Memory Management (Part 1)

Operating Systems

Assoc. Prof. Milos Jovanovik, PhD

Memory Management

- Ideal memory
  - Big, fast, cheap, private, long-term;
- Memory hierarchy
  - A few MB of very fast, expensive, volatile cache memory
  - A few GB of medium-speed, medium-priced, volatile main memory
  - A few TB of slow, cheap, nonvolatile disk storage
- Memory manager
  - It manages the memory hierarchy in order to build an ideal memory

Memory Hierarchy

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 μs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia

Why is Memory Management Important?

- Memory is an important resource and it has to be carefully managed
- Parkinson’s law: „The programs expand with a tendency to fill all available memory“
- Memory has to be efficiently allocated in order to have more processes in the main memory
- It should allow programs bigger than the available physical memory to run

Memory Management Rules (1)

- Relocation
  - The programmer does not know where the program will be in the main memory
  - When the program is executed, the program can be swapped to the disk, and then copied back into the main memory on a different location (relocated)
  - Memory references (branching, data, etc.) in the code must be translated into correct physical locations

Addressing Requirements for a Process

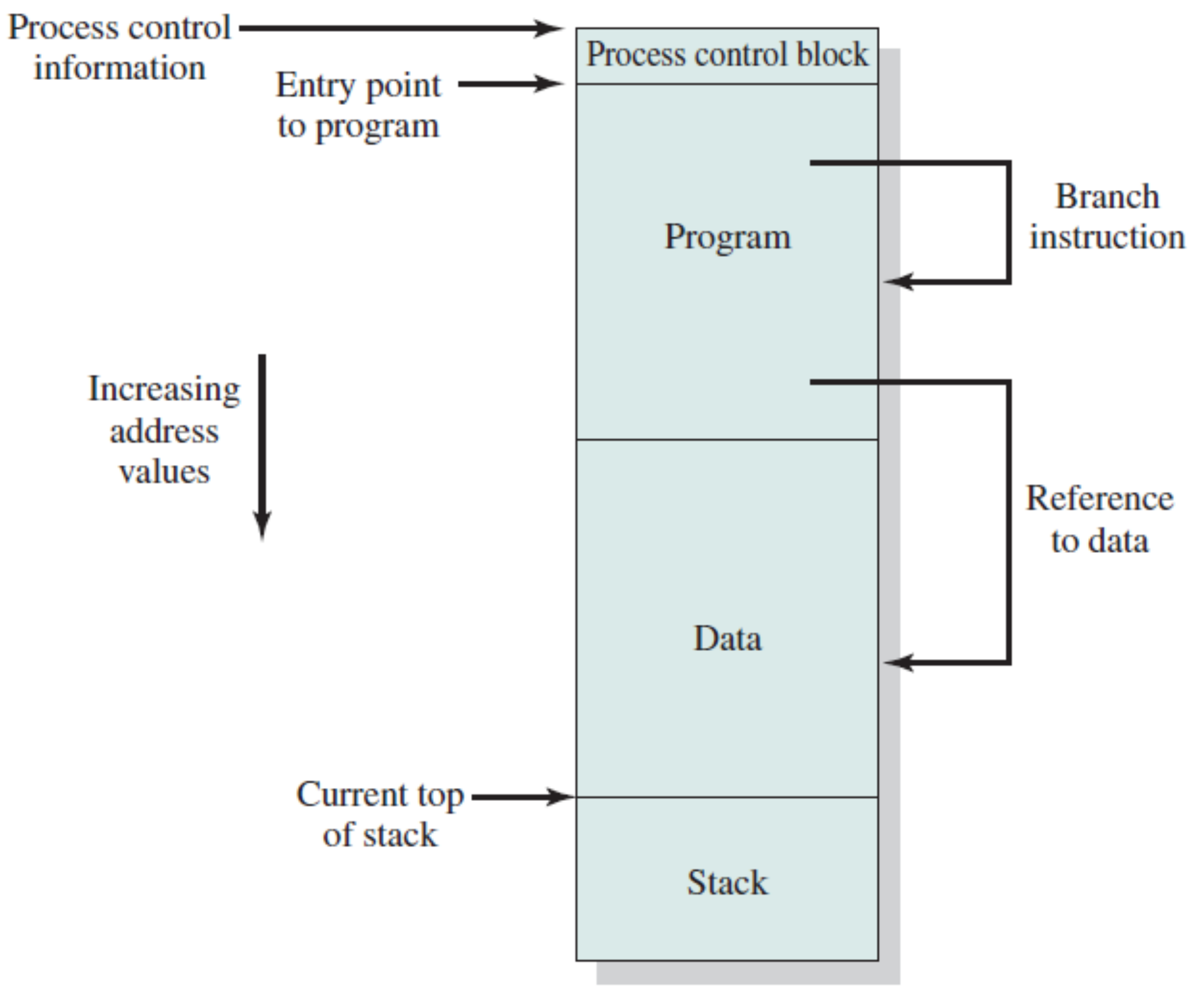


Figure 7.1 Addressing Requirements for a Process

Memory Management Rules (2)

- Protection
  - Processes cannot reference memory locations which belong to the address space of another process, without permission
  - It is not possible to check absolute addresses in programs, because the program may be relocated
  - They have to be checked in execution time
    - The OS does not have a magic stick to foresee all memory references which a process will make

Memory Management Rules (3)

- Sharing
  - It allows several process to access the same piece of memory
  - It is better to allow each process to access the same copy of the program (to read), than to have a separate copy for each of the processes

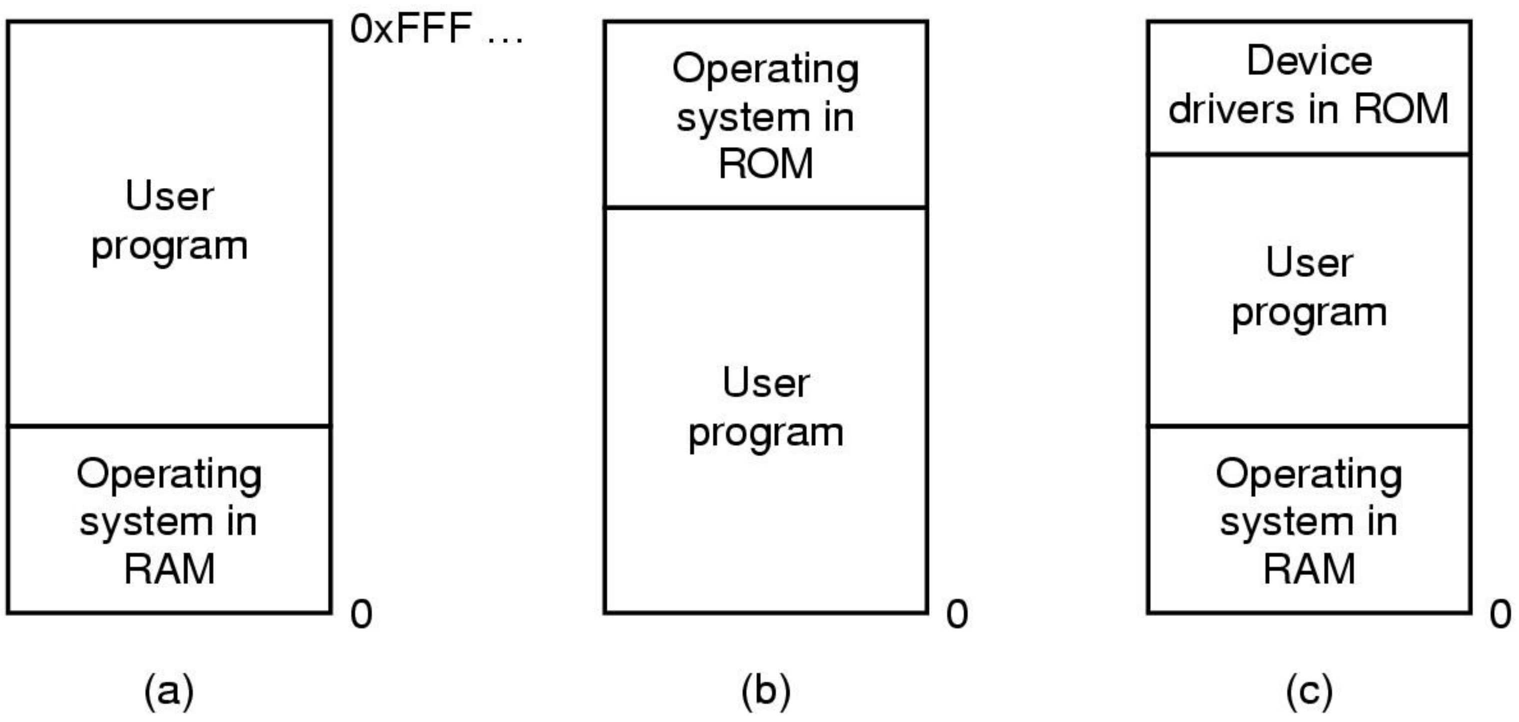
Question

- Which of the following is not a memory management rule?
  - A. Sharing
  - [✓ B. Speed]
  - C. Relocation
  - D. Protection

Memory Manager

- Part of the OS that manages with the memory (memory manager)
  - Accounting: Which parts of the memory are in use
  - Assigns: Allocates memory to processes
  - Takes away: Deallocates memory from processes
  - Swapping: Interchanges between memory and disk

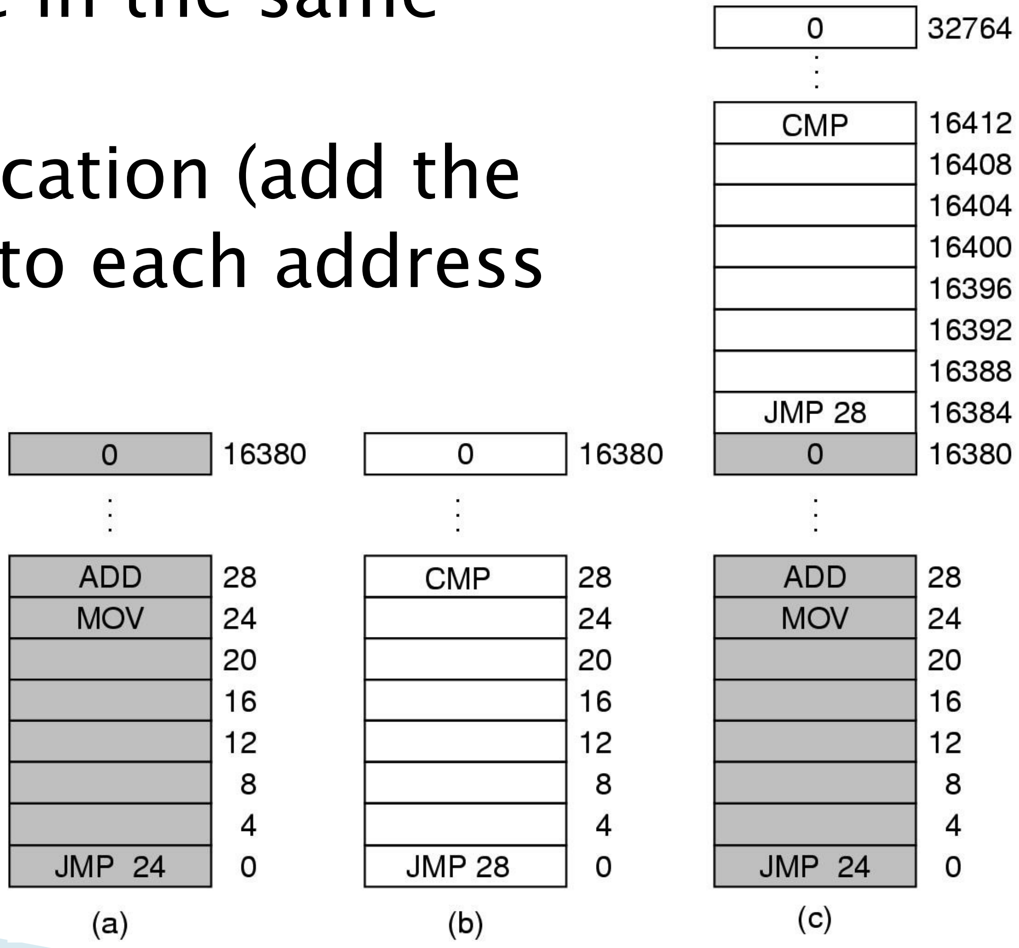
No Memory Abstraction



Three simple ways of organizing memory with an operating system and one user process.

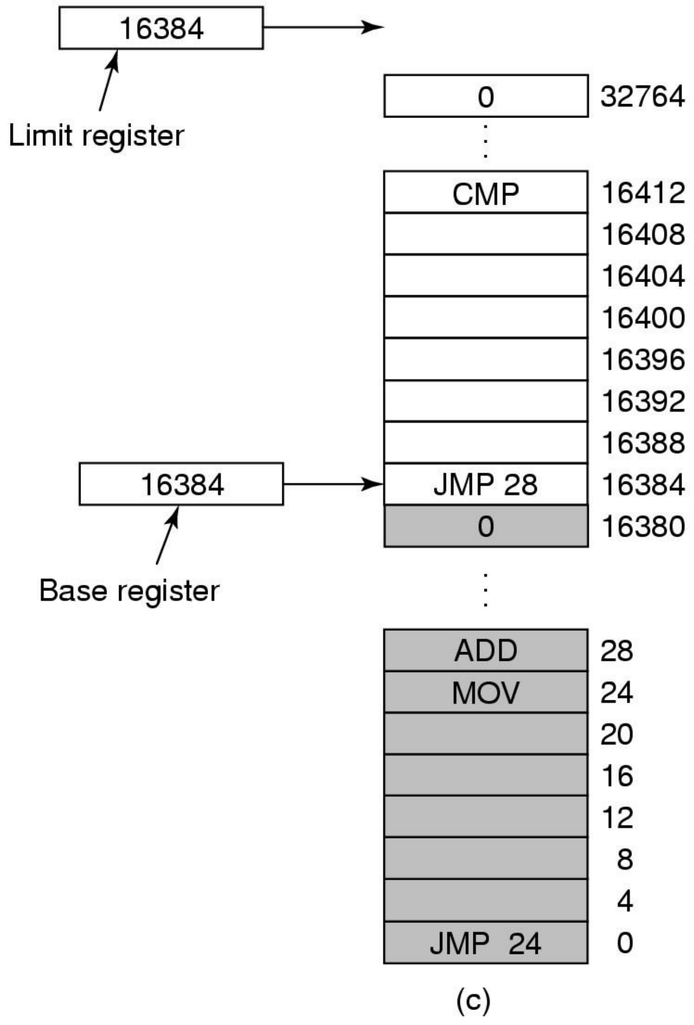
Running Multiple Programs Without a Memory Abstraction

- Relocation problem: We have two programs (a) and (b). Both are in the same address space (c).
- Solution: static relocation (add the first allocated byte to each address of the program)

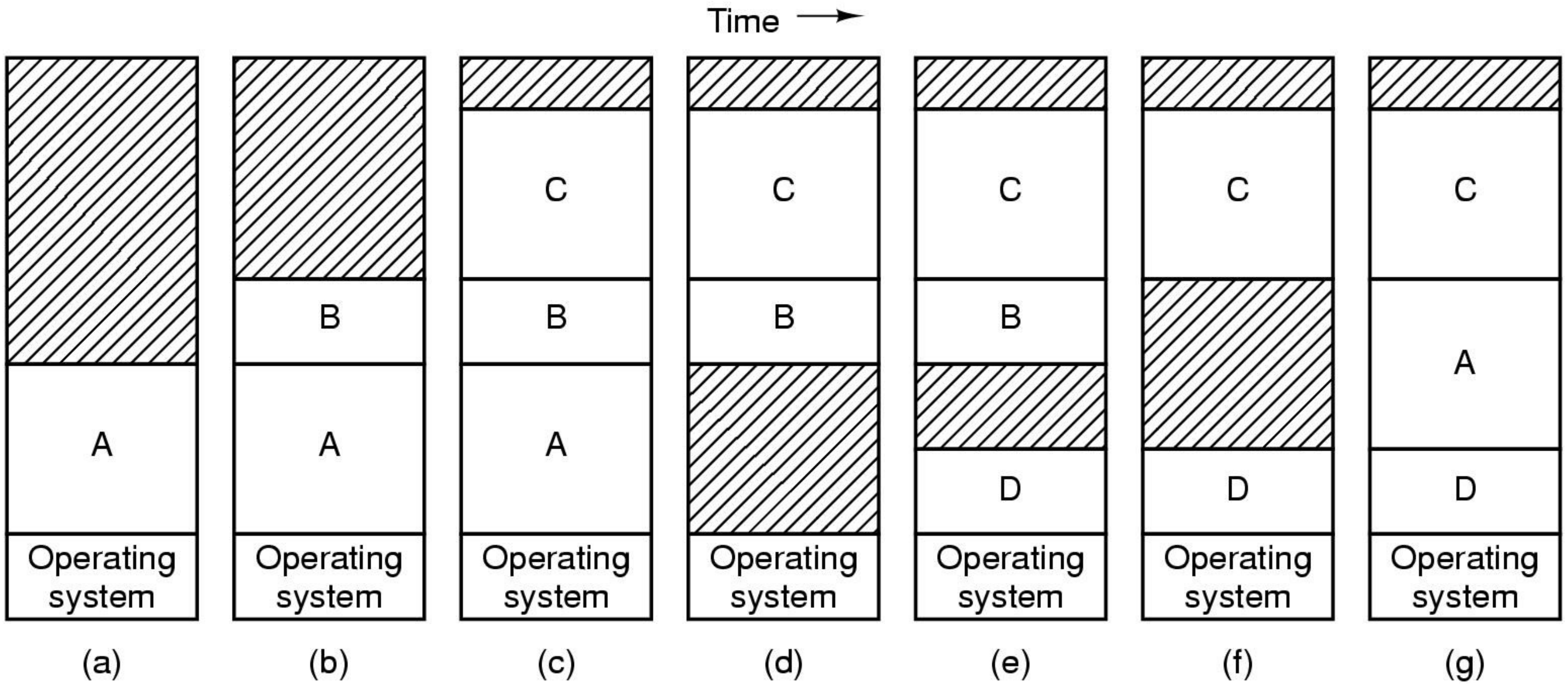


Base and Limit Registers

- Dynamic relocation
- Base and limit registers can be used to give each process a separate address space

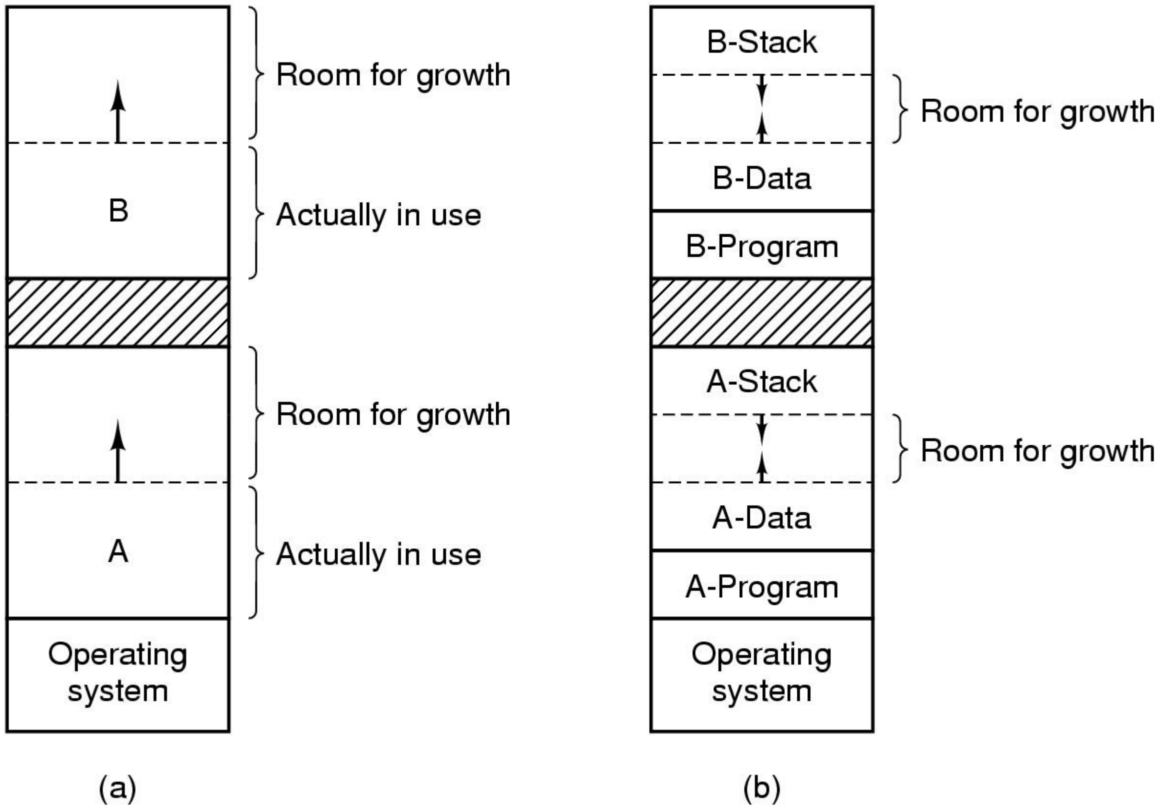


Swapping (1)



Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory.

Swapping (2)



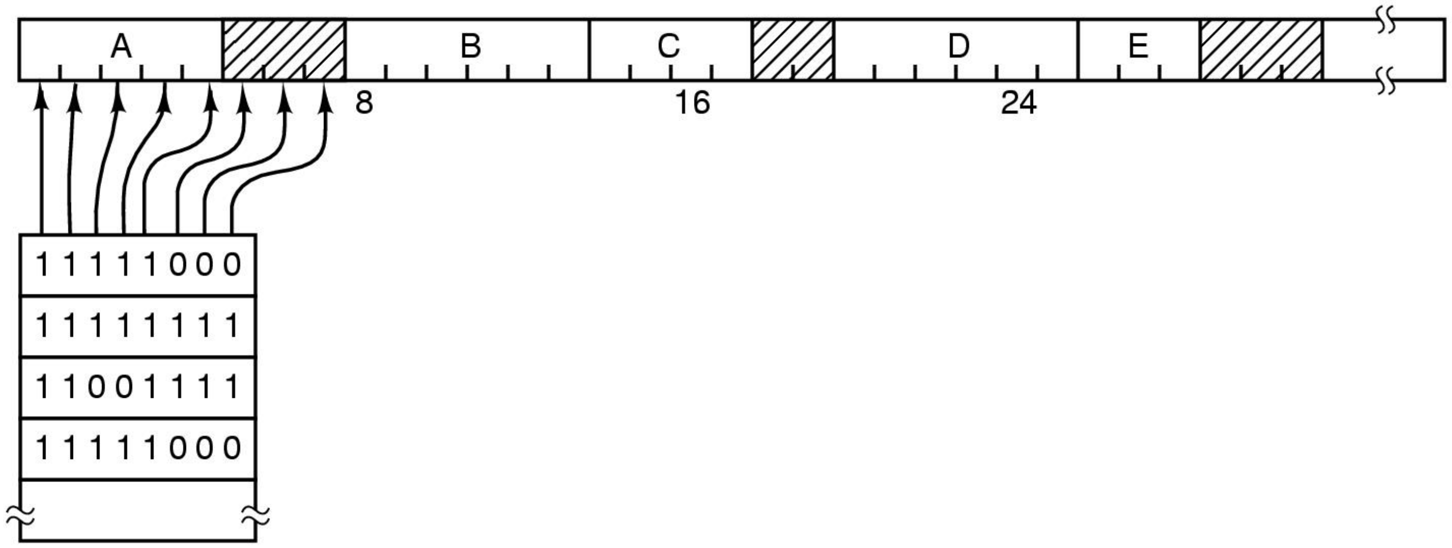
(a) Allocating space for growing data segment. (b) Allocating space for growing stack, growing data segment.

Managing Free Memory

- With dynamic memory management, the OS has to manage the free space, too

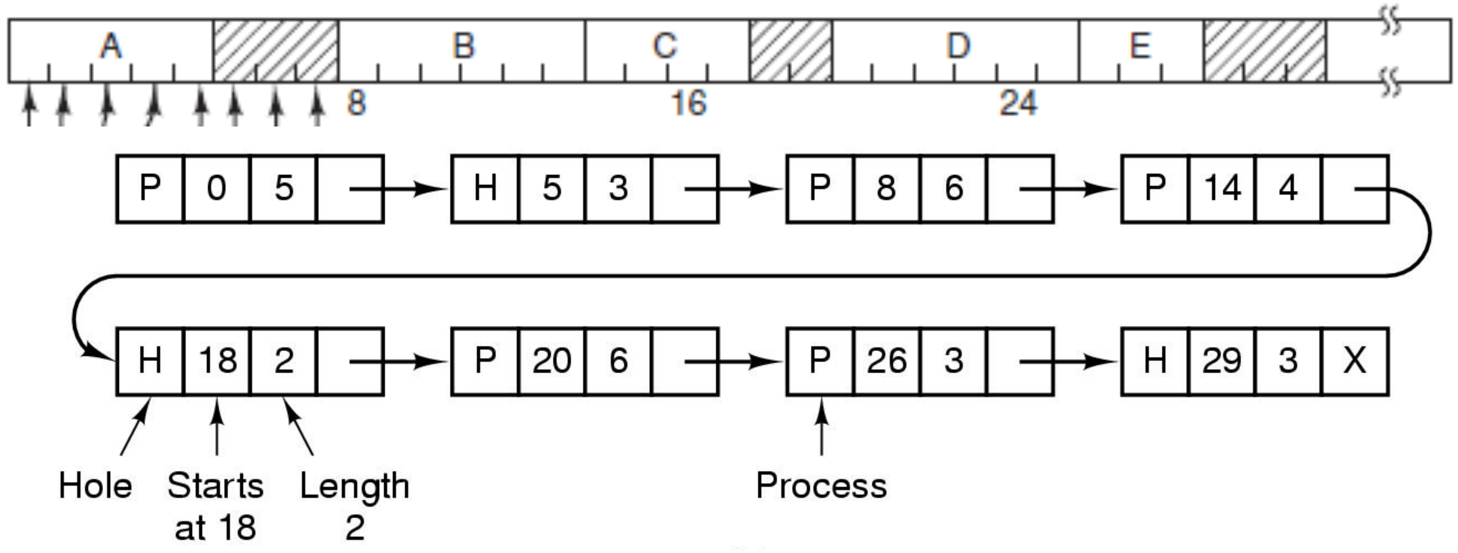
- Two methods:
  - Bitmaps
  - Linked lists

Memory Management with Bitmaps



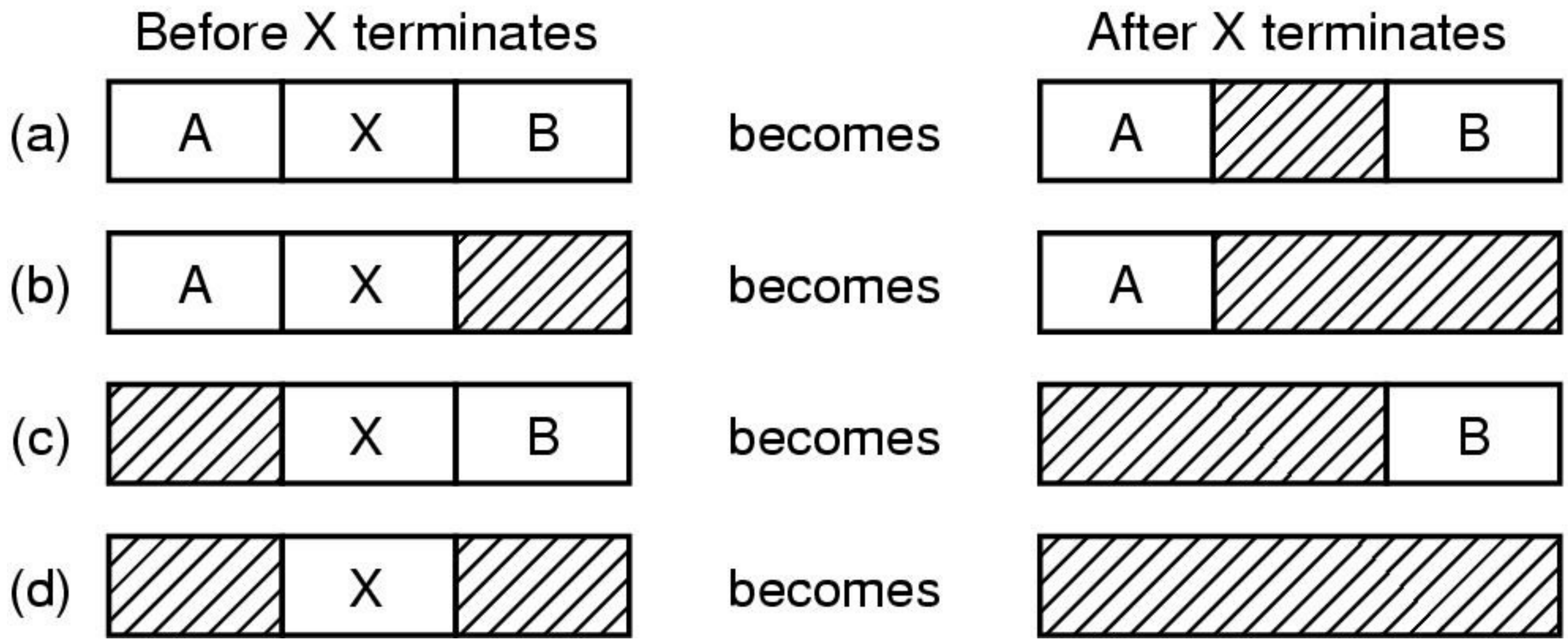
- The memory is divided into allocation units
- For each allocation unit there is a single bit which shows if the unit is free
    - A simple way to keep track of memory words in a fixed amount of memory
    - Problem: bringing a k-unit process into memory requires searching through the bitmap to find k consecutive 0-bits: slow operation

Memory Management with Linked Lists



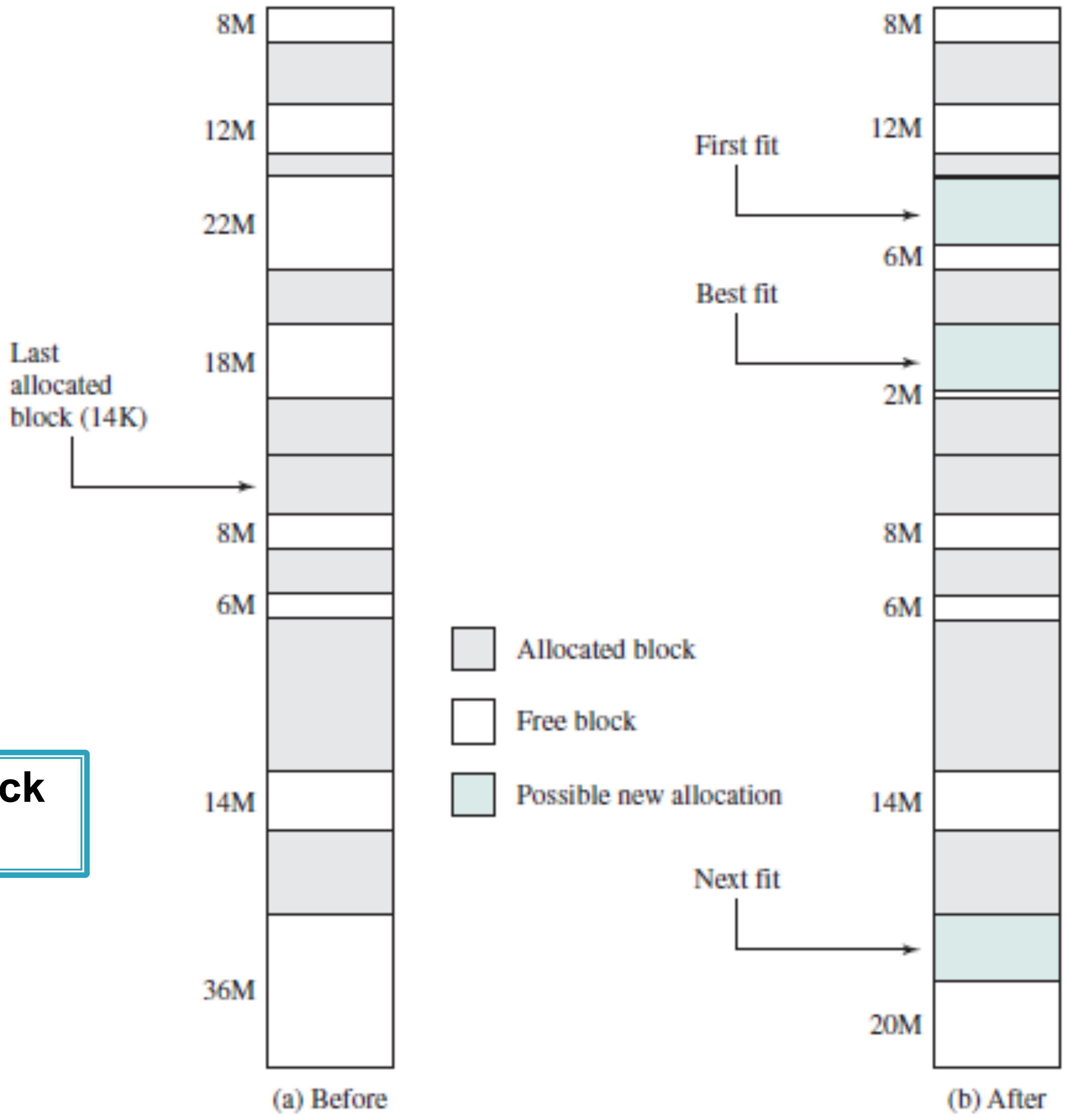
- The memory is divided into allocation units
- For each assigned and free memory block, there is a node which keeps the address at which the block starts, the length, and a pointer to the next item.
- Allocation methods:
  - FIRST FIT, NEXT FIT, BEST FIT, WORST FIT, QUICK FIT

Merging Free Memory Blocks



- Merging free memory blocks if a given block X is freed

Example



Allocation of a block with size 16 MB

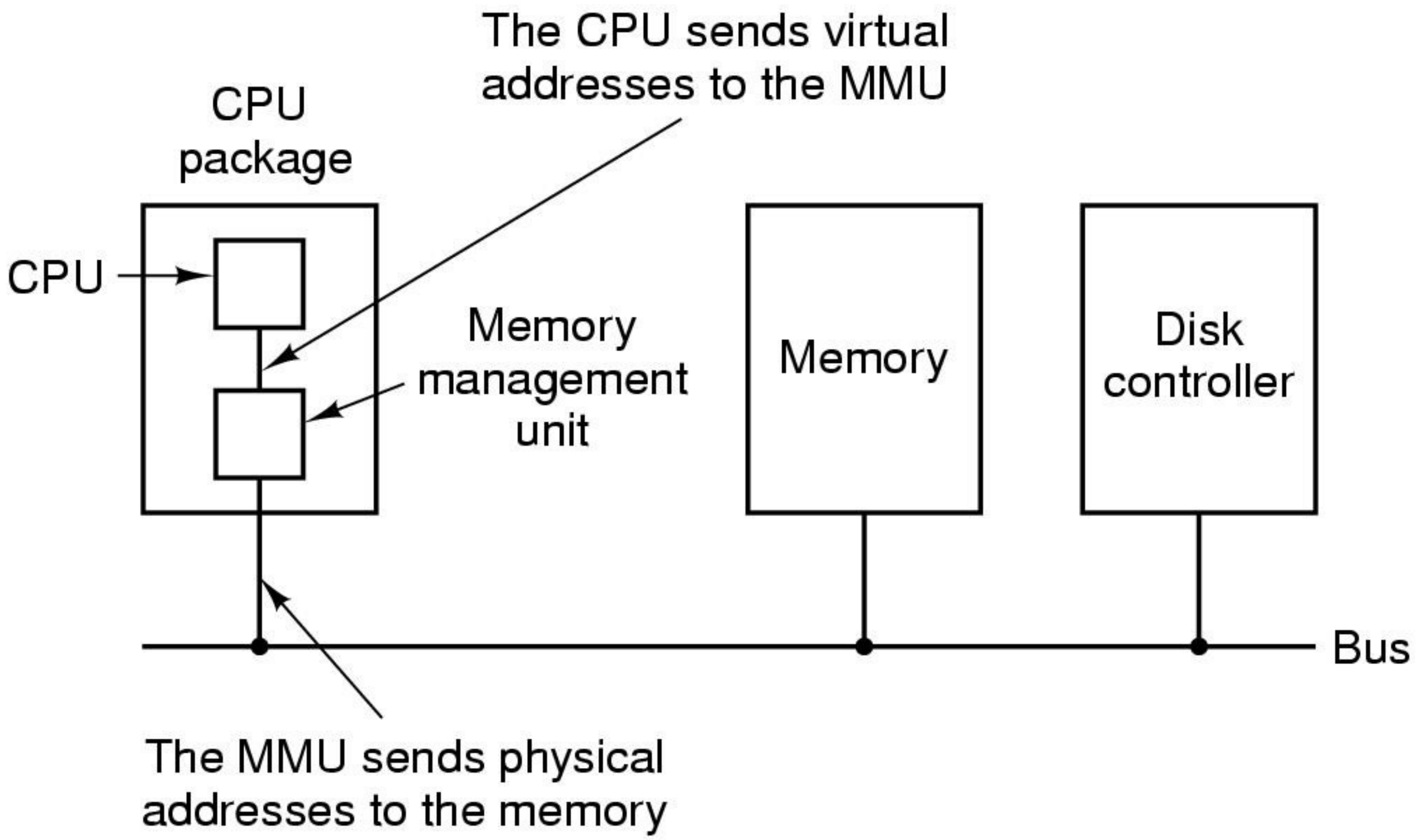
Question

- Base and Limit registers are used in:
  - A. Systems without memory abstraction
  - B. Systems with memory abstraction
  - C. They have nothing to do with memory

Virtual Memory

- It allows extension of the address space above the capacity of the main memory
- Main idea – each program has its own address space which is broken up into chunks called pages
- Pages are mapped in the physical memory
  - Not all pages have to be in physical memory
  - When the program references a part of its address space that is in physical memory, the hardware performs the mapping on-the-fly
  - When the program references a part of its address space that is not in physical memory, the operating system is alerted to go get the missing page

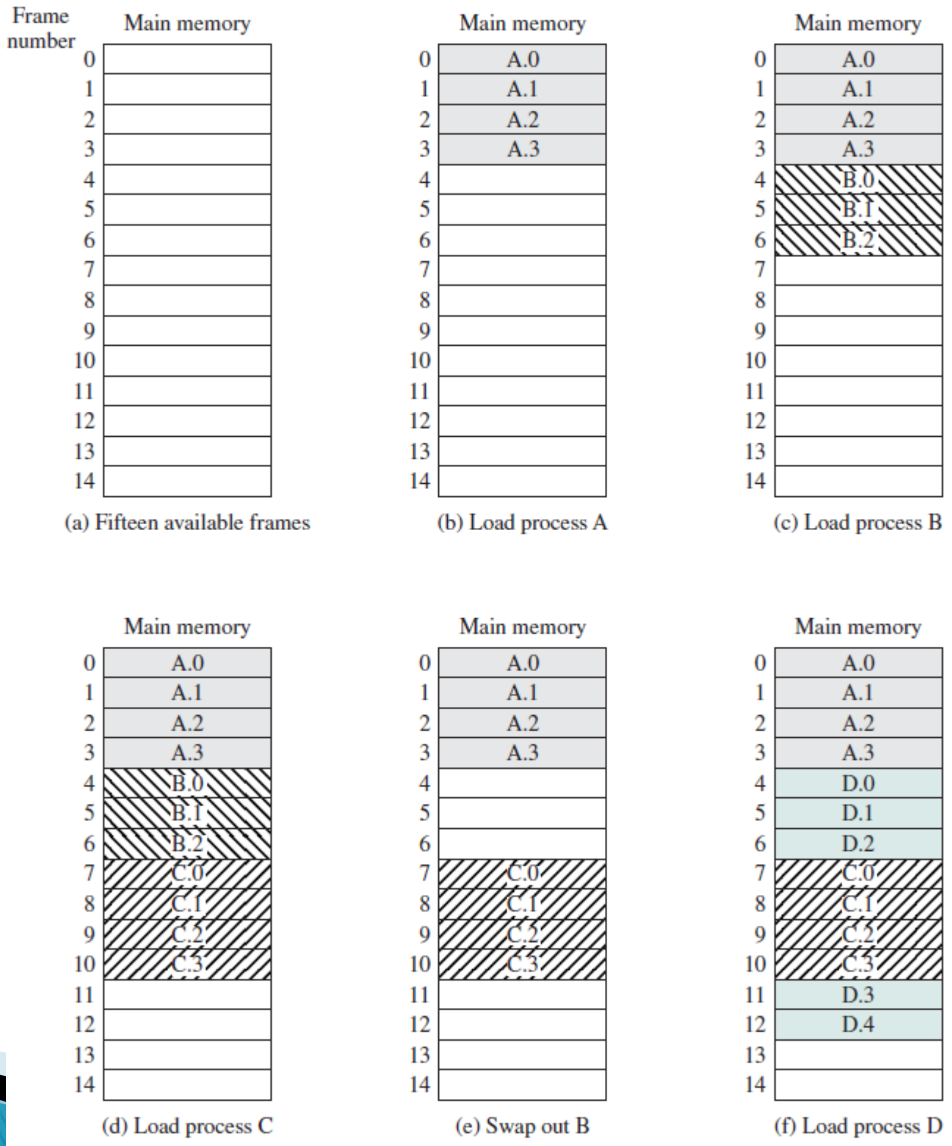
Virtual Memory


The CPU sends virtual addresses to the MMU
CPU package
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

Paging

- Split the memory into small, equally sized units, and split each process in units of the same size
- The process units are called **pages**, and the memory units are called **frames**
- The OS keeps track of a **page table** for each process:
  - It contains the location of the physical memory frame for each page of the process pages
  - Each memory reference contains a **page number** and an **offset** in the same page

Assignment of Process Pages to Available Memory Frames



(a) Fifteen available frames
(b) Load process A
(c) Load process B
(d) Load process C
(e) Swap out B
(f) Load process D

And the Corresponding Page Tables of the Processes

0	0
1	1
2	2
3	3

Process A page table

0	—
1	—
2	—

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

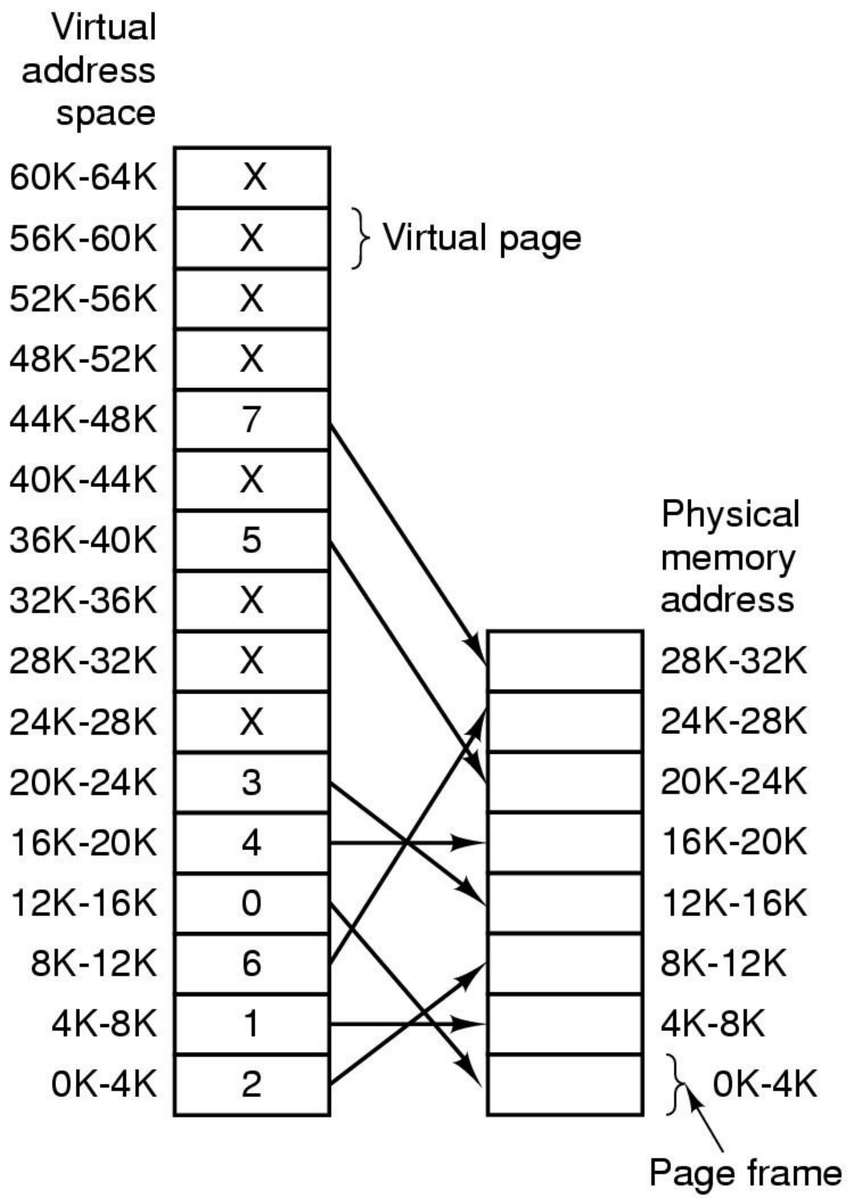
Process D page table

13
14

Free frame list

Paging

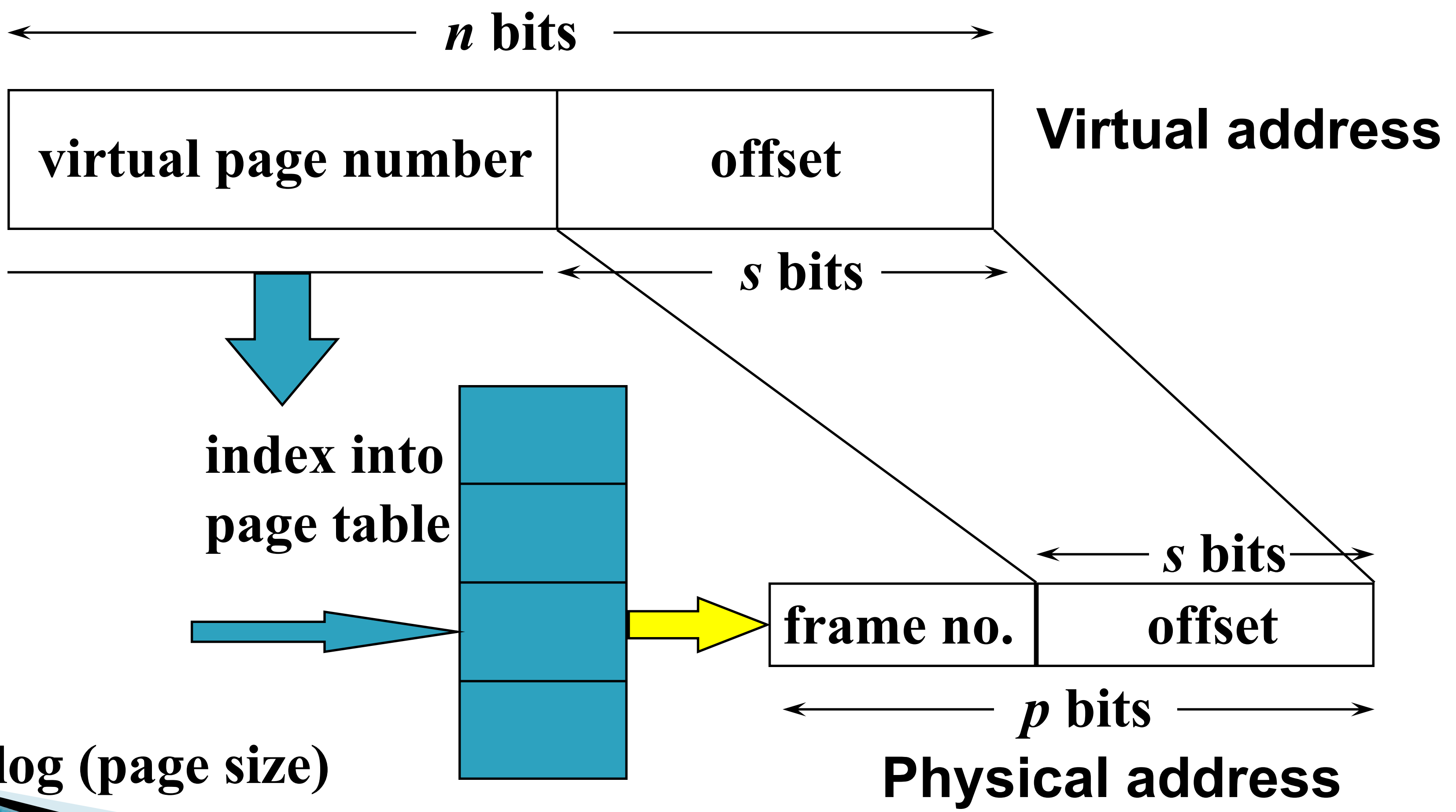
- Use of a **page table** which does the mapping from a virtual to a physical address



Parts of the Virtual Address

- Each address generated by the CPU (virtual address) has two parts:
  - page number – **p**
  - page offset – **s**
- The page number is used as an index in the **page table**
  - The entry contains the **base address** for the page in the physical memory
  - The **base address** is combined with the **page offset** in order to define the physical address which is sent to the memory unit

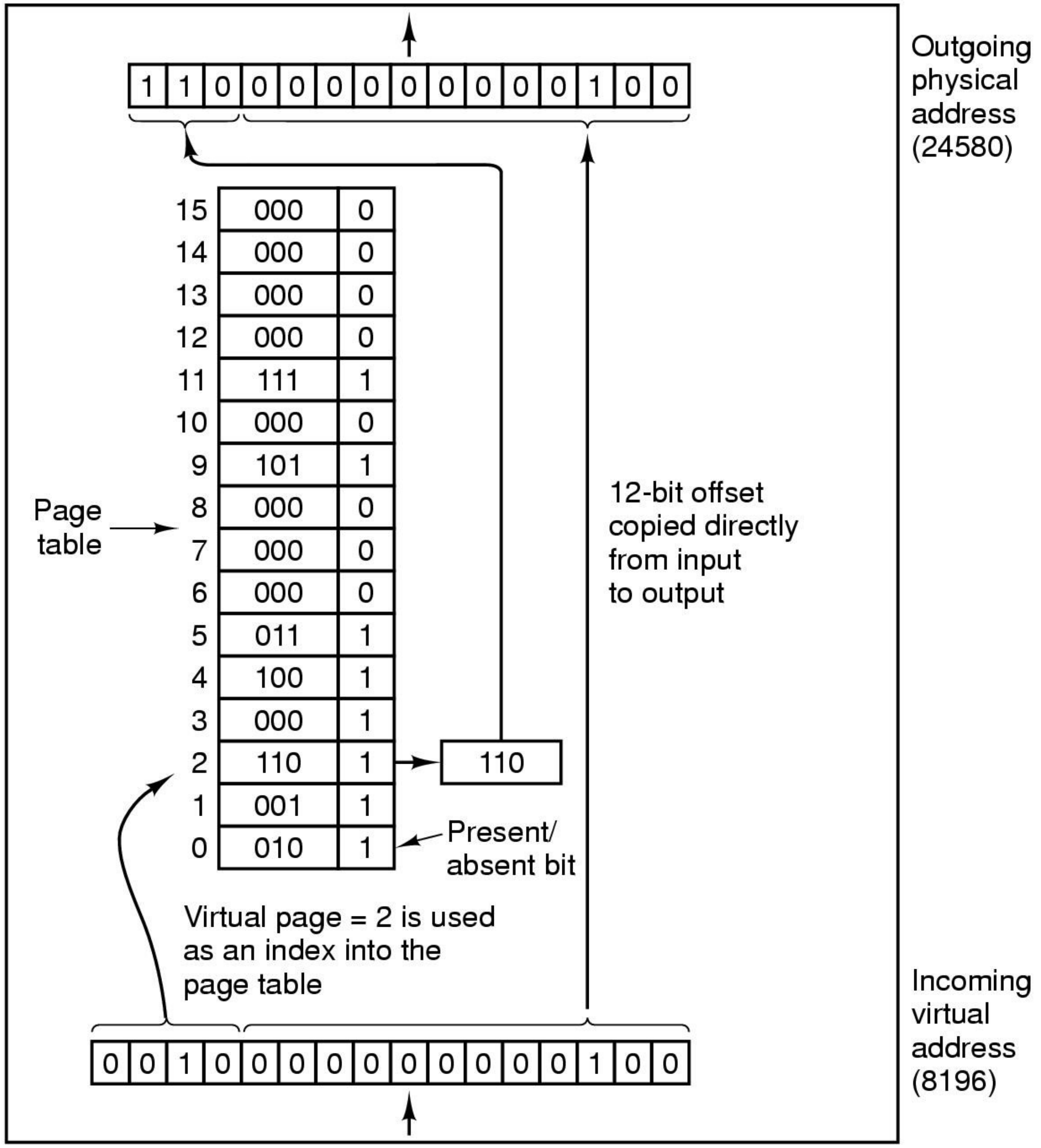
Mapping from VA to PA



s: log (page size)

Page Table

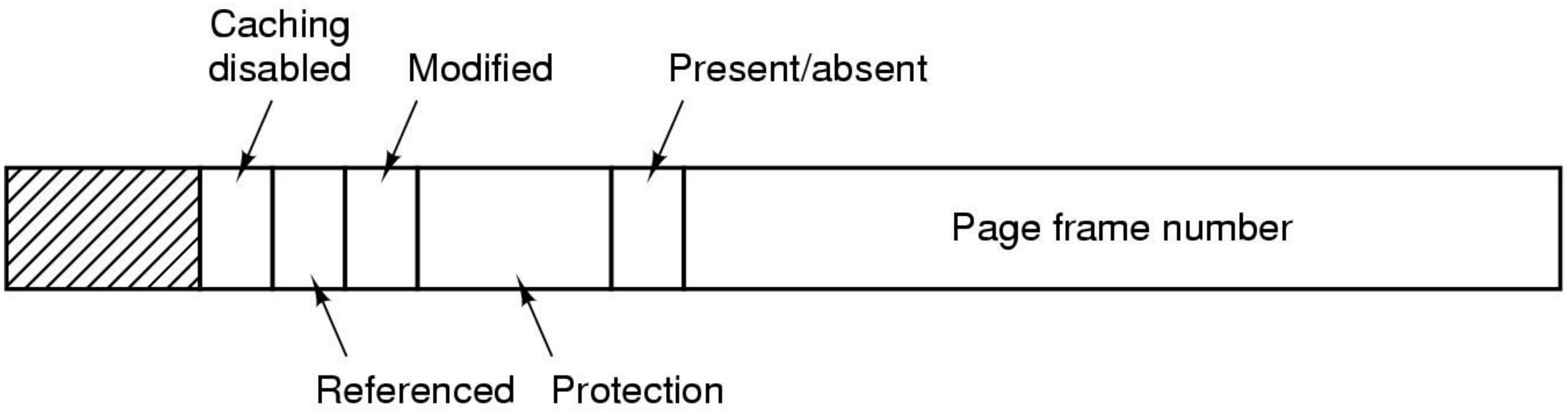
MMU with 16 pages with size of 4 KB



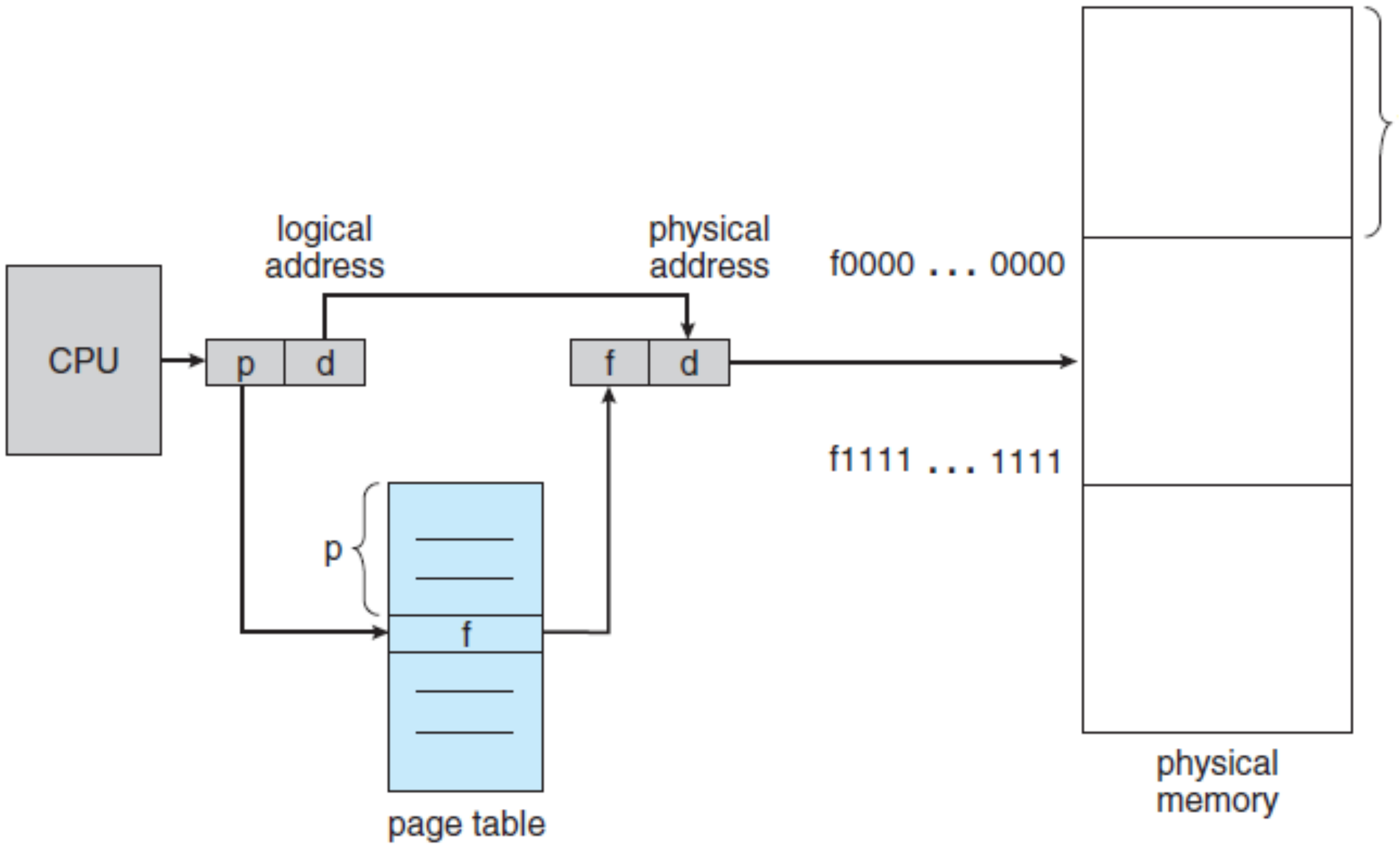
Question

- The physical memory frame and the virtual memory page, have:
  - A. The same size
  - B. Different sizes
  - C. Generally the same size, but they can sometimes be different
  - D. Generally a different size, but they can sometimes be the same

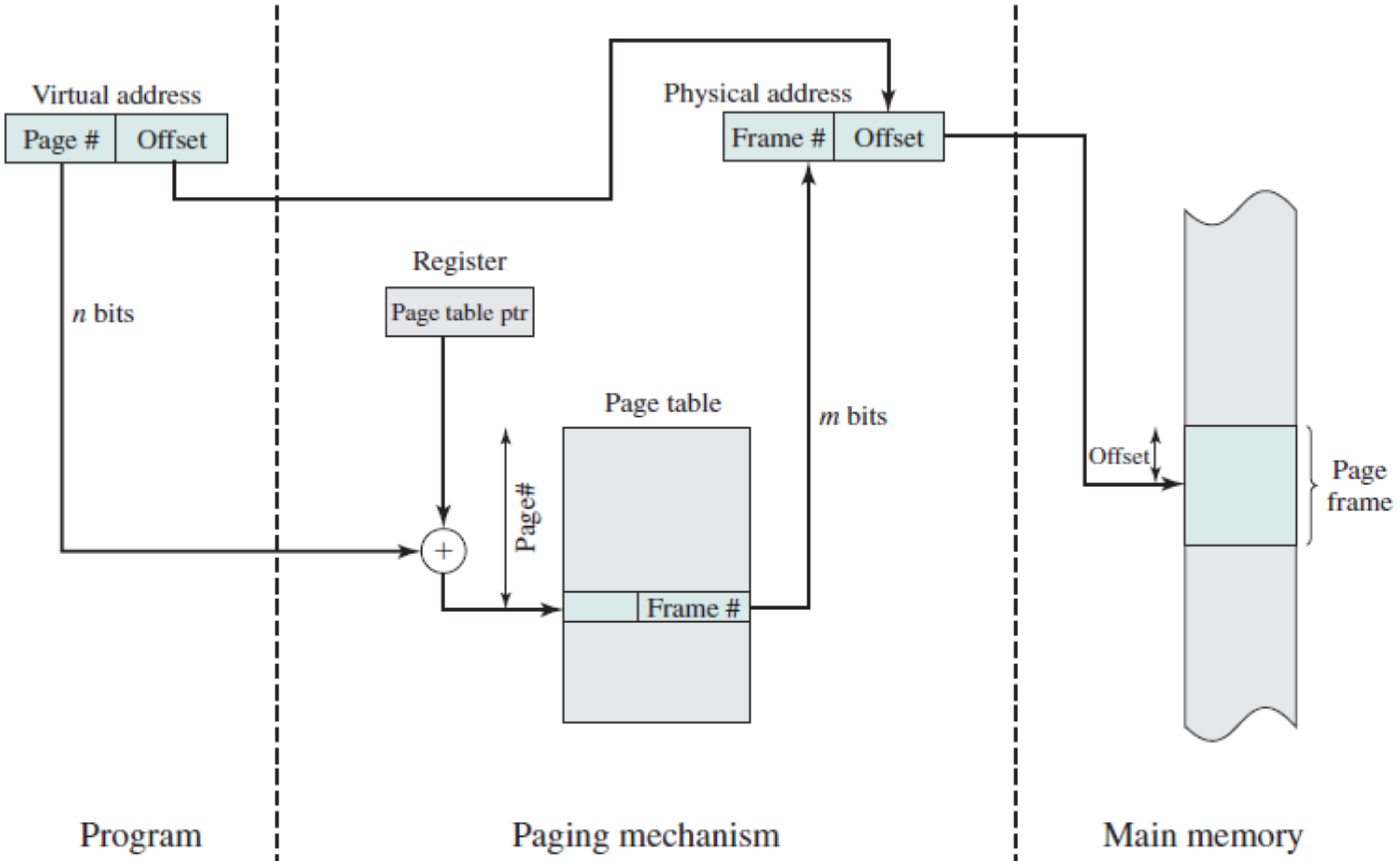
Structure of a Page Table Entry



Page Table in CPU Registers



Page Table in Main Memory



Translation Lookaside Buffer (TLB)

- Problem: each virtual memory reference may require two accesses to physical memory
  - One to take the page table entry
  - One to take the actual data from memory
- To mitigate these performance issues, a fast cache is introduced to store data from the page table
  - Called the TLB (Translation Lookaside Buffer)
  - Eg.: Pentium can store 64 page table entries in the processor;

Translation Lookaside Buffer (TLB)

- TLB is a hardware device for mapping virtual addresses to physical addresses without going through the page table
- TLB is a concept for caching the frequently used rows in the page table
- The size is between 32 and 1024 rows of the page table
- Usually, it’s inside the MMU

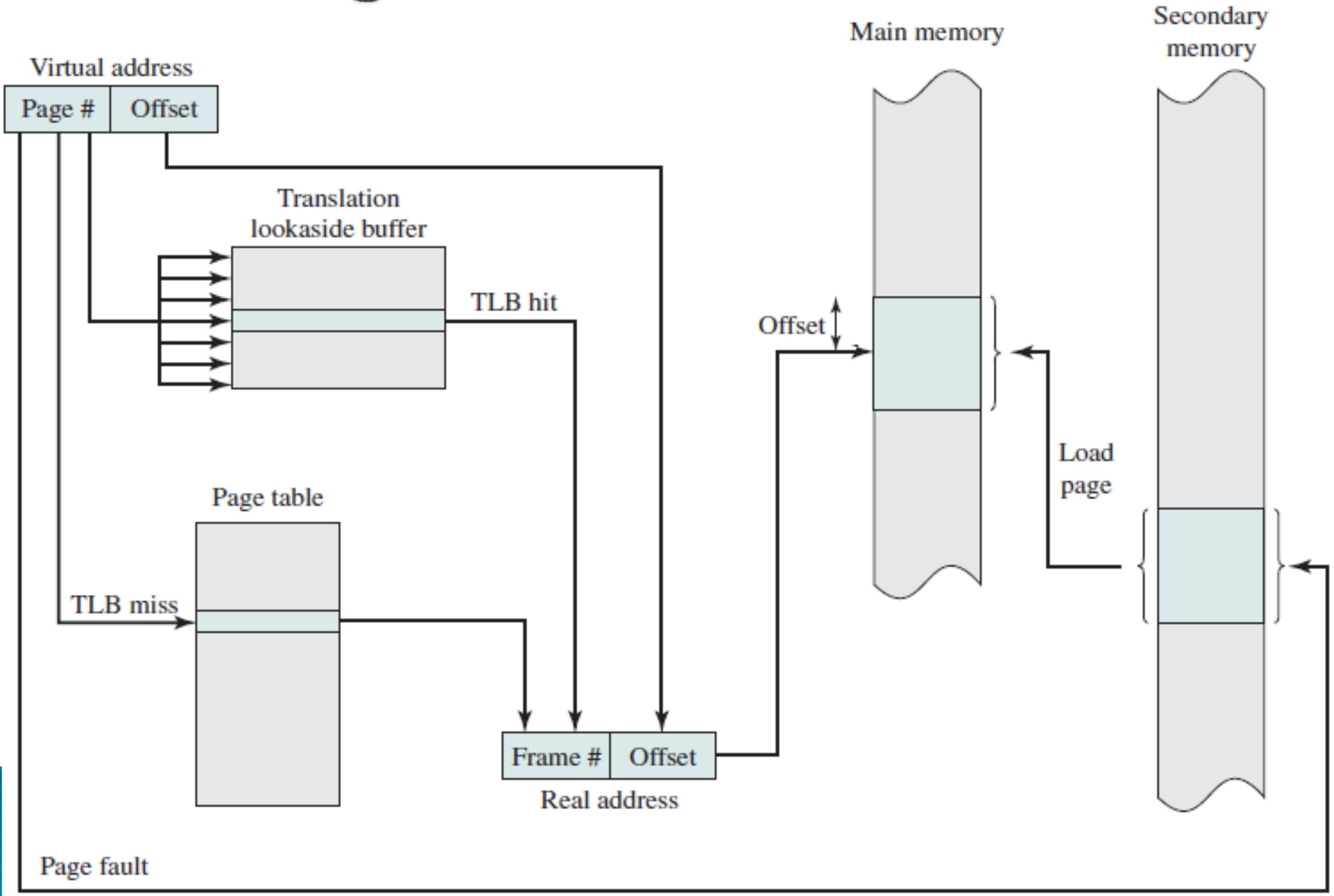
Example Content of TLB

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

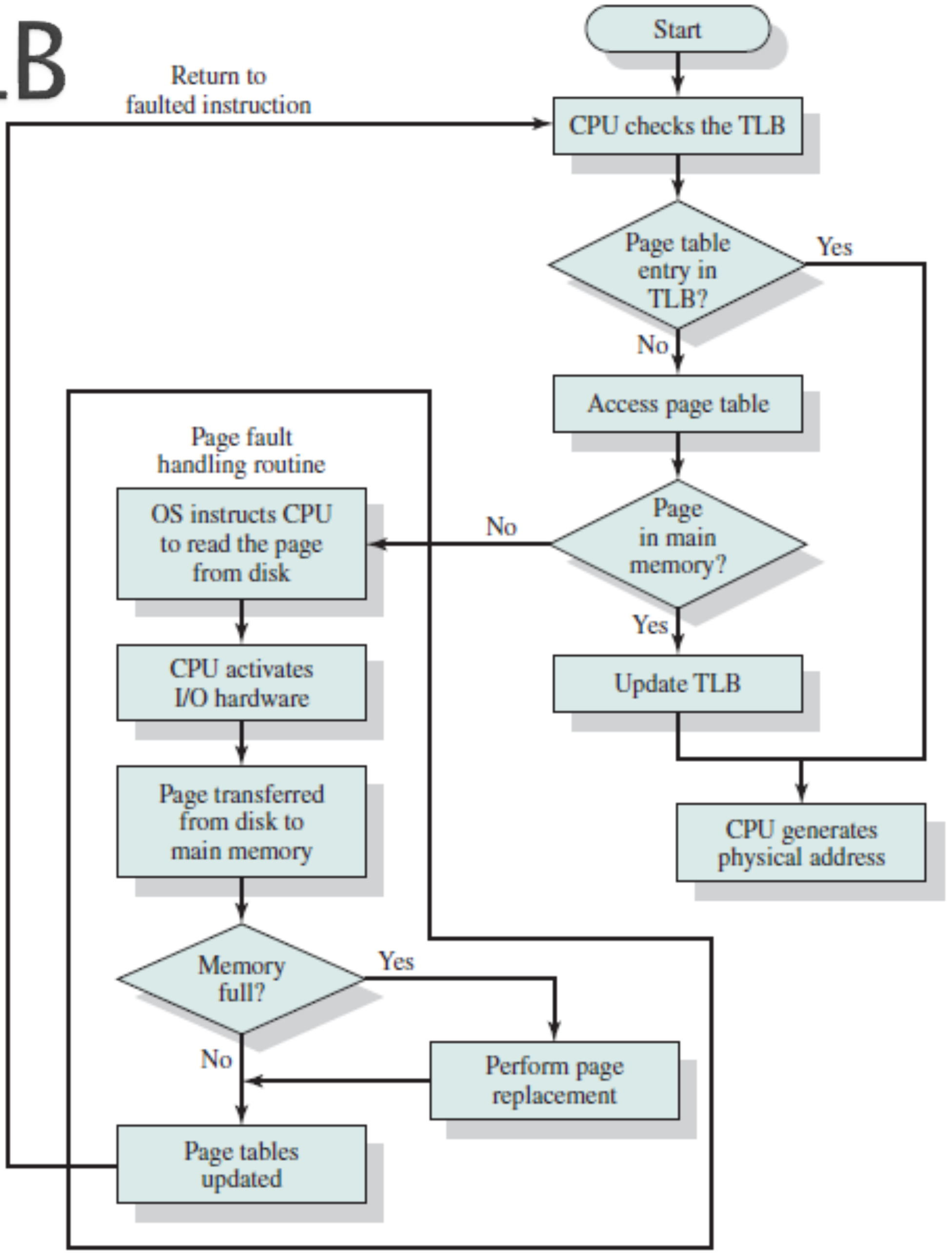
Translation Lookaside Buffer (TLB)

- When a virtual address is presented to the MMU for translation, the hardware first checks to see if its virtual page number is present in the TLB by comparing it to all the entries simultaneously
- If a valid match is found (hit), the frame number is taken directly from the fast TLB and the corresponding physical address is formed
- In the case when a valid match is not found (miss), the page number is used as an index to the page table, and the TLB is updated with the new entry

TLB Usage


Virtual address
Page #
Offset
Translation lookaside buffer
TLB hit
TLB miss
Page table
Frame #
Offset
Real address
Main memory
Secondary memory
Offset
Load page
Page fault

Paging and TLB


Start
CPU checks the TLB
Return to faulted instruction
Page table entry in TLB?
Yes
No
Access page table
Page in main memory?
No
Yes
Update TLB
CPU generates physical address
Page fault handling routine
OS instructs CPU to read the page from disk
CPU activates I/O hardware
Page transferred from disk to main memory
Memory full?
Yes
No
Perform page replacement
Page tables updated

Performance

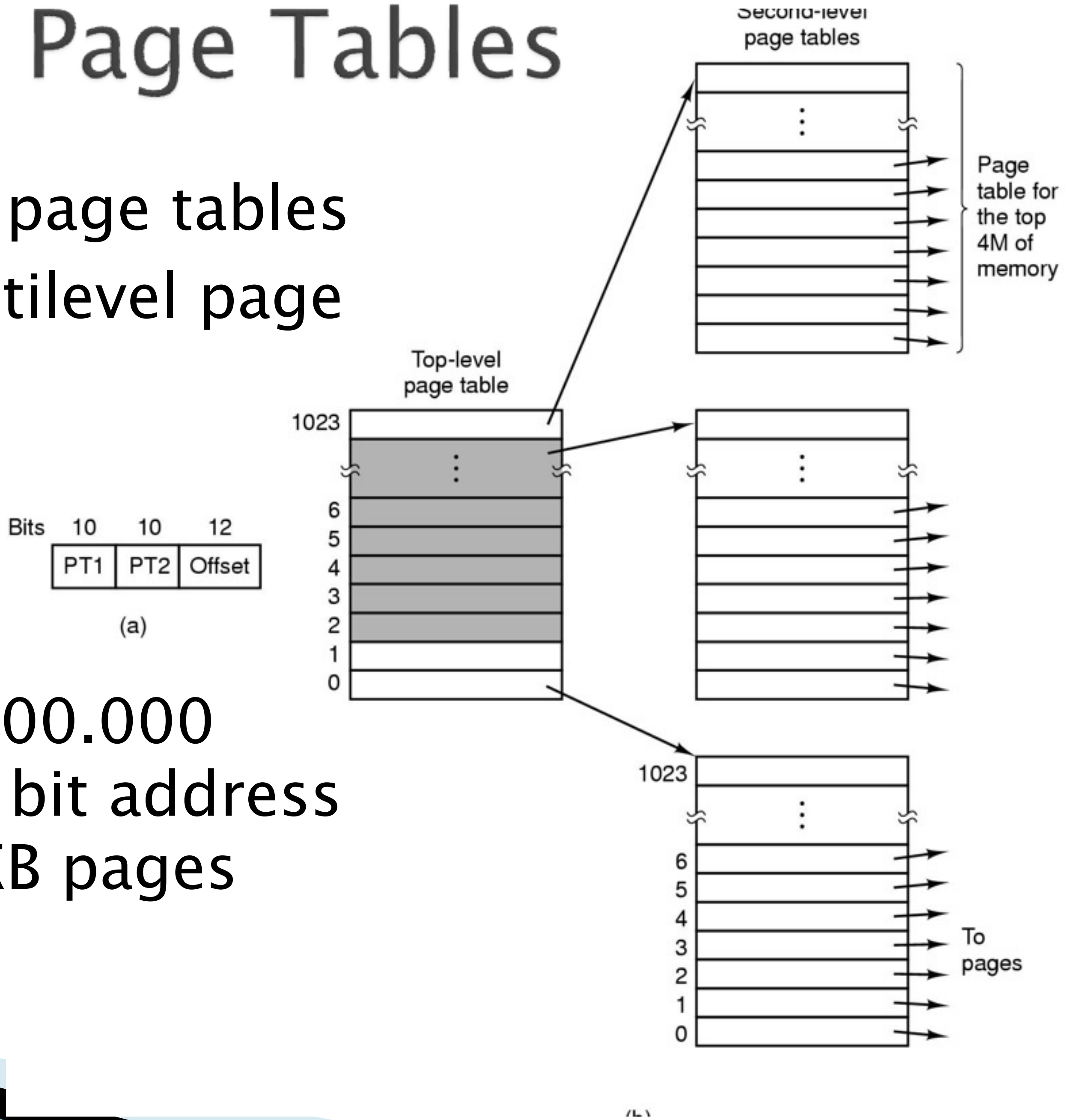
- The part of memory references that can be satisfied with the use of associative memory (TLB), is called the hit rate
- The better the hit rate, the higher the performances of the system

- Example:
  - 100 nsec access time for the page table
  - 20 nsec access time for TLB
  - 90% hit rate

$0.9 * 20 + 0.1 * 100 = 28$ nsec average access time

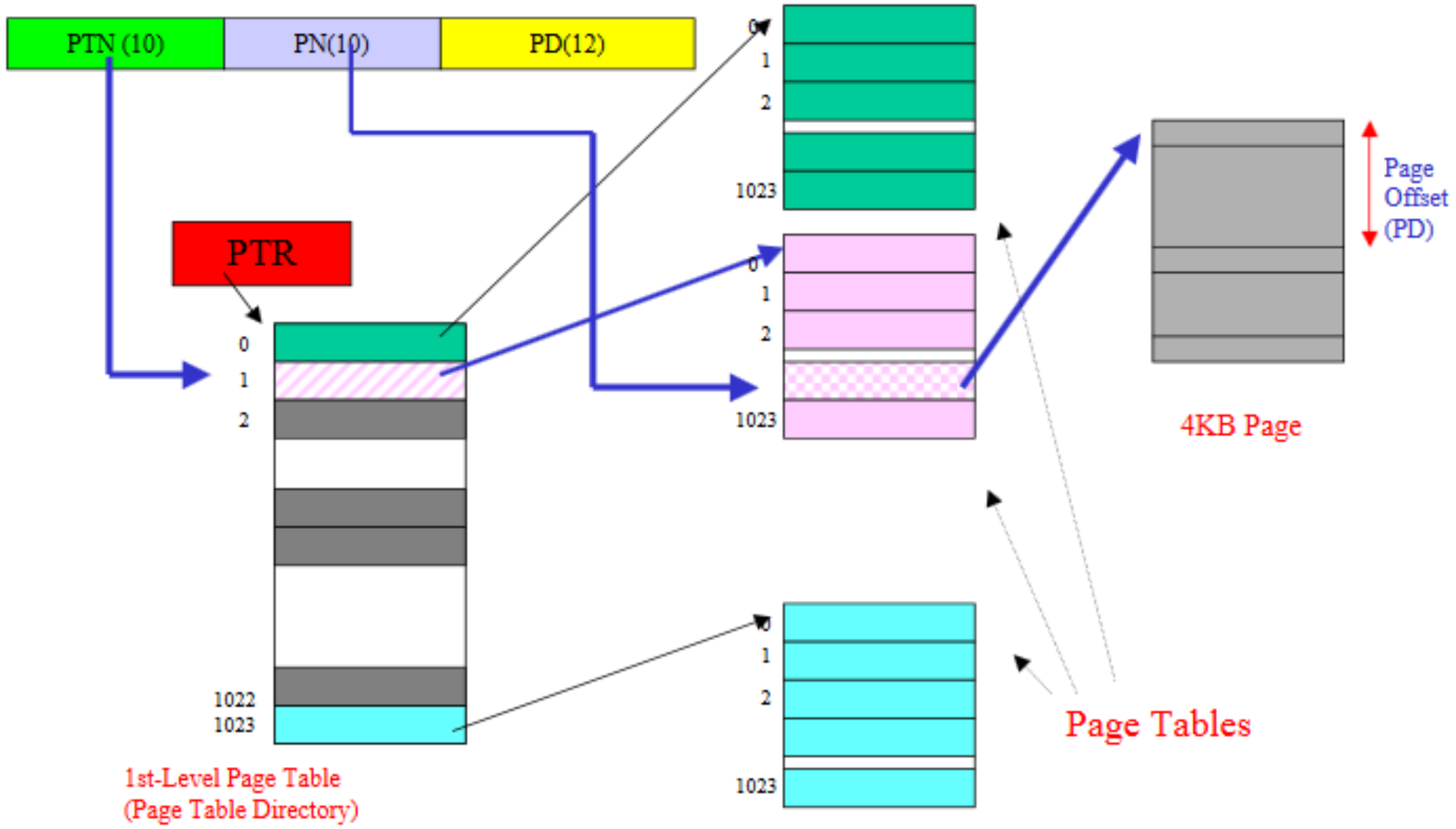
Two–level Page Tables

- Problem: Big page tables
- Solution: Multilevel page tables



- Example: 1.000.000 pages for 32 bit address space and 4KB pages

Two-level Page Tables



Inverted Page Tables

Problem:

- Page tables depend on the size of the virtual memory
- The virtual address space becomes bigger and bigger:
  - A 64-bit system, with a page size of 4KB will have **2^{52} entries** in the page table (4 quadrillions)!

Inverted Page Tables

Solution:

- Page tables should be organized with a focus on the physical memory, not the virtual memory
- A global page table indexed by the frame number, where each entry has a virtual address and a PID
- The ith entry contains information about the virtual page which is located in the ith frame of the physical memory
- Uses TLB to reduce the need to access the page table
- If there is a TLB miss: look through the table for <virtual address, PID>
  - The physical address is obtained using the table index (frame number)

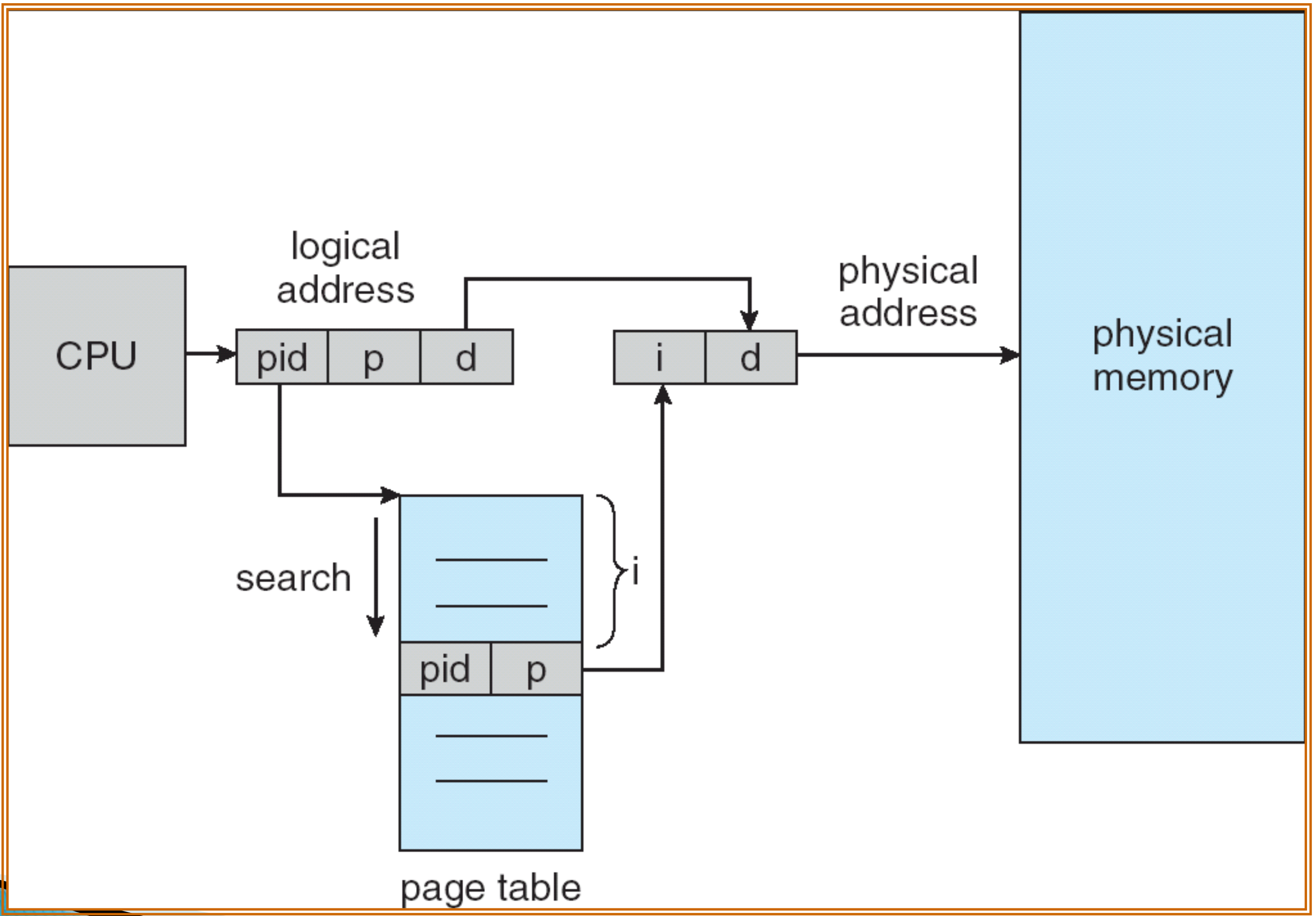
IPT Structure

- Indexed with the frame number
- The entry contains virtual addresses and PID which are using the frame (if there are any!)
- It also contains the bits detailed in this slide

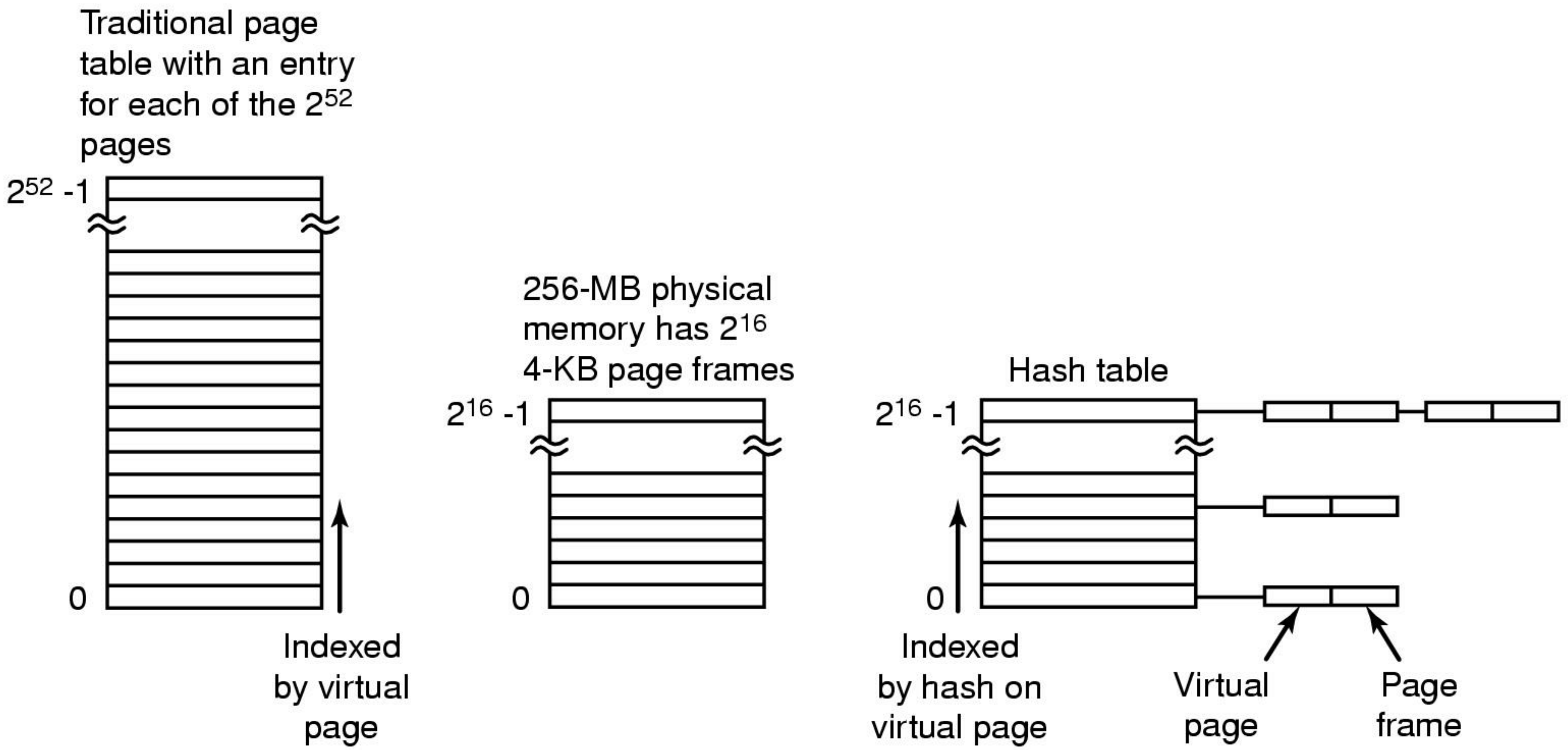
pid	Virtual addr	v	w	r	x	f	m
pid	Virtual addr	v	w	r	x	f	m
pid	Virtual addr	v	w	r	x	f	m

v: valid bit (0 - page fault)
w: write bit
r: read bit
x: execute bit (rare)
f: reference bit (for PR algor.)
m: modified bit (if 1 write to disk)

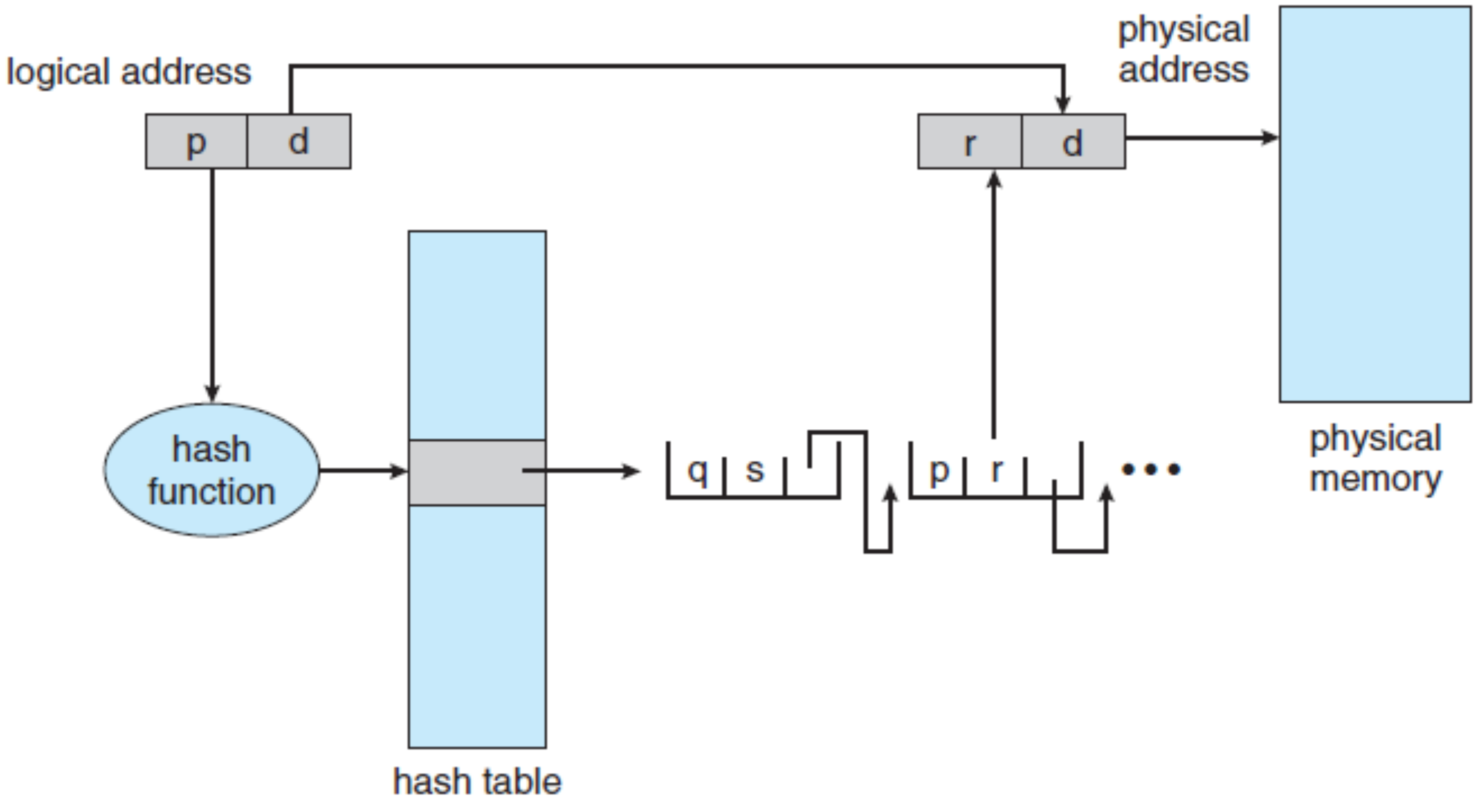
Mapping from a Virtual to a Physical Address



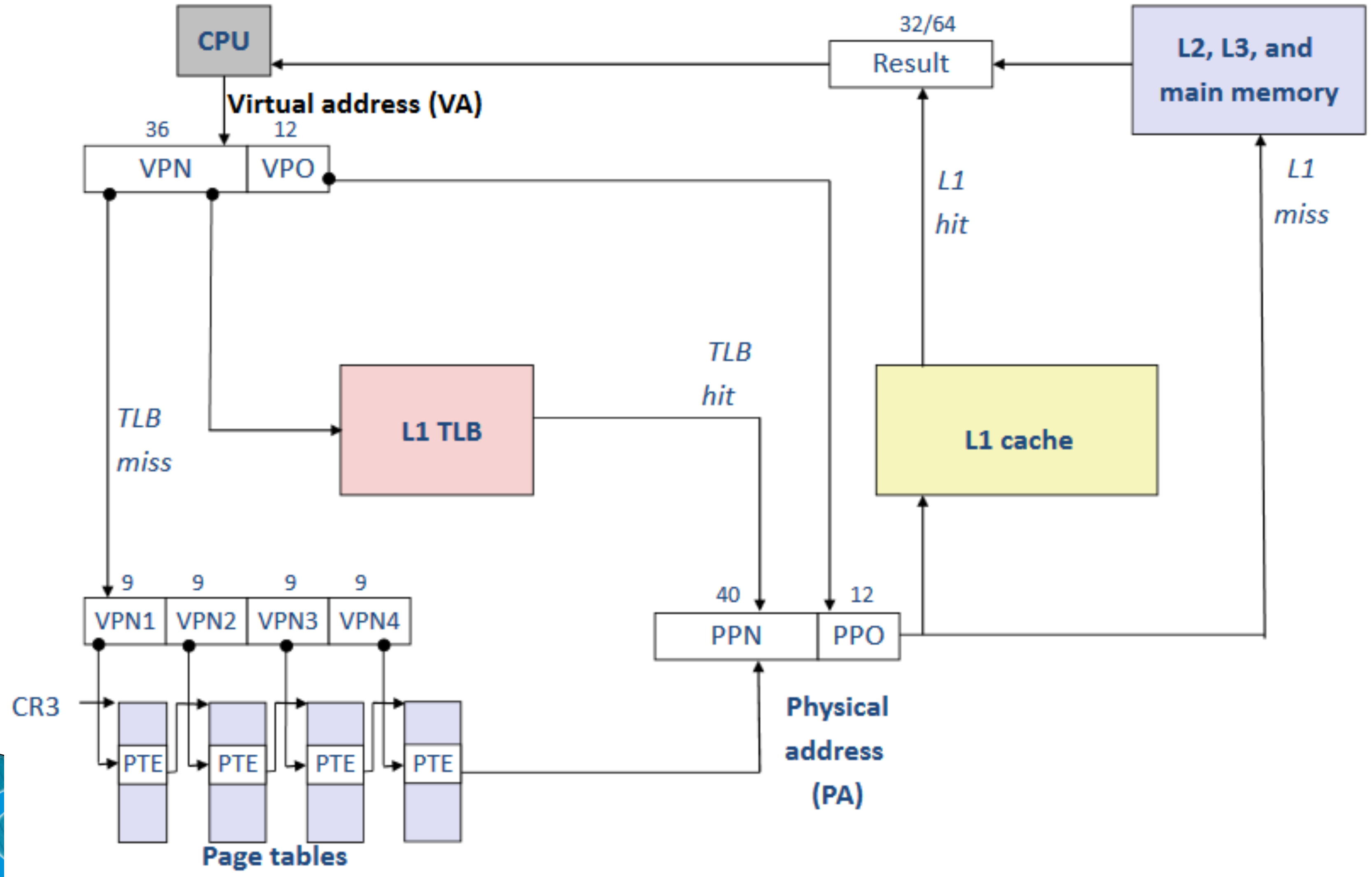
Inverted Page Tables


Traditional page table with an entry for each of the 2^{52} pages
2^{52} -1
0
Indexed by virtual page
256-MB physical memory has 2^{16} 4-KB page frames
2^{16} -1
0
Hash table
2^{16} -1
0
Indexed by hash on virtual page
Virtual page
Page frame

HASH Tables


logical address
p
d
physical address
r
d
hash function
q
s
p
r
•••
physical memory
hash table

End-to-end Core i7 Address Translation



Question

- An entry in a Page Table contains:
  - A. Physical frame number + Process ID
  - B. Virtual page number + Process ID
  - C. Physical frame number
  - D. Virtual page number + Physical frame number

Question

- An entry in an Inverted Page Table contains:
  - A. Physical frame number + Process ID
  - B. Virtual page number + Process ID
  - C. Physical frame number
  - D. Virtual page number + Physical frame number

Question

- An entry in a Hashed Table, hashed on the virtual address, contains:
  - A. Physical frame number + Process ID
  - B. Virtual page number + Process ID
  - C. Physical frame number
  - D. Virtual page number + Physical frame number

Questions?