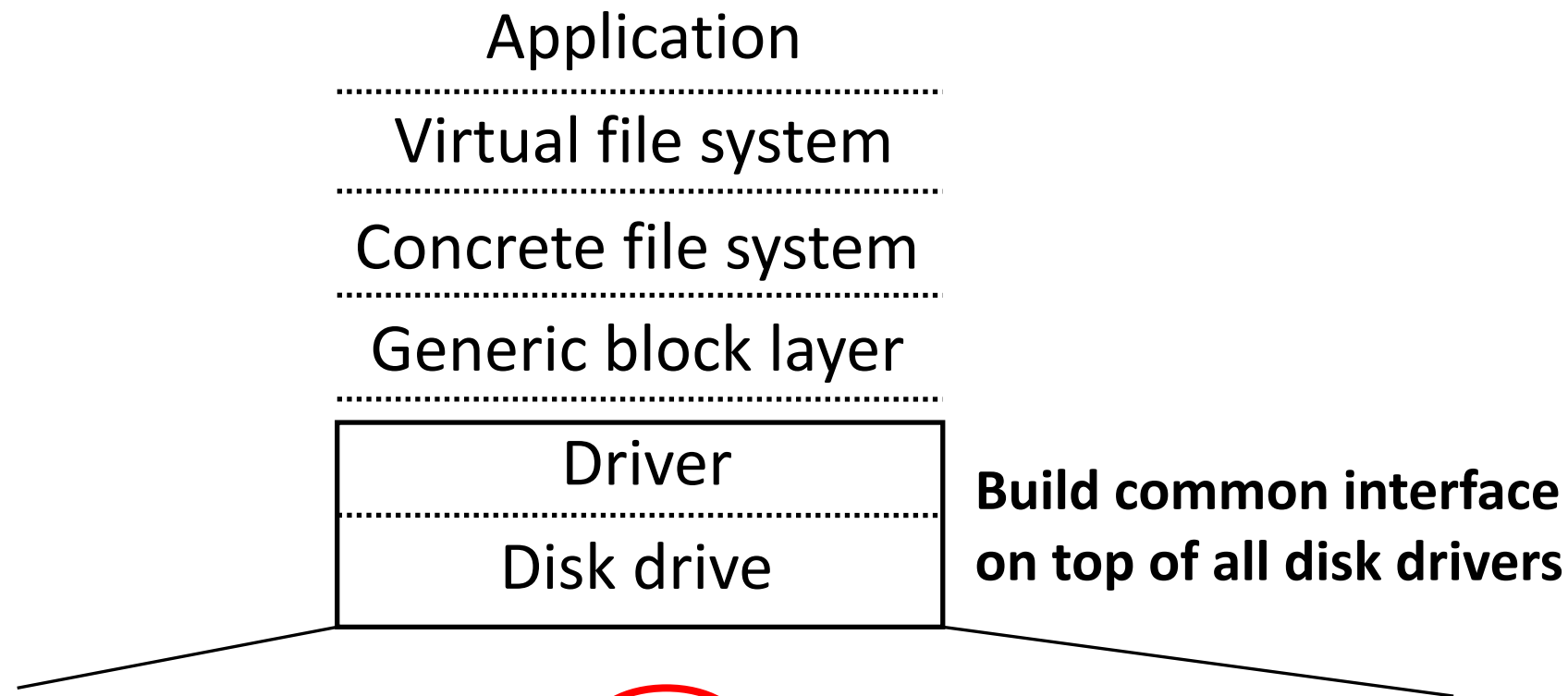
Hard Disk Drives

Nima Honarmand
(Based on slides by Prof. Andrea Arpaci-Dusseau)

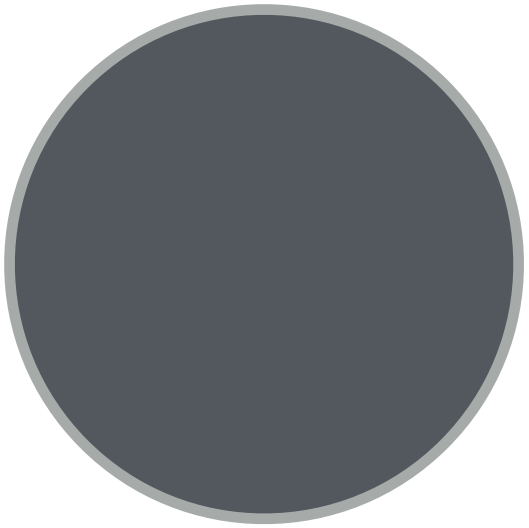
Storage Stack in the OS



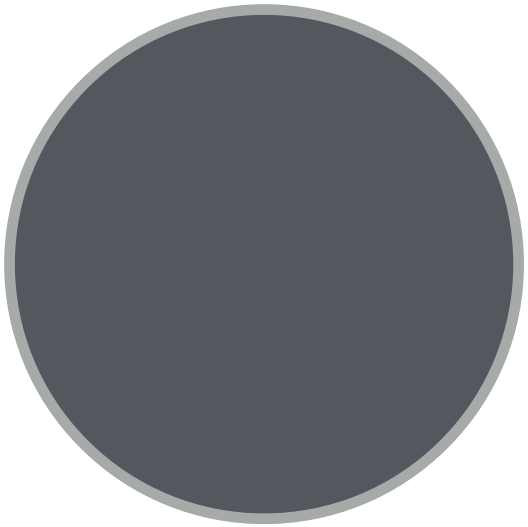
Basic Interface

- Disk has a ***sector***-addressable address space
  - Appears as an array of sectors to the OS
- Sectors are typically 512 bytes or 4096 bytes
- Main operations: reads + writes to sectors
- Mechanical (slow) nature makes management "interesting"

Platter

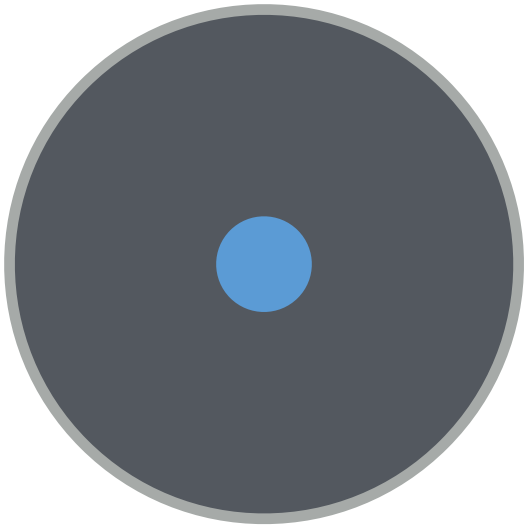


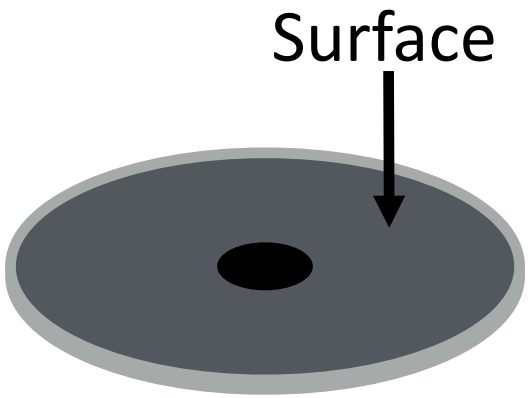
Disk Internals



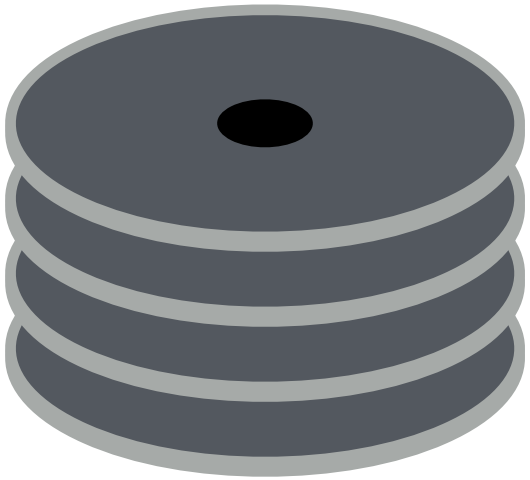
Platter is covered with a magnetic film.

Spindle

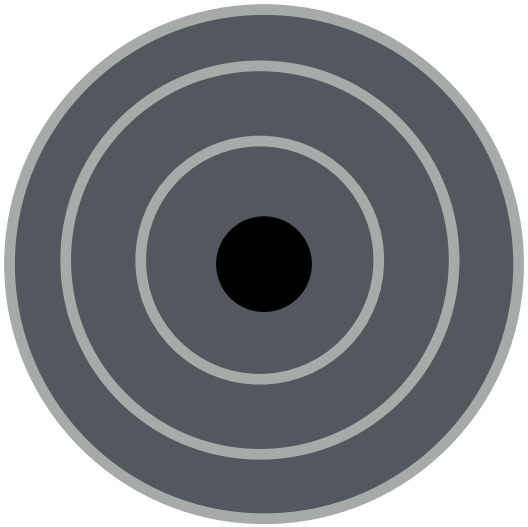




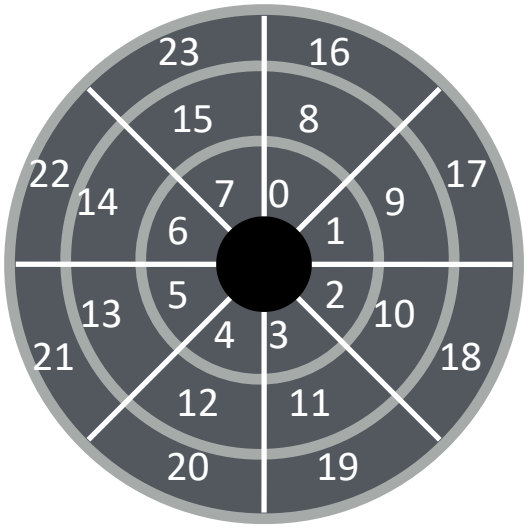
Surface



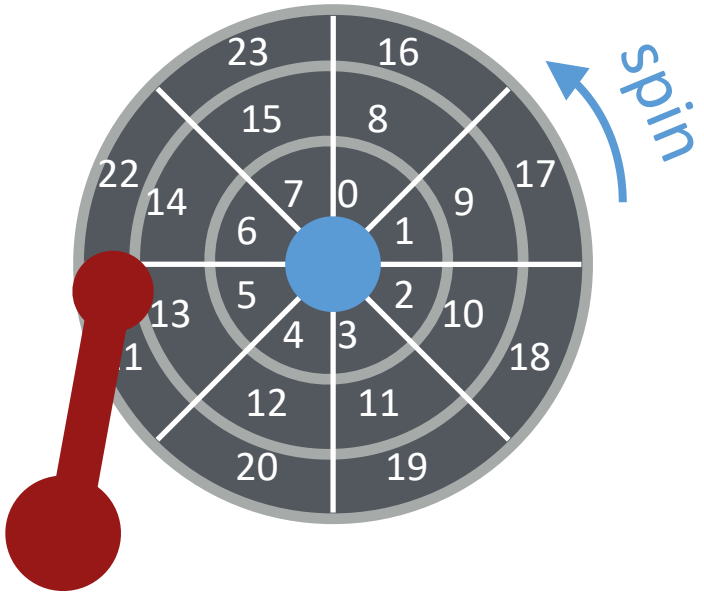
Many platters may be bound to the spindle.



Each surface is divided into rings called a track.
A stack of tracks (across platters) is called a cylinder.



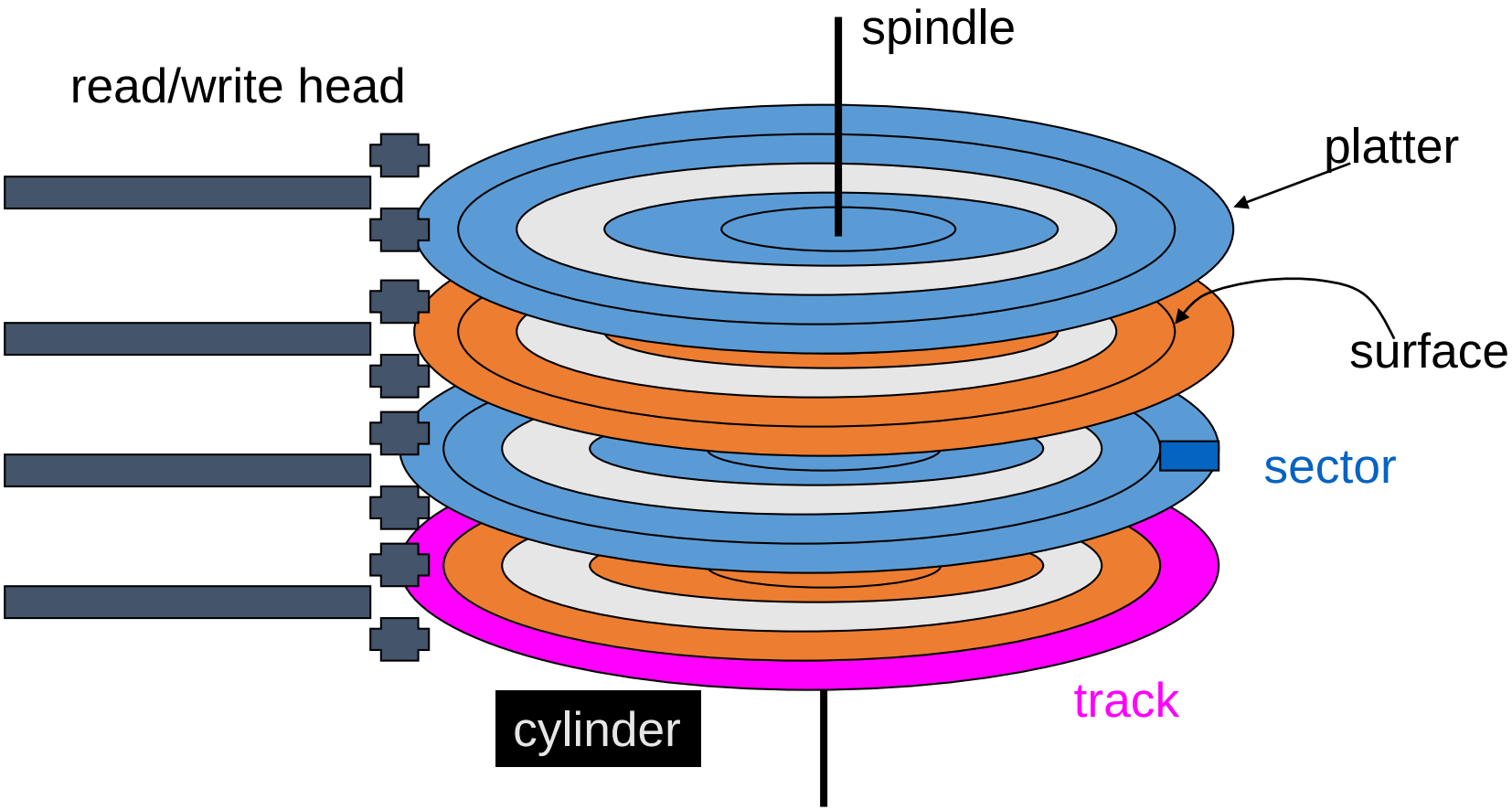
The tracks are divided into numbered sectors.



Spindle/platters rapidly spin.

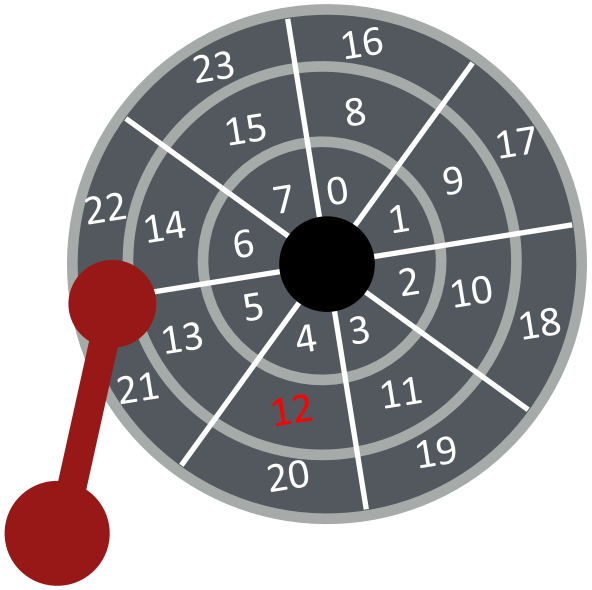
Heads on a moving arm can read from each surface.

Disk Terminology



Let's Read Sector 12!

Step 1: Seek to right track

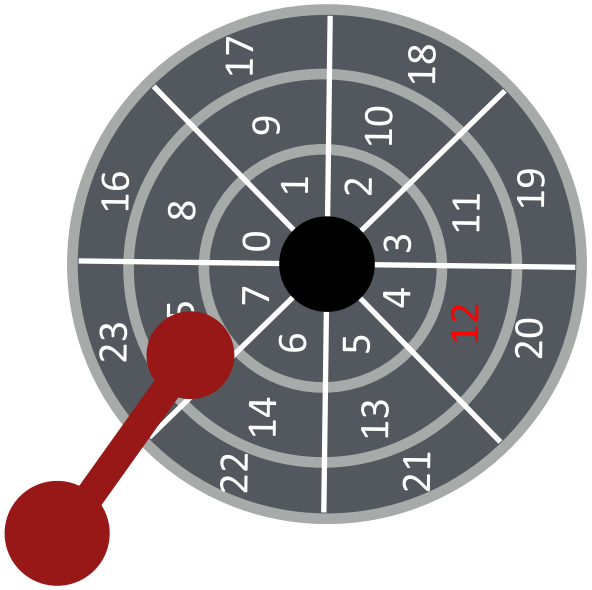


Step 1: Seek to right track

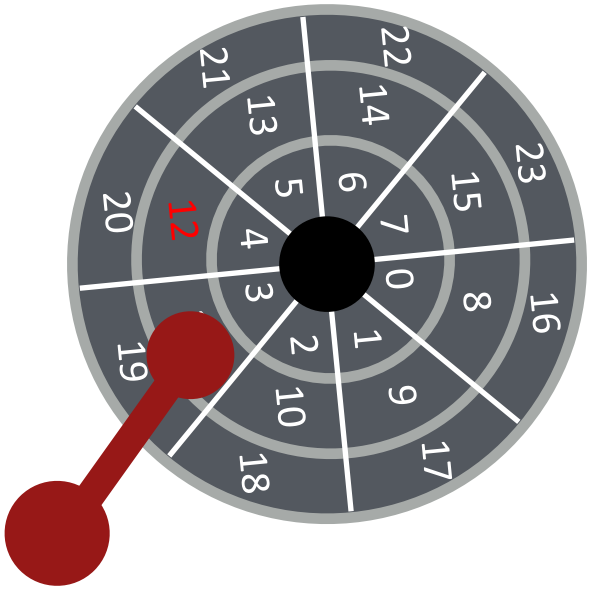
The disk keeps rotating at a constant speed, even when seeking.

Step 2: Wait for rotation

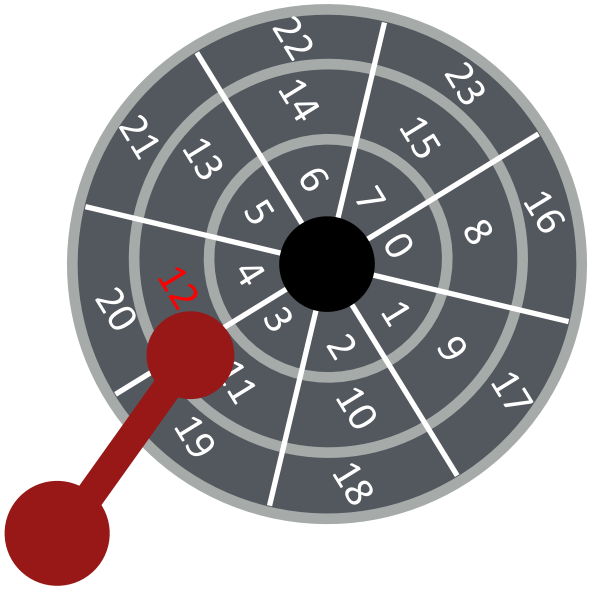
Step 2: Wait for rotation



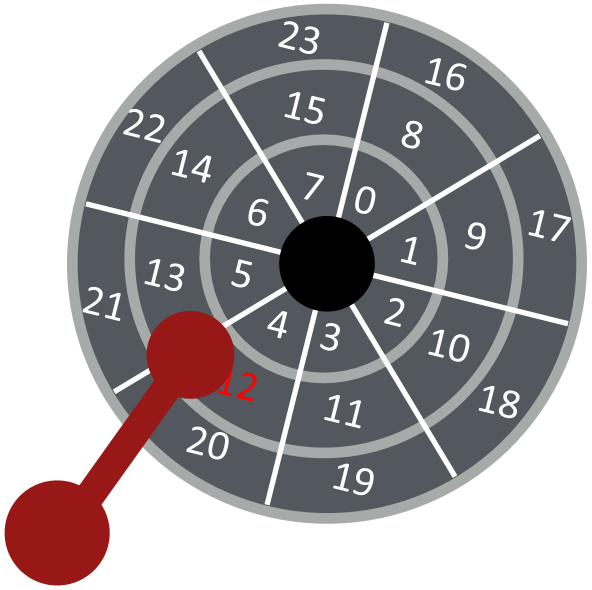
Step 2: Wait for rotation



Step 3: Transfer data



Yay!



HDD Video Demo

- https://www.youtube.com/watch?v=9eMWG3fwiEU&feature=youtu.be&t=30s
- https://www.youtube.com/watch?v=L0nbo1VOF4M

Time to Read/Write a Sector

- Three components
  1) Seek time
  2) Rotation time
  3) Transfer time

- Time = seek + rotation + transfer

1) Seek Time

- Seek time: function of cylinder distance
  - Not purely linear cost
- Must accelerate, coast, decelerate, settle
- Settling alone can take 0.5–2 ms
- Entire seeks often takes several milliseconds
  - 4–10 ms
- Average seek distance?
  - 1/3 max seek distance. Why?

2) Rotation Time

- Depends on disk's rotational speed: Rotations Per Minute (RPM)
  - 7200 RPM is common, 15000 RPM is high end.
- With 7200 RPM, how long to rotate around?
  - 1 / 7200 RPM
    = 1 minute / 7200 rotations
    = 1 second / 120 rotations
    = 8.3 ms / rotation
- Average rotation?
  - 8.3 ms / 2 = 4.15 ms

3) Transfer Time

- Pretty fast — depends on RPM and sector density.
- 100+ MB/s is typical for maximum transfer rate
- How long to transfer 512-bytes?
  - 512 bytes / (100 MBps) = 5 us

Workload Performance

- So…
  - seeks are slow
  - rotations are slow
  - transfers are fast
- What kind of workload is faster for disks?
  - ***Sequential***: access sectors in order (transfer dominated)
  - ***Random***: access sectors arbitrarily (seek+rotation dominated)

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

- Sequential workload: what is throughput for each?
  - Cheetah: 125 MB/s
  - Barracuda: 105 MB/s

Disk Spec

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

- Random workload: what is throughput for each?
- Assume size of each random read is 16KB

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

Time = seek + rotate + transfer

Seek = 4 ms
Full rotation = 60 / (15,000) = 4 ms
Half rotation = 2 ms
Transfer = 16 KB / 125 MBps = 125 us

Throughput = 16 KB / (6.125 ms) = **2.5 MBps**

	Cheetah	Barracuda
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s

Time = seek + rotate + transfer

Seek = 9 ms
Full rotation = 60 / (7,200) = 8.3 ms
Half rotation = 4.1 ms
Transfer = 16 KB / 100 MBps = 160 us

Throughput = 16 KB / (13.260 ms) = **1.2 MBps**

	Cheetah	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Avg Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	32 MB

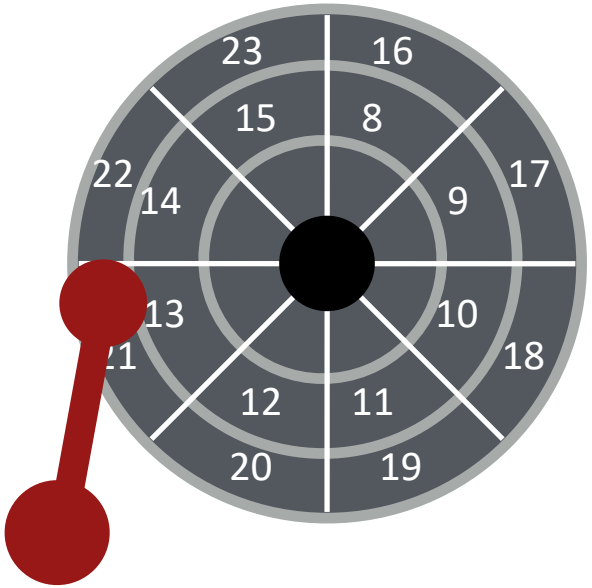
	Cheetah	Barracuda
Sequential	125 MB/s	105 MB/s
Random	2.5 MB/s	1.2 MB/s

This shows the importance of proper ***disk scheduling*** to achieve good disk performance.

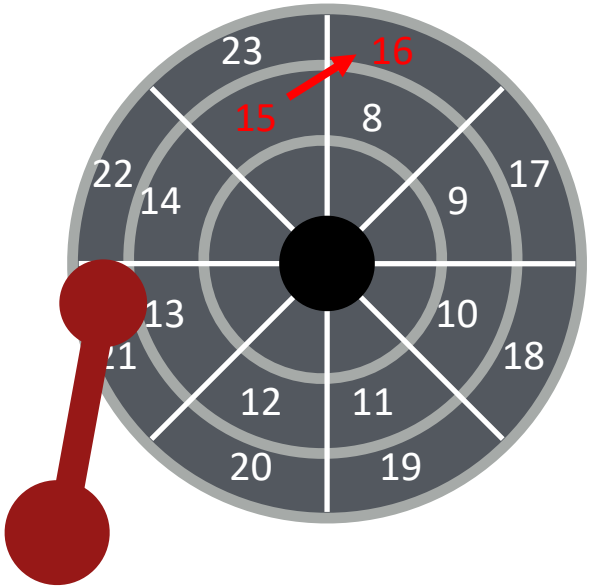
Other Improvements

- **Track Skew**
- Zones
- Drive Cache

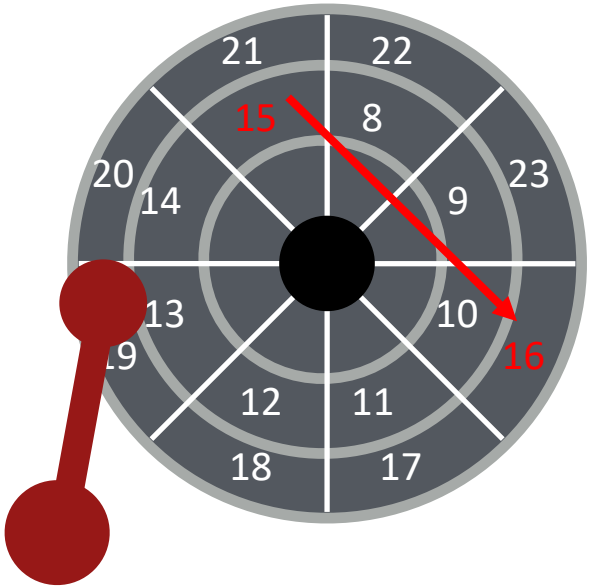
Imagine sequential reading. How should sectors numbers be laid out on disk?



When reading 16 after 15, the head won't settle quick enough, so we need to do a rotation.

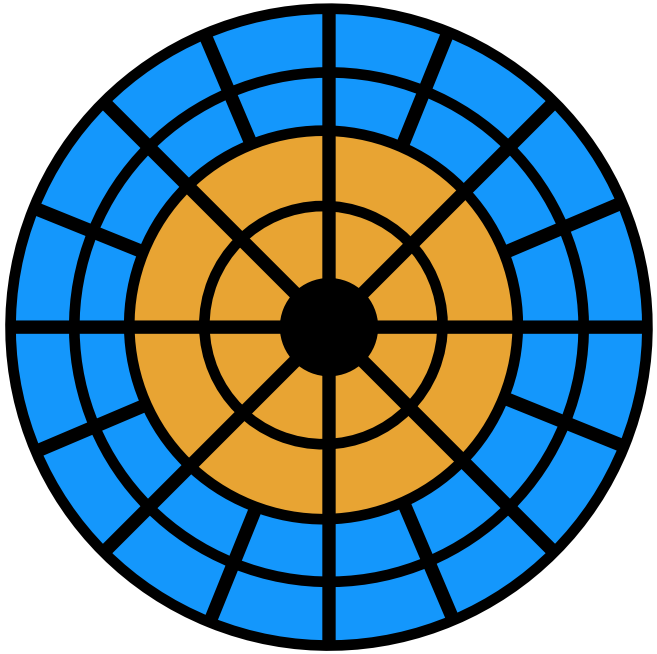


Enough time to settle now!



Other Improvements

- Track Skew
- **Zones**
- Drive Cache



ZBR (Zoned bit recording): More sectors on outer zones

Within each zone, all tracks have the same number of sectors per track.

Other Improvements

- Track Skew
- Zones
- **Drive Cache**

Drive Cache

- Drives may cache both reads and writes
  - OS caches data too
- Disks contain internal memory (2−16MB) used as cache
  - A.k.a. “Track Buffer”
  - Provides multiple benefits

1) Read-ahead
   - Read contents of entire track into memory during rotational delay
   - Can send them to OS if it asks for them later

Drive Cache (2)

2) Write caching
   - Keep write data in the drive cache and claim completion to OS
   - “faster” response time
   - But data could be lost on power failure

3) Tagged command queueing
   - Have multiple outstanding requests to the disk
   - Disk can reorder (schedule) requests for better performance
   - OS tags each request with an ID; disk uses tag to report completion

Disk Scheduling

Disk Scheduling

- We saw importance of proper request ordering in our throughput example
  - 125 MBps for sequential vs. 2.5 MBps for random workload
- Crux: Given a stream of requests, in what order should they be served?
- Much different than CPU scheduling
  - Performance dominated by seek+rotation
  - Position of disk head relative to request position matters more than job length (i.e., request size)

First-Come-First-Serve (FCFS) Scheduler

- Assume seek+rotate = 10 ms for a random request
- How long (roughly) does below workload (1) below take?
  - Requests are given in sector numbers
  - 60 msec
- How about Workload (2)?
  - 20 msec

(1) 300001, 700001, 300002, 700002, 300003, 700003

(2) 300001, 300002, 300003, 700001, 700002, 700003

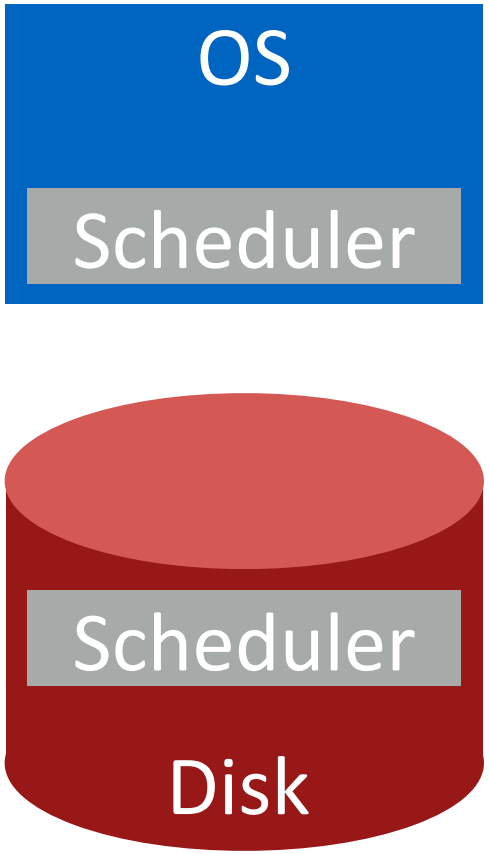
- Small change in request ordering yielded 3x improvement in disk bandwidth utilization!

How to Maximize BW?

- Given a set of requests, which one would you choose next to maximize BW?
- **Strategy**: always choose the request with least positioning time — **Shortest Positioning Time First** (**SPTF**)
- Where best to implement?
  1) Disk Controller
  2) OS
- In general, OS doesn't know the disk geometry. It can approximate **SPTF** by **Nearest Block First** (**NBF**)
- **Disadvantage**: easy for far-away requests to **starve**

Detour: Where to Do Scheduling?

- OS:
  - Positive: can know about all the pending requests; a more global view
  - Negative: does not know disk geometry
- Disk:
  - Positive : knows the geometry
  - Negative: can only hold a few requests to schedule among
- Reality: **both** — OS picks next few "good" requests to send to disk; disk then schedules among them



Tackling Starvation Problem

- ***Elevator*** algorithm (*a.k.a*. ***SCAN***)
  - ***Sweep*** back and forth, from one end of disk other, serving requests as pass that track
- Variations to improve fairness:
  - **F-SCAN**: freeze the queue of requests when doing a sweep in the current direction
    - To avoid starving requests waiting in the other direction
  - **C-SCAN**: only sweep in one direction
    - To be fair to the outer tracks; in original SCAN, middle tracks are passed over twice as often as outer tracks
- Simple old algorithms — not used directly today
  - But the idea is useful and can be part of more complex solutions

Anticipatory Scheduling (1)

- Assume 2 processes each calling `read()` on different files
  Assume OS uses something similar to C-SCAN

```
void reader(int fd) {
    char buf[1024];
    int rv;
    while((rv = read(fd, buf)) != 0) {
        assert(rv);
        // takes short time, e.g., 1ms
        process(buf, rv);
    }
}
```

- Should OS serve P2's read after finishing P1's read?

Anticipatory Scheduling (2)

- Let's say disk is idle and OS receives a read request. Should we send it immediately to disk?
- **Work-preserving schedulers**: yes, let's do work if there is work to be done
- **Anticipatory schedulers**: no, let's wait for some time in case a "better" request arrives soon
- Ideally, OS can observe each process's behavior over time to learn its access pattern
  - And use that to decide whether to wait or not before switching to requests from another process

Completely Fair Queuing (CFQ)

- Linux's current default scheduler

- Queue of requests for each process
- Weighted round-robin between queues, with IO slice time proportional to priority
- Yield slice only if idle for a given time
  - Emulating some sort of anticipatory algorithm
- A back-seek penalty to optimize order within queue
  - Emulating some aspects of elevator algorithms

Hard Disk Summary

- Storage devices provide common sector-based interface
- On a hard disk: Never do random I/O unless you must!
  - Quicksort is a terrible algorithm on disk
- It pays off to spend CPU time to do complex scheduling on slow hard disk devices
- These scheduling algorithms were motivated by hard disk properties; SSDs have different properties and thus require different algorithms ☺
  - Read the OSTEP chapter to learn about these other devices