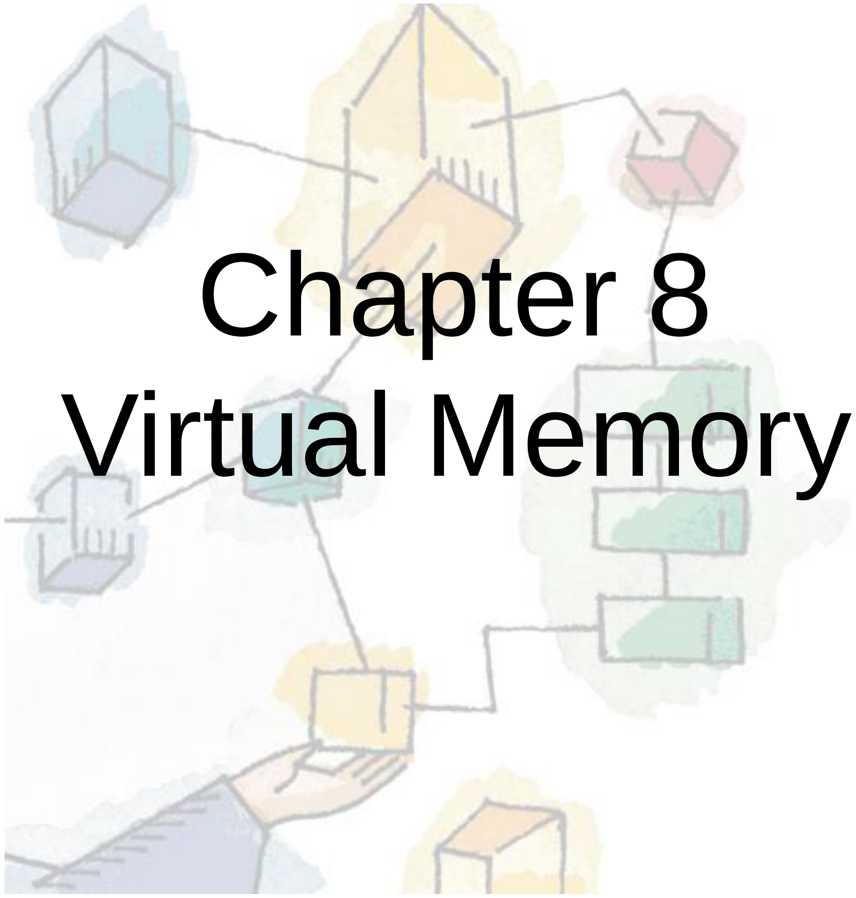
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



Chapter 8
Virtual Memory

Dave Bremer
Otago Polytechnic, N.Z.

Roadmap

→ Hardware and Control Structures

- Operating System Software
- UNIX and Solaris Memory Management
- Linux Memory Management
- Windows Memory Management

Terminology

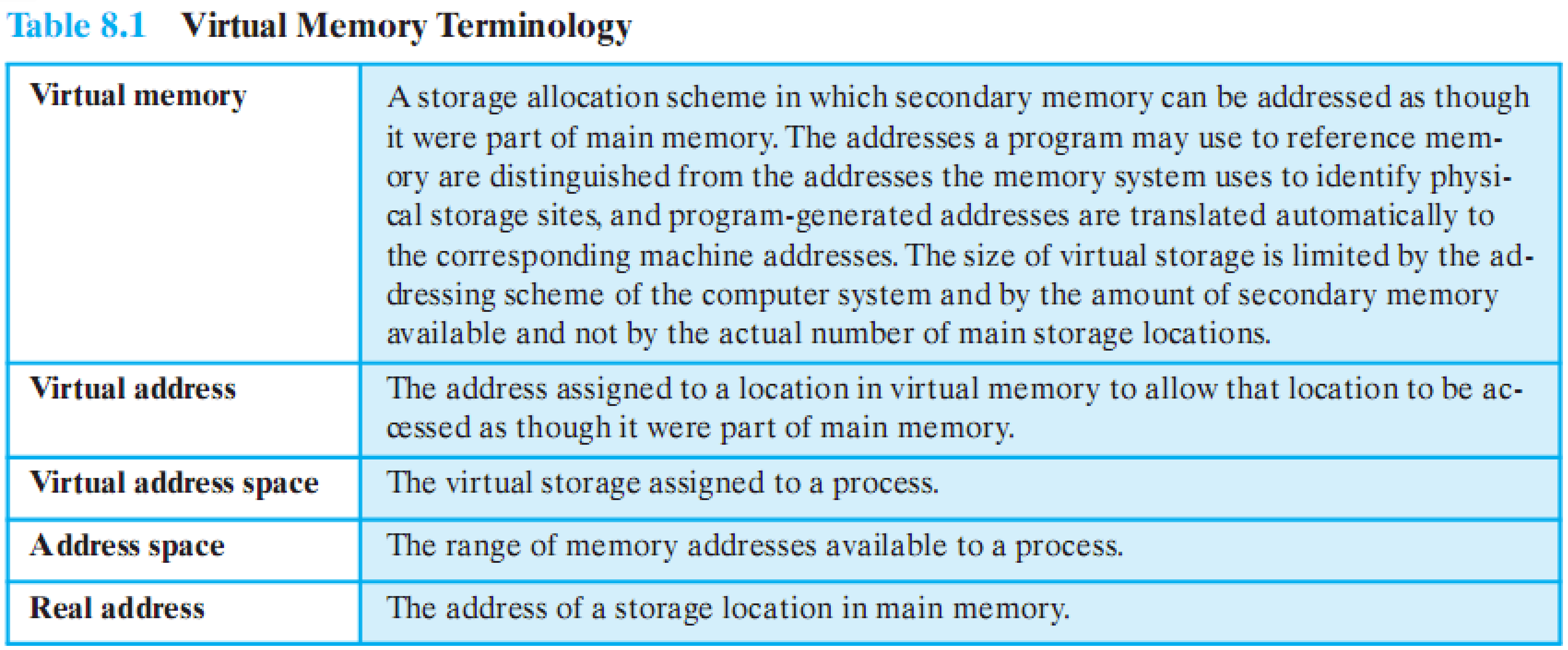
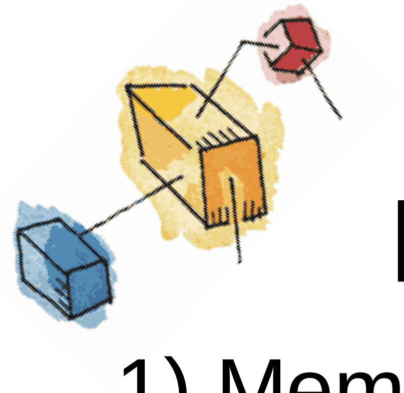


Table 8.1 Virtual Memory Terminology

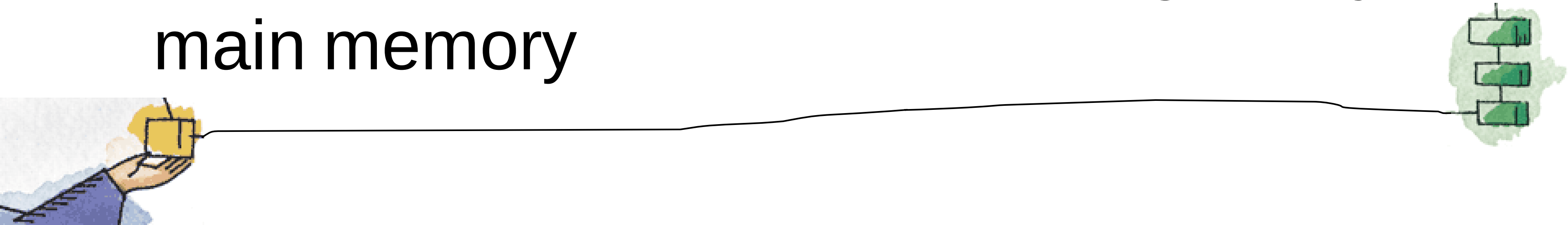
Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses 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.



Key points in Memory Management

1) Memory references are logical addresses dynamically translated into physical addresses at run time
   - A process may be swapped in and out of main memory occupying different regions at different times during execution

2) A process may be broken up into pieces that do not need to located contiguously in main memory



Breakthrough in Memory Management

- **If both** of those two characteristics are present,
  - then it is not necessary that all of the pages or all of the segments of a process be in main memory during execution.
- If the next instruction, and the next data location are in memory then execution can proceed
  - at least for a time

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 complete which causes the operating system to place the affected process in the Ready state

Implications of this new strategy

- 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

Real and Virtual Memory

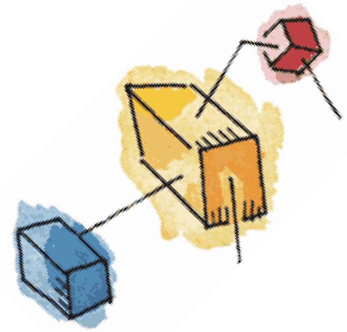
- Real memory
  - Main memory, the actual RAM
- Virtual memory
  - Memory on disk
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory

Thrashing

- A state in which the system spends most of its time swapping pieces rather than executing instructions.
- To avoid this, the operating system tries to guess which pieces are least likely to be used in the near future.
    - The guess is based on recent history

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
- This suggests that virtual memory may work efficiently



Support Needed for Virtual Memory

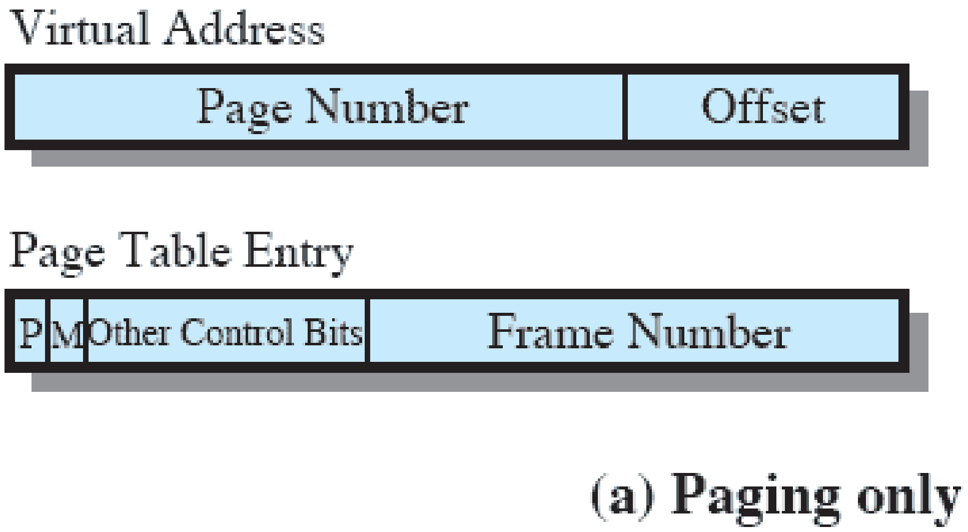
- Hardware must support paging and segmentation
- Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

Paging

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- Two extra bits are needed to indicate:
  - whether the page is in main memory or not
  - Whether the contents of the page has been altered since it was last loaded

(see next slide)

Paging Table



(a) Paging only

Address Translation

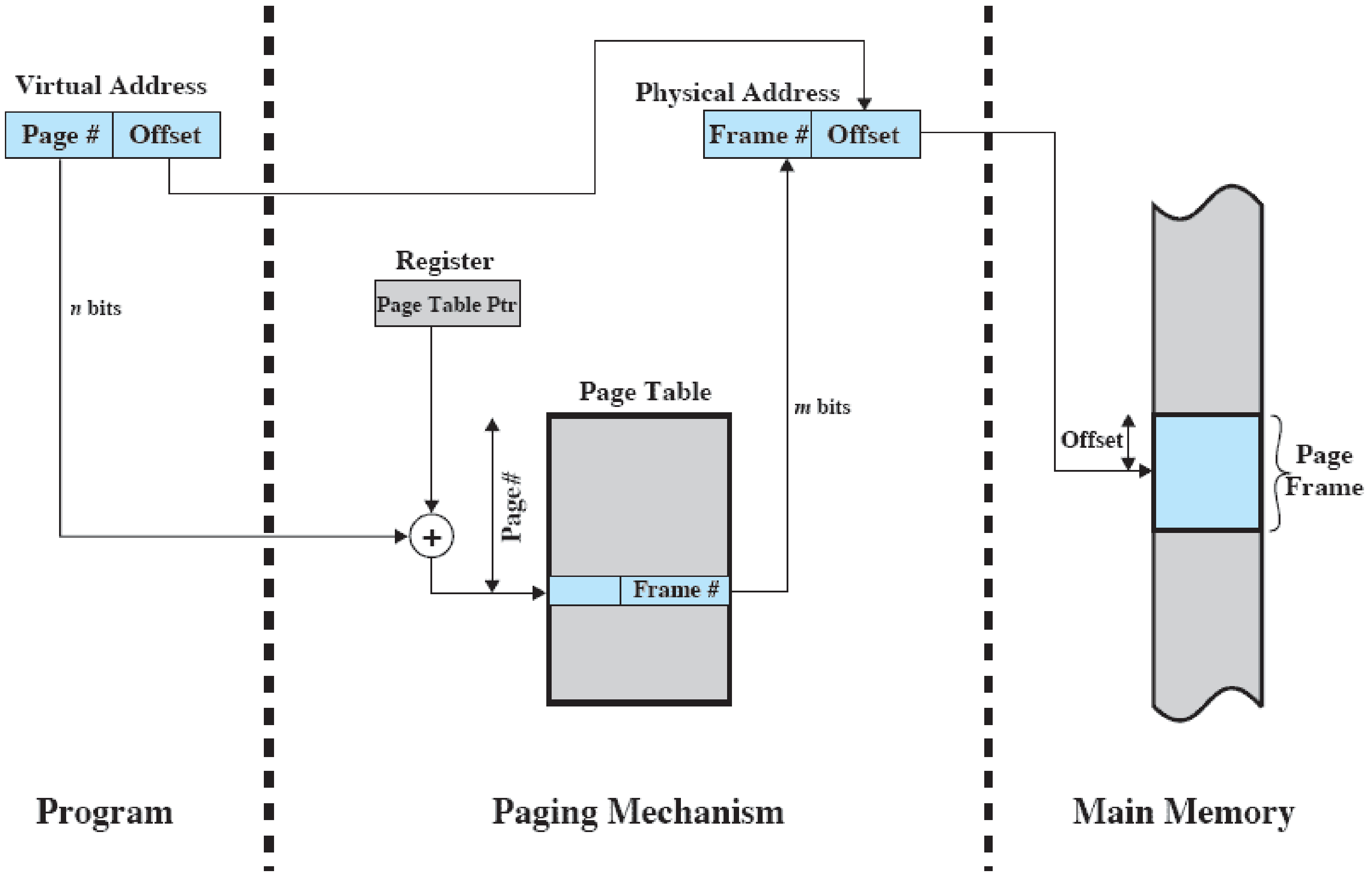


Figure 8.3 Address Translation in a Paging System

Page Tables

- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

Two-Level Hierarchical Page Table

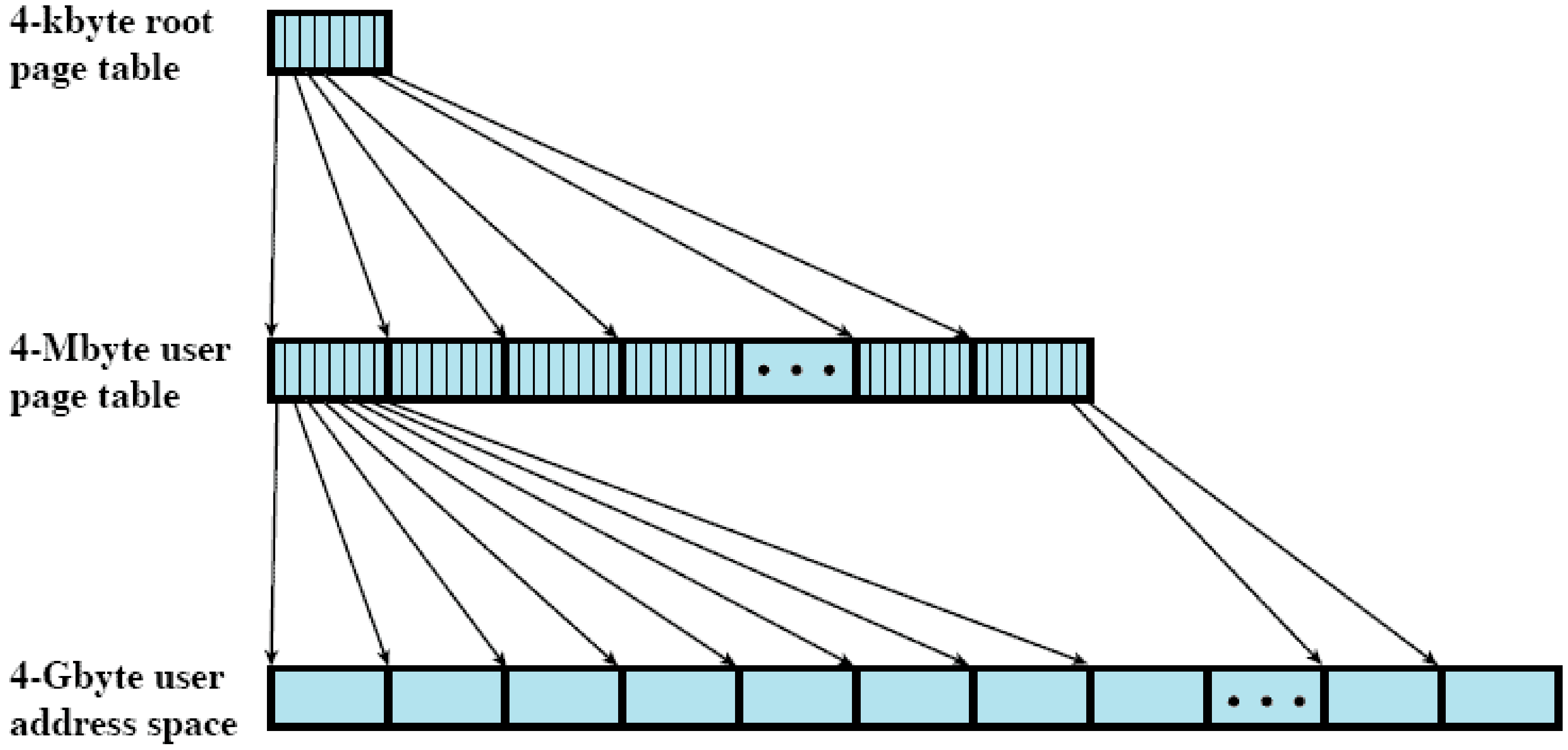


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation for Hierarchical page table

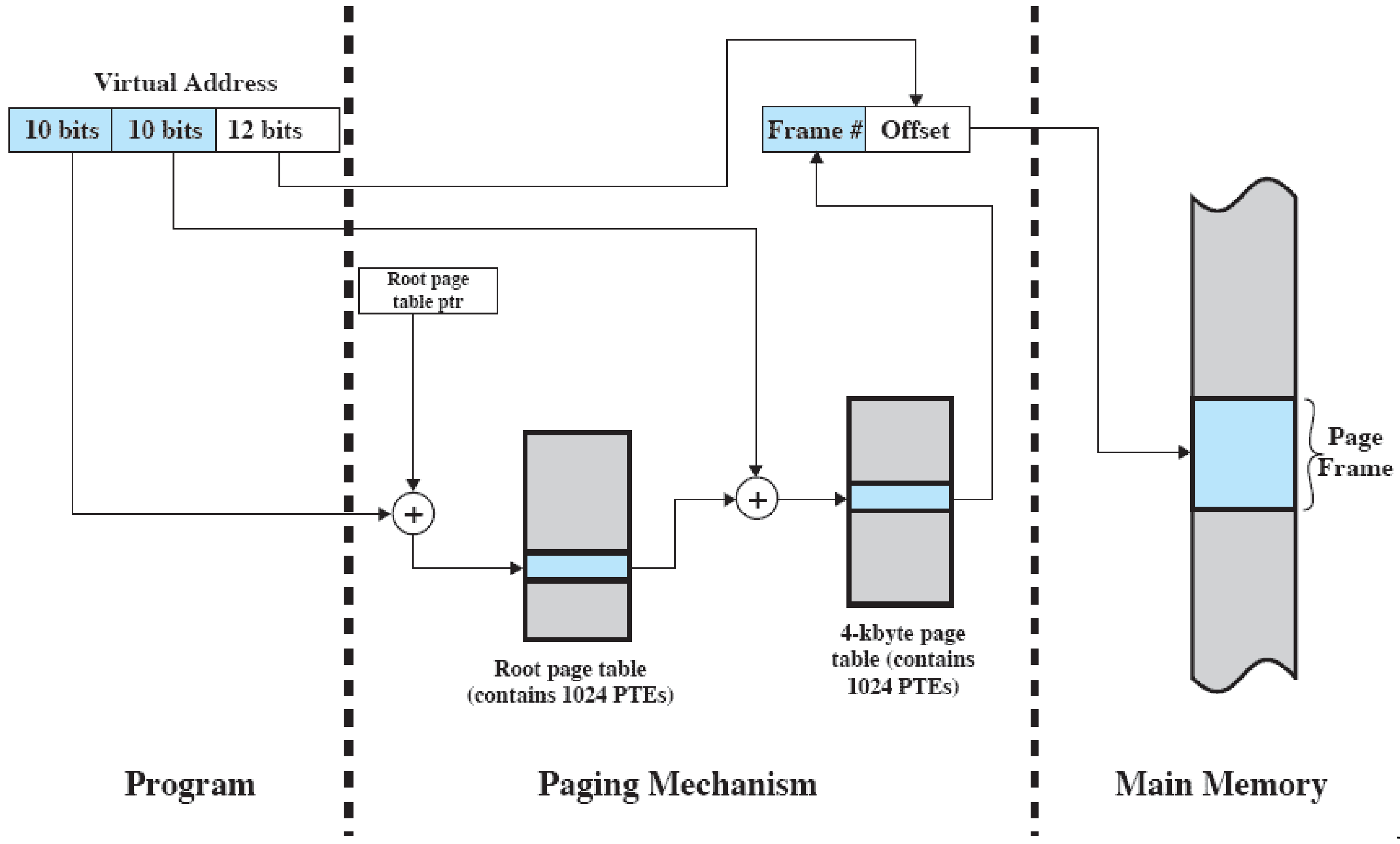


Figure 8.5 Address Translation in a Two-Level Paging System

Page tables grow proportionally

- A drawback of the type of page tables just discussed is that their size is proportional to that of the virtual address space.
- An alternative is Inverted Page Tables

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
- Fixed proportion of real memory is required for the tables regardless of the number of processes

Inverted Page Table

Each entry in the page table includes:

- Page number
- Process identifier
  - The process that owns this page.
- Control bits
  - includes flags, such as valid, referenced, etc
- Chain pointer
  - the index value of the next entry in the chain.

Inverted Page Table

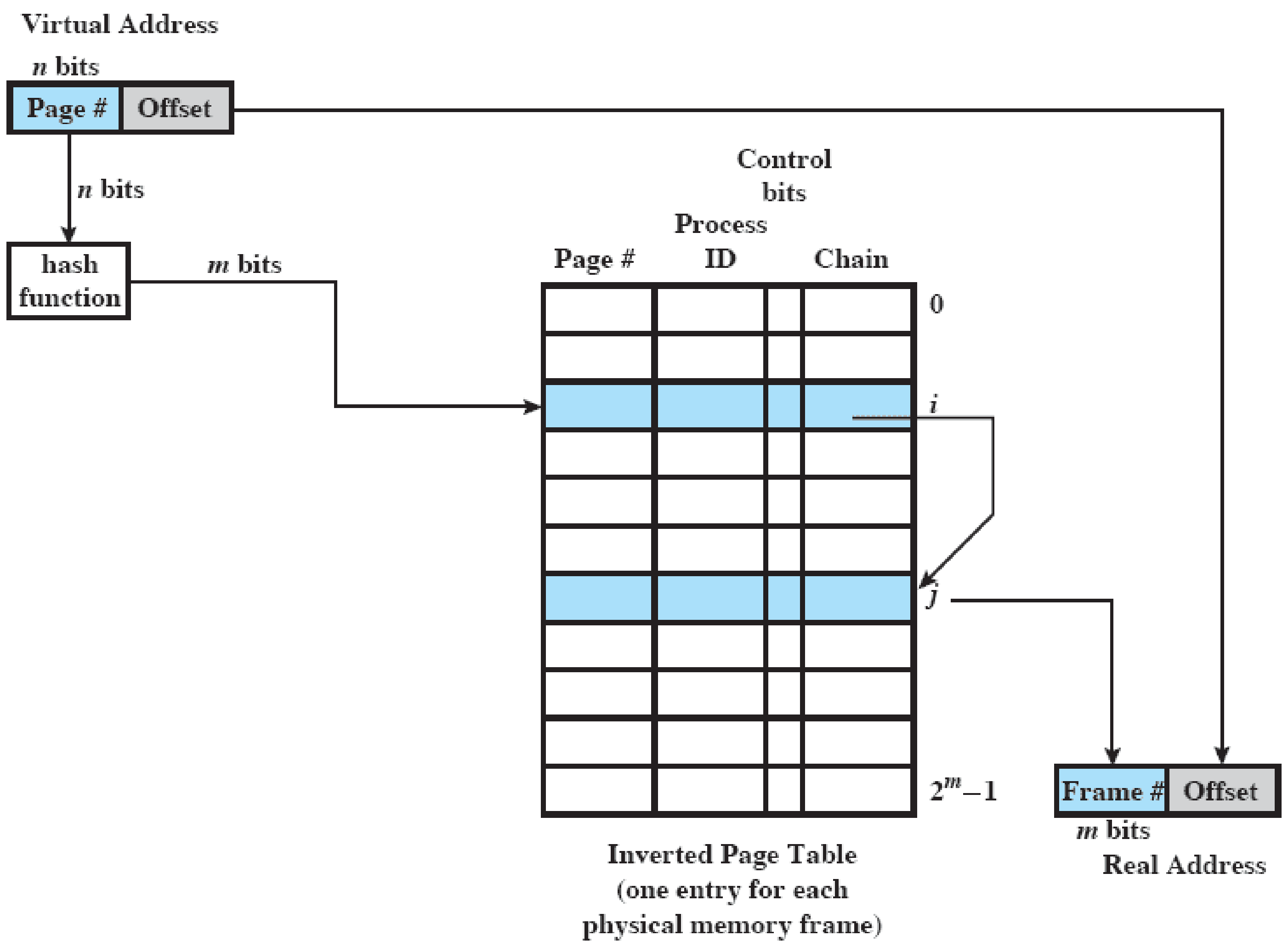
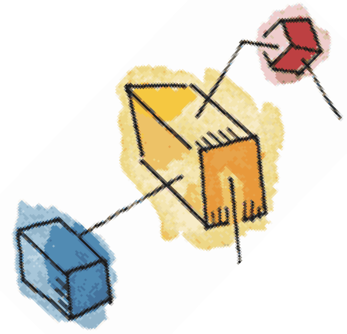


Figure 8.6 Inverted Page Table Structure

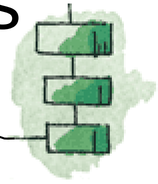
Translation Lookaside Buffer

- 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



TLB Operation

- 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



Looking into the Process Page Table

- 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

Translation Lookaside Buffer

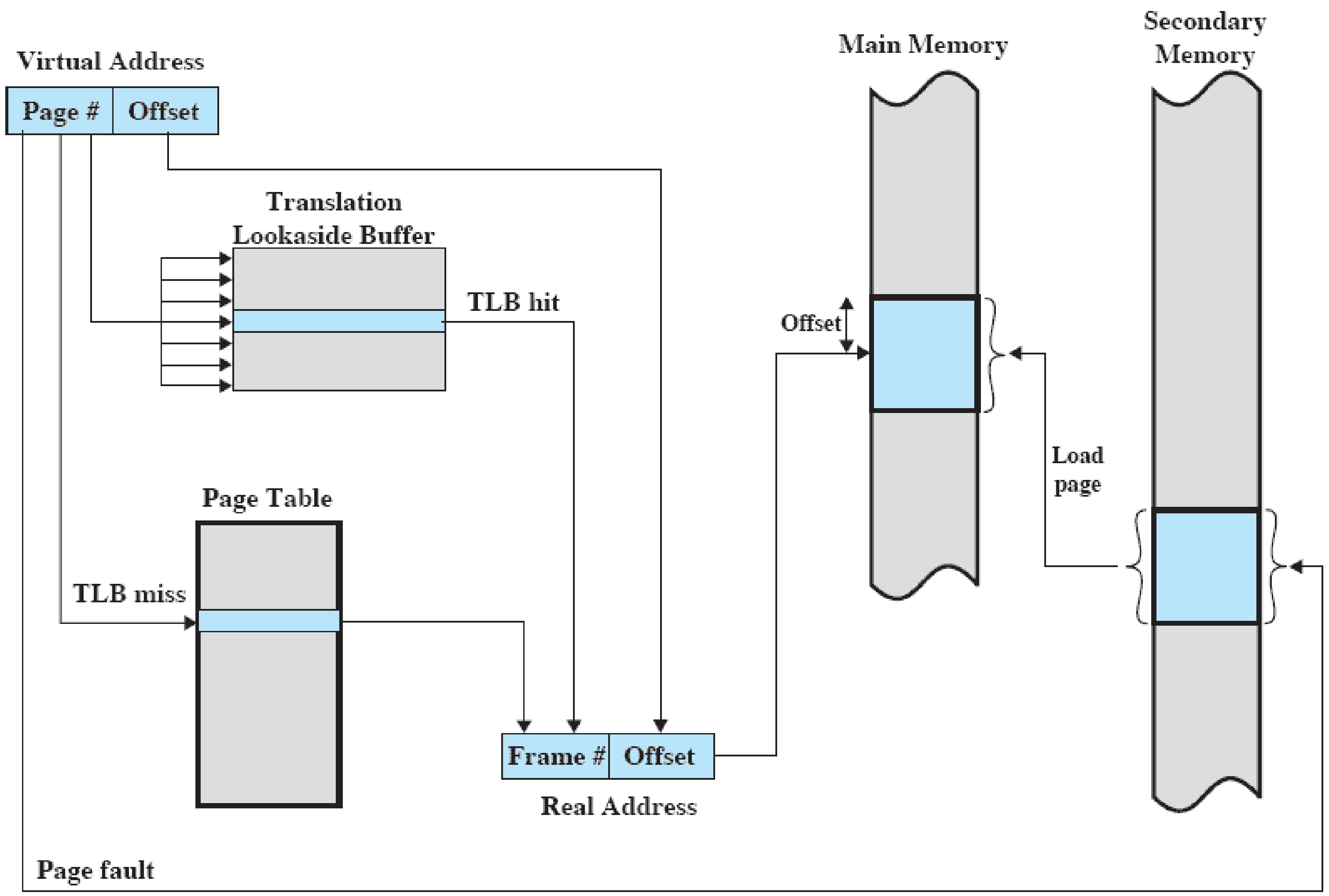
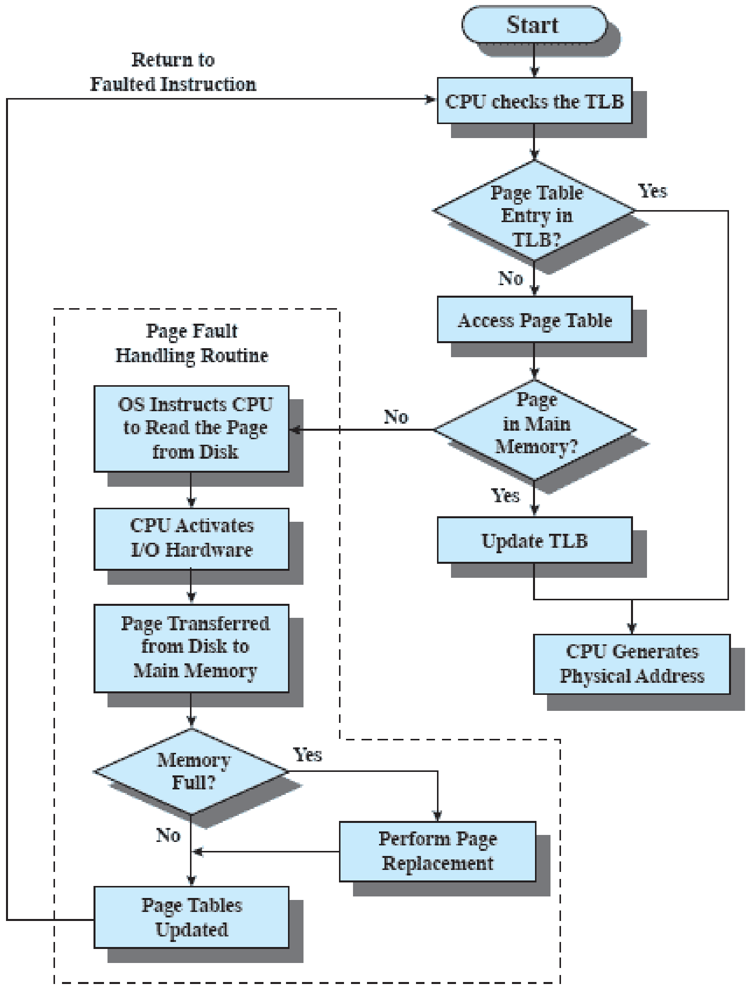


Figure 8.7 Use of a Translation Lookaside Buffer

TLB operation

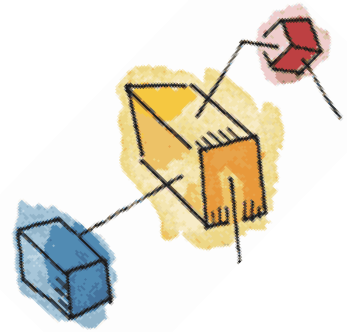


Start
Return to Faulted Instruction
CPU checks the TLB
Page Table Entry in TLB?
Yes
No
Access Page Table
Page Fault Handling Routine
OS Instructs CPU to Read the Page from Disk
No
Page in Main Memory?
Yes
CPU Activates I/O Hardware
Update TLB
Page Transferred from Disk to Main Memory
CPU Generates Physical Address
Memory Full?
Yes
No
Perform Page Replacement
Page Tables Updated

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

Page Size

- Smaller page size, less amount of internal fragmentation
- But 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



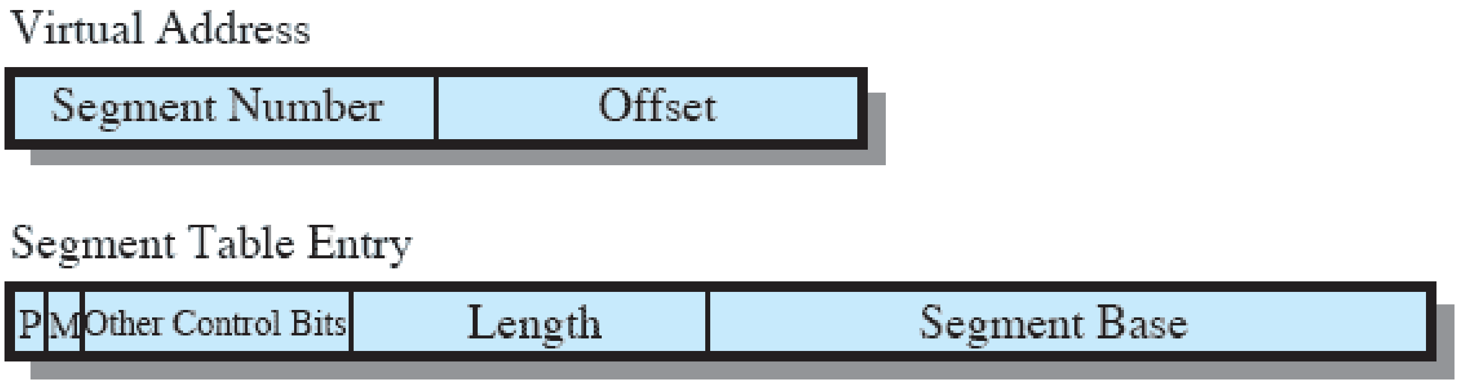
Segmentation

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

- 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

Address Translation in Segmentation

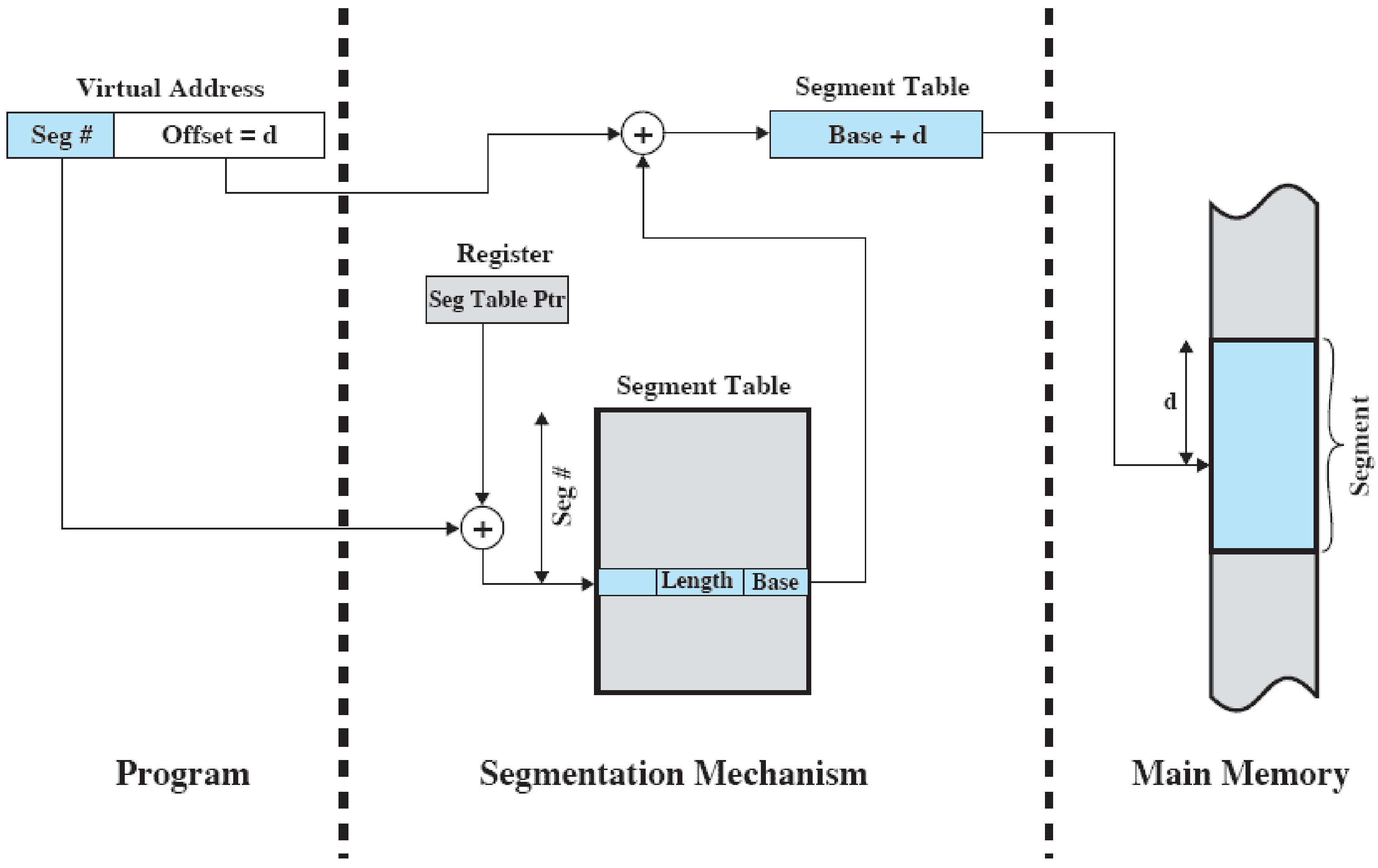


Figure 8.12 Address Translation in a Segmentation System

Protection and sharing

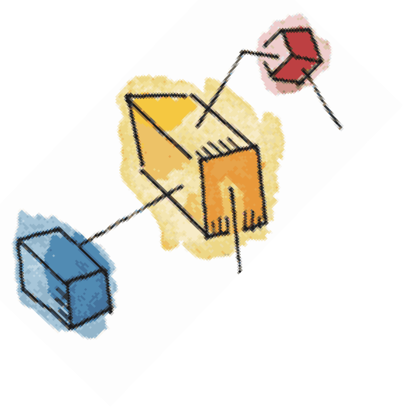
- Segmentation lends itself to the implementation of protection and sharing policies.
- As each entry has a base address and length, inadvertent memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes

Roadmap

- Hardware and Control Structures
- Operating System Software
- UNIX and Solaris Memory Management
- Linux Memory Management
- Windows Memory Management

Memory Management Decisions

- 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



Key Design Elements

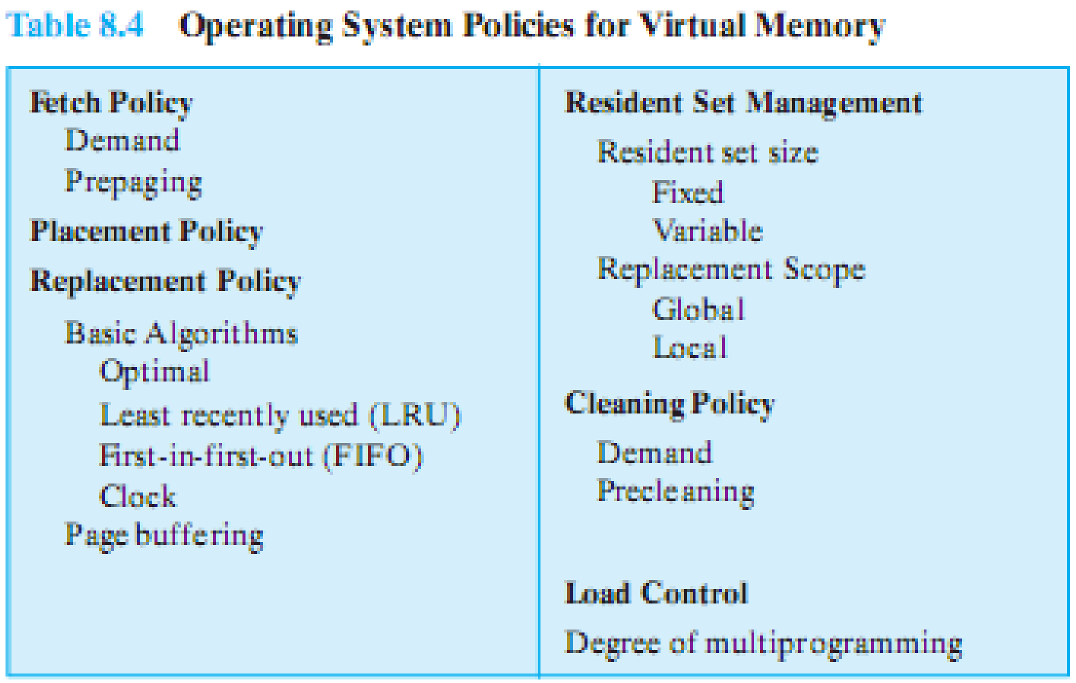
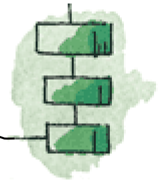


Table 8.4 Operating System Policies for Virtual Memory

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

- Key aim: Minimise page faults
  - No definitive best policy



Fetch Policy

- Determines when a page should be brought into memory
- Two main types:
  - Demand Paging
  - Prepaging

Demand Paging and 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 first started
- **Prepaging**
  - brings in more pages than needed
  - More efficient to bring in pages that reside contiguously on the disk
  - Don't confuse with "swapping"

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

- When all of the frames in main memory are occupied and it is necessary to bring in a new page, the replacement policy determines which page currently in memory is to be replaced.

But…

- Which page is replaced?
- Page removed should be the page least likely to be referenced in the near future
  - How is that determined?
  - Principal of locality again
- Most policies predict the future behavior on the basis of past behavior

Replacement Policy: Frame Locking

- Frame Locking
  - If frame is locked, it may not be replaced
  - Kernel of the operating system
  - Key control structures
  - I/O buffers
  - Associate a lock bit with each frame

Basic Replacement Algorithms

- There are certain basic algorithms that are used for the selection of a page to replace, they include
  - Optimal
  - Least recently used (LRU)
  - First-in-first-out (FIFO)
  - Clock
- Examples

Examples

- An example of the implementation of these policies will use a page address stream formed by executing the program is
  - 2 3 2 1 5 2 4 5 3 2 5 2
- Which means that the first page referenced is 2,
  - the second page referenced is 3,
  - And so on.

Optimal policy

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

Optimal Policy Example

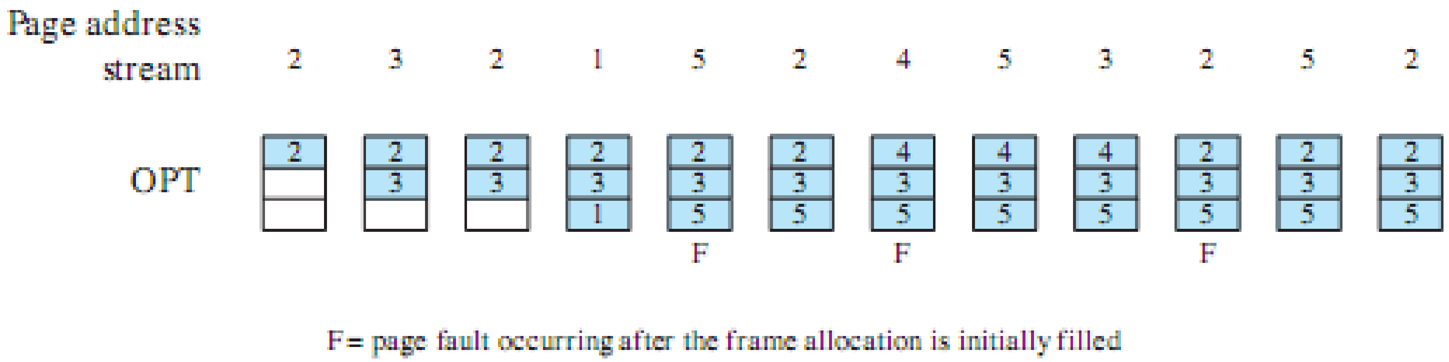
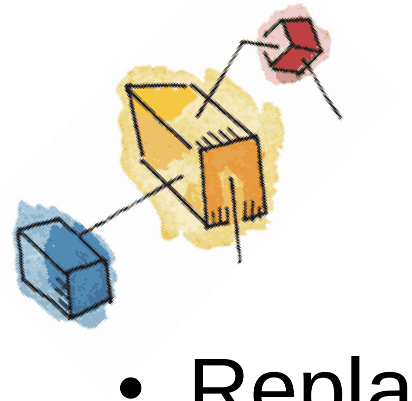

Page address stream
2 3 2 1 5 2 4 5 3 2 5 2
OPT
F
F
F
F= page fault occurring after the frame allocation is initially filled

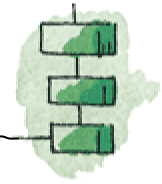
Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- The optimal policy produces three page faults after the frame allocation has been filled.

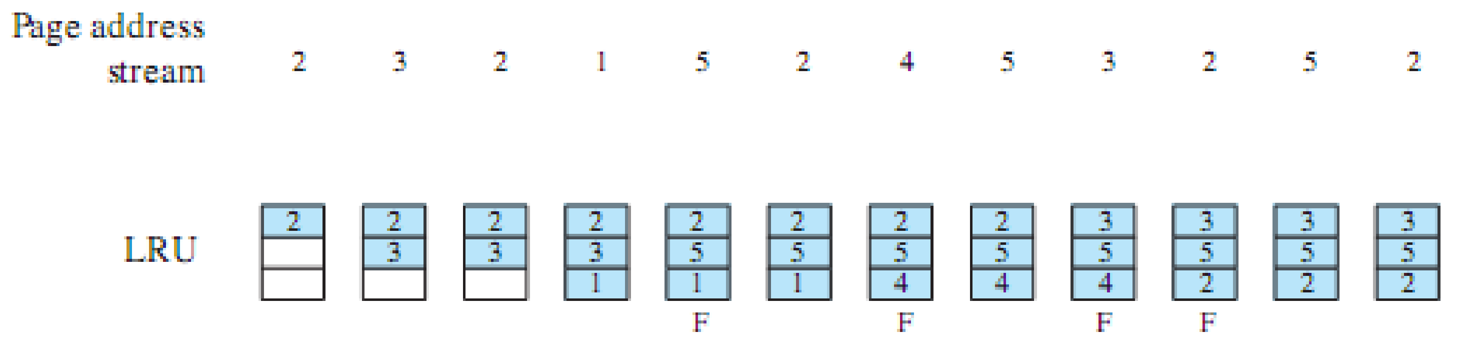


Least Recently Used (LRU)

- 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.



LRU Example



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

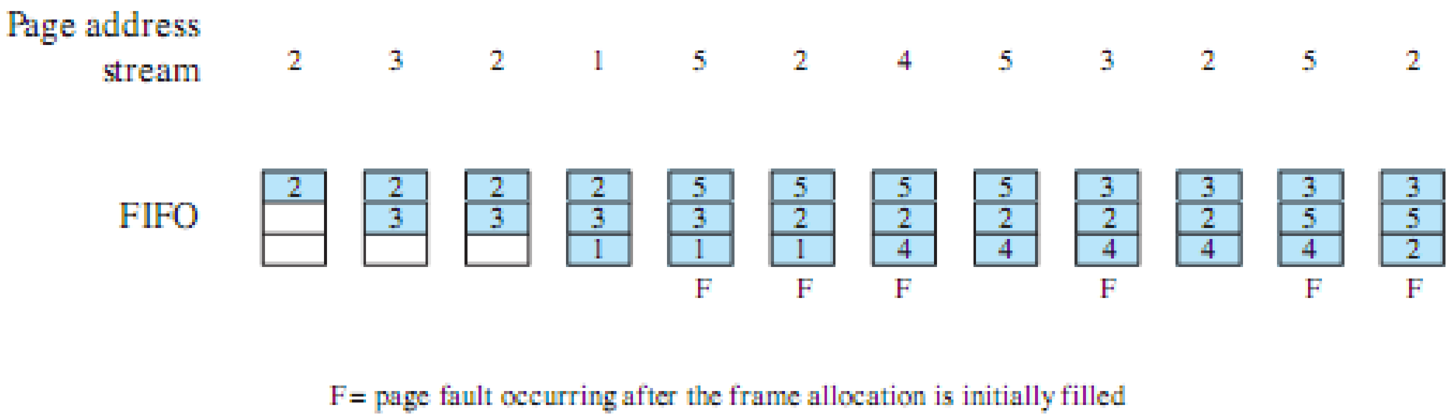
Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- The LRU policy does nearly as well as the optimal policy.
  - In this example, there are four page faults

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
  - But, these pages may be needed again very soon if it hasn't truly fallen out of use

FIFO Example



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

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

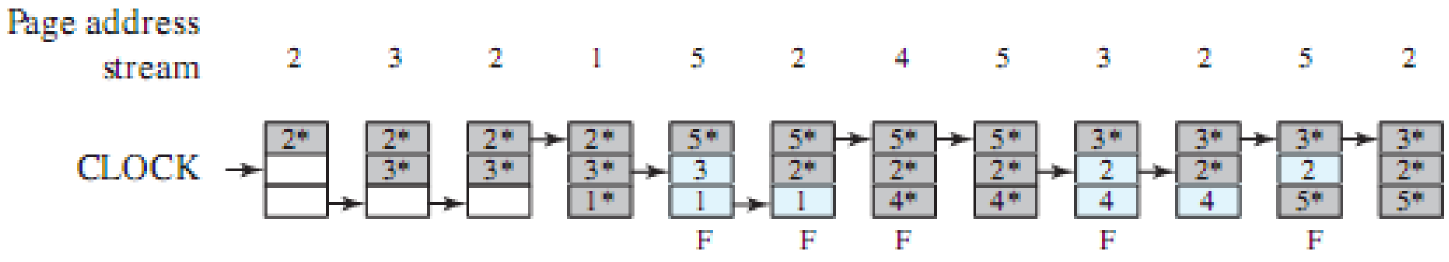
Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- The FIFO policy results in six page faults.
  - Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

Clock Policy

- Uses and additional bit called a “use bit”
- When a page is first loaded in memory or referenced, the use bit is set to 1
- When it is time to replace a page, the OS scans the set flipping all 1’s to 0
- The first frame encountered with the use bit already set to 0 is replaced.

Clock Policy Example

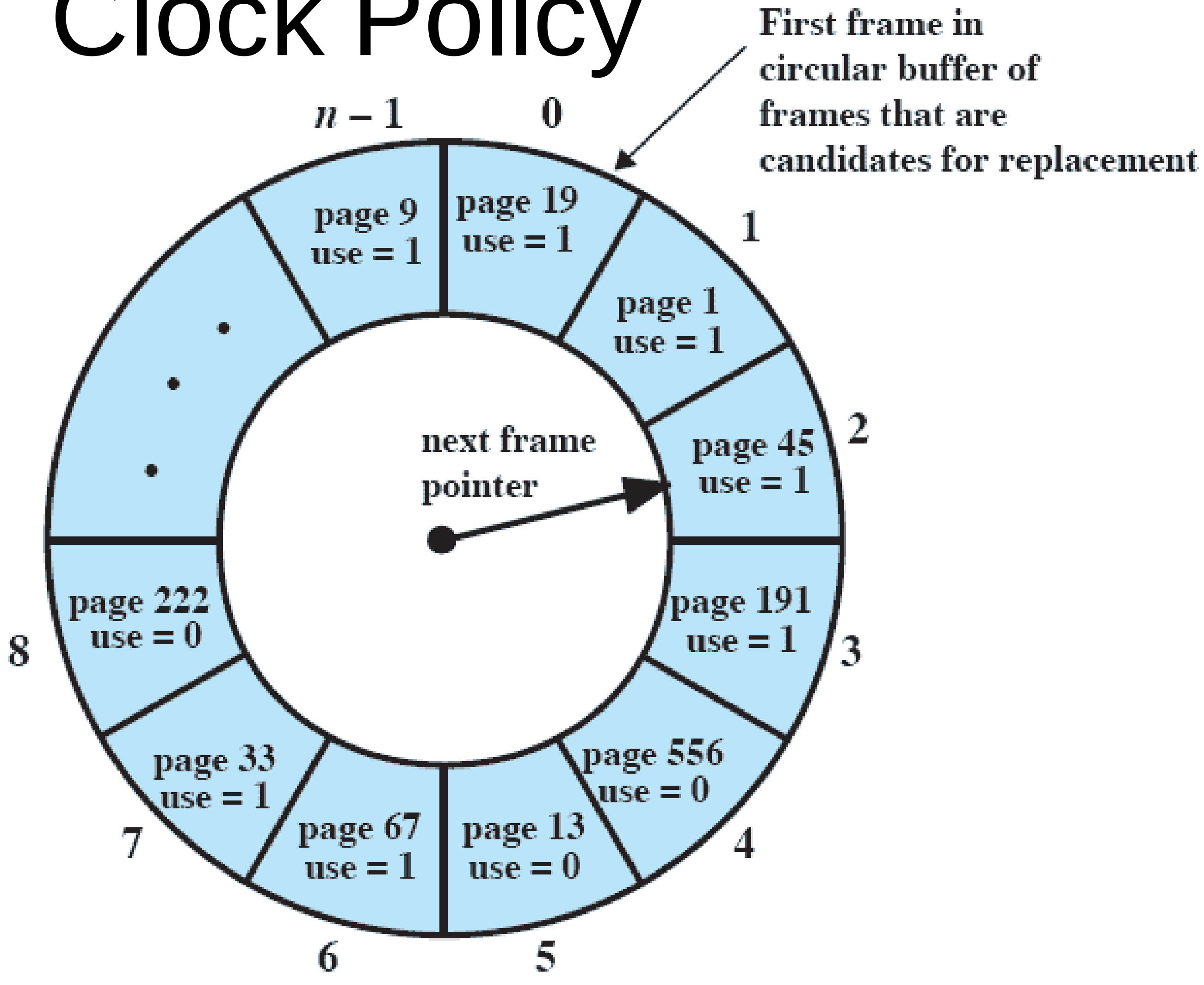


F= page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

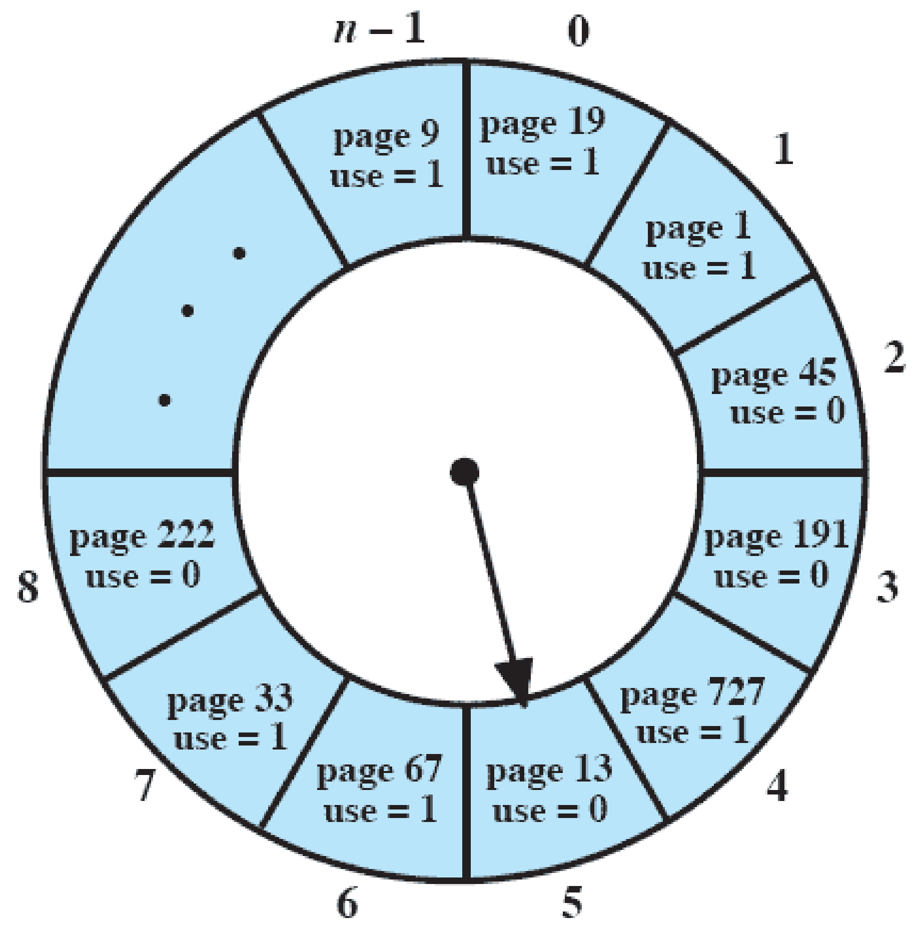
- Note that the clock policy is adept at protecting frames 2 and 5 from replacement.

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 Policy

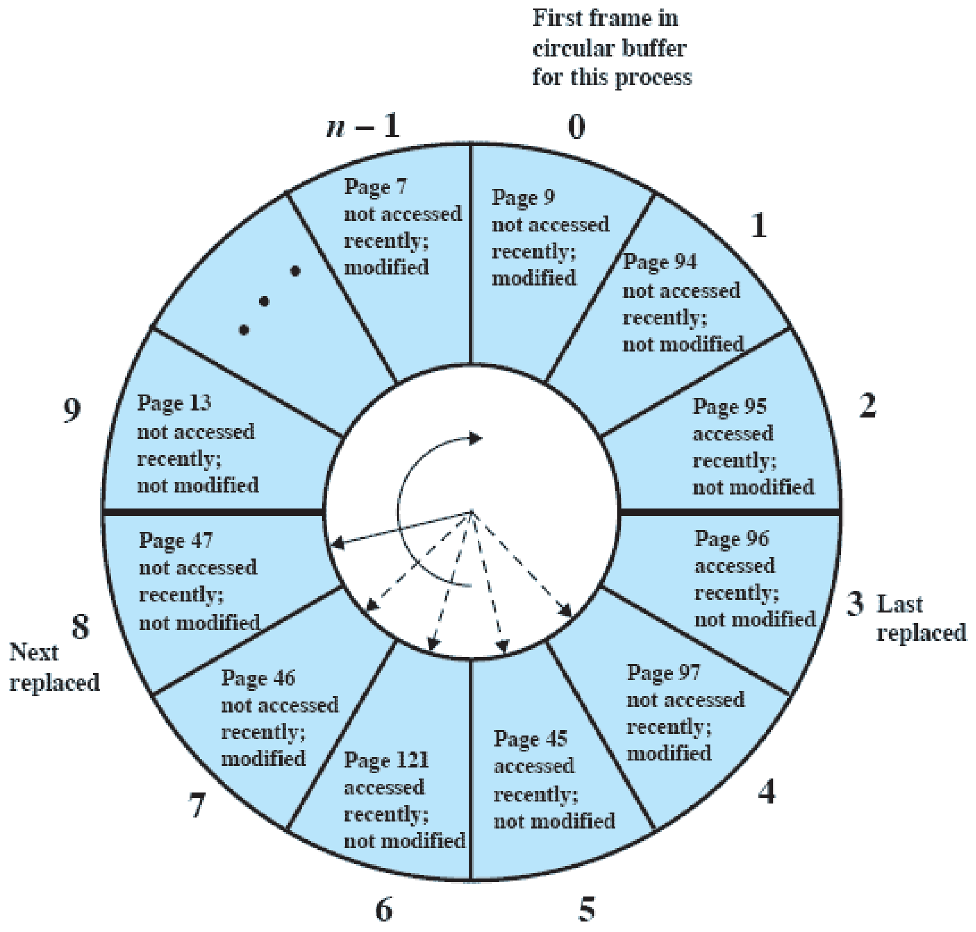
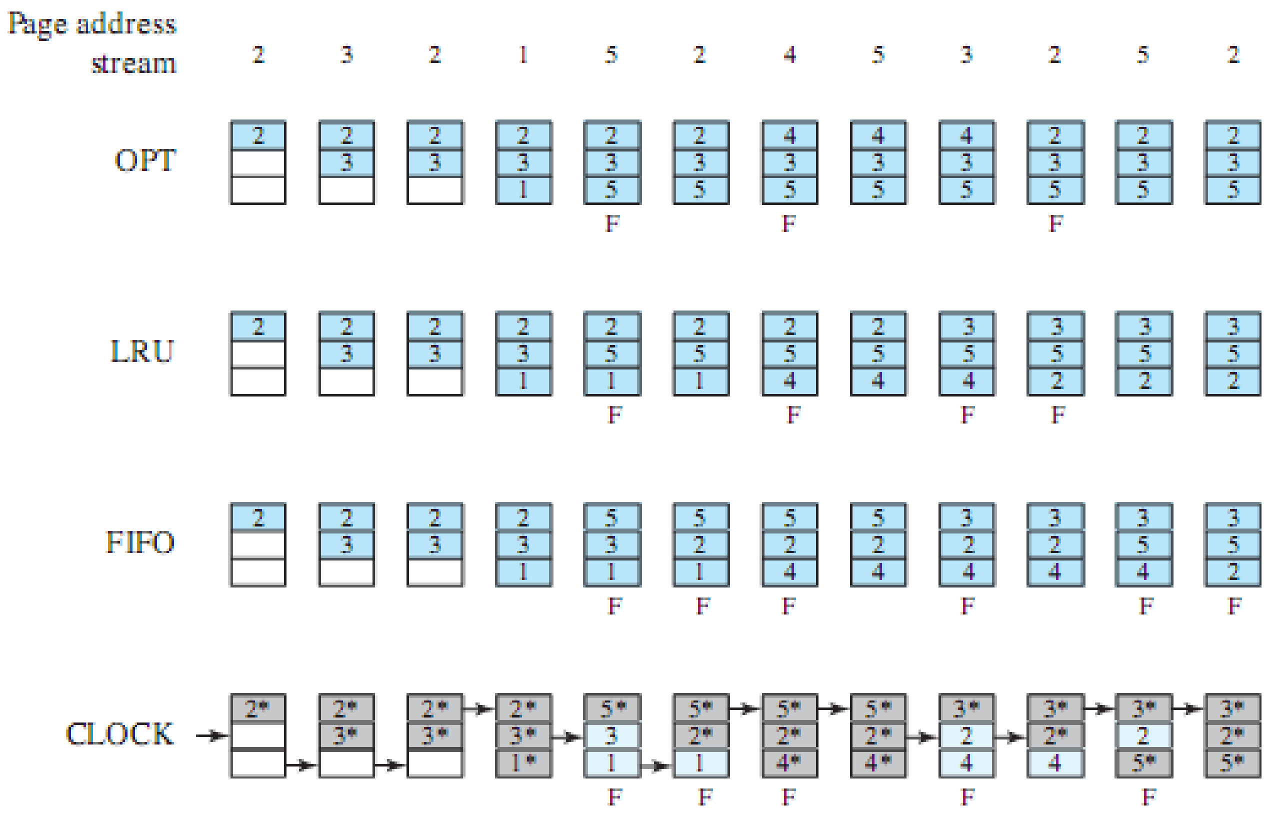

First frame in circular buffer for this process
n – 1
0
1
2
3
4
5
6
7
8
9
Page 7 not accessed recently; modified
Page 9 not accessed recently; modified
Page 94 not accessed recently; not modified
Page 95 accessed recently; not modified
Page 96 accessed recently; not modified
Last replaced
Page 97 not accessed recently; modified
Page 45 accessed recently; not modified
Page 121 accessed recently; not modified
Page 46 not accessed recently; modified
Page 47 not accessed recently; not modified
Next replaced
Page 13 not accessed recently; not modified

Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Combined Examples



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
CLOCK	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* 2 5* F	3* 2* 5*

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

Figure 8.15 Behavior of Four Page Replacement Algorithms

Comparison

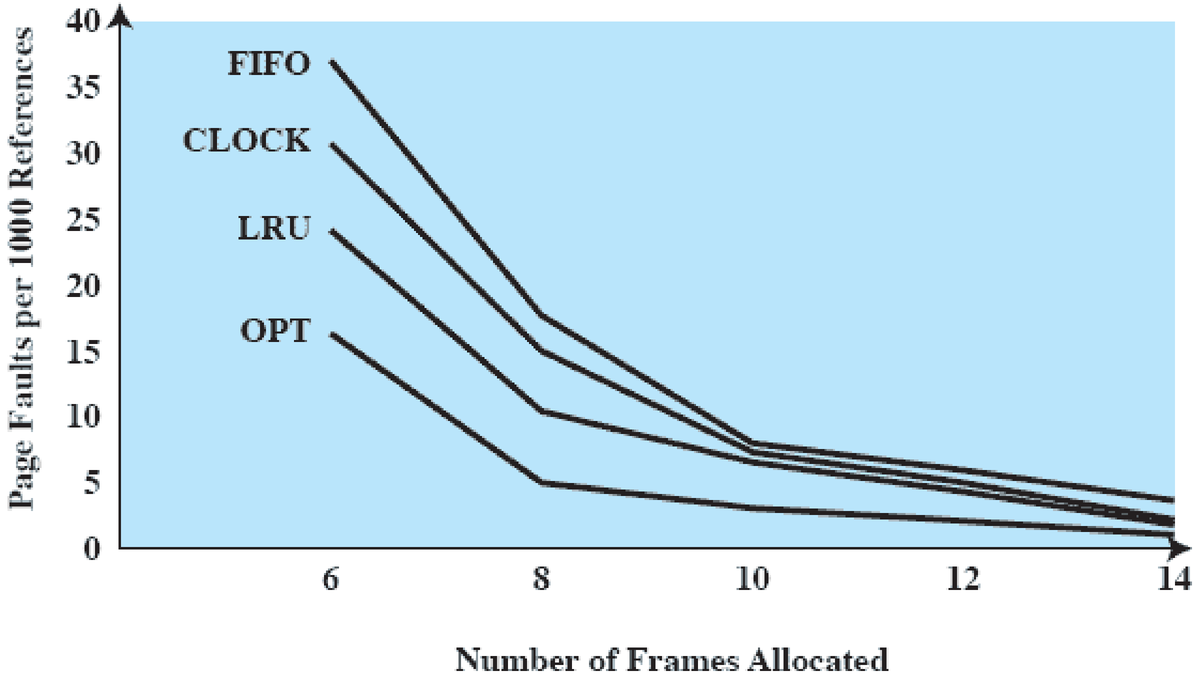


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms