Main Memory

Fan Wu
Department of Computer Science and Engineering
Shanghai Jiao Tong University

Spring 2022

Background

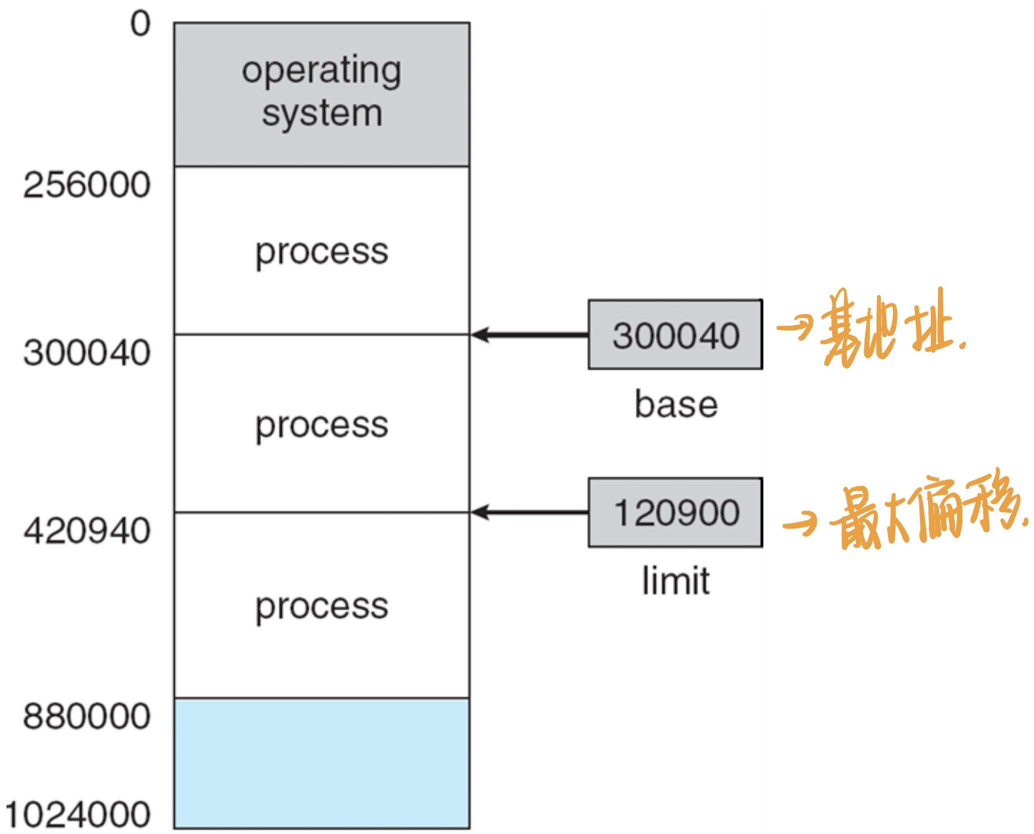
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Register access in one CPU clock cycle
- Main memory may take several cycles

地址+读请求 地址+data+写请求

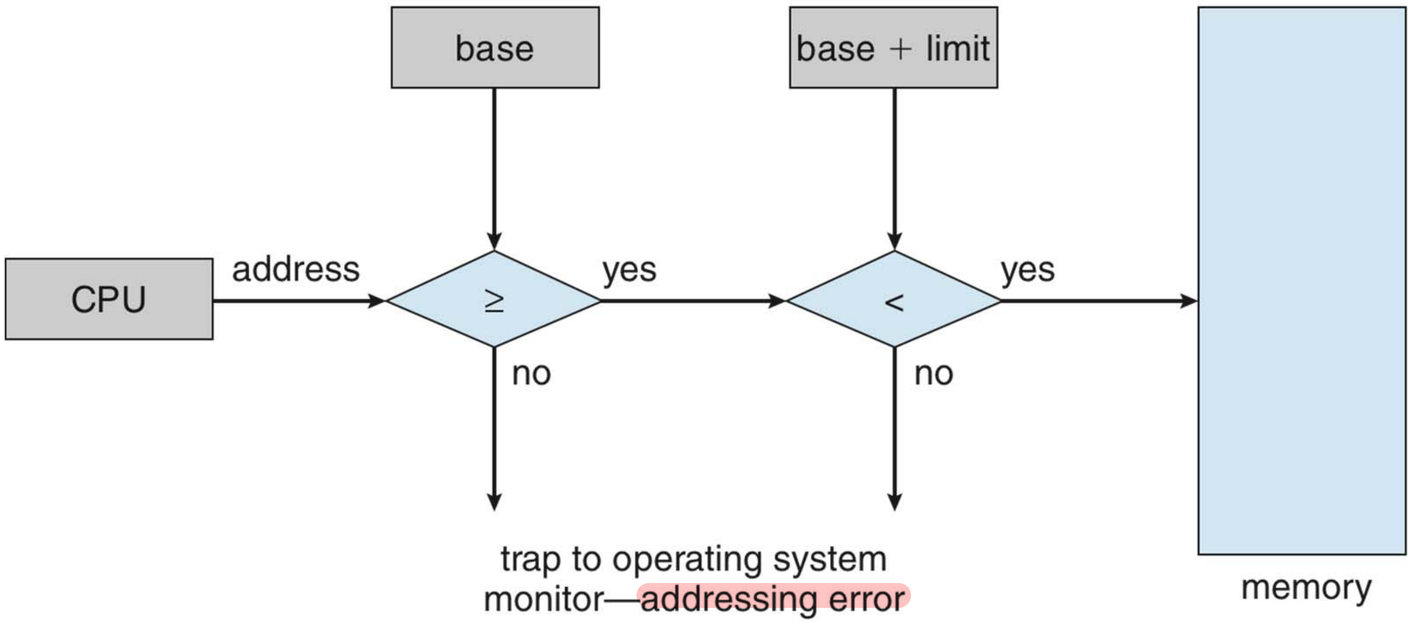
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Protection of memory required to ensure correct operation

Base and Limit Registers

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



Hardware Address Protection



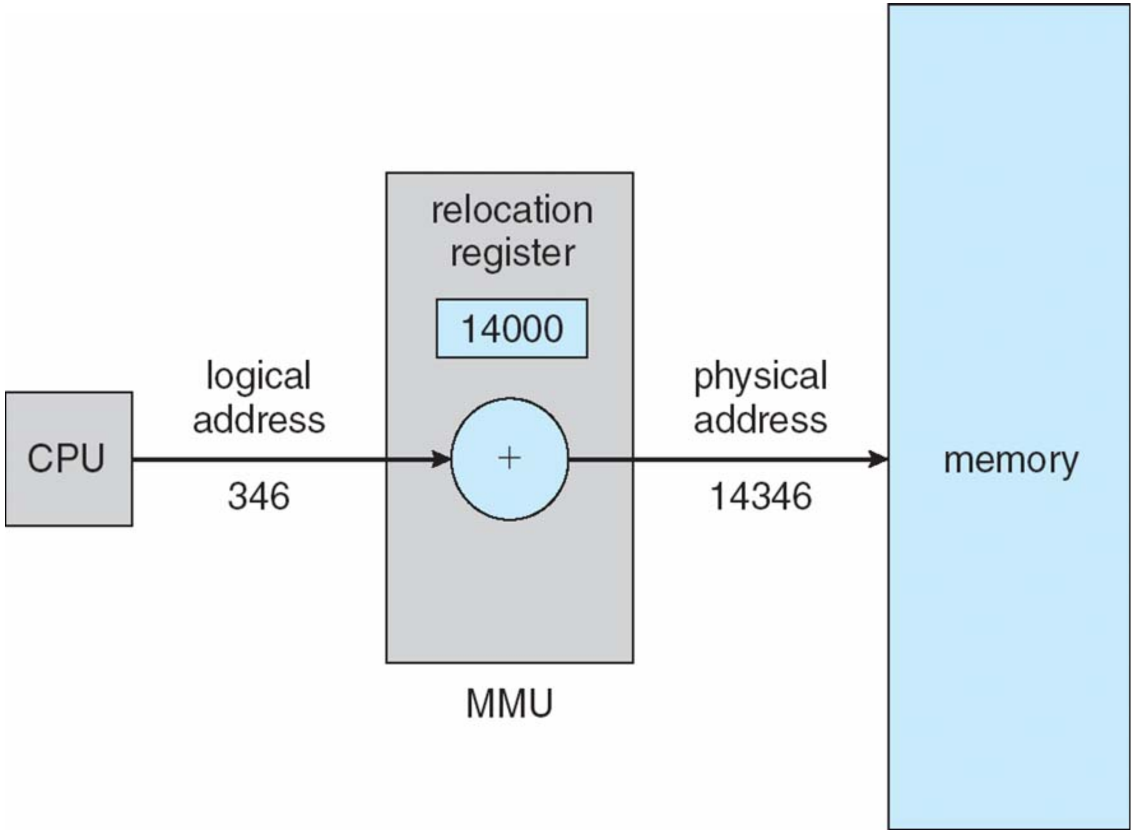
Logical vs. Physical Address Space

- 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** CPU生成的.
  - **Physical address** – address seen by the memory unit
    内存单元所看到的.
- **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 logical to physical address
- Many methods possible, covered in the rest of this chapter
- To start, consider a 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

Dynamic Relocation using a Relocation Register

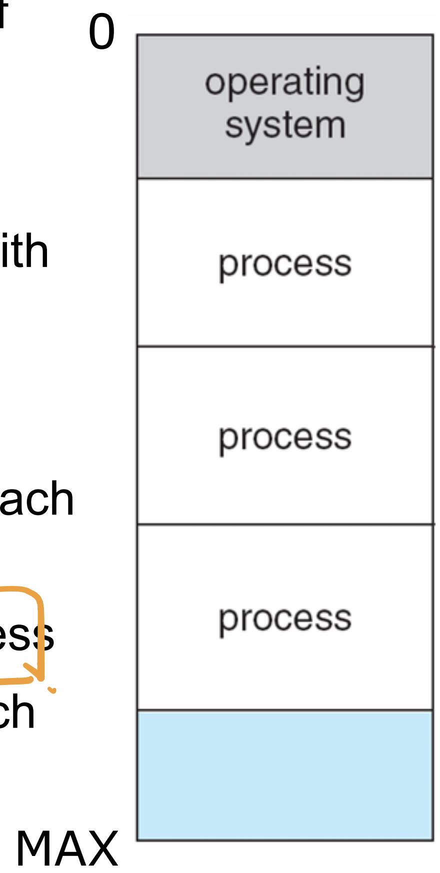


Memory Management

- Contiguous memory allocation 连续
- Non-contiguous memory allocation 非连续
  - Paging
    页

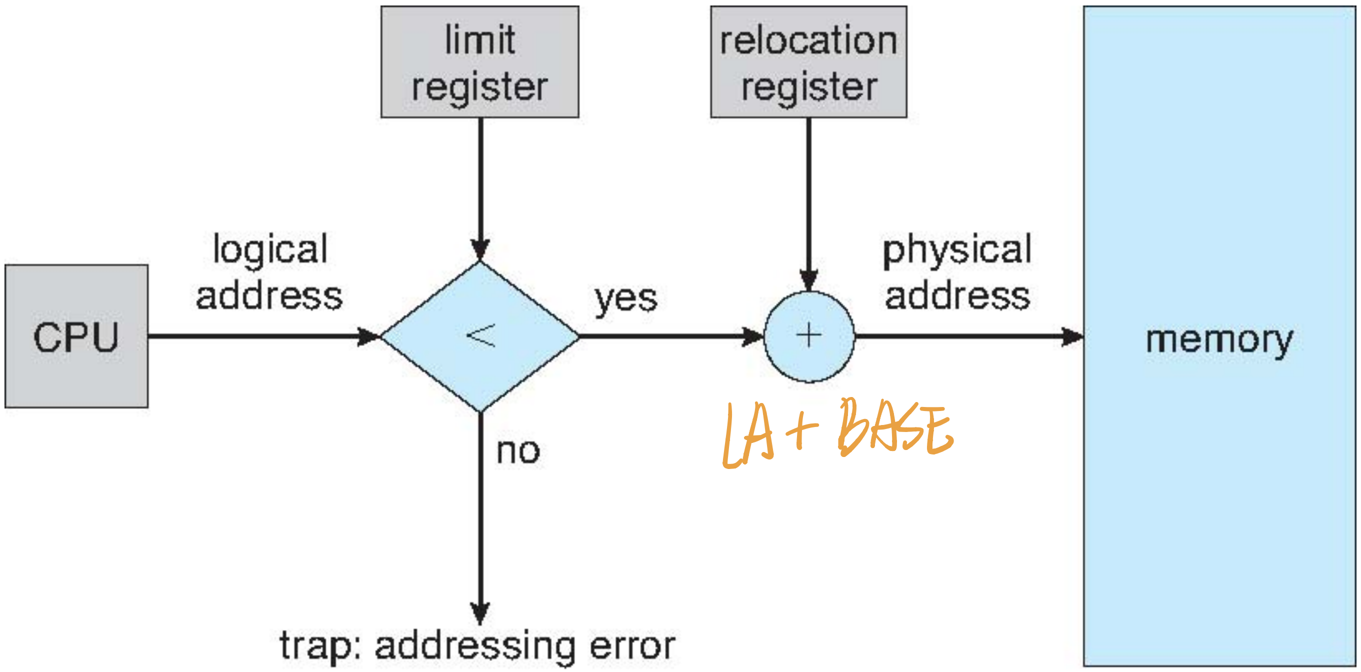
Contiguous Allocation

- Each process is contained in a single contiguous section of memory
- Main memory usually contains two partitions:
  - Resident operating system, held in low/high memory with interrupt vector
  - User processes then held in high/low 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



Hardware Support for Relocation and Limit Registers

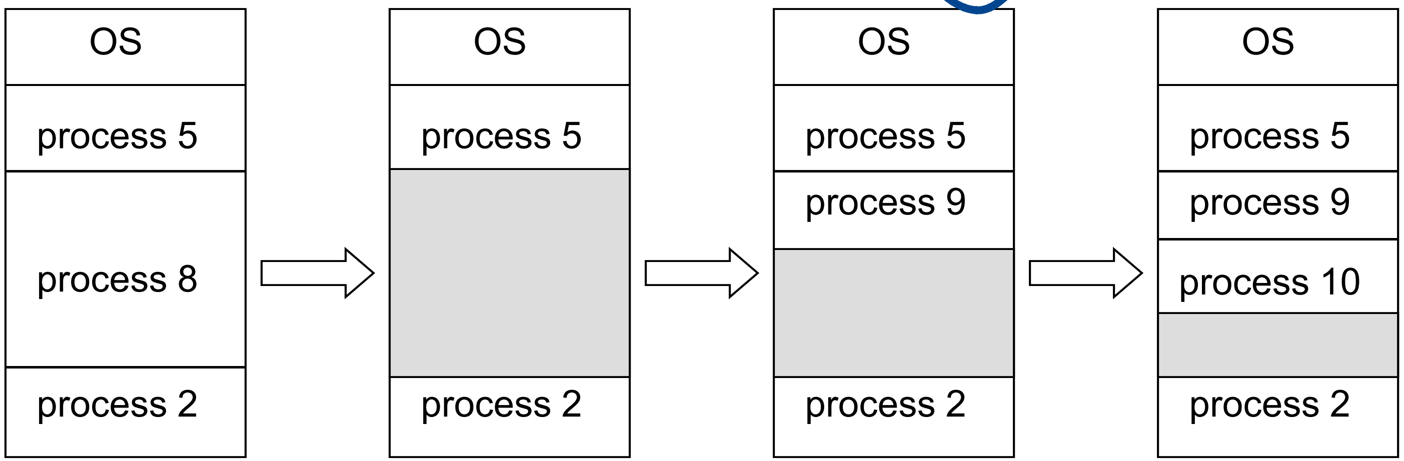
LA范围 0 ~ limit register.



cpu $\xrightarrow{LA}$ mmu $\xrightarrow{PA}$ memory.

Contiguous Allocation (Cont.)

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - 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

Fragmentation

外部碎片 useful

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous

内部碎片

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given *N* blocks allocated, another 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> 50-percent rule

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction** 压缩
  - 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 在程序运行时才能进行压缩.

- Another solution to permit the logical address space of the processes to be **noncontiguous**
  - **paging**
  - **segmentation** 分页与分段

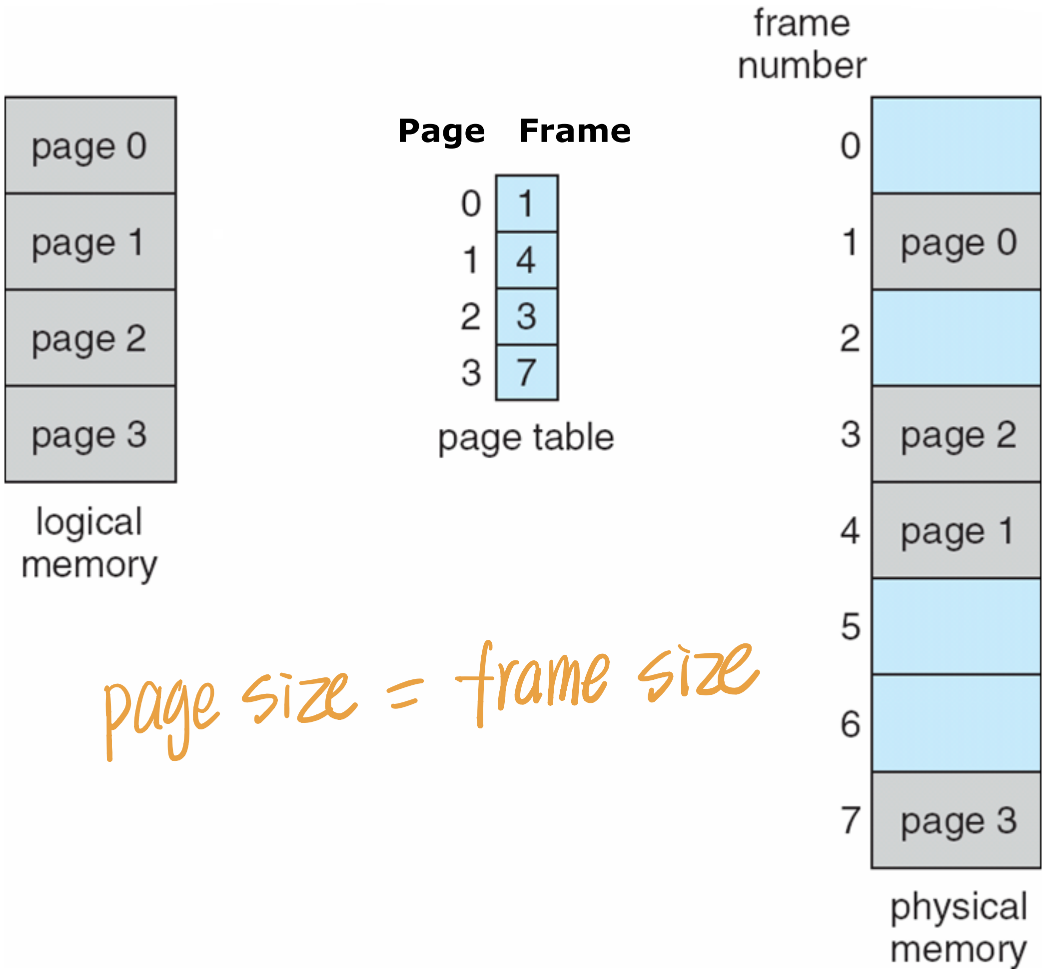
Paging

- Physical 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 16 Mbytes
- Divide logical memory into blocks of same size called **pages**

have same size

- Keep track of all free frames
- To run a program of size *N* pages, need to find up to *N* free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Still have Internal fragmentation

Paging Model of Logical and Physical Memory

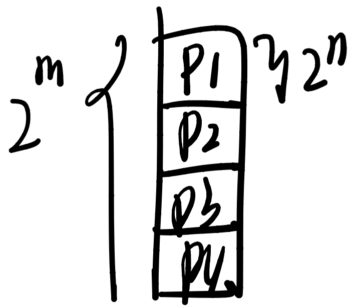


Address Translation Scheme

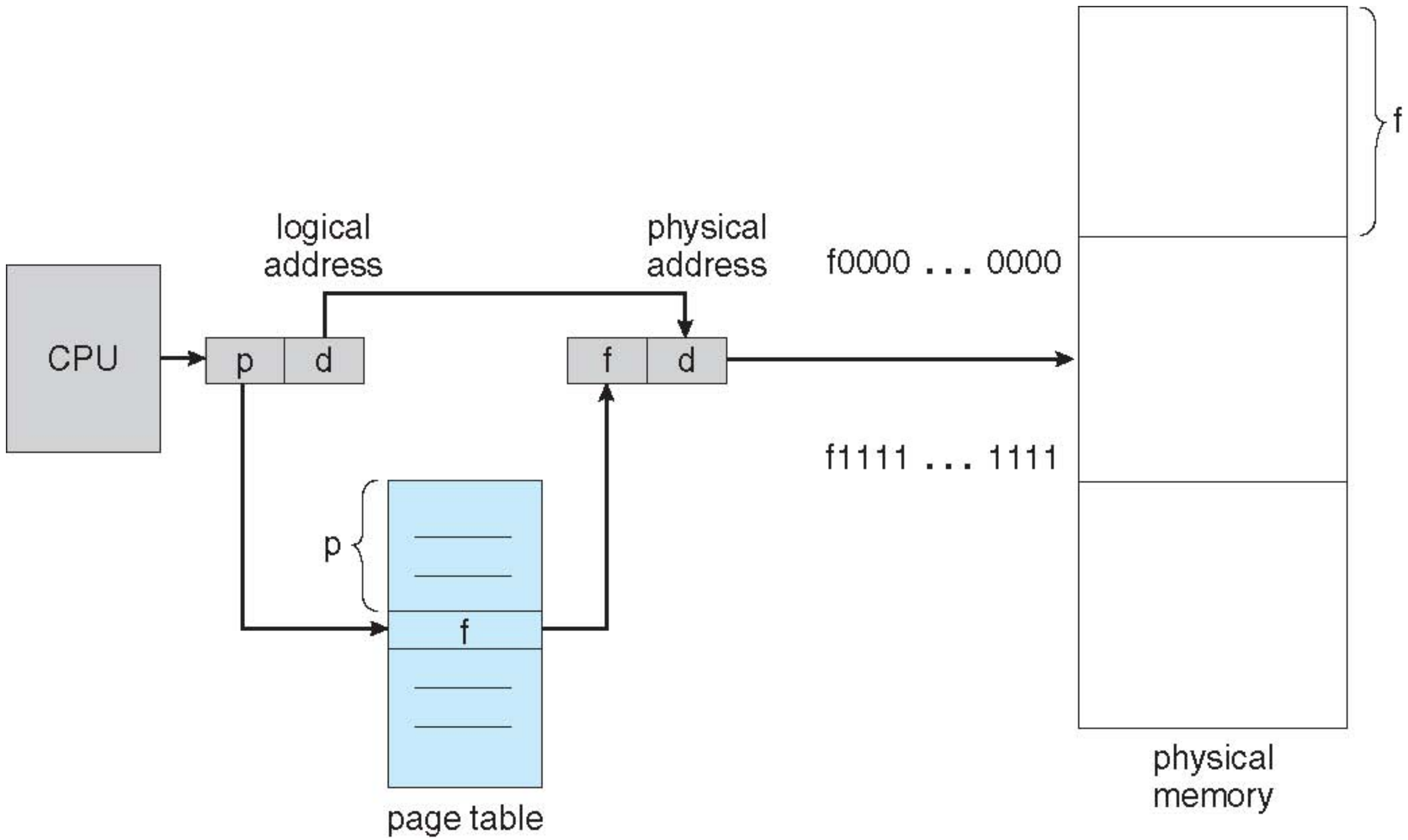
- Address generated by CPU is divided into:
  - **Page number (*p*)** – used as an **index** into a **page table**, which contains base address of each page in physical memory
  - **Page offset (*d*)** – combined with base address to define the physical memory address that is sent to the memory unit

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

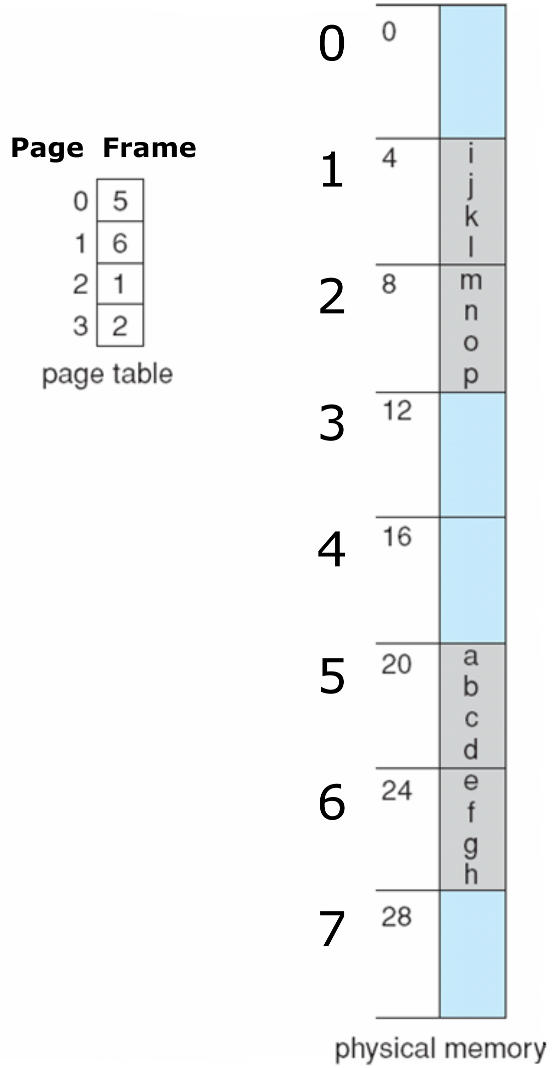
  - For given logical address space 2^m and page size 2^n



Paging Hardware



Paging Example



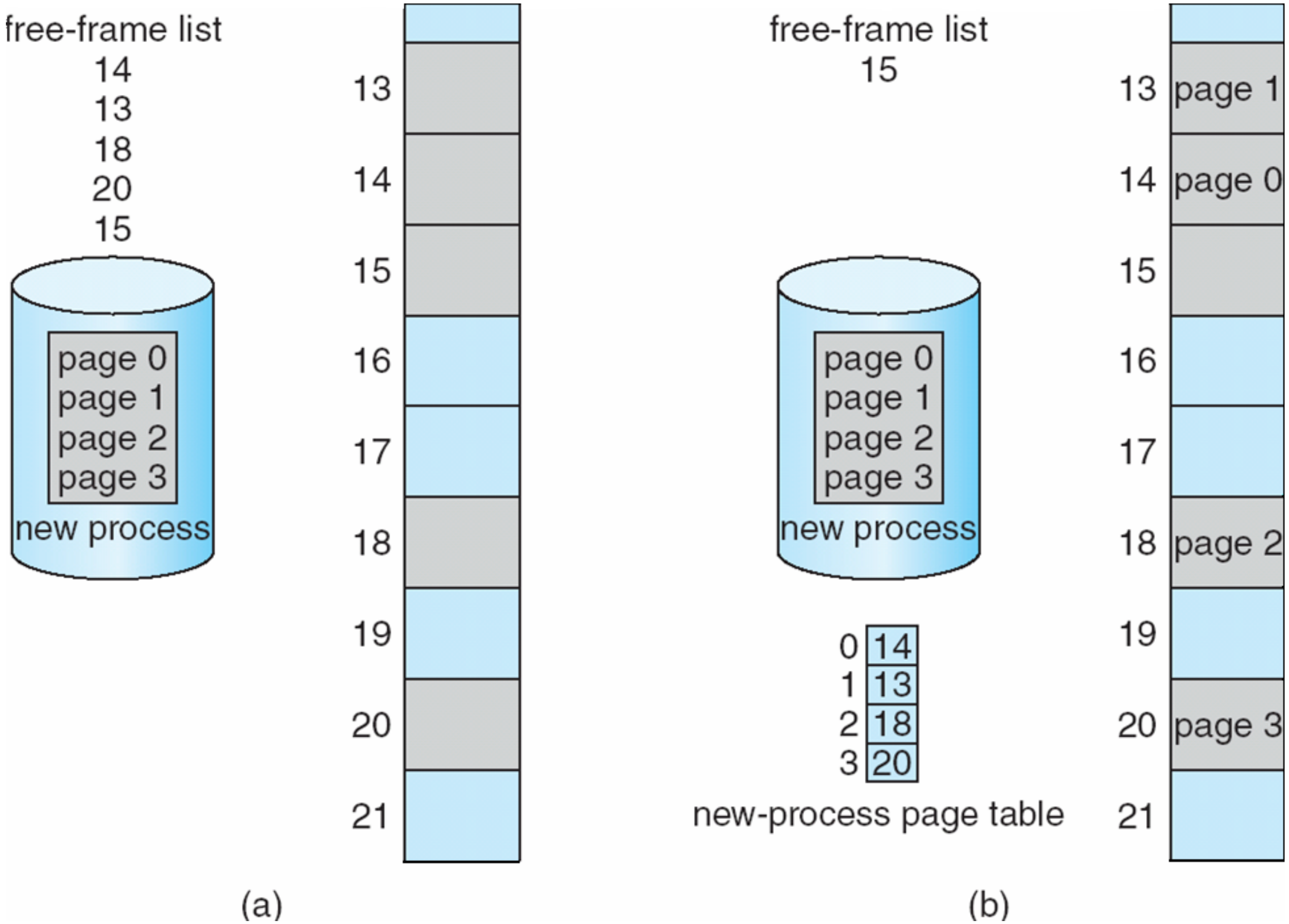
What character does the following logical addresses map to?
(A) 10 = 10 10 j
(B) 5 = 01 01 a
(C) 14 = 11 10 n

n=2 and *m*=4 32-byte memory and 4-byte pages

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
- Internal fragmentation
  - Worst case fragmentation = frame size – 1 byte
  - On average fragmentation = 1 / 2 frame size
- Calculate the page numbers and offsets for the following address, when page size is 1KB:
  - 2375 = 1024 * 2 + 327
  - 19366 = 1024 * 18 + 934

Free Frames



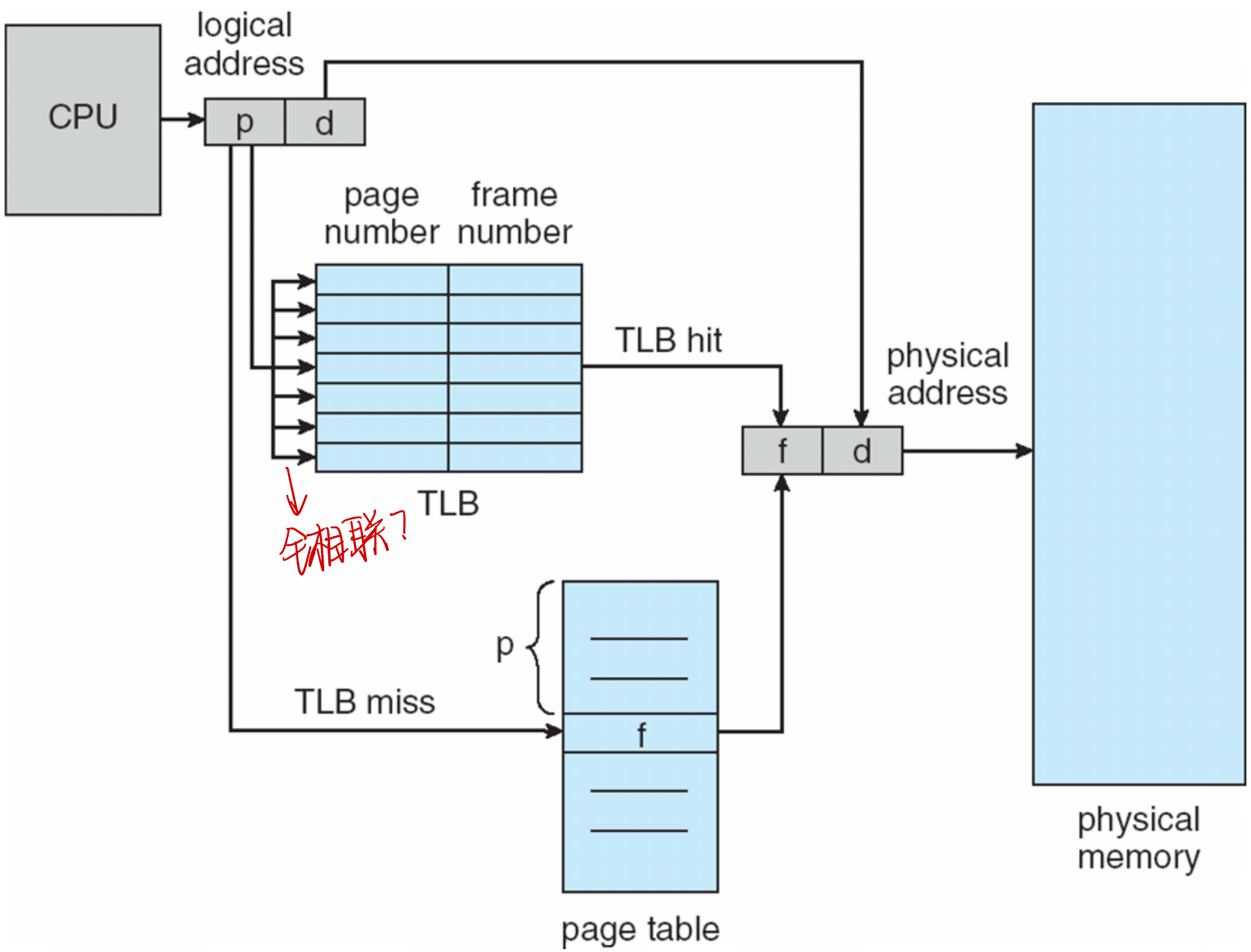
Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory 保存在内存里
  - **Page-table base register (PTBR)** points to the page table
  - **Page-table length register (PTLR)** 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)**
  - TLBs typically small (64 to 1,024 entries)
  - On a TLB miss, value is loaded into the TLB for faster access next time
    - Replacement policies must be considered
    - Some entries can be **wired down** for permanent fast access

Paging Hardware With TLB



上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Effective Access Time

- Associative Lookup = ε time unit
- Hit ratio = α
  - Hit ratio – percentage of times that a page number is found in the associative registers; ratio related to number of associative registers
- Consider α = 80%
- **Effective Access Time (EAT)**

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

①内存访问数据.
②TLB miss时，去内存查看page table.

  - When α = 80%, ε = 20ns for TLB search, 100ns for 1 memory access time unit
  - EAT = 120 x 0.80 + 220 x 0.20 = 140ns

Structure of the Page Table

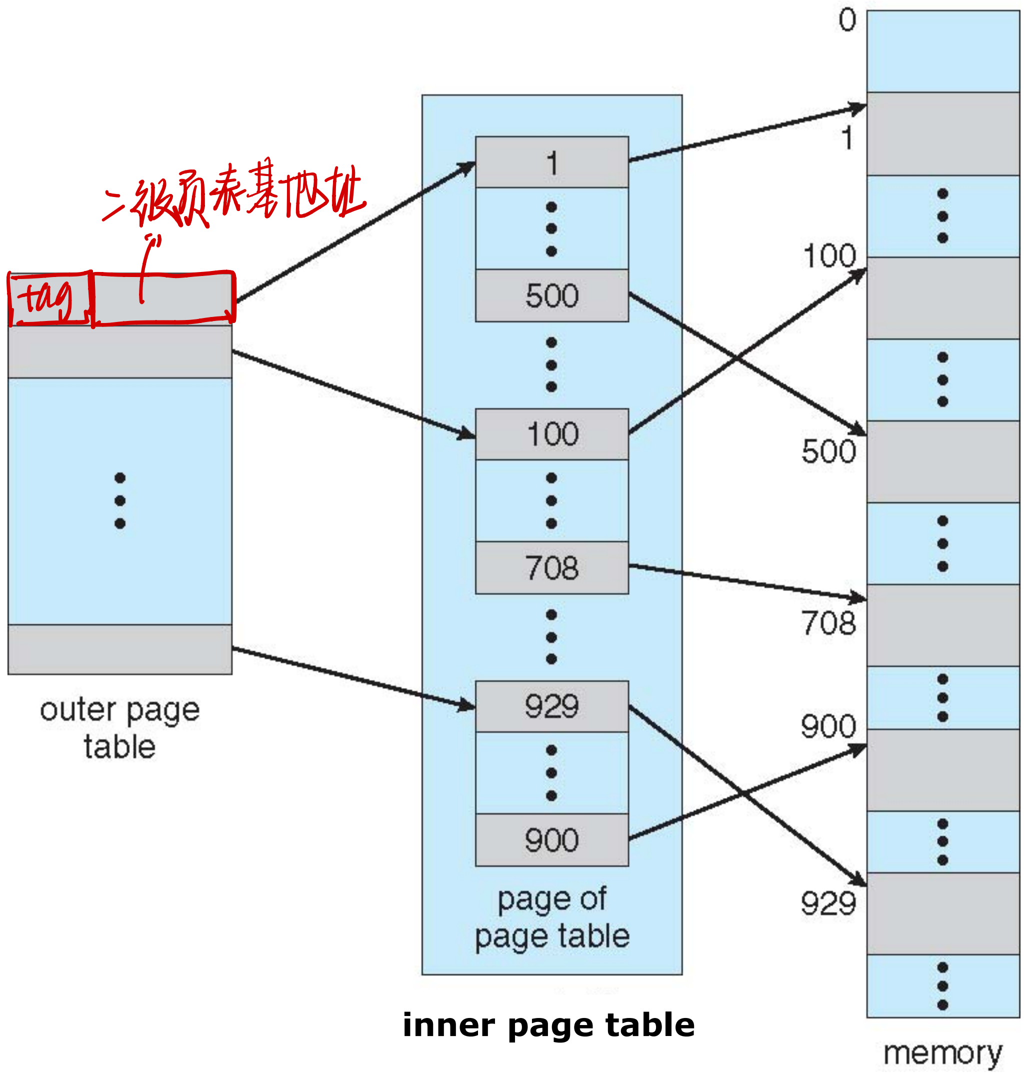
- Memory structures for paging can get huge using straight-forward methods
  - Consider a 32-bit logical address space as on modern computers 2^{32}
  - Page size of 4 KB (2^{12})
  - Page table would have 1 million entries (2^{32} / 2^{12})
  - If each entry is 4 bytes -> 4 MB of physical address space / memory for page table alone $2^4 \times 2^{20} = 4MB$
    - That amount of memory used to cost a lot
    - Don't want to allocate that contiguously in main memory

- 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
- We then page the page table

Two-Level Page-Table Scheme



inner page table

Two-Level Paging Example

$2^{32}/2^{12} = \boxed{2^{20}}$

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

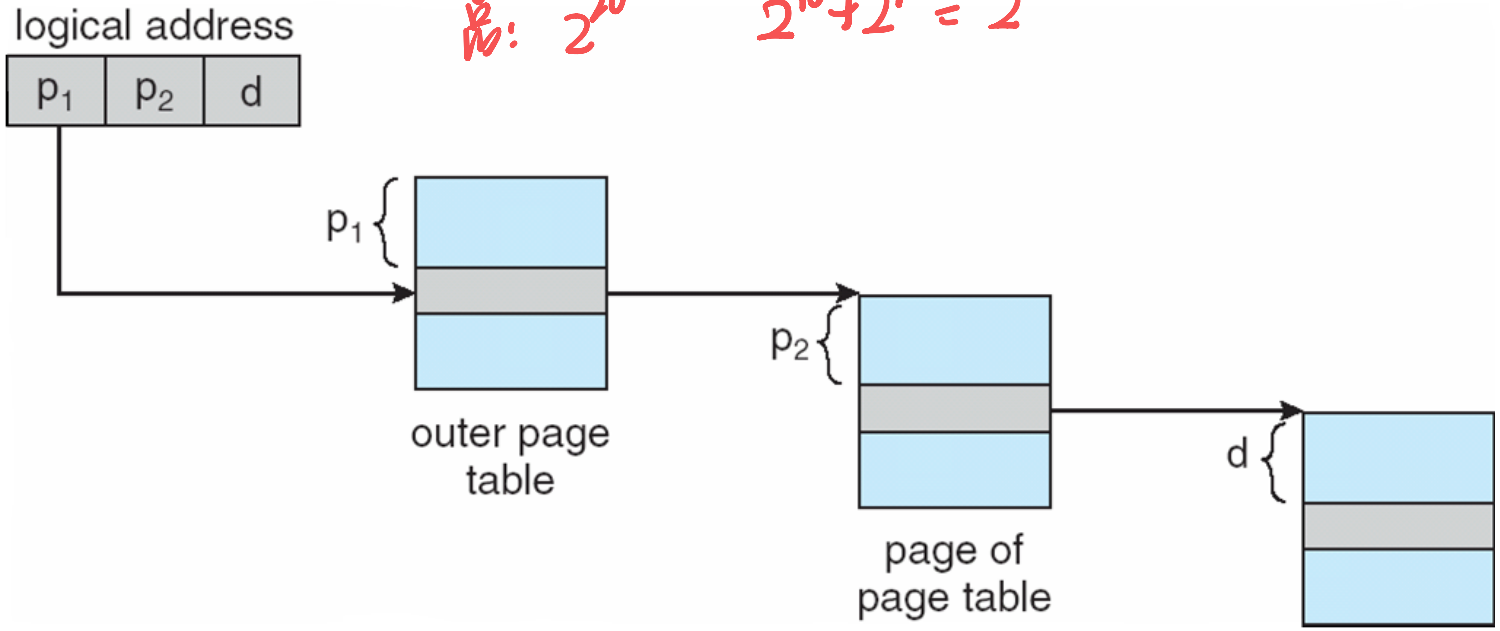
page number		page offset
p_1	p_2	d
10	10	12

- where p_1 is an index into the outer page table, and p_2 is the offset within the page of the inner page table
- Known as **forward-mapped page table**

Address-Translation Scheme

$2^{20} \rightarrow 2^{10} \times 2^{10}.$

总: 2^{20} $\quad 2^{10} + 2^{10} = 2^{11}$



Three-level Paging Scheme

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

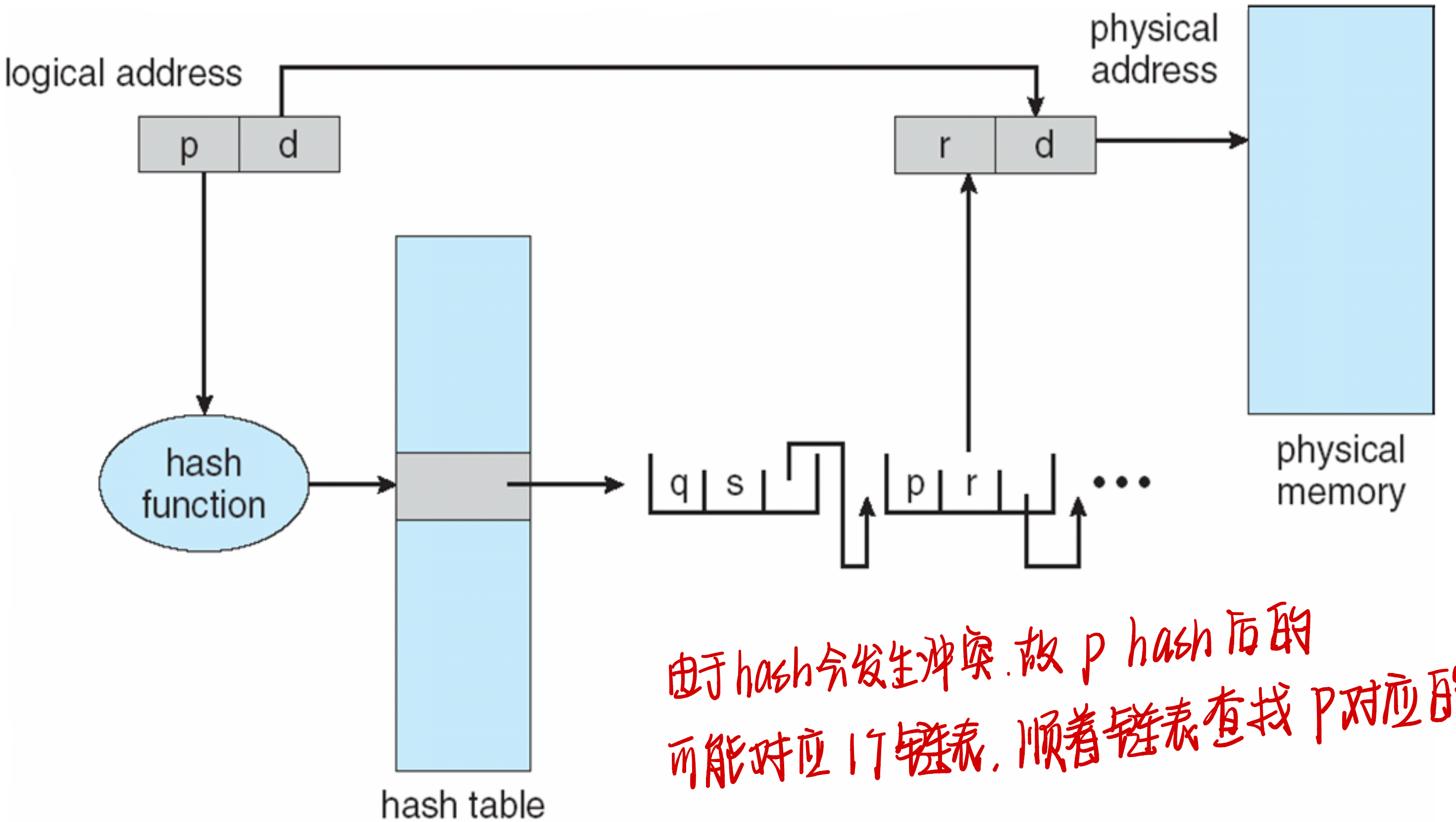
- The virtual page number is hashed into a page table
  - This page table contains a chain of elements hashed to the same location

= VPN + page frame + next pointer.

- Each element contains (1) the virtual page number (2) the address of the mapped page frame (3) a pointer to the next element

- 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



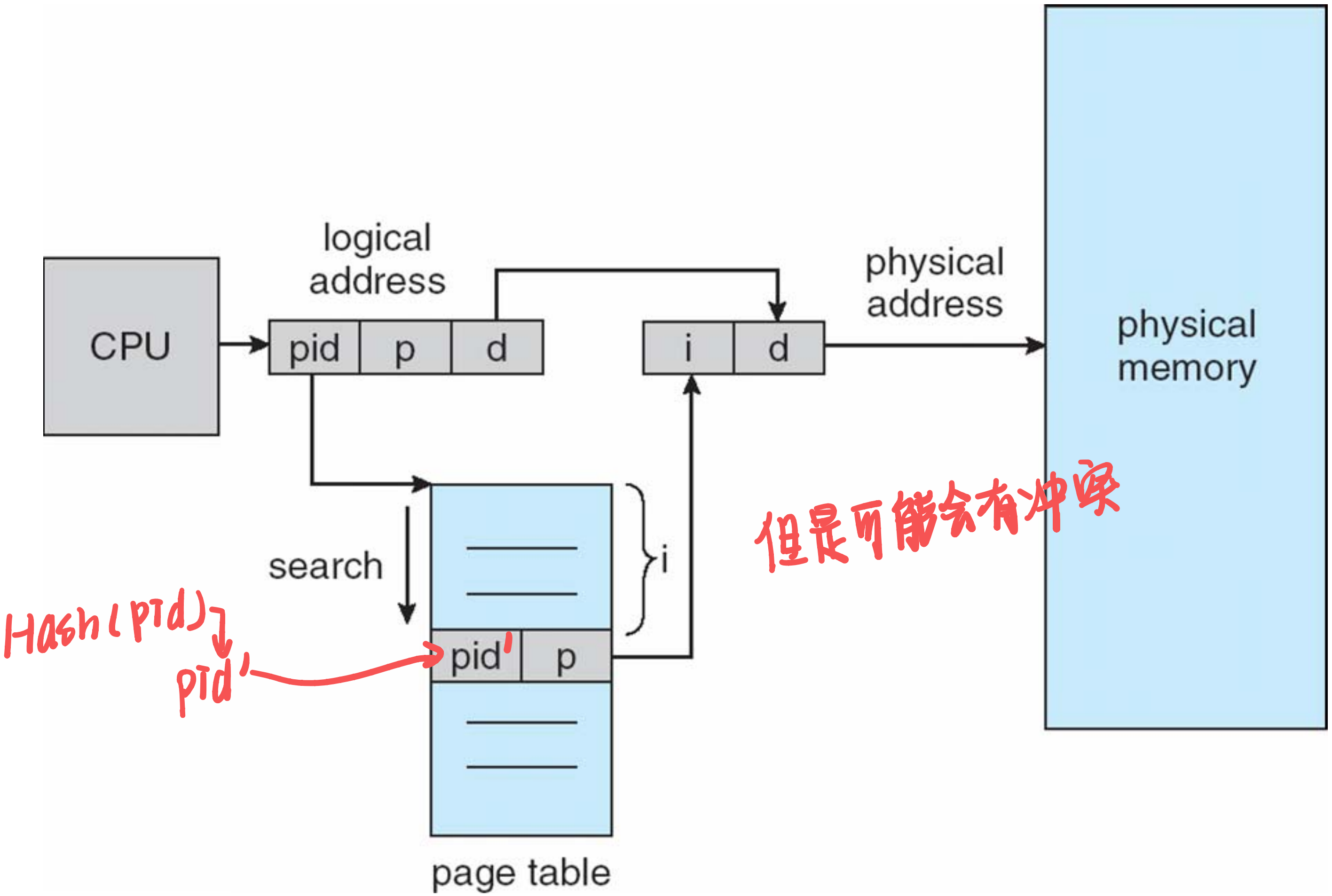
由于hash会发生冲突，故 p hash后的可能对应1个链表，顺着链表查找P对应的 PA.

Inverted Page Table

反向页表

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real frame 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
  - TLB can accelerate access

Inverted Page Table Architecture

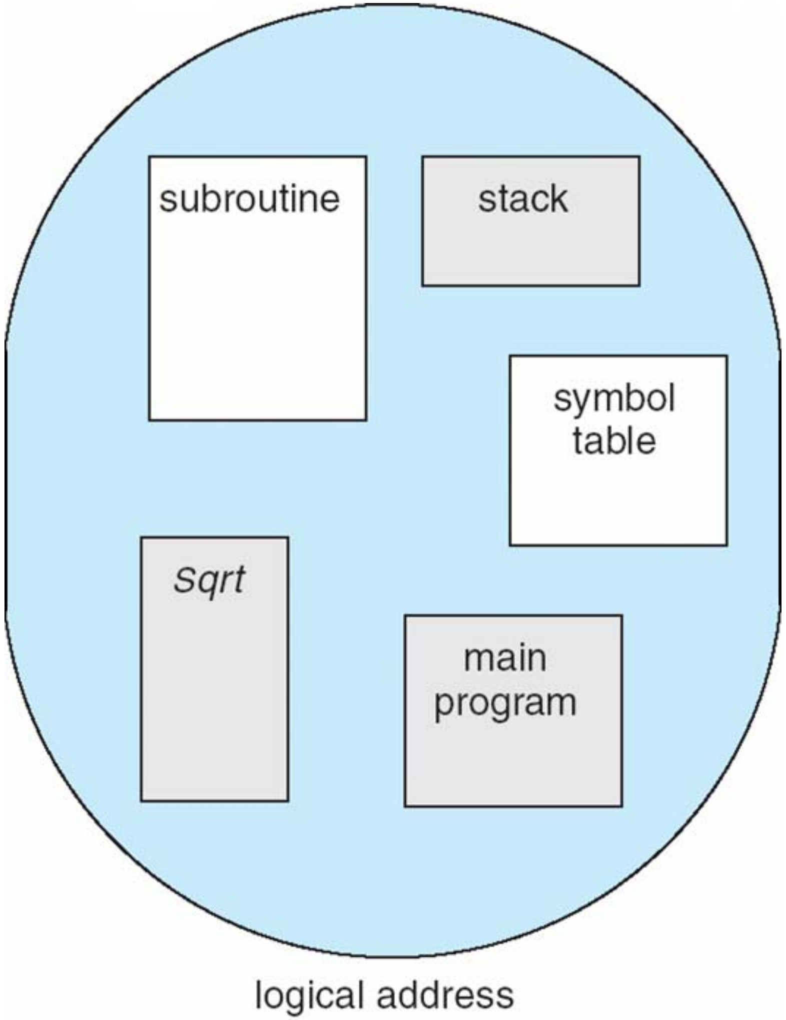


Segmentation

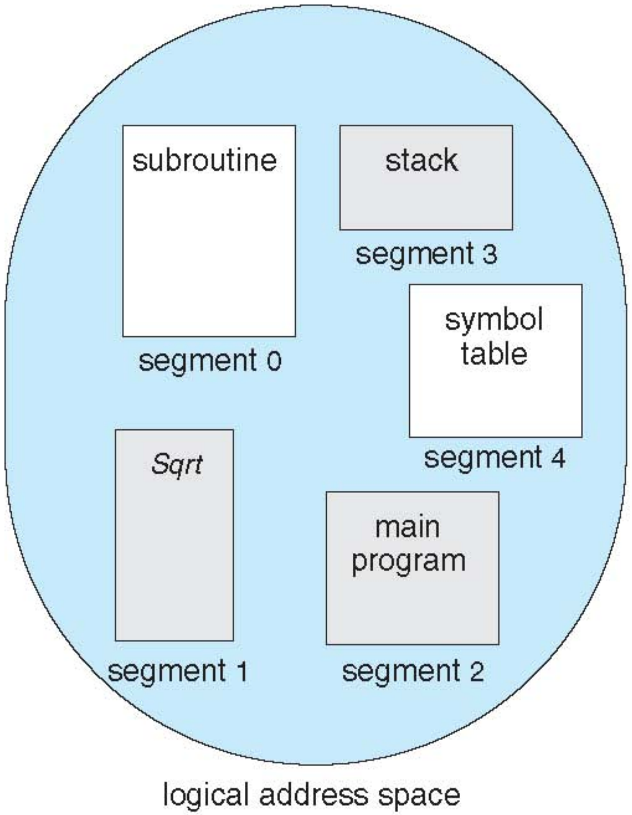
- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

    main program
    procedure
    function
    method
    object
    local variables, global variables
    common block
    stack
    symbol table
    arrays

User's View of a Program

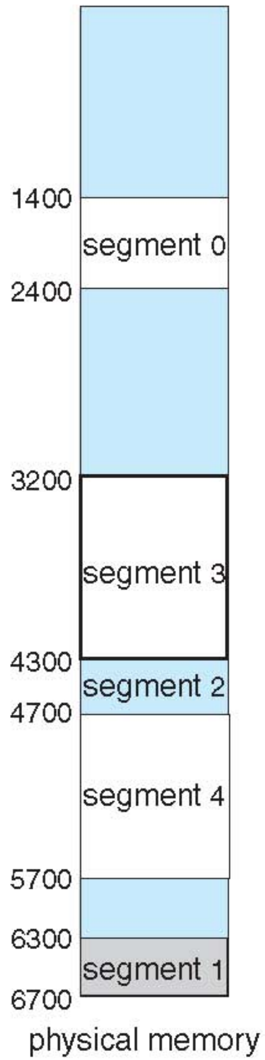


Example of Segmentation



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

segment table



Segmentation Architecture

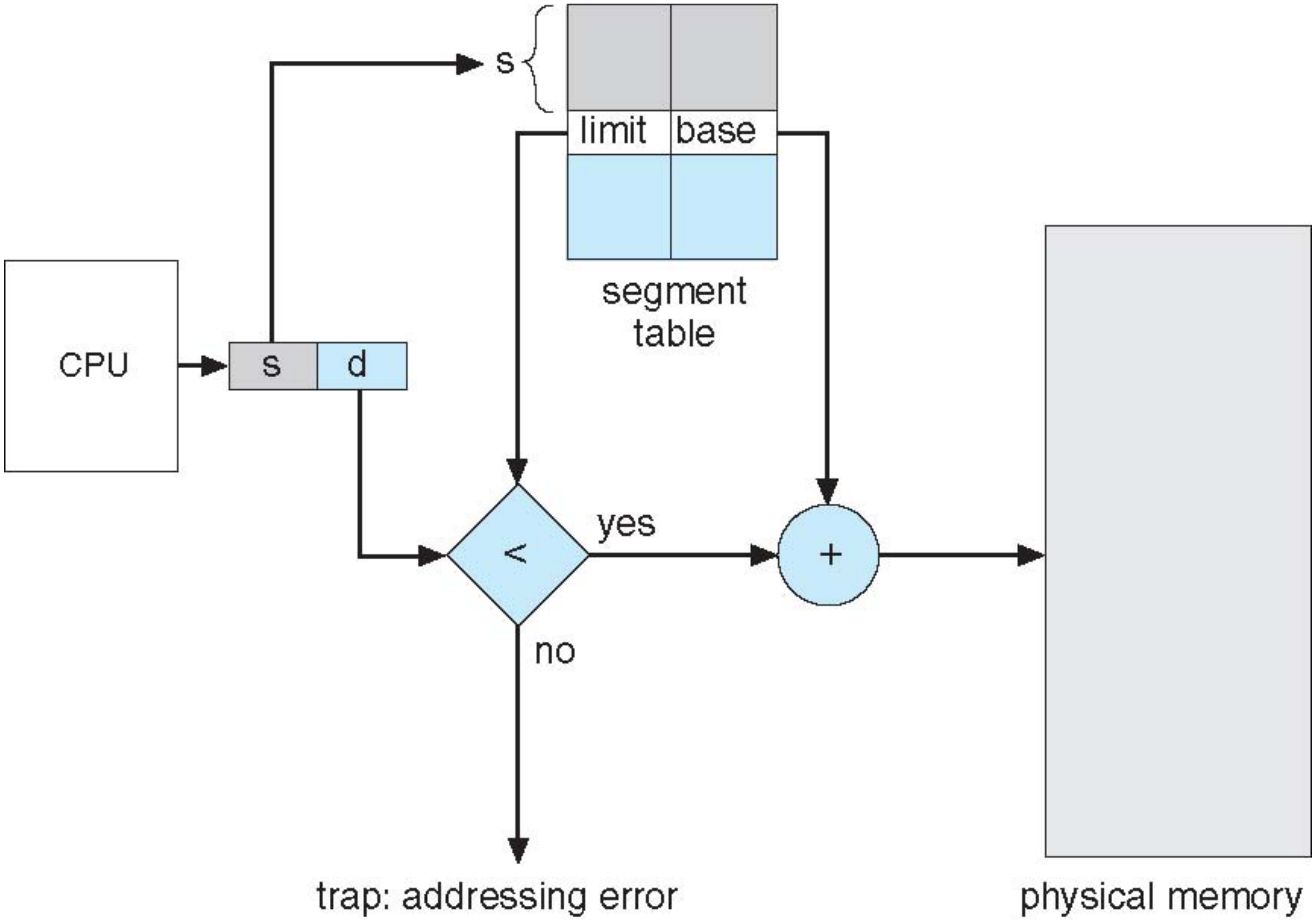
- Logical address consists of a two tuple:

  <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



Pop-Quiz

final exam.

- Consider a 32-bits logical address space
  - Two-level page table
  - 4K page size $= 2^{12}$
  - 10-bit page number
  - 10-bit page offset
  - each entry is 4 bytes

$\frac{2^{12}}{2^{2}} = 2^{10}$

page number		page offset
p_1	p_2	d
原 10	10	12

- Question: How much space is needed to store the page table?

$2^{10} \times 4 + 2^{10} \times 2^{10} \times 4 = 4KB + 4MB$

Homework

- Reading
  - Chapter 8