Cloud Computing Concepts

CS3132

Dr. Anand Kumar Mishra

NIIT University

Virtual Memory
Virtual Address Space
User Mode
Kernel Mode

Virtual Memory

- Virtual memory as a concept is that memory can be backed differently
- Some memory of a process can be on disk, some in main memory, some could even be on a remote network
- This is managed by the OS and transparent to the running user process
- To the user process it's just memory

Virtual Memory

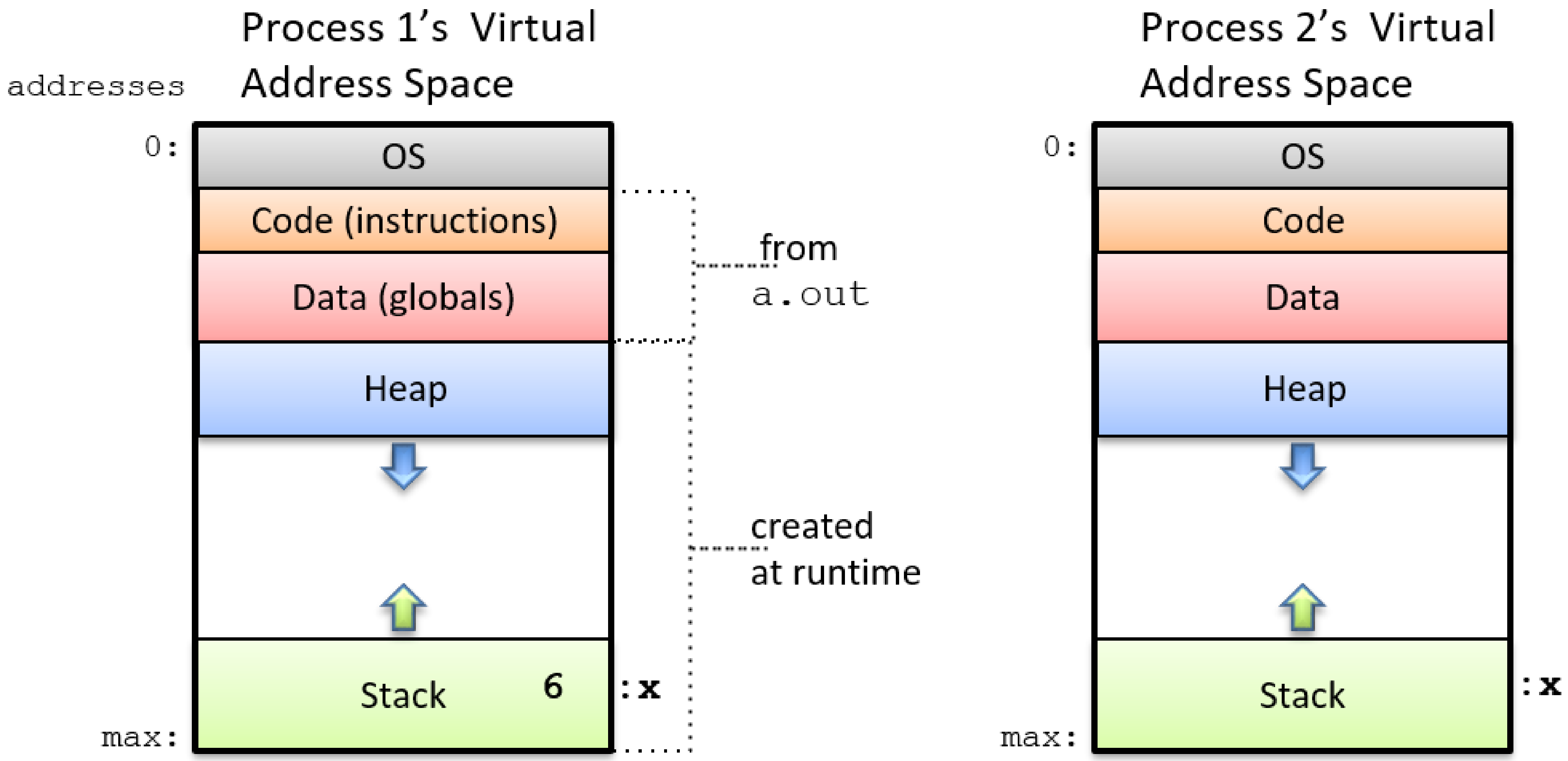
- Virtual memory is a memory management technique where secondary memory can be used as if it were a part of the main memory
- Virtual memory is a common technique used in a computer's operating system (OS)
- Virtual memory uses both hardware and software to enable a computer to compensate for physical memory shortages, temporarily transferring data from random access memory (RAM) to disk storage
- Mapping chunks of memory to disk files enables a computer to treat secondary memory as though it were main memory.

Virtual Address Space

- Virtual memory is a memory management technique developed for multitasking kernels
- Virtual address space is a memory mapping mechanism available in modern operating systems

Virtual Address Space

- The virtual address space for a process is the set of virtual memory addresses that it can use
- The address space for each process is private and cannot be accessed by other processes unless it is shared

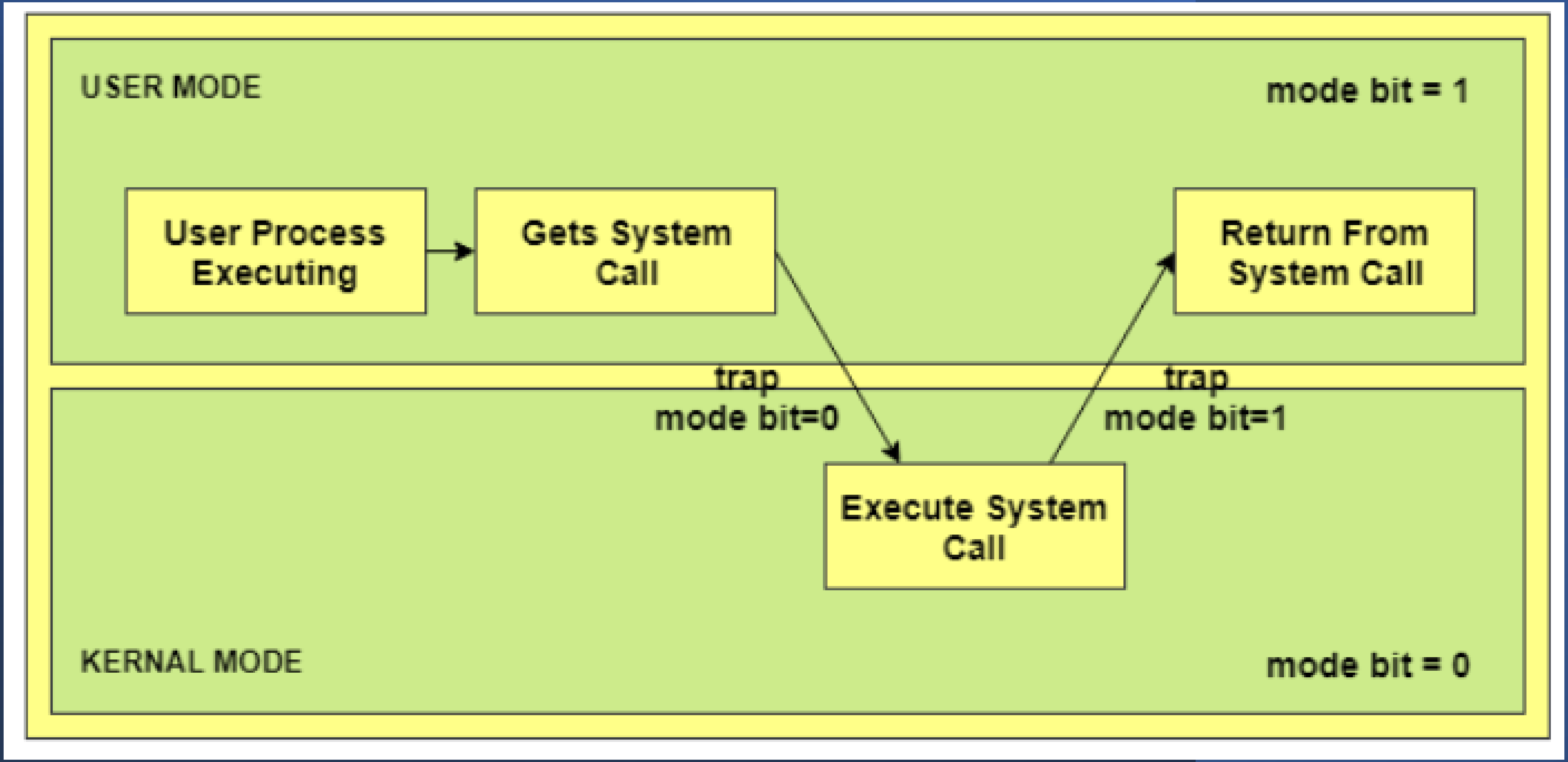

Process 1's Virtual Address Space
addresses
0:
OS
Code (instructions)
Data (globals)
Heap
Stack
6
:x
max:
from
a.out
created
at runtime
Process 2's Virtual Address Space
0:
OS
Code
Data
Heap
Stack
:x
max:

Virtual Address Space

- A virtual address does not represent the actual physical location of an object in memory; instead,
    - the system maintains a page table for each process, which is an internal data structure used to translate virtual addresses into their corresponding physical addresses

User mode and kernel mode

- A processor in a computer running Windows has two different modes: user mode and kernel mode
- The processor switches between the two modes depending on what type of code is running on the processor:
  - Applications run in user mode, and
  - Core operating system components run in kernel mode


USER MODE
mode bit = 1
User Process Executing
Gets System Call
Return From System Call
trap
mode bit=0
trap
mode bit=1
Execute System Call
KERNAL MODE
mode bit = 0

User Mode

- Every user process operates under the user mode
- In this mode, processes do not have direct access to the RAM or other hardware resources
- Processes have to make system calls to the underlying APIs to access these resources

Kernel Mode

- The kernel mode has direct access to all the underlying hardware resources
- In the kernel mode, all memory addresses are accessible and all CPU instructions are executable
- Kernel mode is usually reserved for drivers which need finer control over the hardware they are operating on

Kernel

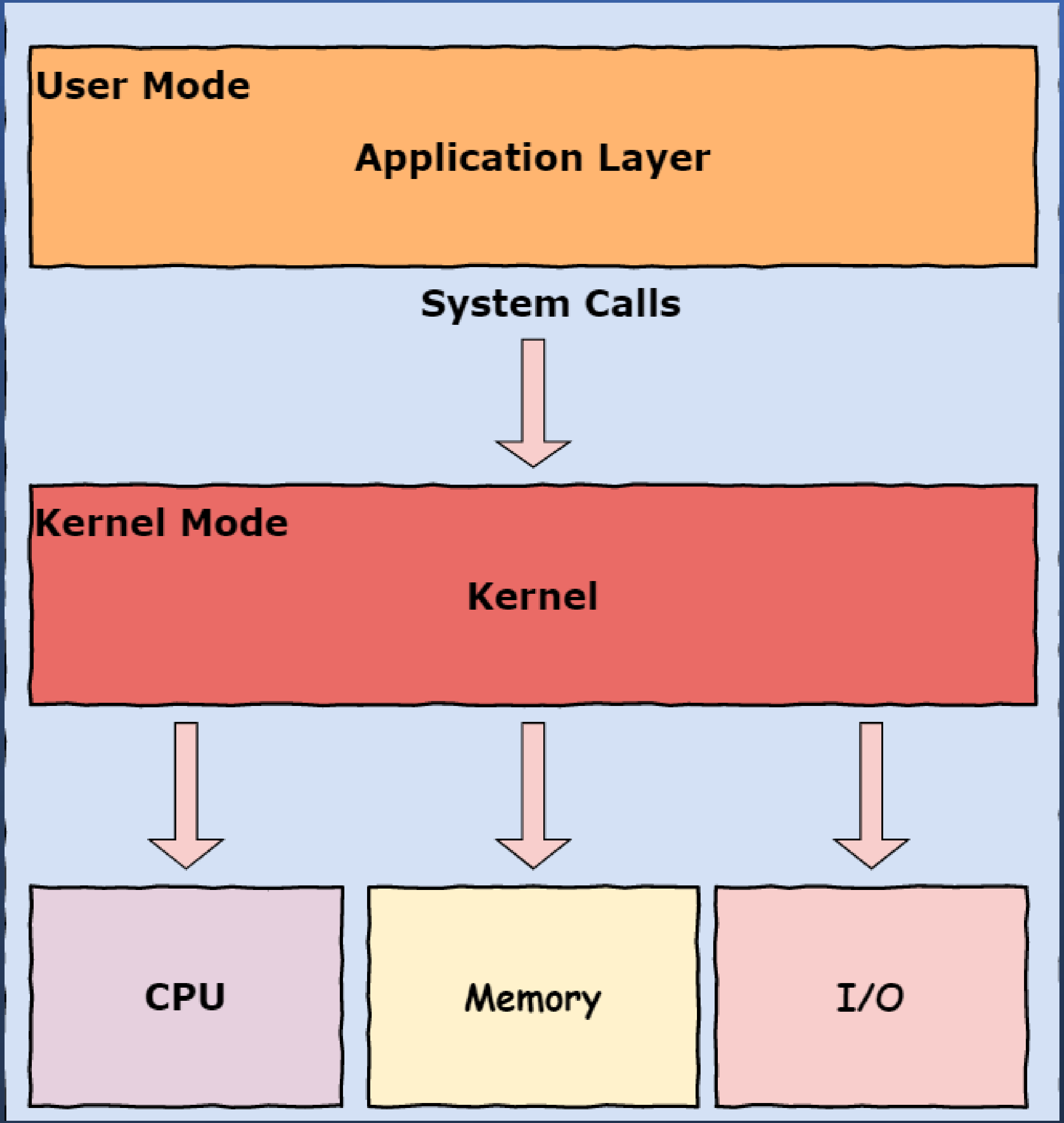
- The kernel is a software program
- It is the core of an operating system and is responsible for managing the system's resources, such as the CPU, memory, and devices
- The kernel also provides an interface between the operating system and the hardware
- The kernel is typically loaded into memory when the computer boots up and remains in memory until the computer is shut down

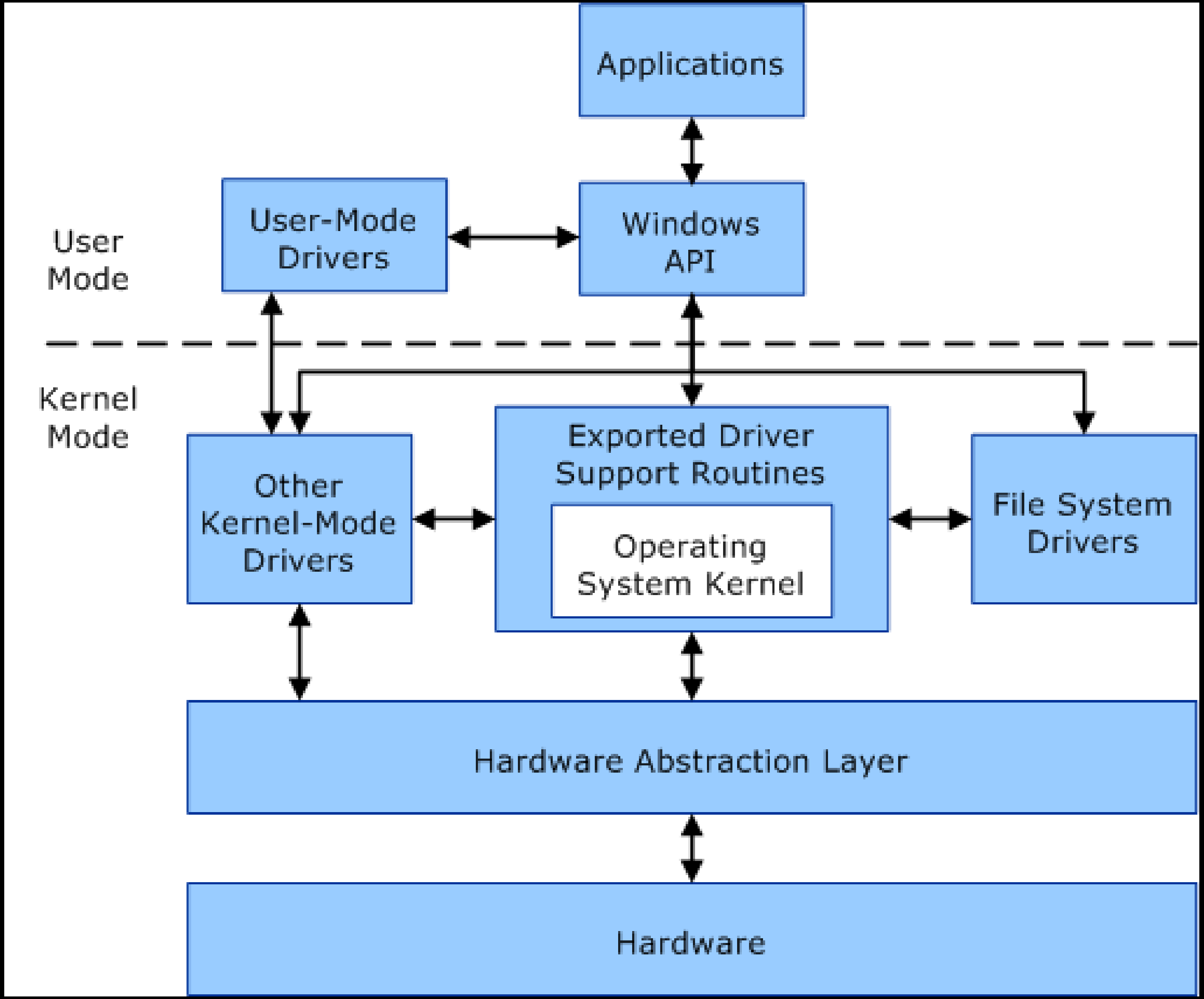
Kernel

- Kernel is responsible for many important tasks, including:
  - Starting and stopping programs
  - Managing memory and CPU usage
  - Handling interrupts from devices
  - Providing a file system
  - Providing a network interface

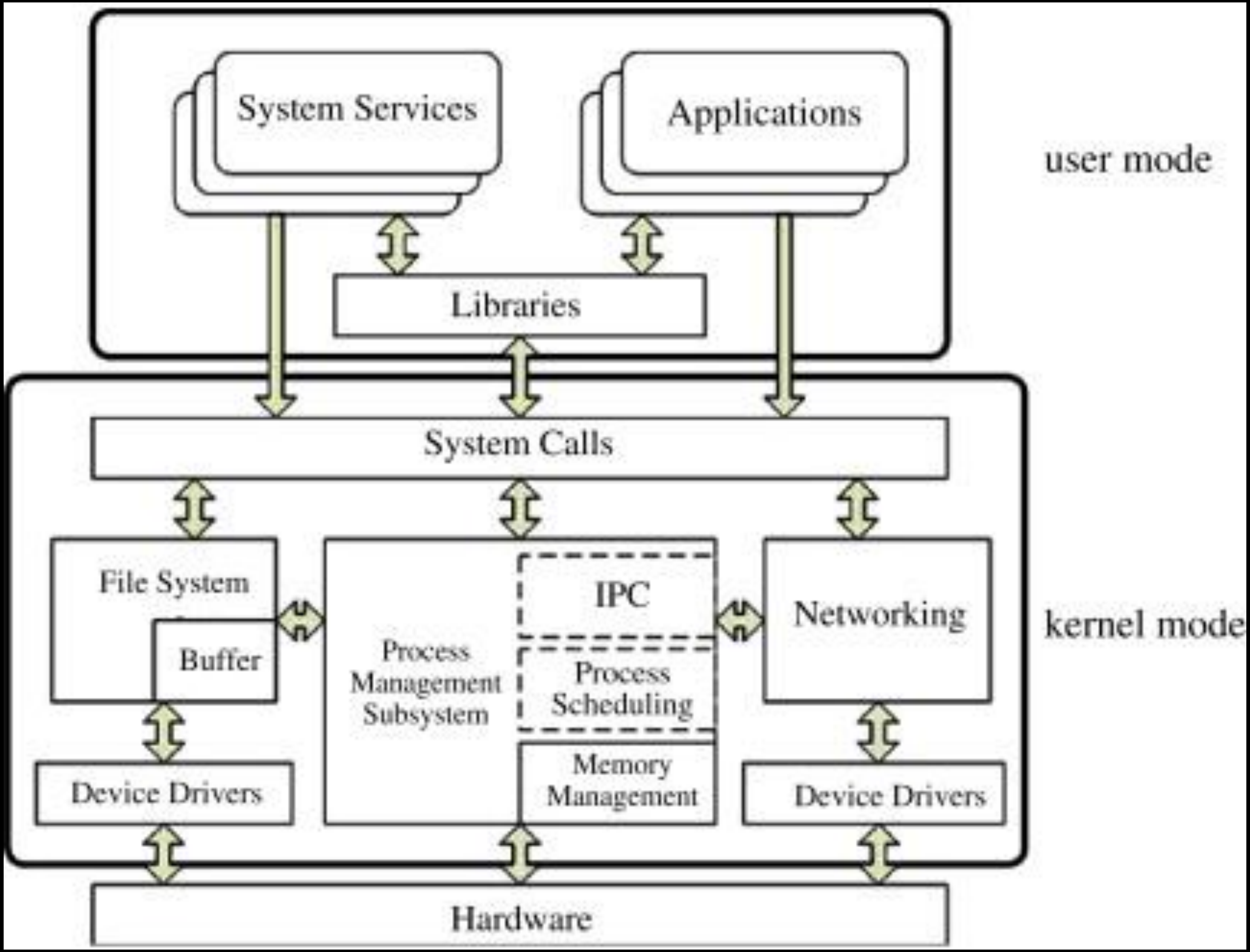
Kernel

- The kernel is essential for the operation of an operating system
- Without the kernel, the operating system would not be able to manage the system's resources or provide an interface to the hardware
- Here are some examples of kernels:
  - Linux kernel
  - Windows kernel
  - macOS kernel
  - iOS kernel
  - Android kernel

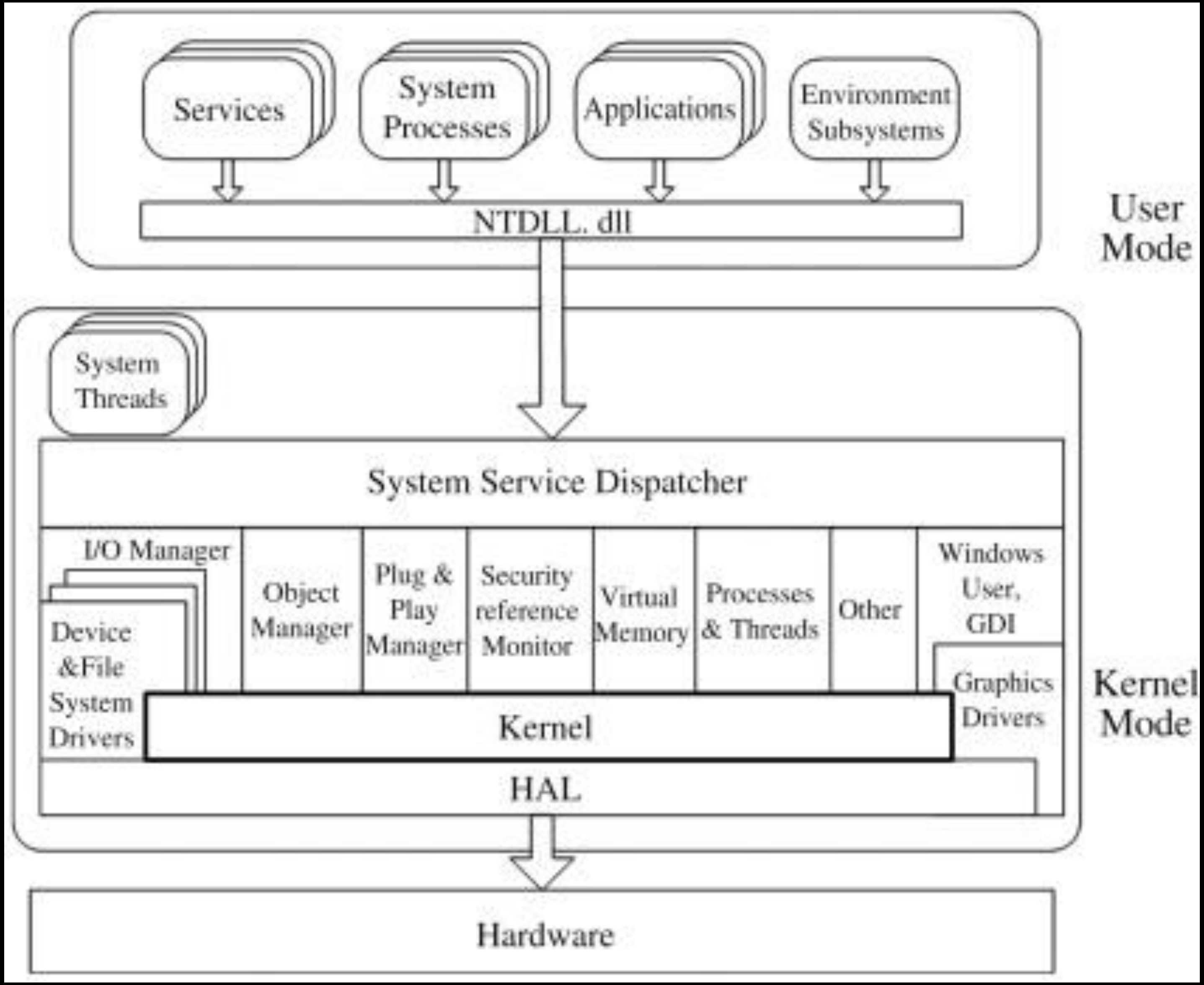

User Mode
Application Layer
System Calls
Kernel Mode
Kernel
CPU
Memory
I/O


Applications
User Mode
User-Mode Drivers
Windows API
Kernel Mode
Other Kernel-Mode Drivers
Exported Driver Support Routines
Operating System Kernel
File System Drivers
Hardware Abstraction Layer
Hardware

Architecture of Linux



Architecture of Windows NT



Kernel Components

- The kernel is divided into several different components, each of which has a specific responsibility. The main components of the kernel are:
- Process scheduler:
  - The process scheduler is responsible for deciding which process to run next
- Memory manager:
  - The memory manager is responsible for allocating and managing memory
- File system:
  - The file system provides a way for programs to access and store data on disk
- Network interface:
  - The network interface provides a way for programs to communicate with other computers over a network
- Device drivers:
  - Device drivers provide an interface between the kernel and the hardware devices connected to the computer

Kernel Code

- Operating system (OS) kernel code is typically written in a low-level programming language, such as C or C++
- This is because kernels need to be efficient and need to have direct access to the hardware
- Kernel code is typically divided into several different files, each of which contains code for a specific component of the kernel
  - For example, there might be a file for the process scheduler, a file for the memory manager, and a file for the file system

```
// This function is called when the system needs to schedule a new process.

void schedule() {
  // Get the next process to run.
  struct process *next_process = get_next_process();

  // Switch to the next process.
  switch_to_process(next_process);
}

// This function is called when the system needs to allocate memory for a process.
void *malloc(size_t size) {
  // Allocate memory from the heap.
  void *ptr = heap_alloc(size);

  // If the allocation failed, return NULL.
  if (ptr == NULL) {
    return NULL;
  }

  // Return the pointer to the allocated memory.
  return ptr;
}

// This function is called when the system needs to free memory that was allocated by malloc().
void free(void *ptr) {
  // Free the memory back to the heap.
  heap_free(ptr);
}
```

simple example of kernel code. In reality, kernel code is much more complex and contains code for many other different tasks

Emulator

- An emulator is a software or hardware tool that replicates the functionality of one computer system on another
- It allows a computer system, known as the host, to mimic the behavior of a different computer system, known as the guest or target
- Emulators are designed to run software or programs written for the guest system on the host system
- Example:
  - Android Emulator is a popular example. It allows developers to run Android applications on a desktop computer for testing and development, even if the developer's machine is not an Android device

Simulator

- A simulator is a software tool that models the behavior of a system or process without necessarily replicating the underlying hardware
- Simulators are used to understand and study the behavior of a system under various conditions or scenarios
- They often provide a higher-level abstraction of a system's functionality
- They allow users to study and experiment with the behavior of a system without the need for the actual hardware
- Example:
  - The Microsoft Flight Simulator is a well-known example of a simulator. It provides a realistic simulation of flying an aircraft, modeling various aspects of flight physics and environmental conditions, but it doesn't emulate a specific physical aircraft.

Emulator and Simulator - Key Differences:

- Scope:
  - Emulators replicate the entire hardware and software environment of a specific system, aiming to run software designed for that system on a different one
  - Simulators focus on modeling the behavior and functionality of a system without necessarily replicating its hardware
- Purpose:
  - Emulators are primarily used for running existing software on different platforms, ensuring compatibility
  - Simulators are used for experimentation, testing, training, and research to understand system behavior

Emulator and Simulator - Key Differences:

- Level of Abstraction:
  - Emulators aim to replicate the exact behavior of the target system, down to the hardware level
  - Simulators provide a higher-level abstraction, focusing on system behavior and functionality
- Examples:
  - Emulator examples include Android Emulator, game console emulators, and virtual machines (e.g., VMware)
  - Simulator examples include flight simulators, network simulators (e.g., NS-3), and hardware description language (HDL) simulators for designing integrated circuits

QEMU - Quick Emulator

- Open-source emulator and virtualization tool
  - allows users to run virtual machines (VMs) on a host system
- It provides a versatile and flexible platform for emulating a wide range of computer architectures, making it a valuable tool for various purposes, including development, testing, and running legacy software on modern hardware
- QEMU can emulate the behavior of various computer architectures, including x86, ARM, PowerPC, MIPS, and more
- It can also operate as a hypervisor to provide full virtualization for running multiple guest operating systems on a single host

MIPS (Microprocessor without Interlocked Pipelined Stages) is a family of reduced instruction set computer (RISC) instruction set architectures (ISA)