



## Chapter 8: Memory Management

- Background
- Contiguous Memory Allocation
- Swapping
- Paging
- Structure of the Page Table
- Segmentation
- Example: The Intel Pentium



(slides selected/reordered by R. Doemer, 01/28/09)



## 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
- Internal fragmentation





## Address Translation Scheme

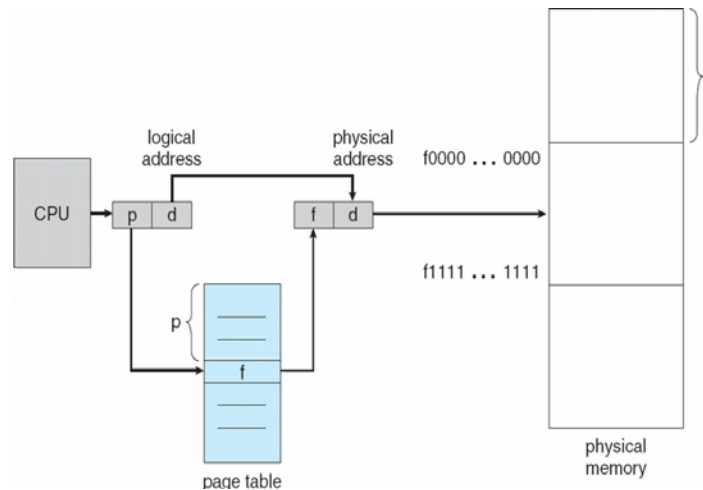
- Address generated by CPU is divided into:
  - **Page number ( $p$ )** – used as an index into a *page table* which contains base address of each page in physical memory
  - **Page offset ( $d$ )** – combined with base address to define the physical memory address that is sent to the memory unit

page number	page offset
$p$	$d$
$m - n$	$n$

- For given logical address space  $2^m$  and page size  $2^n$

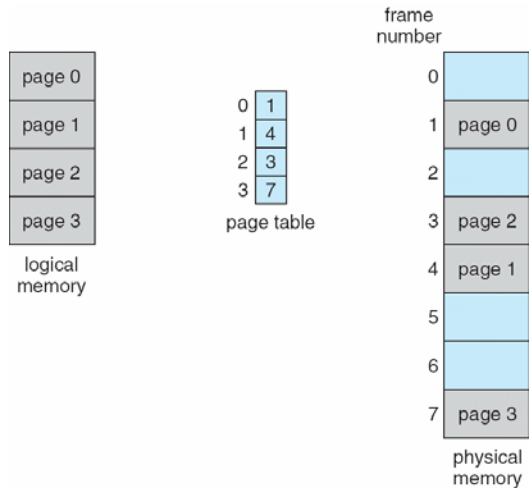


## Paging Hardware

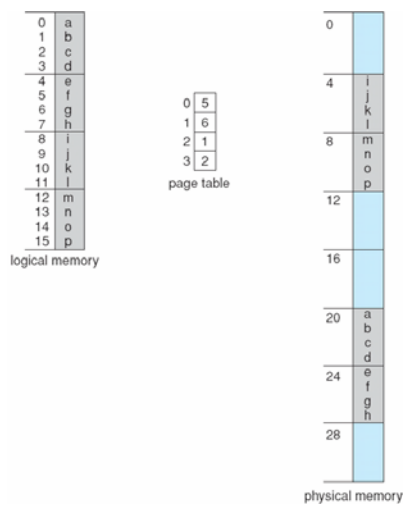




# Paging Model of Logical and Physical Memory



# Paging Example

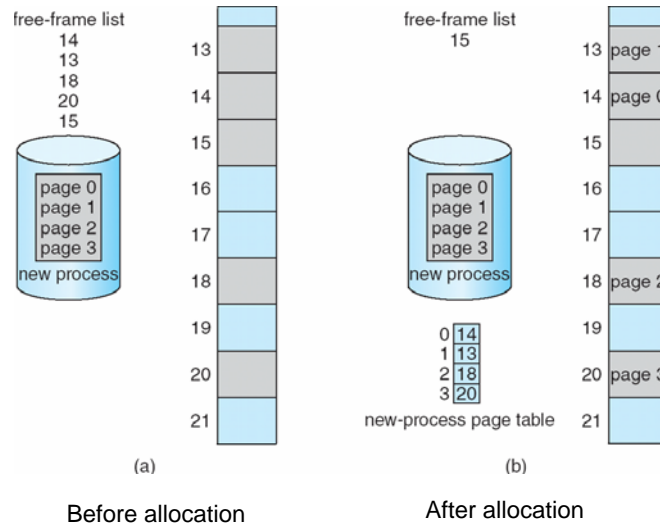


32-byte memory and 4-byte pages





## 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 (PRLR)** 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)**
- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process





## Associative Memory

- Associative memory – parallel search

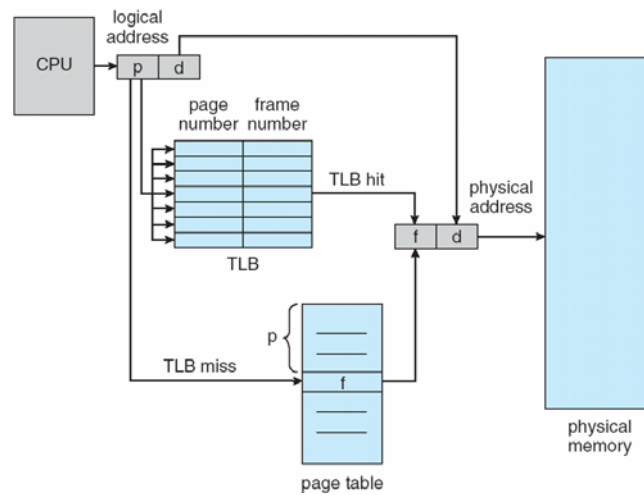
Page #	Frame #

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



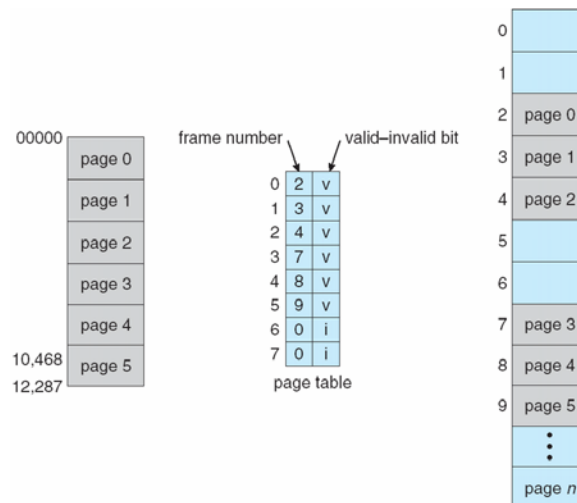


## Memory Protection

- 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



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





## Shared Pages

### ■ Shared code

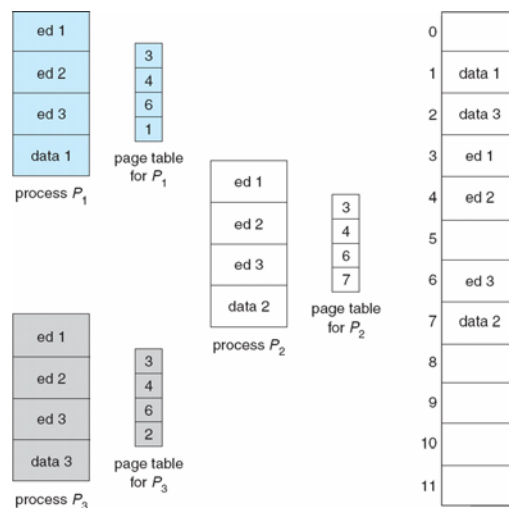
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

### ■ Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space



## Shared Pages Example





## 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

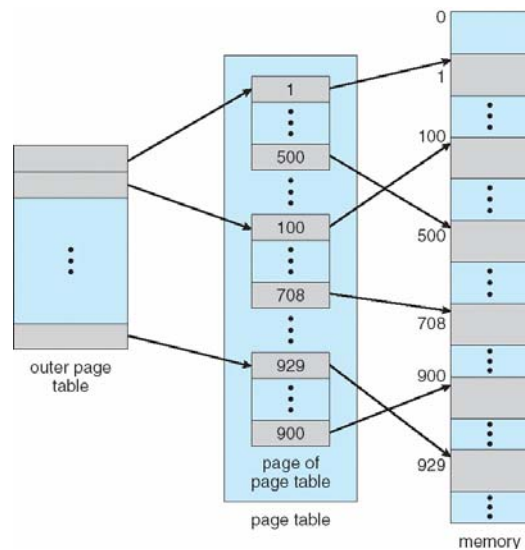
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address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

page number		page offset
$p_1$	$p_2$	$d$
12	10	10

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

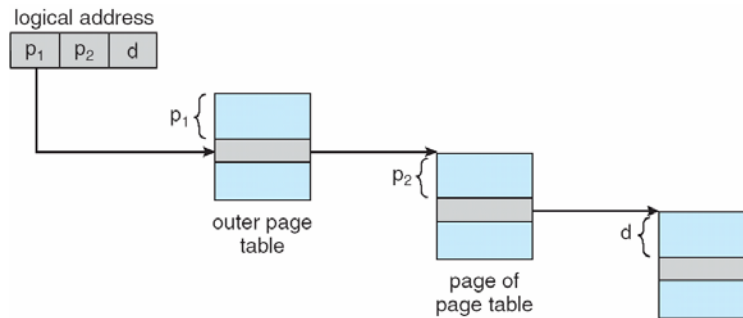


## Two-Level Page-Table Scheme





## Address-Translation Scheme



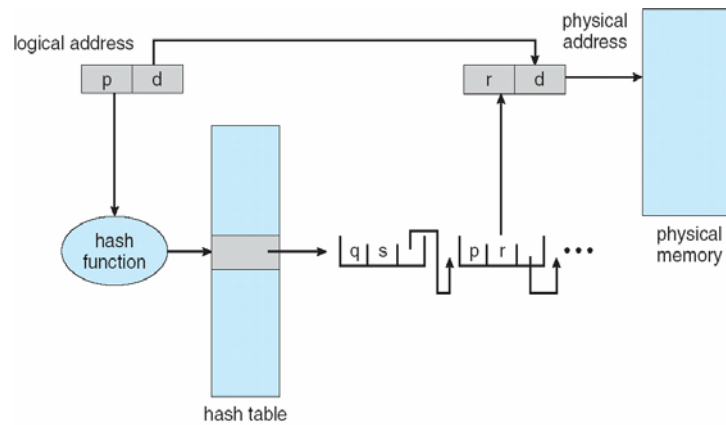
## Hashed Page Tables

- 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



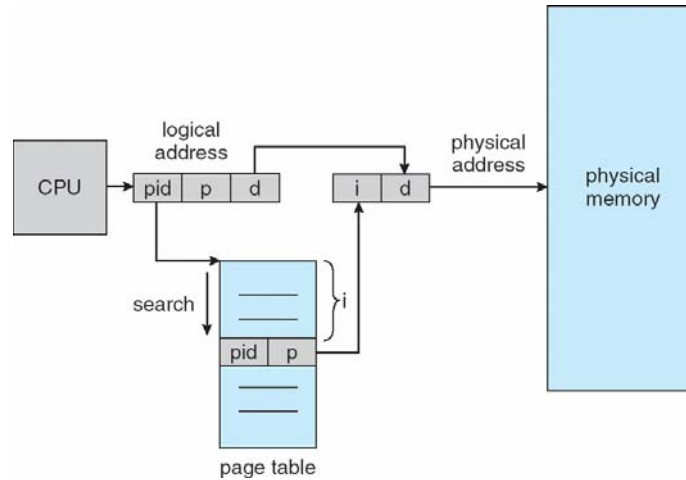
## Inverted Page Table

- 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



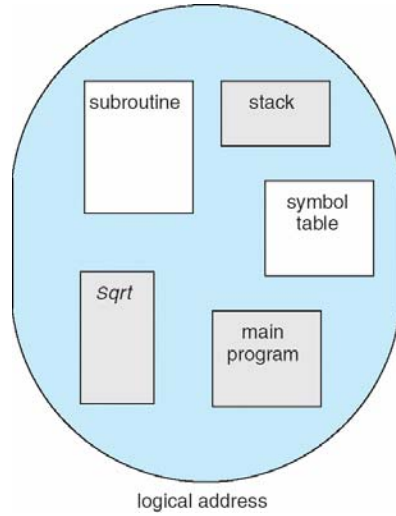
## Segmentation

- 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

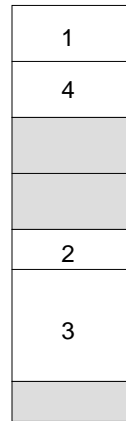
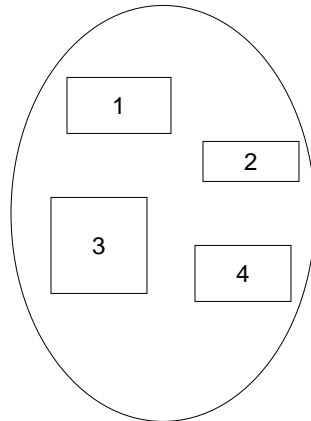




## User's View of a Program



## Logical View of Segmentation





## 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**



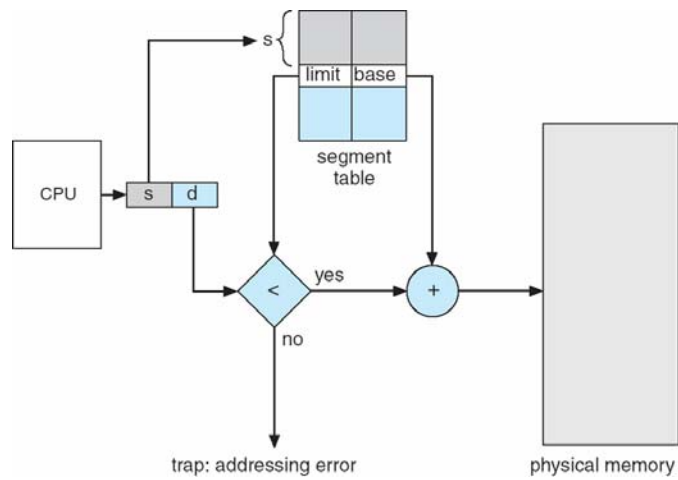
## Segmentation Architecture (Cont.)

- 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

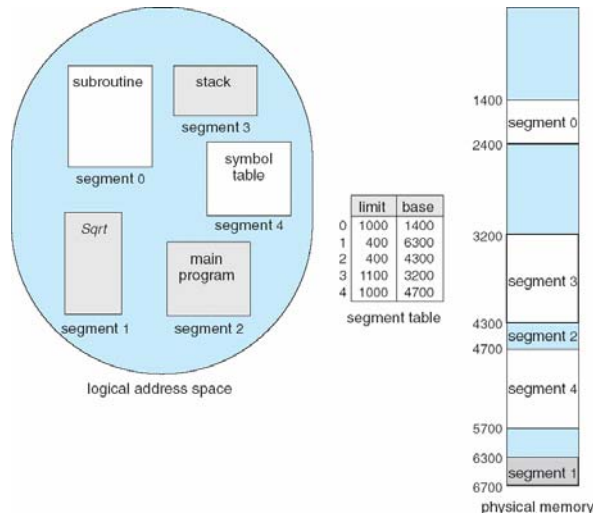




## Segmentation Hardware



## Example of Segmentation



# End of Chapter 8

