COP 4600
Operating Systems Design

Spring 2019

Chapter 8: Main Memory

Objectives

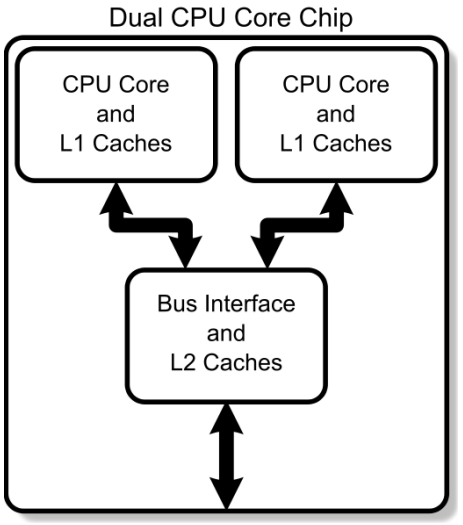
- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation

Background (1)

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage that CPU can access directly
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles
  - **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Background (2): Multi-core Computing

- Single-core processors have stopped historic performance scaling because of the constraints:
  - Power consumption
  - Memory access latency
- Multi-core is dominating
- A multi-core CPU combines two or more independent cores into a single package.
  - Share L2 caches, memory, bus, etc.



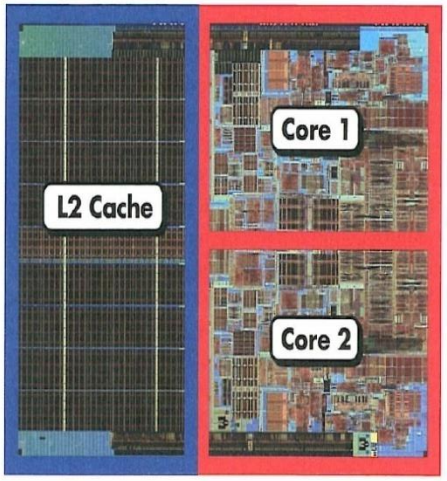
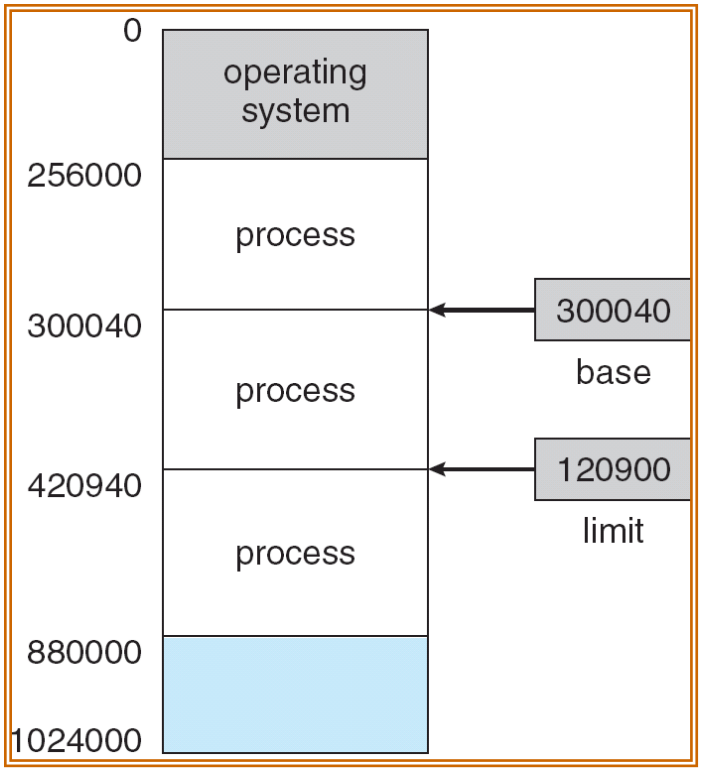


Figure 12-9: The floor plan of an Intel Core Duo processor

Base and Limit Registers

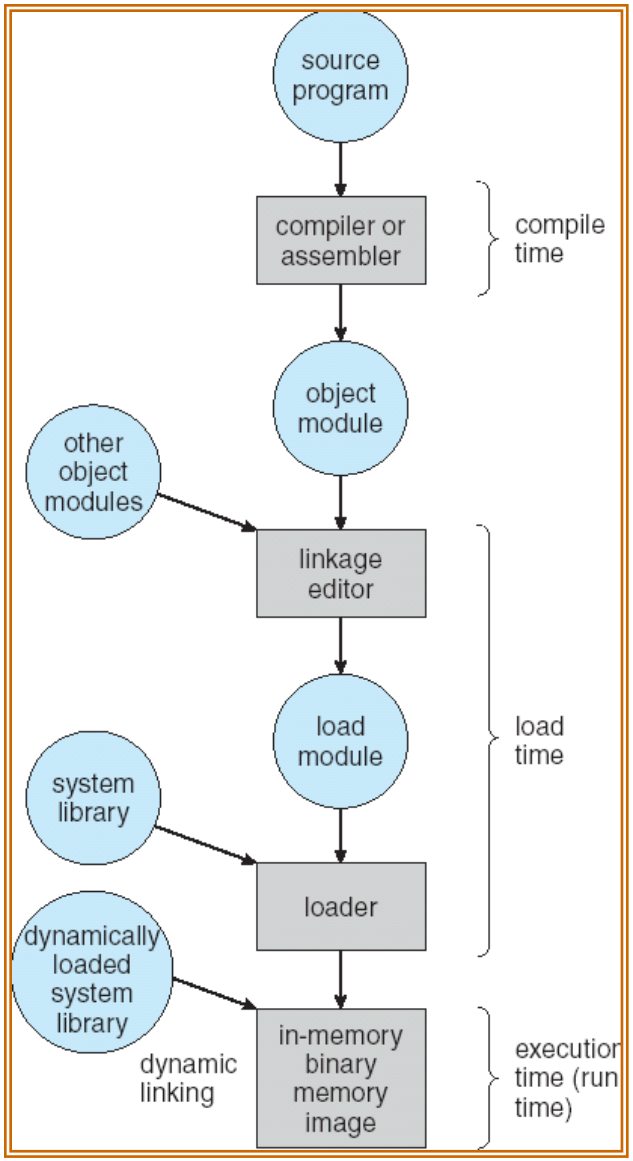
- A pair of **base** and **limit (bound)** registers define the logical address space
  - Base register: starting address for the process
  - Bounds register: ending location of the process



Binding of Instructions and Data to Memory

- 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)

Multistep Processing of a User Program

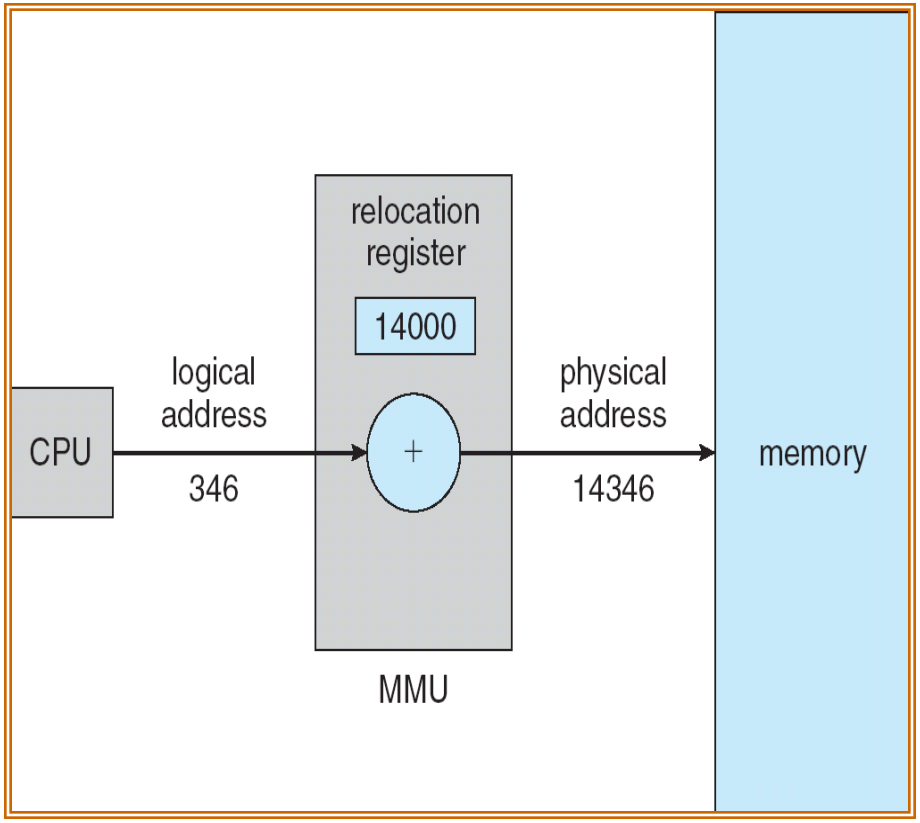


Logical vs. Physical Address Space

- Memory mangement:
  - A logical address space is be bound to a separate physical address space.
  - **Logical address**
    - Reference to a memory location independent of the current assignment of data to memory
    - Translation must be made to the physical address
    - also referred to as **virtual address**
  - **Physical address**
    - The absolute address or actual location in main memory

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address
- In 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
- The user program deals with *logical* addresses; it never sees the *real* physical addresses



Dynamic Loading

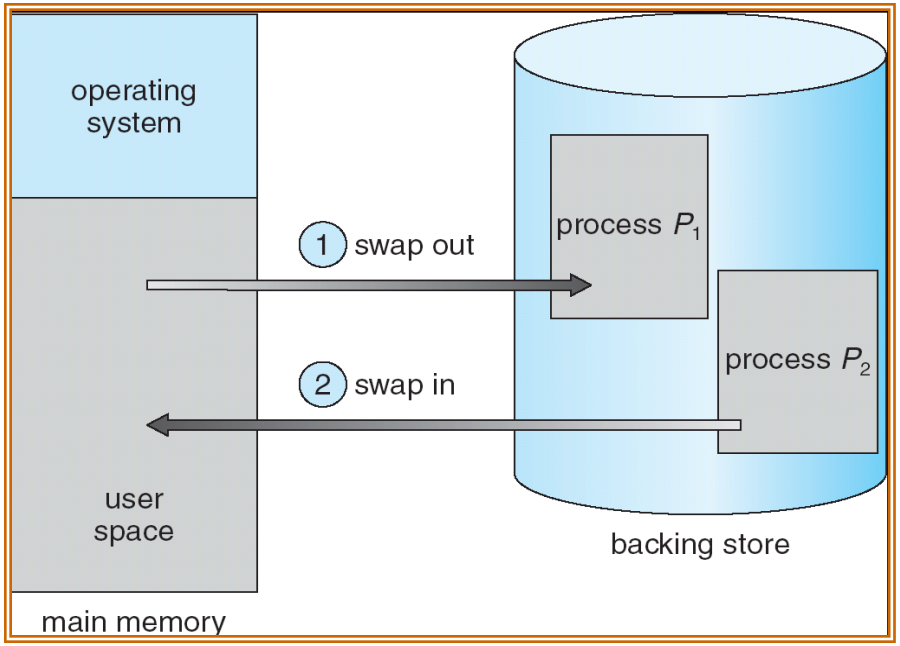
- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- Useful when large amounts of code are needed to handle infrequently occurring cases

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 needed to check if routine is in processes' memory address
- Dynamic linking is particularly useful for libraries
- System also known as **shared libraries**

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- **Roll out, roll in:** lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time.
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk



How to manage memory?

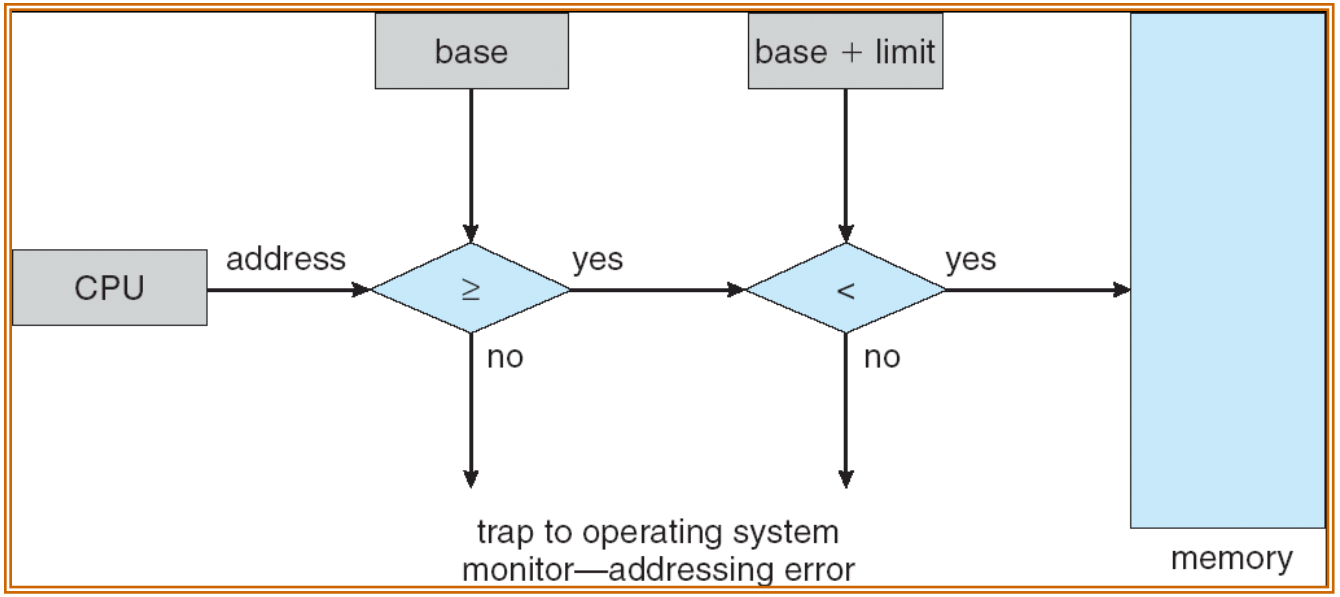
- Contiguous Allocation
- Paging
- Segmentation
- Hybrid (Paging + Segmentation)

Contiguous Allocation

- Main memory usually has two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory

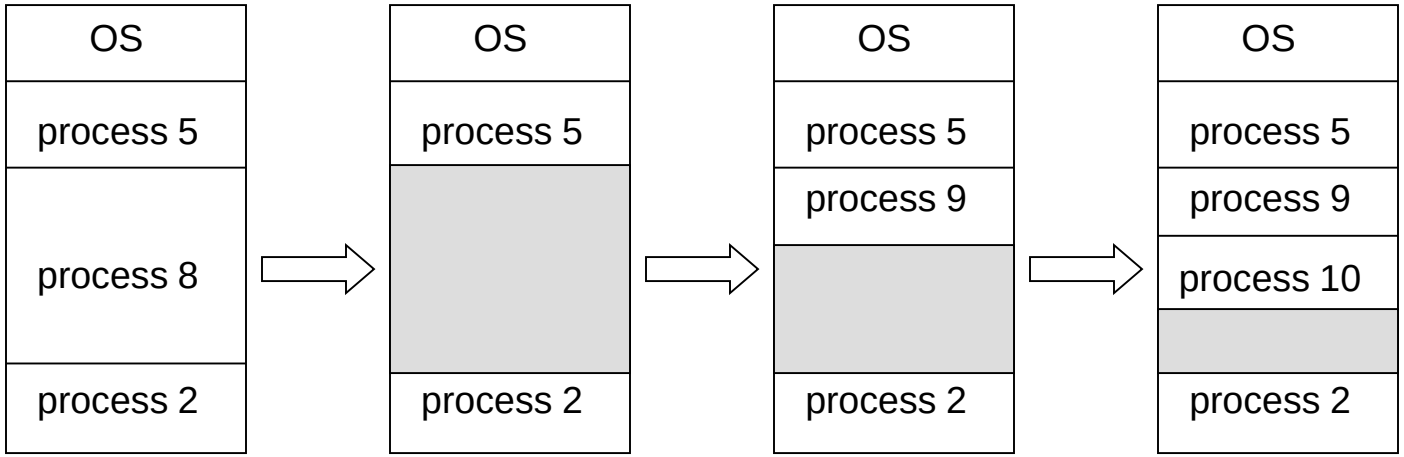
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*

HW address protection with base and limit registers



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 ___ leftover hole
    - small
- **Worst-fit**: Allocate the *largest* hole; must also search entire list
  - Produces the ____ leftover hole
    - large

Fragmentation

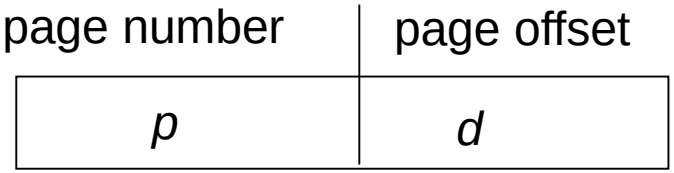
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
  - How to reduce external fragmentation
    - by **compaction**
      - Shuffle memory contents to place all free memory together in one large block
      - Compaction is possible *only* if relocation is dynamic, and is done at execution time
- **Internal Fragmentation** – When we break memory into fix-sized blocks, some memory inside a block may be too small to be used.

Paging

- 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
- Internal fragmentation

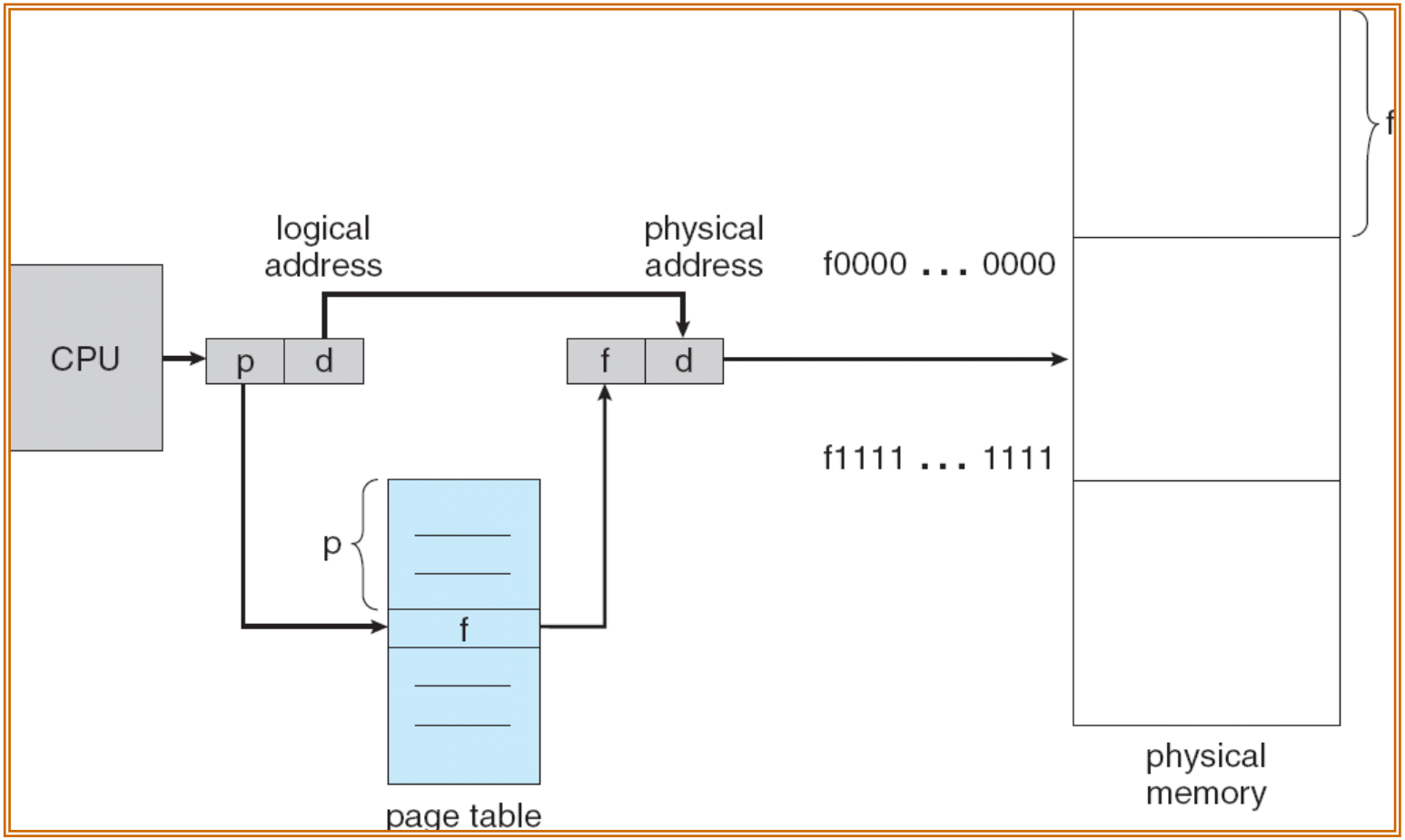
Address Translation Scheme

- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an index into a *page table* which contains base address 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

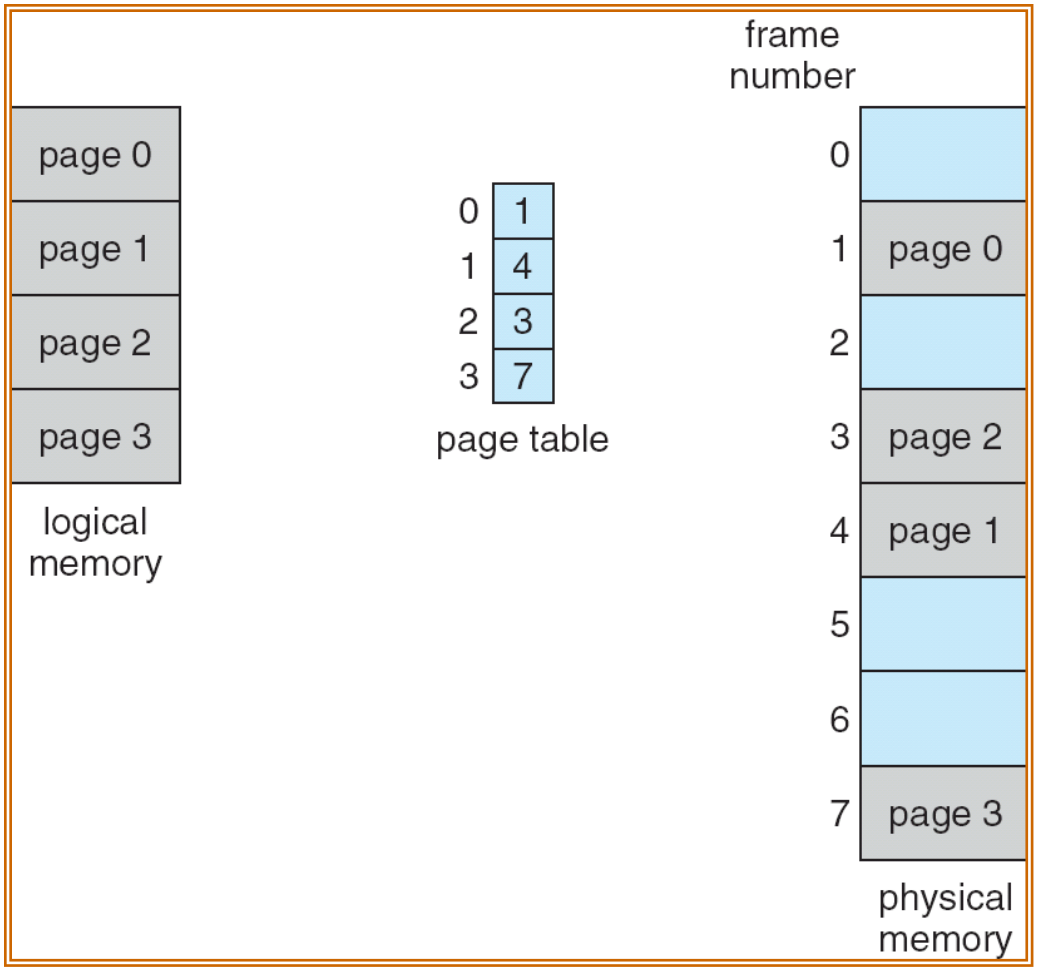


  - *How many pages totally?*
  - *What is the size of a page?*

Paging Hardware



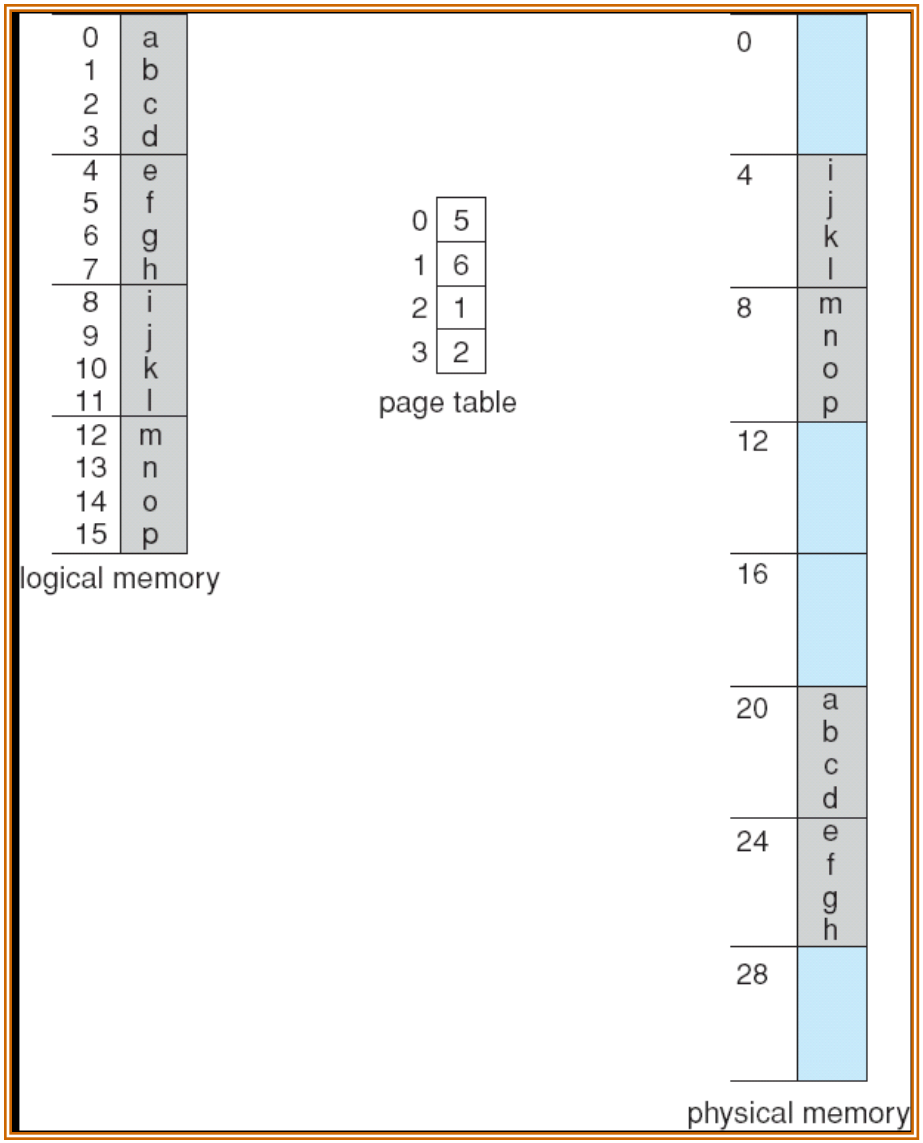
Paging Model of Logical and Physical Memory



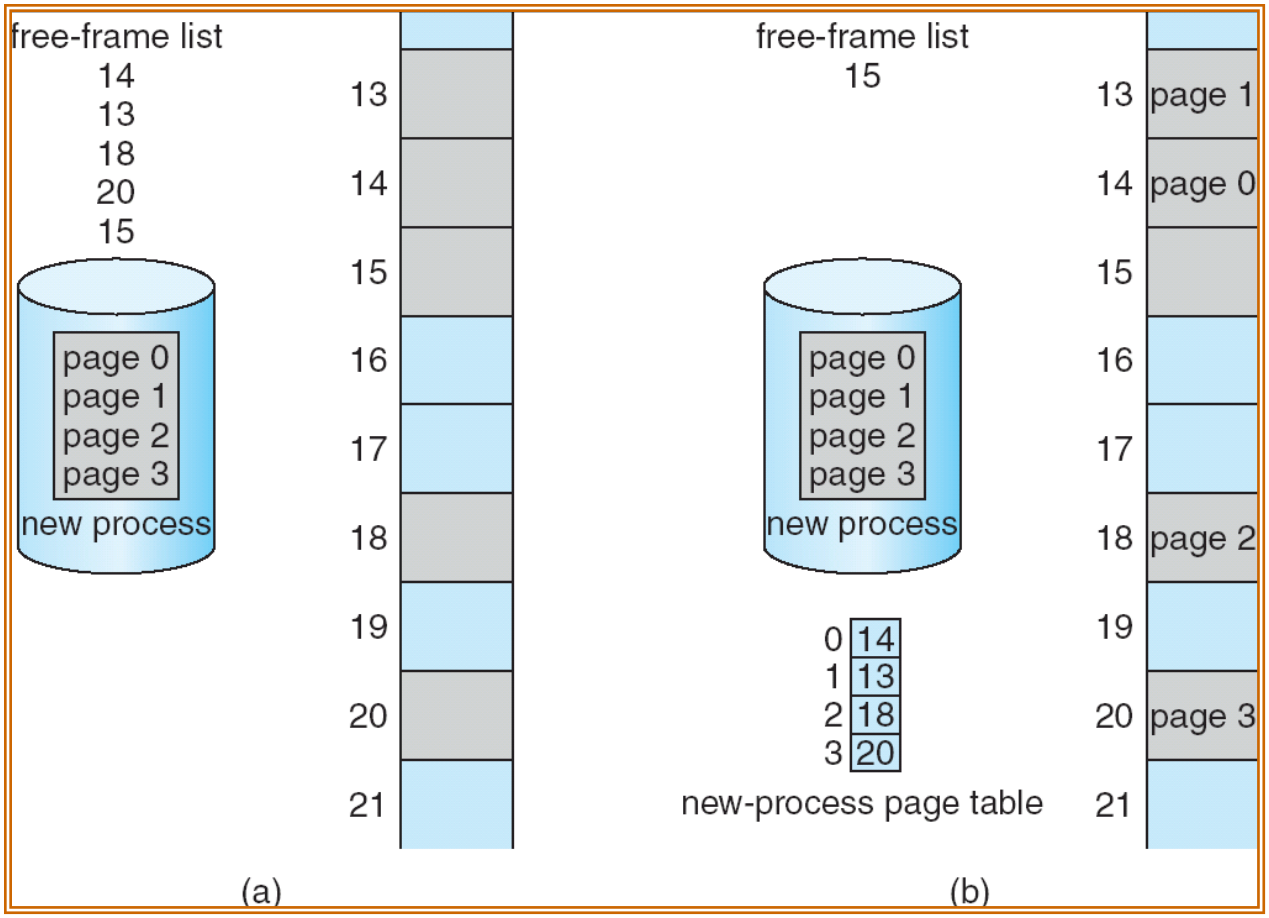
Paging Example

Logically:
- total 4pages,
- each page has 4 addresses

Physically:
- total 8 pages
- each page has 4 addresses



Free Frames



Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires ____ memory accesses.
  - Two. One for the page table and one for the data/instruction.
  - Any solutions?
    - The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Associative Memory

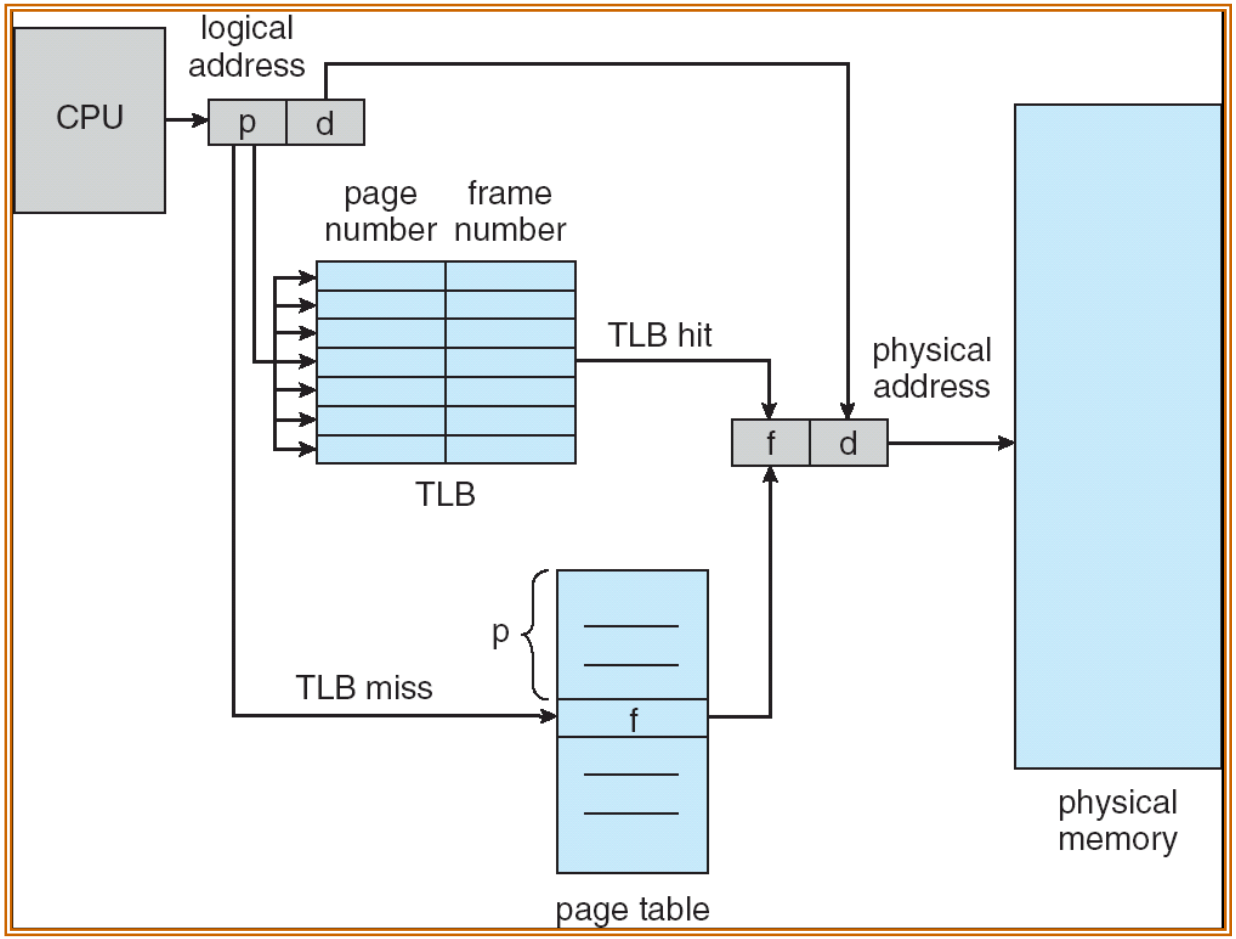
- Associative memory – parallel search

Page #	Frame #

Address translation (p, d)

- If p is in associative register, get frame # out
- Otherwise get frame # from page table in memory

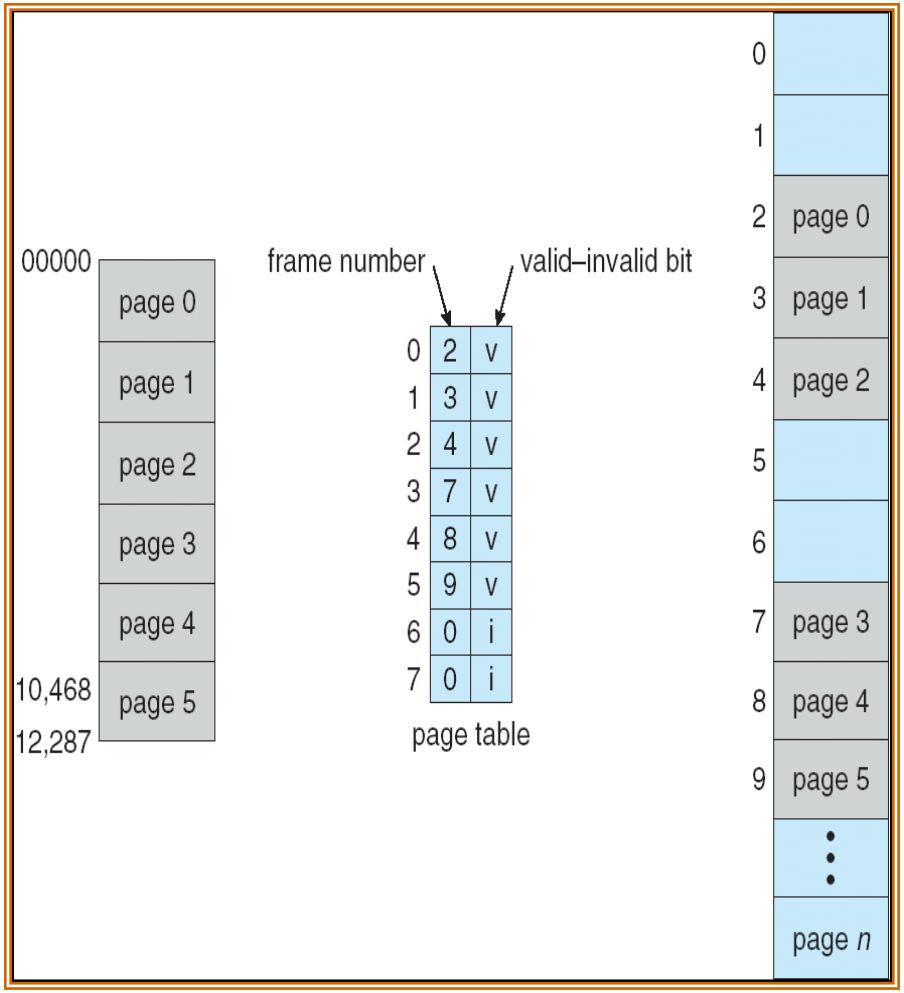
Paging Hardware With TLB



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



Structure of the Page Table

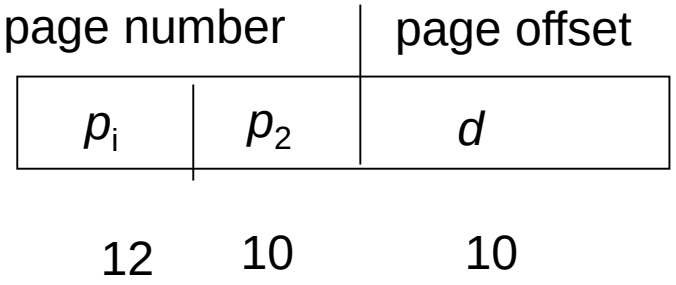
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

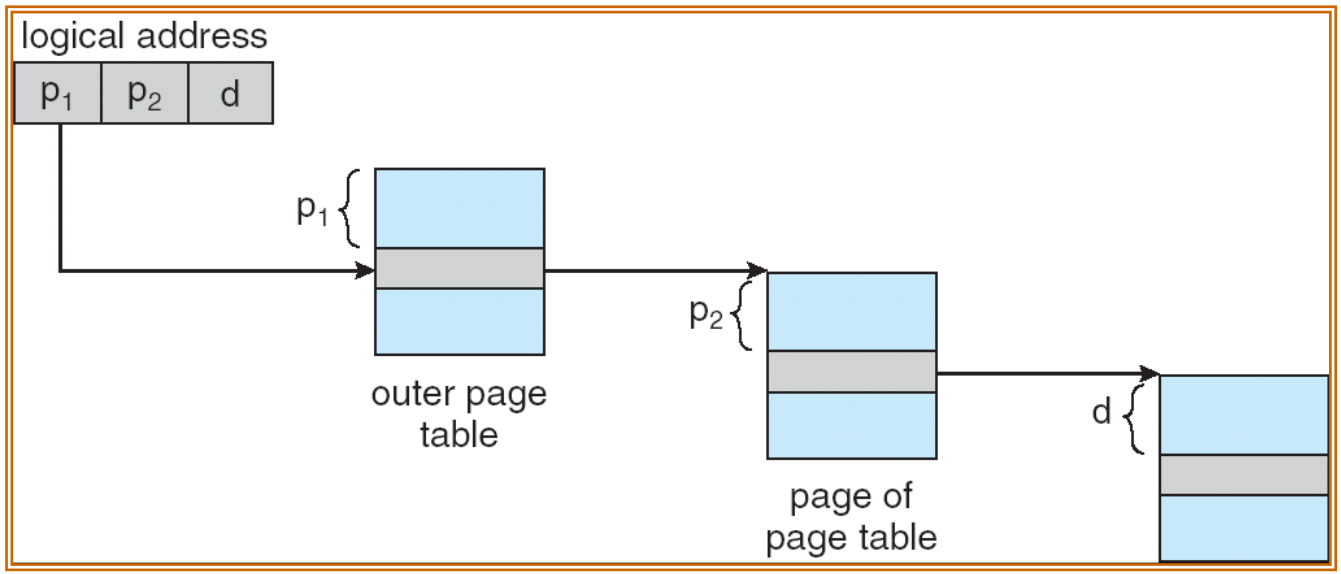
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:



- 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

Address-Translation Scheme



Three-level Paging Scheme

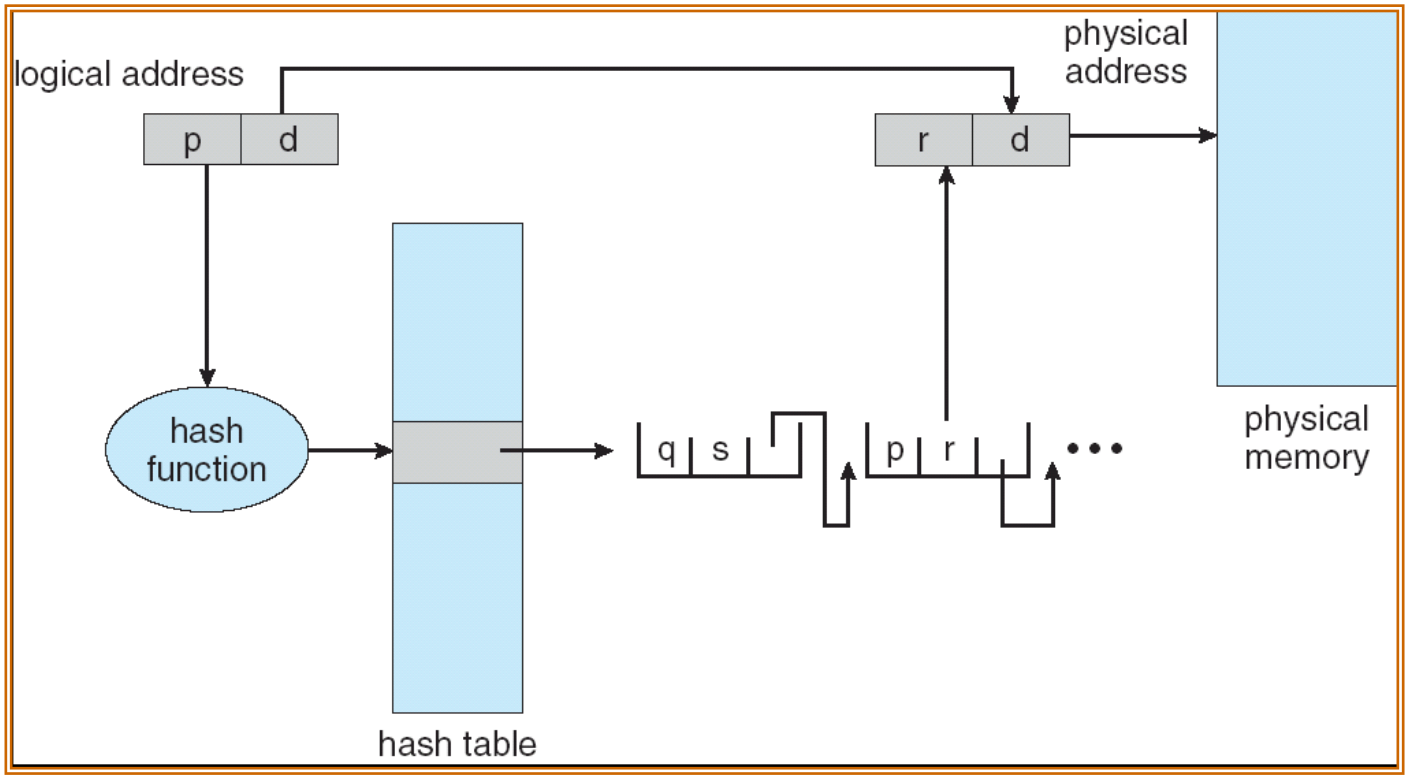
outer page	inner page	offset
p_1	p_2	d
42	10	12

2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

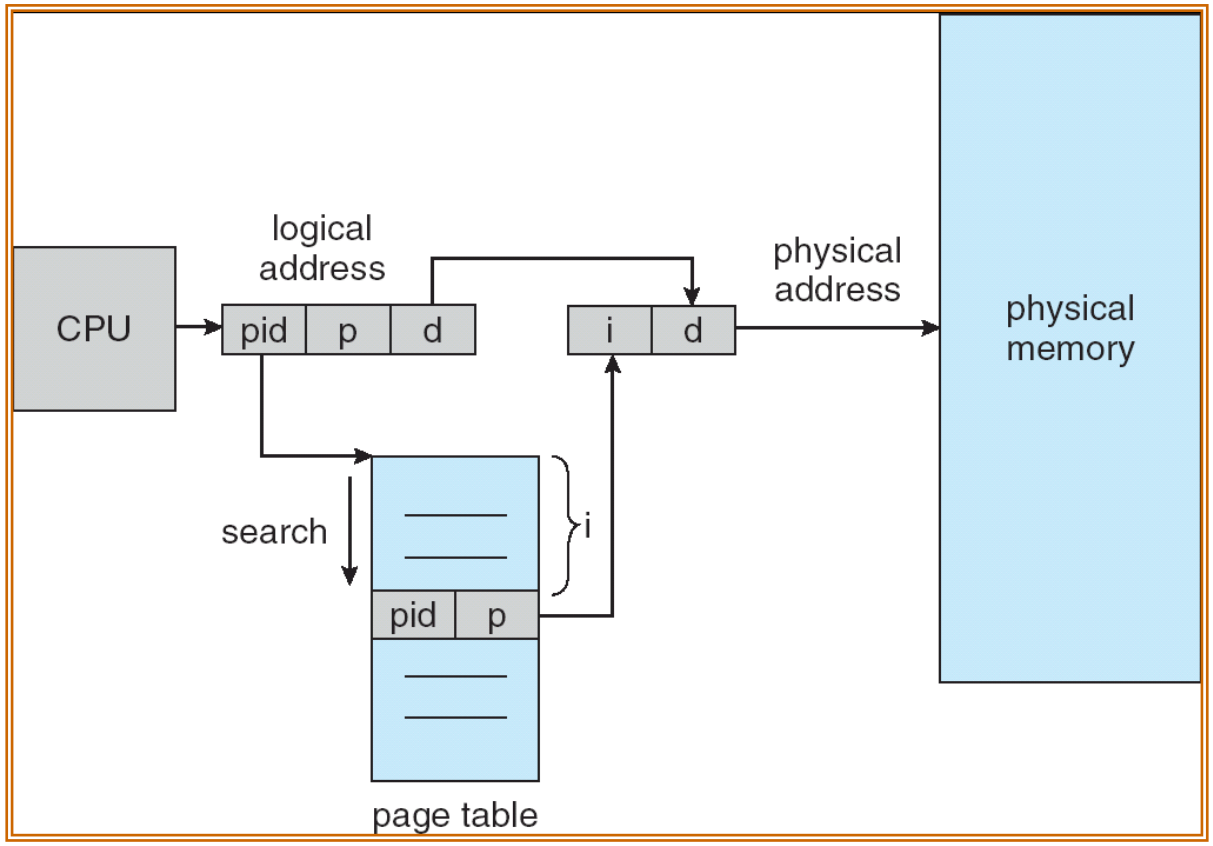
Hashed Page Tables

- 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.
- Virtual page numbers are compared in this chain searching for a match. If a match is found, the corresponding physical frame is extracted.

Hashed Page Table



Inverted Page Table Architecture

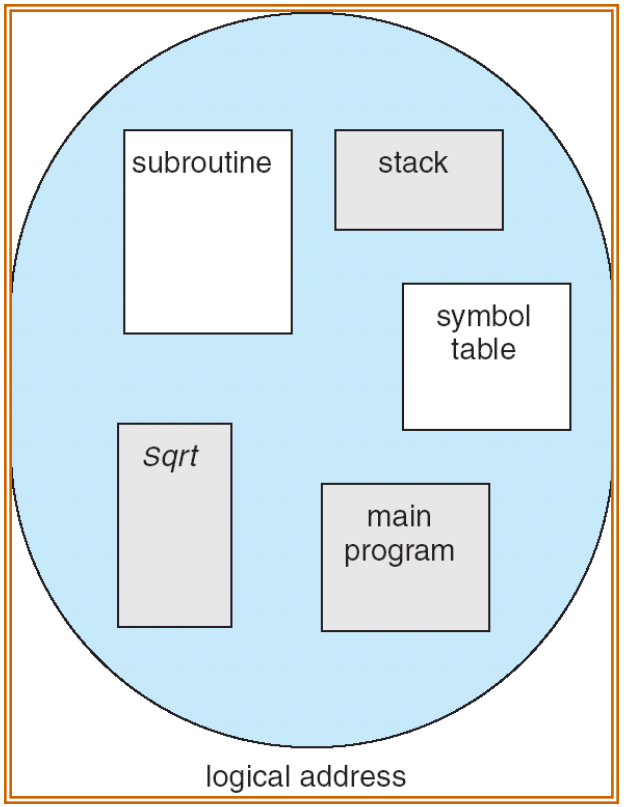


Inverted Page Table

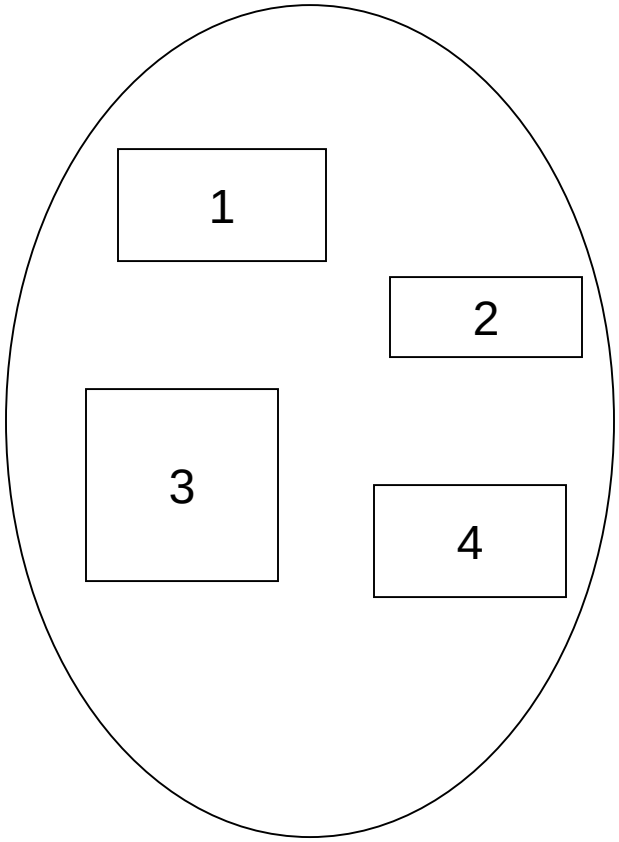
- *One entry for each frame of memory*
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Pros:
  - Decreases memory needed to store each page table
- Cons:
  - increases time needed to search the table when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

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:
  - main program
  - Procedure
  - Function
  - Method
  - Object
  - Local variables, global variables
  - Common block
  - Stack
  - Symbol table, arrays



Logical View of Segmentation



user space

physical memory space

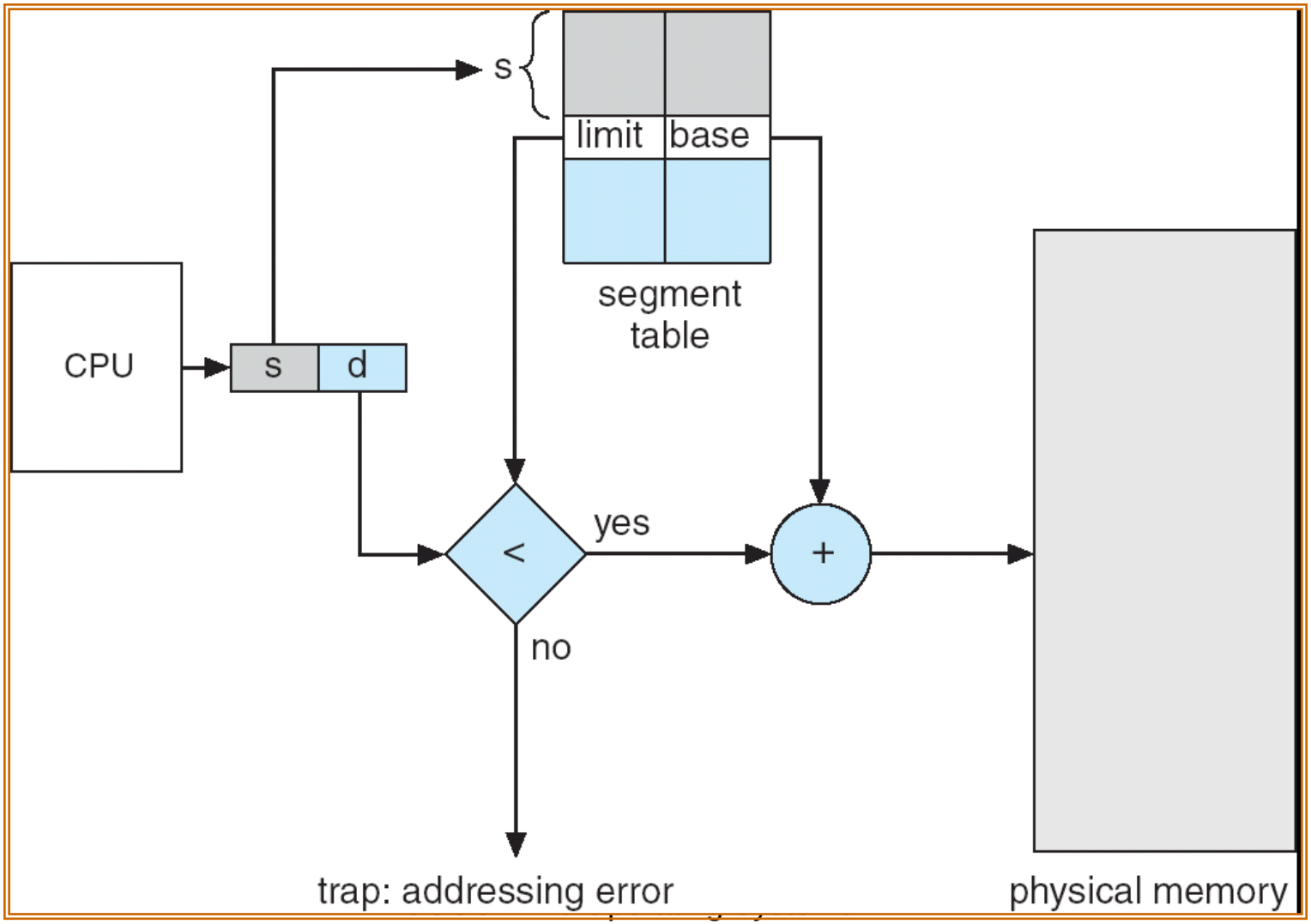
Segmentation Architecture

- Logical address consists of a tuple:

  <segment-number, offset>,

- **Segment table** – maps two-dimensional logic addresses into one-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

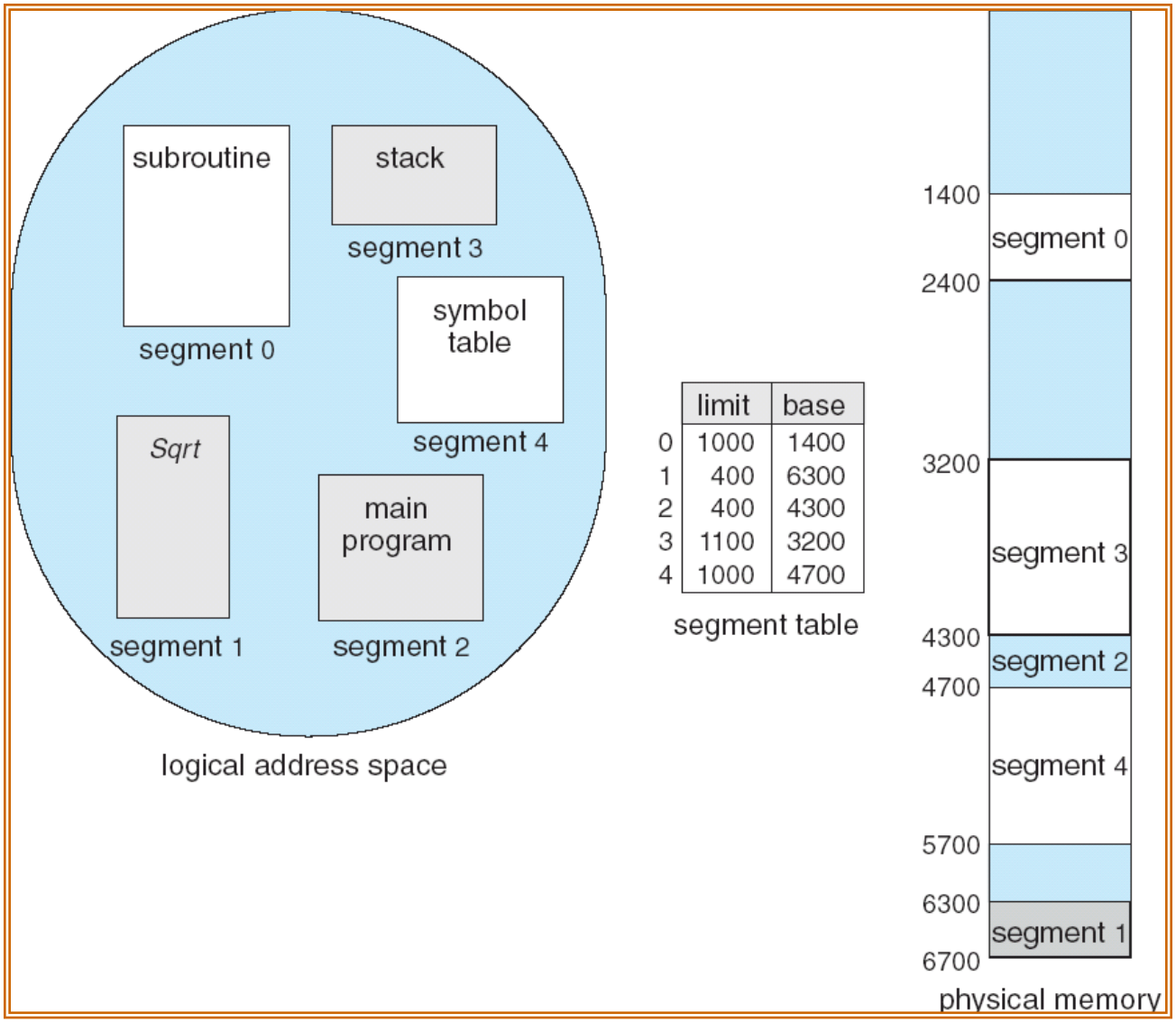
Segmentation Hardware



Segmentation Architecture (Cont.)

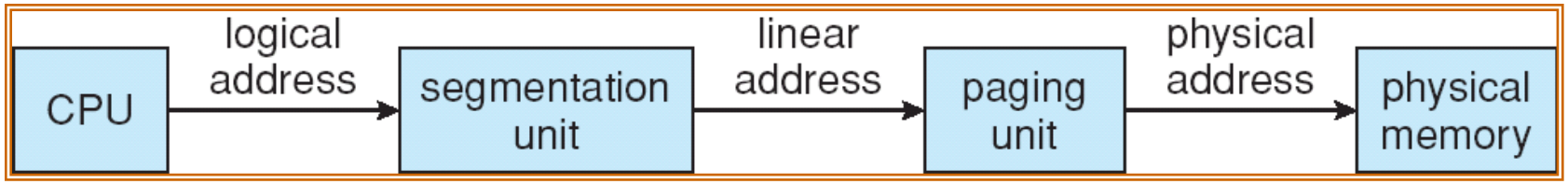
- Protection
  - With each entry in segment table associate:
    - validation bit = 0 ⇒ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Example of Segmentation

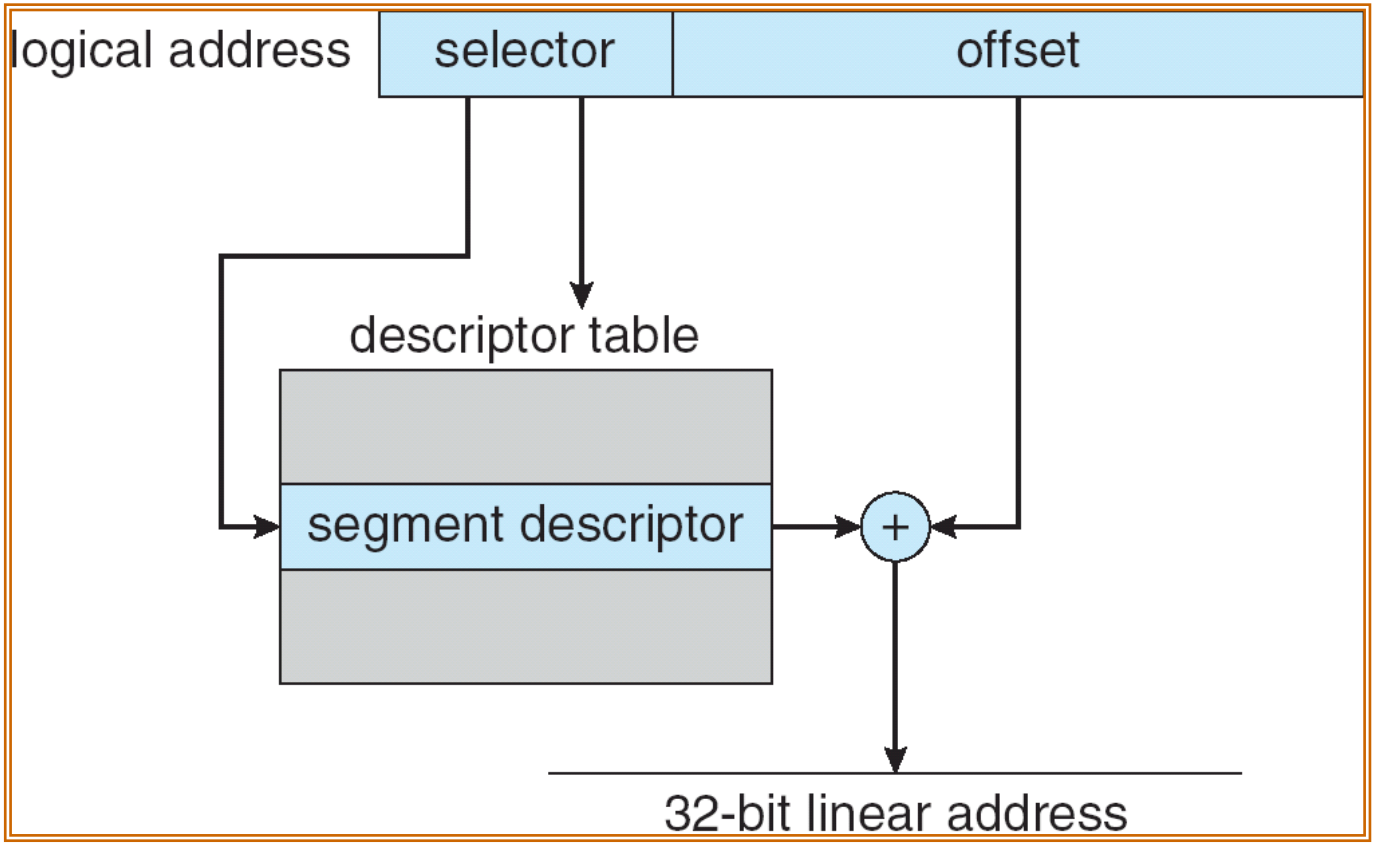


Example: The Intel Pentium

- Supports both segmentation and segmentation with paging
- CPU generates logical address
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU



Intel Pentium Segmentation



Pentium Paging Architecture