

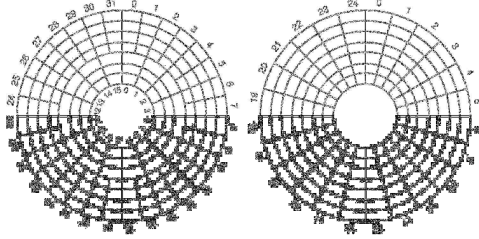
Disk I/O Management

Chapter 5

Disk Management

- Management and ordering of disk access requests is important:
 - Huge speed gap between memory and disk
 - Disk throughput is extremely sensitive to
 - Request order \Rightarrow Disk Scheduling
 - Placement of data on the disk \Rightarrow file system design
 - Disk scheduler must be aware of *disk geometry*

Disk Geometry



- Physical geometry of a disk with two zones
 - Outer tracks can store more sectors than inner without exceed max information density
- A possible virtual geometry for this disk

Evolution of Disk Hardware

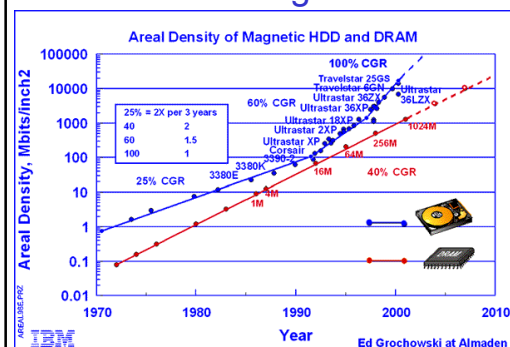
Parameter	IBM 380-KB floppy disk	WD 18300 hard disk
Number of cylinders	40	10800
Tracks per cylinder	2	12
Sectors per track	8	28 ¹ (avg)
Sectors per disk	720	35742000
Bytes per sector	512	512
Disk capacity	380 KB	18.3 GB
Seek time (adjacent cylinders)	6 msec	0.5 msec
Seek time (average case)	77 msec	6.9 msec
Rotation time	200 msec	9.33 msec
Motor stop/start time	250 msec	20 sec
Time to transfer 1 sector	22 msec	17 μ sec

Disk parameters for the original IBM PC floppy disk and a Western Digital WD 18300 hard disk

Things to Note

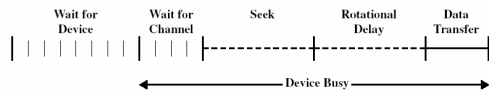
- Average seek time is approx 12 times better
- Rotation time is 24 times faster
- Transfer time is 1300 times faster
 - Most of this gain is due to increase in density
- Represents a gradual engineering improvement

Storage Capacity is 50000 times greater



Disk Performance

- Disk is a moving device \Rightarrow must be positioned correctly for I/O
- Execution of a disk operation involves
 - Wait time: the process waits to be granted device access
 - Wait for device: time the request spend in wait queue
 - Wait for channel: time until a shared I/O channel is available
 - Access time: time hardware need to position the head
 - Seek time: position the head at the desire track
 - Rotational delay (latency): spin disk to the desired sector
 - Transfer time: sectors to be read/written rotate below head



Estimating Access Time

- **Seek time T_s** : Moving the head to the required track
 - not linear in the number of tracks to traverse:
 - \rightarrow startup time
 - \rightarrow settling time
 - Typical average seek time: a few milliseconds
- **Rotational delay**:
 - rotational speed, r , of 5,000 to 10,000rpm
 - At 10,000rpm, one revolution per 6ms \Rightarrow average delay 3ms
- **Transfer time**:
 - to transfer b bytes, with N bytes per track: $T = \frac{b}{rN}$

Total average access time: $T_a = T_s + \frac{1}{2r} + \frac{b}{rN}$

A Timing Comparison

- $T_s = 2$ ms, $r = 10,000$ rpm, 512B sect, 320 sect/track
- Read a file with 2560 sectors (= 1.3MB)
- File stored compactly (8 adjacent tracks):
Read first track

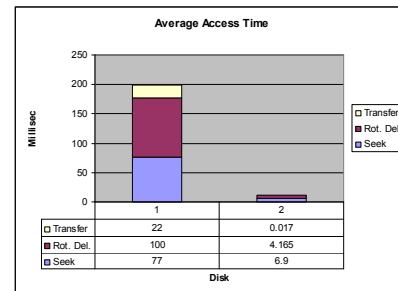
Average seek	2ms
Rot. delay	3ms
Read 320 sectors	6ms

11ms \Rightarrow All sectors: $11 + 7 * 8 = 67$ ms
- Sectors distributed randomly over the disk:
Read any sector

Average seek	2ms
Rot. delay	3ms
Read 1 sector	0.01875ms

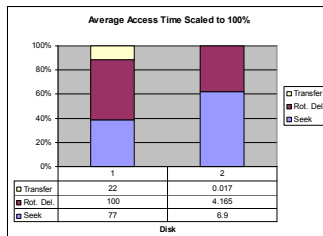
5.01875ms \Rightarrow All: $2560 * 5.01875 = 20,328$ ms

Disk Comparative Performance

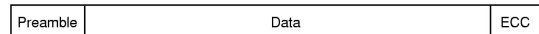


Disk Performance is Entirely Dominated by Seek and Rotational Delays

- Will only get worse as capacity increases much faster than increase in seek time and rotation speed
 - Note it has been easier to spin the disk faster than improve seek time
- Operating System should minimise mechanical delays as much as possible



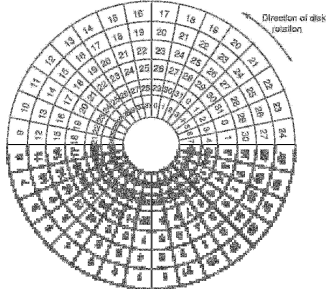
Low-level Disk Formatting



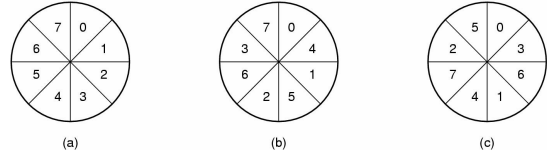
A disk sector

Low-level Disk Formatting

- When reading sequential blocks, the seek time can result in missing block 0 in the next track
- Disk can be formatted using a cylinder skew to avoid this



Low-Level Disk Formatting



- Issue: After reading one sector, the time it takes to transfer the data to the OS and receive the next request results in missing reading the next sector
- To overcome this, we can use interleaving
 - a) No interleaving
 - b) Single interleaving
 - c) Double interleaving

Low-Level Disk Formatting

- Modern drives overcome interleaving type issues by simply reading the entire track (or part thereof) into the on-disk controller and caching it.

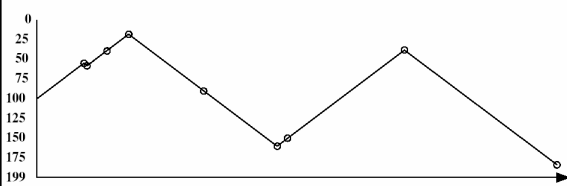
Disk Arm Scheduling Algorithms

- Time required to read or write a disk block determined by 3 factors
 1. Seek time
 2. Rotational delay
 3. Actual transfer time
- Seek time dominates
- For a single disk, there will be a number of I/O requests
 - Processing them in random order leads to worst possible performance

First-in, First-out (FIFO)

- Process requests as they come
- Fair (no starvation)
- Good for a few processes with clustered requests
- Deteriorates to random if there are many processes

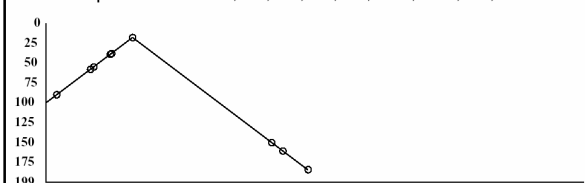
Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



Shortest Seek Time First

- Select request that minimises the seek time
- Generally performs much better than FIFO
- May lead to starvation

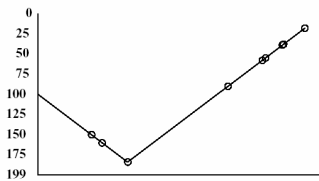
Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



Elevator Algorithm (SCAN)

- Move head in one direction
 - Services requests in track order until it reaches the last track, then reverses direction
- Better than FIFO, usually worse than SSTF
- Avoids starvation
- Makes poor use of sequential reads (on down-scan)

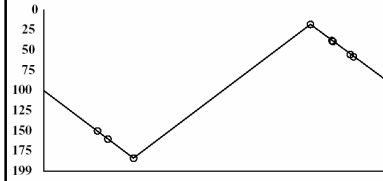
Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184



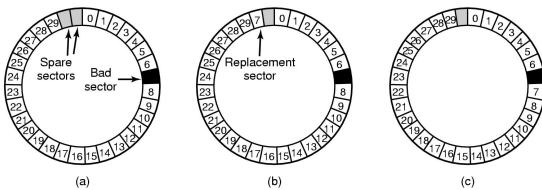
Modified Elevator (Circular SCAN, C-SCAN)

- Like elevator, but reads sectors in only one direction
 - When reaching last track, go back to first track non-stop
- Better locality on sequential reads
- Better use of read ahead cache on controller
- Reduces max delay to read a particular sector

Request tracks: 55, 58, 39, 18, 90, 160, 150, 38, 184

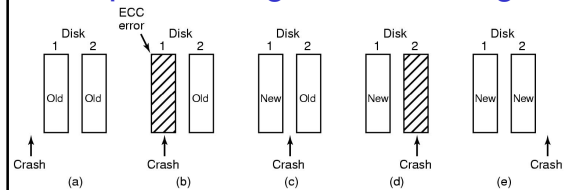


Error Handling



- A disk track with a bad sector
 - Substituting a spare for the bad sector
 - Shifting all the sectors to bypass the bad one
- Bad blocks are usually handled transparently by the on-disk controller

Implementing Stable Storage



- Use two disks to implement stable storage
 - Problem is when a write (update) corrupts old version, without completing write of new version
 - Solution: Write to one disk first, then write to second after completion of first