CSCI320 – Operating Systems

Ahmad Fadlallah

Introduction

References

- These slides are based on the official slides of the following textbooks
  - **Operating Systems: Internals and Design Principles (9th Edition),** *William Stallings*, Pearson
  - **Operating System Concepts with Java (10th Edition)**, *Abraham Silberschatz, Peter B. Galvin and Greg Gagne*, Wiley

Objectives

- To describe the basic organization of computer systems
- To provide a grand tour of the major components of operating systems
- To give an overview of the many types of computing environments

What Does the Term Operating System Mean?

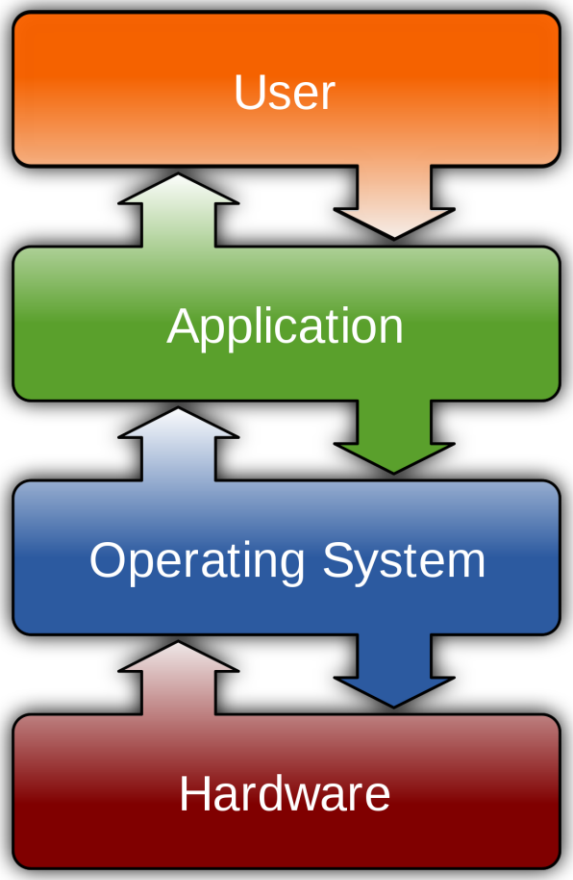
- An operating system is "_____"
- What about:
  - Car
  - Airplane
  - Printer
  - Washing Machine
  - Toaster
  - Compiler
  - Etc.

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

Computer System Structure

- **Components of a Computer System**
  - **Hardware**
    - Provides basic computing resources
      - CPU, memory, I/O devices
  - **Operating system**
    - Controls and coordinates use of hardware *among various applications and users*
  - **Application programs**
    - Define the ways in which the system resources are used to solve the computing problems of the users
      - Word processors, compilers, web browsers, database systems, video games
  - **Users**
    - People, machines, other computers



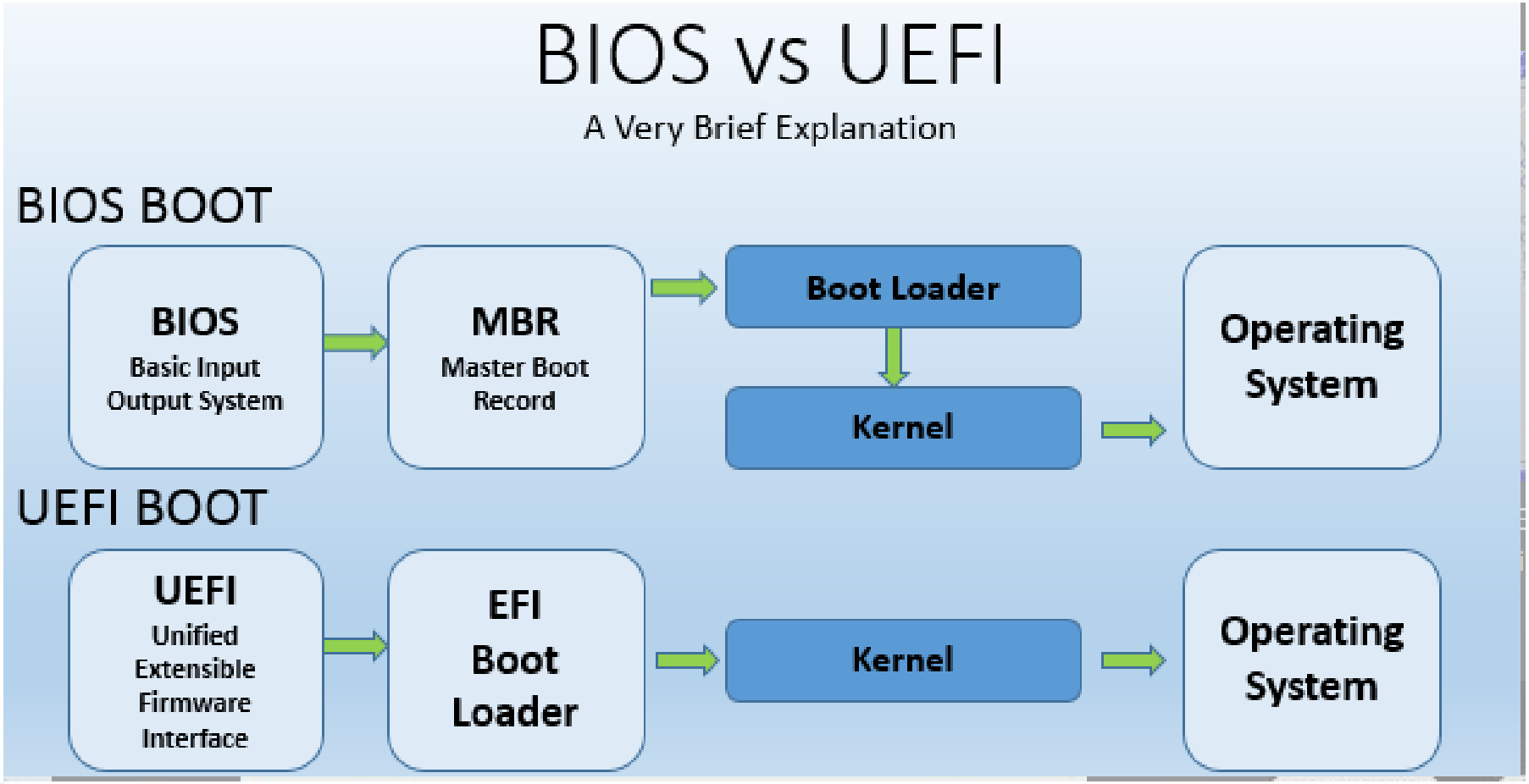
What Operating Systems Do?

- Depends on the point of view
- Users want *convenience*, *ease of use* and *good performance*
  - Don't care about *resource utilization*
- But shared computer (mainframe, minicomputer) must keep all users happy
- Users of dedicated systems (e.g., workstations) have dedicated resources but frequently use shared resources from servers
- Mobile devices (smartphones, tablets) are resource poor, optimized for usability and battery life
  - Mobile user interfaces such as touch screens, voice recognition
- Some computers have little or no user interface, such as embedded computers in devices and automobiles
  - Run primarily without user intervention

What Operating Systems Do?

- OS is a Resource Allocator
  - Manages all *resources*
  - Decides between *conflicting requests* for efficient and fair resource use
- OS is a Control Program
  - Controls execution of programs to **prevent** errors and improper use of the computer

Computer Startup



UEFI: Unified Extensible Firmware Interface

BIOS: Basic Input/Output System

Computer Startup

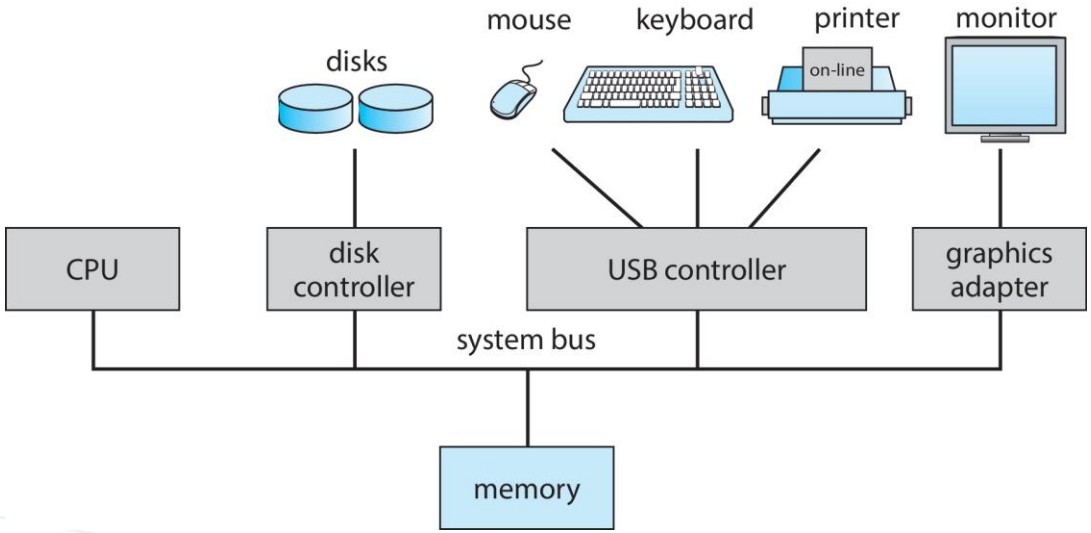
- **Bootstrap program** is loaded at power-up or reboot
  - Typically stored in *ROM/EPROM/EEPROM*, generally known as **firmware**
  - Initializes all aspects of system from *CPU registers* to *device controllers* to *memory contents*
  - Loads **operating system kernel** and starts execution
    - the bootstrap program must locate the operating-system kernel and load it into memory.

Computer Startup (cont'd)

- Once the kernel is loaded and executing, it can start providing services to the system and its users.
- Some services are provided outside of the kernel, by **system programs** that are loaded into memory at boot time to become **system processes**, or **system daemons** that run the entire time the kernel is running.
- Once this phase is complete, the system is fully booted, and the system **waits for some event** to occur.

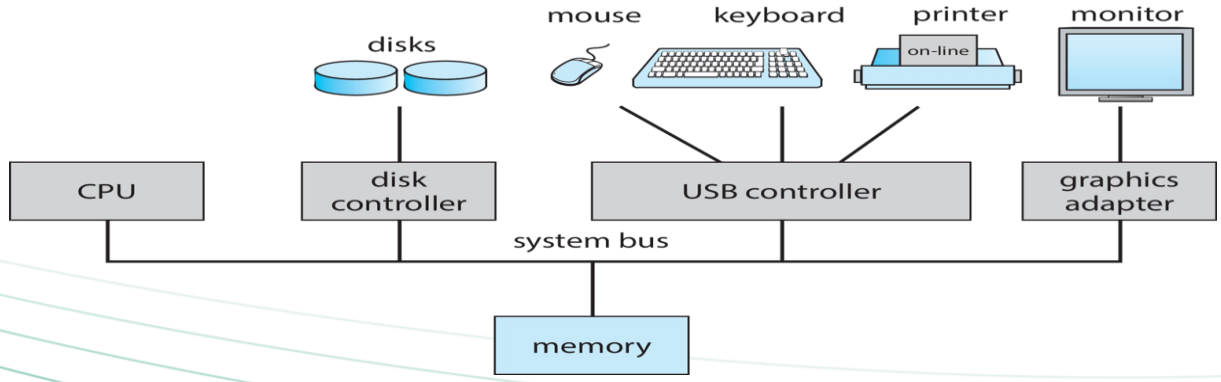
Computer System Organization

- *One or more CPUs*, *device controllers* connect through common bus providing access to shared memory
- ***Concurrent*** execution of CPU(s) and devices *competing* for ***memory cycles***



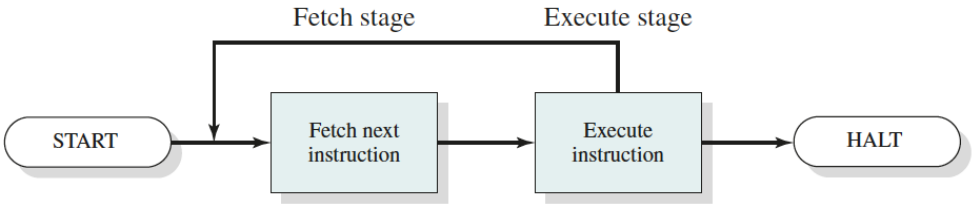
Computer-System Operation

- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- Each device controller type has an operating system device driver to manage it
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**



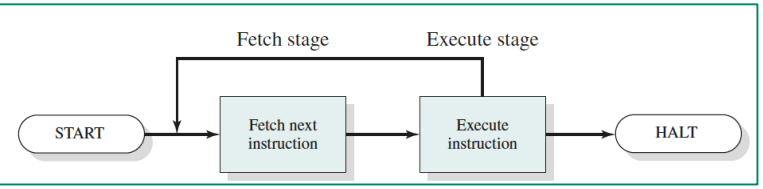
Instruction Execution

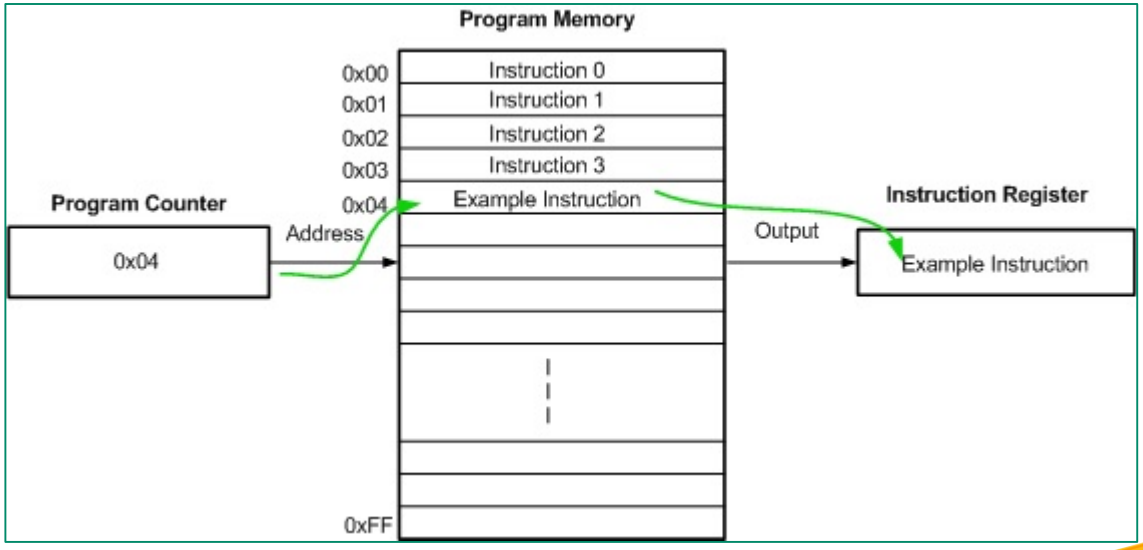
- A **program** to be executed by a processor consists of a **set of instructions** stored in memory.
- In its simplest form, instruction processing consists of two steps:
  - The processor reads (**fetches**) instructions from memory one at a time and **executes** each instruction.
- Program execution consists of repeating the process of **instruction fetch** and **instruction execution**.
  - Instruction execution may involve several operations and depends on the nature of the instruction.
- **Program execution halts** only if the processor is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the processor is encountered.



Instruction Execution (cont'd)

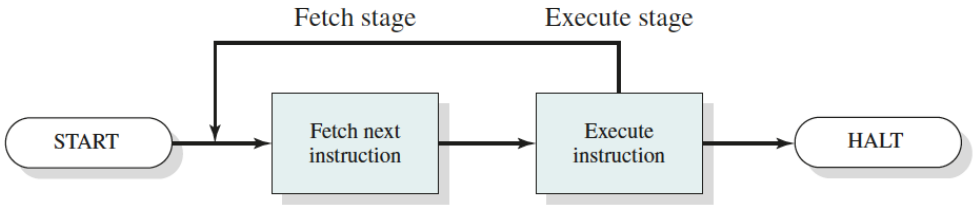
- At the beginning of each instruction cycle, the processor fetches an instruction from memory.
- **The Program Counter (PC)** holds the address of the next instruction to be fetched.
  - Unless instructed otherwise, the processor always increments the PC after each instruction fetch
- The fetched instruction is loaded into the **Instruction register (IR)**.





Instruction Execution (cont'd)

- **Instruction Action Categories**
  - **Processor-memory**: Data may be transferred from processor to memory, or from memory to processor.
  - **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
  - **Data processing:** The processor may perform some arithmetic or logic operation on data.
  - **Control:** An instruction may specify that the sequence of execution be altered.
    - Example: Assigning a new address to fetch the instruction from



Interrupts

- The occurrence of an event is usually signaled by an **interrupt** from either the *hardware* or the *software*.
  - **Hardware** may trigger an interrupt at any time by **sending a signal to the CPU**, usually by way of the *system bus*.
  - **Software** may trigger an interrupt by executing a **special operation** called a **system call** (also called a **monitor call**).

Interrupts – Main Classes

Program

- **Generated by some condition that occurs as a result of an instruction execution**
- Arithmetic overflow, division by zero, reference outside user's allowed memory space.

Timer

- **Generated by a timer within the processor.**
- This allows the operating system to perform certain functions on a regular basis.

I/O

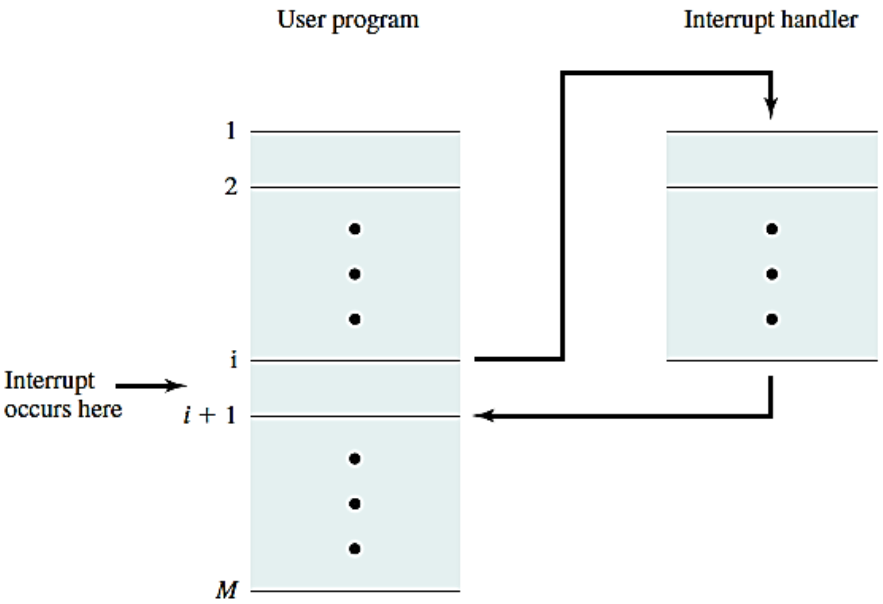
- **Generated by an I/O controller**, to signal normal completion of an operation or to signal a variety of error conditions.

Hardware failure

- **Generated by a failure**, such as power failure or memory parity error.

Interrupts

- *When the CPU is interrupted*, it **stops what it is doing and immediately transfers execution to a fixed location**.
  - The fixed location usually contains the starting address where the service routine for the interrupt **(Interrupt handler)** is located.
  - The interrupt service routine executes
  - On completion, the CPU **resumes the interrupted computation**.

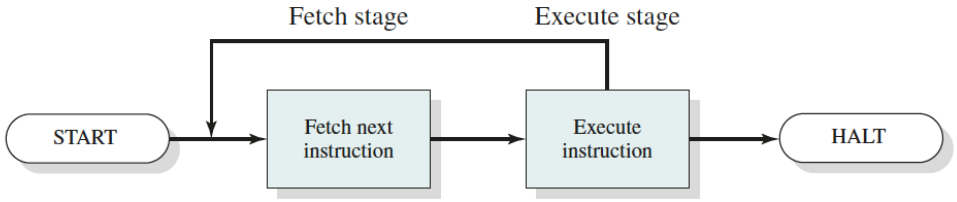


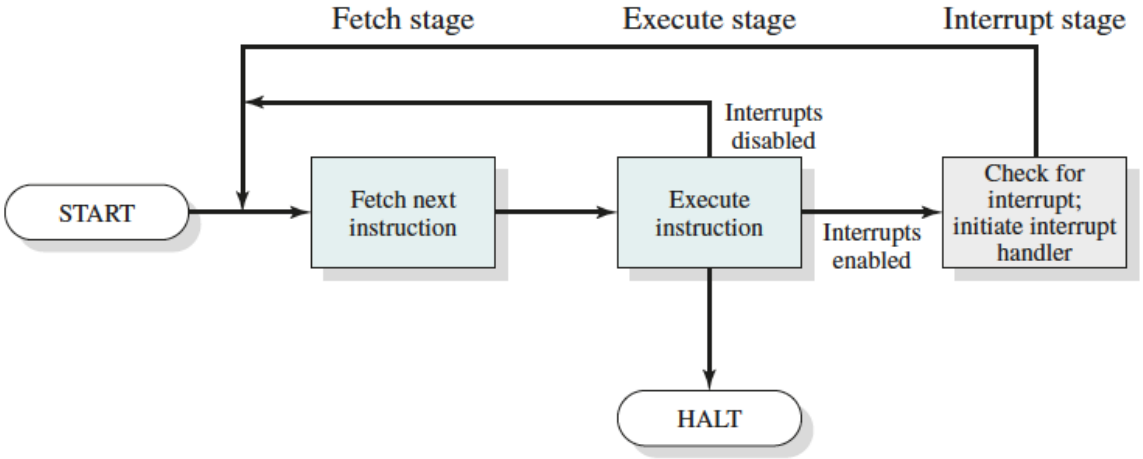
Interrupts (cont'd)

- **Why interrupts are needed?**
  - Interrupts are provided primarily as a way to **improve processor utilization**.
  - *Most I/O devices are much slower than the processor*.
    - **Example1: Writing Data on a HDD**
      - A 1 GHz Processor is 4 millions times faster than a typical HDD (7200 RPM)
    - **Example2: Printing Data**
      - A processor is transferring data to a printer using the instruction cycle scheme (Fetch-Execute).
        - ✓ After each write operation, the processor must pause and remain idle until the printer catches up.
        - ✓ The length of this pause may be on the order of many thousands or even millions of instruction cycles => very wasteful use of the processor.

Interrupts (cont'd)

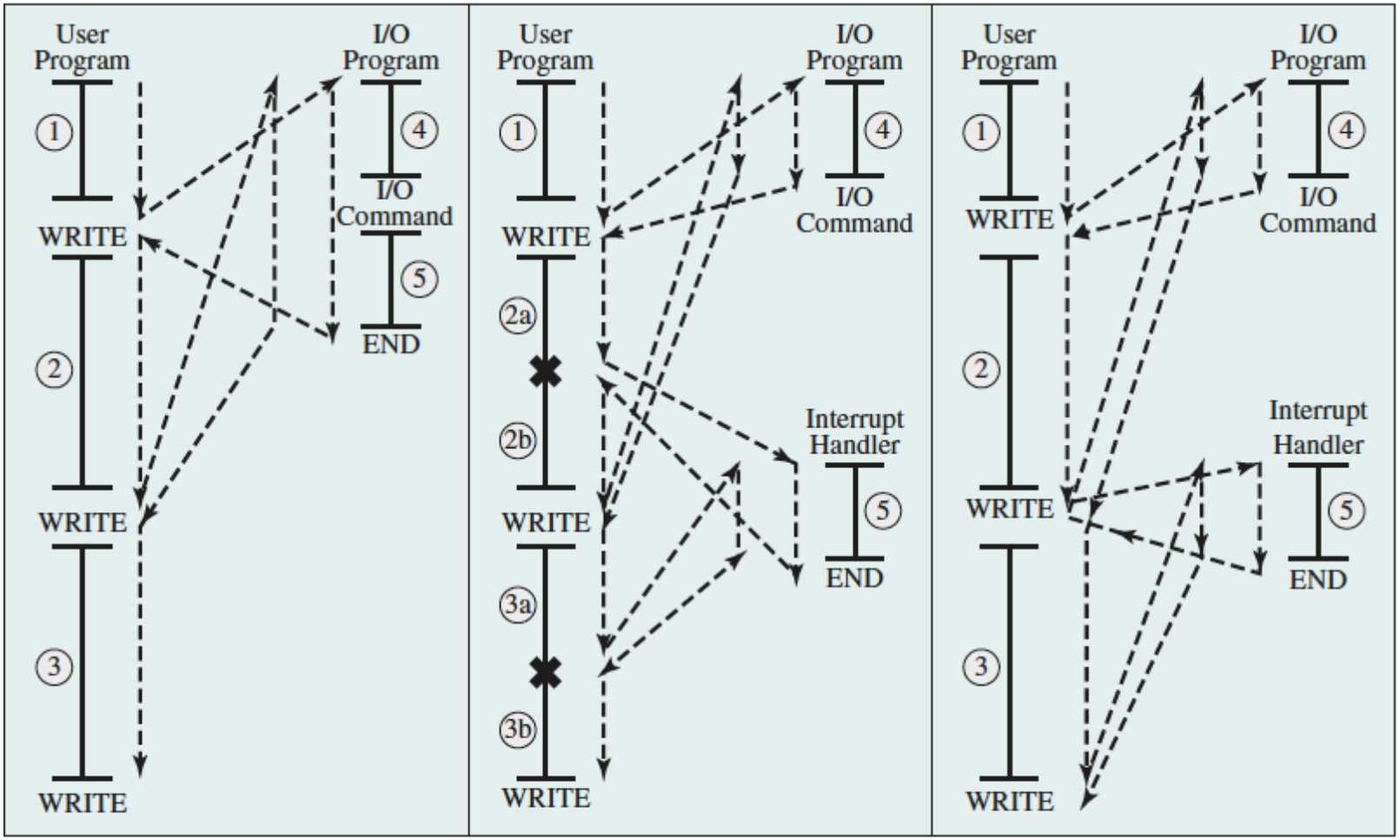
Instruction Cycle without Interrupts





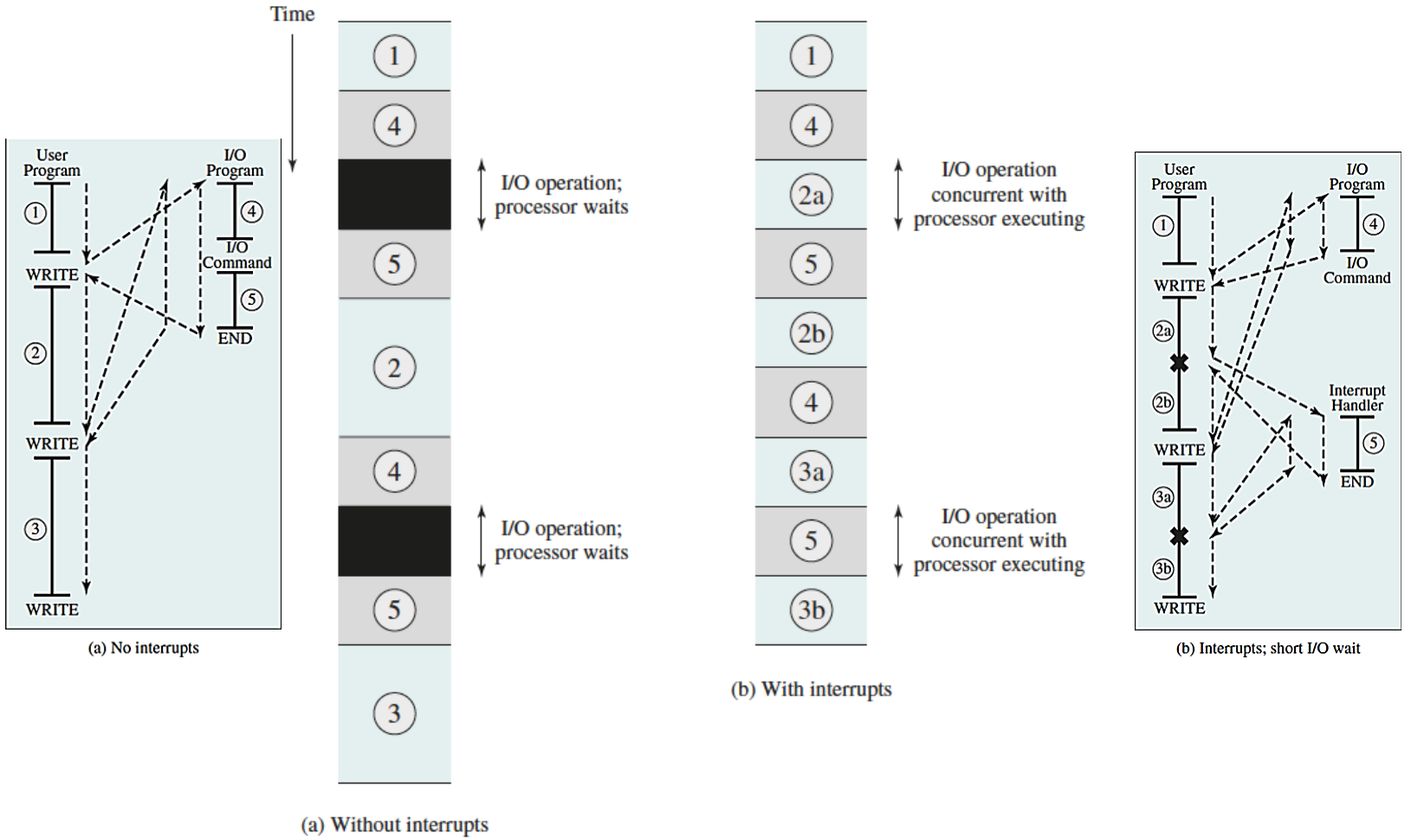
Instruction Cycle with Interrupts

Interrupts (cont'd)

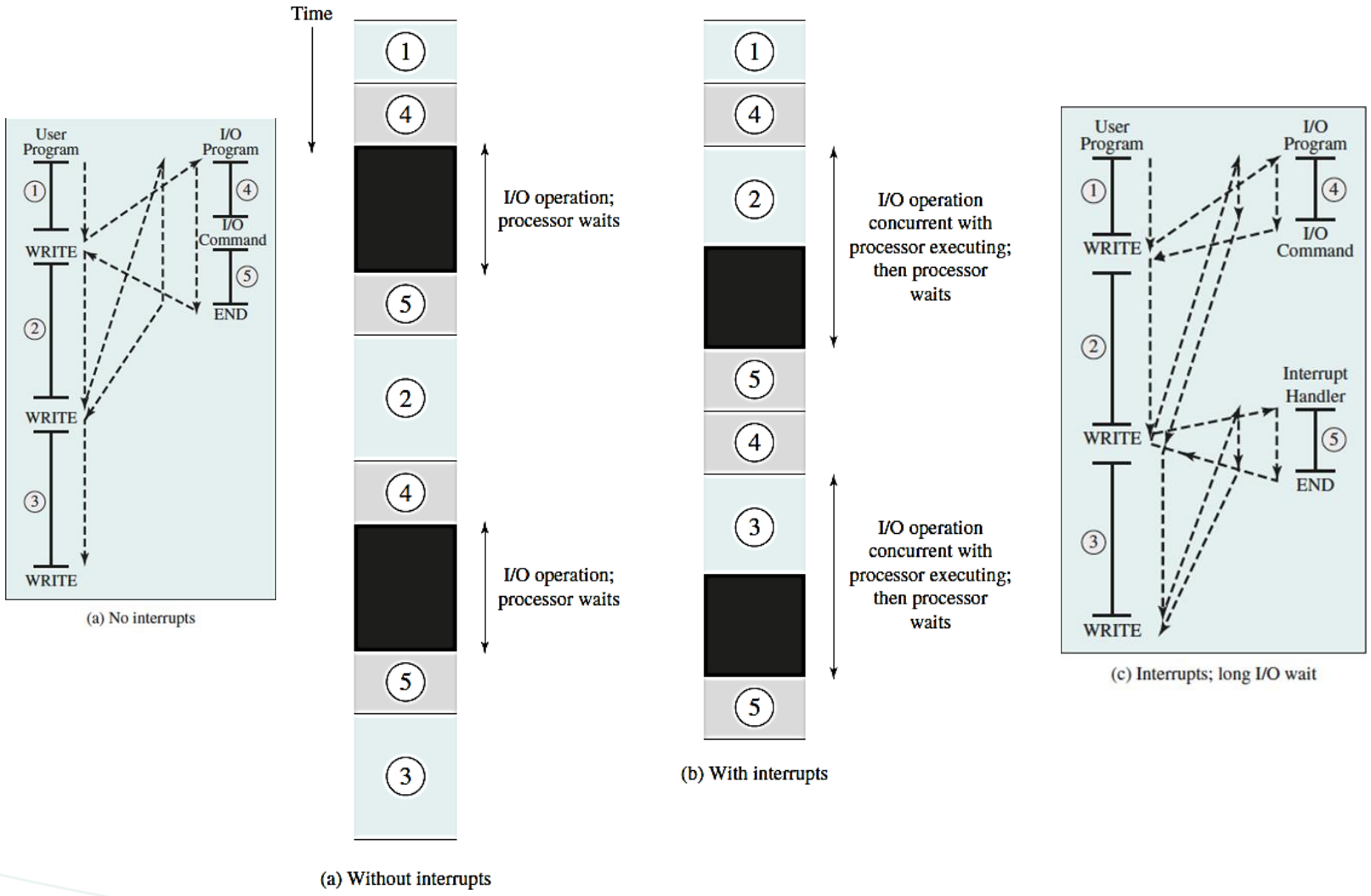


(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

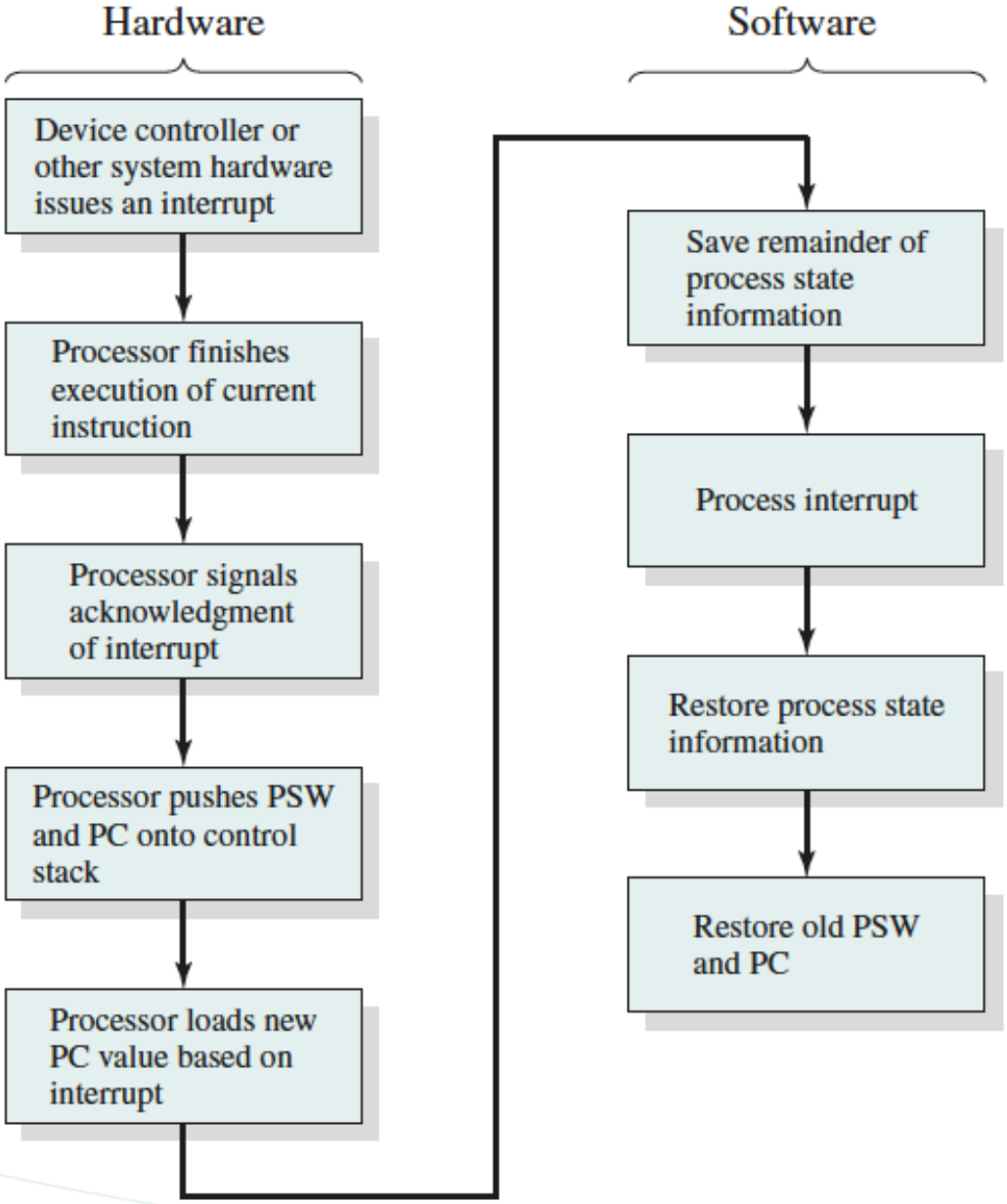
Interrupts (cont'd)



Interrupts (cont'd)

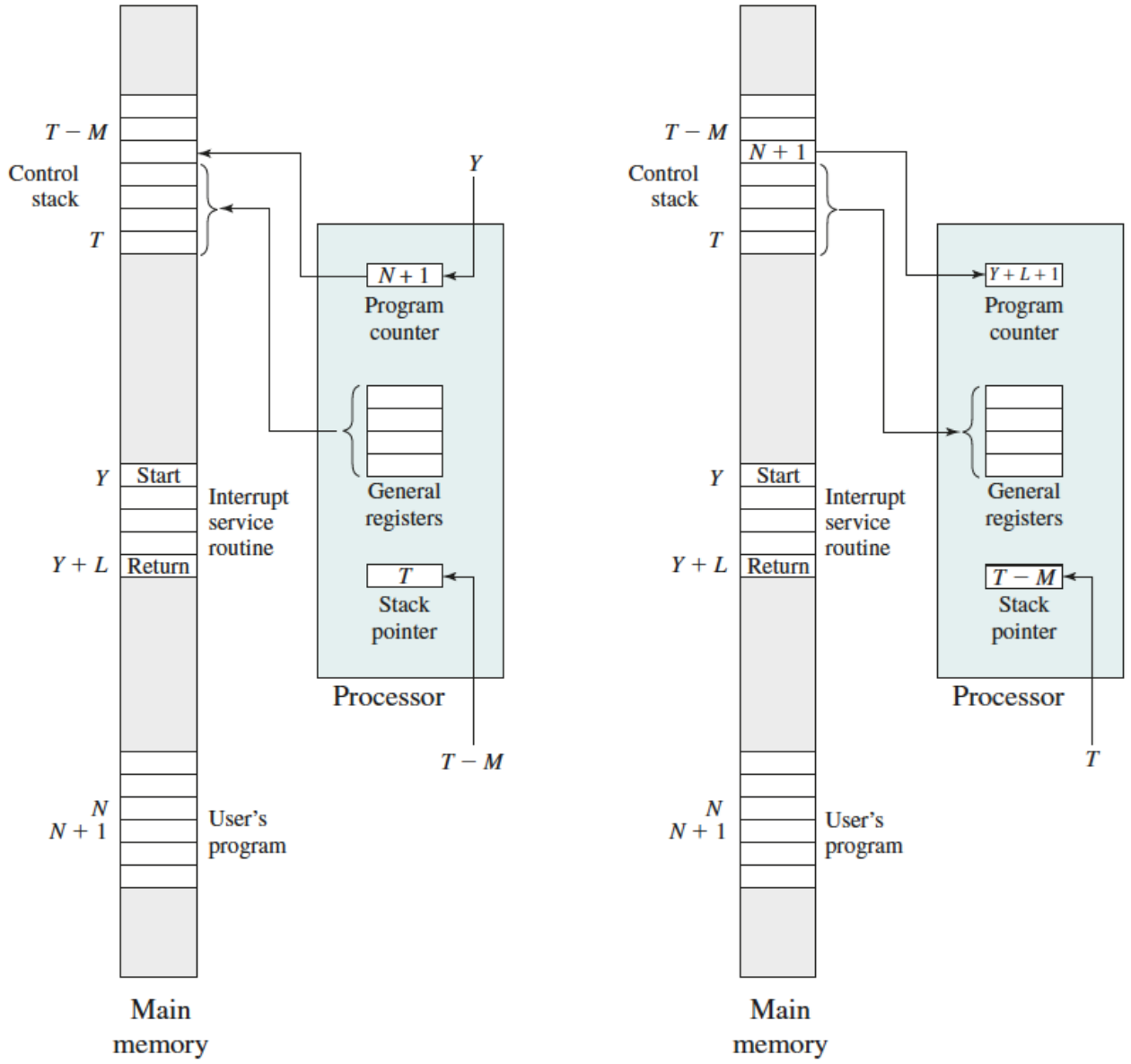


Interrupt processing



The **PSW (Program Status Word)** contains status information about the currently running process, including memory usage information, condition codes, and other status information

Interrupt processing



(a) Interrupt occurs after instruction at location N

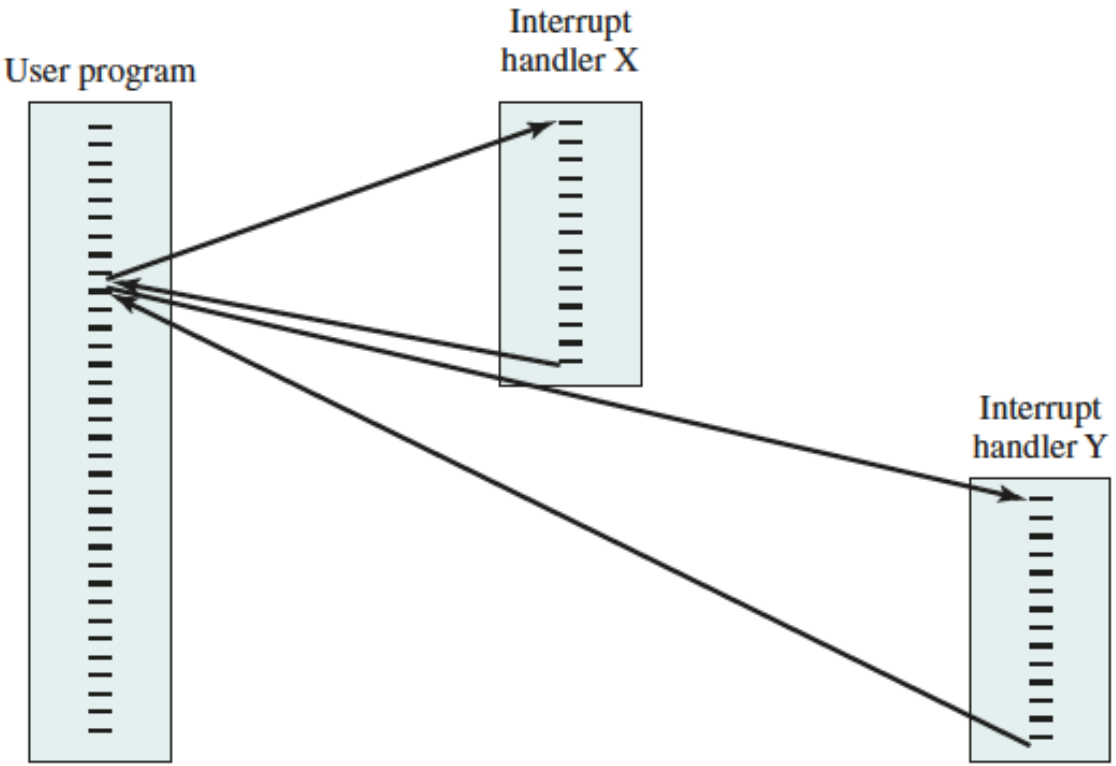
(b) Return from interrupt

Multiple Interrupts

- **Sequential Interrupt Handling**
  - Disable interrupts while treating an interrupt
  - Interrupts occurring while handling the current interrupt will remain pending
  - Re-enable the interrupt after finishing the interrupt handling
  - Disadvantages
    - Doesn't take into consideration relative priority or time-critical needs
    - Examples: communication data interrupts while printing

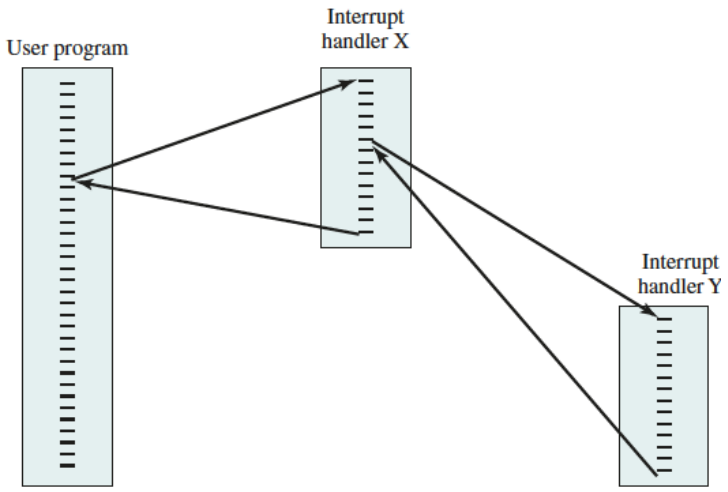
Multiple Interrupts

- **Sequential Interrupt Handling**

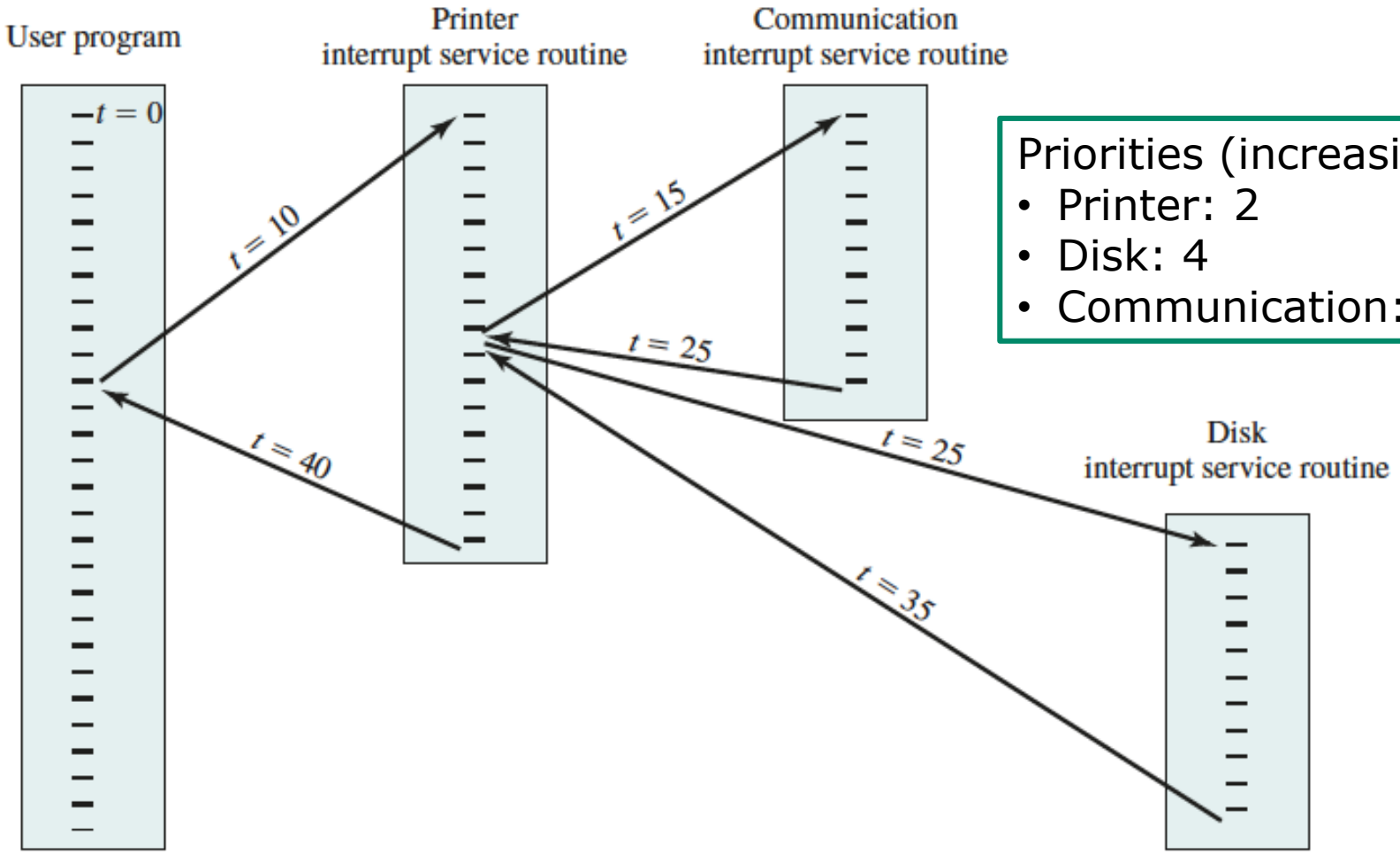


Multiple Interrupts (cont'd)

- **Nested interrupt processing**
  - Define priorities
  - When an interrupt occurs while handling another interrupt, the **priority of the new interrupt** determines if it will be immediately processed or not



Multiple Interrupts (cont'd)



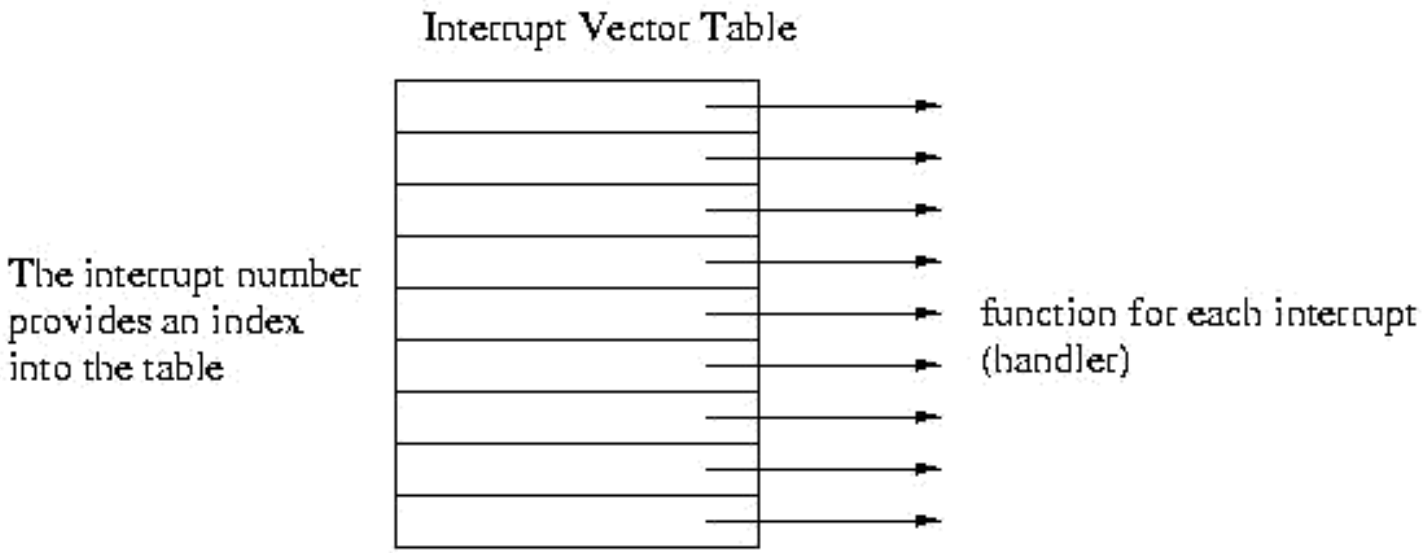
Priorities (increasing)
- Printer: 2
- Disk: 4
- Communication: 5

Interrupt processing

- **How interrupt handler is determined?**
  - Depending on computer architecture and OS design, there may be:
    - **Single program**
    - **One for each type of interrupt**
    - **One for each device and each type of interrupt**.
- If there is more than one interrupt-handling routine, the processor must determine which one to invoke.
  - Information **included in the original interrupt signal**, or
  - The **processor may have to issue a request to the device** that issued the interrupt to request the needed information.

Interrupt processing (cont'd)

- An **"Interrupt Vector Table" (IVT)** is a data structure that associates a list of interrupt handlers with a list of interrupt requests in a table of **interrupt vectors**.
  - An entry in the interrupt vector is the address of the interrupt handler.



Vectored vs. Non-vectored interrupt

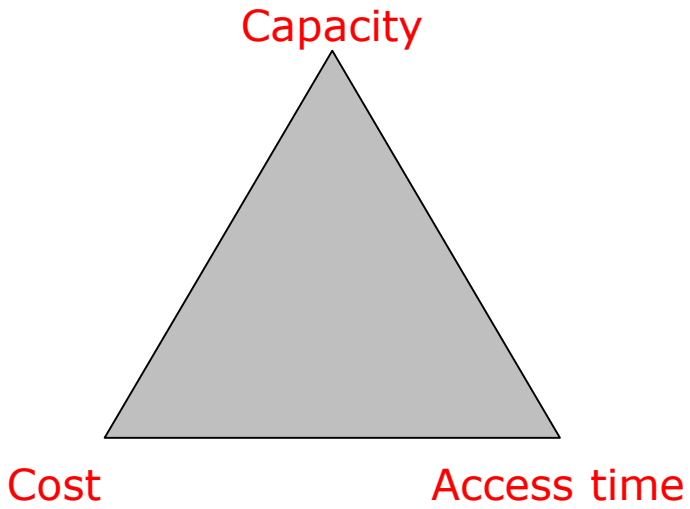
- A **Vectored Interrupt** is where the CPU actually knows the address of the **Interrupt Service Routine (ISR)** in advance.
- All it needs is that the interrupting device sends its unique **vector** via a data bus and through its I/O interface to the CPU.
- The CPU: (1) takes this vector, (2) checks an interrupt table in memory, and then (3) carries out the correct ISR for that device.
- The vectored interrupt allows the CPU to be able to know what ISR to carry out in **software** (memory).

Vectored vs. Non-vectored interrupt

- A **Non-vectored Interrupt** (also called **polled interrupt**) is where the interrupting device **never sends an interrupt vector**.
- Non-vectored interrupt has a **common ISR**, which is *common to all non-vectored interrupts in the system*.
  - Address of this common ISR is known to the CPU.
- It is a specific type of *I/O interrupt* that notifies the part of the computer containing the I/O interface that a device is ready to be read or otherwise handled but does not indicate which device.
- The interrupt controller **must poll** (send a signal out to) each device to determine which one made the request.

Memory Hierarchy

- Memory design constraints
  - How much?
  - How fast?
  - How expensive?

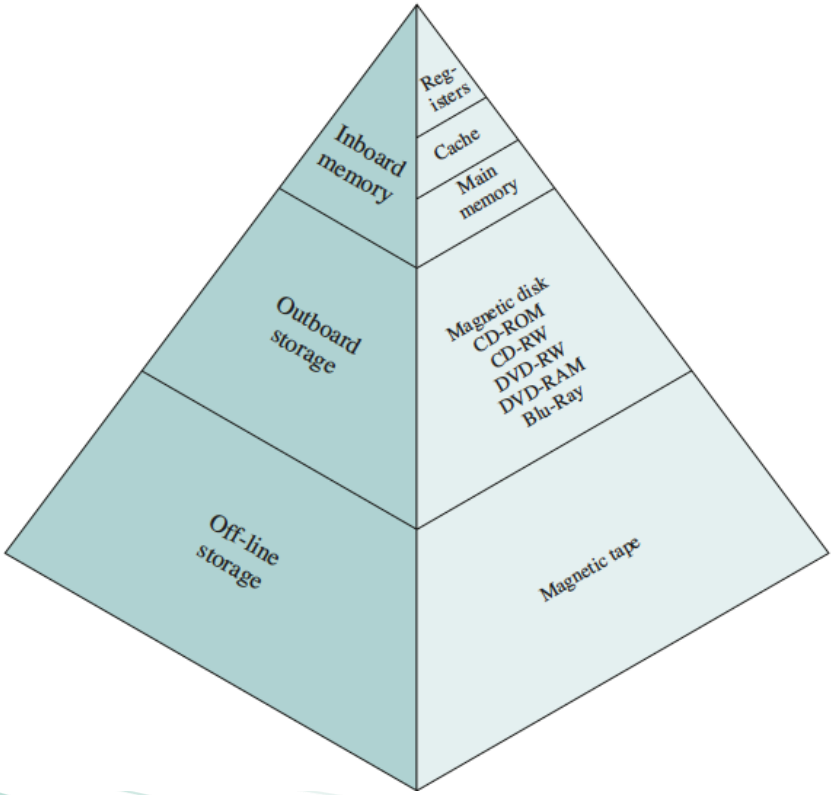


- Tradeoffs between capacity, access time and cost
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access speed

- **How to solve the dilemma?**

Memory Hierarchy (cont'd)

- **Solution**: Not rely on a single memory component or technology, but to employ a **memory hierarchy**



Decreasing *cost per bit*

Increasing *capacity*

Increasing *access time*

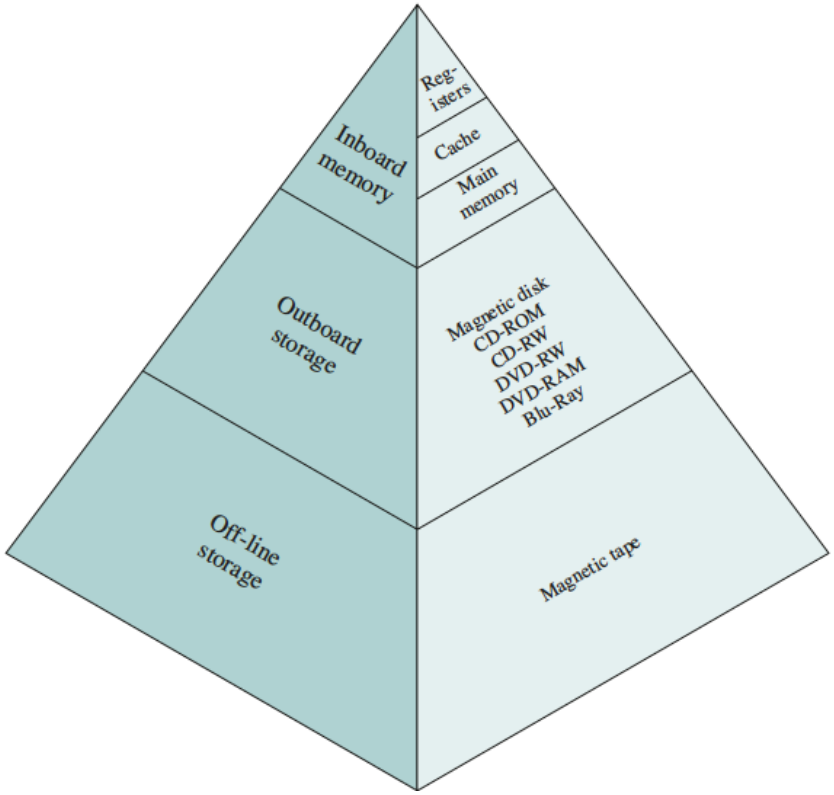
Decreasing *frequency of access* to the memory by the processor

Memory Hierarchy (cont'd)

- **Decreasing frequency of access to the memory by the processor**
  - Principle of **Locality of Reference**
    - During the course of execution of a program, **memory references** (instructions and data) **by the processor**, **tend to cluster**
    - Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references

Memory Hierarchy (cont'd)

- **Secondary Memory**
  - Auxiliary memory
  - External
  - Non-volatile
  - Used to store program and data files

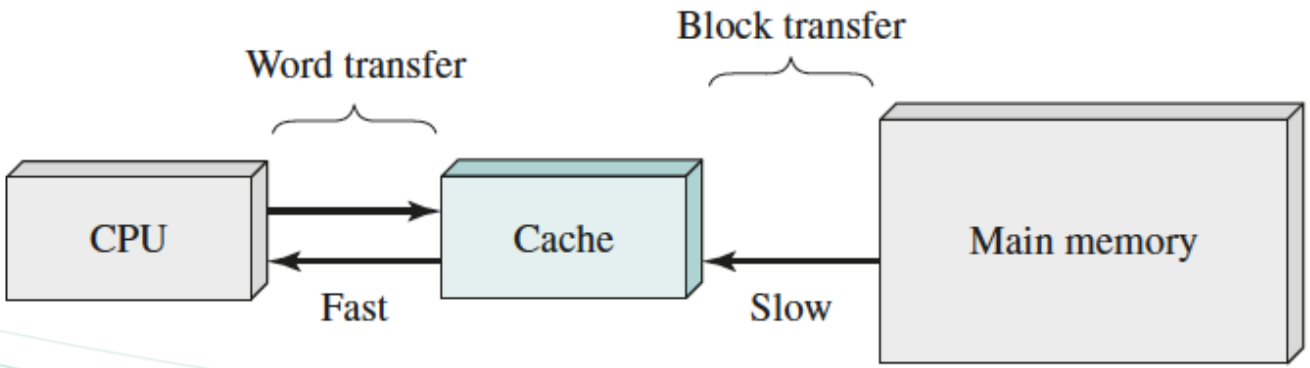


Cache Memory: Motivation

- On all instruction cycles, **the processor accesses memory at least once**, **to fetch the instruction**, and often one or more additional times, to fetch operands and/or store results.
- The **rate** at which the processor can execute instructions is clearly **limited by the memory cycle time** (The time it takes to read one word from or write one word to memory).
- **This limitation has been a significant problem** because of the persistent mismatch between processor and main memory speeds
- **Solution**: Exploit the **principle of locality** by providing a **small, fast memory between the processor and main memory**, namely the **cache**

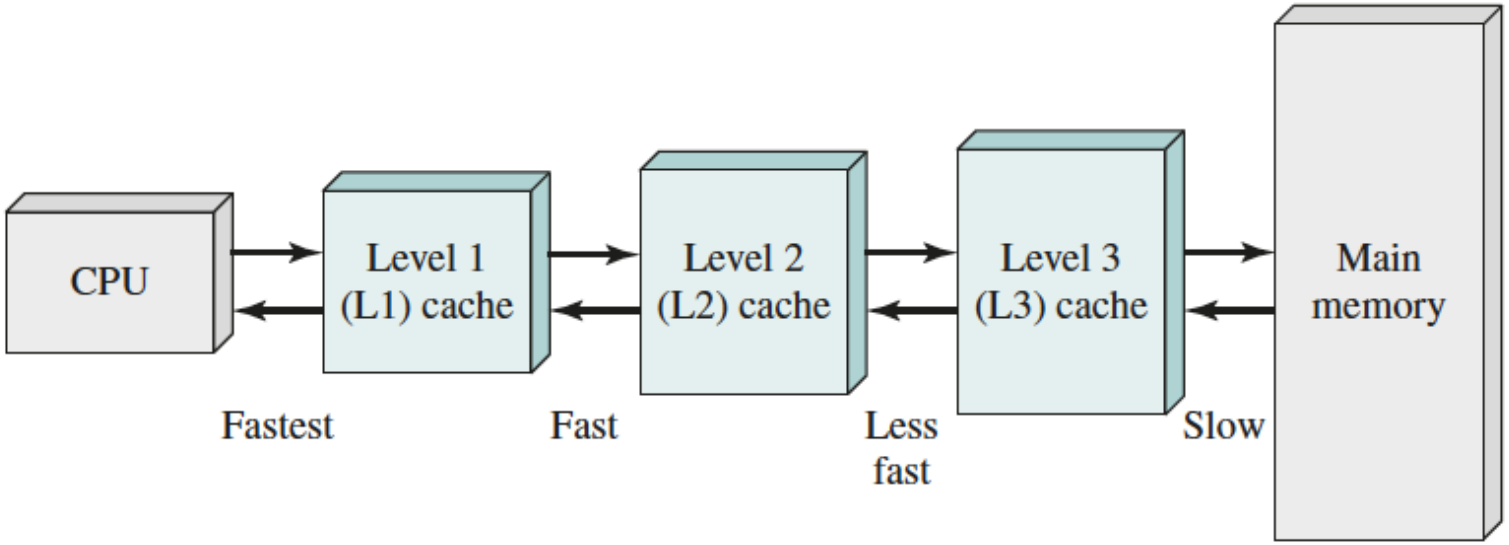
Cache Memory: Principle

- Objectives
  - Provide **memory access time** approaching that of the fastest memories available
  - Support a **large memory size** that has the price of less expensive types of semiconductor memories.

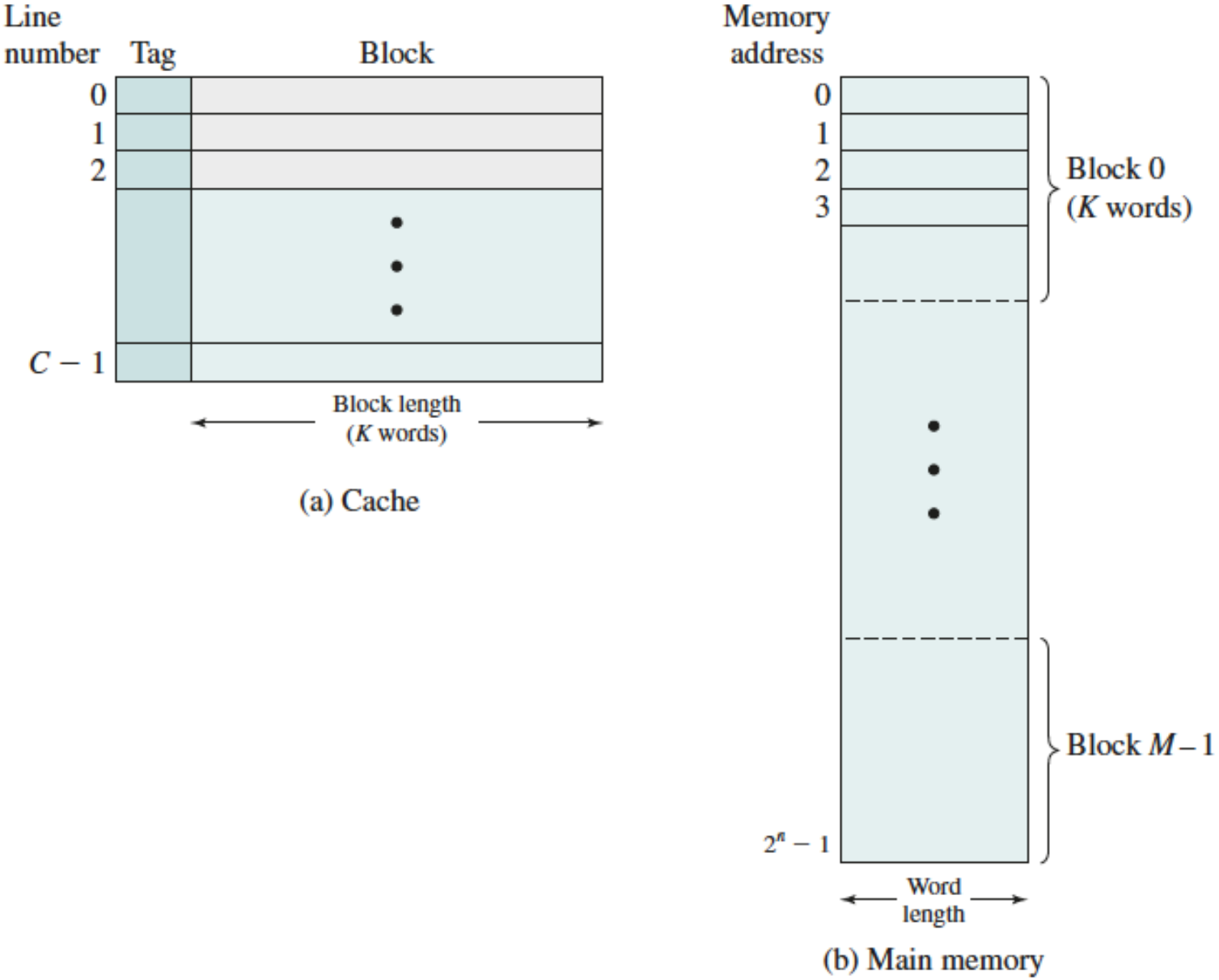


Cache Memory: Principle

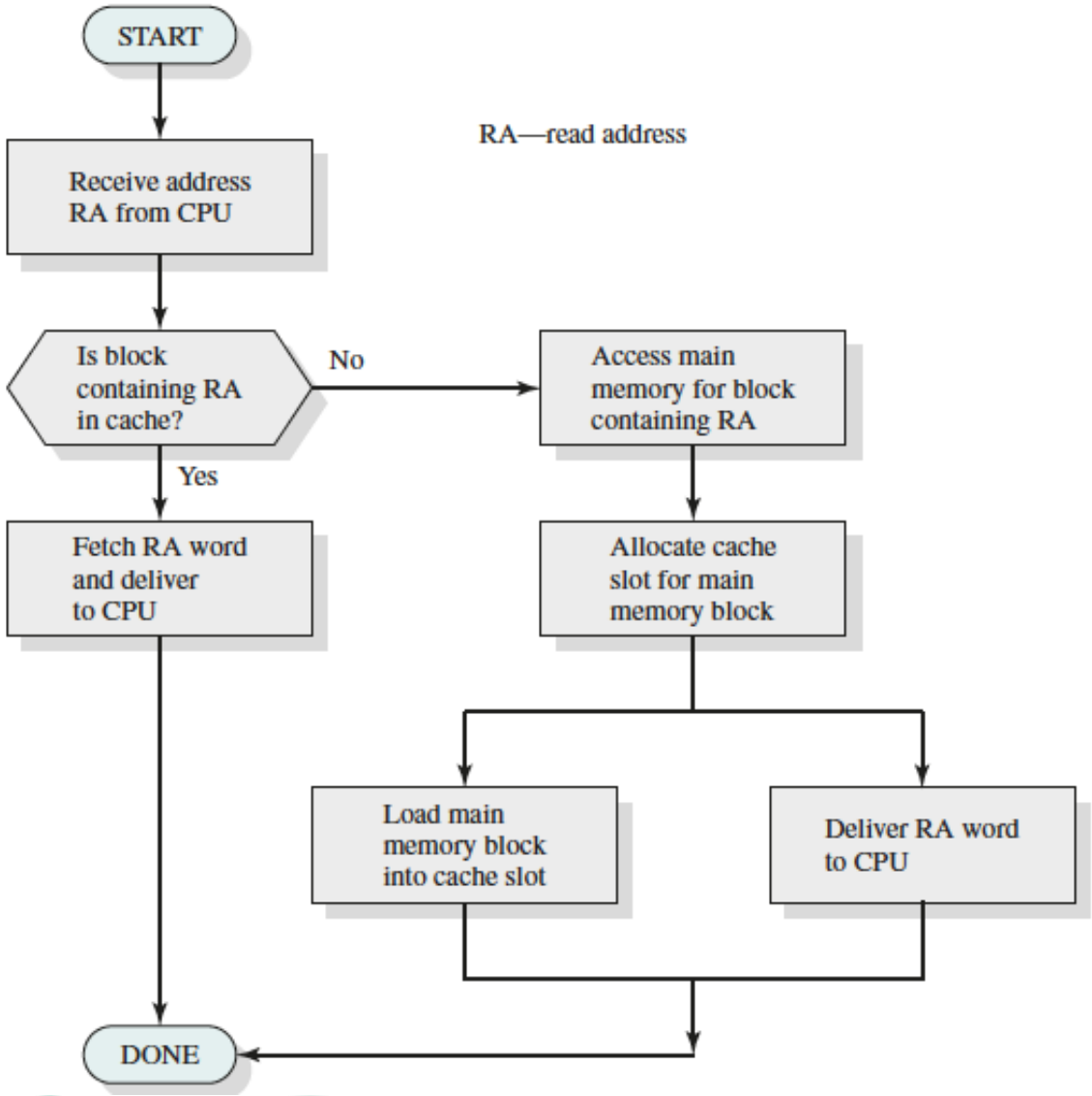
- Multi-level caching



Cache Memory



Cache Memory



Cache Principles

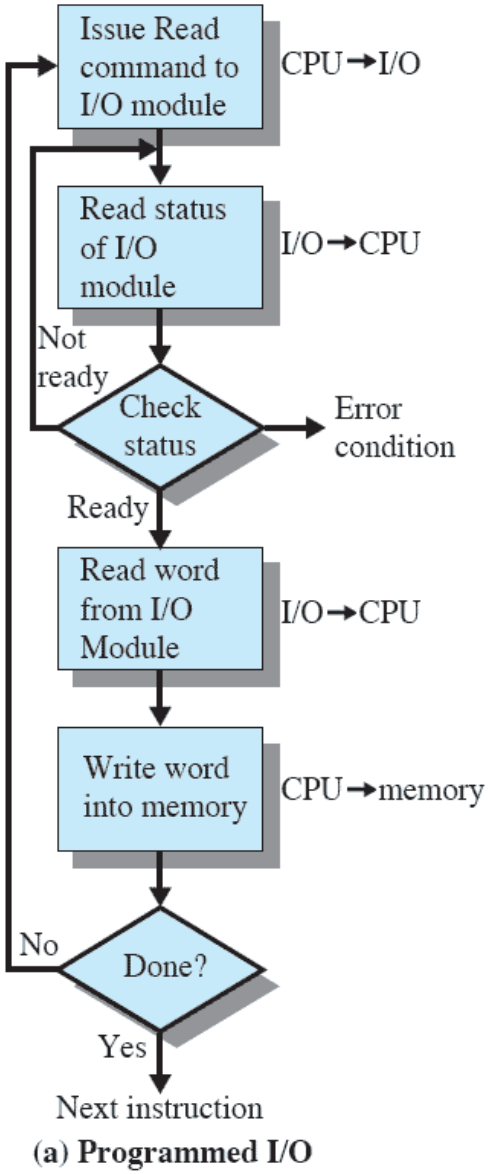
- **Cache size:** Even small caches have significant impact on performance
- **Block size:** The unit of data exchanged between cache and main memory
- **Mapping function:** Determines which cache location the block will occupy
- **Replacement algorithm:** Chooses which block to replace; e.g., Least-recently-used (LRU) algorithm
- **Write policy**
  - If the contents of a block in the cache are altered, then it is necessary to write it back to main memory before replacing it.
  - Dictates **when** the memory write operation takes place
  - Can occur every time the block is updated
  - Can occur when the block is replaced
    - Minimize write operations
    - Leave main memory in an obsolete state

I/O Operations

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module
- Three possible techniques for I/O operations:
  - Programmed I/O
  - Interrupt-driven I/O
  - Direct Memory Access (DMA).

Programmed I/O

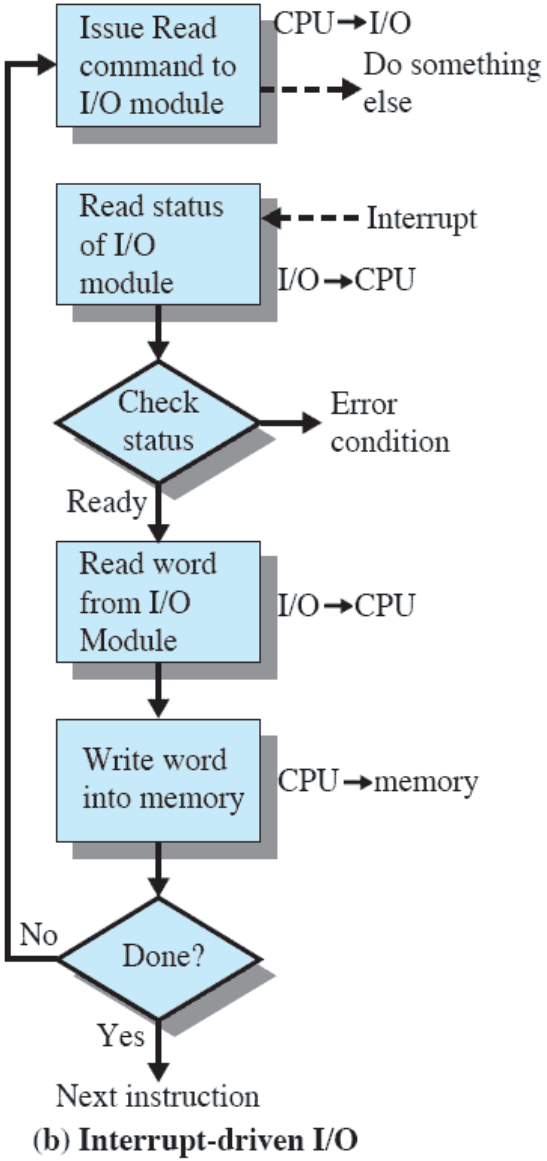
- I/O module performs the action, not the processor
- Sets the appropriate bits in the I/O status register
- No interrupts occur
- Processor checks status until operation is complete
- Degradation of the processor performance because of long wait time



(a) Programmed I/O

Interrupt-Driven I/O

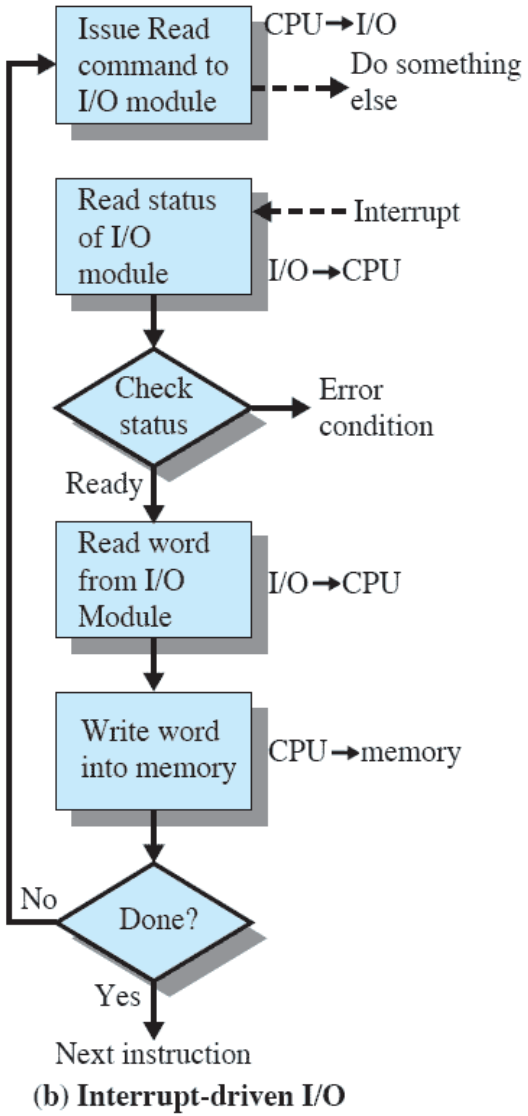
- The processor issues an I/O command to a module and then go on to do some other useful work
- **Processor is interrupted** when I/O module ready to exchange data
- Processor saves context of program executing and begins executing interrupt-handler



(b) Interrupt-driven I/O

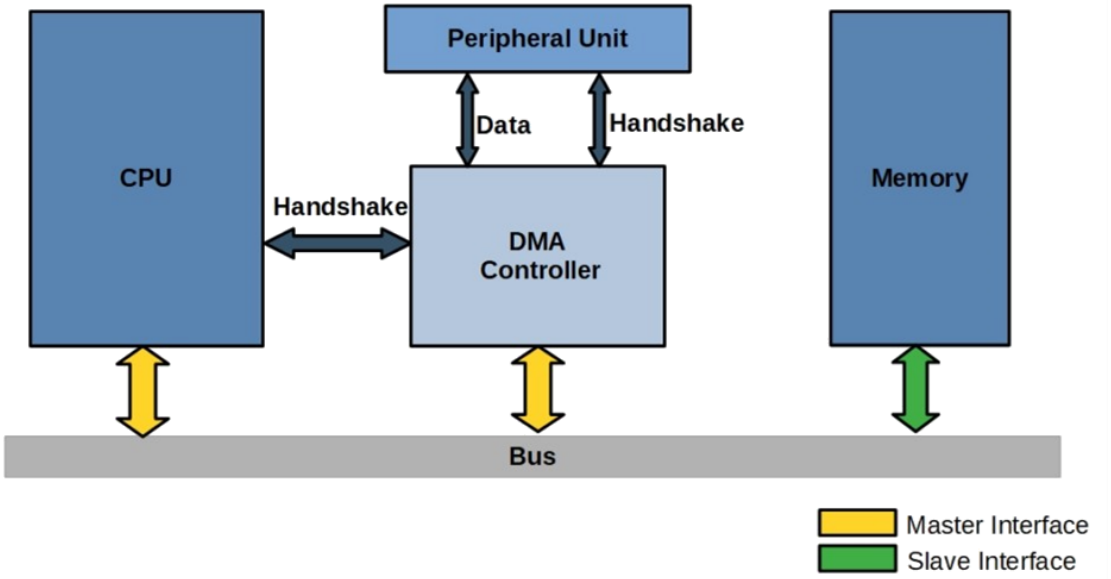
Interrupt-Driven I/O

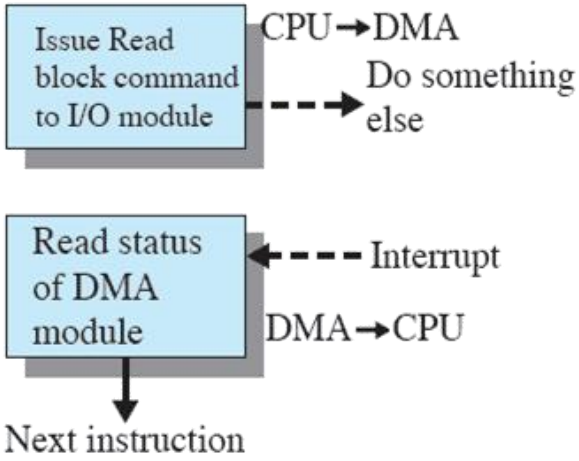
- No needless waiting
- Consumes a lot of processor time because every word read or written passes through the processor



(b) Interrupt-driven I/O

Direct Memory Access (DMA)



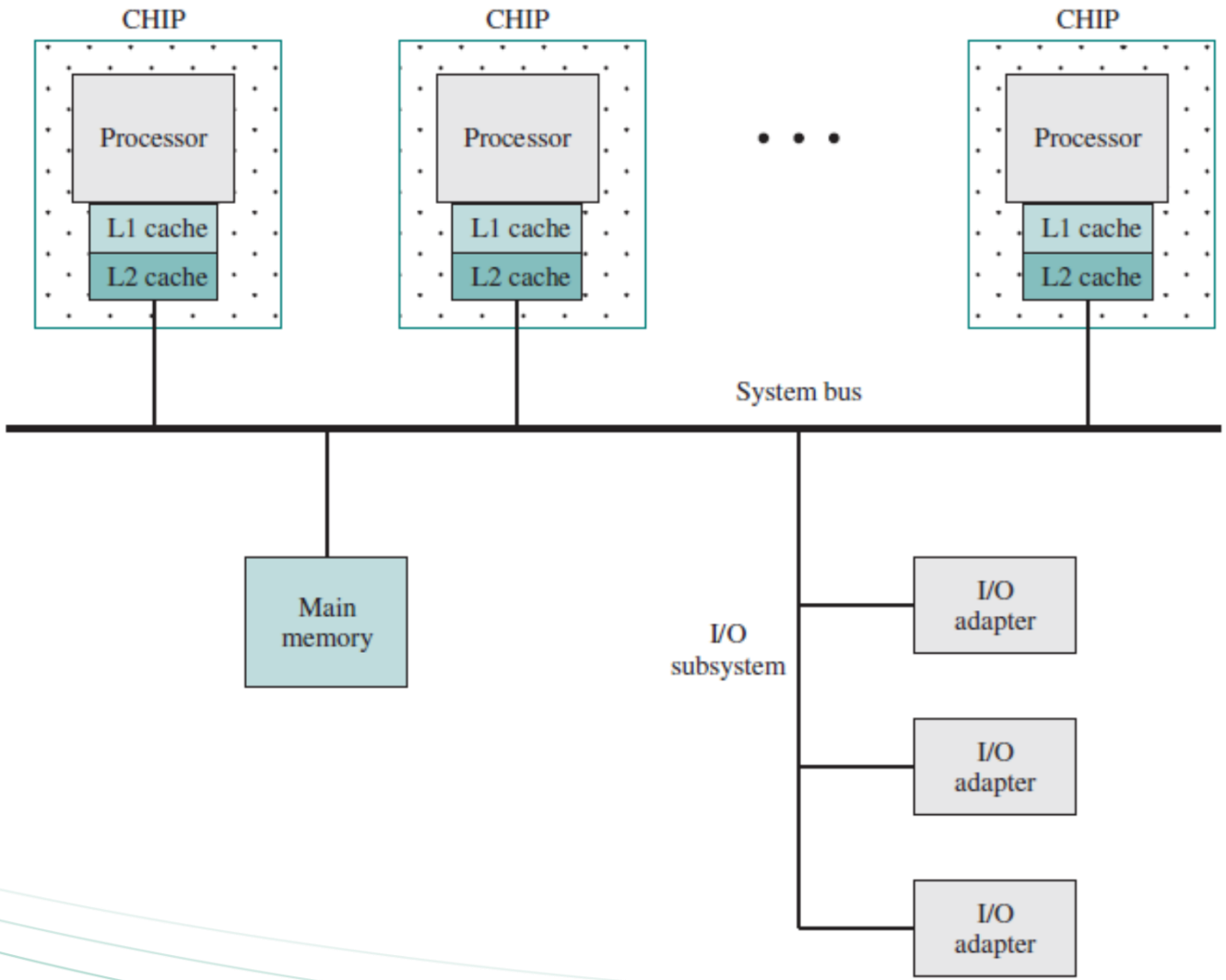


- The DMA function can be performed by a separate module on the system bus or it can be incorporated into an I/O module.
- Transfers a block of data directly to or from memory
- An interrupt is sent when the transfer is complete
- More efficient

Multiprocessor and multicore organization

- Most popular approaches to providing parallelism by replicating processors:
  - Symmetric Multi-Processors (SMPs)
  - Multicore computers
  - Clusters

Symmetric Multiprocessors (SMP)



Symmetric Multiprocessors (SMP)

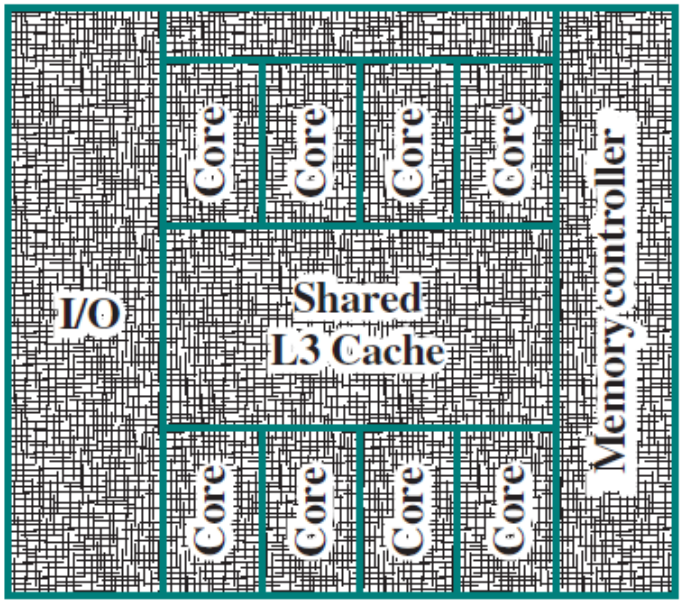
- A stand-alone computer system with the following characteristics:
  - Two or more similar processors of comparable capability
  - Processors share the same main memory and are interconnected by a bus or other internal connection scheme
  - Processors share access to I/O devices
  - All processors can perform the same functions
  - The system is controlled by an integrated operating system that provides interaction between processors and their programs at the job, task, file, and data element levels

SMP Advantages

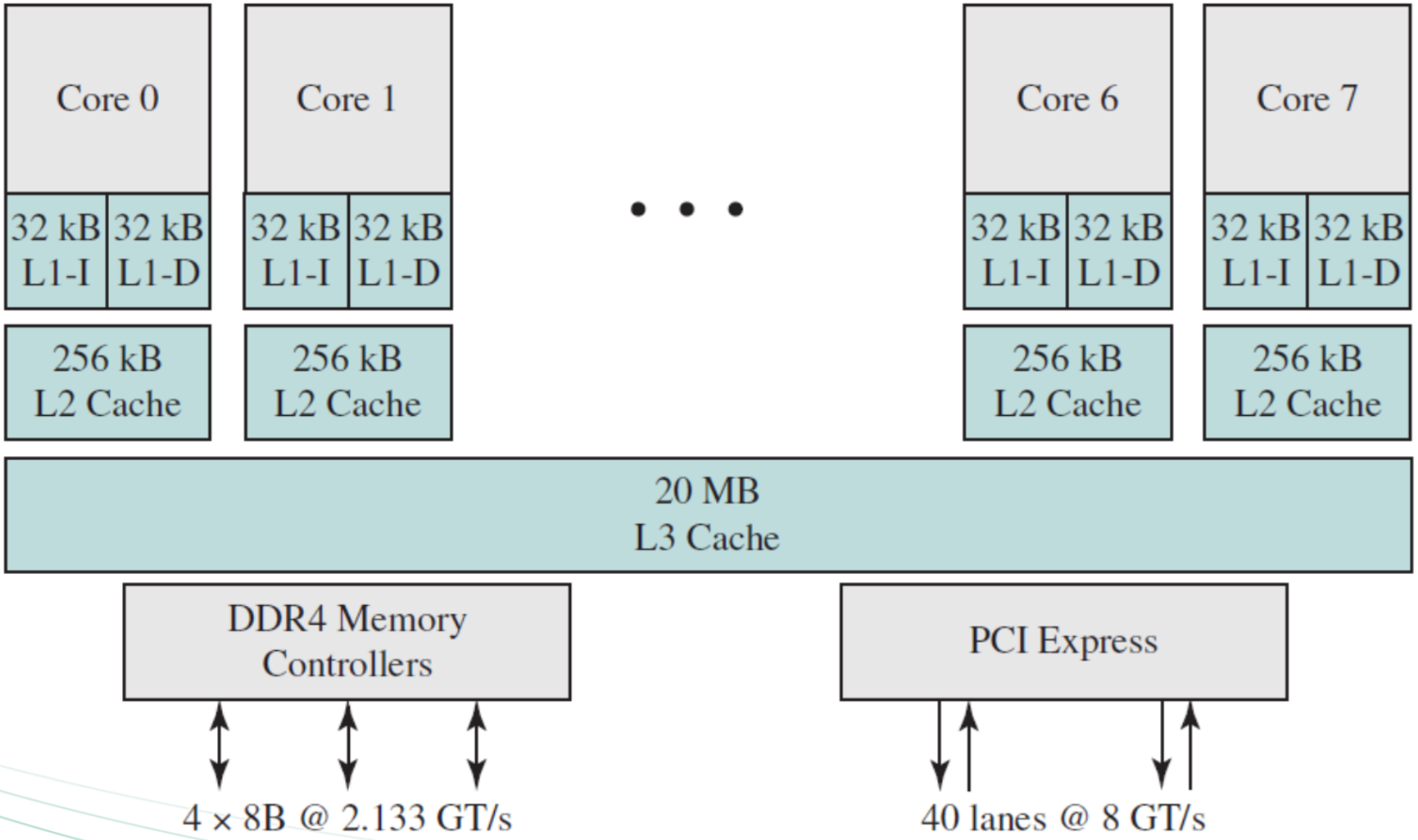
Performance	• a system with multiple processors will yield greater performance if work can be done in parallel
Availability	• the failure of a single processor does not halt the machine
Incremental Growth	• An additional processor can be added to enhance performance
Scaling	• Vendors can offer a range of products with different price and performance characteristics

Multicore Computer

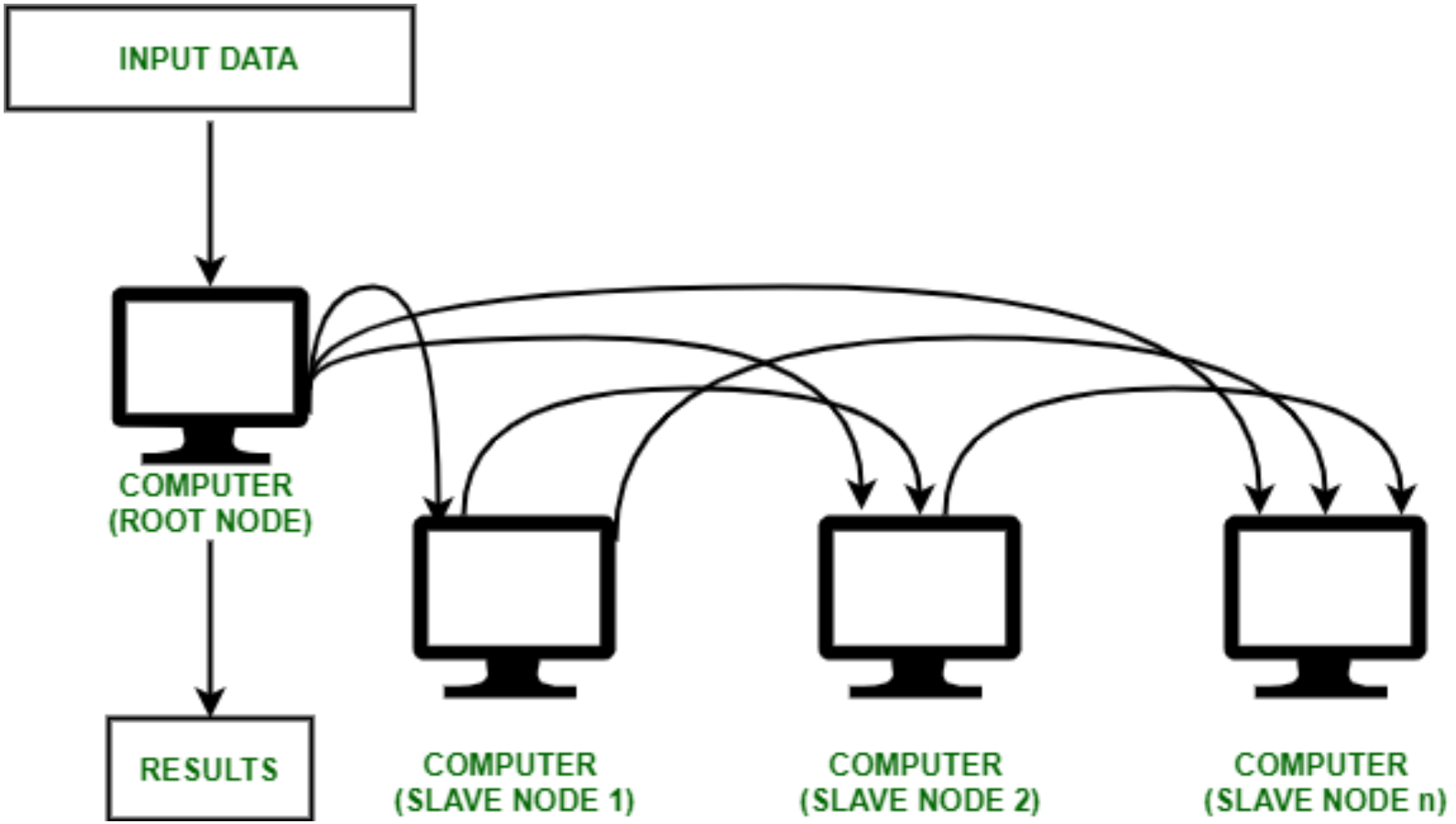
- Also known as a chip multiprocessor
- Combines two or more processors (**cores**) on a single piece of silicon (die)
  - Each core consists of all of the components of an independent processor
- Multicore chips also include L2 cache and in some cases L3 cache



Multicore Computer



Clusters


INPUT DATA
COMPUTER
(ROOT NODE)
RESULTS
COMPUTER
(SLAVE NODE 1)
COMPUTER
(SLAVE NODE 2)
COMPUTER
(SLAVE NODE n)