Operating Systems: Internals and Design Principles

Chapter 8
Virtual Memory

Eighth Edition
William Stallings

Hardware and Control Structures

- Two characteristics fundamental to memory management:
  1) All memory references are logical addresses that are dynamically translated into physical addresses at run time
  2) A process may be broken up into a number of pieces that don't need to be contiguously located in main memory during execution

Hardware and Control Structures

- Two characteristics fundamental to memory management:
  1) Dynamic translation of logical to physical addresses
  2) A process may be broken up into a number of pieces
- **It is not necessary that all of the pages or segments of a process be in main memory during execution!**

Execution of a Process

- Operating system brings into main memory only a few pieces of the program and the necessary data
  - Resident set: portion of process that is in main memory
- Execution proceeds
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process into a **Blocked** state

Continued . . .

Execution of a Process

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 is complete, which causes the operating system to place the affected process into the **Ready** state

Virtual Memory Implications

- 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 that some process will be in the Ready state at any particular time
- A process may be larger than all of main memory

Virtual Memory Definitions

- Virtual memory: the process of splitting active processes across primary and secondary storage
- Virtual address space: portion of virtual memory assigned to a process
- Virtual address: the logical address for a piece of information associated with the process. Appears as if it were a physical address
- Real address: the physical address for a piece of information

A Challenge: Thrashing

- A state in which the OS spends more time swapping virtual memory between primary and secondary storage than on actually executing the processes
- This is a serious challenge: to address this, the OS will spend some resources on guessing which parts of virtual memory are least likely to be used in the near future

Principle of Locality

- Only a few pieces of a process will be needed over a short period of time
- Program and data references within a process tend to cluster
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future

Support Needed for Virtual Memory

For virtual memory to be practical and effective:

- Hardware must support paging and segmentation
- Operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

Approaches to Virtual Memory

- Paging: only deal with fixed-size blocks of memory
  - Solves external fragmentation, but subject to internal fragmentation
- Segmentation:
  - Solves internal fragmentation, but subject to external fragmentation
  - Limits on segment sizes can be substantial
- Hybrid paging and segmentation: compromise between the two

Virtual Address

Page Number	Offset

Page Table Entry

P	M	Other Control Bits	Frame Number

(a) Paging only

Virtual Address

Segment Number	Offset

Segment Table Entry

P	M	Other Control Bits	Length	Segment Base

(b) Segmentation only

Virtual Address

Segment Number	Page Number	Offset

Segment Table Entry

Control Bits	Length	Segment Base

Page Table Entry

P	M	Other Control Bits	Frame Number

P= present bit
M = Modified bit

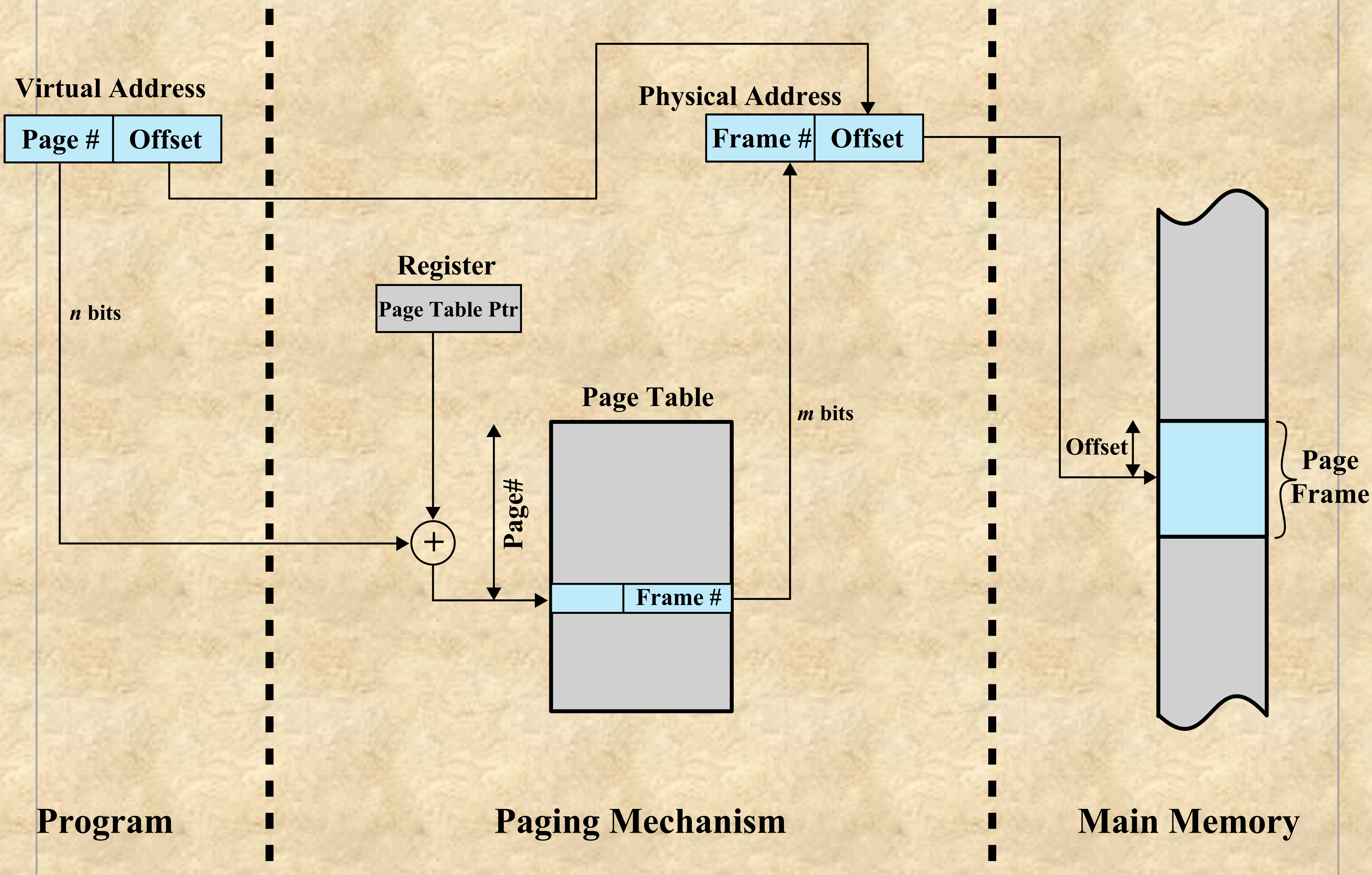
(c) Combined segmentation and paging

Frame Table Entries

Includes:

- Frame number
- P control bit: is the page in main memory or not?
- M control bit: has the main memory copy of the page been modified?

Segmentation tables maintain similar information



Virtual Address
Page #
Offset
Physical Address
Frame #
Offset
Register
Page Table Ptr
n bits
Page Table
m bits
Page#
Frame #
Offset
Page Frame
Program
Paging Mechanism
Main Memory

Table Challenges

- Virtual memory can be rather large
- This means that we need very large page/segment tables
  - Big waste of space, especially for small processes
- The fix: hierarchical tables

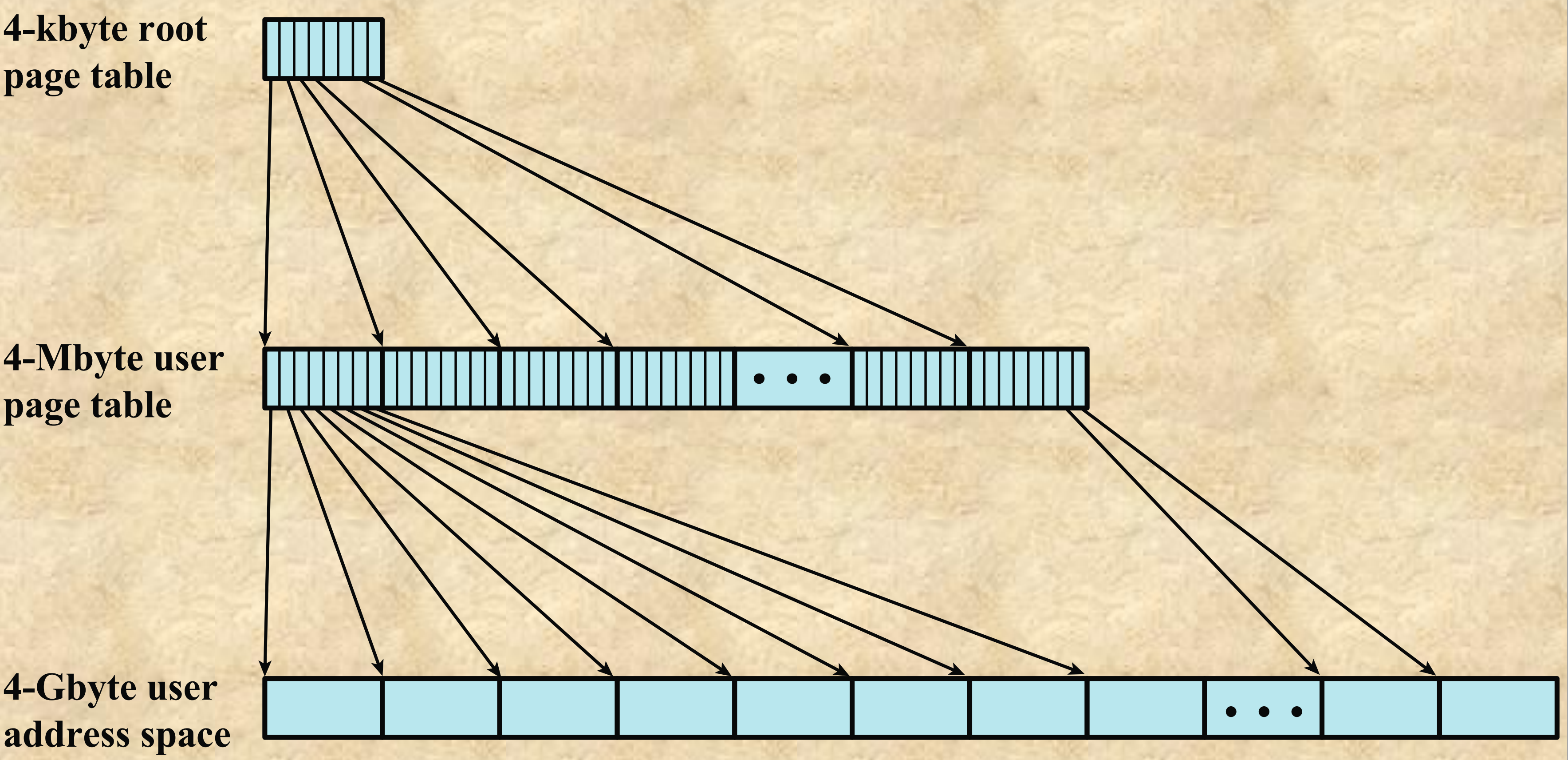


Figure 8.3 A Two-Level Hierarchical Page Table

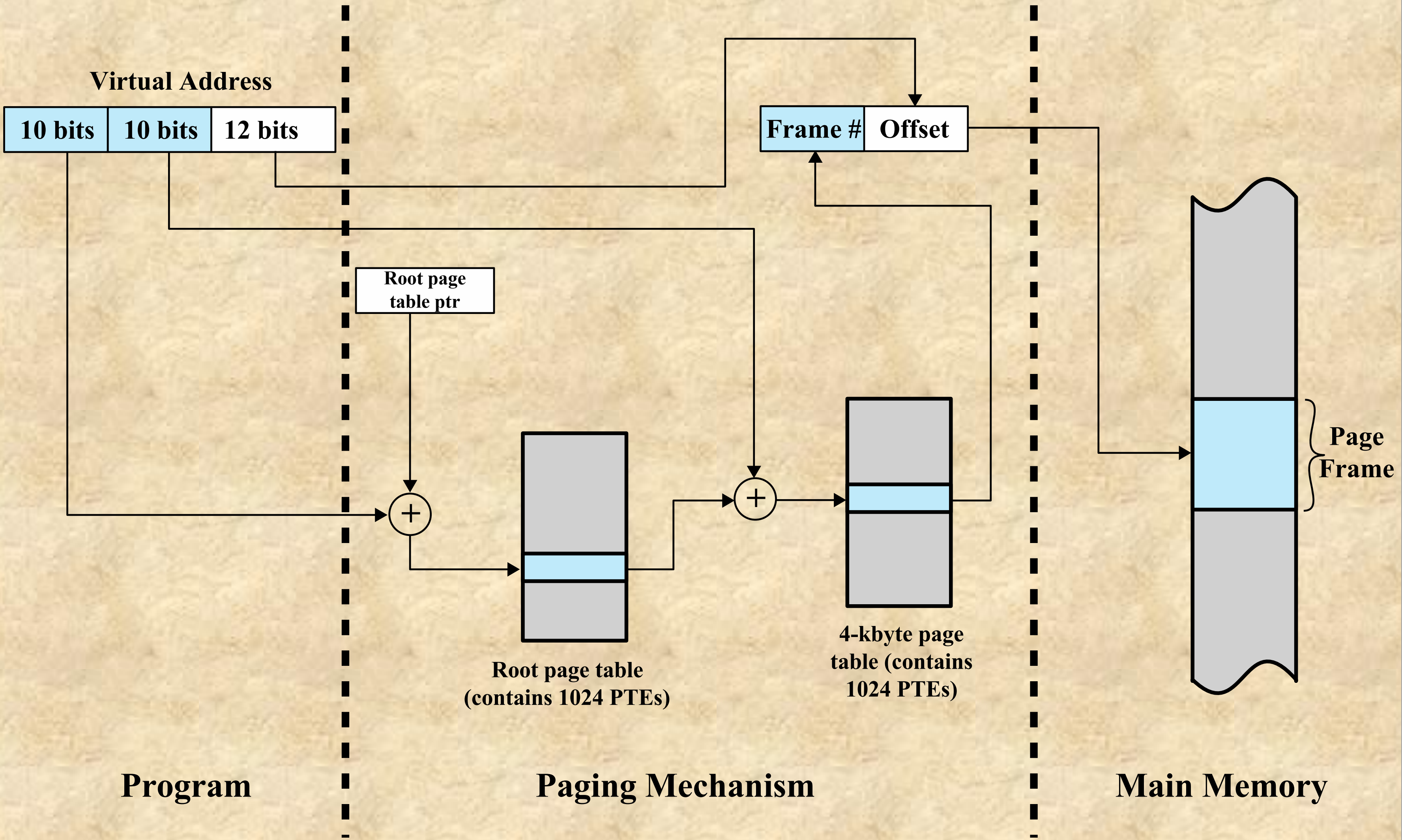


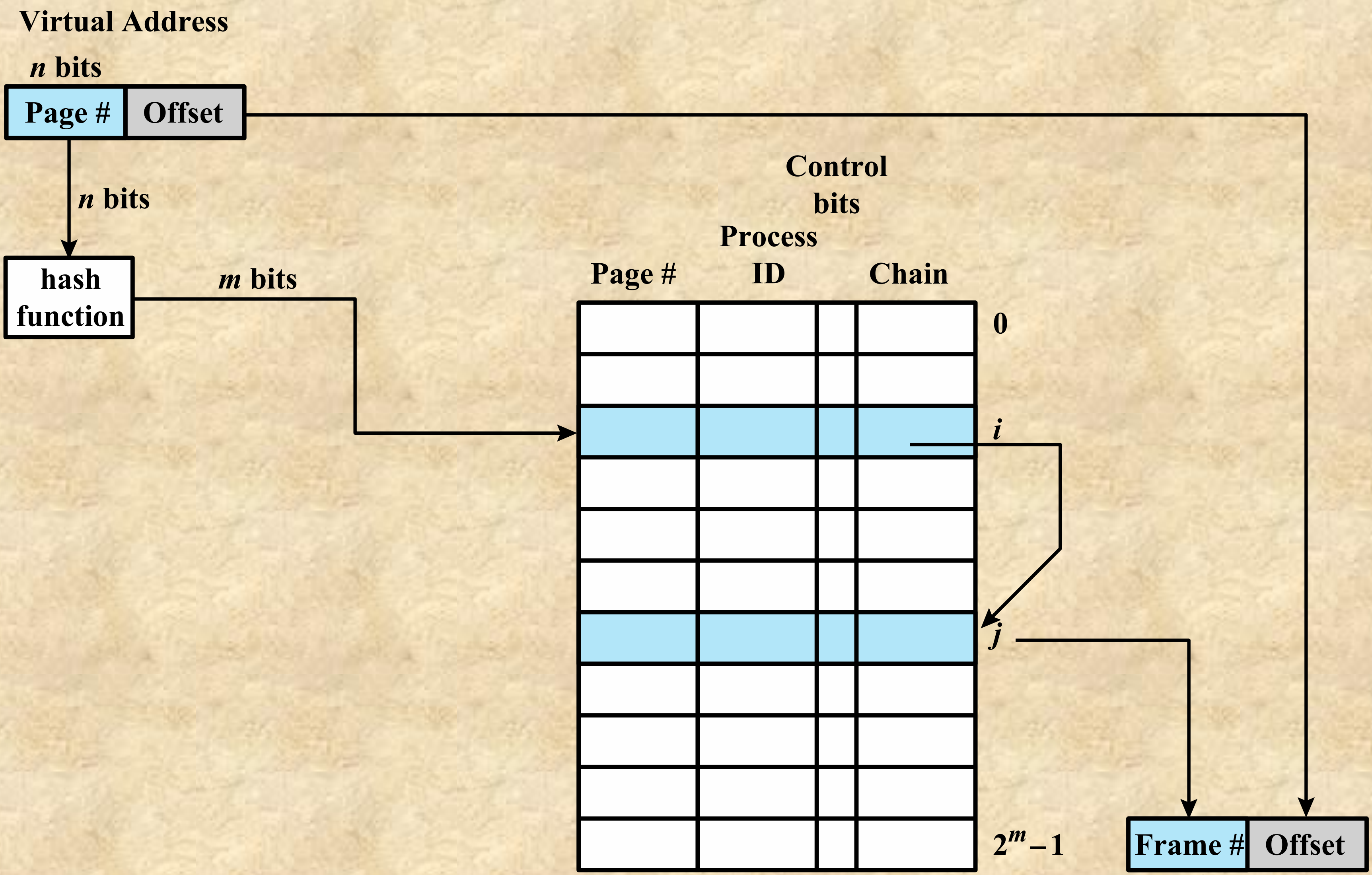
Figure 8.4 Address Translation in a Two-Level Paging System

Hierarhical Tables

- Total size of all tables is much smaller than with monolithic tables
- But, the size must be big enough to cover all of the virtual memory space for the process (which is still relatively large)

Inverted Page Table

- Page number portion of a virtual address is mapped by a hash value
  - The hash value is the index into the inverted page table
  - The inverted page table entry maintains a pointer to the first candidate frame
  - Collisions are handled through additional chaining to other table entries
- Need only one table entry per physical memory frame



Inverted Page Table
(one entry for each
physical memory frame)

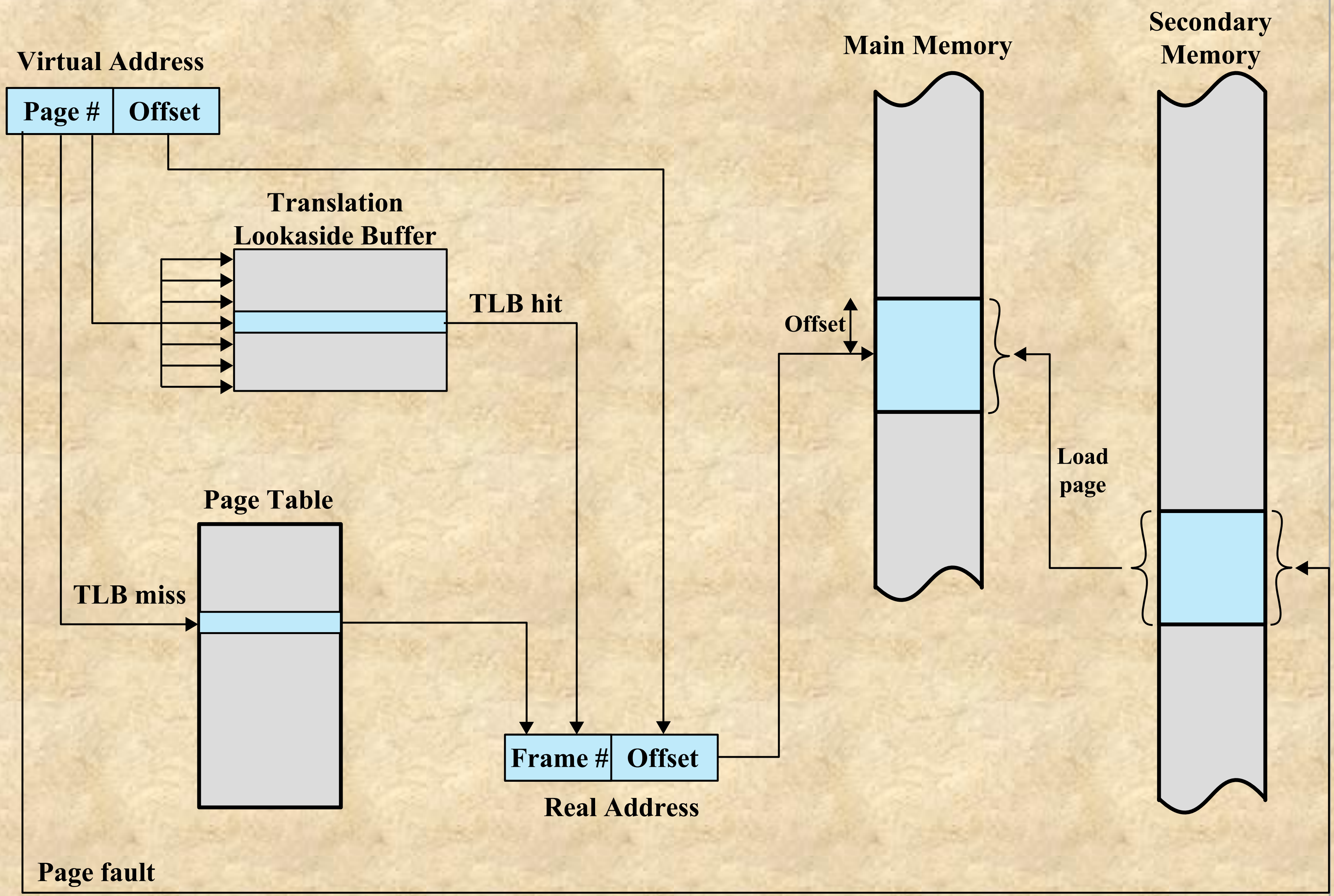
Inverted Page Table

Algorithm:

- Hash function: n bits -> m bits
- Compare n bits to table entry n bits
  - If match, then the m bits tell us the frame #; append this to the offset and we are done
  - If no match, then follow the chain. Repeat comparison
  - If at the end of the chain: raise an interrupt

Translation Look-aside Buffer (TLB)

- Up to now: a memory access by the program actually requires at least two memory accesses:
  - Look up the page table entry
  - Actually access the memory
- Translation Look-aside Buffer adds:
  - A cache for the page table access
  - Look-up is associative



Virtual Address
Page #
Offset
Translation
Lookaside Buffer
TLB hit
TLB miss
Page Table
Frame #
Offset
Real Address
Main Memory
Offset
Load
page
Secondary
Memory
Page fault

Translation Look-aside Buffer (TLB)

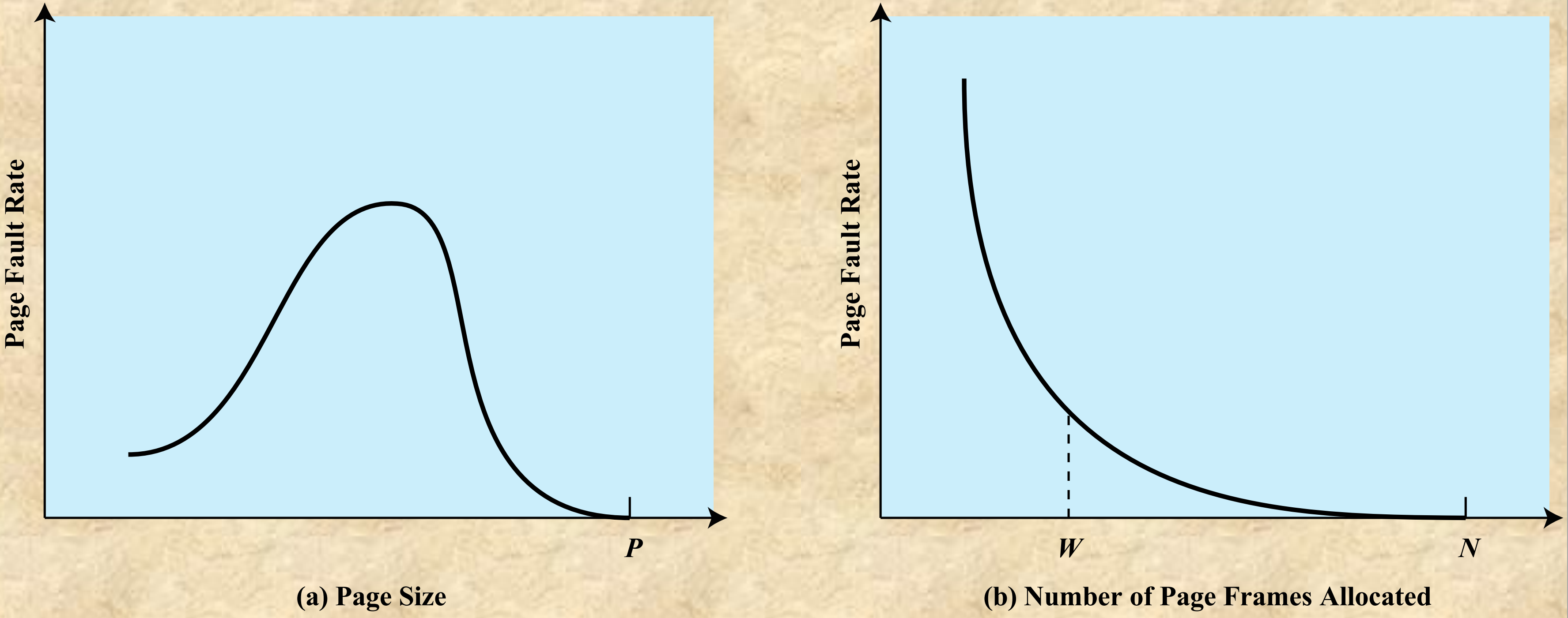
Algorithm: Does the TLB cache contain an entry for the page #?

- Yes (cache hit): append entry's frame # to the offset & we are done
- No (cache miss): fetch the page table entry from memory
  - Append frame # to the offset
  - Copy entry into the cache
- If there is no page table entry: page fault
  - Must request that the page be fetched from secondary memory

Page Size

The smaller the page size, the lesser the internal fragmentation

- However, more pages are required per process
- More pages per process means larger page tables
- For large programs in a heavily multiprogrammed environment, some portion of the page tables of active processes must be in virtual memory instead of main memory
- The physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer of data



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

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes 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

Table 8.3

Example Page Sizes

Page Size

the design issue of page size is related to the size of physical main memory and program size

main memory is getting larger and address space used by applications is also growing

most obvious on personal computers where applications are becoming increasingly complex

- Contemporary programming techniques used in large programs tend to decrease the locality of references within a process

Segmentation

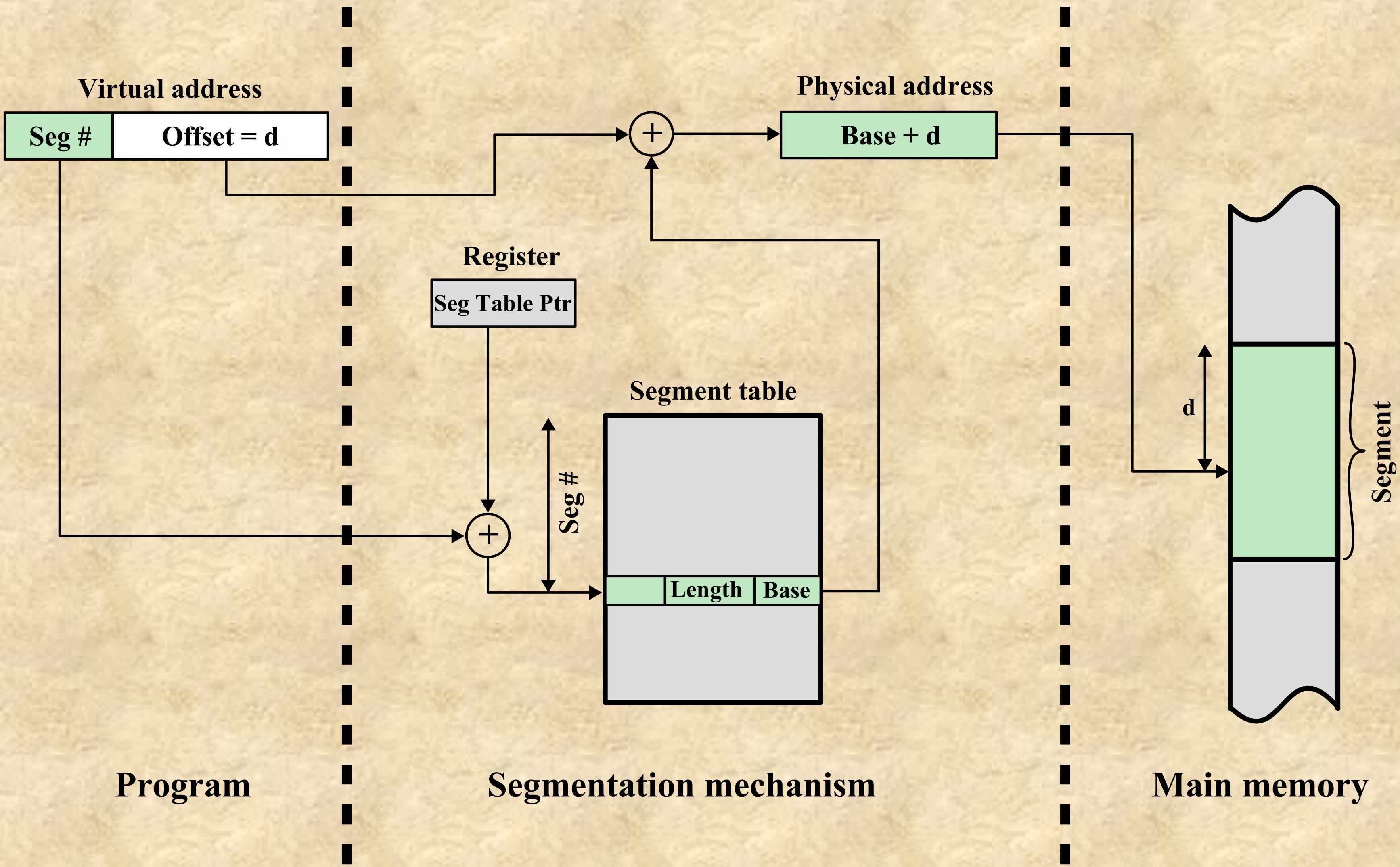
Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments

Advantages:

- 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 Organization

- Each segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment
- A bit is needed to determine if the 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



Virtual address
Seg #
Offset = d
Physical address
Base + d
Register
Seg Table Ptr
Segment table
Seg #
Length
Base
d
Segment
Program
Segmentation mechanism
Main memory

Combined Paging and Segmentation

In a combined paging/segmentation system a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages which are equal in length to a main memory frame

- Segmentation is visible to the programmer
- Paging is transparent to the programmer

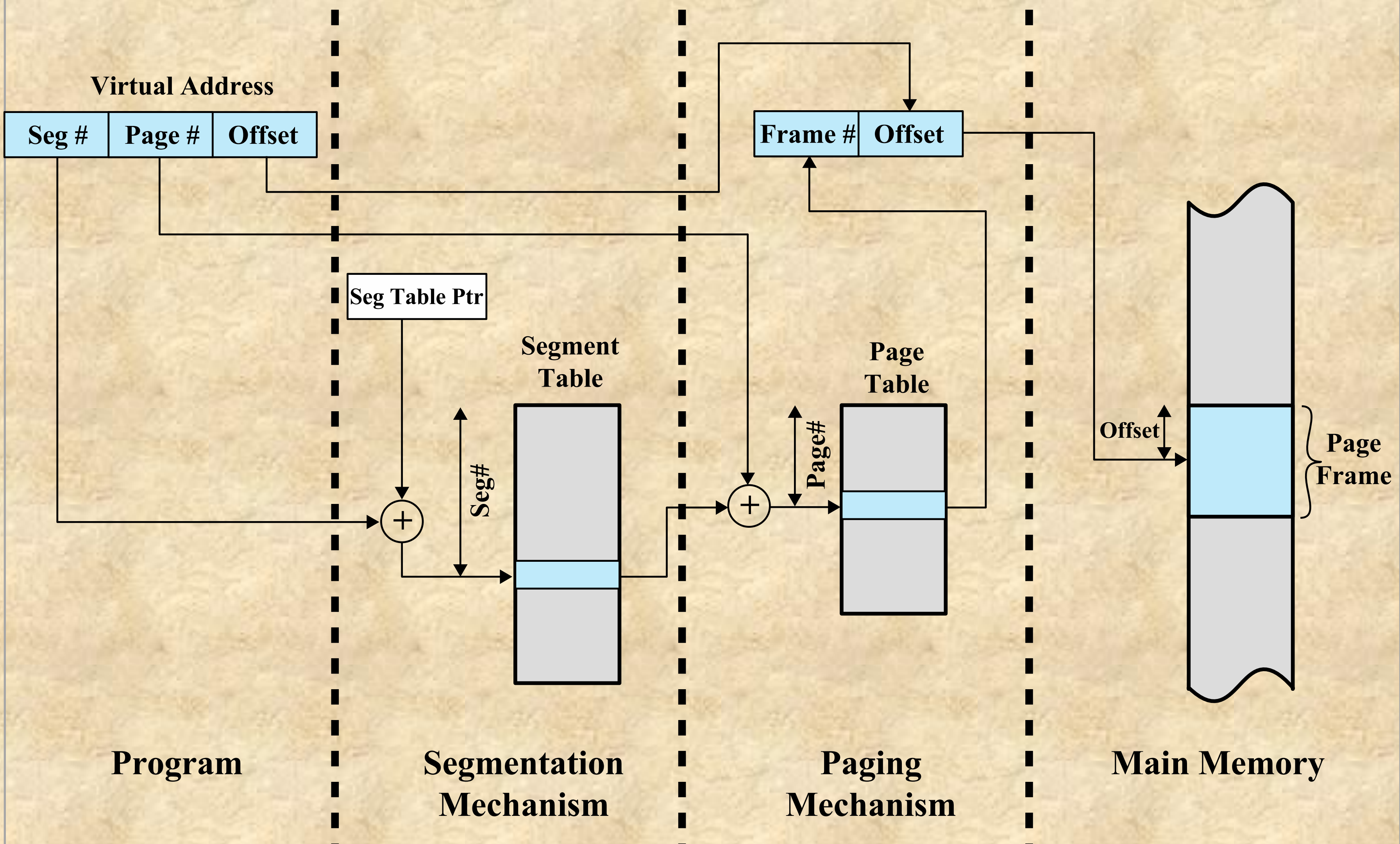


Figure 8.12 Address Translation in a Segmentation/Paging System

Virtual Address

Segment Number	Page Number	Offset

Segment Table Entry

Control Bits	Length	Segment Base

Page Table Entry

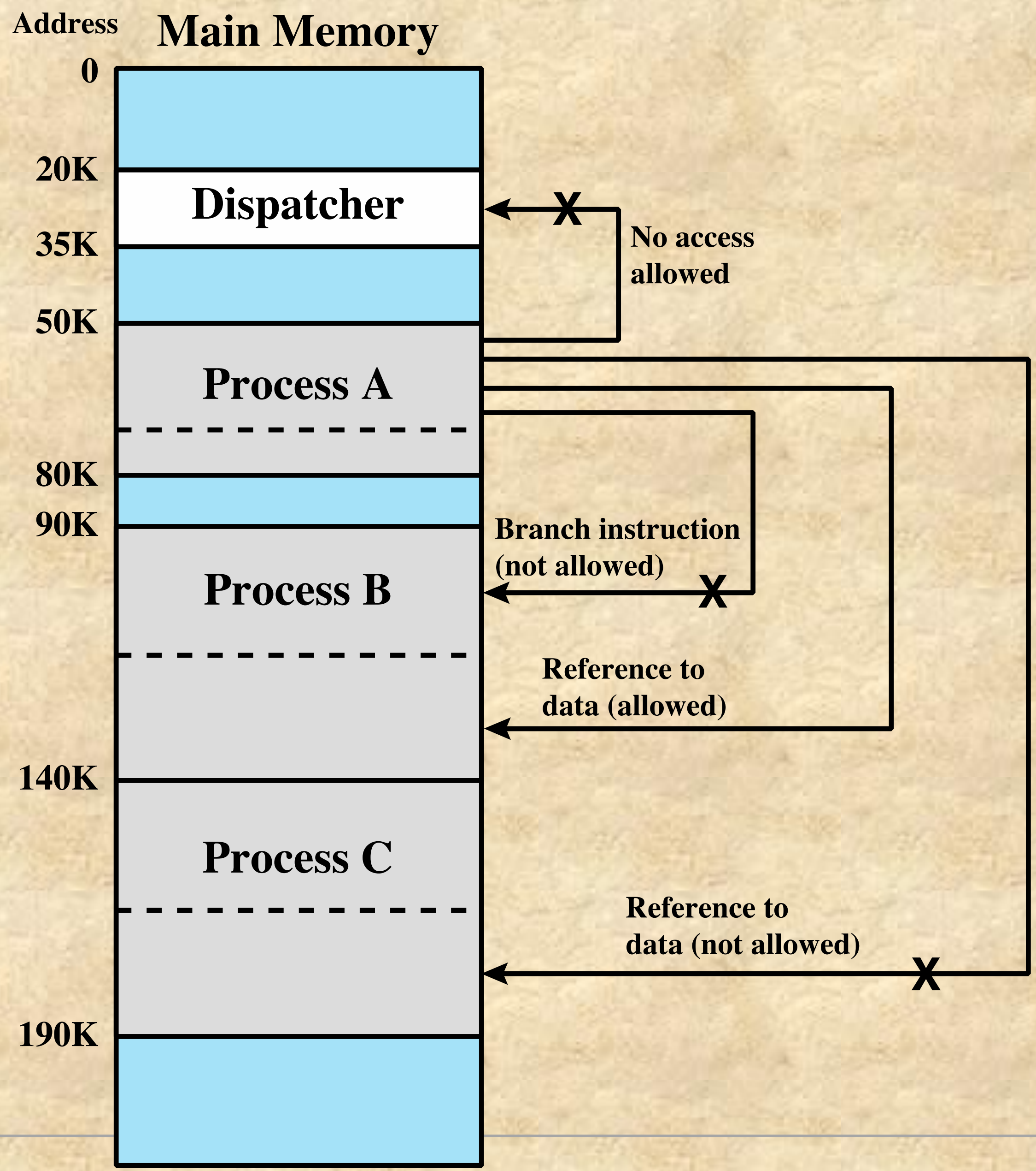
P	M	Other Control Bits	Frame Number

P= present bit
M = Modified bit

(c) Combined segmentation and paging

Protection and Sharing

- Segmentation lends itself to the implementation of protection and sharing policies
- Each entry has a base address and length so inadvertent memory access can be controlled
- Sharing can be achieved by multiple processes referencing the same segment


Address
Main Memory
0
20K
Dispatcher
35K
50K
Process A
80K
90K
Process B
140K
Process C
190K
No access
allowed
Branch instruction
(not allowed)
Reference to
data (allowed)
Reference to
data (not allowed)

Operating System Software

The design of the memory management portion of an operating system depends on three fundamental areas of choice:

- Whether or not to use virtual memory techniques
- The use of paging or segmentation … or both
- The algorithms employed for various aspects of memory management

Fetch Policy Demand paging Prepaging **Placement Policy** **Replacement Policy** Basic Algorithms Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page Buffering	**Resident Set Management** Resident set size Fixed Variable Replacement Scope Global Local **Cleaning Policy** Demand Precleaning **Load Control** Degree of multiprogramming

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy

Determines when a page should be brought into memory

Two main types:

- Demand Paging
- Prepaging

Demand Paging

- Only brings pages into main memory when a reference is made to a location on the page
- Many page faults when process is first started
- Principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level

Prepaging

- Pages other than the one demanded by a page fault are brought in
- Exploits the characteristics of most secondary memory devices
  - If pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
- Ineffective if extra pages are not referenced in the near future
- Should not be confused with "swapping"

Replacement Policy

- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
  - The goal is that the page that is removed be the page least likely to be referenced in the near future
- The more elaborate the replacement policy the greater the hardware and software overhead to implement it

Basic Algorithms

Algorithms used for the selection of a page to replace:

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)
- Clock

Least Recently Used (LRU)

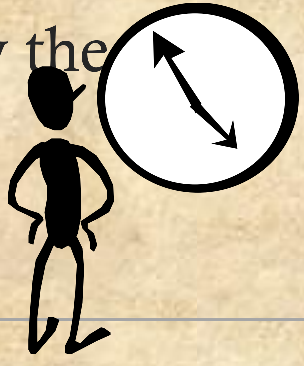
- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement
  - One approach is to tag each page with the time of last reference
  - This requires a great deal of overhead

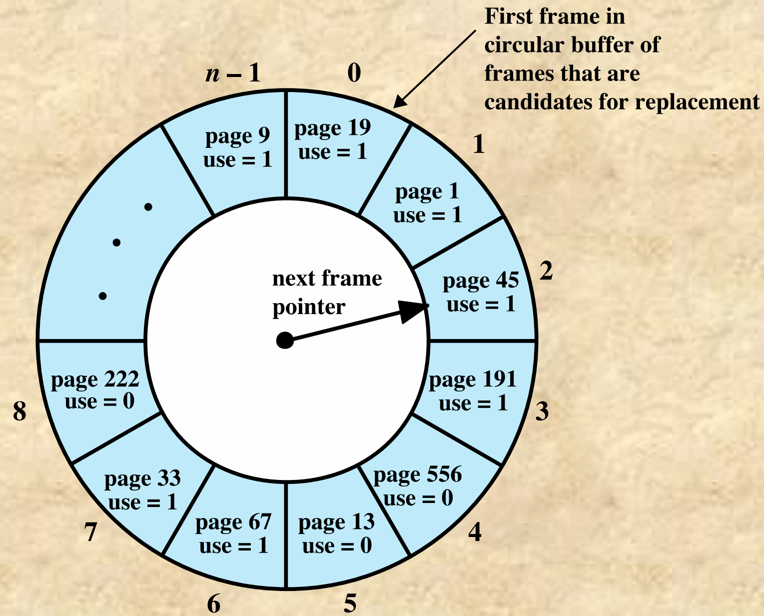
First-in-First-out (FIFO)

- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
  - Simple replacement policy to implement
- Page that has been in memory the longest is replaced

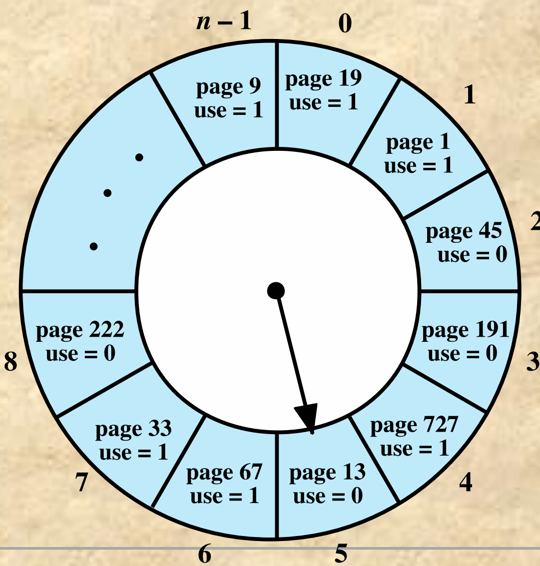
Clock Policy

- Requires the association of an additional bit with each frame
  - referred to as the ***use*** bit
- When a page is first loaded in memory or referenced, the use bit is set to 1
- The set of frames is considered to be a circular buffer
- Any frame with a use bit of 1 is passed over by the algorithm
- Page frames visualized as laid out in a circle

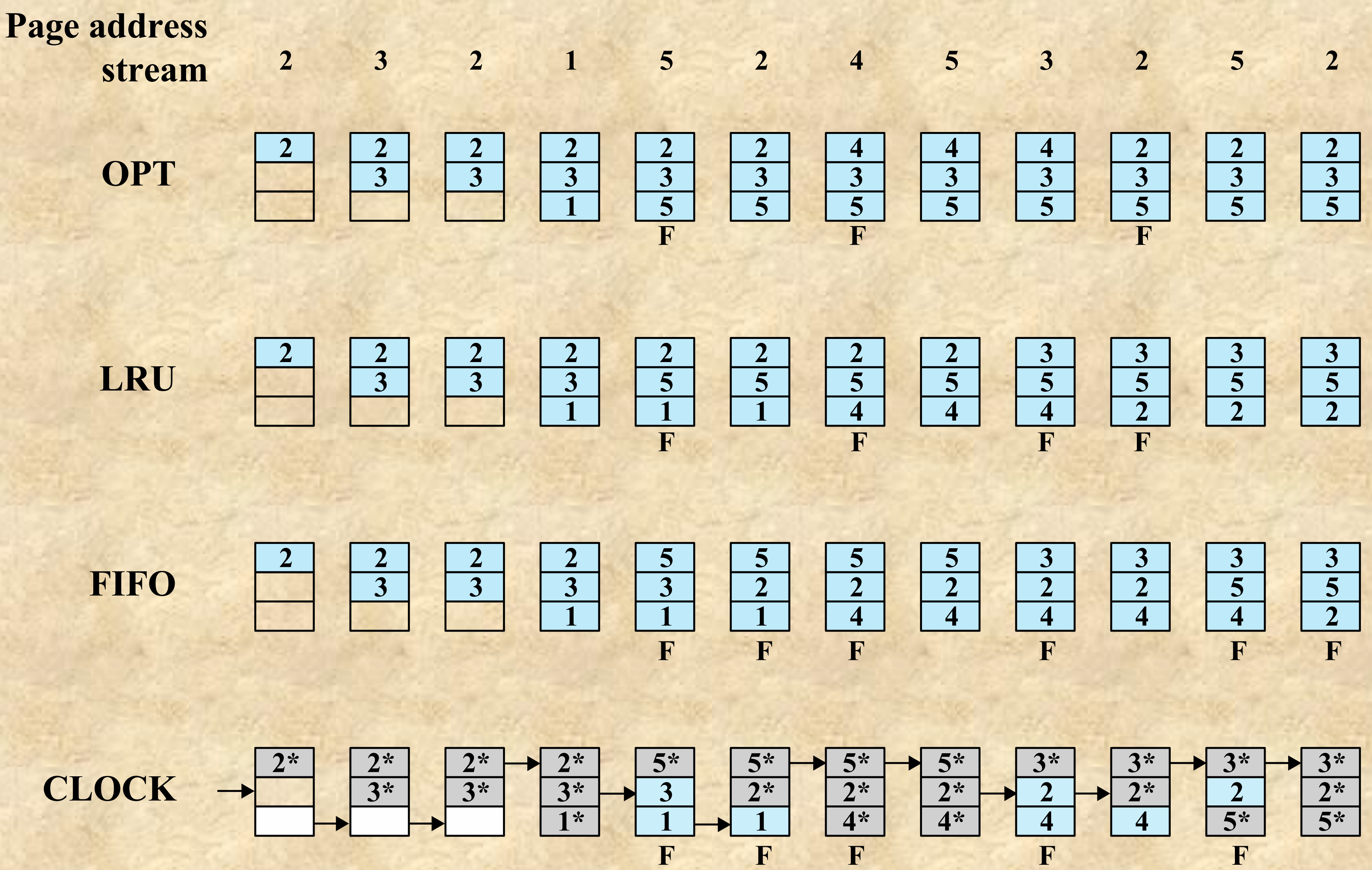




(a) State of buffer just prior to a page replacement



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



F = page fault occurring after the frame allocation is initially filled

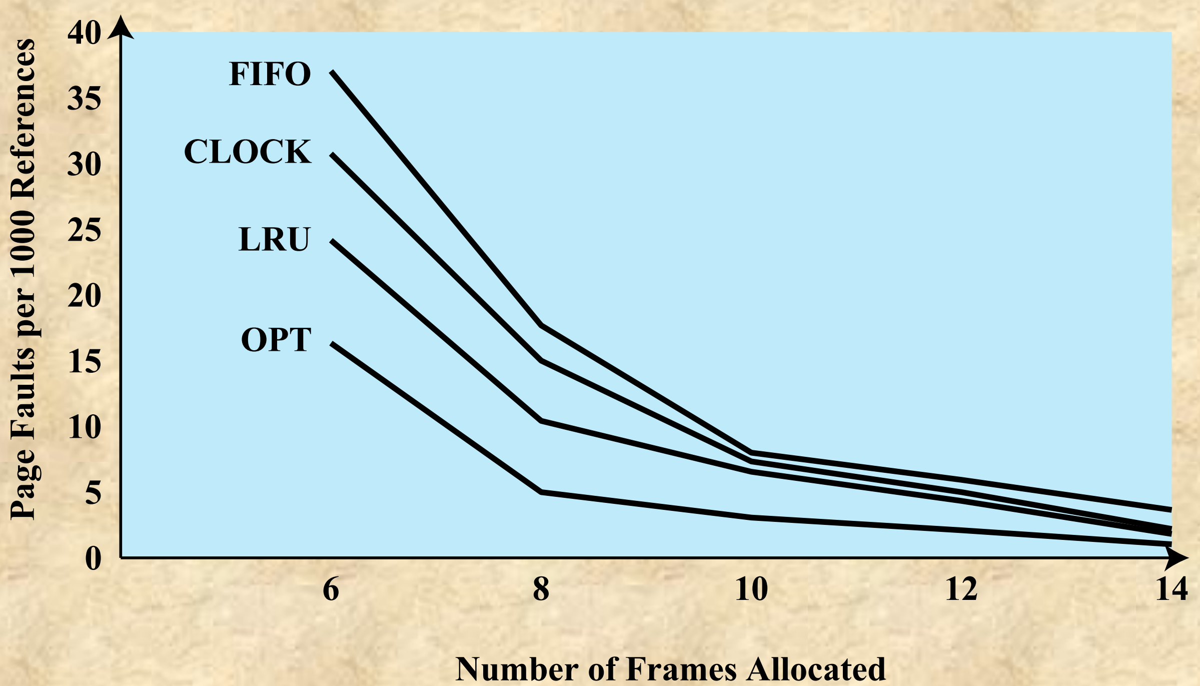


Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Summary

- Translating logical addresses to physical ones
  - Page tables, segment tables, inverted page tables translation lookahead buffers
  - Multi-level tables
- Fetch policies
- Replacement policies