# Overview & I/O Structure

Prof. Li-Pin Chang

CS@NYCU

# THE THREE PILLARS
# (of an operating system)

# Pillar 1: Process Management

- A process is a program in execution; Program is a passive entity, process is an active entity
- Process needs resources to accomplish its task
  - CPU time for execution
  - Memory space for program store
  - I/O bandwidth for communication
  - Storage space for data
- A system has many processes running concurrently on one or more CPUs

# Process Management Activities

- The operating system is responsible to the following activities regarding process management:
  - Creating and terminating processes
  - Providing mechanisms for process communication
  - Providing mechanisms for process synchronization

# Pillar 2: Memory Management

- All data and instructions must be in memory for read/write and execution, respectively
- Memory management activities
  - Allocating and deallocating memory space as needed
  - Deciding which processes (or parts thereof) and data to move into and out of memory (paging)
  - Allowing programs allocate much more memory than physical memory (virtual memory)
  - Keeping track of which parts of memory are currently being used and by whom (protection)

# Pillar 3: File/Storage Management

- OS provides uniform, logical view of information storage, i.e., blocks and file
- File-System management
  - File and directory organization
  - Access control on to determine who can access what
  - Storage space management
- File system basic activities
  - Creating and deleting files and directories
  - Reading and writing files and directories
  - Mapping files from secondary storage to main memory
  - Security and protection
  - Crash recovery and data consistency

# Storage Management

- <span style="color:red">Entire speed of computer operation hinges on storage subsystem and its algorithms</span>
  - Storage ~1 ms; memory ~100ns;
- Optimizing latency and throughput of data access
  - I/O scheduling
  - Data caching and buffering

# OPERATING SYSTEM DEFINITION

# Computer System Overview
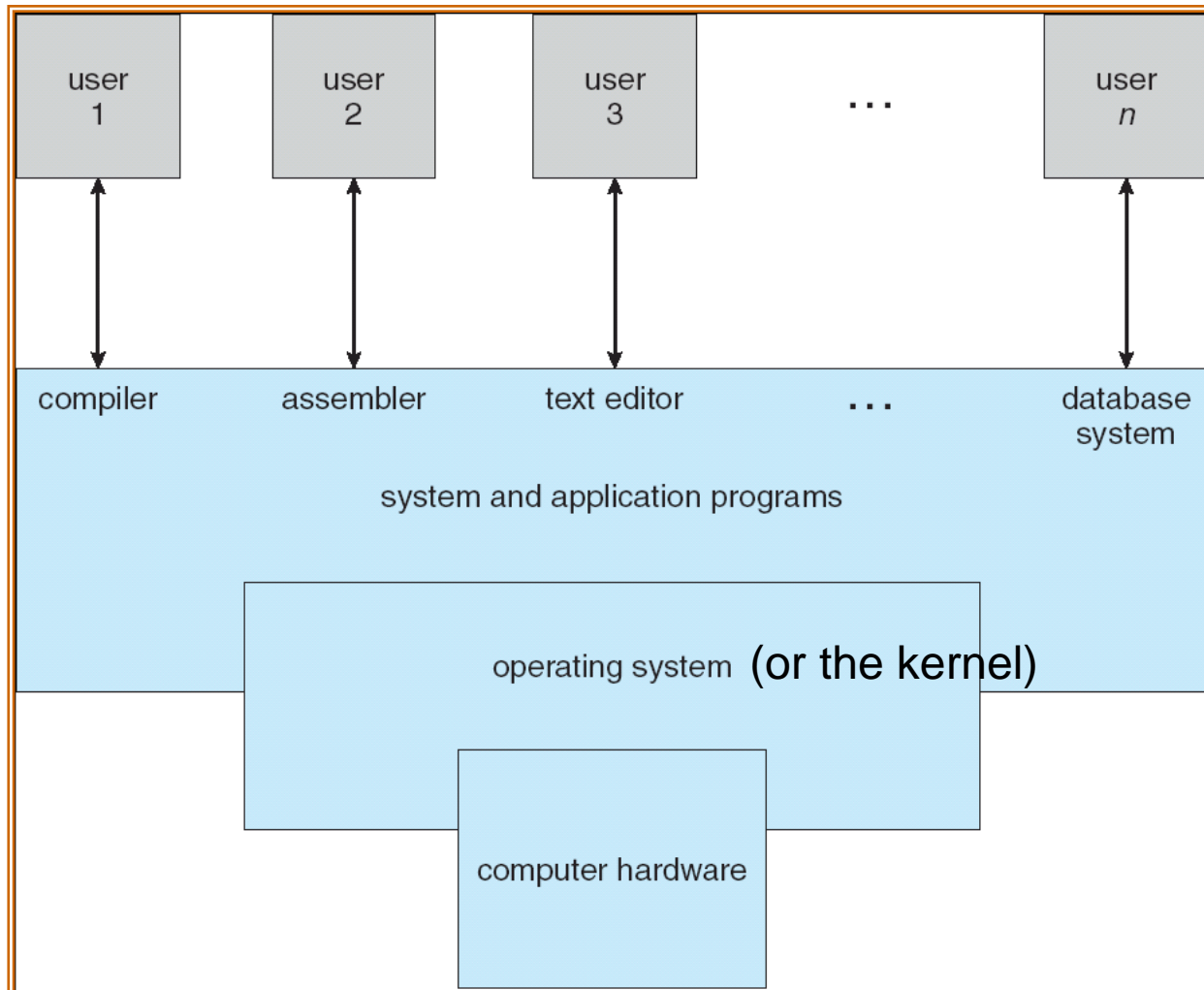
# What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

# Operating System Definition

- OS is a resource allocator
  - Manages all resources
  - Decides between conflicting requests for <span style="color:red">efficient</span> and <span style="color:red">fair</span> resource use

- OS is a control program
  - Controls (or monitor) execution of programs to <span style="color:red">prevent errors and improper use</span> of the computer

# Operating System Definition (Cont'd)

No black-and-white definition

- "*Everything a vendor ships when you order an operating system*" is a rough approximation
  - Retrospect: Internet Explorer in Windows 95
- "The one program running at all times on the computer" is the <span style="color:red">kernel</span>
  - Everything else is either a user application or system program
- Example: the UNIX OS
  - POSIX (IEEE 1003.1-2017)
  - A kernel, plus
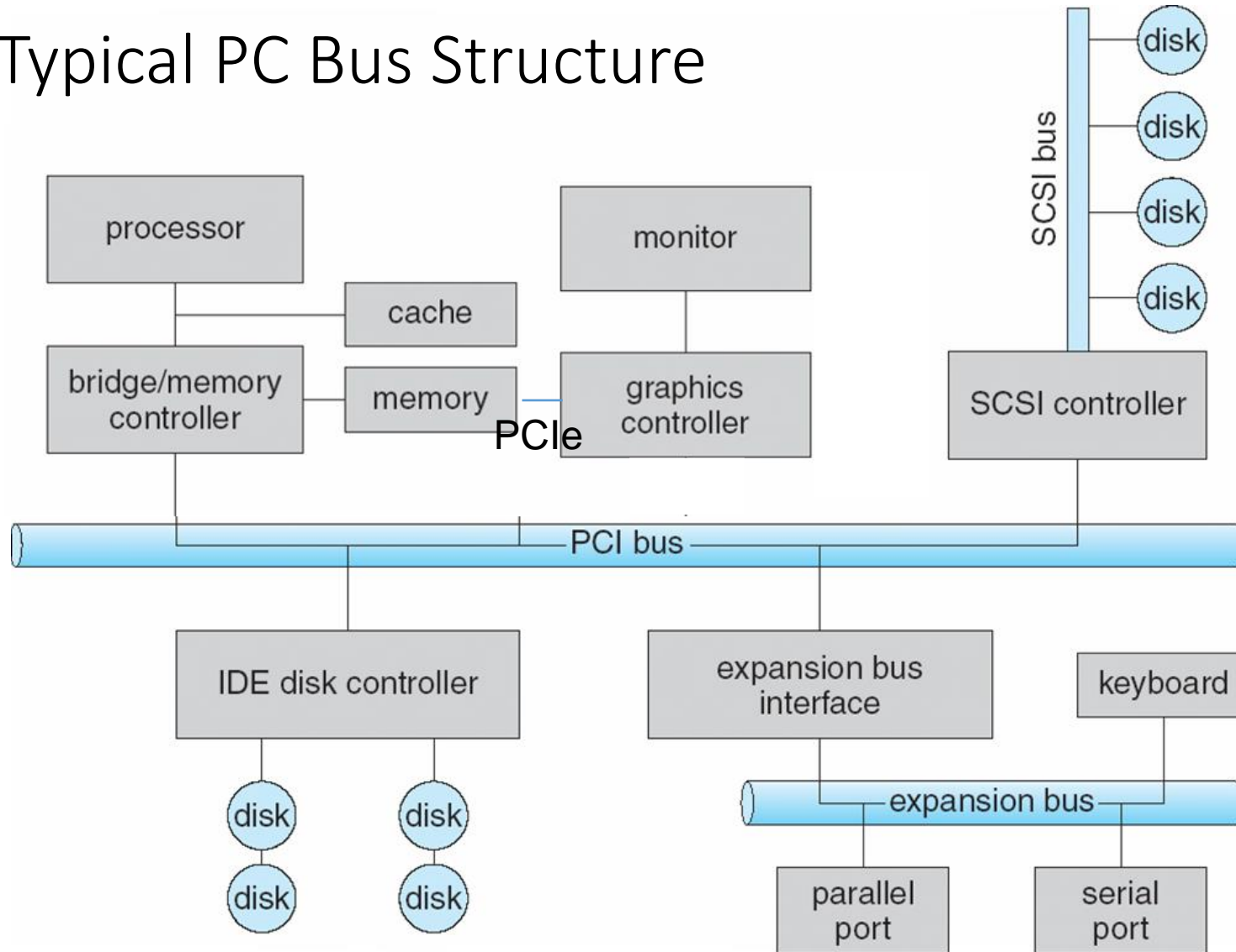  - A collection of system programs ~= coreutil + binutil

- Which one(s) of the following are part of operating systems?
  a) The CPU scheduler
  b) Compilers
  c) Word processors
  d) Binary utilities, such as ls, cp, etc

# I/O STRUCTURE

# Why Bothering with I/Os?

- The kernel is built surrounding interrupts
- The sequential execution of programs is just an illusion that the OS creates; in fact, the CPU jumps back and forth among user programs (multitasking)
- User programs run on the CPU in parallel with operations on I/O devices; OS strives to optimize CPU utilization and I/O utilization at the same time

# A Typical PC Bus Structure



processor

cache

bridge/memory controller

memory

PCIe

monitor

graphics controller

SCSI bus

disk

disk

disk

disk

SCSI controller

PCI bus

IDE disk controller

disk

disk

disk

disk

expansion bus interface

keyboard

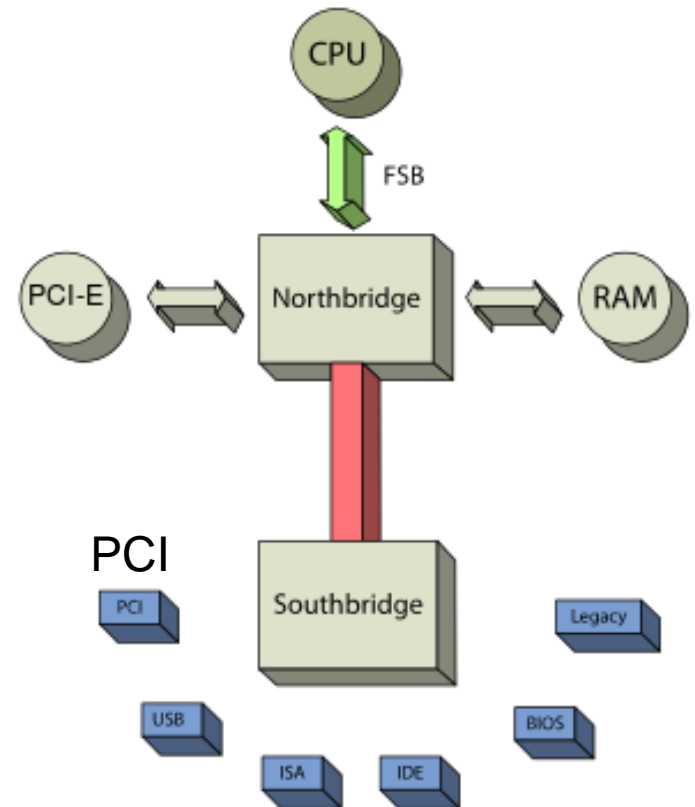expansion bus

parallel port

serial port

# Typical PC Organization

- North bridge (memory hub)
  - CPU, Memory, fast PCIe devices, like graphics, NVMe SSDs
  - Has been integrated into the CPU
- South bridge (IO hub)
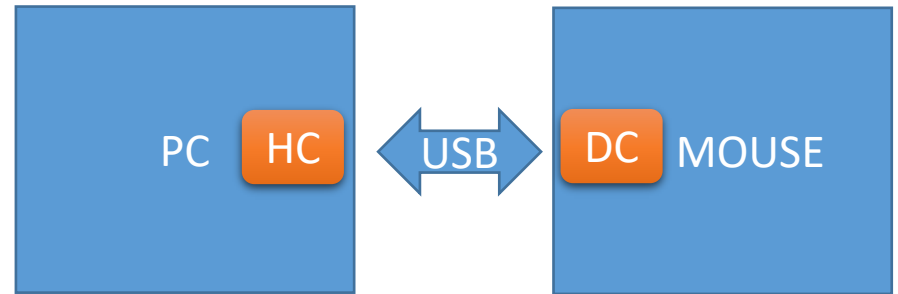  - PCI, USB, SCSI, SATA
  - …

Remark
PCI (parallel, slow, on south)
PCIe (serial, fast, on north)

# I/O Hardware

- Incredible variety of I/O devices
- Common concepts
  - Host controller
  - Bus or interface
  - Device controller
- I/O instructions control devices
  - Direct I/O
  - Memory-mapped I/O

PC  HC  USB  DC  MOUSE

# Partial Device I/O Ports for Legacy PCs (Direct I/O)

| I/O address range (hexadecimal) | device |
|---|---|
| 000–00F | DMA controller |
| 020–021 | interrupt controller |
| 040–043 | timer |
| 200–20F | game controller |
| 2F8–2FF | serial port (secondary) |
| 320–32F | hard-disk controller |
| 378–37F | parallel port |
| 3D0–3DF | graphics controller |
| 3F0–3F7 | diskette-drive controller |
| 3F8–3FF | serial port (primary) |

# Example: Set the x86 Interval Timer (Direct I/O)

```
mov al, 0x36
out 0x43, al    ;tell the PIT which channel we're setting

mov ax, 11931
out 0x40, al    ;send low byte
mov al, ah
out 0x40, al    ;send high byte
```

$11931820 / x = f.$
Set x=11931 → f=1000Hz

| Channel 0 | 0x40 | System Timer |
| Channel 1 | 0x41 | DRAM refresh (obsolete) |
| Channel 2 | 0x42 | Buzzer |
| Command | 0x43 | Command I/O register |

https://en.wikibooks.org/wiki/X86_Assembly/Programmable_Interval_Timer
https://wiki.osdev.org/Programmable_Interval_Timer

# I/O Operations

I/O devices and CPUs execute concurrently

- Polling I/O: The CPU waits on an I/O device until completion

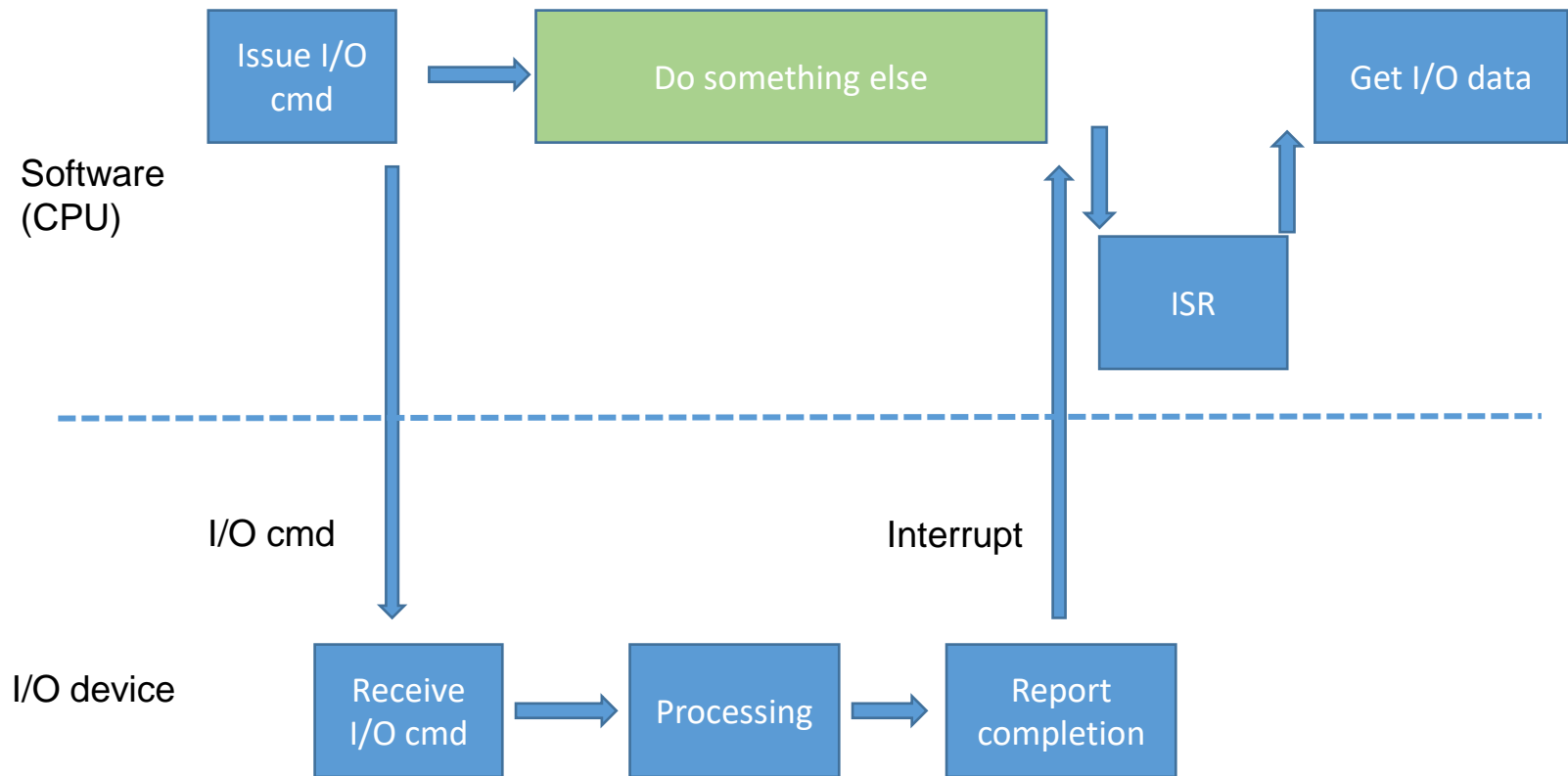- Interrupt I/O: The CPU will be notified of I/O completion

# Polling

- CPU determines state of device
  - Busy
  - Data ready
  - Error
- Busy-wait cycle to wait for I/O from device

No                                         No

| Issue a cmd | → | Is data ready? | → | Read data | → | Read Done? | → | Finish |
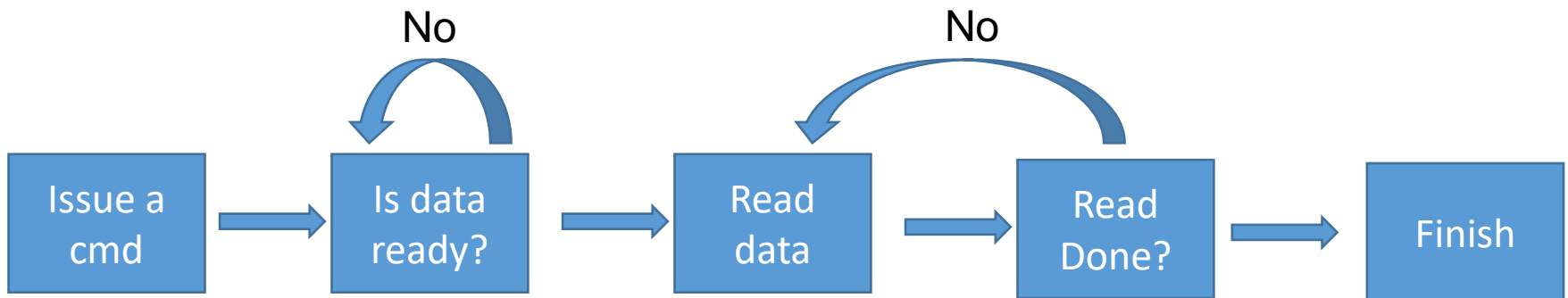
# Interrupts

- CPU Interrupt-request line triggered by I/O device
- Interrupt handler (ISR) receives interrupts
- Interrupt vector to dispatch interrupt to correct handler
- Interrupt mechanism also used for exceptions

# A (Simplified) Interrupt-Driven I/O

Software (CPU)

Issue I/O cmd → Do something else

Get I/O data

ISR

I/O cmd

Interrupt

I/O device

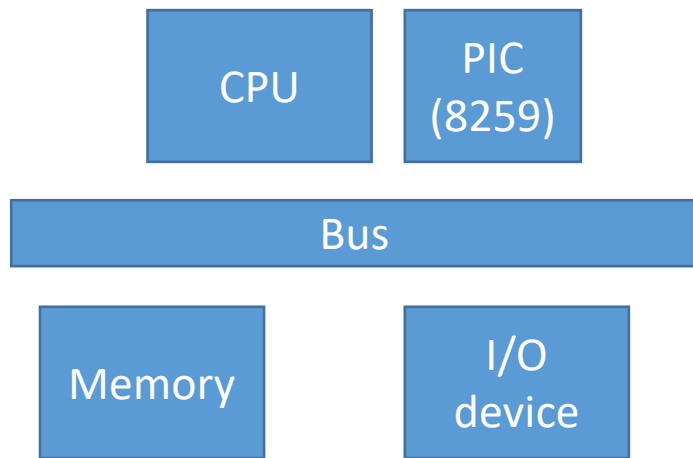Receive I/O cmd → Processing → Report completion

# Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine (ISR) generally, through the interrupt vector, which contains the addresses of all the service routines
  - A trap is a software-generated interrupt caused either by an error or a user request
  - An IRQ is generated by hardware

- An operating system is interrupt-driven

- Hardware: interrupt request (IRQ)
  - I/O completion, timer, etc

- Software: Trap
  - divide by zero, access violation, system call

# Legacy PC interrupts

| Interrupt | Function |
|---|---|
| 00h | Divide By Zero |
| 01h | Single Step |
| 02h | Nonmaskable Interrupt (NMI) |
| 03h | Breakpoint Instruction |
| 04h | Overflow Instruction |
| 05h | Print Screen |
| 05h | Bounds Exception (80286, 80386) |
| 06h | Invalid Op Code (80286, 80386) |
| 07h | Math Coprocessor Not Present |
| 08h | Double Exception Error (80286, 80386) (AT Only) |
| 08h | System Timer - IRQ 0 |
| 09h | Keyboard - IRQ 1 |
| 09h | Math Coprocessor Segment Overrun (80286, 80386) (AT Only) |
| 0Ah | IRQ 2 - Cascade from Second programmable Interrupt Controller |
| 0Ah | Invalid Task Segment State (80286, 80286) (AT Only) |
| 0Ah | IRQ 2 - General Adapter Use (PC Only) |
| 0Bh | IRQ 3 - Serial Communications (COM 2) |
| 0Bh | Segment Not Present (80286, 80386) |
| 0Ch | IRQ 4 - Serial Communications (COM 1) |
| 0Ch | Stack Segment Overflow (80286, 80386) |
| 0Dh | Parallel Printer (LPT 2) (AT Only) |
| 0Dh | IRQ 5 - Fixed Disk (XT Only) |
| 0Dh | General Protection Fault (80286, 80386) |
| 0Eh | IRQ 6- Diskette Drive Controller |
| 0Eh | Page Fault (80386 Only) |
| 0Fh | IRQ 7 - Parallel printer (LPT 1) |

# Interrupt Handling

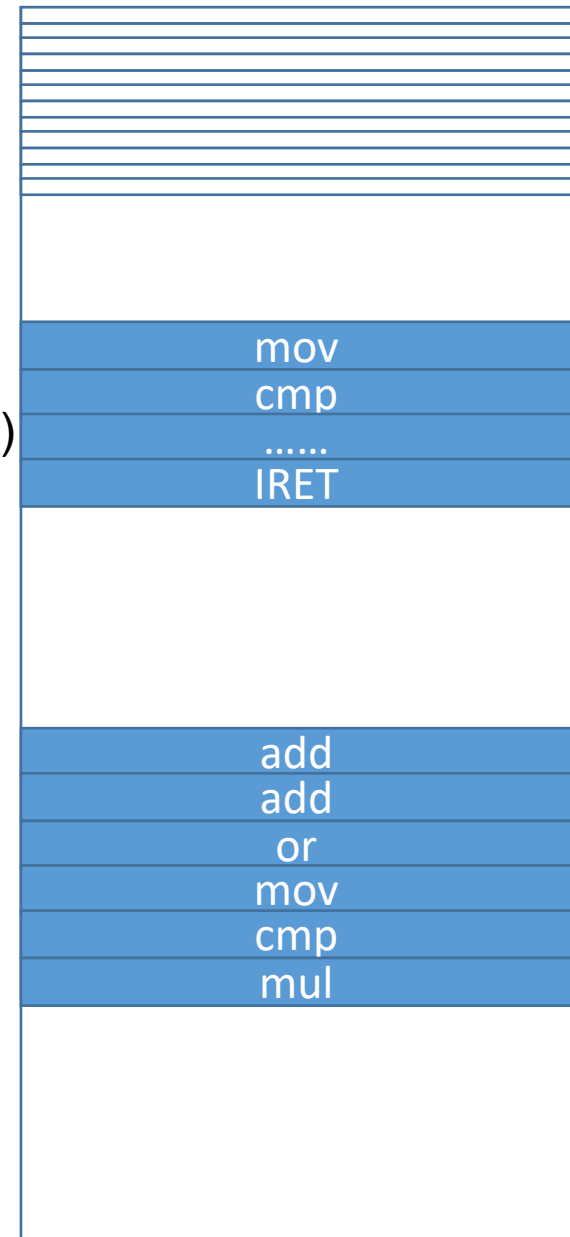- The operating system preserves the state of the CPU by storing <span style="color:red">registers</span> and the <span style="color:red">program counter</span>
- Determines which type of interrupt has occurred:
  - Reading I/O registers
  - vectored interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt
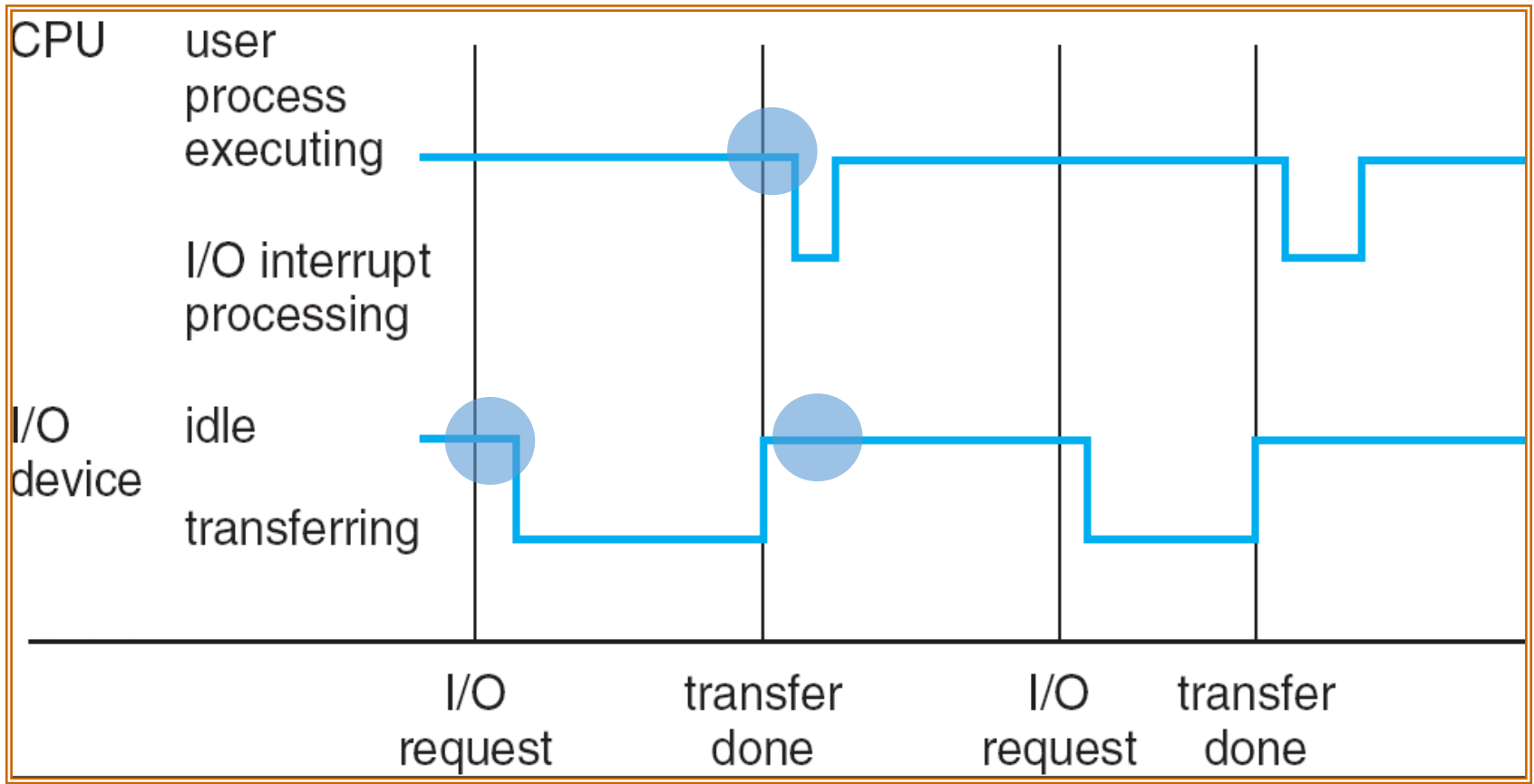
IVT

ISR
(part of OS)

| mov |
| cmp |
| ...... |
| IRET |

CPU

PIC
(8259)

Bus

Memory

I/O
device

User
program
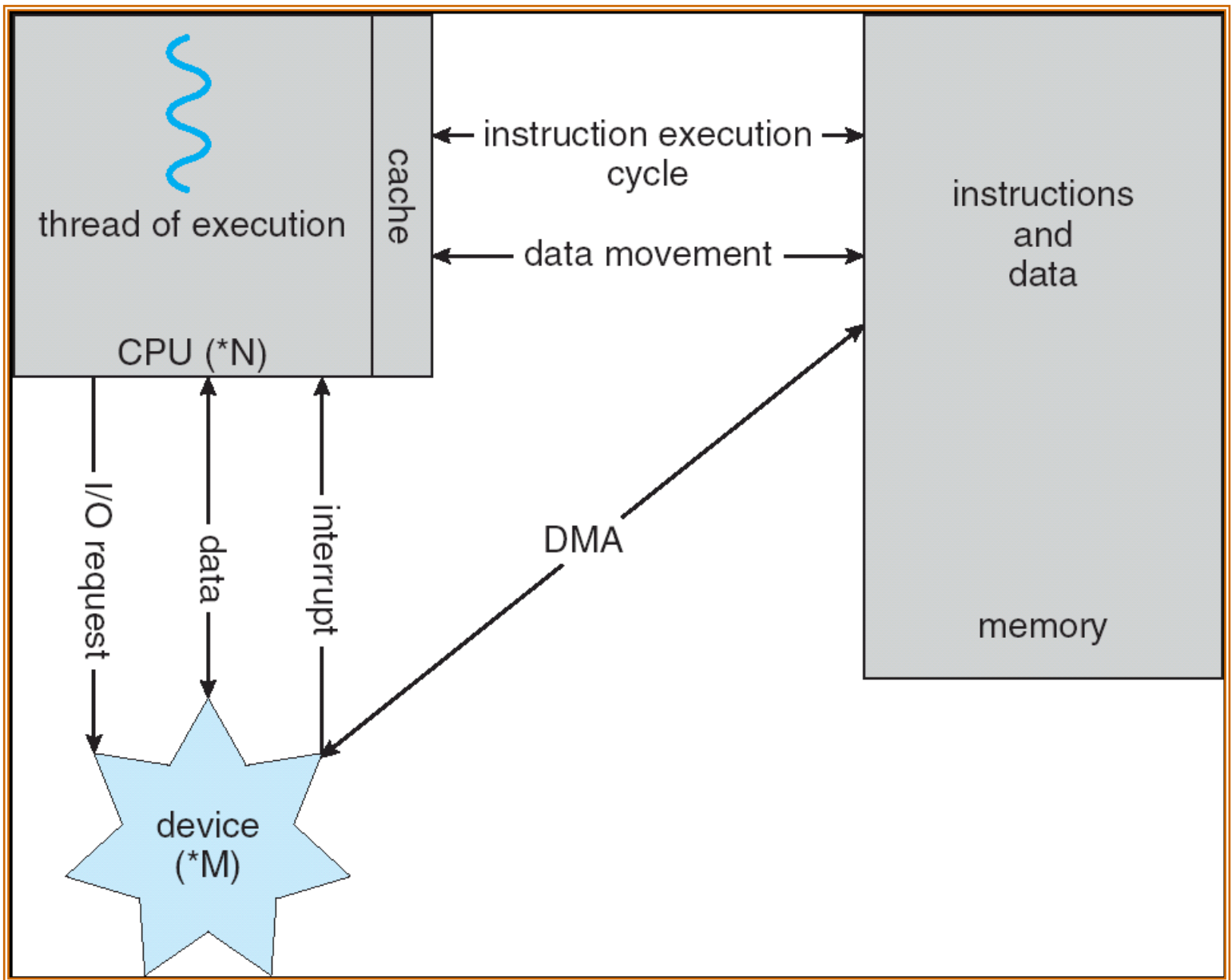
| add |
| add |
| or |
| mov |
| cmp |
| mul |

IVT=interrupt vector table
ISR=interrupt service routine
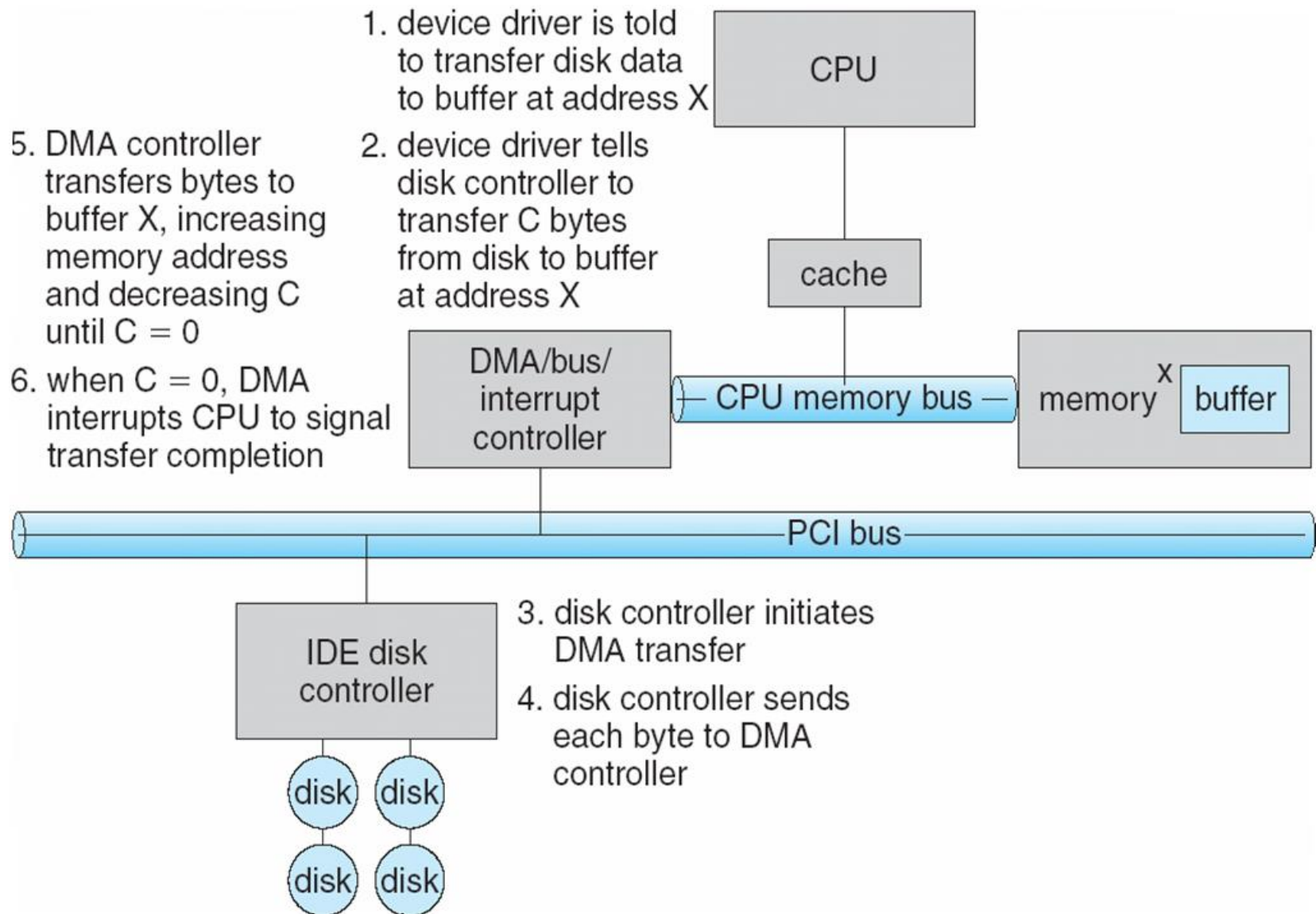PIC=programmable interrupt controller

# Interrupt Timeline

# Direct Memory Access

- Data movement of I/O operation
  - Each device controller has a local buffer
  - Option 1: CPU moves data between main memory and I/O local buffers
  - Option 2: CPU offloads data movement to DMA

- DMA: Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only on interrupt is generated per block, rather than the one interrupt per byte

1. device driver is told to transfer disk data to buffer at address X

2. device driver tells disk controller to transfer C bytes from disk to buffer at address X

3. disk controller initiates DMA transfer

4. disk controller sends each byte to DMA controller

5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0

6. when C = 0, DMA interrupts CPU to signal transfer completion

From Ch13

# Synchronous I/O vs. Asynchronous I/O

- After I/O starts, control returns to the program only upon I/O completion (blocking call; sync I/O)
  - Example: regular I/O reading
    read() or fread()

- After I/O starts, control returns to the program without waiting for I/O completion (non-blocking call; async I/O)
  - Example: regular I/O writing
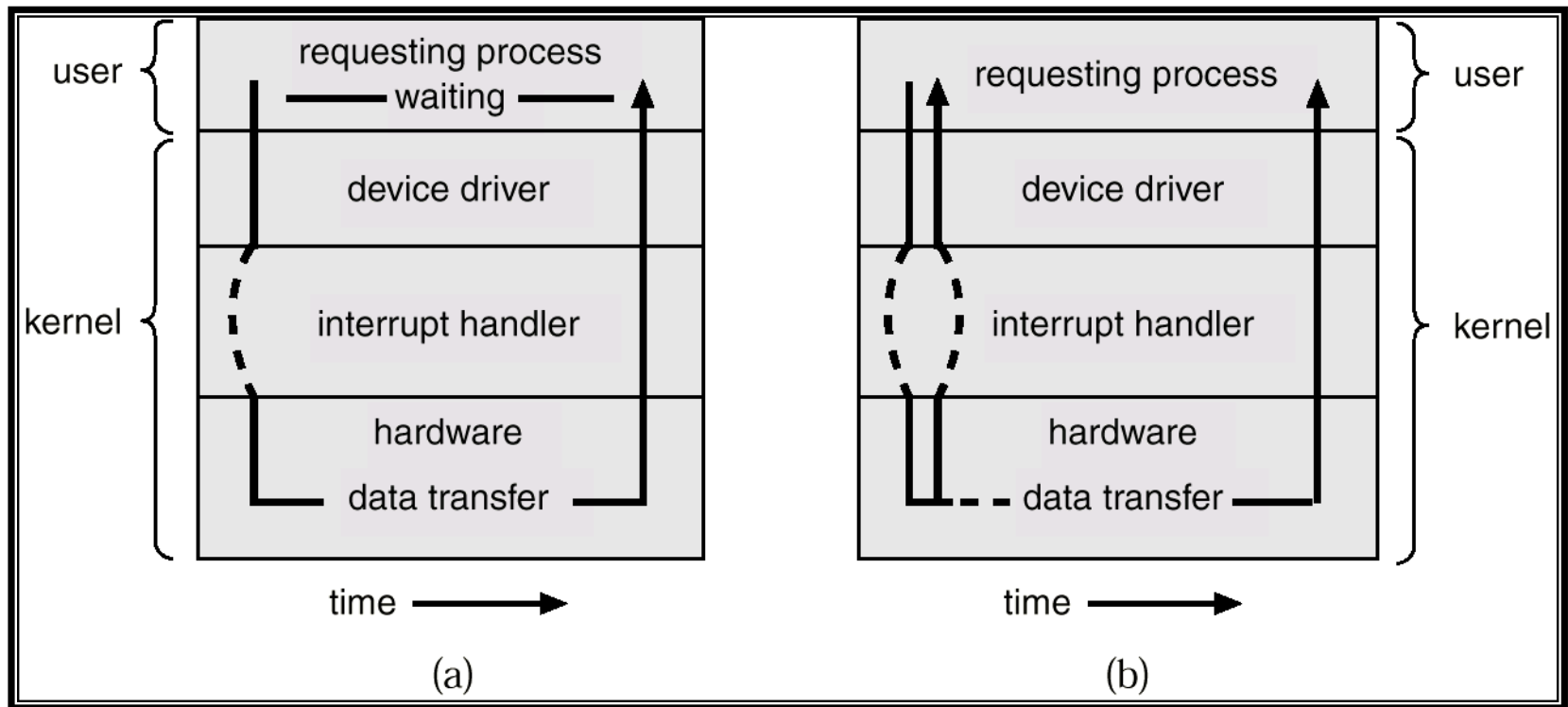    write() or fwrite()

# I/O Structure (Async vs Sync)
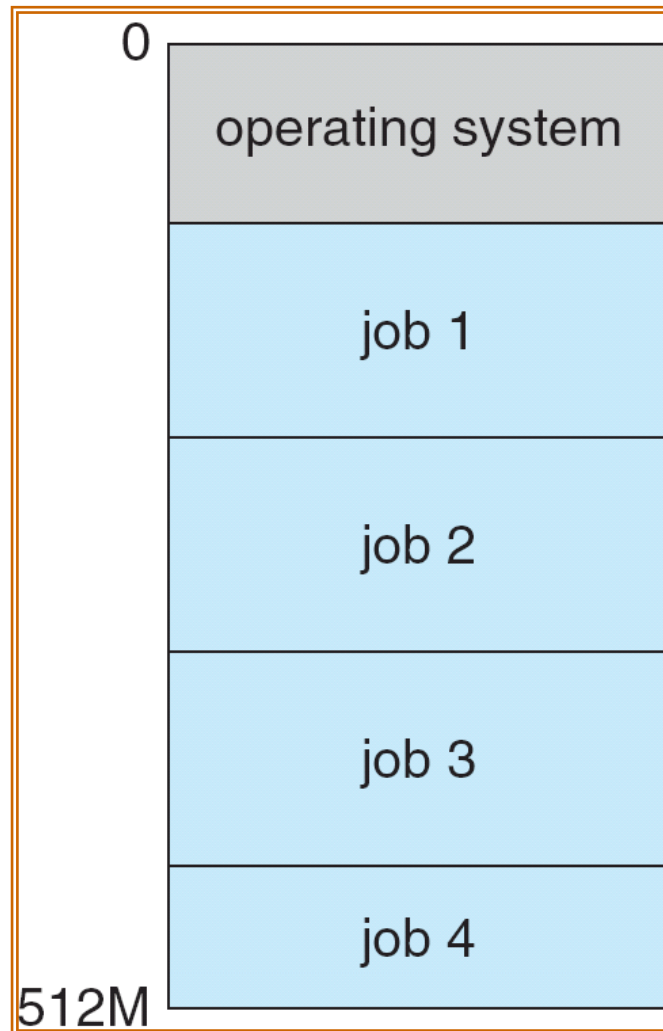


Synchronous / Blocking I/O

Asynchronous / Non-blocking I/O

- Are the following operations synchronous or asynchronous?
  - read()
  - write()
  - fsync()
  - aio_read()

# Multiprogramming

# A Possible Memory Layout of Multiprogramming Systems

# Multiprogramming

- Single process cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes processes so CPU always has one to execute
- A subset of all processes in system is kept in memory
- One process selected and run via process scheduling
- When it has to wait (for I/O for example), OS switches to another process

# Timesharing

- Timesharing is logical extension in which CPU switches processes so frequently that users can interact with each job while it is running, creating interactive computing
  - Switching frequency usually >= 100 Hz, driven by timer IRQ
  - Each user has at least one program executing in memory ⇨ process (ch3)
  - If several jobs ready to run at the same time ⇨ CPU scheduling (ch5)
  - If processes don't fit in memory, swapping moves them in and out to run (ch5)
  - Virtual memory allows execution of processes not completely in memory (ch8)

# Know the Terminology

- Multiprogramming
  - multiple processes are loaded into memory for execution
- Multitasking
  - Multiprogramming with overlapped task execution
- Timesharing
  - multitasking + periodic switch among processes

# Before the OS

# Before the OS: Computer Startup

- Bootstrap program is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM, generally known as firmware
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution
- Legacy PC bootloading procedure
  - FFFF:0000
  - BIOS
  - MBR (Master Boot Record)
  - Boot Manager
  - Pre-OS
  - OS

"BIOS+MBR" has been replaced by "UEFI+GPT", an extensive spec.
However, their purposes are highly similar.

# PC Boot Sequence
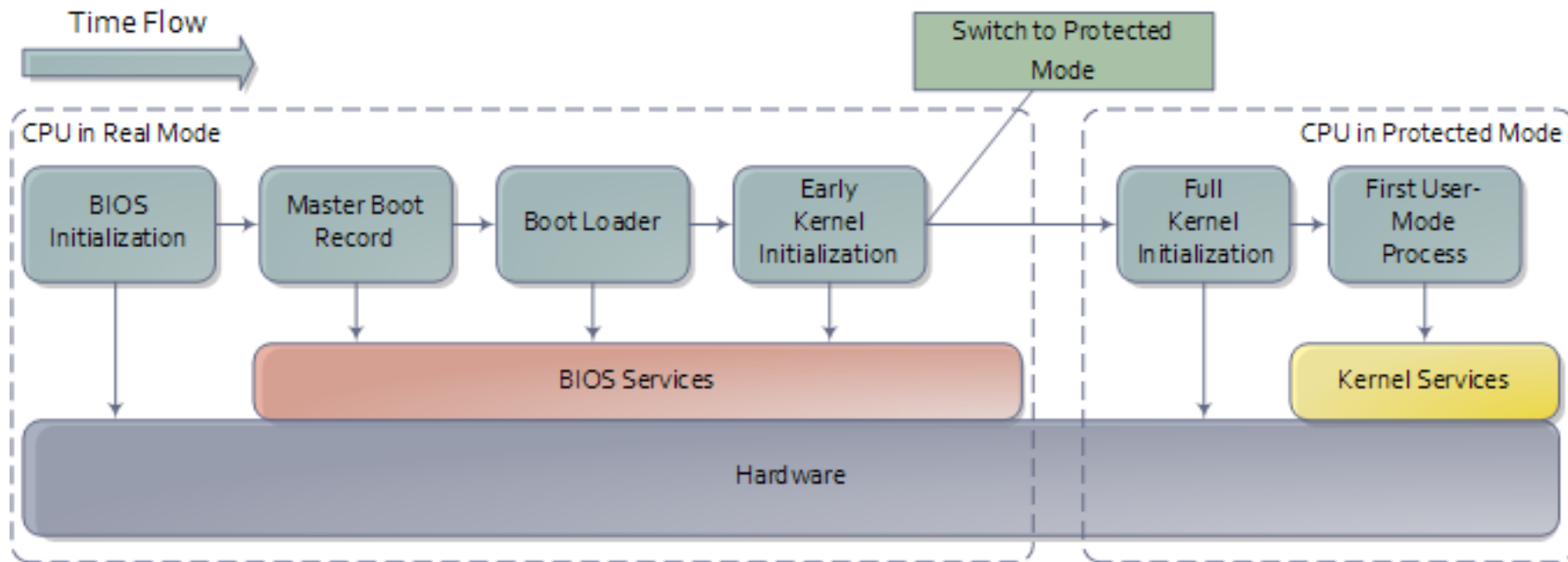
Absolute sector 0 (cylinder 0, head 0, sector 1)

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000   EB 48 90 00 00 00 00 00 00 00 00 00 00 00 00 00   .H..............
0010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 03 02   ................
0040   80 00 00 80 DF 0A 93 01 00 08 FA EA 50 7C 00 00   ............P|..
0050   31 C0 8E D8 8E D0 BC 00 20 FB A0 40 7C 3C FF 74   1....... ..@|<.t
0060   02 88 C2 52 BE 76 7D E8 34 01 F6 C2 80 74 54 B4   ...R.v}.4....tT.
0070   41 BB AA 55 CD 13 5A 52 72 49 81 FB 55 AA 75 43   A..U..ZRrI..U.uC
0080   A0 41 7C 84 C0 75 05 83 E1 01 74 37 66 8B 4C 10   .A|..u....t7f.L.
0090   BE 05 7C C6 44 FF 01 66 8B 1E 44 7C C7 04 10 00   ..|.D..f..D|....
00A0   C7 44 02 01 00 66 89 5C 08 C7 44 06 00 70 66 31   .D...f.\..D..pf1
00B0   C0 89 44 04 66 89 44 0C B4 42 CD 13 72 05 BB 00   ..D.f.D..B..r...
00C0   70 EB 7D B4 08 CD 13 73 0A F6 C2 80 0F 84 F3 00   p.}....s........
00D0   E9 8D 00 BE 05 7C C6 44 FF 00 66 31 C0 88 F0 40   .....|.D..f1...@
00E0   66 89 44 04 31 D2 88 CA C1 E2 02 88 E8 88 F4 40   f.D.1..........@
00F0   89 44 08 31 C0 88 D0 C0 E8 02 66 89 04 66 A1 44   .D.1......f..f.D
0100   7C 66 31 D2 66 F7 34 88 54 0A 66 31 D2 66 F7 74   |f1.f.4.T.f1.f.t
0110   04 88 54 0B 89 44 0C 3B 44 08 7D 3C 8A 54 0D C0   ..T..D.;D.}<.T..
0120   E2 06 8A 4C 0A FE C1 08 D1 8A 6C 0C 5A 8A 74 0B   ...L......l.Z.t.
0130   BB 00 70 8E C3 31 DB B8 01 02 CD 13 72 2A 8C C3   ..p..1......r*..
0140   8E 06 48 7C 60 1E B9 00 01 8E DB 31 F6 31 FF FC   ..H|`......1.1..
0150   F3 A5 1F 61 FF 26 42 7C BE 7C 7D E8 40 00 EB 0E   ...a.&B|.|}.@...
0160   BE 81 7D E8 38 00 EB 06 BE 8B 7D E8 30 00 BE 90   ..}.8.....}.O...
0170   7D E8 2A 00 EB FE 47 52 55 42 20 00 47 65 6F 6D   }.*...GRUB .Geom
0180   00 48 61 72 64 20 44 69 73 6B 00 52 65 61 64 00   .Hard Disk.Read.
0190   20 45 72 72 6F 72 00 BB 01 00 B4 0E CD 10 AC 3C    Error.........<
01A0   00 75 F4 C3 00 00 00 00 00 00 00 00 00 00 00 00   .u..............
01B0   00 00 00 00 00 00 00 00 A8 E1 A8 E1 00 00 80 01   ................
01C0   01 00 07 FE FF 6D 3F 00 00 00 AF 39 D7 00 00 00   .....m?....9....
01D0   C1 6E 0C FE FF FF EE 39 D7 00 BD 86 BB 00 00 FE   .n.....9........
01E0   FF FF 83 FE FF FF AB C0 92 01 CD 2F 03 00 00 FE   .........../....
01F0   FF FF 0F FE FF FF 78 F0 95 01 83 AF CC 00 55 AA   ......x.......U.
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

**Boot code** →

**Partition table** →

Disk MBR that uses GRUB 0.92/0.93 boot code

**MBR signature**

# End of Chapter 1

# Review Questions

1. PCIe vs. PCI, which one is in the north bridge, and which one uses the serial protocol? Why?

2. Explain the entire process of interrupt handling, from the firing of a hardware event until the first line of code of the corresponding ISR is executed

3. Consider read() and write(). Whose call latency is subject to the speed of the I/O device? (hint: blocking and non-blocking)

4. Describe the boot procedure of the legacy PC