

Chapter 8: Memory Management





Chapter 8: Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation





Background

- Program must be brought into memory and placed within a process for it to be run
- **Input queue** – collection of processes on the disk that are waiting to be brought into memory to run the program
- User programs go through several steps before being run





Binding of Instructions and Data to Memory

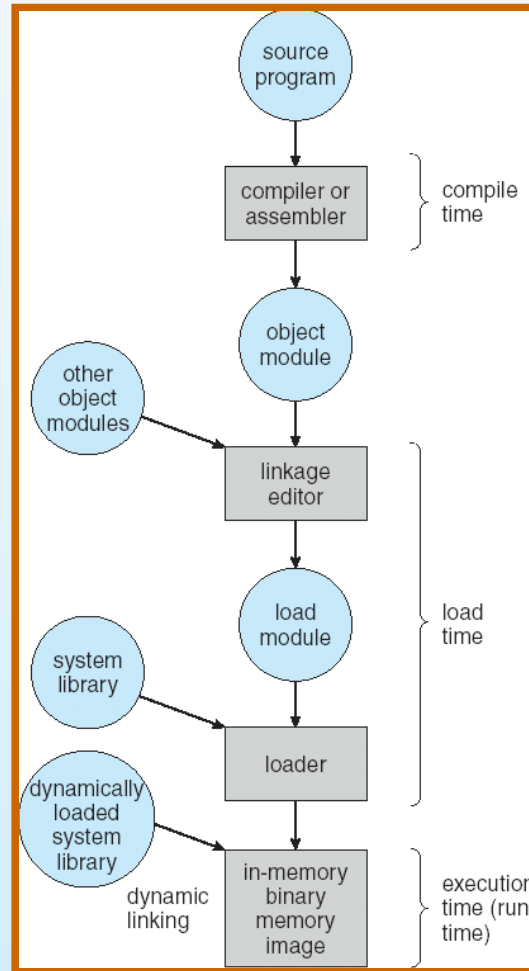
Address binding of instructions and data to memory addresses can happen at three different stages

- **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

- 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





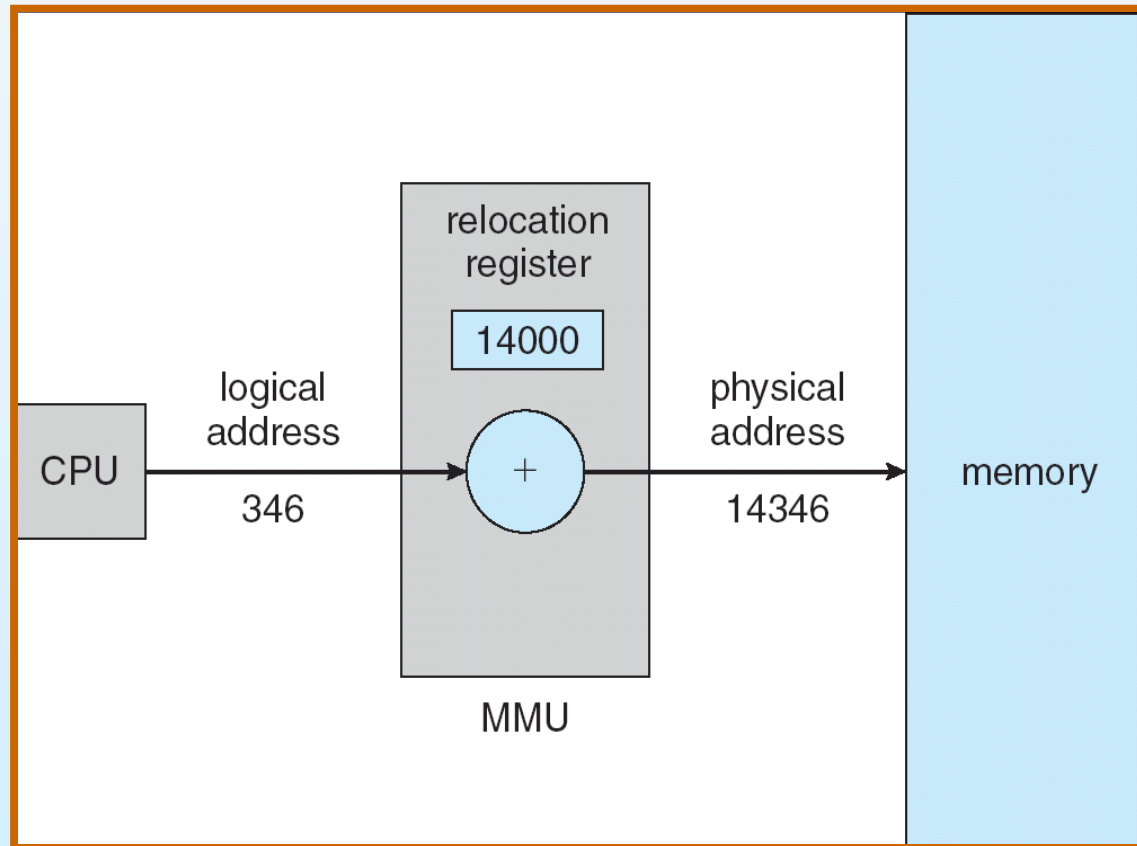
Memory-Management Unit (MMU)

- 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 relocation using a relocation register





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





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





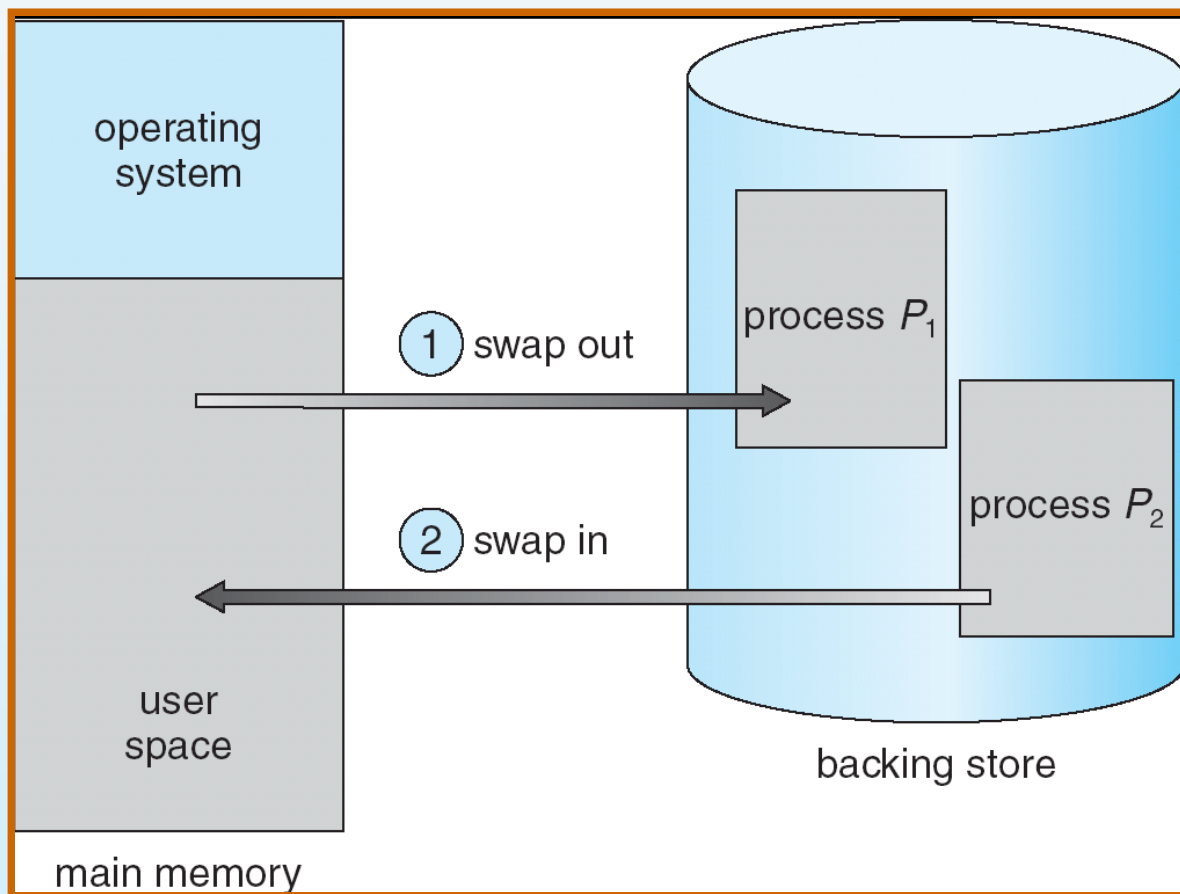
Swapping

- 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
- 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)





Schematic View of Swapping

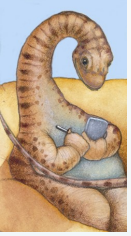




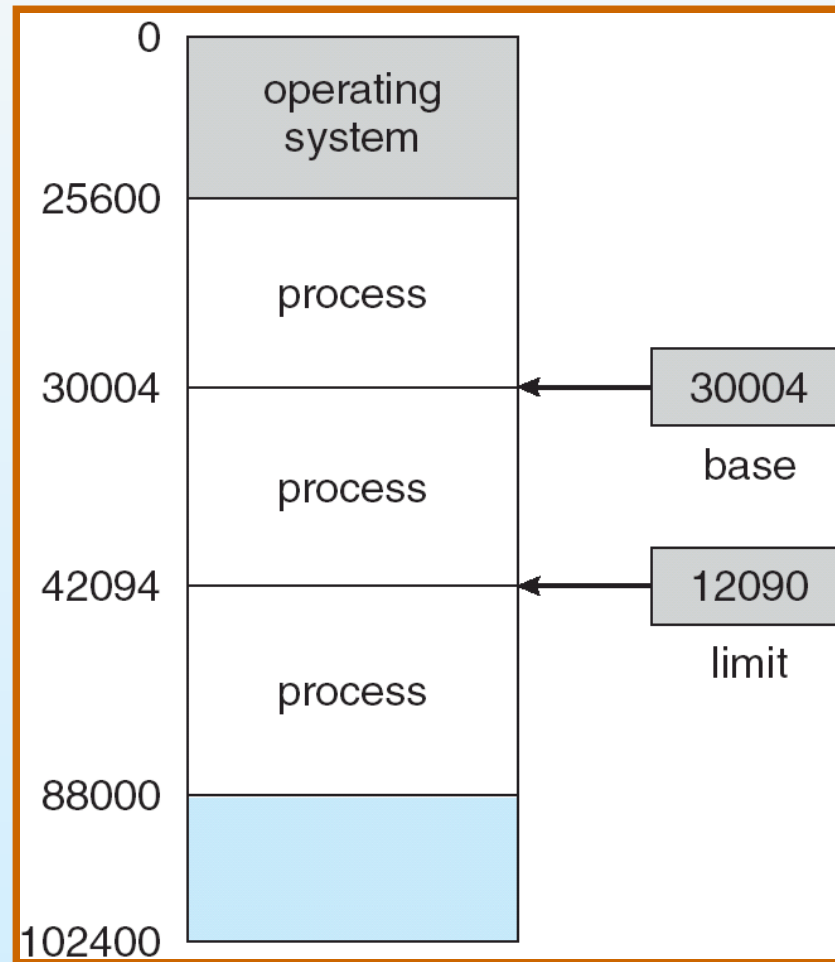
Contiguous Allocation

- Main memory usually into two partitions:
 - Resident operating system, usually held in low memory with interrupt vector
 - User processes then held in high memory
- Memory mapping and protection
 - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data
 - 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



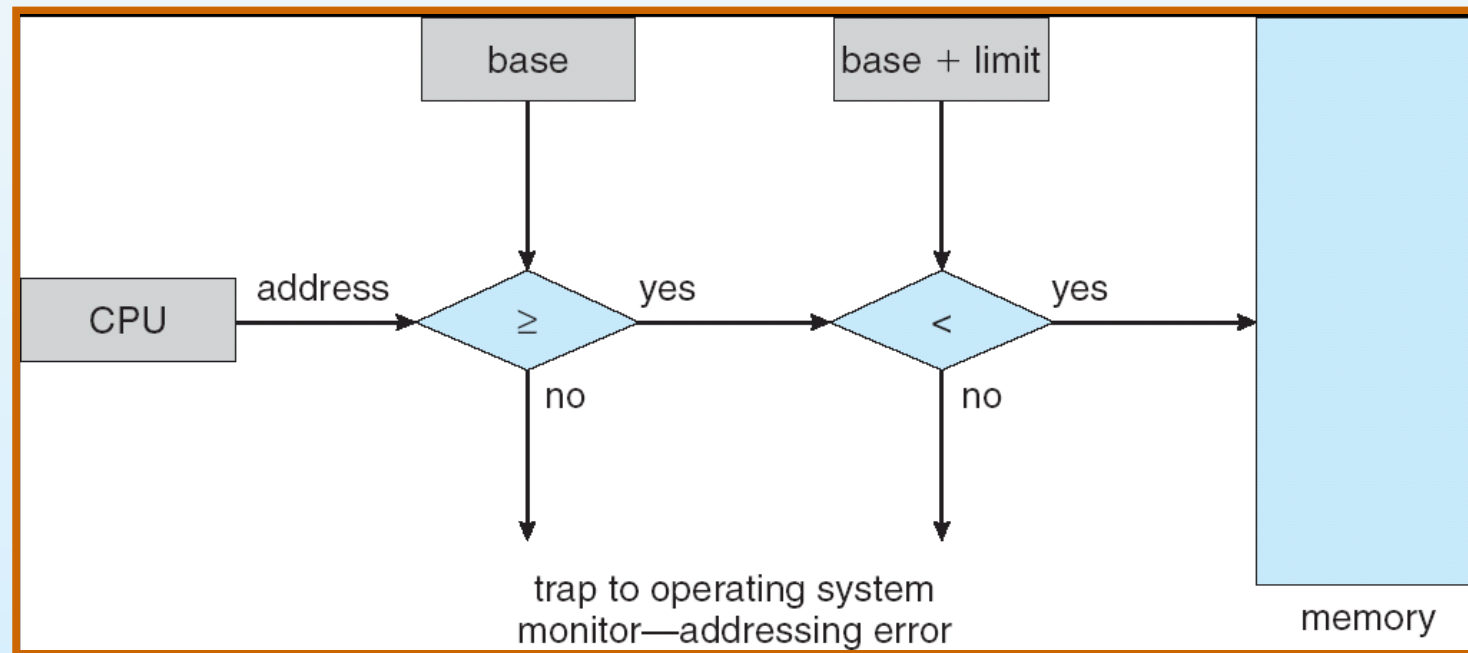


A base and a limit register define a logical address space





HW address protection with base and limit registers

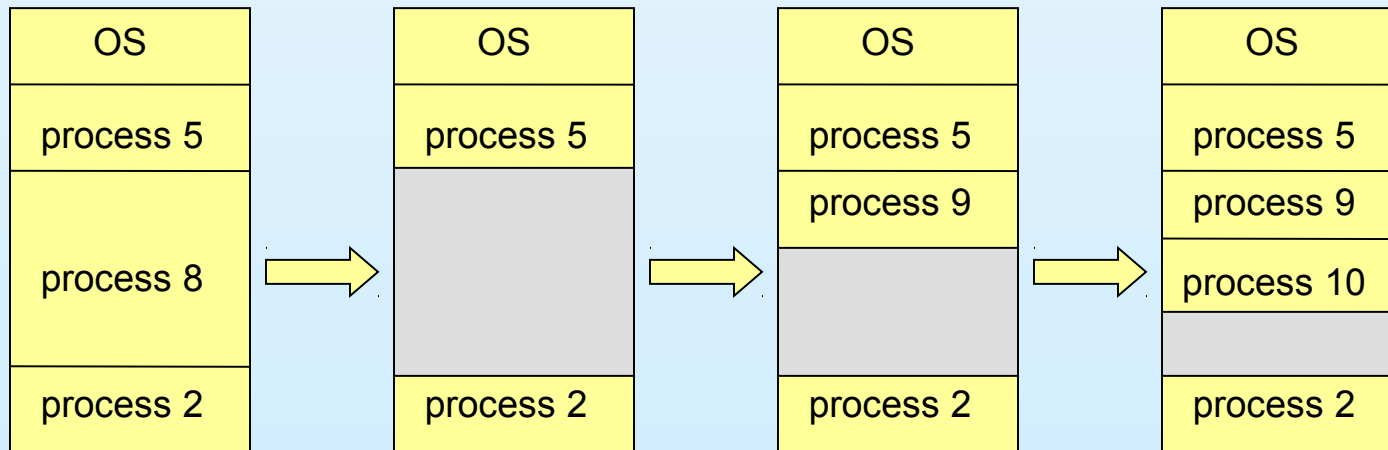




Contiguous Allocation (Cont.)

■ 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 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
- 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 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





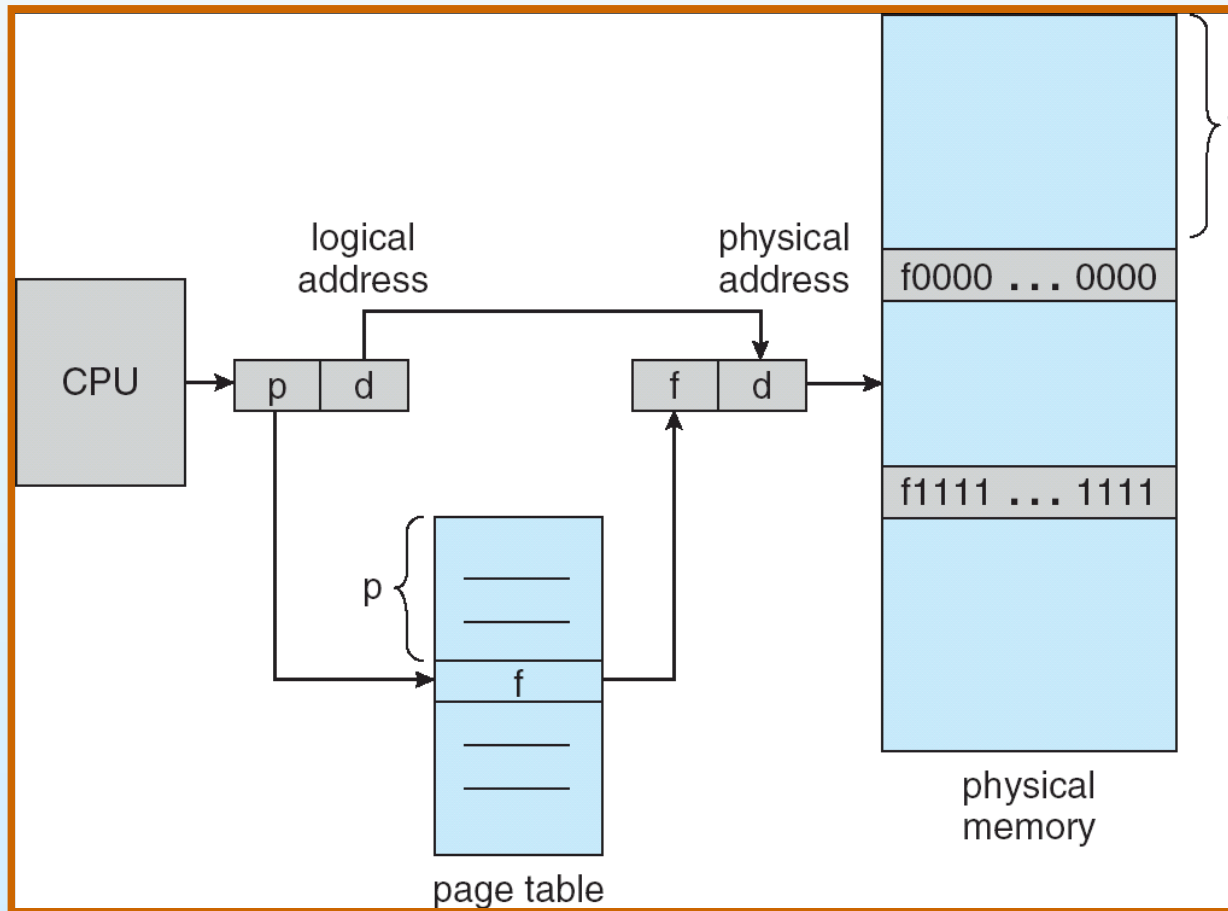
Address Translation Scheme

- 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



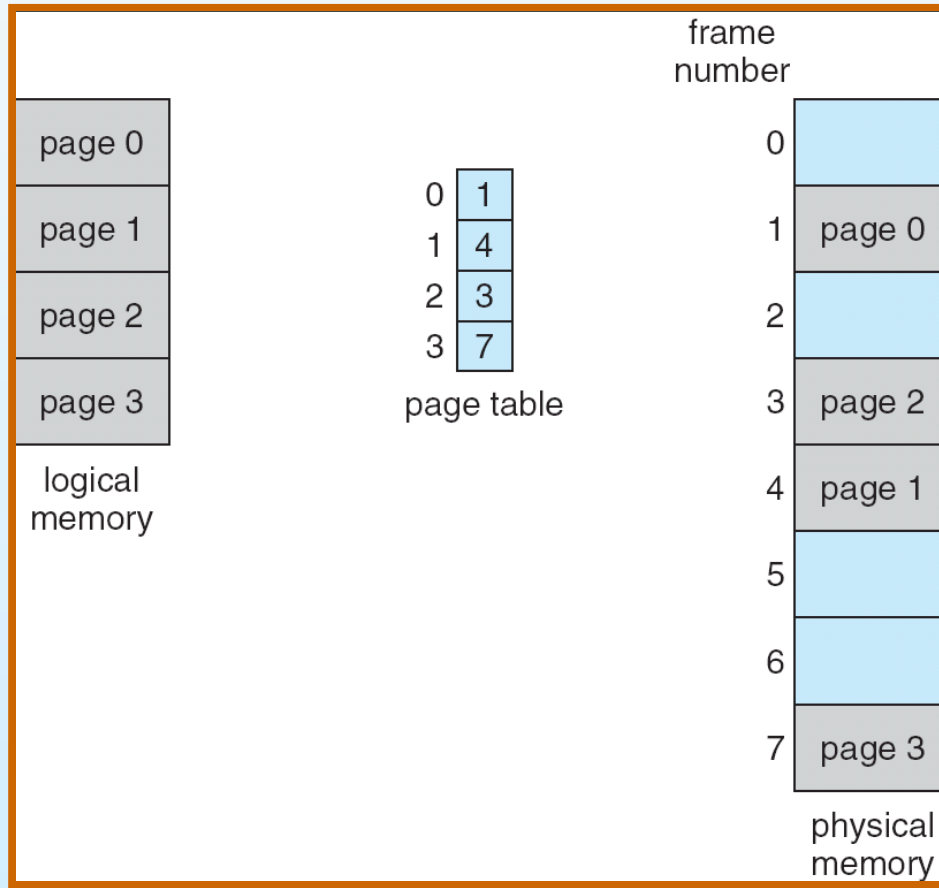


Address Translation Architecture



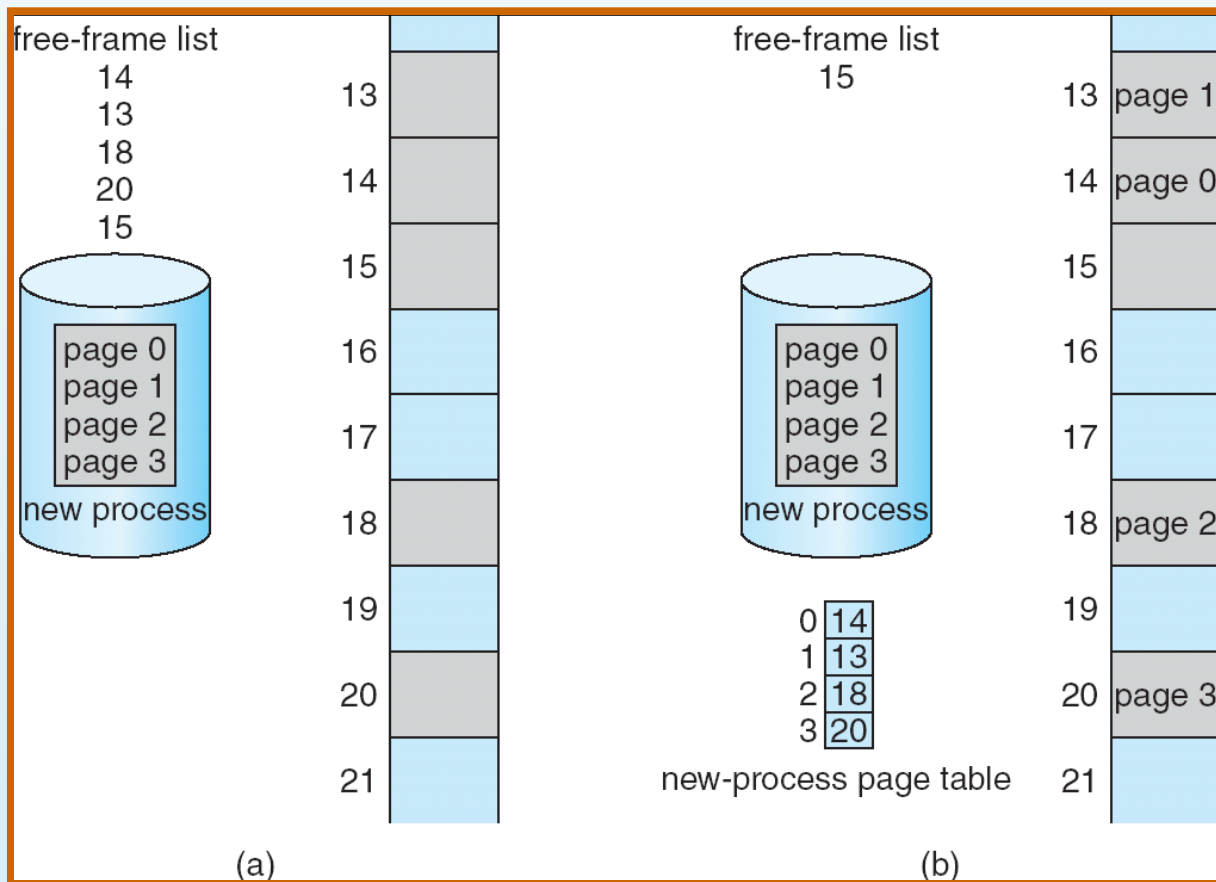


Paging Example





Free Frames





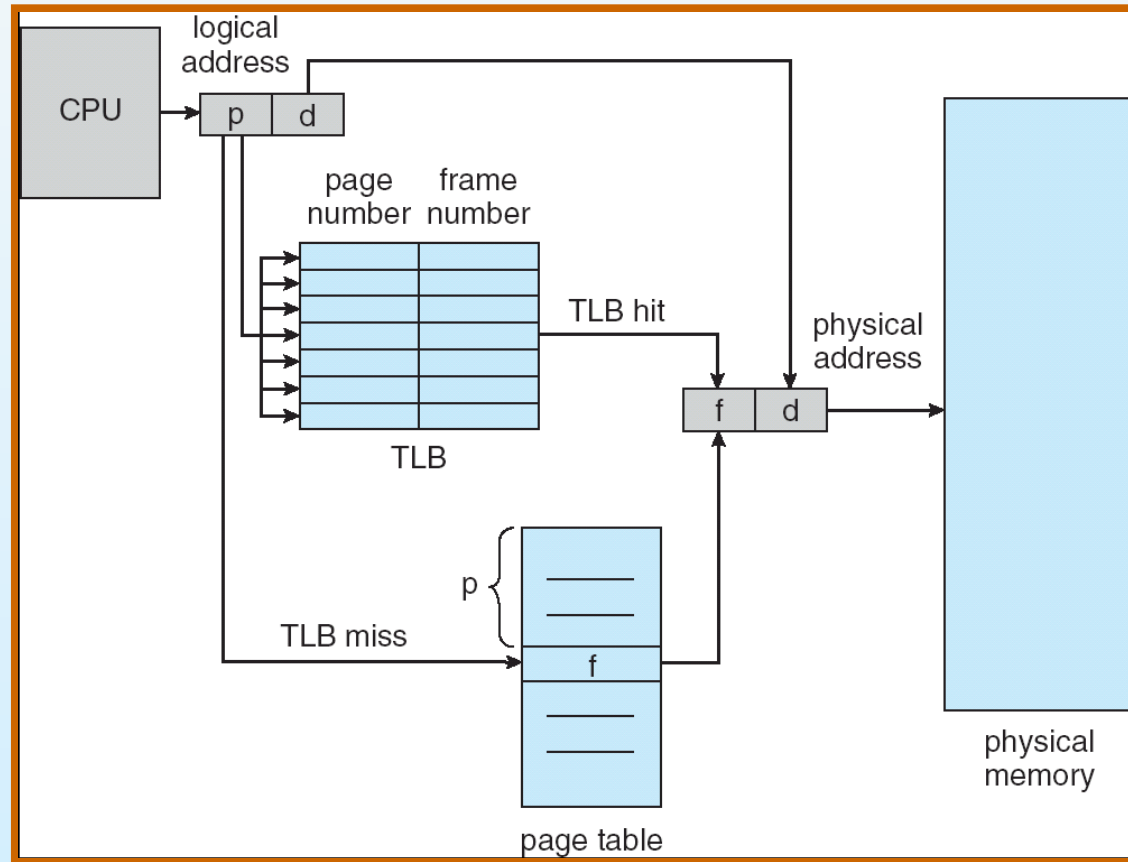
Implementation of Page Table

- Page table is kept in main memory
- *Page-table base register* (PTBR) points to the page table
- *Page-table length register* (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 **associative memory** or **translation look-aside buffers (TLBs)**





Paging Hardware With TLB





Page Table Structure

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables





Hierarchical Page Tables

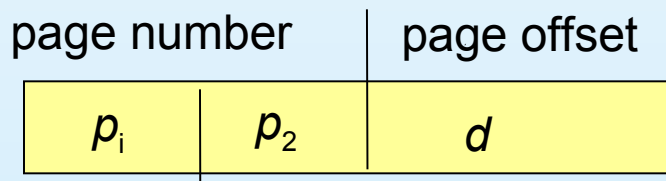
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table





Two-Level Paging Example

- A logical address (on 32-bit machine with 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 p_1 is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

10

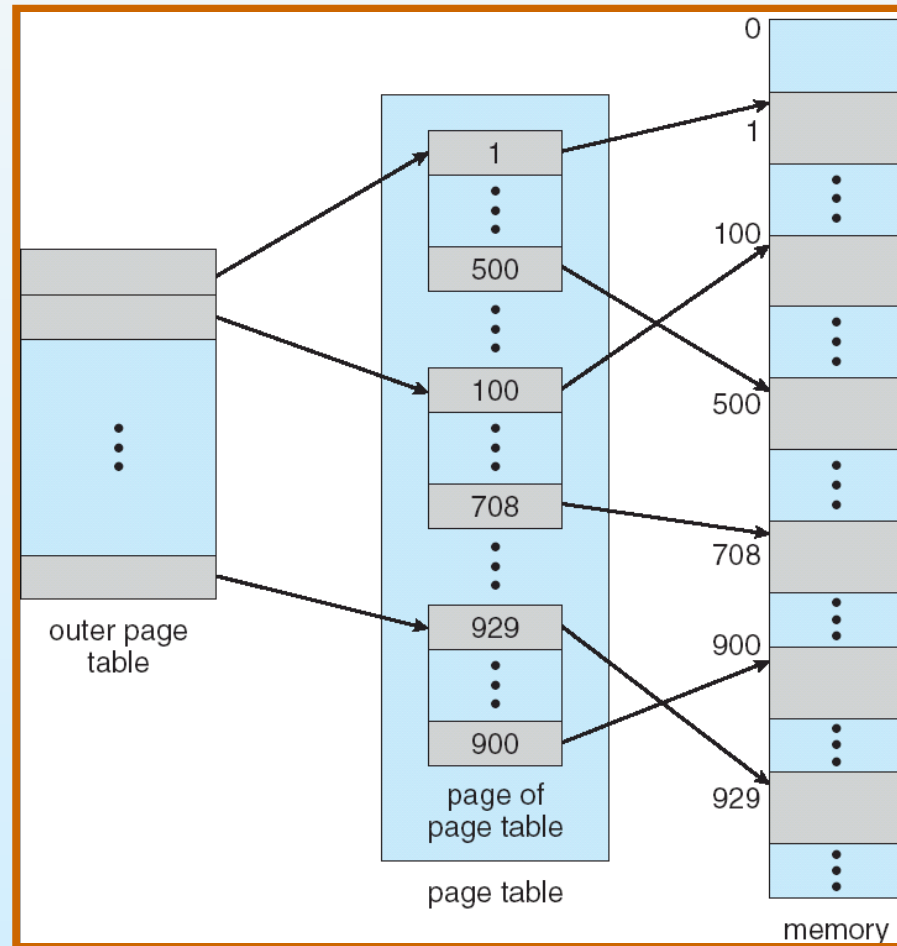
10

12





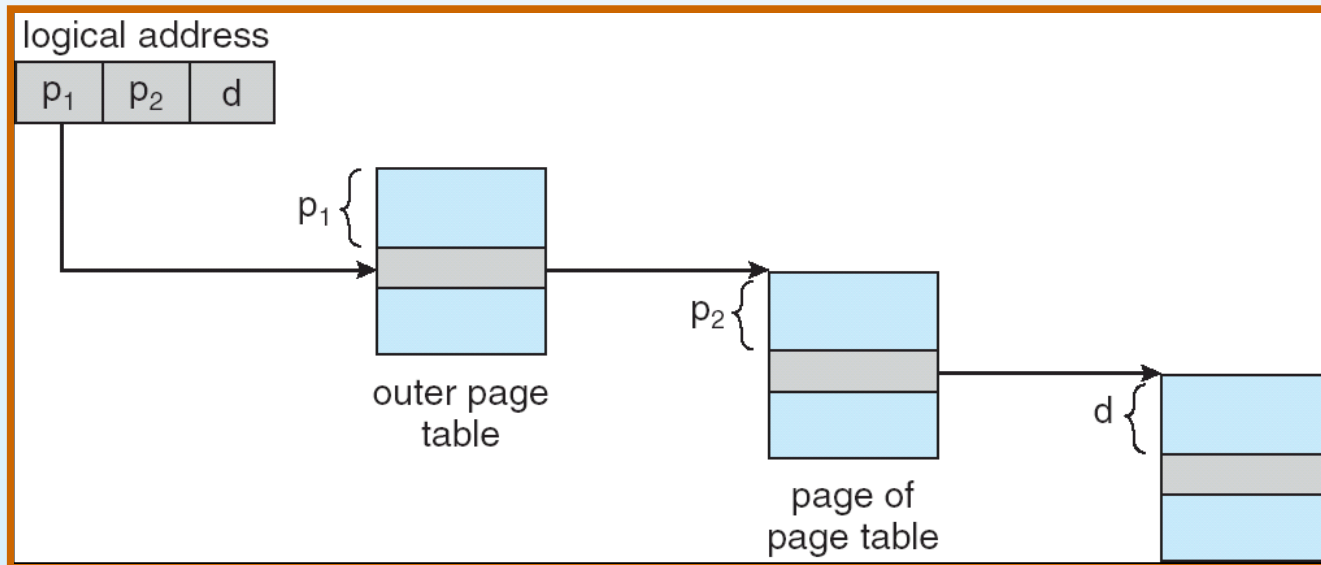
Two-Level Page-Table Scheme





Address-Translation Scheme

- Address-translation scheme for a two-level 32-bit paging architecture





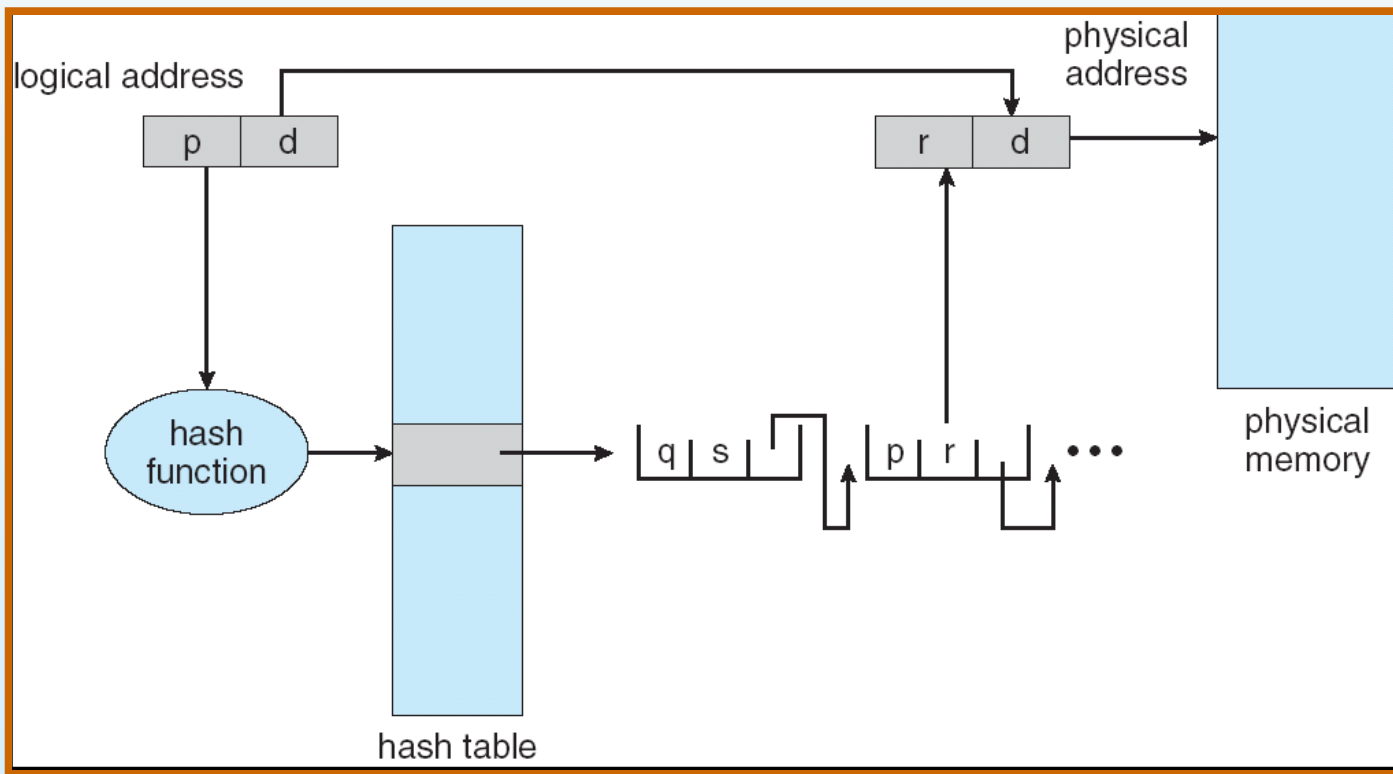
Hashed Page Tables

- 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





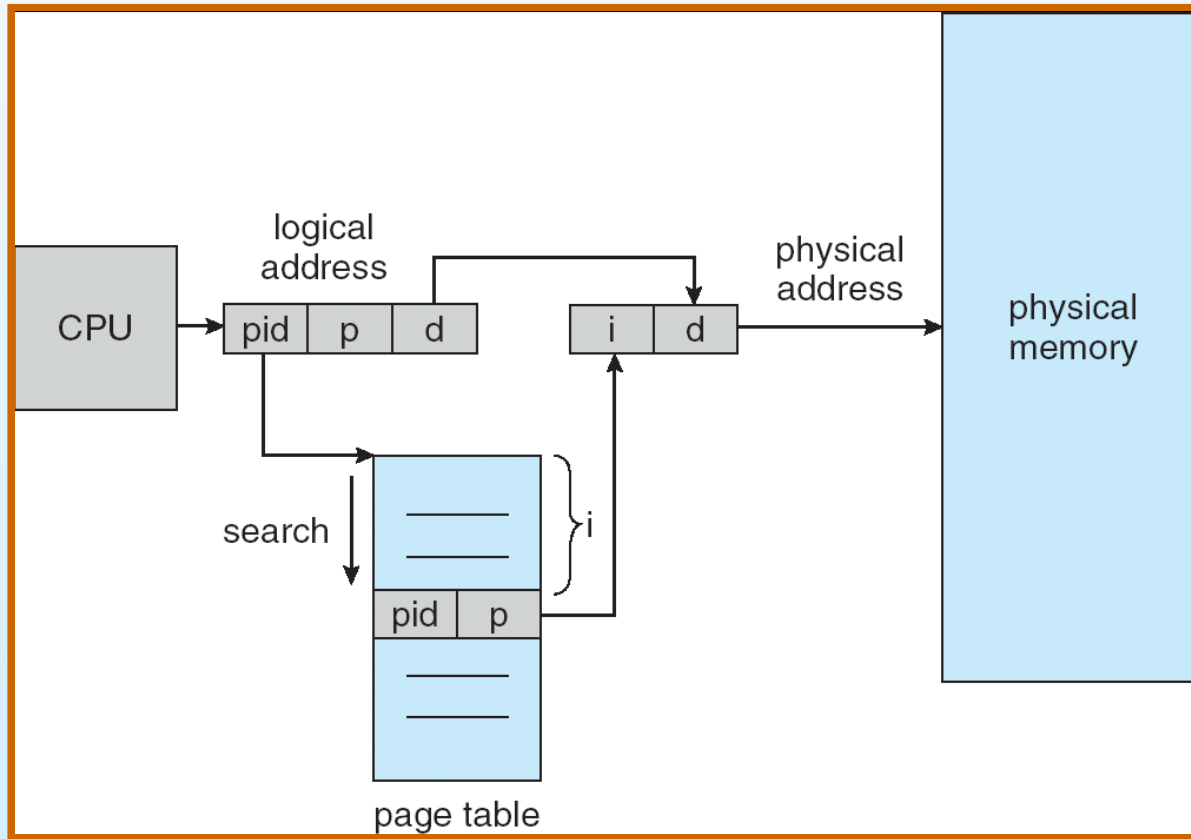
Inverted Page Table

- 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 Table Architecture





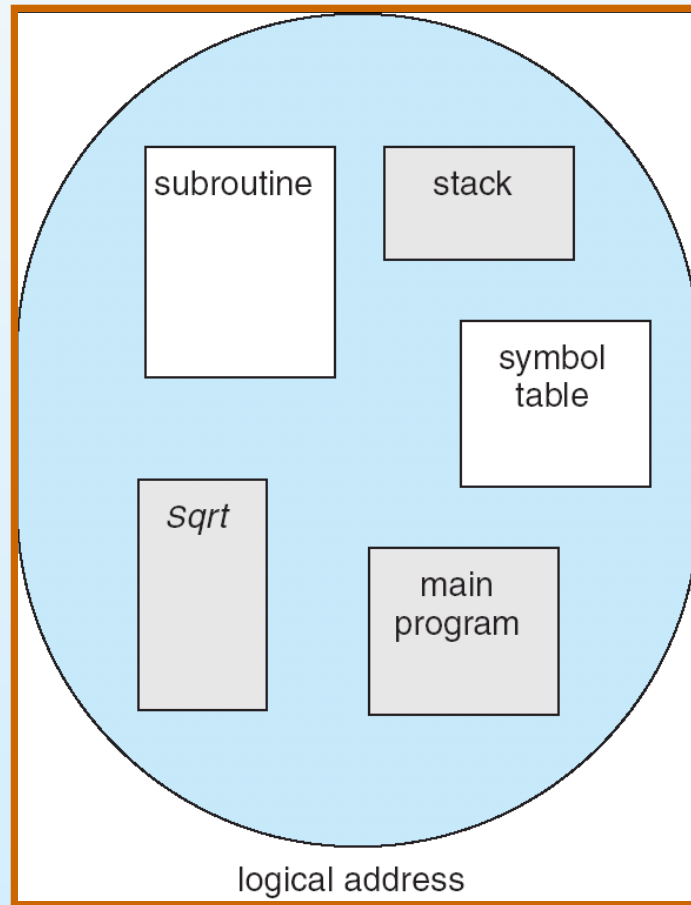
Segmentation

- 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



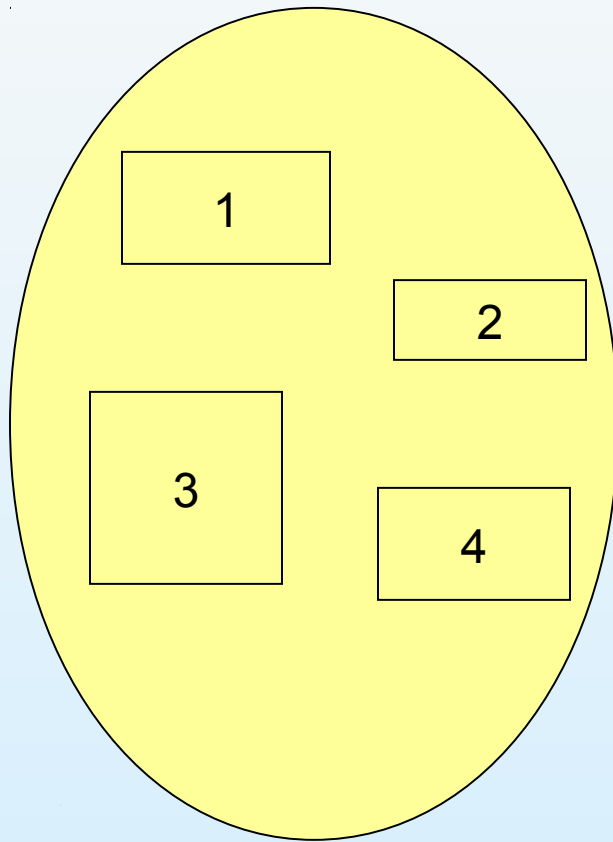


User's View of a Program

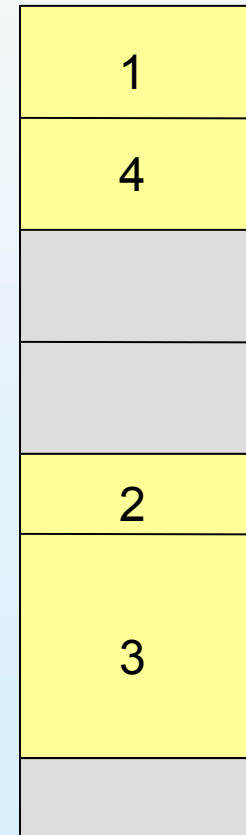




Logical View of Segmentation



user space

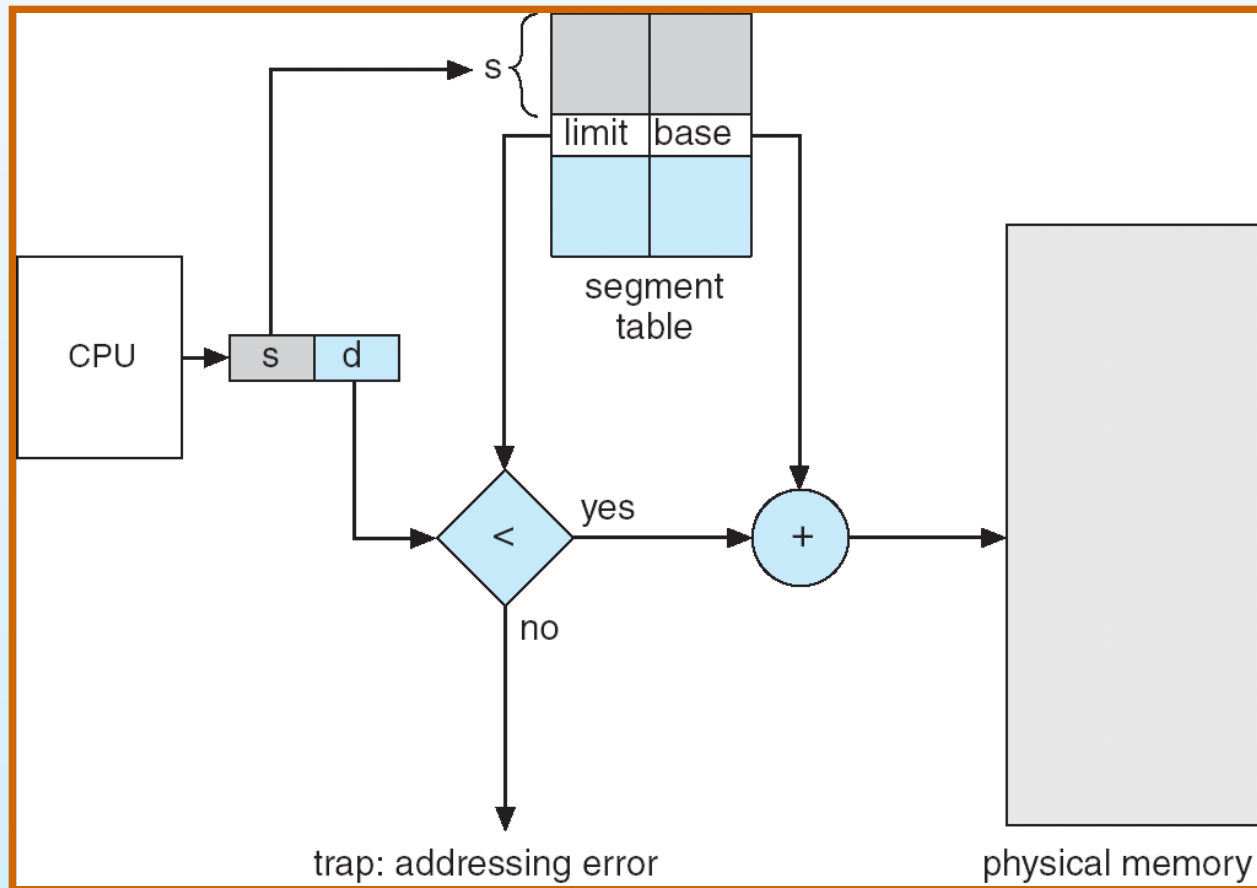


physical memory space



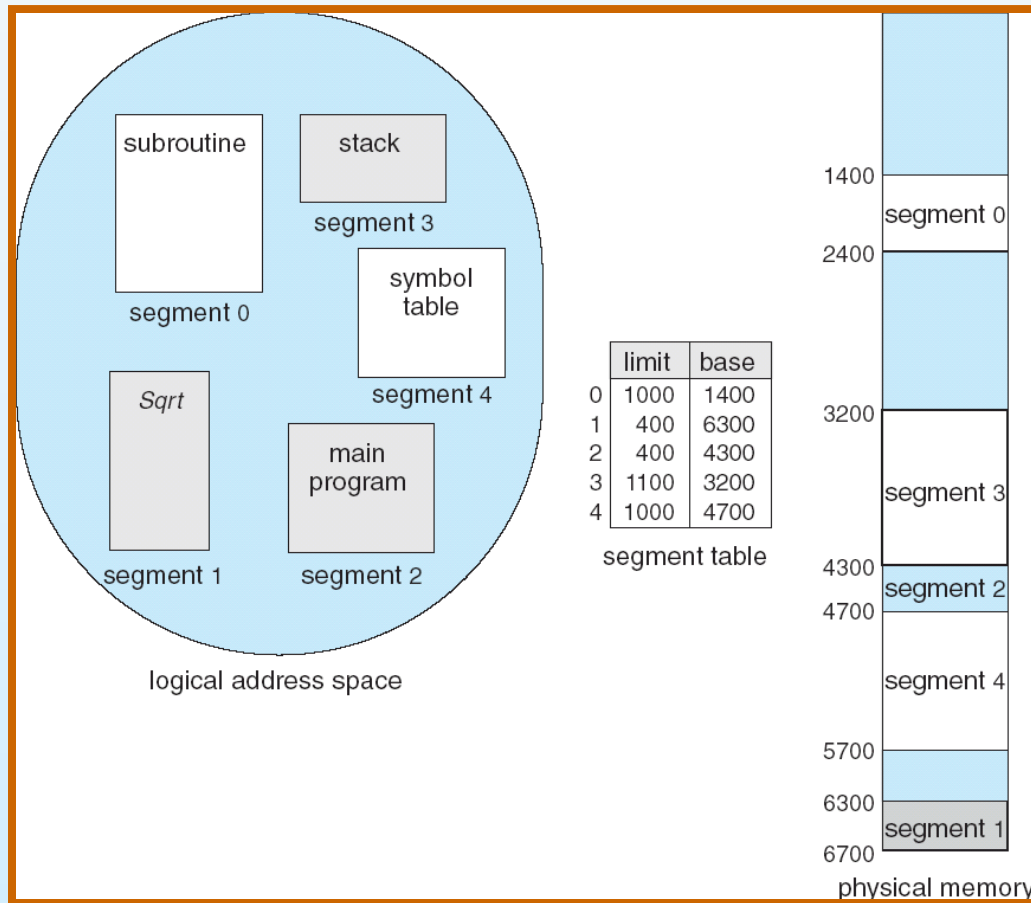


Address Translation Architecture





Example of Segmentation



End of Chapter 8

