OPERATING SYSTEM

Introduction

What Is An Operating System?

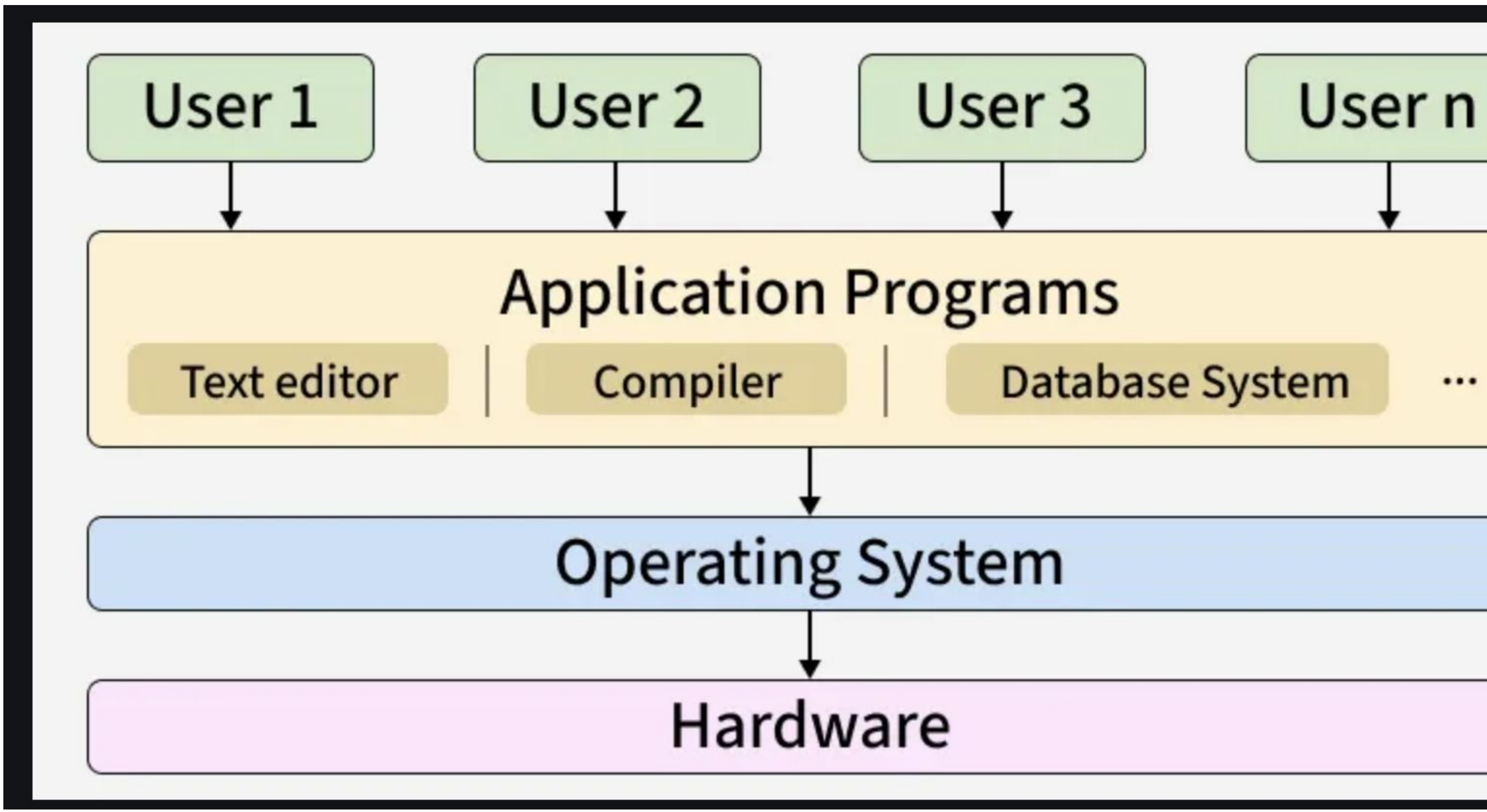
Lots of hardware !!

- One or more processors
- Main memory
- Disks
- Printers
- Various input/output devices

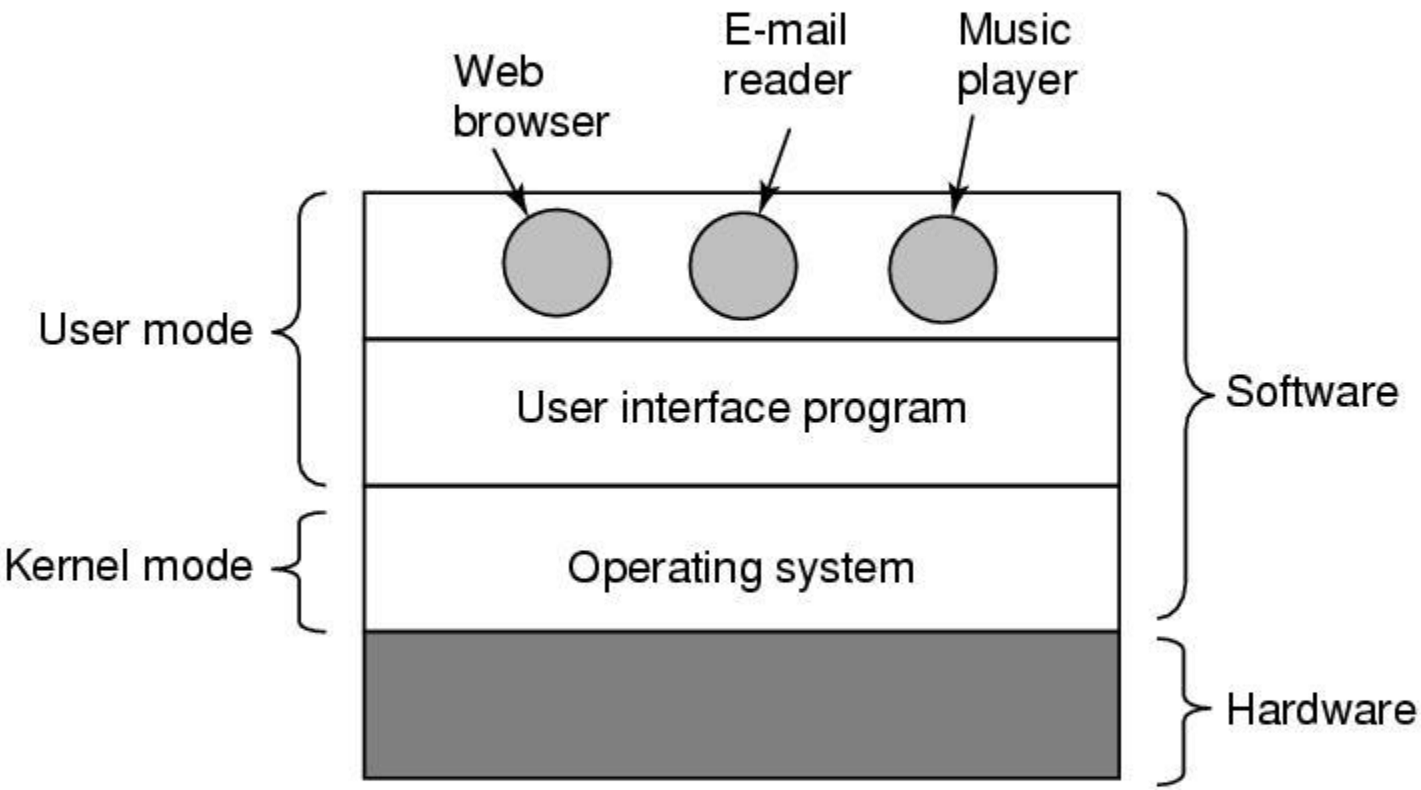
Managing all these components requires a layer of software – the **operating system**

What Is An Operating System?

- Create a abstract view for the user
- intermediary between User and Hardware

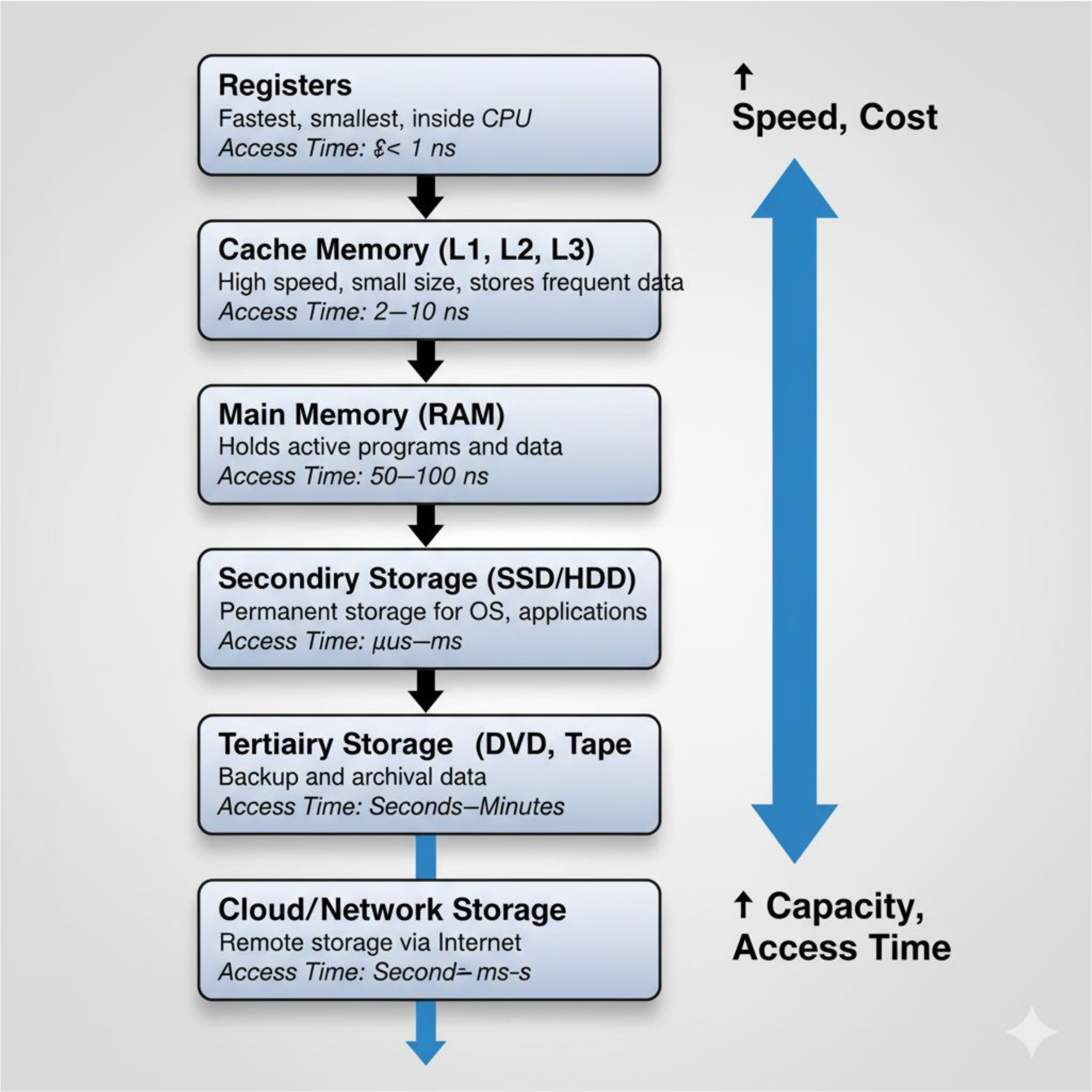


Where is the software?

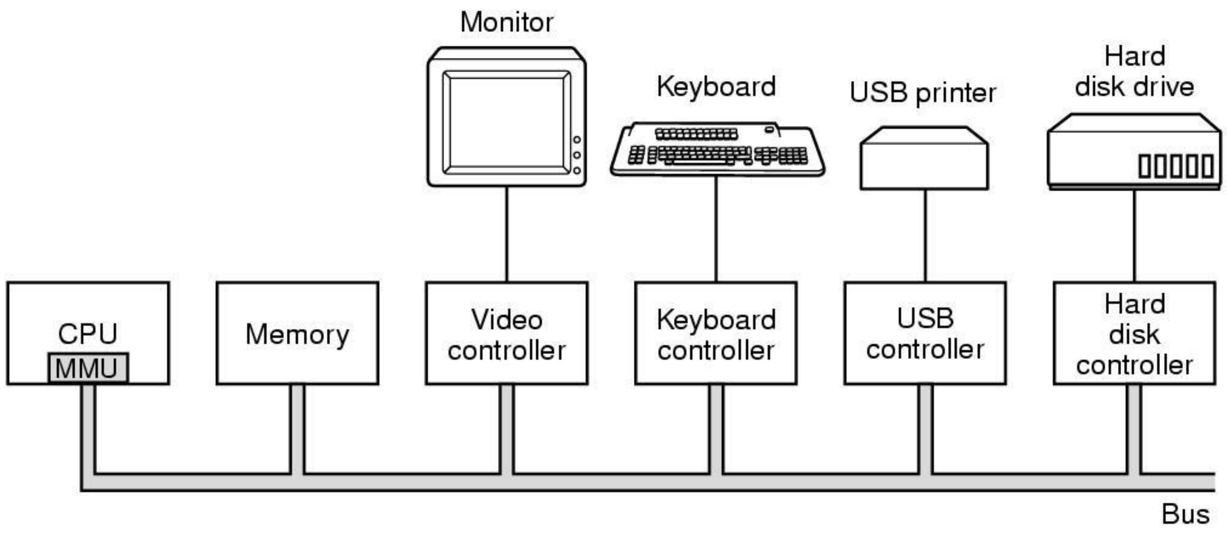


Where the operating system fits in.

Computer Hardware Review



Computer Hardware Review



Some of the components of a simple personal computer.

Main Purposes of Operating System

Process Management

- Handles process creation, scheduling, and termination.
  - CPU scheduling, context switching

Memory Management

- Keeps track of memory usage and allocates memory.
- Paging, segmentation

File Management

- Manages file storage, retrieval, and permissions.
- File system, directories

I/O Management

- Controls and coordinates I/O devices.
- Buffering, spooling

Security & Protection

- Controls access to system resources.
- User authentication

Resource Allocation

- Distributes resources among users/processes.
- CPU time, memory slots

Error Detection

- Detects and handles system errors.
- Hardware/software failure recovery

Main Purposes of Operating System

Process Management

A **process** is a **program in execution**.
It includes: The program code (text section), Current activity (Program Counter, registers), Process Stack (function calls, parameters), Data

Process Control Block (PCB)

Each process is represented by a **PCB** — a data structure the OS uses to manage processes, containing Process ID (PID) Process State, Program Counter etc.

Process Creation: The OS creates processes using system calls

CPU Scheduling

When multiple processes are in the **Ready Queue**, the OS decides **which process gets the CPU next**. This decision is made by the **CPU Scheduler**.

Algorithm: FCFS (First Come First Served), SJF (Shortest Job First), RR (Round Robin), Priority Scheduling

Main Purposes of Operating System

Memory Management Tasks

- Tracking: Keeps a record of used and free memory blocks
- Allocation: Assigns memory to processes
- Deallocation: Frees memory after process ends
- Swapping: Moves processes between main memory and disk

What is Paging: Paging is a memory management technique that avoids external fragmentation by dividing memory into fixed-size blocks.

What is Segmentation: Segmentation divides a program into logical segments based on functionality or data type — unlike paging, which uses fixed-size blocks. Each segment represents a logical unit such as: Code segment, Data segment, Stack segment

Main Purposes of Operating System

Virtual Address Space (per process)

Page 0 (4 KB)	← Virtual Address 0x0000
Page 1 (4 KB)	
Page 2 (4 KB)	
Page 3 (4 KB)	

↓

Page Table (managed by OS)

Page #	Present?	Frame # (in RAM)	
0	Yes	3	→ Points to Frame 3
1	No	—	→ Not in RAM (on disk)
2	Yes	1	→ Points to Frame 1
3	Yes	4	→ Points to Frame 4

↓

Physical Memory (RAM)

Frame 0 (4 KB)	
Frame 1 ← Page 2	← Loaded
Frame 2	
Frame 3 ← Page 0	← Loaded
Frame 4 ← Page 3	← Loaded

Main Purposes of Operating System

File Management Tasks

File Management refers to the part of the OS responsible for:

- Storing files on disk
- Retrieving them efficiently
- Organizing files into directories/folders
- Controlling access and permissions

Some system calls related to file management: Create, Open, Read, Write, Delete, Close, Seek

File Permissions and Protection: To ensure security, OS enforces access control for files.

r, w, x for Owner, Group, Others

example permission of a file: -rwxr-xr--

I/O operations

Enable communication between the computer system and the external environment. The operating system manages these devices to ensure smooth data transfer between hardware and software.

Categories of I/O Operations

- **Programmed I/O:** CPU directly controls all I/O operations — simple but inefficient.
- **Interrupt-Driven I/O:** CPU performs other tasks and is interrupted only when I/O is ready.
- **Direct Memory Access (DMA):** Allows devices to transfer data directly to/from memory without CPU intervention — improves performance.

I/O operations

1. Programmed I/O (Polling I/O)

How It Works:

In Programmed I/O, the CPU directly controls every aspect of the I/O operation. It repeatedly checks (or *polls*) the status of the I/O device to see if it is ready to send or receive data. Once the device is ready, the CPU transfers the data between the device and memory.

Example:

Imagine you're waiting for a pizza delivery:

- Instead of doing anything else, you stand by the door every second, checking if the delivery person has arrived.
- Only when they arrive do you take the pizza.
- While waiting, you do nothing else.

I/O operations

1. Programmed I/O (Polling I/O)

Advantages:

- Simple to implement—no complex hardware needed.
- Good for very slow or simple devices (e.g., keyboard in early systems).

Disadvantages:

- Wastes CPU cycles—CPU is idle while polling.
- Inefficient for high-speed or bursty I/O.
- Poor system throughput—CPU can't do useful work during I/O.

```
while (device_status != READY);  // CPU keeps polling
data = read_from_device();       // CPU reads data
write_to_memory(data);
```

I/O operations

2. Interrupt-Driven I/O

How It Works:

Here, the CPU initiates an I/O request and then goes on to do other tasks. When the I/O device is ready (e.g., data is available or buffer is free), it sends an interrupt signal to the CPU. The CPU then pauses its current work, saves its state, and runs an Interrupt Service Routine (ISR) to handle the I/O. Data is still transferred via the CPU, but only when needed.

Example:

- You order the pizza, then go back to watching a movie.
- When the delivery person arrives, they ring the doorbell (interrupt).
- You pause the movie, get the pizza, then resume watching.

Simplified Flow:

- CPU starts I/O
- CPU continues other work
- Device finishes → sends interrupt

I/O operations

2. Interrupt-Driven I/O

Advantages:

- Much more efficient—CPU isn’t wasted polling.
- Better system responsiveness and multitasking.

Disadvantages:

- Still involves CPU for every data transfer
- Overhead from frequent interrupts (e.g., for high-speed devices like disks or network cards, this causes interrupt storms).
- Latency in responding to interrupts.

I/O operations

3. Direct Memory Access (DMA)

How It Works:

DMA uses a dedicated hardware controller (the DMA controller) to transfer data directly between I/O devices and main memory, without CPU involvement during the actual data transfer.

The CPU only: Initializes the DMA transfer (by providing source/destination addresses and byte count). Gets interrupted when the transfer is complete.

During the transfer, the CPU is free to execute other instructions.

Example: Pizza delivery with a butler

You tell your butler: “When pizza arrives, put it in the fridge.”

You go to work.

The butler handles everything—takes pizza, puts it away.

Only when done does he notify you (optional).

I/O operations

3. Direct Memory Access (DMA)

Advantages:

Massive performance gain—no CPU overhead during bulk transfers.

Ideal for high-speed, large-data devices: disk drives, network cards, video cards.

Reduces interrupt overhead (only 1 interrupt per transfer vs. per byte).

Disadvantages:

Requires extra hardware (DMA controller).

Bus contention: DMA and CPU may compete for memory bus access.

More complex to implement.