Persistent Storage Devices

Note: Some slides and/or pictures in the following are adapted from the text books on OS by Silberschatz, Galvin, and Gagne AND Andrew S. Tanenbaum and Albert S.Woodhull. Slides courtesy of Kubiatowicz, AJ Shankar, George Necula, Alex Aiken, Eric Brewer, Ras Bodik, Ion Stoica, Doug Tygar, David Wagner, Hakim Weatherspoon, etc.

Outline

PART-I

- Disks
  - How does a computer system permanently store data?
  - Disk Structure

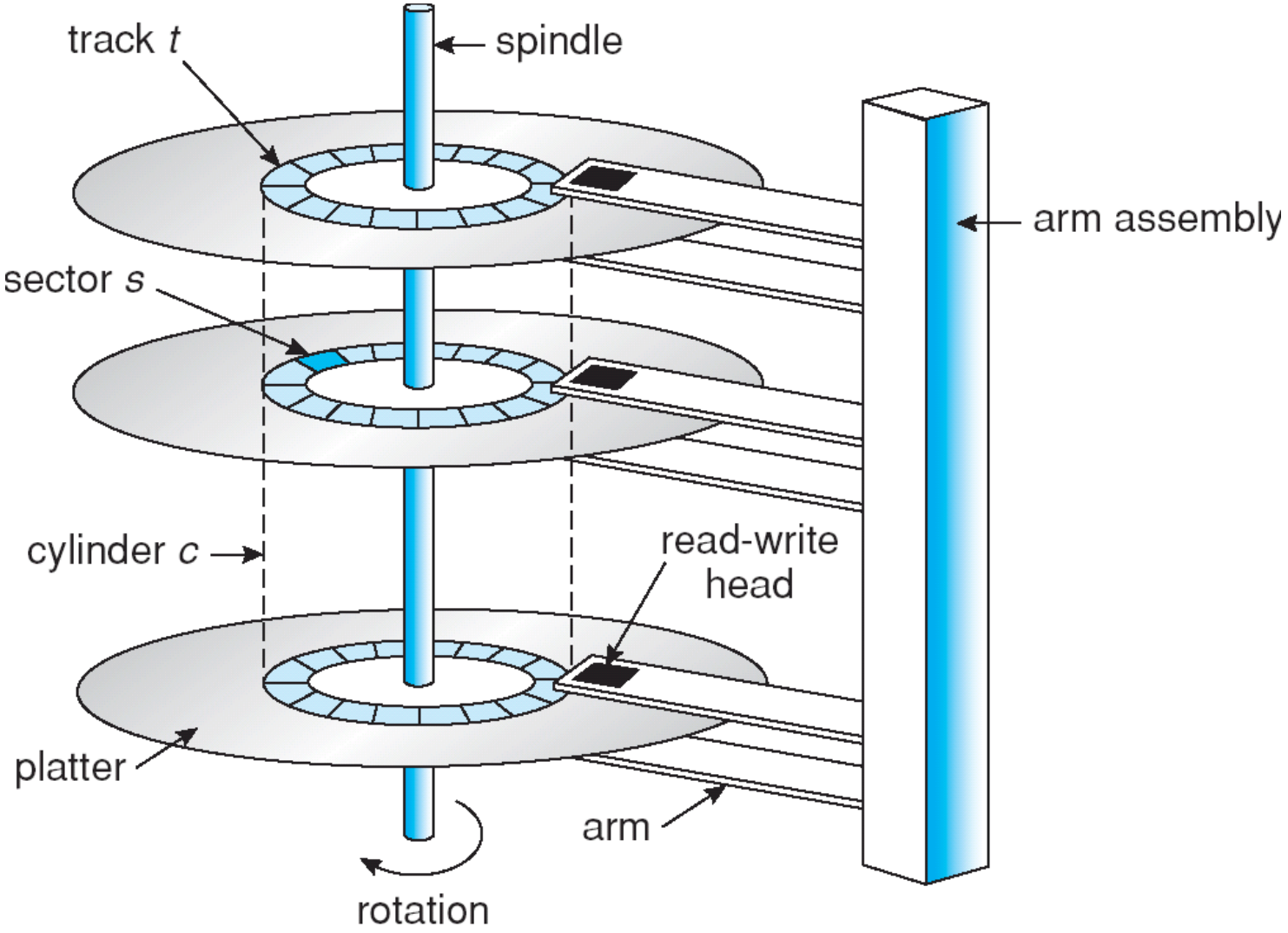
PART-II

- Disk Scheduling
  - Scheduling of I/O requests for improved I/O throughput
- Disk Formatting

PART-III

- RAID
  - How to make storage both efficient and reliable?
  - RAID levels

What does Magnetic Disk look like?



One motor, so all arms move together

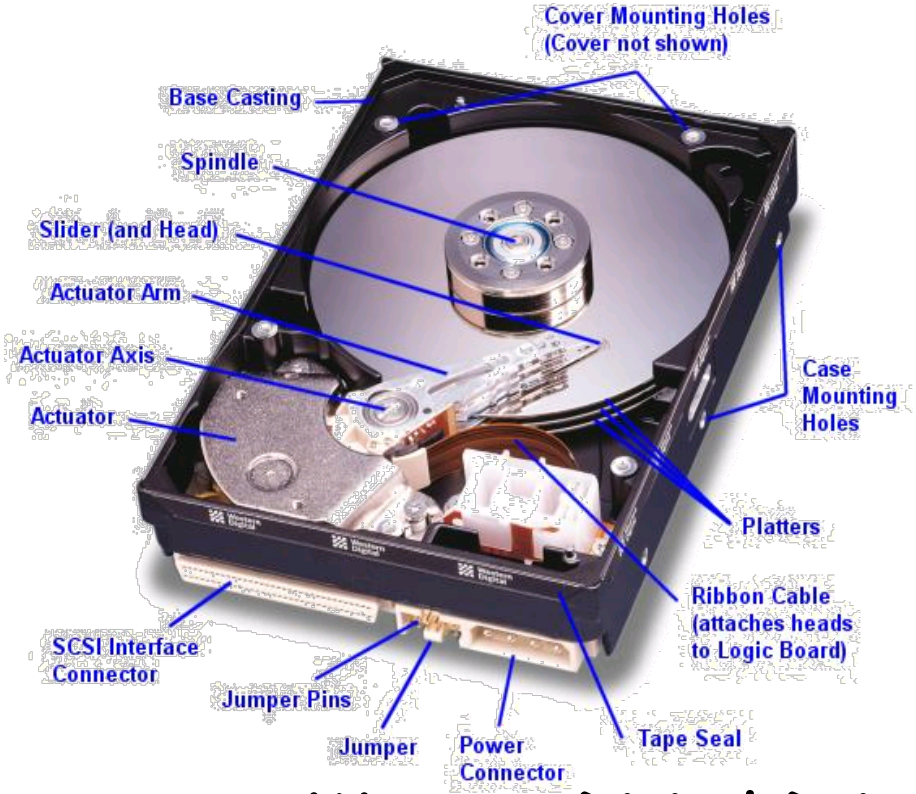
64 years ago…!

- On 13th September 1956, IBM 305 RAMAC computer system first to use disk storage

- 80000 times more data on the 8GB 1-inch drive in his right hand than on the 24-inch RAMAC one in his left having ~0.1 MB!

Some parameters & properties

- 2-30 heads (platters * 2)
  - Diameter of 3.5'' to 1.8''
- 700-20480 tracks per surface
- 16-1600 sectors per track
- Sector size:
  - 64-8KB
  - 512 B for most PCs
- Capacity: 100GB-8TB
- Main adjectives: BIG, Slow
- Properties
  - Head moves in to address circular track of information
  - Independently addressable element: sector #
    - OS always transfers groups of sectors together—"blocks"
  - Items addressable without moving head: cylinder
  - A disk can be rewritten in place: it is possible to read/modify/write a block from the disk



Western Digital Drive

Disks vs. Main Memory

- Smallest write: sector
- Atomic write = sector
- Random access: 5ms
  - not on a good curve
- Sequential access: 20 MB/s
- Cheap
- Crash: doesn't matter ("non-volatile")

- (usually) bytes
- byte, word (4 bytes)
- 50 ns
  - faster all the time
- 20 GB/s
- Relatively expensive
- contents gone ("volatile")

Disk Structure

- Disk drives addressed as 1-dimensional arrays of *logical blocks*
  - the logical block is the smallest unit of transfer
  - Low-level formatting creates logical blocks on physical media
- This array mapped sequentially onto disk sectors
  - Address 0 is 1st sector of 1st track of the outermost cylinder
  - Addresses incremented within track, then within tracks of the cylinder, then across cylinders, from innermost to outermost
- Translation is theoretically possible, but usually difficult
  - Some sectors might be defective
  - Number of sectors per track is not a constant!

No. of Sectors / track

- Two Approaches:
  - Density of bits per track is uniform: More sectors per track on the outer layers (Fig. A) in DVDs
  - Reduce bit density per track for outer layers (Fig. B): Used in Disks for keeping data rate constant with constant rotation speed

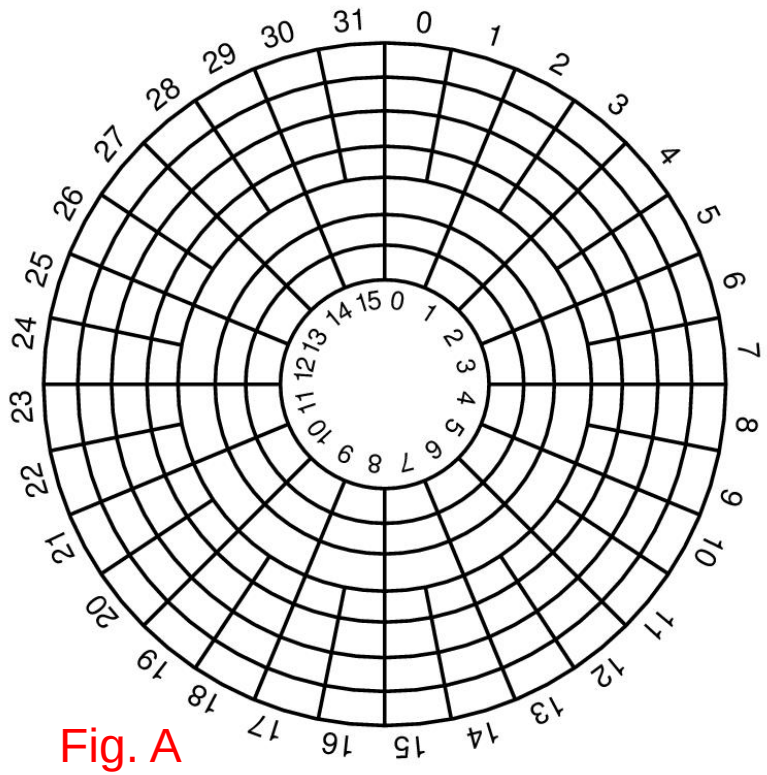


Fig. A

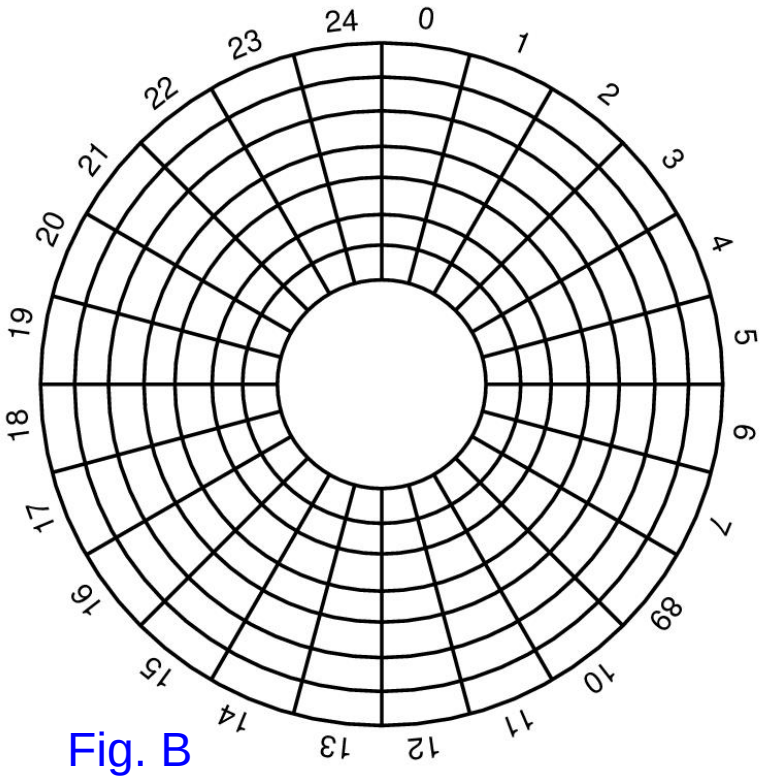
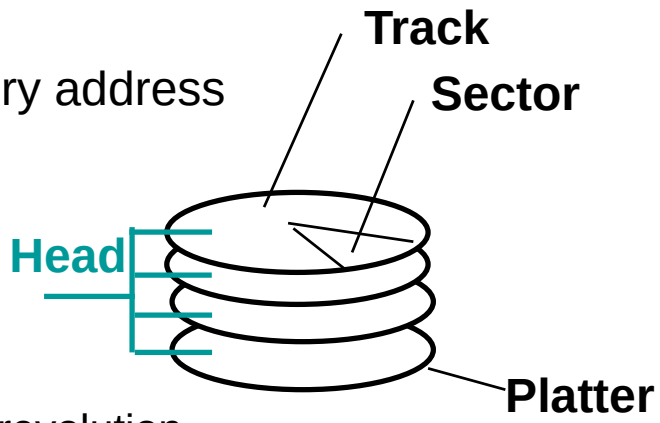
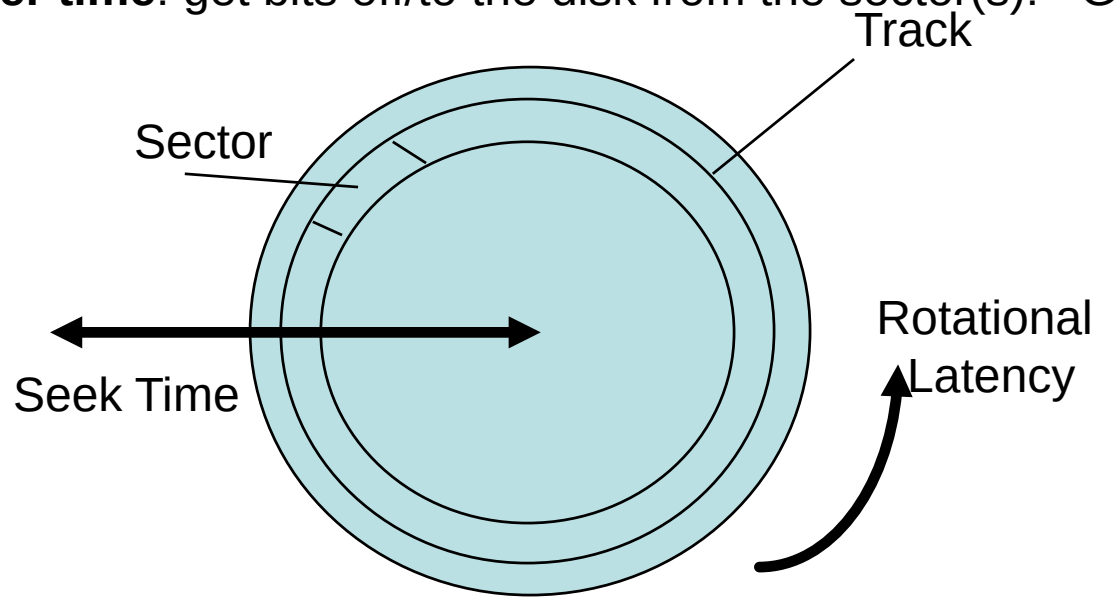


Fig. B

Disk Overheads

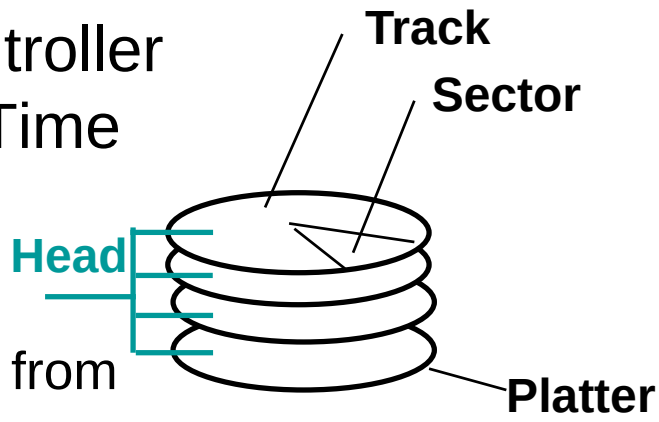
- To read/write from/to disk, we must specify:
  - cylinder #, surface #, sector #, transfer size, memory address
- Transfer time includes:
  - **Seek time**: to get to the track/cylinder
    - Min: < 1ms, Max: 12 ms, Common: 9 ms
  - **Rotational Latency time**: to get to the sector and
    - Spin rates of 5.4/7.2/10/15K RPM → 11 to 4 ms per revolution
    - An average latency to the desired sector is halfway around the disk
  - **Transfer time**: get bits off/to the disk from the sector(s): ~Gbps

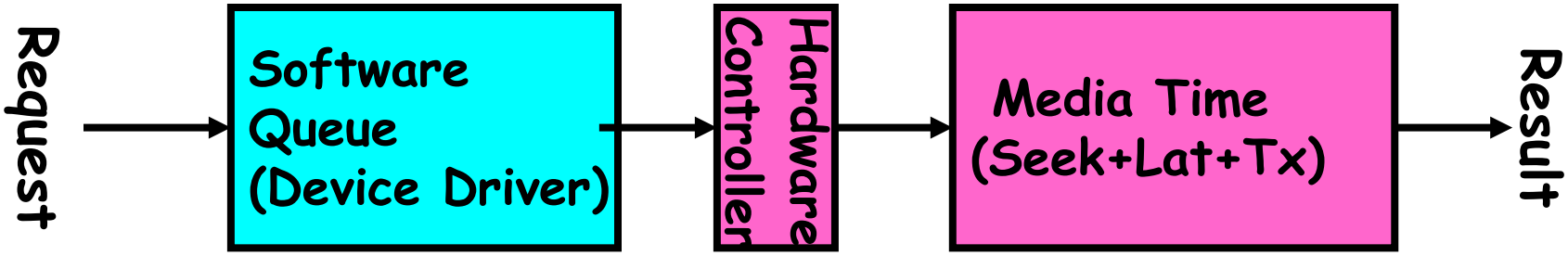




Disk Latency

- **Disk Latency** = Queueing Time + Controller time + Seek Time + Rotation Latency Time + Transfer Time
- Highest Bandwidth?
  - Transfer large group of blocks sequentially from one track/cylinder





Disk Performance

- Assumptions:
  - Ignoring queuing and controller times for now
  - Avg seek time of 5ms, avg rotational latency of 4ms
  - Transfer rate of 4MByte/s, sector size of 1 KByte
- Random place on disk:
  - Seek (5ms) + Rot. Latency (4ms) + Transfer (0.25ms)
  - Roughly 10ms to fetch/put data: 100 KByte/sec
- Random place in same cylinder:
  - Rot. Latency (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 latency → disk scheduling

END of PART-I

Disk & its structure

PART-II

Disk Scheduling

Disk Formatting

Disk Scheduling

- The operating system tries to use hardware efficiently
  - for disk drives $\Rightarrow$ having fast access time, disk bandwidth
- Access time has two major components
  - *Seek time* is time to move the heads to the cylinder containing the desired sector #
  - *Rotational latency* is additional time waiting to rotate the desired sector # to the disk head.
- Minimize seek time
- Seek time $\approx$ seek distance
- Disk bandwidth is total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling (Cont.)

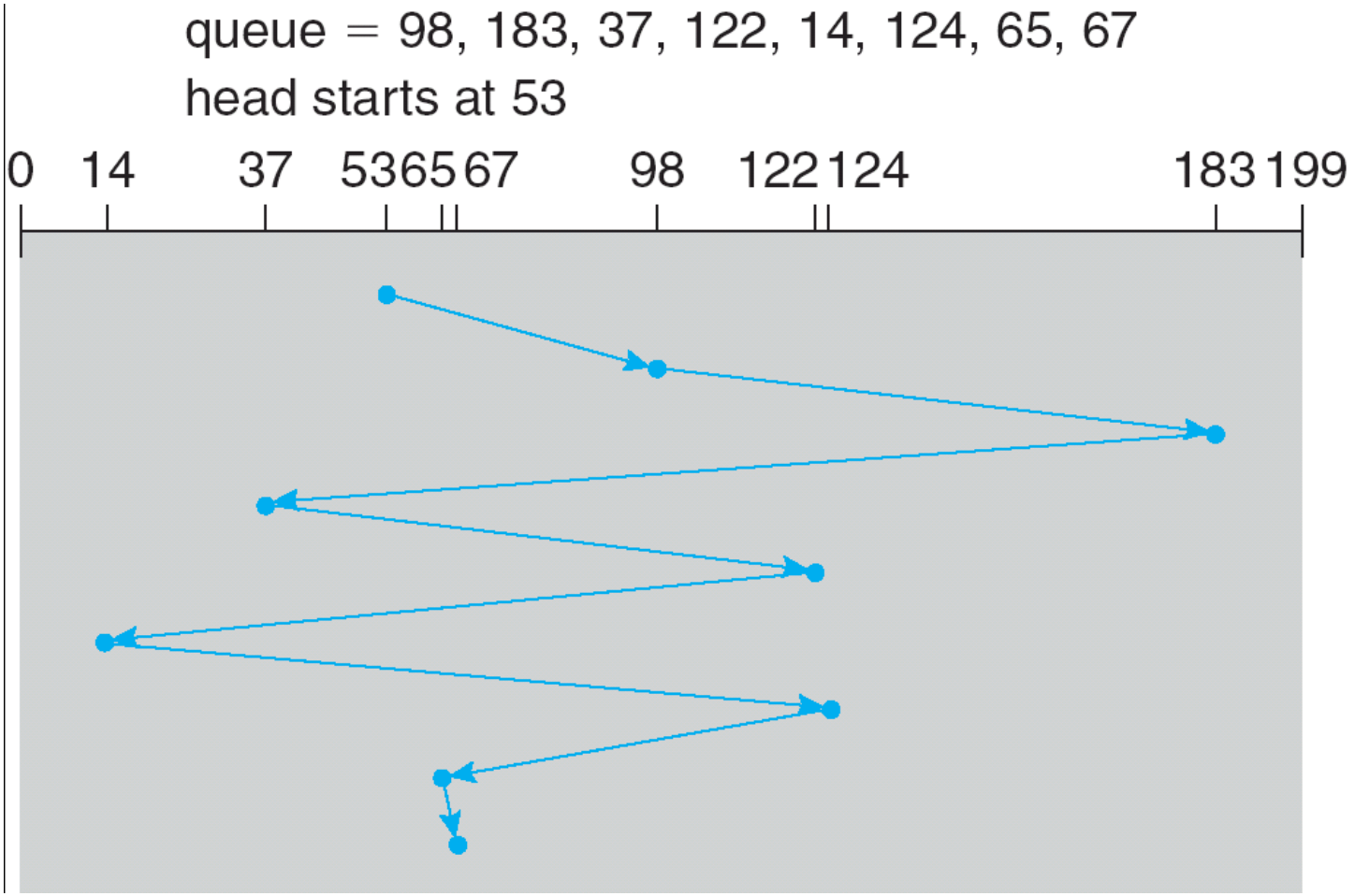
- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several scheduling algos exist for serving disk I/O requests.
- The following discussion is true for one or many platters
- We illustrate them with a request queue of tracks (0-199) on a single platter.

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

Head pointer is at track #53

FCFS

Illustration shows total head movement of 640 cylinders.



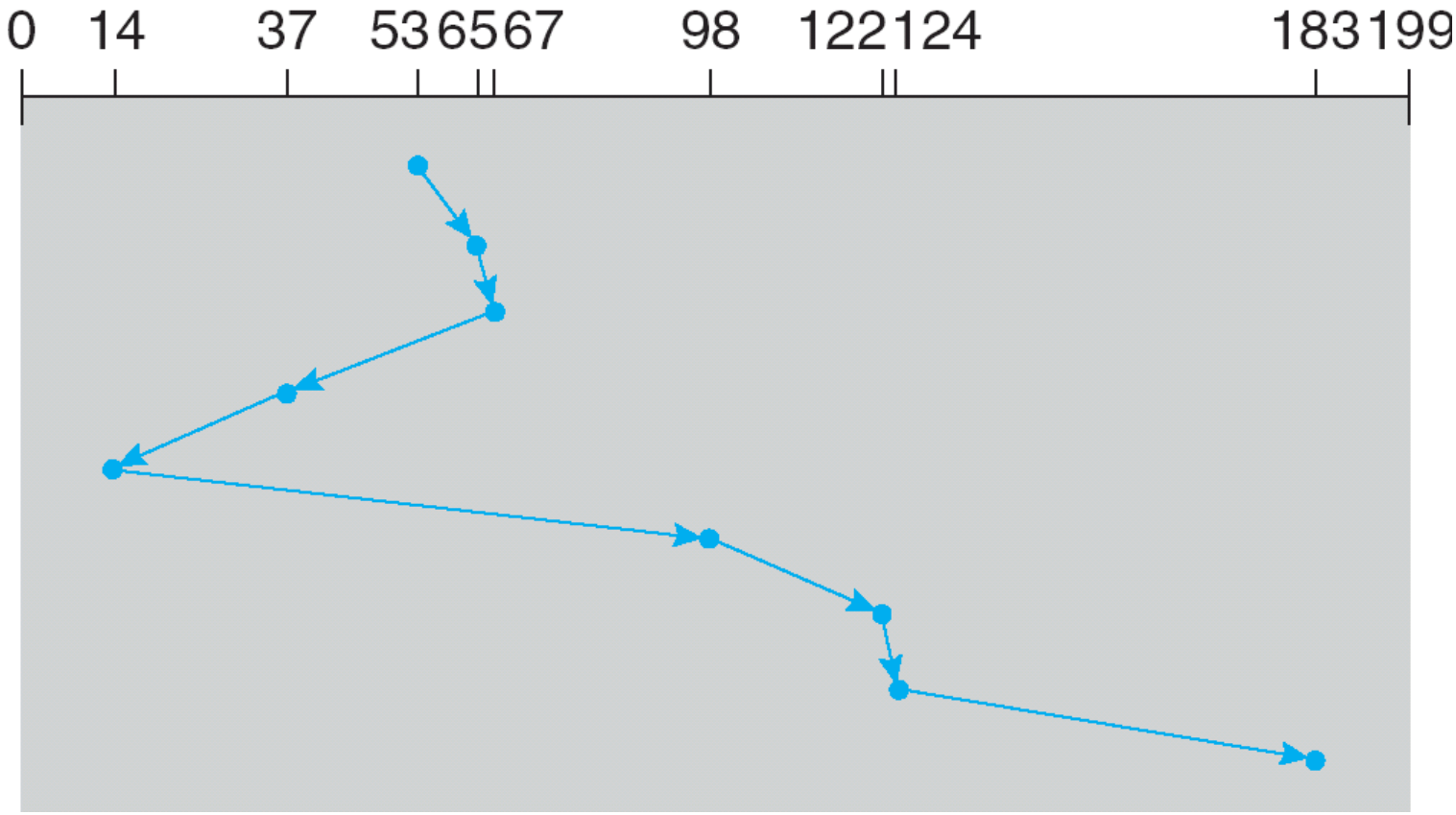
SSTF

- Selects request with minimum seek time from current head position
  - It’s not optimal
- 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

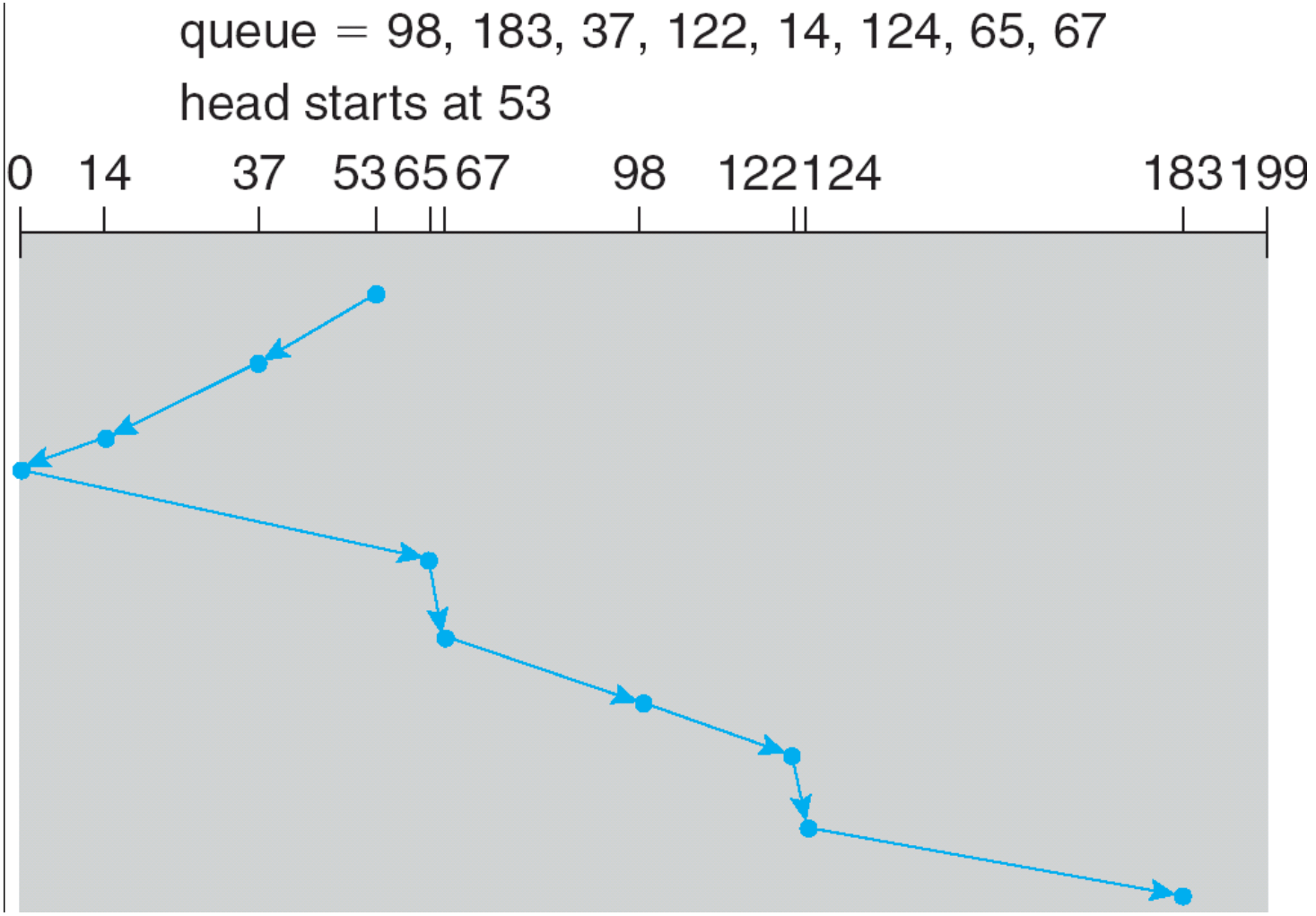
head starts at 53



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 236 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

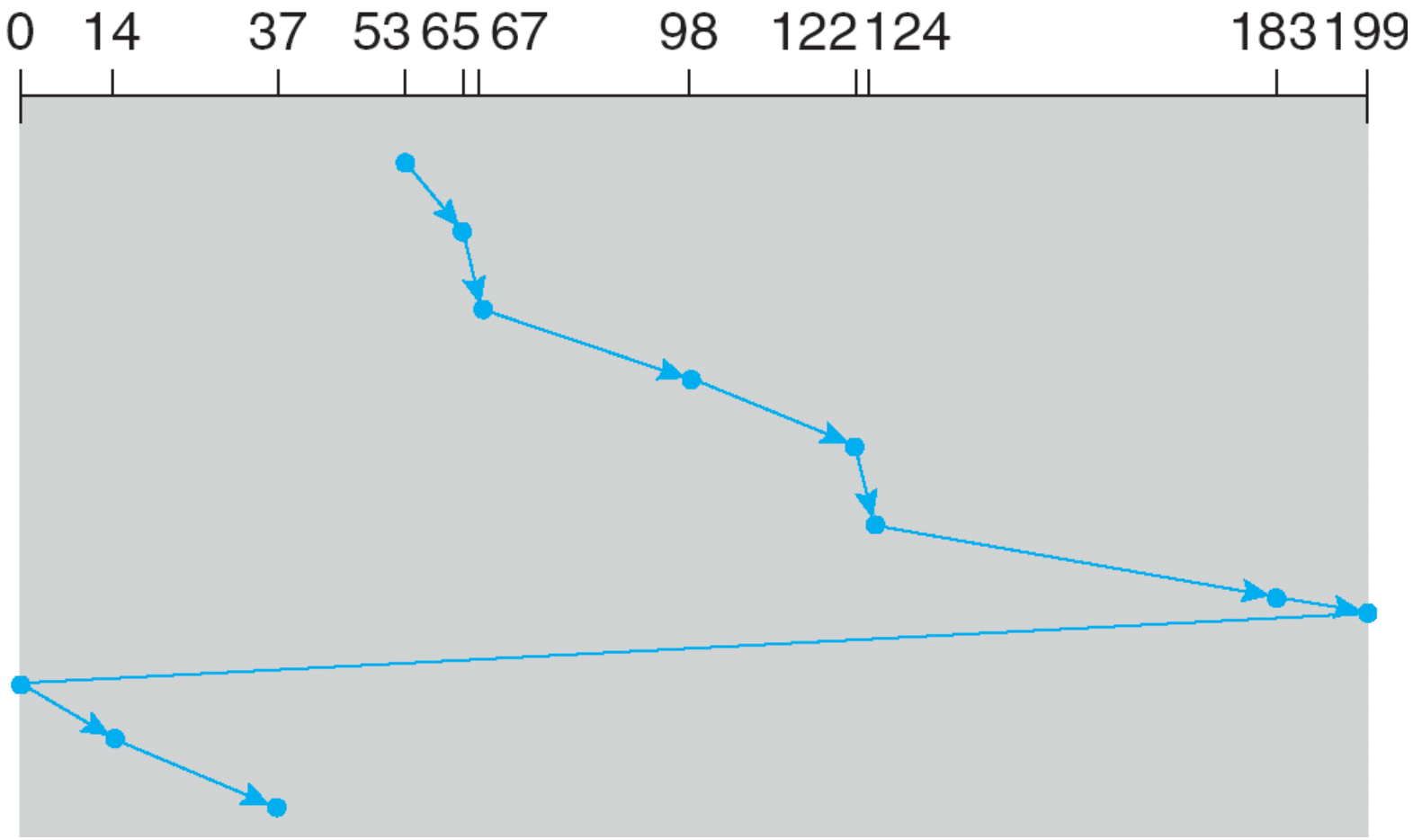
C-SCAN

- Provides a more uniform wait time than SCAN.
  - Better fairness
- 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



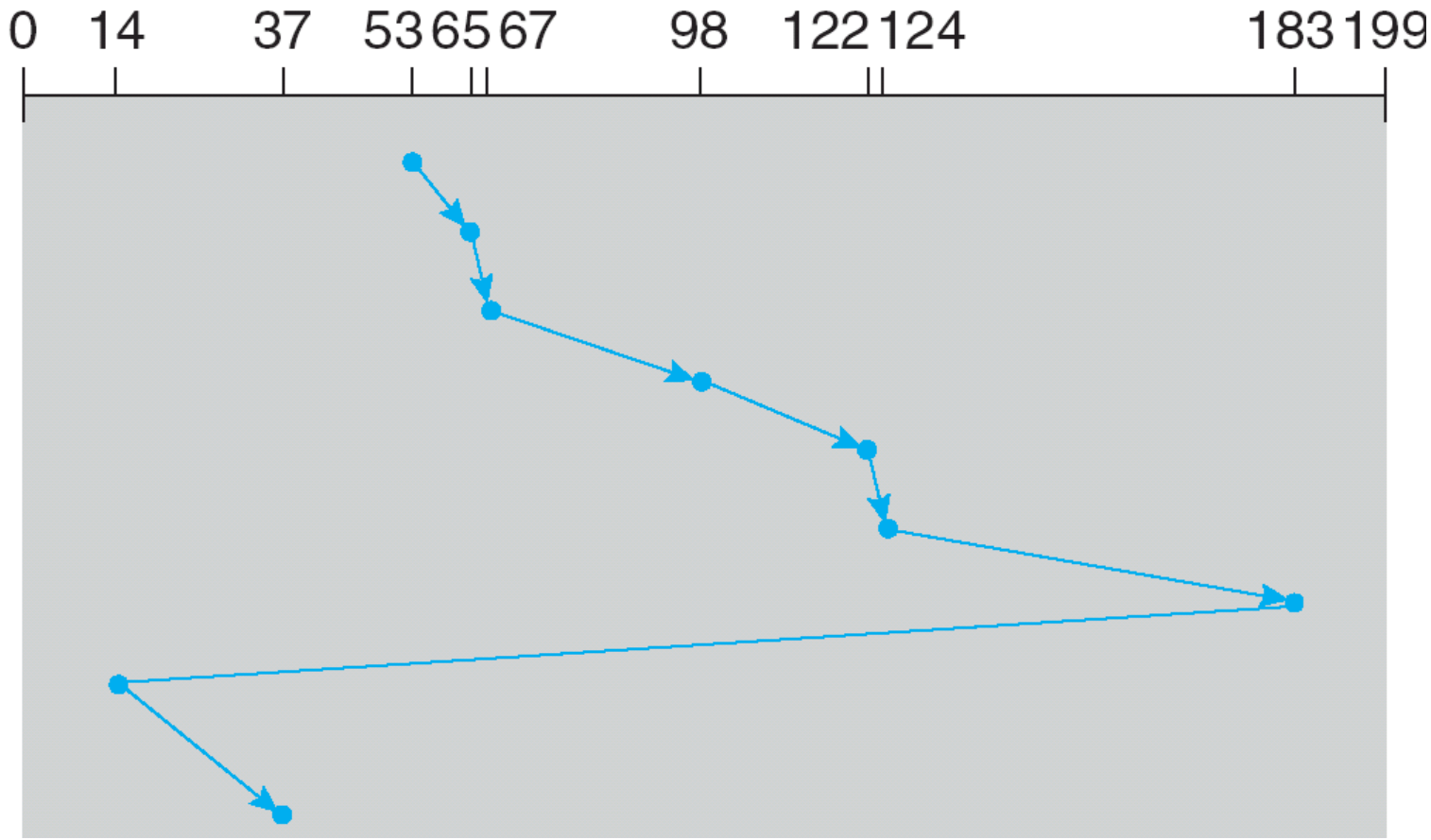
C-LOOK

- 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 Disk Scheduling 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 algo 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 algo
- Linux maintains separate read and write queues, gives read high priority
  - Because processes more likely to block on read than write
  - Deadline scheduler with a combination of FCFS & C-SCAN

How is the disk formatted?

- After manufacturing disk has no information
  - Is stack of platters coated with magnetizable metal oxide
- Before use, each platter receives low-level format
  - Format has series of concentric tracks
  - Each track contains some sectors

- Preamble allows h/w to recognize start of sector
  - Also contains cylinder and sector numbers
  - Data is usually 512 bytes
  - ECC field used to detect and recover from a few no. of bit errors during read operations

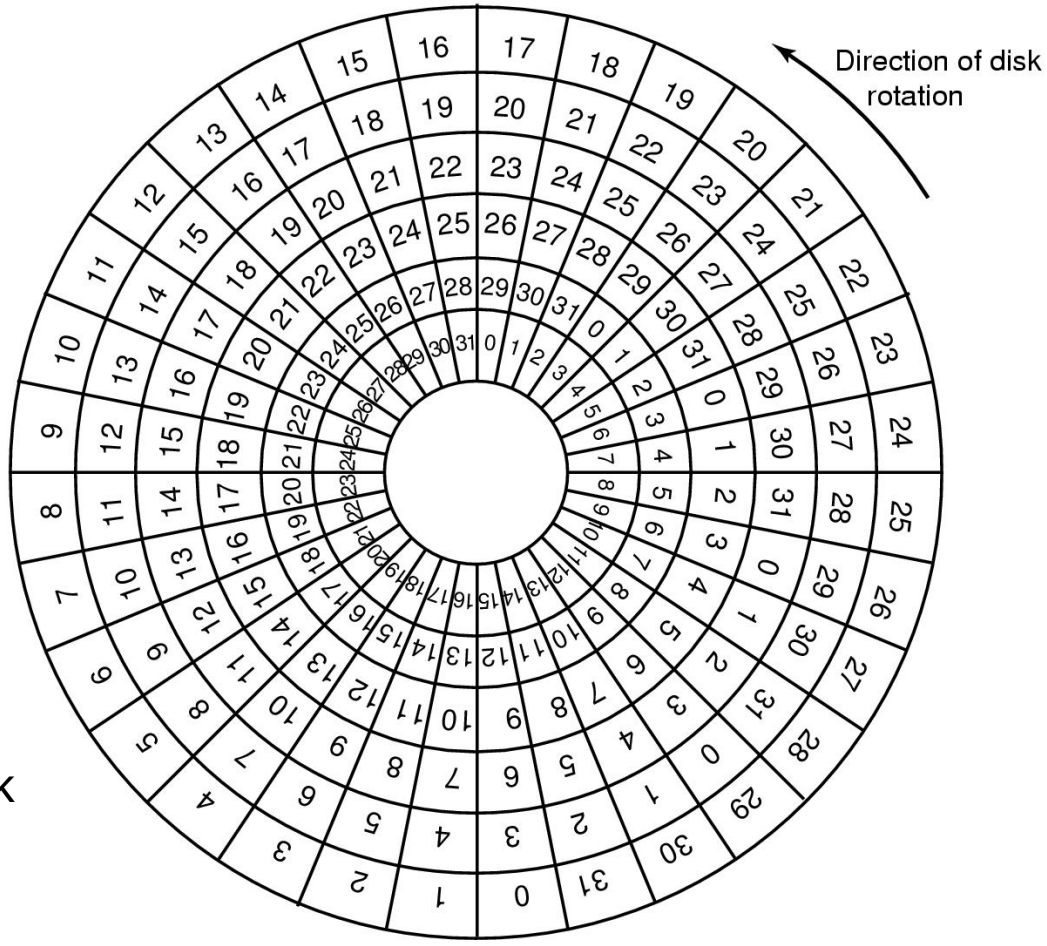
Cylinder Skew

- Why cylinder skew?

- How much skew?
- Example, if
  - 10000 rpm
    - Drive rotates in 6 ms
  - Track has 300 sectors
    - New sector every 20 µs
  - If track seek time 800 µs

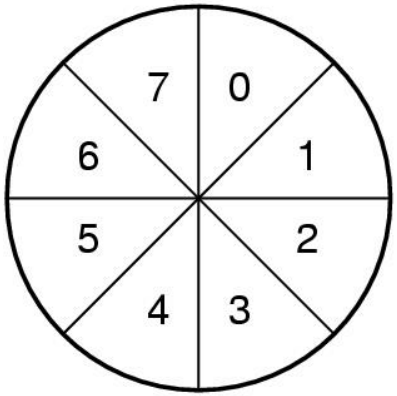
    $\Rightarrow$40 sectors pass on seek

Cylinder skew: 40 sectors

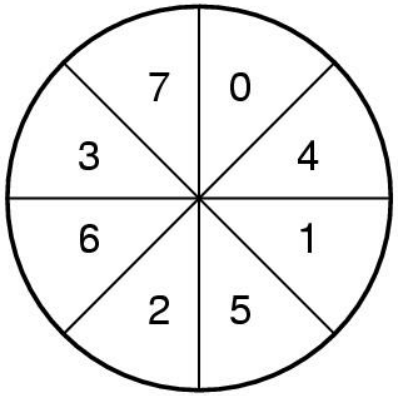


Formatting and Performance

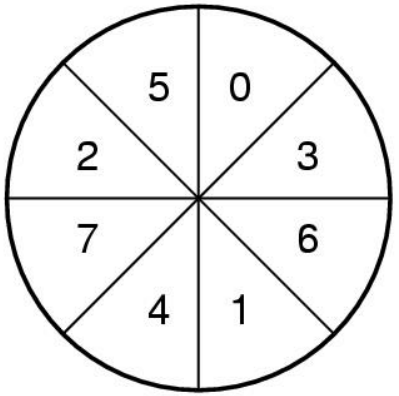
- If 10K RPM, 300 sectors of 512 bytes per track
  - 153,600 bytes every 6 ms $\Rightarrow$ 24.4 MB/sec transfer rate
- If disk controller buffer can store only one sector
  - For 2 consecutive reads, 2nd sector flies past during memory transfer of 1st track
  - Idea: Use single/double interleaving



(a)



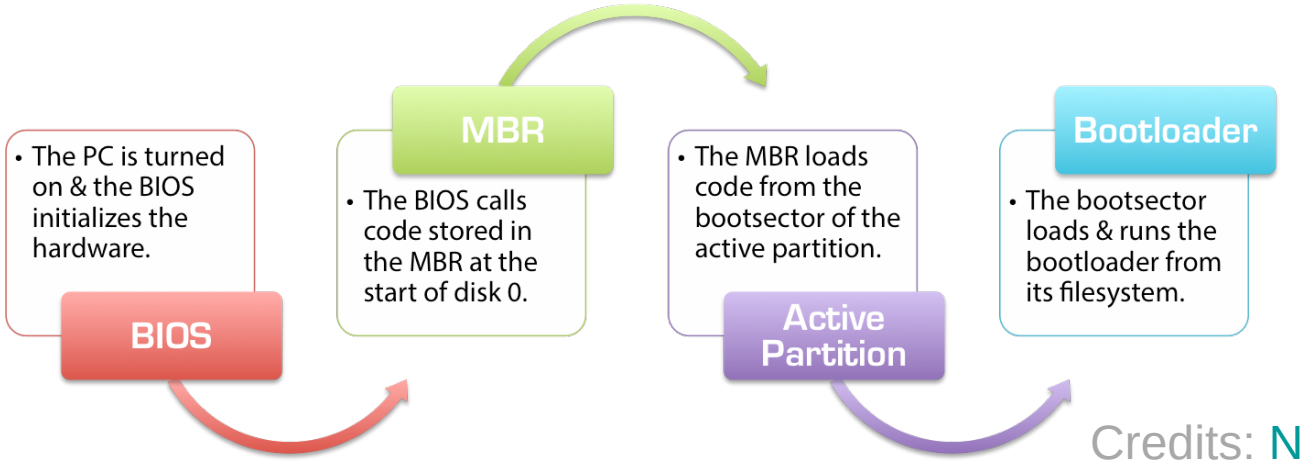
(b)



(c)

Disk Partitioning

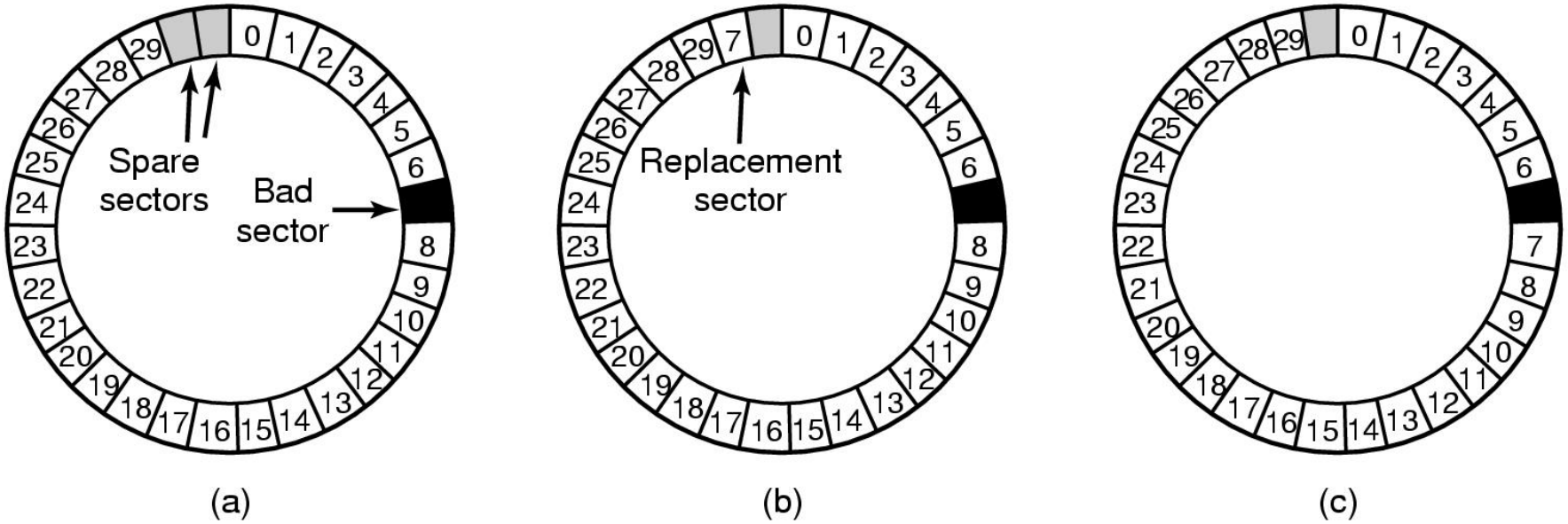
- Each partition is like a separate disk
- Sector 0 is MBR
  - Contains boot code + partition table
  - Partition table has starting sector and size of each partition
- High-level formatting
  - Done for each partition
  - Specifies boot block, free list, root directory, empty file system
- What happens on boot?
  - BIOS loads MBR, boot program checks to see active partition
  - Reads boot sector from that partition that then loads OS kernel, etc.



Credits: NeoSmart Tech

Handling Errors

- A disk track with a bad sector
- Solutions:
  - Substitute a spare for the bad sector (sector sparing)
  - Shift all sectors to bypass bad one (sector forwarding)



Nonvolatile Memory Devices

- If disk-drive like, then called **solid-state disks** (**SSD**s)
- Other forms include **USB drives** (thumb drive, flash drive), and main storage in devices like smartphones
- Can be more reliable than HDDs
- More expensive per MB
- Can only be erased a limited number of times before worn out: ~ 100,000
  - So maybe having shorter life span
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

END of PART-II

Disk Scheduling

Disk Formatting

PART-III

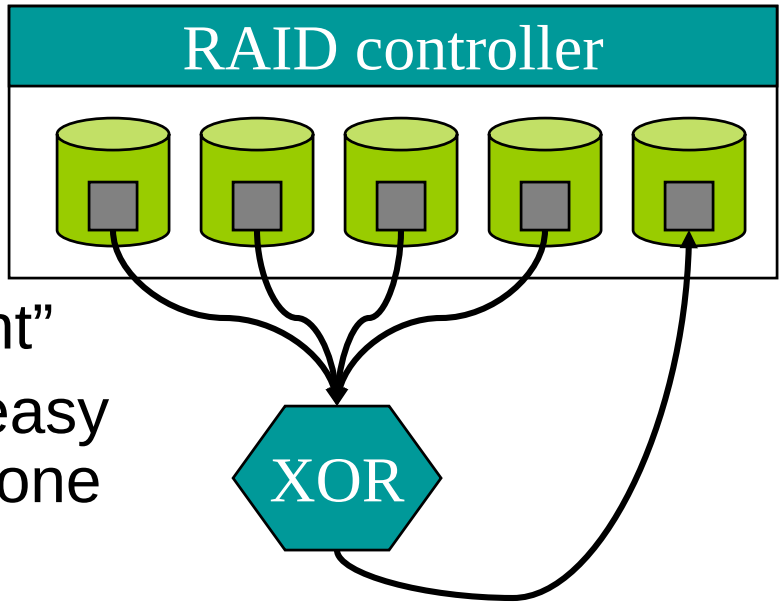
RAID

RAID: Motivation

- Disks are improving, but not as fast as CPUs
  - 1970s seek time: 50-100 ms
  - 2000s seek time: <5 ms
  - Factor of 20 improvement in 3 decades in disk vs 1000's fold increase in CPU performance
- We can use multiple disks for improving performance
  - By Striping files across multiple disks (placing parts of each file on a different disk), parallel I/O can improve access time
- But striping reduces reliability
  - Mean time to failure of a single disk, say 100,000 hours
  - 100 disks have 1/100th mean time between failures of one disk☹
- So, we need Striping for performance, but we need something else to help with reliability/availability
  - To improve reliability, we can add redundant disks, in addition to Striping

RAID

- A RAID is a Redundant Array of Inexpensive Disks
  - The alternative is SLED: single large expensive disk
  - In industry, “I” is for “Independent”
- Disks are small and cheap, so it’s easy to put lots of disks (10s to 100s) in one box for increased storage, performance, and availability
- The RAID box with a RAID controller looks just like a SLED to the computer
- Data plus some redundant information is Striped across the disks in some way
  - How that Striping is done is key to performance and reliability



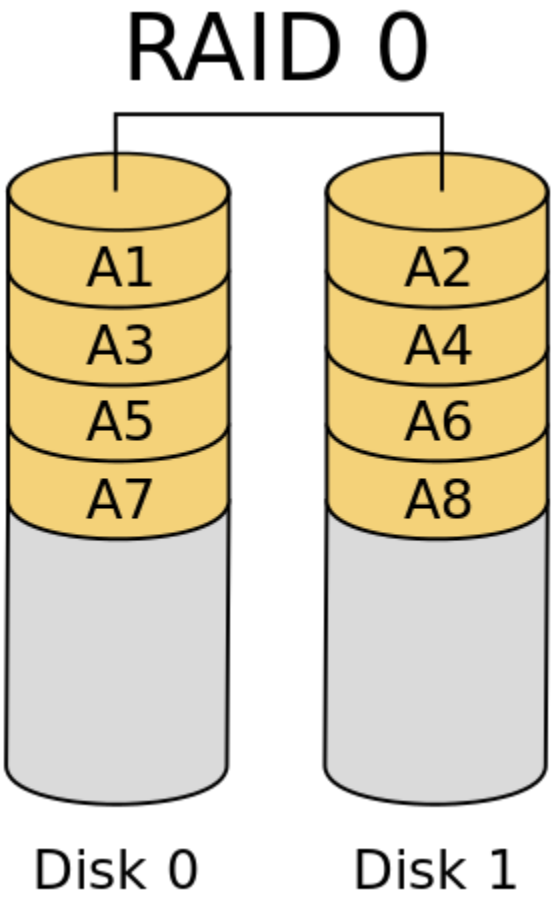
Some RAID Issues

- **Granularity**
  - Fine-grained: Stripe each file over all disks.
    - This gives high throughput for the file, but limits to transfer of 1 file at a time
  - Coarse-grained: Stripe each file over only a few disks.
    - This limits throughput for 1 file but allows more parallel file accesses

- **Redundancy**
  - Concentrate redundancy info on a small number of disks: partition the disk set into data disks and redundant disks
  - Uniformly distribute redundancy info on disks: avoids load imbalancing problems

Raid Level 0

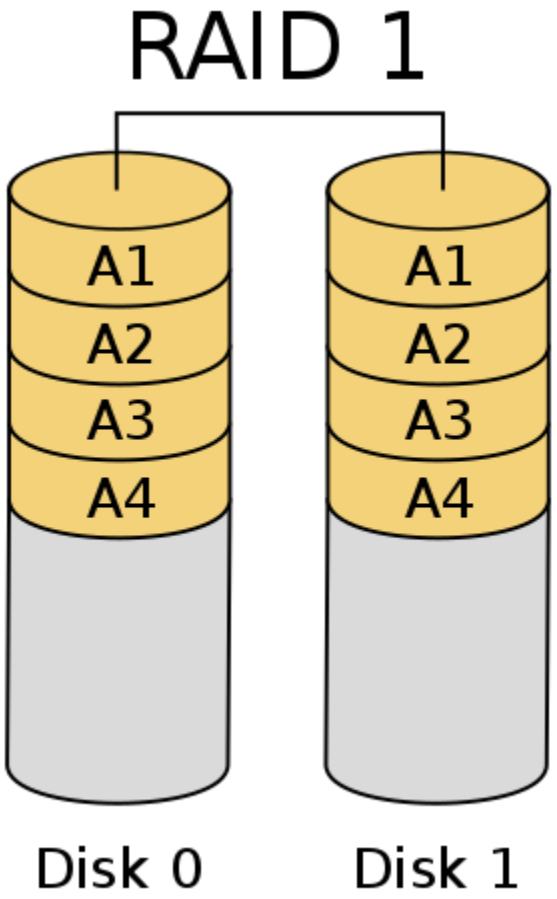
- Level 0 is non-redundant disk array
- Files are Striped across disks, but no redundant info
- Consider a file A with blocks A1-A8
- High read throughput
- Best write throughput (no redundant info to write)
- Any disk failure results in complete data loss
  - Reliability worse than SLED, why?



Credits: Cburnett, Wikipedia

Raid Level 1

- Mirrored Disks
- Data is written to two places
  - On failure, just use surviving disk
- On read, choose fastest to read
  - Write performance is same as single drive, read performance is 2x better
- Expensive



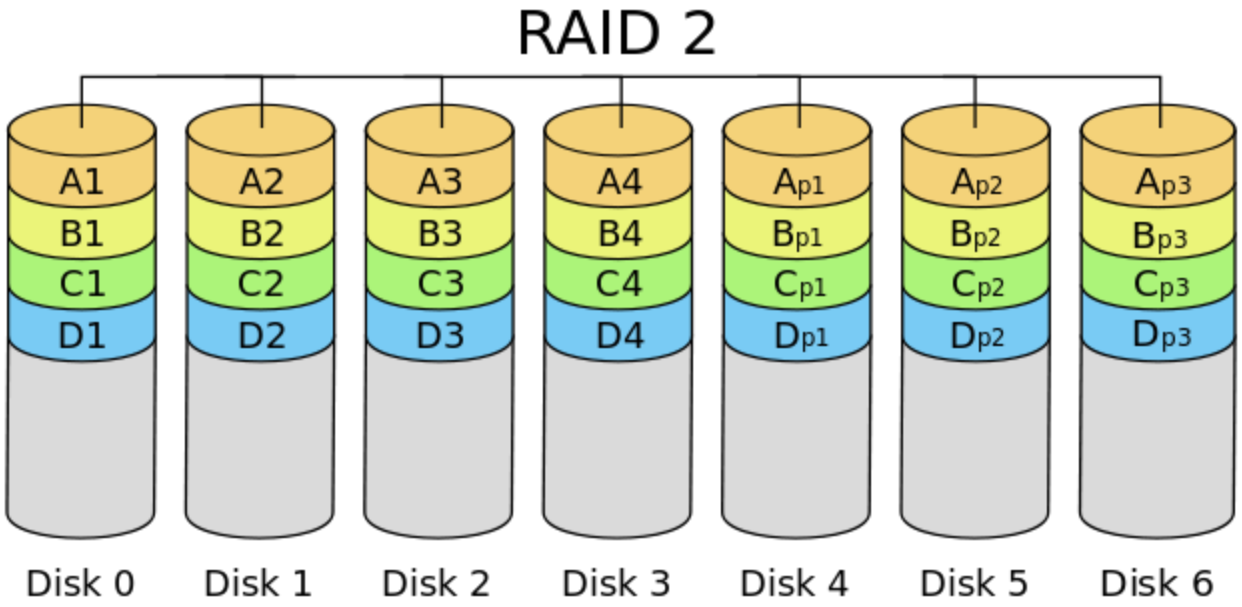
Credits: Cburnett, Wikipedia

Parity Bits

- What do you need to do in order to detect and correct a one-bit error ?
  - Suppose you have a binary number, represented as a collection of bits: <b3, b2, b1, b0>, e.g. 0110
- Detection is easy
- Parity:
  - Count the number of bits that are SET, see if it's odd or even
    - EVEN parity is 0 if the number of 1 bits is even
  - Parity(<b3, b2, b1, b0 >) = P0 = b0 ⊗ b1 ⊗ b2 ⊗ b3
  - Parity(<b3, b2, b1, b0, p0>) = 0 if all bits are intact
  - Parity(0110) = 0, Parity(01100) = 0
  - Parity(11100) = 1 => ERROR!
  - Parity can detect a single error, but can't tell you which of the bits got flipped
  - Detection and correction require more work
  - Hamming codes can be used to detect & correct bit errors

Raid Level 2

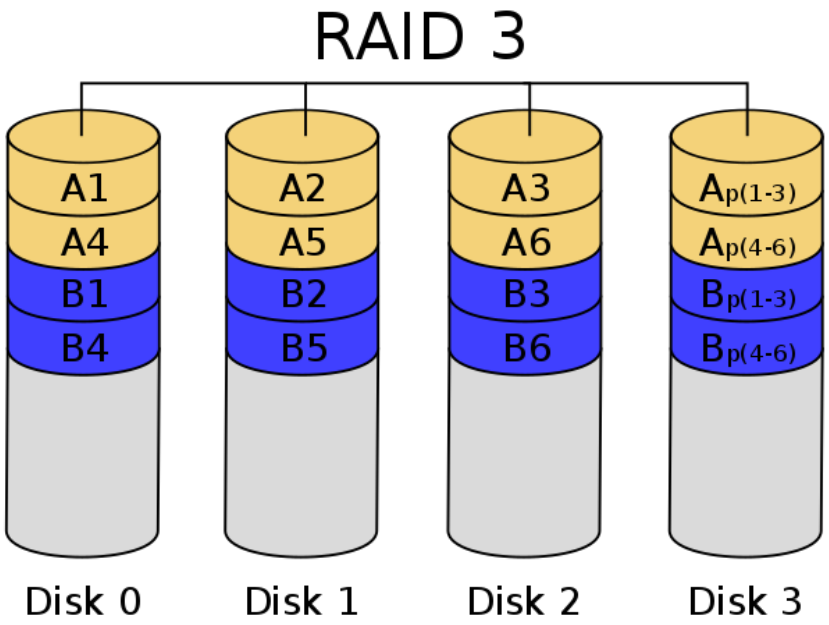
- Bit-level Striping with Hamming (ECC) codes for error correction
- Example: 4 files A-D each with 4 bits
- All 7 disk arms are synchronized and move in unison
- Complicated RAID controller
- Single file access at a time
- Tolerates only one error, but with no performance degradation



Credits: Cburnett, Wikipedia

Raid Level 3

- Use only a single parity disk!
  - Each bit on the parity disk is a parity function of the corresponding bits on all the other disks
- A read accesses all the data disks
- A write accesses all data disks plus the parity disk
- On disk failure, read remaining disks plus parity disk to compute the missing data

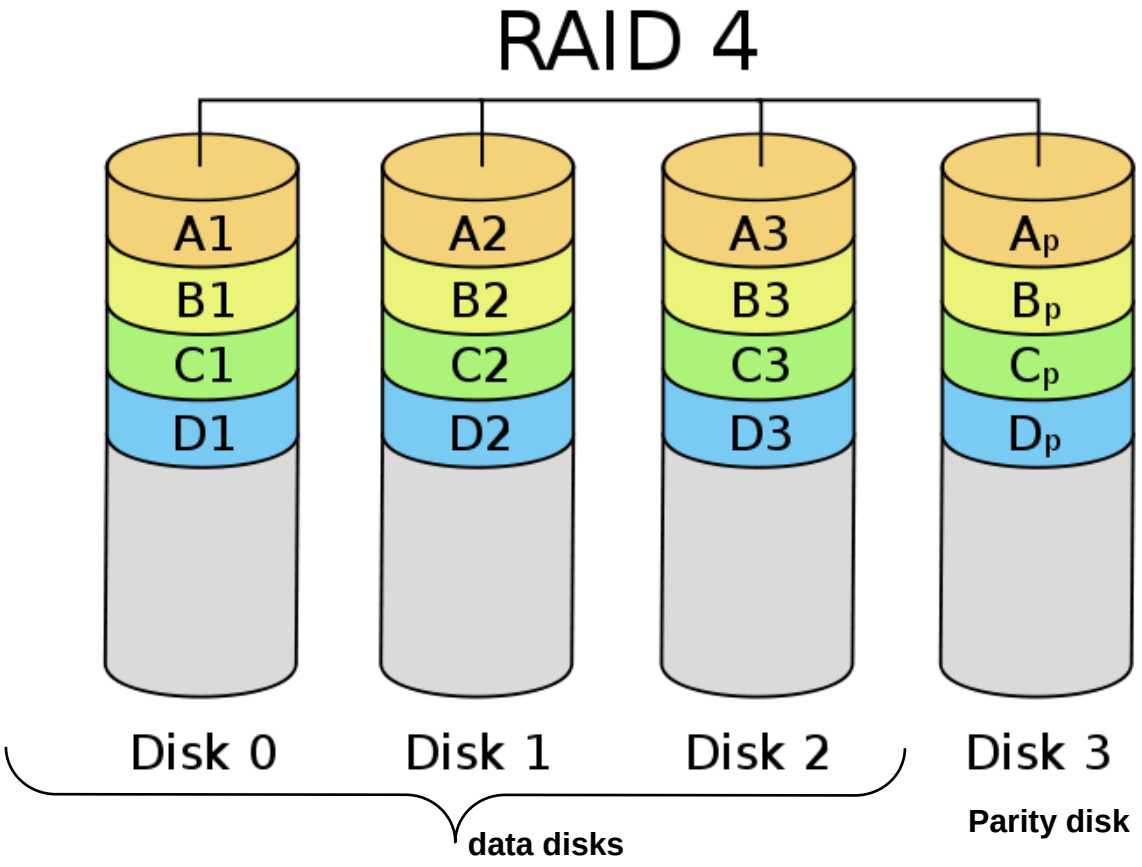


Single parity disk can be used to detect and correct errors, as disk controller knows which bit got flipped!

Credits: Cburnett, Wikipedia

Raid Level 4

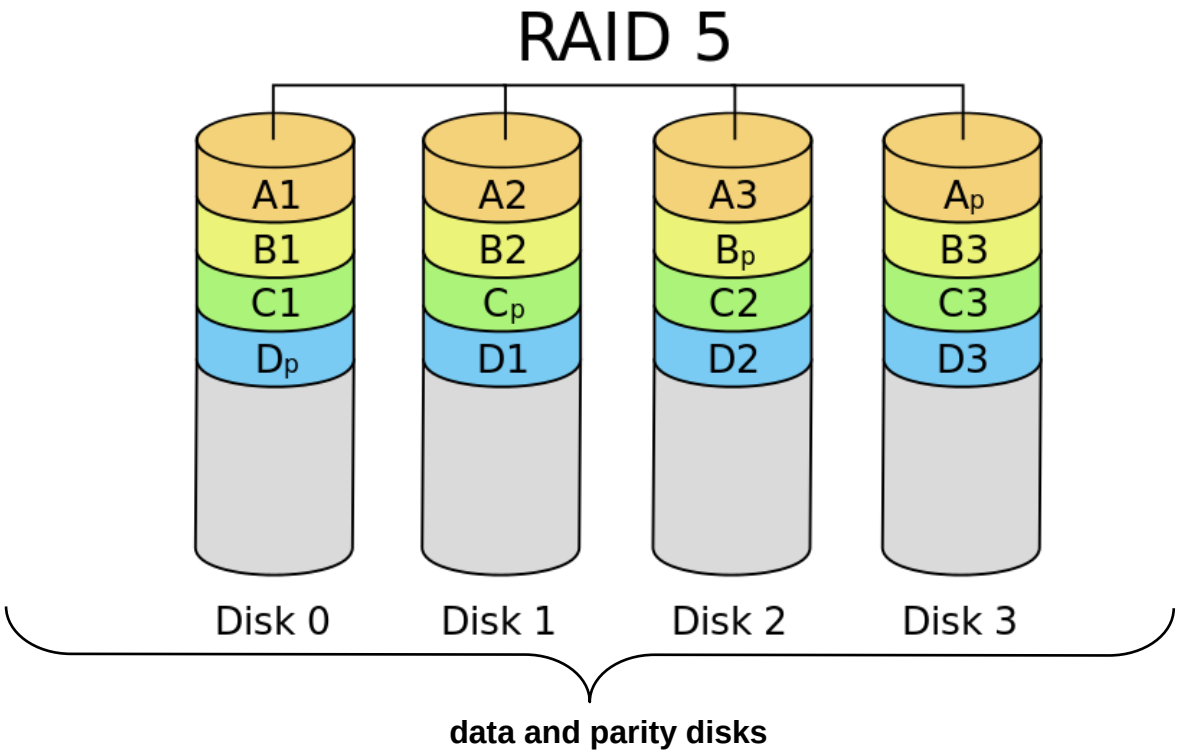
- Combines Level 0 and 3 – block-level parity with Stripes
- A read accesses all the data disks
- A write accesses all data disks plus the parity disk
- Heavy load on the parity disk!



Credits: Cburnett, Wikipedia

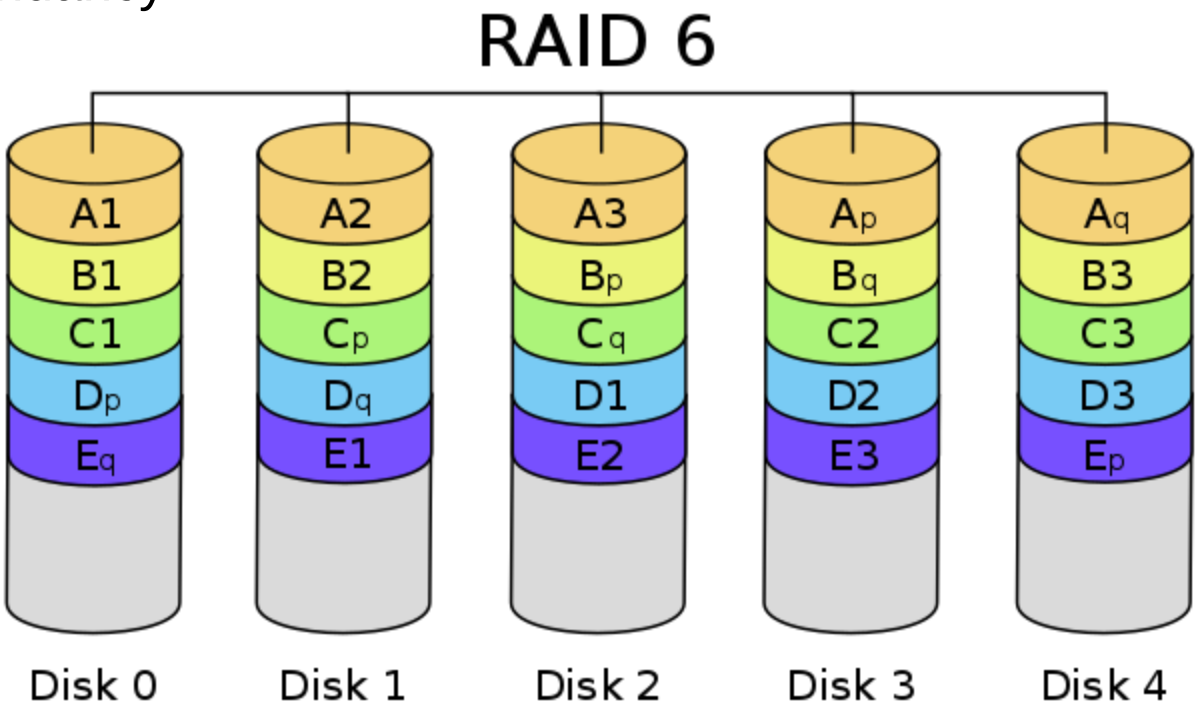
Raid Level 5

- Block Interleaved Distributed Parity
- Like parity scheme, but distribute the parity info over all disks (as well as data over all disks)
- Better read performance, large write performance
  - Reads can outperform SLEDs and RAID-0

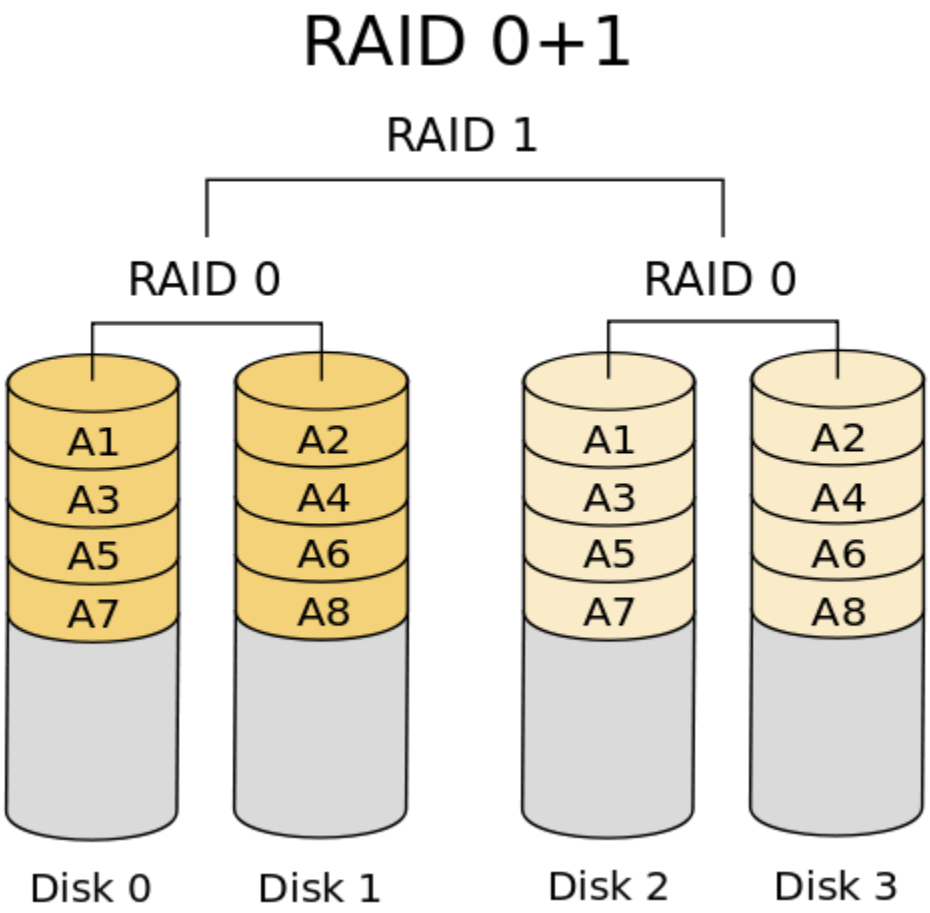


Raid Level 6

- Level 5 with an extra parity bit in an extra disk
- Can tolerate two disk failures
  - What are the odds of having two concurrent failures ?
- May outperform Level-5 on reads, slower on writes
- **Block interleaved parity** (**RAID 4, 5, 6**) uses much less redundancy



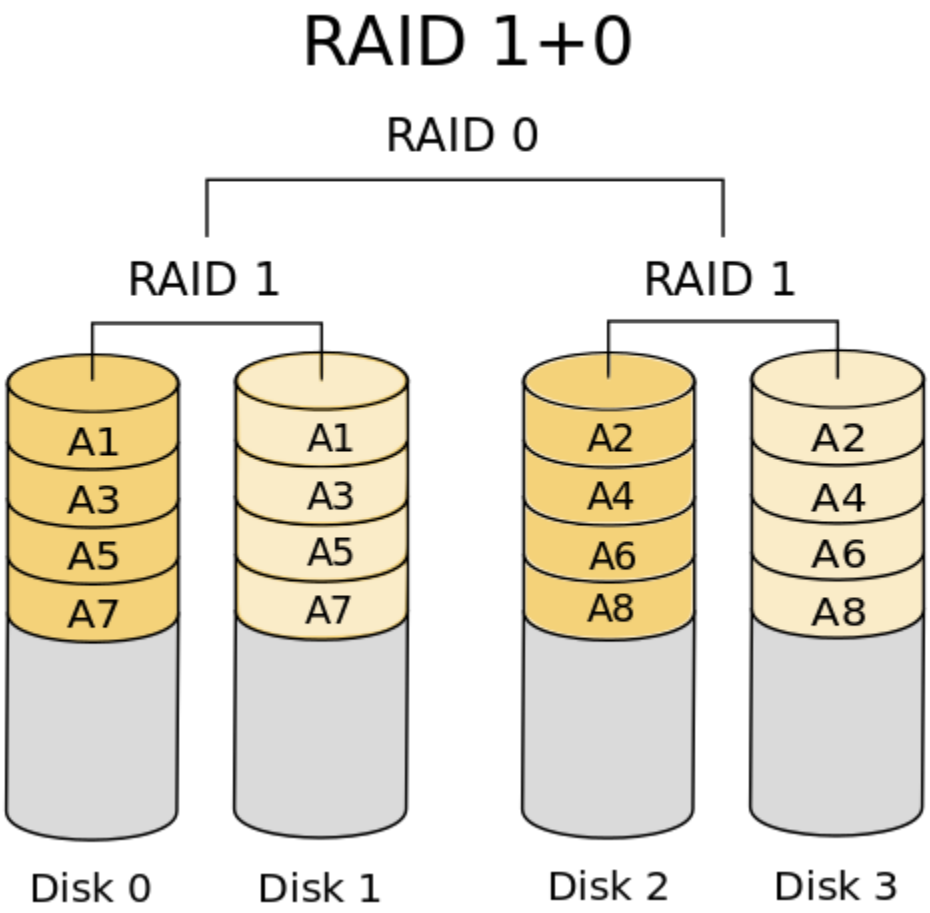
Nested RAIDs: RAID 0+1



Striped mirrors

Credits: Cburnett, Wikipedia

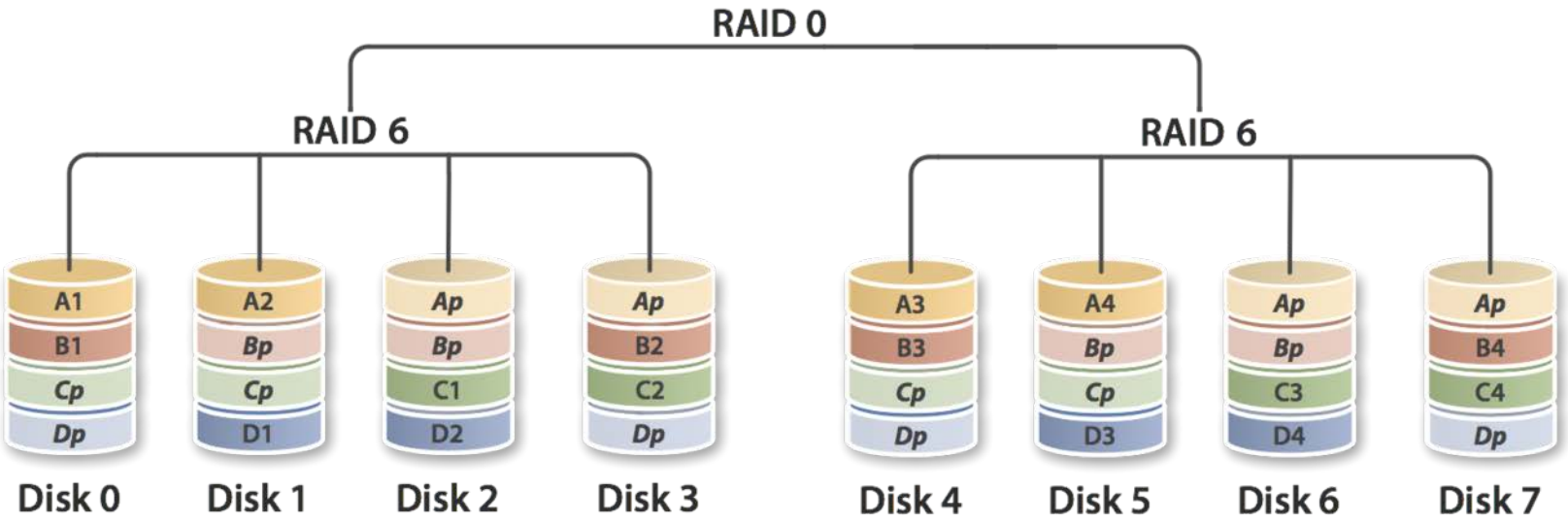
Nested RAIDs: RAID 1+0



Mirrored Stripes

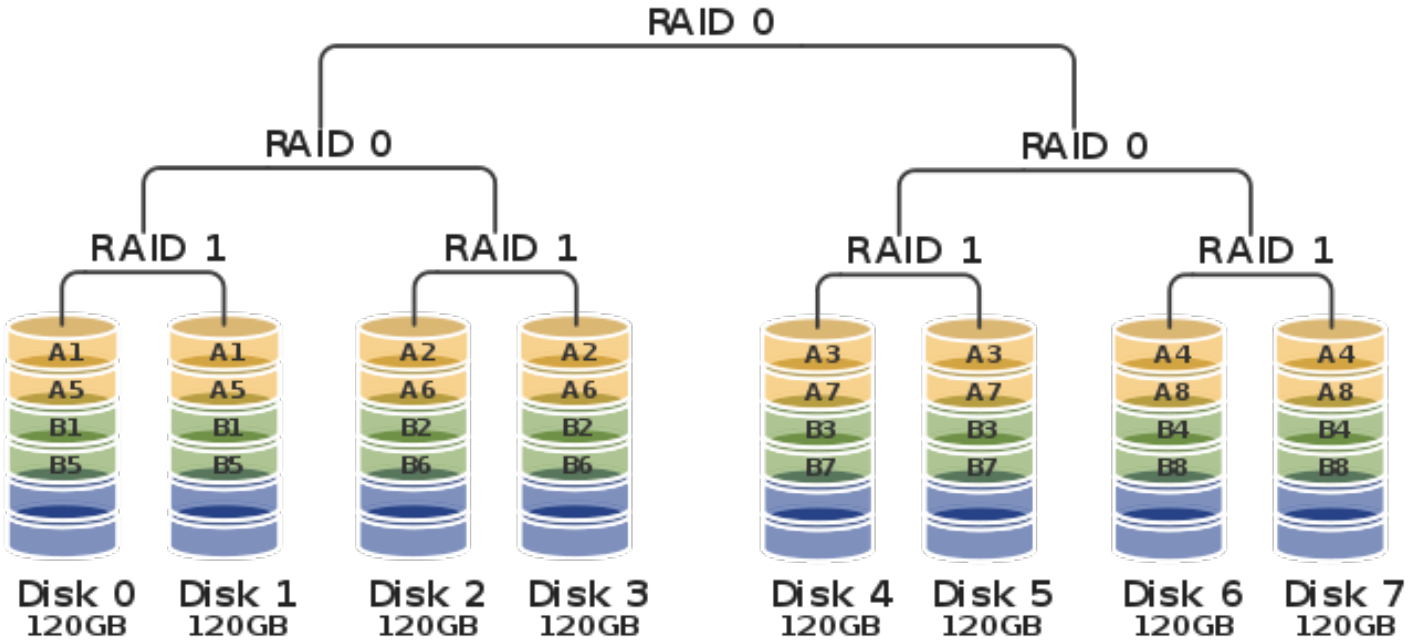
Credits: Cburnett, Wikipedia

Nested RAIDs: RAID 6+0



Credits: Cburnett, Wikipedia

Nested RAIDs: RAID 1+0+0



Generally implemented using software RAID 0 over hardware RAID 1+0

Credits: Cburnett, Wikipedia

Summary

- Disks
  - Latency Seek + Rotational + Transfer
    - Also, queuing time
  - Rotational latency: on average ½ rotation
- SSDs are faster but expensive
- Improve performance via intelligent scheduling
- Many different RAID levels offering varied degree of performance and reliability
- RAID within a storage array can still fail if the array fails, so automatic replication of the data between arrays is common
  - Frequently, a small number of hot-spare disks are left unallocated, automatically replacing a failed disk and having data rebuilt onto them

References

- Chapter 10 in Galvin et al 9th Edition
- https://en.wikipedia.org/wiki/Standard_RAID_levels
- https://en.wikipedia.org/wiki/Nested_RAID_levels

Parity and Hamming Code

- Detection and correction require more work
- Hamming codes can detect double bit errors and detect & correct single bit errors
- 7/4 Hamming Code
  - h0 = b0 ⊗ b1 ⊗ b3
  - h1 = b0 ⊗ b2 ⊗ b3
  - h2 = b1 ⊗ b2 ⊗ b3
  - H0(<1101>) = 0
  - H1(<1101>) = 1
  - H2(<1101>) = 0
  - Hamming(<1101>) = <b3, b2, b1, h2, b0, h1, h0> = <1100110>
  - If a bit is flipped, e.g. <1110110>
  - Hamming(<1111>) = <h2, h1, h0> = <111> compared to <010>, <101> are in error. Error occurred in bit 5.