Virtual Memory

Chapter 8

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 be 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

Execution of a Program

- 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

Breaking up Processes: Advantages

- 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

Risk of 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

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

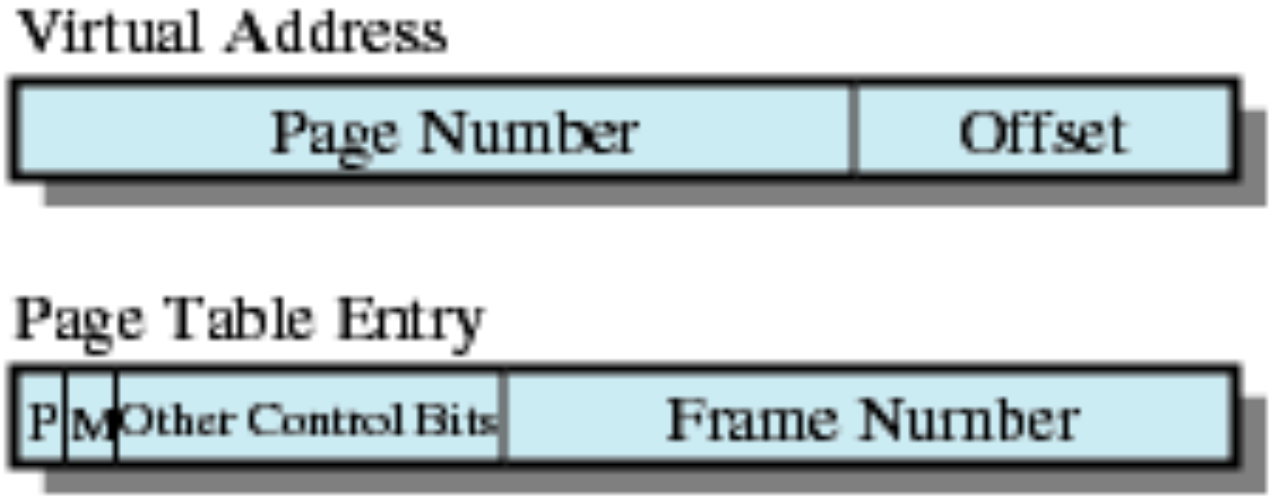
Support Needed for Virtual Memory

- Hardware must support paging and segmentation
- Operating system must be able to manage 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

Paging

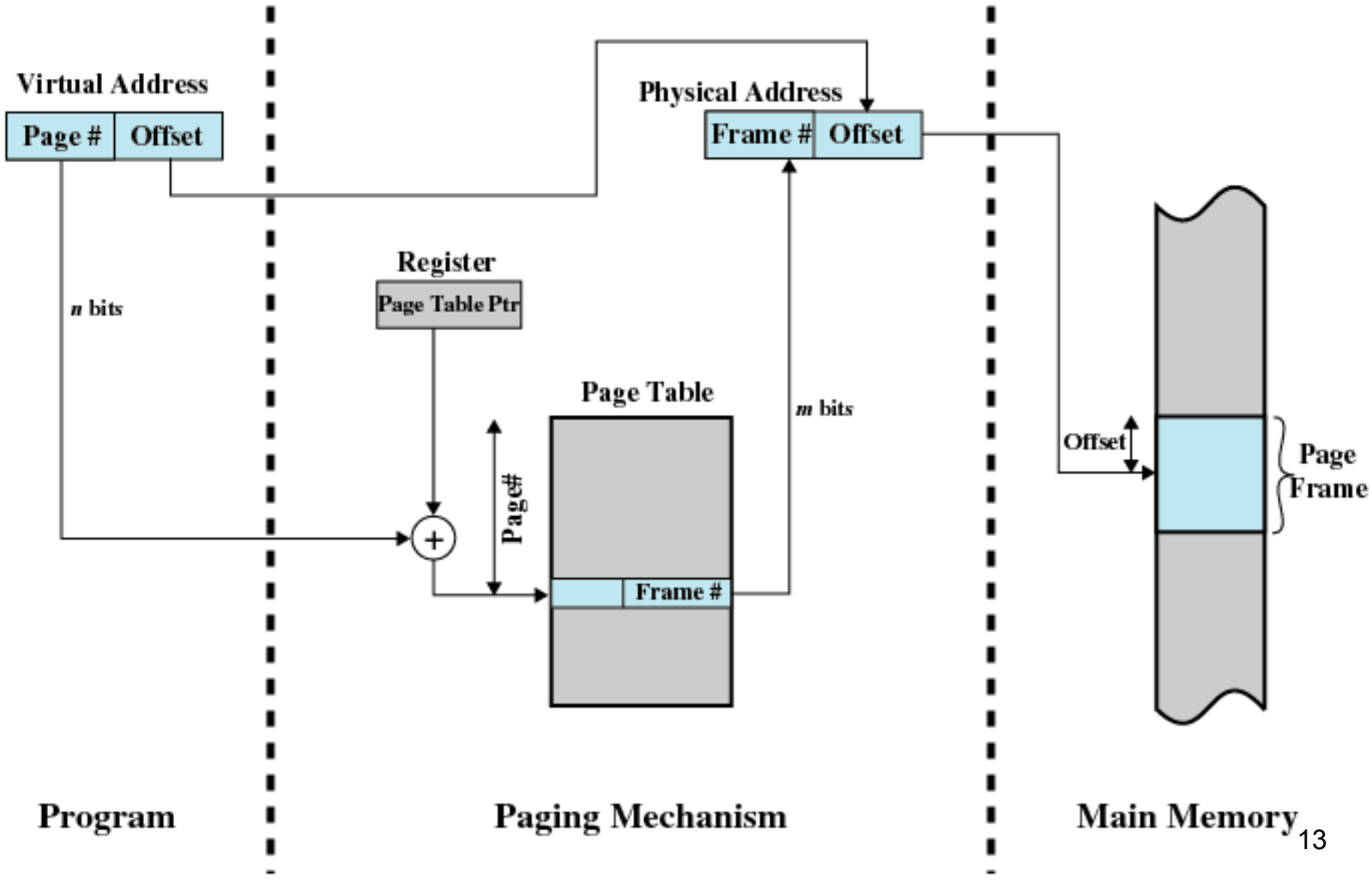


(a) Paging only

Modify Bit in Page Table

- 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

Paging: Address Translation



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

Two-Level Scheme: 32-bit Address

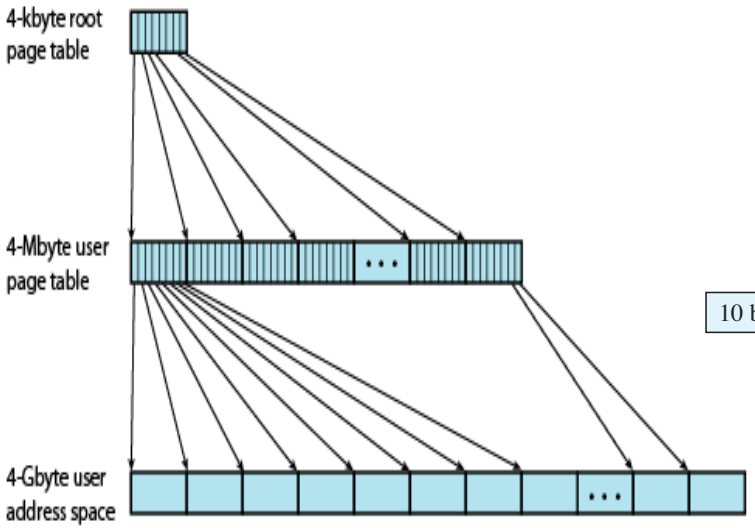


Figure 8.4 A Two-Level Hierarchical Page Table

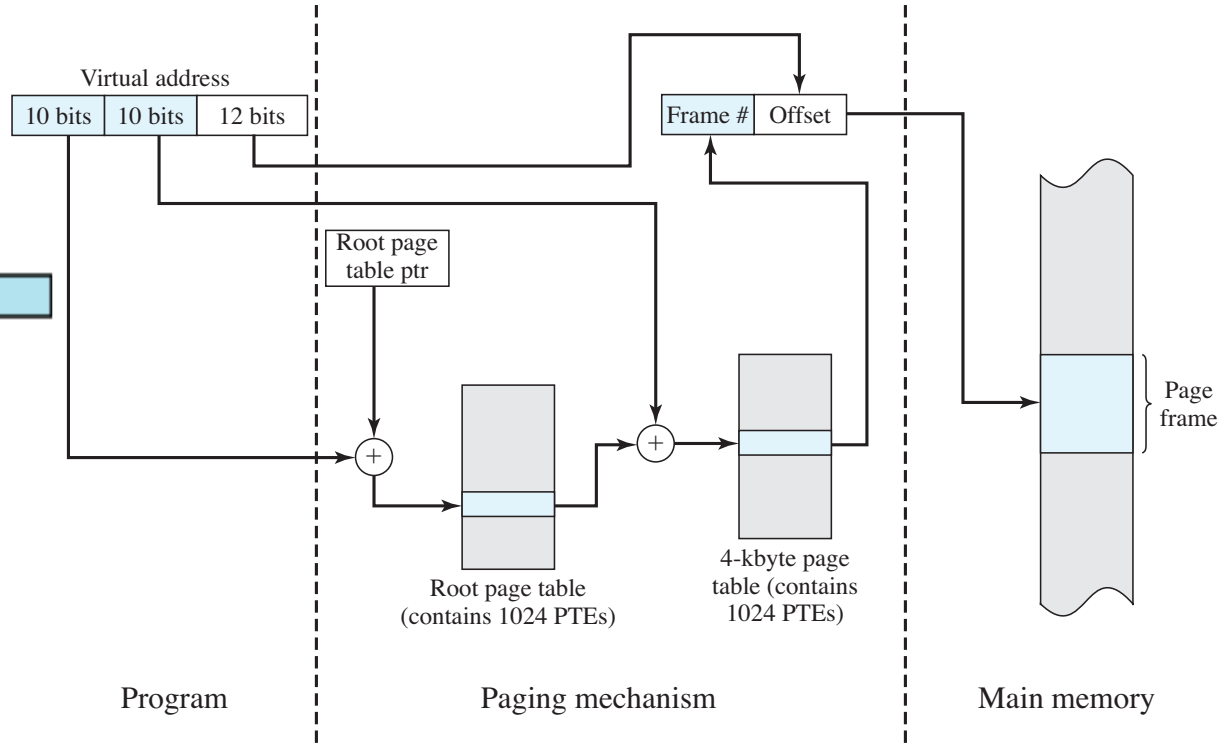


Figure 8.5 **Address Translation in a Two-Level Paging System**

Translation Lookaside Buffer

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed **cache** is set up for page table entries
  - Called a Translation Lookaside Buffer (TLB)
  - Contains most recently used PT Entries

Translation Lookaside Buffer

- Given a virtual address, processor examines the TLB
- If (TLB hit), the frame number is retrieved and the real address is formed
- If (TLB miss), the page number is used to index the process page table; TLB updated for new page entry
- If Present Bit not set => Page Fault =>New page is loaded; PT updated; TLB updated

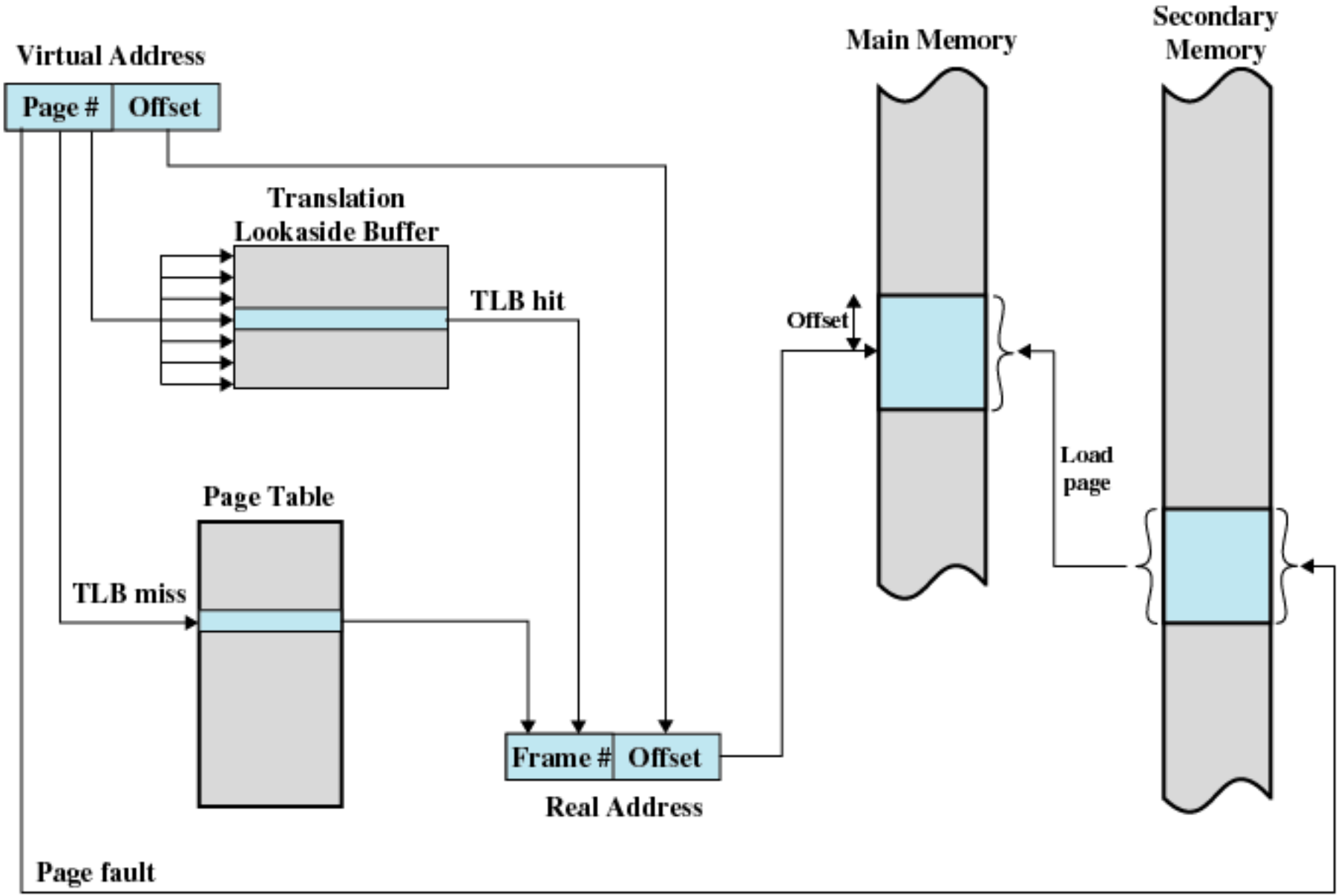


Figure 8.7 Use of a Translation Lookaside Buffer

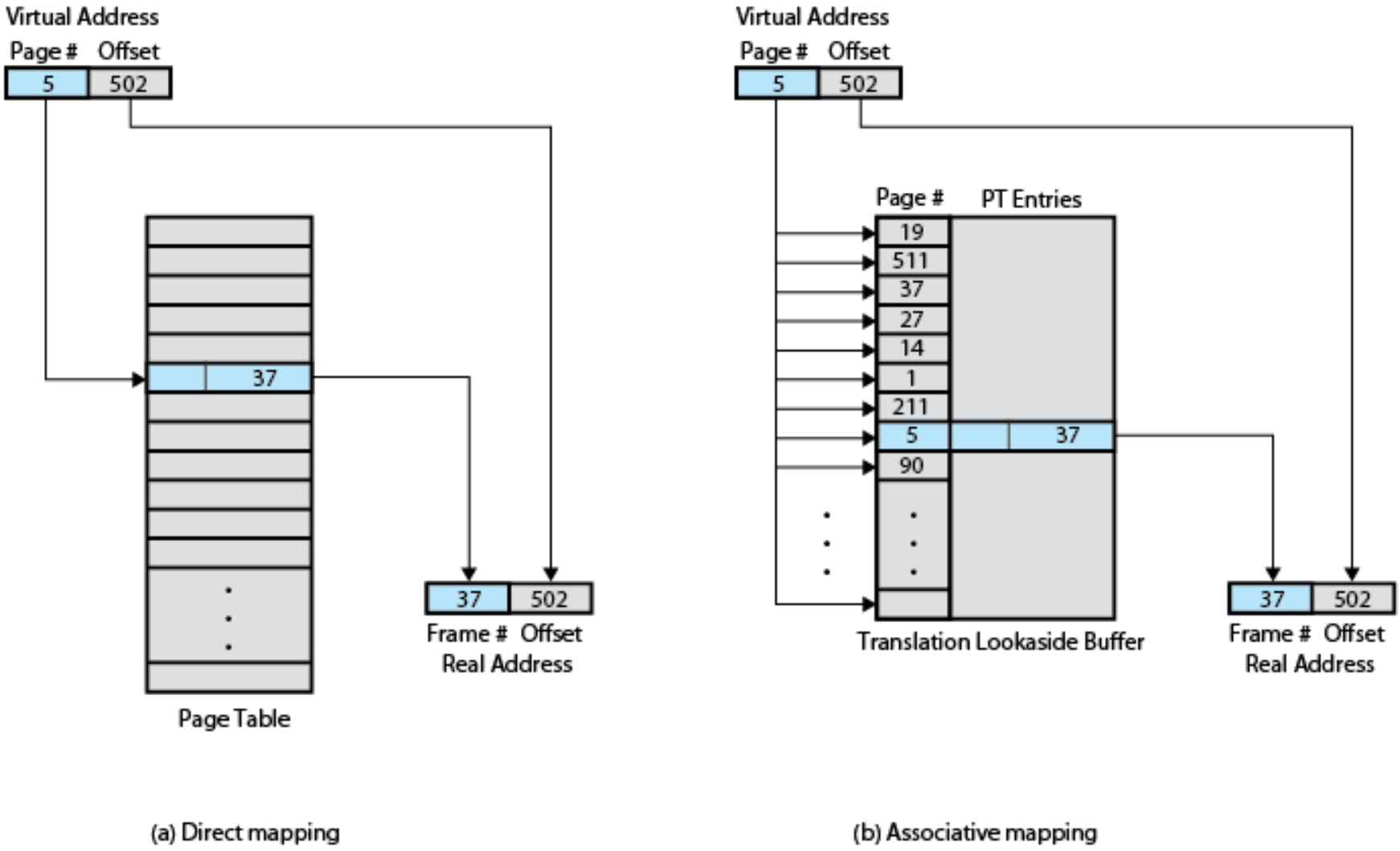
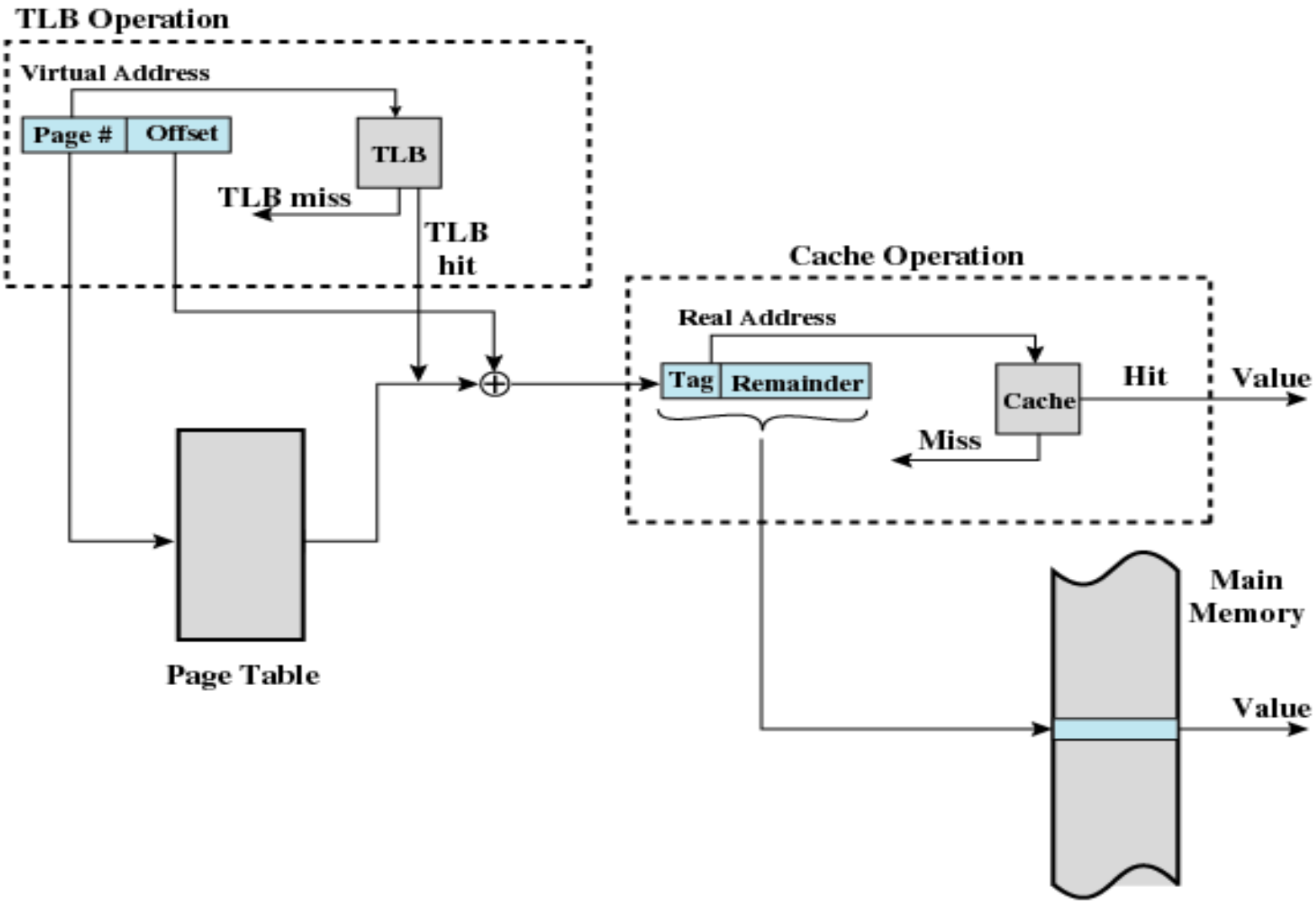


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

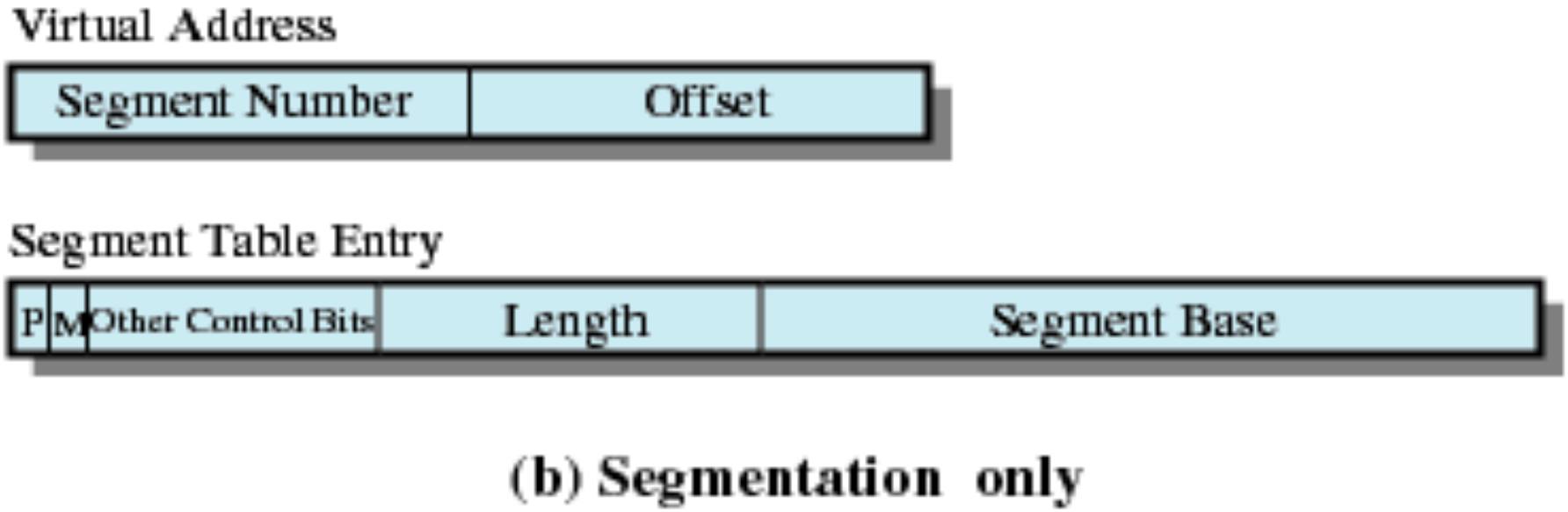
TLB & MemCache Operations



Segment Tables

- Corresponding segment in main memory
- Each entry contains the length of the segment
- A bit is needed to determine if segment is already in main memory
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory

Segment Table Entries



(b) Segmentation only

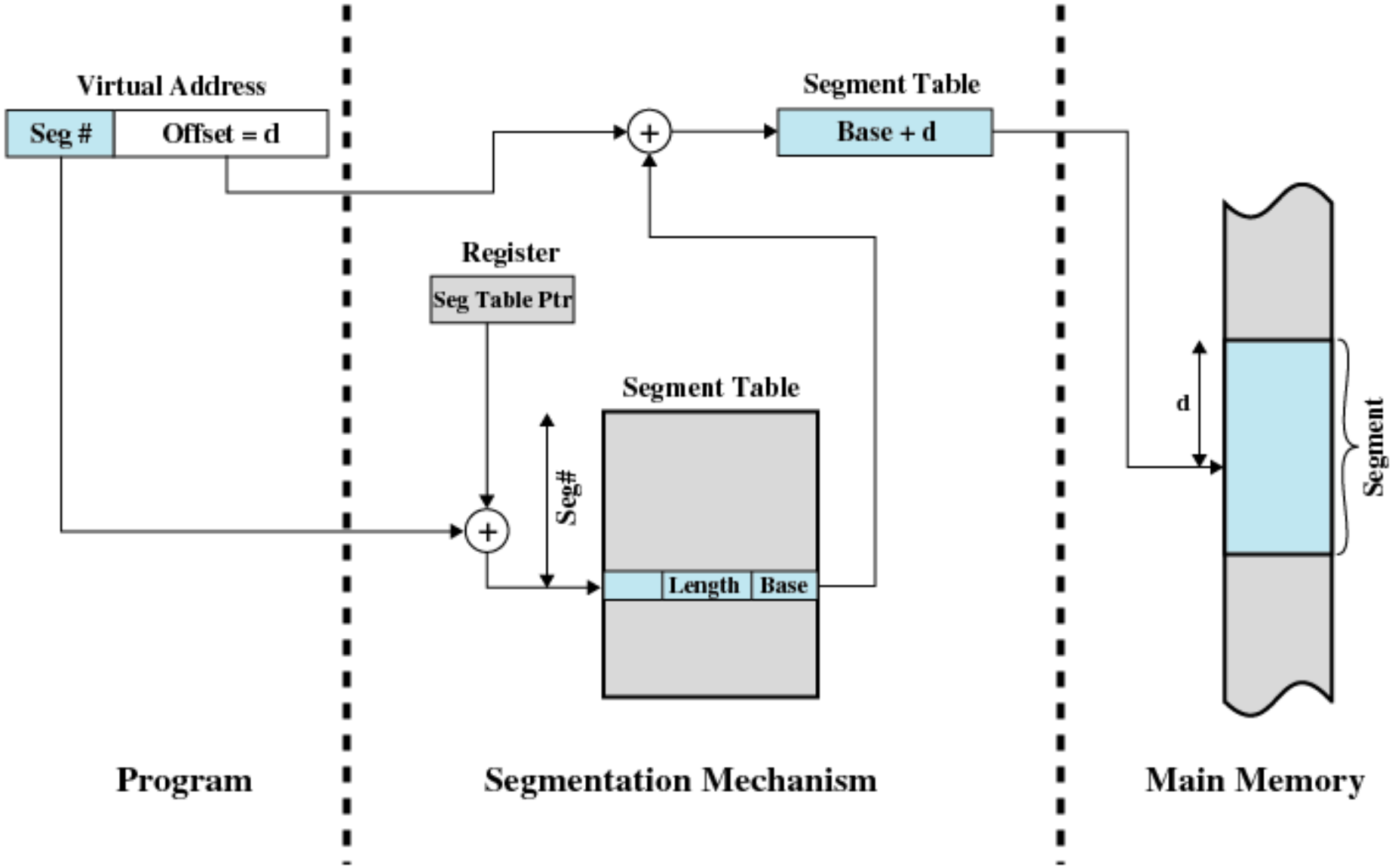
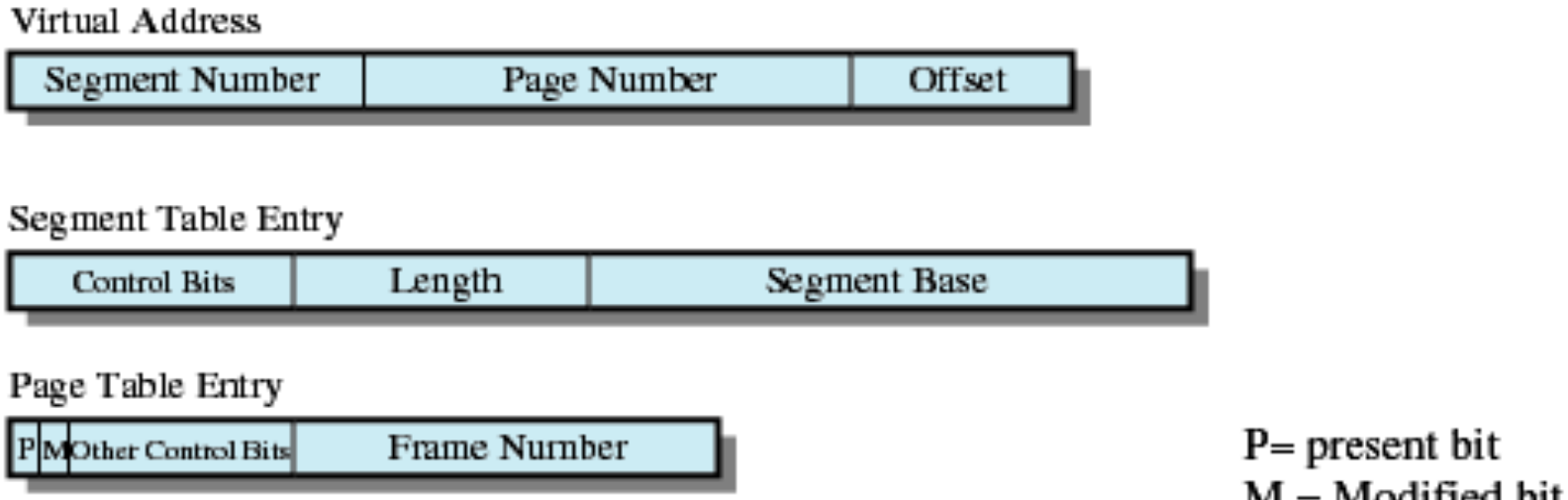


Figure 8.12 Address Translation in a Segmentation System

Combined Paging and Segmentation

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages

Combined Segmentation and Paging



(c) Combined segmentation and paging

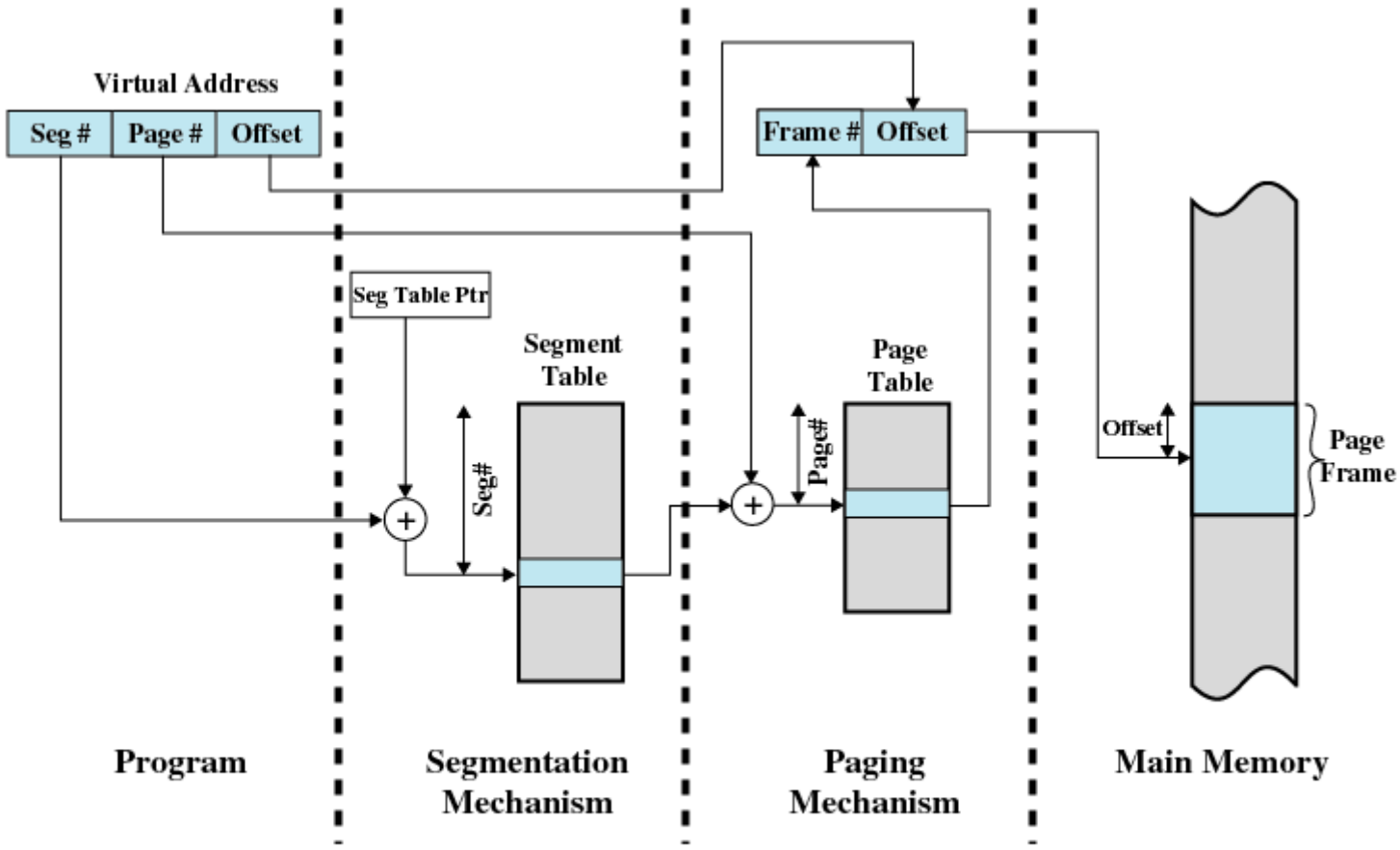


Figure 8.13 Address Translation in a Segmentation/Paging System

Fetch Policy

- Fetch Policy
  - Determines when a page should be brought into memory

  a. Demand Paging: Bring page when referenced
    - Many page faults when process first started

  b. Prepaging: Bring in several continuous pages

Placement Policy

- Determines where in real memory a process piece is to reside
- Important in a segmentation system
- Paging or combined paging with segmentation hardware performs address translation

Replacement Policy

- Placement Policy
  - Which page is replaced?
  - Page removed should be the page least likely to be referenced in the near future
  - Most policies predict the future behavior on the basis of past behavior

Replacement Policy

- Frame Locking
  - If frame is locked, it may not be replaced
  - Kernel of the operating system
  - Control structures
  - I/O buffers
  - Associate a lock bit with each frame (frame table or page table)

Basic Replacement Algorithms

- Optimal policy (Benchmark)
  - Selects for replacement that page for which the time to the next reference is the longest
  - Impossible to have perfect knowledge of future events

Basic Replacement Algorithms

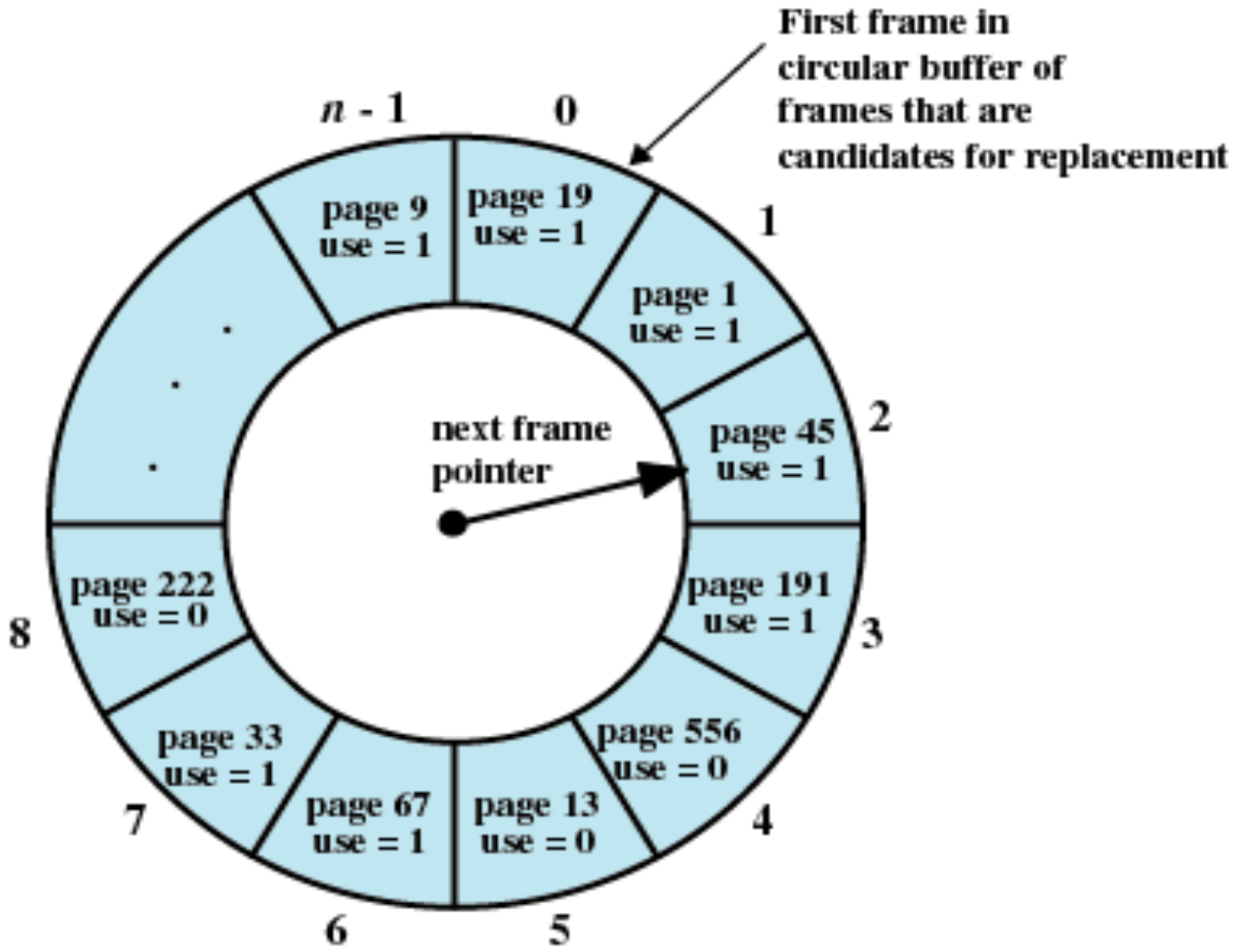
- Least Recently Used (LRU)
  - Replaces the page that has not been referenced for the longest time
  - By the principle of locality, this should be the page least likely to be referenced in the near future
  - Each page could be tagged with the time of last reference = **overhead**

Basic Replacement Algorithms

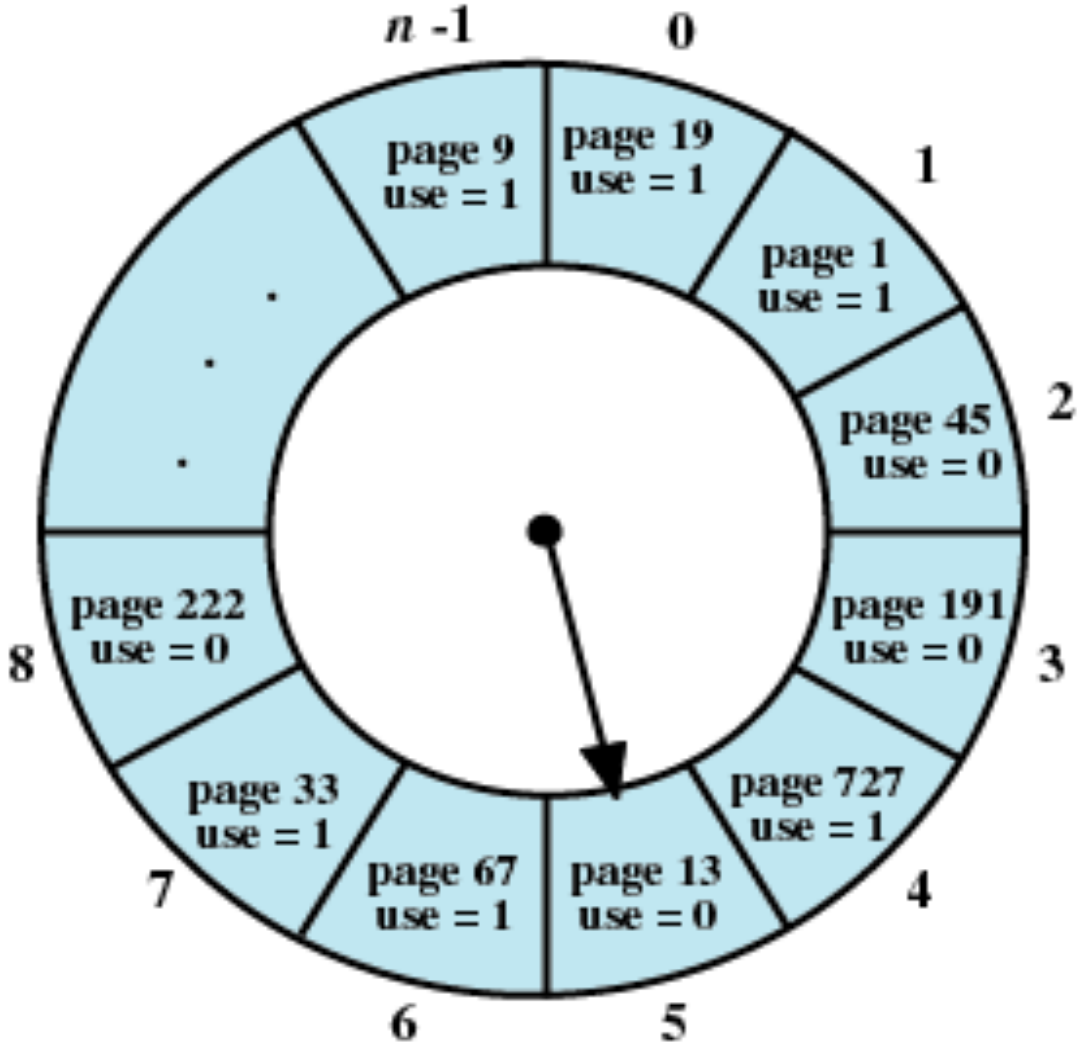
- First-in, first-out (FIFO)
  - Treats page frames allocated to a process as a circular buffer
  - Pages are removed in round-robin style
  - Simplest replacement policy to implement
  - Page that has been in memory the longest is replaced
  - These pages may be needed again very soon

Basic Replacement Algorithms

- Clock Policy
  - Additional bit called a use bit
  - When a page is first loaded in memory, the use bit is set to 1
  - When the page is referenced, the use bit is set to 1
  - When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
  - During the search for replacement, each use bit set to 1 is changed to 0



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

Clock with Used and Modified Bits

- Each frame two bits associated:
  - U: used
  - M: modified

Algorithm:

1) First scan to find (u = 0; m = 0)

2) Second scan to find (u = 0; m = 1); Set u = 0 for all encountered

3) Repeat Step 1;

Basic Replacement Algorithms

- Page Buffering
  - Replaced page PT-entry is added to one of two lists
    - Free page list, if page has not been modified
    - Modified page list, otherwise
  - Free page list used to read-in new pages (overwrite)
  - Modified page list used to write out modified pages in clusters