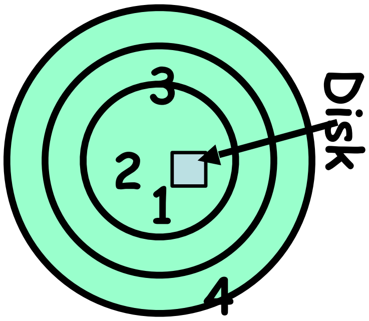
外部存储管理

背景



Where is my mp3?

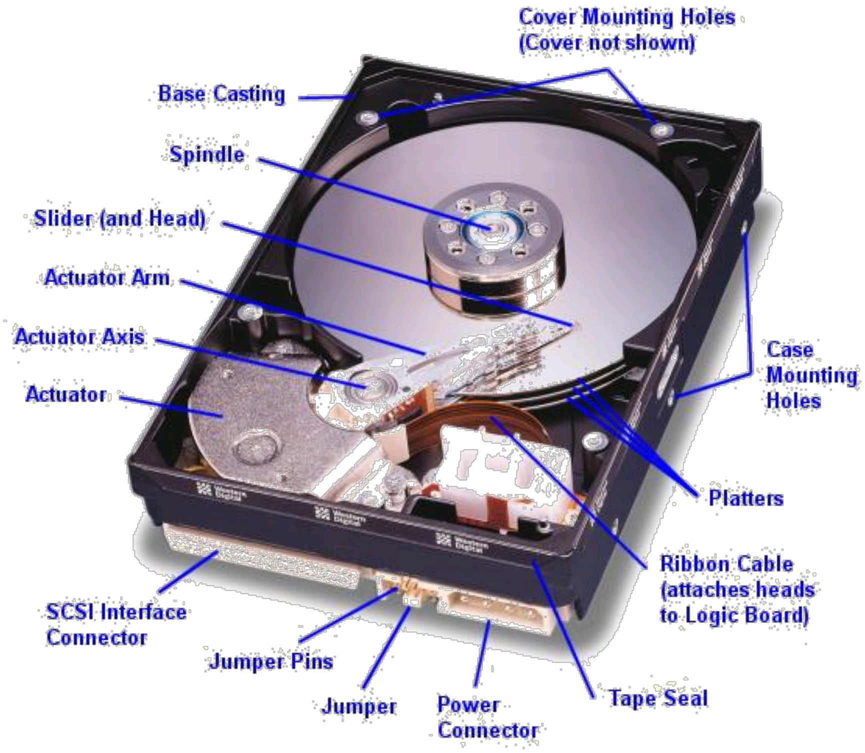
存储介质

（1）介质种类：磁盘，磁带，光盘。

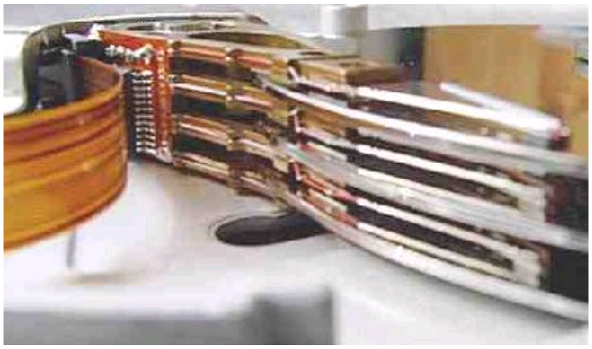
（2）物理块：在文件系统中，文件的存储设备常常划分为若干大小相等的物理块。同时也将文件信息划分成相同大小的逻辑块，所有块统一编号。以块为单位进行信息的存储、传输和分配。

（3）磁带：**永久保存大容量数据的顺序存取设备。前面的物理块被存取访问之后，才能存取后续的物理块的内容。存取速度较慢，主要用于后备存储，或存储不经常用的信息，或用于传递数据的介质。**

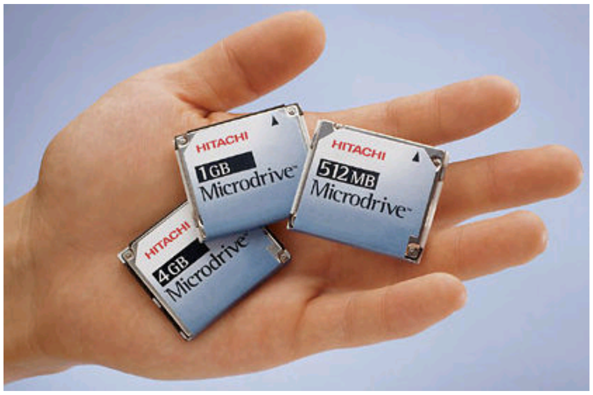
Hard Disk Drives



Western Digital Drive
http://www.storagereview.com/guide/

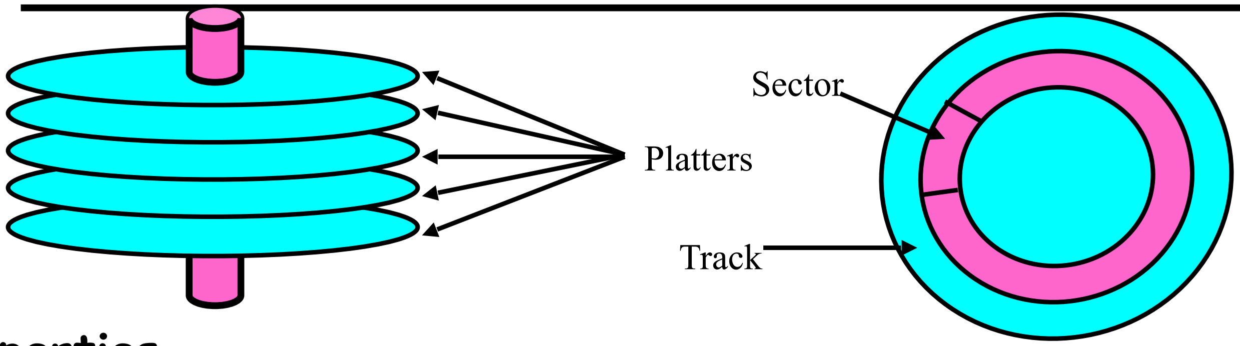


Read/Write Head
Side View

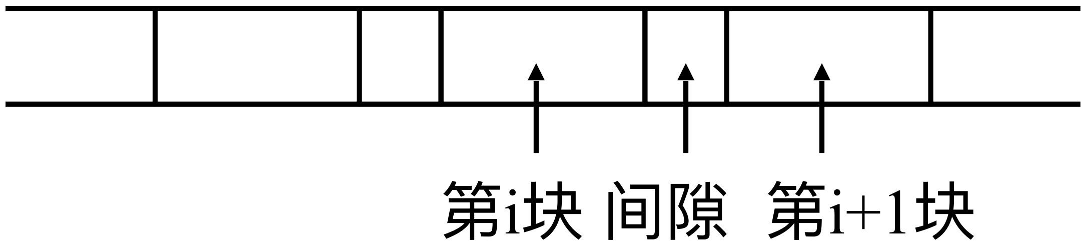


IBM/Hitachi Microdrive

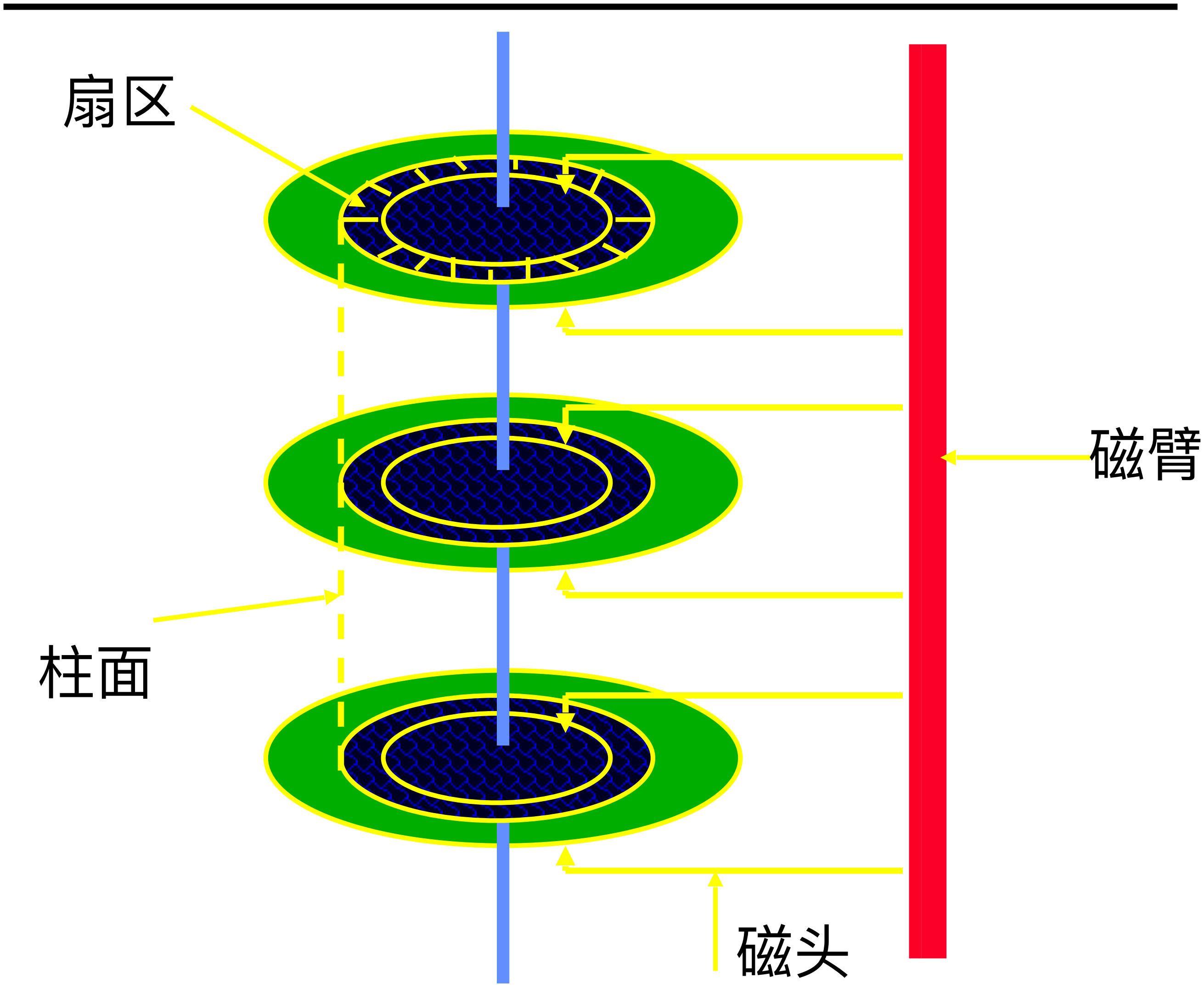
Properties of a Hard Magnetic Disk



- Properties
  - Independently addressable element: sector
    - OS always transfers groups of sectors together—"blocks"
  - A disk can access directly any given block of information it contains (random access). Can access any file either sequentially or randomly.
  - A disk can be rewritten in place: it is possible to read/modify/write a block from the disk
- Typical numbers (depending on the disk size):
  - 500 to more than 20,000 tracks per surface
  - 32 to 800 sectors per track
    - A sector is the smallest unit that can be read or written
- Zoned bit recording
  - Constant bit density: more sectors on outer tracks
  - Speed varies with track location



- **磁盘：直接（随机）存取设备，存取磁盘上任一物理块的时间不依赖于该物理块所处的位置。**信息记录在磁道上，多个盘片，正反两面都用来记录信息，每面一个磁头。所有盘面中处于同一磁道号上的所有磁道组成一个柱面。**物理地址形式：磁头号（盘面号）、磁道号（柱面号）、扇区号。磁盘系统由磁盘本身和驱动控制设备组成，实际存取读写的动作过程是由磁盘驱动控制设备按照主机要求完成的。**


扇区
磁臂
柱面
磁头

（1）一次访盘请求：读/写，磁盘地址（设备号，柱面号，磁头号，扇区号），内存地址（源/目）。

（2）完成访盘过程的三个动作：

1.寻道（时间）：磁头移动定位到指定磁道。

2.旋转延迟（时间）：等待指定扇区从磁头下旋转经过。

3.数据传输（时间）：数据在磁盘与内存之间的实际传输。

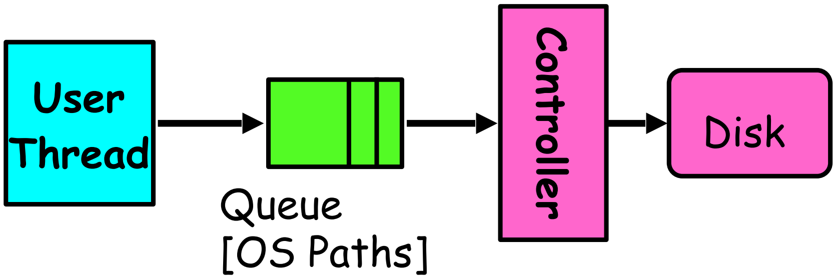
（3）很多系统允许有些磁盘是可装卸的节省驱动设备成本，增加灵活性和便携性。

（4）硬盘又分为两种：

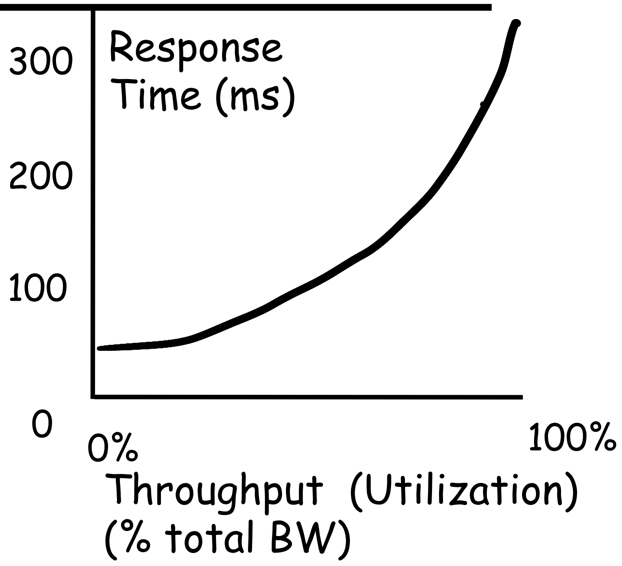
1.固定头磁盘：每个磁道设置一个磁头，变换磁道时不需要磁头的机械移动，速度快但成本高。

2.移动头磁盘：一个盘面只有一个磁头，变换磁道时需要移动磁头，速度慢但成本低。

Disk I/O Performance

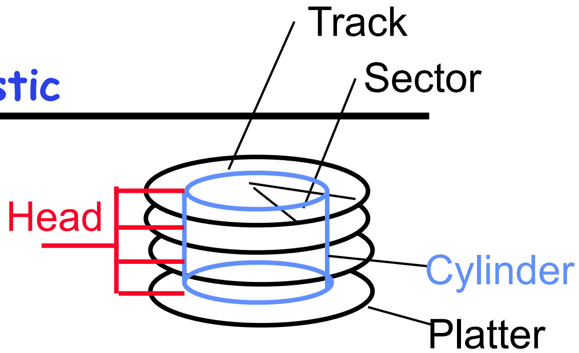


Response Time = Queue+Disk Service Time



- 硬盘/文件系统性能分析
  - **Metrics:** 响应时间，吞吐**(throughput)**
  - **Contributing factors to latency:**
    - » **Software paths (can be loosely modeled by a queue)**
    - » **Hardware controller**
    - » **Physical disk media**
  - **Queuing behavior:**
    - » **Can lead to big increases of latency as utilization approaches 100%**

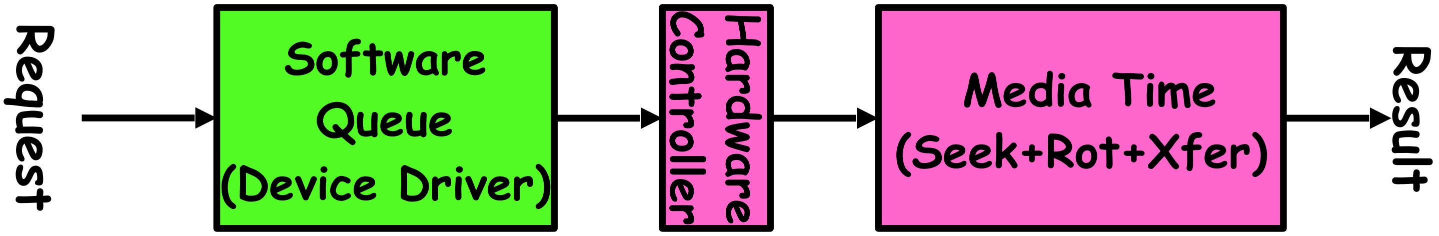
Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surface

- Read/write data is a three-stage process:
  - 寻道时间/Seek time: position the head/arm over the proper track (into proper cylinder)
  - 旋转延迟/Rotational latency: wait for the desired sector to rotate under the read/write head
  - 传输时间/Transfer time: transfer a block of bits (sector) under the read-write head

- Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



Request

Software Queue (Device Driver)

Hardware Controller

Media Time (Seek+Rot+Xfer)

Result

- Highest Bandwidth:
  - Transfer large group of blocks sequentially from one track

Typical Numbers of a Magnetic Disk

- Average seek time as reported by the industry:
  - Typically in the range of 8 ms to 12 ms
  - Due to locality of disk reference may only be 25% to 33% of the advertised number
- Rotational Latency:
  - Most disks rotate at 3,600 to 7200 RPM (Up to 15,000RPM or more)
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- Transfer Time is a function of:
  - Transfer size (usually a sector): 512B – 1KB per sector
  - Rotation speed: 3600 RPM to 15000 RPM
  - Recording density: bits per inch on a track
  - Diameter: ranges from 1 in to 5.25 in
  - Typical values: 2 to 50 MB per second
- Controller time depends on controller hardware
- Cost drops by factor of two per year (since 1991)

Disk Performance

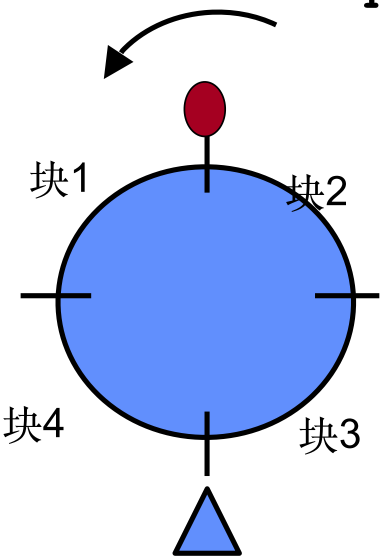
- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms, avg rotational delay of 4ms
  - Transfer rate of 4MByte/s, sector size of 1 KByte
- Random place on disk:
  - Seek (5ms) + Rot. Delay (4ms) + Transfer (0.25ms)
  - Roughly 10ms to fetch/put data: 100 KByte/sec
- Random place in same cylinder:
  - Rot. Delay (4ms) + Transfer (0.25ms)
  - Roughly 5ms to fetch/put data: 200 KByte/sec
- Next sector on same track:
  - Transfer (0.25ms): 4 MByte/sec
- Key to using disk effectively (esp. for filesystems) is to minimize seek and rotational delays

Disk Tradeoffs

- How do manufacturers choose disk sector sizes?
  - Need 100-1000 bits between each sector to allow system to measure how fast disk is spinning and to tolerate small (thermal) changes in track length
- What if sector was 1 byte?
  - Space efficiency – only 1% of disk has useful space
  - Time efficiency – each seek takes 10 ms, transfer rate of 50 – 100 Bytes/sec
- What if sector was 1 KByte?
  - Space efficiency – only 90% of disk has useful space
  - Time efficiency – transfer rate of 100 KByte/sec
- What if sector was 1 MByte?
  - Space efficiency – almost all of disk has useful space
  - Time efficiency – transfer rate of 4 MByte/sec

磁盘的访问优化

- 循环排序
  - 按照数据的分布对输入/输出请求进行排序，提高处理的效率
  - 举例
    - 假设每个磁道上保存4个记录（块），磁盘旋转速度是20ms/转，如果收到如下请求序列：读记录4、读记录3、读记录2、读记录1，则如何安排输入/输出顺序，到达理想的处理性能。
      - 按请求次序读取上述记录，总的处理时间：(1/2+1/4+3*3/4) * 20 = 60 (ms)
      - 按读取记录1，2，3，4的顺序，则总的处理时间为：(3/4 + ¼ + 3* ¼) * 20 = 35 (ms)
      - 如果知道当前读位置为记录3，则按读取记录4,1,2,3顺序，则总的处理时间为：(4 * ¼) * 20 = 20 (ms)



磁盘访问优化

- 优化分布
  - 按照数据处理的规律，合理安排其磁盘上的分布，以提高处理的效率
  - 举例
    - 假设每个磁道上划分为10个块，分别存放A~J十个逻辑记录，磁盘旋转速度是20ms/转。如果处理程序读出每个记录后花4ms进行处理，则如何安排逻辑记录的存放位置，以达到理想的处理性能？

1	A
2	H
3	E
4	B

2*2 = 4ms

磁盘访问优化

- 交替地址
  - 通过数据的冗余存放来提高访问的速度
  - 缺点：
    - 消耗较多的存储空间
    - 数据一致性问题决定其较适合于数据记录总是读出使用的方式

Disk Management with Speedup Designs

Disks vs. Memory

• Smallest write: sector	• (usually) bytes
• Atomic write = sector	• byte, word
• Random access: 5ms – not on a good curve	• 50 ns – faster all the time
• Sequential access: 200MB/s	• 200-1000MB/s
• Cost $.002MB	• $.10MB
• Crash: doesn’t matter (“non-volatile”)	• contents gone (“volatile”)

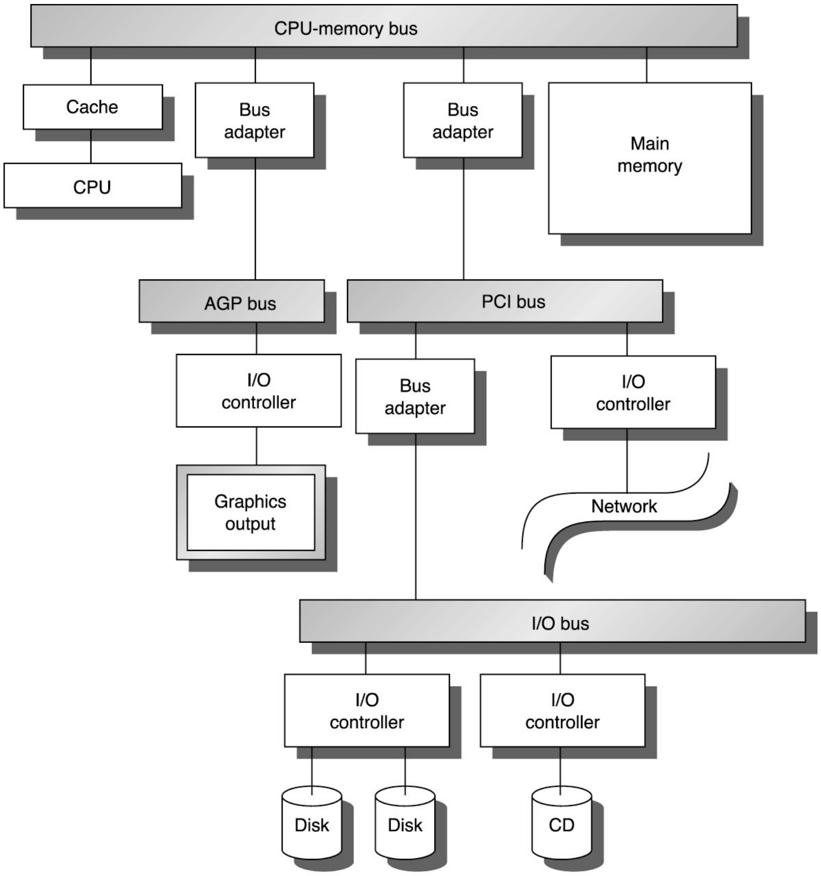
Using RAM for Storage

- Disks are about 100 times cheaper ($/MB)
- DRAM is about 100,000 faster (latency)
- Solid-State Disks
  - Actually, a DRAM and a battery
    - Much faster than disk, more reliable
    - Expensive (not very good for archives and such)
- Flash memory
  - Much faster than disks, but slower than DRAM
  - Very low power consumption
  - Can be sold in small sizes (few GB, but tiny)

Busses for I/O

- Traditionally, two kinds of busses
  - CPU-Memory bus (fast, short)
  - I/O bus (can be slower and longer)
- Now: PCI
  - Pretty fast and relatively short
  - Can connect fast devices directly
  - Can connect to longer, slower I/O busses
- Data transfers over a bus: transactions

Buses in a System



Bus Design Decisions

- Split transactions
  - Traditionally, bus stays occupied between request and response on a read
  - Now, get bus, send request, free bus (when response ready, get bus, send response, free us)
- Bus mastering
  - Which devices can initiate transfers on the bus
  - CPU can always be the master
  - But we can also allow other devices to be masters
  - With multiple masters, need arbitration

CPU-Device Interface

- Devices typically accessible to CPU through control and data registers
- These registers can be either
  - Memory mapped
    - Some physical memory addresses actually map to I/O device registers
    - Read/write through LS/ST
    - Most RISC processors support only this kind of I/O mapping
  - Be in a separate I/O address space
    - Read/write through special IN/OUT instrs
    - Used in x86, but even in x86 PCs some I/O is memory mapped

CPU-Device Interface

- Devices can be very slow
  - When given some data, a device may take a long time to become ready to receive more
  - Usually we have a Done bit in status register
- Checking the Done bit
  - Polling: test the Done bit in a loop
  - Interrupt: interrupt CPU when Done bit becomes 1
  - Interrupts if I/O events infrequent or if device is slow
    - Each interrupt has some OS and HW overhead
  - Polling better for devices that are done quickly
    - Even then, buffering data in the device lets us use interrupts
  - Interrupt-driven I/O used today in most systems

Dependability

- Quality of delivered service that justifies us relying on the system to provide that service
  - Delivered service is the *actual behavior*
  - Each module has an ideal *specified behavior*
- Faults, Errors, Failures
  - Failure: actual deviates from specified behavior
  - Error: defect that results in failure
  - Fault: cause of error

Reliability and Availability

- Reliability
  - Measure of continuous service accomplishment
  - Typically, Mean Time To Failure (MTTF)
- Availability
  - Service accomplishment as a fraction of overall time
  - Also looks at Mean Time To Repair (MTTR)
    - MTTR is the average duration of service interruption
  - Availability=MTTF/(MTTF+MTTR)

Faults Classified by Cause

- Hardware Faults
  - Hardware devices fail to perform as designed
- Design Faults
  - Faults in software and some faults in HW
  - E.g. the Pentium FDIV bug was a design fault
- Operation Faults
  - Operator and user mistakes
- Environmental Faults
  - Fire, power failure, sabotage, etc.

Faults Classified by Duration

- Transient Faults
  - Last for a limited time and are not recurring
  - An alpha particle can flip a bit in memory but usually does not damage the memory HW
- Intermittent Faults
  - Last for a limited time but are recurring
  - E.g. overclocked system works fine for a while, but then crashes… then we reboot it and it does it again
- Permanent Faults
  - Do not get corrected when time passes
  - E.g. the processor has a large round hole in it because we wanted to see what's inside…

Improving Reliability

- Fault Avoidance
  - Prevent occurrence of faults by construction
- Fault Tolerance
  - Prevent faults from becoming failures
  - Typically done through redundancy
- Error Removal
  - Removing latent errors by verification
- Error Forecasting
  - Estimate presence, creation, and consequences of errors

独立磁盘冗余阵列(RAID)

- 基本思路：
  - 用一组较小容量的、独立的、可并行工作的磁盘驱动器组成阵列来代替单一的大容量磁盘，并加入冗余技术，数据能够以多种方式组织和分布存储。
- 优点：
  - 数据的分布存储，提高了单个**I/O**请求的处理性能
  - 数据的冗余，提高了系统的可靠性

Disk Fault Tolerance with RAID

- Redundant Array of Inexpensive Disks
  - Several smaller disks play a role of one big disk
- Can improve performance
  - Data spread among multiple disks
  - Accesses to different disks go in parallel
- Can improve reliability
  - Data can be kept with some redundancy

RAID的特性：

1.RAID是一组物理磁盘驱动器，可被操作系统看作是单一逻辑磁盘驱动器;

2.数据被分布存储在阵列横跨的物理驱动器上;

3.冗余磁盘的作用是保存奇偶校验信息，当磁盘出现失误时它能确保数据的恢复。

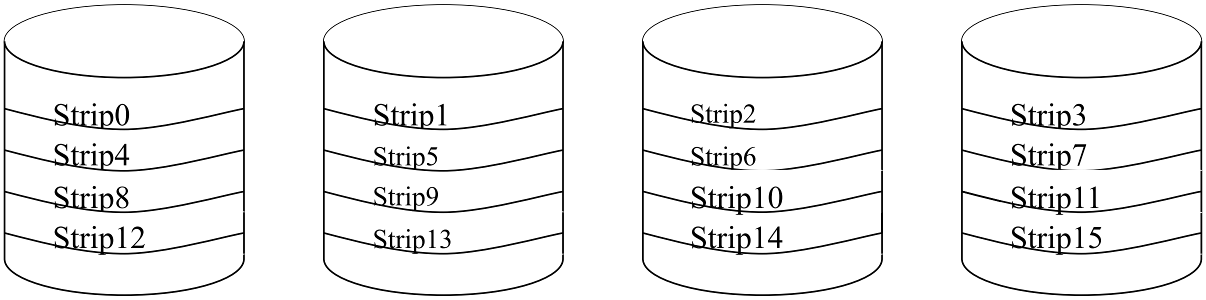
RAID level 0：

1.数据划成条块被分布存储在横跨阵列中的所有磁盘上;

2.逻辑上连续的数据条块，在物理上可被依次存储在横向相邻的磁盘驱动器上;

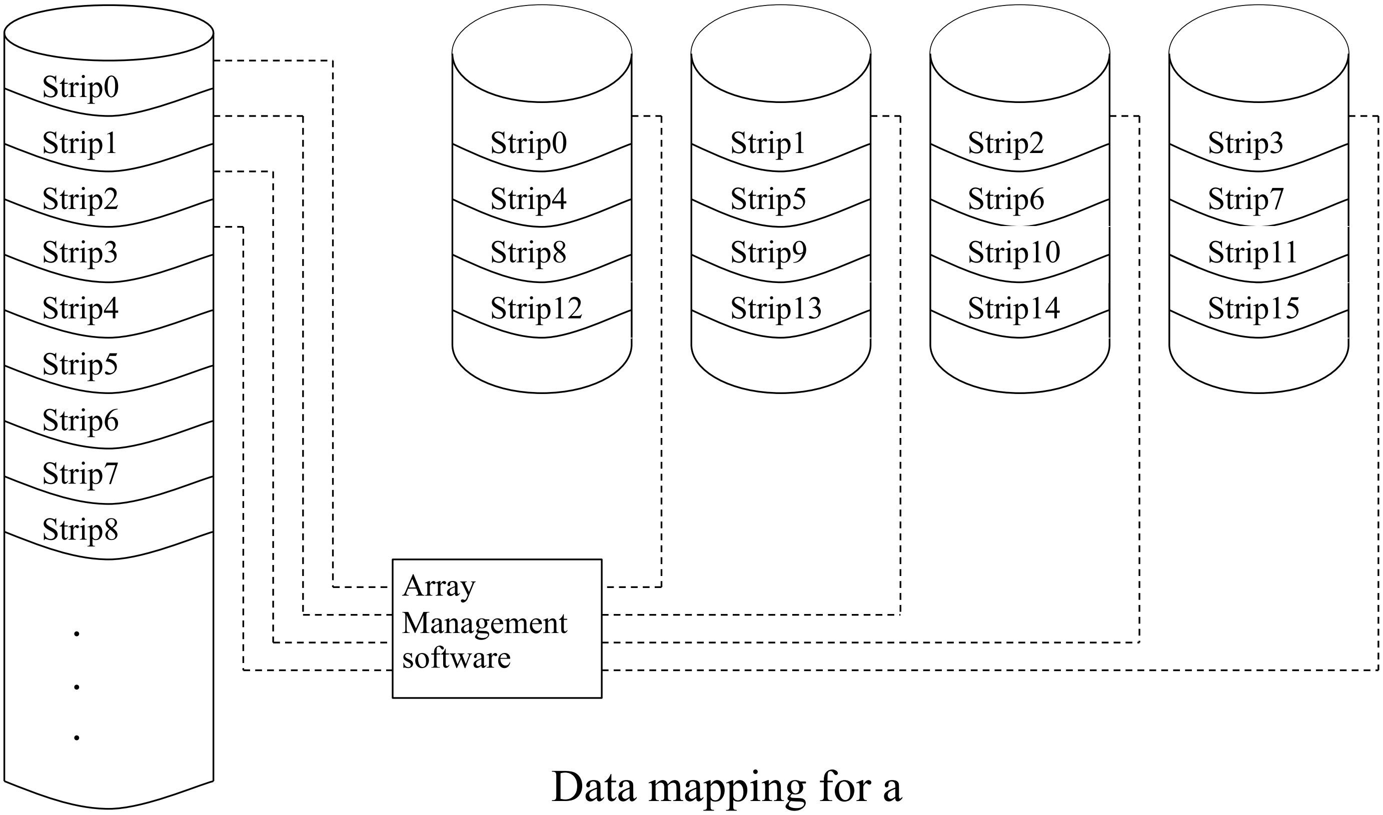
3.通过阵列管理软件进行逻辑地址空间到物理地址空间的映射。

RAID Level 0



RAID 0

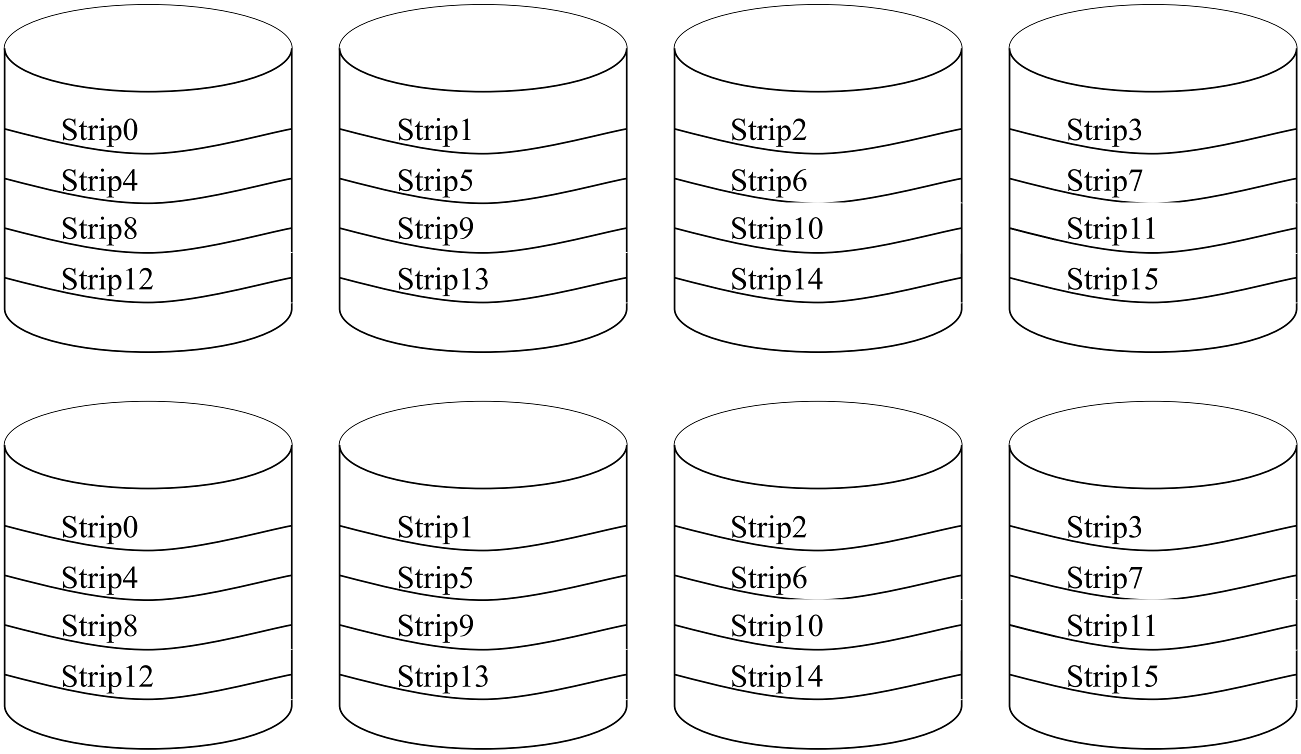
- Striping used to improve performance
  - Data stored on disks in array so that consecutive “stripes” of data are stored on different disks
  - Makes disks share the load, improving
    - Throughput: all disks can work in parallel
    - Latency: less queuing delay – a queue for each disk
- No Redundancy
  - Reliability actually lower than with single disk
    (if *any* disk in array fails, we have a problem)



Data mapping for a
RAID Level 0 Array

RAID 1

- Disk mirroring
  - Disks paired up, keep identical data
  - A write must update copies on both disks
  - A read can read any of the two copies
- Improved performance and reliability
  - Can do more reads per unit time
  - If one disk fails, its mirror still has the data
- If we have more than 2 disks (e.g. 8 disks)
  - "Striped mirrors" (RAID 1+0)
    - Pair disks for mirroring, striping across the 4 pairs
  - "Mirrored stripes" (RAID 0+1)
    - Do striping using 4 disks, then mirror that using the other 4



RAID Level 1 (Mirrored)

RAID level 1：双份数据，每个盘都有一个包含相同数据的镜像盘。

1.读请求能通过包含相同请求数据中的任何一个磁盘提供服务，其中的一个所化查找和搜索时间最少;

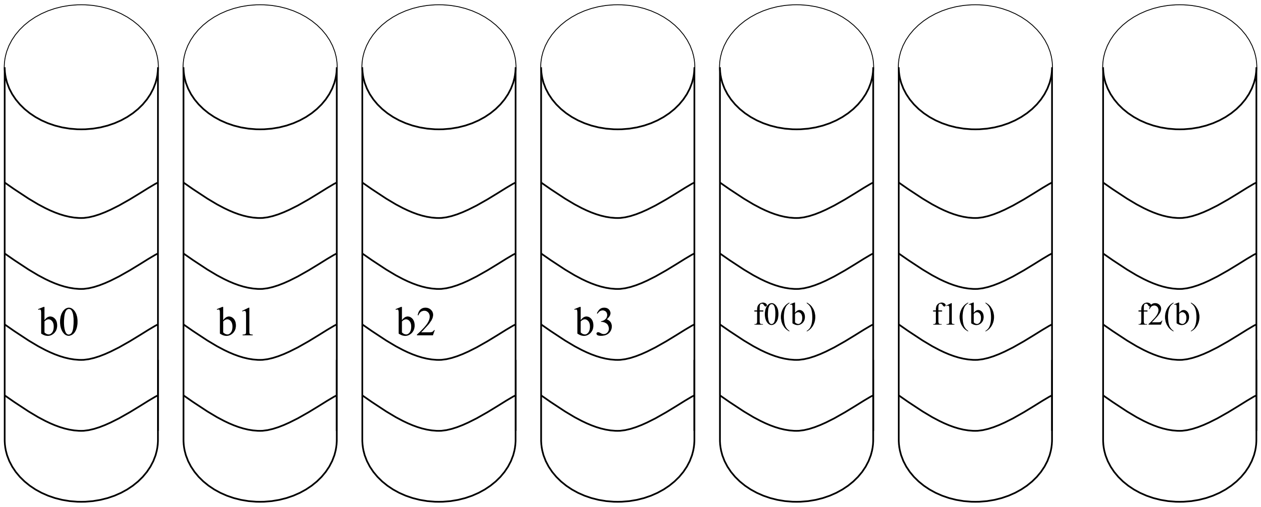
2.写操作时，要求改写对应的两个数据子块，可采用并行操作，写操作的性能由并行操作中较慢的一个决定;

3.一个驱动器出现故障，数据可以从镜像盘获得。

RAID level 2 ：

1.采用并行存取技术，驱动器的移动臂同步工作，每个磁盘的磁头都在相同位置。纠错码按照横跨的每个数据盘的相应位计算，并存储在多只校验盘的相应位的位置。

2.校验磁盘的数量与数据盘的多少成比例。



RAID Level 2 (Redundancy through Hamming Code)

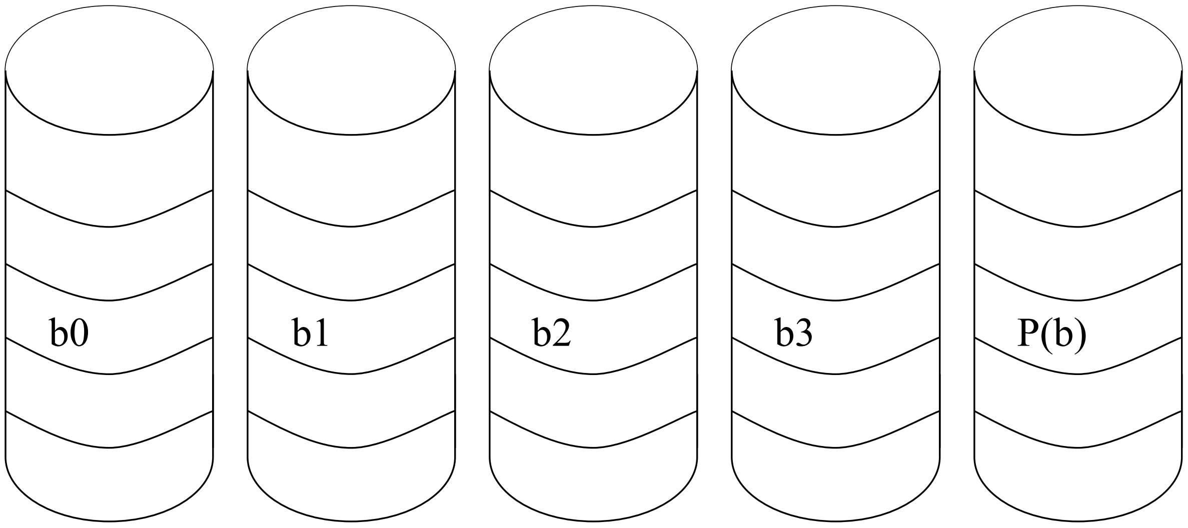
RAID level 3：

（1）RAID3仅使用一只冗余盘， 出现故障时，使用奇偶校验盘的信息校验，数据可用剩下的磁盘的信息重新构造，若X0到X3存放数据，X4为奇偶盘，对于第i位的奇偶校验位可如下计算:

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

假定驱动器X1出故障，如果把X4(i)，X1(i)加到上面等式两边，得到

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



RAID Level 3 (Bit interleaved Parity)

RAID level 4：

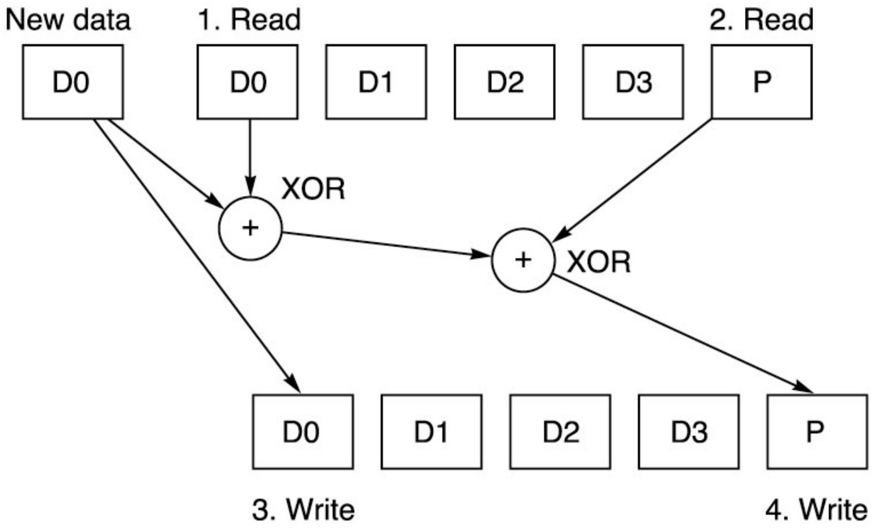
1.RAID4和RAID5使用独立存取技术，在一个独立存取的磁盘阵列中，每个驱动器都可以独立地工作，所以，独立的I/O请求可以被并行地得到满足。因此独立存取阵列适合于有频繁I/O请求的应用。

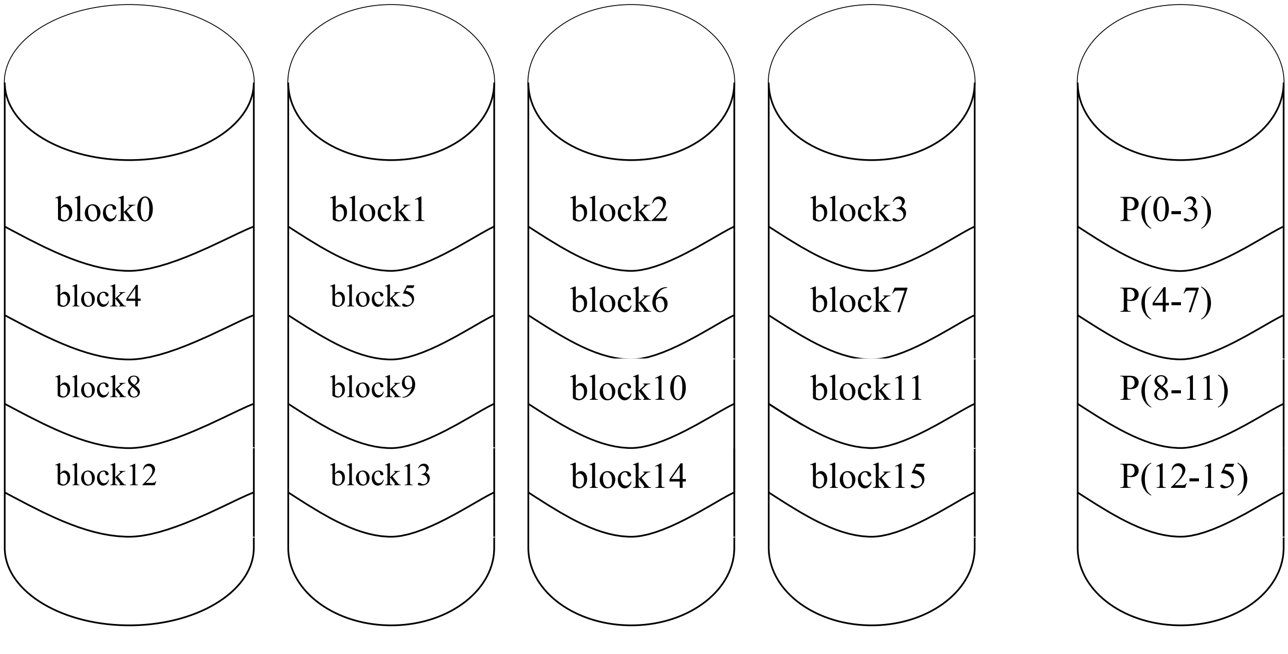
2.每当执行一个小数据量写操作时，阵列管理软件不旦要修改用户数据，而且也要修改对应的奇偶校验位。

RAID 4

- Block-interleaved parity
  - A read accesses only the data disk where the data is
  - A write must update the data block and its parity block
  - Can recover from an error on any one disk
    - Use parity and other data disks to restore lost data

RAID 4 Parity Update





RAID Level 4 (Block level Parity)

RAID level 5：

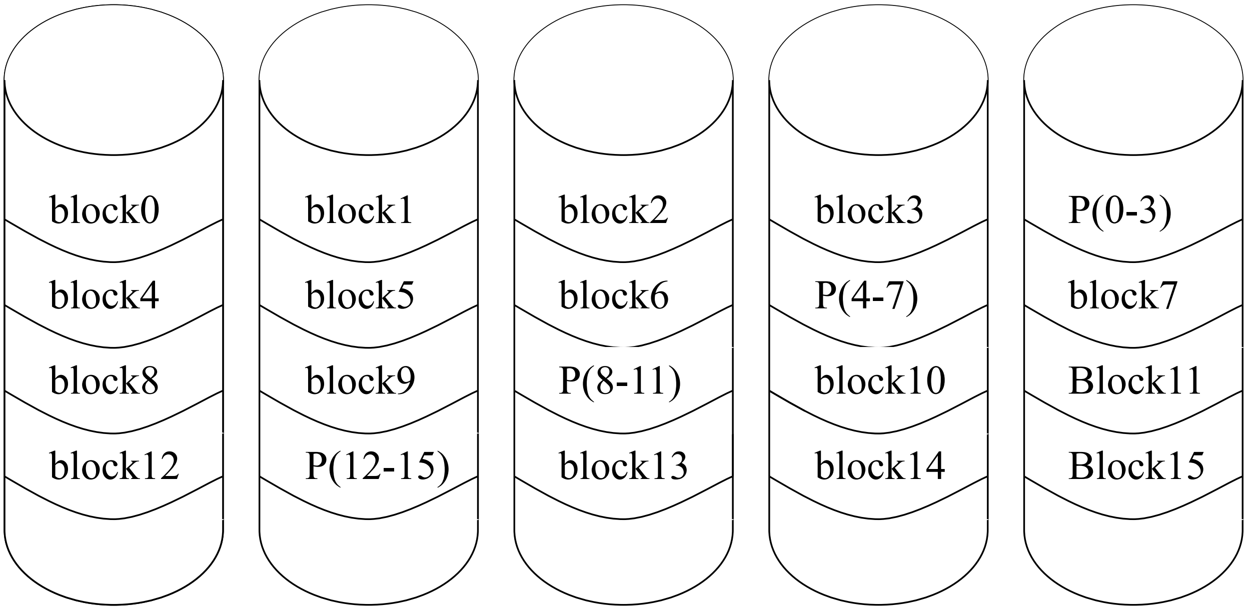
1.RAID5的奇偶校验码是分布横跨轮转存放在所有的磁盘上，设有n个磁盘的阵列，则开头的n个奇偶校验码螺旋式地位于n个磁盘上，能避免RAID4发生的奇偶校验盘瓶颈口问题。

RAID level 6和RAID level 7：

1.增强型RAID。RAID6中设置了专用快速的异步校验磁盘，具有独立的数据访问通路，比低级RAID性能更好，但价格昂贵。

2.RAID7对RAID6作了改进，该阵列中的所有磁盘都有较高传输速率，性能优异，但价格也很高。

RAID Level 5 (Block level Distributed parity)



block0	block1	block2	block3	P(0-3)
block4	block5	block6	P(4-7)	block7
block8	block9	P(8-11)	block10	Block11
block12	P(12-15)	block13	block14	Block15

RAID 6

- Two different (P and Q) check blocks
  - Each protection group has
    - N-2 data blocks
    - One parity block
    - Another check block (not the same as parity)
- Can recover when two disks are lost
  - Think of P as the sum and Q as the product of D blocks
  - If two blocks are missing, solve equations to get both back
- More space overhead (only N-2 of N are data)
- More write overhead (must update both P and Q)

磁盘的搜索定位

- 对于移动臂磁盘设备，除了旋转位置外，还有搜索定位的问题（寻道）
- 常见的移动臂调度算法:
  - » 先来先服务
  - » 电梯调度算法
  - » 最短查找时间优先算法
  - » 扫描算法
  - » 分步扫描算法
  - » 循环扫描算法

Disk Scheduling

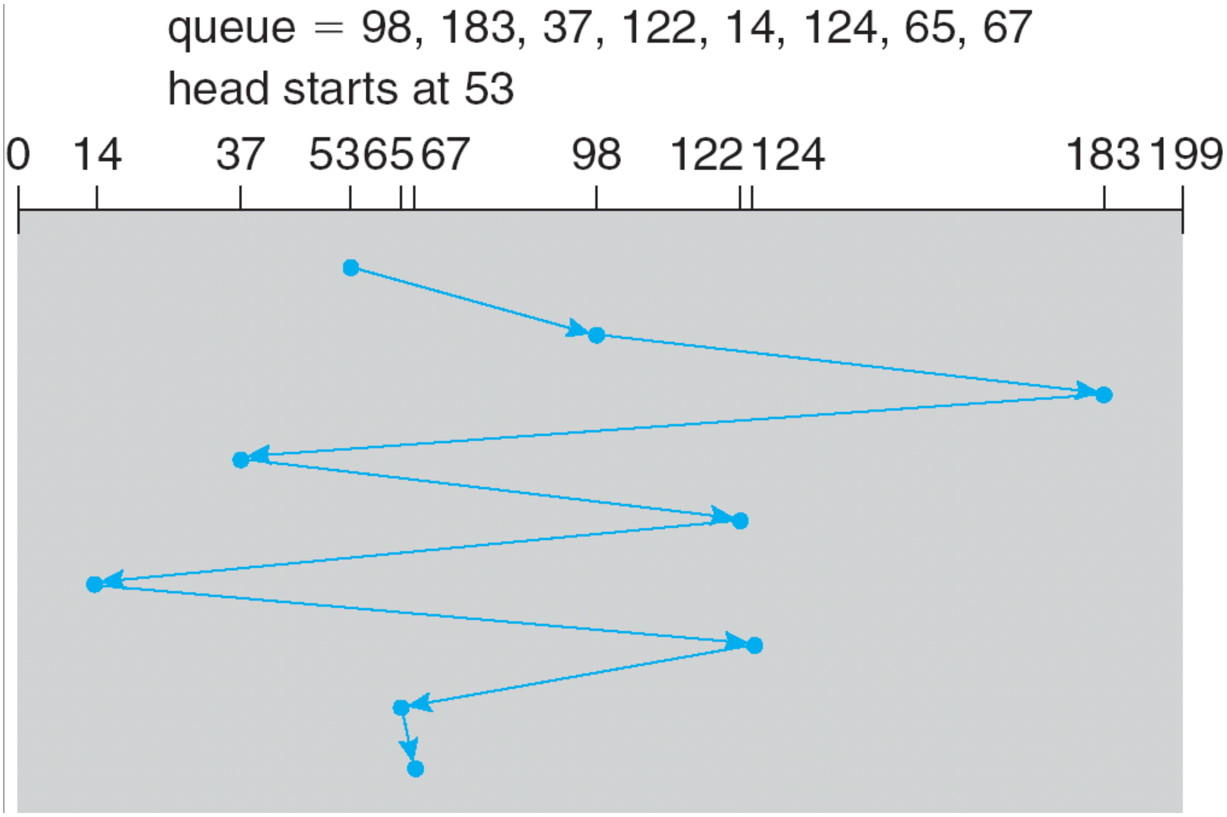
- Several scheduling algos exist service disk I/O requests.
- We illustrate them with a request queue (0-199).

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

Head pointer 53

FCFS

Illustration shows total head movement of 640 cylinders.



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

head starts at 53

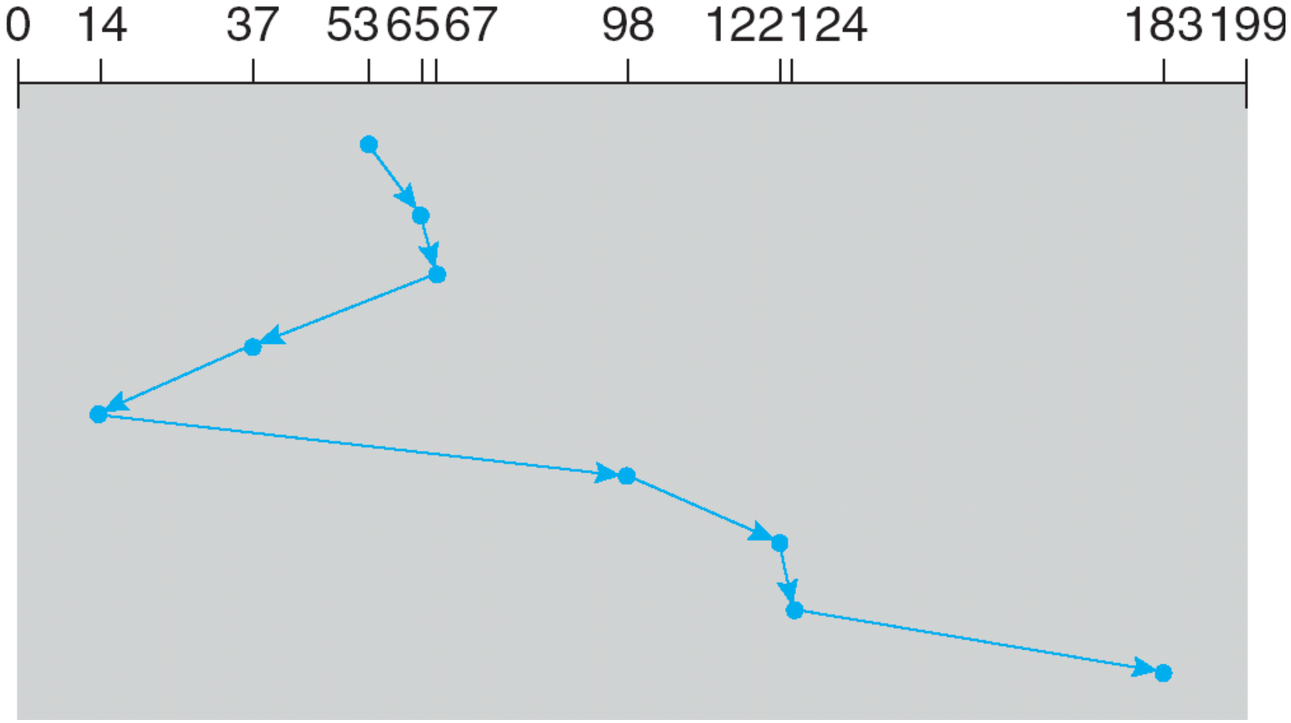
SSTF

- Selects request with minimum seek time from current head position
- SSTF scheduling is a form of SJF scheduling
  - may cause starvation of some requests.
- Illustration shows total head movement of 236 cylinders.

SSTF (Cont.)

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

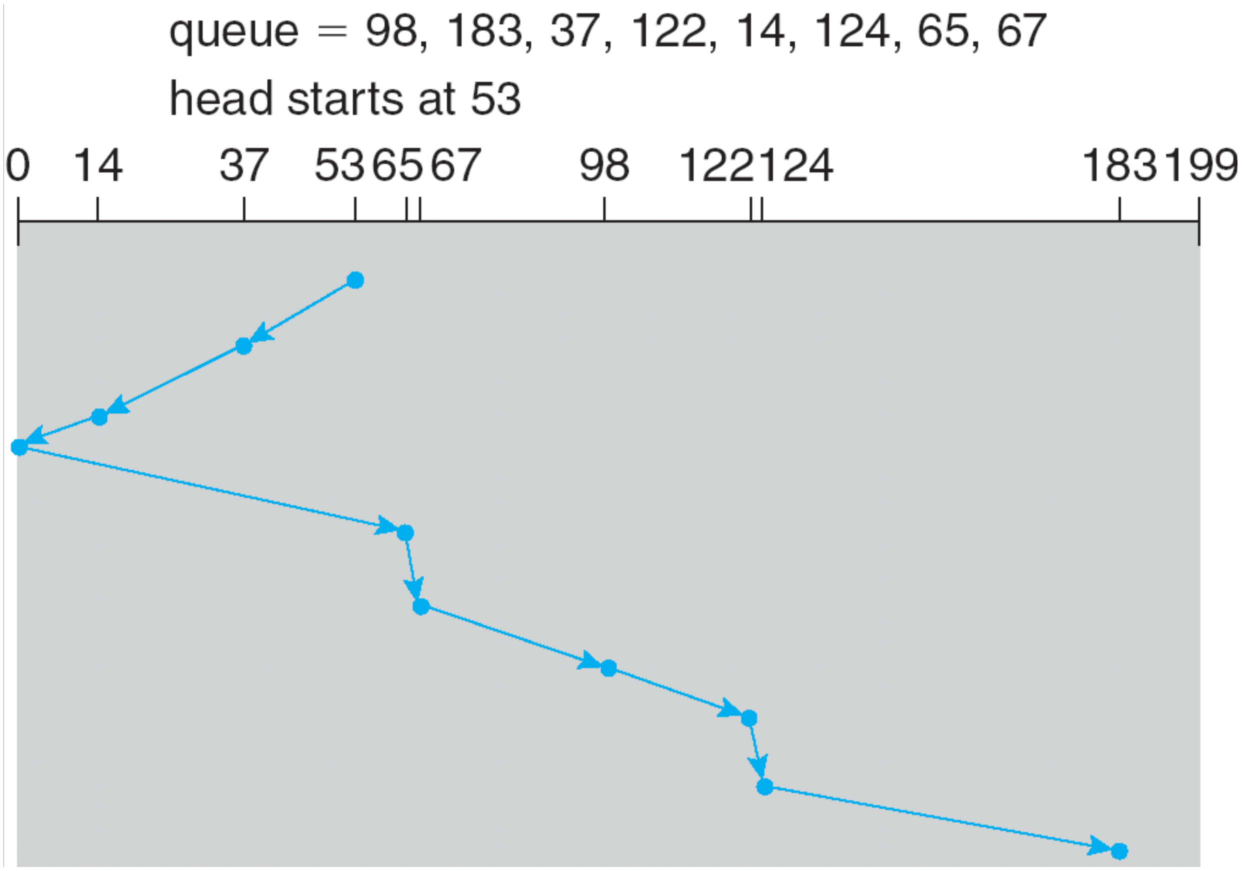
head starts at 53



SCAN

- The disk arm starts at one end of the disk,
  - moves toward the other end, servicing requests
  - head movement is reversed when it gets to the other end of disk
  - servicing continues.
- Sometimes called the *elevator algorithm*.
- Illustration shows total head movement of 208 cylinders.

SCAN (Cont.)


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

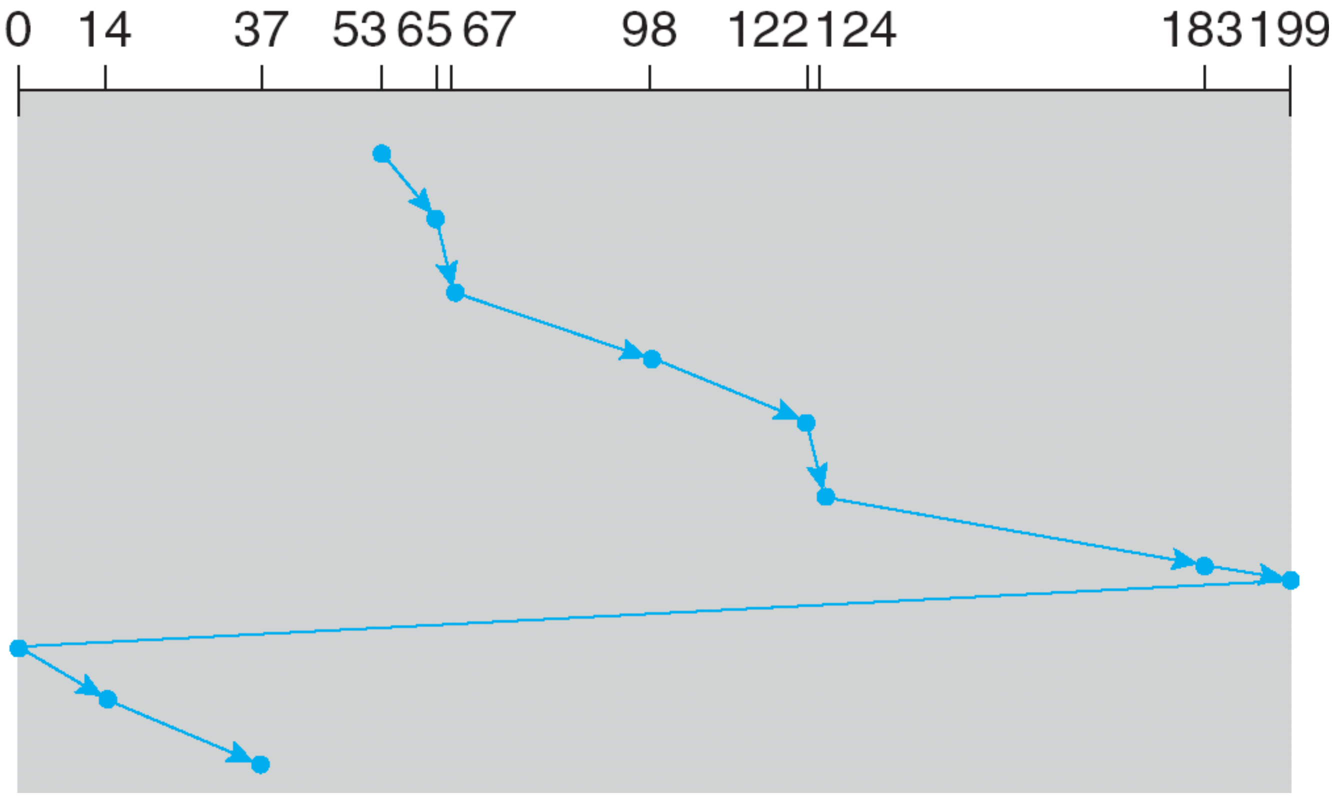
C-SCAN

- Provides a more uniform wait time than SCAN.
- The head moves from one end of the disk to the other.
  - servicing requests as it goes.
  - When it reaches the other end it immediately returns to beginning of the disk
    - No requests serviced on the return trip.
- Treats the cylinders as a circular list
  - that wraps around from the last cylinder to the first one.

C-SCAN (Cont.)

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

head starts at 53



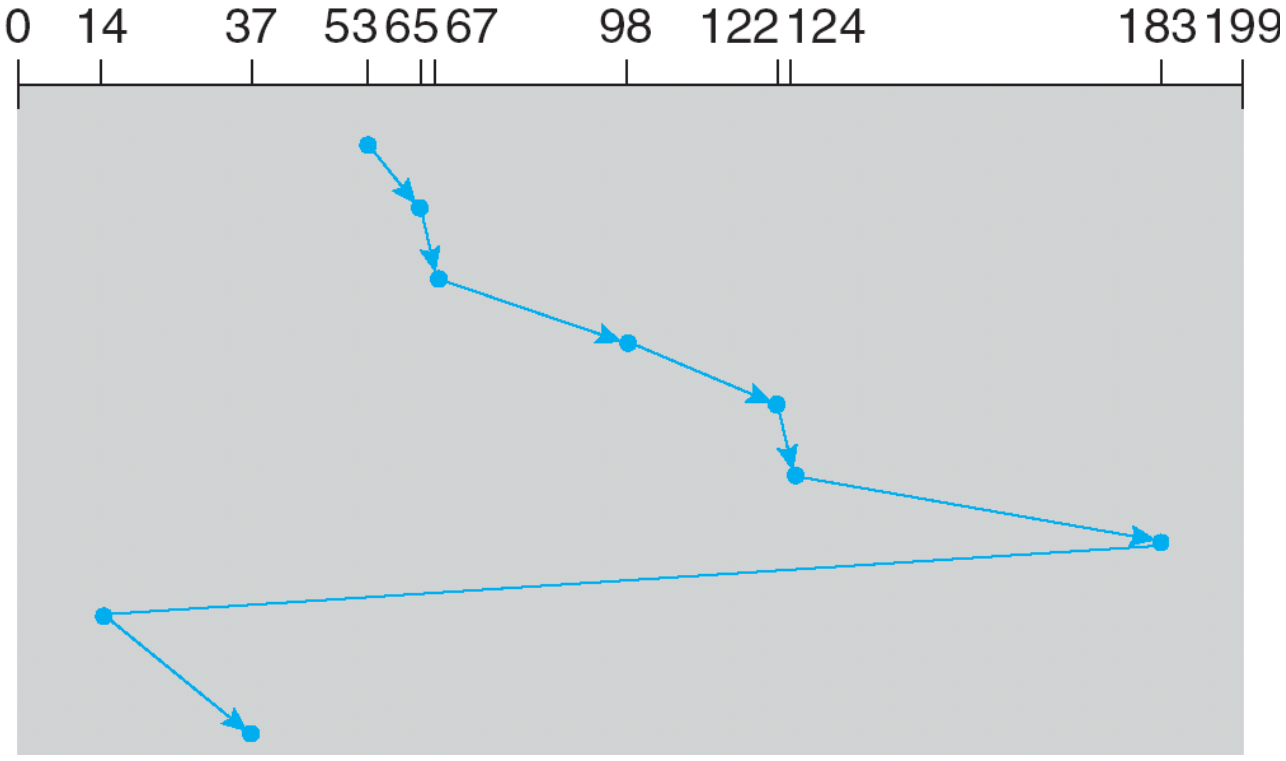
C-LOOK

- Version of C-SCAN
- Arm only goes as far as last request in each direction,
  - then reverses direction immediately,
  - without first going all the way to the end of the disk.

C-LOOK (Cont.)

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

head starts at 53



Selecting a Good Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better under heavy load
- Performance depends on number and types of requests
- Requests for disk service can be influenced by the file-allocation method.
- Disk-scheduling algorithm should be a separate OS module
  - allowing it to be replaced with a different algorithm if necessary.
- Either SSTF or LOOK is a reasonable default algorithm

磁盘的搜索定位

- 对于移动臂磁盘设备，除了旋转位置外，还有搜索定位的问题（寻道）
- 常见的移动臂调度算法:
  - 先来先服务
  - 电梯调度算法
  - 最短查找时间优先算法
  - 扫描算法
  - 分步扫描算法
  - 循环扫描算法

算法比较:

1. 1，2两种算法，单位时间内处理的I/O请求多即吞吐量大，但请求的等待时间可能较长。
2. “扫描”算法适宜于磁盘负载重的系统，它不分具体情况扫过所有柱面造成性能不够好。
3. “分步扫描”算法使得I/O请求等待时间之间的差距最小，吞吐量适中。
4. “电梯调度”算法杜绝饥饿，性能适中。
5. “循环扫描”算法适应不断有大批量柱面均匀分布的I/O请求，且磁道上存放记录数量较大的情况。

磁盘存储性能

(1) **磁盘服务：**其速度和可靠性成为文件系统性能和可靠性的主要瓶颈，设计文件系统时应尽可能减少磁盘访问次数。

(2) **块高速缓存：**系统在内存中保存一些块，逻辑上它们属于磁盘，检查所有的读请求，看所需的块是否在高速缓存中。如果在，则可直接进行读操作。否则，首先要将块读到高速缓存，再拷贝到所需的地方，如果高速缓存已满，则需要进行淘汰。

(3) **合理分配磁盘空间：**分配块时，把有可能顺序存取的块放在一起，最好在同一柱面上，从而减少磁盘臂的移动次数。

(4) **磁盘调度：**当多个访盘请求在等待时，采用一定的策略，对这些请求的服务顺序调整安排，旨在降低平均磁盘服务时间，达到公平、高效。

1.公平：一个I/O请求在有限时间内满足。

2.高效：减少设备机械运动所带来的时间浪费。

3.磁盘调度考虑的问题：一次访盘时间 = 寻道时间+旋转延迟时间+存取时间。

1) 减少寻道时间（活动头磁盘）。

2) 减少延迟时间（固定头磁盘）。

- 光盘：**光盘容量大，速度快，价格便宜，可读写光盘驱动器价格贵，写过程较麻烦。光盘的空间结构与磁盘类似。**

- 外存的特点：

（1）容量大，断电后仍可保存信息，速度较慢，成本较低。

（2）两部分组成：驱动部分+存储介质。

（3）种类很多。

（4）外存空间组织与地址、存取方式非常复杂。

（5）I/O过程方式非常复杂。

驱动调度技术

- 什么是驱动调度?
  - 系统运行时，同时会有多个访问辅助存储器的进程请求输入/输出操作，操作系统必须采用一种调度策略，使其能按最佳的次序执行各访问请求。
- 调度效率指标:
  - 若干个输入/输出请求服务所需的总时间越少，则系统效率越高

驱动调度技术

- 影响存取访问速度的因素:
  - 调度算法（策略），即如何对访问请求进行优化排序
  - 信息在辅助存储器上的排列方式
  - 存储空间的分配方法

驱动调度技术

- 提高磁盘I/O速度的一些方法
  - 提前读
  - 延迟写
  - 虚拟盘

用户对外存的要求：方便、效率、安全。

1.在读写外存时不涉及硬件细节，使用逻辑地址和逻辑操作。

2.存取速度尽可能快，容量大且空间利用率高。

3.外存上存放的信息安全可靠，防止来自硬件的故障和他人的侵权。

4.可以方便地共享，动态扩缩，携带拆卸，了解存储情况和使用情况。

5.以尽可能小的代价完成上述要求。