

*Operating
Systems:
Internals
and Design
Principles*

Chapter 8 Virtual Memory

Ninth Edition
William Stallings

Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

Table 8.1 Virtual Memory Terminology

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
- If these two characteristics are present, 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 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

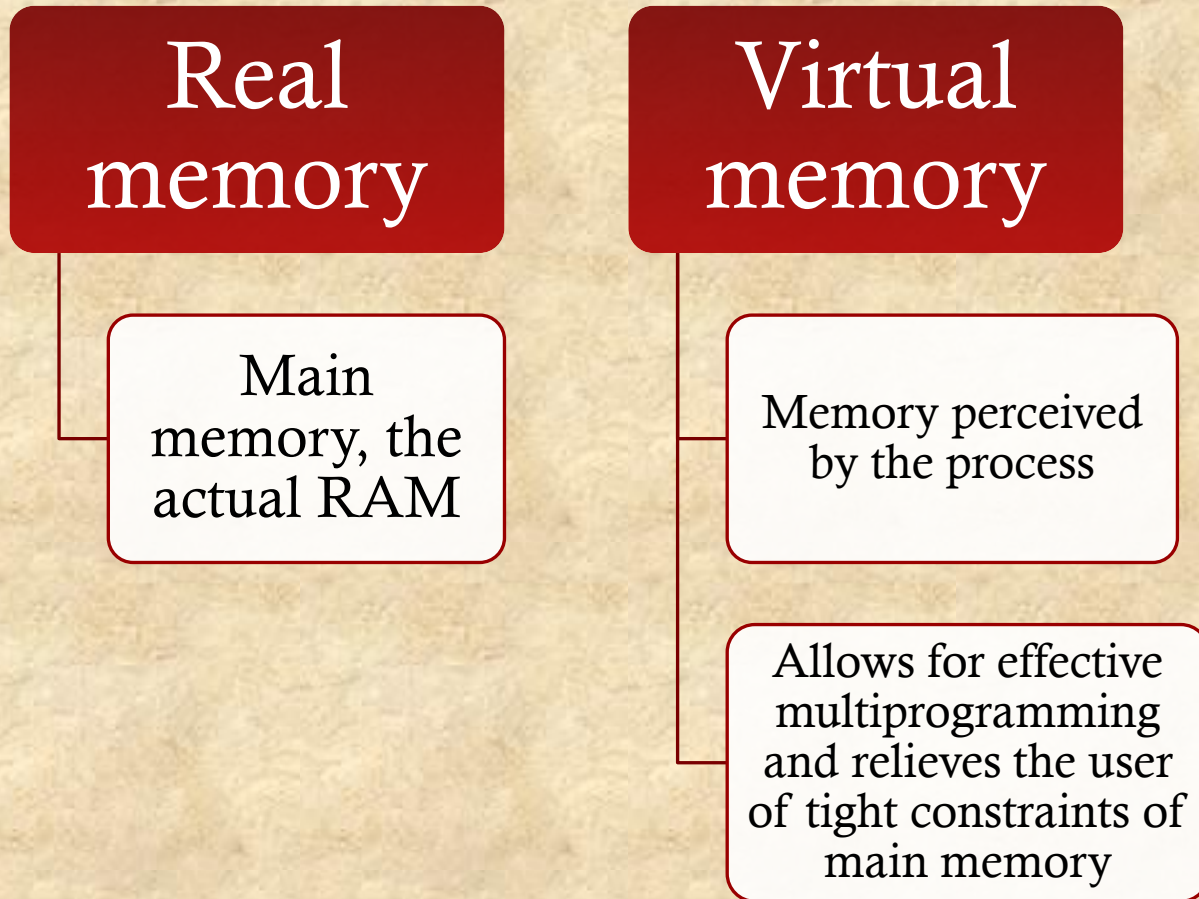
Execution of a Process

- 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 in the Ready state

Implications

- More processes may be maintained in main memory
 - Because only some of the pieces of any particular process are loaded, there is room for more processes
 - This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in a Ready state at any particular time
- A process may be larger than all of main memory
 - If the program being written is too large, the programmer must devise ways to structure the program into pieces that can be loaded separately in some sort of overlay strategy
 - With virtual memory based on paging or segmentation, that job is left to the OS and the hardware
 - The OS automatically loads pieces of a process into main memory as required

Real and Virtual Memory

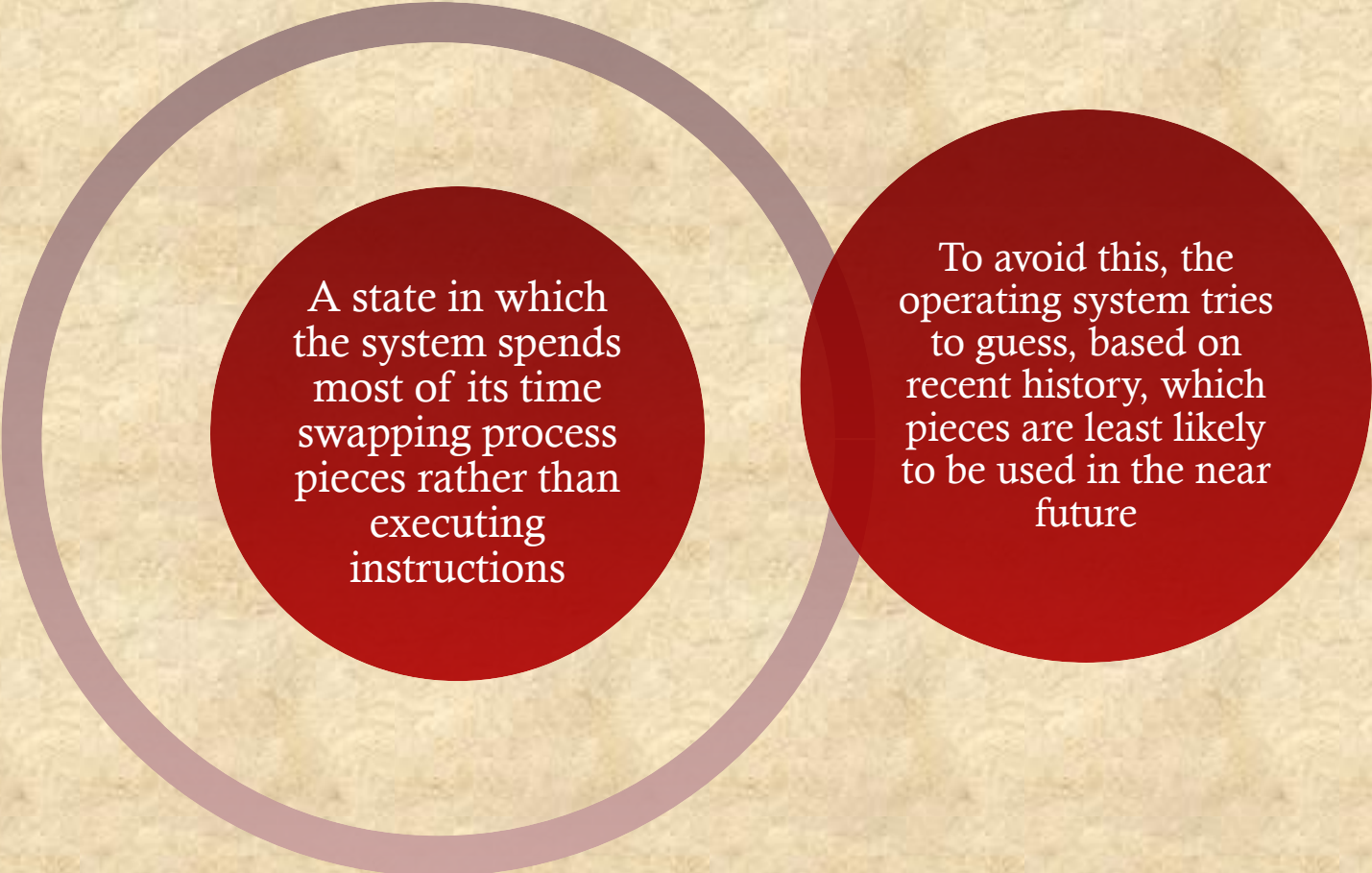


Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames		Main memory not partitioned	
Program broken into pages by the compiler or memory management system		Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	
Internal fragmentation within frames		No internal fragmentation	
No external fragmentation		External fragmentation	
Operating system must maintain a page table for each process showing which frame each page occupies		Operating system must maintain a segment table for each process showing the load address and length of each segment	
Operating system must maintain a free frame list		Operating system must maintain a list of free holes in main memory	
Processor uses page number, offset to calculate absolute address		Processor uses segment number, offset to calculate absolute address	
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Table 8.2

Characteristics of Paging and Segmentation

Thrashing



A diagram illustrating the concept of thrashing. It features a large, light purple circle on the left and a smaller, solid dark red circle on the right. The large circle contains a smaller dark red circle in its center. The text 'A state in which the system spends most of its time swapping process pieces rather than executing instructions' is written in white inside the central dark red circle. The text 'To avoid this, the operating system tries to guess, based on recent history, which pieces are least likely to be used in the near future' is written in white inside the right-hand dark red circle.

A state in which the system spends most of its time swapping process pieces rather than executing instructions

To avoid this, the operating system tries to guess, based on recent history, 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
- Only a few pieces of a process will be needed over a short period of time
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- Avoids thrashing

Support Needed for Virtual Memory

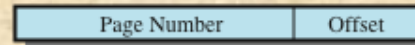
For virtual memory to be practical and effective:

- 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

Paging

- The term *virtual memory* is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the Atlas computer
- Each process has its own page table
 - Each page table entry (PTE) contains the frame number of the corresponding page in main memory
 - A page table is also needed for a virtual memory scheme based on paging

Virtual Address

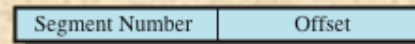


Page Table Entry

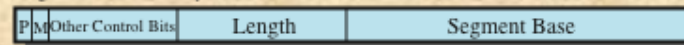


(a) Paging only

Virtual Address

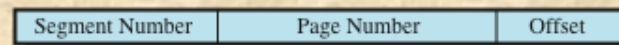


Segment Table Entry



(b) Segmentation only

Virtual Address



Segment Table Entry



Page Table Entry



P= present bit
M = Modified bit

(c) Combined segmentation and paging

Figure 8.1 Typical Memory Management Formats

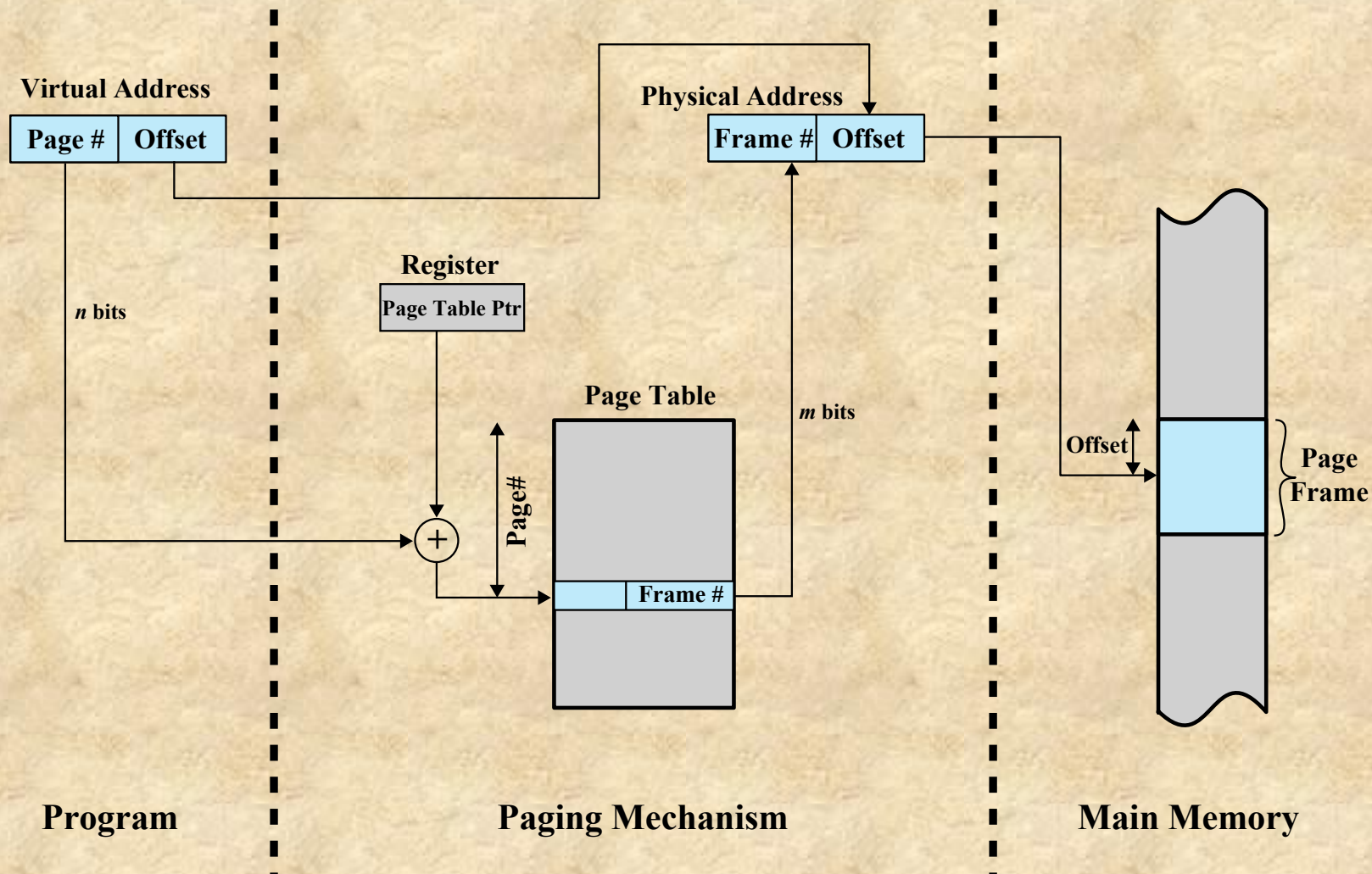


Figure 8.2 Address Translation in a Paging System

**4-kbyte root
page table**

**4-Mbyte user
page table**

**4-Gbyte user
address space**

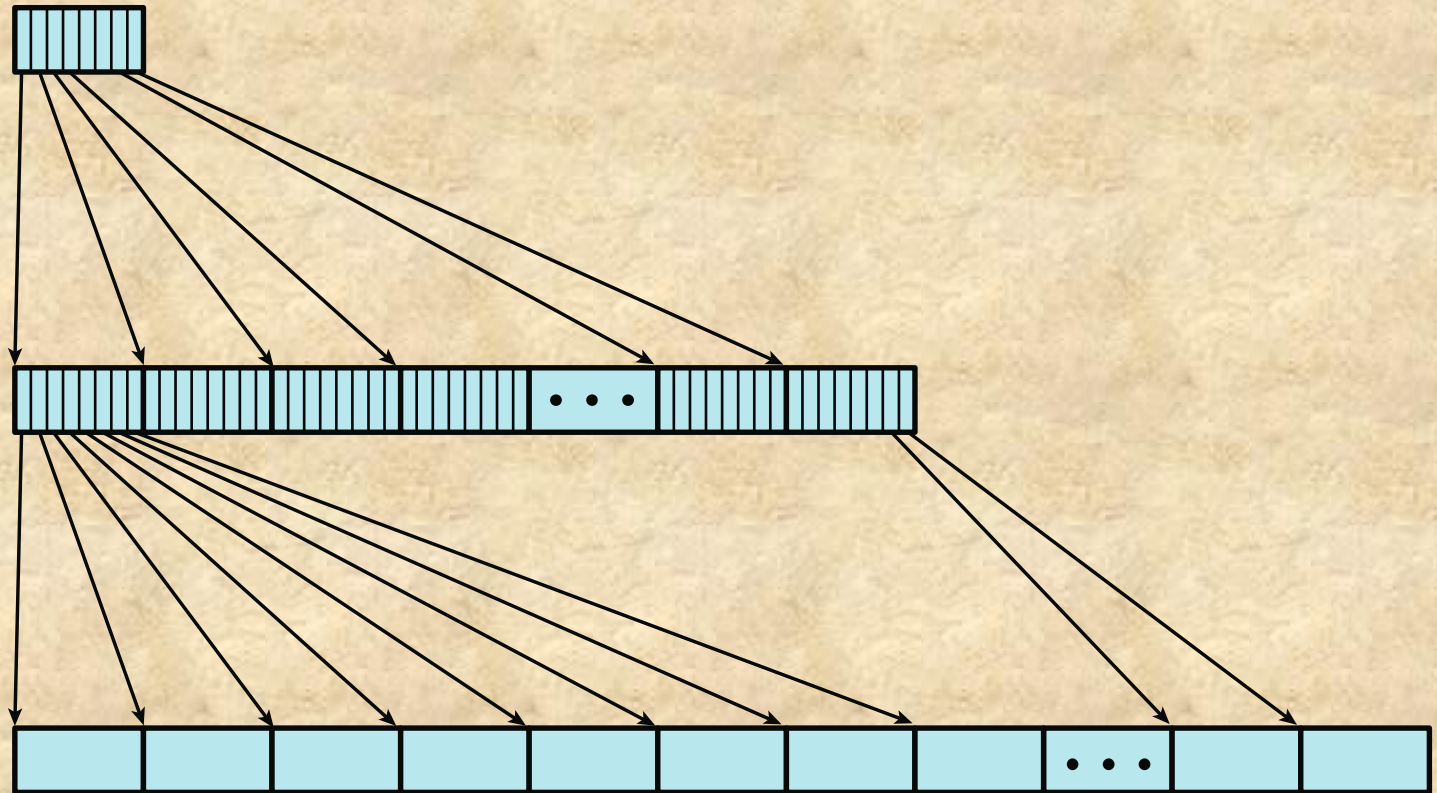


Figure 8.3 A Two-Level Hierarchical Page Table

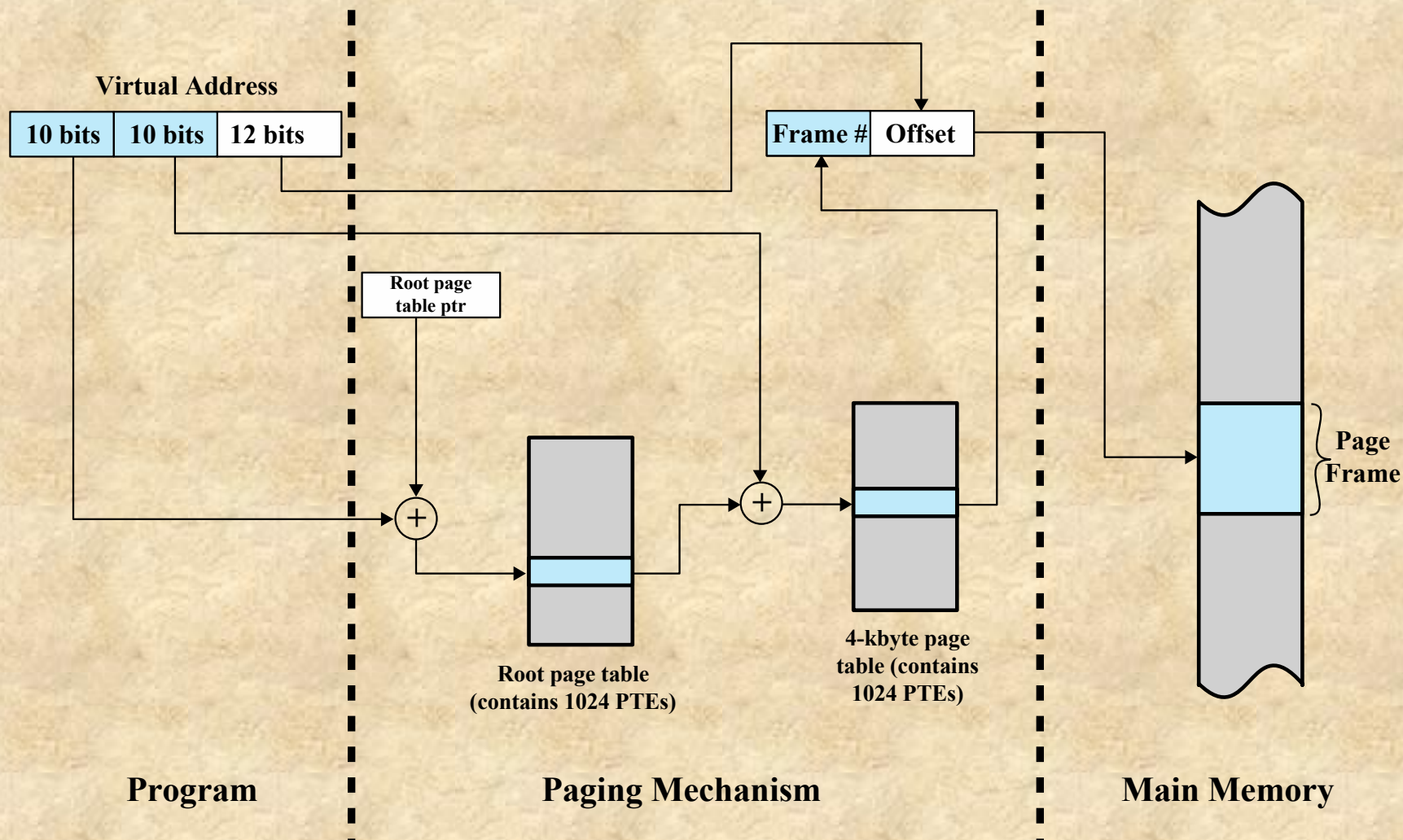


Figure 8.4 Address Translation in a Two-Level Paging System

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses:
 - One to fetch the page table entry
 - One to fetch the data
- To overcome the effect of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache called a *translation lookaside buffer* (TLB)
 - This cache functions in the same way as a memory cache and contains those page table entities that have been most recently used

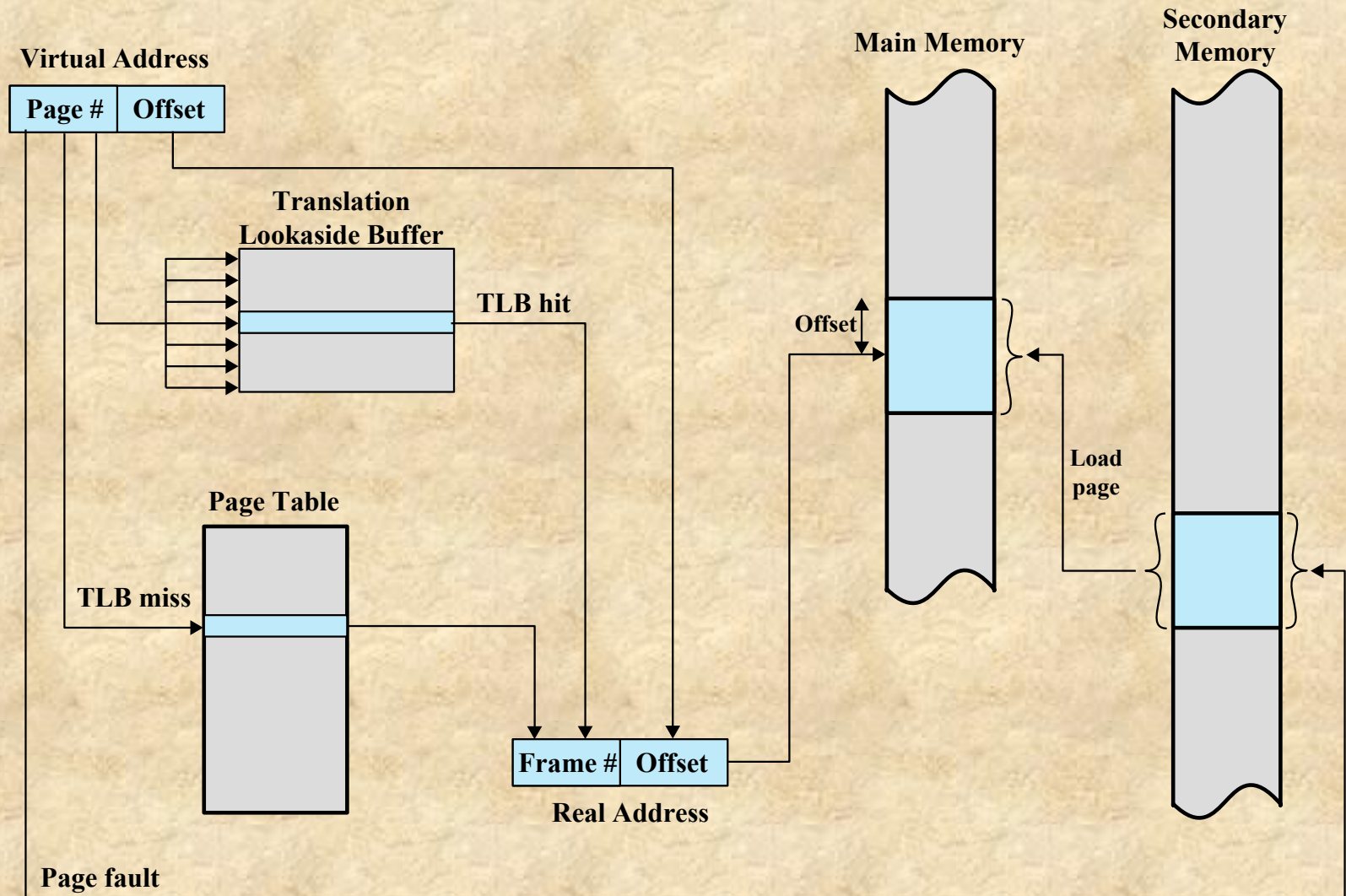


Figure 8.6 Use of a Translation Lookaside Buffer

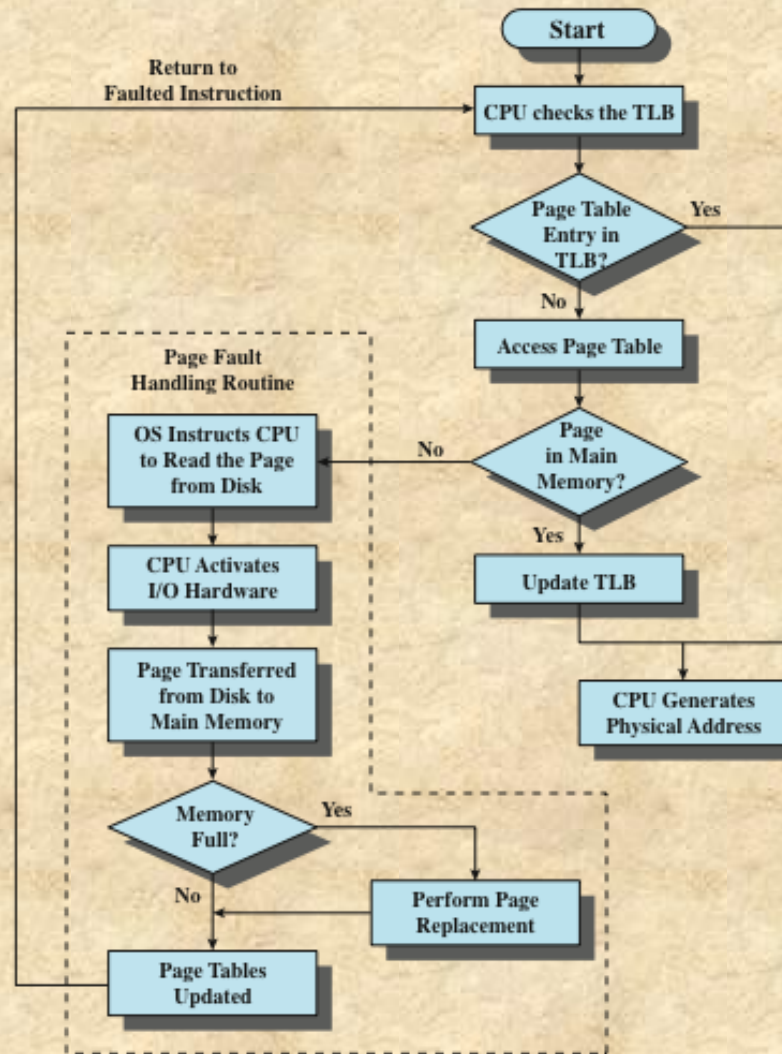


Figure 8.7 Operation of Paging and Translation Lookaside Buffer (TLB)

Associative Mapping

- The TLB only contains some of the page table entries so we cannot simply index into the TLB based on page number
 - Each TLB entry must include the page number as well as the complete page table entry
- The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number

TLB Operation

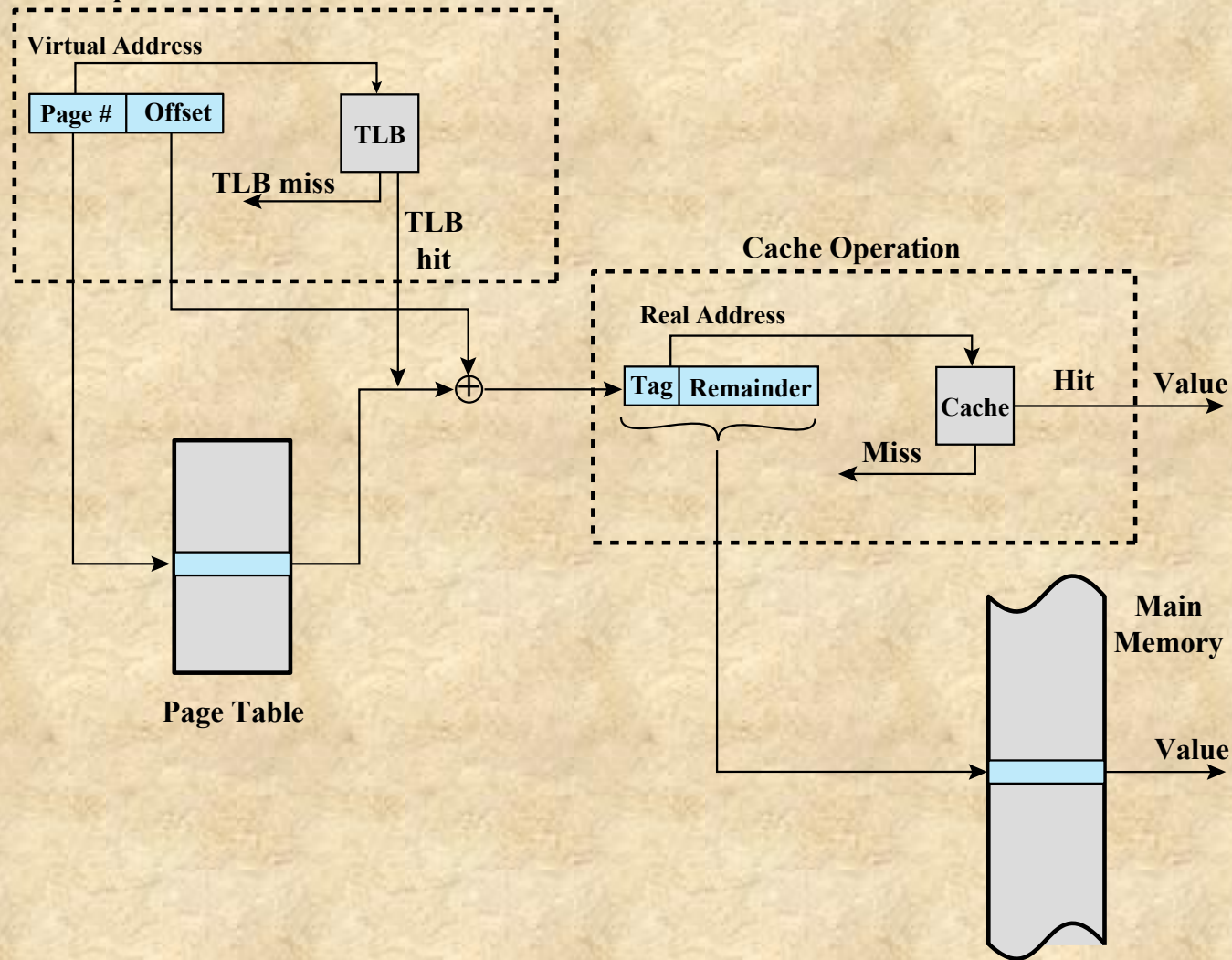


Figure 8.9 Translation Lookaside Buffer and Cache Operation

Operating System Software

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

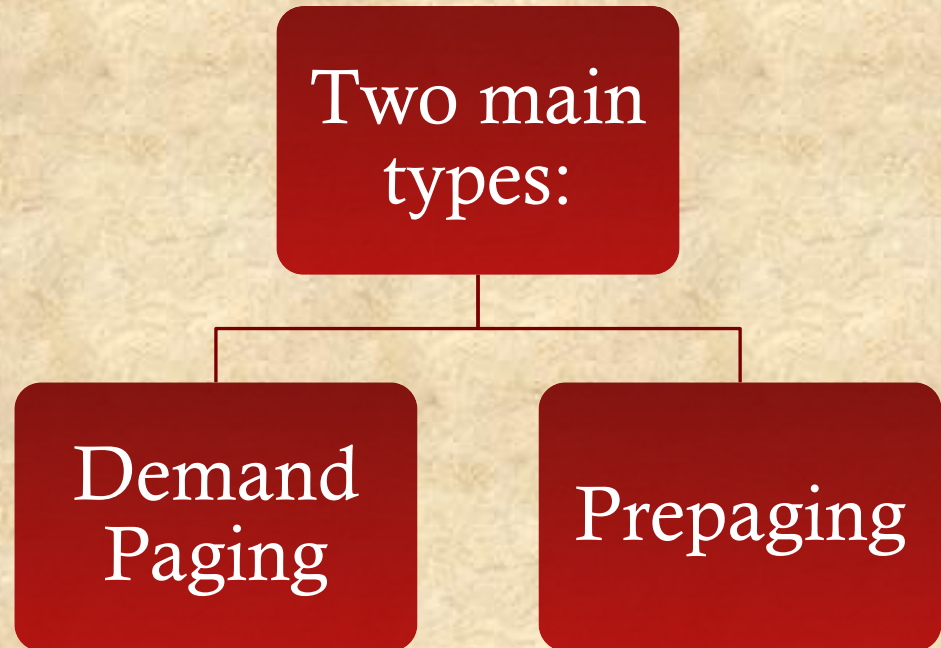
- Whether or not to use virtual memory techniques
- The use of paging or segmentation or both
- The algorithms employed for various aspects of memory management

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



Demand Paging

■ 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

■ 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
- Should not be confused with “swapping”

Placement Policy

- Determines where in real memory a process piece is to reside
- Important design issue in a segmentation system
- Paging or combined paging with segmentation placing is irrelevant because hardware performs functions with equal efficiency
- For NUMA systems an automatic placement strategy is desirable

Replacement Policy

- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
 - Objective 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

Frame Locking

- When a frame is locked the page currently stored in that frame may not be replaced
 - Kernel of the OS as well as key control structures are held in locked frames
 - I/O buffers and time-critical areas may be locked into main memory frames
 - Locking is achieved by associating a lock bit with each frame

Basic Algorithms



Algorithms used for the selection of a page to replace:

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

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		

LRU

2	2	2	2	2	2	2	2	3	3	3	3
	3	3	3	5	5	5	5	5	5	5	5
			1	1	1	4	4	4	2	2	2
				F		F		F	F		

FIFO

2	2	2	2	5	5	5	5	3	3	3	3
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2
				F	F	F		F		F	F

CLOCK

2*	2*	2*	2*	5*	5*	5*	5*	3*	3*	3*	3*
	3*	3*	3*	3	2*	2*	2*	2	2*	2	2*
			1*	1	1	4*	4*	4	4	5*	5*
				F	F	F		F		F	

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

Figure 8.14 Behavior of Four Page-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
- 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