Kernelogy

Memory Management Process

A Deep Dive into How Linux Handles RAM

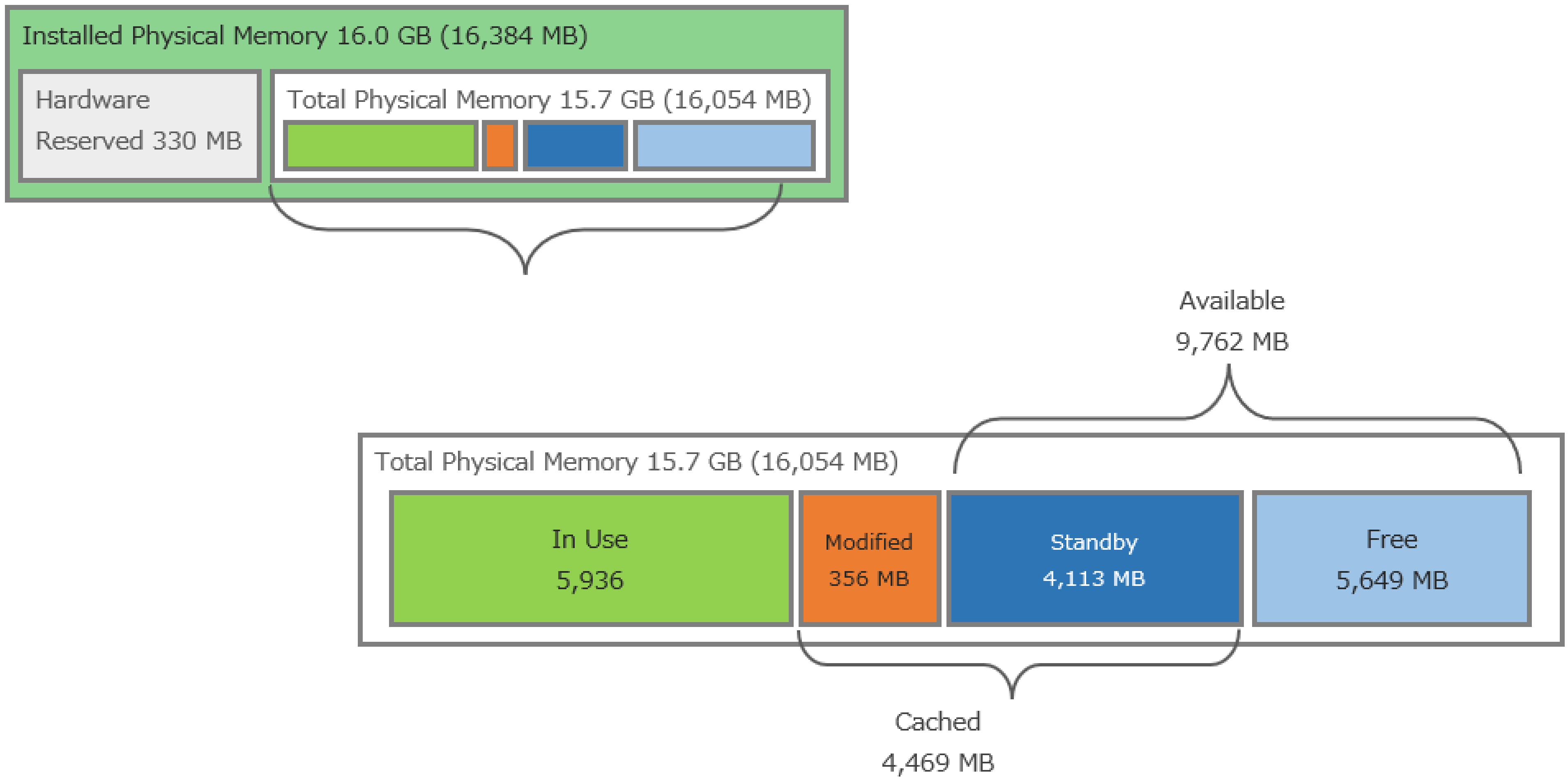
M. Moslemi AbarGhan

Introduction to Memory Management

Introduction to Memory Management

Memory Management is one of the **Core Responsibilities** of the **Linux Kernel**. It serves as the foundation for **Multitasking**, **Application Execution**, and **System Stability**. **The Kernel Abstracts** the **Physical RAM** into **Manageable** and **Protected Virtual Memory** spaces for each **Process**, allowing **Efficient** use and **Isolation** of **Memory Resources**.

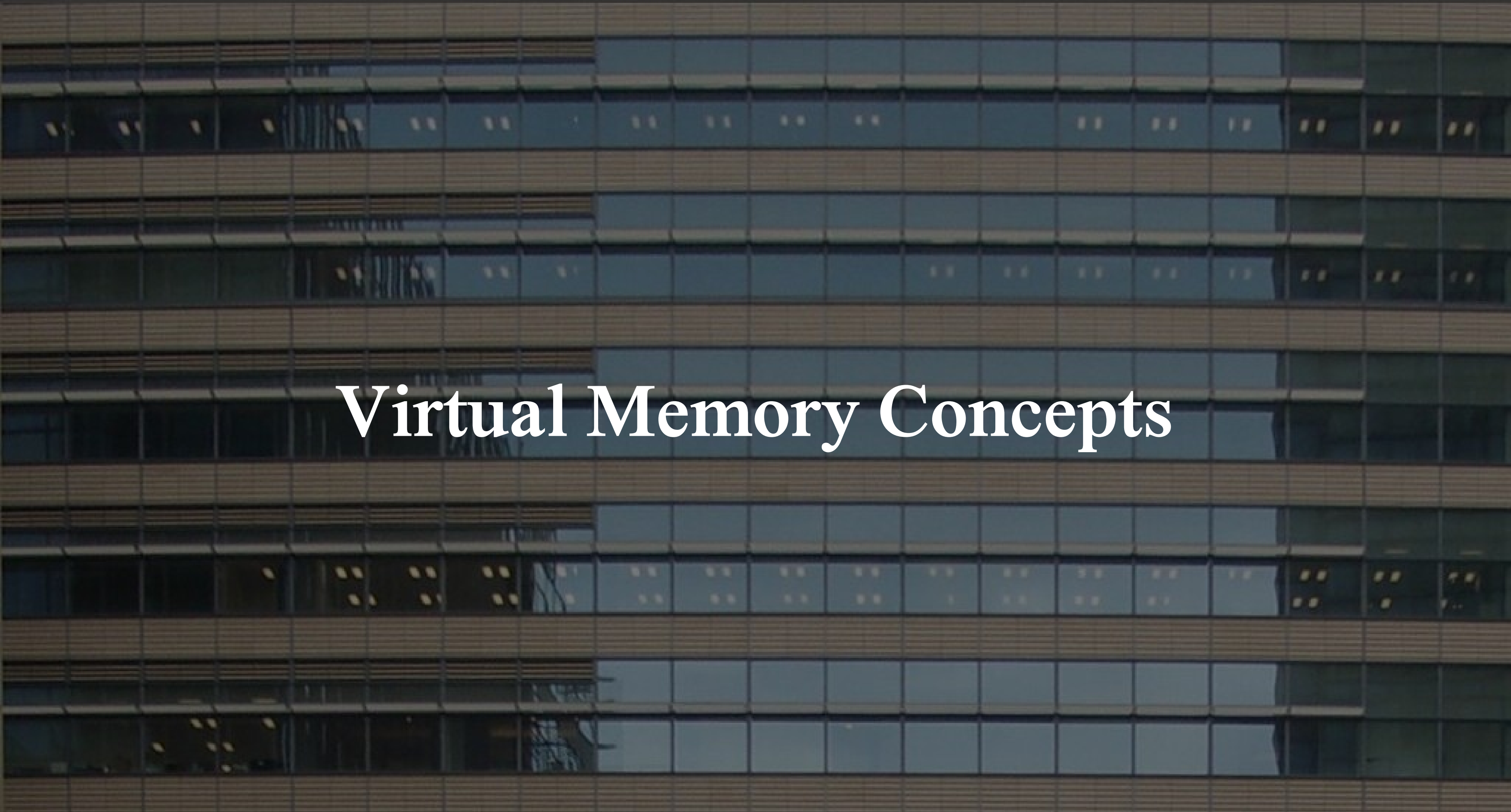
Introduction to Memory Management



Introduction to Memory Management

This Abstraction not only improves **Security** by **Preventing Processes** from **Accessing** each other's **Data** but also allows the **System** to **Simulate** more **Memory** than physically available through **Techniques** like **Swapping** and **Demand Paging**. As a result, memory management **Ensures** that **System Resources** are used **Optimally**, **Processes** are **Kept** in **Check**, and the **Overall Performance** of the **Operating System** remains **Stable**.

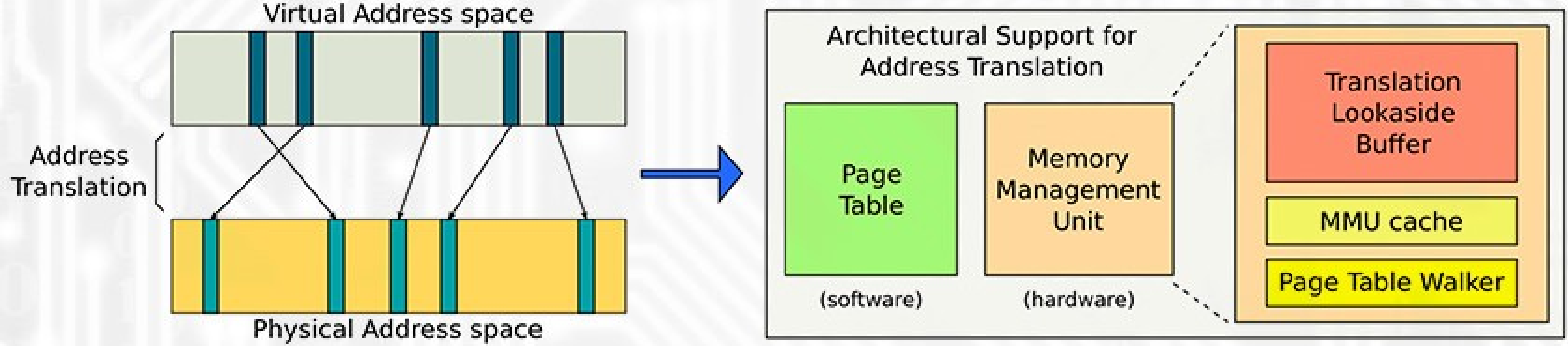
Virtual Memory Concepts



Virtual Memory Concepts

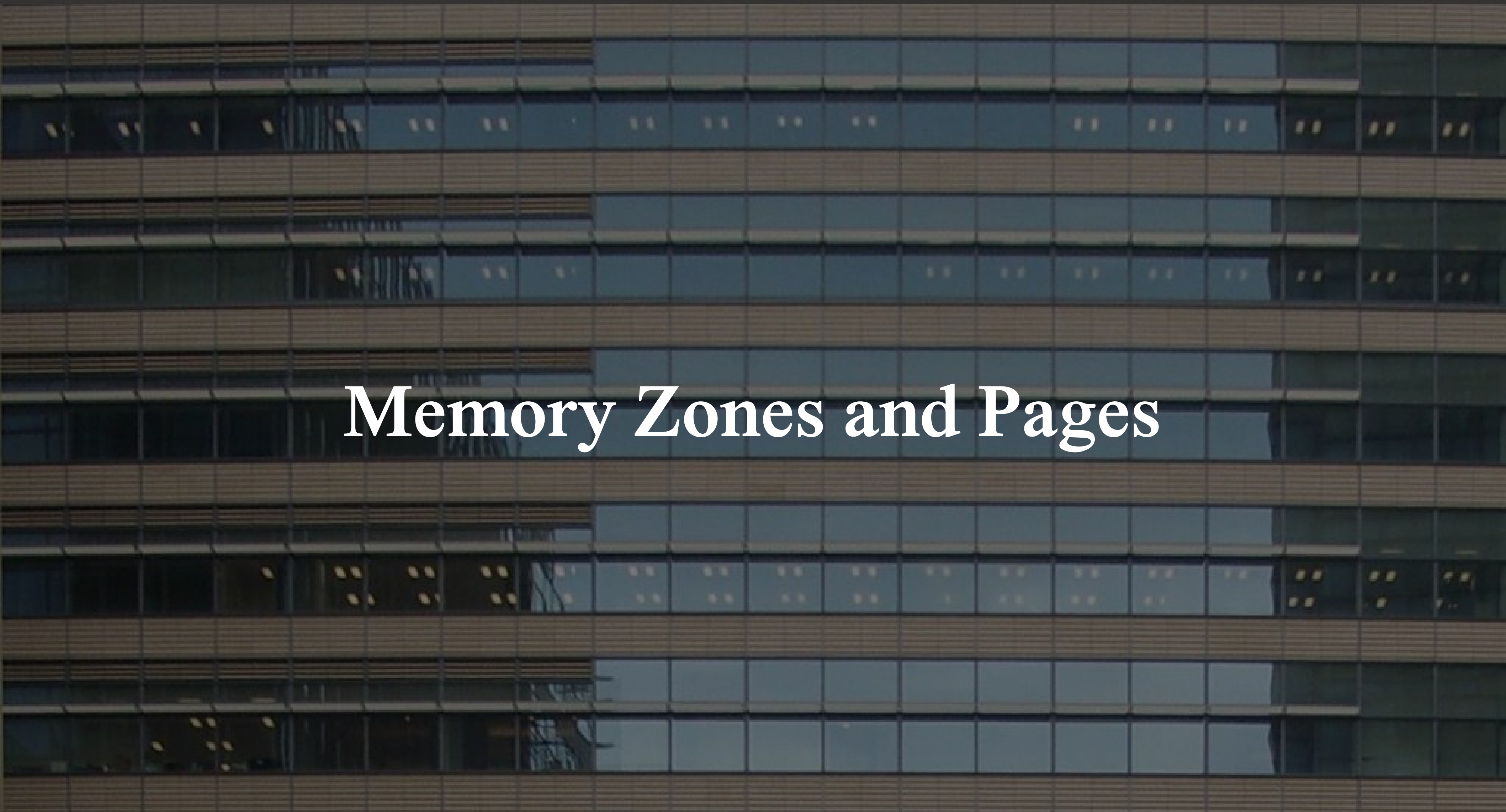
Virtual Memory is a layer of **Abstraction** between the **Physical Hardware** and **User Applications**. In **Linux**, each Process is given the illusion of having its own **Dedicated**, **Contiguous Block** of **Memory**. Behind the scenes, this **Virtual Address** space is **Mapped** to **Physical Memory** through **Page Tables** and managed by the **Memory Management Unit** (**MMU**).

Virtual Memory Concepts



Virtual Memory Concepts

Virtual Memory enables **Crucial** features such as **Memory Isolation** between processes, **Shared Libraries**,, and more. It also allows the **Kernel** to **Swap Inactive Pages** to **Disk** when **RAM** is limited, thereby enabling the execution of programs that collectively use more memory than the **System** physically has. This **Mechanism** is **Essential** for **Modern Multitasking Operating Systems**.



Memory Zones and Pages

Memory Zones and Pages

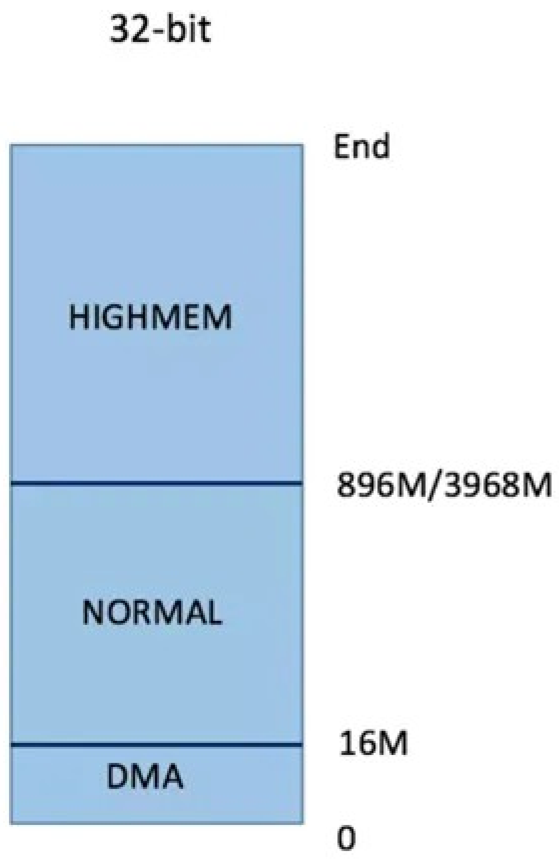
To efficiently handle **Diverse Memory Needs** and **Hardware Constraints**, the **Linux Kernel** divides **Physical Memory** into **Logical Regions** called **Zones**.

1) **ZONE_DMA** (for **Legacy Devices** requiring **Low Memory Addresses**)

2) **ZONE_NORMAL** (directly **Mapped Memory** for the **Kernel**)

3) **ZONE_HIGHMEM** (used in **32-bit Systems** when **Addressable RAM** exceeds **Kernel Mapped Space**).

Memory Zones and Pages



Memory Zones and Pages

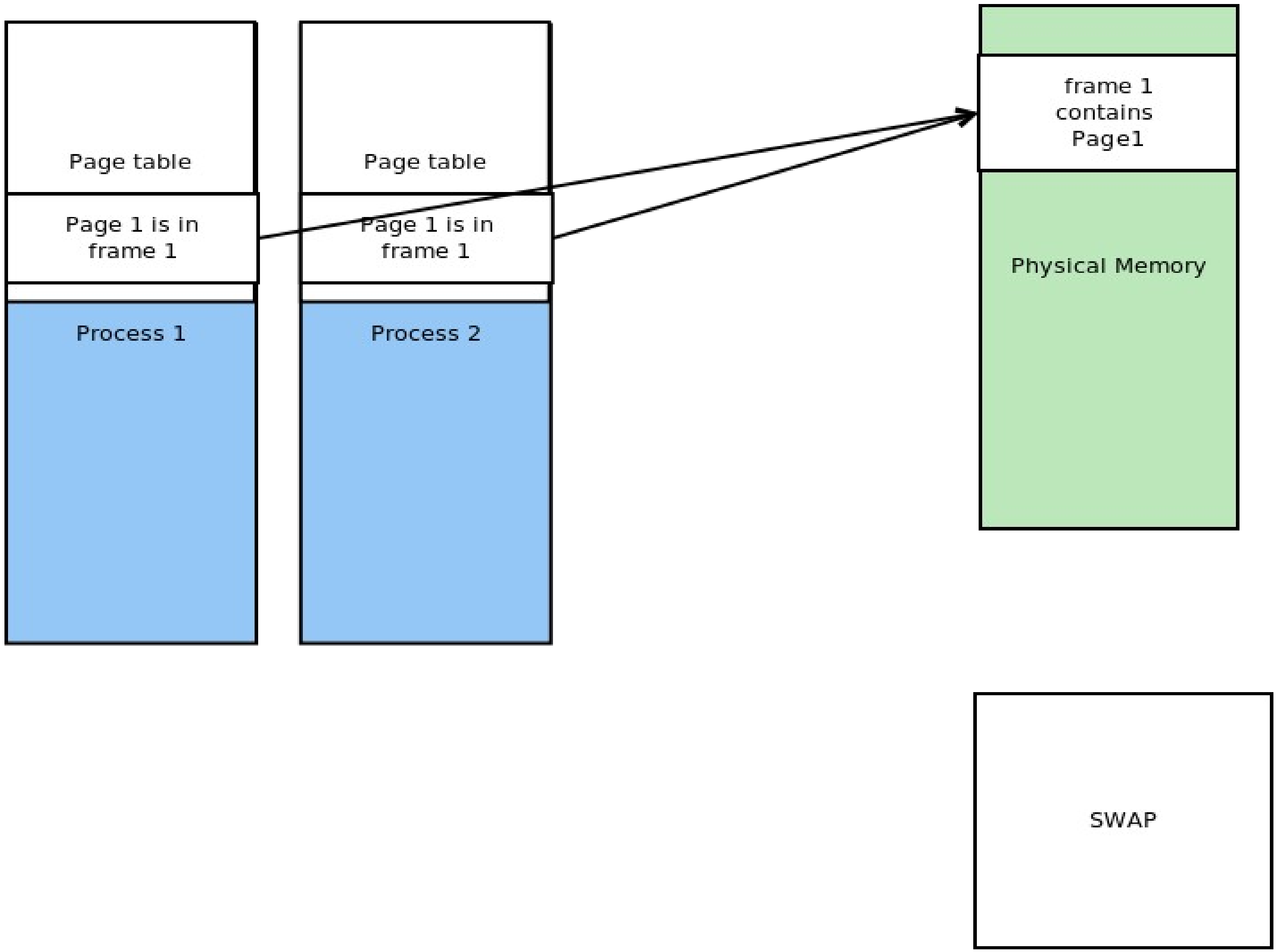
Memory is further subdivided into **Pages**, which are **Typically 4KB** in size. Each page is tracked by a **Struct Page**, which holds **Metadata** about the page's **Status**, **Usage**, and **Ownership**. These **Zones** help the **Kernel** make informed decisions about **Memory Allocation**, especially under pressure.

Page Frame Management

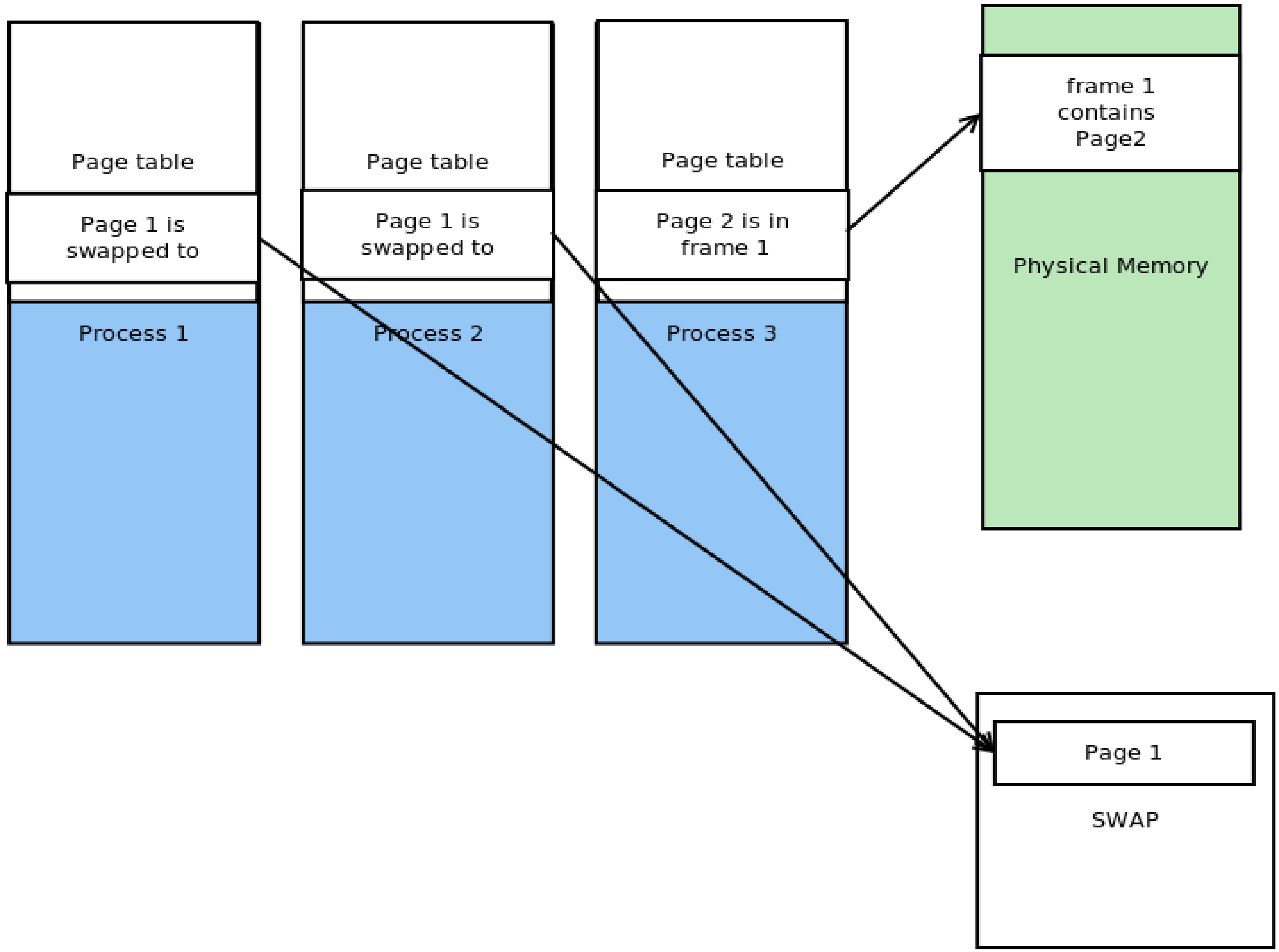
Page Frame Management

Page Frame Management is the **Kernel's** way of keeping **Track** of which physical memory pages (also known as **Page Frames**) are **Free**, **Used**, or **Available** for **Reclamation**. The **Linux Kernel** maintains lists of **Free Pages**, as well as **Active** and **Inactive** page lists for each **Zone**.

Page Frame Management



Page Frame Management



Page Frame Management

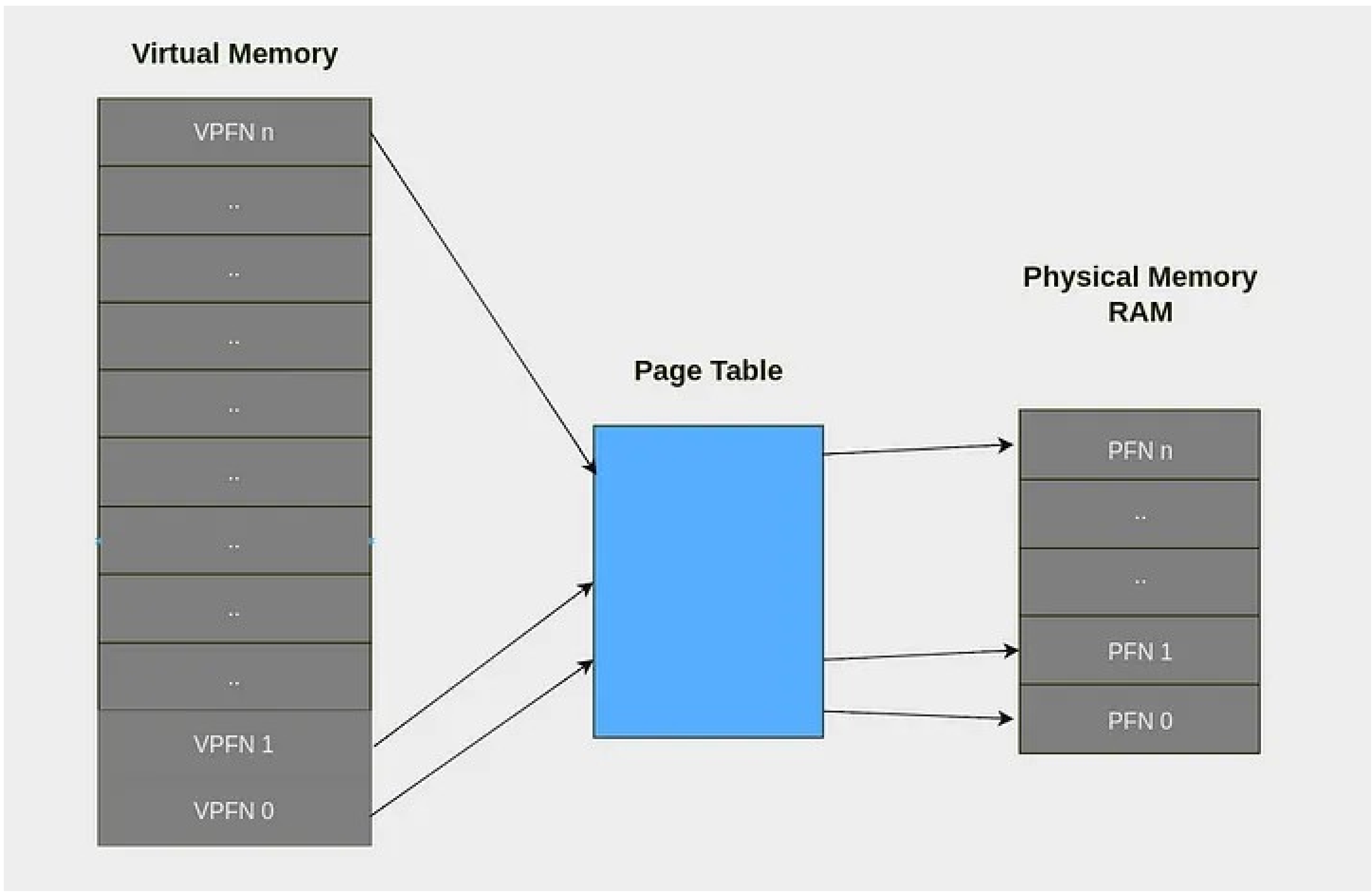
When a page is **Allocated**, it is removed from the free list and assigned to a **Process** or the **Kernel**. When memory becomes **Scarce**, the **Kernel's Page** reclaim **Algorithm** kicks in, **Scanning** and **Freeing Inactive** pages or writing them to swap. **Sophisticated Mechanisms** like **Least Recently Used** (**LRU**) **Caching**, **Page Reference Counting**, and **Page Coloring** help improve **Cache Efficiency** and **Reduce Fragmentation**. **Effective Page Frame Management** is key to maintaining **Performance** and **Stability** in **Memory Intensive Workloads**.

Demand Paging & Page Faults

Demand Paging & Page Faults

Linux uses a **Memory Optimization** technique called **Demand Paging**, where **Memory Pages** are not loaded into **RAM** until they are **Actually Accessed**. When a process tries to access a page that hasn't been mapped into **Physical Memory** yet, the **CPU** triggers a **Page Fault**. The **Kernel** then handles this fault by locating the data (from the **Executable File**, **Shared Library**, or **Swap** space), loading it into a **Free Page Frame**, and updating the **Page Table**.

Demand Paging & Page Faults



Demand Paging & Page Faults

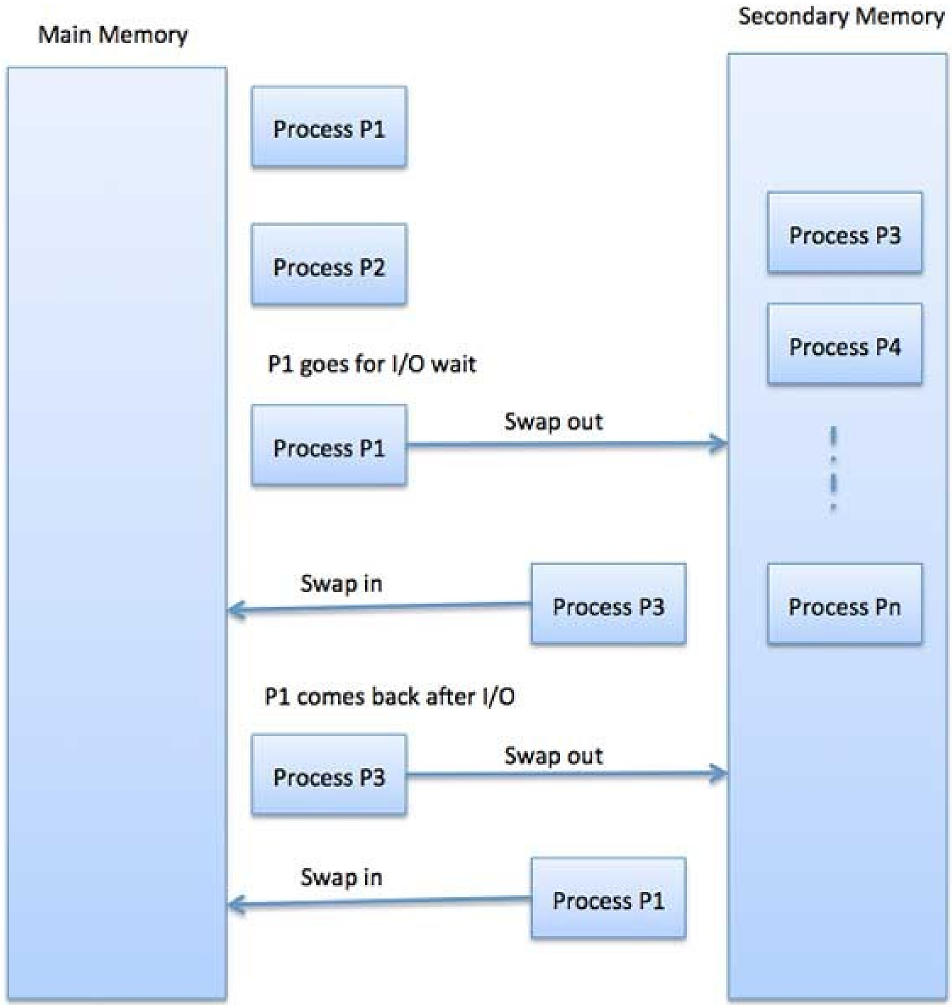
This Mechanism allows the **System** to **Save Memory** and **Load Times** by only loading necessary parts of a program. While most **Page Faults** are handled transparently, frequent or "**Hard**" page faults **—**especially those involving swap**—** can **Slow** the **System Down**. Understanding **Demand Paging** is **Essential** for grasping how **Linux** Balances **Performance** and **Resource Efficiency**.

Swapping & the OOM Killer

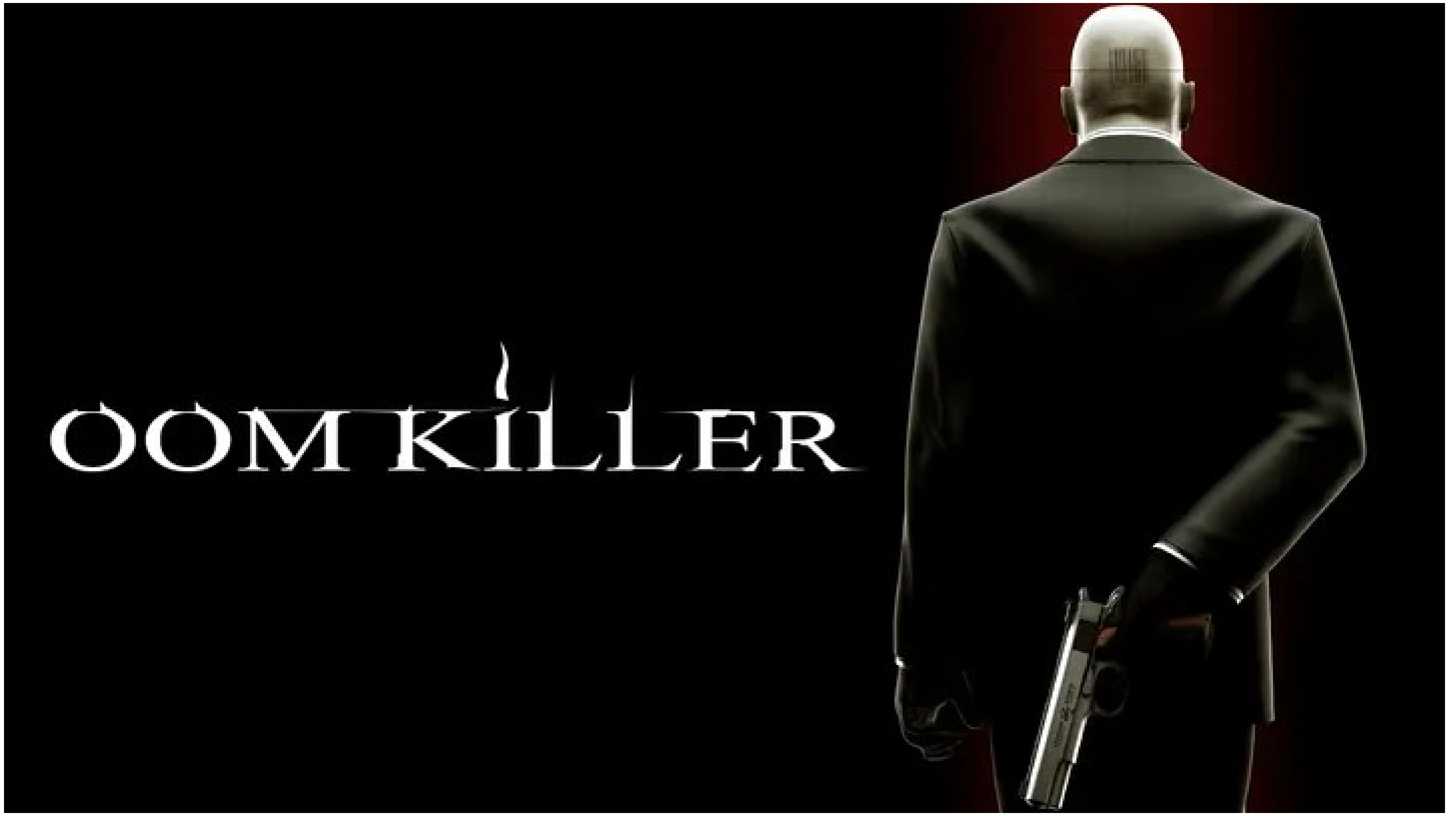
Swapping & the OOM Killer

When **Physical Memory** becomes **Scarce**, the **Linux Kernel** may move less frequently used pages to **Swap Space**, typically located on **Disk**. This process, known as swapping, frees up RAM for more active processes but comes at the cost of performance, as **Disk Access** is much **Slower** than **RAM**.

Swapping & the OOM Killer



Swapping & the OOM Killer



Swapping & the OOM Killer

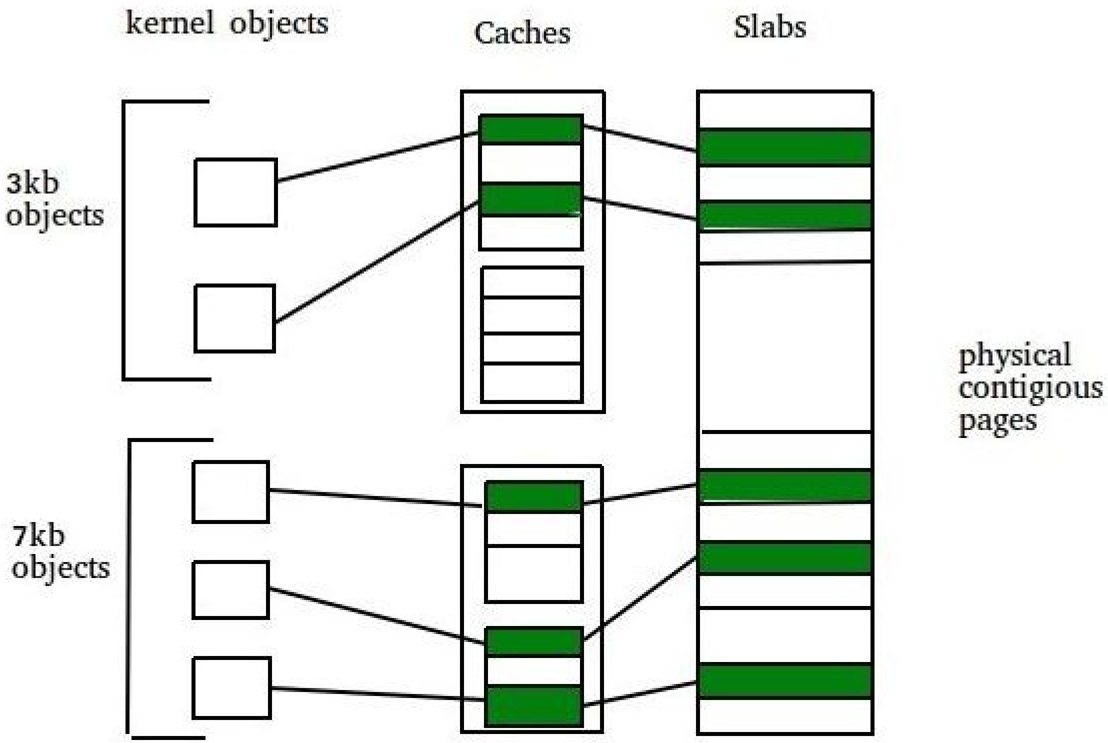
If the **System** runs out of both **RAM** and **Swap**, it invokes the **Out of Memory** (**OOM**) Killer. The **OOM Killer** analyzes **Running Processes** and **Terminates** one or more deemed expendable **—**usually based on **Memory Usage** and **Importance—** to **Reclaim Memory** and keep the **System** from **Crashing**. Although undesirable, this mechanism is a necessary safety net to ensure **System Survival** under **Extreme Memory Pressure**.

Memory Allocator & Kernel Caches

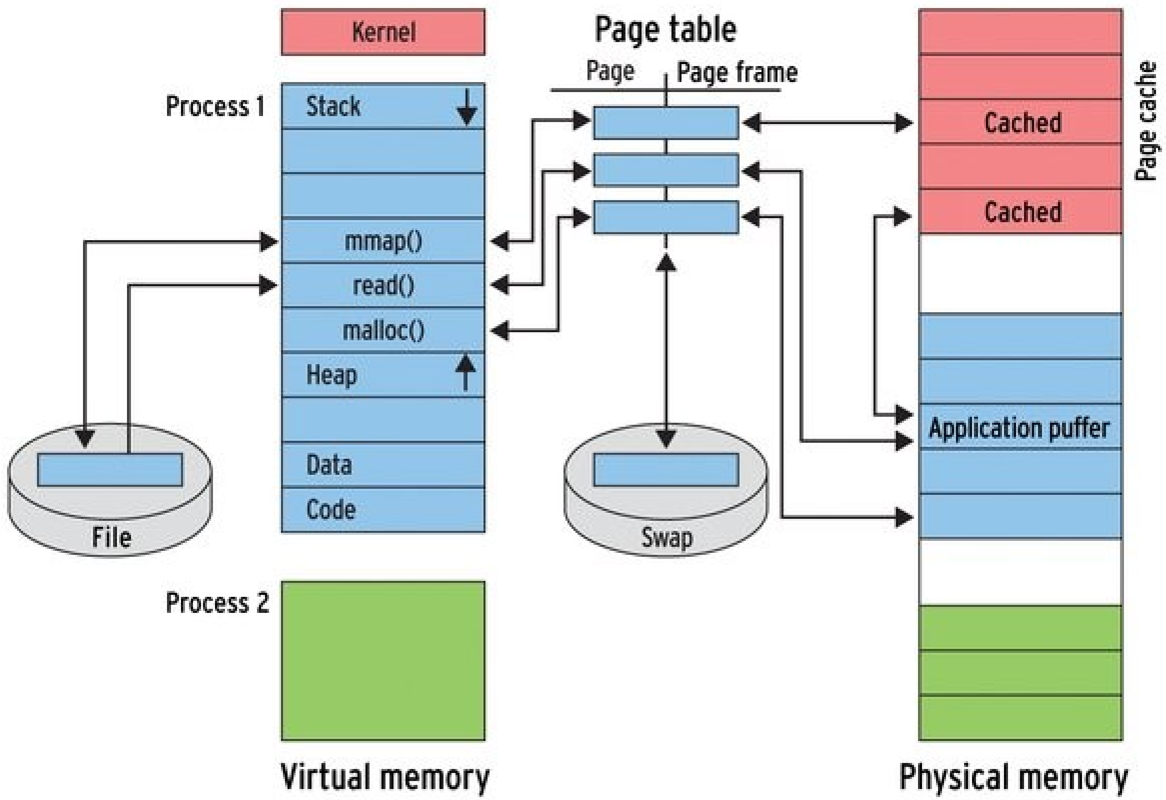
Memory Allocator & Kernel Caches

The **Linux Kernel** frequently **Allocates** and **Deallocates Small Memory Objects**, such as **File Descriptors**, **Inodes**, or **Task Structures**. To manage these efficiently, it uses the **Slab Allocator —**a **Caching System** that **Preallocates Memory Chunks** of **Fixed Sizes**. This avoids **Fragmentation** and **Reduces** the **Overhead** of repeated allocation.

Memory Allocator & Kernel Caches

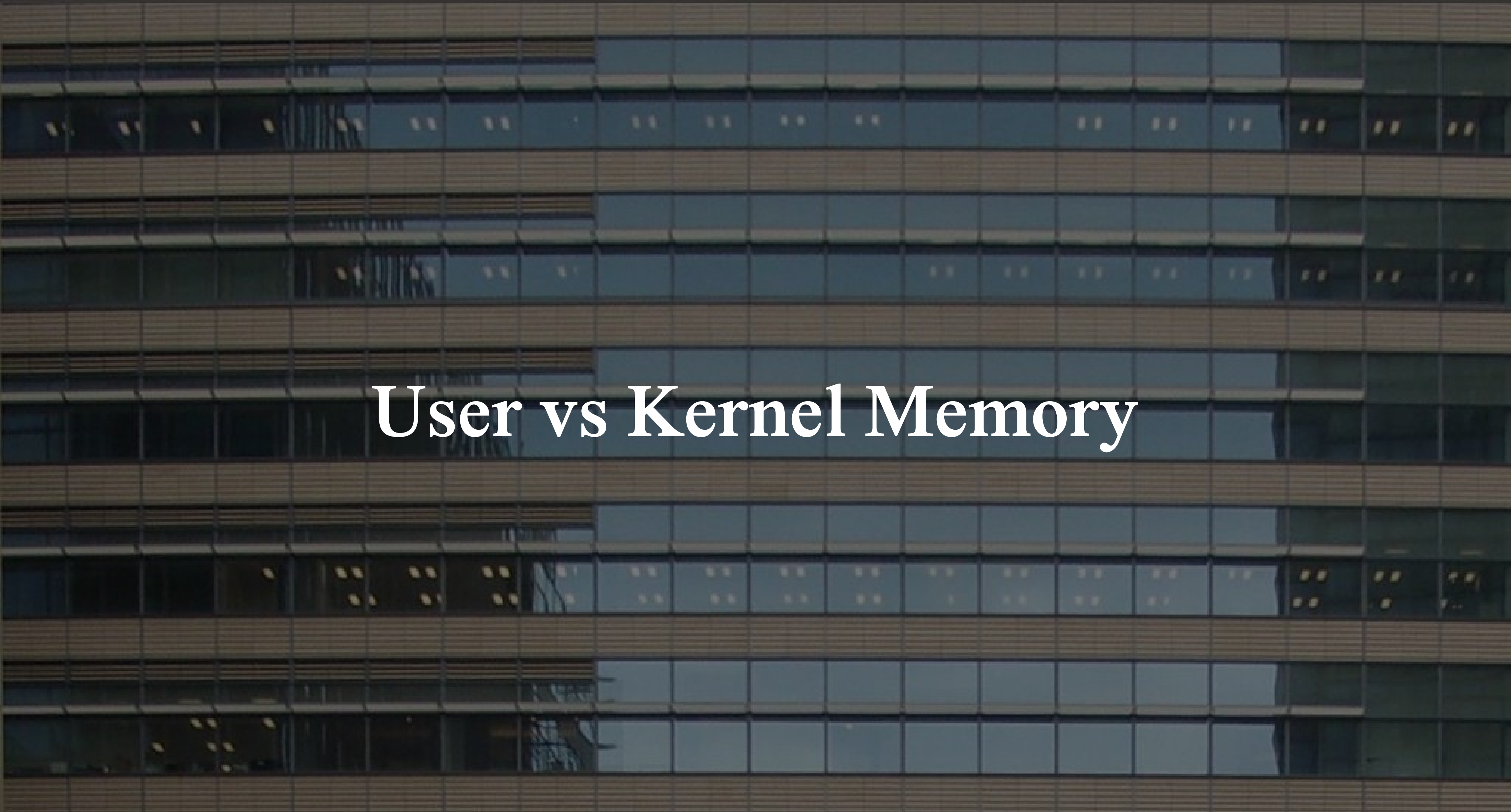


Memory Allocator & Kernel Caches



Memory Allocator & Kernel Caches

The **Kernel** maintains various **Caches** for different **Object Types**, which are reused instead of being repeatedly allocated and freed. **Slab**, **Slub**, and **Slob** are different allocator implementations available in the **Kernel**, each optimized for different **Scenarios**. These **Caching Techniques** significantly boost **Performance** and **Stability** in high load environments by minimizing **Dynamic Memory Management Overhead**.

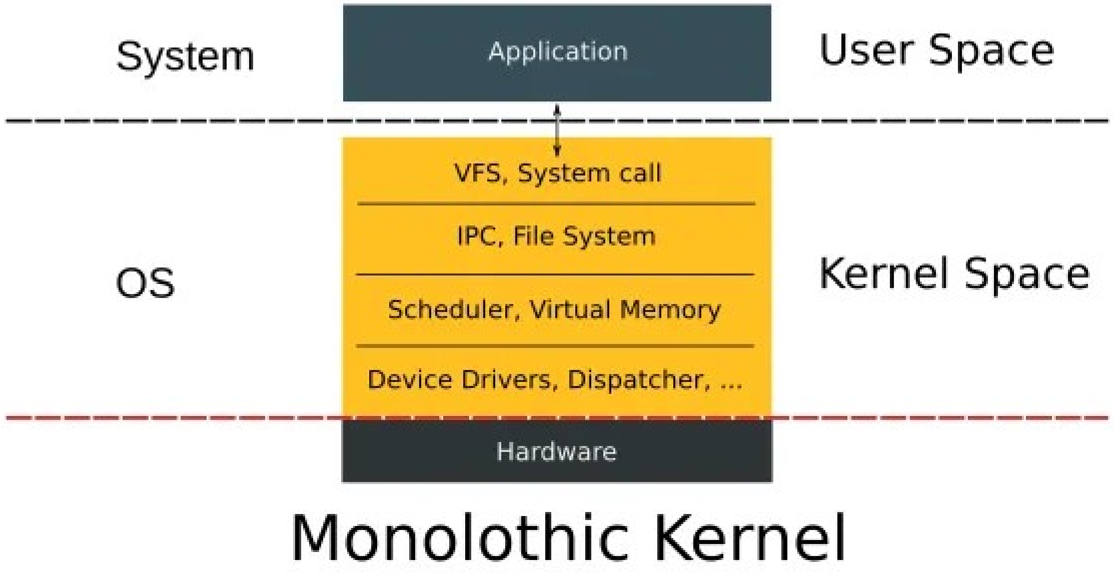


User vs Kernel Memory

User vs Kernel Memory

In Linux, Memory is divided between User Space and **Kernel Space**. User space is where **Regular Applications** run; it is **Isolated** and **Protected** to prevent one process from interfering with another or with the **Kernel** itself.

User vs Kernel Memory



User vs Kernel Memory

Kernel Space is reserved for **Privileged Code**, such as **Device Drivers**, **File Systems**, and the **Core Kernel**. It has **Unrestricted Access** to **Hardware** and the **Entire System's Memory**. The distinction ensures both **Security** and **Stability**.

References

- https://www.wikipedia.org/
- https://docs.kernel.org/
- https://ion-utale.medium.com/how-linux-uses-ram-d88abdaa4ef8
- https://codeahoy.com/learn/computersos/ch4/
- https://users.cs.fiu.edu/~cpoellab/teaching/cop4610_fall22/project4.html
- https://www.spiceworks.com/tech/devops/articles/what-is-virtual-memory/

[Thanks for your Attention]