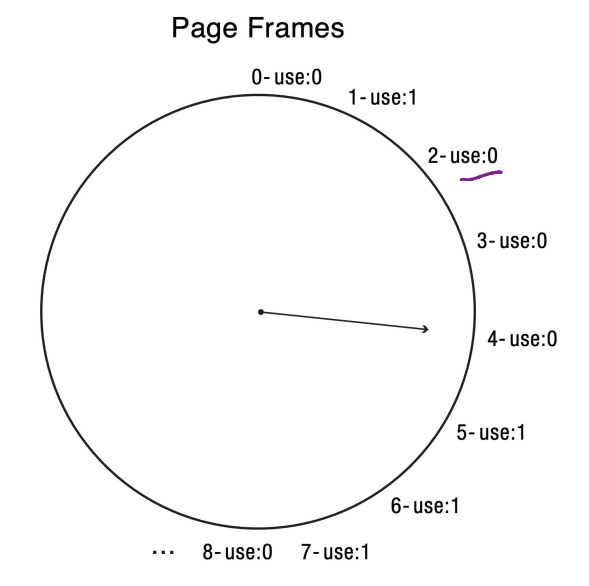
11/9 Eviction Wrap up & Storage devices

LRU => Clock Clock doesn't consider the cost of evicting frames.

→ clean copy on disk (code page, unchanged data)



Second Chance / Enhanced Clock

→ access bit, dirty bit (1 if written to)

→ A, D

A	D	
0	0	(if hasn't been accessed recently, it's a clean page)
1	0	(clear the access bit)
0	1	(clear the dirty bit, move on)
1	1	→ (dirty page bit) (clear access bit & move on)

Storage Devices (I/O Devices)

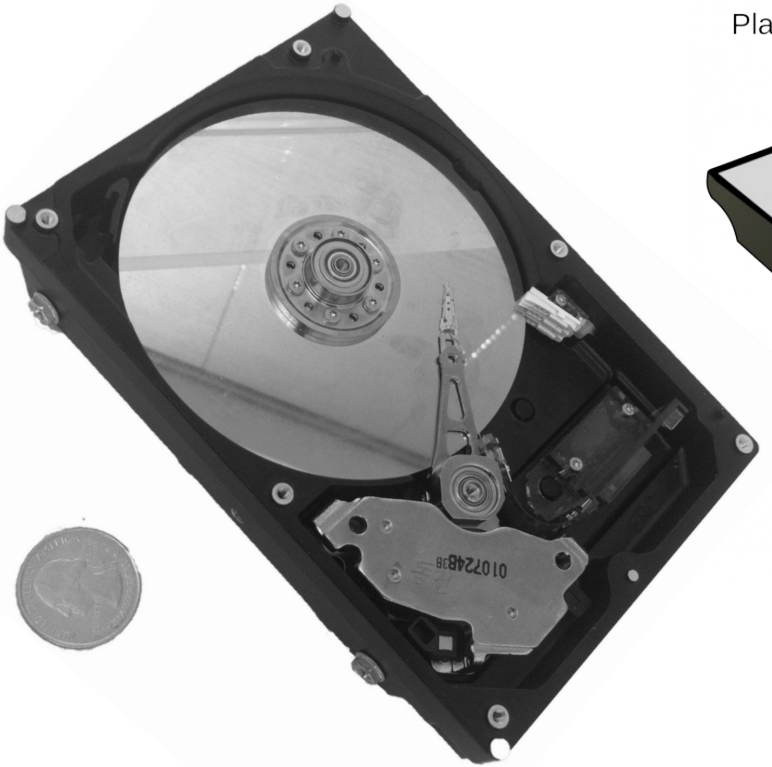
-> persistent (nonvolatile)

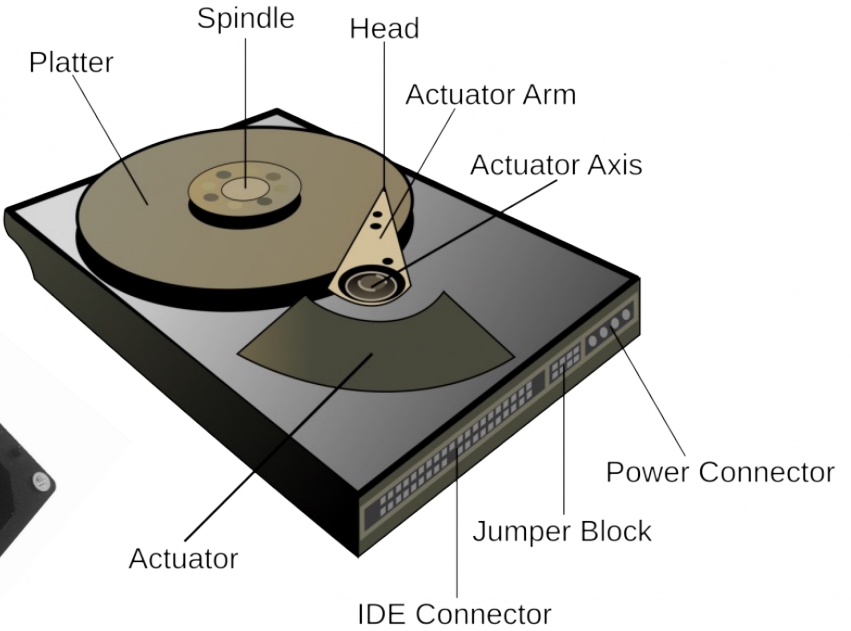
-> hard drive / spinning disk (HDD)

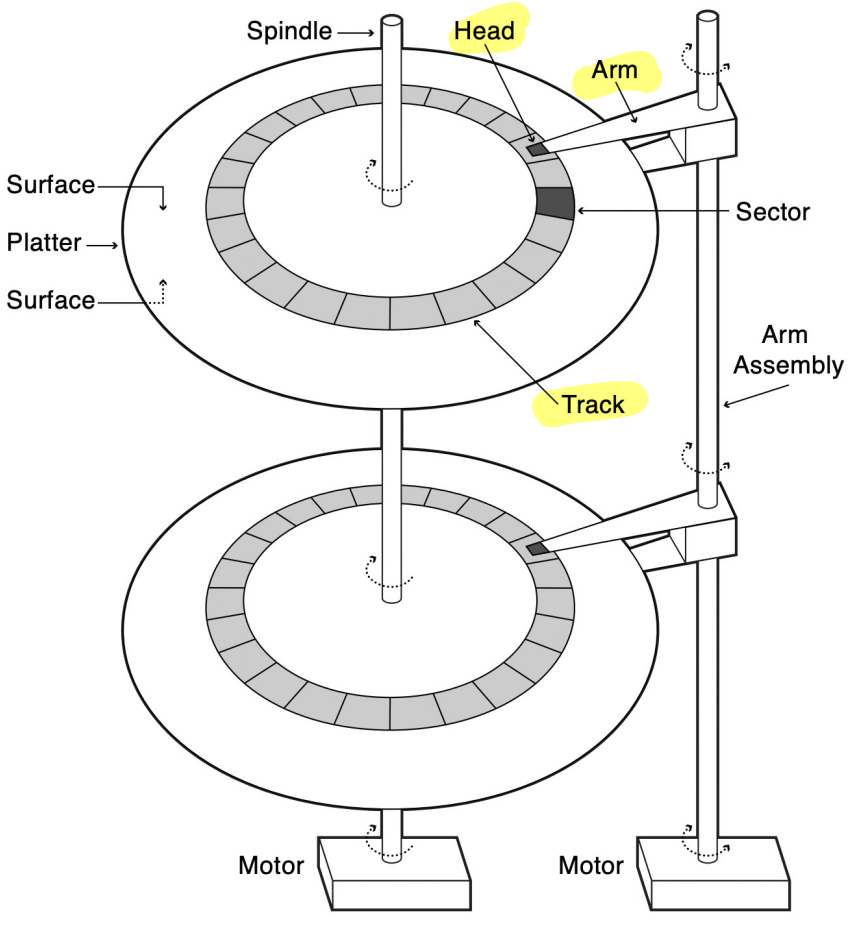
- low cost (10-20$ per TB), large capacity
- physical moving parts, slow access (10-20ms)

-> solid state drive (SSD)

- higher cost (3x disk, 60-100$ per TB), large capacity
- no physical moving parts, µs scale access latency (50-100 µs)




Spindle
Head
Platter
Actuator Arm
Actuator Axis
Power Connector
Jumper Block
Actuator
IDE Connector



Disk Anatomy

→ sector addressable (unit of read/write)

↳ 512 bytes

↳ has error correcting code.

disk read steps.

① kernel send the request (ide.c, iderw)

② disk find right platter & surface

③ move arm to track, wait for desired sector to be under the head

④ disk head reads & transfers data back to the kernel.

Disk Performance

total time for a request = seek time (arm to track) + rotation time (sector under disk head) + transfer time (data read/write)

1). seek time: 1-20ms depending on how far to seek (let's say 10ms on average)

2). Rotation time: specified as RPM, eg. 7200 RPM

(assume it takes half a rotation for the desired sector to be in the right place, 4ms)

→ need to convert to ms per rotation ≈ 8.3 ms per rotation

3). Transfer time: specified as disk bandwidth, eg. 100MB/s = 100B/μs = 5μs per sector = 0.005 ms

Sequential vs. Random access

Access 10 consecutive sectors

# of seek? 1

10ms + 4ms + 0.005x10 = 14.05 ms

Access 10 random sectors

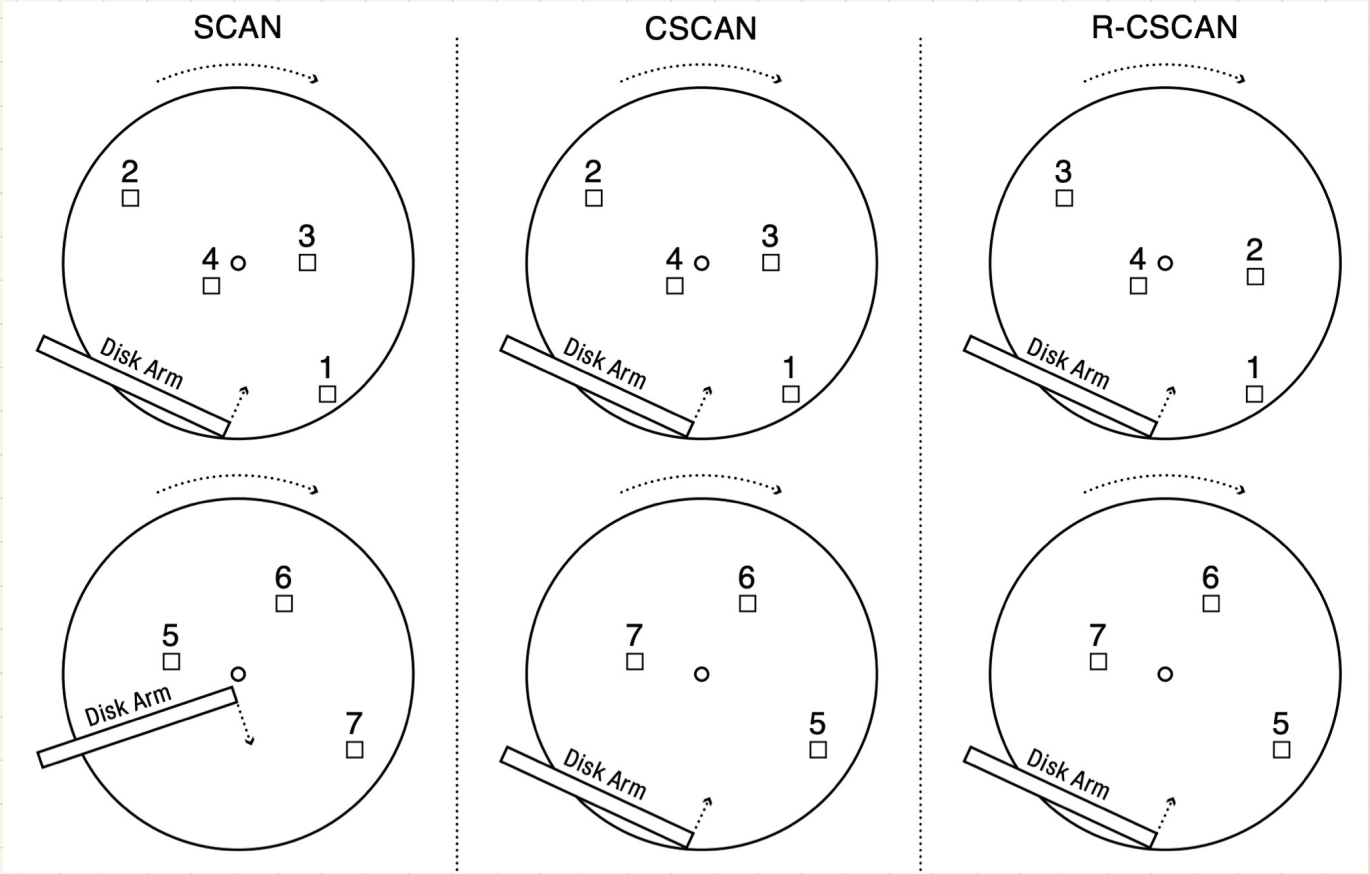
# of seeks? 10

10x10 + 4x10 + 0.005x10 = 140.05 ms

Access pattern matters!

Disk Scheduling

- reorder I/O requests for better performance
- Shortest seek time first
  - → Serve the request with the shortest seek-time next (closest track)
  - → Starvation!
- Elevator / Scan family (SCAN, CSCAN, R-CSCAN)
  - → SCAN: arm moves from inner most to outer most, the outer most to inner most, serving request along the way.
  - → CSCAN: arm moves from inner most to outer most, once it reaches the end, moves back to inner most & start again
  - → R-CSCAN: takes rotation delay into account


SCAN
CSCAN
R-CSCAN
2
3
4
1
Disk Arm
6
5
7
Disk Arm