Operating Systems

Virtual Memory

Chapter 8

Outline

- Hardware and Control Structures
  - Locality and virtual memory
  - Paging
  - Segmentation
  - Combined Paging and Segmentation
  - Protection and sharing

- Operating System Software
  - Fetch policy
  - Placement policy
  - Replacement policy
  - Resident set management
  - Cleaning policy
  - Load Control

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 (memory access fault) 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

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
- We need to avoid thrashing
  - Which pieces are least likely to be used in the near future?

Principle of Locality

- Program and data references within a process tend to cluster (Fig. 8.1, page 338)
- 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/or 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 (the P bit) is needed to indicate whether the page is in main memory or not

Modify Bit in Page Table

- Another modify bit (the M 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

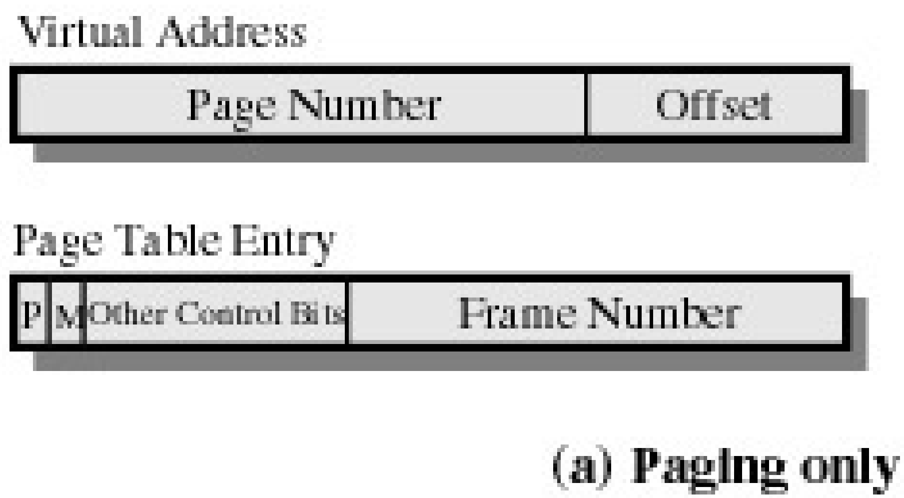


Figure 8.2 Typical Memory Management Formats

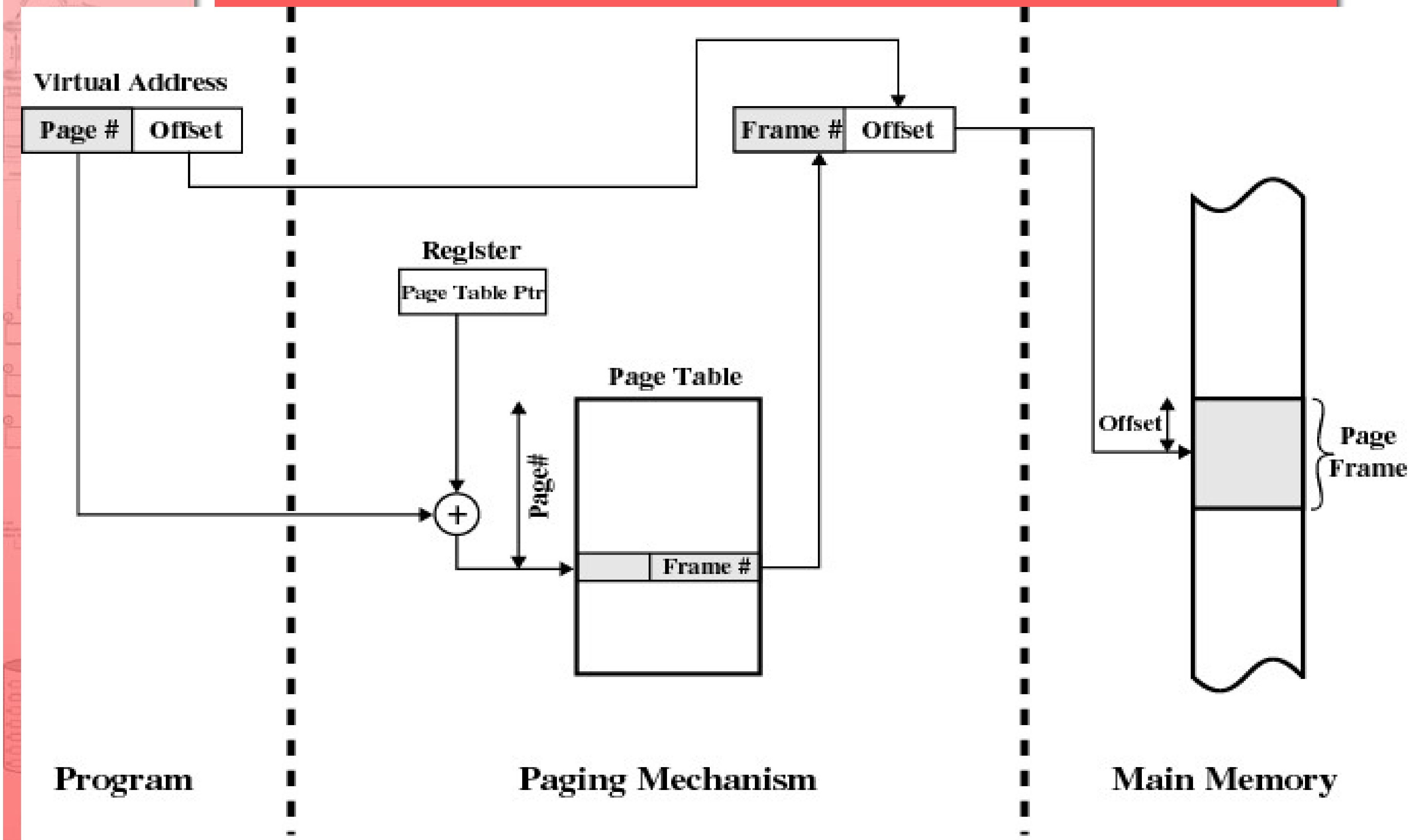


Figure 8.3 Address Translation in a Paging System

Page Tables

- The entire page table may take up too much main memory
- Page tables are also stored in virtual memory (see page 340)
- When a process is running, part of its page table is in main memory

In such manner, page tables are subject to paging too!

Translation Lookaside Buffer

- Each virtual memory reference can cause *two* physical memory accesses
  - one to fetch the page table entry
  - one to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
  - called the TLB - Translation Lookaside Buffer

Translation Lookaside Buffer

- Contains page table entries that have been most recently used
- Functions same way as a memory cache

Translation Lookaside Buffer

- Given a virtual address, processor examines the TLB
- If page table entry is present (a hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (a miss), the page number is used to index the process page table

Translation Lookaside Buffer

- First checks if page is already in main memory
  - if not in main memory a *page fault* is issued
- The TLB is updated to include the new page entry

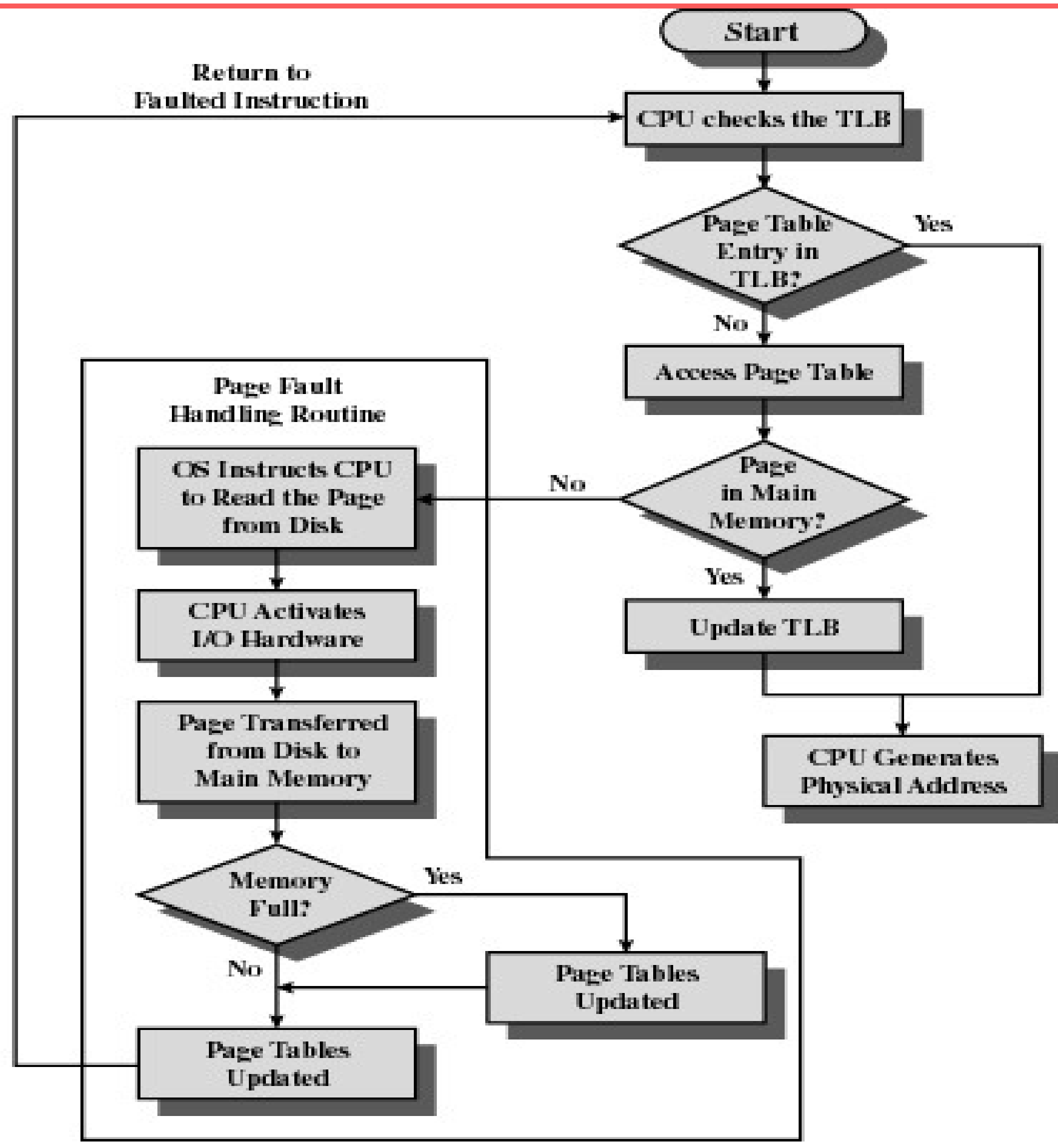


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

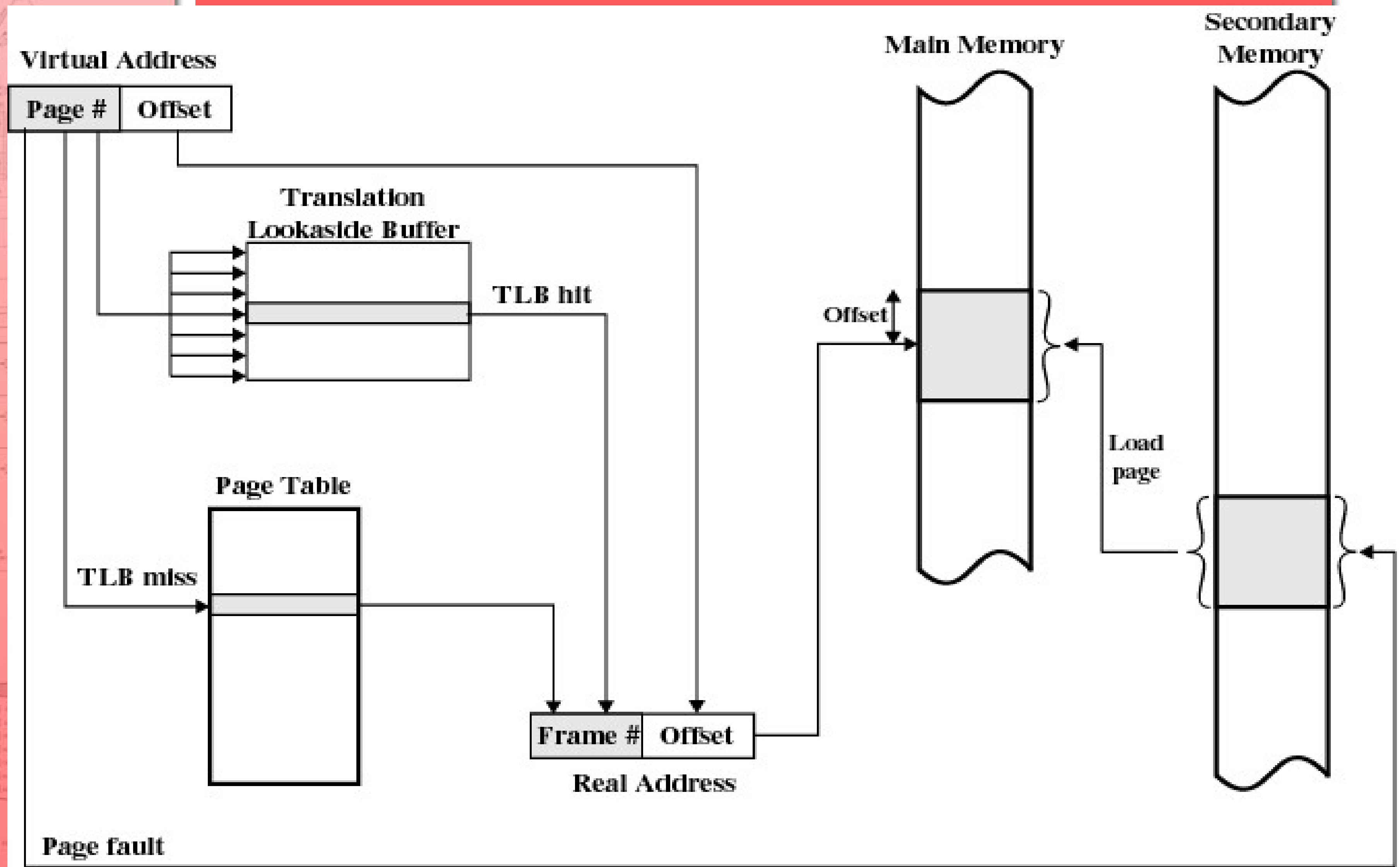


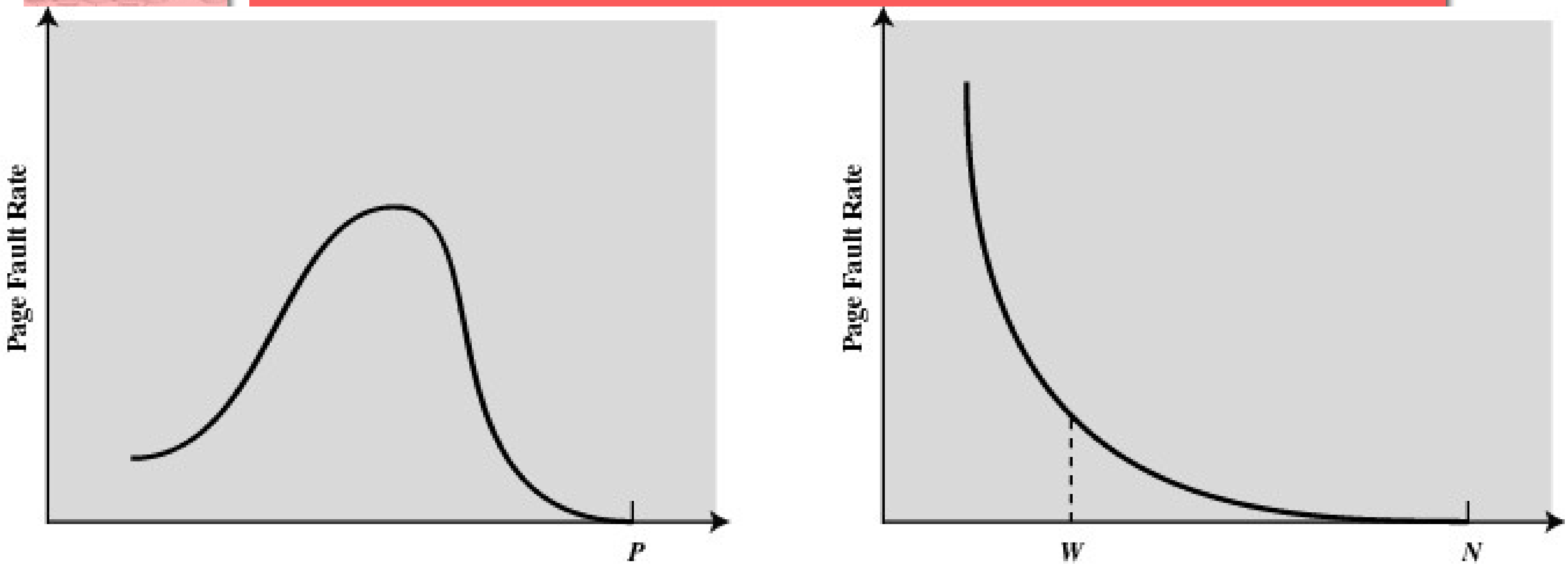
Figure 8.7 Use of a Translation Lookaside Buffer

Page Size

- Smaller page size, less amount of internal fragmentation
- Smaller page size, more pages required per process
- More pages per process means larger page tables
- Larger page tables means large portion of page tables in virtual memory
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Page Size

- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references → Page faults low.
- Increased page size causes pages to contain locations further from any recent reference→ Page faults rise.



(a) Page Size

(b) Number of Page Frames Allocated

P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.11 Typical Paging Behavior of a Program

Page Size

- Multiple page sizes provide the flexibility needed to effectively use a TLB
- Large pages can be used for program instructions
- Small pages can be used for threads
- *Most operating system support only one page size*

Example Page Sizes

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 kbyes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
PowerPc	4 Kbytes

Segmentation

- May be unequal, dynamic size
- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently
- Lends itself to sharing data among processes
- Lends itself to protection

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

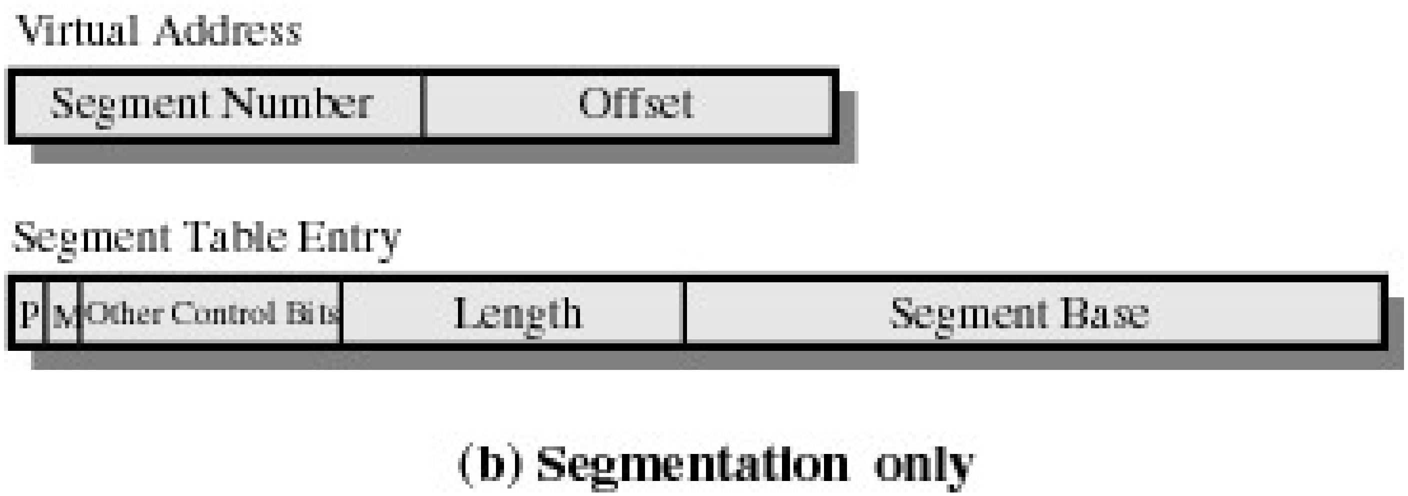


Figure 8.2 Typical Memory Management Formats

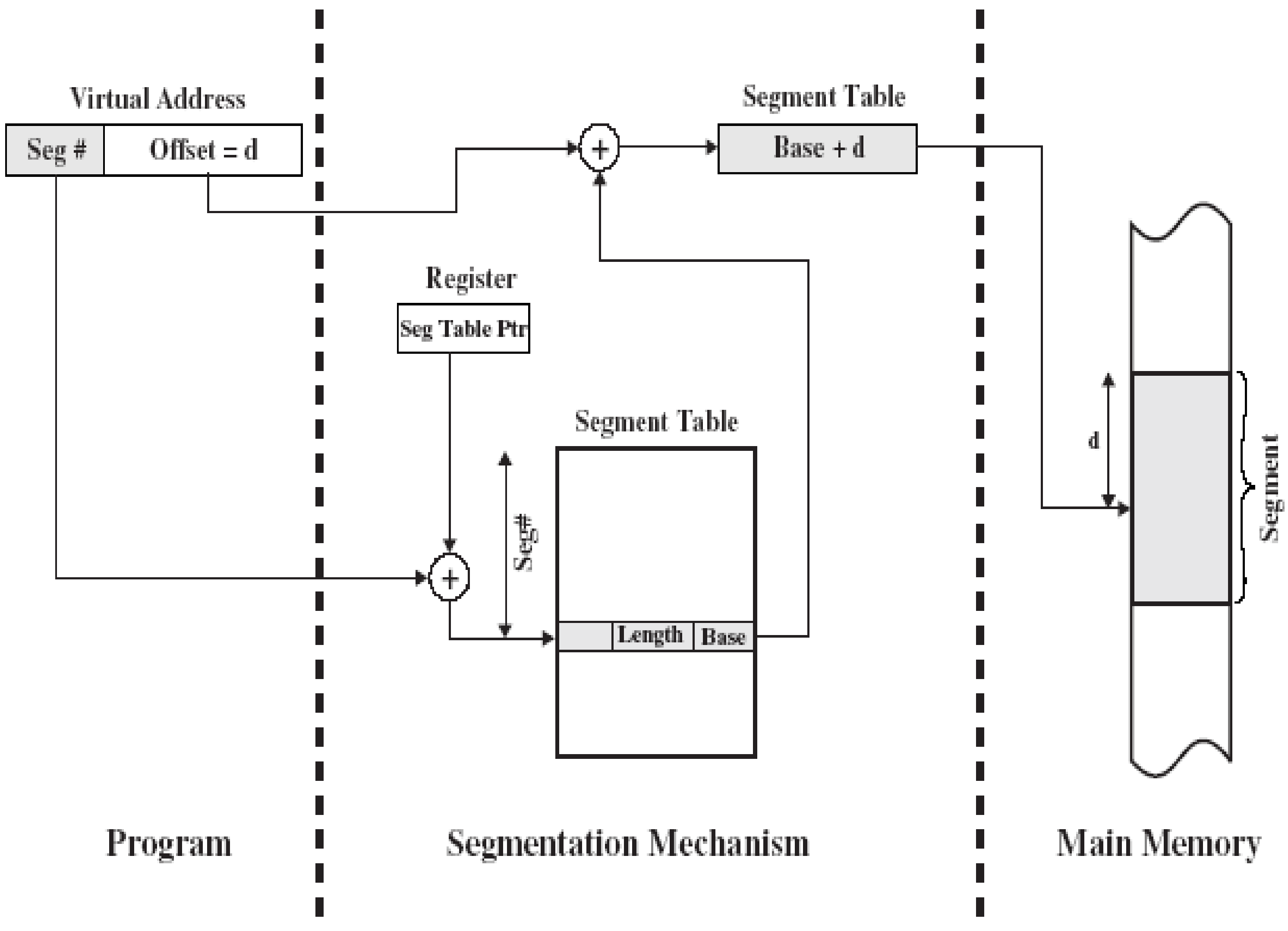


Figure 8.12 Address Translation in a Segmentation System

Combined Paging and Segmentation

- Paging is transparent to the programmer
- Paging eliminates external fragmentation
- Segmentation is visible to the programmer
- Segmentation allows for growing data structures, modularity, and support for sharing and protection
- Each segment is broken into fixed-size pages

Combined Segmentation and Paging

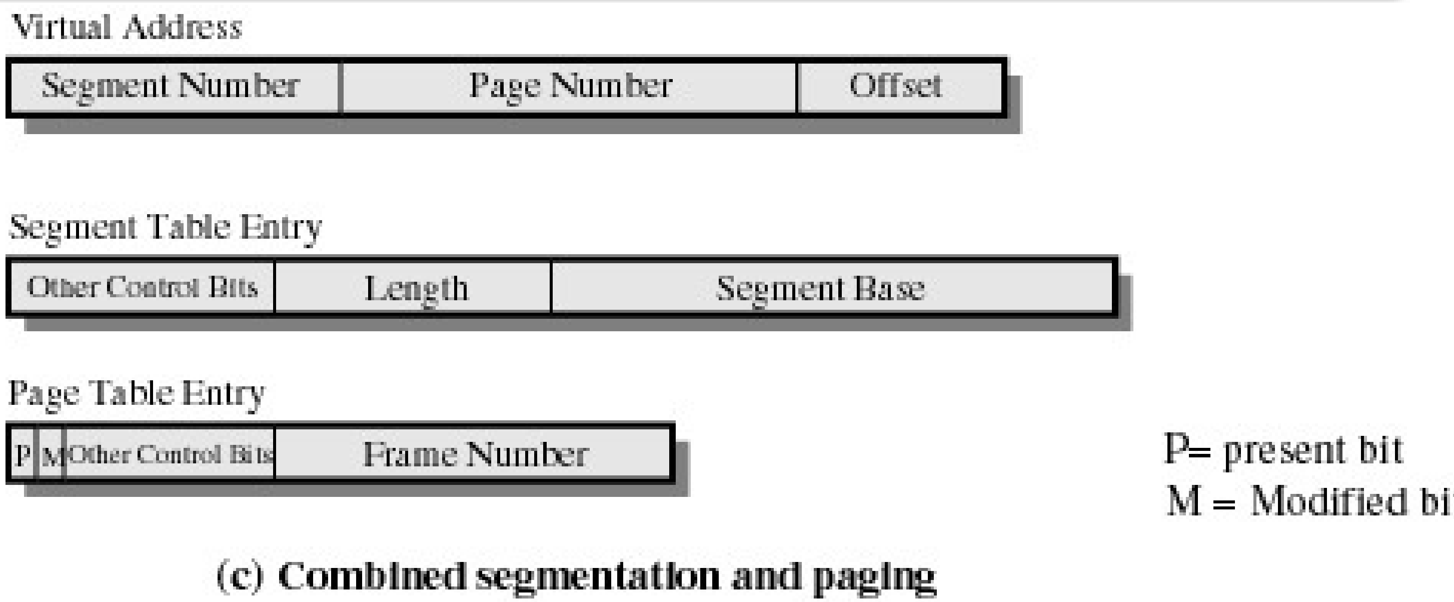


Figure 8.2 Typical Memory Management Formats

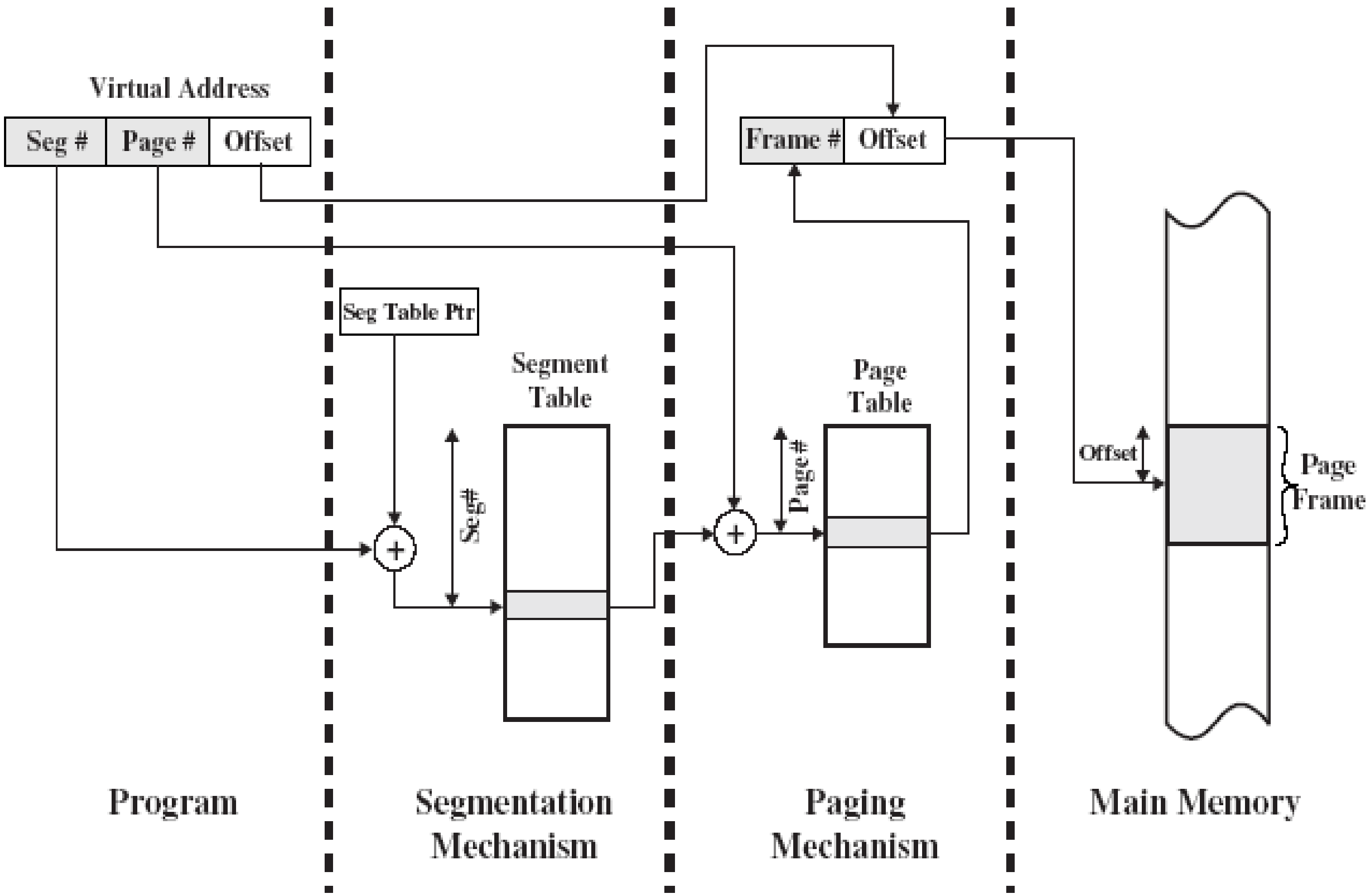


Figure 8.13 Address Translation in a Segmentation/Paging System

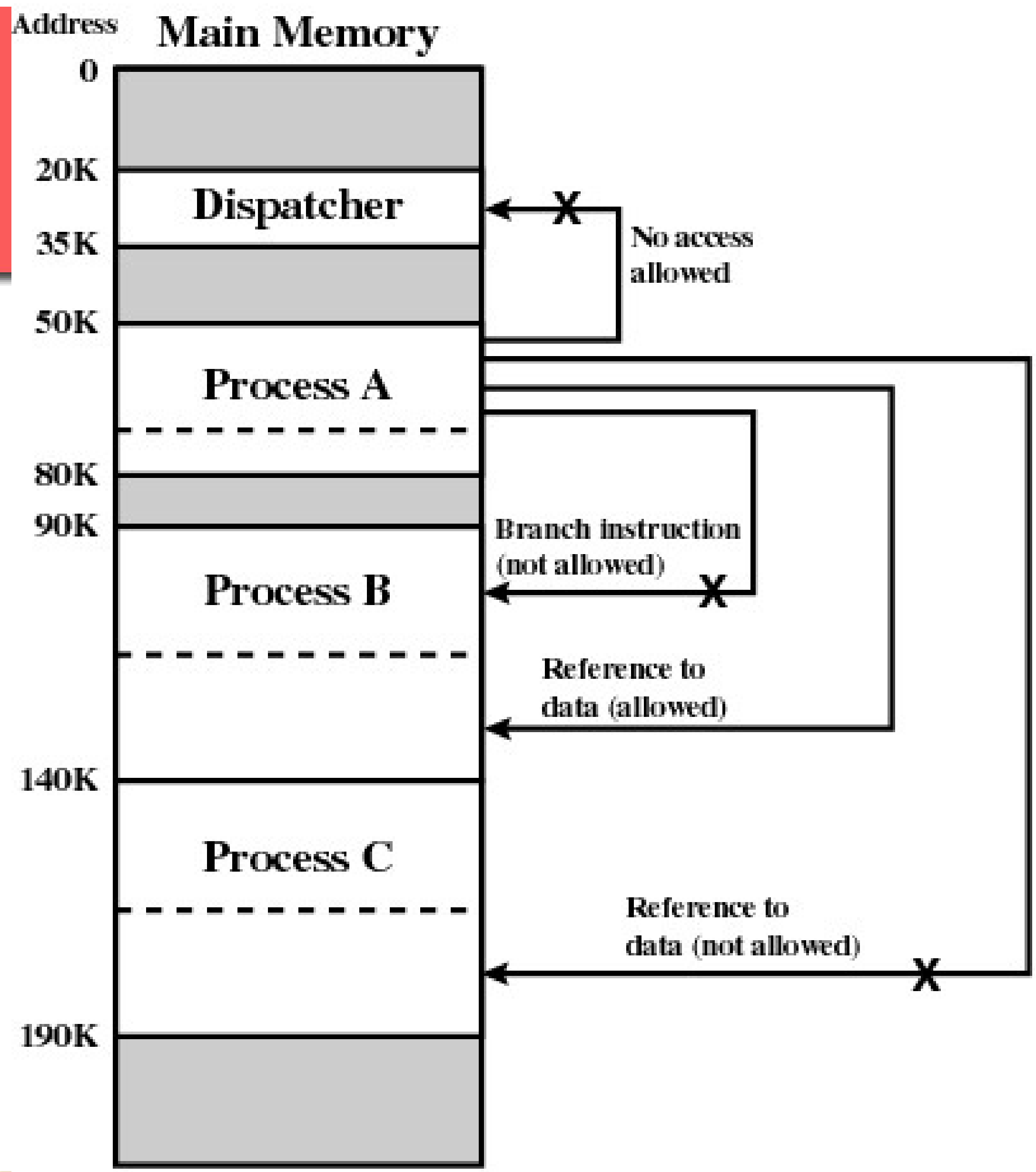


Figure 8.14 Protection Relationships Between Segments

OS Software

- Fetch Policy
- Placement Policy
- Replacement Policy
- Resident Set Management
- Cleaning Policy
- Load Control

Fetch Policy

- Fetch Policy
  - Determines when a page should be brought into memory
  - Demand paging only brings pages into main memory when a reference is made to a location on the page
    - Many page faults when process first started
  - Prepaging brings in more pages than needed
    - More efficient to bring in pages that reside contiguously on the disk

Placement Policy

- Where in real memory a process piece is to reside?
- Pure Segmentation → best-fit, first-fit, and so on (see chapter 7)
- Pure paging or combined with segmentation → *placement irrelevant*, since address translation HW and main memory access HW can perform their functions for any page-frame combination

Replacement 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

Basic Replacement Algorithms

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

Fixed frame allocation of 3 frames

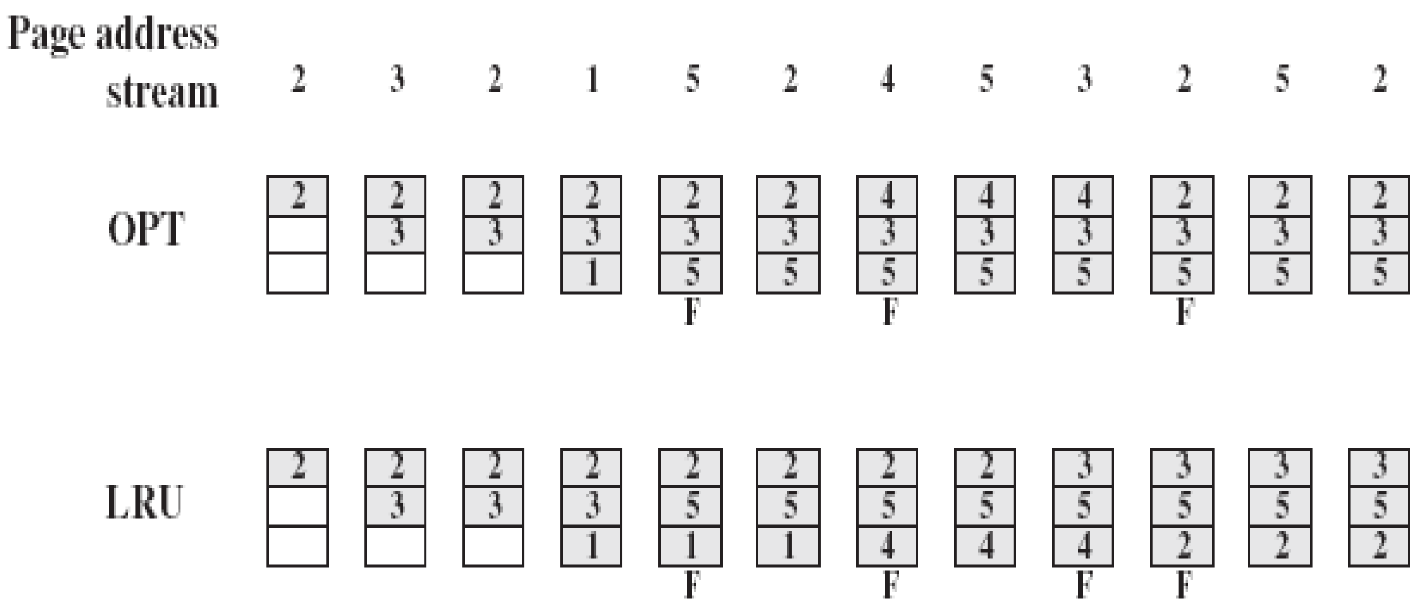
Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		

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. This would require a great deal of overhead.

Basic Replacement Algorithms

- Least Recently Used (LRU)



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

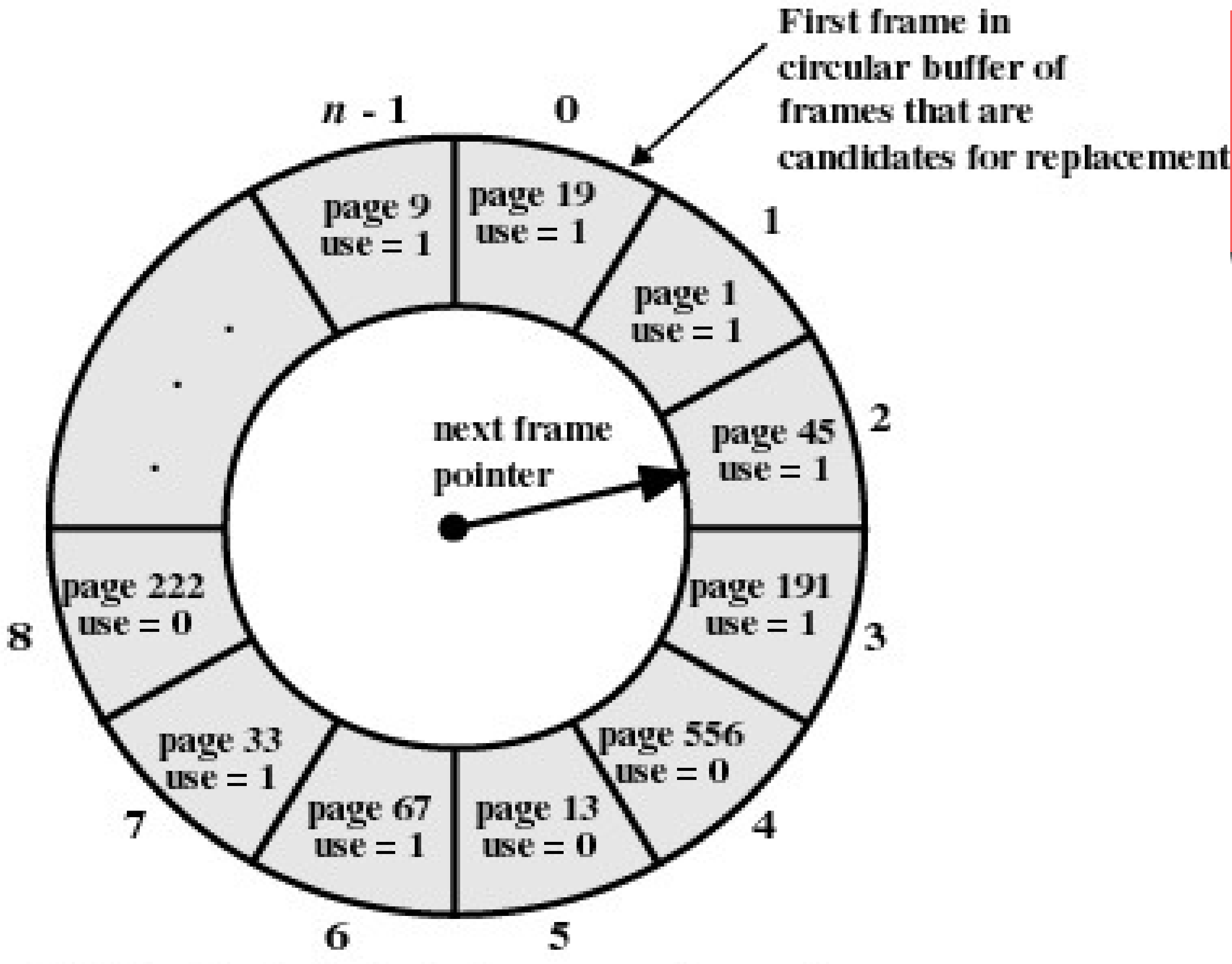
Basic Replacement Algorithms

- First-in, first-out (FIFO)

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5 F	2 3 5	4 3 5 F	4 3 5	4 3 5	2 3 5 F	2 3 5	2 3 5
LRU	2	2 3	2 3	2 3 1	2 5 1 F	2 5 1	2 5 4 F	2 5 4	3 5 4 F	3 5 2 F	3 5 2	3 5 2
FIFO	2	2 3	2 3	2 3 1	5 3 1 F	5 2 1 F	5 2 4 F	5 2 4	3 2 4 F	3 2 4	3 5 4 F	3 5 2 F

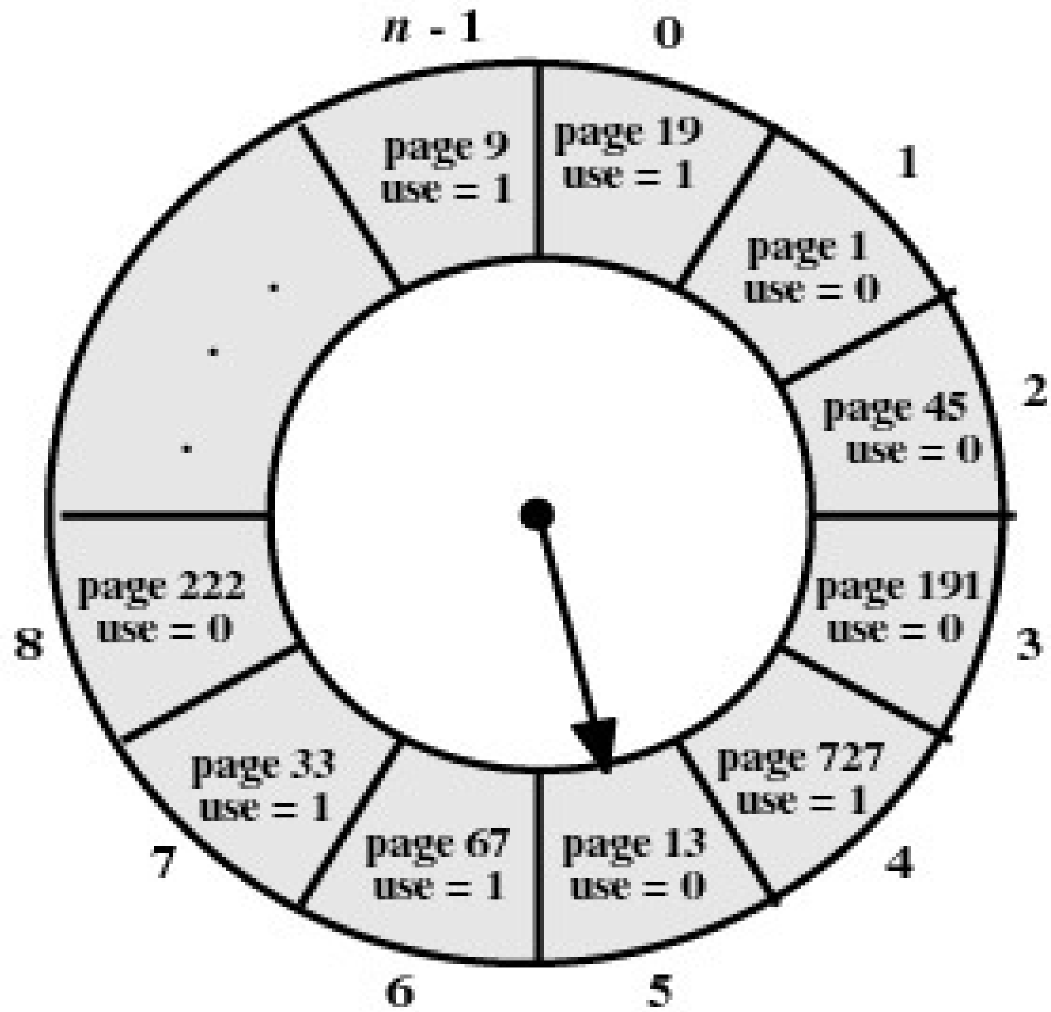
Basic Replacement Algorithms

- Clock Policy
  - Additional bit called a use bit
  - When a page is first loaded in memory, the use bit is set to 0
  - 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
  - If all the frames have a use bit of 1, the pointer will make one complete cycle through the buffer, setting all use bits to zero, and stop at original position, replacing page in the frame
  - Frames candidate for replacement → circular buffer with a pointer. When a page is replaced, the pointer is set to indicate the next frame in the buffer



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

Figure 8.16 Example of Clock Policy Operation



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

Figure 8.16 Example of Clock Policy Operation

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5	2 3 5	4 3 5	4 3 5	4 3 5	2 3 5	2 3 5	2 3 5
					F		F			F		
LRU	2	2 3	2 3	2 3 1	2 5 1	2 5 1	2 5 4	2 5 4	3 5 4	3 5 2	3 5 2	3 5 2
					F		F		F	F		
FIFO	2	2 3	2 3	2 3 1	5 3 1	5 2 1	5 2 4	5 2 4	3 2 4	3 2 4	3 5 4	3 5 2
					F	F	F		F		F	F
CLOCK	2*	2* 3*	2* 3*	2* 3* 1*	5* 3 1	5* 2* 1	5* 2* 4*	5* 2* 4*	3* 2 4	3* 2* 4	3* 2 5*	3* 2* 5*
					F	F	F		F		F	

*: use bit is 1 arrow: current position of the pointer

Basic Replacement Algorithms

- Please read Page Buffering algorithm on page 361
  - Representative is VAX VMS
  - Simple FIFO
  - In addition, a replaced page is not lost
  - Replaced page is added to one of two lists
    - free page list if page has not been modified
    - modified page list

Resident Set Size

- Fixed-allocation
  - gives a process a fixed number of pages within which to execute
  - when a page fault occurs, one of the pages of that process must be replaced
- Variable-allocation
  - number of pages allocated to a process varies over the lifetime of the process

Replacement Scope

- Local replacement policy
  - Chooses from resident pages of the process that generated the page fault
- Global replacement policy
  - Consider all unlocked pages as candidates for replacement

Variable Allocation, Global Scope

- Easiest to implement
- Adopted by many operating systems
- Operating system keeps list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no free frame, replaces one from another process

Variable Allocation, Local Scope

- When new process added, allocate number of page frames based on application type, program request, or other criteria
- When page fault occurs, select page from among the resident set of the process that suffers the fault
- Reevaluate allocation from time to time

Cleaning Policy

When a modified page should be written out to secondary memory?

- Demand cleaning
  - a page is written out only when it has been selected for replacement
- Precleaning
  - pages are written out in batches

Cleaning Policy

- Best approach uses page buffering
  - Replaced pages are placed in two lists
    - Modified and unmodified
  - Pages in the modified list are periodically written out in batches
  - Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Load Control

- Determines the number of processes that will be resident in main memory (multiprogramming level)
- Too few processes → many occasions when all processes will be blocked and much time will be spent in swapping
- Too many processes will lead to thrashing

Process Suspension

To reduce the degree of multiprogramming, one or more of the current resident processes must be suspended, but which?

- Lowest priority process
- Faulting process
  - this process does not have its working set in main memory so it will be blocked anyway
- Last process activated
  - this process is least likely to have its working set resident

Process Suspension

- Process with smallest resident set
  - this process requires the least future effort to reload
- Largest process
  - obtains the most free frames
- Process with the largest remaining execution window