Memory Management

**The need for memory management:**

* Memory is cheap today, and getting cheaper - but applications are demanding more and more memory, there is never enough!
* Memory Management, involves swapping blocks of data from secondary storage.
* Memory I/O is slow compared to a CPU - the OS must cleverly time the swapping to maximise the CPU’s efficiency.

Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

***Memory Management Requirements:***

* Relocation.
* Protection.
* Sharing.
* Logical organisation.
* Physical organisation.

**Relocation.**

* The programmer does not know where the program will be placed in memory when it is executed - it may be swapped to disk and return to main memory at a different location (relocated).
* Memory references must be translated to the actual physical memory address.

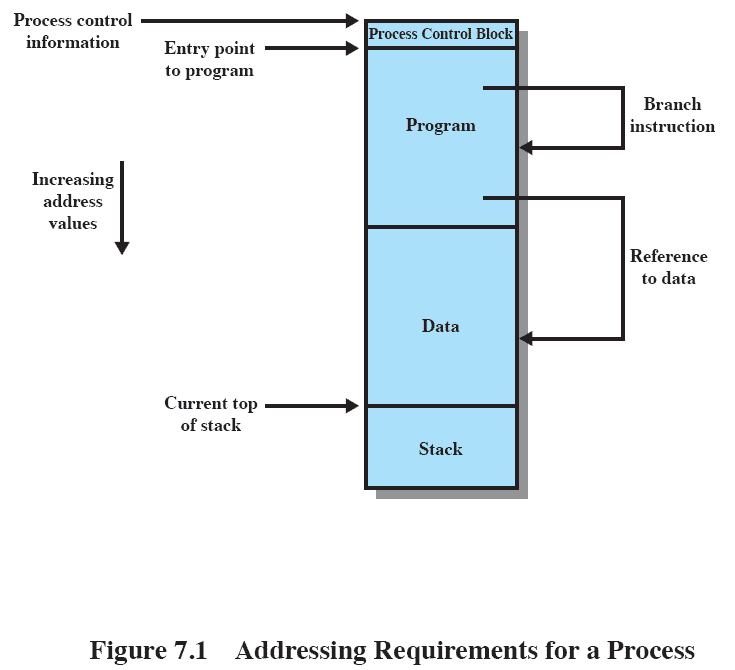
**Memory Management Terms:**

Frame - ***Fixed***-length block of main memory.

Page - ***Fixed***-length block of data in secondary memory (e.g. on disk).

Segment - ***Variable-length*** block of data that resides in secondary memory.

**Addressing.**

****

Because the operating system knows this information because it is managing memory and is responsible for bringing this process into main memory. However, the processor must deal with memory references within the program. Branch instructions contain an address to reference the instruction to be executed next. Data reference instructions contain the address of the byte or word of data referenced. Somehow, the processor hardware and operating system software must be able to translate the memory references found in the code of the program into actual physical memory addresses, reflecting the current location of the program in main memory.

**Protection.**

* Processes should not be able to reference memory locations in another process without permission.
* Impossible to check absolute addresses at compile time.
* Must be checked at run time.

**Sharing.**

* Allow several processes to access the same portion of memory.
* Better to allow each process access to the same copy of the program rather than have their own separate copy.

**Logical Organization.**

* Memory is organized linearly (usually).
* Programs are written in modules - Modules can be written and compiled independently.
* Different degrees of protection given to modules (read-only, execute-only).
* Share modules among processes.
* Segmentation helps here.

**Physical Organization.**

* Cannot leave the programmer with the responsibility to manage memory.
* Memory available for a program plus its data may be insufficient - Overlaying allows various modules to be assigned the same region of memory but is time consuming to program.
* Programmer does not know how much space will be available.

Partitioning

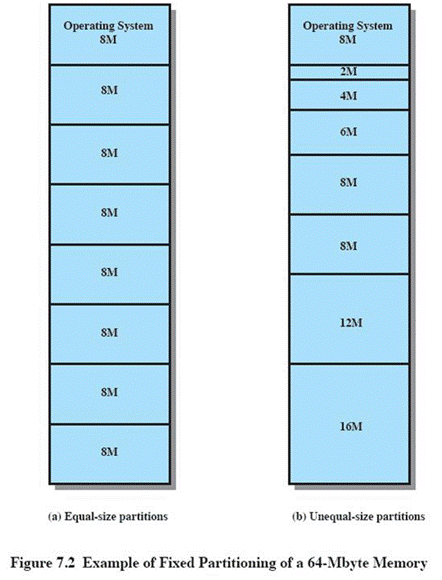
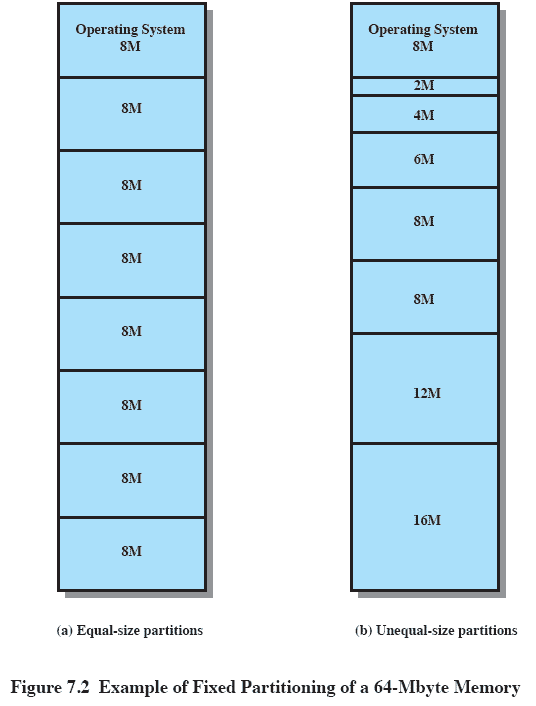
* An early method of managing memory.
  + Pre-virtual memory.
  + Not used much now.
* But, it will clarify the later discussion of virtual memory if we look first at partitioning - Virtual Memory has evolved from the partitioning methods.

***Types of Partitioning:***

* Fixed Partitioning
* Dynamic Partitioning
* Simple Paging
* Simple Segmentation
* Virtual Memory Paging
* Virtual Memory Segmentation

**Fixed Partitioning.**

* Equal-size partitions - Any process whose size is less than or equal to the partition size can be loaded into an available partition.
* The operating system can swap a process out of a partition - If none are in a ready or running state.

**Fixed Partitioning Problems:**

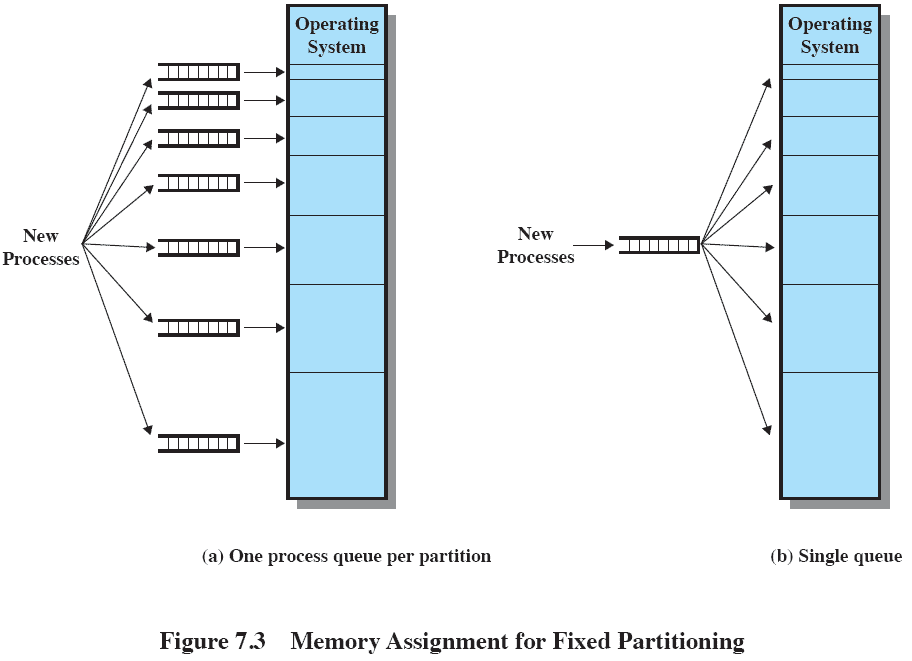
* A program may not fit in a partition - The programmer must design the program with overlays.
* Main memory use is inefficient. Any program, no matter how small, occupies an entire partition - This results in ***internal fragmentation****.*

**Solution – Unequal Size Partitions:**

* Lessens both problems
  + but doesn’t solve completely
* In Fig,
  + Programs up to 16M can be accommodated without overlay
  + Smaller programs can be placed in smaller partitions, reducing internal fragmentation

**Placement Algorithm:**

* Equal-size - Placement is trivial (no options).
* Unequal-size:
  + Can assign each process to the smallest partition within which it will fit.
  + Queue for each partition.
  + Processes are assigned in such a way as to minimize wasted memory within a partition.

**Fixed Partitioning.**

**(Single queue is better.)**

**Remaining Problems with Fixed Partitions:**

* The number of active processes is limited by the system - I.E limited by the pre-determined number of partitions.
* A large number of very small process will not use the space efficiently - in either fixed or variable length partition methods.

**Dynamic Partitioning.**

* Partitions are of variable length and number.
* Process is allocated exactly as much memory as required.

OS (8M)

**Dynamic Partitioning Example:**

P2

(14M)

* ***External Fragmentation.***
* Memory external to all processes is fragmented.

P4(8M)

* Can resolve using ***compaction.***

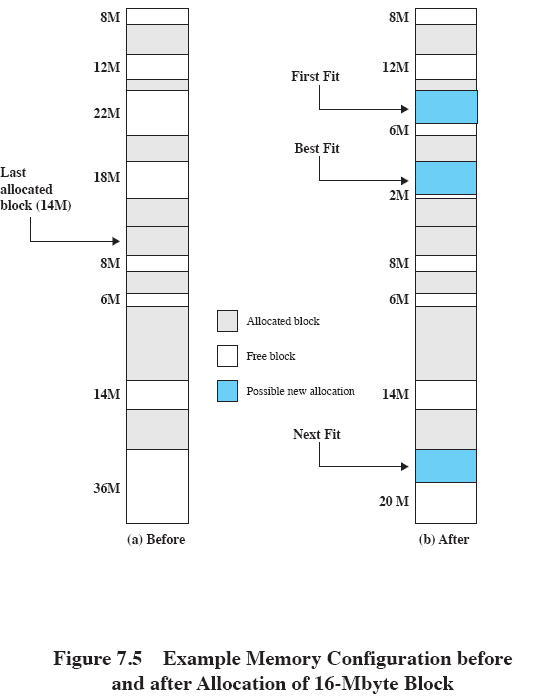
Empty

(6M)

* + OS moves processes so that they are contiguous.
  + Time consuming and wastes CPU time.

P3

(18M)

**Allocation.**

Empty (4M)

**Dynamic Partitioning.**

* Operating system must decide which free block to allocate to a process

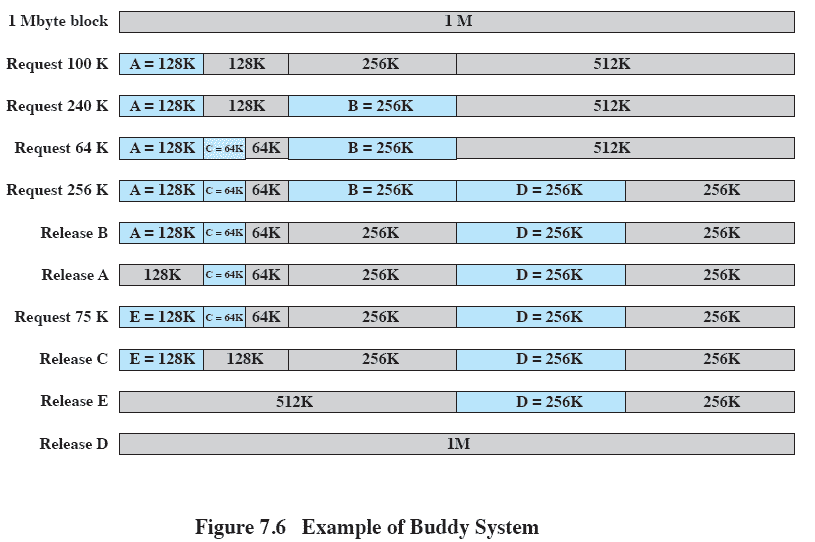
**Best-fit algorithm:**

* + Chooses the block that is closest in size to the request.
  + Worst performer overall.
  + Since smallest block is found for process, the smallest amount of fragmentation is left.
  + Memory compaction must be done more often.
* **First-fit algorithm:**
  + Scans memory form the beginning and chooses the first available block that is large enough
  + Fastest
  + May have many process loaded in the front end of memory that must be searched over when trying to find a free block
* **Next-fit algorithm:**
  + Scans memory from the location of the last placement
  + More often allocate a block of memory at the end of memory where the largest block is found
  + The largest block of memory is broken up into smaller blocks
  + Compaction is required to obtain a large block at the end of memory

**Buddy System.**

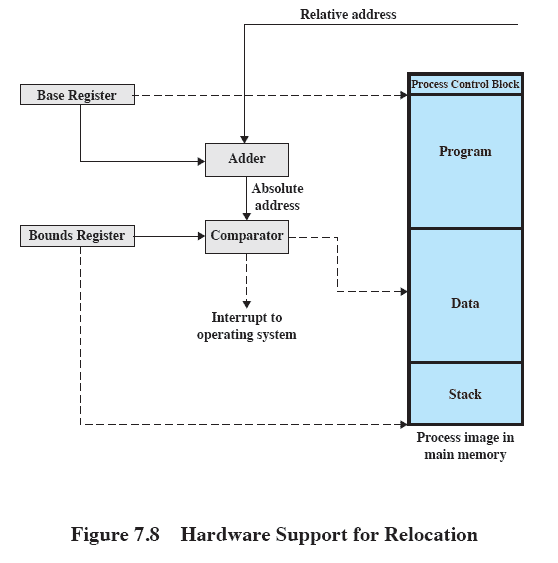
* Entire space available is treated as a single block of 2U.
* If a request of size *s* where 2*U*-1 < *s* <= 2*U -* entire block is allocated.
* Otherwise block is split into two equal buddies - process continues until smallest block greater than or equal to *s* is generated.

**Example of Buddy System.**

****

**Relocation.**

* When program loaded into memory the actual (absolute) memory locations are determined.
* A process may occupy different partitions which means different absolute memory locations during execution.
  + Swapping.
  + Compaction.



**Addresses:**

* **Logical** - Reference to a memory location independent of the current assignment of data to memory.
* **Relative** - Address expressed as a location relative to some known point.
* **Physical or Absolute -** The absolute address or actual location in main memory.

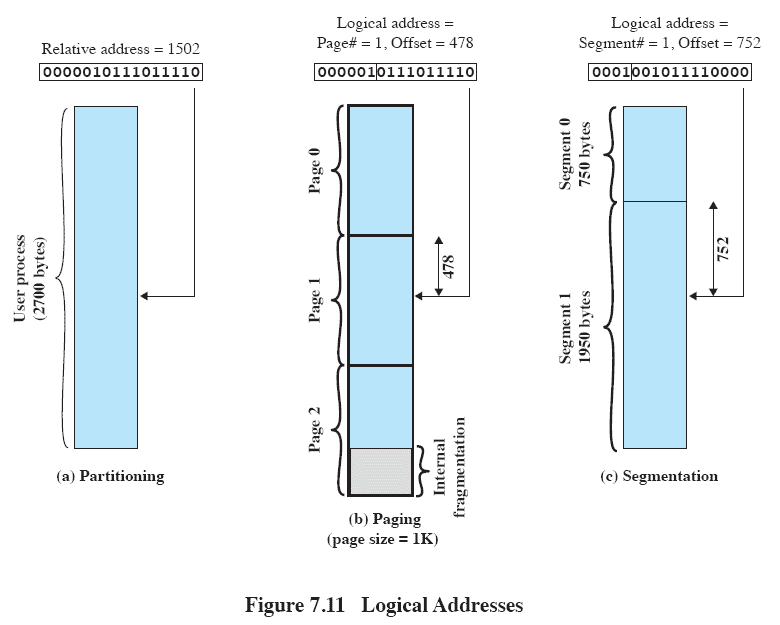
**Paging.**

* Partition memory into small equal fixed-size chunks and divide each process into the same size chunks.
* The chunks of a process are called ***pages***.
* The chunks of memory are called ***frames.***
* Operating system maintains a page table for each process:
* Contains the frame location for each page in the process.
* Memory address consist of a page number and offset within the page.

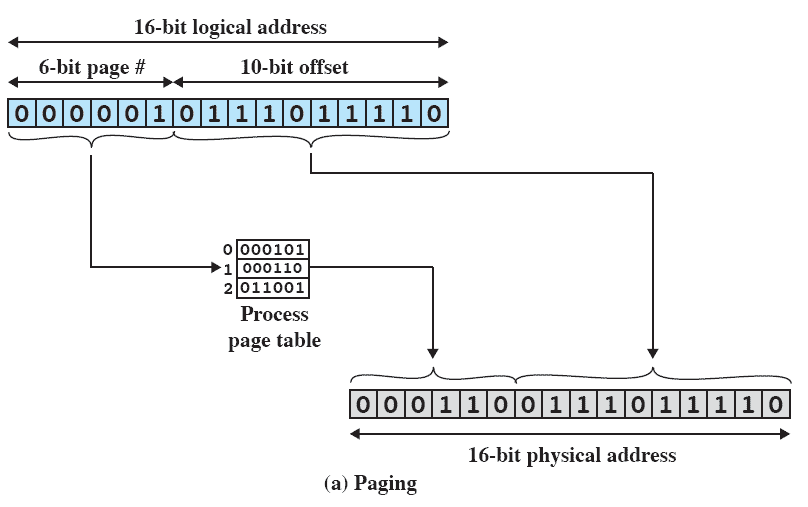
**Segmentation.**

* A program can be subdivided into segments:
  + Segments may vary in length.
  + There is a maximum segment length.
* Addressing consist of two parts: a segment number and an offset.
* Segmentation is similar to dynamic partitioning.

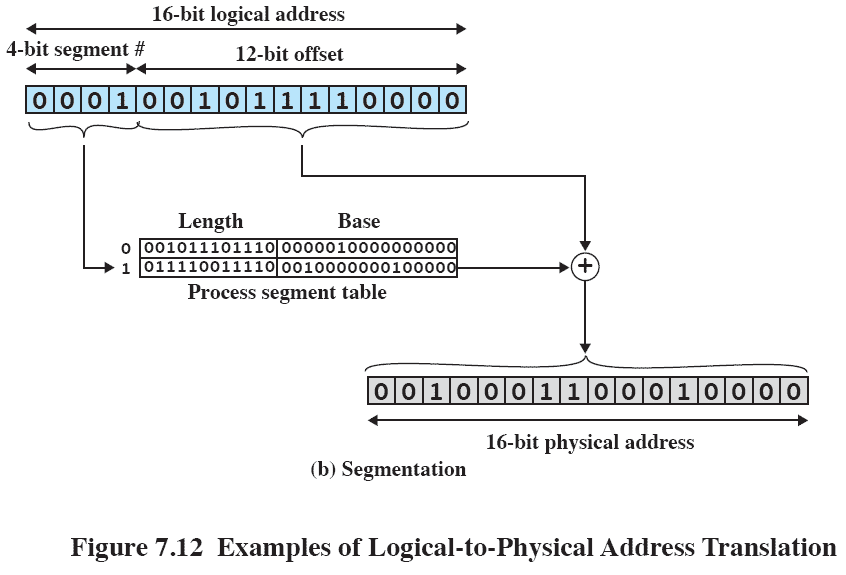
**Logical Addresses.**

****

**Paging.**

****

**Segmentation.**

****

VIRTUAL MEMORY

**Hardware and Control Structures:**

* Memory references are dynamically translated into physical addresses at run time – a process may be swapped in and out of main memory such that it occupies different regions.
* A process may be broken up into pieces that do not need to located contiguously in main memory - all pieces of a process do not need to be loaded in main memory during execution.

**Execution of a Program:**

* Operating system brings into main memory a few pieces of the program.
* Resident set - portion of process that is in main memory.
* An interrupt is generated when an address is needed that is not in main memory.
* Operating system places the process in a blocking state.

Piece of process that contains the logical address is brought into main memory.

* + Operating system issues a disk I/O Read request.
  + Another process is dispatched to run while the disk I/O takes place.
  + An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state.

**Advantages of Breaking up a Process:**

* More processes may be maintained in main memory.
  + Only load in some of the pieces of each process.
  + With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time.
* A process may be larger than all of main memory.

**Types of Memory:**

* Real memory - main memory
* Virtual memory - memory on disk - allows for effective multiprogramming and relieves the user of tight constraints of main memory.

**Thrashing.**

* Swapping out a piece of a process just before that piece is needed.
* The processor spends most of its time swapping pieces rather than executing user instructions.

**Support Needed for Virtual Memory.**

* Hardware must support paging and segmentation.
* Operating system must be able to management the movement of pages and/or segments between secondary memory and main memory.

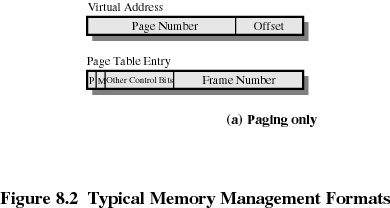
**Paging.**

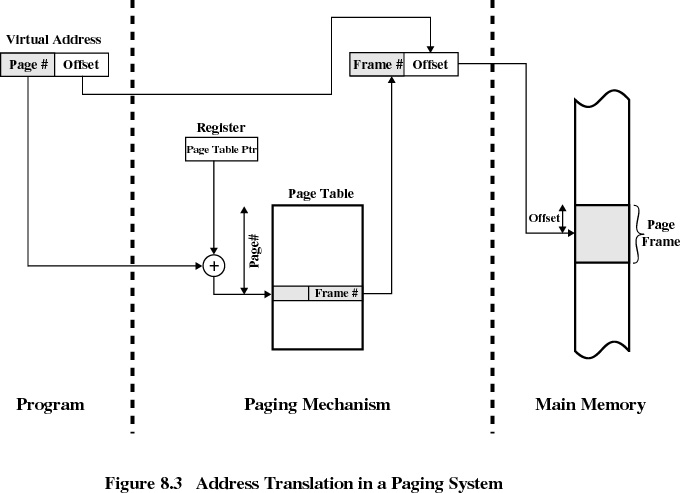
* Each process has its own page table.
* Each page table entry contains the frame number of the corresponding page in main memory.
* A bit is needed to indicate whether the page is in main memory or not.

**Modify Bit in Page Table.**

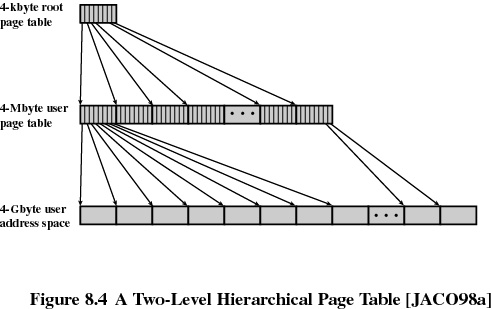
* Another modify bit is needed to indicate if the page has been altered since it was last loaded into main memory.
* If no change has been made, the page does not have to be written to the disk when it needs to be swapped out.

**Page Table Entries.**

****

****

**Two-Level Scheme for 32-bit Address.**

****

* If we assume byte-level addressing and 4-kbyte (212) pages,
  + then the 4-Gbyte (232) virtual address space is composed of 220pages.

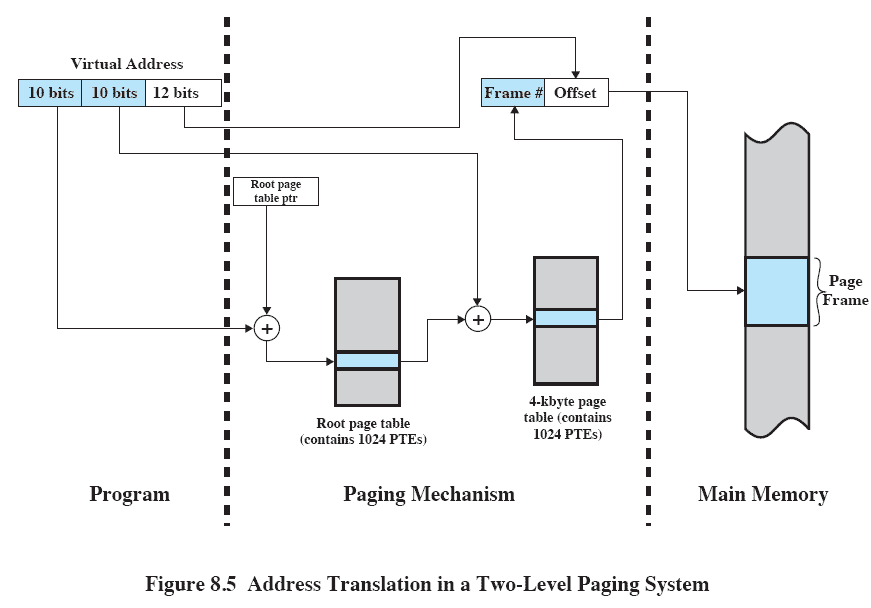
If each of these pages is mapped by a 4-byte page table entry (PTE), we can create a user page table composed of 220 PTEs requiring 4 Mbyte (222 ) bytes.

This huge user page table, occupying 210 pages, can be kept in virtual memory and mapped by a root page table with 210 PTEs occupying

* 4 Kbyte (212 ) of main memory.

**Page Tables:**

* The entire page table may take up too much main memory.
* Page tables are also stored in virtual memory.
* When a process is running, part of its page table is in main memory.
* The root page always remains in main memory.
* The first 10 bits of a virtual address are used to index into the root page to find a PTE for a page of the user page table.
  + If that page is not in main memory, a page fault occurs.
  + If that page is in main memory, then the next 10 bits of the virtual address index into the user PTE page to find the PTE for the page that is referenced by the virtual address.

**Address Translation for Hierarchical page table.** ****