Main Memory

Contents of Lecture:

- ❖ Background
- ❖ Swapping
- ❖ Contiguous Memory Allocation
- ❖ Paging

References for This Lecture:

- ✓ *Abraham Silberschatz, Peter Bear Galvin and Greg Gagne, Operating System Concepts, 9th Edition, **Chapter 8***

Background

- ❖ Memory is central to the operation of a modern computer system. Memory consists of a large array of bytes, each with its own address.
- ❖ The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.

Basic Hardware

- ❖ Main memory and the registers built into the processor itself are the only general-purpose storage that the CPU can access directly.
  - ✓ Registers that are built into the CPU are generally accessible within one cycle of the CPU clock.
  - ✓ Main memory accessed via a transaction on the memory bus. Completing a memory access may take many cycles of the CPU clock. In such cases, the processor normally needs to stall, since it does not have the data required to complete the instruction that it is executing.

- ❖ Memory unit only sees a stream of addresses + read requests, or address + data and write requests.
- ❖ Cache sits between main memory and CPU registers.

- ❖ Each process must have a separate memory space (Separate per-process).This memory space protects the processes from each other.
- ❖ To separate memory spaces, need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
  - ✓ Can provide this protection by using two registers, usually a **base** and a **limit**, as illustrated innext figure (Figure 8.1).
    - ▪ The **base register** holds the **smallest** legal physical memory address;
    - ▪ the **limit register** specifies the **size** of the range.

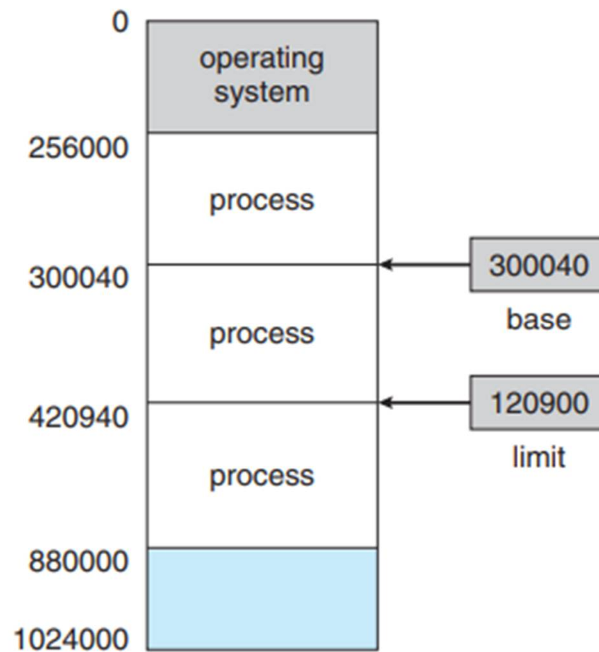


Figure 8.1 A base and a limit register define a logical address space.

- For example, if the base register holds 300040 and the limit register is 120900, then the program can legally access all addresses from 300040 through 420939 (inclusive).

- A pair of **base** and **limit** registers define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user

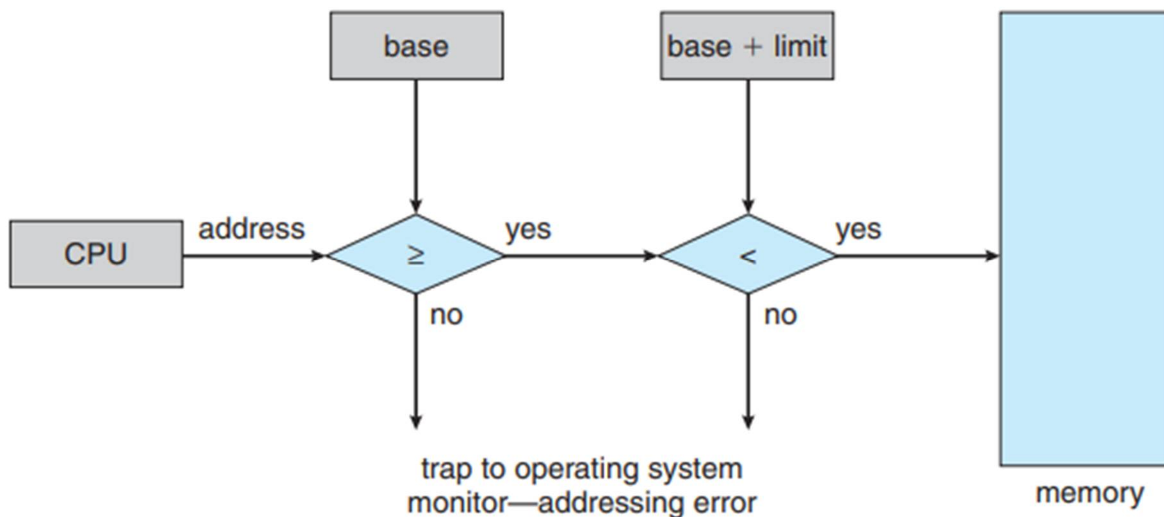


Figure 8.2 Hardware address protection with base and limit registers.

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 address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address
- To start, consider simple scheme where the value in the **relocation register** is added to every address generated by a user process at the time it is sent to memory
    - ✓ **Base register** now called **relocation register**

- The user program deals with logical addresses; it never sees the real physical addresses
    - ✓ Execution-time binding occurs when reference is made to location in memory
    - ✓ Logical address bound to physical addresses

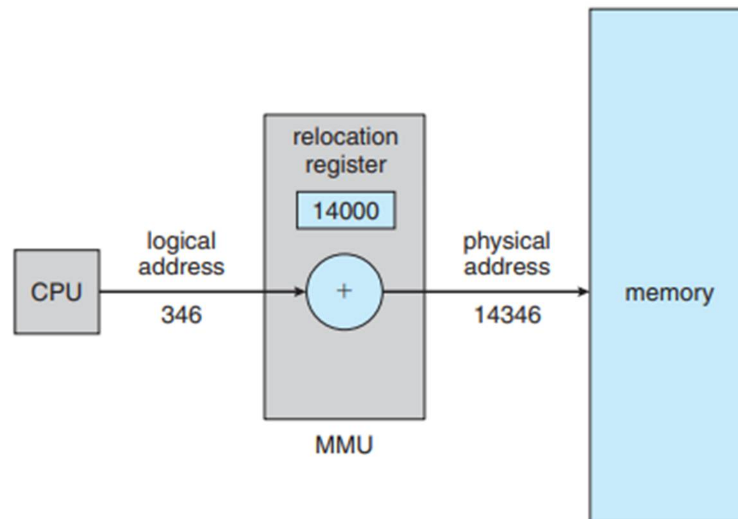


Figure 8.4 Dynamic relocation using a relocation register.

Swapping

- A process must be in memory to be executed. A process, can be swapped temporarily out of memory to a **backing store** and then brought back into memory for continued execution next figure (Figure 8.5).
    - ✓ Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system, thus increasing the degree of multiprogramming in a system.

- Standard swapping involves moving processes between main memory and a **backing store**.
- The **backing store** is commonly a fast disk.
    - It must be large enough to accommodate copies of all memory images for all users.
    - And it must provide direct access to these memory images.

- The system maintains a **ready queue** consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- **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

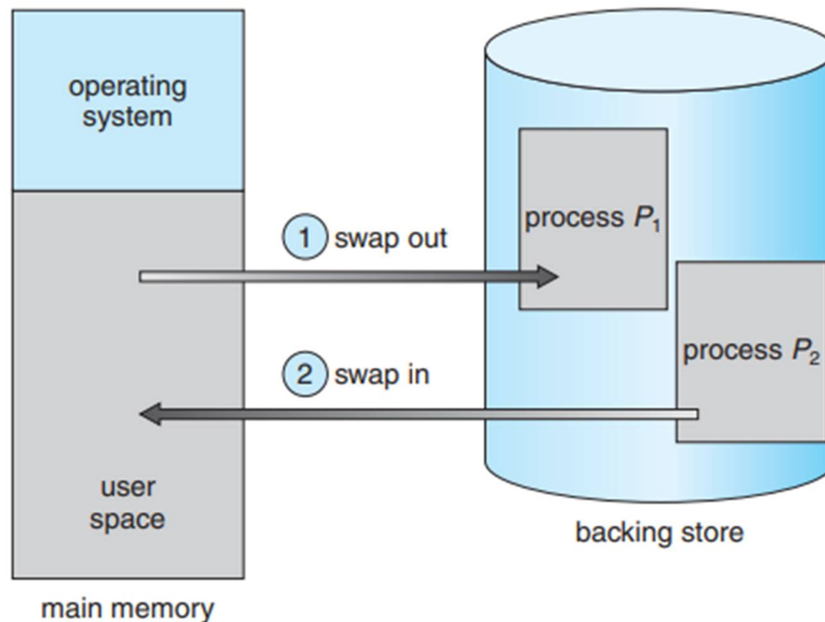


Figure 8.5 Swapping of two processes using a disk as a backing store.

Contiguous Memory Allocation

- The main memory must support both the operating system and the various user processes.
    - Limited resource, must allocate efficiently
- Main memory usually divided into two partitions:
    - Resident operating system, usually held in low memory
    - User processes held in high memory
- In contiguous memory allocation, each process is contained in a single section of memory that is contiguous to the section containing the next process.

Memory Protection

- 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

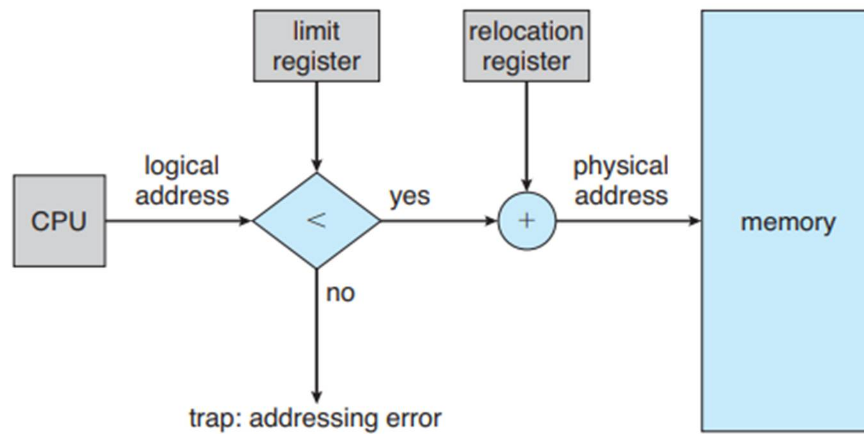
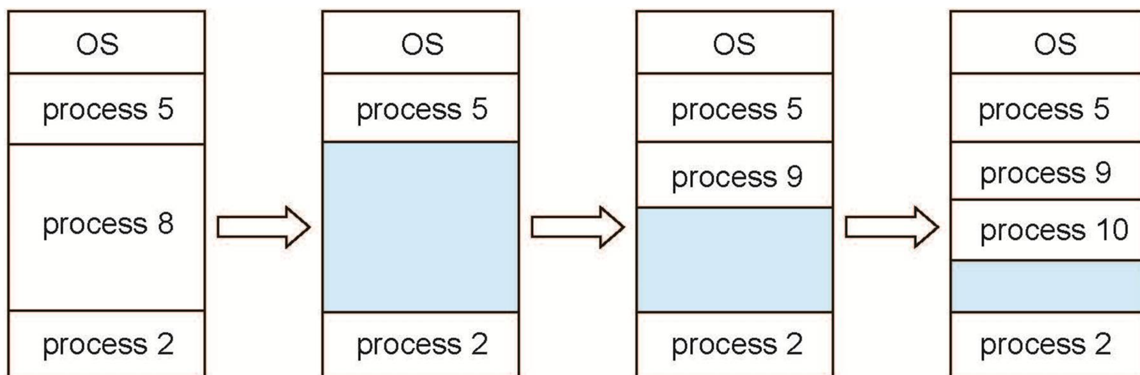


Figure 8.6 Hardware support for relocation and limit registers.

Multiple-partition allocation

- Degree of multiprogramming limited by number of partitions
- **Variable-partition sizes** for efficiency (sized to a given process' needs)
- **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
- Process exiting frees its partition, adjacent free partitions combined
- 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

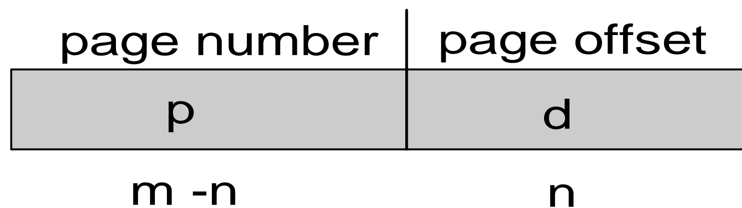
Paging

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever it is available

- Divide physical memory into fixed-sized blocks called **frames**
  - ✓ Size is power of 2, between 512 bytes and 16 Mbytes
- 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 addresses to physical addresses

- 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



For given logical address space 2^{m} and page size 2^{n}

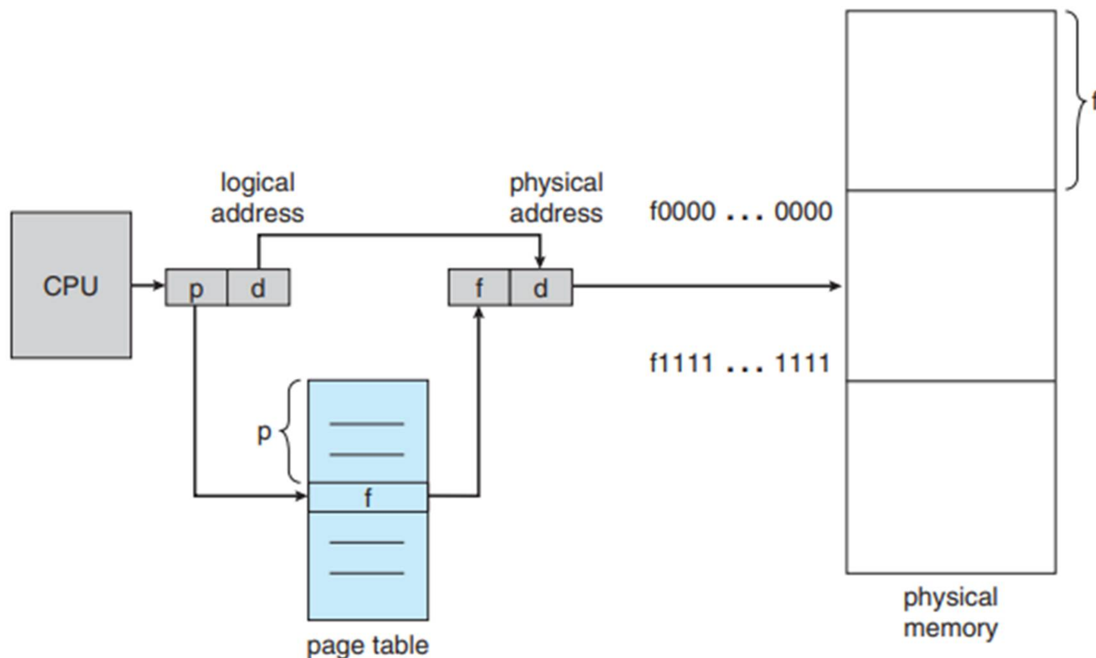


Figure 8.10 Paging hardware.

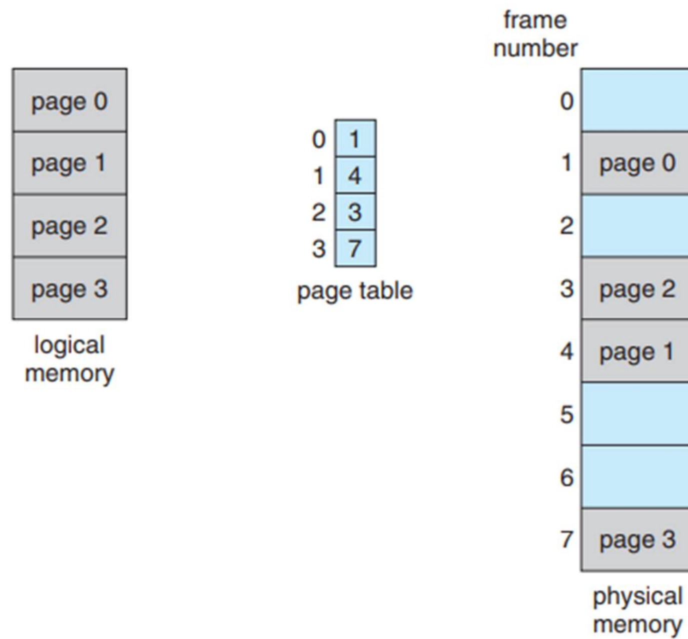


Figure 8.11 Paging model of logical and physical memory.

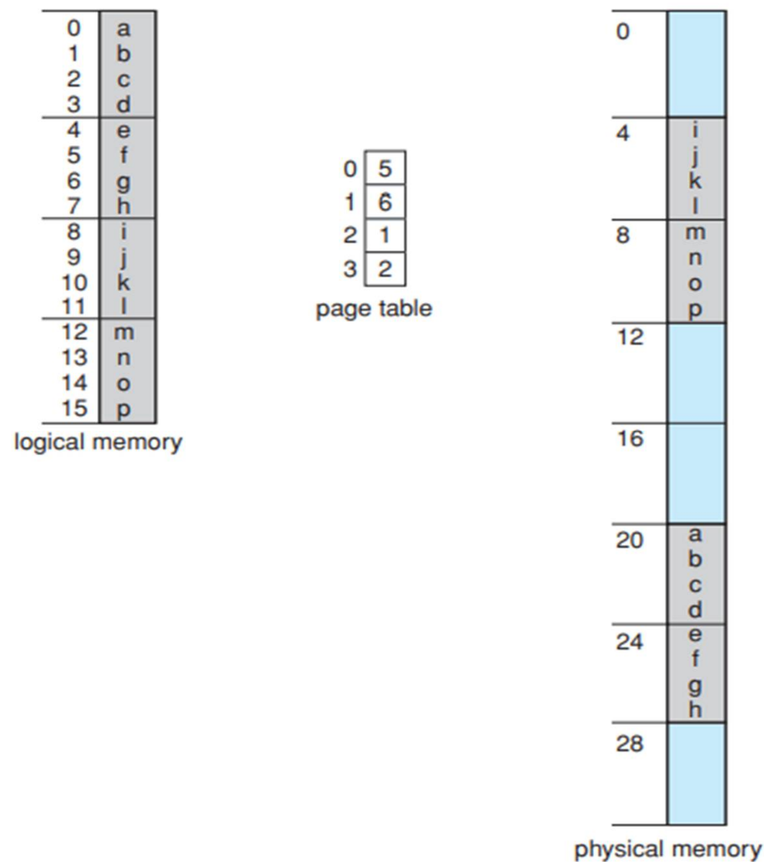


Figure 8.12 Paging example for a 32-byte memory with 4-byte pages.

❖ When use a paging scheme, there is no external fragmentation: any free frame can be allocated to a process that needs it. However, may have some internal fragmentation.

❖ **<u>For example:</u>**

- ✓ If page size is 2,048 bytes and Process size = 72,766 bytes
- ✓ Then process of 72,766 bytes will need 35 pages plus 1,086 bytes.
  - It will be allocated 36 frames
  - Resulting in internal fragmentation of 2,048 − 1,086 = 962 bytes.

- ✓ In the worst case, a process would need n pages plus 1 byte.
  - It would be allocated n + 1 frames, resulting in internal fragmentation of almost an entire frame.

- ✓ If process size is independent of page size, expect internal fragmentation to average one-half page per process.

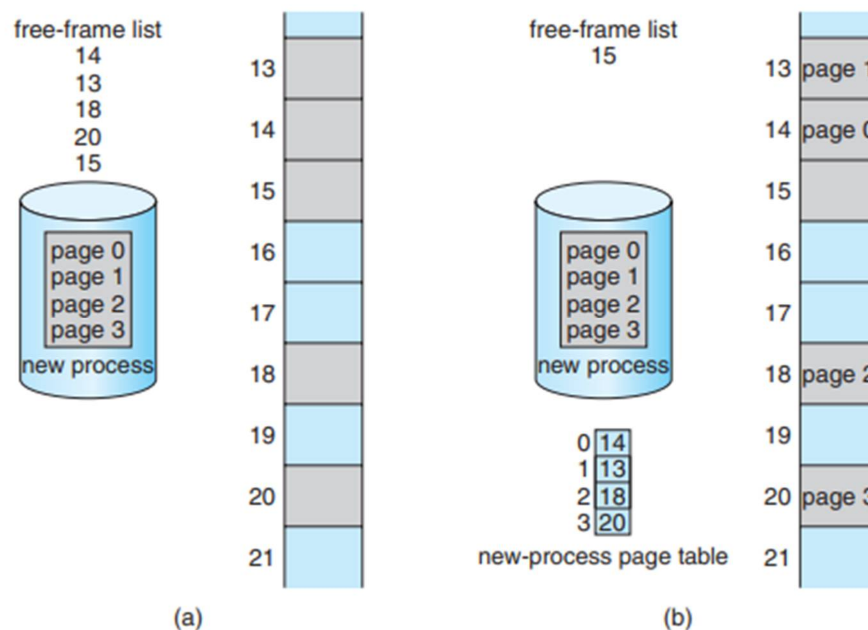


Figure 8.13 Free frames (a) before allocation and (b) after allocation.

<u>Memory Protection</u>

❖ Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed

- ✓ Can also add more bits to indicate page execute-only, and so on

❖ **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

❖ Any violations result in a trap to the kernel

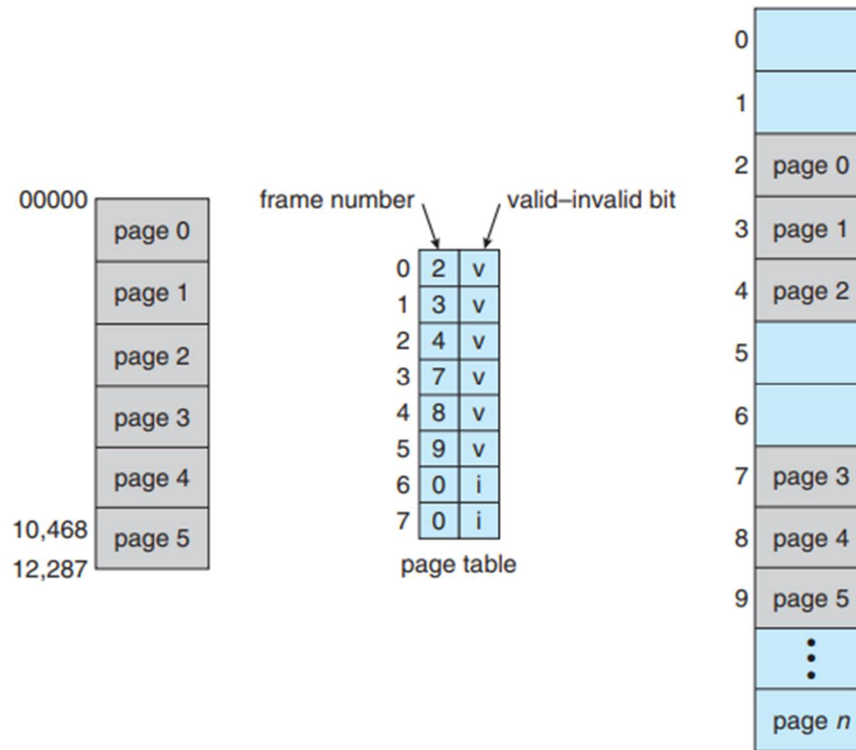


Figure 8.15 Valid (v) or invalid (i) bit in a page table.

Structure of the Page Table

- ❖ Memory structures for paging can get huge using straight-forward methods
  - ✓ Consider a 32-bit logical address space as on modern computers
  - ✓ Page size of 4 KB (2^{12})
  - ✓ Page table would have 1 million entries (2^{32} / 2^{12})
  - ✓ If each entry is 4 bytes -> 4 MB of physical address space for page table alone
    - ▪ That amount of memory used to cost a lot
    - ▪ Don’t want to allocate that contiguously in main memory

- ❖ **Structure of the Page Table**
  1. Hierarchical Paging
  2. Hashed Page Tables
  3. Inverted Page Tables

Hierarchical Paging

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

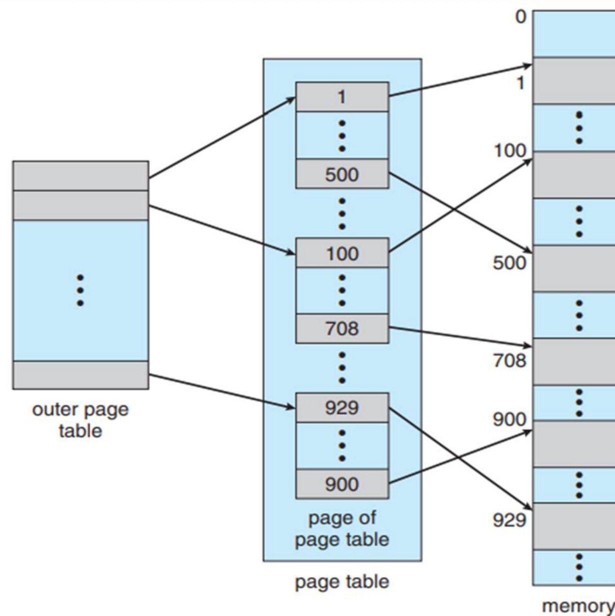
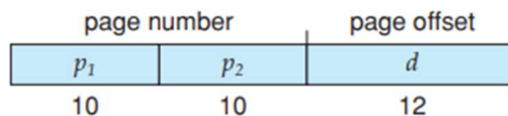


Figure 8.17 A two-level page-table scheme.

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 p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table

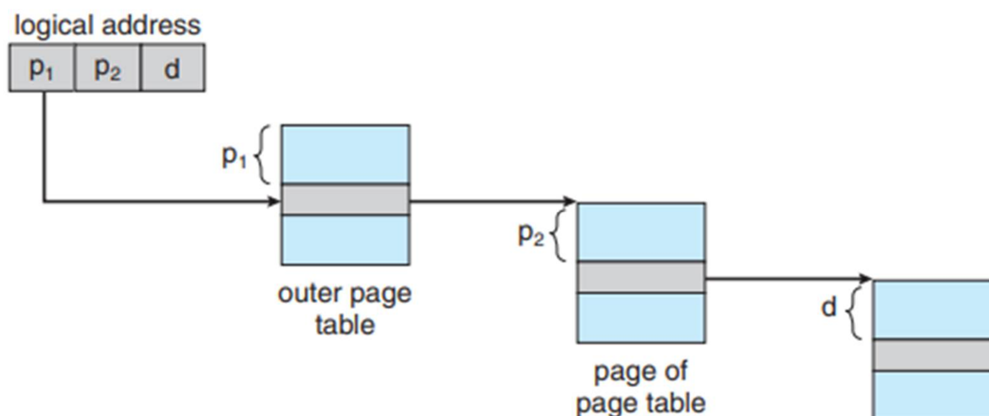


Figure 8.18 Address translation for a two-level 32-bit paging architecture.