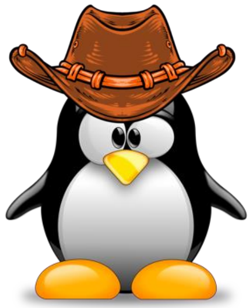
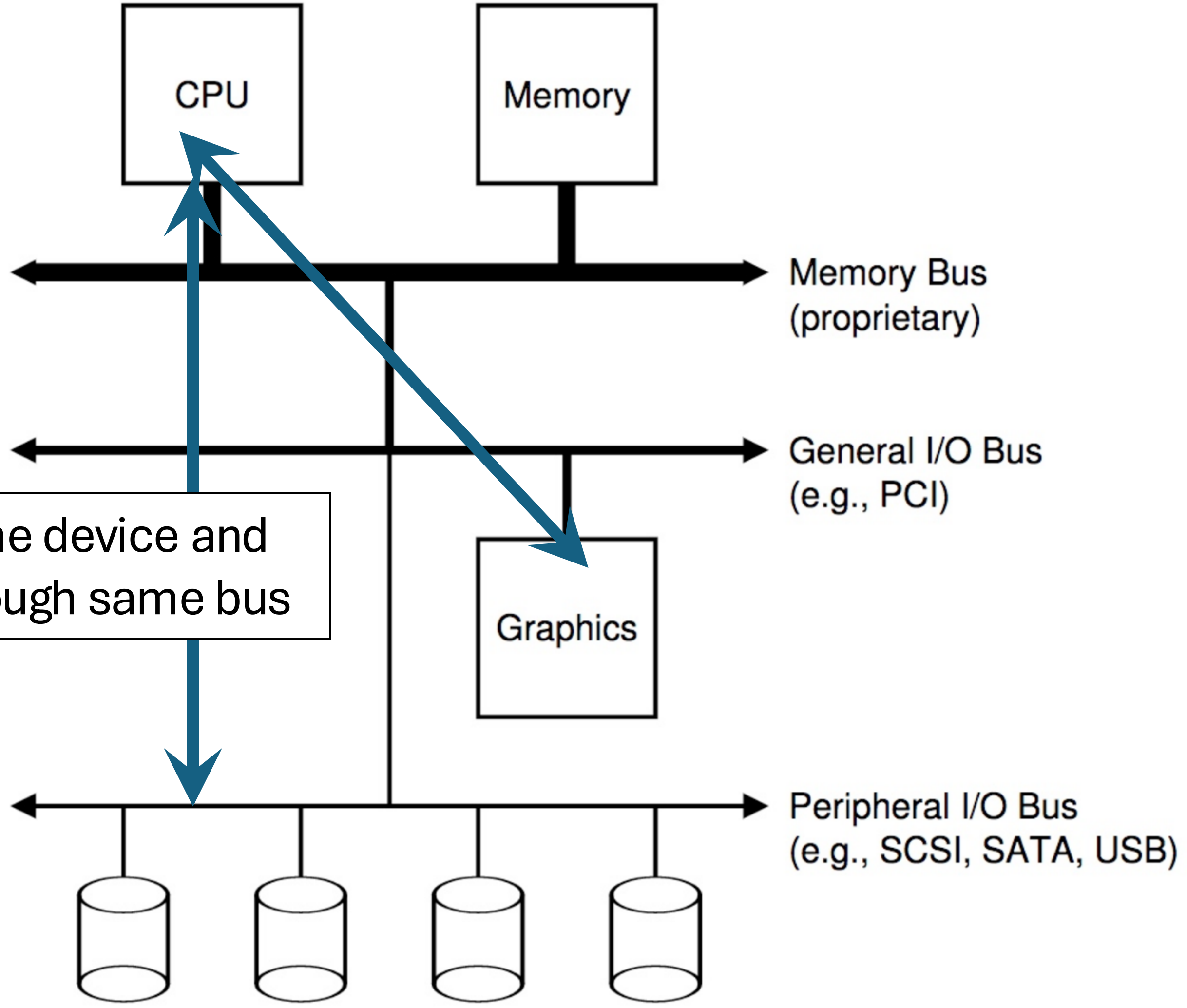


CSE 330: Operating Systems

Adil Ahmad

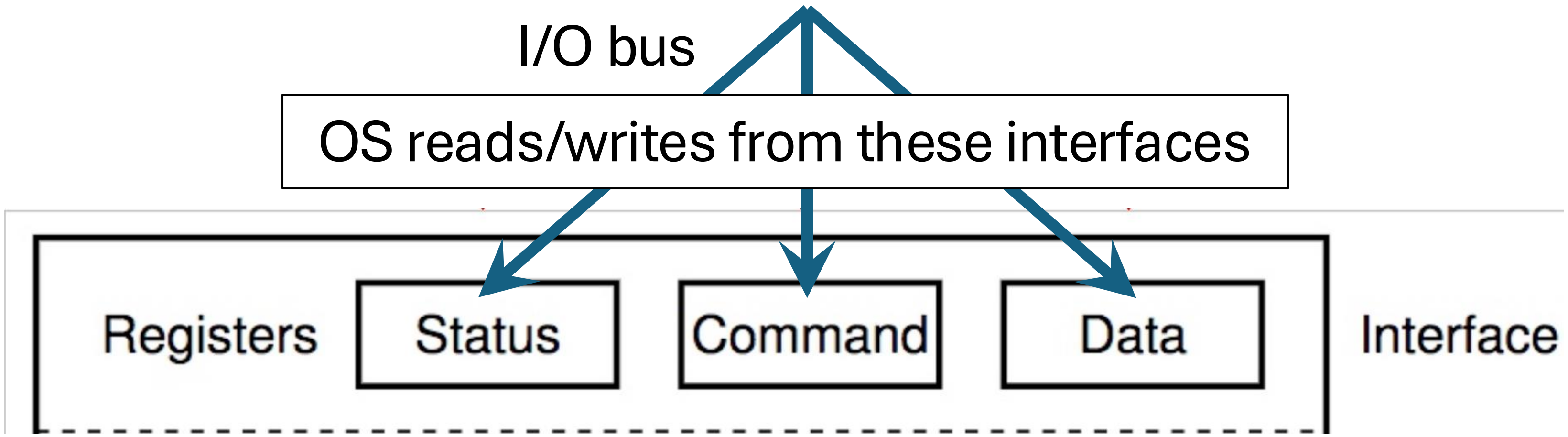
Lecture #15: Device examples, HDDs and SSDs

Bus interconnect allows two-way communication



Send messages to the device and **receive response** through same bus

Let’s see what a typical device looks like to the OS

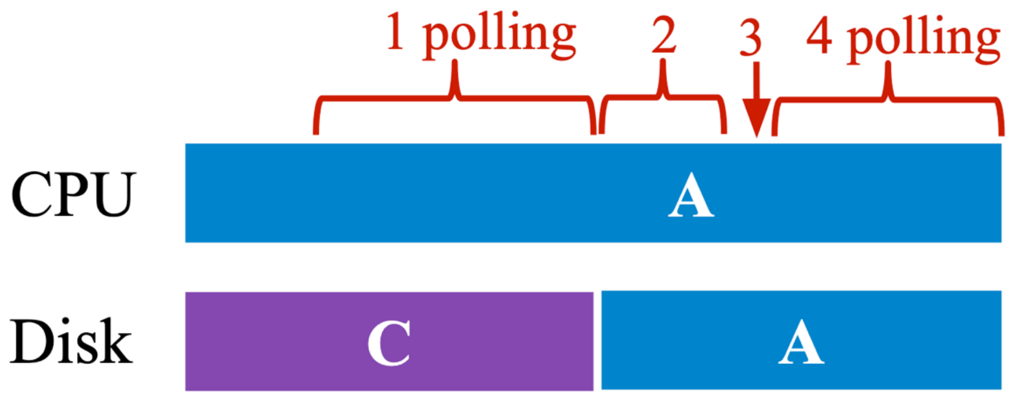


- **Status:** Tells the OS whether the device is ready, busy, initialized, etc.
- **Command:** OS will write what it needs here, and if device is ready, it will perform the corresponding operation
- **Data:** OS will write any information the device needs to complete its task at this location

Basic I/O protocol from the OS perspective

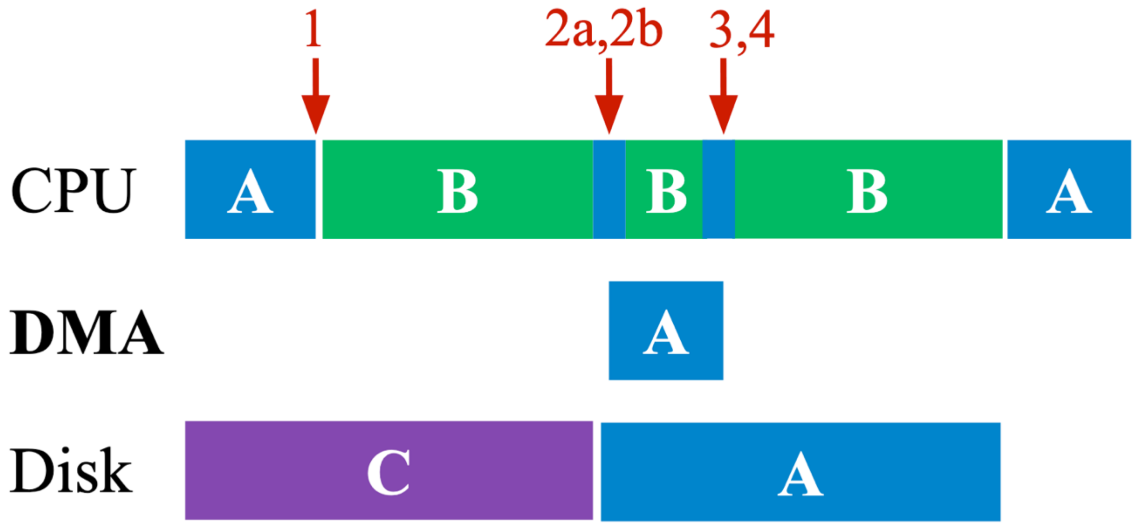
```
while (STATUS == BUSY)
    ; // spin
Write data to DATA register
Write command to COMMAND register
while (STATUS == BUSY)
    ; // spin
```

Solving the inefficiency problem using **interrupts**




```
while (STATUS == BUSY)                  //1
   wait for interrupt;
Write data to DATA register             //2
Write command to COMMAND register       //3
while (STATUS == BUSY)                  //4
   wait for interrupt;
```

Code-level illustration for Direct Memory Access (after)




```
while (STATUS == BUSY)               //1
   wait for interrupt;
Initiate DMA transfer                //2a
Wait for interrupt                   //2b
Write command to COMMAND register    //3
while (STATUS == BUSY)               //4
   wait for interrupt;
```

Is there a security threat with DMA?

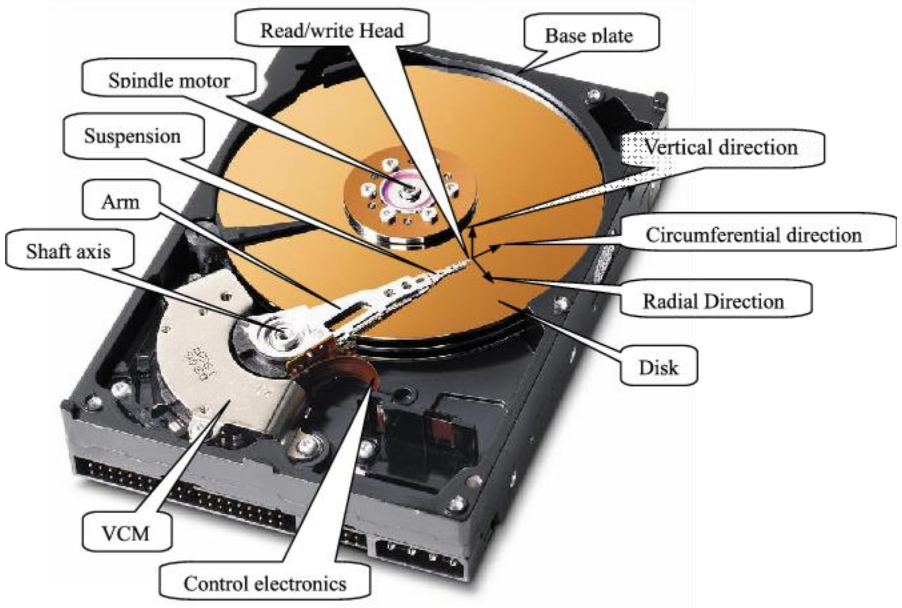
- Sadly, yes! Can directly access physical memory region
- Malicious devices can send DMA requests for any regions
  - e.g., belonging to the OS, secret keys, etc.
  - Check out the BadUSB attack! ☺
- OSs can also be malicious and leverage DMA to steal data
  - **Any example scenario?**
  - Consider the OS running inside your VirtualBox setups
  - If it is malicious, it can try to use a device (e.g., keyboard) to access data from your host OS using DMA

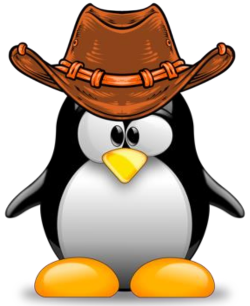
Adding translation tables for DMA protection

- Enabled by a hardware unit inside the CPU called the *I/O memory management unit (IOMMU)*
  - Like the memory management unit (MMU)

- The OS sets **I/O page tables** for DMA and all access to the DRAM from devices occurs through **I/O virtual address**

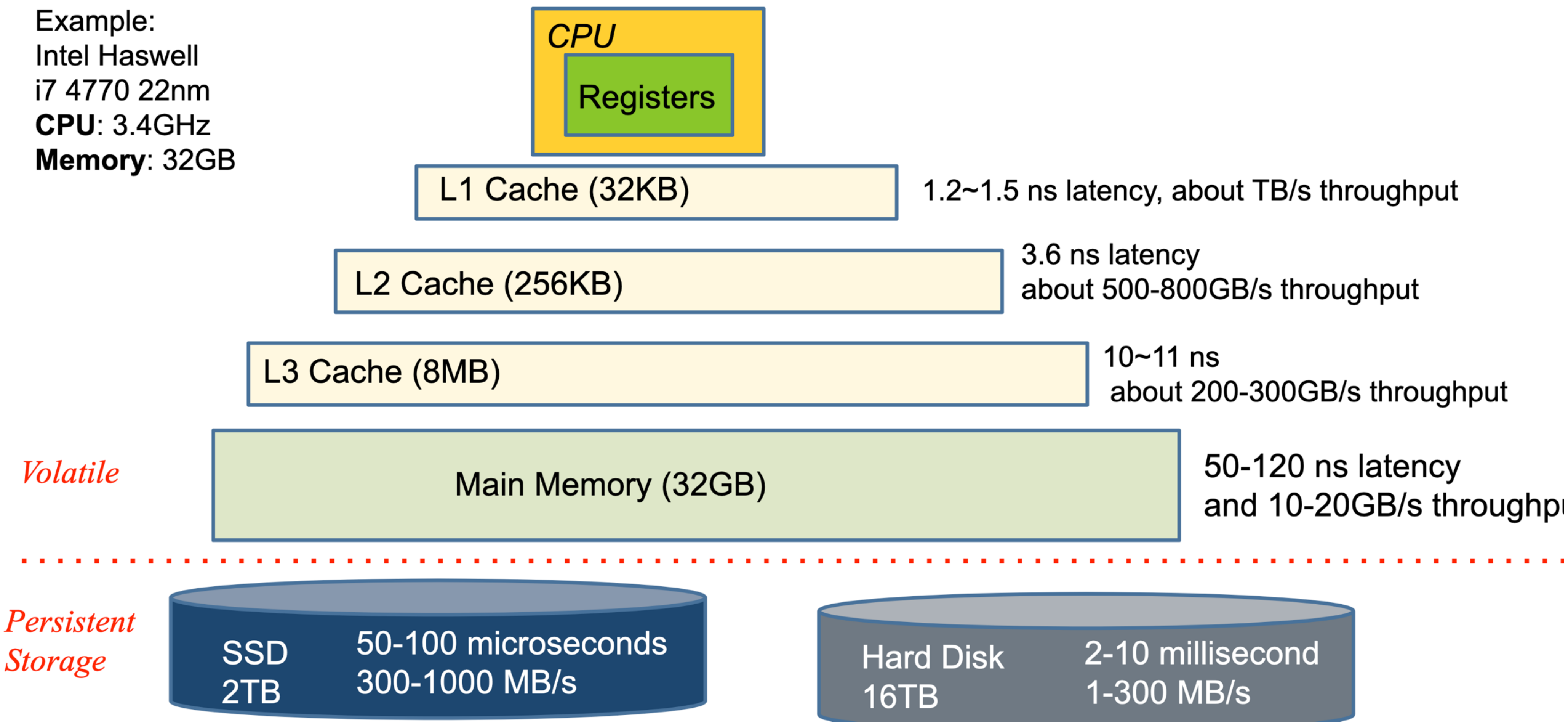
- Like normal page tables, the OS can define some regions as read-protected, write-protected, etc.





Device example: Hard disk drives (HDDs)

The memory-storage hierarchy of modern computers



Basic interface of an HDD

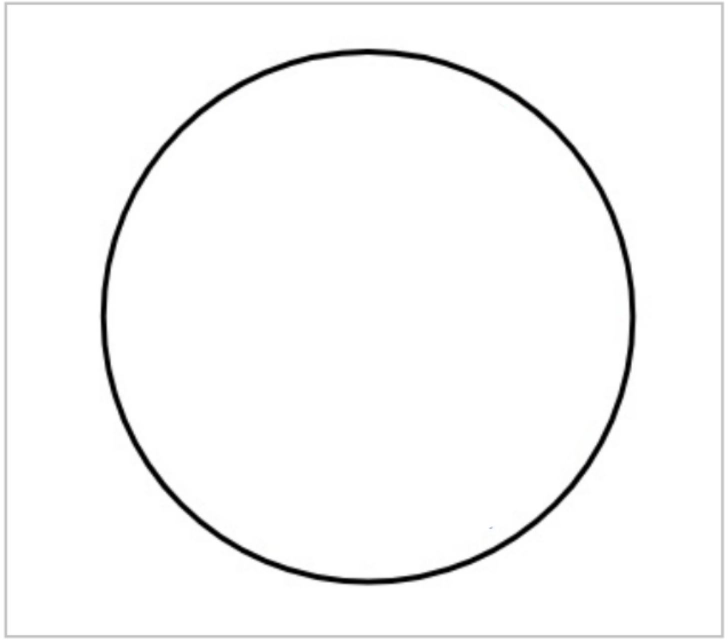
- A magnetic HDD has a **sector-addressable** space
  - You can think of a disk as an array of sectors
  - Each sector (also called "logical block") is the smallest unit of transfer
- Sectors are typically 512 or 4096 bytes
  - 100 GB disk of 512-byte sectors = 209715200 total sectors
- HDDs support two general I/O commands:
  - Read from sectors (blocks)
  - Write to sectors (blocks)

High-level description of an HDD’s internal structure

- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost

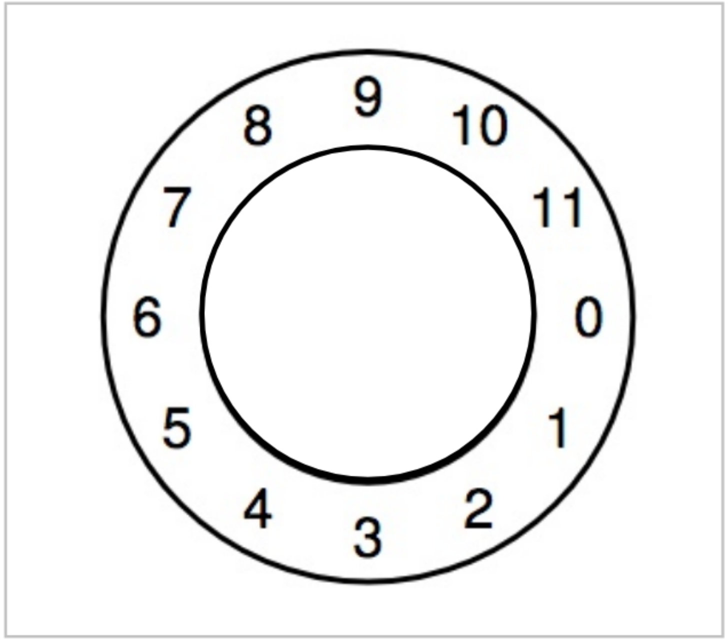
Internal workings of a Hard Disk Drive (HDD)

Platter
Covered with a magnetic
film



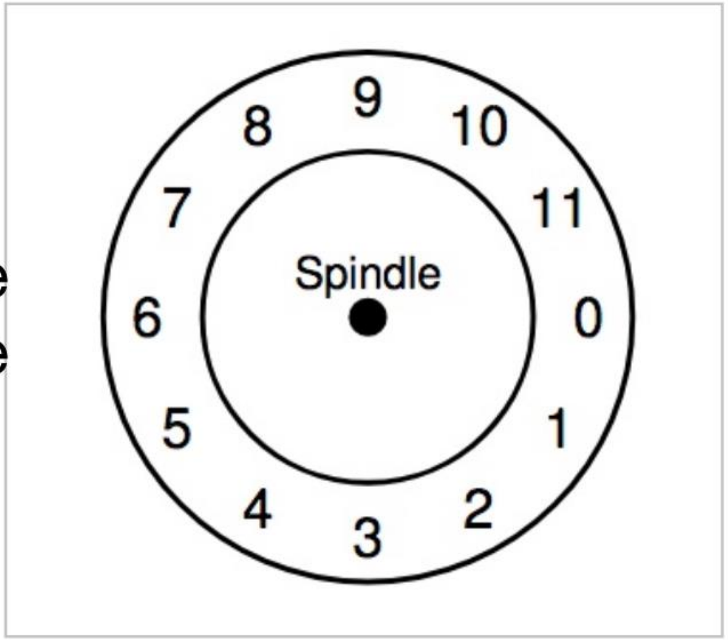
Internal workings of a Hard Disk Drive (HDD)

A single track example



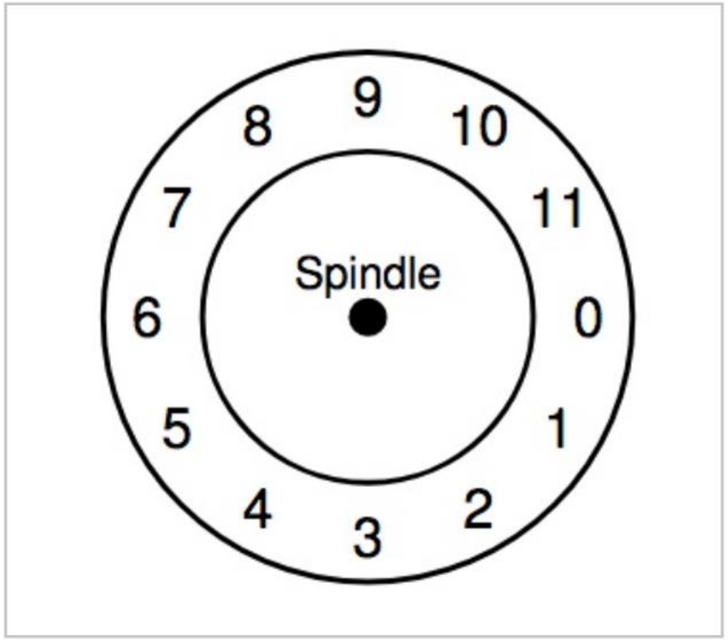
Internal workings of a Hard Disk Drive (HDD)

Spindle in the center of the surface



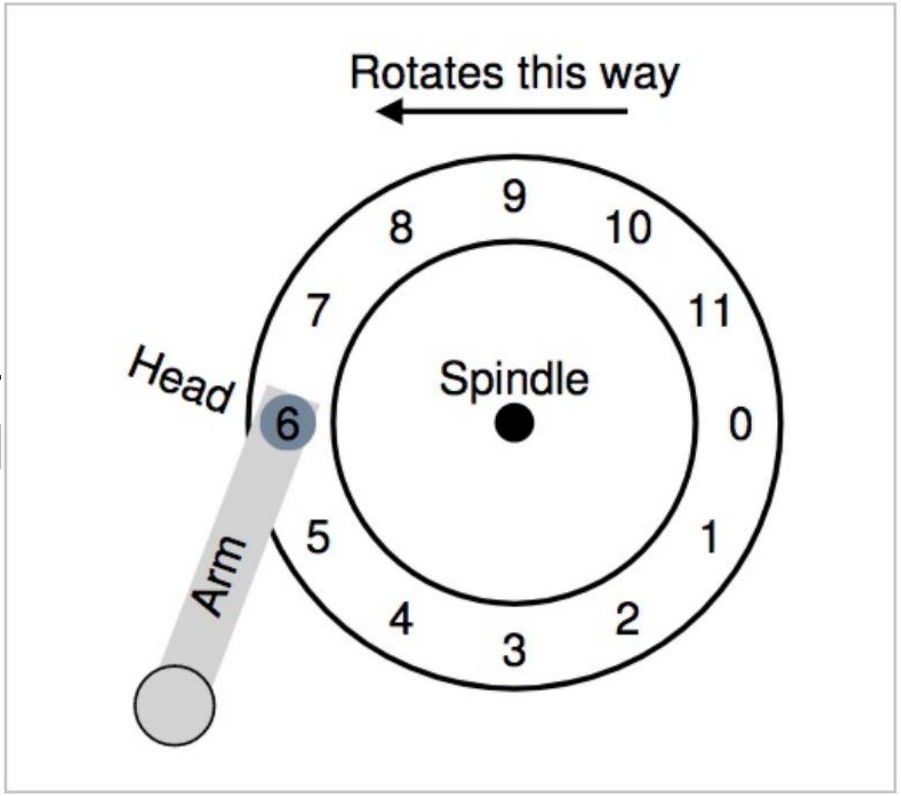
Internal workings of a Hard Disk Drive (HDD)

The track is divided into numbered sectors

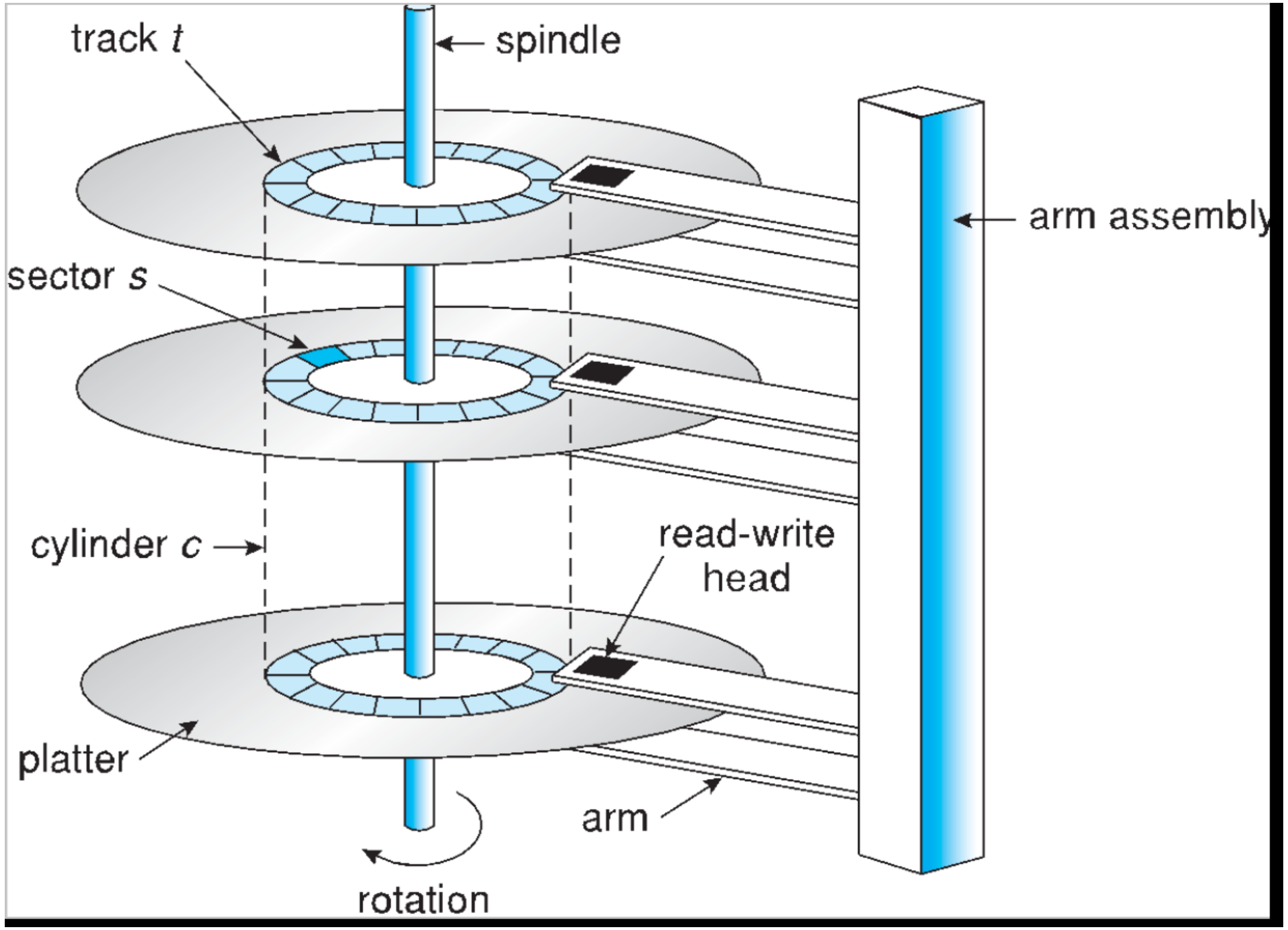


Internal workings of a Hard Disk Drive (HDD)

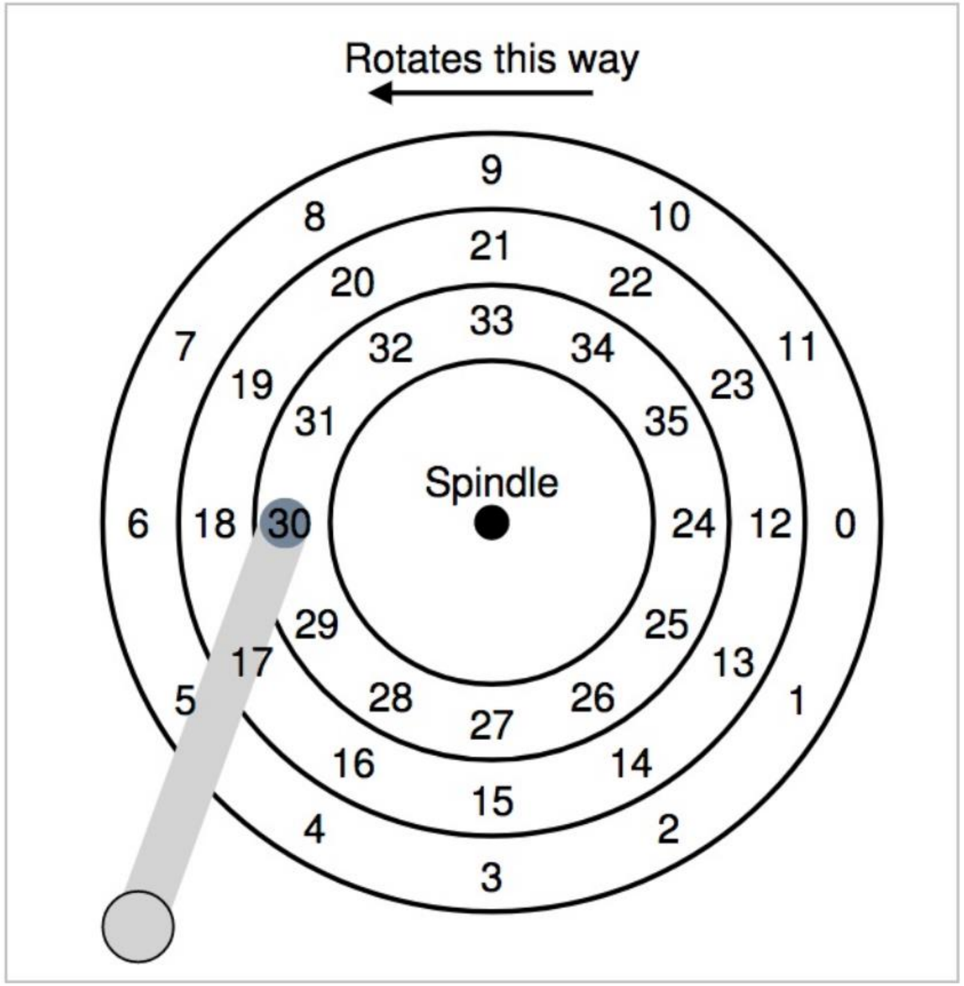
A single track + an arm + a head



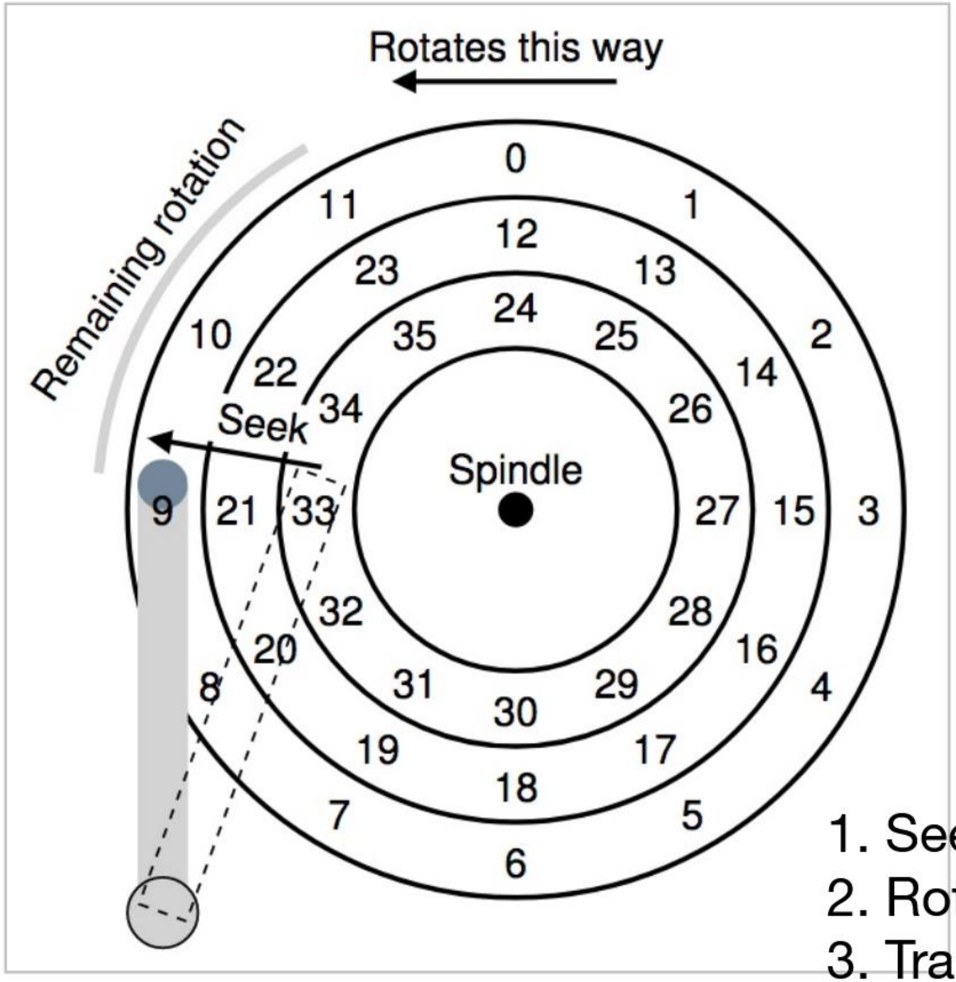
Internal workings of a Hard Disk Drive (HDD) – 3D view



Let's try to read sector 0. How would you do it?

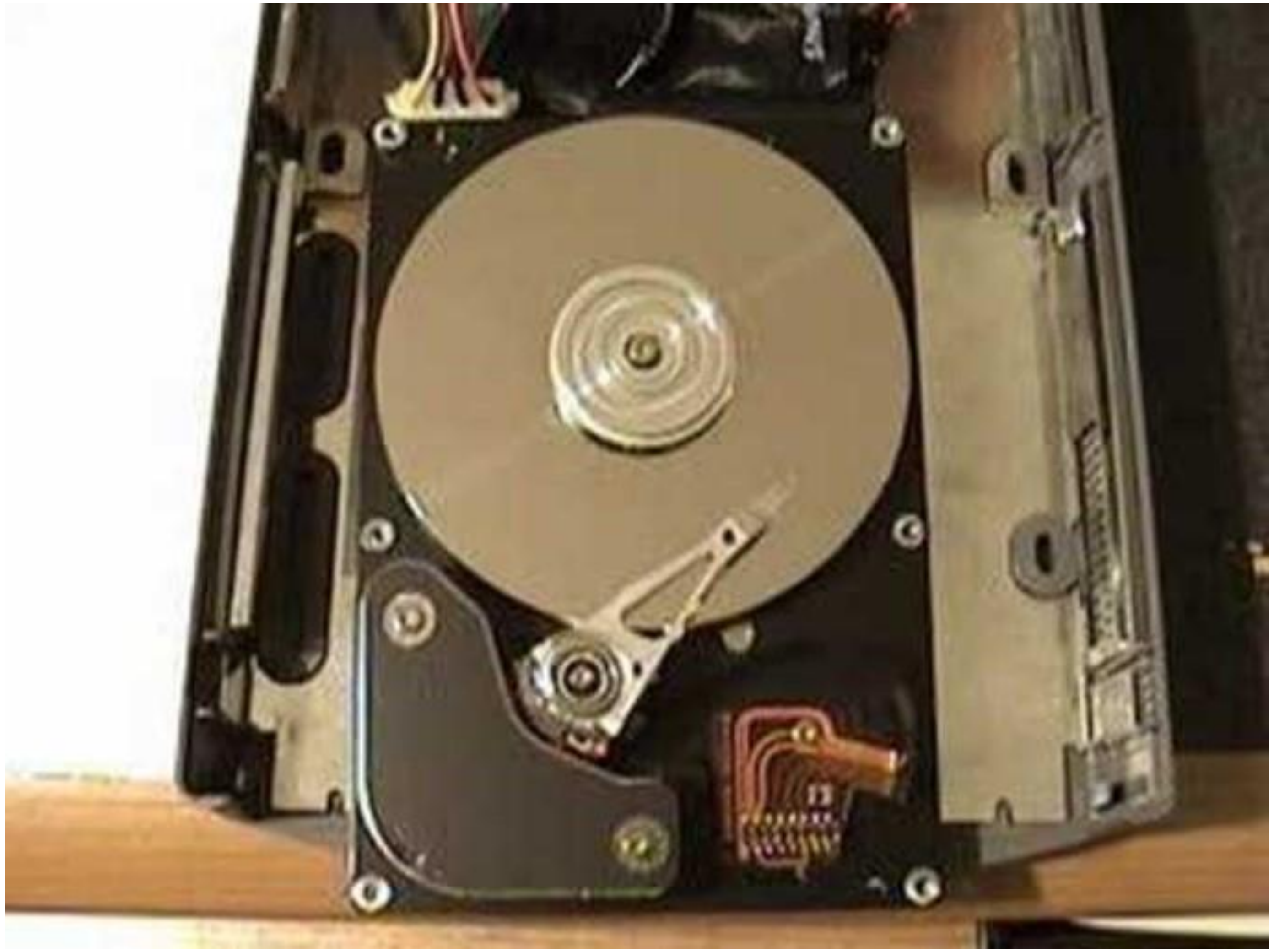


Let's try to read sector 0. How would you do it?



1. Seek for right track
2. Rotate (sector 9 → 0)
3. Transfer data (sector 0)

Let's watch a video on the *internal HDD workings!*



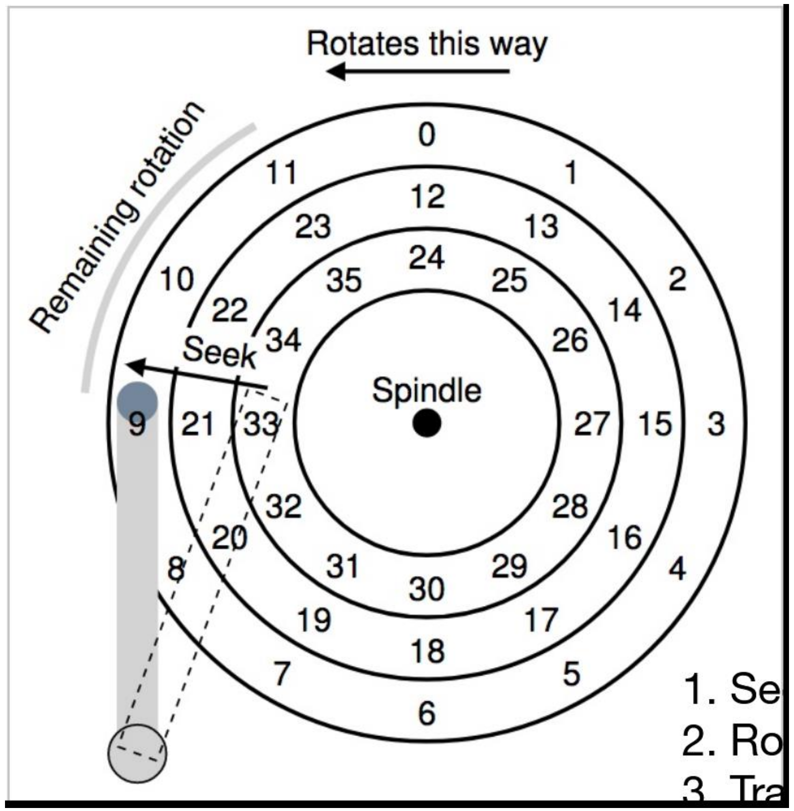
https://www.youtube.com/watch?v=9eMWG3fwiEU&t=30s

Understanding workings of HDD is important for the OS

- Disk **performance depends significantly** on the working mechanism
- The disk latency can be calculated by considering the three steps that the HDD must perform:
  - HDD I/O latency = $\textbf{Latency}_{\textbf{seek}}$ + $\textbf{Latency}_{\textbf{rotate}}$ + $\textbf{Latency}_{\textbf{transfer}}$
- Let's see how many milliseconds are taken by these *(seek, rotate, and transfer)* operations!

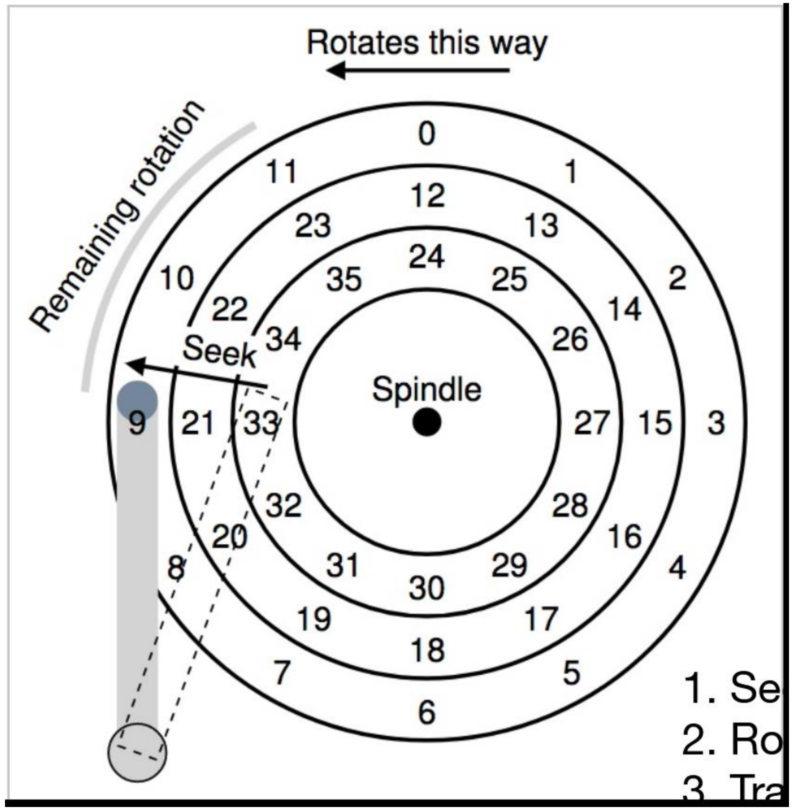
Seek, Rotate, Transfer

- Seek may take several milliseconds (ms)
  - Depends on hardware
- Settling along can take 0.5 - 2ms
- Entire seek (**L_{seek}**) takes 4 – 5 ms for a decent HDD, but it may take up to 10 ms for slow (past) HDDs



Seek, **Rotate**, Transfer

- Depends on Rotation Per Minute (RPM) supported by the HDD
  - 7200 RPM is common nowadays
  - 15000 RPM is high end
  - Old computers may have 5400 RPM disks
- L_{rotate} for 7200 RPM
  - = 7200 rotations / 60 s = 120 rots / sec
  - = 1 sec / 120 rotations = **8.3 ms** / rotation
- It may take 4.2 ms **on average** to rotate to target (0.5 * 8.3 ms)

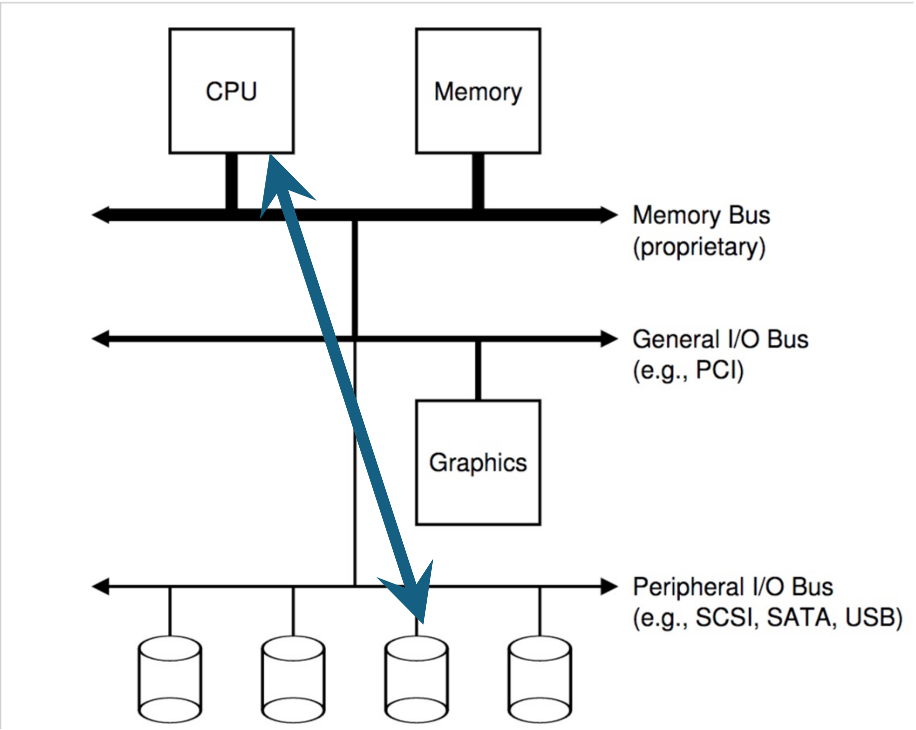


Seek, Rotate, **Transfer**

- Relatively fast
  - Depends mostly on the link speed
  - SATA is the link for HDDs today

- 100+ MB/s is typical for SATA
  - Up to **600MB/s** for SATA III)

- Calculating $\mathbf{L_{transfer}}$
  - 1s / 100MB = 10ms / MB = 4.9us /sector
    - Assuming 512-byte sector



Examining workload suitability for HDDs

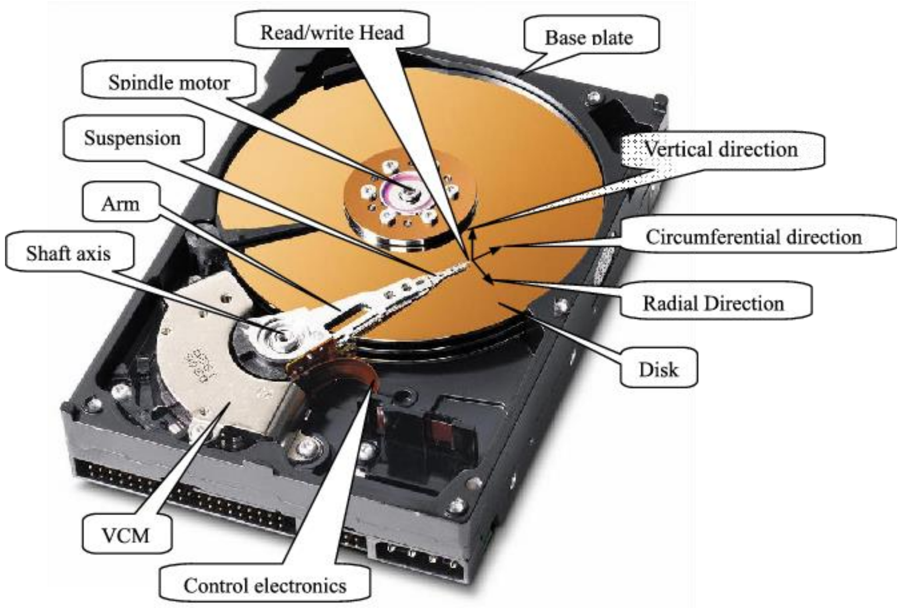
- Seeks and rotations are slow while transfer is relatively fast
  - seek = ~4 – 10 ms, rotate = ~4 – 5 ms, transfer = ~5 us
- **What kind of workload is best suited for disks?**
  - **Sequential I/O**: access sectors in order (transfer dominated)
- **Random** workloads access sectors in a random order (seek+rotation dominated)
  - Typically, slow on disks
  - Never do random I/O unless you must!
  - For instance, **quicksort** is a terrible algorithm for disk

Let's do some disk performance calculations together

- Seagate Enterprise SATA III HDD
  - Rotations-per-minute = 7200
  - Average seek = 4.16 ms
  - Max transfer = 500 MB/s
- **How long does an average 4KB read take?**
  - **Latency$_{seek}$** + **Latency$_{rotate}$** + **Latency$_{transfer}$**
  - Transfer = $\frac{1s}{500 * 1024\ KB/s} * 4$ = 0.00000781248 seconds
    - = ~8 us
  - 4.16ms + 4.2 ms + 8 us = 8.368 ms

A fun look at the oldest commercial HDD

- 1956 IBM RAMDAC computer
  - 5M (7-bit) characters
  - 50 x 24" platters
  - Access time = ~1 sec



Disk (HDD) scheduling algorithms

Disk scheduling introduction and requirements

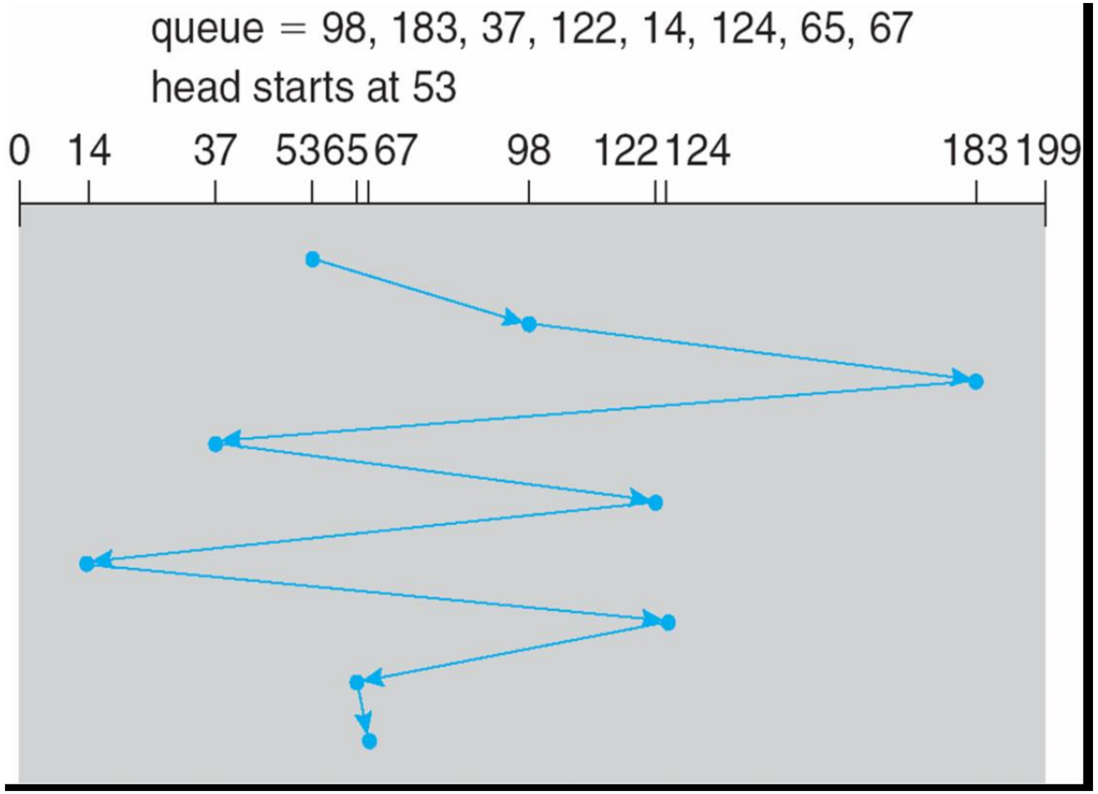
- The OS must answer: “Given a stream of I/O requests, in what order should they be served?”
- Strategy: **reorder** requests to meet certain goals
  - Performance (e.g., by making I/O sequential)
  - Fairness
  - Consistent latency
- **Performance objective:** minimize seek + rotation time
  - Minimize the distance the head needs to go

Queuing and disk scheduling algorithms

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- OSs also maintain their own buffers in-memory before sending a request to the disk
- *Disk scheduling algorithms:*
  - Algorithms that schedule the orders of disk I/O requests

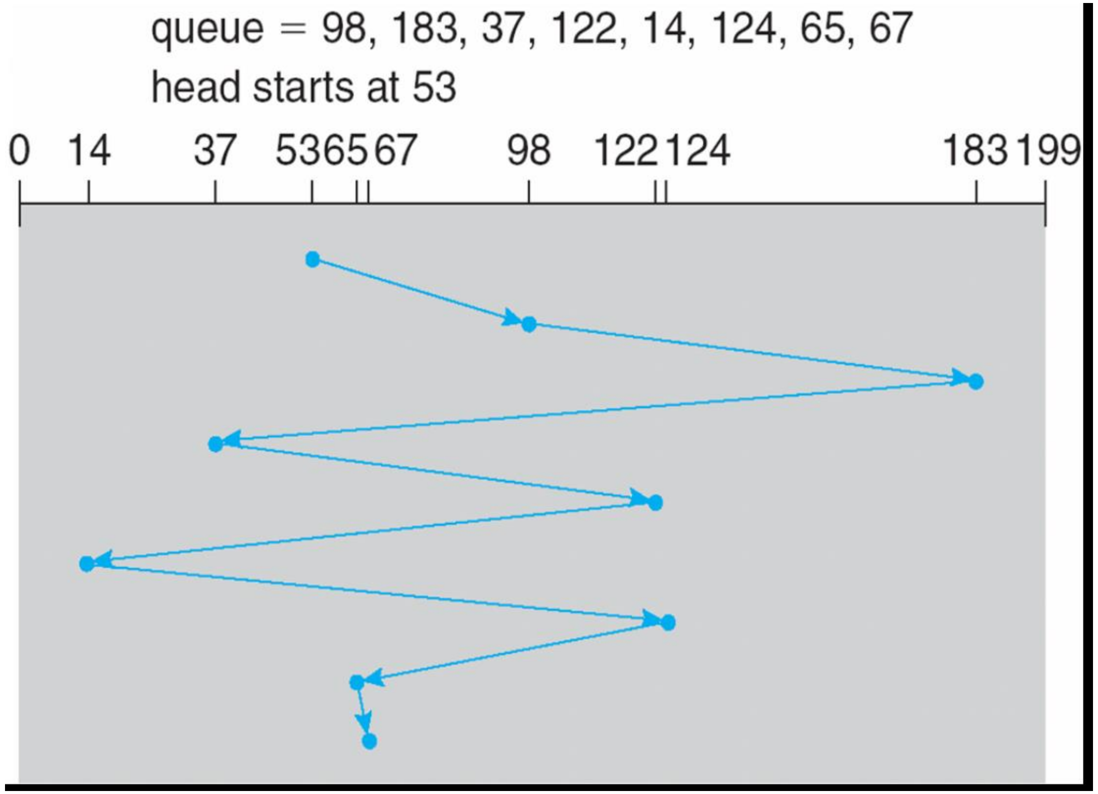
The simplest approach: First-In-First-Out (FIFO)

- **Idea:** Serve the I/O request in the order they arrive



Is there any problem with FIFO?

- **Idea:** Serve the I/O request in the order they arrive



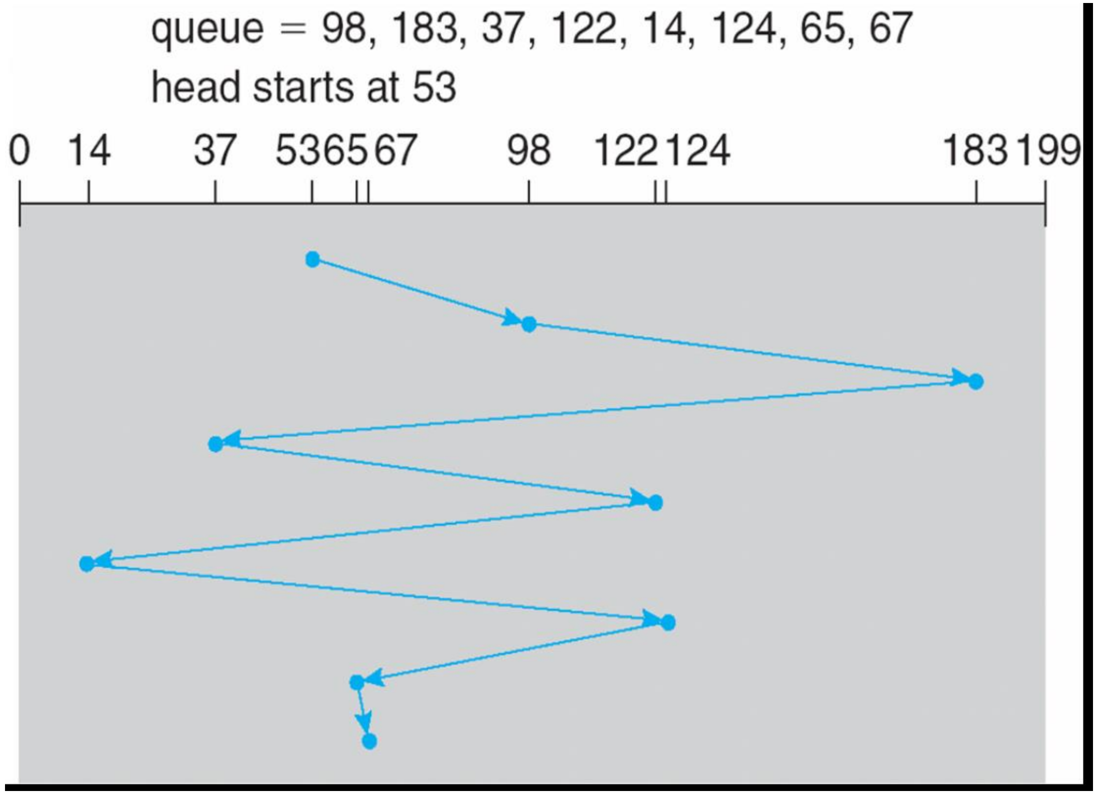
Problem: many total head movement of 640 cylinders

Calculating the number of head movements

- **Idea:** Serve the I/O request in the order they arrive

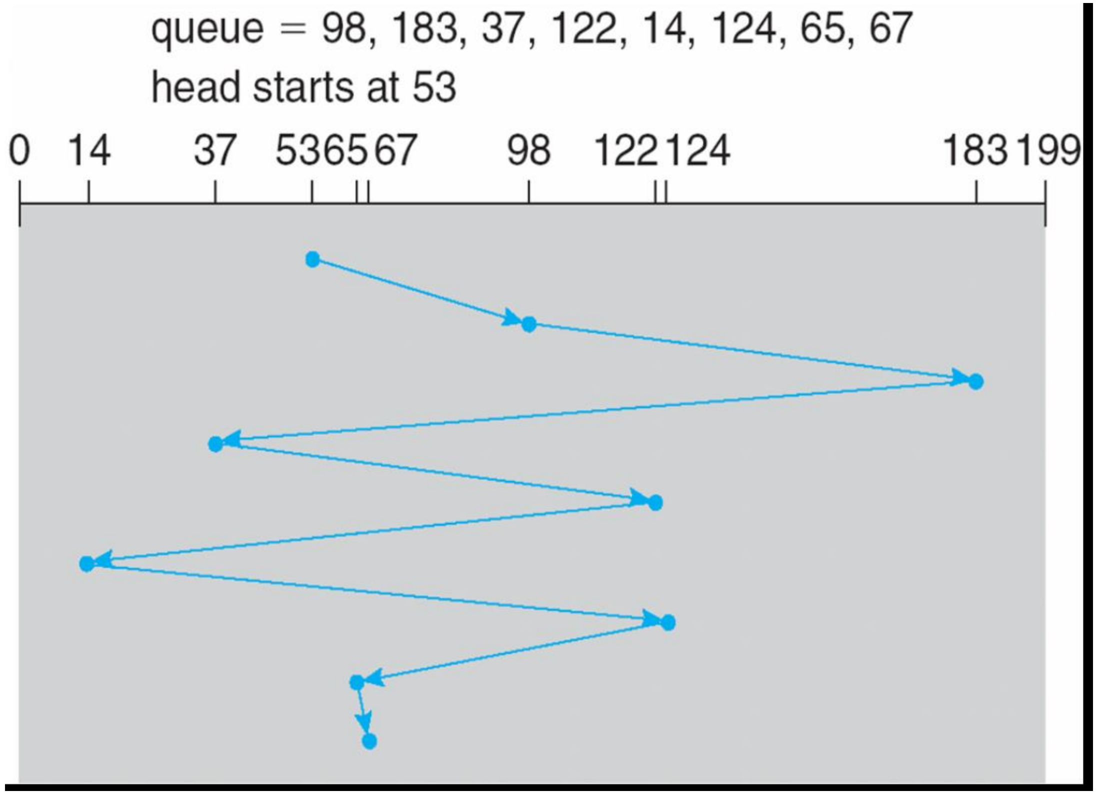
640 cylinders:

- (98 – 53)
- (183 – 98)
- (183 – 37)
- (122 – 37)
- (122 – 14)
- (124 – 14)
- (124 – 65)
- (67 – 65)



How can we minimize head movement?

- **Idea:** Serve the I/O request in the order they arrive



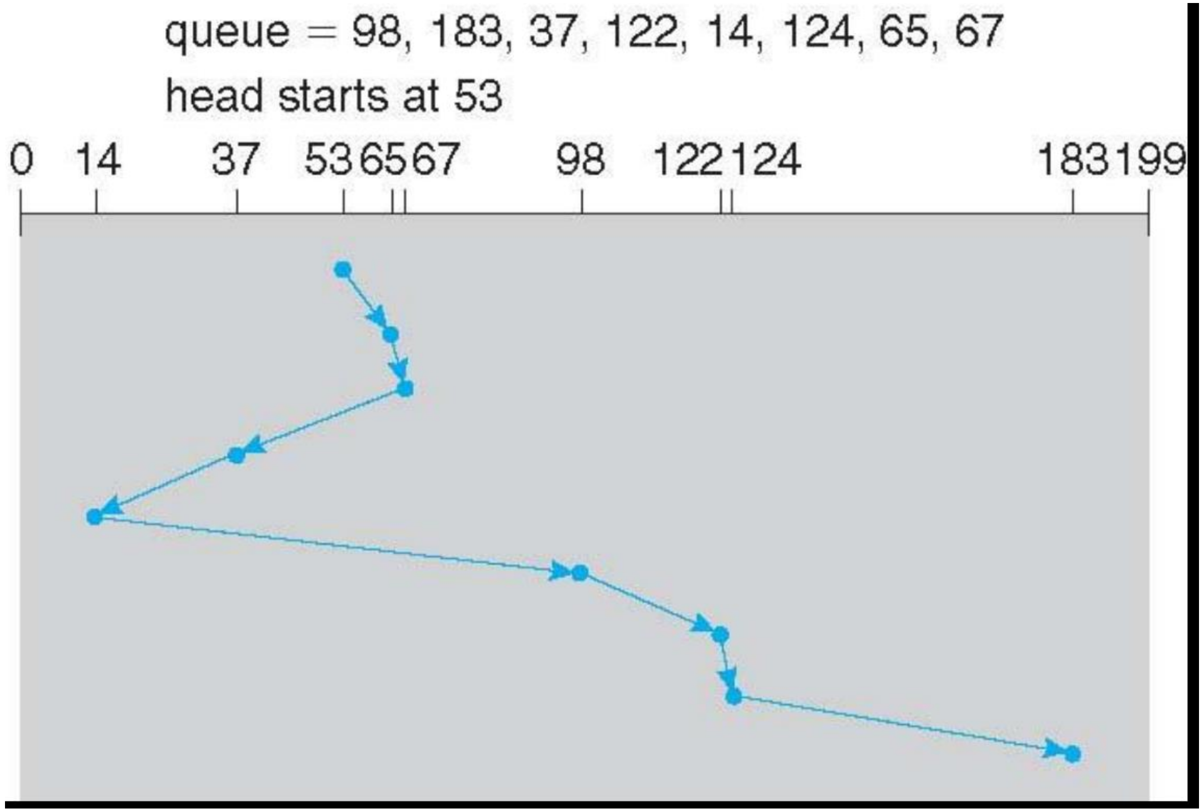
640 cylinders:

- (98 – 53)
- (183 – 98)
- (183 – 37)
- (122 – 37)
- (122 – 14)
- (124 – 14)
- (124 – 65)
- (67 – 65)

Shortest *Positioning* Time First (SPTF)

- **Idea:** Select the request with the least time for seeking and rotating

- Illustration shows total head movement of **236** cylinders.

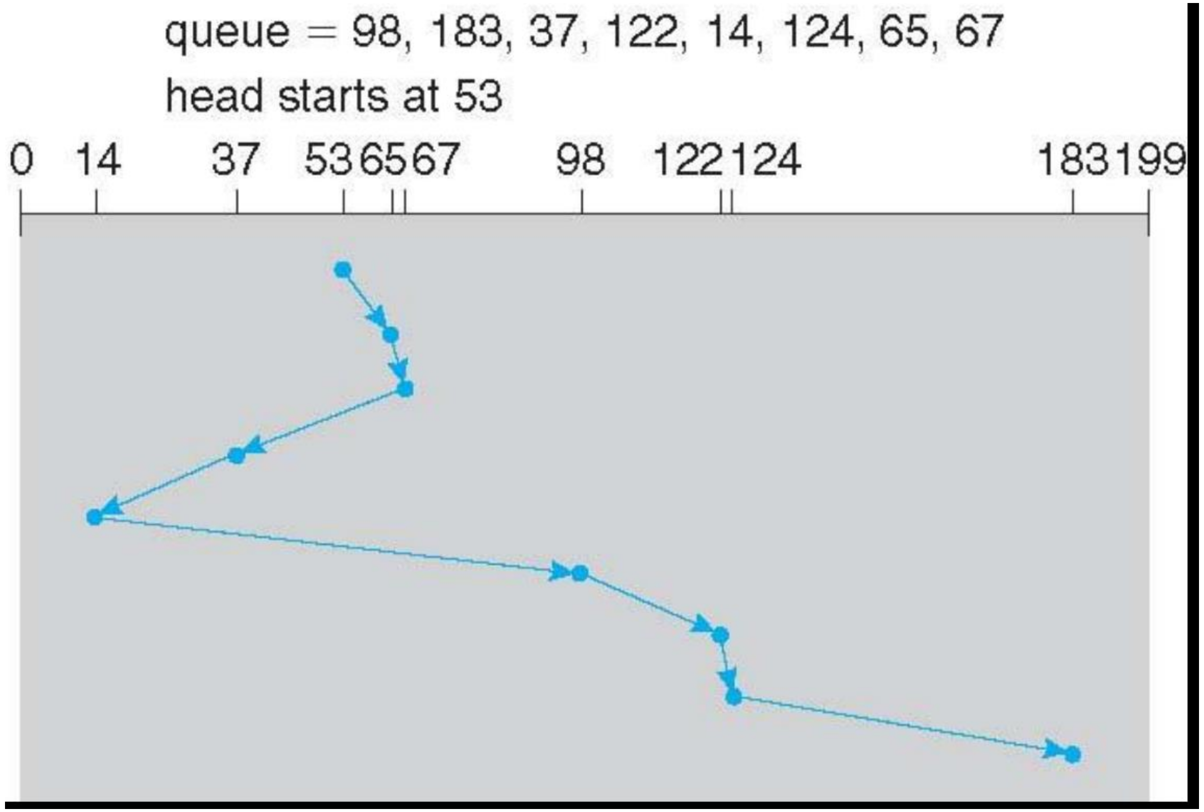


Is there any problem with SPTF?

- **Idea:** Select the request with the least time for seeking and rotating

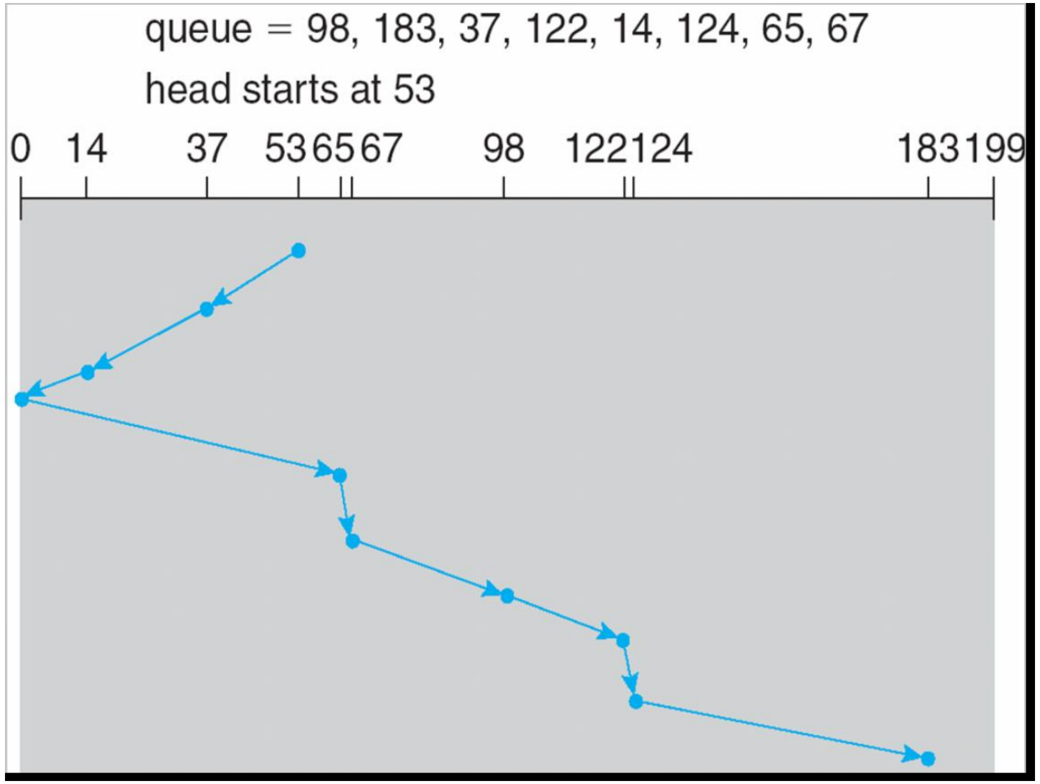
- Illustration shows total head movement of **236** cylinders.

- **Problem:** Like Shortest-Job-First (SJF) scheduling: may cause starvation of some requests!



SCAN

- **Idea:** Sweep back and forth, from one end of disk to the other, serving requests as you go (also called an “elevator” algorithm)
- Illustration shows total head movement of **236** cylinders.

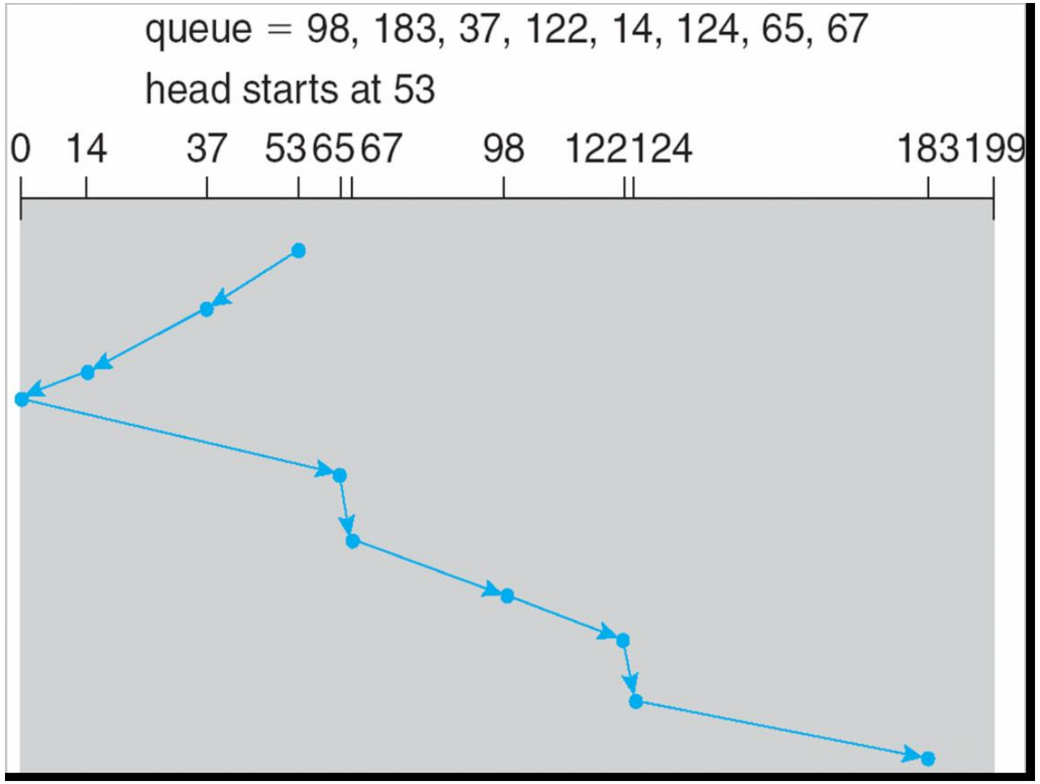


Is there any problem with SCAN?

- **Idea:** Sweep back and forth, from one end of disk to the other, serving requests as you go (also called an "elevator" algorithm)

- Illustration shows total head movement of **236** cylinders.

- **It is a bit unfair!**
  - Cylinders in the middle get better service, while at edges might have to wait for long periods of time.



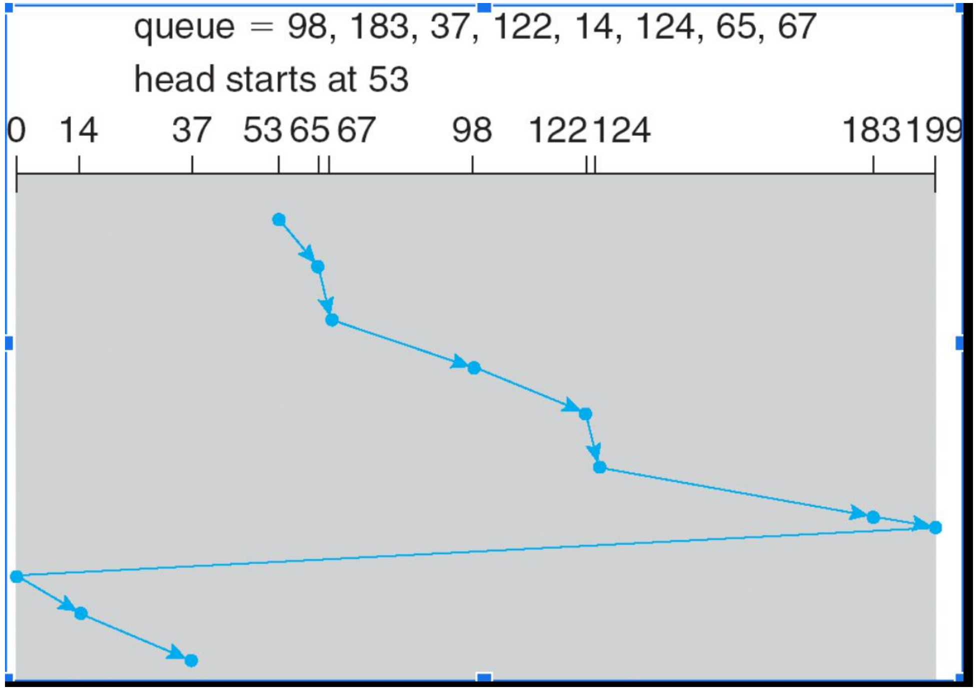
C-SCAN

Idea: Only sweep in **ONE** direction.

- When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip

- Illustration shows total head movement of **382** cylinders.

- A bit slower than SCAN, but fairer to different computations

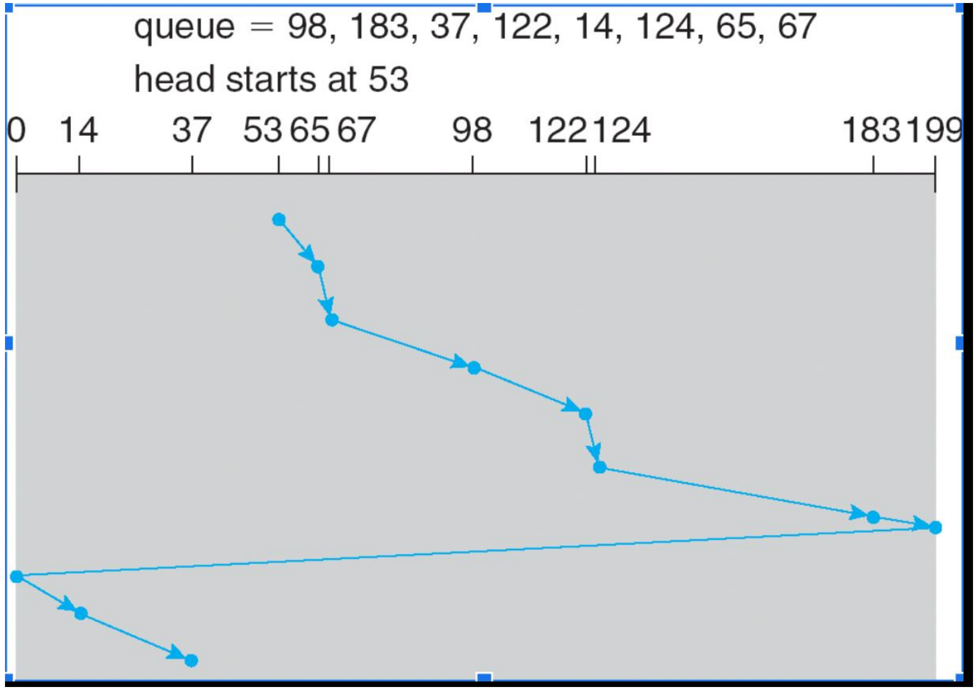


Is there any problem with naive C-SCAN?

Idea: Only sweep in **ONE** direction.

- When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip

- **We are moving to the end of each direction (199 – 0), even though there is no such task**

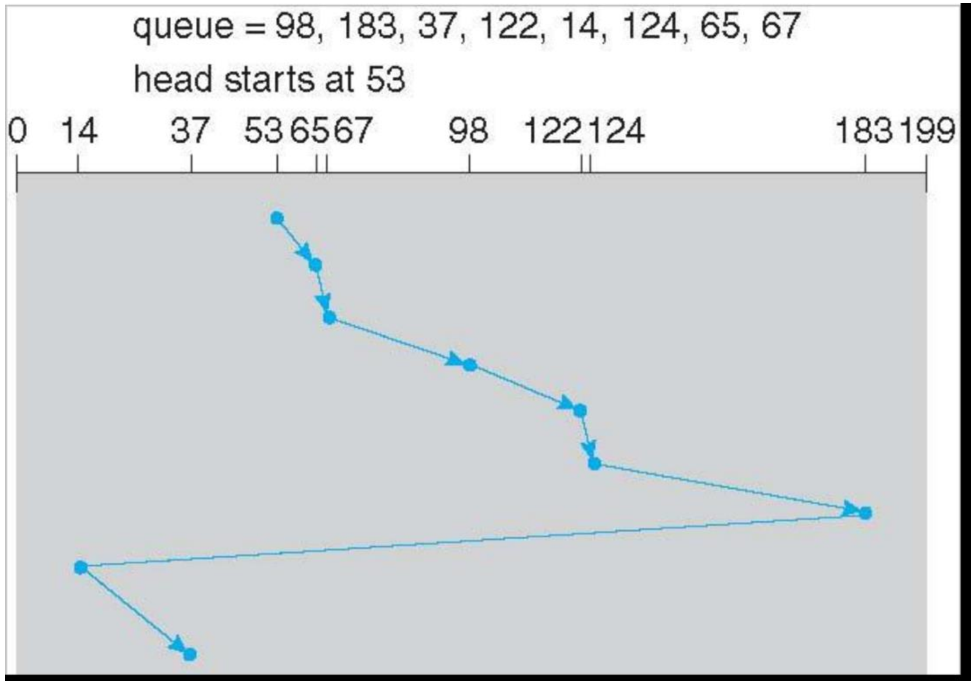


C-LOOK

Idea: Only sweep in **ONE** direction, but also **look ahead**

- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk

- Illustration shows total head movement of **322** cylinders.
  - (65-53)+(67-65)+(98-67)+(122-98)+(124-122)+(183-124)+(183-14)+(37-14) = 322



Work conservation versus anticipation in scheduling

- **Work conserving** scheduling will always try to perform an I/O operation without any delay if
  (a) there's an I/O request waiting to be handled
  (b) the disk is ready to handle an I/O request

- Sometimes, it is better to **wait (delay) instead if you anticipate** another request will appear nearby
  - E.g., consider the current queue: 0, 199
  - After a small time, there might be a request (e.g., 65)

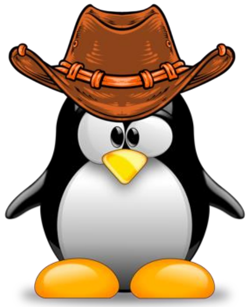
➢ Such non-work-conserving schedulers are called *anticipatory*

Default disk scheduling in Linux OSs: CFQ

- CFQ stands for “Completely Fair Queueing”
- OS keeps a I/O request queue for each process
- Do weighted round-robin among each process queue, with slice time proportional to process priority
- Optimize order within queue (e.g., using C-SCAN)
- Typically, disk scheduling in Linux is **anticipatory**

Summarizing different HDD scheduling

- SPTF has a natural appeal towards minimizing average movements
  - ➢ **Starvation is a real problem**
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - ➢ **Better fairness since all I/O requests will be served after fixed time**
- Performance depends on the workload (i.e., number and types of requests)
- Requests for disk service can be impacted by the file-allocation method/pattern and metadata layout – **topic of file systems**



Questions? Otherwise, see you next class!