Memory Management

Dr Tran Duc Minh and Dr Hung Tran

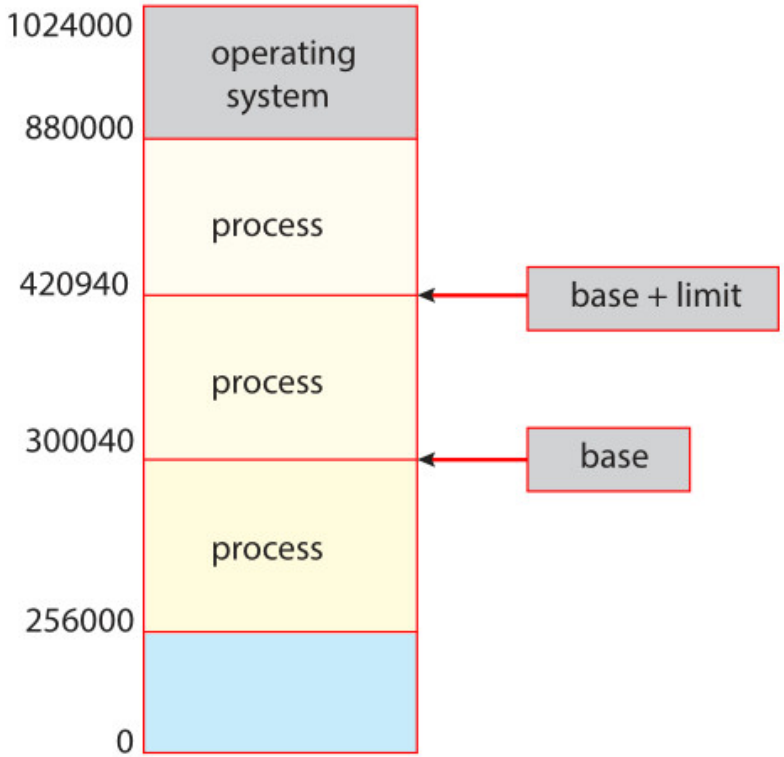
DATCOM Lab
Faculty of Data Science and Artificial Intelligence
College of Technology, National Economics University
Email: minhdt@neu.edu.vn, hung.tran@neu.edu.vn

April 21, 2025

Outline

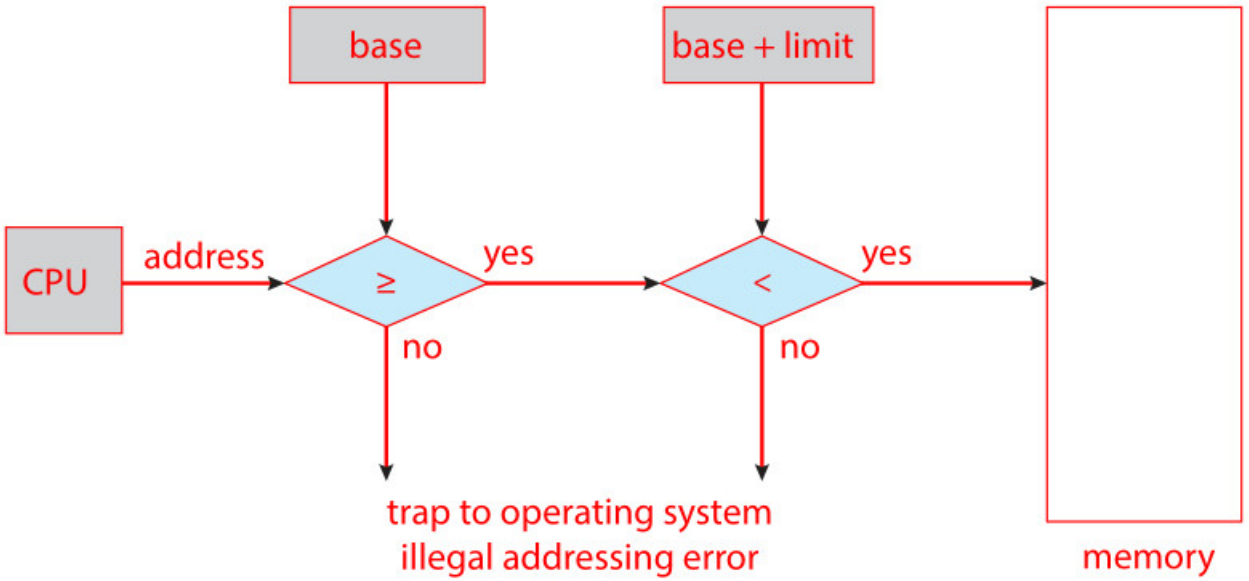
Basic Hardware

- Need to ensure that a process can access only those addresses in its address space.
- We can provide this protection by using a pair of **base** and **limit registers** define the logical address space of a process



Basic Hardware

- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user.



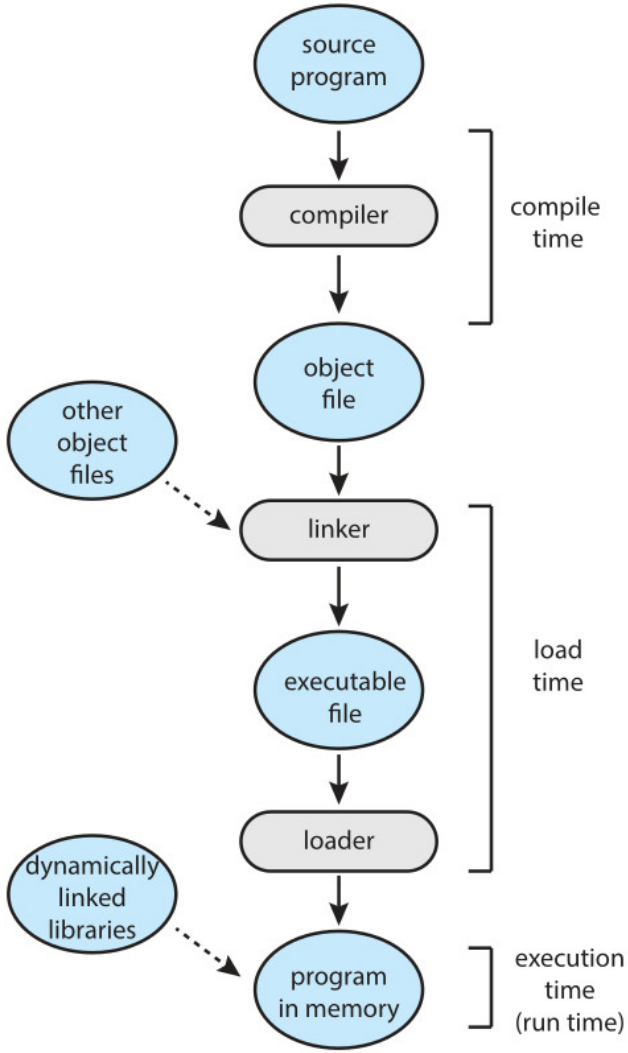
- The instructions to loading the base and limit registers are privileged

Address Binding

- As the process executes, it accesses instructions and data from memory. When the process terminates, its memory is reclaimed for use by other processes.
- Most systems allow a user process to reside in any part of the physical memory. Thus, although the address space of the computer may start at 00000, the first address of the user process need not be 00000.
- Addresses represented in different ways at different stages of a program's life:
  - Addresses in the source program are generally symbolic.
    - ★ Such as the variable **count**
  - A compiler typically **binds** these symbolic addresses to relocatable addresses.
    - ★ Such as **"14 bytes from the beginning of this module"**
  - The linker or loader in turn binds the relocatable addresses to absolute addresses.
    - ★ Such as **74014**.
  - Each binding maps one address space to another

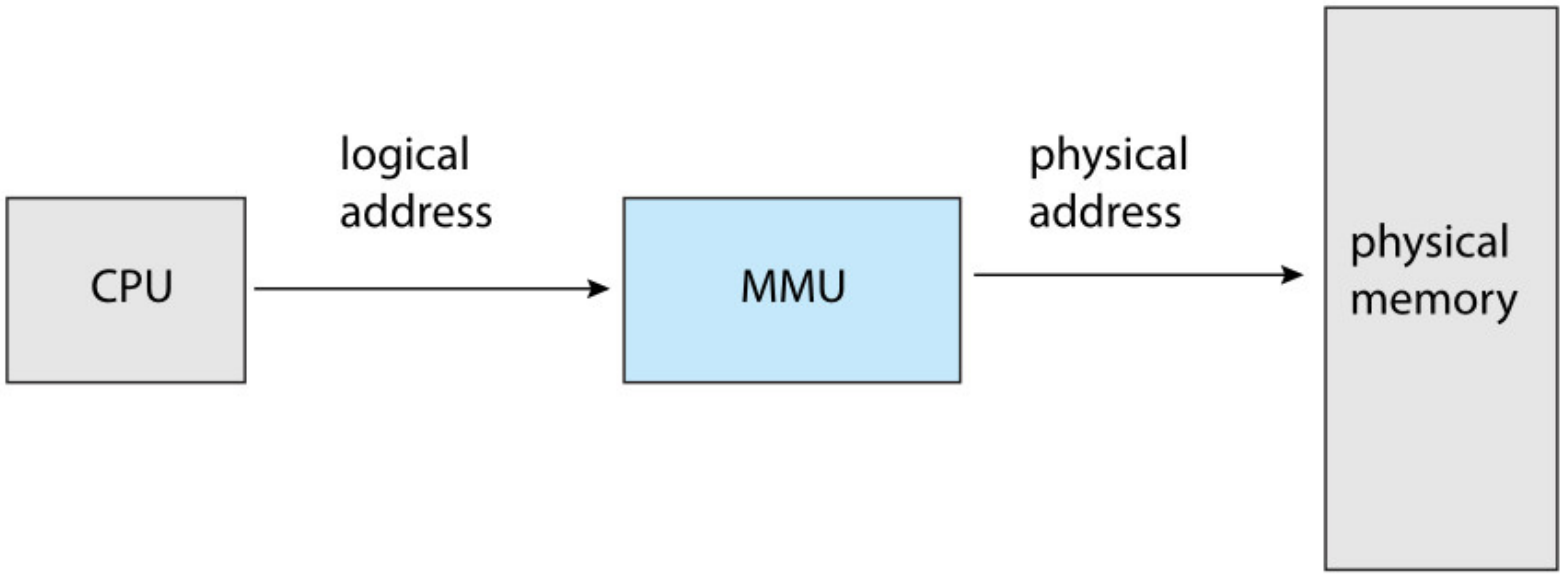
Address Binding

- Address binding of instructions and data to memory addresses can happen at three different stages.
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - ★ Need hardware support for address maps (e.g., base and limit registers)



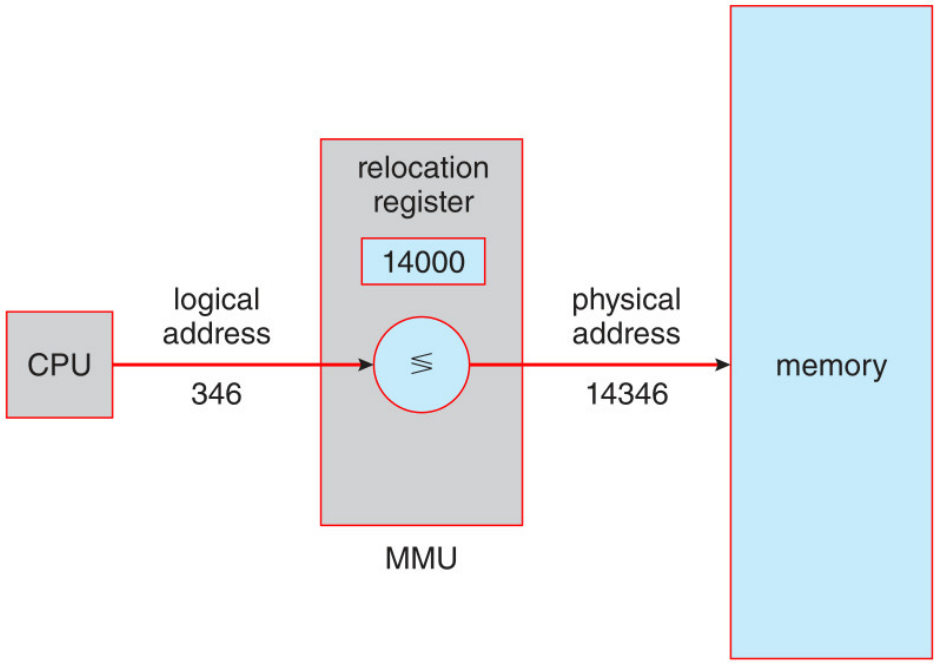
Logical Versus Physical Address Space

- MMU (Memory-Management Unit): Hardware device that at run time maps virtual to physical address.



Logical Versus Physical Address Space

- Consider simple scheme. which is a generalization of the base-register scheme.
  - The base register now called relocation register.
  - The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.



Dynamic Loading

- The entire program does need to be in memory to execute.
- Routine is not loaded until it is called.
- Better memory-space utilization; unused routine is never loaded.
- All routines kept on disk in relocatable load format.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required
  - Implemented through program design.
  - OS can help by providing libraries to implement dynamic loading.

Dynamic Linking

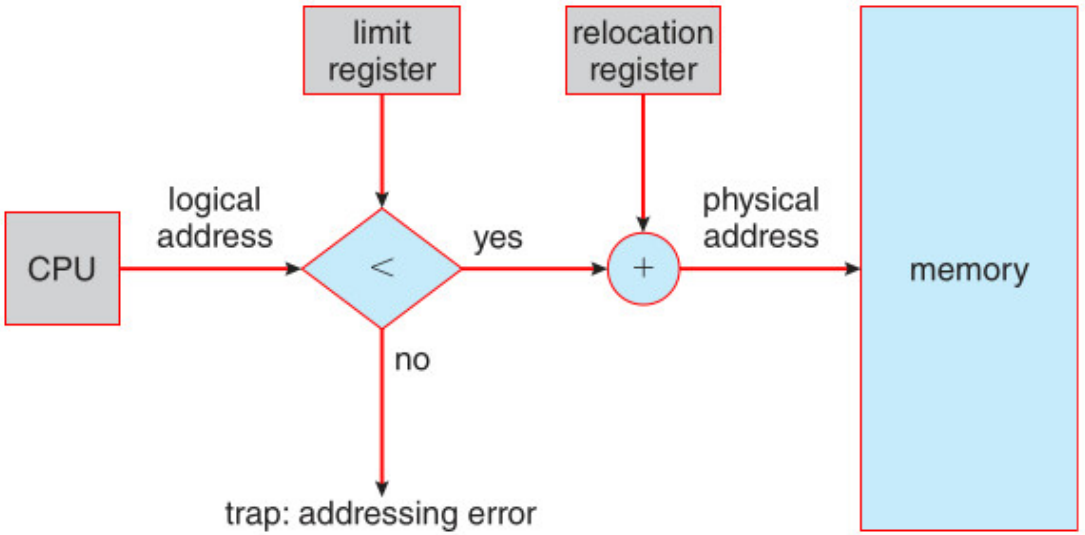
- **Static linking**: system libraries and program code combined by the loader into the binary program image.
- Dynamic linking: Linking postponed until execution time.
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine.
- Stub replaces itself with the address of the routine, and executes the routine.
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space.
- Dynamic linking is particularly useful for libraries.
- System also known as **shared libraries**.
- Consider applicability to patching system libraries
  - Versioning may be needed.

Contiguous Memory Allocation

- Main memory must support both OS and user processes.
- Limited resource, must allocate efficiently.
- Contiguous allocation is often used in simple operating systems or early memory management stages.
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.
  - Each process contained in single contiguous section of memory.
- Relocation registers used to protect user processes from each other and from changing operating-system code and data
  - Base register contains value of smallest physical address.
  - Limit register contains range of logical addresses - each logical address must be less than the limit register.
  - MMU maps logical address dynamically.

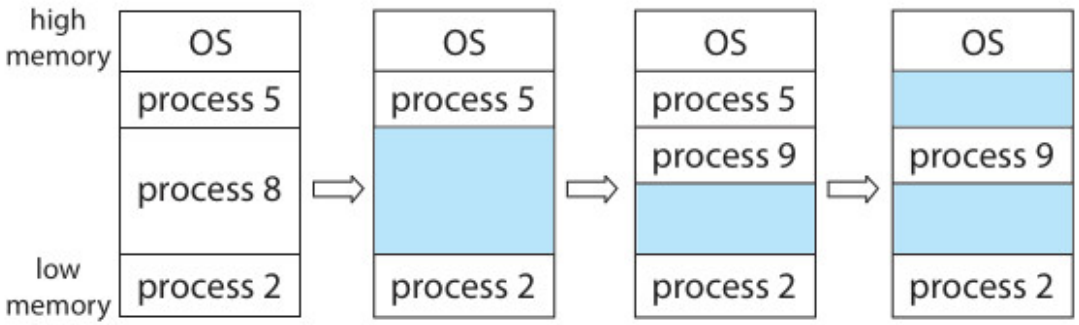
Memory Protection

- The relocation register contains the value of the smallest physical address; the limit register contains the range of logical addresses.
- Each logical address must fall within the range specified by the limit register.
- The MMU maps the logical address dynamically by adding the value in the relocation register.



Memory Allocation

- Degree of multiprogramming limited by number of partitions.
- **Variable-partition** sizes for efficiency (sized to a given process' needs).
- **Hole**: Block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Process exiting frees its partition, adjacent free partitions combined.
- Operating system maintains information about:
  1. Allocated partitions.
  2. Free partitions (hole)



Memory Allocation

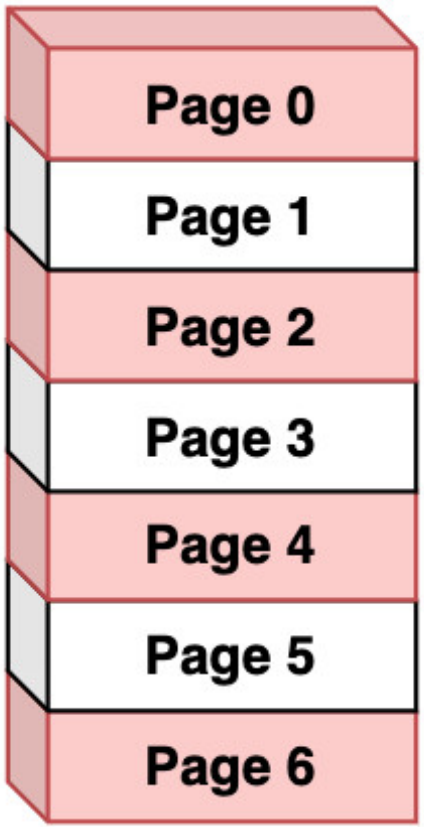
- How to satisfy a request of size n from a list of free holes ?
  - **First-fit**: Allocate the first hole that is big enough.
  - **Best-fit**: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size.
    - ★ Produces the smallest leftover hole.
  - **Worst-fit**: Allocate the largest hole; must also search entire list.
    - ★ Produces the largest leftover hole.
  - First-fit and best-fit better than worst-fit in terms of speed and storage utilization.

Fragmentation

- **External Fragmentation**: Total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation**: Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- One solution to the problem of external fragmentation is **compaction**:
  - The goal is to shuffle the memory contents so as to place all free memory together in one large block.
  - Compaction is not always possible. If relocation is static and is done at assembly or load time, compaction cannot be done.
  - When compaction is possible, we must determine its cost. The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive.

Introduction

- **Paging** is a memory management scheme that permits a process's physical address space to be noncontiguous.
- Paging avoids external fragmentation and the associated need for compaction, two problems that plague contiguous memory allocation.
- Paging is implemented through cooperation between the operating system and the computer hardware.



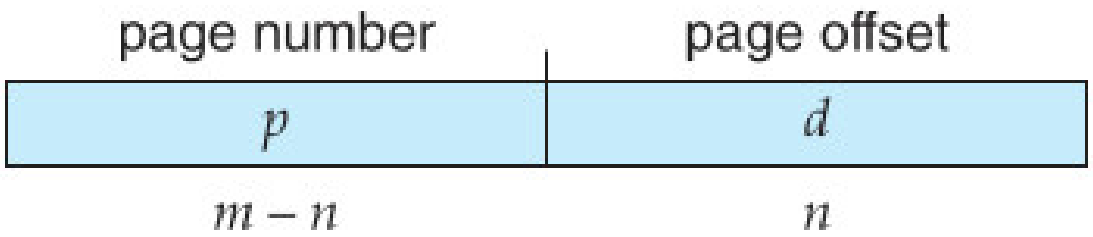
Paging

Basic Method

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available.
  - Avoids external fragmentation.
  - Avoids problem of varying sized memory chunks.
- Divide physical memory into fixed-sized blocks called **frames**.
  - Size is power of 2, between 512 bytes and 16 Mbytes.
- Divide logical memory into blocks of same size called **pages**.
- Keep track of all free frames.
- To run a program of size **N** pages, need to find **N** free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Still have Internal fragmentation.

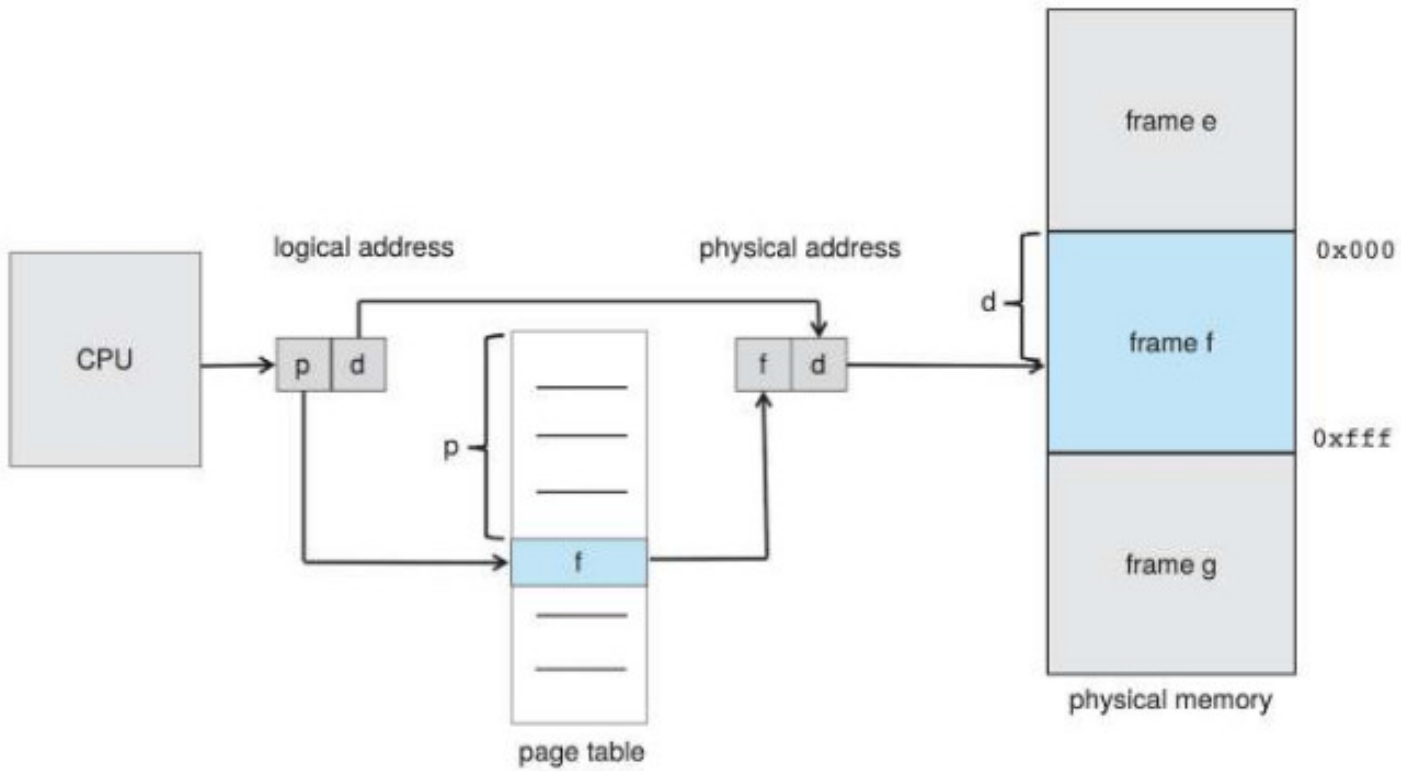
Basic Method

- Address generated by CPU is divided into:
  - **Page number (p)** used as an index into a **page table** which contains base address of each page in physical memory.
  - **Page offset (d)** combined with base address to define the physical memory address that is sent to the memory unit.
  - If the size of the logical address space is 2^m , and a page size is 2^n bytes, then the high-order $m - n$ bits of a logical address designate the page number, and the n low-order bits designate the page offset.



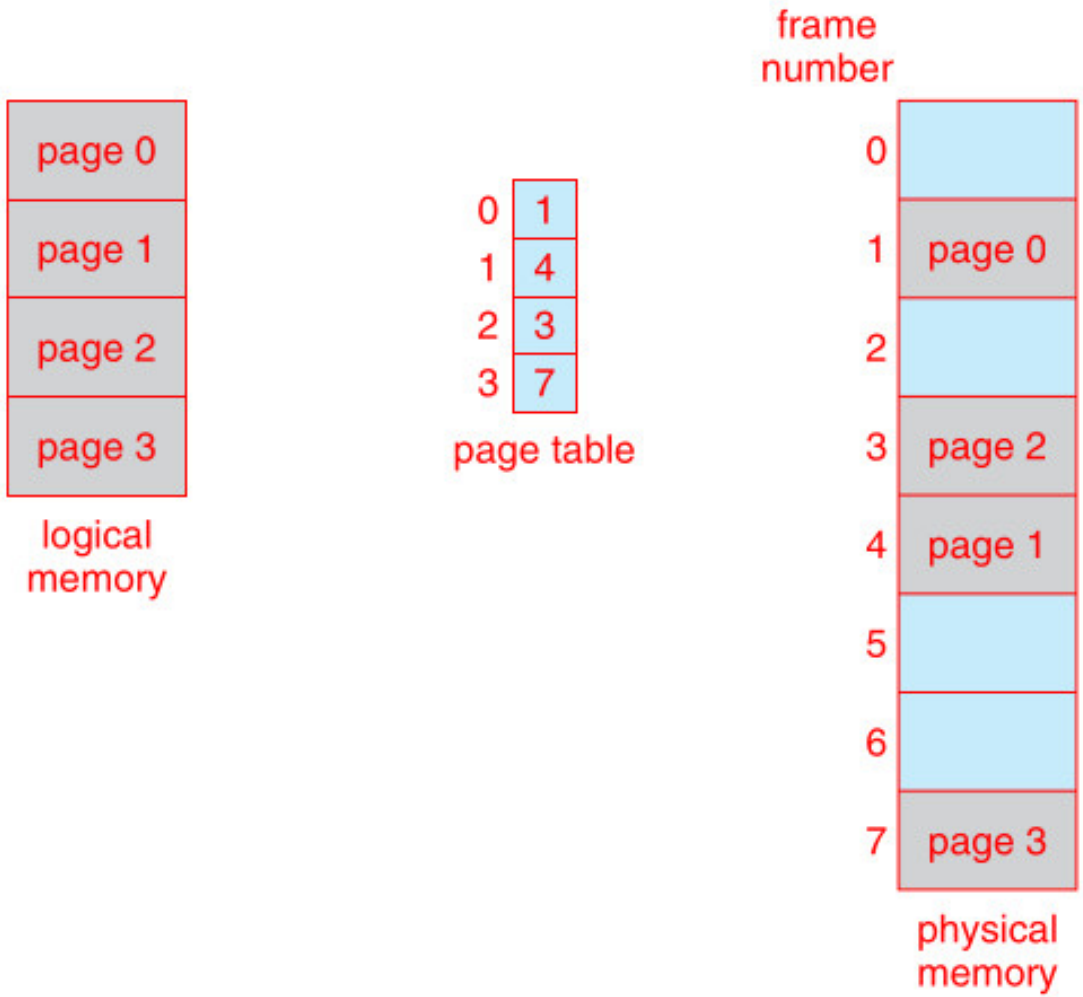
Basic Method

- The following outlines the steps taken by the MMU to translate a logical address generated by the CPU to a physical address:
  1. Extract the page number p and use it as an index into the page table.
  2. Extract the corresponding frame number f from the page table.
  3. Replace the page number p in the logical address with the frame number f.



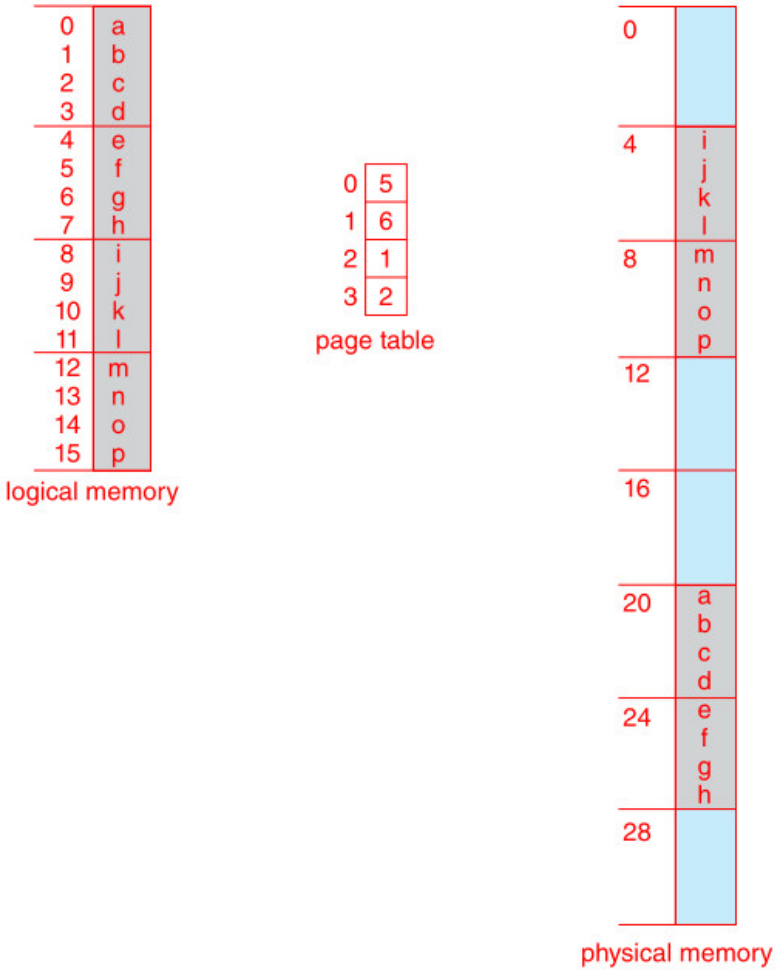
Basic Method

- Paging model of logical and physical memory



Basic Method

- For example: $n = 2$ and $m = 4$. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages)



Basic Method

- For example:
  - Logical address 0 (page 0, offset 0) maps to physical address $20 = (5x4) + 0$.
  - Logical address 3 (page 0, offset 3) maps to physical address $23 = (5x4) + 3$.
  - Logical address 4 (page 1, offset 0) maps to physical address $24 = (6x4) + 0$.
  - Logical address 10 (page 2, offset 2) maps to physical address $6 = (1x4) + 2$.
  - Logical address 13 (page 3, offset 1) maps to physical address $9 = (2x4) + 1$.

Basic Method

- (a): Before allocation
- (b): After allocation

Hardware support

- Page table is kept in main memory:
  - **Page-table base register (PTBR)** points to the page table.
  - **Page-table length register (PTLR)** indicates size of the page table.

  In this scheme every data/instruction access requires **two memory accesses**. One for the page table and one for the data/instruction.
- The two-memory access problem can be solved by the use of a special **fast-lookup hardware cache** called **translation look-aside buffers (TLBs)** (also called **associative memory**).

Hardware support

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry - uniquely identifies each process to provide address-space protection for that process.
  - For example - Suppose there are two processes: Process A (ASID = 1) and Process B (ASID = 2). When the CPU handles Process A, the TLB only uses entries with ASID = 1, preventing Process A from accessing Process B's data.
- If the TLB does not use ASIDs, it must be flushed (cleared) during every context switch between processes to prevent the new process from accessing the old process's data.

Hardware support

- TLBs are usually small, containing 64 to 1,024 entries, due to hardware constraints and to ensure fast access.
- When a TLB miss occurs (the required address is not in the TLB), the system looks up the mapping in the page table and loads it into the TLB for faster access in the future.
- Some TLB entries can be "wired down" (fixed), meaning they are not replaced, ensuring permanent fast access. This is often used for critical addresses, like those of the operating system.

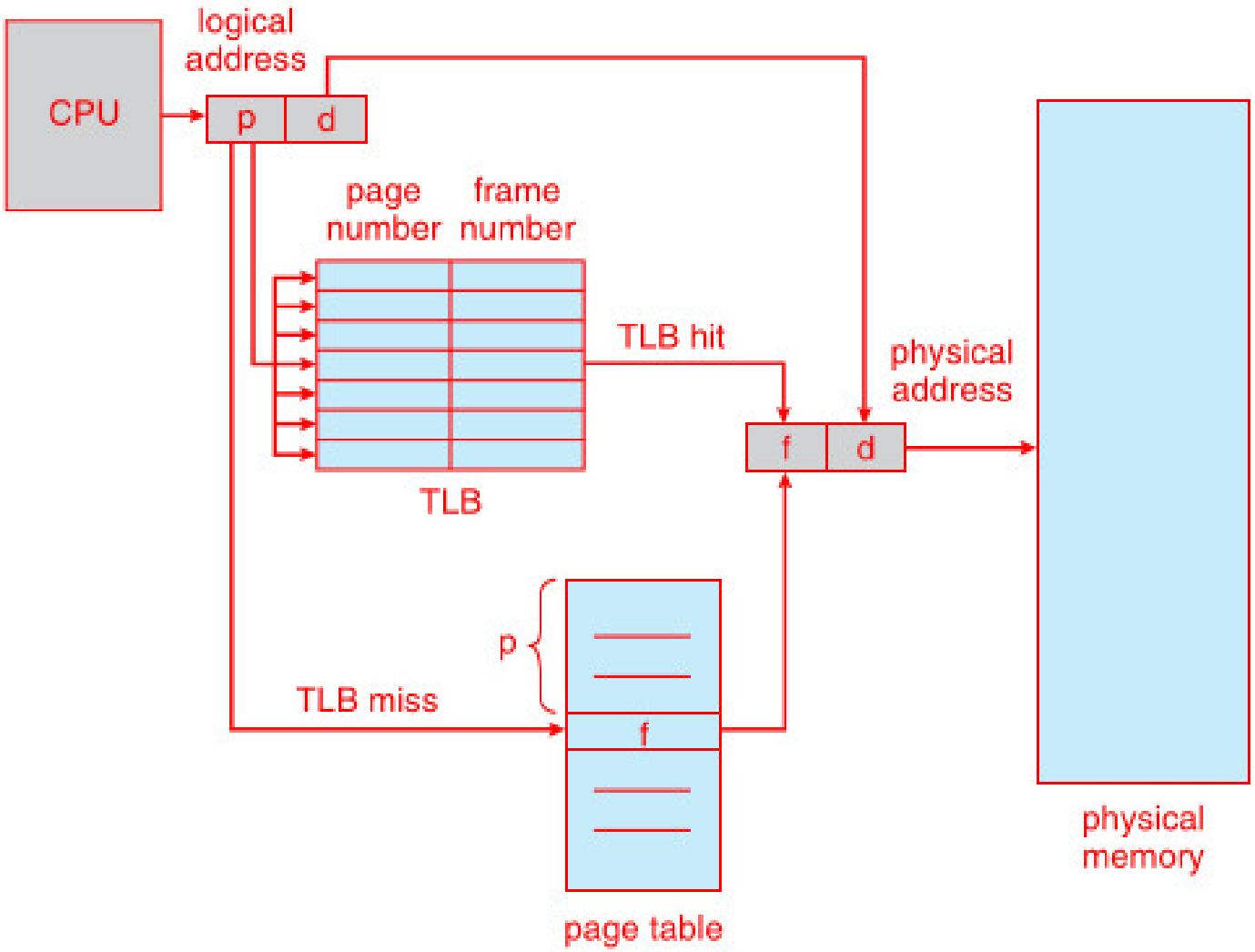
Hardware support

- Associative memory - parallel search

Page #	Frame #

- Address translation (p, d)
  - If p is in associative register, get frame # out.
  - Otherwise get frame # from page table in memory.

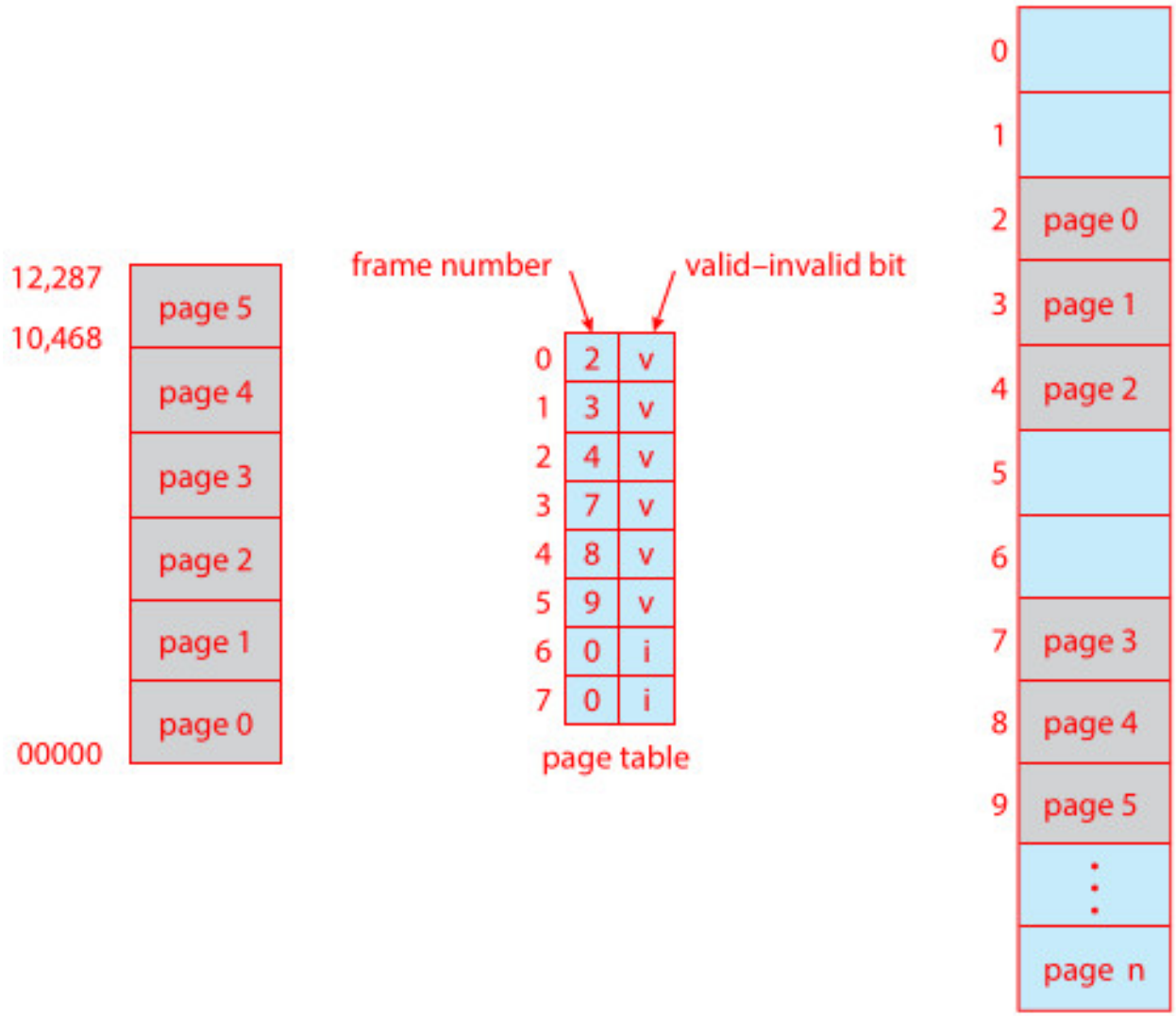
Hardware support



Memory Protection

- **Protection Bits**: Stored in the page table, define access rights for each frame (e.g., read-write, read-only).
- **Access Violation**: Unauthorized access (e.g., writing to a read-only page) triggers a hardware trap or memory-protection violation.
- **Finer Control**: Use separate bits for read, write, and execute permissions; illegal attempts are trapped by the OS.
- **Valid-Invalid Bit**:
  - Marks a page as valid (in process's logical address space) or invalid (not in the process's space).
  - Illegal addresses are trapped by the valid-invalid bit.

Memory Protection



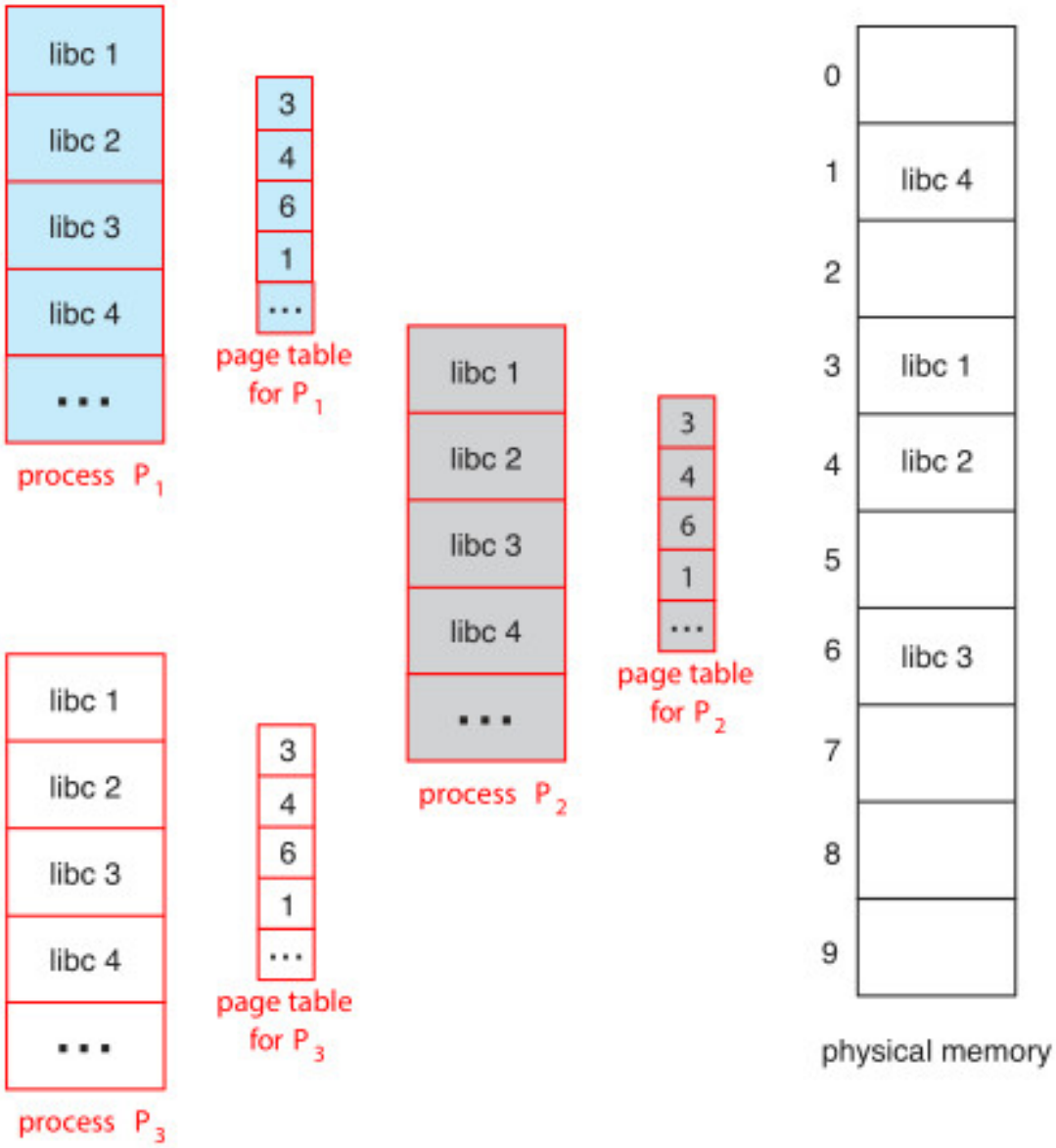
Memory Protection

- **For example:** A system has a 14-bit address space, a page size of 2 KB and a program use only addresses 0 to 10468.
  - Addresses 0 to 16383; pages 0 - 5 are valid; otherwise invalid.
  - Issue: Address 10468 is valid, but 12288 to 16383 are invalid due to internal fragmentation.
    - ★ References to page 5 are classified as valid, so accesses to addresses up to 12287 are valid.
- **Sparse Address Usage**: Most processes use a small fraction of their address space, wasting page table entries.
- **Solution**: Page-Table Length Register (PTLR):
  - Tracks the size of the page table.
  - Ensures only valid addresses are accessed; failures cause an OS trap.

Shared Pages

- **Advantage of Paging**: Enables sharing common code in multi-process environments.
- **Example**: Standard C library (`libc`) on Linux, typically 2 MB, required by most processes.
- **Without Sharing**: 40 processes x 2 MB = 80 MB of memory for individual `libc` copies.
- **Reentrant Code**: Non-self-modifying, read-only code that can be shared.
- **Sharing Mechanism**:
  - Multiple processes share `libc` pages.
  - Each process has its own registers and data storage for process-specific data.

Shared Pages



Shared Pages

- **Memory Savings**: One copy of `libc` (2 MB) in physical memory; page tables of 40 processes map to it, reducing usage to 2 MB from 80 MB.
- **Applicability**: Other reentrant programs (compilers, database systems) can also be shared.
- **Requirements**:
  - Code must be reentrant; OS enforces read-only property.
- **Related Concepts**:
  - Shared libraries use shared pages.
  - Shared memory for interprocess communication and thread address space sharing use similar techniques.

Questions