Memory Management: Organization

by ***AZZI Hadjer***

Outline

- **Introduction**
- **What is memory management?**
- **Levels of memory**
- **User and Kernel spaces**
- **Logical and Physical Address Space**
- **Memory Management Unit (MMU)**
- **Page tables:**
    - Storage methods
    - Hardware implementaion
    - Structures
- **Conclusion**

Introduction

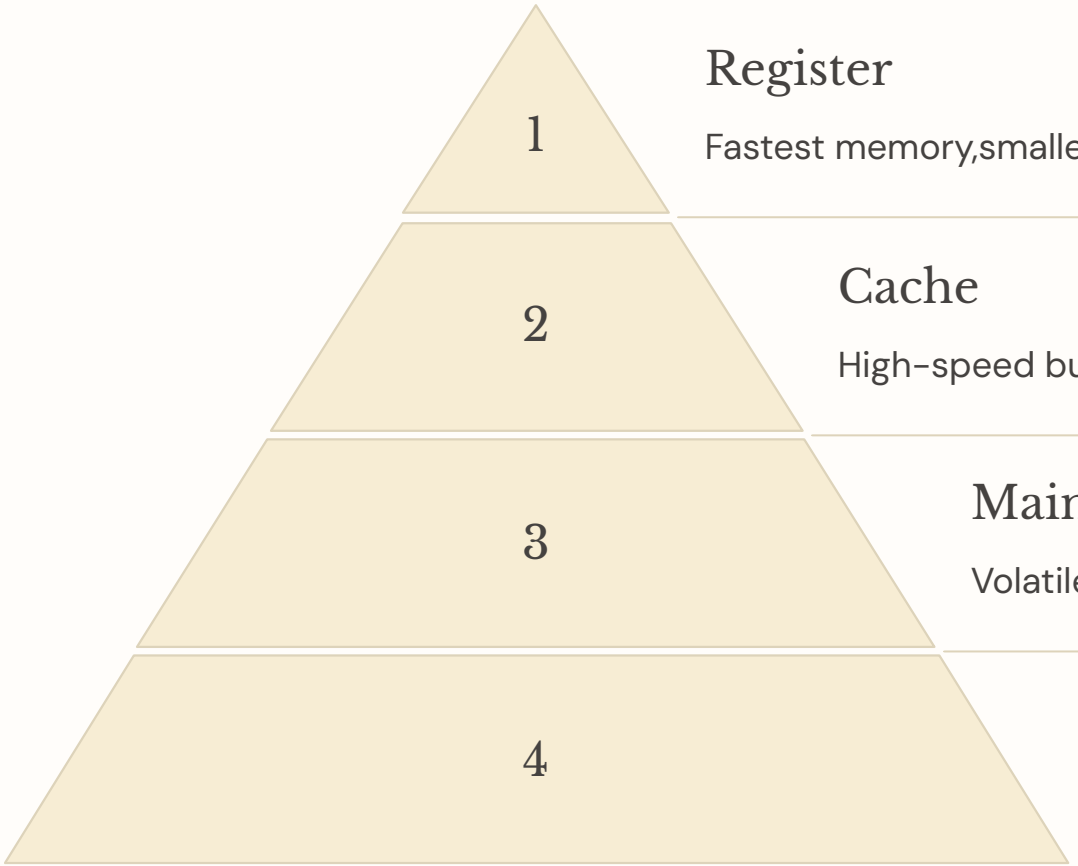
Efficient memory management optimizes system performance by dividing memory into **user space** and **kernel space**. Page tables map virtual addresses to physical memory, ensuring process isolation and efficient allocation. This presentation covers memory organization, user/kernel space roles, and the function of page tables in memory management.

1. What is Memory Management?

In a multiprogramming computer, the operating system resides in a part of memory, and the rest is used by multiple processes. The process of dividing memory among these processes is called **Memory Management**. It is a method used by the operating system to manage the interaction between main memory and disk during process execution. The main goal of memory management is to ensure efficient use of memory.

ref: **Memory Management in Operating System - GeeksforGeeks**

2. Levels of Memory



Register

Fastest memory,smallest capacity, directly accessible by the CPU.

Cache

High-speed buffer between CPU and main memory.

Main Memory (RAM)

Volatile, for active processes.

Secondary Storage

Non-volatile, for long-term storage.

ref: [Memory Hierarchy Design and its Characteristics - GeeksforGeeks]

3.User and Kernel spaces:

All the modern computer operating systems generally divide the virtual memory into two following spaces:

- Kernel space
- User space

The primary reason for this separation is to protect memory and hardware from malicious or third-party software's errant behavior.

ref: **Userspace vs Kernel Space: A Comprehensive Guide ✨ | by Neel Shah | MediumWhat is the difference between the kernel and user spaces?**

3.1 User space:

The **user space** is also known as userland and is the memory space where all user applications or application software executes. Everything other than OS cores and kernel runs here.

- Processes in user space **cannot** directly access hardware or kernel resources (like memory management, hardware devices, etc.) for safety and security reasons.

3.2 Kernel space:

The memory space where the core of the operating system (kernel) executes and provides its services is known as **kernel space**. The kernel has full access to the hardware and can manage low-level operations like memory management, device drivers, and process scheduling.

- It acts as a **bridge** between user processes and the physical hardware.

3.3 Role of the Kernel in Process Management

- The kernel manages **user processes** in user space and ensures that they run **independently** without interfering with each other.

Key functions of the kernel in process management include:

  - **Allocating resources** like CPU time, memory, and I/O.
  - **Scheduling** processes, ensuring fair execution.
  - **Isolation** between processes to prevent one process from crashing or tampering with another.

3.4 System Calls: Accessing Kernel Space

- **System calls** allow user processes to interact with the kernel. They act as the **interface** between user space and kernel space.
- When a user process needs to perform an action that requires kernel privileges (like reading from a file, allocating memory, or interacting with hardware), it uses a **system call**.
- For example, functions like `open()`, `read()`, and `write()` are system calls that allow user processes to access resources managed by the kernel.
- System calls are **safe** because they are controlled by the kernel, ensuring that user processes can't directly access or manipulate critical system resources.

3.5 Bridging User and Kernel Spaces

System Calls

Controlled interfaces allow user programs to access kernel-managed resources and services in a safe and supervised manner.
System calls or interrupts allow controlled communication between user and kernel spaces.

Separation of Concerns

Isolating user and kernel spaces ensures :

Stability: ensures system reliability by isolating user errors.
Security: prevents user applications from corrupting OS operations.

Performance: allows the OS to manage resources efficiently.

4. Logical and Physical Address Space

Logical Address

An address generated by the CPU is known as a “Logical Address”. It is also known as a **Virual address.** Logical address space can be defined as the size of the process.

Physical Address

An address seen by the memory unit (i.e the one loaded into the memory address register of the memory) is commonly known as a “Physical Address” is also known as a Real address.

ref: **Memory Management in Operating System - GeeksforGeeks**

5. Memory Management Unit (MMU):

- A hardware unit that translates **virtual addresses** to **physical addresses** during memory access.

Whenever a page has to be accessed by CPU using the logical address, it requires physical address for accessing the page. Logical address comprises of two parts: Page Number and Offset.

ref: **Paging in Operating System - GeeksforGeeks**

6. Page Table :

The **page table** is a data structure used by the **operating system** to manage virtual memory. It keeps track of the mapping between **virtual pages** and **physical frames** in **main memory**.

- The page table enables the **Memory Management Unit (MMU)** to translate virtual addresses to physical addresses by providing the mapping for each virtual page to the corresponding physical frame in RAM.
- **Content**:
  - Each **entry** in the page table contains the **frame number** that corresponds to a **virtual page**.
  - The **frame number** represents the physical memory location where the virtual page resides.

When a program accesses a virtual address, the MMU uses the page table to find the corresponding **physical frame**, allowing it to generate the **physical address**.

ref : [Operating System Concepts 9th Edition by Abraham Silberschatz.pdf - Google Drive]

6.1 Storing page tables methods:

there are two methods, including:

Per-Process Page Tables

- **Description**: Each process has its own page table.
- **Storage**: A pointer to the page table is stored in the process control block (PCB) along with other register values.
- **Switching**: During a context switch, the dispatcher reloads user registers and updates hardware page-table values.
- **Advantages**: Simple and process-specific mapping.
- **Disadvantages**: Overhead during context switching.

Shared Page Tables:

- **Description**: Only one or a few page tables are maintained for the entire system.
- **Advantages**: Reduces overhead during context switching.
- **Disadvantages**: Potentially less flexible compared to per-process page tables.

ref : [Operating System Concepts 9th Edition by Abraham Silberschatz.pdf - Google Drive]

6.2 Hardware Implementation of Page Tables:

a. Using dedicated high-speed hardware registers.

b. Main Memory with a Page-Table Base Register (PTBR)

c. Translation Lookaside Buffer (TLB)

ref : Operating System Concepts 9th Edition by Abraham Silberschatz.pdf - Google Drive

a. Dedicated high-speed registers:

in this method, the page table entries are stored in high-speed hardware registers

Page Table Setup

1

Load page table entries into dedicated hardware registers during process initialization.

Registers are populated by the operating system.

Address Translation:

2

When the CPU generates a logical address:

- Extract the page number from the logical address.
- Use the page number as an index to access the corresponding page table entry in the register.

Memory Access:

3

- Obtain the frame number from the page table register.
- Combine the frame number with the page offset to calculate the physical address.

Execution:

4

Access the desired memory location using the physical address.

Context Switching:

5

During a process switch, reload the dedicated registers with the new process's page table entries.

b. kept in mamin memory:

This method stores the page table in main memory and uses a register to hold the base address of the page table:

Page Table Setup

1

- The operating system creates a page table in main memory for the process.
- Store the base address of the page table in the PTBR.

Address Translation:

2

When the CPU generates a logical address:

- Extract the page number from the logical address.
- Add the page number to the PTBR to locate the page table entry in main memory.

Memory Access:

3

- Fetch the frame number from the page table entry.
- Combine the frame number with the page offset to compute the physical address.

Execution:

4

Access the memory location using the physical address.

Context Switching:

5

Update the PTBR with the base address of the new process's page table.

c. Translation Lookaside Buffer (TLB):

This method stores the page table in main memory and uses a register to hold the base address of the page table:

1 Page Table Setup

- The operating system populates the TLB with entries during process initialization or runtime.

2 Address Translation:

When the CPU generates a logical address:

- Extract the page number from the logical address.
- Search for the page number in the TLB (associative lookup).

3 TLB Hit:

If the page number is found in the TLB:

- Retrieve the corresponding frame number from the TLB.
- Combine the frame number with the page offset to compute the physical address.

4 TLB Miss:

If the page number is not in the TLB:

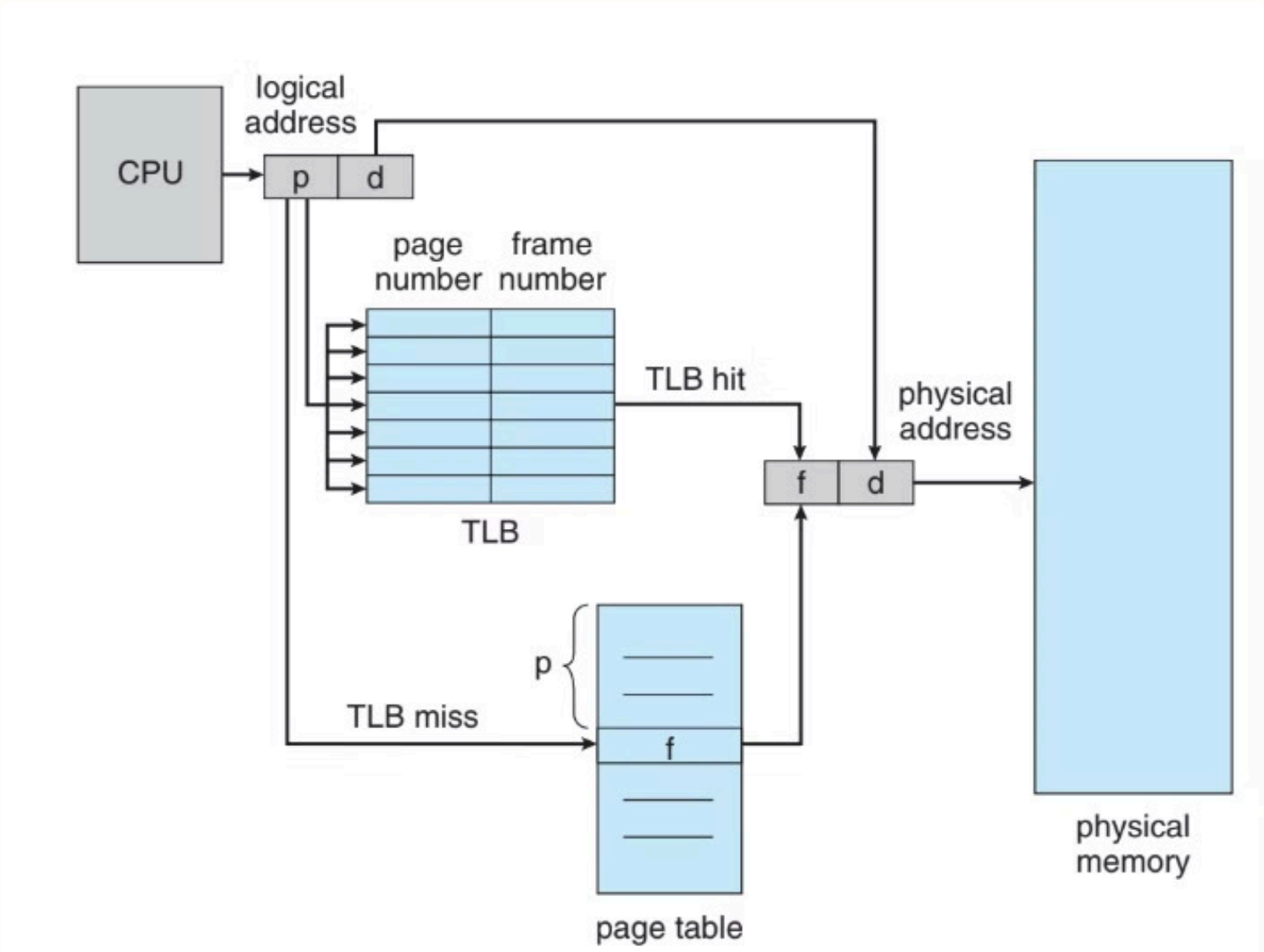
- Access the main memory page table using the PTBR to fetch the frame number.
- Update the TLB with the new page table entry (based on replacement policy).

5 Execution:

Access the memory location using the physical address.

6 Context Switching:

Flush or partially flush the TLB (depending on hardware) and reload with the new process's entries.



ref : [Operating System Concepts 9th Edition by Abraham Silberschatz.pdf - Google Drive]

	Dedicated Registers	PTBR	TLB
Speed	Fastest (direct access).	Slower (requires memory lookup).	Fast (associative cache lookup).
Memory Access	No memory access needed.	One memory access for page table entry.	Access TLB first, fallback to memory.
Context Switching	Reload registers	Update PTBR.	Flush or partially flush TLB.
Scalability	Limited to small tables.	Scales to large tables.	Scales but limited by TLB size.

6.3 Structure of the page table

explore some of the most common techniques for structuring the page table, including:

a. Hierarchical Paging.

b. Hashed page tables.

c. Inverted page tables.

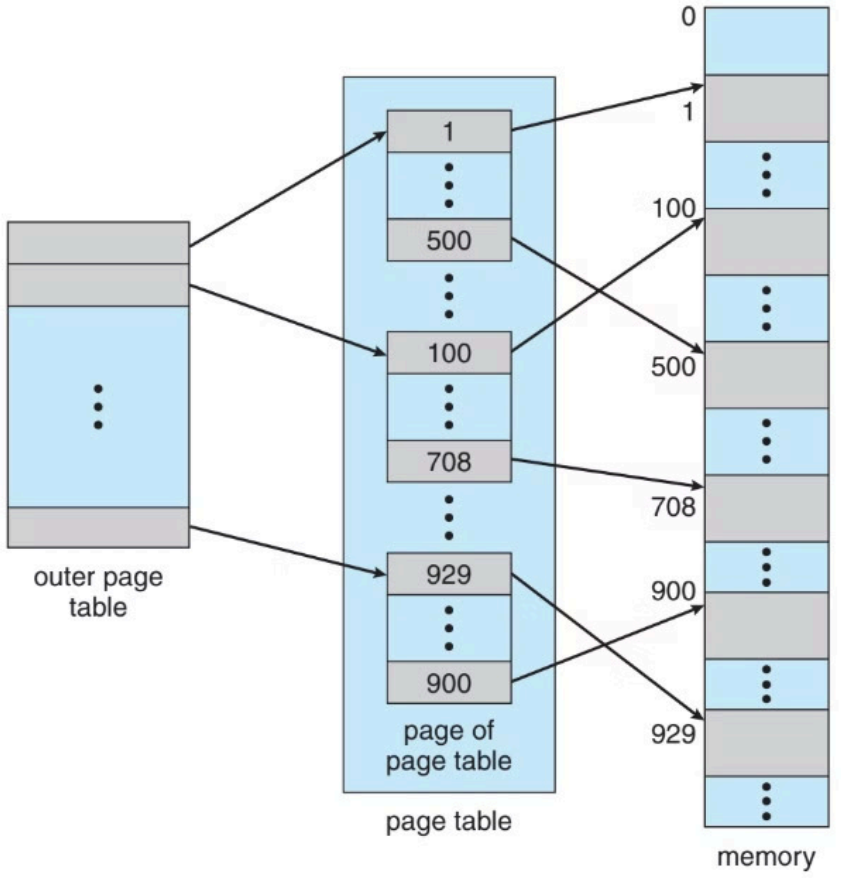
Operating System Concepts 9th Edition by Abraham Silberschatz.pdf - Google Drive

a. Hierarchical Paging

Another name for Hierarchical Paging is multilevel paging.

- There might be a case where the page table is too big to fit in a contiguous space, so we may have a hierarchy with several levels.
- In this type of Paging the logical address space is broke up into Multiple page tables.
- Hierarchical Paging is one of the simplest techniques.

the figure : A two-level page-table scheme.

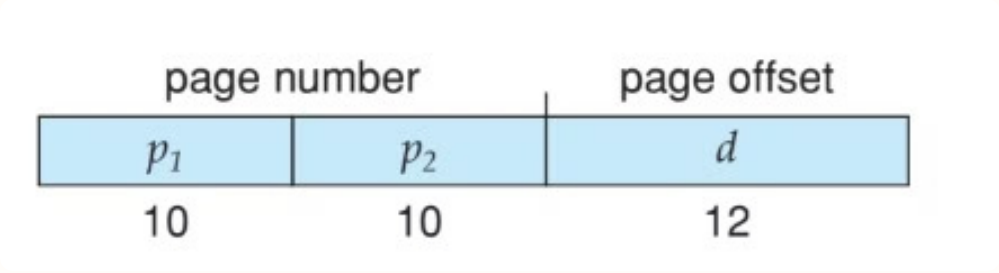


Address Translation

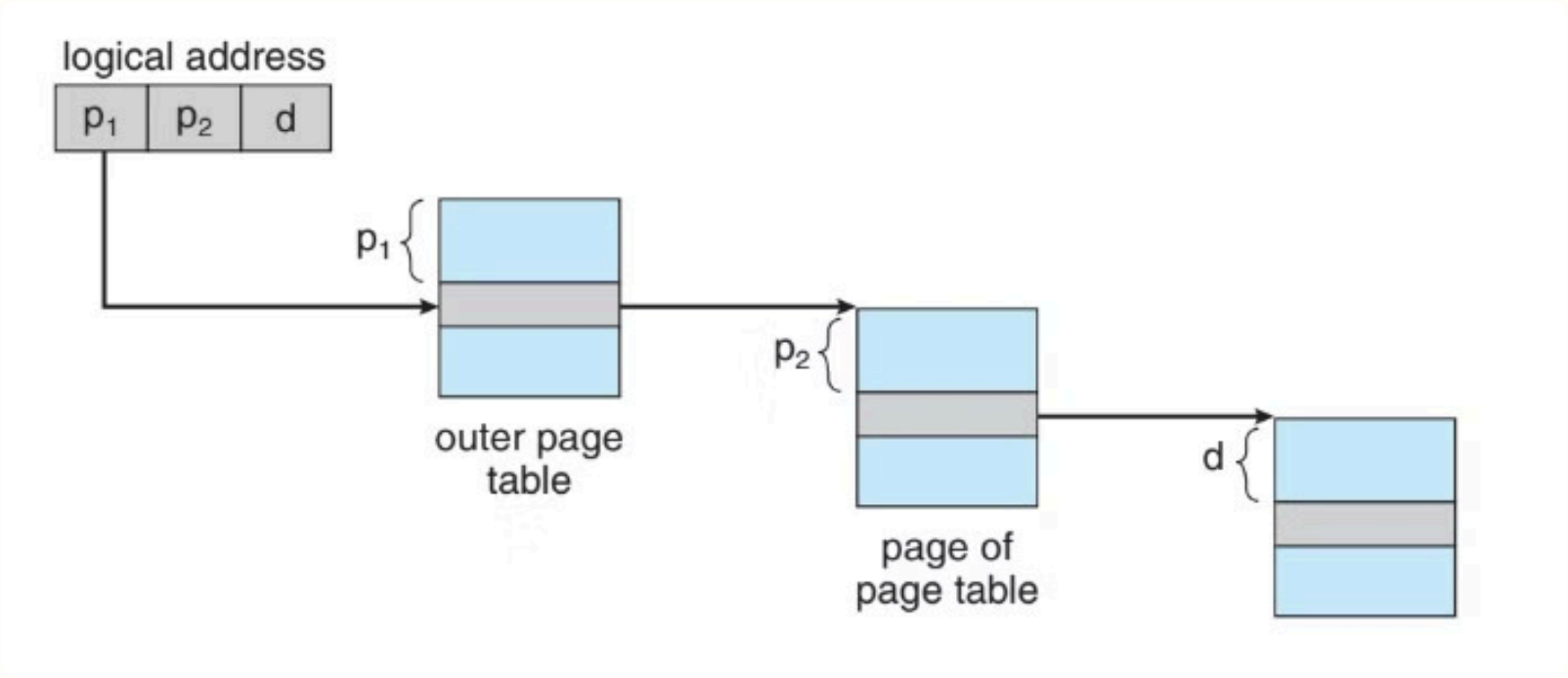
Divide the logical address :

The 32-bit logical address consists of a page number and a page offset. Since the page size is 4 KB, the page offset is 12 bits (because 2 12 = 4096 2 12 =4096 bytes = 4 KB).

- 12-bit **page offset**.
- 20-bit **page number** (split into 10-bit **outer index** and 10-bit **inner index**).



- Use the **outer page table** (indexed by the outer index) to find the address of the inner page table.
- Use the **inner page table** (indexed by the inner index) to find the physical frame number.
- Combine the **frame number** and **page offset** to get the **physical address**.

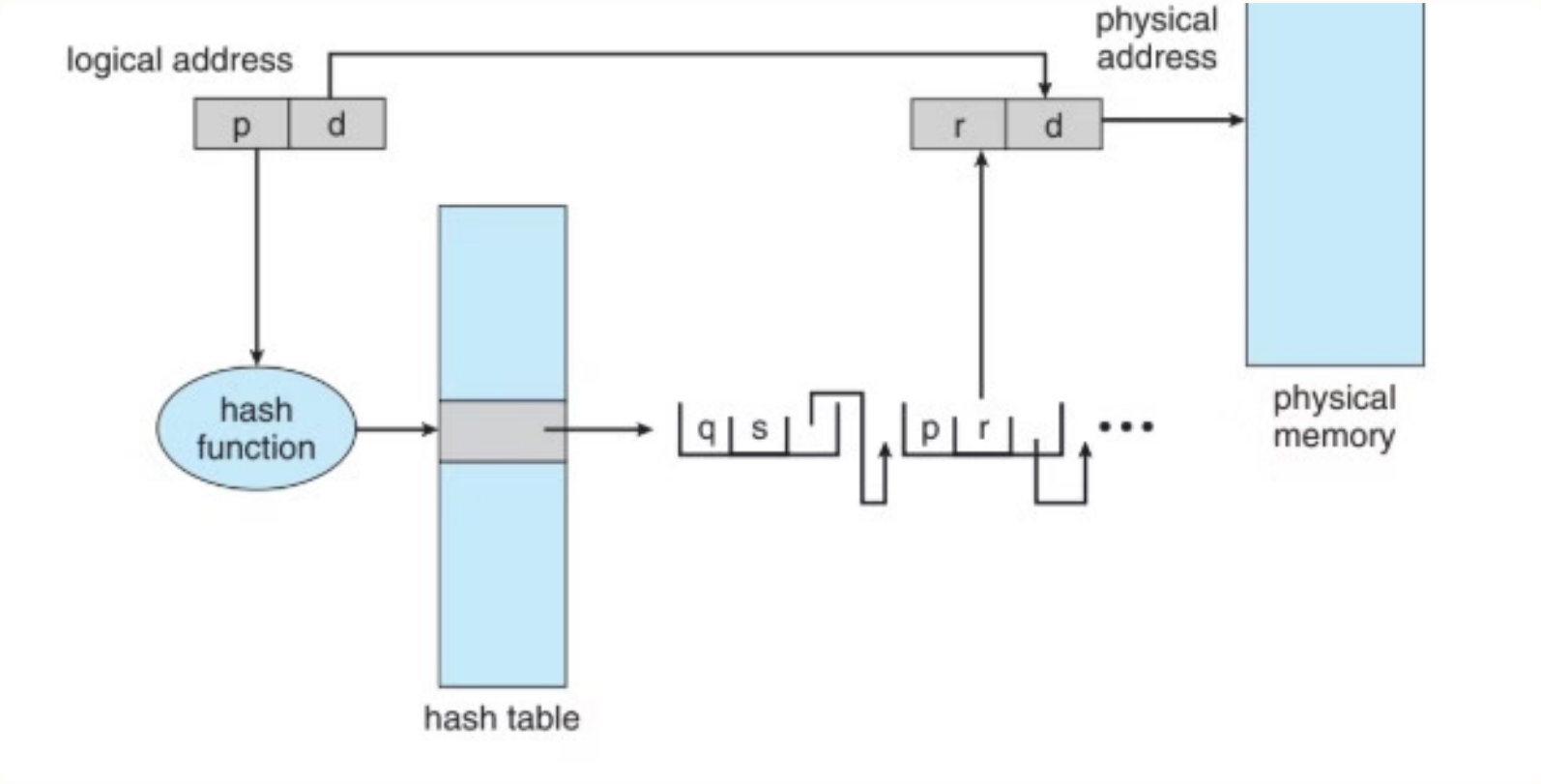


b. Hashed page table

is used to handle address spaces that are larger than 32 bits.

Maps virtual pages to physical frames using a hash table.

1. **Hashing Mechanism**:
    - Virtual Page Number (VPN) from the virtual address is hashed into the hash table.
    - The VPN excludes the page offset bits.
2. **Collision Handling**:
    - A linked list is used for entries hashing to the same location in the table.
3. **Fields in Hash Table Entry**:
    - **Field 1**: Virtual Page Number (hash value).
    - **Field 2**: Mapped page frame number (physical frame).
    - **Field 3**: Pointer to the next element in the linked list.
4. **Translation Process**:
    - VPN is hashed to find the linked list in the hash table.
    - Compare VPN with Field 1 in the first element of the linked list.
    - On match, use Field 2 (page frame) to form the physical address.
    - If no match, continue traversing the linked list.



Conclusion

In conclusion, memory organization plays a crucial role in efficient system performance. The division between user space and kernel space ensures that user programs operate in isolation from the operating system, maintaining system stability and security. Page tables are essential for managing virtual memory, translating virtual addresses to physical memory. Sorting in page tables optimizes the memory access process, and the hardware implementation of page tables involves specialized memory management units (MMUs) for fast lookups. The three structures of page tables offer trade-offs in terms of memory usage and access speed, with the choice depending on the system's design and requirements.