Chapter 8 - Lecture

Stallings 9e

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 (page fault)
- 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 (or physical) memory
  - Main memory
- Logical memory
  - what the processor sees
- 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

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

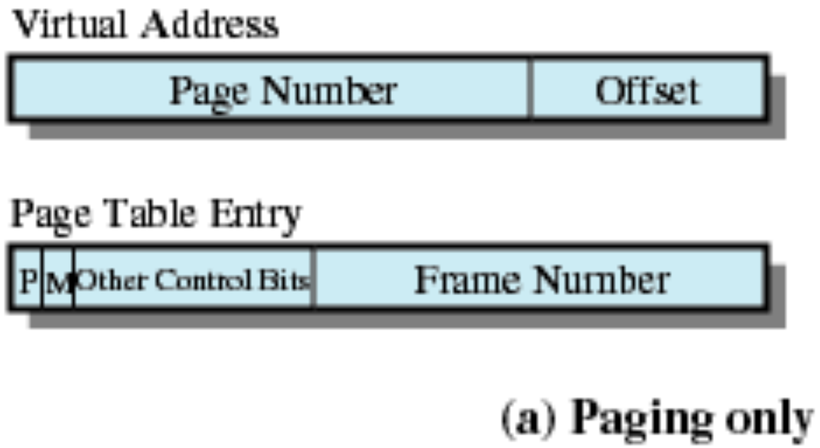
Support Needed for Virtual Memory

- Hardware must support paging and/or 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

Paging



(a) Paging only

P : present
M: modified

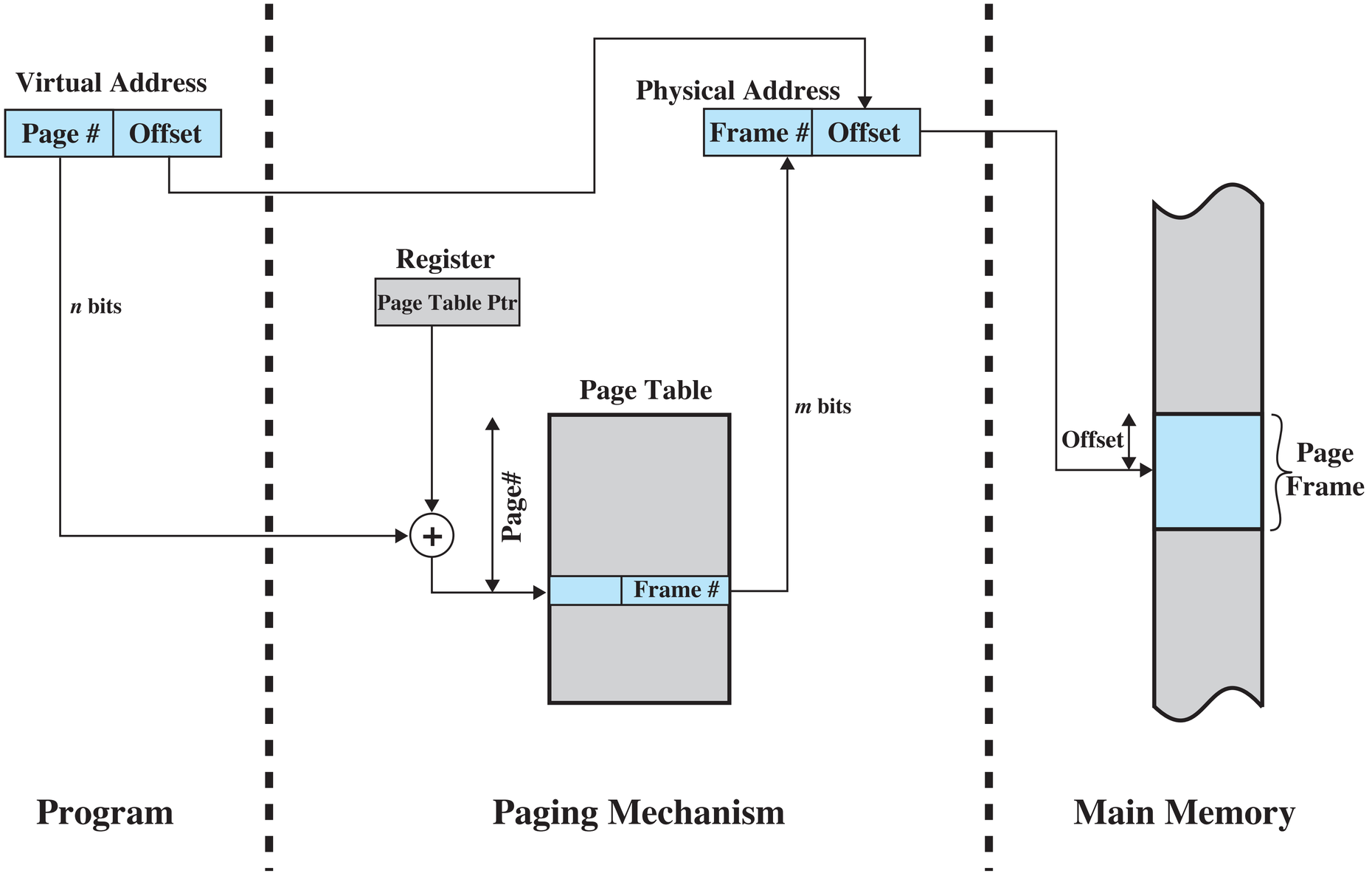


Figure 8.2 Address Translation in a Paging System

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

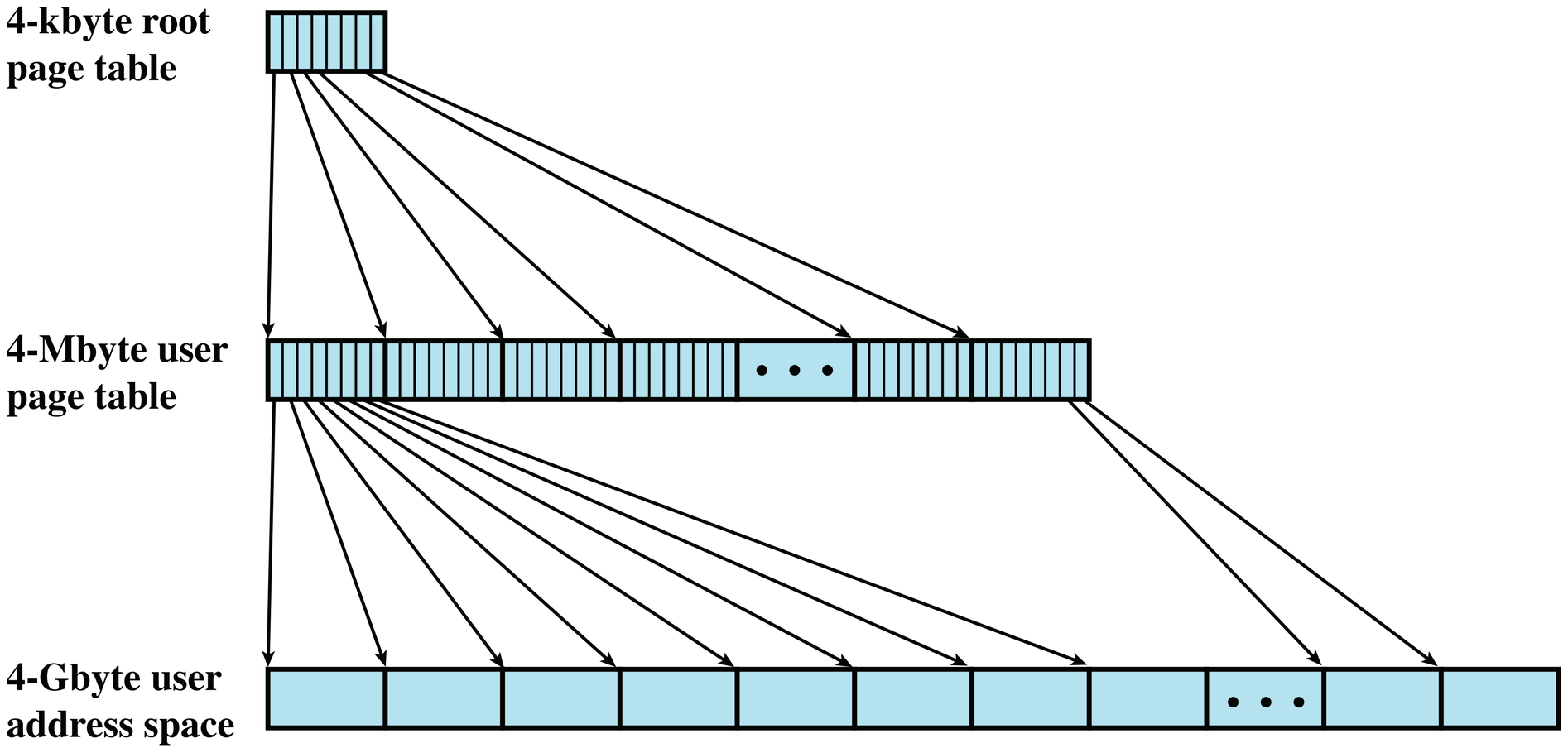


Figure 8.3 A Two-Level Hierarchical Page Table

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

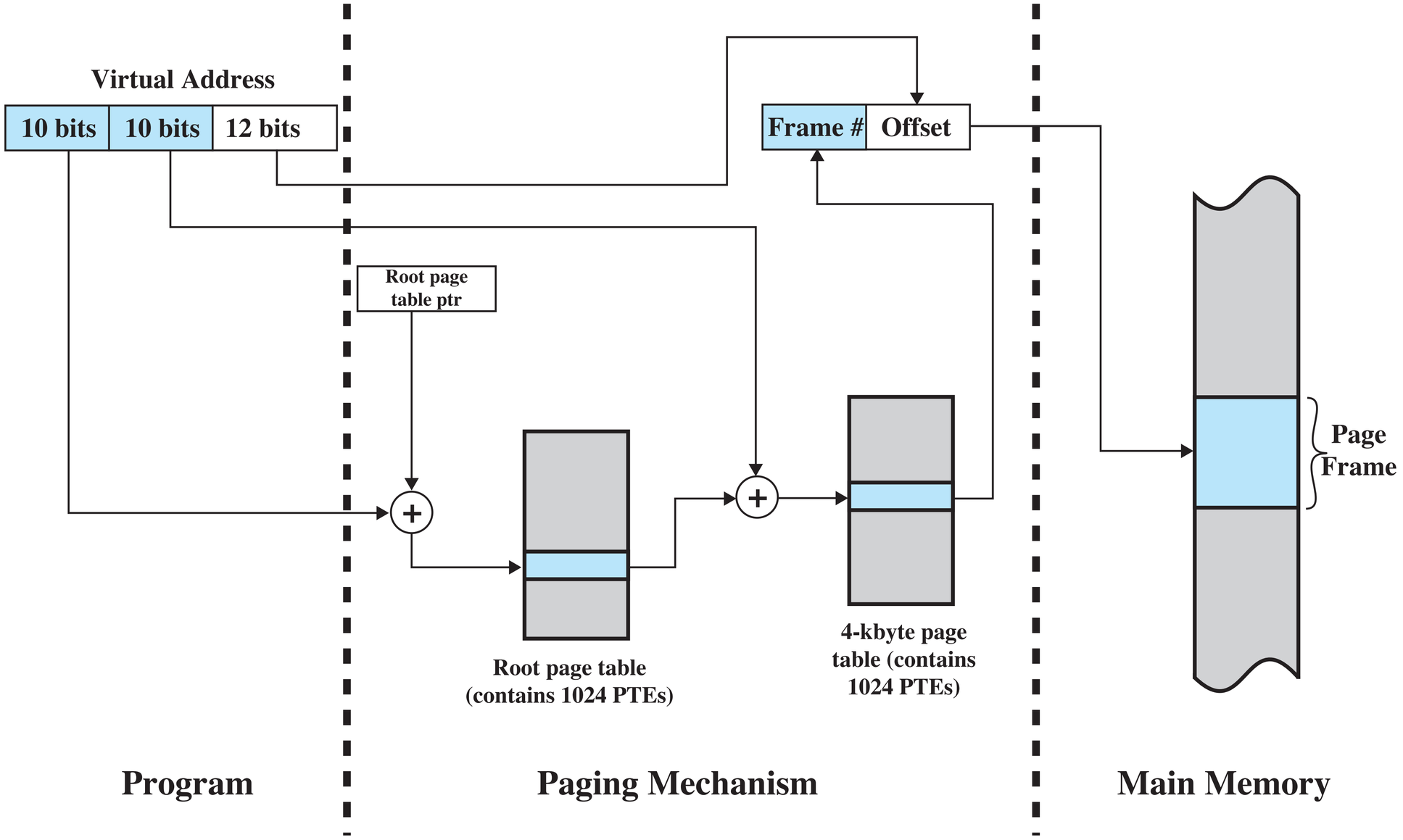


Figure 8.4 Address Translation in a Two-Level Paging System

Inverted Page Table

- Alternative to (multi-level) page table
  - Used on PowerPC, UltraSPARC, and IA-64 architecture
  - One entry in table for each physical memory frame
  - Page number portion of a virtual address is mapped to a hash value
  - Fixed proportion of real memory is required for the tables regardless of the number of processes

Inverted Page Table

- Page table entries:
  - Page number
  - Process identifier
  - Control bits
  - Chain pointer

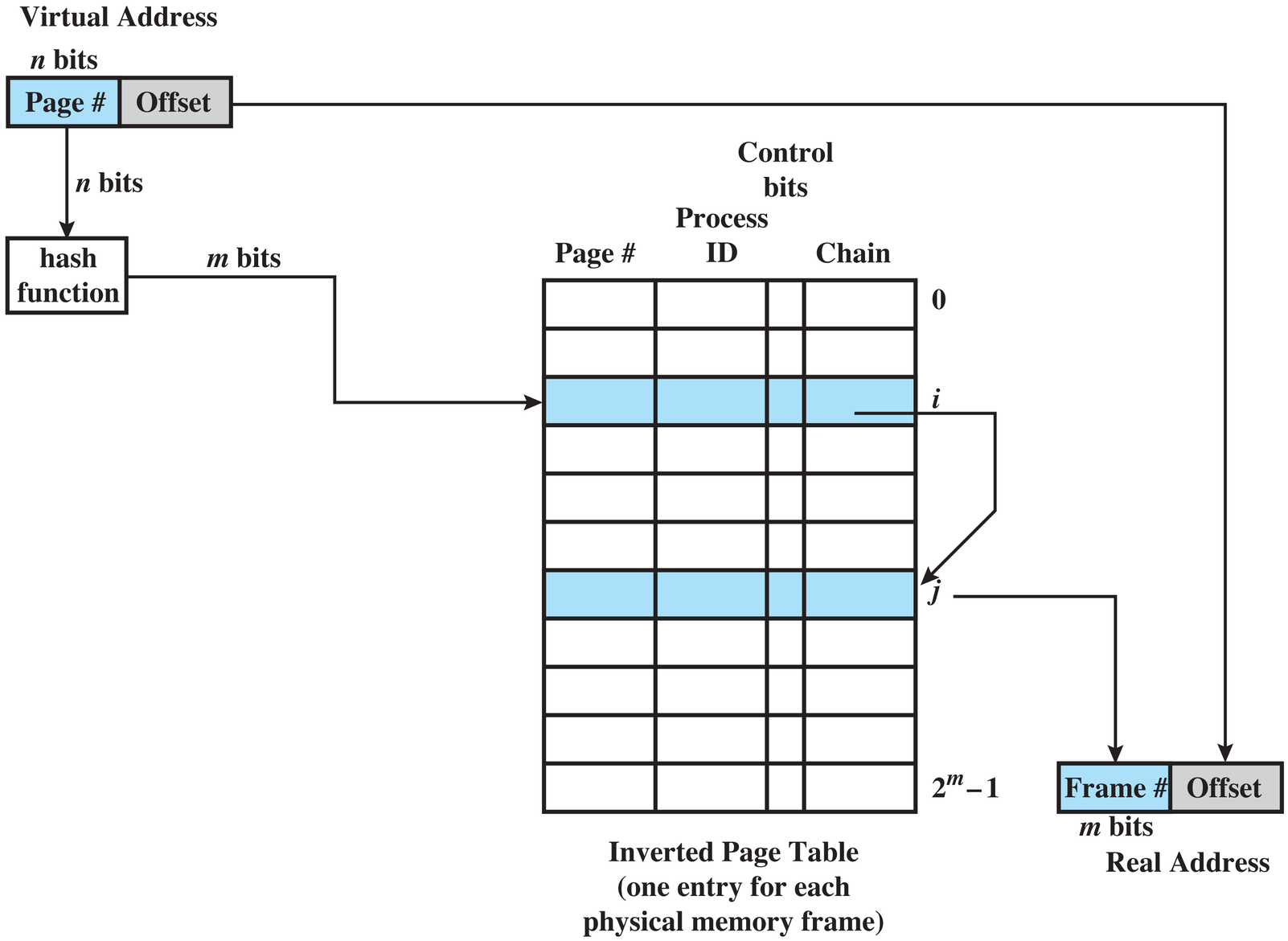


Figure 8.5 Inverted Page Table Structure

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 Translation Lookaside Buffer (TLB)
- Contains page table entries that have been most recently used

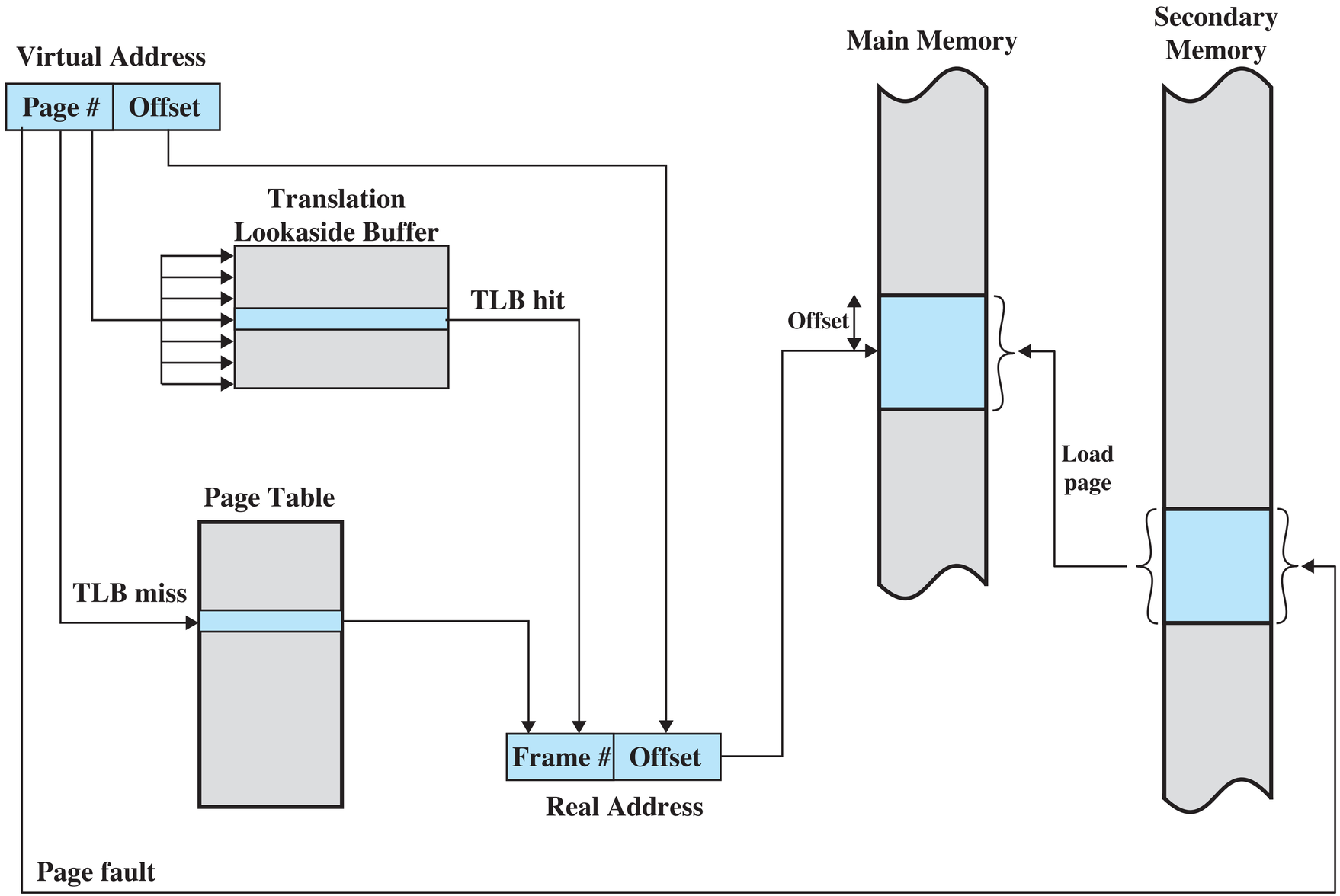


Figure 8.6 Use of a Translation Lookaside Buffer

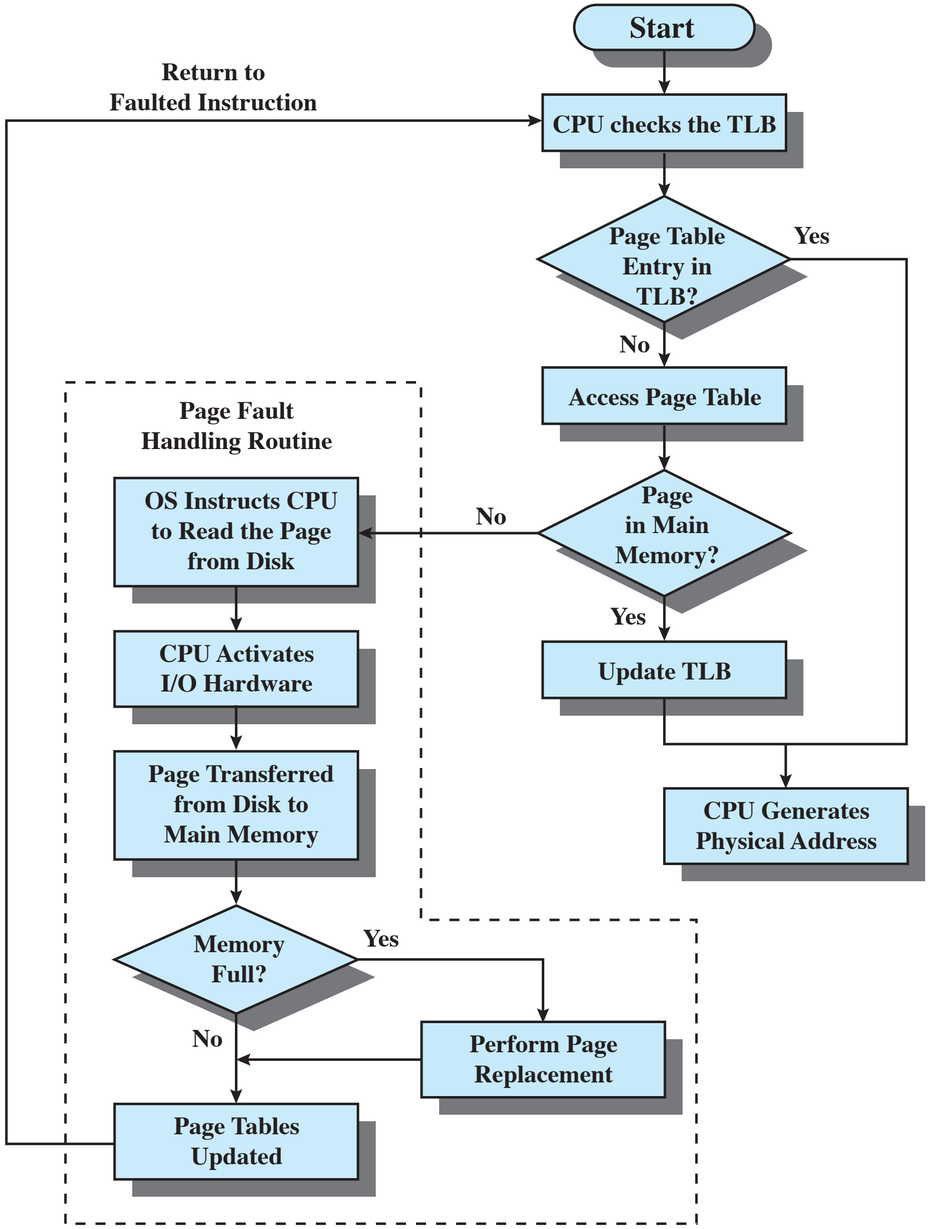


Figure 8.7 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

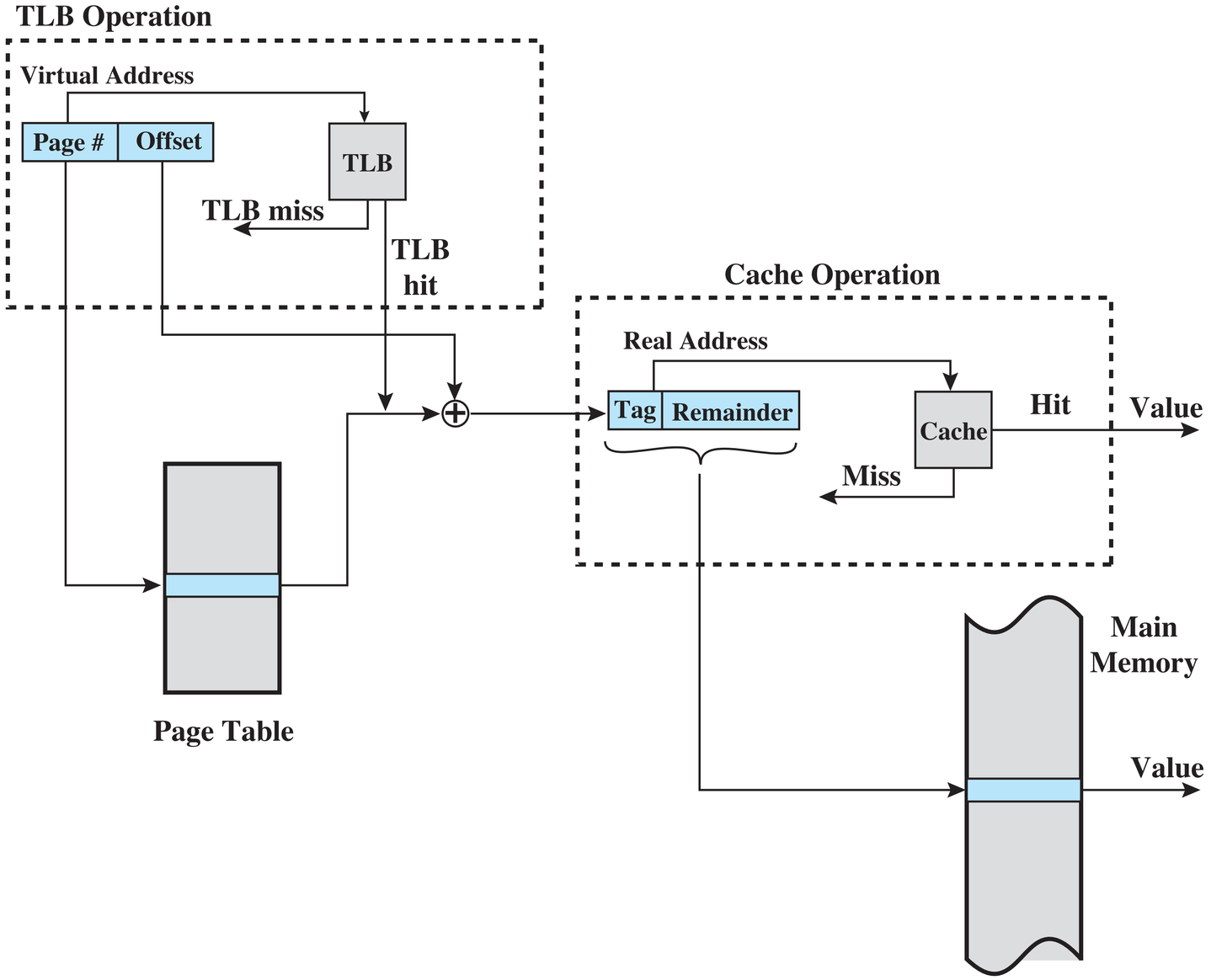


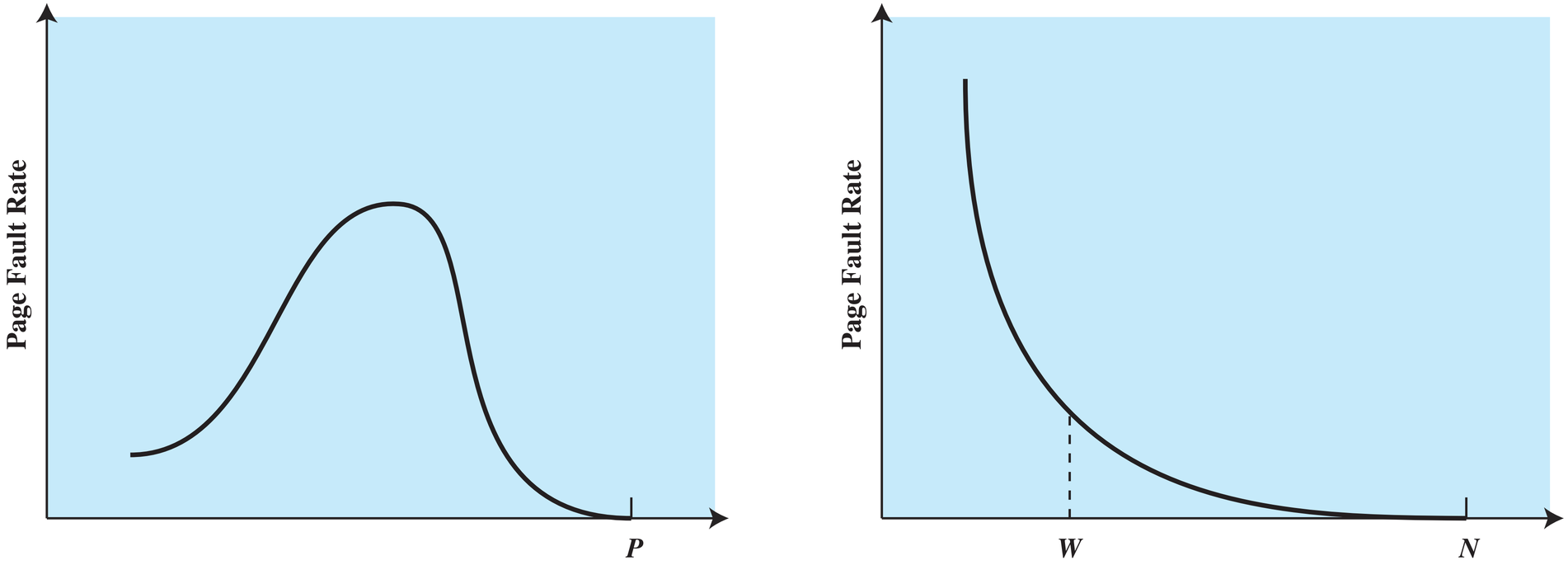
Figure 8.9 Translation Lookaside Buffer and Cache Operation

Page Size - A Tradeoff Space

- 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.



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

Figure 8.10 Typical Paging Behavior of a Program

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
Itanium	4 Kbytes to 256 Mbytes

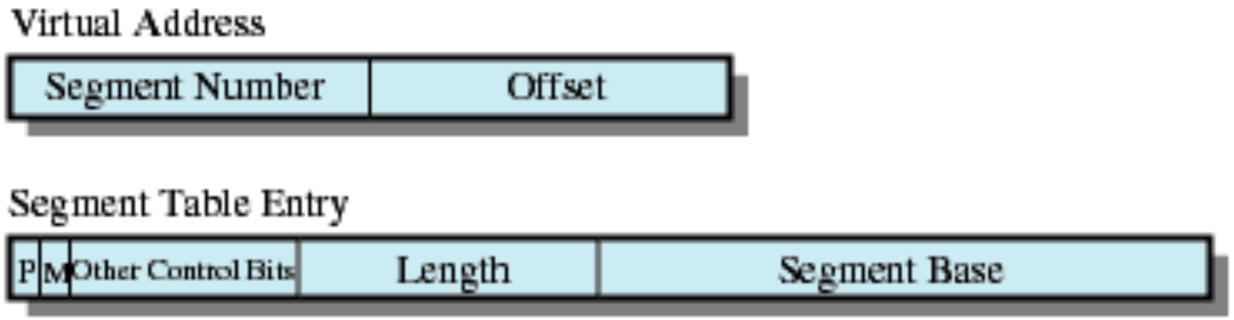
Segmentation

- 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



(b) Segmentation only

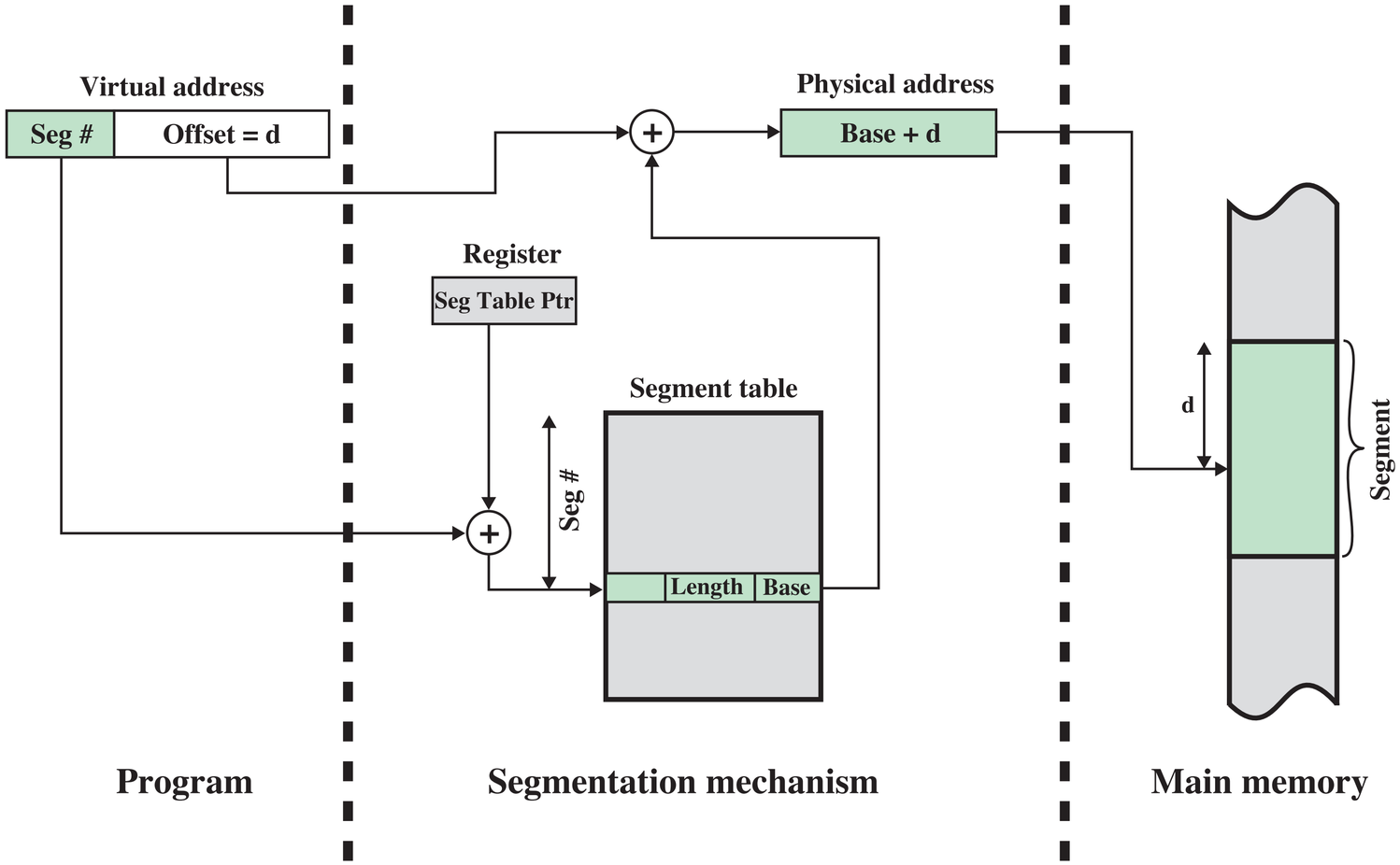
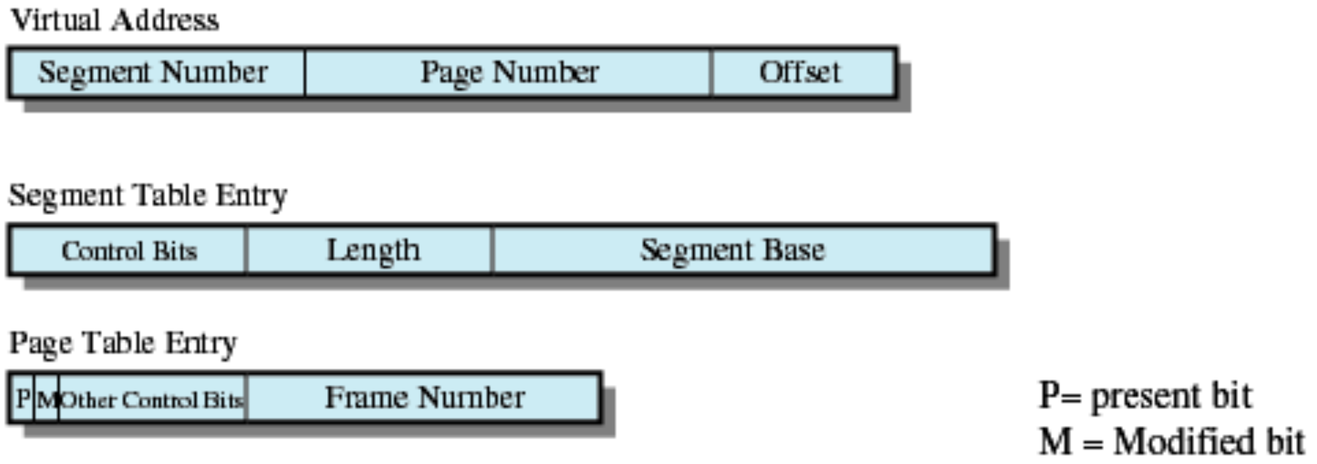


Figure 8.11 Address Translation in a Segmentation System

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

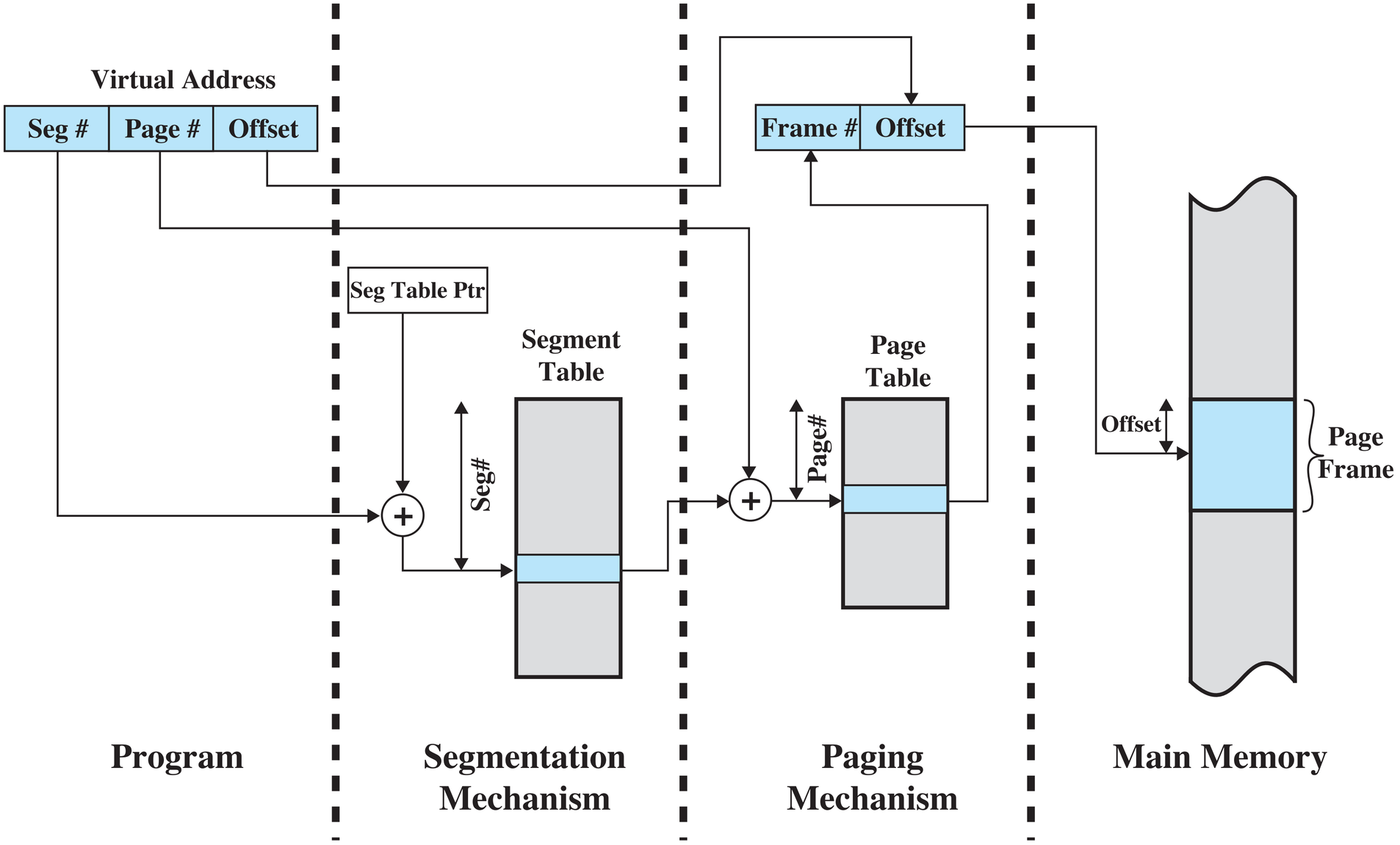


Figure 8.12 Address Translation in a Segmentation/Paging System

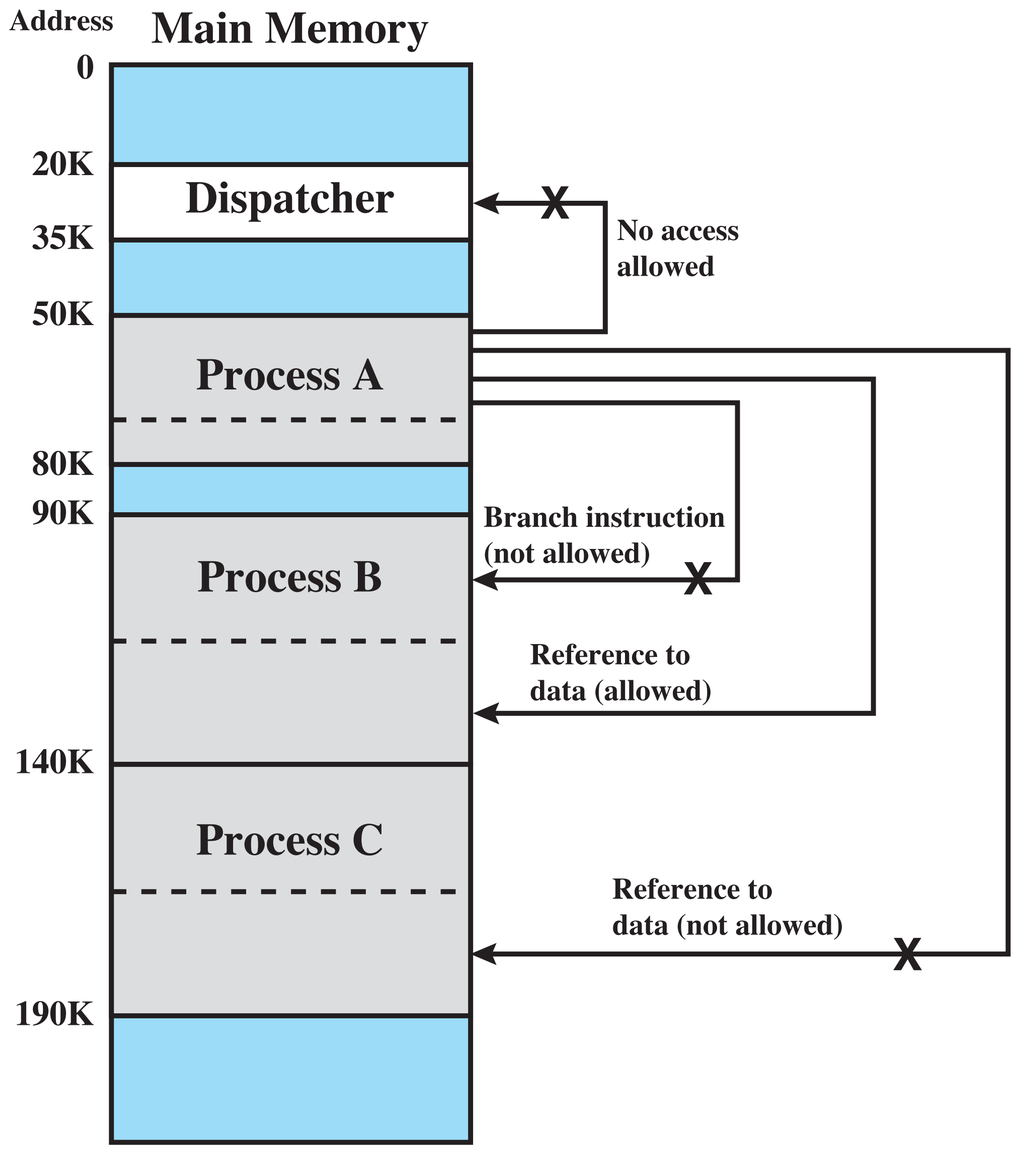


Figure 8.13 Protection Relationships Between Segments

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

- 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 to be 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
  - This policy is “wishful thinking”, but can serve as a base-line when post-evaluating different policies

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

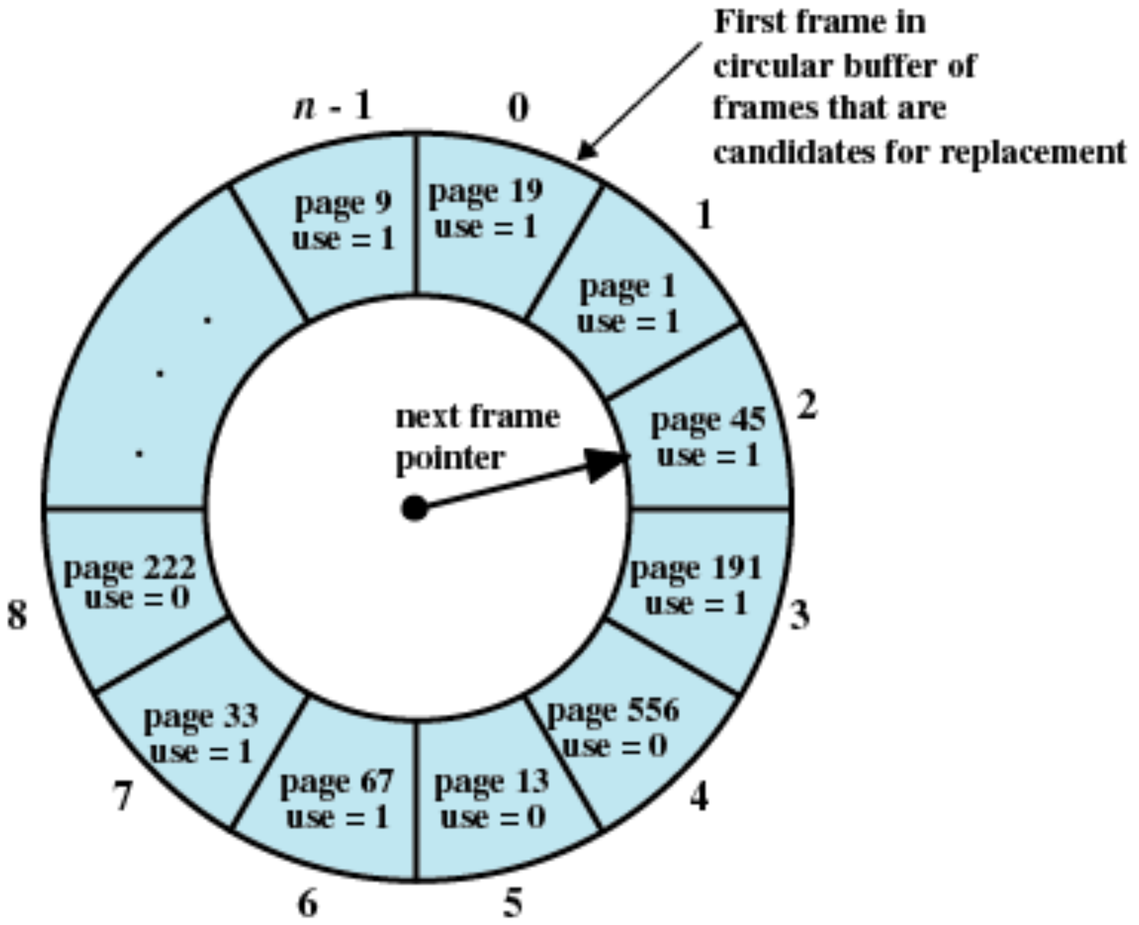
- First-in, first-out (FIFO)
  - Treats page frames allocated to a process as a circular buffer
  - Pages are removed in round-robin style
  - Simplest replacement policy to implement
  - Page that has been in memory the longest is replaced
  - These pages may be needed again very soon
  - Performs relatively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

Basic Replacement Algorithms

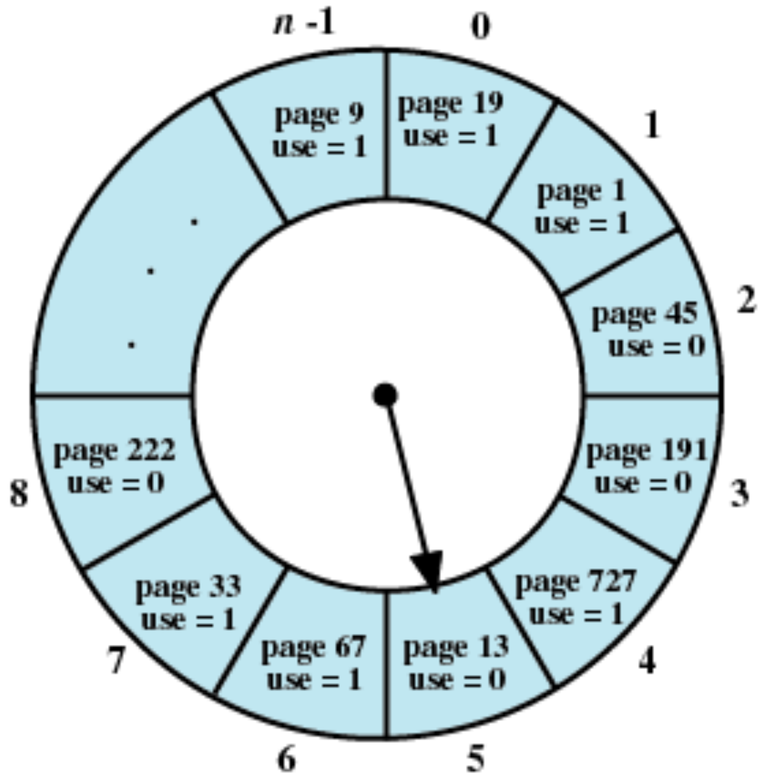
- Clock Policy
  - Additional bit called a *use* bit
  - When a page is first loaded in memory, the *use* bit is set to 1
  - When the page is referenced, the use bit is set to 1
  - When it is time to replace a page, the first frame encountered with the *use* bit set to 0 is replaced.
  - During the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

Basic Algorithms

- Use “use” and “modify” bits
  1. Scan for first frame with u=0, m=0
  2. If 1) fails look for frame with u=0, m=1, setting the use bits to 0 during scan
  3. If 2) failed repeating 1) and 2) will find a replacement



(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

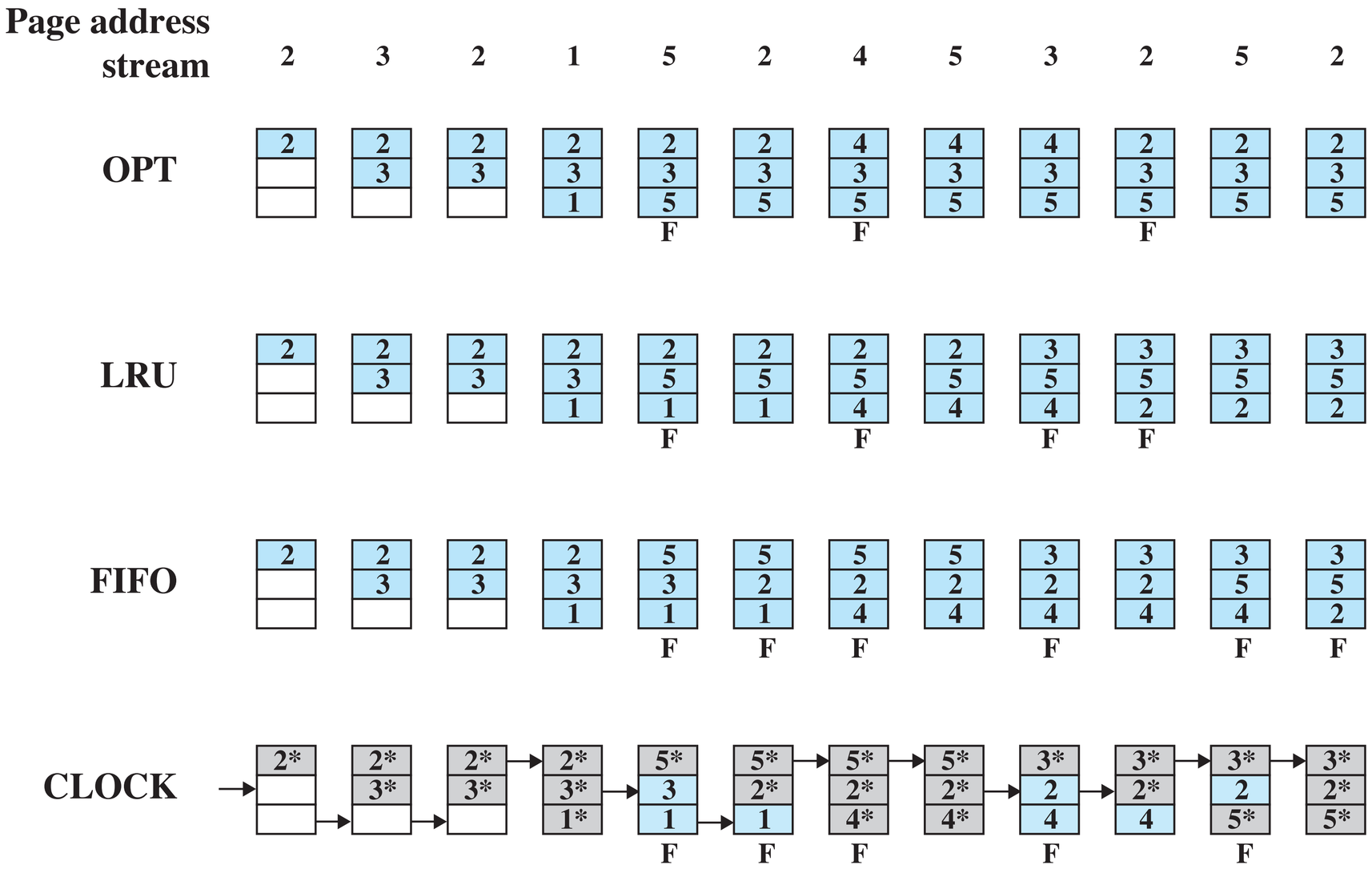


Figure 8.14 Behavior of Four Page-Replacement Algorithms

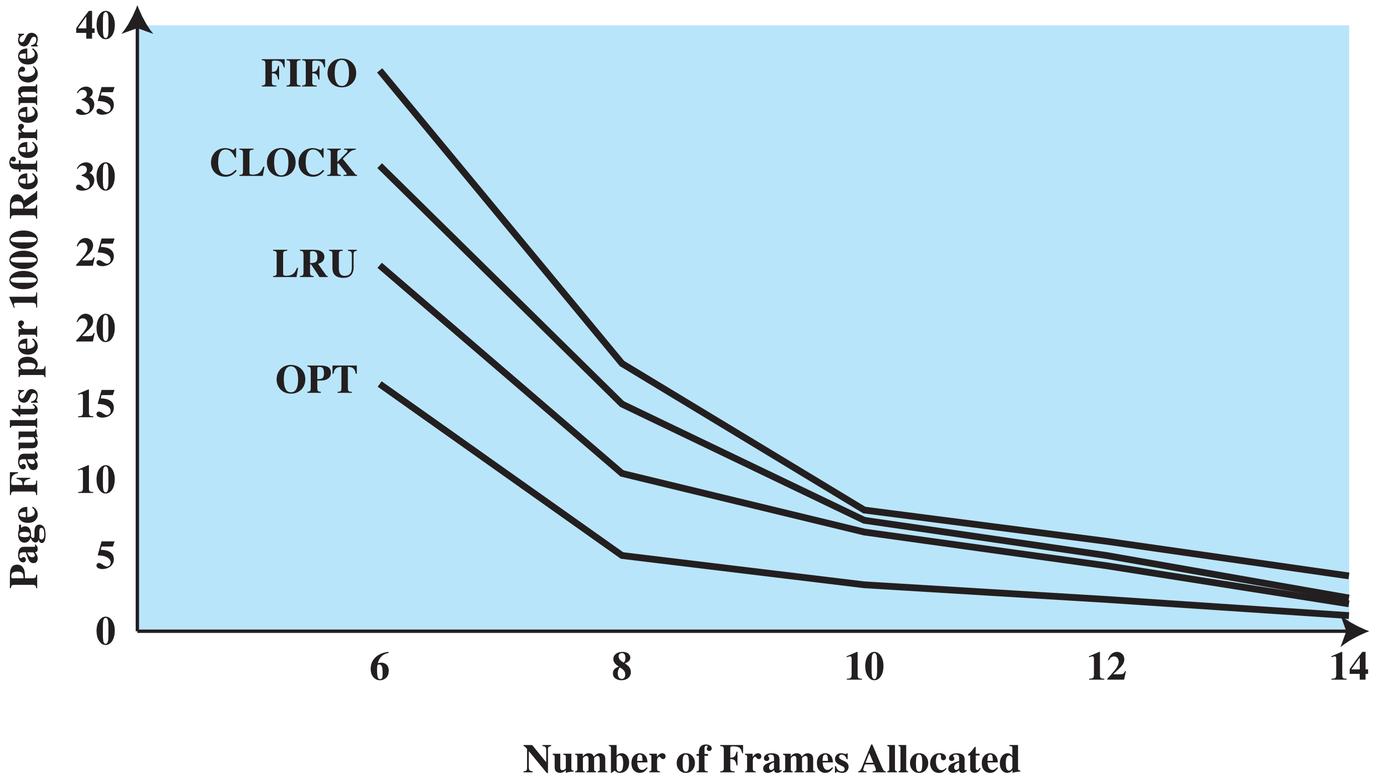


Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

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

Fixed Allocation, Local Scope

- Decide ahead of time the amount of allocation to give a process
  - If allocation is too small, there will be a high page fault rate
  - If allocation is too large there will be too few programs in main memory

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

- 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
    - What is the motivation behind this strategy?
  - 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
  - **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

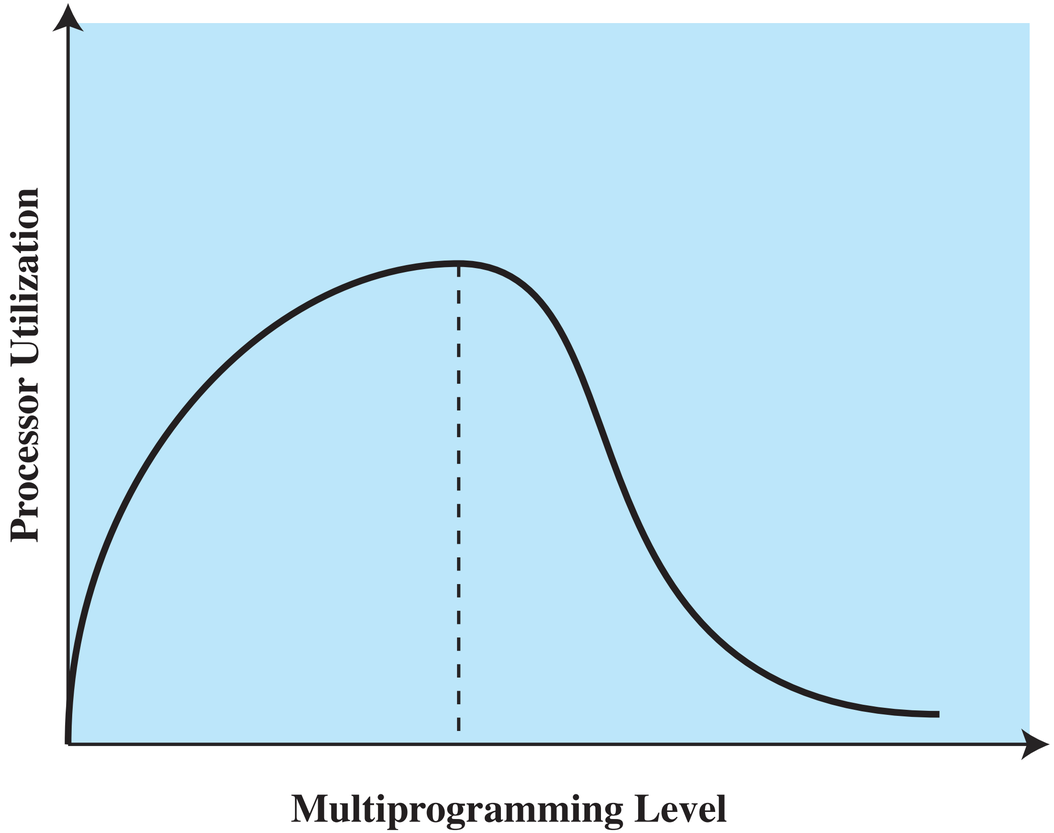


Figure 8.19 Multiprogramming Effects

Process Suspension

- If degree of multiprogramming is to be reduced, suspend:
  - 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 cont.

- 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

UNIX and Solaris Memory Management

- Paging System
  - Page table
  - Disk block descriptor
  - Page frame data table
  - Swap-use table

Table 8.5 UNIX SVR4 Memory Management Parameters (page 1 of 2)

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to zero when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Table 8.5 UNIX SVR4 Memory Management Parameters (page 2 of 2)

Page Frame Data Table Entry

Page State
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

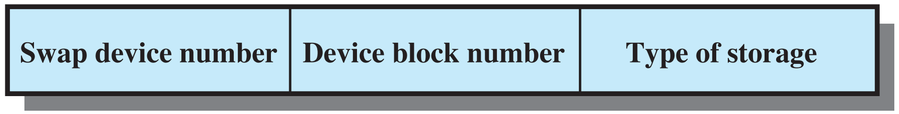
Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

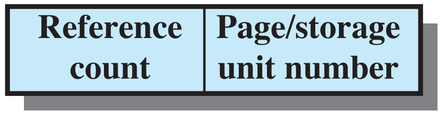
Page/storage unit number
Page identifier on storage unit.

(a) Page table entry



(b) Disk block descriptor

(c) Page frame data table entry



(d) Swap-use table entry

Figure 8.20 UNIX SVR4 Memory Management Formats

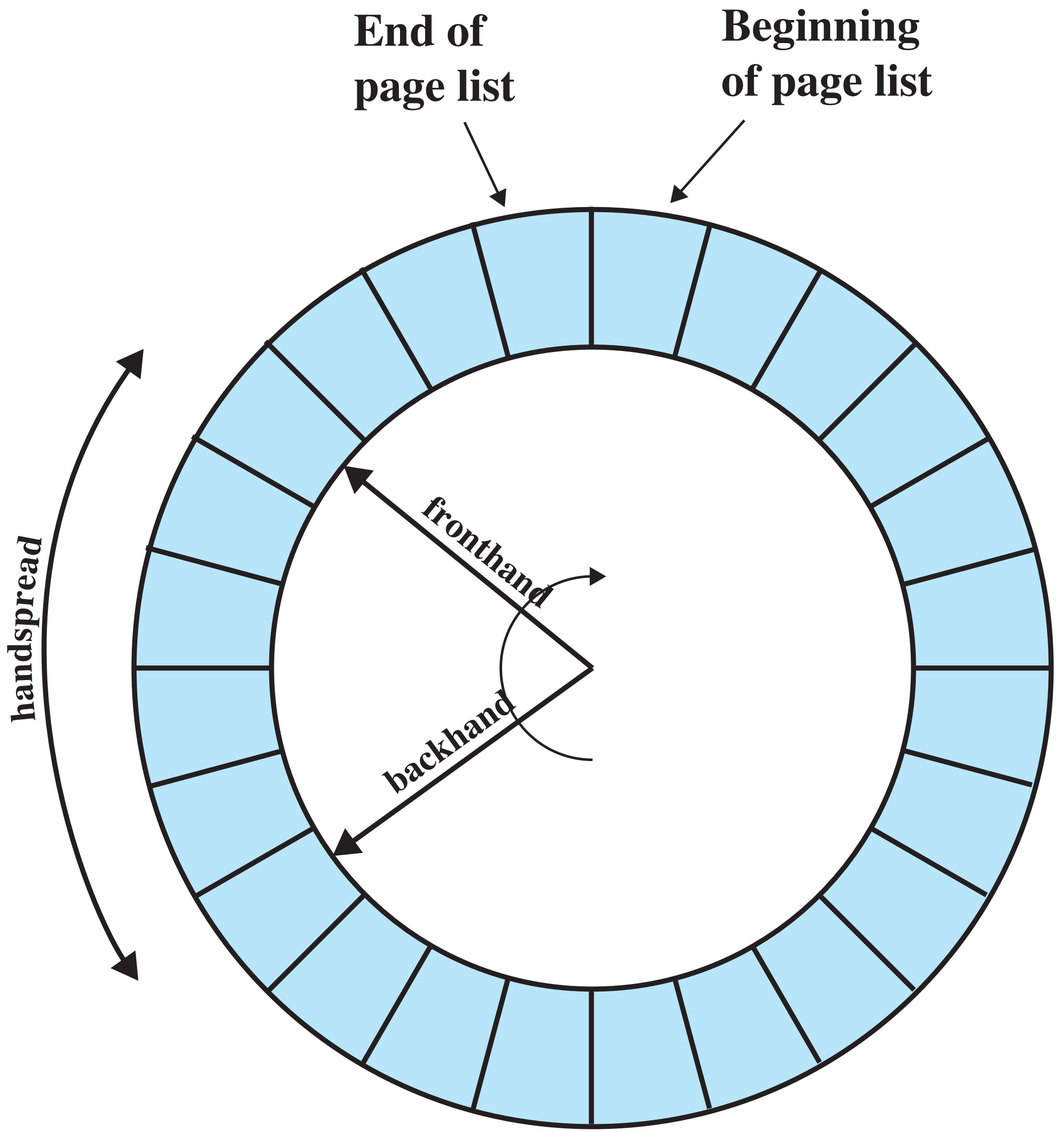


Figure 8.21 Two-Handed Clock Page-Replacement Algorithm

Kernel Memory Allocator

- Lazy buddy system

Initial value of D_i is 0
After an operation, the value of D_i is updated as follows

(I) if the next operation is a block allocate request:
    if there is any free block, select one to allocate
      if the selected block is locally free
          then $D_i := D_i + 2$
          else $D_i := D_i + 1$
    otherwise
      first get two blocks by splitting a larger one into two (recursive operation)
      allocate one and mark the other locally free
      D_i remains unchanged (but D may change for other block sizes because of the recursive call)

(II) if the next operation is a block free request
    Case $D_i \geq 2$
      mark it locally free and free it locally
      $D_i := D_i - 2$
    Case $D_i = 1$
      mark it globally free and free it globally; coalesce if possible
      $D_i := 0$
    Case $D_i = 0$
      mark it globally free and free it globally; coalesce if possible
      select one locally free block of size 2i and free it globally; coalesce if possible
      $D_i := 0$

Figure 8.24 Lazy Buddy System Algorithm

Linux Memory Management

- Page directory
- Page middle directory
- Page table

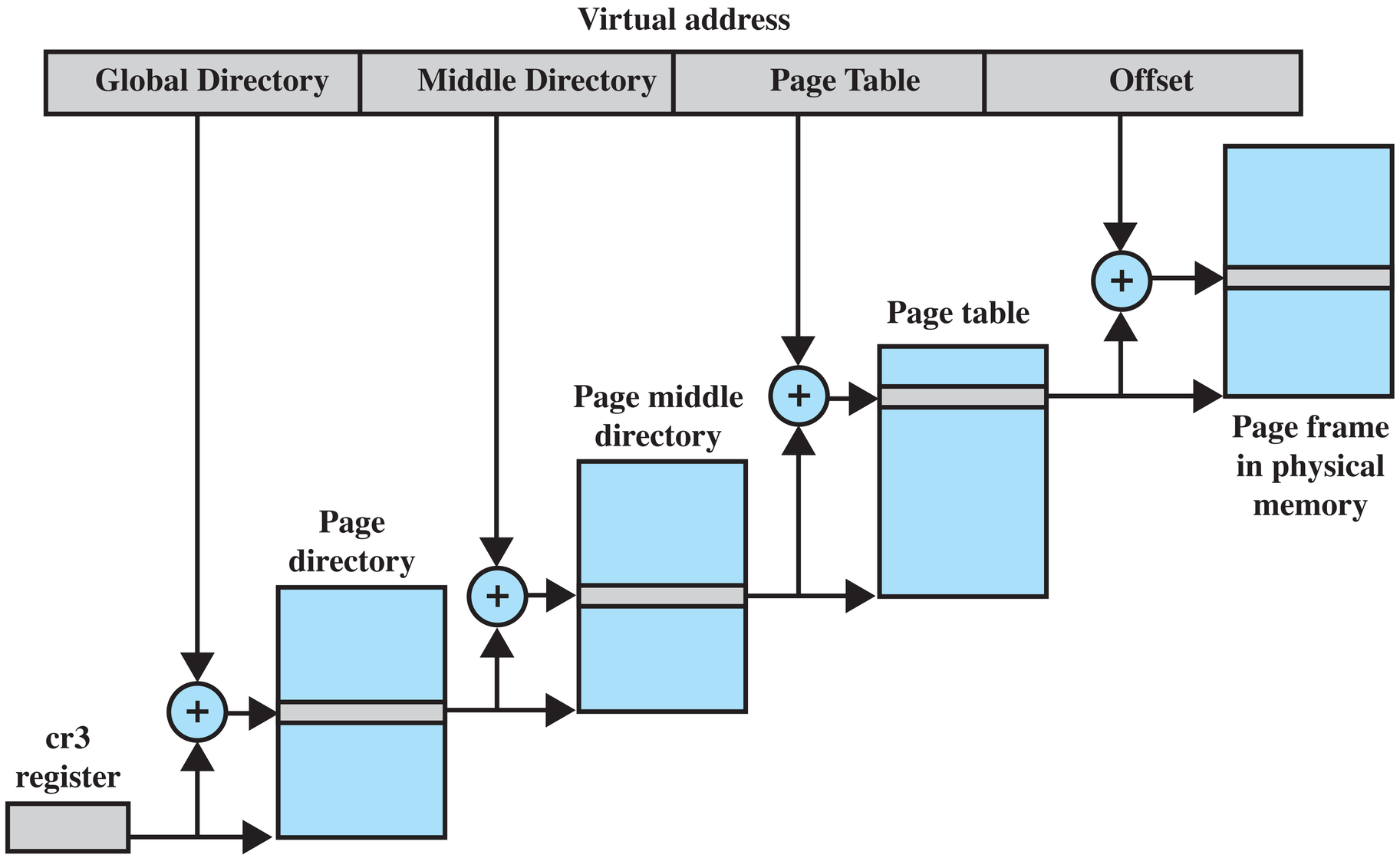


Figure 8.23 Address Translation in Linux Virtual Memory Scheme