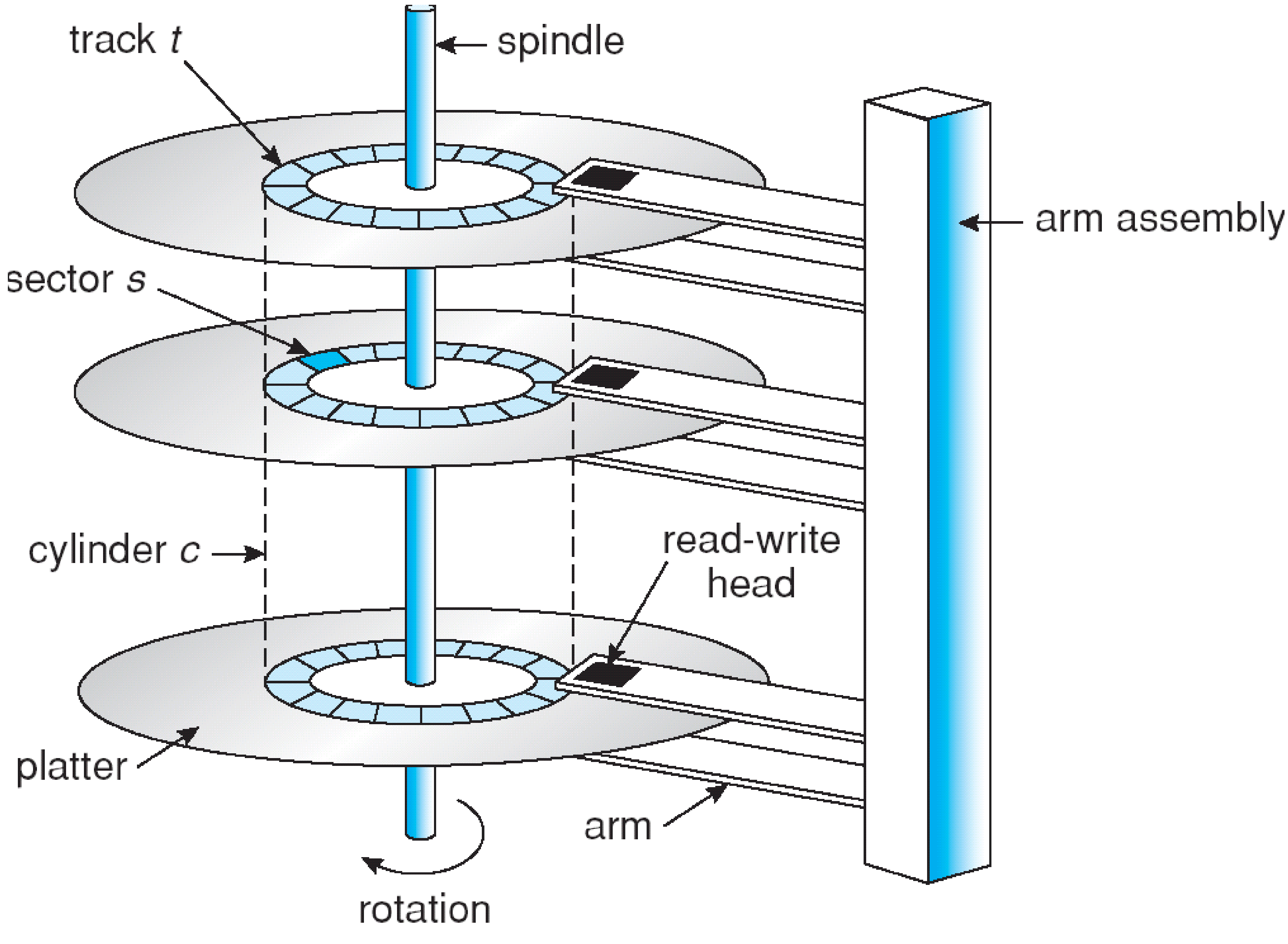
# Disks and RAID

# 50 Years Old!


- 13th September 1956
- The IBM RAMAC 350


- 80000 times more data on the 8GB 1-inch drive in his right hand than on the 24-inch RAMAC one in his left…

# What does the disk look like?



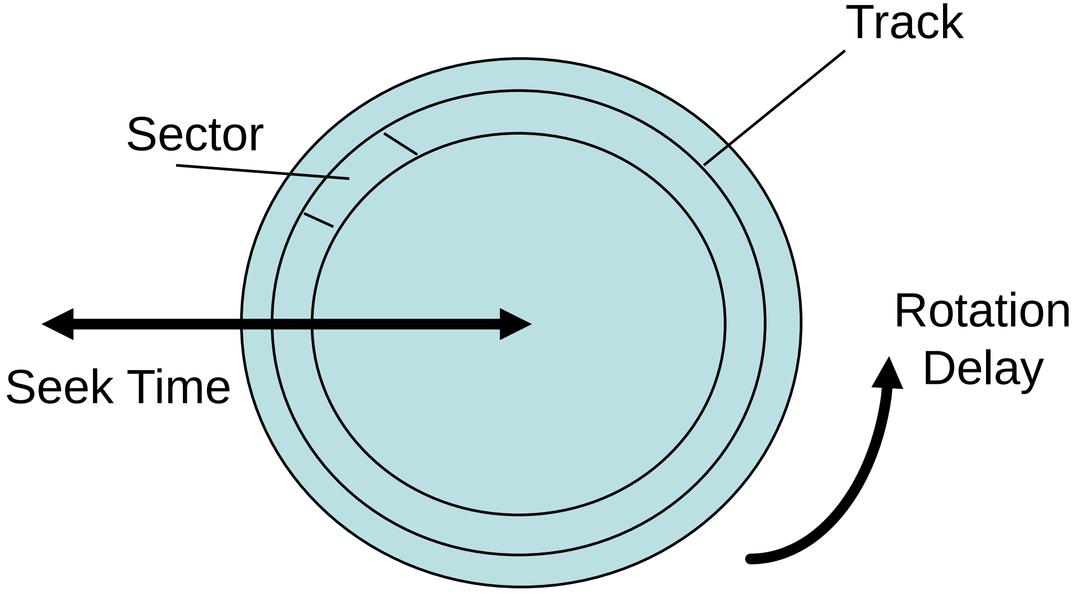
# Some parameters

- 2-30 heads (platters * 2)
  - diameter 14'' to 2.5''
- 700-20480 tracks per surface
- 16-1600 sectors per track
- sector size:
  - 64-8k bytes
  - 512 for most PCs
  - note: inter-sector gaps
- capacity: 20M-100G

- main adjectives: BIG, slow

# Disk overheads

- To read from disk, we must specify:
  - cylinder #, surface #, sector #, transfer size, memory address
- Transfer time includes:
  - Seek time: to get to the track
  - Latency time: to get to the sector and
  - Transfer time: get bits off the disk



# Modern disks

| | **Barracuda 180** | **Cheetah X15 36LP** |
|---|---|---|
| Capacity | 181GB | 36.7GB |
| Disk/Heads | 12/24 | 4/8 |
| Cylinders | 24,247 | 18,479 |
| Sectors/track | ~609 | ~485 |
| Speed | 7200RPM | 15000RPM |
| Latency (ms) | 4.17 | 2.0 |
| Avg seek (ms) | 7.4/8.2 | 3.6/4.2 |
| Track-2- | 0.8/1.1 | 0.3/0.4 |

# Disks vs. Memory

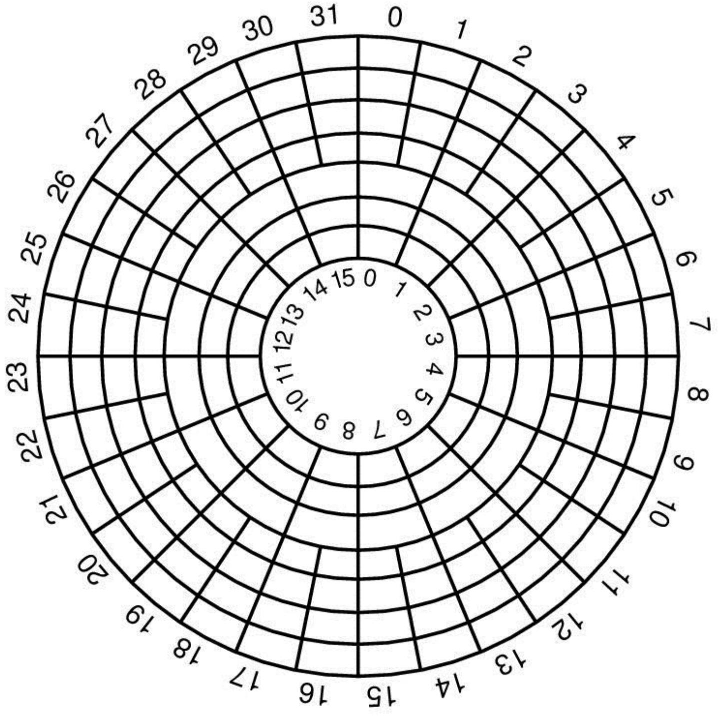
| | |
|---|---|
| • Smallest write: sector | • (usually) bytes |
| • Atomic write = sector | • byte, word |
| • Random access: 5ms<br>– not on a good curve | • 50 ns<br>– faster all the time |
| • Sequential access: 200MB/s | • 200-1000MB/s |
| • Cost $.002MB | • $.10MB |
| • Crash: doesn't matter ("non-volatile")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

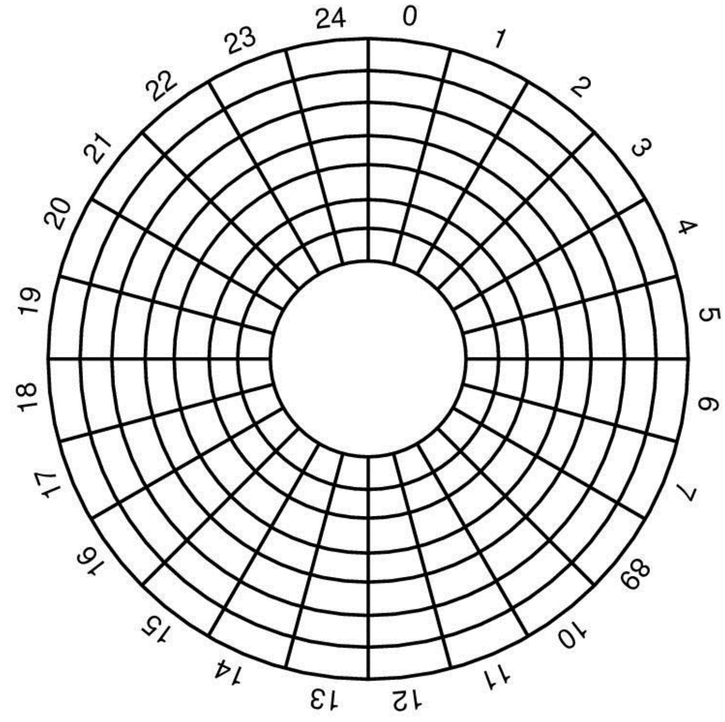
# Disk Structure

- Disk drives addressed as 1-dim arrays of *logical blocks*
  - the logical block is the smallest unit of transfer
- This array mapped sequentially onto disk sectors
  - Address 0 is $1^{st}$ sector of $1^{st}$ 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

# Non-uniform #sectors / track

- Reduce bit density per track for outer layers (Constant Linear Velocity, typically  HDDs)
- Have more sectors per track on the outer layers, and increase rotational speed when reading from outer tracks (Constant Angular Velcity, typically CDs, DVDs)



# 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.)

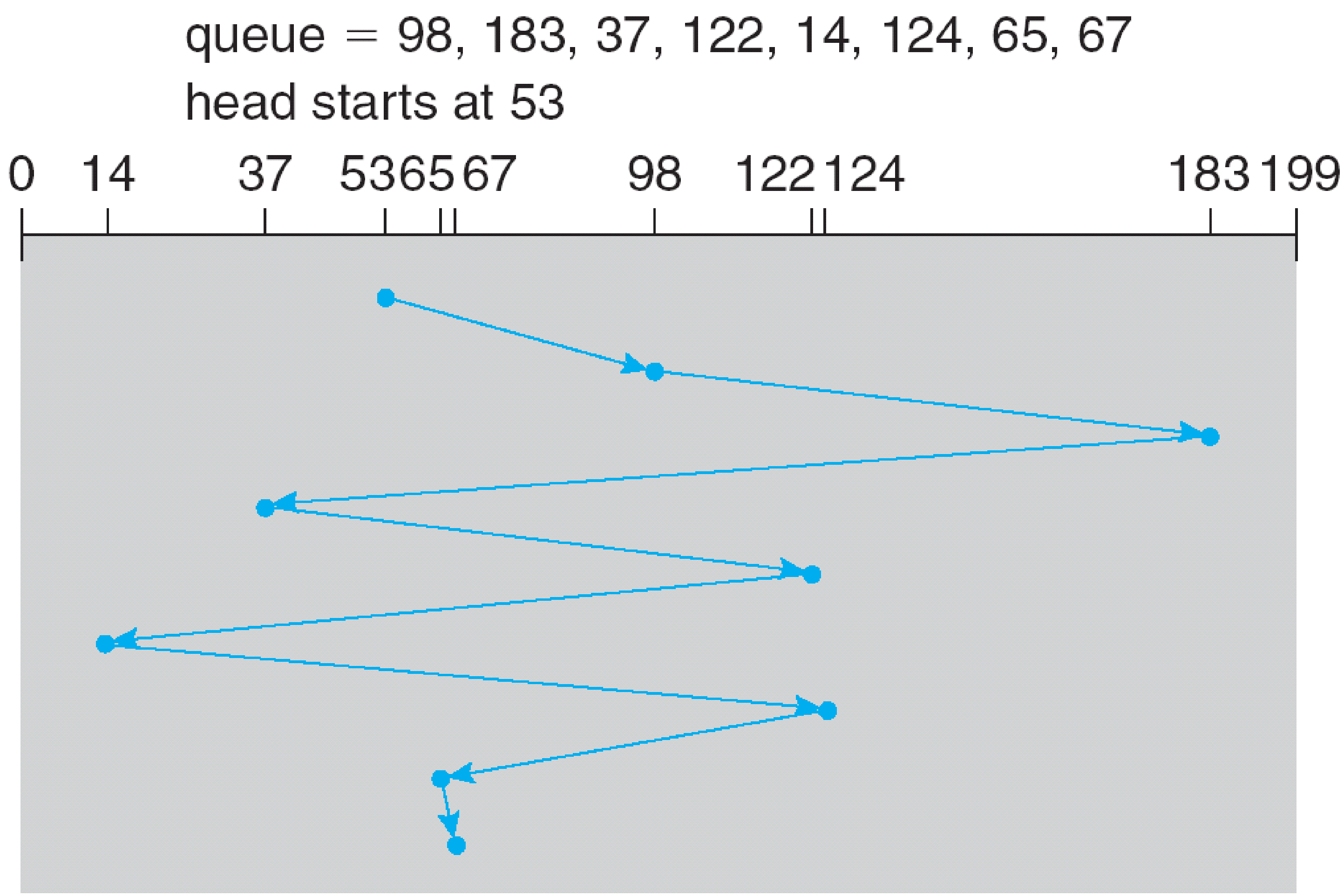
- Several scheduling algos exist service disk I/O requests.
- We illustrate them with a request queue (0-199).

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

Head pointer 53

# FCFS

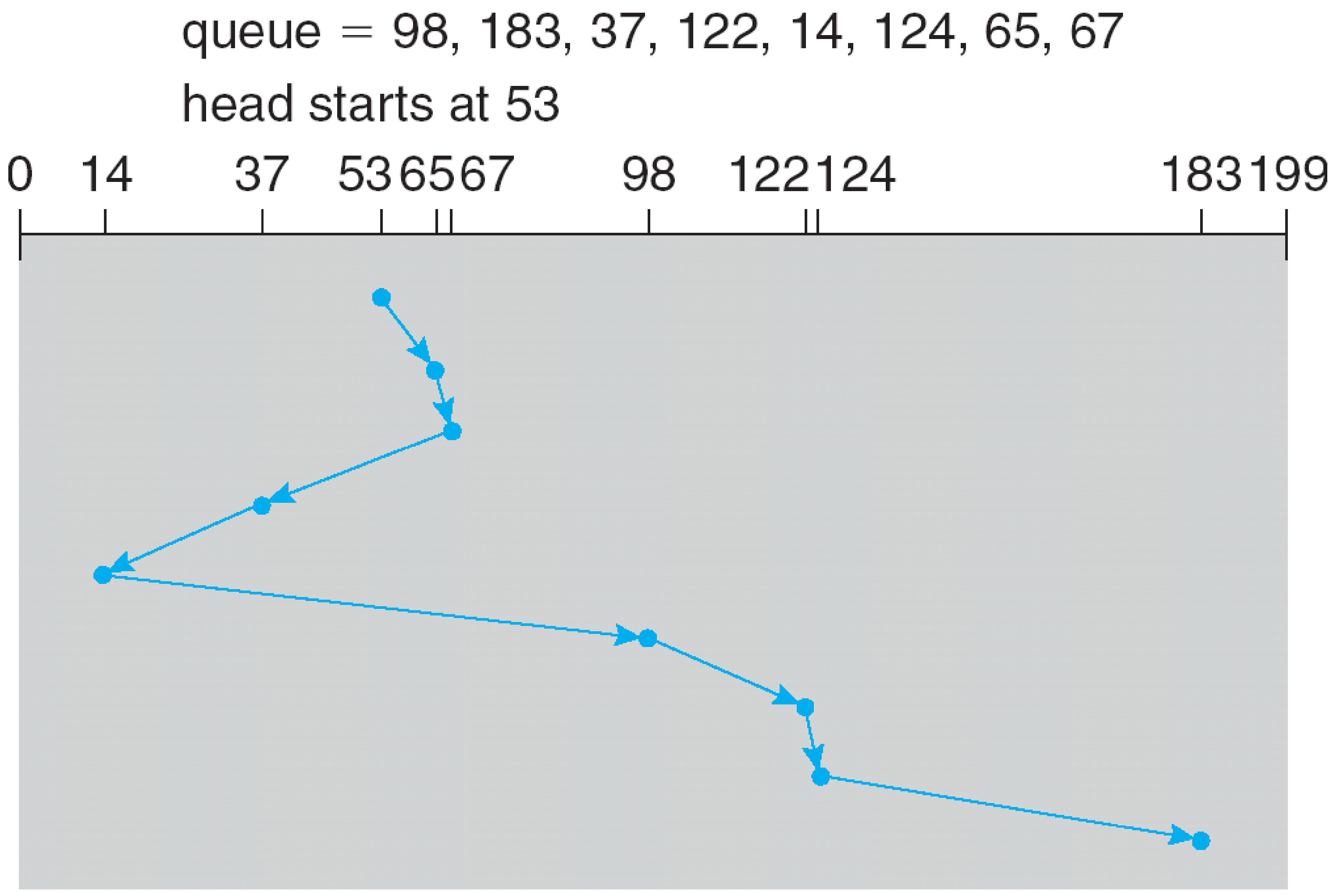
Illustration shows total head movement of 640 cylinders.



# 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.

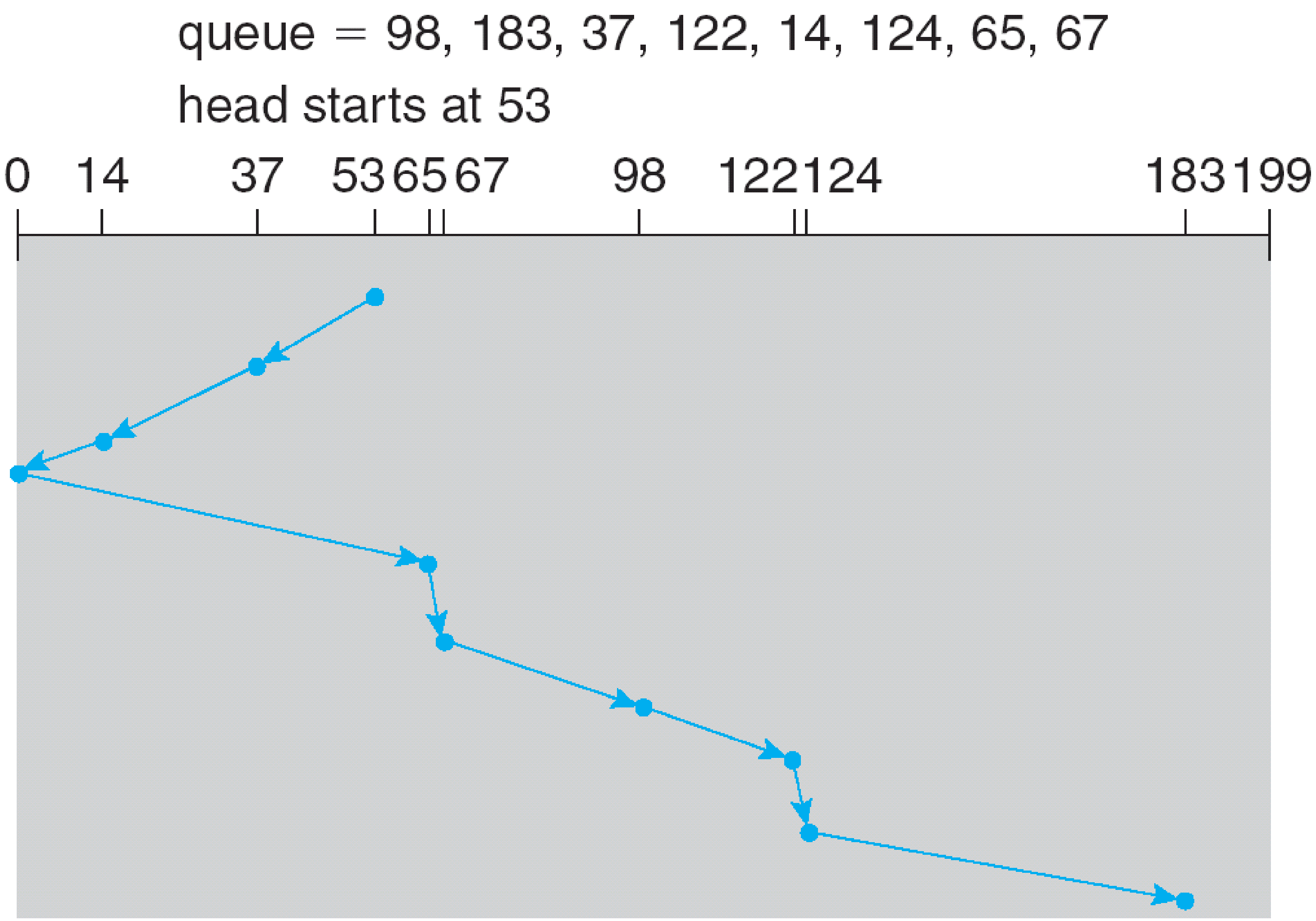
# SSTF (Cont.)


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

# 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

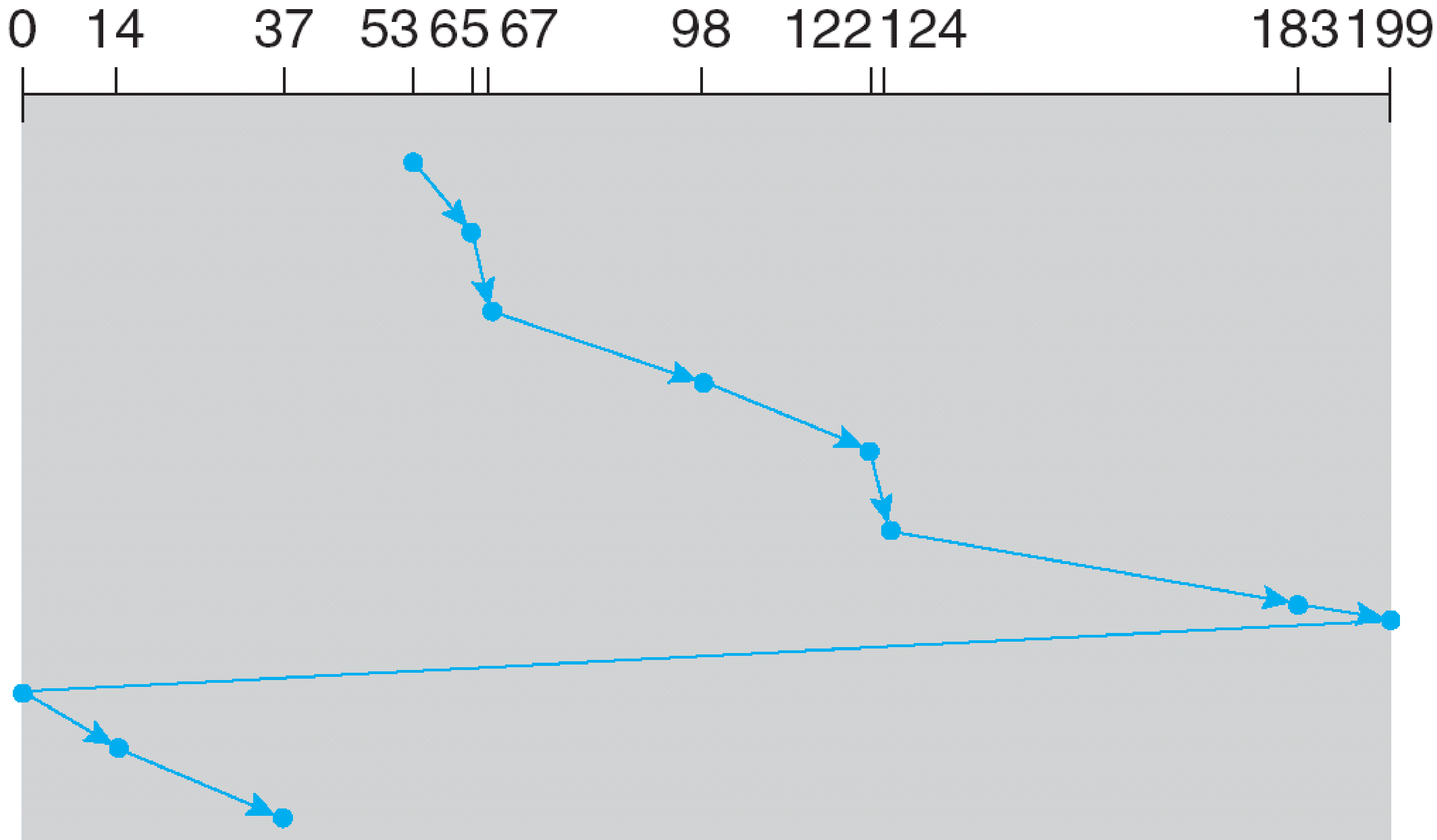
# C-SCAN

- 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



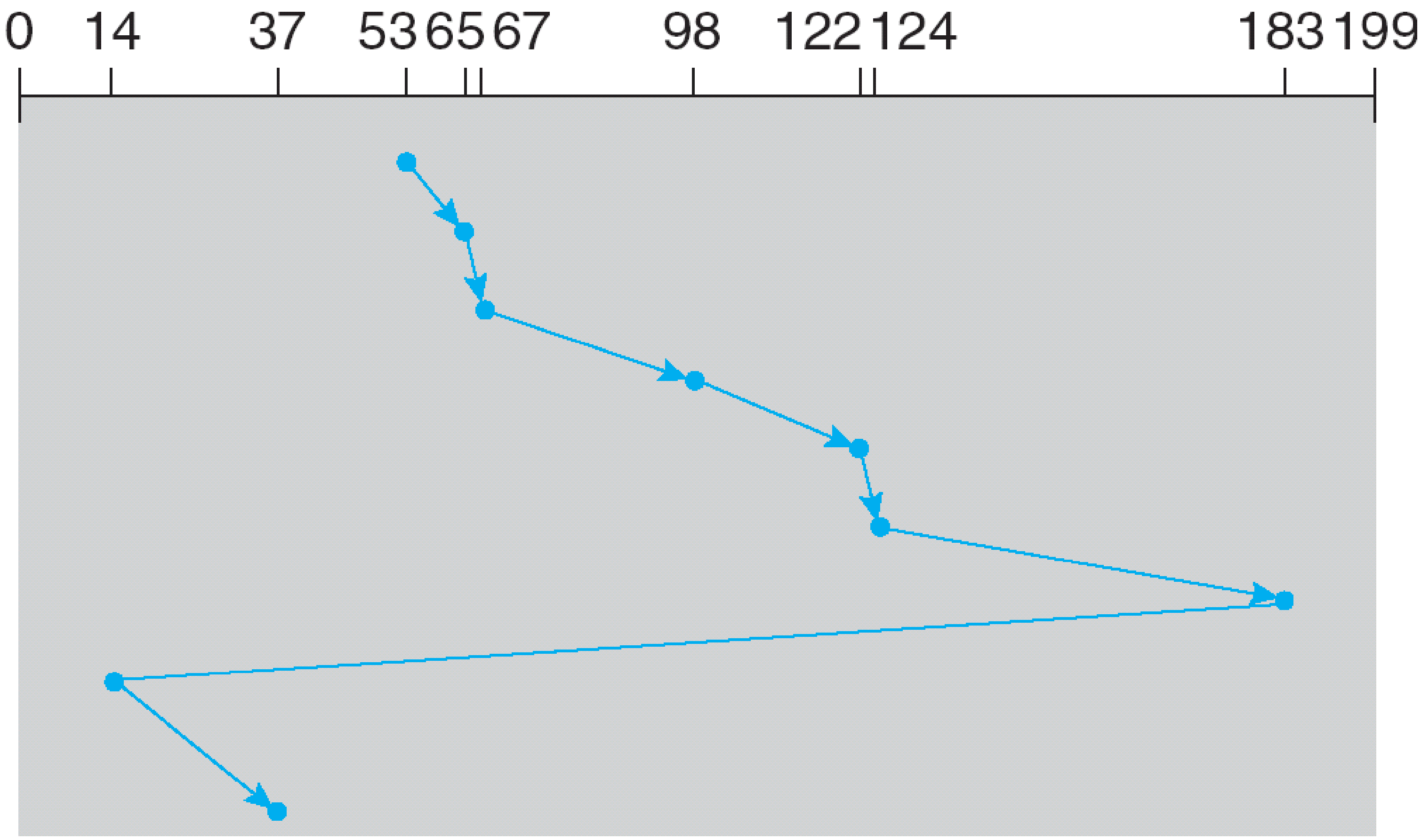
# C-LOOK

- 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

# Disk Formatting

- 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
  - There is a short gap between sectors

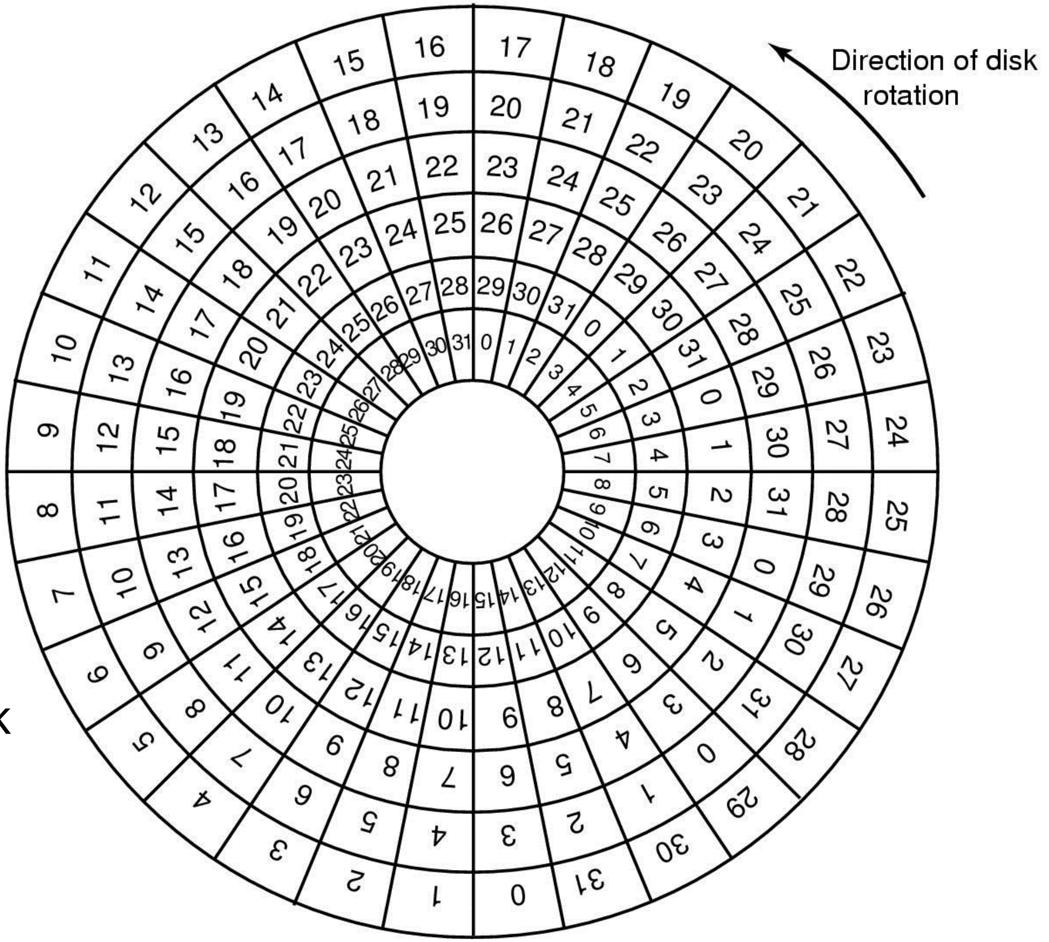

- 
  - Also contains cylinder and sector numbers
  - Data is usually 512 bytes
  - ECC field used to detect and recover from read errors

# Cylinder Skew

- Why cylinder skew?

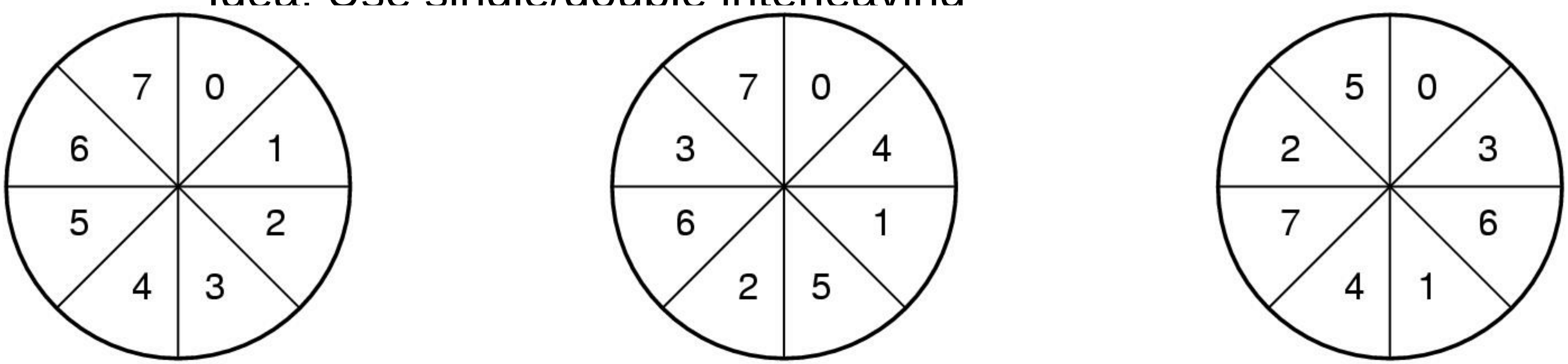
- How much skew?
- Example, if
  - 10000 rpm
    - Drive rotates in 6 ms
  - Track has 300 sectors
    - New sector every 20 µs
  - If track seek time 800 µs
    - ⇒40 sectors pass on seek

Cylinder skew: 40 sectors



# Formatting and Performance

- If 10K rpm, 300 sectors of 512 bytes per track
  - 153600 bytes every 6 ms ⇒ 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



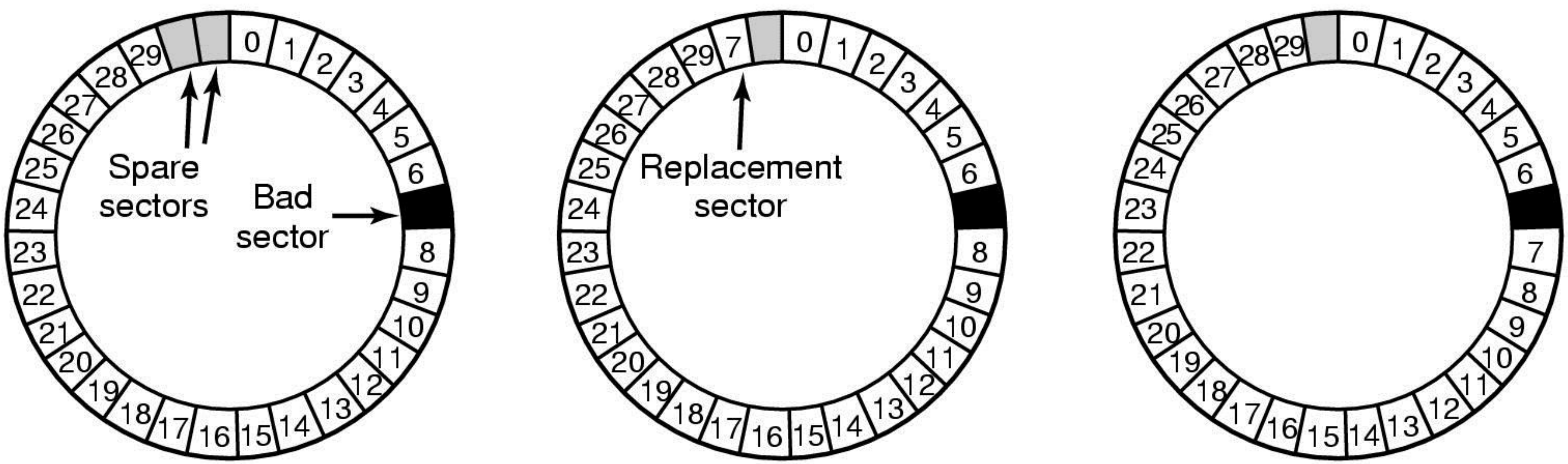

(a) (b) (c)

# 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.

# 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)




(a) (b) (c)

# 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
- 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
- Striping reduces reliability
  - 100 disks have 1/100th mean time between failures of one disk
- So, we need Striping for performance, but we need something to help with reliability / availability
- To improve reliability, we can add redundant data to the disks, in addition to Striping

# RAID

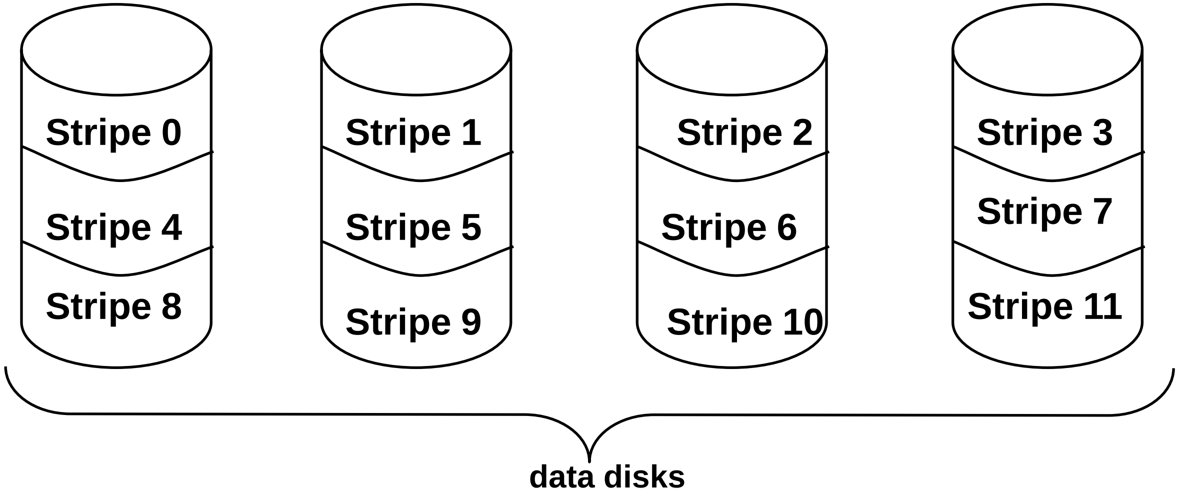
- A RAID is a Redundant Array of Inexpensive Disks
  - In industry, “I” is for “Independent”
  - The alternative is SLED, single large expensive disk
- Disks are small and cheap, so it’s easy to put lots of disks (10s to 100s) in one box for increased storage, performance, and availability
- The RAID 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 access
- **Redundancy**
  - uniformly distribute redundancy info on disks: avoids load-balancing problems
  - concentrate redundancy info on a small number of disks: partition the set into data disks and redundant disks

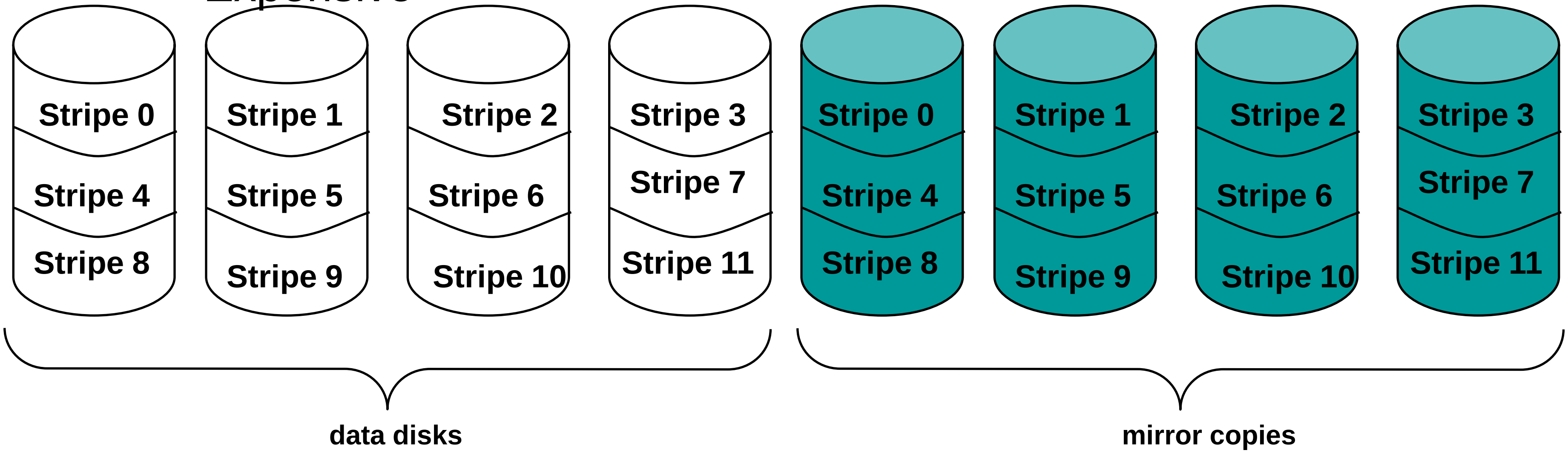
# Raid Level 0

- Level 0 is <u>nonredundant</u> disk array
- Files are Striped across disks, no redundant info
- High read throughput
- Best write throughput (no redundant info to write)
- Any disk failure results in data loss
  - Reliability worse than SLED



# 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



# Parity and Hamming Codes

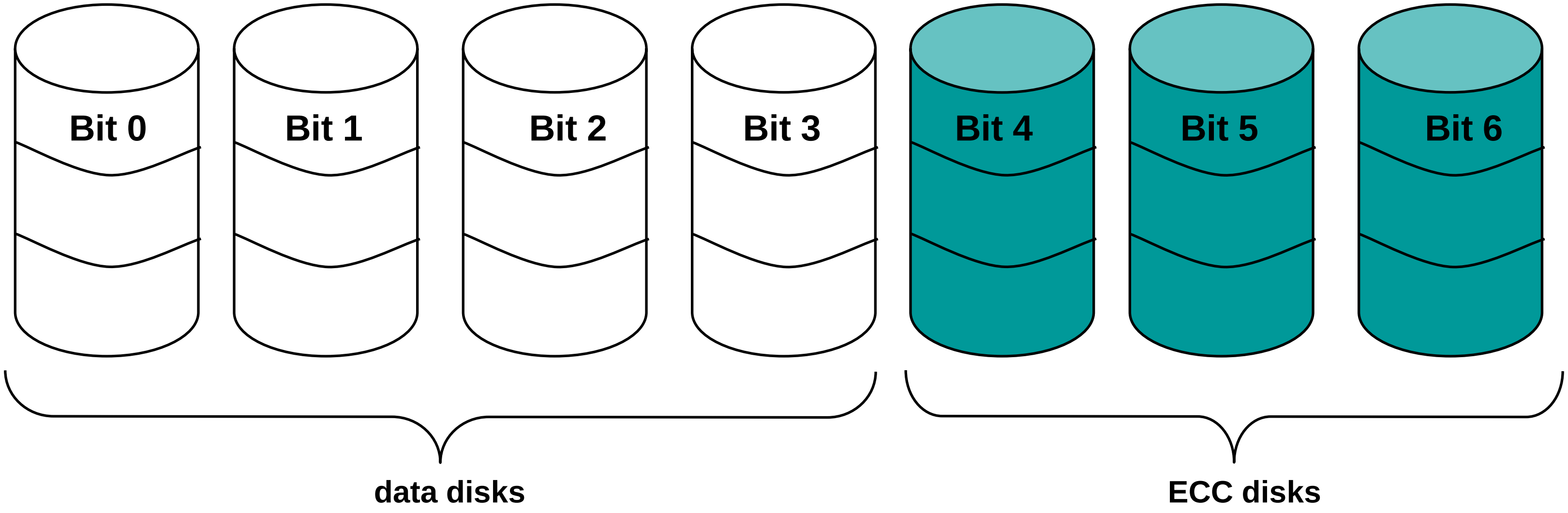
- What do you need to do in order to detect and correct a one-bit error ?
  - Suppose you have a binary number, represented as a collection of bits: <b3, b2, b1, b0>, e.g. 0110
- Detection is easy
- Parity:
  - Count the number of bits that are on, see if it’s odd or even
    - EVEN parity is 0 if the 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

# 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.

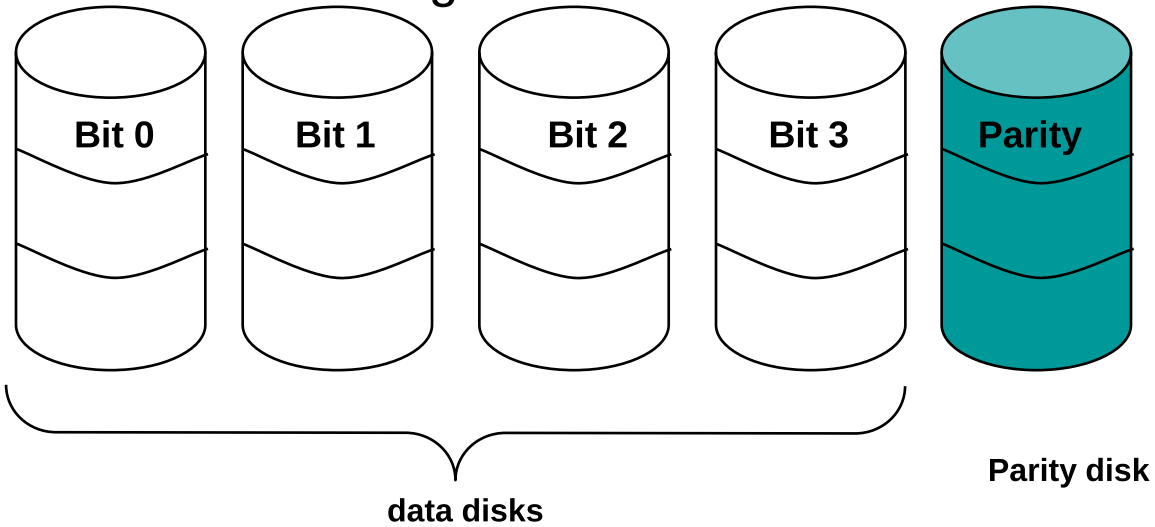
# Raid Level 2

- Bit-level Striping with Hamming (ECC) codes for error correction
- All 7 disk arms are synchronized and move in unison
- Complicated controller
- Single access at a time
- Tolerates only one error, but with no performance degradation



# Raid Level 3

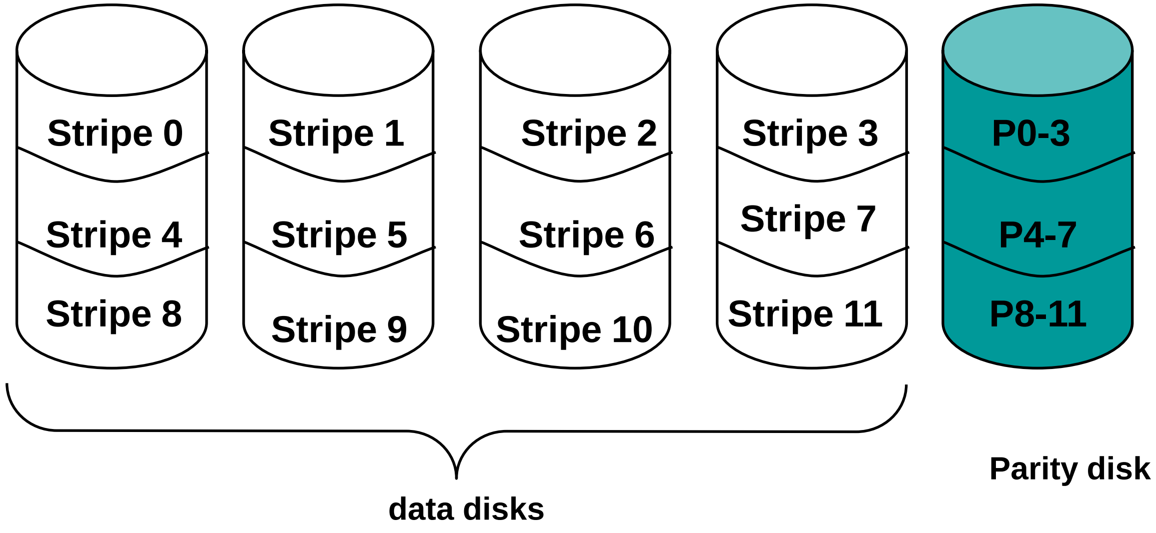
- Use a parity disk
  - Each bit on the parity disk is a parity function of the corresponding bits on all the other disks
- A read accesses all the data disks
- A write accesses all data disks <u>plus</u> the parity disk
- On disk failure, read remaining disks plus parity disk to compute the missing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




**Single parity disk can be used to detect and <u>correct</u> errors**

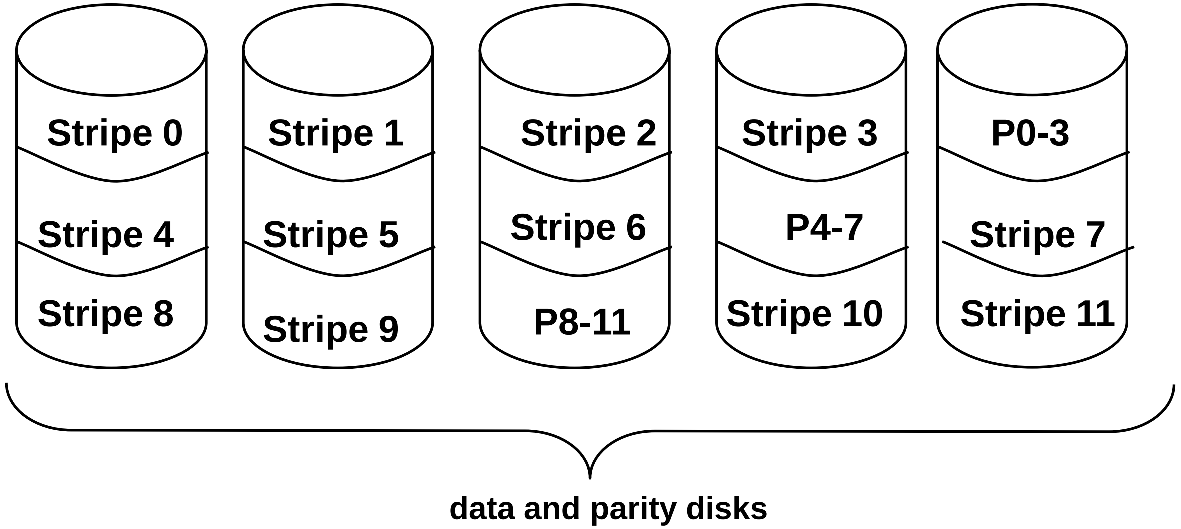
# Raid Level 4

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



# Raid Level 5

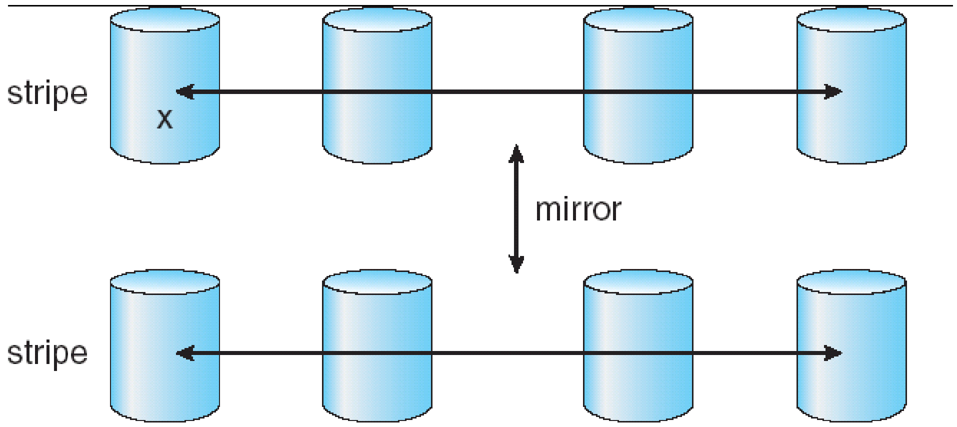
- Block Interleaved Distributed Parity
- Like parity scheme, but distribute the parity info over all disks (as well as data over all disks)
- Better read performance, large write performance
  - Reads can outperform SLEDs and RAID-0



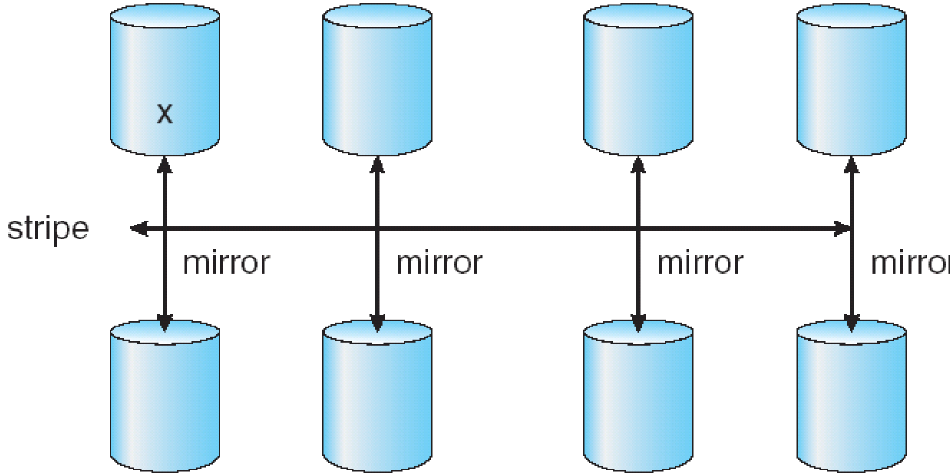
# Raid Level 6

- Level 5 with an extra parity bit
- Can tolerate two failures
  - What are the odds of having two concurrent failures ?
- May outperform Level-5 on reads, slower on writes

# RAID 0+1 and 1+0




a) RAID 0 + 1 with a single disk failure.



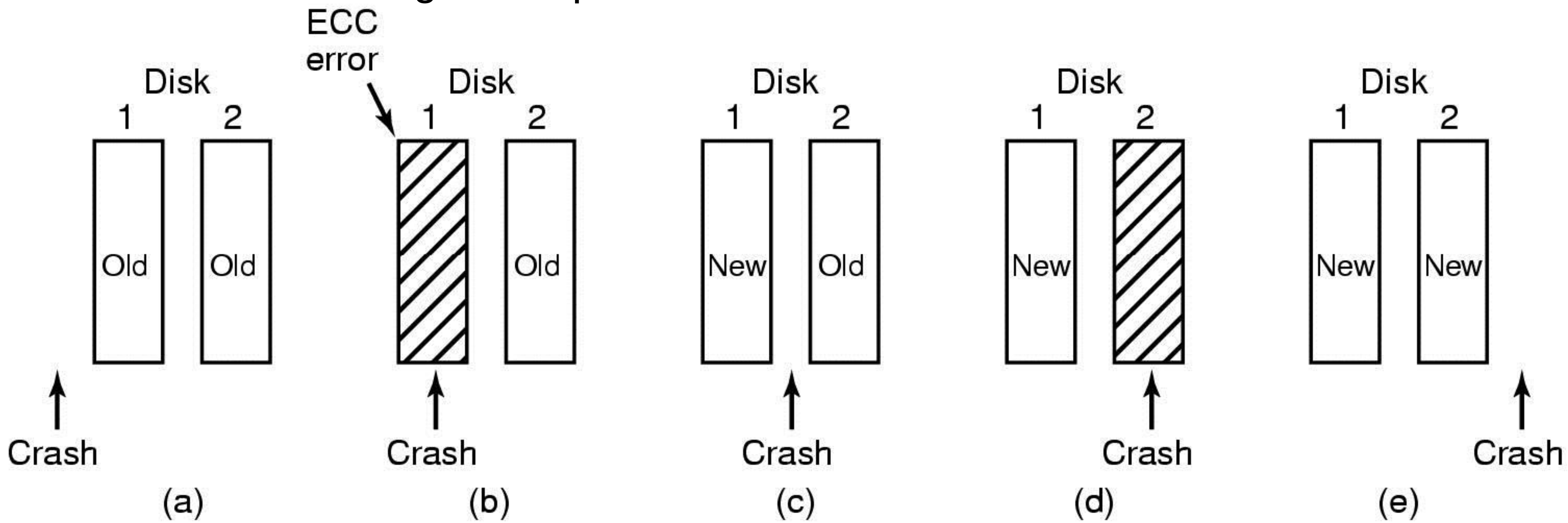

b) RAID 1 + 0 with a single disk failure.

# Stable Storage

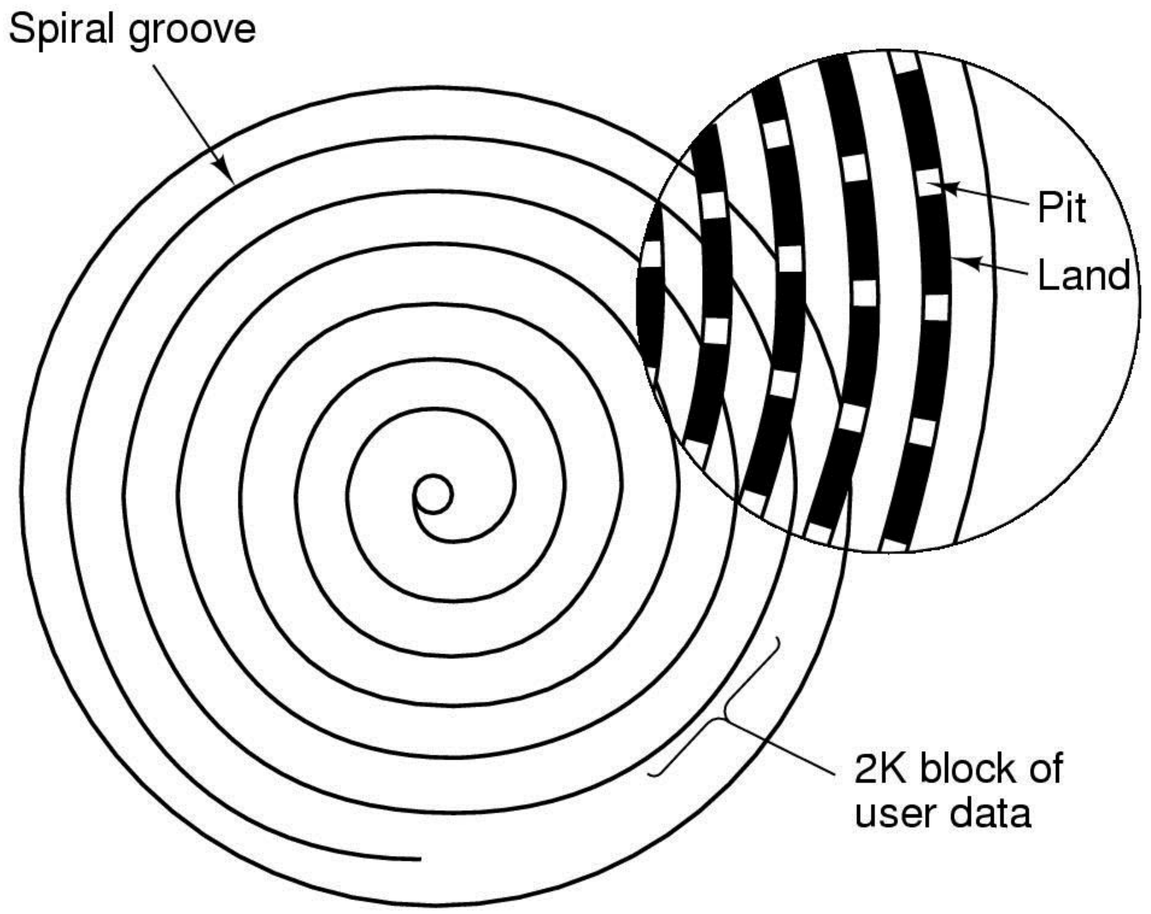
- Handling disk write errors:
  - Write lays down bad data
  - Crash during a write corrupts original data
- What we want to achieve? Stable Storage
  - When a write is issued, the disk either correctly writes data, or it does nothing, leaving existing data intact
- Model:
  - An incorrect disk write can be detected by looking at the ECC
  - It is very rare that same sector goes bad on multiple disks
  - CPU is fail-stop

# Approach

- Use 2 identical disks
  - corresponding blocks on both drives are the same
- 3 operations:
  - Stable write: retry on 1st until successful, then try 2nd disk
  - Stable read: read from 1st. If ECC error, then try 2nd
  - Crash recovery: scan corresponding blocks on both disks
    - If one block is bad, replace with good one
    - If both are good, replace block in 2nd with the one in 1st

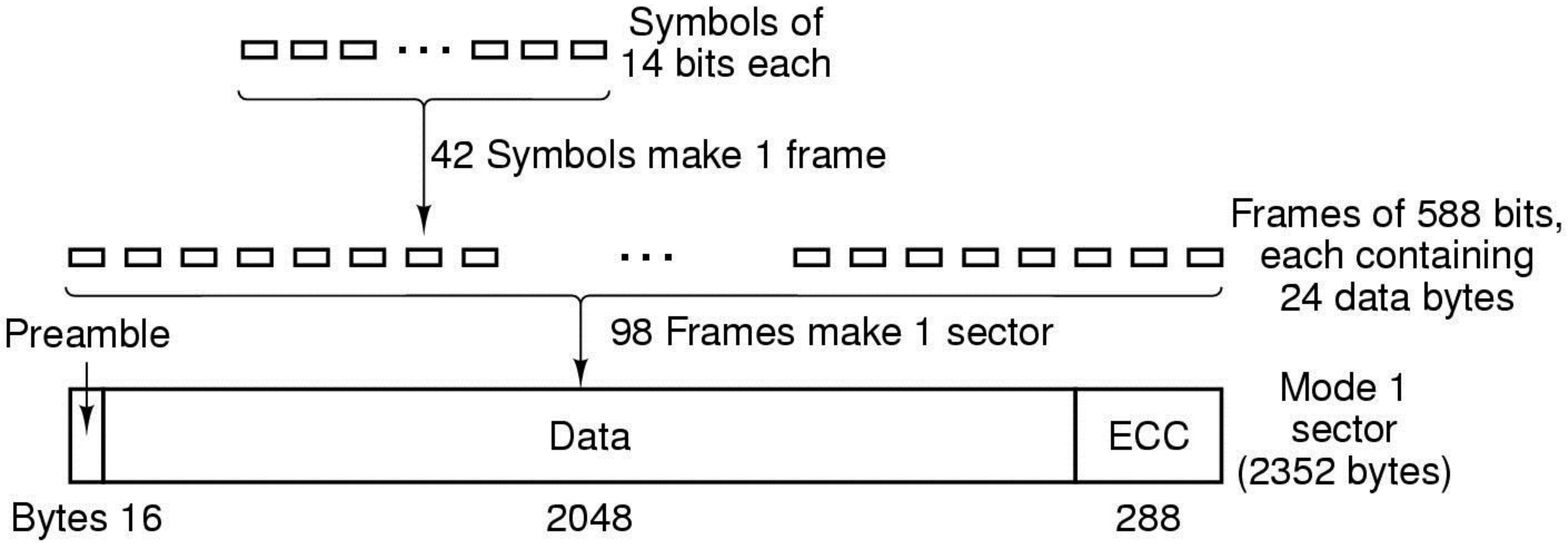


# CD-ROMs




Spiral makes 22,188 revolutions around disk (approx 600/mm).
Will be 5.6 km long. Rotation rate: 530 rpm to 200 rpm

# CD-ROMs




Logical data layout on a CD-ROM