Storage and I/O Management

Contents

- I/O Hardware
- Disk-scheduling algorithms
- Error handling

I/O Hardware

- Interrupts
- Device Drivers
- I/O Controllers
- DMA
- Bus

Long-term Information Storage

Three essential requirements:

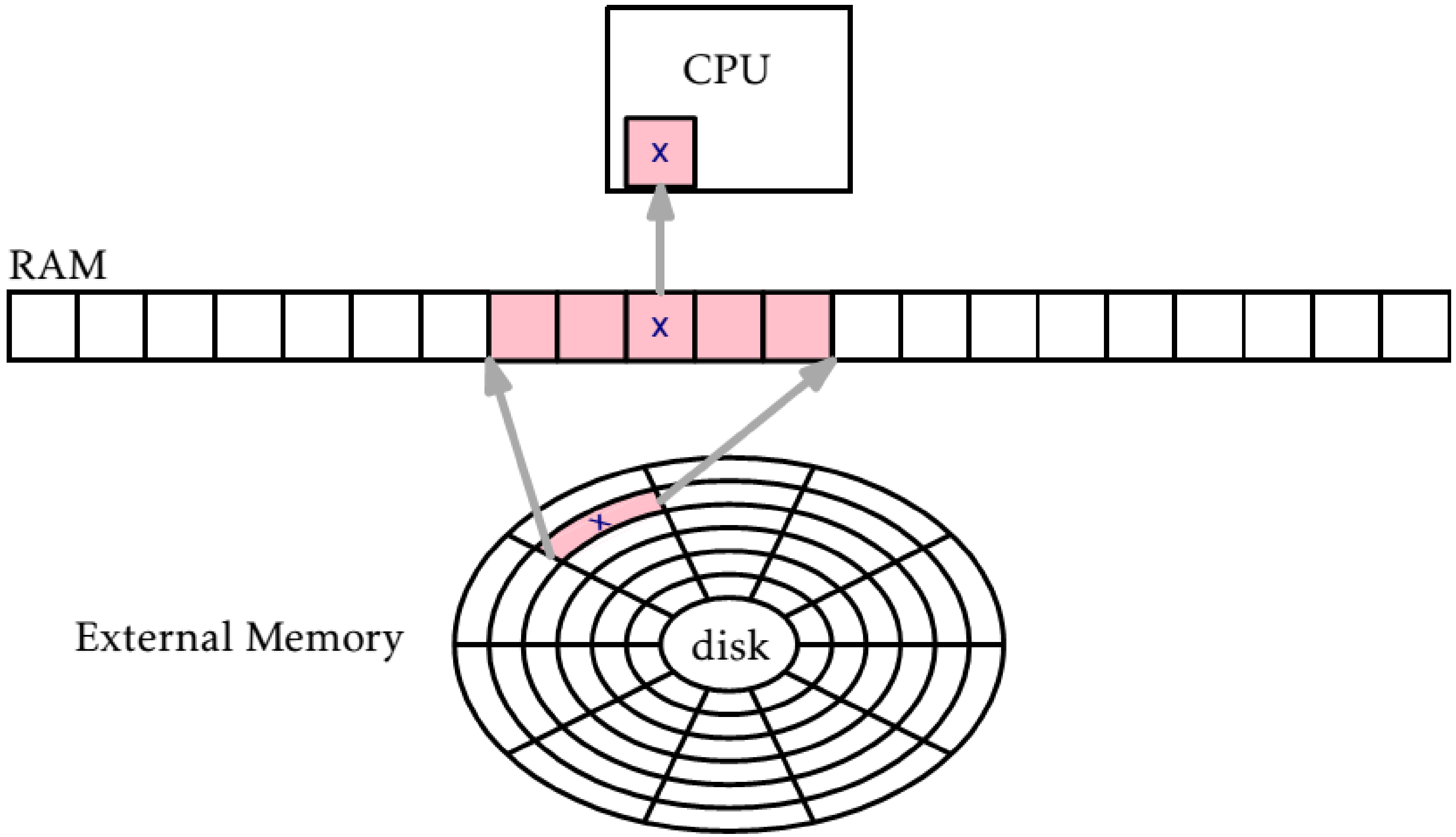
- ❑ Must store large amounts of data
- ❑ Information stored must survive the termination of the process using it (persistence)
- ❑ Multiple processes must be able to access the information concurrently

The First Commercial Disk Drive

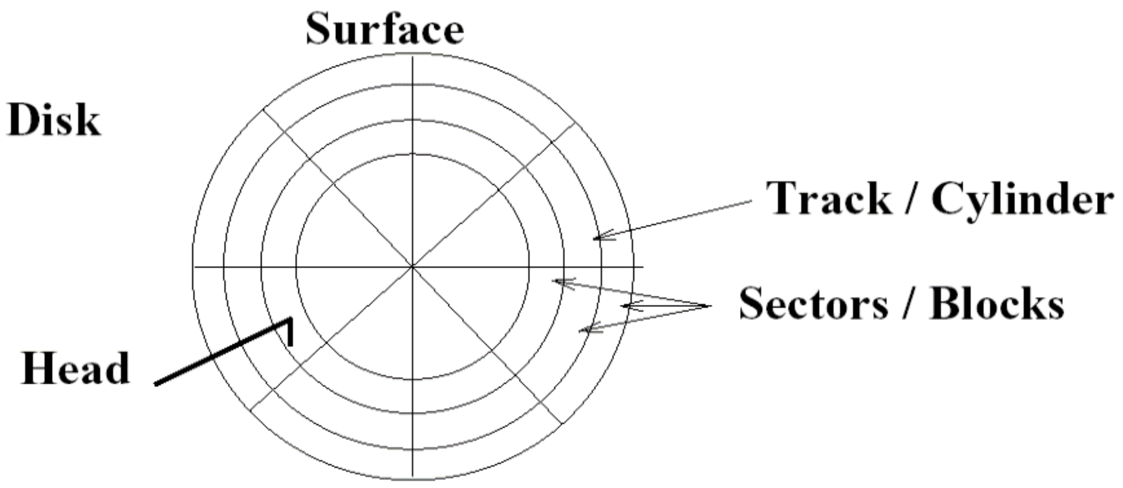
1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

External Memory Model



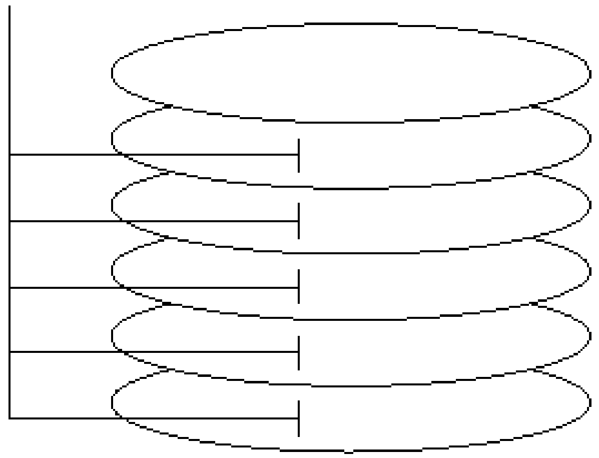
Disk Surface Layout



- Tracks: concentric rings on disk
  - bits laid out serially on tracks
- Tracks split into sectors
  - Addressable unit

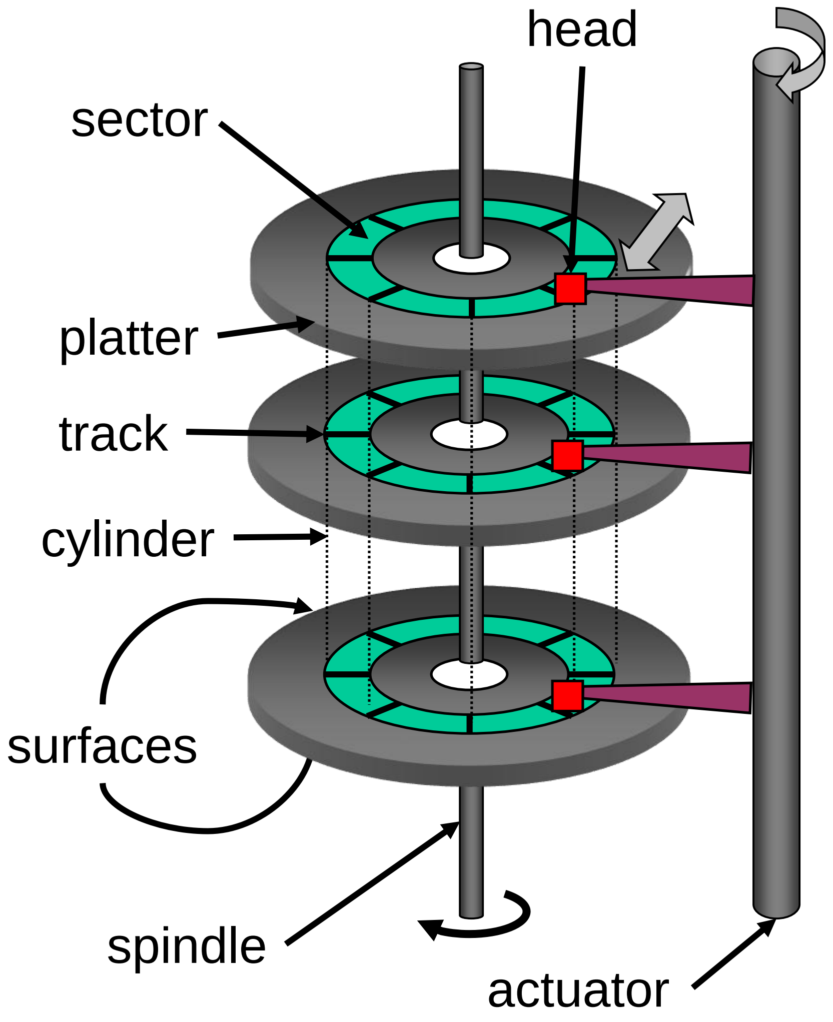
Disk Pack: Multiple Disks

- Disks organized in disk pack = stack of platters
- Use both sides of platters
- Two read-write heads at end of each arm
- Cylinders = matching sectors on each surface

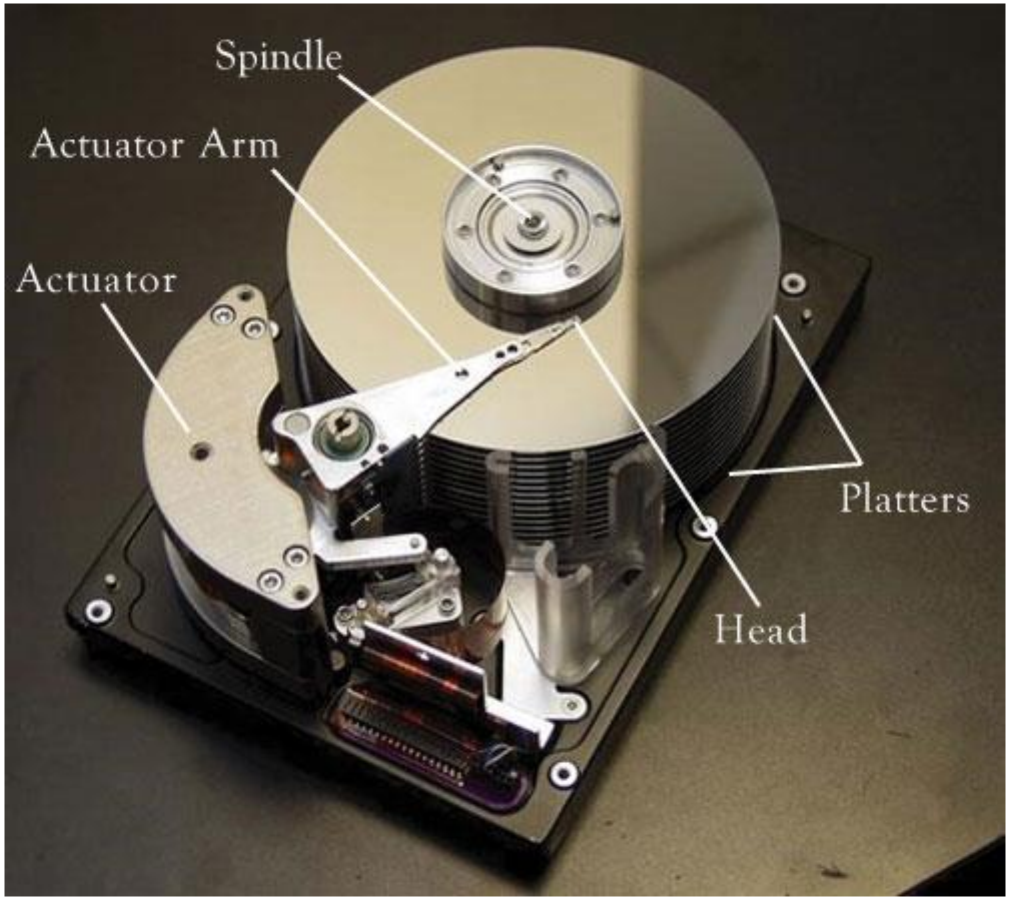


Disk drive structure

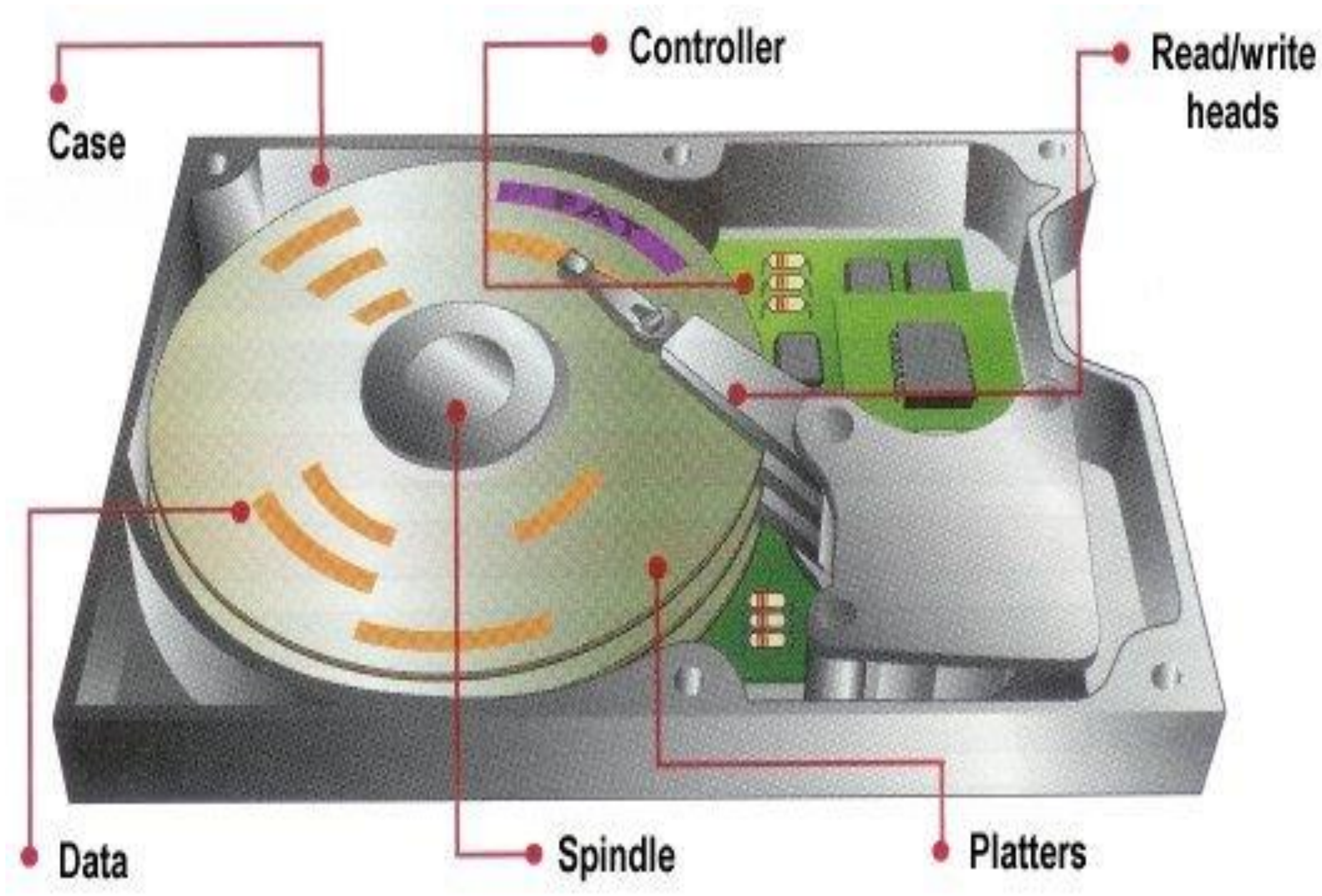
- Data stored on surfaces
  - Up to two surfaces per platter
  - One or more platters per disk
- Data in concentric tracks
  - Tracks broken into sectors
    - 256B-1KB per sector
  - Cylinder: corresponding tracks on all surfaces
- Data read and written by heads
  - Actuator moves heads
  - Heads move in unison



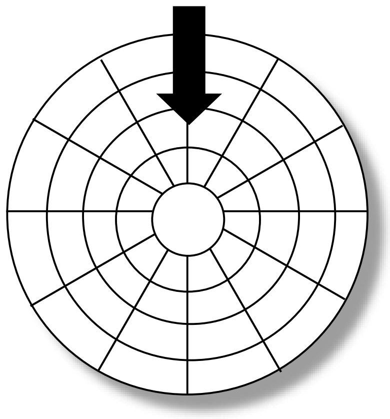
Disk drive structure (cont.)



Disk drive structure (cont.)

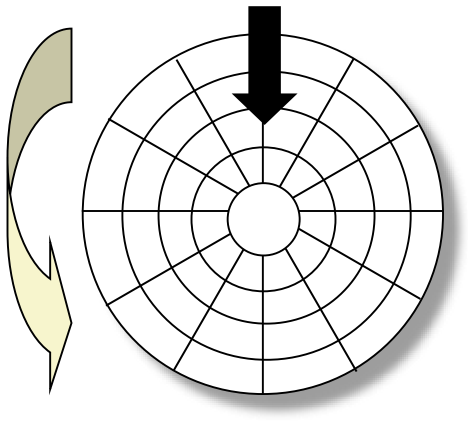


Disk Access



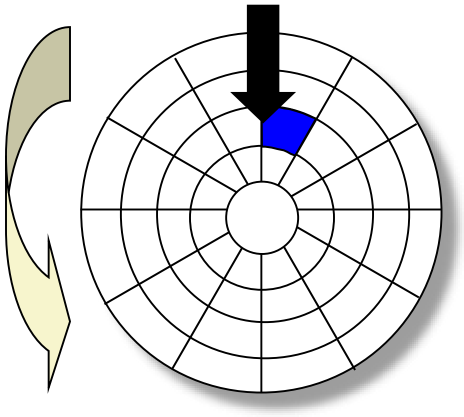
Head in position above a track

Disk Access



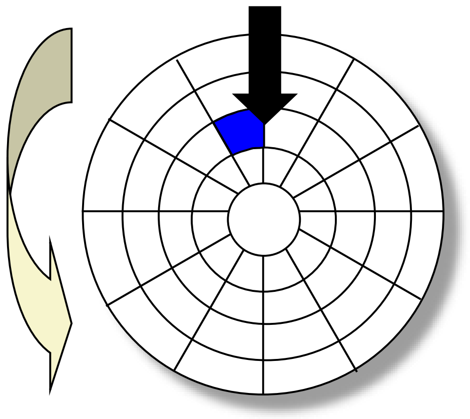
Rotation is counter-clockwise

Disk Access – Read



About to read blue sector

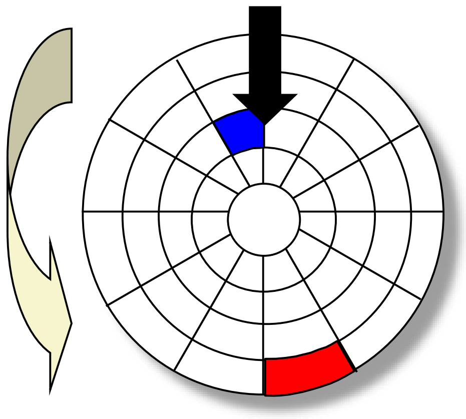
Disk Access – Read



After BLUE read

After reading blue sector

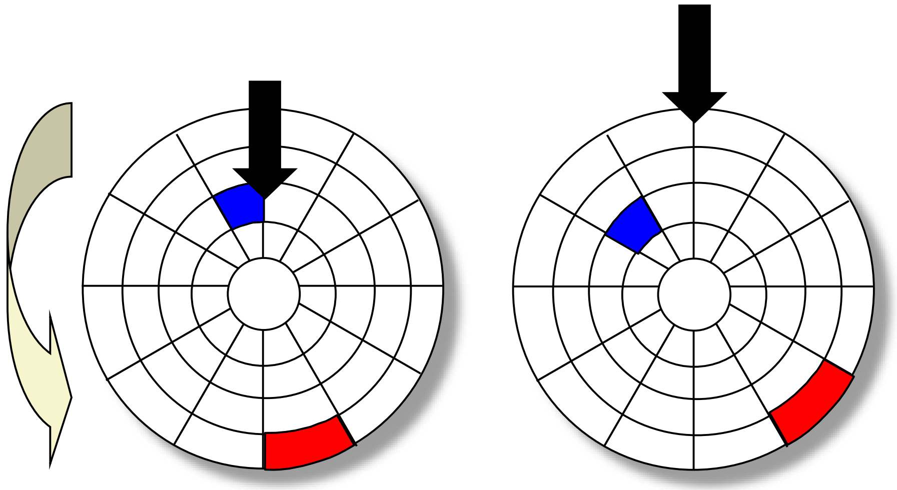
Disk Access - Read



After BLUE read

Red request scheduled next

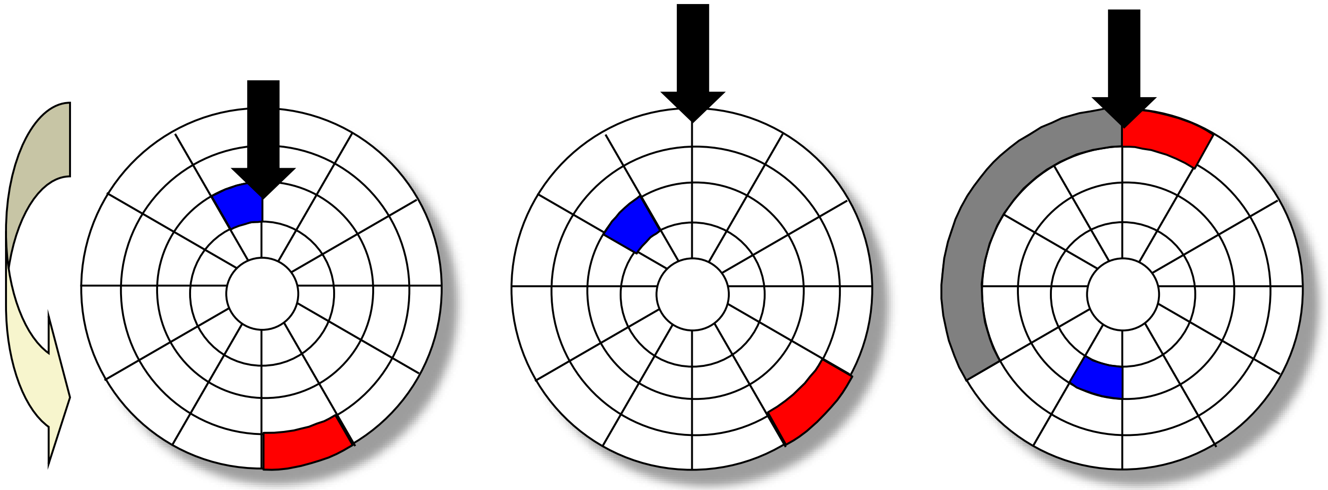
Disk Access - Seek



After BLUE read Seek for RED

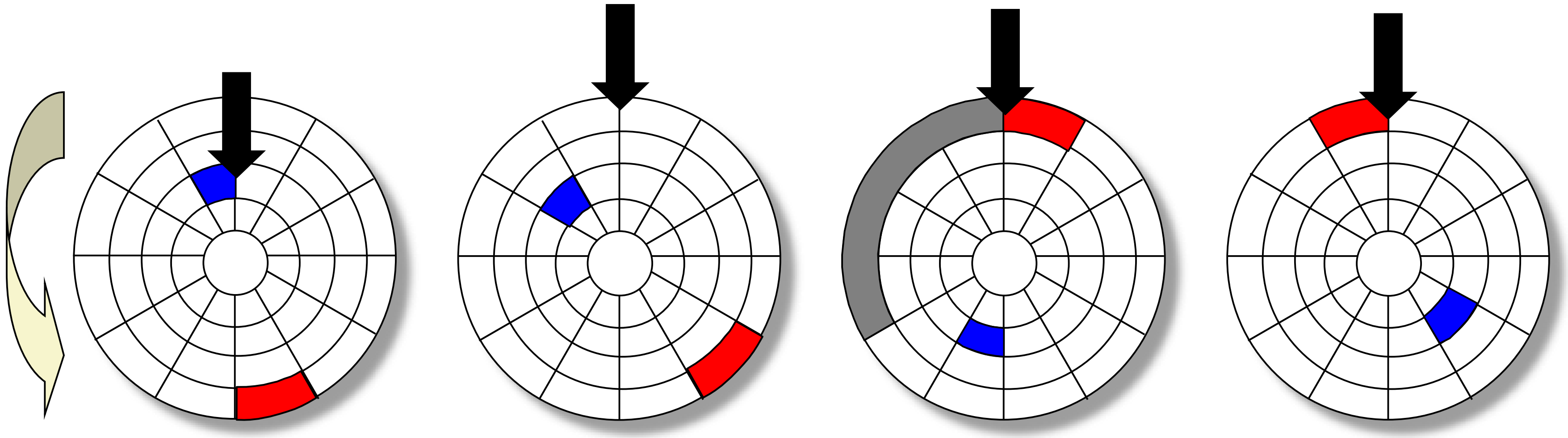
Seek to red's track

Disk Access – Rotational Latency



Wait for red sector to rotate around

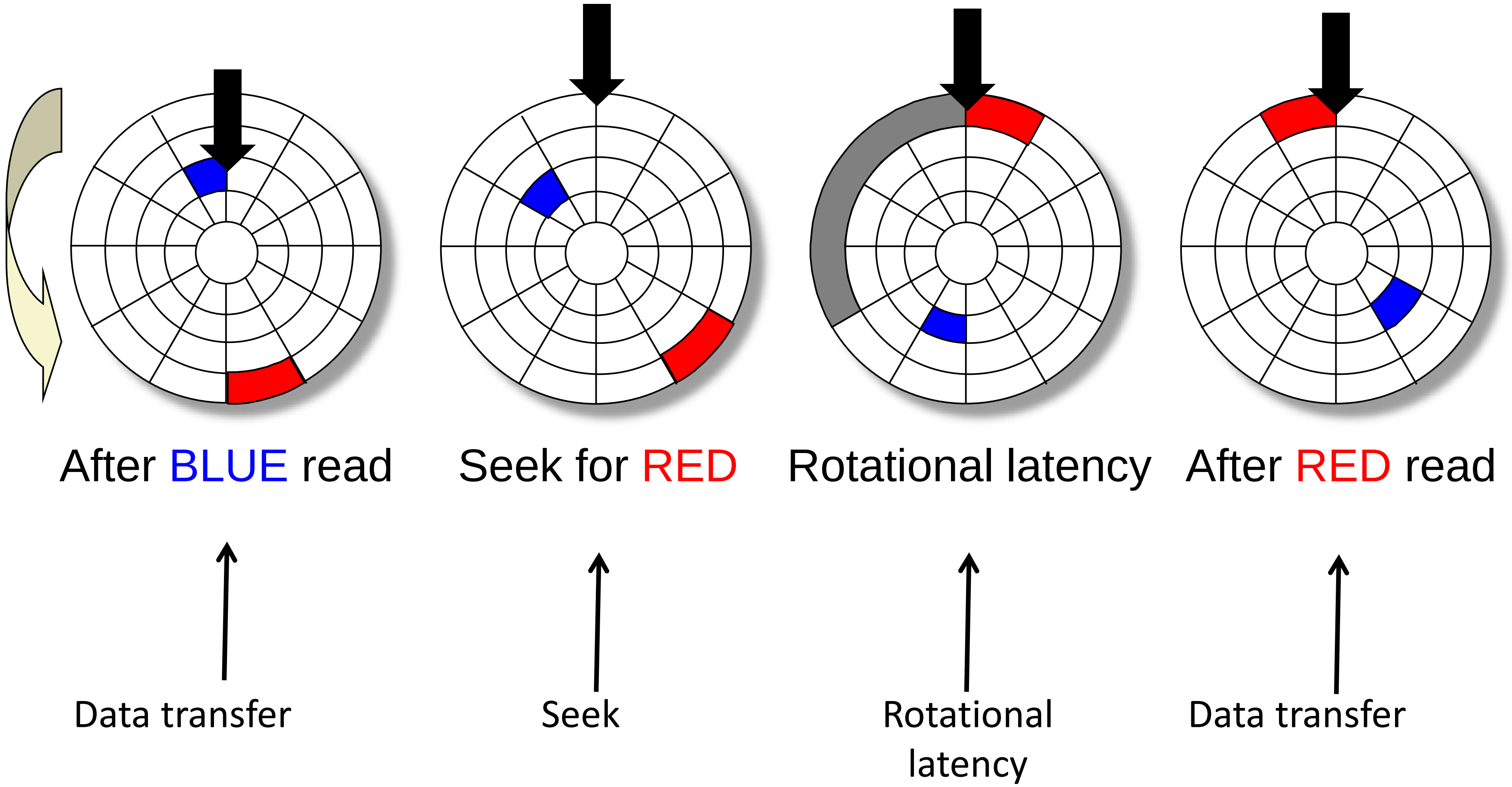
Disk Access - Read



After BLUE read   Seek for RED   Rotational latency   After RED read

Complete read of red

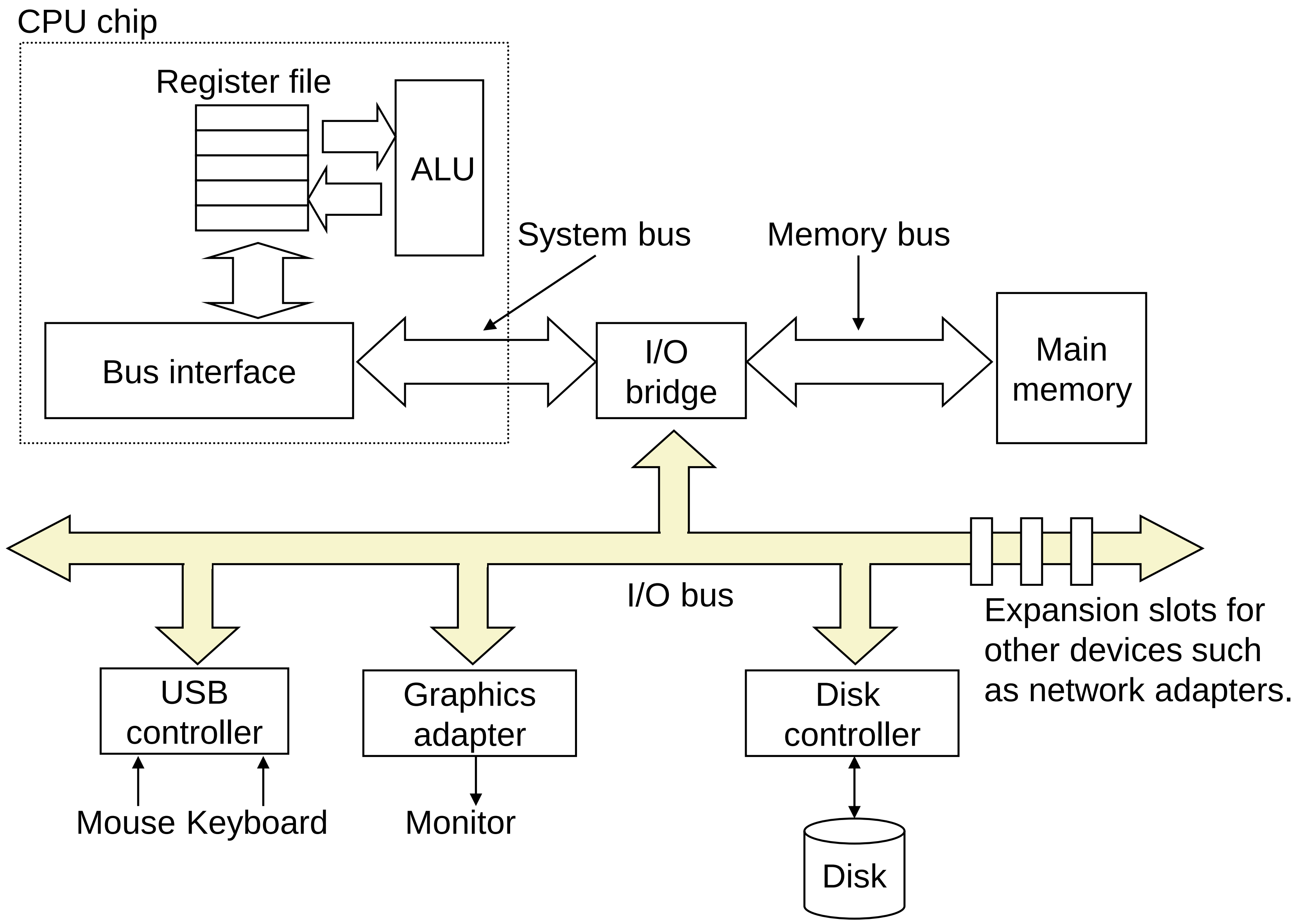
Disk Access – Service Time Components



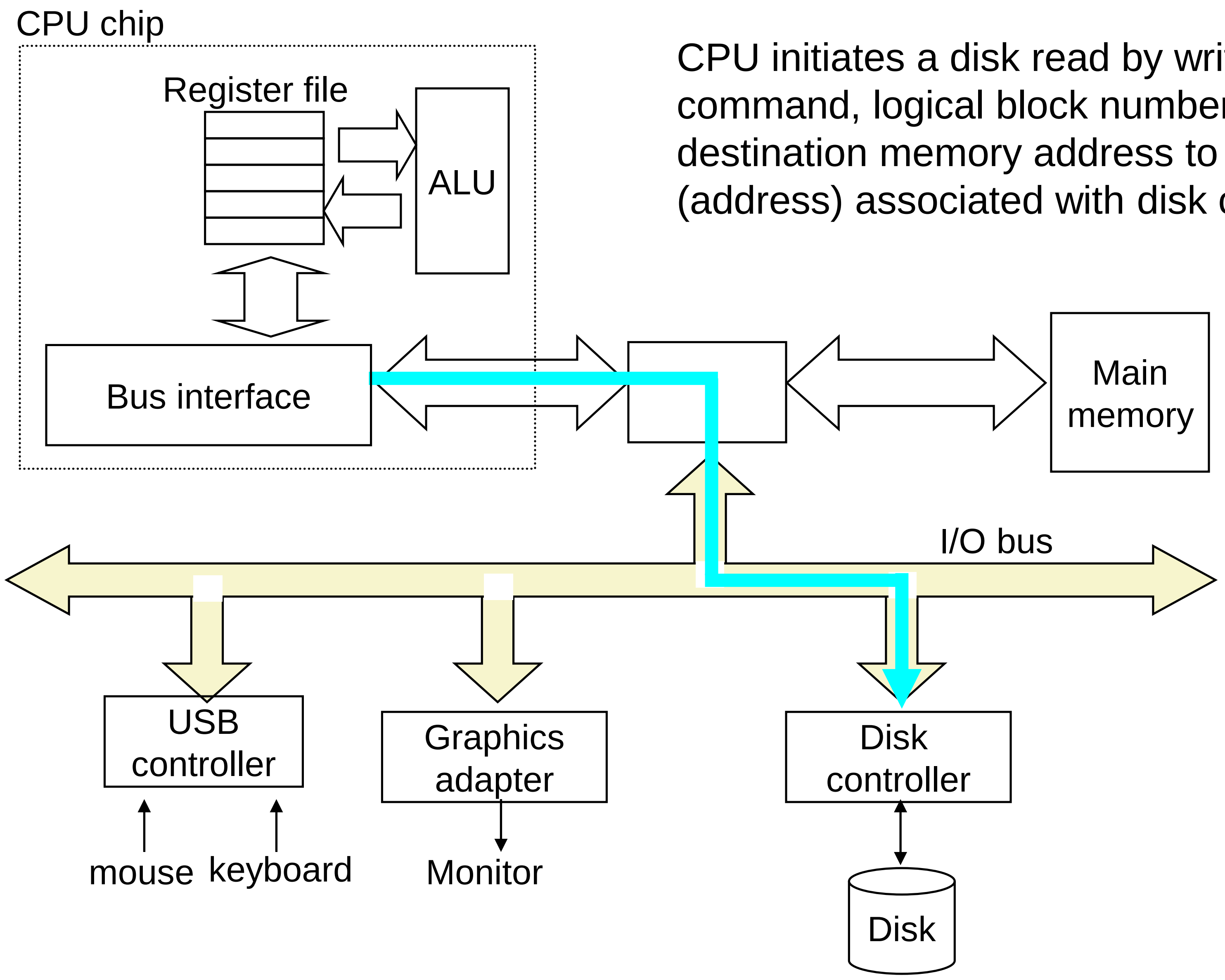
Cost of Disk Operations

- Latency: The time to initiate disk transfer
  - Seek time: The time to position head over correct cylinder
  - Rotational time: The time for correct sector to rotate under disk head
- Usually, the seek time dominates the other two times
- Reducing seek time can improve system performance substantially
- Note: The transfer time for consecutive sectors within a track can be very fast
  - Thus, reading more data than requested and caching it in memory can be very effective in speeding disk accesses

I/O Bus


CPU chip
Register file
ALU
System bus
Memory bus
Bus interface
I/O bridge
Main memory
I/O bus
Expansion slots for other devices such as network adapters.
USB controller
Graphics adapter
Disk controller
Mouse
Keyboard
Monitor
Disk

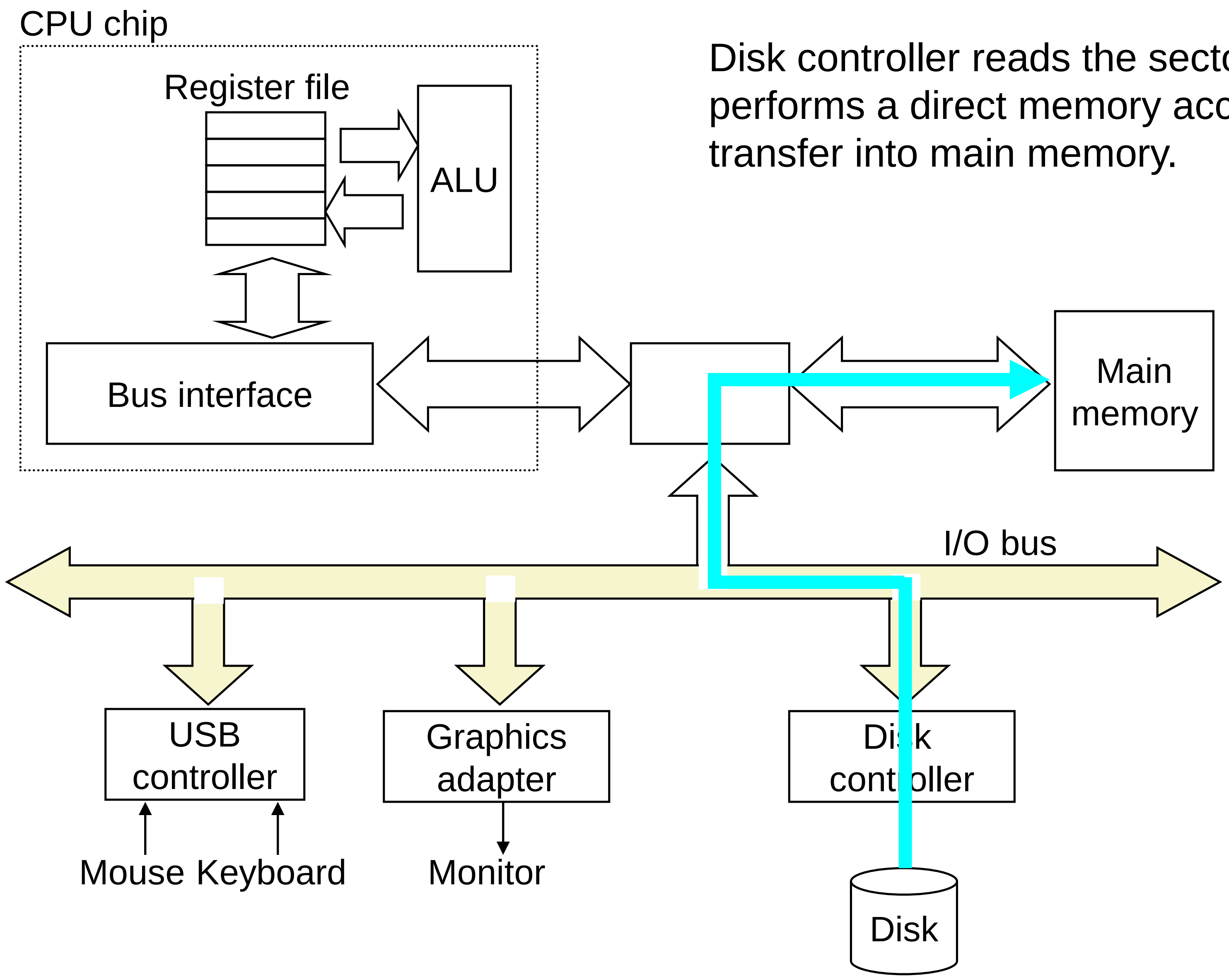
Reading a Disk Sector (1)



CPU initiates a disk read by writing a command, logical block number, and destination memory address to a port (address) associated with disk controller.

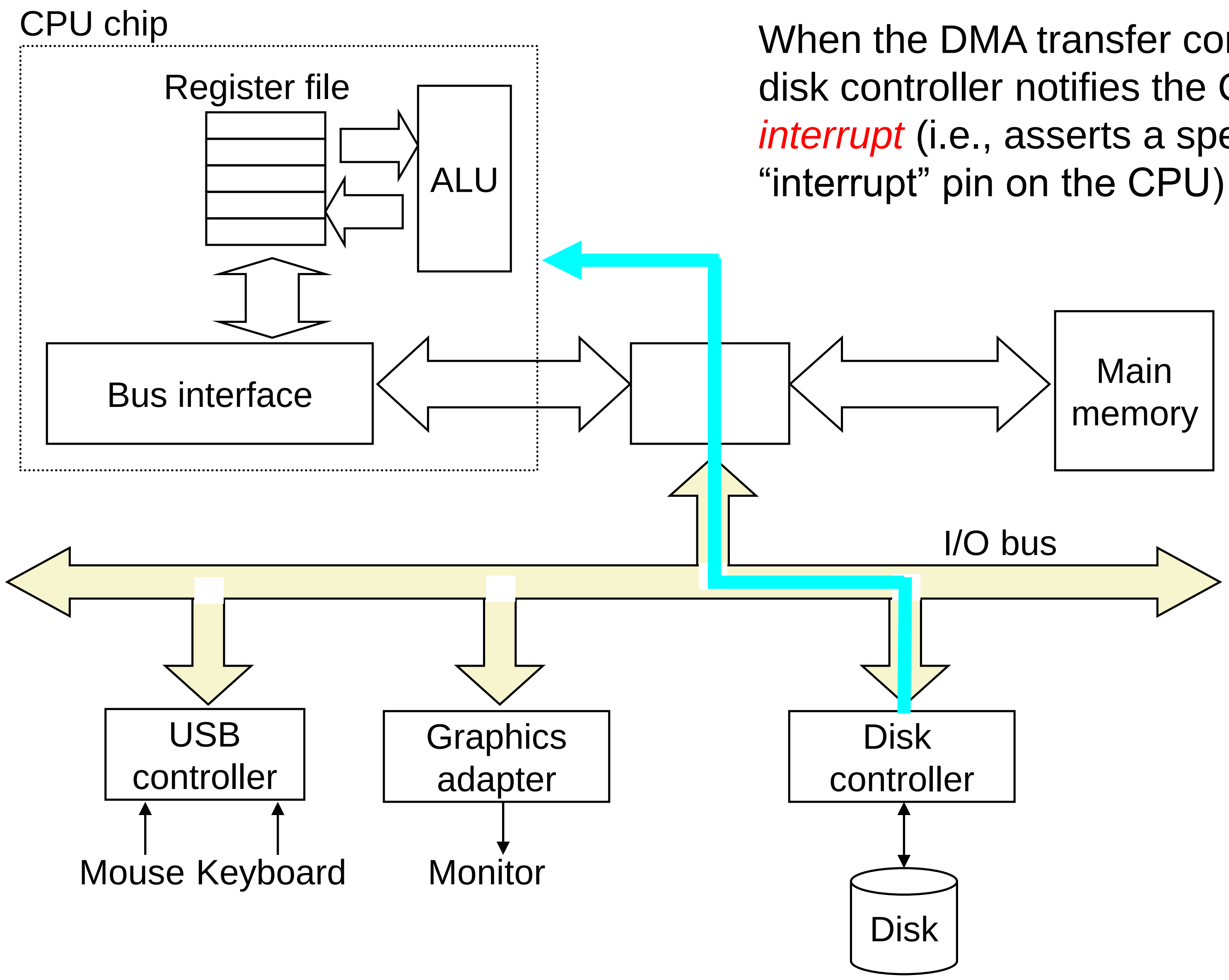
Reading a Disk Sector (2)

Disk controller reads the sector and performs a direct memory access (DMA) transfer into main memory.



Reading a Disk Sector (3)

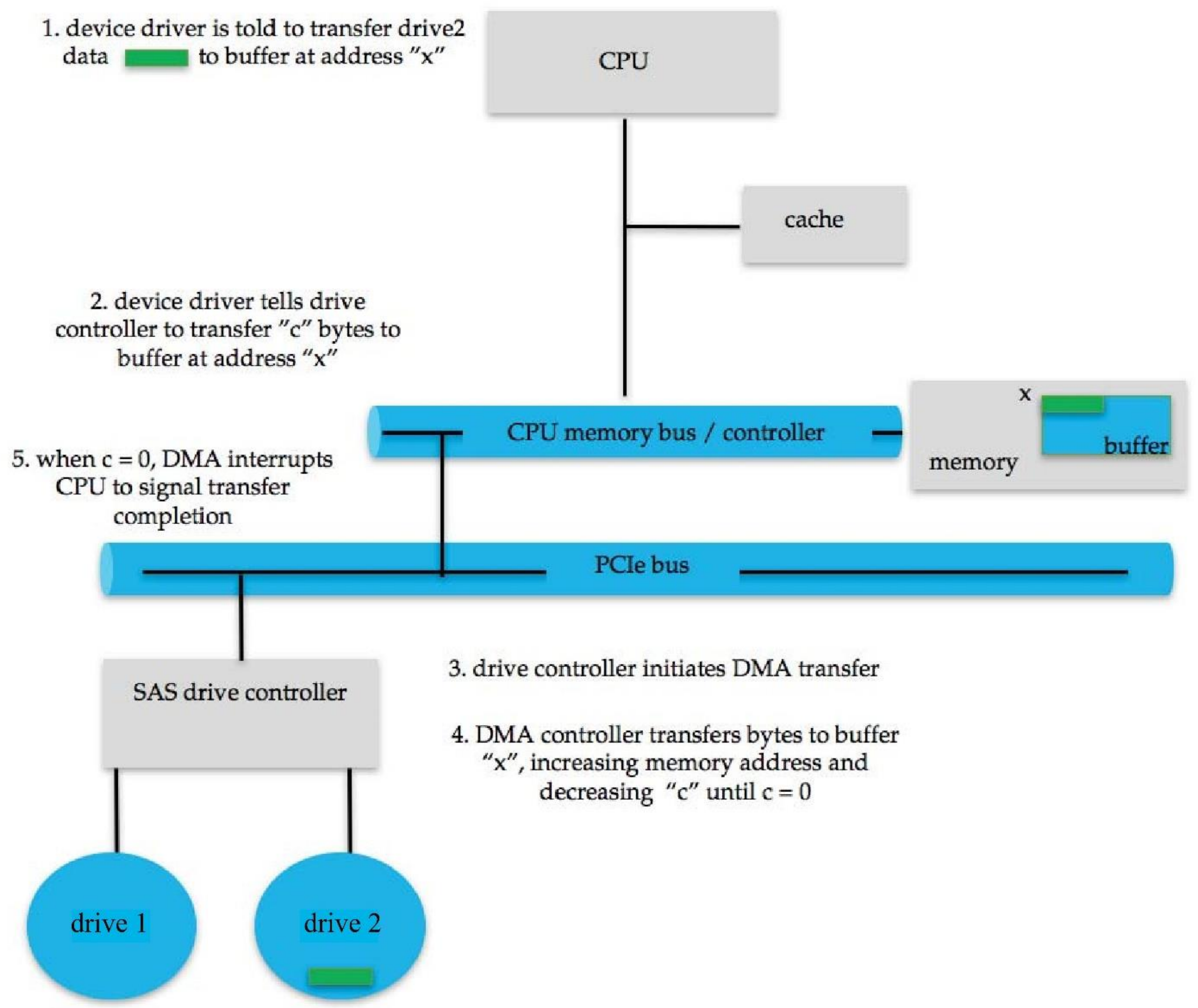
When the DMA transfer completes, the disk controller notifies the CPU with an *interrupt* (i.e., asserts a special "interrupt" pin on the CPU)



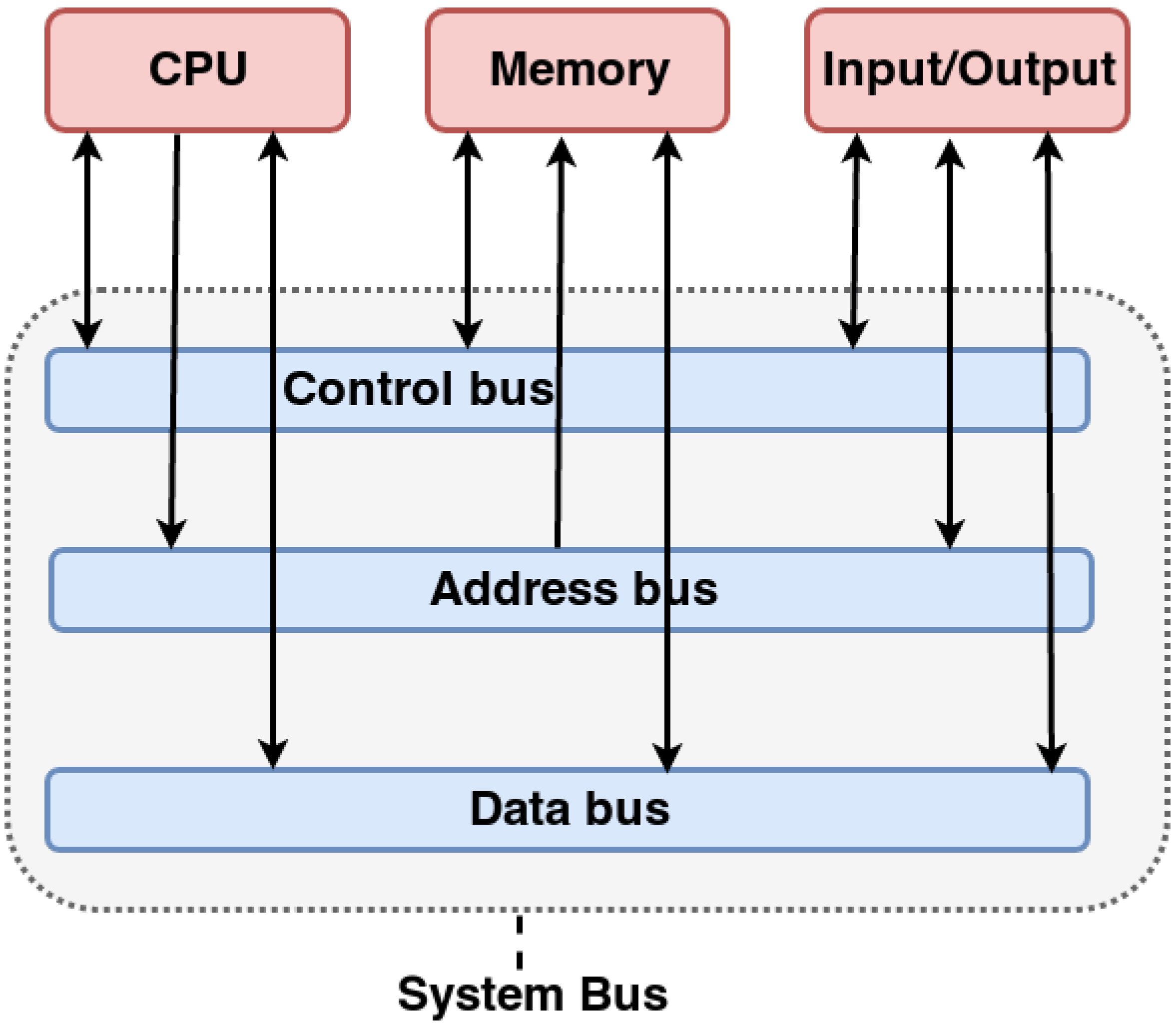
Direct Memory Access

- Used to avoid programmed I/O (one byte at a time) for large data movement
- DMA controller
- Bypasses CPU to transfer data directly between I/O device and memory
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Count of bytes
  - Write location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
    - Cycle stealing from CPU but still much more efficient
  - When done, interrupts to signal completion

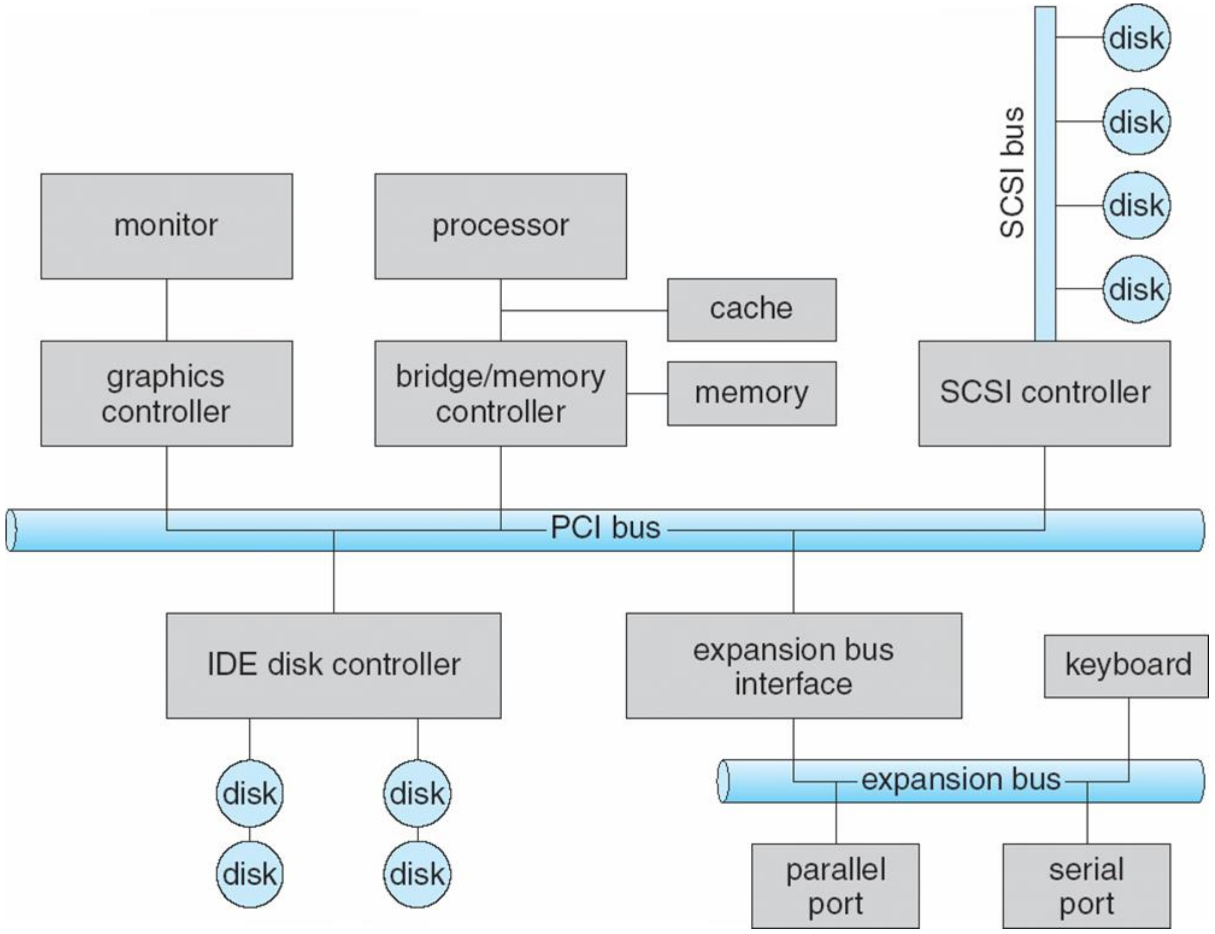
DMA Transfer


1. device driver is told to transfer drive2 data 🟩 to buffer at address "x"
CPU
cache
2. device driver tells drive controller to transfer "c" bytes to buffer at address "x"
x
CPU memory bus / controller
buffer
memory
5. when c = 0, DMA interrupts CPU to signal transfer completion
PCIe bus
SAS drive controller
3. drive controller initiates DMA transfer
4. DMA controller transfers bytes to buffer "x", increasing memory address and decreasing "c" until c = 0
drive 1
drive 2

I/O Busses


CPU
Memory
Input/Output
Control bus
Address bus
Data bus
System Bus

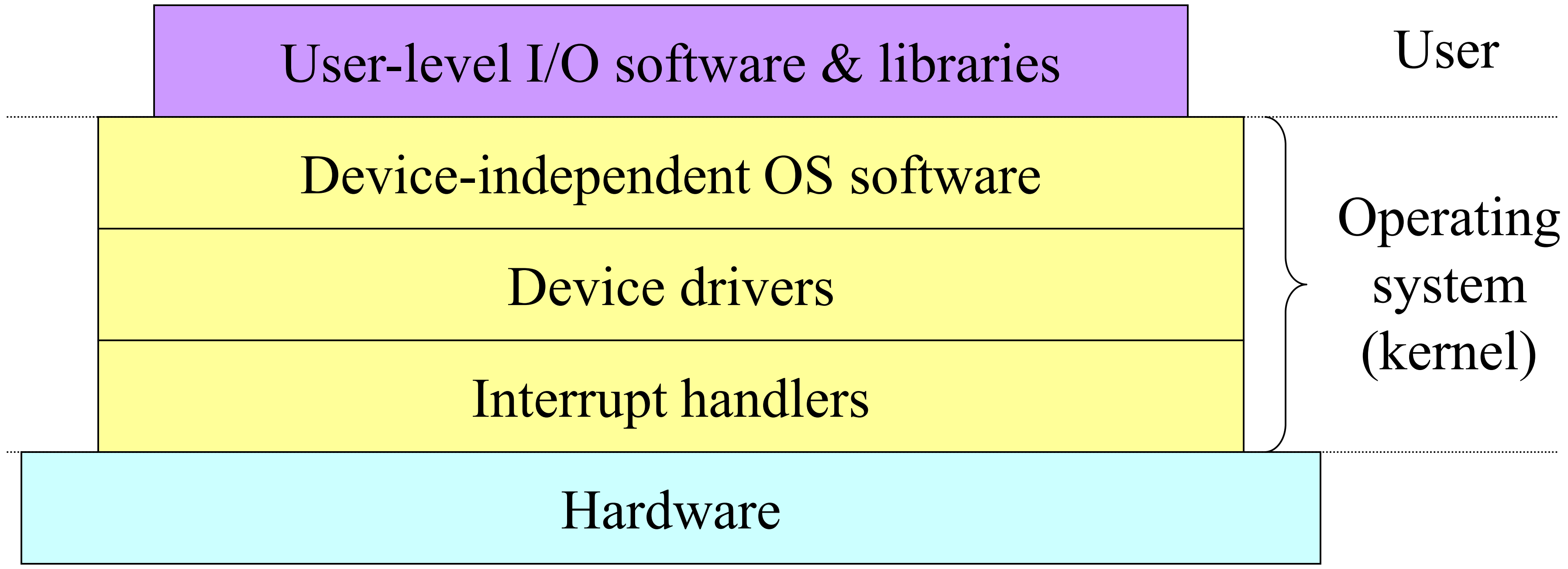
A Typical PC Bus Structure



I/O Busses

- A disk drive is attached to a computer by a set of wires called an I/O bus.
- Types of busses available:
  - Integrated Drive Electrics (IDE)
  - Advanced Technology Attachment (ATA)
  - Serial ATA (SATA)
  - Universal serial bus (USB)
  - Small computer-systems interface (SCSI)

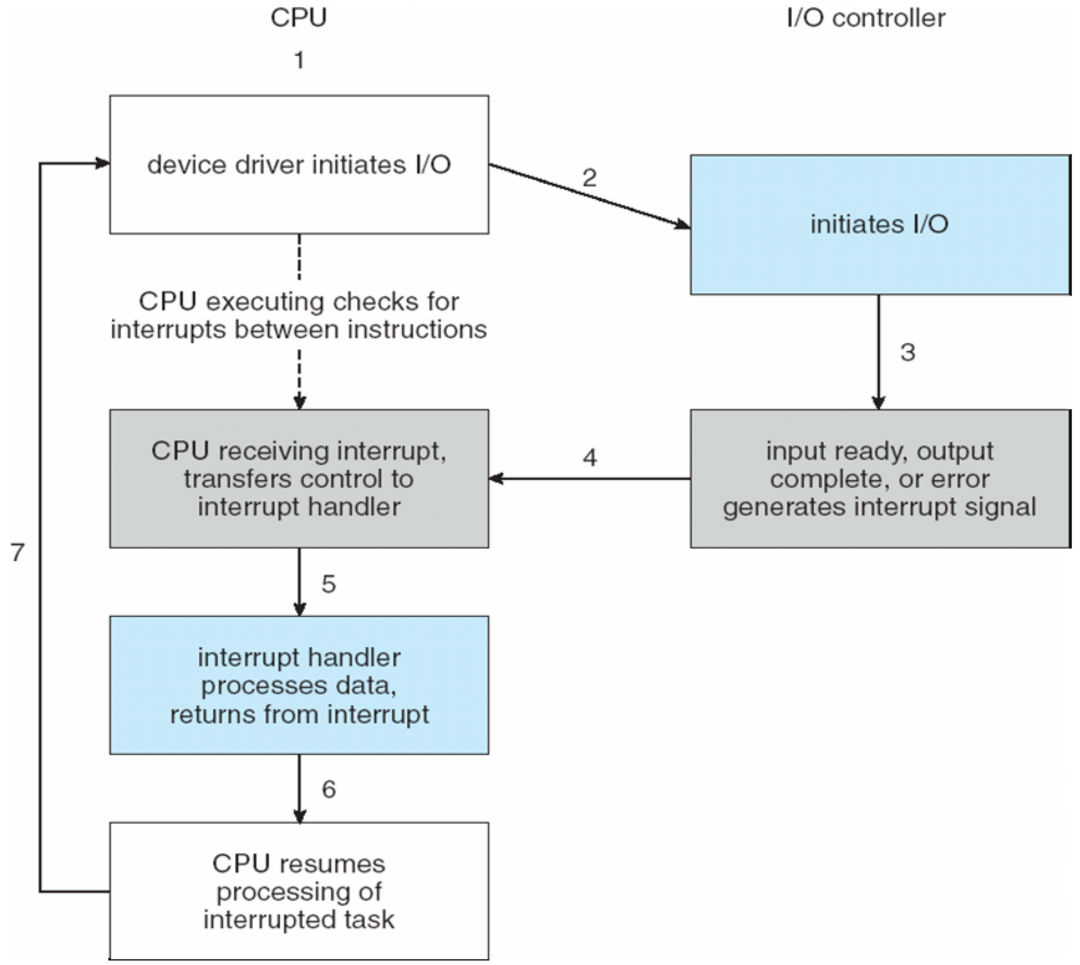
Layers of I/O software


User-level I/O software & libraries
Device-independent OS software
Device drivers
Interrupt handlers
Hardware
User
Operating system (kernel)

Interrupt handlers

- CPU Interrupt-request line triggered by I/O device.
  - Checked by processor after each instruction
- Interrupt handler receives interrupts
  - Maskable to ignore of delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Context switch at start and end
  - Based on priority
  - Some nonmaskable
  - Interrupt chaining if more than one device at same interrupt number.

Interrupt-Driven I/O Cycle


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
7
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task

Intel Event-Vectors

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Interrupt handlers

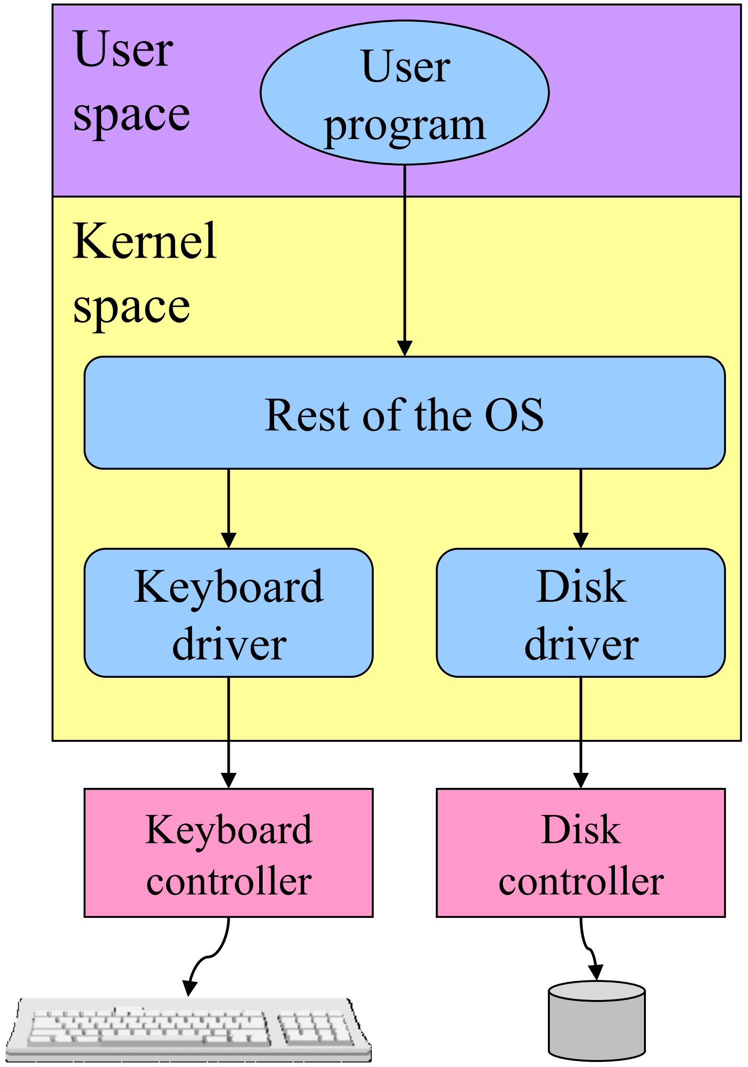
- Interrupt handlers are best hidden
  - Driver starts an I/O operation and blocks
  - Interrupt notifies of completion
- Interrupt procedure does its task
  - Then unblocks driver that started it
  - Perform minimal actions at interrupt time
    - Some of the functionality can be done by the driver after it is unblocked
- Interrupt handler must
  - Save regs not already saved by interrupt hardware
  - Set up context for interrupt service procedure

What happens on an interrupt

- Set up stack for interrupt service procedure
- Ack interrupt controller, reenable interrupts
- Copy registers from where saved
- Run service procedure
- (optional) Pick a new process to run next
- Set up MMU context for process to run next
- Load new process' registers
- Start running the new process

Device drivers

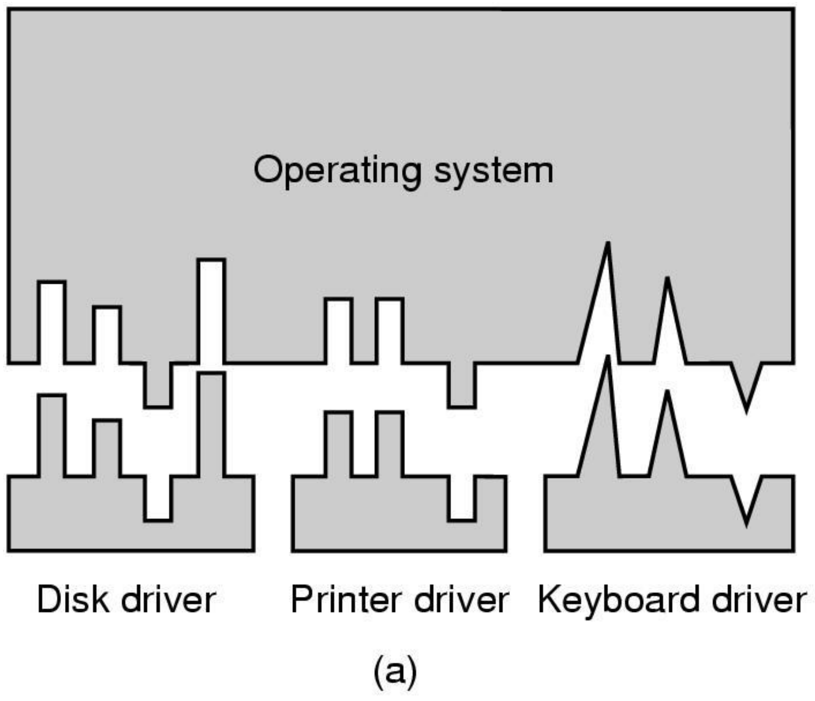
- Device drivers go between device controllers and rest of OS
  - Drivers standardize interface to widely varied devices
- Device drivers communicate with controllers over bus
  - Controllers communicate with devices themselves

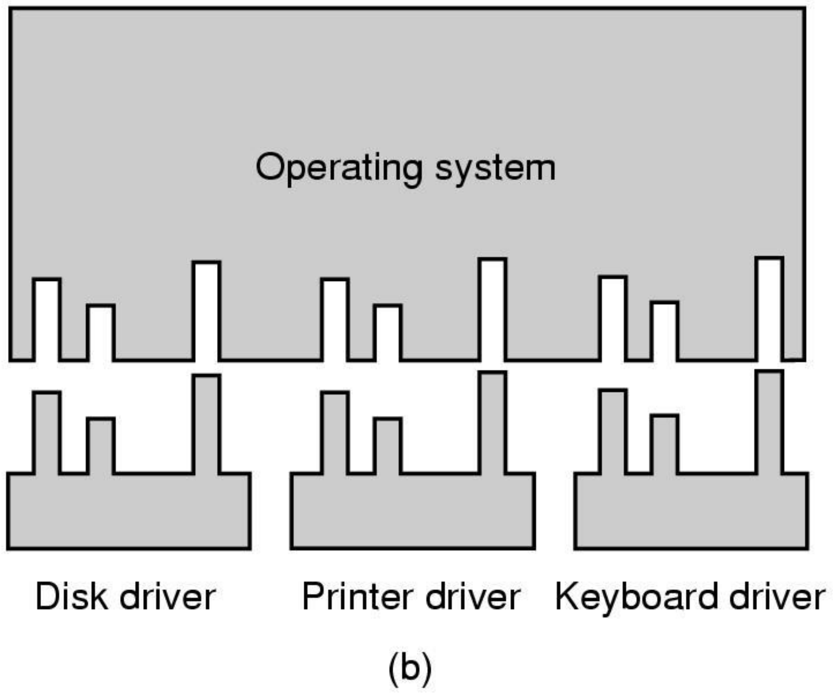


Device-independent I/O software

- Device-independent I/O software provides common “library” routines for I/O software
- Helps drivers maintain a standard appearance to the rest of the OS
- Uniform interface for many device drivers for
  - Buffering
  - Error reporting
  - Allocating and releasing dedicated devices
  - Suspending and resuming processes
- Common resource pool
  - Device-independent block size (keep track of blocks)
  - Other device driver resources

Why a standard driver interface?

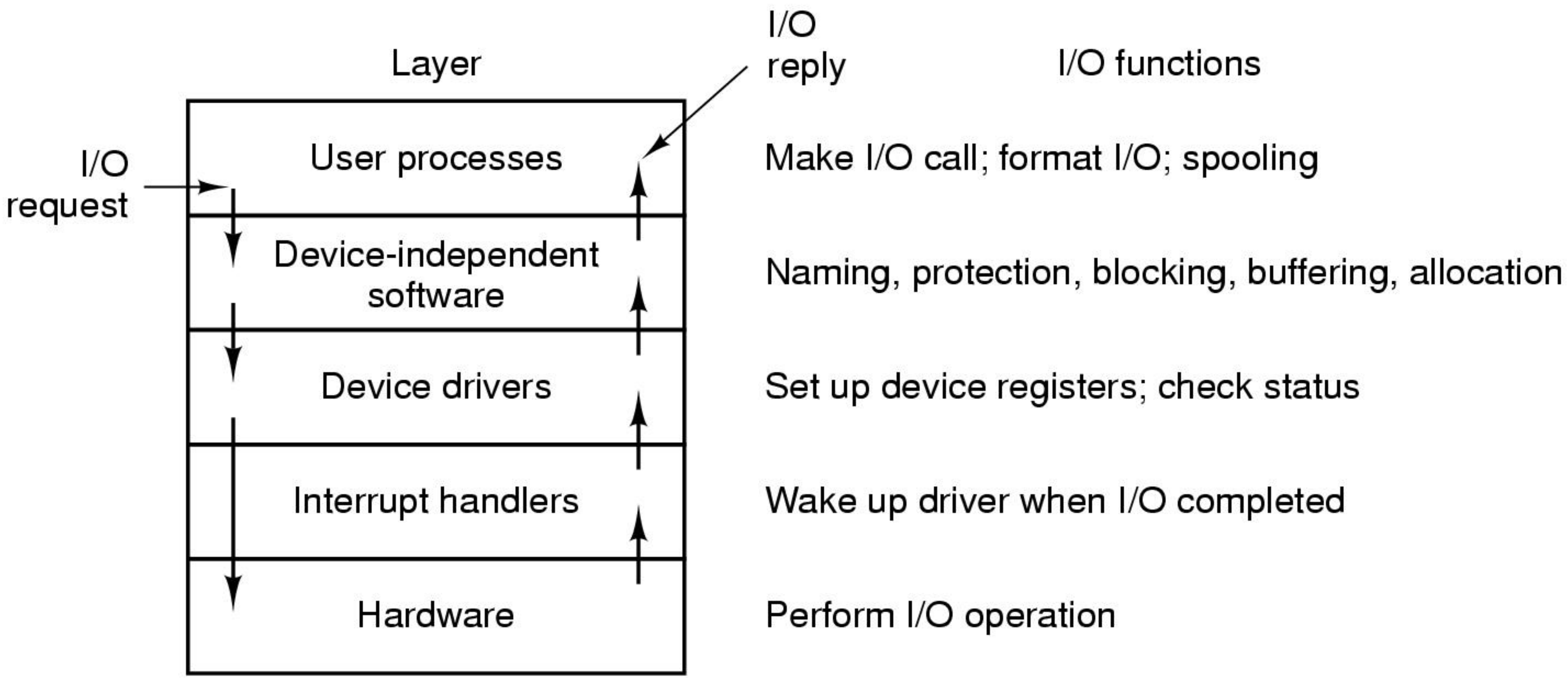




Non-standard driver interfaces

Standard driver interfaces

Anatomy of an I/O request


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

I/O Controllers

- The data transfers on a bus are carried out by special electronic processors called controllers
- The host controller is the controller at the computer end of the bus
- A disk controller is built into each disk drive

I/O Controllers

- To perform a disk I/O operation, the computer places a command into the host controller
- The host controller sends command to the disk controller
- The disk controller operates on the disk-drive hardware to carry out the command
- Disk controllers have caches

Disk Scheduling Algorithms

- FCFS
- SSTF
- SCAN

Issues Handled by Disk Software

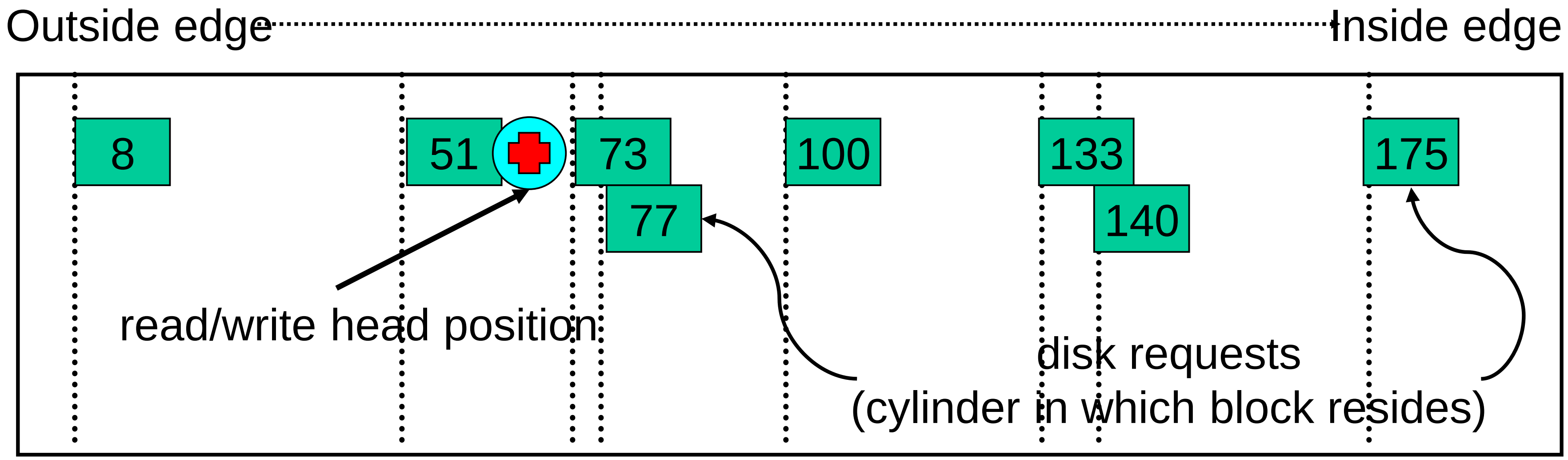
- The disk accepts requests and carries them out
  - What sort of disk arm scheduling algorithm is needed?
- Disks are error prone
  - How are errors handled?

Disk request scheduling

- Goal: use disk hardware efficiently
  - Bandwidth (transmission capacity) as high as possible
  - Disk transferring as often as possible (and not seeking)
- We want to
  - Minimize disk seek time (moving from track to track)
  - Minimize rotational latency (waiting for disk to rotate the desired sector under the read/write head)
- Calculate disk bandwidth by
  - Total bytes transferred / time to service request
  - Seek time & rotational latency are overhead (no data is transferred), and reduce disk bandwidth
- Minimize seek time & rotational latency by
  - Using algorithms to find a good sequence for servicing requests
  - Placing blocks of a given file "near" each other

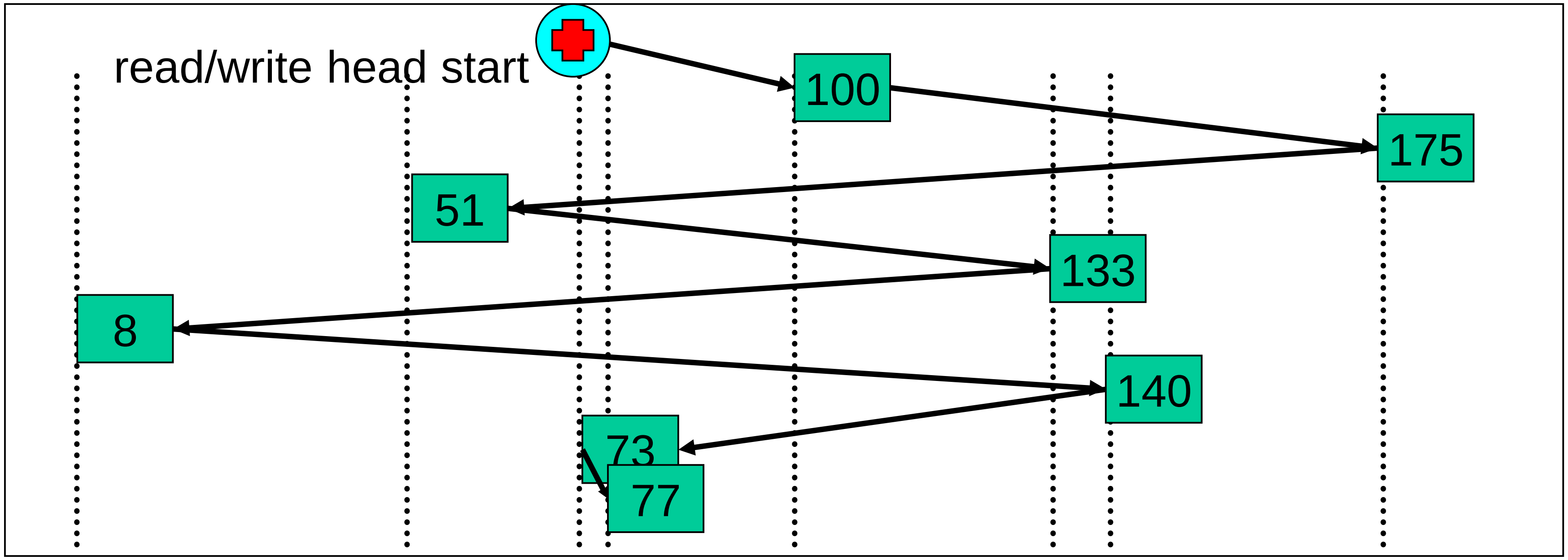
Disk scheduling algorithms

- Schedule disk requests to minimize disk seek time
  - Seek time increases as distance increases (though not linearly)
  - Minimize seek distance -> minimize seek time
- Disk seek algorithm examples assume a request queue & head position (disk has 200 cylinders)
  - Queue = 100, 175, 51, 133, 8, 140, 73, 77
  - Head position = 63



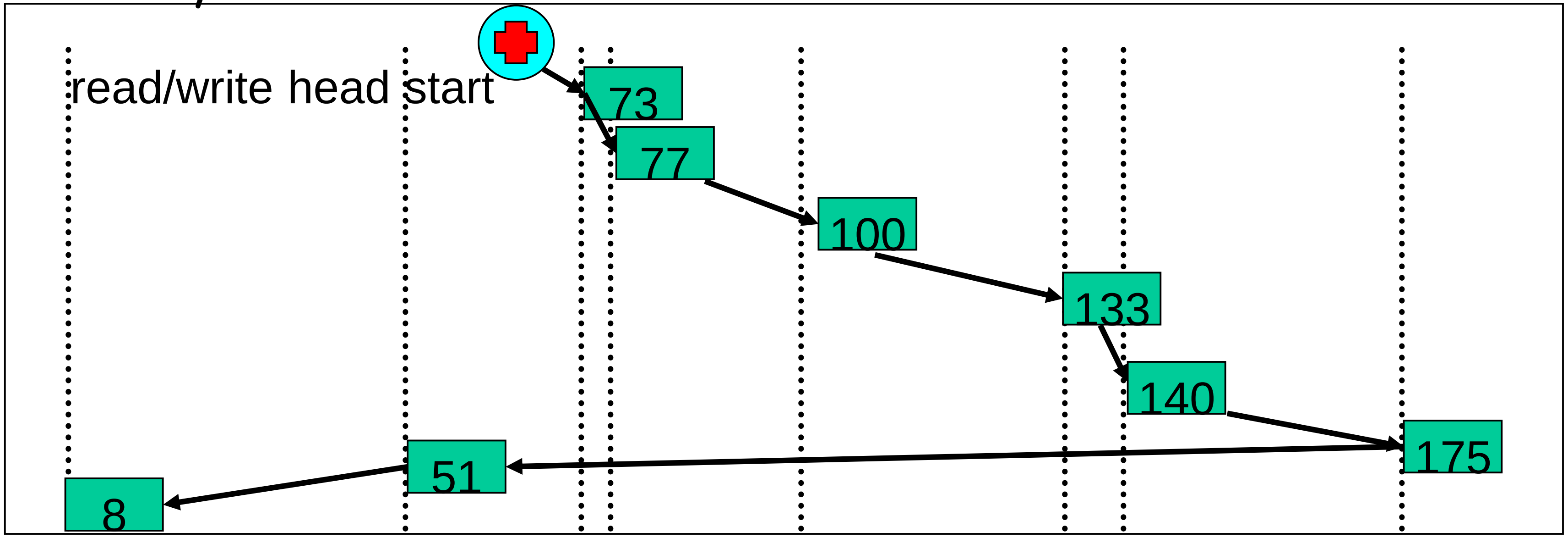
First-Come-First Served (FCFS)

- Requests serviced in the order in which they arrived
  - Easy to implement!
  - May involve lots of unnecessary seek distance
- Seek order = 100, 175, 51, 133, 8, 140, 73, 77
- Seek distance = (100-63) + (175-100) + (175-51) + (133-51) + (133-8) + (140-8) + (140-73) + (77-73) = 646 cylinders



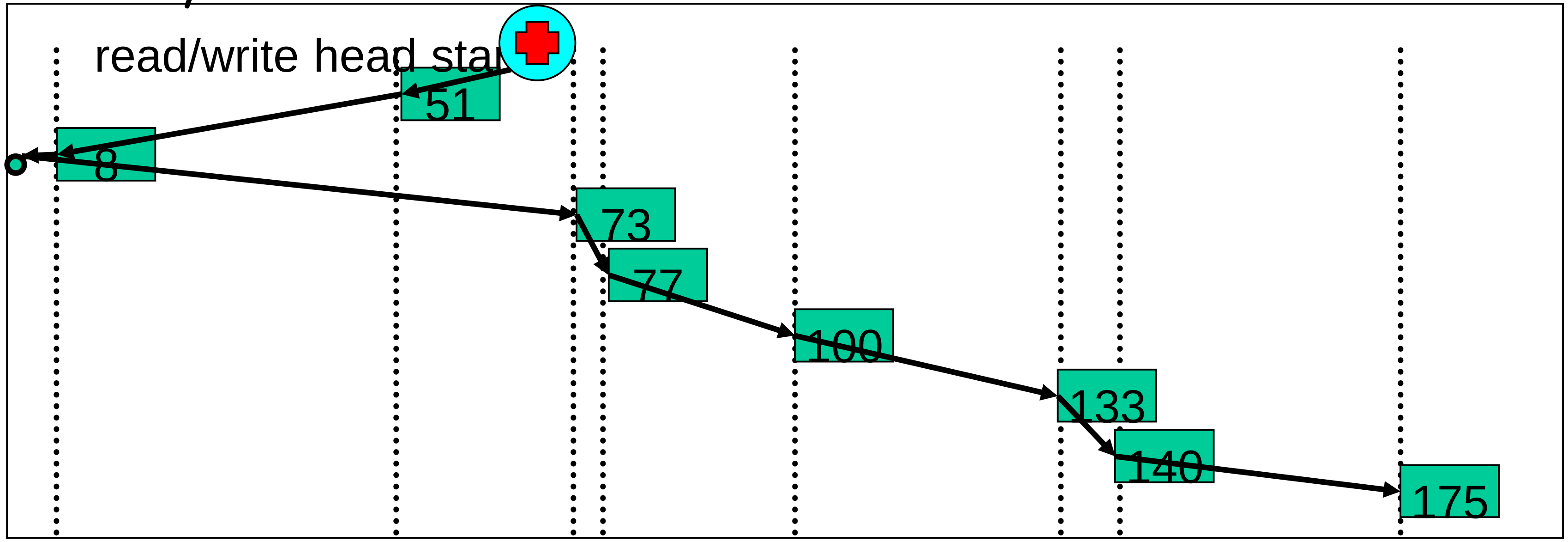
Shortest Seek Time First (SSTF)

- Service the request with the shortest seek time from the current head position
  - Form of SJF scheduling
  - May starve some requests
- Seek order = 73, 77, 100, 133, 140, 175, 51, 8
- Seek distance = 10 + 4 + 23 + 33 + 7 + 35 + 124 + 43 = 279 cylinders



SCAN (elevator algorithm)

- Disk arm starts at one end of the disk and moves towards the other end, servicing requests as it goes
  - Reverses direction when it gets to end of the disk
  - Also known as elevator algorithm
- Seek order = 51, 8, 0 , 73, 77, 100, 133, 140, 175
- Seek distance = 12 + 43 + 8 + 73 + 4 + 23 + 33 + 7 + 35 = 238 cyls

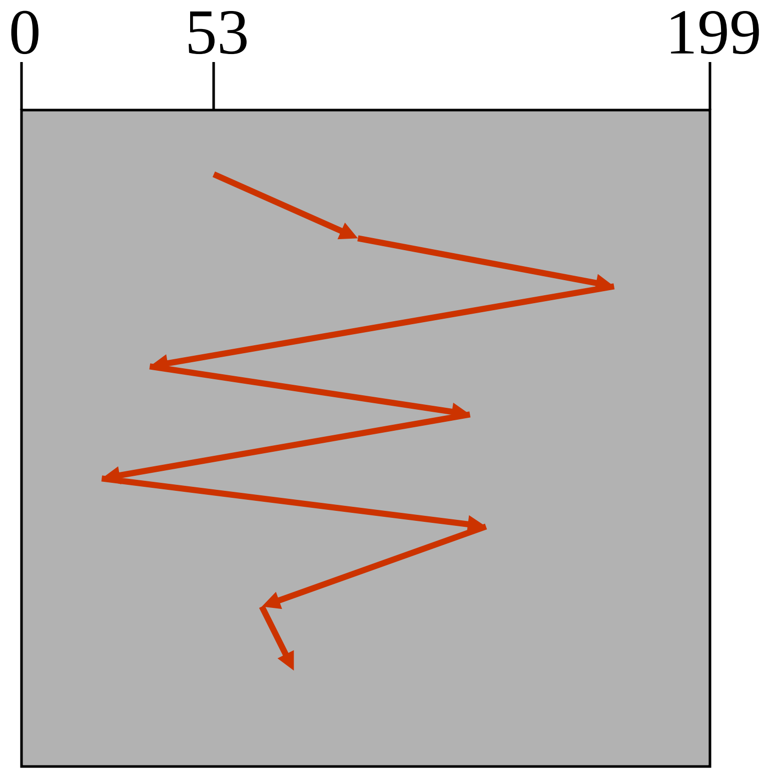


Disk Scheduling: Data Structure

- Software for disks maintain a table called the pending request table
- Table is indexed by cylinder number
- All requests for each cylinder are chained together in a linked list headed by the table entries

First-Come, First-Served (FCFS) order

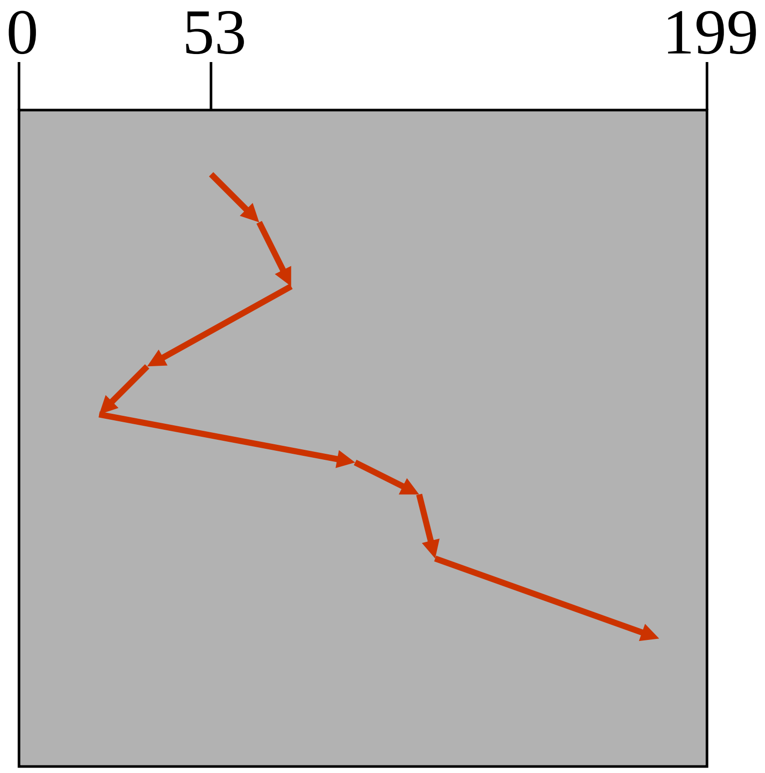
- Method
  - First come first serve
- Pros
  - Fairness among requests
  - In the order applications expect
- Cons
  - Arrival may be on random spots on the disk (long seeks)
  - Wild swing can happen



98, 183, 37, 122, 14, 124, 65, 67

SSTF (Shortest Seek Time First)

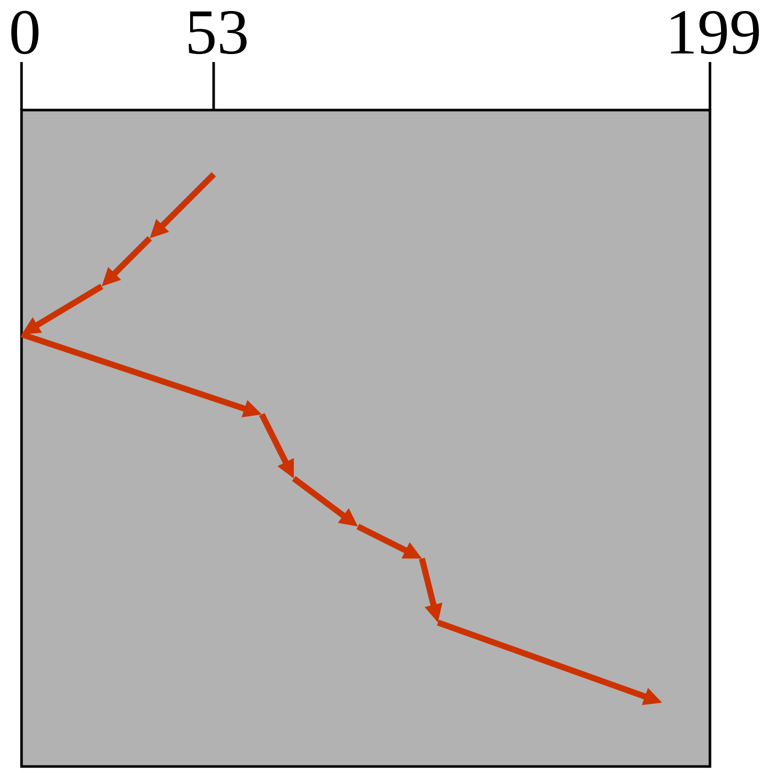
- Method
  - Pick the one closest on disk
  - Rotational delay is in calculation
- Pros
  - Try to minimize seek time
- Cons
  - Starvation
- Question
  - Is SSTF optimal?
  - Can we avoid starvation?



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)

Elevator (SCAN)

- Method
  - Take the closest request in the direction of travel
  - Real implementations do not go to the end (called LOOK)
- Pros
  - Bounded time for each request
- Cons
  - Request at the other end will take a while



98, 183, 37, 122, 14, 124, 65, 67
(37, 14, 65, 67, 98, 122, 124, 183)

Optimization

- Some disk controllers provide a way for the software to inspect the current sector number under the read
- If there are two or more requests for the same cylinder that are pending:
  - The driver can issue a request for the sector that will pass under the head next
  - The information is known from the pending request table
- Caching is needed to hold the additional data

Error Handling

- Types of Error
- Cyclic Redundancy Check (CRC)
- Parity Check
- Checksum Computation
- Codeword and Hamming

Types of Error

- Bit is altered between transmission and reception
- Single-bit error
  - One bit is altered
  - Adjacent bits not affected
  - Caused by White noise
- Burst errors
  - Length B
  - Contiguous sequence of B bits in which first and last bits and any number of intermediate bits are in error
  - Caused by
    - Impulse noise
    - Fading in wireless environment
  - Effects are greater at higher data rates (Why?)

Error Detection (CRC)

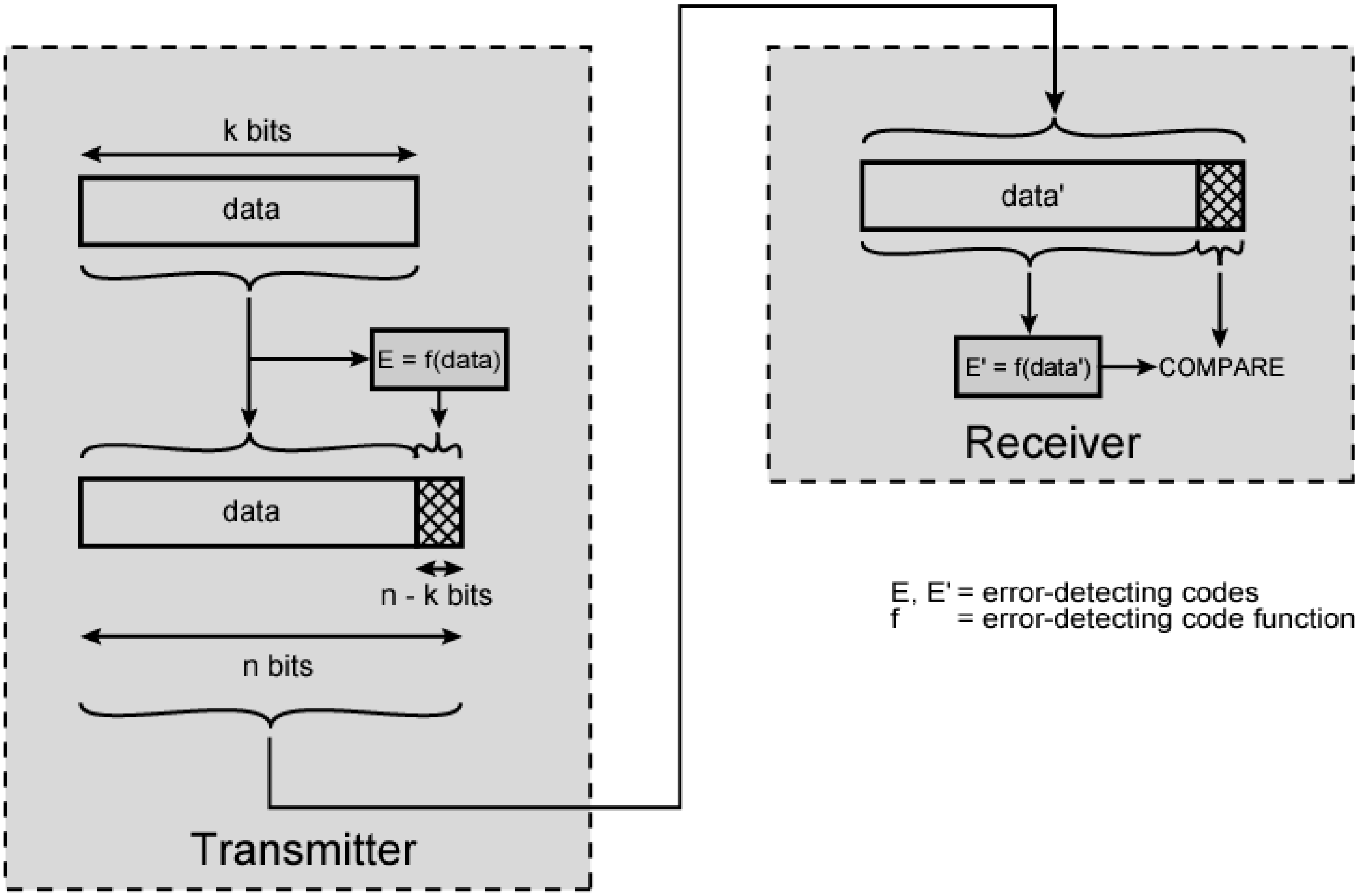
- Perform by calculating an error-detecting code
  - Which is a function of the bits being transmitted
  - Code is appended to the transmitted bits
  - Receiver calculates the code based on the incoming bits and compares it to the incoming code to check for errors

How CRC is Computed?

- Modulo 2 arithmetic
  - Uses binary addition with no carries
- Polynomials
  - Express all values as polynomials in a dummy variable X, with binary coefficients
  - Coefficients correspond to the bits in the binary number

- Digital logic
  - Dividing circuit consisting of XOR gates and a shift register
  - Shift register is a string of 1-bit storage devices
  - Each device has an output line, which indicates the value currently stored, and an input line
  - At discrete time instants, known as clock times, the value in the storage device is replaced by the value indicated by its input line
  - The entire register is clocked simultaneously, causing a 1-bit shift along the entire register

CRC Error Detection Process



E, E' = error-detecting codes
f = error-detecting code function

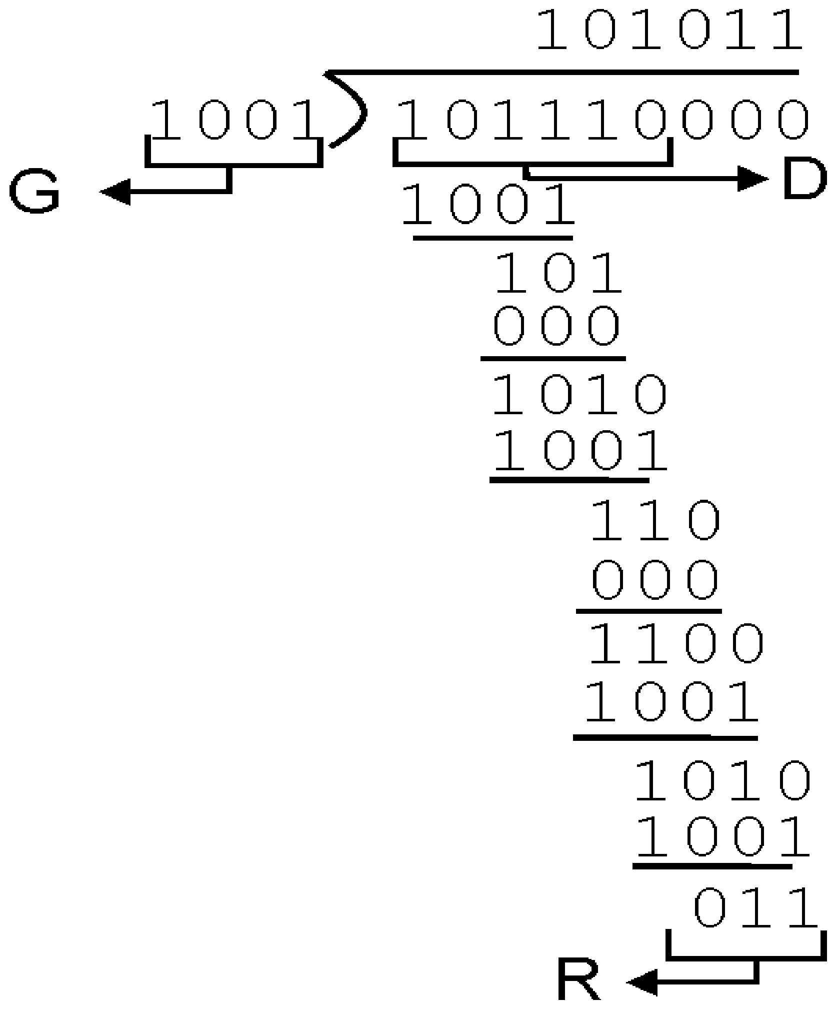
Modulo 2 CRC Example

Sender:

- D : original message
- D = 101110
- G : generator
- G = 1001
- R : CRC remainder
- R = 011
- T : the encoded message (R appended to D)
- T = 101110011

Receiver:

- Divide T by G
- Should have no remainder.



CRC Error Detection

- Additional bits (check bits) added by transmitter for error detection code.
  - For k data bits, (n-k) check bits added
  - n bits are transmitted
- Receiver receives n bits
  - Separate into data bits and check bits
  - Perform error-code calculation for matching

Parity Check

- A parity bit appended to the end of the block of data
  - Typical of character transmission
- Value of parity bit is such that character has even (even parity) or odd (odd parity) number of ones
  - Note: Even number of bit errors goes undetected

Disk Error Checking: Example

- Simple parity bits have been in common use in computer systems for many years.
- Let's look at a simple example
- The parity bit is selected so that the total number of 1's in the data is odd (even) for an odd-parity (even-parity) code.
- When you write you compute the parity bit and add it to the data you are writing.
- When you read you compute it again and see if it is the same parity as you wrote.
- Single parity bits are limited so multiple are needed.

Multiple Parity Bit Check

$$\begin{array}{c|ccc|c}
 & & & & \xrightarrow{\text{row parity}} \\
 & b_{1,1} & \cdots & b_{1,j} & r_1 \\
 & b_{2,1} & \cdots & b_{2,j} & r_2 \\
 & & & & \\
\text{column parity}\downarrow & b_{i,1} & \cdots & b_{i,j} & r_i \\ \cline{2-5}
 & c_1 & \cdots & c_j & p
\end{array}$$

(a) Parity calculation

$$\begin{array}{ccccc|c}
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\ \hline
0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

(b) No errors

$$\begin{array}{ccccc|cl}
0 & 1 & 1 & 1 & 0 & 1 & \\
0 & \textcircled{0} & 1 & 1 & 0 & 1 & \text{row parity error} \\
0 & 1 & 0 & 0 & 0 & 1 & \\
0 & 1 & 0 & 1 & 1 & 1 & \\ \cline{1-6}
0 & 0 & 0 & 1 & 1 & 0 & \\
 & \text{column parity error} & & & & &
\end{array}$$

(c) Correctable single-bit error

$$\begin{array}{ccccccc|c}
0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & \textcircled{0} & 1 & 1 & 0 & \textcircled{1} & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
0 & \textcircled{0} & 0 & 0 & 0 & \textcircled{0} & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline
1 & 1 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}$$

(d) Uncorrectable error pattern

Checksum

- Ones-complement operation.
  - Replace 0 digits with 1 digits and 1 digits with 0 digits.
- Ones-complement addition.
  - The two numbers are treated as unsigned binary integers and added.
  - If there is a carry out of the leftmost bit, add 1 to the sum (end-around carry).

Checksum Computation

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Ones complement of the result	220D

(a) Checksum calculation by sender

Partial sum	0001 F203 F204
Partial sum	F204 F4F5 1E6F9
Carry	E6F9 1 E6FA
Partial sum	E6FA F6F7 1DDF1
Carry	DDF1 1 DDF2
Partial sum	DDF2 220D FFFF

(b) Checksum verification by receiver

Disk Error Checking

- Disks are prone to errors
- Often additional bits are used.
  - These bits are called the checksum
  - The value of these bits is calculated from the data using some function
  - The checksum is stored with the data
- When reading the data from the disk the function used to calculate the checksum is applied again.
  - If what is calculated and the checksum stored are not the same then there is an error

Detecting and Correcting Errors

- Is it possible to detect and correct errors?
- Codeword and Hamming

Error Correction Process

- Transmitter:
  - Mapping k-bit block of data into n-bit codeword
    - By using FEC (forward error correction) encoder
    - totally different from FCS (how?)
  - Then n-bit codeword is transmitted
- Receiver:
  - Codeword passed through the FEC decoder
  - If no errors, output the original data block
  - Some error patterns can be detected and corrected
  - Some error patterns can be detected but not corrected
  - Some (rare) error patterns are not detected
    - Results in incorrect data output from FEC

Calculating the Hamming Code

- The key to the Hamming Code is the use of extra parity bits to allow the identification of a single error. Create the code word as follows:
- Mark all bit positions that are powers of two as parity bits. (positions 1, 2, 4, 8, 16, 32, 64, etc.)
- All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, etc.)
- Example:
  - If data d = $d_1d_2d_3d_4d_5d_6d_7d_8$ …
  - and parity bits: p_1, p_2, p_4, p_8, p_{16}, ….
  - Then, Hamming code = $p_1p_2d_1p_4d_2d_3d_4p_8d_5d_6d_7d_8$ …
- But how are parity bits determined?

Calculating the Hamming Code

- Each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips:
  - Position 1: check 1 bit and skip 1 bit continuously (1,3,5,7,9,11,13,15,...)
  - Position 2: check 2 bits and skip 2 bits continuously (2,3,6,7,10,11,14,15,...)
  - Position 4: check 4 bits and skip 4 bits continuously (4 -7, 12-15, 20-23, ...)
  - Position 8: check 8 bits and skip 8 bits continuously (8-15,24-31,40-47,...)
- For even parity Hamming codeword:
  - Set a parity bit to 1 if the total number of ones in the positions it checks is odd.
  - Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Calculating the Hamming Code

- Example: Find Hamming codeword for 10011010
- Add parity bits: p1 p2 1 p4 0 0 1 p8 1 0 1 0
- P1 checks bits 1,3,5,7,9,11 from: p1 p2 1 p4 0 0 1 p8 1 0 1 0
  - yields even parity so p1= 0
- P2 checks bits 2,3,6,7,10,11 from: 0 p2 1 p4 0 0 1 p8 1 0 1 0
  - yields odd parity so p2 =1
- P4 checks bits 4,5,6,7,12 from: 0 1 1 p4 0 0 1 p8 1 0 1 0.
  - yields odd parity so p4 = 1
- P8 checks bits 8-15: 0 1 1 1 0 0 1 p8 1 0 1 0
  - Yields even parity so p8 = 0
- Thus code word is: 011100101010.

Data at Receiver Side

- Suppose the word received was 011100101110 instead. The data received is not correct.
- Now consider *C* = 011100101110
  - What are the parity bits for the codeword *C*?
  - Can the receiver determine the location of the wrong bit?
  - Can the receiver correct the wrong bit?

Calculating the Hamming Code

- The method is to verify each check bit. Write down all the incorrect parity bits. Doing so, you will discover that parity bits 2 and 8 are incorrect.

- It is not an accident that 2 + 8 = 10, and that bit position 10 is the location of the bad bit. In general, check each parity bit, and add the positions that are wrong. This will give you the location of the bad bit.

Summary

- Hard disk drives and nonvolatile memory are the major secondary I/O components on most computers.
- Disk-scheduling algorithms can improve the effective bandwidth of HDDs.
- Basic hardware involved in I/O are buses, device controllers, and devices themselves.
- DMA controller helps data transfer.
- Data transfer errors can be detected and corrected.

What is Next?

- File Structures will be the next topic to be covered.