

Disks

Professor Hugh C. Lauer
CS-3013 — Operating Systems

(Slides include copyright materials from *Operating Systems: Three Easy Steps*, by Remzi and Andrea Arpaci-Dusseau, from *Modern Operating Systems*, by Andrew S. Tanenbaum, 3rd edition, and from other sources)

Reading Assignment

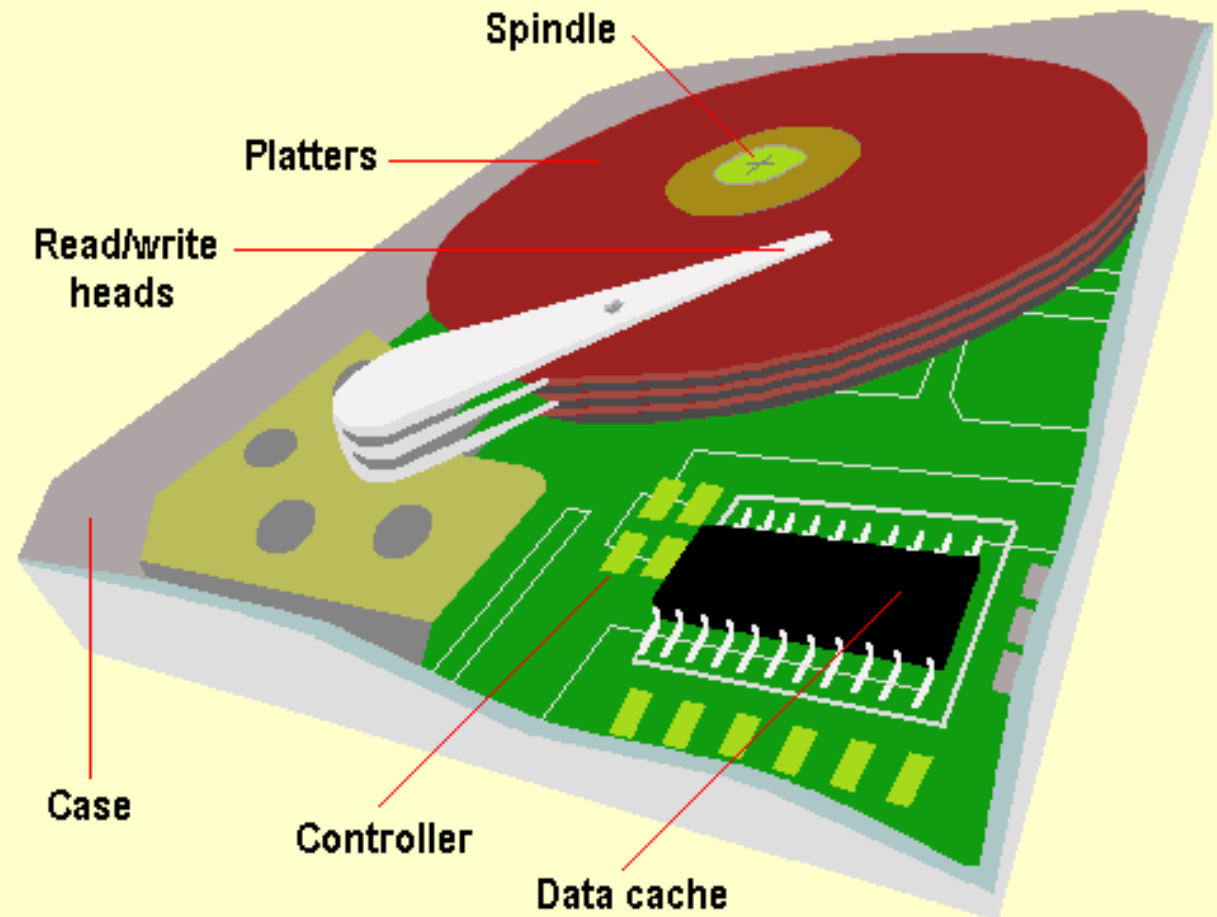
- **OSTEP, §37**

Context

- **Early days: disks were thought of as I/O devices**
 - Controlled like I/O devices
 - Block transfer, DMA, interrupts, etc.
 - Data *in* and *out* of memory (where action was thought to be!)
- **Today: disks are integral part of computing system**
 - Implementer of two fundamental abstractions
 - *Virtual Memory*
 - *Files*
 - Long term storage of information *within* system
 - The real center of action!

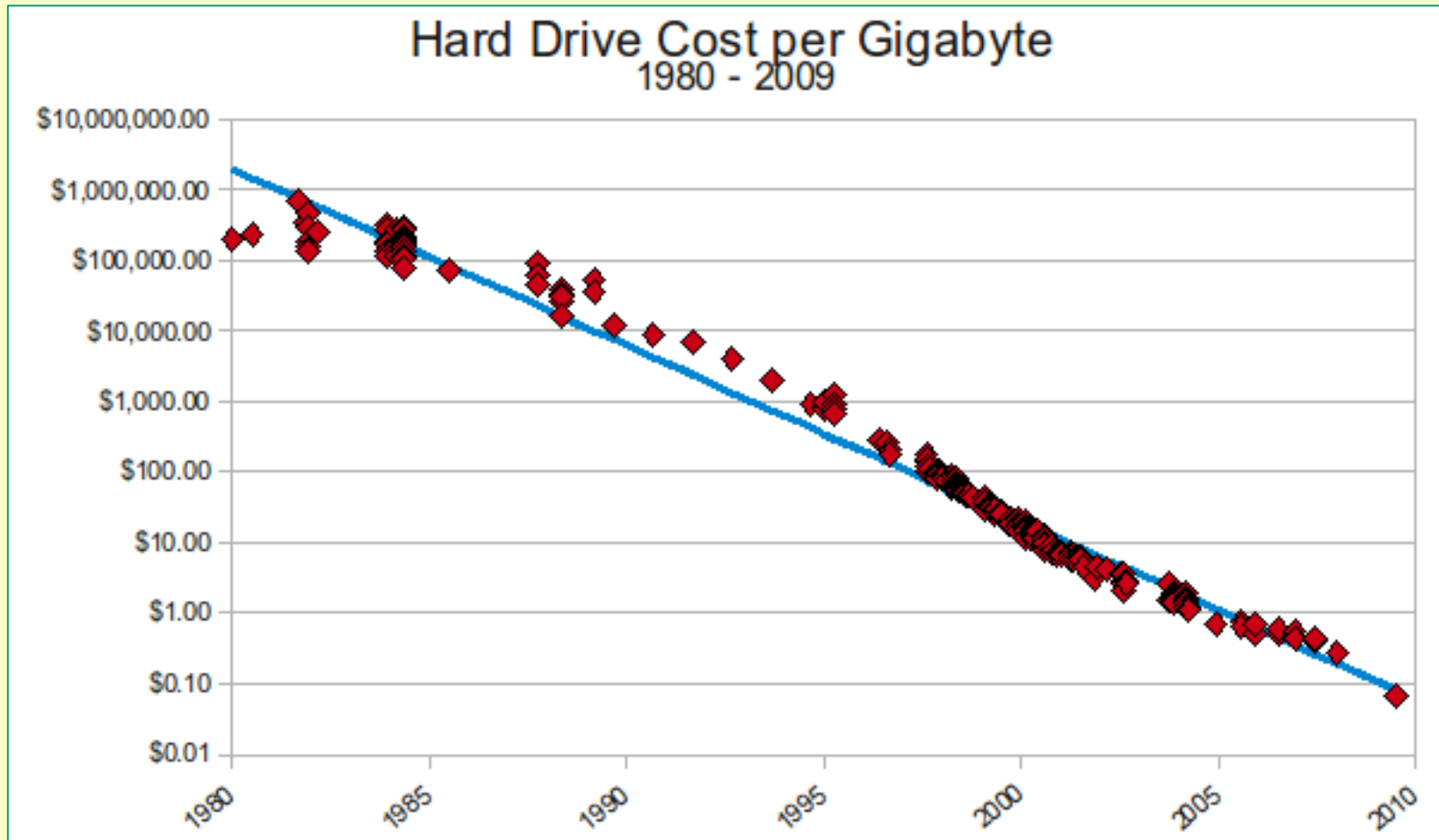
Disk Drives

- External Connection
 - IDE/ATA
 - SCSI
 - USB
- Cache – independent of OS
- Controller
 - Details of read/write
 - Cache management
 - Failure management





Price per Gigabyte of Hard Disks



Source: mkomo.com

Prices per GB (May 2010)

■ 6.3¢ per GB – 2 TB Western Digital Caviar (internal)

- 7200 rpm, 64 MB drive cache
- 3 Gb/sec SATA-II cache to host; 145 MB/sec buffer to disk

■ 5.2¢ per gigabyte – 1.5 TB Barracuda (internal)

- 7200 rpm, 8.5 ms. avg. seek time, 32 MB drive cache
- 3 Gb/sec SATA-II cache to host

■ 33.7¢ per GB – 1 TB Apple Hot Swap (internal)

- 7200 rpm, 32 MB drive cache
- 3 Gb/sec SATA-II cache to host

■ \$1.33 per GB – 300 GB IBM (hot swap)

- 15,000 rpm, avg. seek time not specified
- SATA-II

Prices per GB (Feb 2014)

■ 5.25¢ per GB – 4 TB Western Digital Red (internal)

- 64 MB drive cache
- 6 Gb/sec SATA-II cache to host

■ 5.25¢ per gigabyte – 2 TB Barracuda (internal)

- 7200 rpm, 8.5 ms. avg. seek time, 64 MB drive cache
- 6 Gb/sec SATA-II cache to host

■ 9.9¢ per GB – 1 TB Western Digital (internal)

- USB 3.0 – 5 Gbyte/sec

■ 23.5¢ per GB – 1TB Lenovo (hot swap)

- 7,200 rpm, server usage
- 6 GB/sec SATA-II;

Prices per GB (Feb 2018)

■ 2.4¢ per GB – 4 TB Seagate Barracuda (internal)

- 7200 rpm, 8.5 ms. avg. seek time, 64 MB drive cache
- 6 Gb/sec SATA-II cache to host

■ 2.4¢ per GB – 4 TB Western Digital (external)

- USB 3.0 – 5 Gbyte/sec

■ 2.37¢ per GB – 1TB Dell (hot swap)

- 7,200 rpm, server usage
- 3 GB/sec SATA

Hard Disk Geometry

■ Platters

- Two-sided magnetic material
- 1-16 per drive, 3,000 – 15,000 RPM

■ Tracks

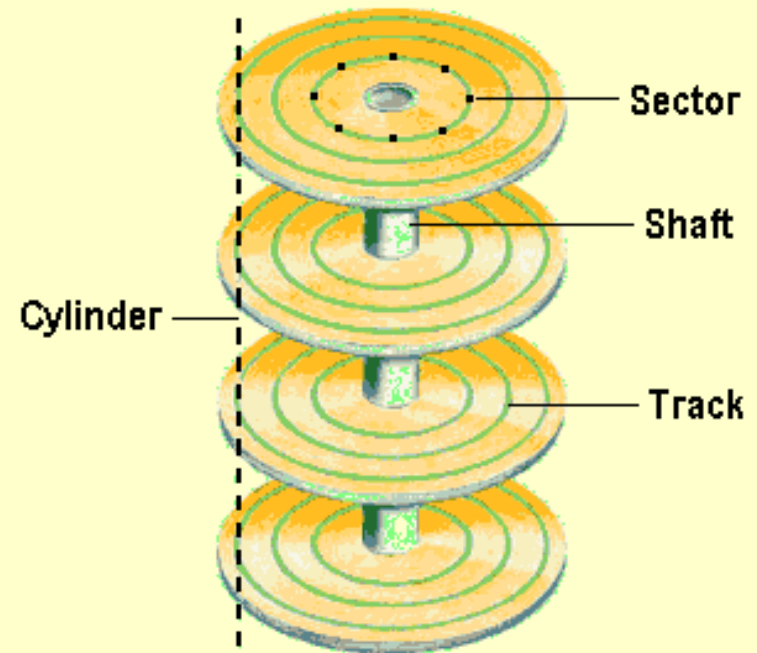
- Concentric rings bits laid out serially
- Divided into *sectors* (addressable)

■ Cylinders

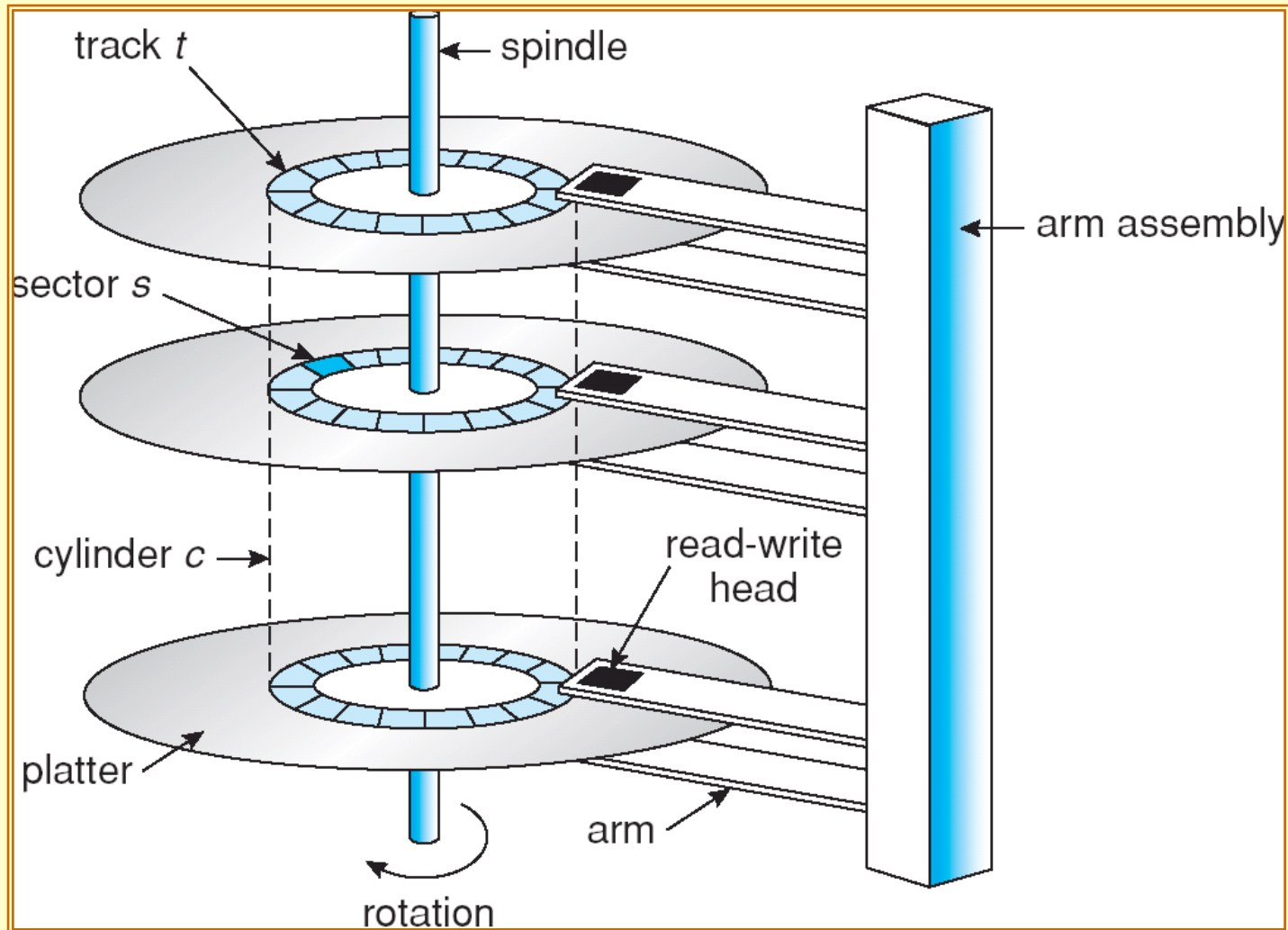
- Same track on each platter
- Arms move together

■ Operation

- *Seek*: move arm to track
- *Read/Write*:
 - wait till sector arrives under head
 - Transfer data



Moving-head Disk Mechanism



More on Hard Disk Drives

- **Manufactured in clean room**
- **Permanent, air-tight enclosure**
 - “Winchester” technology
 - Spindle motor integral with shaft
- **“Flying heads”**
 - Aerodynamically “float” over moving surface
 - Velocities > 100 meters/sec
 - Parking position for heads during power-off
- **Excess capacity**
 - Sector re-mapping for bad blocks
 - Managed by OS or by drive controller
- **20,000-100,000 hours mean time to failure**
 - Disk failure (usually) means total destruction of data & drive!



More on Hard Disk Drives (continued)

■ Early days

- Read or write platters in parallel for higher bandwidth

■ Today

- Extremely narrow tracks, closely spaced
 - tolerances < 5-20 microns
- Thermal variations prevent precise alignment across tracks within a cylinder

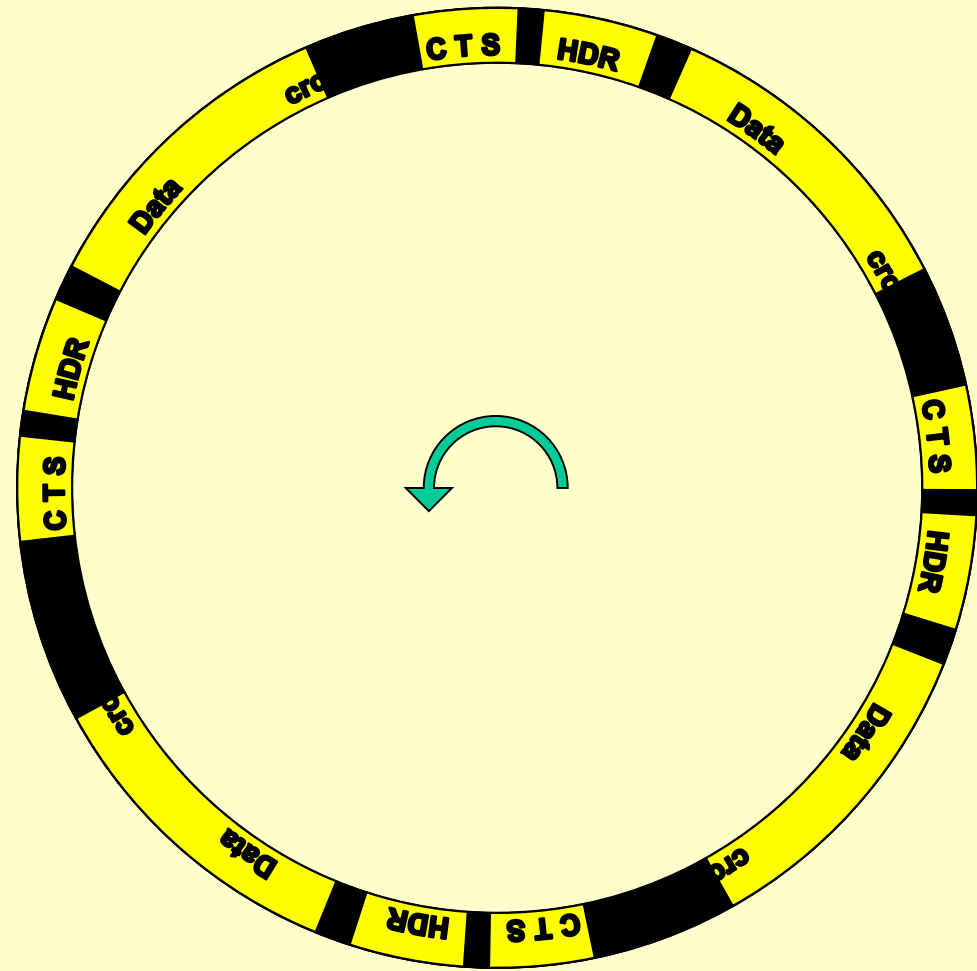


■ Seek operation

- Move arm to approximate position
- Use feedback control for precise alignment
- Seek time $\cong k * distance$

Raw Disk Layout

- **Track format – n sectors**
 - $200 < n < 2000$ s in modern disks
 - Some disks have fewer sectors on inner tracks
- **Inter-sector gap**
 - Enables each sector to be read or written independently
- **Sector format**
 - Sector address (encoded)
 - Optional header (*HDR*)
 - Data
 - Each field separated by small gap and with its own CRC
- **Sector length**
 - Almost all operating systems specify uniform sector length
 - 512 – 4096 bytes (or more)



Cylinder Skew to Improve Performance

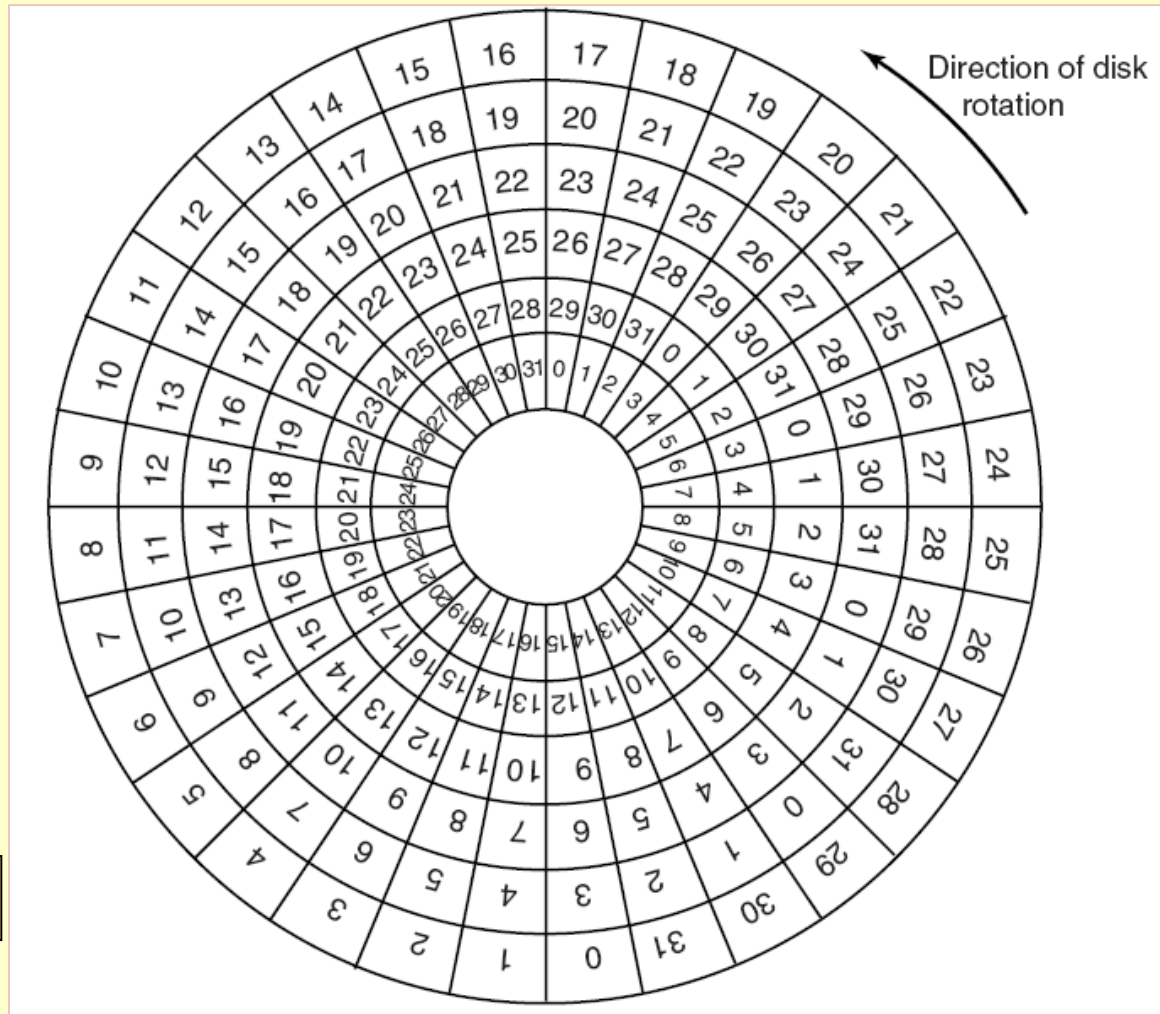


Figure 5-26

Formatting the Disk

- Write all sector addresses with dummy data
- Write and read back various data patterns on all sectors
 - Test all sectors
 - Identify *bad blocks*
- ***Bad block***
 - Any sector that does not reliably return the data that was written to it!

Bad Block Management

One of the few commodity products where manufacturers give buyers extra to compensate for spoilage!

■ Bad blocks are inevitable

- Part of manufacturing process (less than 1%)
 - Detected during formatting
- Occasionally, blocks become bad during operation

■ Manufacturers add extra tracks to all disks

- $\text{Physical capacity} = (1 + x) * \text{rated_capacity}$

■ Who handles them?

- *OS*: Bad block list maintained by OS, bad blocks never used
- *Formatter*: Re-organize track to avoid bad blocks
- *Disk controller*: Bad block list maintained internally
 - Automatically substitutes good blocks

Bad Sector Handling – within track

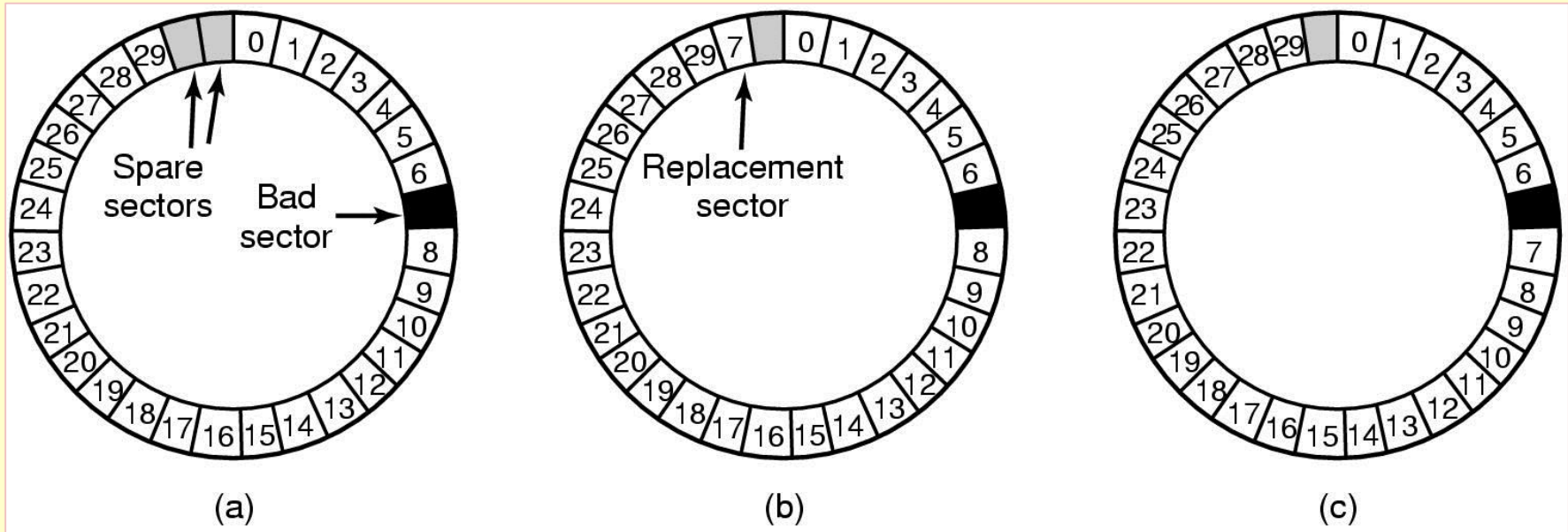


Figure 5-30

- a) A disk track with a bad sector**
- b) Substituting a spare for the bad sector**
- c) Shifting all the sectors to bypass the bad one**



Questions?

Logical vs. Physical Sector Addresses

■ Formerly:—

[cylinder, track, sector]

■ Today:

- Logical sector addresses
- Proprietary, managed by on-board controller
- Caching of whole tracks (and more)



Disk Drive – Performance

■ Seek time

- Position heads over a cylinder – 1 to 25 milliseconds (ms)

■ Rotational *latency*

- Wait for sector to rotate under head
- Full rotation - 4 to 12 msec (15000 to 5400 RPM)
- Latency averages $\frac{1}{2}$ of rotation time

■ Transfer Rate

- approx 40-380 MB/sec disk to cache (aka *bandwidth*)

■ Transfer of 1 Kbyte

- Seek (4 ms) + rotational latency (2ms) + transfer (40 μ sec) = 6.04 ms
- Effective BW here is about 170 KB/sec (misleading!)

Disk Reading Strategies

■ Read and cache a whole track

- Automatic in many modern controllers
- Subsequent reads to same track have zero rotational latency – good for locality of reference!
- Disk arm available to seek to another cylinder

■ Start from current head position

- Start filling cache with first sector under head
- Signal completion when desired sector is read

■ Start with requested sector

- When no cache, or limited cache sizes

All managed for you by on-disk controller

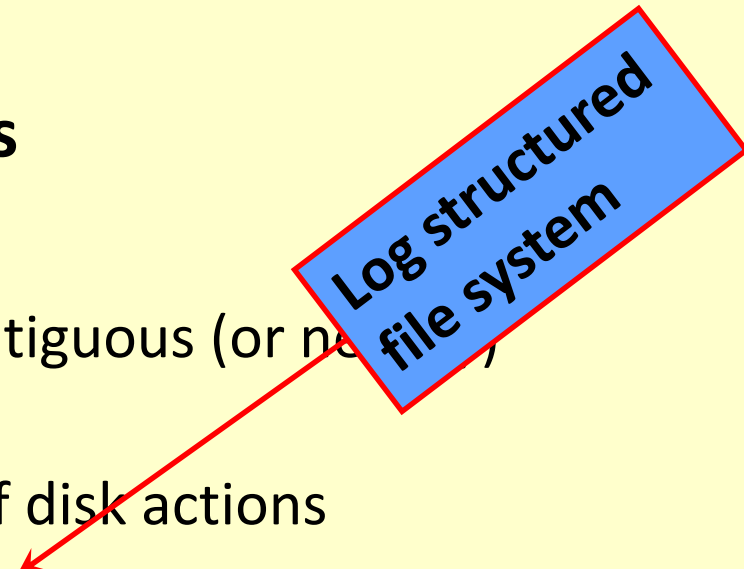
Disk Writing Strategies

- There are few, if any, good ones
- The best one can do is
 - collect together a sequence of contiguous (or nearby) sectors for writing
 - Write them in a single sequence of disk actions
- **Caching for later writing is (often) a *bad idea***
 - Application has no confidence that data is actually written before a failure
 - Some *network disk systems* provide this feature, with battery backup power for protection

Disk Writing Strategies

- There are few, if any, good ones
- The best one can do is
 - collect together a sequence of contiguous (or nearly contiguous) sectors for writing
 - Write them in a single sequence of disk actions
- **Caching for later writing is (often) a *bad idea***
 - Application has no confidence that data is actually written before a failure
 - Some *network disk systems* provide this feature, with battery backup power for protection

Log structured
file system



Disk Arm Scheduling

- A lot of material in textbooks on this subject.
- See
 - OSTEP, §37.5
- Goal
 - Minimize seek time by minimizing seek distance

However ...

Additional slides about
Disk Arm Scheduling
are at the end of this
presentation

- **In real systems, average disk queue length is often < 2 requests**
 - All strategies are approximately equal!
- **If your system typically has queues averaging more than 2 entries, something is seriously wrong with your design!**
- **Disk arm scheduling used only in a few very specialized situations**
 - Multi-media; some transaction-based systems



Performance metrics

■ Transaction & database systems

- Number of transactions per second
- Focus on seek and rotational latency, not bandwidth
- Track caching may be irrelevant (except read-modify-write)

■ Many little files (e.g., Unix, Linux)

- Same

■ Big files

- Focus on bandwidth and contiguous allocation
- Track caching important; seek time is secondary concern

■ Paging support for VM

- A combination of both
- Track caching is highly relevant – locality of reference

Hard Drive Failures ...

- **Fact of life with hard drives**

- **100,000 hours MBTF \Rightarrow**

- One failure in ~11.4 years for a single drive
- > One failure per year for 12 drives!
- > One failure per month for 150 drives
- Daily drive replacements for warehouse scale facilities!

← **On average**

- **Need redundancy and backup!**

Problem

■ Question:—

- If *mean time to failure* of a disk drive is 100,000 hours,
- and if your system has 100 identical disks,
- what is mean time between need to replace a drive?

■ Answer:—

- 1000 hours (i.e., 41.67 days \cong 6 weeks)

■ I.e.:—

- You lose 1% of your data every 6 weeks!

■ But don't worry – you can restore *most* of it from backup!

Can we do better?

■ Yes, *mirrored*

- Write every block twice, on two separate disks
- Mean time between simultaneous failure of *both* disks is >57,000 years

■ Can we do even better?

- E.g., use fewer extra disks?
- E.g., get more performance?

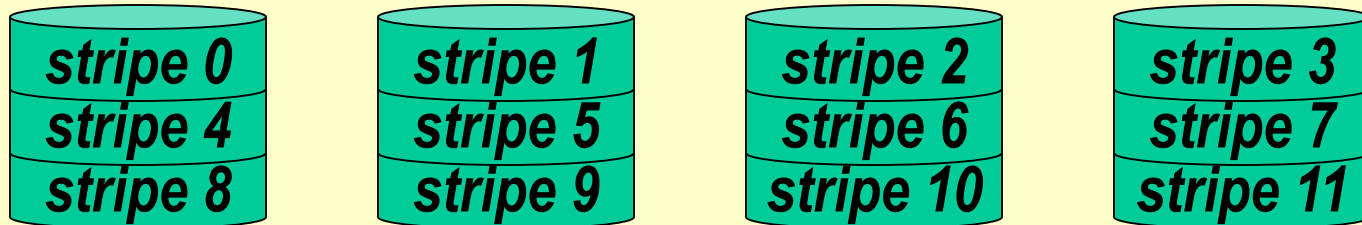
RAID – Redundant Array of Inexpensive Disks

- **Distribute a file system intelligently across multiple disks to**
 - Maintain high reliability *and* availability
 - Enable fast recovery from failure
 - Increase performance

“Levels” of RAID

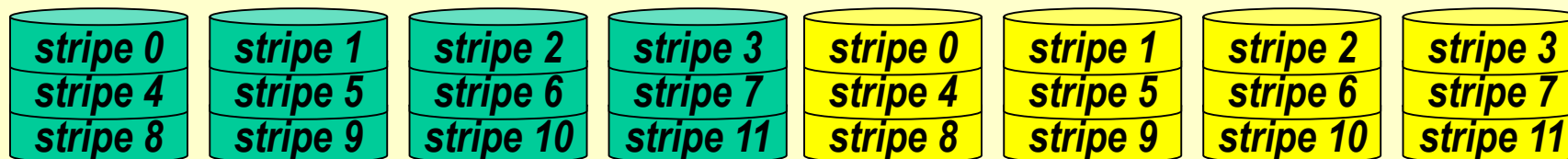
- Level 0 – non-redundant striping of blocks across disk
- Level 1 – simple mirroring
- Level 2 – striping of bytes or bits with Never seriously used
- ~~■ Level 3 – Level 2 with parity, not ECC~~
- ~~■ Level 4 – Level 0 with parity block~~
- Level 5 – Level 4 with distributed parity blocks
- ...

RAID Level 0 – Simple Striping



- Each stripe is one or a group of contiguous blocks
- Block/group i is on disk $(i \bmod n)$
- **Advantage**
 - Read/write n blocks in parallel; n times bandwidth
- **Disadvantage**
 - No redundancy at all. System MBTF is $1/n$ disk MBTF!

RAID Level 1– Striping and Mirroring



- **Each stripe is written twice**
 - Two separate, identical disks
- **Block/group i is on disks $(i \bmod 2n)$ & $(i+n \bmod 2n)$**
- **Advantages**
 - Read/write n blocks in parallel; n times bandwidth
 - Redundancy: System MBTF = (Disk MBTF)² at twice the cost
 - Failed disk can be replaced by copying
- **Disadvantage**
 - A lot of extra disks for much more reliability than we need

RAID Levels 2 & 3

- **Bit- or byte-level striping**
- **Requires synchronized disks**
 - Highly impractical
- **Requires fancy electronics**
 - For ECC calculations
- **Not used; academic interest only**
- **See Tanenbaum, §5.4.1 (pp. 363-367)**

Observation

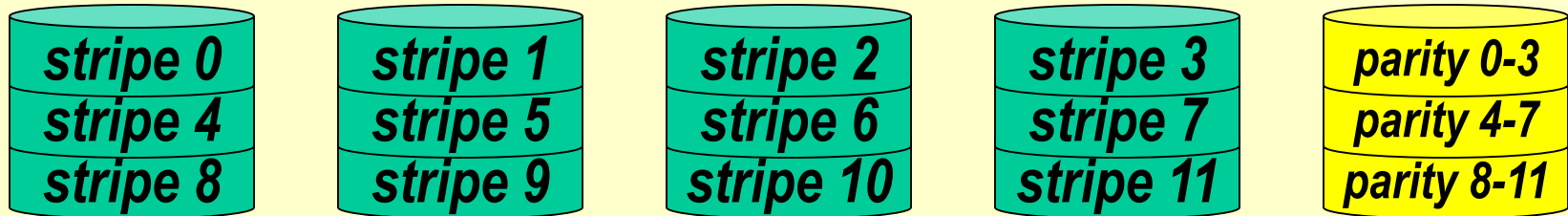
- When a disk or stripe is read incorrectly,

we know which one failed!

- **Conclusion:**

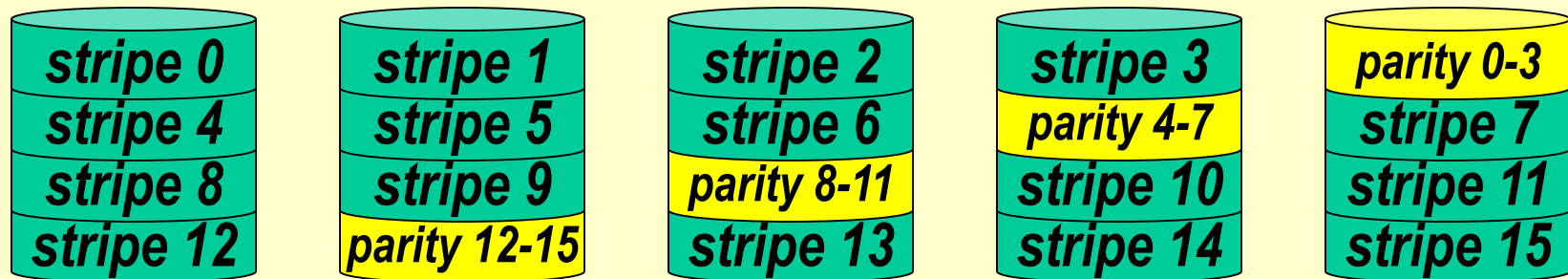
- A simple parity disk can provide very high reliability
 - (unlike simple parity in memory)

RAID Level 4 – Parity Disk



- **parity 0-3 = stripe 0 xor stripe 1 xor stripe 2 xor stripe 3**
- **n stripes plus parity are written/read in parallel**
- **If any disk/stripe fails, it can be reconstructed from others**
 - E.g., stripe 1 = stripe 0 xor stripe 2 xor stripe 3 xor parity 0-3
- **Advantages**
 - n times read bandwidth
 - System MBTF = (Disk MBTF)² at $1/n$ additional cost
 - Failed disk can be reconstructed “on-the-fly” (hot swap)
 - Hot expansion: simply add $n + 1$ disks all initialized to zeros
- **However**
 - Writing requires read-modify-write of parity stripe \Rightarrow only 1x write bandwidth.

RAID Level 5 – Distributed Parity



- Parity calculation is same as RAID Level 4
- Advantages & Disadvantages – Mostly same as RAID Level 4
- Additional advantages
 - avoids beating up on parity disk
 - Some writes in parallel (if no contention for parity drive)
- Writing individual stripes (RAID 4 & 5)
 - Read existing stripe and existing parity
 - Recompute parity
 - Write new stripe and new parity

RAID 4 & 5

- **Very popular in data centers**
 - Corporate and academic servers
- **Built-in support in Windows XP and Linux**
 - Connect a group of disks to fast SCSI port (320 MB/sec bandwidth)
 - OS RAID support does the rest!
- **Other RAID variations also available**

Another Problem

Incomplete Operations

- **Problem – how to protect against disk write operations that don't finish**
 - Power or CPU failure in the middle of a block
 - Related series of writes interrupted before all are completed

- **Examples:**
 - Database update of charge and credit
 - RAID 1, 4, 5 failure between redundant writes

Solution (part 1) – Stable Storage

■ Write everything twice to separate disks

- Be sure 1st write does not invalidate previous 2nd copy
- RAID 1 is okay; RAID 4/5 *not* okay!
- Read blocks back to validate; then report completion

■ Reading both copies

- If 1st copy okay, use it – i.e., newest value
- If 2nd copy different or bad, update it with 1st copy
- If 1st copy is bad; update it with 2nd copy – i.e., *old value*

Stable Storage (continued)

■ Crash recovery

- Scan disks, compare corresponding blocks
- If one is bad, replace with good one
- If both good but different, replace 2nd with 1st copy

■ Result:—

- If 1st block is good, it contains latest value
- If not, 2nd block still contains previous value

■ An *abstraction* of an *atomic disk write* of a single block

- Uninterruptible by power failure, etc.

Stable Storage (continued)

- See Tanenbaum, §5.4.5
- Also:—
 - Lampson, B.W., and Sturgis, H. E., “Crash Recovery in a Distributed Data Storage System,” Xerox PARC Report, June 1, 1979 ([.pdf](#))

Solution (Part 2)

- ***Log-structured file system (aka journaling file system)***
 - Topic for CS-4513

Questions?

Disk Arm Scheduling

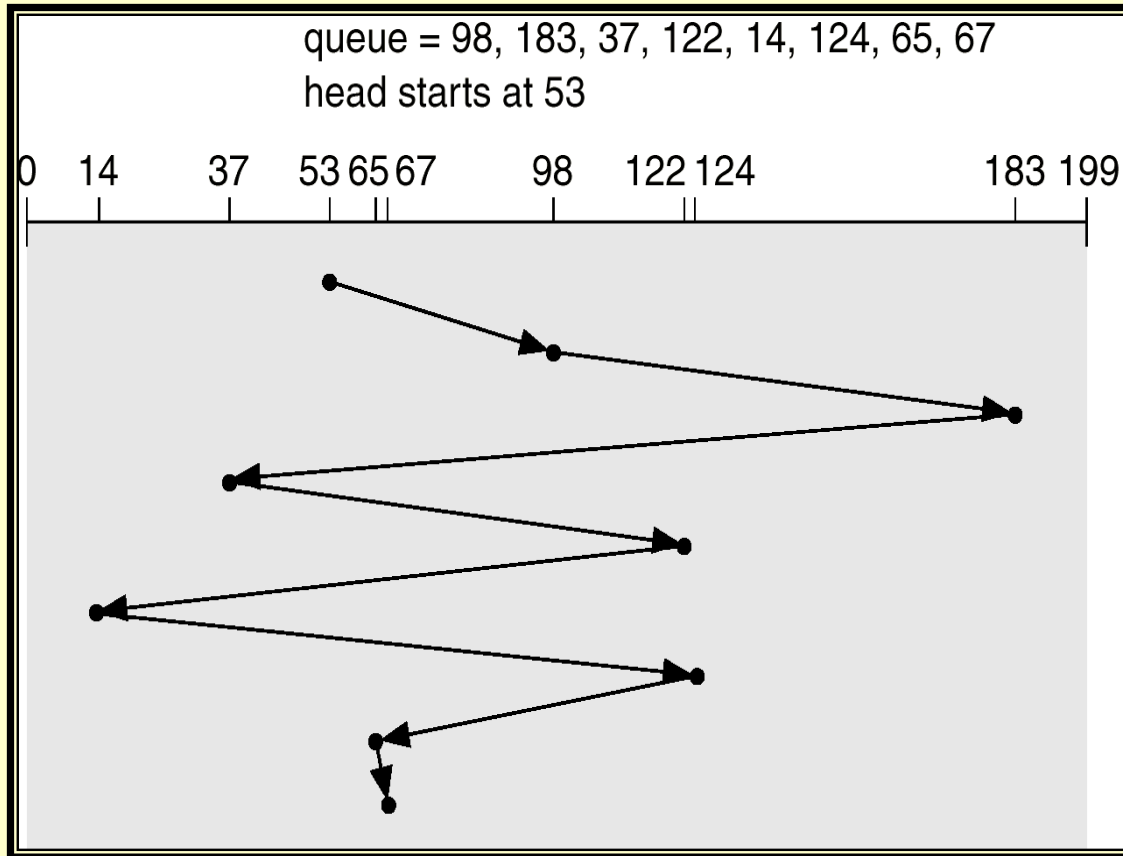
- **Seek time dominates the performance of disk drive transfers**
- **Can the disk arm be moved to improve the effective disk performance?**
- **Assume a request queue (0-199)**
98, 183, 37, 122, 14, 124, 65, 67
with current head pointer at 53

Textbook solutions

- ***FCFS*** – First-come, first-served
- ***SSTF*** – Shortest seek time first
- ***SCAN*** (aka *Elevator*) – scan one direction, then the other
- ***C-SCAN*** – scan in one direction only
- ...

OSTEP, §37.5

FCFS – First come, first served



■ Example

- total head movement of 640 cylinders for request queue

■ Pros

- In order of applications
- Fair to all requests

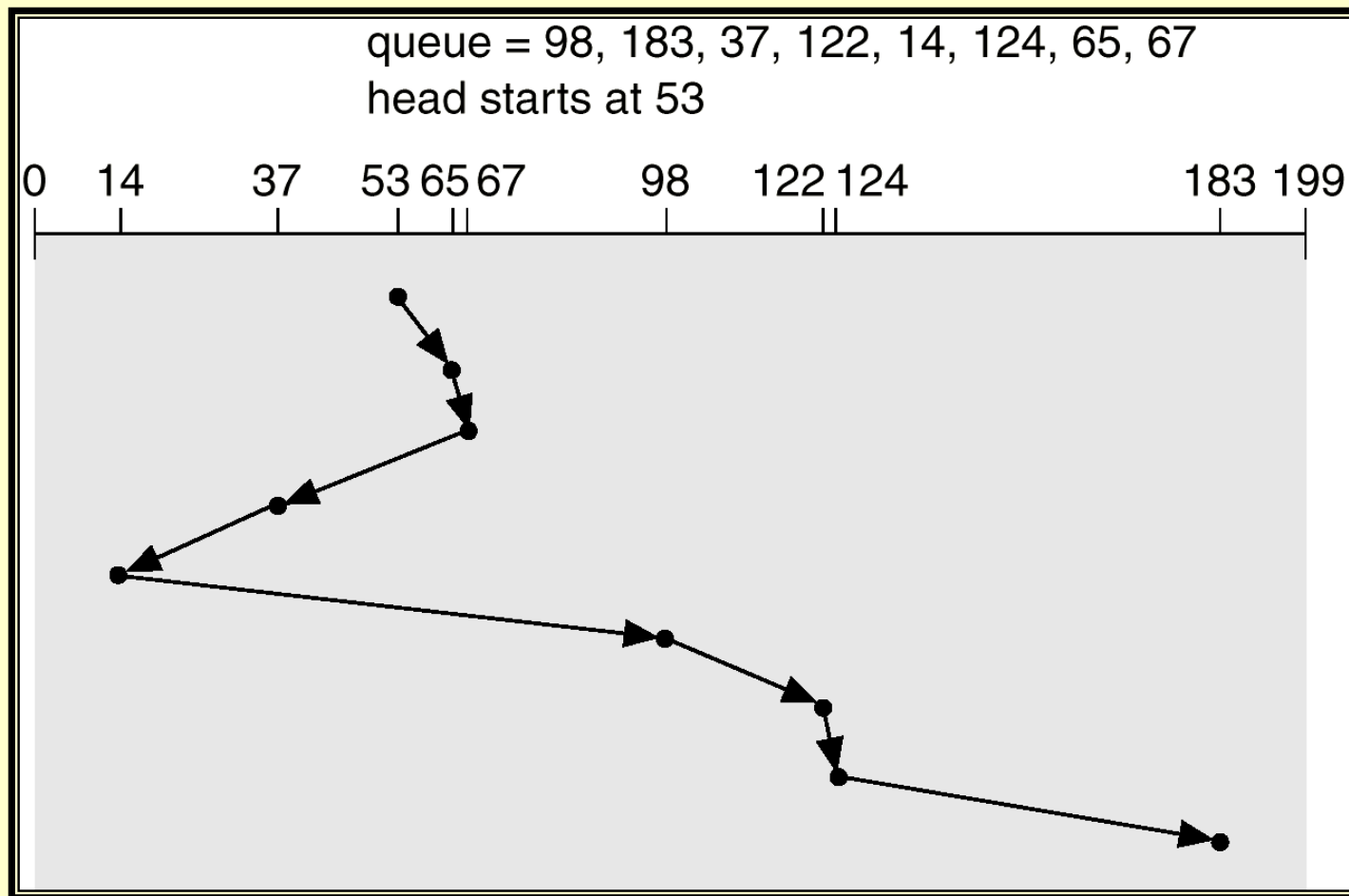
■ Cons

- Long seeks

SSTF

- **Shortest Seek Time First – Selects request with the minimum seek time from current head position.**
- **Pro**
 - minimize seek times
- **Cons**
 - Lingers in areas of high activity
 - Starvation, particularly at edges of disk
- **Example**
 - total head movement of 236 cylinders for request queue

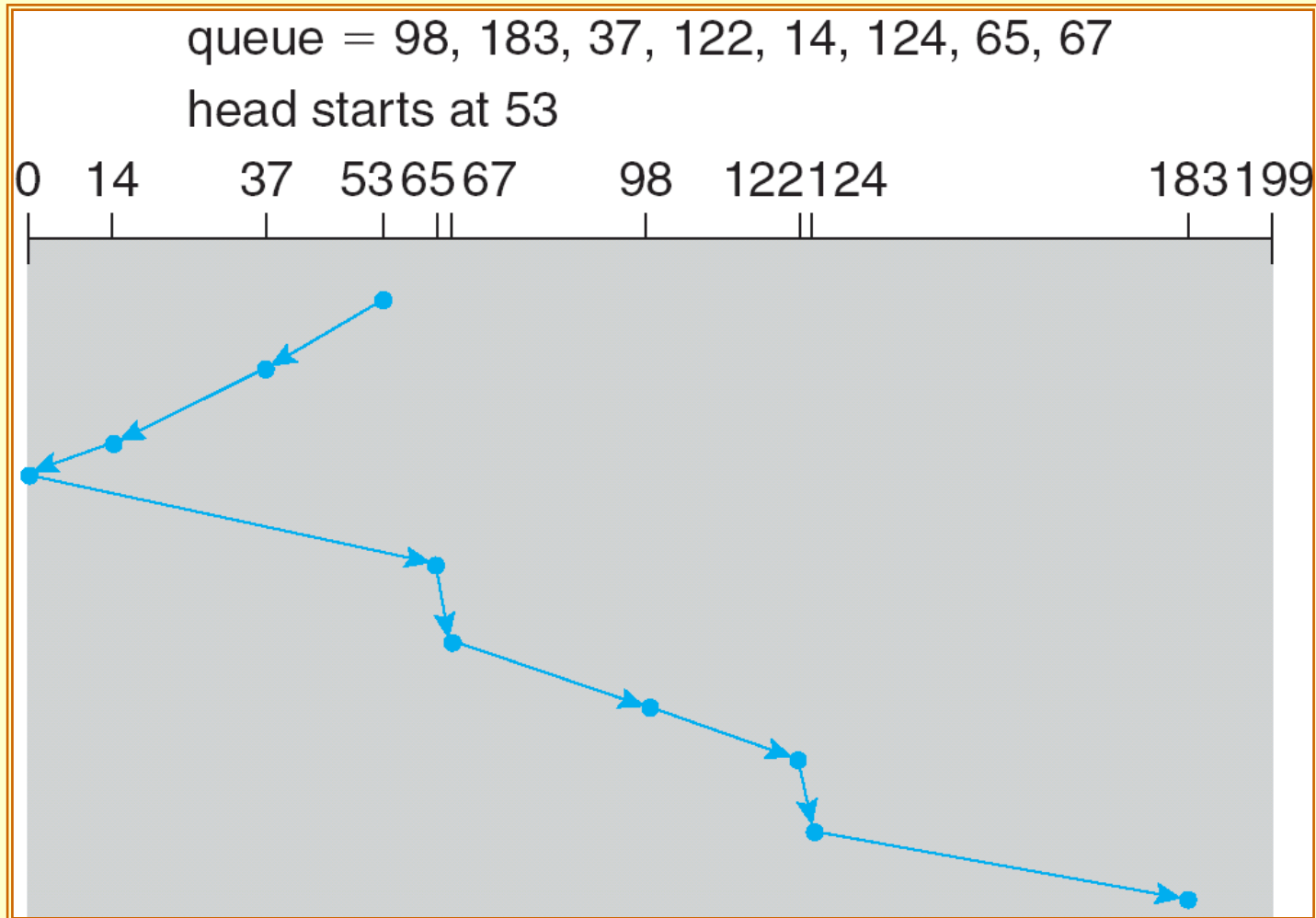
SSTF



SCAN or Elevator

- **The disk arm starts at one end of the disk, moves toward the other end, servicing requests until reaching end of disk, then the head reverses and servicing continues.**
 - i.e. Pick the closest request in same direction as last arm motion
- **Con – more arm motion than SSTF**
- **Pro**
 - Fair
 - Avoids starvation
- **Example total head movement of 208 cylinders.**

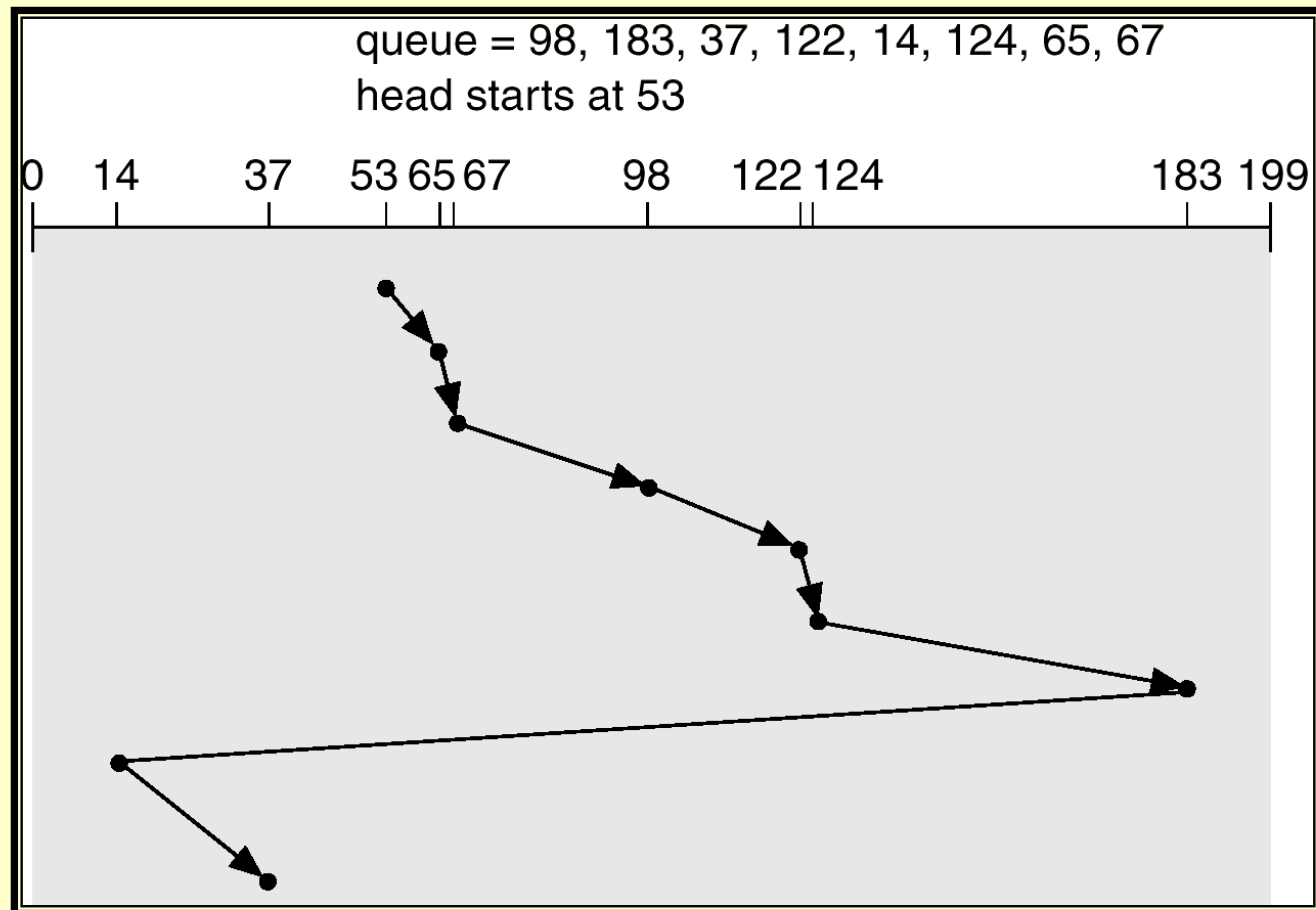
Scan (continued)



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 the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.

C-SCAN (continued)



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal.
- SCAN and C-SCAN perform better for systems that place heavy load on the disk.
- Performance depends on the number and types of requests.
- Requests for disk service are influenced by the file-allocation method.



Questions?

