

CYB510 Operating Systems

Prepared Prof. Lamiaa Fattouh

Lecture 1



CYB510

Operating System Introduction



جامعة الجميع الذكية
EVERYONE'S SMART UNIVERSITY



□ The course aims to explore the importance of the operating system and its function. The different techniques used by the operating system to achieve its goals as resource manager. This course introduce also, the fundamentals of modern operating system function, design, and implementation. Topics include concurrency, synchronization, processes, threads, scheduling, memory management, file systems, device management, and security.



By the end of this course the student will be able to:

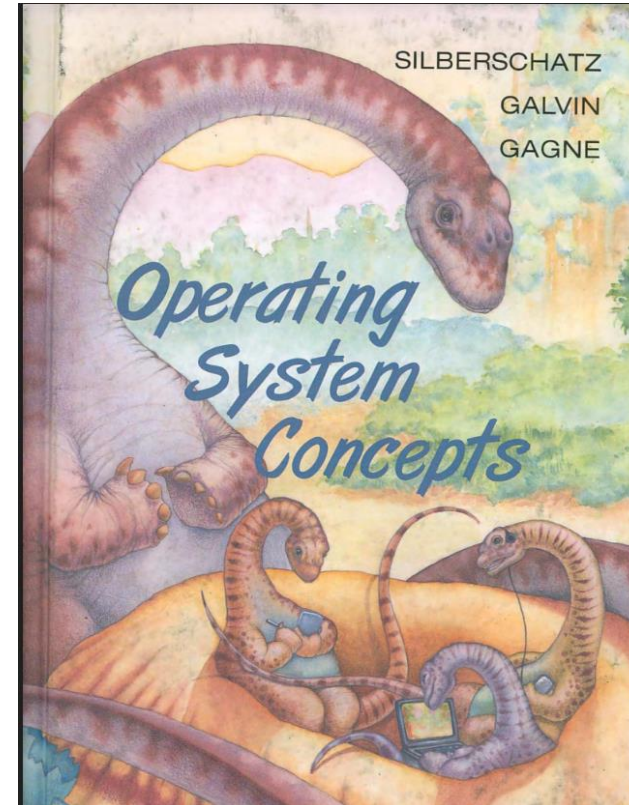
- Recognize the importance of the operating systems.
- Recognize how the applications interact with the operating system as the later working as intermediary program between the machine and the application.
- Know how the operating systems transport the application requests to the hardware.
- Understand how operating systems managing resources such as processors, memory and I/O.
- Realize the efficiency or the deficiency of the different techniques used by some operating systems.



- Module 1: Operating System Introduction.
- Module 2: Processes and threads.
- Module 3: Concurrency.
- Module 4: Memory management.
- Module 5: Scheduling and Deadlock.
- Module 6: File management.
- Module 7: Operating System security.
- Module 8: Virtualization.



Silberschatz, Galvin, Gagne. Operating System Concepts. John Wiley & sons , inc



Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Structure
- Operating-System Operations
- Process Management
- Memory Management
- Storage Management
- Protection and Security
- Distributed Systems
- Special-Purpose Systems
- Computing Environments
- Open-Source Operating Systems



- To provide a grand tour of the major operating systems components
- To provide coverage of basic computer system organization
- Explore core ideas in Operating Systems in two ways:
 - 1) understanding the concepts behind resource management, **abstraction**, hardware **interface**, **protection**, **security**, **concurrency**, and **performance**.
 - 2) practical coding experience with a real OS to understand challenges of systems



What is an Operating

- A **system software** program that acts as an **intermediary** between a user of a computer and the computer hardware. It Provides **abstractions** for resources (CPU, memory and I/O devices)
 - ✓ **User View:** A virtual machine on top of real machine to provide a more convenient user interface for the computer system
 - ✓ hide the details of individual devices and components
 - ✓ high level abstraction of the machine
 - ✓ **System View:** A **resource manager** to make efficient use of computer systems components
 - ▶ Shared resource: CPU, memory and I/O devices
 - ▶ Multiprogram batch systems: allow multiple batch jobs share the computer system
 - ▶ Time-sharing systems: allow multiple interactive jobs share the system
- Provide environment in which **applications software** can execute as if they use the hardware alone.



❑ Batch Systems

- Goal: Maximize amount work done for multiple users
- Applications run one after the other. One application at a time!
- Application uses all computer resources

❑ Batch Multiprogramming

- Multiple applications in memory
- When one waits for I/O, another executes
 - ▶ Better utilization of CPU



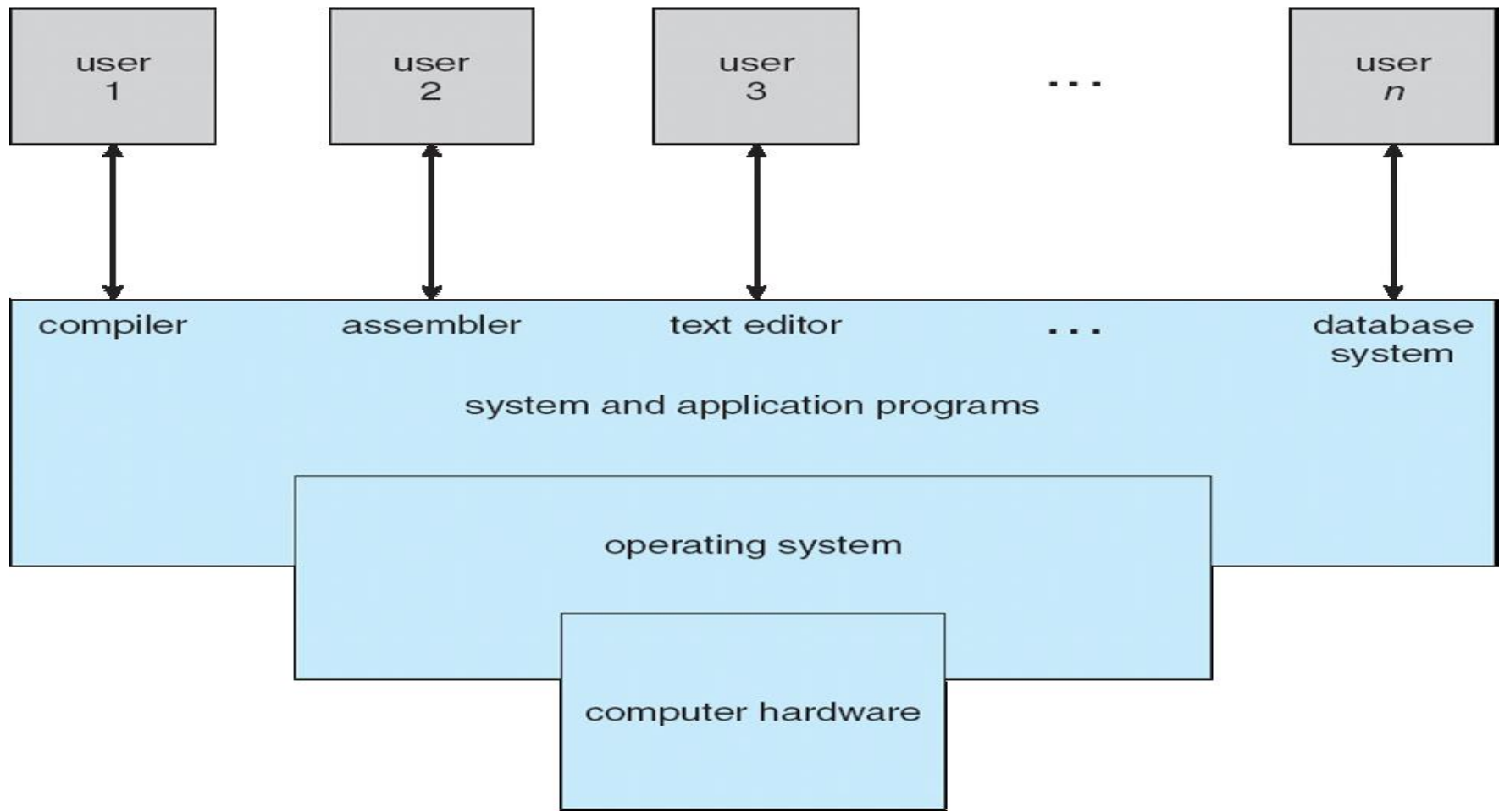
❑ Timesharing Systems

- Interactive use of computers
 - ▶ Responsiveness matters
 - ▶ Expect system to respond to keyboard input immediately
- Several users/applications can share computer
 - ▶ Share CPU, Disk, Memory...

❑ Real Time



Computer System Structure



Four Components



Computer System Structure

- ❑ Computer system can be divided into four components
 - Hardware – provides basic computing resources
 - ▶ CPU, memory, I/O devices
 - Operating system
 - ▶ Controls and coordinates use of hardware among various applications and users
 - System/Application programs – define 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



Operating System Definition

- ❑ OS is a **resource allocator**
 - Manages all resources (CPU, memory, disk space, peripherals)
 - 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
- ❑ No universally accepted definition



Operating System Definition

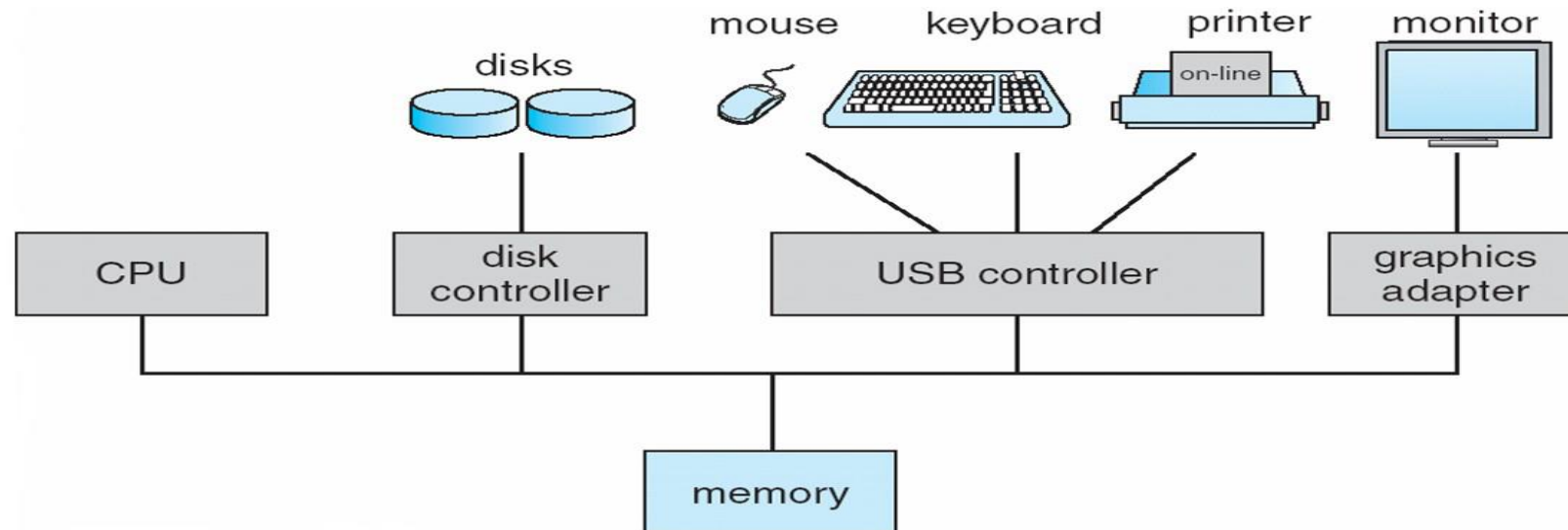
- ❑ “Everything a vendor ships when you order an operating system” is good approximation
 - But varies wildly
- ❑ “The one program running at all times on the computer” is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program. **kernel**
 - provides resources to them
 - Controls their execution



- How can an Operating system being loaded in Memory????
- Answer: we need a program that loads OS from Disk into memory. This program is the bootstraper
- **bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM, generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

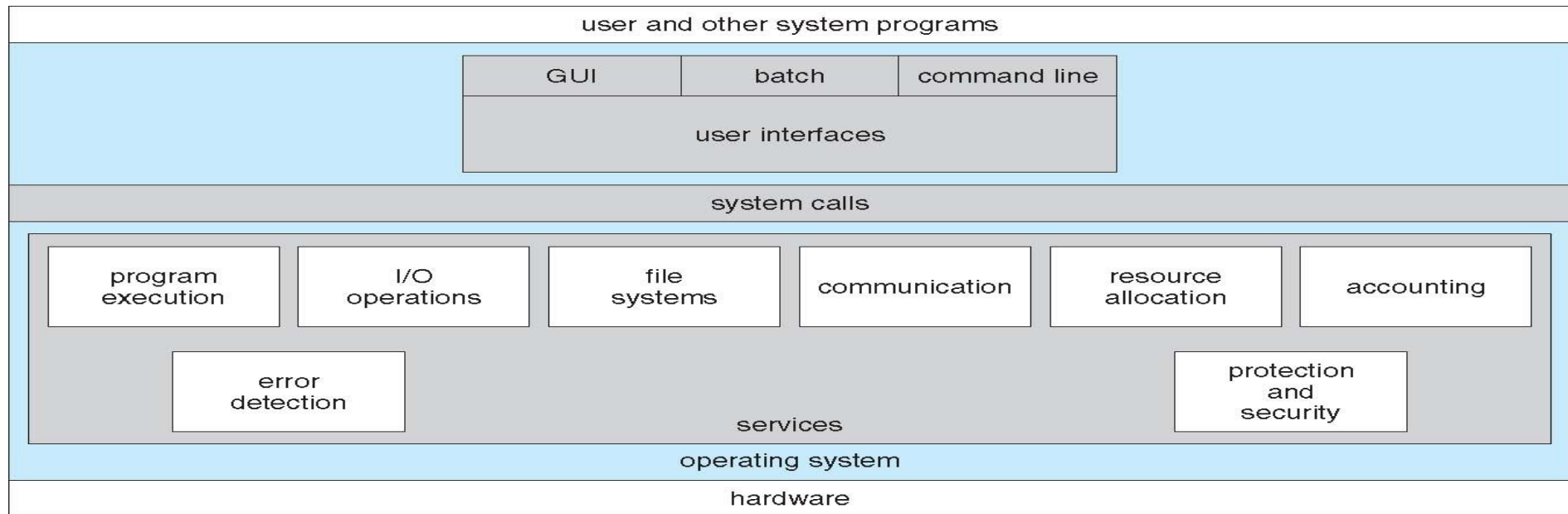


- Computer-system operation
 - One or more CPUs, device controllers connect through common bus providing access to shared memory
 - Concurrent execution of CPUs and devices competing for memory cycles



Chapter 01

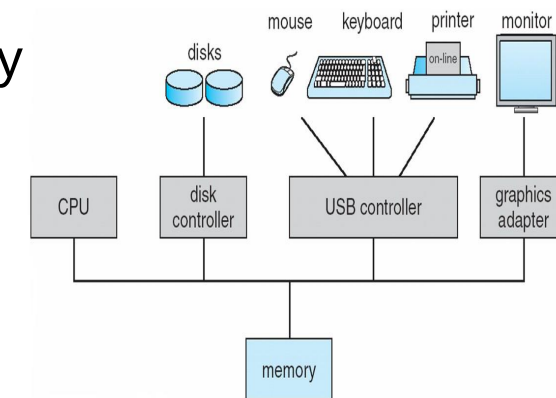
HW Architecture and OS Services



OS Services: I/O Management

- ❖ Each device controller is in charge of a particular device type
- ❖ Each device controller has a local buffer/control regs
- ❖ CPU moves data from/to main memory to/from local buffers
- ❖ Protocol: To get a byte of data from a device
 - CPU sets registers in controller with command to read e.g. character from keyboard
 - 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**
 - CPU moves data from controller to memory

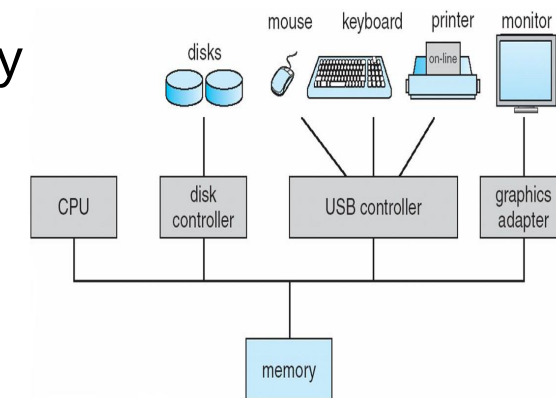
I/O devices and the CPU can execute concurrently



OS Services: I/O Management

- ❖ Each device controller is in charge of a particular device type
- ❖ Each device controller has a local buffer/control regs
- ❖ CPU moves data from/to main memory to/from local buffers
- ❖ Protocol: To get a byte of data from a device
 - CPU sets registers in controller with command to read e.g. character from keyboard
 - 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**
 - CPU moves data from controller to memory

I/O devices and the CPU can execute concurrently



❑ I/O Techniques

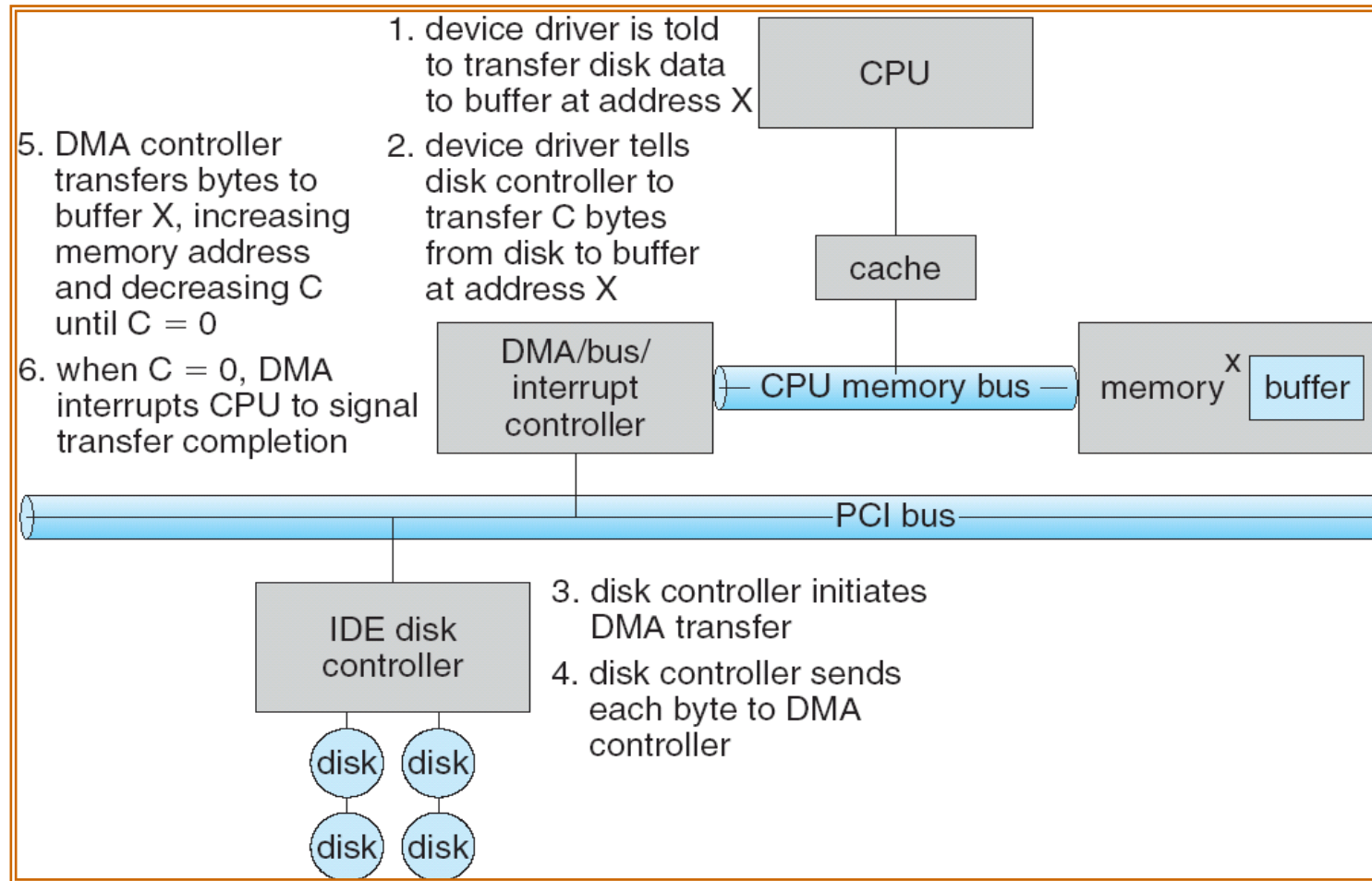
- Programmed I/O: processor interacts with I/O module **directly**
 - issues I/O commands to I/O module
 - interrogates status bits repeatedly (**Pooling**)
 - reads or writes: transfers the data between CPU and I/O module
- Interrupt-Driven I/O: I/O module **interrupts** when I/O module is ready
 - issues I/O commands to I/O module
 - goes off to run other processes (doing something else)
 - get interrupted when I/O module is ready and reads or writes in interrupt handlers

❑ Direct Memory Access (DMA): A separate module called **DMA** module takes care of the interrupt from I/O and data transfer

- Processor sends request to DMA (starting address, number of bytes, address of I/O device, read or write)
- DMA interacts with I/O module and completes the bulk transfer
- DMA interrupts processor when done



Six Step Process to Perform DMA Transfer



Direct Memory Access Structure

- Used for high-speed I/O (e.g. networking card 1G/sec) 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 one interrupt is generated per block, rather than the one interrupt per byte. Block size varies from device to device.



❑ Interrupt

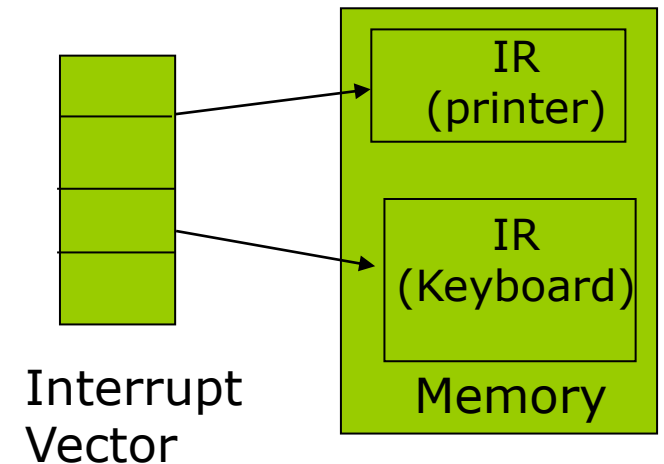
- 1. an **external event** raised by hardware: e.g. I/O
- 2. an **internal event** trigger by software: e.g. **system calls** (means application needs a service from the OS. It calls a service). **TRAP**

❑ Interrupt Handler

- a subroutine in the operating system to process a particular type of interrupt

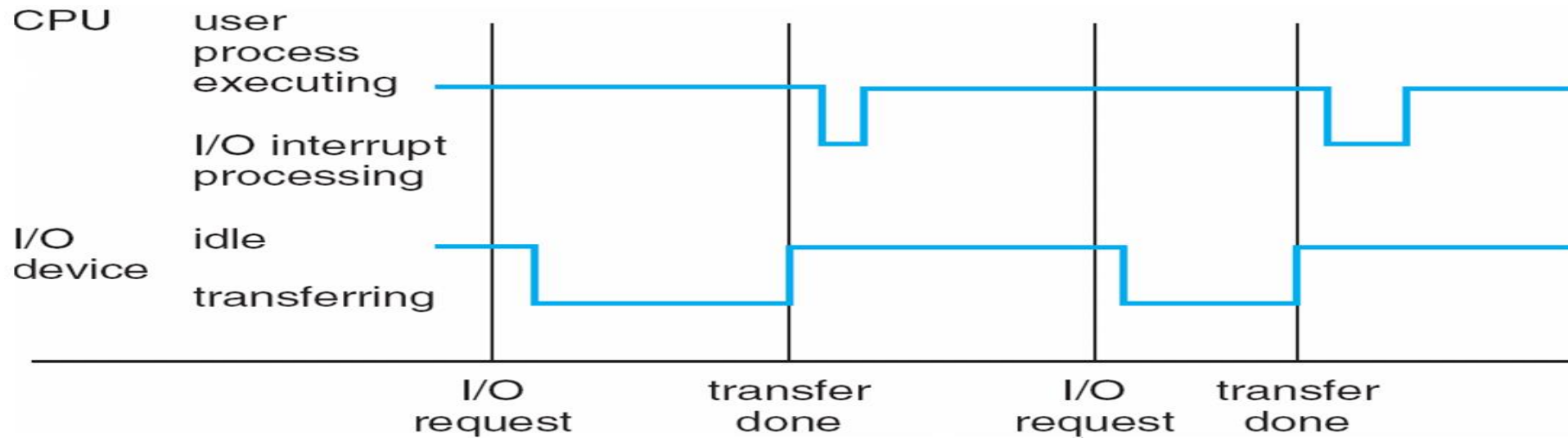
❑ Interrupt Vector

- an array of starting addresses of interrupt handlers
- The index for a particular interrupt to the array is provided by the hardware.



- ❑ Interrupt transfers control to the interrupt service routine (ISR) generally, through the **interrupt vector**, which contains the addresses of all the service routines. ISR is identified by address in interrupt vector
- ❑ Interruption handling (overhead)
 - Interrupt architecture must save the address of the interrupted instruction
 - After servicing interrupt, CPU resumes execution at previously interrupted address (or “flow of control”)





Interrupt Handling

- Separate segments of code determine what action should be taken for each type of interrupt
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*
- An operating system is **interrupt driven**



- Operating systems have a device driver for each I/O controller.
 - It understands the device controller (hide the peculiarities of the device)
 - presents a uniform interface of the device to the rest of the operating system.



- **To get a service from a device driver application make**
 - **System call** – request to the operating system to allow user to wait for I/O completion
 - ▶ **Device-status table** contains entry for each I/O device indicating its type, address, and state

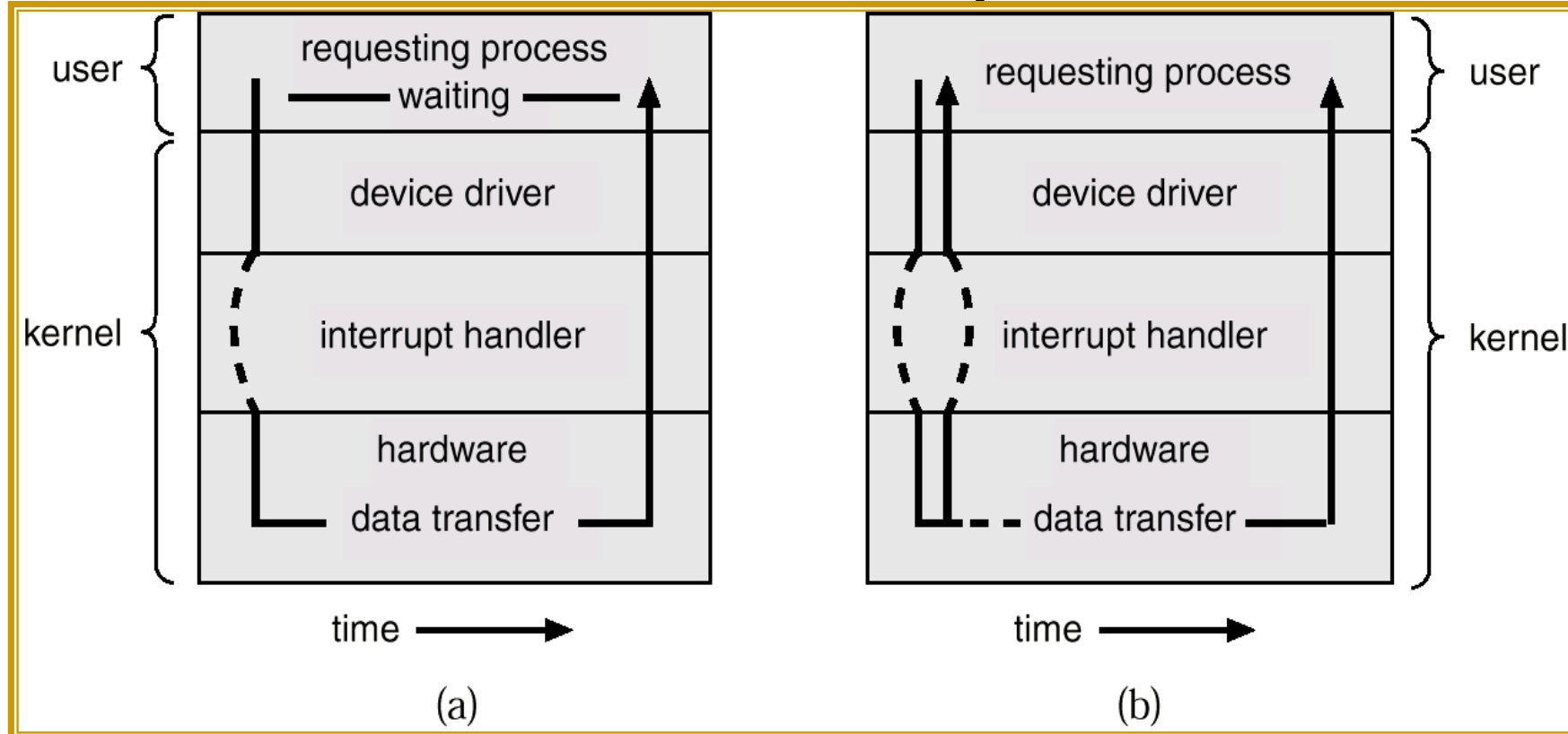


I/O Structure- Sync and Async I/O

- **Synchronous** : After I/O starts, control returns to user program only upon I/O completion
- **Asynchronous**: After I/O starts, control returns to user program without waiting for I/O completion

Synchronous

Asynchronous



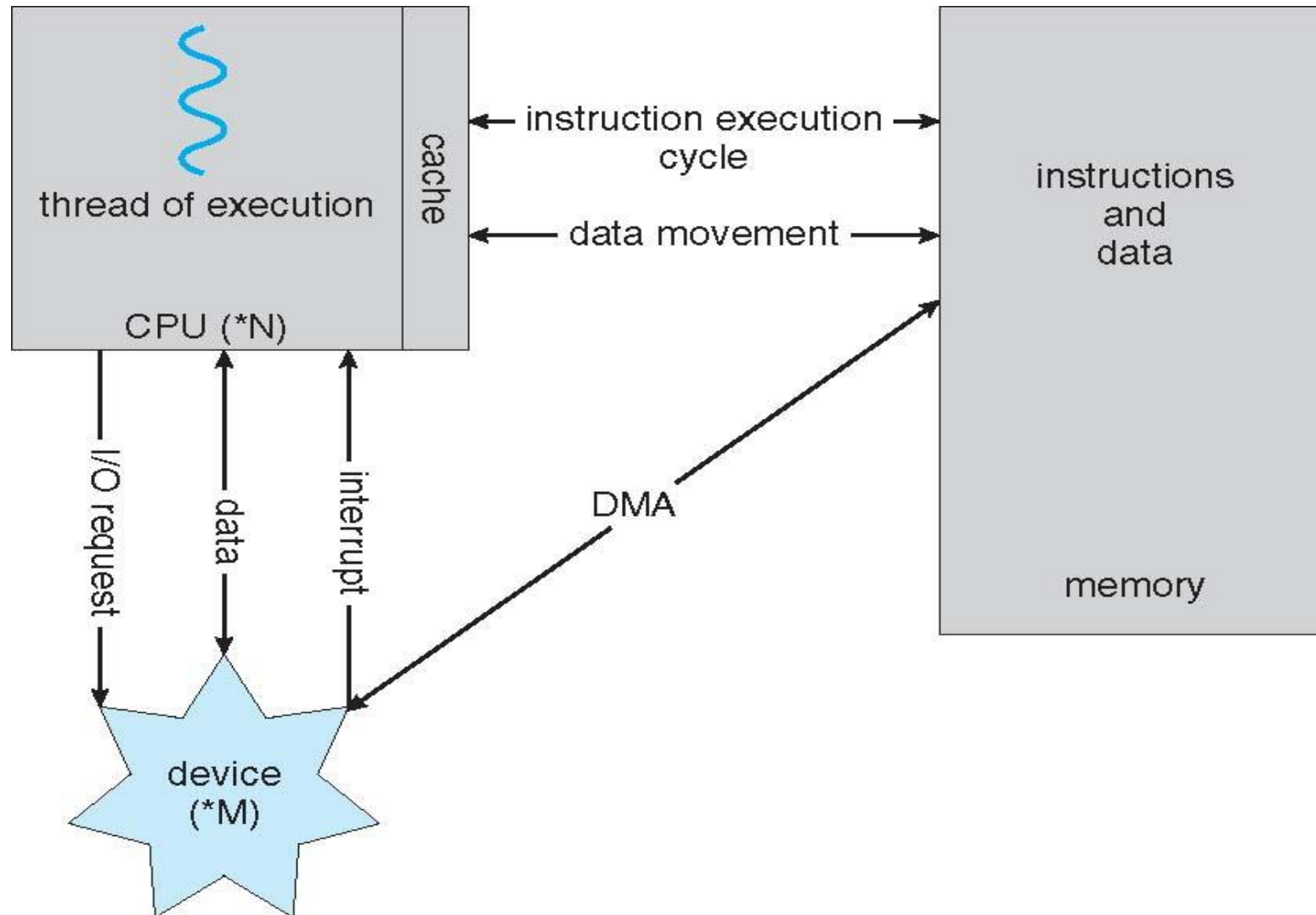
How a Modern Computer Works

- CPU : unit where instructions are Executed
- Memory: storage where both program and data are stored
- I/O subsystem: unit to input and output data
- Instruction Execution Cycle
do forever
 1. fetch next instruction
 2. execute the instruction
 3. check and process interrupt



Chapter 01

How a Modern Computer Work



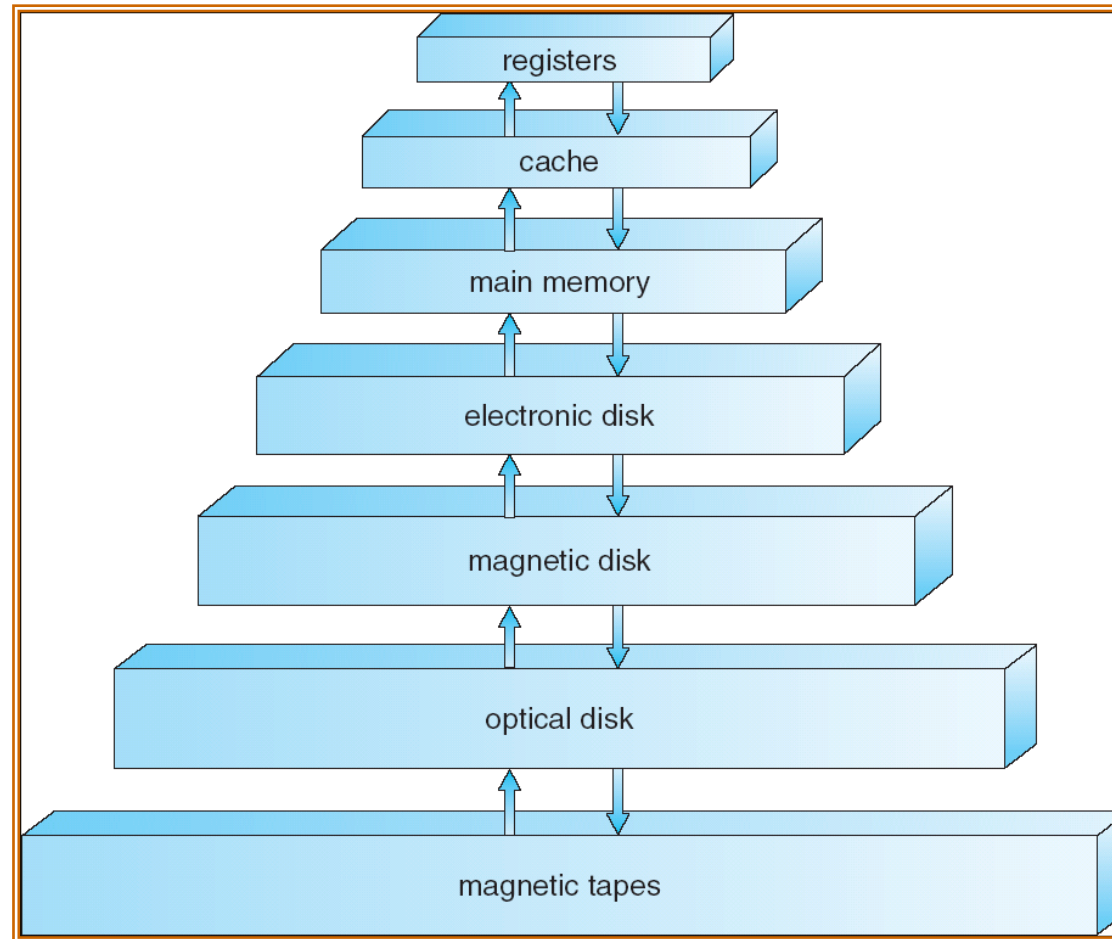
- n Main memory – only large storage media that the CPU can access directly.
- n Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- n Magnetic disks – rigid metal or glass platters covered with magnetic recording material
 - | Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
 - | The *disk controller* determines the logical interaction between the device and the computer.



- n Storage systems organized in hierarchy.
 - | Speed
 - | Cost
 - | Volatility
- n *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.



Storage-Device Hierarchy



- n Important principle, performed at many levels in a computer (in hardware, operating system, software)
- n Information in use copied from slower to faster storage temporarily
- n Faster storage (cache) checked first to determine if information is there
 - | If it is, information used directly from the cache (fast)
 - | If not, data copied to cache and used there
- n Cache smaller than storage being cached
 - | Cache management important design problem
 - | Cache size and replacement policy

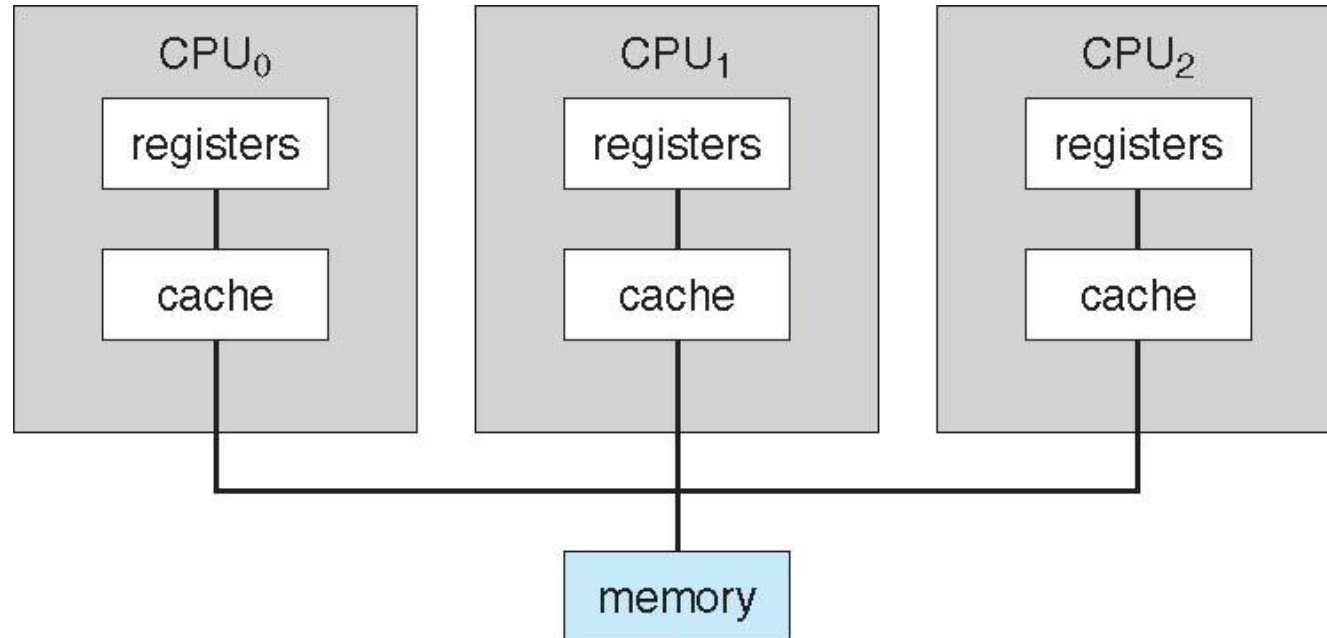


- ▶ Most systems use a single general /special-purpose processor
- ▶ **Multiprocessors** systems growing in use & importance
 - | Advantages
 1. Increased throughput
 2. Economy of scale
 3. Increased reliability
 - | Two types
 1. Asymmetric Multiprocessing
 2. Symmetric Multiprocessing

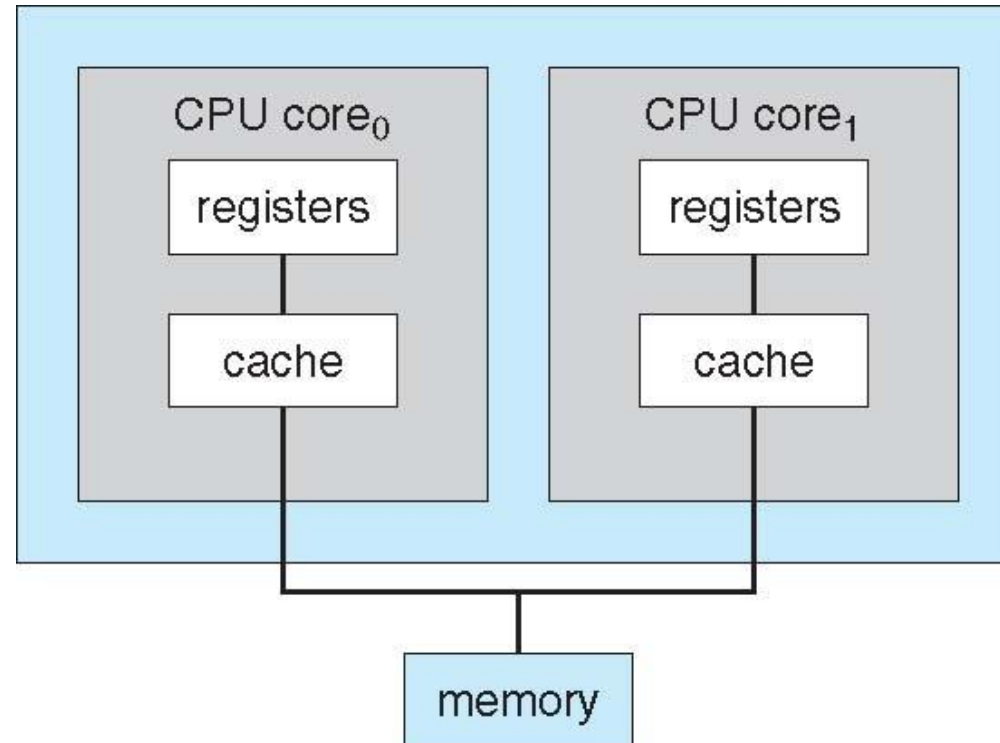


Chapter 01

Symmetric Multiprocessing Architecture



A Dual-Core Design



END LECTURE 1
THANK YOU

