I/O Device Interfacing and Disks

CS439: Principles of Computer Systems

March 25, 2015

Last Time

Garbage Collection!

- Reachability: Tracing, reference counting
- Mark-Sweep
- Mark-Compact
- Semi-Space
- Generational Hypothesis

Today’s Agenda

- I/O Device Interfacing
  - How I/O hardware influences the OS
  - I/O Services provided by the OS
    - Implementation
  - How the OS can improve performance of I/O
- Disks
  - How they work
  - Scheduling Algorithms
  - Optimizations

Overview

Programmer

`<high-level interface>`

Operating System

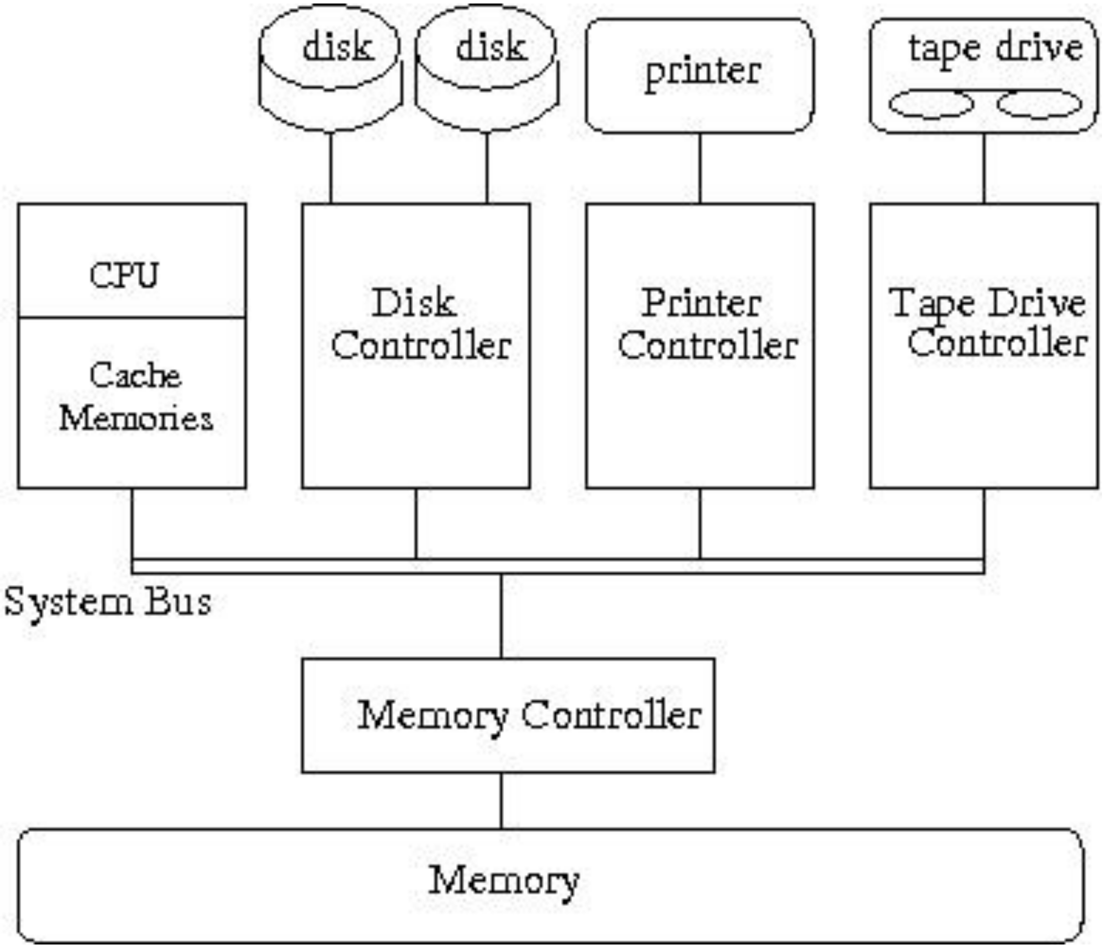
`<low-level interface>`

Device Controller

`<lowest-level interface>`

Device

I/O Interfaces


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.

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

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
- Examples: keyboard, disk

Communication

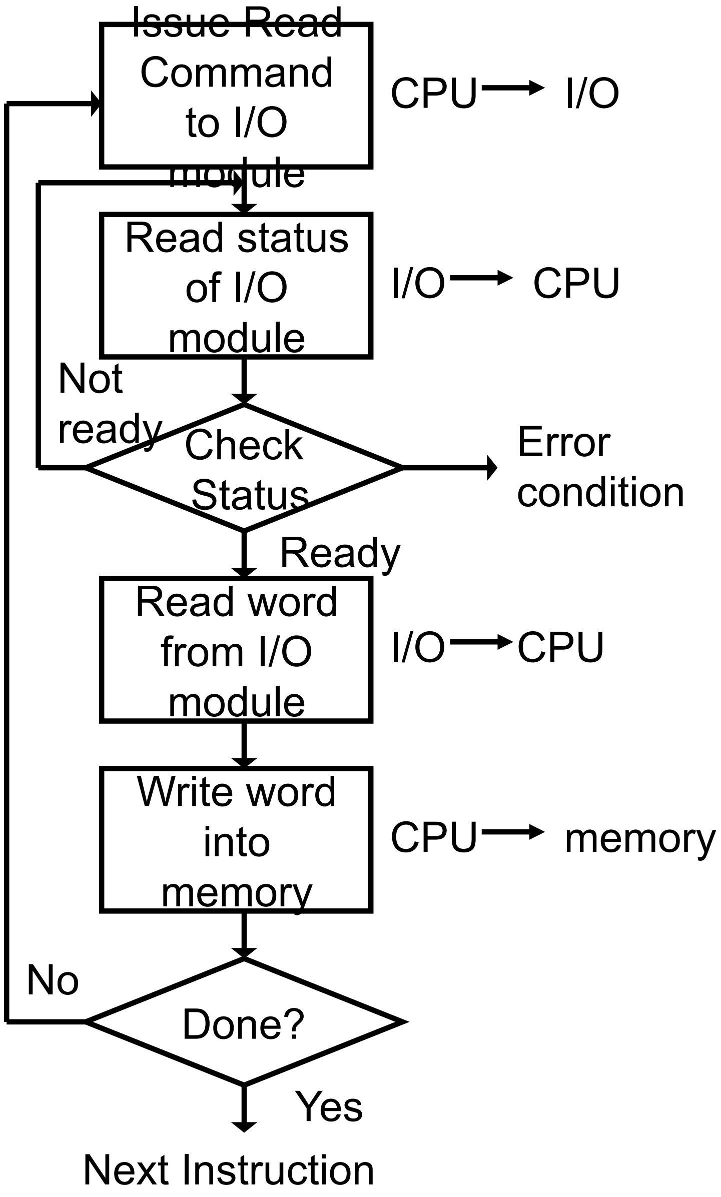
- The OS typically communicates with I/O devices through their controller
- OS places information in the device’s registers for the controller to send to the device
- Controller places information from the device in the registers for the OS

Three ways for this communication to take place…

Method One: Polling (also known as Programmed I/O)

- 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
- CPU observes the change to idle and , if it were an input operation, reads the date

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

iClicker Question

What happens if the device is busy when the CPU has a request?

A. The CPU moves the operation to a pending list, and it moves to other tasks until it receives an interrupt from the device

B. The CPU becomes idle

C. The CPU moves to another task, but it keeps the operation on its todo list and tries back

Polling: Advantages

- Handles data promptly
  - good choice for modems or keyboards where data will be lost if it isn’t moved from the device fast enough

- Are there disadvantages?

- What happens if the device is faster than the polling rate? slower?

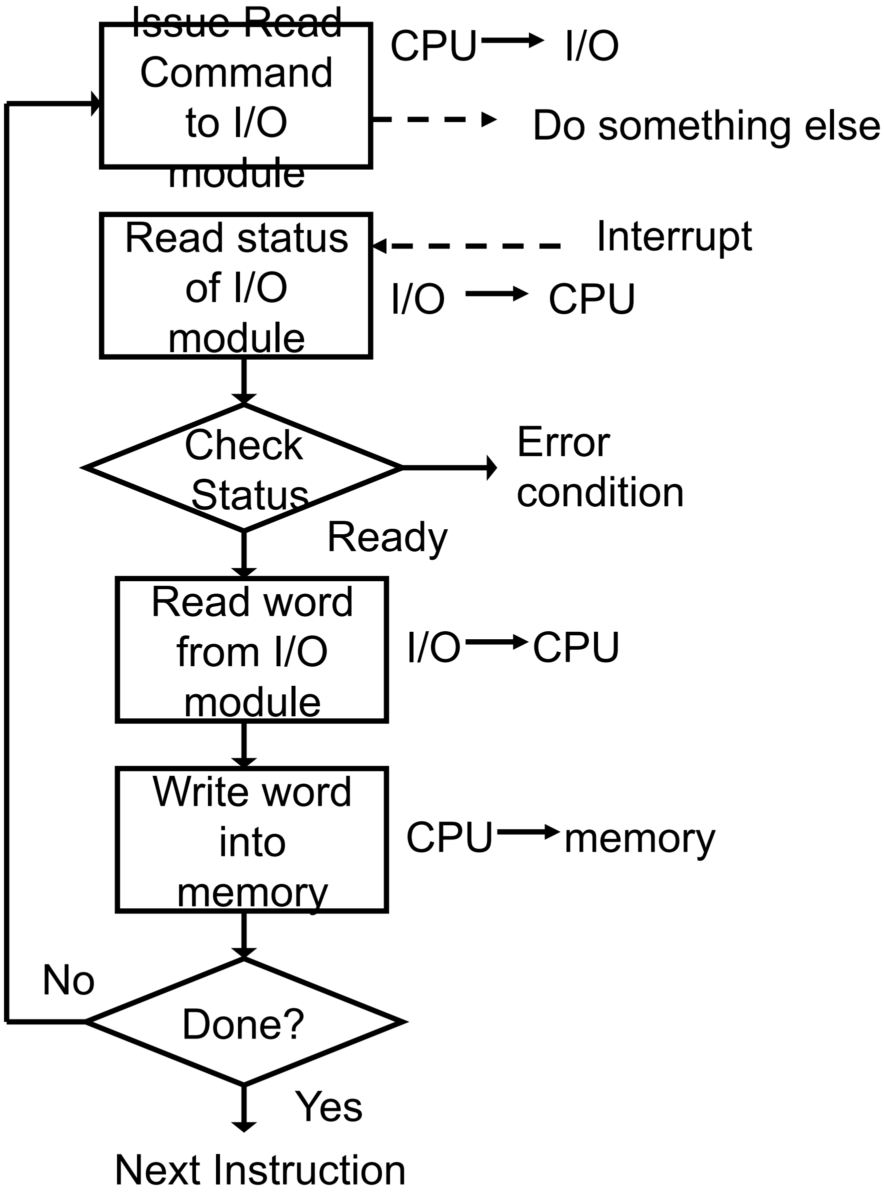
Method Two: Interrupts

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

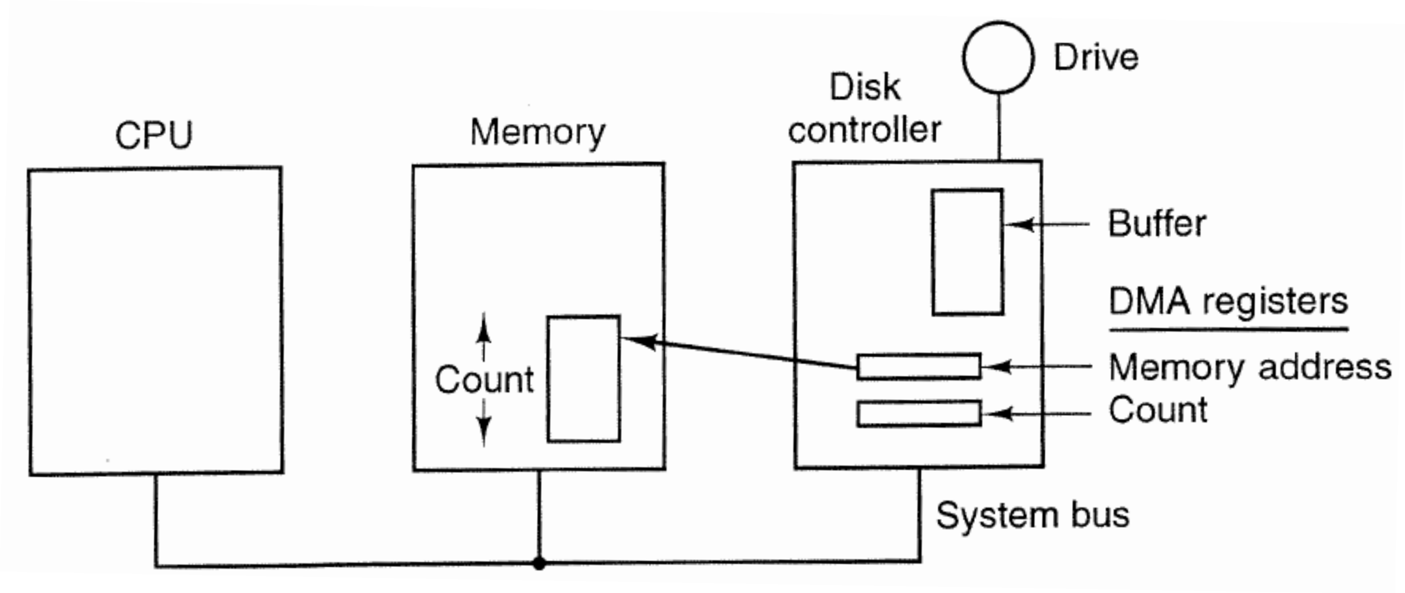
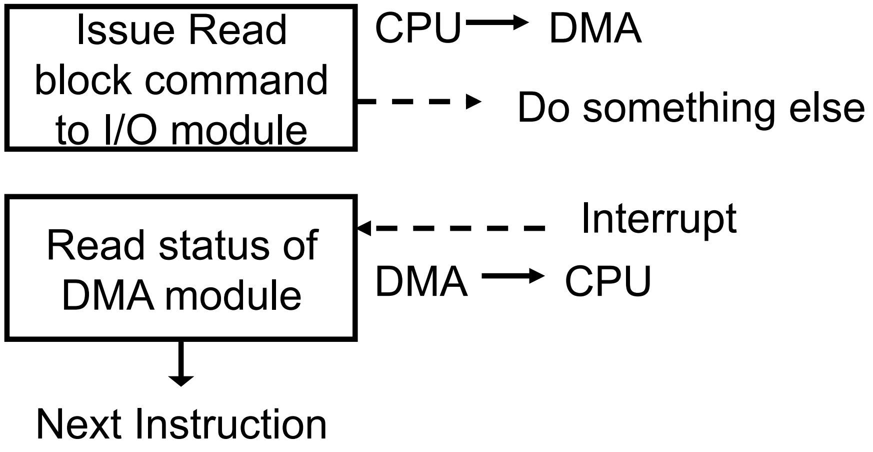


Diagram: DMA Transfer

- 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



Issue Read block command to I/O module
CPU → DMA
Do something else
Interrupt
Read status of DMA module
DMA → CPU
Next Instruction

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

iClicker Question

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

A. Polling

B. Interrupt

C. DMA

Improving Performance: I/O Buffering

I/O devices typically contain a small on-board memory where they can store data temporarily before transferring to/from memory

- A disk buffer stores a block when it is read from the disk
- It is transferred over the bus by the DMA controller into a buffer in physical memory
- The DMA controller interrupts the CPU when the transfer is complete

Advantages to Buffering

- Cope with speed mismatches between the device and the CPU
  - Compute the context of a display and place in buffer (slow), then zap buffer to screen (fast)
- Cope with devices that have different data transfer sizes
  - ftp brings a file over the network one packet at a time, but stores to disk happen one block at a time
- Minimize the time a user process is blocked on a write
  - To write to a file, the write is done immediately to a kernel buffer and control is returned to the user program. The write is done later.

Improving Performance: Caching

- Improve disk performance by reducing the number of disk accesses
- Keep recently used disk blocks in main memory after the I/O call that brought them into memory completes
- What happens if we write to the cache?
  - *write-through*: write to all levels of memory containing the block, including the disk (High reliability)
  - *write-back*: write to only the fastest memory containing the block, write to slower memories and disk sometime later (faster)

Application Programmer’s View

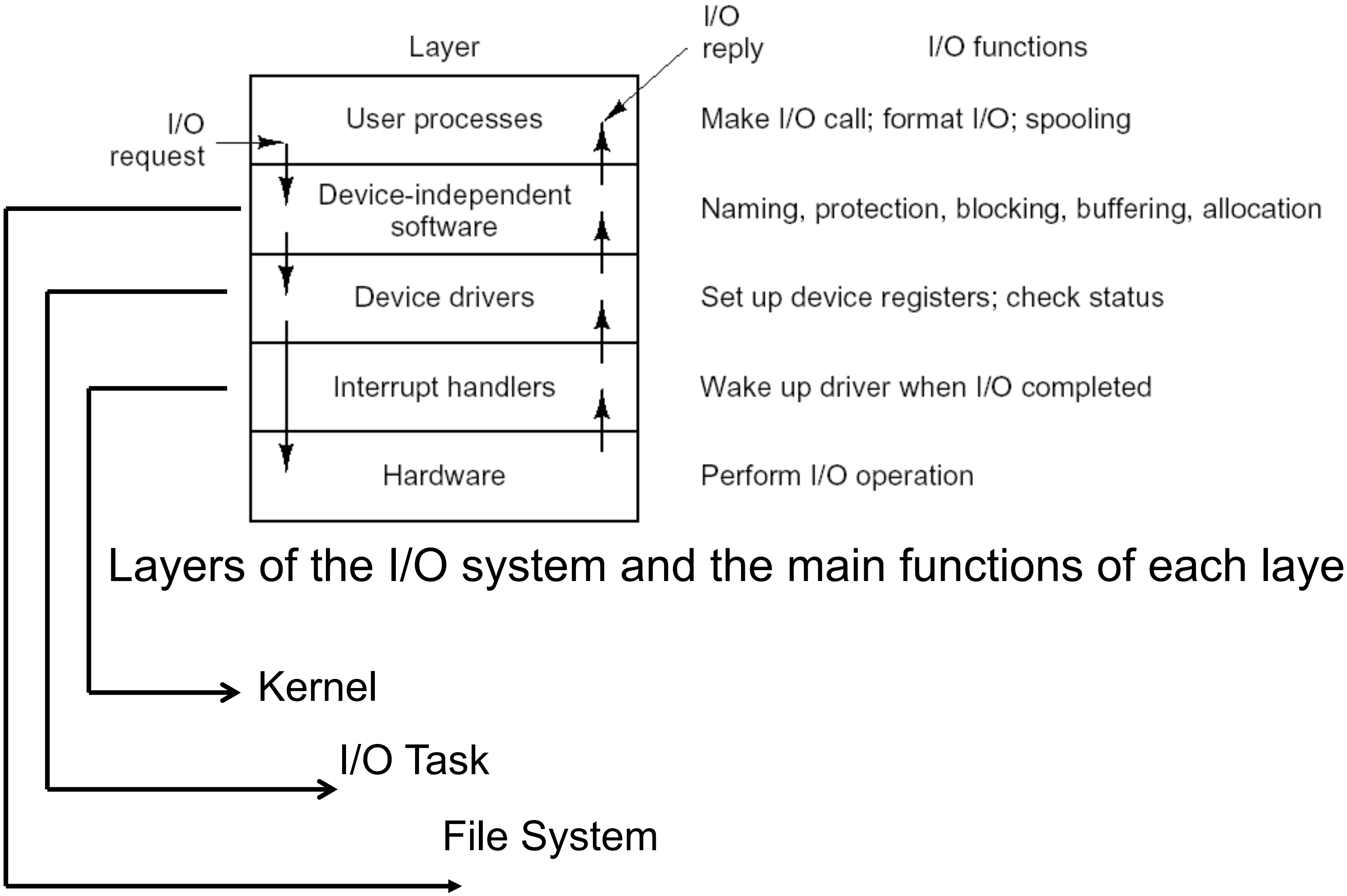
The OS provides a high-level interface to devices, greatly simplifying the programmer’s job.

- 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

- (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
- Access control (protection)
- Operations appropriate to the devices
- Device allocation and release
- Buffering, caching, and spooling to allow efficient communication
- I/O scheduling
- Error handling and failure recovery associated with devices
- Device drivers that implement device-specific behaviors

Tracing an I/O Transaction



Layer	I/O functions
User processes	Make I/O call; format I/O; spooling
Device-independent software	Naming, protection, blocking, buffering, allocation
Device drivers	Set up device registers; check status
Interrupt handlers	Wake up driver when I/O completed
Hardware	Perform I/O operation

Layers of the I/O system and the main functions of each layer.

Kernel

I/O Task

File System

Tracing an I/O Transaction: Text Description

- There are 5 layers to the transaction
  - 1. User processes
    - functions: make I/O call, format I/O, spooling
  - 2. Device-independent software
    - functions: naming, protection, blocking, buffering, allocation
    - part of the kernel
  - 3. Device Drivers
    - functions: set up device registers, check status
    - part of the I/O task
  - 4. Interrupt handlers
    - functions: wake up driver when I/O completed
    - part of the file system
  - 5. Hardware
    - functions: perform I/O operation
- Transaction order for an I/O request
  - User processes
  - Device-independent software
  - device drivers
  - hardware
  - note: the interrupt handlers part was skipped
- Transaction order for an I/O reply – generally opposite order from I/O request
  - hardware
  - interrupt handlers
  - device drivers
  - device-independent software
  - user processes

Putting It All Together: A Typical Read Call

1. User process requests a read from a device
2. OS checks if data is in a buffer. If not,
   a) OS tells the device driver to perform input
   b) Device driver tells the DMA controller what to do and blocks itself
   c) DMA controller transfers the data to the kernel buffer when it has all been retrieved from the device
   d) DMA controller interrupts the CPU when the transfer is complete
3. OS transfers the data to the user process and places the process in the ready queue
4. When the process gets the CPU, it begins execution following the system call

Disks

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 = .1GB/$1, Disks = 14.6GB/$1
- Also known as *Secondary Storage*

Storage Devices

- We'll focus on two types of persistent storage
  - magnetic disks
    - servers, workstations, laptops, desktops
  - flash memory
    - smart phones, tablets, cameras, some laptops
- Others exist(ed)
  - tapes
  - drums
  - clay tablets

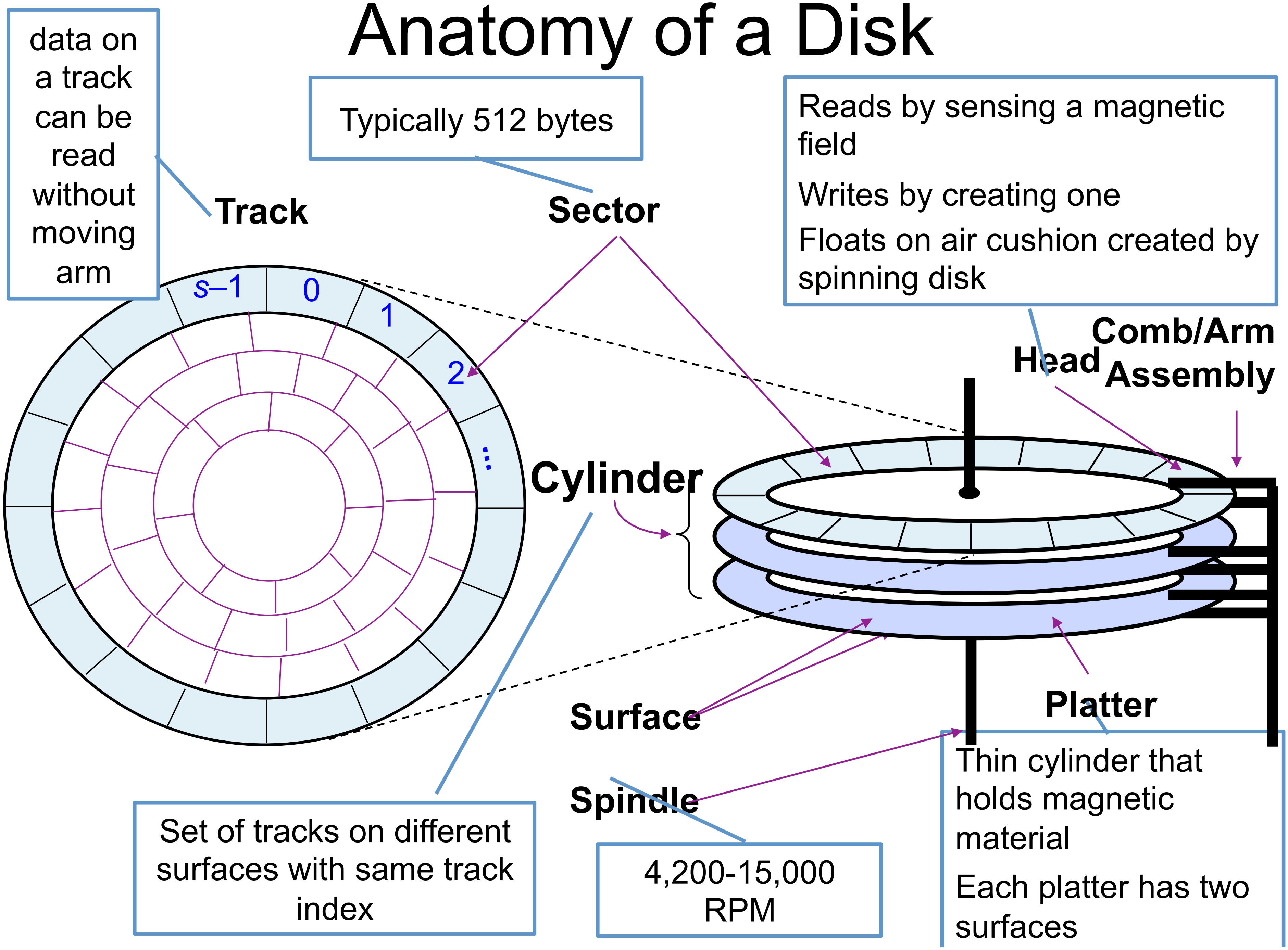
How Magnetic Disks Work

- 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

How Disks Work

- Disks come organized in a *disk pack* consisting of a stack of *platters*.
- Disk packs use both sides of the platters, except on the ends.
- *Tracks* are concentric rings on disk with the bits laid out serially
- Each track is split into *sectors* or *blocks*, the minimum unit of transfer from the disk
- *Cylinders* are matching sectors on each surface
- Reads are performed by *read/write heads* at the end of *arms*
- In disk packs, the arms and read/write heads are called the *comb*

Anatomy of a 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

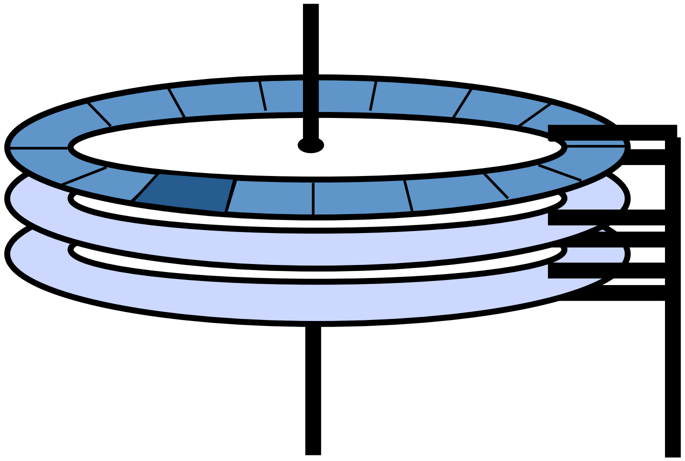
Each platter has two surfaces

Anatomy of a 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

Disk Operations

- Disk operations are in terms of radial coordinates
- 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 spin



Read time:

seek time + *rotation time* + *transfer time*

A Closer Look: Seek Time

- *Seek time*: time to position head over track/ cylinder
  - depends on how fast the hardware moves the arm
- Maximum: time to go from innermost to outermost track
  - more than 10ms, may be over 20ms
- Average: average across seeks between each possible pair of tracks
  - approximately time to seek 1/3 of the way across the disk
  - often pessimistic estimate
- 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

A Closer Look: Rotation Time

- *Rotation time*: time for the sector to rotate underneath the head
  - depends on how fast the disk spins
- Today most disks rotate at 4,200 to 15,000 RPM
  - 15ms to 4ms per rotation
  - good estimate is half that amount
- Head starts reading as soon as it settles

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)

iClicker Question

Which component of disk access time is the longest?

A. Rotation Time

B. Transfer Time

C. Seek Time

Example: Toshiba MK3254GSY

Size	
Platters/Heads	2/4
Capacity	320 GB
Performance	
Spindle Speed	7200 RPM
Average seek time R/W	10.5/12.0 ms
Maximum seek time R/W	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

Example: Toshiba MK3254GSY Plain Text

- Size metrics:
  - 2 platters
  - 4 heads
  - 320 GB capacity
- Performance metrics:
  - Spindle spins at 7200 RPM
  - Average seek time: for read 10.5 ms, for write 12.0 ms
  - Maximum seek time for both read and write is 19 ms
  - Seeking from a track to the next track takes 1 ms
  - Surface transfer time is 54 - 128 MB/s
  - Host transfer time is 375 MB/s
  - Buffer Memory is 16 MB
- Power Metrics
  - Typical power is 16.35 W
  - Idle power is 11.68 W

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: 500 Random Reads Plain Text

- Note: size, performance, and power metrics are the same as slide 45
- Workload
  - 500 read requests
  - Randomly chosen sector
  - Served in FIFO order
- How long to service them?
  - Seek time is 10.5 ms, on average
  - Rotation time calculations
    - 7200 RPM is 120 RPS
    - Rotation time is 8.3 ms
    - On average, rotation time requires half of a rotation, so 4.15 ms
  - Transfer time
    - At least 54 MB/s
    - 512 bytes transferred in 9.25 microseconds
  - Total time for all 500 requests
    - 500 * read time = 500 * (10.5 + 4.15 + .009) = 7.33 sec

Example: 500 Sequential 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

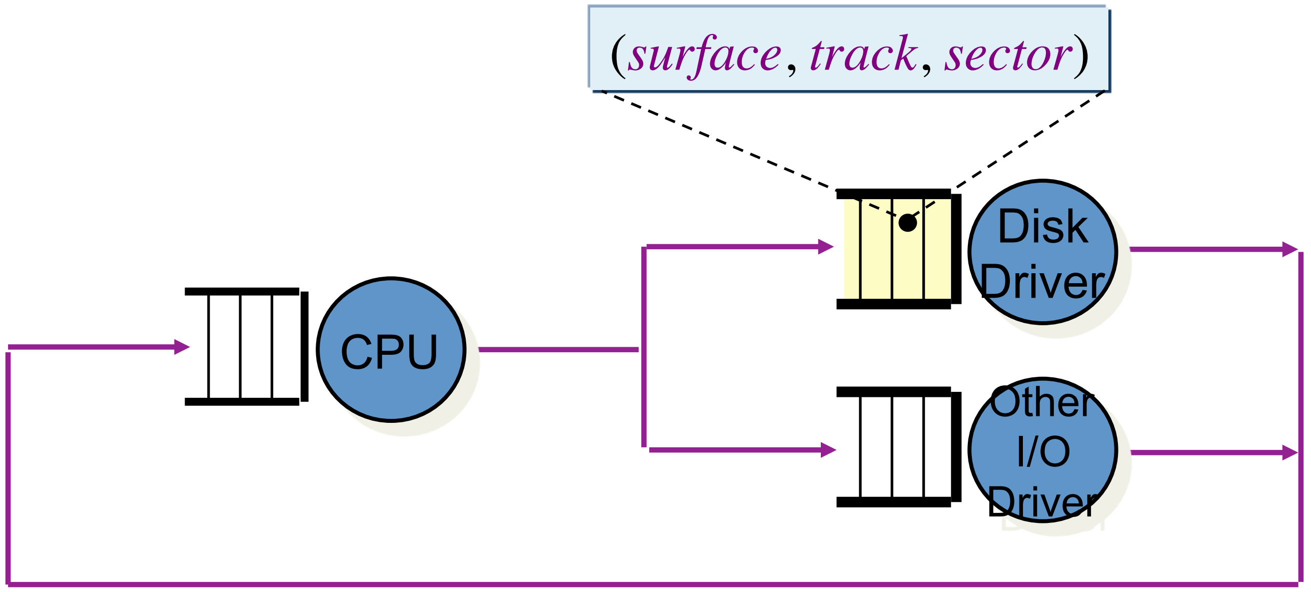
- Workload
  - 500 read requests
  - Sequential sectors on the same track
  - Served in FIFO order
- How long to service them?
  - seek time: 10.5ms (avg)
  - rotation time
    - On average, half of that: 4.15ms
    - Same as before
  - transfer time
    - outer track: 500 x (.5/128000) = 2ms
    - inner track: 500 x (.5/54000) = 4.6ms
  - Total time:
    - outer track: 2 + 4.15 + 10.5 = 16.65ms
    - inner track: 4.6 + 4.15 + 10.5 = 19.25ms

Example: 500 Sequential Reads Plain Text

- Example: 500 Sequential Reads
- Note: size, performance, and power metrics are the same as slide 45
- Workload
  - 500 read requests
  - Sequential sectors on the same track
  - Served in FIFO order
- How long to service them?
  - Seek time is 10.5 ms on average
  - Rotation time
    - On average, half of that: 415 ms
    - Same as before
  - Transfer time
    - Outer track: 500 * (.5/128000) = 2 ms
    - Inner track: 500 * (.5/54000) = 4.6 ms
    - Note: because of circular rotation outer tracks spin faster than inner tracks (more data will pass under the head in the same amount of time)
  - Total time:
    - Outer track: 2 + 4.15 + 10.5 = 16.65 ms
    - Inner track: 4.6 + 4.15 + 10.5 = 19.25 ms

Disk Head Scheduling

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



The OS maximizes disk I/O throughput by minimizing head movement through *disk head scheduling.*

Disk Head Scheduling: Plain Text

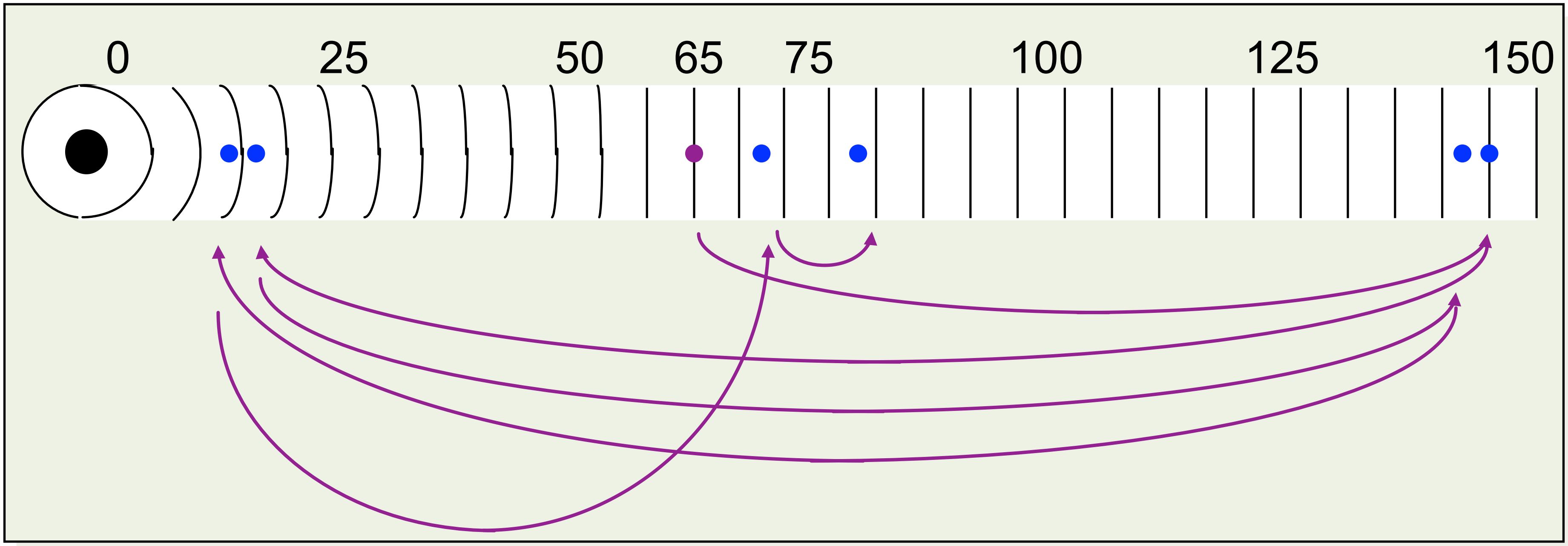
- In a multiprogramming/time sharing environment, a queue of disk I/O requests can form
  - Requests are generated by running program and then handled by the disk driver, where the queue forms
  - After I/O, information goes back to the CPU
- The OS maximizes disk I/O throughput by minimizing head movement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

Disk Head Scheduling: FCFS/FIFO

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

83	72	14	147	16	150

and the head is on



FCFS/FIFO scheduling results in the head moving 550 tracks

Can we do better?

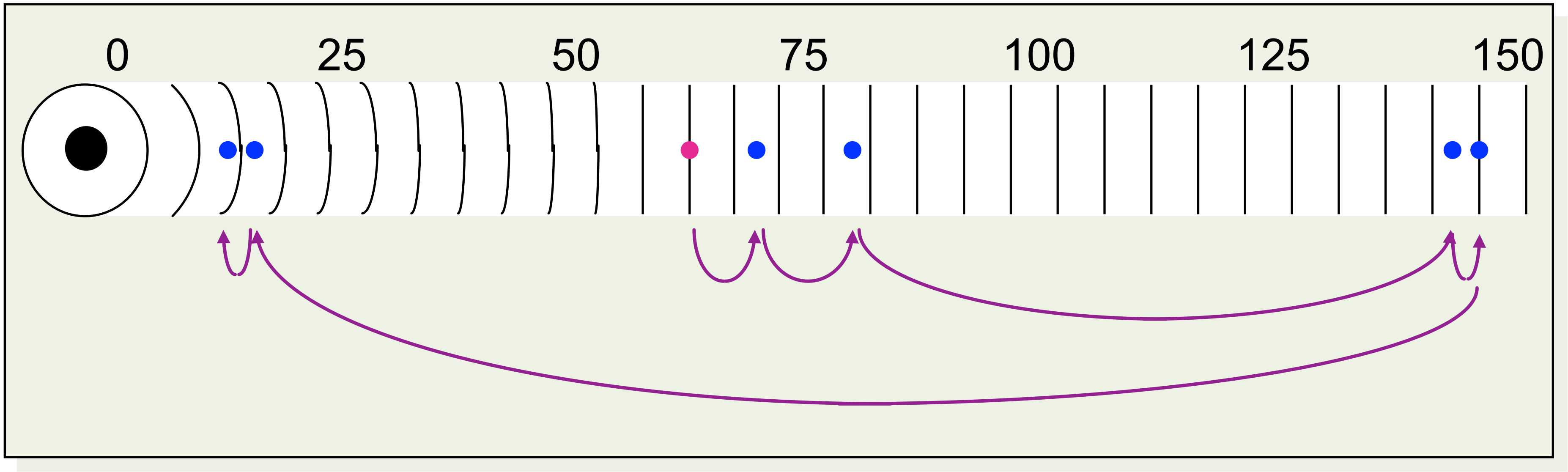
Disk Head Scheduling: FCFS/FIFO Text Description

- 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	82	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.

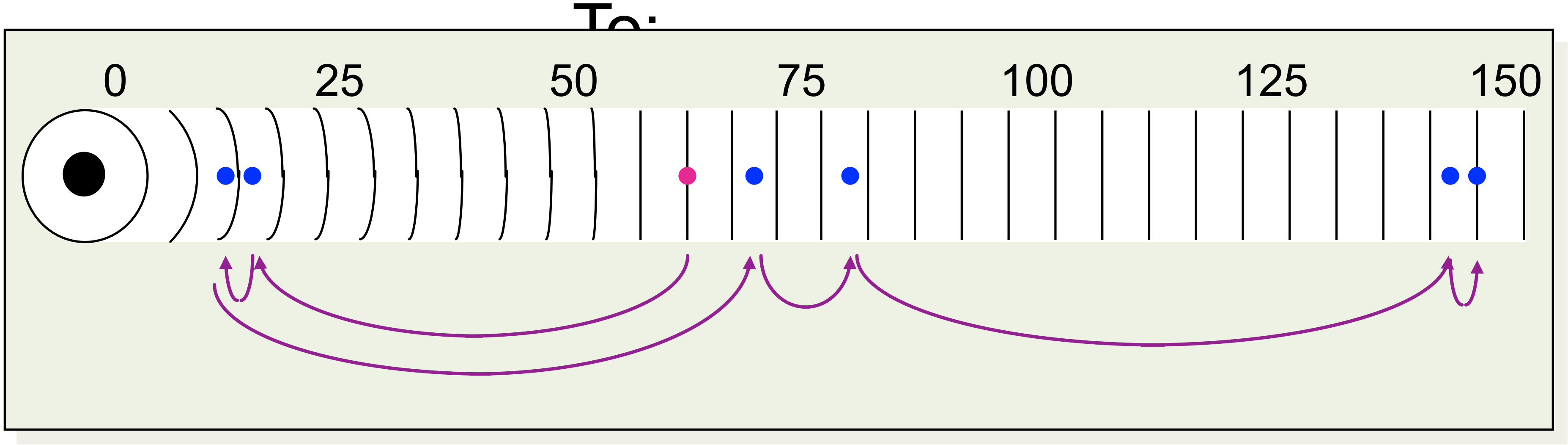
Simple optimization: the head is reset when no more requests exist between the current head position and the approaching edge of the disk (*LOOK* scheduling)

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

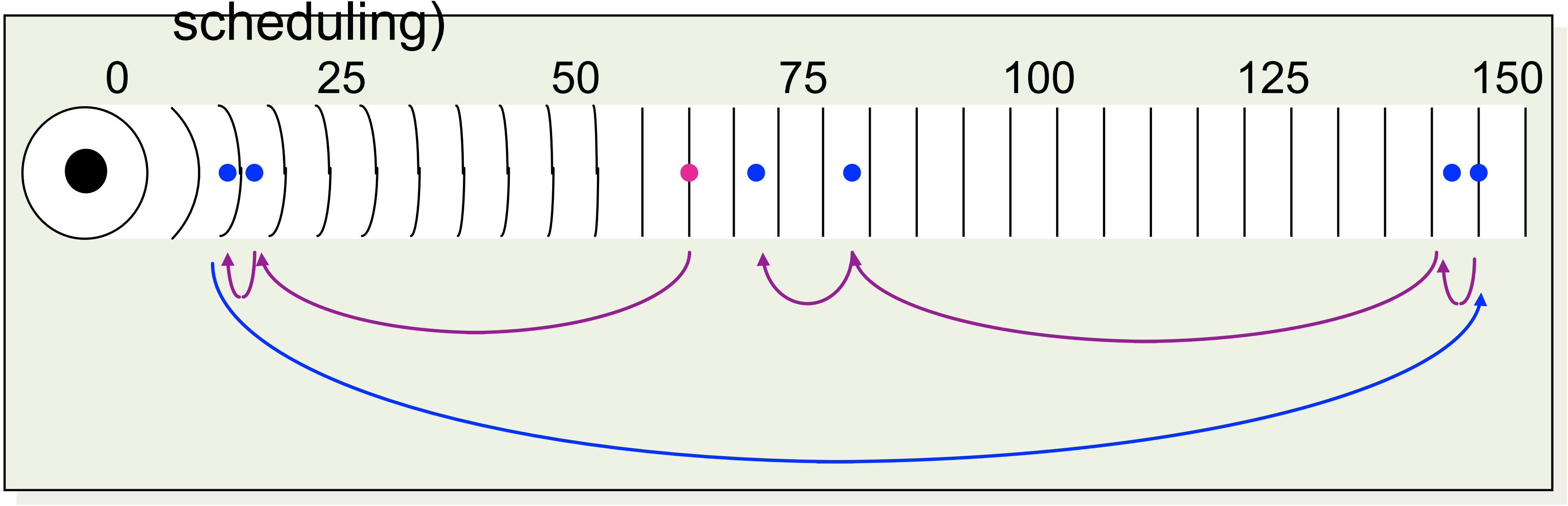
- SCAN/Elevator scheduling: move the head in one direction until all requests have been serviced and then reverse
  - Simple optimization: the head is reset when no more requests exist between the current head position and the approaching edge of the disk (LOOK scheduling)
- Original queue: 150, 16, 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: 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 block
    - For 500 blocks
      - 500 x 5.22ms = 2.61s

Example: Effects on Disk Scheduling/C-SCAN
Plain Text

- Disk metrics (Note that these are the same as those on slide 45):
  - Size metrics:
    - 2 platters
    - 4 heads
    - 320 GB capacity
  - Performance metrics:
    - Spindle spins at 7200 RPM
    - Average seek time: for read 10.5 ms, for write 12.0 ms
    - Maximum seek time for both read and write is 19 ms
    - Seeking from a track to the next track takes 1 ms
    - Surface transfer time is 54 - 128 MB/s
    - Host transfer time is 375 MB/s
    - Buffer Memory is 16 MB
  - Power Metrics
    - Typical power is 16.35 W
    - Idle power is 11.68 W
- 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 average seek time
      - 1 + (⅔.33)*10.5 = 1.06 ms
  - Rotation time
    - On average, half of that: 4.15 ms
    - Same as before (don’t know head position when seek ends)
  - Transfer time
    - Just as before, at least 9.5 microseconds
  - Total time
    - 5.22 ms per block
    - For 500 blocks = 500 * 5.22 ms = 2.61 s

iClicker Question

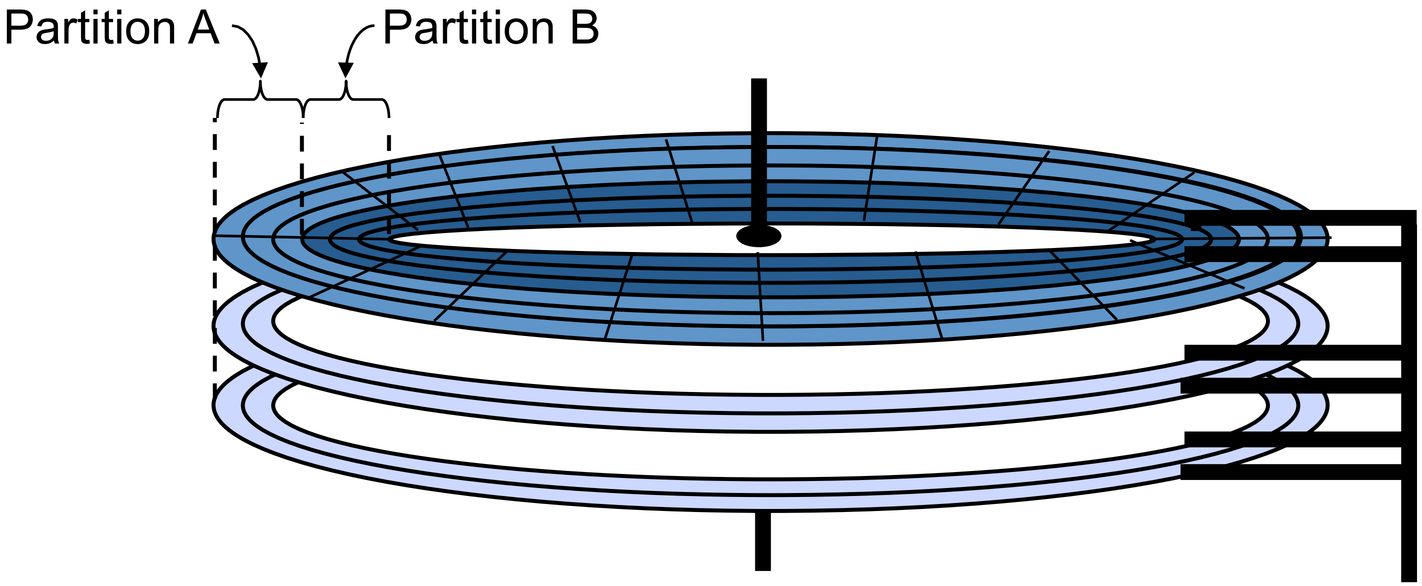
Which algorithm has the best response time?

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

Other Ways to Improve Performance: Partitioning

Disks are typically partitioned to minimize the largest possible seek time

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



Other Ways to Improve Performance: Interleaving

- *Problem:* Contiguous allocation of files on disk blocks only makes sense if the OS can react to one disk response and issue the next command before the disk spins past the next block
- *Idea*: Interleaving. Don’t allocate blocks that are physically contiguous, but those that are temporally contiguous relative to the speed with which a second disk request can be received and the rotational speed of the disk

Other Ways to Improve Performance: Buffering

- Read blocks from the disk ahead of the user’s request and place in buffer on the disk controller
  - Read those that are spinning by the head anyway
- Reduces the number of seeks
- Avoids “shoulda coulda”
  - Exploits locality

Reducing Overhead Summary

To get the quickest disk response time, we must minimize seek time and rotational latency:

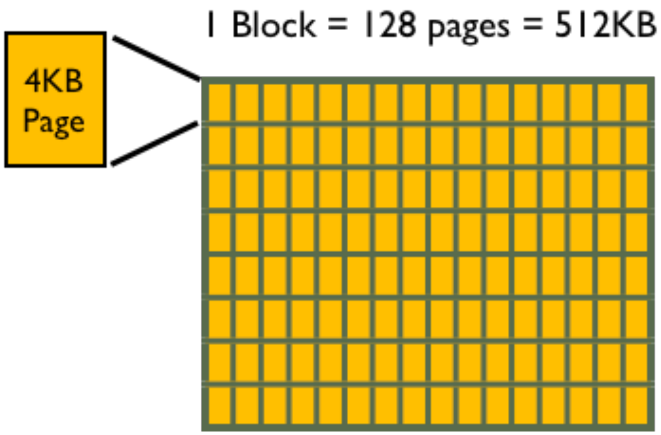
- Make disks smaller
- Spin disks faster
- Schedule disk operations to minimize head movement
- Lay out data on disk so that related data are on nearby tracks
- Place commonly used files where? on disk
- We should also pick our block size carefully:
  - If block size is too small, we will have low transfer rate because we need to perform more seeks for same amount of data
  - If block size is too large, we will have lots of internal fragmentation

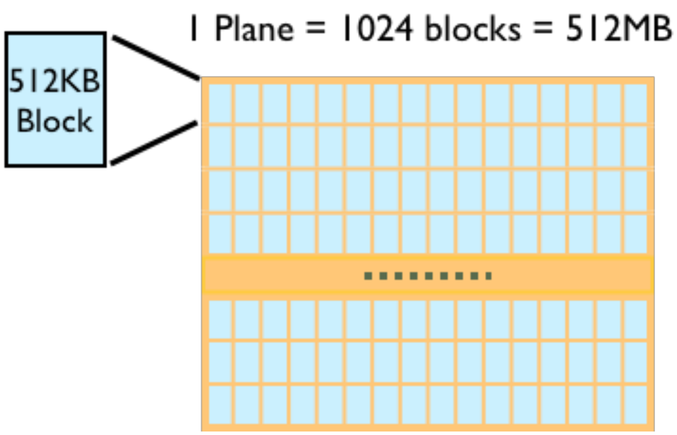
Another Way to Improve Performance: Flash Storage

No moving parts

- Better random access performance
- Less power
- More resistant to physical damage

NAND Flash Units





- Operations
  - Erase block
    - Before a page can be written, needs to be set to logical “1”
    - Erases must occur in block units
    - Operation takes several ms
  - Write page
    - tens of μs
  - Read page
    - tens of μs
  - Reads and writes *only* occur in page units
- Flash devices can have multiple independent data paths
  - OS can issue multiple concurrent requests to maximize bandwidth

NAND Flash Units

- 1 block is made of 128 pages is 512 KB
  - Each page is 4 KB
  - Note that block and page in this context have different meanings than a page in the virtual memory context or a block in the spinning disk context
- 1 plane is 1024 blocks, so it holds 512 MB
- Operations
  - Erase block
    - Before it can be written, needs to be set to logical “1”
    - Erases must occur in block units
    - Operation takes several ms
  - Write page
    - takes tens of microseconds
  - Read page
    - takes tens of microseconds as well
  - Reads and writes *only* occur in page units
- Flash devices can have multiple independent data paths
  - OS can issue multiple concurrent requests to maximize bandwidth

Example: Remapping Flash Drives

- Flash drive specs

  4 KB page

  3ms erase

  512KB erasure block (128 pages)

  50μs read page/write page

- How long to naively read/erase/and write a single page?

  $128 \times (50 \times 10^{-3} + 50 \times 10^{-3}) + 3 = 15.8\text{ms}$

- Suppose we use remapping, and we always have a free erasure block available. How long now?

  $3/128 + 50 \times 10^{-3} = 73.4\mu\text{s}$

Flash Durability

- Flash memory stops reliably storing a bit
  - after many erasures (on the order of 10^3 to 10^6)
  - after a few years without power
  - after nearby cell is read many times (read disturb)
- To improve durability
  - error correcting codes
    - extra bytes in every page
  - management of defective pages/erasure blocks
    - firmware marks them as bad
    - wear leveling
      - spreads updates to hot logical pages to many physical pages
    - spares (pages and erasure blocks)
      - for both wear leveling and managing bad pages and blocks

series
Solid State Drive

Consider 500 read requests to randomly chosen pages. How long will servicing them take?

- 500 x 26µs = 13ms
  - spinning disk: 7.8s

How do random and sequential read performance compare?

- effective bw random
  - (500 x 4)KB / 13ms ≈ 154MB/s
- ratio: 154/270 = 57%

500 random writes

- 500s/2000 = 250ms

How do random and sequential write compare?

- effective bw random
  - (500 x 4)KB / 250ms = 8MB/s
- ratio: 8/210 = 3.8%

Size	
Capacity	300 GB
Page Size	4KB
Performance	
Bandwidth (seq reads)	270 MB/s
Bandwidth (seq writes)	210 MB/s
Read/Write Latency	75µs
Random reads/sec	38,500 (one every 26 µs)
Random writes/sec	2,000 (2,400 with 20% space reserve)
Interface	SATA 3GB/s
Buffer Memory	16MB
Endurance	
Endurance	1.1 PB (1.5 PB with 20% space reserve)
Power	
Active	3.7W
Idle	0.7W

Example: Intel 710 Series Solid State Drive
Plain Text

- Drive Metrics
  - Size metrics
    - 300 GB capacity
    - Page size is 4 KB
  - Performance Metrics
    - Bandwidth for sequential reads is 720 MB/s
    - Bandwidth for sequential writes is 210 MB/s
    - Read/Write latency is 75 microseconds
    - Random reads/sec is 38,500, so one every 26 microseconds
    - Random writes/sec is 2,000, can be 2,400 with 20% space reserve
    - Interface is SATA 3 GB/s
    - Buffer memory is 16 MB in size
  - Endurance Metrics
    - Endurance is 1.1 PB (1.5 PB with 20% space reserve)
  - Power Metrics
    - Active is 3.7 W
    - Idle is 0.7 W
- Consider 500 read requests to randomly chosen pages. How long will servicing them take?
  - 500 * 26 microseconds = 13 ms
    - spinning disk would be 7.8 s
- How do random and sequential read performances compare?
  - Effective bw random
    - (500 * 4)KB / 13 ms is about 154 MB/s
  - Ratio: 154/ 270 = 57%
- 500 random writes
  - 500s/ 2000 = 250 ms
- How do random and sequential write compare?
  - Effective bw random
    - (500 * 4)KB / 250 ms = 8 MB/s
  - Ratio: 8/ 210 = 3.8 %

Spinning Disk vs. Flash

Metric	Spinning Disk	Flash
Capacity/Cost	Excellent	Good
SequentialBW/Cost	Good	Good
Random I/O per sec/Cost	Poor	Good
Power Consumption	Fair	Good
Physical Size	Good	Excellent

Spinning Disk vs. Flash: Plain Text

- Spinning Disk
  - Capacity/Cost ratio is excellent
  - Sequential BW/Cost ratio is good
  - Random I/O per sec/Cost ratio is poor
  - Power consumption is fair
  - Physical size is good
- Flash
  - Capacity/Cost ratio is good
  - Sequential BW/Cost ratio is good
  - Random I/O per sec/Cost ratio is good
  - Power consumption is good
  - Physical size is excellent

I/O Summary

I/O is expensive for several reasons:

- typically involves physical movement which is slow
- multiple processes often contend for the devices
- typically supported by system calls and interrupt handling, which are slow

Approaches to improving performance:

- reduce the number of times data is copied by caching values in memory
- reduce the frequency of interrupts by using large data transfers when possible
- offload computation from the main CPU by using DMA controllers

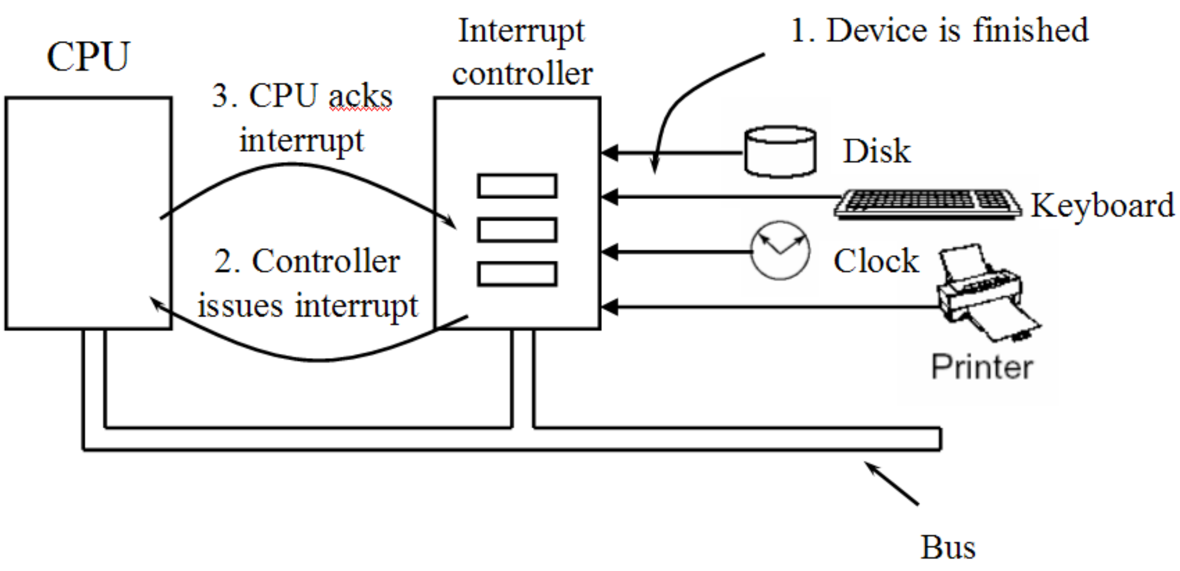
Disk Summary

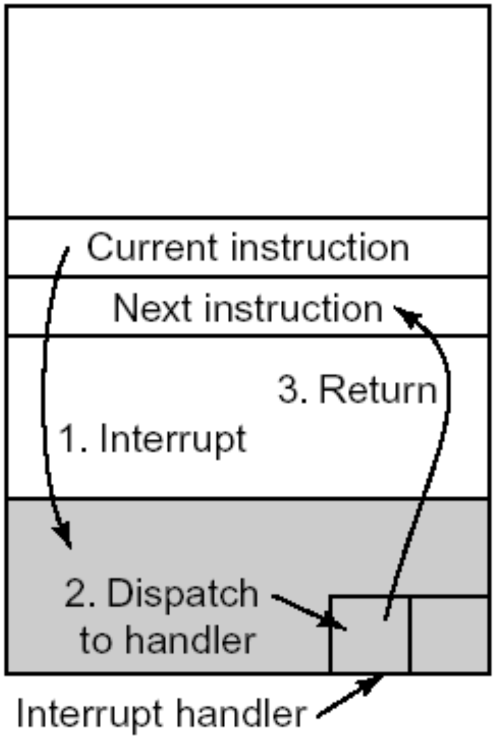
- Disks are slow devices relative to CPUs
- Spinning disks:
  - made up of sectors, tracks, cylinders, platters
  - Read/Write overhead = seek time+rotational delay +transfer time
    - Seek time is the largest cost
    - disk head scheduling algorithms to minimize it
  - We can also improve disk performance using partitions, interleaving, buffering
- Solid State drives:
  - Much faster
  - User less power
  - Have durability issues

Announcements

- Homework 7 due Friday 8:45a
- Project 2 due Friday 11:59pm
- Project 3 is posted due Friday, 4/17
  - No hand-holding
  - Project 2 must be working
    - Except multi-oom
    - No, we will not give you solutions
- Exam 2 in two weeks (Wednesday 4/8)
  - UTC 2.122A 7p-9p

Interrupt Handling





Interrupt processing involves taking the interrupt, running the interrupt handler, and returning to the user program.

More About Interrupts

To handle an interrupt:

- Hardware saves program counter, etc.
- Hardware loads new program counter from *interrupt vector*
- Assembly language procedure saves registers
- Assembly language procedure sets up new stack
- C interrupt service routine runs (for input, would read and buffer it)
- Scheduler decides which process is to run next
- C procedure returns to the assembly code
- Assembly language procedure starts up a new current process

Interrupts: Precise

Precise interrupts have four properties:

- The program counter is saved in a known place
- All instructions before the one pointed to by the PC have fully executed
- No instruction after the one pointed to by the PC has been executed
- The execution state of the instruction pointed to by the PC is known

Interrupts: Imprecise

- Any interrupt that is not precise (no, really, it's true…)
- Can occur on any architecture that:
  - has a pipeline
  - has out-of-order execution
  - is superscalar
- On these architectures, the PC does not reflect execution
- But the process must still be saved and restored
- Hardware provides lots of internal state to the OS, but the OS must sort out what has and hasn't happened (ugh!)

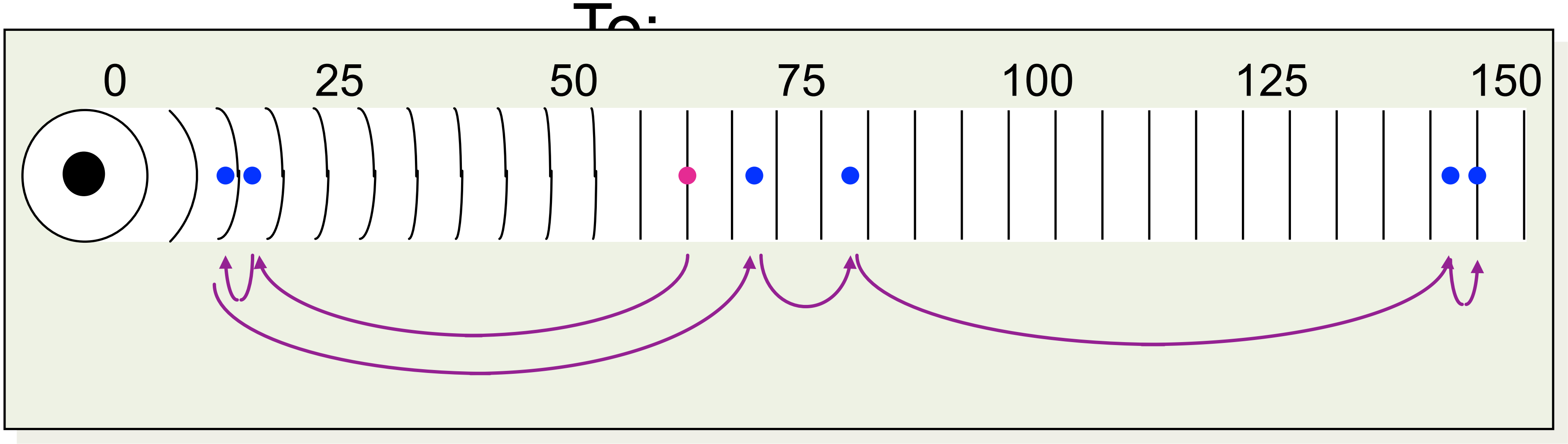
Disk Head Scheduling: SCAN

Rearrange queue
from:

83	72	14	147	16	150

To:

150	147	83	72	14	16



SCAN/Elevator scheduling: Move the head in one direction until all requests have been serviced and then reverse.

Moves the head 187 tracks