TUNIS
SOUTH MEDITERRANEAN
UNIVERSITY

Chapter 1: Introduction

Presented by: **Dr. Ikram Saadaoui**

Overview of Computer System Structure

Computer System Structure

The Bootstrap

- A computer needs to have an initial program to start running when it is powered up or rebooted.

bootstrap program

-Stored in ROM or firmware (like BIOS/UEFI), the boot process begins by detecting hardware devices and choosing a boot device (e.g., hard drive, SSD, or USB).

-The bootloader then locates the operating system kernel and loads it into memory.

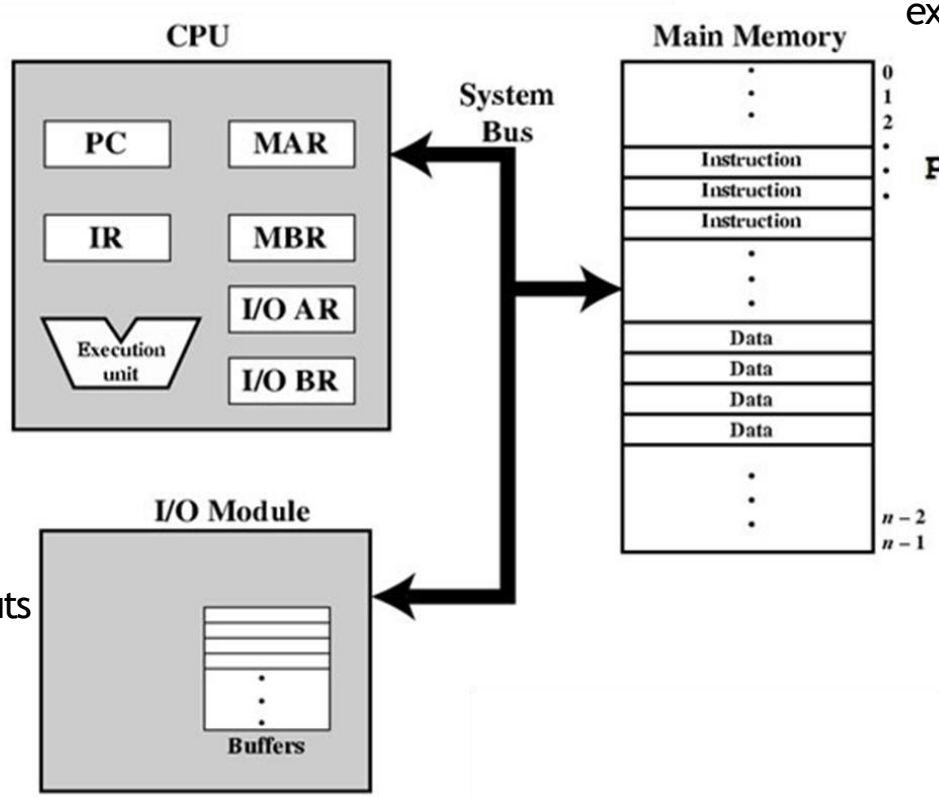
-Once the kernel is loaded and executed, it initializes the system. This includes setting up CPU registers, configuring device controllers, and initializing memory contents.

-After initialization, the operating system displays a login screen or desktop interface. At this point, the computer is fully operational and ready for use.

Computer System Structure: Hardware

run =
CPU fetch/execute cycle runs through the instruction series

Charge =
Main Memory charge programs to be executed

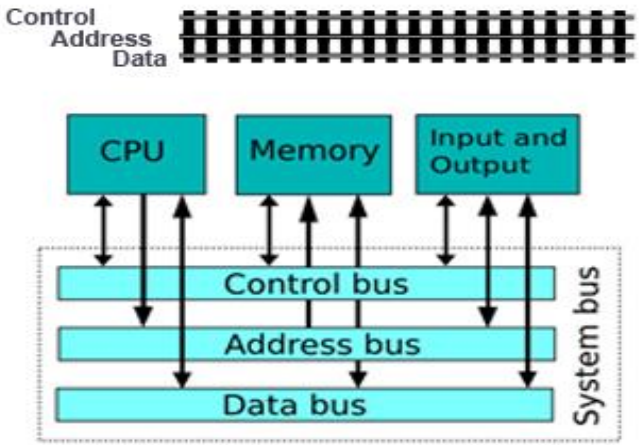


PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

program =
series of CPU instructions

I/O Module Inputs and outputs data to and from the CPU

Computer System Hardware: System Bus



- The system bus is a critical communication pathway (essentially a collection of wires and protocols) that connects various components of a computer, allowing them to transfer data, signals, or power among each other.
- It facilitates the communication between the CPU, memory, and I/O devices.
- Components of a bus system:
  - **Data Bus:** Carries the data that needs processing
  - **Address Bus:** Determines where data should be sent
  - **Control Bus:** Carries control signals that manage and coordinate the various operations of the computer. These signals include read/write data commands, interrupt requests, clock signals, and status signals.

Computer System Hardware: CPU

- CPU refers to the central processing unit
- It performs basic arithmetic, logic, controlling, and input/output (I/O) operations specified by the instructions in the program.

A typical instruction-execution cycle (Fetch-Decode-Execute):

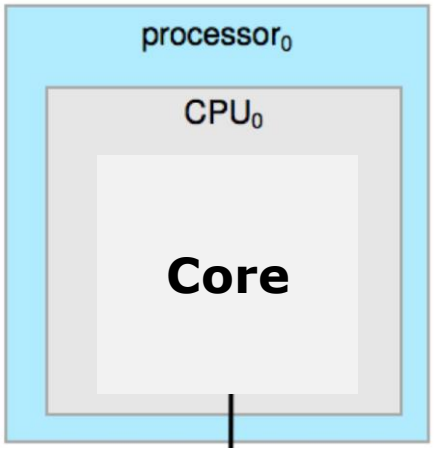
- Fetches an instruction from memory using the address in the Program counter (PC) and stores that instruction in the instruction register (IR).
- The Control Unit decodes (interprets) the fetched instruction and determines the required actions.
- The ALU performs the required operations, and results are stored in registers (like the accumulator) or memory.
- The PC is updated to point to the next instruction.

Computer System Hardware: CPU

DEFINITIONS OF COMPUTER SYSTEM COMPONENTS

- CPU—The hardware that executes instructions.
- Processor—A physical chip that contains one or more CPUs.
- Core—The basic computation unit of the CPU.
- Multicore—Including multiple computing cores on the same CPU.
- Multiprocessor—Including multiple processors.

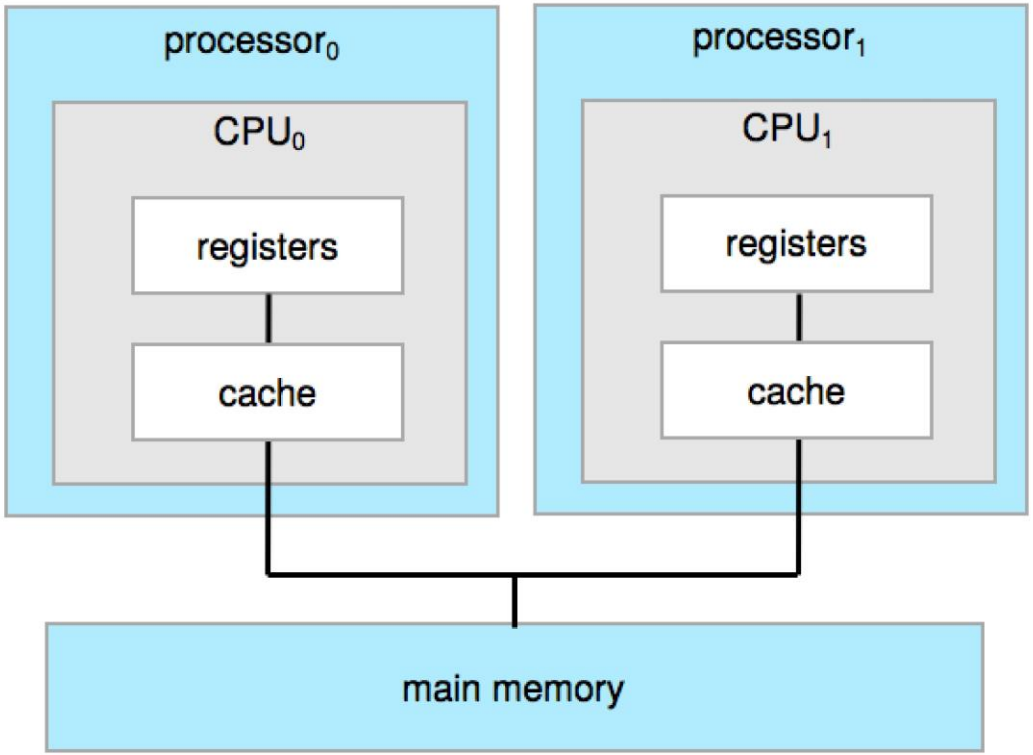
Although virtually all systems are now multicore, we use the general term ***CPU*** when referring to a single computational unit of a computer system and ***core*** as well as ***multicore*** when specifically referring to one or more cores on a CPU.



Computer-System Architecture

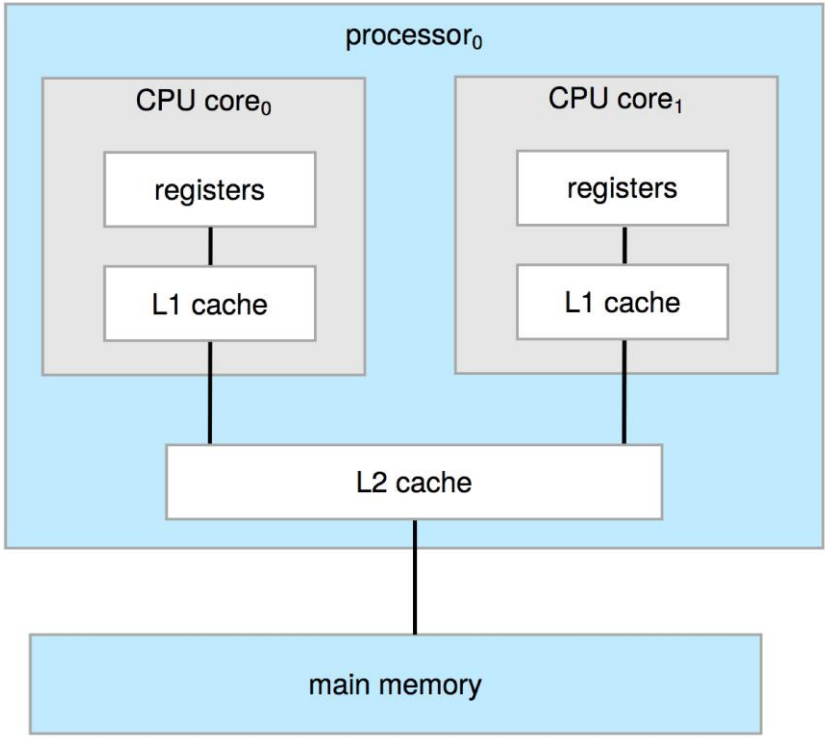
- Most systems use a single general-purpose processor
  - Most systems have special-purpose processors as well
  - Used a single processor containing one CPU with a single processing core
- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems**, **tightly-coupled 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 specific task.
    2. **Symmetric Multiprocessing** – each processor performs all tasks

Symmetric Multiprocessing Architecture



Dual-Core Design

- Multi-chip and **multicore**



Storage Structure

Computer System Hardware: Memory

- **Main memory** – only large storage media that the CPU can access directly
  - **Random access**
  - Typically, **volatile**

Main memory

Computer System Hardware: Memory

- **Secondary storage** – an extension of main memory that provides large **nonvolatile** storage capacity
- **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**
- **Non-volatile memory** (**NVM**) devices– faster than hard disks, nonvolatile
  - Various technologies (Solid State Drive: SSD)
  - Becoming more popular as capacity and performance increases, price drops

HDD

SSD

Caching

- Caching is a fundamental principle used at various levels in a computer system (hardware, OS, software) to improve performance. It involves temporarily storing frequently accessed data in a smaller, faster storage area called a cache.
  - Think of it like this: Imagine you're working at your desk. Instead of going to the library every time you need a document, you keep the documents you're using most often in a stack right on your desk. This "desk stack" is your cache.

Caching

- Here's how it works:

1. Data Copying: When data is needed, the system first checks the cache.
2. Cache Hit: If present, the data is retrieved from the fast cache.
3. Cache Miss: If absent, the data is retrieved from slower main storage, copied to the cache, then used.

- Cache Size Constraint: Caches are smaller than the storage they cache.
- Cache Management: Efficient cache management (size and replacement policy) is crucial for performance.

Storage Definitions and Notation Review (OPTIONAL)

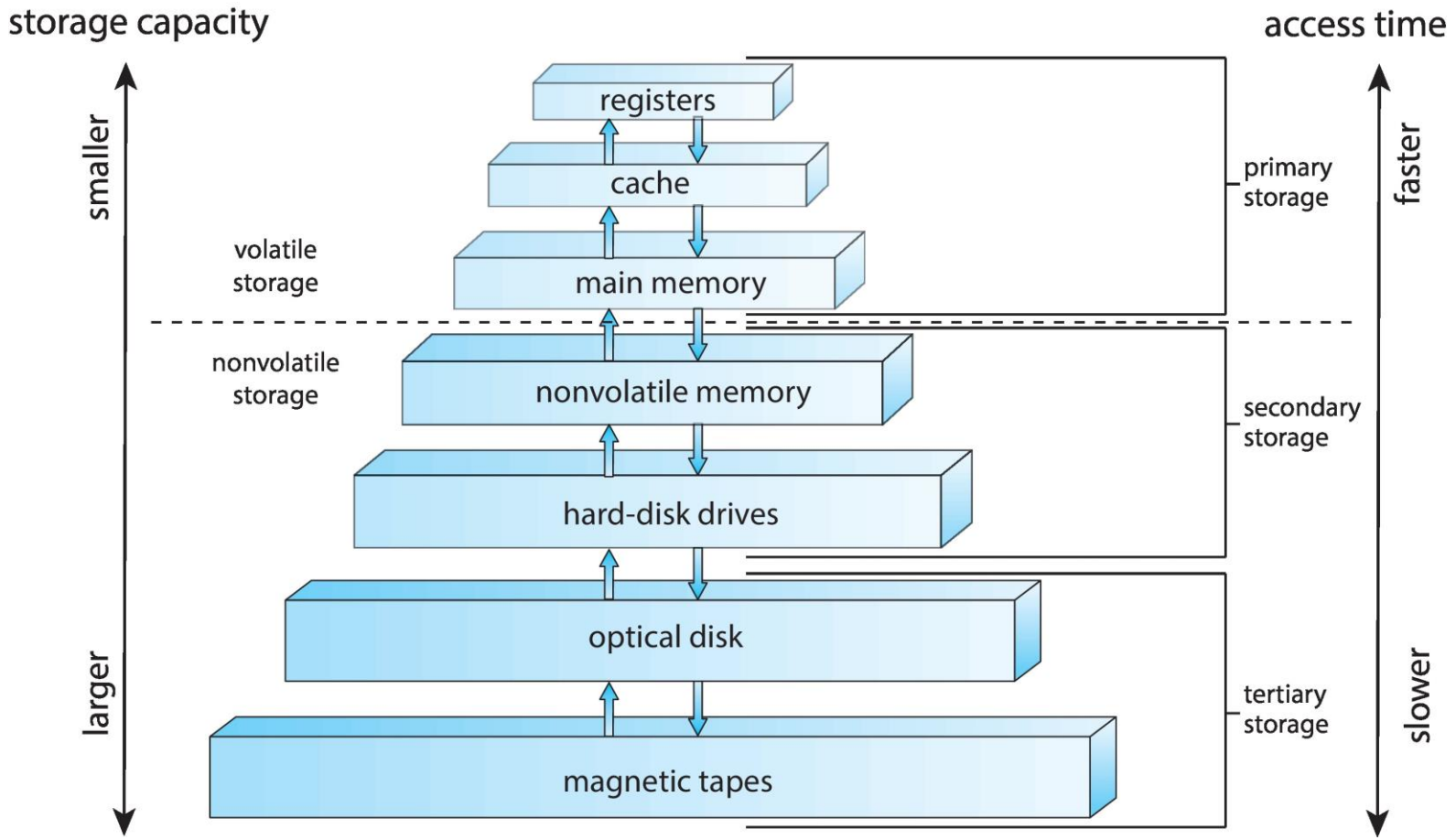
The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, 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; and 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 Structure: Memory

- Storage systems organized in a hierarchy
  - Speed
  - Cost
  - Volatility

Storage-Device Hierarchy



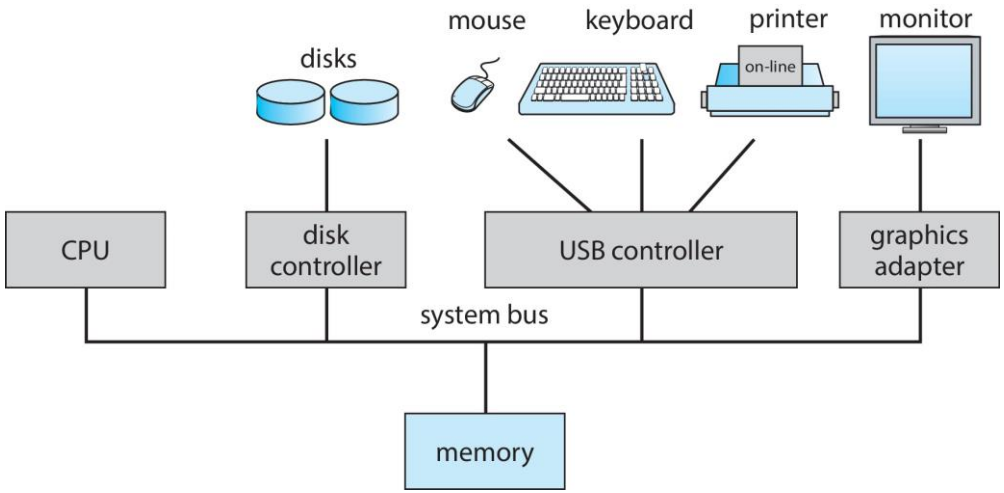
Characteristics of Various Types of Storage (OPTIONAL)

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

I/O devices

Computer System Structure: I/O devices



One or more CPUs and device controllers connect through a common **bus** providing access to shared memory

- **Device controllers** It's a small electronic part that knows how to speak the device's language
- The CPU sends commands, and the controller ensures the connected device understands them.

Example:
CPU: "Hey printer, print this!"
Device controller: "Alright, let me tell the printer to do that and get back to you."

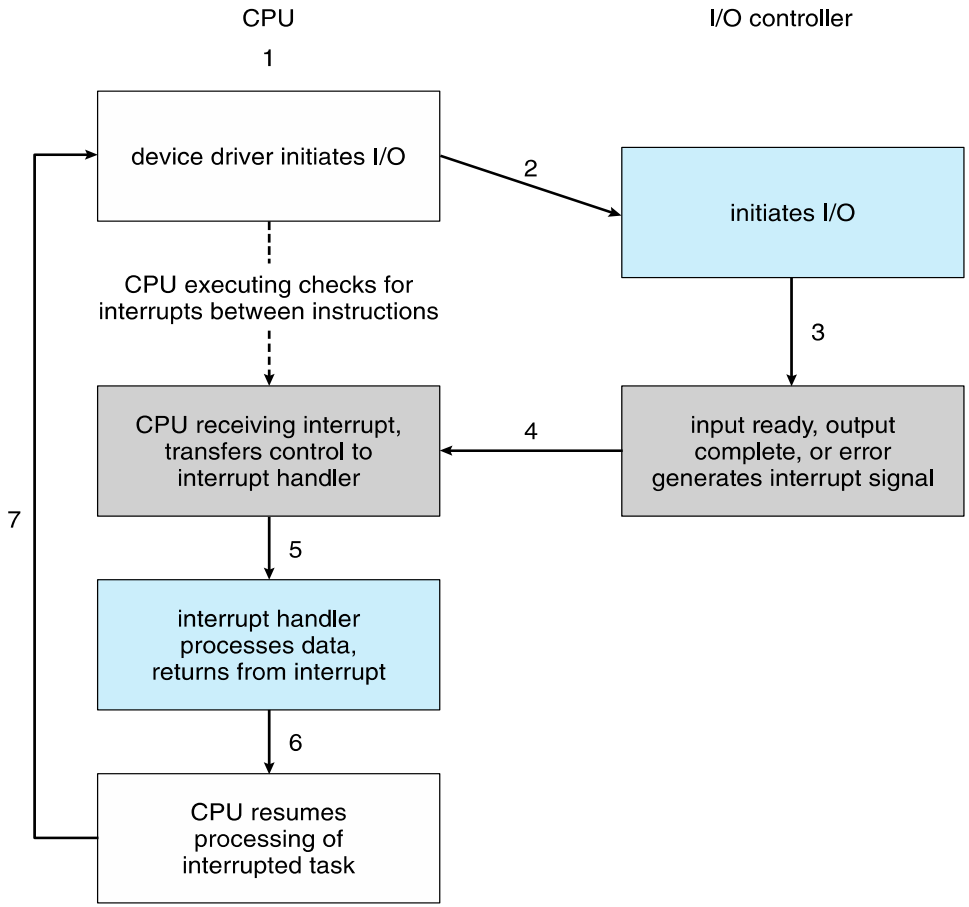
Computer System Structure: I/O devices

- The CPU and I/O devices can work simultaneously.
  - Think of it like this: the CPU is the main worker, and I/O devices (like your keyboard, mouse, or printer) are tools that help it.
- Each I/O device has its manager, called a **device controller**. This controller handles all the communication with its specific device.
  - For example, there's a controller for your keyboard, a different one for your printer, and so on.
- Every device controller has its temporary storage area called a **local buffer**.
  - Imagine this as a small waiting room where data is held temporarily.
- To make sure the OS understands how to use each device controller, the operating system provides a system software called a **device driver**.
  - This driver acts like a translator, teaching the OS how to communicate with and control each specific device controller.

Computer System Structure: I/O devices

Here's how data flows:

- The device driver initializes an I/O operation by loading appropriate registers in the device controller.
- The device controller checks the registers to decide the required action (e.g., read a character from the keyboard). The controller transfers data from the device to its local buffer.
- When the device controller completes its task (like reading data from the keyboard or printing a page), it sends an interrupt signal to the CPU. This signal informs the CPU that the controller has finished and is ready for more work.



Interrupt-drive I/O Cycle

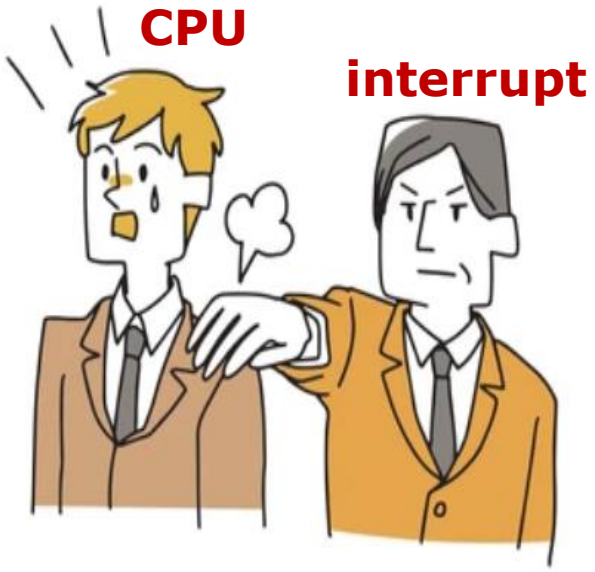
Operating Systems Operations:

Interruption Handling

The occurrence of an event is usually signaled by an interrupt from either the **hardware** or **the software**.

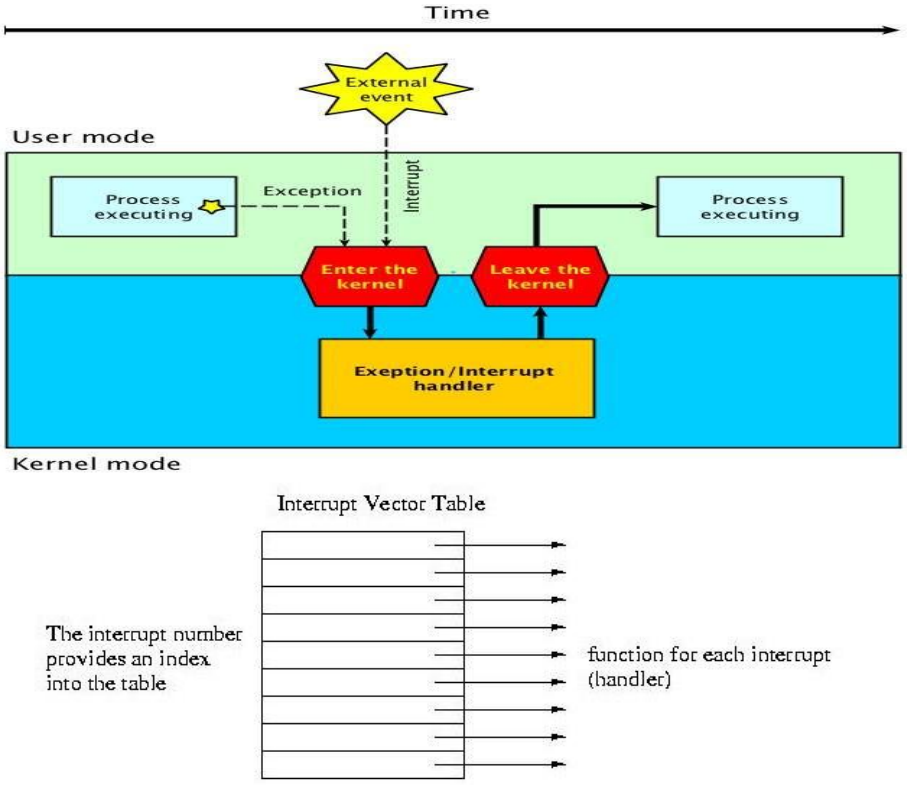
- Hardware may trigger an **interrupt** by sending a signal to the CPU, usually through the system bus.
- Software may trigger an interrupt either by an error **exception (or trap)** or by user request **System Call**.

An interruption is like a tap on the shoulder of the CPU, signaling it to pay attention to something important.



Operating System operations:
Interruption Handling

- When the CPU is interrupted,
  - It stops what it's doing and jumps to a special helper program called Interrupt Service Routine or (ISR) to deal with the interrupt.
  - The CPU uses an interrupt vector (a list of addresses) to find the right helper program for the task.
  - Before jumping, it saves where it stopped (the address of the interrupted instruction) so it can return and continue later.



Computer System Structure:
I/O devices and Memory

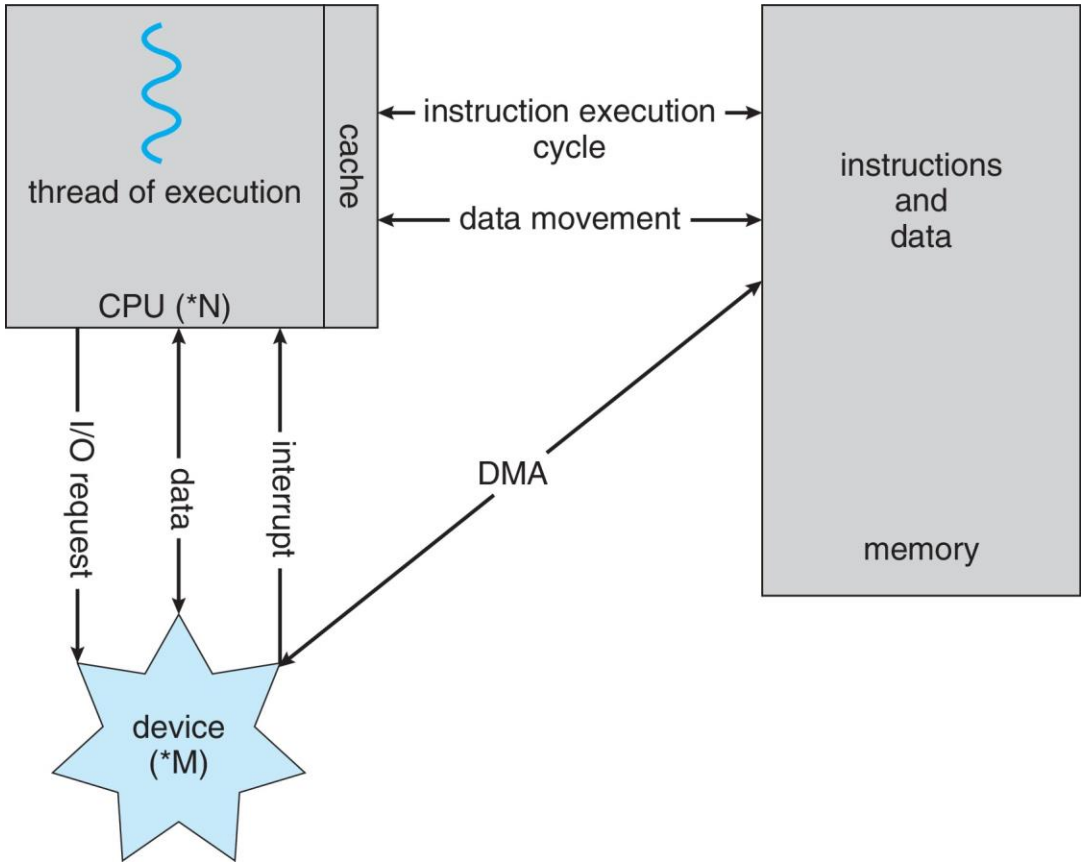
- Direct Memory Access (DMA) is a way to make data transfer faster and smarter by skipping the CPU for most of the work.

- **Normal I/O**:
  In regular data transfer, the CPU handles every little step, like moving each byte of data between the device and memory.
  - the CPU gets interrupted **for** every byte of data transferred. This takes a lot of time and effort for the CPU.

- **DMA**:
  With DMA, the **device controller** can directly move large chunks of data from the device (like a hard drive) to the computer's memory **without involving the CPU**.

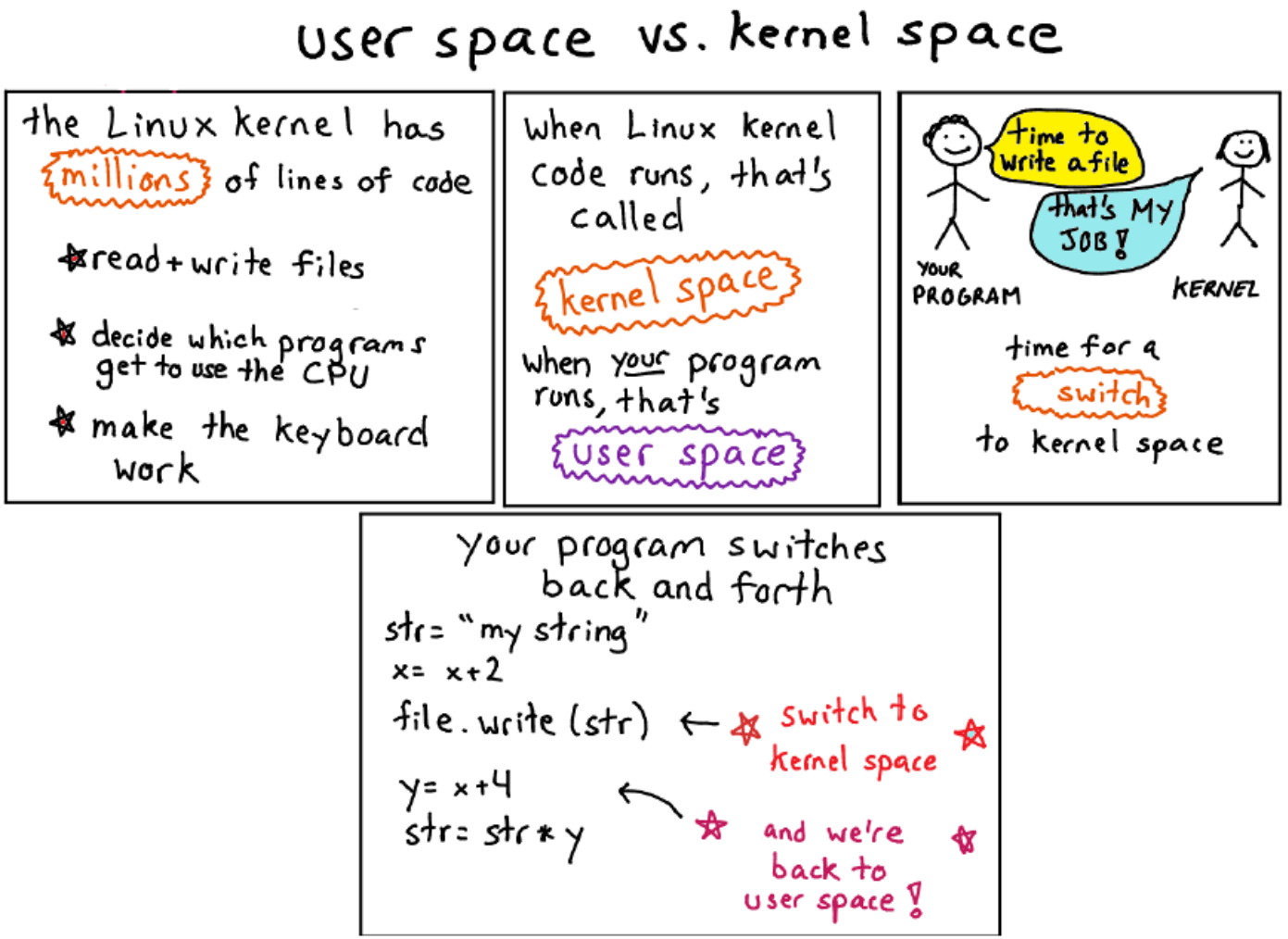
Analogy Imagine you want to fill a bucket with water:
Without DMA: You (CPU) fill the bucket one cup at a time (slow & exhausting).
With DMA: A hose (device controller) fills the bucket all at once, and the CPU only checks when it's done!

How a Modern Computer Works



A von Neumann architecture

Dual-mode Operation

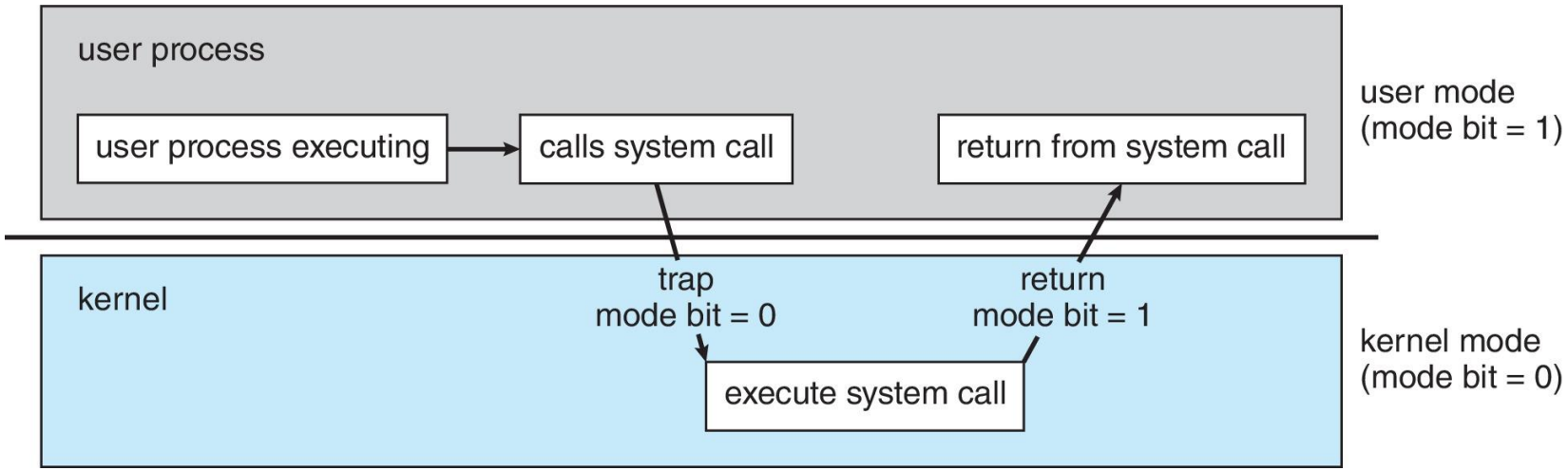


Dual-mode Operation

- **Dual mode** is a feature in computer systems that provides two distinct operating modes:
  - **User Mode:** A non-privileged mode of operation, when executing on behalf of a user (i.e. application programs) with restricted access to system resources.
  - **Kernel Mode:** A privileged mode of operation, when executing on behalf of the OS with full access to hardware and system resources.

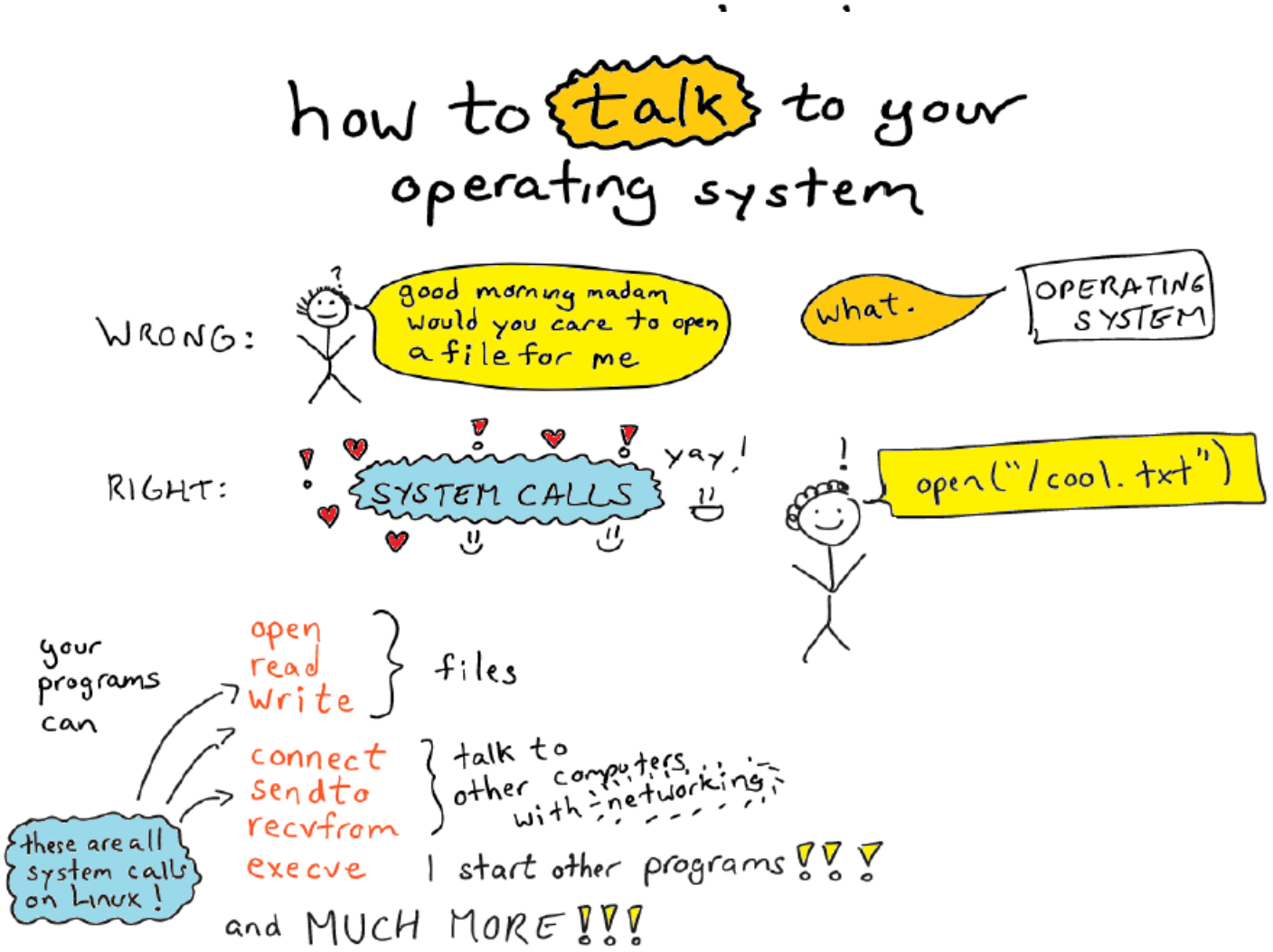
- Mode is indicated by a status bit in a protected CPU control register: kernel(0) or user(1).
- This helps to protect the operating system and the system's resources from being accessed or modified by applications in an unauthorized or malicious manner.
- When a user application requests a service from the operating system (via a system call), the system must transition from user to kernel mode to fulfill the request.

Transition from User to Kernel Mode



Operating-System Services

How to talk to OS: System Call



System Calls

- The **OS** provides many services, like managing files, running programs, and controlling hardware.
- Programs need a way to **talk to the OS** to use these services without worrying about low-level system details.
  - **For example,** in C language you should not care how scanf works

- This is done through an **Application Programming Interface (API)**.
- An **API** is like a translator between a program and the OS, typically written in a high-level language (C or C++) that translates program requests into **system calls**, which communicate directly with the OS.

- **Example:**

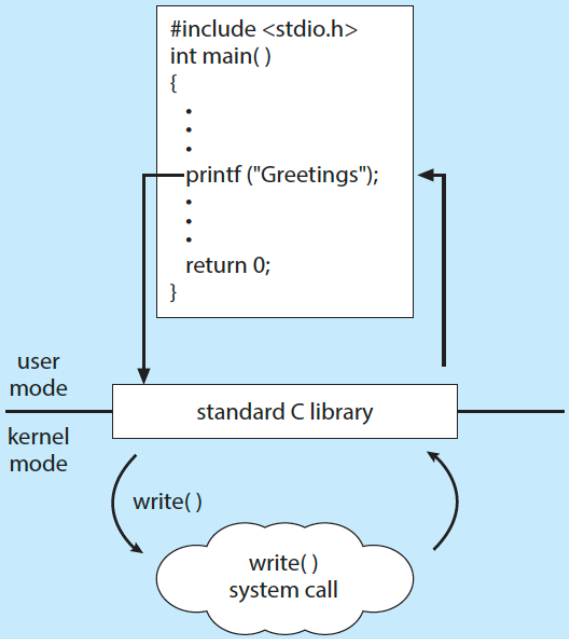
  Imagine you want to save a file in a program:
  - The program uses an API to ask the OS to save the file.
  - The API translates the request into a system call.
  - The OS handles the file-saving operation and returns a success message.

Standard C Library Example

- C program invoking printf() library call, which calls write() system call

THE STANDARD C LIBRARY

The standard C library provides a portion of the system-call interface for many versions of UNIX and Linux. As an example, let's assume a C program invokes the `printf()` statement. The C library intercepts this call and invokes the necessary system call (or calls) in the operating system—in this instance, the `write()` system call. The C library takes the value returned by `write()` and passes it back to the user program:



Examples of Windows and Unix System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	Windows	Unix

Process control	`CreateProcess()`
	`ExitProcess()`
	`WaitForSingleObject()`
		`fork()`
		`exit()`
		`wait()`

File management	`CreateFile()`
	`ReadFile()`
	`WriteFile()`
	`CloseHandle()`
		`open()`
		`read()`
		`write()`
		`close()`

Device management	`SetConsoleMode()`
	`ReadConsole()`
	`WriteConsole()`
		`ioctl()`
		`read()`
		`write()`

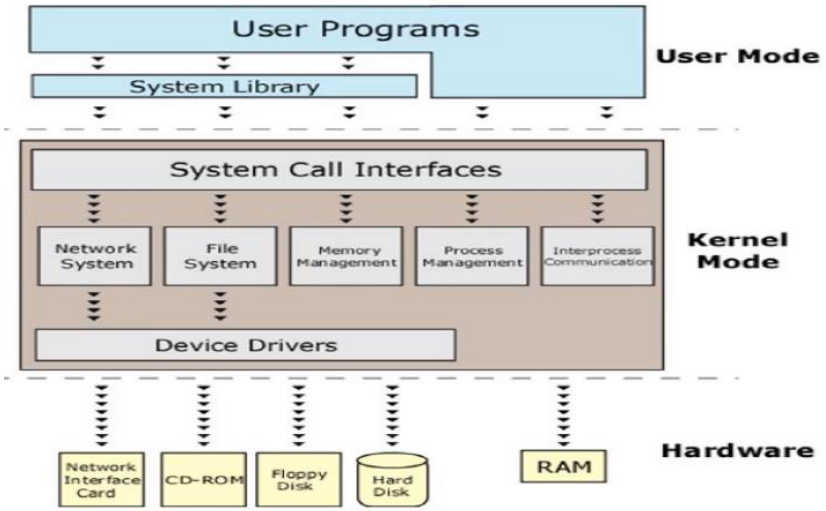
Information maintenance	`GetCurrentProcessID()`
	`SetTimer()`
	`Sleep()`
		`getpid()`
		`alarm()`
		`sleep()`

Communications	`CreatePipe()`
	`CreateFileMapping()`
	`MapViewOfFile()`
		`pipe()`
		`shm_open()`
		`mmap()`

Protection	`SetFileSecurity()`
	`InitlializeSecurityDescriptor()`
	`SetSecurityDescriptorGroup()`
		`chmod()`
		`umask()`
		`chown()`

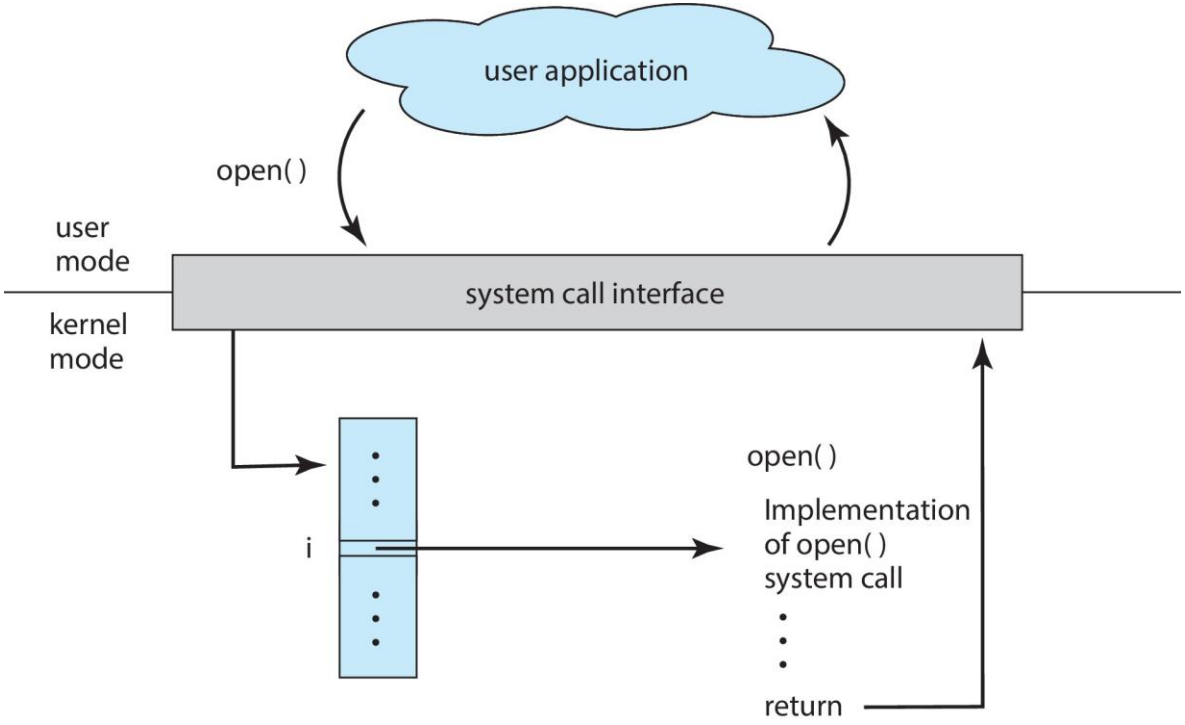
System Call

- A **system call** is how a program asks the operating system (OS) to perform a specific task, like open(), read(), write(), for(), wait(), exec(),clone()...etc.
- The **system call interface** manages this process.



- The OS keeps a table where each number points to the function that performs the system call in the OS kernel.
- Each system call is given a unique number that identifies the system call in the OS.

API – System Call – OS Relationship



Process Flow:

- A program makes a request using the API.
- The system call interface matches the request to the right system call using the table.
- The system call runs in the kernel, does the requested task, and sends back the result (status or data).

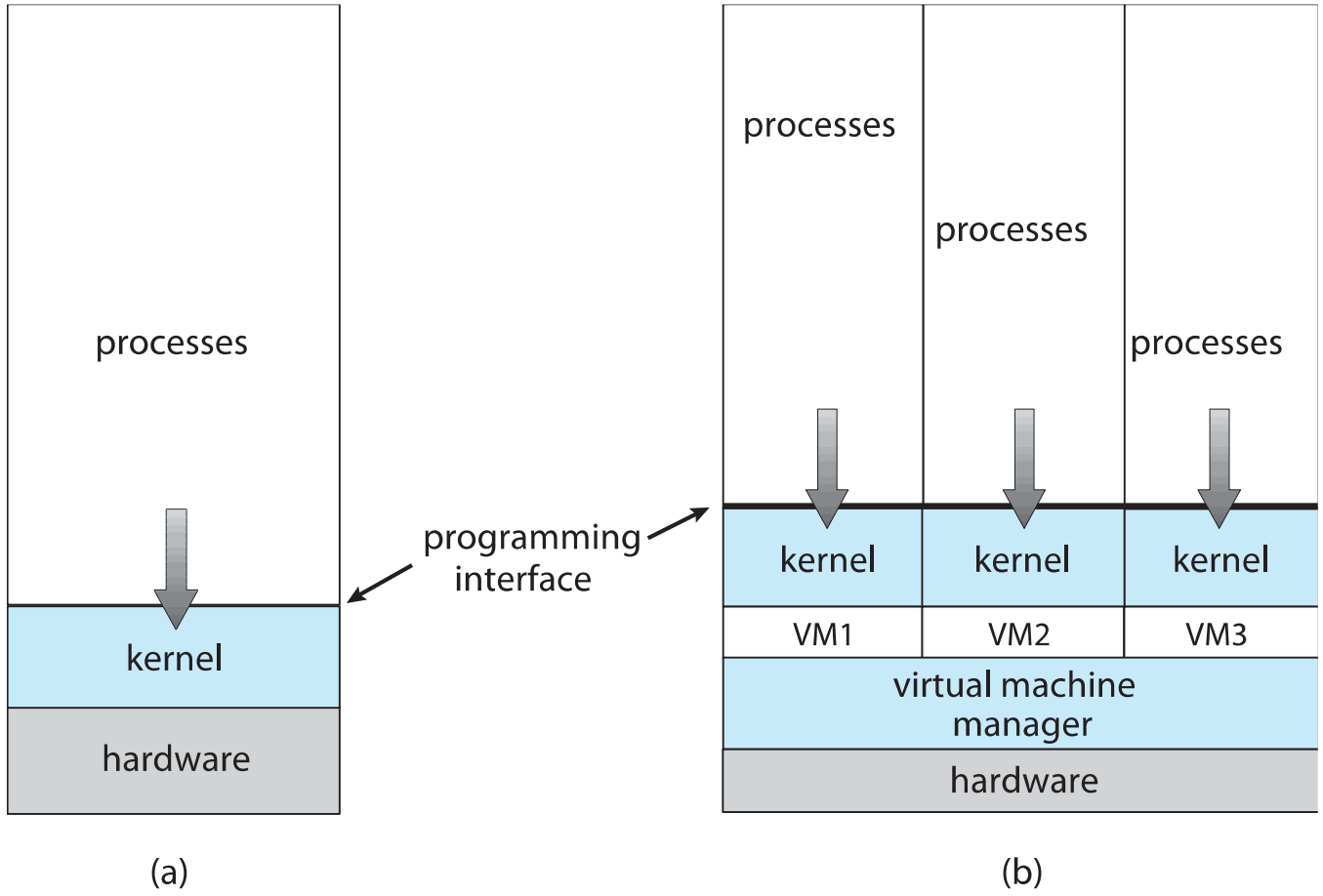
Timer

- A timer is a mechanism used by the operating system to maintain control over processes and ensure fairness in resource usage.
- **Purpose of a Timer:**
  - **Prevent Infinite Loops:** Stops a program from running endlessly and hogging the CPU.
  - **Resource Management:** Ensures no single process uses more time than allowed.
- **How It Works:**
  - **Timer Setup:** The operating system sets a timer before scheduling a process. This is done using a privileged instruction, meaning only the OS can set the timer.
  - **Decrementing Counter:** The timer has a counter that is decremented by the computer's physical clock.
  - **Interrupt on Timeout:** When the counter reaches zero, the timer generates an interrupt. The interrupt gives control back to the OS.
  - **Action by OS:** The OS either: Stops or reschedules the process that exceeded its time limit. Terminates it if necessary.

Virtualization

- **Virtualization:**
  - Allows multiple operating systems (OSes) to run on the same hardware simultaneously.
  - Example: Running Windows XP inside Windows 10 using VMware.
- **How Virtualization Works:**
  - The **host OS** runs a **Virtual Machine Manager (VMM)**, which provides virtualization services.
  - The VMM creates **virtual machines (VMs)** that allow guest OSes to run as if they have their own hardware.
  - Both the host OS and guest OSes are **natively compiled** for the same CPU.

Computing Environments - Virtualization



Why Applications are Operating System Specific

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java)
  - Use standard language (like C), compile separately on each operating system to run on each
- **Application Binary Interface** (**ABI**) is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.