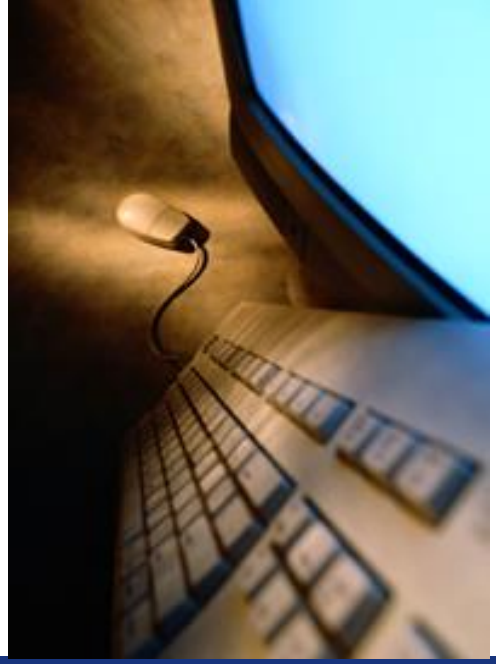
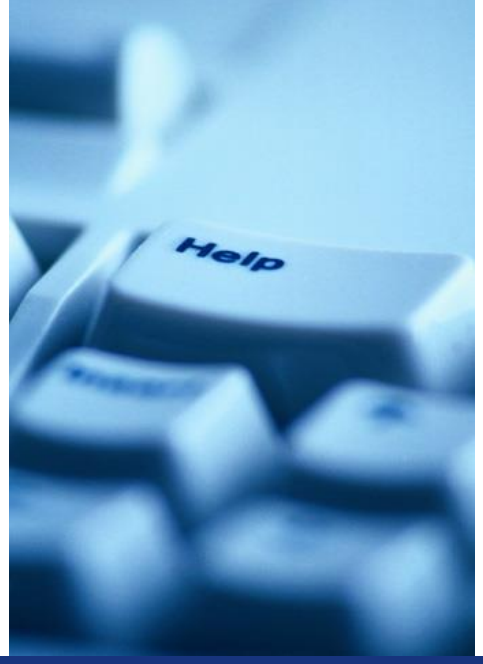
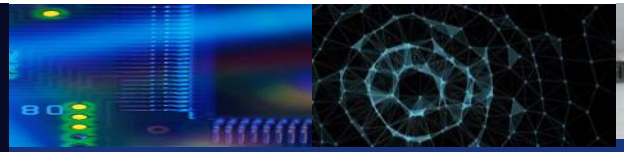



操作系统原理

Operating Systems Principles

张青
计算机学院

第十一讲 — 大容量存储



目标

- 描述外存设备的物理结构及其对设备使用的影响；
- 解释大容量存储设备的性能特点；
- 评估磁盘调度算法；
- 讨论大容量存储（RAID）提供的操作系统服务；

存储系统

User Space

Processes

Operating System Kernel

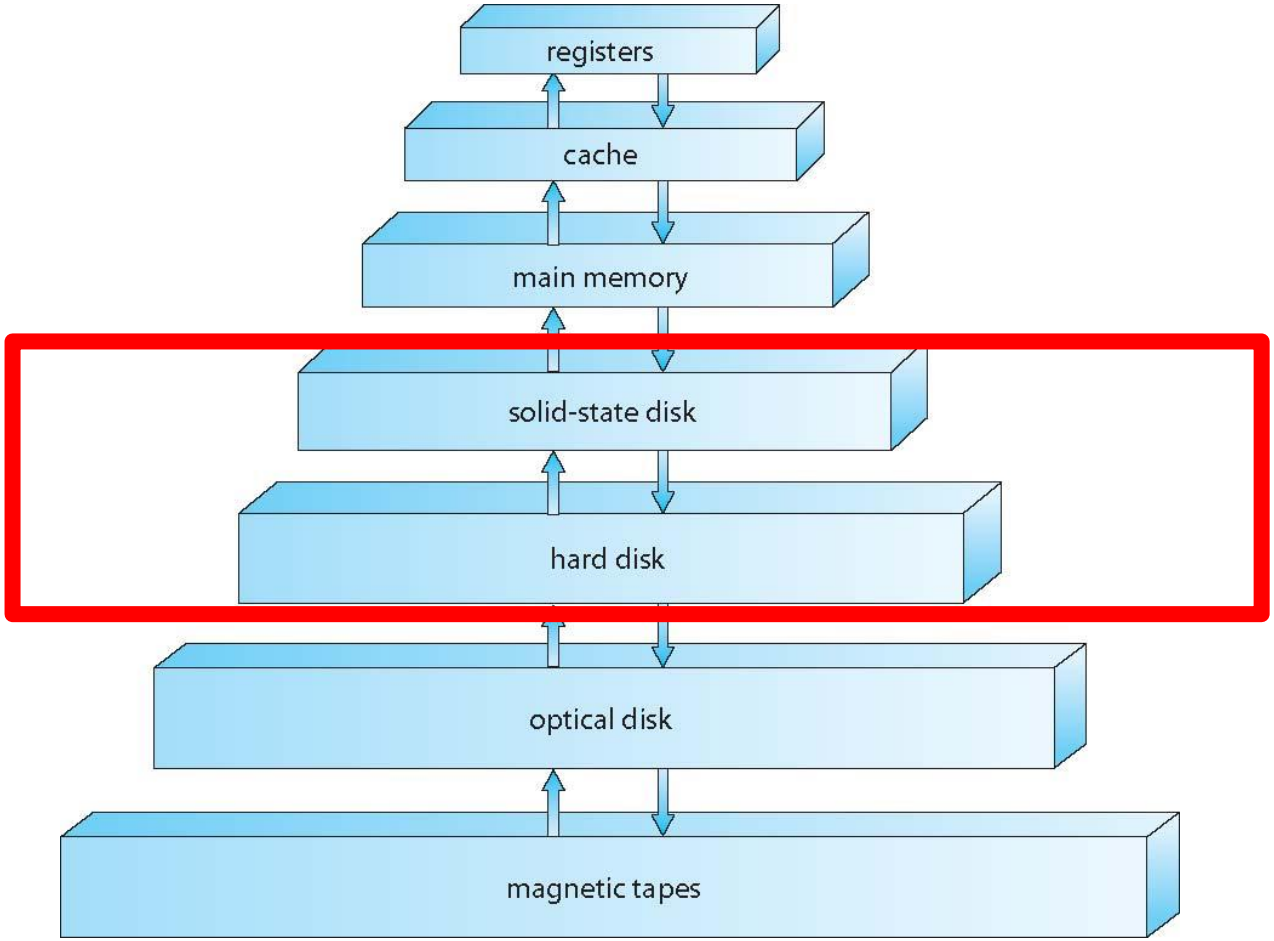
File system Implementation

FAT32, EXT2/3
KV, Distributed FS,
Graph System…

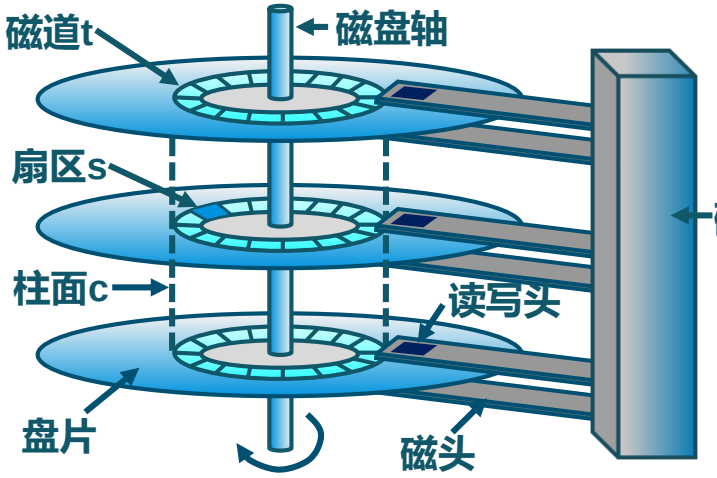
File System Operations

Devices

存储分层


registers
cache
main memory
solid-state disk
hard disk
optical disk
magnetic tapes

磁盘工作机制：移动磁头的磁盘装置



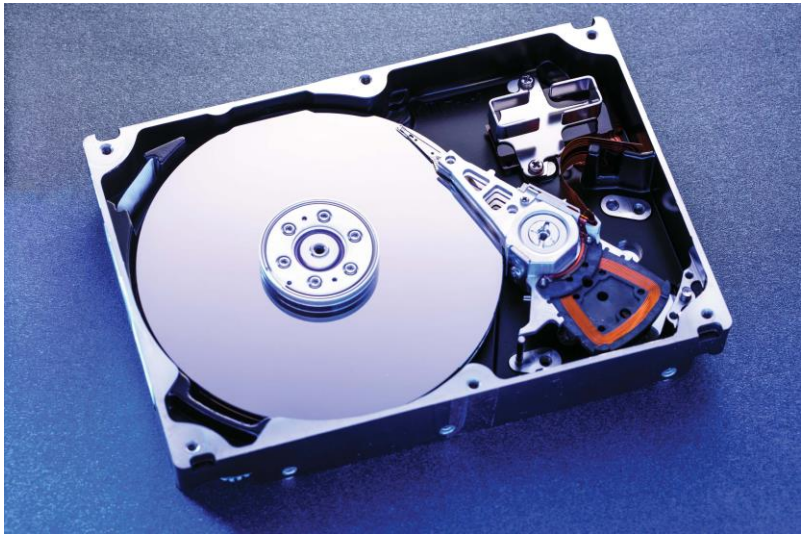
驱动器电机高速旋转磁盘，每秒旋转60-250次，一般按每分钟转数（rotation per minute, RPM）来计

- 读取或写入时，磁头必须被定位在**期望的磁道**，并从所**期望的柱面和扇区**的开始
- 寻道时间
  - 定位到期望的磁道所花费的时间
- 旋转延迟
  - 从零扇区开始处到达目的地花费的时间

平均旋转延迟时间=磁盘旋转一周时间的一半

磁盘结构

- **普通盘片的直径为1.8-3.5英寸（1英寸=2.54厘米）**
  - 通常是 3.5、2.5、1.8
- **一般容量在 30GB到3TB之间**
- **性能**
  - 传输速率 – 理论上 – 6 Gb/sec
  - 有效传输速率 – 实际值 – 1Gb/sec
  - 寻道时间在 3ms到12ms – 通常是9ms
  - 基于主轴速度的延迟
    - 1 / (RPM / 60) = 60 / RPM



HDD性能

- **Access Latency = Average access time = average seek time + average latency**
  - For fastest disk 3ms + 2ms = 5ms
  - For slow disk 9ms + 5.56ms = 14.56ms
- **Average I/O time = average access time + (amount to transfer / transfer rate) + controller overhead**
- **For example to transfer a 4KB block on a 7200 RPM disk with a 5ms average seek time, 1Gb/sec transfer rate with a .1ms controller overhead =**
  - 5ms + 4.17ms + 0.1ms + transfer time =
  - Transfer time = 4KB / (1Gb/s /(8 * 1024) = 32 / (1024^2) = 0.031 ms
  - Average I/O time for 4KB block = 9.27ms + .031ms = 9.301ms

磁盘调度

磁盘性能参数

- 磁盘I/O的实际操作细节取决于以下几个因素：
  - 计算机系统；
  - 操作系统；
  - I/O通道和磁盘控制器硬件特性；

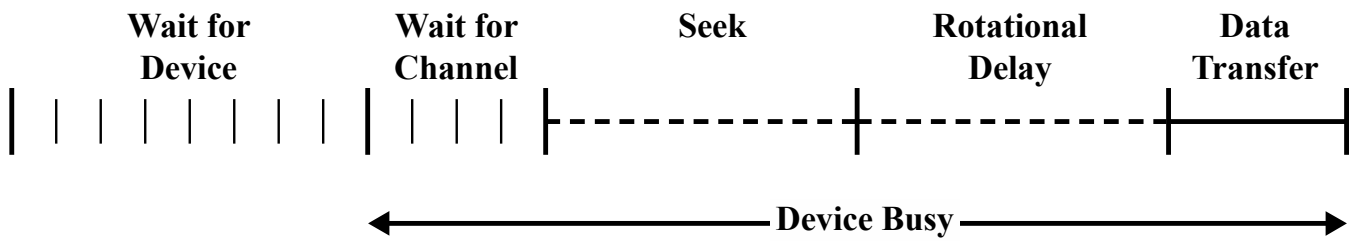


Figure 11.6 Timing of a Disk I/O Transfer

磁盘I/O传输时间

$$T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$$

访问时间

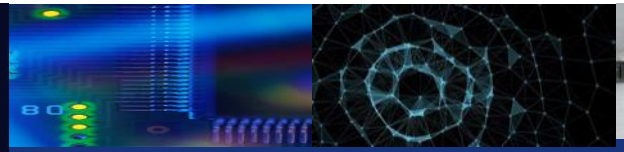
第一块硬盘

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

非易失性内存（固态硬盘）

- 固态硬盘（SSD）为非易失性存储，可用作硬盘
- 有多个变形，如带有电池的DRAM以及闪存技术
- SSD具有与传统硬盘相同的特性，但具有如下优势：
  - ✓ 更加可靠，因为没有移动部件
  - ✓ 更快，因为没有寻道或旋转延迟
  - ✓ 电源消耗更少
- SSD的每兆字节更昂贵，寿命也更短，所以用途有限。常用于存储文件系统元数据以提高性能，以及一些追求更小、更快、更节能的笔记本电脑
- 由于SSD速度很快，标准总线接口可能对吞吐量造成限制。因此有些SSD设计成直接连接到系统总线（如PCI）
- SSD没有磁头，所以调度算法不适用

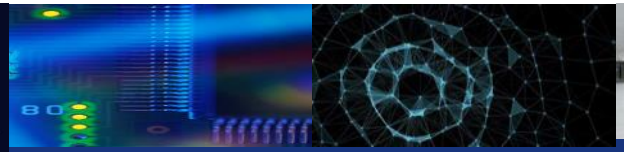


易失性存储

- 常用作大容量存储设备的DRAM
  - 技术上不是辅助存储(磁盘)，因为易失性，但可以有文件系统，可以像非常快速的辅助存储一样使用
- RAM驱动器（具有许多名称，包括RAM磁盘）作为原始块设备显示，通常为文件系统格式
- 计算机通过RAM进行缓冲和缓存
  - 由程序员、操作系统、硬件分配/管理的缓存/缓冲区
  - 用户控制下的RAM驱动器
  - 在所有主要的操作系统中都可以找到
    - Linux/dev/ram，创建它们的macOS diskutil，文件系统类型为tmpfs的Linux/tmp
- 用作高速临时存储器
  - 程序可以通过读/写RAM驱动器快速共享批量数据

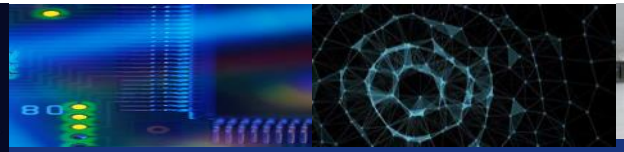
磁盘与主机之间的连接

- 通过与I/O总线的I/O端口连接到计算机；
- 有几种总线可用，包括硬盘接口连接（ATA）、串行ATA（Serial ATA, SATA）、外部串行ATA(eSATA)、串行连接SCSI（SAS）、通用串口总线（USB）和光纤通道（FC）。
- 最常见的是SATA
- 由称为控制器（controller）的专门电子处理器在总线上进行数据传输
  - 主机控制器（host controller）为总线在计算机端的控制器，磁盘控制器（disk controller）是磁盘驱动器内置的；
  - 计算机使用内存映射的I/O端口将命令放置在主机控制器上
    - 主机控制器向设备控制器发送消息;
    - 通过DMA在设备和计算机DRAM之间传输的数据



磁盘结构

- 现代磁盘驱动器可以看作逻辑块的一位数组
- 其中逻辑块是最小的传输单元，通常为512字节
  - 有些磁盘可以通过低级格式化来选择不同大小的逻辑块
- 逻辑块的一维数组按顺序映射到磁盘的扇区
  - 扇区0是最外层圆柱体上第一个轨迹的第一个扇区
  - 磁头依次通过该磁道，然后是该柱面中的其余磁道，然后从最外层到最内层扫描其他柱面；
  - 执行逻辑到磁盘地址的映射难：
    - 大多数磁盘都有一些坏扇区；映射必须用磁盘上的其他空闲扇区来替代这些缺陷扇区。此外，有些驱动器，每个磁道的扇区数并不是常量
    - 通过恒定角速度的方式访问，每条磁道的扇区数不同；最外层的轨道通常比最内层的轨道多拥有40%的扇区



磁盘调度

- 操作系统负责高效地使用硬件——对于磁盘驱动器，这意味着具有快速的访问时间和较宽的磁盘带宽；
- 最小化寻道时间；

  寻道时间是磁臂移动到目标扇区的柱面的时间；
- 磁盘带宽（ Disk bandwidth ）是传输的总字节数除以从服务请求开始到最后传输完成之间的总时间；
- 每当进程需要进行磁盘I/O操作时，它就向操作系统发出一个系统调用，这个请求涉及如下信息：
  - ✓ 这个操作是输入还是输出
  - ✓ 传输的磁盘地址是什么
  - ✓ 传输的内存地址是什么
  - ✓ 传输的扇区数是多少
- 如果所需的磁盘驱动器和控制器空闲，则立即处理请求。如果磁盘驱动器或控制器忙，则任何新的服务请求都会添加到磁盘驱动器的待处理请求队列

磁盘调度

- 磁盘I/O请求的来源很多
  - 操作系统
  - 系统进程
  - 用户进程
- I/O请求包括输入或输出模式、磁盘地址、内存地址、要传输的扇区数；
- 操作系统维护每个磁盘或设备的请求队列；
- 空闲磁盘可以立即处理I/O请求，繁忙磁盘意味着工作必须排队
  - 只有当队列存在时，优化算法才有意义；
- 过去，操作系统负责队列管理、磁盘驱动器调度
  - 现在，内置到存储设备本身的控制器中
  - 只需提供LBA（logical block addressing），处理请求的排序
    - 下面将介绍他们使用的一些算法

磁盘调度算法

磁盘I/O请求的磁道序列（初始时磁头位于编号100的磁道）：

55, 58, 39, 18, 90, 160, 150, 38, 184.

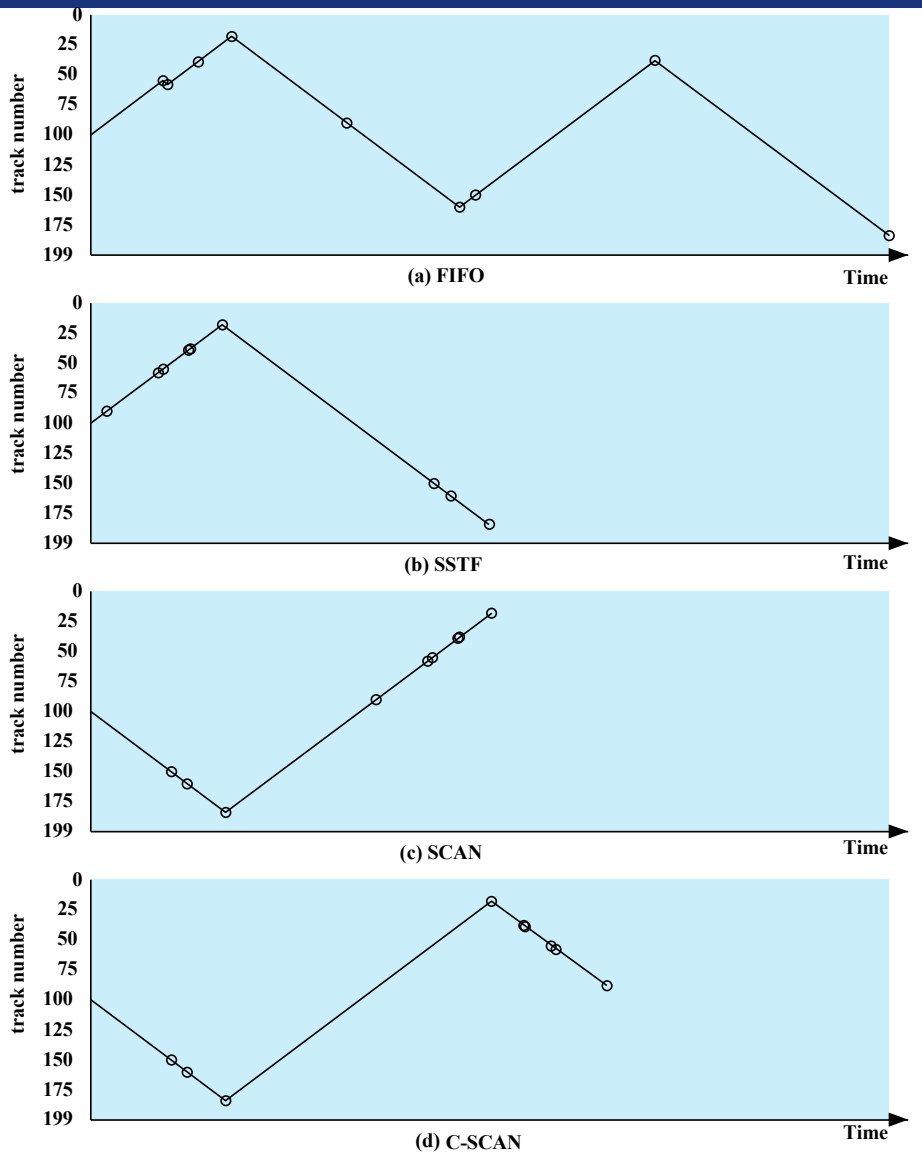


Figure 11.7 Comparison of Disk Scheduling Algorithms (see Table 11.3)

不同的磁盘调度算法

55, 58, 39, 18, 90, 160, 150, 38, 184.

(a) FIFO (starting at track 100)		(b) SSTF (starting at track 100)		(c) SCAN (starting at track 100, in the direction of increasing track number)		(d) C-SCAN (starting at track 100, in the direction of increasing track number)	
Next track accessed	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**	**Next track accessed**	**Number of tracks traversed**
55	45	90	10	150	50	150	50
58	3	58	32	160	10	160	10
39	19	55	3	184	24	184	24
18	21	39	16	90	94	18	166
90	72	38	1	58	32	38	20
160	70	18	20	55	3	39	1
150	10	150	132	39	16	55	16
38	112	160	10	38	1	58	3
184	146	184	24	18	20	90	32
Average seek length	55.3	**Average seek length**	27.5	**Average seek length**	27.8	**Average seek length**	35.8

Table 11.2 Comparison of Disk Scheduling Algorithms

磁盘调度算法

Name	Description	Remarks
Selection according to requestor		
Random	Random scheduling	For analysis and simulation
FIFO	First in first out	Fairest of them all
PRI	Priority by process	Control outside of disk queue management
LIFO	Last in first out	Maximize locality and resource utilization
Selection according to requested item		
SSTF	Shortest service time first	High utilization, small queues
SCAN	Back and forth over disk	Better service distribution
C-SCAN	One way with fast return	Lower service variability
N-step-SCAN	SCAN of N records at a time	Service guarantee
FSCAN	N-step-SCAN with N = queue size at beginning of SCAN cycle	Load sensitive

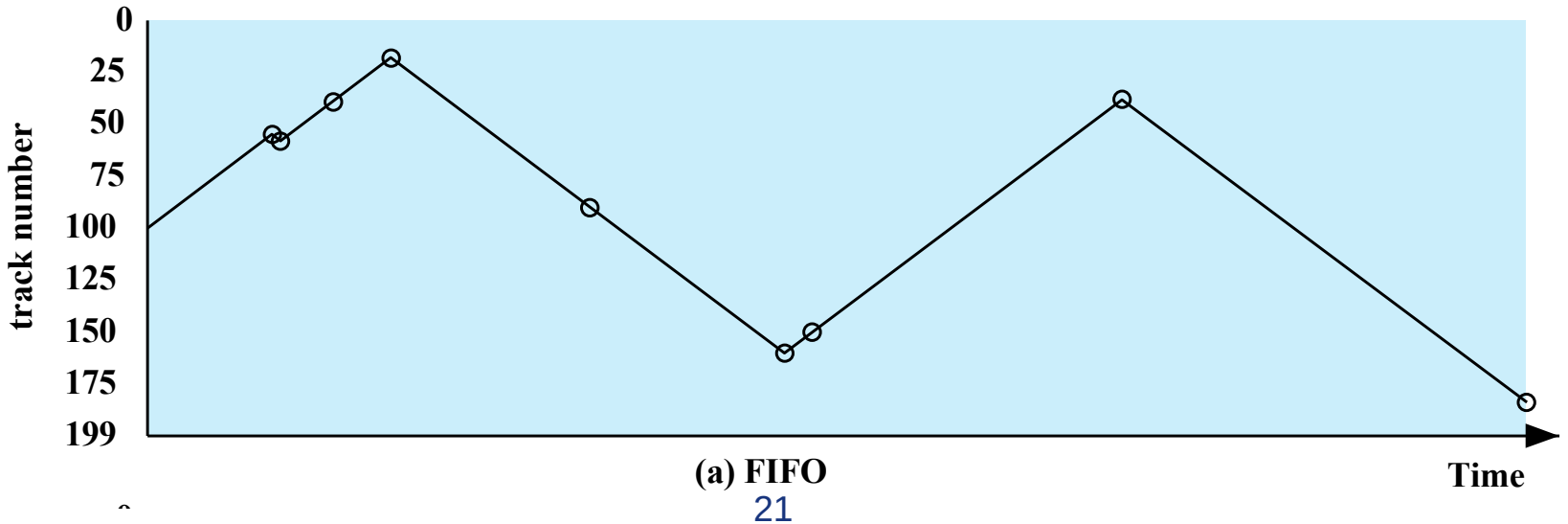
不同的磁盘调度算法

First-In, First-Out (FIFO)

磁盘磁道请求序列：

55, 58, 39, 18, 90, 160, 150, 38, 184.

- Processes in sequential order
- Fair to all processes
- Approximates random scheduling in performance if there are many processes competing for the disk



优先级（PRI）

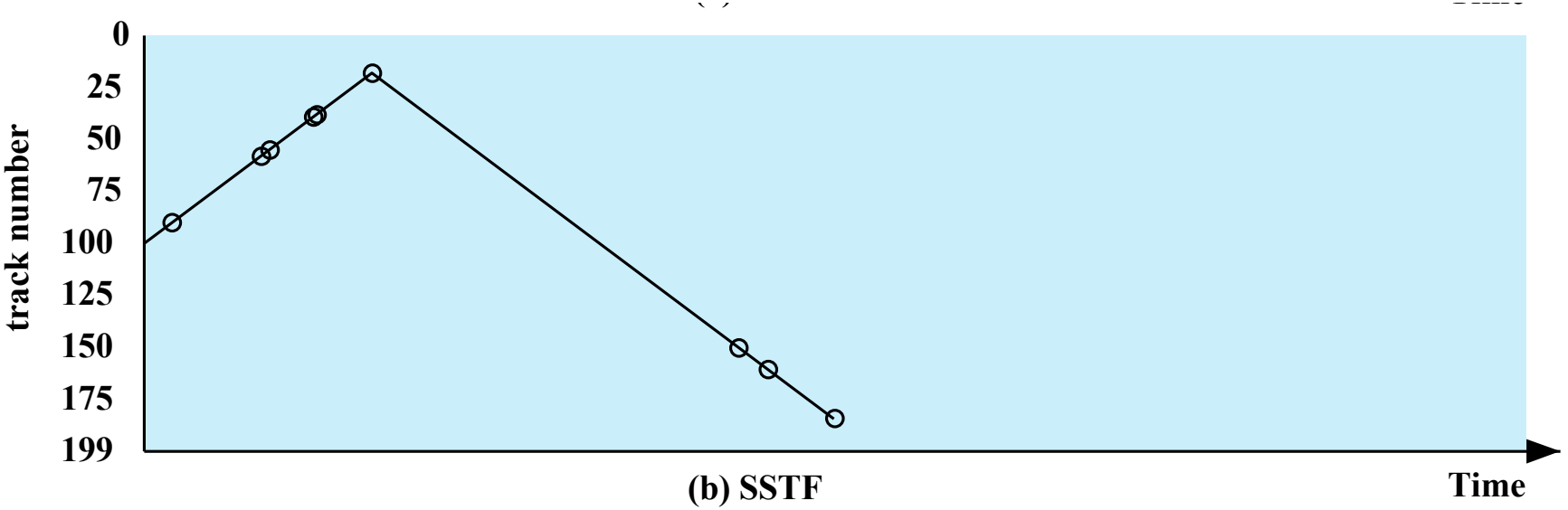
- 调度的控制不受磁盘管理软件的控制
- 目标不是优化磁盘利用率而是满足其他目标
- 短批作业和交互式作业具有更高的优先级
- 提供良好的交互响应时间
- 较长的作业可能需要等待过长的时间
- 数据库系统的糟糕策略
  - 可能导致用户将作业分成更小的部分以击败系统

最短服务时间优先(SSTF)

- 选择使磁头臂从当前位置开始移动最少的磁盘I/O请求
- 总是选择导致最小寻道时间的请求；

磁盘请求序列：

55, 58, 39, 18, 90, 160, 150, 38, 184.



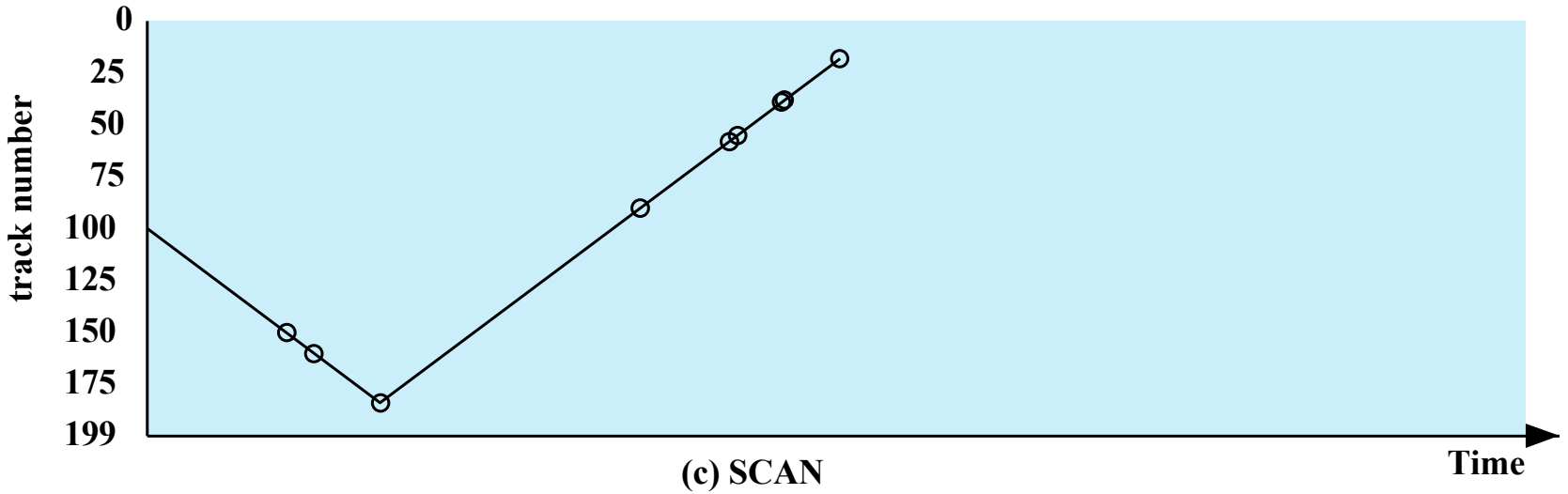
SCAN扫描算法

- **Also known as the elevator algorithm(电梯算法)**
- **Arm moves in one direction only**
  - satisfies all outstanding requests until it reaches the last track in that direction then the direction is reversed

磁盘请求序列：

55, 58, 39, 18, 90, 160, 150, 38, 184.

- Favors jobs whose requests are for tracks nearest to both innermost and outermost tracks



(c) SCAN

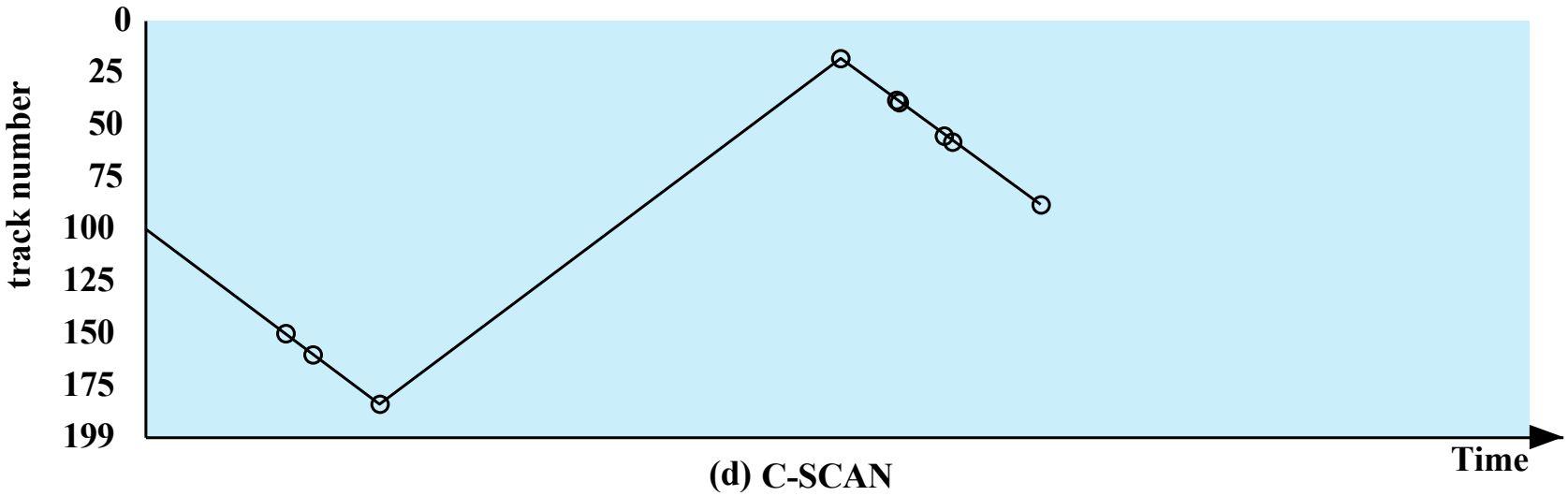
C-SCAN扫描算法

磁盘请求序列：

55, 58, 39, 18, 90, 160, 150, 38, 184

- **Restricts scanning to one direction only**
- **When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again**

改进版：LOOK和C-LOOK；



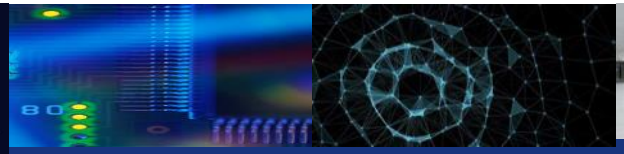
(d) C-SCAN

N步SCAN策略

- Segments the disk request queue into subqueues of length *N*
- Subqueues are processed one at a time, using SCAN
- While a queue is being processed new requests must be added to some other queue
- If fewer than *N* requests are available at the end of a scan, all of them are processed with the next scan

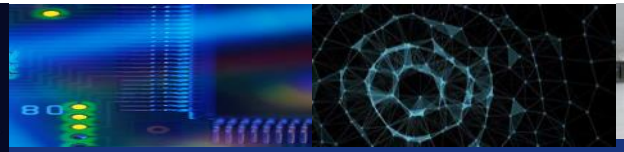
FSCAN策略

- Uses two subqueues
- When a scan begins, all of the requests are in one of the queues, with the other empty
- During scan, all new requests are put into the other queue
- Service of new requests is deferred until all of the old requests have been processed



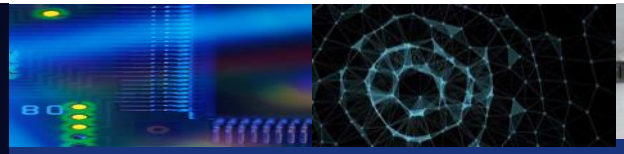
选择磁盘调度算法

- SSTF是常见的，具有天然的吸引力
- 文件分配方式可以极大影响磁盘服务的性能。
  - 如程序读取连续分配文件时，生成多个相近位置的磁盘请求，导致有限的磁头移动。相比之下，链接或索引的文件可能包括分散在磁盘不同位置多个块，导致更多的磁头移动
- 目录和索引块的位置也影响磁盘服务性能。
  - 打开文件需要搜索目录结构，所以目录会被经常访问。假设目录条目位于第一个柱面，而文件位于最后柱面。此时，磁头必须移过整个磁盘的快读。如果目录条目位于中间柱面，则磁头只需要移过不到一半的磁盘
- SCAN和C-SCAN对于磁盘负载较大的系统性能更好
  - 减少饥饿，但仍有可能出现饥饿；
- 为了避免饥饿，Linux实现了截止时间调度器。以确保每个I/O请求在一定的时间内一定要被服务到，以此来避免某个请求饥饿



NVM调度

- 没有磁头或旋转延迟，但仍有优化空间
- 在RHEL 7中，使用NOOP（无调度），但合并相邻的LBA请求
  - NVM最佳随机I/O，硬盘最佳顺序I/O;
  - 使用NVM时，每秒输入/输出操作数（IOPS）要高得多（几十万比几百）
  - 但是写放大（一次写，导致垃圾收集和多次读/写）会降低性能优势

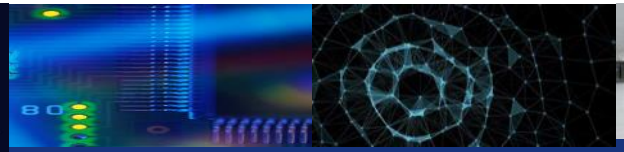


错误检测与纠正

- **许多计算部分的基本功能（内存、网络、存储）**
- **错误检测确定是否发生了问题（例如位翻转）**
  - 如果检测到，可以停止操作
  - 经常通过奇偶校验位进行检测
- **奇偶校验：一种形式的校验，使用模运算来计算、存储、比较固定长度字的值；**
  - 网络中常见的另一种错误检测方法是循环冗余校验（CRC），它使用哈希函数检测多个位错误
- **纠错码（ECC）不仅可以检测，而且可以纠正一些错误**
  - 软错误可纠正，硬错误已检测但未纠正；

磁盘格式化

- **低级格式化或物理格式化—将磁盘划分为磁盘控制器可以读写的扇区**
  - 每个扇区可以保存头信息、数据和纠错码（ECC）
  - 通常512字节的数据，但可以选择；
- **要使用磁盘保存文件，操作系统仍然需要在磁盘上记录自己的数据结构**
  - 第一步分区：将磁盘划分为一组或多组柱面集合，每组柱面都视为一个单独逻辑磁盘
  - 第二步逻辑格式化或“创建文件系统”：操作系统将初始的文件系统数据结构存储到磁盘上。这些数据结构包括空闲和已分配的空间和一个初始为空的目录
  - 为了提高效率，大多数文件系统将块分组到簇中
    - 磁盘I/O在块中完成
    - 文件I/O在簇中完成

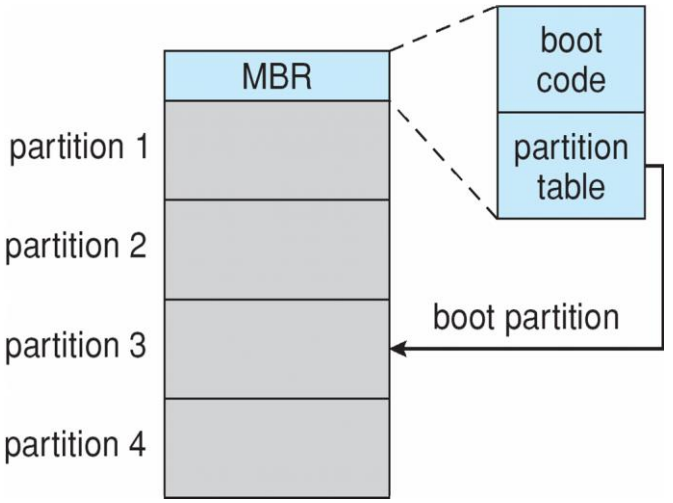


存储设备管理

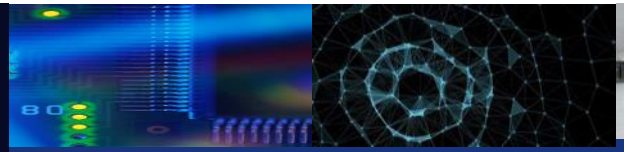
- **根分区包含操作系统，其他分区可以容纳其他操作系统、其他文件系统，也可以是原始分区**
  - 在启动时挂载
  - 其他分区可以自动或手动挂载
- **挂载时，检查文件系统一致性**
  - 所有元数据都正确吗？
    - 如果没有，修复它，然后重试；
    - 如果是，添加到挂载表，允许访问；
- **引导块可以指向包含足够代码的引导卷或引导加载程序块，以了解如何从文件系统加载内核**
  - 或用于多操作系统引导的引导管理程序

引导块

- 原始磁盘（raw disk）访问适用于希望自己进行数据管理的应用程序，操作系统不介入管理（例如数据库）
- 引导块：为了开始运行计算机，如打开电源或重启时，必须有一个程序来运行
  - 自举（bootstrap）程序存储在ROM固件中。它初始化系统的所有部分，涉及CPU寄存器、设备控制器和内存，并负责启动操作系统。
  - ROM不需要初始化，位置固定，便于执行
  - ROM是只读的，不会受到计算机病毒影响
  - 存储在引导扇区的引导加载程序(MBR)；
- 用于处理坏块的方法，如扇区备用



在Windows中从辅助存储引导

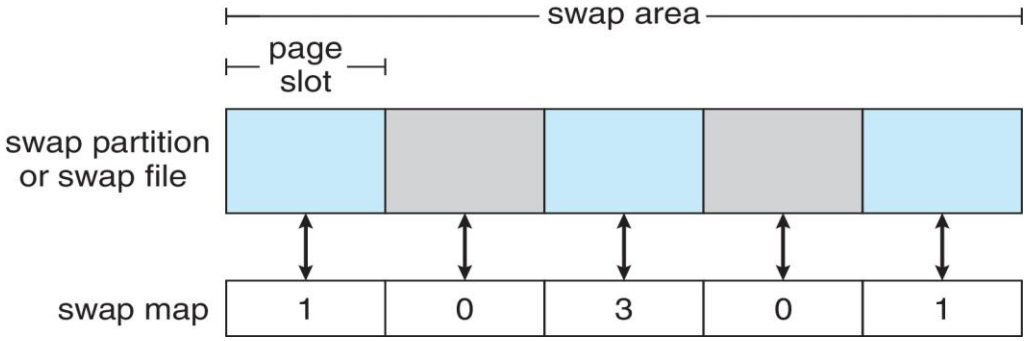


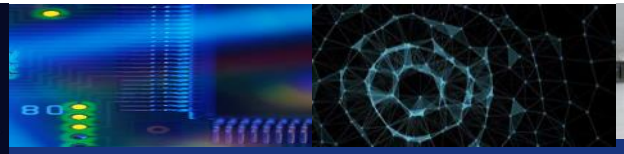
坏块

- 磁盘具有移动部件并且容错差（磁头飞行在磁盘表明上方），容易出现故障
- 有时故障是彻底的，此时需要更换磁盘，并从备份介质上将其内容恢复到新的磁盘
- 更为常见的是，一个或多个扇区坏掉。大多数磁盘出厂时就有坏块（bad block）
- 处理坏块的策略：
  - ✓ 对于采用IDE控制器的简单磁盘，可以在格式化磁盘时扫描磁盘来发现坏块，并将其标记为不可用，以便文件系统不再分配它们。如果在正常操作时出现坏块，则必须人工运行特殊程序（如Linux命令badlocks）锁定坏块。此时坏块中的数据通常会丢失
  - ✓ 扇区备用：对更复杂的磁盘，其控制器维护磁盘内的坏块列表。这个列表在出厂低级格式化时初始化，并且在磁盘使用寿命内更新。低级格式化将一些块放在一边作为备用，操作系统看不到这些块。控制器可以采用备用块来逻辑地替换坏块。
  - ✓ 扇区滑动可作为扇区备用的替代方案

交换空间管理

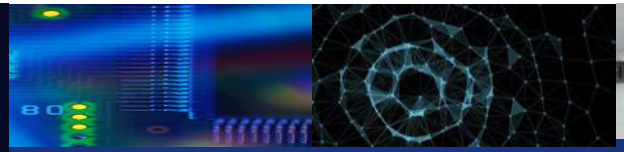
- 当物理内存不足以容纳所有进程时，用于将整个进程（交换）或页面（分页）从内存移动到辅助存储器
- 操作系统提供交换空间管理
  - 辅助存储比内存慢，因此对优化性能非常重要
  - 通常可能有多个交换空间 - 减少任何给定设备上的I/O负载
  - 最好使用专用设备
  - 可以在原始分区中，也可以在文件系统中的文件中（为了方便添加）
  - 用于在Linux系统上交换的数据结构：





交换空间管理

- 交换空间的设计和实现目标：为虚拟内存提供最佳吞吐量
- 需要考虑三个方面的问题：（1）如何使用交换空间，（2）交换在磁盘上的什么位置，（3）如何管理交换空间
- 交换空间的使用：
  - ✓ 不同系统使用方式不同，取决于内存管理算法。可以使用交换空间来保存整个进程映像，包括代码和数据段。分页系统可能只是存储换出内存的页面。系统所需交换空间的数量可以是数MB到数GB的磁盘空间
  - ✓ 交换空间数量的高估要比低估更为安全，因为当系统用完了交换空间时，可能迫使进程中止或使得整个系统死机。高估只是浪费了一些本来可以用于存储文件的空间，但没有其他的损害
  - ✓ Linux等操作系统允许使用多个交换空间，包括文件和专用交换分区。这些交换空间通常放在不同的磁盘上，确保分页和交换的I/O负荷可以分散在各个系统I/O的带宽上



交换空间管理

- 交换空间位置：它可以位于普通文件系统上，或者可以是一个单独的磁盘分区
- 如果交换空间只是文件系统内的一个大的文件，则可以采用普通文件系统程序员来创建、命名以及分配它的空间
  - ✓ 这种方式实现容易，但效率低。目录结构和磁盘分配数据结构的浏览需要时间和可能的磁盘访问。外部碎片可能由于在读写进程镜像时强制多次寻道，增加交换时间。
  - ✓ 将块位置信息缓存在物理内存中，并采用特殊工具为交换文件分配物理上连续的块等技术，可以改善性能，但是遍历文件系统数据结构的开销仍然存在。
  - ✓ 也可以在单独的原始分区上创建交换空间，此时不存放文件系统和目录的结构。管理器可以对速度而非存储效率进行优化，因为交换空间比文件系统访问更加频繁。
  - ✓ Linux允许采用原始分区和文件系统空间进行交换，更为灵活

交换空间管理

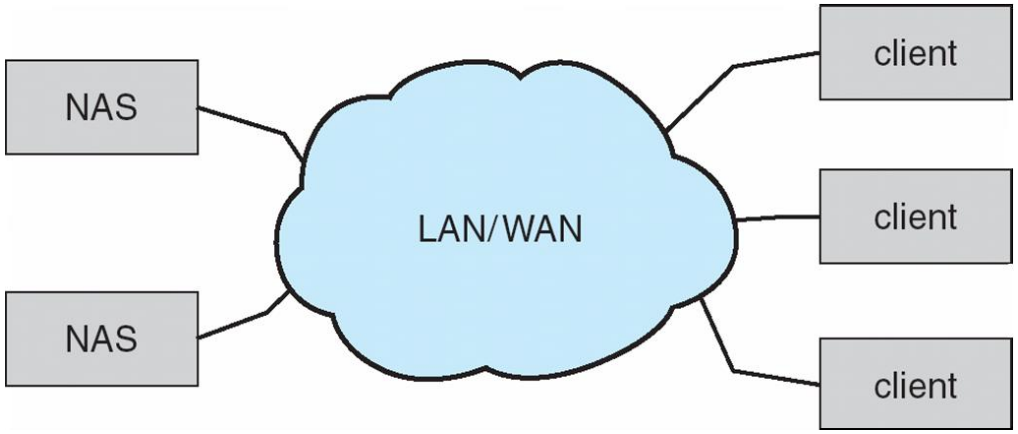
- 传统的Unix内核实现了交换，以便在连续磁盘区域和内存之间复制整个进程。随着分页硬件的出现，Unix演变为混合采用交换和分页
- Solaris1（SunOS）改进了Unix的方法：
  - ✓ 当进程执行时，包含代码和文本段页面从文件系统调入，在内存中访问，在需要换出时就会丢弃。再次从文件系统读入页面，要比先写到交换空间再从那里重读，更为高效
  - ✓ 交换空间仅仅用作备份以存储匿名内存页面，包括给栈、堆和进程未初始化数据等的分配内存
- 后续Solaris版本最大改变：只有在页面被强制换出物理内存时，而不是首次创建虚拟内存页面时，才分配交换空间。这种方式提高了现代计算机的性能，因为它比旧系统拥有更多物理内存，且换页更少

磁盘连接

- **计算机访问磁盘存储有两种方式：**
  - 通过I/O端口（或主机连接存储），适用于小系统
  - 通过分布式文件系统的远程主机，也称为网络连接存储
- **主机连接存储是通过本地I/O端口来访问的存储**
  - 这些端口使用多种技术。典型的台式PC采用I/O总线架构，如IDE或ATA。这类架构允许每条I/O总线最多支持两个驱动器
  - 高端工作站和服务器通常采用更复杂的I/O架构，例如光纤通道，它是一个高速的串行架构，运行在光纤或四芯铜线上
- **多种存储设备适合用作主机连接存储，包括硬盘驱动器、RAID阵列、CD、DVD和磁带驱动器。**
- **对主机连接存储设备进行数据传输的I/O命令是，针对特定存储单元的逻辑数据块的读和写**

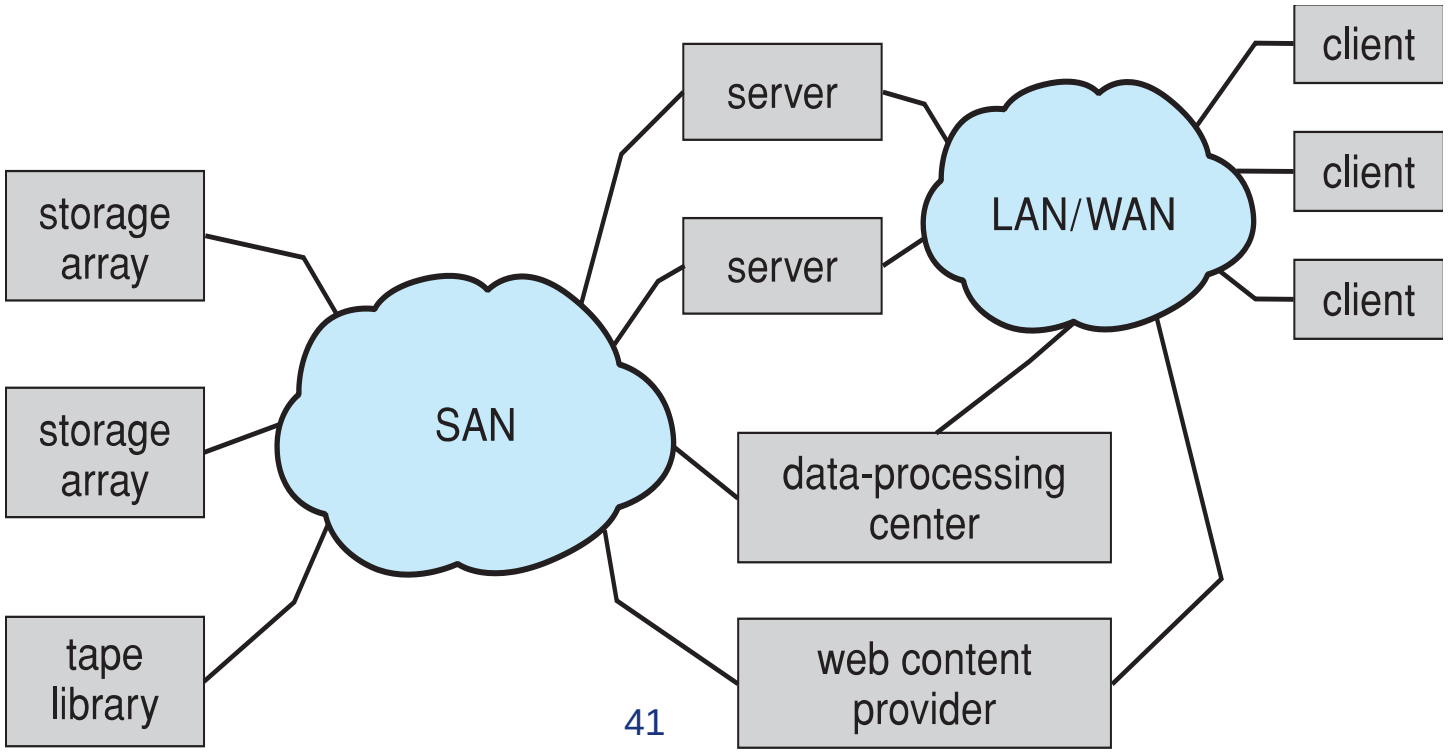
网络连接存储

- 网络连接存储（NAS）设备是一种专用存储系统，可以通过网络而不是通过本地连接（如总线）来远程访问
  - 客户通过远程过程调用（RPC），如UNIX系统的NFS或Windows的CIFS，访问网络连接存储
- 通过主机和存储之间的远程过程调用（RPC）在IP网络上通过TCP或UDP实现
- Internet小型计算机系统接口（iSCSI）是最新网络连接存储协议
  - 主机与存储之间的互联可能是网络，而不是SCSI电缆
  - 本质上采用IP网络协议



存储区域网络

- 网络连接存储的缺点：存储I/O操作消耗数据网络的带宽，从而增加网络通信的延迟
- 存储区域网络为专用网络，采用存储协议而不是网络协议连接服务器和存储单元，它在大型存储环境中常见
- 连接到多个存储阵列的多台主机－灵活



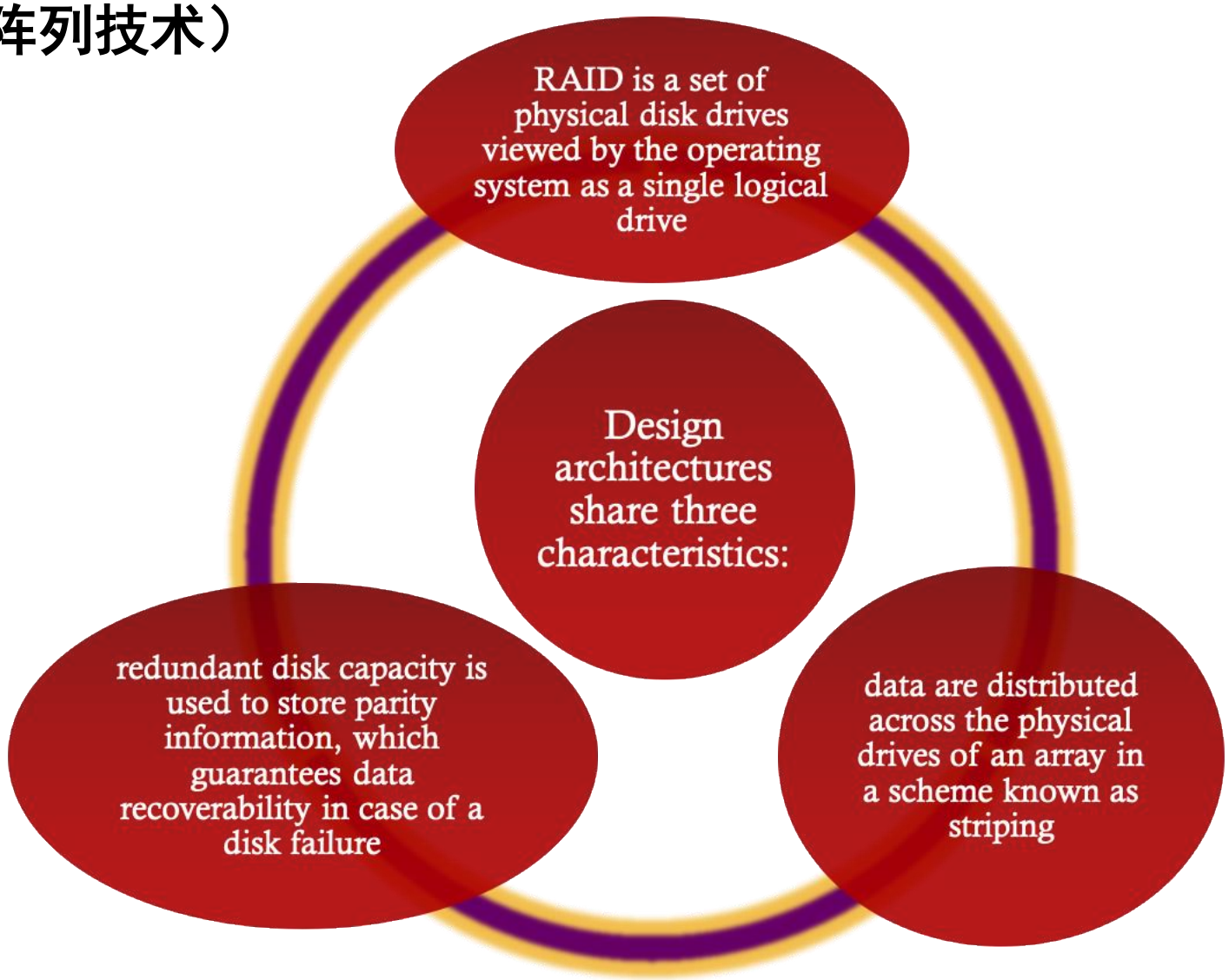
存储区域网络

- SAN是一个或多个存储阵列
  - 连接到一个或多个光纤通道交换机或InfiniBand（IB）网络
- 主机也连接到交换机
- 隐藏从特定阵列到特定服务器提供存储；
- 易于添加或删除存储、添加新主机和分配存储
- FC是最常见的SAN互连，但是iSCSI的使用正在增加

存储阵列

RAID（磁盘冗余阵列技术）

- **Redundant Array of Independent Disks**
- **Consists of seven levels, zero through six**



RAID

- 多种磁盘组织技术统称为磁盘冗余阵列技术（RAID）
- RAID技术通常用于处理性能和可靠性问题
  - ✓ 一个系统如果有大量磁盘，就有机会改善数据的读写速率，因为磁盘操作可以并行进行
  - ✓ 可以提供数据存储的可靠性，因为冗余信息可以存储在多个磁盘上。故而，单个磁盘的故障不会导致数据丢失
- 过去RAID是由小且便宜的磁盘组成，可作为大且昂贵的磁盘的有效替代品。现在RAID的使用主要是因为高可靠性和高数据传输率，而不是经济原因
- 可靠性问题的解决是引入冗余：存储额外信息。最简单的是重复每个磁盘引入镜像。或利用存储的奇偶校验信息来恢复数据
- 采用多个磁盘，可以将数据分散在多个磁盘上，来改善传输率。最简单的方式是数据分条，它将每个字节分散在多个磁盘上

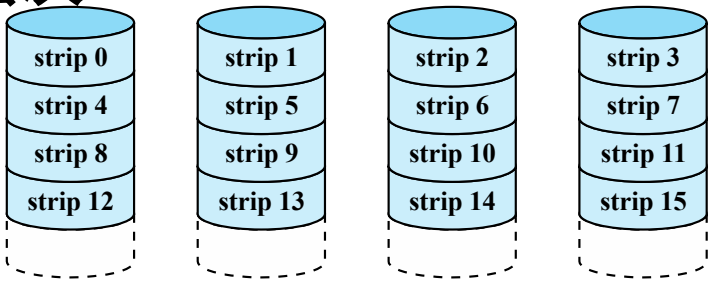
RAID层次

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk	Very high	Very high for both read and write
Mirroring	1	Mirrored	$2N$	Higher than RAID 2, 3, 4, or 5; lower than RAID 6	Higher than single disk for read; similar to single disk for write	Up to twice that of a single disk for read; similar to single disk for write
Parallel access	2	Redundant via Hamming code	$N + m$	Much higher than single disk; comparable to RAID 3, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
	3	Bit-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 4, or 5	Highest of all listed alternatives	Approximately twice that of a single disk
Independent access	4	Block-interleaved parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 5	Similar to RAID 0 for read; significantly lower than single disk for write	Similar to RAID 0 for read; significantly lower than single disk for write
	5	Block-interleaved distributed parity	$N + 1$	Much higher than single disk; comparable to RAID 2, 3, or 4	Similar to RAID 0 for read; lower than single disk for write	Similar to RAID 0 for read; generally lower than single disk for write
	6	Block-interleaved dual distributed parity	$N + 2$	Highest of all listed alternatives	Similar to RAID 0 for read; lower than RAID 5 for write	Similar to RAID 0 for read; significantly lower than RAID 5 for write

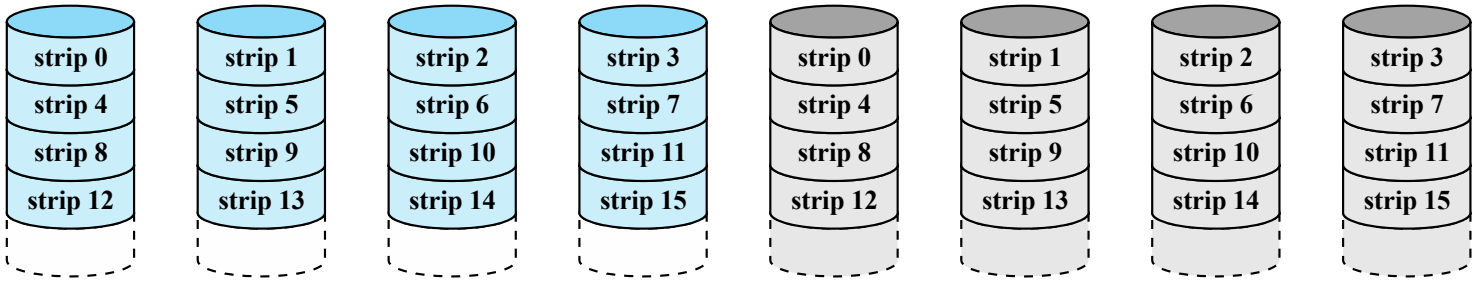
N = number of data disks; m proportional to $\log_2 N$

Table 11.4 RAID Levels

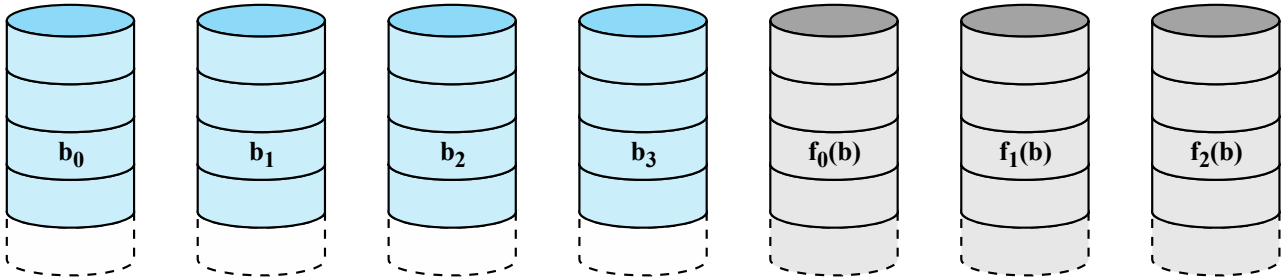
RAID层次



(a) RAID 0 (non-redundant)



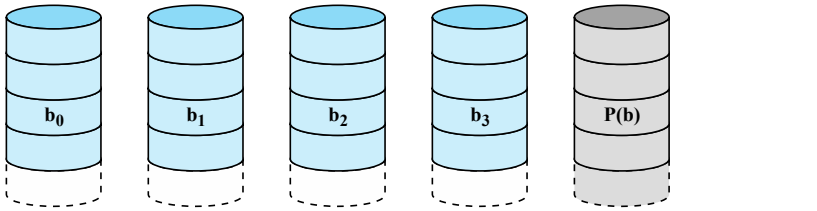
(b) RAID 1 (mirrored)



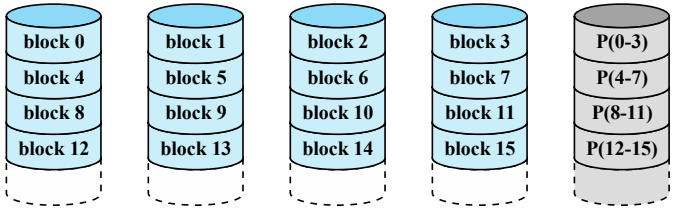
(c) RAID 2 (redundancy through Hamming code)

Figure 11.8 RAID Levels (page 1 of 2)

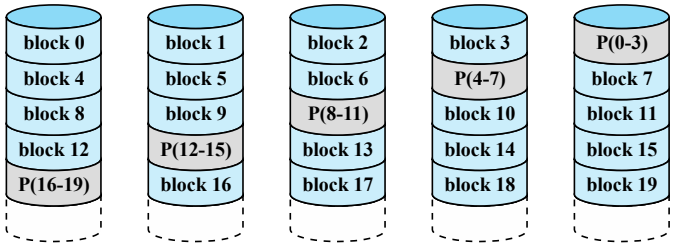
RAID层次



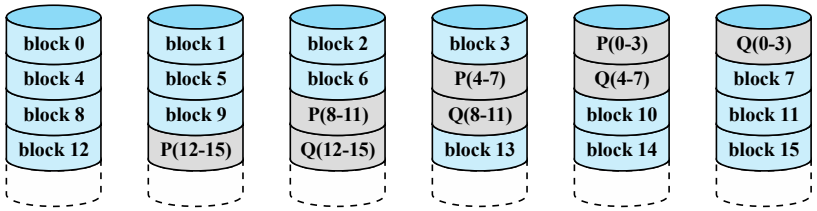
(d) RAID 3 (bit-interleaved parity)



(e) RAID 4 (block-level parity)



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

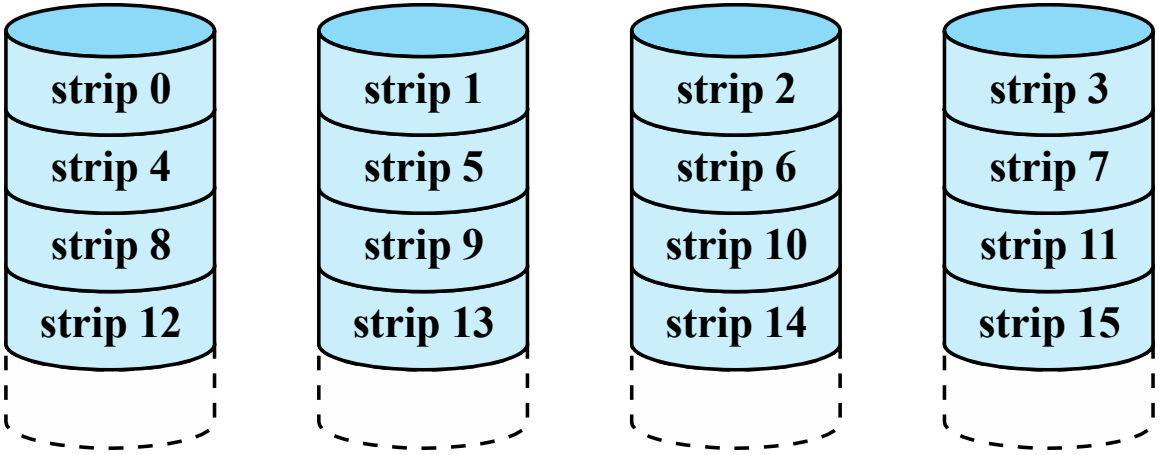


(g) RAID 6 (dual redundancy)

Figure 11.8 RAID Levels (page 2 of 2)

RAID 0

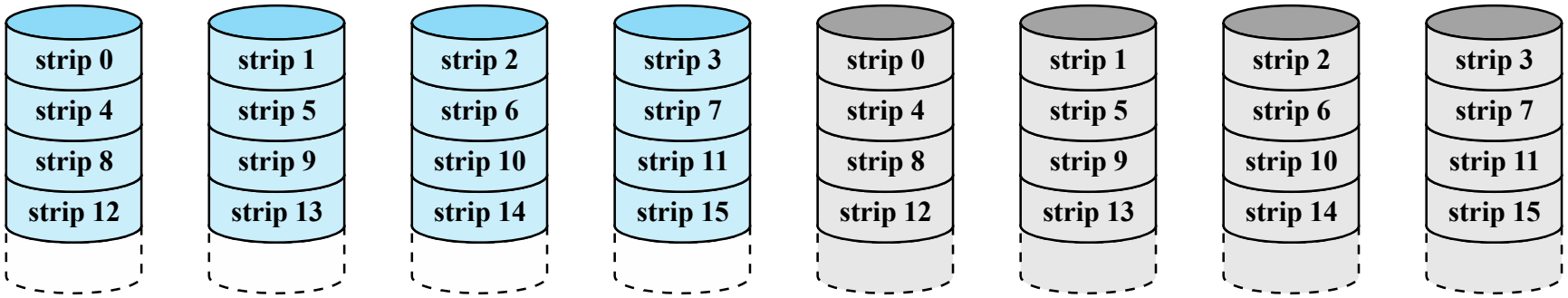
- Not a true RAID because it does not include redundancy to improve performance or provide data protection
- User and system data are distributed across all of the disks in the array
- Logical disk is divided into strips



(a) RAID 0 (non-redundant)

RAID 1

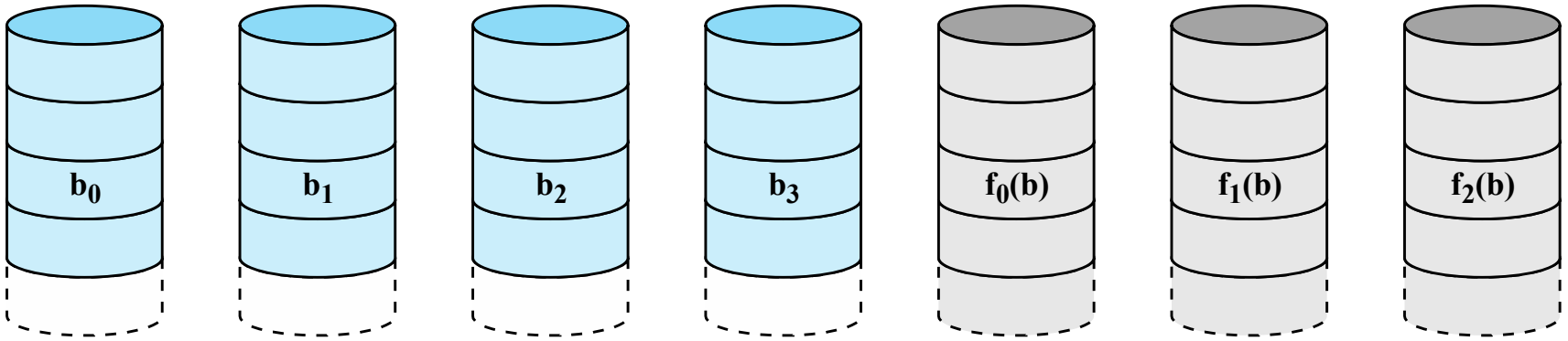
- Redundancy is achieved by duplicating all the data
- A read request can be serviced by either of the two disks that contains the requested data
- A write request requires that both corresponding strips be updated, but this can be done in parallel
- When a drive fails, the data may still be accessed from the second drive. Principal disadvantage is the cost.



(b) RAID 1 (mirrored)

RAID 2（亦称为内存方式的差错纠正组织）

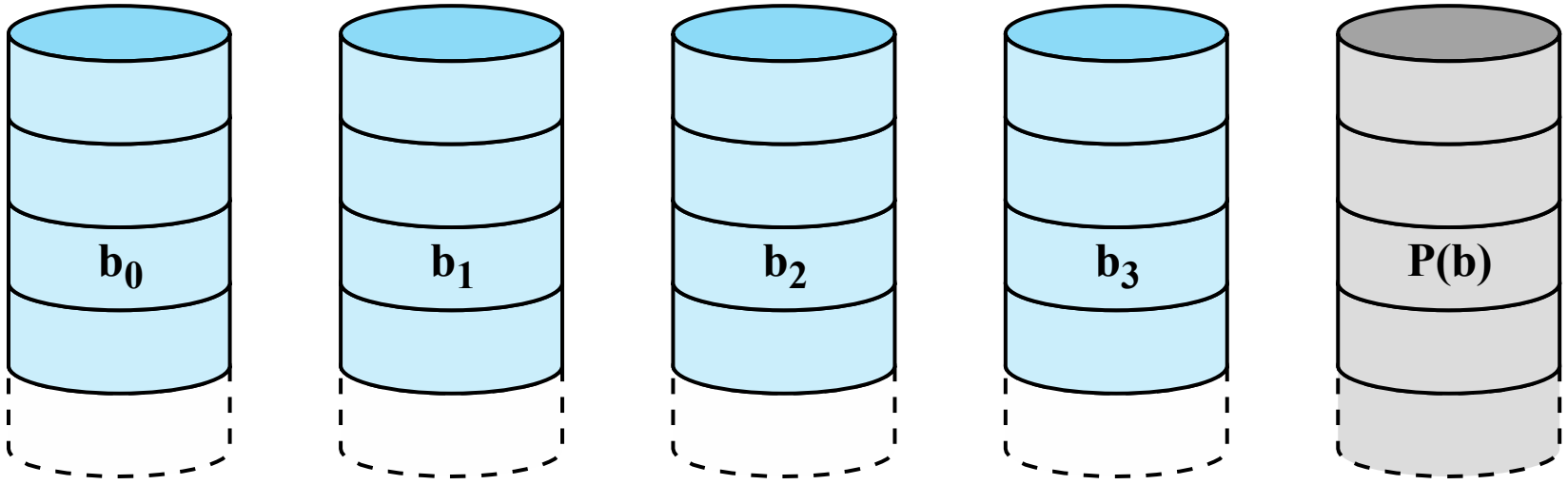
- Makes use of a parallel access technique
- Data striping is used
- Typically a Hamming code is used
- Only be effective choice in an environment in which many disk errors occur. **RAID2实际中并不使用**



(c) RAID 2 (redundancy through Hamming code)

RAID 3

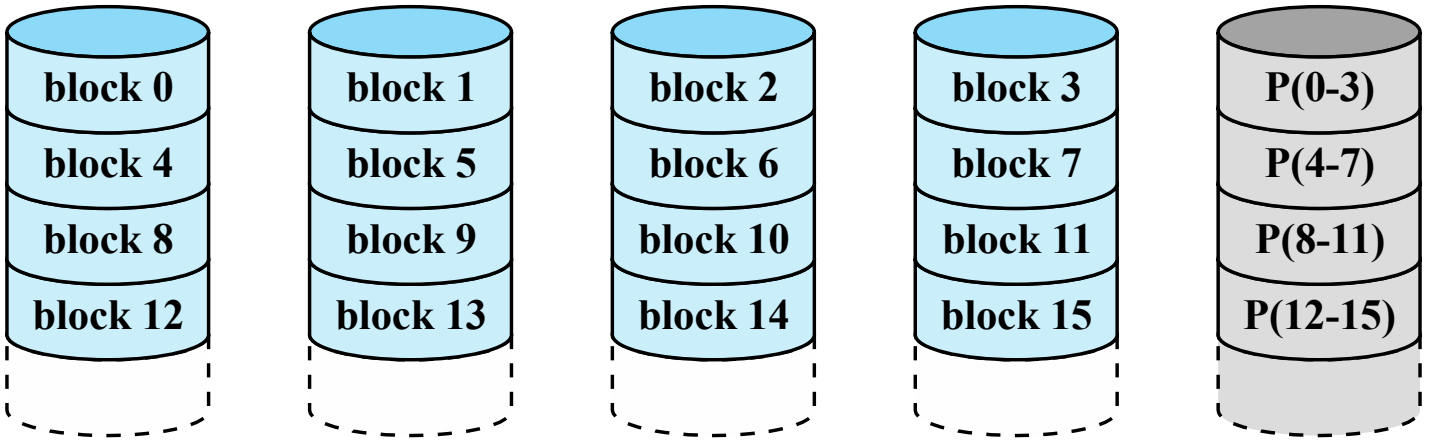
- Requires only a single redundant disk, no matter how large the disk array
- Employs parallel access, with data distributed in small strips
- Can achieve very high data transfer rates



(d) RAID 3 (bit-interleaved parity)

RAID 4

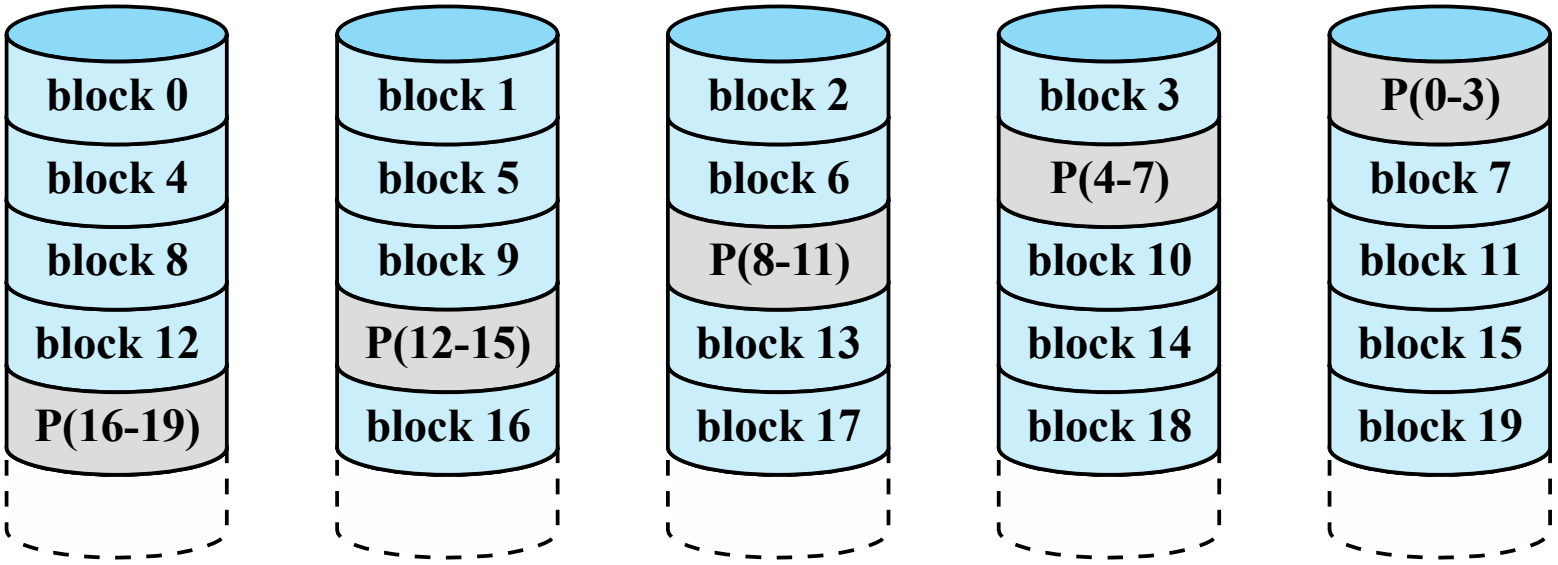
- Makes use of an independent access technique
- A bit-by-bit parity strip is calculated across corresponding strips on each data disk, and the parity bits are stored in the corresponding strip on the parity disk
- Involves a write penalty when an I/O write request of small size is performed



(e) RAID 4 (block-level parity)

RAID 5

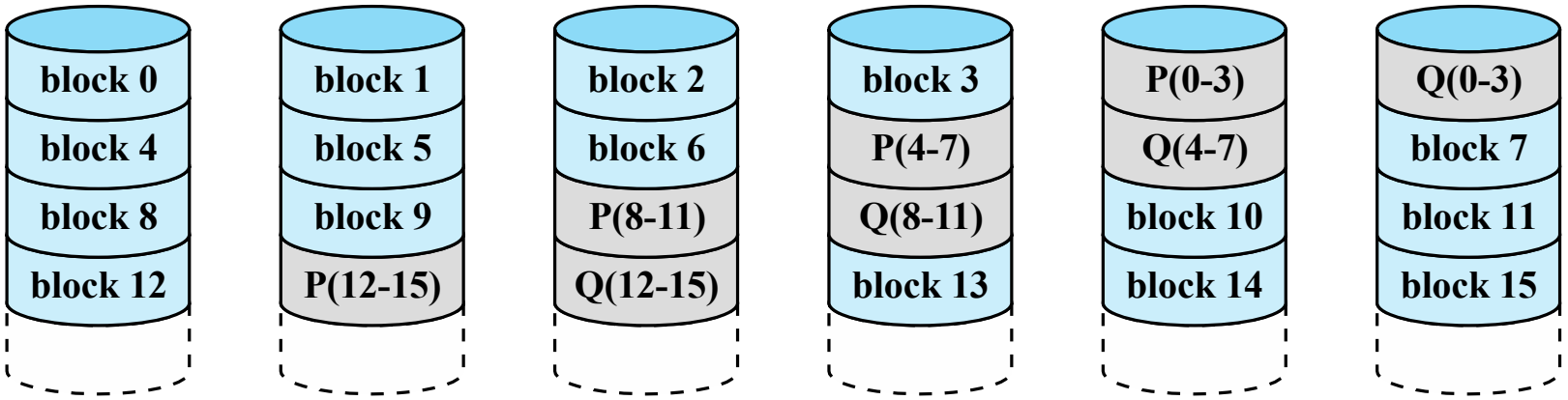
- Similar to RAID-4 but distributes the parity bits across all disks
- Typical allocation is a round-robin scheme
- Has the characteristic that the loss of any one disk does not result in data loss



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

RAID 6

- Two different parity calculations are carried out and stored in separate blocks on different disks
- Provides extremely high data availability
- Incurs a substantial write penalty because each write affects two parity blocks



(g) RAID 6 (dual redundancy)

RAID 0+1和1+0

- RAID级别0+1为RAID级别0和级别1的组合。RAID 0提供了性能，而RAID 1提供了可靠性。通常，它比RAID 5有更好的性能，适用于高性能和高可靠性的环境
  - ✓ 存储所需的磁盘数量加倍，相对昂贵
  - ✓ 结构：一组磁盘分成条，每一条镜像到另一条

- 另一种商业化的RAID选项是RAID 1+0，即磁盘先镜像，再分条。这种RAID比RAID 0+1有一些理论上的优点：
  - ✓ 如果RAID 0+1中的一个磁盘故障，那么整个条就不能访问，虽然所有其它条可用。而对于RAID 1+0，如果单个磁盘不可用，但其镜像仍然如所有其他磁盘一样可用

RAID实现

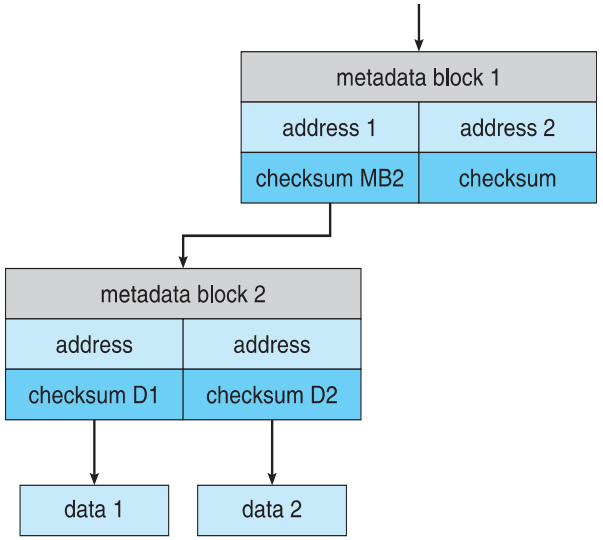
- RAID实现的层次：
  - 卷管理软件可以在内核或系统软件层中实现RAID。此时，存储硬件可以提供最少的功能，但仍是完整RAID解决方案的一部分。奇偶校验的软件实现相当慢，所以通常采用RAID 0、 RAID 1或RAID 0+1
  - RAID实现可以采用主机总线适配器硬件。只有直接连到HBA的磁盘才能成为给定RAID集的一部分。这个解决方案的成本很低，但不灵活
  - RAID实现可以采用存储阵列硬件。其中可以创建各种级别的RAID集，甚至可以将这些集合分成更小的卷，再提供给操作系统。操作系统只需要在每个卷上实现文件系统
  - RAID实现可以采用磁盘虚拟化设备的SAN互连层。在这种情况下，设备位于主机和存储之间。它接受来自服务器的命令，并管理访问存储。例如，通过将每块写到两个单独的存储设备来提供镜像
- **其它特征，如快照和复制，在每个级别中丢可以实现**
- **大多数RAID实现利用热备份**

RAID级别的选择

- RAID级别有很多，系统设计人员如何选择RAID级别？
- 一个考虑是重构性能。如果磁盘故障，则重建数据的所需时间可能很多
  - 重建性能因RAID级别不同而不同。RAID级别1的重建最简单。低于其它级别，由于需要访问阵列内的所有磁盘，以便重建故障磁盘的数据，所以需要耗费大量时间。例如，对于大磁盘集的RAID 5重建，可能需要几个小时
- RAID系统设计人员和存储管理员还必须考虑以下因素：
  - 给定RAID集应有多少磁盘—如果阵列内的磁盘越多，则数据传输率就越高，但是系统成本也越高
  - 每个奇偶校验位应保护多少位—如果奇偶校验位保护的位越多，则由于奇偶校验位而导致的空间开销就越低，但在第一个故障磁盘需要替换之前而第二个磁盘出现故障并且导致数据丢失的可能性越高

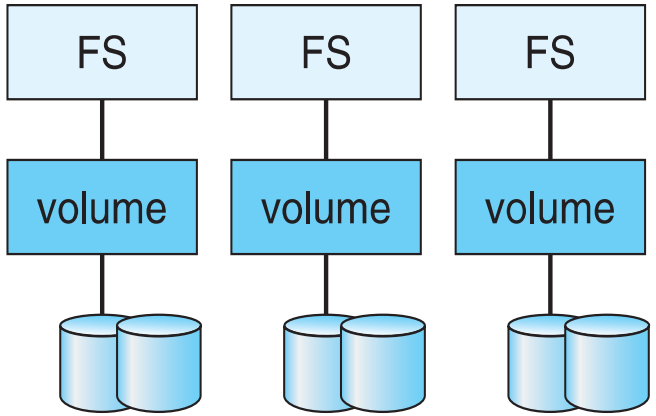
扩展

- RAID防范磁盘故障，但不是其他硬件和软件错误，系统数据潜在危险仍存在
- Solaris ZFS添加所有数据和元数据的内部校验和（checksum）
- 与指向对象的指针保持校验和，以检测对象是否正确以及是否已更改
- 可以检测并更正数据和元数据损坏
- ZFS还删除卷、分区和分区，通过RAID集注册存储池。每个池可以容纳一个或多个ZFS文件系统
  - 在池中分配的磁盘
  - 具有池的文件系统共享该池，使用和释放空间，如malloc（）和free（）内存分配/释放调用

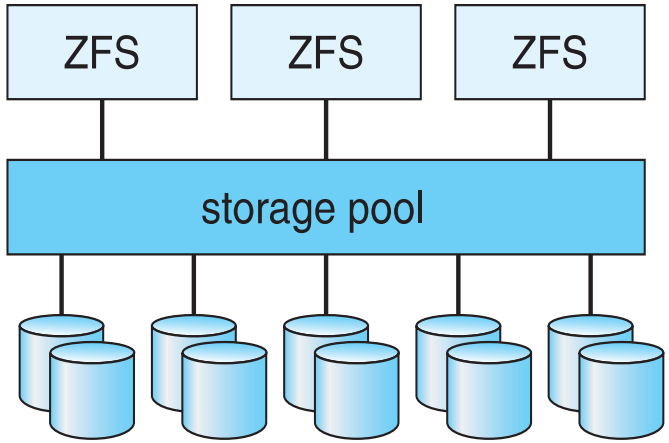


ZFS对所有元数据和数据进行校验和

扩展



(a) Traditional volumes and file systems.



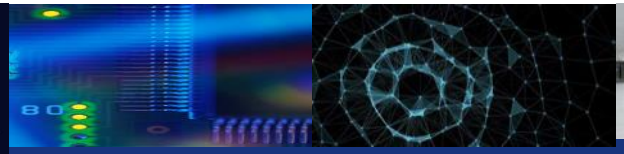
(b) ZFS and pooled storage.

稳定存储实现

- 位于稳定存储的数据永远不会丢失
- 为了实现这种存储，需要复制所需信息到多个具有独立故障模式的存储设备（通常为磁盘），且需要协调更新写入，保证更新过程中的故障不会让所有副本处于损坏状态
- 磁盘写入导致三种结果：
  - ✓ **成功完成**：数据正确写到磁盘
  - ✓ **部分故障**：在传输中出现故障，这样有些扇区写了新数据，而在故障发生时正在写的扇区可能被破坏
  - ✓ **完全故障**：在磁盘写入开始之前发生故障，因此磁盘上的以前数据值保持不变
  - ✓ 对于写入故障，系统需要检测，并调用恢复程序使得数据块恢复到一致状态。为此，系统为每个逻辑块维护两个物理块，并只有当将同样信息成功写入到第二个物理块时，才声明操作完成

小结

- 磁盘内部结构
- 固态磁盘
- 磁盘连接：主机连接、网络连接、存储区域网络
- 磁盘调度：FCFS、SSTF、SCAN、CSCAN等
- 磁盘管理：低级格式化、逻辑格式化、引导块、坏块、交换空间管理、RAID（多级）、RAID级别的选择、RAID的问题，稳定存储的实现



谢谢