# *Operating Systems*

## CSE 231

## Instructor: Sambuddho Chakravarty

(Semester: Monsoon 2020)

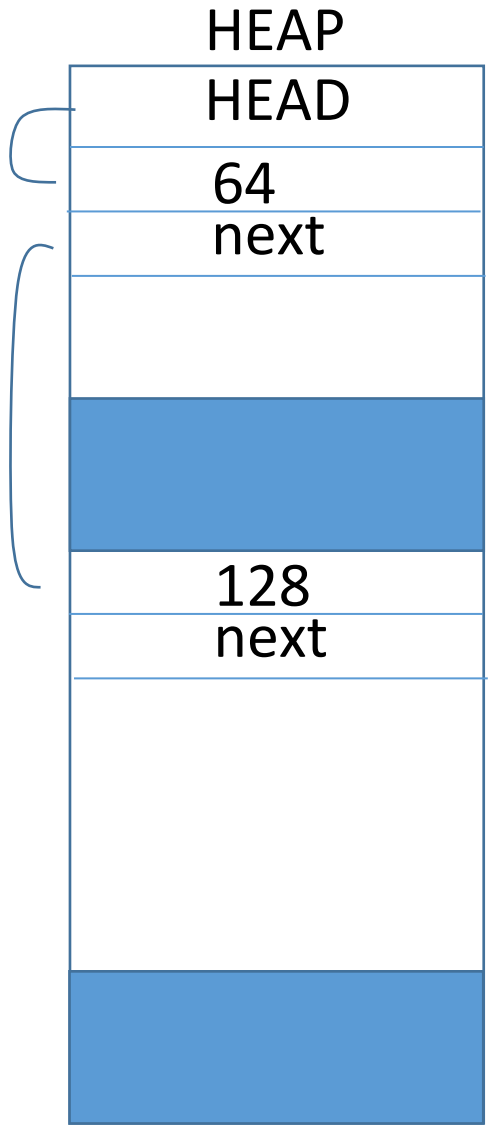
Week 8: Nov 9 – Nov 12

# C Style Memory Allocators

- **malloc:** This allocates a given number of bytes and returns a pointer to them. If there isn't enough memory available, it returns a null pointer.
- **free:** This takes a pointer to a segment of memory allocated by malloc, and returns it for later use by the program or the operating system
- Actually, some malloc implementations can only return memory back to the program, not to the operating system.

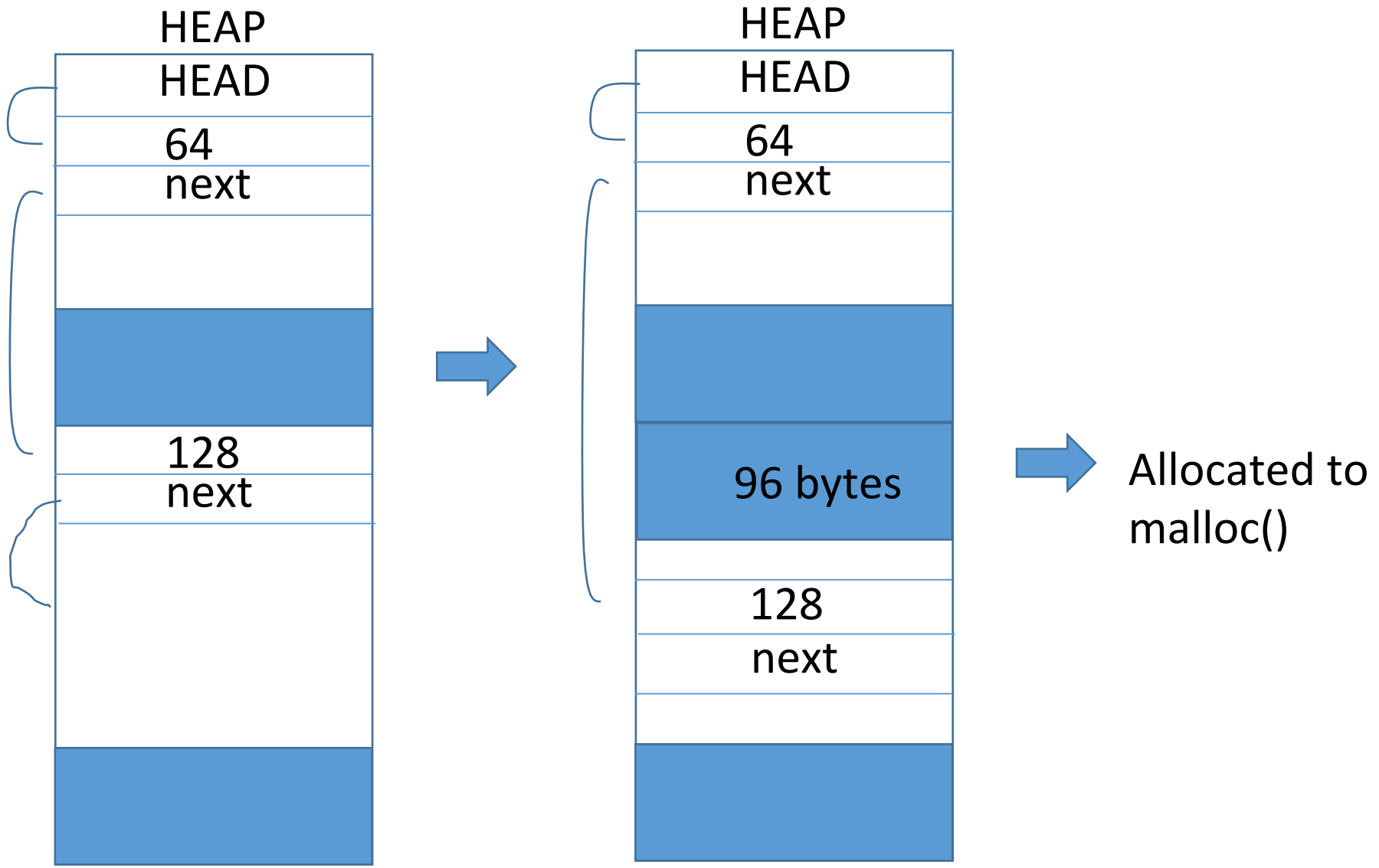
# How malloc() Works

- Traditional heap organization for malloc() and free().
  - Carves out a large chunk of heap.
  - Unallocated data arranged as free list (linked list).
  - Each chunk tagged with a ‘size’ prefix.



# How malloc() Works

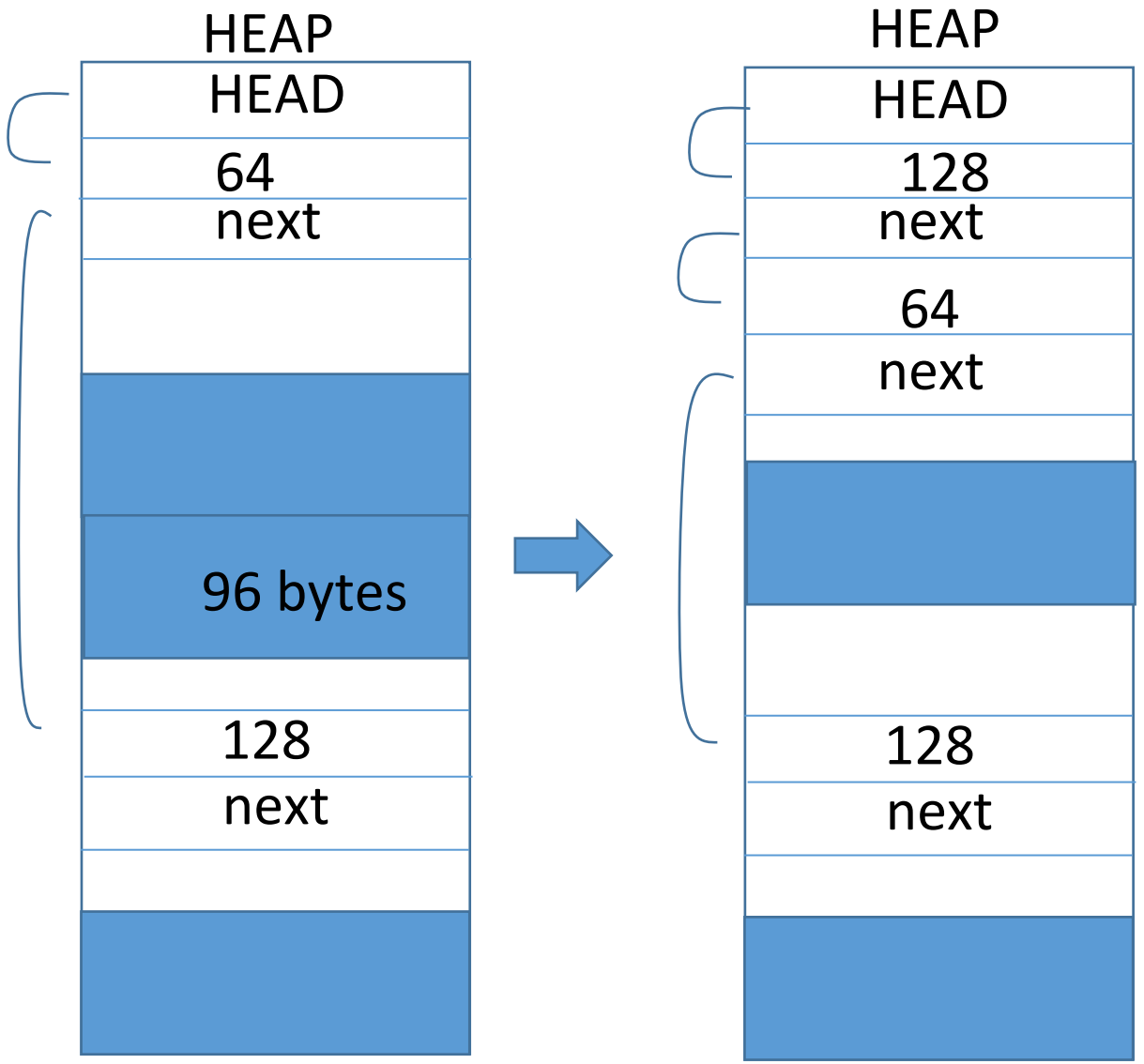
malloc(96)



# How malloc() Works

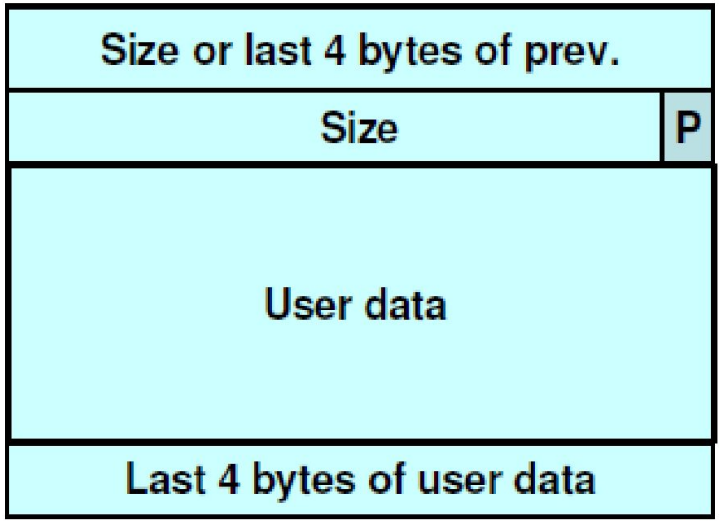
free(p)

- Chunk added back to free-list (at the head).



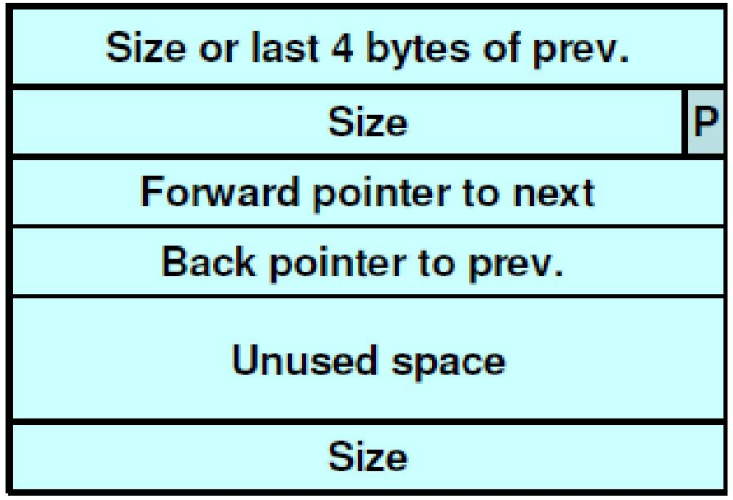
# Doug Lea's Memory Allocator (dlmalloc)

- Used in most modern GNU/Linux libc implementations.
- Free chunks organized into doubly linked lists (similar to traditional malloc).




**Allocated chunk**

The first four bytes of allocated chunks contain the last four bytes of user data of the previous chunk.



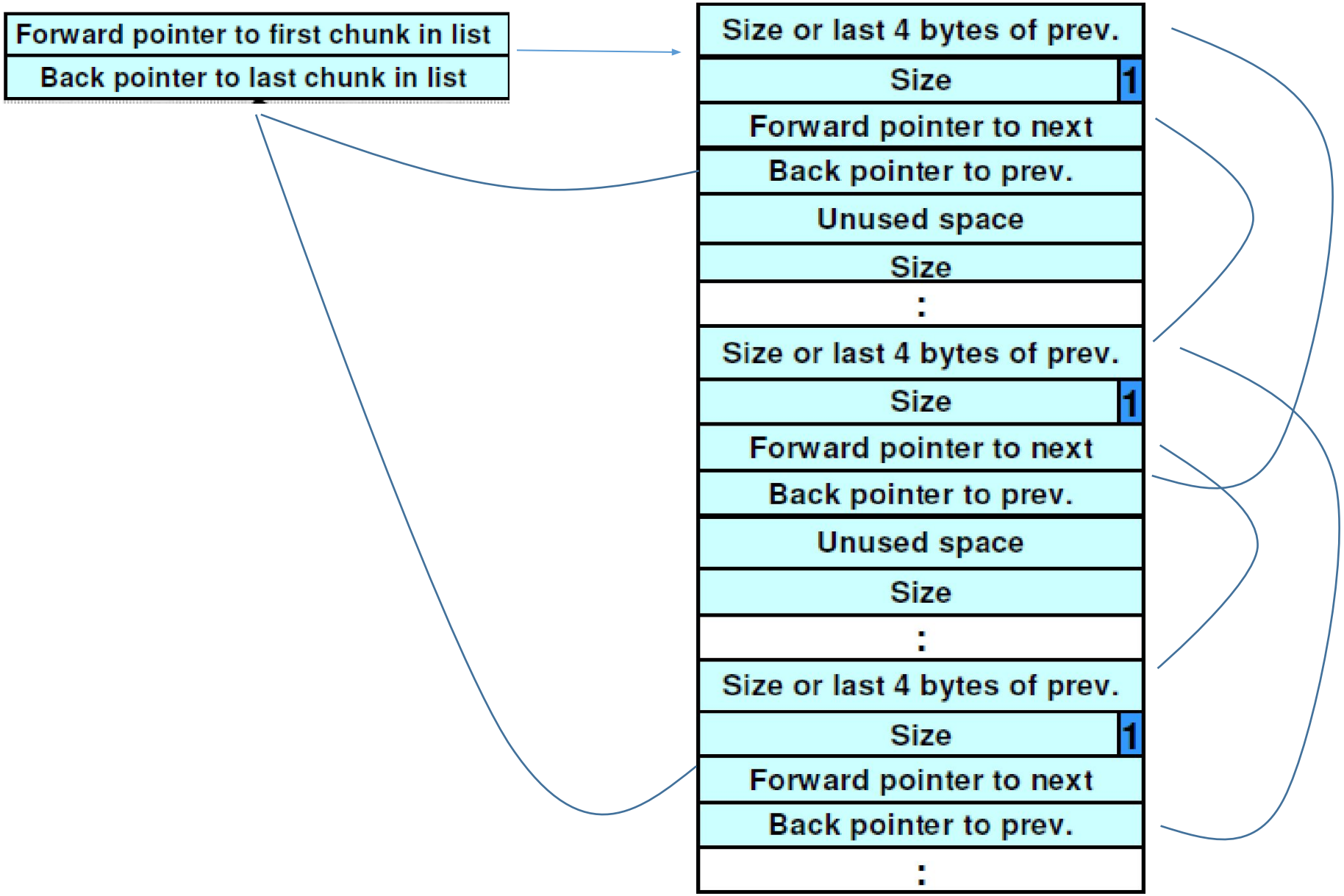

**Free chunk**

The first four bytes of free chunks contain the size of the previous chunk in the list.

# PREV_INUSE Bit:

- Indicates weather the previous bit is allocated or not.
- Chunks sizes are always 2-byte multiples – the LSB (PREV_INUSE) being 0.
- PREV_INUSE == 1 . Chunk allocated.

# Free List of dlmalloc



## Memory management fundamental :

The word “memory” □ virtual memory

□ physical main memory + swap space

□ local memory + shared memory

**The maximum amount of virtual memory that can be allocated is limited by two variable :**

- Result of physical hardware restriction
- Maximum address space permited by operating system

**The Operating System manages two type of memory :**

-Local memory

□ allocated precisely to one operating system process

-Shared memory

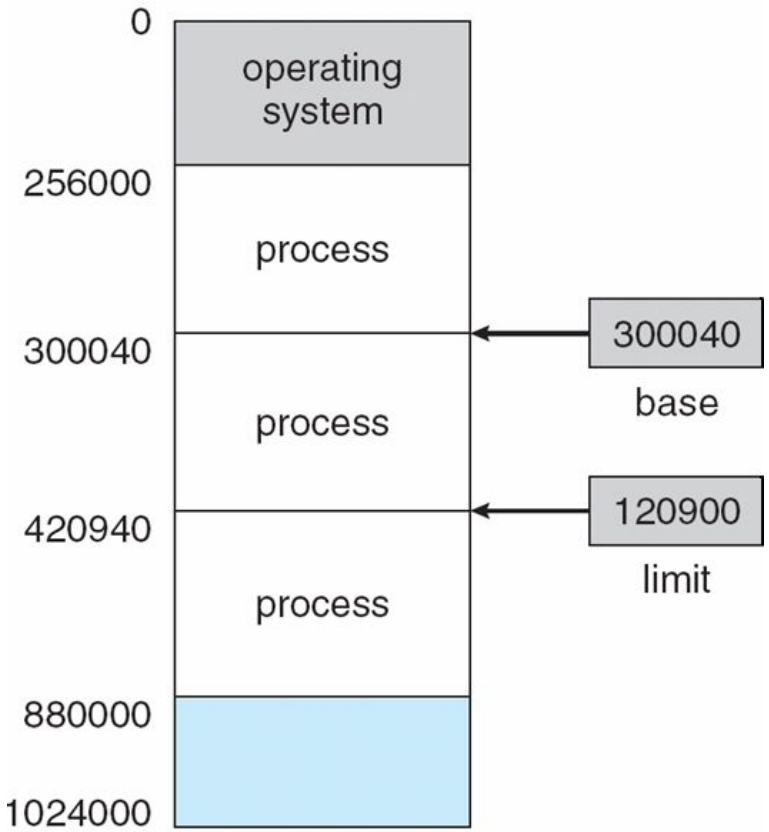
□ accessible to multiple operating system process

# Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation

# Base and Limit Registers

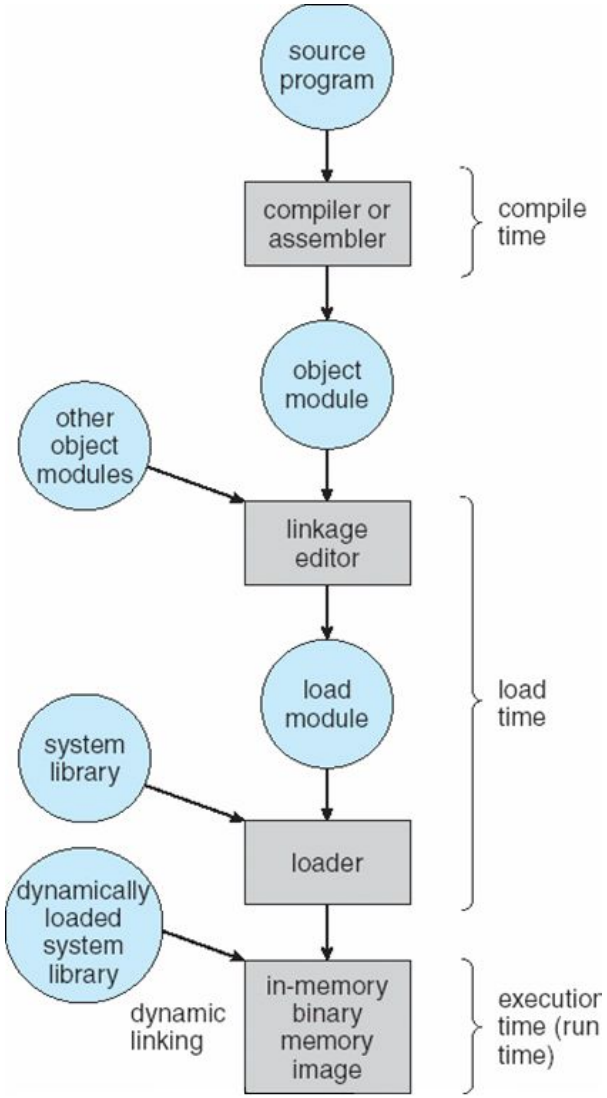
- A pair of **base** and **limit** registers define the logical address space



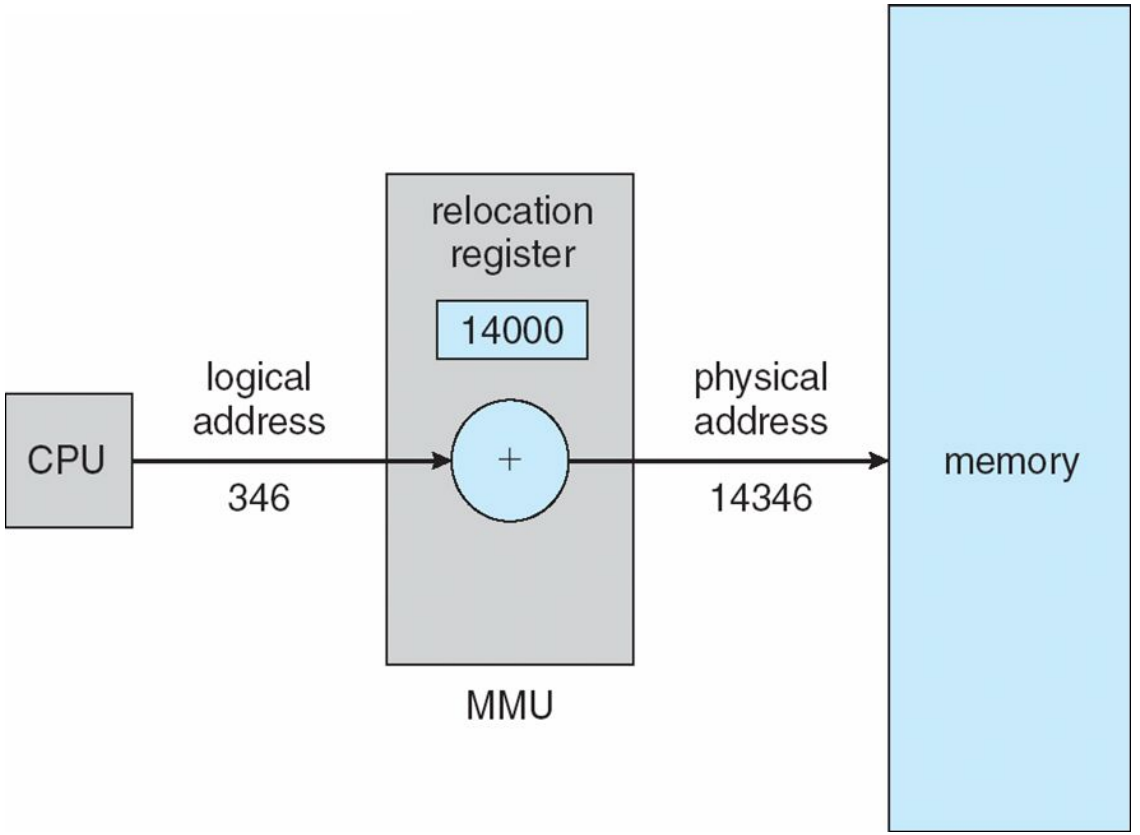
# Computational Time

- **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)

# Multistep Processing of a User Program



# Logical Address vs Physical Address



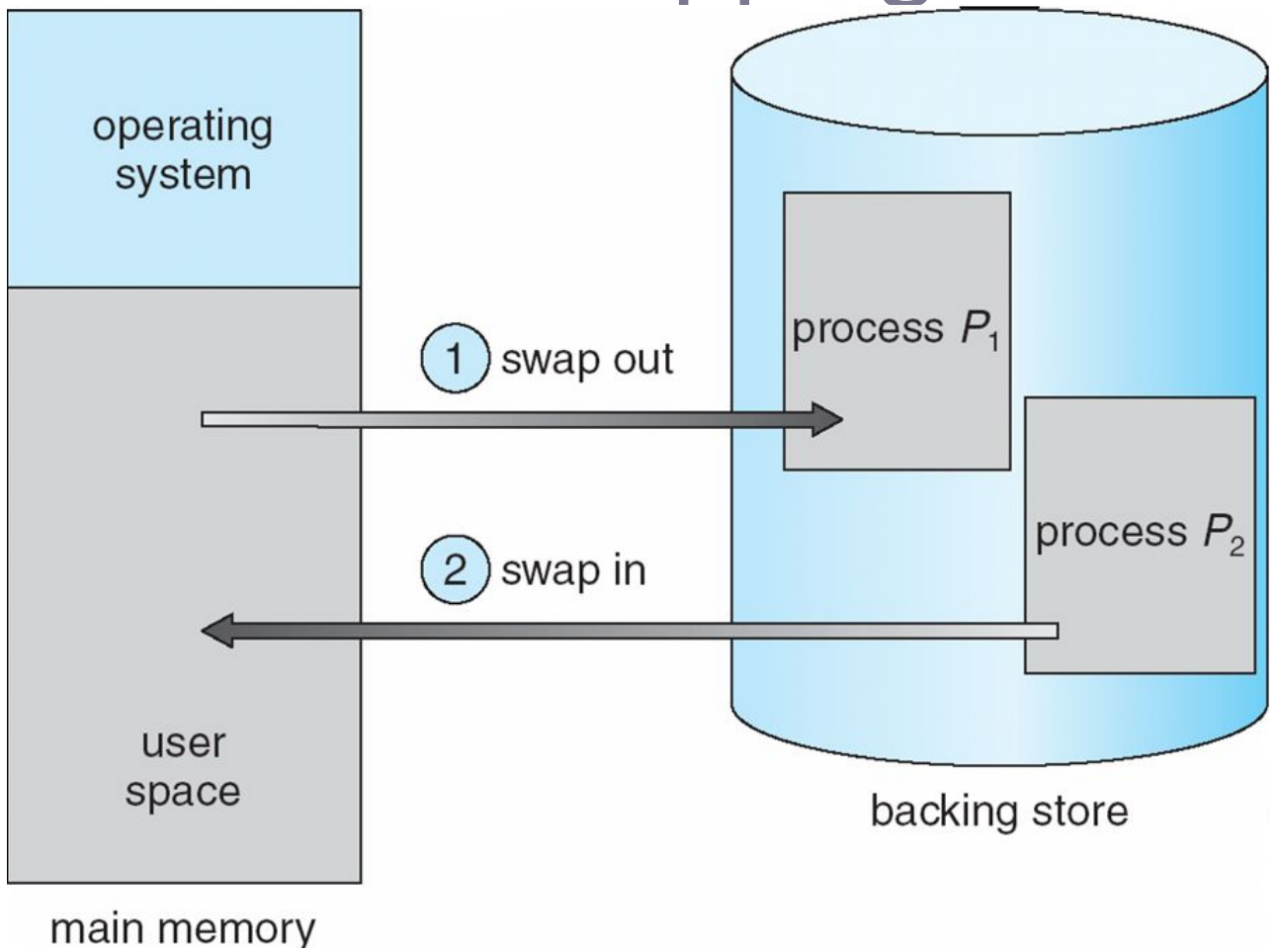
# Techniques

- Swapping
- Contiguous Memory Allocation
- Paging
- Segmentation

# Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

# Schematic View of Swapping

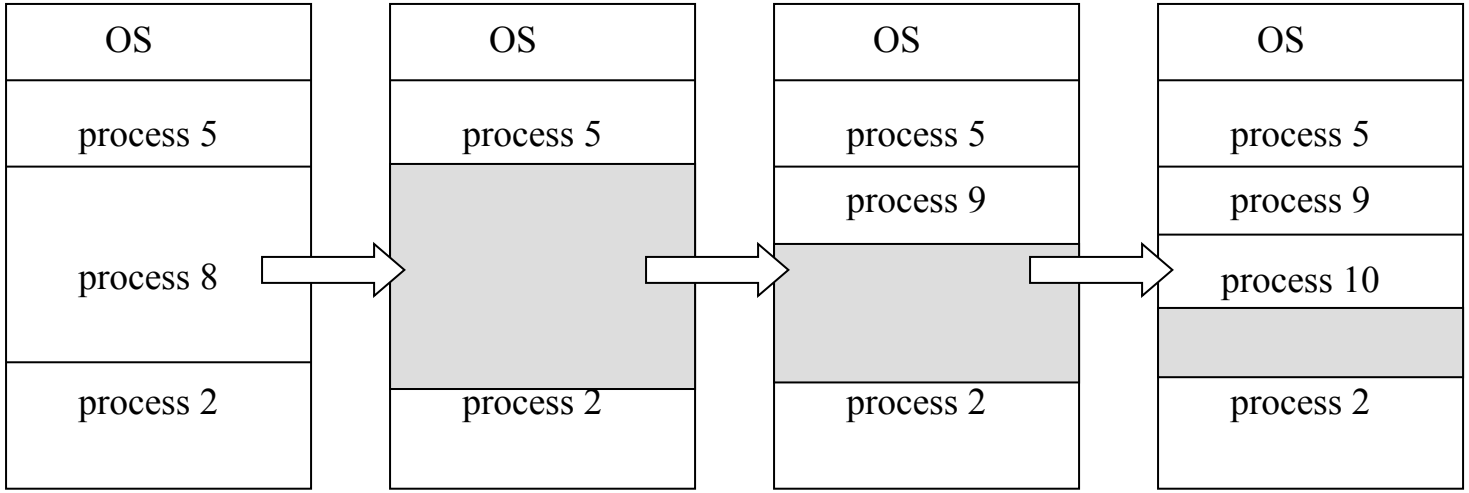


# Contiguous Allocation

- Main memory must accommodate both OS and user processes and we need to allocate main memory in the most efficient manner.
- Main memory usually into two parts:
  - Resident operating system
  - User processes

# Contiguous Allocation (Cont)

- 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)



# Dynamic Storage-Allocation Problem

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

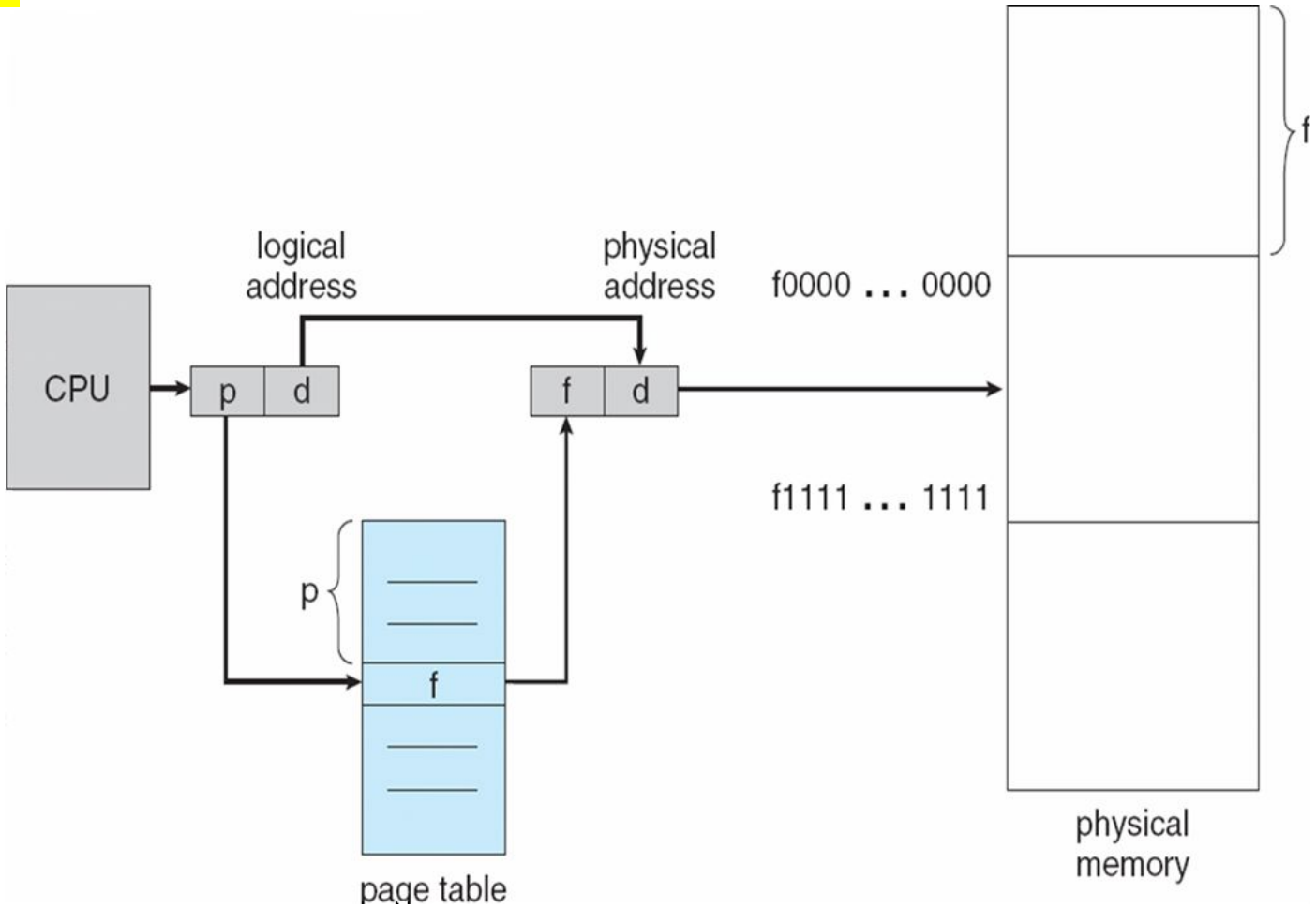
# Fragmentation

- **Fist-fit and Best-fit causes 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 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

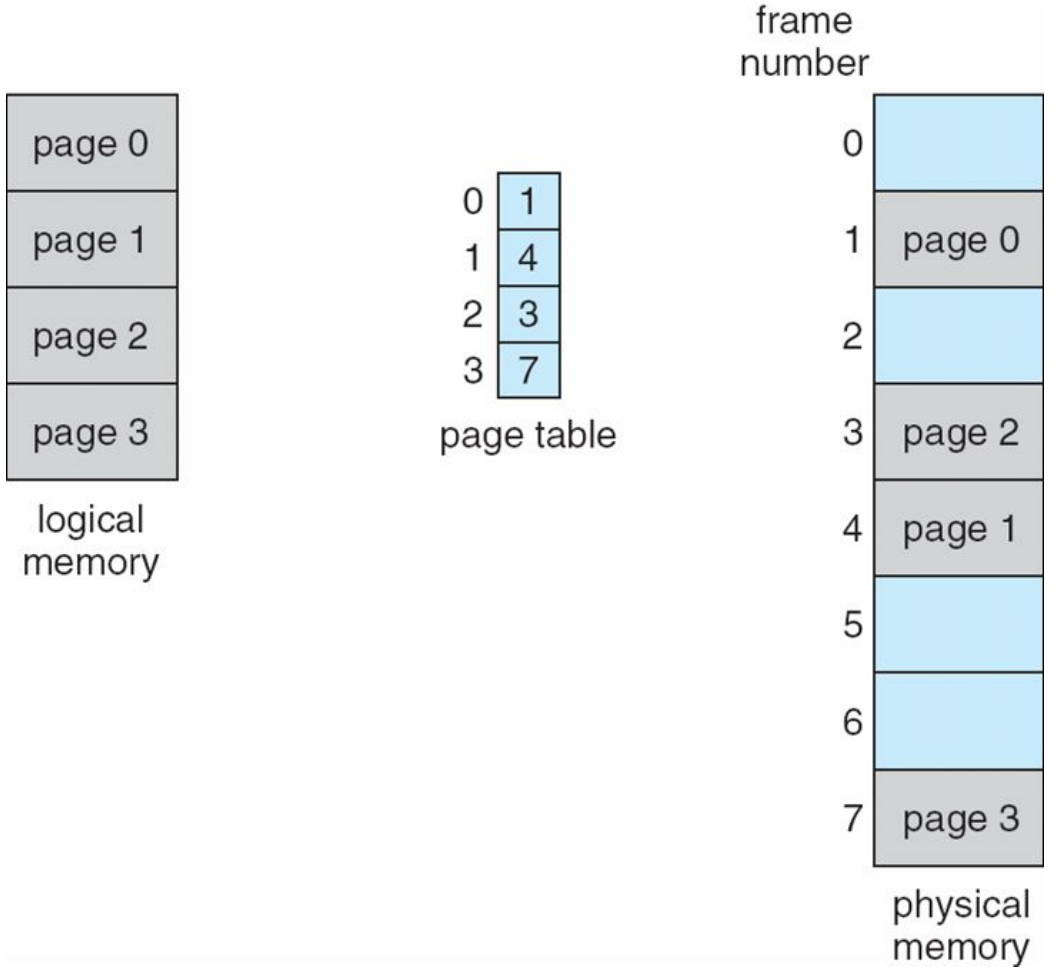
# Paging

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 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

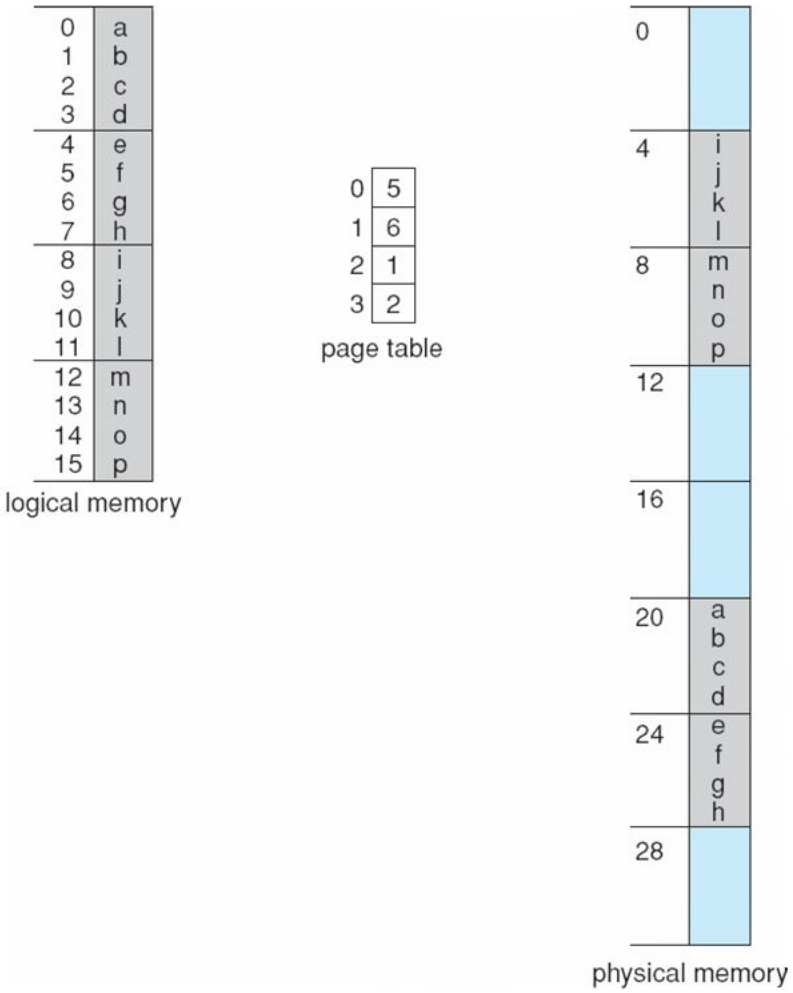
# Paging Hardware



# Paging Model of Logical and Physical Memory

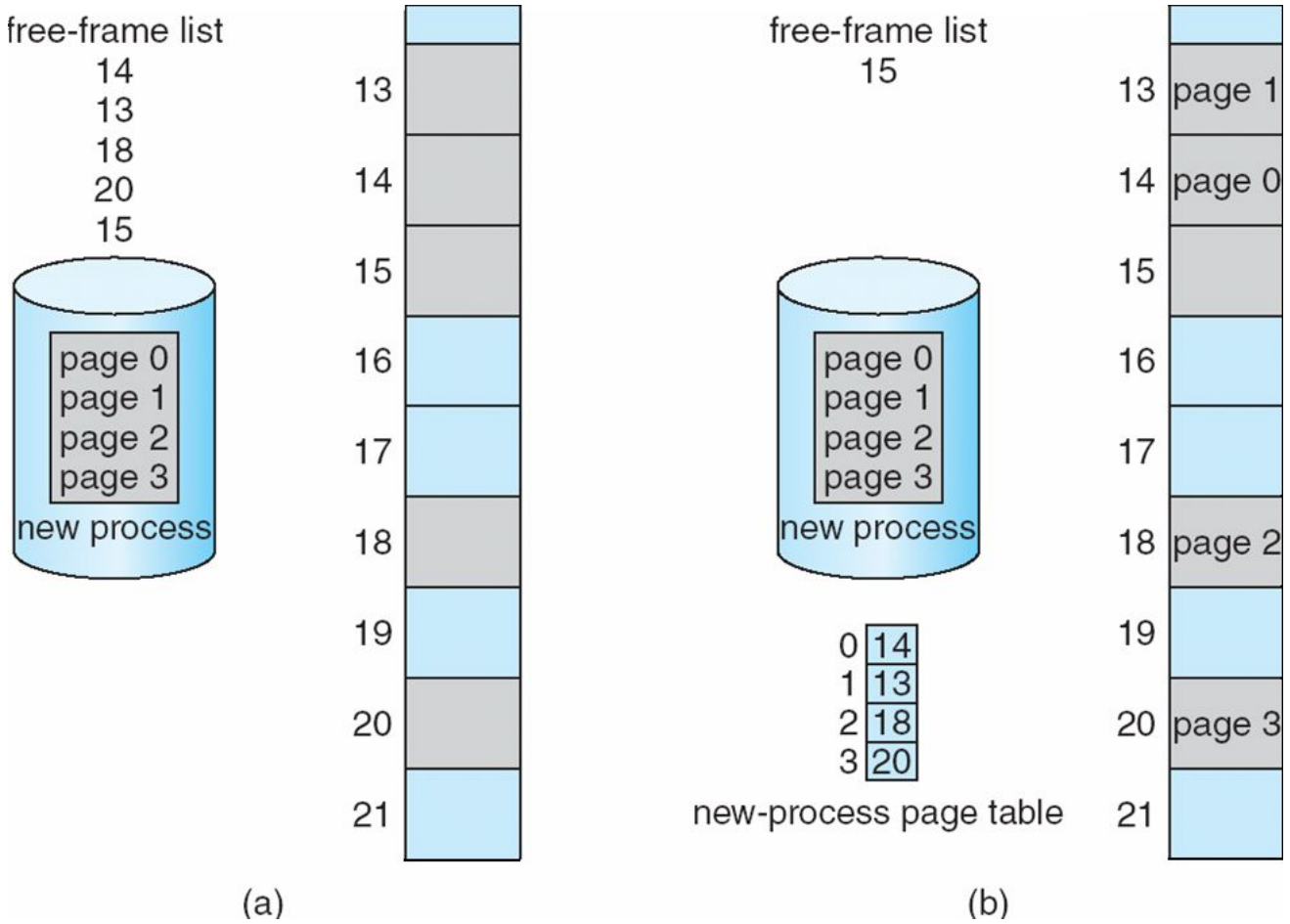


# Paging Example




32-byte memory and 4-byte pages

# Free Frames




Before allocation

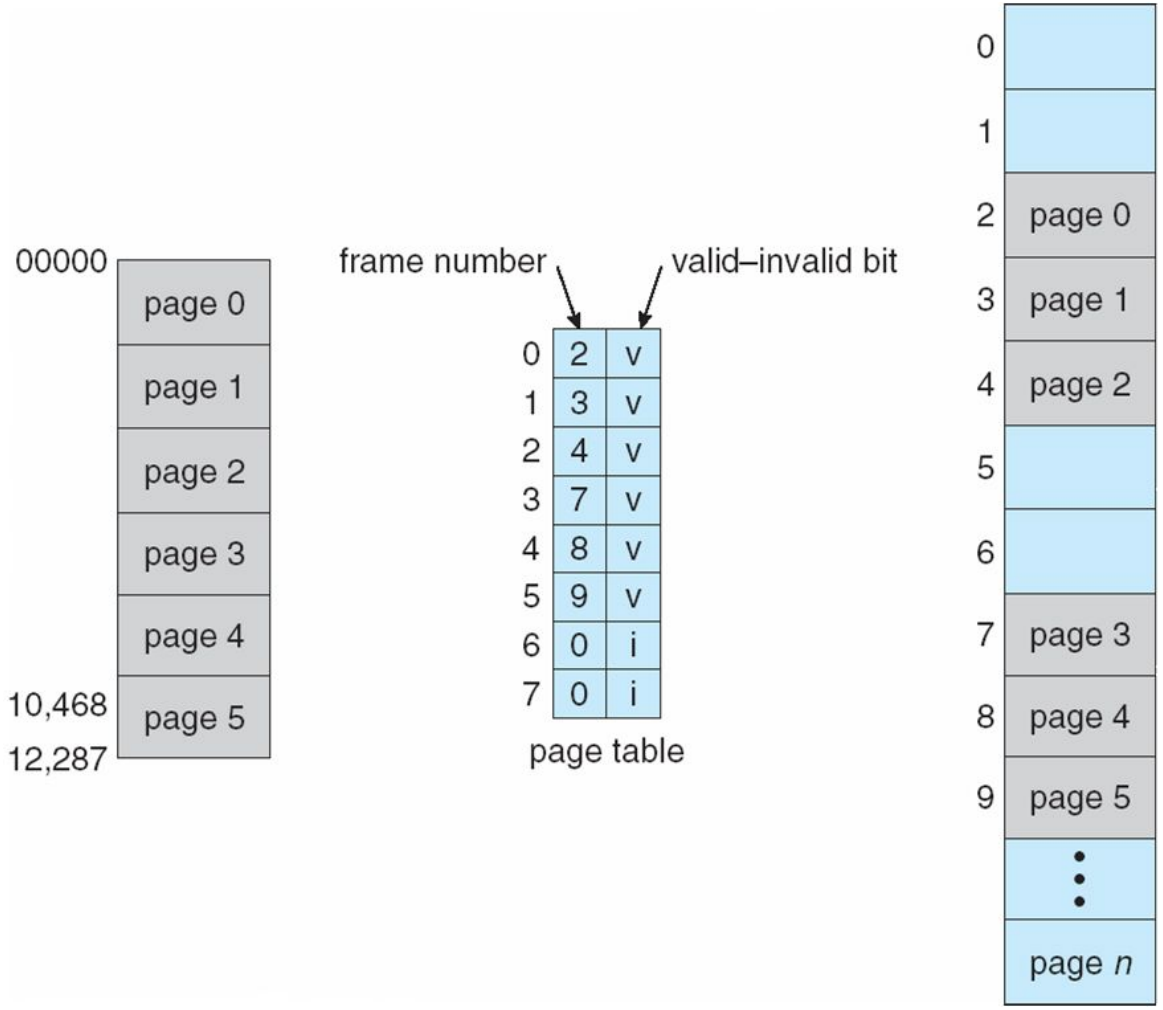
After allocation

# 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’
  - “invalid” indicates that the page is not in the process’

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

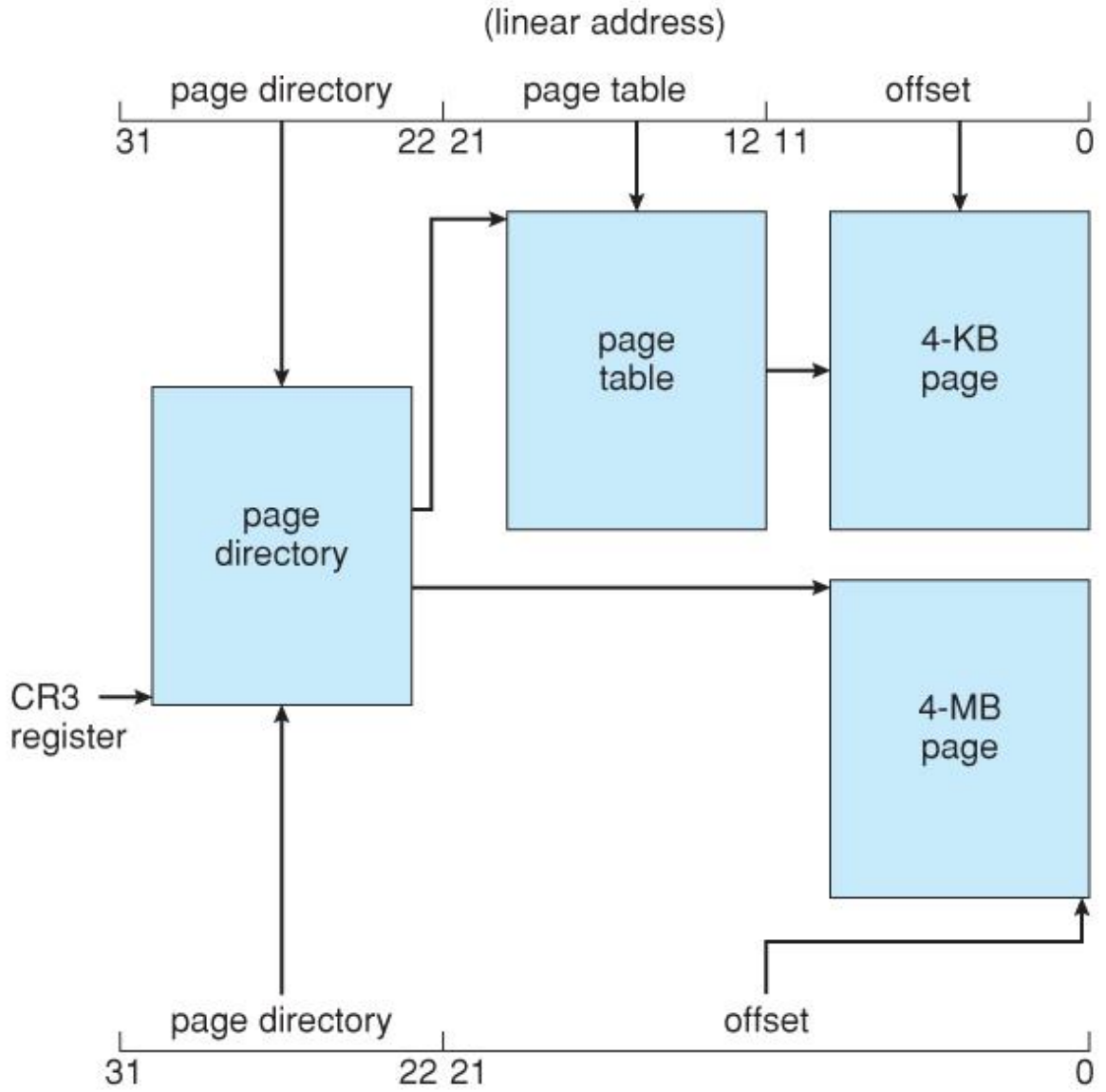


# Structure of the Page Table

- Hierarchical Paging: Multi-level paging, used in most architectures
- Hashed Page Tables: Instead of offsets, a hash function is calculated to determine the offset in the page table.
- Inverted Page Tables: A single page table which is used for all processes, and stores the pid for each process along with other information.

# Intel Paging Architecture (x86)

- 32-Bit addresses.
- Supports 4 KB or 4 MB pages.
- Multi-level paging support.



# Segmentation

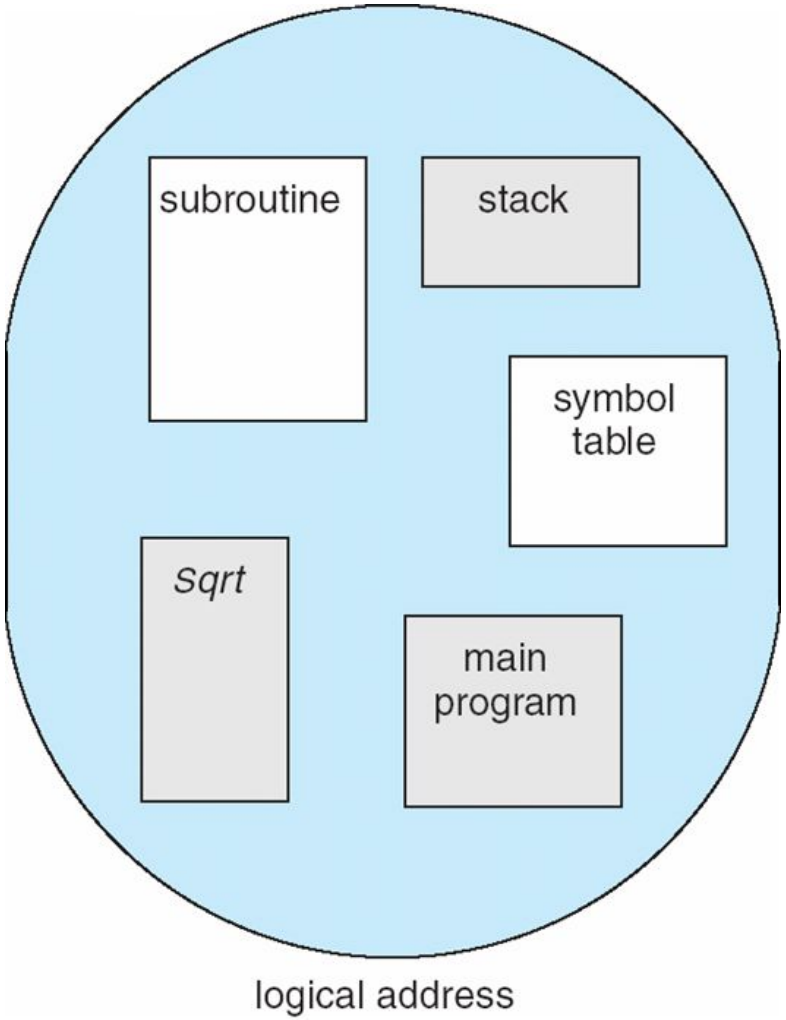
- Memory-management scheme that supports user view of memory

- A program is a collection of segments

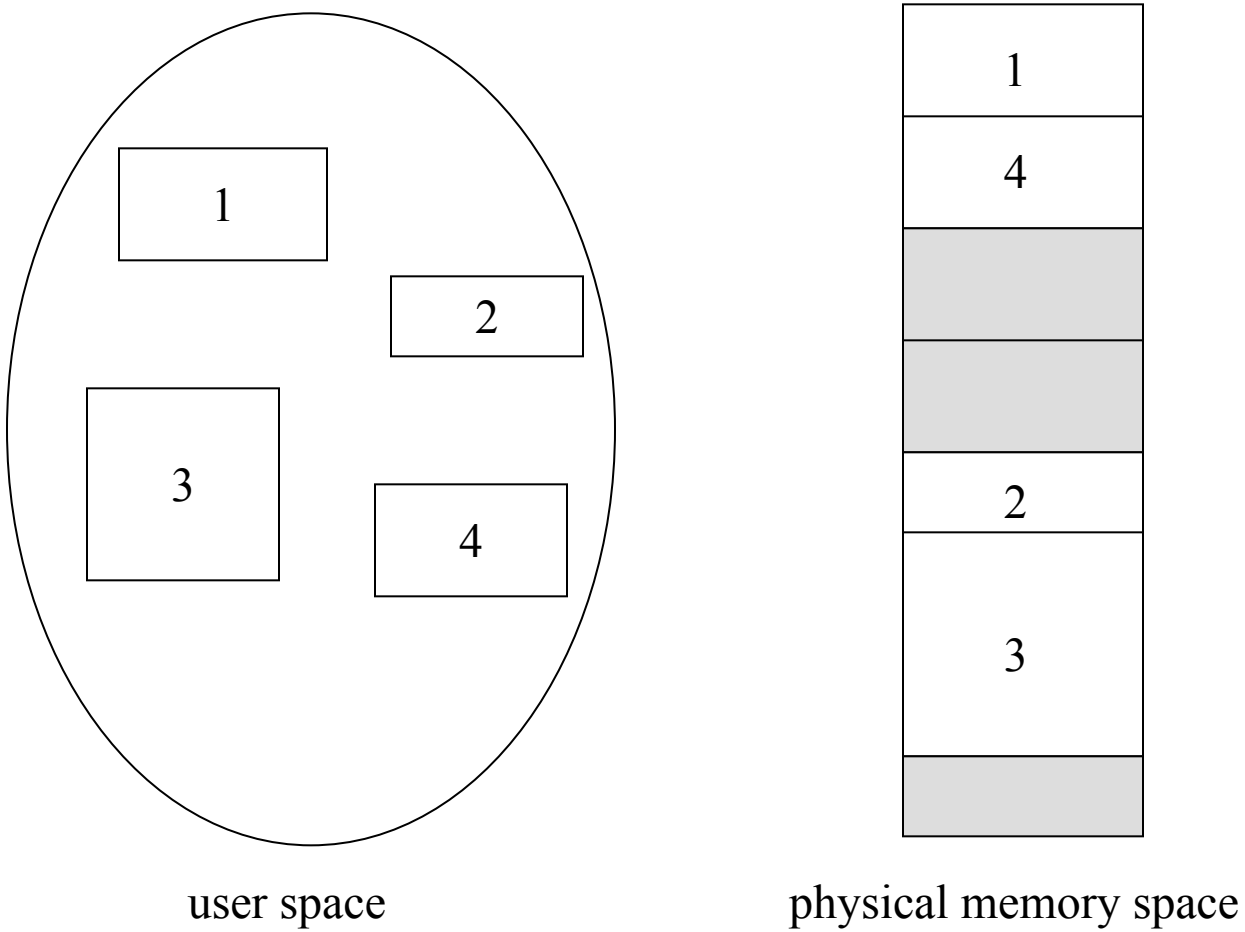
  A segment is a logical unit such as:

  Code, stack, data, heap, BSS, RODATA

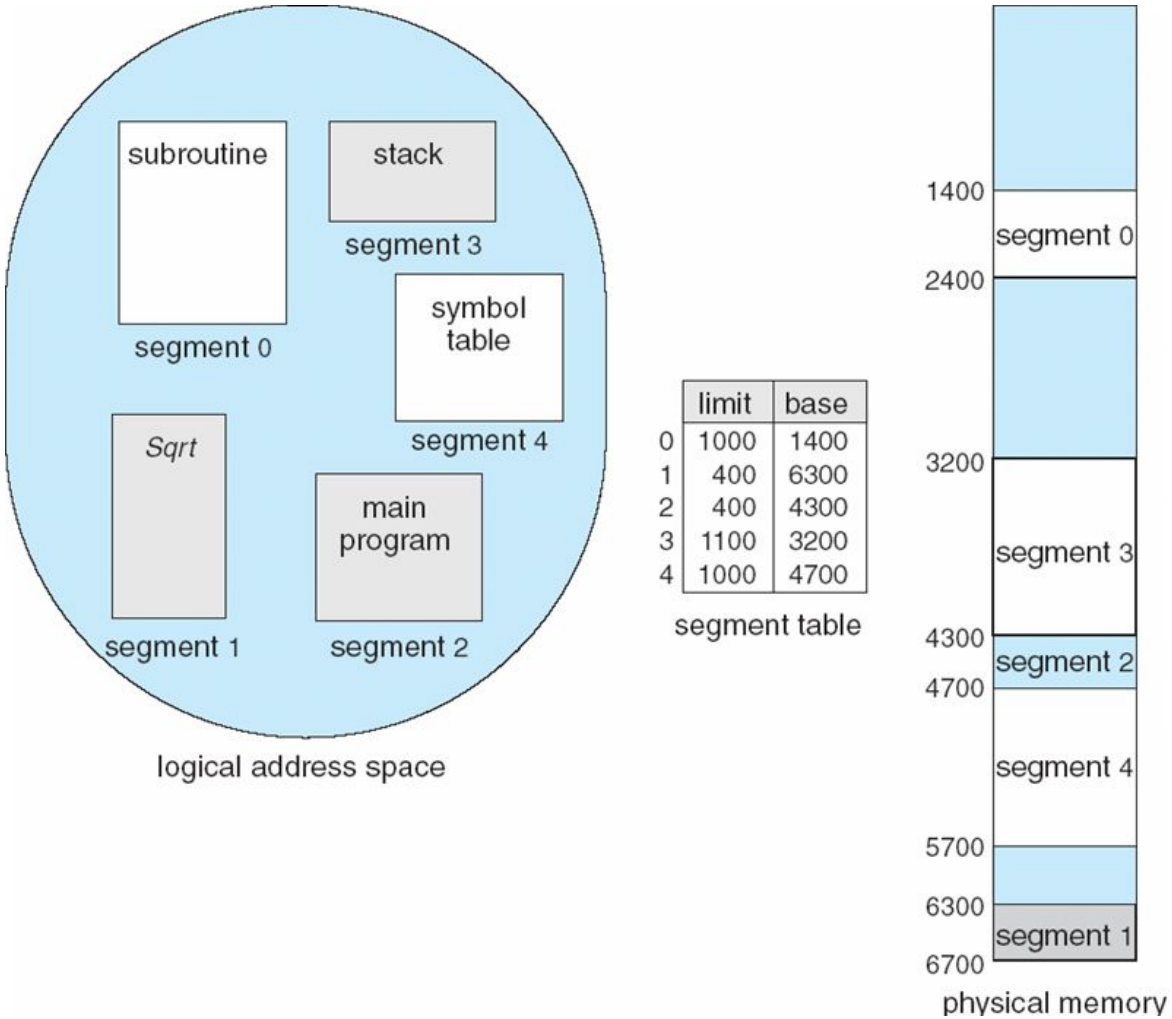
# User's View of a Program



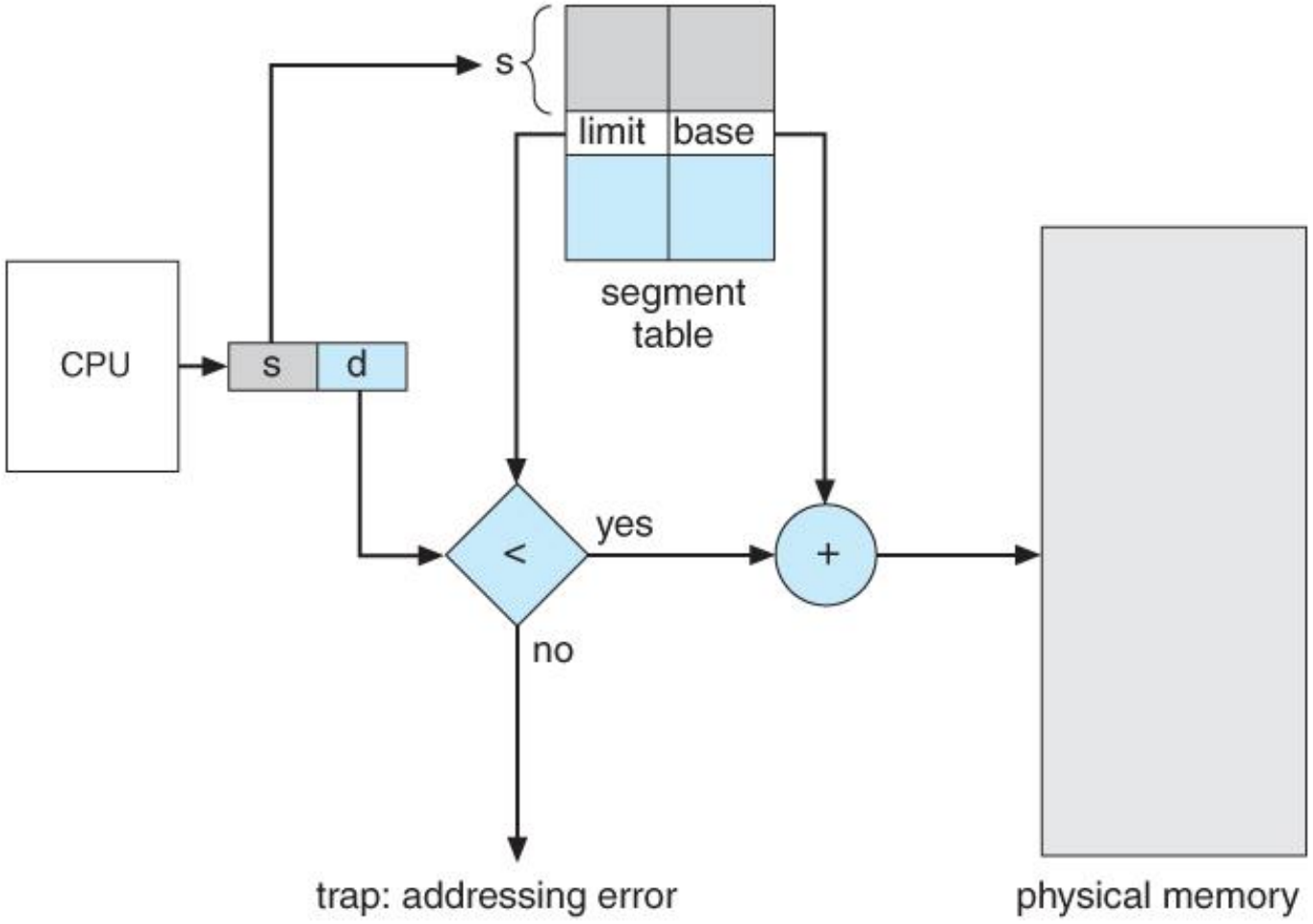
# Logical View of Segmentation



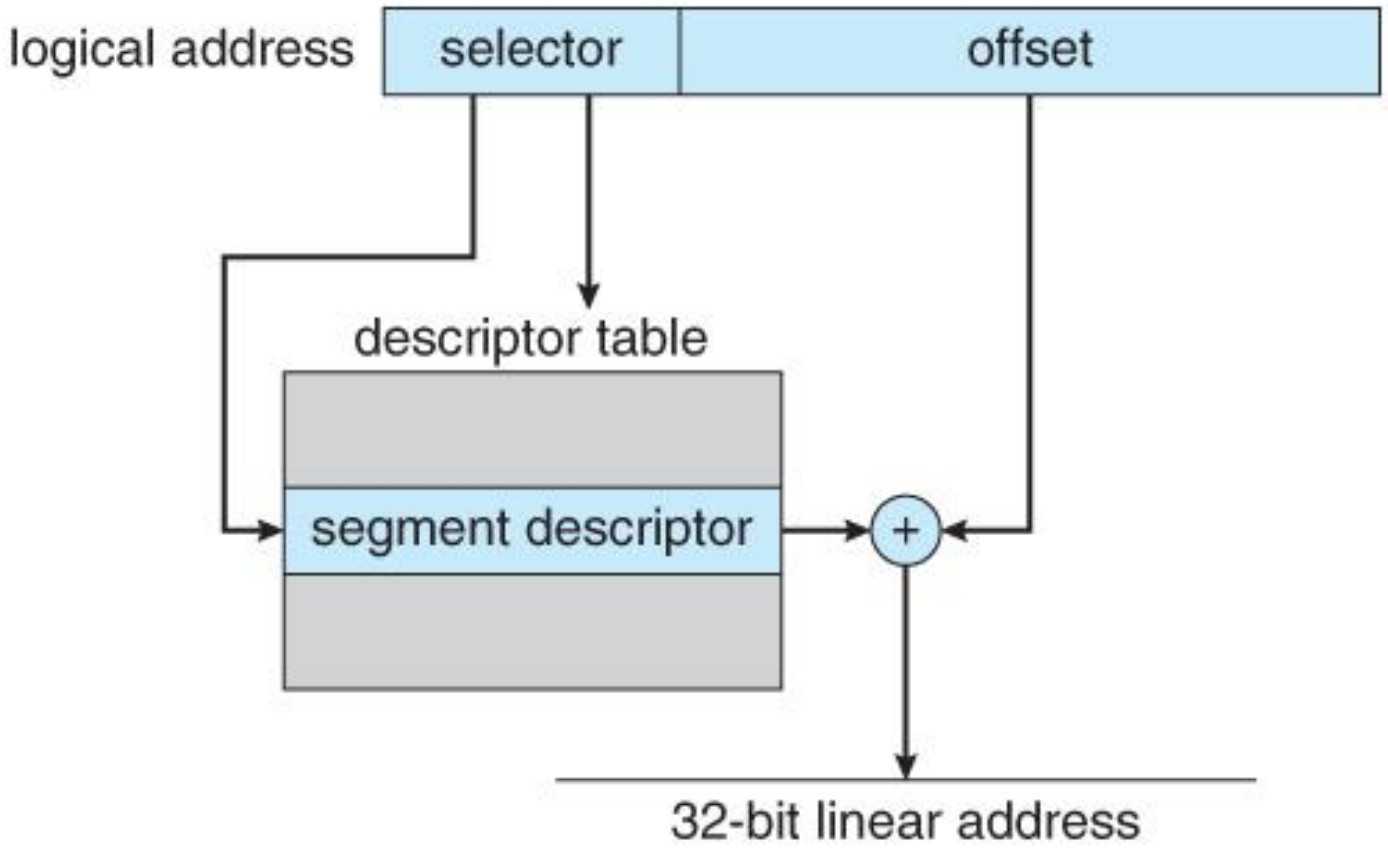
# Example of Segmentation



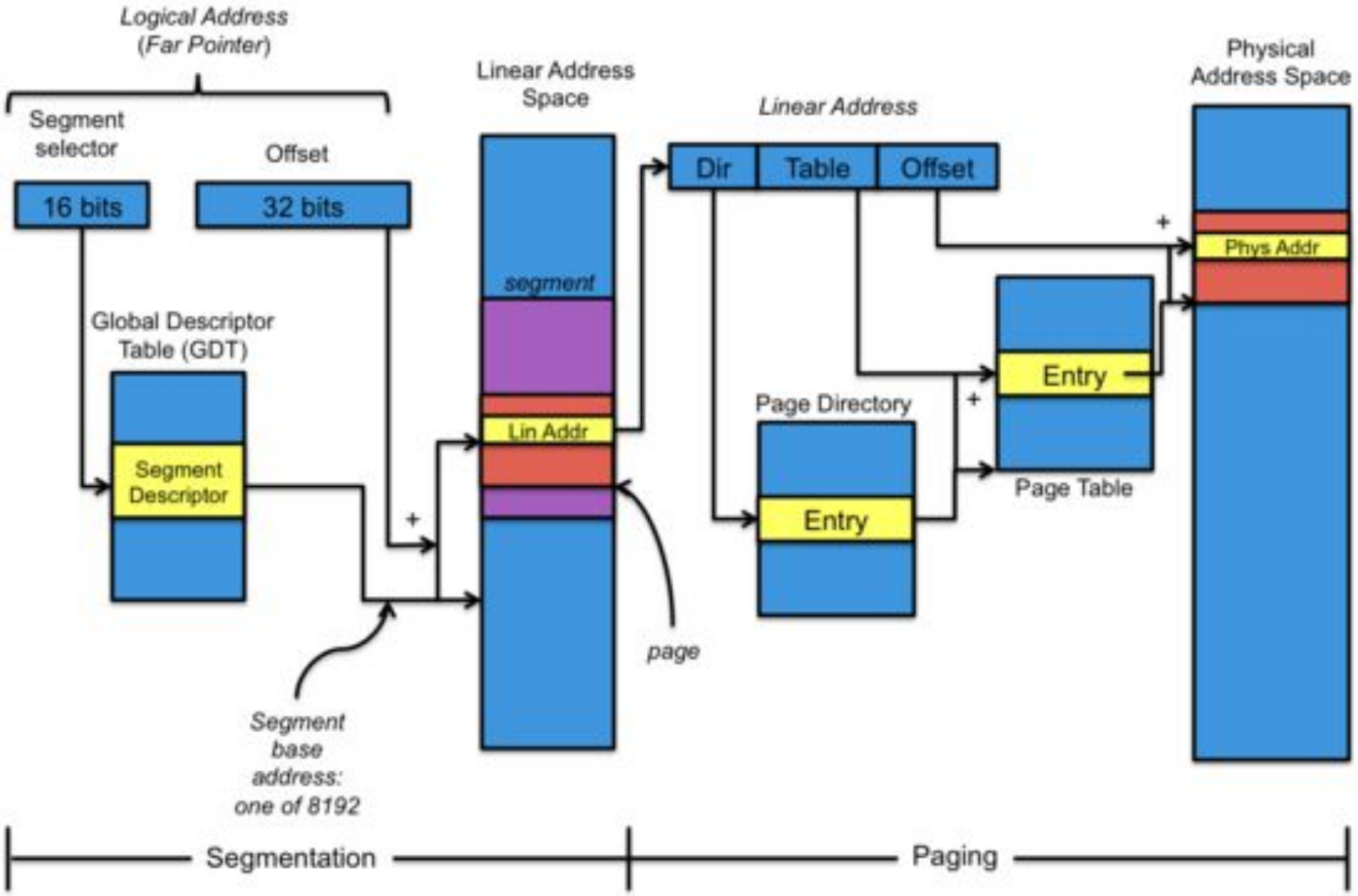
# Segmentation Hardware



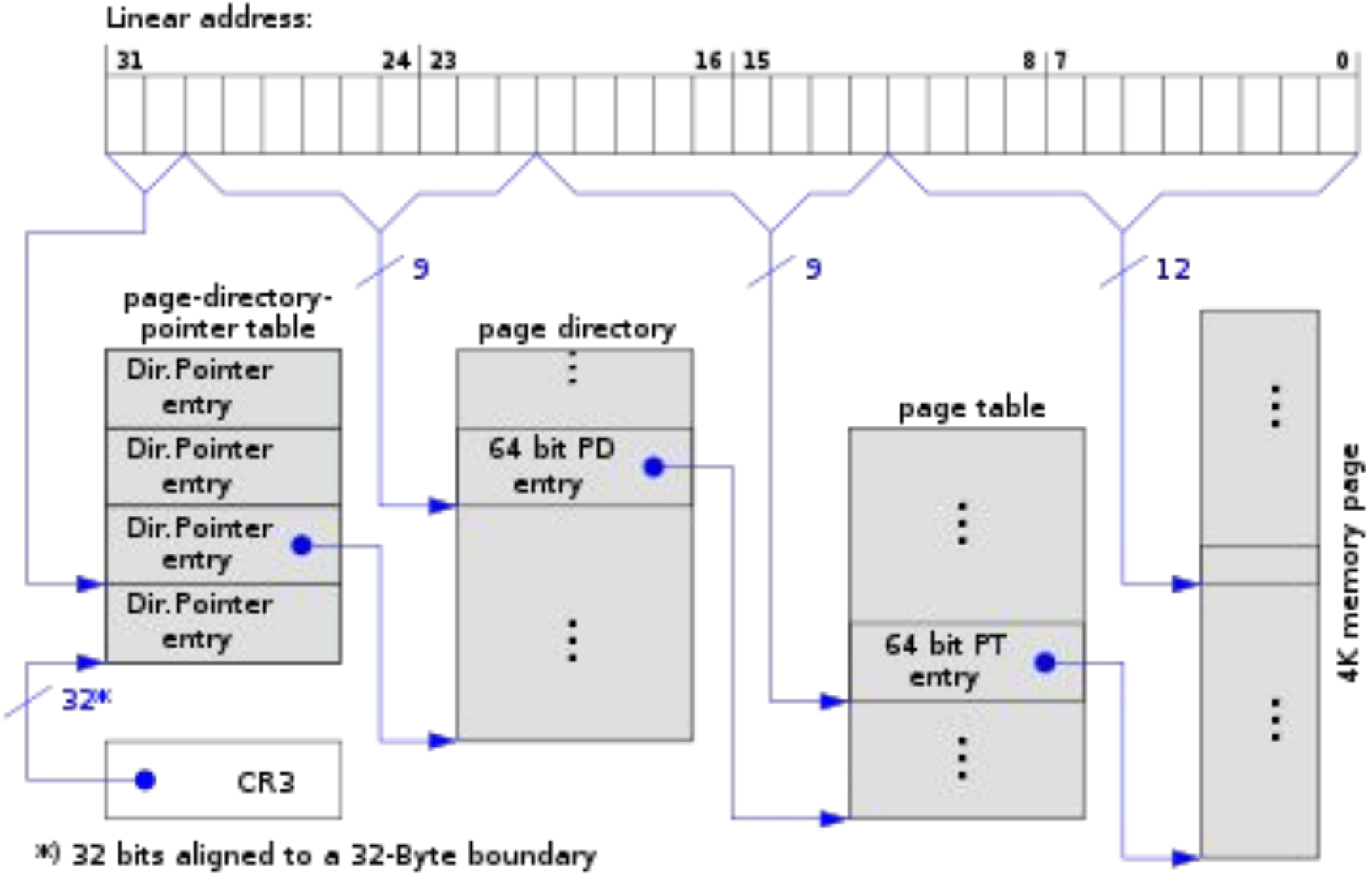
# Segmentation Hardware – Intel x86



# Segmentation + Paging – Intel x86

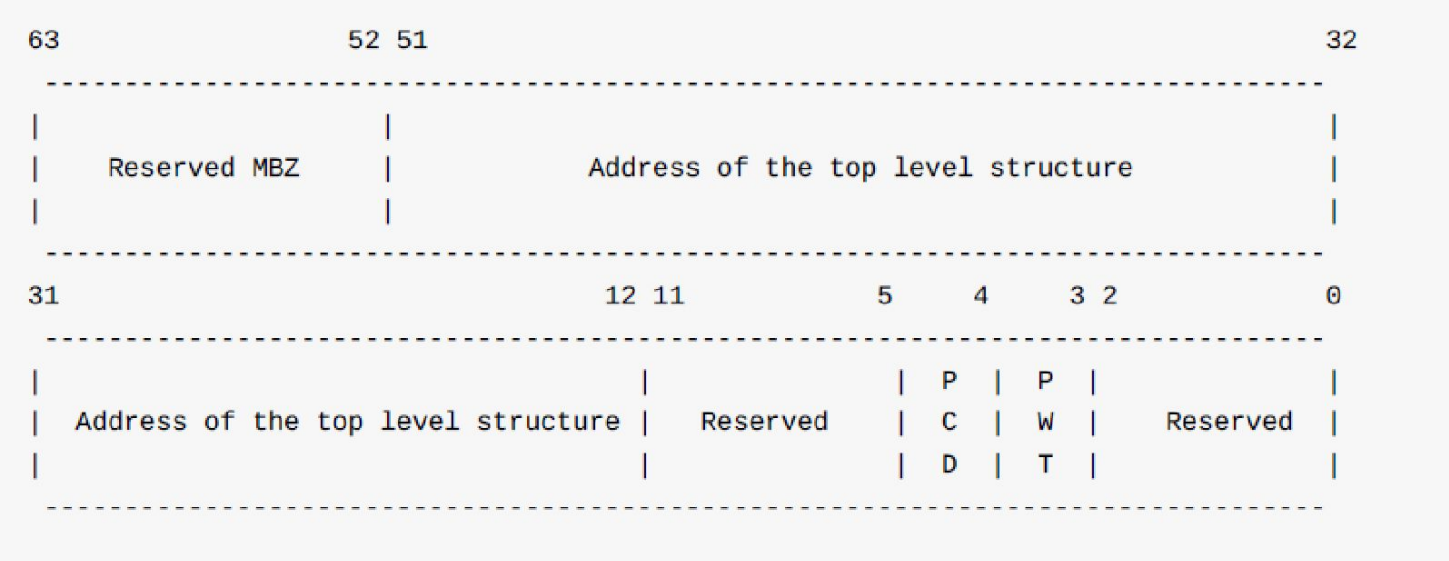


# Intel IA-32 Page Enable Extension (supports over 4 GB pages)

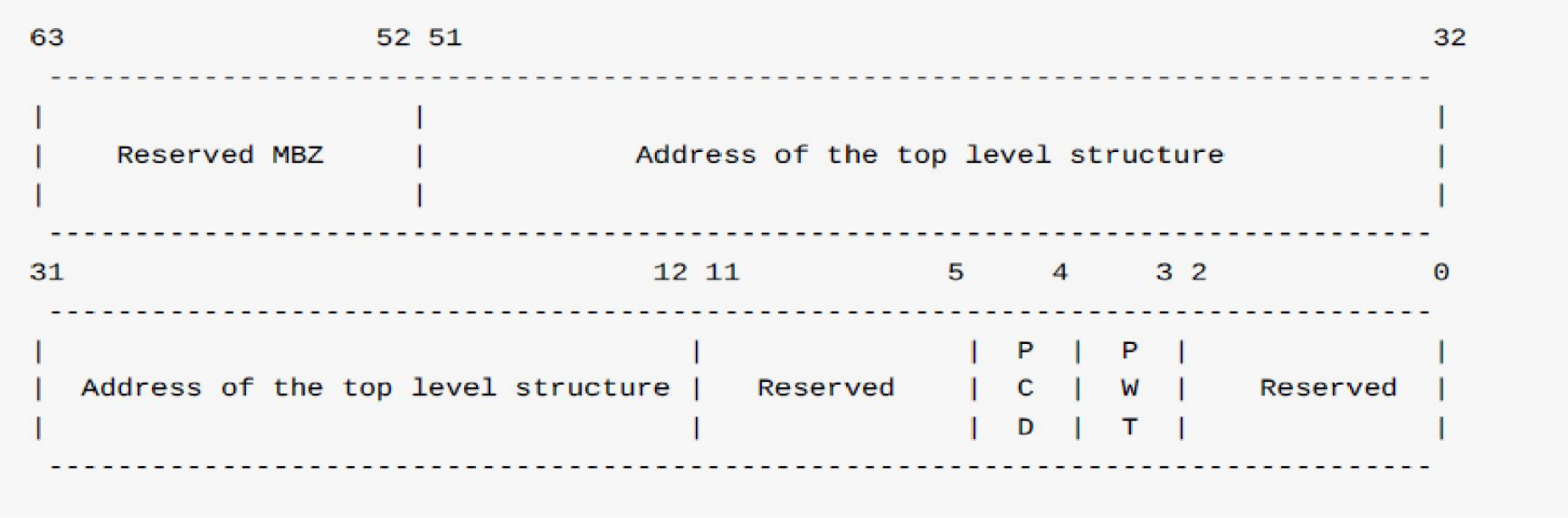


# Paging in x86_64

- 4 level of paging.
- Here cr3 is used to store the address of the top-level structure, the PML4 or Page Global Directory as it is called in the Linux kernel. cr3 is 64-bit register and has the following structure:



# Paging in x86_64




Bits 63:52 - reserved must be 0.
Bits 51:12 - stores the address of the top level paging structure;
Reserved - reserved must be 0;
Bits 4 : 3 - PWT or Page-Level Writethrough and PCD or Page-level cache disable
indicate. These bits control the way the page or Page Table is handled by the hardware
cache;
Bits 2 : 0 - ignored

# Linear Address Translation