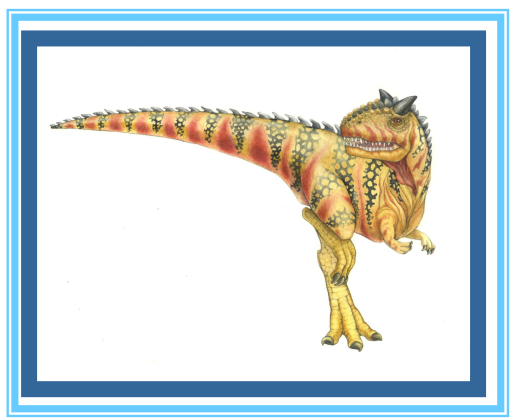
Chapter 8: Main Memory



Chapter 8: Main Memory

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

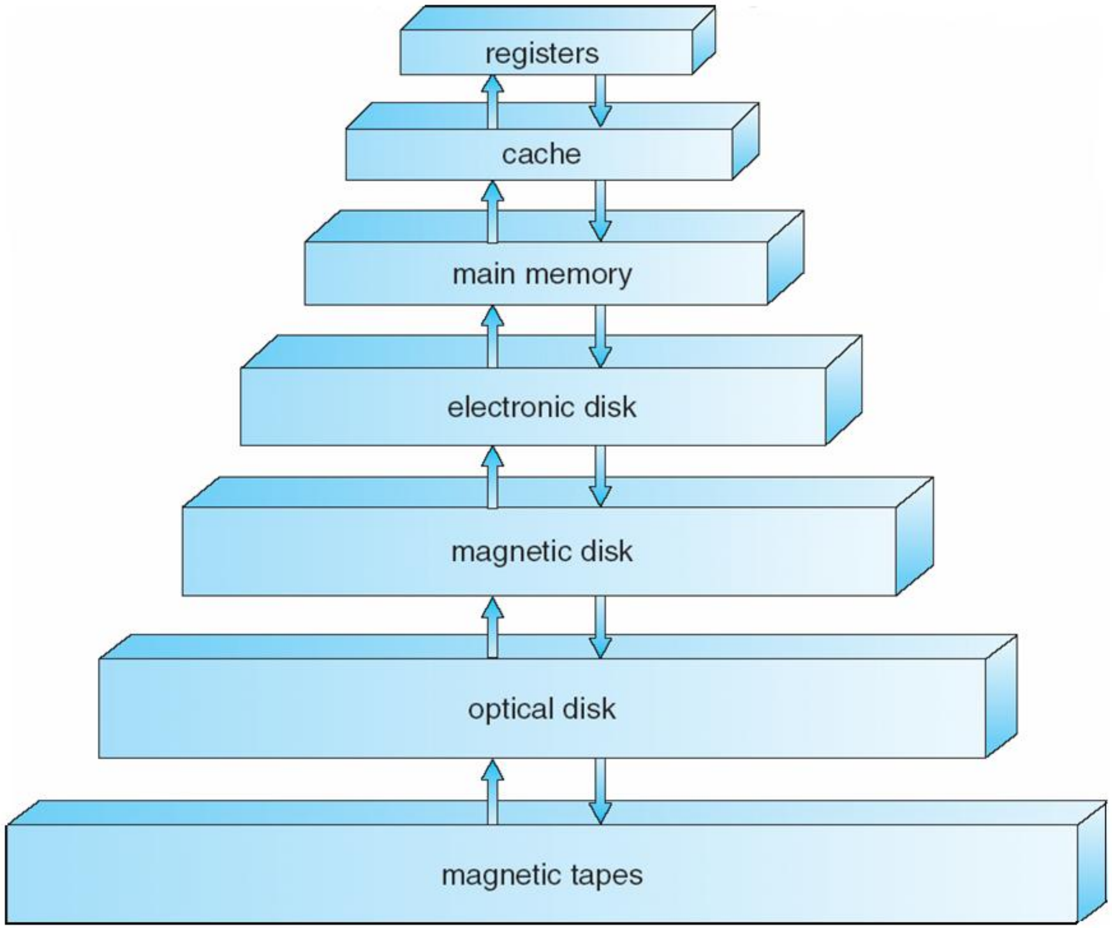
Objectives

- To provide a detailed description of various ways of organizing memory hardware
- To discuss various memory-management techniques, including paging and segmentation
- To provide a detailed description of the Intel Pentium, which supports both pure segmentation and segmentation with paging

Background

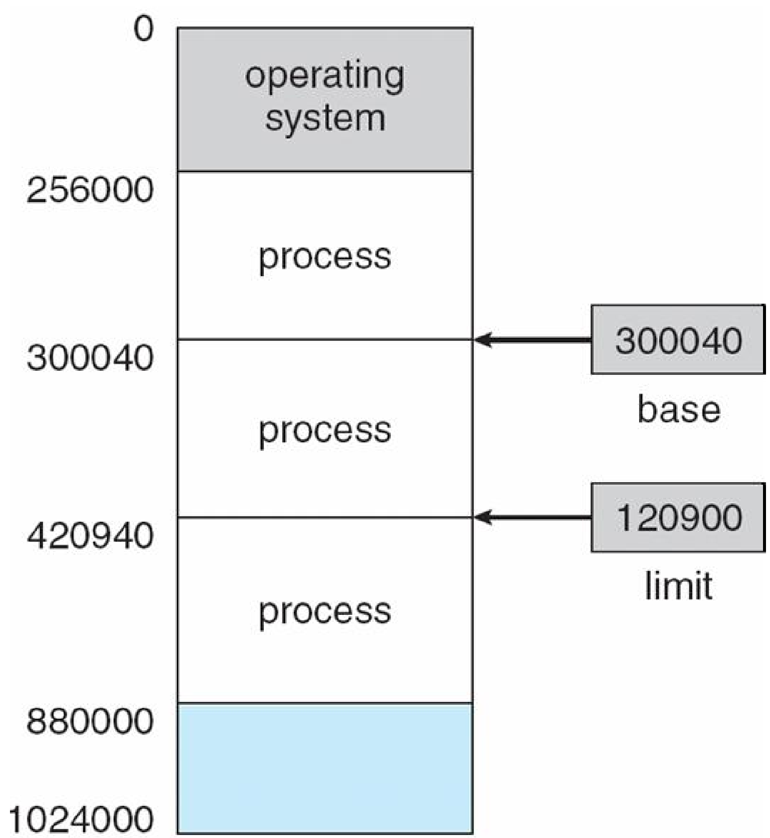
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock (or less)
- Main memory can take many cycles
- **Cache** sits between main memory and CPU registers

Storage-Device Hierarchy



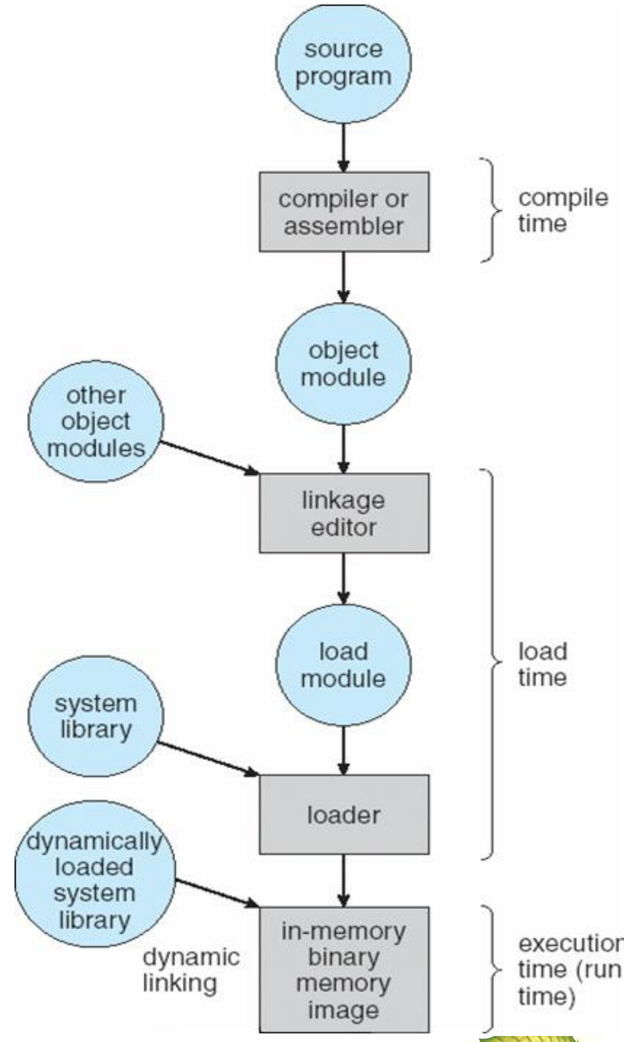
Base and Limit Registers

- Protection of memory required to ensure correct operation
- A pair of **base** and **limit** registers define the logical address space



Binding of Instructions and Data to Memory

- In most cases, a user program will go through several steps—some of which may be optional---before being executed.
- Addresses may be different in these steps.
  - In source program: symbolic address
  - After compiled: relative address
  - After linked or loaded: absolute address



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)

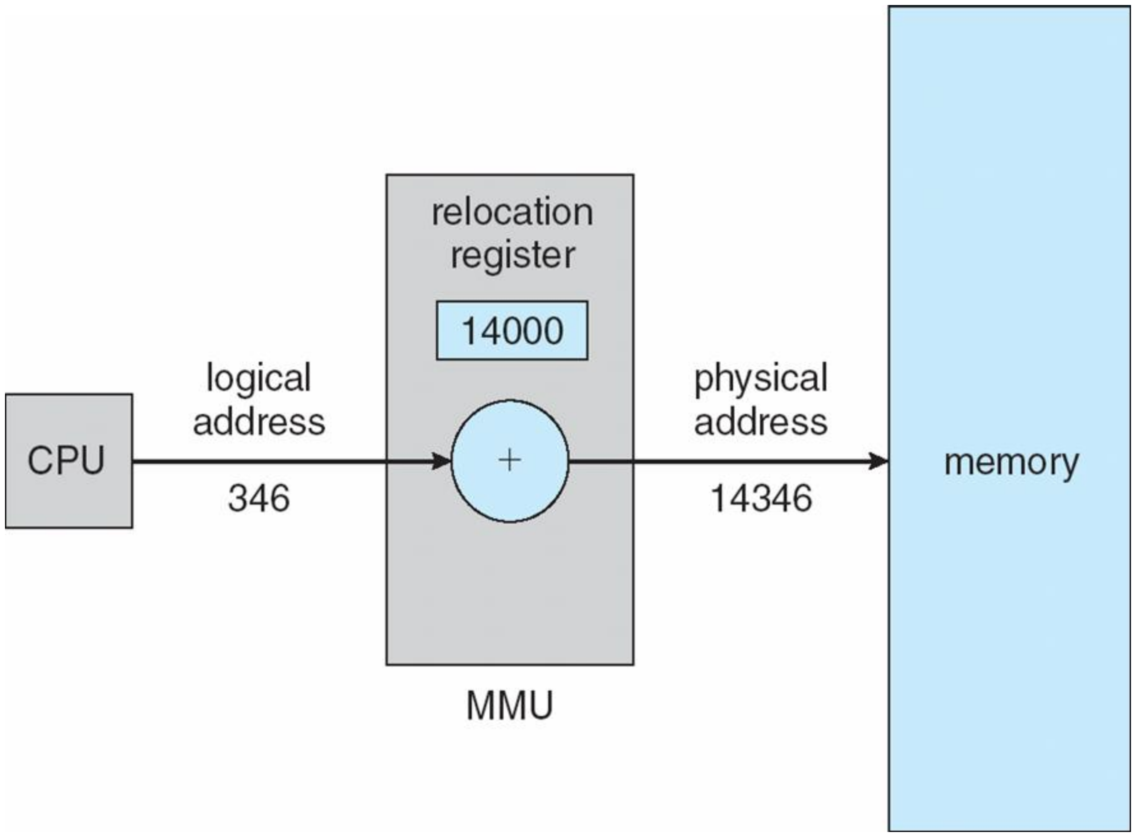
Logical vs. Physical Address Space

- 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

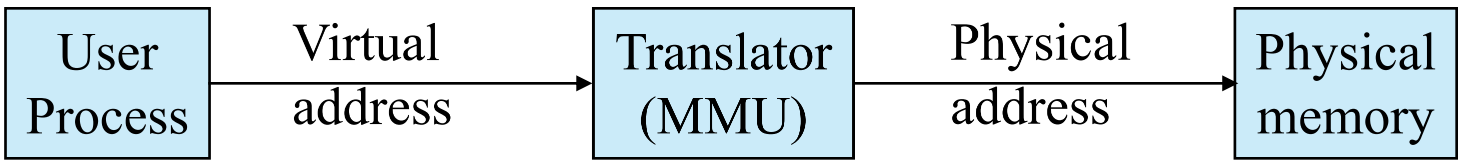
Memory-Management Unit (MMU)

- The run-time mapping from virtual to physical addresses is done by a hardware called MMU
- 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 Address Translation



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. It is implemented through program design.

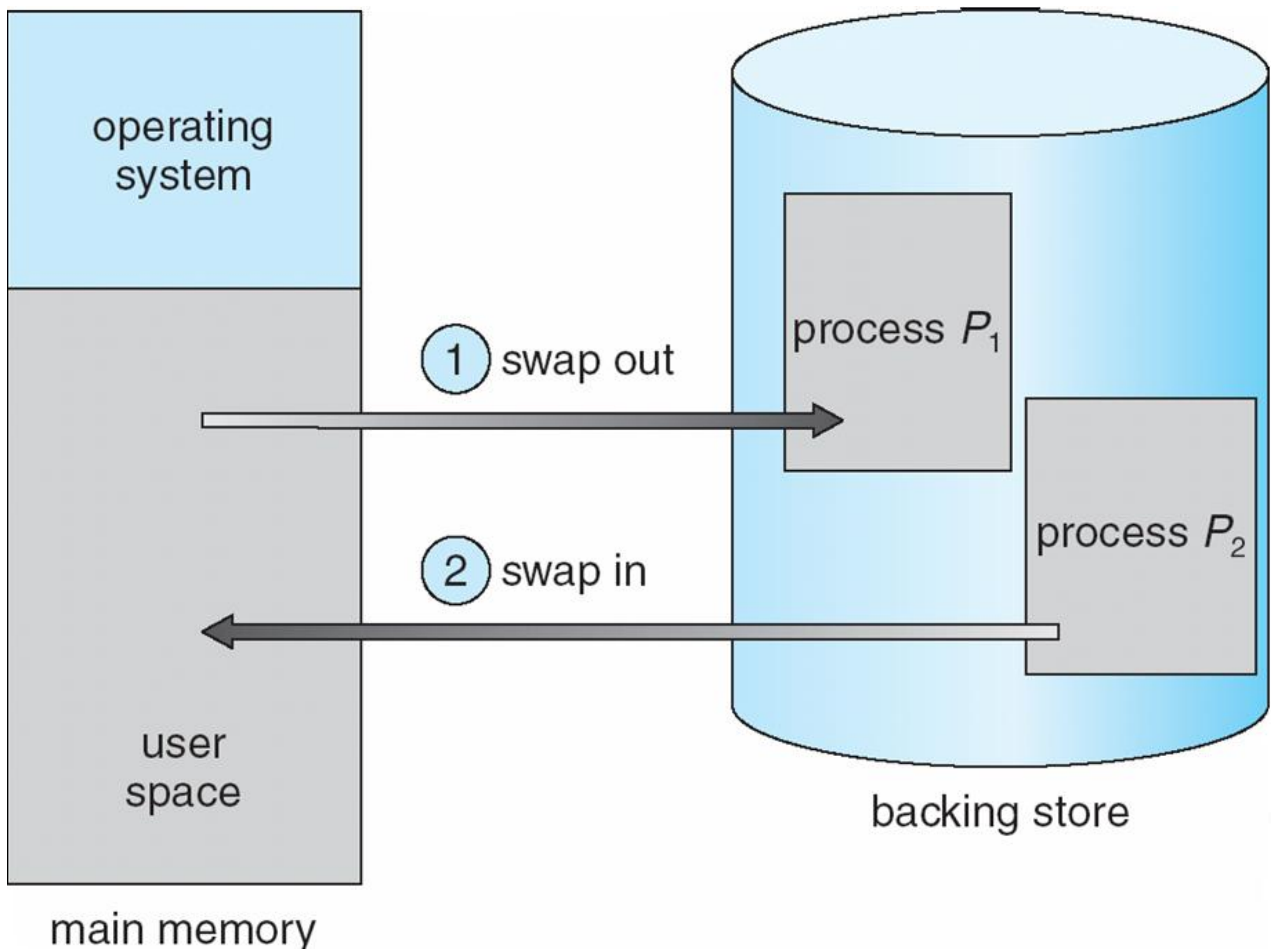
Dynamic Linking

- 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

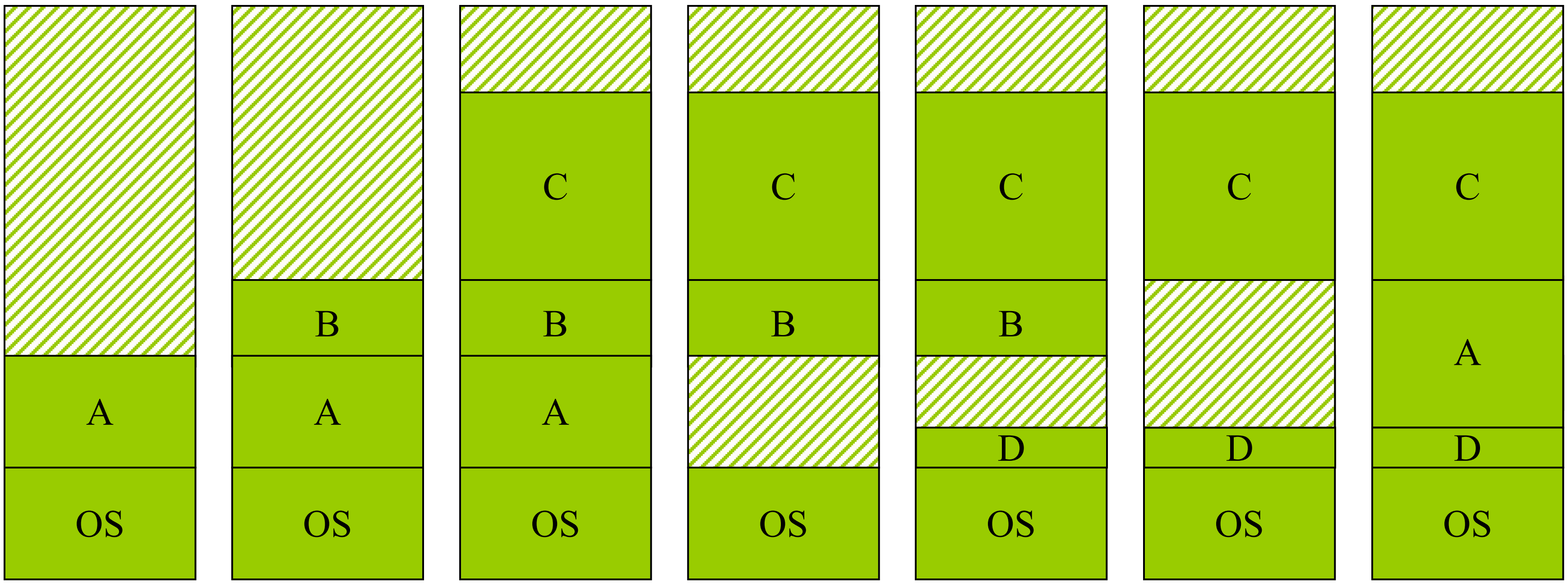
Swapping

- 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
- **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
- 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



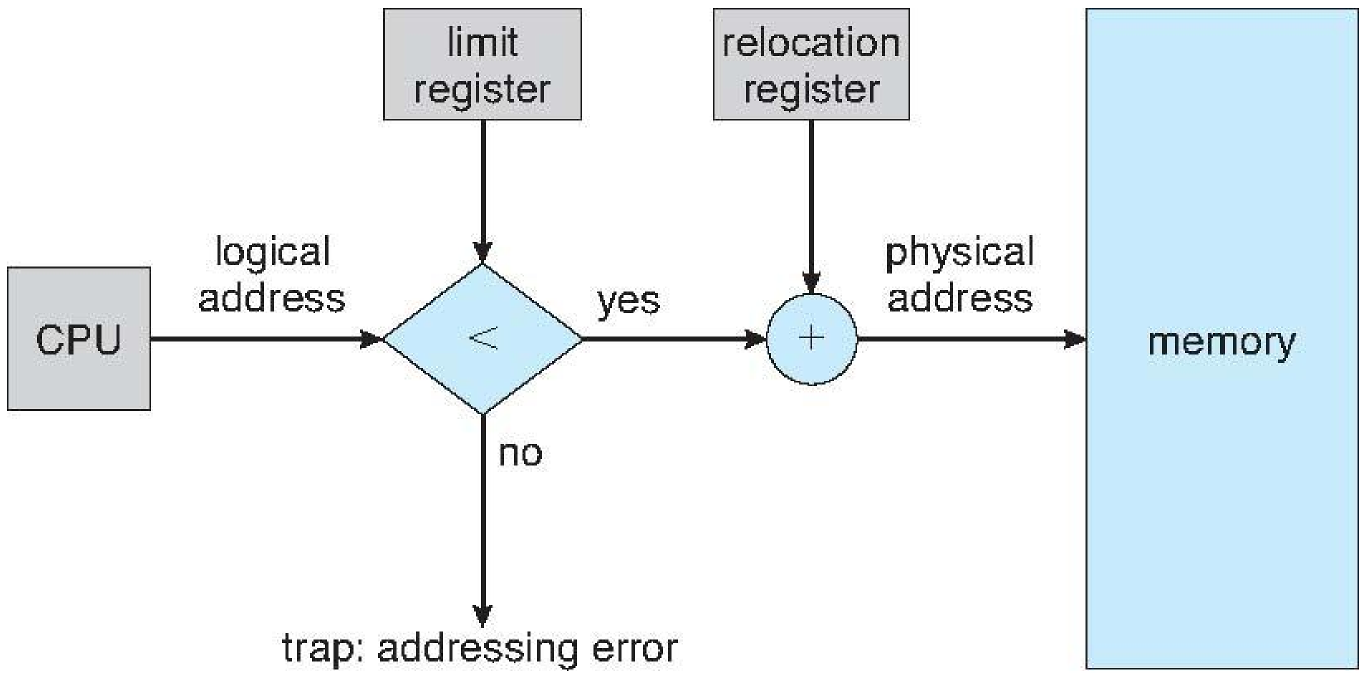
Swapping



Contiguous Allocation

- Main memory is usually divided into 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*

Hardware Support for Relocation and Limit Registers



Contiguous Allocation

- Fixed Partitioning
- Dynamic Partitioning
- Dynamic Relocationable Partitioning

Contiguous Allocation

- Fixed Partitioning
  - Divide memory into a number of fixed-sized partitions.
  - Each partition may contain exactly one process.
  - The degree of multiprogramming is bound by the number of partitions.
  - The OS keeps a table indicating which parts of memory are available and which are occupied.

Fixed Partitioning

No.	size	Beginning address	Free or not
1	16K	20K	0
2	32K	36K	0
3	64K	68K	0
4	124K	132K	1

Partition table

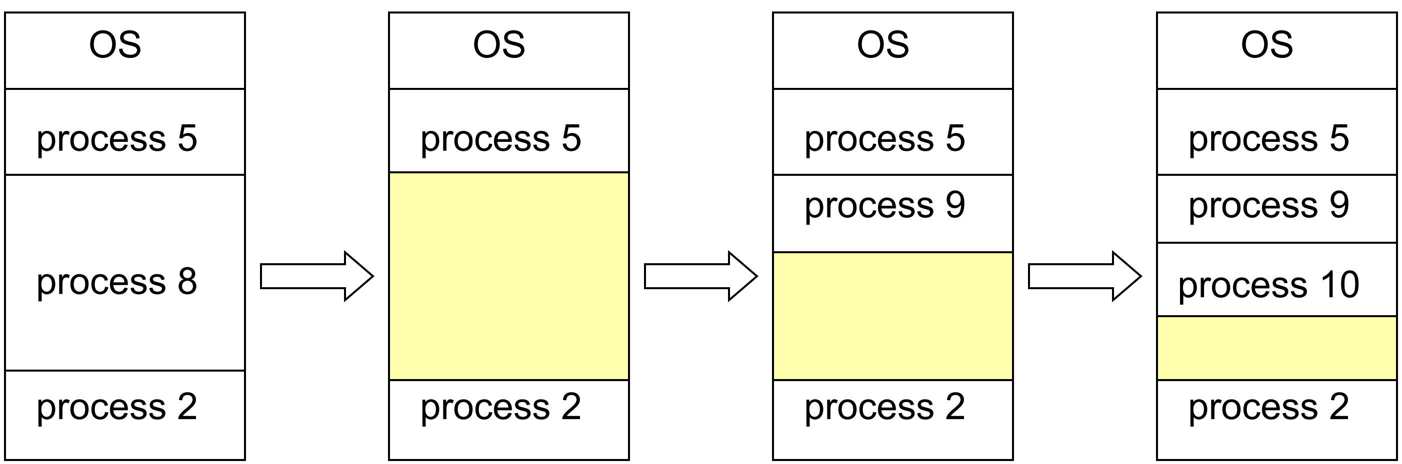
memory

Fixed Partitioning

- It is easy to implement
- disadvantages：
  - Fixed partition size: internal fragmentation
  - Fixed partition number: The degree of multiprogramming is bound

Contiguous Allocation (Cont)

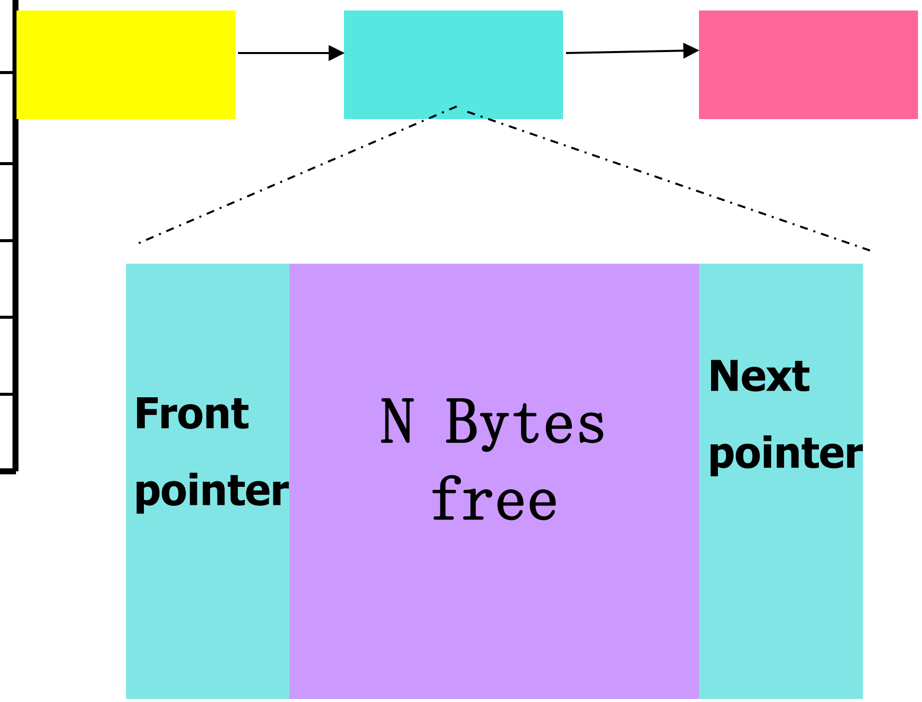
- Dynamic-partition allocation
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Partition management

No.	size	Beginning address	state
1	48K	116K	free
2	252K	260K	free
3	---	---	----
4	---	---	----
5	---	---	----

Free Partition table



Free partition list

Dynamic Storage-Allocation Problem

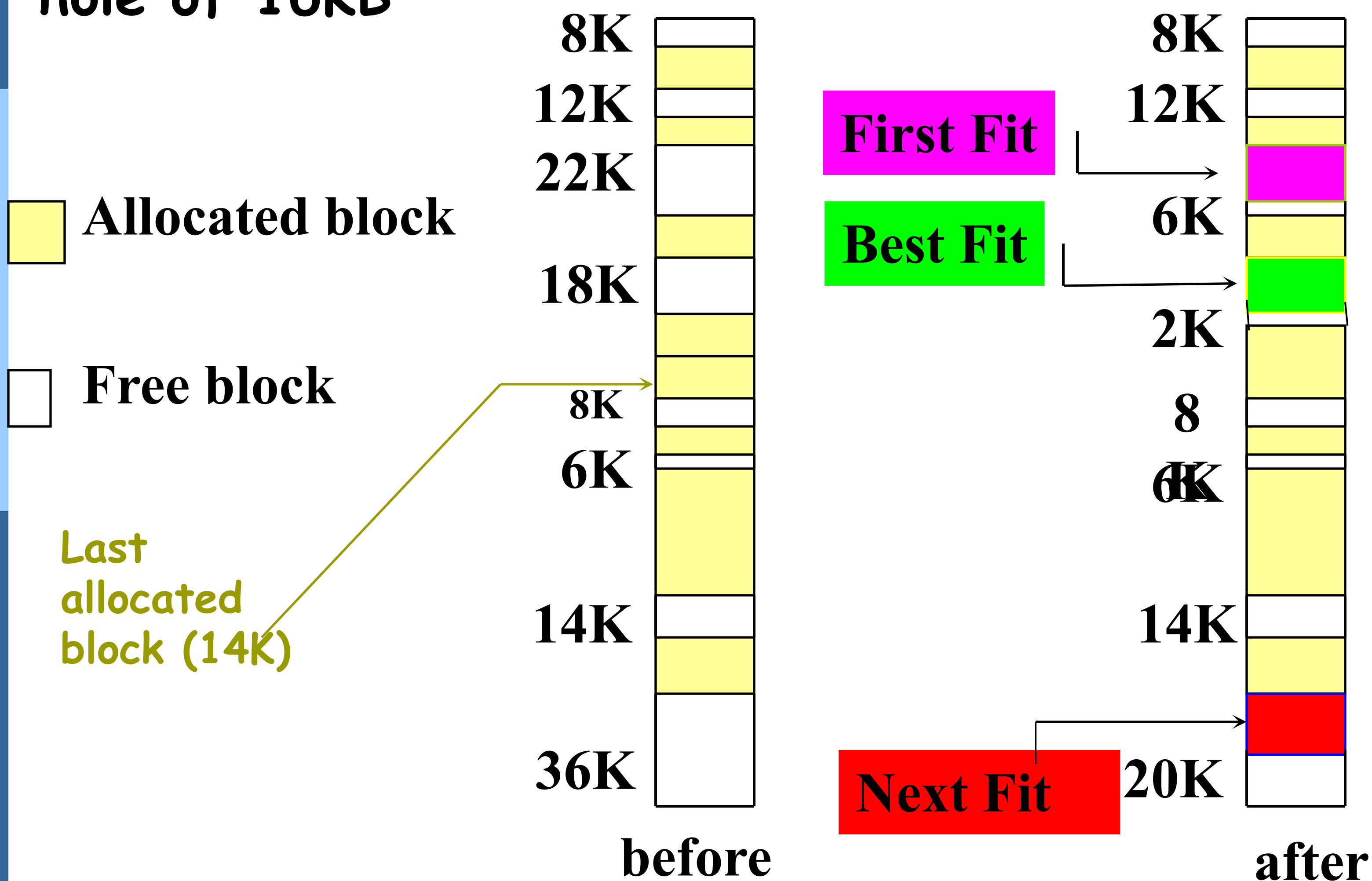
How to satisfy a request of size *n* from a list of free holes

- **First-fit**: Allocate the *first* hole that is big enough
- **Next-fit**: Search from the last allocated hole
- **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

example

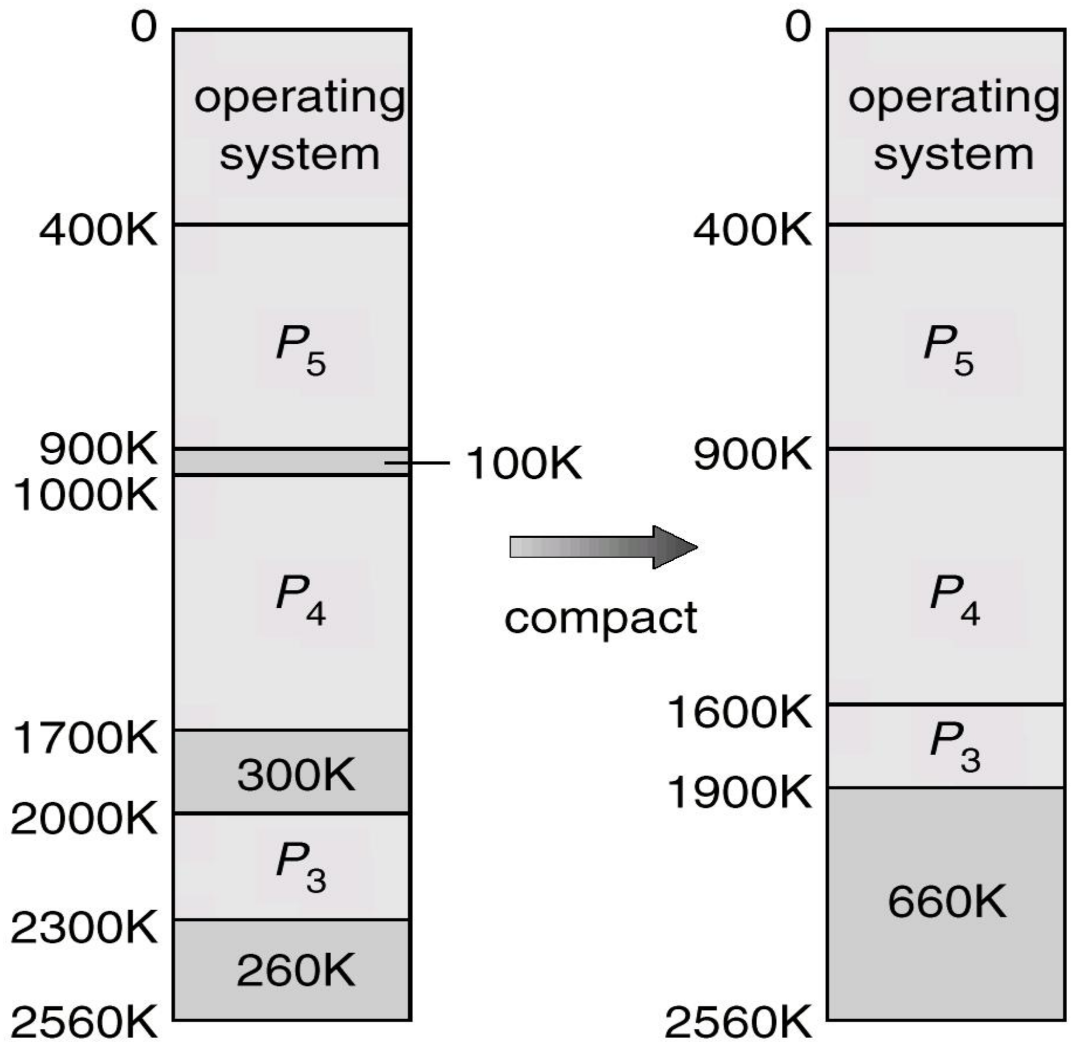
To allocate a hole of 16KB



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

Fragmentation



Paging

- One solution to external fragmentation is compaction.
  - Can be expensive.
- Another solution is to permit the logical address space of a process can be noncontiguous, thus allowing a process to be allocated physical memory whenever the latter is available.

Paging

- Physical address space of a process can be noncontiguous;
- 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

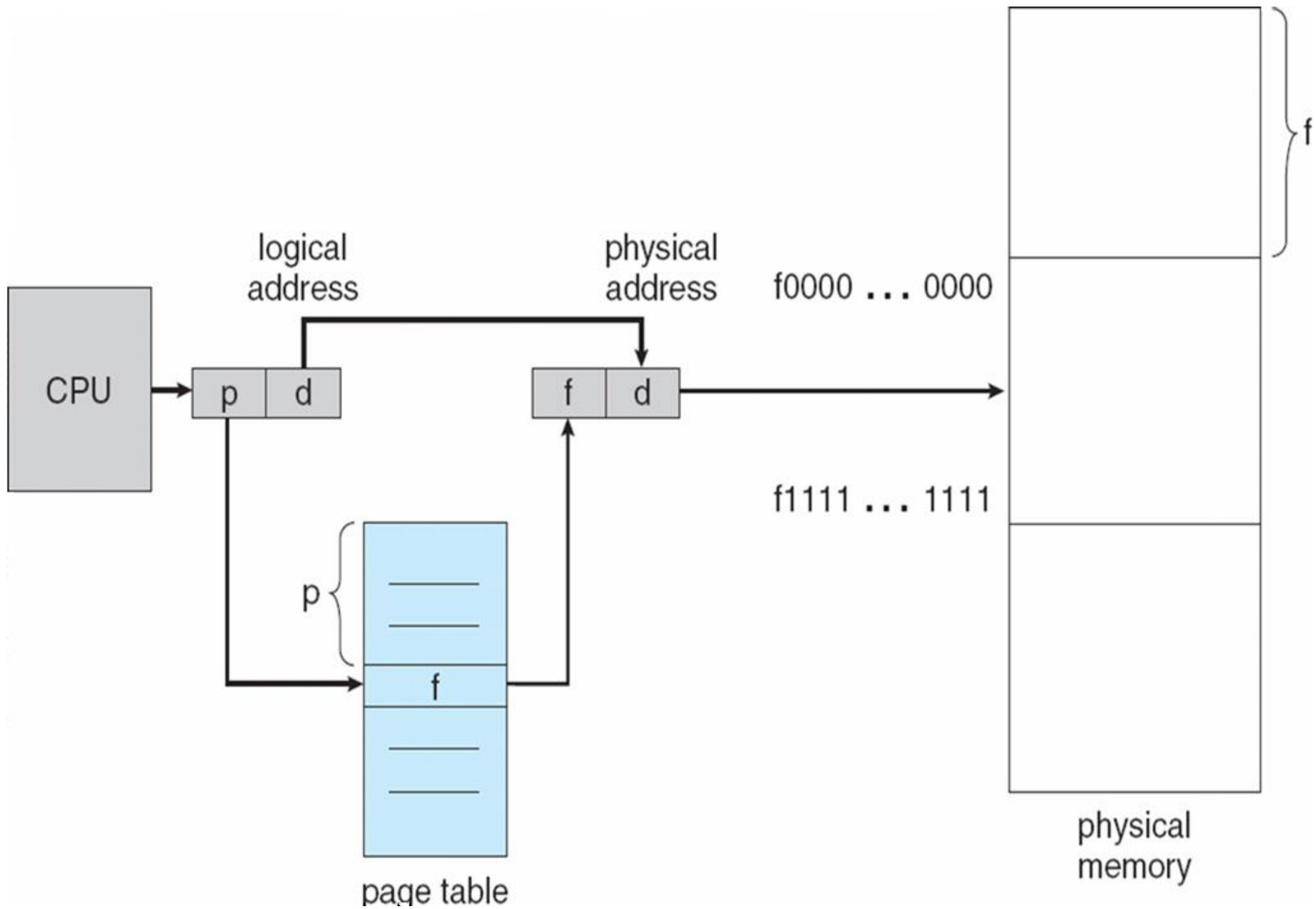
Address Translation Scheme

- 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

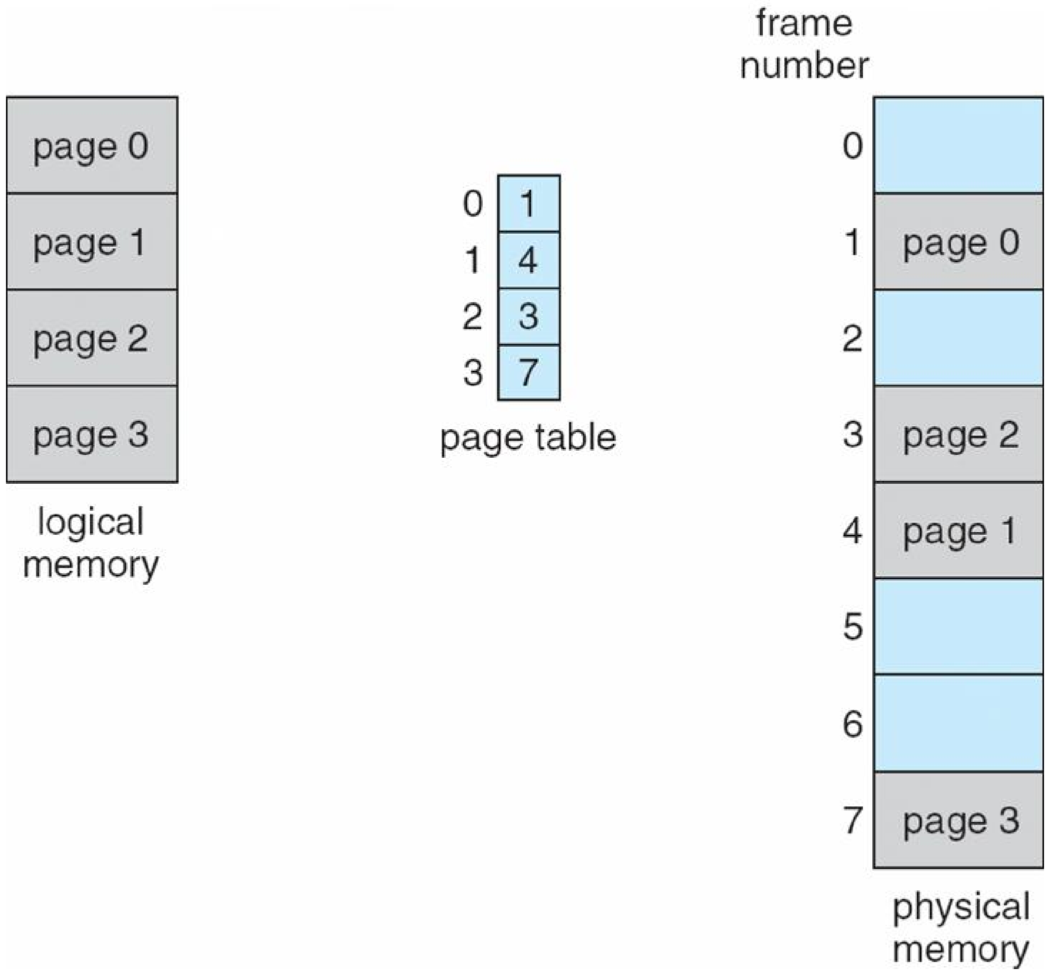
page number	page offset
p	d
$m - n$	n

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

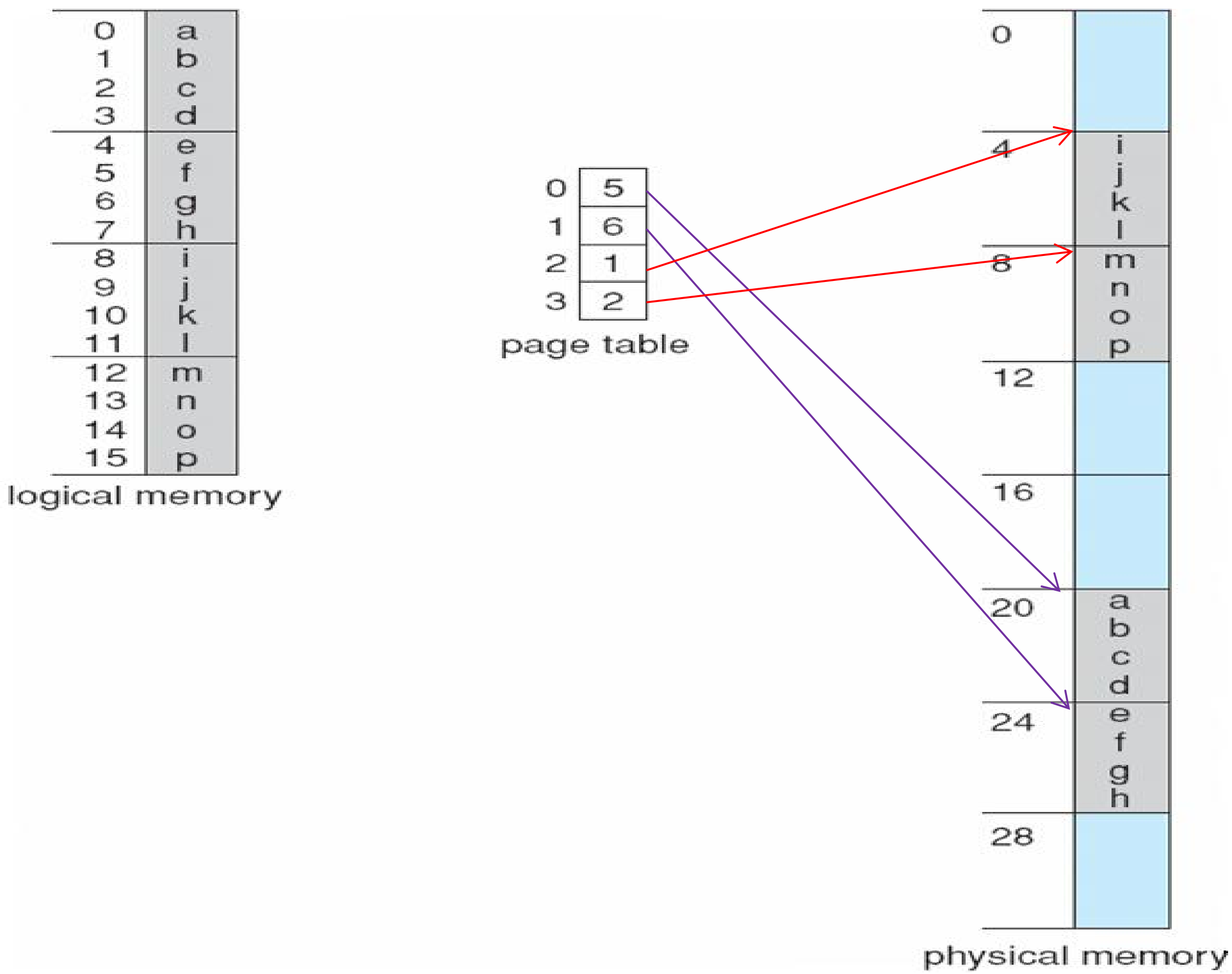
Paging Hardware



Paging Model of Logical and Physical Memory

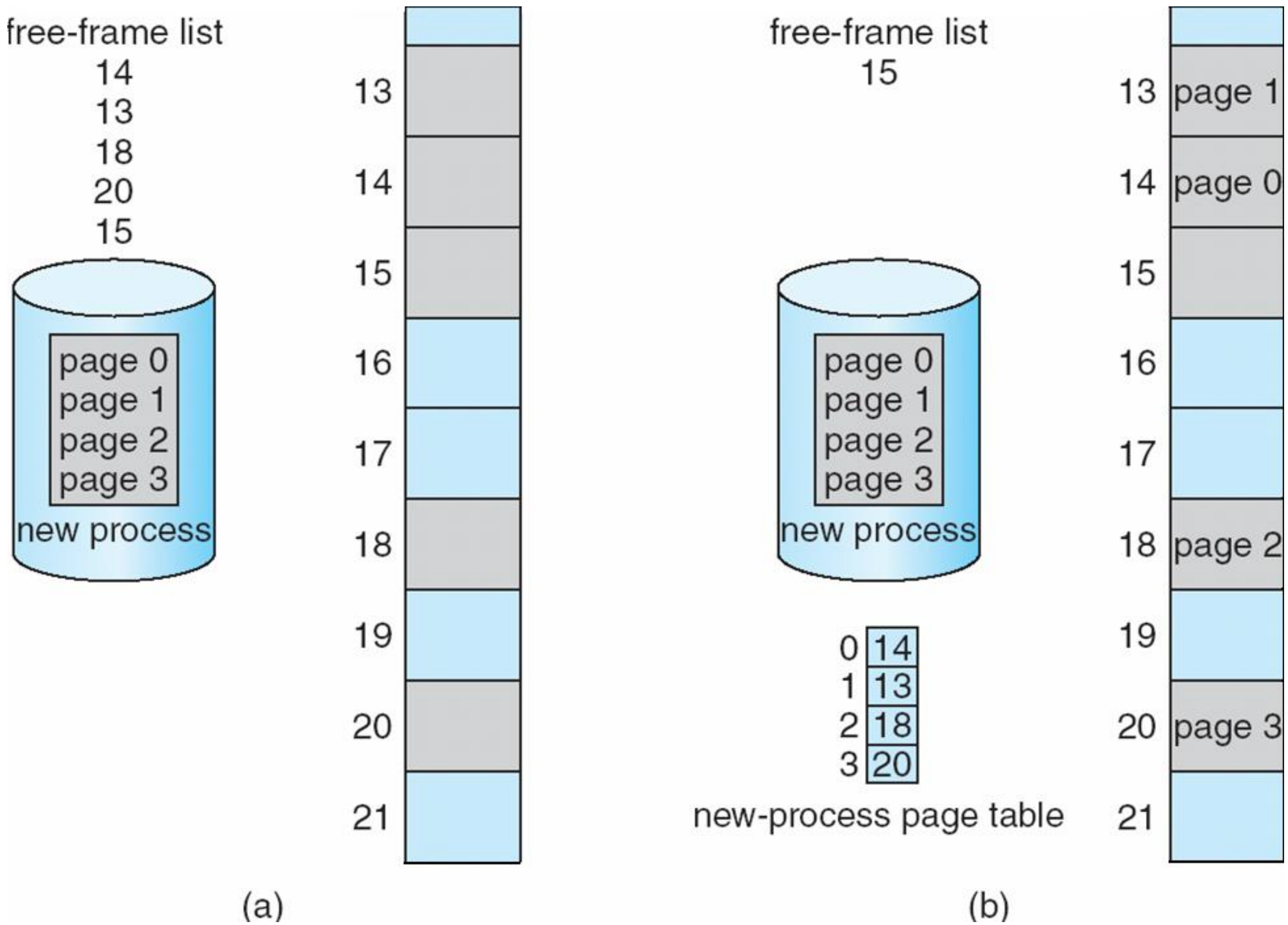


Paging Example



32-byte memory and 4-byte pages

Free Frames

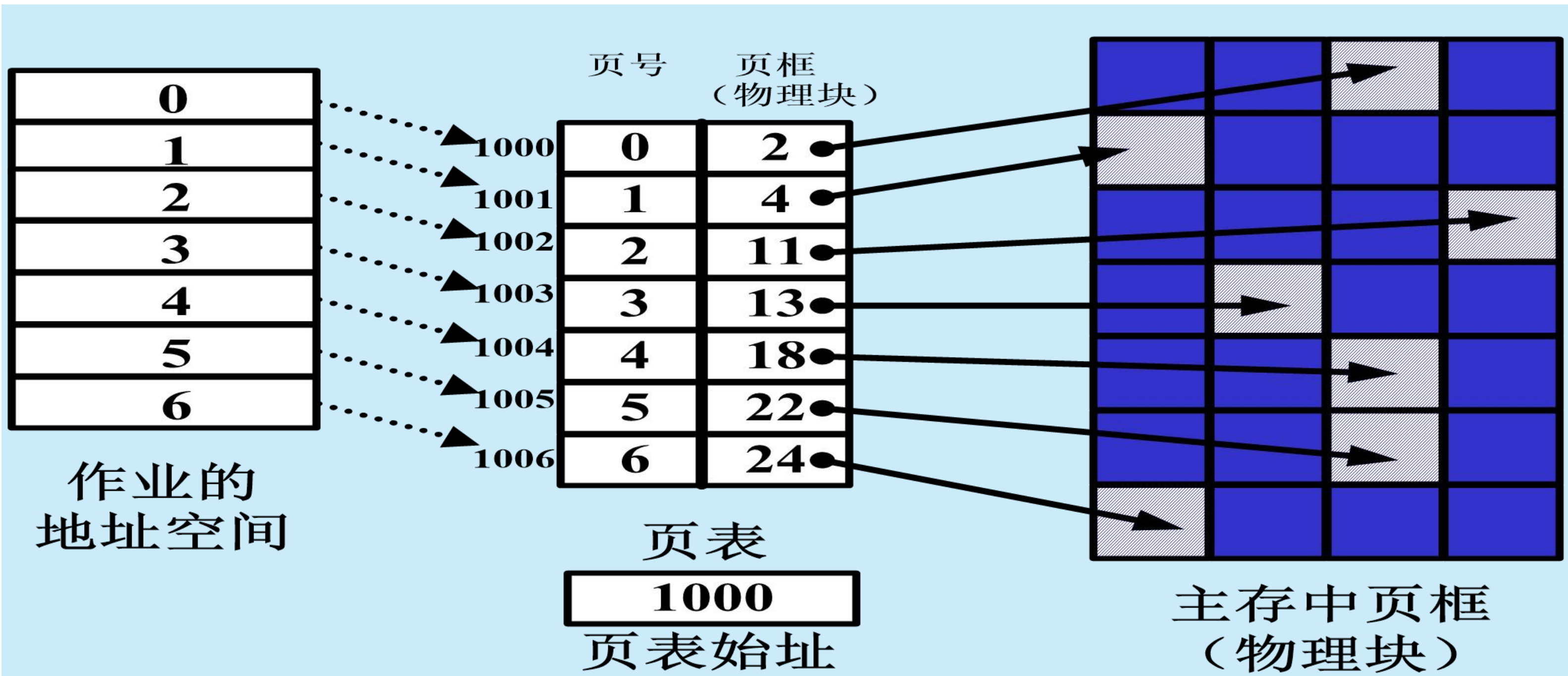


Before allocation

After allocation

Page table

- Map page number to frame number



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
- One for the data/instruction.

TLB

- 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)**
- **Fast but expensive**
- **Numbering between 64-1024**
- 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

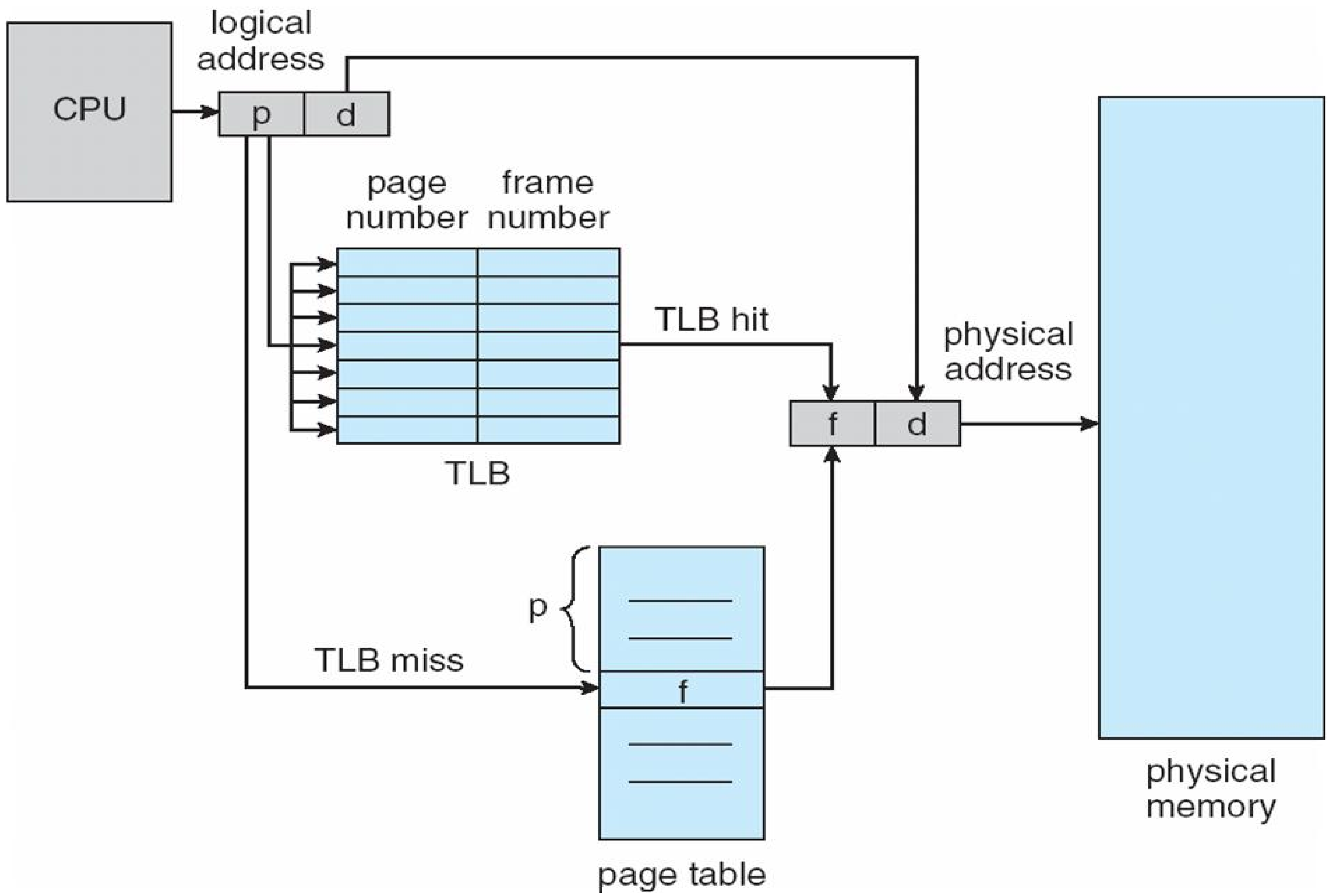
- Associative memory – parallel search

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



Effective Access Time

- Associative Lookup = ε time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Hit ratio = α
- **Effective Access Time** (EAT)

$$\begin{aligned} \text{EAT} &= (1 + \varepsilon)\, \alpha + (2 + \varepsilon)(1 - \alpha) \\ &= 2 + \varepsilon - \alpha \end{aligned}$$

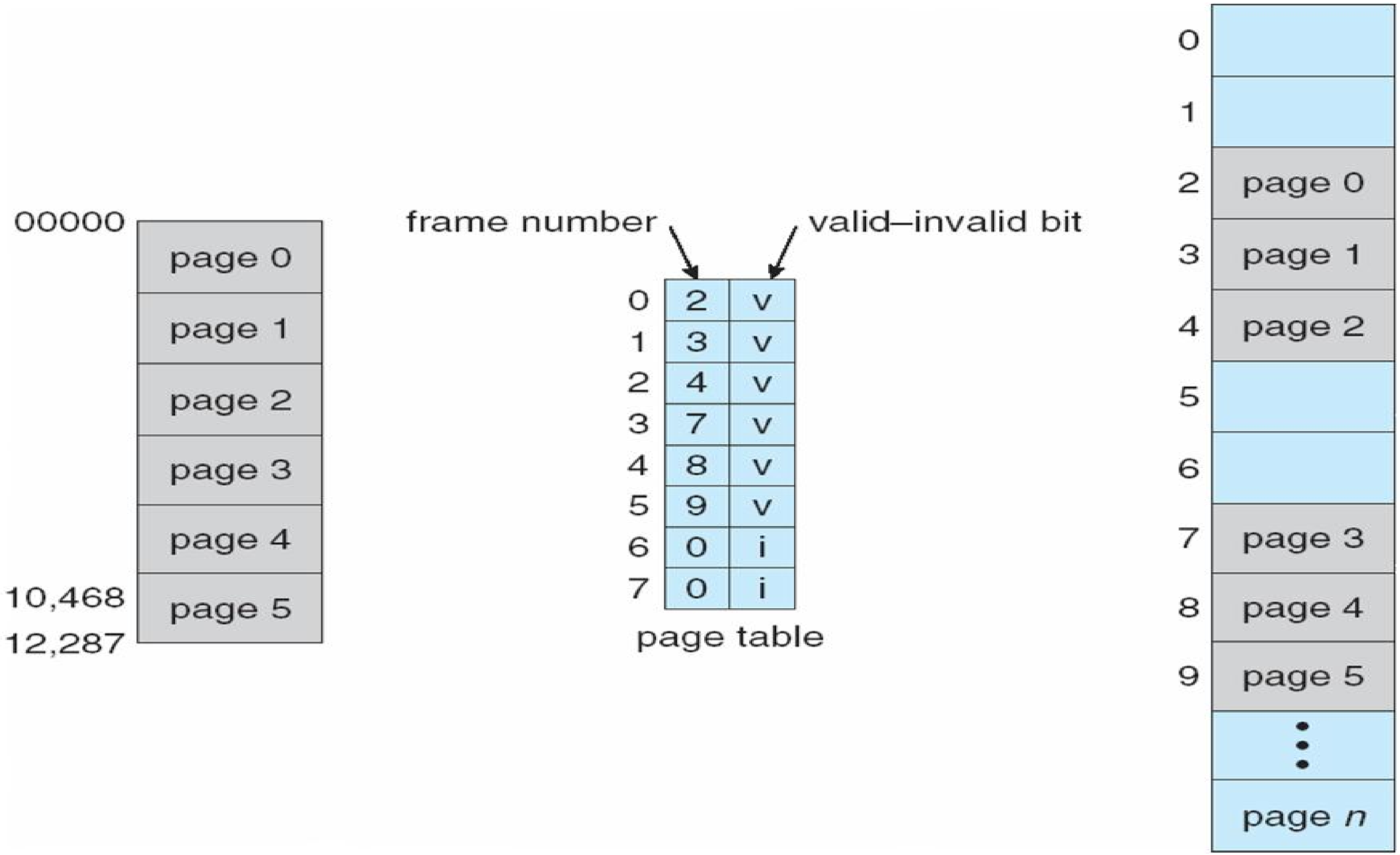
Effective Access Time

- Assume that associative lookup time is 20ns, memory cycle time is 100ns, the TLB hit ratio is 85%, then effective access time is as follow:
  - T=0.85*120+0.15*220=135ns.

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

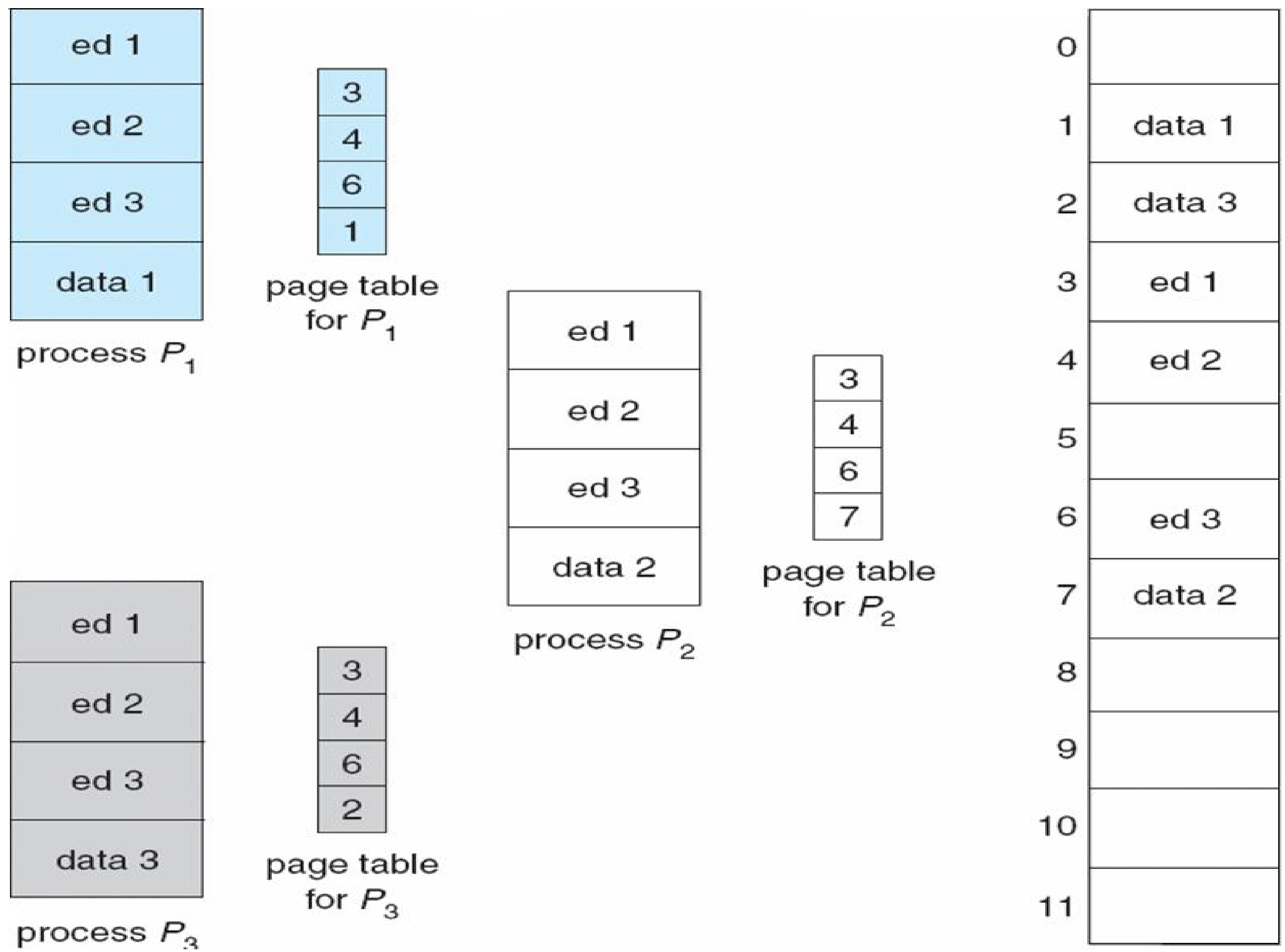
Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Each user's page table maps onto the same physical copy of the shared code.
- **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

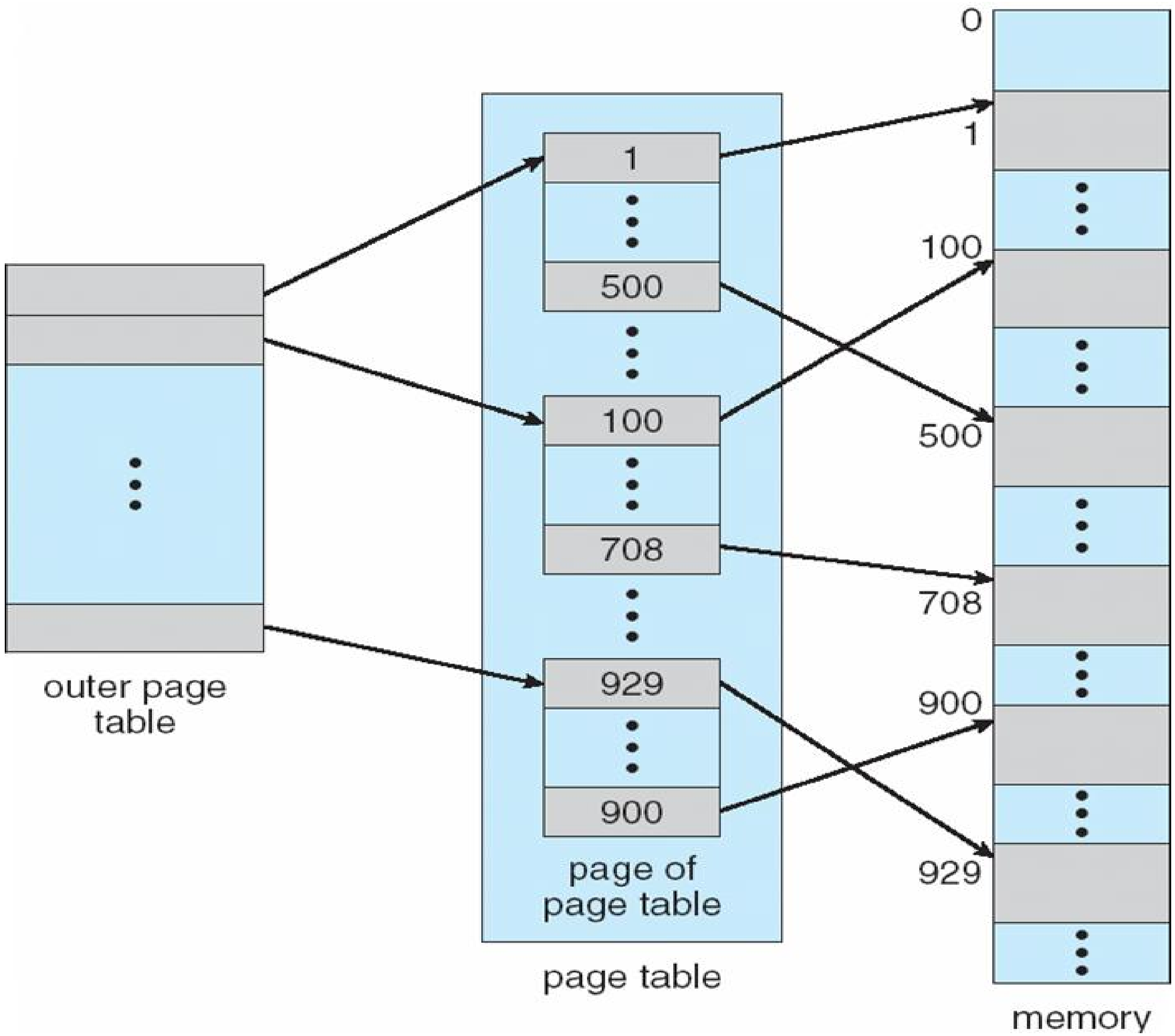
Hierarchical Page Tables

- Most modern computer systems support a large logical address space(2^{32} ---- 2^{64})
- The page table itself becomes excessively large.
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

Two-Level Page-Table Scheme

- Consider a system with a 32-bit logical address space(2^{32}B).
- If the page size is 4KB（2^{12}B）， then the page table may consist of up to I million entries(2^{20}). If each entry consists of 4 bytes, then a process may need up to 4MB of contiguous physical address space for the page table alone.
- Solution: divide the page table into small pieces.

Two-Level Page-Table Scheme



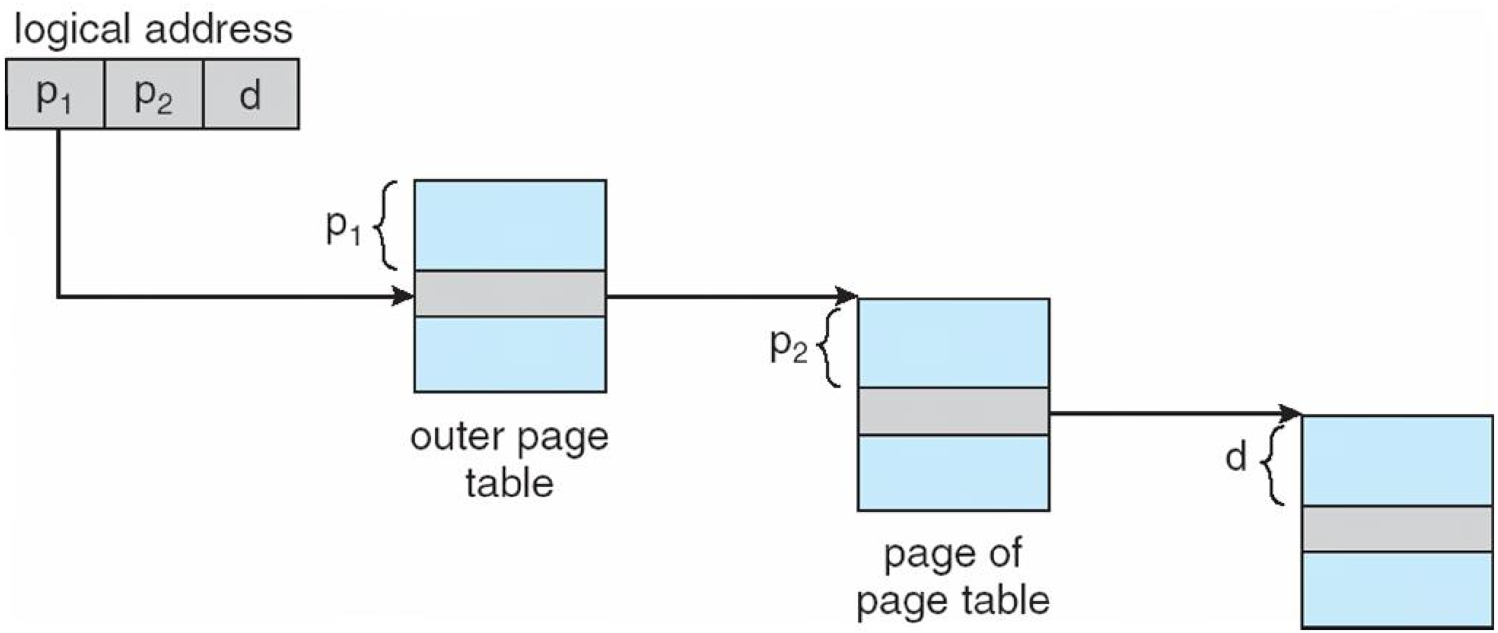
Two-Level Paging Example

- 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:

page number		page offset
p_i	p_2	d
12	10	10

- 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

Multilevel Paging and Performance

- Multilevel paging affects system performance.
- Given that each level is stored as a separate table in memory, converting a logical address to a physical one may take 4 memory access in a 4 level paging system.
- Cache hit rate of 98 percent yields:

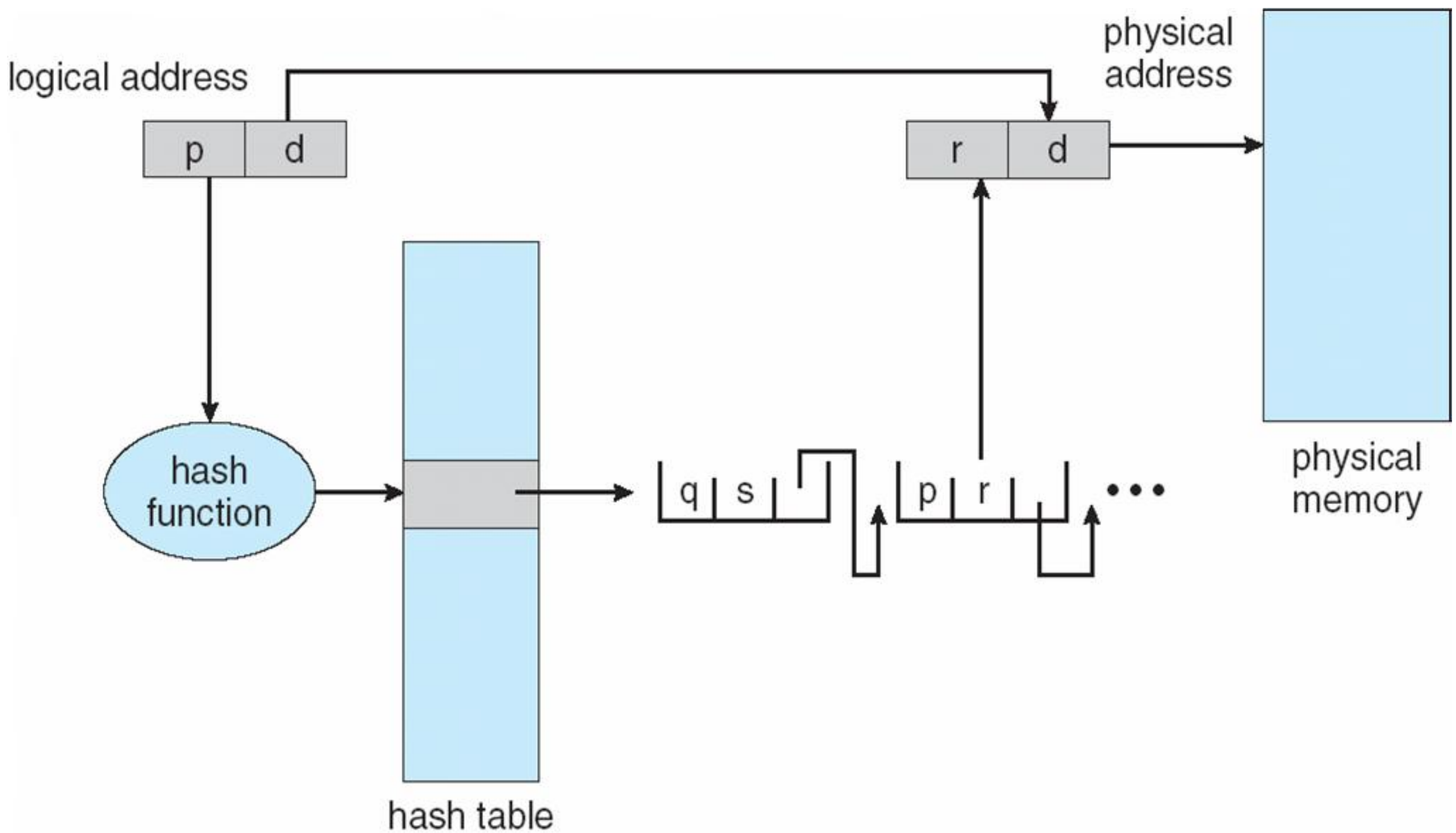
  effective access time = 0.98 x 120 + 0.02 x 520= 128 ns

- which is only a 28 percent slowdown in memory access time.

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

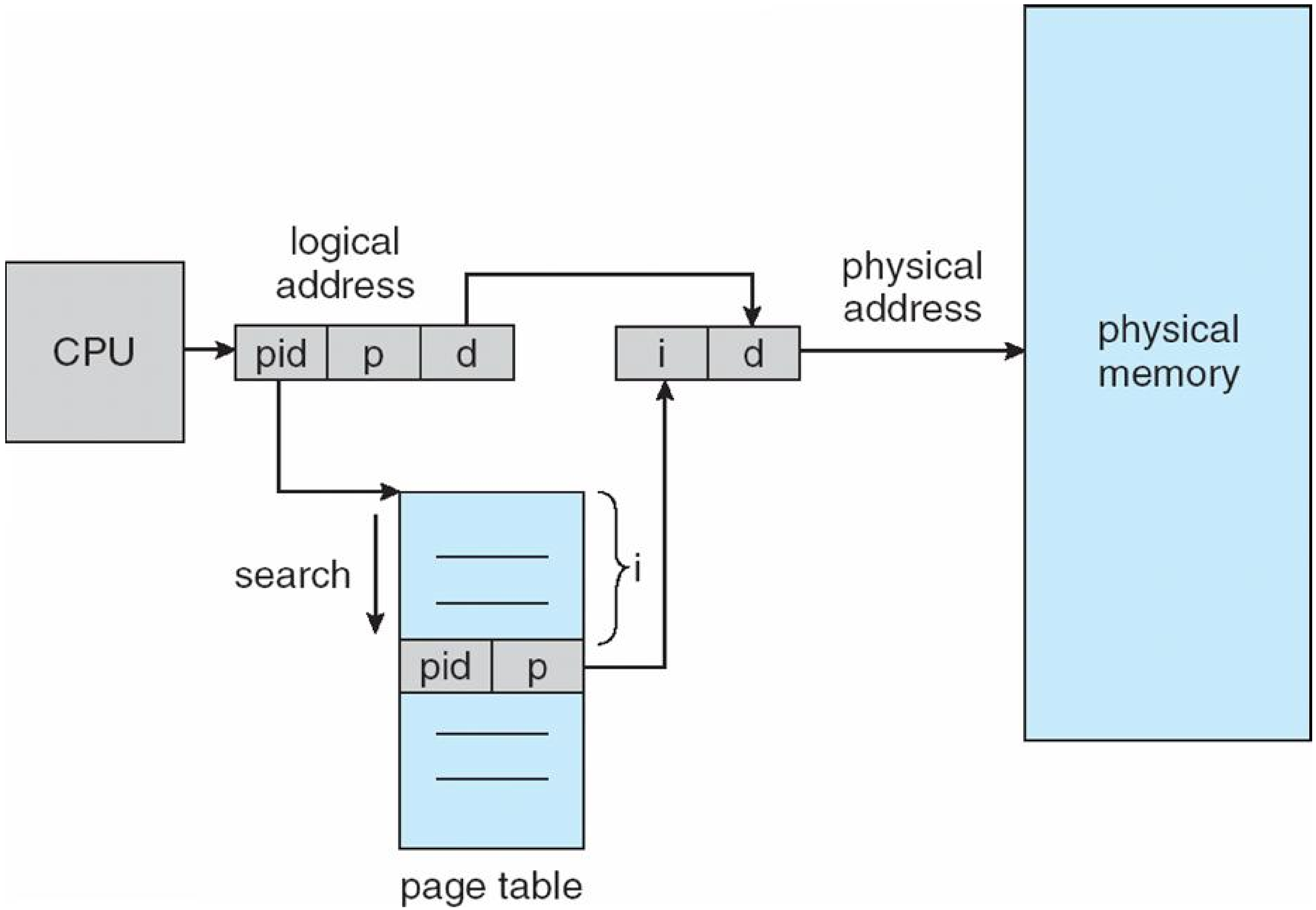
Hashed Page Table



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

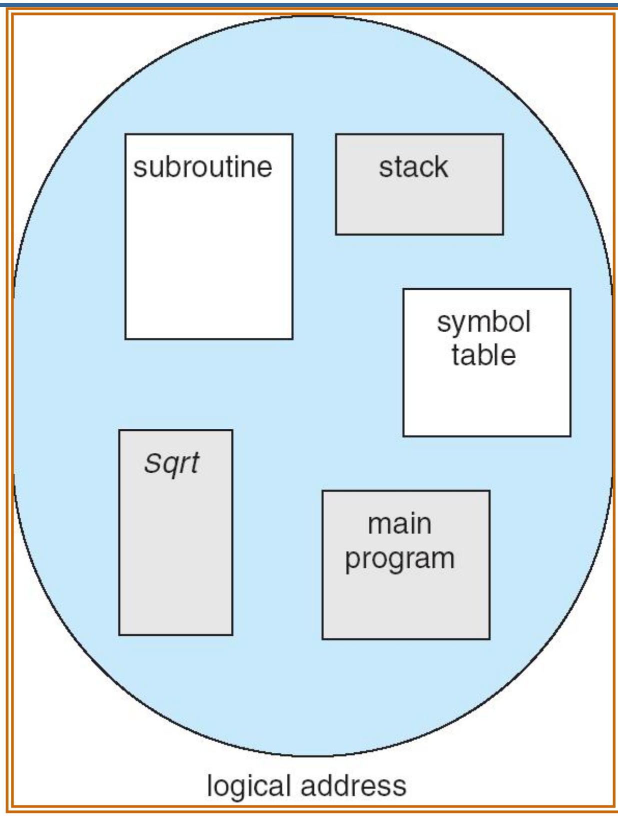
    common block

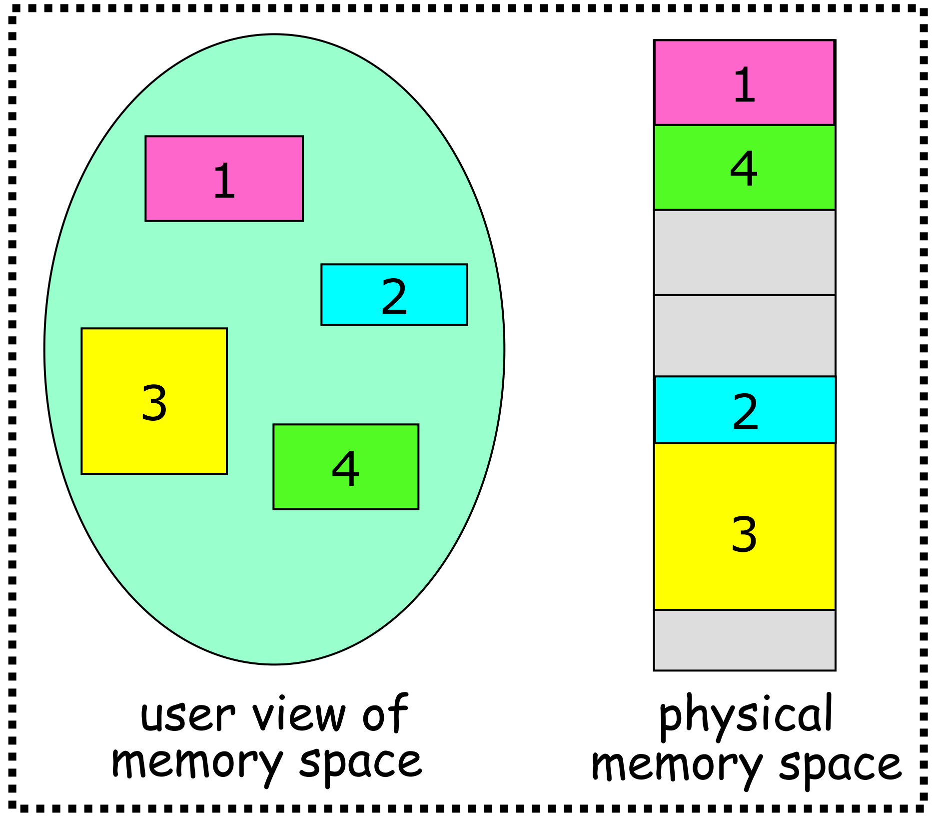
    stack

    symbol table

    arrays

More Flexible Segmentation





- Logical View: multiple separate segments
  - Typical: Code, Data, Stack
  - Others: memory sharing, etc
- Each segment is given region of contiguous memory
  - Has a base and limit
  - Can reside anywhere in physical memory

Segmentation Architecture

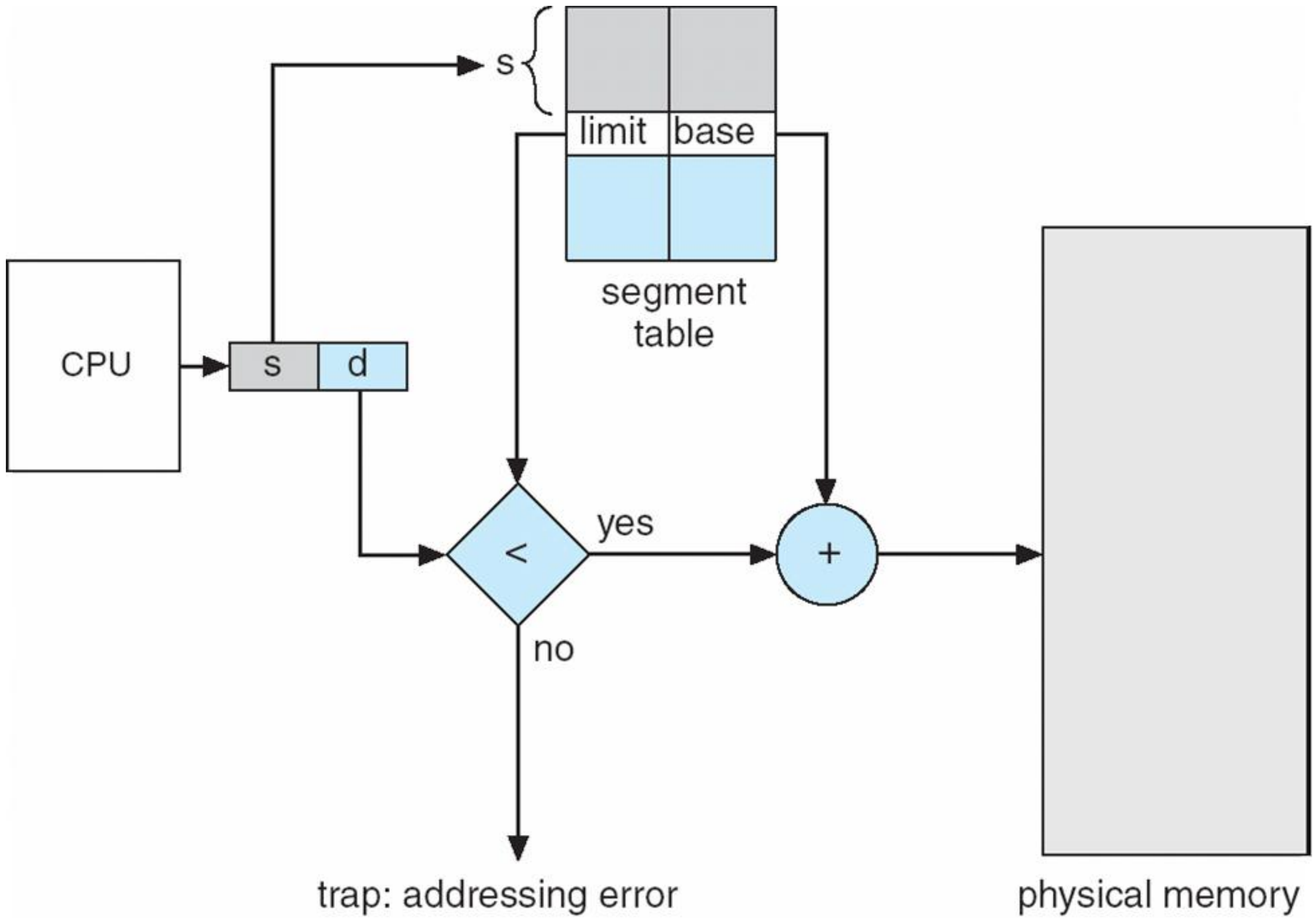
- Logical address consists of a two tuple:

  <segment-number, offset>,

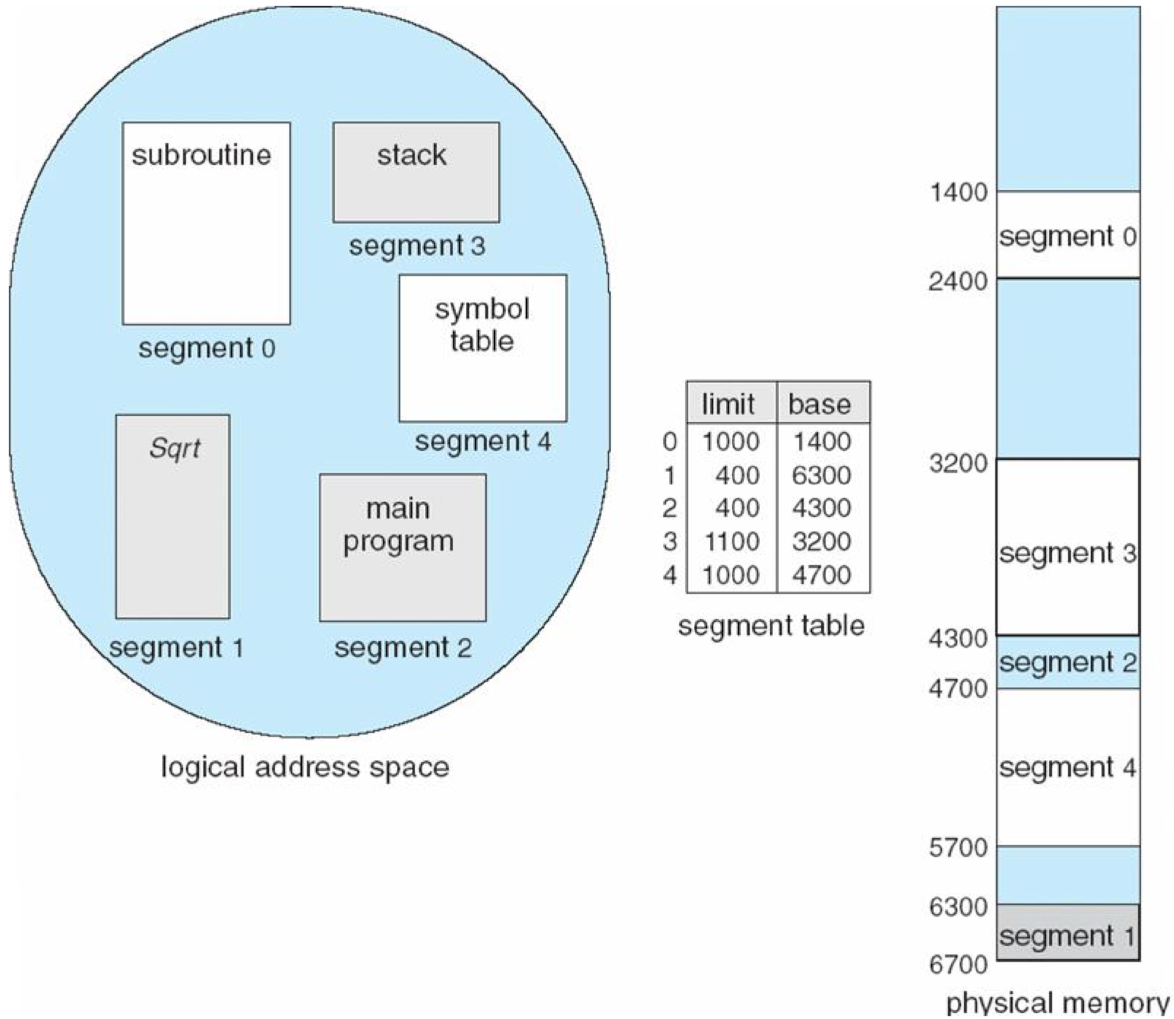
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

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

Segmentation Hardware



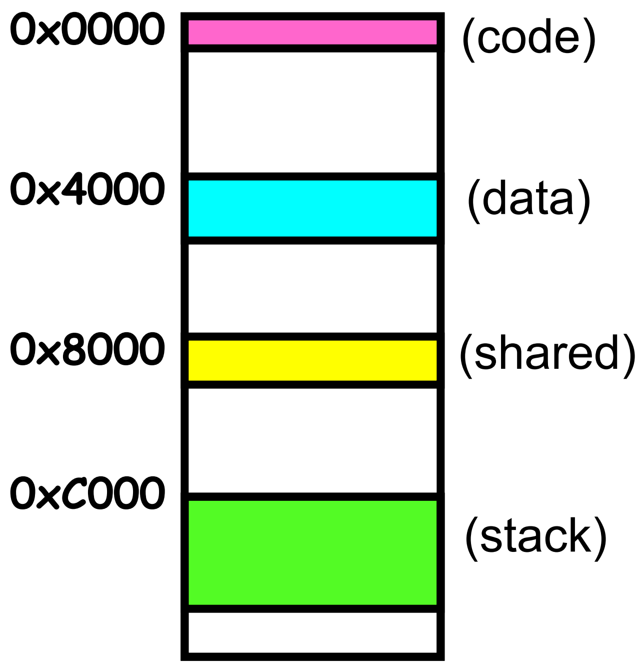
Example of Segmentation



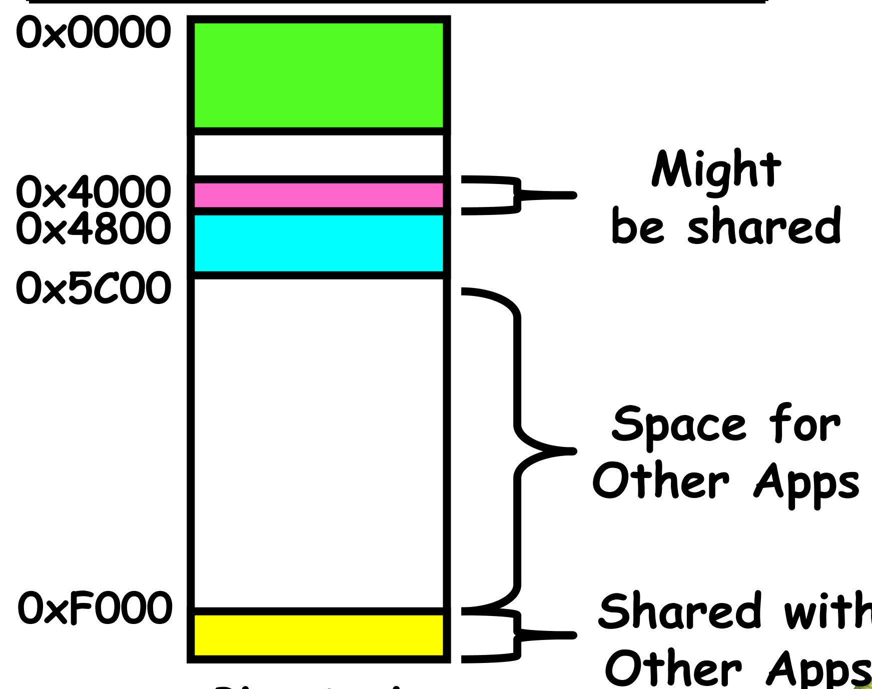
Example: Four Segments (16 bit addr)

Seg ID #	Base	Limit
0 (code)	0x4000	0x0800
1 (data)	0x4800	0x1400
2 (shared)	0xF000	0x1000
3 (stack)	0x0000	0x3000

Virtual Address Format



Virtual Address Space

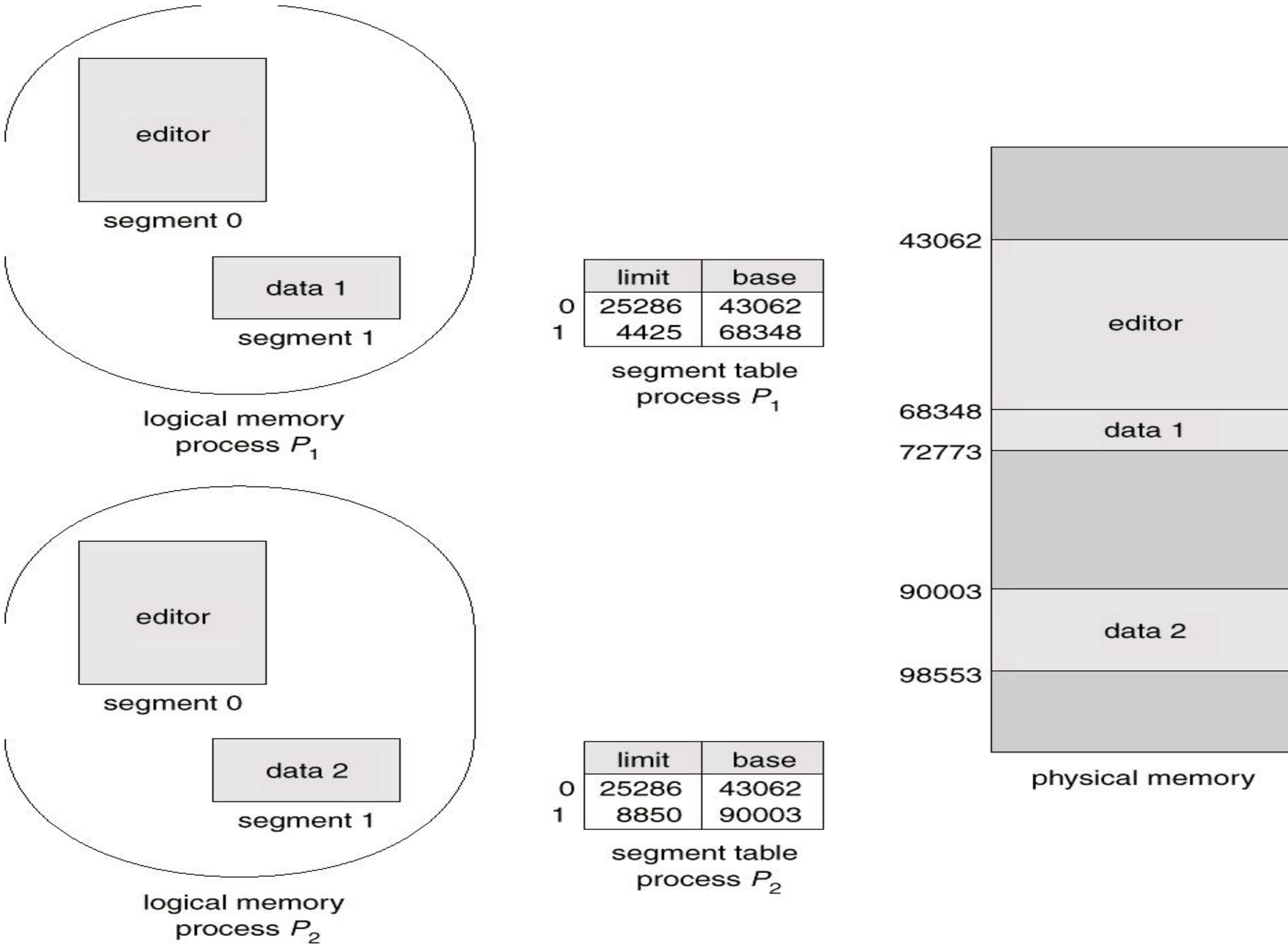


Physical Address Space

Segmentation Protection

- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level

Segmentation Sharing



Observations about Segmentation

- Relocation.
  - Since segments vary in length, memory allocation is a dynamic storage-allocation problem
  - by segment table
- Sharing.
  - shared segments: code segment, data segment
  - code segment : same segment number
- Allocation.
  - first fit/best fit

Segmentation with Paging

段号(S)	段内页号(P)	页内地址(W)

段表、页表与内存关系

段表地址寄存器

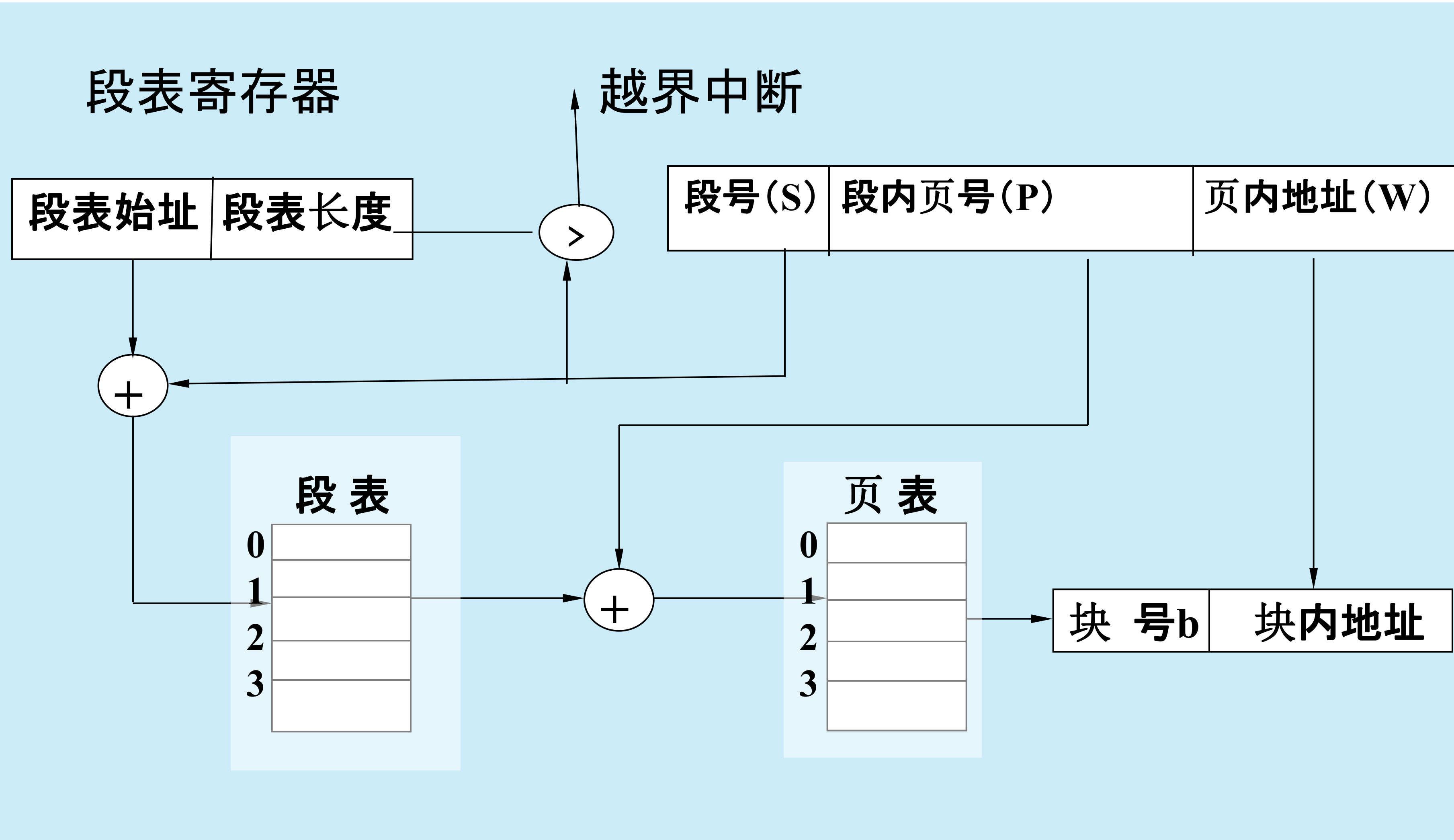
段表长度	起始地址

段号	其他	页表长度	起始地址
0		5	1024
1		7	1029
2		9	1036

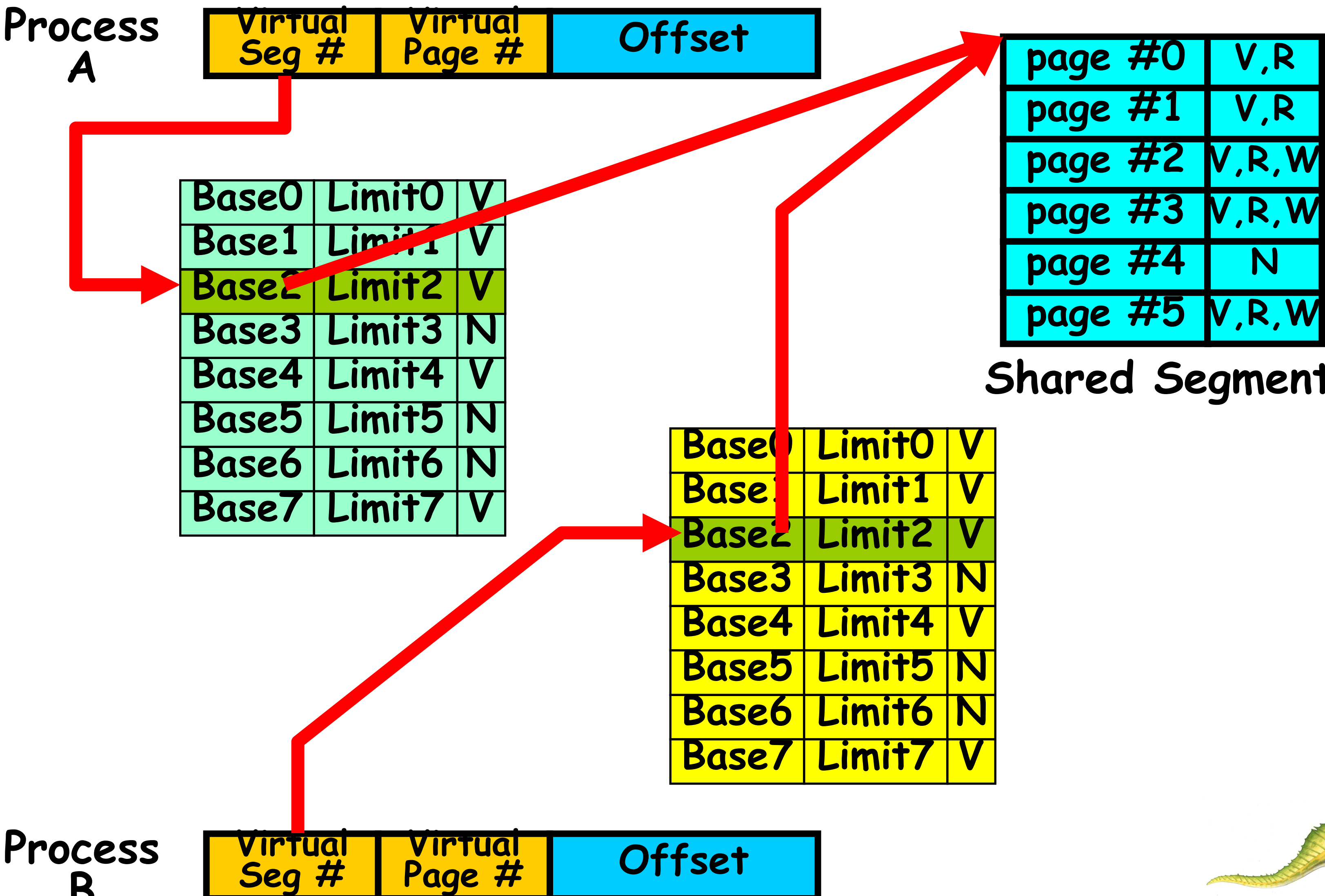
页号	其他	页面
1		12
2		19
3		21
4		8
5		10

页号	其他	页面
1		29
3		⋮

Address translation in Segmentation with Paging



What about Sharing (Complete Segment)?

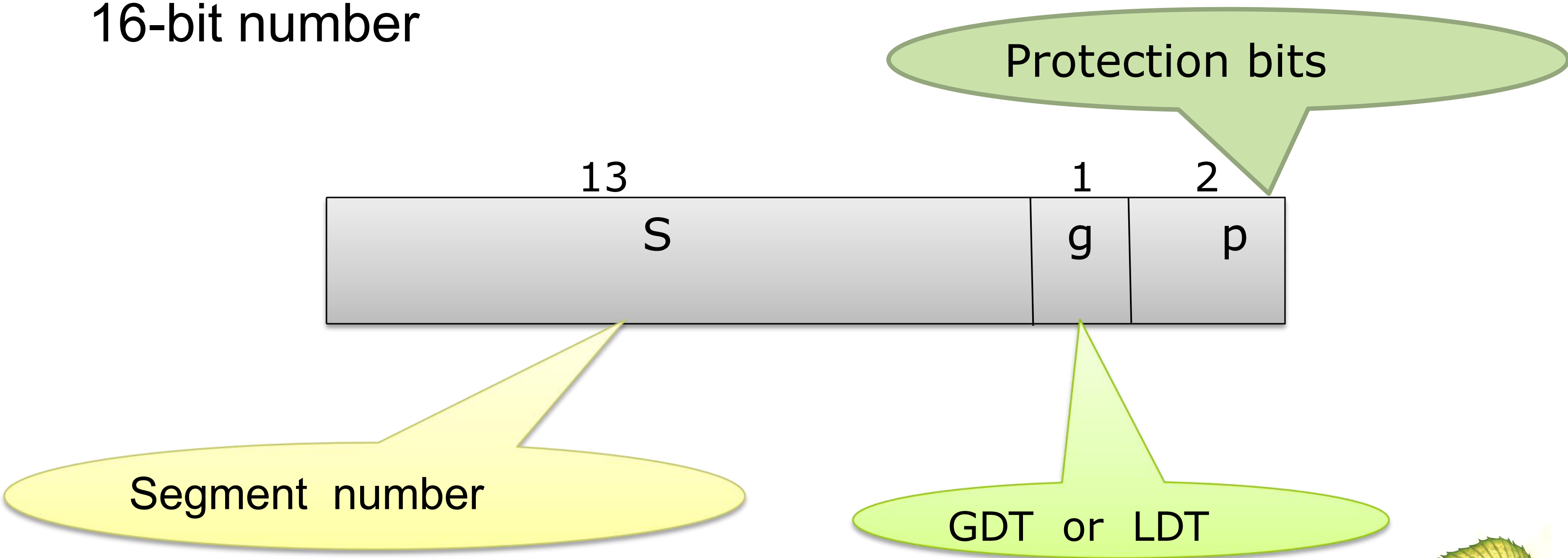


Example: The Intel Pentium

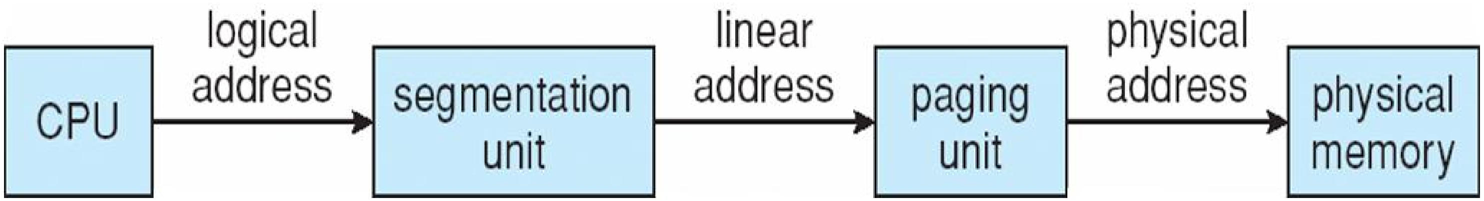
- Supports both segmentation and segmentation with paging
- CPU generates logical address
  - Given to segmentation unit
    - Which produces linear addresses
  - Linear address given to paging unit
    - Which generates physical address in main memory
    - Paging units form equivalent of MMU

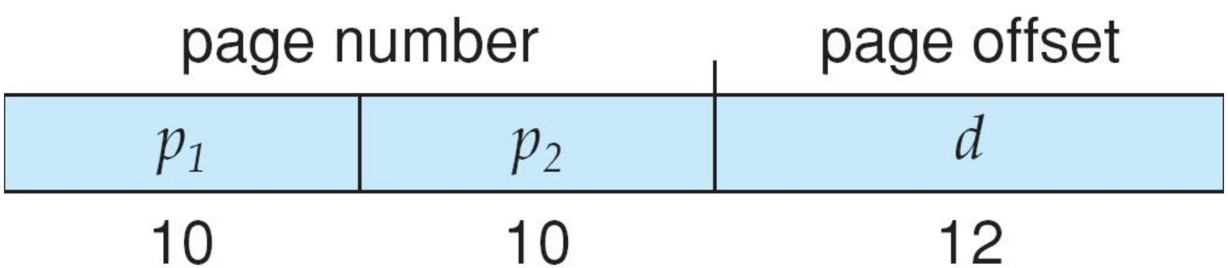
Example: The Intel Pentium

- The maximum number of segments per process is 16K
- Allows a segment to be as large as 4 GB
- The page size is 4KB
- The logical address is a pair (selector, offset), selector is a 16-bit number

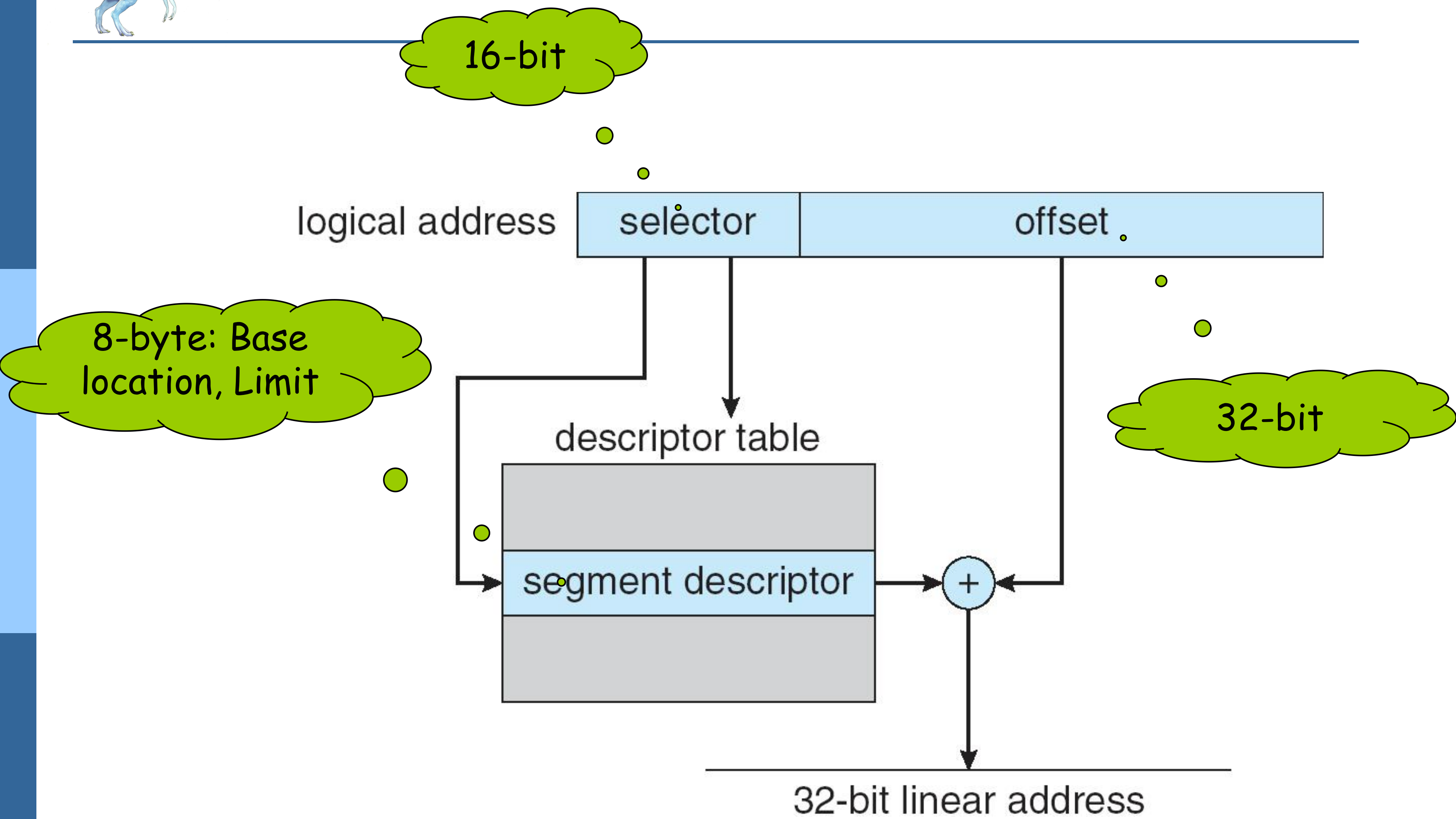


Logical to Physical Address Translation in Pentium

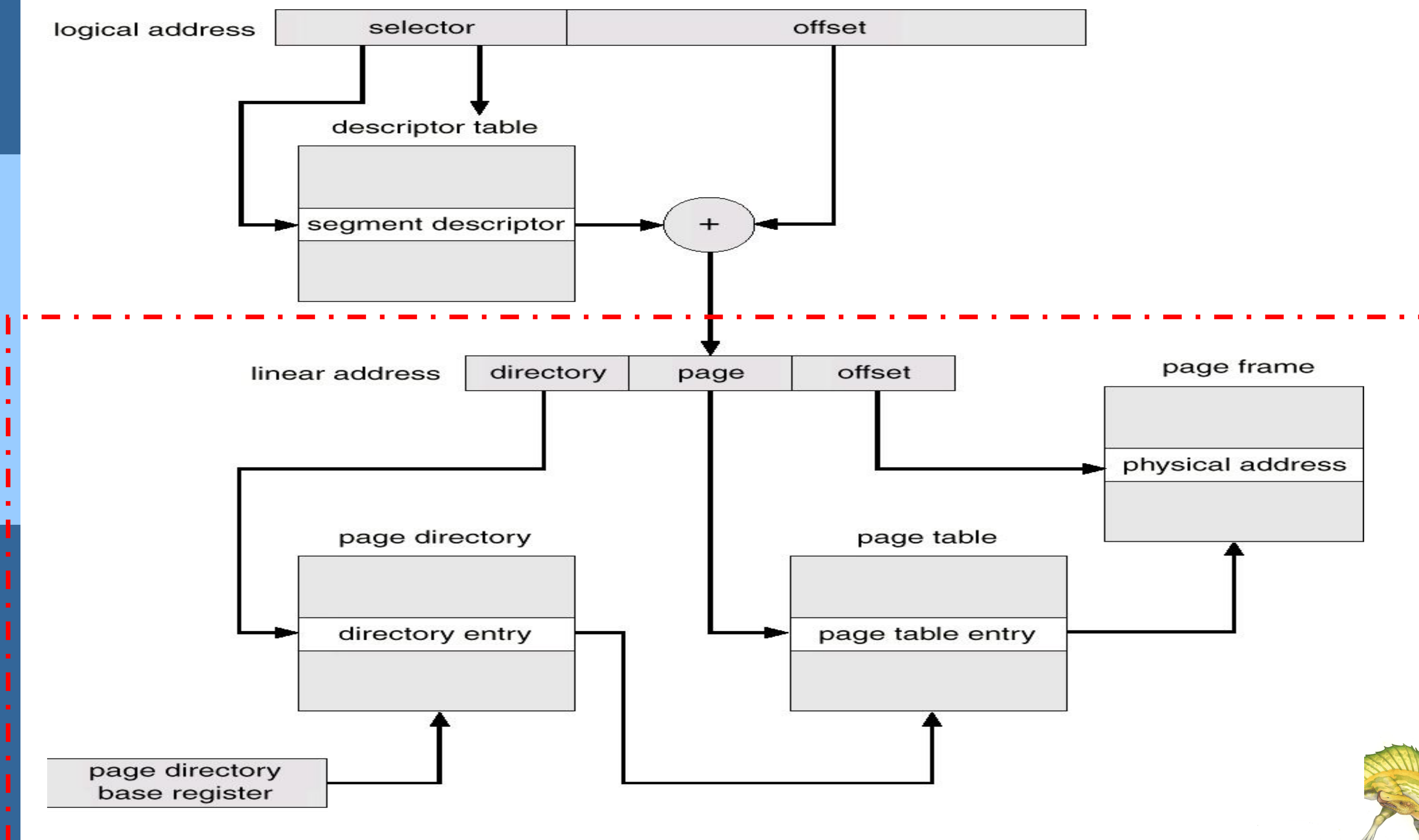




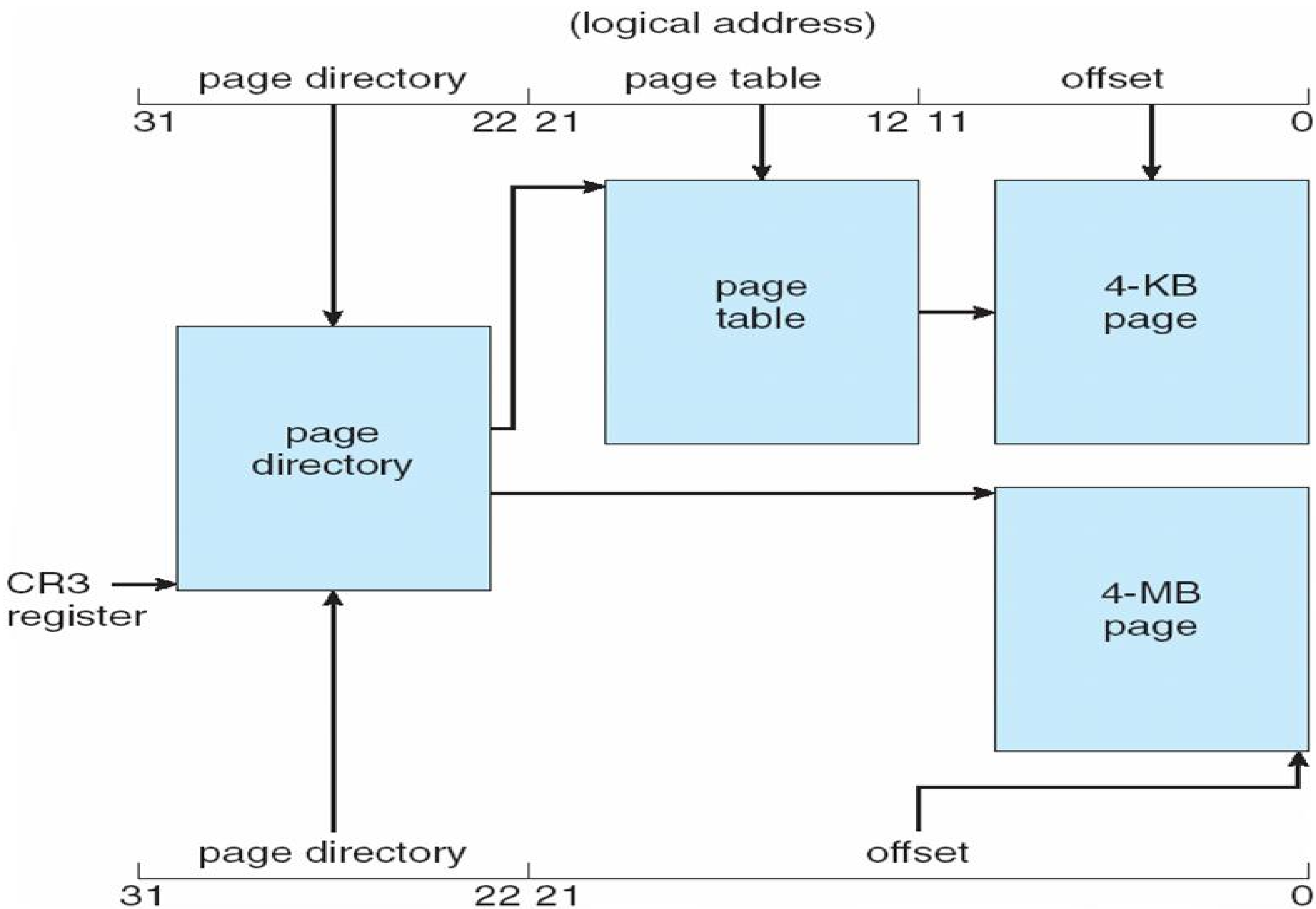
Intel Pentium Segmentation



Pentium Paging Architecture



Pentium Paging Architecture



Linear Address in Linux

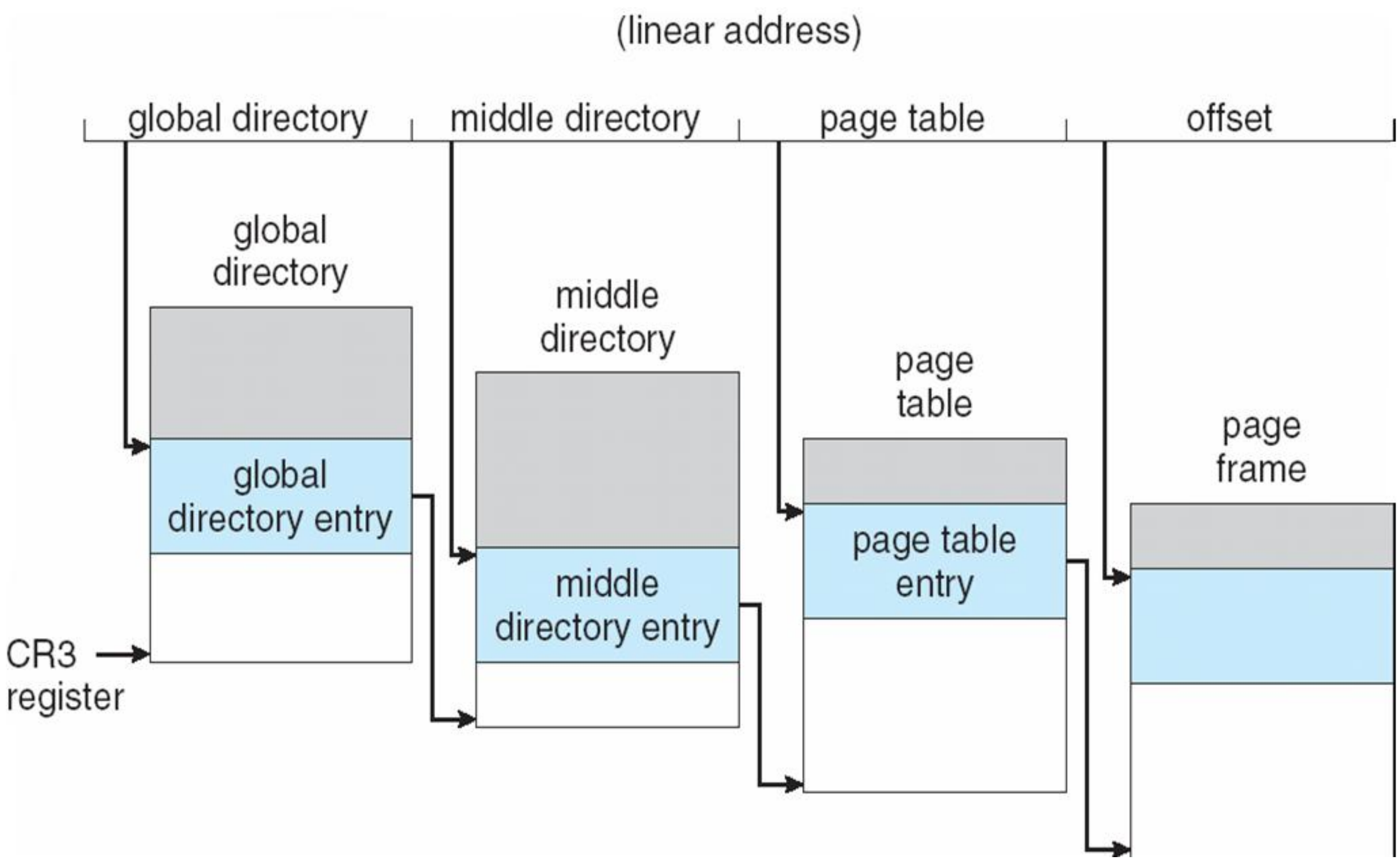
Broken into four parts to work well for both 32-bit and 64-bit architectures

The number of bits in each part of the linear address varies according to the architecture

global directory	middle directory	page table	offset

On Pentium system, the size of the middle directory is zero bits.

Three-level Paging in Linux



assignment

- P 310-312
  - 8.1
  - 8.3
  - 8.9
  - 8.12

End of Chapter 8