# Disks and RAID

CS 4410
Operating Systems

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

# Storage Devices

- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for streaming access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (50x disk)
  - Block level random access
  - Good performance for reads; worse for random writes

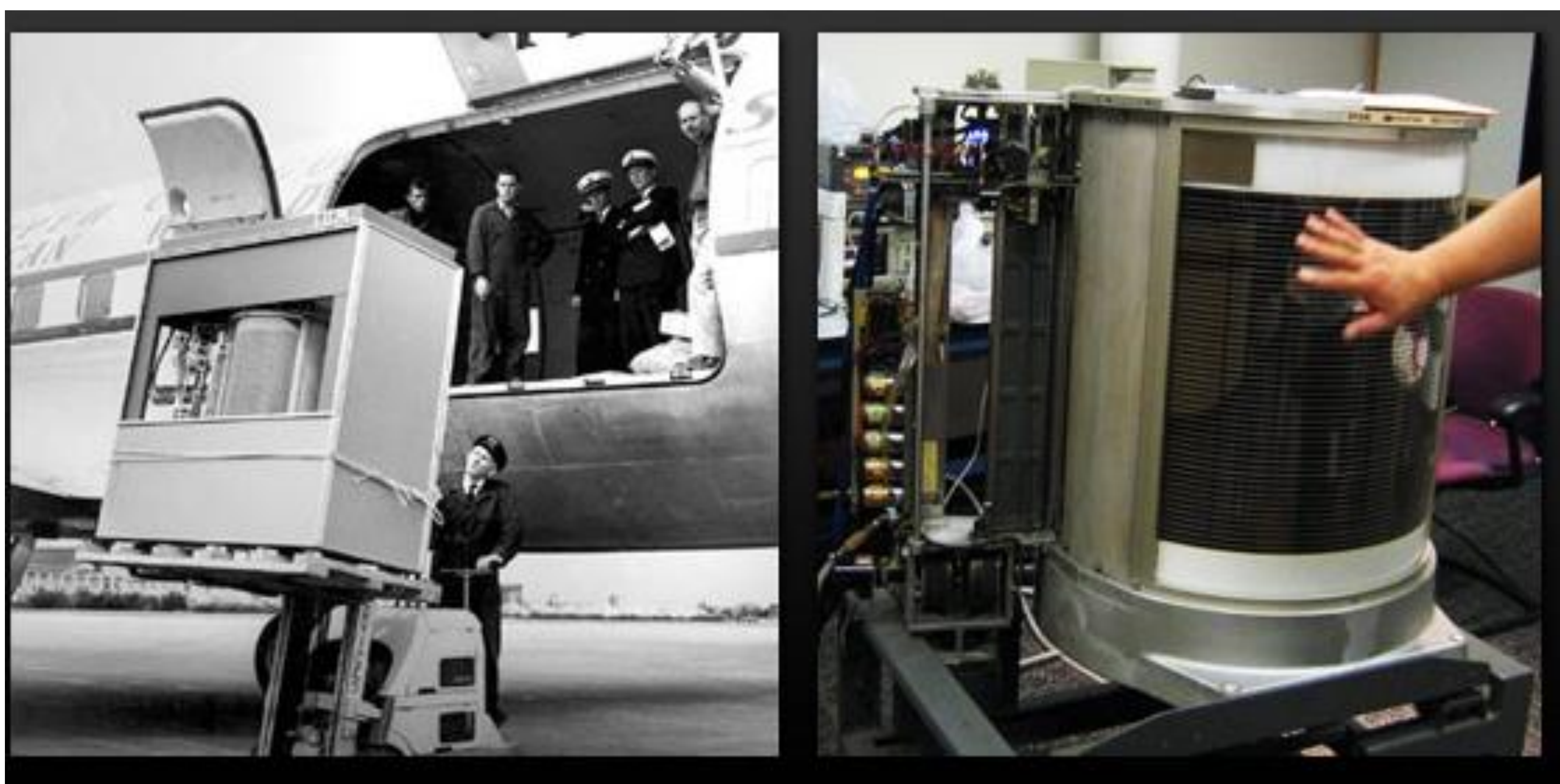
# Magnetic Disks are 60 years old!

**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 1 TB



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

# RAM (Memory) vs. HDD (Disk), 2018

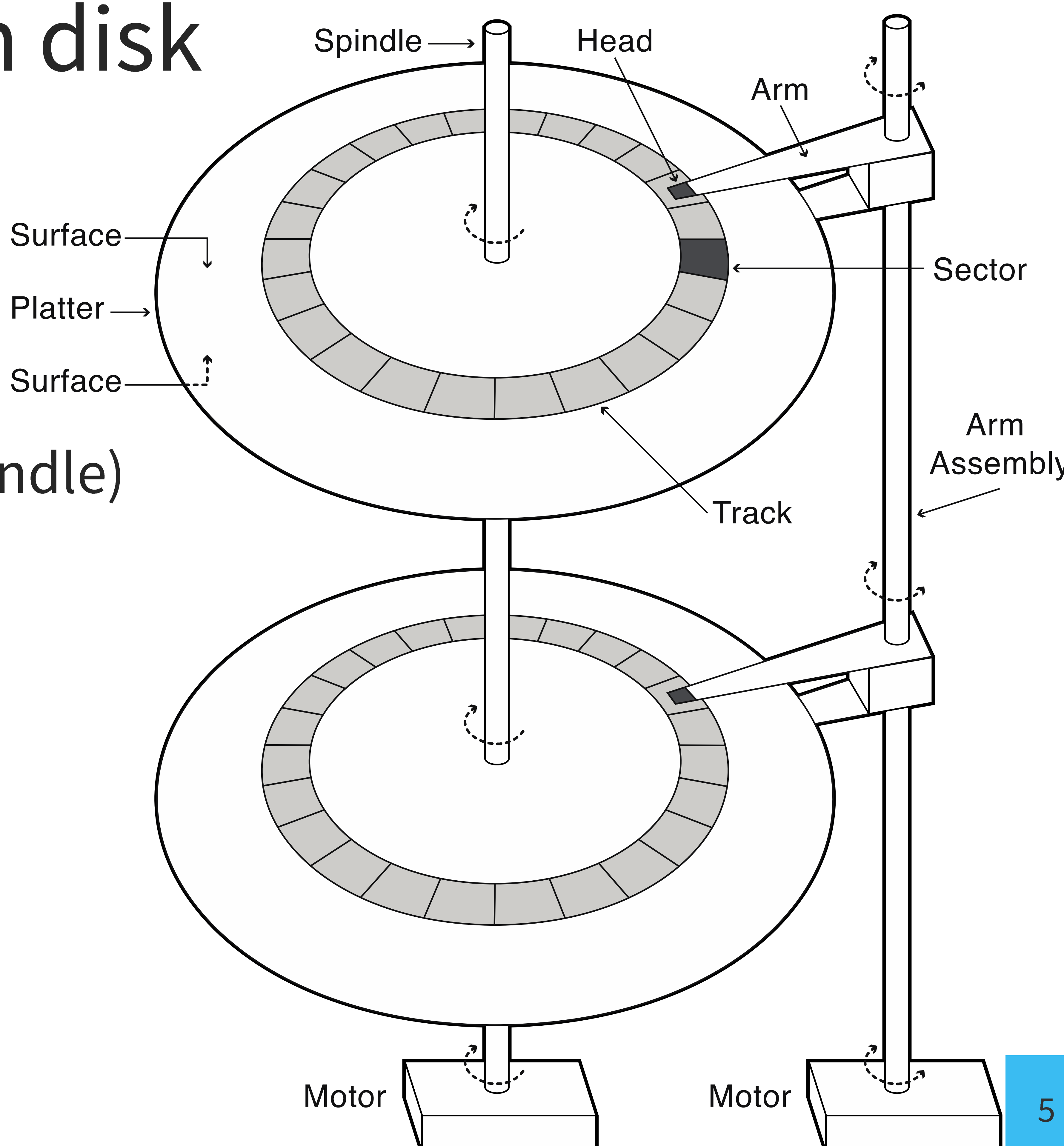
| | RAM | HDD |
|---|---|---|
| **Typical Size** | 8 GB | 1 TB |
| **Cost** | $10 per GB | $0.05 per GB |
| **Power** | 3 W | 2.5 W |
| **Latency** | 15 ns | 15 ms |
| **Throughput (Sequential)** | 8000 MB/s | 175 MB/s |
| **Read/Write Granularity** | word | sector |
| **Power Reliance** | volatile | non-volatile |

[C. Tan, buildcomputers.net, codecapsule.com, crucial.com, wikipedia]

# Reading from disk

Must specify:

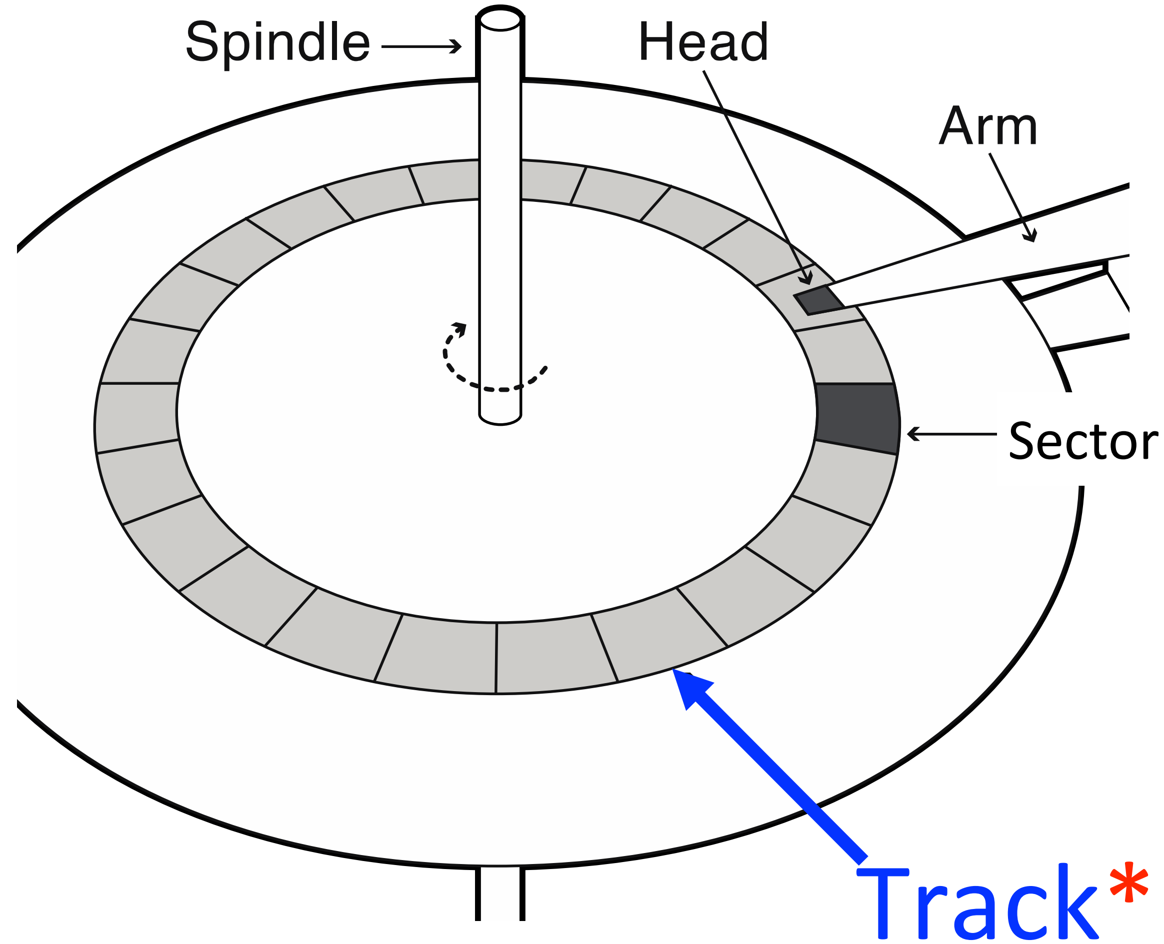
- cylinder # (distance from spindle)
- head #
- sector #
- transfer size
- memory address



# Disk Tracks

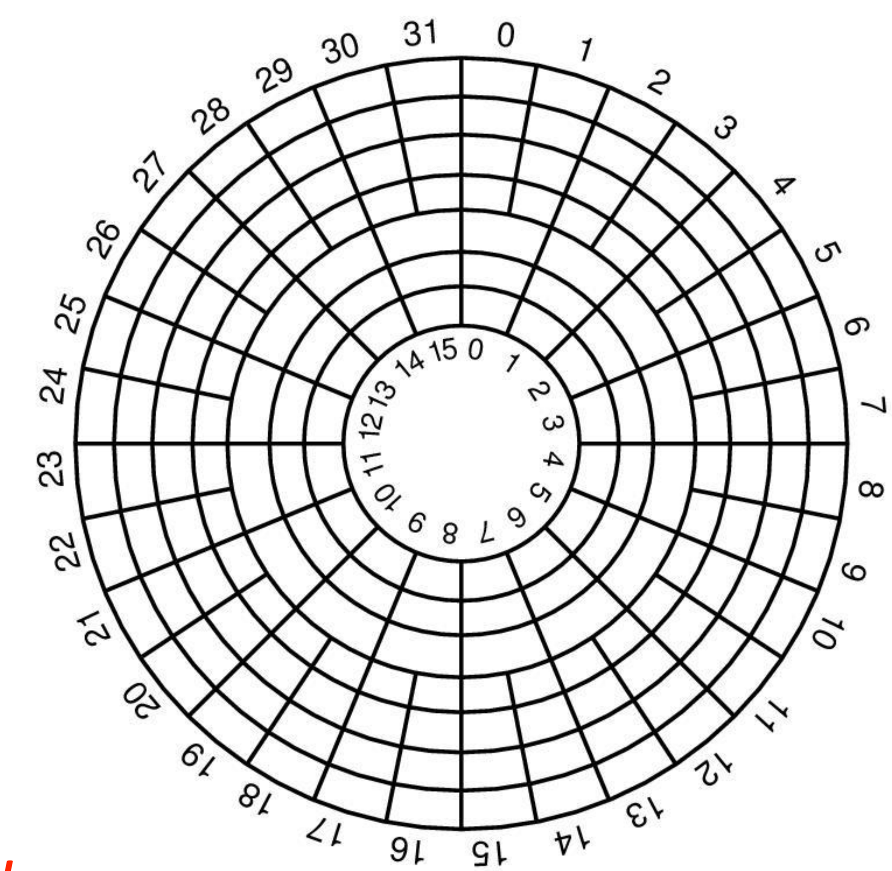
~ 1 micron wide (1000 nm)

- 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

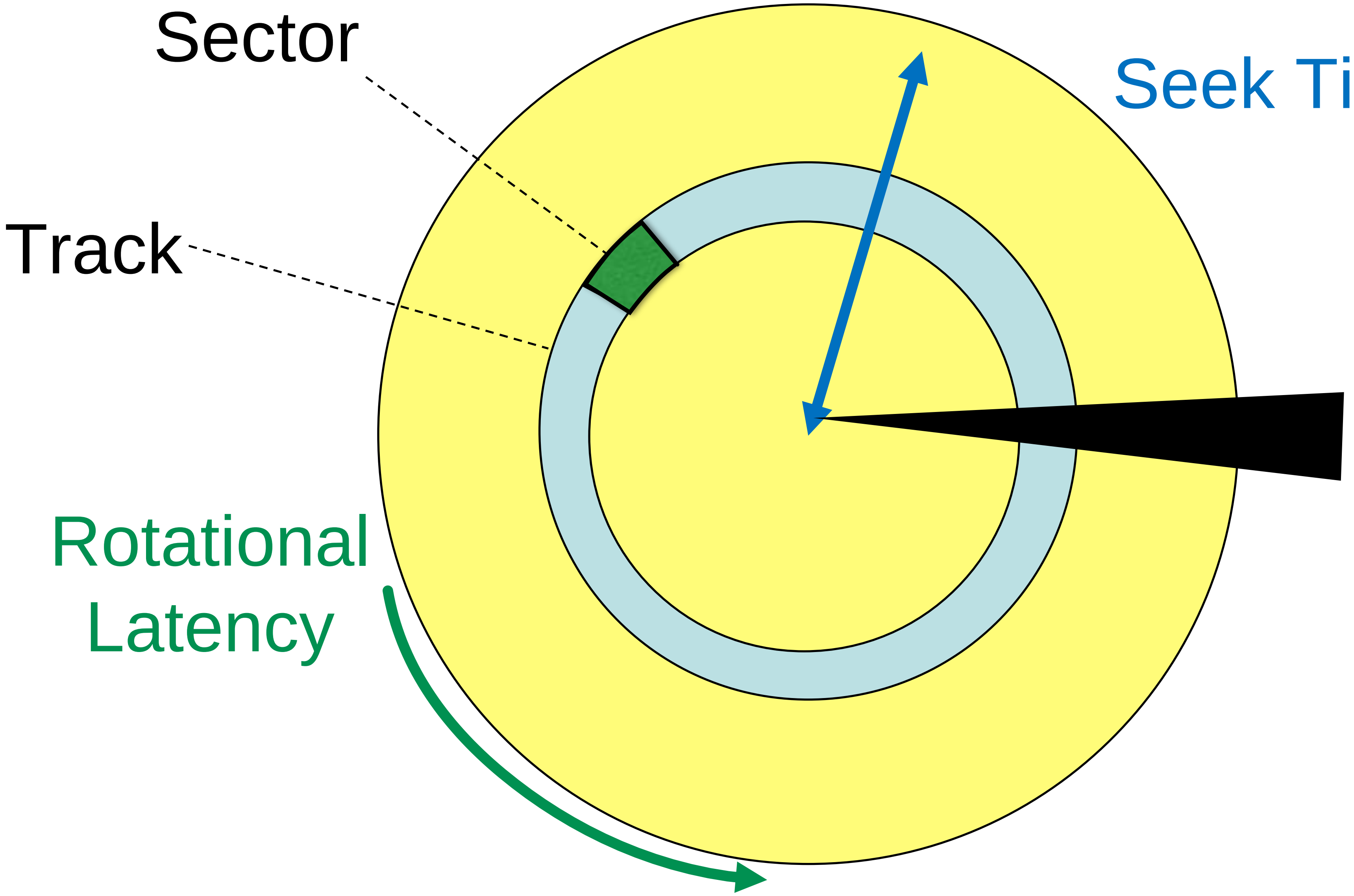



*not to scale: head is actually much bigger than a track*

# 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!

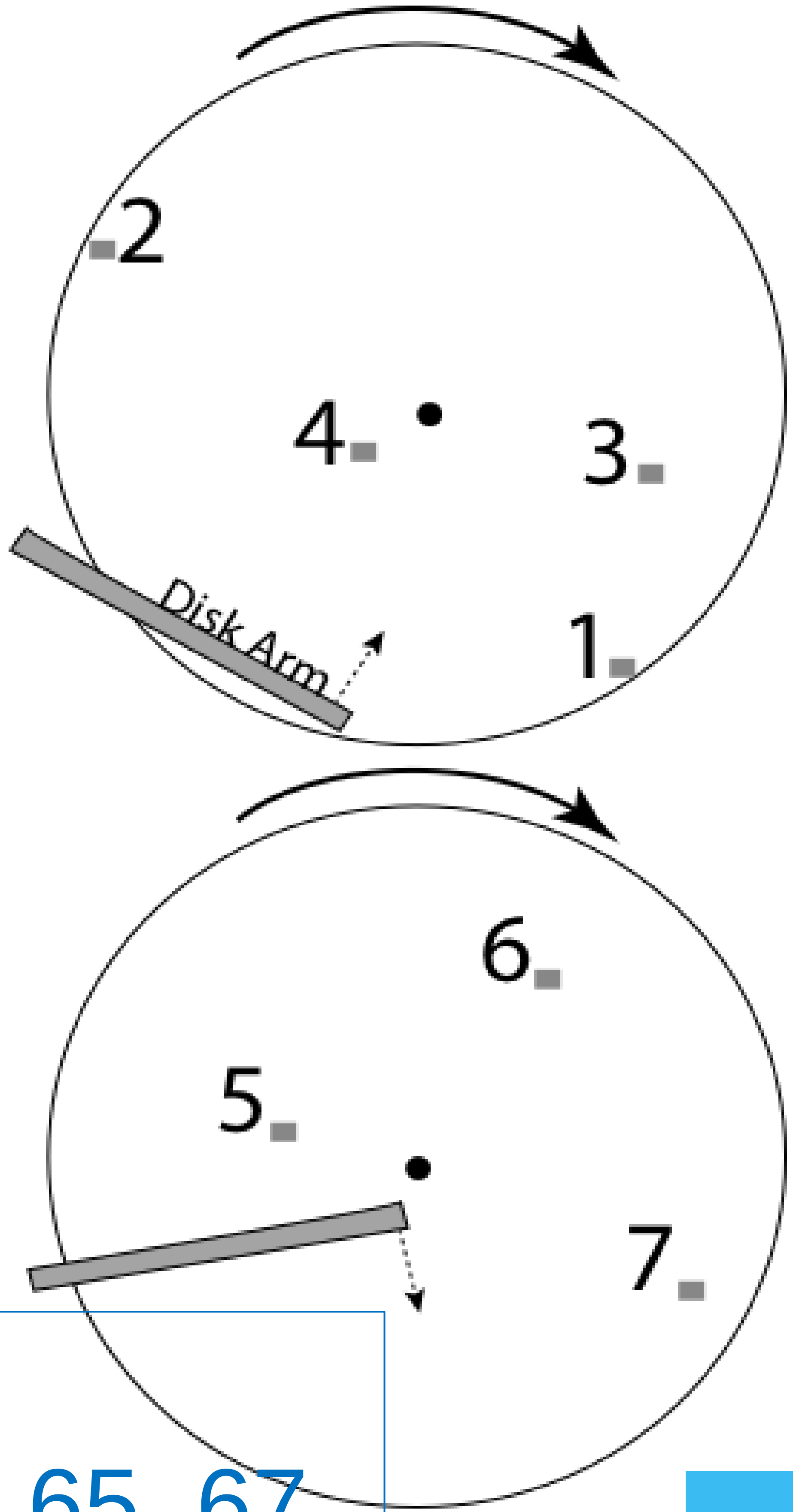
**SSTF Schedule?**
**Total head movement?**

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

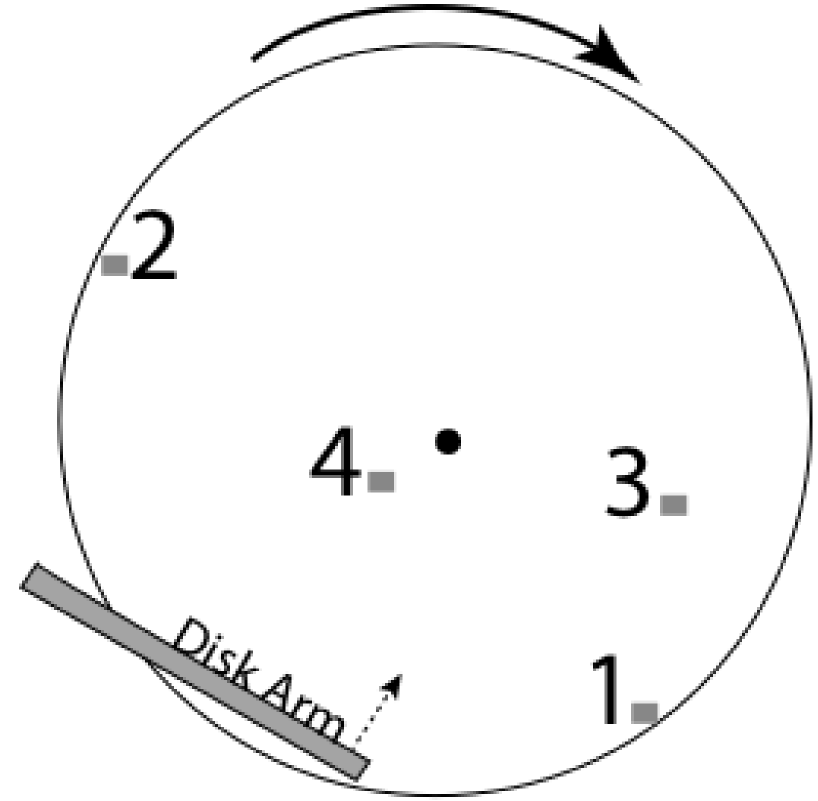
Queue: 98, 183, 37, 122, 14, 124, 65, 67

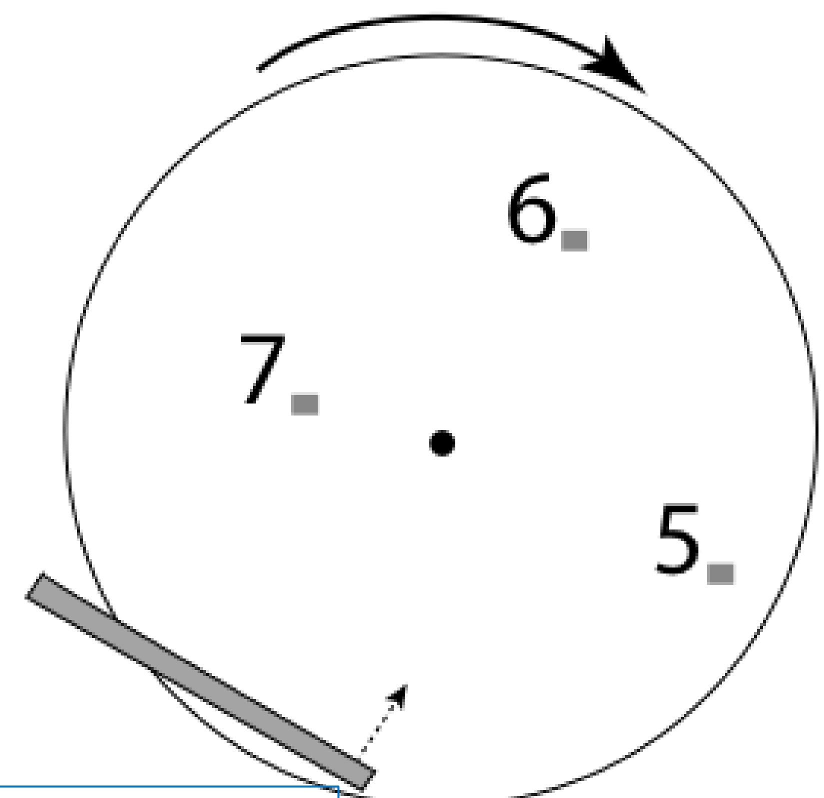
# Disk Scheduling: C-SCAN

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

# Terminology: SCAN vs LOOK

- SCAN: Continue moving head to end of disk, *even if there are no more requests*
  - Extra tracks of movement: from 14 to 0, then back to 65
- LOOK: Reverse direction as soon as there are no more requests in this direction
  - C-LOOK: Reset to beginning as soon as there are no more requests in forward direction
- LOOK versions are what we actually use
- SCAN was easier to implement

# RAM vs. HDD vs SSD, 2018

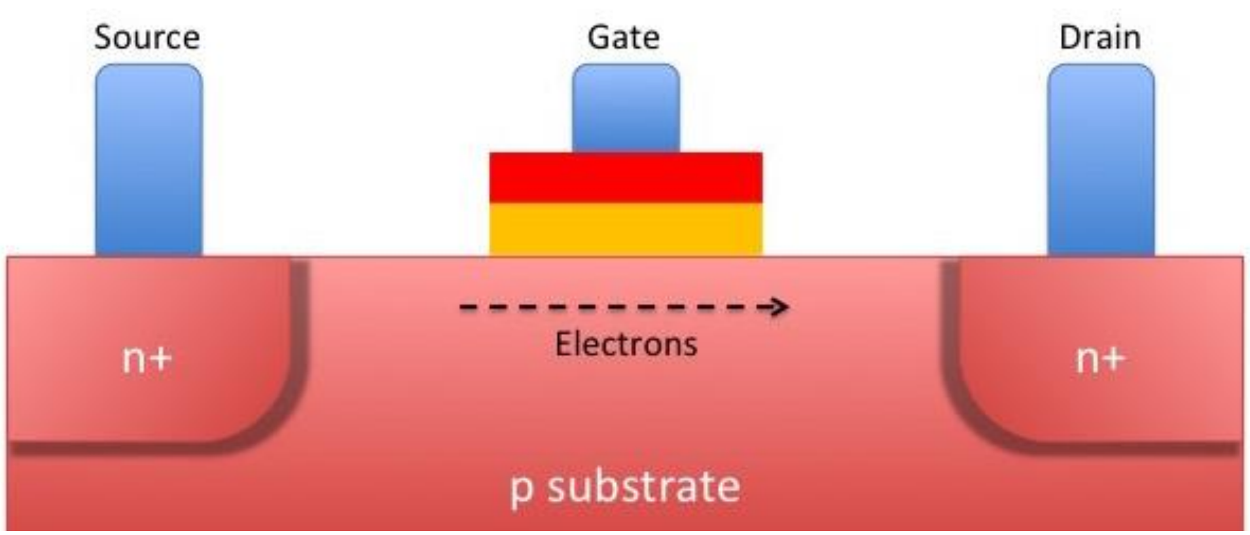
| | RAM | HDD | SSD |
|---|---|---|---|
| **Typical Size** | 8 GB | 1 TB | 256 GB |
| **Cost** | $10 per GB | $0.05 per GB | $0.32 per GB |
| **Power** | 3 W | 2.5 W | 1.5 W |
| **Read Latency** | 15 ns | 15 ms | 30 µs |
| **Read Speed (Seq.)** | 8000 MB/s | 175 MB/s | 550 MB/s |
| **Read/Write Granularity** | word | sector | page* |
| **Power Reliance** | volatile | non-volatile | non-volatile |
| **Write Endurance** | * | ** | 100 TB |

[C. Tan, **buildcomputers.net, codecapsule.com, crucial.com, wikipedia**]

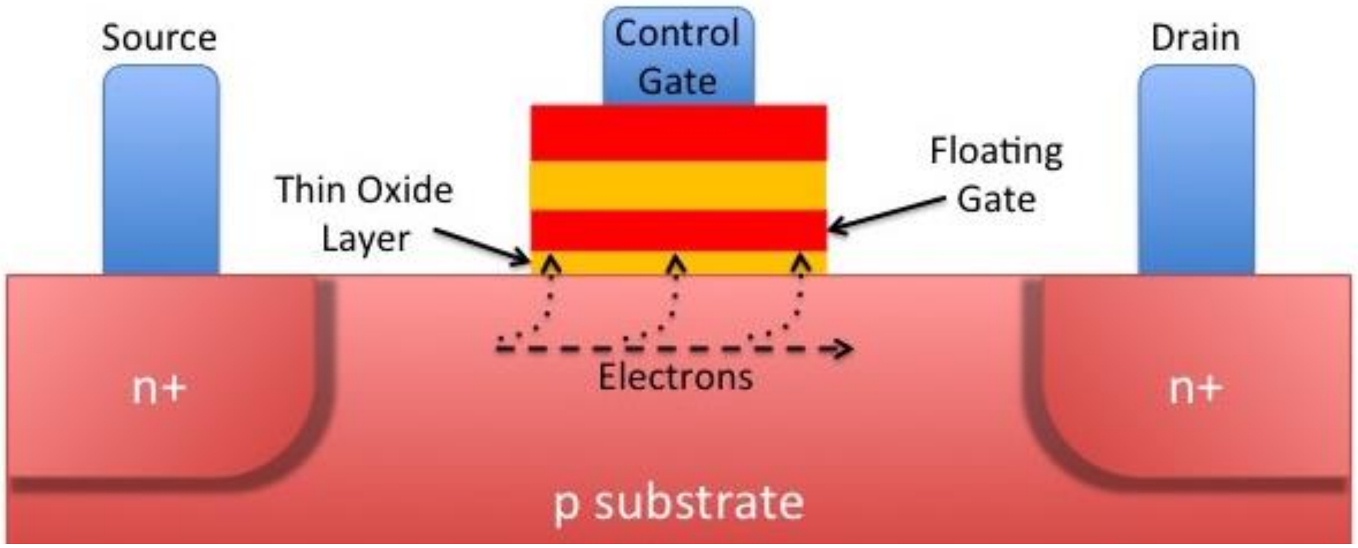
# Solid State Drives (Flash)

## Most SSDs based on NAND-flash

- retains its state for months to years without power




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



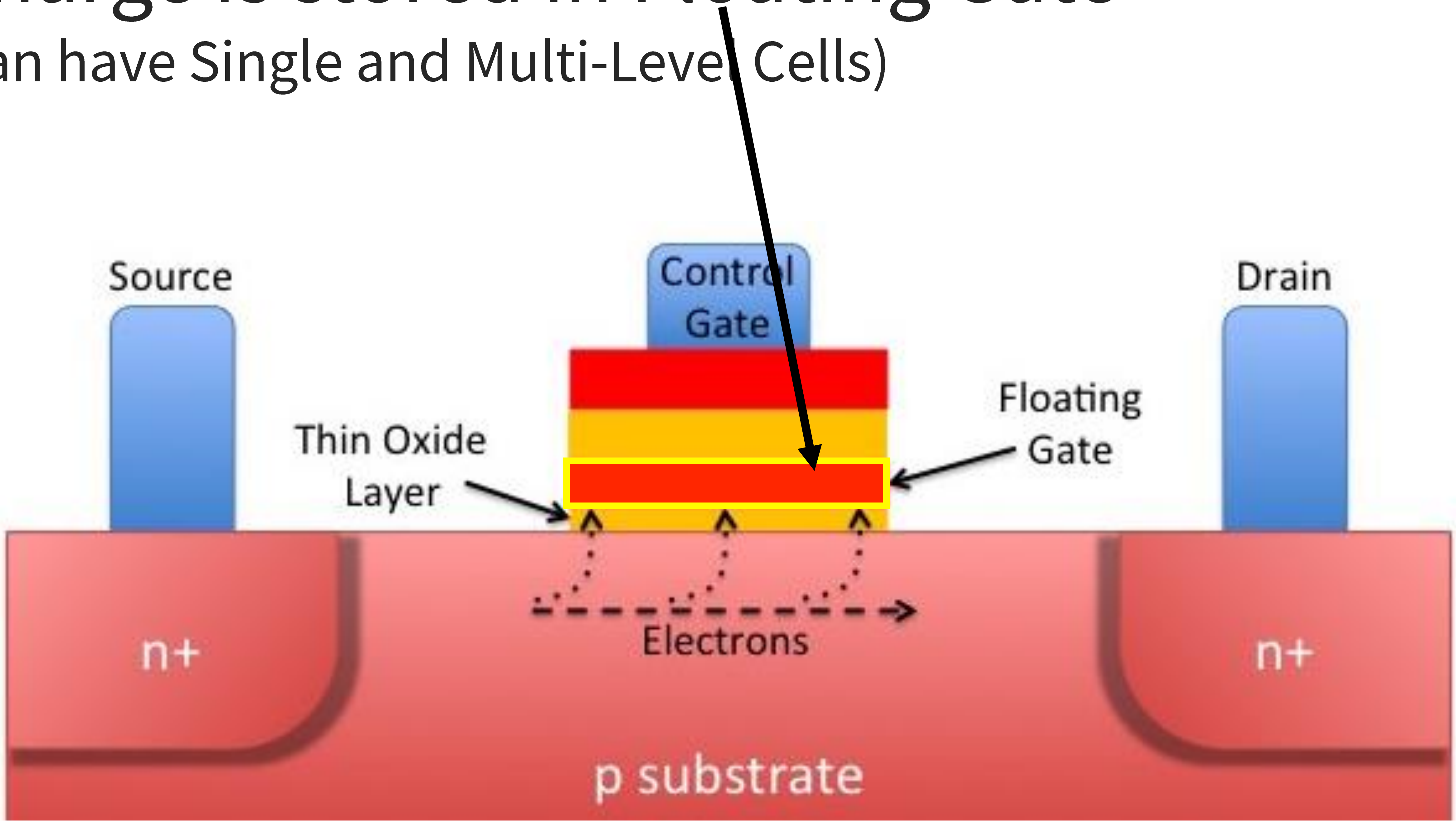

**Floating Gate MOSFET** (FGMOS)

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 page:** can only write erased pages
  - write granularity = 1 page = 2-4KBytes
  - time: 10s of milliseconds
- **Read page:**
  - read granularity = 1 page = 2-4KBytes
  - time: 10s of microseconds

# 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
- 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**

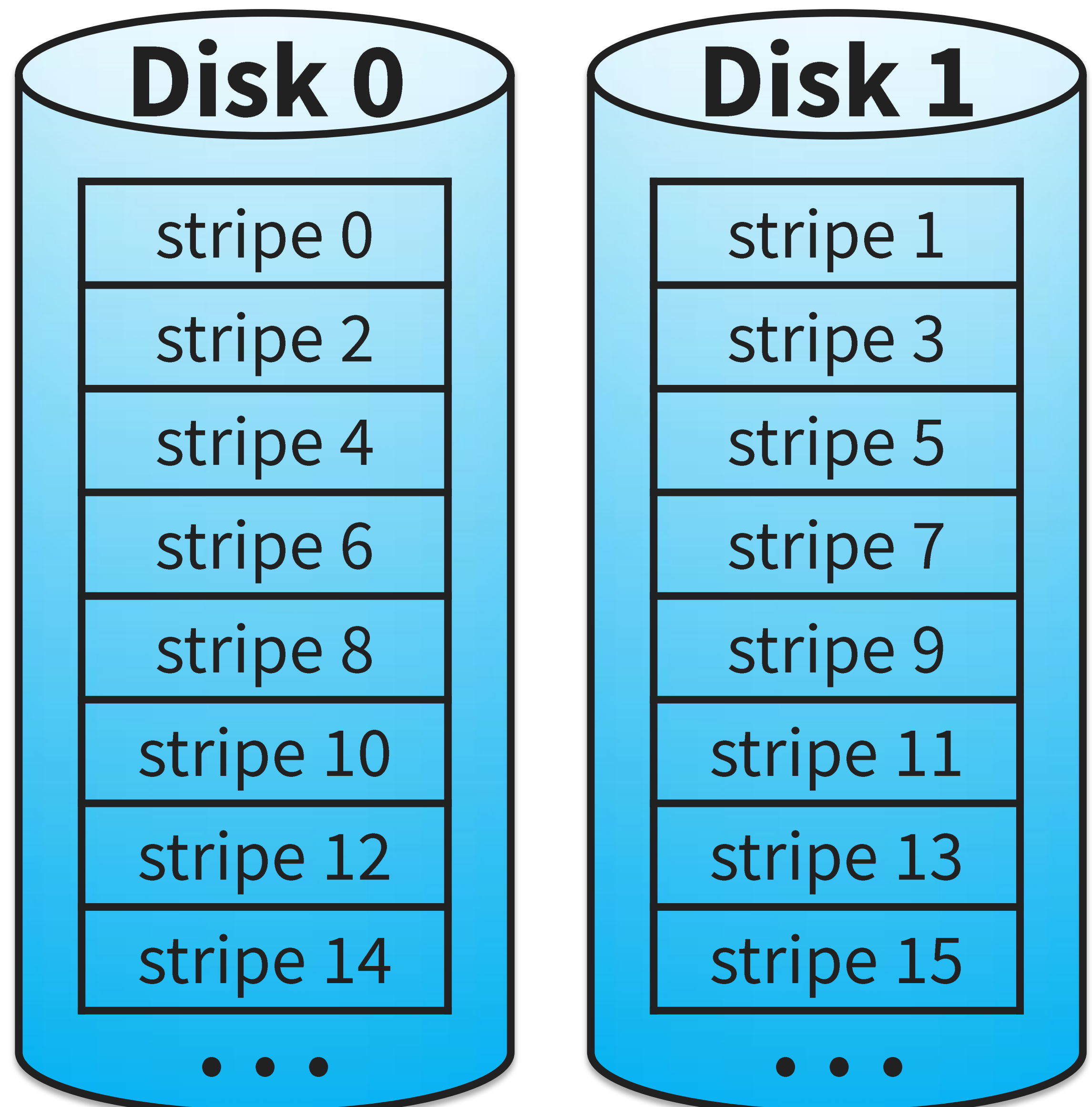
**+ Fast**

**latency?**

**throughput?**

**+ Cheap**

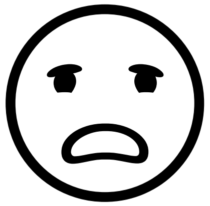
**– Unreliable**

# 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

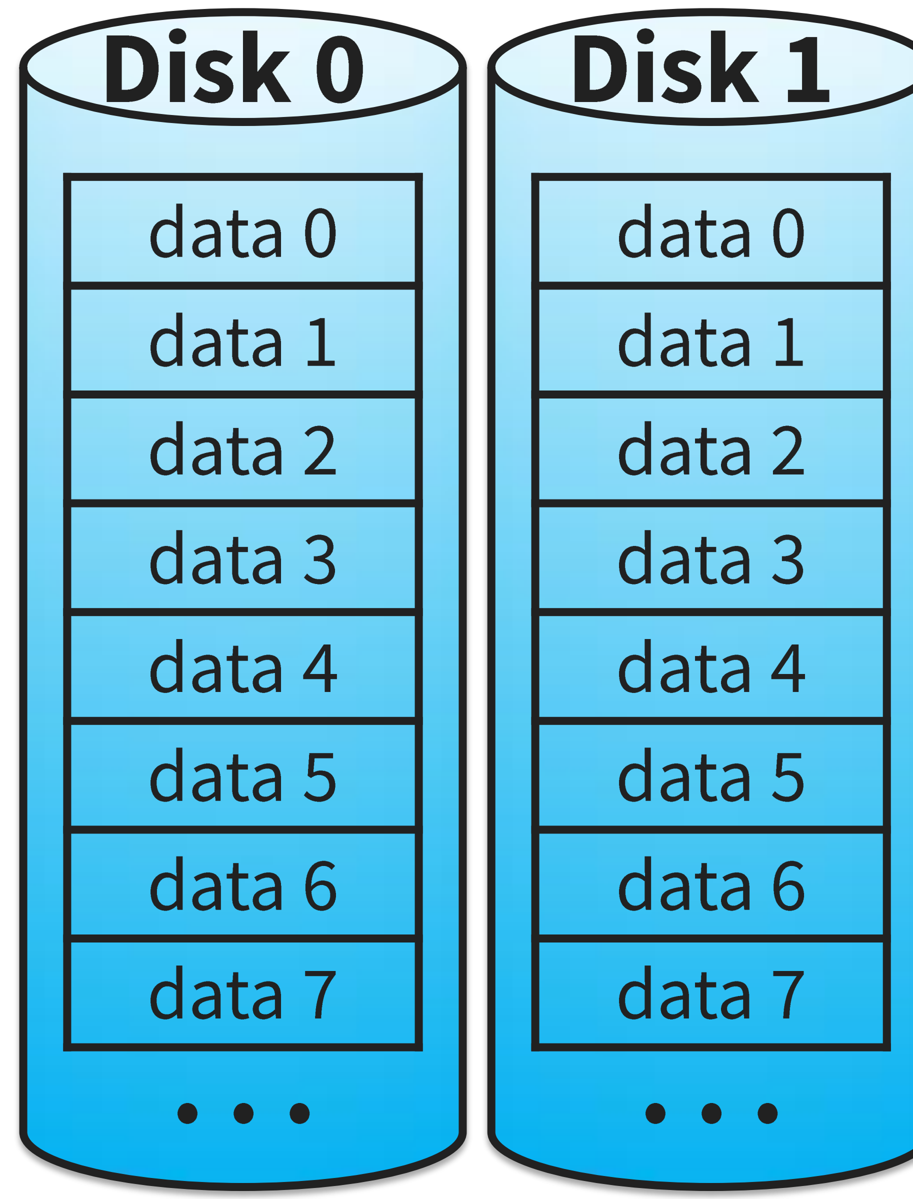

**+ Reliable**

**+ 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

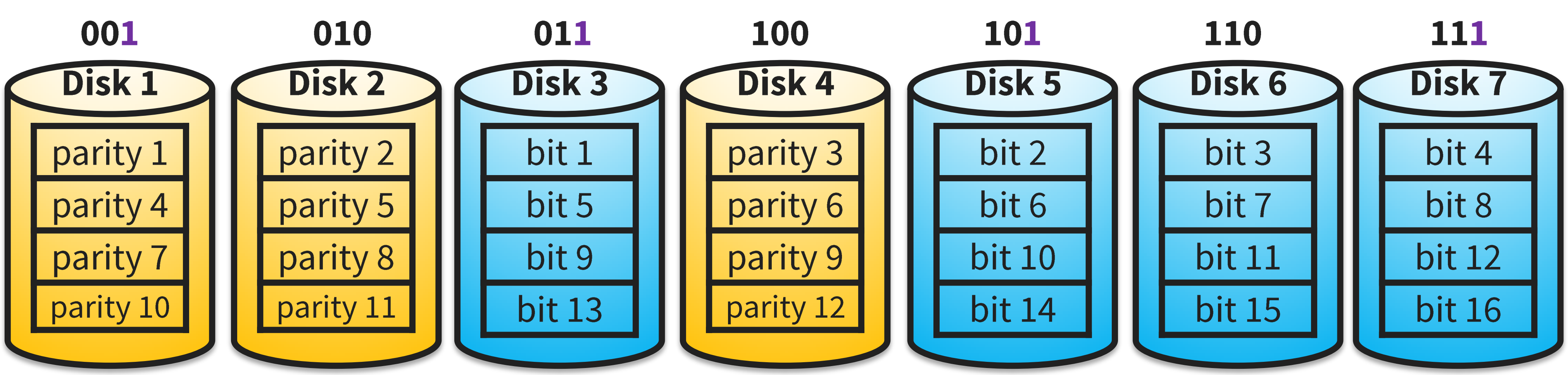
**+ Reliable**

**– 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)

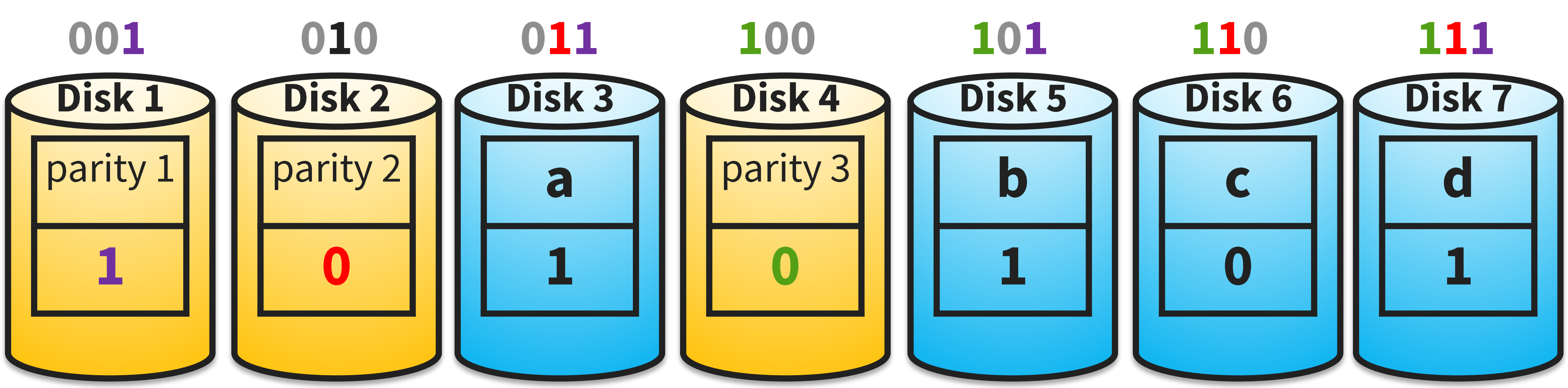


# RAID-2 Generating Parity

**parity 1** $= 3 \oplus 5 \oplus 7$ (all disks whose # has 1 in LSB, xx**1**)
$= a \oplus b \oplus d = 1 \oplus 1 \oplus 1 =$ **1**

**parity 2** $= 3 \oplus 6 \oplus 7$ (all disks whose # has 1 in 2nd bit, x**1**x)
$= a \oplus c \oplus d = 1 \oplus 0 \oplus 1 =$ **0**

**parity 4** $= 5 \oplus 6 \oplus 7$ (all disks whose # has 1 in MSB, **1**xx)
$= b \oplus c \oplus d = 1 \oplus 0 \oplus 1 =$ **0**



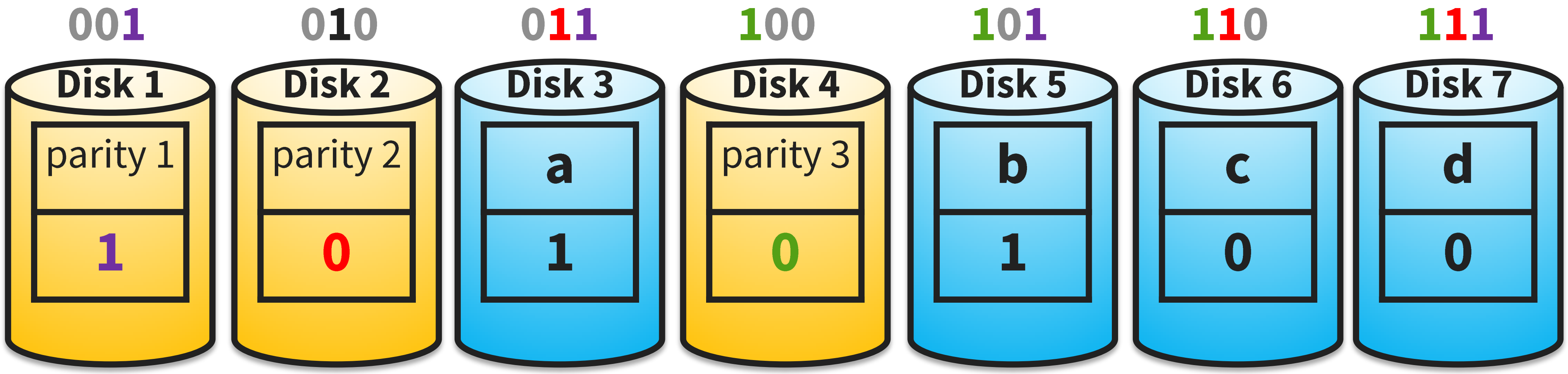
# RAID-2 Detect and Correct

***I flipped a bit. Which one?***

**parity 1** = 3⊕5⊕7 (all disks whose # has 1 in LSB, xx**1**)
= a⊕b⊕d = 1⊕1⊕0 = **0** ← **problem**

**parity 2** = 3⊕6⊕7 (all disks whose # has 1 in 2nd bit, x**1**x)
= a⊕c⊕d = 1⊕0⊕0 = **1** ← **problem**

**parity 4** = 5⊕6⊕7 (all disks whose # has 1 in MSB, **1**xx)
= b⊕c⊕d = 1⊕0⊕0 = **1** ← **problem**




**Problem @ xx1, x1x, 1xx → 111, d was flipped**

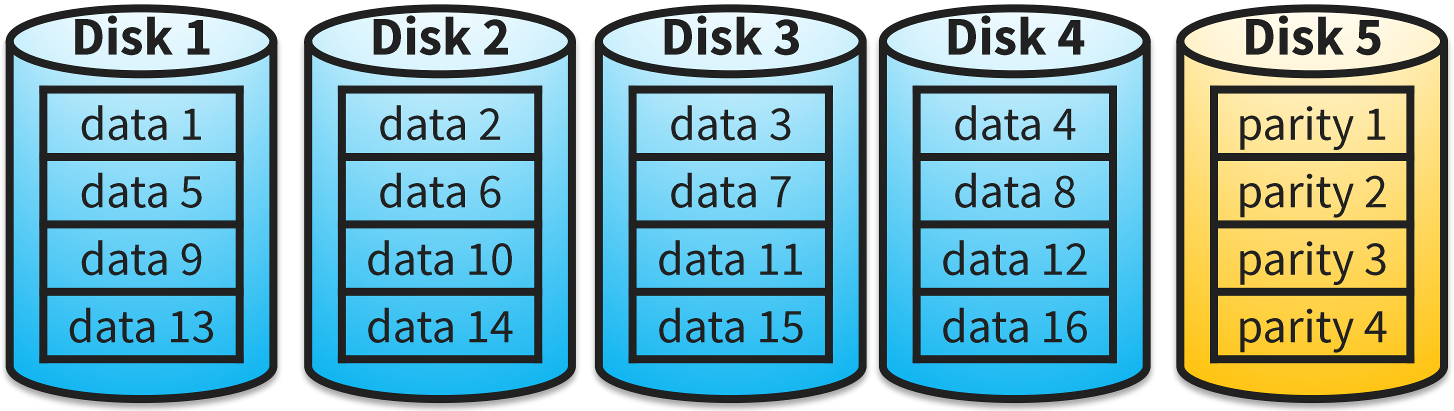
# 2 more rarely-used RAIDS

**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

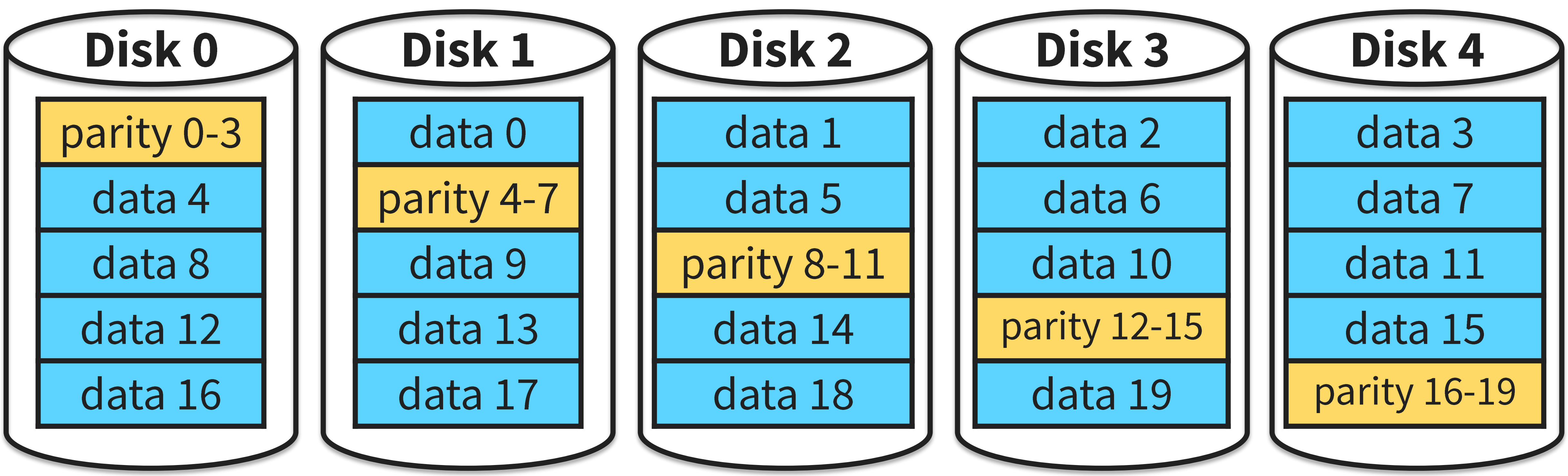
# RAID 5: Rotating Parity w/Striping

**+ Reliable**

**+ Fast**

**+ Affordable**

What if you have 2 simultaneous failures?
(A second failure while recovering from the first?)



# RAID 6: Additional Parity Blocks

- Reed-Solomon Codes: Can recover 2 bits of error

**+ More Reliable**

**+ Fast**

**– Slightly less affordable**