OPERATING SYSTEMS (THEORY)

LECTURE - 9

K.ARIVUSELVAN

Assistant Professor (Senior) – (SITE)

VIT University

MEMORY MANAGEMENT

Background

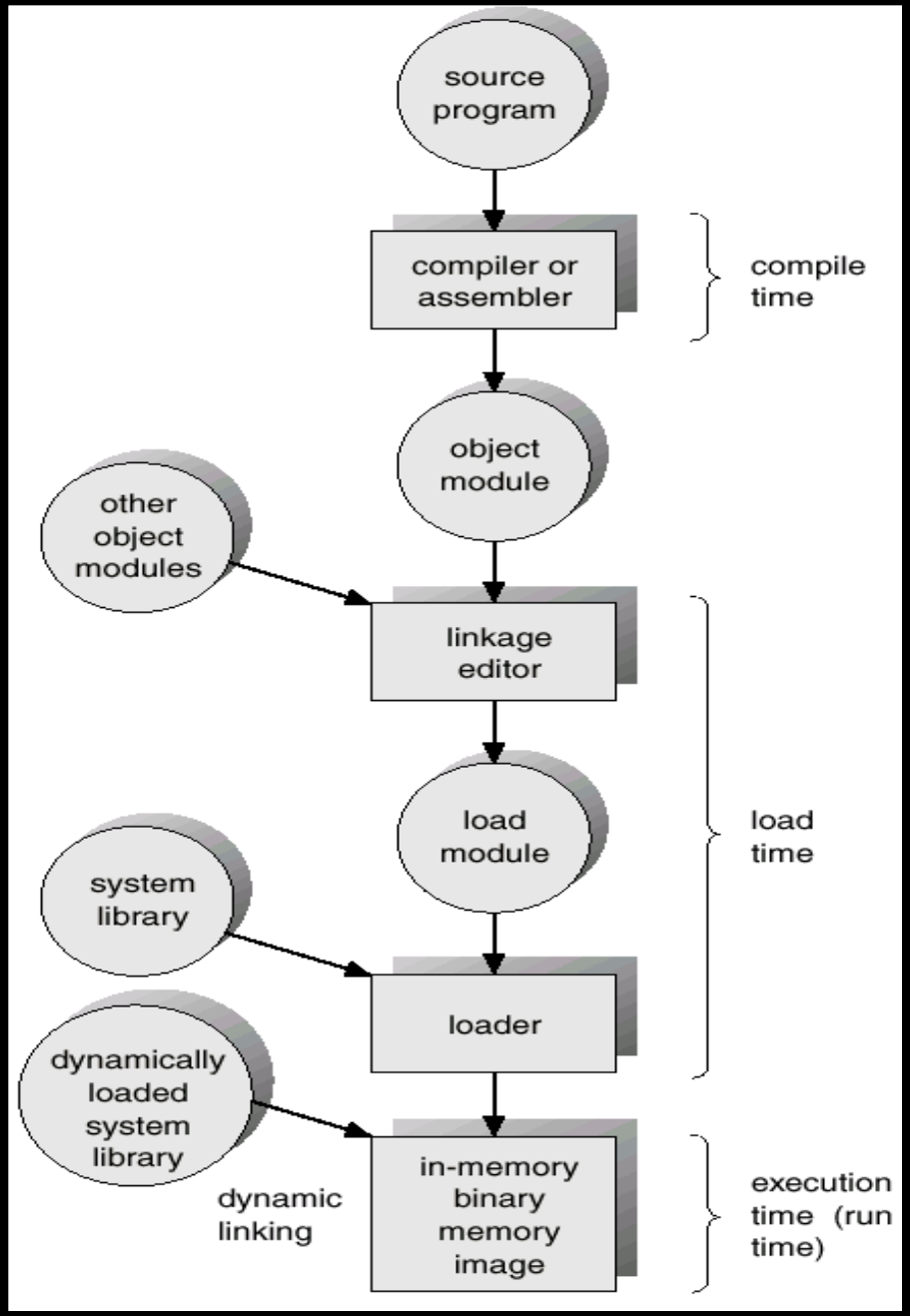
- Memory consists of a large **array of words or bytes**, each with its **own address**.

- The **CPU fetches** instructions from **memory** according to the value of the **program counter.**

- **Memory unit** sees only a stream of **memory addresses**.

- User programs go through several steps before being run.

Multistep Processing of a User Program



Binding of Instructions and Data to Memory

- **Address binding** of **instructions and data to memory addresses** can happen at three different stages.

(1) Compile time: If memory location known a **priori**, **absolute code** can be generated; must recompile code if starting location changes.

Example: .COM-format programs in MS-DOS.

(2) Load time: Must generate ***relocatable* code** if memory location is **not known** at compile time.

Binding of Instructions and Data to Memory

(3) Execution time: **Binding delayed** until run time, if the process can be moved during its execution from one memory segment to another.

- **Need Special hardware support** for address maps

(1) 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.

(2) Dynamic Linking

- Linking is postponed until execution time.

- Small piece of code, *stub*, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.

Overlays

For execution, entire program and data must be in Physical memory.

What happens, if a process size is larger than Memory allocated?

Keep in memory only the overlay (those instructions and data that are) needed at any given phase/time.

Overlays are designed/implemented by programmer. Needs an overlay driver.

No special support needed from operating system, but program design of overlays structure is complex.

Overlays

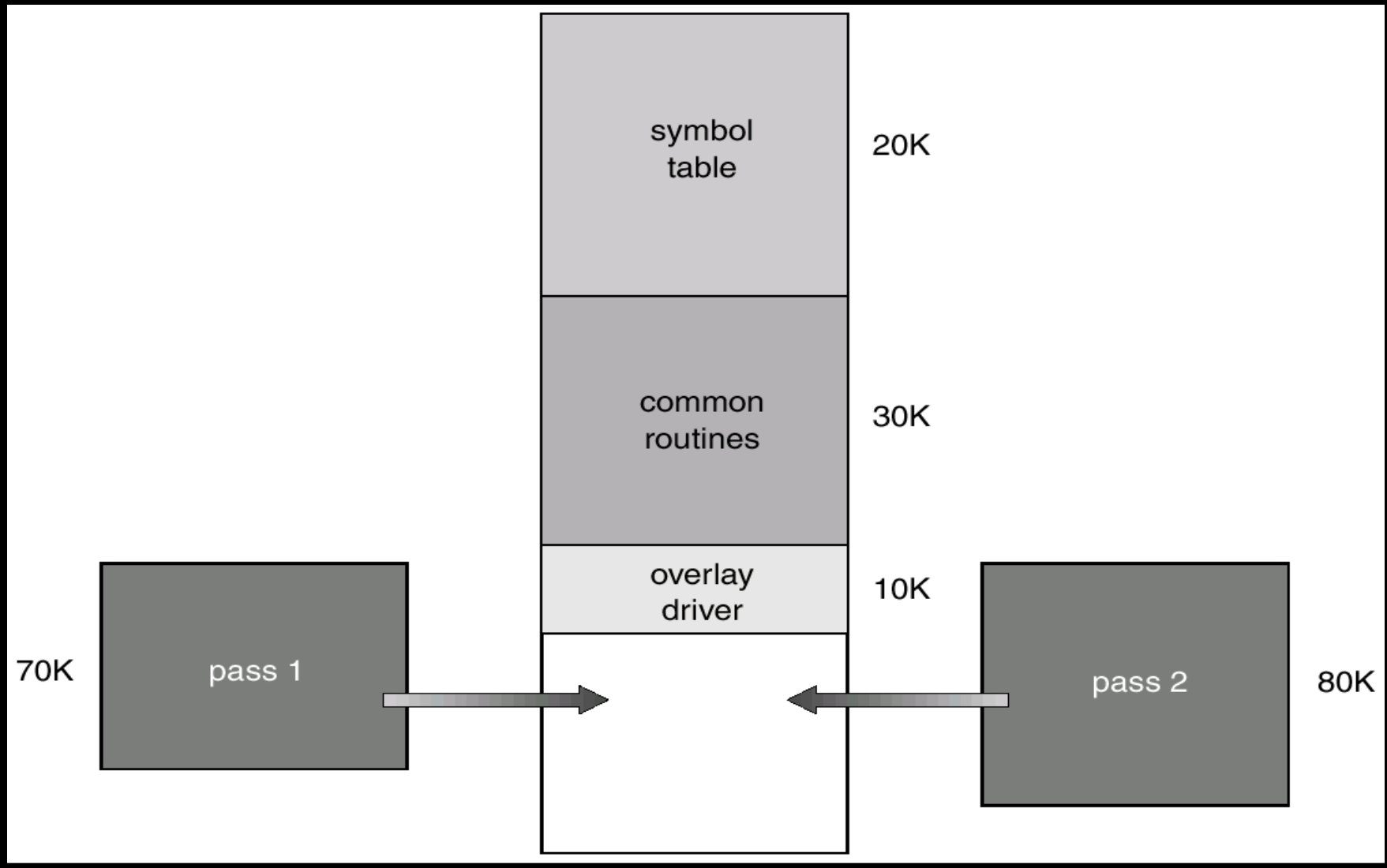
Pass 1	70 K
Pass 2	80 K
Symbol Table	20 K
Common Routines	30 K

- To load everything we would require 200 K of memory.
- What happens if available memory is only 150 K?

Solution:

- Define two Overlays.

Overlays for a Two-Pass Assembler



Logical vs. Physical Address Space

Logical address **– address generated by the CPU; also referred to as *virtual address*.**

Physical address **– address seen by the memory unit**

- **The set of all logical addresses generated by a program is a logical address space;**
- **The set of all physical addresses corresponding to these logical addresses is a physical address space.**

Memory Management Unit [MMU]

- Hardware device that maps virtual address to physical address.

- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

- In a simple MMU scheme,
- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.

Single-partition allocation

- **Main memory consists of:**

=> Operating system, usually held in low memory

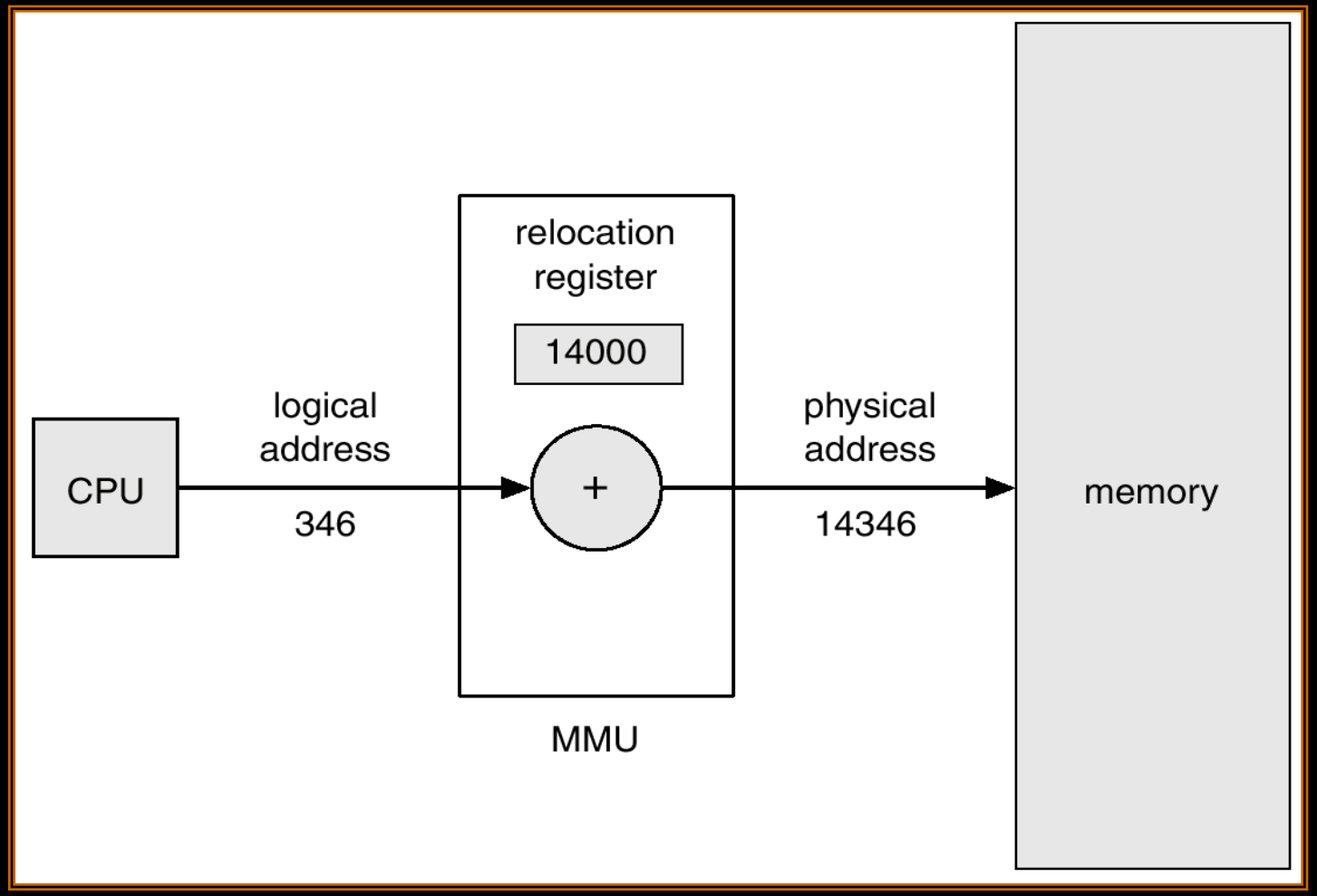
=> User processes, held in high memory

❑ **Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.**

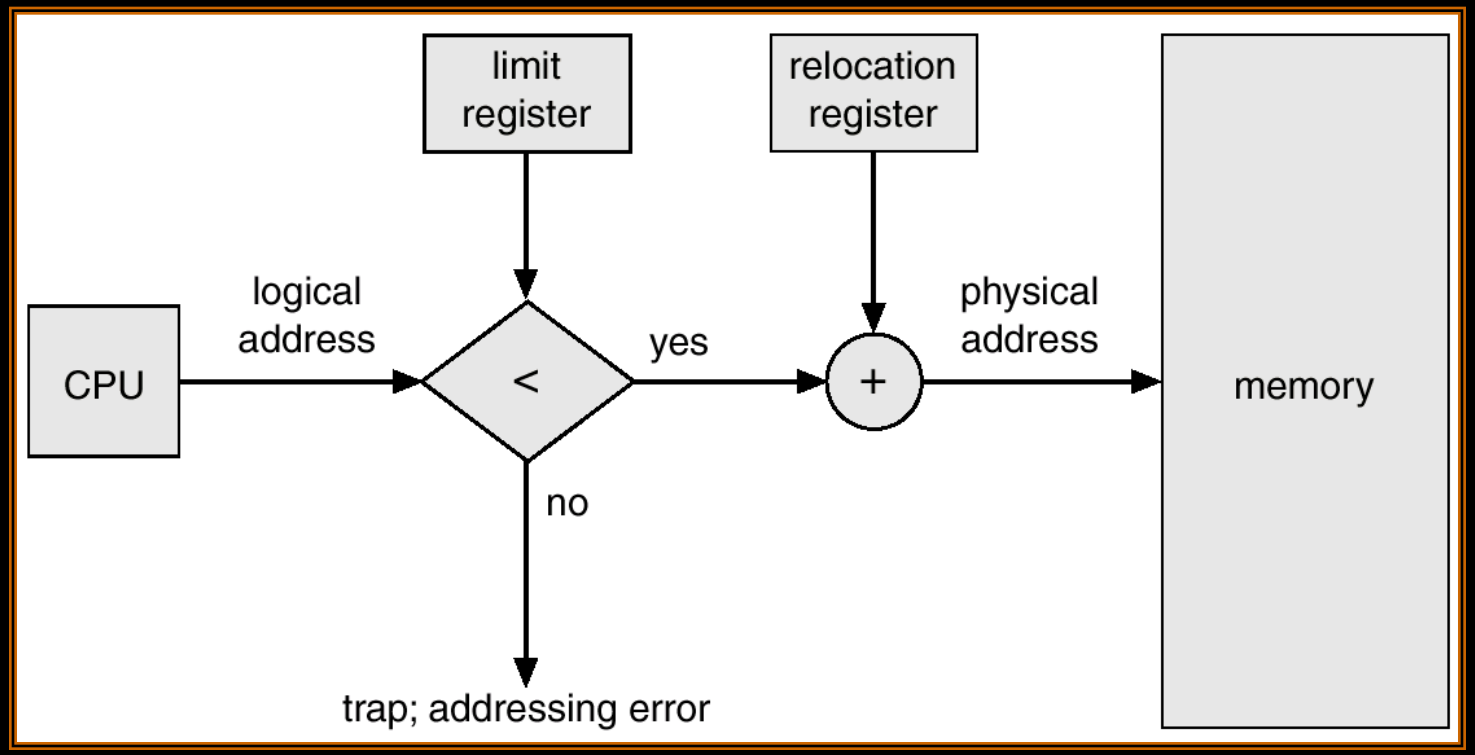
Single-partition allocation

- **Relocation register :** contains value of **smallest physical address**
- **Limit register :** contains **range of logical addresses**
- Each **logical address** must be **less than** the **limit register**

Dynamic relocation using a relocation register



Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

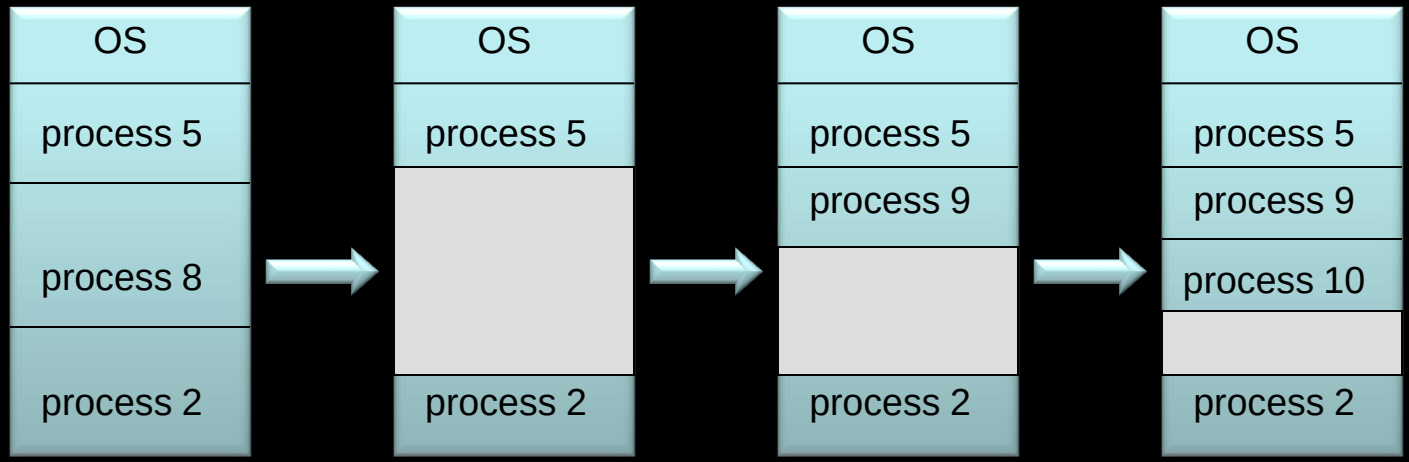
Problem:

- How to allocate available memory to the various processes

Solution:

- Divide the memory into a number of fixed-sized partitions

- Each partition contains one process



Dynamic Storage-Allocation Problem

- **How to satisfy a request of size *n* from a list of free partitions.**

- **First-fit**: Allocate the *first* partition that is big enough.

- **Best-fit**: Allocate the *smallest* partition that is big enough; must search entire list, unless ordered by size.

- **Worst-fit**: Allocate the *largest* partition; must also search entire list.

- First-fit and best-fit are better than worst-fit in terms of **speed and storage utilization**.

Fragmentation

- As a process which are loaded or removed from memory. the free memory space is broken into Little pieces, such types of pieces may or may not be of any use to be allocated individually to any process.
- This may give rise to term memory waste or fragmentation.

- Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

- External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous.

How to reduce external fragmentation ?

Solution:

Using technique called compaction

Compaction:

- **Shuffle memory contents to place all free memory together in one large block.**

Paging

- Logical memory is divided into Fixed Size blocks called pages

- Physical memory is divided into fixed-sized blocks called frames (size is power of 2, for example 512 bytes).

- When a process is to be executed, its Pages are loaded into any available memory frames from backing store.

Address Translation Scheme

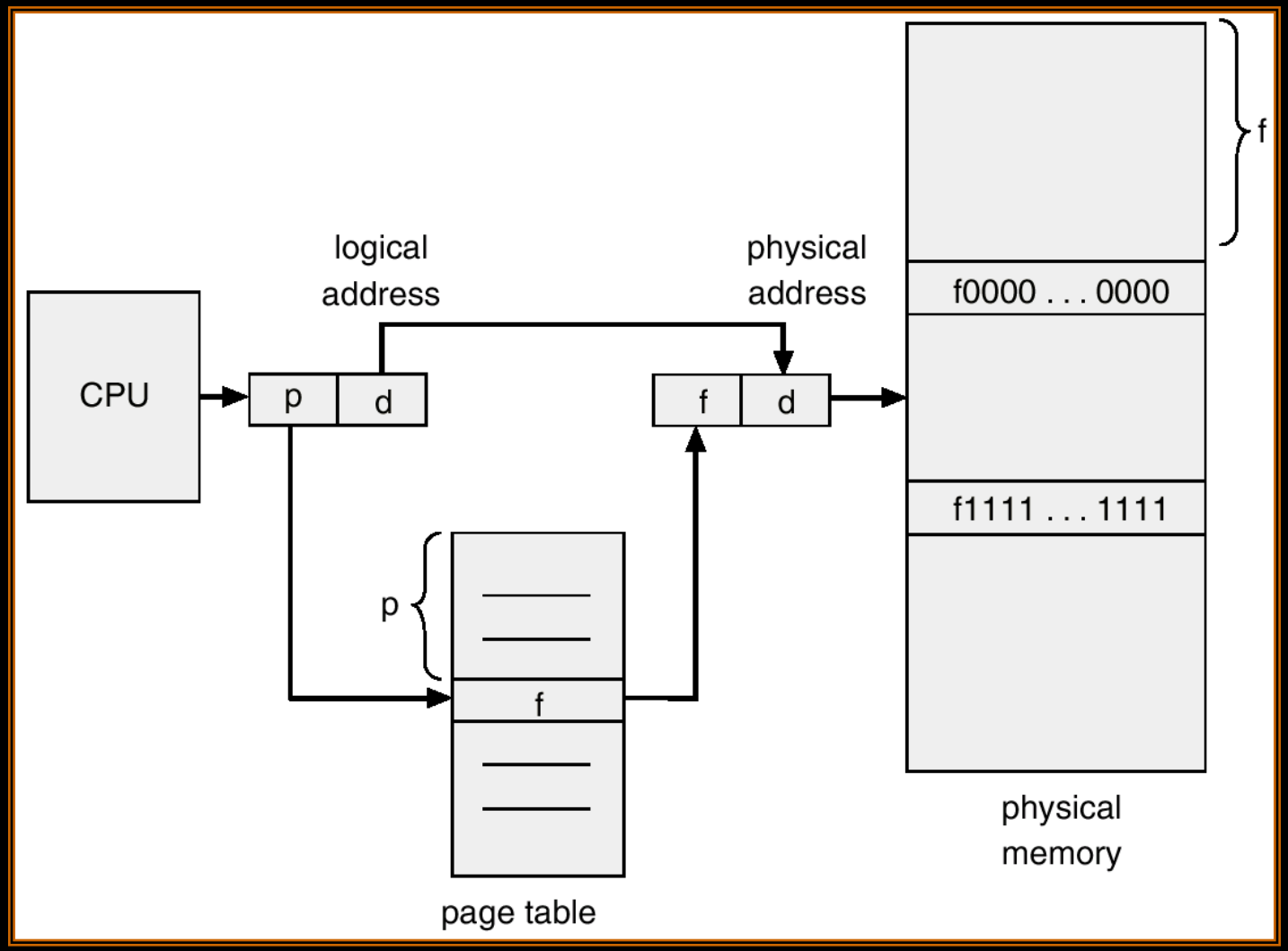
❑ **Address generated by CPU is divided into:**

➢ ***Page number (p)*** **– used as an index into a** ***page table*** **which contains base address(Frame no.) of each page in physical memory.**

➢ ***Page offset (d) – combined with base address (Frame no.) to define the physical memory address that is sent to the memory unit.***

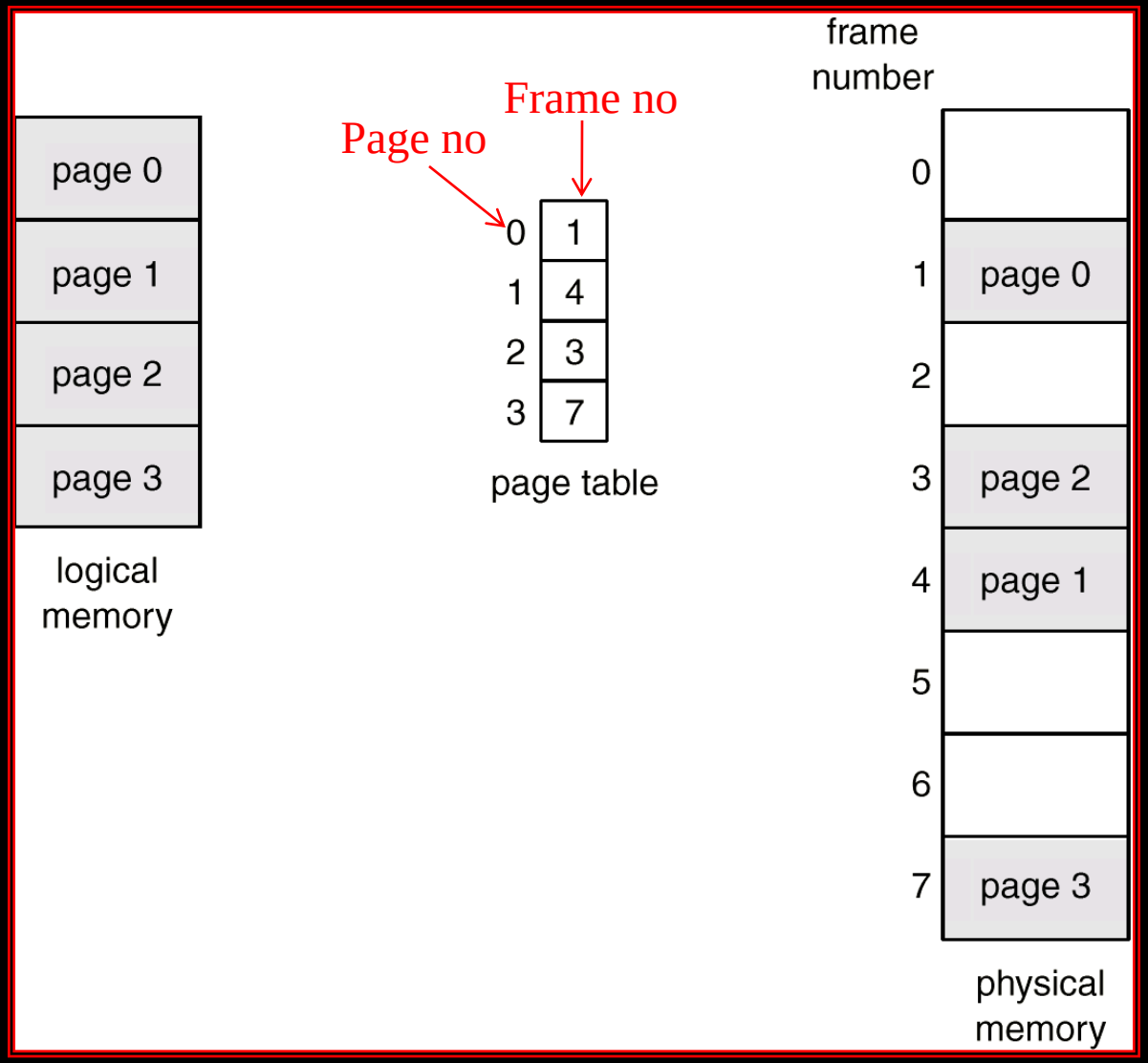
❑ **Size of a page = Power of 2 (varying between 512 bytes & 8192 bytes per page)**

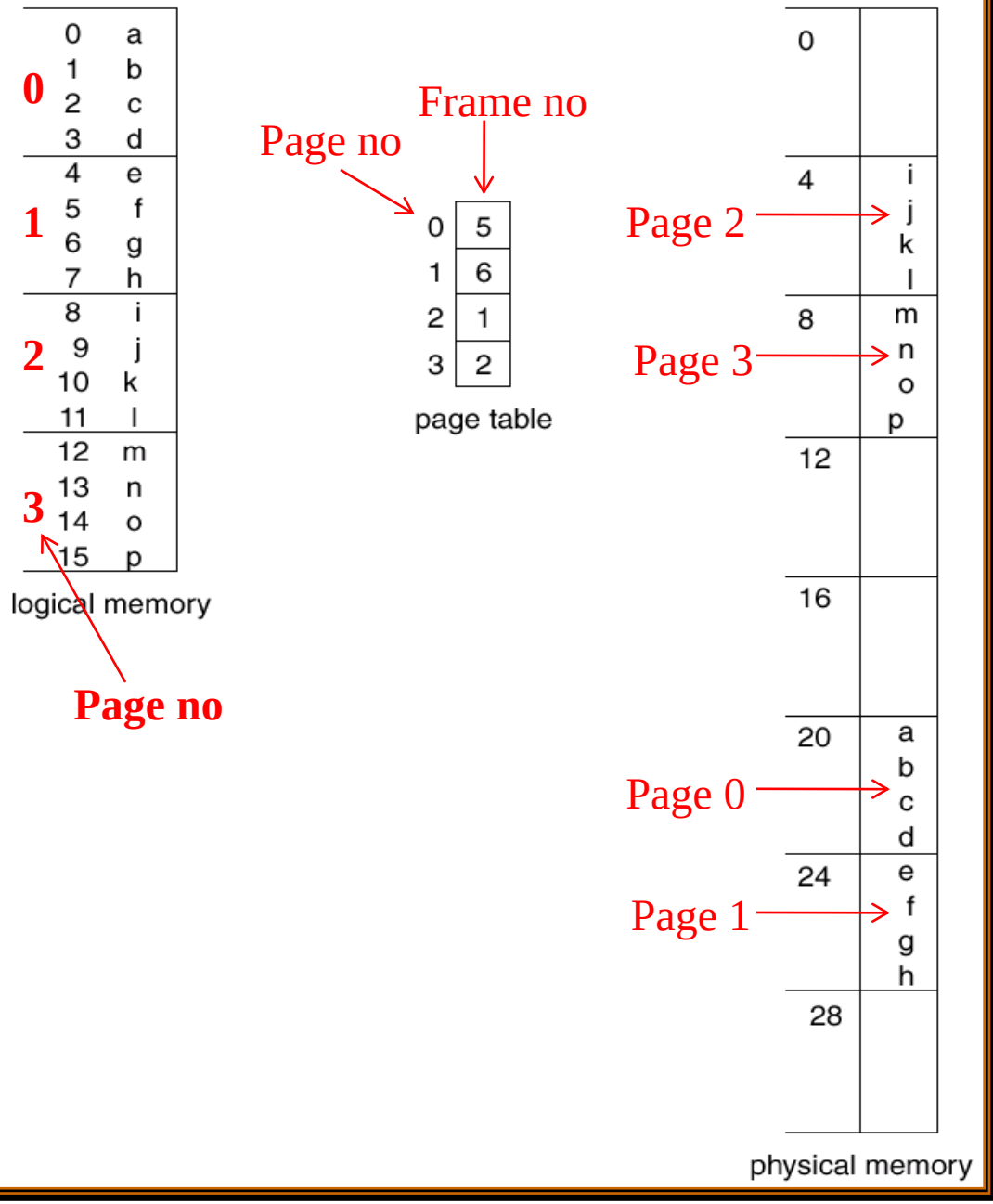
Address Translation Architecture



Paging Hardware

Paging model of logical and physical memory





Paging example for a 32-byte memory with 4-byte pages

Mapping Logical Address to Physical Address

- Physical Memory size = 32 bytes
- Page size = 4 bytes

Physical Address = (Frame number x page size) + Offset value

- Logical address 0 (page 0 , offset 0) maps to Physical address 20

  ((5 x 4) + 0) = 20

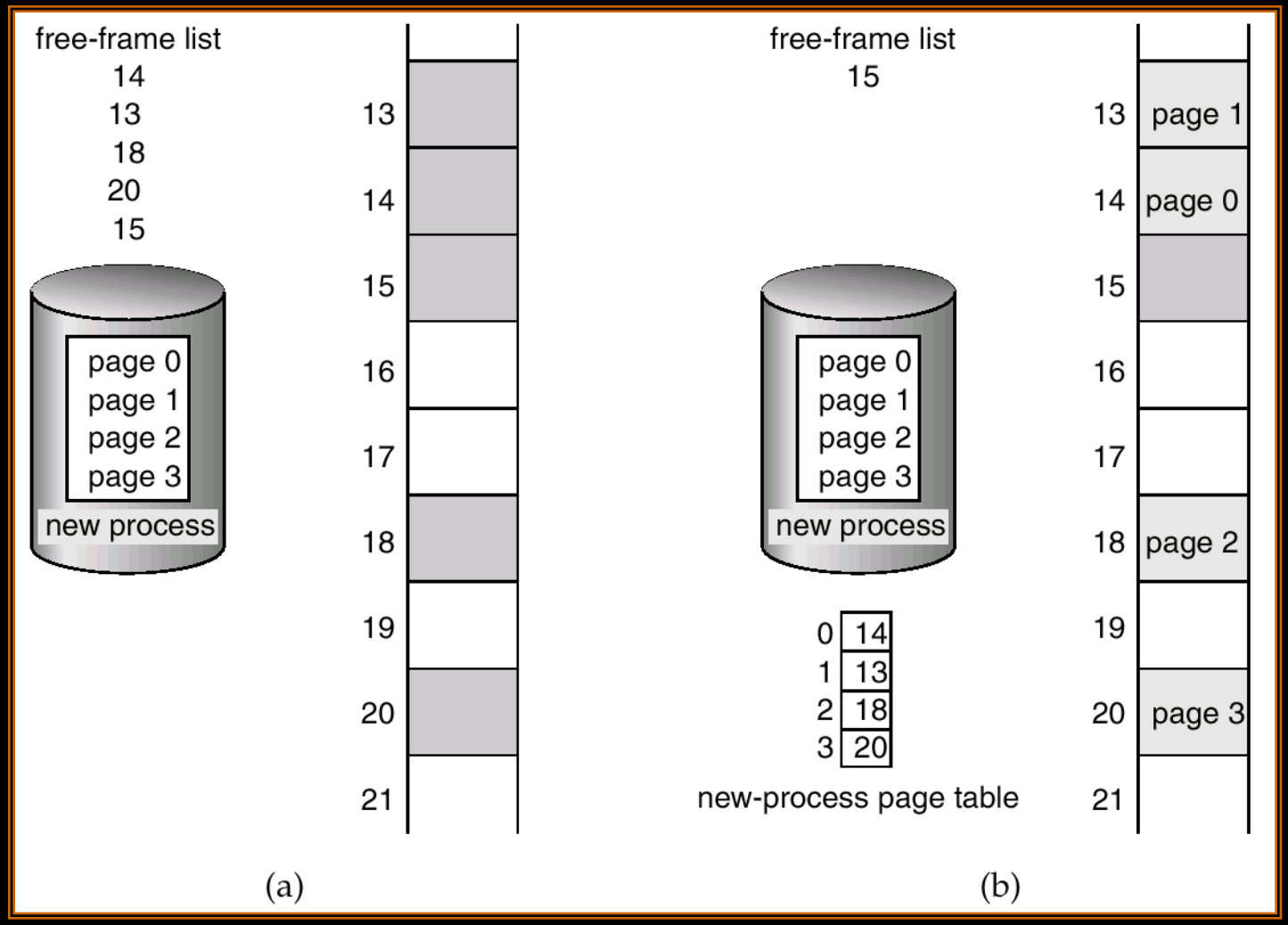
- Logical address 3 (page 0 , offset 3) maps to Physical address 23

  ((5 x 4) + 3) = 23

- Logical address 4 (page 1 , offset 0) maps to Physical address 24

  ((6 x 4) + 0) = 24

Free Frames



Before allocation

After allocation

Paging

Paging Scheme : (Con's)

- Internal fragmentation may occur

i.e. Memory requirements of a process do not fall on page boundaries , last frame allocated may not be completely full.

Example:

- Page Size = 2^{11} (2048 bytes)
- Process Size = 72,766 bytes (Need 35 Pages + 1086 bytes)

- System Allocate 36 Pages for the process
- So, 2048 – 1086 = 962 bytes are Free, which results in Internal Fragmentation.

Implementation of Page Table

- Page table is kept in **main memory.**

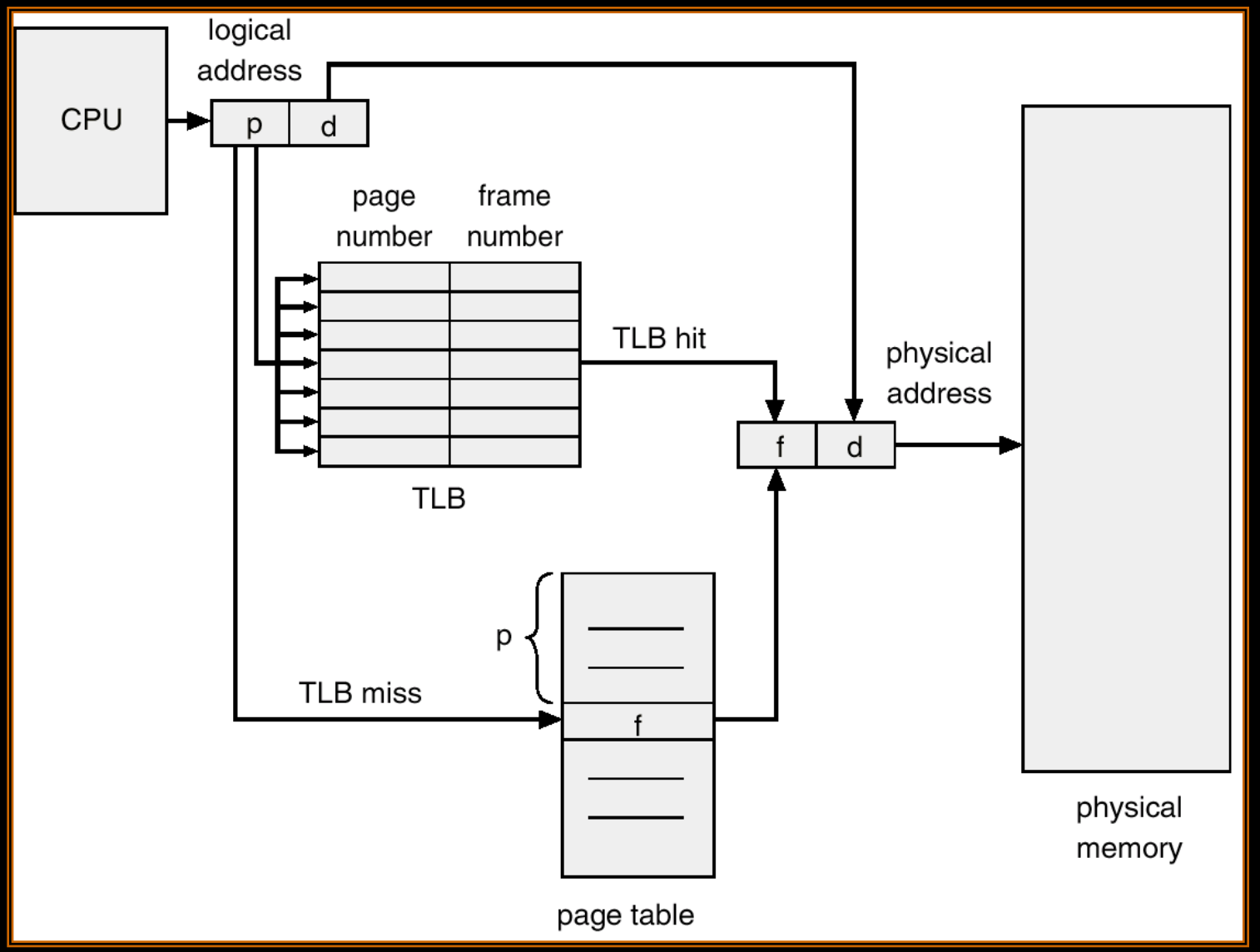
- ***Page-table base register*** **(PTBR)** points to the **page table.**

- In this scheme every **data/instruction-byte** access requires **two memory accesses.** One for the **page-table entry** and one for the **byte**

- The **two memory access problem** can be solved by the use of a special **fast-lookup hardware cache** called ***associative registers*** or ***translation look-aside buffers (TLBs).***

- Typically, the **number of entries** in a TLB is between **32 and 1024.**

Paging Hardware With TLB



Hit Ratio

- **Hit Ratio:** the **percentage of times** that a page number is found in the associative registers

- The Intel 80486 CPU has **32 associative registers**, and claims a **98-percent hit ratio**.

Memory Protection

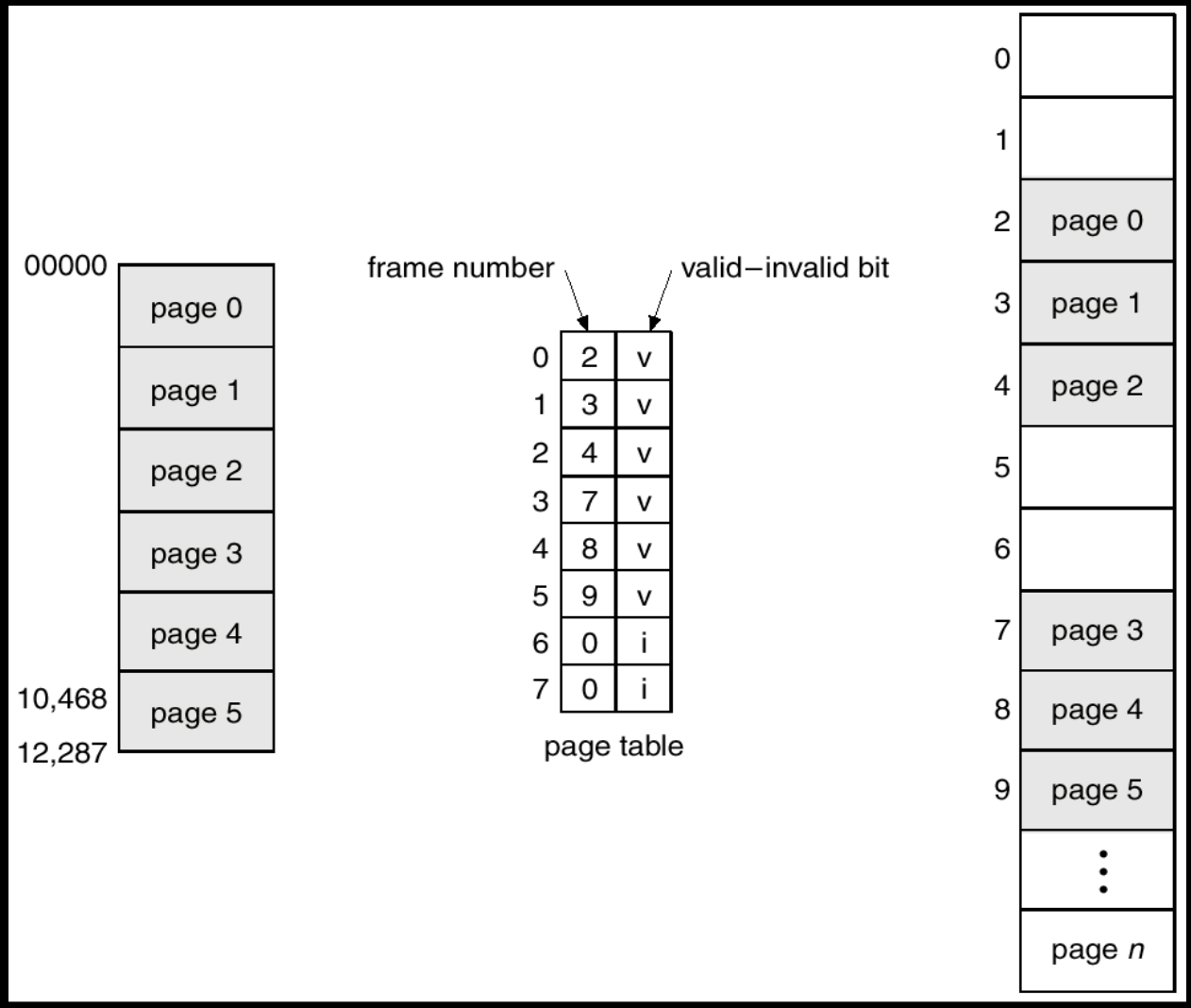
- **Memory protection implemented by associating protection bit with each frame.**

- ***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.**

Valid (v) or Invalid (i) Bit In A Page Table



Multilevel Paging

- Break up the logical address space into multiple page tables.

- A simple technique is a two-level page table.

Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:

=> a page number consisting of 20 bits.

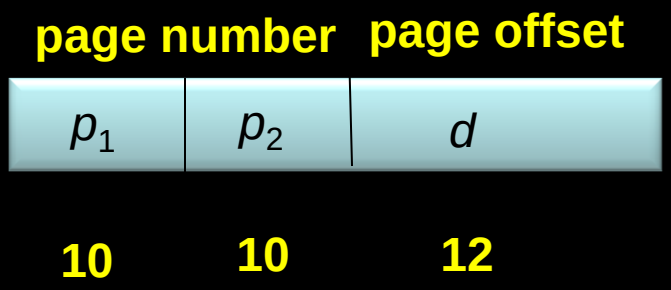
=> a page offset consisting of 12 bits.

- Since the page table is paged, the page number is further divided into:

=> a 10-bit page number.

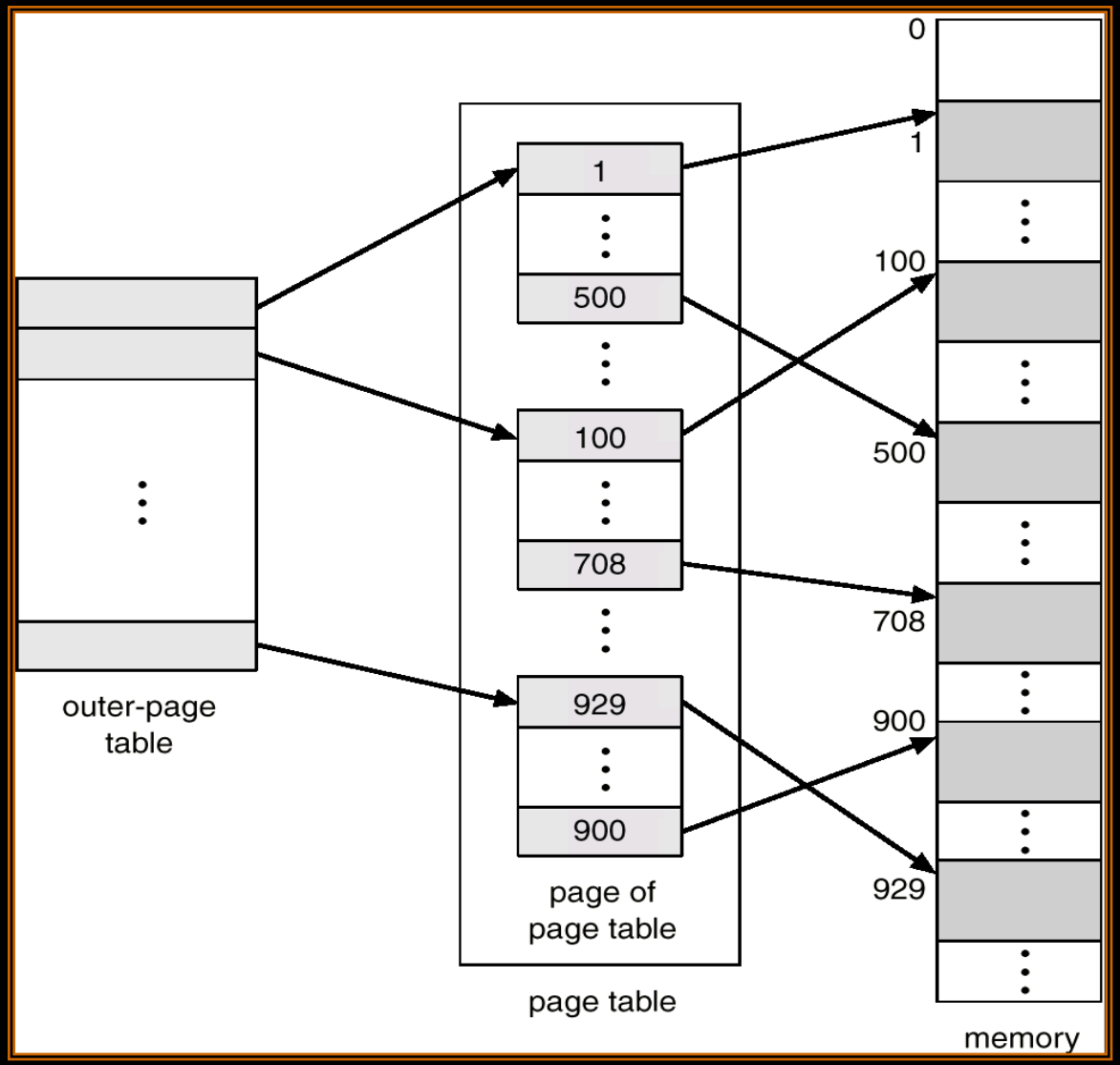
=> a 10-bit page offset.

- Thus, a **logical address** is as follows:

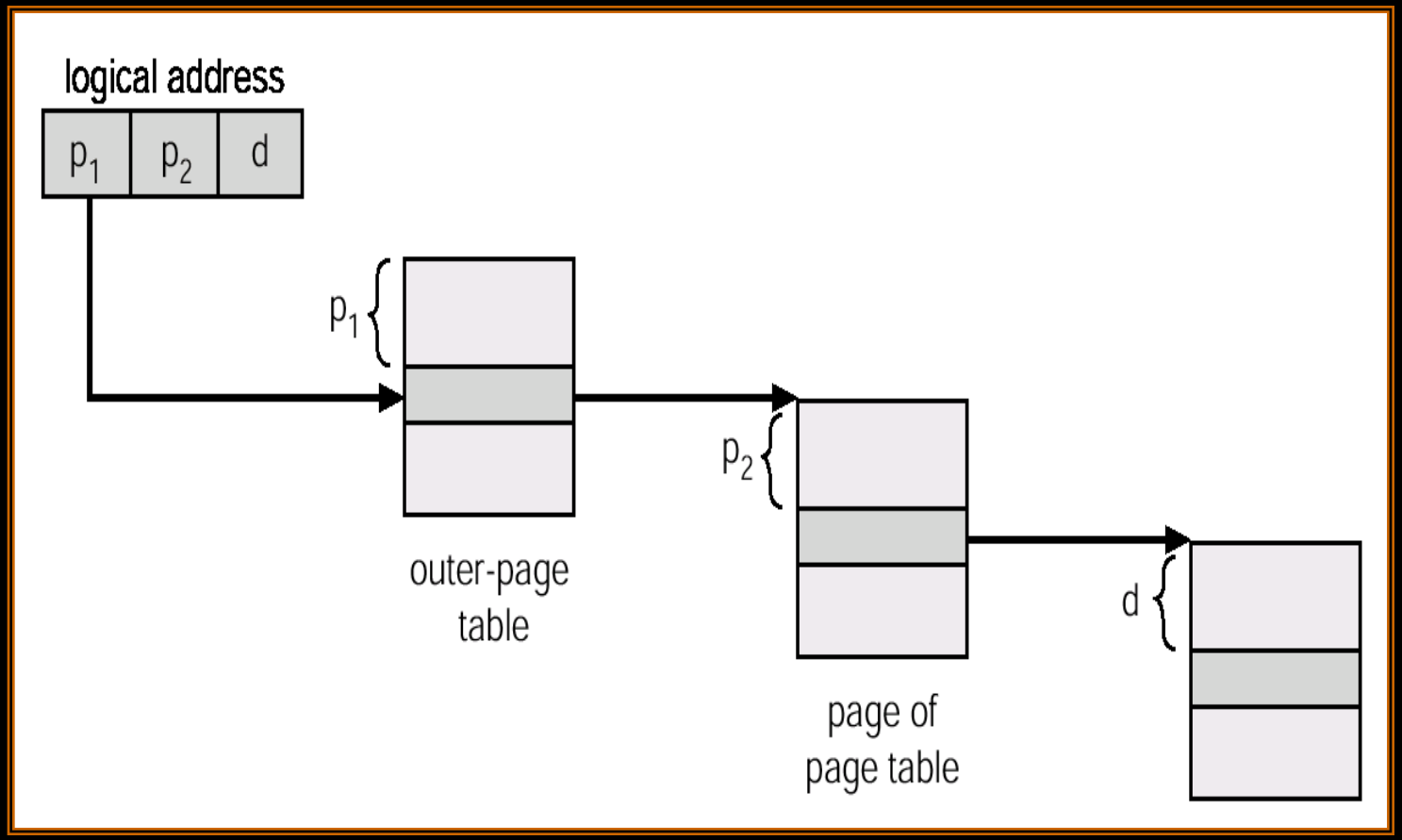


- where ***p1*** is an index into the **outer page table**, and ***p2*** is the **displacement** within the **page** of the **outer page table**.

Two-Level Page-Table Scheme



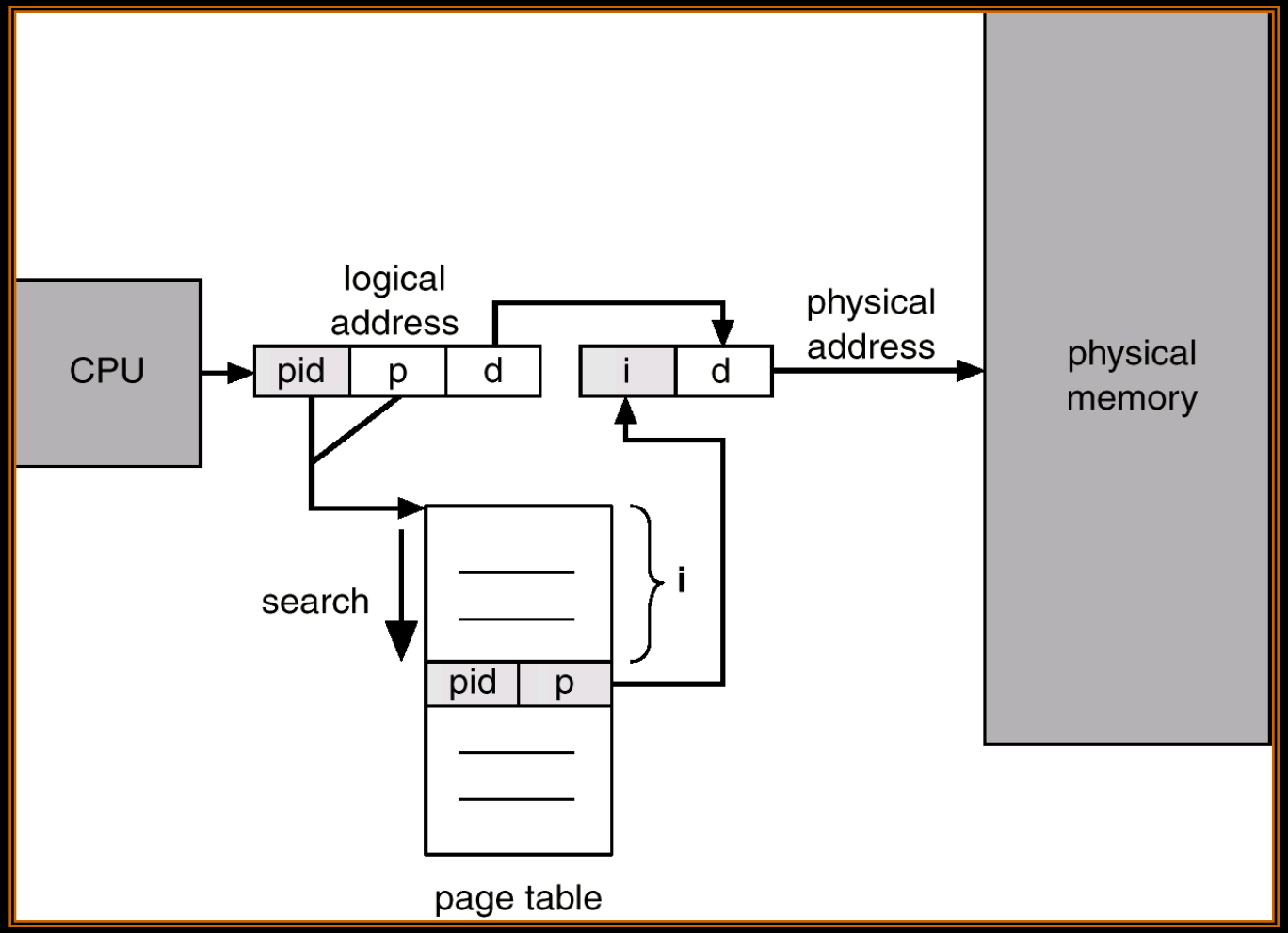
Address-Translation Scheme



Inverted Page Table

- 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.
- Essential when you need to do work on the page and must find out what process owns it.
- Use hash table to limit the search to one - or at most a few - page table entries

Inverted Page Table Architecture

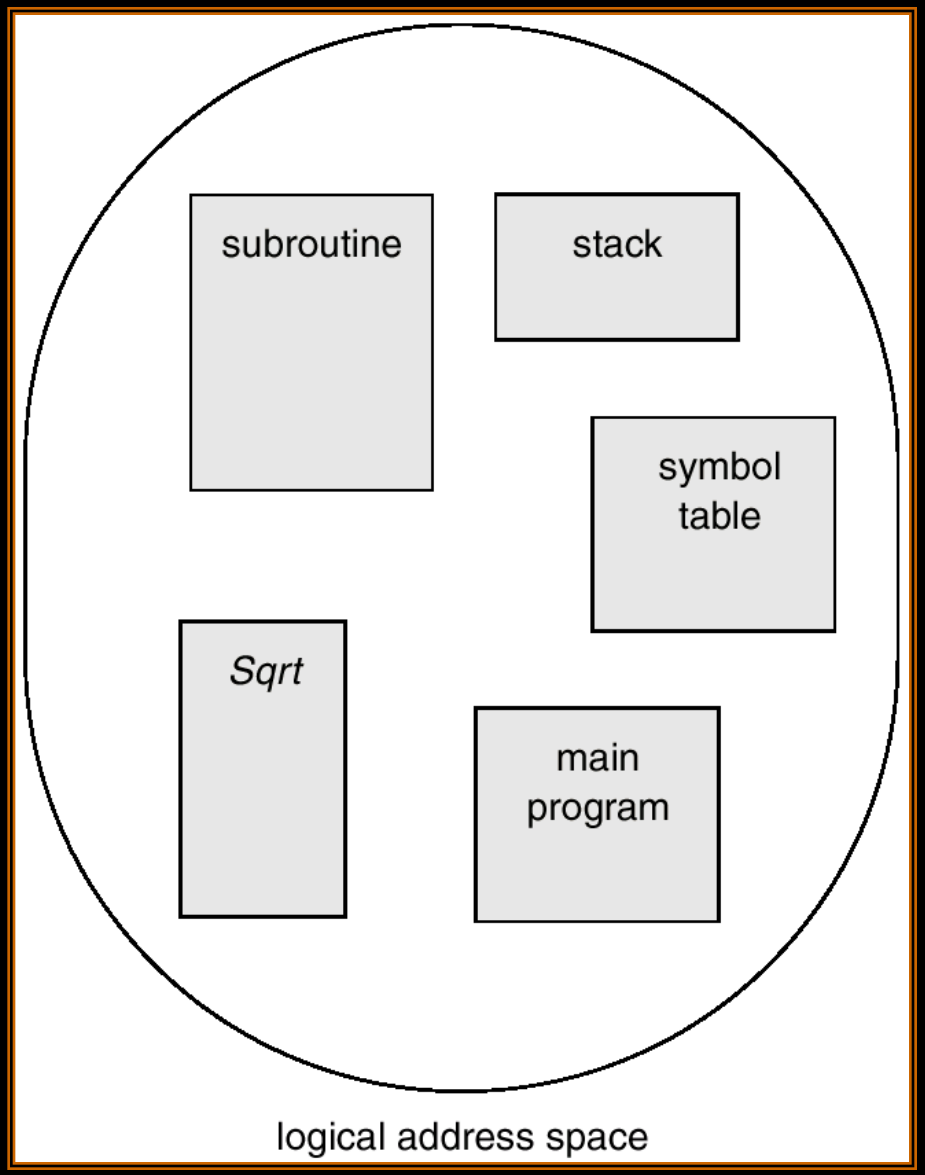


Segmentation

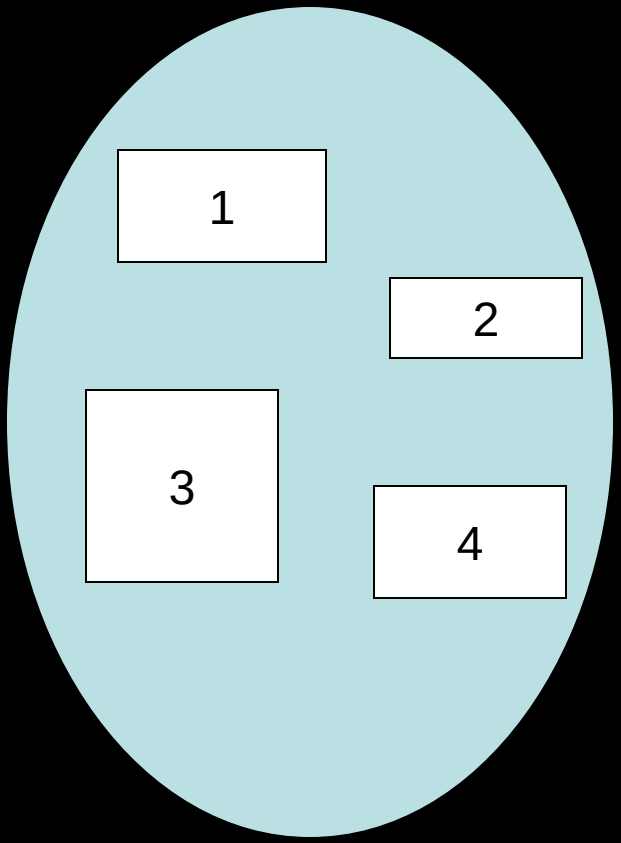
- A program is a collection of segments. A segment is a logical unit such as:

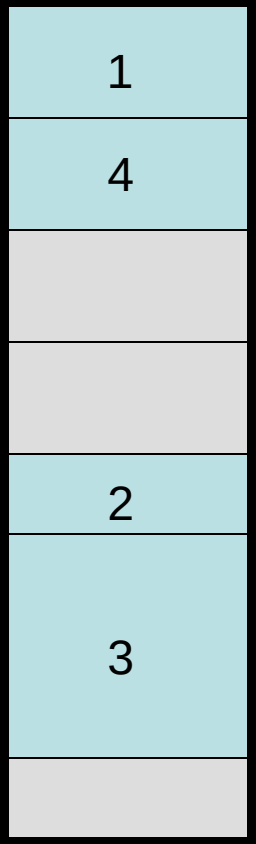
main program,
procedure,
function,
method,
object,
local variables, global variables,
common block,
stack,
symbol table, arrays

User's View of a Program



Logical View of Segmentation





Segmentation Architecture

- **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.**

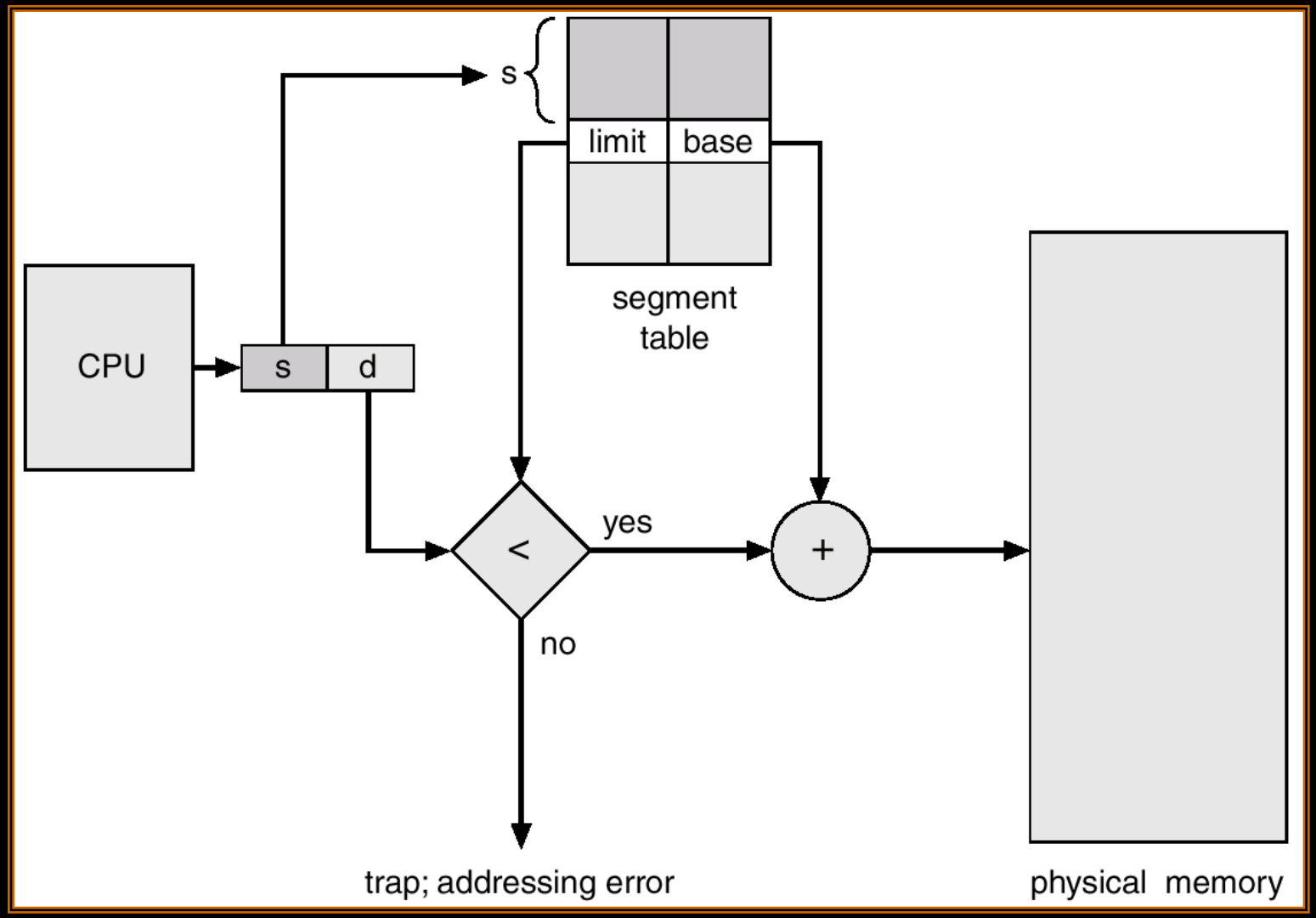
***limit* – specifies the length of the segment.**

❑ *Segment-table base register (STBR)* points to the segment table's location in memory.

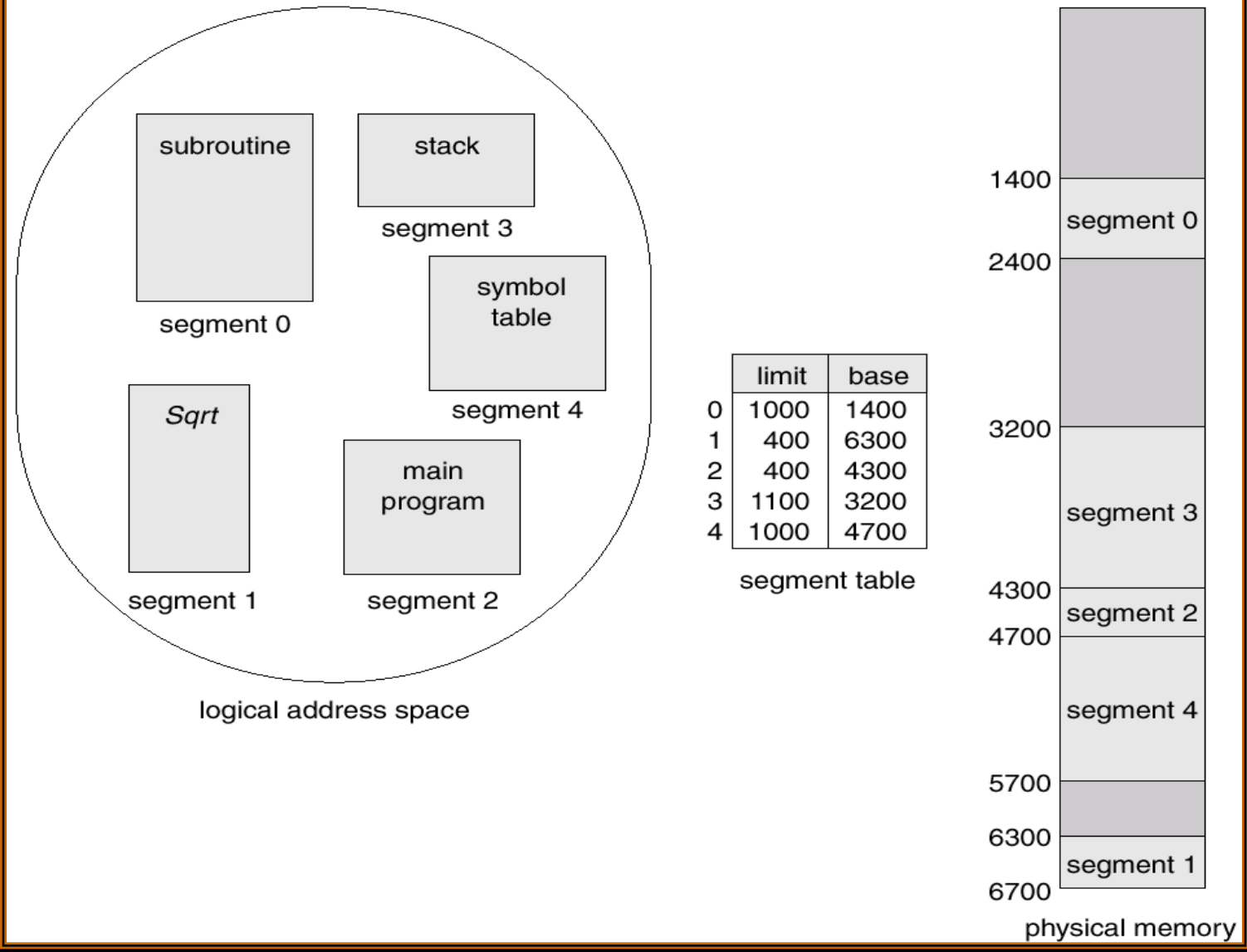
❑ *Segment-table length register (STLR)* indicates number of segments used by a program;

segment number *s*, is legal if *s* < STLR

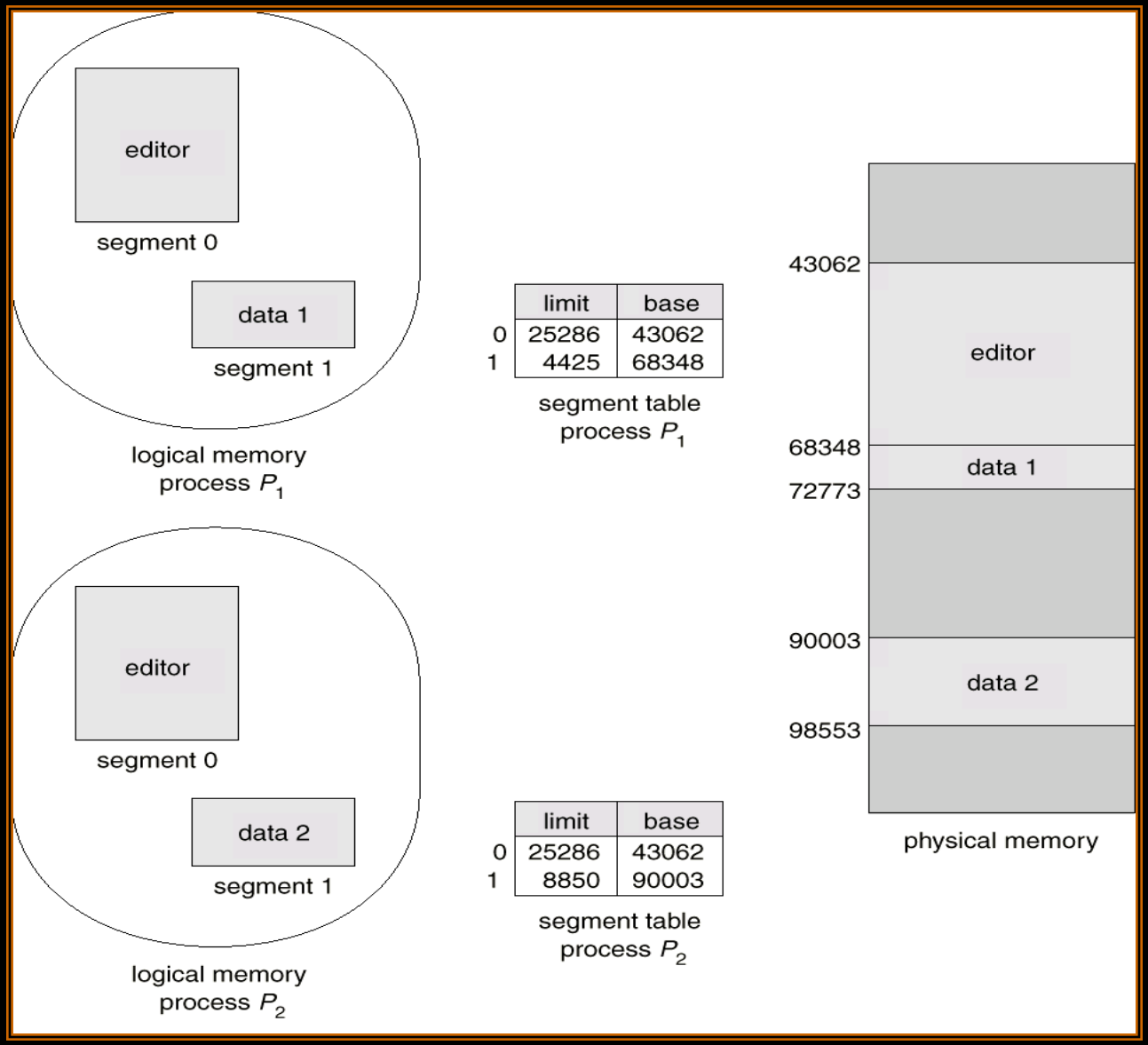
Segmentation Hardware



Example of Segmentation



Sharing of Segments



- Consider the following Segment Table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

- What are the physical addressed for the following logical addresses?

(a) 0,430
(b) 1,10
(c) 2,500
(d) 3,400
(e) 4,112

(a) 219 + 430 = 649

(b) 2300 + 10 = 2310

(c) illegal reference; traps to operating system

(d) 1327 + 400 = 1727

(e) illegal reference; traps to operating system