Hard Disk Drives (HDDs)

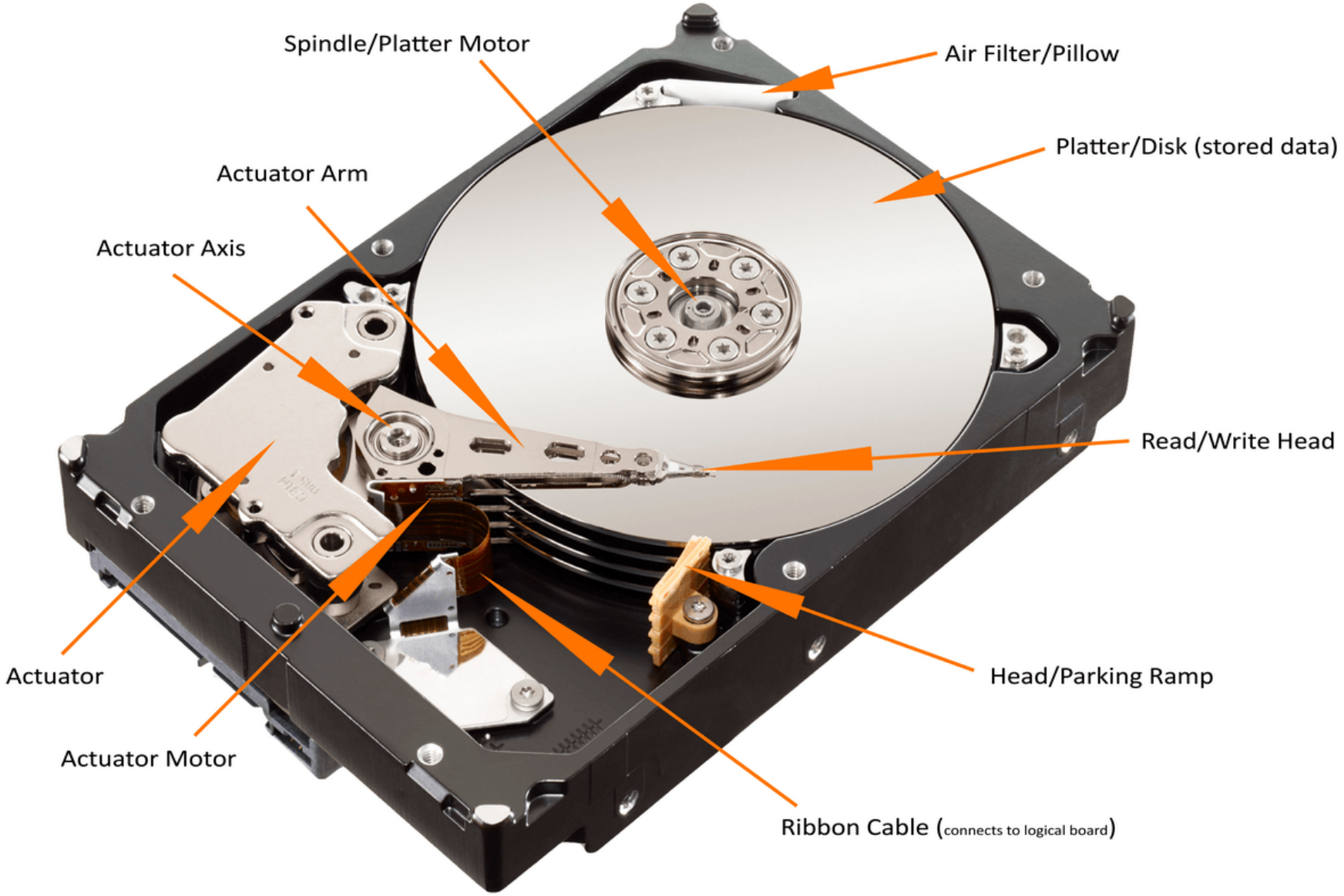
ICS332
Operating Systems

Henri Casanova (henric@hawaii.edu)

HDDs? Isn't it all SSDs Nowadays?

- Most likely, most of us have SSDs, not HDDs, in our personal laptop/ desktop computers
- A few years ago, some people predicted the death of HDDs, and predicted an SDD-only world
- That did not happen, even though SSD sales have increased in number of units
- But HDDs are still widely used today and in fact there has been some recent growth in their share
  - HDDs currently store ~80% of the world's data
- One explanation: HDDs are still much cheaper per GB than SDDs (less than $0.01/GB!), and AI requires us to store enormous datasets
  - Use faster SSDs for "hot" data, but massive numbers of HDDs for "cold" data
- So HDDs, for the foreseeable future, are here to stay
- And in fact, there are still HDD innovations! (the recent HAMR technology that greatly increases platter storage density)

HDD Structure

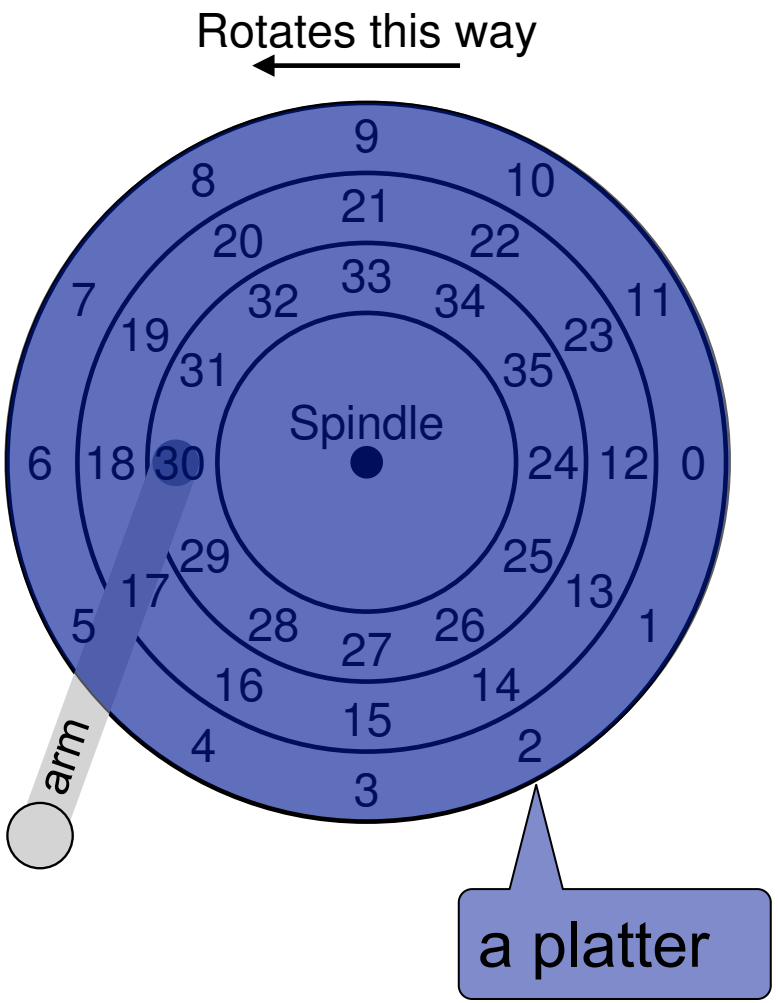


Platters, Tracks, Sectors

- A disk has multiple platters
- Each platter has multiple tracks
- Each track has multiple sectors
- Each sector stores 512 bytes of data, that can be read/written atomically by the HDD's controller

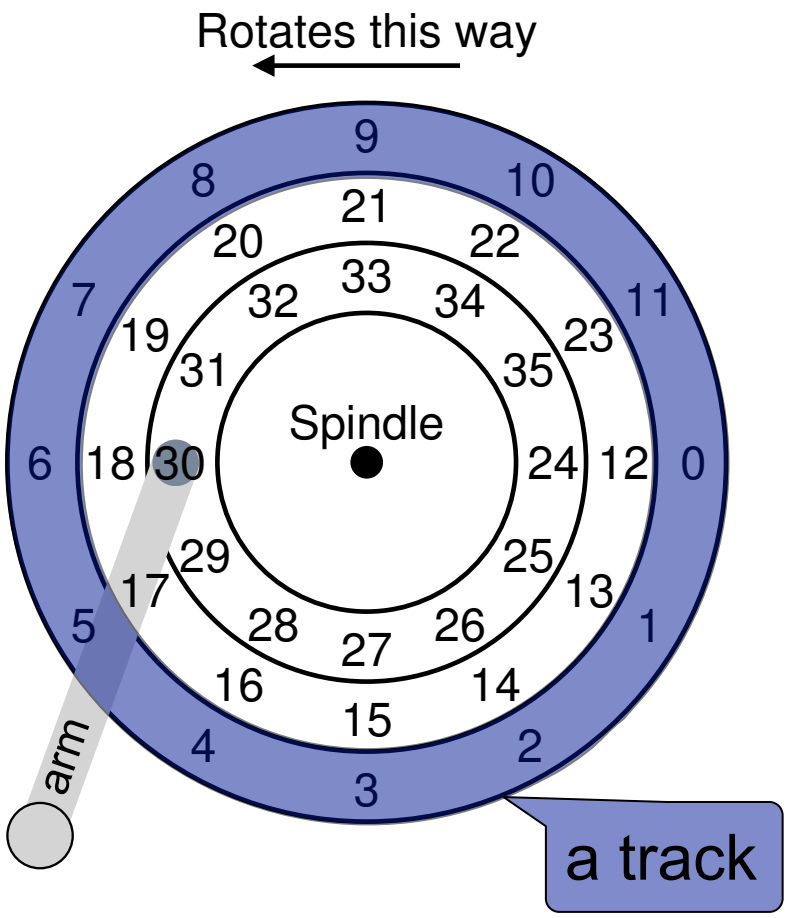
Platters, Tracks, Sectors

- A disk has multiple platters
- Each platter has multiple tracks
- Each track has multiple sectors
- Each sector stores 512 bytes of data, that can be read/written atomically by the HDD's controller



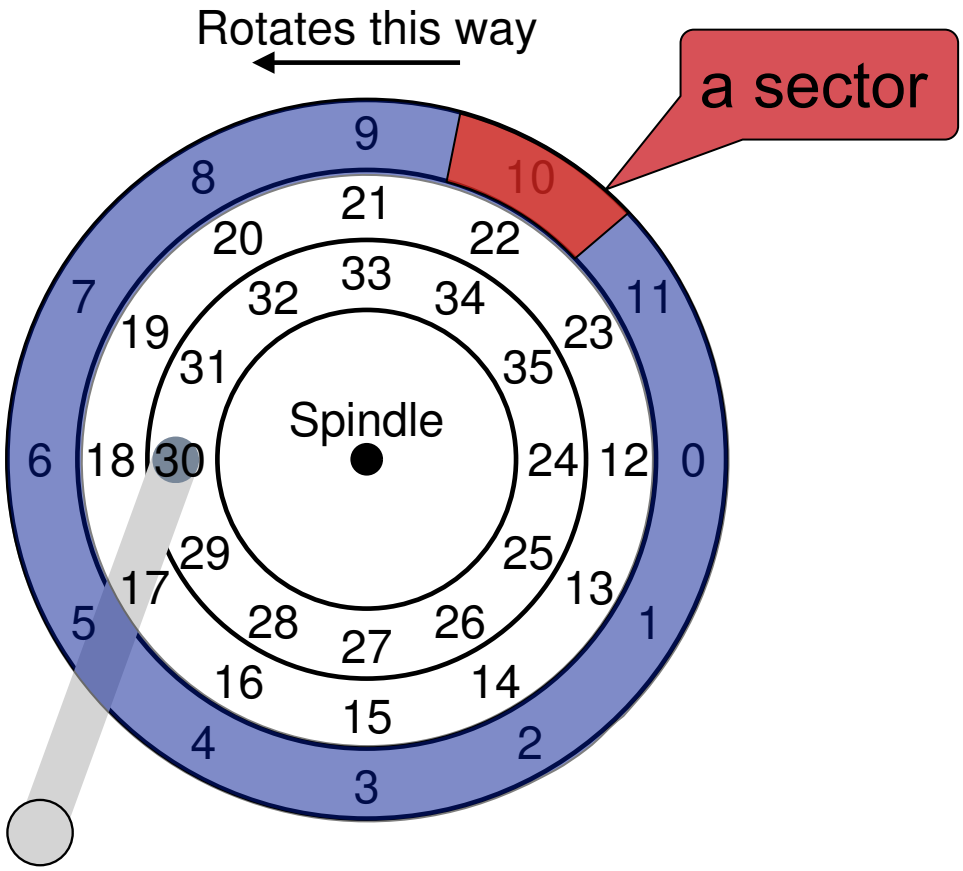
Platters, Tracks, Sectors

- A disk has multiple platters
- Each platter has multiple tracks
- Each track has multiple sectors
- Each sector stores 512 bytes of data, that can be read/written atomically by the HDD's controller



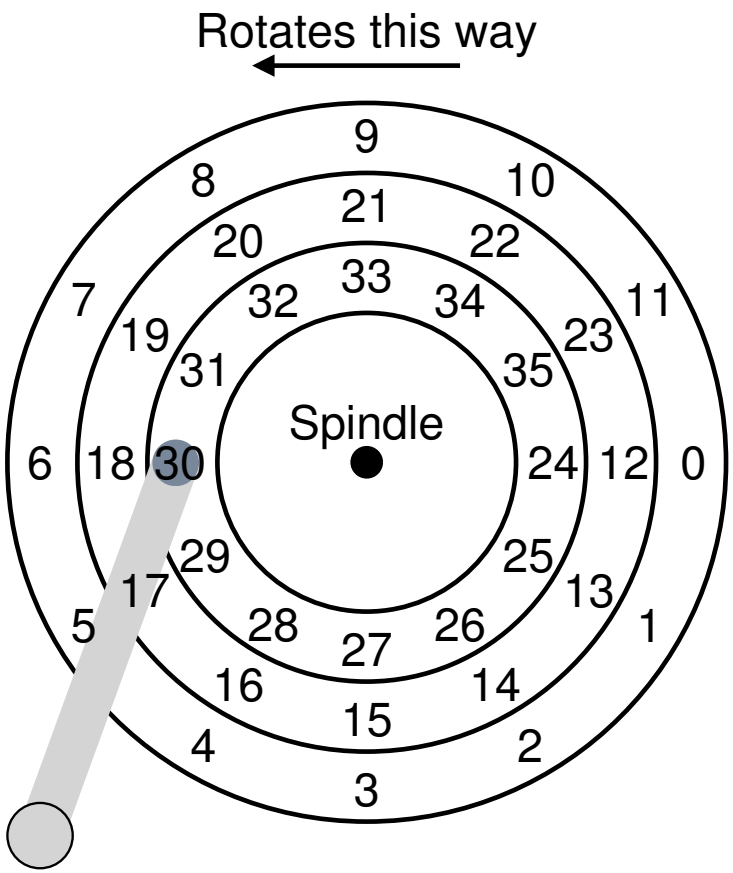
Platters, Tracks, Sectors

- A disk has multiple platters
- Each platter has multiple tracks
- Each track has multiple sectors
- Each sector stores 512 bytes of data, that can be read/written atomically by the HDD's controller



Surfaces, Cylinder

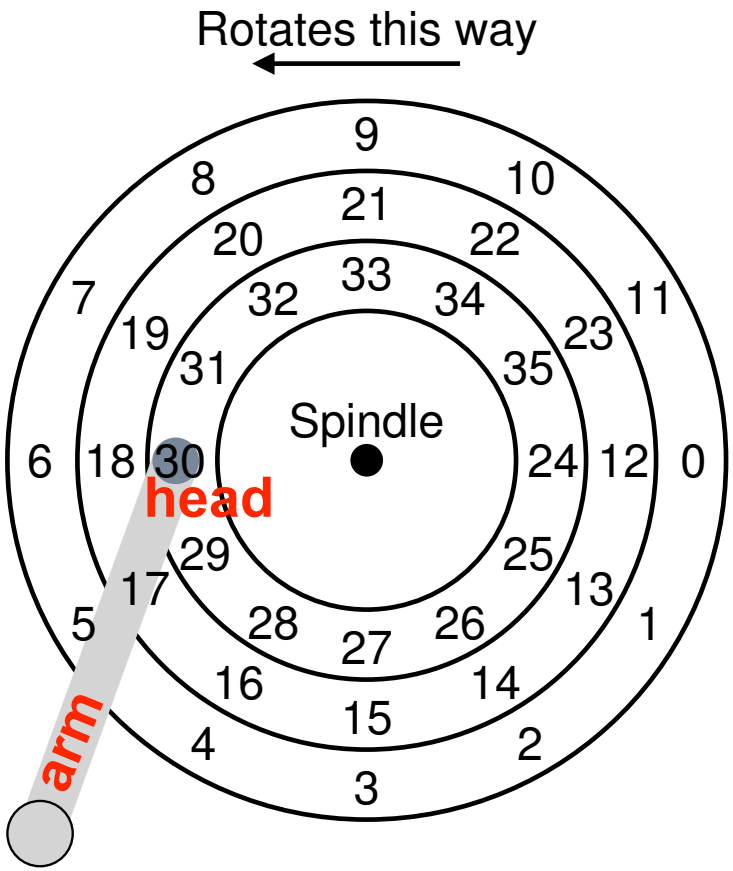
- Platters are actually double-sided with two surfaces
- Sectors have different sizes to deal with varying densities and radial speeds with respect to the distance to the spindle
- Outer tracks have in fact more sectors than inner tracks
- The set of all tracks that are the same distance away from the spindle on different platters is called a cylinder



a platter

Arm, Head

- Each platter surface has a head, which can read/modify the magnetic patterns on the surface
- All heads are attached to an arm that can move across the surface to position the head on a particular track



a platter

Hardware Interface

- The HDD's hardware interface makes it look like it's just an array of 512-byte blocks, indexed from 0 to N
- The HDD's hardware controller performs the translation from a single block # to a (platter #, track #, sector #) tuple
- This makes it easier to write OS device drivers for HDDs
  - The OS sees a HDD as just an array of blocks
- This also allows the HDD's controller to do all kinds of work transparently
  - i.e., "hide" bad (damaged) sectors by reserving some spares in each cylinder
  - Yay abstraction and virtualization

Performance

- HDDs are “slow” because they have moving parts inside of them
- Furthermore, the performance is highly depends on the current position of the head
- When doing a read/write operation, the latency of that operation depends on:
  - The rotational delay
  - The seek time
  - The transfer time

Rotational Delay

- Say the head is already positioned on the track that contains the sector that should be read/written
- One has to wait until the platter rotates until the head is positioned over the sector
- The rotational delay depends on the HDD's RPMs
  - Most HDDs today have more or less similar RPMs around 7,000 RPMs, but up to 10,000 RPMs is possible
- If the disk's RPM is x, then in the worst case, the rotational delay is about 1/x minutes
  - For 7,000 RPMs, that's about 8.5ms
  - An *eternity* from the CPU's perspective!!

Seek Time

- If the head is not positioned over the right track, then the arm has to move
  - This is called "seeking"
- There are physical/mechanical limits on how fast the arm can move
  - There is an acceleration phase, a coasting phase, a deceleration phase, and a settling phase (to finely adjust the position)
- This is, again, on the order of ms
  - Again, an eternity from the CPU's perspective
- However, if reading data sequentially, as opposed to randomly, few seeks are needed!
  - i.e., which happens hopefully often due to locality
- The ratio between the speed of random accesses and that of sequential accesses can be 100x-200x!
  - Also because there is cache (up to 16MB) that keeps sector data that the head went over on a track, assuming that these sectors will actually be needed too

Transfer Time

- Once the head is on the right track and at the beginning of the right sector, it can beginning read/write data
- This is the time for the sector to have passed entirely under the head
- This is typically on the order of microseconds
  - Much shorter than the seek time and the rotation delay

In the end:

$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

Disk Scheduling

- When processes may I/O requests to the disk, the OS basically sends to the disk a stream of block #, which correspond to various sectors in various tracks
- The disk scheduling question: in which order should these requests served?
- Given the current position of the head, given a list of blocks to access, which block should be read next?
- The goal is to minimize average access time
  - Which is typically achieved by trying to minimize seek time, but rotation delay also counts

Disk Scheduling: FIFO

- A basic algorithm is First In First Out
  - Answer requests in the order they come in
- Let's run:

```
./disk-modified.py -G -p FIFO -a 10,33,8,17,4
```

- The main problem is that this is inefficient, not serving requests that are "nearby"

Disk Scheduling: SSTF

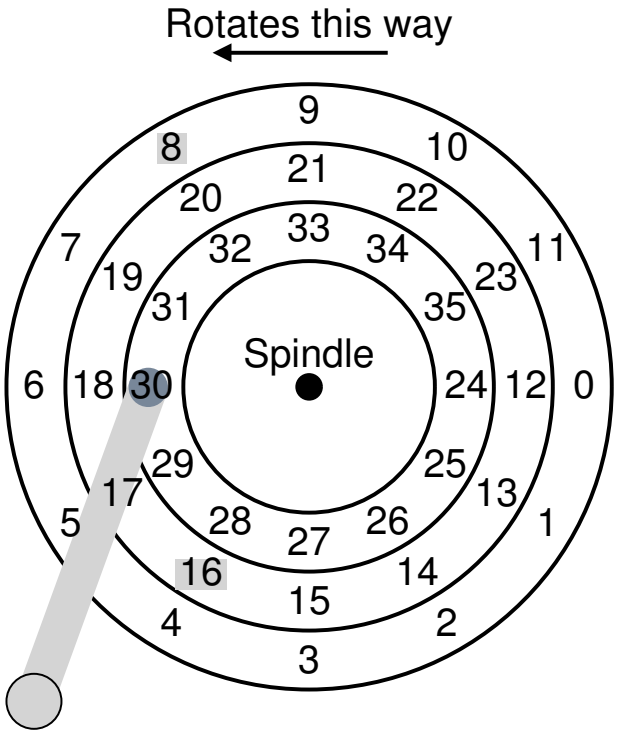
- A better algorithm is Shortest Seek Time First
  - Always answer the request that's on the nearest track
- Let's run:

```
./disk-modified.py -G -p SSTF -a 10,33,8,17,4
```

- The main problem is starvation: if there is a stream of request for the same track, then some requests will never be served

Disk Scheduling: SATF

- An even better algorithm: Shortest Access Time First
- Takes into account seek time and rotational delay to pick the next requests
  - Given the current position of the head, compute the time to serve each request, accounting for seeking and rotation; serve the and serve the “quickest” one



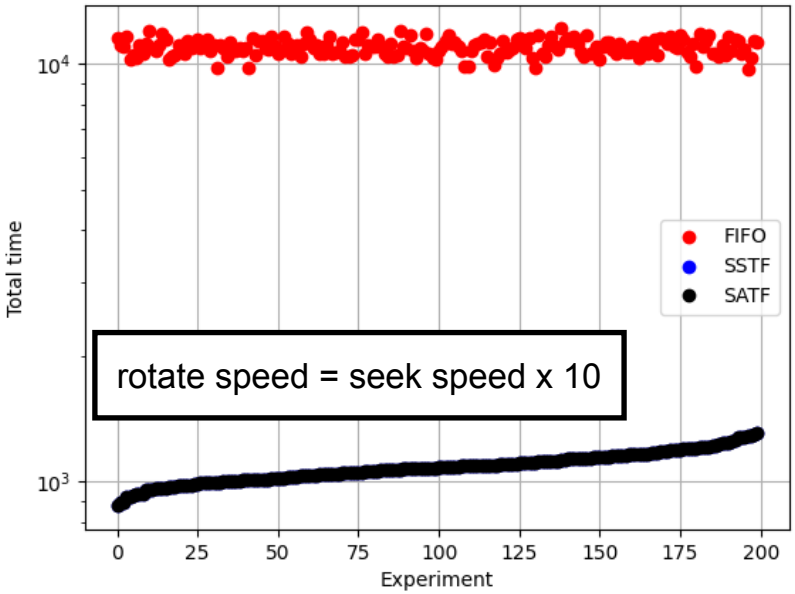
- Say the head is on sector 30, and there are two requests to serve: 8 an 16?
- What should we do?
  - Depends on the rotation speed and the arm speed

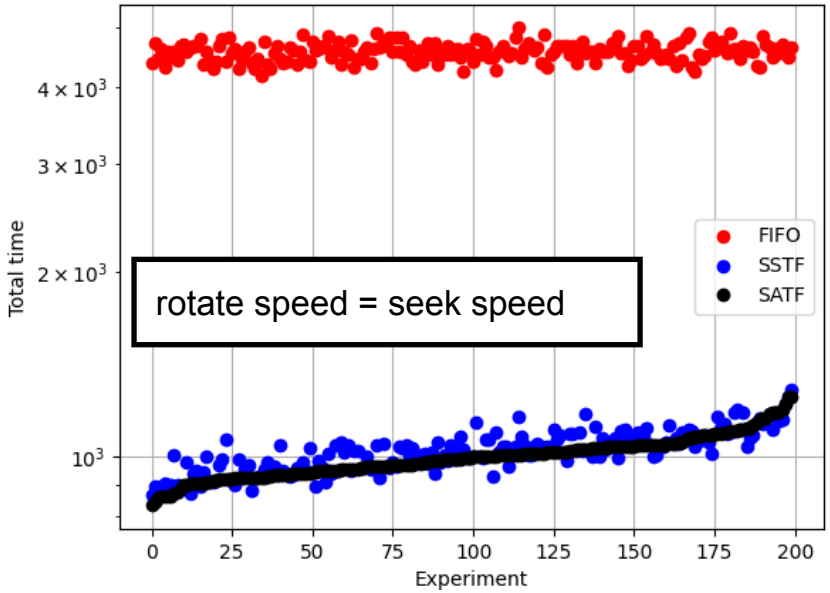
```
./disk-modified.py -G -p
SATF -a 10,33,8,17,4
```

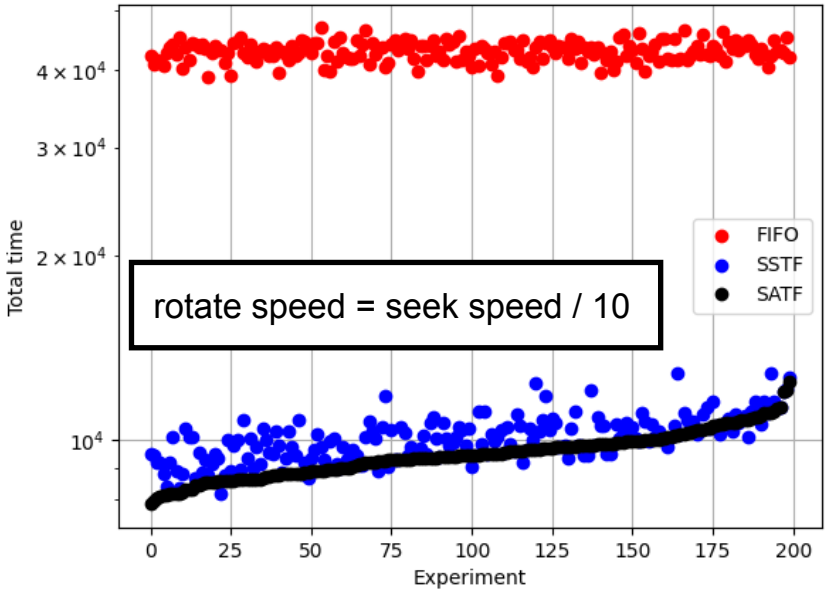
- Has the sam starvation problem: if there is a stream of request that are near each other, then some requests will never be served

Disk Scheduling Experiments

- We can use our Python program to do a simple experimental campaign (without the visualization)
- Here are some results!







Solving Starvation: Elevator

- Elevator Algorithms (SCAN, F-SCAN, LOOK, etc.)
- Just like an elevator in a building, the head goes from the inner track to the outer track, back and forth, serving requests along the way
  - The different algorithms are different tweaks of this main idea, e.g., when they reverse direction
- **Problem:** Requests that come in for the current track may have to wait until the "elevator" comes back
- This is really not good to minimize request service time
  - Just like when you "just missed" the elevator that is now going 30 floors up before coming back down to where you are
  - These algorithms are very far from a shortest job first idea, and
- Also, these algorithms completely ignore rotational delay!
  - On modern HDDs, rotation speed and arm speed are both of similar magnitudes (a few ms)
  - So it's really important to account for both

Disk Scheduling in the OS?

- OSes used to do disk scheduling
- This was really hard because the OS would need to know all the details of the HDD to do a good job
  - And besides, the block # are virtualized, so the OS doesn't know sector numbers
- Nowadays, disk scheduling is implemented in the HDD's controller
  - So the entire HDD is virtualized, and the API the device driver exposes to the OS is (essentially) **`read(block #)`** and **`write(block#, data)`**

HDD: Formatting

- Physical Formatting
  - Divides the disk into sectors
  - Fills the disk with a special data structure for each sector
    - A header, a data area (512 bytes), and a trailer
  - In the header and trailer is the sector number, and extra bits for error-correcting code (ECC)
    - The ECC data is updated by the disk controller on each write and checked on each read
    - If only a few bits of data have been corrupted, the controller can use the ECC to fix those bits
    - Otherwise the sector is now known as “bad” which is reported to the OS
  - All done at the factory before shipping
- Logical Formatting
  - The OS first partitions the disk into one or more groups of cylinders: the partitions
  - The OS then treats each partition as a separate disk
  - Then, file system information is written to the partitions
    - See the upcoming File System lecture notes

Bad Blocks

- Sometimes, data on the disk is corrupted and the ECC can't fix it
- Errors occur due to
  - Damage to the platter's surface
  - Defect in the magnetic medium due to wear
  - Temporary mechanical error (e.g., head touching the platter)
  - Temporary thermal fluctuation
- The OS then gets a notification that the I/O operation has failed
- Upon reboot, the disk controller can be told to replace a bad block by a spare
  - Each time the OS asks for the bad block, it is given the spare instead
  - The controller maintains an entire block map
  - Yay virtualization again
- Problem: the OS's view of disk locality may be very different from physical locality
  - Solution #1: Spares in each cylinders and a spare cylinder
    - Always try to find spares physically "close" to the bad block
  - Solution #2: Shuffle sectors to bring the spare block next to the bad block
    - Called sector splitting

Main Takeaways

- HDDs are slow due to moving parts
- Therefore, a key question is that of picking a good order in which disk blocks are read/written given a list of requested I/O operations
- Different algorithms exist, nowadays all implemented in the HDD controller, and in practice they are variations on the “elevator algorithm” theme
- The concept of formatting
- The controller hides bad block
- Modern OSes see the disk simply as a logical array of blocks, and all the complexity is in the controller

Conclusion

- HDDs are still very common, and won't disappear any time soon
- Their big advantage is that they can have very large capacity at low cost
- But they are not fast and are prone to failures
  - HDD hardware controllers do all they can to improve performance and reliability
- Next up: Solid State Drives (SSDs)