CSCI 3453 Operating System Concepts

- **Email**: salah.boukhris@ucdenver.edu
- **Office**: Lawrence Street Center Room 320E
- **Office Hours**: Wednesday: 3:30 PM to 5:00 PM, and by appointment.
- **Textbook:** *Operating System Concepts*, 10th Edition by Silberschatz. Galvin, and Gangne, Published by Wiley

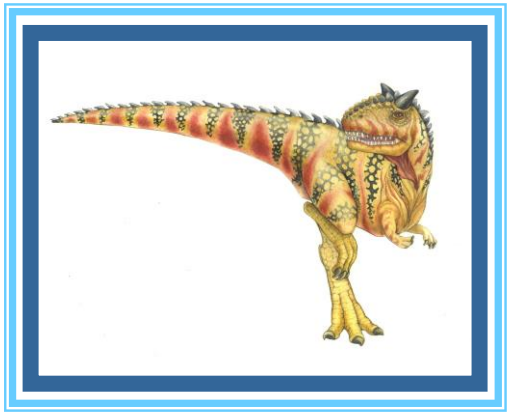
Grade Distribution

Item	Weight
Attendance & Class Participation	10%
Project	20%
Quizzes	20%
Midterm Exam	25%
Final Exam	25%

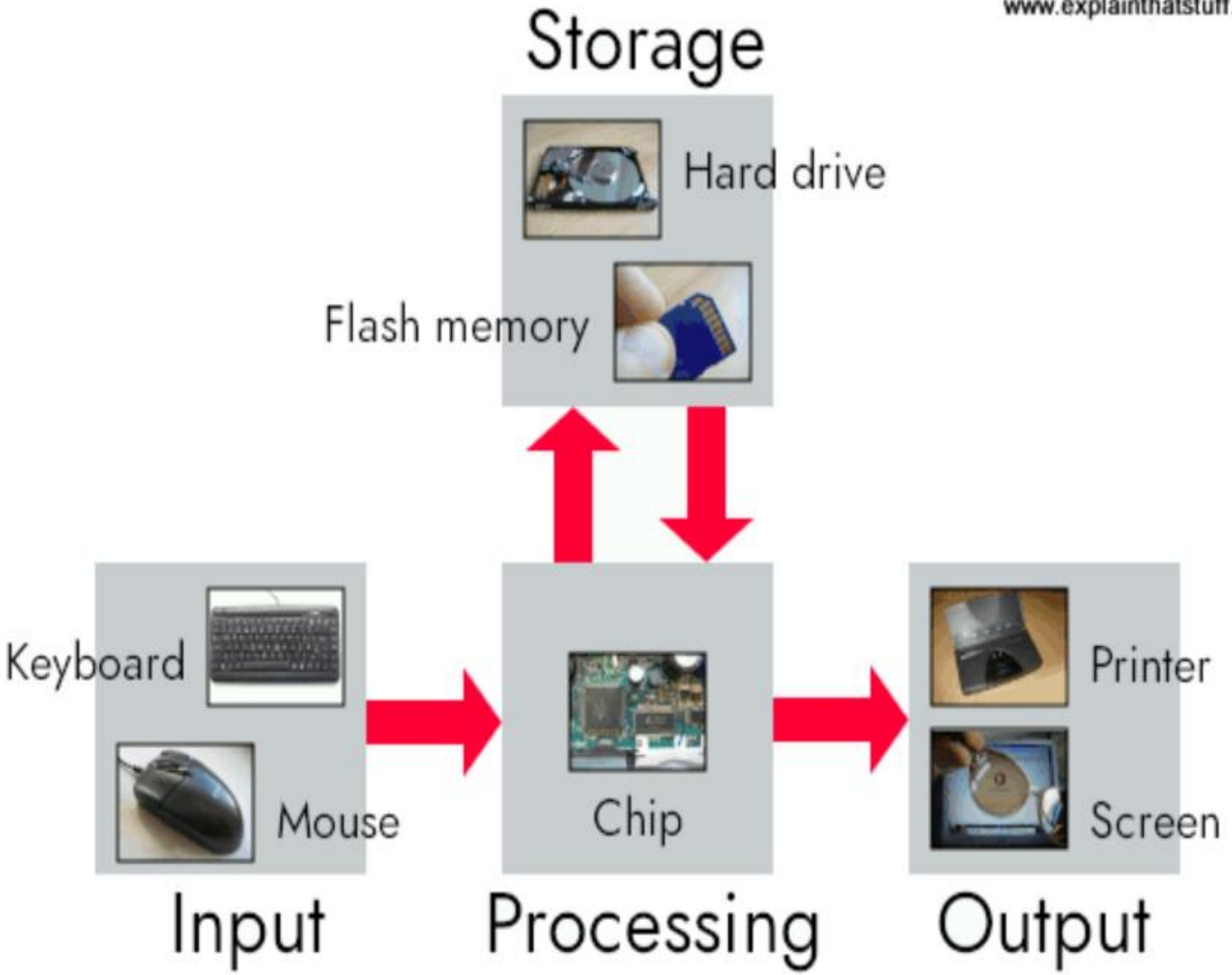
- No late work will be accepted
- Please put the "CSCI 3453" in the subject line of your email


NO
TEXTING
NO CELL PHONE
USE IN THIS
CLASSROOM
Reorder: NHE-14113 www.ComplianceSigns.com

Chapter 1: Introduction

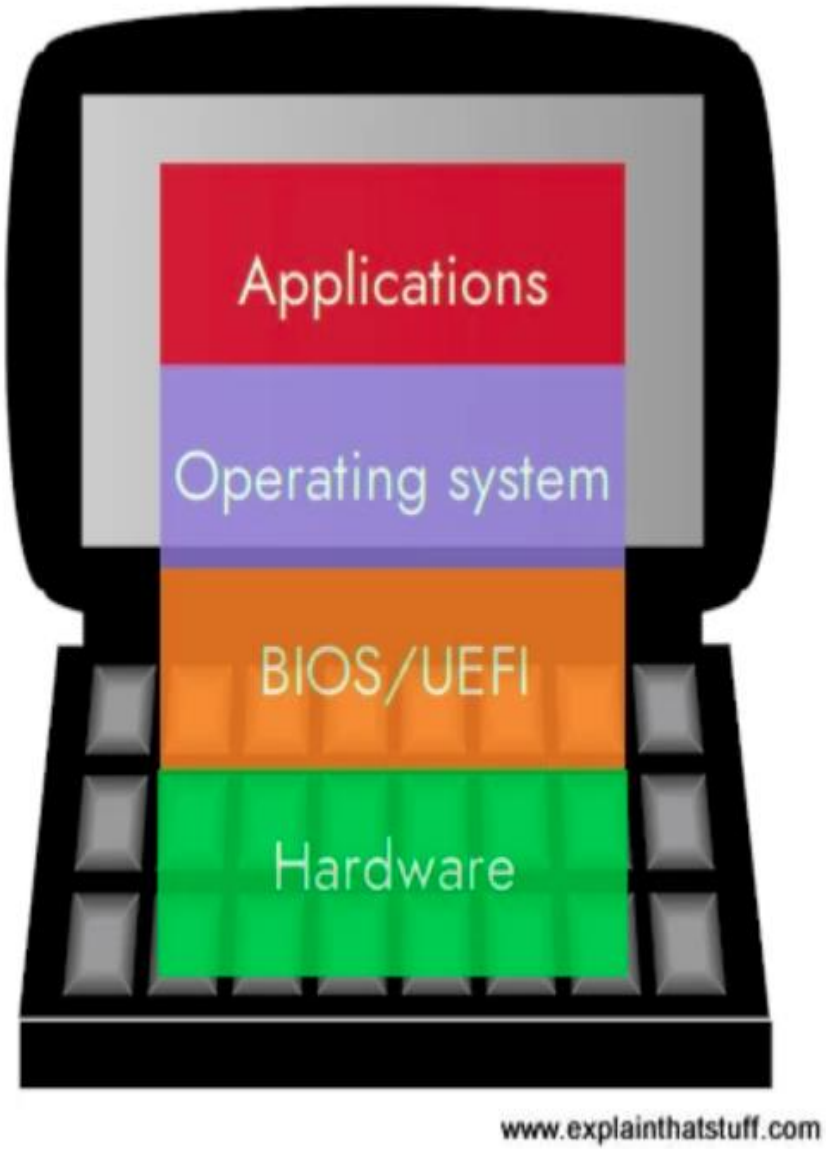


What is a computer?



Typical computer architecture

- **Applications** rely on the operating system to carry out their basic input, output, and so on. (ex. word processing)
- **Operating system**: it's the core software in a computer that (essentially) controls the basic tasks of input, output, storage, and processing. The same on all machines (ex. windows)
- **BIOS (Basic Input Output System)**, which is the link between the operating system software and the hardware.
- BIOS does vary from machine to machine according to the hardware configuration written by the hardware manufacturer.
- The BIOS is not, strictly speaking, *software*: it's a program semi-permanently stored into one of the computer's main chips, so it's known as **firmware**
- **Hardware,** physical components: CPU, RAM..



Chapter 1: Introduction

- What Operating Systems Do
- Computer-System Organization
- Computer-System Architecture
- Operating-System Operations
- Resource Management
- Security and Protection
- Virtualization
- Distributed Systems
- Kernel Data Structures
- Computing Environments
- Free/Libre and Open-Source Operating Systems

Objectives

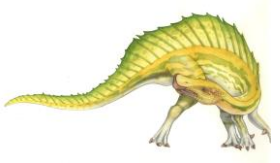
- Describe the general organization of a computer system and the role of interrupts
- Describe the components in a modern, multiprocessor computer system
- Illustrate the transition from user mode to kernel mode
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

What Does the Term Operating System Mean?

- An operating system is “fill in the blanks”
- What about:
  - **Car**: An operating system in a car might refer to the embedded system that controls the car's electronics, like the engine management system or infotainment system.
  - **Airplane**: Similar to a car, an airplane's operating system manages avionics, navigation, and communication systems to ensure the aircraft operates safely.
  - **Printer**: The operating system in a printer manages the print jobs, processes commands from the computer, and controls the mechanical parts of the printer.
  - **Washing Machine**: A washing machine's operating system controls the washing cycles, water levels, and temperature settings based on the selected program.
  - **Toaster**: The "operating system" in a toaster would be a simple control system that manages toasting time and temperature based on user input.
  - **Compiler**: In software, a compiler isn't an operating system itself, but it translates code written in a programming language into machine code that the operating system can execute
  - Etc.

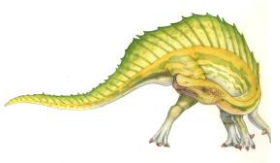
What is an Operating System?

- 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

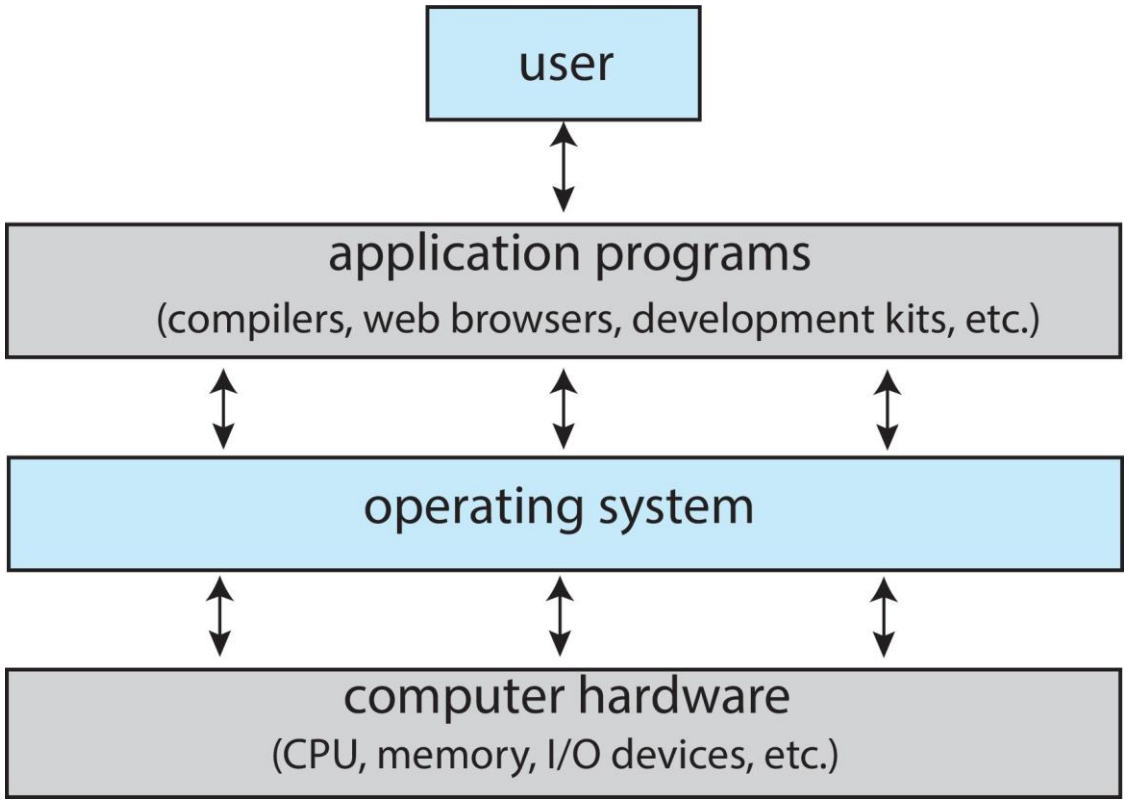


Computer System Structure

- 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, database systems, video games
  - **Users**
    - People, machines, other computers



Abstract View of Components of Computer



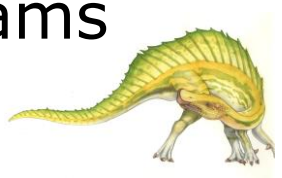
What Operating Systems Do?

- Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
  - Operating system is a **resource allocator** and **control program** making efficient use of HW and managing execution of user programs
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Mobile devices like smartphones and tables 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

Defining Operating Systems

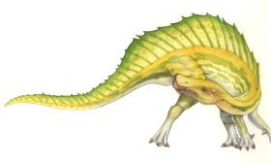
- Term OS covers many roles
  - Because of many designs and uses of OSes
  - Present in toasters through ships, spacecraft, game machines, TVs and industrial control systems
  - Born when fixed use computers for military became more general purpose and needed resource management and program control

Operating systems are everywhere,
from simple devices to advanced machines,
to help manage resources and run programs



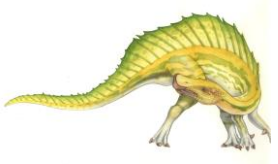
Operating System Definition

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- **Kernel:** "The one program running at all times on the computer" part of the operating system
- **Everything else is either**
  - A ***system program*** (ships with the operating system, but not part of the kernel) , or
  - An ***application program***, all programs not associated with the operating system
- Today's OSes for general purpose and mobile computing also include ***middleware*** – a set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics



System programs

- Manage the system **but do not** directly interact with the hardware like the kernel does.
  - **File Management Utilities**:
    - **File Explorer** (Windows) / **Finder** (macOS
    - **Command-line tools**ne.
  - **System Monitoring Tools**:
    - **Task Manager** (Windows) / **Activity Monitor** (macOS)
  - **User Management Tools**:
    - **User Account Control** (Windows).
  - **Networking Utilities**:
    - **Ping**, **Traceroute**.
    - **ifconfig** (Linux) / **ipconfig** (Windows).
  - **System Backup and Restore Utilities**:
    - **Time Machine** (macOS).
    - **Backup and Restore** (Windows).
  - **System Configuration Tools**:
    - **Control Panel** (Windows) / **System Preferences** (macOS).
    - **sysctl** (Linux/Unix).
  - **Security Utilities**:
    - **Windows Defender**
    - **Firewall**
  - **Software Update Managers**:
    - **Windows Update**: Manages updates for the Windows operating system.



The kernel

- It is the **core** component of an operating system that **directly interacts** with the hardware and manages system resources.
- Manages all the **low-level details**, allowing the user and applications to interact with the hardware in a simplified manner

1. **Process Management:**

   Scheduler: The kernel's scheduler decides which processes get to run on the CPU and for how long. It manages process prioritization and ensures that CPU time is fairly distributed among all running processes.

   Process creation and termination: The kernel provides system calls (e.g., fork, exec, exit in Unix-like systems) to create, execute, and terminate processes.

2. **Memory Management:**

   Virtual Memory: The kernel manages virtual memory, providing processes with a consistent address space while handling the actual physical memory allocation. This includes paging and swapping.

   Memory allocation: The kernel provides mechanisms for allocating and freeing memory, such as malloc and free in user-space programs, which are managed by the kernel in the background.

3. **Device Drivers:**

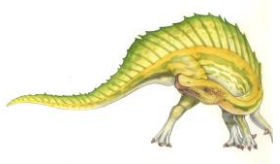
   Hardware Abstraction: The kernel includes device drivers, which are modules that allow the operating system to communicate with hardware devices like hard drives, network cards, printers, and more.

   Input/Output Operations: The kernel manages I/O operations, facilitating the transfer of data between hardware devices and processes.

The kernel

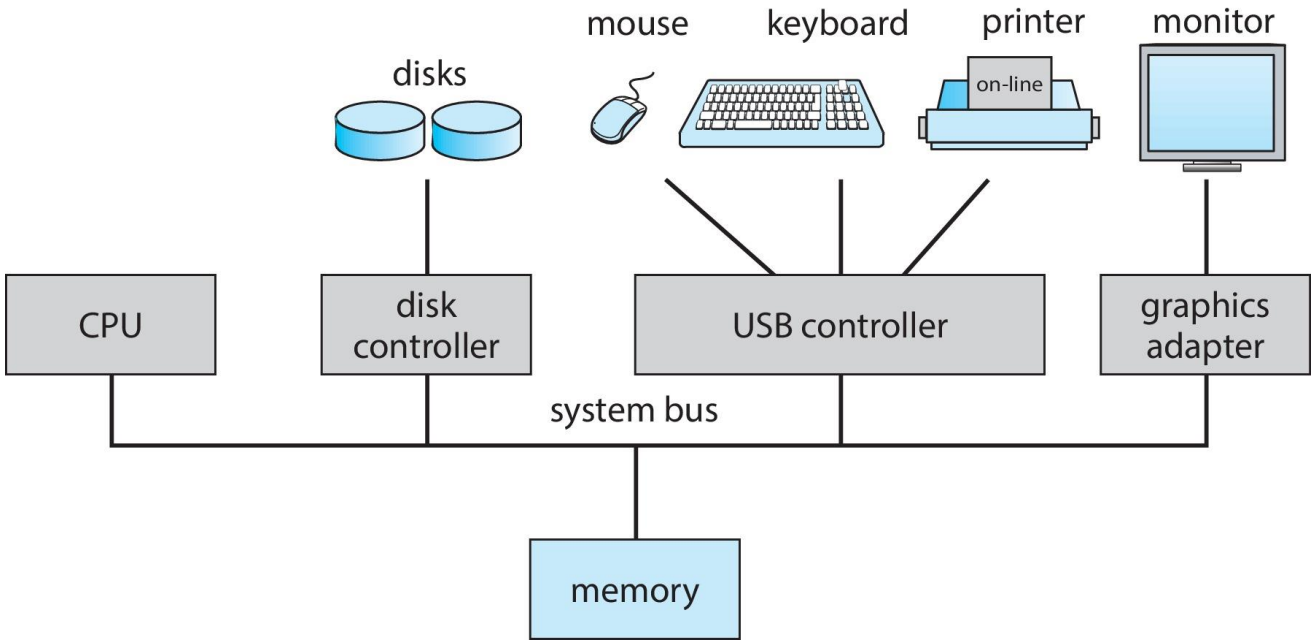
- **4. File System Management:**
  - **File Access**: The kernel controls access to files and directories, handling file read/write operations and ensuring data integrity.
  - **Mounting and Unmounting File Systems**: The kernel is responsible for mounting file systems, allowing the operating system to access storage devices.
- **5. Interprocess Communication (IPC):**
  - **Signals**: The kernel handles signals, which are used by processes to communicate with each other or to handle events like interrupts.
  - **Pipes and Message Queues**: The kernel provides IPC mechanisms like pipes and message queues for communication between processes.
- **6. Security and Access Control:**
  - **User Authentication**: The kernel enforces user authentication and access control mechanisms, ensuring that only authorized users can access certain resources.
  - **Permission Enforcement**: The kernel checks file and resource permissions, ensuring that processes can only access resources they have been granted permission to.

Overview of Computer System Structure



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 Operation

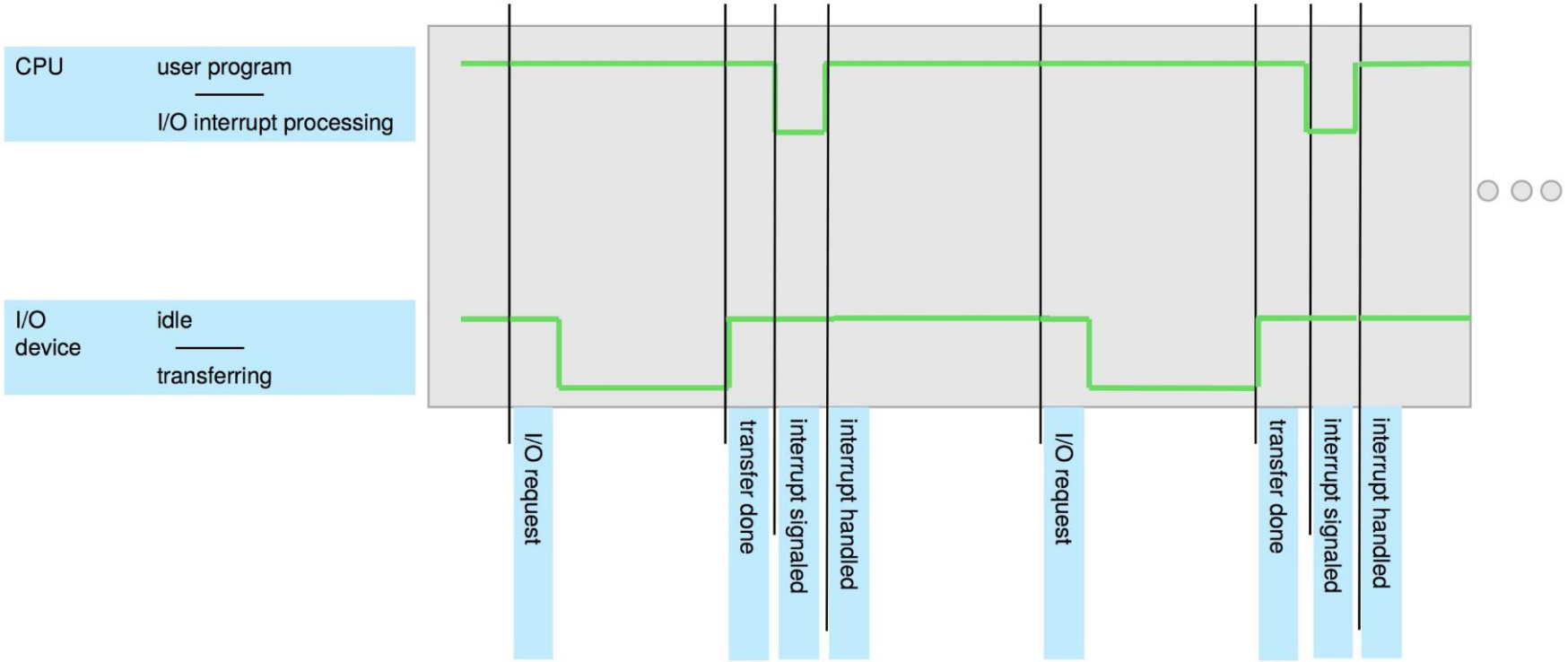
- I/O devices and the CPU can execute **concurrently**
- Each device controller is in charge of a particular device **type**
- Each device controller has a local **buffer**
- Each device controller type has an operating system **device driver** to manage it
- CPU moves data from/to main memory to/from 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**
- **Example Scenario:** When you save a file on your computer, the following kernel activities occur:
  - The **file system manager** in the kernel identifies the file and checks permissions.
  - The process scheduler allocates CPU time to the process requesting the save.
  - The **memory manager** ensures that data is correctly written to disk, using virtual memory if needed.
  - The **device driver** communicates with the storage hardware to physically write the data.

Common Functions of Interrupts

- **Interrupt** transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- An operating system is **interrupt driven**

Interrupt Timeline

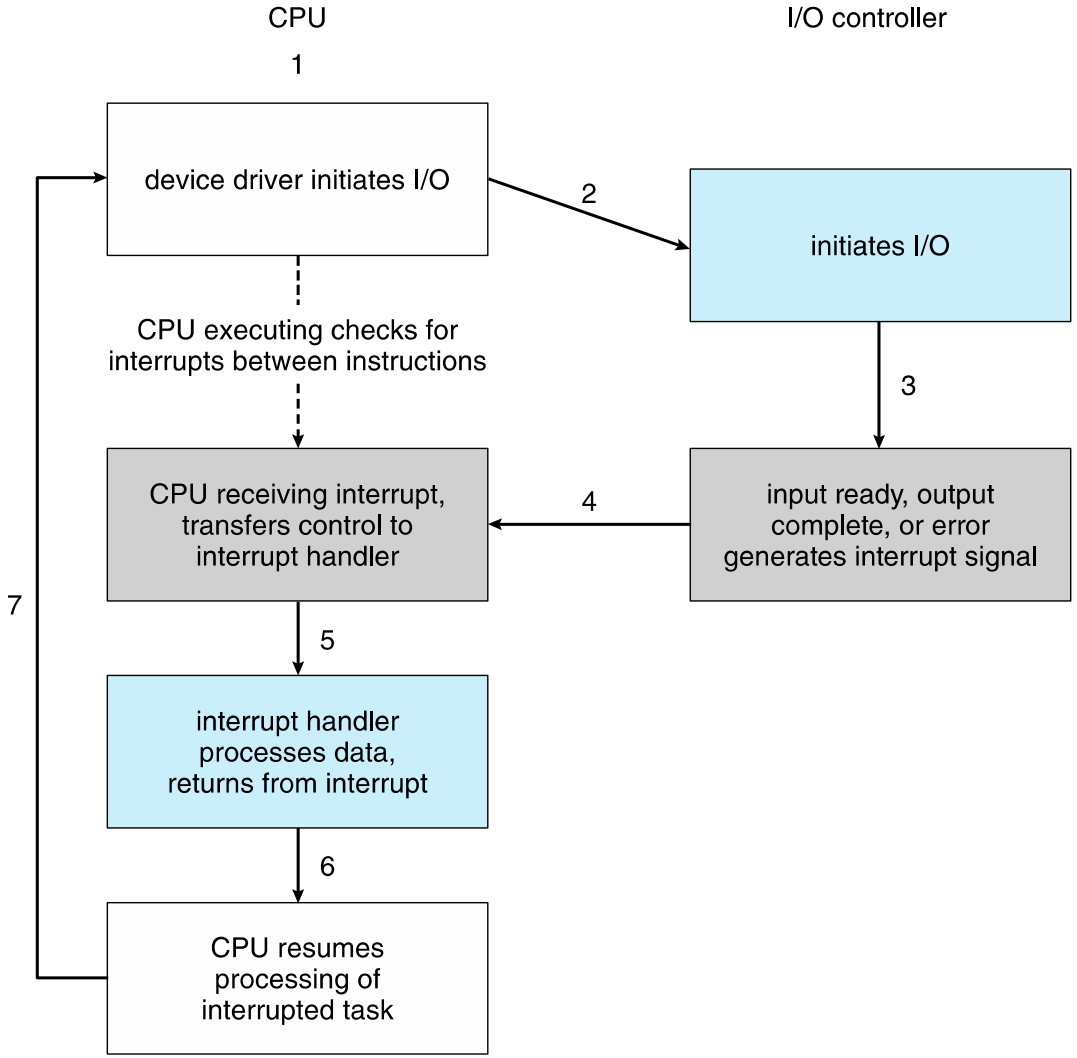
Interrupt sent after transfer is done



Interrupt Handling

- The operating system **preserves** the state of the CPU by storing the registers and the program counter
- Determines which **type** of interrupt has occurred:
- Separate segments of code **determine** what action should be taken for each type of interrupt

Interrupt-drive I/O Cycle



Examples of interrupts

1. Interrupt Transfers Control to the Interrupt Service Routine (ISR):

Example: press a key on your keyboard, generates an **interrupt**. The interrupt controller transfers control to the Interrupt Service Routine (ISR) for the keyboard, which is located at a specific **address** in the interrupt **vector**, determining which key was pressed, and then returns control to the operating system.

2. A Trap or Exception is a Software-Generated Interrupt:

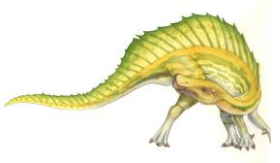
Example: tries to divide a number by zero in a program, causes a divide-by-zero exception, which is a **trap**. The CPU generates this trap and transfers control to the operating system, which then either handles the error (e.g., by showing an error message) or terminates the program.

3. An Operating System is Interrupt Driven:

Example: A desktop operating system, like Windows, constantly waits for interrupts to manage tasks. For instance, if a USB device is plugged into the computer, the OS doesn't need to constantly check the USB port (which would be inefficient). Instead, the USB hardware sends an **interrupt** signal when the device is connected, and the OS immediately handles this event, loading the necessary drivers and making the device ready for use.

I/O Structure

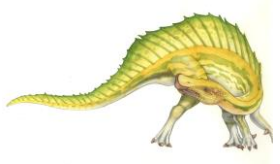
- Two methods for handling I/O
  - After I/O starts, control returns to user program only upon I/O completion
  - After I/O starts, control returns to user program without waiting for I/O completion



I/O Structure (Cont.)

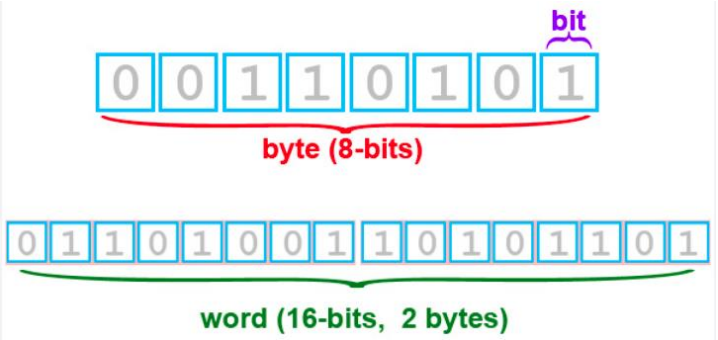
- After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing
- After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS 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
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

Storage Structure



Storage Definitions and Notation Review

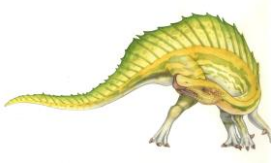
- **Basic Unit of Storage**: The bit is the fundamental unit of computer storage, representing values of 0 or 1.
- **Bytes**: A byte consists of 8 bits and is the smallest convenient chunk of storage in most computers.
- **Words**: A word is a unit of data specific to a computer's architecture, typically composed of multiple bytes (e.g., a 64-bit word consists of 8 bytes).
- **Computer Operations**: Computers often execute operations using their native word size rather than bytes.
- **Storage Measurement**: Computer storage is measured in bytes and larger units derived from bytes:
  - Kilobyte (KB) = 1,024 bytes
  - Megabyte (MB) = $1{,}024^2$ bytes
  - Gigabyte (GB) = $1{,}024^3$ bytes
  - Terabyte (TB) = $1{,}024^4$ bytes
  - Petabyte (PB) = $1{,}024^5$ bytes



- **Manufacturers' Rounding**: Computer manufacturers often round these numbers, approximating 1 MB as 1 million bytes and 1 GB as 1 billion bytes.
- **Networking Exception**: Networking measurements are typically given in bits, as networks transfer data one bit at a time.

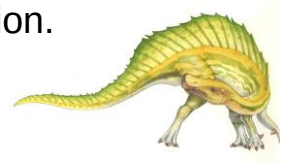
Storage Structure

- Main memory – only large storage media that the CPU can access directly
  - **Random access**
  - Typically **volatile**
  - Typically **random-access memory** in the form of **Dynamic Random-access Memory (DRAM)**
- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
- **Register in Processors/CPUs**
  - Built directly into the processor or CPU
  - Store and manipulate data during the execution of instructions
  - Referred to as the 32-bit processors and 64-bit processors



Types of CPU Registers

- Depending on the CPUs architecture and design, the type and number of registers can vary. Common types of registers found in a CPU may include:
- **Program Counter (PC)**: The Program Counter keeps track of the memory address of the next instruction to be fetched and executed.
- **Instruction Register (IR)**: The Instruction Register holds the currently fetched instruction being executed.
- **Accumulator (ACC)**: The Accumulator is a general-purpose register used for arithmetic and logical operations. It stores intermediate results during calculations.
- **General-Purpose Registers (R0, R1, R2…)**: These registers are used to store data during calculations and data manipulation. They can be accessed and utilized by the programmer for various purposes.
- **Address Registers (AR)**: Address Registers store memory addresses for data access or for transferring data between different memory locations.
- **Stack Pointer (SP)**: The Stack Pointer points to the top of the stack, which is a region of memory used for temporary storage during function calls and other operations.
- **Data Registers (DR)**: These registers store data fetched from memory or obtained from input/output operations.
- **Status Register/Flags Register (SR)**: The Status Register or Flags Register contains individual bits that indicate the outcome of operations, such as carry, overflow, zero result, and others. These flags help in making decisions and controlling program flow based on the results of previous operations.
- **Control Registers (CR)**: Control Registers manage various control settings and parameters related to the CPU's operation, such as interrupt handling, memory management, and system configuration.

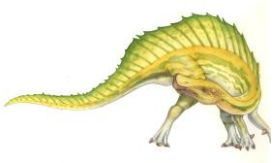


Storage Structure (Cont.)

- **Hard Disk Drives (HDD)** – 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
- **Non-volatile memory (NVM)** devices– faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular as capacity and performance increases, price drops

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage, high-speed memory located either within or very close to the CPU.
- **Device Driver** for each device controller to manage I/O
  - Provides uniform interface between controller and kernel



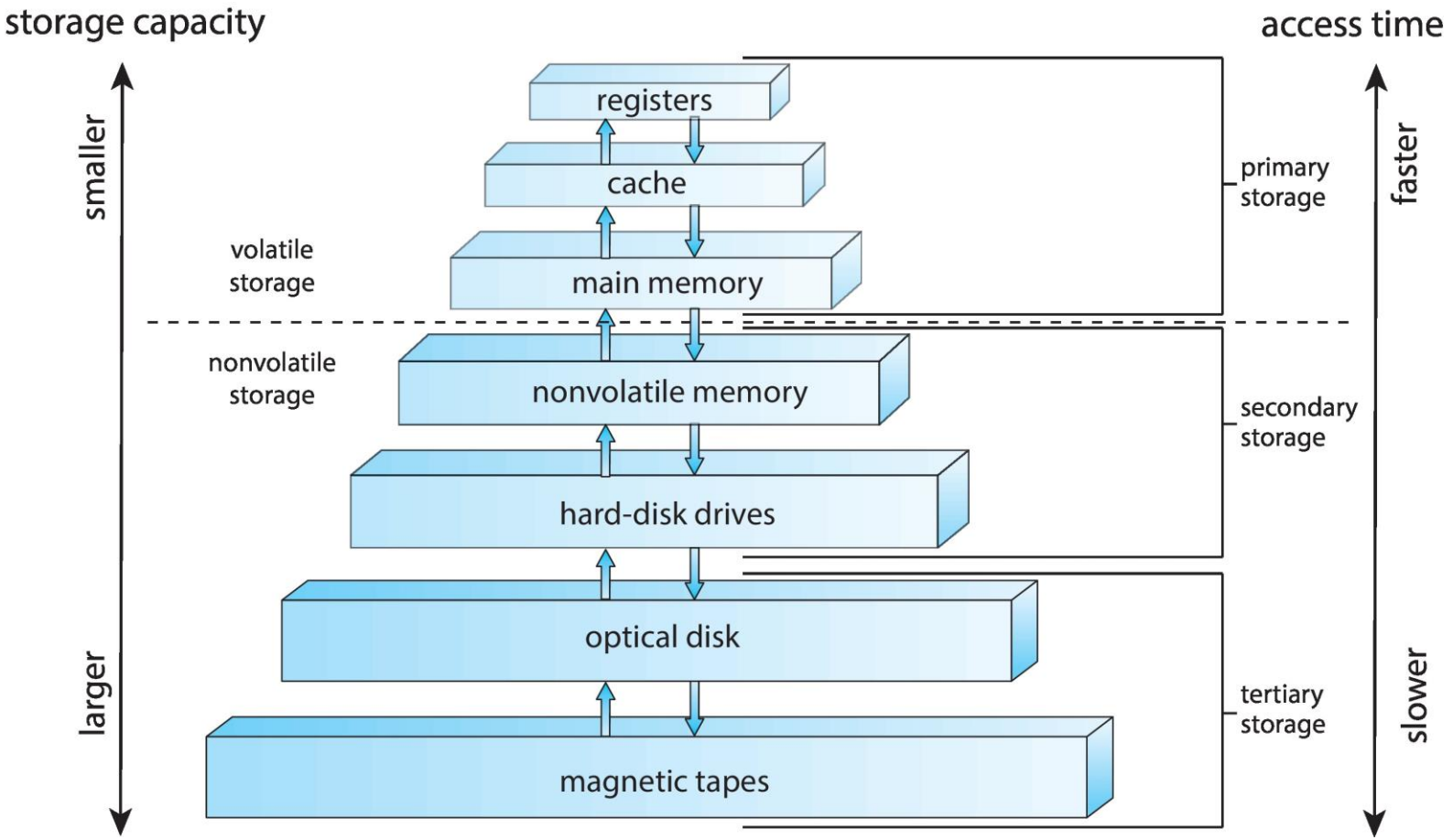
Cache is a type of memory

- **1. Location and Integration:**
  - **Within the CPU**: The most critical levels of cache (L1 and often L2) are typically located directly within the CPU chip itself. This allows for extremely fast access to data and instructions that the CPU needs to execute. These caches are very small but incredibly fast.
  - **Near the CPU**: L3 cache, if present, is often larger than L1 and L2 and is sometimes shared between multiple CPU cores. It might be located on the CPU chip or on a separate chip but still within close proximity to the CPU.
- **2. Type of Memory:**
  - **SRAM (Static RAM)**: Cache memory is usually made from SRAM, which is faster but more expensive than the DRAM (Dynamic RAM) used for the main memory. Unlike DRAM, which needs to be refreshed periodically, SRAM retains data as long as power is supplied, making it faster but also more costly.
- **3. Purpose and Function:**
  - **Temporary Storage**: Cache serves as temporary storage for frequently accessed data and instructions. Its purpose is to speed up data access for the CPU by storing this information closer to the processing unit.
  - **Data Replication**: Cache doesn’t hold original data. Instead, it holds copies of data from the main memory that the CPU is likely to need soon. If the CPU finds the required data in the cache (a "cache hit"), it avoids the slower process of fetching it from the main memory.

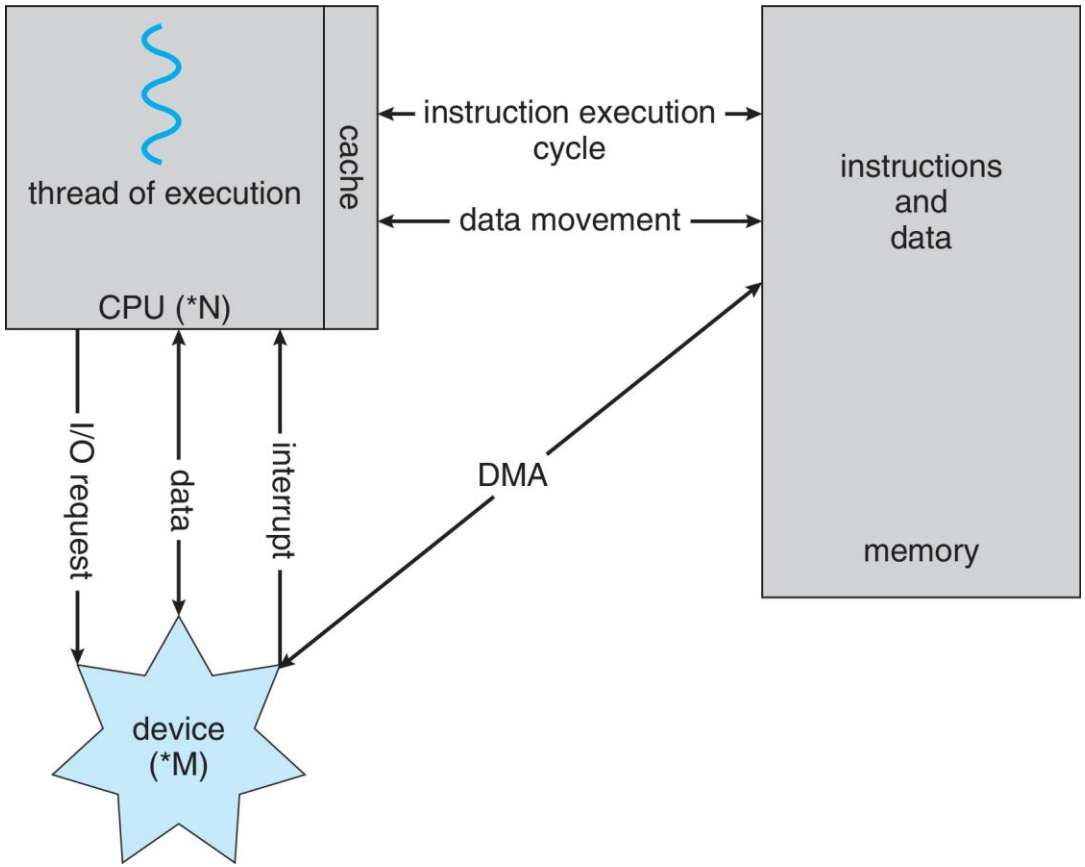
Cache is a type of memory

- **4. Hierarchical Structure:**
  - **Multi-Level Caches**: Modern CPUs typically use multiple levels of cache:
    - **L1 Cache**: The smallest and fastest, directly integrated into each CPU core.
    - **L2 Cache**: Larger but slightly slower, can be either per core or shared between cores.
    - **L3 Cache**: Even larger and slower, usually shared among all cores in a CPU.
- **5. Relationship with Main Memory:**
  - **Complementary Role**: Cache is not a replacement for main memory but a complement to it. While the main memory (RAM) stores larger amounts of data, cache provides quicker access to the most frequently used portions of that data.
- **6. Volatile Memory:**

Storage-Device Hierarchy



How a Modern Computer Works



- Input
- Processing
- Memory
- Storage
- Output
- Data Flow
- Operating System

A von Neumann architecture

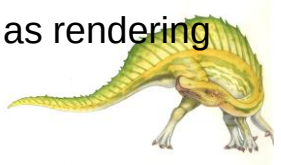
Process vs Thread

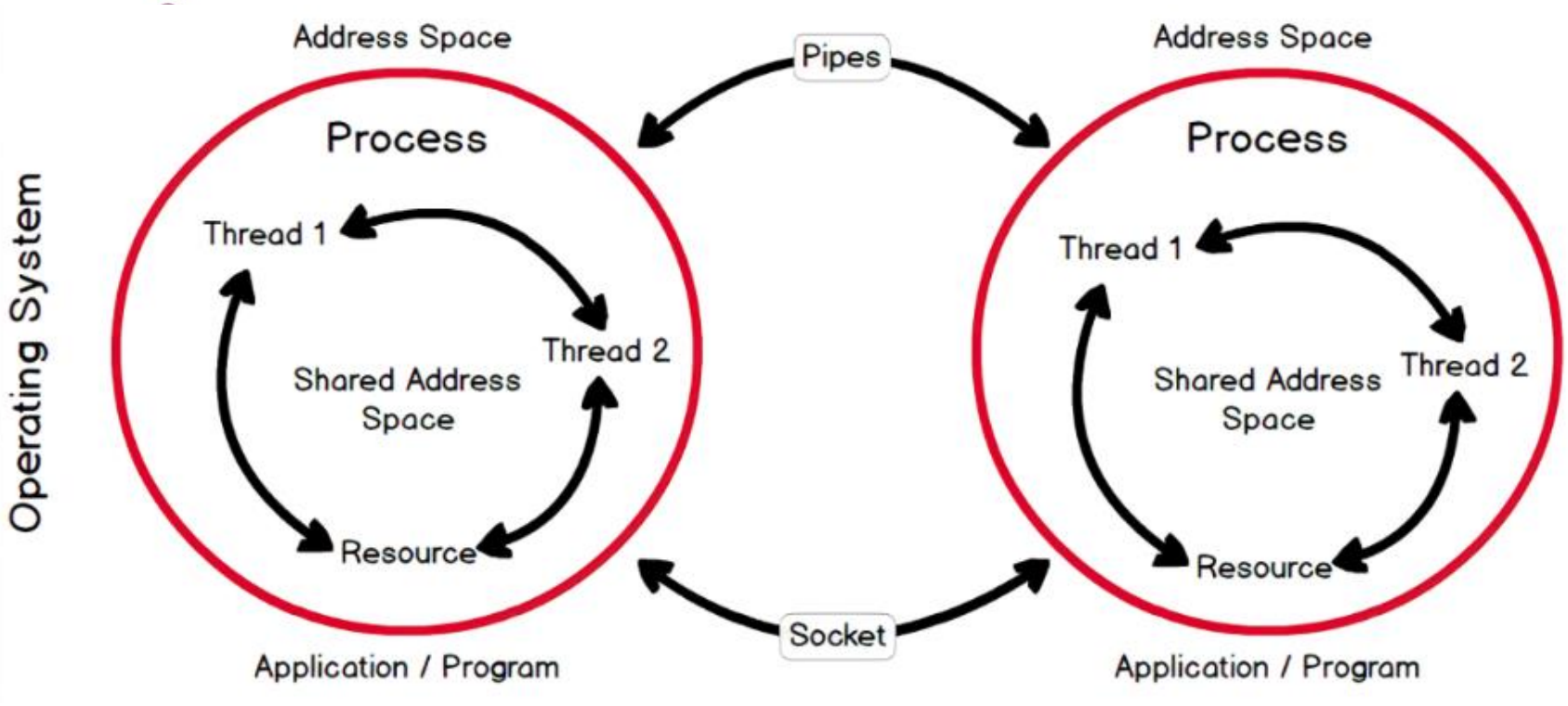
Processes:

- **Definition**: A process is an independent program in execution, including its own memory space, code, data, and system resources.
- **Memory**: Each process has its own separate memory space. This isolation helps prevent one process from affecting another.
- **Resource Allocation**: Processes are allocated system resources like CPU time and memory independently.
- **Communication**: Inter-process communication (IPC) is required for processes to exchange data. IPC mechanisms include pipes, message queues, and shared memory.
- **Examples**: Running a web browser, a text editor, or a file manager as separate applications.

Threads:

- **Definition**: A thread is a smaller unit of execution within a process. Multiple threads can exist within a single process and share the same memory space.
- **Memory**: Threads within the same process share the same memory space, including code, data, and resources.
- **Resource Allocation**: Threads share resources of the process they belong to, such as open files and memory, making it more efficient in terms of resource usage.
- **Communication**: Threads within the same process can communicate more easily and quickly since they share memory space.
- **Examples**: A web browser may use multiple threads to handle different tasks such as rendering web pages, handling user input, and managing network connections concurrently





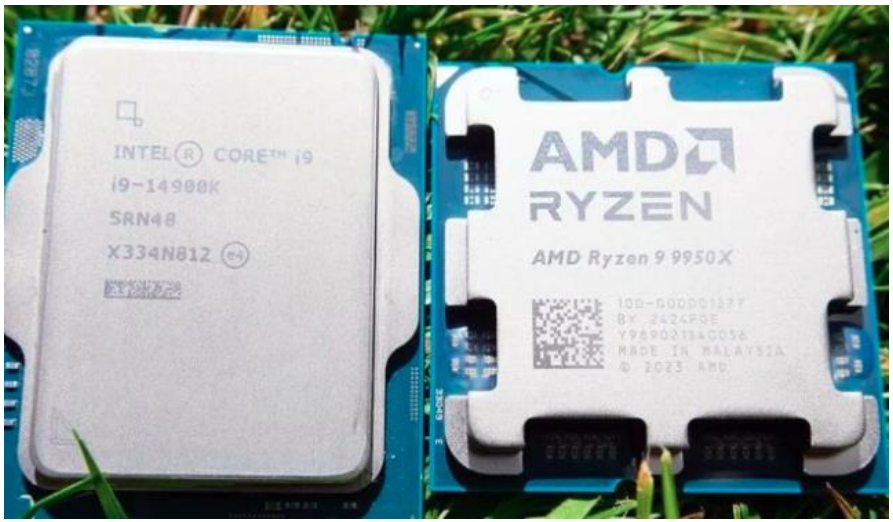
- **Pipes**:
  - **Use Case**: Simple IPC between processes on the same machine, command-line environment.
  - **Data Flow**: Can be unidirectional or bidirectional.
  - **Creation**: pipe() (anonymous) and mkfifo() (named).

- **Sockets**:
  - **Use Case**: More flexible IPC for network communication or local communication between unrelated processes, suitable for building networked applications.
  - **Data Flow**: Bidirectional and can handle both local and remote communication.
  - **Creation**: socket(), bind(), listen(), accept(), connect().

Specs

Model	Cores / Threads	TDP	L3 Cache	Base Clock	Boost Clock
Ryzen 9 9950X	16 / 32	170W	64MB	4.3GHz	5.7GHz
Core i9-14900K	24 / 32	125/253W	36MB	3.2GHz	6.0GHz

Note: TDP stands for Thermal Design Power, indicates the maximum amount of heat the processor is expected to generate under typical load conditions.



File Options View

Processes | Performance | App history | Startup | Users | Details | Services

CPU
25% 2.39 GHz

Memory
6.4/7.9 GB (81%)

Disk 0 (C:)
SSD
1%

Ethernet
Ethernet 3
S: 0 R: 0 Kbps

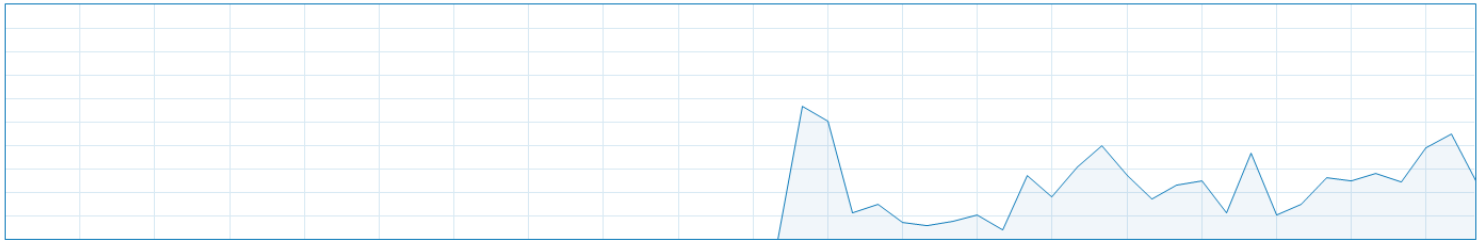
Wi-Fi
Wi-Fi
S: 0 R: 0 Kbps

GPU 0
Intel(R) HD Graphics 5...
1%

CPU

Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz

% Utilization 100%



60 seconds 0

Utilization	Speed	Processes	Threads	Handles
25%	2.39 GHz	265	3783	116769

Up time
2:02:08:18

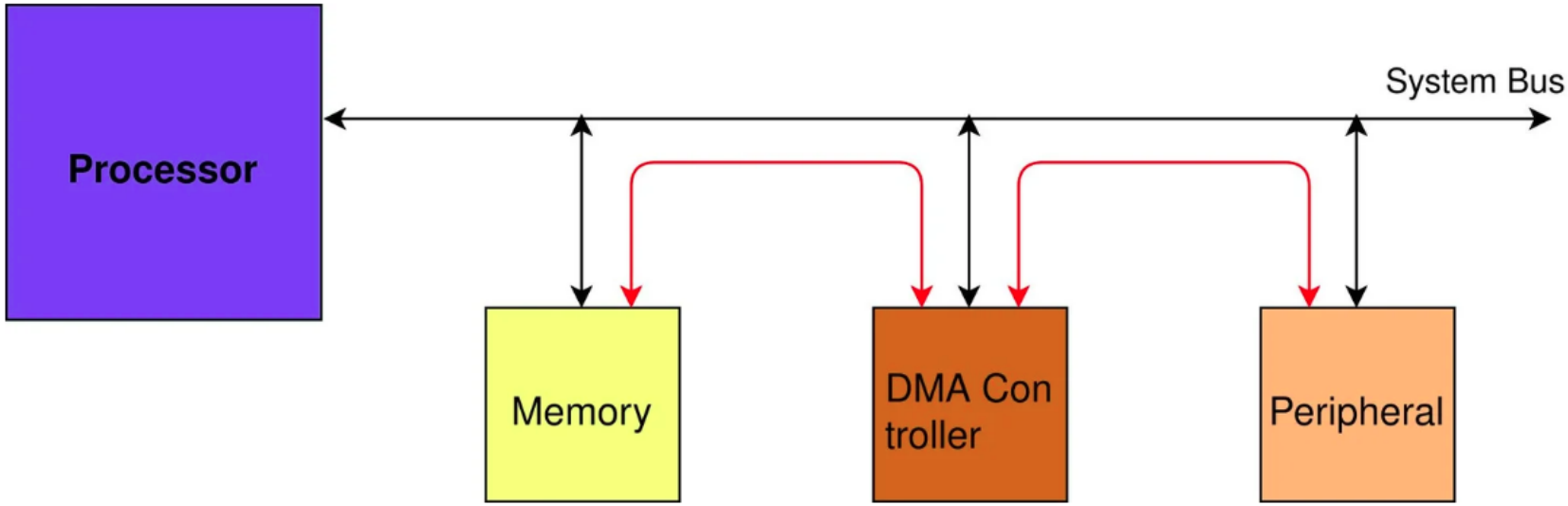
Base speed:	2.59 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualization:	Disabled
Hyper-V support:	Yes
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	4.0 MB

Fewer details | Open Resource Monitor

Direct Memory Access Structure

- 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 one interrupt is generated per block, rather than the one interrupt per byte

Direct Memory Access (DMA) allows hardware devices to transfer data directly to and from memory without involving the CPU.
Example: moving files from one folder to another. Drag and drop each file (like the CPU handling data transfers). With DMA, is an assistant that can move those files directly between folders, so CPU is freeing up to do other tasks.



Operating-System Operations

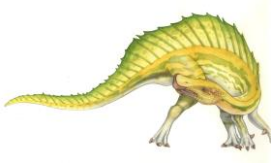
- Bootstrap program – simple code to initialize the system, load the kernel
- Kernel loads
- Starts **system daemons** (background process services provided outside of the kernel)
- Kernel **interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap**):
    - Software error (e.g., division by zero)
    - Request for operating system service – **system call**
    - Other process problems include infinite loop, processes modifying each other or the operating system

•**Bootstrap program:** Think of it as the ignition key in a car. When you turn the key, the car's engine starts up. Similarly, the bootstrap program is a small piece of code that starts up the computer, loading the operating system's kernel.

•**Interrupts:** These are like alerts or signals. For example, if you press the brake (a hardware interrupt), the car responds immediately. Similarly, if software needs help (like encountering a division by zero), it sends a signal (a software interrupt) to the kernel for handling.

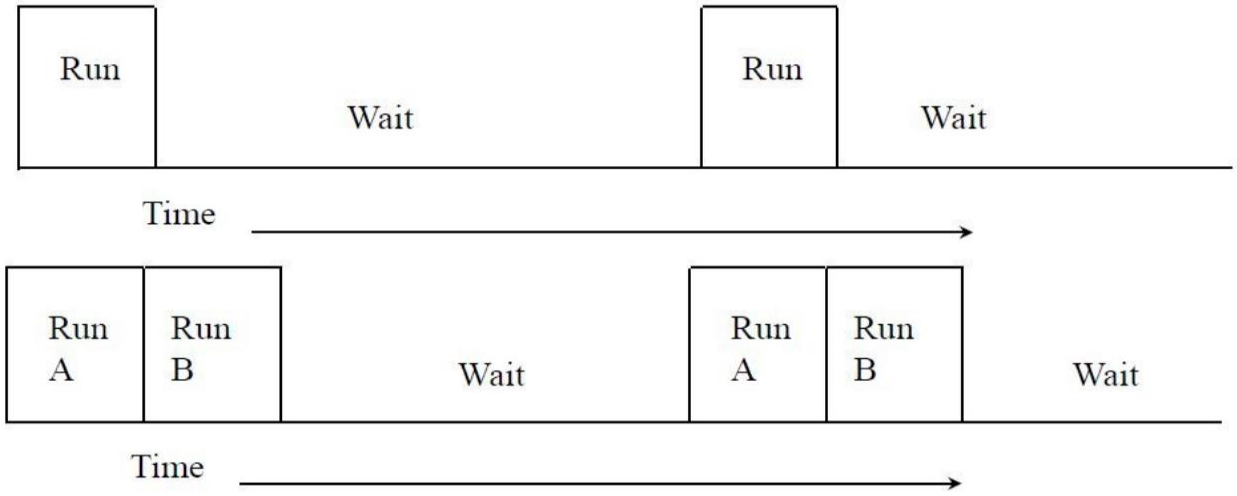
•**Problems:** If something goes wrong, like pressing the gas forever (an infinite loop) or trying to mess with the car's engine (a process modifying the operating system), the system needs to handle or prevent it to keep everything running smoothly.

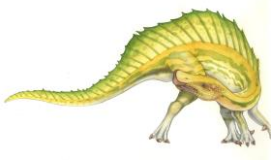
•**Kernel:** Once the kernel is loaded, it's like the car's engine, managing everything under the hood. It starts essential background services (like turning on the headlights, radio, etc.) called **daemons**.



Multiprogramming (Batch system)

- Single user cannot always keep CPU and I/O devices busy
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When job has to wait (for I/O for example), OS switches to another job

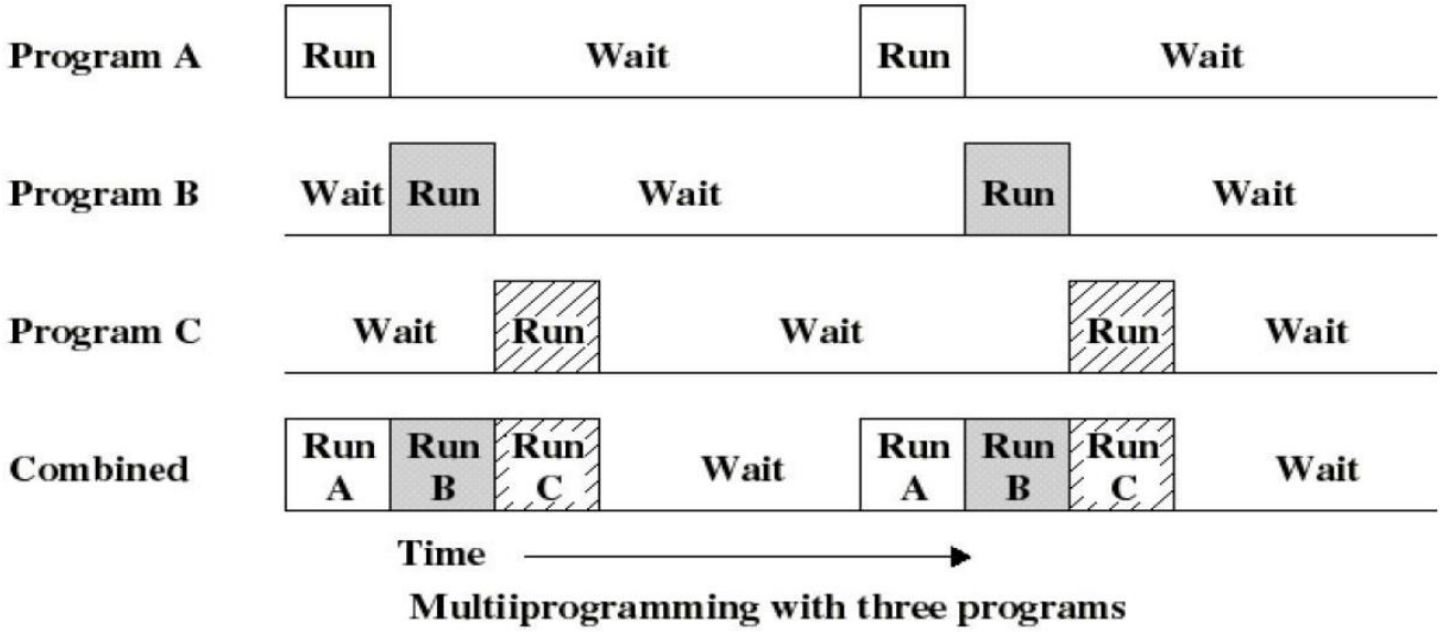




Like having multiple PWMs that fit within eachother (don't overlap)

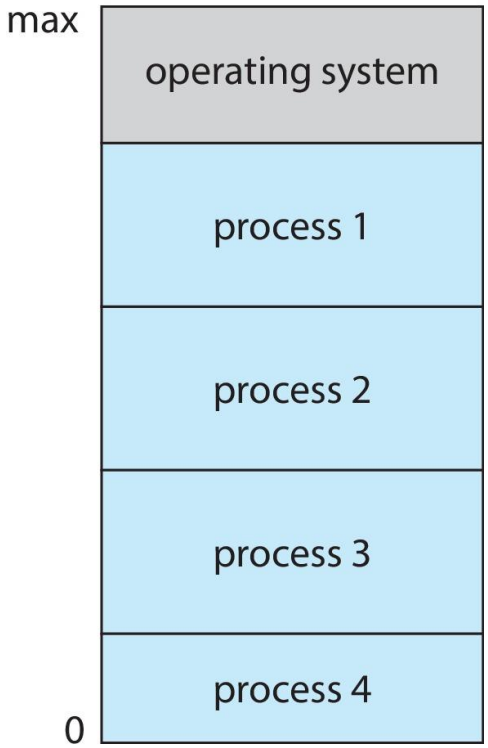
Multitasking (Timesharing)

- A logical extension of Batch systems– the CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second Actually < 1 ns
  - Each user has at least one program executing in memory ⇨ **process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory



Multiiprogramming with three programs

Memory Layout for Multiprogrammed System

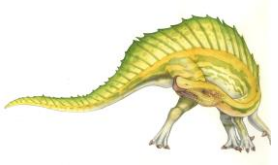


If memory is full and another process is needed. Load the process in the virtual memory instead of swapping out a process in the memory.

Virtual memory allows your computer to use part of the hard drive as an extension of RAM, so it can handle more tasks even if the actual RAM is full.

Dual-mode Operation

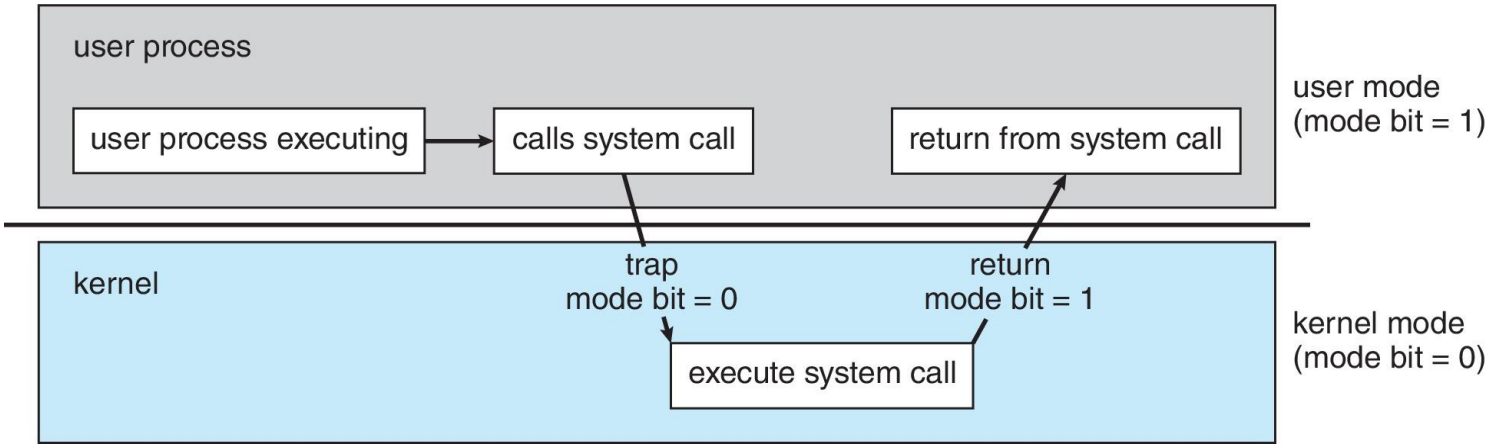
- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
- **Mode bit** provided by hardware
  - Provides ability to distinguish when system is running user code or kernel code.
  - When a user is running ⇨ mode bit is “user”
  - When kernel code is executing ⇨ mode bit is “kernel”
- How do we guarantee that user does not explicitly set the mode bit to “kernel”?
  - System call changes mode to kernel, return from call resets it to user
- Some instructions designated as **privileged**, only executable in kernel mode



- **Dual-mode operation** is like having two separate rooms in a building: a public room (user mode) and a secure control room (kernel mode).
- **User mode** is when you're in the public room, where you can do everyday tasks but can't access the building's controls.
- **Kernel mode** is when you're in the control room, where you have access to everything and can control the entire building.
- **Mode bit** is like a switch that tells the system whether you're in the public room or the control room.
- **System call** is like pressing a button to ask the control room to do something for you, such as unlocking a door.
- **Privileged instructions** are special controls that only the control room can operate.

This setup ensures that regular users can't access or damage critical parts of the system.

Transition from User to Kernel Mode



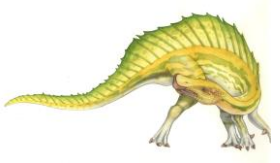
Example: Reading a File

1. User Application Requests File Read
2. Mode Transition to Kernel Mode
3. Kernel Mode Executes File Read
4. Mode Transition Back to User Mode
5. Application Receives Data and Continues

Timer

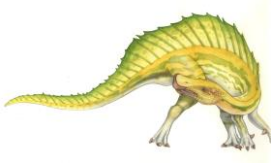
- Timer to prevent infinite loop (or process hogging resources)
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Example: timer mechanism is like a safeguard that ensures no single program (like the game) can keep the computer's resources for too long, allowing the OS to manage time and resources fairly among all running programs



Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a ***passive entity;*** process is an ***active entity***.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads



- **Process as a program in action:** Imagine you have a text document (program) saved on your computer. When you open it with a word processor (process), the document becomes active, allowing you to read, write, and edit it.
- **Resources for the process:** To edit the document (process), the word processor needs resources: CPU to process your commands, memory to hold the document while you work on it, and files to save your changes.
- **Process termination:** When you close the word processor, it cleans up by saving your document and freeing up the memory and CPU that were being used.
- **Single-threaded process:** If you're working on one document (single-threaded), the word processor follows your commands one by one, like typing a letter or saving a file, until you're done.
- **Multi-threaded process:** If you have multiple documents open (multi-threaded), each document window has its own cursor (program counter), and you can switch between them, working on them simultaneously.
- **Concurrency:** If you have many programs open, like a web browser, a word processor, and an email client, your computer's CPU manages all these programs (processes) by giving each a little bit of time to run, making it seem like they're all running at the same time.

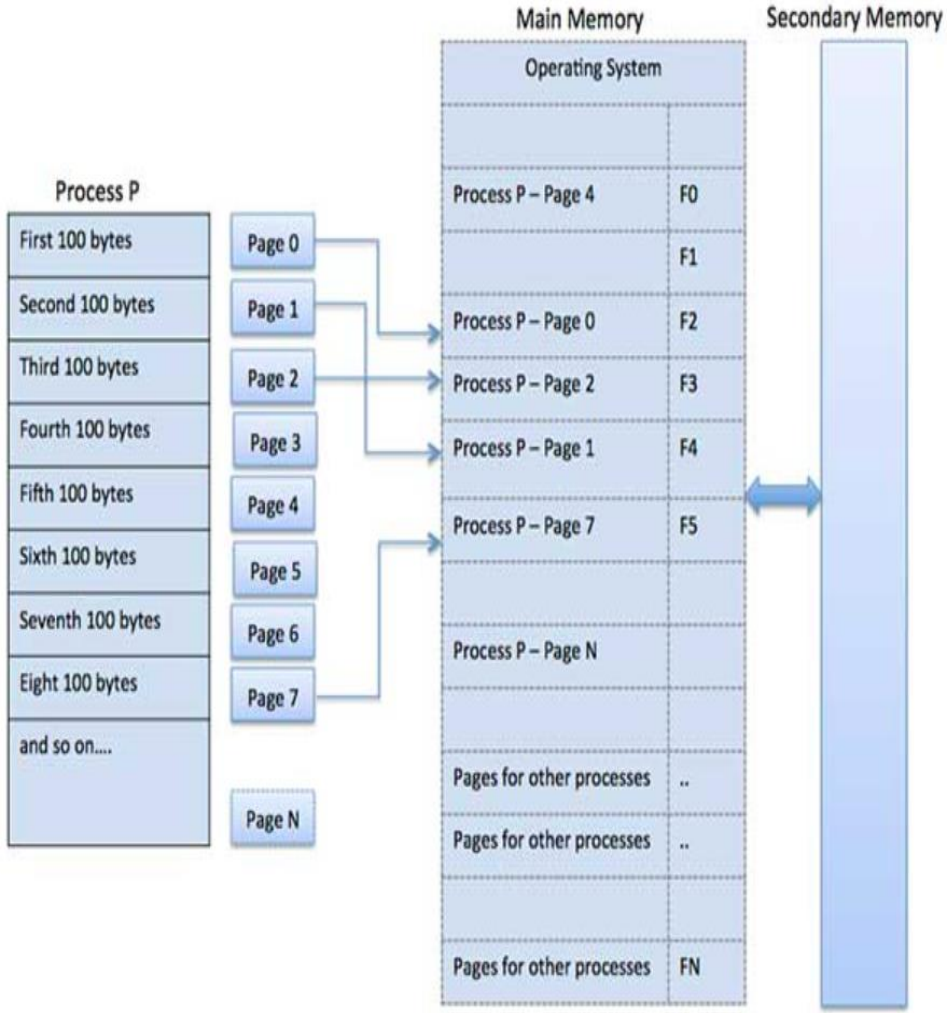
Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Memory Management

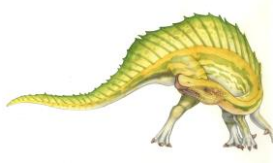
- To execute a program all (or part) of the instructions must be in memory
- All (or part) of the data that is needed by the program must be in memory
- Memory management determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Memory management activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed



https://www.tutorialspoint.com/operating_system/os_memory_management.htm

File-system Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control on most systems to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media



Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Mounting and unmounting
    - **mounting** is like connecting the USB drive to the computer so you can use it, and **unmounting** is like safely disconnecting it when you're done.
  - Free-space management
  - Storage allocation
  - Disk scheduling
  - Partitioning
  - Protection

Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- 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
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Characteristics of Various Types of Storage

1 nanosecond = 10^{-9} seconds = 0.000000001 seconds.

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

Explicit Movement:

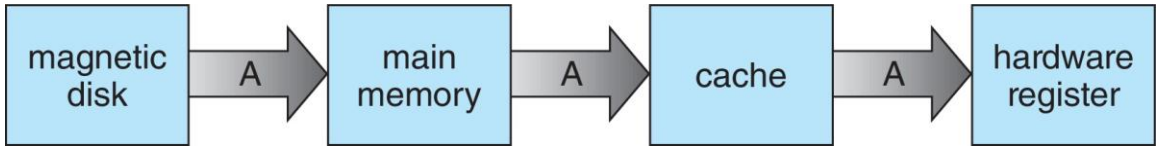
- When save a document from the computer's RAM (temporary storage) to the hard drive (permanent storage), **explicitly** telling the system to move data from one level of storage to another.

Implicit Movement:

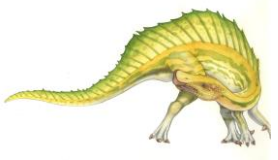
- When computer automatically moves data between RAM and the hard drive (like when it uses virtual memory), the operating system handles this behind the scenes without needing to do anything. This movement is implicit because the system decides when and how to move data based on its own needs.

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy



- Multiprocessor environment must provide **cache coherency** in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a data can exist
  - Various solutions covered in Chapter 19



Scenario:

CPU 1 reads a value from the database, say the balance of a bank account, and stores it in its cache. Let's say the balance is $1,000.

CPU 2 also reads the same balance and stores it in its cache, which is also $1,000.

Update:

Now, **CPU 1** processes a transaction that updates the balance to $1,200. It updates its cache with the new balance.

However, **CPU 2** still has the old balance of $1,000 in its cache.

Problem Without Cache Coherency:

If **CPU 2** tries to read the balance again, it might mistakenly use the outdated $1,000 value from its cache, leading to incorrect calculations or decisions.

Cache Coherency:

To avoid this problem, the multiprocessor system implements **cache coherency**.

When **CPU 1** updates the balance to $1,200, this change is automatically communicated to **CPU 2**, which updates its cache with the new balance.

Both CPUs have the correct, most recent balance of $1,200 in their caches.

I/O Subsystem

- Responsible for managing all the input and output operations in a computer system.
- An intermediary between the CPU and the peripheral devices, such as keyboards, mice, printers, disk drives, and network interfaces.

Key Responsibilities of the I/O Subsystem:

- **Memory Management for I/O**:
  - **Buffering**: Temporarily stores data while it's being transferred to smooth out differences in data flow rates.
  - **Caching**: Stores frequently accessed data in faster storage to improve performance.
  - **Spooling**: Manages the overlapping of input/output operations, allowing one job to proceed while another job's output is being processed.
- **General Device-Driver Interface**:
  - Provides a standard way for the OS to interact with various hardware devices without needing to know the specifics of each one.
- **Specific Device Drivers**:
  - These are specialized pieces of software that control specific hardware devices, translating general commands from the OS into specific instructions that the hardware can understand

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

Technical Example: User Account Control (UAC) in Windows prompts for administrator approval before allowing certain actions. This combines both protection (only allowing authorized actions) and security (verifying that the user is authorized).

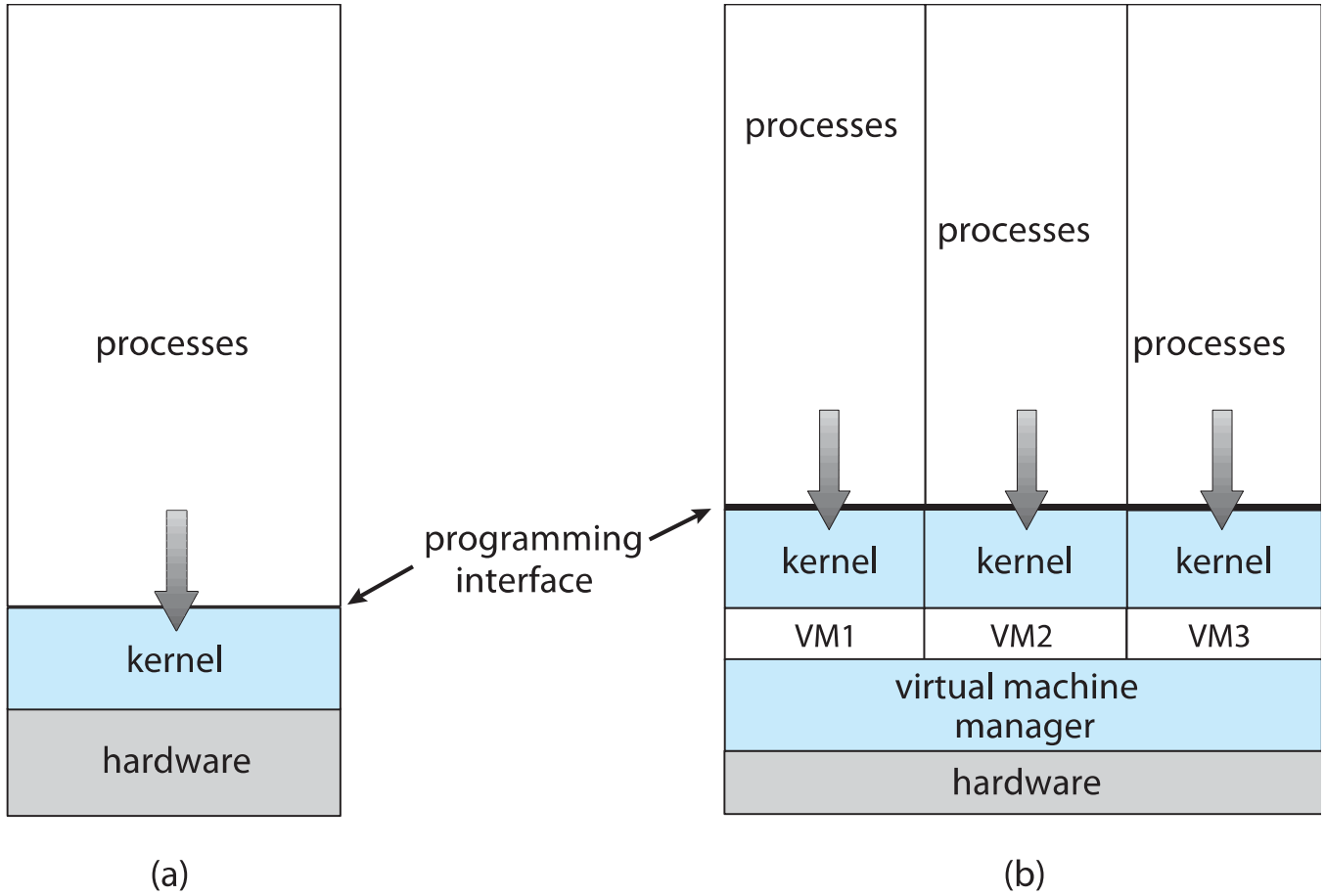
Virtualization

- Allows operating systems to run applications within other OSes
  - Vast and growing industry
- **Emulation** used when source CPU type different from target type (i.e. PowerPC to Intel x86)
  - Generally slowest method
  - When computer language not compiled to native code – **Interpretation**
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
  - Consider VMware running WinXP guests, each running applications, all on native WinXP **host** OS
  - **VMM** (virtual machine Manager) provides virtualization services

Virtualization (cont.)

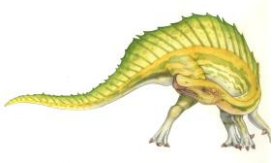
- Use cases involve laptops and desktops running multiple OSes for exploration or compatibility
  - Apple laptop running Mac OS X host, Windows as a guest
  - Developing apps for multiple OSes without having multiple systems
  - Quality assurance testing applications without having multiple systems
  - Executing and managing compute environments within data centers
- VMM can run natively, in which case they are also the host
  - There is no general-purpose host then (VMware ESX and Citrix XenServer)

Computing Environments - Virtualization

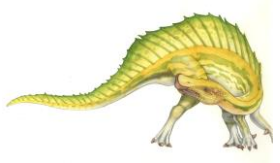


Distributed Systems

- Collection of separate, possibly heterogeneous, systems networked together
  - **Network** is a communications path, **TCP/IP** most common
    - **Local Area Network** (**LAN**) for small areas like homes or offices
    - **Wide Area Network** (**WAN**) connects large areas, like cities or countries, often across long distances
    - **Metropolitan Area Network** (**MAN**) connects a city or a large campus
    - **Personal Area Network** (**PAN**) connects personal devices within a very short range
- **Network Operating System** provides features between systems across network
  - Communication scheme allows systems to exchange messages
  - Illusion of a single system



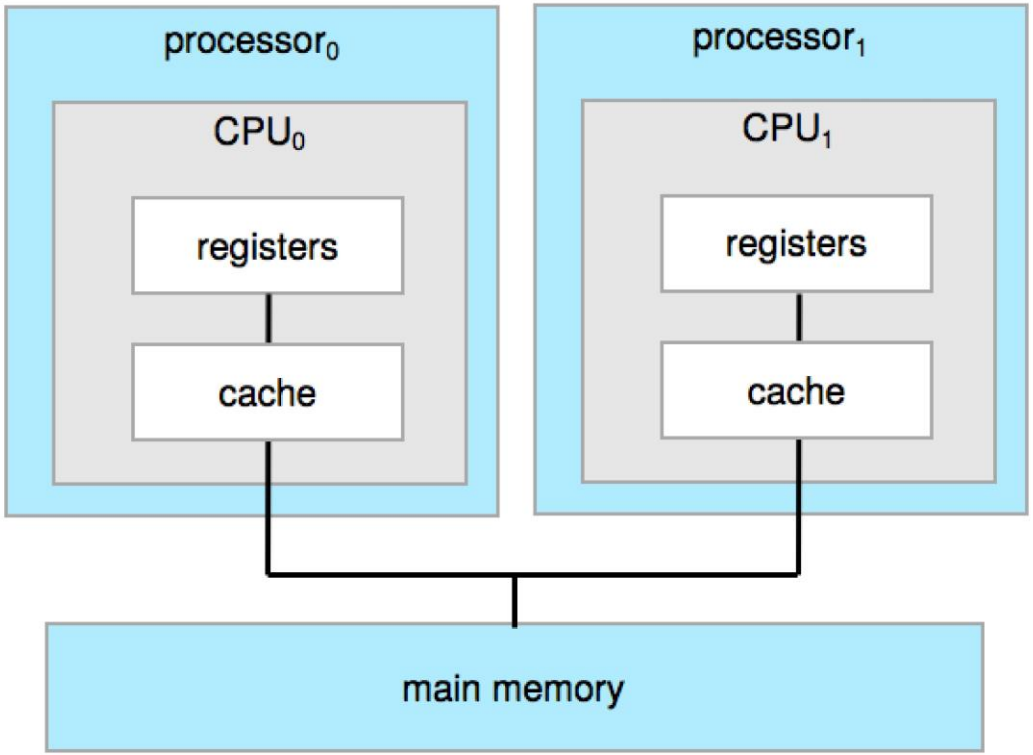
Computer System Architecture

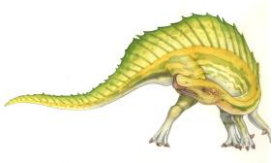


Computer-System Architecture

- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance
  - Two types:
    1. **Asymmetric Multiprocessing** – each processor is assigned a specie task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks

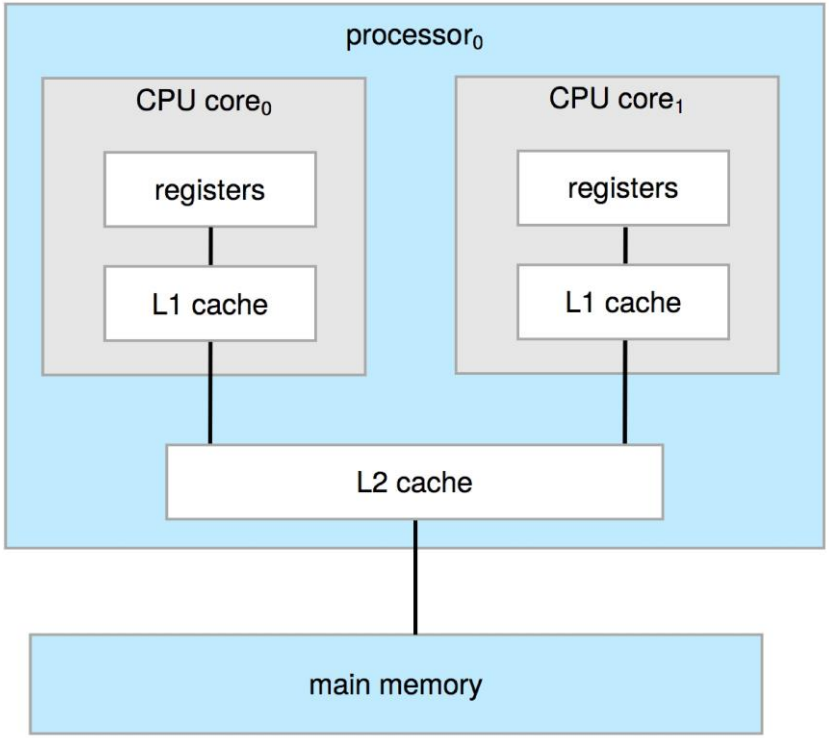
Symmetric Multiprocessing Architecture





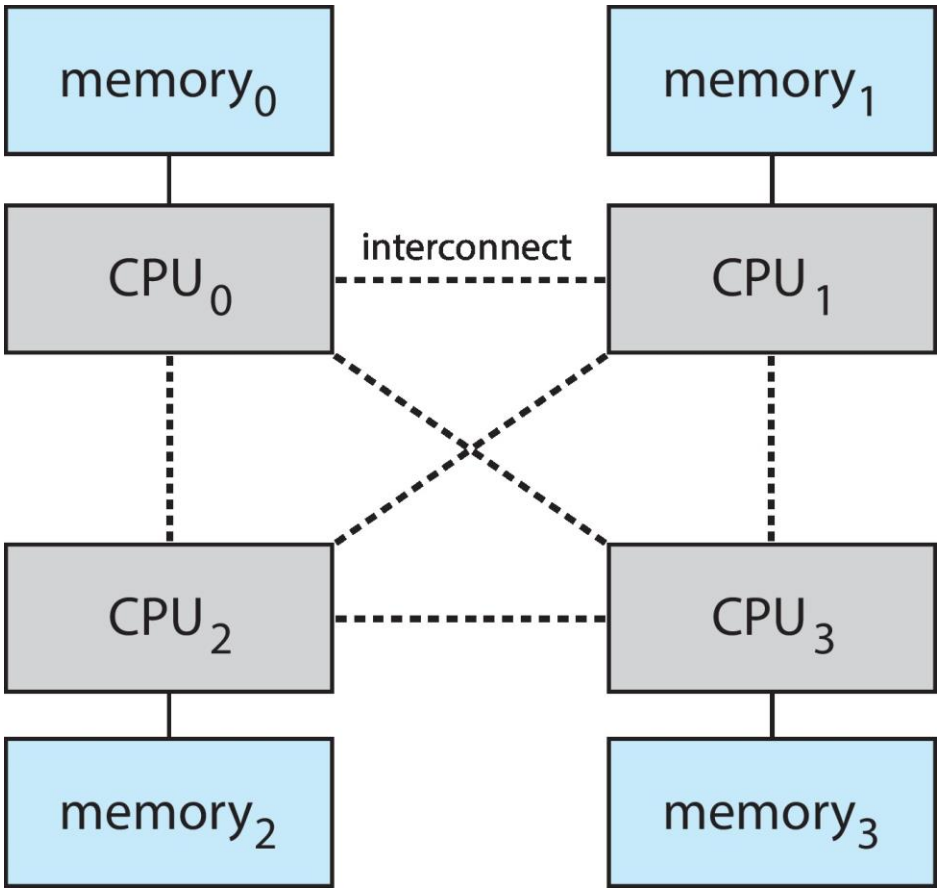
Dual-Core Design

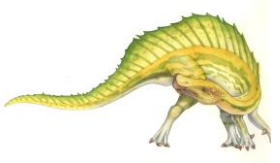
- Multi-chip and **multicore**
- Systems containing all chips
  - Chassis containing multiple separate systems



Non-Uniform Memory Access System

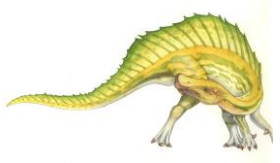
where the time it takes to access memory varies depending on the memory location relative to the processor.



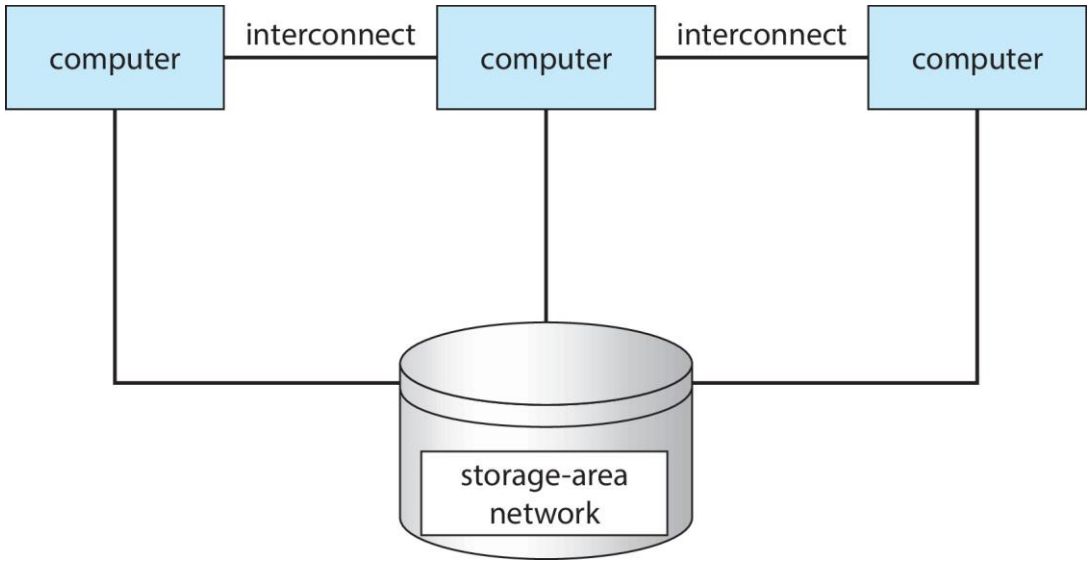


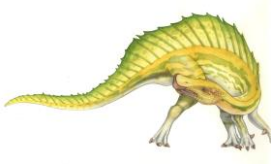
Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network** (**SAN**)
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other
  - Some clusters are for **high-performance computing** (**HPC**)
    - Applications must be written to use **parallelization**
  - Some have **distributed lock manager** (**DLM**) to avoid conflicting operations



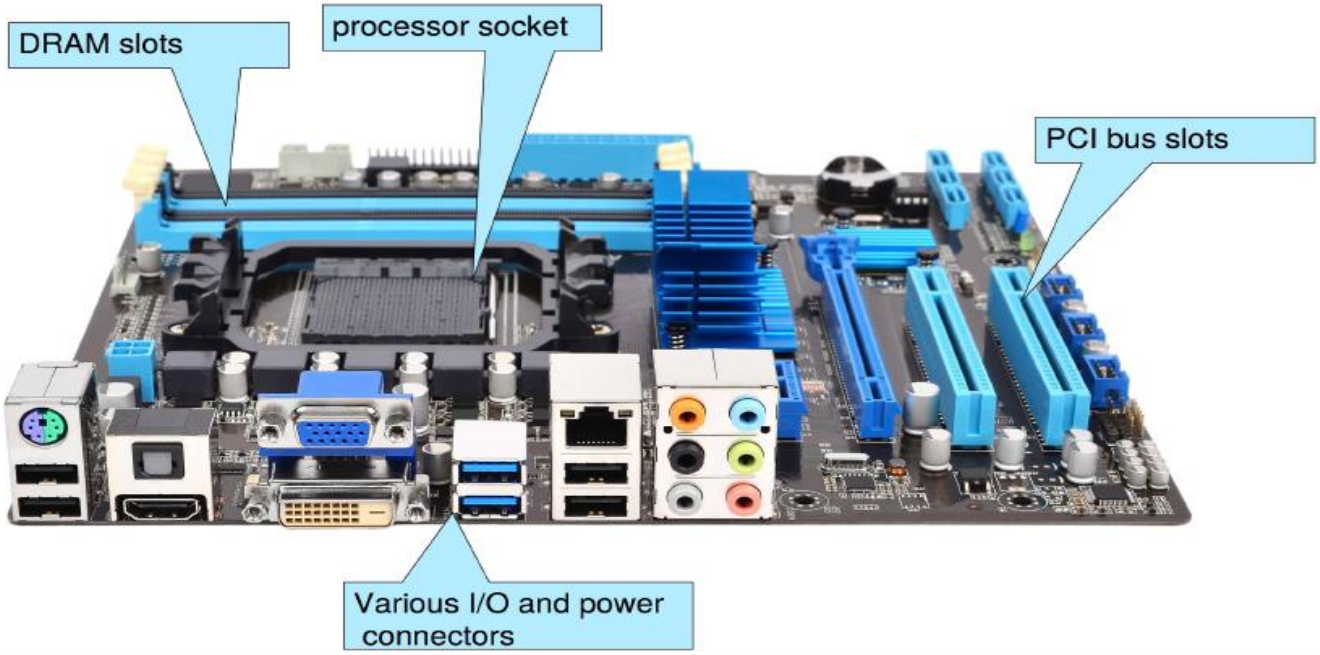
Clustered Systems





PC Motherboard

Consider the desktop PC motherboard with a processor socket shown below:

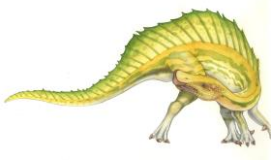


This board is a fully-functioning computer, once its slots are populated. It consists of a processor socket containing a CPU, DRAM sockets, PCIe bus slots, and I/O connectors of various types. Even the lowest-cost general-purpose CPU contains multiple cores. Some motherboards contain multiple processor sockets. More advanced computers allow more than one system board, creating NUMA systems.

Computer System Environments

Computing Environments

- Traditional
- Mobile
- Client Server
- Peer-to-Peer
- Cloud computing
- Real-time Embedded



Traditional

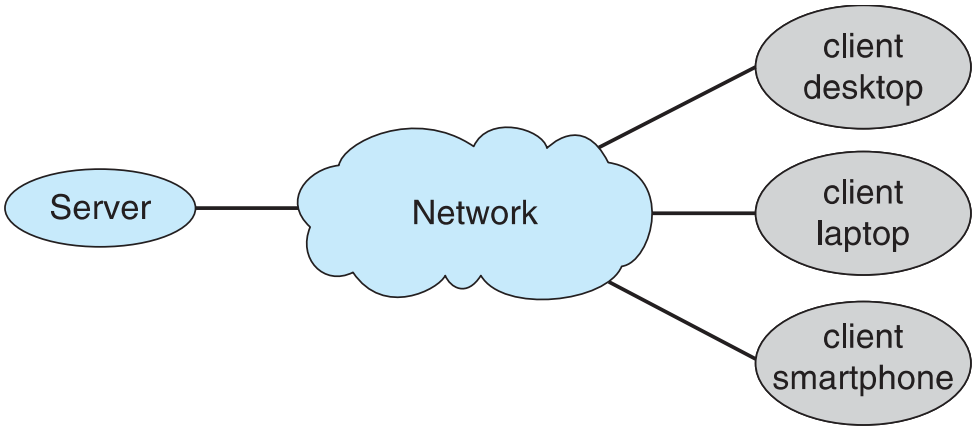
- Stand-alone general-purpose machines
- But blurred as most systems interconnect with others (i.e., the Internet)
- **Portals** provide web access to internal systems
- **Network computers** (**thin clients**) are like Web terminals
- Mobile computers interconnect via **wireless networks**
- Networking becoming universal – even home systems use **firewalls** to protect home computers from Internet attacks

Mobile

- Handheld smartphones, tablets, etc.
- What is the functional difference between them and a “traditional” laptop?
- Extra feature – more OS features (GPS, gyroscope)
- Allows new types of apps like ***augmented reality***
- Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

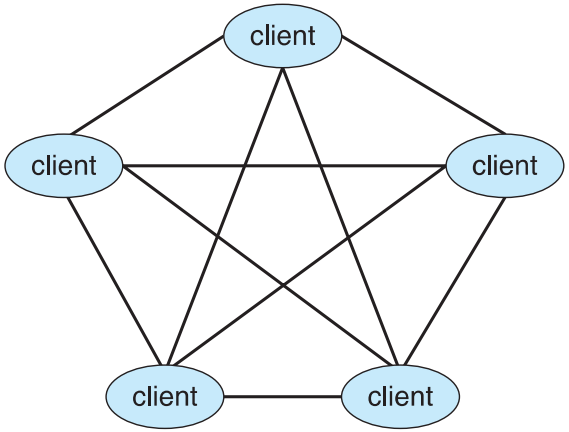
Client Server

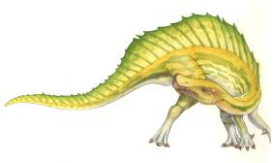
- Client-Server Computing
  - Dumb terminals supplanted by smart PCs
  - Many systems now **servers**, responding to requests generated by **clients**
    - **Compute-server system** provides an interface to client to request services (i.e., database)
    - **File-server system** provides interface for clients to store and retrieve files



Peer-to-Peer

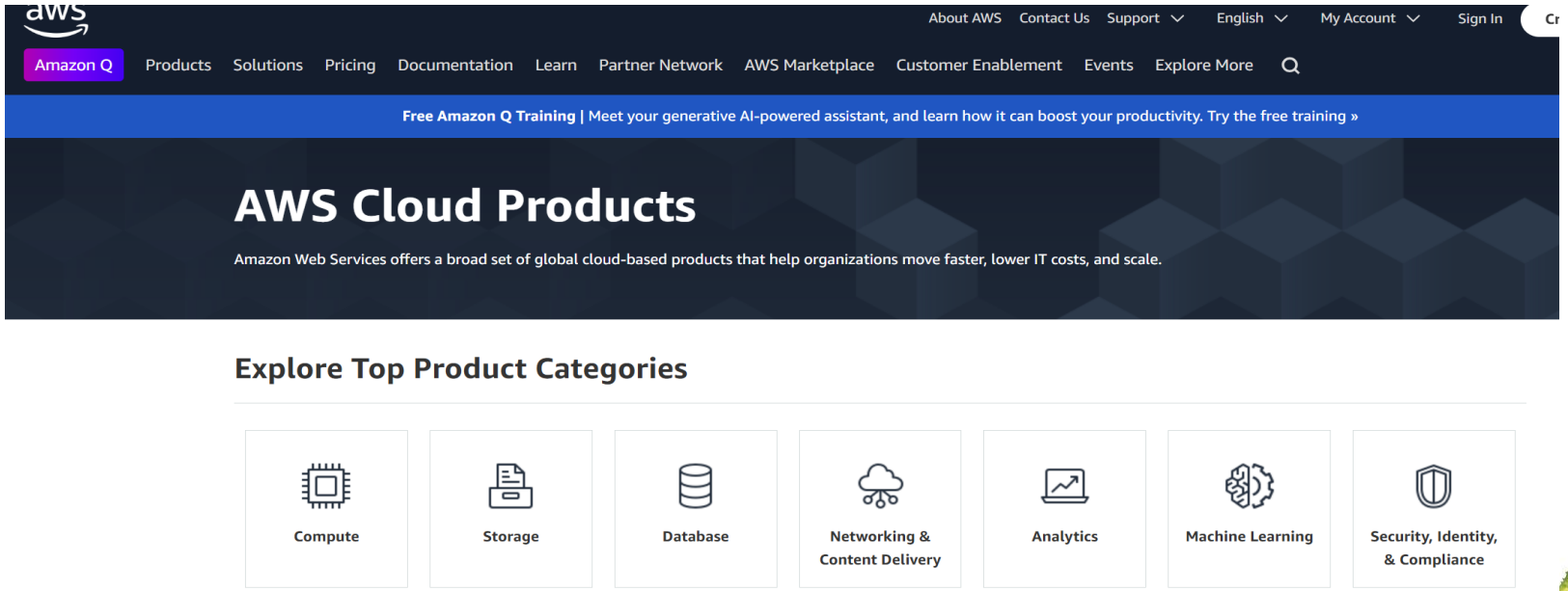
- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via ***discovery protocol***
  - Examples include Napster and Gnutella, **Voice over IP** (**VoIP**) such as Skype





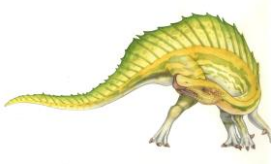
Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage



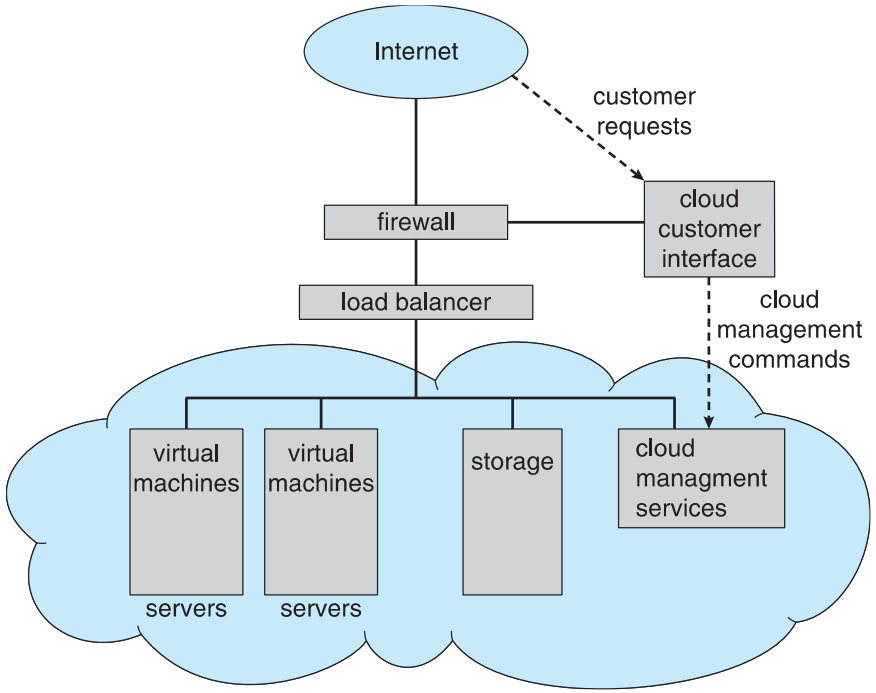
Cloud Computing (Cont.)

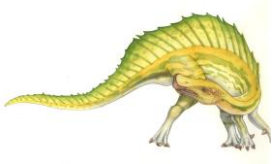
- Many types
  - **Public cloud** – available via Internet to anyone willing to pay
  - **Private cloud** – run by a company for the company's own use
  - **Hybrid cloud** – includes both public and private cloud components
  - Software as a Service (**SaaS**) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (**PaaS**) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (**IaaS**) – servers or storage available over Internet (i.e., storage available for backup use)



Cloud Computing (cont.)

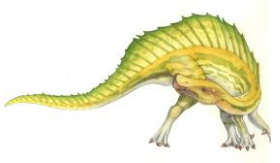
- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications





Real-Time Embedded Systems

- Real-time embedded systems most prevalent form of computers
  - Vary considerable, special purpose, limited purpose OS, **real-time OS**
  - Use expanding
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met

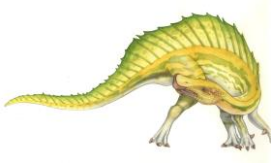


Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source** and **proprietary**
- Counter to the **copy protection** and **Digital Rights Management** (**DRM**) movement
- Started by **Free Software Foundation** (**FSF**), which has "copyleft" **GNU Public License** (**GPL**)
  - Free software and open-source software are two different ideas championed by different groups of people
    - **https://www.gnu.org/philosophy/open-source-misses-the-point.en.html**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run guest operating systems for exploration

The Study of Operating Systems

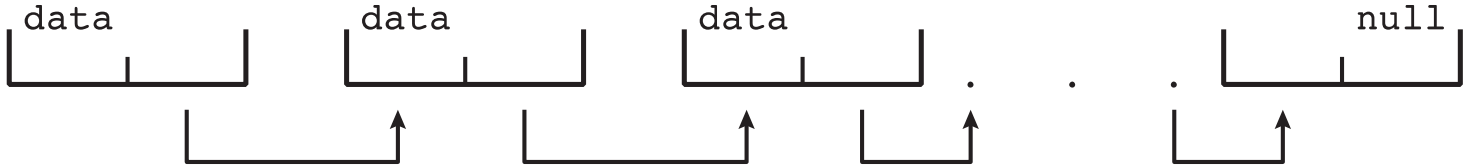
- **Studying Operating Systems is More Accessible**:
  - Open-source operating systems are available in both source code and executable formats.
  - Examples include Linux, BSD UNIX, Solaris, and parts of macOS.
- **Source Code Availability**:
  - You can now examine and learn from the actual source code of operating systems. This allows a deeper understanding beyond just documentation and behavior.
    https://curlie.org/Computers/Software/Operating_Systems/Open_Source/
- **Study Older Systems**:
  - Some outdated operating systems are open-sourced, helping us understand how they worked with limited resources.
- **Virtualization**:
  - Virtualization tools like VMware and VirtualBox allow running multiple operating systems on a single core system.
  - Students can experiment with many operating systems without needing extra hardware.
    http://www.virtualbox.com
- **Creating New Distributions**:
  - With access to open-source code, students can create and customize new operating system distributions.
  - Previously, access to source code was limited, but now it is more open with just basic resources.



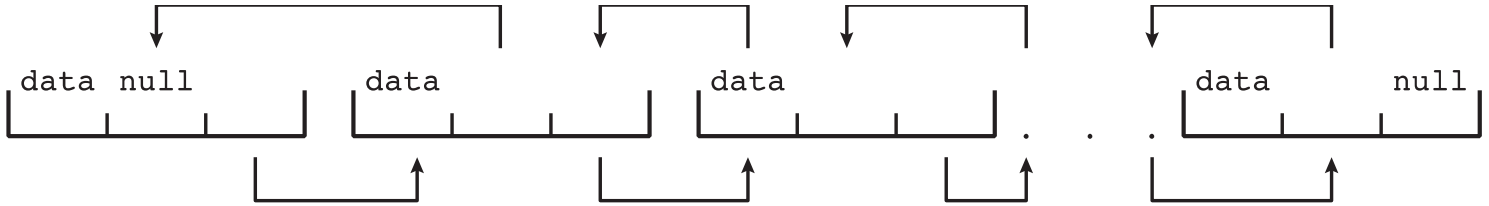
Kernel Data Structure

Kernel Data Structures

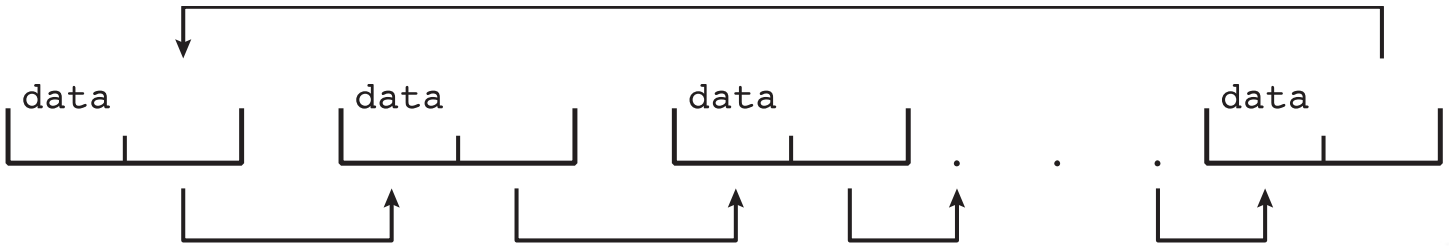
- Many similar to standard programming data structures
- ***Singly linked list***



- ***Doubly linked list***

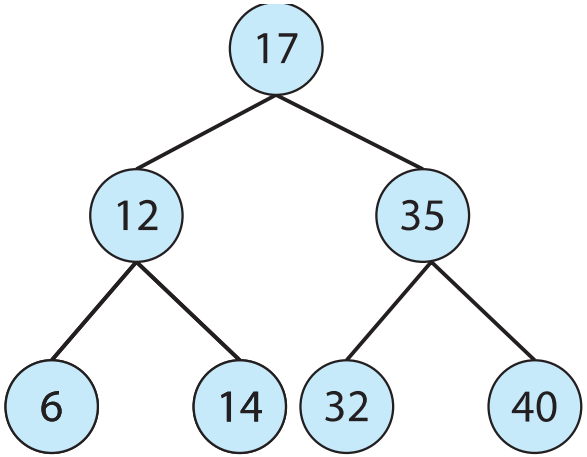


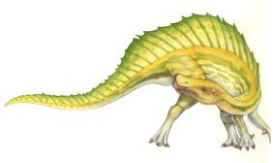
- ***Circular linked list***



Kernel Data Structures

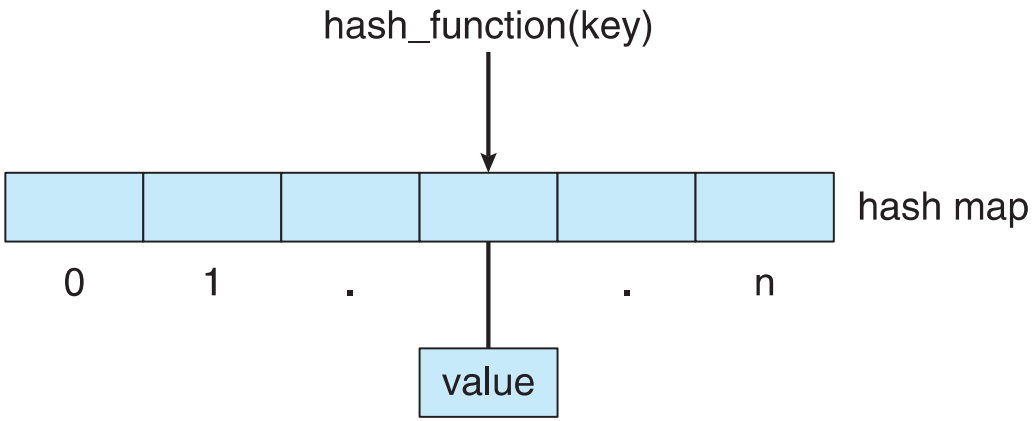
- **Binary search tree**
  left <= right
  - Search performance is *O(n)*
  - **Balanced binary search tree** is *O(lg n)*





Kernel Data Structures

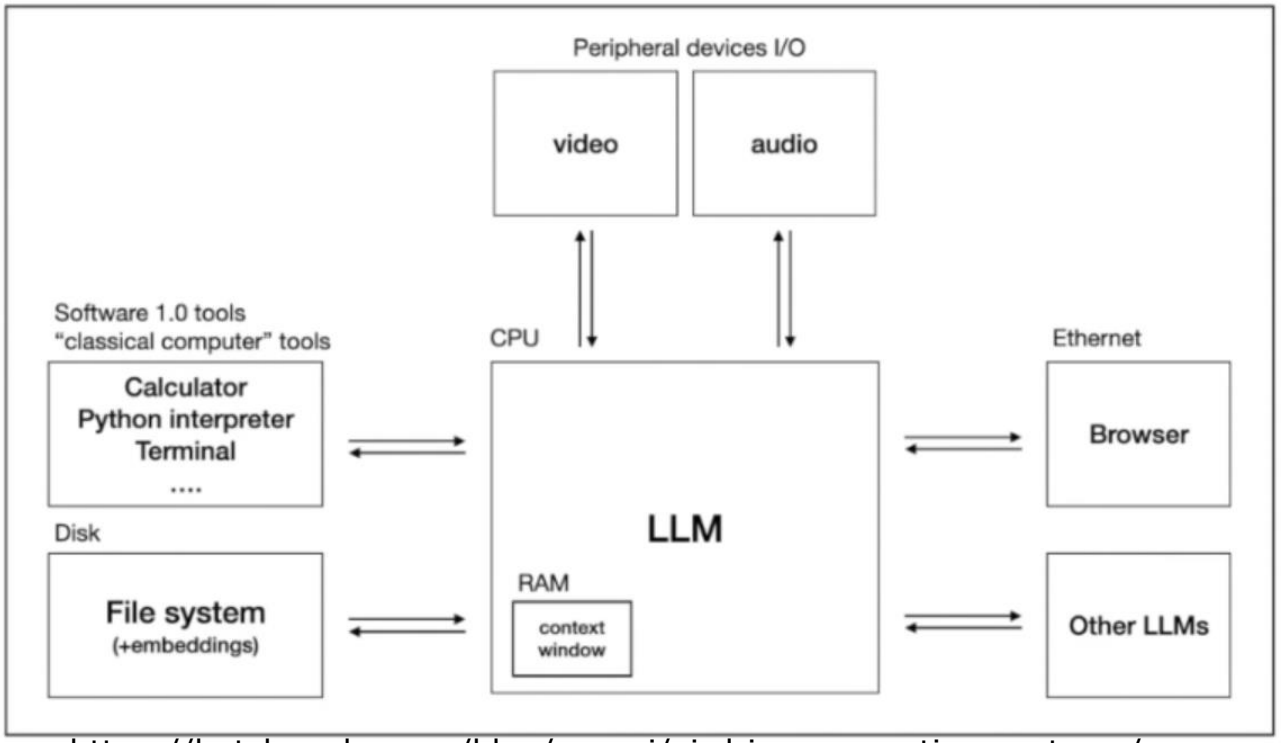
- **Hash function** can create a **hash map**



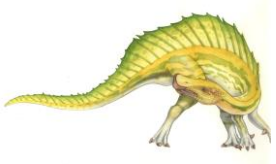
- **Bitmap** – string of *n* binary digits representing the status of *n* items
- Linux data structures defined in ***include*** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

AI: The Next Operating System?

- The integration of AI and Operating Systems is a key area for innovation.
- AI is used to optimize OS functionalities like memory management, process scheduling, and intrusion detection
- For example, an AI OS can anticipate a user's needs, predicting the right apps to suggest at different times of the day or adjusting settings based on the environment for optimal comfort.



https://hatchworks.com/blog/gen-ai/ai-driven-operating-systems/



End of Chapter 1