操作系统原理与设计

第8章 Main Memory Management (存储管理)

陈香兰

中国科学技术大学计算机学院

June 7, 2014

Outline

Problem:

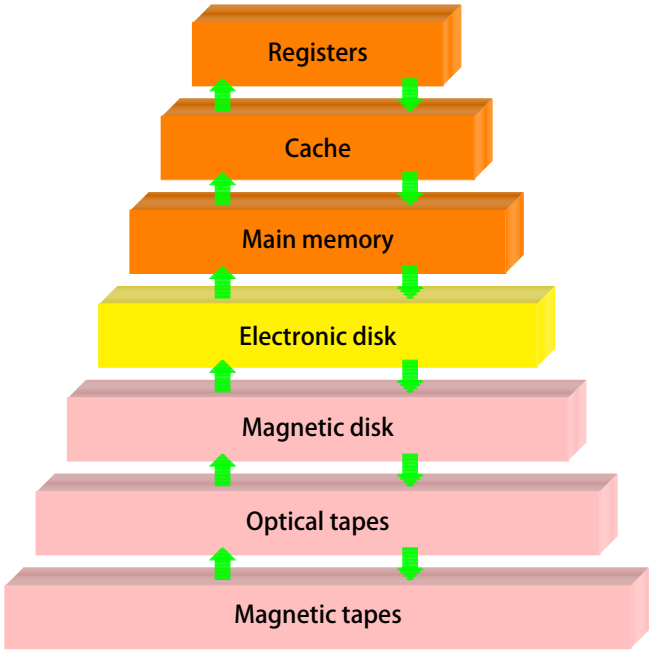
- How to load a program into MM and execute it?

Background

- Background:
  1. Storage hierarchy(存储的层次结构)
  2. Address types, address spaces and address binding
  3. Contiguous Memory Allocation(连续内存分配)
  4. Discrete Memory Allocation(离散内存分配)

Background 1: Storage hierarchy

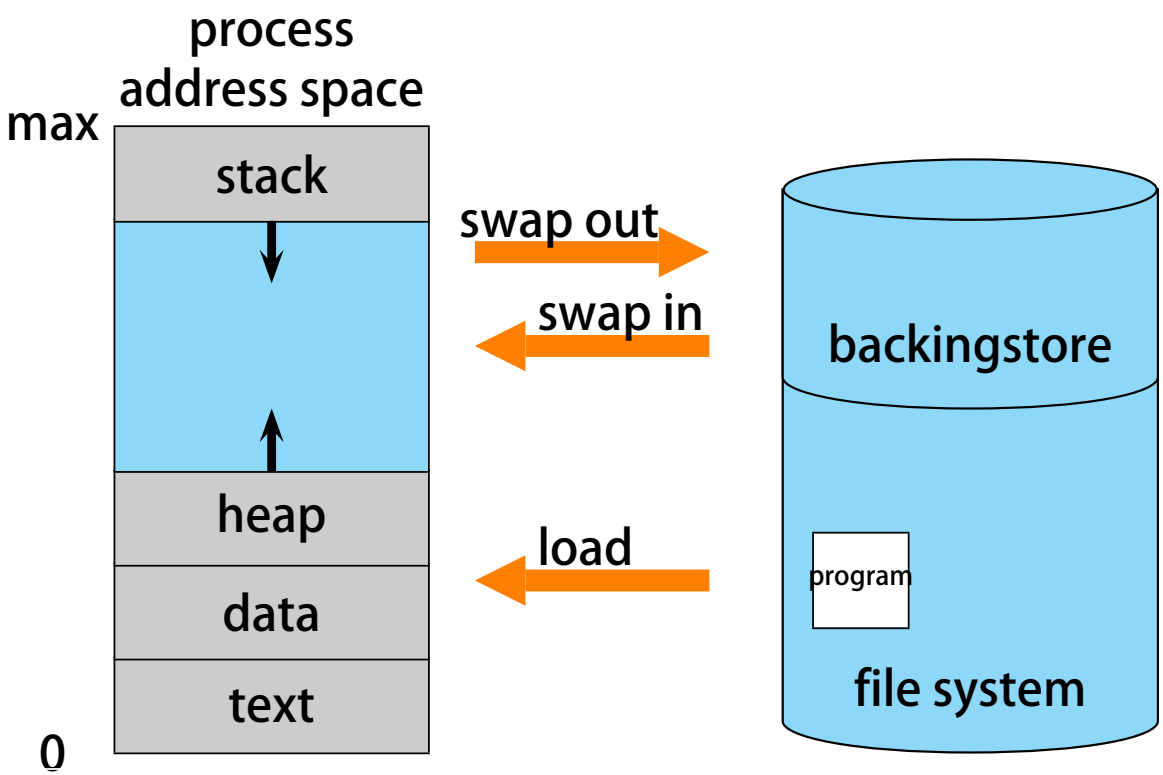
- Storage systems in a CS can be organized in a hierarchy.
  1. The contradiction(矛盾) between COST, SPEED, and CAPACITY.
  2. Volatility (易失性) VS. persistence (持久性).
- MM is a scarce resource (稀缺资源).



Storage hierarchy

Background 2-0: A typical program in execution

- A typical program in execution

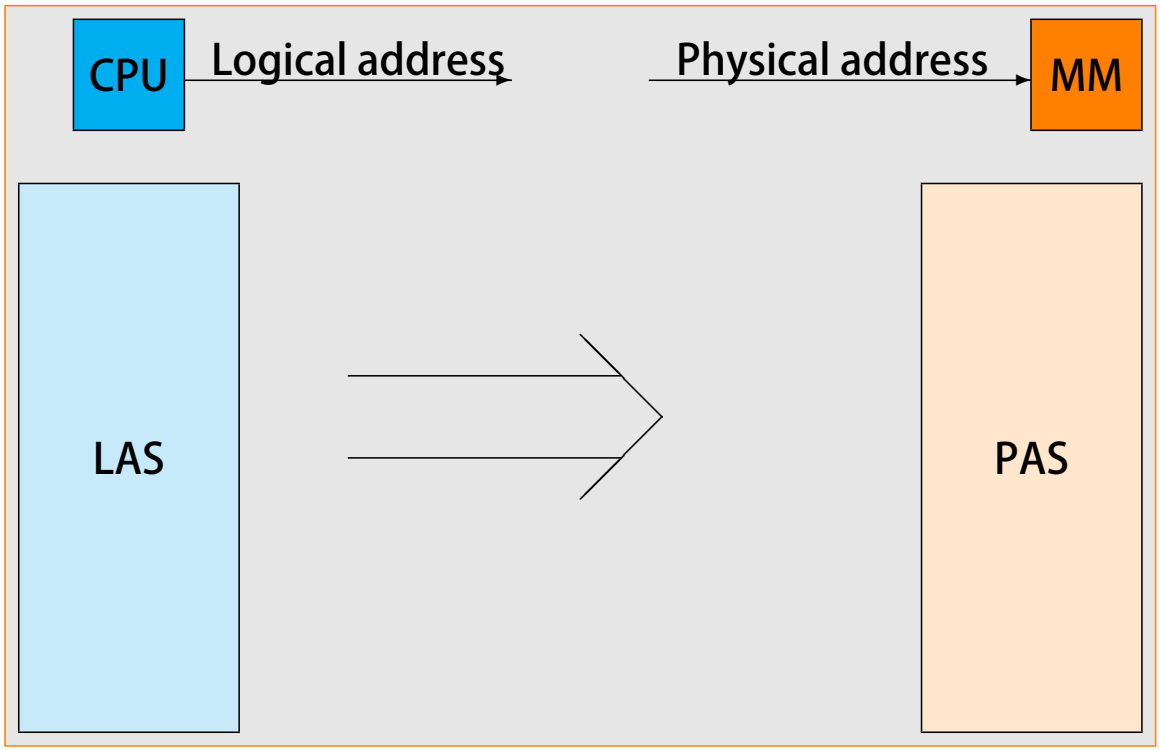


Background 2-1: Address Types

- Absolute address (绝对地址): Address seen by the memory unit
  ALSO: Physical address (物理地址)
- Relative address (相对地址)
  ALSO: Linear address (线性地址)
- Logical address (逻辑地址): Generated by the CPU
  ALSO: Virtual address (虚拟地址)

Background 2-2: Logical vs. Physical Address Space

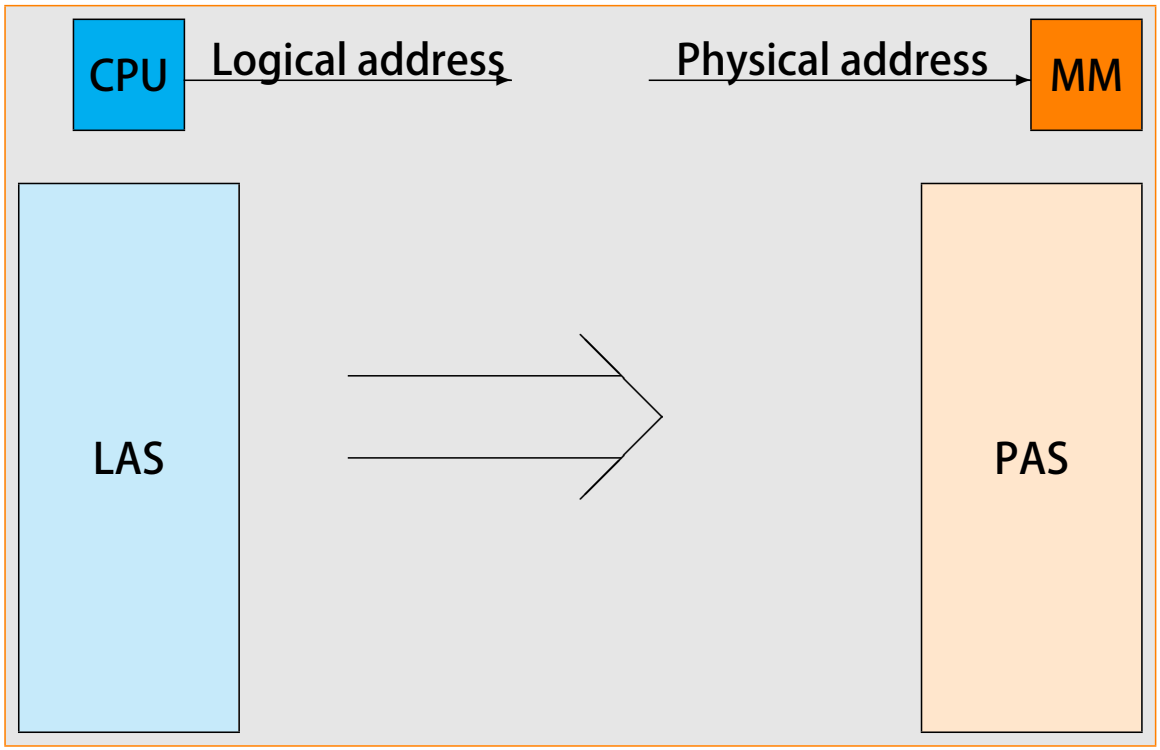
- **Logical address space**:
  the set of all logical address generated by a program
- **Physical address space**:
  the set of all physical address



- WHEN can the absolute address be decided?

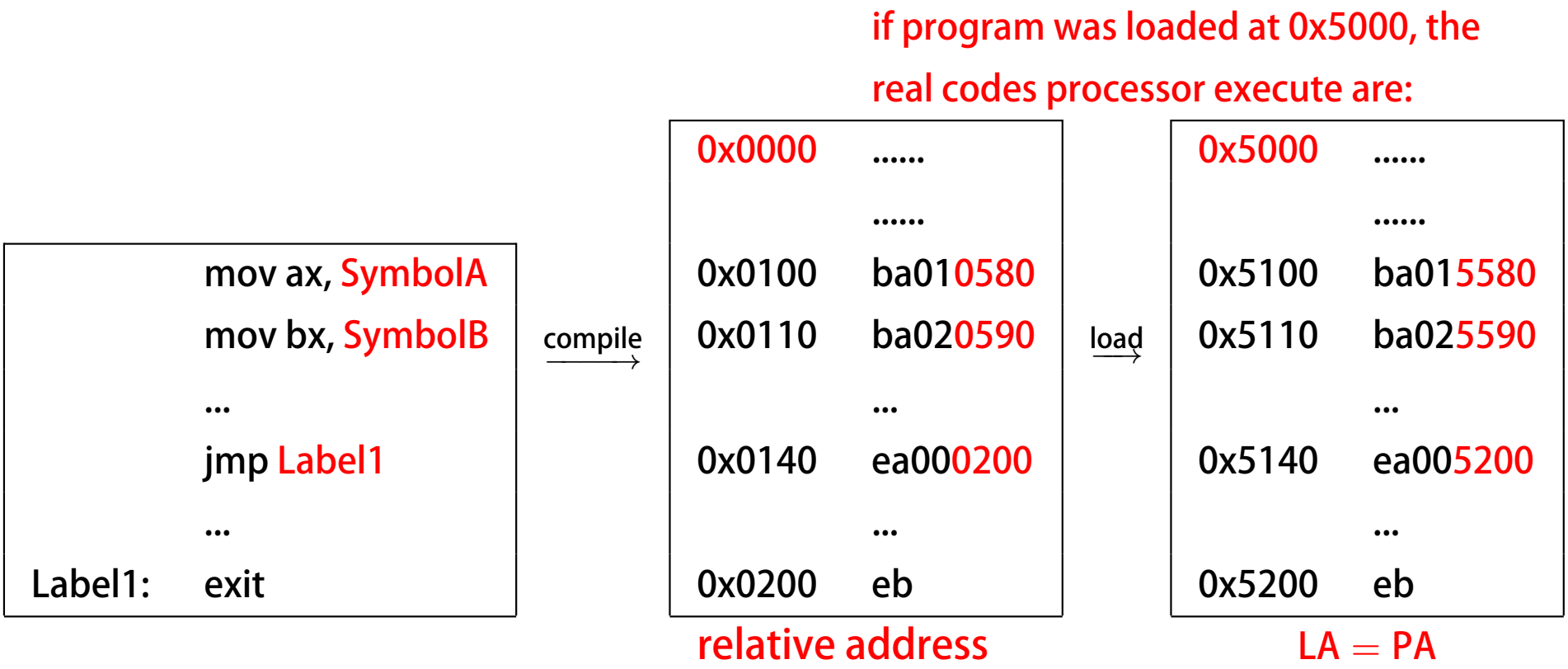
Background 2-2: Logical vs. Physical Address Space

- **Logical address space**:
  the set of all logical address generated by a program
- **Physical address space**:
  the set of all physical address



- **WHEN** can the absolute address be decided?

Example

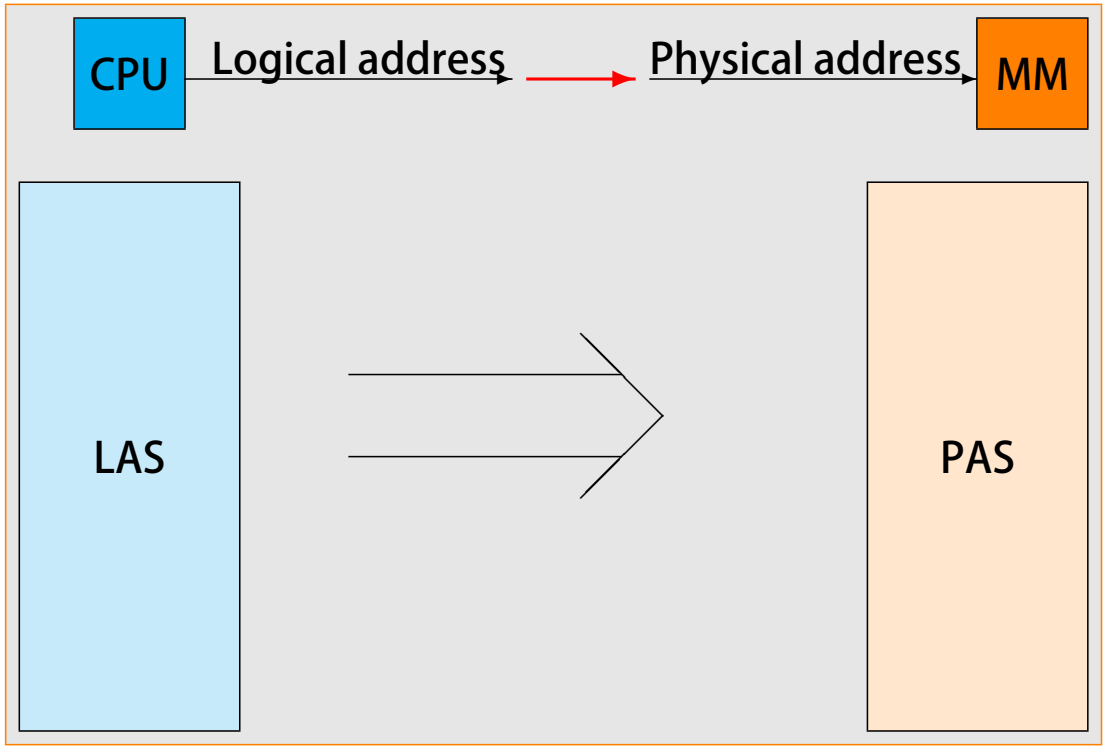

if program was loaded at 0x5000, the
real codes processor execute are:
mov ax, SymbolA
mov bx, SymbolB
...
jmp Label1
...
Label1: exit
compile
0x0000 .....
.....
0x0100 ba010580
0x0110 ba020590
...
0x0140 ea000200
...
0x0200 eb
load
0x5000 .....
.....
0x5100 ba015580
0x5110 ba025590
...
0x5140 ea005200
...
0x5200 eb
relative address
LA = PA

Background 2-3: Address binding

- Address binding of instructions and data to memory addresses can happen at three different stages
  1. Compile time:
     - If memory location known a priori, absolute code (绝对代码) can be generated;
     - Must recompile code if starting location changes;
     - Example: MS-DOS .COM-format programs
  2. Load time:
     - Must use relocatable code (可重定位代码)
  3. Execution time:
     - Binding delayed until run time.
     - Need hardware support for address maps

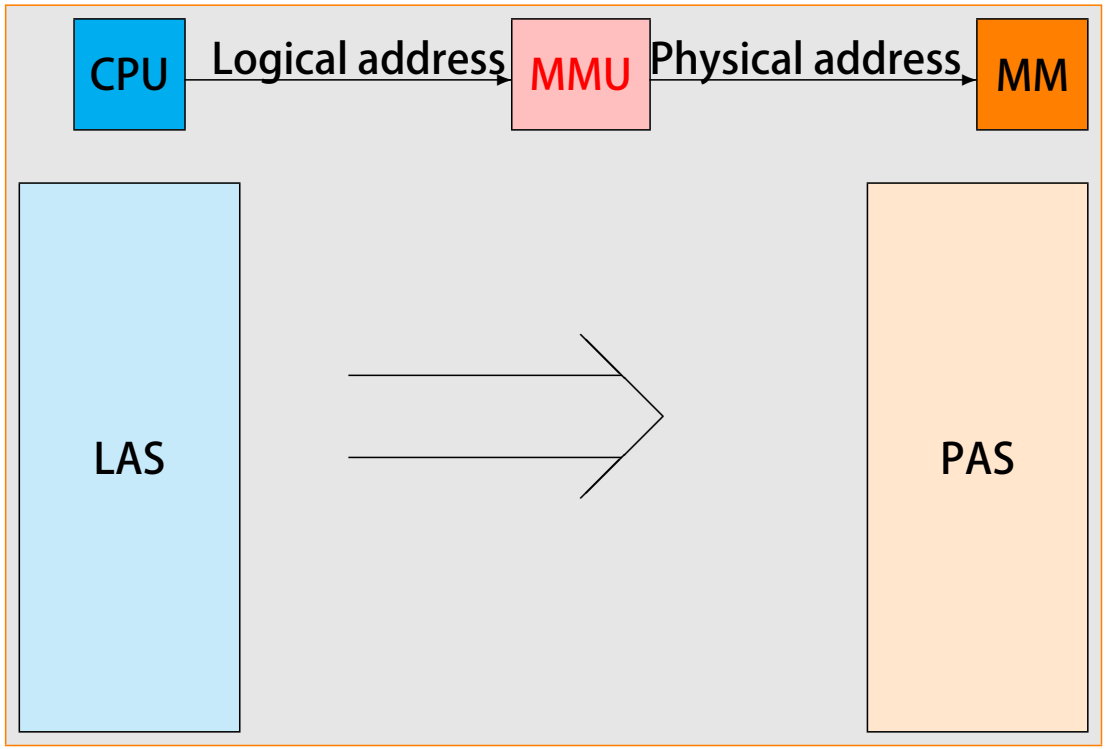
Background 2-3: Address binding

1. In compile-time and load-time address-binding schemes:
   - Logical address = physical address



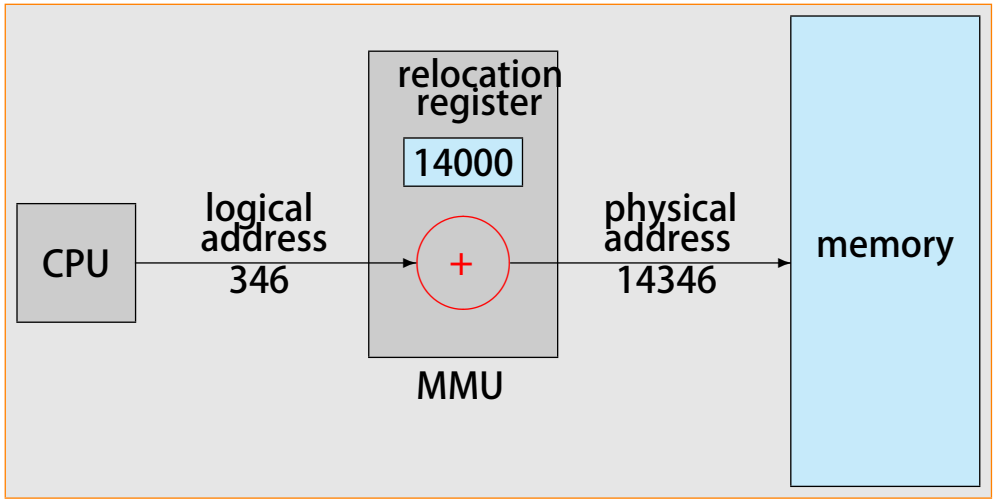
Background 2-3: Address binding

2. In execution-time address-binding scheme:
   - Logical address $\neq$ physical address
   - Need MMU



Background 2-4: Memory-Management Unit (MMU)

- MMU: Hardware device that maps logical to physical address
  - Example: dynamic relocation using a relocation register
    - The value in the relocation register (重定位寄存器) is added to every address generated by a user process at the time it is sent to memory

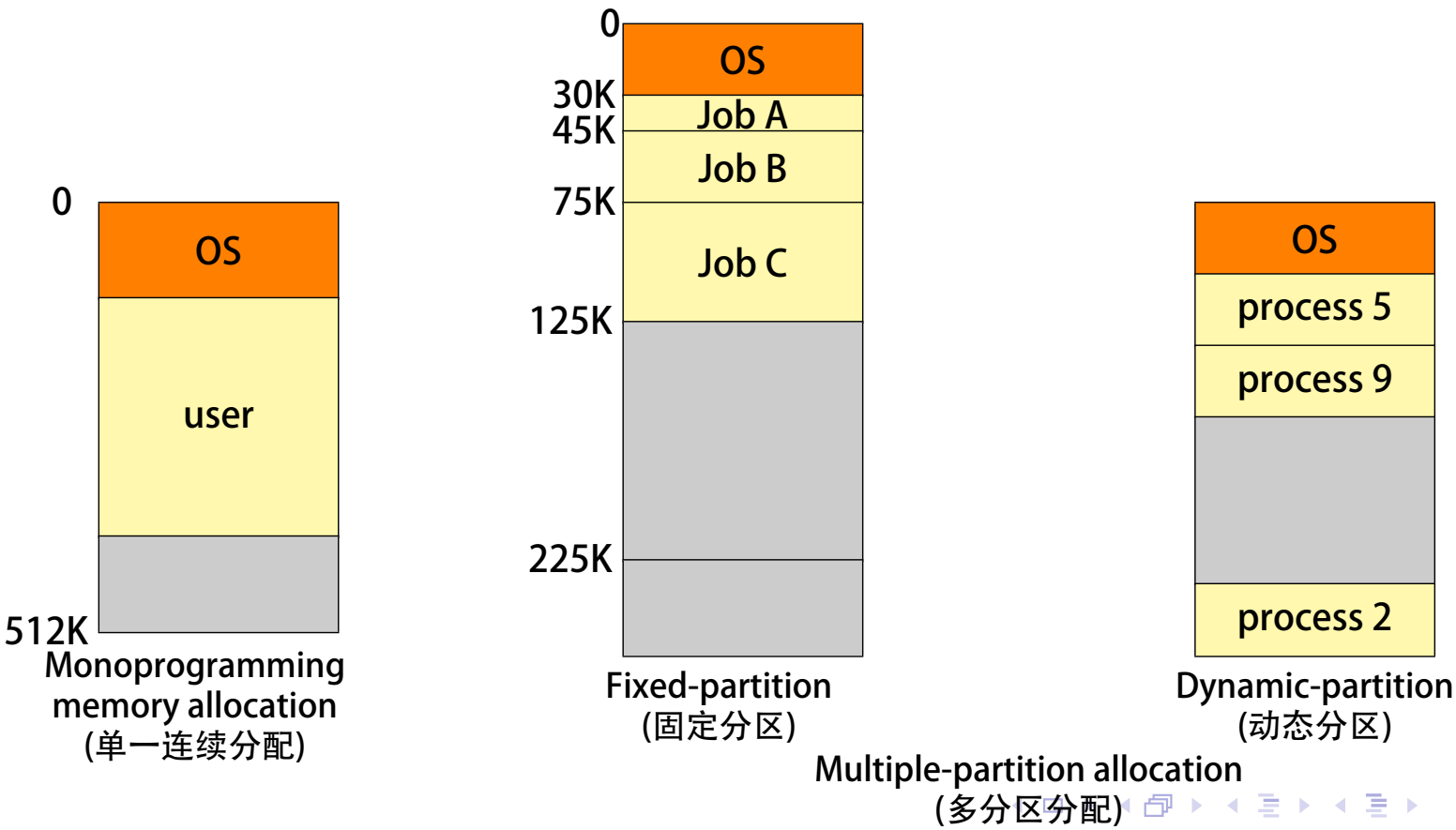


- The user program deals with logical addresses [0, MAX);
  It never sees the real physical addresses [R+0, R+MAX)

Background 3: Contiguous Memory Allocation (连续内存分配)

Contiguous Memory Allocation

Each process is contained in a single contiguous section of memory.



Background 3: Contiguous Memory Allocation (连续内存分配)

- Disadvantage of multiple-partition allocation
  - Poor memory utility
  - Internal fragmentation(内部碎片)
    External fragmentation(外部碎片)
- Solution: Reduce external fragmentation by compaction (紧凑)
  - Need dynamic relocation at execution time
    (运行时的动态可重定位技术)

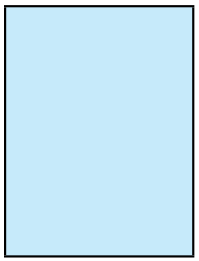
Background 4: Discrete (离散) Memory Allocation

1. Paging（分页）√
   - Internal fragmentation $<$one page
2. Segmentation（分段）
   - Logical
3. Combined paging & segmentation（段页式）

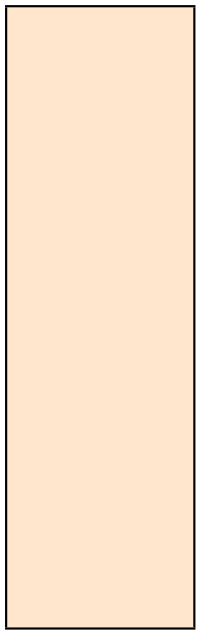
Outline

Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  1. Divide physical memory into fixed-sized blocks called frames (物理页框): size is power of 2, 512B–8,192B
     - Page Frame Number (物理页框号, PFN): $0, 1, \ldots,$ PFN_{max}



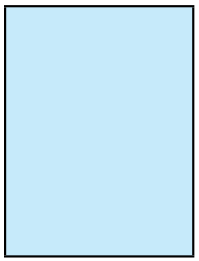
Logical memory



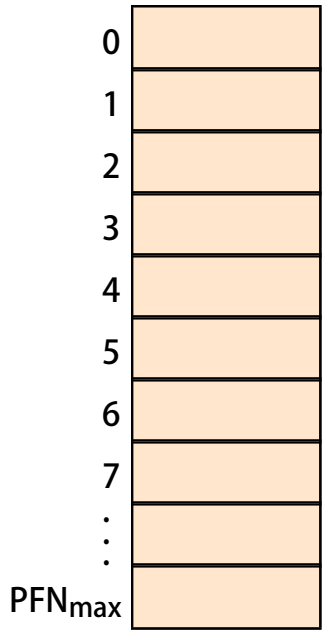
Physical memory

Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  1. Divide physical memory into fixed-sized blocks called frames (物理页框): size is power of 2, 512B–8,192B
     - Page Frame Number (物理页框号, PFN): $0, 1, \ldots,$ PFN_{max}



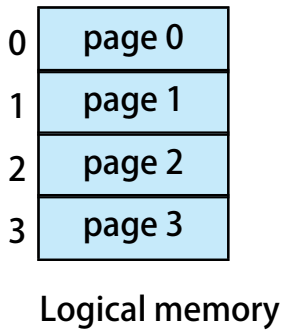
Logical memory



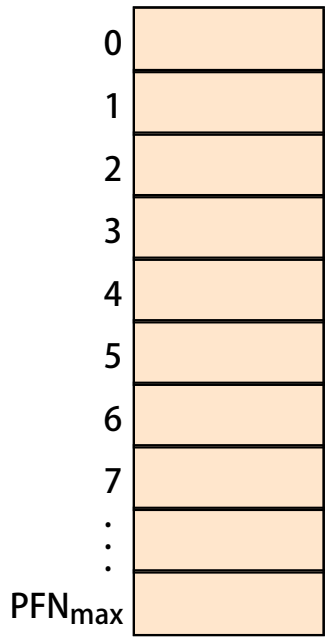
Physical memory

Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  2. Divide logical memory into blocks of same size called pages (逻辑页，页)
     - Logical Frame Number (逻辑页框号, LFN): $0, 1, \ldots,$ LFN_{max}



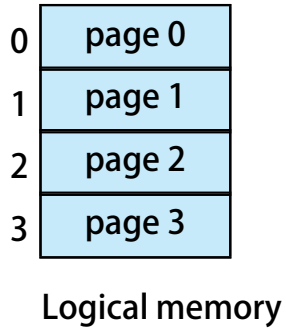
Logical memory

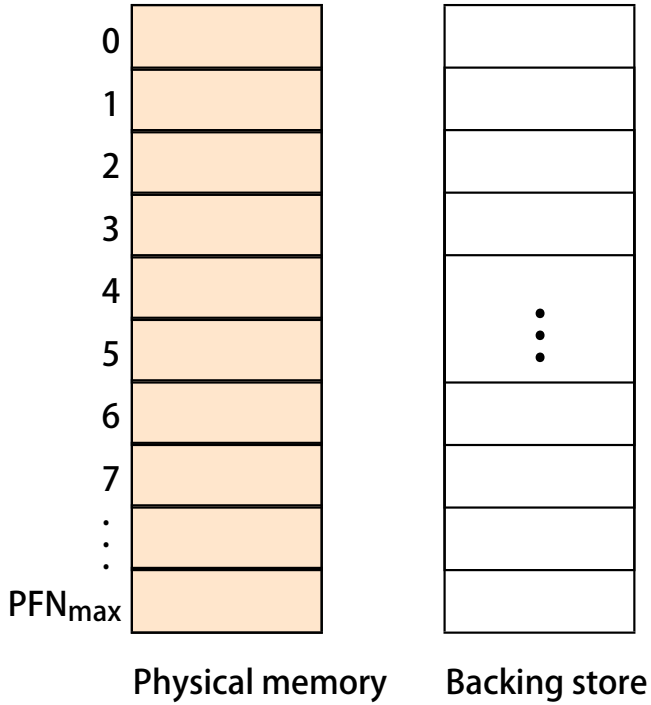


Physical memory

Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  3. The backing store is also divided into fixed-sized blocks of same size as frames



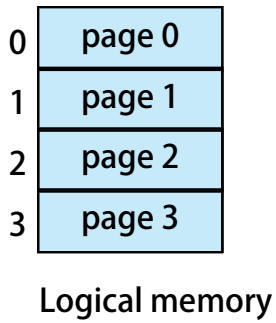


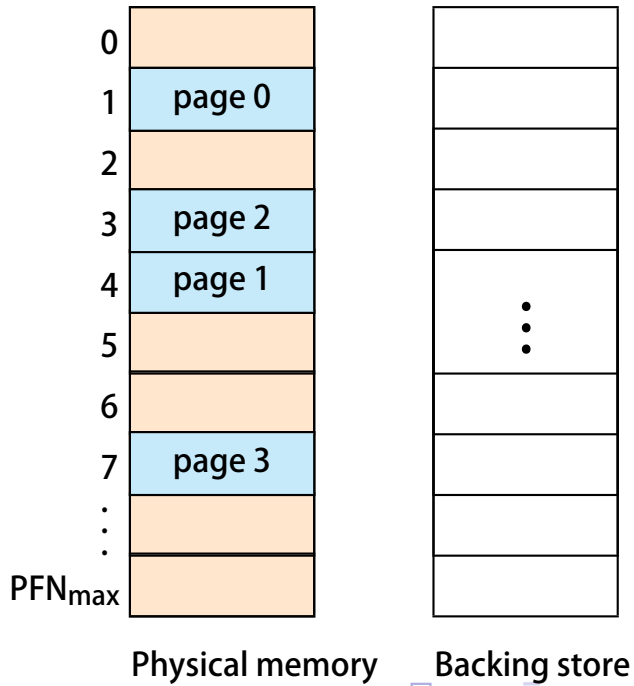
Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  4. Need hardware and software support for paging
     1. Keep track of all free frames

Paging (分页)

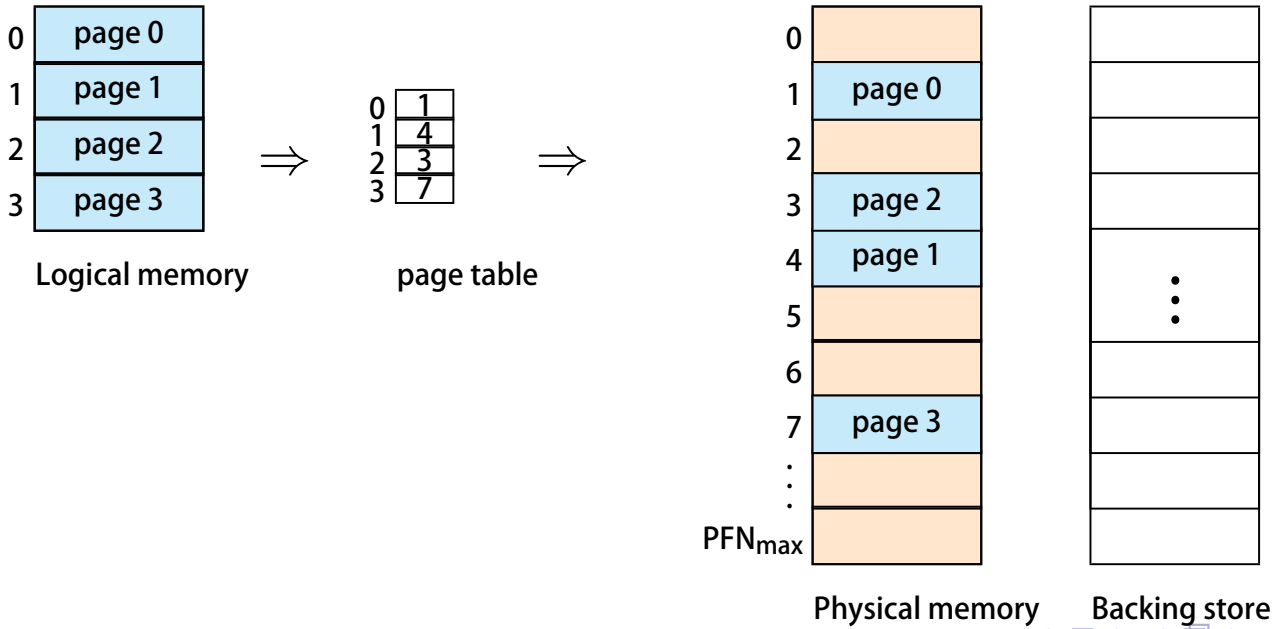
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  4. Need hardware and software support for paging
     1. Keep track of all free frames
     2. To run a program of size n pages, need to find n free frames and load program





Paging (分页)

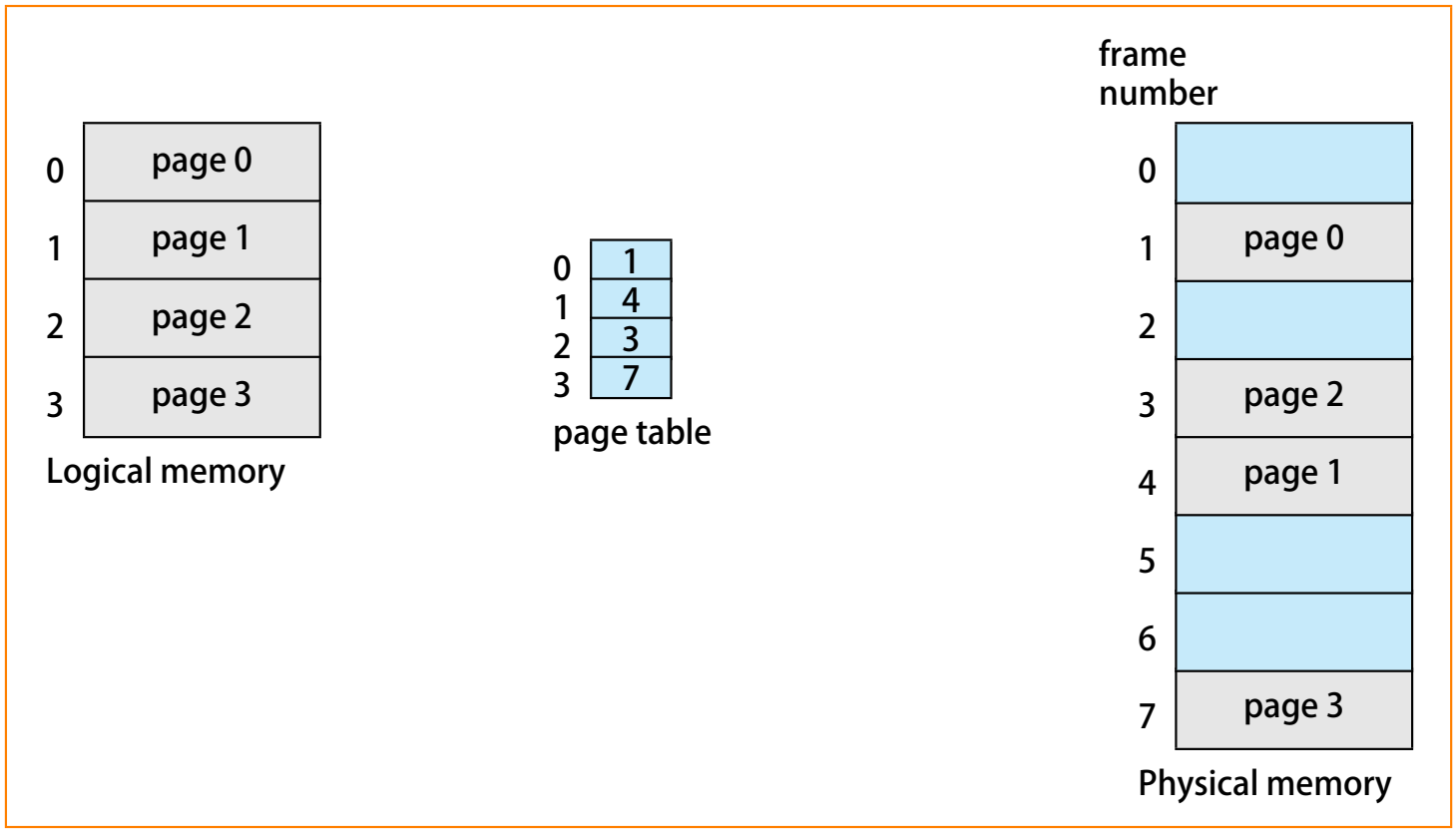
- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  4. Need hardware and software support for paging
     1. Keep track of all free frames
     2. To run a program of size n pages, need to find n free frames and load program
     3. Set up a page table to translate logical to physical addresses for each process



Paging (分页)

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
- Basic Method
  1. Divide physical memory into fixed-sized blocks called frames (物理页框): size is power of 2, 512B−8,192B
     - Page Frame Number (物理页框号, PFN): $0, 1, \ldots,$ PFN_{max}
  2. Divide logical memory into blocks of same size called pages (逻辑页， 页)
     - Logical Frame Number (逻辑页框号, LFN): $0, 1, \ldots,$ LFN_{max}
  3. The backing store is also divided into fixed-sized blocks of same size as frames
  4. Need hardware and software support for paging
     1. Keep track of all free frames
     2. To run a program of size n pages, need to find n free frames and load program
     3. Set up a page table to translate logical to physical addresses for each process
- Internal fragmentation $<$ page size

Paging Model of Logical and Physical Memory



Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number (p), LFN
  - Page offset (d)

- How to get p and d?

Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number (p), LFN
  - Page offset (d)

- How to get p and d?
  - Let
    A: An address, either logical address or physical address
    L: The size of a page or page frame
    p and d: The corresponding number of the page (frame), and page offset

$$\begin{cases} p = A \,\%\, L \\ d = A \bmod L \end{cases}$$

Address Translation Scheme

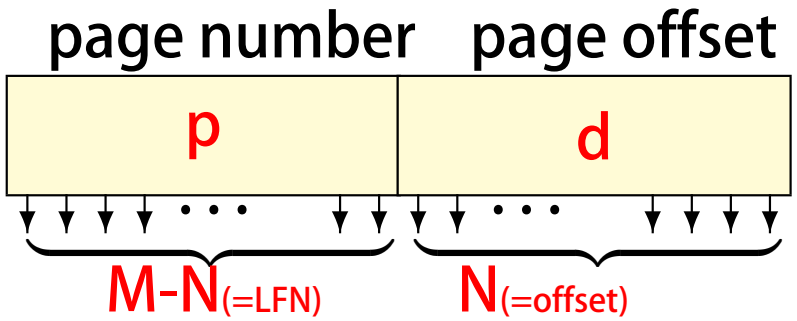
- Address generated by CPU is divided into:
  - Page number (p), LFN
  - Page offset (d)

- How to get p and d?
  - Suppose $L = 2^N$:

$$\begin{cases} p = A \text{ right\_shrift } N, \text{ 即A的高}(M-N)\text{位} \\ d = \text{A的低端N位} \end{cases}$$

Address Translation Scheme

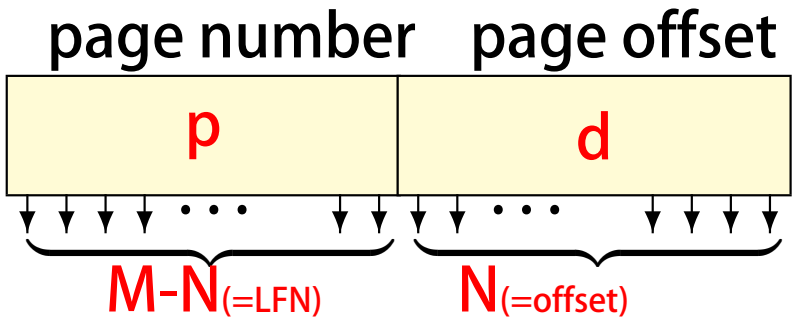
- Address generated by CPU is divided into:
  - Page number (p), LFN
  - Page offset (d)



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

Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number (p), LFN
  - Page offset (d)



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

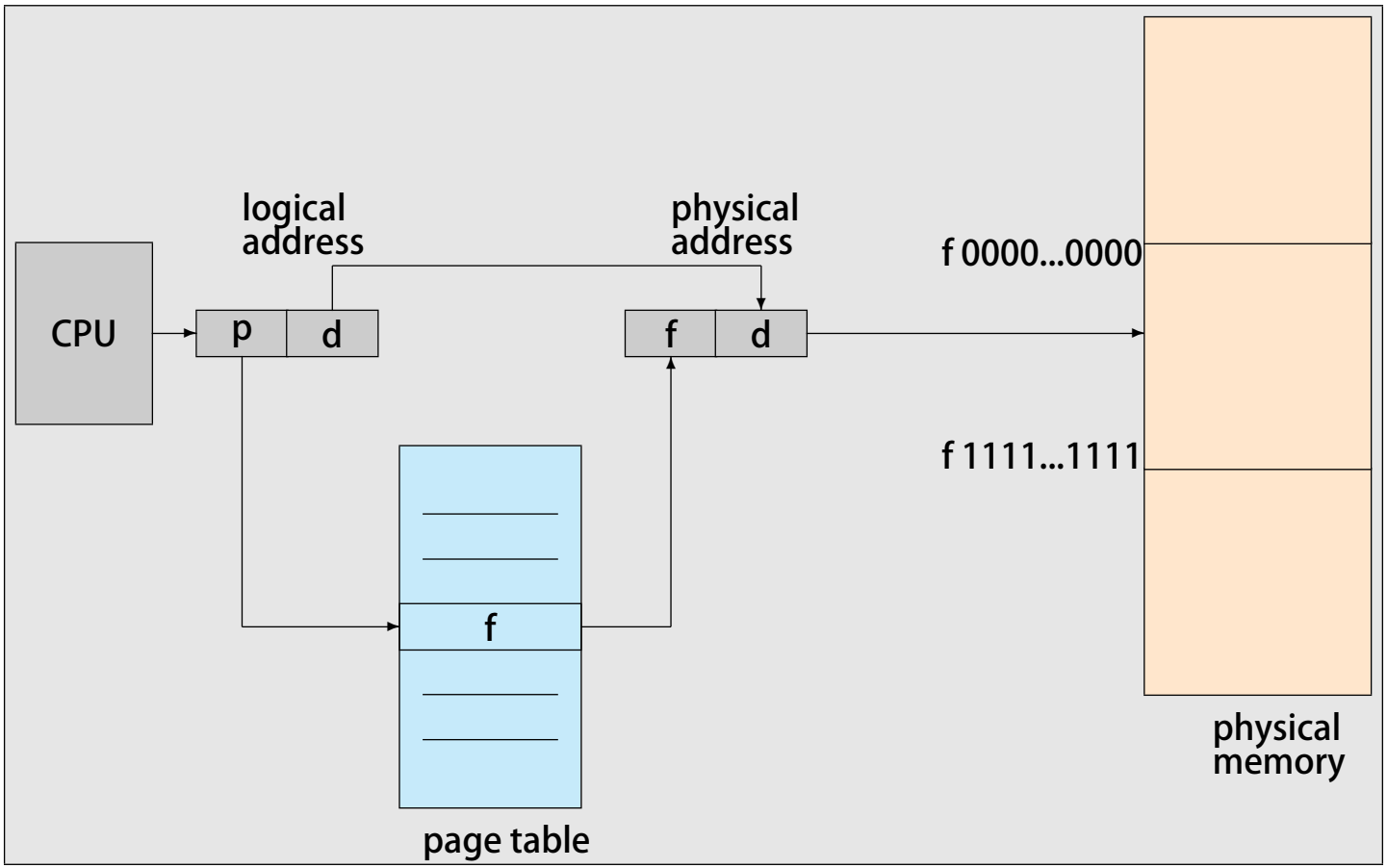
- For 32bits system & 4KB page size, M $= 32$, N $= 12$, M $-$ N $= 20$

Example : A $=$ 0x $\underbrace{12345}_{p}$ $\underbrace{678}_{d}$

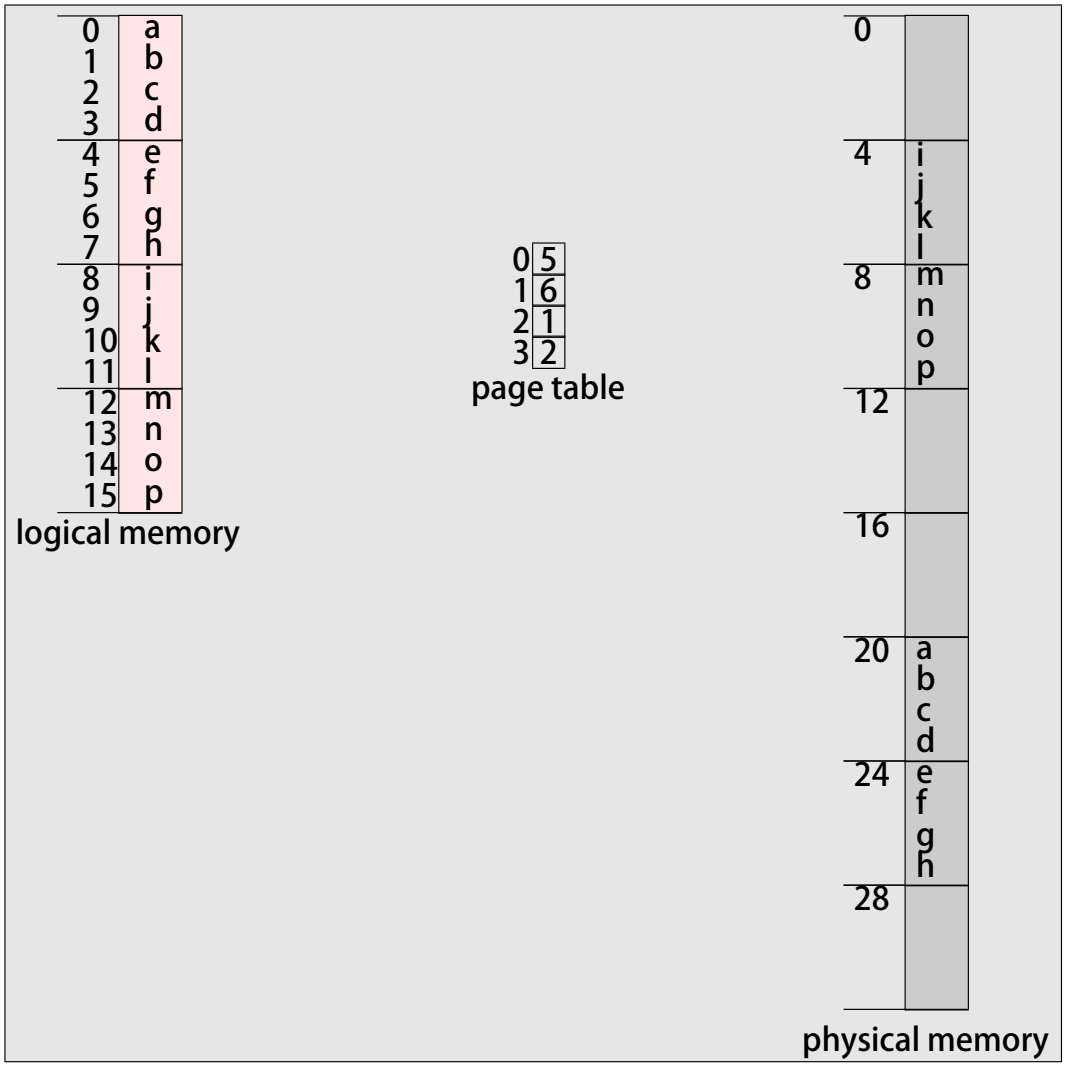
Address Translation Scheme: Paging Hardware

- Paging Hardware:

$$\underbrace{\text{LFN (p)} + \text{offset (d)}}_{\text{Logical address}} \rightarrow \underbrace{\text{PFN (f)} + \text{offset (d)}}_{\text{Physical address}}$$



Paging Example



32-byte memory with 4-byte pages

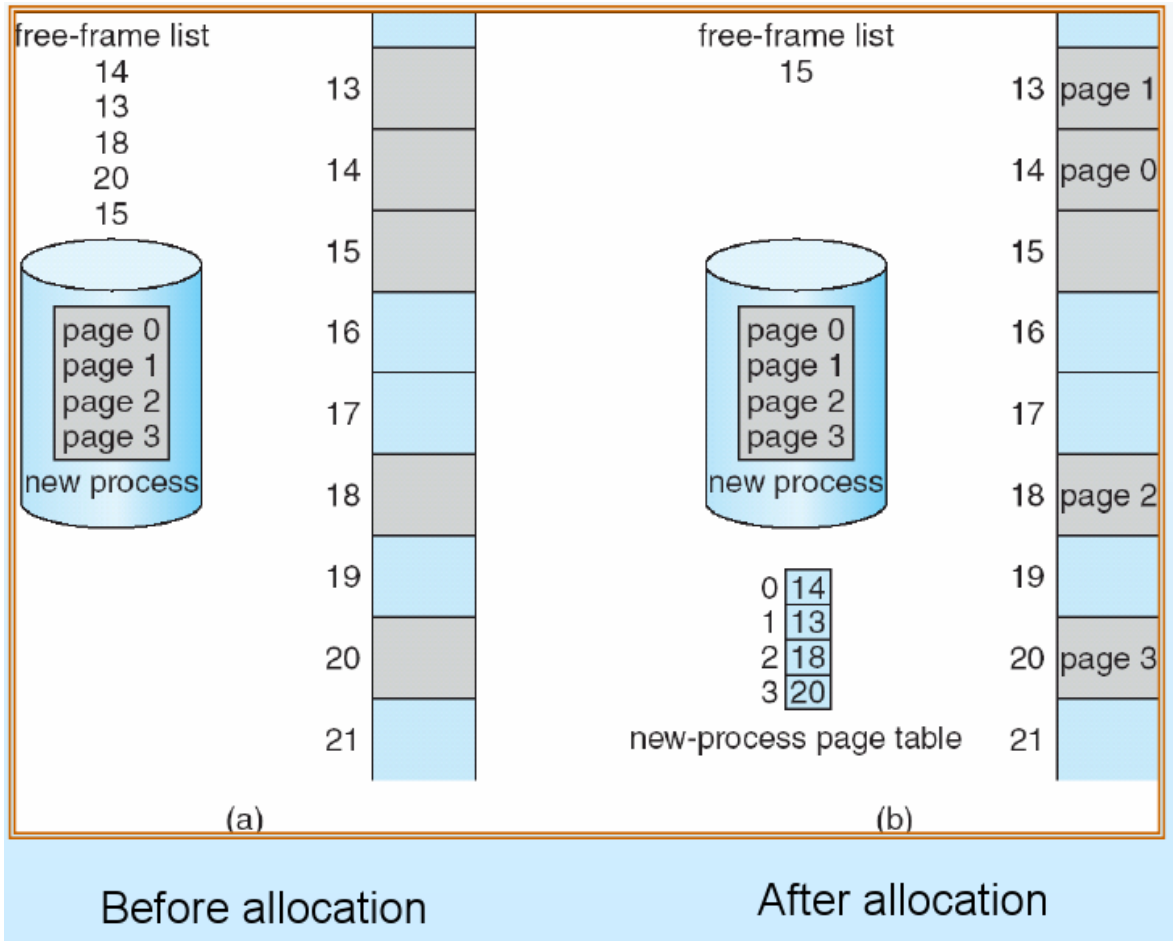
- What if read logical address 9?

Keep track of all frames

- Since OS is managing physical memory, it must be aware of the allocation details of physical memory
  - which frames are allocated
  - which frames are available
  - how many total frames
  - ...
- Frame table: one entry for each physical page frame

Keep track of all frames

- **Free-frame list**



Conclusion: basic method

1. Paging(分页)
2. Address translation scheme and paging hardware
3. Keep track of all frames

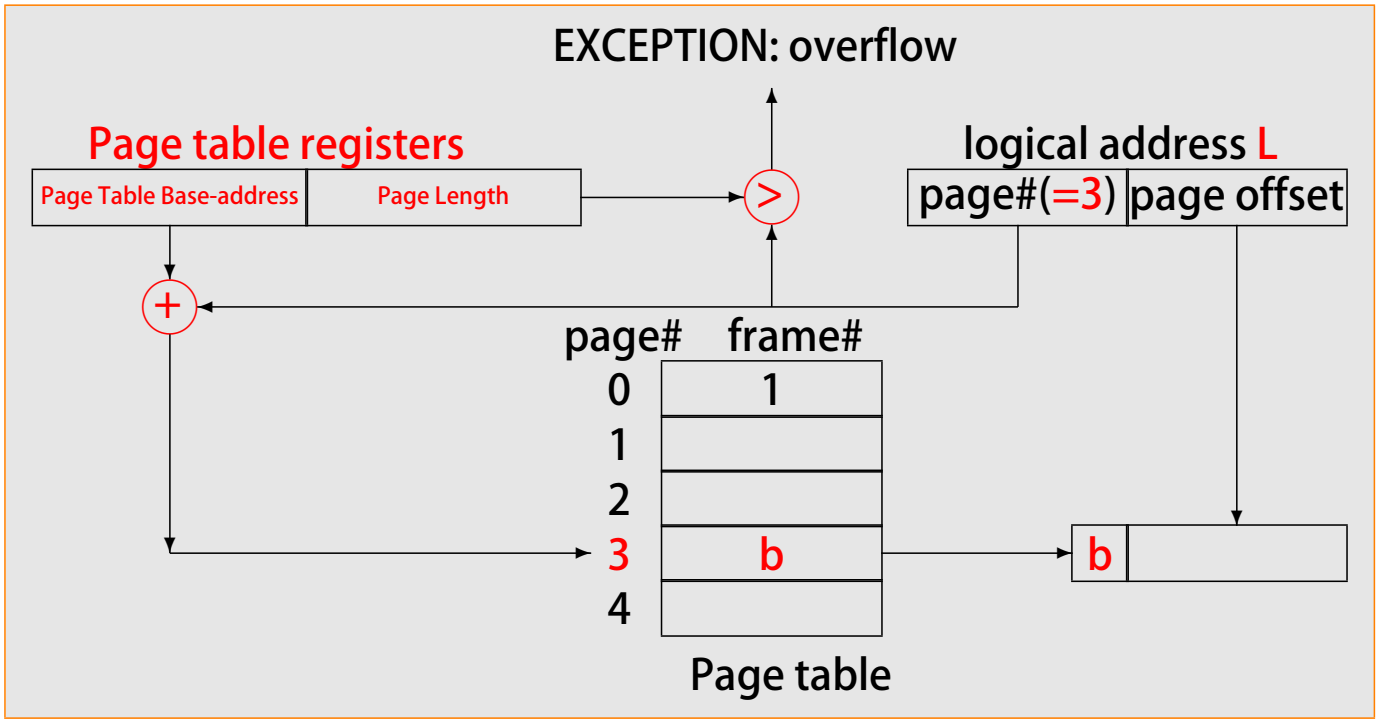
Outline

Hardware support

- Special hardware (software) is needed to implement page table
  1. Basic paging hardware
  2. Paging hardware with TLB

Hardware support

1. Implementation of Page Table : basic paging hardware
   - 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



Hardware support

1. Implementation of Page Table : basic paging hardware
   - 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
   - Context switch: HOW?
     - Each process is associated with a page table.
     - Page table must be switched, too.

Hardware support

1. Implementation of Page Table : basic paging hardware
   - Effective memory-Access Time (EAT, 有效访问时间)
     - Every data/instruction access requires two memory accesses.

     1. One for the page table
     2. One for the data/instruction.

Assume memory cycle time is t microsecond, then

$$\text{EAT} = 2\text{t}$$

Hardware support

1. Implementation of Page Table : basic paging hardware
   - Effective memory-Access Time (EAT, 有效访问时间)
     - Every data/instruction access requires two memory accesses.
       1. One for the page table
       2. One for the data/instruction.

Assume memory cycle time is t microsecond, then

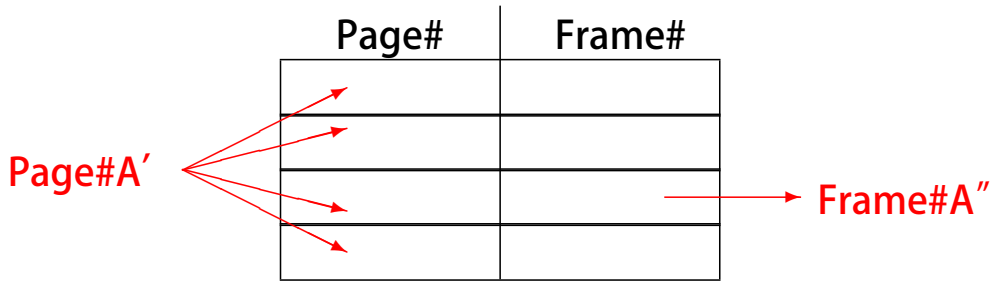
$$\text{EAT} = 2\text{t}$$

   - Solution to two memory access problem:
     - A special fast-lookup hardware cache called associative memory or translation look-aside buffers (TLBs, 快表)

Hardware support

2 Paging Hardware With TLB

- **Associative Memory**
  - Each register: a key & a value
  - Parallel search (high speed)
  - Expensive, typically 8~2048 entries

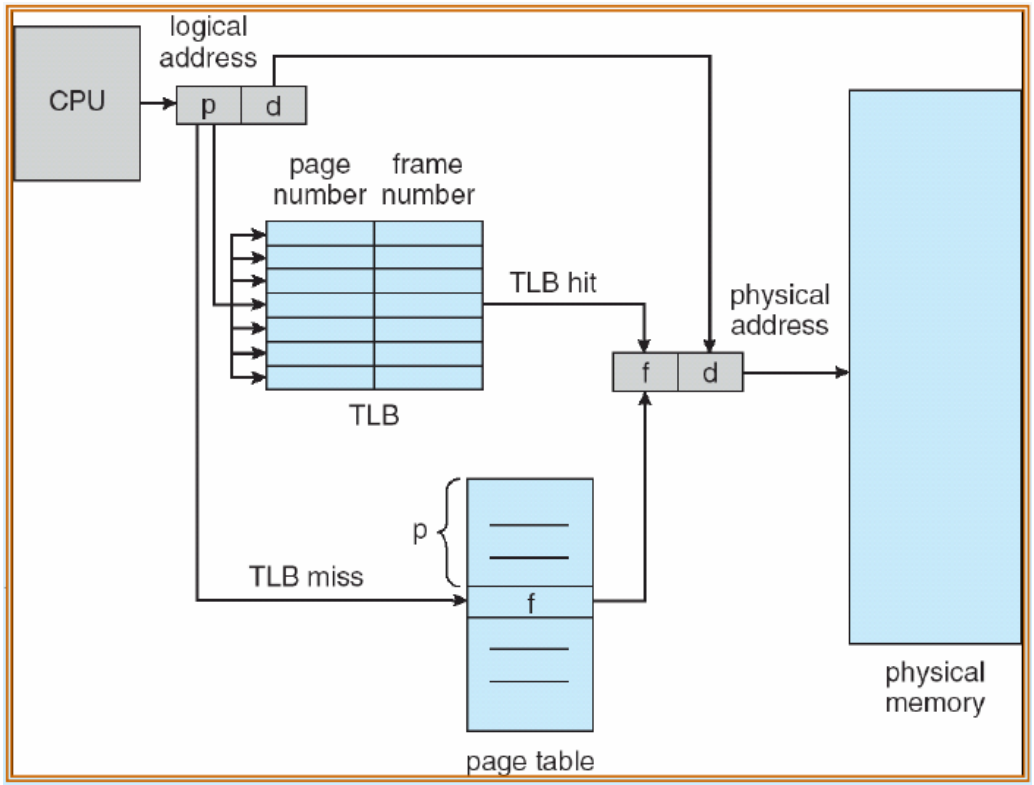


Address translation (A′ , A″)

- If A′ is in associative register, get frame # out
- Otherwise get frame # from page table in memory

Hardware support

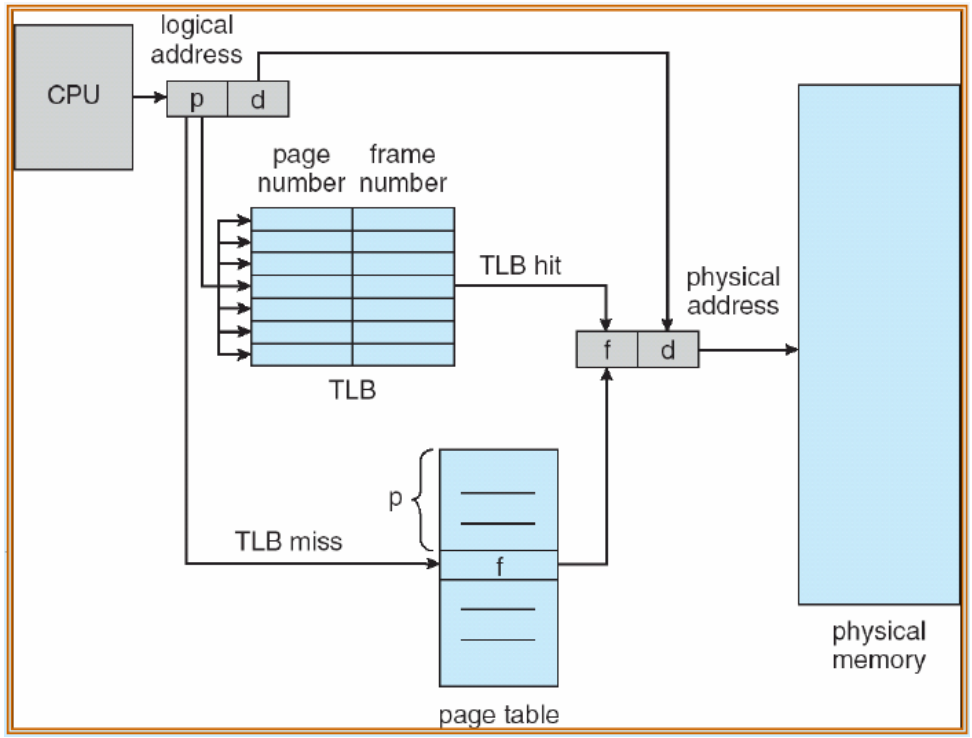
2 Paging Hardware With TLB



- For Context Switch:
  - TLB must be flushed after context is switched!

Hardware support

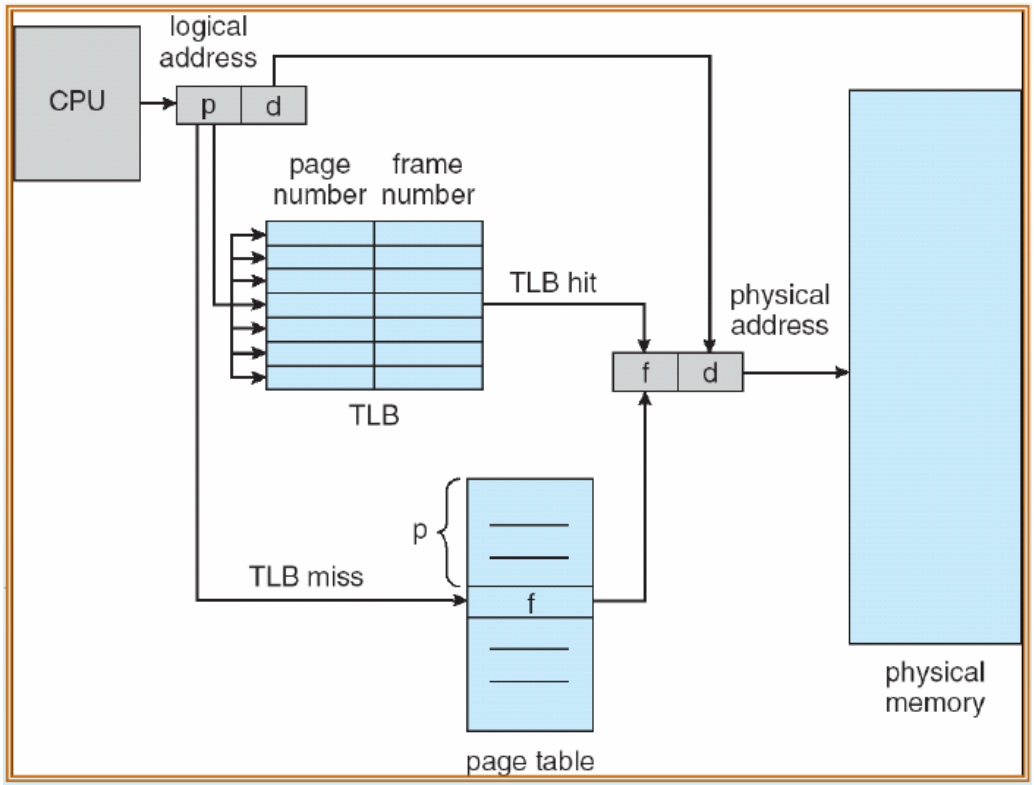
2 Paging Hardware With TLB



- Some TLBs store address-space identifiers (ASIDs) in each TLB entry
  - Uniquely identifies each process to provide address-space protection for that process

Hardware support

2 Paging Hardware With TLB



- NOTE: CACHE VS. TLB

TLB Miss

- TLB miss (TLB缺失)
  - If the page number is not in the associative registers
    - Get & store
- TLB hit (TLB命中) and Hit ratio (命中率)
  - Hit ratio:
    The percentage of times that a page number is found in the associative registers
  - Ratio related to number of associative registers

1. What will be happened if the TLB is full?
   - TLB replacement algorithm
2. What will be happened after context is swiched?

Effective Access Time (有效访问时间)

- If
  - Associative Lookup = ϵ time unit
  - Assume memory cycle time is t microsecond
  - Hit ratio = α
- Then Effective Access Time (EAT)

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

- If $\epsilon = 20$ns, t $= 100$ns, $\alpha 1 = 80\%$, $\alpha 2 = 98\%$:
  - If TLB hit: $20 + 100 = 120$ns
  - If TLB miss: $20 + 100 + 100 = 220$ns
  - EAT$1 = 120 * 0.8 + 220 * 0.2 = 140$ns
  - EAT$2 = 120 * 0.98 + 220 * 0.02 = 122$ns

Conclusion: hardware support

- Special hardware (software) is needed to implement page table
  1. Basic paging hardware
     - EAT
  2. Paging hardware with TLB
     - Associate Memory
     - TLB hit; TLB miss
     - EAT

Conclusion

Thanks!
请提问!