CS 330 - Operating Systems

I/O Devices

16-10-2025

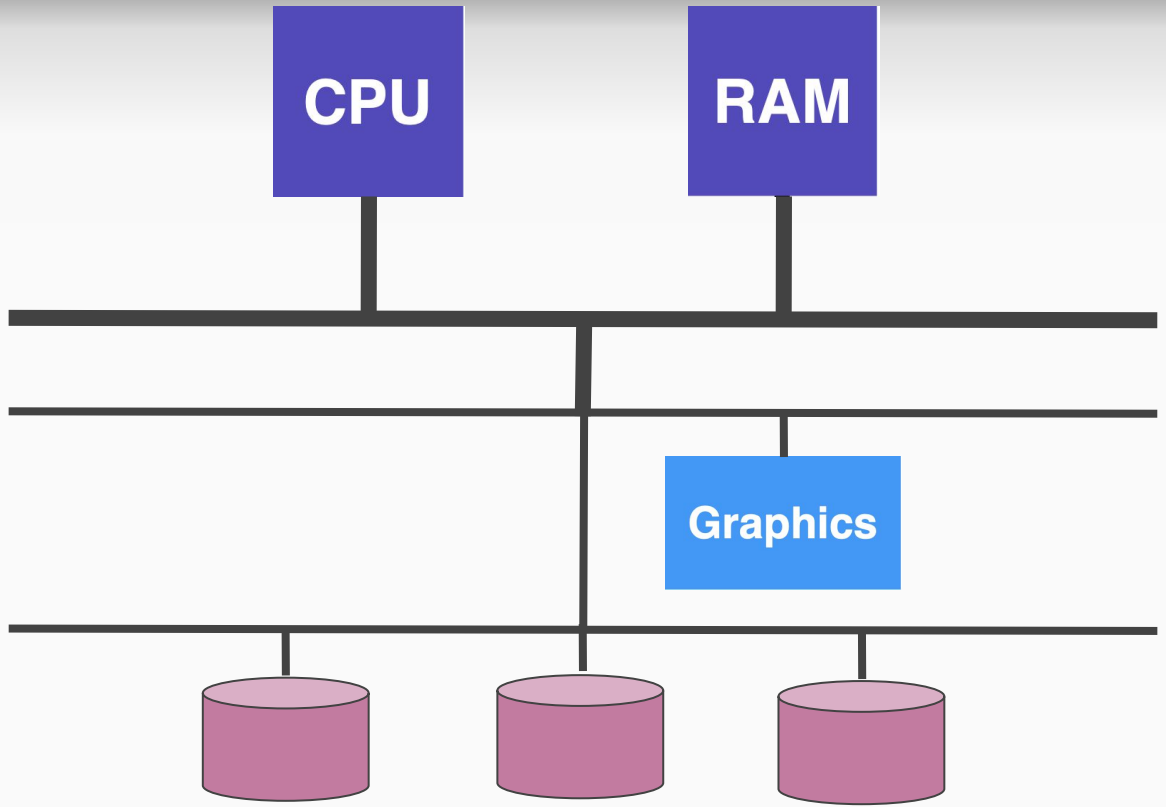
Abhishek Bichhawat

I/O in Systems

- Computer without I/O devices?
  - Not interesting without a range of inputs and some outputs
- Systems need
  - H/W that will let us plug in different devices
  - OS that can interact with different combinations

- How should I/O be integrated in systems?

System Architecture

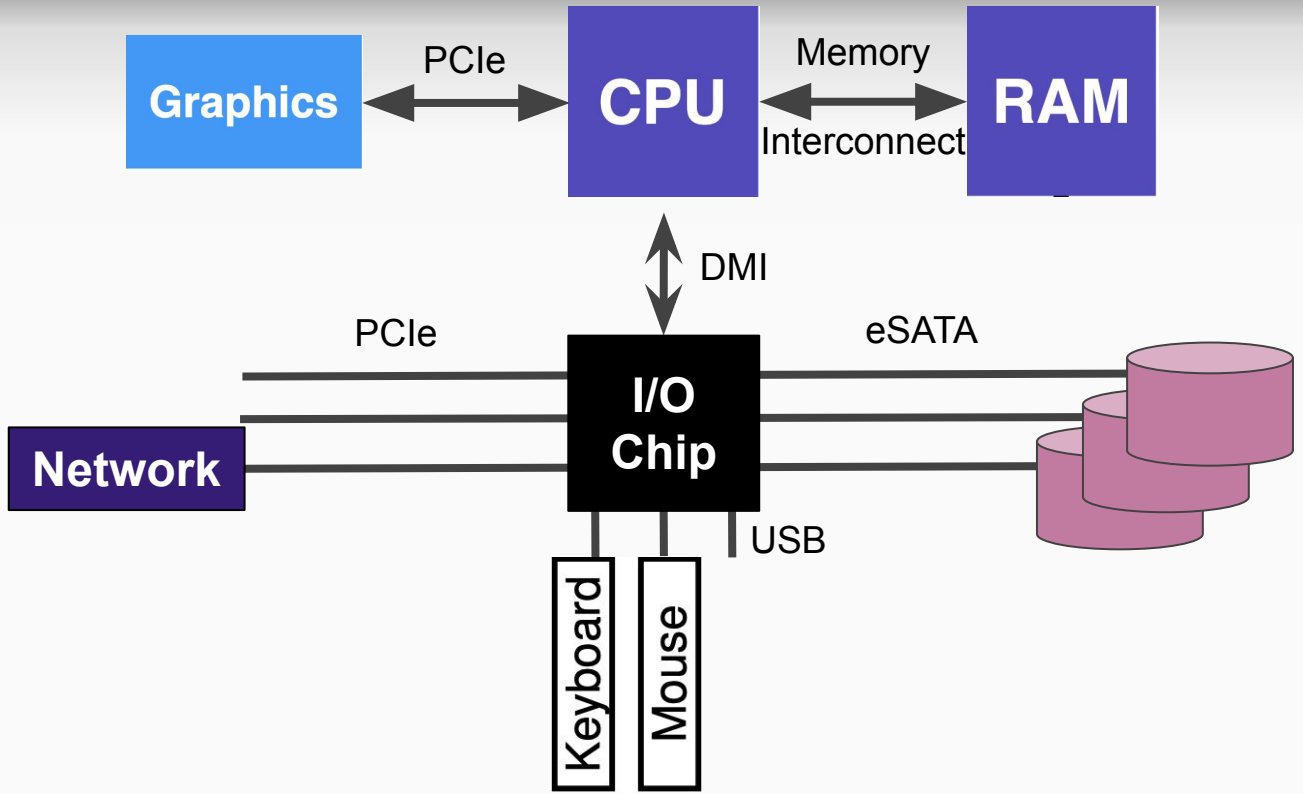


Memory Bus

General I/O Bus
(e.g., PCI)

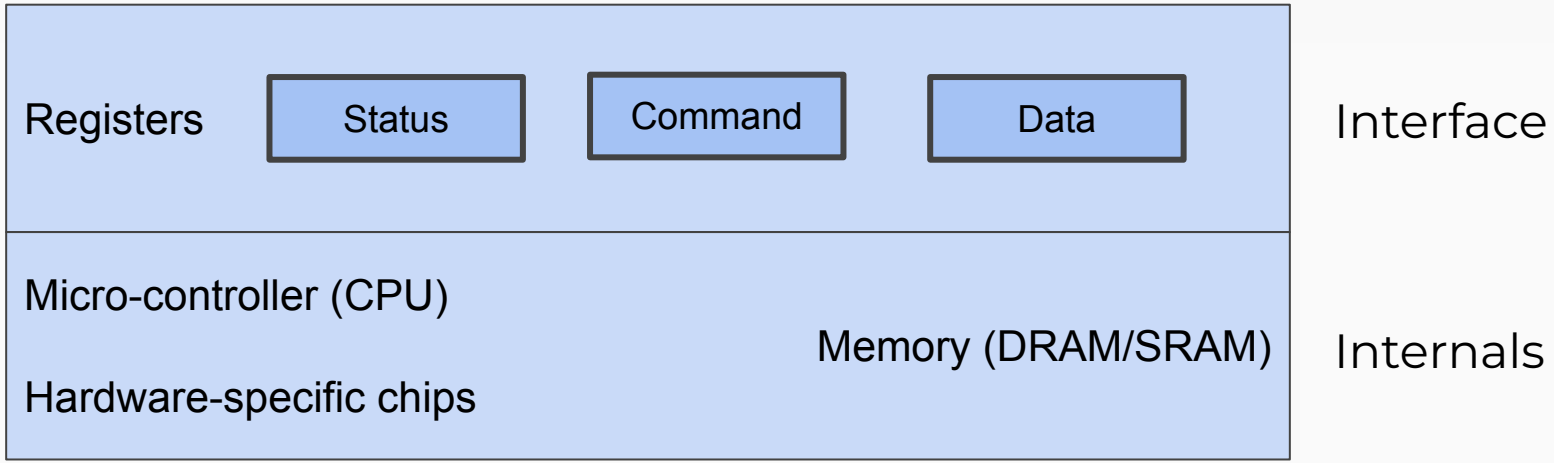
Peripheral I/O Bus
(e.g., SCSI, SATA, USB)

Modern Systems

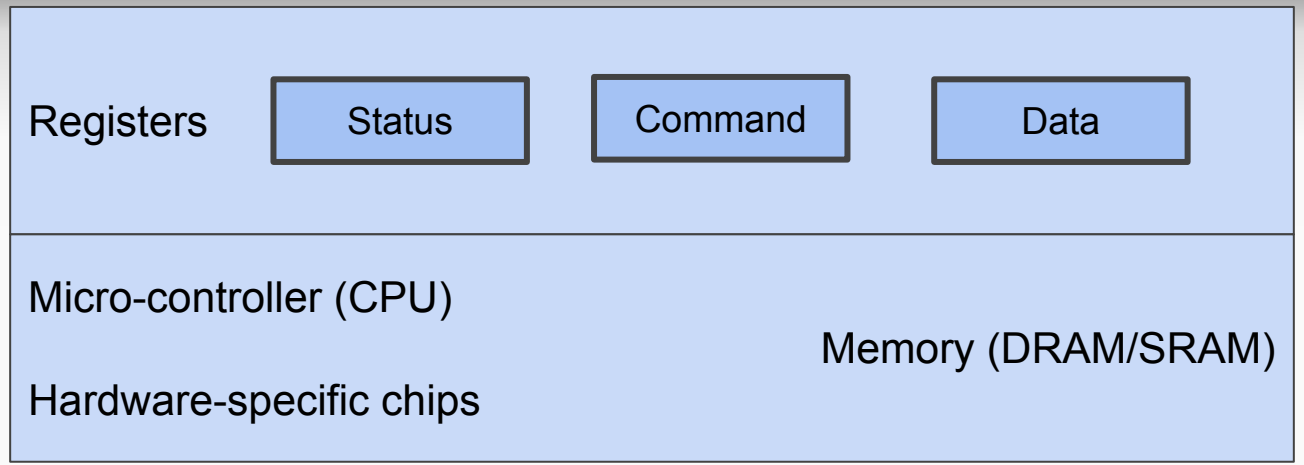


Device Interactions with OS

Device Components



Device Interactions with OS

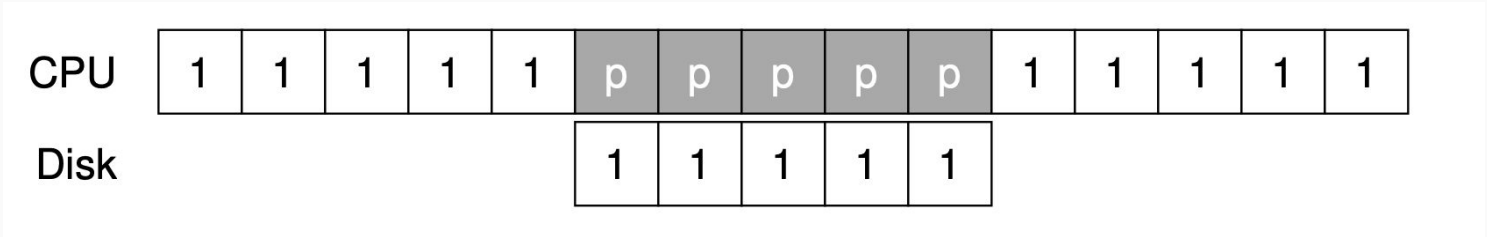


Polling

```
While (STATUS == BUSY)
   ; // wait until device is not busy
Write data to DATA register
Write command to COMMAND register
   (starts the device and executes the command)
While (STATUS == BUSY)
   ; // wait until device is done with your request
```

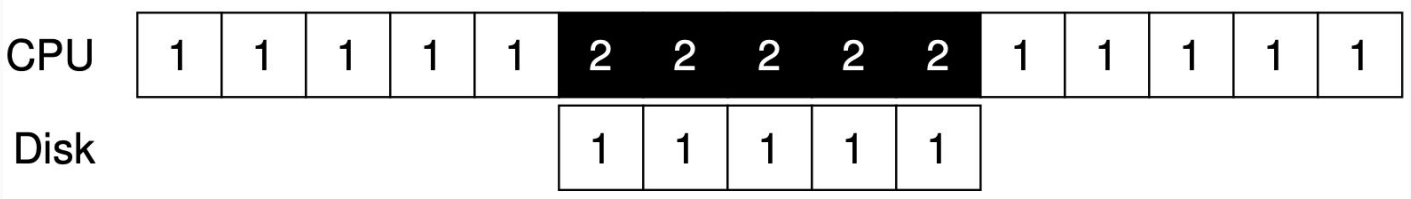
Device Interactions with OS

- Protocol before is simple but inefficient
  - Polling wastes CPU cycles



Device Interactions with OS

- Protocol before is simple but inefficient
  - Polling wastes CPU cycles
- Use **interrupts** instead
  - Issue request
  - Put calling process to sleep
  - Context switch
  - Perform device operation
  - Raise hardware interrupt
  - Finish the request (wake calling process waiting for I/O)



Interrupts vs. Polling

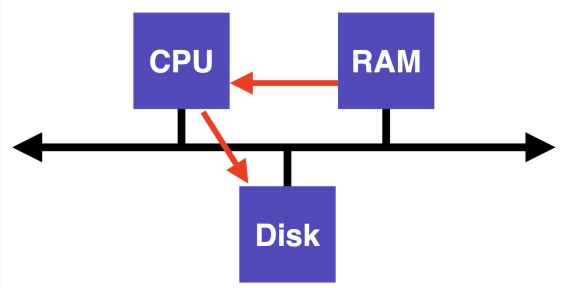
- Polling is good for fast devices
- Interrupts are not helpful with networks
  - **Livelock**
    - only process interrupts w/o running user-level processes

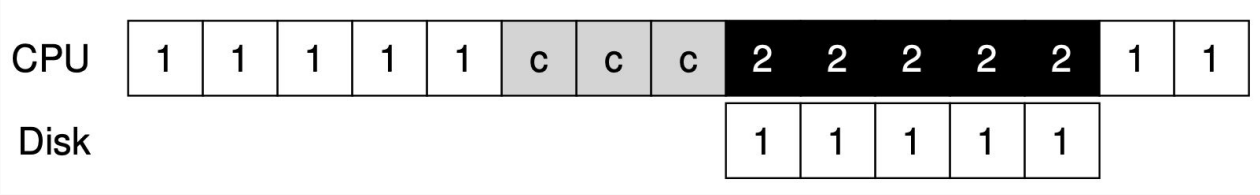
- Interrupts are good for slow devices allowing overlaps
- Can use **coalescing** for optimizing
  - Wait before delivering interrupt (group interrupts together)

- Two-phased approach
  - Start with polling
  - Switch to interrupt after a fixed time

DMA

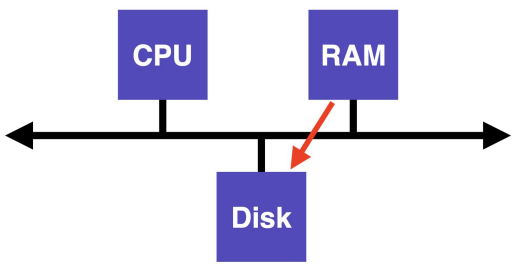
- Large data transfer (Programmed I/O)
  - CPU spends time transferring data to device

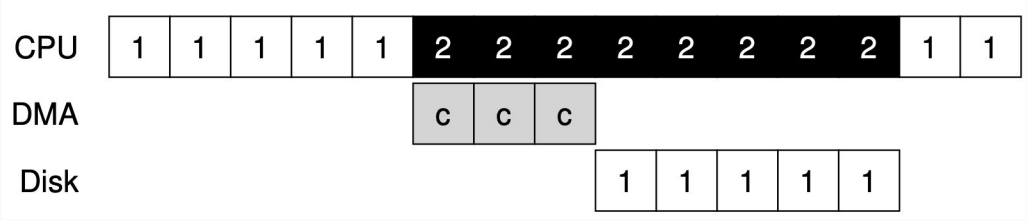




DMA

- Large data transfer (Programmed I/O)
  - CPU spends time transferring data to device
- Direct Memory Access (DMA)
  - Device to transfer data directly between other devices & memory
  - OS provides details of data transfer
  - Thereafter, DMA engine takes care of transfer
  - Raises interrupt when done





Device interactions with OS

- How does OS communicate with the devices?
- Explicit I/O instructions
  - each device has a port
  - **in**/**out** instructions (x86) communicate with device
  - Privileged instructions

Device interactions with OS

- Explicit I/O instructions
  - each device has a port
  - **in**/**out** instructions (x86) communicate with device
  - Privileged instructions

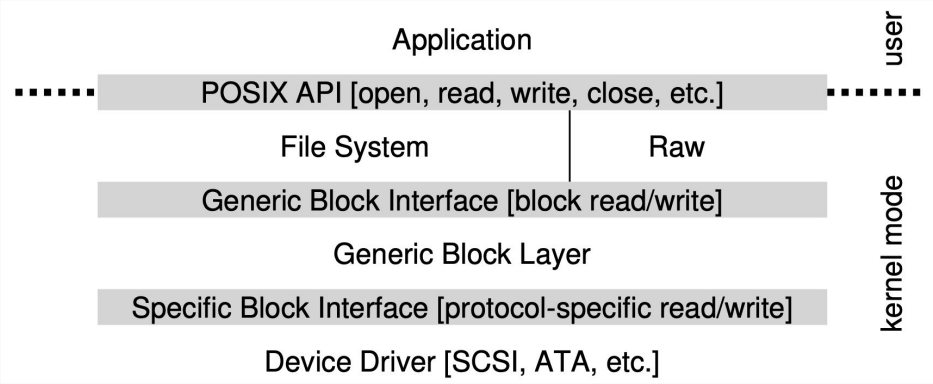
- Memory-mapped I/O
  - H/W maps registers into address space
  - loads/stores sent to device

Device drivers

- Different devices have different interfaces
  - Many devices with different features and specific interfaces
  - Different variants
    - polling vs. interrupts
    - PIO vs. DMA
    - special instructions vs. memory-mapped I/O
- Should work with general OS!
  - File system
    - Should work with SCSI disks, USB drives etc.

Device drivers

- Abstraction
  - Module within OS knows how device works
  - Known as **device driver** (encapsulates device interaction)
- > 70% of Linux kernel code is device drivers
  - Primary contributors of kernel crashes!

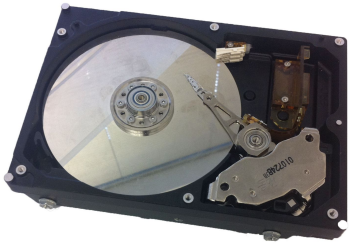


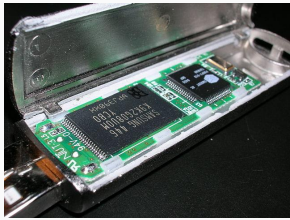
Storage Devices - Hard disk drives

- File systems are abstraction on top of persistent storage
  - Magnetic disk & Flash memory (e.g., USB thumb drive)
  - File systems persist data even if there is a crash
  - Provide names, and control access
- Persistent storage devices provide
  - Storage that (usually) survives across machine crashes
  - Block level (random) access
  - Large capacity at low cost
  - Relatively slow performance

Storage Devices - Hard disk drives

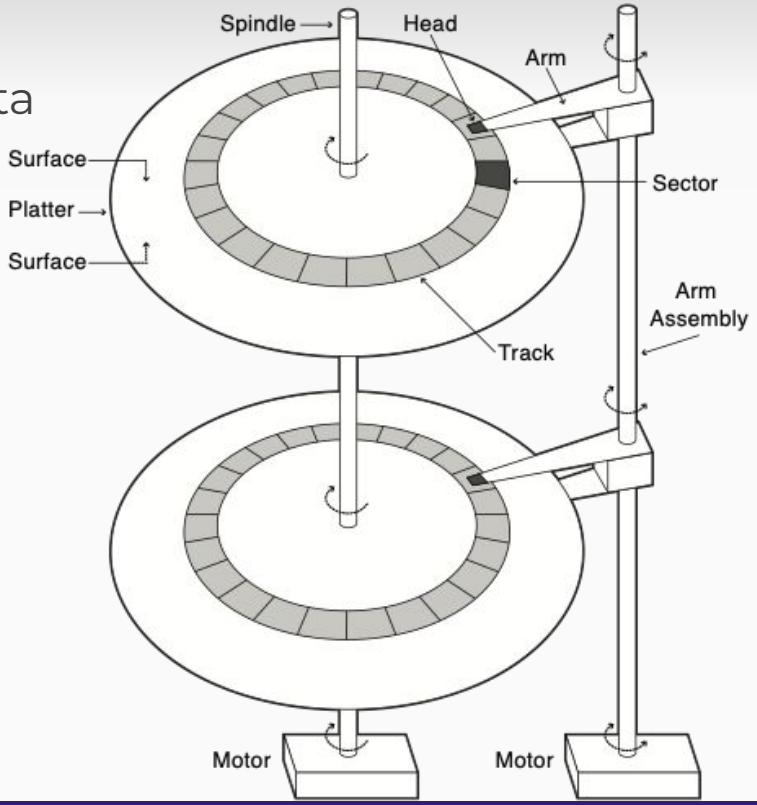
- Magnetic disks
  - Storage that rarely becomes corrupted
  - Large capacity at low cost
  - Block level random access
  - Slow performance for random access
  - Better performance for streaming access
- Flash memory
  - Storage that rarely becomes corrupted
  - Capacity at intermediate cost (50x disk)
  - Block level random access
  - Good performance for reads; worse for random writes





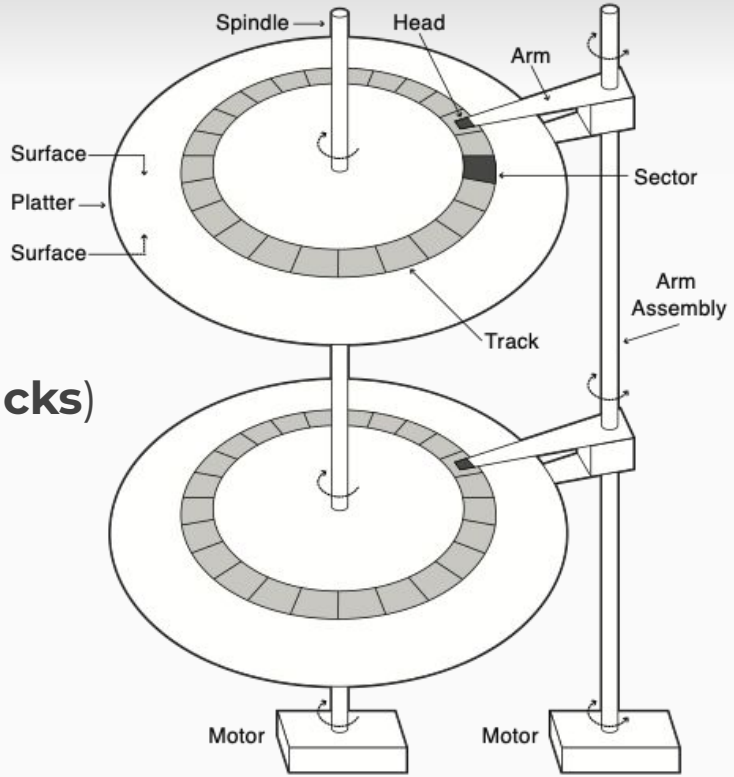
Storage Devices - Hard disk drives

- Platter
  - 2-sided circular surface that stores data
  - Coated with thin magnetic layer that enables persistent storage
  - Multiple platters are bound together by spindle
- Spindle
  - Platters spin around this
  - 7200 - 15000 RPM



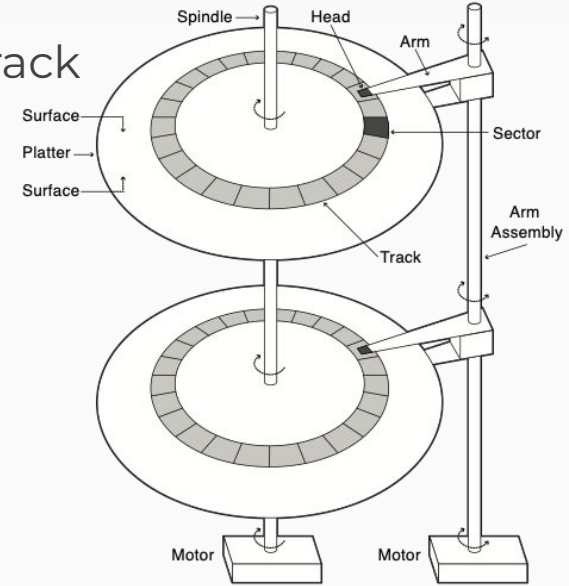
Storage Devices - Hard disk drives

- Sector
  - 512-byte blocks
  - Disk is array of sectors (blocks)
  - File systems can read multi-sectors
  - Update to single sector is **atomic**
  - Accessing nearby blocks is faster
  - Multiple concentric circles (called **tracks**)
  - Tracks are ~ 1 micron wide



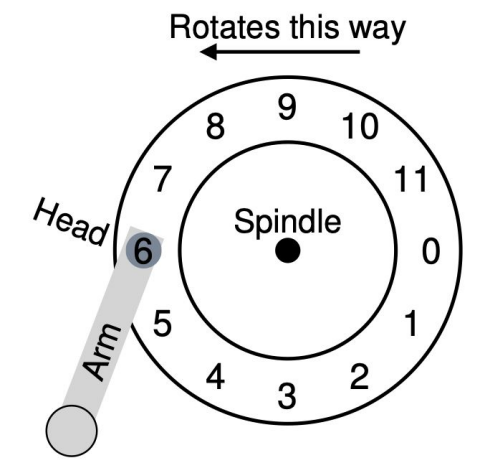
Storage Devices - Hard disk drives

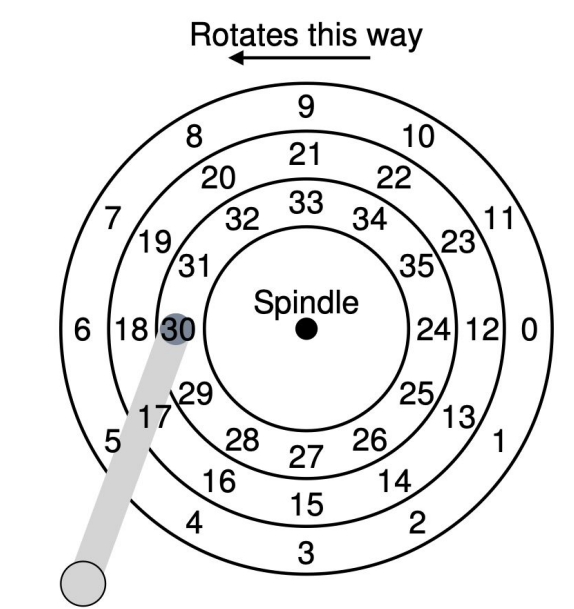
- Disk Head
  - Read and write from surface
  - One head per surface
  - Attached to **arm** that moves to desired track



Reading from Disk

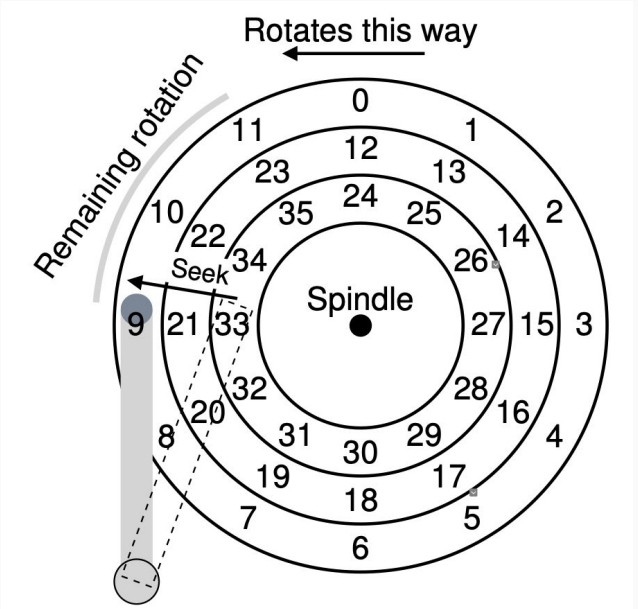
- Rotational Delay
  - Time taken for desired sector to rotate under head





Reading from Disk

- Seek
  - Time taken for desired sector to rotate under head
  - Expensive
  - Phases
    - Acceleration
    - Coasting
    - Deceleration
  - Settling time
    - Time taken to find right track
- Transfer



Writing to Disk

- All steps before and ...
- Acknowledge the write either when
  - the data is put in memory (write-back caching)
  - the data is written to disk (write-through)

Time taken for I/O

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

$$R_{I/O} = Size_{transfer} / T_{I/O}$$

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Time taken for I/O

- Random read of 4KB

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Time taken for I/O

- Random read of 4KB
- On Cheetah
  - Average seek time - 4 ms
  - Rotational delay - 2 ms
  - Transfer time - 30 us

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Time taken for I/O

- Random read of 4KB
- On Barracuda

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Time taken for I/O

- Sequential workload
- Single seek and rotation with long transfer of 100 MB
  - I/O time is 800 ms and 950 ms

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

Disk Scheduling

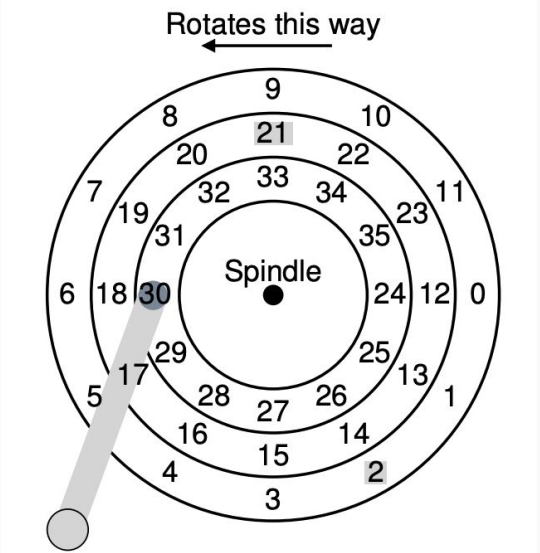
- Cost of I/O is high
- OS decides how the I/O happens
  - From a set of I/O requests, the OS decides which to schedule next
  - Can make a fair guess about the length of an I/O job
- Disk can do only one request at a time
- What order to choose to do queued requests?

FIFO Scheduling

- First request to arrive is serviced first
- Fair among requesters
- Order of arrival may be to random spots on the disk
- Very long seeks

Shortest Seek Time First

- Estimate seek time and rotational delay
- Schedule the job with least seek time first
  - Pick the request that's closest on the disk
    - Order the requests by tracks
    - Requests on the nearest track are picked first
  - Although called SSTF, includes rotational delay in calculation, since rotation can be as long as seek



How does an elevator work?

- Suppose you are going down from floor 10 to 0, and somebody got on at 3 and pressed 4, and the elevator went up to 4 because it was “closer” than 1!
  At 4 someone else got in and pressed 6...

How does an elevator work?

- Suppose you are going down from floor 10 to 0, and somebody got on at 3 and pressed 4, and the elevator went up to 4 because it was “closer” than 1!
- The **elevator algorithm** in real life prevents fights from taking place on elevators in such scenarios.
- If the elevator is going down, it completes the down trip before starting to go back up.

Shortest Seek Time First

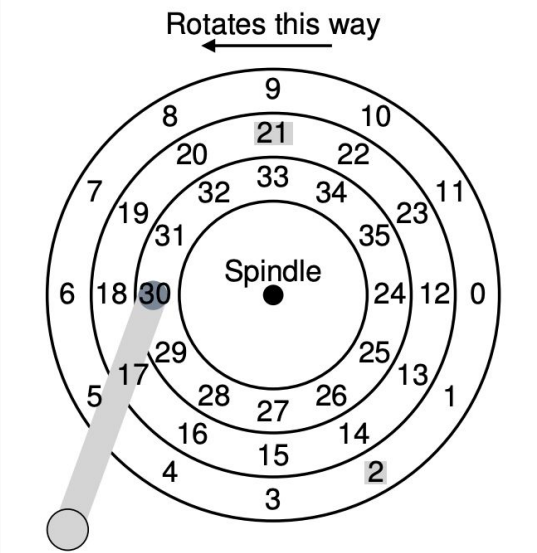
- Estimate seek time and rotational delay
- Schedule the job with least seek time first
  - Pick the request that's closest on the disk
  - Although called SSTF, includes rotational delay in calculation, since rotation can be as long as seek
- SSTF good at reducing seeks, but may lead to starvation
- Geometry of the drive may not be available
  - use block addresses to determine nearest block

SCAN

- Algorithm moves back and forth across the disk servicing requests in order across the tracks.
  - Just like an elevator!
- One pass from outer to inner tracks is called a **sweep**
  - Requests for blocks already serviced on the sweep are queued
  - Handled in the next sweep
  - Take the closest request in the direction of travel
  - No starvation, but retains flavor of SSTF

C-SCAN

- Circular, less-biassed version of SCAN
- SCAN services the layers on the sweep back making the requests at outer layer that were missed to be delayed
- Instead service only in one direction
- When sweeping back to the original position, do not service any requests

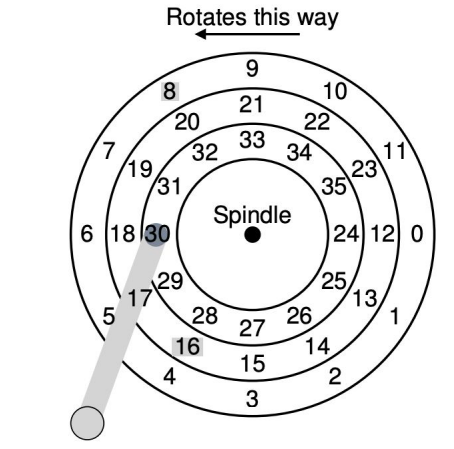


F-SCAN

- Freezes the queue to be serviced when doing a sweep
- Places requests that come in during the sweep into another queue to be serviced later.
- Avoids starvation of far-away requests
  - Delays the servicing of late-arriving (but nearer) requests

Optimal Algorithm

- In the example below, we have requests to serve block 16 and 8. Which one do we serve first?



Optimal Algorithm - SPTF

- In the example below, we have requests to serve block 16 and 8. Which one do we serve first?
- If seek time is much more than rotational delay, then serve 16 followed by 8
- If seeking is faster or equal to rotation, then serve 8 while the disk rotates so that 16 comes closer than it is!
- Shortest Positioning Time First