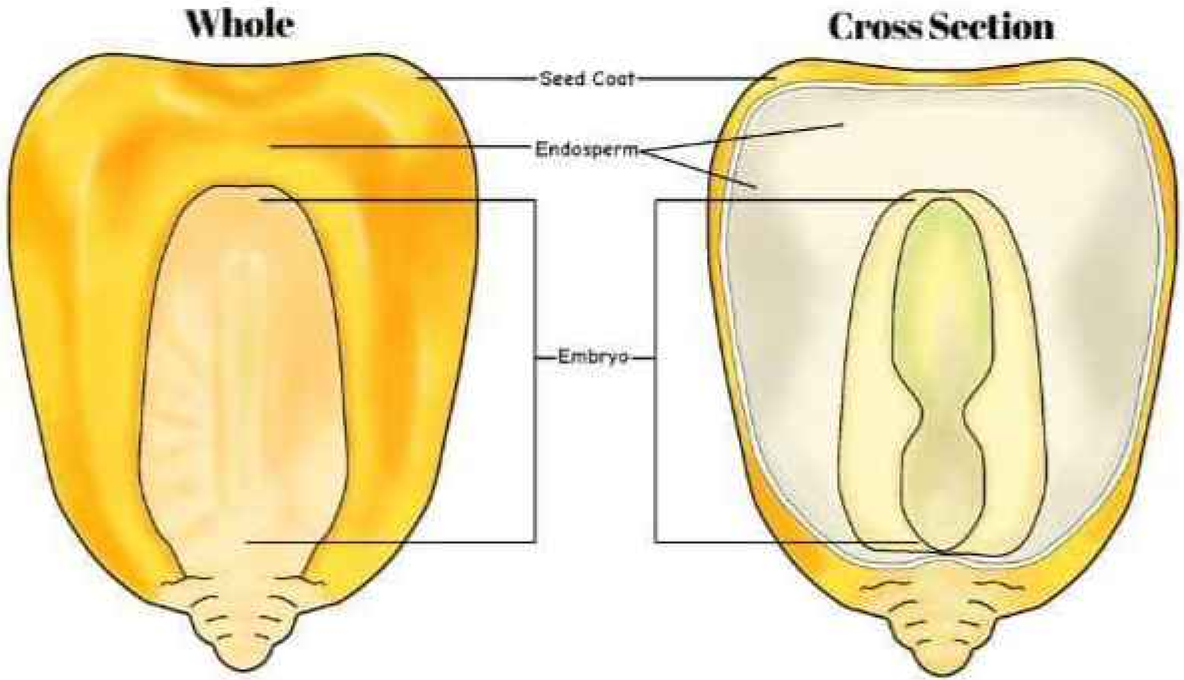
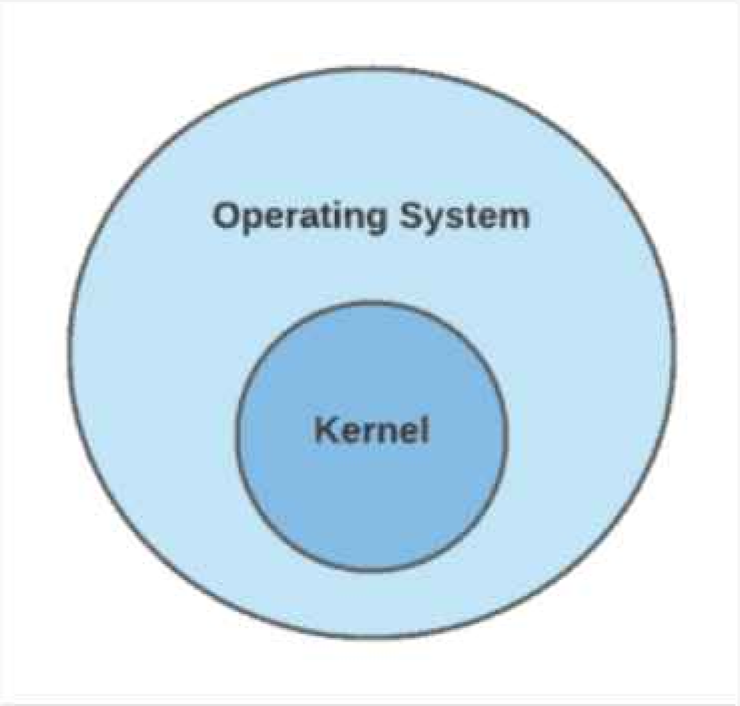
KERNEL

Seminar-03

Corn Kernel

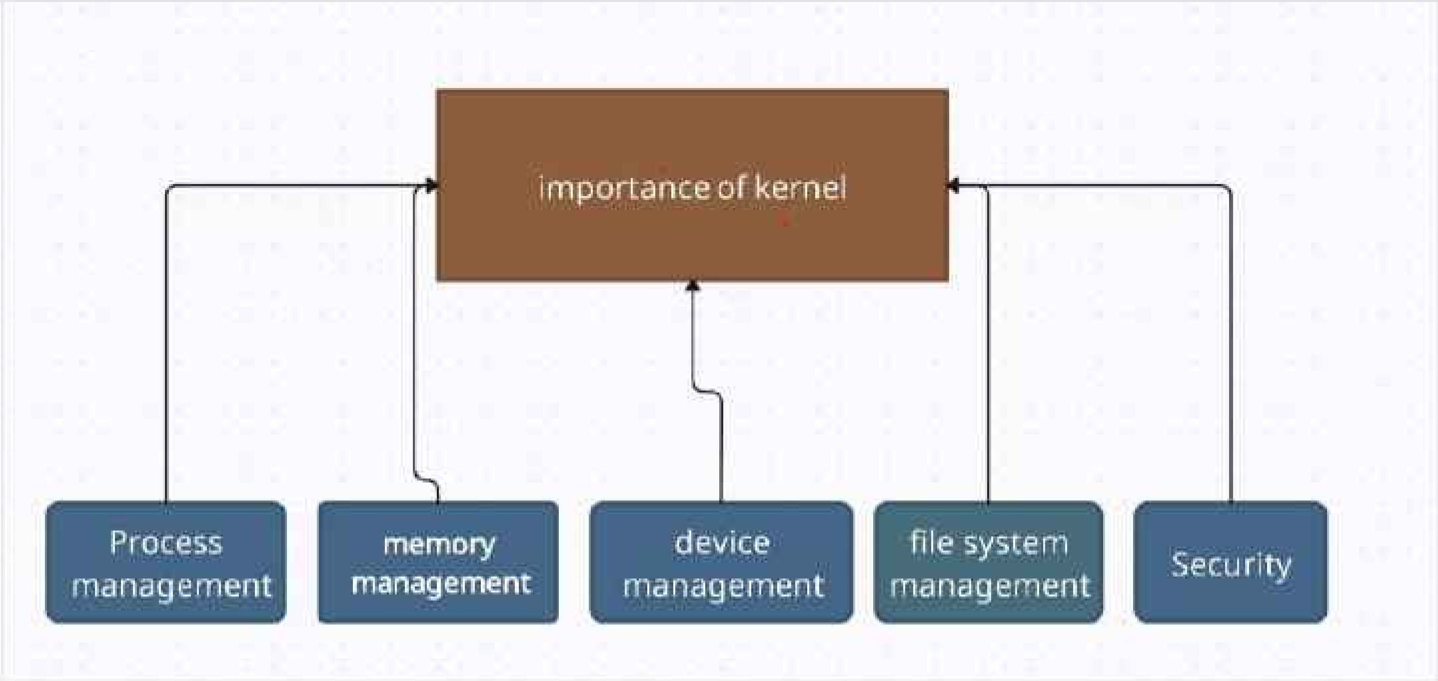




The kernel is a computer program at the core of operating system in managing computer operations and hardware resources.

• Kernel importance

The kernel plays a crucial role in the operation of an operating system by providing the following key functionalities:



Source: Linux kernel guide

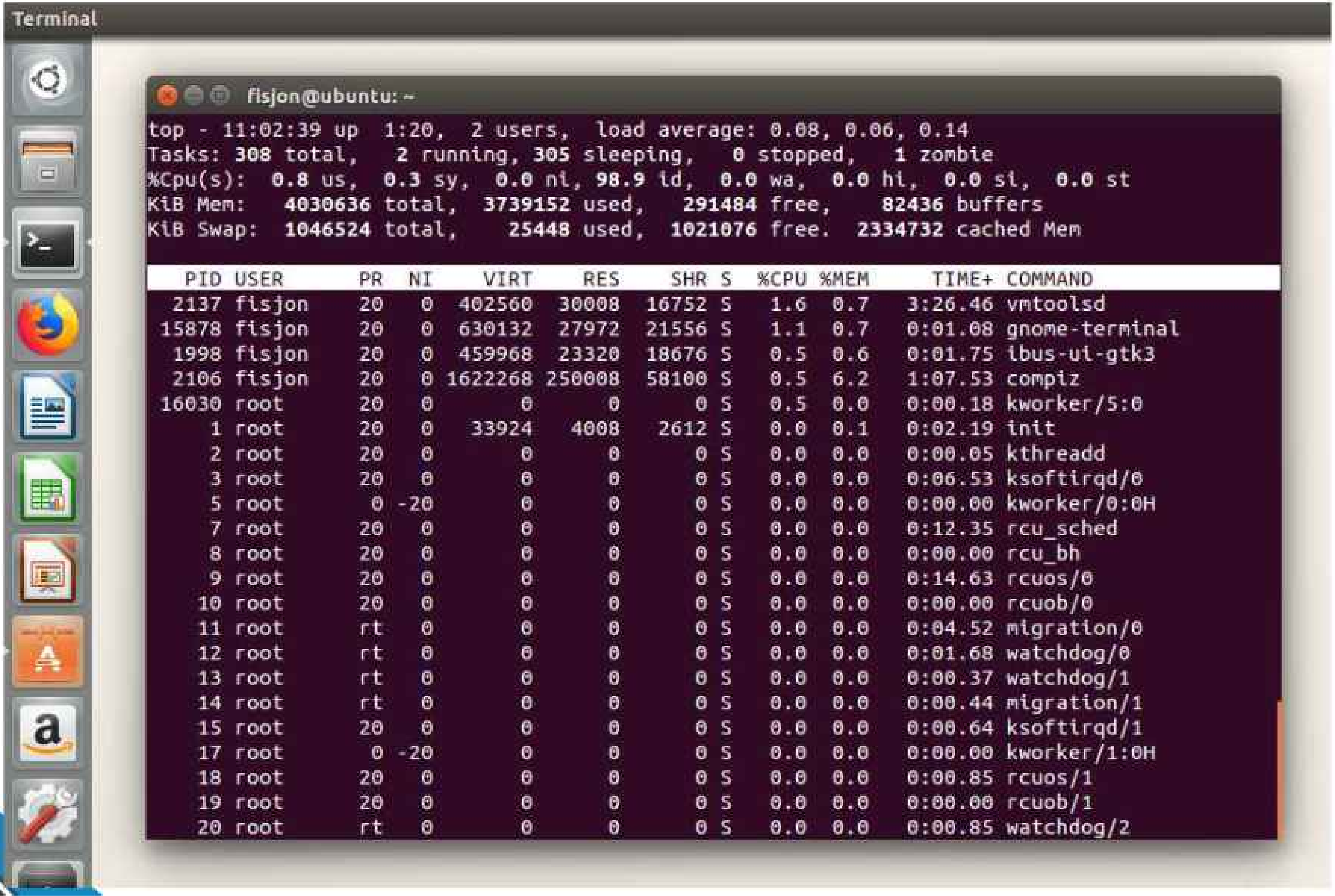
Kernel & OS interaction:

- The **operating system** encompasses a broader set of components, including the kernel, device drivers, system libraries, and utilities.
- The **kernel** is the heart of the operating system, managing system resources such as the CPU, memory, and I/O devices.
- While the operating system provides a higher-level interface to users (such as GUIs and file systems), the kernel offers low-level services to other parts of the operating system.

System Monitoring (practical)

Commands:

- Top
- Htop
- Cat /proc/cpuinfo
- Cat /proc/meminfo
- Uptime
- free



Terminal
fisjon@ubuntu: ~
top - 11:02:39 up 1:20, 2 users, load average: 0.08, 0.06, 0.14
Tasks: 308 total, 2 running, 305 sleeping, 0 stopped, 1 zombie
%Cpu(s): 0.8 us, 0.3 sy, 0.0 ni, 98.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 4030636 total, 3739152 used, 291484 free, 82436 buffers
KiB Swap: 1046524 total, 25448 used, 1021076 free. 2334732 cached Mem
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
2137 fisjon 20 0 402560 30008 16752 S 1.6 0.7 3:26.46 vmtoolsd
15878 fisjon 20 0 630132 27972 21556 S 1.1 0.7 0:01.08 gnome-terminal
1998 fisjon 20 0 459968 23320 18676 S 0.5 0.6 0:01.75 ibus-ui-gtk3
2106 fisjon 20 0 1622268 250008 58100 S 0.5 6.2 1:07.53 compiz
16030 root 20 0 0 0 0 S 0.5 0.0 0:00.18 kworker/5:0
1 root 20 0 33924 4008 2612 S 0.0 0.1 0:02.19 init
2 root 20 0 0 0 0 S 0.0 0.0 0:00.05 kthreadd
3 root 20 0 0 0 0 S 0.0 0.0 0:06.53 ksoftirqd/0
5 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/0:0H
7 root 20 0 0 0 0 S 0.0 0.0 0:12.35 rcu_sched
8 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcu_bh
9 root 20 0 0 0 0 S 0.0 0.0 0:14.63 rcuos/0
10 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcuob/0
11 root rt 0 0 0 0 S 0.0 0.0 0:04.52 migration/0
12 root rt 0 0 0 0 S 0.0 0.0 0:01.68 watchdog/0
13 root rt 0 0 0 0 S 0.0 0.0 0:00.37 watchdog/1
14 root rt 0 0 0 0 S 0.0 0.0 0:00.44 migration/1
15 root 20 0 0 0 0 S 0.0 0.0 0:00.64 ksoftirqd/1
17 root 0 -20 0 0 0 S 0.0 0.0 0:00.00 kworker/1:0H
18 root 20 0 0 0 0 S 0.0 0.0 0:00.85 rcuos/1
19 root 20 0 0 0 0 S 0.0 0.0 0:00.00 rcuob/1
20 root rt 0 0 0 0 S 0.0 0.0 0:00.85 watchdog/2

1)PID: process ID

2)USER: username of current user.

3)PR: priority of the task.

4)NI: nice value of the task.

5)VIRT: total virtual memory used by task.

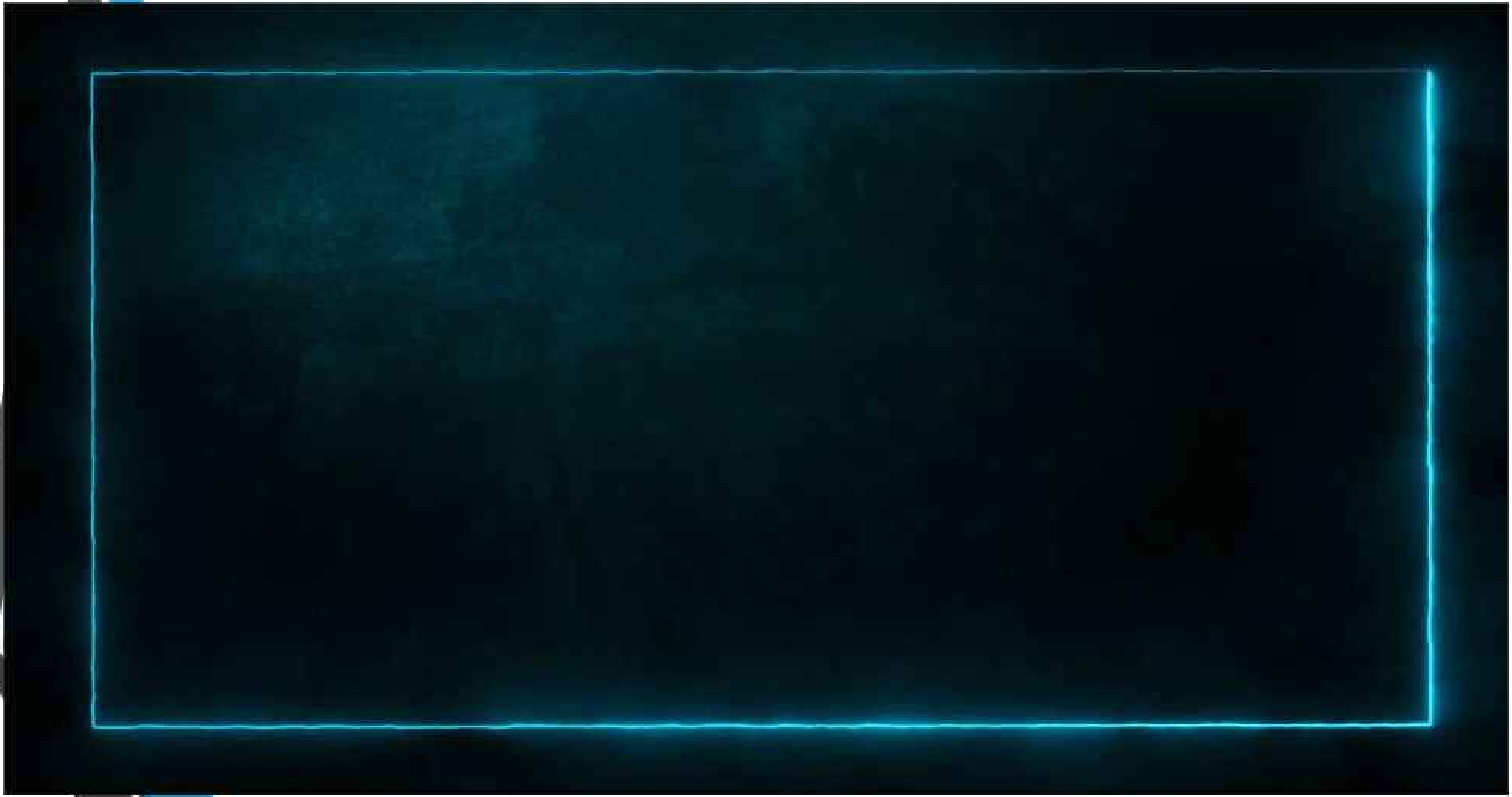
6)RES: resident memory used by the task.

7)SHR: share memory.

8)S: state of the task.

Memory spaces


windows 10
please shutdown
update time xd
linux
$ shutdown -P now
ok

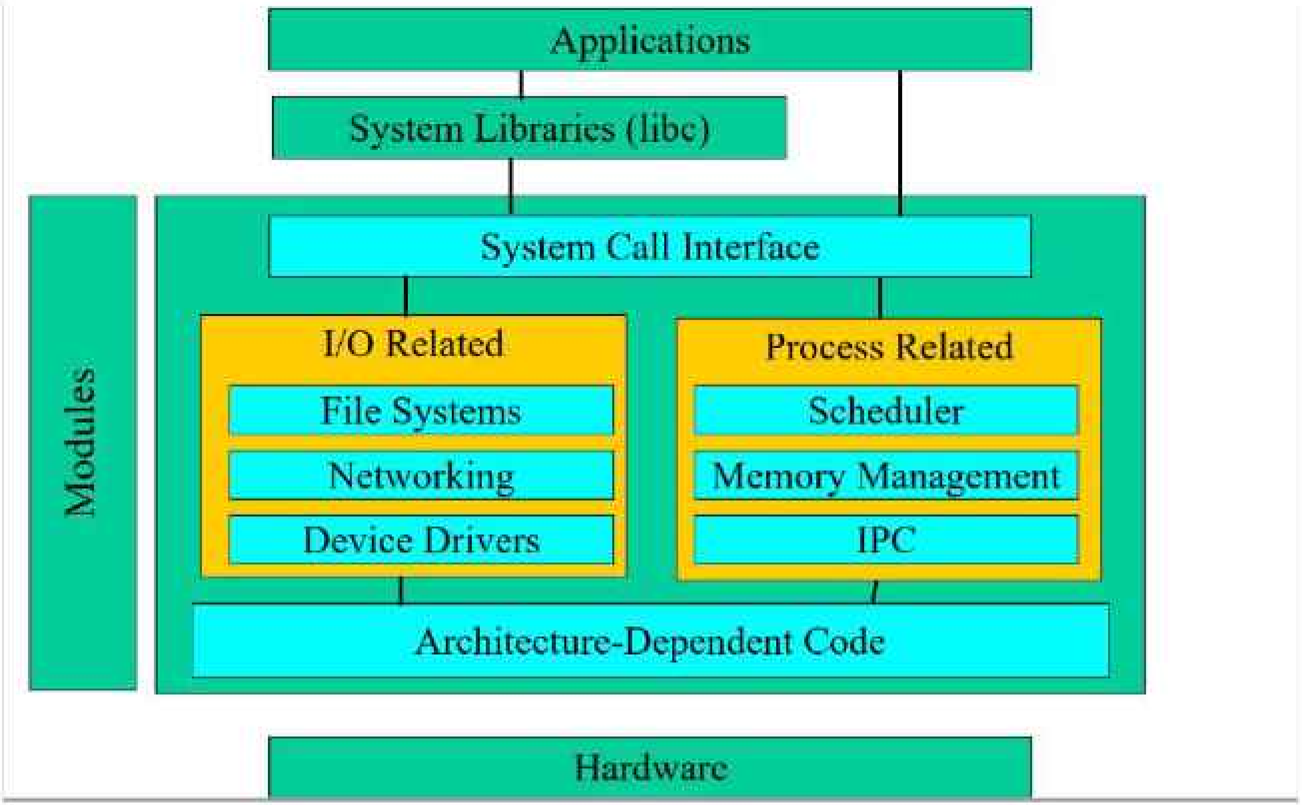


- **Topics covered:**
  - Kernel architecture
  - Kernel Space vs. User Space
  - Types Of kernel
  - Process management
  - Memory management
  - Device drivers
  - File systems
  - Kernel Security
  - ?

Overview on "**kernel architecture**"

- The kernel architecture refers to the **internal structure** and design of the kernel
- **dictates** how it manages system **resources** and provides essential services to the operating system and applications.
- **encompasses** various components and subsystems that **work together** to facilitate the **operation of the system**

- **Core kernel**


Applications
System Libraries (libc)
Modules
System Call Interface
I/O Related
File Systems
Networking
Device Drivers
Process Related
Scheduler
Memory Management
IPC
Architecture-Dependent Code
Hardware

software components that interact **directly** with the kernel to perform **various tasks.** These applications can include system utilities, user-level programs, and **system services.**

Applications

is the collections of **precompiled code** modules that provide essential **functions** and **services** to **applications** and the **operating system** itself.

System libraries

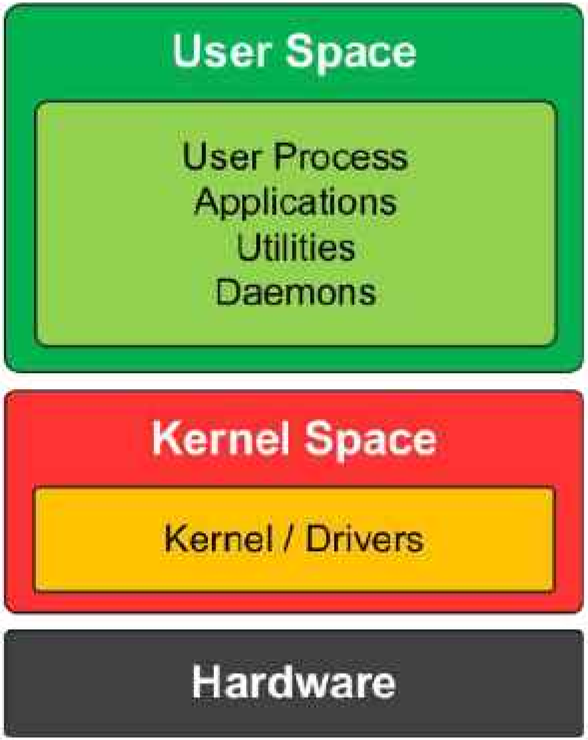
piece of **code** that can be dynamically **loaded** and **unloaded** into the kernel at **runtime.**

Modules

Components like **CPU**, Memory, Storage Devices, **Input Devices Output Devices** etc.

Hardware

Kernel vs User space



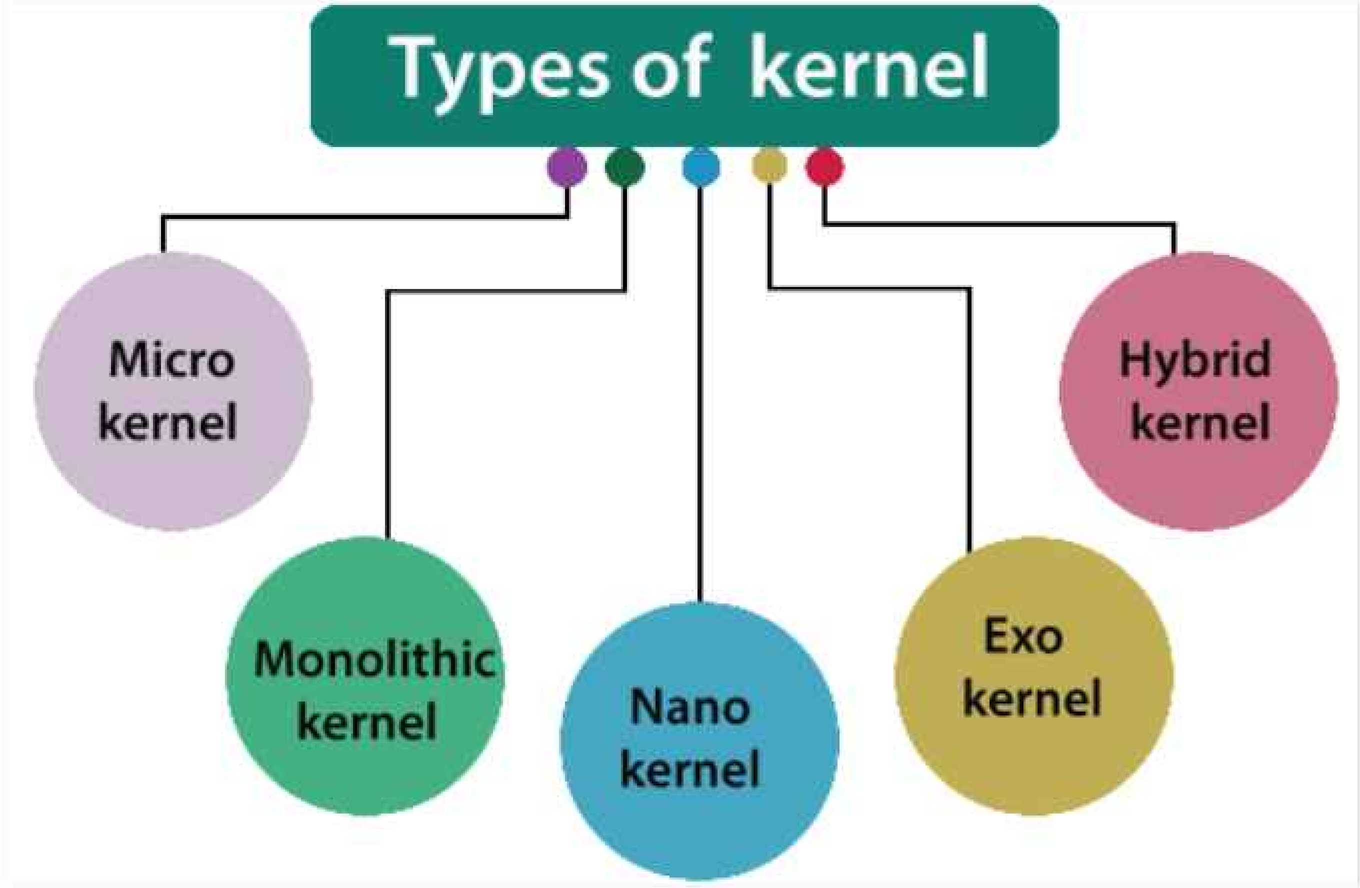
User Space:

- **user applications** and **processes** run.
- Like **web browsers**, **word processors**, and **games**, execute in user space.
- **restricted environment** where applications have limited access to system resources and are isolated from the critical functions of the operating system.
- Applications communicate & access through **system calls**, which act as a bridge between user space and kernel space.

Kernel Space:

- Kernel space, also known as **supervisor mode** or **system space**, is the privileged area of memory where the operating system kernel resides.
- The kernel is responsible for managing system resources, providing essential services, and enforcing security policies.
- **process management**, **memory management**, **device drivers**, and **hardware abstraction** –> implemented within the kernel.
- The kernel has full access to hardware resources and can execute **privileged instructions** that are restricted in **user space**.

Ubuntu For User and Kernel Space.
(prcatical)


Types of kernel
Micro kernel
Monolithic kernel
Nano kernel
Exo kernel
Hybrid kernel

Types

OR

&

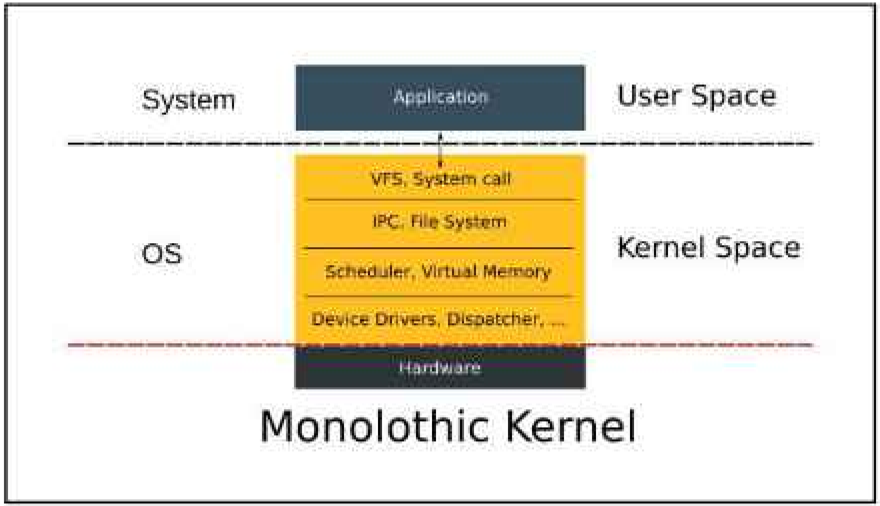
(practical)

Monolithic Kernel:

•process, device management, Runs in same address space.

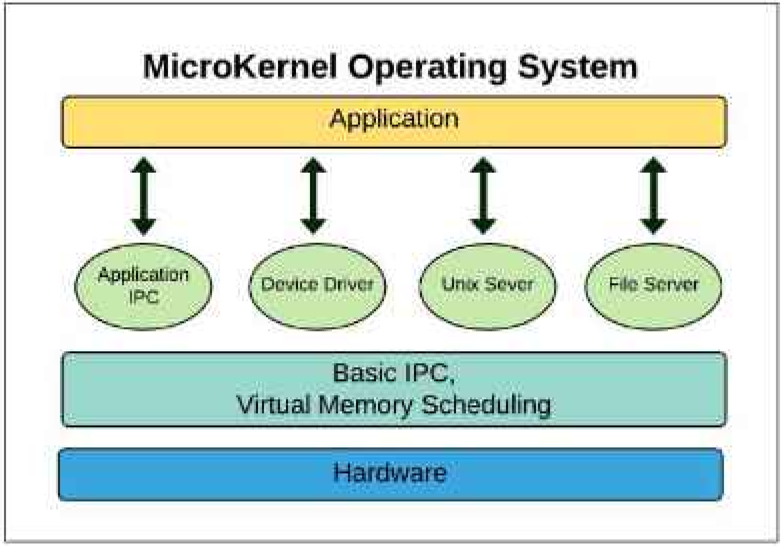
•Provides fast communication.

•Examples include Linux kernel versions before 2.6, Unix, and early versions of Windows.



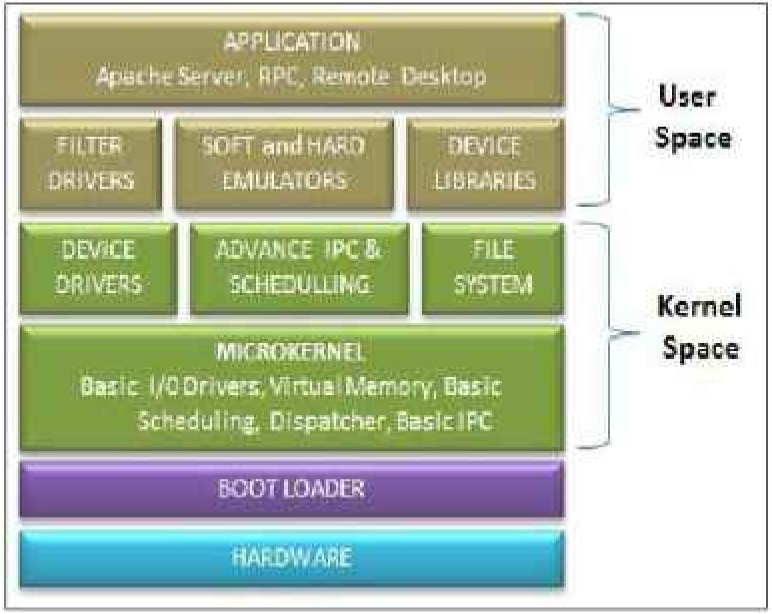
Microkernel:

- Memory management, IPC is been managed.
- Additional functionalities like device drivers, file systems, etc., implemented at user space.
- Small and efficiency, avoid effecting kernel bugs to system stability.
- Examples include QNX, MINIX, and early versions of macOS (XNU kernel).



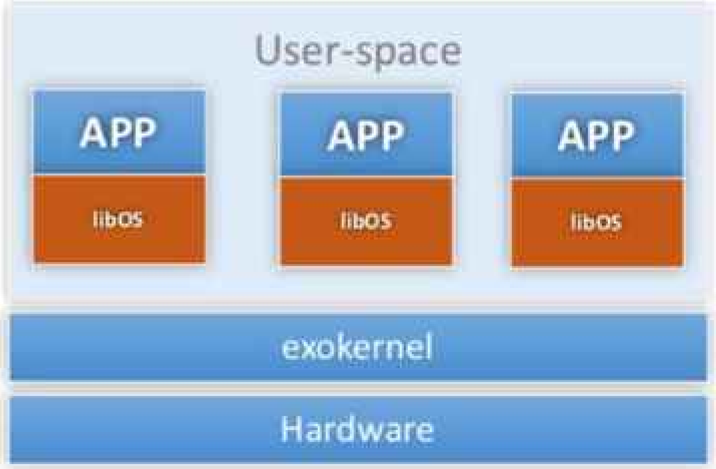
Hybrid Kernel:

- Hybrid kernels combine features of both monolithic and microkernel architectures.
- some in kernel, some in user space.
- This approach allows for flexibility in designing the operating system and optimizing performance without sacrificing stability.
- Examples include Windows NT kernel used in modern versions of Windows, macOS (XNU kernel), and recent versions of the Linux kernel



Exokernel:

- More functionality to applications.
- Apps manage hardware resources.
- Apps gain more control, but security measures must be look out.
- Examples include ExOS and SPIN.

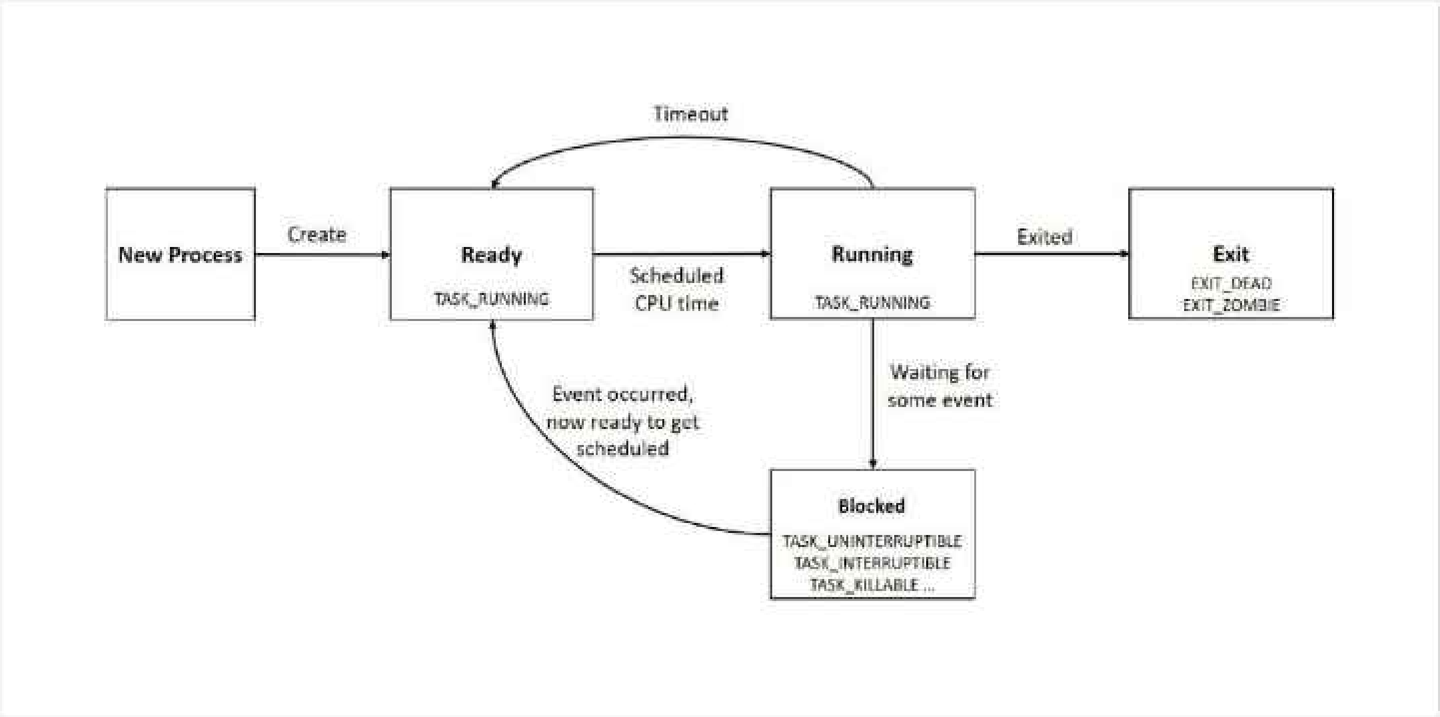


Nano Kernel:

1. Nano kernel is an extremely **lightweight kernel** that provides only the most **basic functionality**, such as **interrupt handling** and **thread scheduling**.
2. It relies heavily on **external components** and libraries to provide higher-level functionality.
3. Nano kernels are often used in embedded systems and real-time operating systems where resource constraints are tight, and performance is critical.

Process Management:

Is a **program** in **execution**, performing assigned **tasks i.e., Instructions**.



States:

- **TASK_RUNNING (0):** The task is either executing or contending (Waiting) for CPU in the **scheduler run-queue**.
- **TASK_INTERRUPTIBLE (1):** The task is in an interruptible wait state; it remains in wait until an awaited condition becomes true, such as the availability of mutual exclusion locks, device ready for I/O, lapse of sleep time, or an exclusive wake-up call. While in this wait state, any signals generated for the process are delivered, causing it to wake up before the wait condition is met. TASK_KILLABLE: This is similar to TASK_INTERRUPTIBLE, with the exception that interruptions can only occur on fatal signals, which makes it a better alternative to TASK_INTERRUPTIBLE.
- **TASK_UNINTERRUTPIBLE (2):** The task is in uninterruptible wait state similar to TASK_INTERRUPTIBLE, except that generated signals to the sleeping process do not cause wake-up. When the event occurs for which it is waiting, the process transitions to TASK_RUNNING. This process state is rarely used.
- **TASK_ STOPPED (4):** The task has received a STOP signal. It will be back to running on receiving the continue signal (SIGCONT).
- **TASK_TRACED (8):** A process is said to be in traced state when it is being combed, probably by a debugger.
- **EXIT_ZOMBIE (32):** The process is terminated, but its resources are not yet reclaimed.
- **EXIT_DEAD (16):** The child is terminated and all the resources held by it freed, after the parent collects the exit status of the child using wait.

Background Process

kthread

kworker/

ksoftirqd/

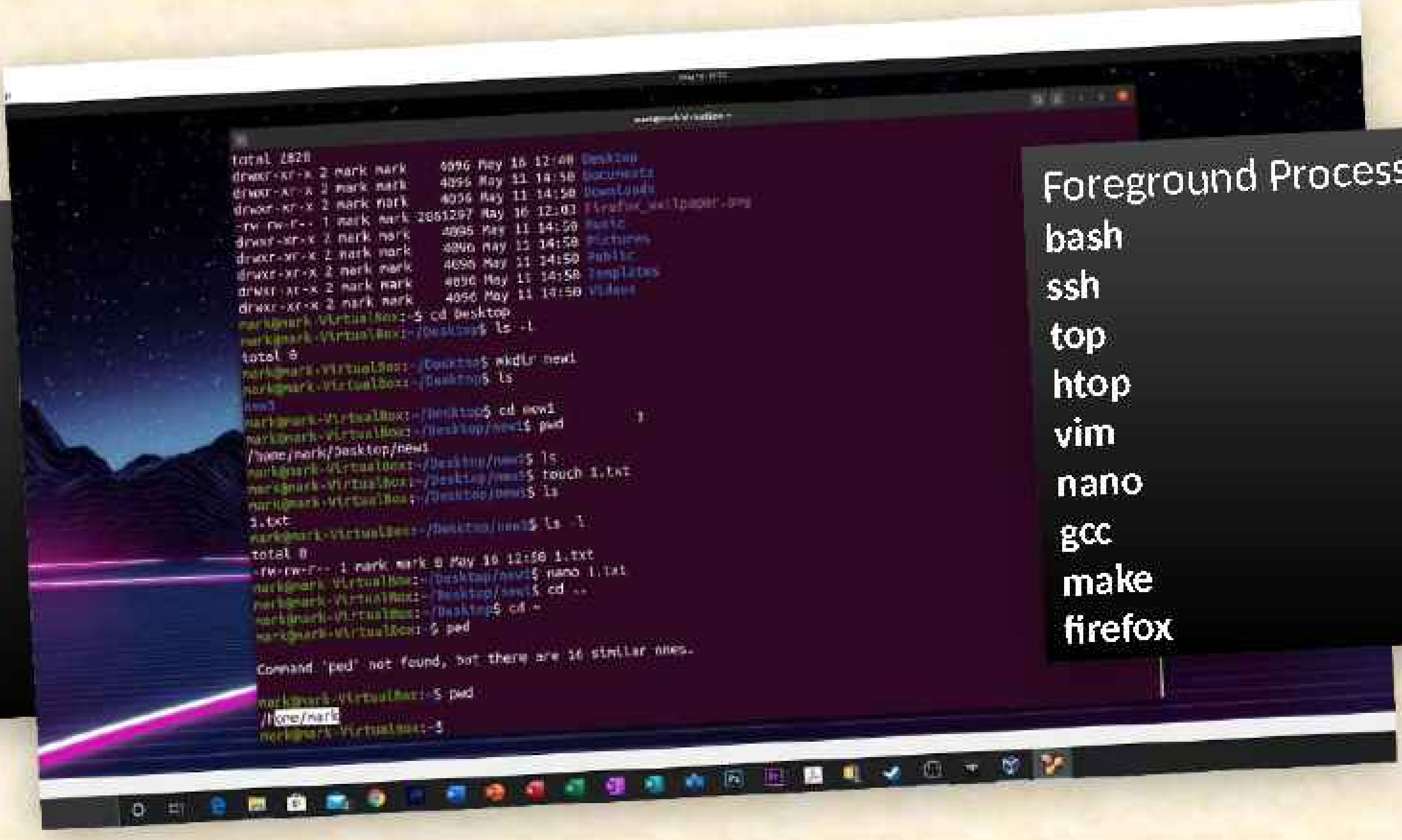
rc/u_sched

rcu_bh

kswapd

ksmd

jbd2



Foreground Process

bash

ssh

top

htop

vim

nano

gcc

make

firefox

Memory management

Memory management is crucial for the efficient operation of a computer system, as it handles the **allocation** and **management** of memory resources.

Key concepts and mechanisms involved:

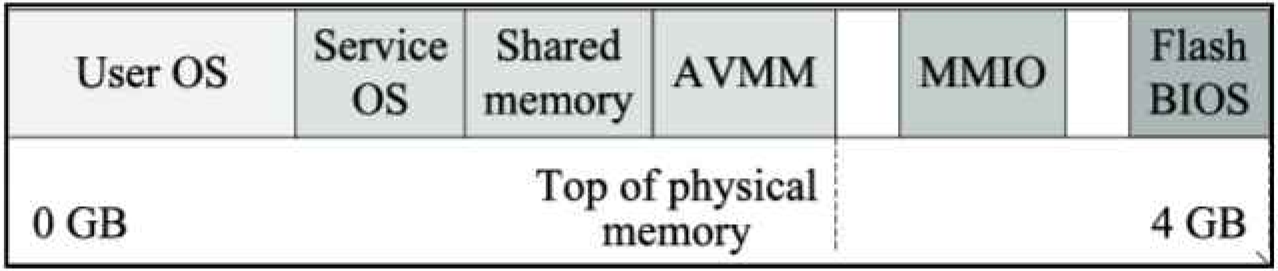
- Physical memory allocation
- Nodes and Zones
- Page allocator
- Buddy System
- Kmalloc & Vmalloc allocations
- Slab allocator
- Contiguous Memory Allocations

Physical Memory Allocation:

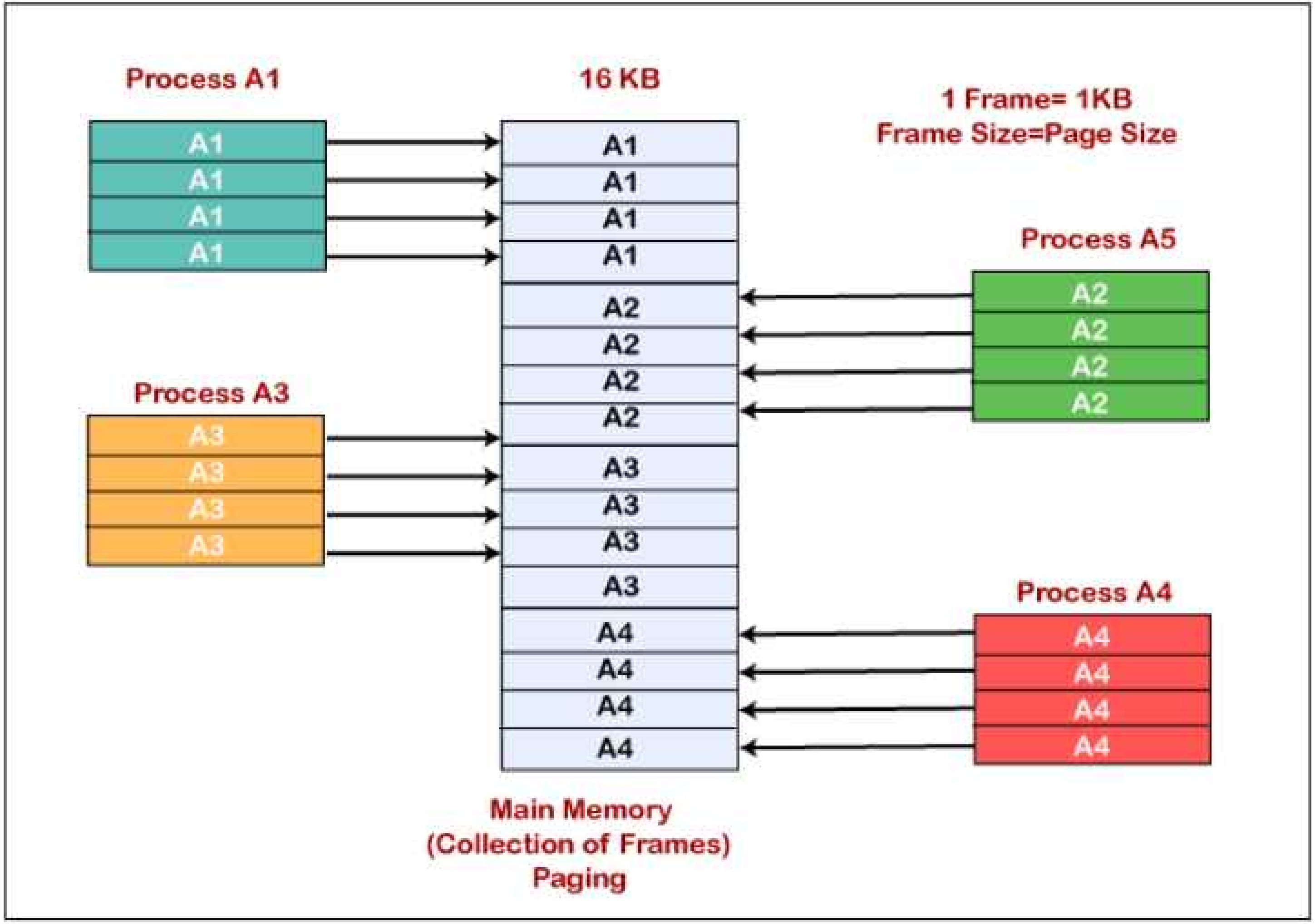
Address Spaces: Memory is divided into physical addresses (actual RAM locations) and virtual addresses (used by applications). The kernel maps virtual addresses to physical addresses.

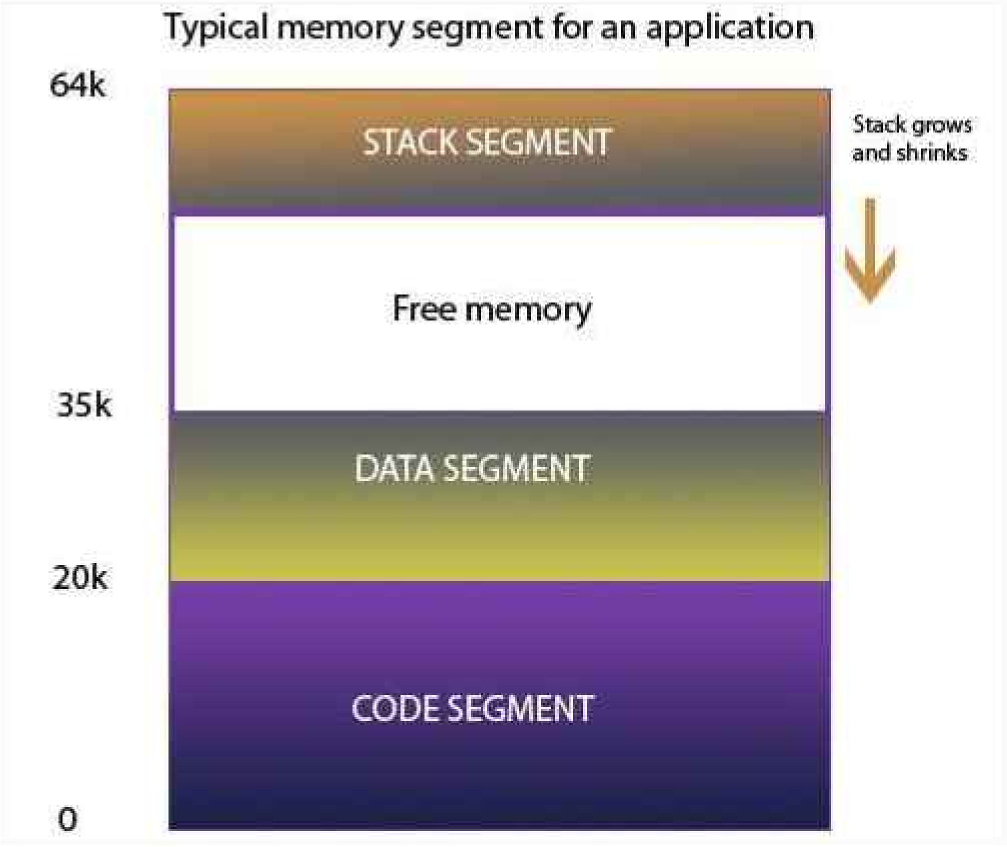
Paging: This technique divides memory into fixed-size pages, which helps in managing memory efficiently and protecting processes from each other.

Segmentation: Although less common now, segmentation involves dividing memory into different segments based on data type (e.g., code, data, stack).



Organization of physical memory


Process A1
16 KB
1 Frame= 1KB
Frame Size=Page Size
A1
A1
A1
A1
A1
A1
A1
A1
Process A5
A2
A2
A2
A2
A2
A2
A2
A2
Process A3
A3
A3
A3
A3
A3
A3
A3
A3
Process A4
A4
A4
A4
A4
A4
A4
A4
A4
Main Memory
(Collection of Frames)
Paging


Typical memory segment for an application
64k
STACK SEGMENT
Stack grows
and shrinks
Free memory
35k
DATA SEGMENT
20k
CODE SEGMENT
0

Nodes & Zones:

Nodes: In NUMA (Non-Uniform Memory Access) systems, memory is divided into nodes, each with its own memory and CPUs.
Zones: Memory is categorized into zones based on accessibility and purpose, such as DMA (Direct Memory Access), Normal, and Highmem zones.

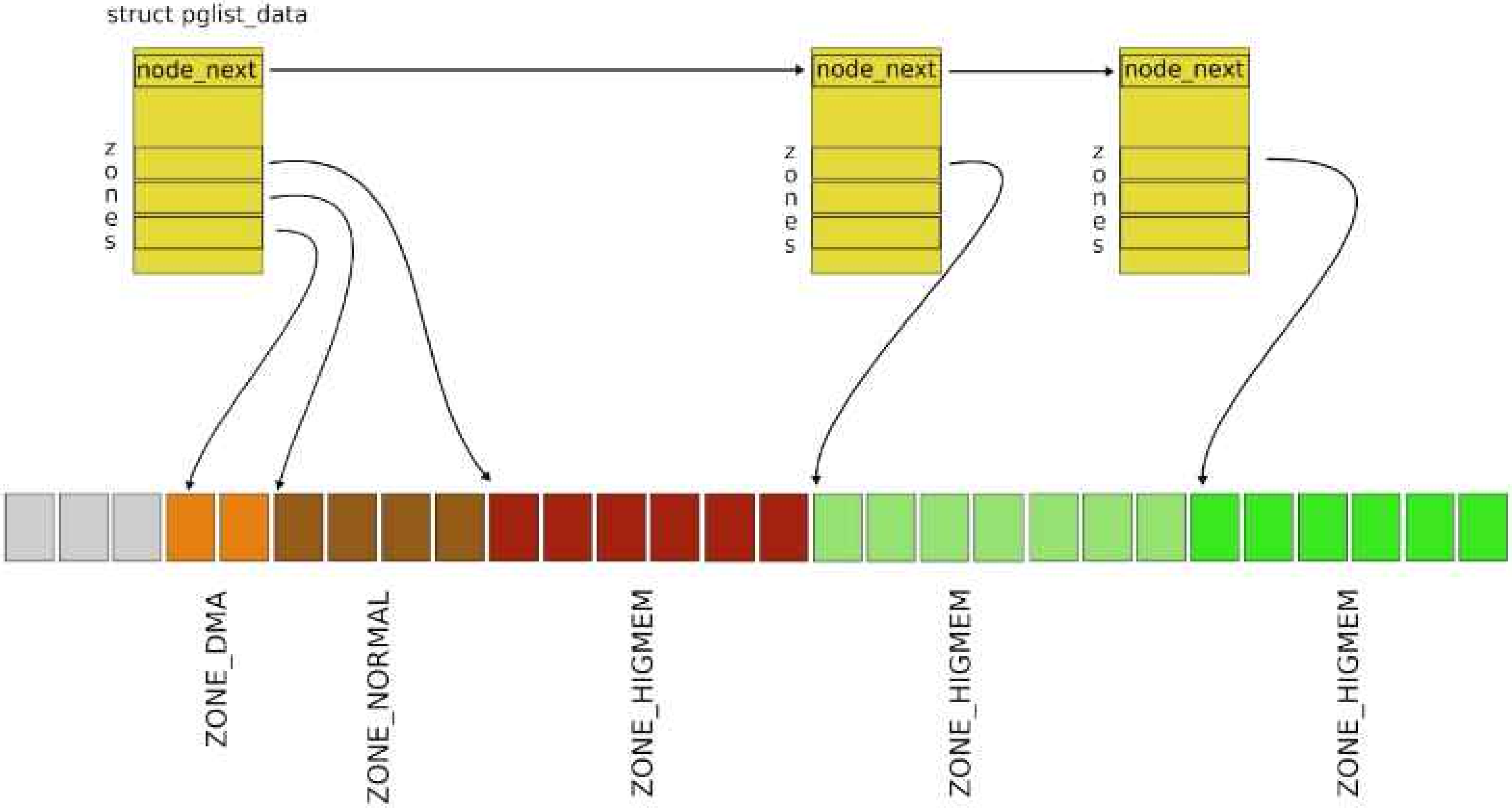
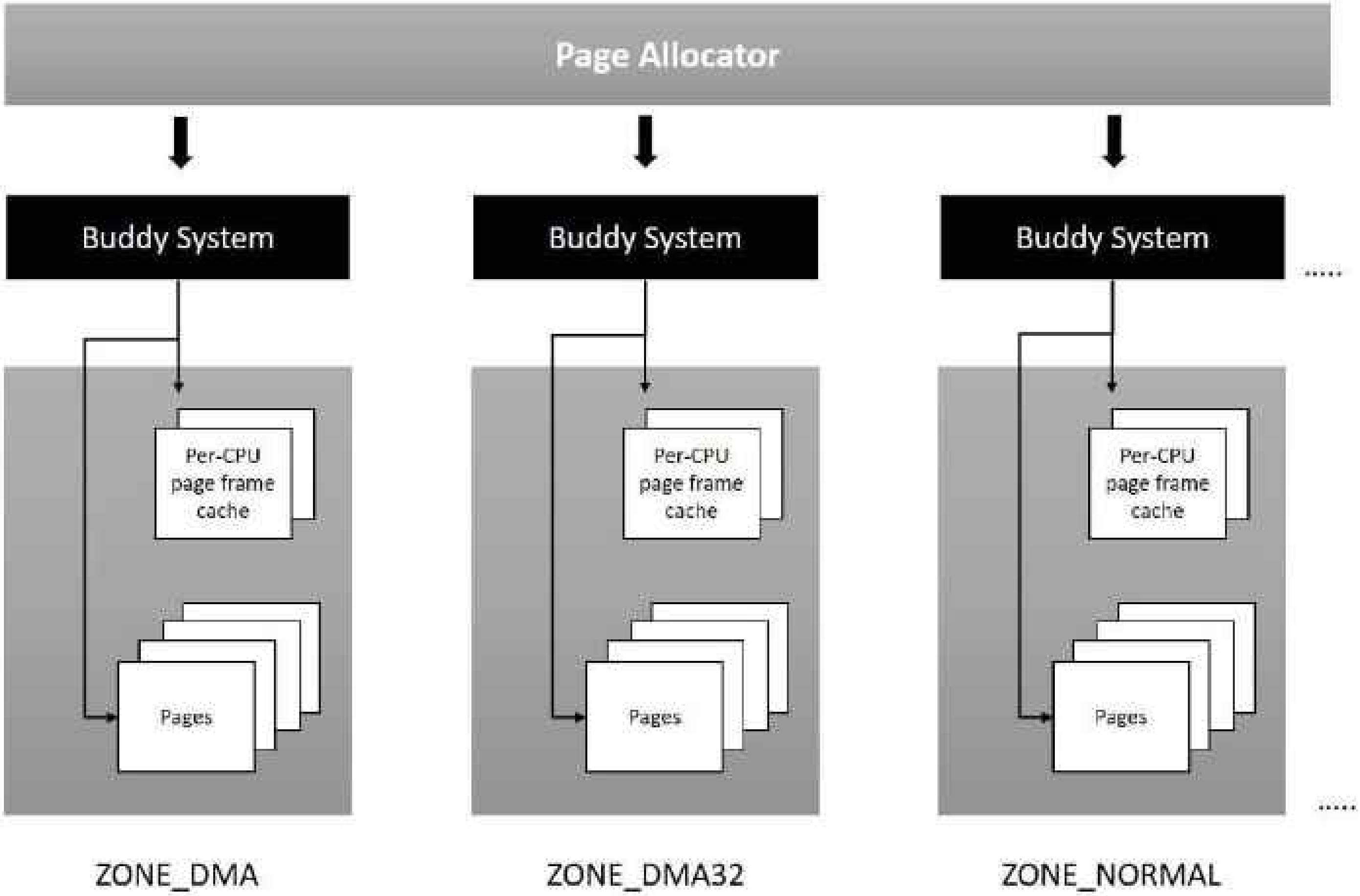


Image representing nodes in memory

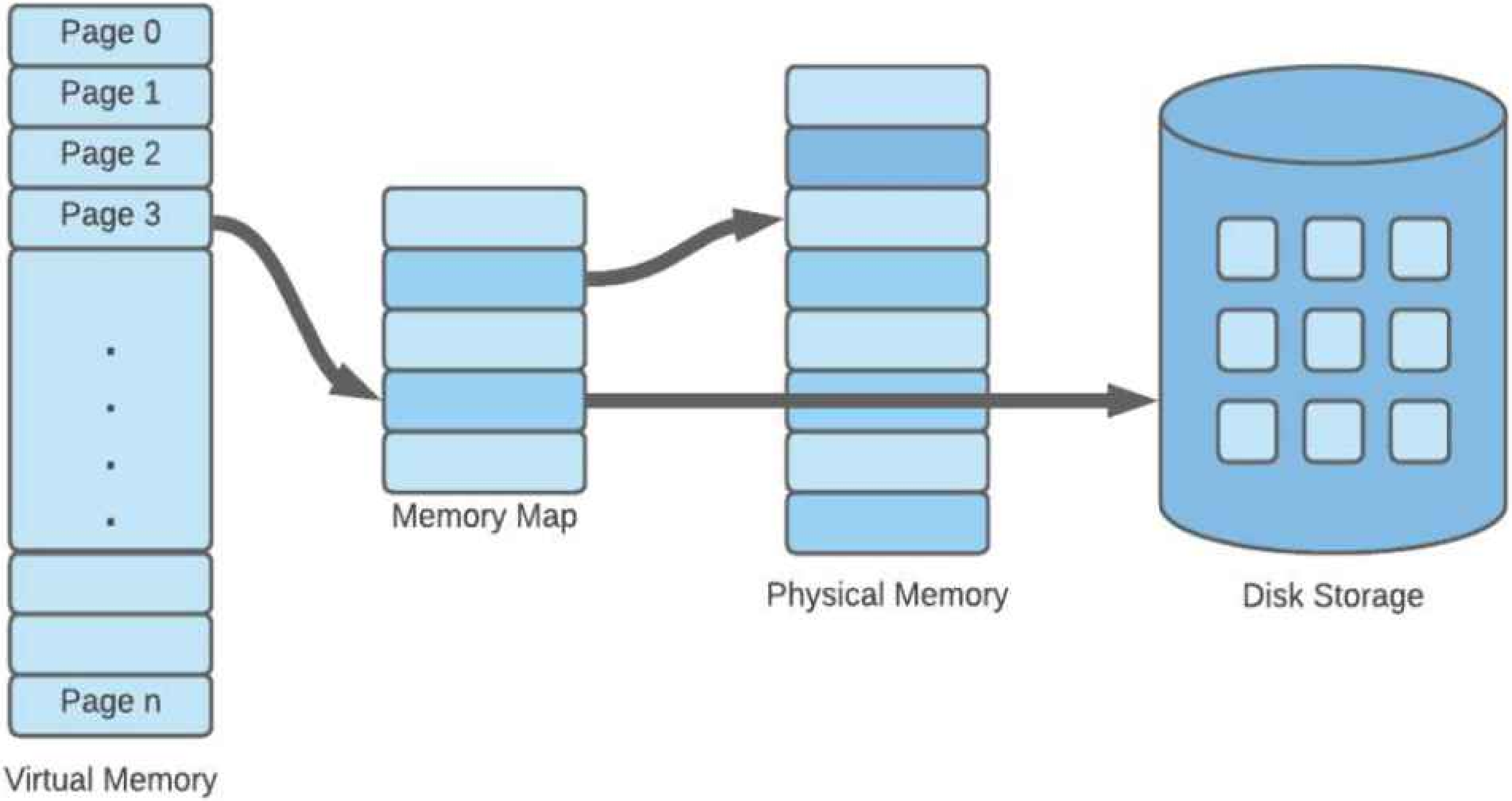
Page Allocator: Responsible for allocating and deallocating pages of memory. It works by keeping track of free pages and efficiently assigning them to processes as needed.
Buddy System: A memory allocation algorithm that divides memory into pairs of blocks (buddies). When memory is freed, it can be merged with its buddy if it is also free, reducing fragmentation.
Slab Allocator: Designed for efficient memory allocation of small objects. It uses caches to store pre-allocated memory chunks, which reduces the overhead of frequent allocations and deallocations.



Page Allocator
Buddy System
Buddy System
Buddy System
.....
Per-CPU page frame cache
Per-CPU page frame cache
Per-CPU page frame cache
Pages
Pages
Pages
.....
ZONE_DMA
ZONE_DMA32
ZONE_NORMAL

Kmalloc Allocations: kmalloc is used for dynamic memory allocation in the kernel. It allocates memory in bytes and is used for small to medium-sized allocations.
Vmalloc Allocations: vmalloc is used for allocating larger, contiguous blocks of memory. It maps non-contiguous physical memory into a contiguous virtual address space.

Contiguous Memory Allocator (CMA): CMA is a specific method used in some operating systems and devices to allocate contiguous memory blocks for specific purposes, such as for device drivers or for specialized tasks. It reserves a pool of physically contiguous memory during system initialization, which can then be allocated as needed by the system or specific components. To achieve this, **Memory Compaction** technique is used.



Page 0
Page 1
Page 2
Page 3
Page n
Virtual Memory
Memory Map
Physical Memory
Disk Storage

• Device drivers

Device drivers are loadable **kernel modules** that manage data transfers b/w kernel and hardware, also allow the kernel to communicate with specific hardware devices, enabling the OS to control and utilize them effectively.

Kernel APIs for Device Driver Management:
APIs (Application Programming Interfaces) are like a set of rules and tools that the kernel and device drivers use to talk to each other. For example, there are specific functions like **request_irq** and **register_chrdev** that the kernel provides for managing device drivers. These functions help the kernel and drivers communicate smoothly.

• File System

The kernel manages file systems by providing a set of core functionalities and interfaces that allow the operating system to interact with storage devices and organize data into files and directories.

Virtual File System (VFS):

- The kernel includes a layer called the Virtual File System (VFS) that abstracts different file systems into a unified interface.
- VFS provides a common set of operations for interacting with files and directories, regardless of the underlying file system format.
- This abstraction allows applications and system components to work with files and directories in a consistent manner, regardless of where the data is stored or how it's formatted.

Kernel Security:

Ensuring the security of the kernel is paramount as it directly impacts the overall security and integrity of the system.

Key Security Challenges:

Kernel-level exploits: Vulnerabilities in the kernel can be exploited by attackers to gain unauthorized access, escalate privileges, or execute malicious code.

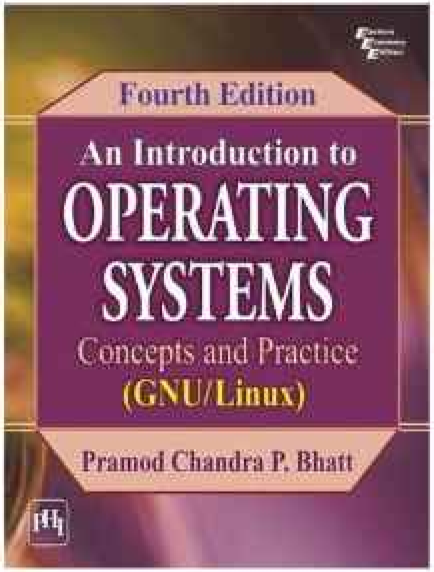
Memory protection: Unauthorized access to kernel memory can lead to data breaches and system compromise.

Denial-of-service attacks: Kernel vulnerabilities can be exploited to disrupt system operations, leading to service interruptions or system crashes.

Kernel Customization

With the source code freely available, it is possible for **users** to make their own version of the kernel.

A user can take the source code select only the parts of the kernel that are relevant to user, and leave out the rest. It is possible, to get a working Linux kernel in single **1.44 MB floppy disk**. One Can modify the source code for a targeted application.



Author: PCP bhatt

Release date: February 13, 2014

This book deals with OS concepts, it's architecture and some projects on OS

THANK YOU!