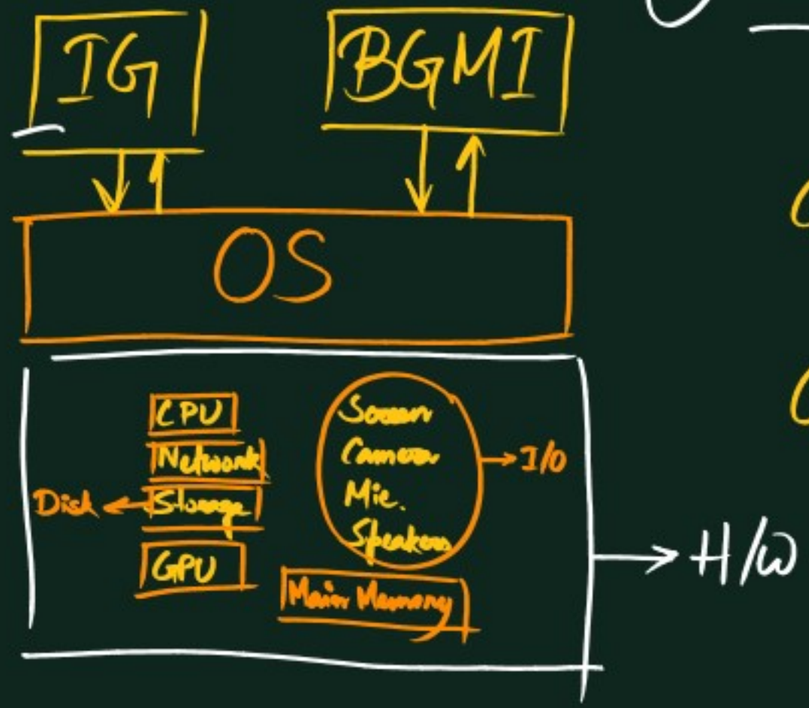
Operating Systems

- What?
- Why?

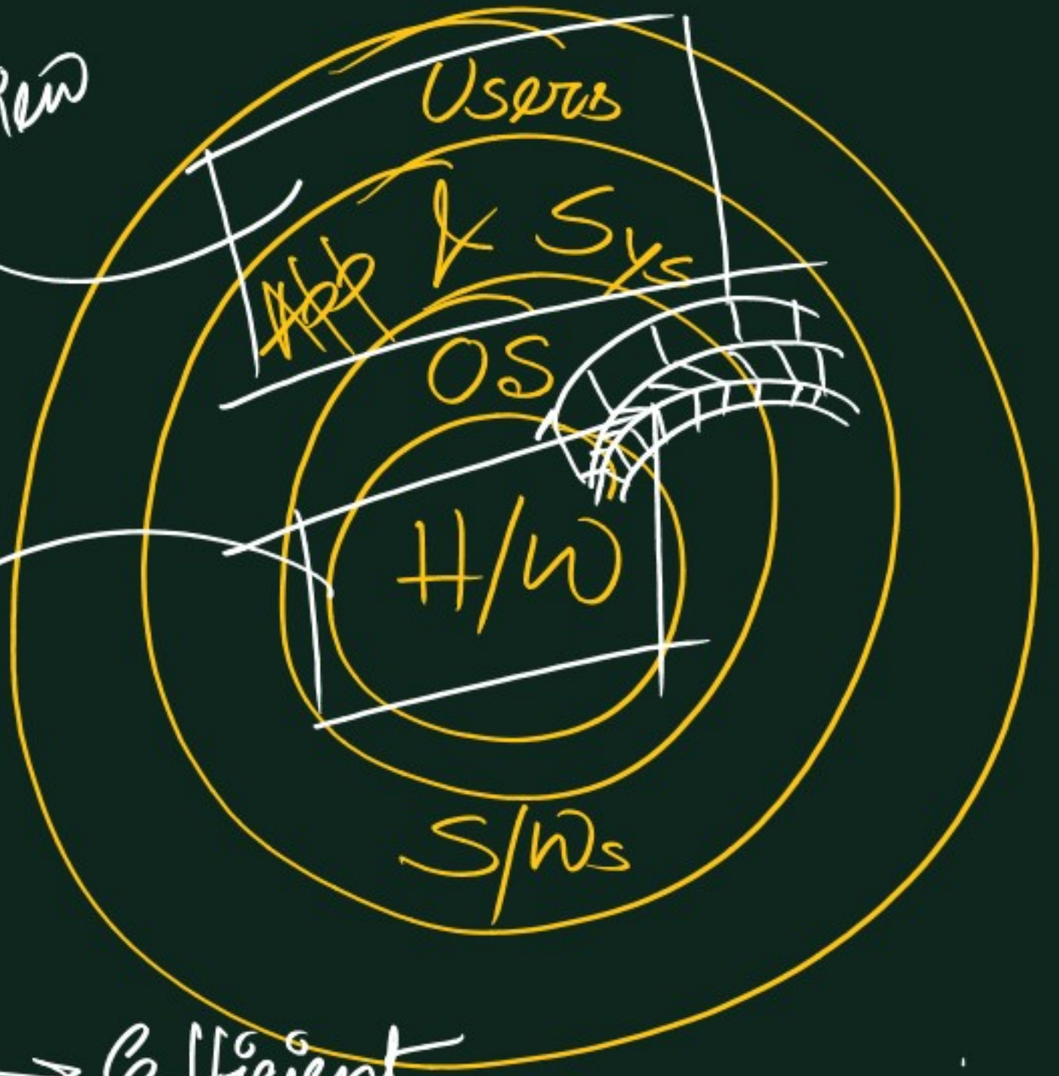


① Resource Management

	IG	BGMI
CPU:	2%	50%
Mem:	10%	60%
GPU:	5%	20%

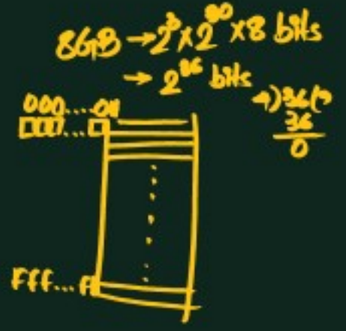
② Interface: ✓

Convenience ← User View



System View → Efficient Management

⑧ Memory Management



IG

→ Code of IG

≃ 50 MB

1 GB

→ Mem Mgmt

BGMI

→ Code of BGMI

≃ 2 GB

1 GB

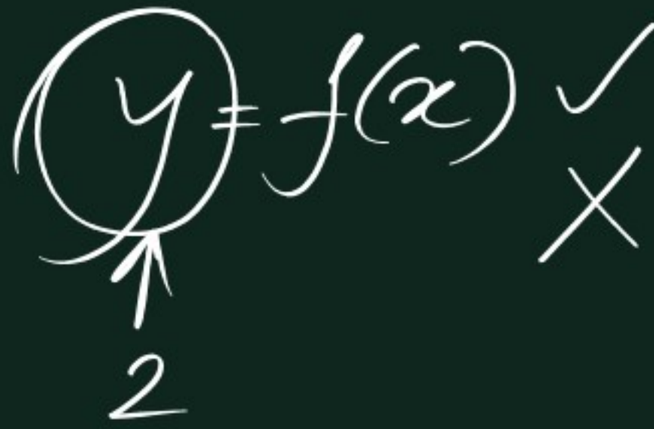
Mem Mgmt

malloc, new, etc.

OS

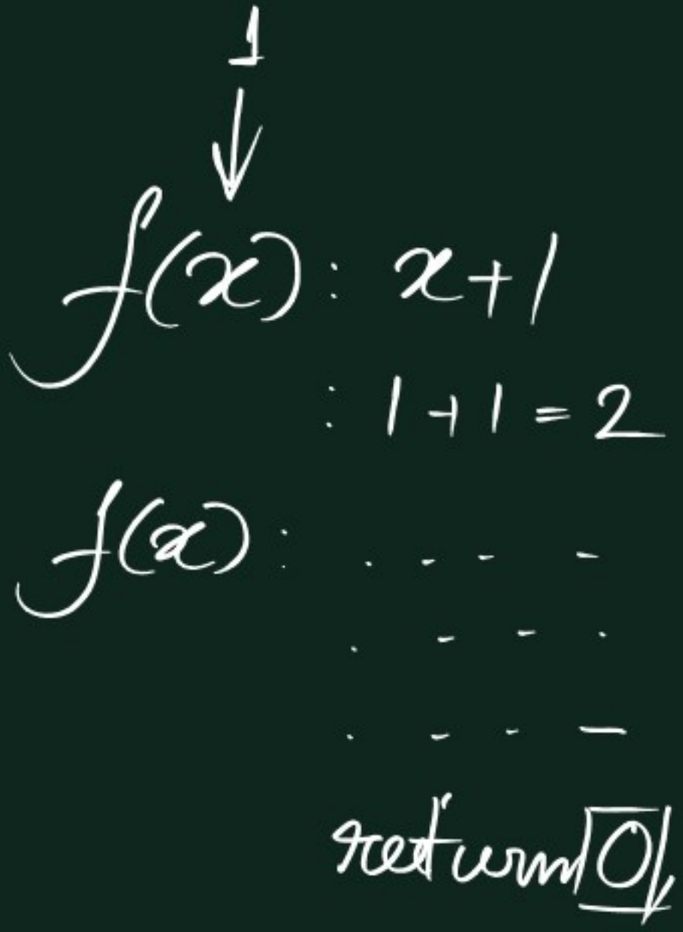
DRY

↳ Don't Repeat Yourself/Your Code

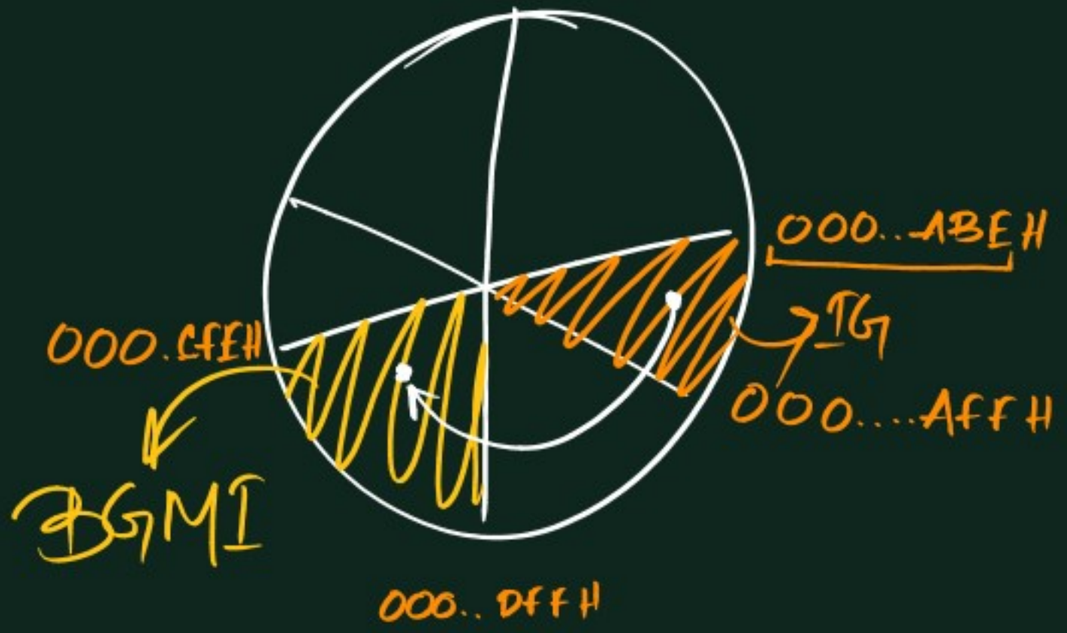

y = f(x)
2

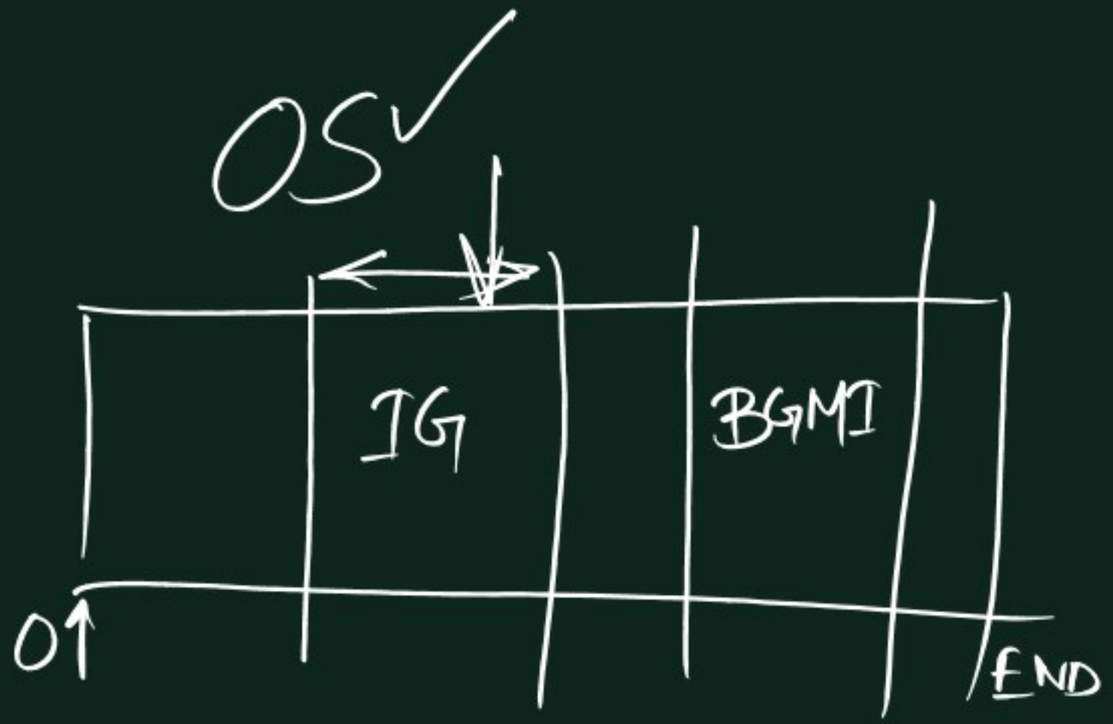
✓

✗


1
f(x) : x+1
: 1+1 = 2
f(x) :
return 0;

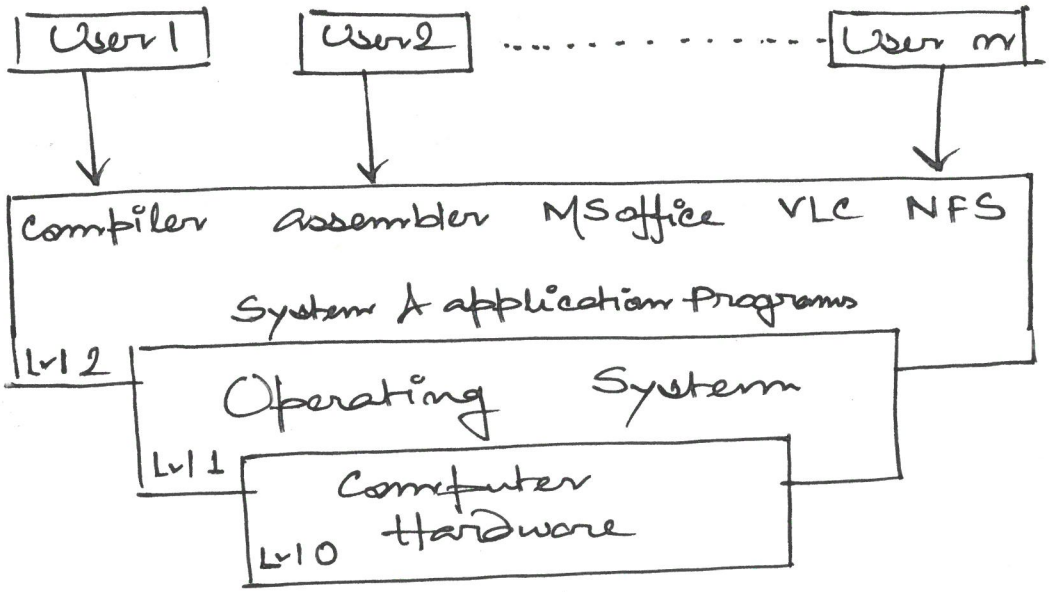
④ Isolation & Protection:





- **Definition:** It's a system software which,
  - (I) acts as an intermediatory between hardware & user.
  - (II) manages system resources in an unbiased fashion for both h/w & s/w
  - (III) provides a platform on which other application programs are installed.

- **Abstract view:**



- **Goals & Functions:**
  - **Goals:**
    - Primary goal: Convenience / User friendlyness
    - Secondary goal: Efficiency
  - **Functions:**
    - Process management
    - Memory management
    - I/O device management
    - File management
    - Network management
    - Security & Protection

Class 1

Topic1: Understanding the Operating System (OS)

Definition:
An **Operating System (OS)** is a piece of software that manages all the resources of a computer system—both hardware and software—and provides an environment in which the user can execute programs conveniently and efficiently. It hides the underlying complexity of hardware and acts as a **resource manager**.

Why Do We Need an OS?

1. **Without an OS:**
    - **Bulky and Complex Applications:** Every app would need its own hardware-interaction code, making the code base huge.
    - **Resource Exploitation by a Single App:** Apps like **Instagram** or **BGMI** could monopolize CPU, memory, or GPU resources, causing instability.
    - **No Memory Protection:** One app could overwrite another app's memory.
2. **Efficient Resource Allocation:**
    - The OS ensures fair and controlled distribution of resources (e.g., **Instagram** gets 5%, **BGMI** gets 50%).
    - Prevents conflicts and maintains system stability.

Key Functions of an OS

- **Resource Management (Arbitration):** Allocates and manages CPU, memory, storage, devices, files, security, and processes.
- **Access to Hardware:** Provides controlled access to the computer's hardware components.
- **Interface/Bridge:** Acts as an interface between the user, applications, and the hardware.
- **Isolation & Protection:** Prevents apps from interfering with each other (e.g., **Instagram** cannot overwrite **BGMI**'s memory).
- **Abstraction:** Hides the underlying hardware complexity, letting developers focus on application logic.
- **Program Facilitation:** Enables smooth execution of applications by providing isolation and protection.

Q: Why in OS resource management is often called arbitration?

Arbitration – English meaning:
The word *arbitration* comes from *arbiter*, meaning a neutral judge or referee. In everyday English it

refers to **settling a dispute or making a fair decision when two or more parties want the same thing**.

In Operating Systems:
Resources such as the CPU, memory, disk, or I/O devices are limited, while many programs may request them at the same time. The OS acts like that neutral judge—it **arbitrates** by deciding **who gets the resource, for how long, and in what order**.

Examples:

- If several processes are ready to run, the OS scheduler **arbitrates** CPU time according to a scheduling algorithm.
- When multiple programs want to write to a file or send data to the printer, the OS **arbitrates** to prevent conflicts and ensure fairness or priority handling.

So, in OS terminology, **resource management is often called arbitration** because the core of managing resources is this **decision-making process of fair allocation and conflict resolution**, just as an arbiter settles disputes in everyday life.

Topic2: User View and System View:

What is a "View" in the Context of an Operating System?

In Operating Systems, a **view** means **the perspective from which the OS is observed or experienced**.

- **User View:** How the OS appears to end-users or application developers—the *experience* of using the computer.
- **System View:** How the OS appears to the computer's hardware and the low-level processes—the *internal management* of resources.

Think of it like looking at the same city through two lenses:

- As a **tourist**, you care about roads, attractions, and easy navigation.
- As a **city planner**, you focus on traffic signals, water lines, and power grids.
  Both are the same city, just different perspectives.

---

1. User View of an OS

This is the **outside experience**, focusing on **convenience and responsiveness**.

Key Points & Relatable Examples

- **Ease of Use:** A user just wants to run apps like **Instagram** or **BGMI** and play music or edit documents without worrying about CPU scheduling.
- **Abstraction of Hardware:** The OS hides complex hardware details. Clicking "Save" doesn't require you to know how disk sectors are written.

- **Interactivity & Interface:** Graphical interfaces (Windows, macOS, Android) or command lines (Linux terminal) are all user-side views.
- **Performance & Reliability:** Users judge the OS by speed, stability, and how smoothly multiple apps run together.

Analogy: Like driving a car—you care about the steering wheel and dashboard, not the engine timing.

---

2. System View of an OS

This is the **inside management perspective**, focusing on **efficient resource utilization and protection**.

Key Points & Relatable Examples

- **Resource Manager / Arbiter:** The OS must allocate CPU cycles, memory, disk, and I/O devices fairly. It **arbitrates** when Instagram and BGMI both request GPU power.
- **Process Coordination & Protection:** Prevents one process from overwriting another's memory; enforces isolation and security.
- **Scheduling & Control:** Chooses which process runs next, manages interrupts, and balances loads for best throughput.
- **Hardware Interaction:** Deals with device drivers, interrupts, and low-level tasks invisible to the user.

Analogy: Like the city planner ensuring water supply, power distribution, and traffic flow behind the scenes so citizens never see the complexity.

Summary Table

Aspect	**User View**	**System View**
Focus	Convenience, responsiveness, interface	Efficient resource management and protection
Concern	"Can I run my apps smoothly?"	"How do I allocate CPU/memory safely?"
Example	Launching Instagram or BGMI	Scheduling CPU time among all active tasks
Abstraction	Hides hardware complexity	Deals directly with hardware mechanisms

Takeaway:
The **User View** highlights the **experience**, while the **System View** highlights the **engineering**.
Together they explain why an Operating System is both a *service provider* for users and an *arbiter* for hardware resources.
