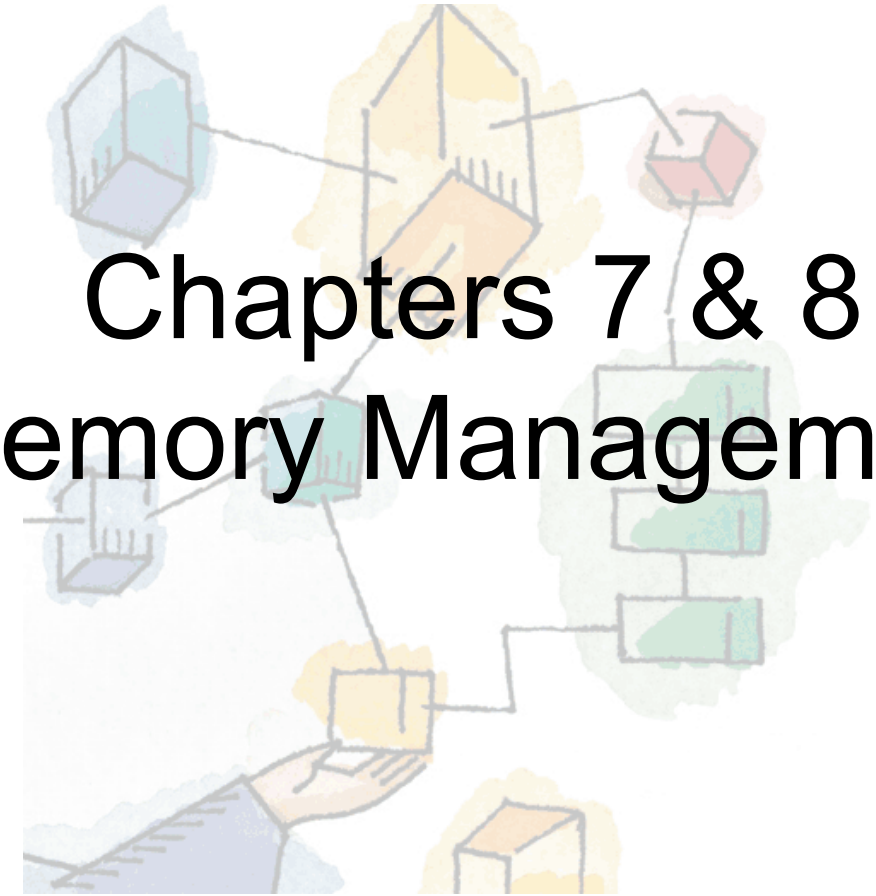
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



Chapters 7 & 8
Memory Management

Patricia Roy
Manatee Community College, Venice, FL

Memory Management

- Subdividing memory to accommodate multiple processes – creating “process image”
- Protection
- Relocation
- Sharing
- Efficient use of memory + reasonable supply of ready processes to achieve a high level of multiprogramming

Use of relative address

- Programmer does not know where the program will be placed in memory when it is executed
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated)
- Memory references in the code must be translated to actual physical memory address

Addresses

- Relative
  - Address expressed as a location relative to some known point, e.g., 10th instruction in code
- Physical or Absolute
  - Actual location in main memory

Registers Used during Execution

- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded or when the process is swapped in

Translation

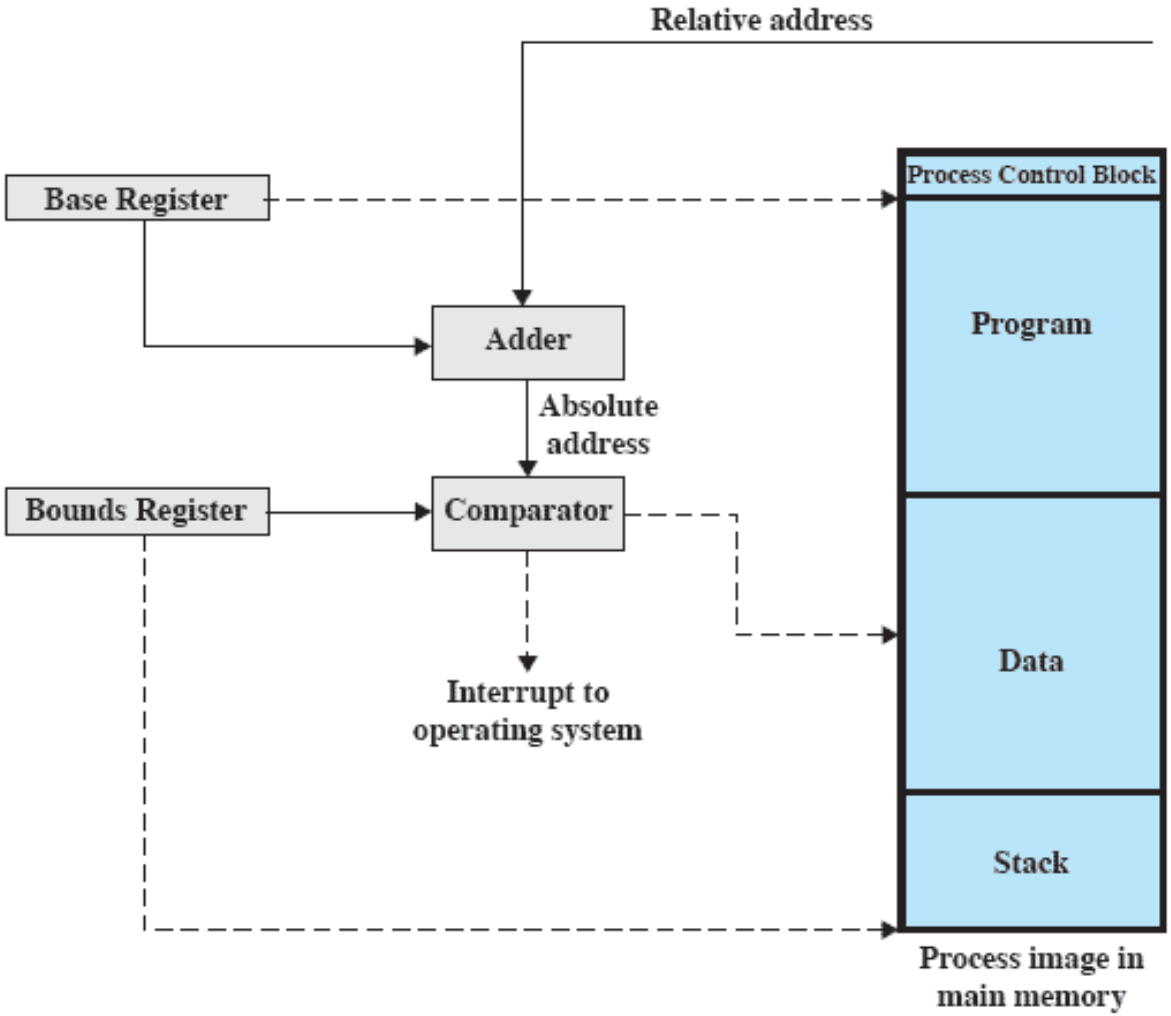


Figure 7.8 Hardware Support for Relocation

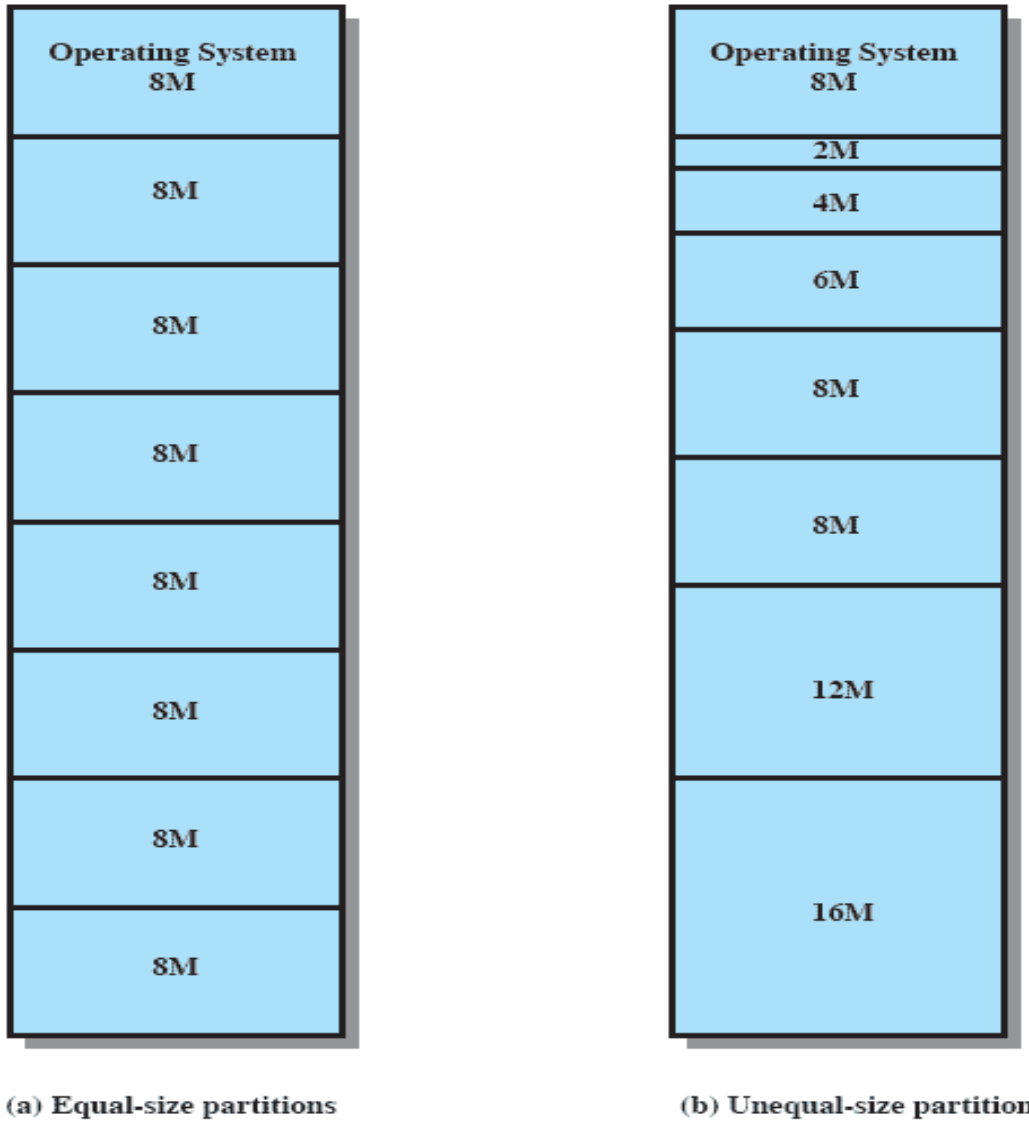
Calculating Absolute Address

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

Fixed Partitioning

- Any process whose size is less than or equal to the partition size can be loaded into an available partition
- If all partitions are full, the operating system can swap a process out of a partition

Example of Fixed Partitioning



(a) Equal-size partitions

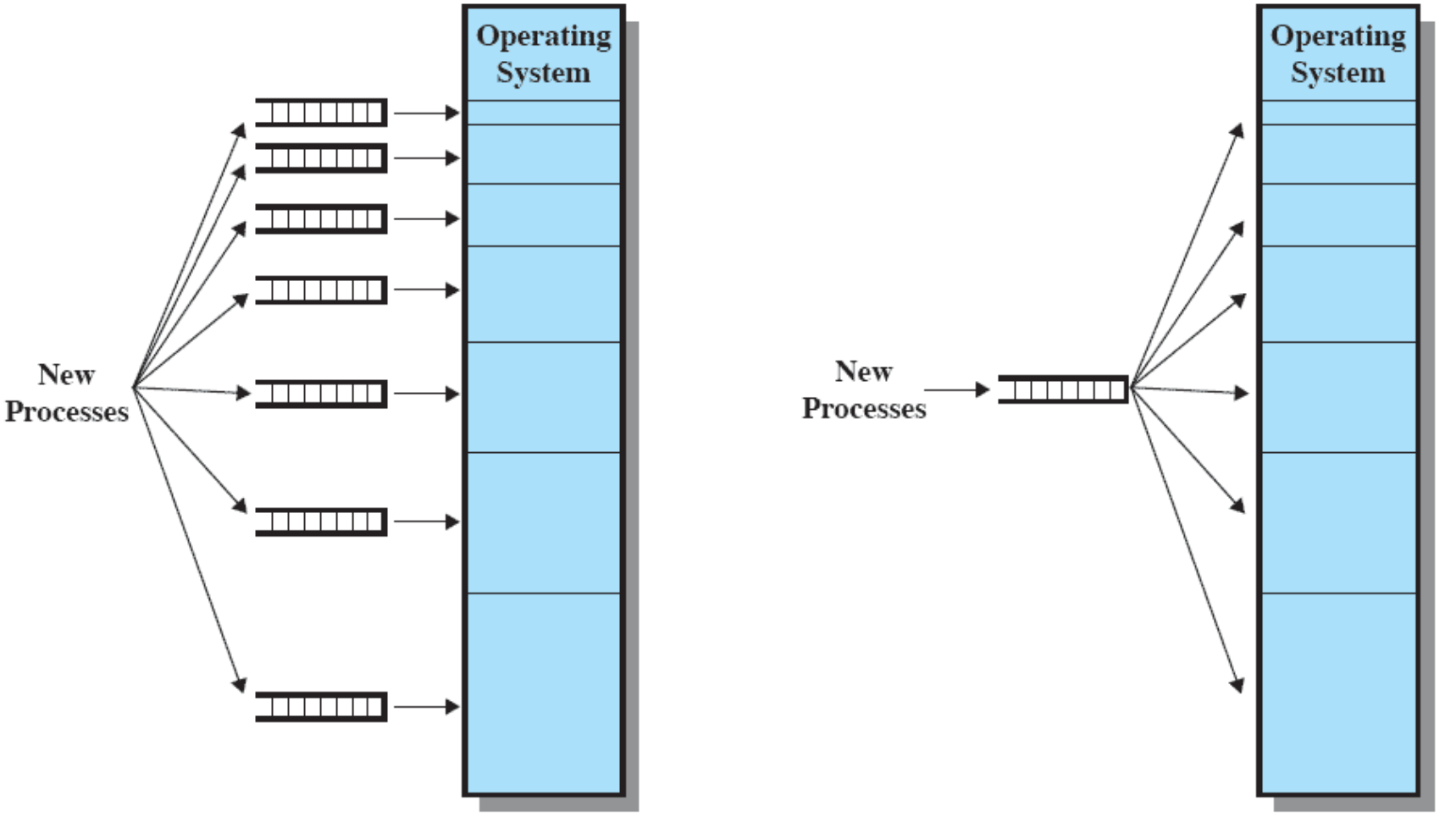
(b) Unequal-size partitions

Figure 7.2 Example of Fixed Partitioning of a 64-Mbyte Memory

Placement Algorithm

- Equal-size
  - Placement is trivial
- Unequal-size
  - Can assign each process to the smallest partition within which it will fit – best fit
  - Single queue or queues for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition

Examples of Fixed Partitioning



(a) One process queue per partition

(b) Single queue

Figure 7.3 Memory Assignment for Fixed Partitioning

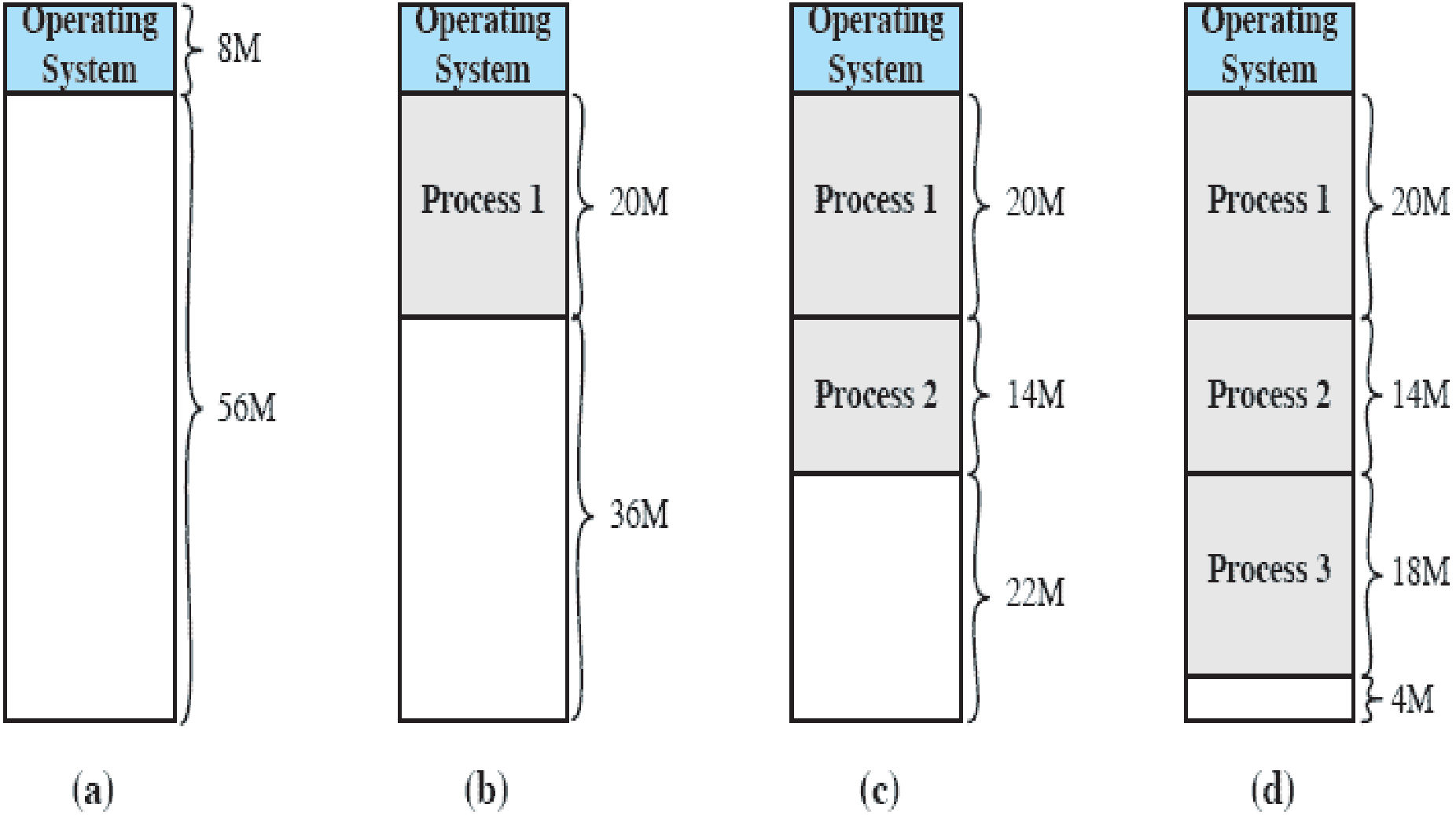
Problems with Fixed Partitioning

- 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 is called **internal fragmentation**.

Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required, so there is no internal fragmentation

Example of Dynamic Partitioning



Example of Dynamic Partitioning (ctd.)

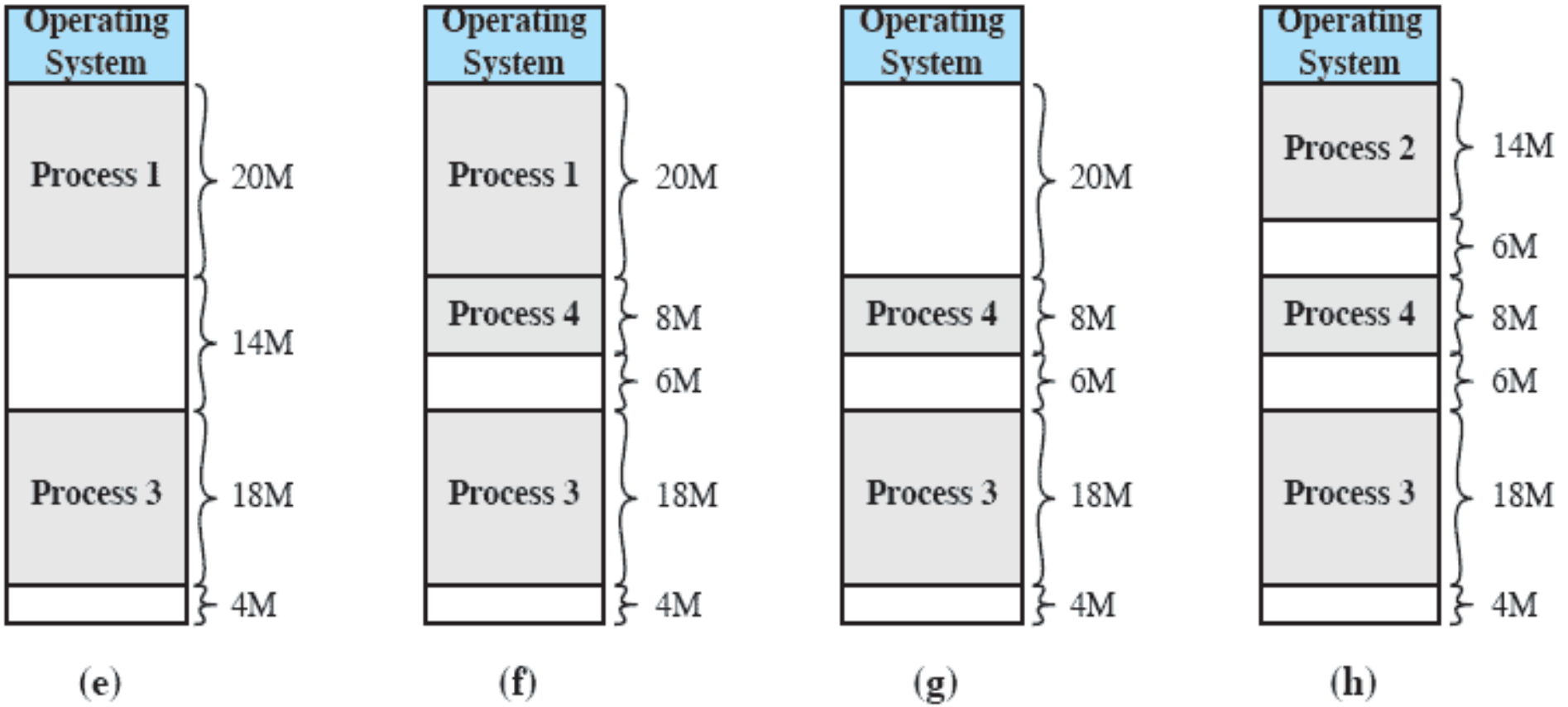


Figure 7.4 The Effect of Dynamic Partitioning

Allocation algorithms

- **Best fit**: chooses the block that is closest in size to the request … lot of little holes
- **First fit**: scans memory form the beginning and chooses the first available block that is large enough
- **Next fit**: scans memory from the location of the last placement
- Eventually get holes in the memory. This is called **external fragmentation**
- Must use **compaction** to shift processes so that they are contiguous and all free memory is in one block

Allocation

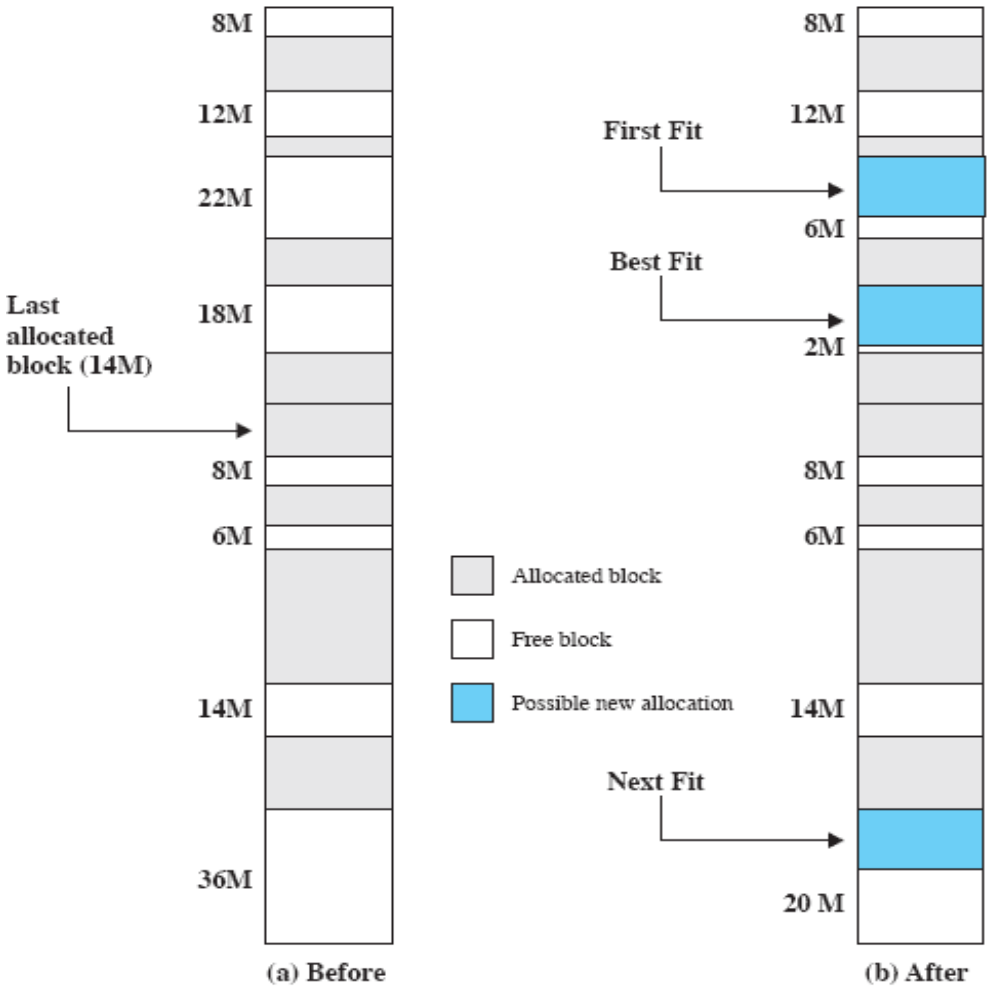


Figure 7.5 Example Memory Configuration before and after Allocation of 16-Mbyte Block

Buddy System

- Entire space available is treated as a single block of 2^U
- If a request of size s such that

  $2^{(U-1)} < s <= 2^U$

  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

UNIX kernel memory allocation

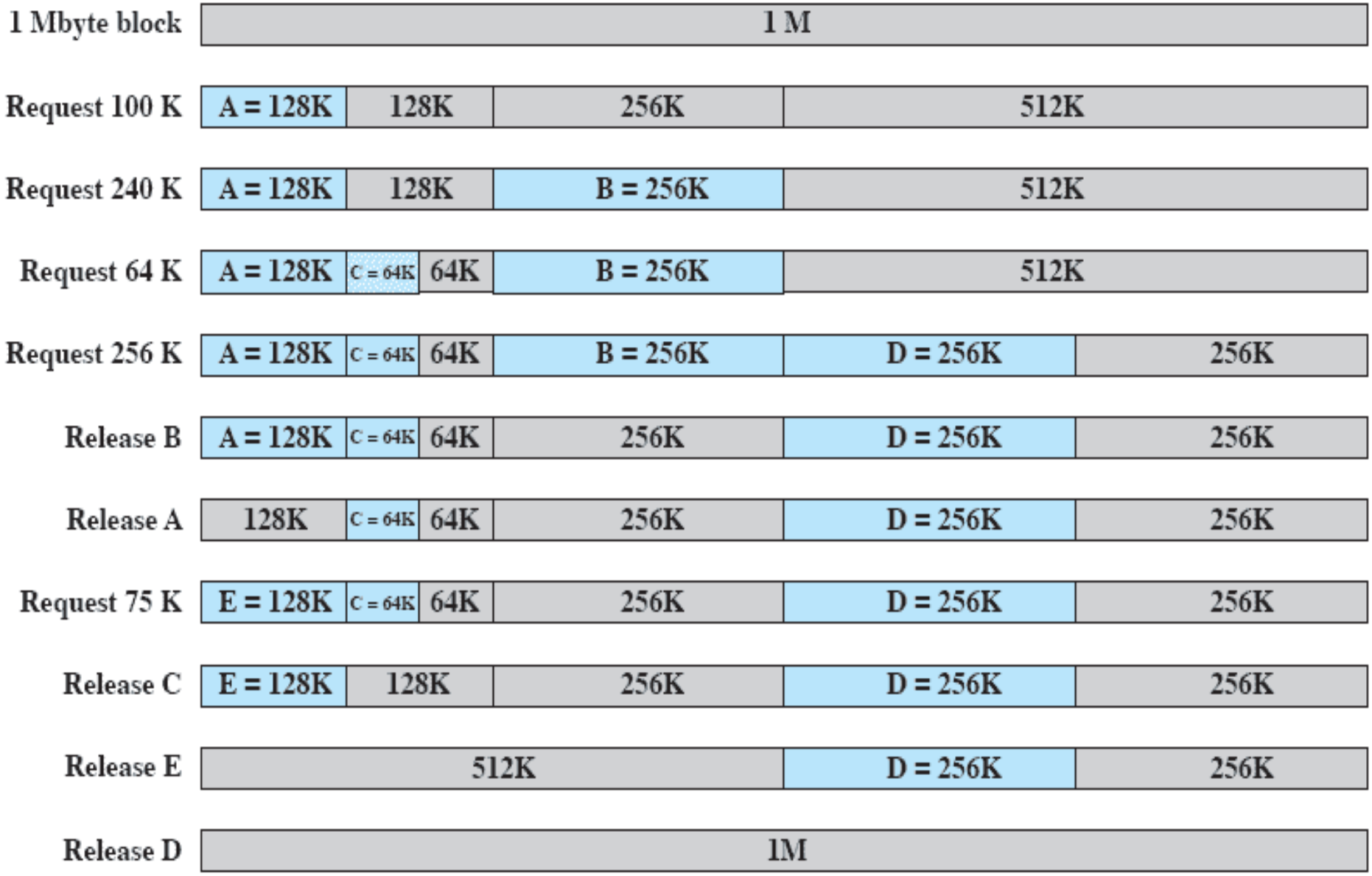


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

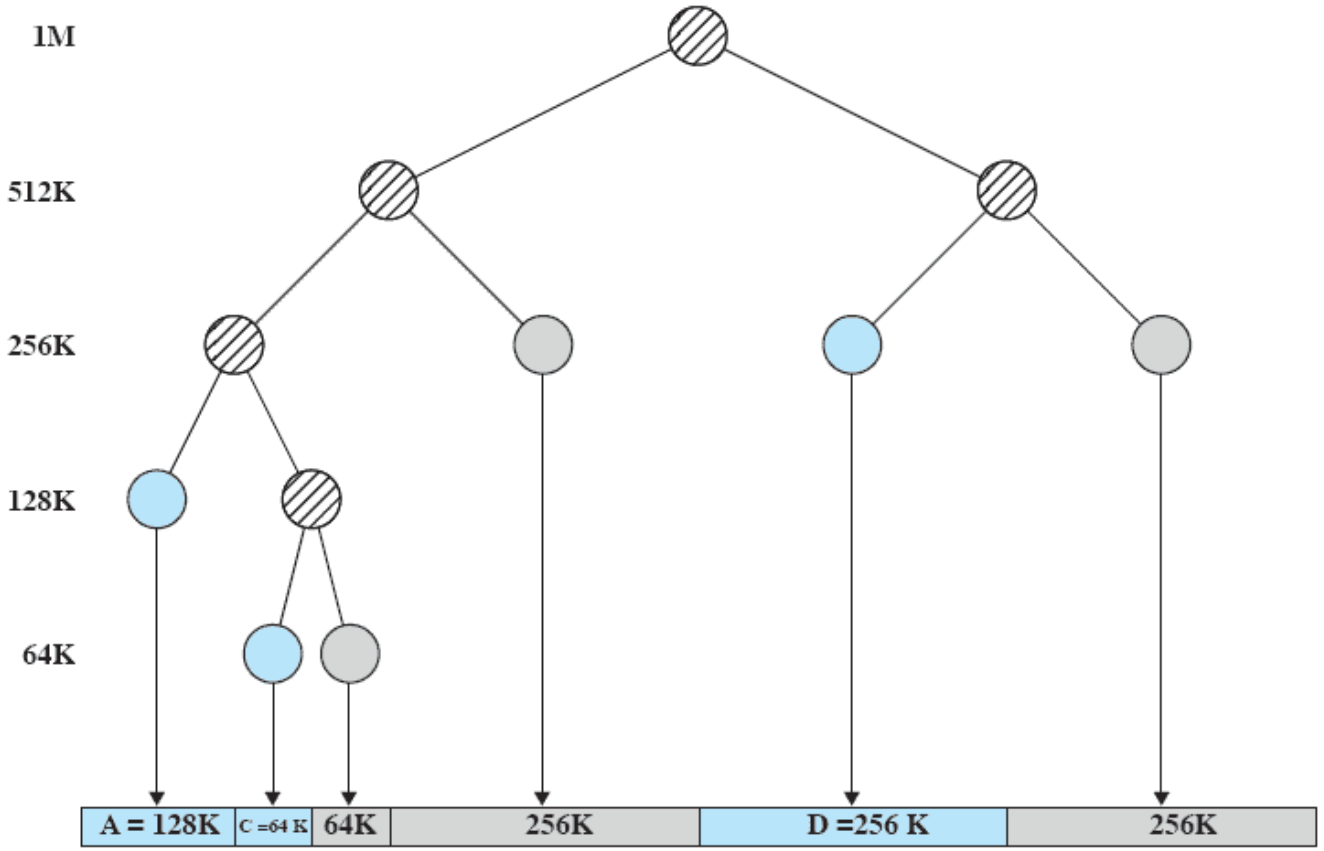


Figure 7.7 Tree Representation of Buddy System

So far:

- 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
- Protection is provided by the use of registers

The Main Idea – Virtual Memory

- Not every piece (page) of a process need to be loaded in main memory at the beginning of execution
- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in 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

Execution of a Program

- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state and issues Read system call

Execution of a Program (ctd.)

- 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

Improved System Utilization

- 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

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

Thrashing

- To many faults: e.g. 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

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** and chunks of memory are called **page frames**

Page Table

- Operating system maintains a page table for each process
  - Contains the frame location for each page resident in memory
  - Memory address consist of a page number and offset within the page
  - Cf. indexed addressing

Page Table

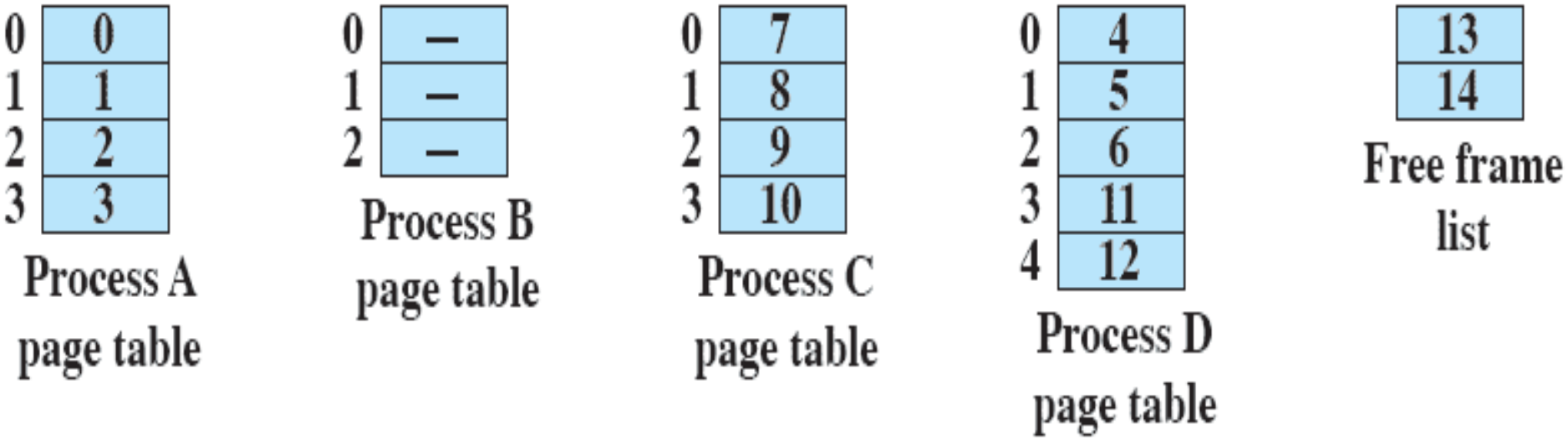


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Segmentation

- All segments of all programs (code, data, stack) do not have to be of the same length
- Addressing consist of two parts - a segment number and an offset
- Since segments are not equal, segmentation is similar to dynamic partitioning

Segmentation

- Simplifies handling of growing data structures
- Lends itself to sharing data among processes, e.g. by sharing data segment
- Lends itself to specific protection policies, different access rights to different segments

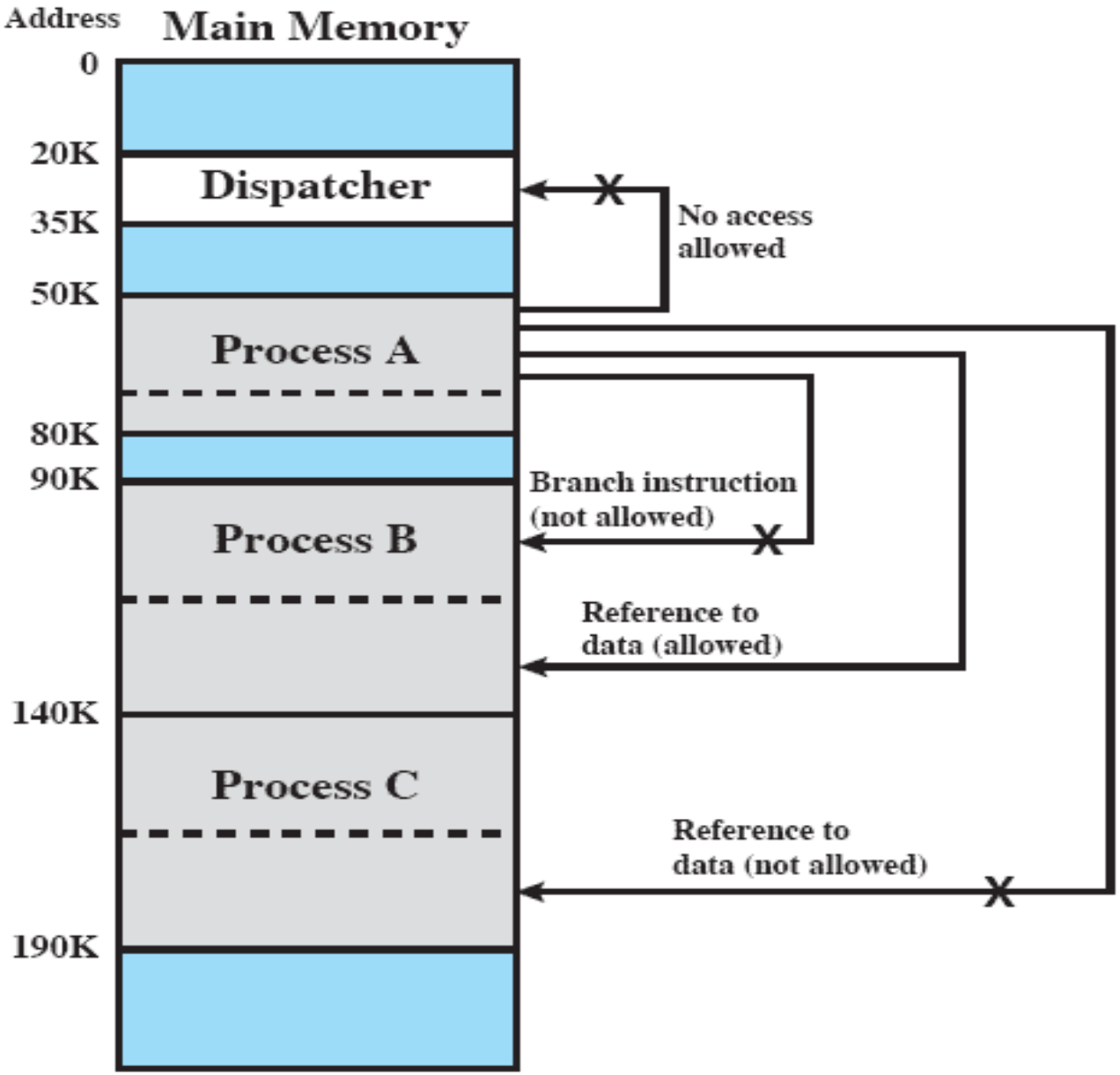


Figure 8.14 Protection Relationships Between Segments

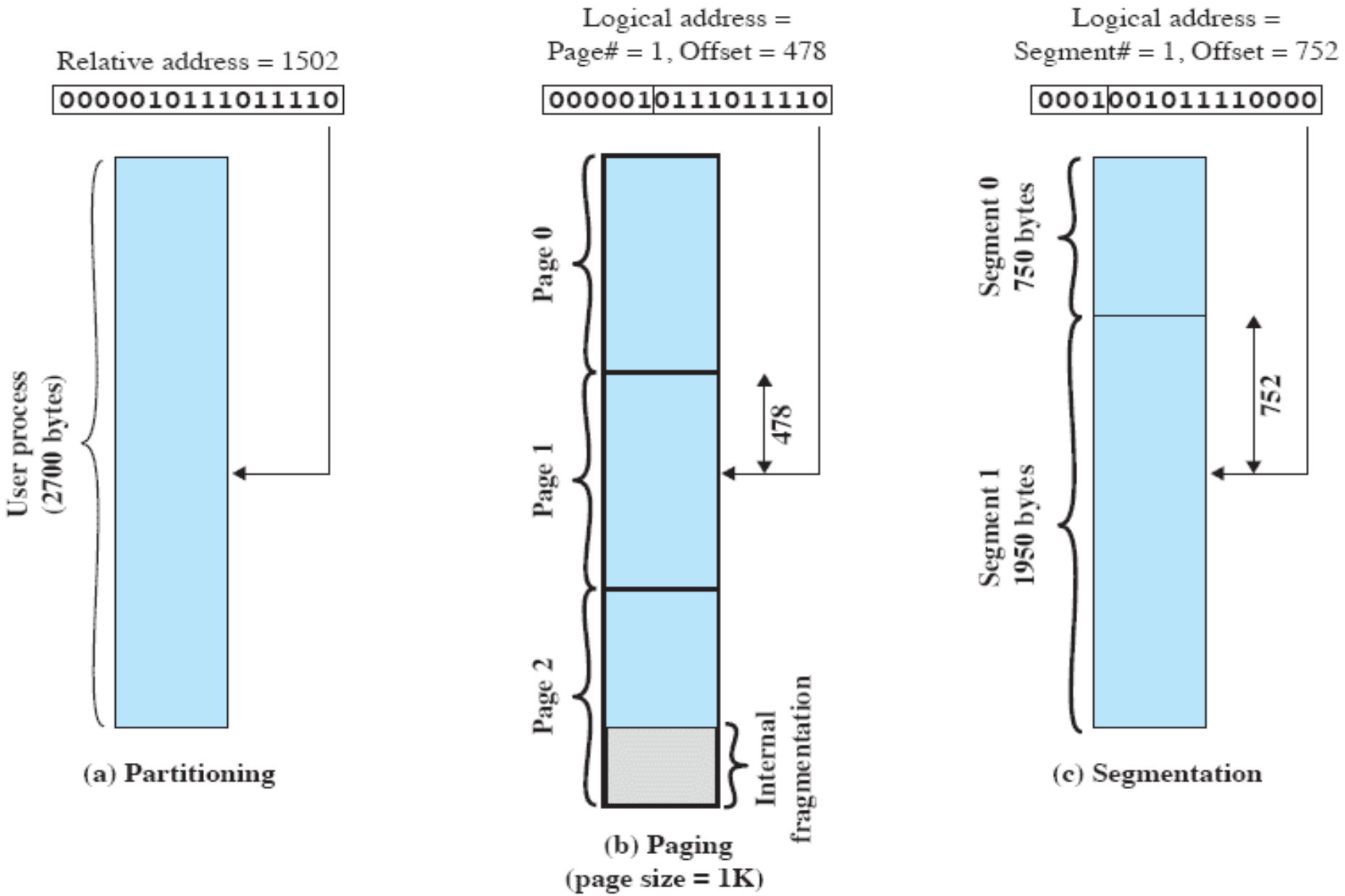
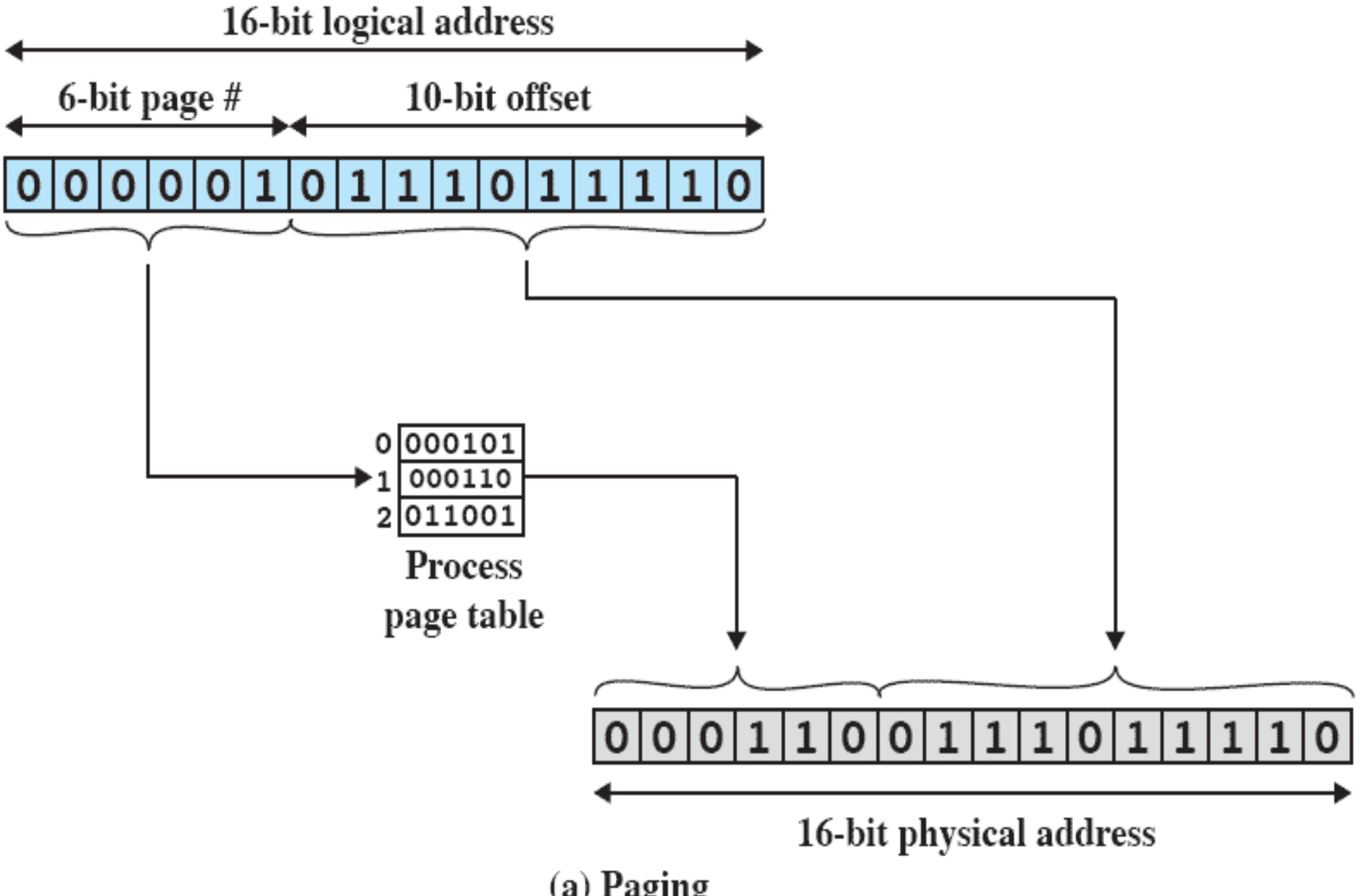


Figure 7.11 Logical Addresses

Translation with Paging



(a) Paging

Translation with Segmentation

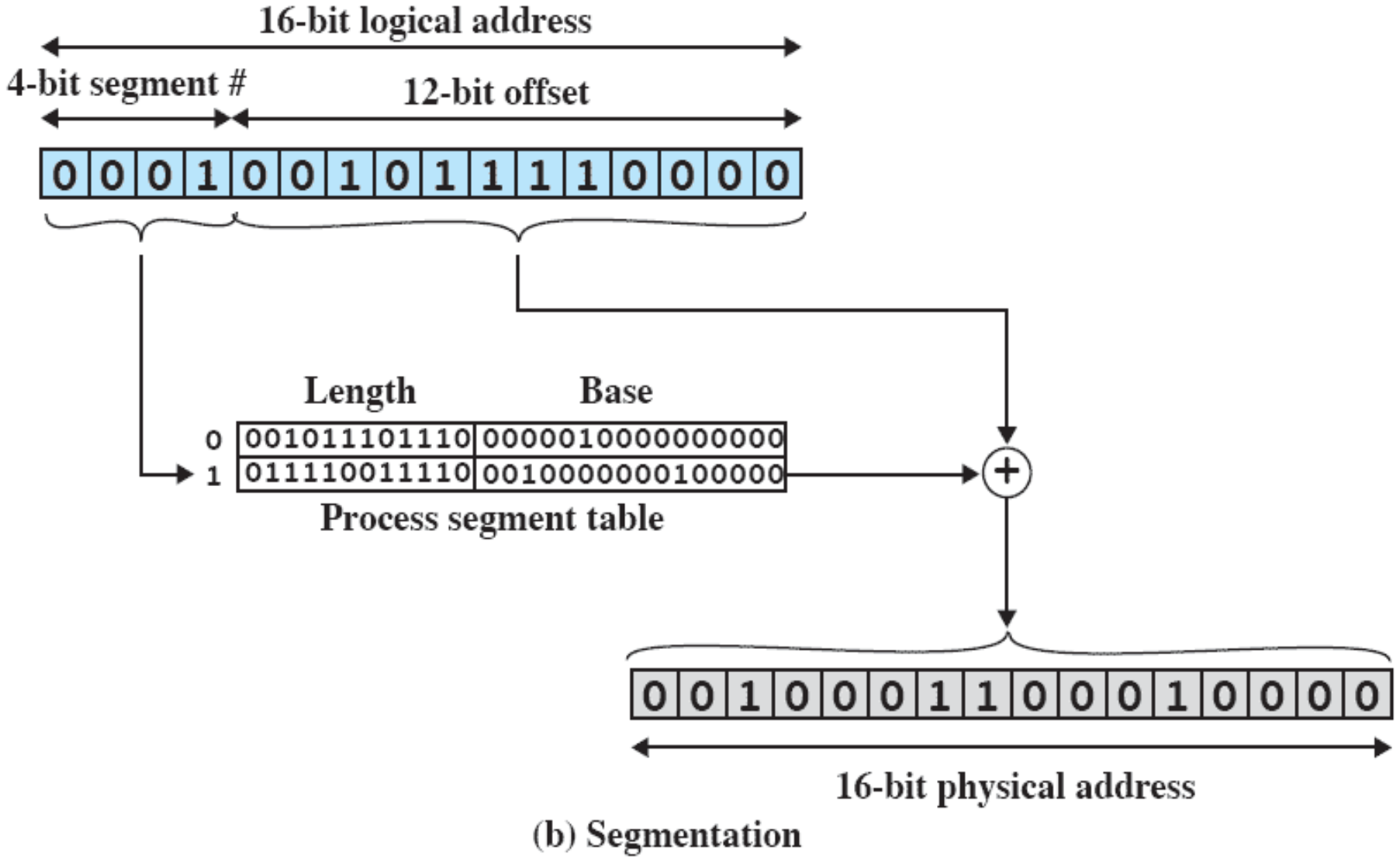


Figure 7.12 Examples of Logical-to-Physical Address Translation

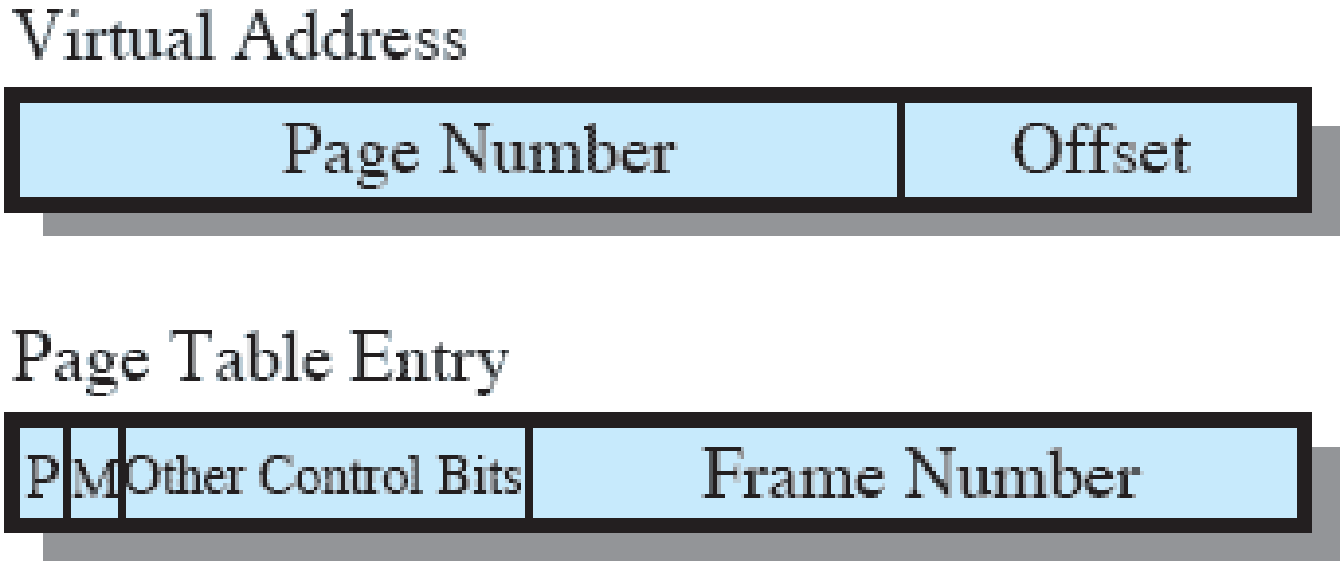
Support Needed for Virtual Memory

- Hardware must support paging and/or segmentation – suitable (indexed, displacement) addressing modes
- 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 corresponding to the page in main memory (if any – partial map)
- A bit is needed to indicate whether the page is in main memory or not
- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

Page Table Entry



(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 replaced

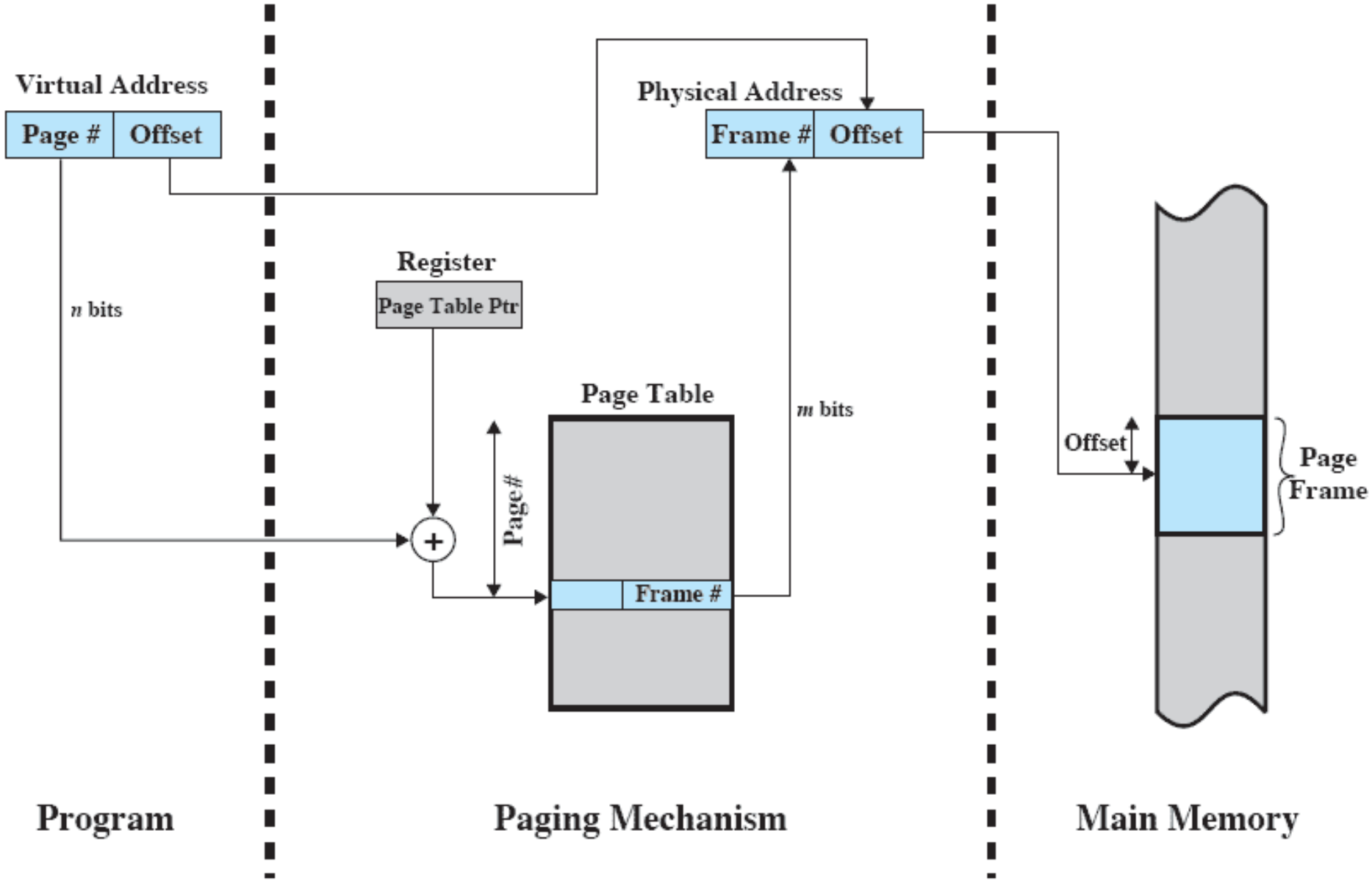


Figure 8.3 Address Translation in a Paging System

Size of Page Table

- 4GB virtual address space and 4KB pages
- 1 million pages and, say, 4 byte page entries
- 4MB page table in memory – although it can be paged

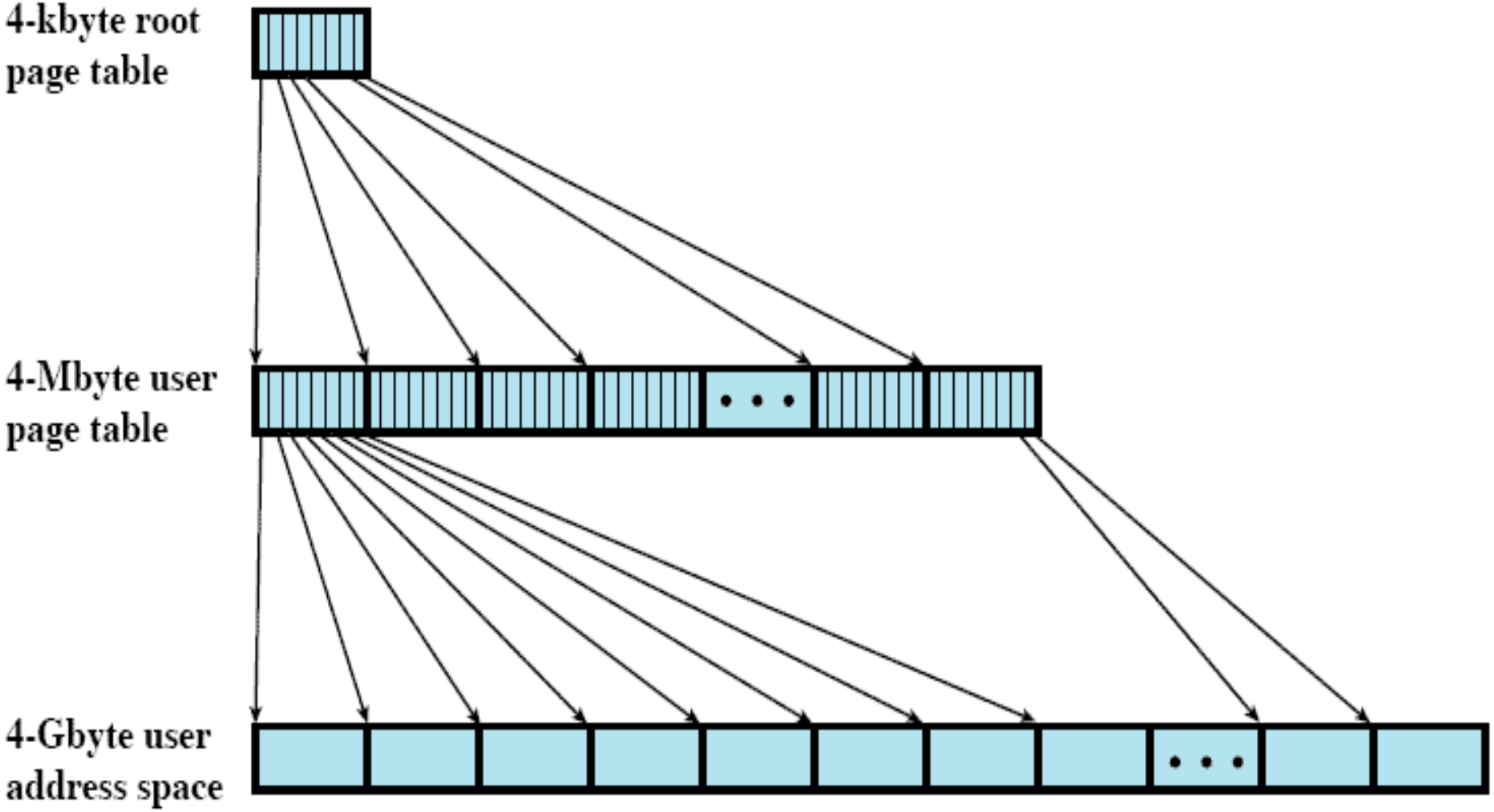


Figure 8.4 A Two-Level Hierarchical Page Table

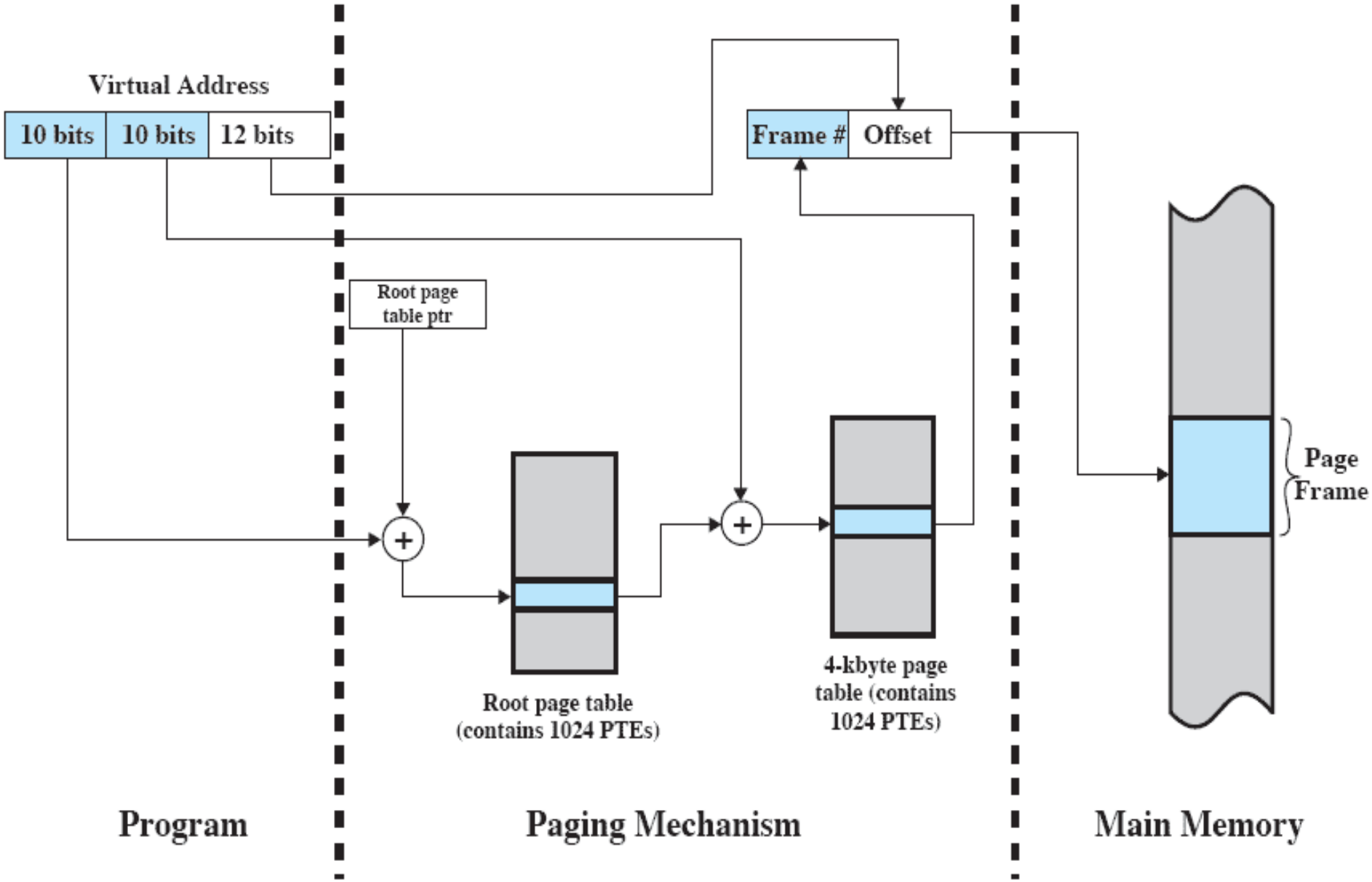


Figure 8.5 Address Translation in a Two-Level Paging System

Size of Multilevel Page Table

- 4GB virtual address space, 4KB pages
- First 10 bits (PTE in root), next 10 bits (PTE in 2nd level table), last 12 bits (offset in page)
- Root page table: 1K entries, 4KB size
- Second level page tables: 1K entries, 4KB size
- Only the root and one 2nd level table have to be in memory at a time!

Inverted Page Table

- Used on PowerPC, UltraSPARC, and IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table entry -> corresponding frame
- Fixed proportion of real memory (number of frames) is required for the table regardless of the number of processes
- Single table, but longer look-up

Inverted Page Table Entry

- Page number
- Process identifier
- Control bits
- Chain pointer

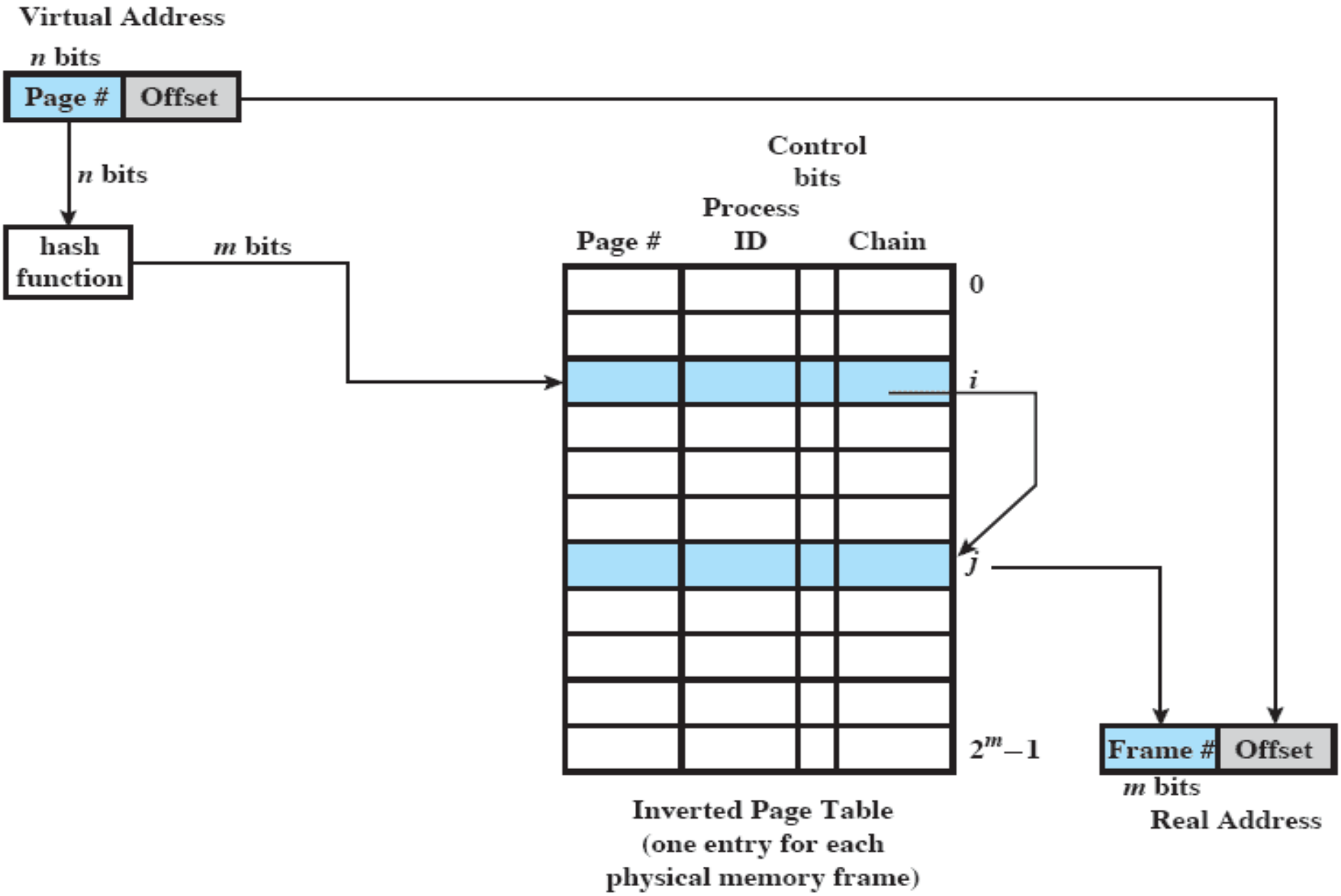


Figure 8.6 Inverted Page Table Structure

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

Translation Lookaside Buffer (ctd.)

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

Translation Lookaside Buffer (ctd.)

- 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

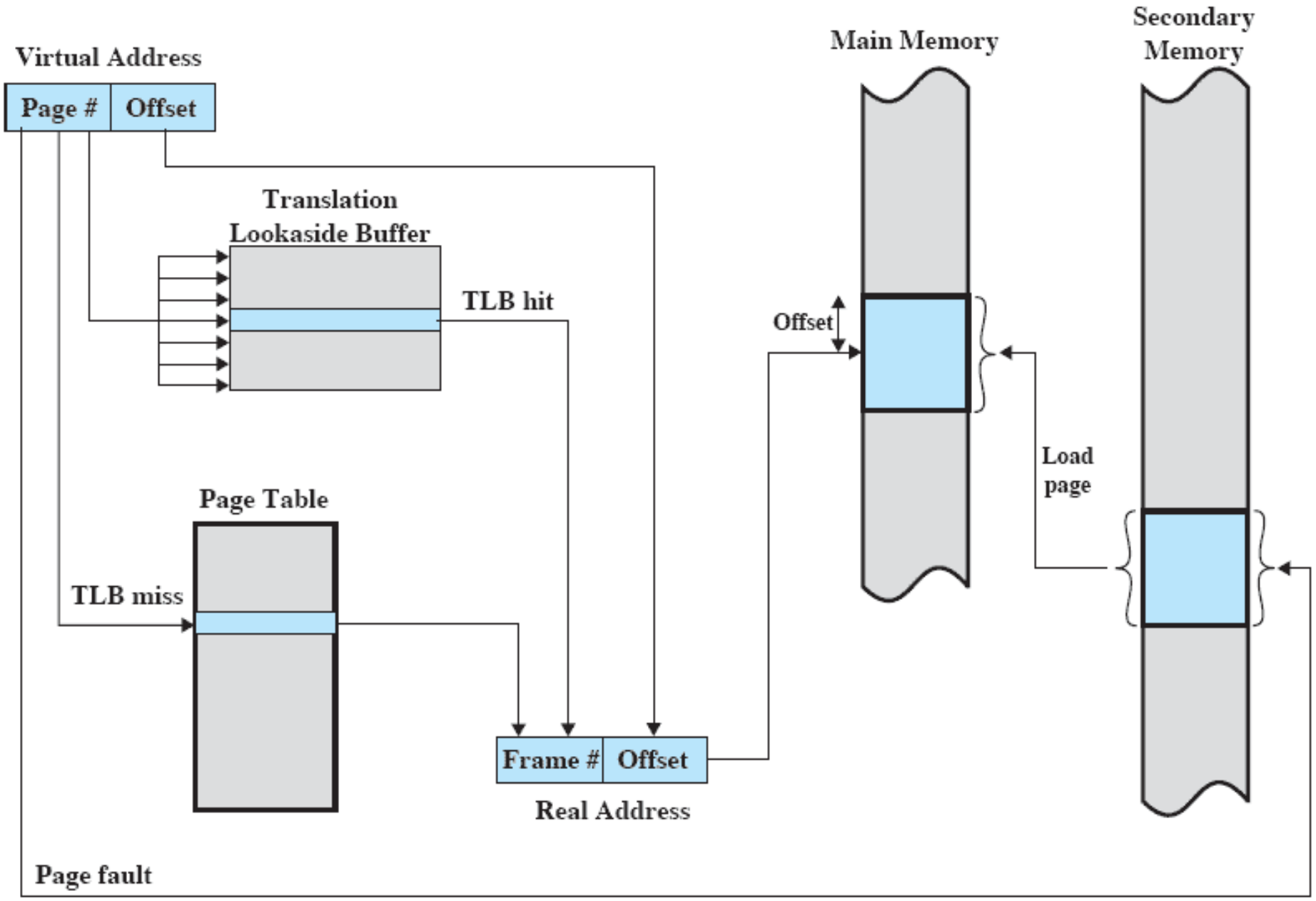


Figure 8.7 Use of a Translation Lookaside Buffer

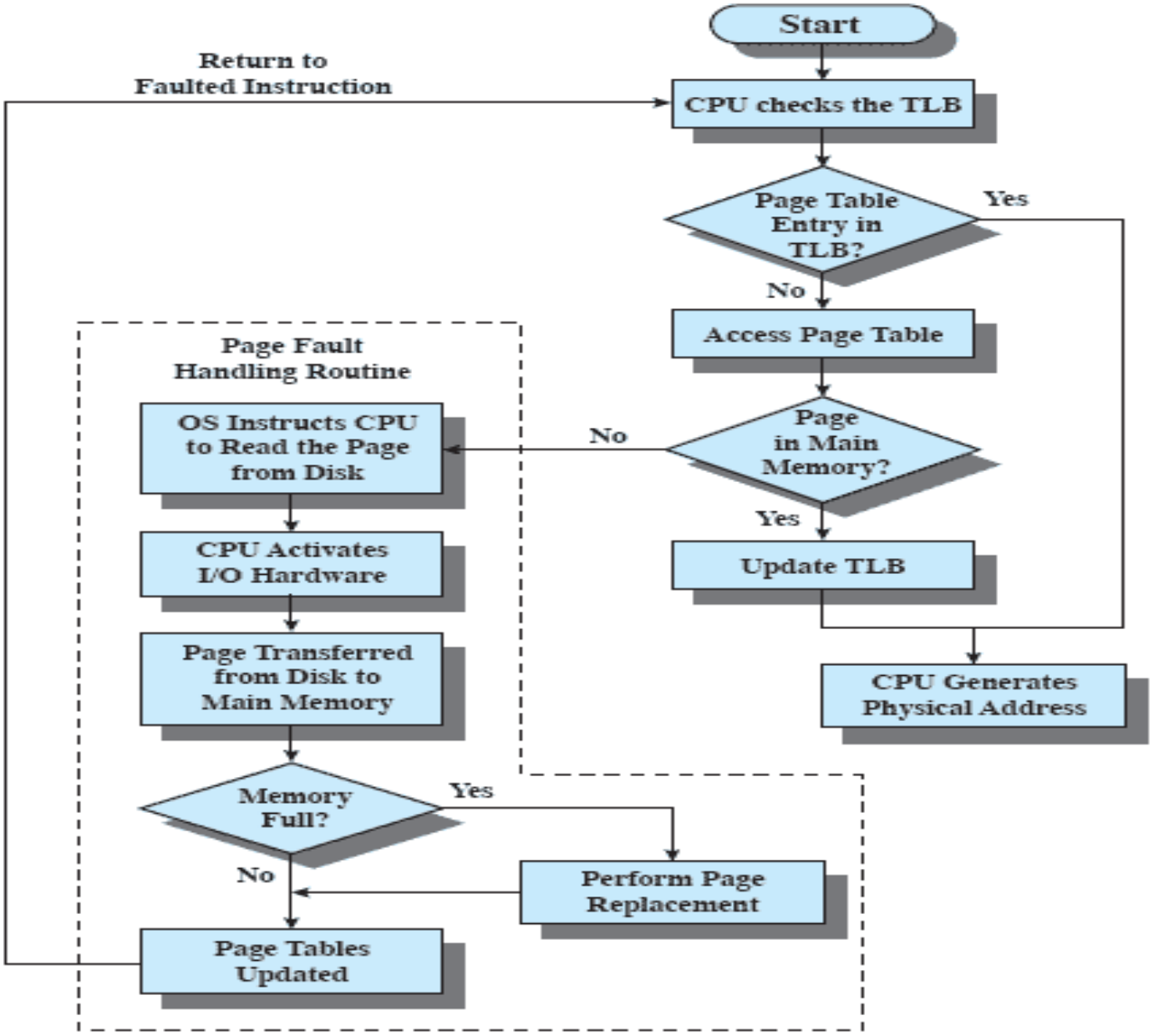


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

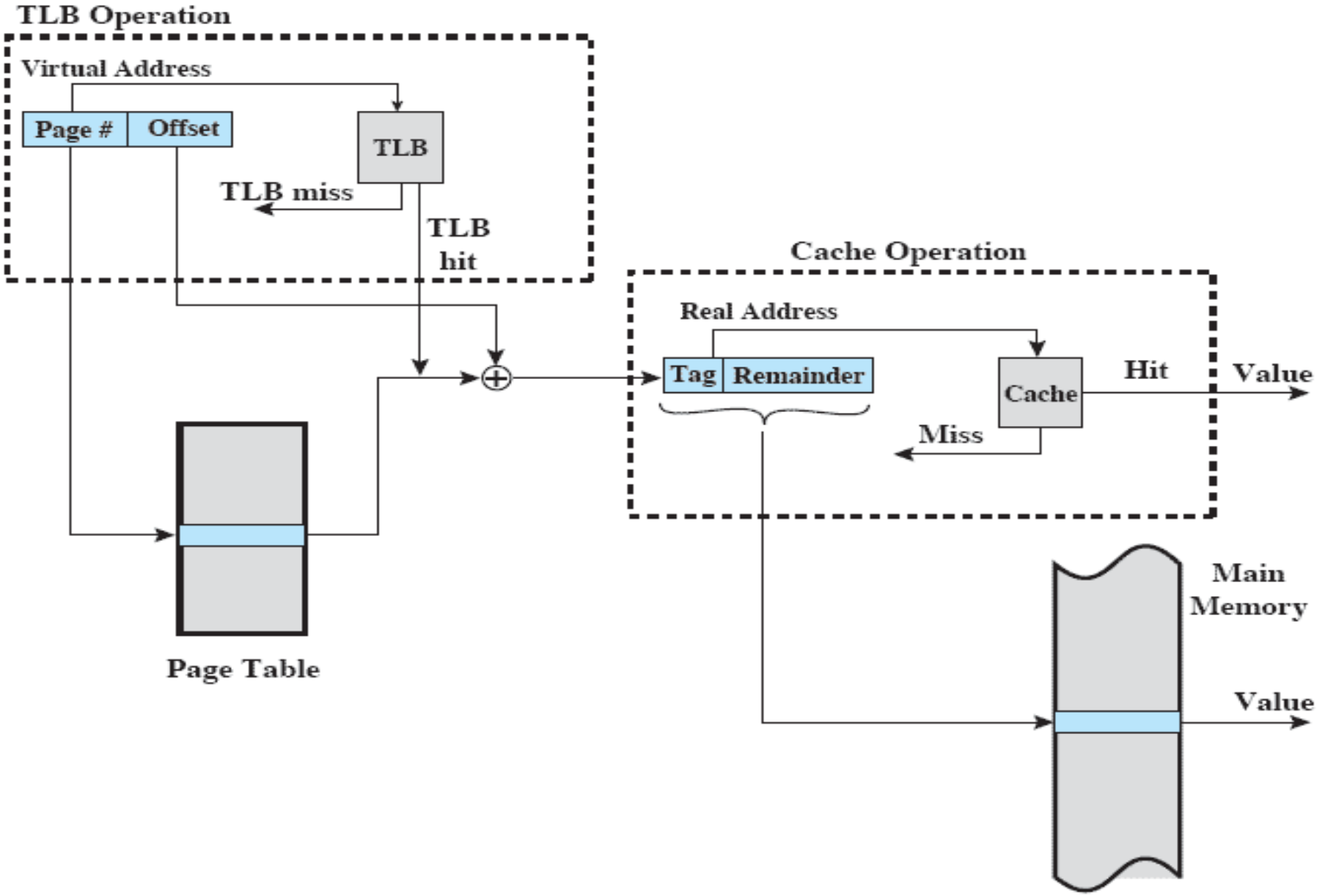


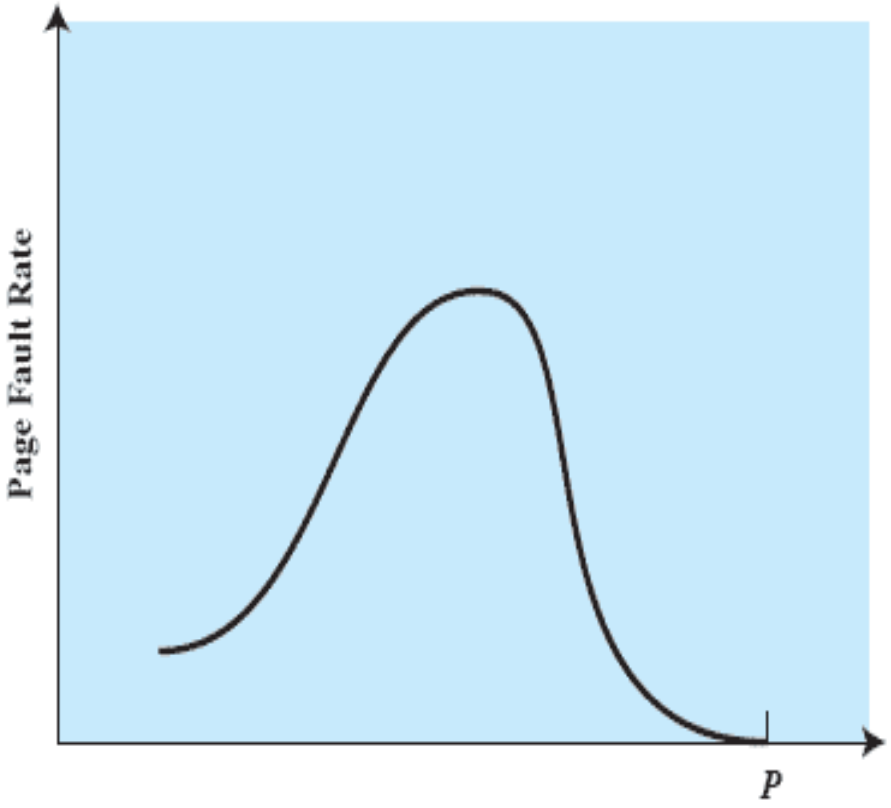
Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page Size

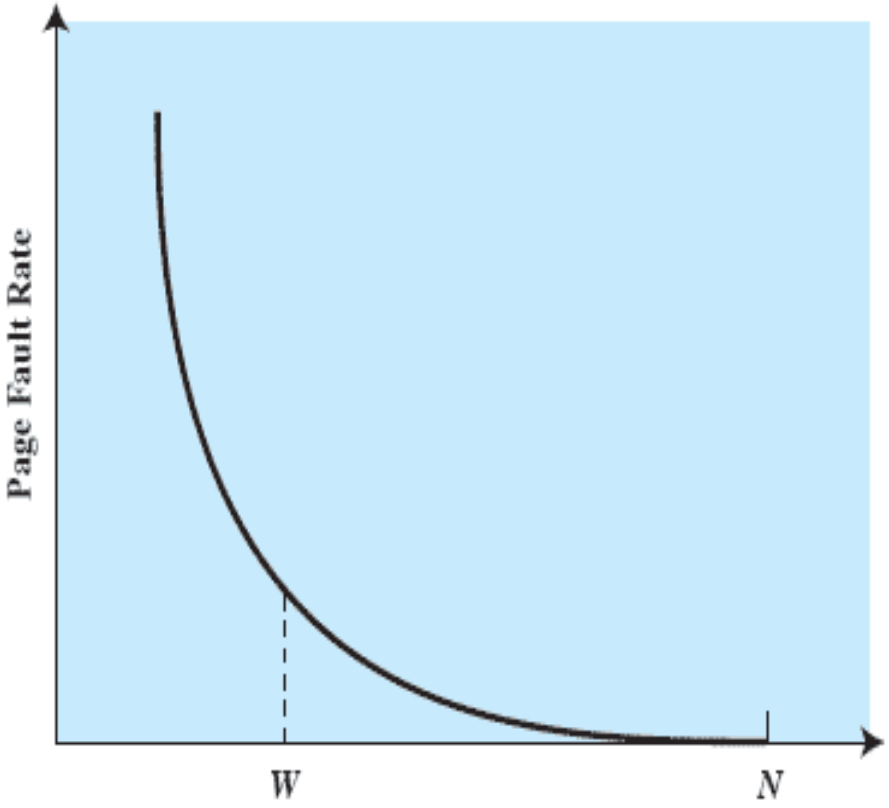
- Smaller page size, less amount of internal fragmentation
- Smaller page size, more pages required per process
- More pages per process means larger page tables
- Larger page tables means large portion of page tables in virtual memory

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.
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better in this respect



(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

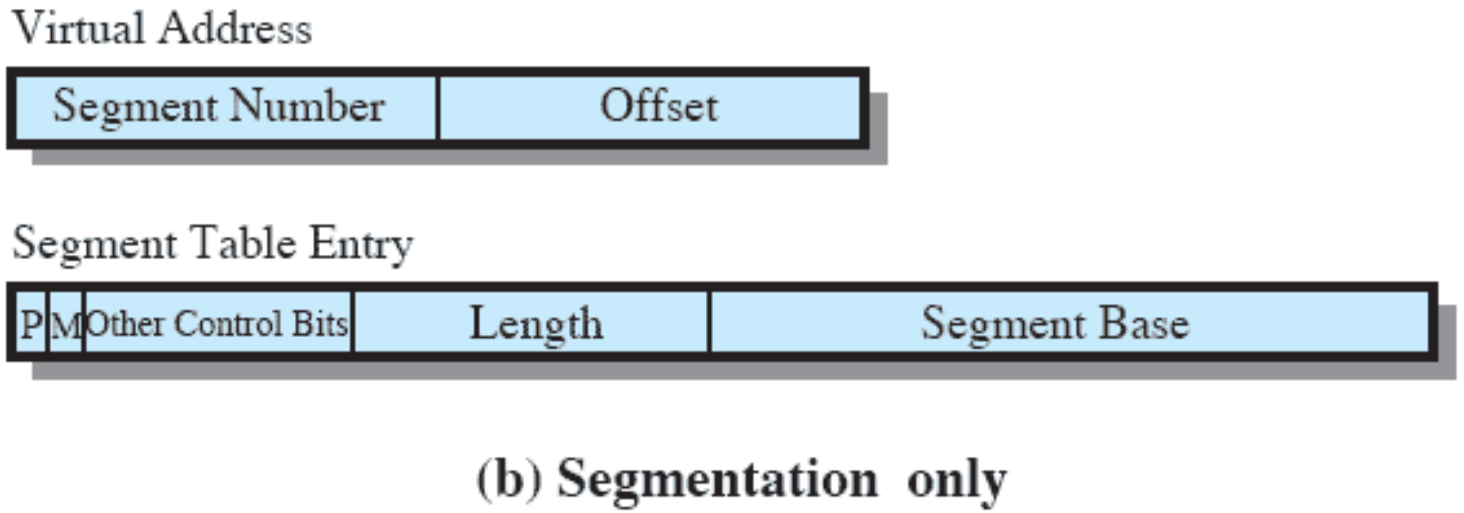
Table 8.3 Example Page Sizes

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
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Segment Tables

- Starting address 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

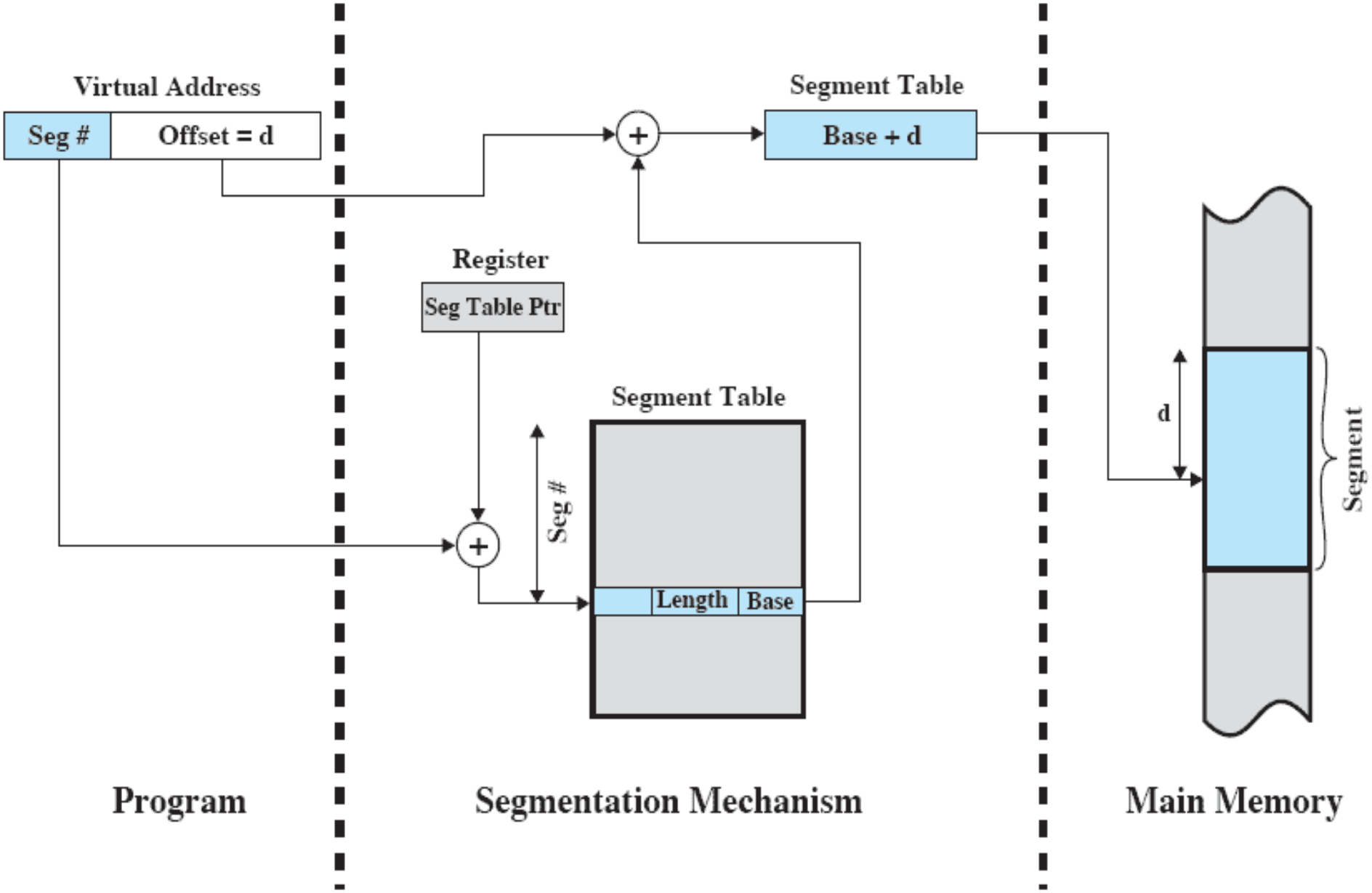


Figure 8.12 Address Translation in a Segmentation System

Combined Paging and Segmentation

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

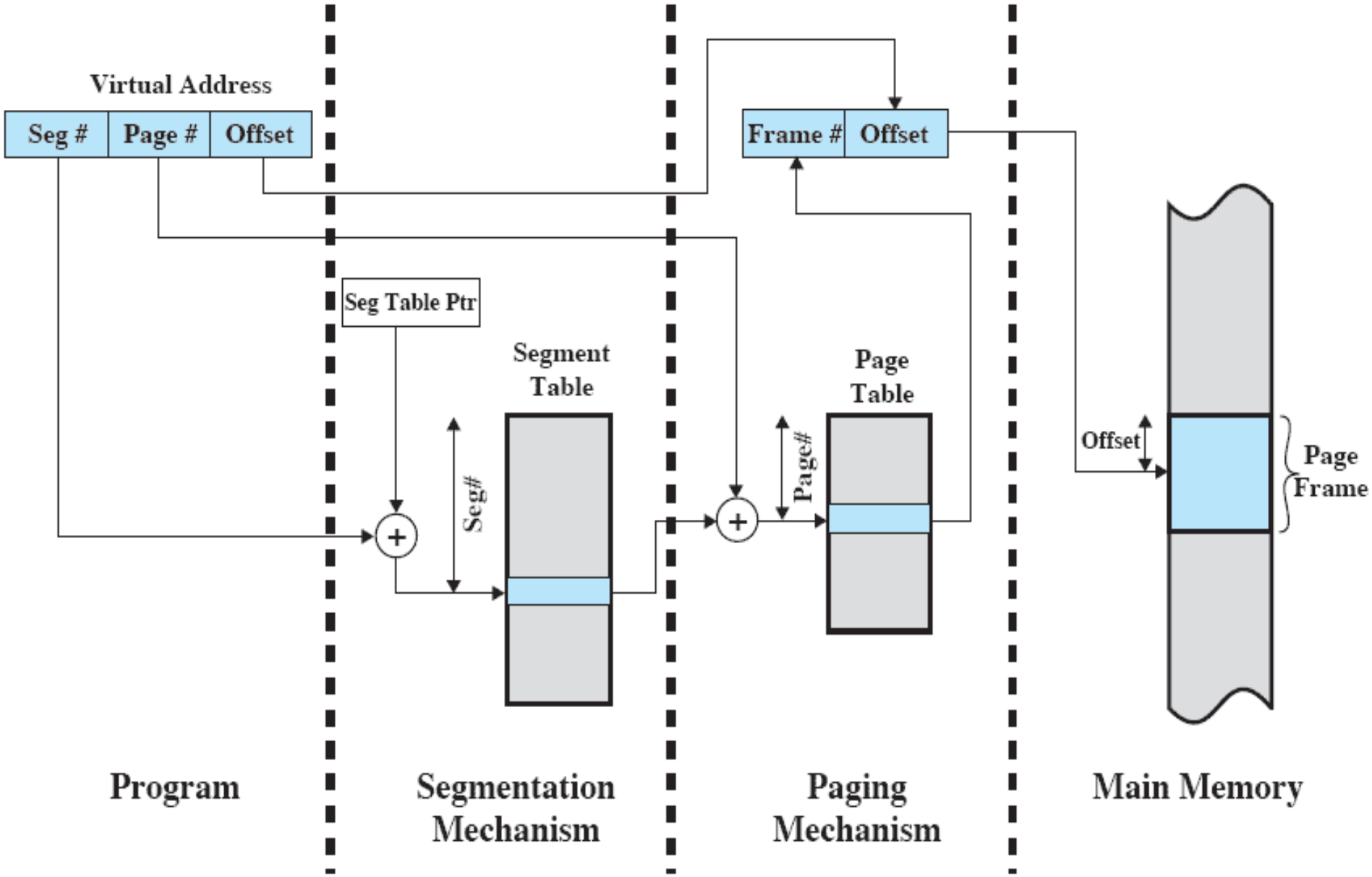


Figure 8.13 Address Translation in a Segmentation/Paging System

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
- **Pre-paging** 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 pure segmentation system, cf. fragmentation
- Less important with paging, but NUMA (non-uniform memory access)

Replacement 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 behaviour on the basis of past behaviour
- Frame Locking
  - Associate a lock bit with each frame, if frame is locked, it may not be replaced
  - Kernel of the operating system, key control structures, I/O buffers

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
- Used as benchmark

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

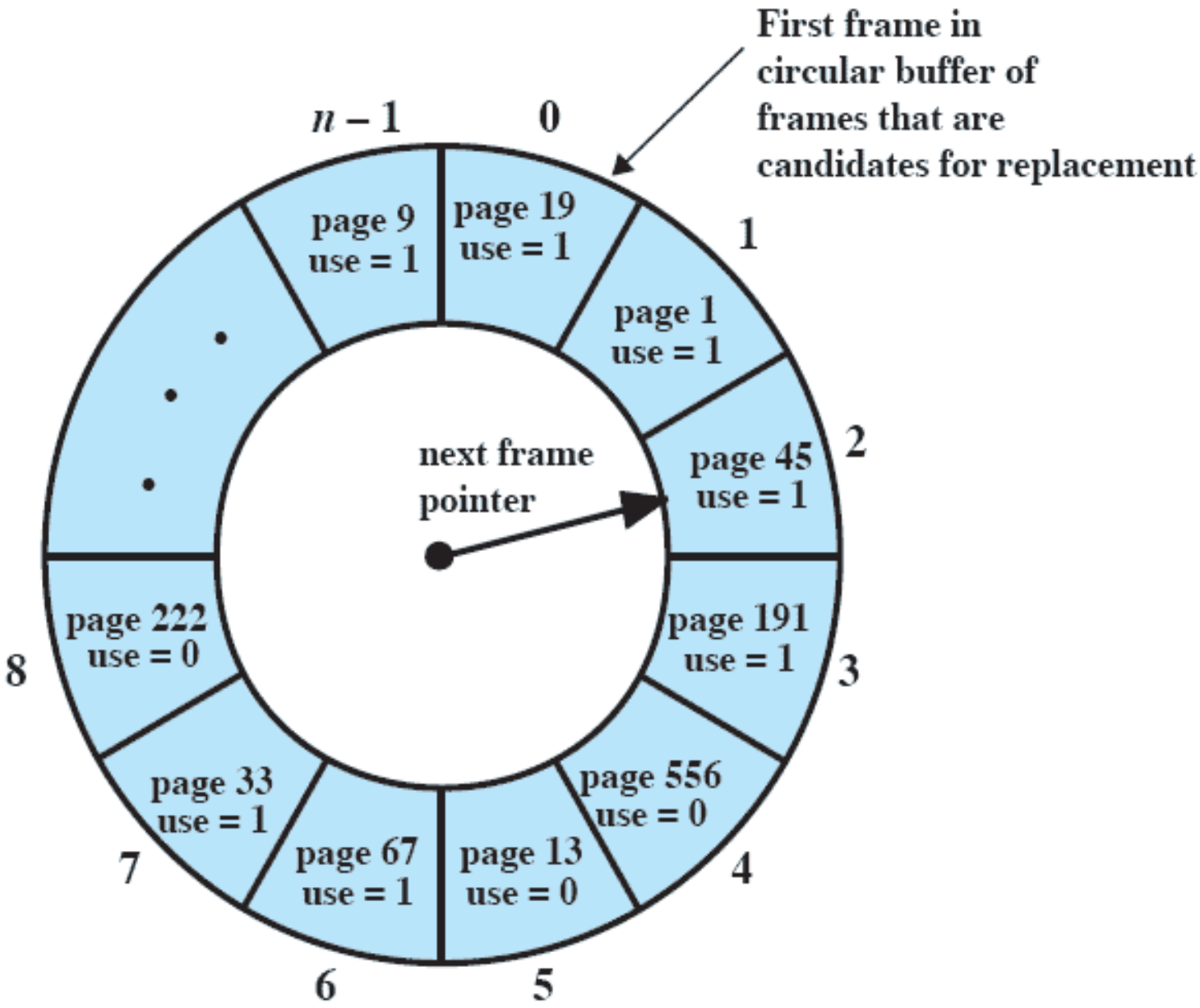
Least Recently Used (LRU)

- 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.

Clock Policy

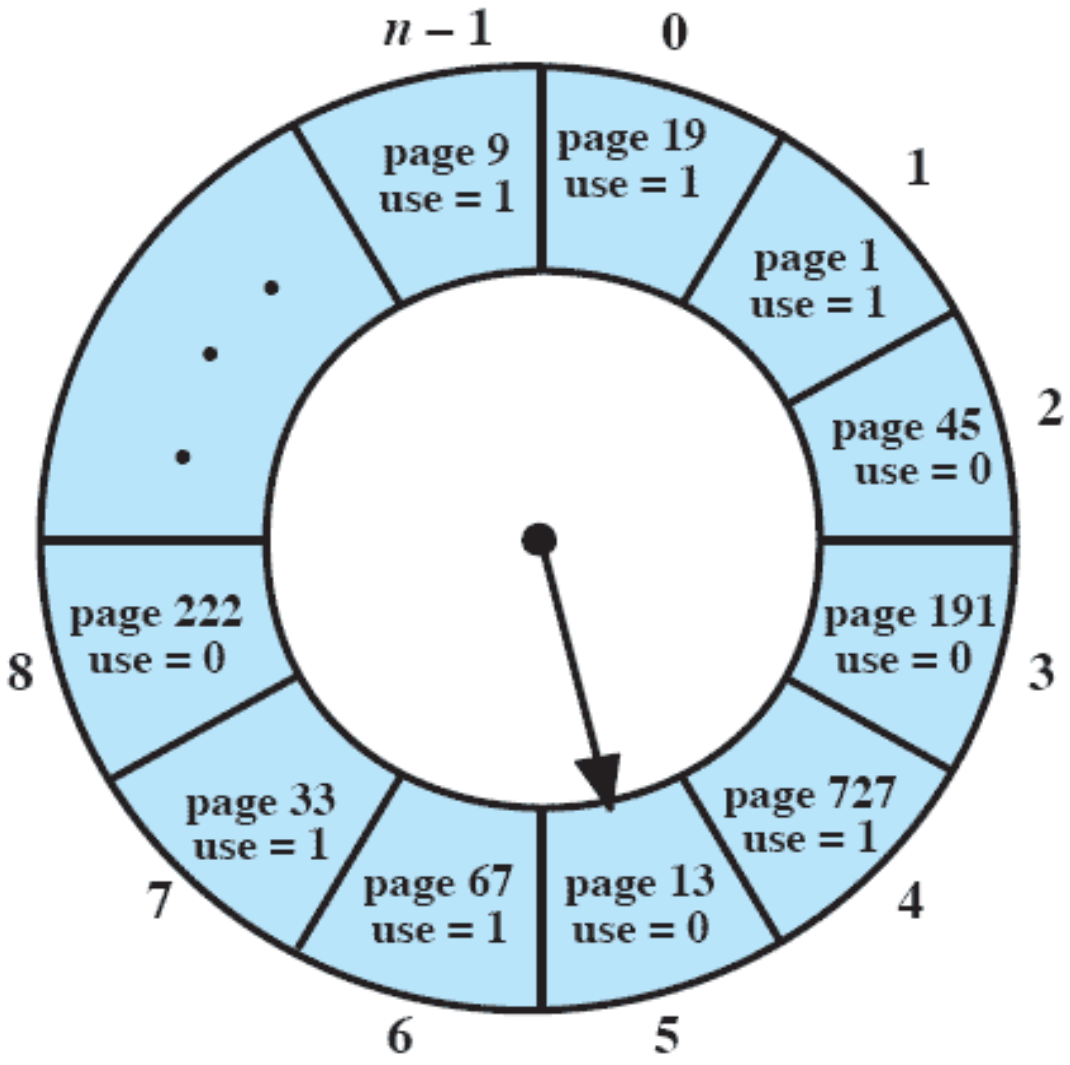
- Additional bit called a use bit
- When a page is first loaded in memory, the use bit is set to 1
- When the page is referenced, the use bit is set to 1
- When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
- During the search for replacement, each use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

Clock Policy



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

Clock Policy



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

Figure 8.16 Example of Clock Policy Operation

Clock with Modified Bit

- Used (u) and modified (m) bits
- 1st round: look for page with u=0 and m=0, do not modify u
- 2nd round: look for page with u=0 and m=1, clear u
- 3rd round: repeat 1st round, then 2nd round
- Saves on writes to hard disk

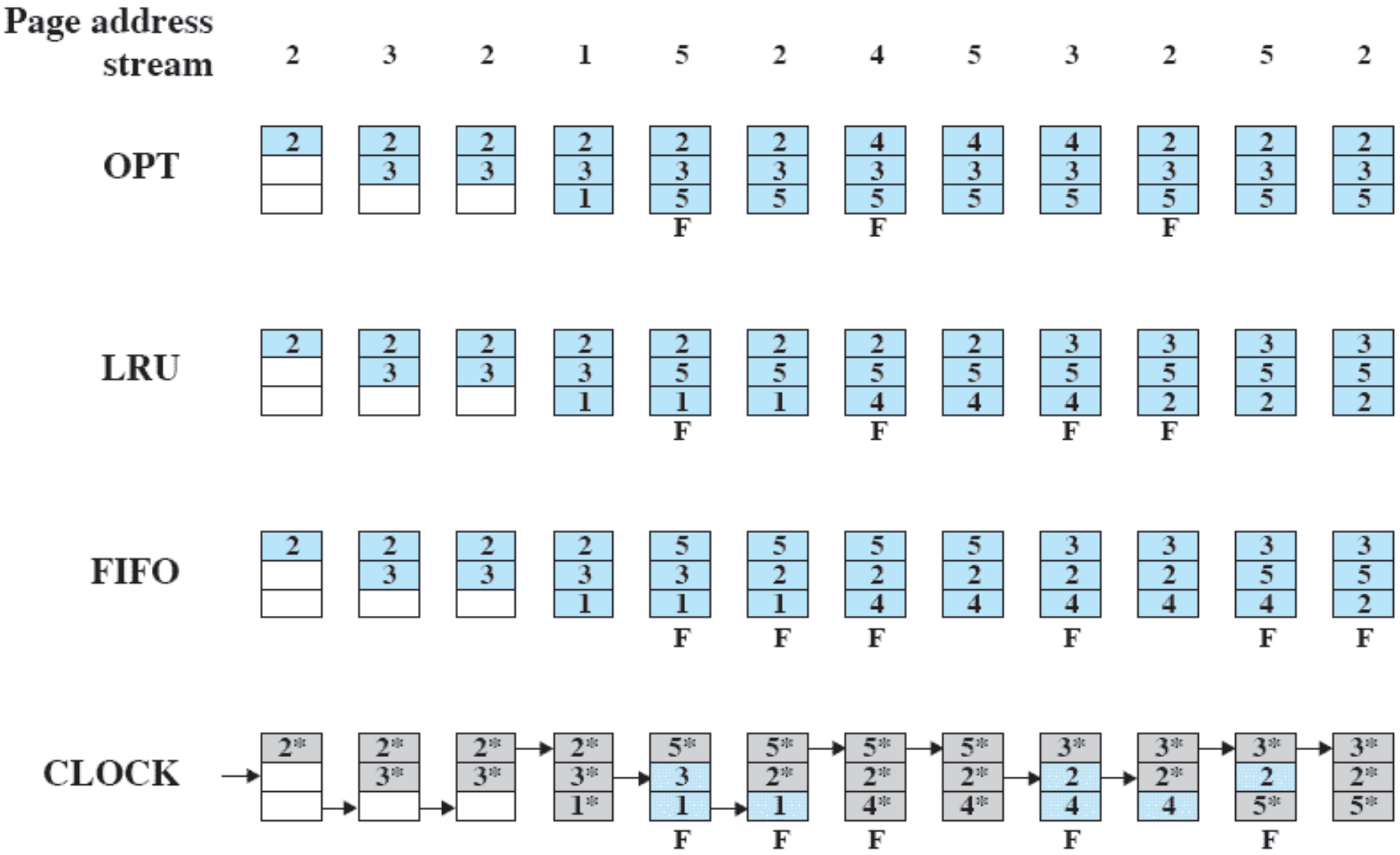


Figure 8.15 Behavior of Four Page-Replacement Algorithms

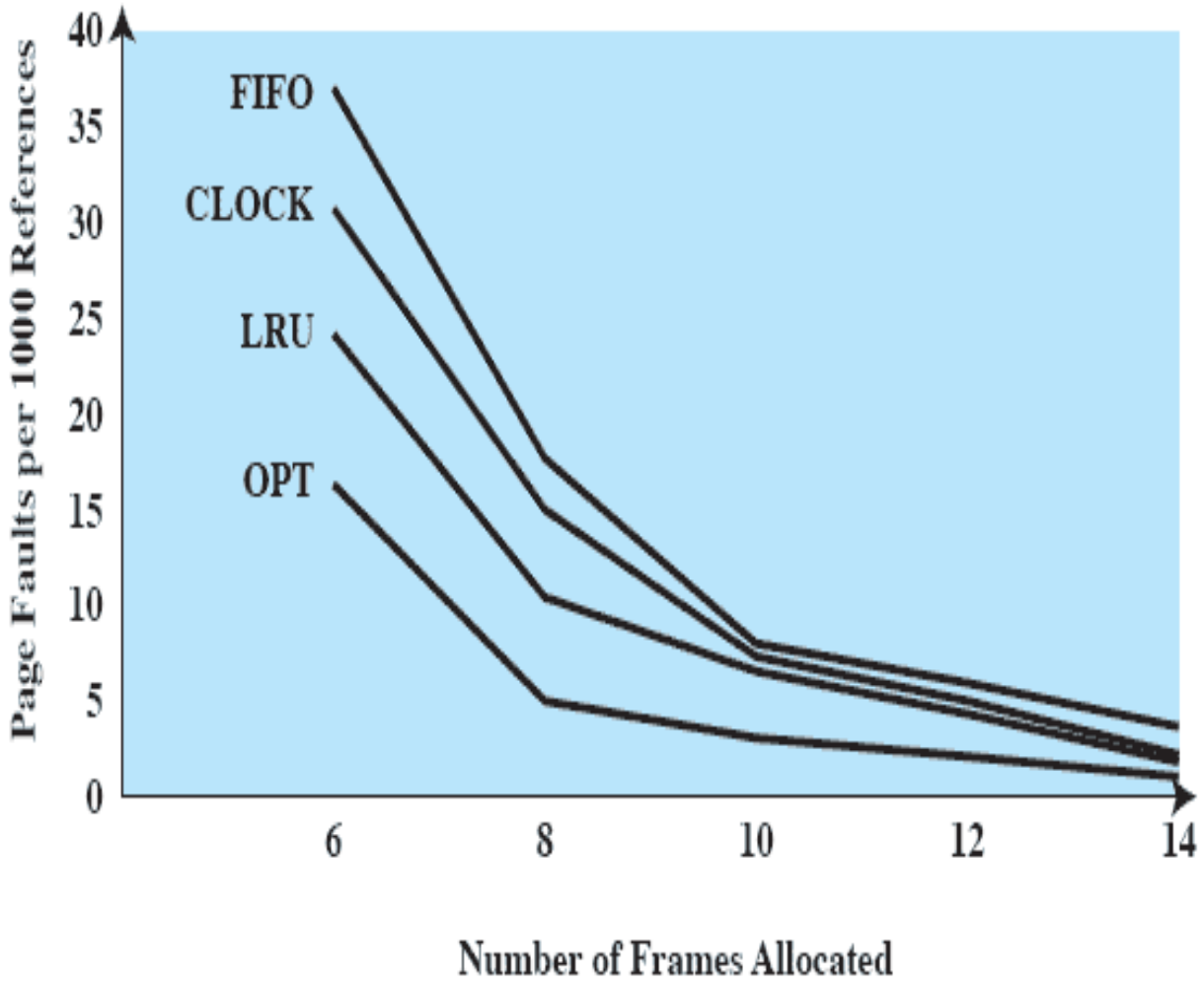


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

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 a small number of programs in main memory
  - Processor idle time
  - Swapping

Variable Allocation, Global Scope

- Easiest to implement
- Adopted by many operating systems (UNIX)
- 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 any process

Variable Allocation, Local Scope

- When new process added, allocate number of page frames based on application type, program request, or other criteria – working set
- When page fault occurs, select page from the resident set of the process that causes the fault
- Re-evaluate allocation from time to time
- Windows

Cleaning Policy

- Demand cleaning
  - A page is written out only when it has been selected for replacement
- Pre-cleaning
  - 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

Small number of processes: many occasions when all processes will be blocked and much time will be spent in swapping

Too many processes: thrashing

Swapper or medium-term scheduler

Multiprogramming

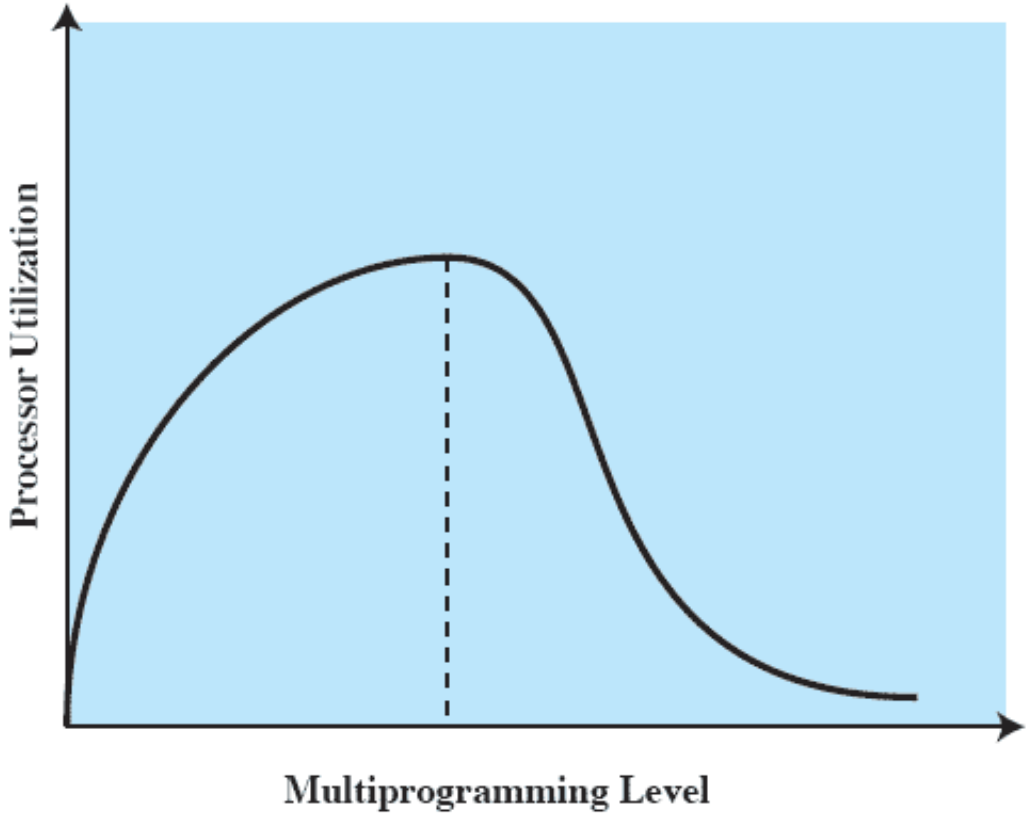


Figure 8.21 Multiprogramming Effects

Process Suspension

- 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