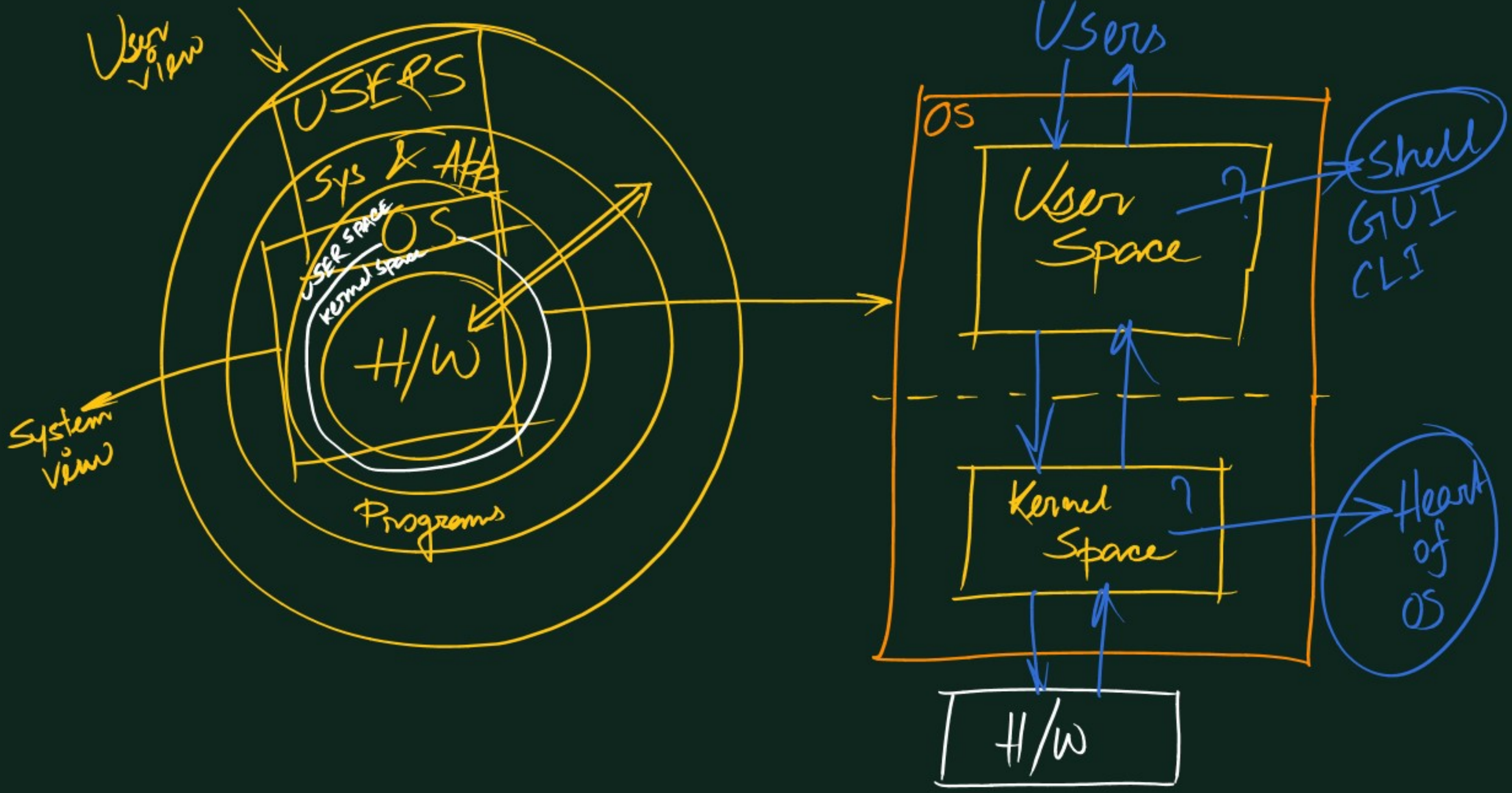
User's view
USERS
Sys & Apps
OS
USER SPACE
KERNEL SPACE
H/W
Programs
System's view
Users
OS
User Space
Shell
GUI
CLI
Kernel Space
Heart of OS
H/W

Class 6:

Topics:

1. **Application Program**
2. **System Program**
3. **Kernel**

---

1. Application Program

- **Definition:**
  Software designed to perform a specific task for the user.
- **Purpose:**
  Directly helps the user to do work, entertainment, or communication.
- **Examples:**
  MS Word (writing), Photoshop (editing), Chrome (browsing), VLC (media).
- **Analogy:**
  Think of an application program like the "apps" on your smartphone. They are like *furniture in a house*—customized to your needs (sofa, table, TV).

---

2. System Program

- **Definition:**
  Software that helps run the computer hardware and provides a platform for application programs.
- **Purpose:**
  Acts as a bridge between the user applications and the hardware, ensuring smooth execution.
- **Examples:**
  - Operating systems components (like file management utilities, disk defragmenter).
  - Compilers, assemblers, device drivers.
- **Analogy:**
  System programs are like the **electric wiring and plumbing** in a house—hidden but essential for everything else to work. You don't directly use them for fun, but without them, your furniture (applications) won't function properly.

---

3. Kernel

- **Definition:**
  The **core part of the operating system** that directly interacts with the hardware.
- **Purpose:**

- Manages CPU, memory, I/O devices.
- Provides essential services to system and application programs.

- **Examples:**
  Linux kernel, Windows NT kernel.
- **Analogy:**
  The kernel is like the **foundation and skeleton of the house**—you don't see it or interact with it directly, but without it, the whole house collapses.

🌐 **Relationship Between Them**

- **Application Program:** User-facing → "What you see and use."
- **System Program:** Middle layer → "Helps applications run smoothly on the system."
- **Kernel:** Deepest layer → "Controls the hardware resources."

👉 You can also draw a **3-layer diagram**:

[Application Programs] → Games, Browsers, MS Office

[System Programs] → Compiler, File manager, Drivers

[Kernel] → CPU, Memory, Device control

[Hardware] → Actual computer hardware

I. Components of the Operating System

The OS can be logically divided into two areas:

- **User Space** (User Mode) → Where users and applications live.
- **Kernel Space** (Kernel Mode) → The protected zone where the kernel runs.

Together, they make up the Operating System.

A. Kernel Space (Kernel Mode)

The Kernel Space is the protected area where the **Kernel** runs.

- The **Heart of the OS** and its **core component**.
- The **first part of the OS to load at startup**.
- Has **direct access to hardware** (CPU, memory, devices).
- Performs the most crucial tasks of the OS.

👉 Think of it as the **foundation and skeleton of a house** — invisible to the user, but without it everything collapses.

B. User Space (User Mode)

The User Space is the environment where applications and shells run.

- Applications here **cannot directly access hardware** (they rely on the kernel).
- Provides the environment for **user interaction with the OS**.
- User Space must interact *with the kernel* to access hardware.

User interaction occurs through two types of shells (interfaces):

- **GUI (Graphical Shell):** e.g., Windows Explorer (Desktop, Taskbar, File Explorer, Finder in macOS).
- **CLI (Command-line Shell):** e.g., Command Prompt, PowerShell, Bash.

👉 Analogy: GUI and CLI are **two doors to the same house**. Whether you click *New → Folder* on the desktop or type mkdir OS in Command Prompt, both shells send requests to the **same kernel**, which does the real work.

💡 **Interaction Example: Creating or Deleting a Folder**

1. **User →** creates a folder "OS"
    - **GUI way:** Right-click → New → Folder → Name it "OS."
    - **CLI way:** mkdir OS.
2. **Shell (GUI or CLI) →** interprets the user's action.
3. **System Programs →** File manager routines check where and how to create the folder.
4. **Kernel →** updates the file system metadata, allocates space, and tells the hardware (disk) to store it.
5. **Hardware →** physically creates the folder entry.

Deleting works the same way:

- **GUI delete (Explorer):** Usually moves the folder to Recycle Bin (unless Shift+Delete).
- **CLI delete (rmdir OS):** Removes the entry directly.

👉 The **shell doesn't delete anything itself** — it simply asks the kernel to do it.

II. Functions of the Kernel

The kernel manages **system resources** and executes the OS's critical tasks.

1. Process Management

- Create, terminate, suspend, resume processes.
- CPU scheduling (who gets CPU next).
- Context switching.
- Synchronization & communication between processes.

2. Memory Management

- Allocate and deallocate memory.
- Keep track of which process uses which memory.
- Reclaim memory when processes exit.

3. File Management

- Create and delete files/directories.
- Maintain a **hierarchical structure** (root → subfolders → files).
- Map files to secondary storage.
- Provide backup and recovery support.

👉 This is where your **mkdir / delete example** directly fits in — file management lives here.

4. I/O Management

- Add and manage I/O devices.
- Coordinate data transfer between CPU and devices.
- Notify the OS when new devices connect.

Key Techniques:

- **Spooling:** Handling speed mismatch (e.g., print jobs queued).
- **Buffering:** Temporary storage within a process (e.g., YouTube buffering).
- **Caching:** Frequently used data kept handy (e.g., memory cache, web cache).

III. Types of Kernels

1. Monolithic Kernel

- All functions (process, memory, file, I/O management) are inside one big structure.
- High performance (fast communication inside kernel).
- Less reliable (one crash = whole OS crash).
- Examples: Linux, Unix, MS-DOS.

2. Microkernel

- Only essential services (process, memory) are in the kernel.
- File, I/O management moved to User Space.
- More reliable, but slower due to frequent **mode switches**.
- Examples: MINIX, Symbian OS, L4.

3. Hybrid Kernel

- Mix of both approaches: speed of monolithic + stability of microkernel.
- Keeps critical services inside, moves others outside.
- Examples: Windows NT, Windows 10, macOS.

IV. Inter-Process Communication (IPC)

Since processes are isolated, they need **IPC** to talk.

- **Shared Memory:** Both processes read/write from the same block.
- **Message Passing:** Processes send messages via OS-provided channels (pipes, queues).

🏠 **House Analogy Recap**

- **Application Programs** = Furniture (Word, Chrome, Games — what you actually use).
- **System Programs** = Wiring & Plumbing (compilers, file managers, drivers).
- **Shells (CLI/GUI)** = Doors (different ways of entering the house).
- **Kernel** = Foundation & Control Room (keeps the house stable and running).
- **Hardware** = The actual building material (CPU, RAM, Disk).
