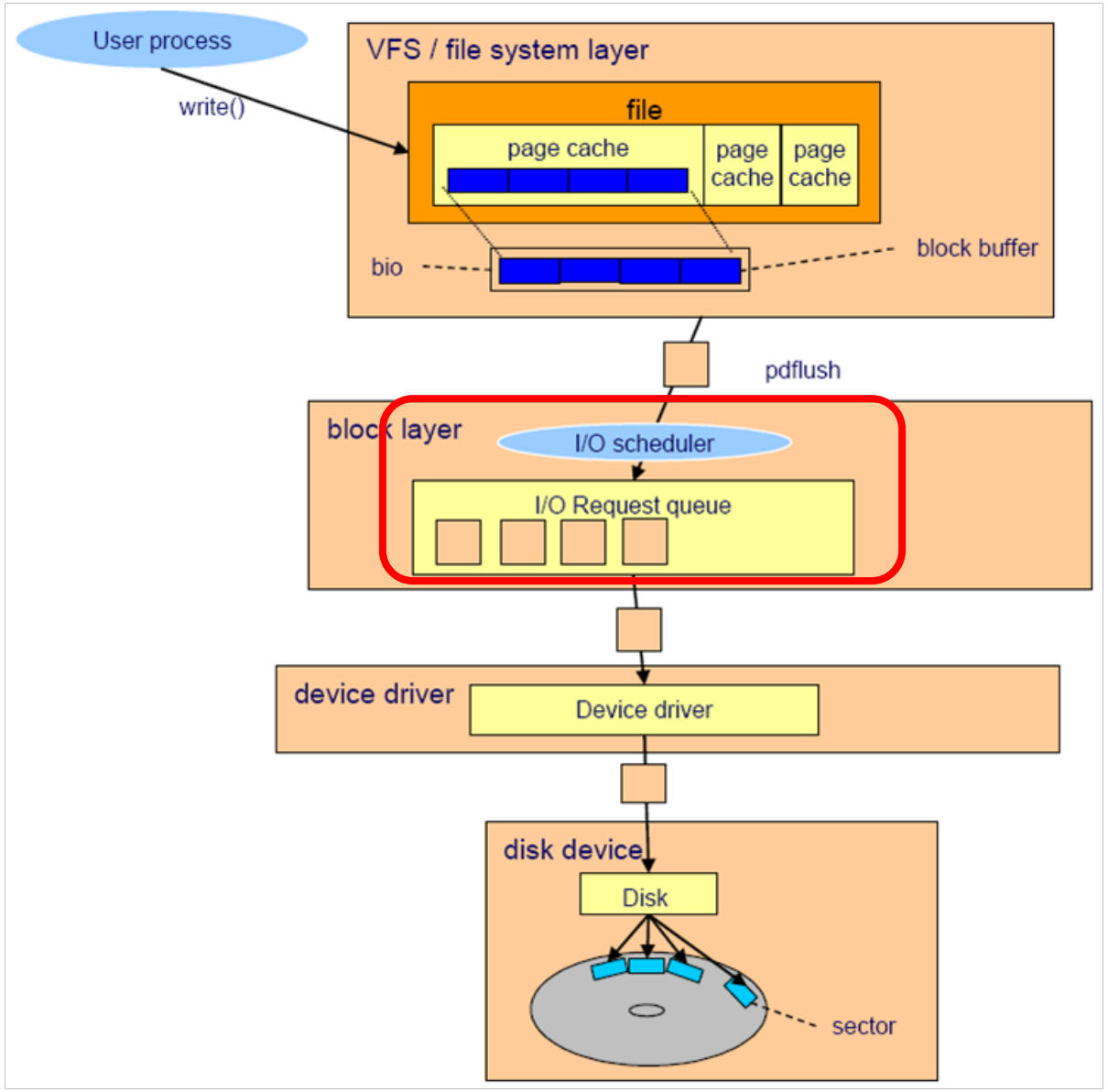
02

磁盘调度背景



实例: 文件写操作流程示意图（ 左图）

磁盘调度背景

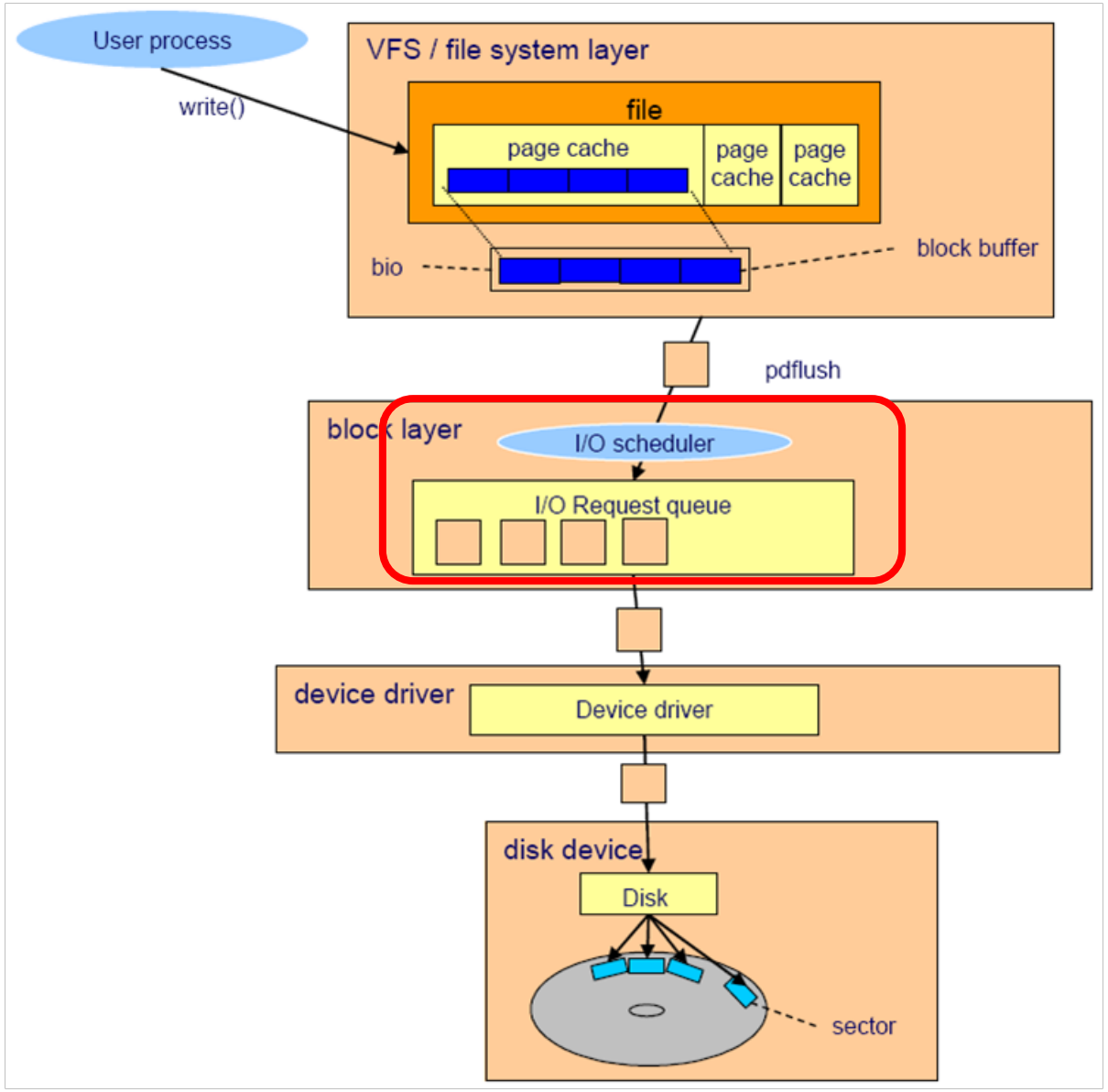


实例: 文件写操作流程示意图（左图）

问题分析

- 应用访问磁盘，生成I/O请求
- I/O请求进入I/O请求队列
- 操作系统在服务I/O请求时，充分考虑机械磁盘的特性，对I/O请求进行排序

磁盘调度背景

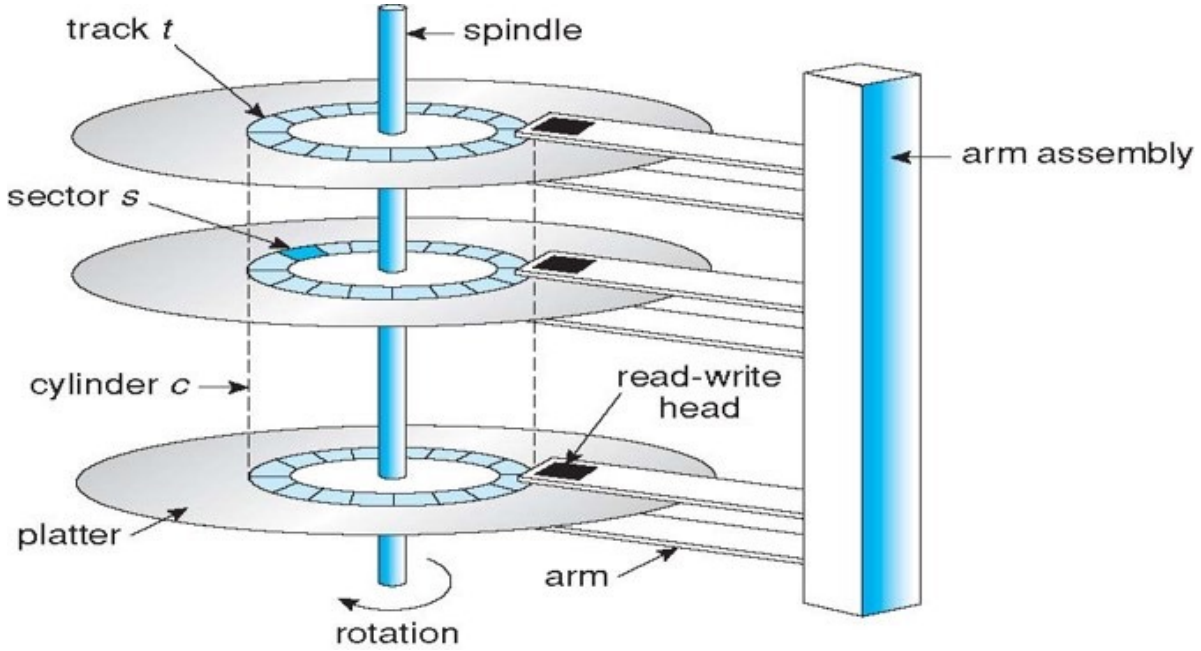


实例: 文件写操作流程示意图（左图）

需要对I/O请求进行排队调度，
充分提升I/O性能

磁盘调度背景

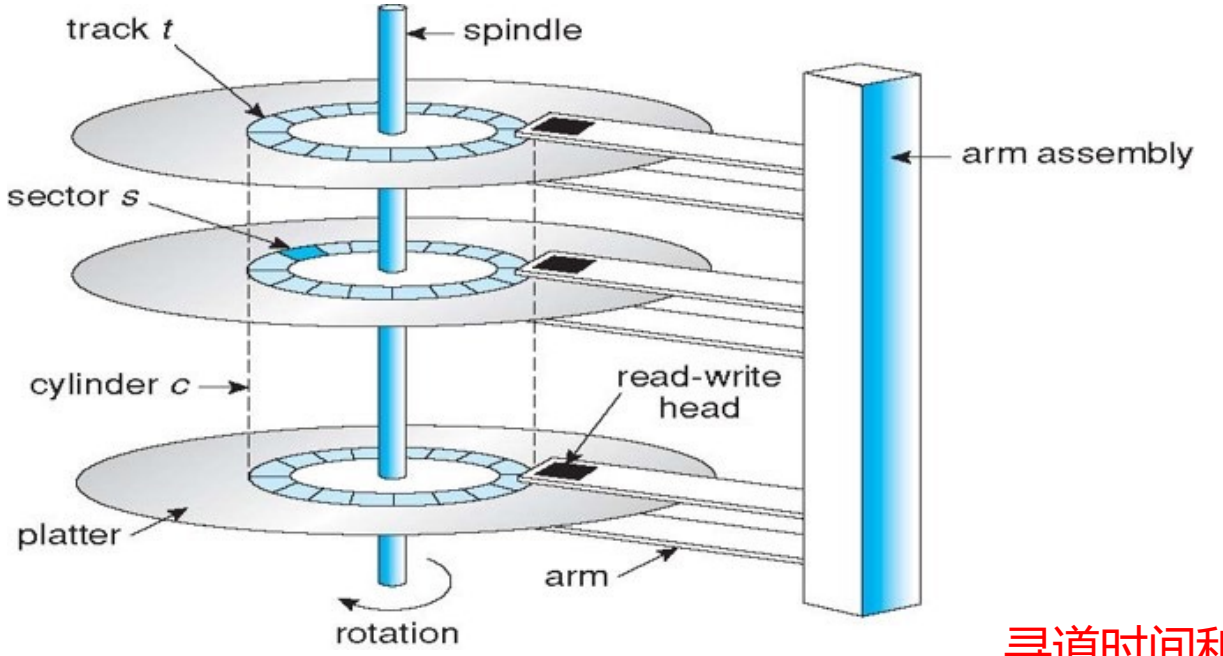
问题：要如何调度，才能提升硬盘I/O效率？



寻道时间：3ms~12ms
旋转延迟：1/(RPM*60)
平均旋转延迟 = 0.5*旋转延迟

磁盘调度背景

问题：要如何调度，才能提升硬盘I/O效率？

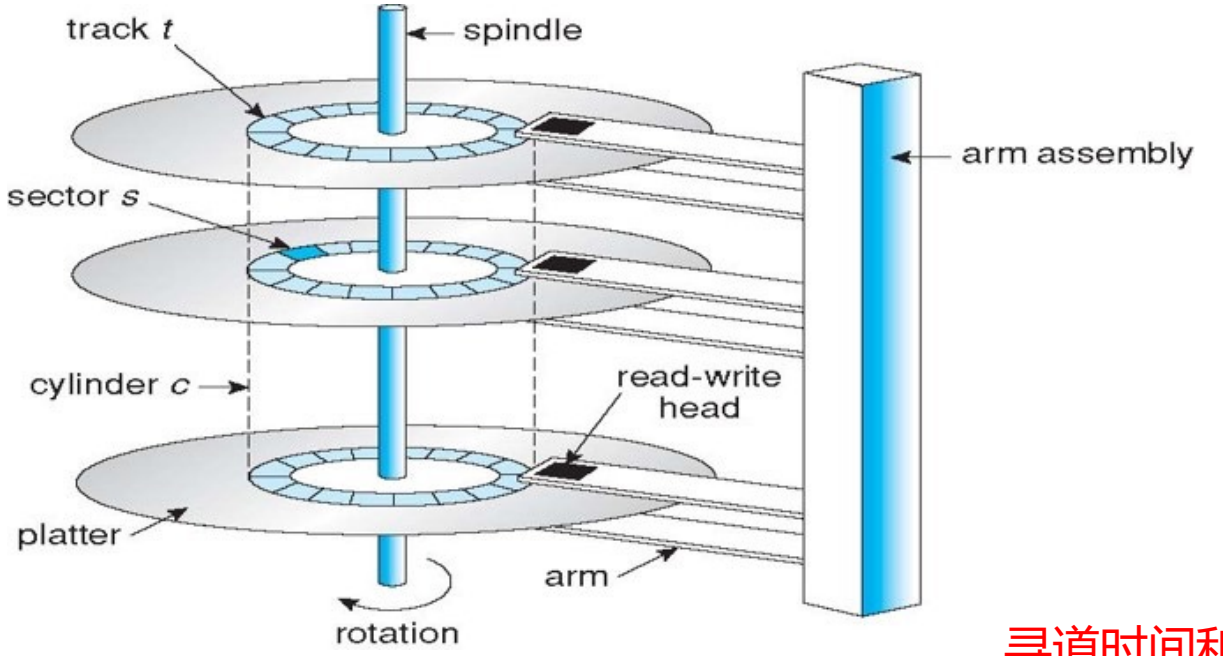


寻道时间：3ms~12ms
旋转延迟：1/(RPM*60)
平均旋转延迟 = 0.5*旋转延迟

寻道时间和旋转延迟，哪个便于通过I/O调度加以控制？

磁盘调度背景

问题：要如何调度，才能提升硬盘I/O效率？



寻道时间：3ms~12ms
旋转延迟：1/(RPM*60)
平均旋转延迟 = 0.5*旋转延迟

寻道时间和旋转延迟，哪个便于通过I/O调度加以控制？

通过对I/O请求的调度，最小化磁头寻道距离
=> 减少寻道距离，从而缩短总体寻道时间

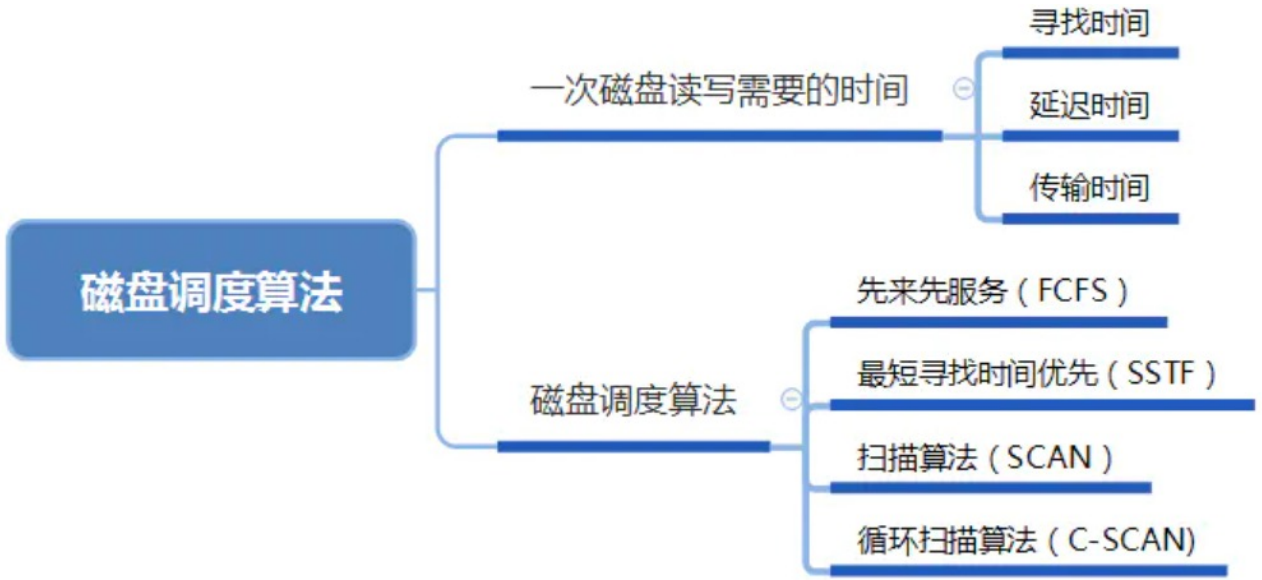
磁盘调度问题描述

磁盘调度的优化指标：寻道时间

- 可由操作系统控制
- 调度算法目标：最小化磁头寻道距离

磁盘调度算法输入：IO请求队列

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

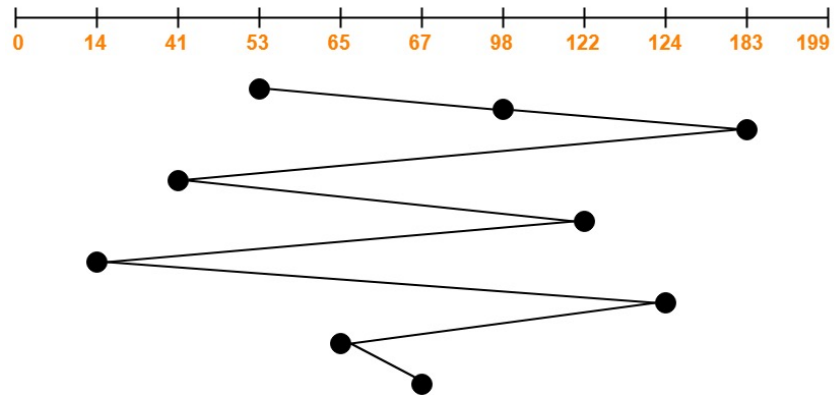


每个数字表示一个具体I/O请求所访问磁盘块所对应的磁道号
不同的磁盘调度算法，服务IO请求的顺序不同，算法效果也因此不同

FCFS算法

FCFS算法

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

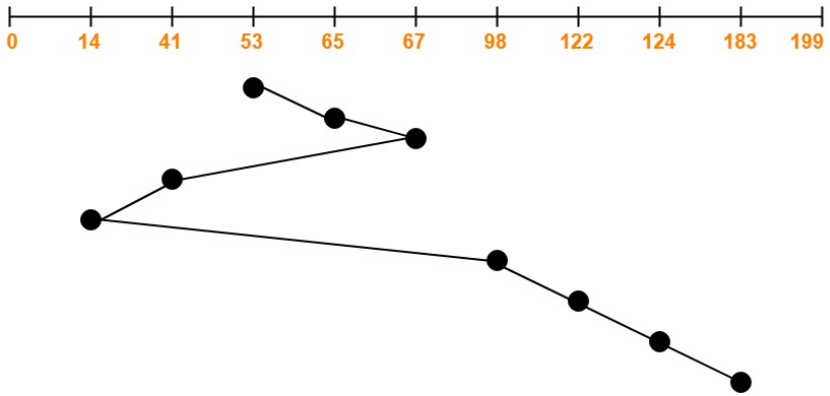


Total head movements incurred while servicing these requests
= (98 – 53) + (183 – 98) + (183 – 41) + (122 – 41) + (122 – 14) + (124 – 14) + (124 – 65) + (67 – 65)
= 45 + 85 + 142 + 81 + 108 + 110 + 59 + 2
= 632

SSTF算法

SSTF算法

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

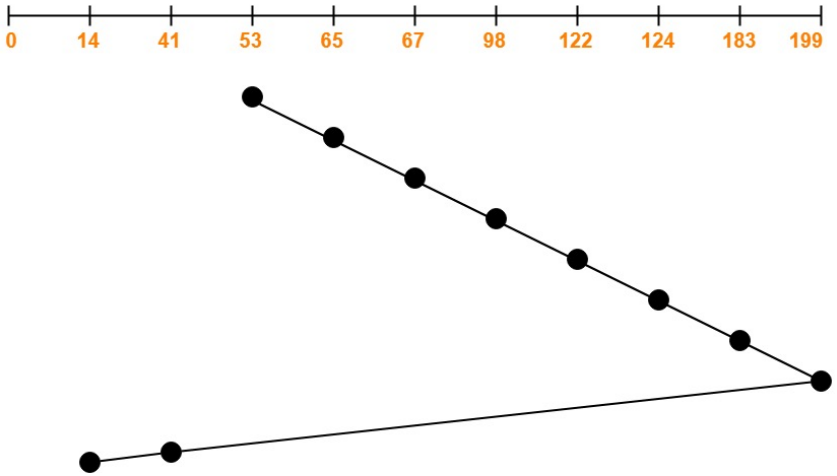


Total head movements incurred while servicing these requests
= (65 – 53) + (67 – 65) + (67 – 41) + (41 – 14) + (98 – 14) + (122 – 98) + (124 – 122) + (183 – 124)
= 12 + 2 + 26 + 27 + 84 + 24 + 2 + 59
= 236

SCAN算法

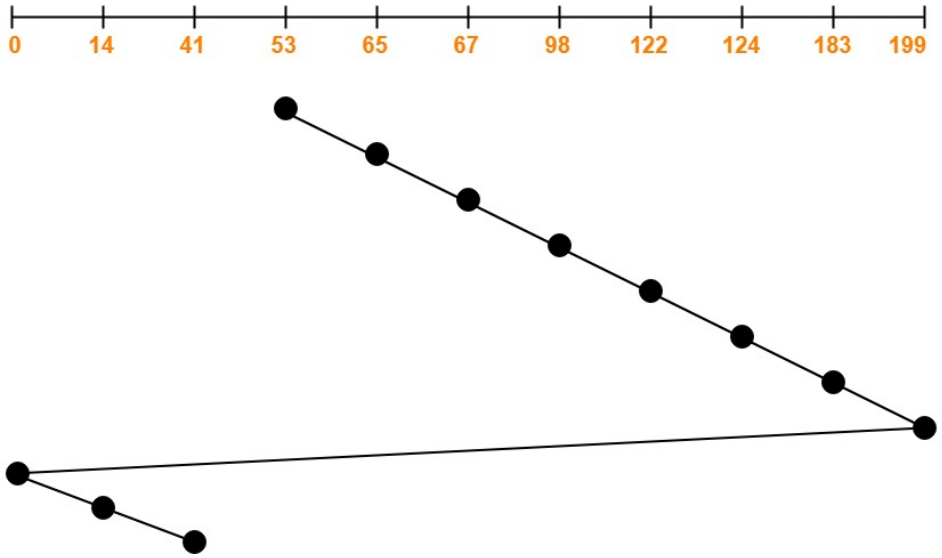
SCAN算法

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



Total head movements
= (199 – 53) + (199 – 14)
= 331

C-SCAN算法

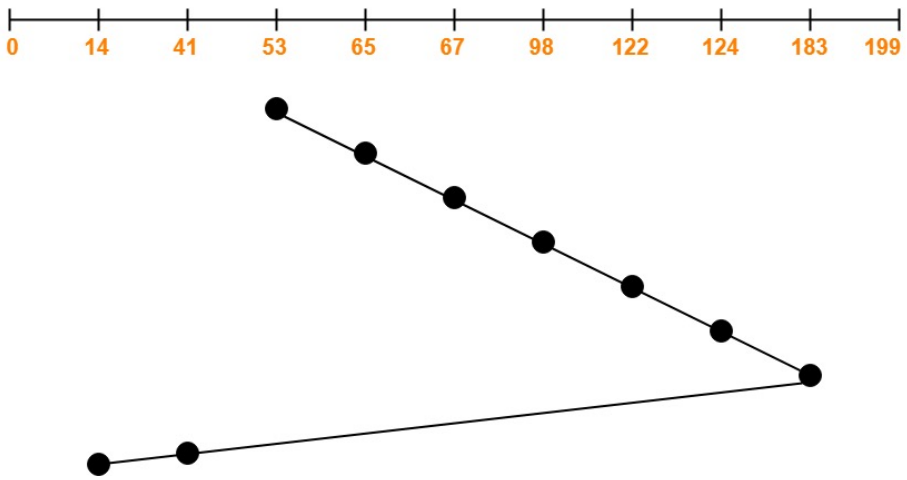


Total head movements
= (199 – 53) + (199 – 0) + (41-0)
= 386

LOOK算法

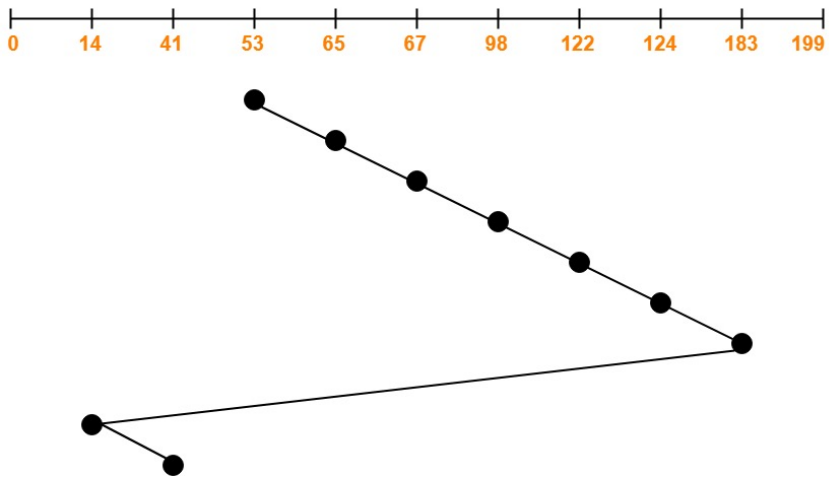
请求队列=98, 183, 37, 122, 14, 124, 65, 67

LOOK算法



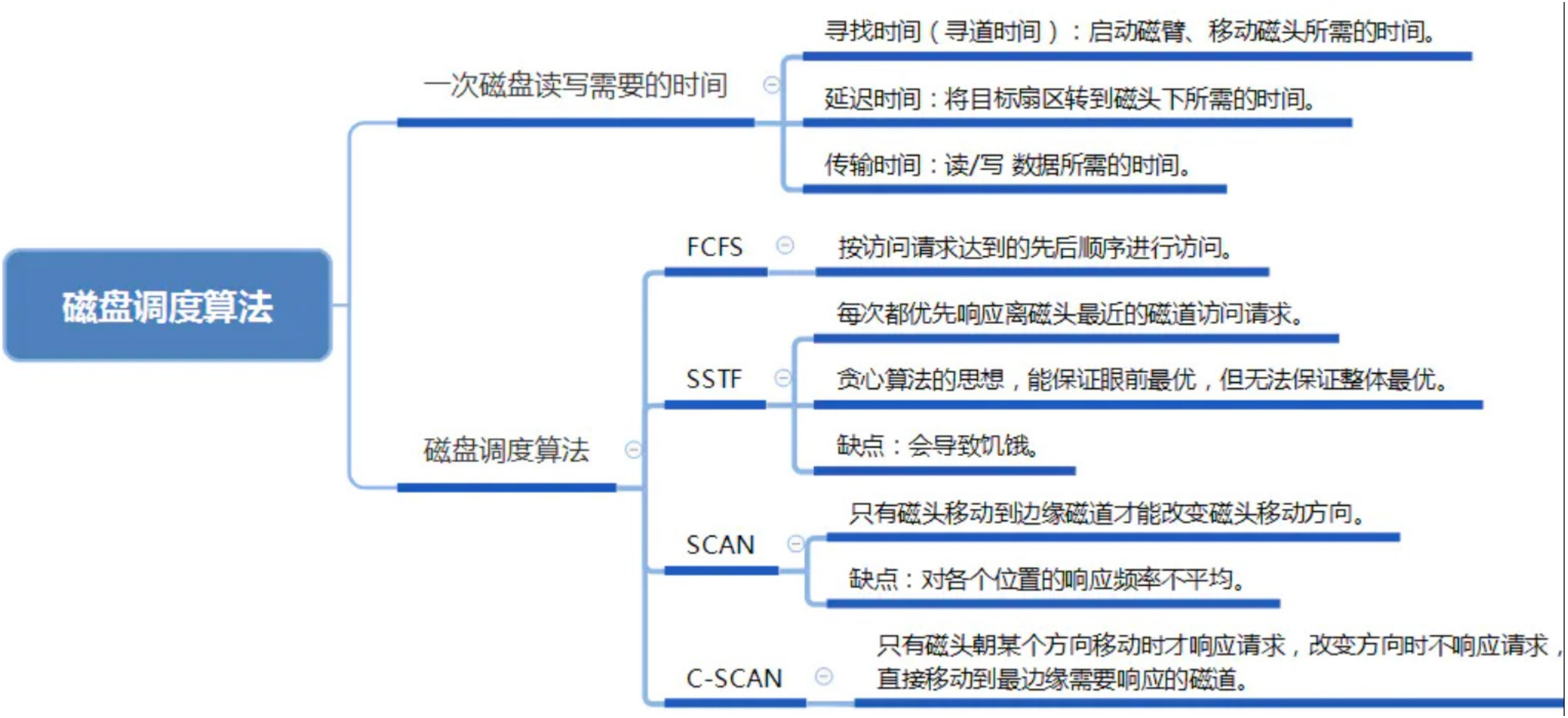
Total head movements
= (183 – 53) + (183 – 14)
= 299

C-LOOK算法



Total head movements
= (183 – 53) + (183 – 14) + (41-14)
= 326

磁盘调度算法小结


磁盘调度算法
一次磁盘读写需要的时间
寻找时间（寻道时间）：启动磁臂、移动磁头所需的时间。
延迟时间：将目标扇区转到磁头下所需的时间。
传输时间：读/写 数据所需的时间。
磁盘调度算法
FCFS
按访问请求达到的先后顺序进行访问。
SSTF
每次都优先响应离磁头最近的磁道访问请求。
贪心算法的思想，能保证眼前最优，但无法保证整体最优。
缺点：会导致饥饿。
SCAN
只有磁头移动到边缘磁道才能改变磁头移动方向。
缺点：对各个位置的响应频率不平均。
C-SCAN
只有磁头朝某个方向移动时才响应请求，改变方向时不响应请求，
直接移动到最边缘需要响应的磁道。

磁盘交换空间管理

Swap Space Management

03

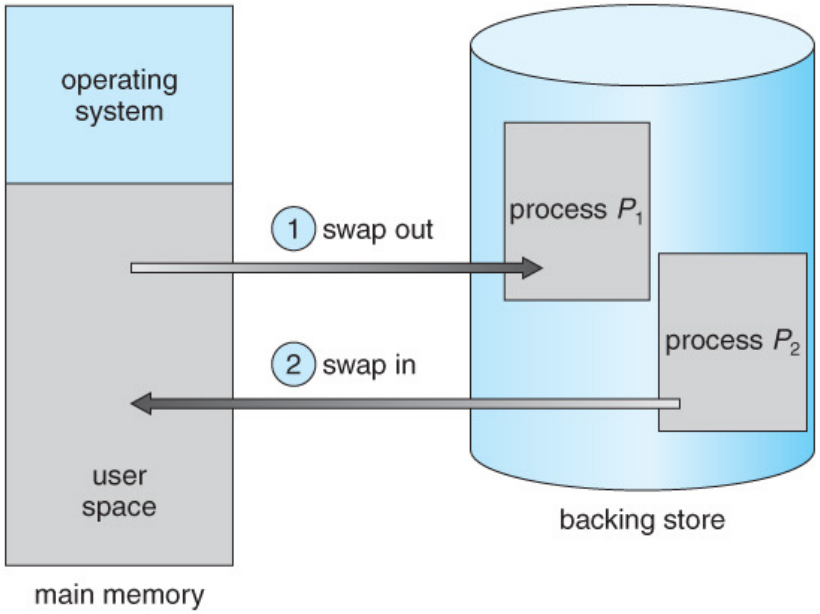
3-磁盘交换空间管理

Why do we need it

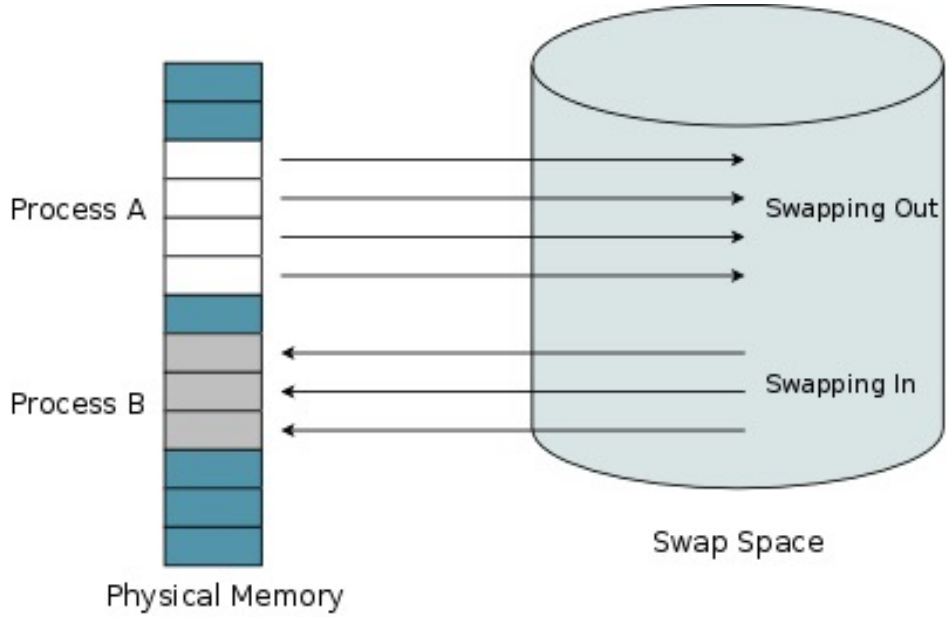
Q: 操作系统中，为何需要磁盘交换空间

内存不管如何扩大， 总是觉得拥挤
（内存大，应用体积也在变大；同时运行的任务也越来越多）

于是，引入磁盘交换空间，借助于磁盘空间，对内存做个扩充



整个进程映像交换



以页为单位进行交换

磁盘交换空间

Q: 如何设计磁盘空间的设计结构？

选择1：交换文件（resides in normal file system）　　代表：Windows交换文件

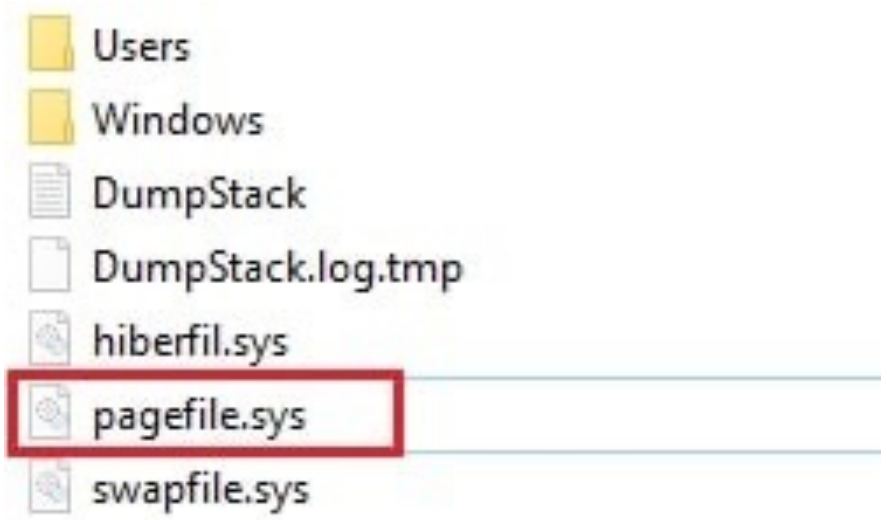
Less efficient

选择2：交换空间（use a raw partition specially for swap management）

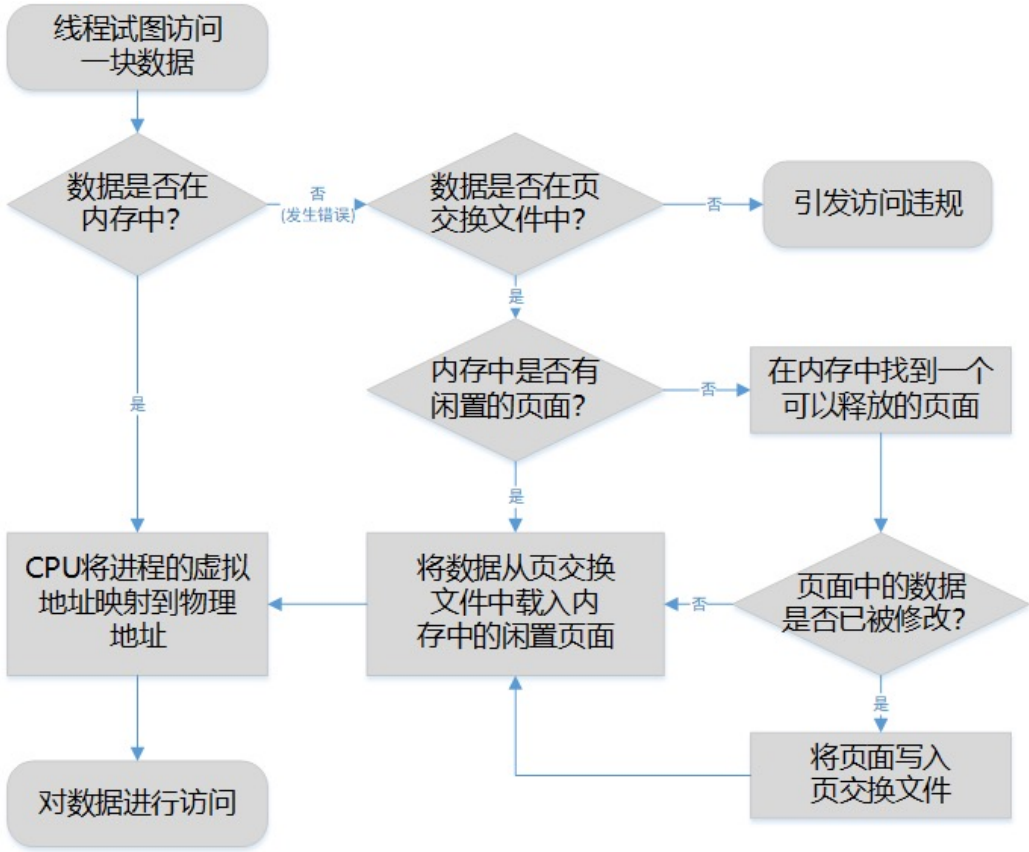
代表：Linux交换分区

示例:Windows交换机制

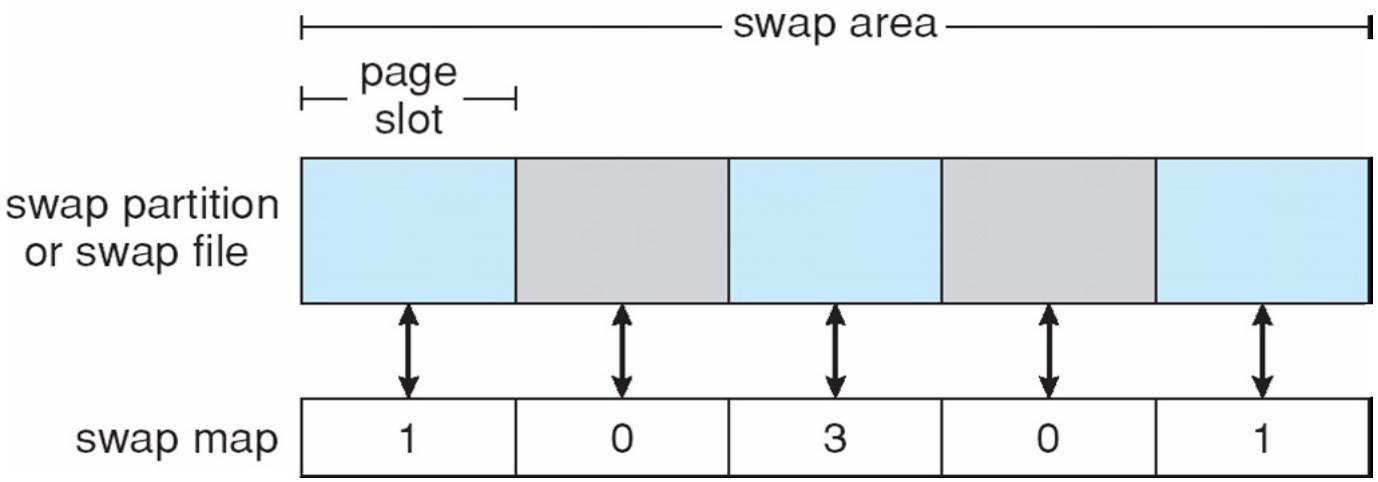
Windows交换空间
-交换空间是一个名为pagefile.sys的文件，交换的内容均放在该页文件中



样例：Win10系统盘



示例：Linux磁盘交换分区



Linux的swap分区结构，基于swap map进行管理

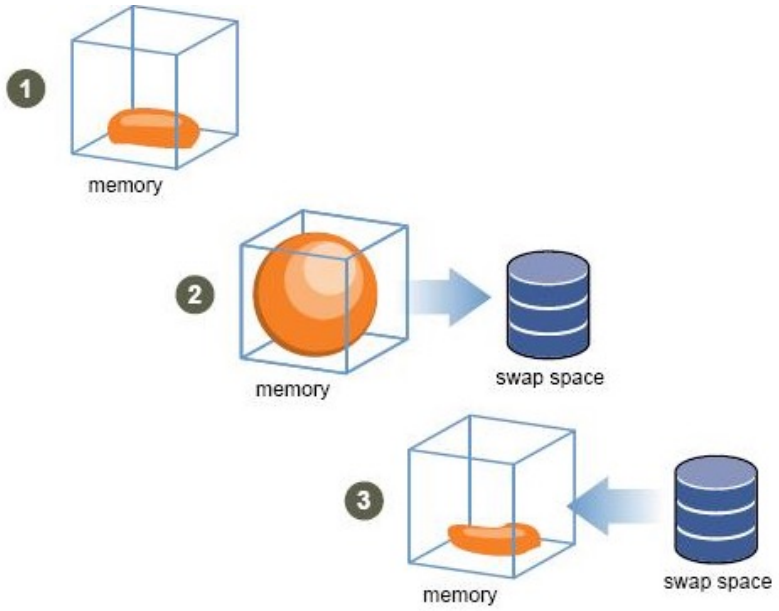
如果块对应的swap map值为0
=> 相应交换空间空闲

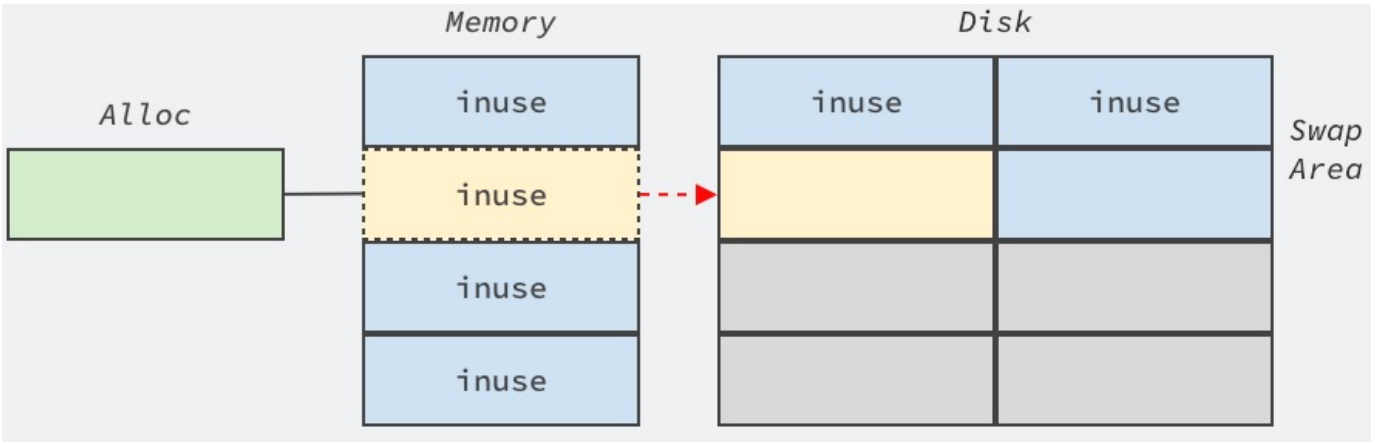
如果块对应的swap map值>0
=> 相应交换空间被占用

例：块对应的swap map值=3
=> 相应交换空间被映射到3个进程

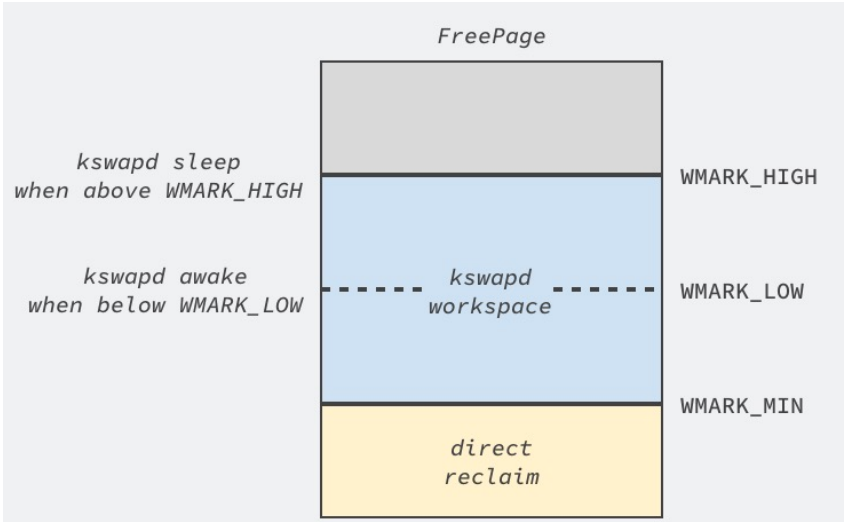
Linux交换机制简介

How linux swap works

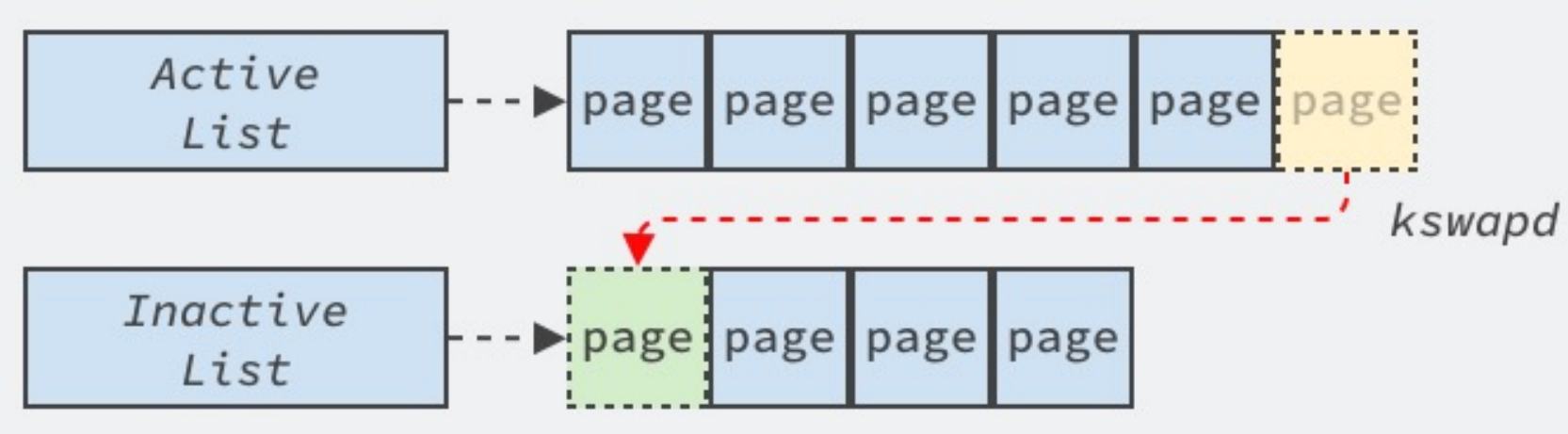




Swap out when memory is crowded



Linux交换机制简介



The Linux operating system uses the Least Recently Used (LRU) algorithm to replace pages in memory, and each zone in the system holds active_list and inactive_list chains in memory, where the former contains active memory pages and the latter stores memory pages that are candidates for recycling.

Whenever a memory page is accessed, Linux moves the accessed memory page to the head of the chain, so the 'oldest' memory page in the chain is at the end of the active chain. The role of the daemon kswapd is to balance the length of the two chains and move the memory page at the end of the active chain to the head of the inactive chain to be recycled, while the function shrink_zones is responsible for recycling the inactive memory pages in the LRU chain.

磁盘交换空间

Linux交换空间
-Linux交换空间形式是专门的分区（swap分区）
-大小通常为RAM的50%到100%

System	Swap Space
Solaris	Swap space is equal to the amount of physical memory.
Linux	Swap space is double the amount of physical memory

Summary

小结：

- 磁盘物理结构简介
- 磁盘调度算法
- 磁盘交换空间管理

操作系统

L22 I/O子系统（上）

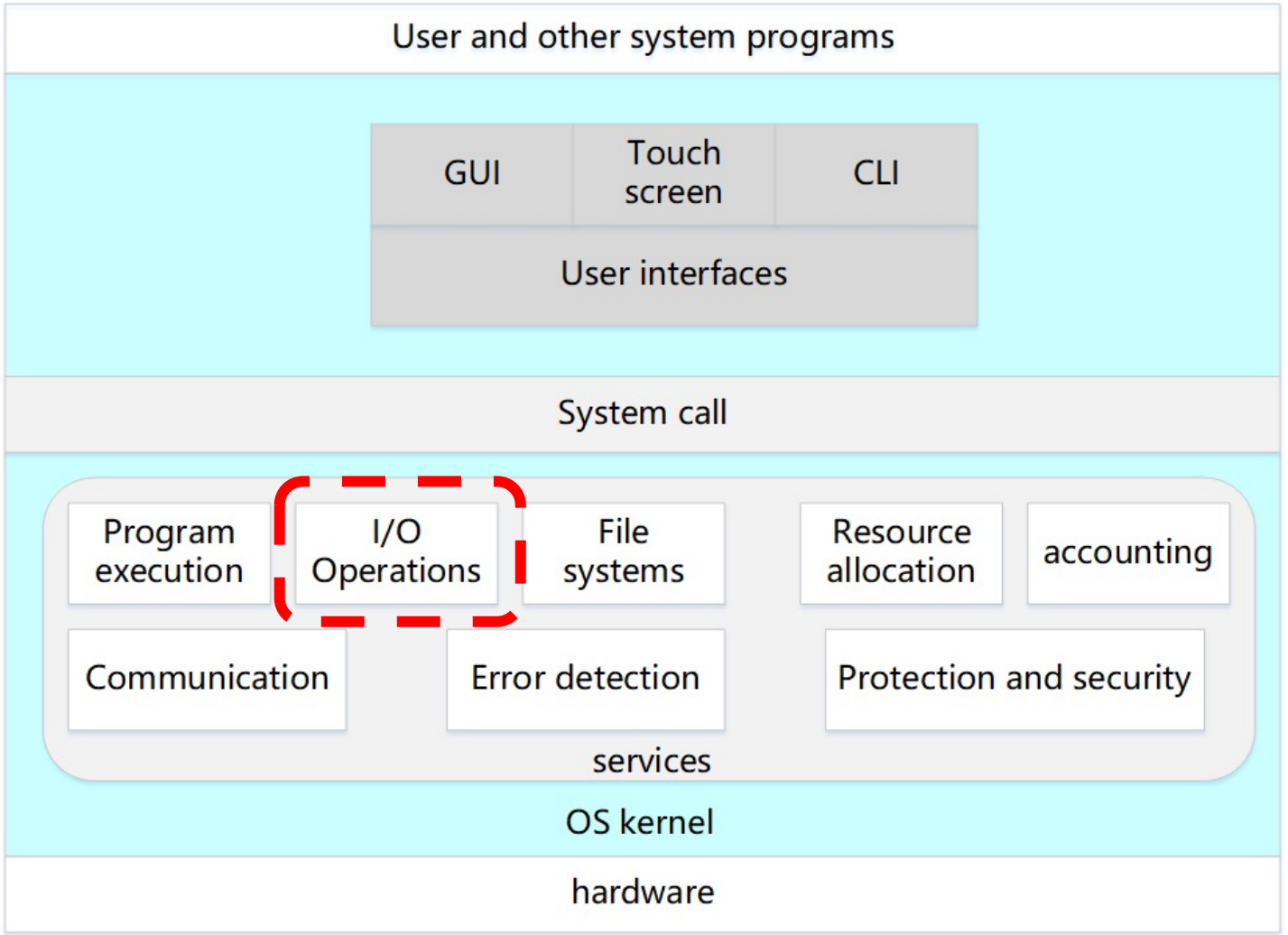
胡燕
大连理工大学 软件学院

I/O子系统概述

I/O Subsystem Introduction

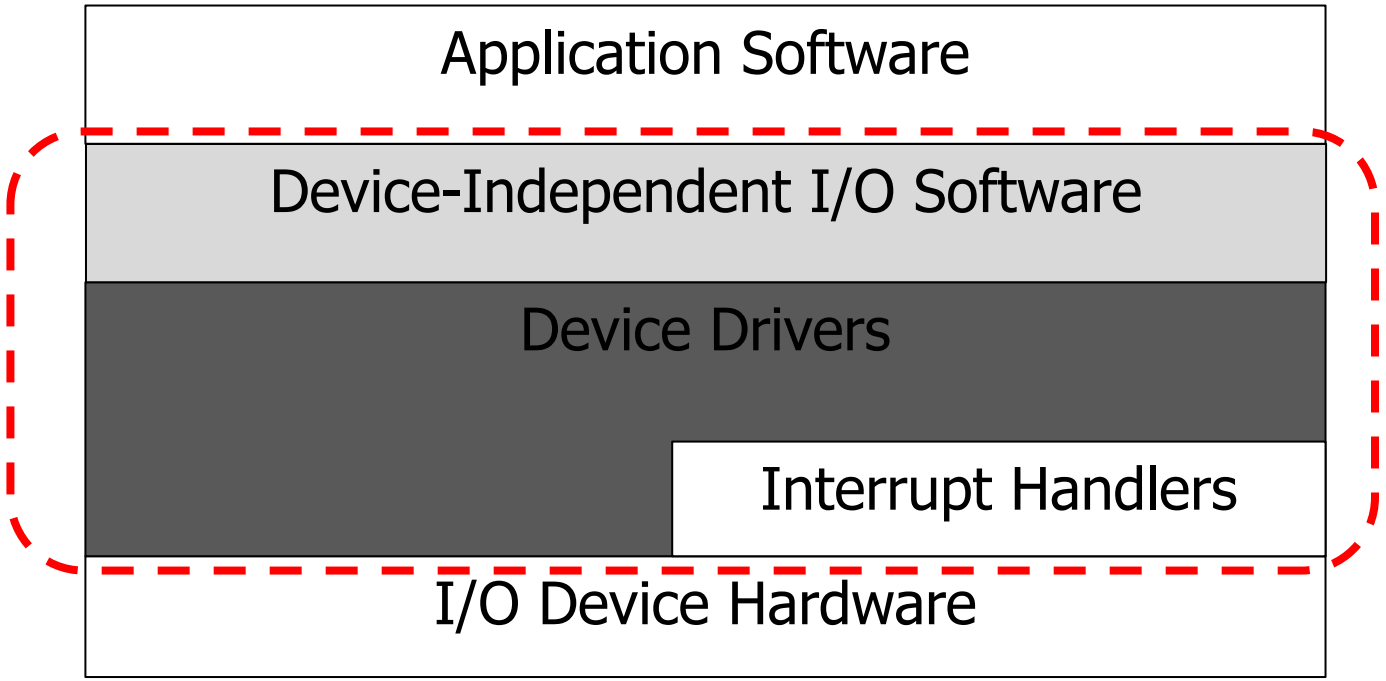
01

I/O子系统功能



需要I/O子系统来提供I/O方面的服务

I/O子系统功能

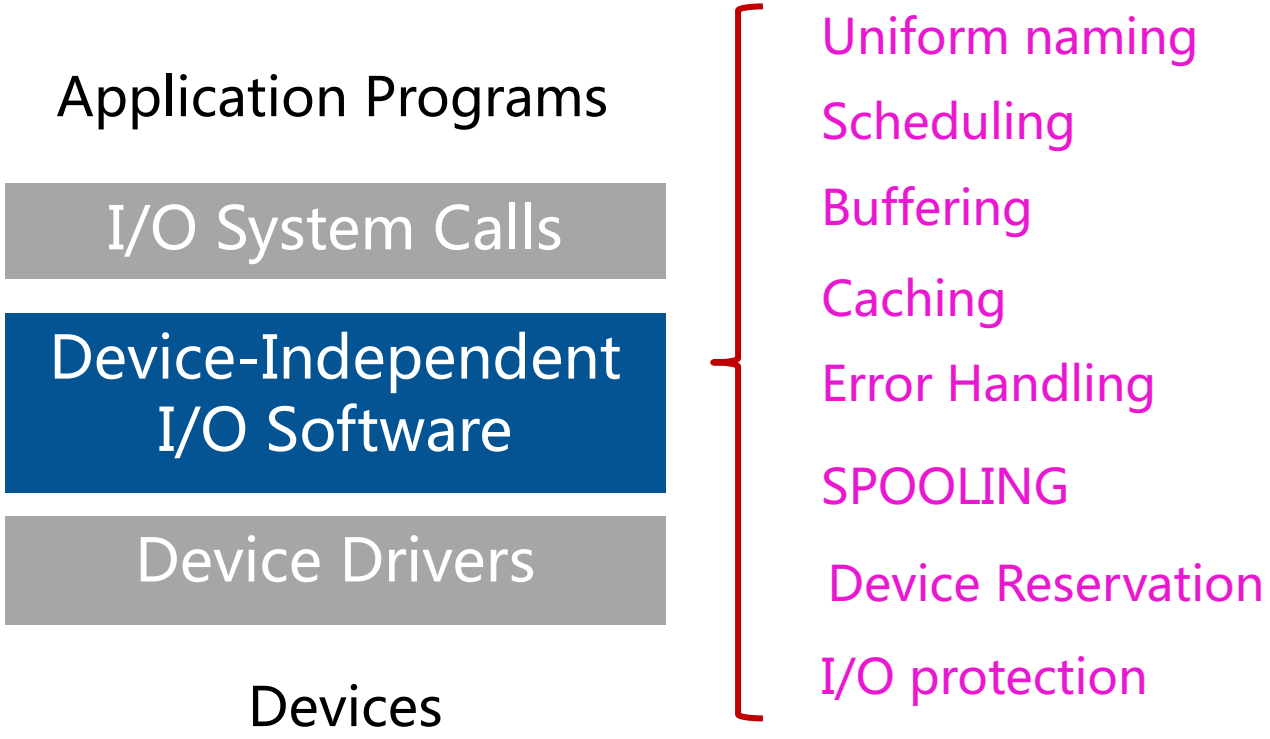


Generic

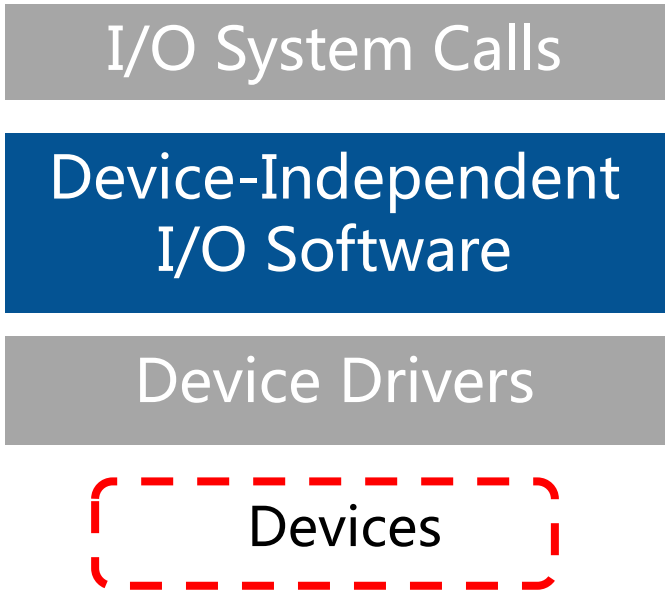
Specific Details

Kernel I/O Subsystem:
内核的I/O子系统与设备驱动程序和中断处理程序一起，对I/O设备进行管理，为应用层提供I/O相关的服务

I/O子系统功能


Application Programs
I/O System Calls
Device-Independent I/O Software
Device Drivers
Devices
Uniform naming
Scheduling
Buffering
Caching
Error Handling
SPOOLING
Device Reservation
I/O protection

I/O子系统功能

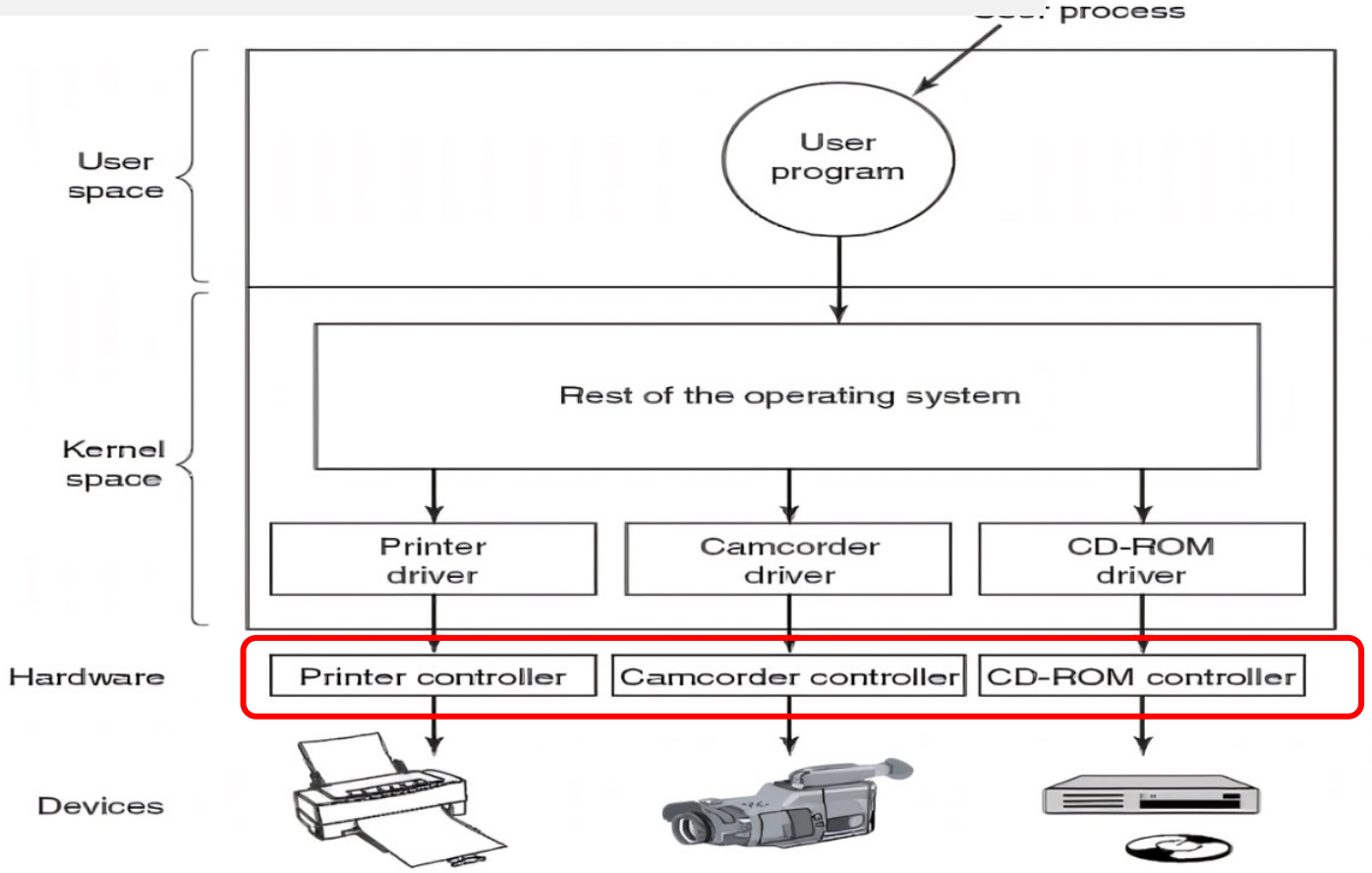


I/O子系统学习任务1：了解设备类别

设备控制器

I/O子系统学习任务2：了解设备控制器

设备控制器是IO设备的控制中枢

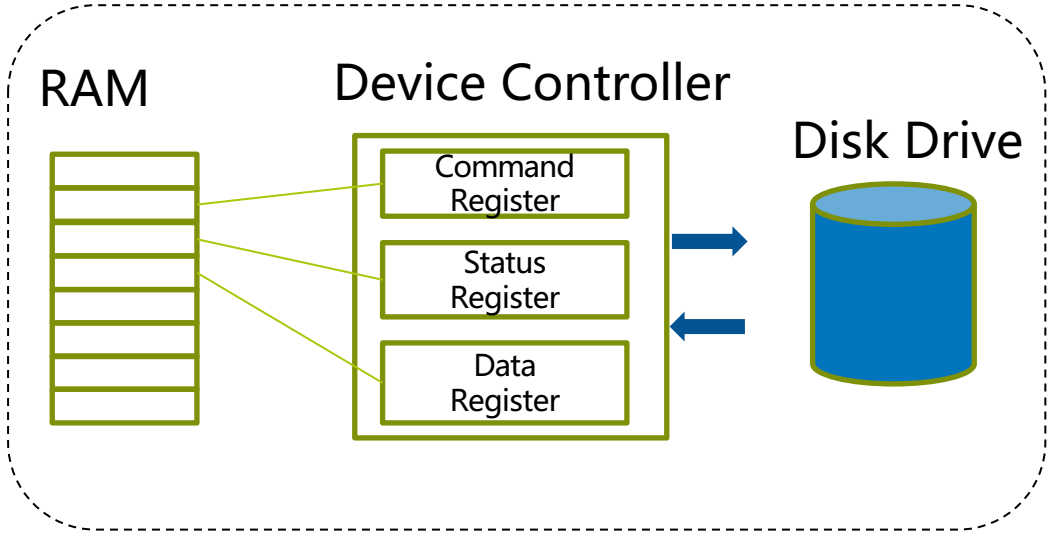


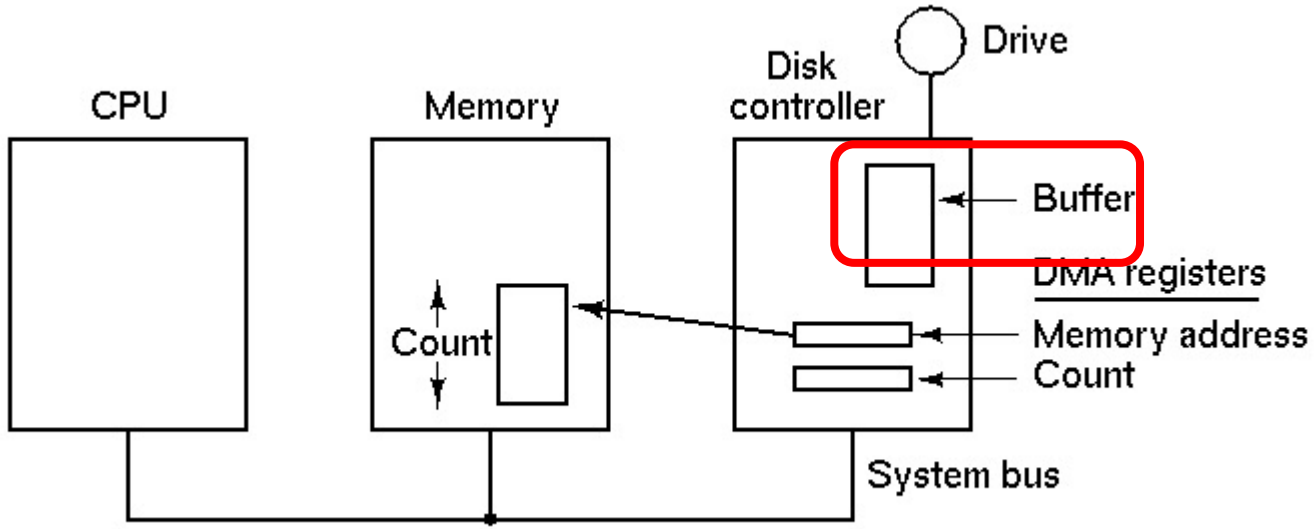
A controller is an electronic card (PC’s) or a unit (main frames) which preform blocking (from serial bit stream), analog signal generation (to move disk arm, to drive CRT tubes in screens), execution of I/O commands.

设备驱动程序，核心是对设备控制器进行编程

设备控制器

设备控制器示例：磁盘控制器

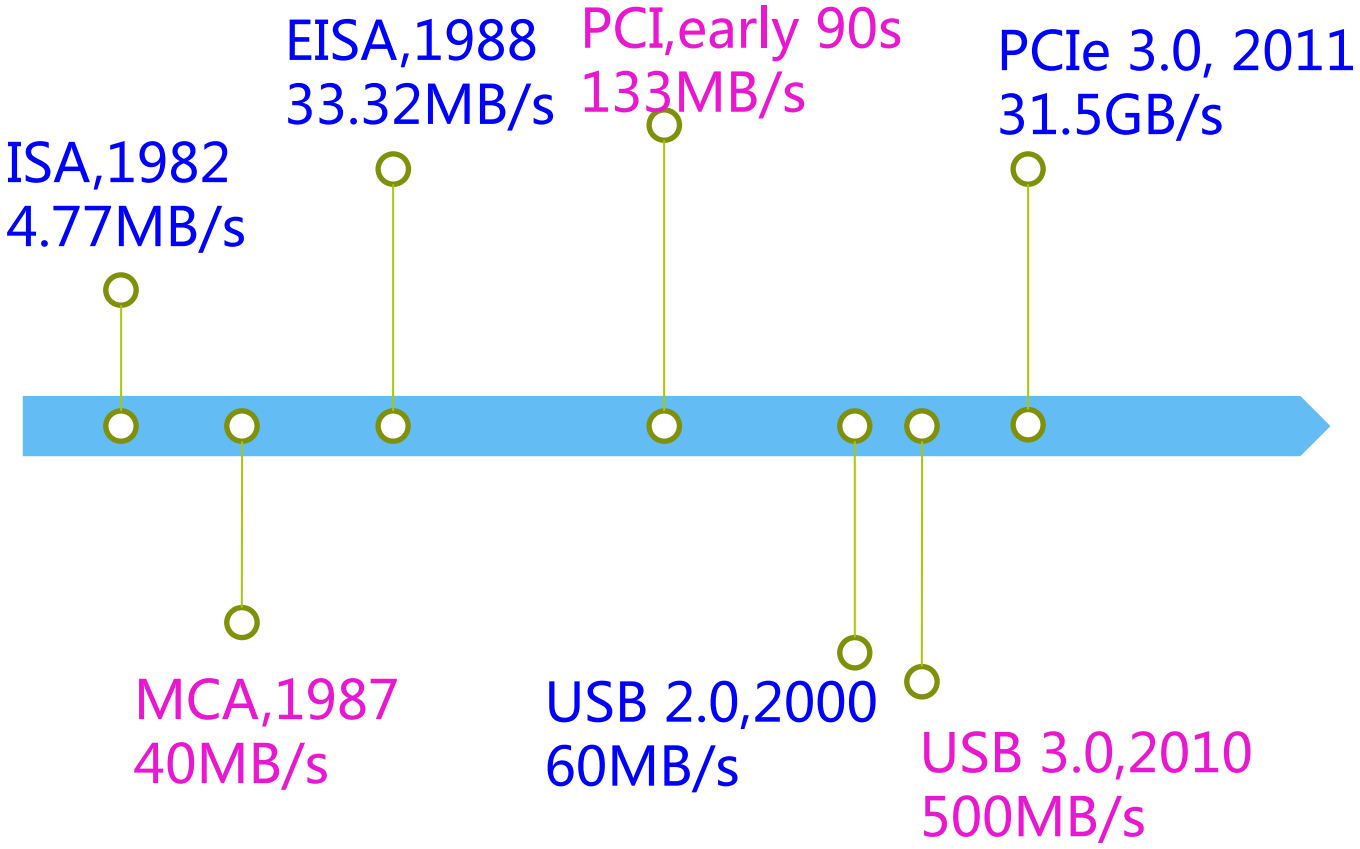




在I/O控制器的协同下，CPU与I/O并行工作（以输入为例）：
1-CPU发出I/O命令，设置设备控制器的命令寄存器和数据寄存器
2-I/O接收到命令，完成初始化，并启动I/O （此时CPU可以转做其他与此I/O无关事情）
3-I/O设备完成I/O，输入数据存储在控制器缓存
4-将设备控制器缓存中的数据写到内存（此时，涉及到与CPU竞争使用内存总线）
5-I/O完成，发出中断 （CPU检测到中断，转去执行中断处理例程）

Bus

PC总线发展史



PCIe 6.0版本
2022年1月发布
速度最高可达128GB/s

I/O子系统概念汇总

I/O子系统功能

I/O设备分类

设备控制器

总线（Bus）

I/O控制方式

I/O Control Mode

02

典型的I/O控制方式

通道控制IO

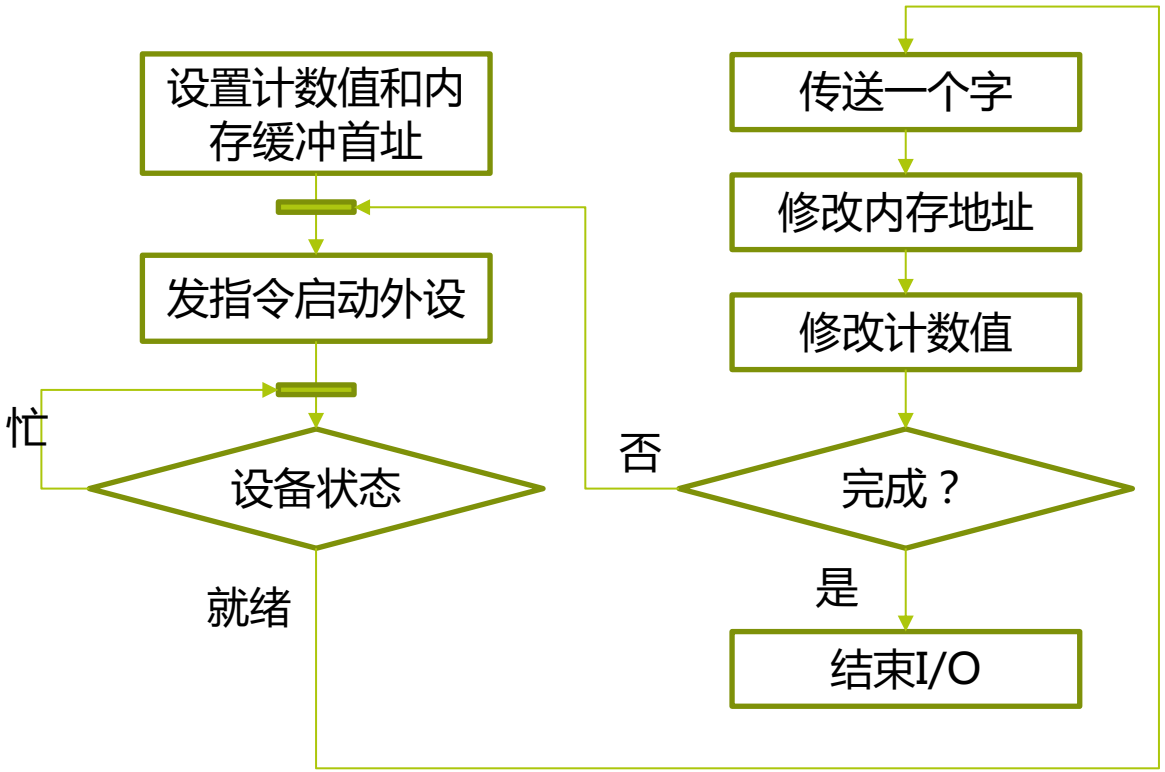
DMA

中断控制IO

程序控制IO

程序控制I/O

程序控制IO: CPU指令控制整个IO过程



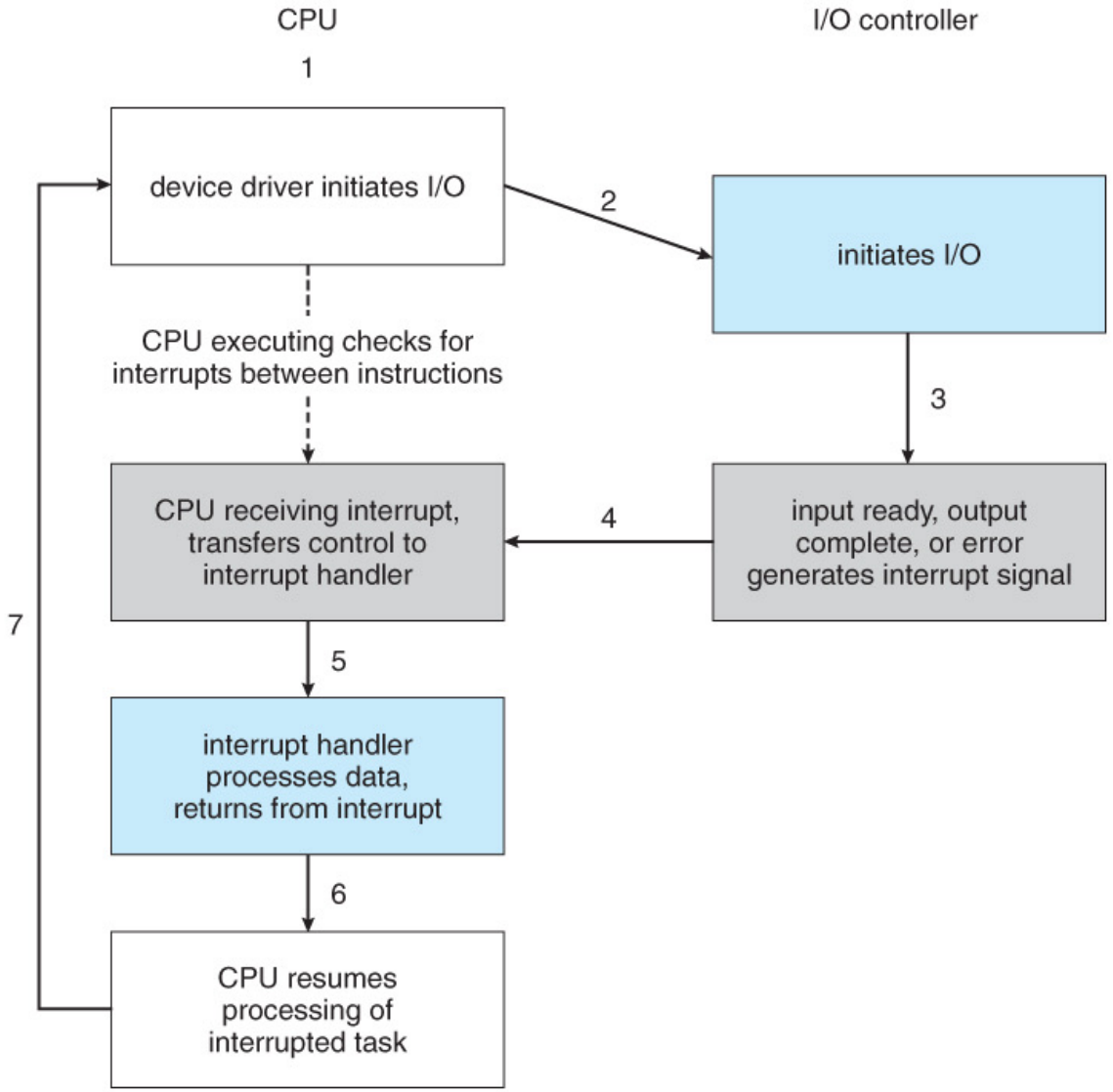
由用户进程直接控制主存或CPU 和外围设备之间的信息传送

又称轮询方式(忙等方式)

编程控制IO在旧接口的设备上使用
-串口
-非ECP模式的并口

程序控制I/O方式的缺点：
CPU过度参与I/O，忙等（浪费CPU资源）

中断控制I/O



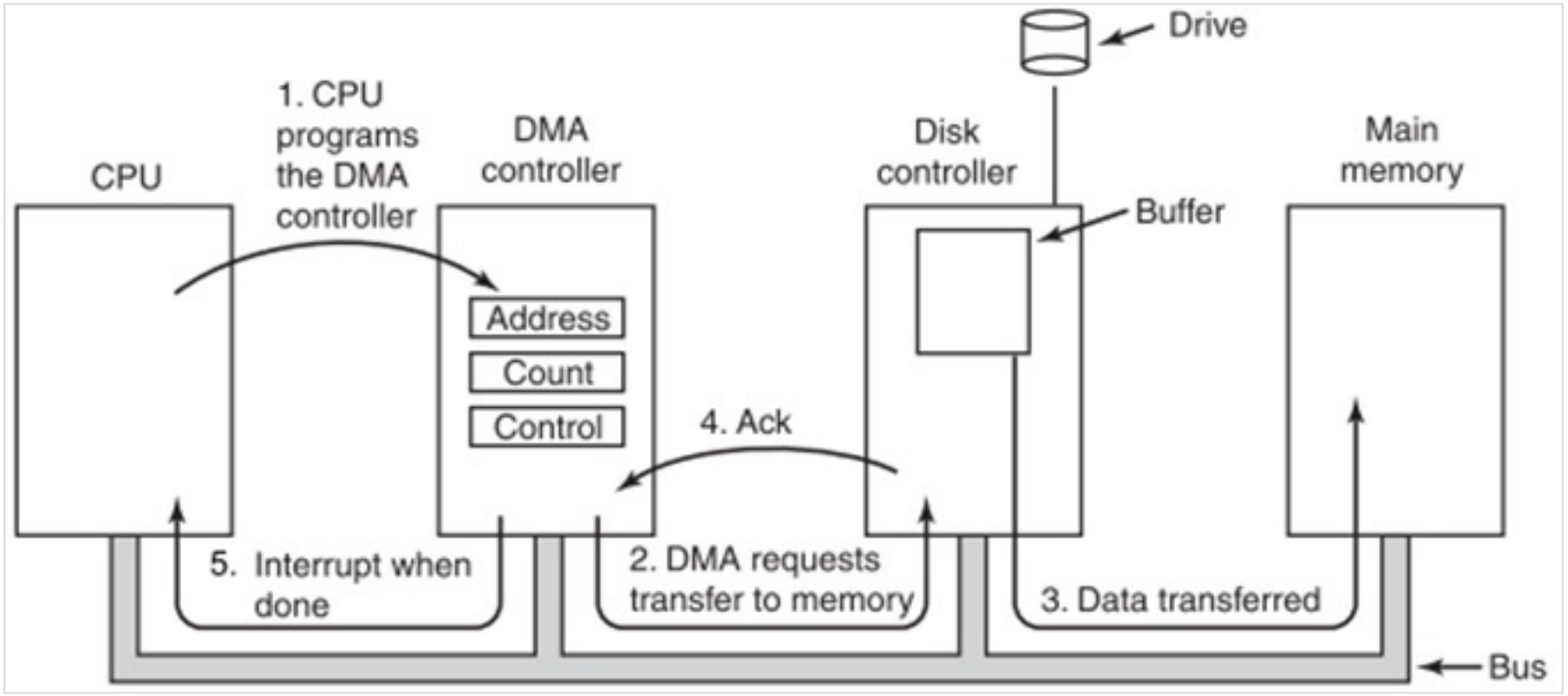
中断控制I/O（Interrupt-driven IO）

在I/O操作进行期间，CPU正常执行其他与I/O无关操作（只是，在每条指令的末尾，CPU的硬件逻辑中会有对中断信号的检查）

优势:
CPU与I/O设备并行工作（能够显著提升系统效率）

DMA控制

Direct Memory Access (直接内存访问)



DMA控制

DMA的两种工作模式：Cycle Stealing （周期窃取）和Burst模式

Burst:
DMA controller acquires the bus (exclusively), issues several transfers, and releases.

Cycle Stealing:
DMA controller grabs bus for one word at a time, it competes with CPU bus access.

机器指令周期
1.IF：Instruction Fetch(must mem access)
2.DE：Decode
3.FO：Fetch Operand(optional mem access)
4.EX：Execute
5.WM：Write result to Memory(optional mem access)

- 尽量使用CPU不使用Memory的时间，DMA占用系统总线进行内存与I/O Device之间的数据传输
- 万一CPU与DMA controller对memory存取发生conflict，则给DMA较高的优先权：将CPU暂停一个周期

通道

大型计算机系统中采用通道处理机

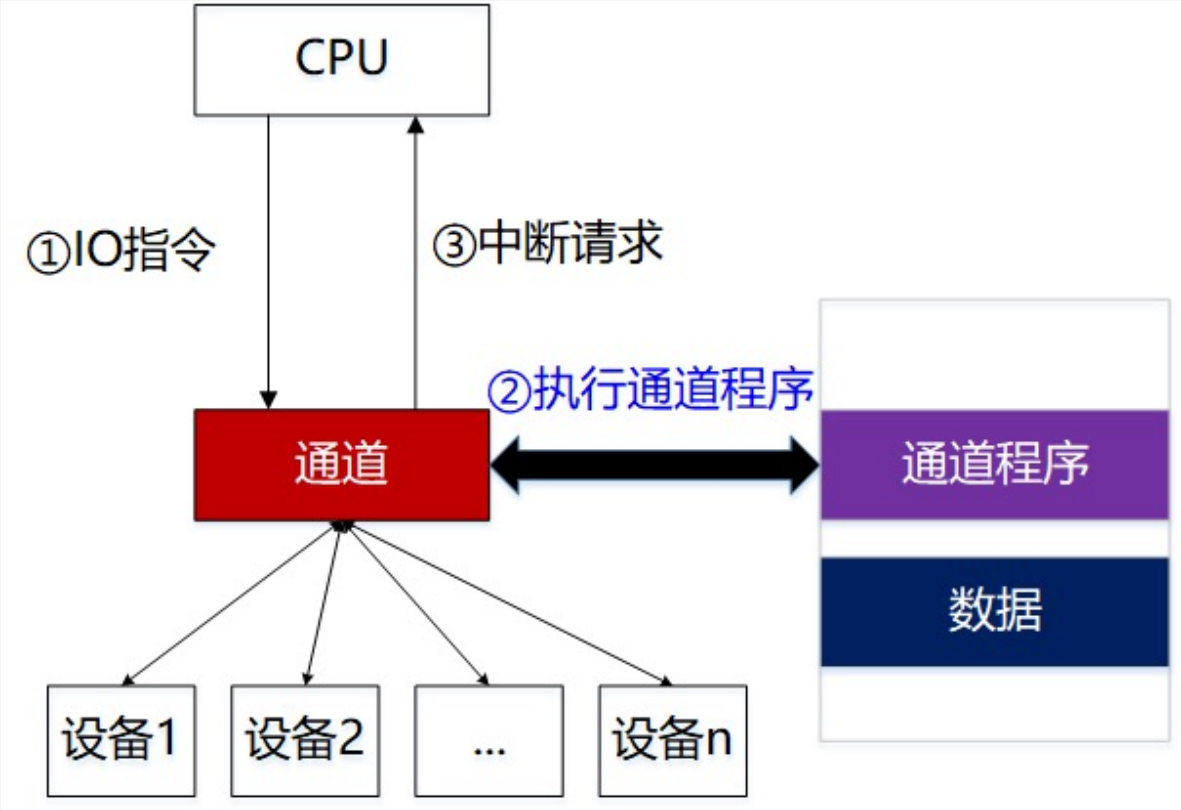
- 使CPU摆脱繁重的IO处理负担
- 共享IO接口

大型计算机系统中，如果仅采用程序控制、DMA等常规IO控制方式，在进行设备管理时，会面临如下问题

- 大量外围设备的I/O工作要由CPU管理，挤占CPU支持应用程序执行的时间
- 大型计算机系统中的外围设备很多，但一般不同时工作
  - 为每个设备都配备一个接口，代价很高

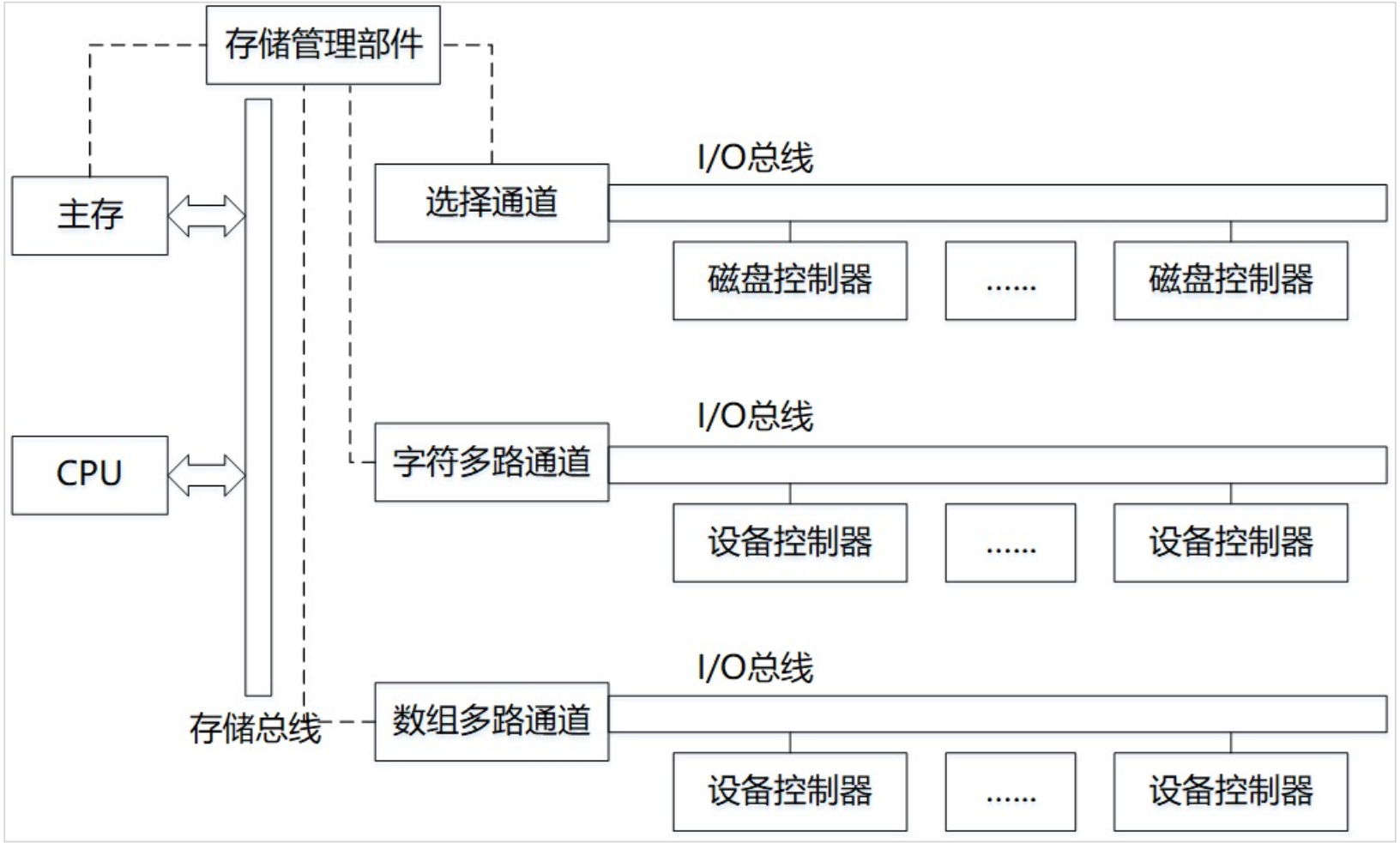
通道

Channelled IO
通道控制



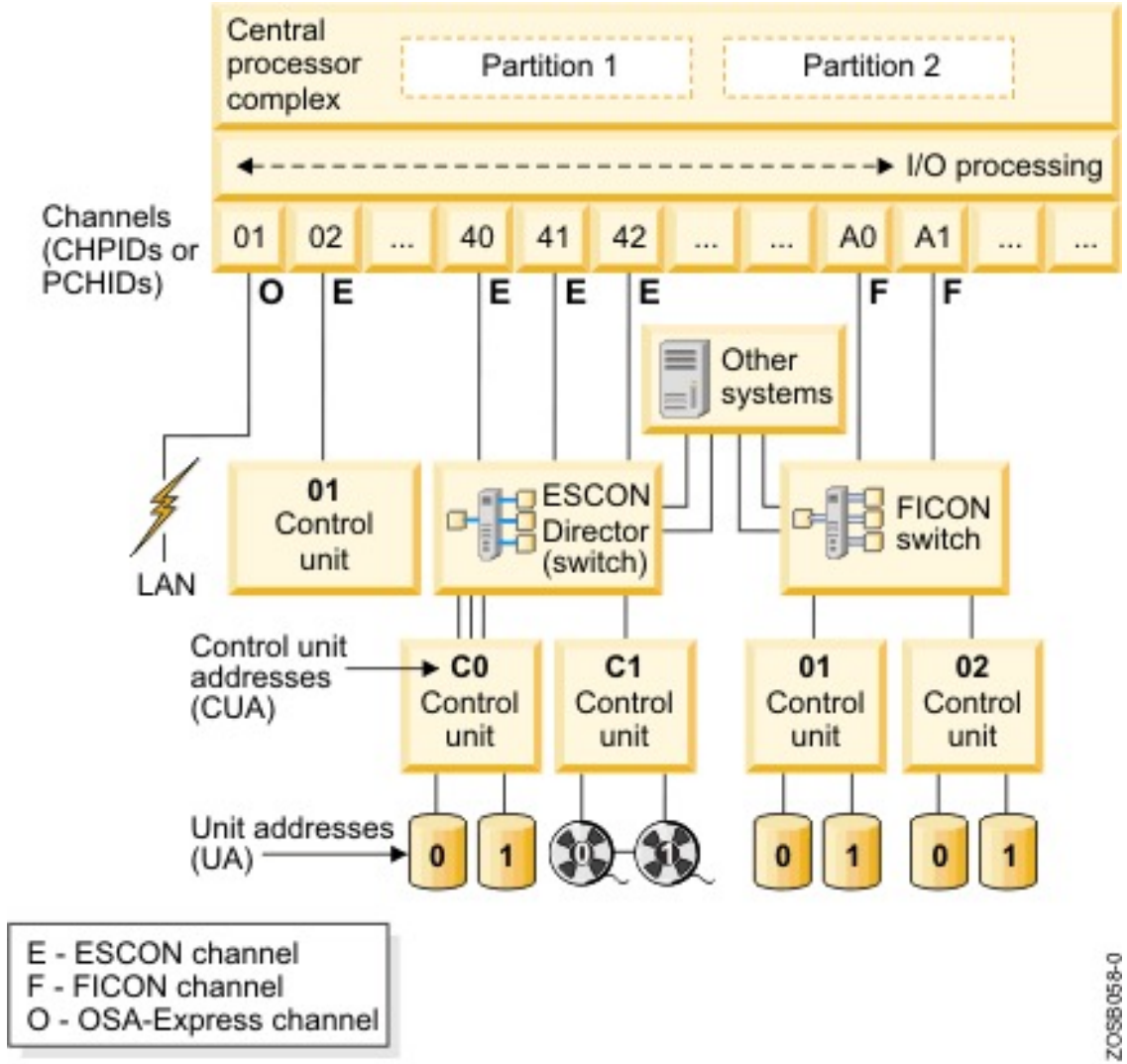
通道

具有通道的计算机系统典型结构



通道

IBM z9 io structure Demostration



I/O缓冲机制

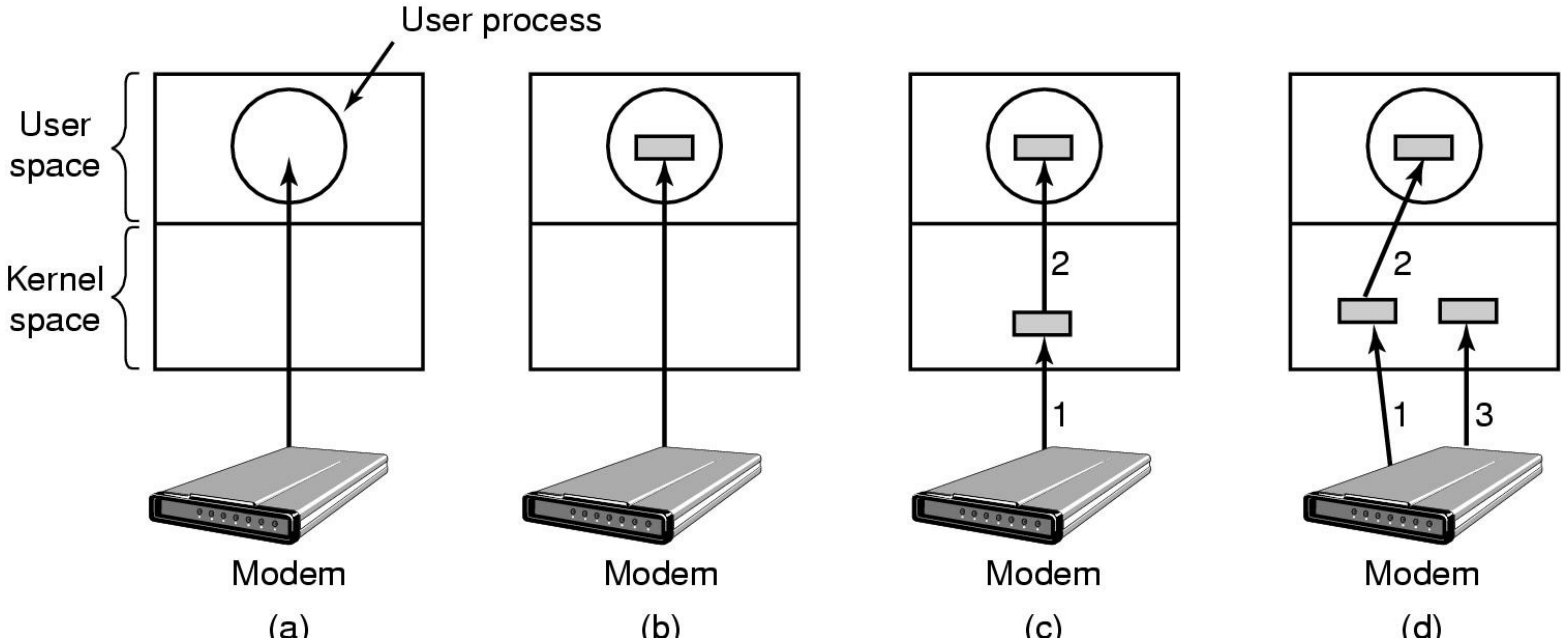
Buffering in I/O Subsystem

03

缓冲概念

I/O子系统中引入Buffering机制的核心目的：

缓解CPU与I/O速度之间速度不匹配的矛盾

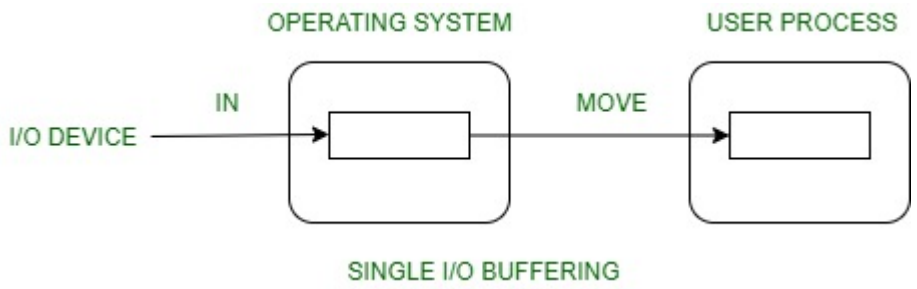


(a) Unbuffered input
(b) Buffering in user space
(c) Buffering in the kernel followed by copying to user space
(d) Double buffering in the kernel

缓冲区类型

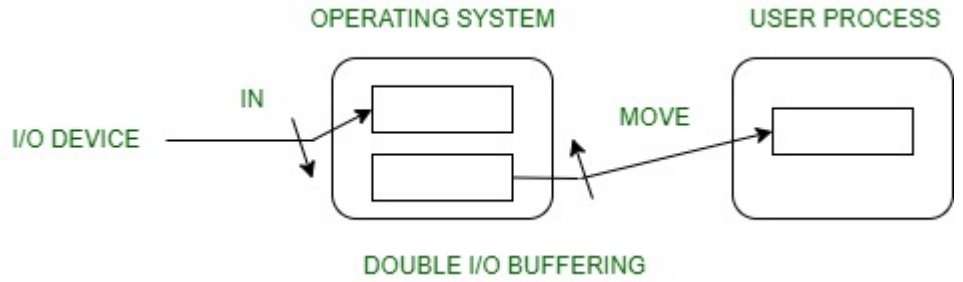
Single Buffer

-提供足够大的单个缓冲区，可以减少访问设备的频次，提升效率



Double Buffer

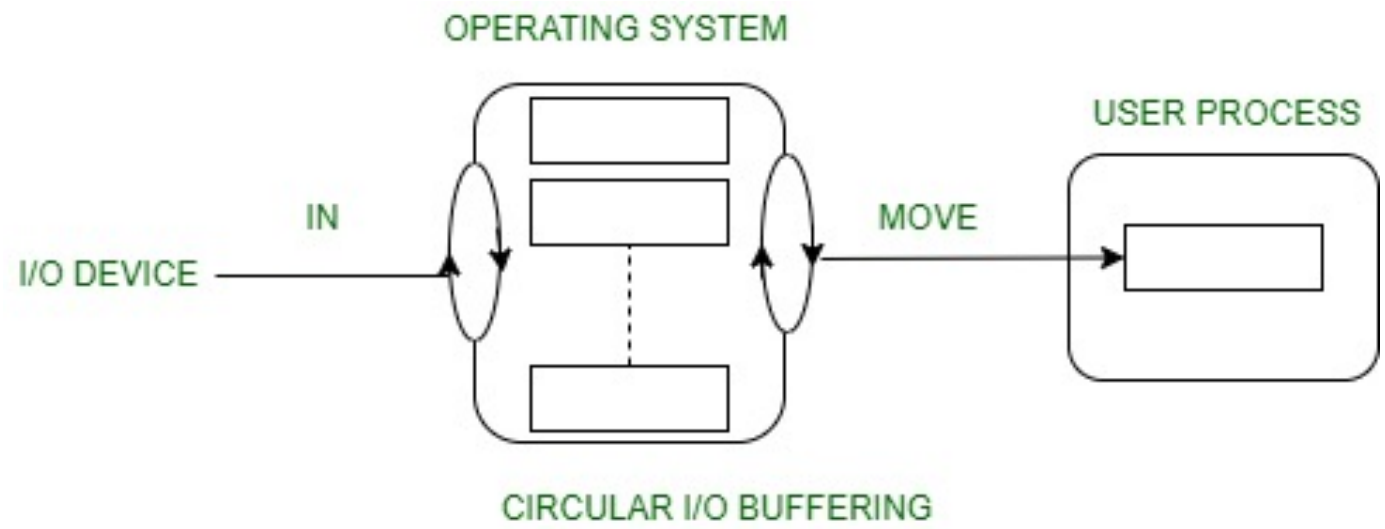
-对单缓冲的优化，通过增加一个系统buffer来获得数据输入与数据处理的并发



Android系统内对GPU渲染处理过程应用了双缓冲机制，来提升处理效率，避免画面卡顿

环形缓冲

双缓冲依旧存在的不足：
-无法应对IO bursts（例如，网卡，可能突然有大量数据涌入，需要IO子系统处理）

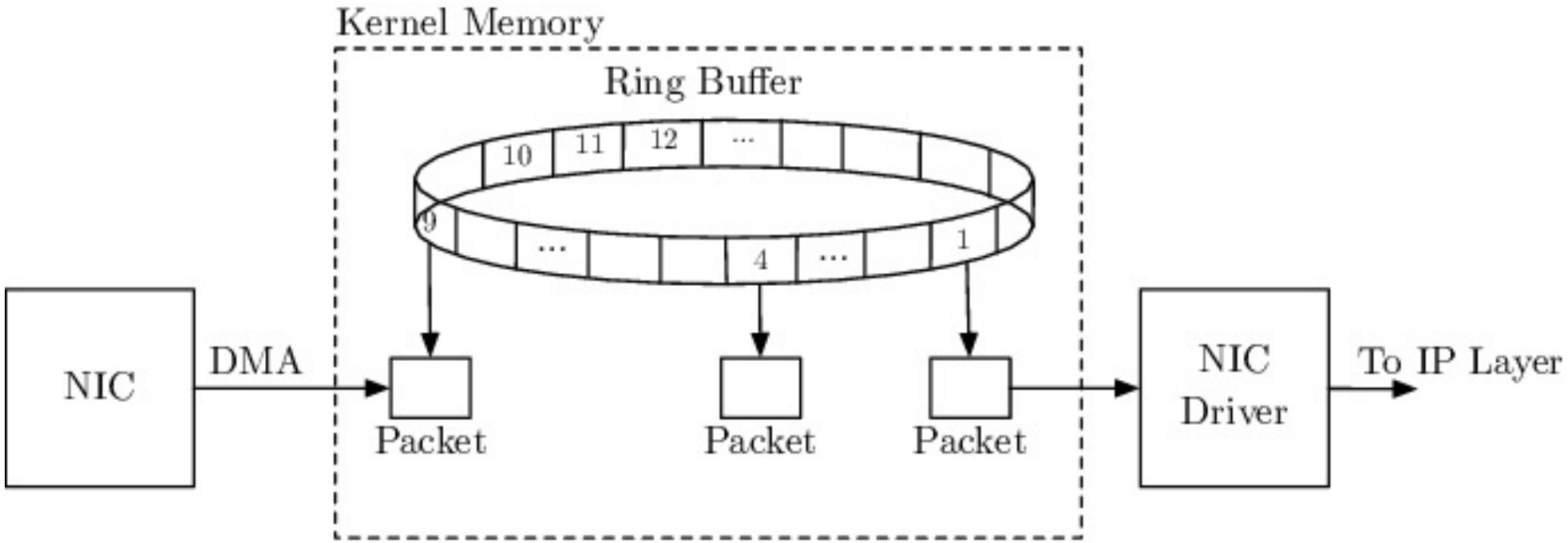


Circular Buffer
-有限环形缓冲，通过缓冲区数量的扩容，加上有限缓冲的并发处理，可以应对IO Bursts

环形缓冲

双缓冲依旧存在的不足：

-无法应对IO bursts（例如，网卡，可能突然有大量数据涌入，需要IO子系统处理）



Summary

小结：

- I/O子系统概述
- I/O控制方式
- I/O缓冲机制

讨论

Linux中的设备分类

字符设备、块设备、网卡设备

不同的I/O控制方式

设备驱动模式有何区别？

关于缓冲池

缓冲池的优势

谢 谢 ！
Thank you!