Chapter 8
Memory Management

Contents

- Background
- Swapping
- Contiguous Memory Allocation
- Paging
- Structure of the Page Table
- Segmentation

Background

Basic Hardware

Address Binding

Logical VS Physical Address Space

Dynamic Loading

Dynamic Linking and Shared Libraries

Basic Hardware

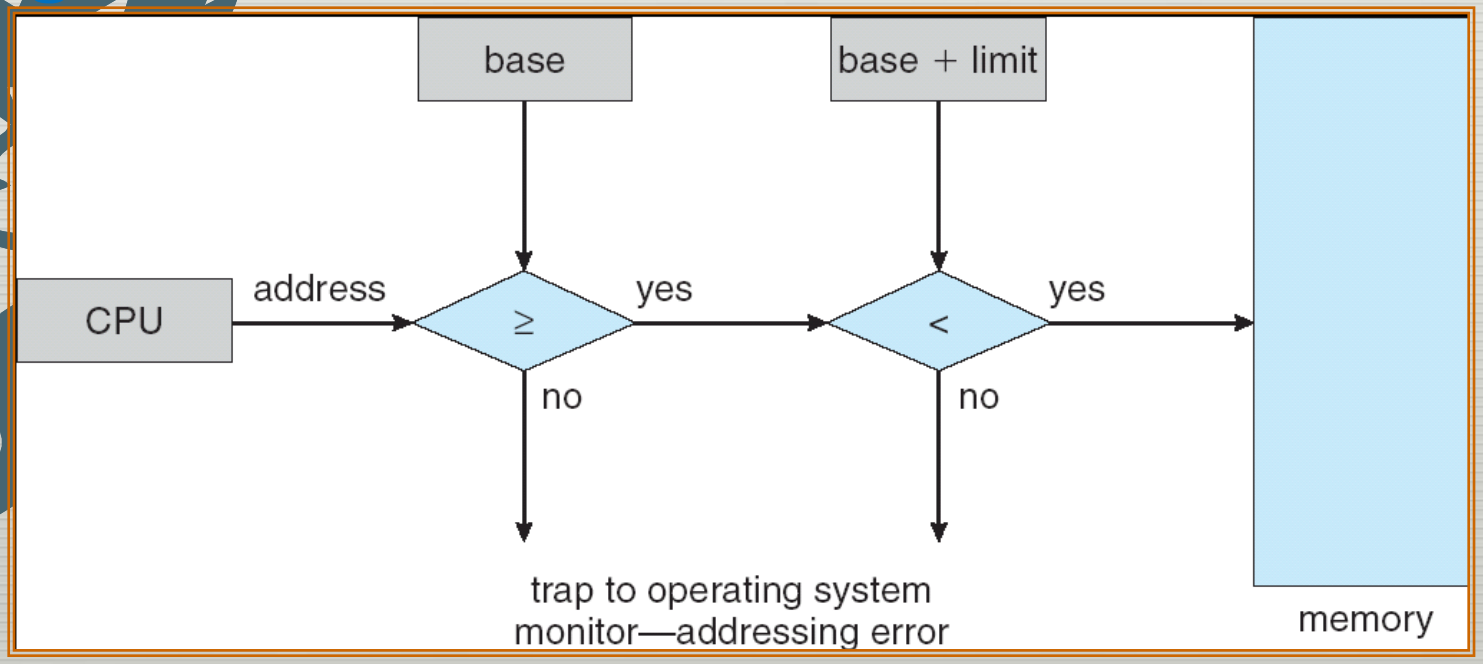
- Main memory and registers built into the processor itself are **the only storage** that the **CPU can access directly**.
- Any instructions and any data must be in one of these **direct-access** storage devices.

Basic Hardware

- Most CPUs can **decode** instructions and **perform** simple operations on register contents at the rate of **one or more operations per clock tick**.
- The same **can't** be said of main memory, which is accessed via a transaction on the memory bus.
- In such case, the processor needs to stall.
- The remedy is to add **fast memory between** the CPU and main memory. - Cache

Basic Hardware

- We also must ensure correct operation to **protect** the operating system from access by the user process. – Base register / Limit register



Address Binding

- **To be executed**, a program resides on a **disk** must be **brought into memory** and **placed within a process**.
- Depending on the **memory management** in use, **the process** may be **moved** between **disk and memory** during in execution.
- The processes on the disk that are waiting to be brought into memory for execution from the input queue.

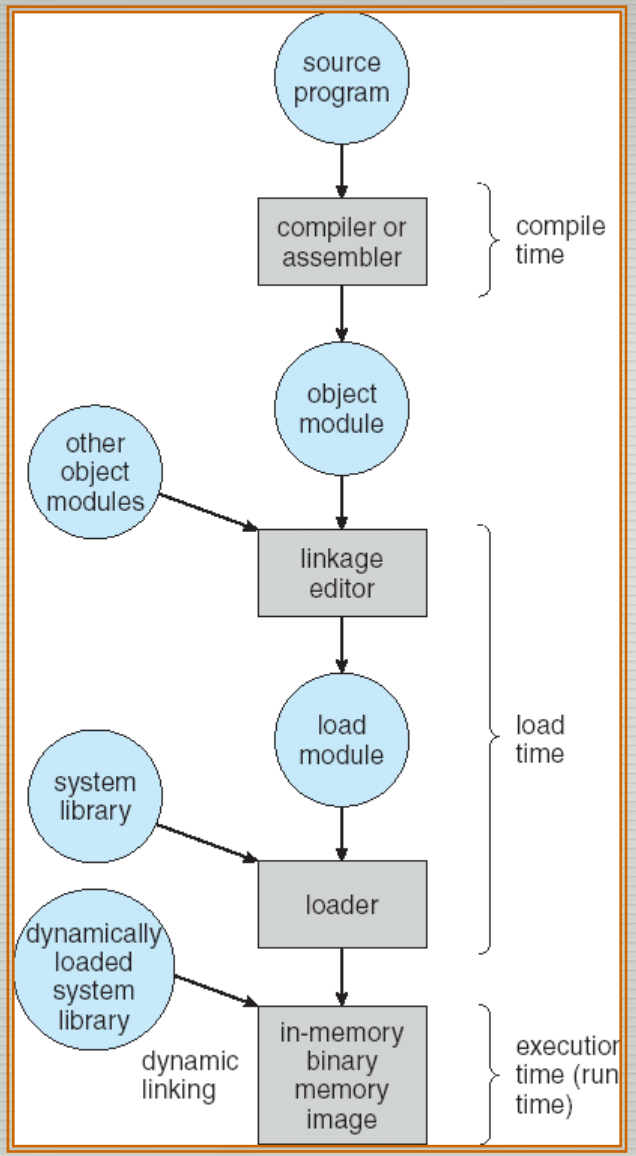
Address binding

- Most systems allow a user process to reside in any part of the physical memory.
- This approach **affects the address** that the user process can use. - A user program will go through several steps.
- Addresses may be **represented** in different ways during these steps. - Symbolic address.
- A compiler will typically bind these symbolic addresses to relocatable addresses.

Address Binding

- **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*).

Address Binding



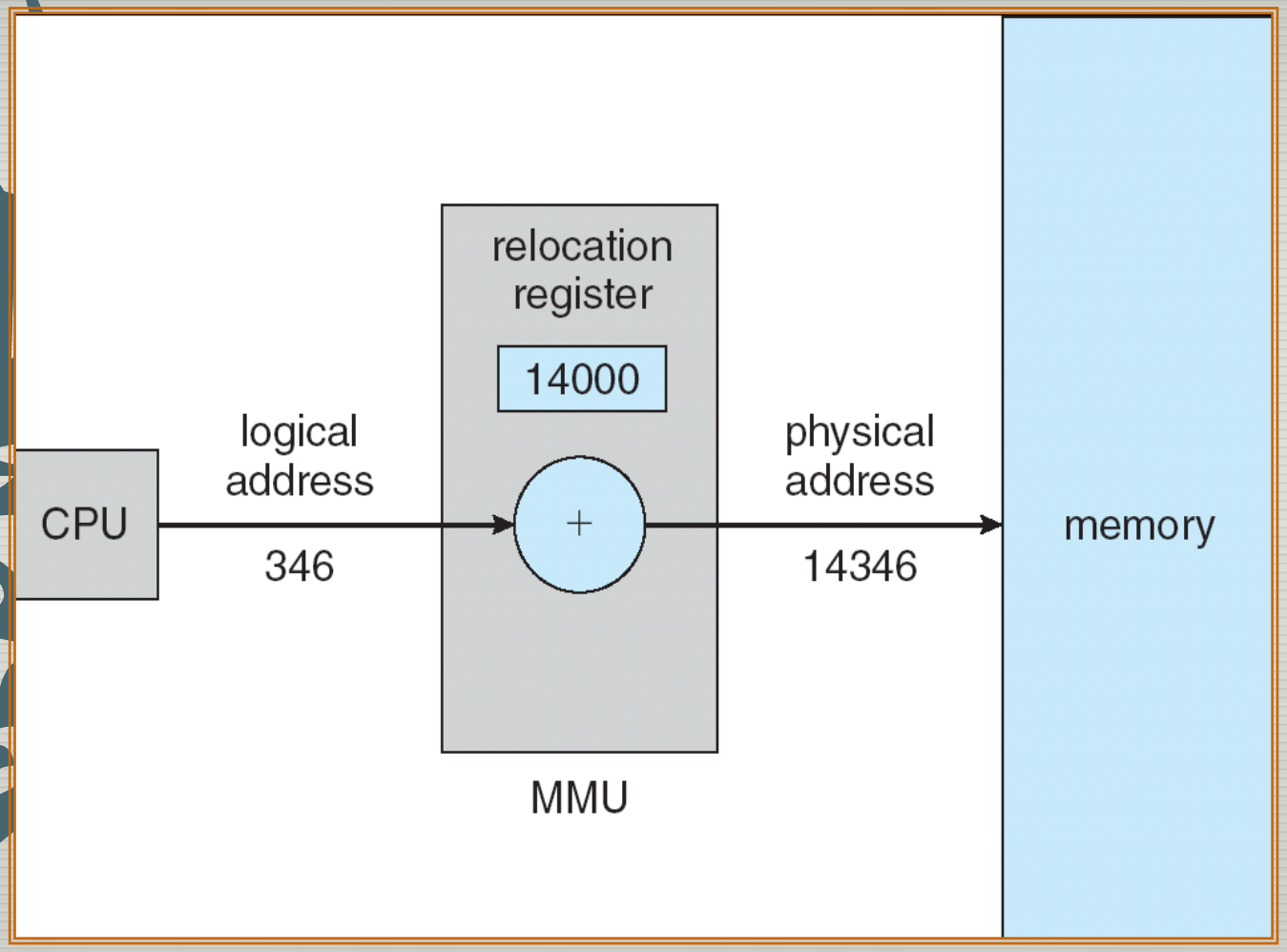
Logical VS Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to **proper memory management**
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the **memory unit**
- Logical and physical addresses are the **same in compile-time and load-time address-binding** schemes; logical (virtual) and physical addresses **differ in execution-time address-binding** scheme

Logical VS Physical Address Space

- The run-time mapping from virtual to physical address is done by a hardware device called the memory-management unit(MMU).
- **The user program deals with logical addresses**; it never sees the real physical addresses

Logical VS Physical Address Space



Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; **unused routine is never loaded**
- 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. – User Code

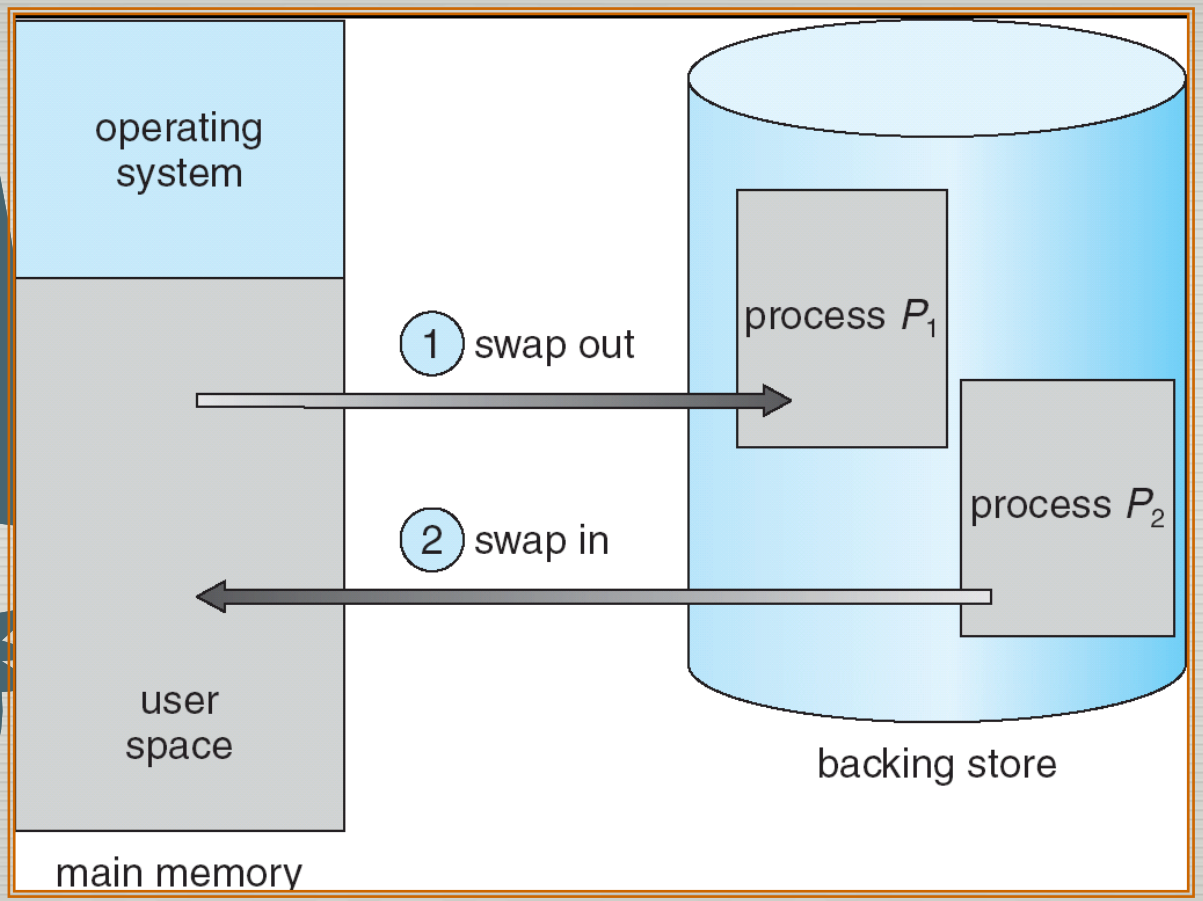
Dynamic Linking

- 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.**

Swapping

Swapping

Swapping


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Contiguous Memory Allocation

Contiguous Memory Allocation

Memory Mapping and Protecting

Memory Allocation

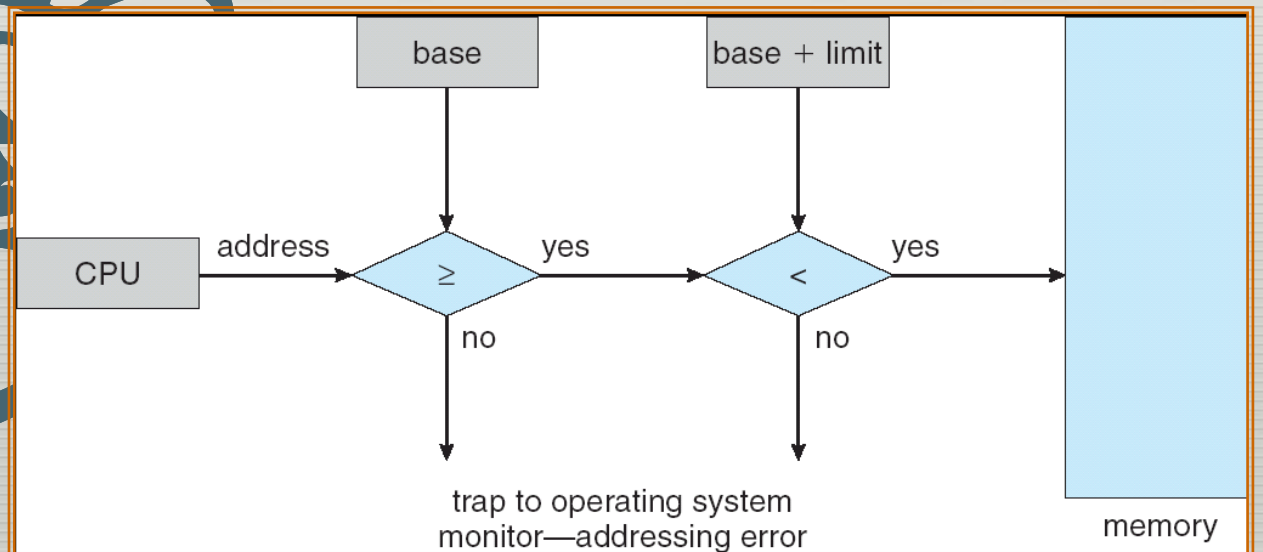
Fragmentation

Contiguous Memory Allocation

- Main memory usually into **two partitions**:
  - Resident **operating system**, usually held in low memory with interrupt vector
  - **User processes** then held in high memory
- We usually want **several user processes** to reside in memory **at the same time.**
- In contiguous memory allocation, each processes is contained a **singled contiguous section of memory**.

Memory Mapping and Protection

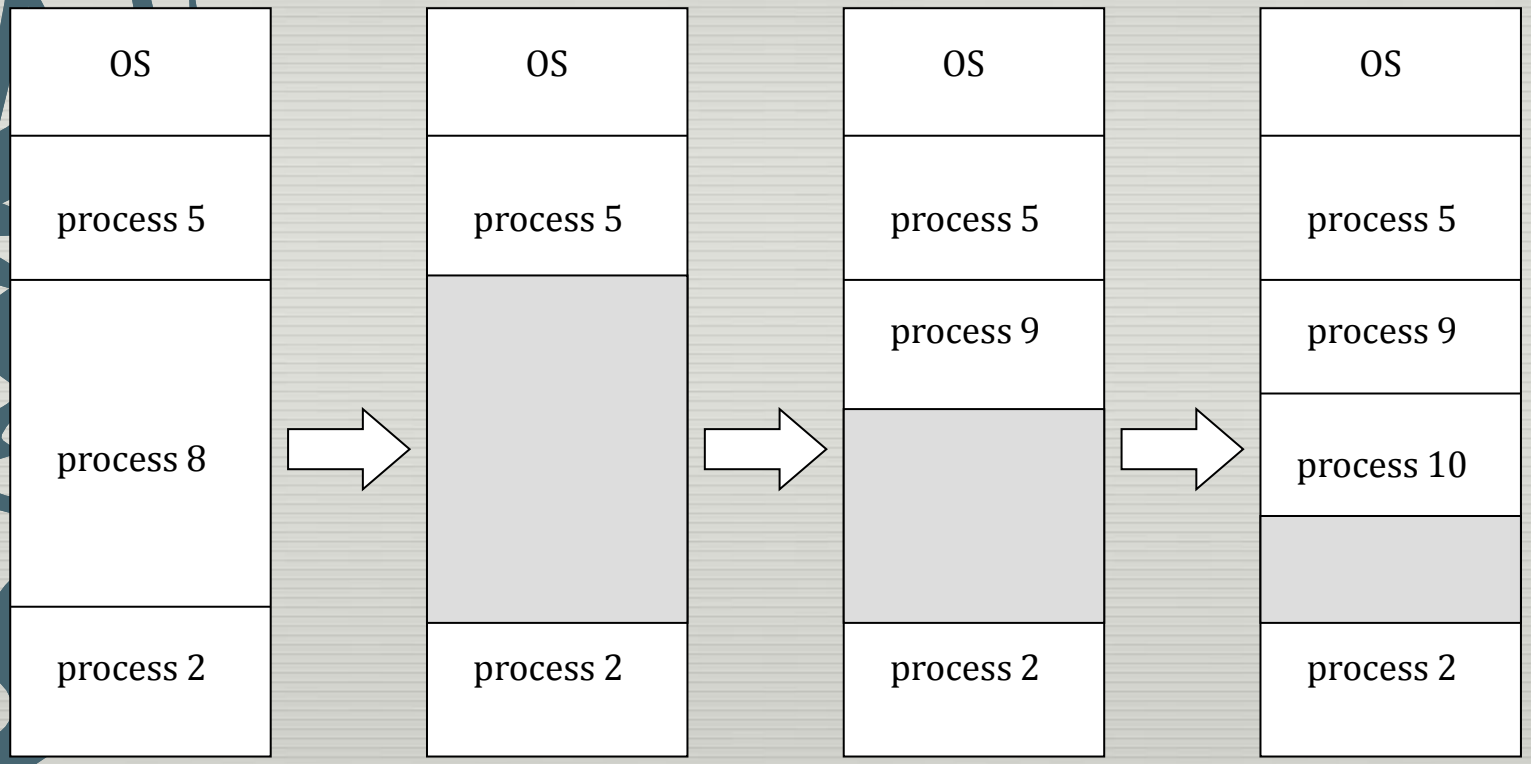
- **PCB** contains **base and limit registers.**
- The relocation-register scheme provides an effective way to allow operating system size to change dynamically - Transient operating system code is needed.



Memory Allocation

- Multiple-partition allocation
  - 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
  - **Operating system** maintains information about: **a) allocated partitions b) free partitions (hole)**

Memory Allocation



Memory Allocation

- **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.**

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
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible only if **relocation is dynamic**, and is **done at execution time**

Paging

Paging

Basic Method

Hardware Support

Protection

Shared Pages

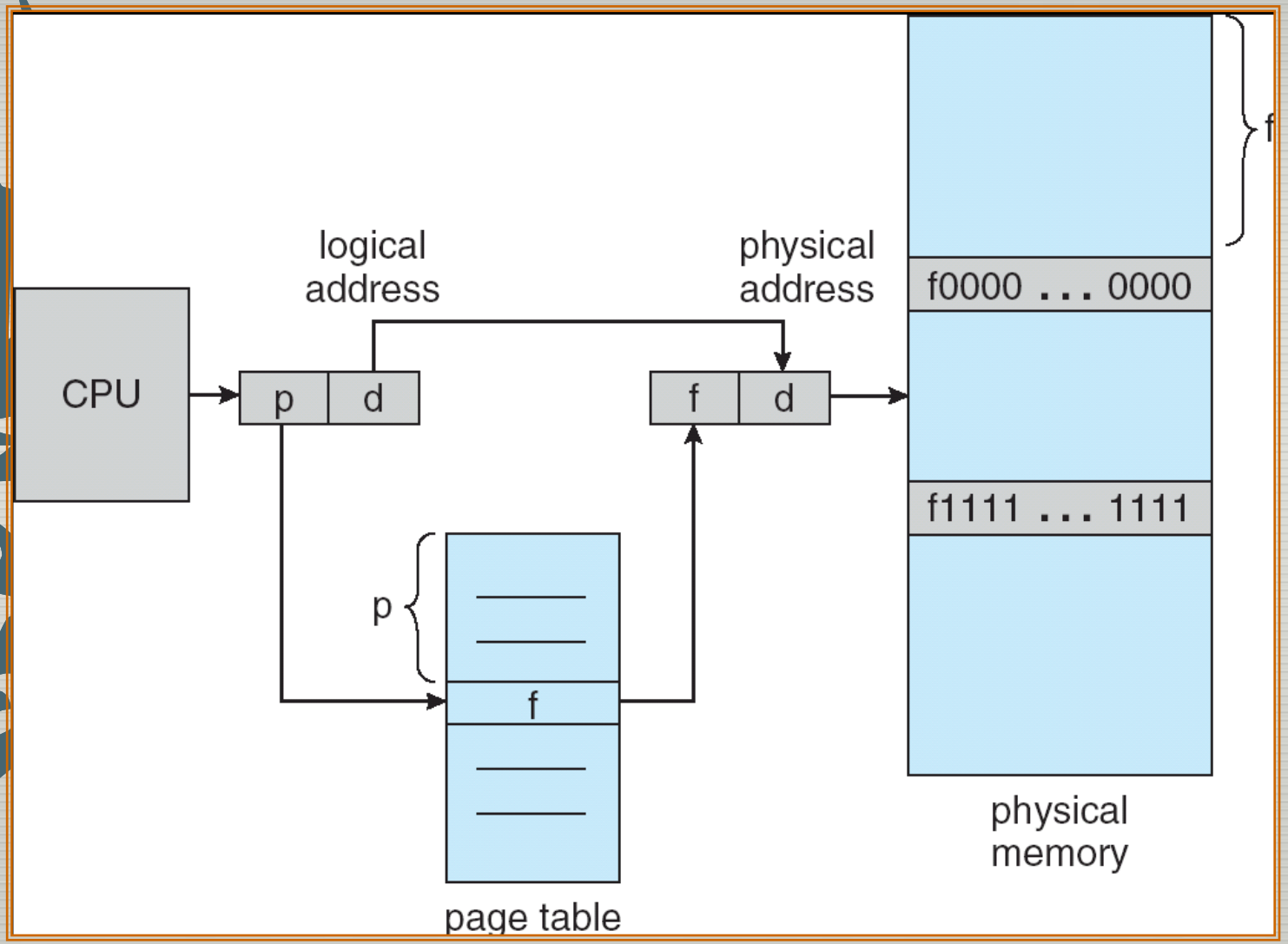
Paging

- Paging is a memory-management scheme that permits the physical address space of a process to be **noncontiguous**.

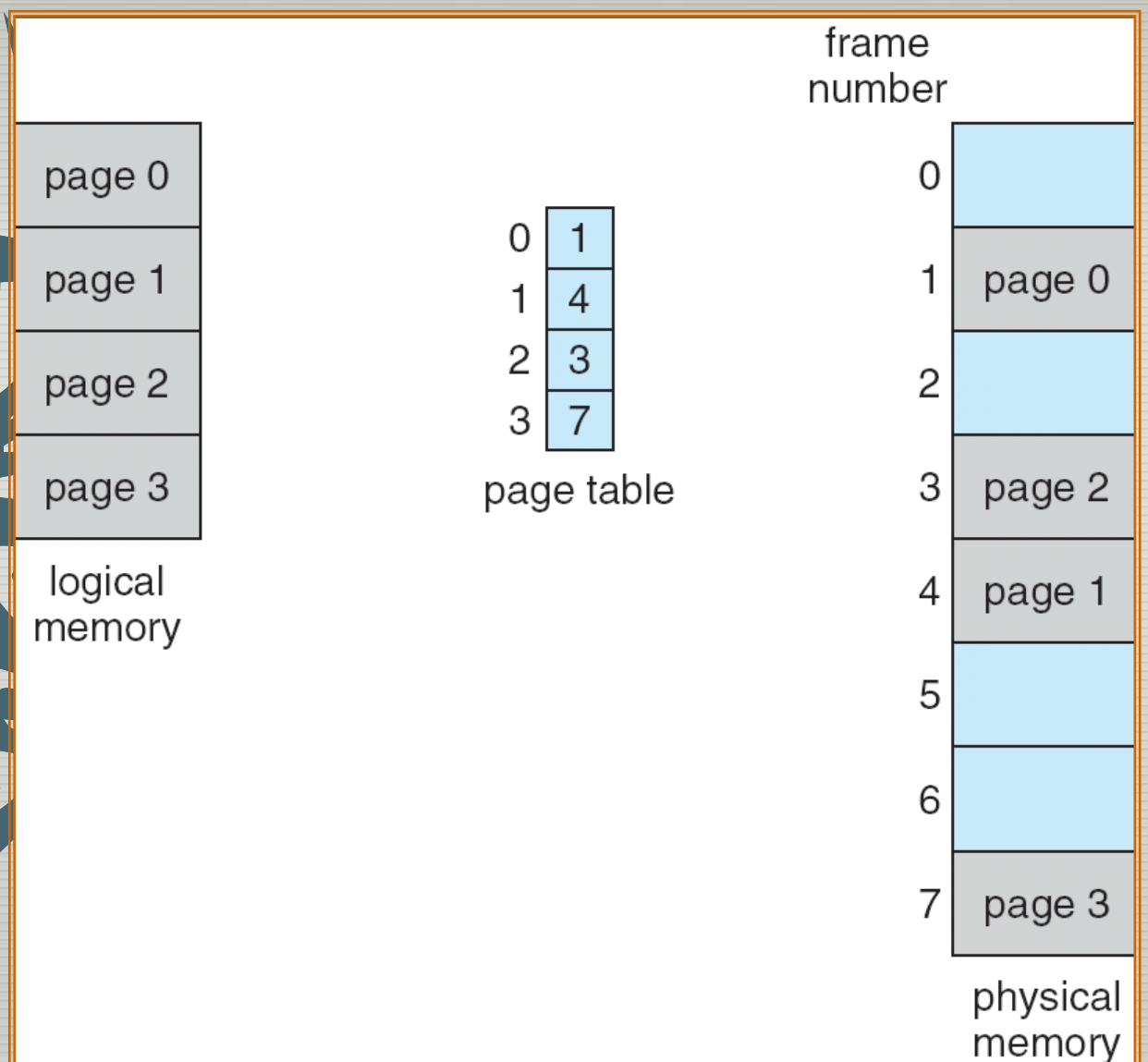
Basic Method

- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes)
- 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
- Internal fragmentation

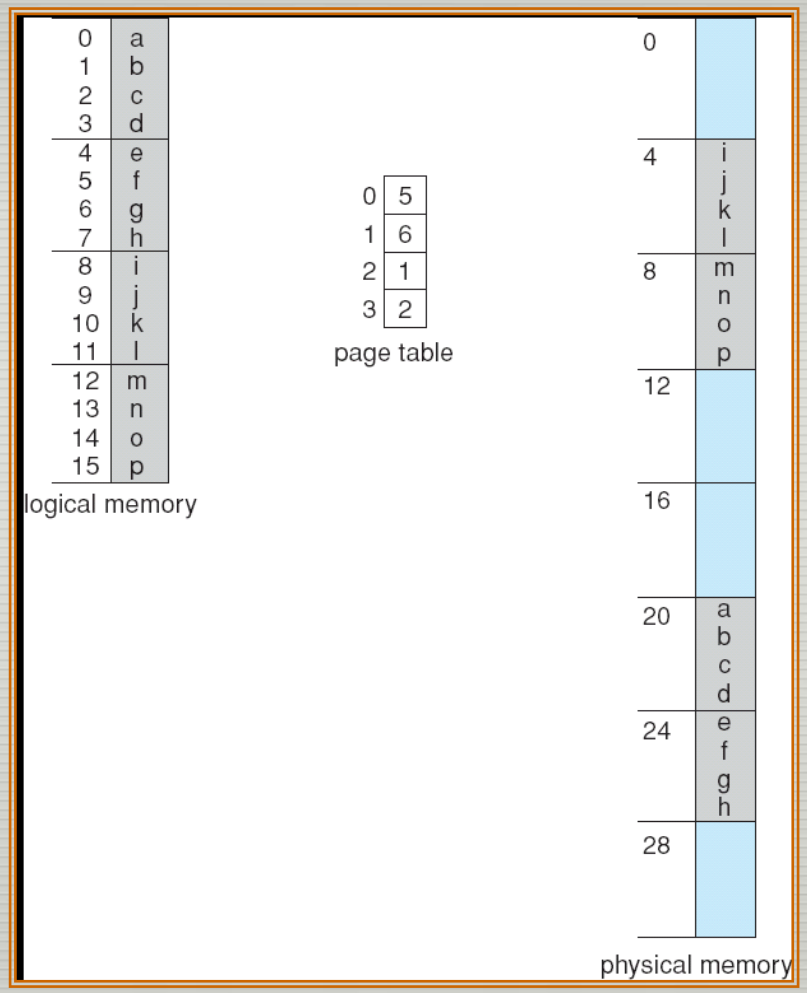
Basic Method



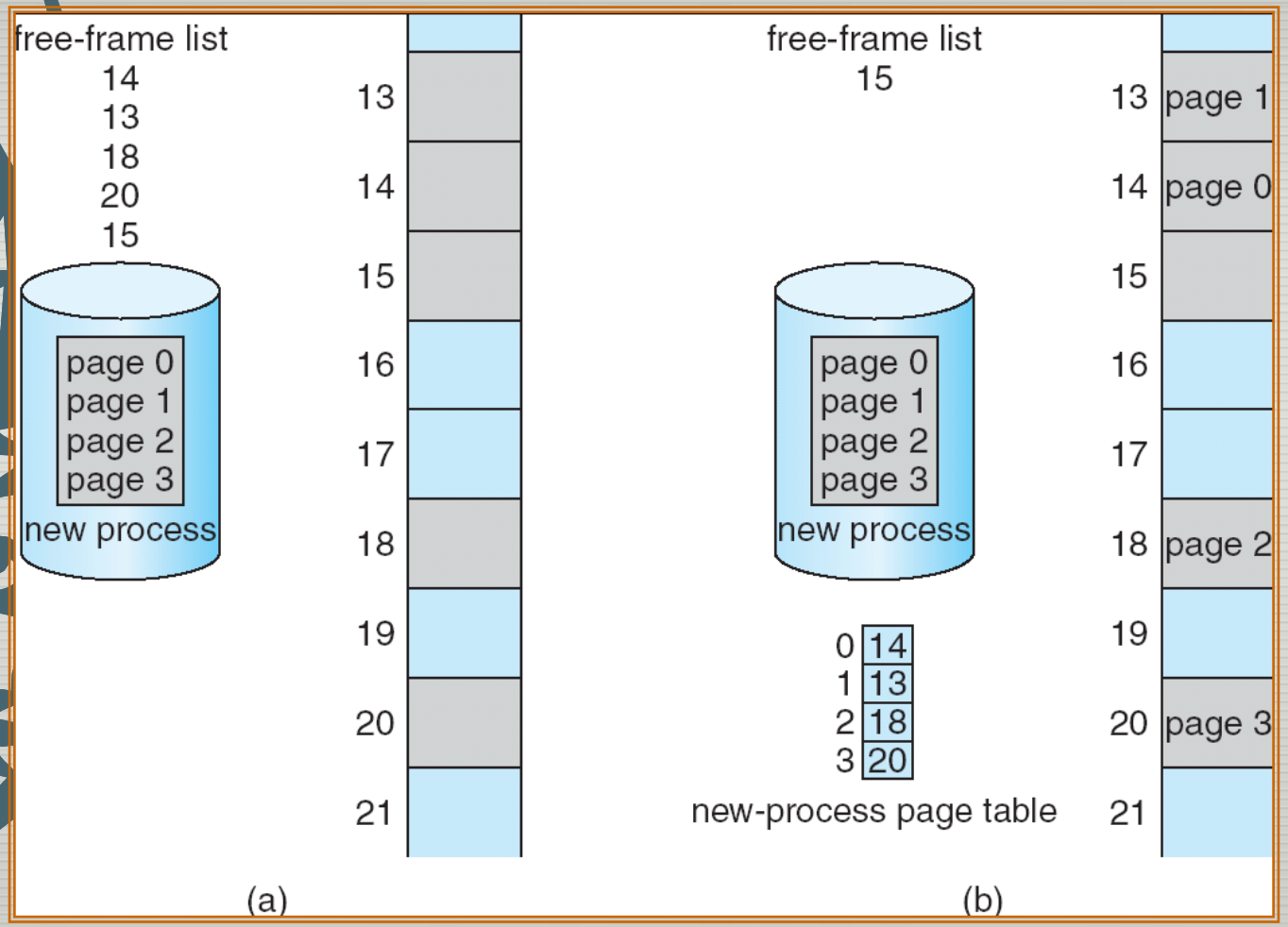
Basic Method



Basic Method



Basic Method



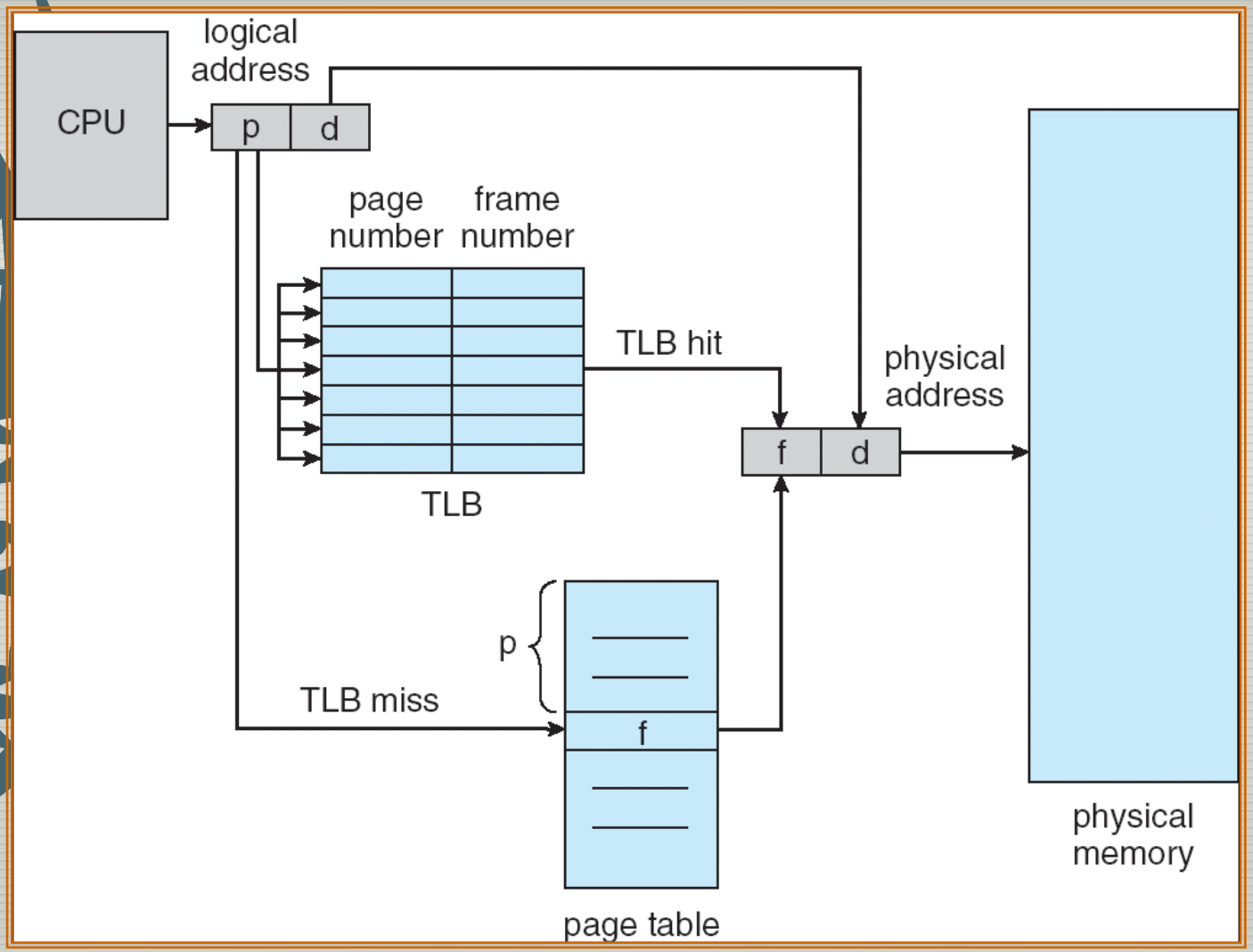
Hardware Support

- Most operating systems allocate page table for each process.
- A pointer to the page table is stored with the other register values in the process control block.
- In the simplest case, the page table is implemented as a set of dedicated registers.
- The use of registers for page table is satisfactory if the page table is reasonable small. – Keeping page table in memory and page table base register(PTBR)

Hardware Support

- **PTBR** – Two memory accesses are needed to access a byte.(for accessing **page table** entry and for **byte**) – translation look-aside buffer(TLB)
- The TLB is associative, high-speed memory.

Hardware Support



Hardware Support

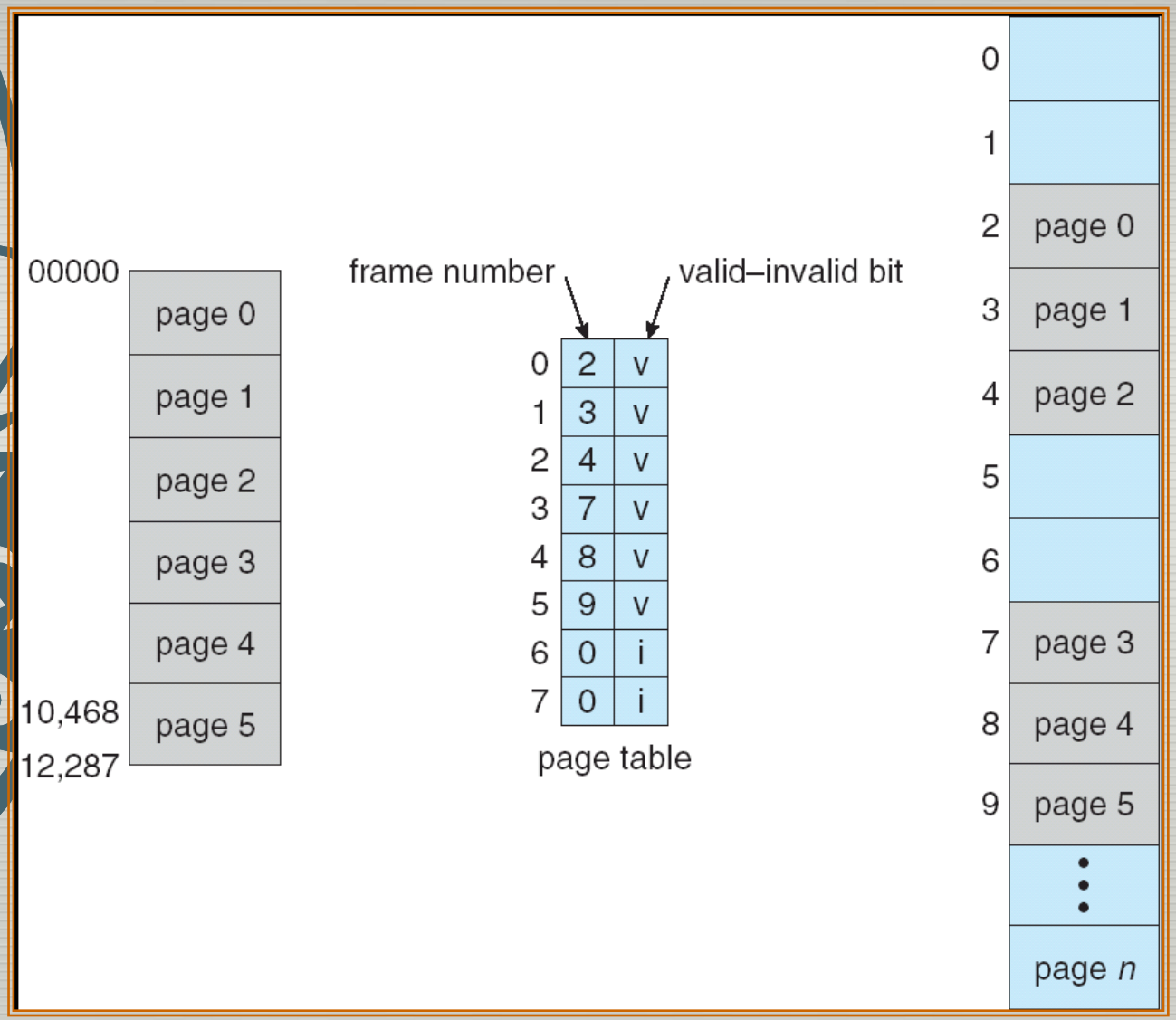
- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – **percentage of times** that a page number is found in the associative registers; ration related to number of associative registers
- Hit ratio = α
- **Effective Access Time** (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

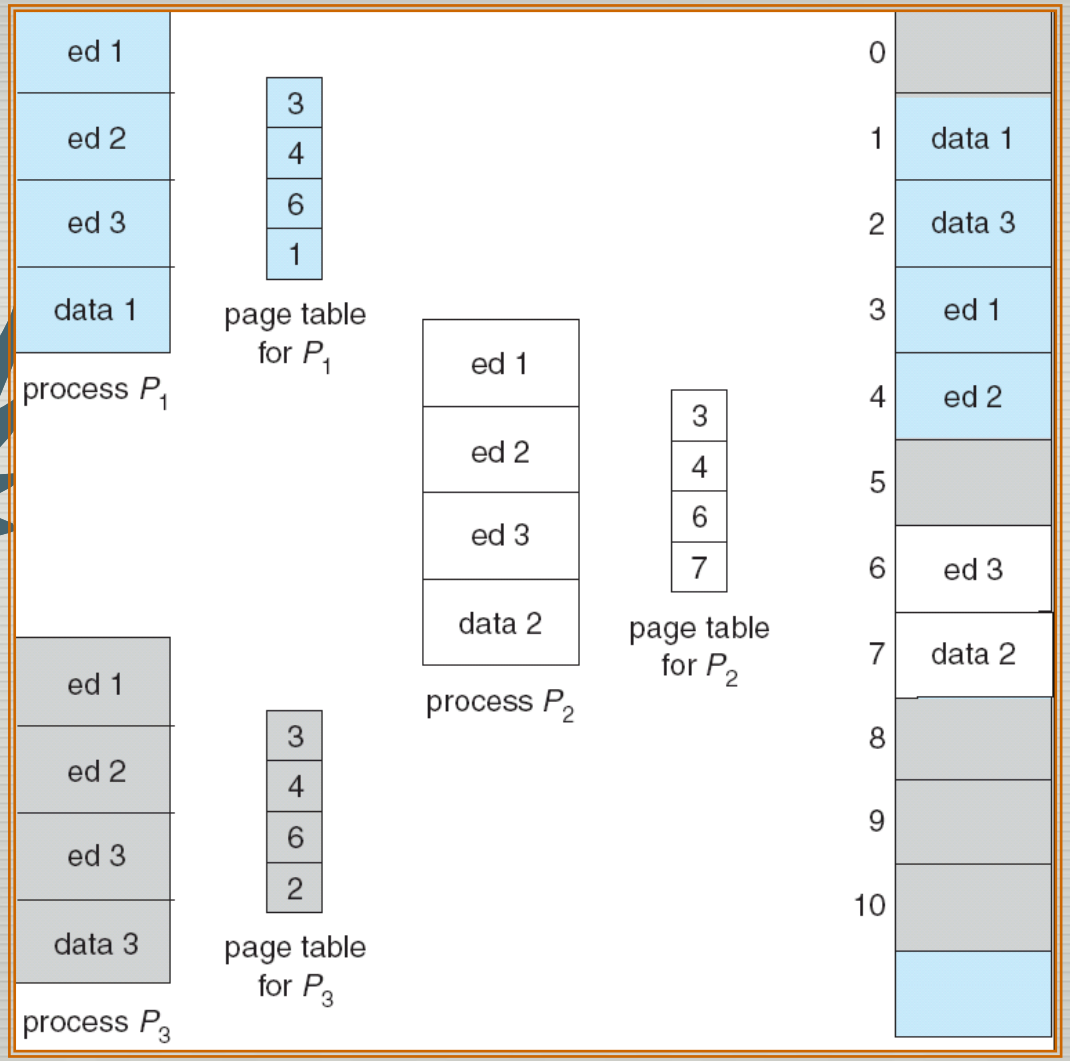
Protection

- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space

Protection


00000
page 0
page 1
page 2
page 3
page 4
10,468
page 5
12,287
frame number
valid–invalid bit
0 2 v
1 3 v
2 4 v
3 7 v
4 8 v
5 9 v
6 0 i
7 0 i
page table
0
1
2 page 0
3 page 1
4 page 2
5
6
7 page 3
8 page 4
9 page 5
page n

Shared Pages



Structure of the Page Table

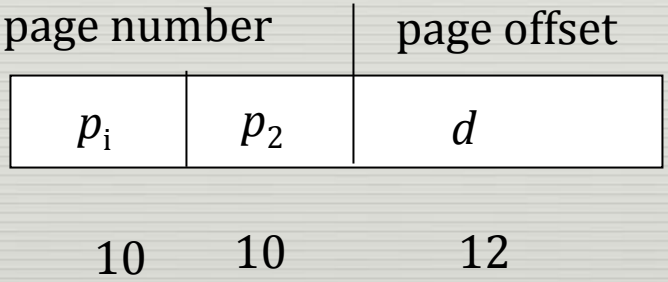
Hierarchical Paging

Hashed Page Tables

Inverted Page Tables

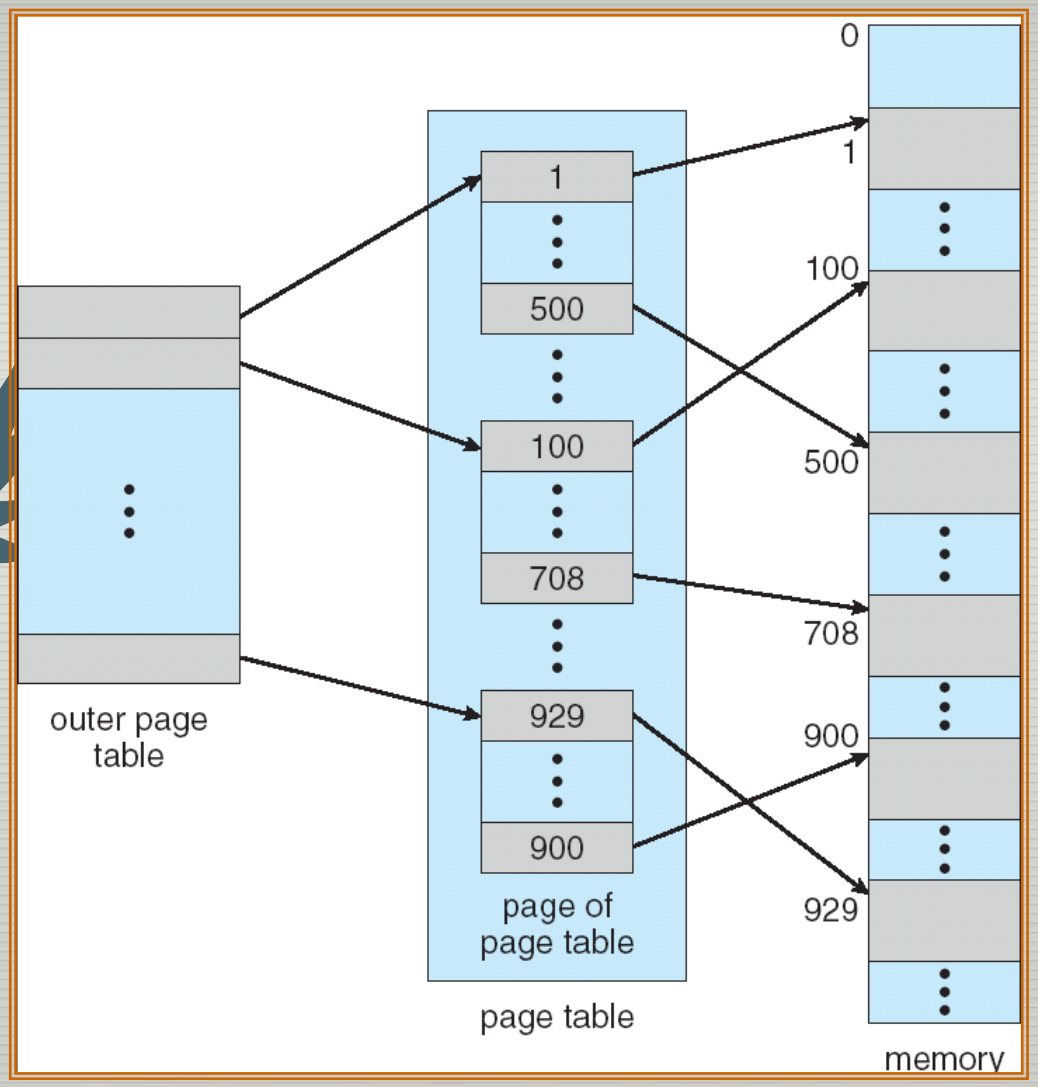
Hierarchical Paging

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits
  - a page offset consisting of 12 bits
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number
  - a 10-bit page offset

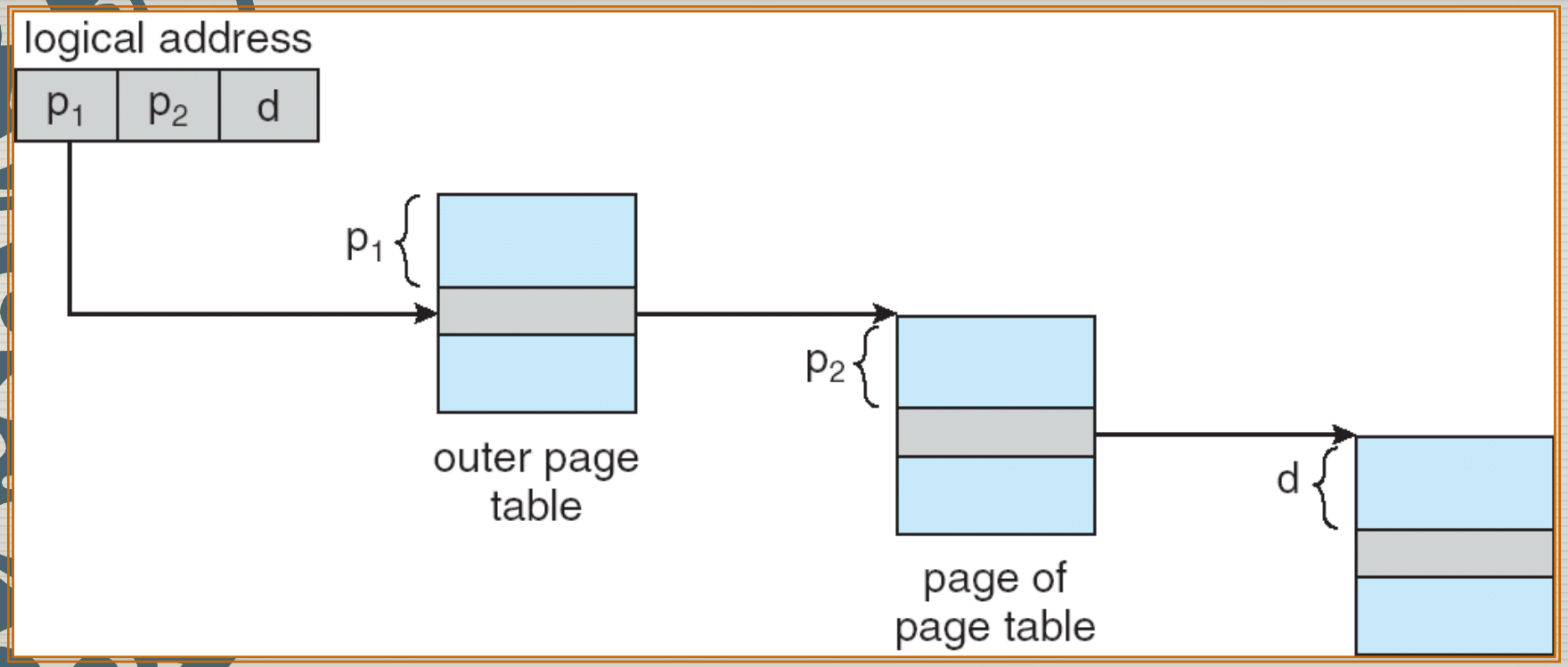


where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Hierarchical Paging

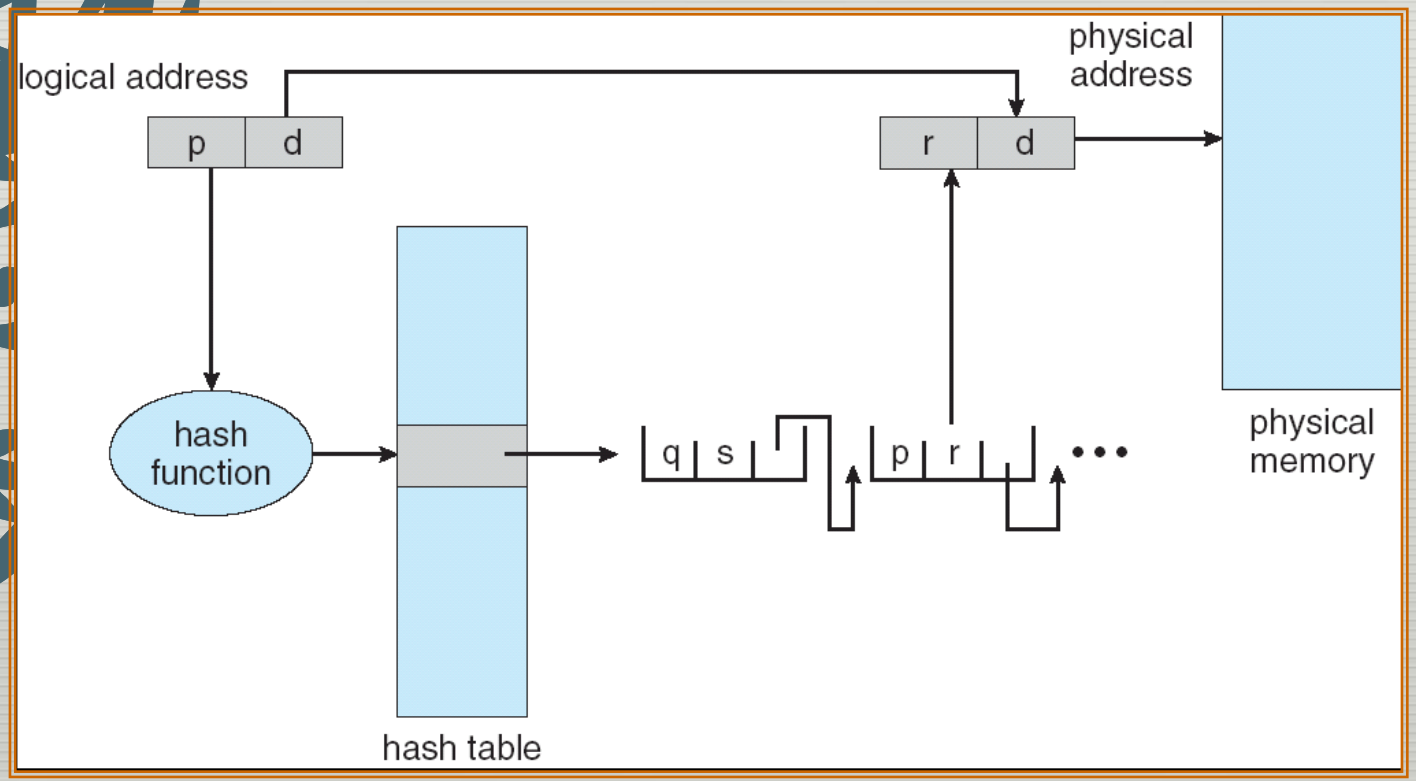


Hierarchical Paging



Hashed Page Tables

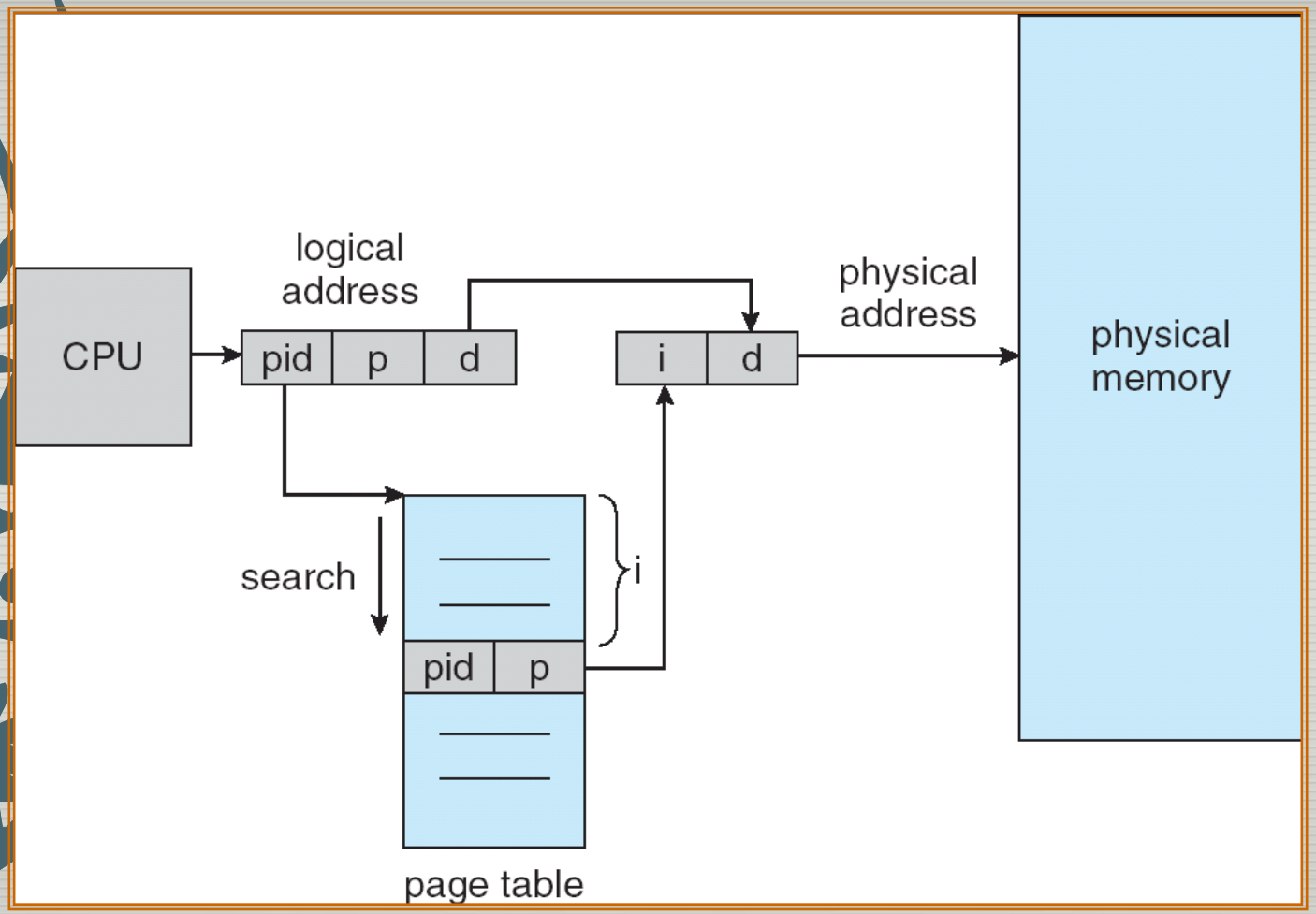
- Common in address spaces > 32 bits
- **The virtual page number is hashed** into a page table. This page table contains a chain of elements hashing to the same location.



Inverted Page Tables

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- **Decreases memory needed to store** each page table, but **increases time needed to search the table** when a page reference occurs
- **Use hash table** to limit the search to one - or at most a few - page-table entries

Inverted Page Tables

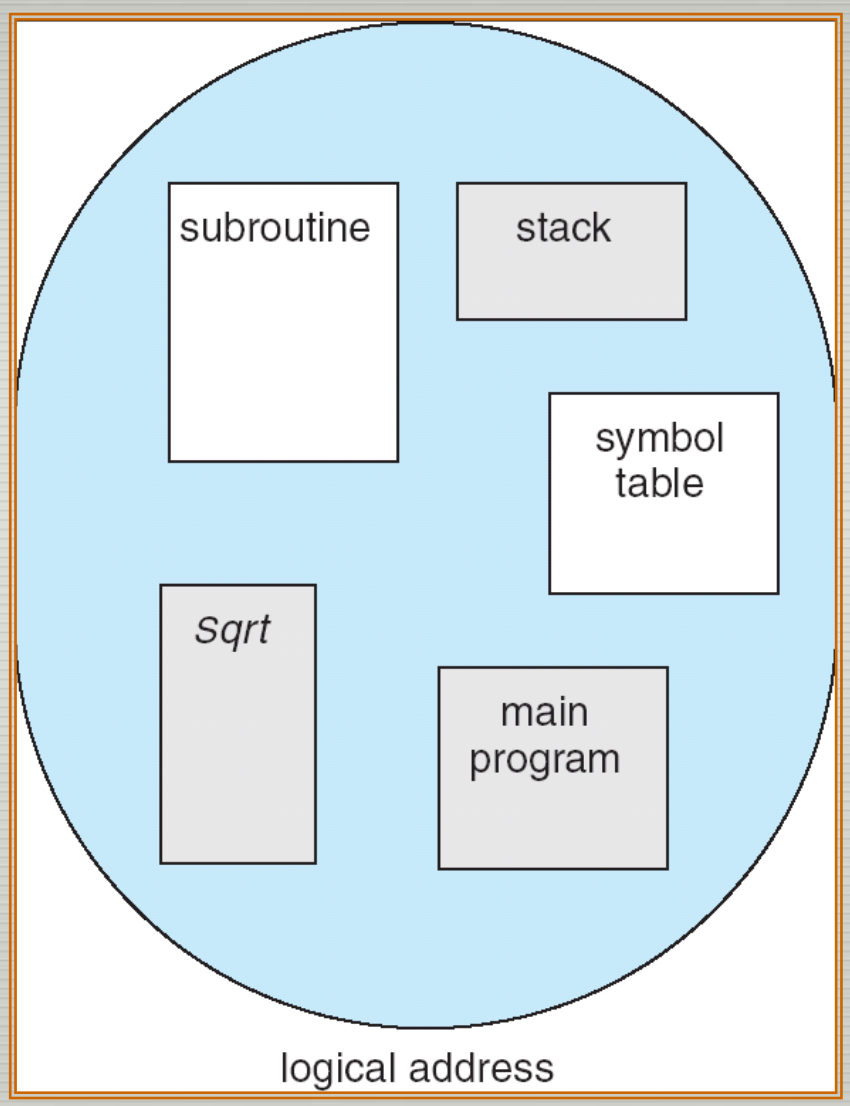


Segmentation

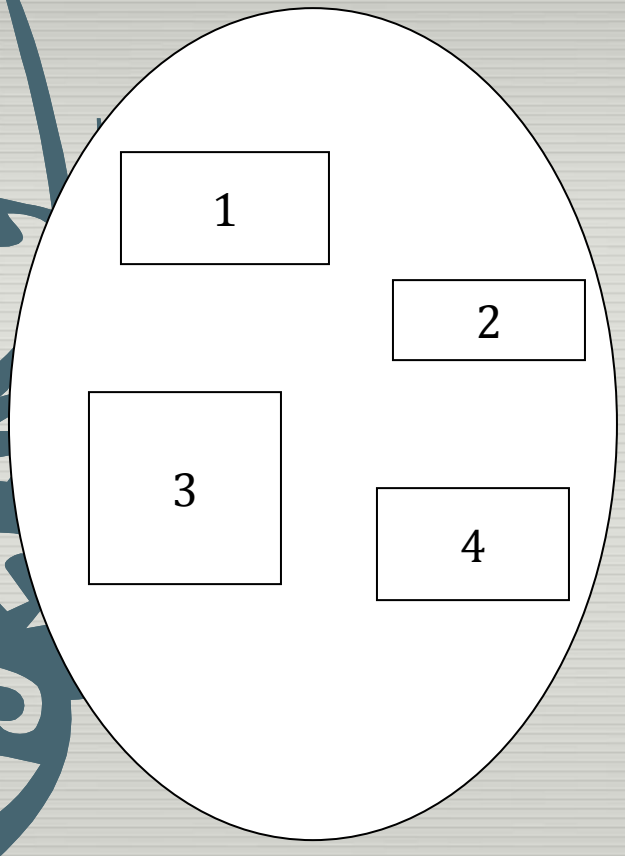
User's view of a Program

Segmentation

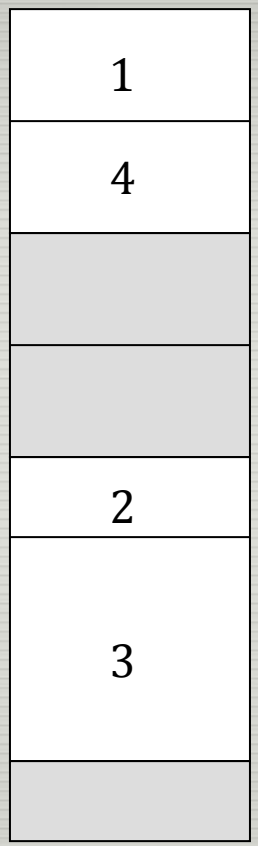
User's view of a Program



Segmentation



user space



physical memory space

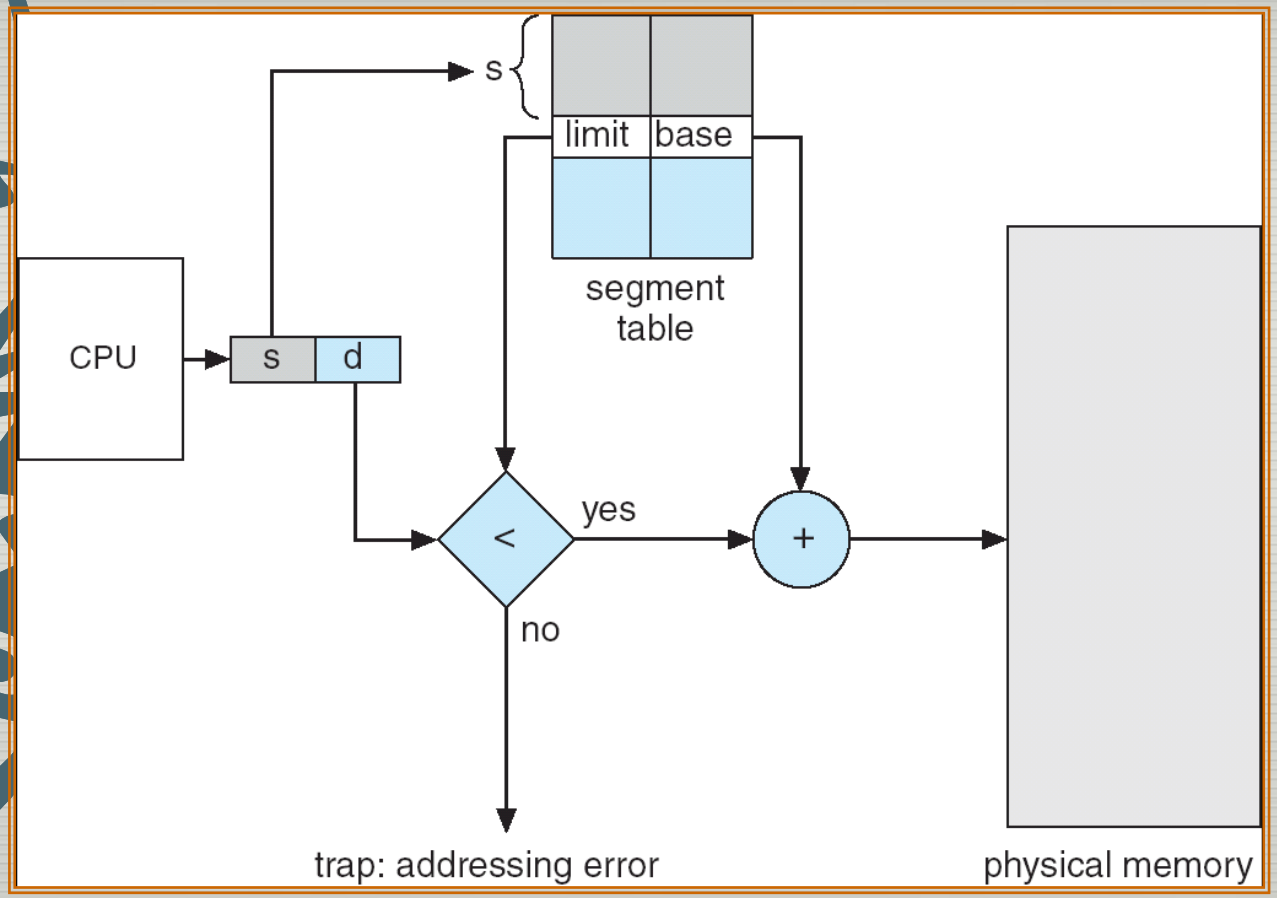
Segmentation

- Logical address consists of a two tuple:

  <segment-number, offset>,
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory
  - limit – specifies the length of the segment
- Segment-table base register (STBR) points to the segment table's location in memory
- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number s is legal if s < STLR

Segmentation



Segmentation