Operating Systems - CSEN 602

Module 1: *Introduction to Operating Systems*

Lecture 02: *Operating System Functionalities & Design Aspects*

Dr. Eng. Catherine M. Elias

catherine.elias@guc.edu.eg

Lecturer, Computer Science and Engineering,
Faculty of Media Engineering and Technology, German University in Cairo

Outline

- OS Elements
- OS Design Principle
- OS Components
- Kernel Space vs. User Space
- OS Protection Boundary
- System Call Flowchart
- Crossing the OS Boundary

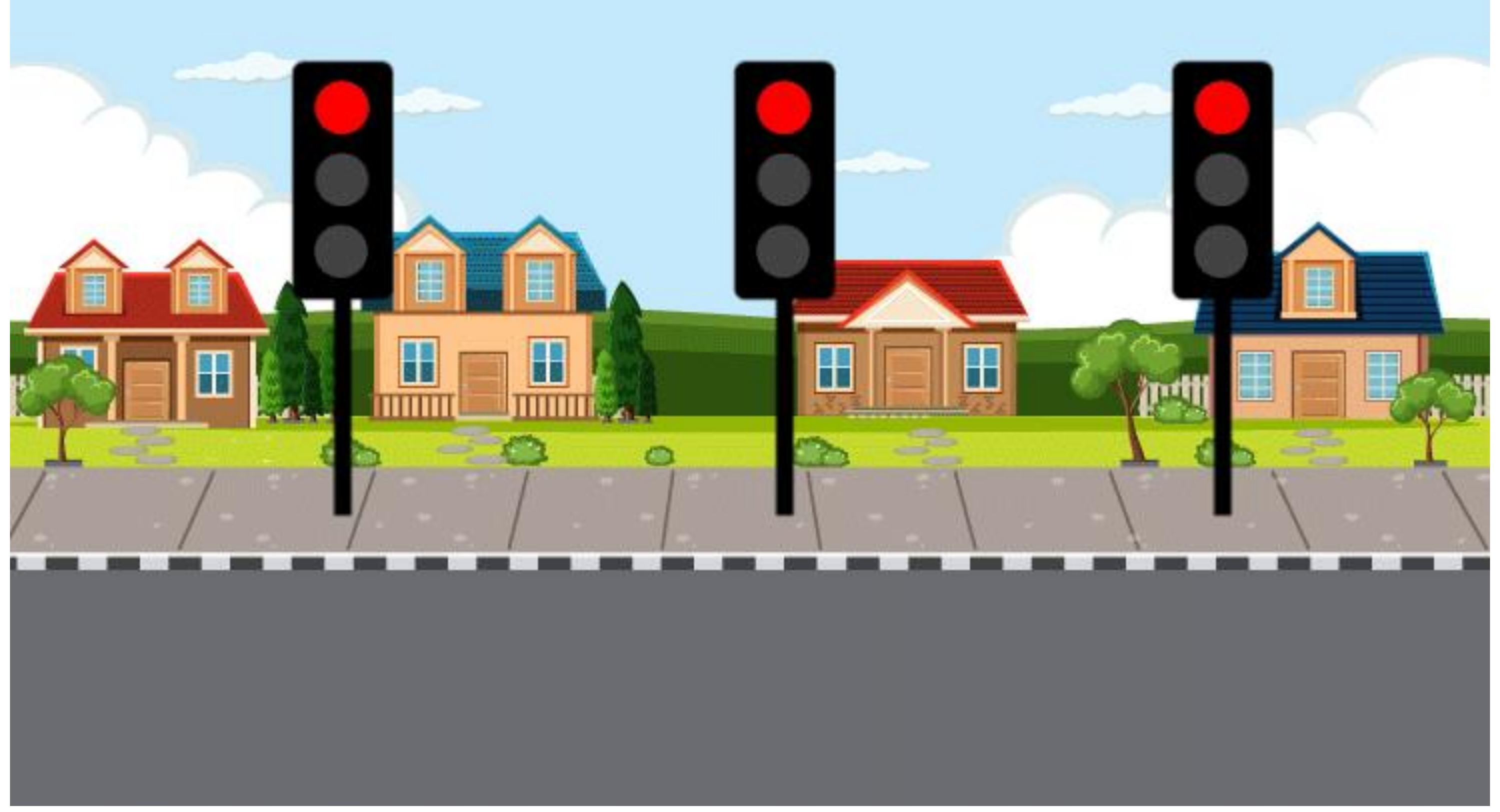
OS Elements

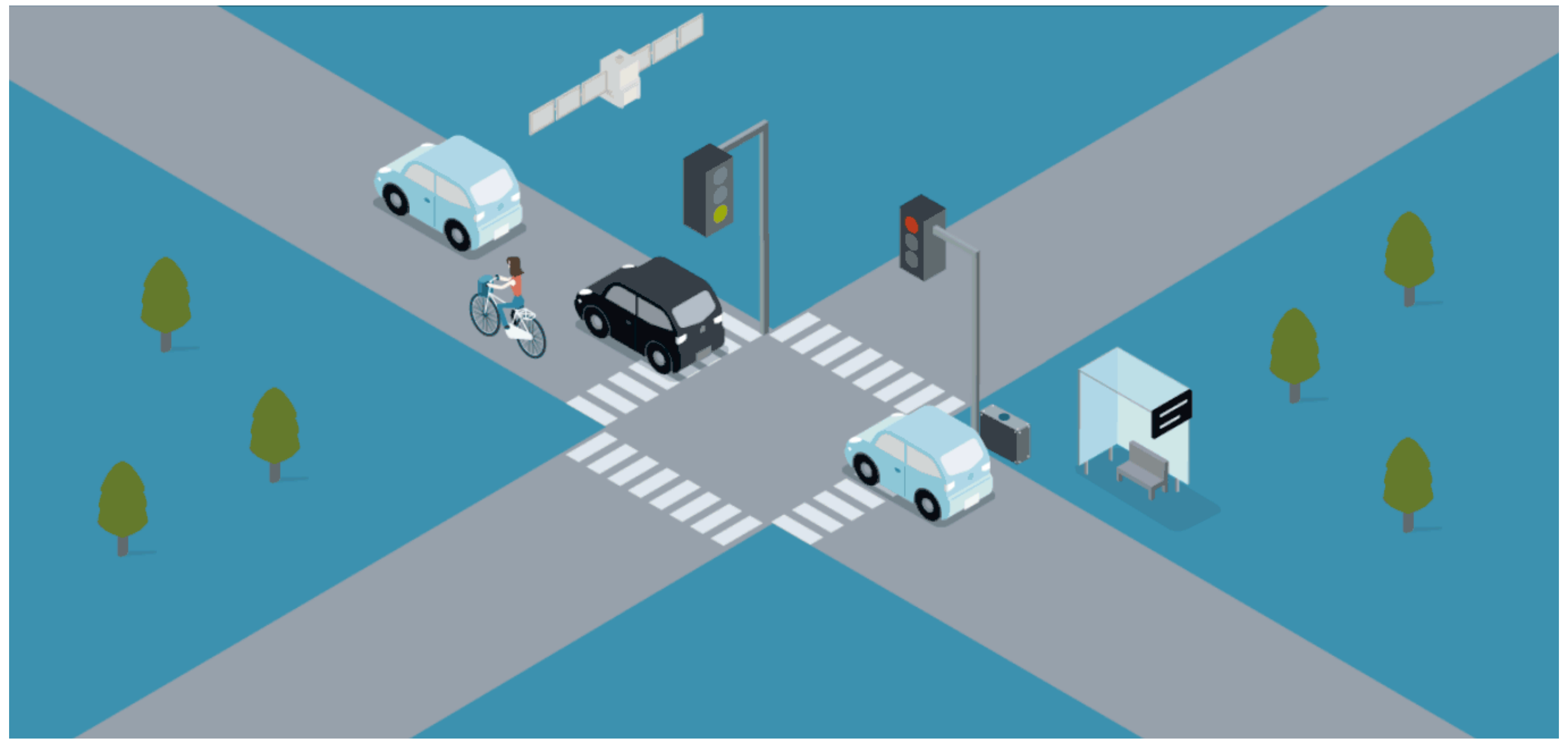
The Key OS Elements

Abstractions

Mechanisms

Policies





OS Elements

GUC German University in Cairo
الجامعة الألمانية بالقاهرة

The Key OS Elements

Abstractions

OS provide abstraction to the underlying hardware, allowing programmers and users to interact with the system without needing to understand the intricate details of the hardware. This abstraction simplifies development and enhances portability.

Examples: file systems (File, Socket), memory management (RAM, ROM, DISK), and process management (Process, Thread).

Mechanisms

Mechanisms are the implementation details that allow the operating system to provide its services and enforce its policies. These mechanisms include algorithms, data structures, and other low-level components that perform tasks.

Examples: scheduling processes (Create, Schedule), managing memory (Allocate), controlling access to resources (Lock, Unlock), and handling interrupts (Open, write).

Policies

Policies determine how the system behaves and how resources are allocated. They define rules and guidelines for resource management, security, and system behavior. Operating systems implement policies on top of mechanisms to enforce desired behavior.

Examples: Round-Robin Scheduling Policy, Least Frequently Used Policy

OS Design Principles

GUC German University in Cairo
الجامعة الألمانية بالقاهرة

There are two key operating system design principles that should be considered…

1. **Separation of mechanism and policy**
    - This principle emphasizes the importance of **decoupling** the **implementation details** (mechanisms) from the **rules and guidelines** (policies) that govern the behavior of the OS.
    - By implementing **flexible mechanisms**, OS can support a **wide range of policies** without having to modify the underlying mechanisms.
    - This separation enhances the **adaptability** and **customizability** of the OS, allowing it to meet diverse requirements and accommodate changes in policy without extensive reengineering.

OS Design Principles

GUC German University in Cairo
الجامعة الألمانية بالقاهرة

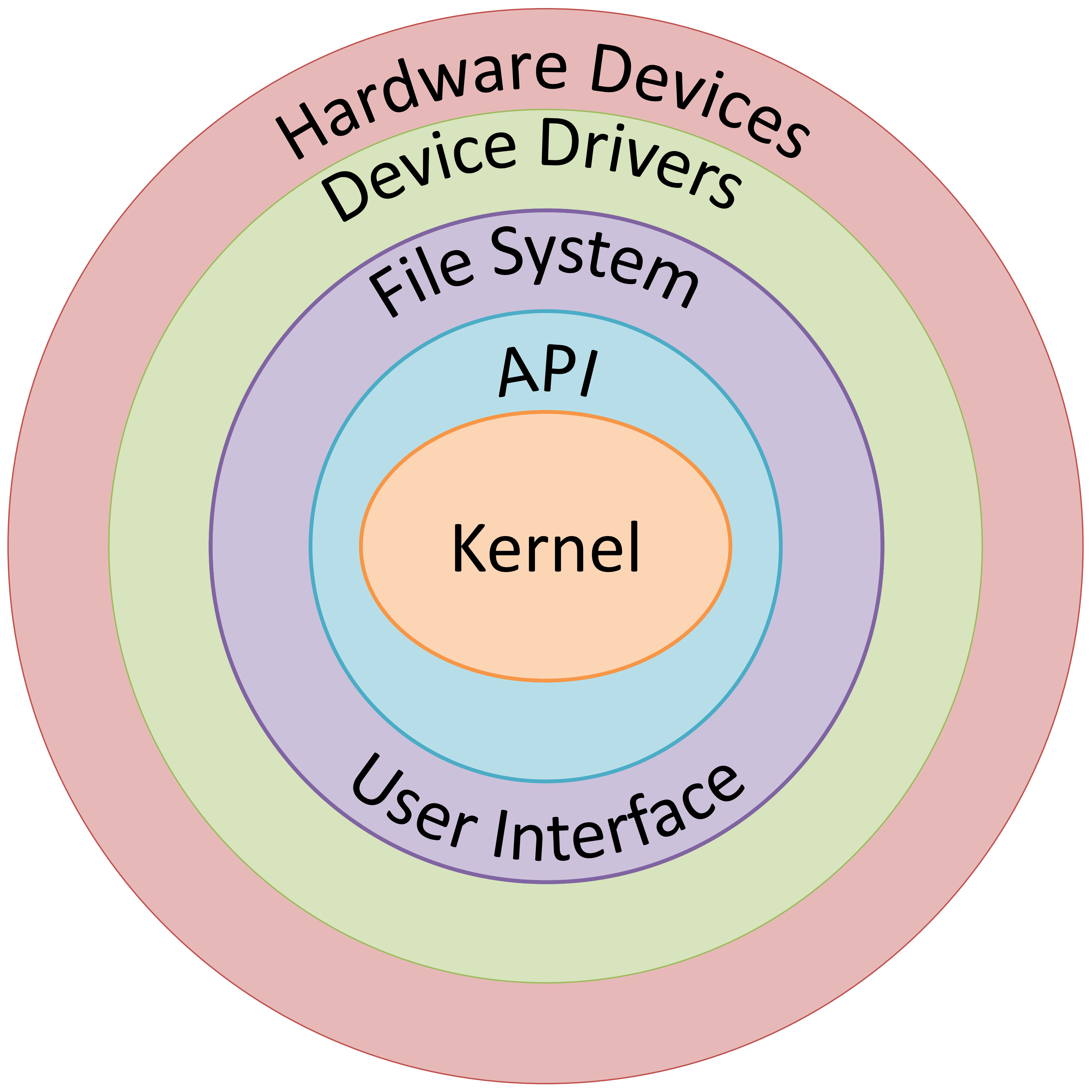
There are two key operating system design principles that should be considered…

2. Optimization for common case

- This principle underscores the importance of designing the OS to **efficiently** **handle typical workloads and usage scenarios**.
- OS should be **optimized** based on the **expected usage patterns**, **workload requirements**, and the **intended environment** where they will be deployed.
- By identifying and prioritizing the common use cases, operating systems can streamline their design and implementation to deliver **better performance**, **responsiveness**, and **resource utilization in those scenarios**.
- This optimization **enhances the overall user experience** and ensures that the operating system performs well under typical conditions.

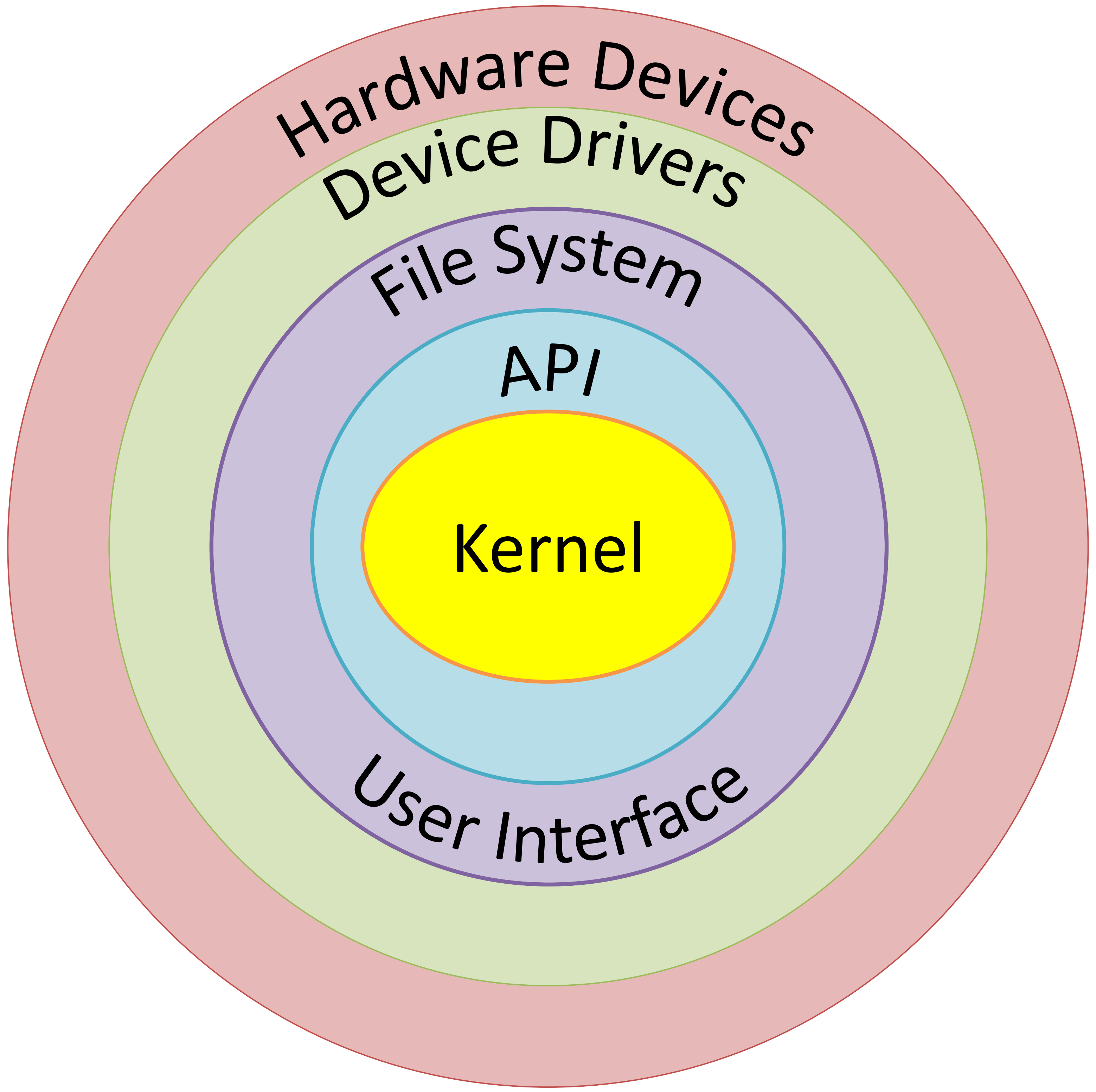
OS Components

There are several components to build up an OS



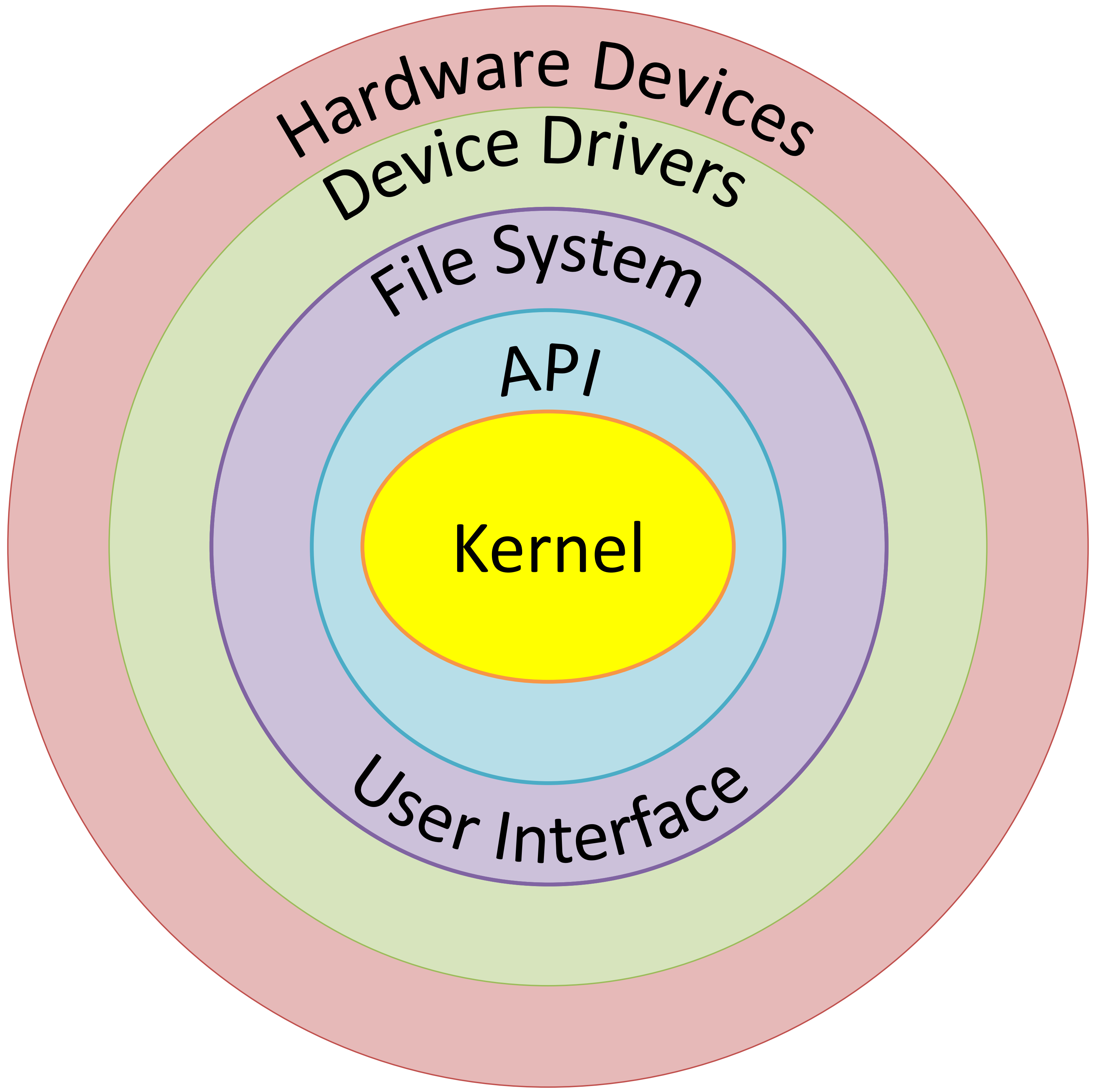
The Kernal

- The kernel is **the core component** of the OS responsible for providing essential OS services such as:
  - ➢process management,
  - ➢memory management,
  - ➢file system management,
  - ➢device management, and
  - ➢system call handling.
- It directly interacts with the hardware and manages its resources.



The Interaction

- All applications, inclusive of containerized applications, rely on the **underlying kernel.**
- The kernel provides an API to these applications via **system calls.**
- Versioning of this API matters as it's the "**glue**" that ensures deterministic communication between the ***user space*** and ***kernel space***.



Kernel Space vs. User Space

- In an operating system, there are two primary spaces where code can execute:

User Space → ***Unprivileged***

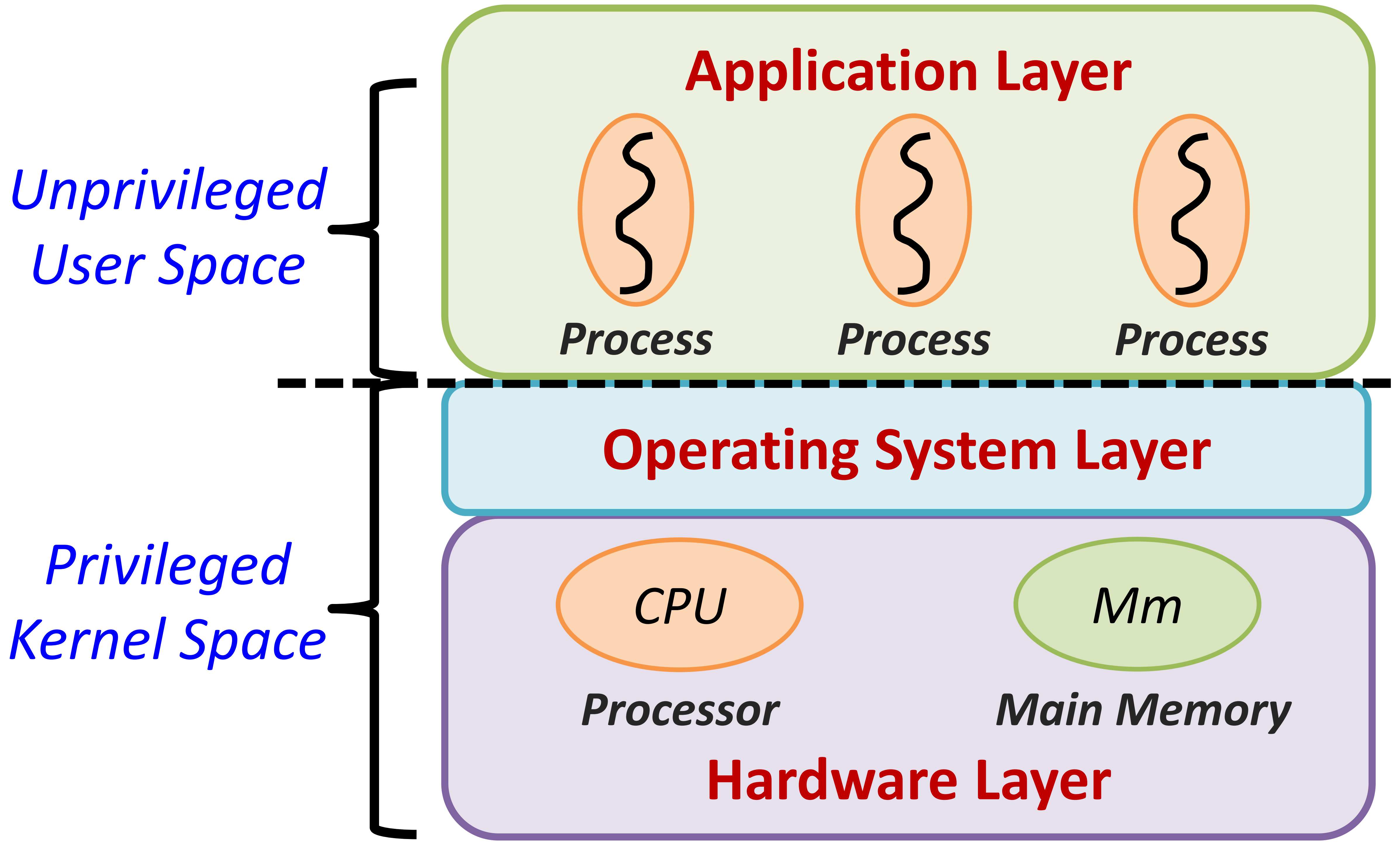
&

Kernel Space → ***Privileged***

- User space is where user applications execute, while kernel space is where the OS itself and other privileged components execute.
- In kernel space, code has direct access to system resources like memory and hardware, enabling privileged operations not available in user space.

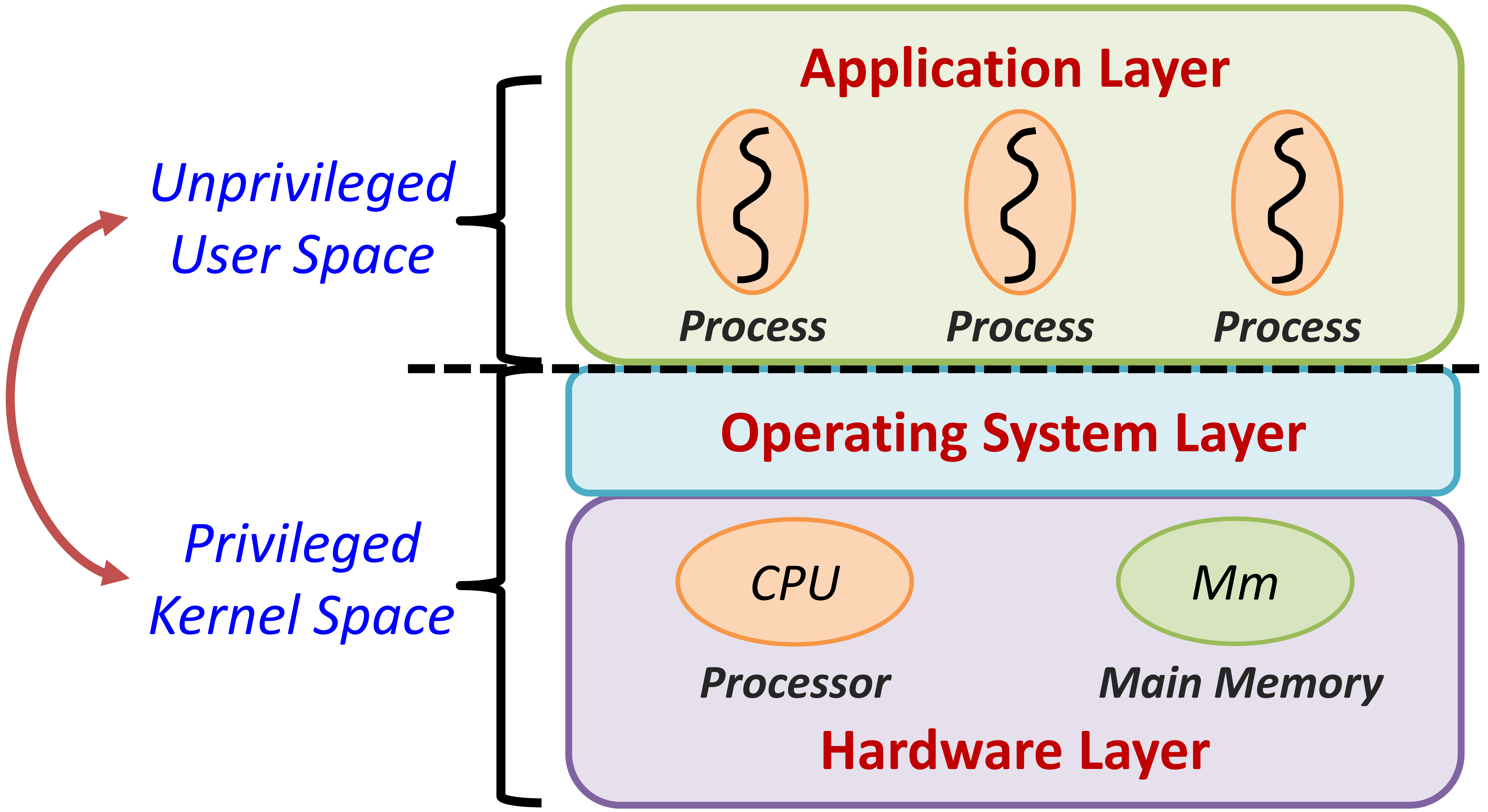
OS Protection Boundary

- Generally, applications operate in unprivileged mode (user level) while operating systems operate in privileged mode (kernel level)
- Kernel level software is able to access hardware directly.
- By design, kernel space is separate from user space, which houses user applications and processes.
- This separation aims to **prevent unauthorized access** and **maintain system stability.**
- This can happen by isolating the essential operations of the kernel from potential interference or damage caused by user applications.



OS Protection Boundary

- User-kernel switch is supported by hardware
  - ➢trap instructions
  - ➢system calls (open send malloc ...)
  - ➢signals

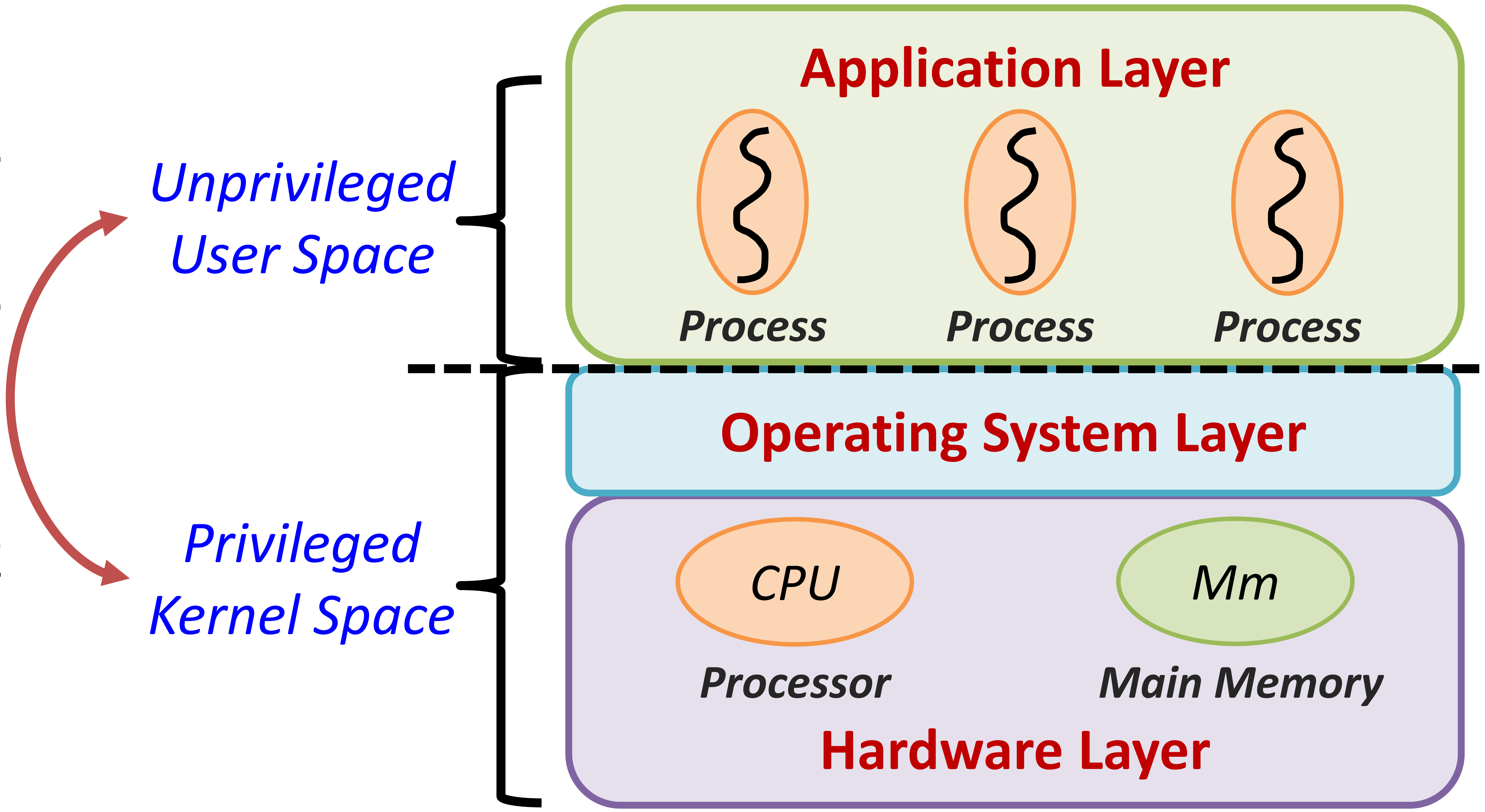


The Instruction Trap

- There are two ways to enter kernel mode:

  Interrupt & Exception

- When either occurs, the processor dispatches to the appropriate handler in its interrupt dispatch table (or similar mechanism) defined by the OS.
- The trap is a mechanism used by OS to handle exceptional conditions or events that occur during the execution of a program.
- When a trap occurs, the CPU interrupts the normal execution flow of the program and transfers control to a predefined exception handler routine in the OS kernel.

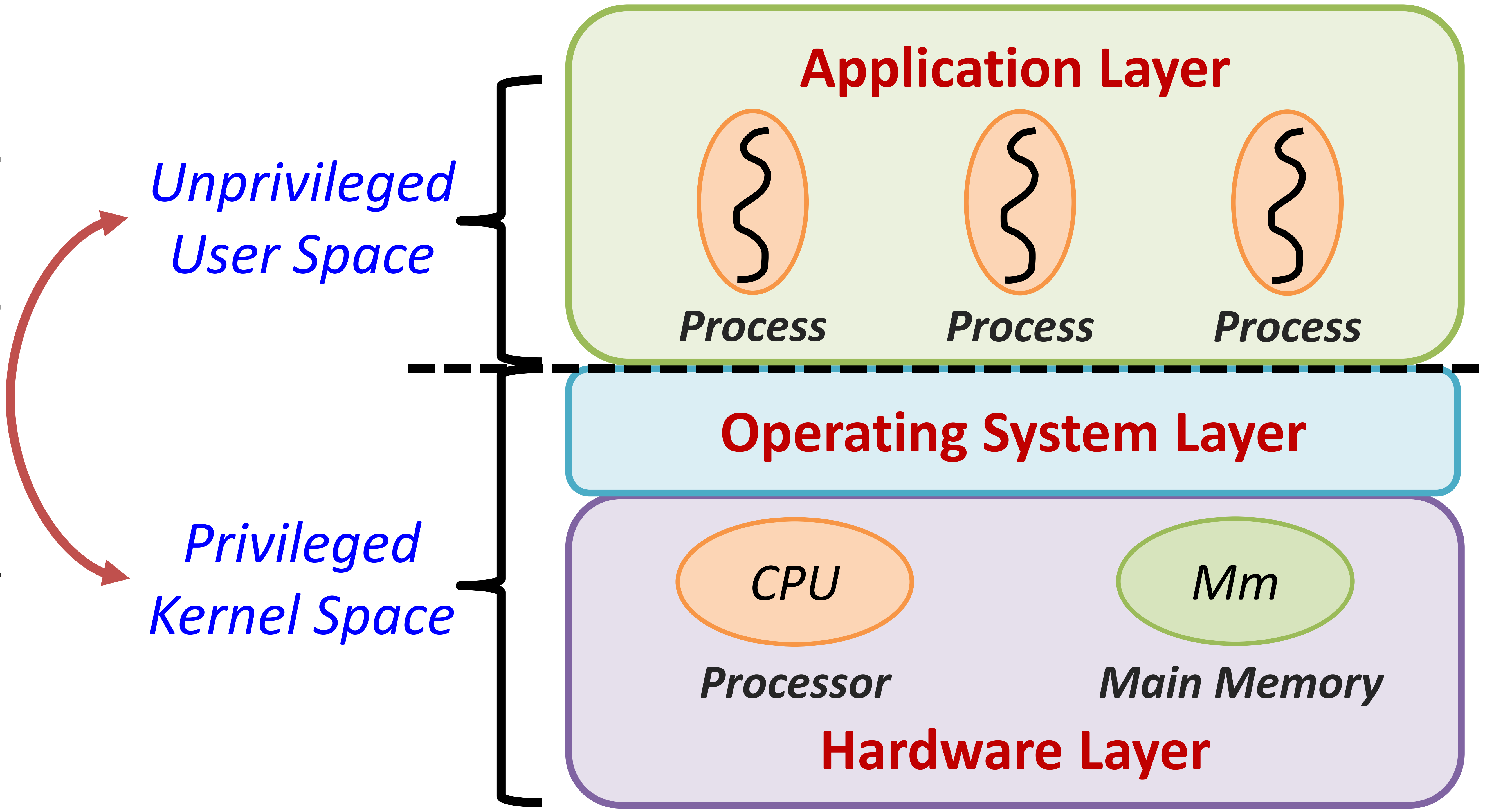


The Instruction Trap

- There are two ways to enter kernel mode:

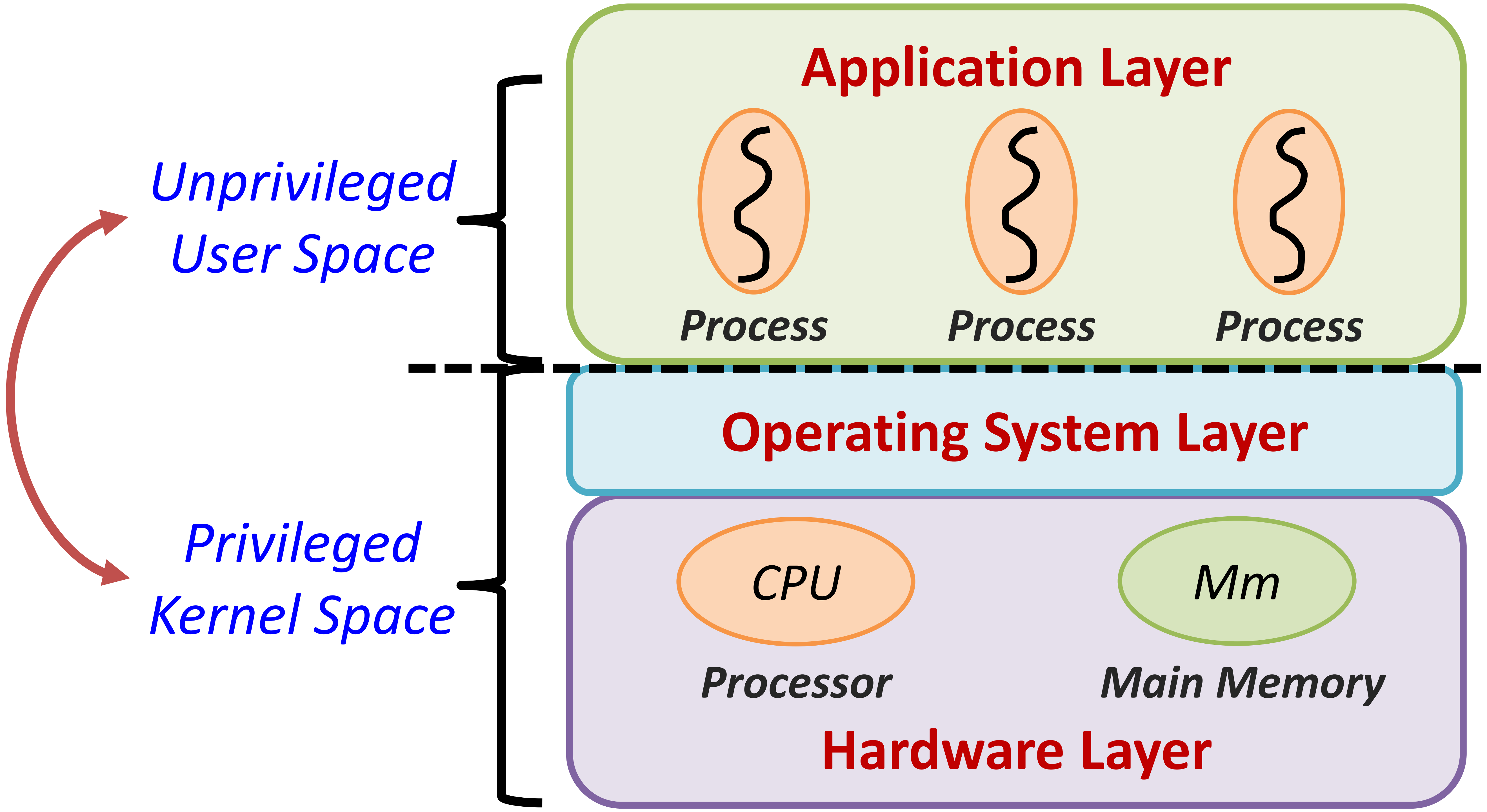
  Interrupt & Exception

- When either occurs, the processor dispatches to the appropriate handler in its interrupt dispatch table (or similar mechanism) defined by the OS.
- The trap is a mechanism used by OS to handle exceptional conditions or events that occur during the execution of a program.
- When a trap occurs, the CPU interrupts the normal execution flow of the program and transfers control to a predefined exception handler routine in the OS kernel.



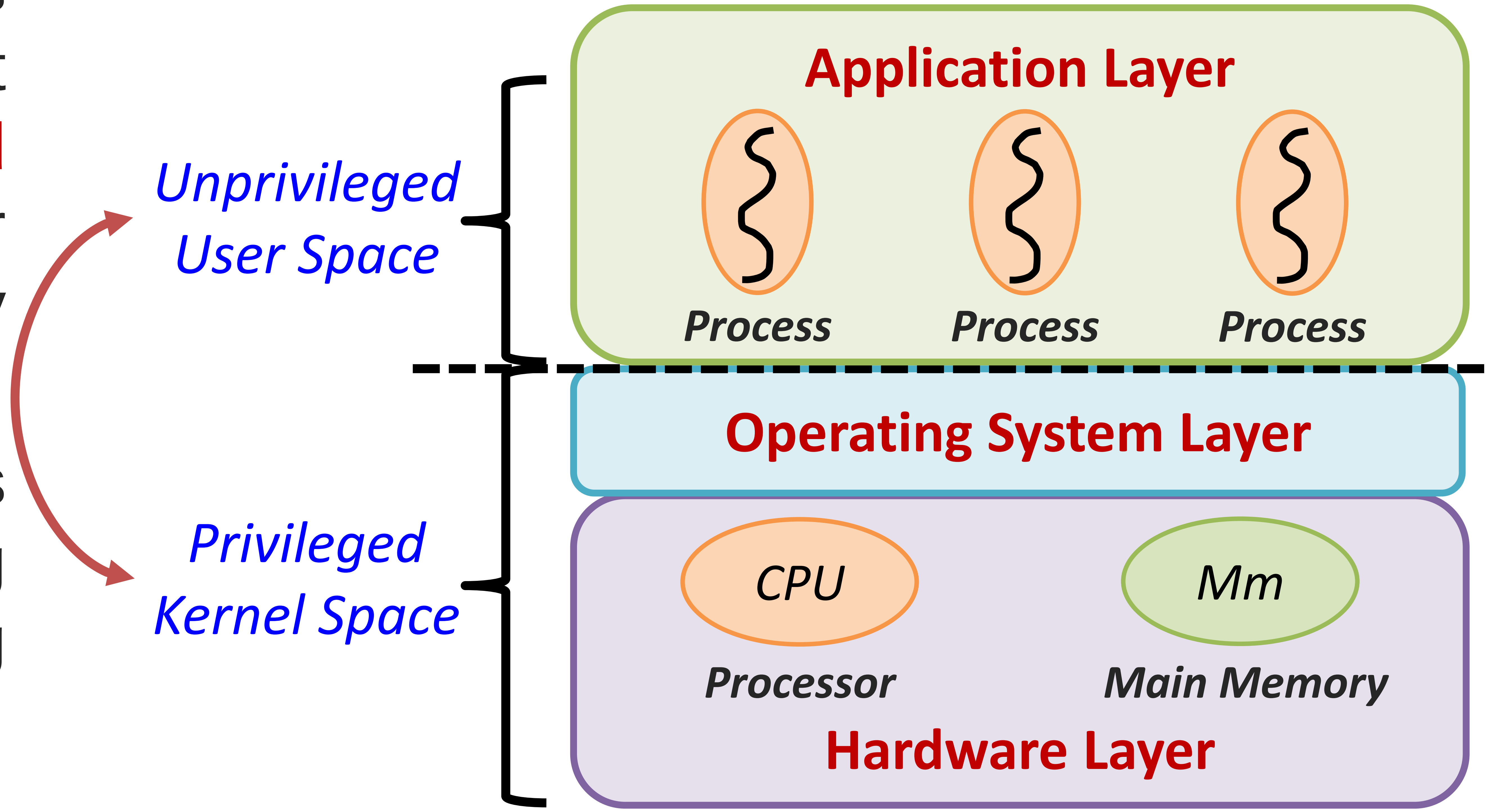
The System Calls

- System Calls serve as a bridge, allowing **user applications to request specific services from the kernel**.
- When an application makes a system call, it triggers a controlled switch from user space to kernel space, enabling the kernel to execute the requested service on behalf of the user application.



The System Calls

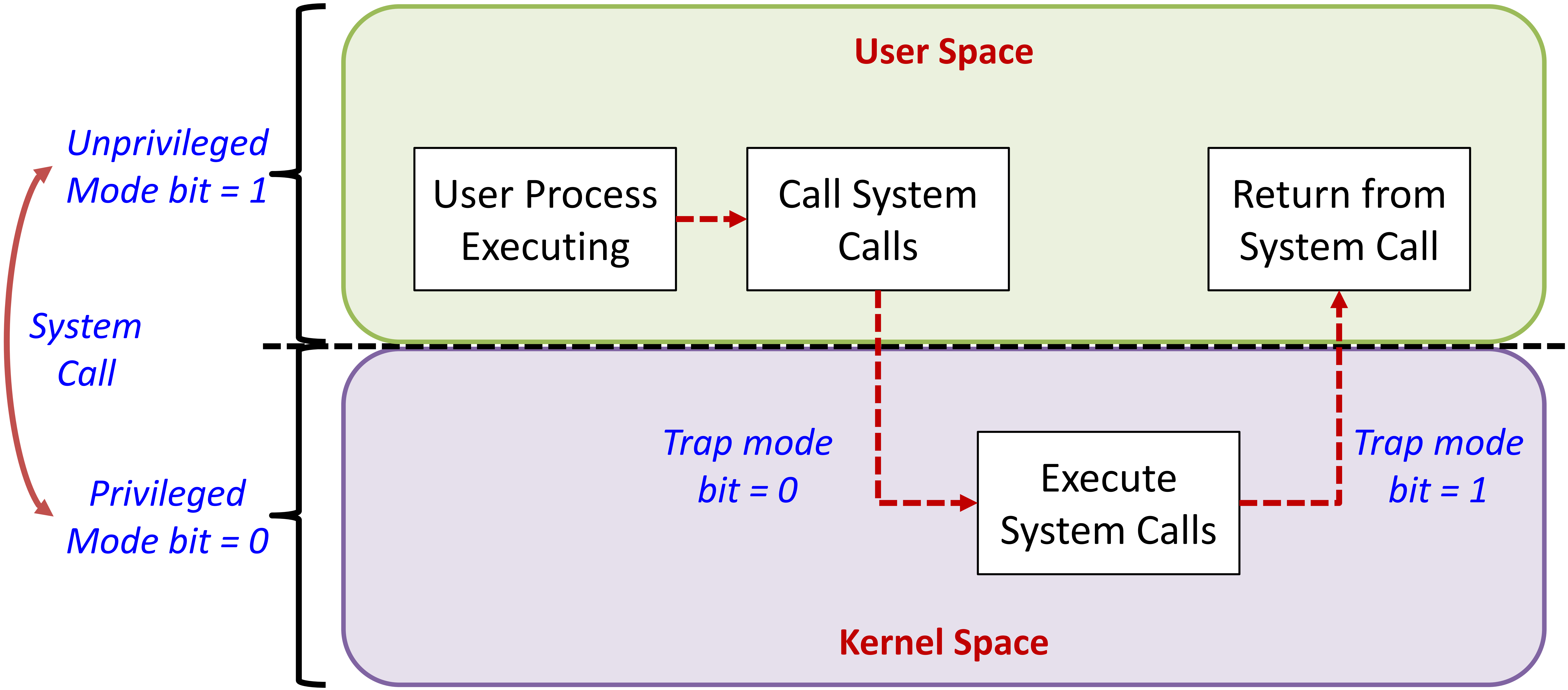
- Since the kernel handles these tasks within the protected kernel space, it **safeguards the system's integrity and stability** while still allowing user applications to access necessary resources and services.
- Some common examples of system calls include opening and closing files, reading from or writing to a file, and creating processes.



System Call Flowchart

To make a system call, an application must

- Write Arguments
- Save relevant data at well-defined location
- Make system calls



System Call Flowchart

In Synchronous mode, the process will wait till the system call ends

Crossing the OS Boundary

- Applications will need to utilize user-kernel transitions which is accomplished by hardware.
- This involves several **instructions** and **switches locality**.
- Switching locality will **affect hardware cache**

The transitions are costly!!!

- Hardware Cache
  - Because context switches will swap the data/addresses currently in cache, the performance of applications can benefit or suffer based on how a context switch changes what is in cache at the time they are accessing it.
  - A cache would be considered hot (**fire**) if an application is accessing the cache when it contains the data/addresses it needs.
  - Likewise, a cache would be considered cold (**ice**) if an application is accessing the cache when it does not contain the data/addresses it needs -- forcing it to retrieve data/addresses from main memory.

Next Time

- During next week's lecture, we will cover:
  - ➢OS Organizations
  - ➢Processes and Processes Management

For Further Inquiries, Please send an email

Catherine.elias@guc.edu.eg, Catherine.elias@ieee.org

Thank you for your attention!

See you next time ☺