Stable Storage and File Systems

Bringing It Together

- Processes are the OS's main abstraction for protection
- A process defines an address space---but this is a virtual/logical address space
- Virtual address spaces are mapped onto physical address spaces in physical memory
- *Paging* describes how the OS provides the illusion of infinite memory
  - Page tables aid address translation and data finding
  - Page faults identify which pages are needed and not yet in memory
  - Page replacement algorithms decide which pages to evict when a free frame is needed
  - Or we can create free frames ahead of time using the working set
- The heap is part of a process's virtual address space (and is thus paged!)
  - Heap can be managed explicitly---user directs the heap memory manager
    - Allocation policies: free list and bump pointer
    - Deallocation: only for free list!
  - Heap may be managed automatically---user still allocates memory, but the heap manager automatically manages it
    - Same allocation policies
    - Deallocation: garbage collection!
      - Identification and Collection
- *All of what we have discussed so far uses main memory (RAM)… except what?*

External Devices

Including Disks

External Devices

What types of external devices do you have on your computer now?

- Keyboard/mouse
- Hard drive
- Network card
- Monitor
- Microphone
- Printer
- and many more….

What purpose do these devices serve?

Overview

Programmer

<AMI/API>

Operating System

<HAL>

Device Controller

<lowest-level interface>

Device

Application Programmer's View

The OS provides a high-level interface to devices, greatly simplifying the programmer's job.

- Standard interfaces are provided for related devices
  - if you write to a file from a user program, you don't need to know if that file is on a hard drive or a USB stick
- Device intricacies are encapsulated in device drivers
- New devices can be supported by an OS simply by providing a device driver with the device

I/O Services Provided by the OS

- Operations appropriate to the devices
- Device allocation and release
- Device drivers that implement device-specific behaviors
- I/O scheduling
- Access control (protection)
- Buffering, caching, and spooling to allow efficient communication
- (Uniform) naming of files and devices
  - On Unix/Linux, devices appear as files at the highest level of abstraction. /dev contains file names, most of which actually represent devices
- Error handling and failure recovery associated with devices

Devices

Characteristics:

- Transfer unit: character or block
- Access method: sequential or random
- Timing: synchronous or asynchronous

  (Note: most devices are asynchronous, but I/O system calls are synchronous. The OS implements blocking *I/O.)*
- Shared or dedicated
- Speed
- Operations: input, output, or both

Architecture of I/O Systems

The hardware associated with an I/O device consists of four pieces:

- A *bus* that allows the device to communicate with the CPU
  - typically shared by multiple devices
- A device *port* typically consisting of 4 registers
  - *Status*: whether the device is busy, whether data is ready, whether an error occurred
  - *Control*: command to perform
  - *Data-in*: data being sent from the device to the CPU
  - *Data-out*: data being sent from the CPU to the device
- A *controller* that can receive commands from the system bus, translate commands into device actions, and read data from and write data to the system bus
- The device itself

The Disk

A Very Important External Device

Disks

- Just like memory, only different
- Memory is volatile, disks lasts forever(ish)
- Memory is small (order of GB), disks are large (order of 100s-1000s of GB)
  - Used for short term storage of memory contents (swap space)
  - Reduces what must be kept in memory (e.g. code pages)
- Memory is expensive, disks are cheap
  - Memory = .27GB/$1, Disks = 50.2GB/$1
- A type of *stable storage*

How Magnetic Disks Work



- Store data magnetically on thin metallic film bonded to a rotating disk of glass, ceramic, or aluminum
- The disk surface is circular
- The disk is always spinning

Anatomy of a Magnetic Disk

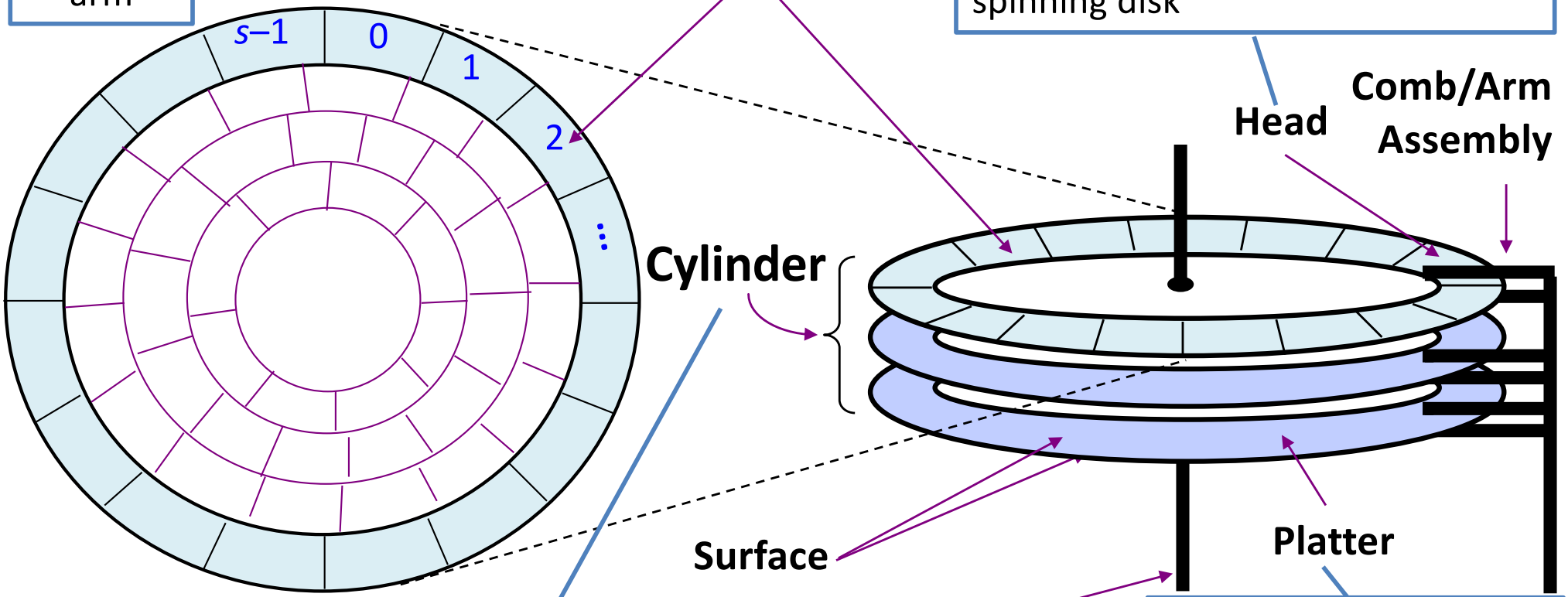
data on a track can be read without moving arm

Typically 512 bytes

Reads by sensing a magnetic field

Writes by creating one

Floats on air cushion created by spinning disk



Set of tracks on different surfaces with same track index

4,200-15,000 RPM

Thin cylinder that holds magnetic material

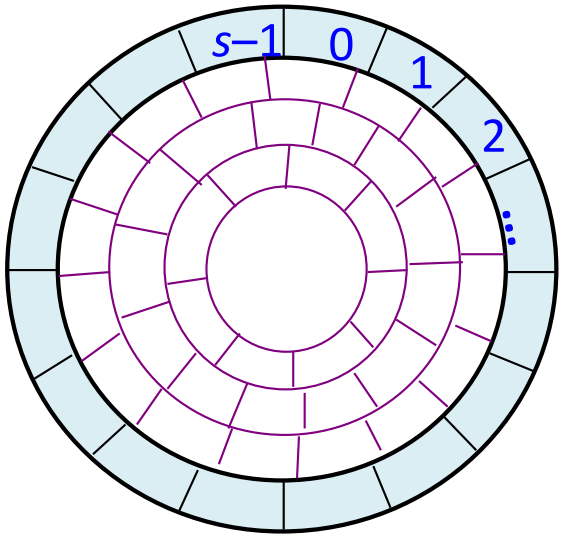
Each platter has two surfaces

Anatomy of a Magnetic Disk: Text Description

- Note: this explanation will start at the smallest part and increase by size
- Sector: a contiguous collection of bytes on disk, typically 512 bytes in size
- Track: a contiguous collection of sectors
  - Data can be read from a track without having to move the arm
- Surface: a concentric collection of tracks, is circular in shape
  - Tracks are circular in shape and radiate out from the center of the surface
- Platter: thin disk covered in magnetic material
  - Each platter typically has 2 surfaces, one on top and the other on bottom
- Cylinder: set of tracks on different surfaces with same track index
  - So line up all the way down the whole disk---creating a hollow cylinder
- Spindle: runs through the center of the disk, creates axis around which the platters spin
  - Speed: 4,200 - 15,000 RPM
- Head: used to actually read/write the data off the disk
  - Reads by sensing a magnetic field
  - Writes by creating a magnetic field
- Arm: extends head over the surface
  - Each surface has its own head and arm
- Comb/arm assembly: Spine to which each arm is attached
  - Looks similar to a comb
  - Moves arms in and our over disk
  - Controls all arms at once and in unison (arms cannot be moved independently)
- Recap:
  - A platter has 2 surfaces, one on top and the other on bottom
  - Surfaces are made up of tracks
  - Tracks are made up of sectors
    - Concentric tracks across platters are called a cylinder

An Aside: Sectors and Blocks

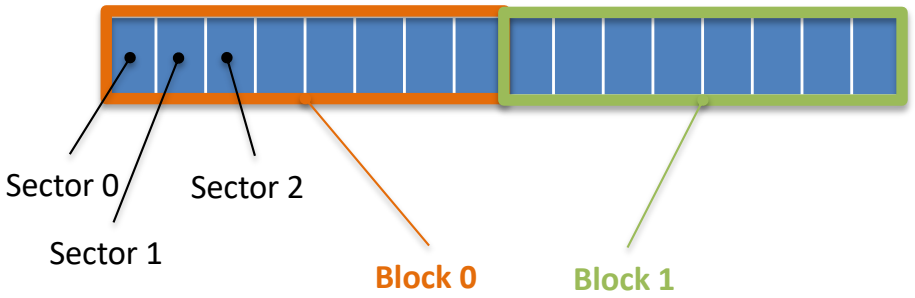
Disks: Sectors



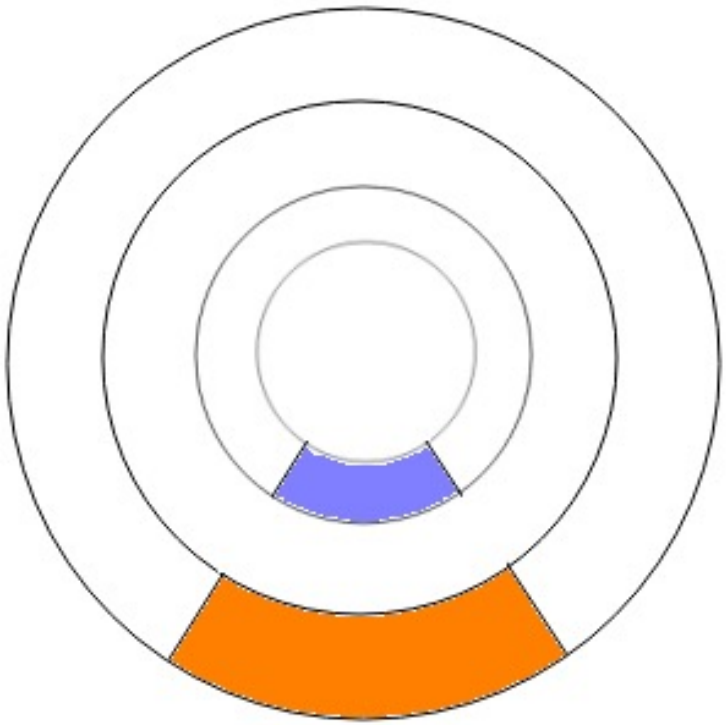
Sector: Smallest unit that the drive can read or write, usually 512 bytes large.

File Systems: Blocks

Blocks are the smallest unit which the software (usually) accesses the disk at, which is an integer number of contiguous sectors.



Note: Outer parts of the disk are faster!



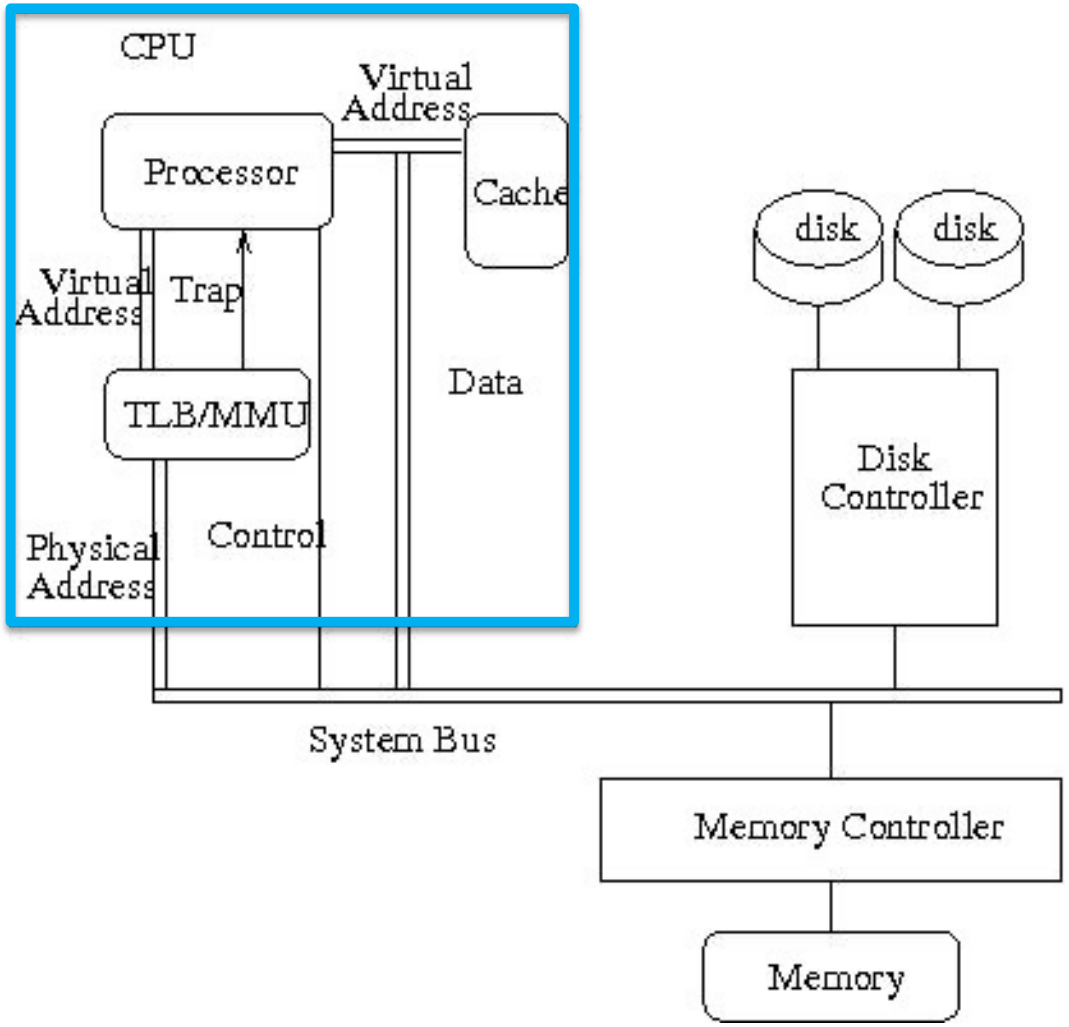
Why? Because *more disk* travels past the read head per unit time closer to the edge of the platter!

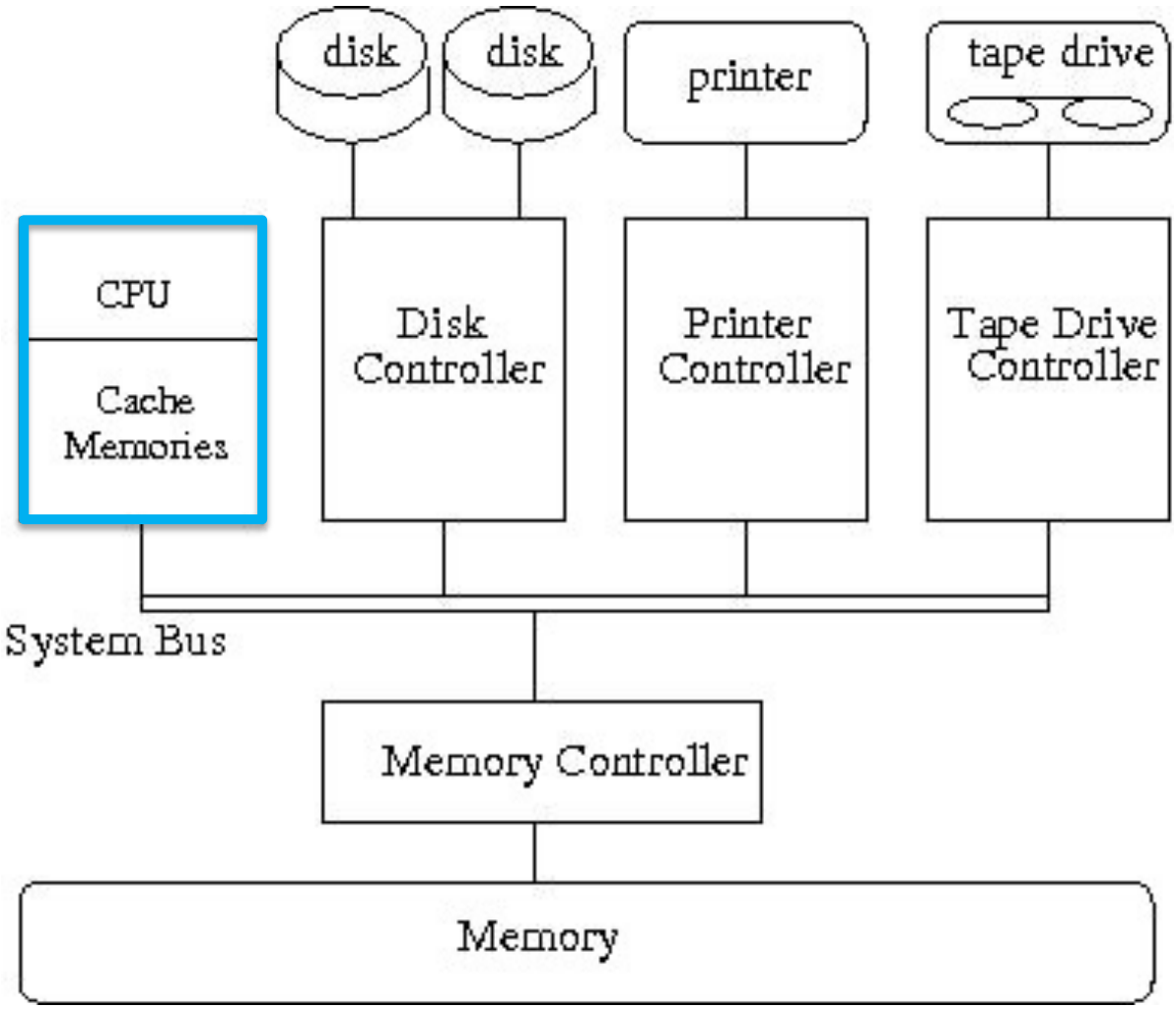
Please pretend the tracks are the same width. Drawing is hard.

But where is the disk?

Disks and the Storage Hierarchy

Back to Architecture




disk
disk
printer
tape drive
CPU
Cache Memories
Disk Controller
Printer Controller
Tape Drive Controller
System Bus
Memory Controller
Memory

Hardware Picture: Text Description

- System components communicate through the system bus
- Hanging off the system bus are:
  - CPU, cache, memories (one component)
  - Memory controller, which then is attached to memory
  - Disk controller, which then is attached to one or more disks
  - Printer controller, which then is attached to a printer
  - Tape drive controller, which then is attached to a tape drive
- Note that this is an example architecture. Real systems may have different components, and some systems have both a system and a memory bus.

Okay, but…

What does all this really mean as part of the system?

Computer Memory Hierarchy

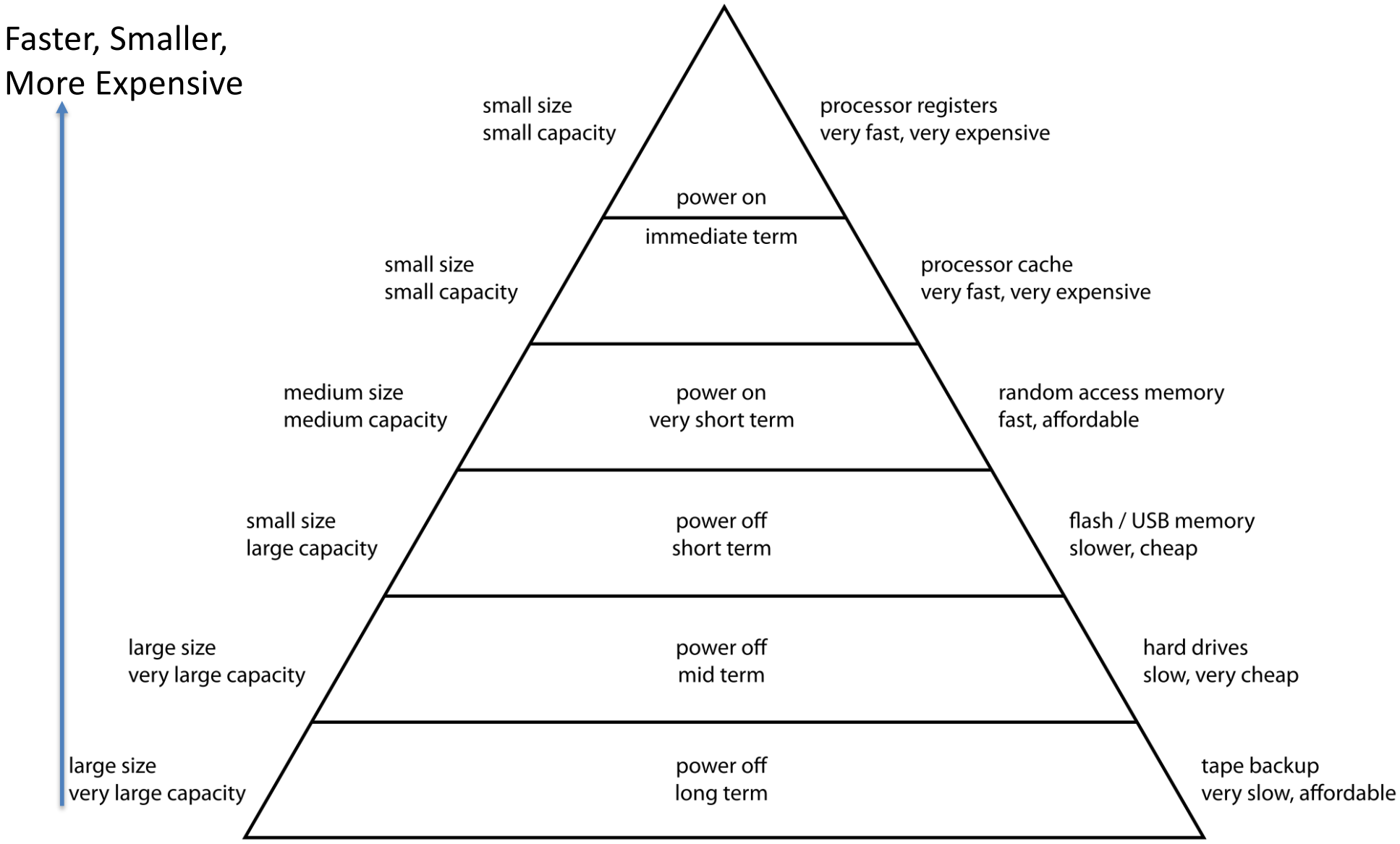


Image Credits: Danlash @ en.wikipedia.org

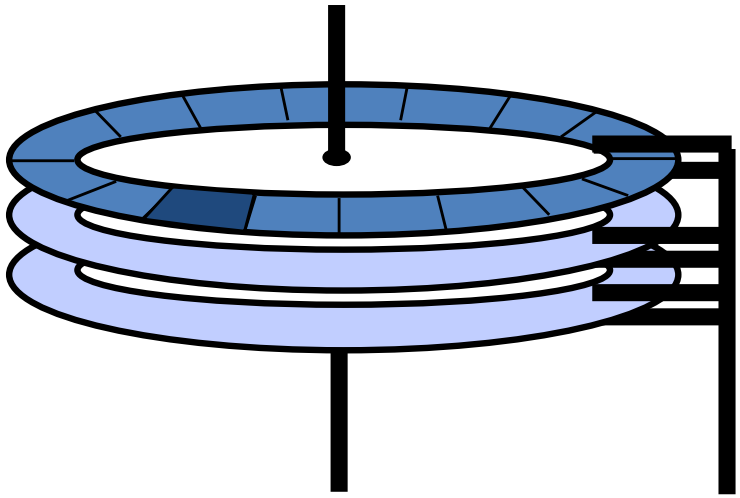
Okay, but...

Why is it slow?

Disk Operations and Performance

Disk Operations

- Present disk with a sector address
  - *Old: DA* = (*drive*, *surface*, *track*, *sector*)
  - New: Logical block address (linear addressing) converted by disk controller
- Heads moved to appropriate track
  - Seek time, settle time
- Appropriate head is enabled
- Wait for sector to appear under head
  - rotation time
- Read/write sector as it spins by
  - transfer time



Read time:
seek time + rotation time + transfer time

A Closer Look: Seek, Rotation, and Transfer Time

- *Seek time*: time to position head over track/cylinder
  - Maximum: time to go from innermost to outermost track
  - Average: average across seeks between each possible pair of tracks
    - approximately time to seek 1/3 of the way across the disk
  - Head Switch Time: time to move from a track on one surface to the same track on a different surface
    - range similar to minimum seek time
- *Rotation time*: time for the sector to rotate underneath the head
  - Today most disks rotate at 4,200 to 15,000 RPM
  - good estimate is half that amount
- *Transfer time*: time to move the bytes from disk to memory
  - *Surface transfer time*: time to transfer one or more sequential sectors to/from surface after head reads/writes first sector
    - **much smaller** than seek time or rotation time
    - higher bandwidth for outer tracks than inner ones---same RPM but more space
  - *Host transfer time*: time to transfer data between host memory and disk buffer

A Closer Look: Transfer Time

- *Transfer time*: time to move the bytes from disk to memory
  - two pieces: surface and host
- *Surface transfer time*: time to transfer one or more sequential sectors to/from surface after head reads/writes first sector
  - **much smaller** than seek time or rotation time
    - 512 bytes at 100MB/s = 5microseconds (0.005ms)
  - higher bandwidth for outer tracks than inner ones
    - same RPM but more space
- *Host transfer time*: time to transfer data between host memory and disk buffer
  - 60MB/s (USB) to 2.5GB/s (Fibre Channel 20GFC)

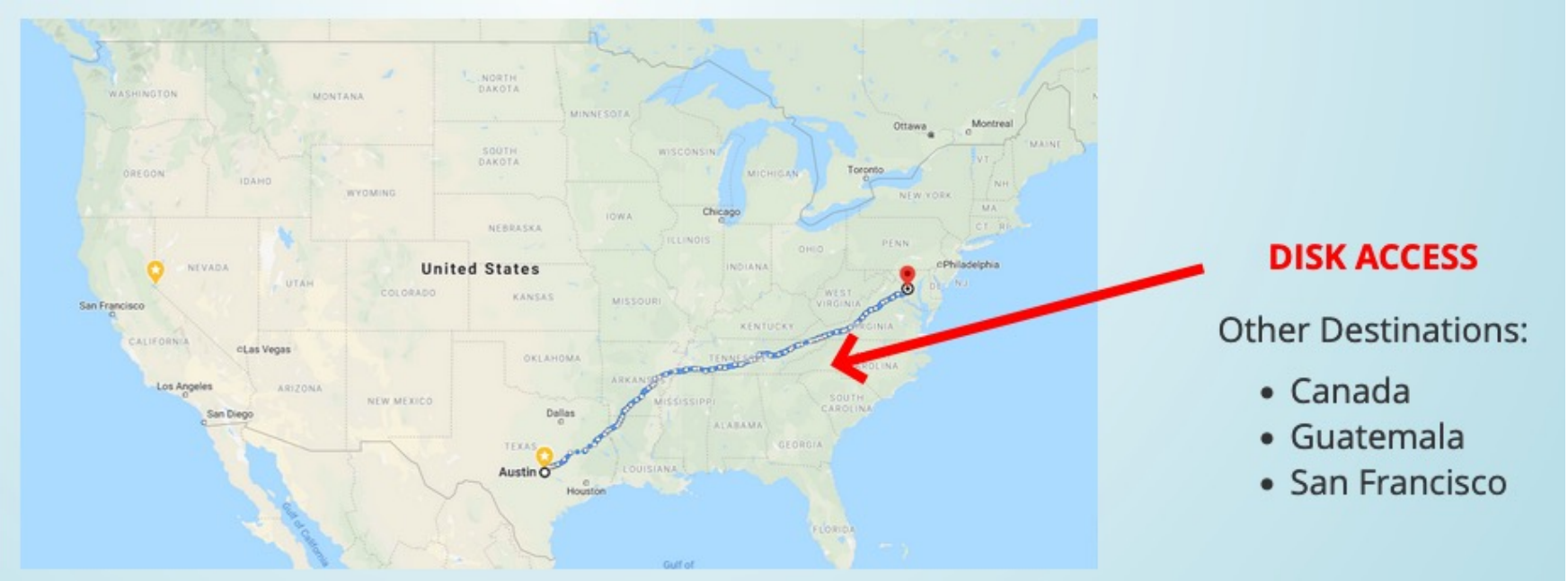
Okay, but…

How slow?

Speed

For a relatively fast hard drive, the average data access has a latency (request made to data available) of **12.0 ms**.

Action	Latency	1ns = 1s	Conversation Action
L1 Cache Hit	1ns	1s	Talking Normally
L2 Cache Hit	4ns	4s	Long Pause
Cache Miss*	100ns	1m 40s	Go upstairs to ask Dr. Norman
Disk Seek	12ms	138d 21hr+	Walk to Washington D.C. and back



Okay, but…

So… a few questions:

1) How does the OS communicate with these devices, which need input and output data, but are much slower than the CPU? Does it do something to hide latency?
2) Does the I/O subsystem to anything to hide latency?
3) What about the file system? Does it anything to hide latency?

Communicating with I/O Devices

Communication

- The OS typically communicates with I/O devices through their controller
- OS places information in the device's registers for the controller to send to the device
- Controller places information from the device in the registers for the OS

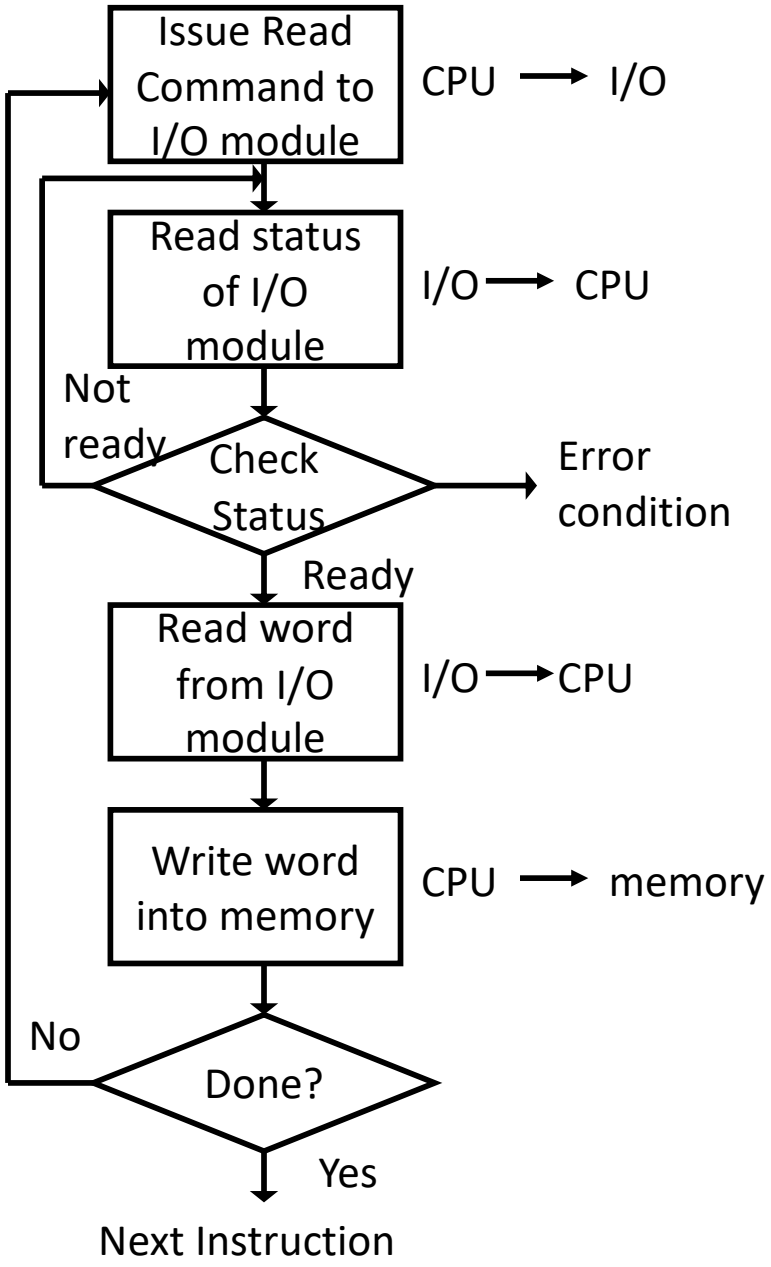
Three ways for this communication to take place…

Method One: Polling

- OS keeps checking until the status of the I/O device is idle
  - How often the OS checks is known as the *polling rate*
- OS sets the command register and, if it is an output operation, places value in data-out
- OS sets the status to command-ready
- Controller reacts to command-ready and sets the status to busy
- Controller reads the command register and performs the command and, if it is an input operation, places a value in data-in
- Assuming the operation succeeded, the controller changes the status to idle
  - Otherwise the status is set to error
- OS observes the change to idle and, if it were an input operation, reads the data

Also known as *Programmed I/O*.

Example Polling: Input of a Block of Data



Polling Example: Input of a Block of Data Text Description

- Flow of Control:
  - Issue read command to I/O module
  - Read status of I/O module until the status is ready
    - During read check for error conditions
    - If status is not ready must keep performing status read at speed set by polling rate
  - Once I/O module is ready, read word from the I/O module
  - Write the word into memory
  - If you are done go on to next instruction
    - If not done must go back to “issue read command” step
- Note: reading each word individually

Question

In polling, what happens if the device is busy when the OS has a request?

A. The OS moves the operation to a pending list, and it moves to other tasks until it receives an interrupt from the device
B. The OS busy waits
C. The OS moves to another task, but it keeps the operation on its todo list and tries back

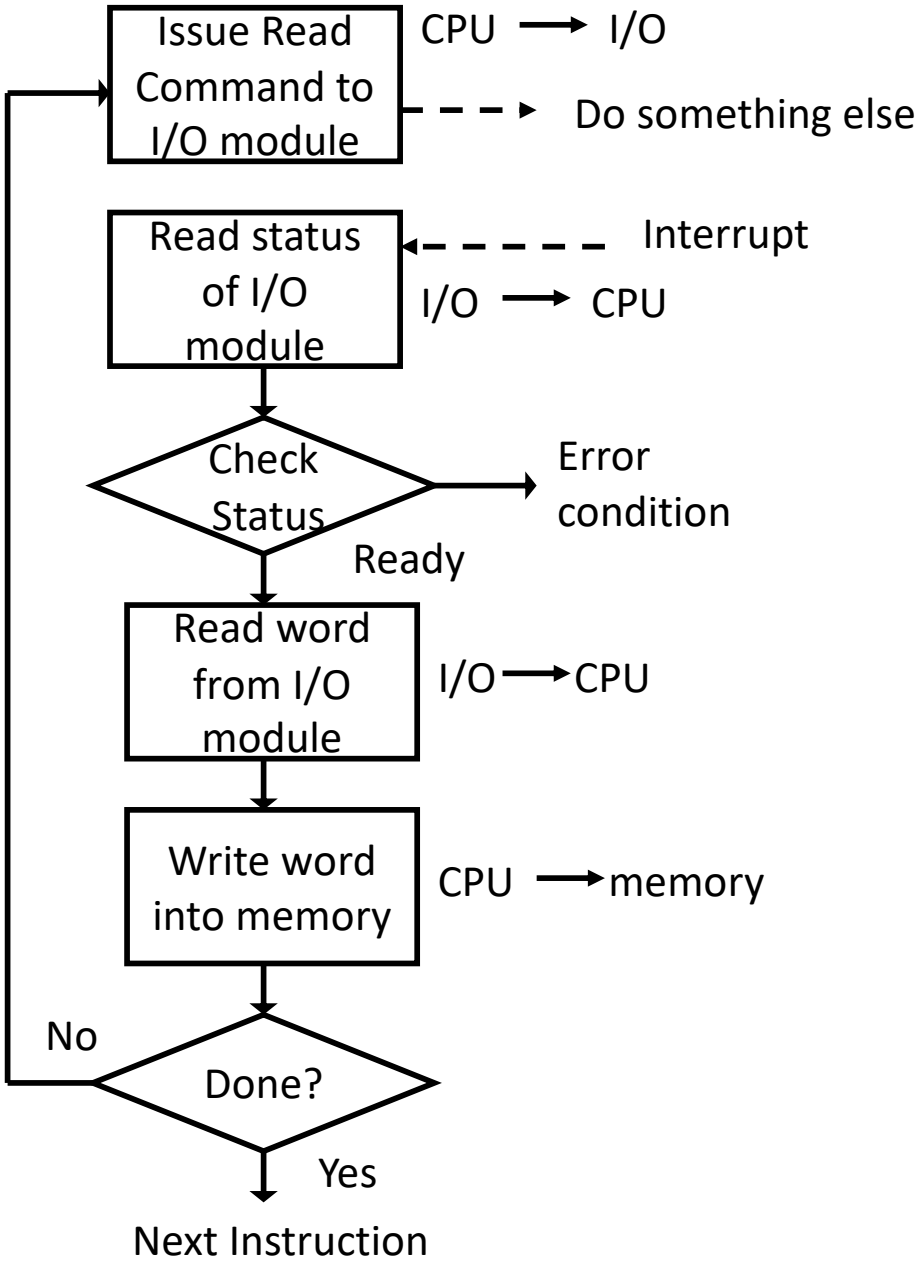
Method Two: Interrupts

Rather than have the CPU continually check if a device is available, the device can interrupt the CPU when it completes an I/O operation.

On an I/O interrupt:

- Determine which device caused the interrupt
- If the last command was an input operation, retrieve the data from the device register
- Start the next operation for that device

Interrupts Example: Input of a Block of Data



Interrupts Example: Input of a Block of Data Text Description

- Flow of Control:
  - Issue read command to I/O module
    - Now can go off and do something else until interrupted
  - Interrupt is issued - indicates that status should be ready
    - Read status of I/O module
    - If status is not ready, check for error conditions
    - If ready, move on
  - Read word from I/O module
  - Write word into memory
  - Check if you are done, if you are can go on to next instruction
    - if not done must go back to “issue read command” step
- Note: still only ready a word at a time

Method Three: Direct Memory Access

- The device uses a more sophisticated controller, a Direct Memory Access (DMA) controller, that can write directly to memory
  - No data-in/data-out registers---instead it has an address register
- The CPU tells the DMA controller the location of the source and the destination of the transfer
- The DMA controller operates the bus and interrupts the CPU when the **entire transfer** is complete, instead of when each word is ready
- The DMA controller and the CPU compete for the memory bus, slowing down the CPU somewhat, but this method still provides better performance than if the CPU had to do the transfer itself

Diagram: DMA Transfer

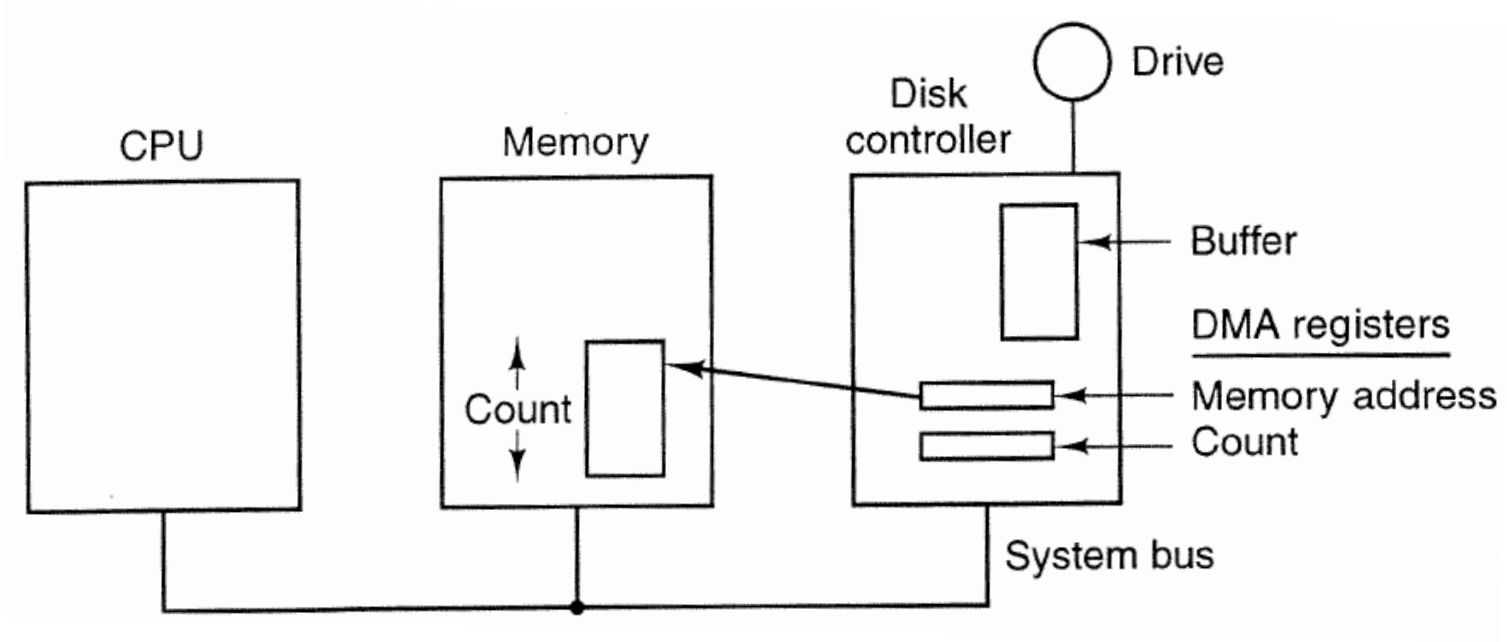
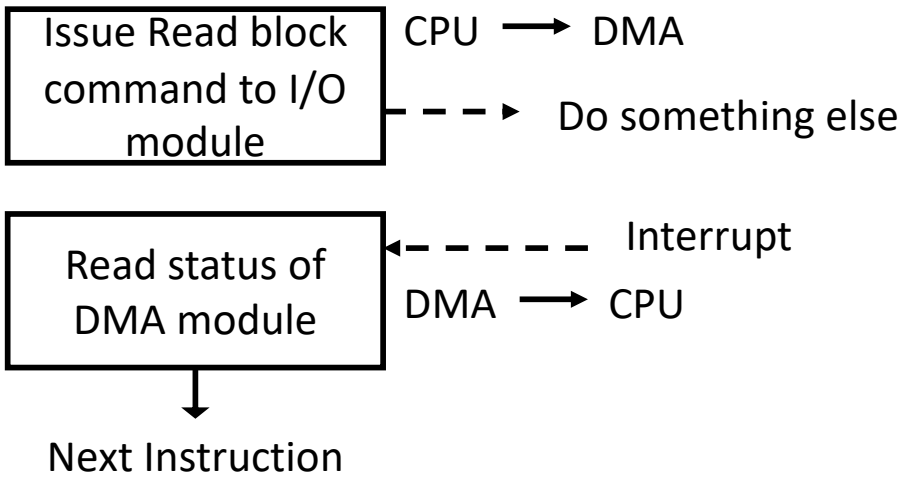


Diagram: DMA Transfer

- CPU, Memory, and Disk Controller all connected by the system bus
- Disk controller has components within it:
  - A buffer
  - DMA registers
    - Keep track of memory address and count
    - This information is referencing data in memory
  - Disk Controller connects to the drive

DMA Example: Input of a Block of Data



DMA Example: Input of a Block of Data Text Description

- Issue read block command to I/O module
  - Now CPU can do something else
- Interrupt is issued by DMA controller--- indicates the information is ready
- Read status of DMA module
- Go on to the next instruction (assuming read completed correctly)
  - No need to check to see if you're done because you read the whole block of memory at once instead of just a byte at a time

Question

Which method of device/OS communication is best for a file read?

A. Polling
B. Interrupt
C. DMA

Okay, but…

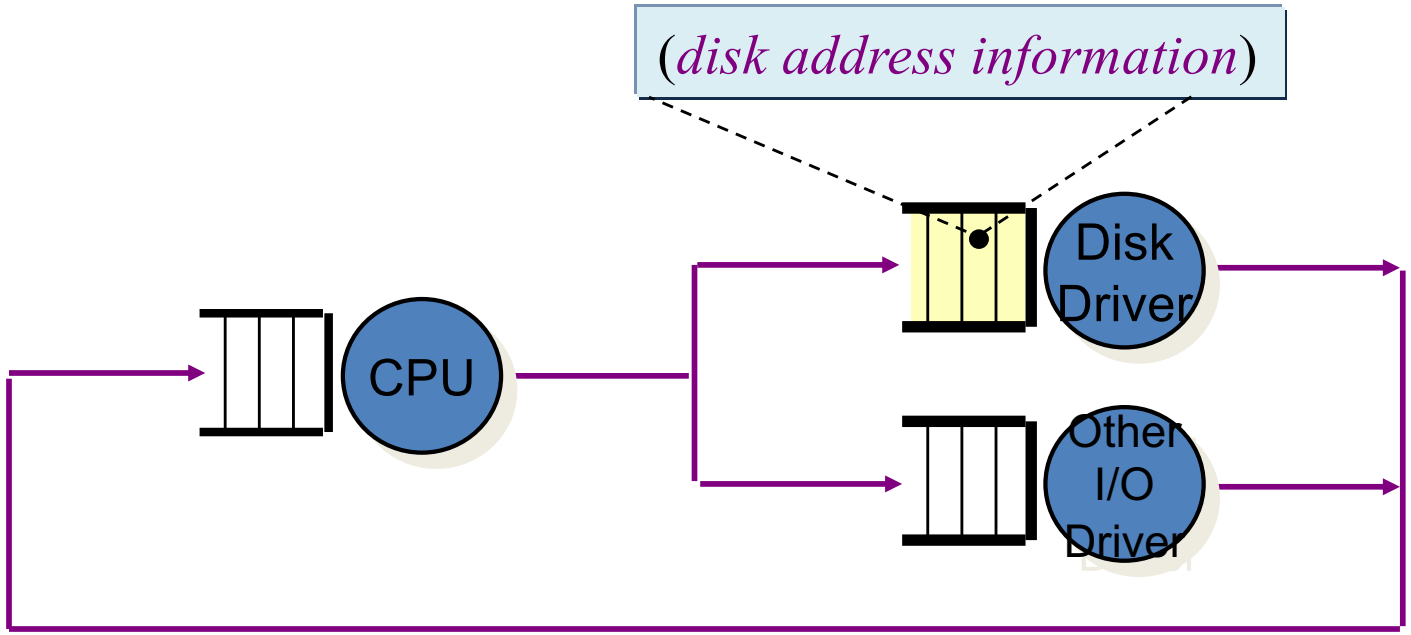
So… a few questions:

1) How does the OS communicate with these devices, which need input and output data, but are much slower than the CPU? Does it do something to hide latency?
2) Does the I/O subsystem to anything to hide latency?
3) What about the file system? Does it anything to hide latency?

Disk Head Scheduling

Disk Head Scheduling

In a multiprogramming/timesharing environment, a queue of disk I/O requests can form



Disk I/O throughput is improved by minimizing head movement through *disk head scheduling.*

Disk Head Scheduling: Plain Text

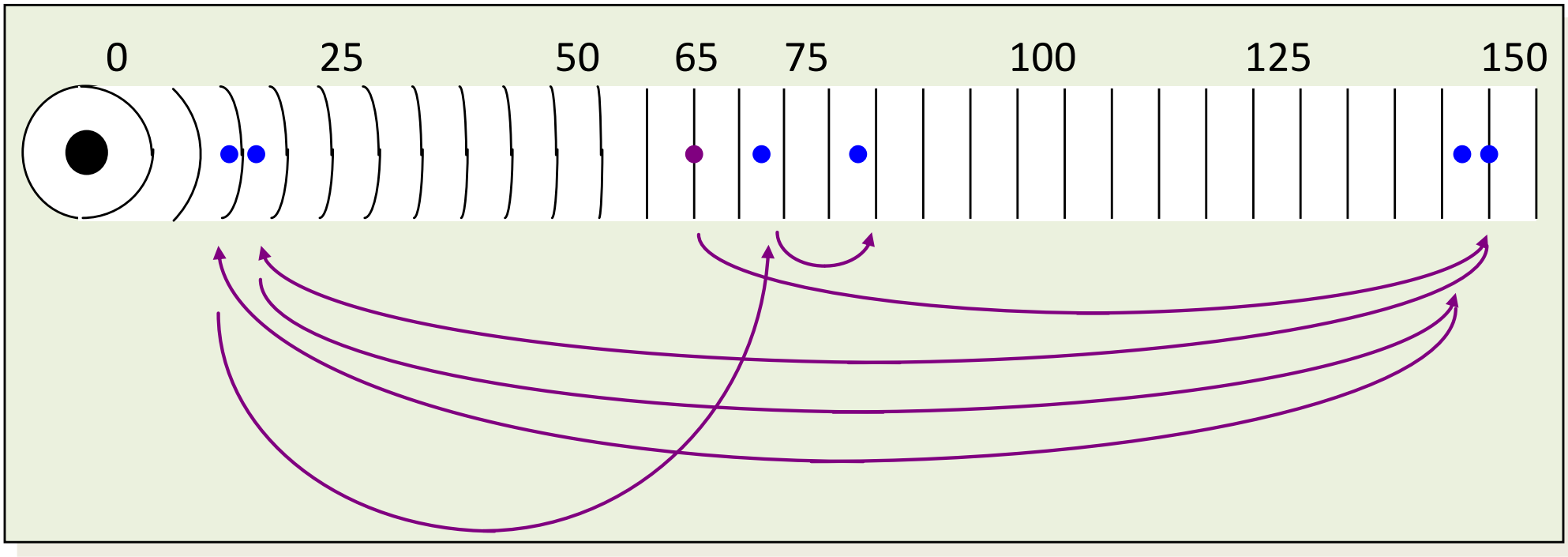
- In a multiprogramming/time sharing environment, a queue of disk I/O requests can form
  - Requests are generated by running program and then handled by the disk driver, where the queue forms
  - After I/O, information goes back to the CPU
- The OS maximizes disk I/O throughput by minimizing head movement through disk head scheduling

Disk Head Scheduling: FCFS/FIFO

Assume a queue of requests exists to read/write tracks:



and the head is on track 65



FCFS/FIFO scheduling results in the head moving 550 tracks

Can we do better?

Disk Head Scheduling: FCFS/FIFO Text Description

- FCFS – First Come, First Served
- FIFO – First In, First Out
- Assume a queue of requests exists to read/write tracks
  - Head is on track 65 to start
  - Request order: 150, 16, 147, 14, 72 and 83
- Order of track moves
  - Move from track 65 to track 150, means moving 85 tracks
  - Move from track 150 to track 16, means moving 134 tracks
  - Move from track 16 to track 147, means moving 131 tracks
  - Move from track 147 to track 14, means moving 133 tracks
  - Move from track 14 to track 72, means moving 58 tracks
  - Move from track 72 to track 83, means moving 11 tracks
- Total number of track moves: 85 + 134 + 131 + 133 + 58 + 11 = 552 tracks
- This seems inefficient, we can do better

Disk Head Scheduling: SSTF

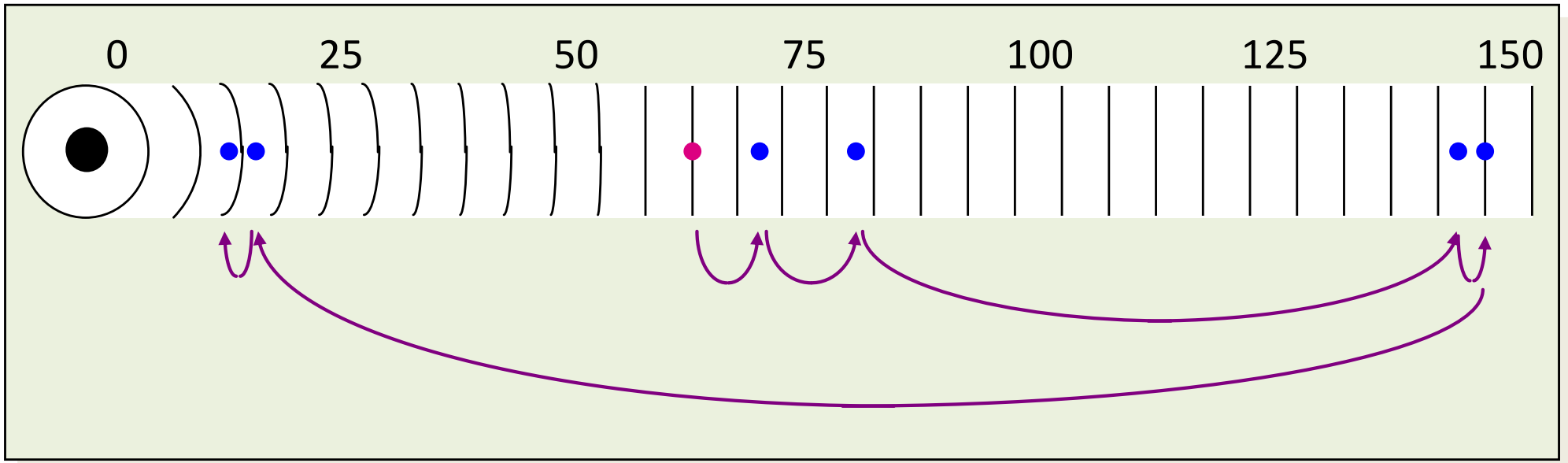
Greedy scheduling: *Shortest Seek Time First*

Rearrange queue from:

83	72	14	147	16	150

To:

14	16	150	147	83	72



SSTF scheduling results in the head moving 221 tracks

Can we do better?

Disk Head Scheduling: SSTF Text Description

- Greedy scheduling: Shortest Seek Time First
- Original queue of requests: 150, 16, 147, 14, 72 and 83
- Head still starts on track 65
- Rearrange queue to be: 72, 83, 147, 150, 16 and 14
  - Now requests are in shortest seek time first order
- Order of track moves
  - Move from track 65 to track 72, means moving 7 tracks
  - Move from track 72 to track 83, means moving 11 tracks
  - Move from track 83 to track 147, means moving 64 tracks
  - Move from track 147 to track 150, means moving 3 tracks
  - Move from track 150 to track 16, means moving 134 tracks
  - Move from track 16 to track 14, means moving 2 tracks
- Total number of track moves: 7 + 11 + 64 + 3 + 134 + 2 = 221 tracks
- Could we do better?

Disk Head Scheduling: SCAN/Elevator/LOOK

SCAN/Elevator scheduling: Move the head in one direction until the end of the disk is reached and then reverse.

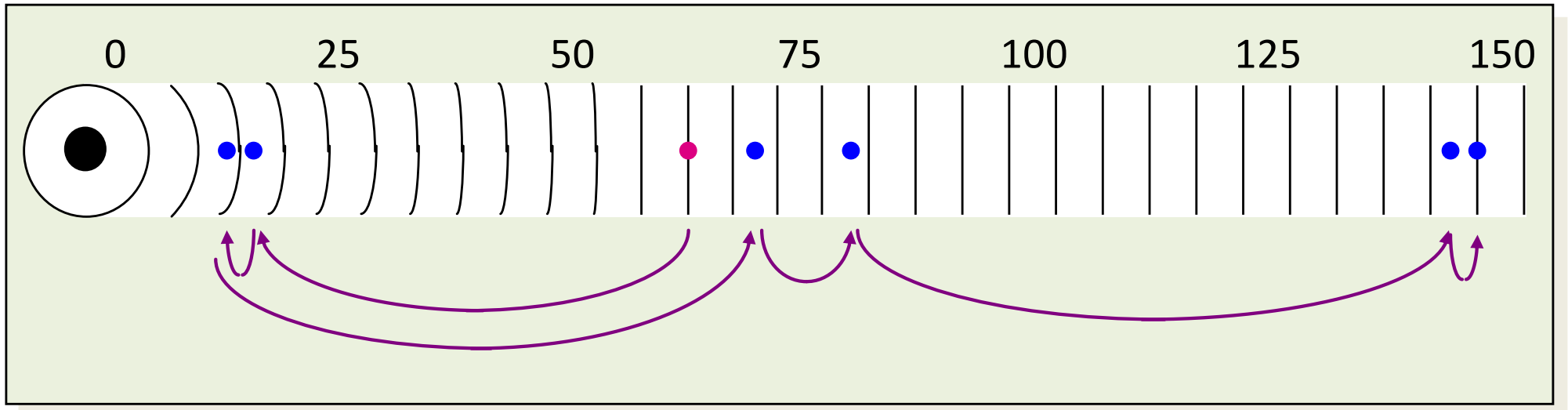
Simple optimization: the head is reset when no more requests exist between the current head position and the approaching edge of the disk (*LOOK* scheduling)

Rearrange queue from:

83	72	14	147	16	150

To:

150	147	83	72	14	16



Moves the head 187 tracks

Disk Head Scheduling: SCAN/Elevator/LOOK Text Description

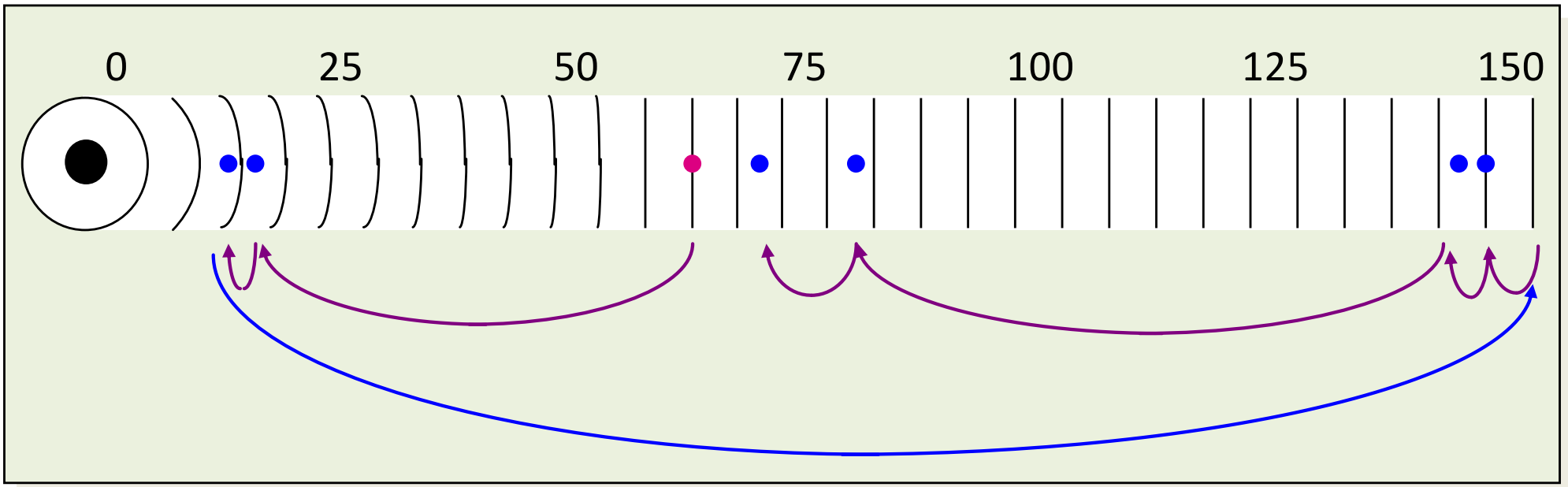
- SCAN/Elevator scheduling: move the head in one direction until all requests have been serviced and then reverse
  - Simple optimization: the head is reset when no more requests exist between the current head position and the approaching edge of the disk (LOOK scheduling)
- Original queue: 150, 16, 147, 14, 72 and 83
- Head starts at track 65 (as before) and is moving left towards lower addresses
- Rearrange queue to be: 16, 14, 72, 83, 147 and 150
- Order of track moves
  - Move from track 65 to track 16, means moving 49 tracks
  - Move from track 16 to track 14, means moving 2 tracks
    - Last request, so switch directions so head moves right towards higher addresses
  - Move from track 14 to track 72, means moving 58 tracks
  - Move from track 72 to track 83, means moving 11 tracks
  - Move from track 83 to track 147 means moving 64 tracks
  - Move from track 147 to track 150, means moving 3 tracks
- Total number of track moves: 49 + 2 + 58 + 11 + 64 + 2 = 187 tracks

Disk Head Scheduling: C-SCAN/C-LOOK

C-SCAN scheduling (“*Circular*”-*SCAN*)

- Move the head in one direction until an edge of the disk is reached and then reset to the opposite edge

Simple optimization: the head is reset when no more requests exist between the current head position and the approaching edge of the disk (called *C-LOOK* scheduling)



Disk Head Scheduling: C-SCAN/C-LOOK Text Description

- C-SCAN scheduling ("Circular"-SCAN)
  - move the head in one direction until an edge of the disk is reached and then reset to the opposite edge
  - simple optimization: the head is reset when no one requests exist between the current head position and the approaching edge of the disk, called C-LOOK scheduling
- Original queue: 150, 16, 147, 14, 72 and 83
- Head starts at track 65 (as before) and is moving left towards lower addresses
- Rearrange queue to be: 16, 14, 150, 147, 83 and 72
  - Head is always moving left towards lower addresses
  - Works off the optimized jump to the outer edge of the disk

Example: 500 Random Reads

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle Speed	7200 RPM
Average seek time	10.5/12.0 ms
Maximum seek time	19 ms
Track-to-track	1ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer Memory	16MB
Power	
Typical	16.35W
Idle	11.68W

- Workload
  - 500 read requests
  - Randomly chosen sector
  - Served in FIFO order
- How long to service them?
  - seek time: 10.5ms (avg)
  - rotation time
    - 7200 RPM = 120 RPS
    - Rotation time 8.3ms
    - On average, half of that: 4.15ms
  - transfer time
    - at least 54 MB/s
    - 512 bytes transferred in 9.25 microseconds
  - Total time:
    - 500 x (10.5 + 4.15 + .009) = 7.33 sec

Example: Effects on Disk Scheduling/C-SCAN

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle Speed	7200 RPM
Average seek time	10.5/12.0 ms
Maximum seek time	19 ms
Track-to-track	1ms
Surface transfer time	54-128 MB/s
Host transfer time	375 MB/s
Buffer Memory	16MB
Power	
Typical	16.35W
Idle	11.68W

- Workload
  - 500 read requests
  - Randomly chosen sectors
  - Disk head on outside track
  - Served in C-SCAN order
- How long to service them?
  - seek time
    - average seek for one request = 0.2% across disk
    - estimate as 1-track seek + interpolation with avg seek time
      - 1 + (2/33.3) x 10.5 = 1.06ms
  - rotation time
    - On average, half of that: 4.15ms
    - Same as before (don't know head position when seek ends)
  - transfer time
    - just as before, at least 9.5 microseconds
  - Total time:
    - 5.22ms per sector
    - For 500 sectors
      - 500 x 5.22ms = 2.61s

Question

Which algorithm has the best average response time?

A. FIFO
B. SSTF
C. SCAN/Elevator
D. C-SCAN

Okay, but…

So… a few questions:

1) How does the OS communicate with these devices, which need input and output data, but are much slower than the CPU? Does it do something to hide latency?
2) Does the I/O subsystem to anything to hide latency?
3) What about the file system? Does it anything to hide latency?

File System Design

Or Making the Disk Useful

History of File Systems

- Phase One: Basic FS
  - Technical motivation: make it work
- Phase Two: Performance (efficiency and locality)
  - Technical motivation: optimize disk usage, support many file types
- Phase Three: Optimize Writes
  - Technical motivation: lots of memory, writes dominates reads
- Phase Four: Scale and Distribution
  - Technical motivation: networks, servers, big data

The File System Abstraction

- Presents applications with **persistent**, **named** data
- A **file** is a named collection of related information recorded on secondary storage. Has two parts:
  - Data: what a user or application puts in the file
    - array of untyped bytes (In the end, it's all just bits!)
  - Metadata: file attributes added and managed by the OS
    - Name, type, location, size, creator, creation time, security info, modification time
  - File operations include create, open, read, write, seek, delete ….
- A directory provides names for files
  - list of human readable names
  - a mapping from each name to a specific underlying file or directory (hard link). [A soft link is a mapping from a file name to another file name]

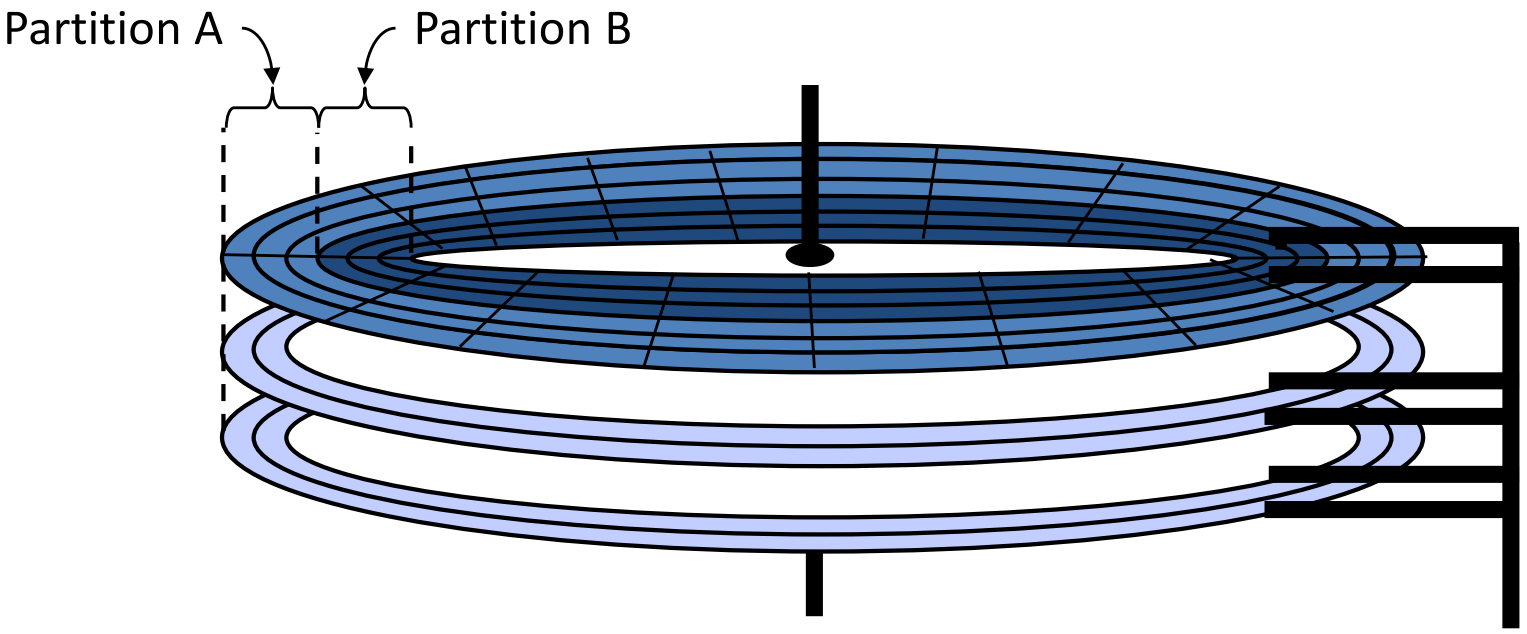
File System Functionality

- Provide the file naming organization, such as directories
- Manage partition layout
  - Pick the blocks that constitute a file
  - Balance locality with expandability
  - Manage free space
- Translate **from** file name and offset **to** data block
  - Support sequential and random access

Disk Partitions

Disks are typically partitioned to minimize the largest possible seek time

- A partition is a collection of cylinders
- Each partition is logically separate



File System Metadata

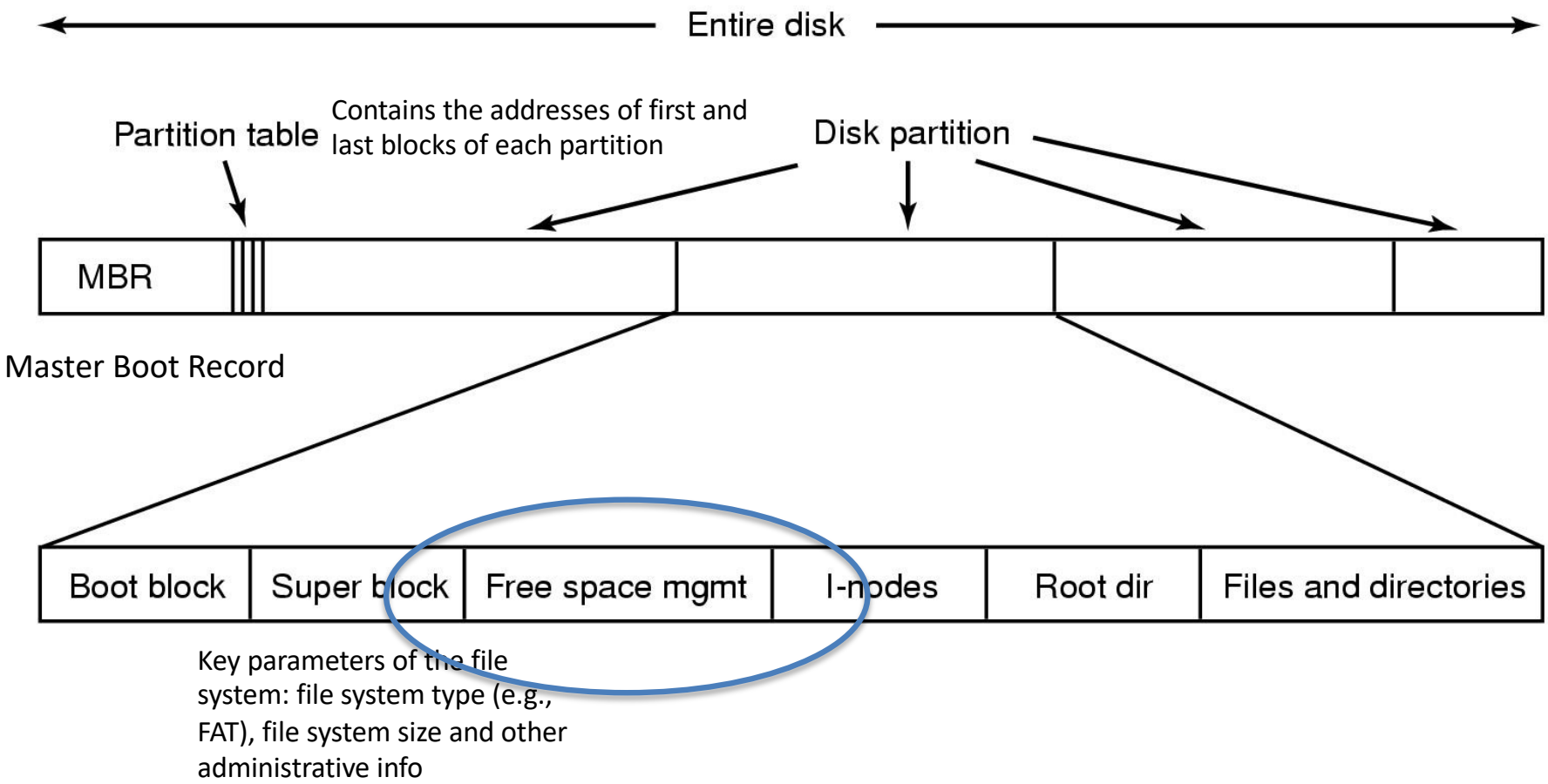
- The file system has metadata:
  - File system type
  - Number of blocks in the file system
  - File system permissions (e.g., read-only)
  - Free space information
  - …
- FS metadata is stored in the *superblock*
  - If the metadata is too big for the block, then the location of the metadata is stored

File System Data

File system data includes all the data about file locations and placement

- file headers (contain file metadata… more soon!)
- file data
- …

File System Layout on Disk



File System Layout on Disk: Text Description

- Components of the entire disk
  - MBR - Master Boot Record
  - Partition table: contains the addresses of first and last blocks of each partition
  - Disk partitions
- Components of each partition
  - Boot block
  - Super block
  - Free space management
  - I-nodes
  - Root directory
  - Files and directories
- Components of Super Block
  - File system type (eg FAT)
  - File system size
  - Key parameters of system
  - Other administrative info

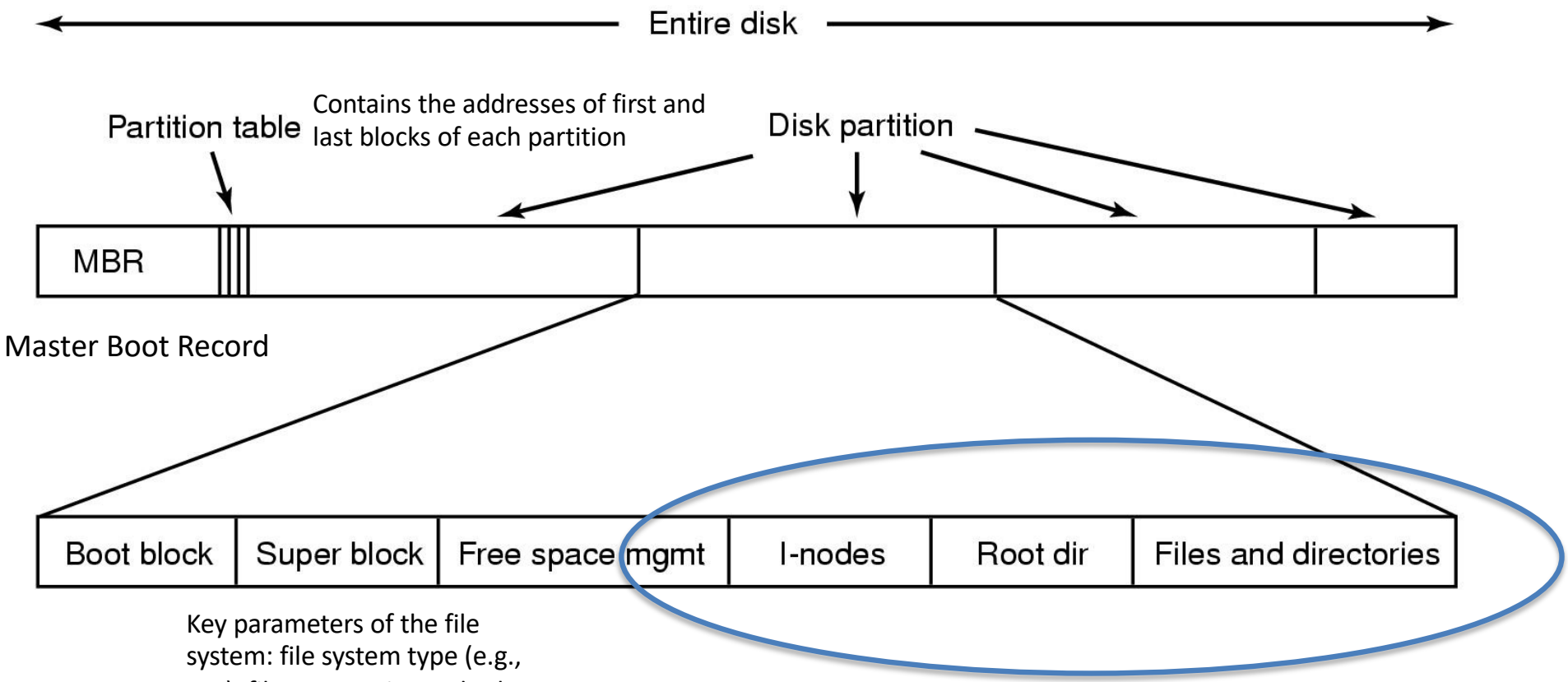
Tracking Free Space

- Separate free space data structures for data and metadata

- Two methods:
  - Free list
  - Bitmap

    11111000011111100111010111110011
    - one bit for each block on the disk
    - if bit is 1 then in is allocated, if 0 it is free

File System Layout on Disk



Goals in File System Design

- Efficiency (in space and time!)
  - Use disk space effectively
  - Optimize disk accesses
- Usability
  - Easy interface for application programmers
  - Easy interface for general computer users
- Reliability
  - Writes to stable storage can be interrupted at any time
  - Protecting user data from corruption is important

Let’s design the file system!