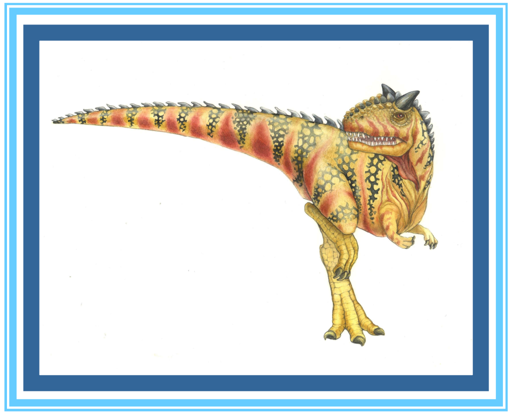
UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE
ELECTRONICA

# Teoría de Sistemas Operativos

## *ELO 321 – Clase 01*

Dr. Ioannis Vourkas
*Oficina B320, Email:* `ioannis.vourkas@usm.cl`

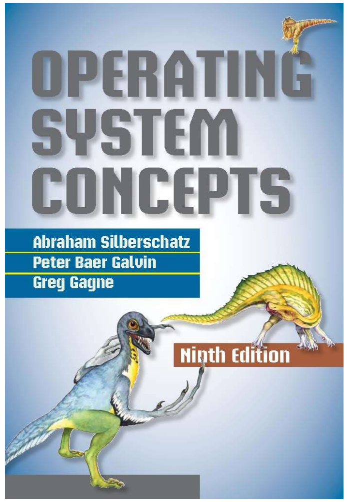


08 de septiembre, 2020, Valparaíso, Chile

# Course Information

- **Evaluation**
  - **2** Certamenes (75%) y **2** tareas (homework) (25%)…

- **Course book**: A. Silberschatz, P. Galvin, y G. Gagne (2012). *Operating System Concepts*, Wiley, 9th Ed. (http://www.os-book.com)

# Course Objectives

- Explain what operating systems are, what they do, and how they are designed and constructed.
- Contents are organized as follows:
  - Process management. Describe the process concept and concurrency as the heart of modern operating systems. A process is the unit of work in a system. Cover methods for *process scheduling, inter-process communication, process synchronization*, etc.
  - Memory management. Management of main memory during the execution of a process. There are many different *memory-management schemes*, reflecting various approaches to memory management
  - Storage management. Describe how the *file system and I/O* are handled in a modern computer system. We describe the classic internal algorithms and structures of storage management and provide a firm practical understanding of the *algorithms* used—their *properties*, *advantages*, and *disadvantages*.
  - Protection and security...

# 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
- Kernel Data Structures

## Objectives

- To describe the *basic organization of computer systems*
- To provide a grand tour of the *major components* of operating systems

# What is an Operating System?

- An operating system is a program that manages a computer's hardware.
  - It provides a basis for application programs
  - It acts as an ***intermediary*** between the user and the hardware.

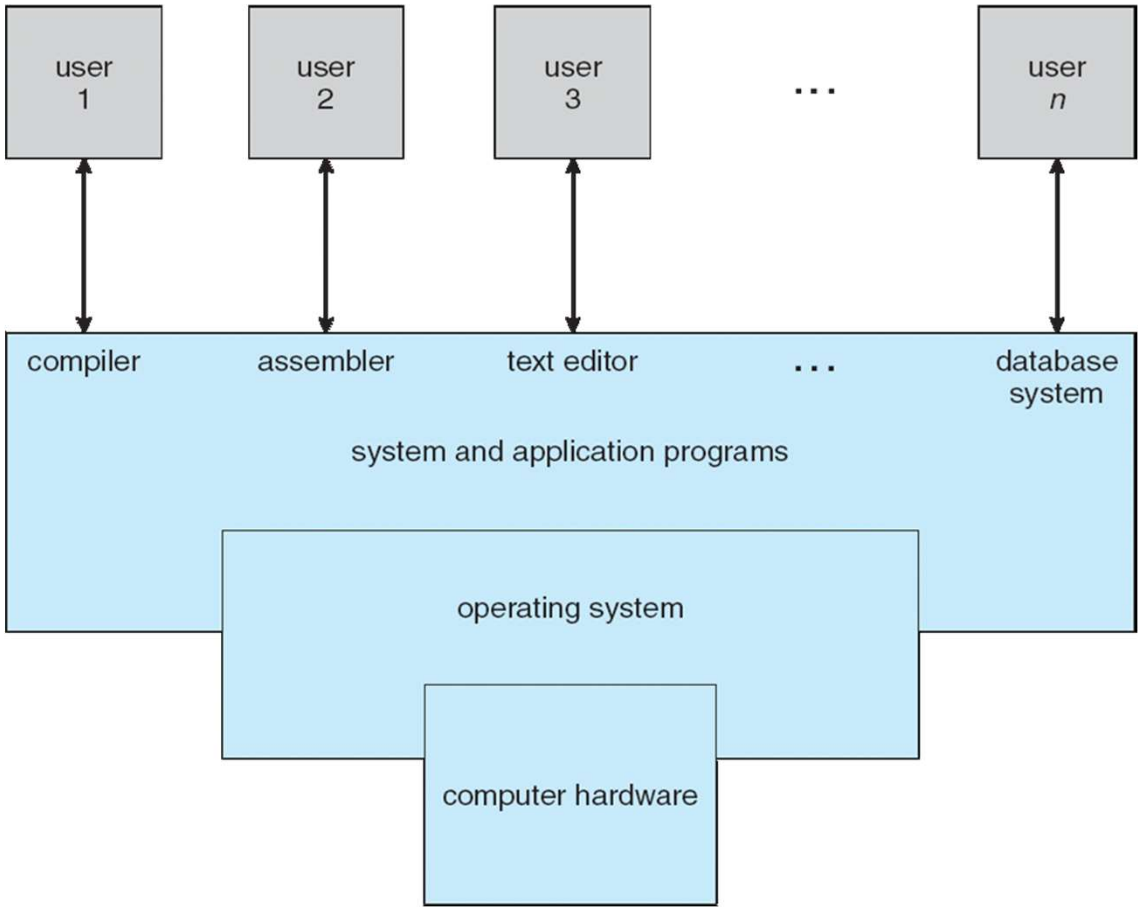
- Some operating systems are designed to be convenient, others to be efficient, and others to be some combination of the two.

- Next we provide a general overview of the major components of a contemporary computer system as well as the functions provided by the operating system.

# 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
  - 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,…
  - Users
    - People, machines, other computers

# Four Components of a Computer System



# What Operating Systems Do

## User View

- In system designed for one user, the operating system is designed mostly for **ease of use**

## System View

- The operating system is a **resource allocator**.
  - CPU time, memory space, file-storage space, I/O devices, etc
  - Facing numerous (possibly conflicting), it must decide how to allocate resources to specific programs and users to operate the computer system efficiently and fairly.

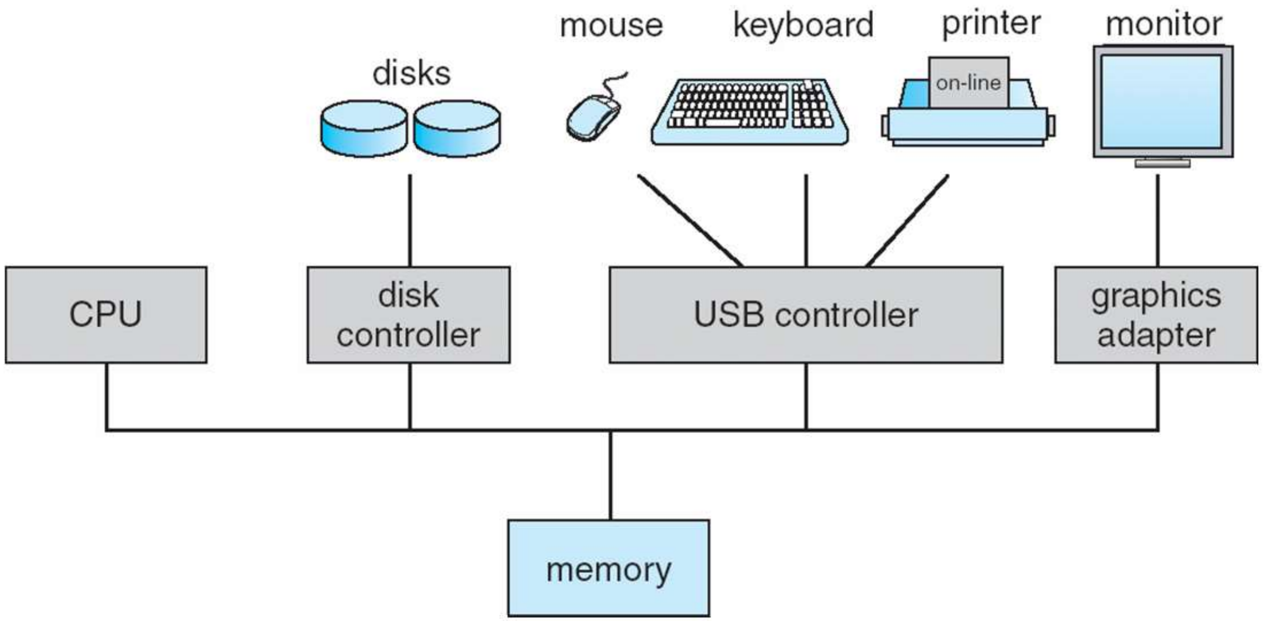
- A slightly different view of an operating system emphasizes the need to control the various I/O devices and user programs.
  - It manages the execution of user programs to prevent errors and improper use of the computer.

# Operating System Definition

- The term operating system covers many roles and functions. → How, then, can we define what an operating system is?
  - In general, we have no completely adequate definition of an operating system.
  - Operating systems exist because they offer a reasonable way to create a usable computing system.
- Since bare hardware alone is not particularly easy to use, application programs are developed.
  - These programs require certain common operations, such as those controlling the I/O devices.
  - The **common functions of controlling and allocating resources are then brought together into one piece of software: the operating system**.
- A more common definition is that the operating system is the program running at all times on the computer—usually called the **kernel**.

# Computer System Organization

- **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

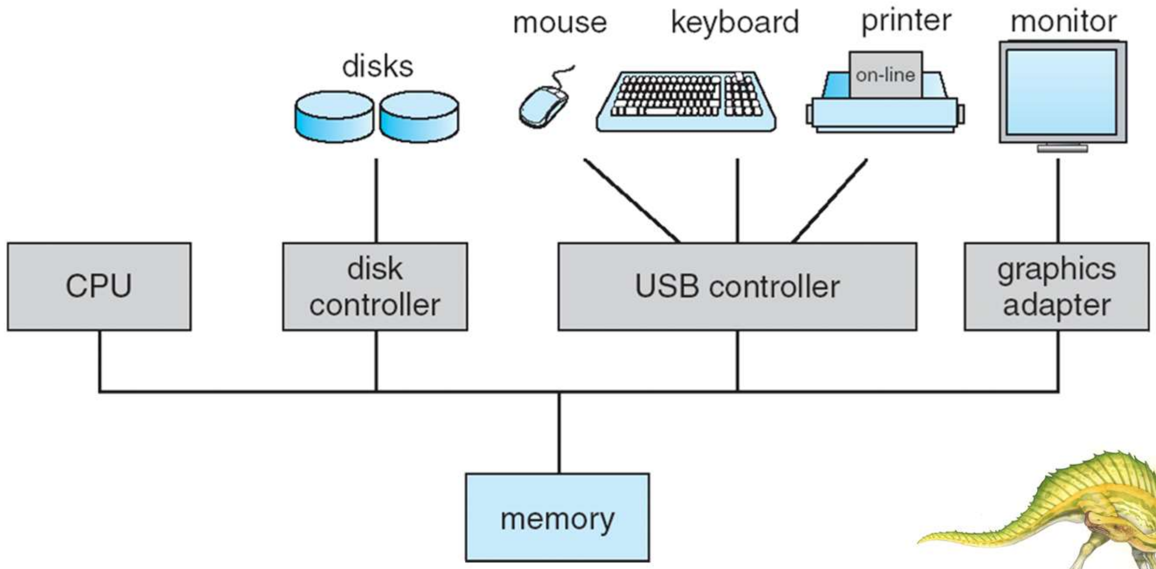


# Computer System Organization

- For a computer to start running—when it is powered up or rebooted—it needs to have an initial program to run.
  - This initial program, or **bootstrap program**, is stored within the computer hardware in read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**)
  - It initializes CPU registers, device controllers and memory contents.
    - The bootstrap program must locate the operating-system kernel and load it into memory.
- System programs that are loaded into memory at boot time to become system processes.

# Computer-System Operation

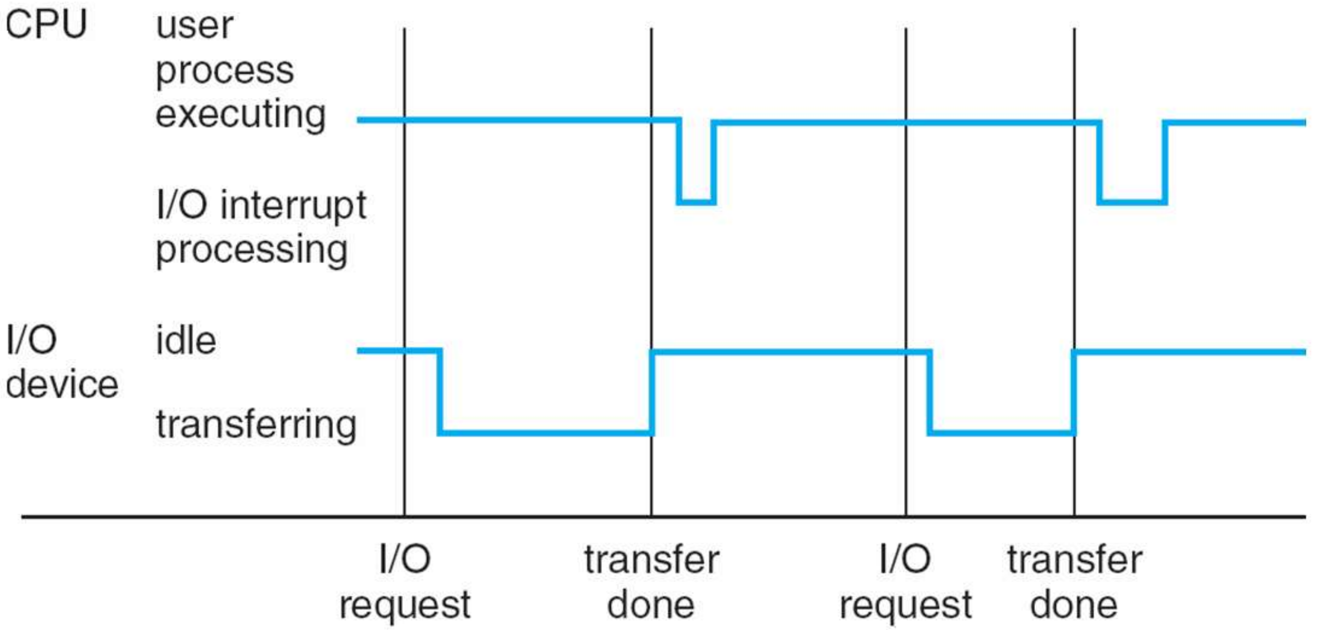
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs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



# Common Functions of Interrupts

- An operating system is **interrupt driven**
- The occurrence of an event is usually signaled by an interrupt from either the hardware or the software.
  - Software may trigger an interrupt (**trap** or **exception**) by executing a special operation called a system call
- **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 is located.
  - The interrupt service routine executes; on completion, the CPU resumes the interrupted computation.
- Interrupts **must be handled quickly**.
  - Since only a predefined number of interrupts is possible, **a table of pointers to interrupt service routines** can be used
  - the table of pointers is stored in low memory (the first hundred or so locations), which is called **interrupt vector**.

# Interrupt Timeline



# Storage Structure

- The **CPU can load instructions only from main memory** (also called random-access memory, or **RAM**), so any programs to run must be stored there.
  - Computers use other forms of memory as well (cache memory, CD-ROM, etc).

- All forms of memory provide an array of bytes. **Each byte has its own address**.

- The load instruction moves a byte or word from main memory to a register within the CPU, whereas the store instruction moves the content of a register to main memory.
- **Ideally**, we want the programs and data to reside in main memory permanently. This arrangement usually is not possible:
  1. Main memory is **usually too small** to store all needed programs and data permanently.
  2. Main memory is a **volatile** storage device that loses its contents when power is turned off

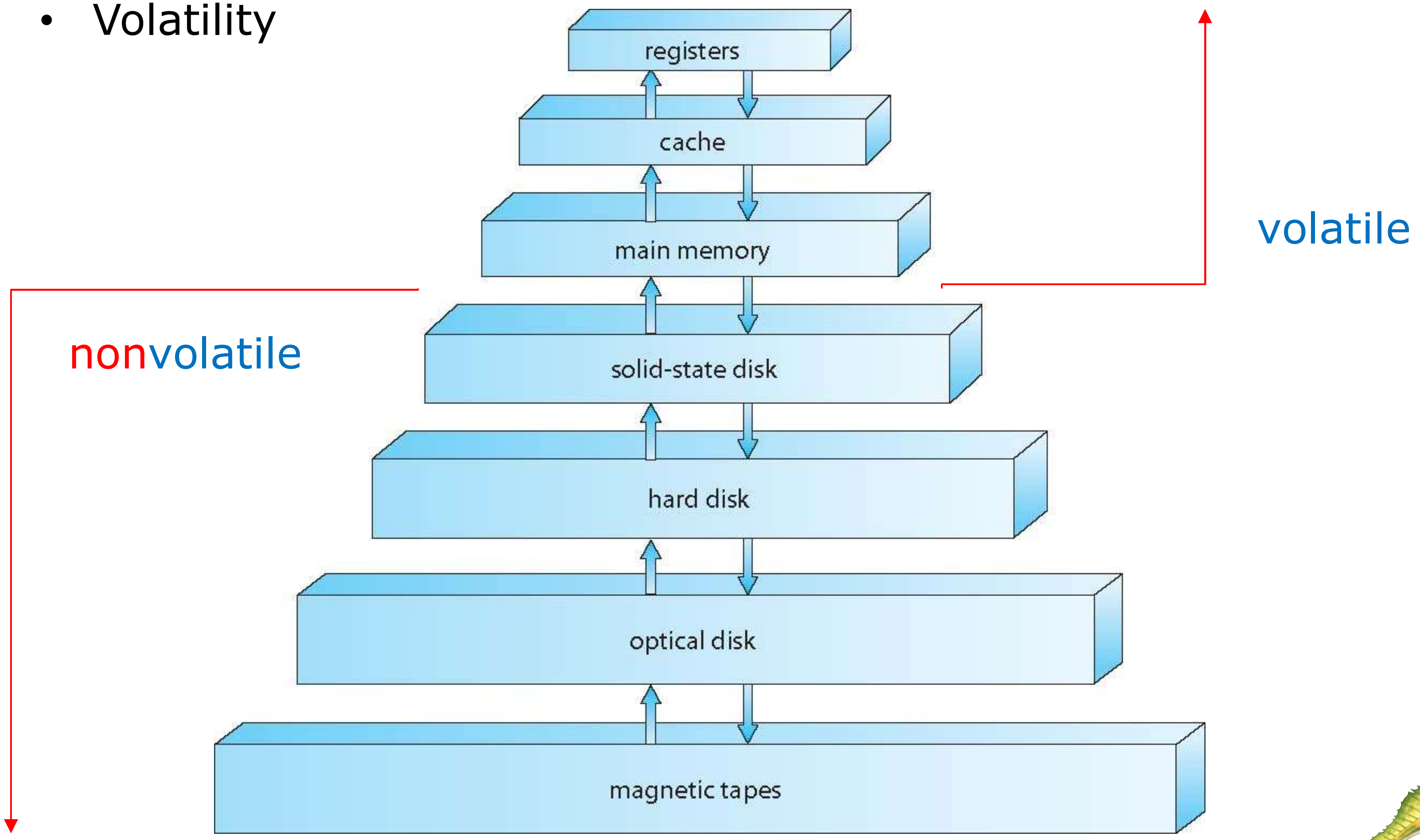
# Storage Structure

- Thus, most computer systems provide **secondary storage** as an extension of main memory.
- The main requirement for secondary storage is that it be able to *hold large quantities of data permanently*.
- The most common secondary-storage device is a **magnetic disk**
- The main differences among the various storage systems lie in *speed, cost, size, and volatility*.
- The wide variety of storage systems can be organized in a **hierarchy** according to *speed* and *cost*. The higher levels are expensive, but they are fast.

# Storage-Device Hierarchy

Storage systems organized in hierarchy

- Speed
- Cost
- Volatility



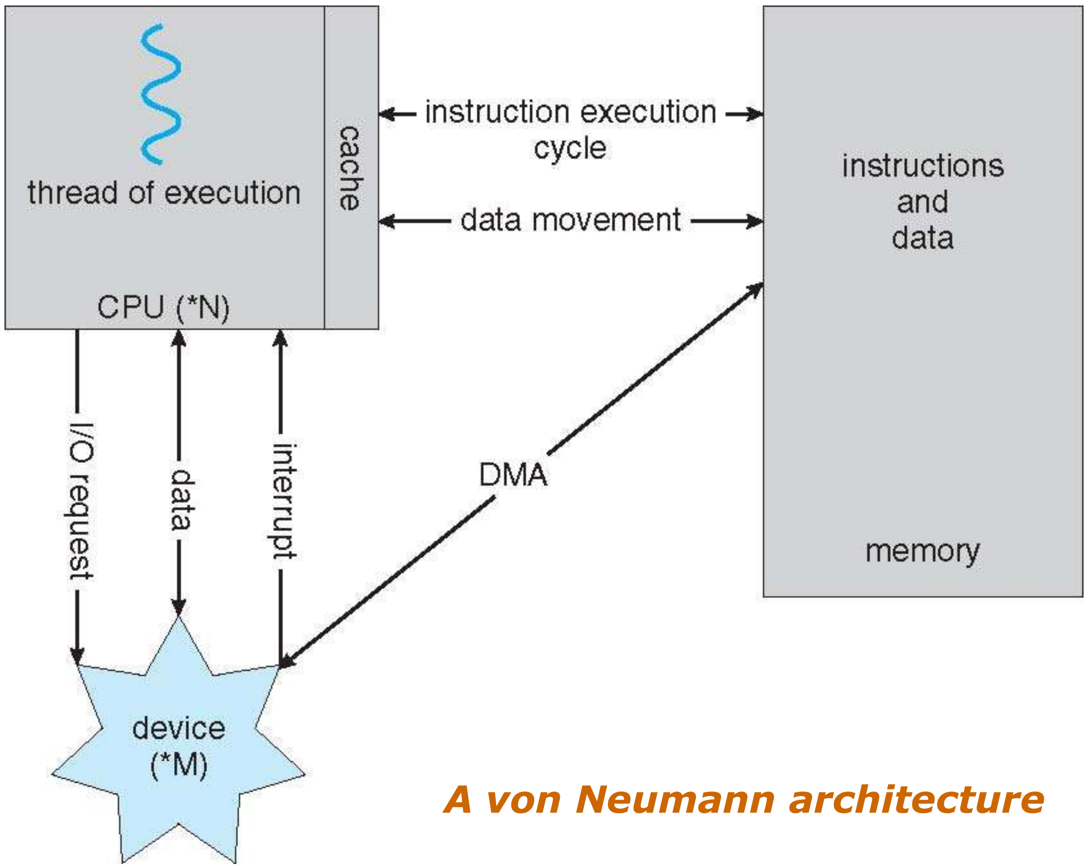
# I/O Structure

- A large portion of operating system code is **dedicated to managing I/O**, because of its importance to the reliability and performance of a system.
- A general-purpose computer system consists of CPUs and **multiple device controllers** that are connected through a common bus.
  - Each device controller is in charge of a specific type of device.
  - It is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.
- Typically, operating systems have a **device driver** for each device controller.
  - This device driver provides the operating system with a **uniform interface to the device**.

# I/O Structure

- To **start an I/O operation**, the device driver loads the appropriate registers within the device controller.
- The device controller examines the contents of these registers to **determine what action to take**
  - The controller starts the transfer of data from the device to its local buffer.
  - Once the transfer of data is complete, the device controller informs the device driver via an **interrupt**
- This form of interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement.
- To solve this problem, **direct memory access (DMA)** is used.
  - the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.
  - While the device controller is performing these operations, the CPU is available to accomplish other work.

# How a Modern Computer Works




*A von Neumann architecture*

# Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1{,}024^2$ bytes
a **gigabyte**, or **GB**, is $1{,}024^3$ bytes
a **terabyte**, or **TB**, is $1{,}024^4$ bytes
a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

# Computer-System Architecture

- **Single-Processor Systems**
  - One main CPU capable of executing a general-purpose instruction set

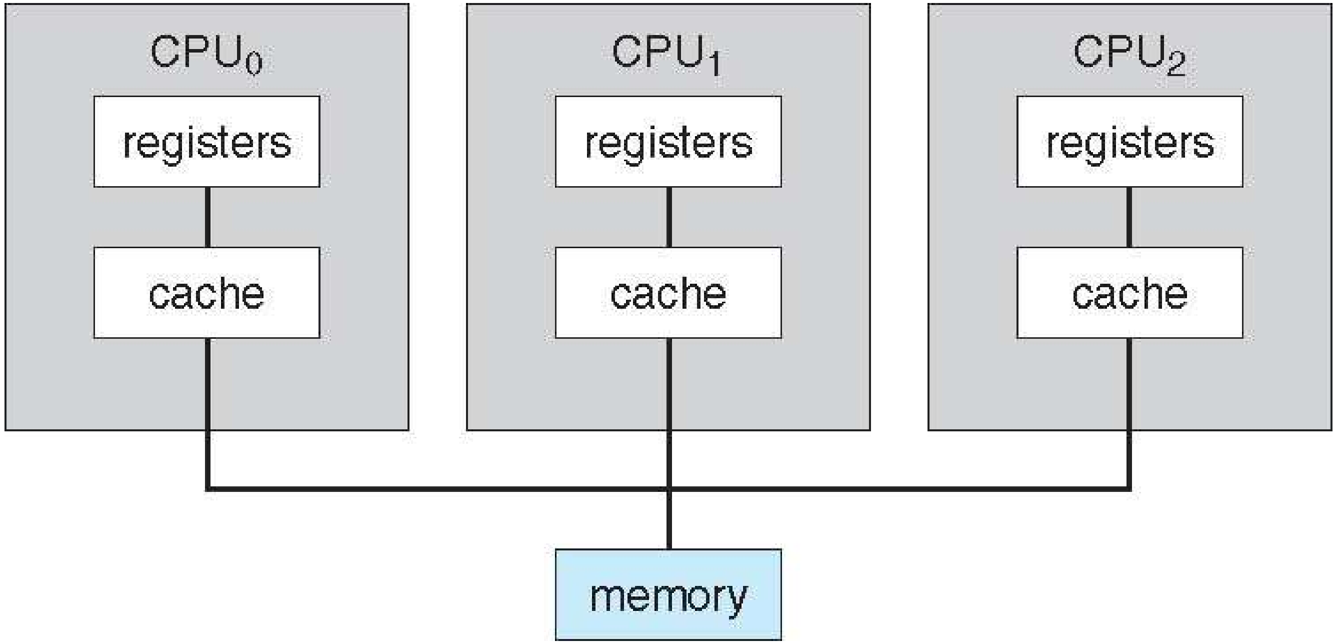
- **Multi-processor (multicore) Systems**
  - Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.

- Multiprocessor systems have three **main advantages**:
  1. **Increased throughput**. More work done in less time. The speed-up ratio with N processors is not N, however; rather, it is less than N.
  2. **Economy of scale**. Multiprocessor systems can cost less than equivalent multiple single-processor systems
  3. **Increased reliability**. Failure of one processor will not halt the system, only slow it down.

# Computer-System Architecture

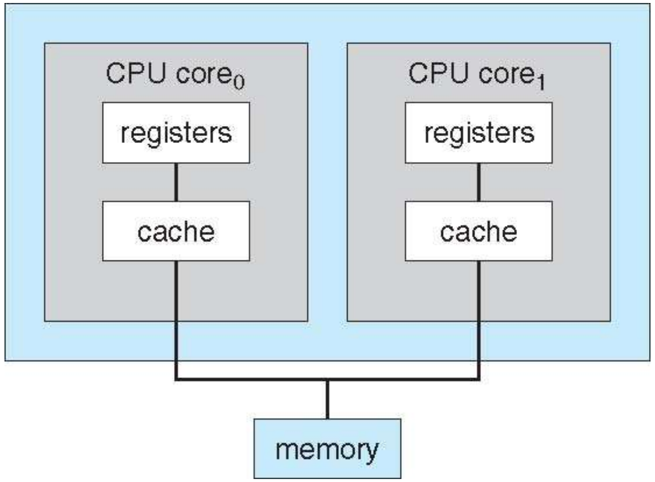
- The multiple-processor systems are of two types:
  1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
     - *This scheme defines a boss–worker relationship. The boss processor schedules and allocates work to the worker processors.*
  2. **Symmetric Multiprocessing** – each processor performs all tasks
     - *All processors are peers; Each processor has its own set of registers, as well as a private—or local —cache. However, all processors share physical memory*
     - *The benefit of this model is that many processes can run simultaneously—N processes can run if there are N CPUs—*

# Symmetric Multiprocessing Architecture



# A Dual-Core Design

- Multi-chip and **multicore**
- *A recent trend* in CPU design is to include multiple computing cores on a single chip.
- Such multiprocessor systems are termed multicore. They can be *more efficient* than multiple chips with single cores because on-chip **communication** is faster than between-chip communication.
- In addition, one chip with multiple cores uses *significantly less power* than multiple single-core chips.

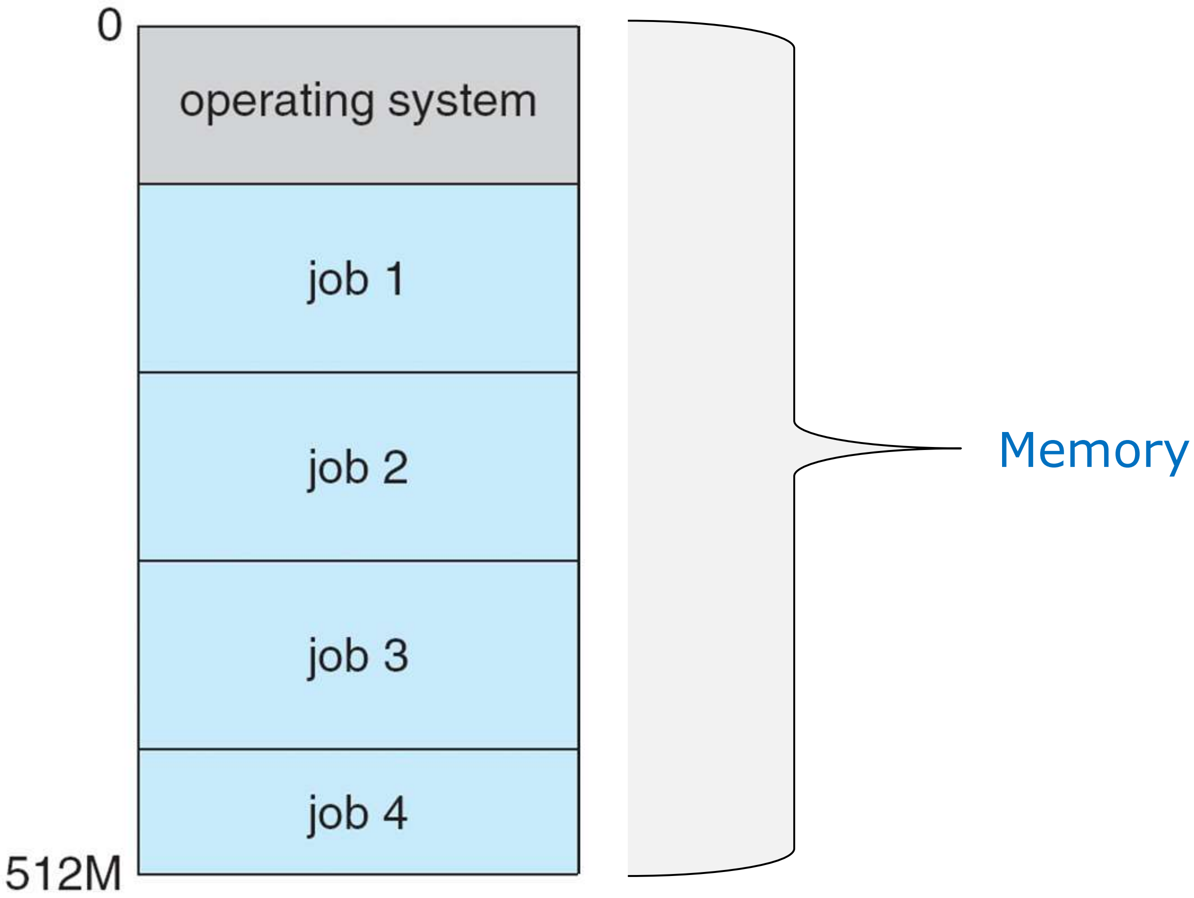



- In this design, each core has its *own register set as well as its own local cache*. Other designs might use a shared cache or a combination of local and shared caches.
- These multicore CPUs appear to the operating system as N standard processors.

# Operating System Structure

- One of the most important aspects of operating systems is the ability to **multi-program**.
  - Multiprogramming **increases CPU utilization** by organizing jobs (code and data) so that the **CPU always has one to execute**.
- The idea is as follows:
  - The operating system keeps several jobs in memory simultaneously
  - Since main *memory is too small to accommodate all jobs*, the jobs are kept initially on the disk in the **job pool**.
  - This pool consists of all processes residing on disk *awaiting allocation of main memory*.
  - The operating system picks and begins to execute one of the jobs in memory.
  - Eventually, the job may have to wait for some task, such as an I/O operation, to complete. In a multi-programmed system, the operating system *simply switches to, and executes, another job*.
- As long as at least one job needs to execute, **the CPU is never idle**.

# Memory Layout for Multiprogrammed System



# Operating System Structure

- **Time sharing** (or multitasking) is a logical extension of multiprogramming.
  - The CPU **executes multiple jobs by switching among them**, but the switches occur so frequently that the *users can interact with each program* while it is running.
  - A time-shared operating system allows many users to share the computer simultaneously.
    - It provides each user with a small portion of a time-shared computer → *each user is given the impression that the entire computer system is dedicated to his use*
  - **A program loaded into memory and executing is called a process**. When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O.
  - Time sharing and multiprogramming require that several jobs be kept simultaneously in memory.
    - If several jobs are ready to be brought into memory, and if there is **not enough room** for all of them, then the system must choose among them. This involves **job scheduling**!

# Operating System Structure

- Having several programs in memory at the same time requires some form of **memory management**.
- If several jobs are ready to run at the same time, the system must choose which job will run first. Making this decision is **CPU scheduling**.

- **Virtual memory**
  - A technique that allows the execution of *a process that is not completely in memory*.
  - It enables users to *run programs that are larger than actual physical memory*.
  - Further, it abstracts main memory into a large, uniform array of storage

- A time-sharing system must also provide a **file system**. The file system resides on a collection of disks; hence, **disk management** must be provided
- To ensure orderly execution, the system must provide mechanisms for job **synchronization** and **communication**, and it may ensure that jobs do not get stuck in a **deadlock**, forever waiting for one another

# Operating-System Operations

- As mentioned earlier, modern operating systems are **interrupt** driven (both from hardware and software).
  - If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen.

- **Dual-Mode** and Multi-mode Operation
  - In order to ensure the proper execution of the operating system, we must be able to *distinguish between the execution of operating-system code and user-defined code*.
  - Hardware support that allows us to differentiate among various modes of execution.

# Operating-System Operations

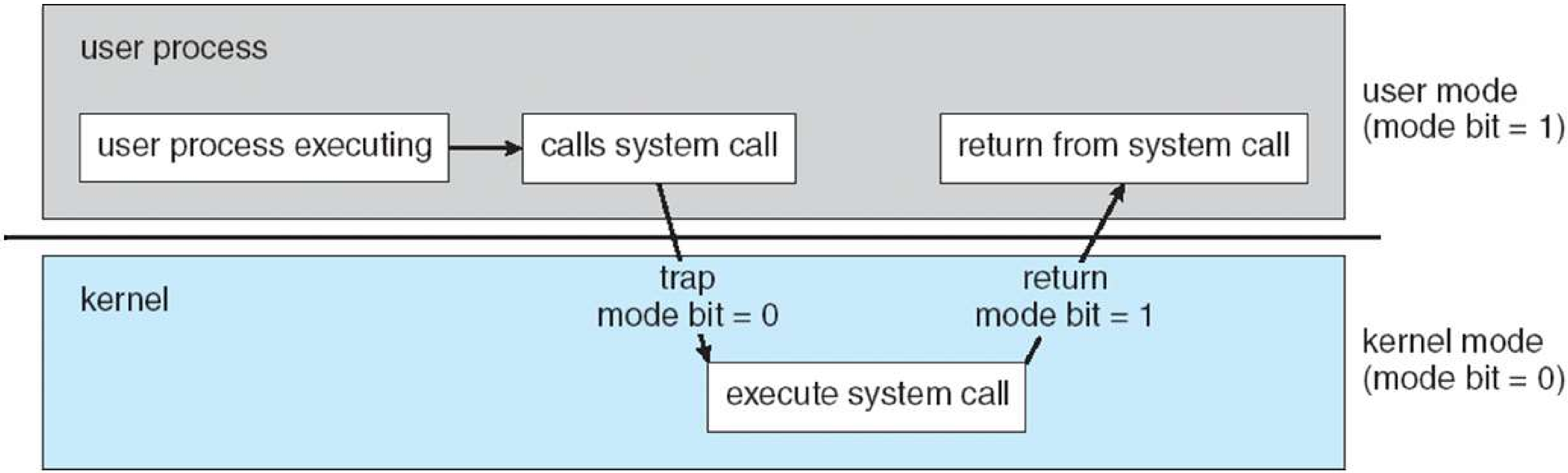
- **Hardware support**
  - At the very least, we need two separate modes of operation: **user** mode and **kernel** mode (also called privileged mode).
  - A bit, called the **mode bit**, is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

- At system boot time, the hardware starts in kernel mode. The operating system is then loaded and starts user applications in user mode.

- When a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode

- Some instructions designated as **privileged**, only executable in kernel mode

# Transition from User to Kernel Mode



# Operating-System Operations

- **Dual-Mode** and Multi-mode Operation
- The lack of a hardware-supported dual mode can cause serious shortcomings in an operating system.
- **System calls** provide the means for a user program to ask the operating system to perform *tasks reserved for the operating system* on the user program's behalf.

- We must ensure that the operating system **maintains control over the CPU**.
  - We **cannot allow a user program to get stuck in an infinite loop** or to **fail** to call system services and never return control to the operating system.
  - To accomplish this goal, we can use a timer (a counter that is decremented by the physical clock)
- A timer can be set to *interrupt the computer after a specified period*.
  - If the timer interrupts, control transfers automatically to the operating system

# Fin de la Clase

Gracias por
su asistencia y atención

¿ Preguntas ?