Chapter 4

Memory Management

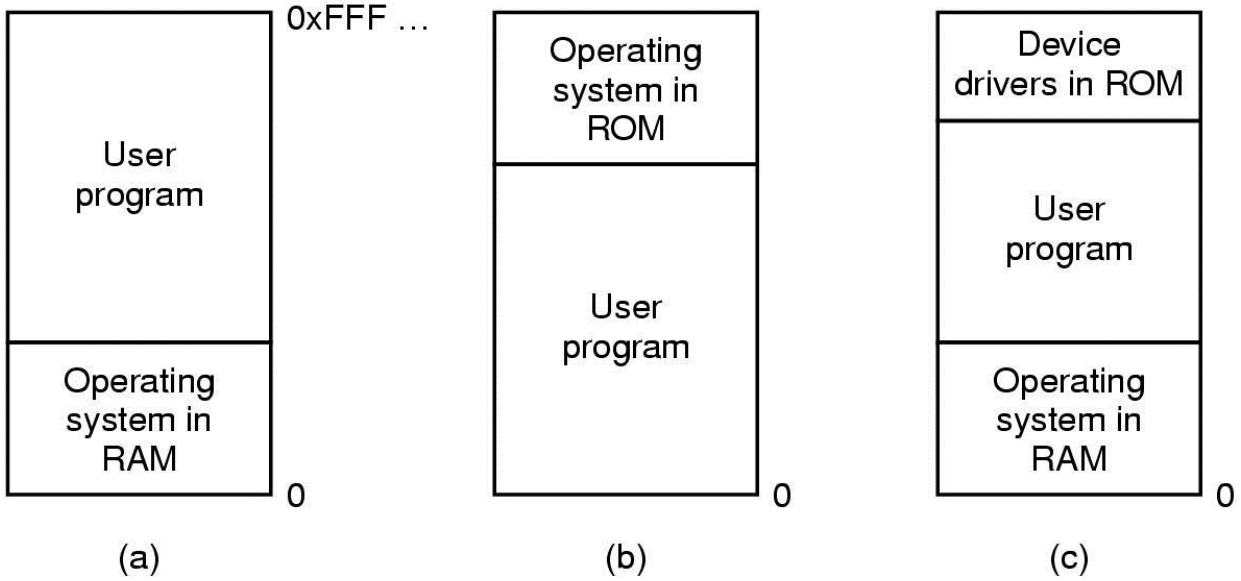
4.1 Basic memory management
4.2 Swapping
4.3 Virtual memory
4.4 Page replacement algorithms
4.8 Segmentation

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - nonvolatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager
  - part of the operating system that manages the memory hierarchy
  - Keeps track of which parts of memory are in use and which parts are not in use
  - Allocate memory to processes and deallocate it
  - Manages swapping between main memory and disk when main memory is too small to hold all the processes

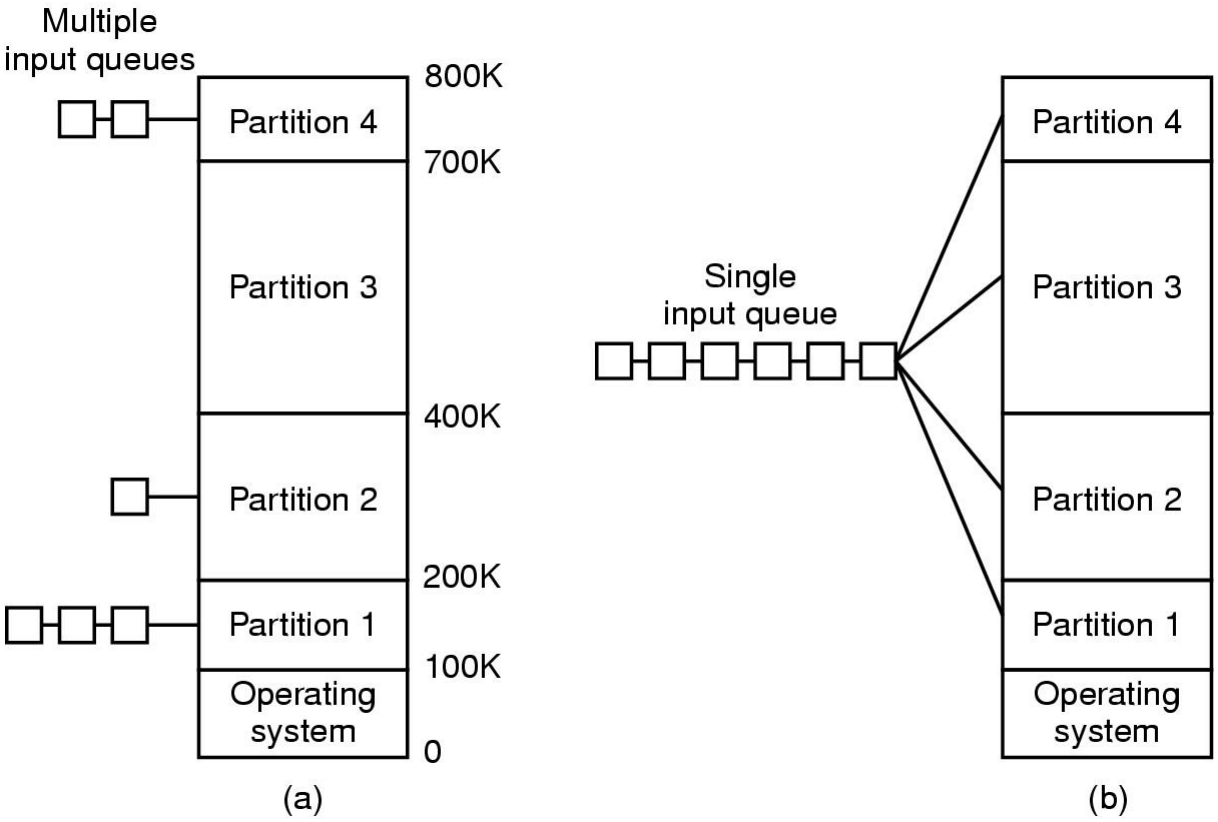
Basic Memory Management

Monoprogramming without Swapping or Paging



- an operating system with one user process
  - Only one process at a time is running
- Three simple ways of organizing memory
  - an operating system with one user process
  - (a) mainframes, minicomputers
  - (b) palmtop computers, embedded systems
  - (c) personal computers

Multiprogramming with Fixed Partitions



- Fixed memory partitions
  - separate input queues for each partition
  - single input queue

Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - Relocation
    - address locations of variables, code routines cannot be absolute
  - Protection
    - must keep a program out of other processes' partitions

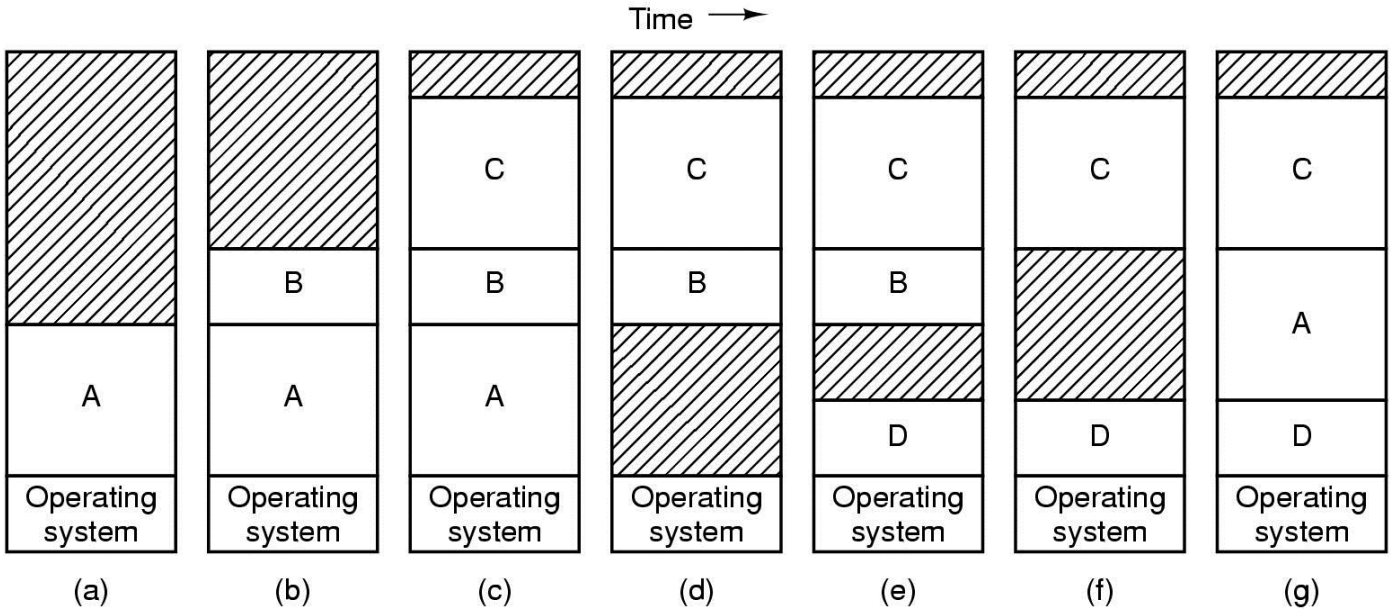
- Relocation and Protection
  - Modify the instruction as the program is loaded into memory
    - Linker should tell which program words are addresses to be relocated
    - Protection code
  - Use base and limit registers
    - address locations added to base value to map to physical addr
    - address locations larger than limit value is an error

Memory Management

- Swapping
  - Bring in each process in its entirety, running it for a while, then put it back on disk
- Virtual Memory
  - Allows programs to run even when they are only partially in main memory

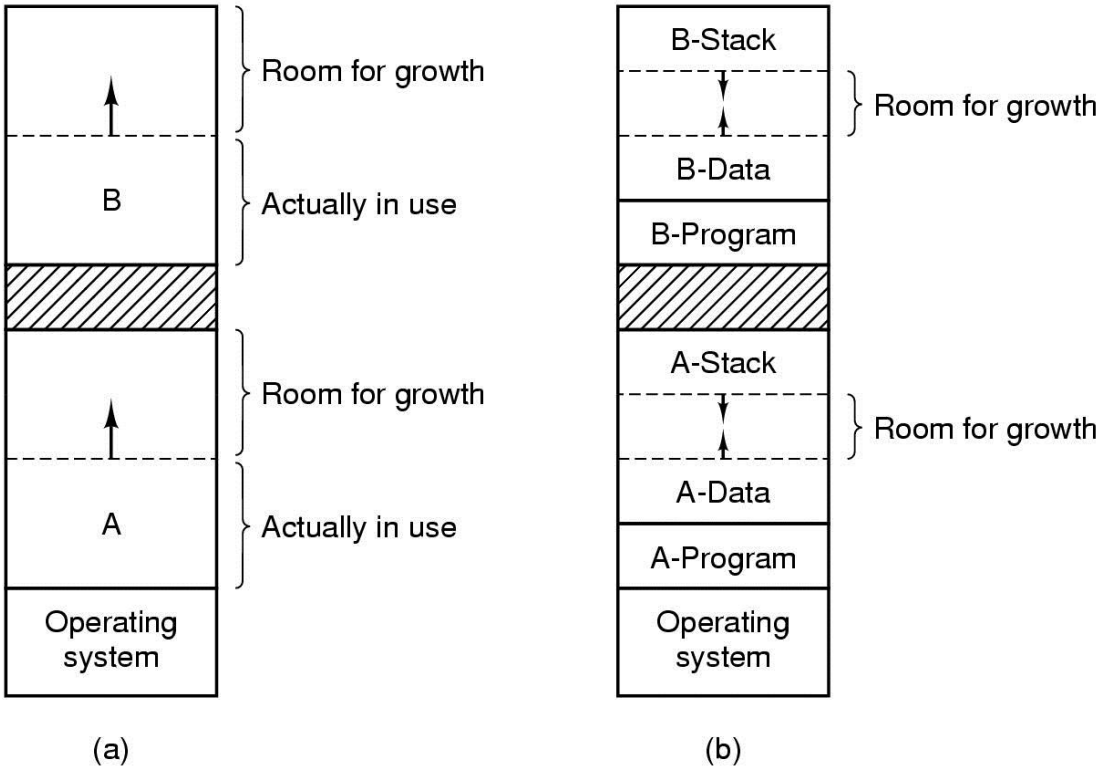
Swapping

Multiprogramming with Variable Partitions



- Memory allocation changes as
  - processes come into memory
  - leave memory
- Shaded regions are unused memory
- Memory compaction
  - Combine multiple holes into one big one by moving all the processes downward

Swapping



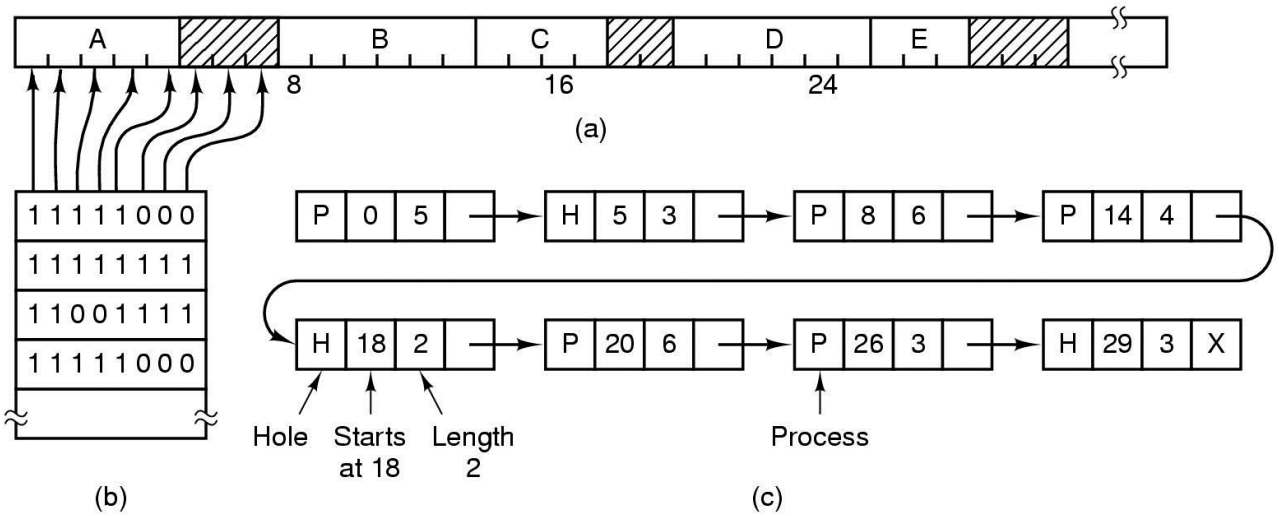
(a) Allocating space for growing data segment

(b) Allocating space for growing stack & data segment

Keeping Track of Memory Usage

- Bit Maps
- Free Lists

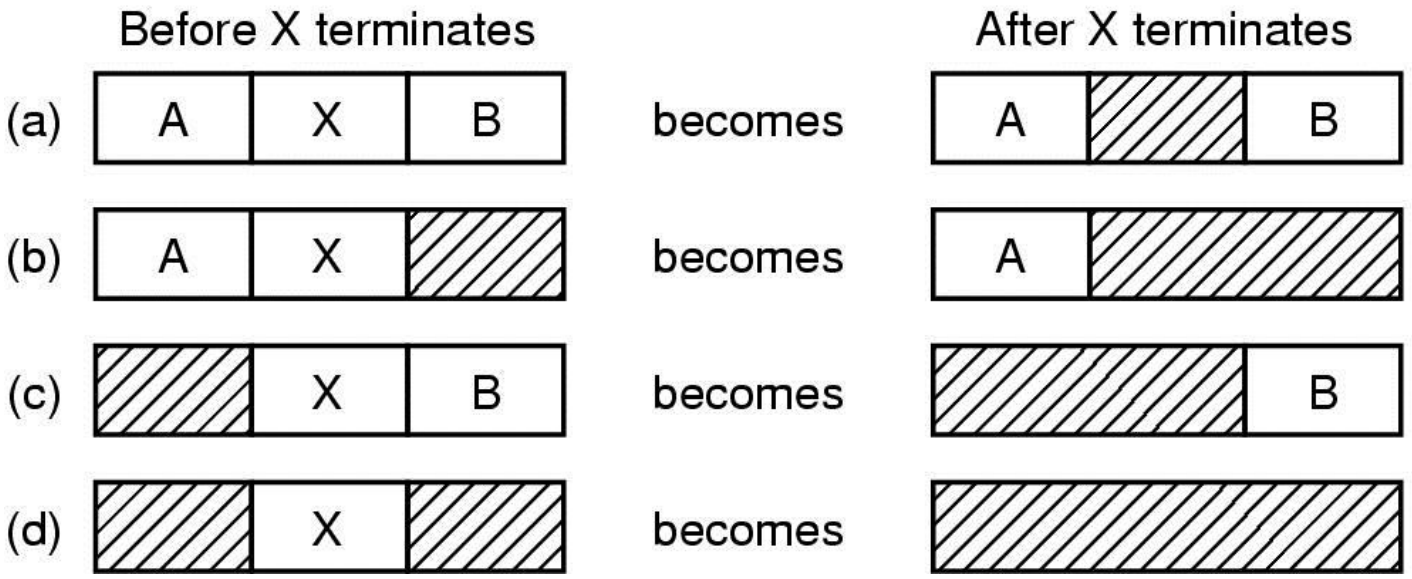
Memory Management with Bit Maps



- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Size of the allocation unit
  - The smaller the allocation unit, the larger the bitmap
  - The larger the allocation unit, the more memory is wasted
- Searching a bitmap for a run of a given length is a slow operation.

Memory Management with Linked Lists

- Maintains a linked list of allocated and free memory segments
- Segment List is kept sorted by address
  - Updating the list is straightforward
- Four neighbor combinations for the terminating process X



Memory Management with Linked Lists

- Allocating Memory for a newly created process
  - First fit
    - Scans along the list until it finds a hole that is big enough
  - Best fit
    - Searches the entire list and takes the smallest hole available
  - Worst fit
    - Takes the largest hole available

Virtual Memory

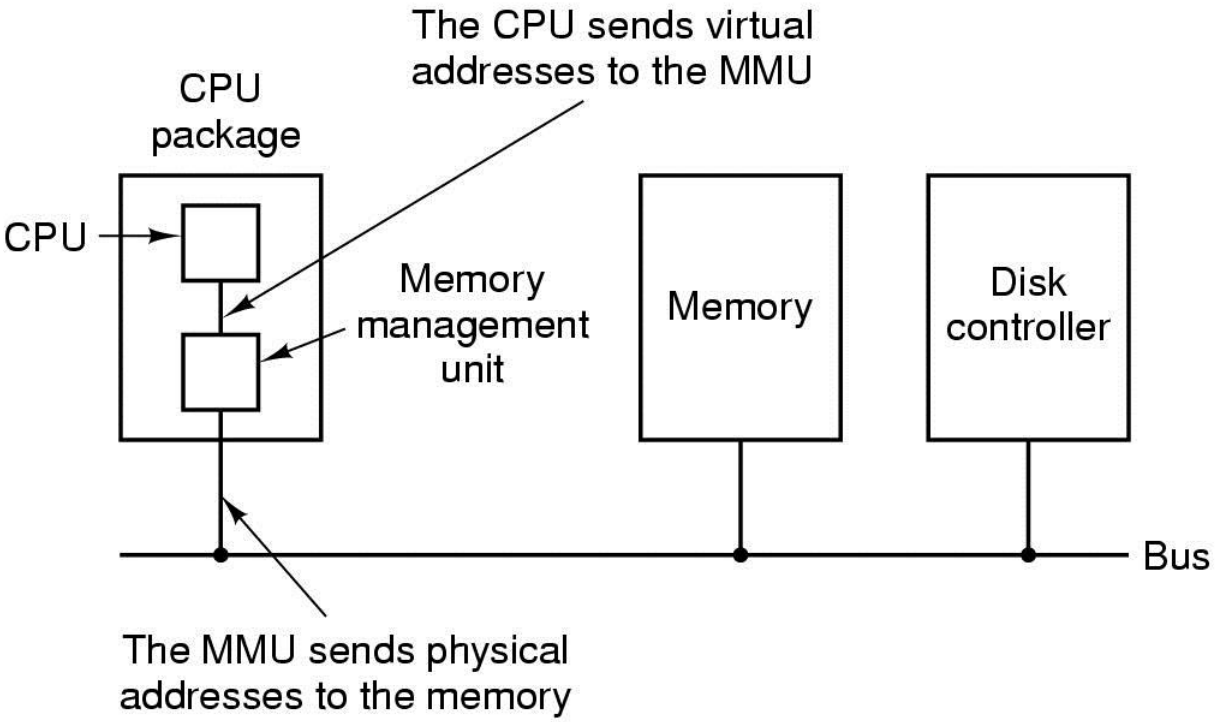
- Programs that are too big to fit in the available memory
- Solution
  - Overlay
    - A programmer splits the program into pieces, called overlays.
    - When the first overlay is done, it calls another overlay, which is swapped into the memory from the disk.
  - Virtual memory
    - Computer handles the job.
    - Paging

Virtual Memory
Paging

- Virtual address
  - Program generated address
    - MOV REG, 1000
  - Forms the virtual address space
  - Virtual address space is divided up into units called *pages* and the corresponding units in the physical memory are called *page frames*.
  - Pages and page frames are always the same size.
  - Transfers between RAM and disk are always in units of a page.

Virtual Memory
Paging

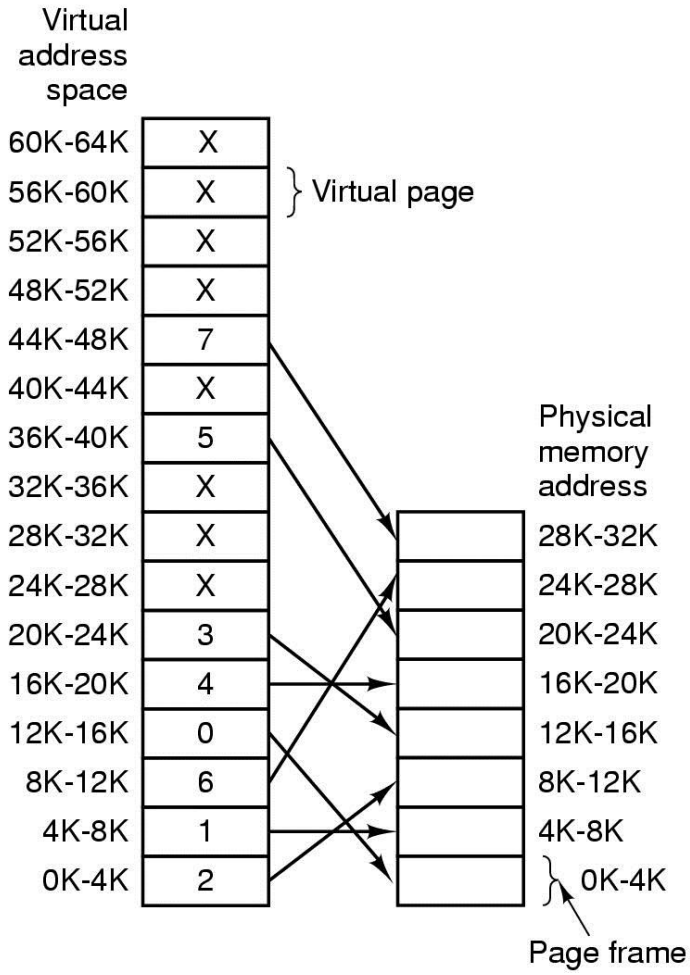
- The position and function of the MMU



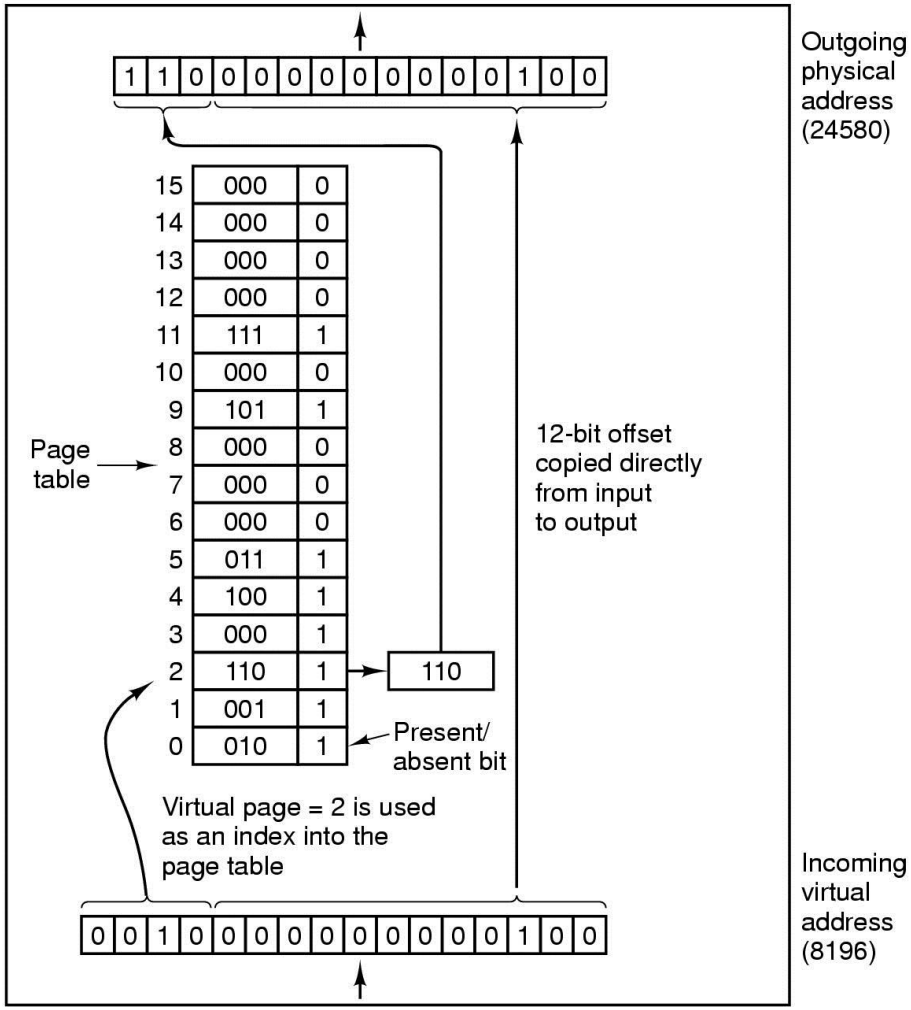
Paging

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

- Page fault
  - When the MMU notices that the page is unmapped, it causes the CPU to trap to the operating system.
  - The OS picks a little-used page frame and writes its contents back to the disk.
  - It then fetches the page just referenced into the page frame just freed, changes the map, and restarts the trapped instruction.



Page Tables



Internal operation of MMU with 16 4 KB pages

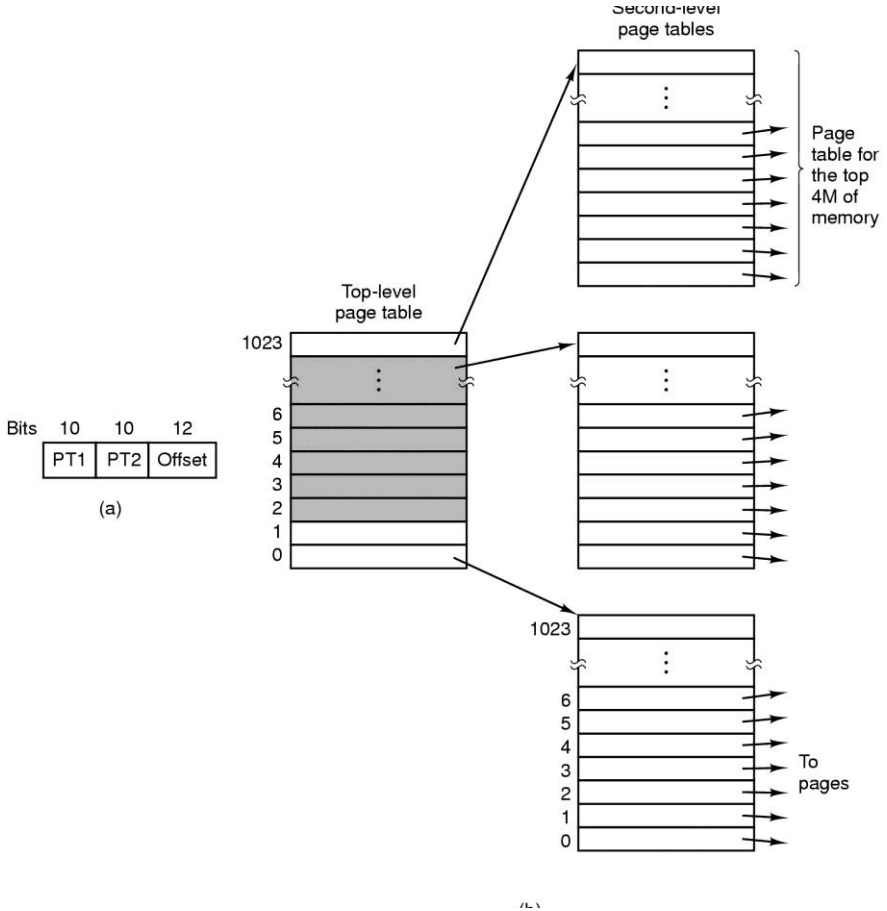
Page Tables

- Two major issues
  - The page table can be extremely large.
    - 32-bit address space
      - 1 million 4-KB pages
      - The page table must have 1 million entries.
    - 64-bit address space
  - The mapping must be fast.
    - The virtual-to-physical mapping must be done on every memory reference.

Page Tables

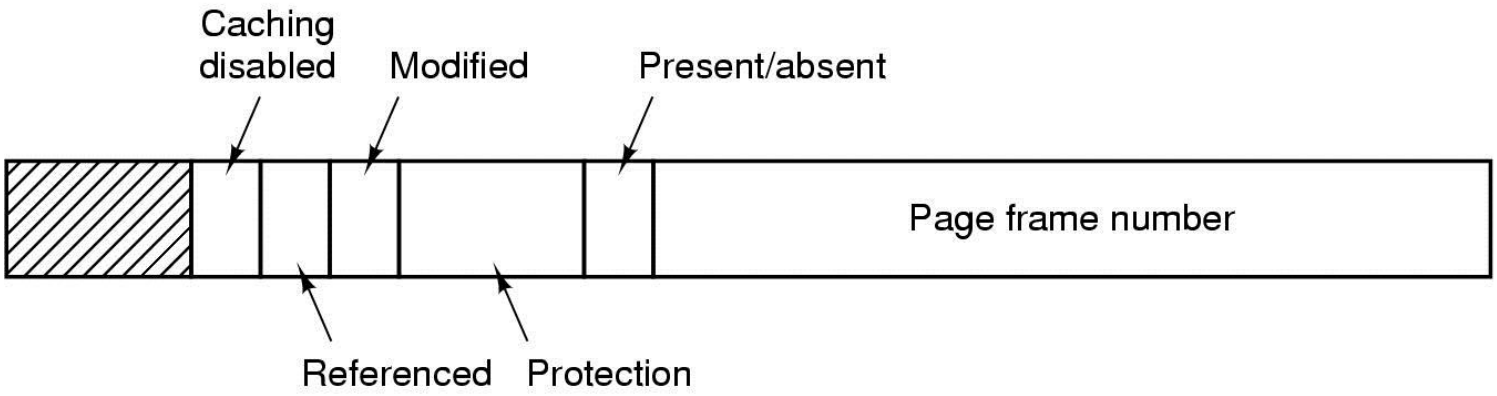
- Two designs
  - Single page table consisting of an array of fast hardware registers (figure in the previous page)
    - Requires no memory references during mapping
    - Loading the full page table at every context switch is expensive.
  - Page table in main memory
    - A single register points to the start of the page table.
    - Fast memory map change by reloading one register
    - Requires memory references during the execution of each instruction

Multilevel Page Table



- Avoids keeping all the page tables in memory all the time.
- 32 bit address with 2 page table fields
- Two-level page tables

Page Tables (3)



- Typical page table entry
  - Page frame number
  - Present/absent bit
  - Protection bits: what kinds of access are permitted
    - e.g. 0: read/write, 1: read only
  - Modified bit(dirty bit) : sets when a page is modified
  - Referenced: sets when a page is referenced (for reading or writing)
  - Caching disabled: for memory-mapped I/O

TLBs – Translation Lookaside Buffers

- TLB or associative memory
  - Based on the observation that most programs tend to make a large number of references to a small number of pages
  - Small hardware device for mapping virtual addresses to physical addresses without going through the page table
  - It is usually inside the MMU and consists of a small number of entries.
- How the TLB functions
  - A virtual address is presented to the MMU.
  - The hardware checks to see if the virtual page number is present in the TLB.
  - If the valid match is found and the access does not violate the protection bits, the page frame is taken directly from the TLB.
  - If not, the MMU detects the miss and does an ordinary page table look up. It then evicts one of the entries from the TLB and replaces it with the page table entry just looked up.
  - When an entry is purged from the TLB, the modified bit is copied back into the page table entry in memory.

TLBs – Translation Lookaside Buffers

- A TLB to speed up paging

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

Inverted Page Tables

Comparison of a traditional page table with an inverted page table