Chapter 8: Main Memory

课前习题

桌上有一个空盘，允许存放一只水果。爸爸可以向盘中放苹果，也可向盘中放桔子，儿子专等吃盘中的桔子，女儿专等吃盘中的苹果。规定当盘空时一次只能放一只水果供吃者使用，请用P、V原语实现爸爸、儿子、女儿三个并发进程的同步。提示：生产者、消费者问题的变形

```
Binary_semaphore S=1;
Binary_semaphore Sa=0;
Binary_semaphore So=0;
Main()
{
begin
    Father()
    Son()
    Daughter();
end;
}
```

```
Father()
{
While(1)
{
P(S);
放水果到盘中
;
If(放的是桔子)
        V(So))
Else
        V(Sa);
}
}
```

```
son()
{
While(1)
{
P(So);
取桔子
V(S)
吃桔子
}
}
```

```
daughter()
{
While(1)
{
P(Sa);
取苹果
V(S)
吃苹果
}
}
```

课前习题

假定某计算机系统有R1和R2两类可再使用资源，其中R1有两个单位，R2有一个单位，它们被进程P1和P2所共享，且已知两个进程均以下列顺序使用两类资源：

--申请R1—申请R2—申请R1—释放R1—释放R2—释放R1—

试求出系统运行过程中可能到达的死锁点，并画出死锁点的资源分配图。

课前习题

假定某计算机系统有R1和R2两类资源，其中R1有两个单位，R2有一个单位，它们被进程P1和P2所共享，且已知两个进程均以下列顺序使用两类资源：

--申请R1—申请R2—申请R1—释放R1—释放R2—释放R1—

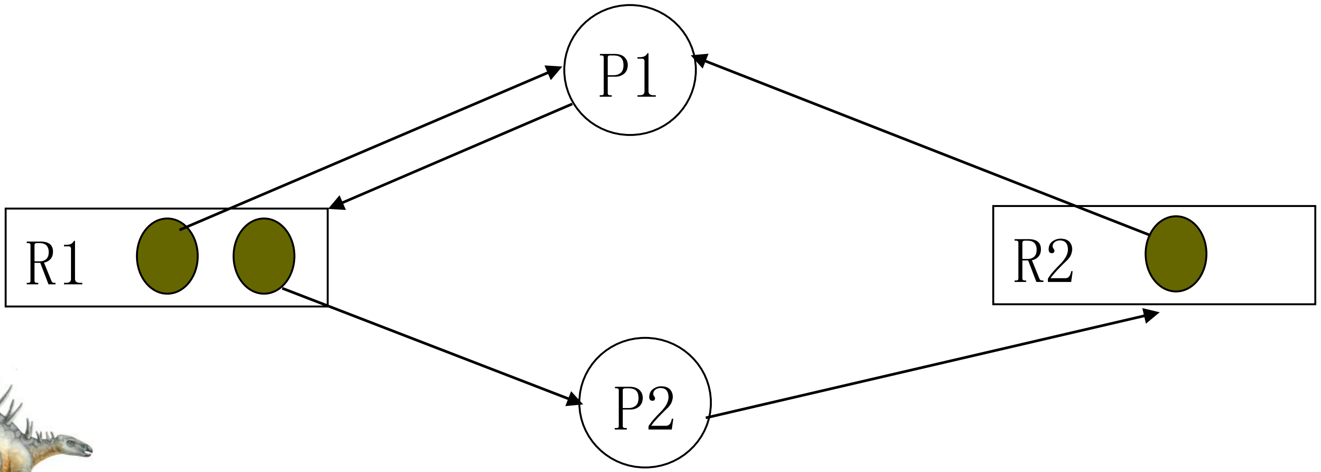
试求出系统运行过程中可能到达的死锁点，并画出死锁点的资源分配图。

本题中，当两个进程都执行完第一步后，即进程P1和P2都申请到了一个R1类资源时，系统进入不‘安全状态。随着两个进程的向前推进，无论哪个进程执行完第二步，系统都将进入死锁状态。

可能的到达的死锁点是：

P1占有一个单位R1及一个单位R2，P2占有一个单位R1。

P2占有一个单位R1及一个单位R2，P1占有一个单位R1。



Chapter 8: Memory Management

- Background
- 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
- Protection of memory required to ensure correct operation

Logical vs. Physical Address Space

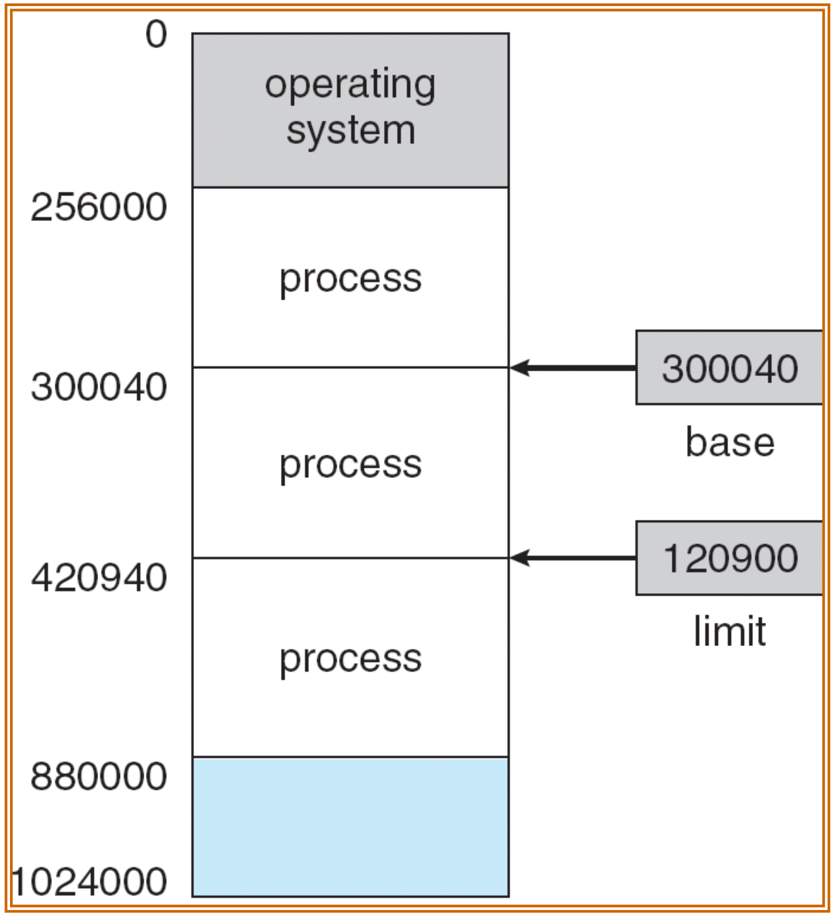
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical 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)

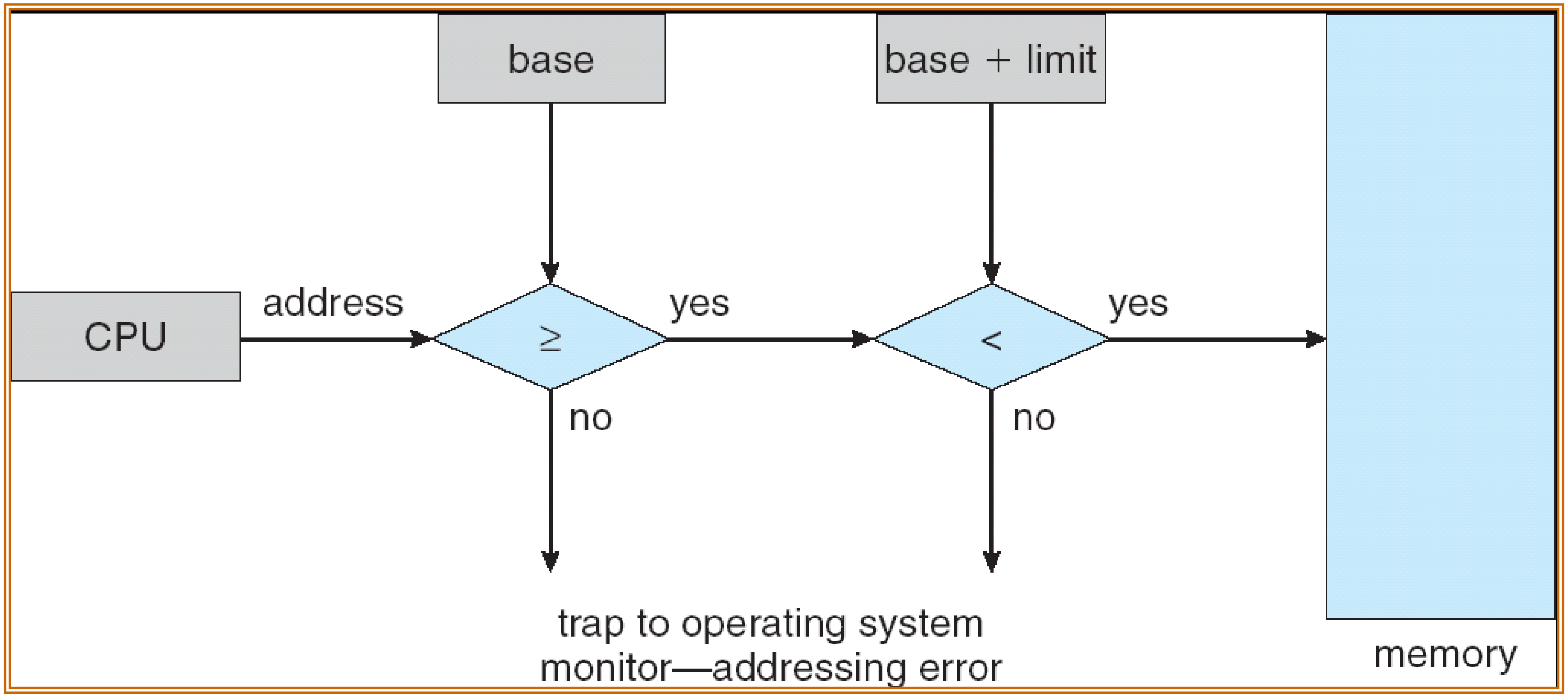
- Hardware device that maps virtual to physical address
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



HW address protection with base and limit registers

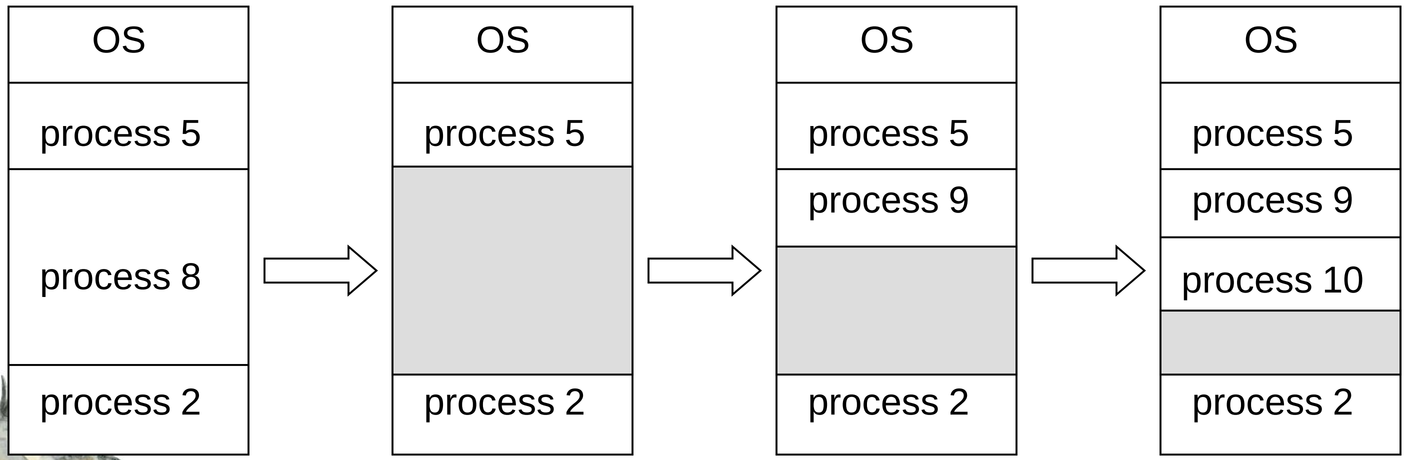


Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
- 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*

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

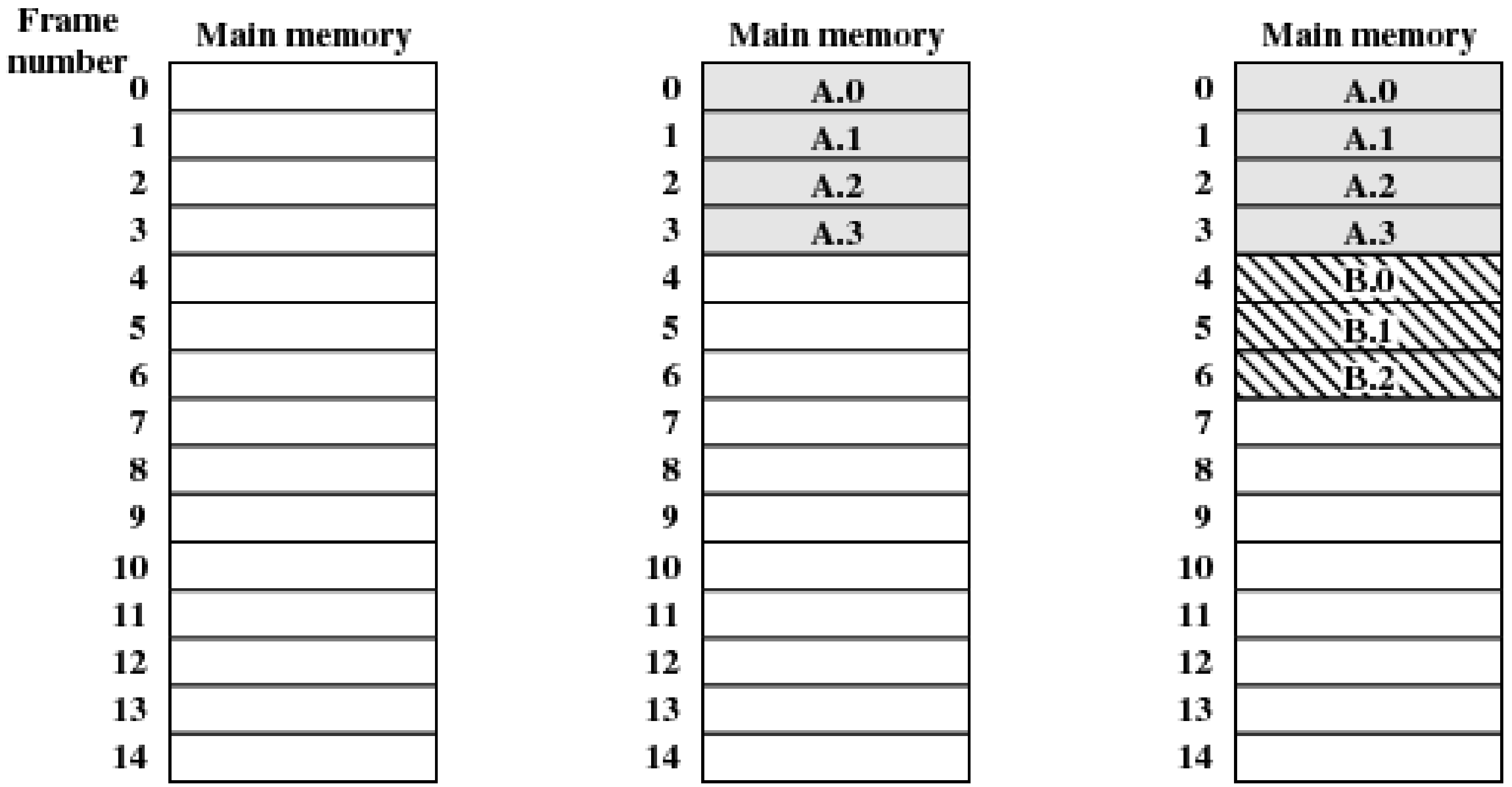
分页技术

大小不等的固定分区和大小可变的分区效率低下，会产生内部或外部碎片。若将主存划分为**固定大小块**，并且块较小，每个进程也被分为同样大小的小块，可以将进程的块（称为页）指定到内存中的可用块（称为帧或页帧），最后一页可能形成内部碎片，没有外部碎片。

Paging

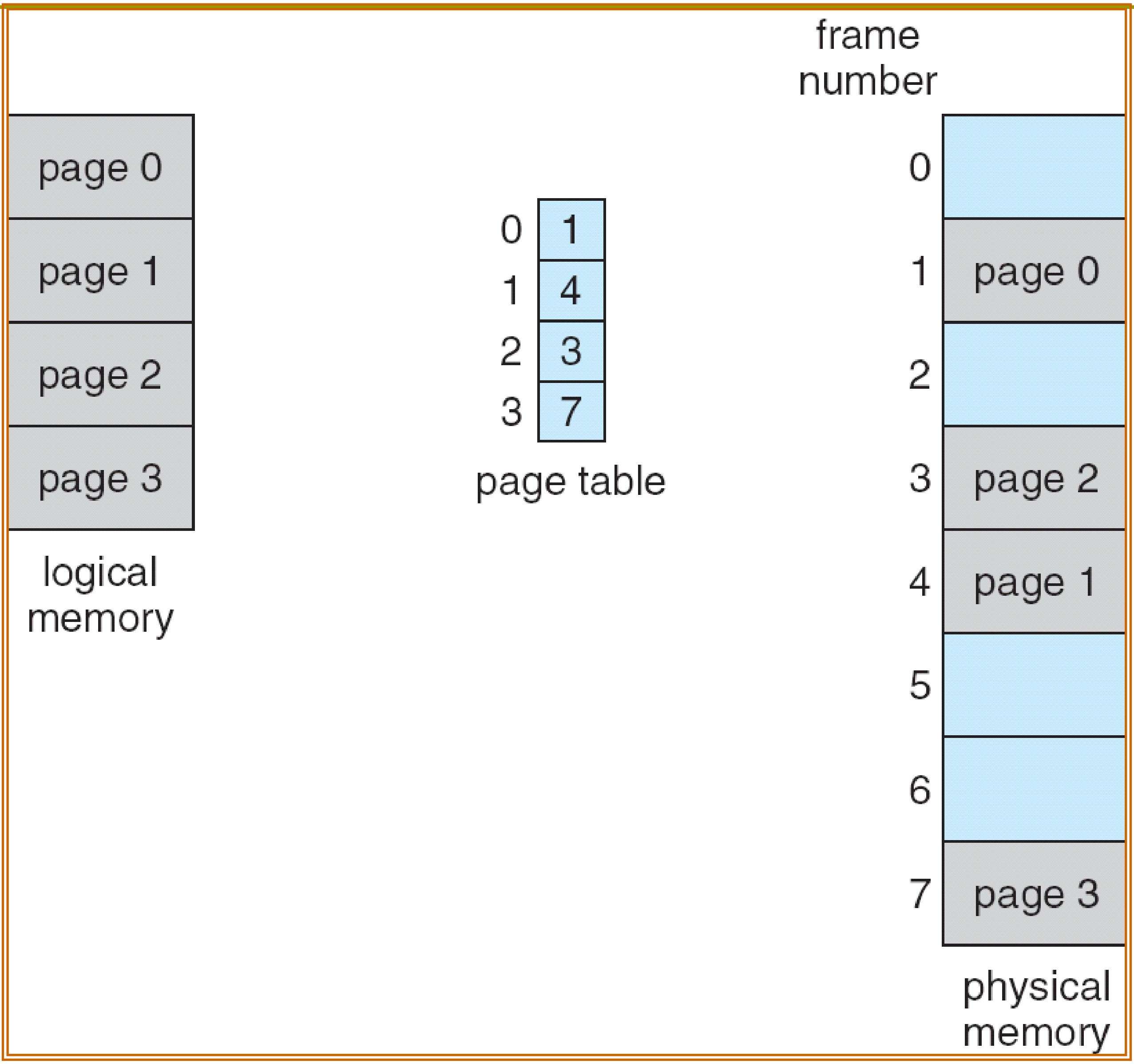
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 8,192 bytes)
- Divide logical memory into blocks of same size 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

分页技术

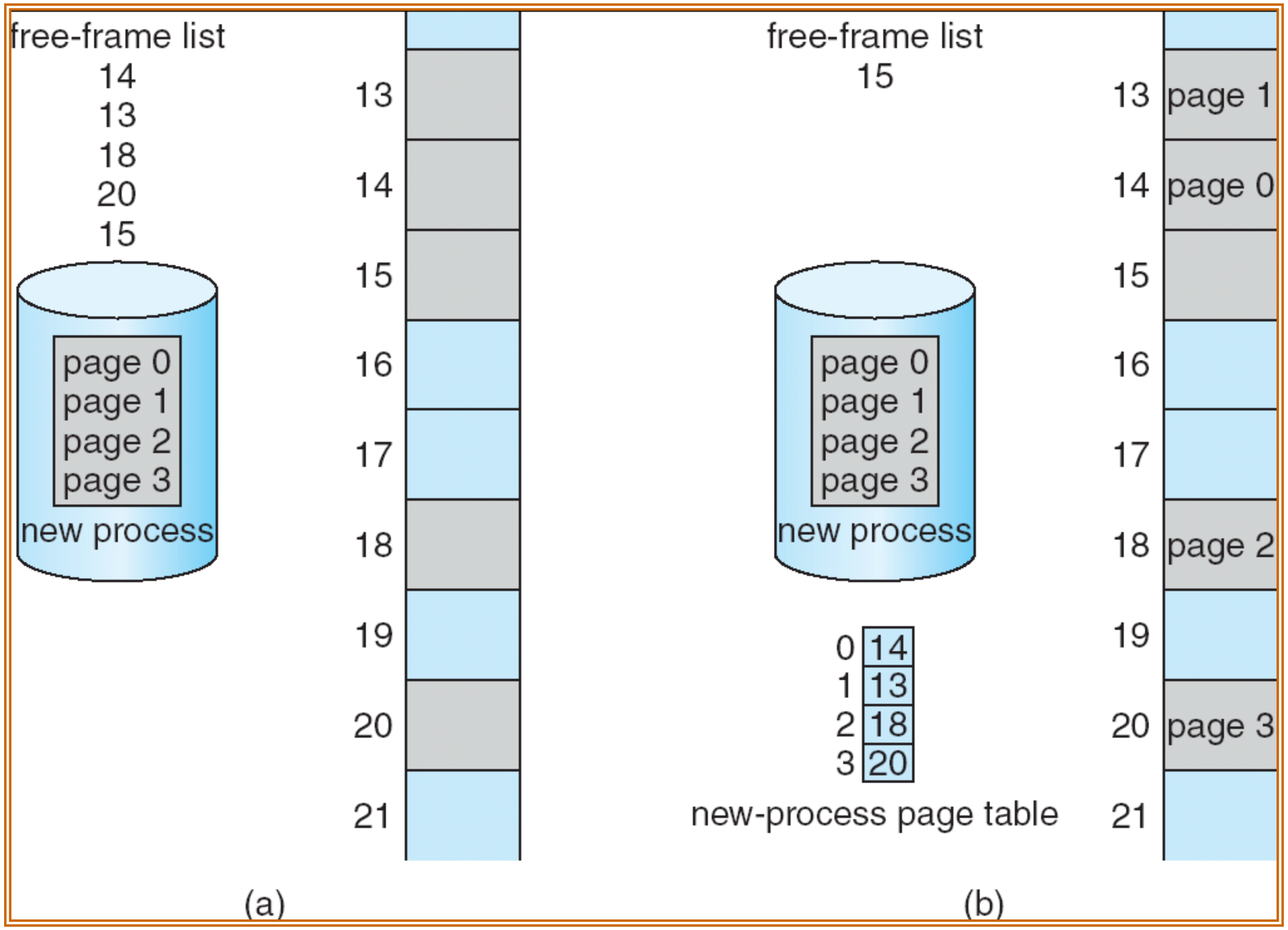


(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

Paging Model of Logical and Physical Memory



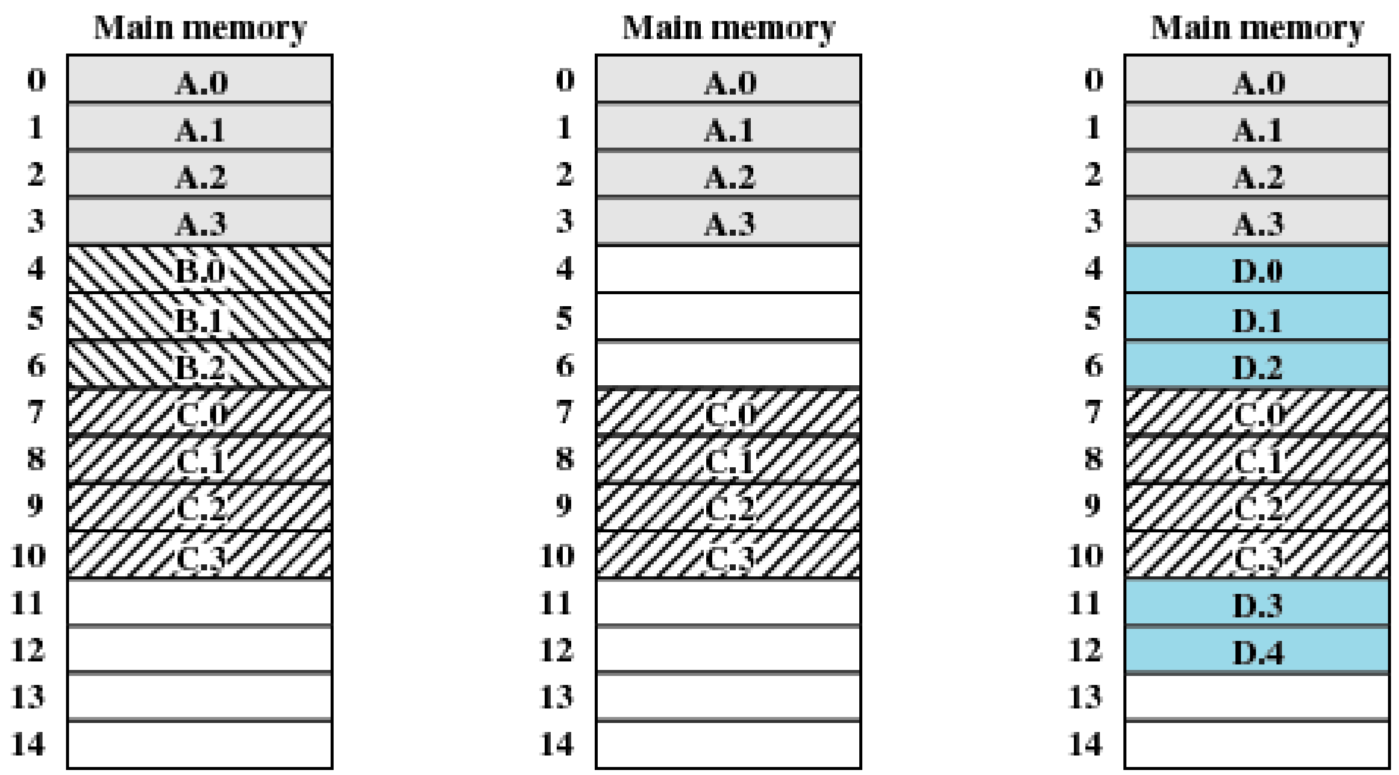
Free Frames



Before allocation

After allocation

分页技术



(d) Load Process C (e) Swap out B (f) Load Process D

0	0
1	1
2	2
3	3

Process A page table

0	N
1	N
2	N

Process B page table

0	7
1	8
2	9
3	10

Process C page table

0	4
1	5
2	6
3	11
4	12

Process D

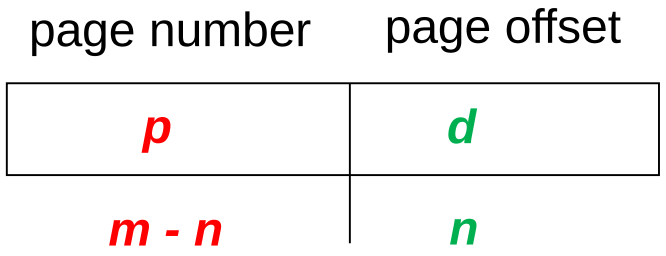
13
14

Free frame list

页表

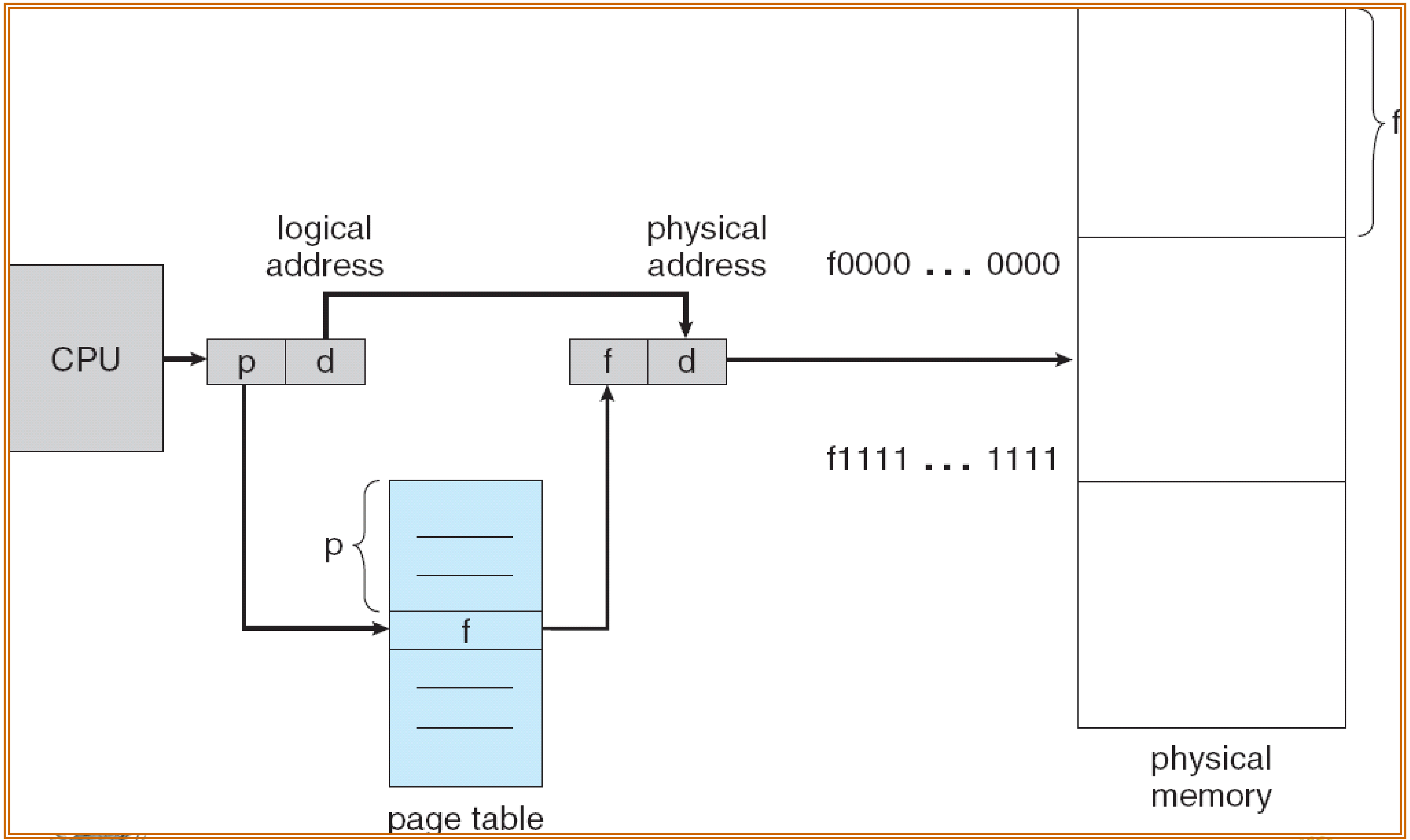
Address Translation Scheme

- 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

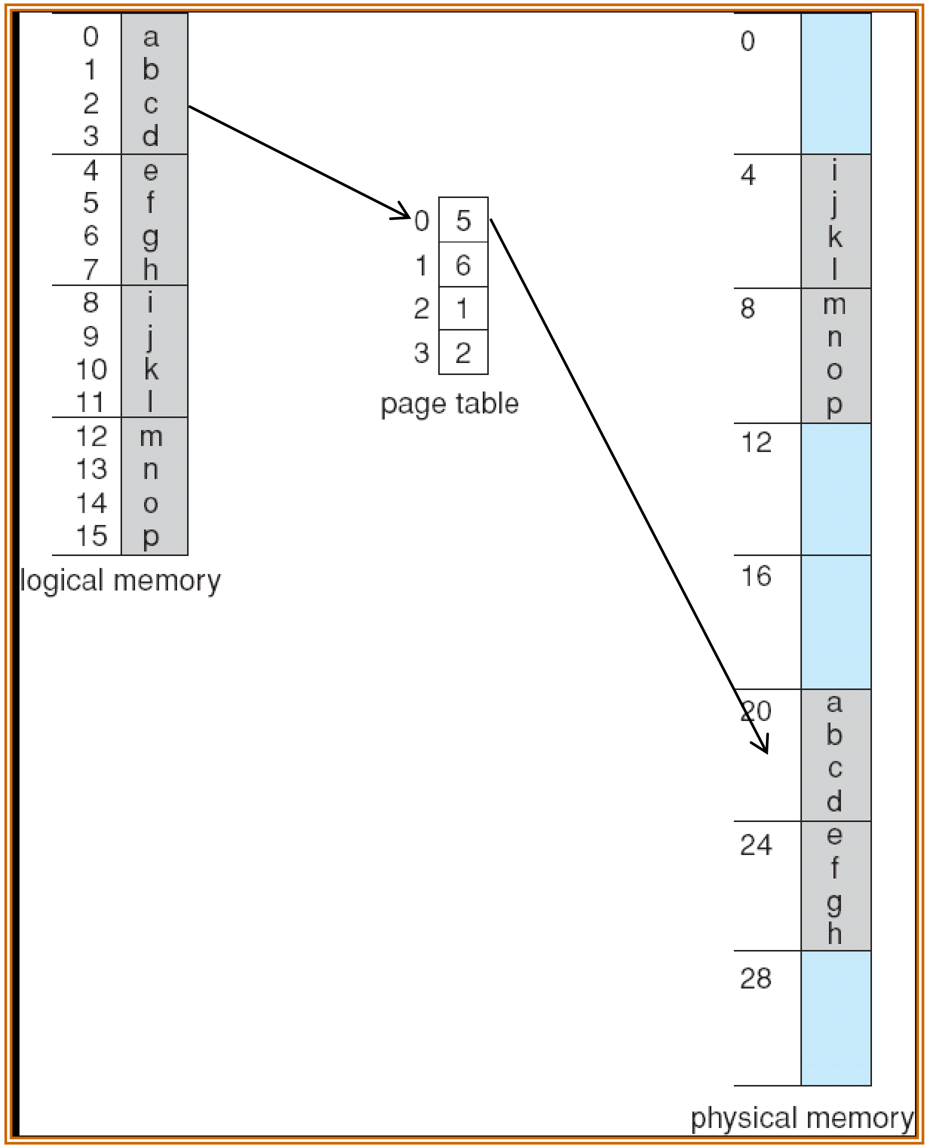


  - For given logical address space 2^m *and page size* 2^n

Paging Hardware



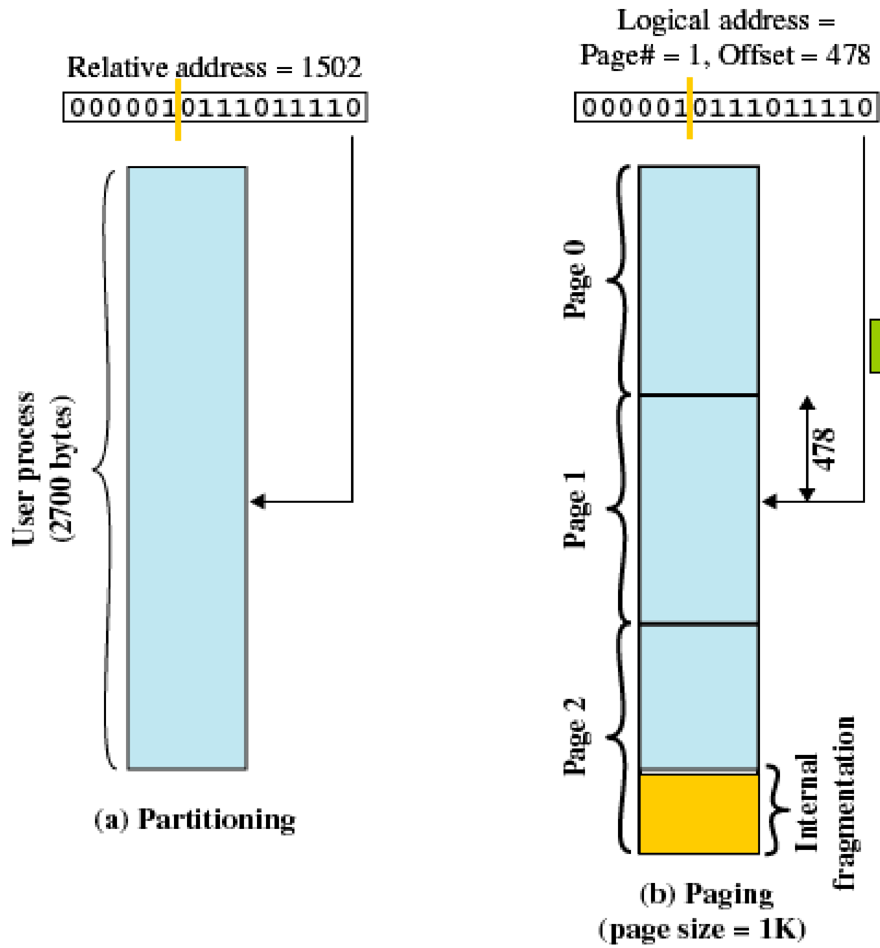
Paging Example

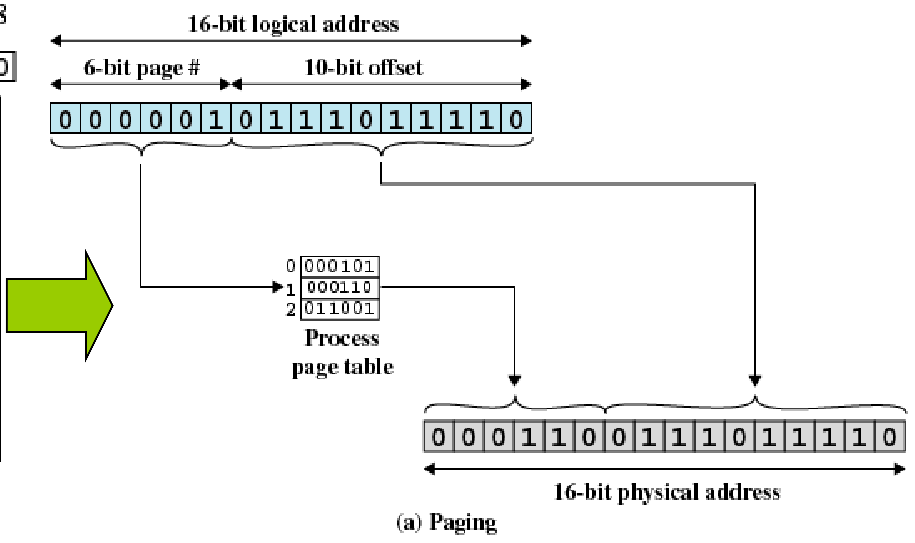


32-byte memory and 4-byte pages

分页技术

- 为了方便使用，规定页的大小及帧的大小必须是2的幂，以便容易表示出相对地址；
- 相对地址由程序的起点和逻辑地址定义，可以用页号和偏移量表示。





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

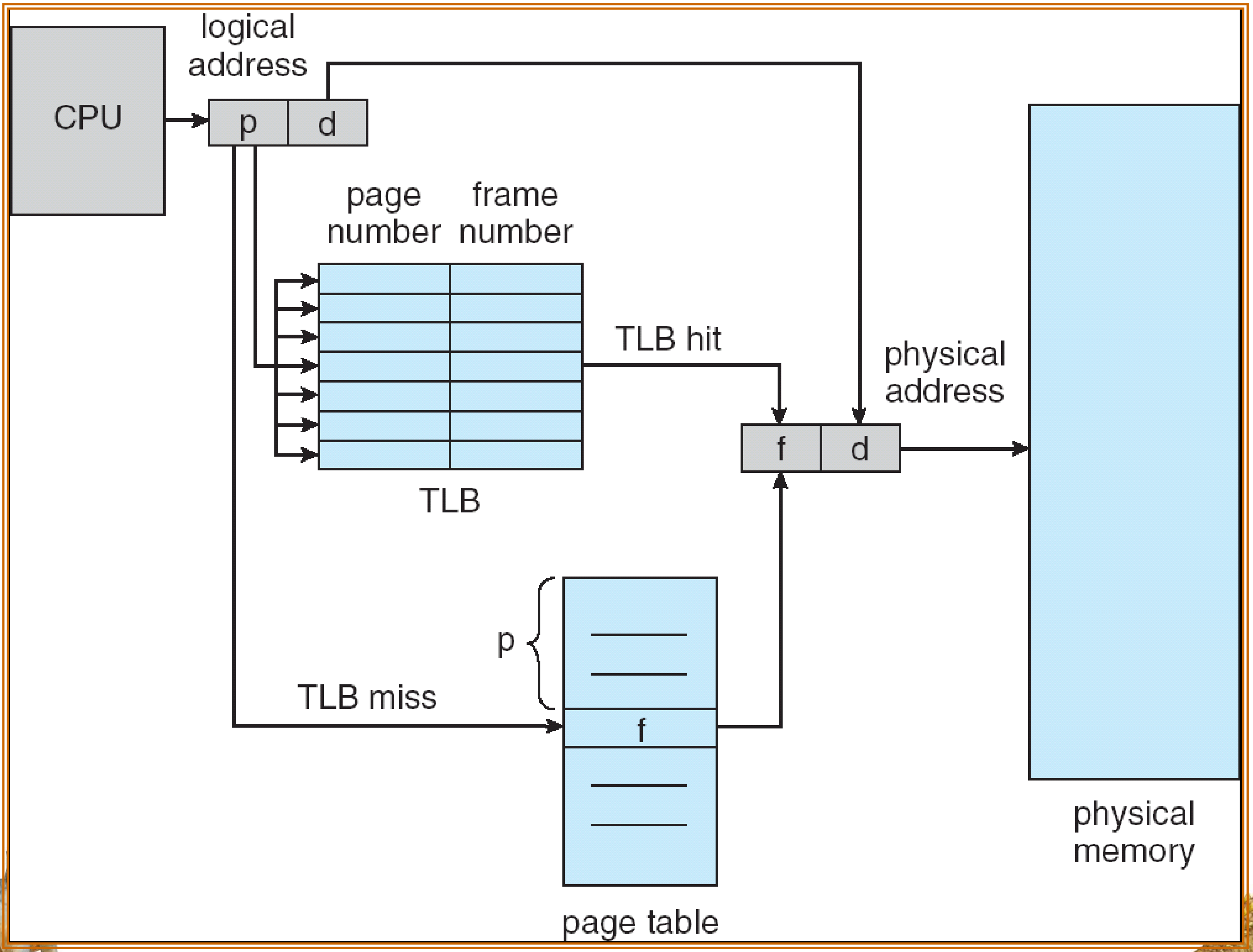
- Associative memory – parallel search

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



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)

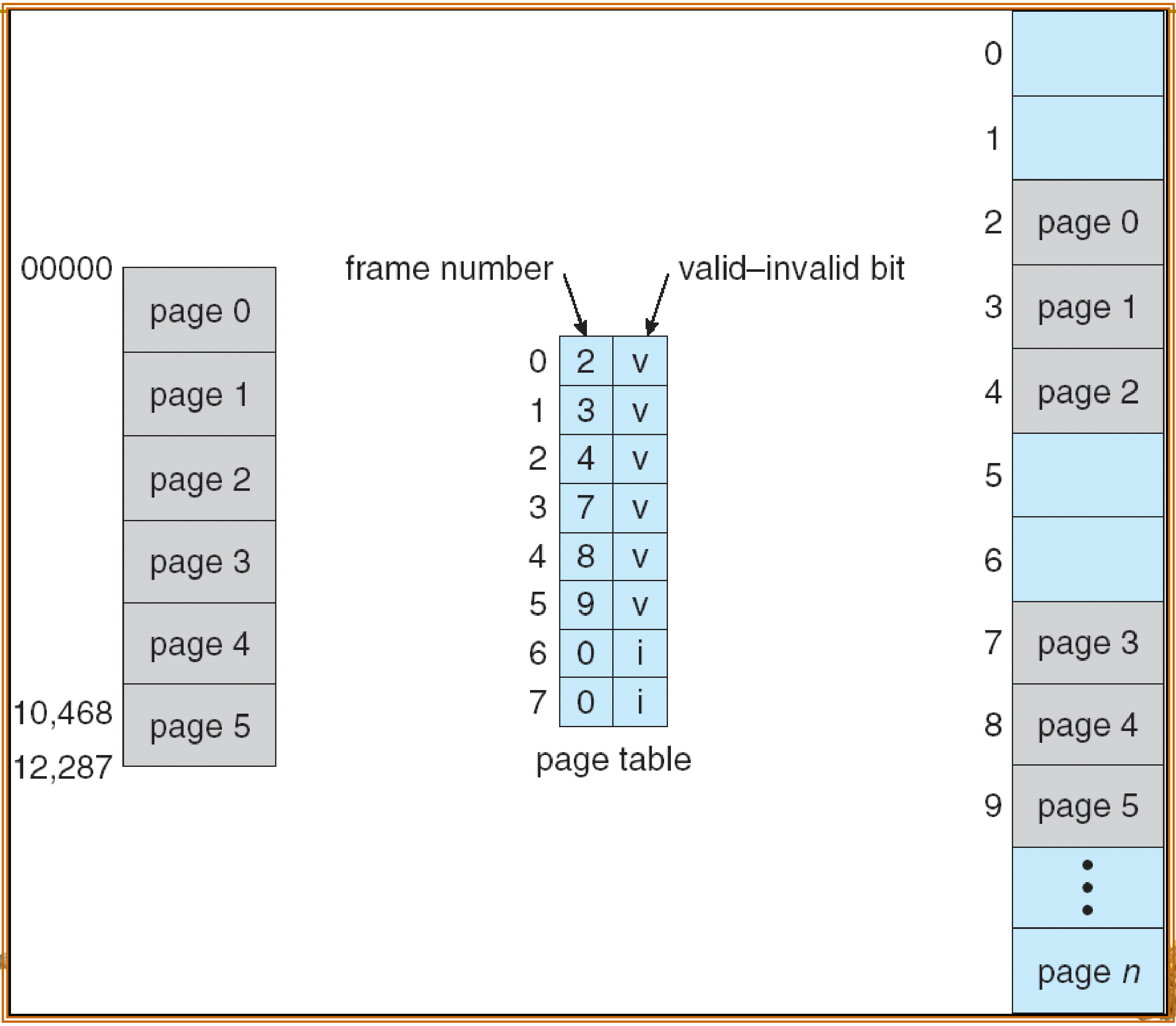
$$\begin{aligned} \text{EAT} &= (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha) \\ &= 2 + \varepsilon - \alpha \end{aligned}$$

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



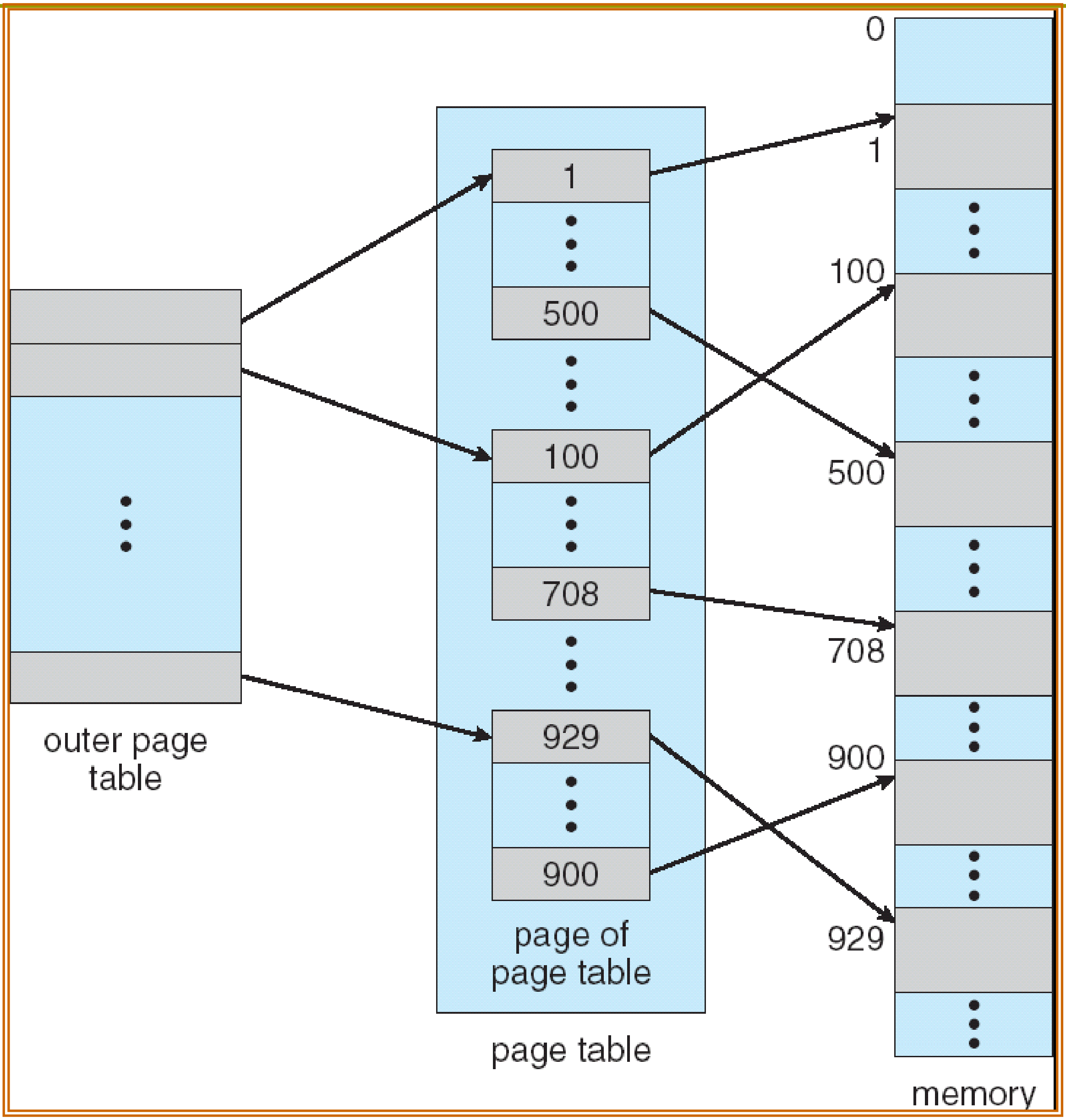
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

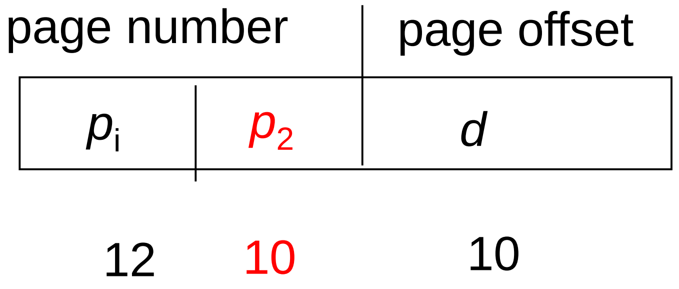
- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

Two-Level Page-Table Scheme



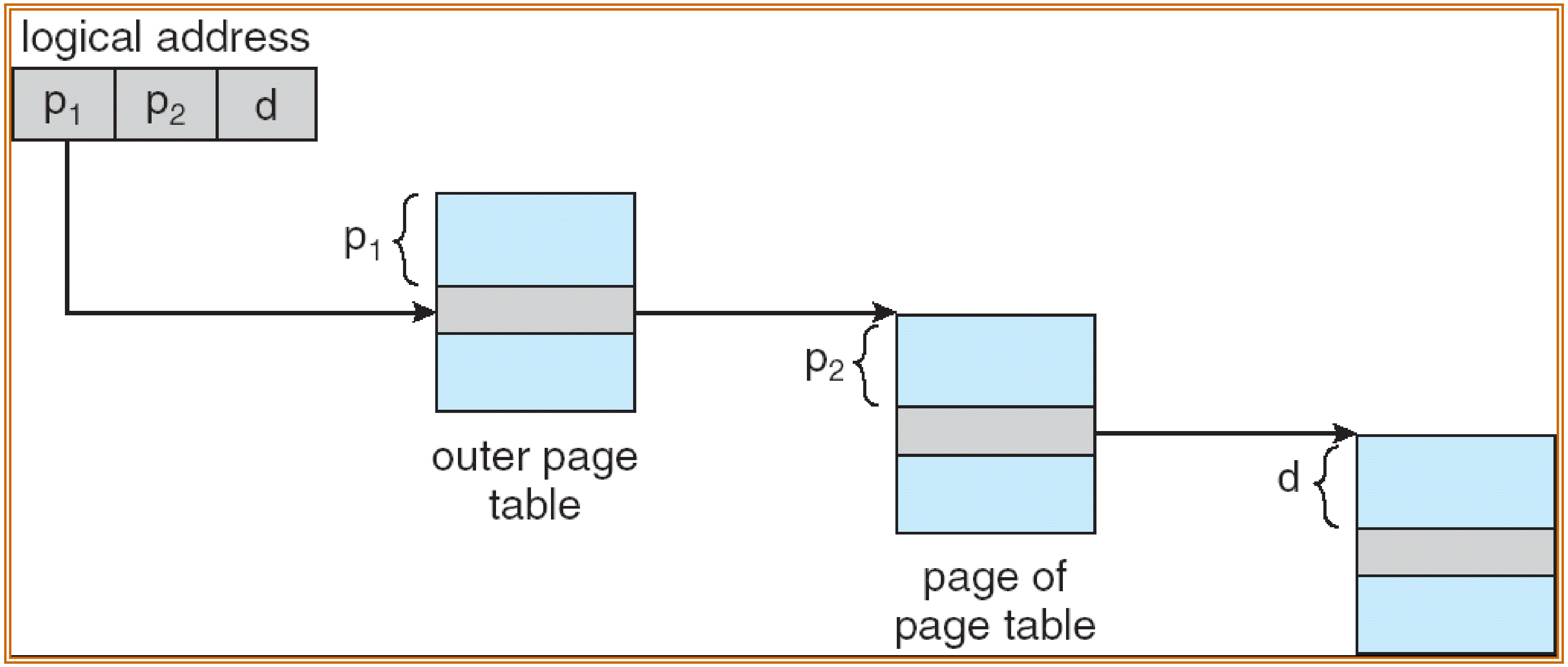
Two-Level Paging Example

- 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 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

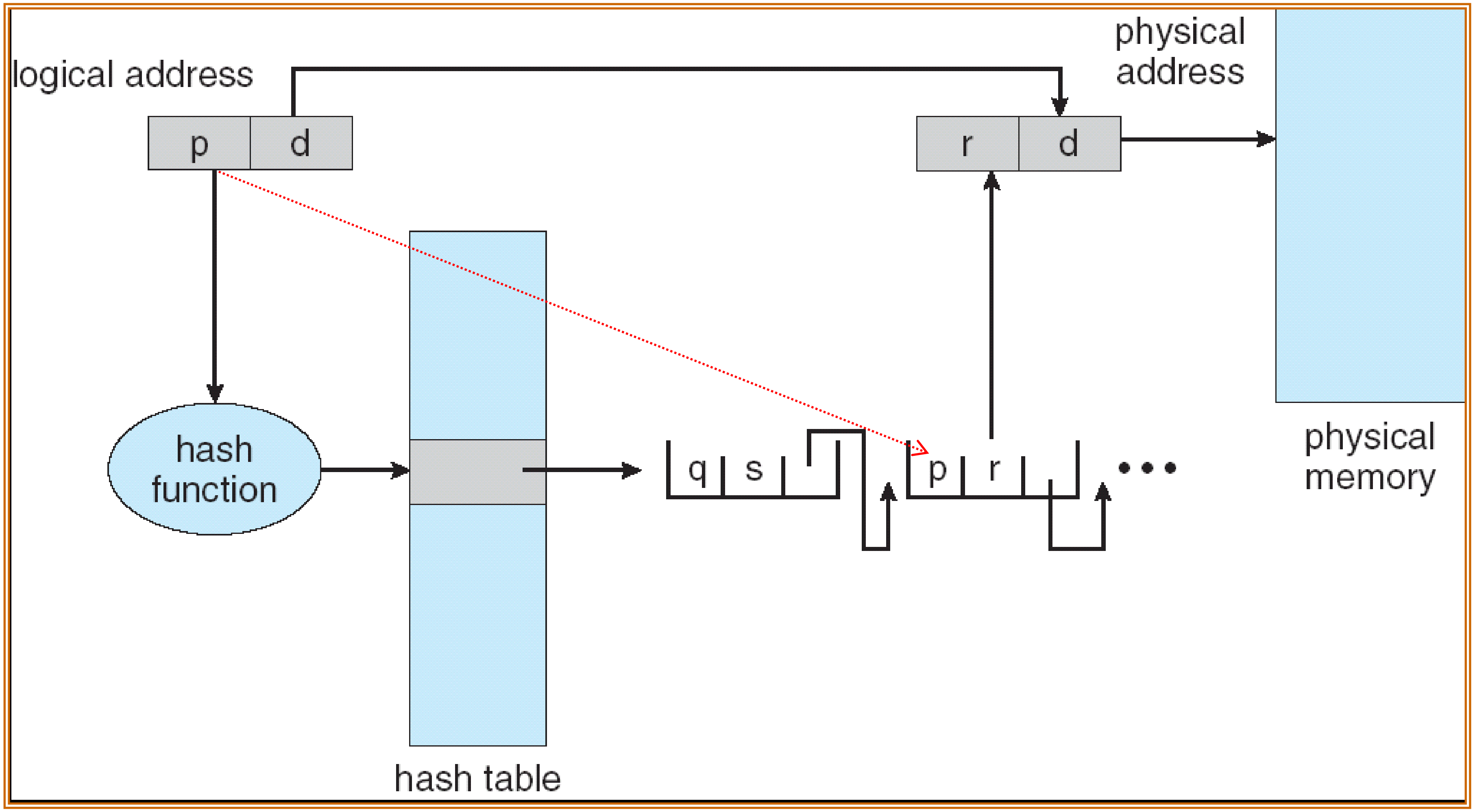
outer page	inner page	offset
p_1	p_2	d
42	10	12

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

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.

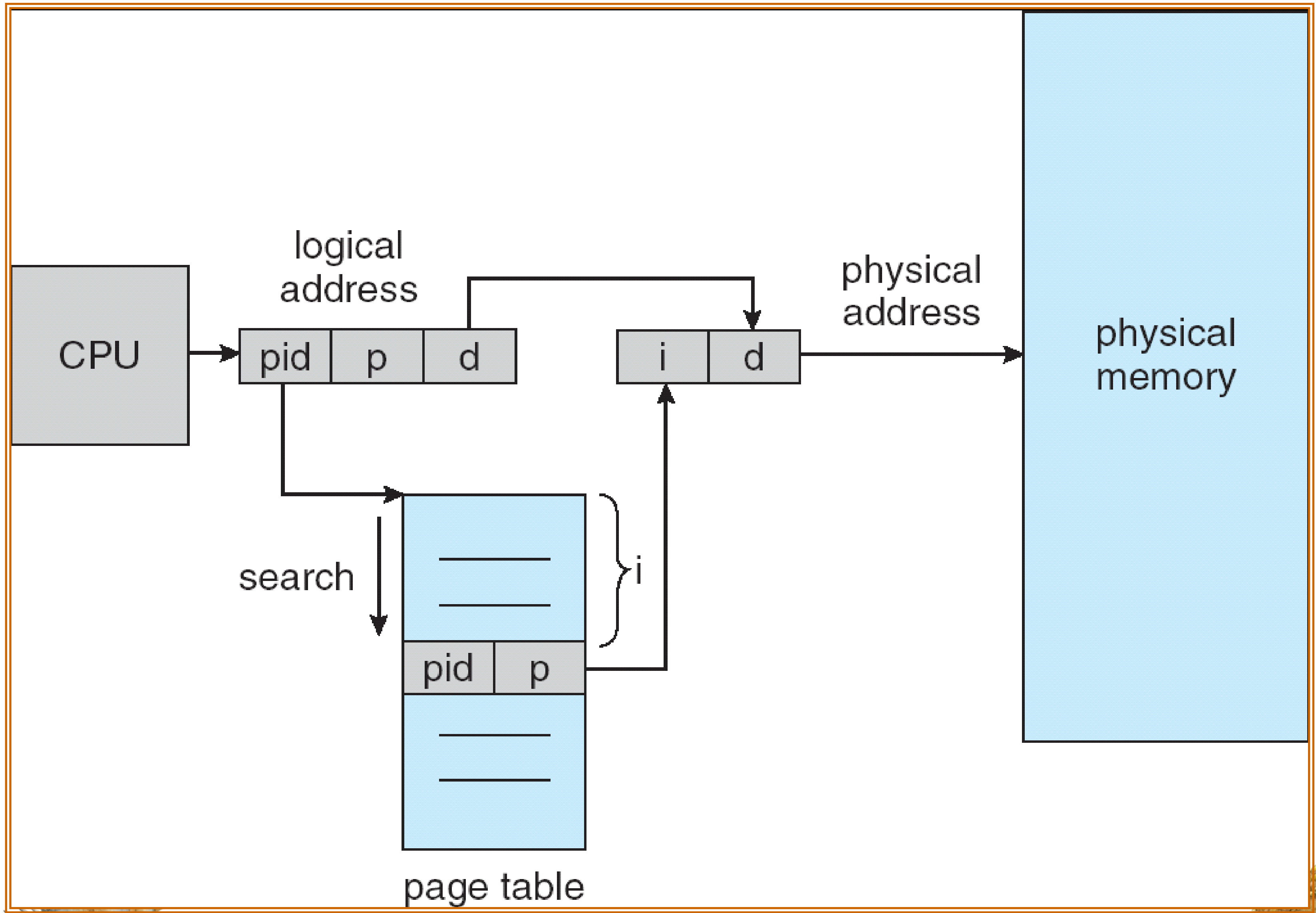
Hashed Page Table



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

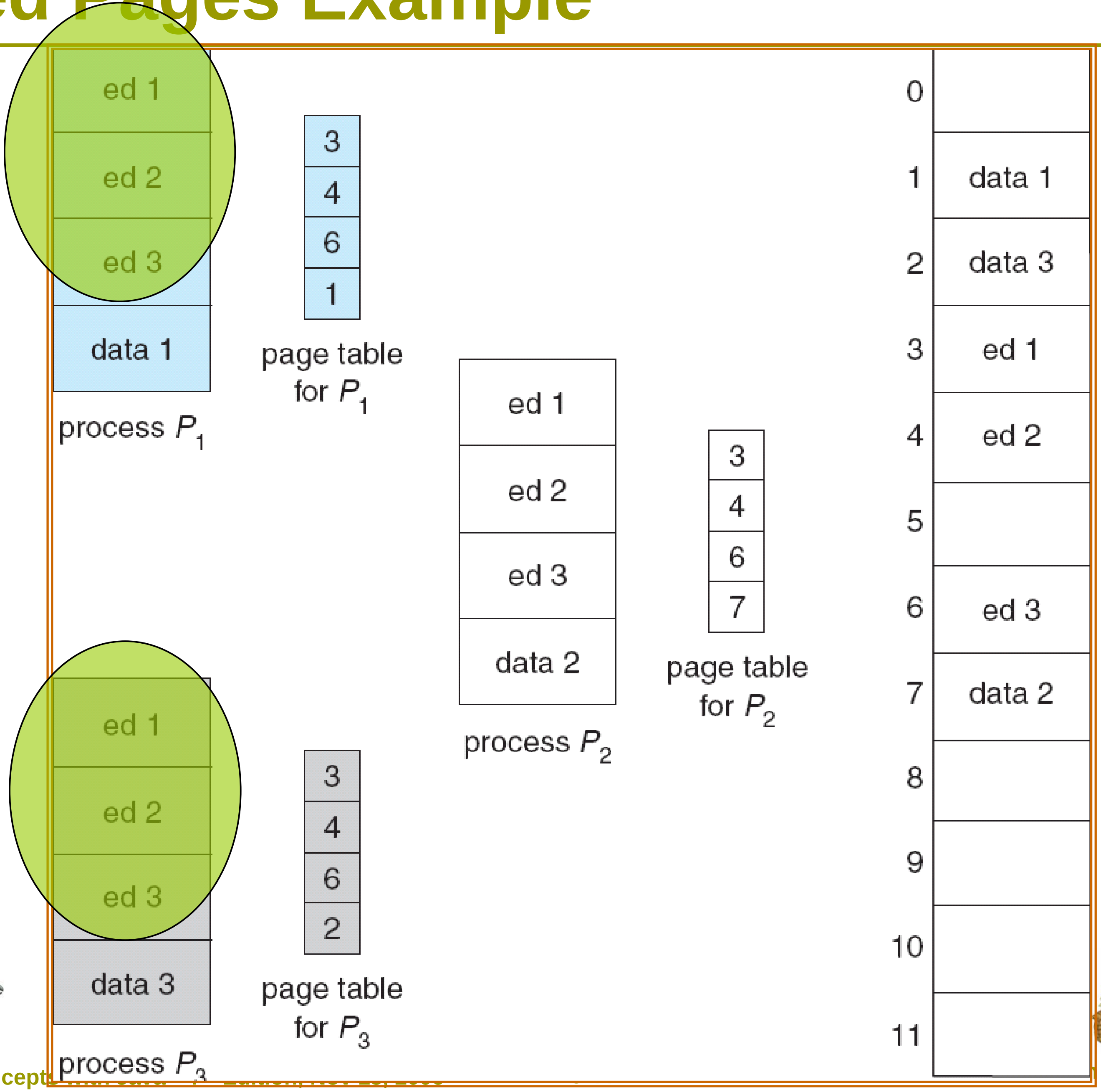
Inverted Page Table Architecture



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



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,

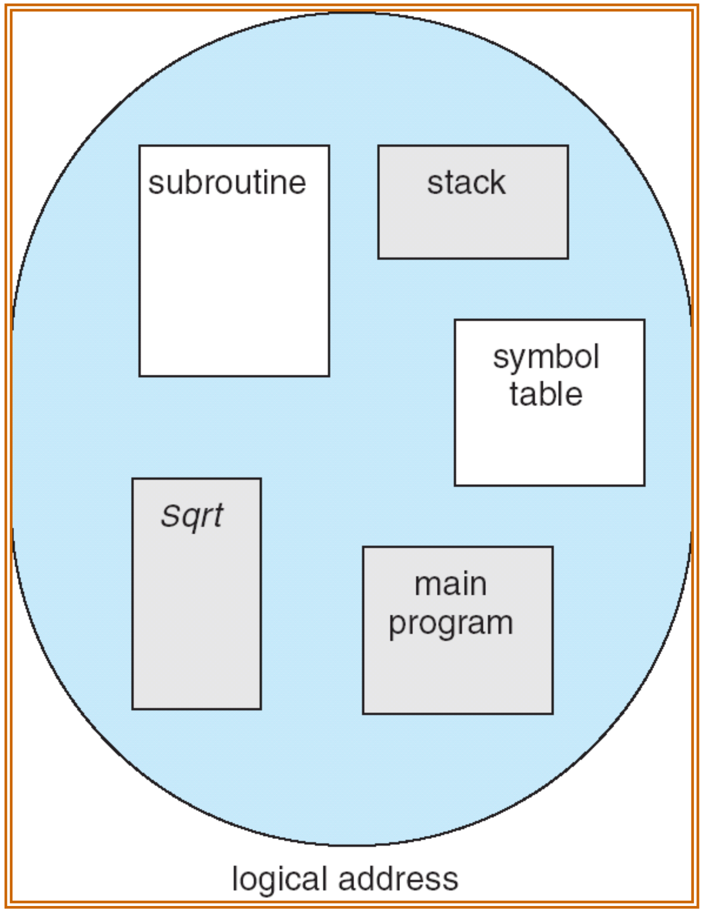
  local variables, global variables,

  common block,

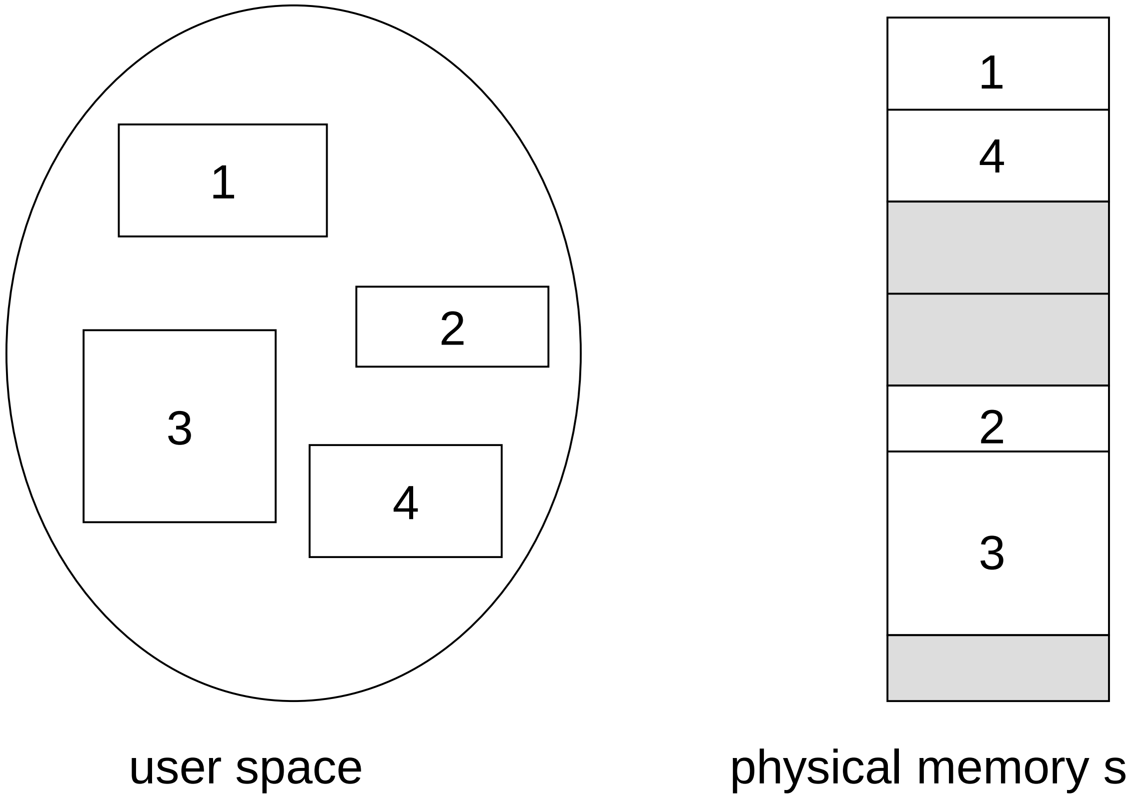
  stack,

  symbol table, arrays

User’s View of a Program



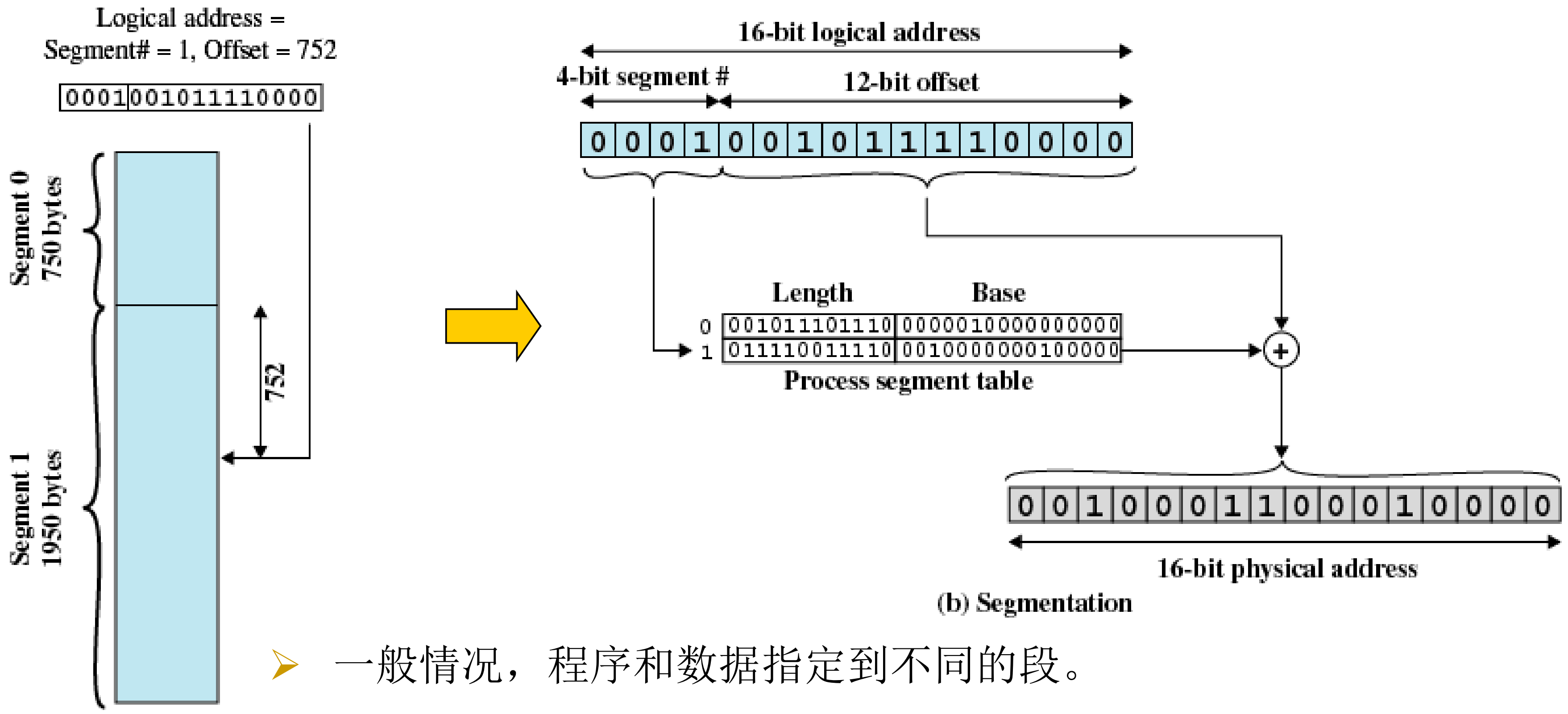
Logical View of Segmentation



分段技术

- 程序和相关的数据被划分成一组段（**segment**）。
- 分段技术的逻辑地址由两部分组成：**段号**和**偏移量**。
- 由于使用大小不等的段，分段类似于动态分区。与动态分区不同的是，在分段方案中，一个程序可以占据多个分区，并且这些分区不要求是连续的。
- 分段消除了内部碎片，但是和动态分区一样，会产生很多外部碎片。
- 由于进程被分为许多小块，**外部碎片较小**。

分段技术



(b) Segmentation

- 一般情况，程序和数据指定到不同的段。
- 上图中，段长度=4，偏移量=12。最大段长度2^{12}=4096
  - 提取段号（最左边4位）；
  - 查找物理地址；
  - 比较偏移量和段长度，是否地址有效；
  - 物理地址=该段起始地址+偏移量

Segmentation Architecture

- Logical address consists of a two tuple:

  <segment-number, offset>,

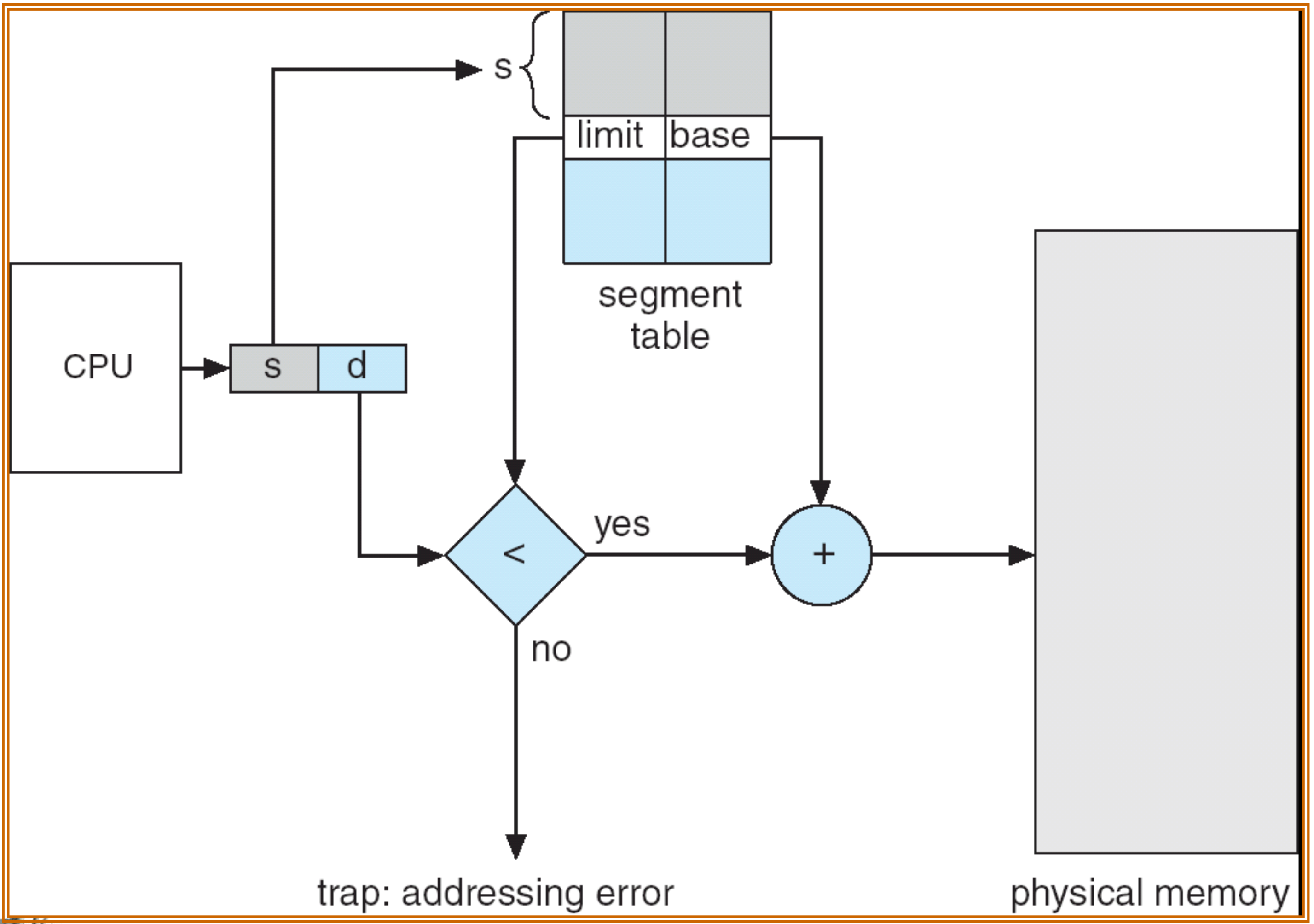
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location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

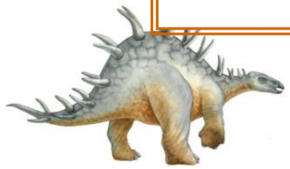
  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Architecture (Cont.)

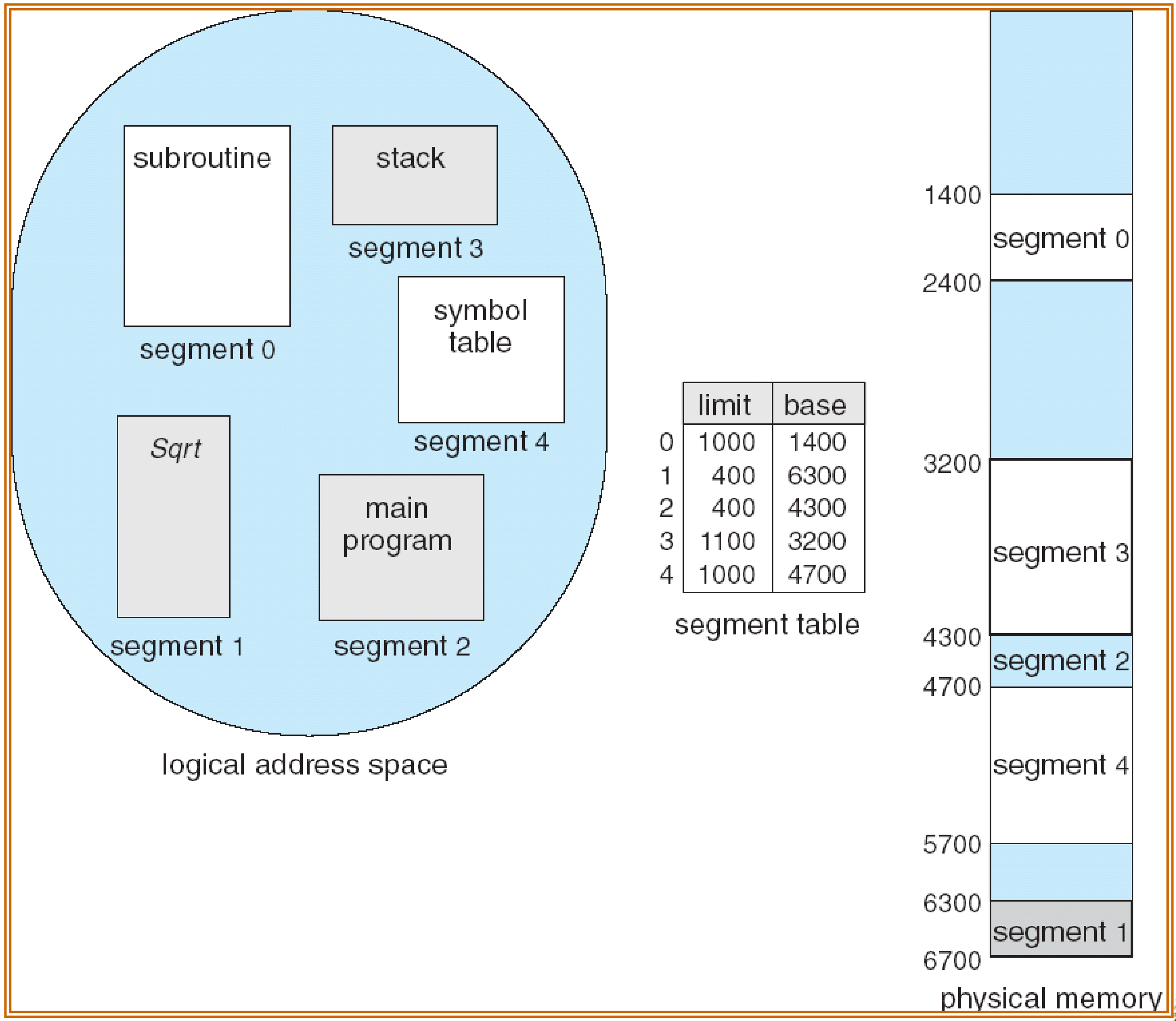
- Protection
  - With each entry in segment table associate:
    - validation bit = 0 $\Rightarrow$ illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem
- A segmentation example is shown in the following diagram

Segmentation Hardware





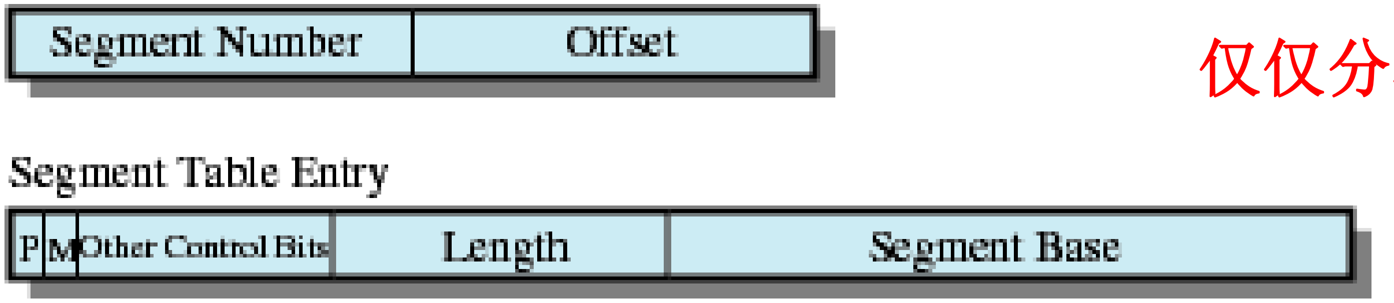
Example of Segmentation

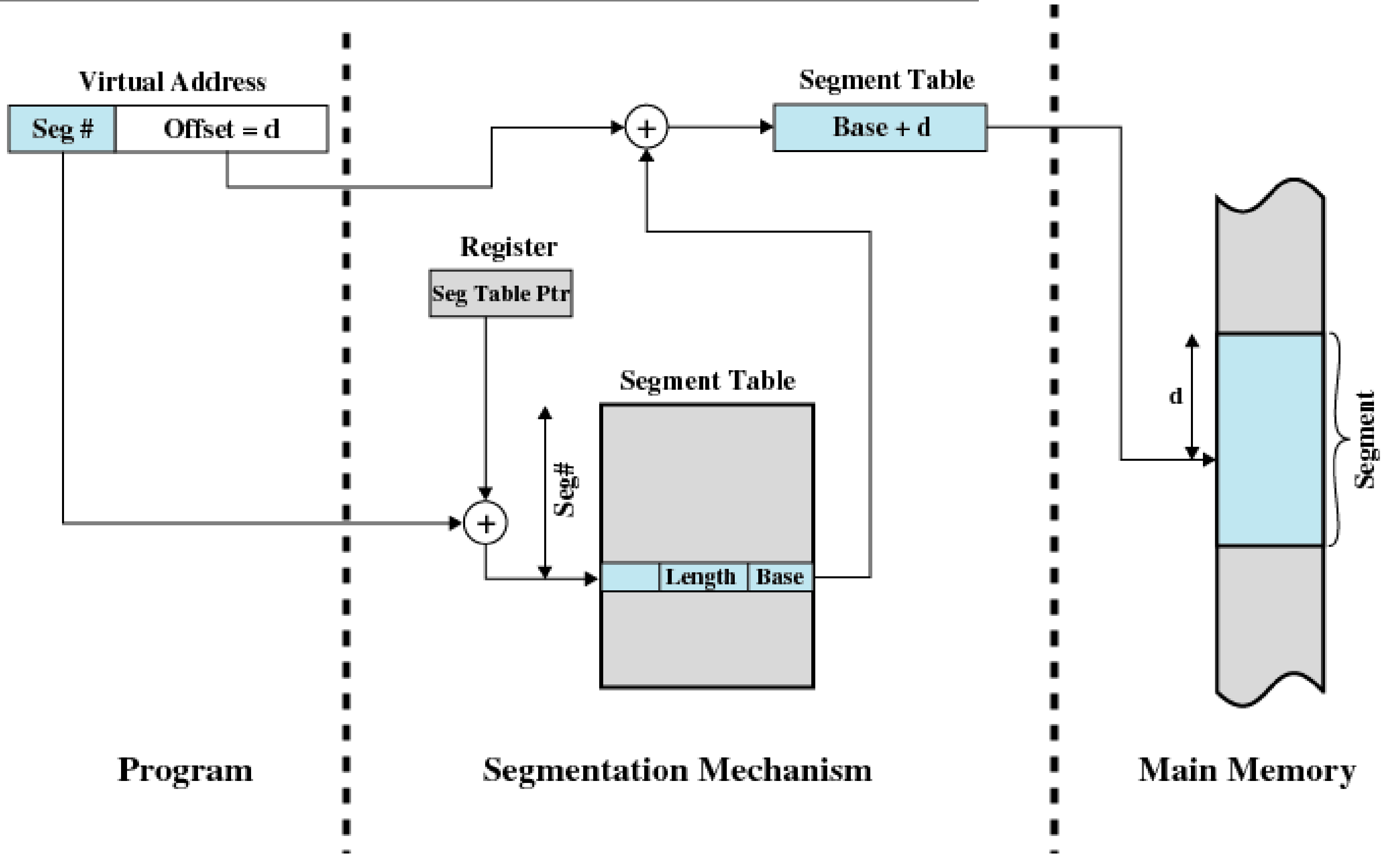


	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

分段式虚拟内存

仅仅分段

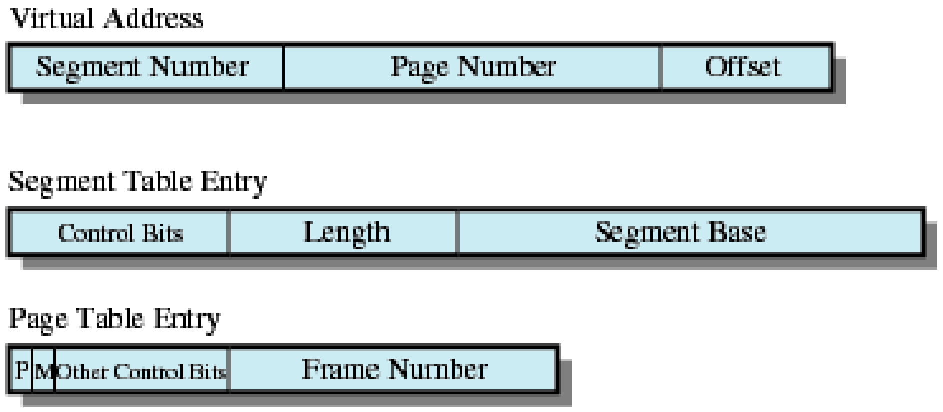




分段和分页结合

用户的地址空间被程序员划分为多个段，段由大小相同的若干页构成，页面有页内地址。

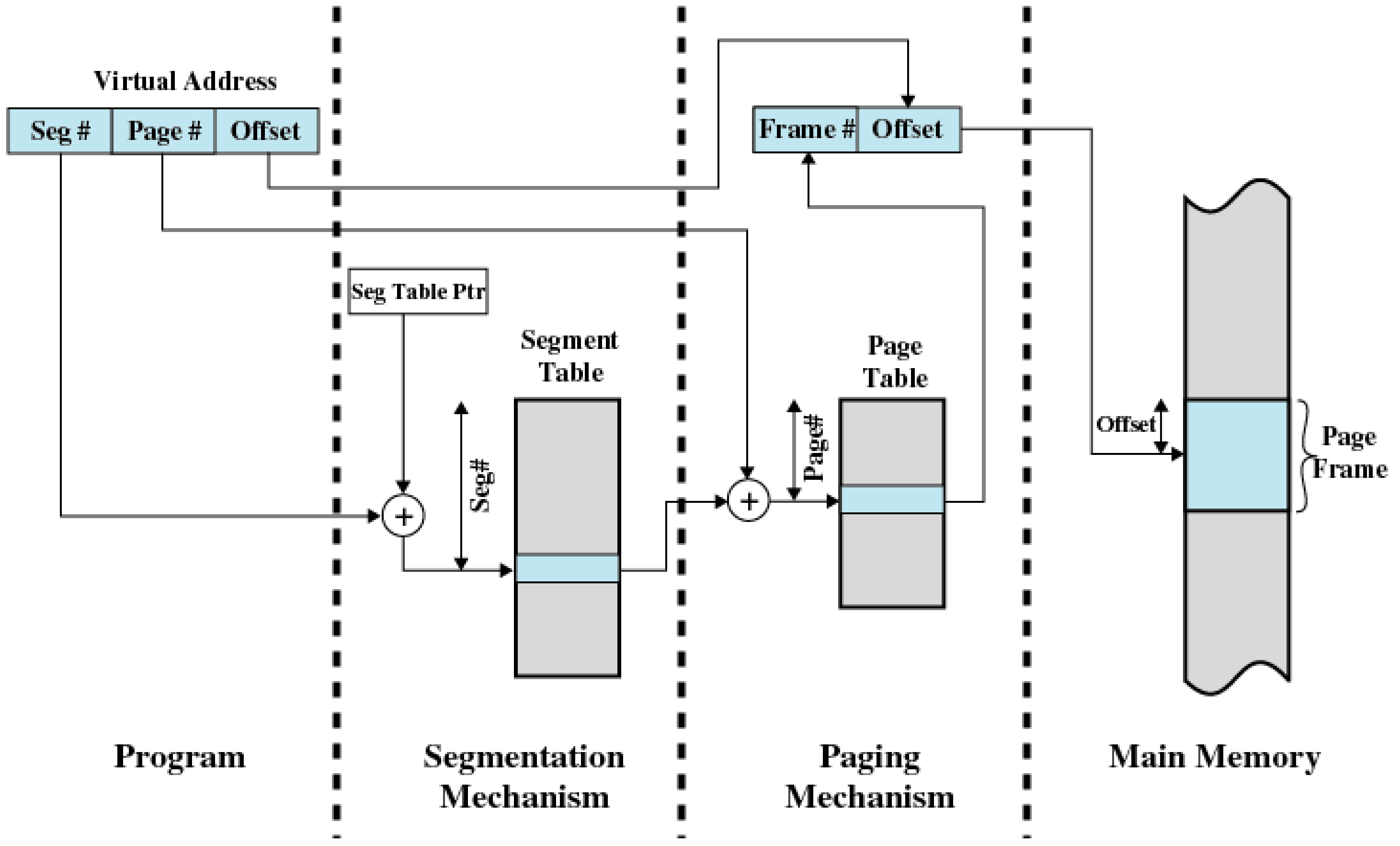
段页式管理系统的地址结构为（S, P, d），其中S为段号，P为段内页号，d为页内偏移量。



P= present bit
M = Modified bit

(c) Combined segmentation and paging

分段和分页结合


Virtual Address
Seg #
Page #
Offset
Frame #
Offset
Seg Table Ptr
Segment Table
Seg#
+
Page Table
Page#
+
Offset
Page Frame
Program
Segmentation Mechanism
Paging Mechanism
Main Memory

End of Chapter 8