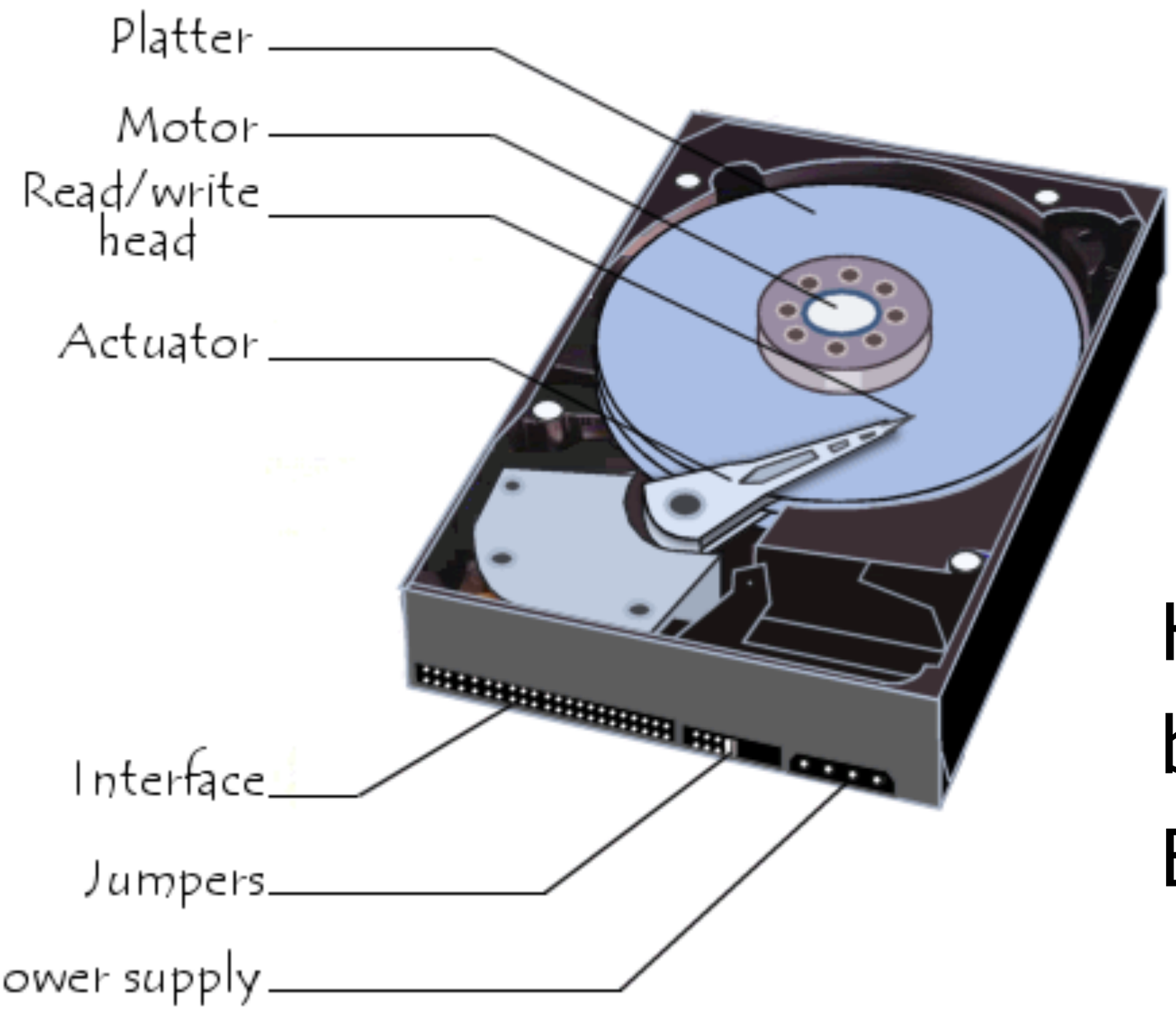
CS 423
Operating System Design

https://cs423-uiuc.github.io

Tianyin Xu
tyxu@illinois.edu

* Thanks Adam Bates for the slides.

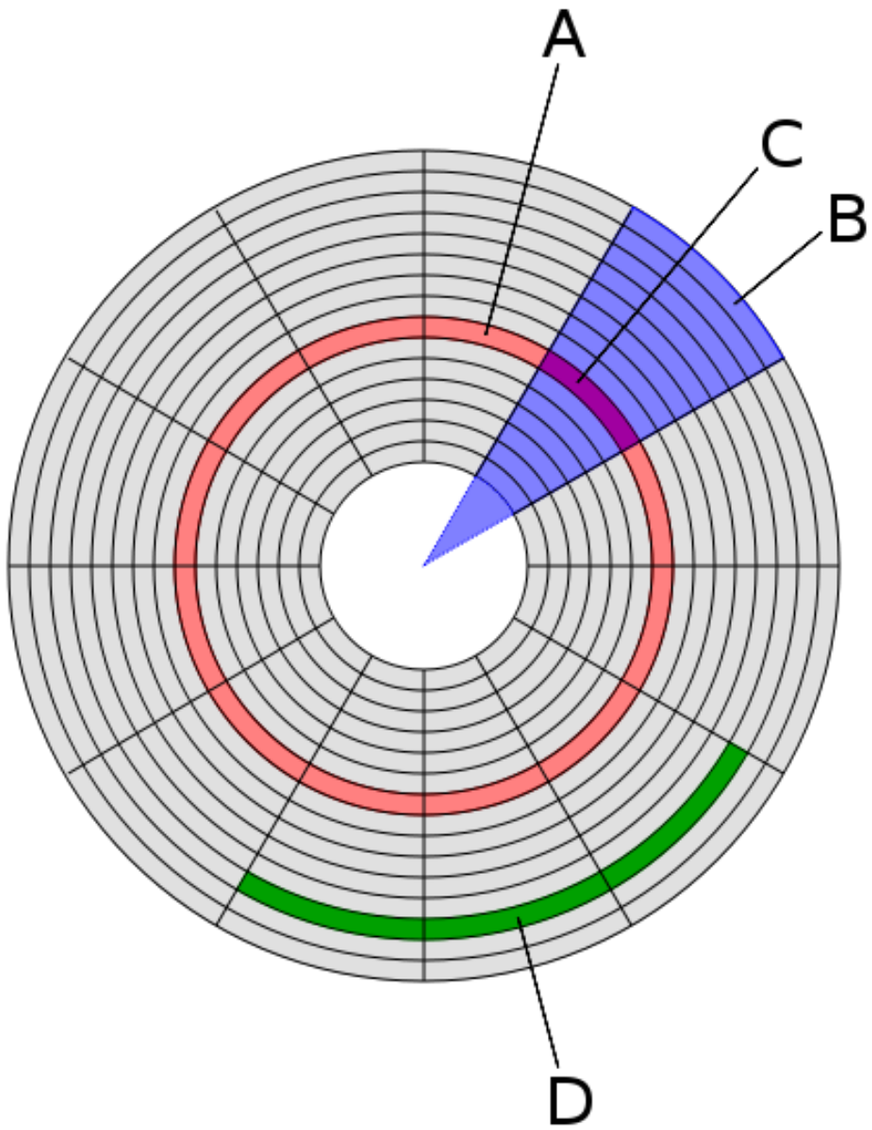
Disk



HDDs not SOTA
by any means
But still relevant!

Hard Disk Internals

A: Track.
B: Sector.
C: Sector of Track.
D: File

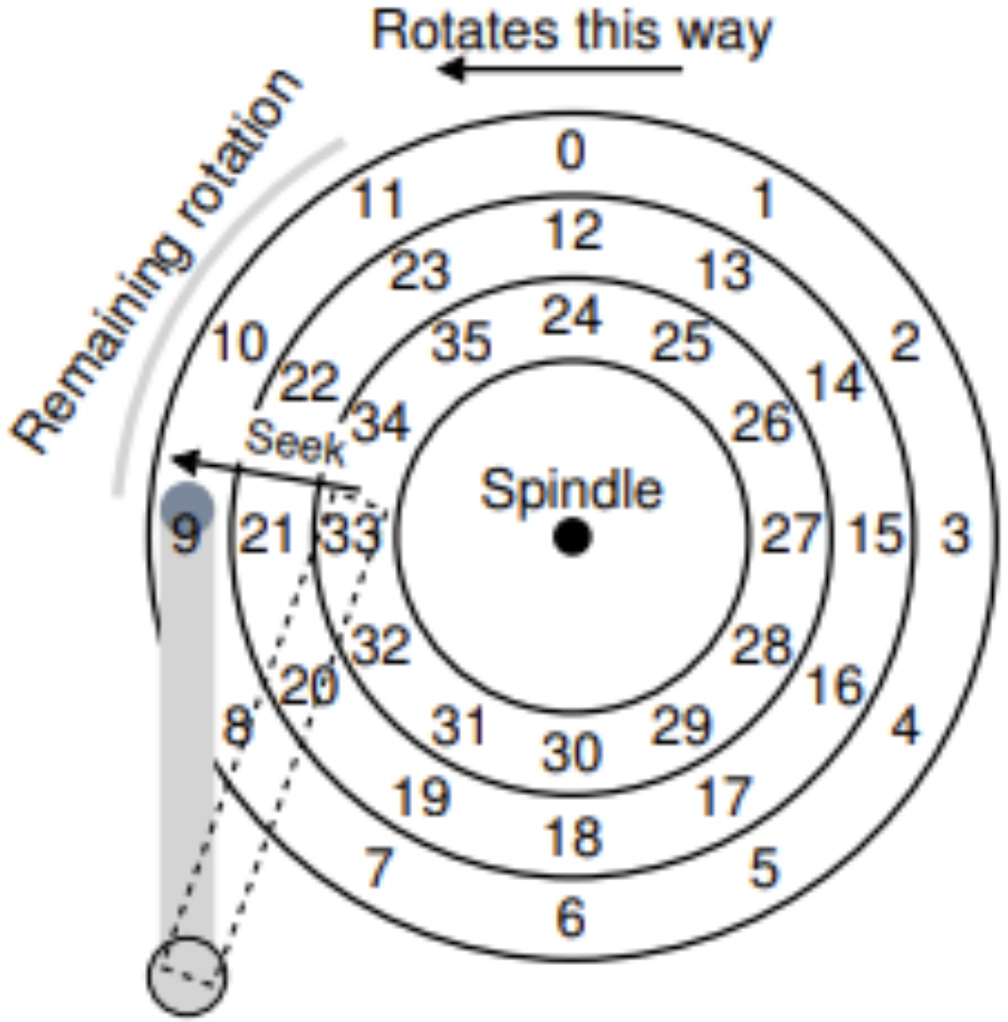


Disk Internals

Seek: move head to the target track

Rotate: wait for target sector to be under head

Transfer: access data



HDD in Action

Disk Access Time Example

- Disk Parameters
  - Advertised average seek time is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
- Assume idle disk (i.e., no queuing delay)

```
Disk Access Time=seek time +
                 rotational delay +
                  transfer time
```

Disk Access Time Example

- Disk Parameters
  - Advertised average seek time is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
- Assume idle disk (i.e., no queuing delay)
  - Q1: What is the total time to read 500 random sectors?
  - Q2: What is the total time to read 500 sequential sectors (assume on same track)?

Disk Access Time Example

- What is the total time to read 500 random sectors?

Disk Access Time Example

- What is the total time to read 500 sequential sectors (assume on same track)?

Disk Access Time Example

See the difference between random and sequential IO speeds on hard drives?

Always design for sequential IO on HDDs!

Random IO performance (somewhat) better with SSDs. High-level reason?

Disk Access Time Example

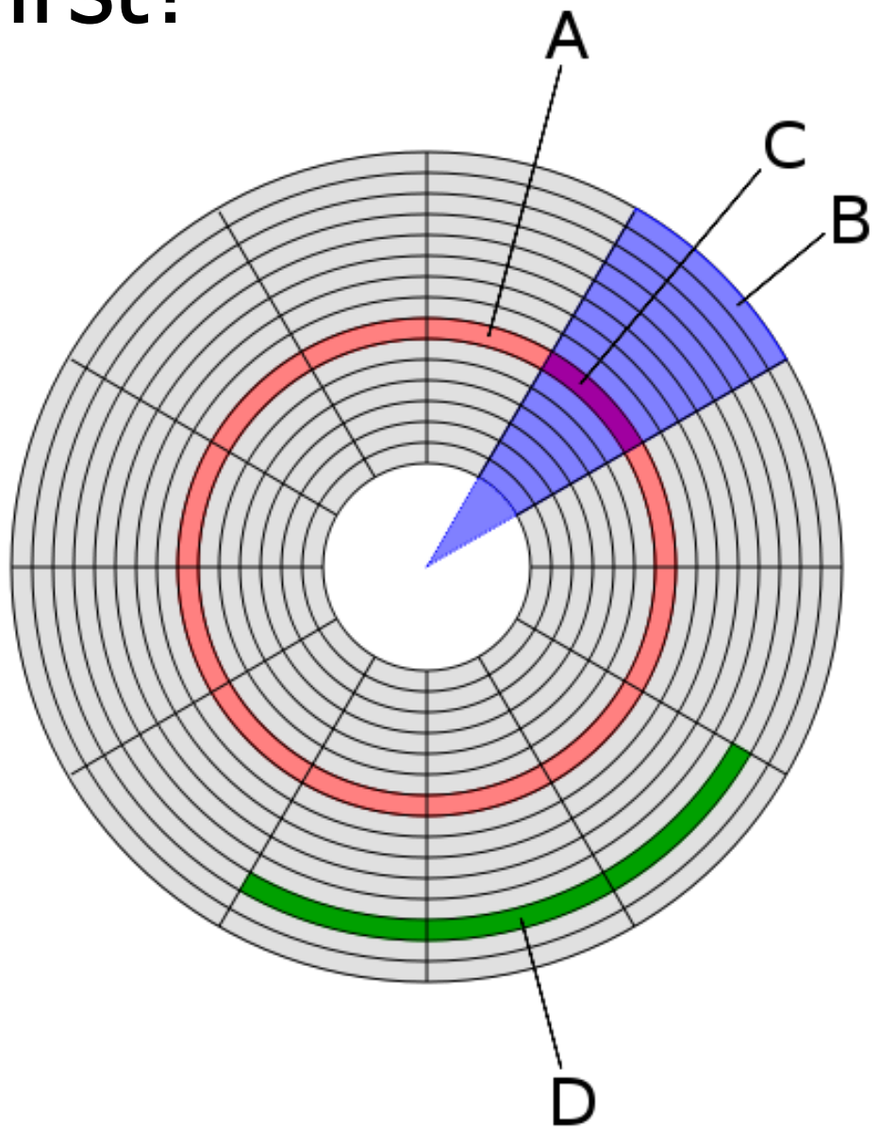
Which one do you think will be faster on HDD?

copying many small files

vs. copy one large file?

Disk Scheduling

- Which disk request is serviced first?
  - FCFS
  - Shortest seek time first
  - SCAN (Elevator)
  - C-SCAN (Circular SCAN)

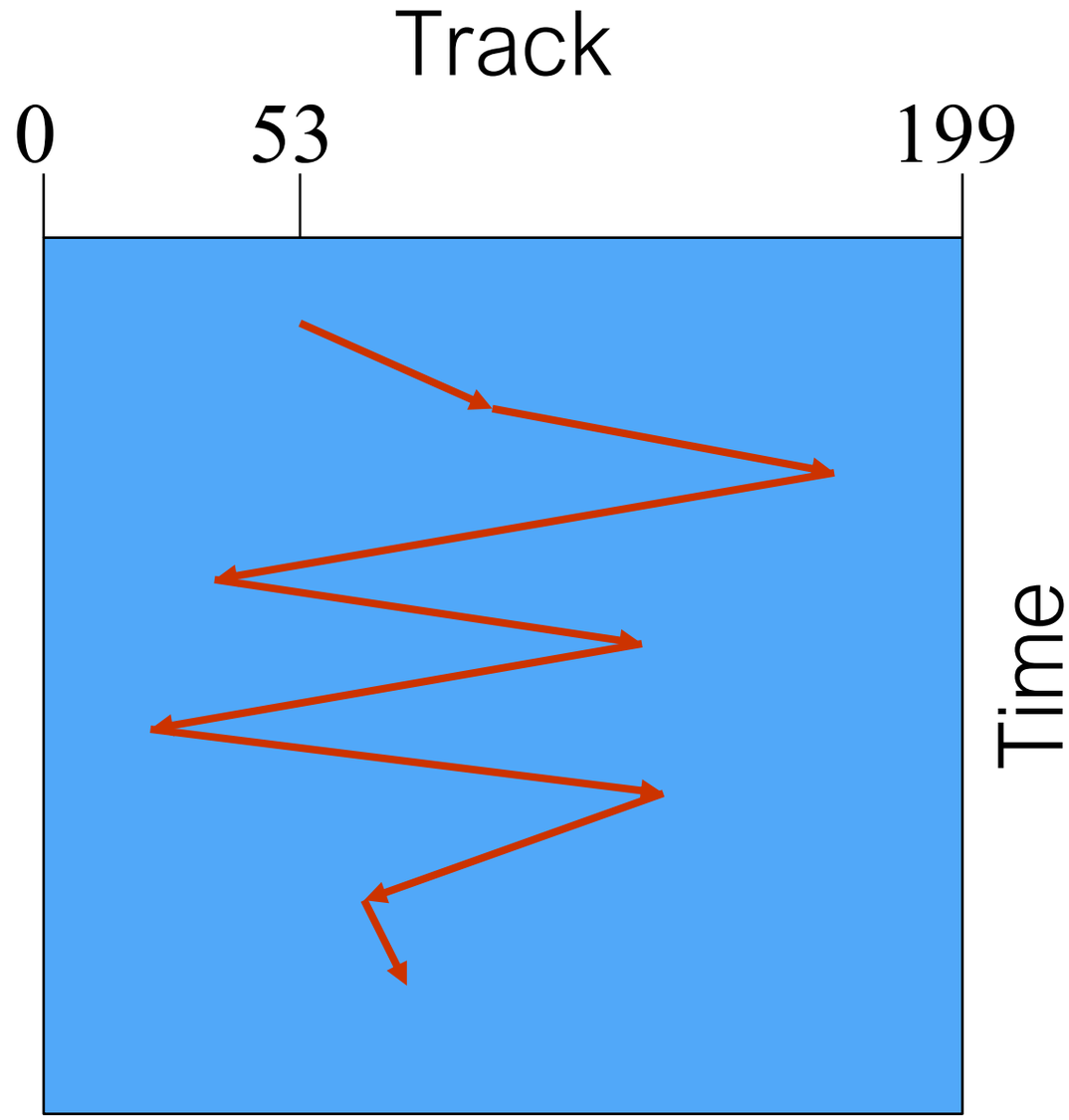


A: Track.
B: Sector.
C: Sector of Track.
D: File

Disk Scheduling Decision — Given a series of access requests, on which track should the disk arm be placed next to maximize fairness, throughput, etc?

FIFO (FCFS) Order

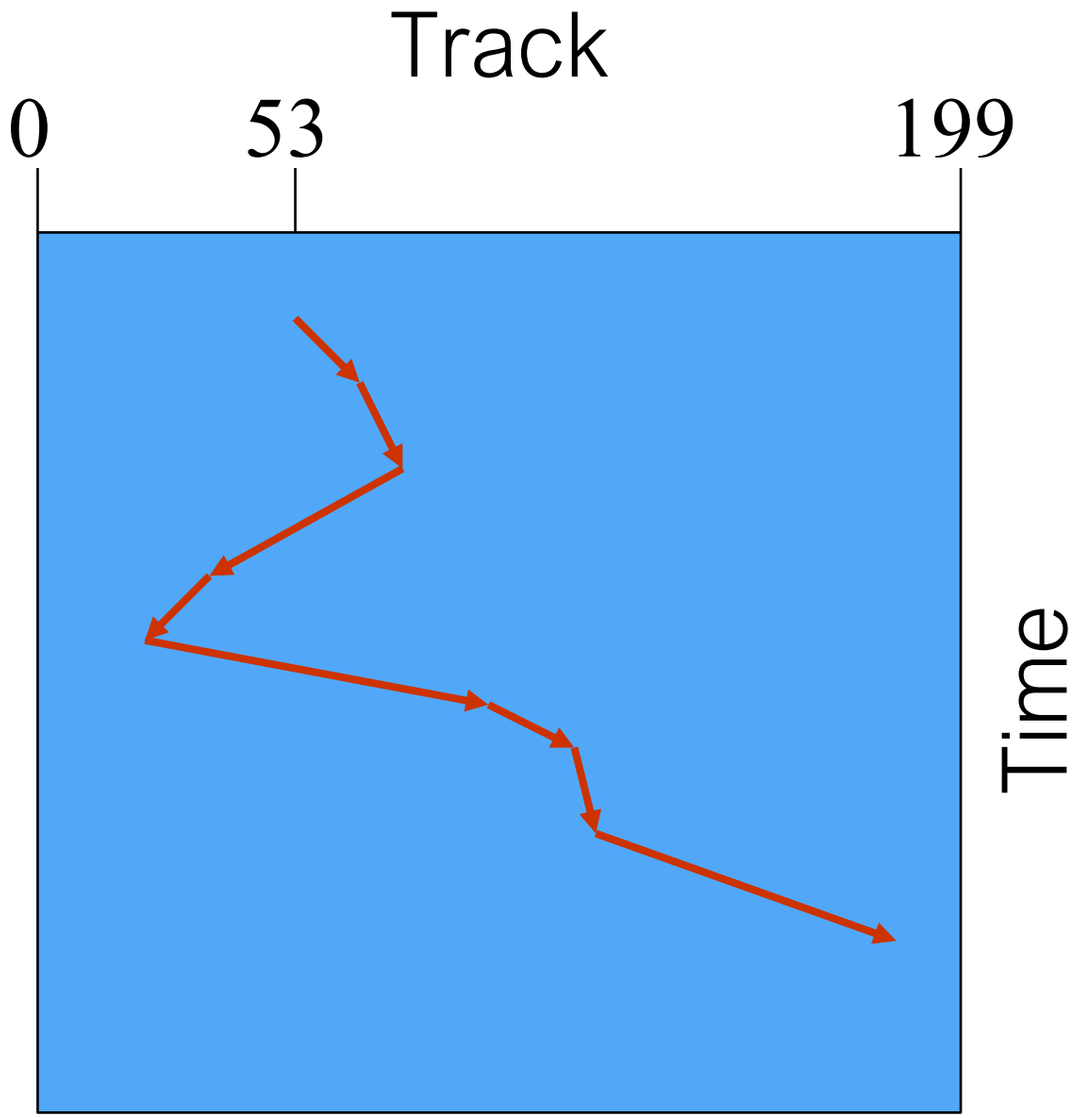
- Method
  - First come first serve
- Pros?
  - Fairness among requests
  - In the order applications expect
- Cons?
  - Arrival may be on random spots on the disk (long seeks)
  - When is it particularly bad?



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

SSTF (Shortest Seek Time First)

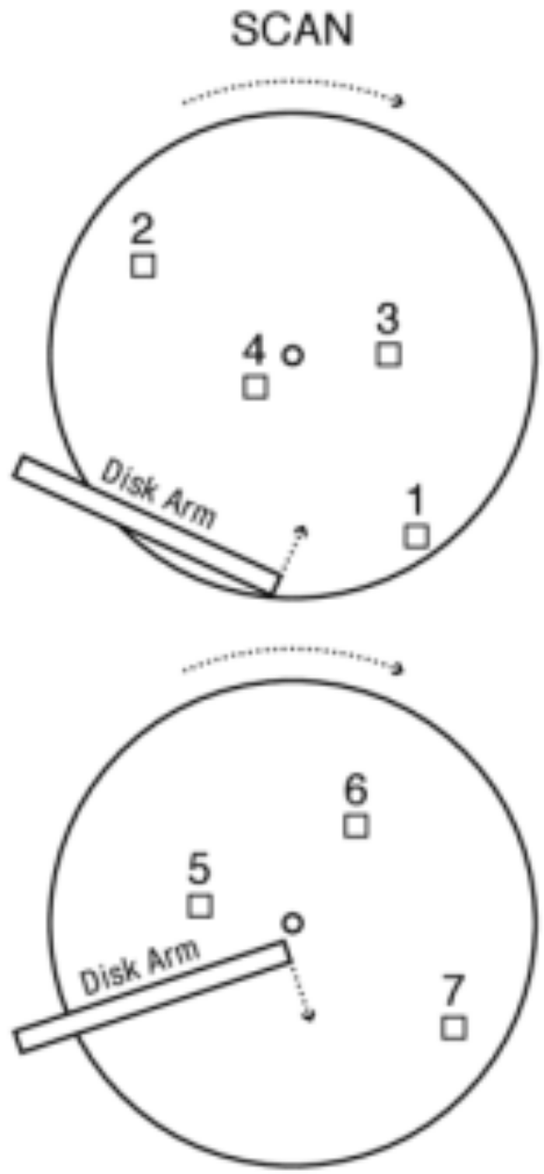
- Method
  - Pick the one closest on disk (greedy approach)
- Pros?
  - Tries to minimize seek time
- Cons?
  - Starvation
- Questions
  - Is SSTF optimal?
  - Is this fair to all disk accesses?
  - Can we avoid starvation?



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)

SCAN (Elevator)

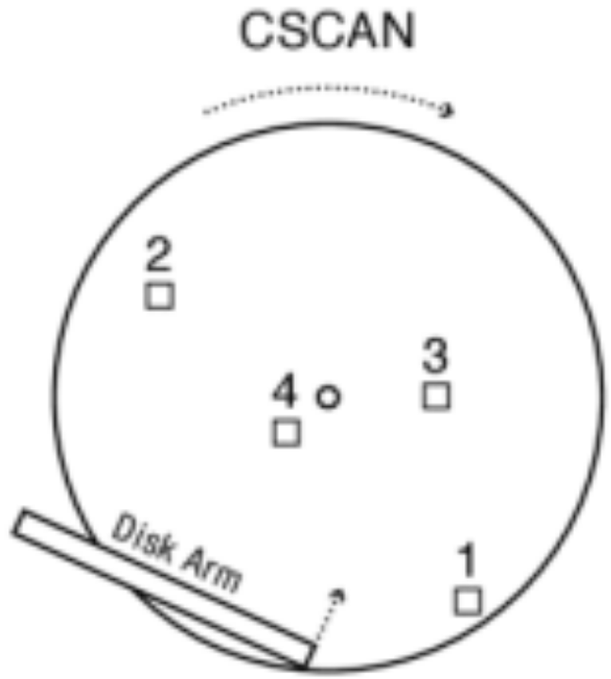
- Move outer to inner – service all requests along the way
- Move inner to outer – service all along the way



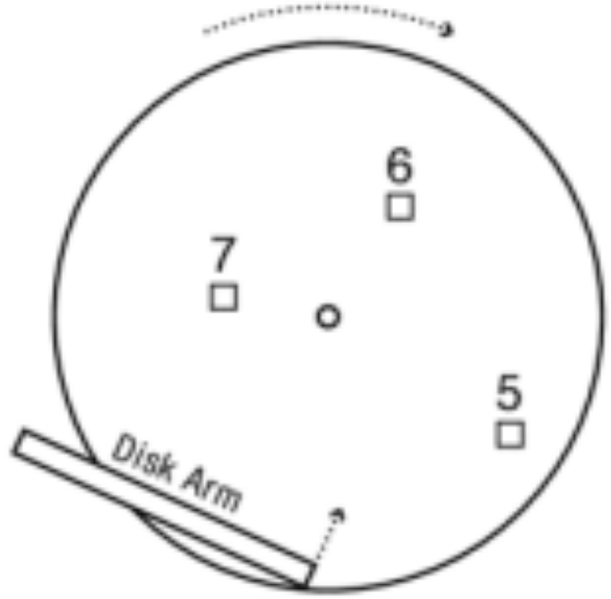
- Adv compared to SSTF:
  - Bounded time for each request

C-SCAN (Circular SCAN)

Like SCAN
But, wrap around (i.e., only one direction)



- Adv over SCAN
  - By seeking to opposite side, moves head to where pending requests are likely to be denser
  - More fair
- Cons
  - Do nothing on the return (i.e., higher overhead)



Scheduling Algorithms

Algorithm Name	Description
FCFS	First-come first-served
SSTF	Shortest seek time first; process the request that reduces next seek time
SCAN (aka Elevator)	Move head from end to end (has a current direction)
C-SCAN	Only service requests in one direction (circular SCAN)
LOOK	Similar to SCAN, but do not go all the way to the end of the disk.
C-LOOK	Circular LOOK. Similar to C-SCAN, but do not go all the way to the end of the disk.

Who does Scheduling?

The OS?

The disk itself?

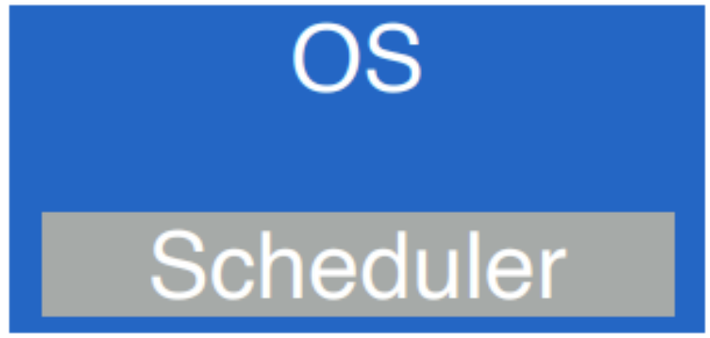
Both?

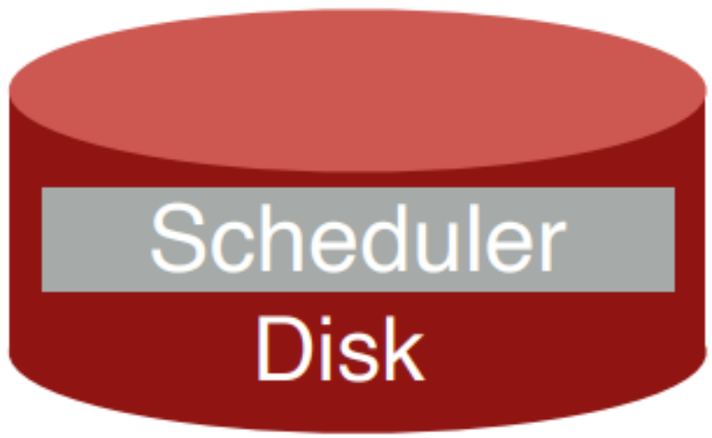
Who does Scheduling?

The OS?

The disk itself?

Both?





Linux I/O Schedulers

- What disk (I/O) schedulers are available in Linux?

```
$ cat /sys/block/sda/queue/scheduler
noop deadline [cfq]
```

^ scheduler enabled on our VMs

- As of Linux 2.6.10, it is possible to change the IO scheduler for a given block device on the fly!
- How to enable a specific scheduler?

```
$ echo SCHEDNAME > /sys/block/DEV/queue/scheduler
```

  - SCHEDNAME = Desired I/O scheduler
  - DEV = device name (e.g., sda)

Linux NOOP Scheduler

- Insert all incoming I/O requests into a simple FIFO
- Merges duplicate requests (results can be cached)
- When would this be useful?

Linux NOOP Scheduler

- Insert all incoming I/O requests into a simple FIFO
- Merges duplicate requests (results can be cached)
- When would this be useful?
  - Solid State Drives! Avoids scheduling overhead
  - Scheduling is handled at a lower layer of the I/O stack (e.g., Disk firmware, RAID Controller, Network-Attached)
  - Host doesn't actually know details of sector positions

Linux Deadline Scheduler

- Imposes a deadline on all I/O operations to prevent starvation of requests
- Maintains 4 queues:
  - 2 Sorted Queues (R, W), order by Sector
  - 2 Deadline Queues (R, W), order by Exp Time
- Scheduling Decision:
  - Check if 1st request in deadline queue has expired.
  - Otherwise, serve request(s) from Sorted Queue.
  - Prioritizes reads (DL=500ms) over writes (DL=5s) .Why?

Linux CFQ Scheduler

- CFQ = Completely Fair Queueing!
- Maintain per-process queues.
- Allocate time slices for each queue to access the disk
- I/O Priority dictates time slice, # requests per queue
- Asynchronous requests handled separately — batched together in priority queues
- CFQ is often the default scheduler

What Happens?

Assume 2 processes each calling read() with 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);
}
}
```

What Happens?

Assume 2 processes each calling read() with 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);

}

}
```

P1: read 100, 101

P2: read 900, 901

After 1 ms

P1: read 102, 103

P2: read 902, 903

Work Conservation

Work conserving schedulers always try to do work if there's work to be done

Sometimes, it's **better to wait** instead if system anticipates another request will arrive

Possible improvements from I/O merging

Linux Anticipatory Scheduler

- Deceptive Idleness: A process appears to be finished reading from disk, but is actually processing data. Another (nearby) request is coming soon!
- Bad for synchronous read workloads because seek time is increased.
- Anticipatory Scheduling: Idle for a few milliseconds after a read operation in *anticipation* of another close-by read request.
- Deprecated — CFQ can approximate.

Summary

Disks: specific geometry with platters, spindle, tracks, sector, head, etc

DAT = seek time + rotation delay + transfer time

Sequential bandwidth is much higher than random bandwidth

Scheduling approaches: FCFS, SSTF, SCAN, C-SCAN

Schedulers are at multiple layers of the stack

Need to think together (e.g., Linux NOOP)

What is above?

- Above the disk and IO scheduler? **The file system**!

Abstracts many of the underlying details to higher-level applications

1. Presents data as named files– neat, clean abstraction: need not work with sector #s

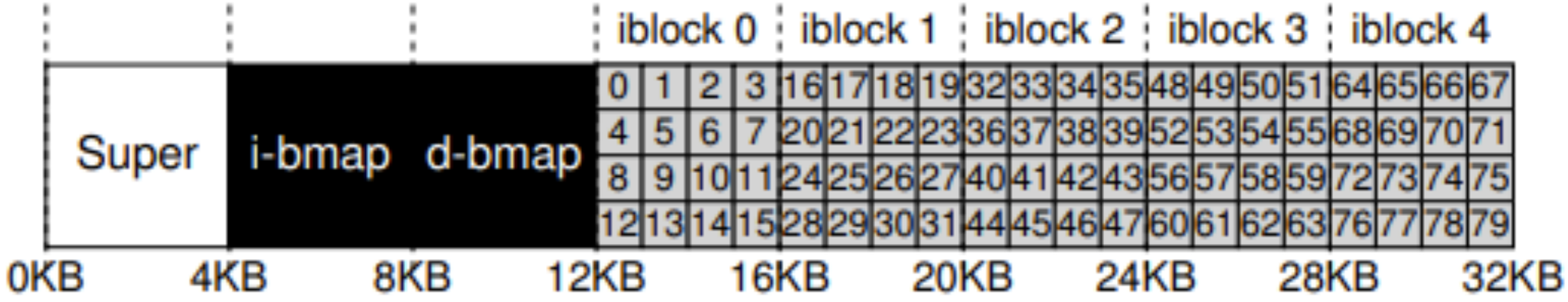
2. Can be byte-oriented instead of blocks/sectors

3. Offer protection and sharing among users

4. Ensures data reliability

Disk Layout for a FS

Disk layout in a typical file system:



- Data Structures:
  - File data blocks: File contents (not shown)
  - Inodes: low-level file number
  - Directories: File names pointing to inodes
  - Bitmaps: track which disk blocks are free
    - Data bitmap (d-bmap)
    - Inode bitmap (i-bmap)