Disks and RAID

CS 4410

Operating Systems

Spring 2017

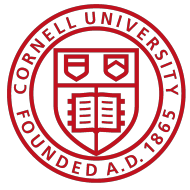
Cornell University

Lorenzo Alvisi

Anne Bracy

See: Ch 12, 14.2 in OSPP textbook

The slides are the product of many rounds of teaching CS 4410 by Professors Sirer, Bracy, Agarwal, George, and 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

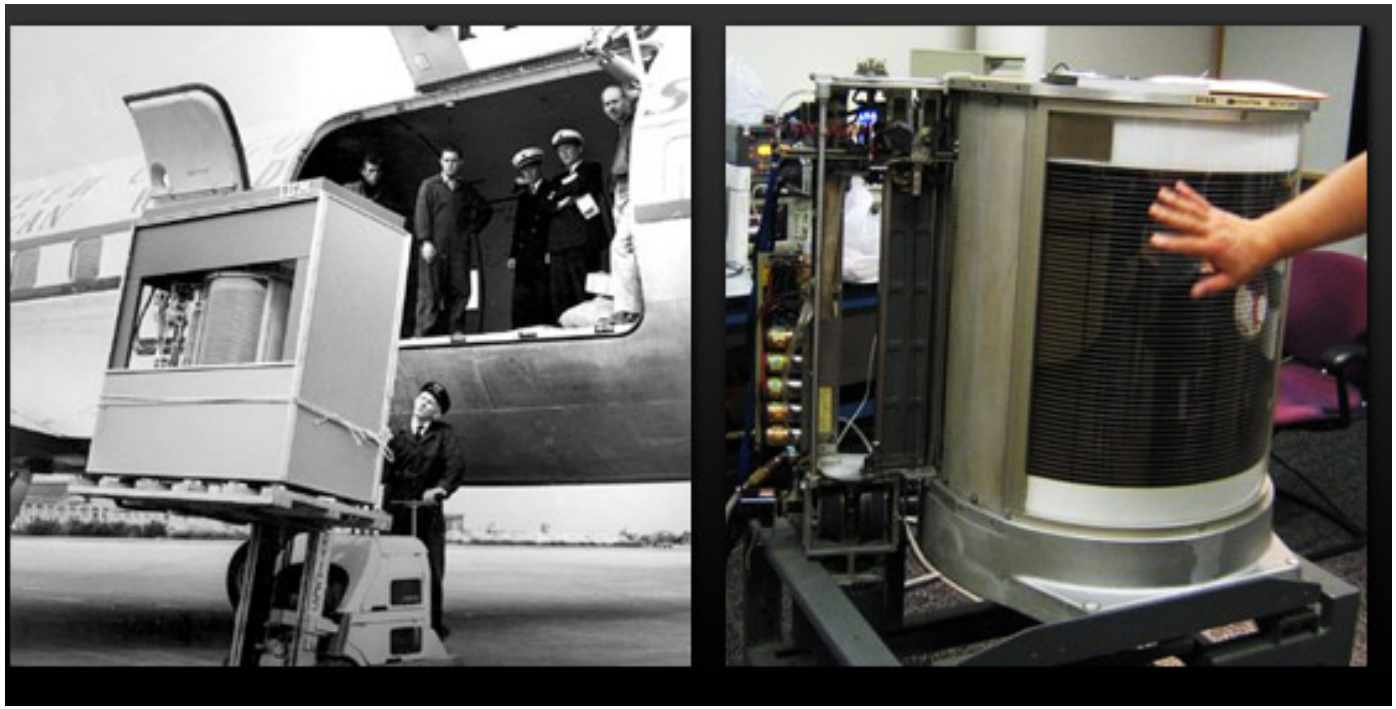
Magnetic Disks are 60 years old!

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

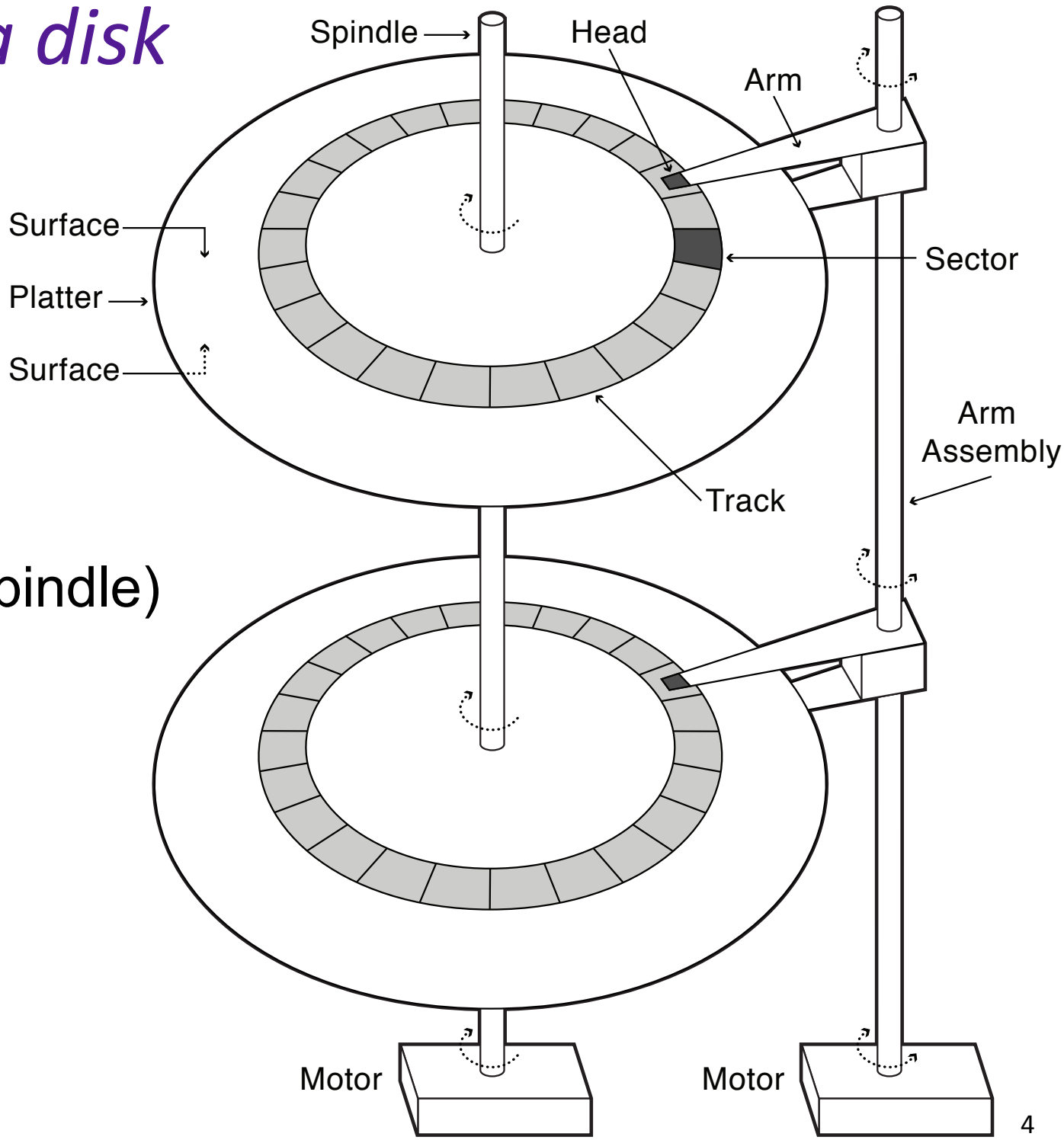


http://royal.pingdom.com/2008/04/08/the-history-of-computer-data-storage-in-pictures/

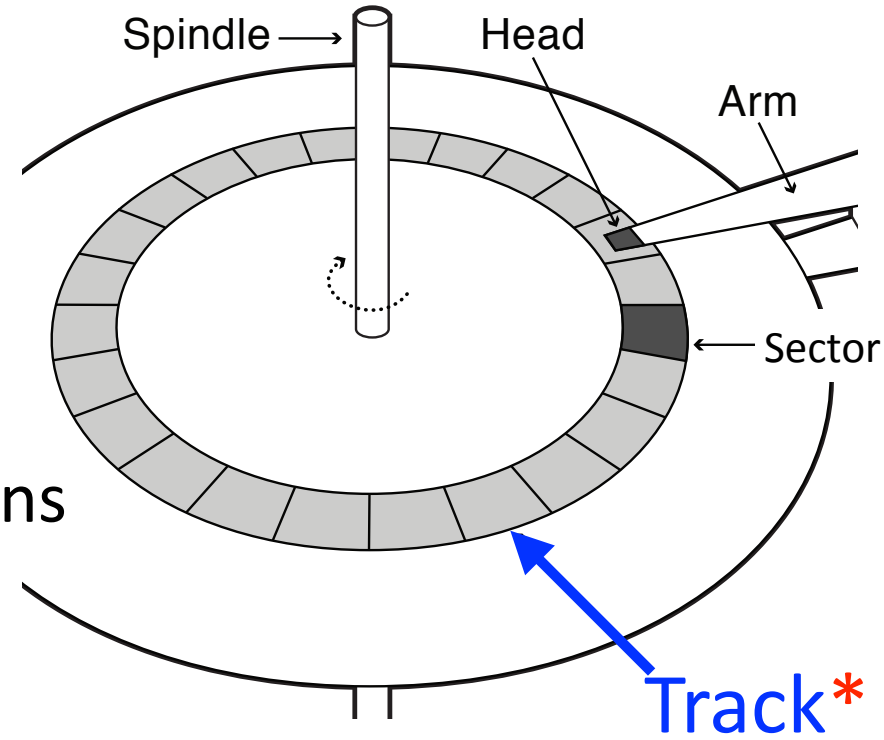
Reading from a disk

Must specify:

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



Disk Tracks

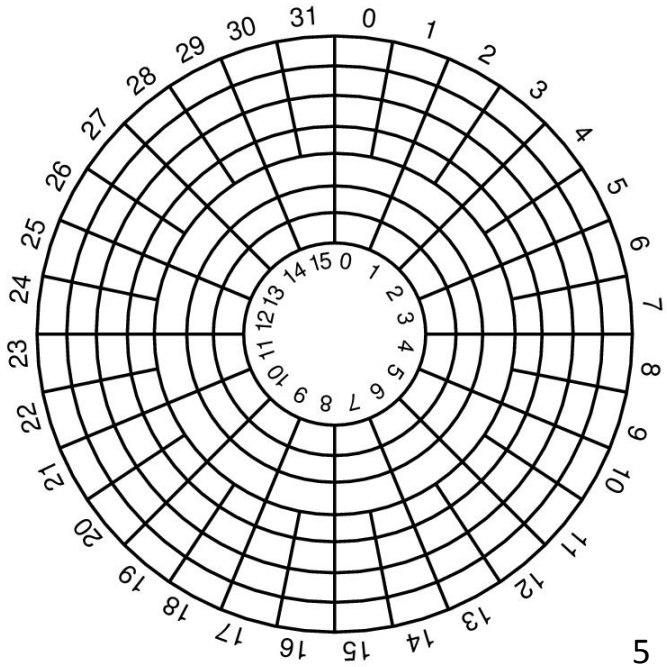


~ 1 micron wide (1000 nm)

- 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 of disk

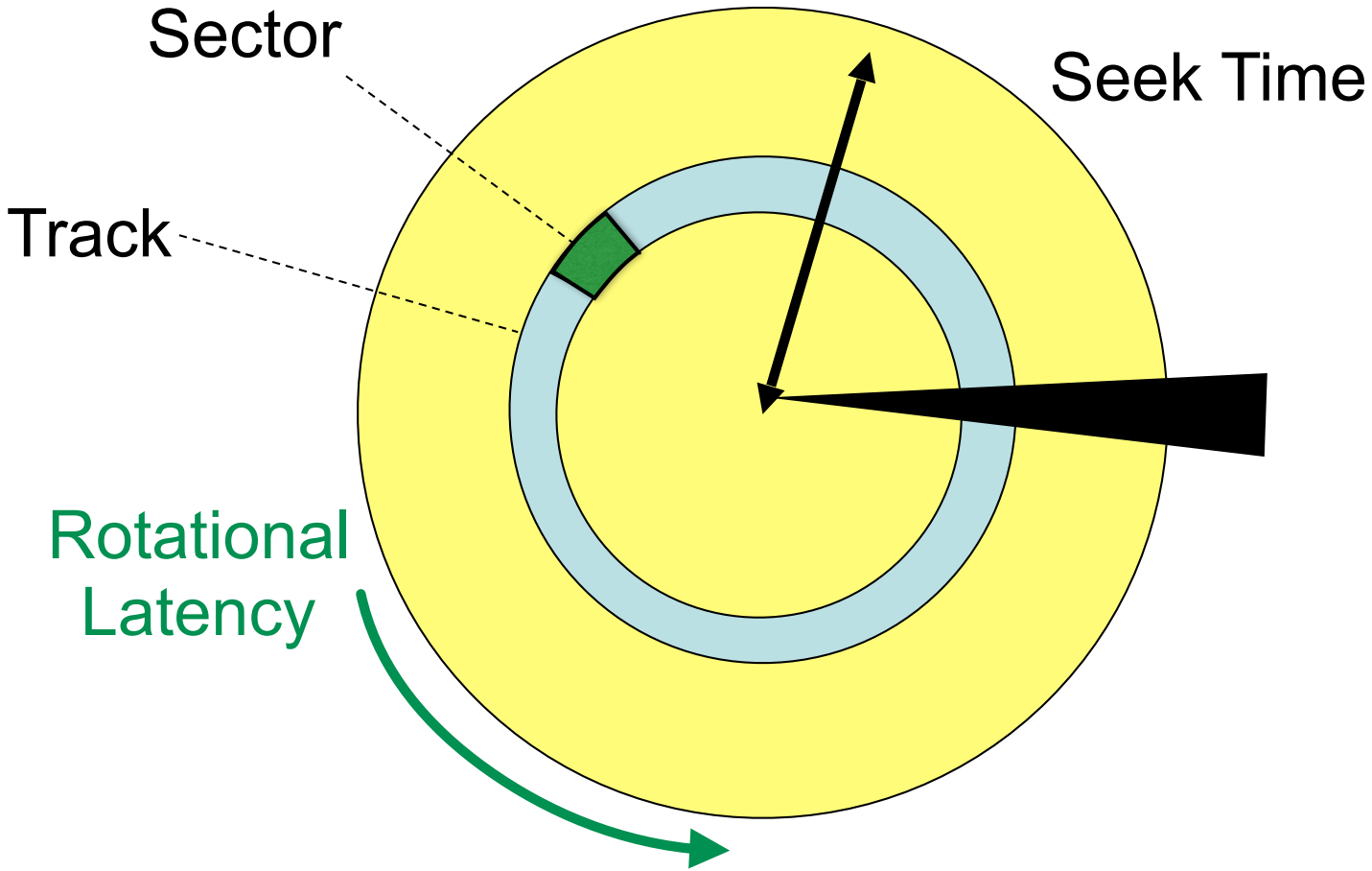


*not to scale: head is actually much bigger than a track

Disk overheads

Disk Latency = Seek Time + Rotation Time + Transfer Time

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



Hard Disks vs. RAM

	Hard Disks	RAM
Smallest write	sector	word
Atomic write	sector	word
Random access	5 ms	10-1000 ns
Sequential access	200 MB/s	200-1000MB/s
Cost	$50 / terabyte	$5 / gigabyte
Power reliance **(survives power outage?)**	Non-volatile (yes)	Volatile (no)

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?

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

FIFO Schedule?

Total head movement?

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!

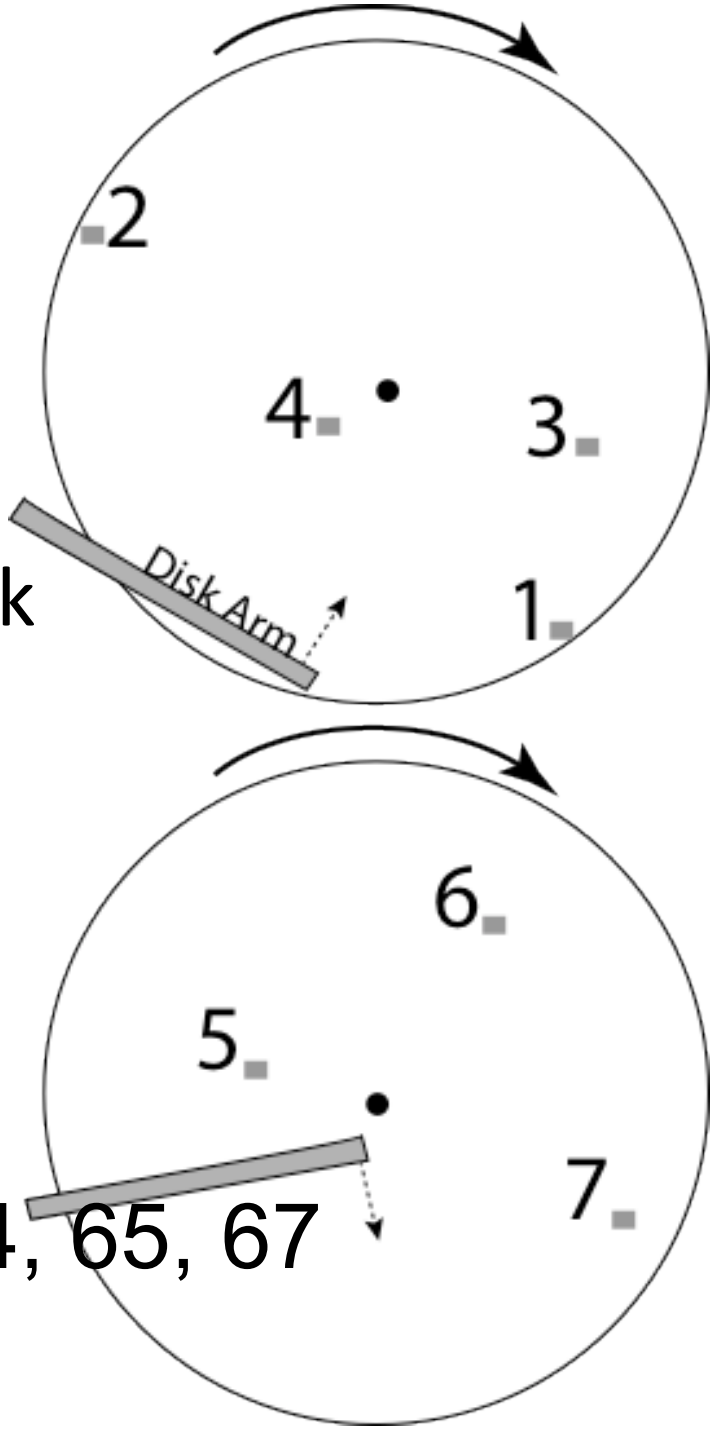
Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

SSTF Schedule?

Total head movement?

Disk Scheduling: SCAN

- Arm starts at one end of disk
  - moves toward other end, servicing requests
  - movement reversed @ end of disk
  - repeat
- AKA elevator algorithm



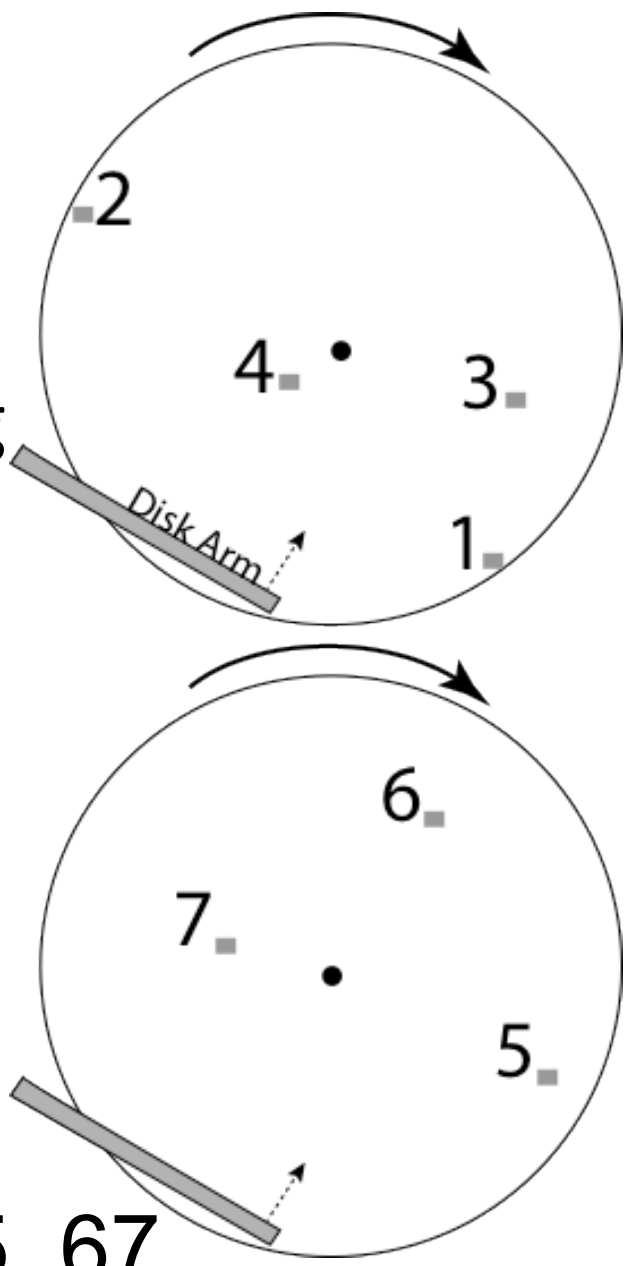
Head pointer @ 53
Queue: 98, 183, 37, 122, 14, 124, 65, 67

SCAN Schedule?

Total head movement?

Disk Scheduling: C-SCAN

- Head moves from one end to other
  - servicing requests as it goes
  - reaches the end, returns to beginning
  - No requests serviced on return trip
- Treats cylinders as a circular list
  - wraps around from last to first
- More uniform wait time than SCAN



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

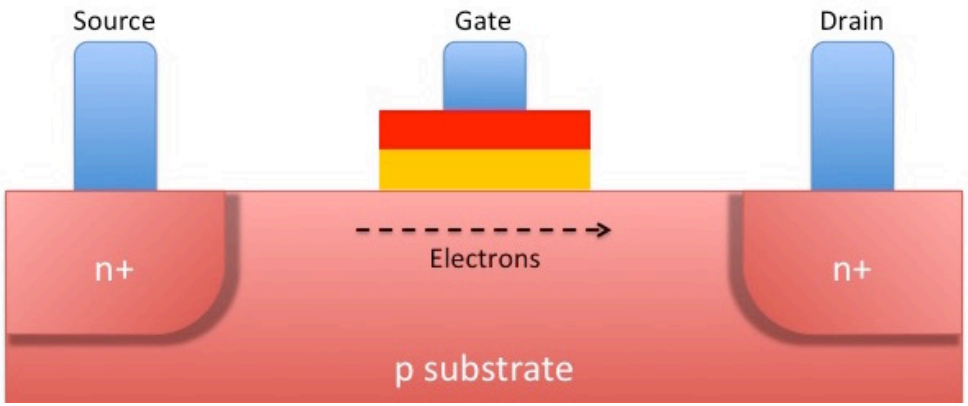
C-SCAN Schedule?

Total Head movement?

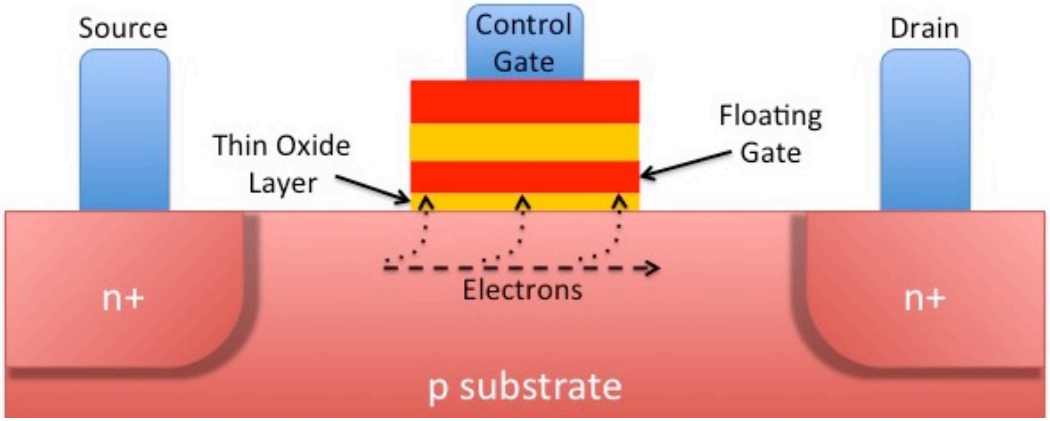
Solid State Drives (Flash)

Most SSDs based on NAND-flash

- retains its state for months to years without power



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



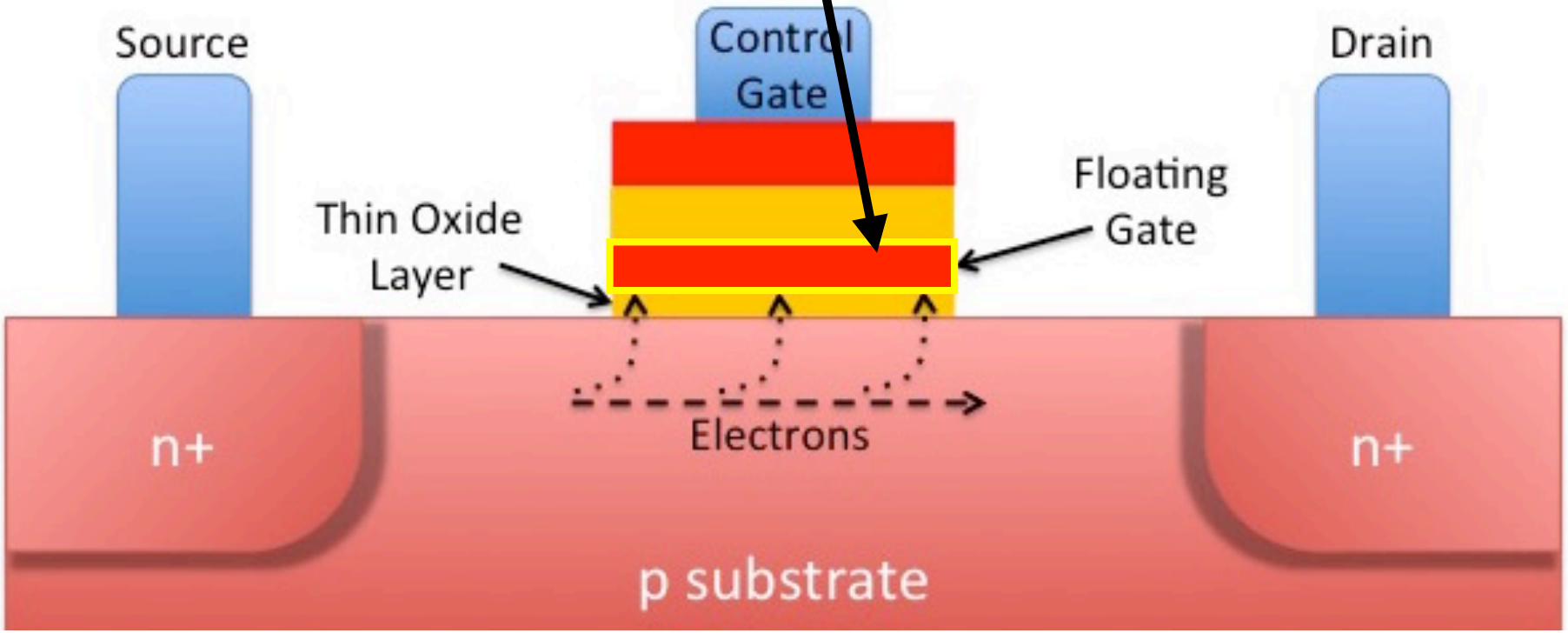
Floating Gate MOSFET (FGMOS)

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 ms

Read page:

- read granularity = 1 page = 2-4KBytes
- time: 10s of ms

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

SSD vs HDD

	SSD	HDD
Cost	10cts/gig	6cts/gig
Power	2-3W	6-7W
Typical Capacity	1TB	2TB
Write Speed	250MB/sec	200MB/sec
Read Speed	700MB/sec	200MB/sec

What do we want?

Performance: keeping up with the CPU

- CPU 2x faster every 2 years (until recently)
- Disks 20x faster *in 3 decades*

What can we do to improve Disk Performance?

Hint #1: Disks did get cheaper in the past 3 decades…

Hint #2: When CPUs stopped getting faster, we also did this…

RAID, Step 0: Striping

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!*)

GOALS:

1. Performance
   - Parallelize individual requests
   - Support parallel requests

TECHNIQUES:

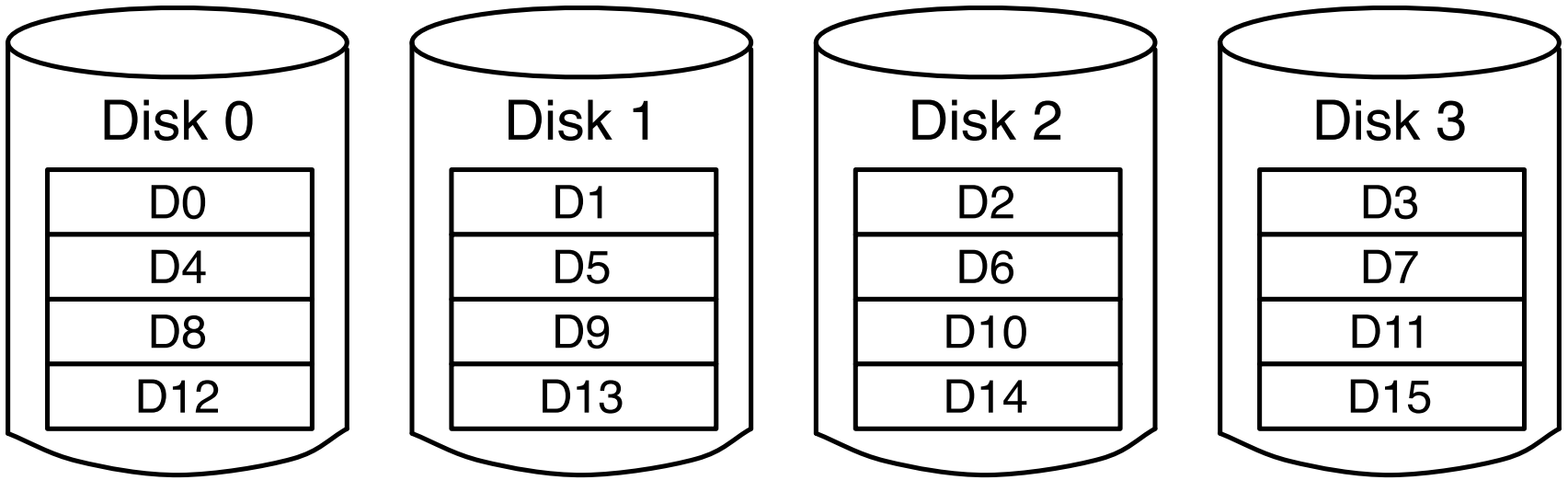
0. Striping

RAID-0

Files striped across disks

- **Read:** high throughput (parallel I/O)
- **Write:** best throughput

Downsides?



What could possibly go wrong?

Failure can occur for:

(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, mechanical wear out
- Detected by device driver, accesses return error codes
- annual failure rates or Mean Time To Failure (MTTF)

*What do we **also** want?*

Reliability: data fetched is what you stored

Availability: data is there when you want it

- More disks ➜ higher probability of some disk failing
- Striping reduces reliability 😞
  - N disks: 1/nth mean time between failures of 1 disk

What can we do to improve Disk Reliability?

Hint #1: When CPUs stopped being reliable, we also did this…

RAID, Step 1: Mirroring

To improve reliability, add *redundancy*

GOALS:

1. Performance
   - Parallelize individual requests
   - Support parallel requests
2. Reliability

TECHNIQUES:

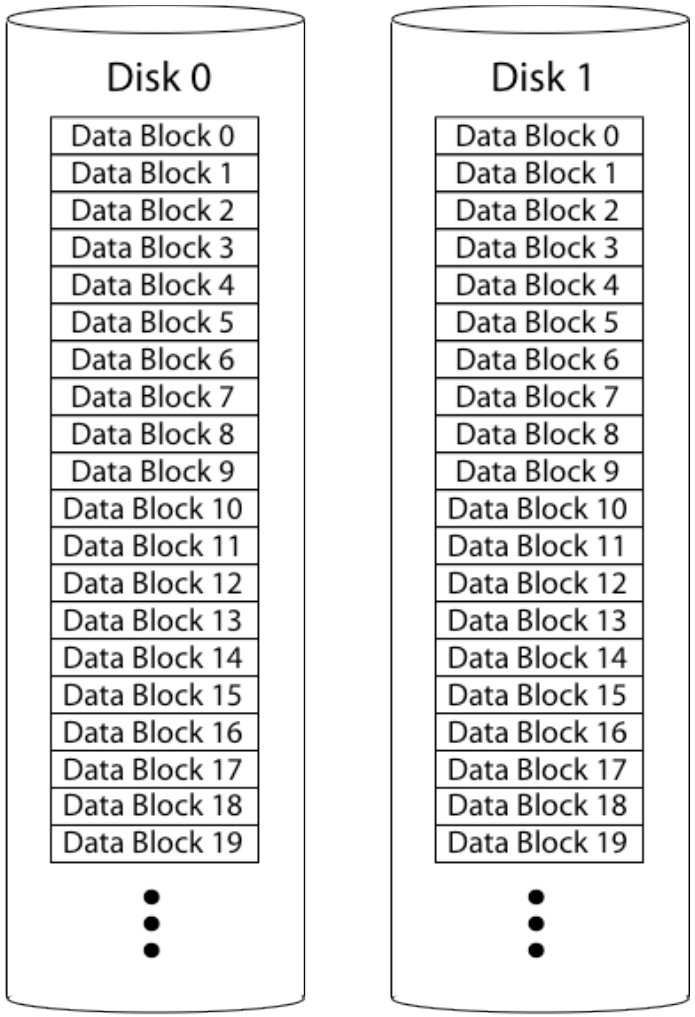
0. Striping
1. Mirroring

RAID-1

Disks Mirrored: data written in 2 places

Simple, expensive

Example: Google File System replicated data on 3 disks, spread across multiple racks



Reads: go to either disk

➜ 2x faster than SLED

- **Write:** replicate to every mirrored disk

➜ same speed as SLED

Full Disk Failure: use surviving disk

Bit Flip Error: Detect? Correct?

RAID, Step 2: Parity

To recover from failures, add *parity*

- *n-input XOR gives bit-level parity (1 = odd, 0 = even)*
- 1101 ⊕ 1100 ⊕ 0110 = 0111 (parity block)
- Can reconstruct any missing block from the others

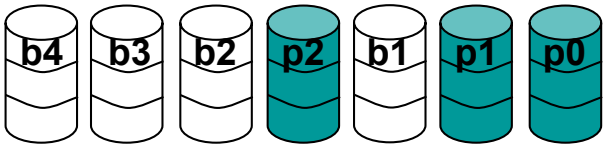
GOALS:

1. Performance
   - Parallelize individual requests
   - Support parallel requests
2. Reliability

TECHNIQUES:

0. Striping
1. Mirroring
2. Parity

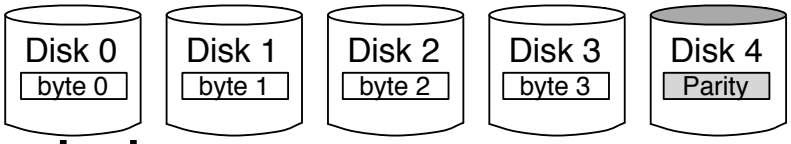
Lesser Loved RAIDS



RAID-2: bit-level striping with ECC codes

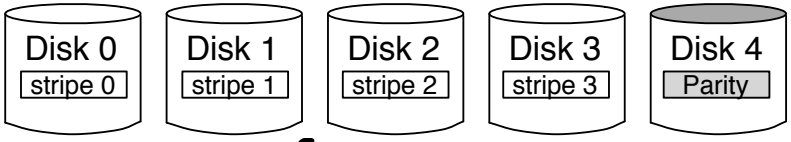
- 7 disk arms synchronized and move in unison
- Complicated controller (and hence very unpopular)
- Tolerates 1 error with no performance degradation

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
- parity disk is write bottleneck and wears out faster

A word about Granularity

Bit-level ➜ byte-level ➜ block level

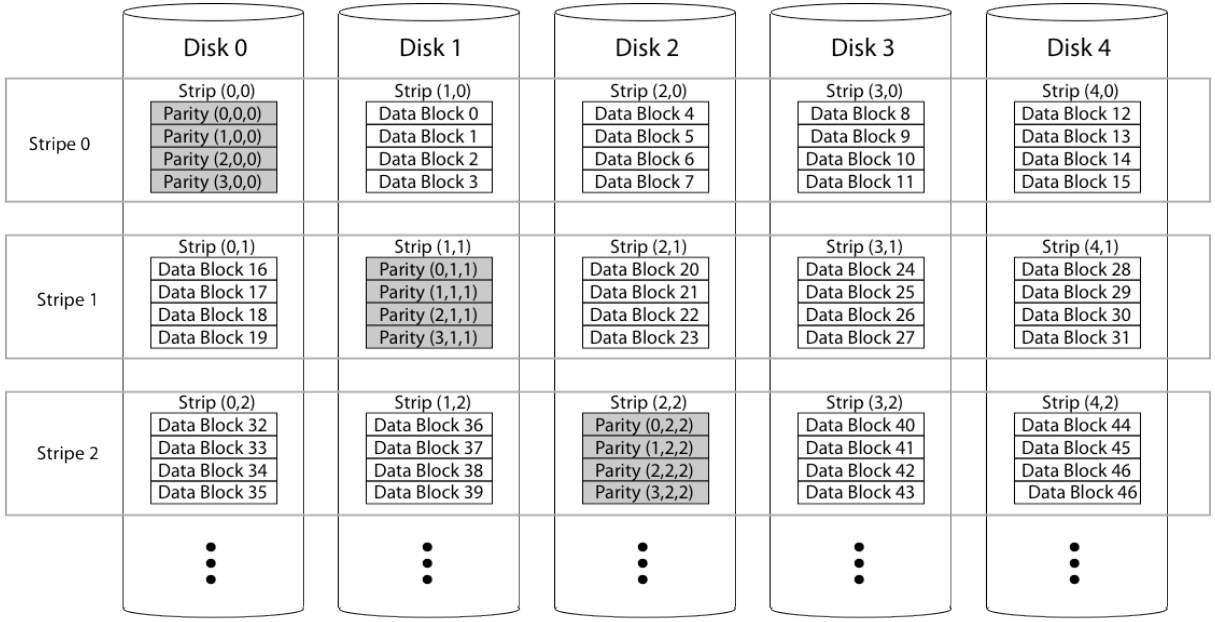
- *fine-grained:* Stripe each file across all disks
  - \+ high throughput for the file
  - \- wasted disk seek time
  - \- limits to transfer of 1 file at a time
- *coarse-grained:* Stripe each file over a few disks
  - \- limits throughput for 1 file
  - \+ better use of spatial locality (for disk seek)
  - \+ allows more parallel file access

RAID 5: Rotating Parity w/Striping

	Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
Stripe 0	Strip (0,0) Parity (0,0,0) Parity (1,0,0) Parity (2,0,0) Parity (3,0,0)	Strip (1,0) Data Block 0 Data Block 1 Data Block 2 Data Block 3	Strip (2,0) Data Block 4 Data Block 5 Data Block 6 Data Block 7	Strip (3,0) Data Block 8 Data Block 9 Data Block 10 Data Block 11	Strip (4,0) Data Block 12 Data Block 13 Data Block 14 Data Block 15
Stripe 1	Strip (0,1) Data Block 16 Data Block 17 Data Block 18 Data Block 19	Strip (1,1) Parity (0,1,1) Parity (1,1,1) Parity (2,1,1) Parity (3,1,1)	Strip (2,1) Data Block 20 Data Block 21 Data Block 22 Data Block 23	Strip (3,1) Data Block 24 Data Block 25 Data Block 26 Data Block 27	Strip (4,1) Data Block 28 Data Block 29 Data Block 30 Data Block 31
Stripe 2	Strip (0,2) Data Block 32 Data Block 33 Data Block 34 Data Block 35	Strip (1,2) Data Block 36 Data Block 37 Data Block 38 Data Block 39	Strip (2,2) Parity (0,2,2) Parity (1,2,2) Parity (2,2,2) Parity (3,2,2)	Strip (3,2) Data Block 40 Data Block 41 Data Block 42 Data Block 43	Strip (4,2) Data Block 44 Data Block 45 Data Block 46 Data Block 46
	⋮	⋮	⋮	⋮	⋮

RAID 5: Rotating Parity w/Striping

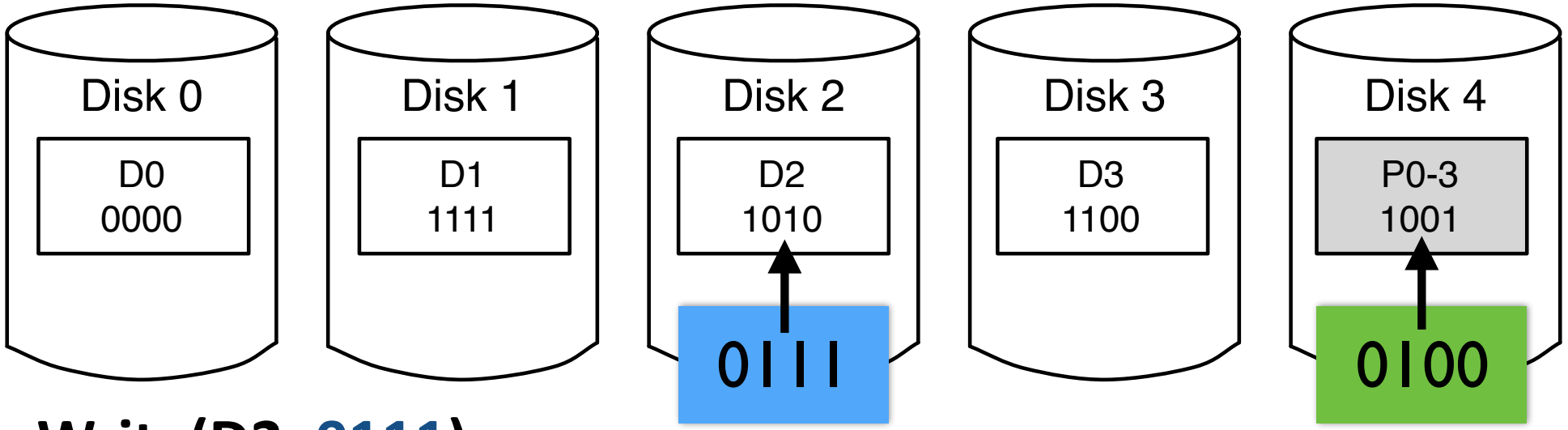
- **Read:** go to correct disk, can outperform SLEDs and RAID-0



- **Write 1 block:**
  - Read old data block
  - Read old parity block
  - Write new data block
  - Write new parity block (old data ⊕ old parity ⊕ new data)
- **Write entire stripe:**
  - Write data blocks and parity for each strip in stripe

Good write performance

RAID 5: Write Example



- **Write(D2, 0111)**
  - Read old data block (1010)
  - Read old parity block (1001)
  - Write new data block (0111)
  - Write new parity block

    (old data ⊕ old parity ⊕ new data)

    1010 ⊕ 1001 ⊕ 0111 = **0100**