Chapter 8: Main Memory

Chapter 8: Memory Management

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

Objectives

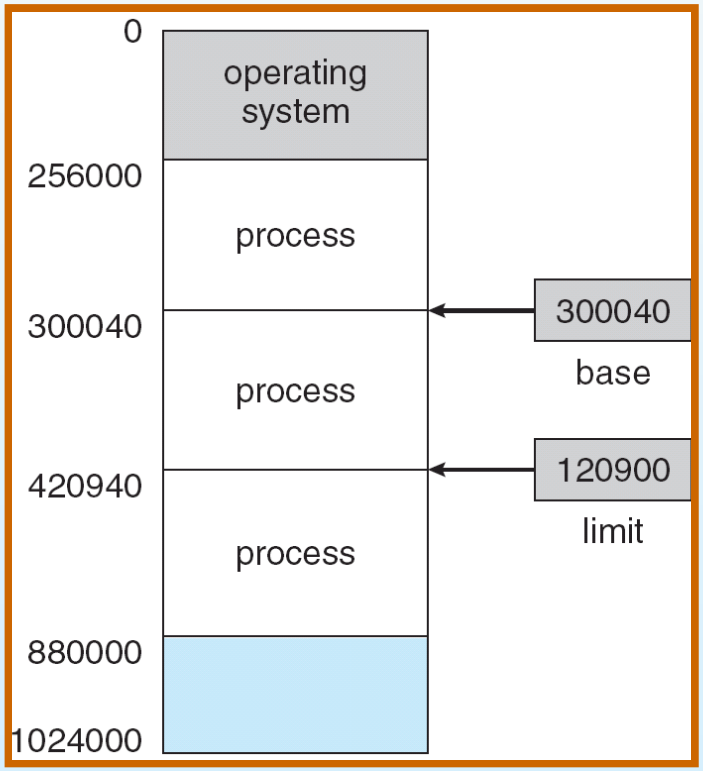
- Know how to organize memory
- Know memory-management techniques
  - Paging
  - Segmentation
- Know the Intel Pentium, which supports
  - Pure segmentation
  - Pure paging
  - Segmentation with paging

Background

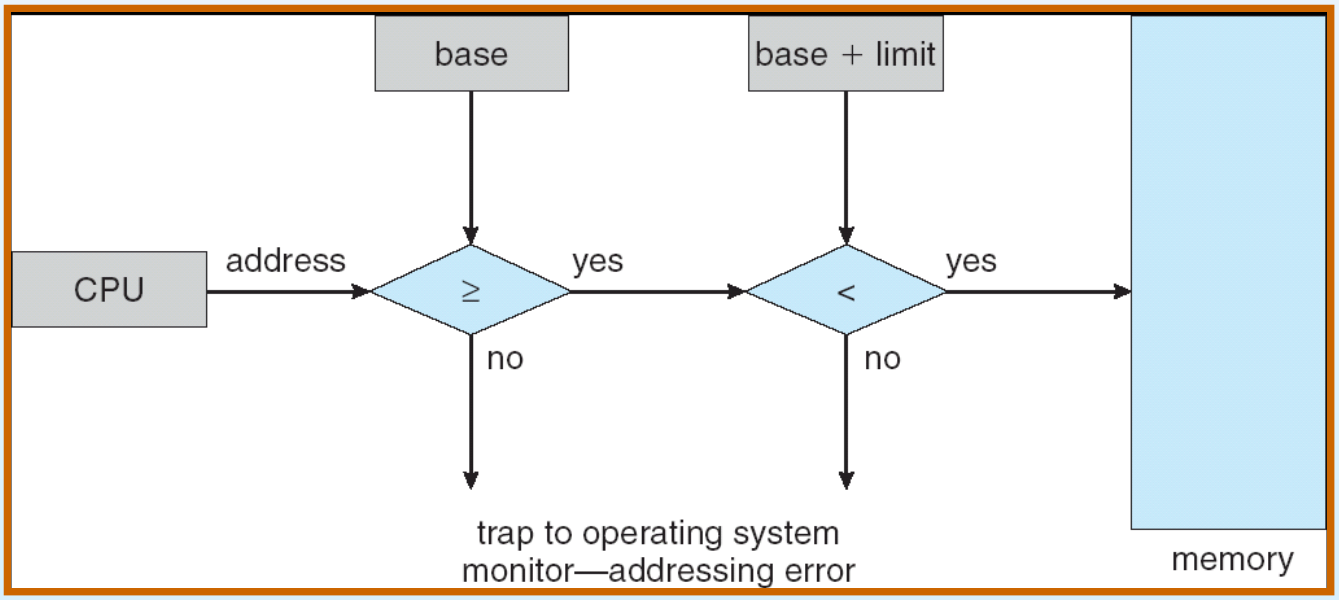
- Before running, a program must be
  - Brought (from disk) into memory
  - Placed within a process
- The only storage that CPU can access directly
  - Main memory
  - Registers
- Access time
  - Register: one CPU clock (or less)
  - Main memory: many cycles
  - So, cache comes!
- The way to ensure correct operation
  - Protection of memory (by hardware)

Base and Limit Registers

- Logical address space defined
  - By base and limit registers
- Only OS can set base and limit
  - Privileged Instructions
- Verify *every* address generated in user mode
- If bad address found , trap into kernel
- Can kernel access user code and data?



HW address protection with base and limit registers



Binding of Instructions and Data to Memory

- Address binding happens at
  - Compile time
    - If memory location known, generate absolute code
    - Must recompile code if starting location changes
  - Load time
    - At compile time, memory location is unknown
    - Generate relocatable code
  - Execution time
    - Process can be moved from one memory segment to another.
    - Need hardware support (e.g., base and limit registers)

Logical vs. Physical Address Space

- Logical address
  - Generated by the CPU
  - Also referred to as virtual address
- Physical address
  - Address seen by the memory unit

- They are the same in
  - Compile-time address-binding
  - Load-time address-binding
- They are different in
  - Execution-time address-binding
  - Translated by MMU

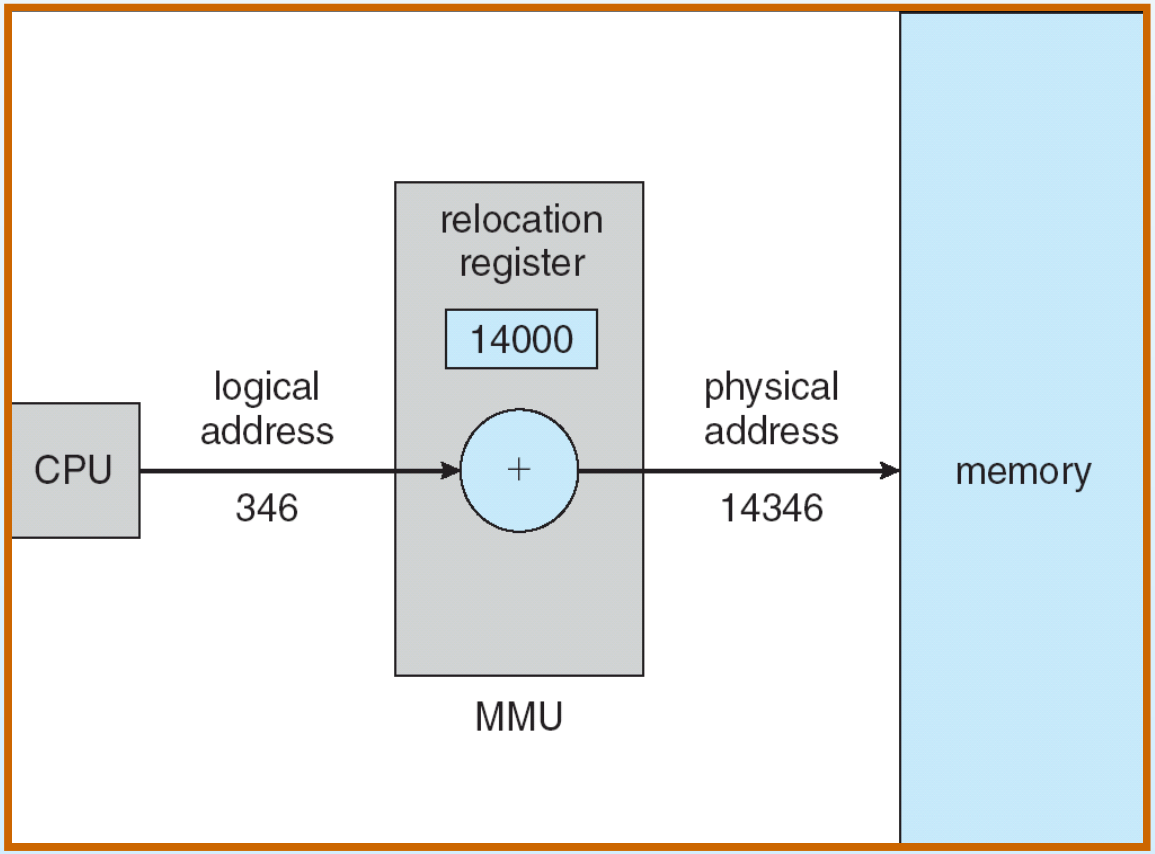
Memory-Management Unit (MMU)

- Hardware device that maps logical to physical address

- In MMU scheme
  - Physical address
    = relocation register + logical address
  - MS-DOS works in this way

- The user program
  - Deals with *logical* addresses
  - 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
- Implemented through program design
- No special support from the OS

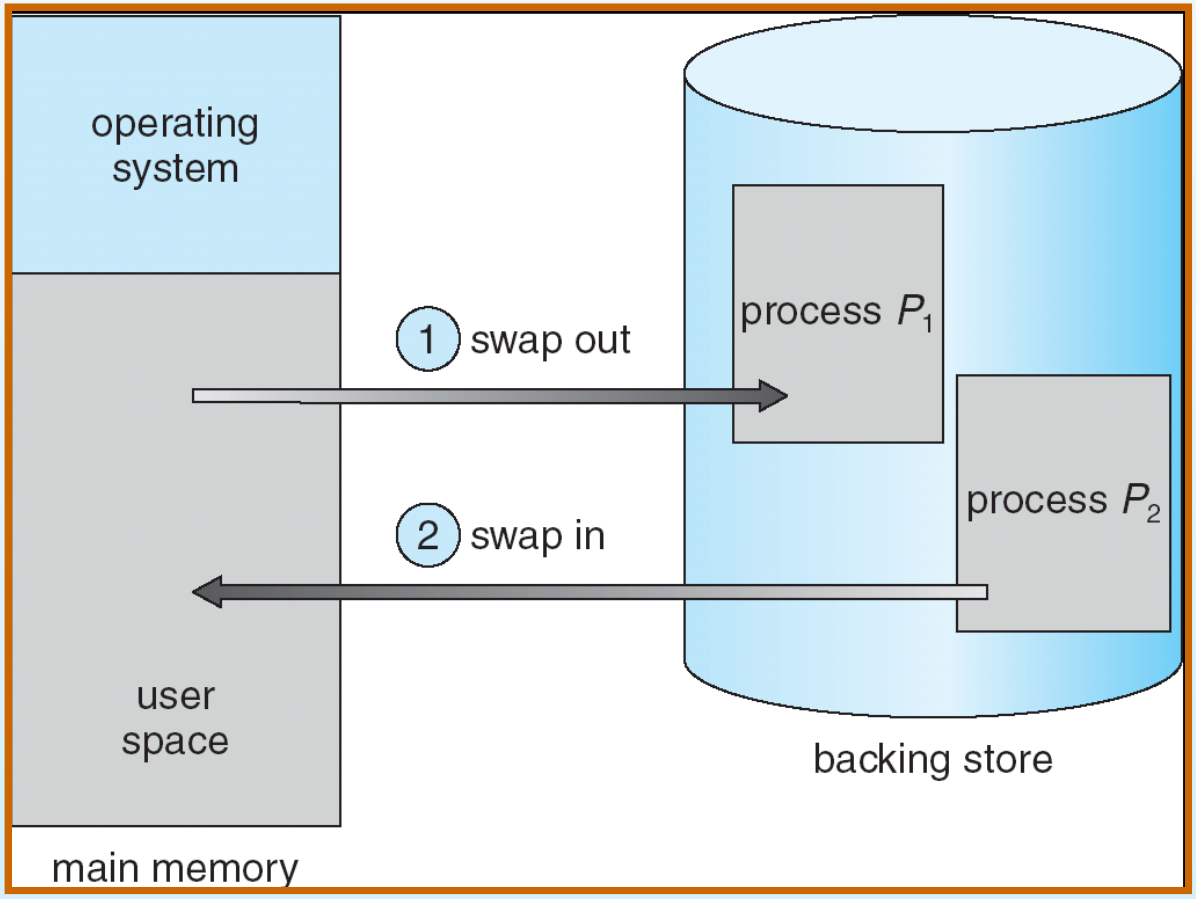
Dynamic Linking and Shared Libraries

- Linking postponed until execution time
  - Useful for system libraries, language libraries
  - All processes execute only one copy of the routine
  - Easy to update
    - Version must be controlled
- Stub
  - Used to locate the appropriate library routine
  - Replaces itself with the address of the routine
  - Executes the routine
- OS maps the routine into processes' memory address

Swapping

- A process can be
  - Swapped temporarily out to a backing store
  - Brought back for continued execution
- Logical space relocation is needed
- Swap out, swap in
  - Use priority-based scheduling algorithms
- Overhead
  - Transfer time is fairly high
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

Schematic View of Swapping

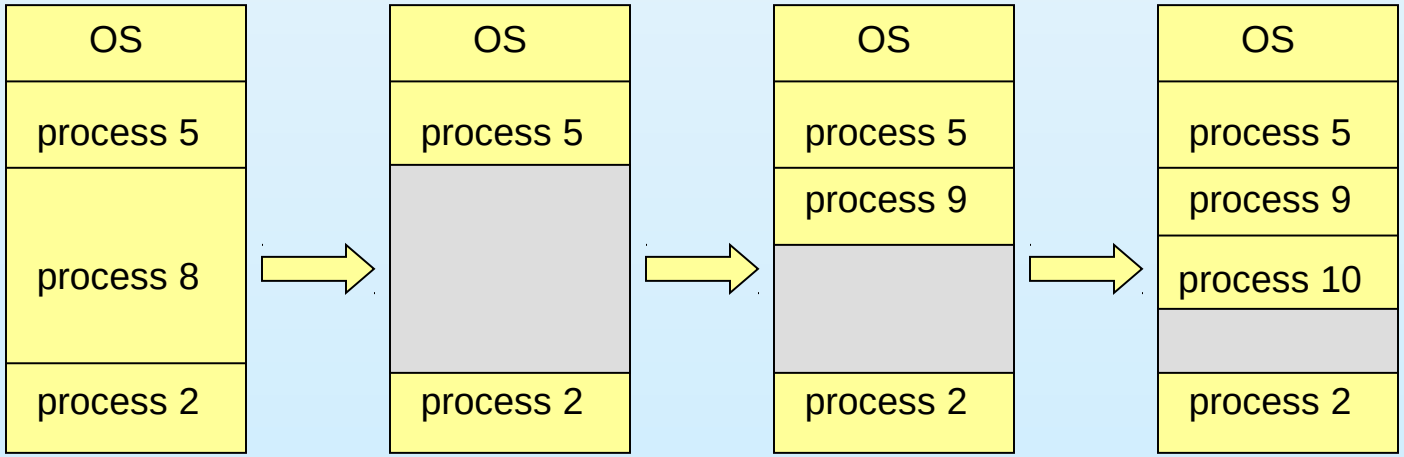


Contiguous Allocation

- Main memory includes two partitions:
  - OS kernel
    - Usually in low memory
  - User processes
    - Usually in high memory
- Relocation registers used to do protection
  - Base register contains the smallest physical address
  - Limit register contains range of logical addresses
  - Logical address must be less than the limit
  - MMU maps logical address *dynamically*

Contiguous Allocation (Cont.)

- **Multiple-partition allocation**
  - **Hole – block of available memory**
  - **A process arrives**
    - **Allocated memory from a hole large enough**
  - **OS maintains information about:**
    a) allocated partitions b) free partitions (hole)



Dynamic Storage-Allocation Problem

- First-fit
  - Allocate the *first* hole that is big enough
- Next-fit
  - First-fit begins from previous allocation point
- Best-fit
  - Allocate the *smallest* hole that is big enough
  - Produces the smallest leftover hole
  - Must search entire list, unless ordered by size
- Worst-fit
  - Allocate the *largest* hole
  - Produces the largest leftover hole
  - Must also search entire list

Fragmentation

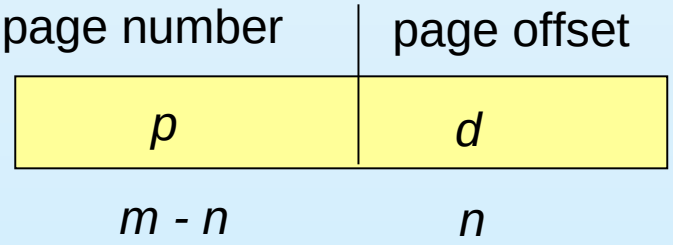
- External Fragmentation
  - Total free space satisfies a request
  - But it is not contiguous
- Internal Fragmentation
  - Allocated memory based on block size
  - Some waste space in the last block
- Reduce external fragmentation by compaction
  - Place all free memory together
  - *Only* if relocation is dynamic
  - I/O problem (DMA)
    - Don't move I/Oing processes
    - Do I/O only into OS buffers

Paging

- Logical address space can be noncontiguous
- Divide physical memory into fixed-sized frames
  - Size is power of 2, between 512 bytes and 16 MB
- Divide logical memory into fixed-sized pages
  - The same as frames
- Keep track of all free frames
  - Run a program of *n* pages, need to find *n* free frames
- Page table
  - Used to translate logical to physical addresses
- Internal fragmentation

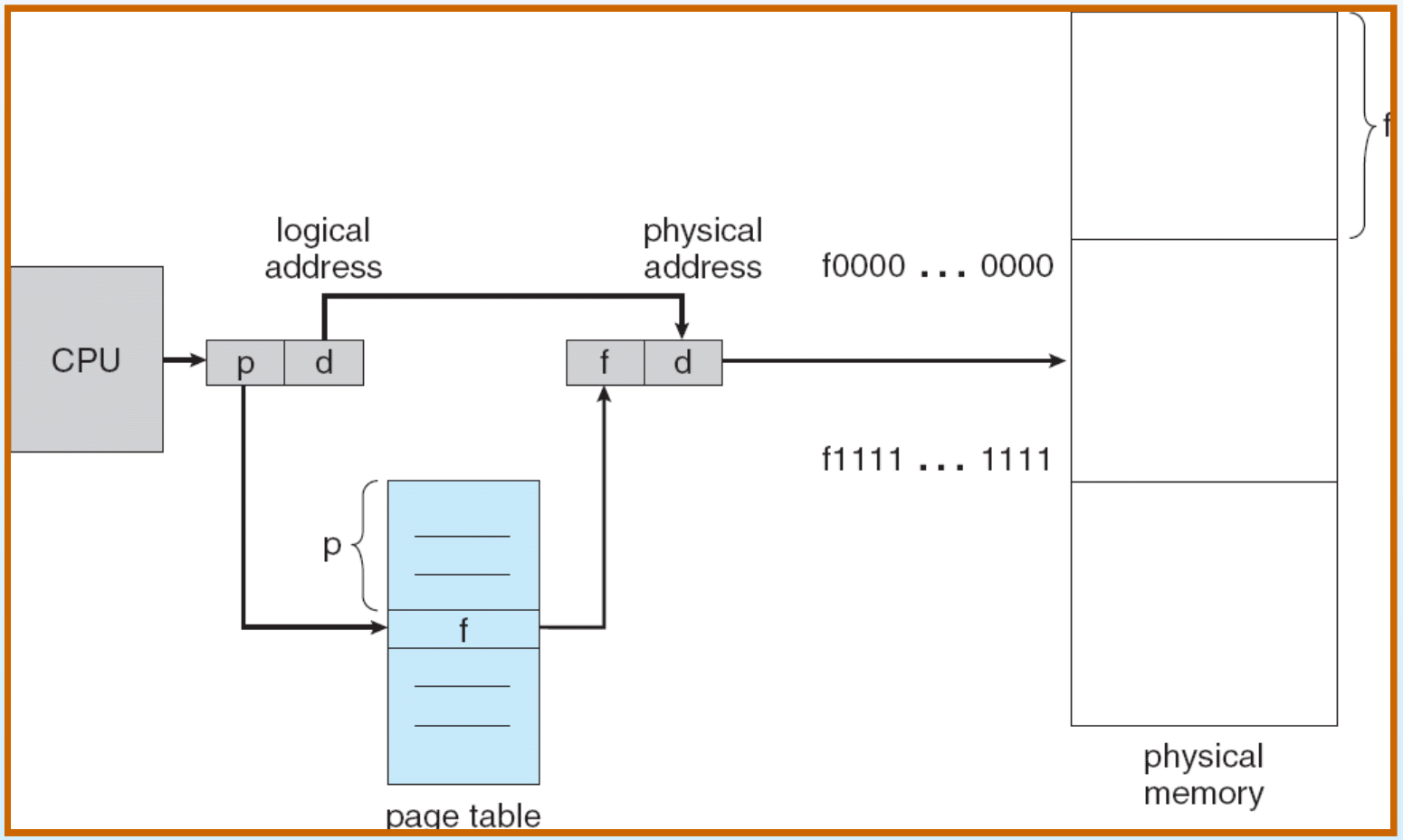
Address Translation Scheme

- Logical address is divided into:
  - Page number (*p*)
    - An index into a *page table*
    - *Page table* contains physical address of each page
  - Page offset (d)
    - Combined with base address to define the physical memory address

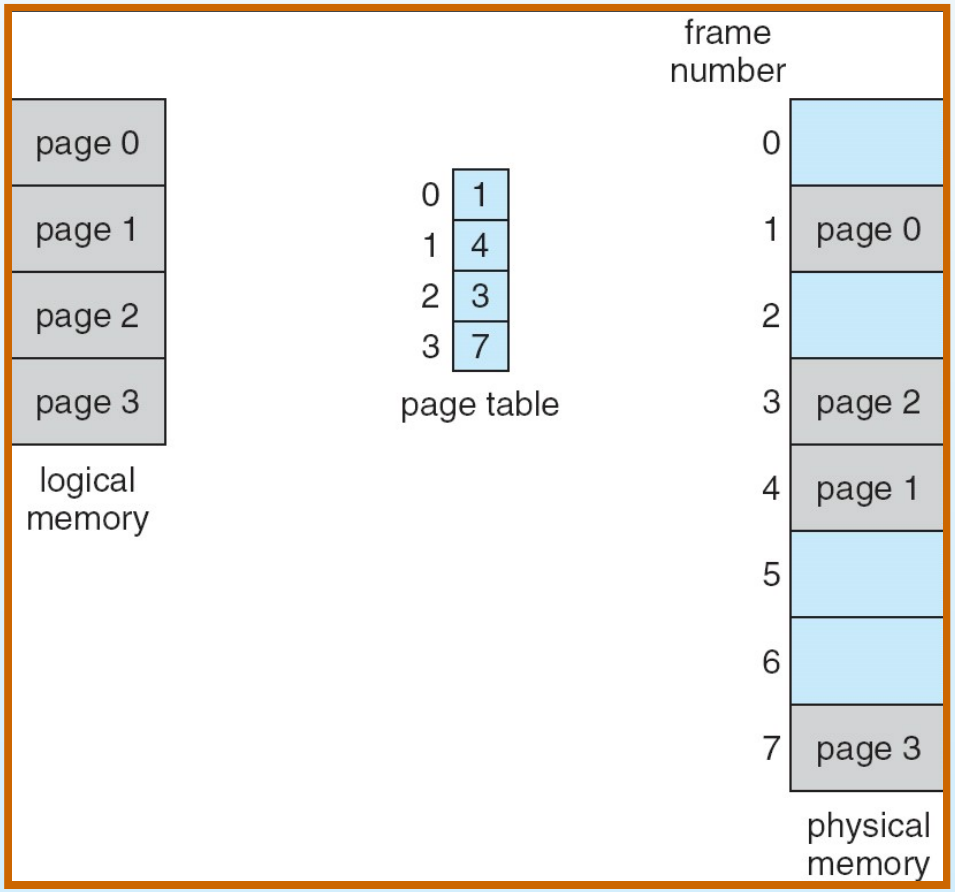


For given logical address space 2^m *and page size* 2^n

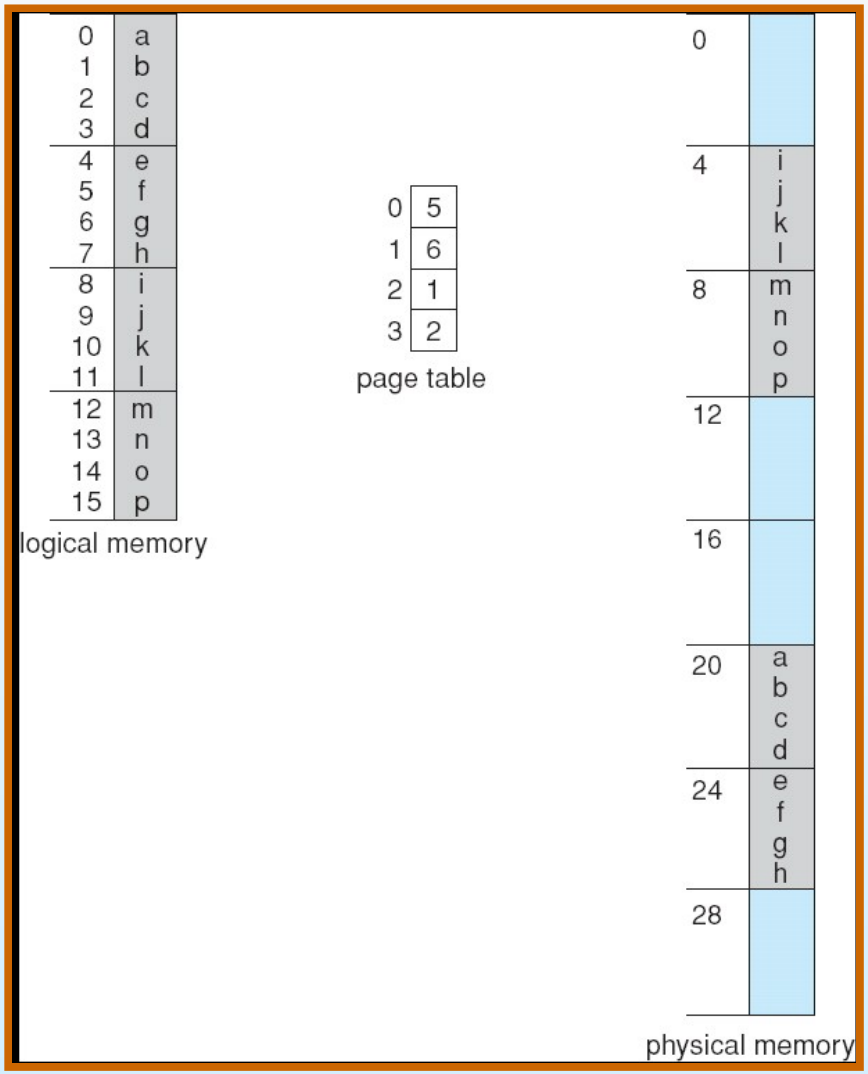
Paging Hardware



Paging Model of Logical and Physical Memory



Paging Example

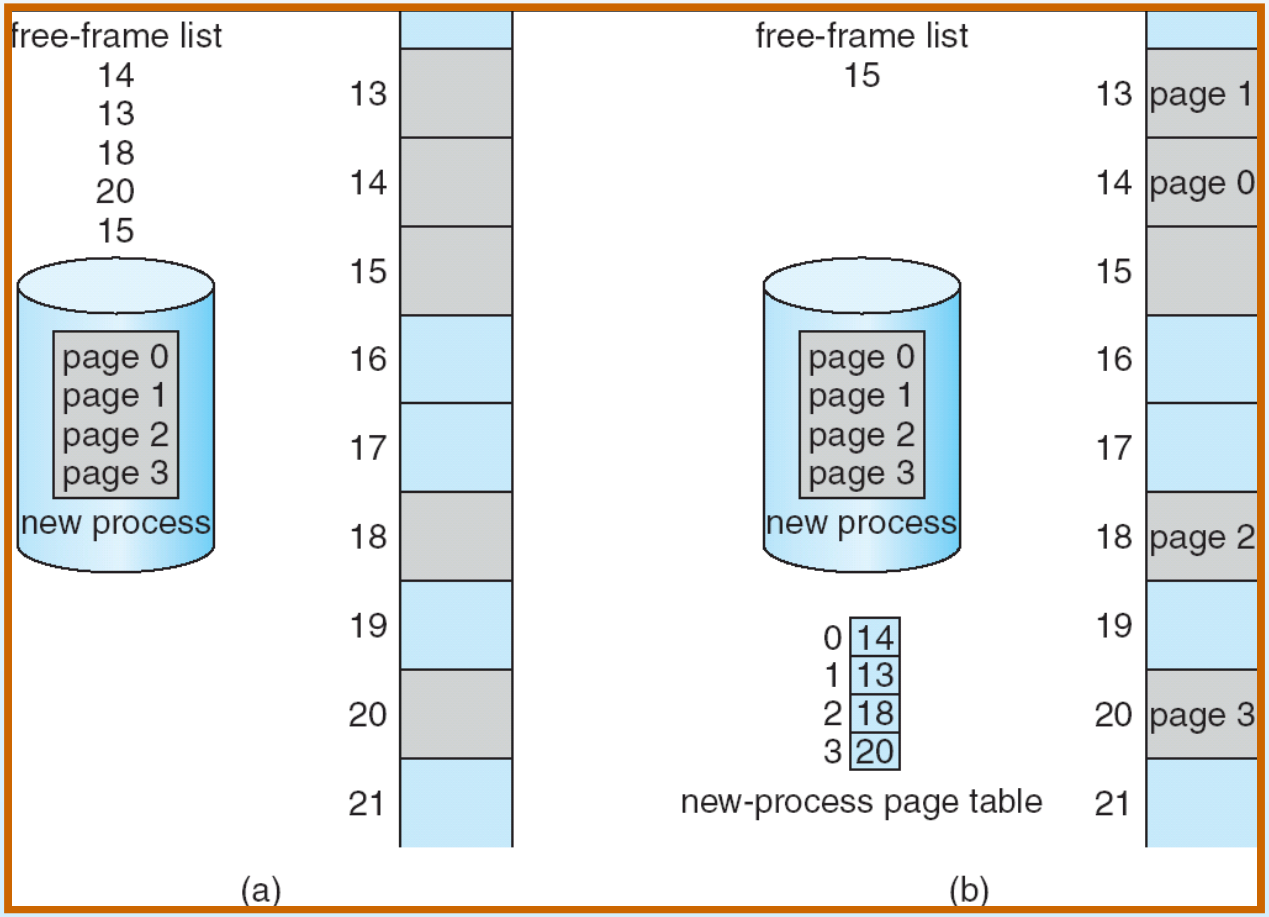


32-byte memory and 4-byte pages

Page Size

- Small page size
  - Decrease internal fragmentation
  - Increase page table size
- Big page size
  - Increase internal fragmentation
  - Decrease page table size
- Usually 4 KB or 8 KB
- Solaris in SPARC cpu supports
  - 8 KB and 4 MB
- Linux 2.6.x support 4KB or 4MB
  - Run “getconf PAGESIZE”

Free Frames



Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory
  - Page-table base register (PTBR) points to it
  - Page-table length register (PRLR) indicates size
- Every data/instruction access requires two memory accesses.
  - For page table, and for data/instruction.
- Solve it by using TLBs
  - Translation look-aside buffers
  - A special fast-lookup hardware cache

TLB

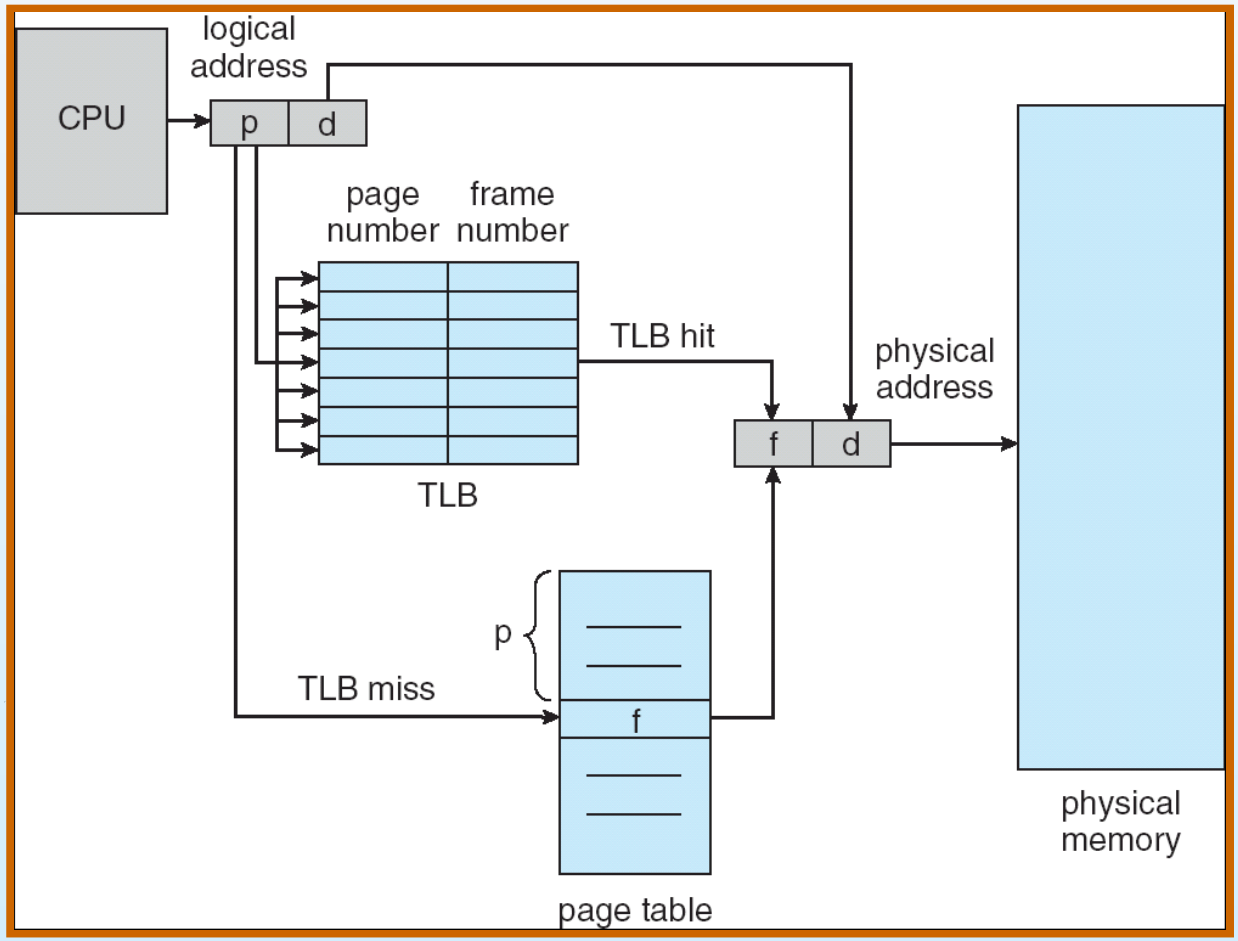
- Parallel search

Page #	Frame #

Address translation (p, d)

- If p is in, get frame # out
- Otherwise
  - Load entry of page p from page table into TLB
  - Get frame #

Paging Hardware With TLB



Effective Access Time

- Assume memory cycle time is 1 time unit
- Associative Lookup = ε time unit
- Hit ratio
  - Percentage of times that a page number is found in TLB
  - Ratio related to size of TLB
- Hit ratio = α
- Effective Access Time (EAT)

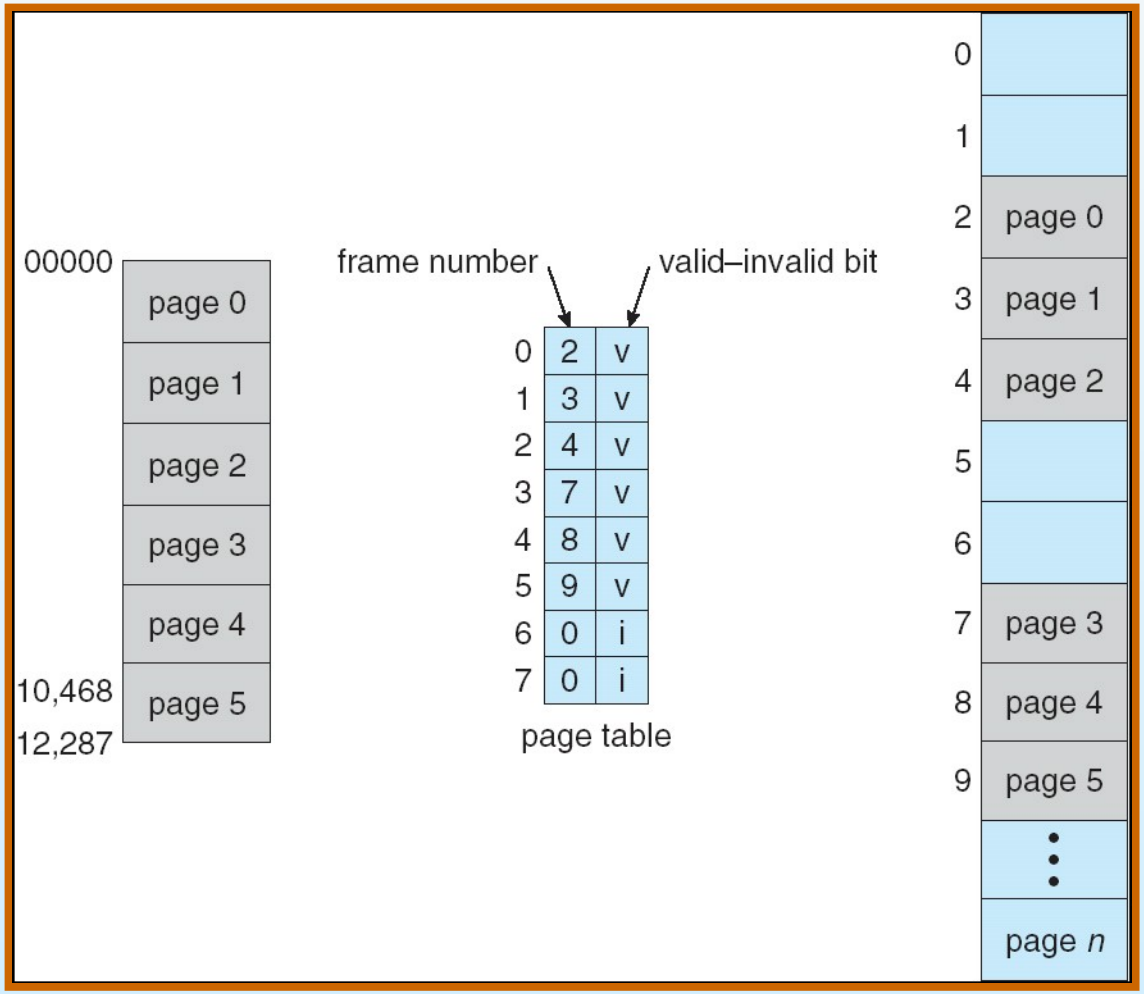
$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$

$$= 2 + \varepsilon - \alpha$$

Memory Protection

- Memory protection implemented by
  - Associating protection bit with each frame
  - Read-write, read-only, execution-only
- Valid-invalid bit attached to each entry in the page table:
  - Valid
    - The page is in the process' logical address space
  - Invalid
    - The page is NOT in the process' logical address space

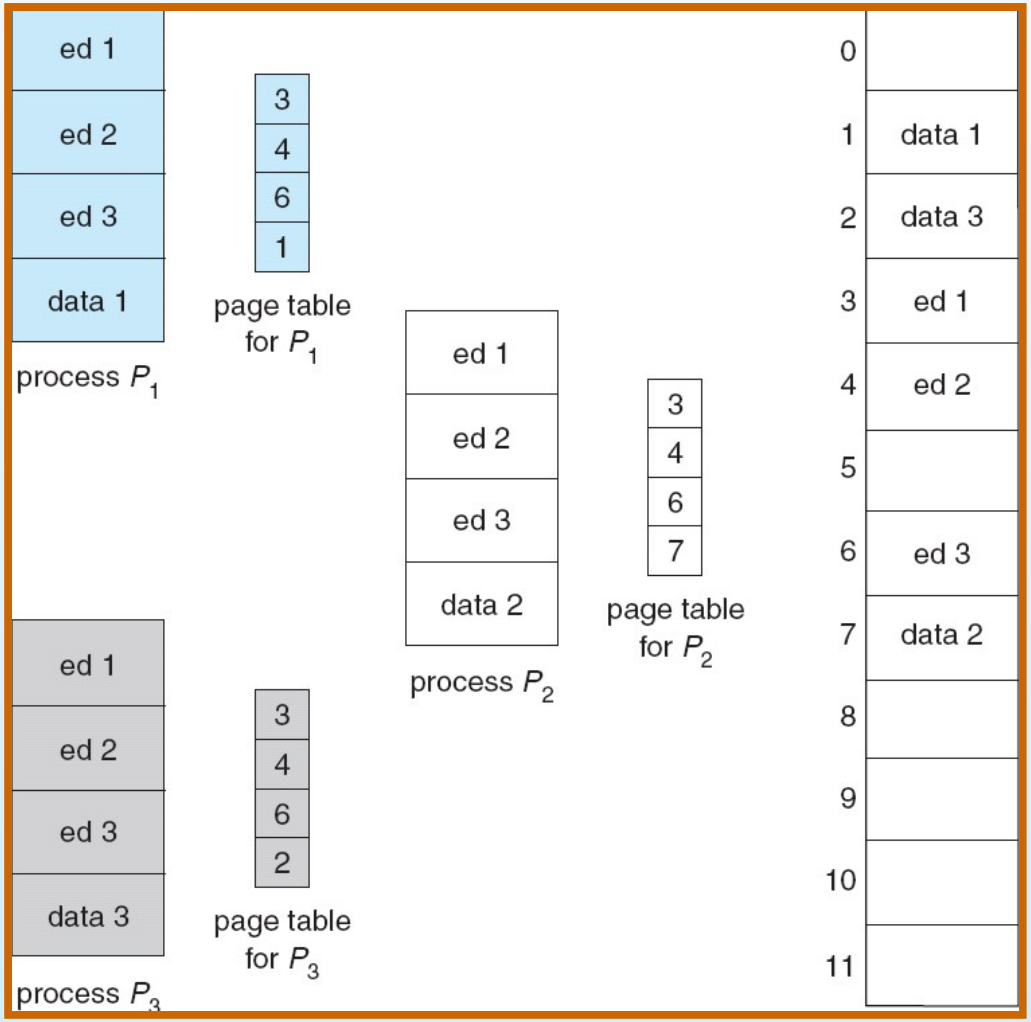
Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

- Shared code
  - One copy of read-only (reentrant) code shared among processes
    - i.e. DLL, Multi-process program
- Private code and data
  - Each process keeps a separate copy of the code and data

Shared Pages Example



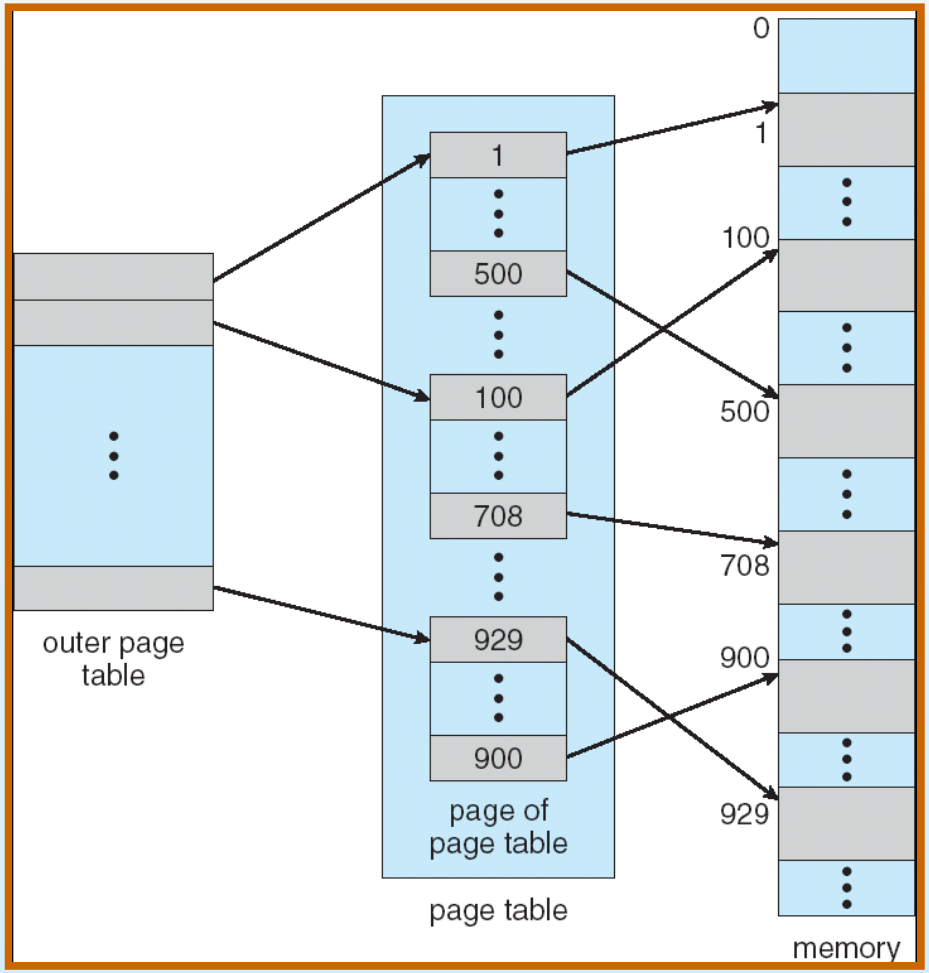
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

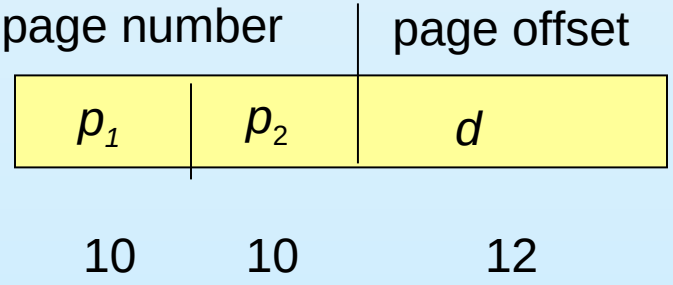
- Break up the logical address space into multiple page tables
  - Adding new tables as requesting new space
- A simple technique is a two-level page table
  - Widely used in IA32

Two-Level Page-Table Scheme

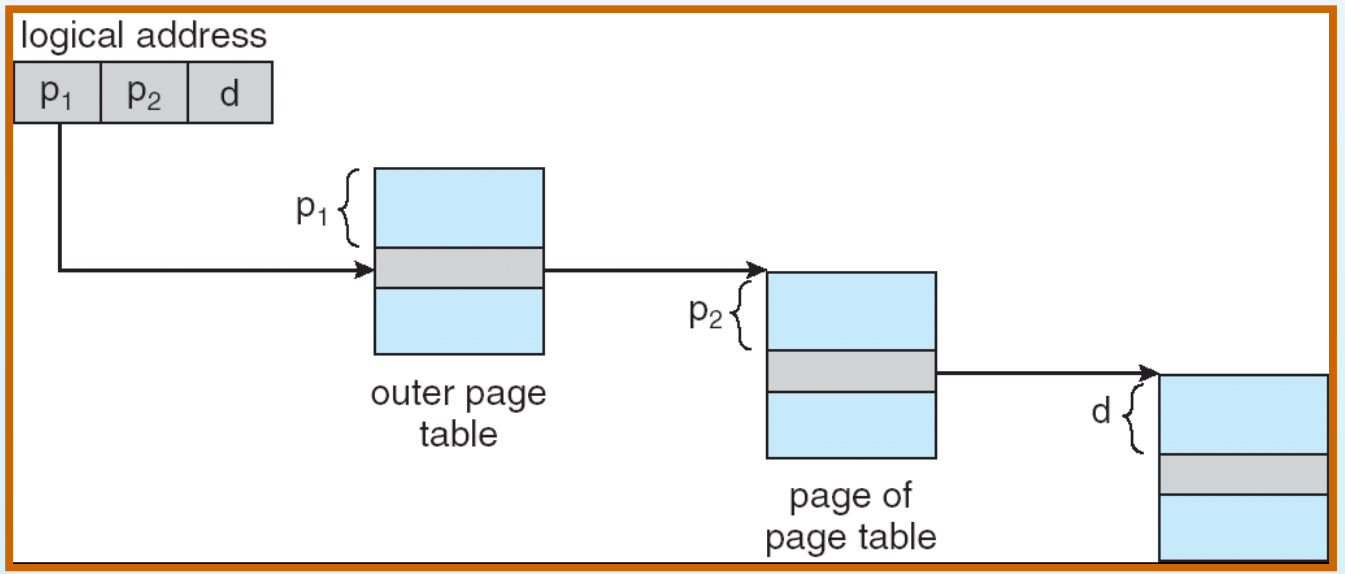


Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a 20 bits page number
  - a 12 bits page offset
- The page number is further divided into
  - a 10-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:



Address-Translation Scheme



Three-level Paging Scheme

outer page	inner page	offset
p_1	p_2	d
42	10	12

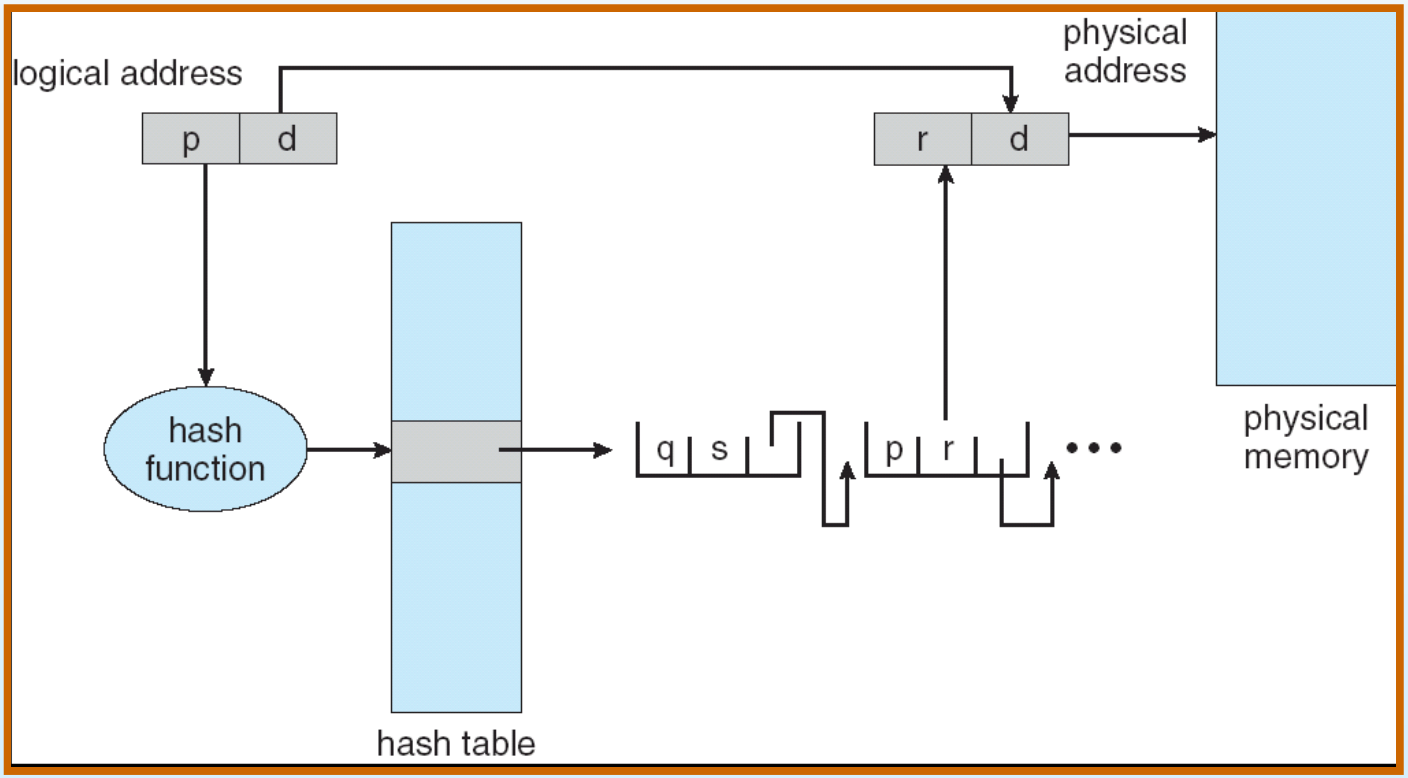
2nd outer page	outer page	inner page	offset
p_1	p_2	p_3	d
32	10	10	12

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.

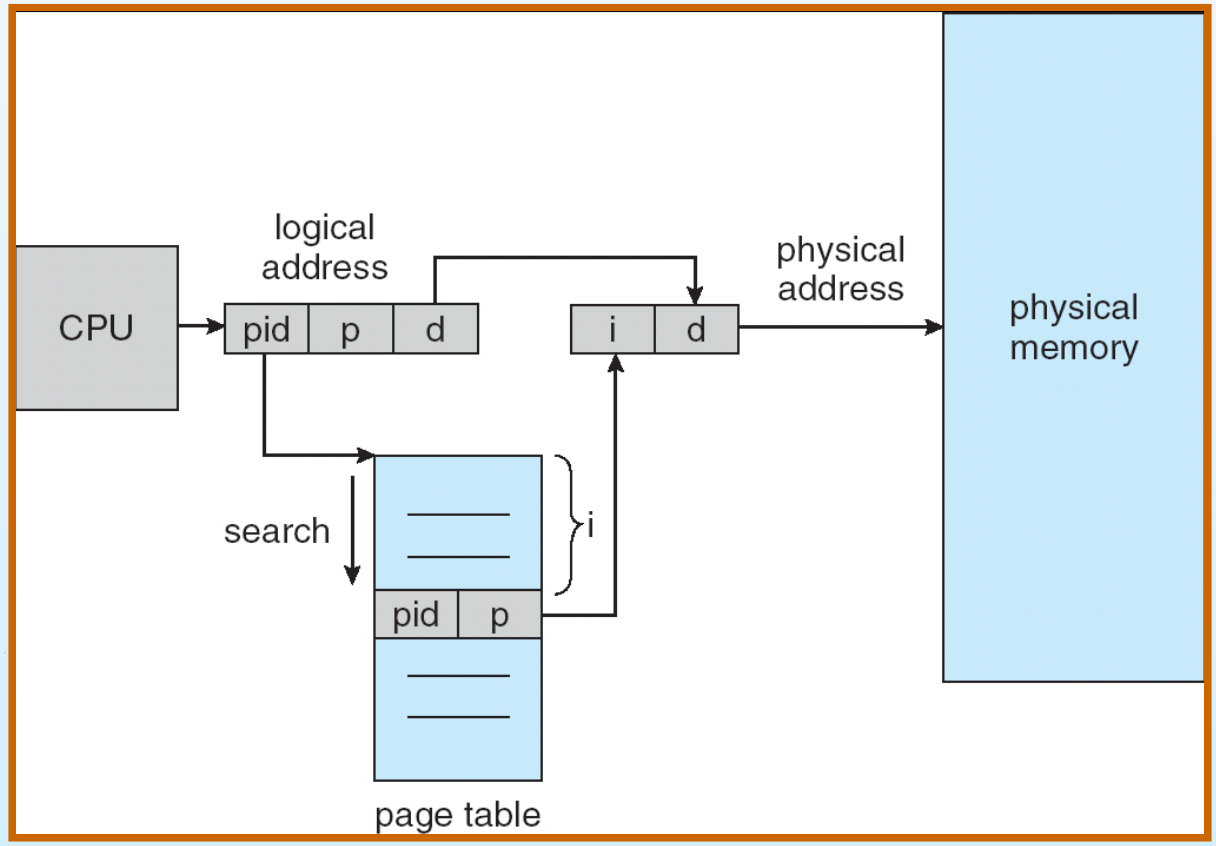
Hashed Page Table



Inverted Page Table

- One entry for each real page of memory
- Entry consists of
  - PID
  - Virtual page number which stored in the real memory
- Decreases memory using but increases time to search the table
- Use hash table to limit the search to one — or at most a few — page-table entries

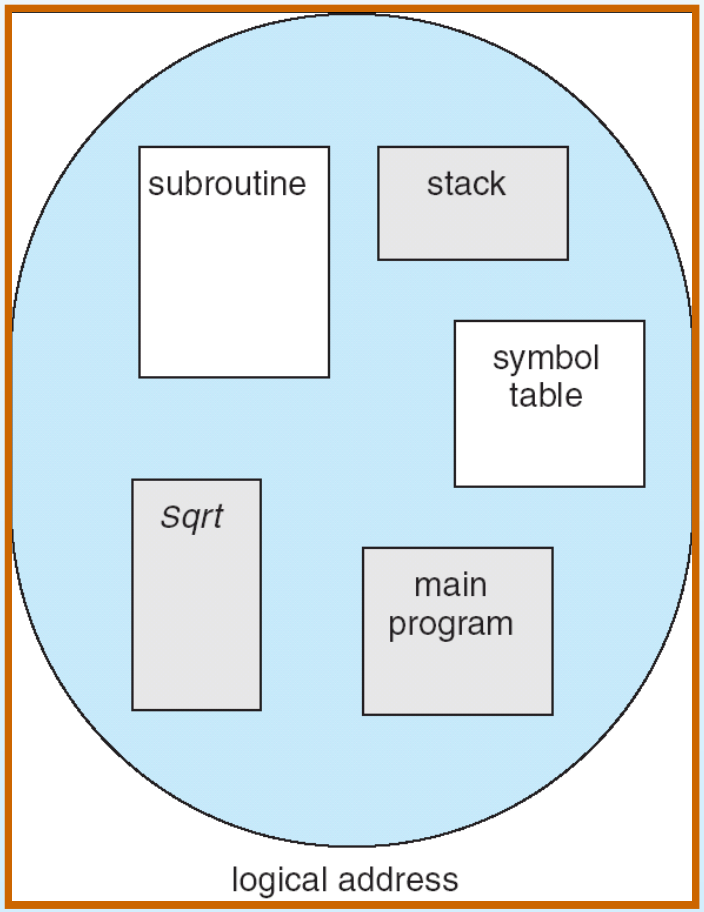
Inverted Page Table Architecture



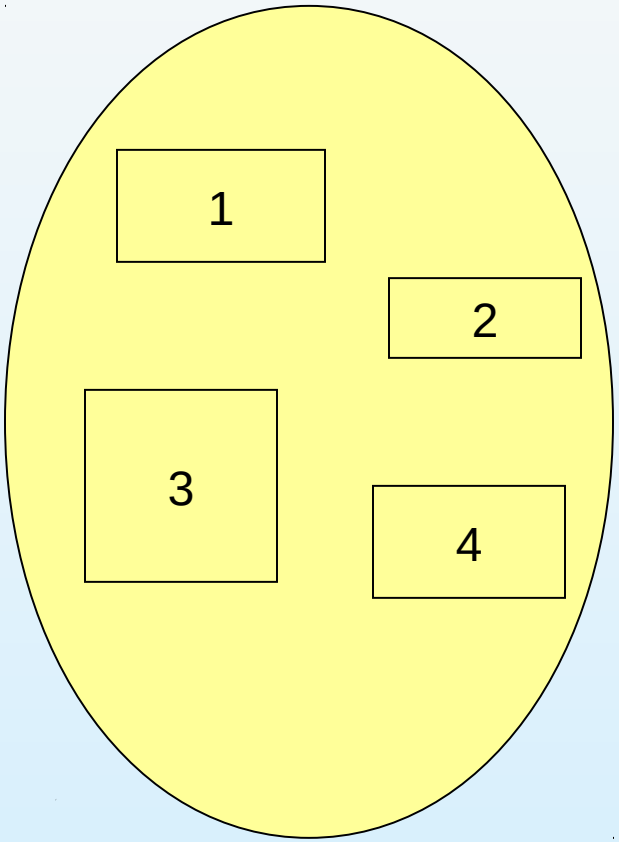
Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments.
- A segment is a logical unit such as:
  - Code
  - Global variable
  - Heap
  - Stacks
  - Libraries

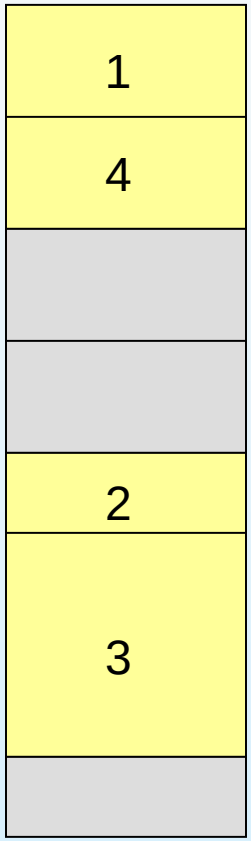
User's View of a Program



Logical View of Segmentation



user space



physical memory space

Segmentation Architecture

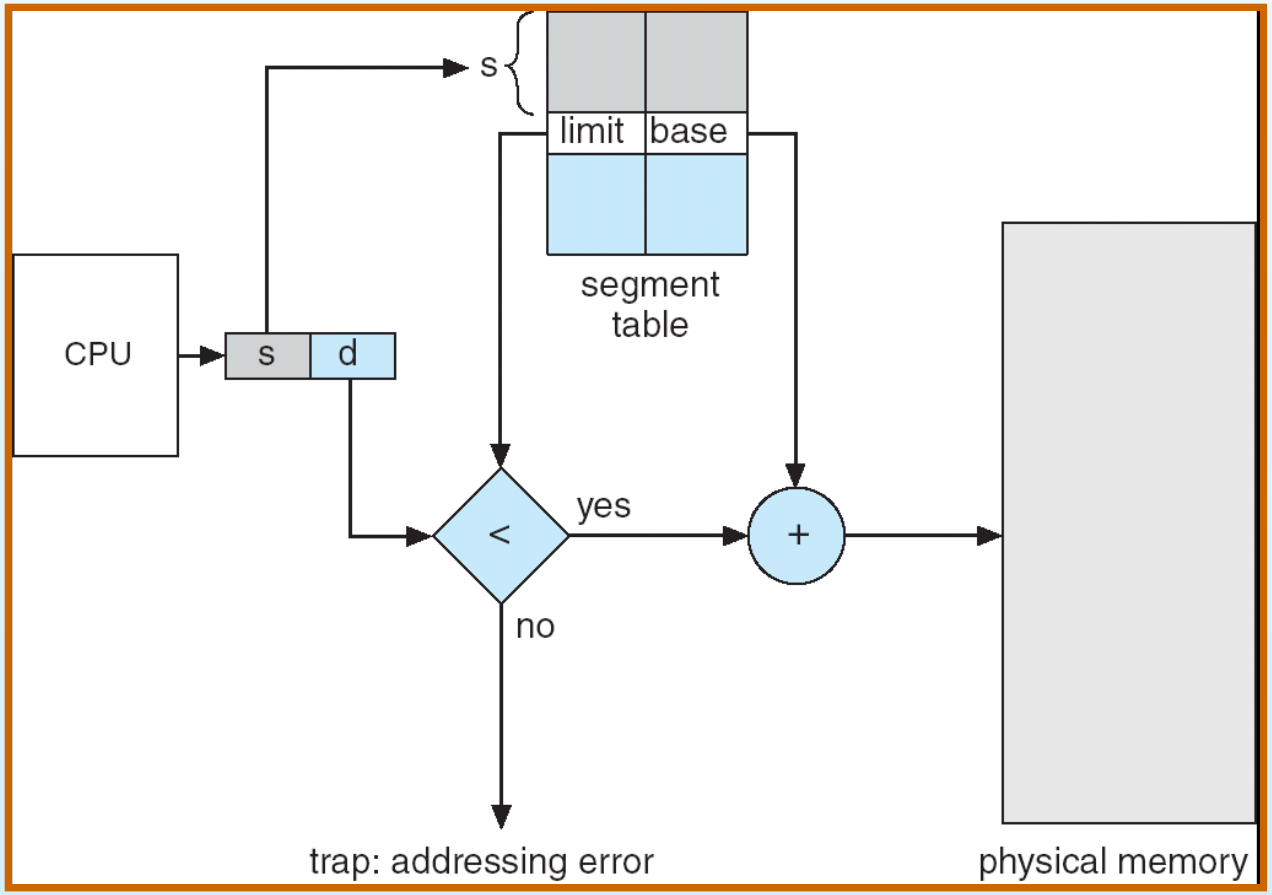
- Logical address consists of a two tuple:

  <segment-number, offset>,

- Segment table do the mapping
- Each table entry has:
  - Base – the starting physical address of segment
  - Limit – the length of the segment
- Segment-table base register (STBR) points to the segment table
- Segment-table length register (STLR) indicates number of segments used by a program;

  segment number *s* is legal if *s* < STLR

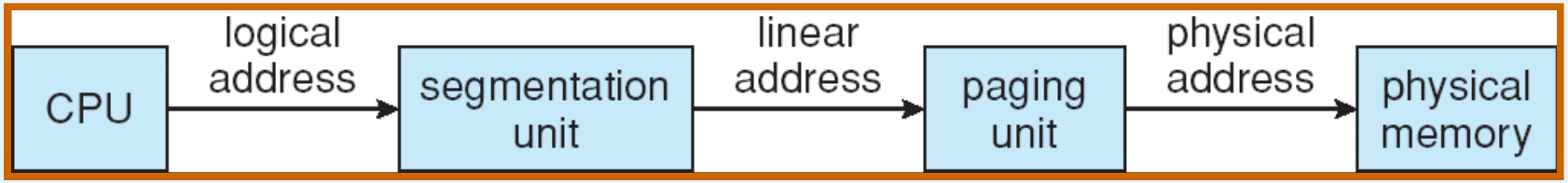
Segmentation Hardware



Example: The Intel Pentium

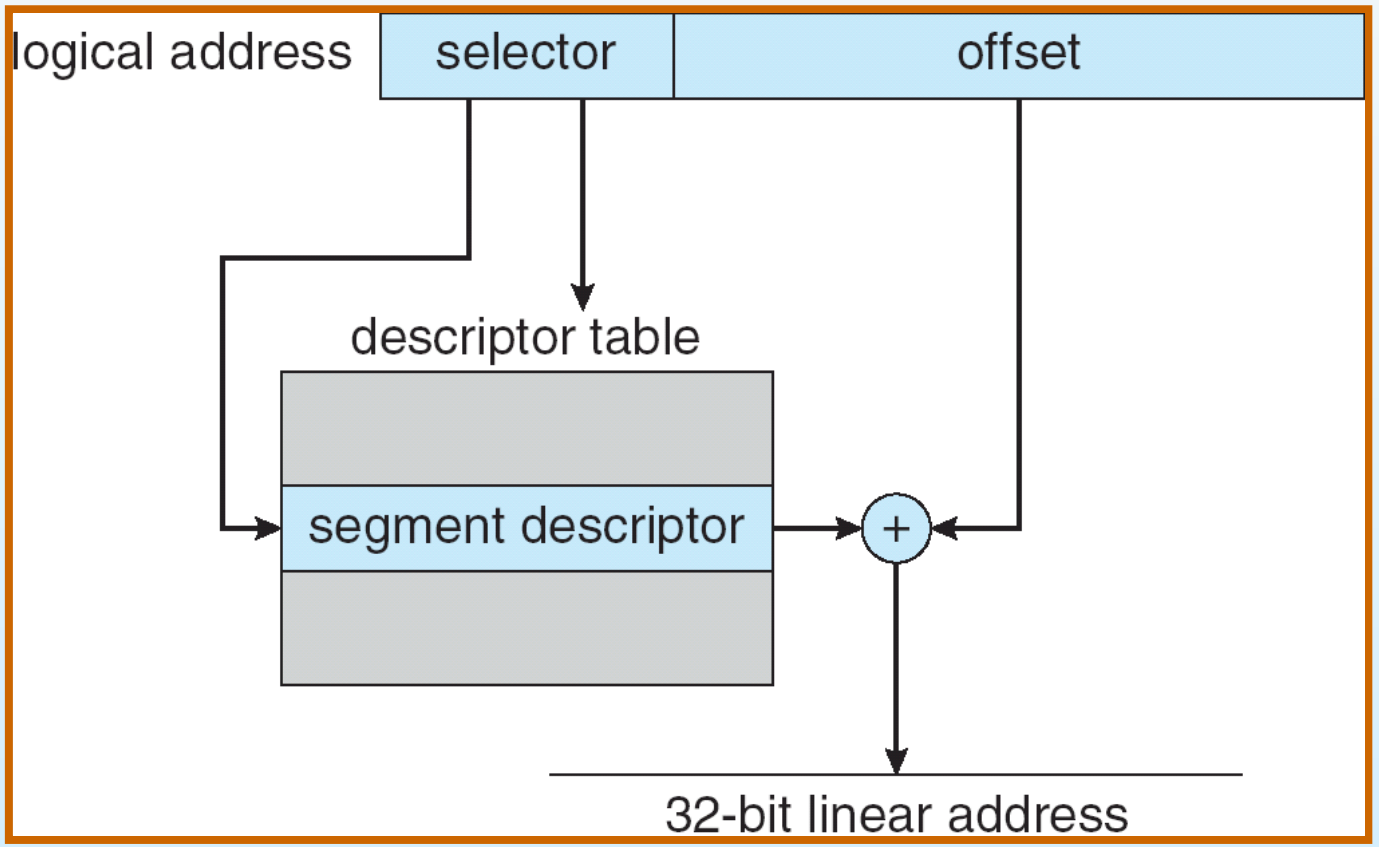
- Supports
  - Segmentation
  - Paging
  - Segmentation with paging
- CPU generates logical address
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address

Logical to Physical Address Translation in Pentium

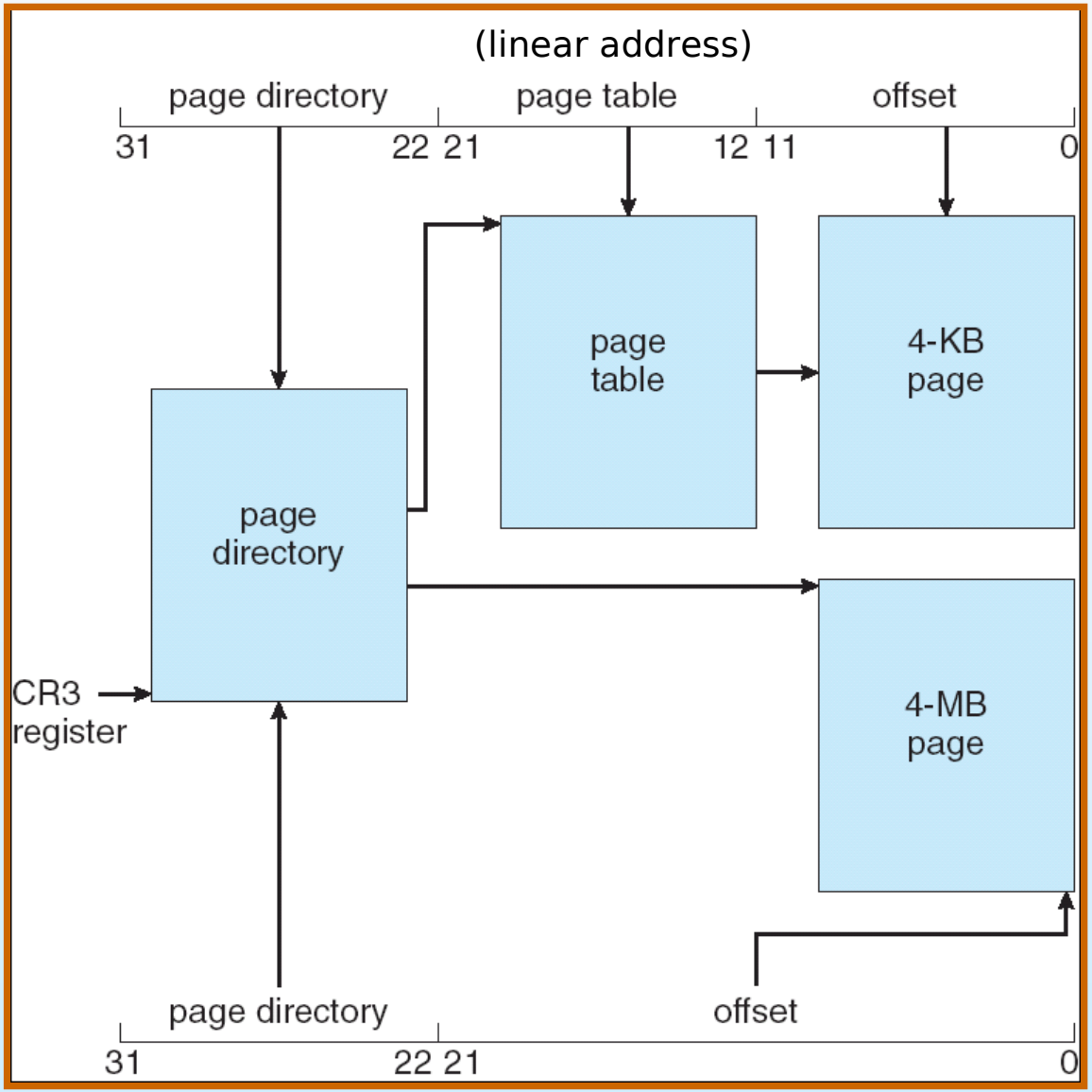


page number		page offset
p_1	p_2	d
10	10	12

Intel Pentium Segmentation



Pentium Paging Architecture

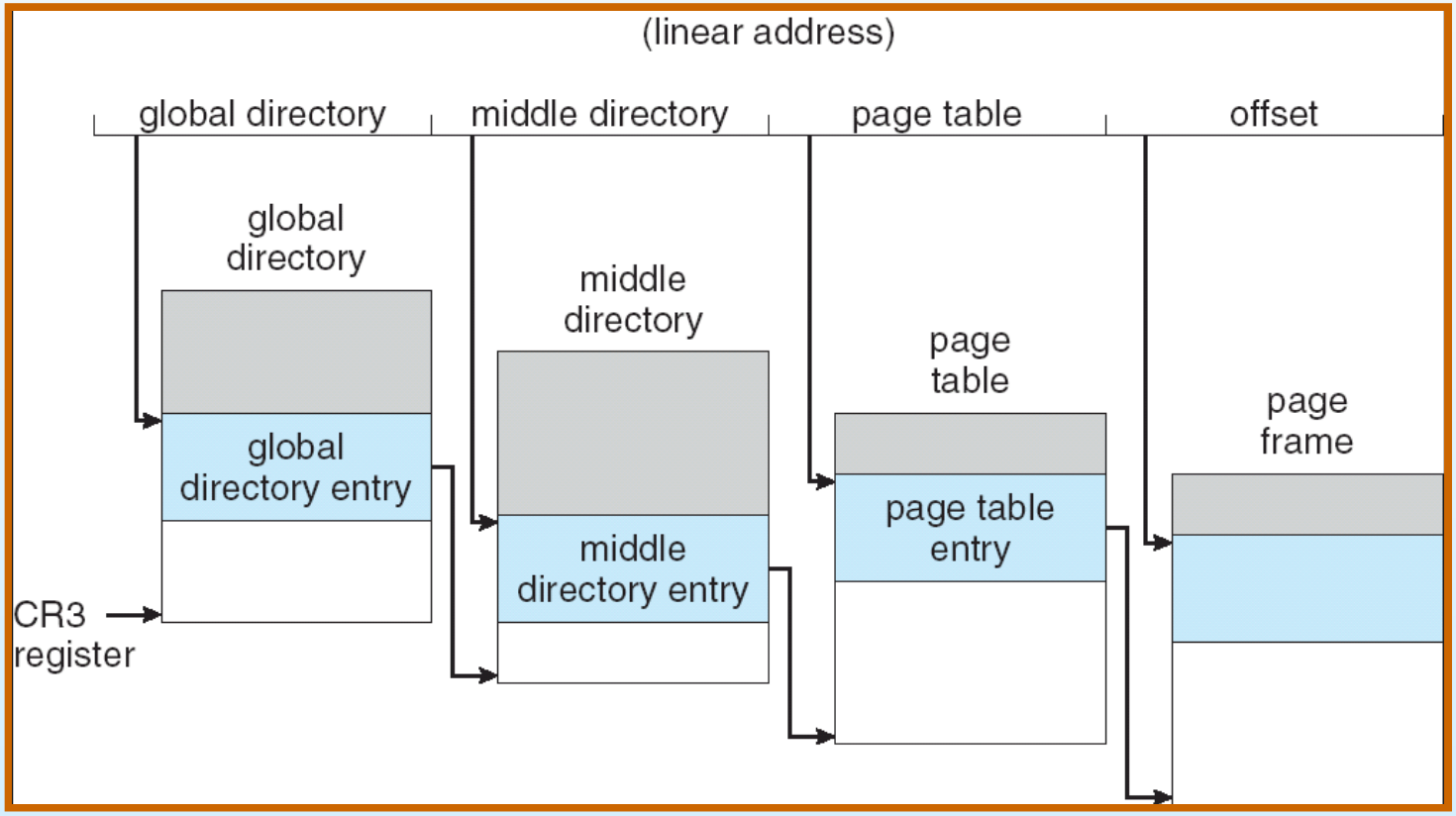


Linear Address in Linux

- To support more platform, linear address is broken into four parts:

- In IA32, the number of bits of middle-directory is 0

Three-level Paging in Linux



End of Chapter 8