# Disks and RAID

CS 4410
Operating Systems

[R. Agarwal, L. Alvisi, A. Bracy, F. Schneider, E. Sirer, R. Van Renesse]

# Storage Devices

- Magnetic disks (HDD)
- Flash drives (SSD)

# What do we want from storage?

- Fast: data is there when you want it
- Reliable: data fetched is what you stored
- Plenty: there should be lots of it
- Affordable: won’t break the bank

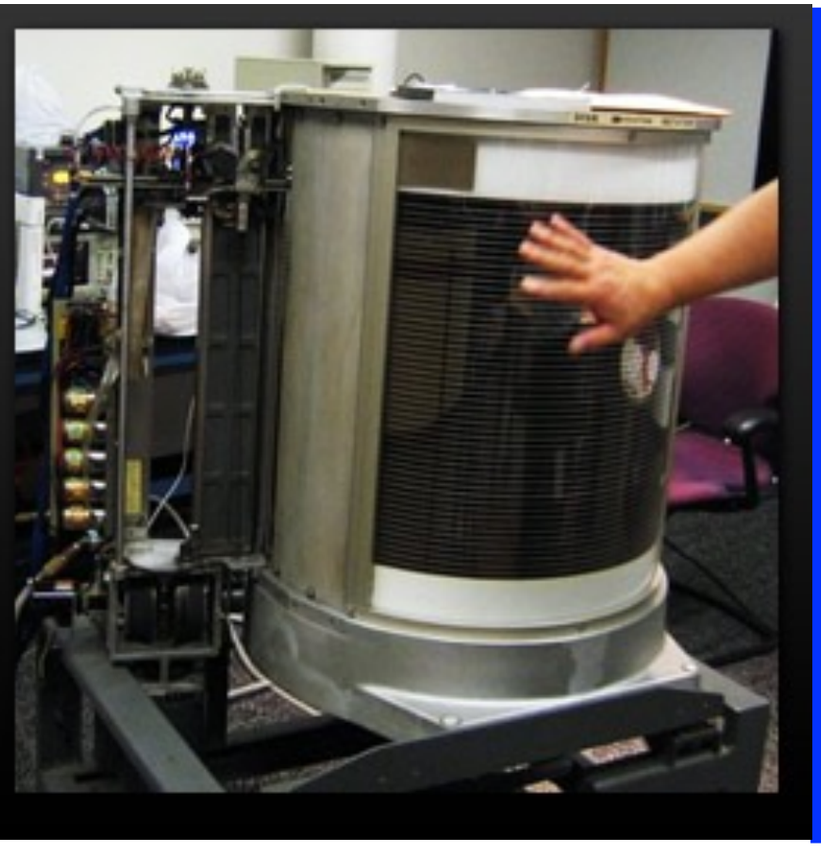
# Magnetic Disks are 65 years old!

**THAT WAS THEN**

- 13th September 1956
- The IBM RAMAC 350
- Total Storage = 5 million characters (just under 5 MB)

**THIS IS NOW**

- 2.5-3.5” hard drive
- Example: 500GB Western Digital Scorpio Blue hard drive
- easily up to a few TB



http://royal.pingdom.com/2008/04/08/the-history-of-computer-data-storage-in-pictures/

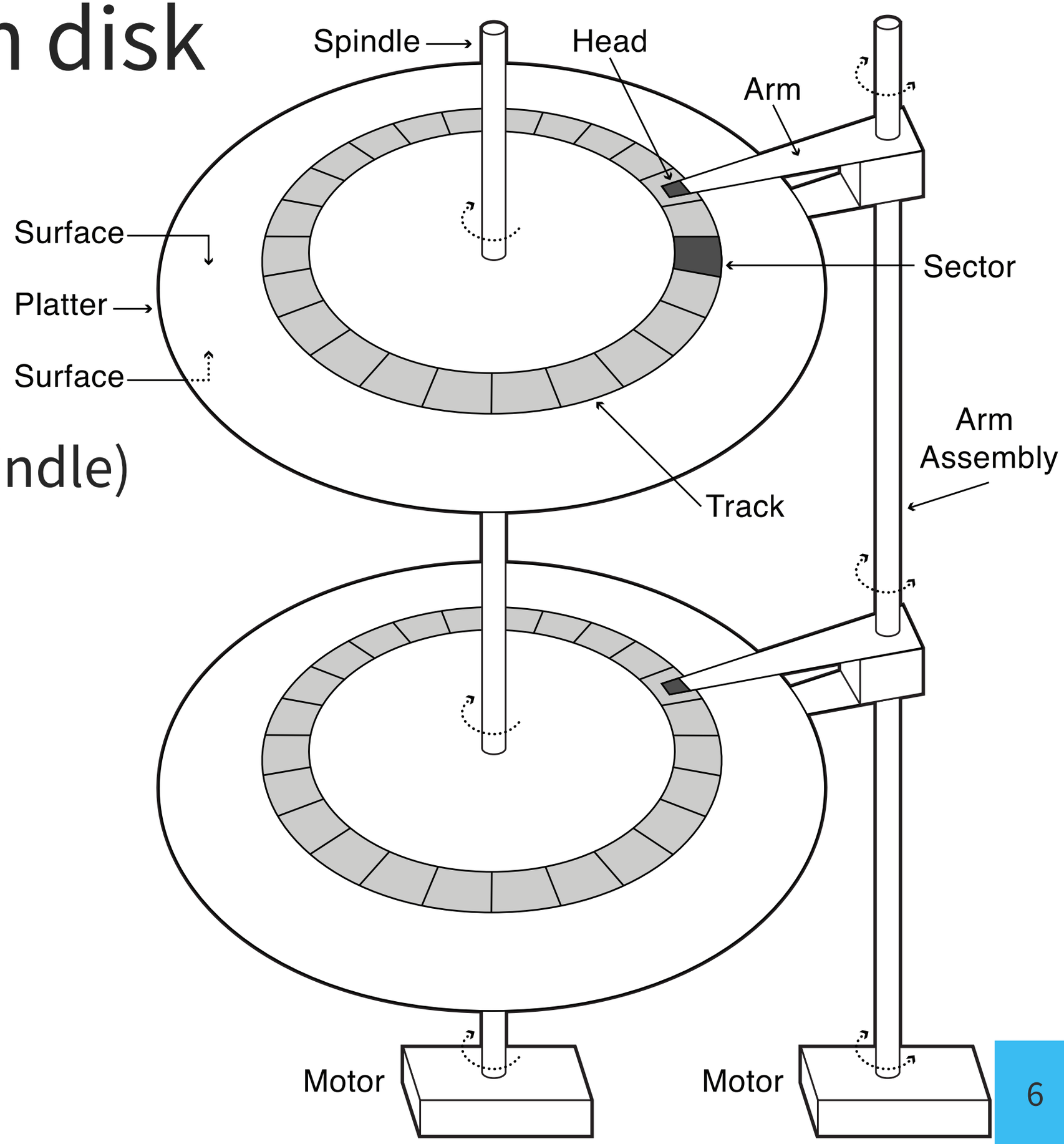
# RAM (Memory) vs. HDD (Disk) vs. SSD, 2020

| | RAM | HDD | SSD |
|---|---|---|---|
| **Typical Size** | 16 GB | 1 TB | 1TB |
| **Cost** | $5-10 per GB | $0.05 per GB | $0.10 per GB |
| **Latency** | 15 ns | 15 ms | 1ms |
| **Throughput (Sequential)** | 8000 MB/s | 175 MB/s | 500 MB/s |
| **Power Reliance** | volatile | non-volatile | non-volatile |

# Reading from disk

Must specify:

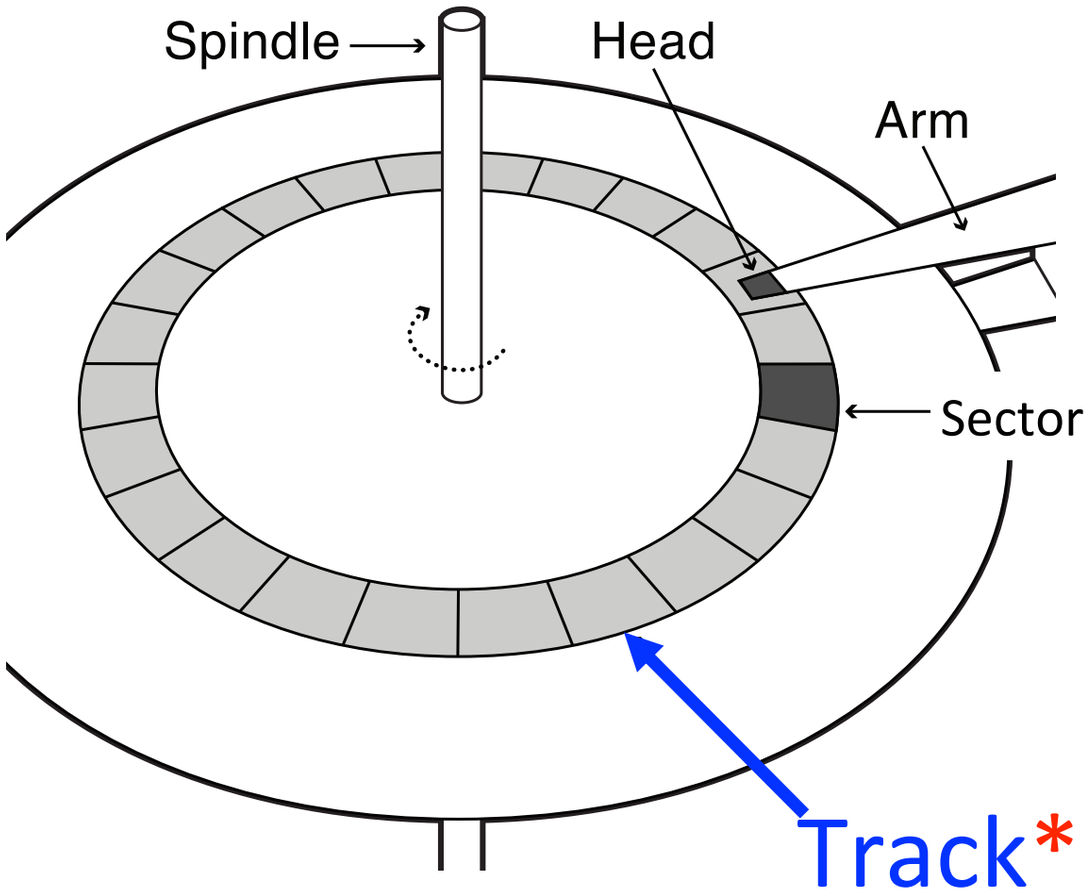
- cylinder #
  (distance from spindle)
- head #
- sector #
- transfer size
- memory address



# Disk Tracks

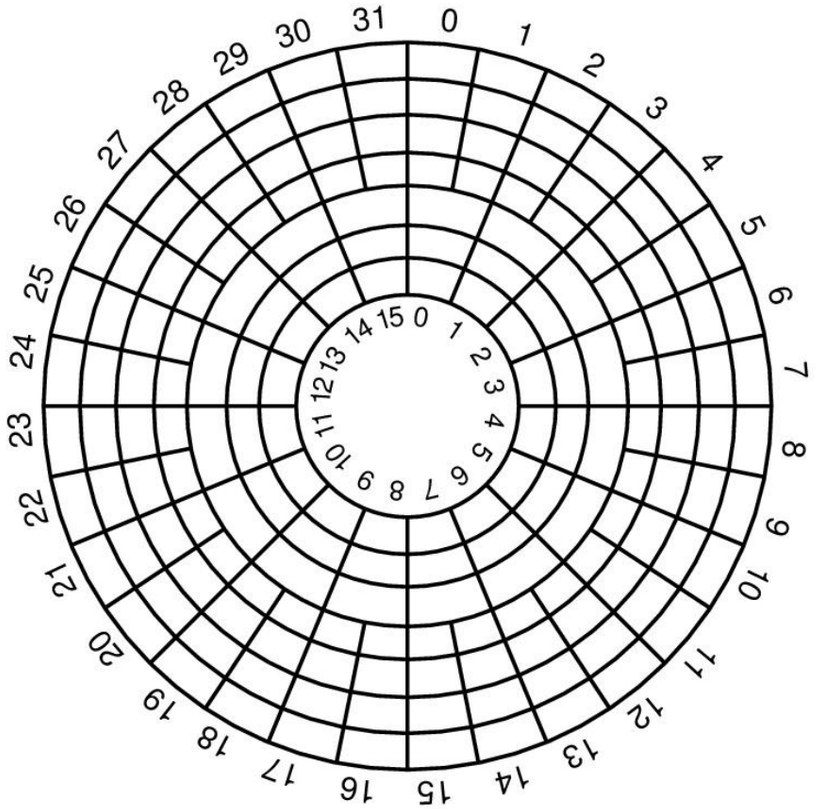
~ 1 micron wide (1000 nm)

- Wavelength of light is ~ 0.5 micron
- Resolution of human eye: 50 microns
- 100K tracks on a typical 2.5” disk




Track length varies across disk

- Outside:
  - More sectors per track
  - Higher bandwidth
- Most of disk area in outer regions

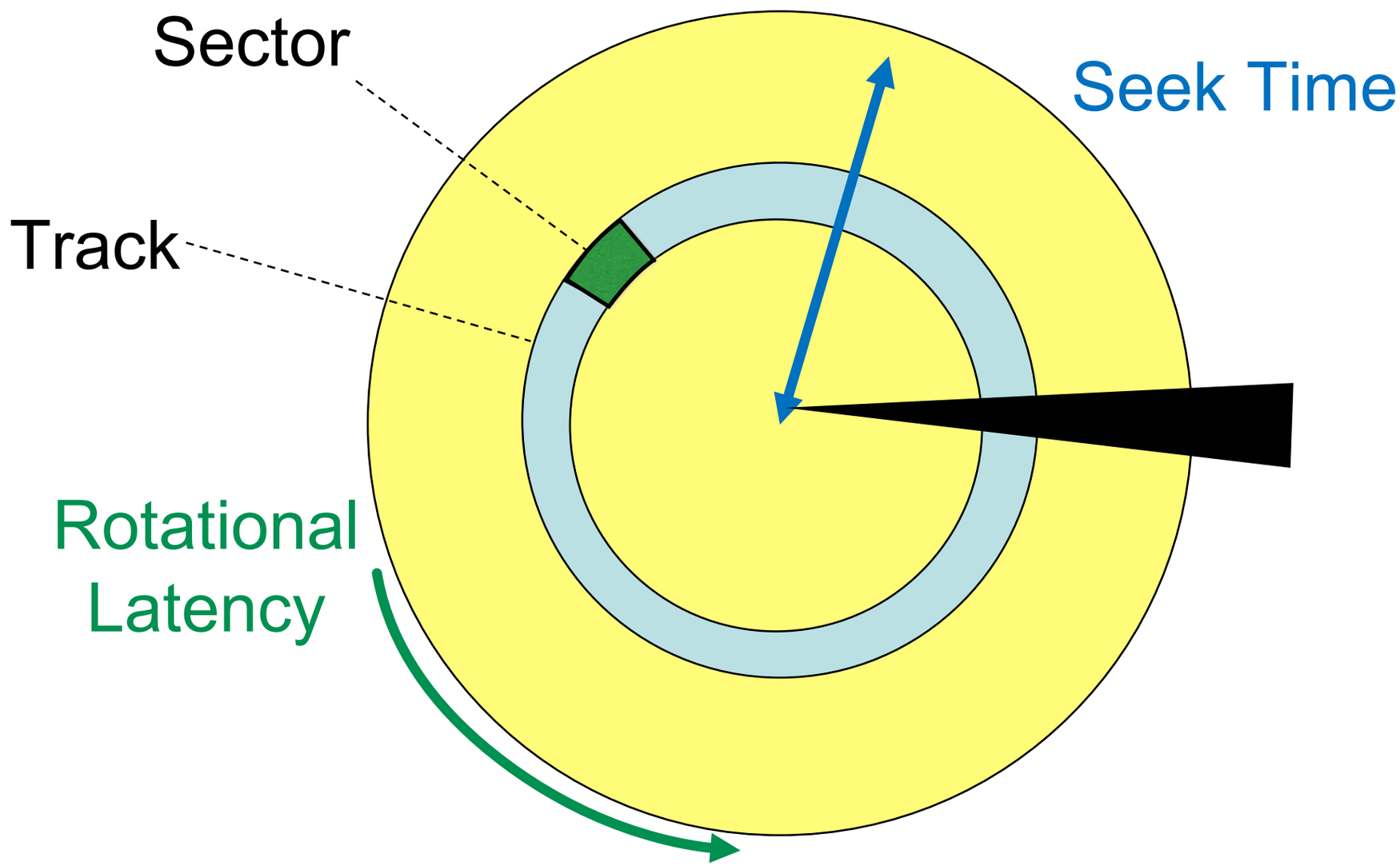


**not to scale: head is actually much bigger than a track*

# Disk overheads

*Disk Latency = **Seek Time** + **Rotation Time** + Transfer Time*

- **Seek:** to get to the track (5-15 millisecs (ms))
- **Rotational Latency:** to get to the sector (4-8 millisecs (ms))
  (on average, only need to wait half a rotation)
- **Transfer:** get bits off the disk (25-50 microsecs (µs))



# Disk Scheduling

**Objective:** minimize seek time

**Context:** a queue of cylinder numbers (#0-199)

Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

**Metric:** how many cylinders traversed?

# Disk Scheduling: **FIFO**

- Schedule disk operations in order they arrive
- Downsides?

**FIFO Schedule?**
**Total head movement?**

Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

# Disk Scheduling: Shortest Seek Time First

- Select request with minimum seek time from current head position
- A form of Shortest Job First (SJF) scheduling
- Not optimal: suppose cluster of requests at far end of disk → starvation!

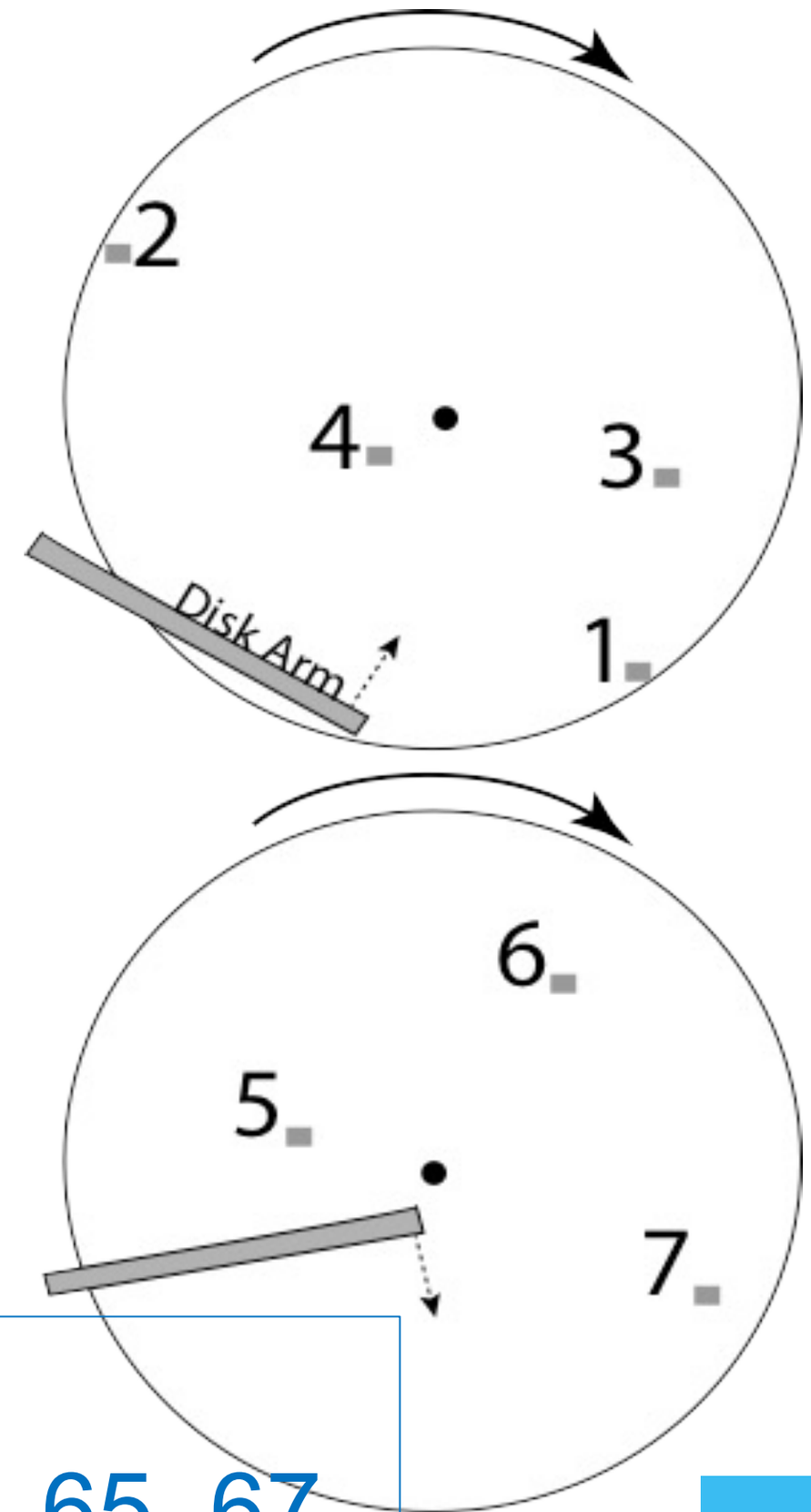
**SSTF Schedule?**
**Total head movement?**

Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

# Disk Scheduling: SCAN

Elevator Algorithm:

- arm starts at one end of disk
- moves to other end, servicing requests
- movement reversed @ end of disk
- repeat




**SCAN Schedule?**
**Total head movement?**

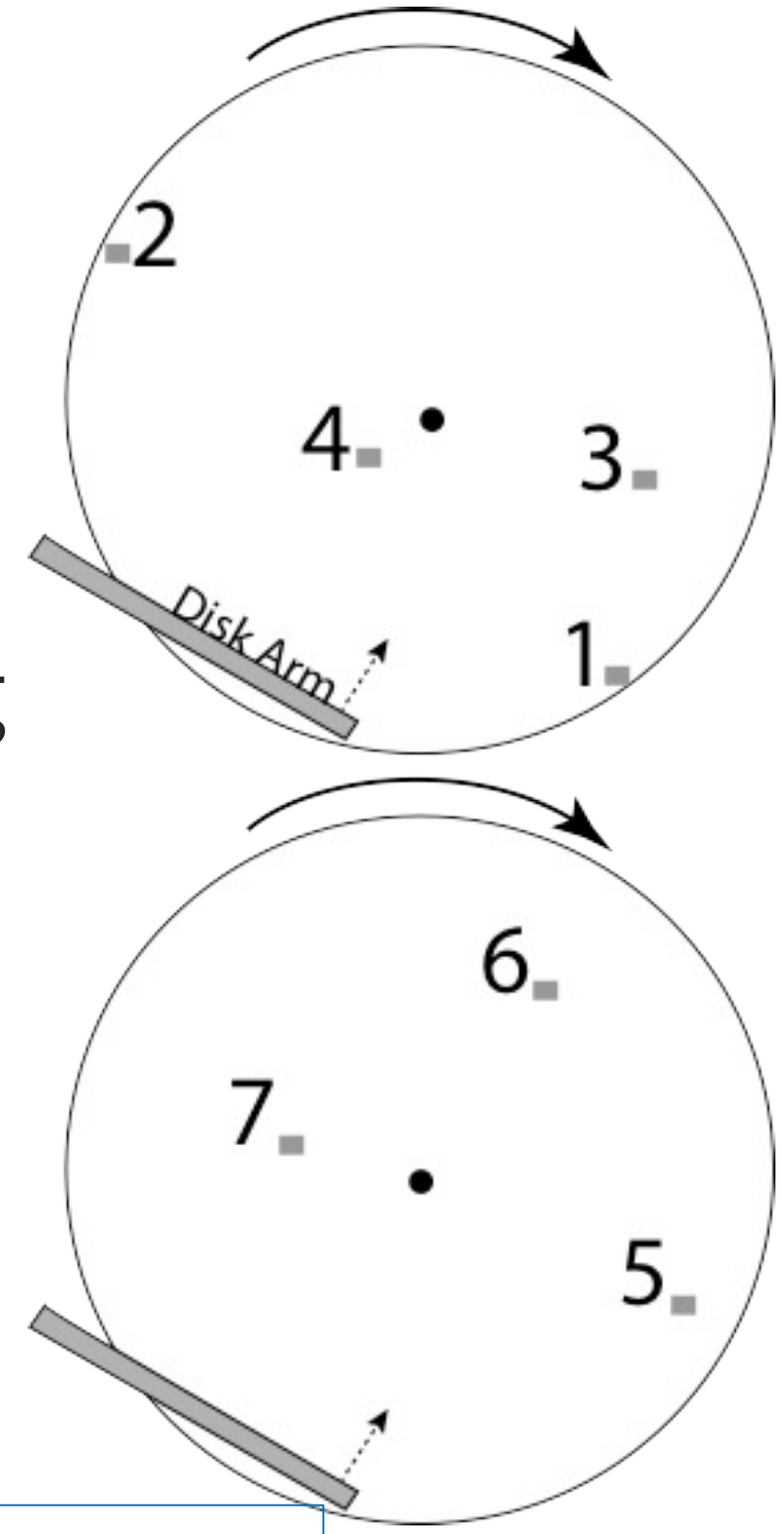
Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

# Disk Scheduling: C-SCAN

Circular list treatment:

- head moves from one end to other
- servicing requests as it goes
- reaches the end, returns to beginning
- no requests serviced on return trip

+ More uniform wait time than SCAN




**C-SCAN Schedule?**

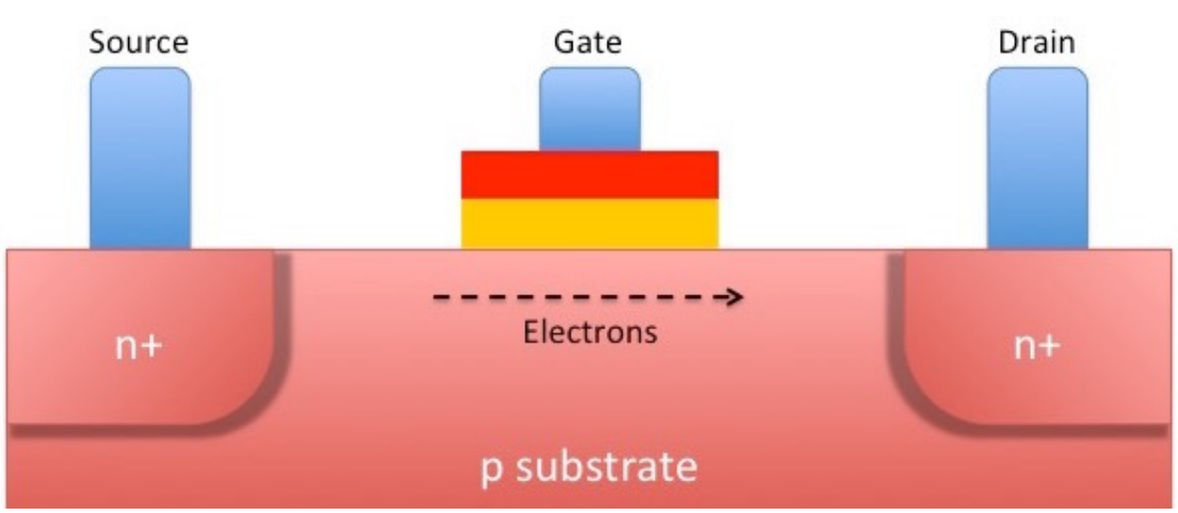
**Total Head movement?(?)**

Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

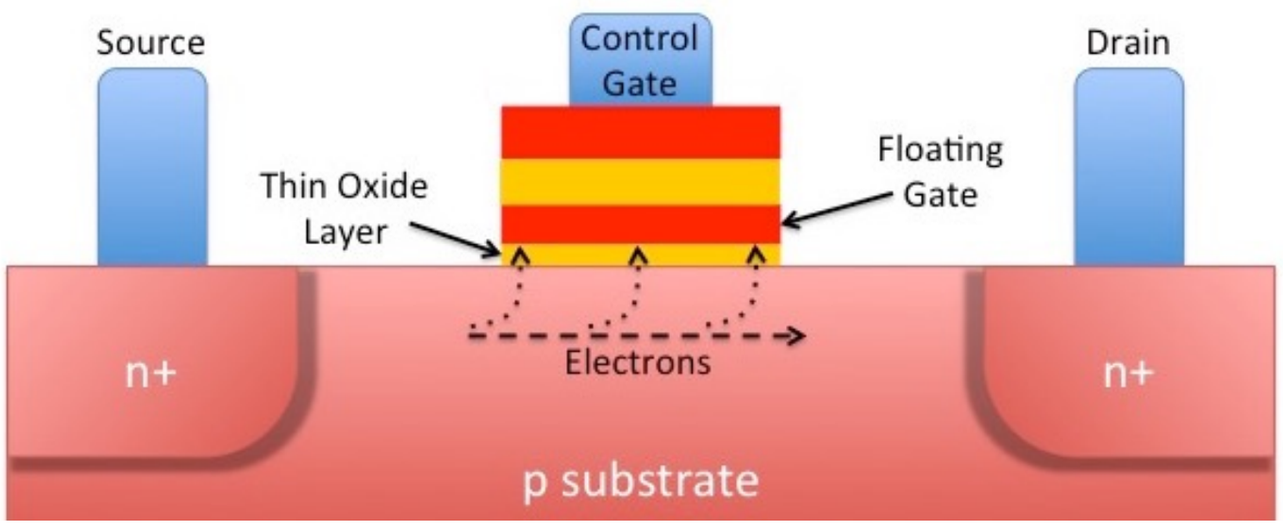
# Solid State Drives (Flash)

## Most SSDs based on NAND-flash

- retains its state for months to years without power




Metal Oxide Semiconductor Field Effect **Transistor** (MOSFET)



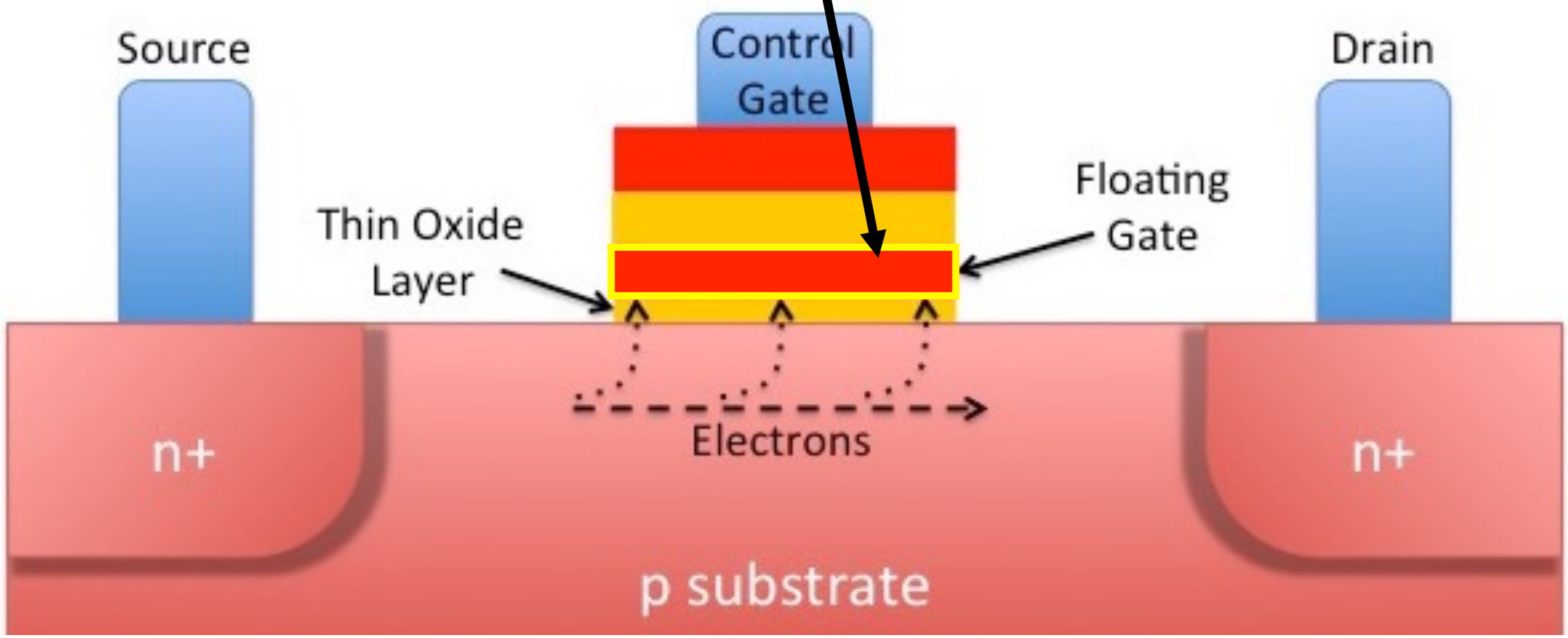

**Floating Gate MOSFET** (FGMOS)

https://flashdba.com/2015/01/09/understanding-flash-floating-gates-and-wear/

# NAND Flash

## Charge is stored in Floating Gate

(can have Single and Multi-Level Cells)




Floating Gate MOSFET (FGMOS)

https://flashdba.com/2015/01/09/understanding-flash-floating-gates-and-wear/

# Flash Operations

- **Erase block:** sets each cell to “1”
  - erase granularity = “erasure block” = 128-512 KB
  - time: several ms
- **Write (aka program) page:** can only write erased pages
  - write granularity = 1 page = 2-4KBytes
  - time: 100s of microseconds
- **Read page:**
  - read granularity = 1 page = 2-4KBytes
  - time: 10s of microseconds

- Flash drive consists of several *banks* that can be accessed in parallel
  - Each bank can have thousands of blocks

# Flash Limitations

- can't write 1 byte/word (must write whole blocks)
- limited # of erase cycles per block (memory wear)
  - $10^3$-$10^6$ erases and the cell wears out
  - reads can "disturb" nearby words and overwrite them with garbage

- **Lots of techniques to compensate:**
  - error correcting codes
  - bad page/erasure block management
  - wear leveling: trying to distribute erasures across the entire driver

# Flash Translation Layer

## Flash device firmware maps logical page # to a physical location

- Garbage collect erasure block by copying live pages to new location, then erase
  - More efficient if blocks stored at same time are deleted at same time (e.g., keep blocks of a file together)
- Wear-levelling: only write each physical page a limited number of times
- Remap pages that no longer work (sector sparing)

## Transparent to the device user

# Disk Failure Cases

## (1) Isolated Disk Sectors (1+ sectors down, rest OK)

**Permanent:** physical malfunction (magnetic coating, scratches, contaminants)

**Transient:** data corrupted but new data can be successfully written to / read from sector

## (2) Entire Device Failure

- Damage to disk head, electronic failure, wear out
- Detected by device driver, accesses return error codes
- Annual failure rates or Mean Time To Failure (MTTF)

# What do we want from storage?

- Fast: data is there when you want it
- Reliable: data fetched is what you stored
- Plenty: there should be lots of it
- Affordable: won't break the bank

Enter: Redundant Array of Inexpensive Disks (RAID)

- In industry, "I" is for "Independent"
- The alternative is SLED, single large expensive disk
- RAID + RAID controller looks just like SLED to computer (*yay, abstraction!*)

# RAID-0

**Files striped across disks**

**+ Fast**

**latency?**

**throughput?**

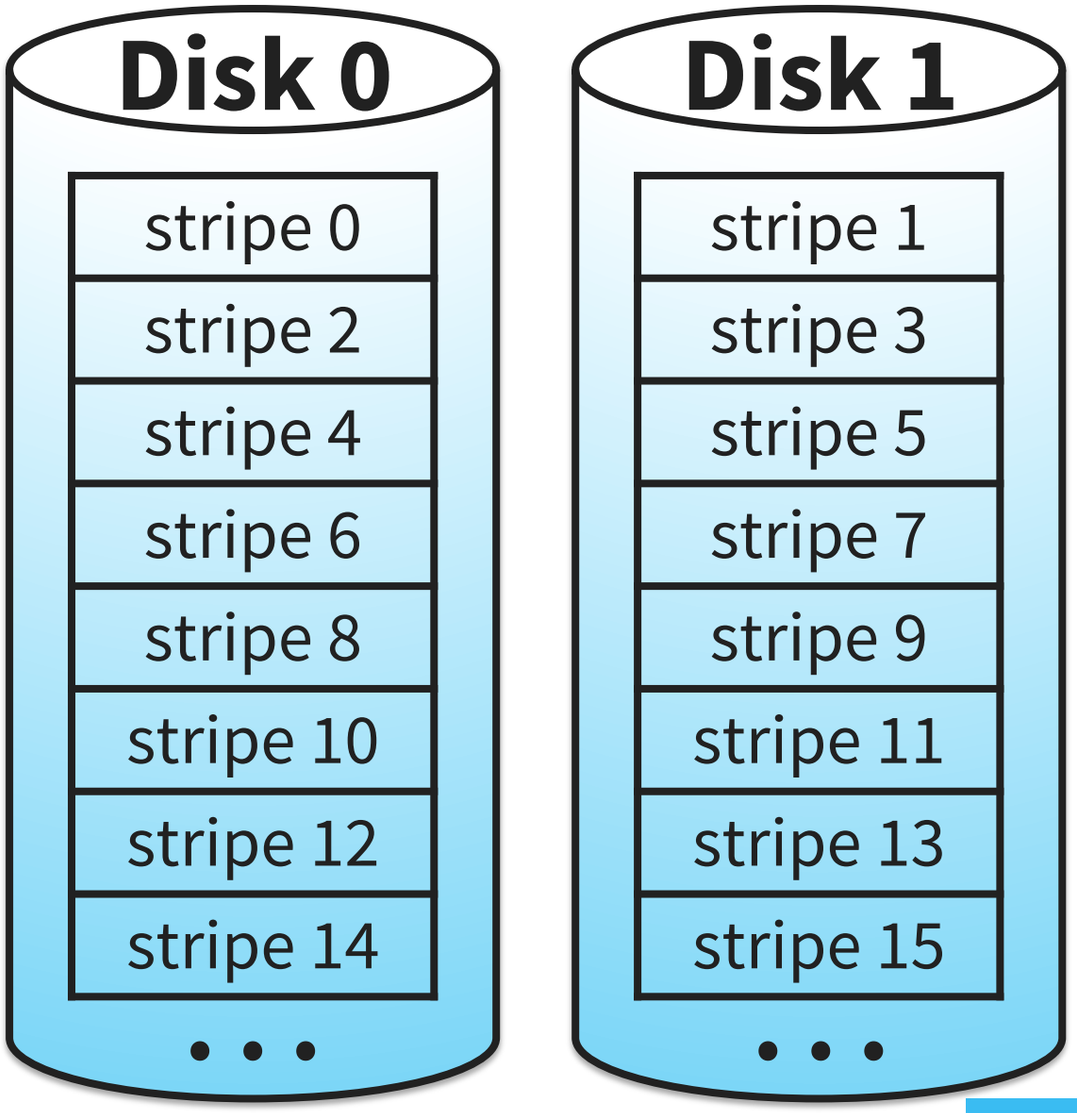
**+ Cheap**

**capacity?**

**– Unreliable**

**max #failures?**

**MTTF?**



# Striping and Reliability

Striping *reduces* reliability

- More disks ➜ higher probability of some disk failing
- $N$ disks: $1/N^{th}$ mean time between failures of 1 disk

What can we do to improve Disk Reliability?

# RAID-1

**Disks Mirrored:**

data written in 2 places

**+ Reliable**

**deals well with disk loss**

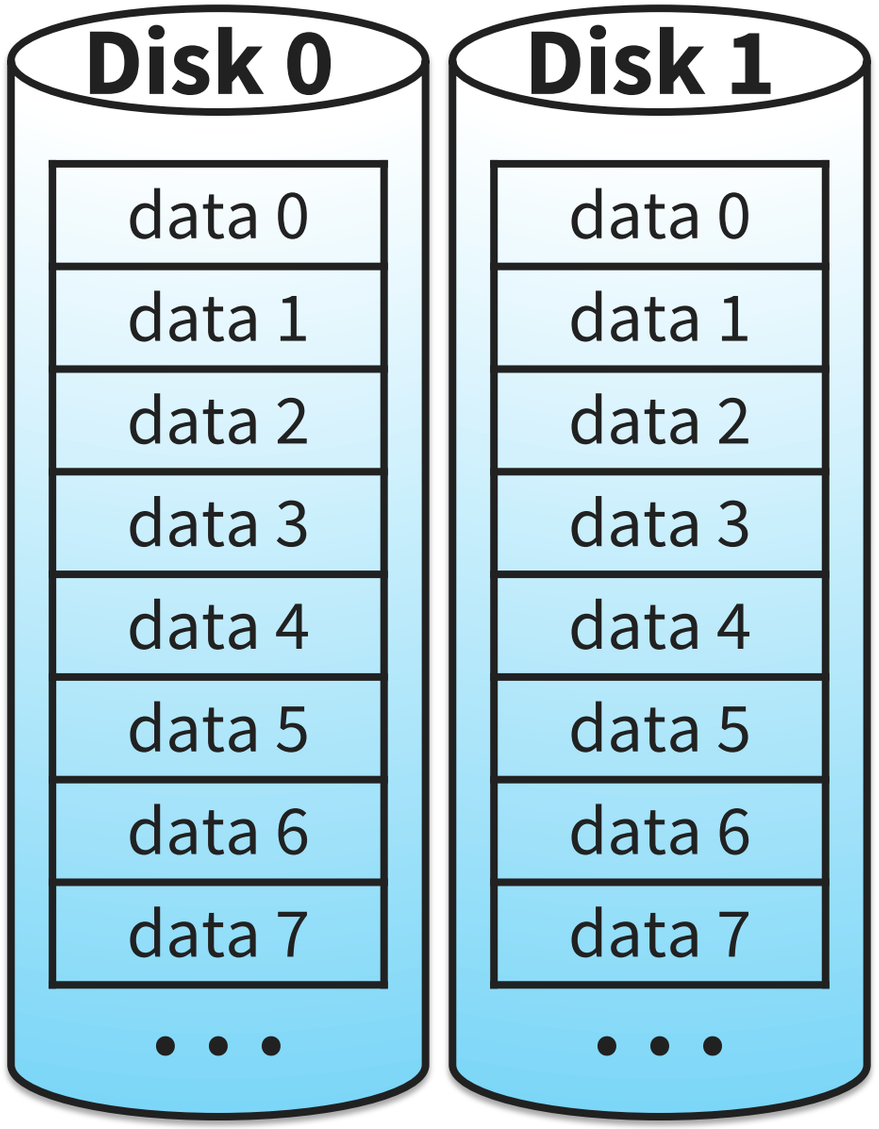
**but not corruption**

**+ Fast**

**latency?**

**throughput?**

**– Expensive**



# RAID-2

**bit**-level striping with ECC codes

- 7 disk arms synchronized, move in unison
- Complicated controller (→ very unpopular)
- Detect & Correct 1 error with no performance degradation

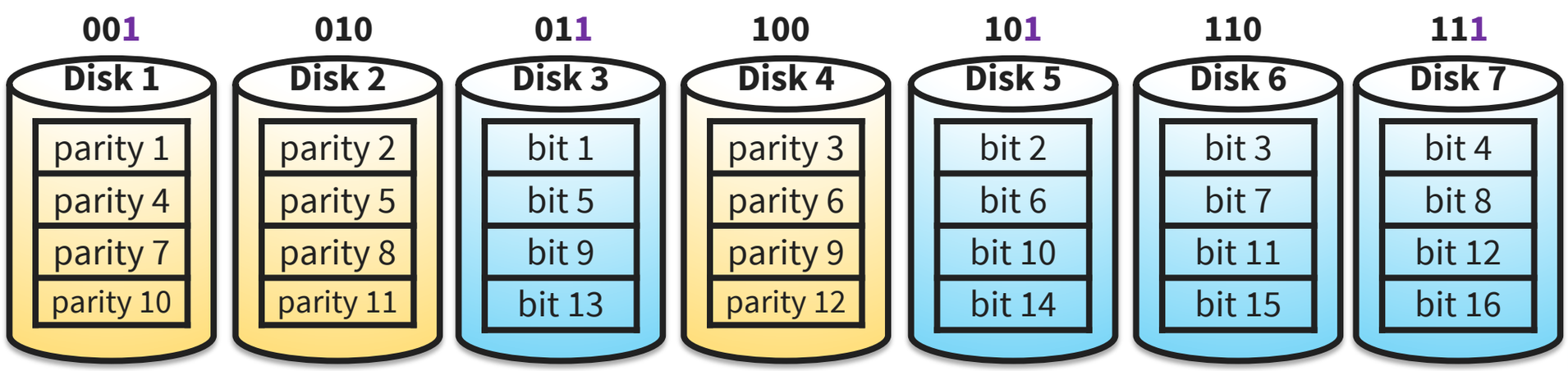
**+ Reliable**

**– Expensive**

**parity 1** = 3⊕5⊕7 (all disks whose # has 1 in LSB, xx1)

parity 2 = 3⊕6⊕7 (all disks whose # has 1 in 2nd bit, x1x)

parity 4 = 5⊕6⊕7 (all disks whose # has 1 in MSB, 1xx)



# Throughput, Bandwidth, and Latency

- Throughput is usually measured in "number of operations per second"
- Bandwidth is usually measured in "number of bytes per second"
- Latency is usually measured in "seconds"

Throughput and bandwidth are essentially the same thing, as each disk read/write operation transfers a fixed number of bytes ("block size")

# Latency vs Throughput

- If you do one operation at a time, then Latency × Throughput = 1.
  - e.g., if it takes 100 ms to do a read or write operation, then you can do 10 operations per second
- But operations can often be pipelined or executed in parallel
  - throughput higher than 1/latency
  - (road analogy)

# Sequential vs Random access

- With disks and file systems, sequential access is usually much faster than random access
- Reasons for faster sequential access:
  - "fewer seeks" on the disk
  - blocks can be "prefetched"
  - updates of parity blocks can be amortized across multiple operations ("batching")

# 2 more rarely-used RAIDS

**RAID-3: byte**-level striping + parity disk

- read accesses all data disks
- write accesses all data disks + parity disk
- On disk failure: read parity disk, compute missing data

**RAID-4: block**-level striping + parity disk

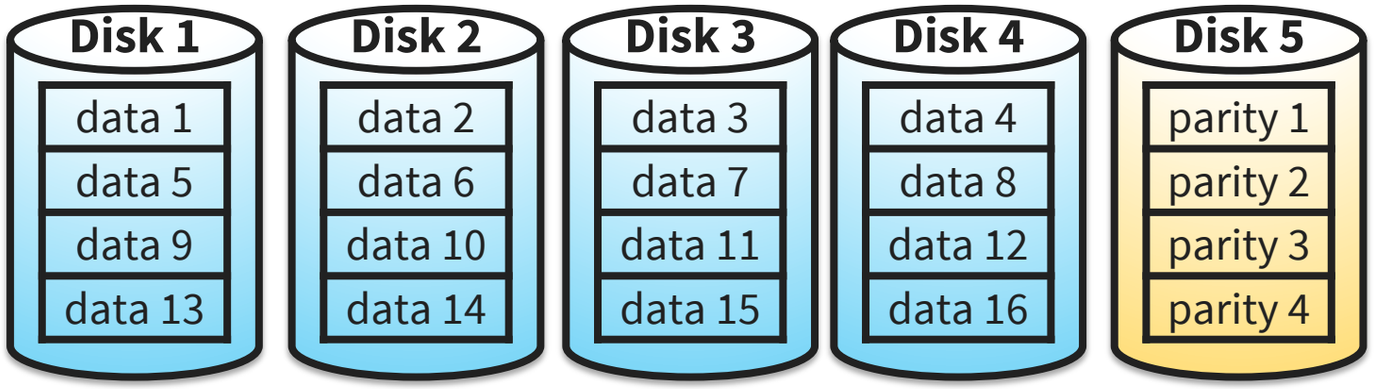
+ better spatial locality for disk access

**+ Cheap**

**– Slow Writes**

**– Unreliable**

parity disk is write bottleneck and wears out faster



# Using a parity disk

- $D_N = D_1 \oplus D_2 \oplus \ldots \oplus D_{N-1}$
  - $\oplus$ = XOR operation
  - therefore $D_1 \oplus D_2 \oplus \ldots \oplus D_N = 0$
- If one of $D_1 \ldots D_{N-1}$ fails, we can reconstruct its data by XOR-ing all the remaining drives
  - $D_i = D_1 \oplus \ldots \oplus D_{i-1} \oplus D_{i+1} \oplus \ldots \oplus D_N$

# Updating a block in RAID-4

- Suppose block lives on disk $D_1$
- Method 1:
  - read corresponding blocks on $D_2 \ldots D_{N-1}$
  - XOR all with new content of block
  - write disk $D_1$ and $D_N$ in parallel
- Method 2:
  - read $D_1$ (old content) and $D_N$
  - $D'_N = D_N \oplus D_1 \oplus D'_1$

    $= D_1 \oplus D_2 \oplus \ldots \oplus D_{N-1} \oplus D_1 \oplus D'_1$

    $= D'_1 \oplus D_2 \oplus \ldots \oplus D_{N-1}$
  - write disk $D_1$ and $D_N$ in parallel
- Either way:
  - write throughput: ½ of single disk
  - write latency: double of single disk

# Streaming update in RAID-4

- Save up updates to stripe across $D_1 \ldots D_{N-1}$
  - Batching!
- Compute $D_N = D_1 \oplus D_2 \oplus \ldots \oplus D_{N-1}$
- Write $D_1 \ldots D_N$ in parallel
- Throughput: $(N-1)$ times single disk

- Note that in all write cases $D_N$ must always be updated

➔ $D_N$ is a write performance bottleneck

# RAID 5: Rotating Parity w/Striping

**+ Reliable**

**you can lose one disk**

**+ Fast**

**($N-1$) x seq. throughput of single disk**

**$N$ x random read throughput of single disk**

**$N/4$ x random write throughput**

**+ Affordable**