OS

Operating Systems

Lecture 15: I/O Subsystem

Department of Computer Science & Technology
Tsinghua University

OS

Outline

- Characteristics of I/O
  - Types of Device Interfaces
  - Synchronous and Asynchronous I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

OS

Three Types of Device Interfaces

- Three common types of device interfaces
  - Character devices
  - Block devices
  - Network devices
- Character Devices
  - Example: keyboard/mouse, serial port, some USB devices
  - Sequential access, single character at a time
  - I/O commands: get(), put(), etc.
  - Often use open file interface and semantics

Block Devices and Network Devices

- Block Devices
  - Example: disk drive, tape drive, DVD-ROM
  - Uniform block I/O interface to access blocks of data
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Network Devices
  - Examples: Ethernet, wireless, bluetooth
  - Different enough from block/character to have own interface
  - Provide special networking interface for supporting various network protocols
  - For example, send/receive network packets

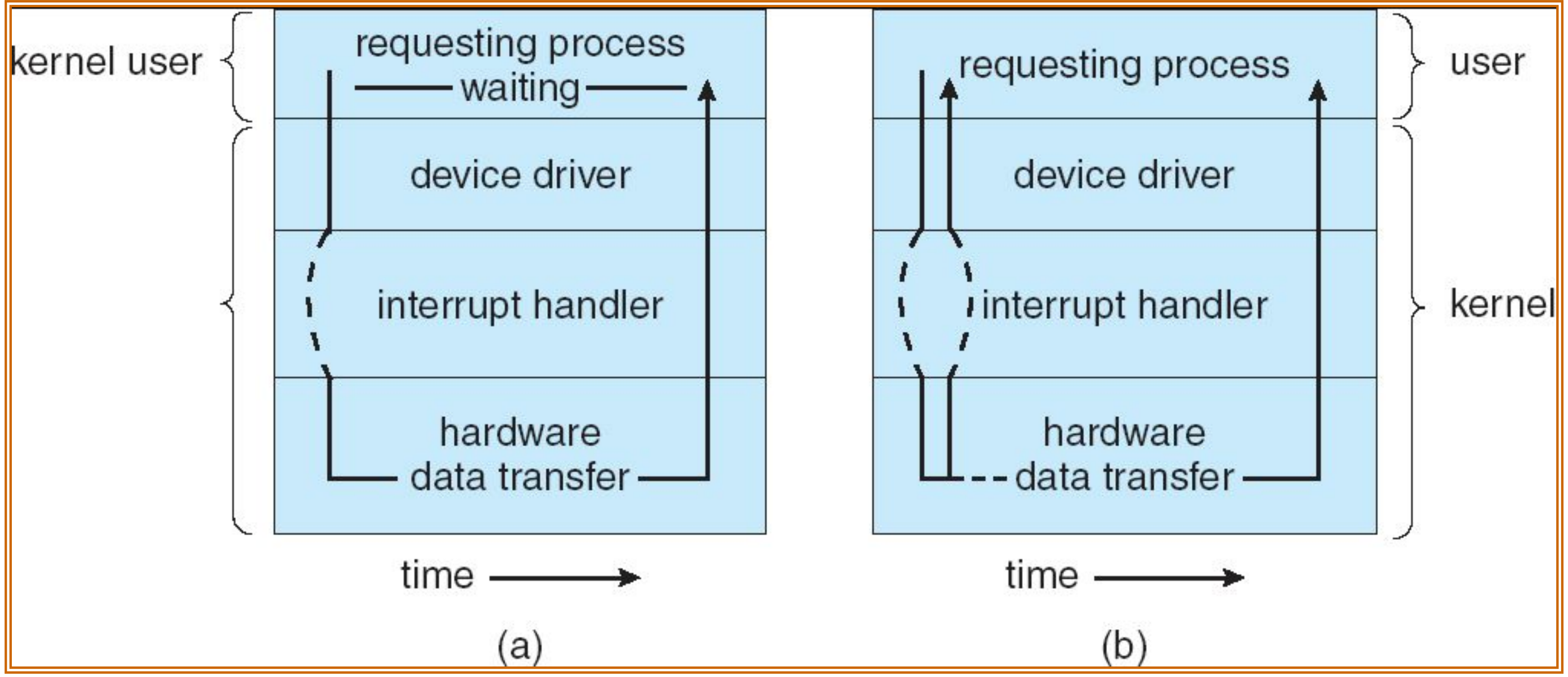
OS

Synchronous and Asynchronous I/O

- Blocking I/O: “Wait”
  - When request data (e.g. read() system call), put process in waiting state until data is ready
  - When write data (e.g. write() system call), put process in waiting state until device is ready for data
- Non-blocking I/O: “Don’t Wait”
  - Returns immediately from read or write request with count of bytes successfully transferred
  - Read may return nothing, write may write nothing
- Asynchronous I/O: “Tell Me Later”
  - When request data, take pointer to user’s buffer, return immediately; later kernel fills buffer and notifies user
  - When send data, take pointer to user’s buffer, return immediately; later kernel takes data and notifies user

OS

Synchronous and Asynchronous I/O

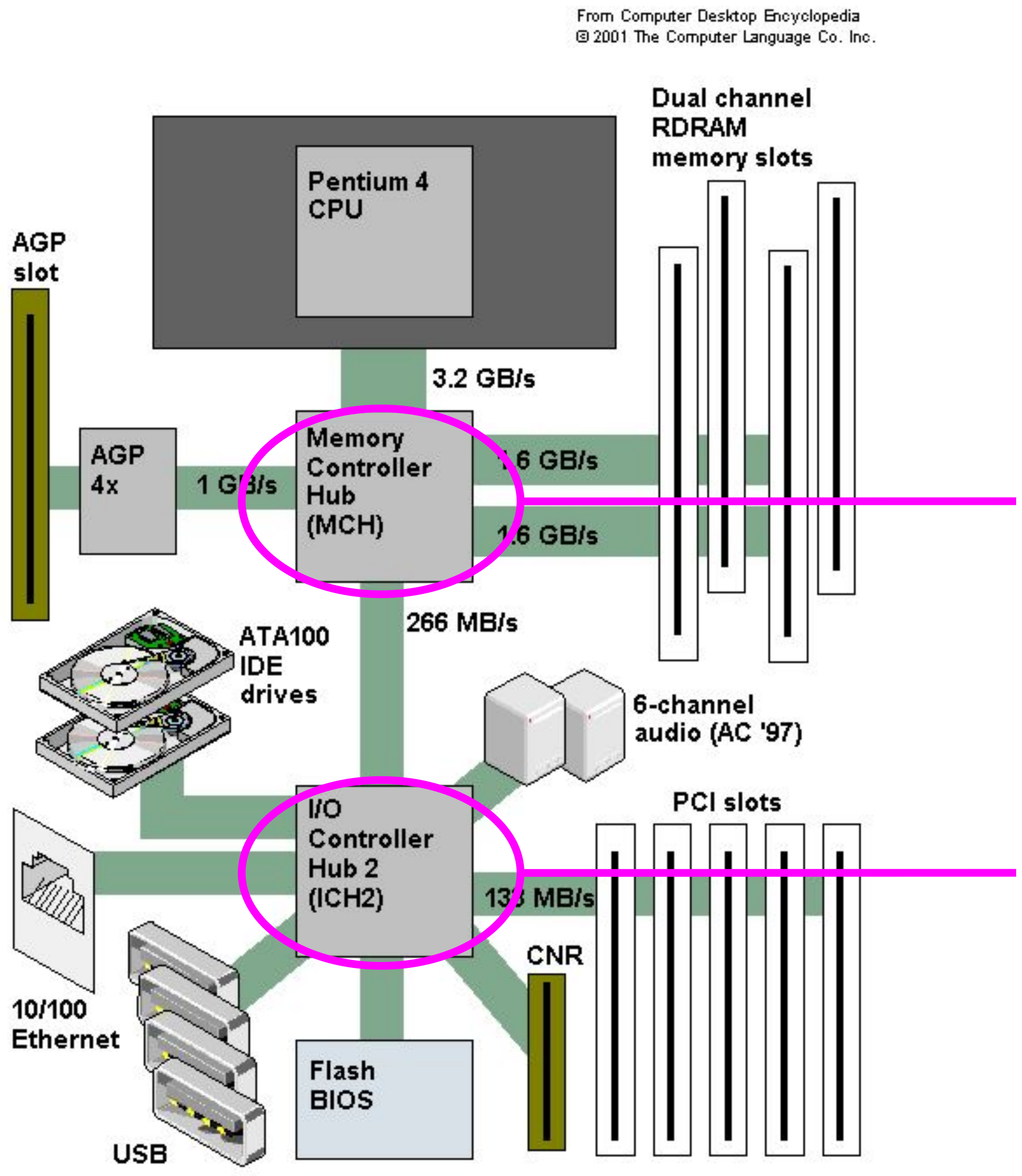


OS

Outline

- Characteristics of I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

I/O Architecture: A Modern Example


From Computer Desktop Encyclopedia
© 2001 The Computer Language Co. Inc.
Pentium 4 CPU
Dual channel RDRAM memory slots
AGP slot
AGP 4x
1 GB/s
3.2 GB/s
Memory Controller Hub (MCH)
1.6 GB/s
1.6 GB/s
266 MB/s
ATA100 IDE drives
6-channel audio (AC '97)
PCI slots
I/O Controller Hub 2 (ICH2)
133 MB/s
CNR
10/100 Ethernet
USB
Flash BIOS

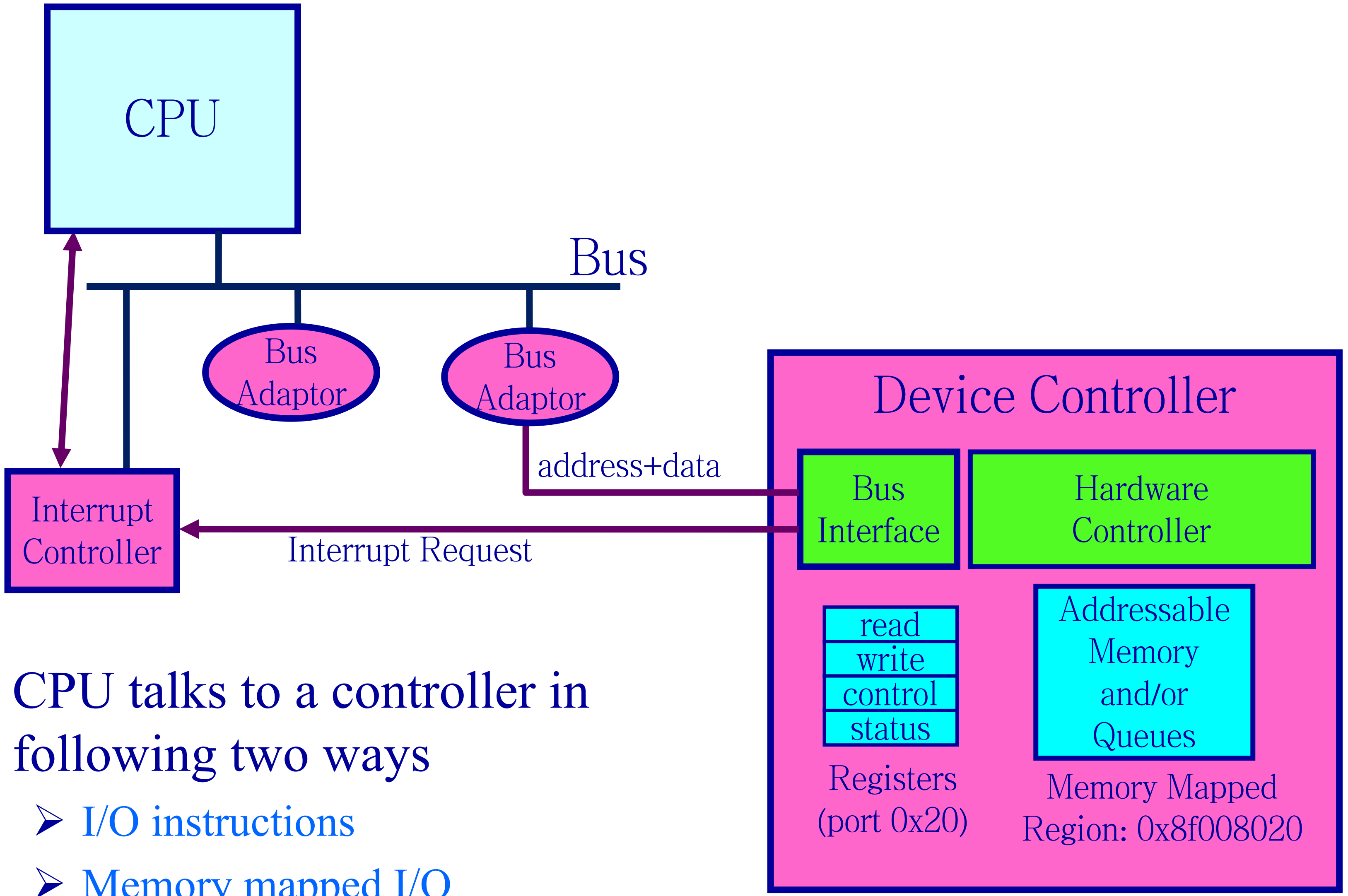
"Northbridge"
- Memory
- AGP/PCI-Express
- Built-in display

"Southbridge"
- ATA/IDE
- PCI bus
- USB/Firewire bus
- Serial/Parallel ports
- DMA controller
- Interrupt controller
- RTC, ACPI, BIOS, …

I/O Hardware

- I/O controllers
  - Interface between CPU and I/O devices
  - Provides CPU with special instructions and registers
- I/O addresses
  - “Names” for CPU to control the I/O hardware
  - Memory locations or port numbers
- OS mechanism
  - Use I/O instruction and I/O address to control a device
  - 3 types of interactions with I/O hardware: polling, interrupt-driven, and DMA

How does CPU Actually Connect to Device?



- CPU talks to a controller in following two ways
  - I/O instructions
  - Memory mapped I/O

I/O Instructions and Memory-Mapped I/O

- I/O instructions
  - Access device’ s registers through I/O port numbers
  - Special CPU instructions dealing with I/O
  - Example from the Intel architecture: out 0x21,AL
- Memory mapped I/O
  - Device’ s registers/memory appear in CPU’ s physical address space
  - I/O accomplished with memory load/store instructions
  - Mapped by MMU, addresses set by hardware jumpers or programming at boot time
  - Can be protected with page tables

OS

Outline

- Characteristics of I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

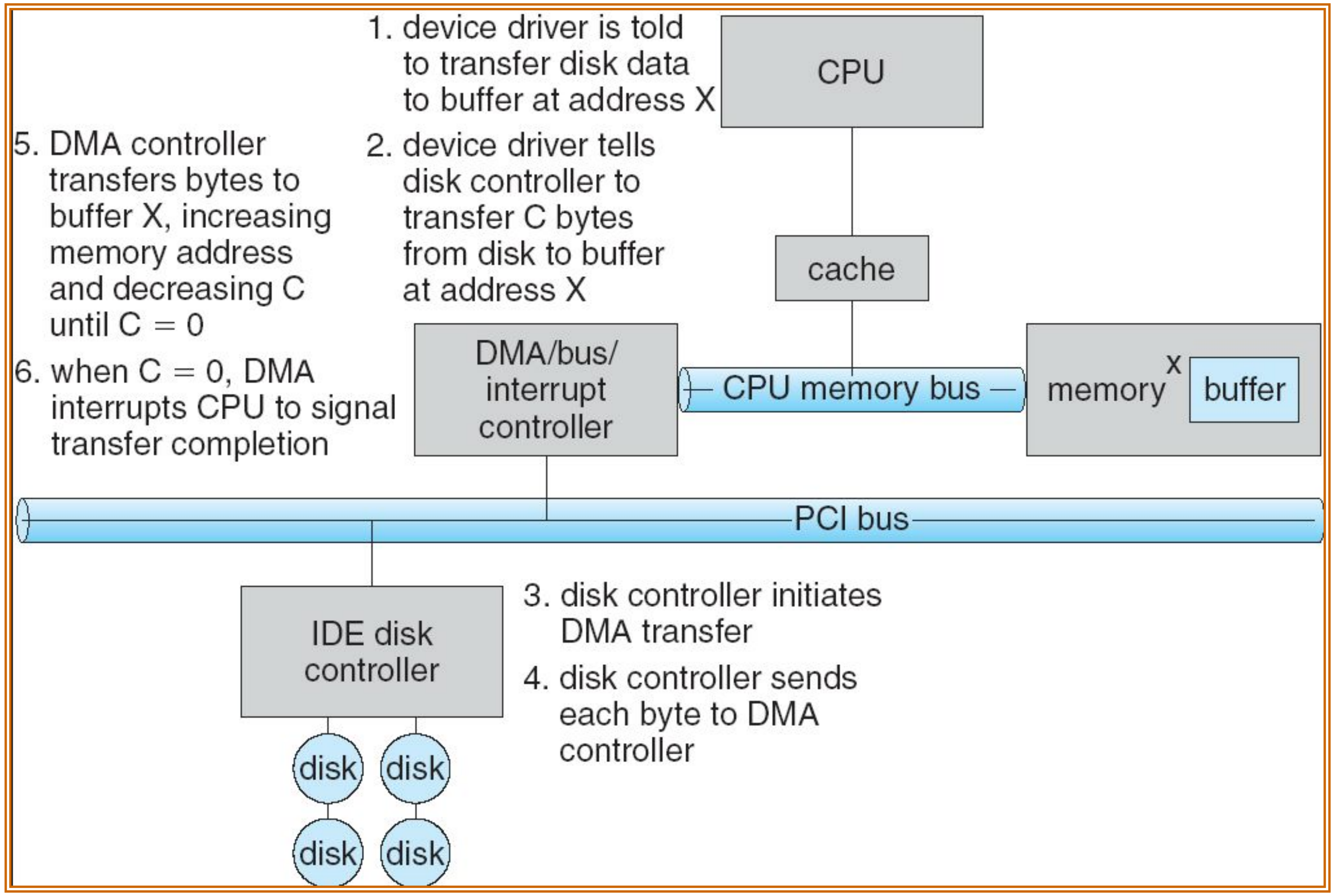
OS

Transfering Data To/From Controller

- Programmed I/O (PIO):
  - Each byte transferred via processor in/out or load/store
  - Pro: Simple hardware, easy to program
  - Con: Consumes processor cycles proportional to data size
  - For small/simple I/O
- Direct Memory Access (DMA):
  - Give controller access to memory bus
  - Ask it to transfer data to/from memory directly
  - Pro: device transfers data without burdening CPU
  - Con: need setup
  - For high throughput I/O

OS

Steps of Disk Read in a DMA Transfer


1. device driver is told to transfer disk data to buffer at address X
CPU
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
cache
6. when C = 0, DMA interrupts CPU to signal transfer completion
DMA/bus/ interrupt controller
CPU memory bus
memory
X
buffer
PCI bus
IDE disk controller
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
disk
disk
disk
disk

I/O Device Notifying the OS

- The OS needs to know when:
  - The I/O device has completed an operation
  - The I/O operation has encountered an error
- Two methods
  - Polling
  - Interrupt-driven

OS

Polling

- I/O device puts completion/error information in device-specific status register
- OS periodically checks the status register

- Pro: simple
- Con: may waste many cycles on polling if infrequent or unpredictable I/O operations

Interrupt-Driven

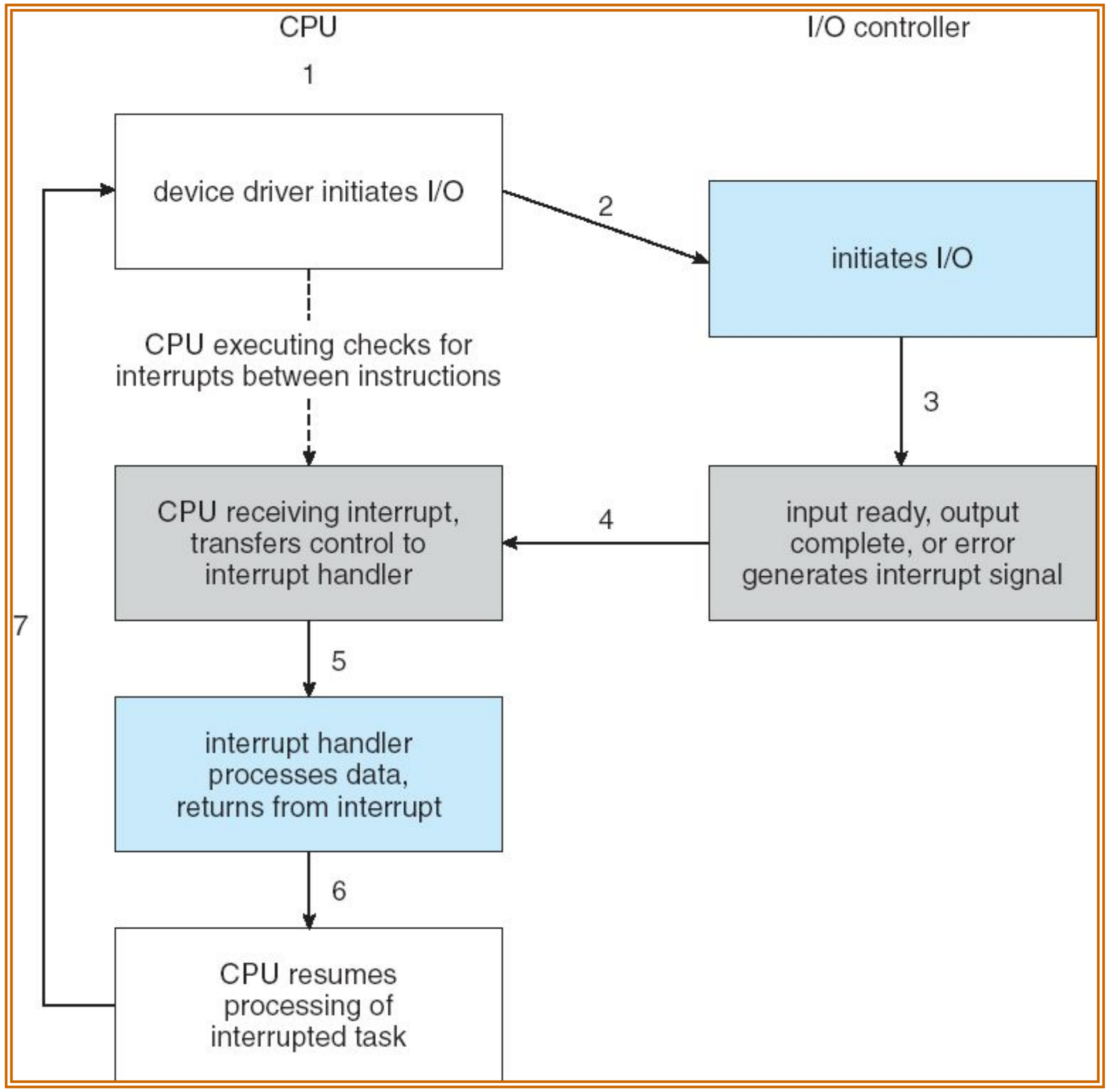
- CPU sets up interrupt handler vector before I/O
- CPU issues I/O request and continues other tasks
- I/O device processes the I/O request
- I/O device triggers CPU interrupt-request line
- Interrupt handler receives interrupts and dispatches to correct handler

- Pro: handles unpredictable events well
- Con: interrupts relatively high overhead

- Some devices may combine both polling and interrupt-driven
  - High-bandwidth network device example: interrupt for first incoming packet, polling for following packets until hardware empty

OS

Interrupt-Driven I/O Cycle

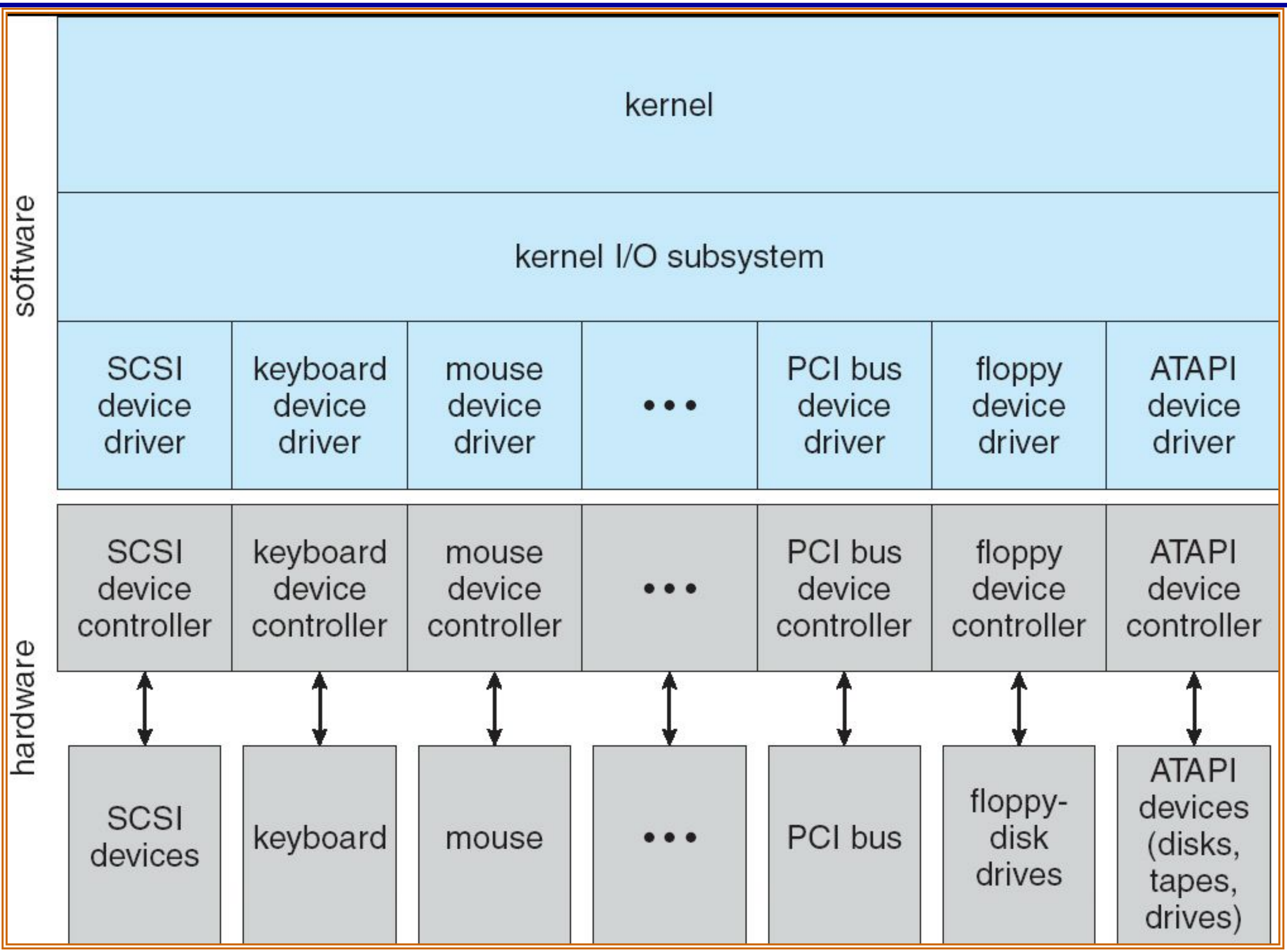

CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from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

Outline

- Characteristics of I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

OS

A Kernel I/O Structure



Device Drivers

- Device-specific code in the kernel that interacts directly with the device hardware
  - Supports a standard internal interface
  - Same kernel I/O system can interact easily with different device drivers
  - Special device-specific configuration supported with the ioctl() system call
- Device drivers typically divided into two pieces:
  - Top half
  - Bottom half

OS

Top Half and Bottom Half

- Device driver top half
  - Accessed in call path from system calls
  - Implements a set of standard, cross-device calls like open(), close(), read(), write(), ioctl(), strategy()
  - This is the kernel’s interface to the device driver
  - Top half will start I/O to device, may put thread to sleep until finished
- Device driver bottom half
  - Run as interrupt routine, often on special kernel stack
  - Gets input or transfers next block of output
  - May wake sleeping threads if I/O now complete

OS

Life Cycle of An I/O Request

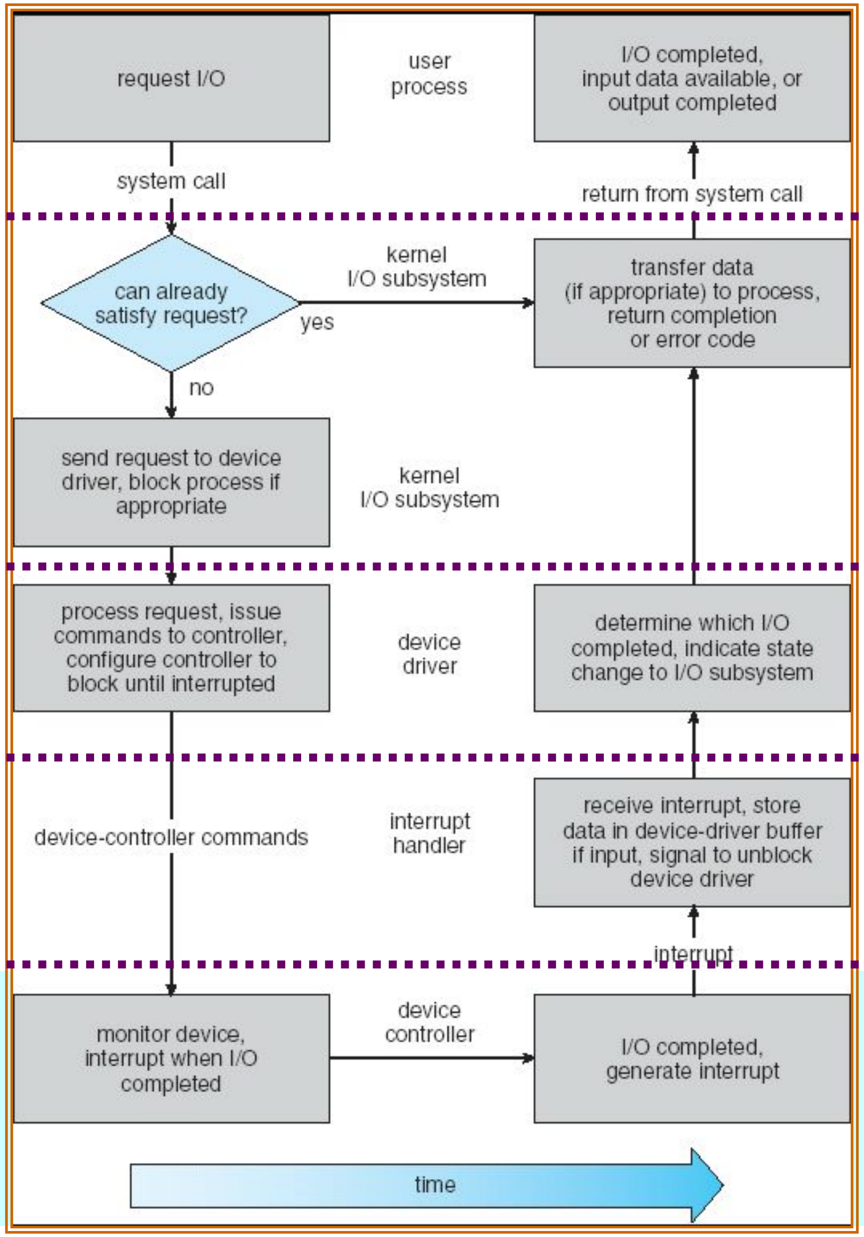
User Program

Kernel I/O Subsystem

Device Driver Top Half

Device Driver Bottom Half

Device Hardware



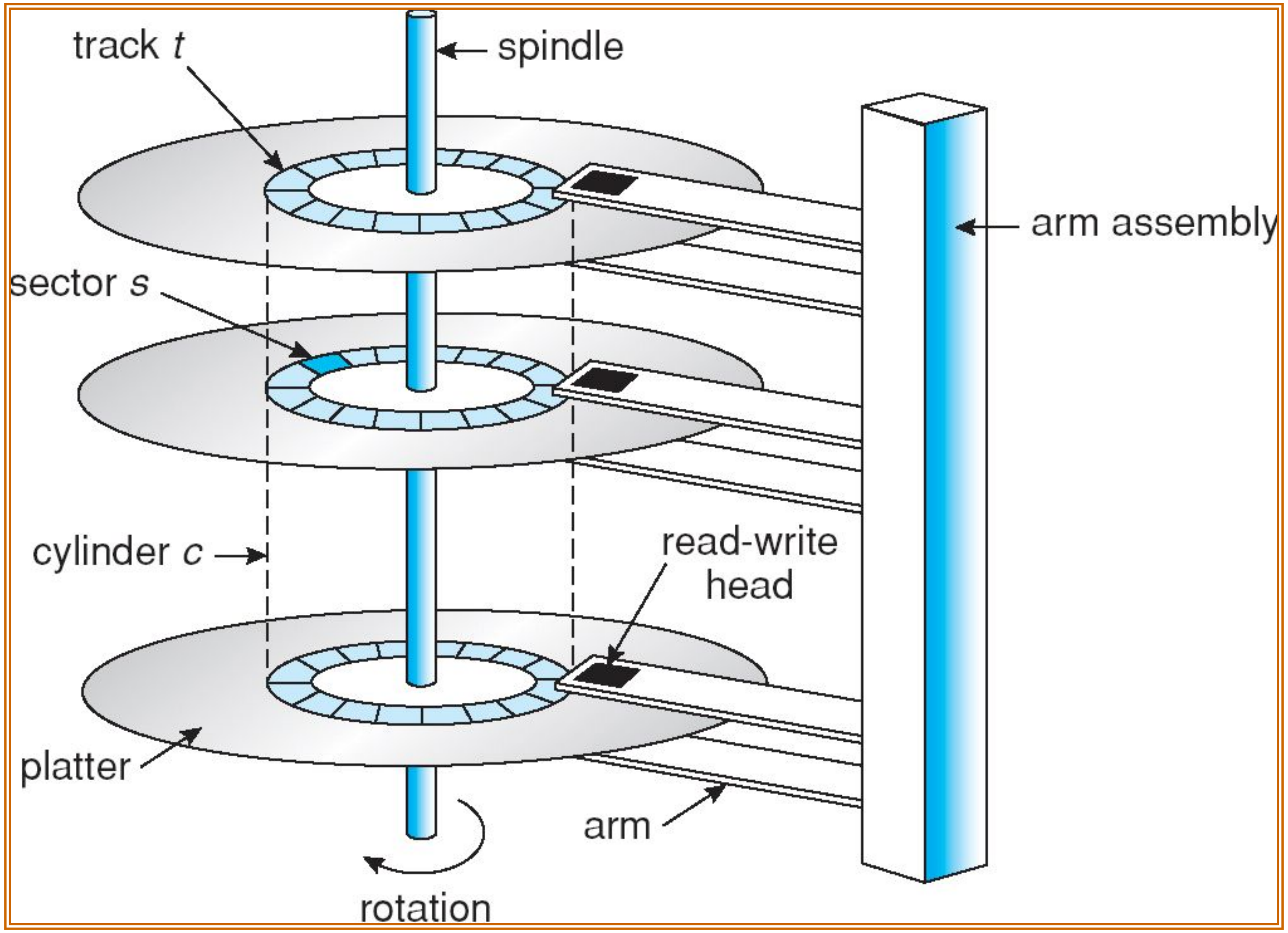
request I/O
user process
I/O completed, input data available, or output completed
system call
return from system call
can already satisfy request?
yes
no
kernel I/O subsystem
transfer data (if appropriate) to process, return completion or error code
send request to device driver, block process if appropriate
kernel I/O subsystem
process request, issue commands to controller, configure controller to block until interrupted
device driver
determine which I/O completed, indicate state change to I/O subsystem
device-controller commands
interrupt handler
receive interrupt, store data in device-driver buffer if input, signal to unblock device driver
interrupt
monitor device, interrupt when I/O completed
device controller
I/O completed, generate interrupt
time

OS

Outline

- Characteristics of I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

Moving-head Disk Mechanism



OS

Disk Performance Parameters

- To read or write, the disk head must be positioned at the desired track and at the beginning of the desired sector
- Seek time
  - Time it takes to position the head at the desired track
- Rotational delay or rotational latency
  - Time its takes for the beginning of the sector to reach the head

Timing of a Disk I/O Transfer

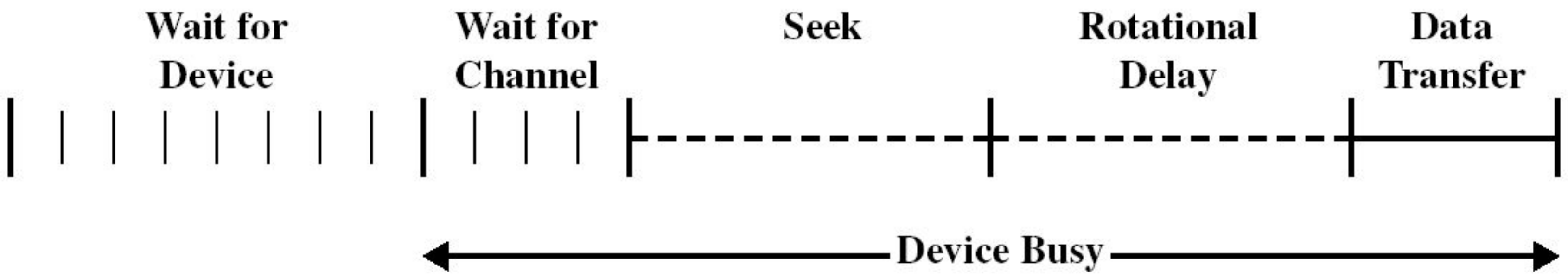


Figure 11.7 Timing of a Disk I/O Transfer

Ts = seek time

Tr = rotational delay

T = transfer time

b = number of bytes to be transferred

N = number of bytes on a track

r = rotation speed of the disk in revolutions per second

$$T_a \quad T_s \quad \frac{1}{2r} \quad \underset{T}{\boxed{\frac{b}{rN}}}$$

OS

Disk Performance Parameters

- Access time
  - Sum of seek time and rotational delay
  - The time it takes to get in position to read or write
- Data transfer occurs as the sector moves under the head

OS

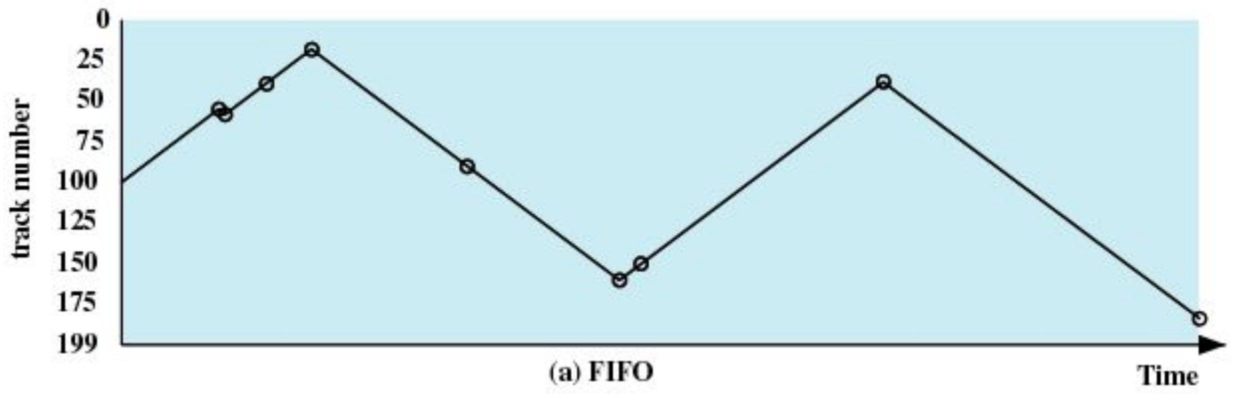
Disk Scheduling Policies

- Seek time is the reason for differences in performance
- For a single disk there will be a number of I/O requests
- If requests are selected randomly, we will poor performance

OS

First-in, first-out (FIFO)

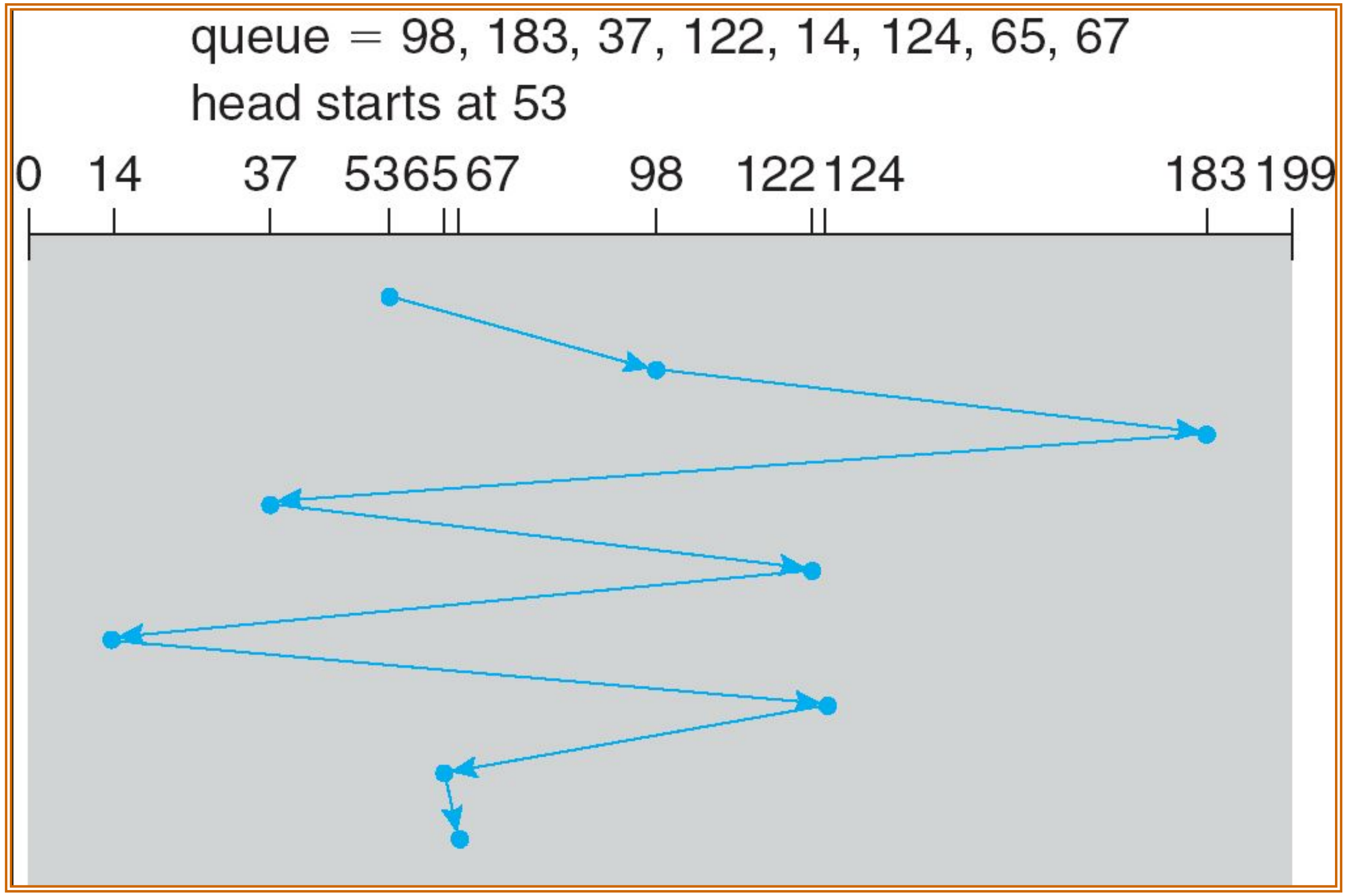
- Process request sequentially
- Fair to all processes
- Approaches random scheduling in performance if there are many processes



OS

FIFO - Example

Illustration shows total head movement of 640 cylinders.



http://cs.uttyler.edu/Faculty/Rainwater/COSC3355/Animations/diskschedulingfcfs.htm

OS

Disk Scheduling Policies - Priority

- Goal is not to optimize disk use but to meet other objectives
- Short batch jobs may have higher priority
- Provide good interactive response time

OS

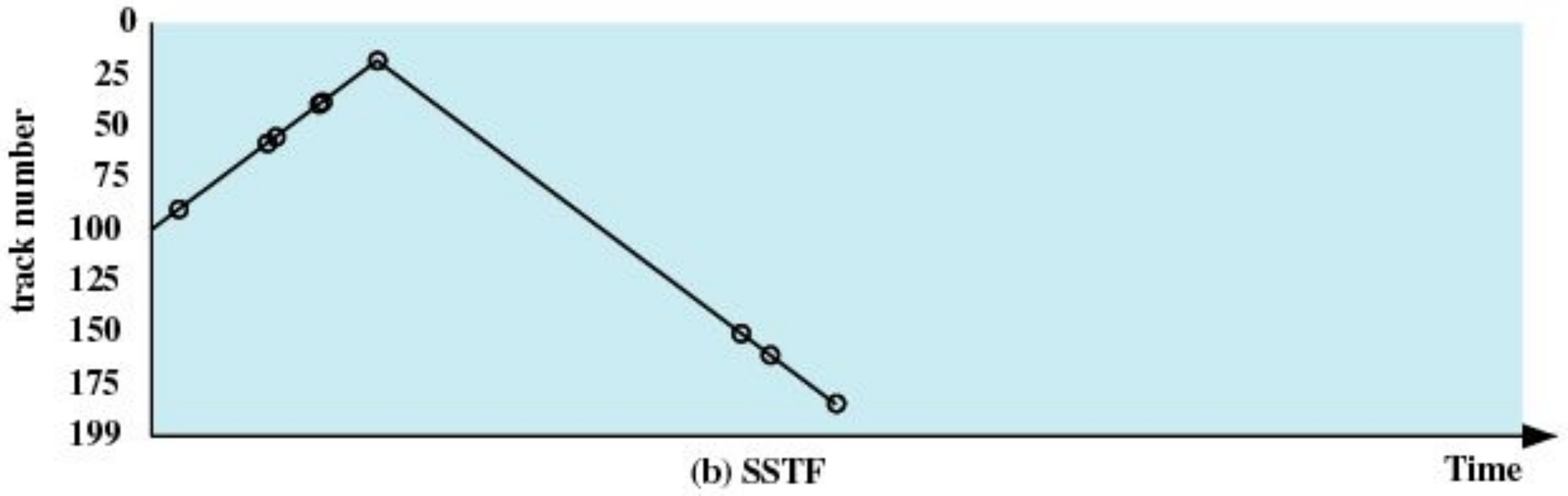
Disk Scheduling Policies - Last-in, first-out

- Good for transaction processing systems
  - The device is given to the most recent user so there should be little arm movement
- Possibility of starvation since a job may never regain the head of the line

OS

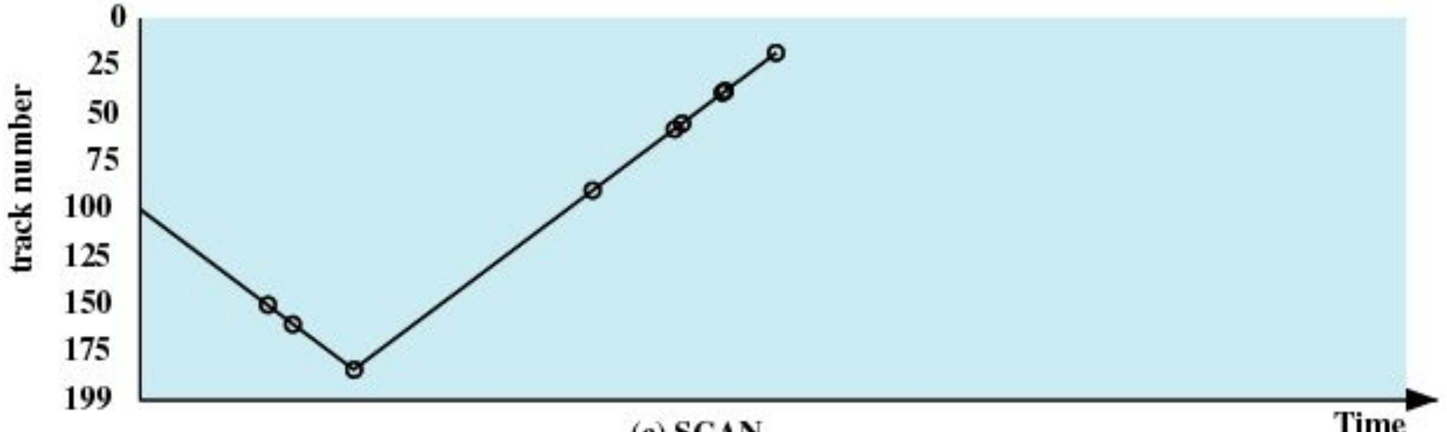
Shortest Service Time First

- Select the disk I/O request that requires the least movement of the disk arm from its current position
- Always choose the minimum Seek time



Disk Scheduling Policies - SCAN

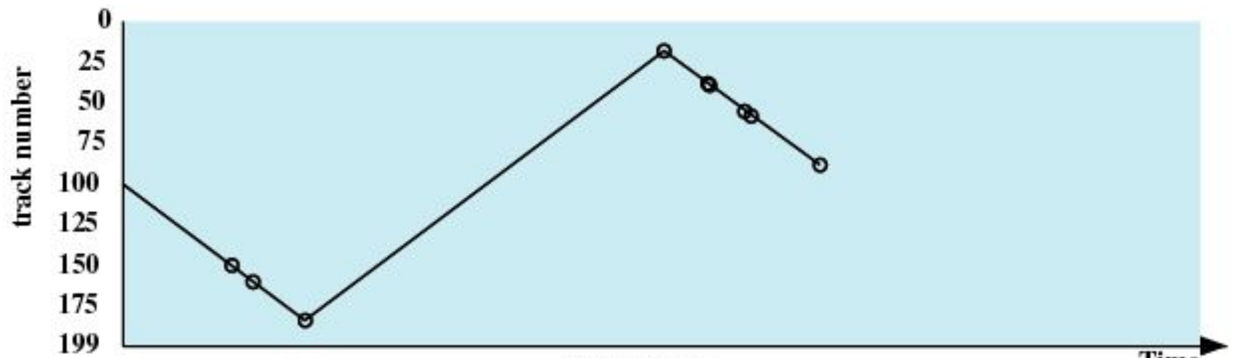
- Arm moves in one direction only, satisfying all outstanding requests until it reaches the last track in that direction
- Direction is reversed
- Sometimes called the elevator algorithm



(c) SCAN

C-SCAN

- Restricts scanning to one direction only
- When the last track has been visited in one direction, the arm is returned to the opposite end of the disk and the scan begins again



(d) C-SCAN

C-LOOK

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

N-step-SCAN & FSCAN

- N-step-SCAN
  - Segments the disk request queue into subqueues of length N
  - Subqueues are processed one at a time, using SCAN
  - New requests added to other queue when queue is processed
- FSCAN
  - Two queues
  - One queue is empty for new requests

Outline

- Characteristics of I/O
- I/O Architecture
- I/O Data Transferring
- I/O Software Layers
- Disk Scheduling
- Disk Cache

Disk Cache

- Buffer in main memory for disk sectors
- Contains a copy of some of the sectors on the disk

OS

Least Recently Used

- The block that has been in the cache the longest with no reference to it is replaced
- The cache consists of a stack of blocks
- Most recently referenced block is on the top of the stack
- When a block is referenced or brought into the cache, it is placed on the top of the stack
- The block on the bottom of the stack is removed when a new block is brought in
- Blocks don’ t actually move around in main memory
- A stack of pointers is used

OS

Least Frequently Used

- The block that has experienced the fewest references is replaced
- A counter is associated with each block
- Counter is incremented each time block accessed
- Block with smallest count is selected for replacement
- Some blocks may be referenced many times in a short period of time and the reference count is misleading