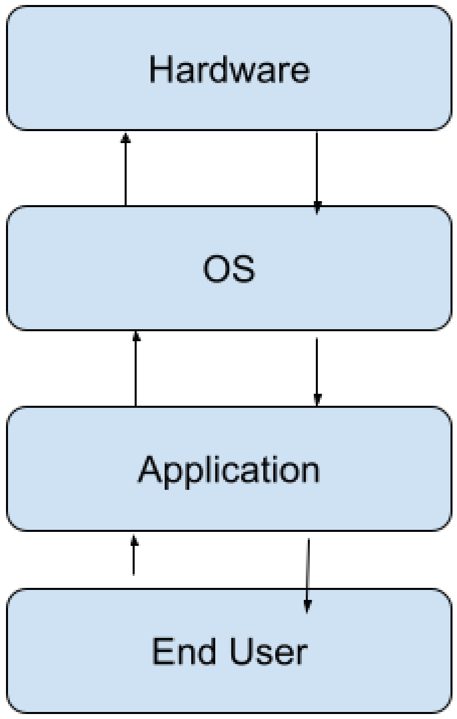
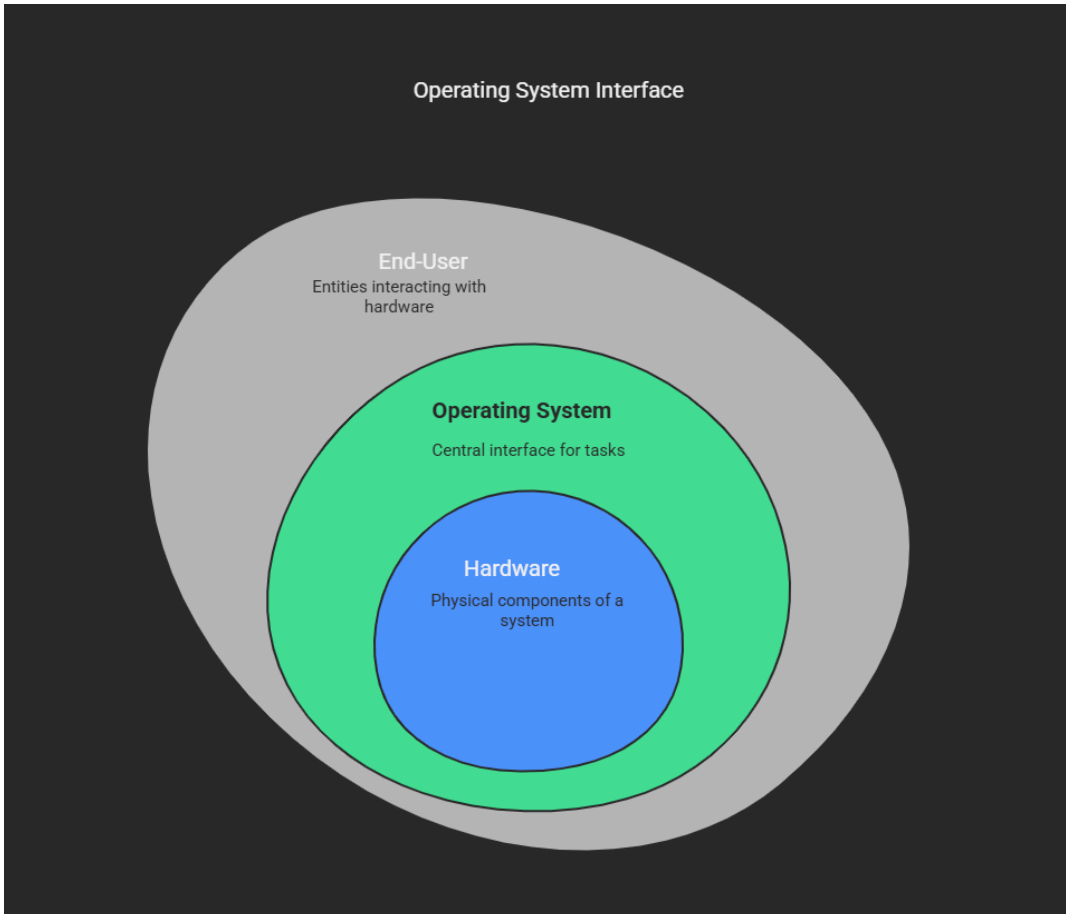
What is an operating System ?

This is an interface between hardware and end-user, where the end-user can be human, robot or any other software.
It provides communication between end-user and hardware to accomplish any task.

Ex: **Linux, Windows, Mac OS, Ubuntu, Cent OS, Fedora.**




Operating System Interface
End-User
Entities interacting with hardware
Operating System
Central interface for tasks
Hardware
Physical components of a system

Linux was developed by **Linus Torvalds in 1991**, he took this as his hobby project to work on the limitations of UNIX. He used the source code of **MiNUX** and released the first development under GNU-GPL.

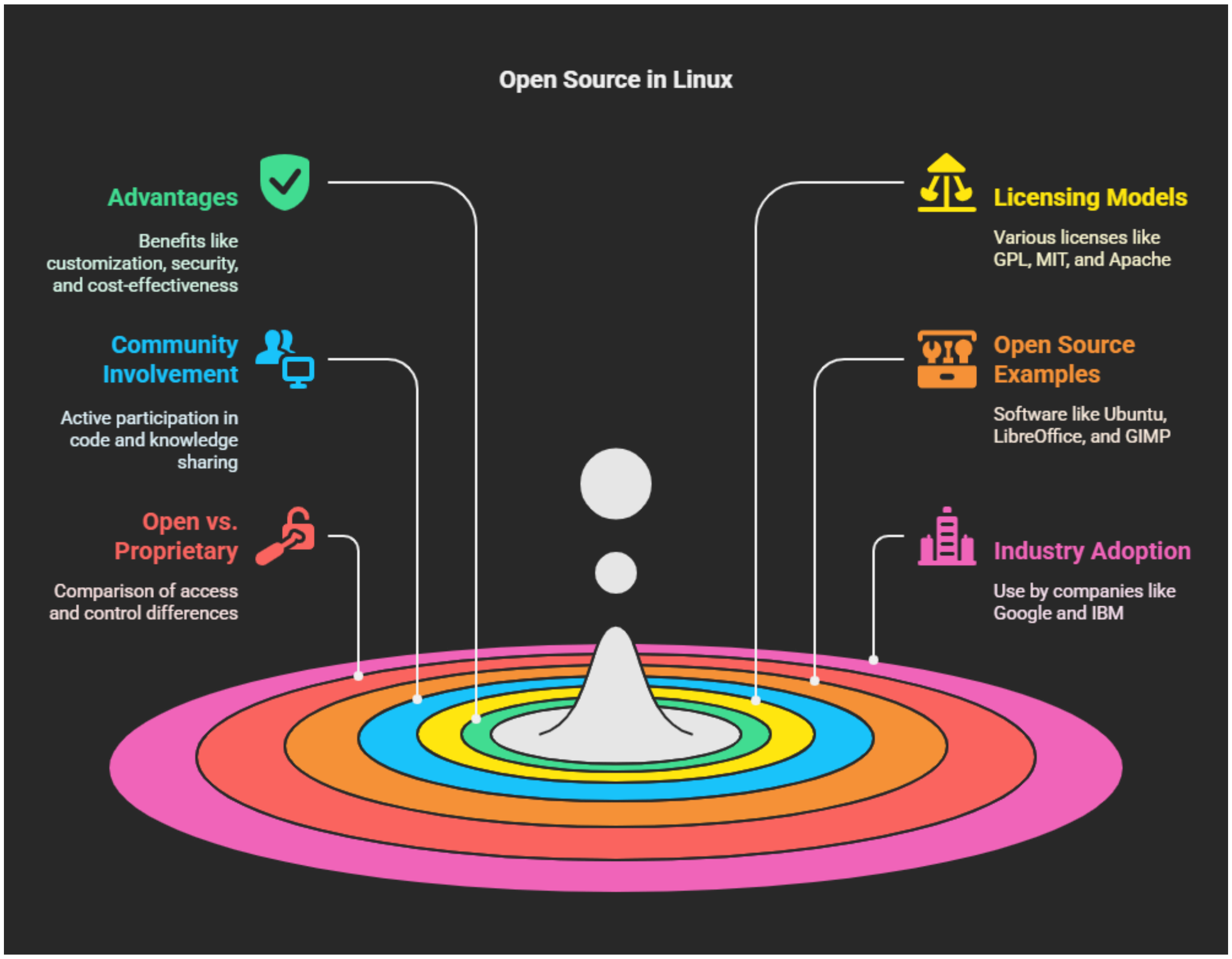
What do we understand by open source in Linux ?

In linux, we can have access to any source code and we have freedom to modify those codes. The codes are openly available to anyone who wants to examine it. This gives you freedom to understand how the software works, identify the bugs, suggest improvements or to contribute in any enhancement to the software.

As it is open source, it is typically **free of cost** and allows you to **download, install and use without any upfront costs.**

Open Source in the context of Linux refers to, licensing and distribution model of operating systems
Where the licensing under Linux runs under open source licenses such as **GNU-General Public License(GPL)** or License Approved by **Open Source Initiative (OSI).**

Open source in the context of Linux indeed involves specific licensing and distribution models that ensure the software's source code is accessible to everyone.


Open Source in Linux
Advantages
Benefits like customization, security, and cost-effectiveness
Community Involvement
Active participation in code and knowledge sharing
Open vs. Proprietary
Comparison of access and control differences
Licensing Models
Various licenses like GPL, MIT, and Apache
Open Source Examples
Software like Ubuntu, LibreOffice, and GIMP
Industry Adoption
Use by companies like Google and IBM

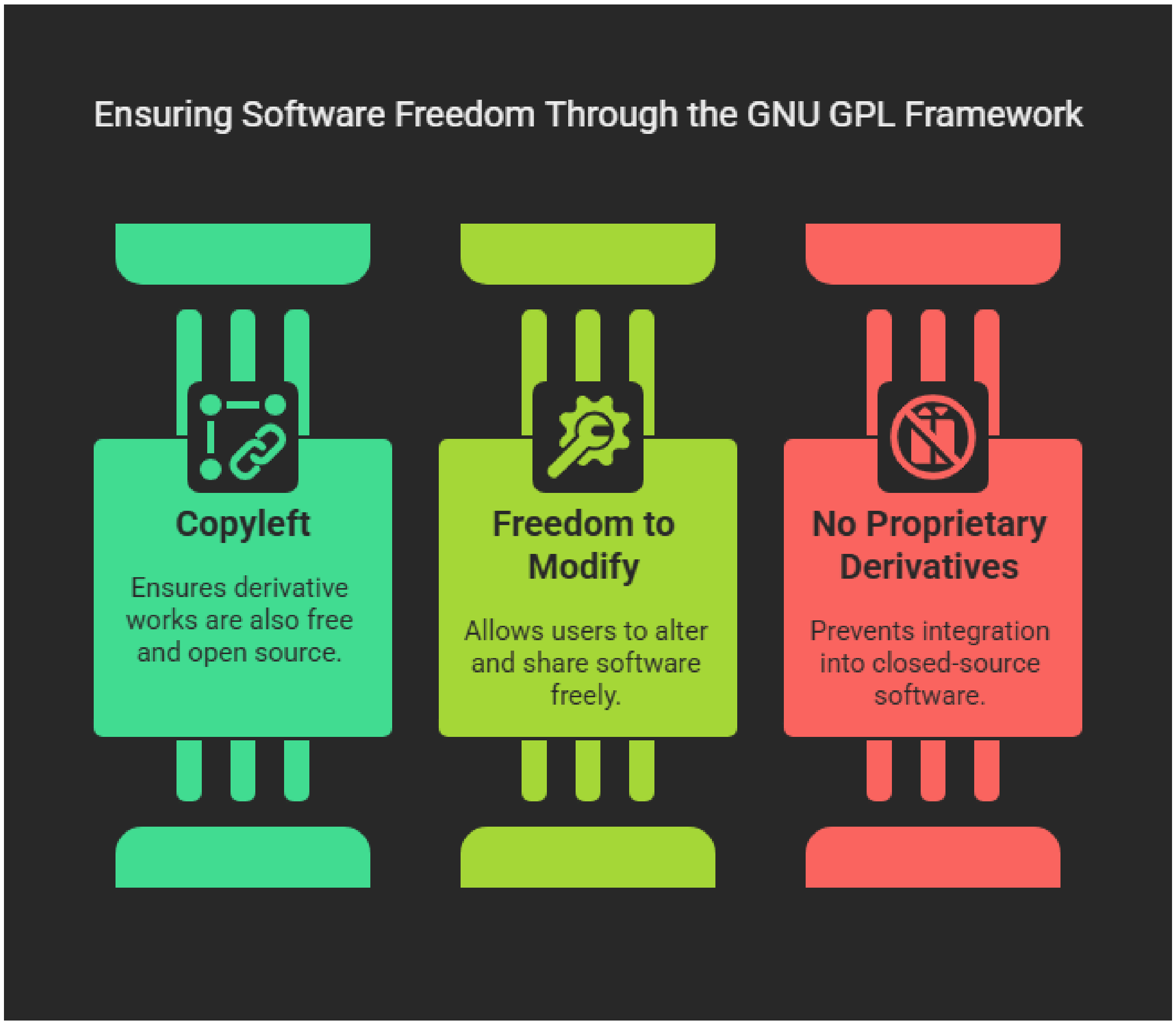
Licensing and Distribution Model in Linux:

- **Licensing**: Most Linux distributions are released under the GPL or other OSI-approved licenses, ensuring that users have access to the source code and the ability to modify and redistribute it.
- **Distribution**: Linux distributions (distros) can be freely distributed, and users can create and share their **own versions of Linux,** contributing to the rich diversity of the Linux ecosystem.

Here’s a breakdown of the key components you mentioned:

GNU General Public License (GPL):

- **Definition**: The GNU General Public License (GPL) is a widely used free software license that ensures **end users have the freedom to run, study, share, and modify the software.**
- **Key Features**:
    - **Copyleft**: The GPL license is a **"copyleft" licens**e, meaning any derivative work based on GPL-licensed software must also be distributed under the same license. This ensures that the **software remains free and open for all future users.**
    - **Freedom to Modify**: Users can modify the source code and distribute their modifications, but they must also release the source code of their modified version.
    - **No Proprietary Derivatives**: GPL-licensed software cannot be integrated into proprietary (closed-source) software without violating the license.


Ensuring Software Freedom Through the GNU GPL Framework
Copyleft
Ensures derivative works are also free and open source.
Freedom to Modify
Allows users to alter and share software freely.
No Proprietary Derivatives
Prevents integration into closed-source software.

Open Source Initiative (OSI) Approved Licenses:

- **Definition**: The Open Source Initiative (OSI) is a non-profit organization that promotes and protects open source software by **reviewing and approving licenses t**hat meet their definition of "open source."
- **Key Features**:
  - **Freedom to Use**: OSI-approved licenses allow users to freely use the software for any purpose.
  - **Freedom to Modify**: Users can access the source code, make changes, and distribute those changes.
  - **Freedom to Distribute**: Users can distribute the software, whether it's in its original form or with modifications, to anyone and for any purpose.
  - **Variety of Licenses**: OSI approves a variety of licenses, ranging from permissive ones like the MIT License, which allows proprietary use, to copyleft licenses like the GPL, which require derivative works to remain open source.

    Open-source licenses come in different types, depending on how much freedom they give users and developers to modify and distribute the software. The **Open Source Initiative (OSI)** approves a variety of these licenses, which generally fall into two main categories:

    1. Permissive Licenses (More Flexibility)

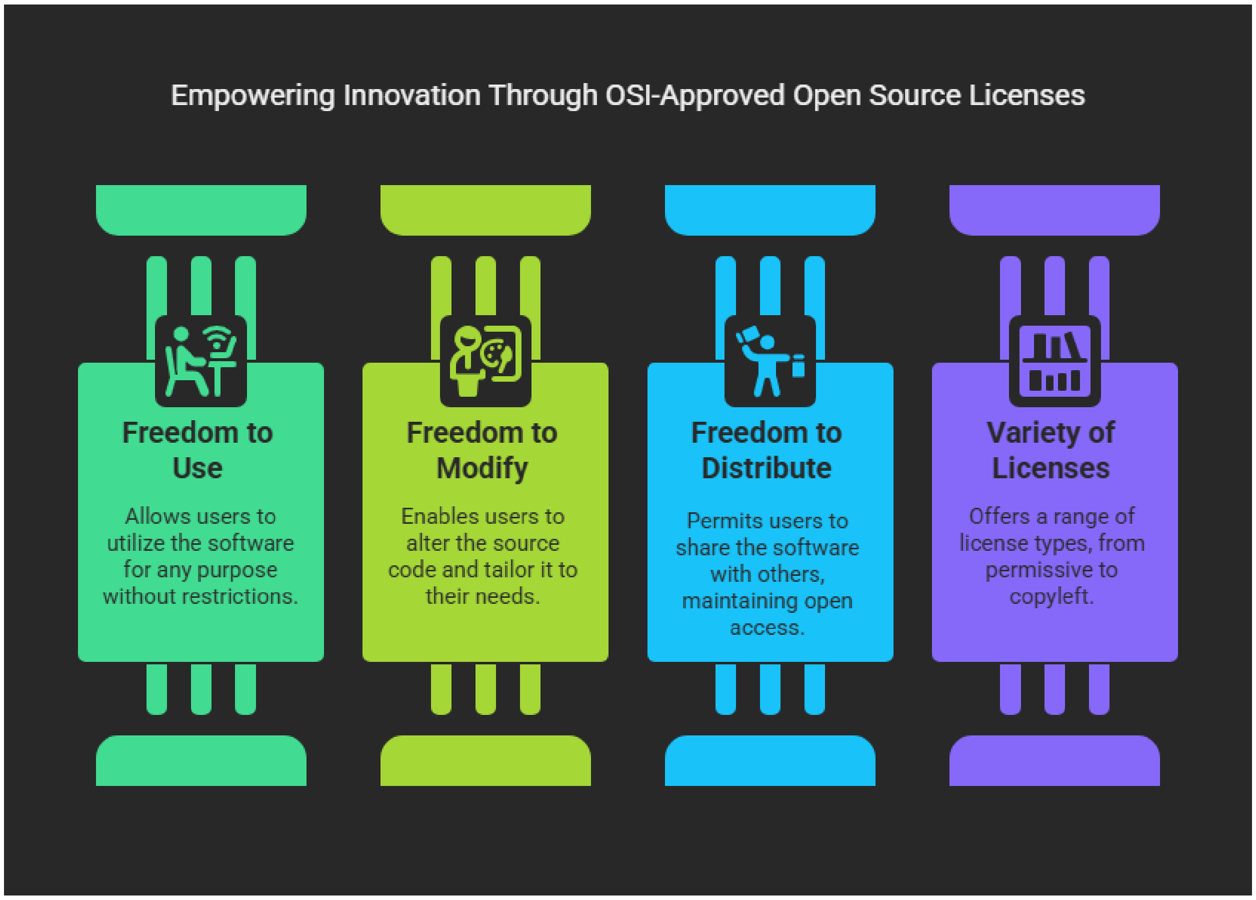
    These licenses give developers the freedom to use, modify, and distribute the software without many restrictions. Even if someone includes the code in a **closed-source (proprietary) project**, they don't have to share their changes.
    Examples: MIT License, Apache License, BSD License.

    2. Copyleft Licenses (More Restrictions)

    These licenses ensure that any modifications or derived versions of the software **must remain open-source**. If someone modifies and redistributes software under a **GPL license**, they are required to share their modified source code under the same open-source license.
    Examples: GNU General Public License (GPL), Lesser General Public License (LGPL).

Key Difference:

- **MIT License (Permissive)** → Developers can freely use the code in both **open-source and commercial projects** without any obligation to share their changes.
- **GPL License (Copyleft)** → Developers can modify and distribute the code, but any **derivative work must also be open-source**, ensuring that improvements remain accessible to everyone.

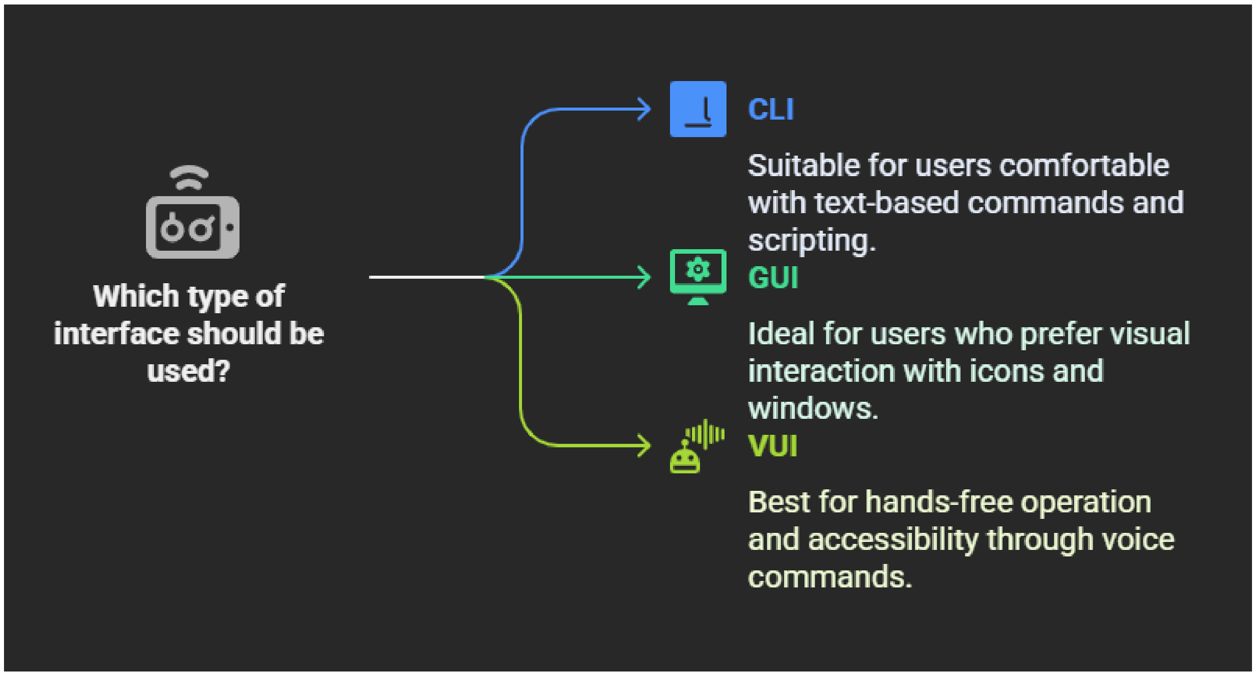

Empowering Innovation Through OSI-Approved Open Source Licenses
Freedom to Use
Allows users to utilize the software for any purpose without restrictions.
Freedom to Modify
Enables users to alter the source code and tailor it to their needs.
Freedom to Distribute
Permits users to share the software with others, maintaining open access.
Variety of Licenses
Offers a range of license types, from permissive to copyleft.

Although Linux is open source it more secure operating system

- There are n number of developers worldwide who are scrutinizing the code, checking the bugs and more often identifying and fixing it rapidly. The collaborative efforts ensure that the issues with the code are addressed promptly.

- As Linux is based on a permission model, that dictates who can access the file and who can execute the file. This kind of control allows administrator to restrict access to sensitive resources.
- Linux distribution provides you security updates that strengthen your system defenses and this protects against any emerging threats.
- **Community Audits**: Open source code allows global experts to continuously review and patch vulnerabilities.
- **Permissions System**: Linux has a strong user and permissions system, limiting unauthorized access.
- **Modularity**: The modular design of Linux allows for greater control over installed components, reducing security risks.
- **Lower Target**: Linux's smaller market share among casual users makes it a less frequent target for malware and attacks.

In OS we get two types of interfaces


Which type of interface should be used?
CLI
Suitable for users comfortable with text-based commands and scripting.
GUI
Ideal for users who prefer visual interaction with icons and windows.
VUI
Best for hands-free operation and accessibility through voice commands.

- **CLI - Command Line Interface,** this is a command based interface where a user types some command to execute the program.

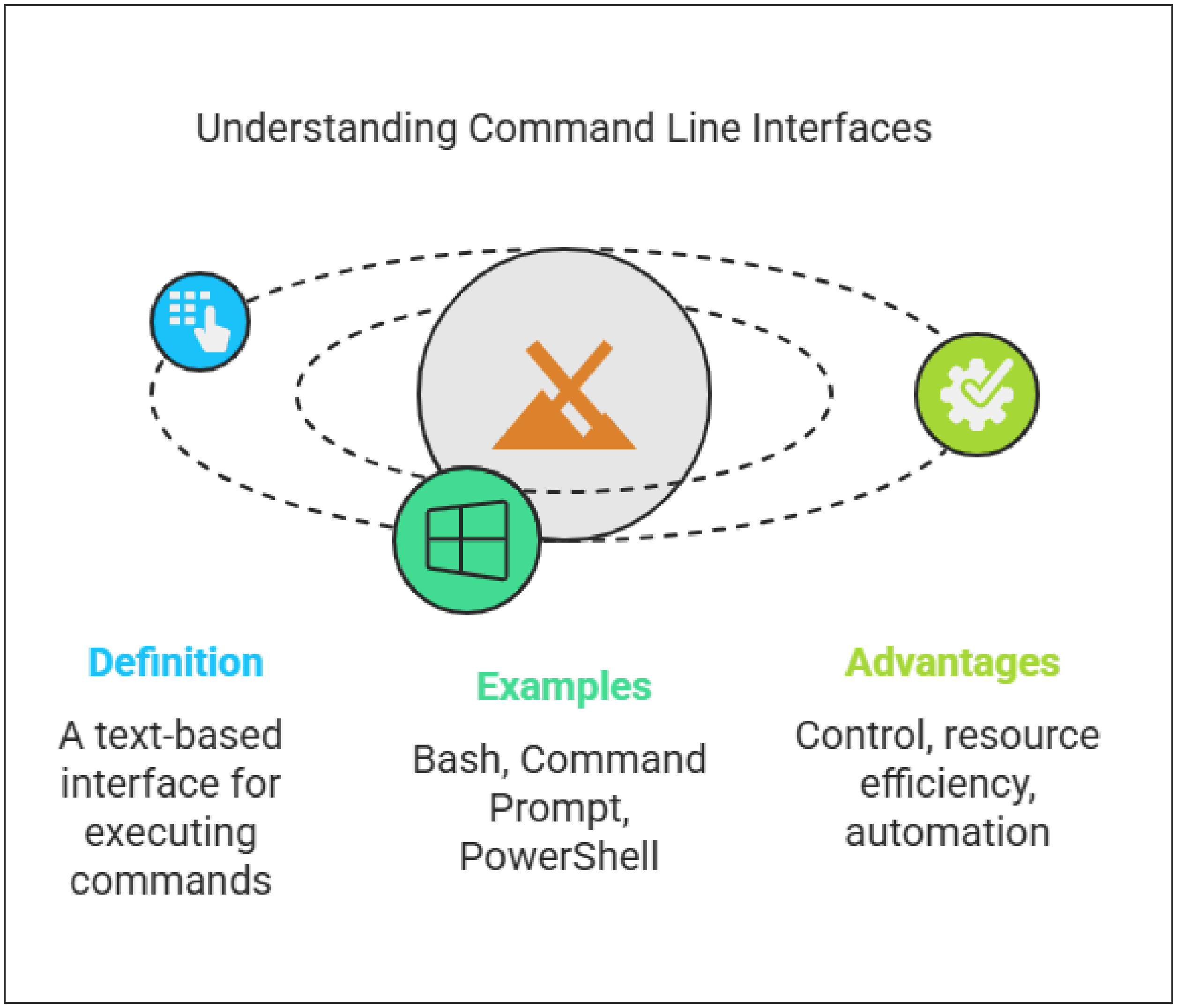
  Definition: A text-based interface where users type commands into a terminal or console window to perform specific tasks.

  Examples:

  - Bash in Linux and UNIX systems.
  - Command Prompt and PowerShell in Windows.

Advantages:

- Offers more control and flexibility for system management tasks.
- Consumes fewer system resources than GUIs.
- Powerful for automating tasks through scripting.


Understanding Command Line Interfaces
Definition
A text-based interface for executing commands
Examples
Bash, Command Prompt, PowerShell
Advantages
Control, resource efficiency, automation

- **GUI - Graphical User Interface**, this is an icon based OS where a user can drag or click on the icons.

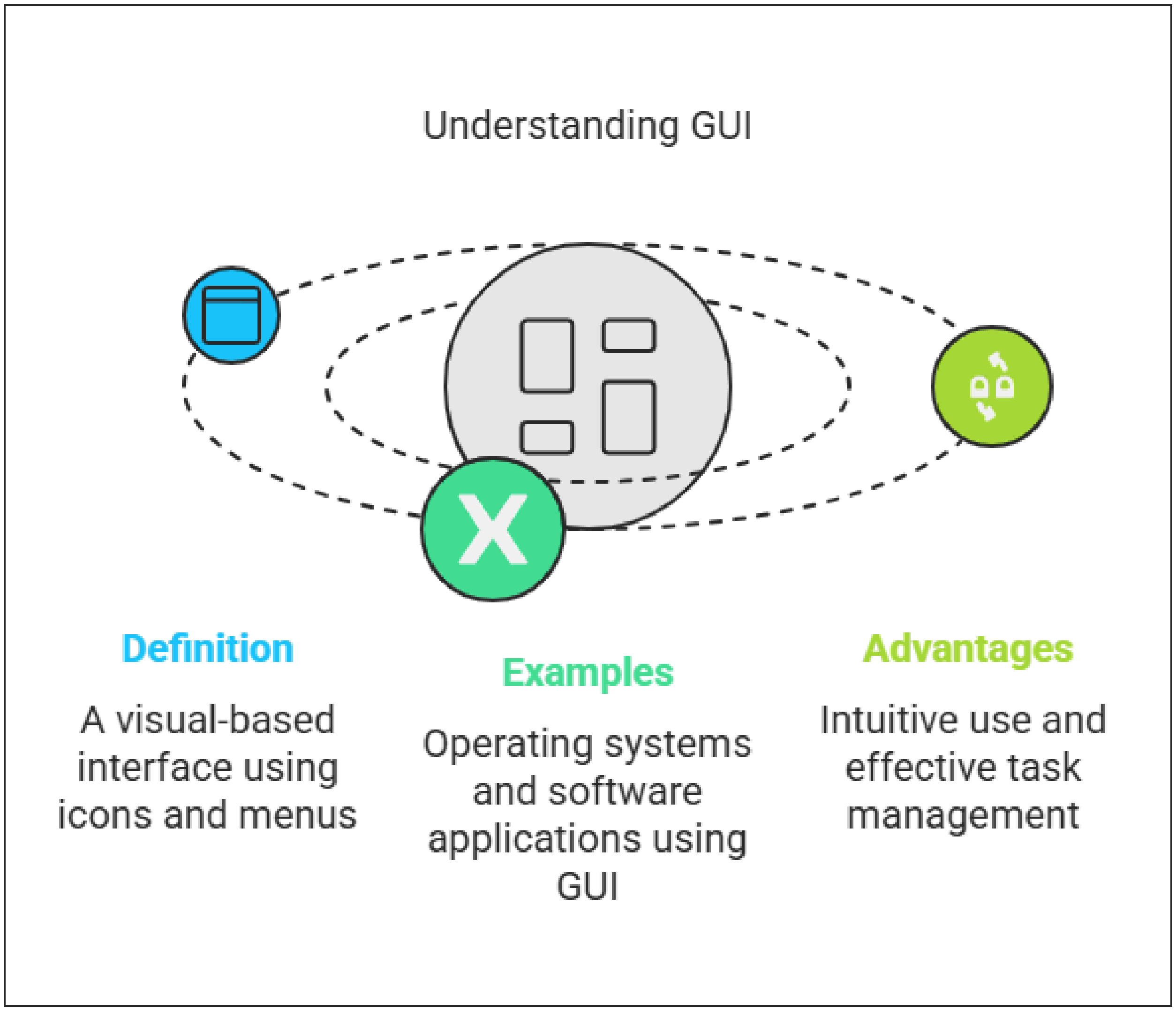
  Definition: A visual-based interface that allows users to interact with electronic devices using graphical icons, visual indicators, and menus instead of text-based command labels or text navigation.

  Examples:

- Operating systems like Windows and macOS use GUIs extensively.
- Software applications like Microsoft Word or Adobe Photoshop.

Advantages:

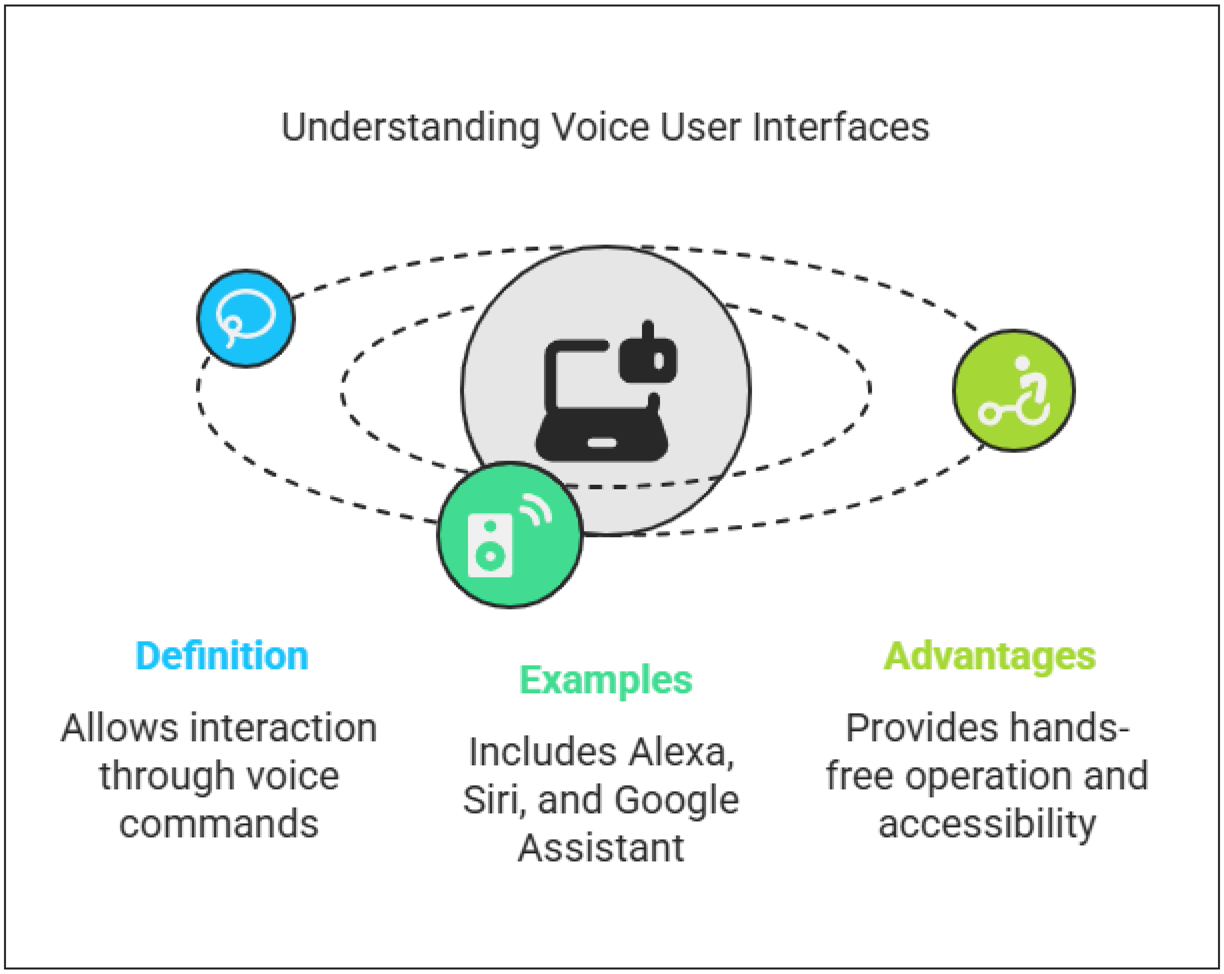
- Intuitive to use, especially for beginners.
- Enables effective management of multiple tasks simultaneously through multi-window and multitasking capabilities.


Understanding GUI
Definition
A visual-based interface using icons and menus
Examples
Operating systems and software applications using GUI
Advantages
Intuitive use and effective task management

Voice User Interface (VUI):

- **Definition:** Allows users to interact with a system through voice or speech commands.

- **Examples:**
  - Voice-activated assistants like Amazon Alexa, Apple Siri, and Google Assistant.
  - Voice-driven applications in smartphones and smart home devices.
- **Advantages:**
  - Hands-free operation makes it accessible and convenient.
  - Enhances accessibility for users with visual impairments or physical disabilities.
  - Useful in situations where using hands is impractical or unsafe, such as while driving.

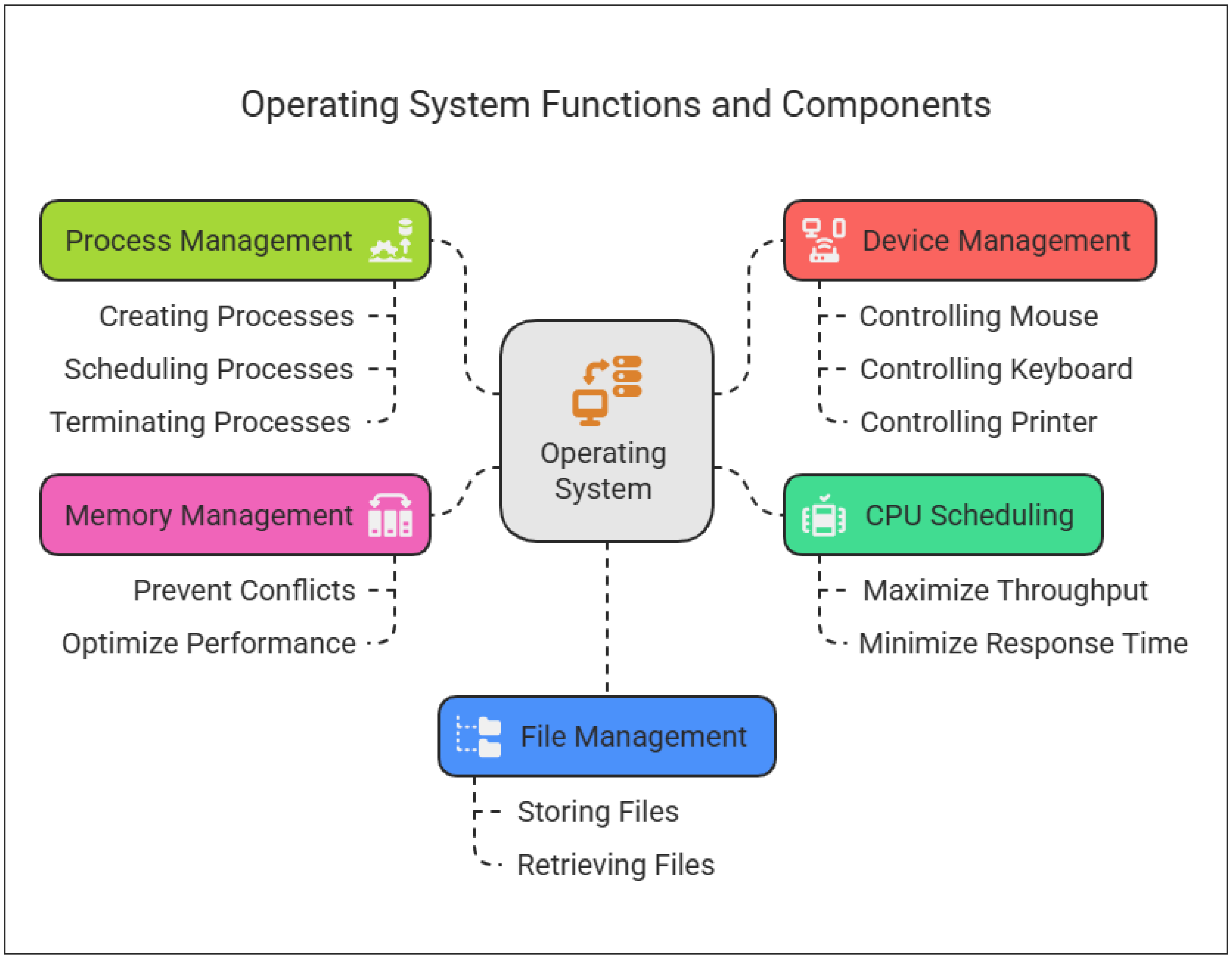

Understanding Voice User Interfaces
Definition
Allows interaction through voice commands
Examples
Includes Alexa, Siri, and Google Assistant
Advantages
Provides hands-free operation and accessibility

OS does

- **CPU Scheduling** - This determines the order in which processes are executed by the CPU, the primary goal of CPU scheduling is to maximize system throughput and minimize response time and to do allocation of CPU time among the competing processes.

- **Process Management** - This is a core component of OS, which is responsible for creating, scheduling, termination and managing processes and threads.
- **Memory Management** - This involves the efficient usage of computers memory, it ensures that programs and processes have access to memory they need while preventing conflict and optimizing system performance.
- **File Management** - This functionality involves, organizing, storing, retrieving and manipulation of files and directories on the storage devices such as Hard Drive, SSD or any network storage.
- **Device Management** - This involves controlling and coordinating access to hardware devices connected to the computer system, the devices which can be included are mouse, keyboard, printer, scanner or any other I/O devices.

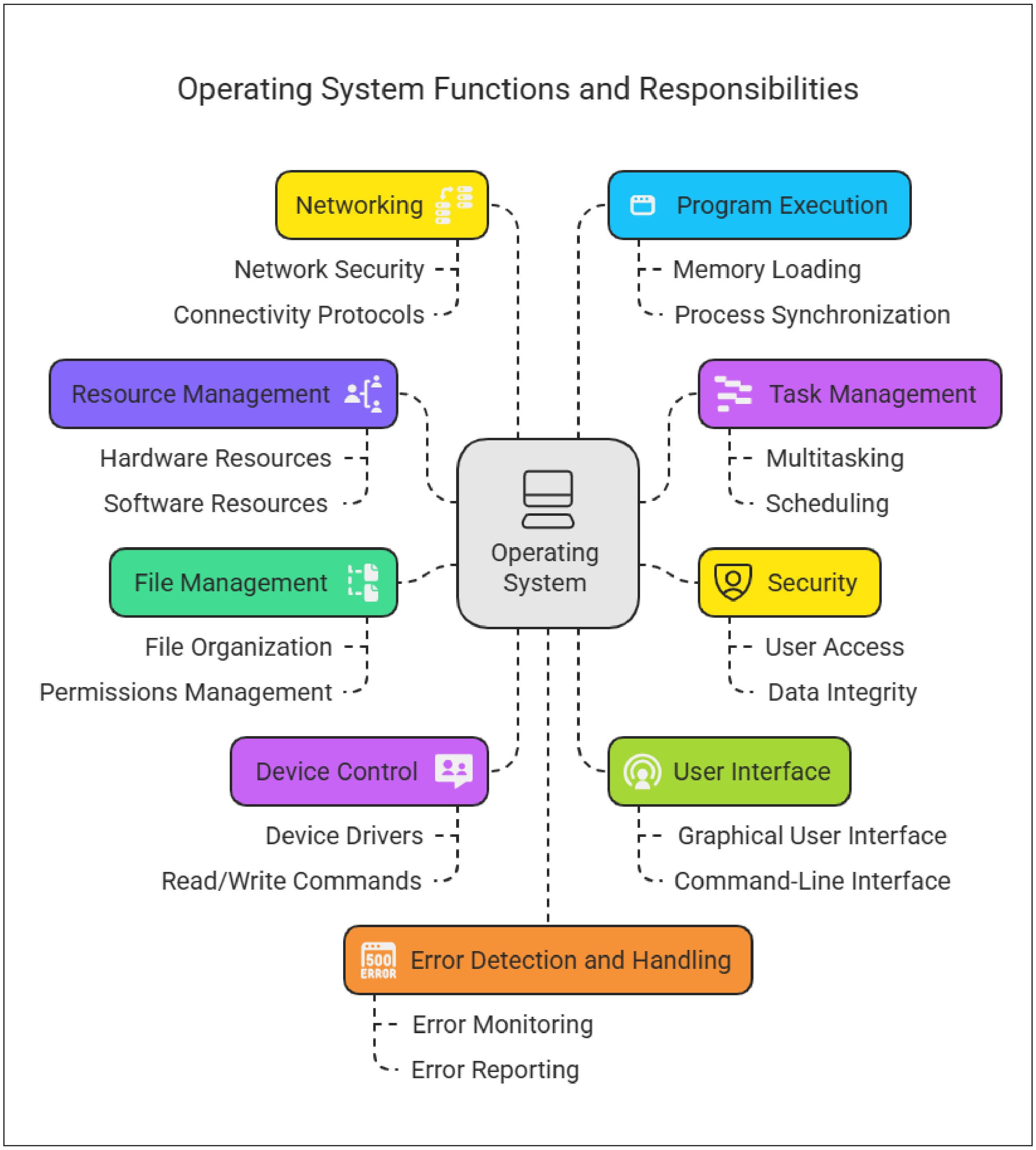
The contributions done by developers/ programmers are usually done on Linux Kernel such as to add features, finding and fixing bugs.



Here are the key functions of an operating system:

1. **Resource Management:**
   - **Hardware Resources:** The OS manages and allocates CPU time, memory space, disk space, and I/O devices. It ensures that different programs and users running concurrently do not interfere with each other.
   - **Software Resources:** The OS handles the execution of programs, provides necessary services to applications, and efficiently manages system resources among all running applications.
2. **Task Management:**
   - The OS is responsible for multitasking, which involves managing and scheduling the execution of multiple tasks. It keeps track of processor status, program status, and system functions.
3. **Security:**
   - An operating system provides a secure environment within which system resources and user data are protected. This includes managing user access to programs and data, protecting against unauthorized access, and ensuring data integrity.
4. **File Management:**
   - The OS manages files on various storage devices, organizes files in directories, and provides functions to create, move, delete, and access files. It also manages permissions and access rights for files and directories.
5. **Device Control:**
   - The OS manages all device communication via respective drivers. It translates the user's read/write commands into device-specific calls, acting as a communicator between the hardware and the software.
6. **User Interface:**
   - Operating systems provide interfaces through which users interact with computers. This can be a graphical user interface (GUI) like in Windows or macOS, a command-line interface (CLI) such as in Linux, or touch interfaces as seen in mobile operating systems like Android and iOS.
7. **Networking:**
   - The OS handles networking capabilities, allowing different computing devices to communicate and share resources over a network. It manages the data exchange, network security, and connectivity protocols.
8. **Program Execution:**
   - The OS loads programs into memory, sets up the execution stack, and handles the execution of applications. It provides mechanisms for process synchronization and inter-process communication.
9. **Error Detection and Handling:**

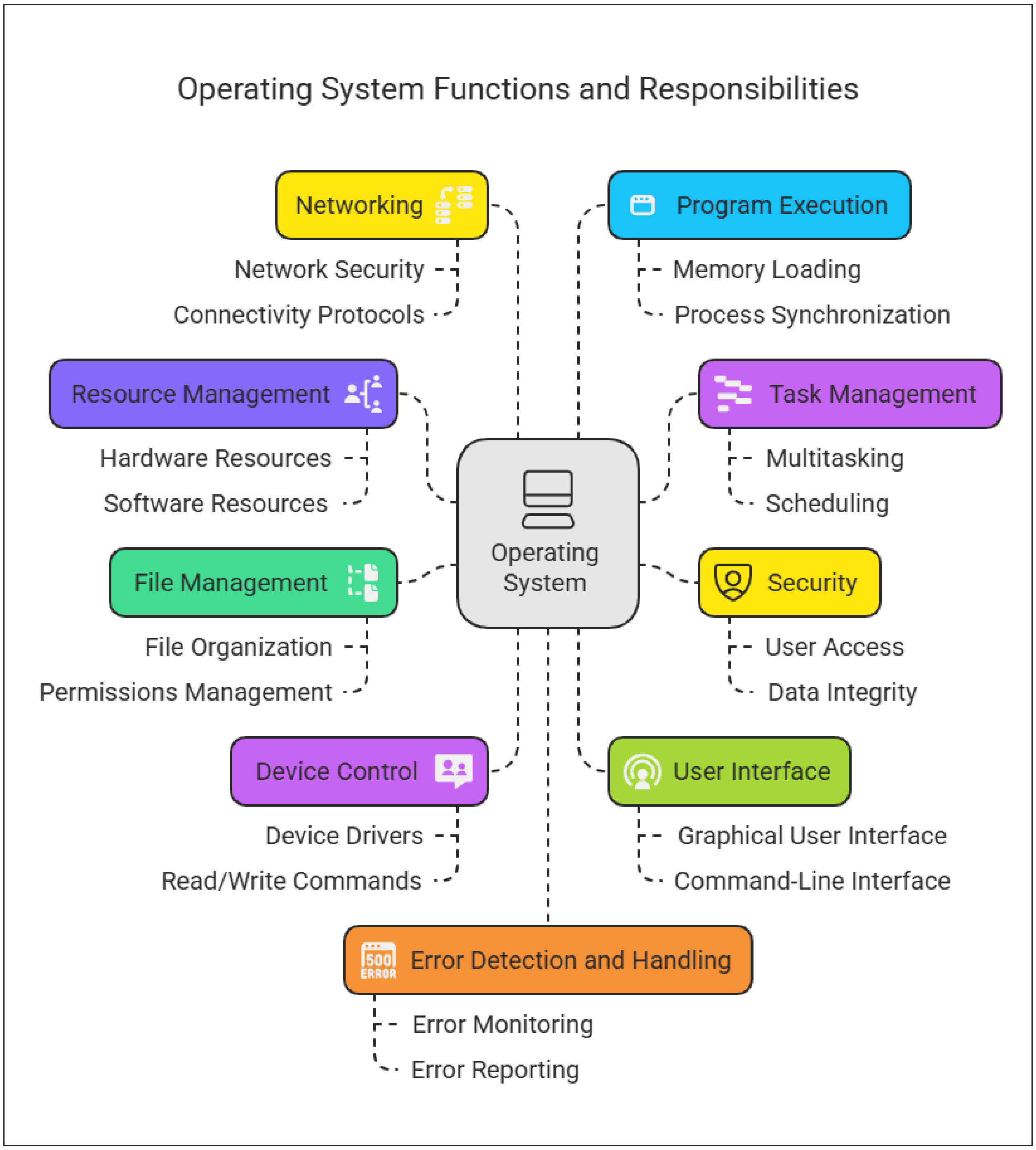
- The operating system continuously monitors the system for possible errors. It provides error-reporting methods to the users and takes appropriate actions to recover from system errors.



Here are the key functions of an operating system:

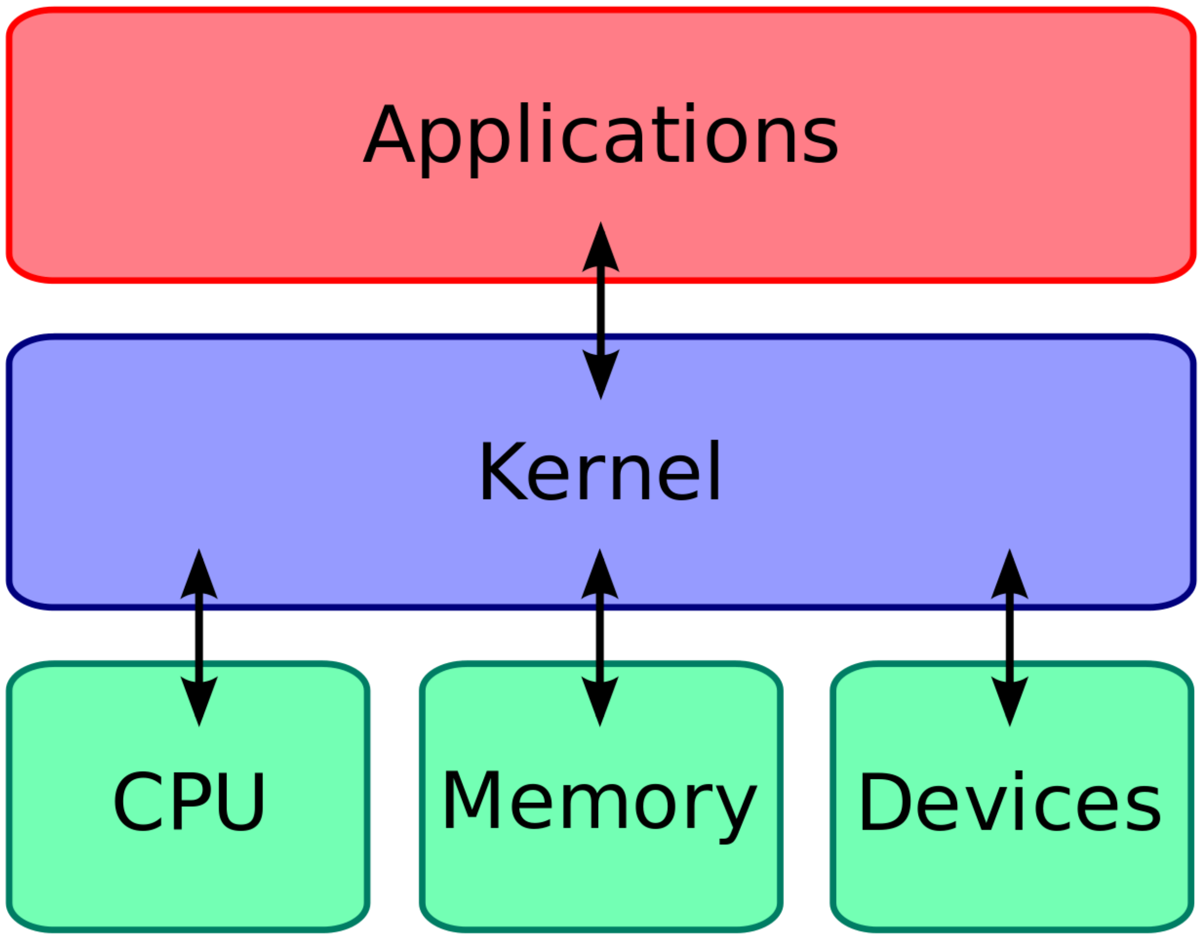
1. **Resource Management:**
    - **Hardware Resources:** The OS manages and allocates CPU time, memory space, disk space, and I/O devices. It ensures that different programs and users running concurrently do not interfere with each other.
    - **Software Resources:** The OS handles the execution of programs, provides necessary services to applications, and efficiently manages system resources among all running applications.
2. **Task Management:**
    - The OS is responsible for multitasking, which involves managing and scheduling the execution of multiple tasks. It keeps track of processor status, program status, and system functions.
3. **Security:**
    - An operating system provides a secure environment within which system resources and user data are protected. This includes managing user access to programs and data, protecting against unauthorized access, and ensuring data integrity.
4. **File Management:**
    - The OS manages files on various storage devices, organizes files in directories, and provides functions to create, move, delete, and access files. It also manages permissions and access rights for files and directories.
5. **Device Control:**
    - The OS manages all device communication via respective drivers. It translates the user's read/write commands into device-specific calls, acting as a communicator between the hardware and the software.
6. **User Interface:**
    - Operating systems provide interfaces through which users interact with computers. This can be a graphical user interface (GUI) like in Windows or macOS, a command-line interface (CLI) such as in Linux, or touch interfaces as seen in mobile operating systems like Android and iOS.
7. **Networking:**
    - The OS handles networking capabilities, allowing different computing devices to communicate and share resources over a network. It manages the data exchange, network security, and connectivity protocols.
8. **Program Execution:**
    - The OS loads programs into memory, sets up the execution stack, and handles the execution of applications. It provides mechanisms for process synchronization and inter-process communication.
9. **Error Detection and Handling:**

- The operating system continuously monitors the system for possible errors. It provides error-reporting methods to the users and takes appropriate actions to recover from system errors.



Kernel

This is a core part to **interact between hardware and software.** It is a core part of OS, that interacts directly with hardware
Kernel is responsible for **managing CPU, memory, I/O devices that handles system calls, interrupt or any exceptions.**



Memory Management:

- The kernel controls and manages system memory. It decides how much memory to allocate to processes and when to reclaim it. It also handles memory swapping between the physical RAM and the hard disk to optimize system performance.

Process Management:

- The kernel is responsible for process scheduling and lifecycle management. It controls process creation, execution, pausing, resuming, and termination. It manages CPU resources by deciding which processes get CPU time, in what order, and for how long.

Device Drivers:

- Device drivers are programs that allow the kernel to interact with hardware devices, like keyboards, mice, disk drives, network adapters, etc. The kernel uses these drivers to abstract the hardware details away from the software, providing a consistent interface to peripherals.

System Calls and Security:

- System calls are the mechanisms by which programs request services from the kernel, such as creating processes, handling files, or communicating over a network. The kernel also enforces security policies and isolates different system processes to prevent unauthorized access to data.

Input/Output Management:

- The kernel manages I/O operations and provides a buffer management system to control the flow of data to and from hardware devices. It optimizes these operations to improve system efficiency and responsiveness.

Networking:

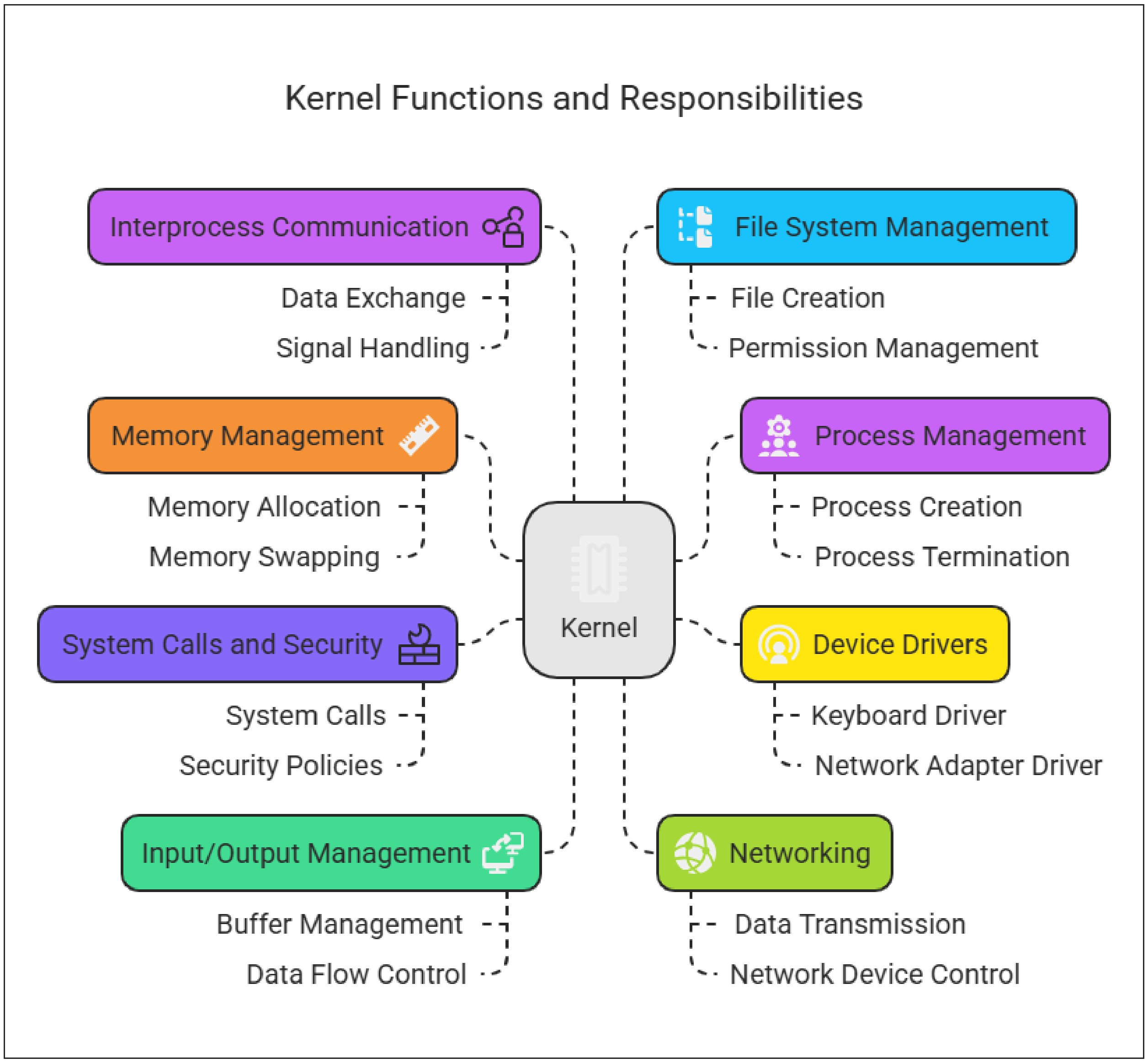
- The kernel handles the networking stack that allows the system to communicate over various network protocols, handling both the data transmission and the network device control.

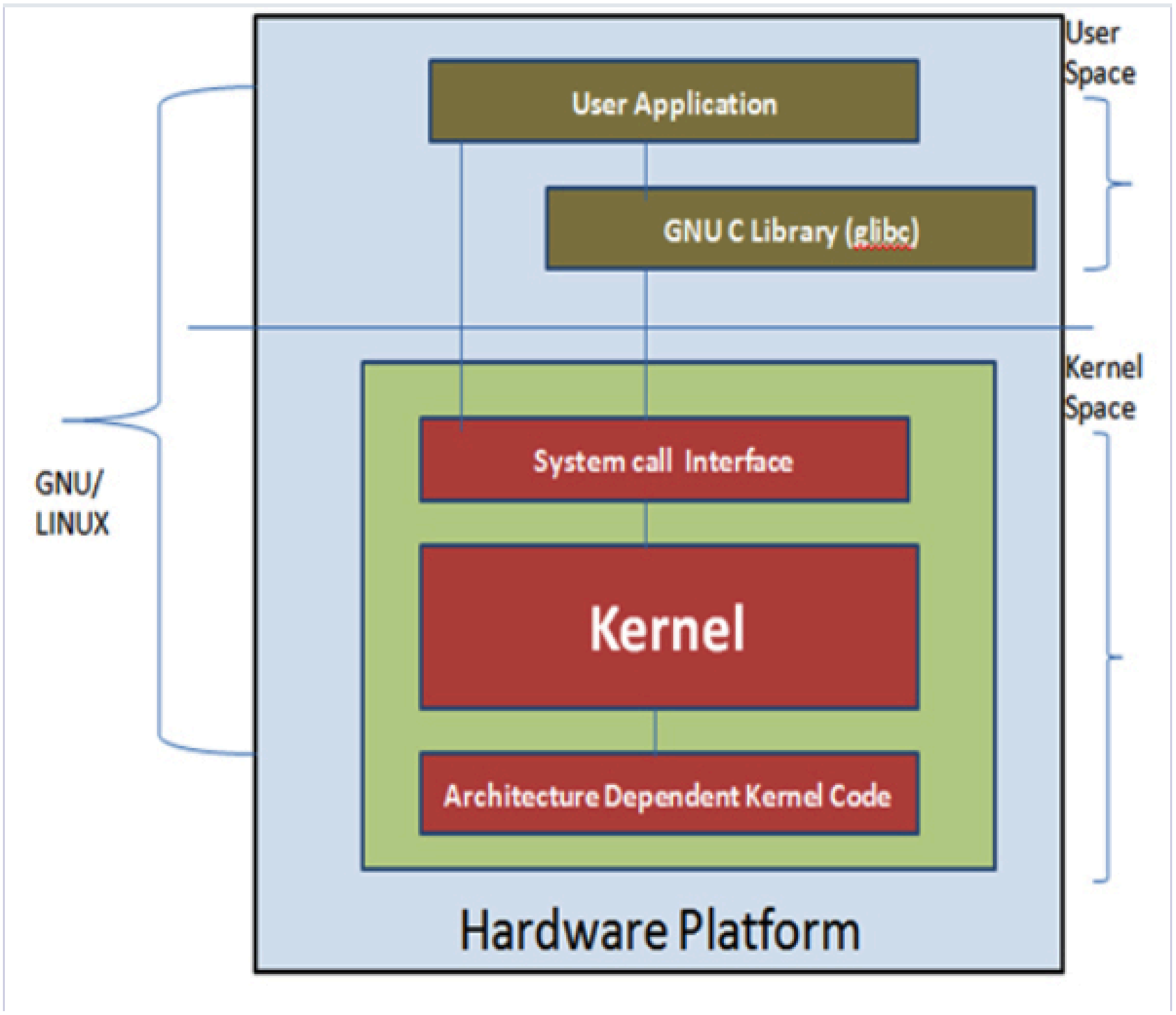
Interprocess Communication:

- The kernel facilitates communication between processes, allowing them to exchange data and signals securely and efficiently. This is crucial for applications that need to work together closely to perform their tasks.

File System Management:

- It manages file systems, which organize and store the files on various storage devices. The kernel provides functions to create, delete, read, and write files and to manage permissions and security settings.


Kernel Functions and Responsibilities
Kernel
Interprocess Communication
Data Exchange
Signal Handling
File System Management
File Creation
Permission Management
Memory Management
Memory Allocation
Memory Swapping
Process Management
Process Creation
Process Termination
System Calls and Security
System Calls
Security Policies
Device Drivers
Keyboard Driver
Network Adapter Driver
Input/Output Management
Buffer Management
Data Flow Control
Networking
Data Transmission
Network Device Control



User Space

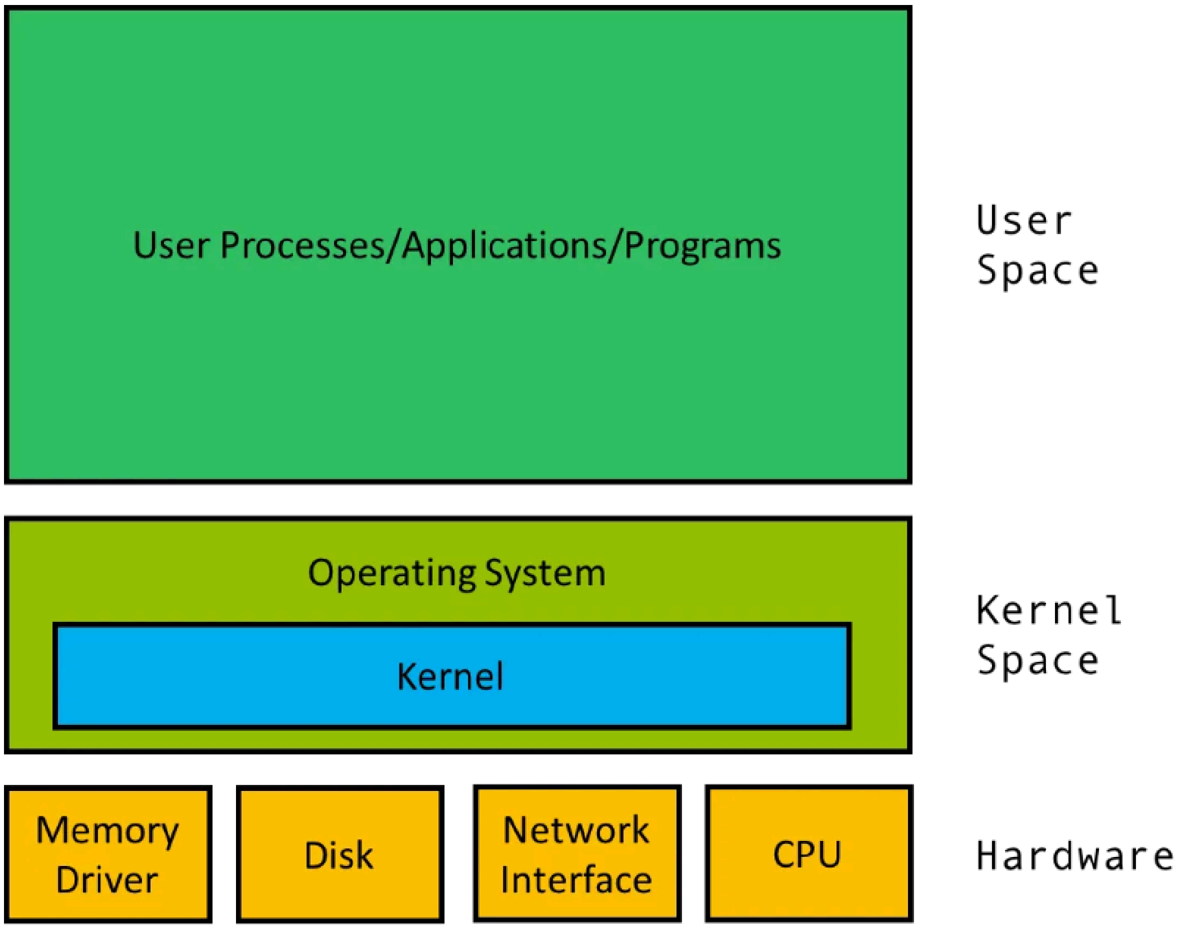
- **Definition:** User space (or user mode) is the region of **system memory where user applications and processes run.** This space is protected and isolated from the kernel space, meaning that **code running here cannot directly interact with the hardware or interfere with kernel operations.**

- **Characteristics:**
  - **Security and Stability:** Errors or crashes in user space do not affect the core of the operating system, which helps maintain overall system stability and security.
  - **Limited Access:** Applications in user space have limited access to system hardware and resources, mediated by the kernel through system calls.

    - **Environment:** User space provides the environment where third-party and user applications run, such as web browsers, games, and office software.

Kernel Space

- **Definition:** Kernel space (or kernel mode) is **where the kernel (the core part of the operating system) operates.** It has **unrestricted access** to the hardware and all system resources.
- **Characteristics:**
    - **Full Access:** Code running in kernel space can directly interact with hardware devices, manage memory, and perform critical system tasks.
    - **Performance:** Operations in kernel space are faster than those in user space because there's no need for the context switch typically required to access hardware.
    - **Risk:** Errors or crashes in kernel space can lead to system failures or security issues since the kernel has complete control over the system.



Interaction Between User Space and Kernel Space

- **System Calls:** The primary method of **interaction between user space and kernel space is through system calls.** Applications in **user space request services from the kernel (like file access, starting a new process, or network communication) using these calls.**

- **APIs and Libraries:** User applications typically **use APIs and libraries** (like the GNU C Library in Linux) that **abstract these system calls into more manageable functions for programming.**

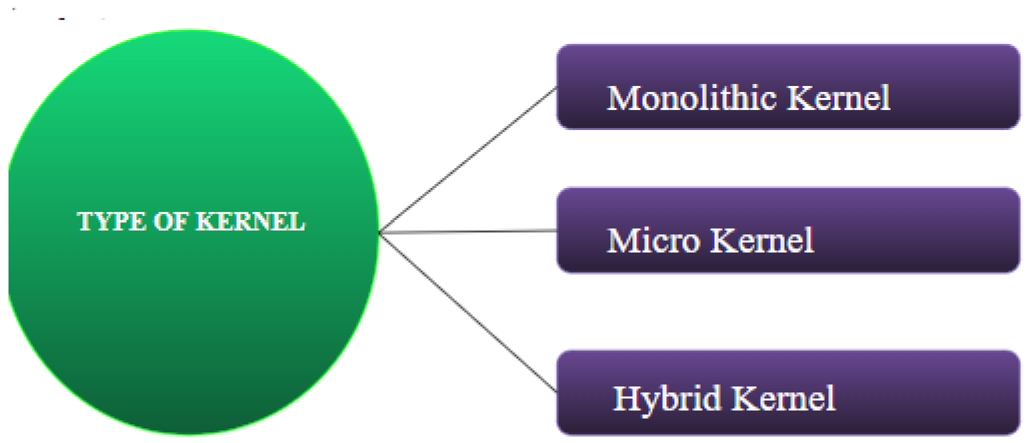
The separation between user space and kernel space is fundamental for system security and reliability. It ensures that malfunctioning programs or malicious software in user space cannot corrupt the kernel or other running applications, thereby protecting the integrity and stability of the operating system.

Kernel loads first into memory, when an OS is loaded and remains into memory until OS is shut down. The kernel has a process table that keeps track of all active processes.Kernel is written in C language.

The difference between Kernel and OS:

Kernel	OS
It is a core part of the OS.	It is an interface between hardware & software.
It is a software / low level program that manages hardware resources including memory, CPU & I/O devices, interrupt and exceptions.	OS not only includes Kernel but various system utilities, libraries and various user interfaces.
It is responsible for doing tasks like process management, memory management , scheduling, hardware management.	Is an interface that helps to interact between user interface.
It is a central point of control and resource management in OS.	It controls, manages system configuration files or tools.

Types of Kernel
There are 3 types of kernel



Monolithic Kernel

- These types of kernel where all OS services **operate in kernel space**, this is one of the oldest types of kernel where the **entire OS is composed in a single large executable file that runs on kernel mode.**
- This is more efficient kernel as it does not have to switch between different address spaces, when executing different tasks. This makes the process execution faster as there is no separate memory space for user and kernel.
- All operating system services run in kernel space, which can lead to better performance but may result in less stability due to the complexity of the kernel.
- Examples: **Linux, Unix.**

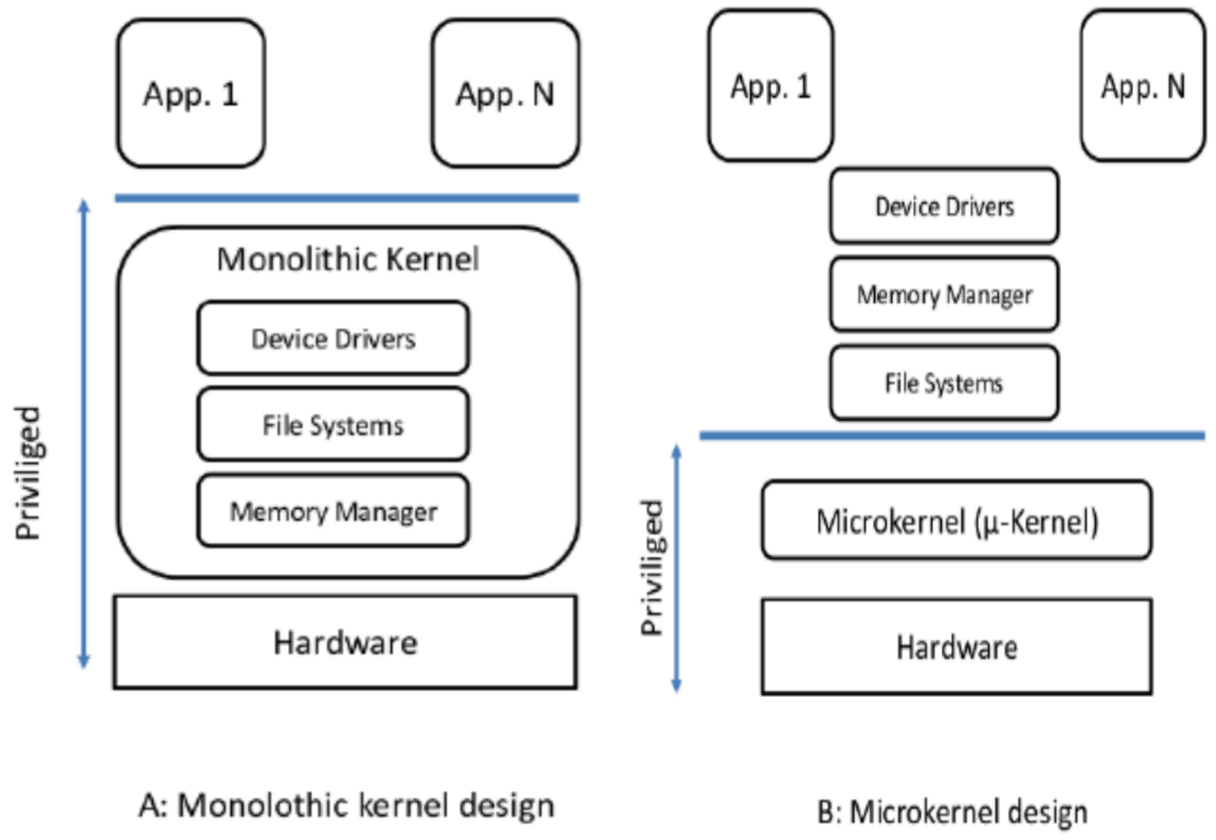
Advantages of **Monolithic Kernel**

- It is **faster because it operates from kernel space.**
- It is more secure than any other kernel because it can be protected easily from malicious code.

Disadvantages

- As the entire OS is composed of a single binary executable file that runs on kernel mode so if anything gets corrupted the whole system gets corrupted.

Examples : **UNIX, LINUX, Open VMS**



Micro-Kernel

It is a type of kernel which works between two spaces i.e, **user space and kernel space.** Micro Kernel works on modularity i.e system services, device drivers run on separate space i.e user space and all the process management, memory management will work under kernel space.
The kernel is minimized, and most services run in user space. This approach can improve stability and security but might incur a performance penalty.
Examples: MINIX, QNX.

Advantages of Micro-Kernel

It is more stable than Monolithic Kernel, if any services get affected or corrupted it doesn't affect the whole system.
Examples: **Blackberry QNX, MiNIX**

Disadvantages

- As it is modular it requires more communication and synchronization between different spaces.
- It uses more system resources such as CPU, Memory than monolithic kernel as it has to maintain synchronization.

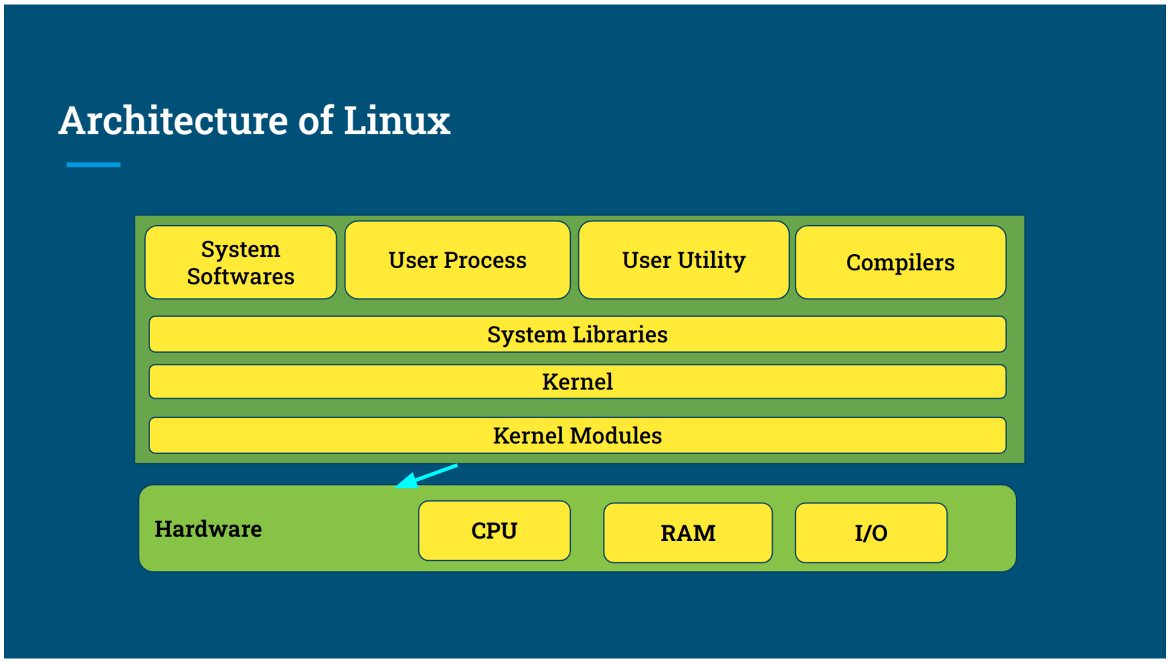
Hybrid Kernel
This is a combination of monolithic and micro kernel, it has **speed of monolithic kernel** while it has modularity and stability of micro kernel. All the essential services like process management, file management, CPU management, CPU scheduling, interrupts are kept in Kernel mode while services like device drivers are placed under User mode.
Combines aspects of both monolithic and microkernel architectures, trying to balance performance and modularity.
Examples: **Windows NT, macOS.**

Examples : MS Windows, MAC OS, Netware.

Difference between Micro and Monolithic Kernel

Micro Kernel	**Monolithic Kernel**
User services and kernel services are kept in separate spaces	Both kernel and user services are kept in same space
Smaller than Monolithic Kernel	They are larger than micro-kernel
It is easier to add new functionality as it works in modularity	It is difficult to add new functionality as everything is there in one space
Failure of one component doesn't affect the whole system	Failure of any one component affects the whole system
Execution speed is slower	Execution speed is faster
Ex : Blackberry QNX	Ex : UNIX

Architecture of linux:



Hardware

CPU - It is a central processing unit which is responsible for executing and controlling any process or task on the system.
RAM - It is said as Random Access Memory that helps to **store data temporarily**
I/O Devices - Input devices are those devices which help you to enter data or commands into the system for example - Keyboard, Mouse, Scanner.
Output devices are those devices which help you to display the output from the system for example - Monitor, Printer, Speaker

Kernel Modules

It is also known as **device driver** or also referred to as **loadable kernel** .
It is called as loadable and unloadable kernel because a piece of code can dynamically be loaded into linux kernel without rebooting your system. These are basically written into C / Assembly language
Examples - Graphic Card, Network Adapter.
Kernel modules are stored in files with extension of '.ko' and are loaded into kernel using 'INSMOD' or using 'MODPROBE' commands.
The main subdirectory for kernel modules is found under '/lib/modules'.

Kernel

It is a core part of the OS that exists between the hardware and software, whenever a system boots kernel is the first part that gets memory and exists till the system shuts down. It is responsible for managing CPU memory, I/O devices that handles system calls, interrupts and any exceptions.

System Libraries

They are **prewritten codes that provide functionality to programs and applications running on OS**. It is a set of functions for user level applications to interact with OS Kernel.
Examples - **C Standard Library (LibC)** this library is used to perform operations like I/O, Memory management and string manipulation,
Math Library (LibM) provides a wide range of **mathematical functions that includes Trigonometry, Logs.**
Database Connectivity (Libpq, LibMySQLClient) These libraries are used to interact with various databases on your system.
System Call Wrappers this library is used to make system call to interact between user level program and kernel
Example - **printf**

User Utility

It provides various functionality tools to user and system administrator to interact and perform various tasks on Linux
Example - **Shell** (Bash, CSH, ZSH), Shell command is used to interact with system by entering some commands
Text Editor this is used to create and edit a text file.
Example - Vim, Nano, Touch

File & Directory Management, this helps you to manage file and directories including managing files by commands like **cp, rm, ls, mkdir**

P**rocess Management** allows you to view and manage the process with commands like **ps,kill,top.**

Package Manager this helps you to install and update system software, Examples - **apt, yum, pacman.**

File Compression this helps you to compress and decompress any file. Examples - Zip, Unzip, Tar

Remote Desktop this utility allows you to remotely access any other system Examples - SSH PUTTY

Networking Tools this helps you check the network using ping,ss commands

User Processes

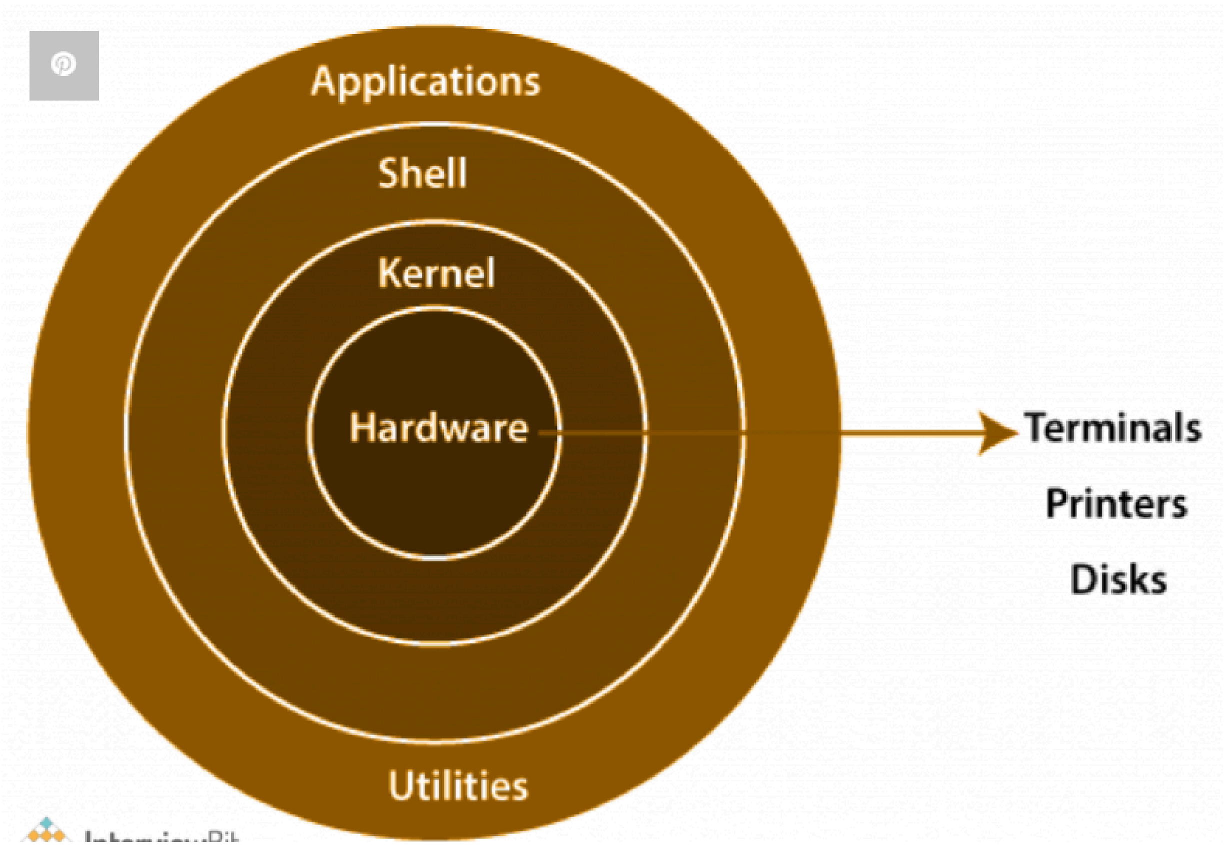
These are those programs that run in **user space** which are protected and isolated areas of memory and **execute in separate space**. They are initiated by user to interact with kernel and system resource through system call following are the different modules through which user processes work :

- User Application - User processes consist of user application which can either be graphical application or command line utility to interact with the system.
- User Space - User processes that operate within user space, it is a protected environment where processes are given limited and direct access to hardware resources.
- Process Creation - Processes are created by users from another parent process fork() ,exec() are system calls that are used to create and replace any processes.
- Resource Management - Processes interact with kernel to manage various resources like CPU Memory, I/O.
- System Calls - User processes interact with kernel by making system calls these calls request services / operation from kernel, few system calls are like creating new processes, allocating memory, doing I/O operations, creating new files and reading files.

System Software

It refers to collection of software and utilities that are required to manage operation on OS Examples -

- **Init -** System is responsible for initializing, managing system services and handling system shutdown process, this is responsible for initializing OS after it loads into memory.
- **Bootloader** - This is responsible for loading OS into memory, it is the first program that is executed whenever you turn on the system.



User space this is a layer where all the user level applications such as GUI, CLI falls. This layer helps to interact with the kernel through system calls to perform various tasks.

Kernel space provides an interface between hardware and software that manages system resources.