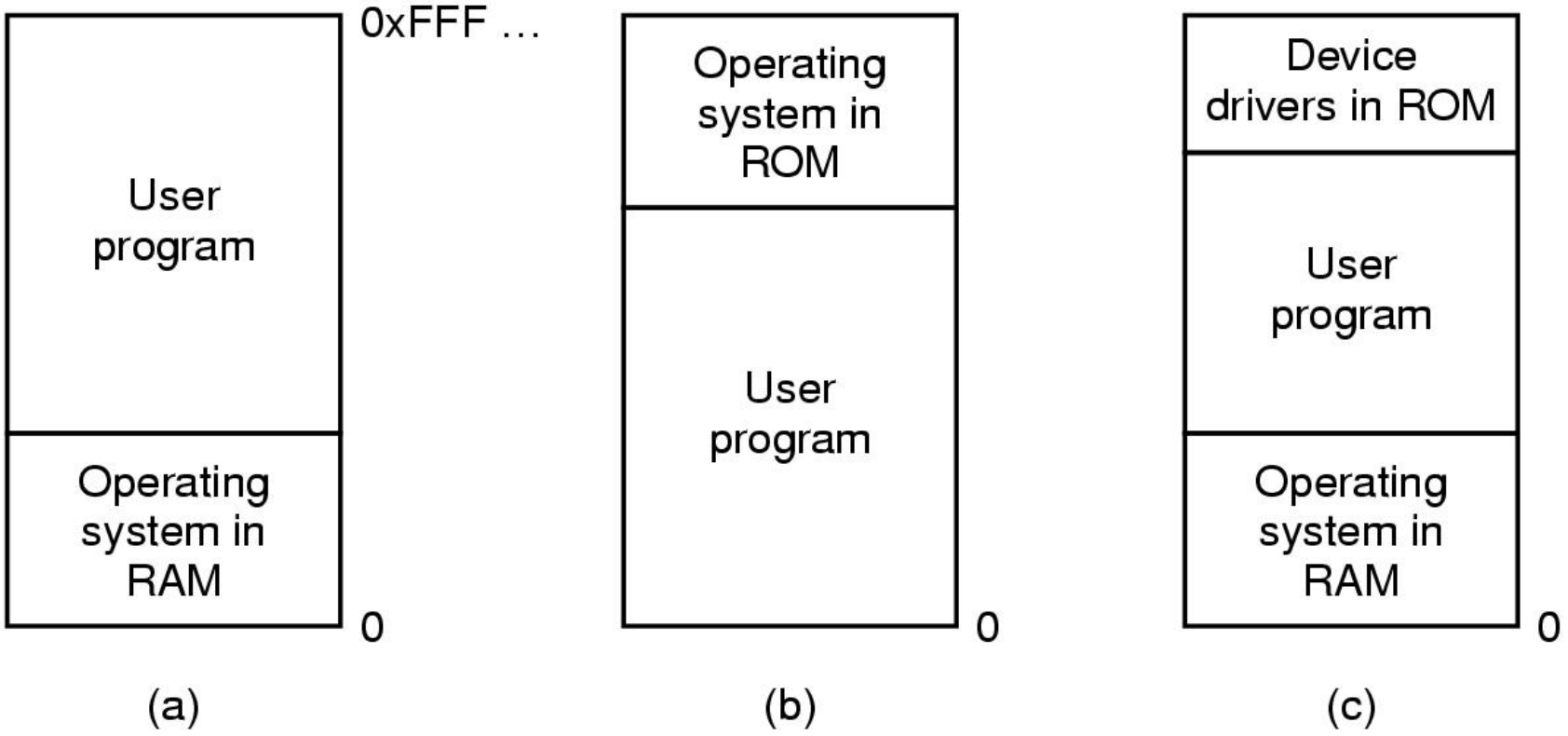
Operating Systems

Jinghui Zhong （钟竞辉）
Office： B3-515
Email： jinghuizhong@scut.edu.cn

Memory Management

- Ideally programmers want memory that is
  - ✓large
  - ✓fast
  - ✓non volatile
- Memory hierarchy
  - ✓small amount of fast, expensive memory – cache
  - ✓some medium-speed, medium price main memory
  - ✓gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

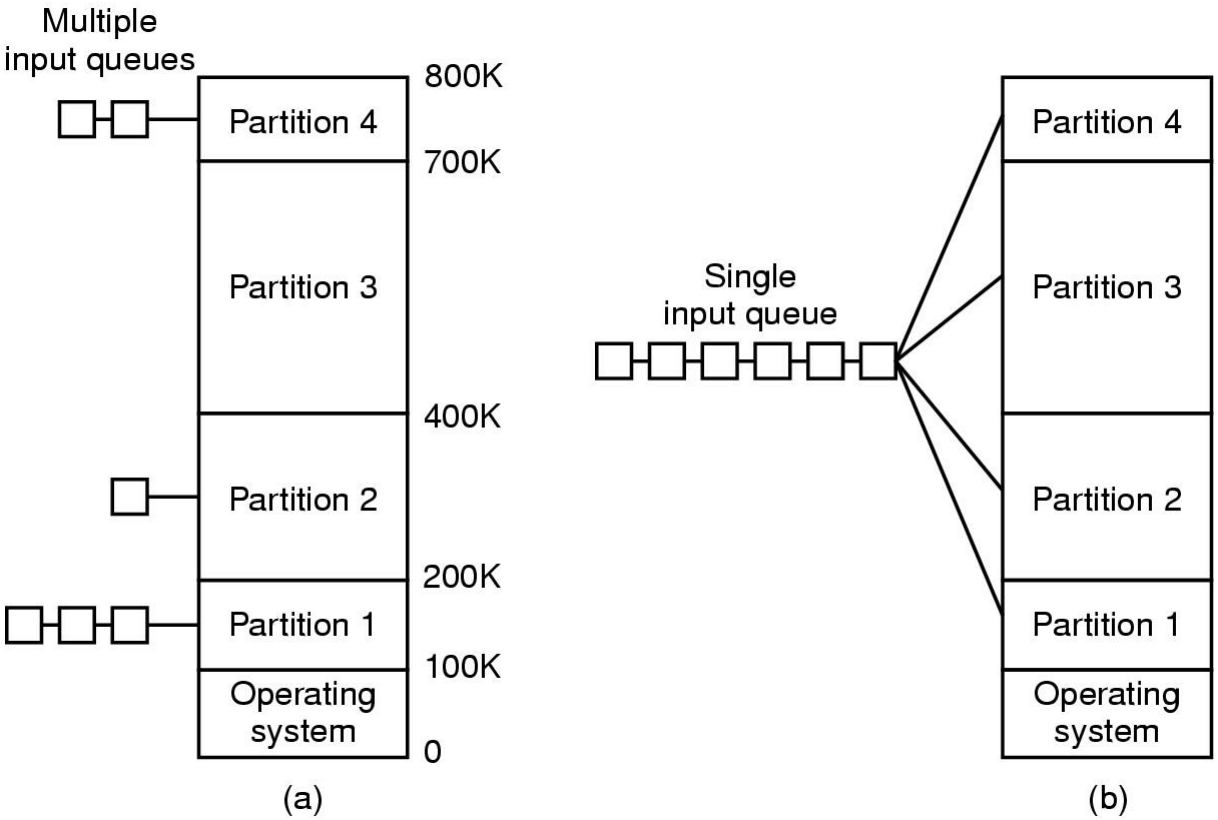
Basic Memory Management



Three simple ways of organizing memory

- an operating system with one user process

Multiprogramming with Fixed Partitions

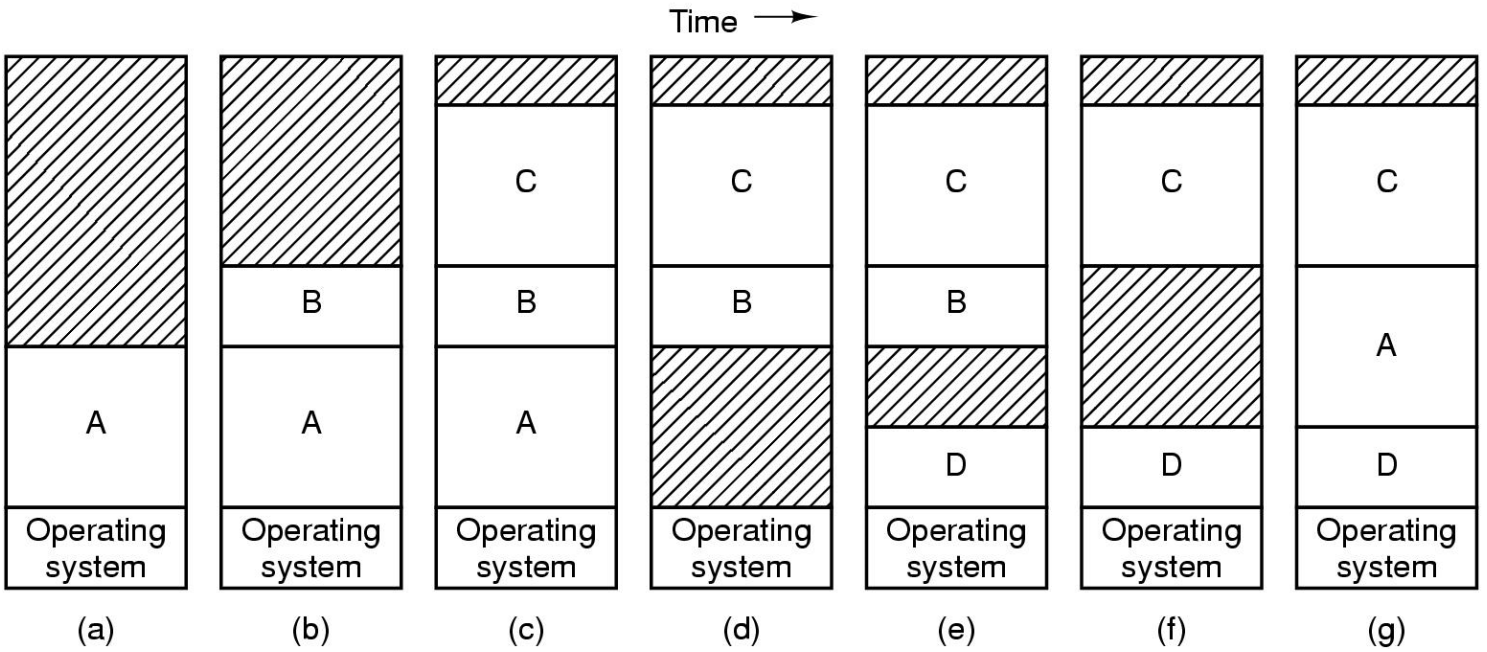


- Separate input queues for each partition
- Single input queue
- As the partition sizes are fixed, any space not used by a particular job is lost.
- It may not be easy to state how big a partition a particular job needs.

Swapping & Virtual Memory

- Two approaches to overcome the limitation of memory:
  - ✓**Swapping** puts a process back and forth in memory and on the disk.
  - ✓**Virtual memory** allows programs to run even when they are only partially in main memory.

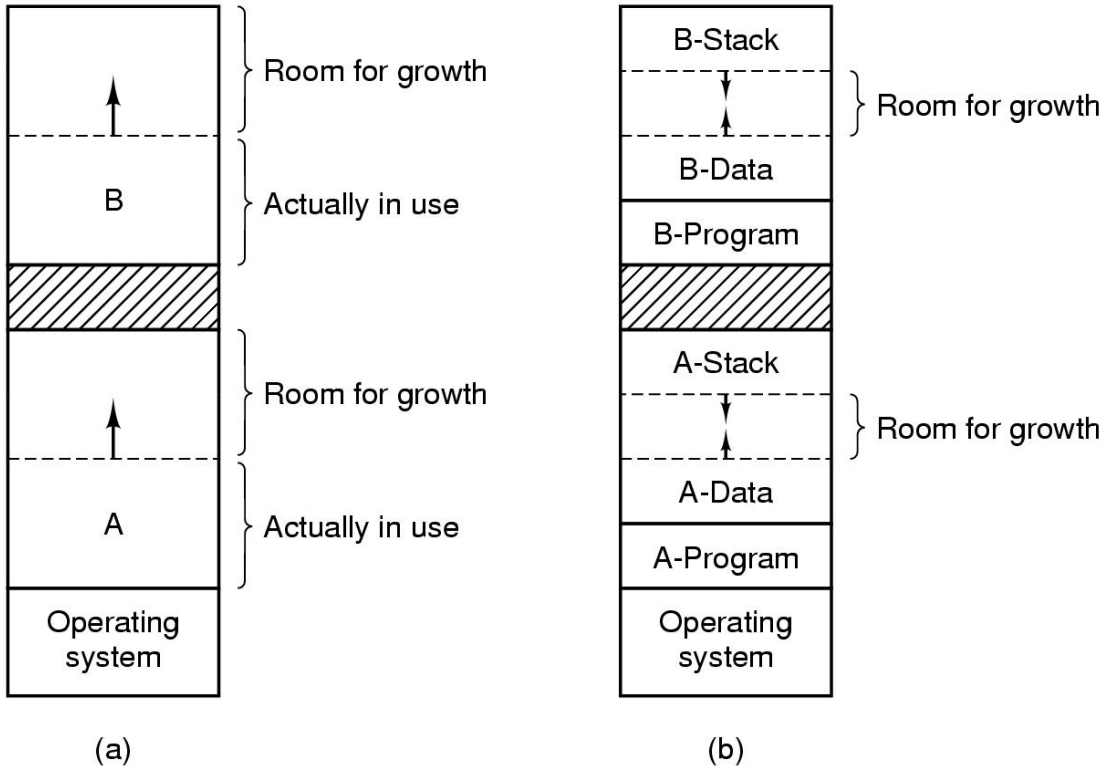
Swapping



Memory allocation changes as processes come into memory and leave memory.

- When swapping creates multiple holes in memory, **memory compaction** can be used to combine them into a big one by moving all processes together.

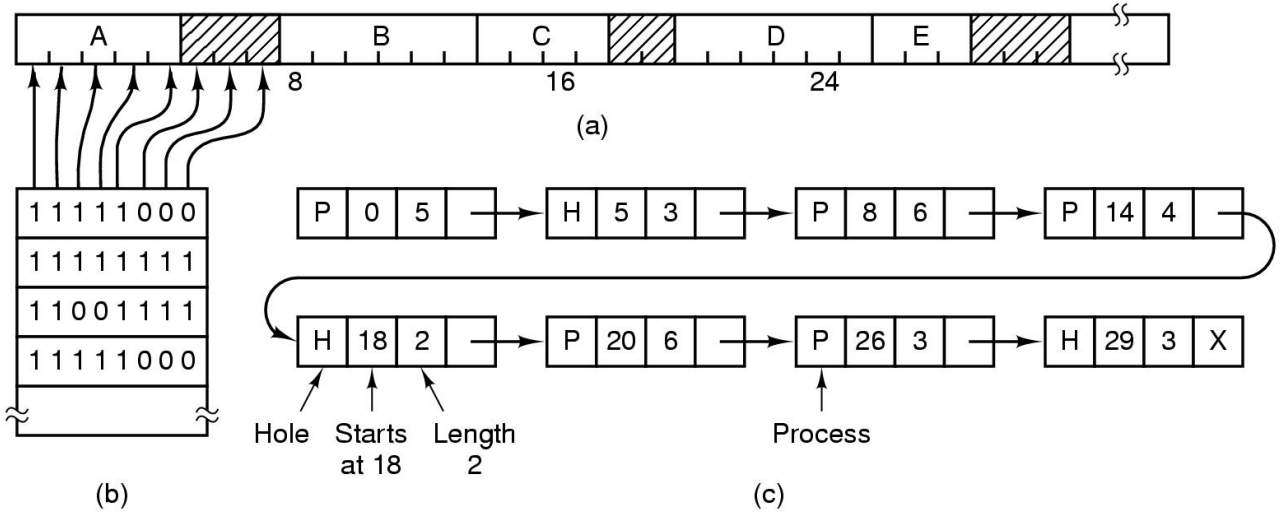
Swapping



- Allocating space for growing data segment
- Allocating space for growing stack & data segment

Memory Management with Bit Maps & List

- Drawback of bitmaps: to find consecutive 0 bits in the map is time-consuming.



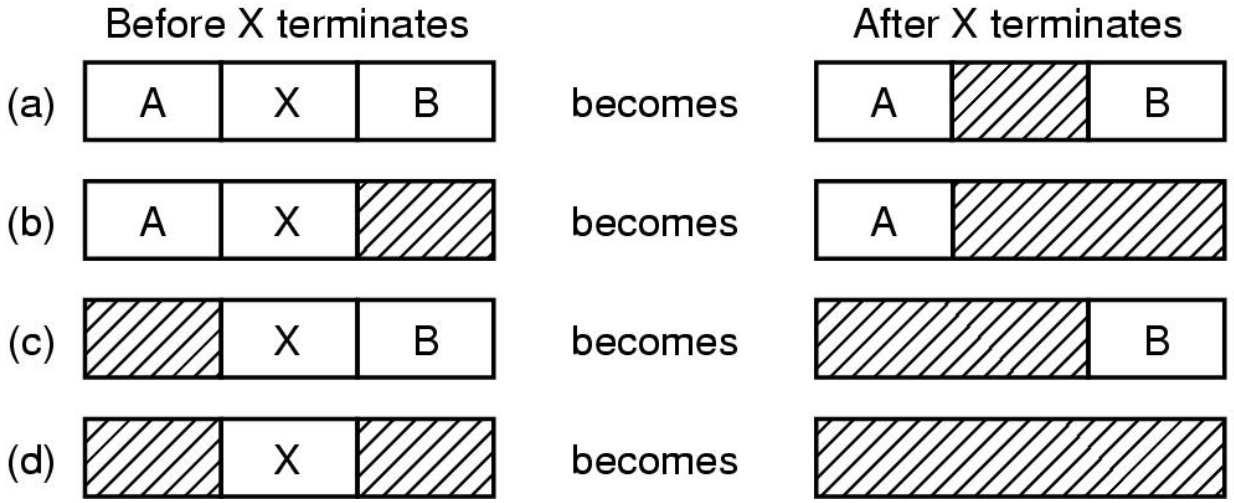
(a) Part of memory with 5 processes, 3 holes
- ✓tick marks show allocation units
- ✓shaded regions are free

(b) Corresponding bit map

(c) Same information as a list

Memory Management with Linked Lists

- Four neighbor combinations for the terminating process

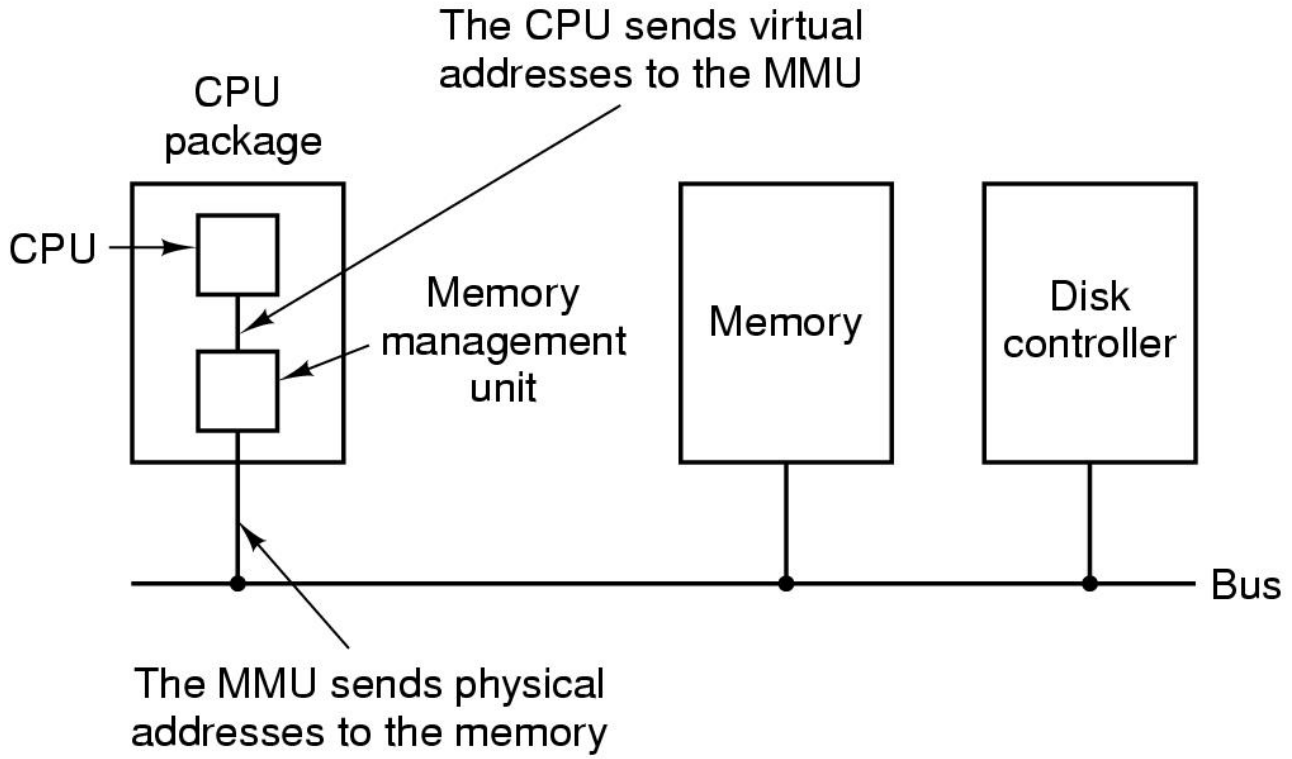


- Four algorithms for memory management:
  - ✓**First fit:** Searches from the beginning for a hole that fits.
  - ✓**Next fit:** Searches from the place where it left off last time for a hole that fits.
  - ✓**Best fit:** Searches the entire list and takes the smallest hole that fits.
  - ✓**Worst fit:** Searches the largest hole that fits.

Virtual Memory

- Problem: Program too large to fit in memory
- Solutions:
  - ✓Programmer splits program into pieces called **Overlays** - too much work
  - ✓**Virtual memory** - OS keeps the part of the program currently in use in memory
- **Paging** is a technique used to implement virtual memory.
- **Virtual Address** is a program generated address.
- The **MMU** (memory management unit) translates a virtual address into a physical address.

MMU



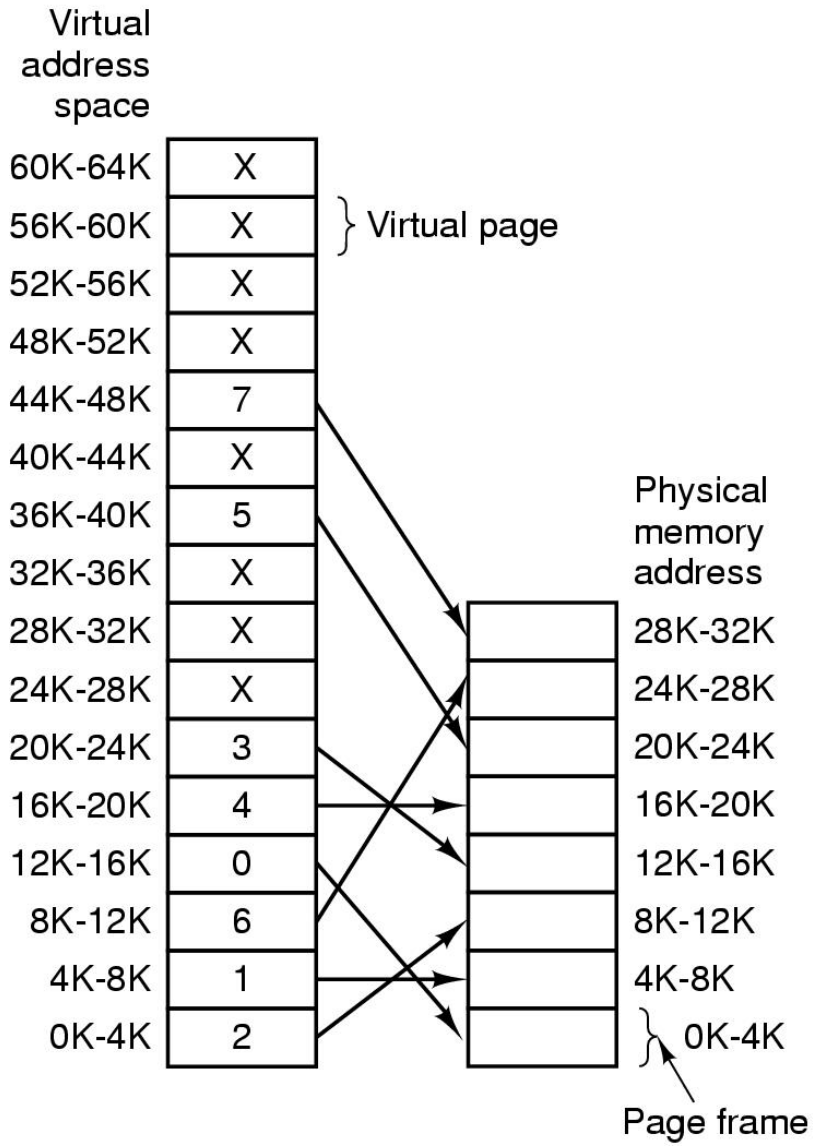
The position and function of the MMU

Virtual Memory

- The virtual address space is divided into (virtual) **pages** and those in the physical memory are (page) **frames**.
- A Present/Absent bit keeps track of whether or not the page is mapped.
- Reference to an unmapped page causes the CPU to trap to the OS.
- This trap is called a **Page fault**. The MMU selects a little used page frame, writes its contents back to disk, fetches the page just referenced, and restarts the trapped instruction.

Page Table

The relation between virtual addresses and physical memory addresses given by page table



Page Tables

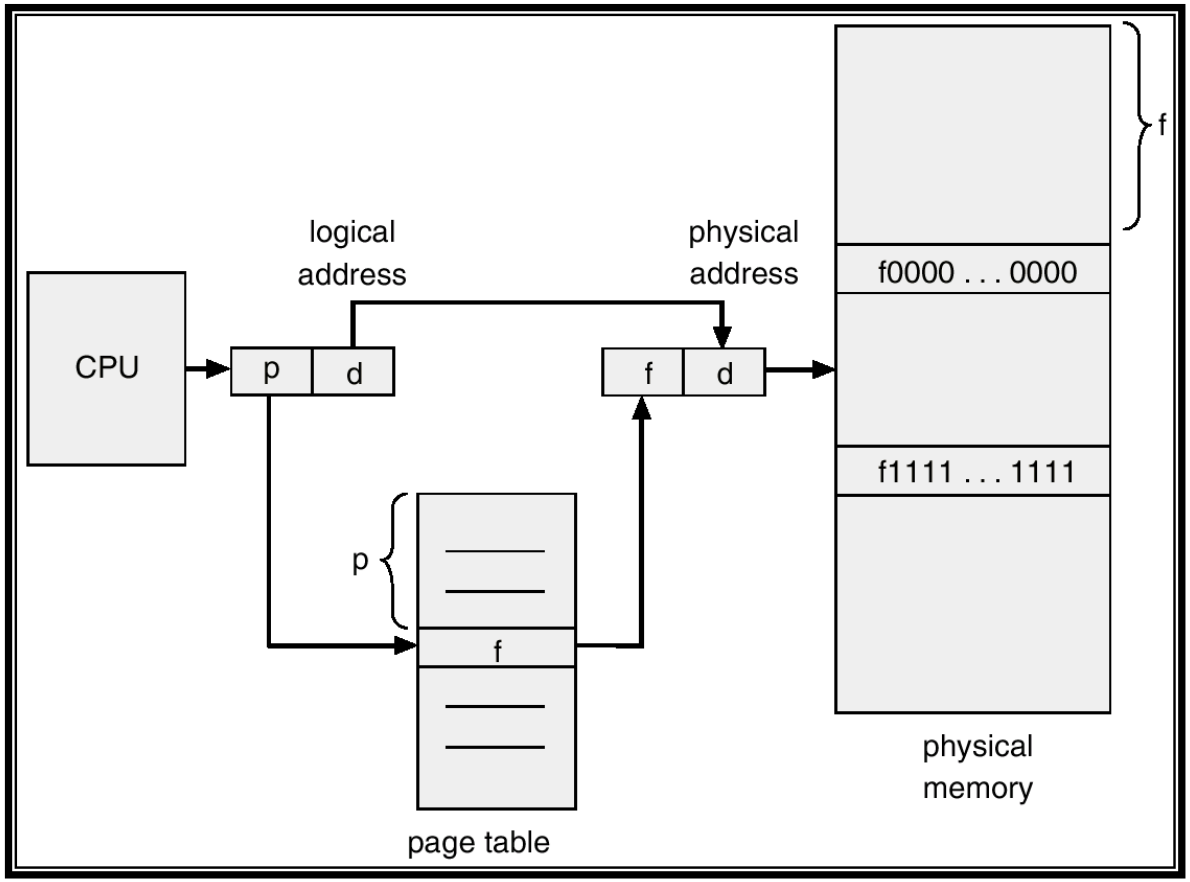
- Example: Virtual address = 4097 = 0001 000000000001

  Virtual page # 12-bit offset

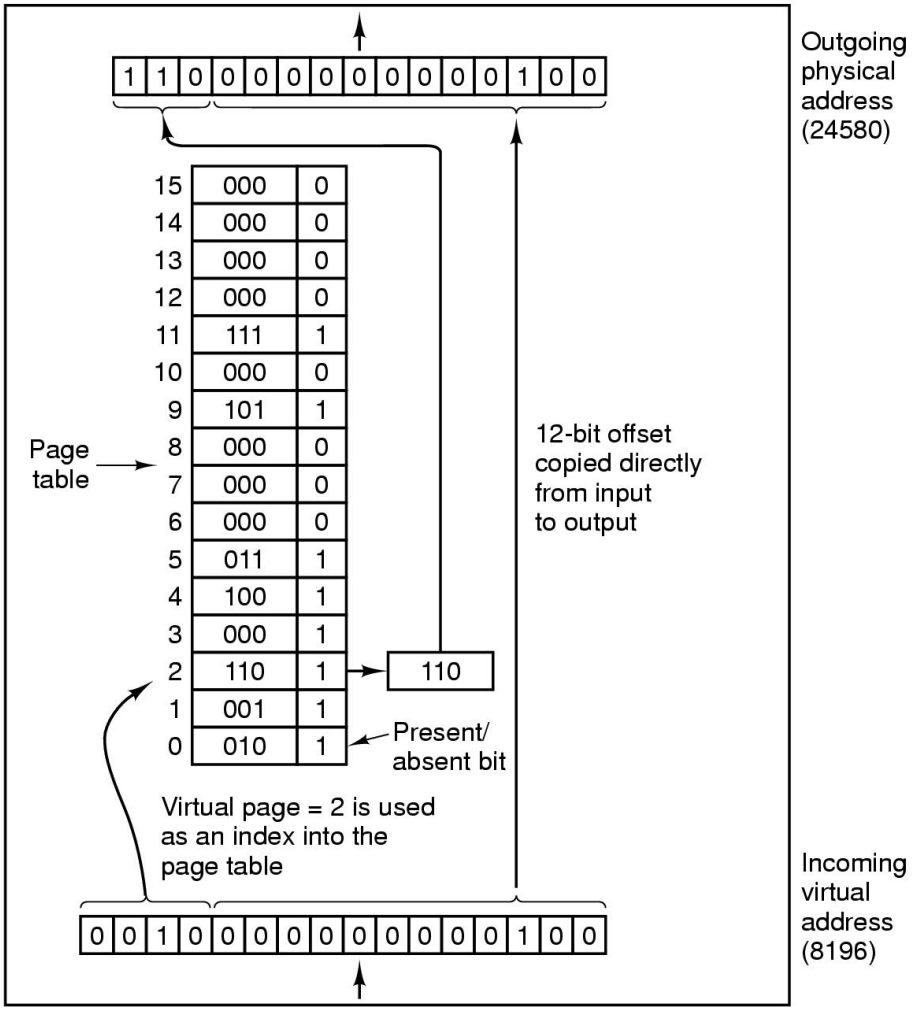
- The purpose of the page table is to map virtual pages into page frames. The page table is a function to map the virtual page to the page frame.

- Two major issues :
  - ✓Page tables may be extremely large (e.g. most computers use) 32-bit address with 4k page size, 12-bit offset
    - ➔ 20 bits for virtual page number
    - ➔1 million entries!
  - ✓The mapping must be fast because it is done on every memory access!!

Pure paging


CPU
logical address
p
d
physical address
f
d
f
p
f
page table
f0000 . . . 0000
f1111 . . . 1111
physical memory

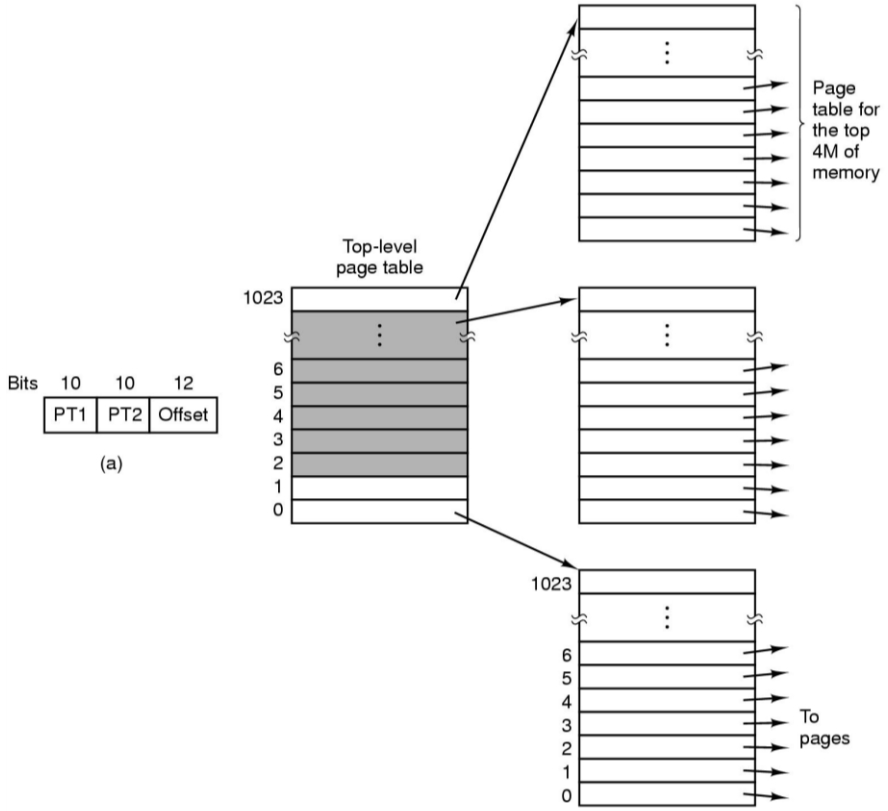
Page Table



Internal operation of MMU with 16 4 KB pages

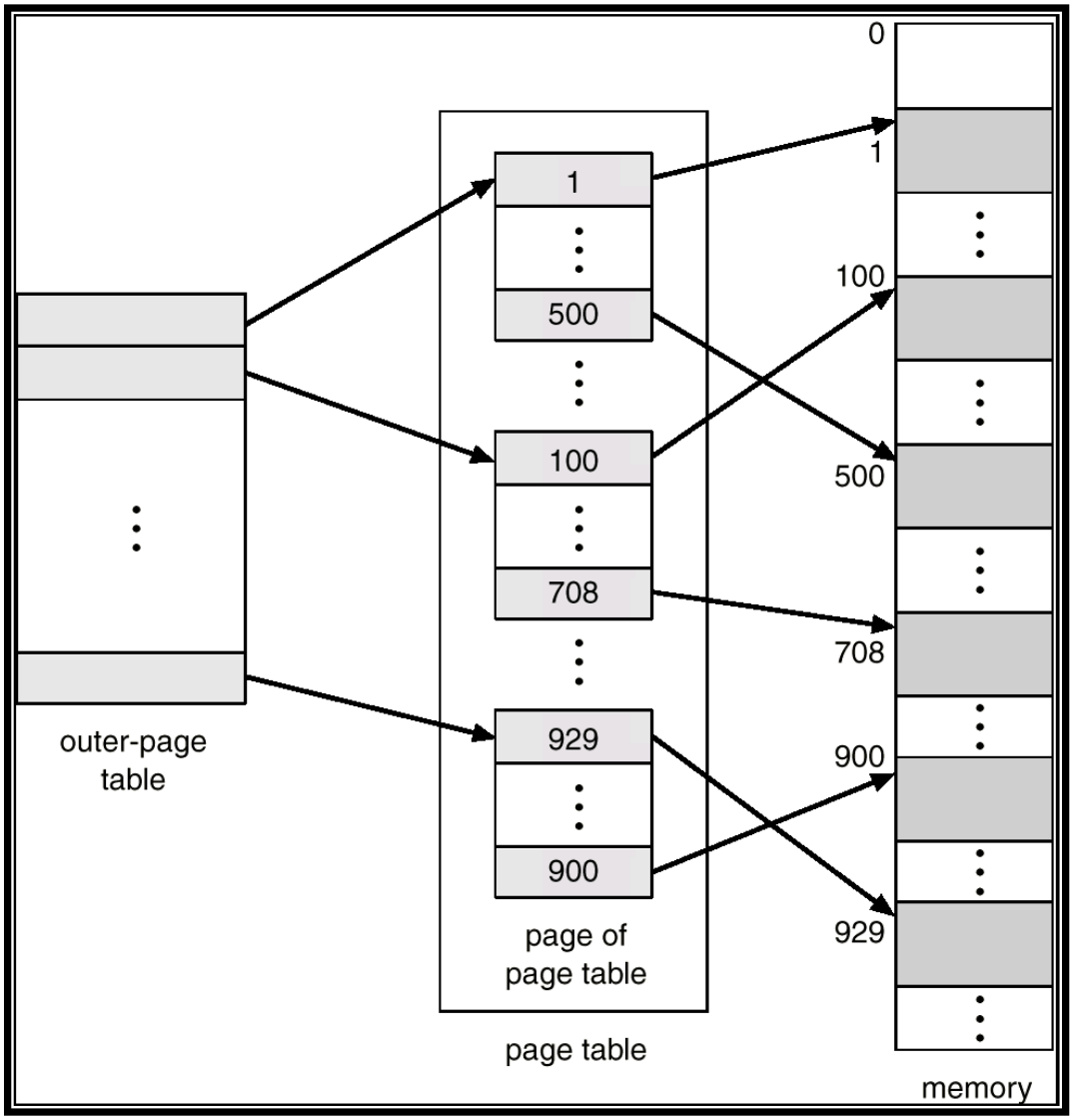
Multilevel Page Tables

- Multilevel page tables - reduce the table size. Also, don't keep page tables in memory that are not needed.



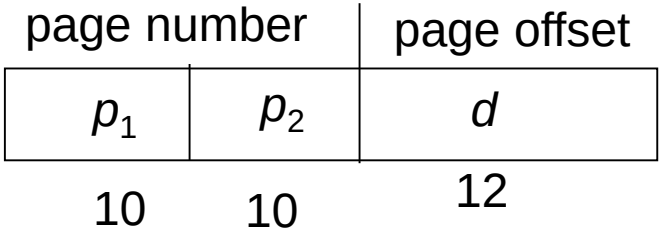
32 bit address with 2 page table fields

Two-Level Page-Table Scheme



Two-Level Paging Example

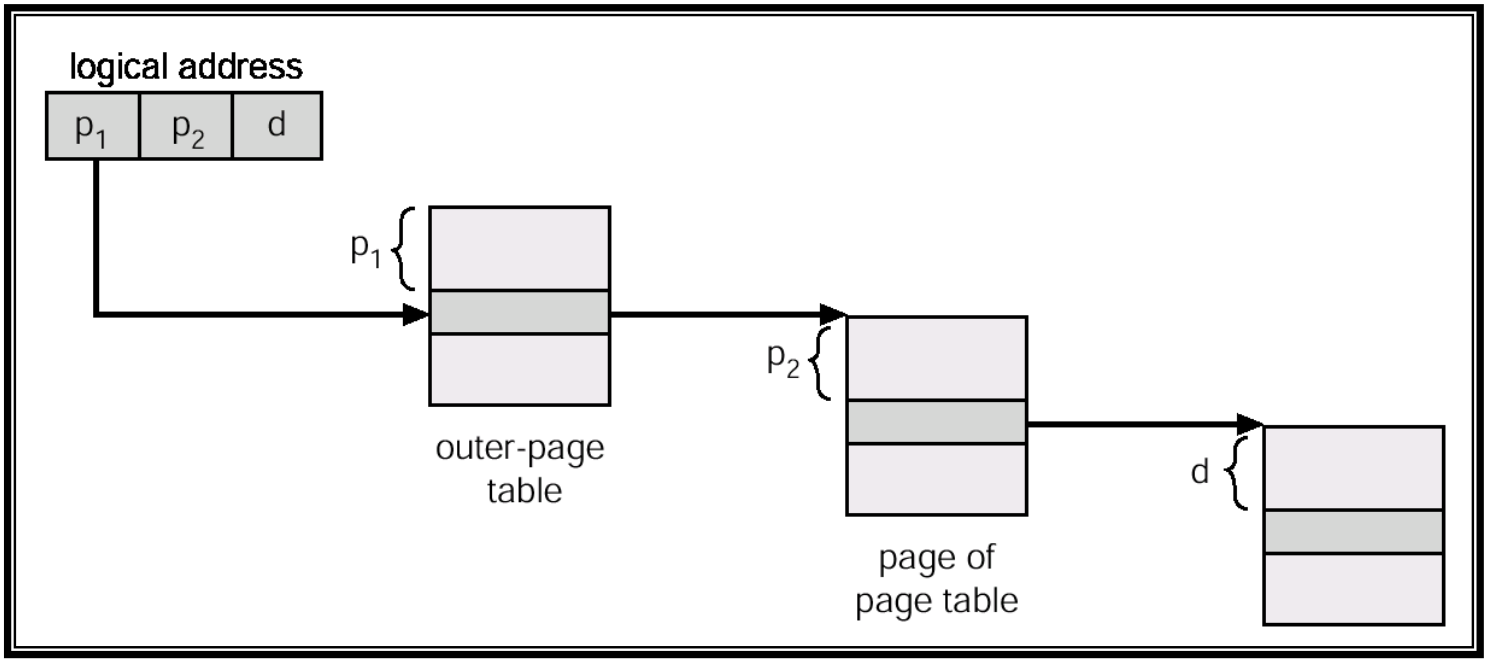
- A logical address (on 32-bit machine with 4K page size) is divided into:
  - ✓ a page number consisting of 20 bits.
  - ✓ a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - ✓ a 10-bit page number.
  - ✓ a 10-bit page offset.
- Thus, a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



where p_1 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

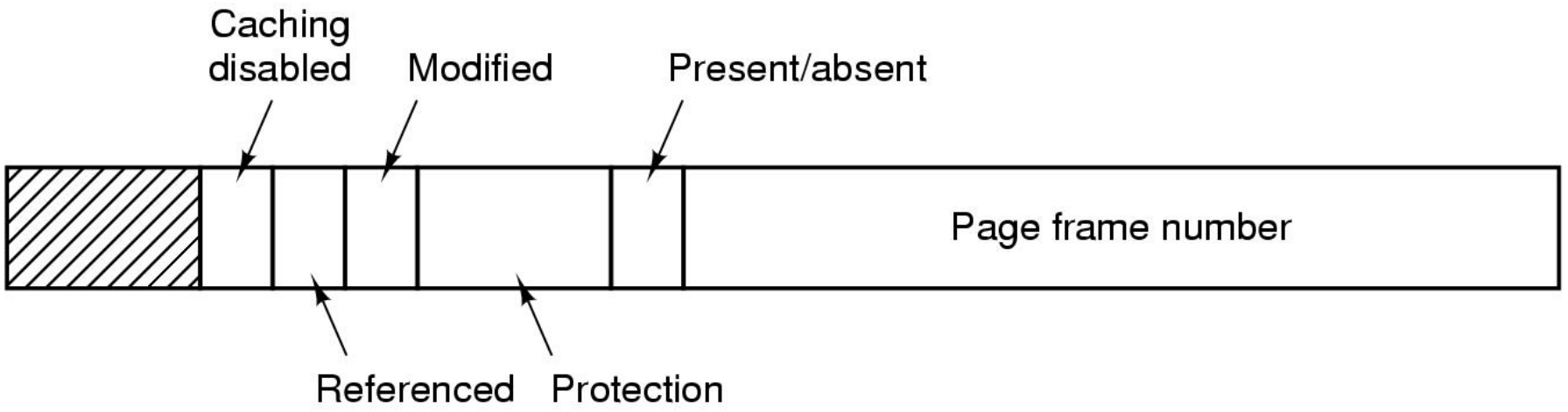
- Address-translation scheme for a two-level 32-bit paging architecture is shown as below.



Page Tables

- Most OSs allocate a page table for each process.
- Single page table consisting of an array of hardware registers. As a process is loaded, the registers are loaded with page table.
  - ✓Advantage - simple
  - ✓Disadvantage - expensive if table is large and loading the full page table at every context switch hurts performance.
- Leave page table in memory - a single register points to the table
  - ✓Advantage - context switch cheap
  - ✓Disadvantage - one or more memory references to read table entries

Page Tables



Typical page table entry

Structure of a Page Table Entry

- Page frame number: map the frame number
- Present/absent bit: 1/0 indicates valid/invalid entry
- Protection bit: what kinds of access are permitted.
- Modified – set when modified and writing to the disk occur
- Referenced - Set when page is referenced (help decide which page to evict)
- Caching disabled - used to keep data that logically belongs on the disk in memory to improve performance

TLB

- **Observation:** Most programs make a large number of references to **a small number of pages**.

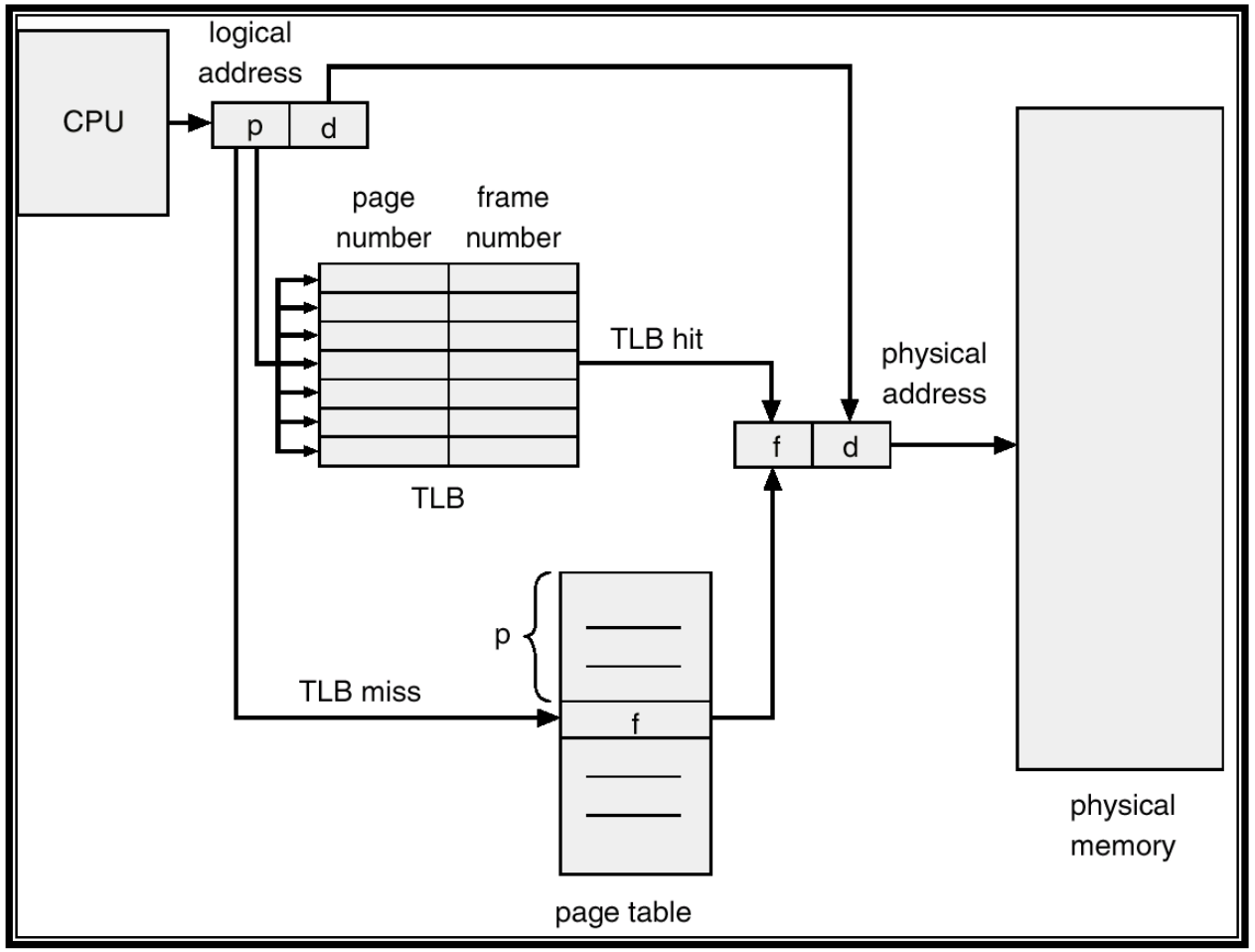
- **Solution:** Equip computers with a small hardware device, called **Translation Look-aside Buffer (TLB)** or **associative memory**, to map virtual addresses to physical addresses without using the page table.

TLB – Translation Lookaside Buffer

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

A TLB to speed up paging

Paging Hardware With TLB



Inverted Page Table

- Usually, each process has a page table associated with it. One of drawbacks of this method is that each page table may consist of millions of entries.
- To solve this problem, an **inverted page table** can be used. There is one entry for each real page (frame) of memory.
- Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Inverted Page Table

- To illustrate this method, a simplified version of the implementation of the inverted page is described as:

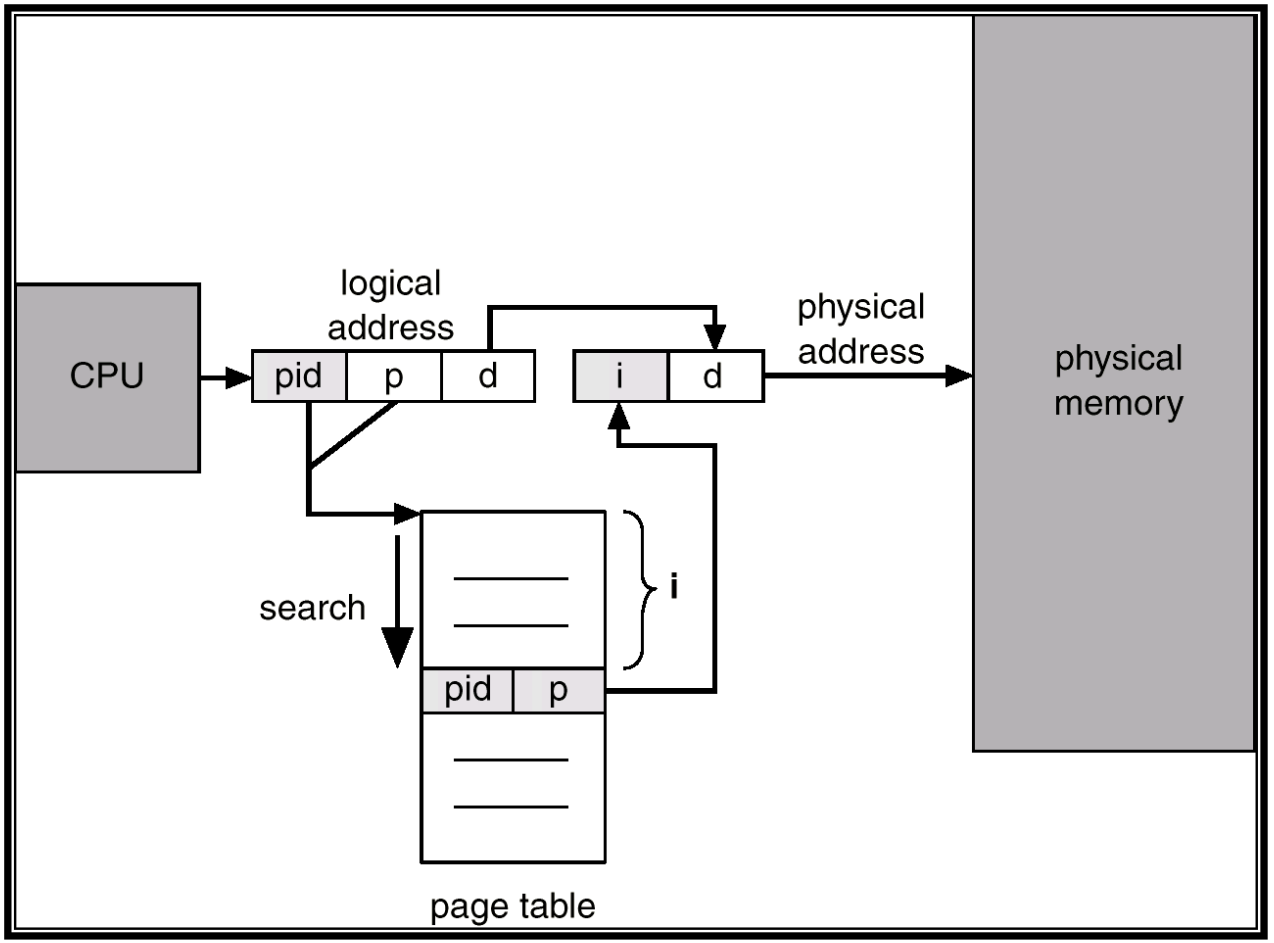
<process-id, page-number, offset>

- Each entry： <process-id, page-number>.

The inverted page table is then searched for a match. If a match *i* is found, then the physical address <i, offset> is generated. Otherwise, an illegal address access has been attempted.

- Although it decreases memory needed to store each page table, but **increases time needed to search the table** when a page reference occurs.

Inverted Page Table Architecture



Check Points

- What is the drawback of the bitmap method for free memory management?
- What is the purpose of virtual memory?
- What is page table?
- What is the purpose of using multi-level page table?
- What is the purpose of using TLB?
- What is invert page table?

Presentation & Poster

① Group 1,10: Process Scheduling
② Group 2,11: The Banker's Algorithm
③ Group 3,12: Virtual Memory Paging
④ Group 4,13: Page Replacement Algorithm
⑤ Group 5,14: Memory Management (Link List)
⑥ Group 6,15: Disk space Management
⑦ Group 7,16: Interrupt/DMA
⑧ Group 8,17: Disk Arm Scheduling Algorithm
⑨ Group 9,18: Files: Link List Allocation

华南理工大学2019～2020学年度第一学期校历

年月、周次 / 日期 / 星期	2019年																		2020年							
	8月	9月				10月				11月					12月				1月		2月					
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	寒 假					
星期一	26	2	9	16	23	30	7	14	21	28	4	11	18	25	2	9	16	23	30	6	13	20	27	3	10	17
星期二	27	3	10	17	24	1	8	15	22	29	5	12	19	26	3	10	17	24	31	7	14	21	28	4	11	18
星期三	28	4	11	18	25	2	9	16	23	30	6	13	20	27	4	11	18	25	1	8	15	22	29	5	12	19
星期四	29	5	12	19	26	3	10	17	24	31	7	14	21	28	5	12	19	26	2	9	16	23	30	6	13	20
星期五	30	6	13	20	27	4	11	18	25	1	8	15	22	29	6	13	20	27	3	10	17	24	31	7	14	21
星期六	31	7	14	21	28	5	12	19	26	2	9	16	23	30	7	14	21	28	4	11	18	25	1	8	15	22
星期日	1	8	15	22	29	6	13	20	27	3	10	17	24	1	8	15	22	29	5	12	19	26	2	9	16	23
	教 学 18 周																		考试2周		寒 假 6 周					

第6周	星期一	5-8节	B3	138
第12周	星期二	5-8节	B3	231
第13周	星期一	1-4节	B3	138
第13周	星期三	1-4节	B3	138
第13周	星期五	1-4节	B3	234