Operating Systems: Internals and Design Principles

Chapter 8
Virtual Memory

Eighth 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
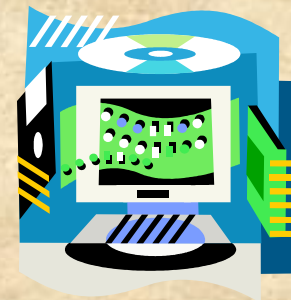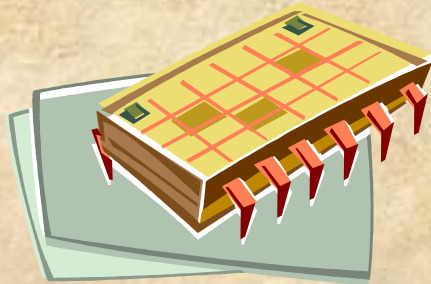
Implications

- More processes may be maintained in main memory
    - 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

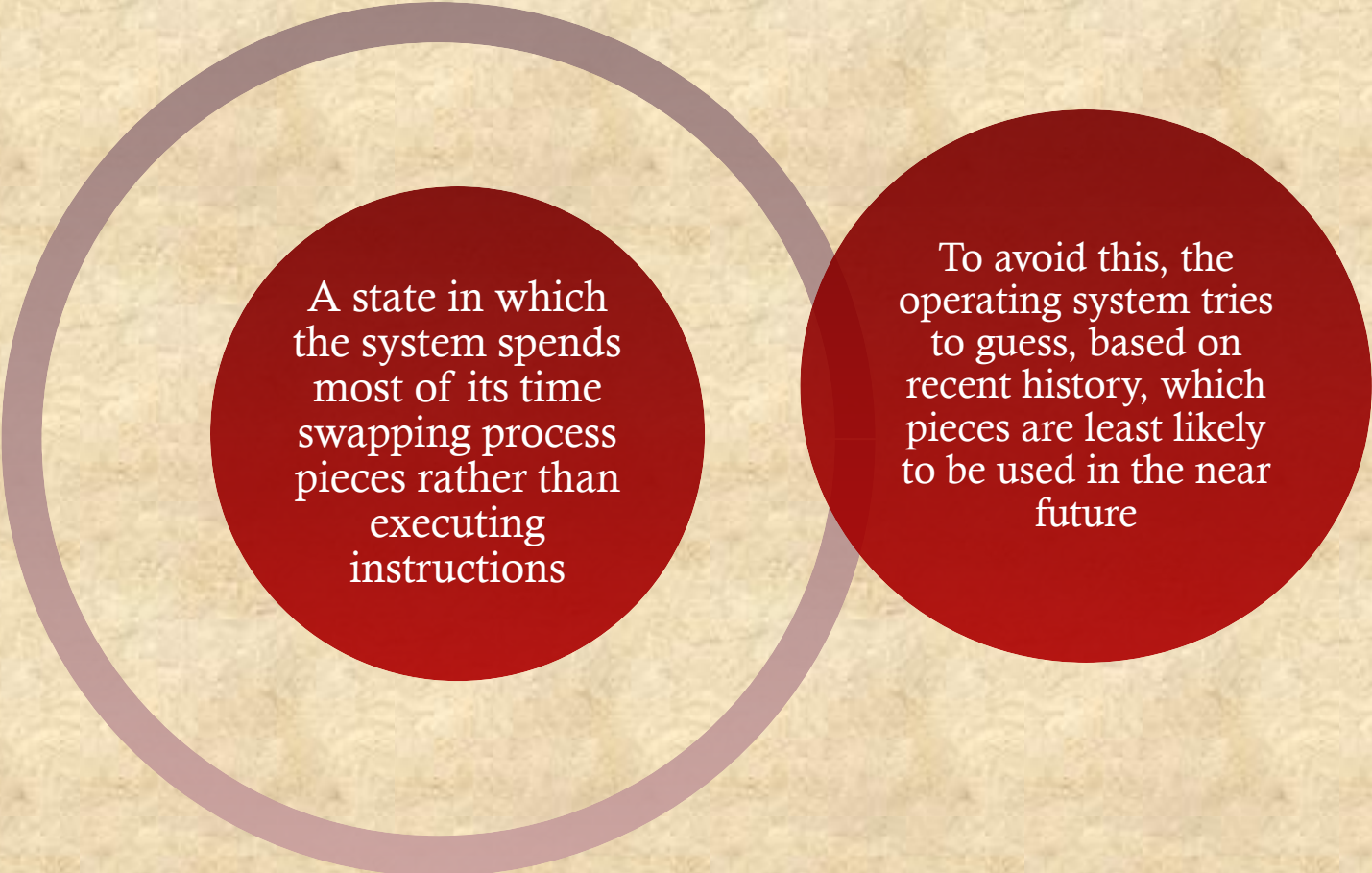main memory, the actual RAM

Virtual memory

memory on disk

allows for effective multiprogramming and relieves the user of tight constraints of main memory

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 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

Cont...

Thrashing" refers to a situation where a computer's performance deteriorates significantly because it spends an excessive amount of time swapping data between physical memory (RAM) and virtual memory (usually a hard drive or SSD).

Thrashing occurs when the system is constantly paging data in and out of RAM because it lacks enough physical memory to meet the demands of running processes.Example::

Imagine you have a computer with only 2GB of RAM, and you are running several memory-intensive applications simultaneously, such as a video editor, a web browser with multiple tabs open, a virtual machine, and a large spreadsheet. Each of these applications requires a significant amount of memory to run smoothly.

Now, as the combined memory requirements of these applications exceed the available physical RAM (2GB), the operating system begins to use a technique called paging or swapping to manage memory. It starts moving portions of data from RAM to the hard drive (or SSD) to free up space in RAM for the currently active processes. This process is known as "paging out."

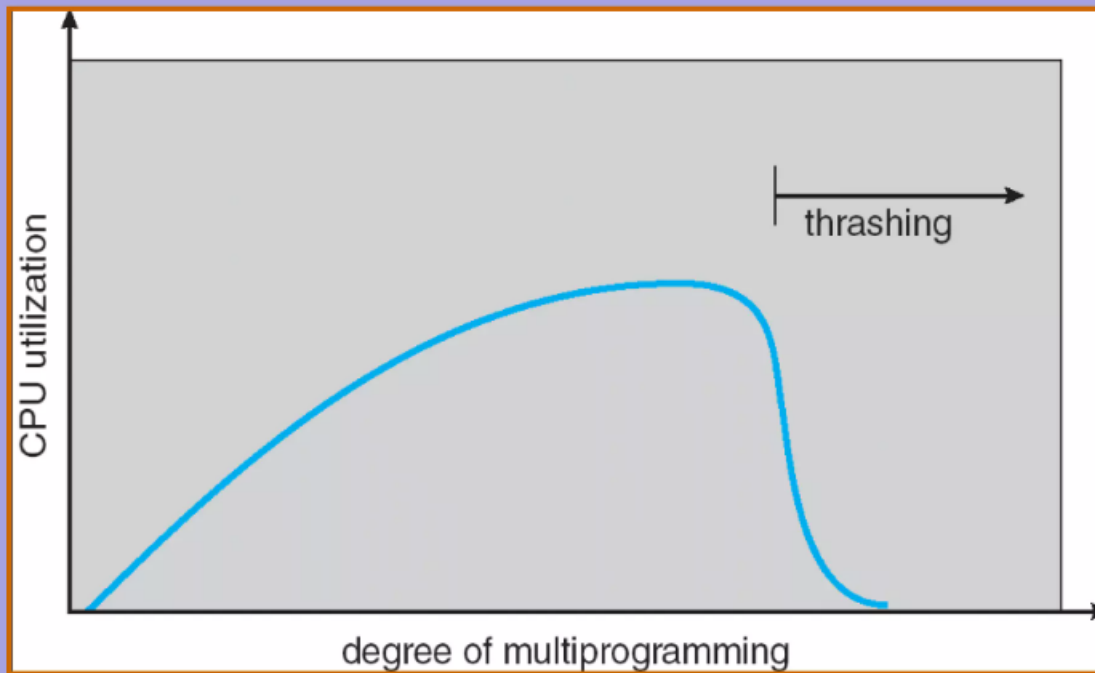The CPU sp... ...executing the actual in...

Disk I/O bec... ...and writing data betwee...

Overall syst... ...ecause the system is ov...

In such a sc... ...e times, frozen scree...

The system ...ng data in and out of memory, and it's unable to make meaningful progress with its workload.

Thrashing

CPU utilization

thrashing

degree of multiprogramming

Why does thrashing occur?

Σ size of locality > total memory size

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
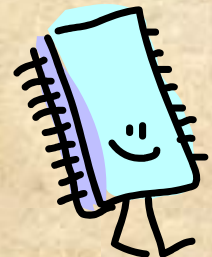
- Avoids thrashing

Support Needed for Virtual Memory

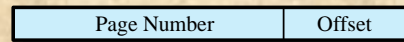**For virtual memory to be practical and effective:**

- 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 contains the frame number of the corresponding page in main memory
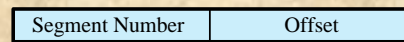
Virtual Address

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
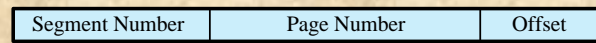
Segment Number	Page Number	Offset

Segment Table Entry

Control Bits	Length	Segment Base

Page Table Entry

P	M	Other Control Bits	Frame Number
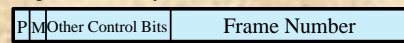
P = present bit
M = Modified bit

(c) Combined segmentation and paging

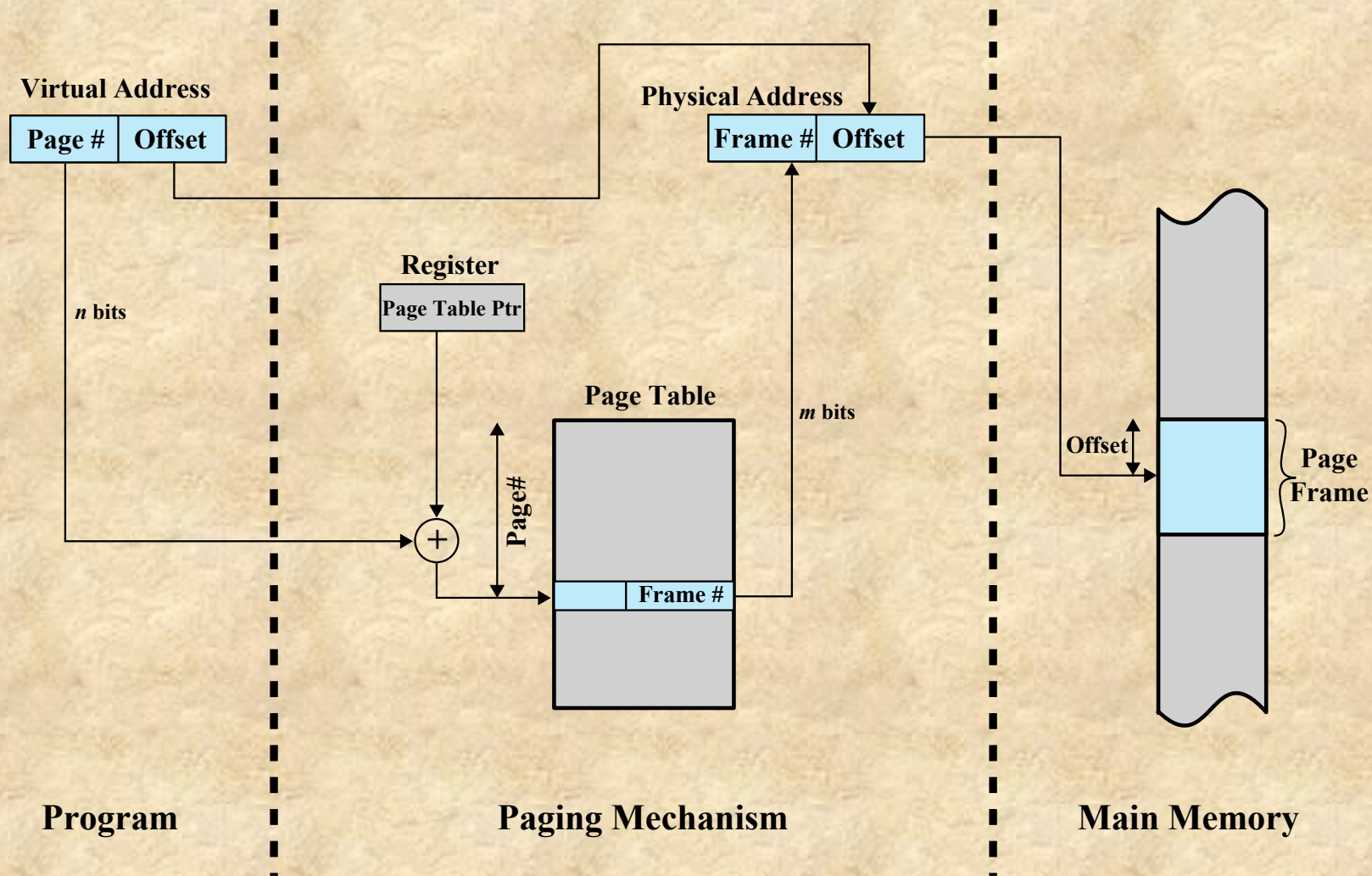**Figure 8.1 Typical Memory Management Formats**

Figure 8.2   Address Translation in a Paging System

4-kbyte root page table

4-Mbyte user page table

4-Gbyte user address space
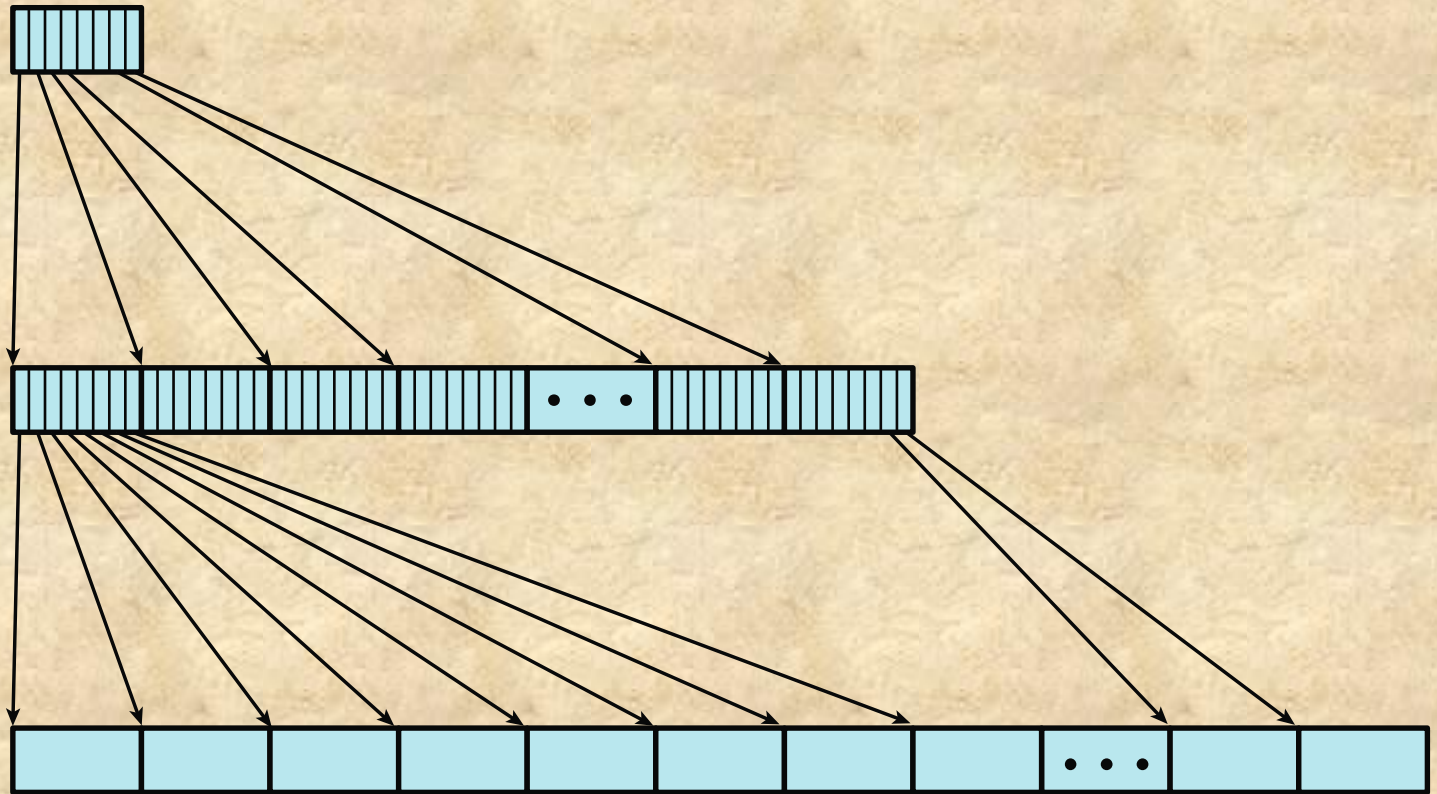
Figure 8.3 A Two-Level Hierarchical Page Table

Virtual Address

10 bits	10 bits	12 bits

Frame #	Offset

Root page table ptr

+

Root page table (contains 1024 PTEs)

+

4-kbyte page table (contains 1024 PTEs)

Page Frame

Program

Paging Mechanism

Main Memory

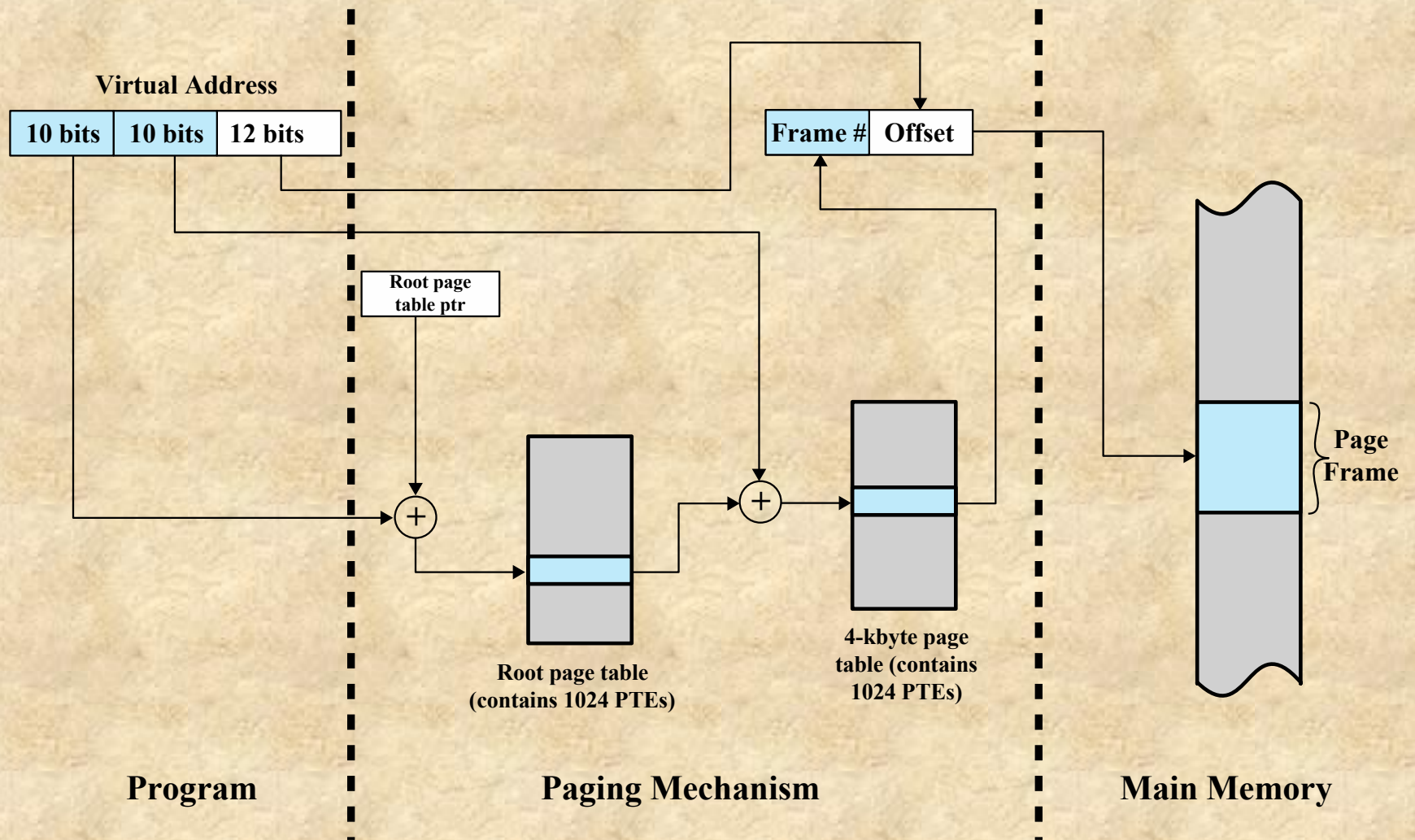**Figure 8.4 Address Translation in a Two-Level Paging System**

Inverted Page Table

- 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 or virtual pages supported

- Structure is called *inverted* because it indexes page table entries by frame number rather than by virtual page number
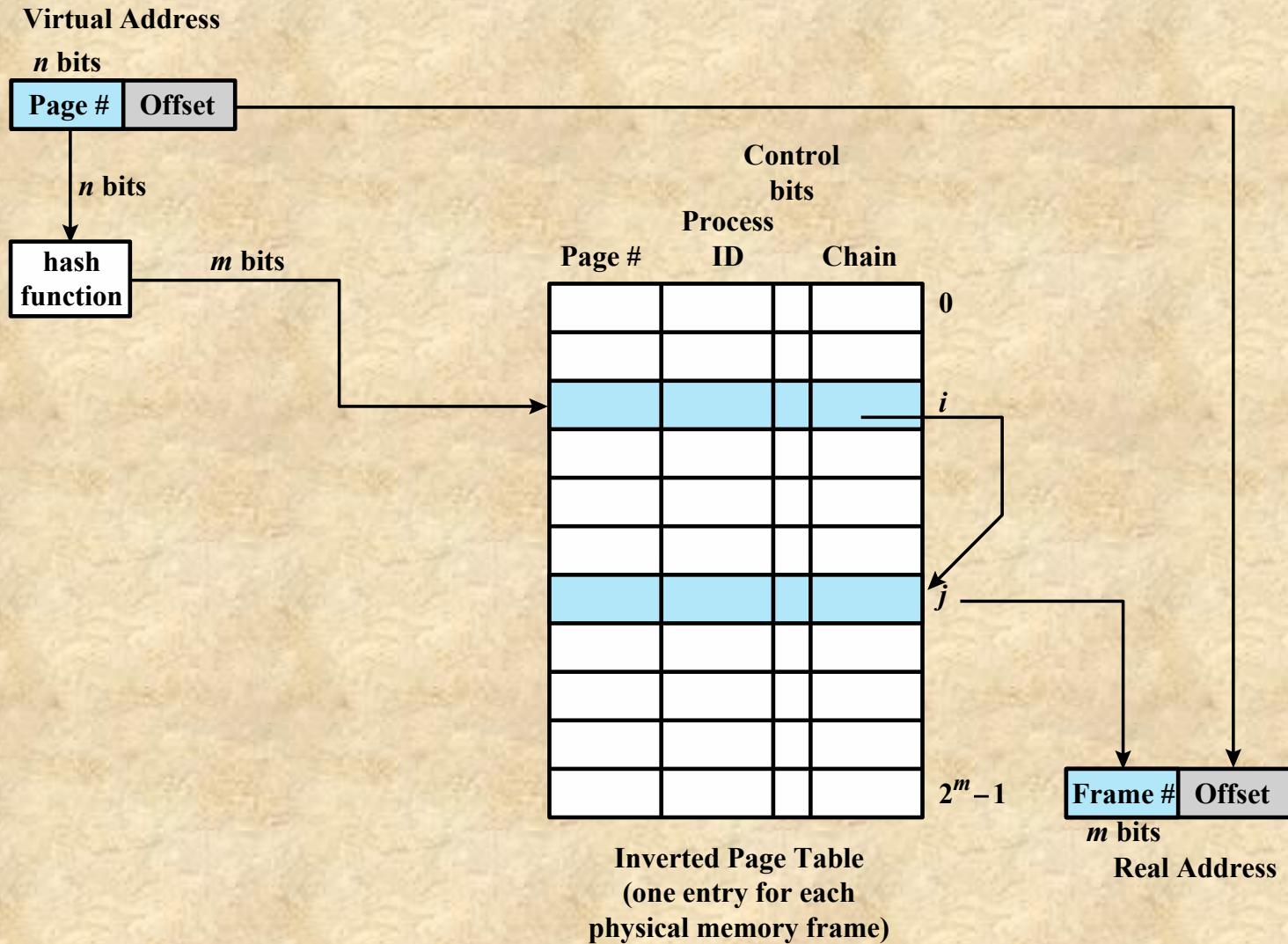
Virtual Address

n **bits**

Page #	Offset

n **bits**

hash function — m **bits** →

Control bits

Page #	Process ID		Chain	
				0
				i
				j
				$2^m - 1$

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

Frame #	Offset

m **bits**

Real Address

Figure 8.5 Inverted Page Table Structure

Inverted Page Table

Each entry in the page table includes:

Page number	Process identifier	Control bits	Chain pointer
	• the process that owns this page	• includes flags and protection and locking information	• the index value of the next entry in the chain
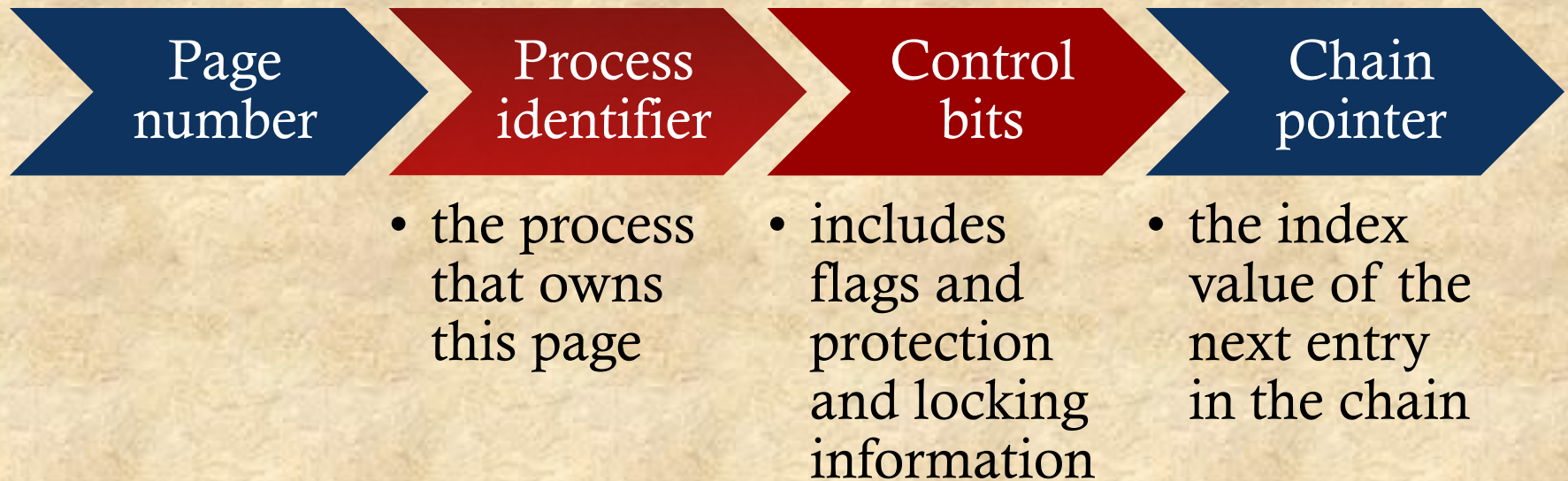
Translation Lookaside Buffer (TLB)

- 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*
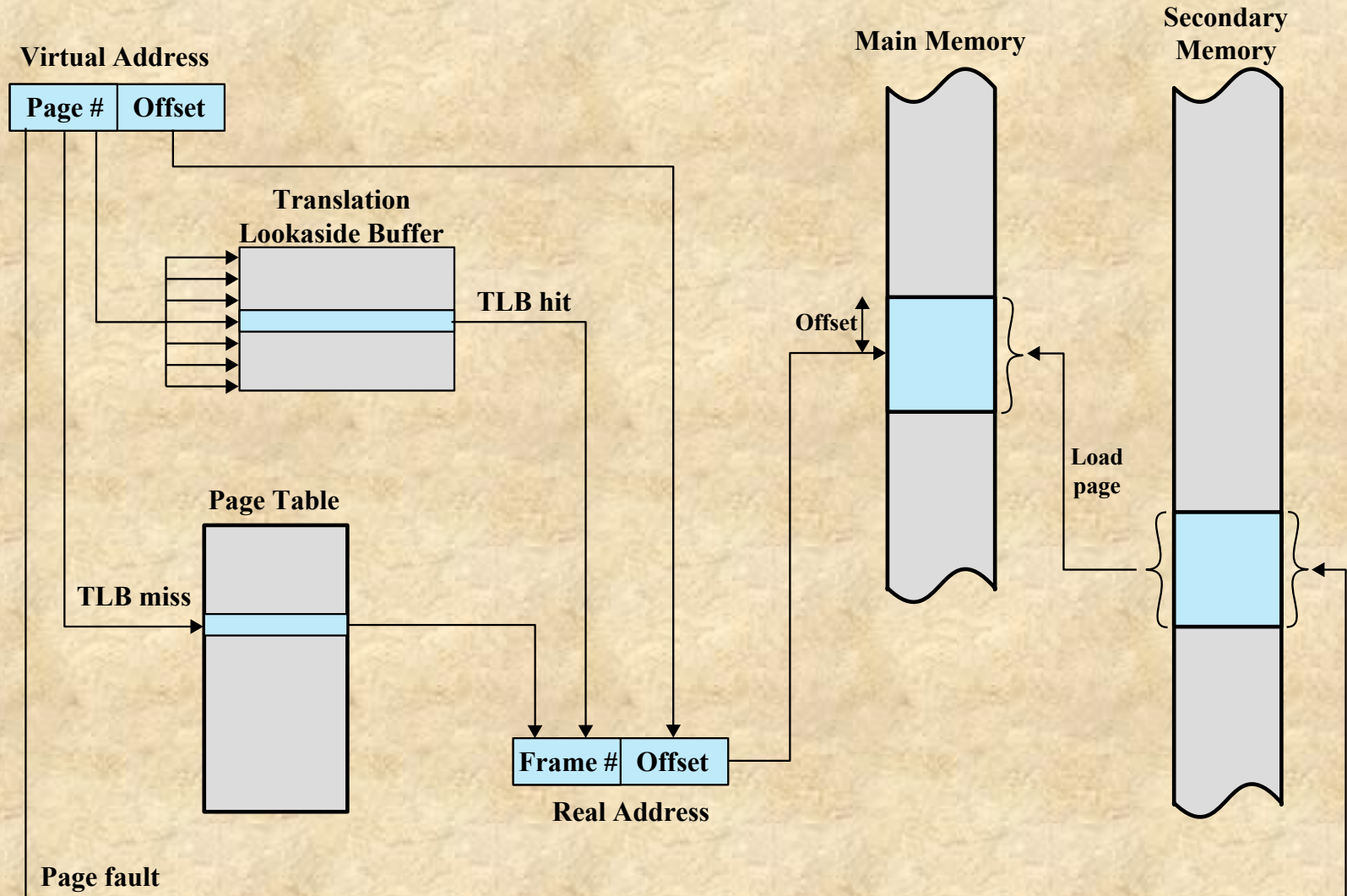
Virtual Address

| Page # | Offset |

Translation Lookaside Buffer

TLB hit

Main Memory

Secondary Memory

Offset

Page Table

TLB miss

Load page

| Frame # | Offset |

Real Address

Page fault

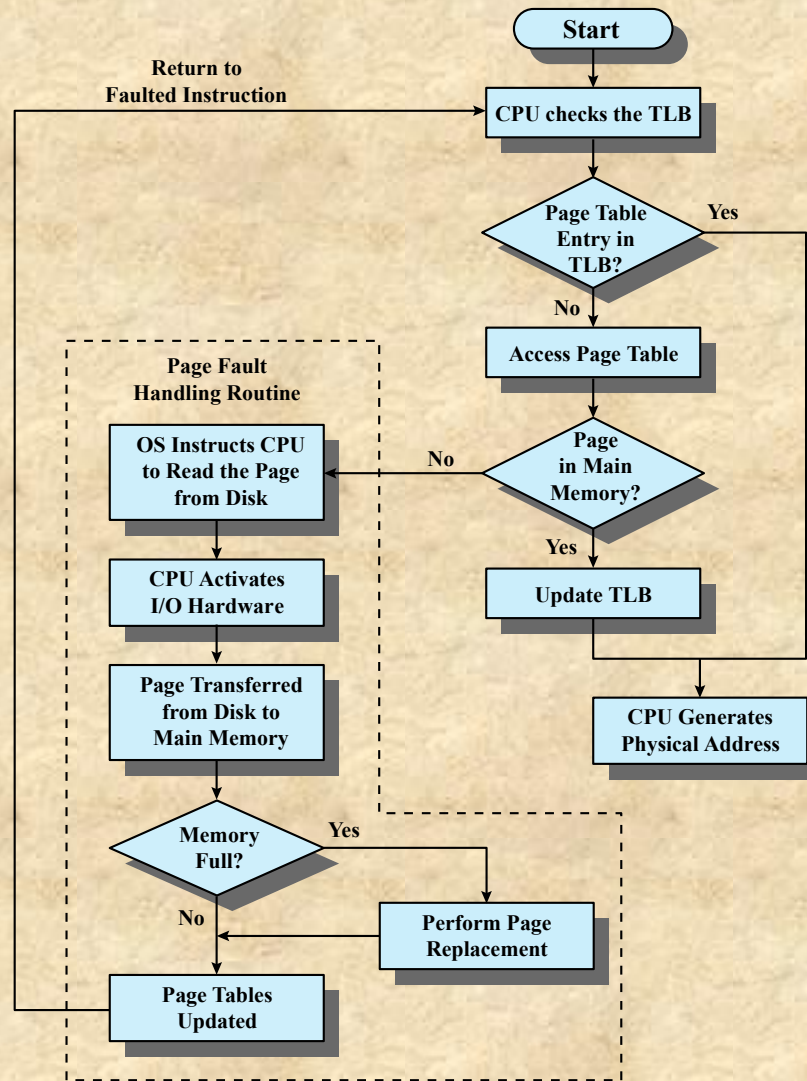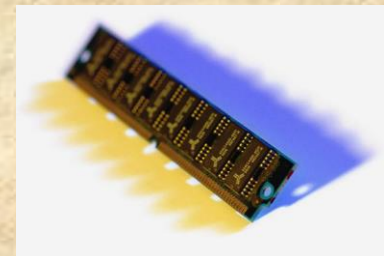**Figure 8.6 Use of a Translation Lookaside Buffer**

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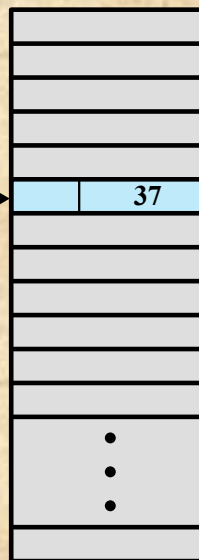
Associative Mapping

- 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

Virtual Address

Page #   Offset

| 5 | 502 |

Virtual Address

Page #   Offset

| 5 | 502 |

Page #   PT Entries

| 19 |
| 511 |
| 37 |
| 27 |
| 14 |
| 1 |
| 211 |
| 5 | 37 |
| 90 |

Translation Lookaside Buffer

| 37 |
... (37 in page table)

| 37 | 502 |

Frame # Offset
Real Address
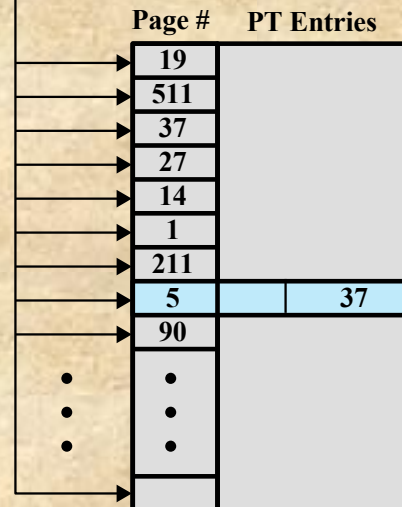
| 37 | 502 |

Frame # Offset
Real Address

Page Table

(a) Direct mapping

(b) Associative mapping

Figure 8.8   Direct Versus Associative Lookup for Page Table Entries
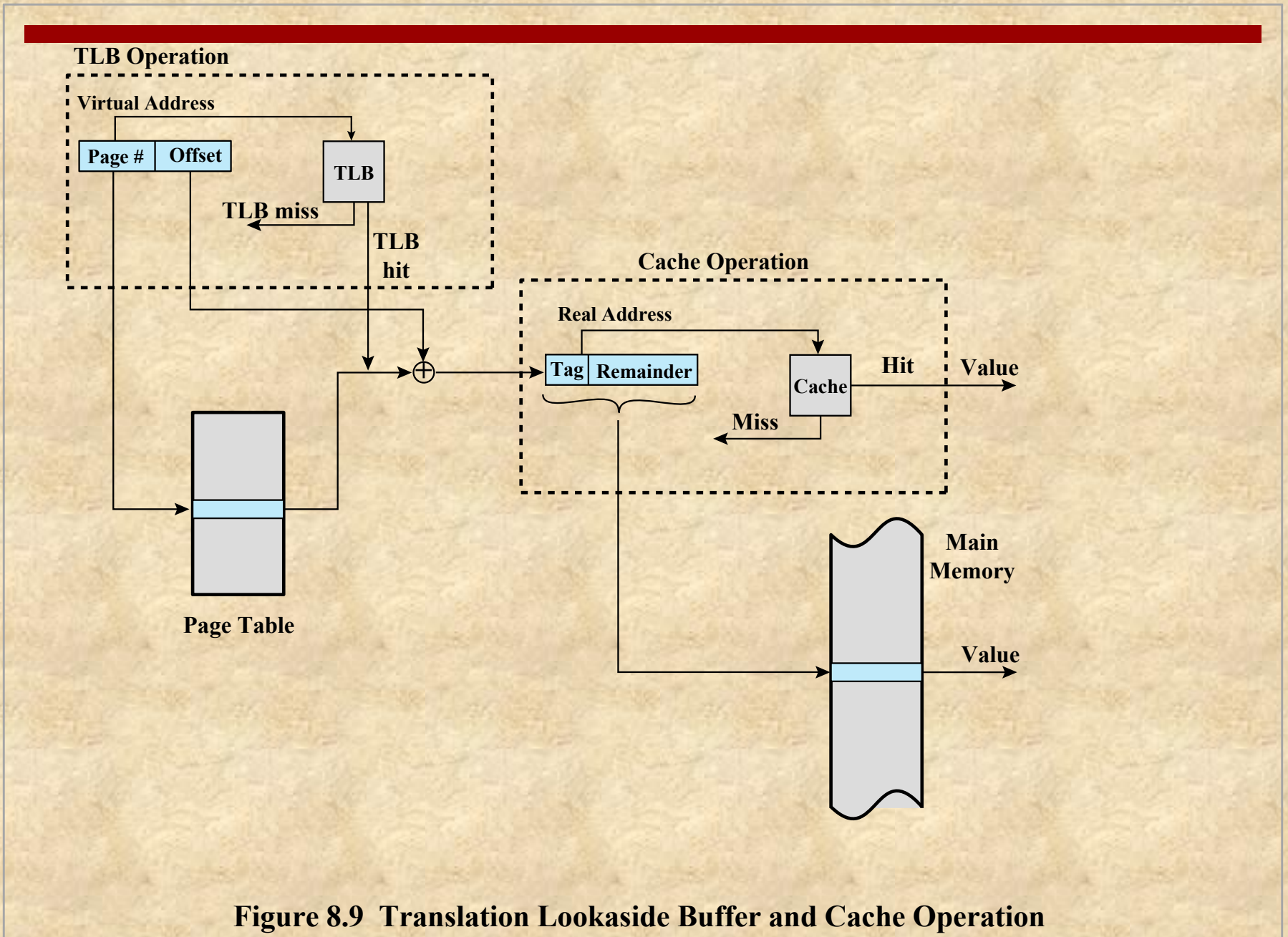
Figure 8.9 Translation Lookaside Buffer and Cache Operation

Page Size

- The smaller the page size, the lesser the amount of 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
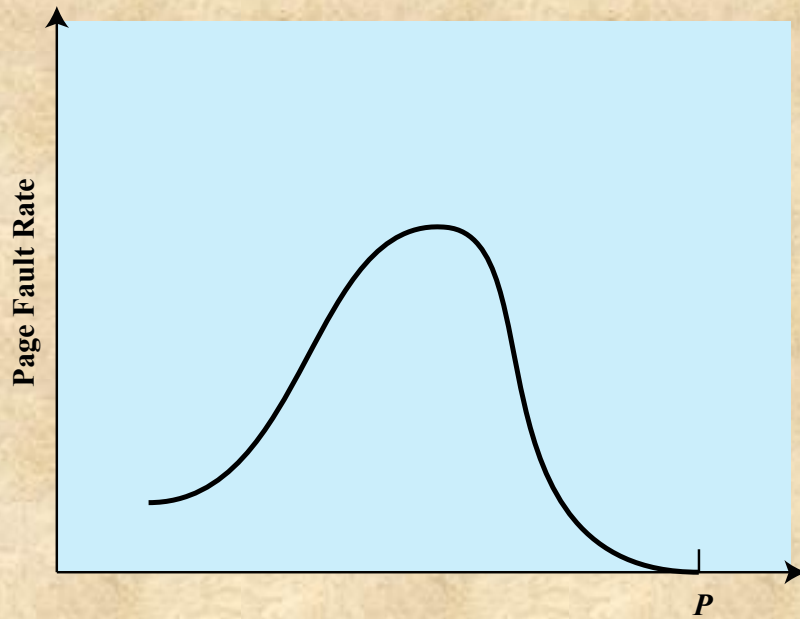
(a) Page Size
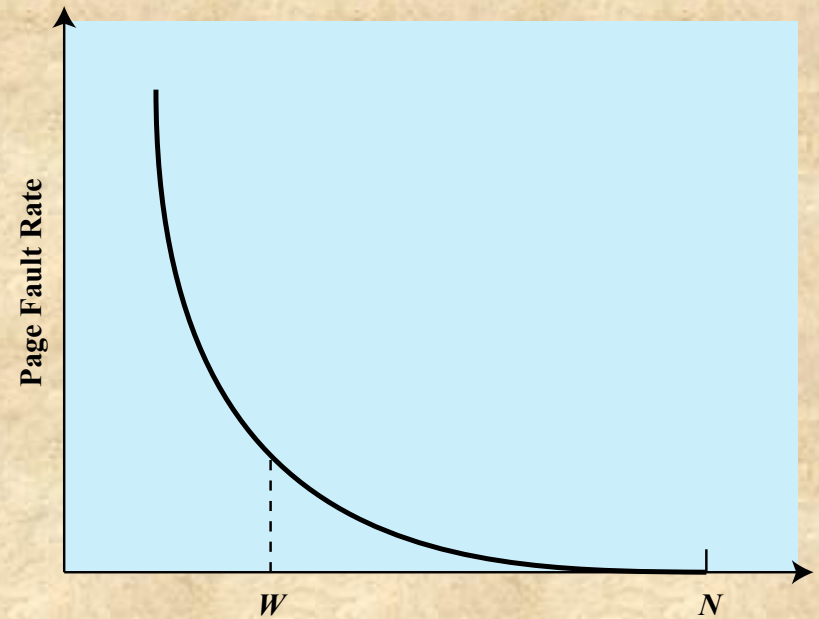
(b) Number of Page Frames Allocated

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

Figure 8.10 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

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

most obvious on personal computers where applications are becoming increasingly complex
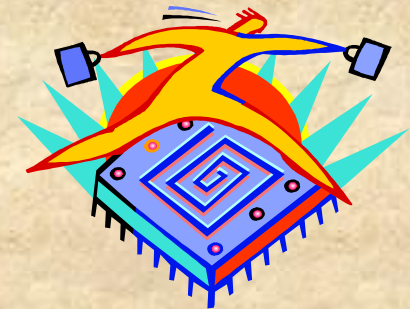
Segmentation

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

why segmentation faults occur?

1.Accessing Unauthorized Memory: One of the most common causes of a segmentation fault is when a program tries to read from or write to a memory location that it doesn't have permission to access. This could be memory outside the program's allocated address space or memory that belongs to another process.

2.Dereferencing a Null Pointer: When a program attempts to dereference a null pointer (a pointer that doesn't point to a valid memory location), it results in a segmentation fault. Null pointers typically occur when a program tries to access an uninitialized pointer or when memory allocation fails.

3.Buffer Overflow: If a program writes data beyond the bounds of an allocated buffer (e.g., an array or a string), it can overwrite memory that it shouldn't, leading to a segmentation fault. Buffer overflows are a common cause of security vulnerabilities and crashes.

4.Stack Overflow: When a program's call stack grows beyond the available stack space (often due to excessive recursion or excessive local variable usage), it can cause a segmentation fault.
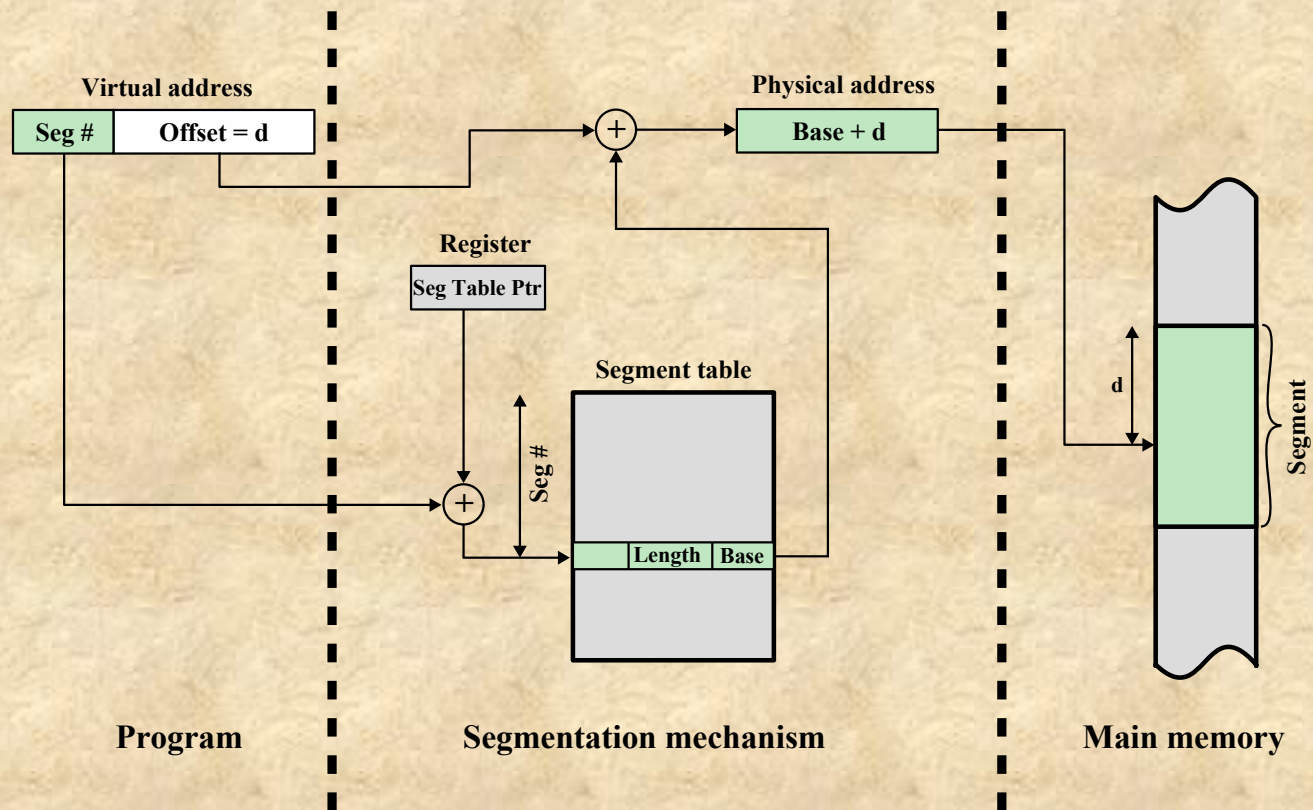
Figure 8.11   Address Translation in a Segmentation System

Drawback of paging and segmentation

Internal Fragmentation: suffer from internal fragmentation. Memory is divided into fixed-size blocks (pages), and when a process doesn't fully utilize a page, the remaining space in that page is wasted, leading to internal fragmentation. This can be inefficient in terms of memory usage.

Page Table Overhead: To keep track of the mapping between logical addresses and physical addresses, a page table is maintained by the operating system. For processes with large address spaces, the page table can become quite large, consuming a significant amount of memory and causing overhead in terms of memory access times.

Increased I/O Overhead: Paging involves frequent reading and writing of pages between RAM and secondary storage (e.g., a hard drive). This can lead to increased I/O operations, potentially causing performance bottlenecks, especially if there's high disk activity.

Complex Memory Management: Managing the page table, handling page faults, and swapping pages in and out of memory can be complex and resource-intensive tasks for the operating system.

■ Segmentation

1.External Fragmentation: When segments of varying sizes are allocated and deallocated, free memory becomes fragmented over time.
This fragmentation can make it challenging to allocate contiguous blocks of memory for processes with specific segment size requirements.
2.Complex Addressing: each address is specified by both a segment identifier and an offset within that segment. This can lead to complex addressing and potentially inefficient memory access times compared to a flat memory model like paging.
3.Dynamic Growth of Segments: Managing dynamically growing or shrinking segments can be complex. When a segment needs to grow beyond its originally allocated size, the operating system must find additional memory space, possibly leading to fragmentation issues.
4.Security Concerns: Segmentation can introduce security vulnerabilities if proper access controls are not enforced correctly. Unauthorized access to segments can lead to data breaches.

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
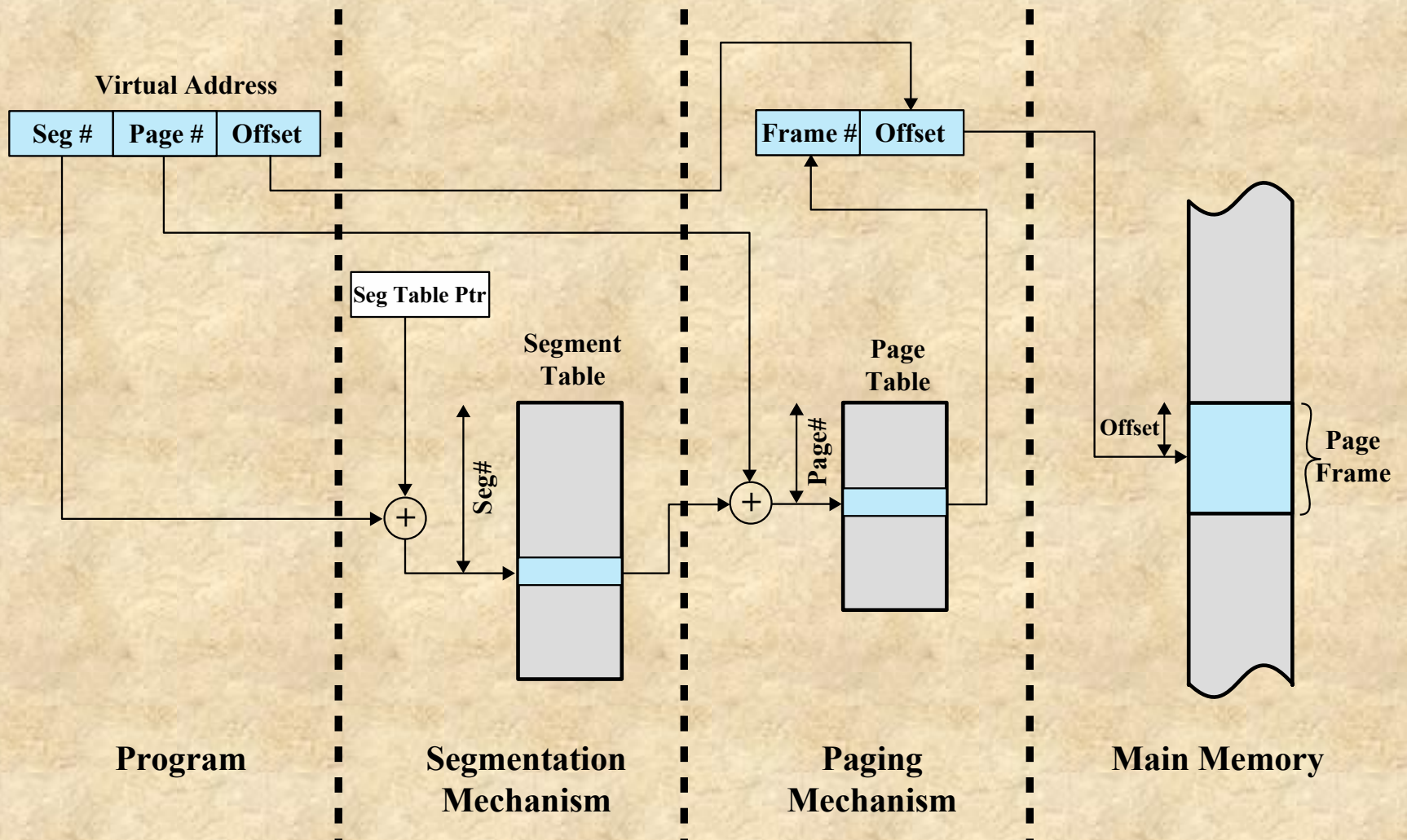
Paging is transparent to the programmer

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
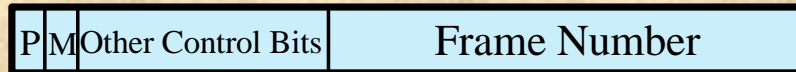
P= present bit
M = Modified bit

(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 segments referencing multiple processes
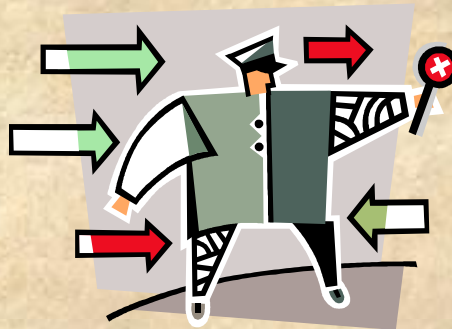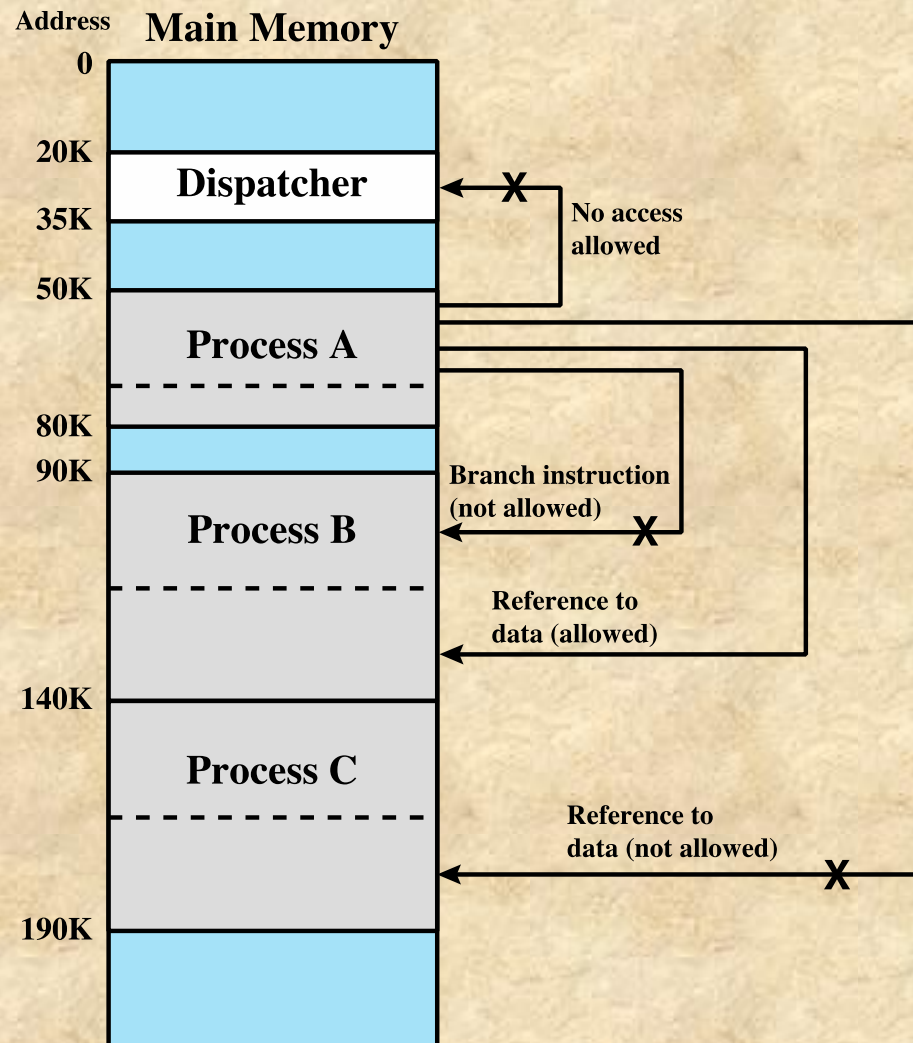
Figure 8.13 Protection Relationships Between Segments

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

- **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
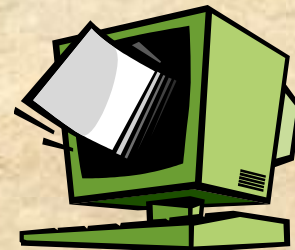
Prepaging

- **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

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
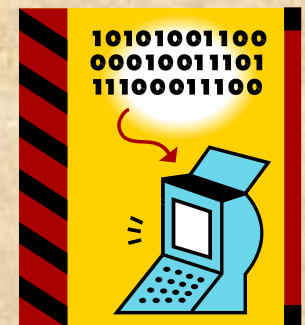
Clock Policy

- 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
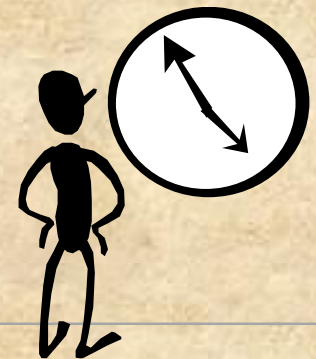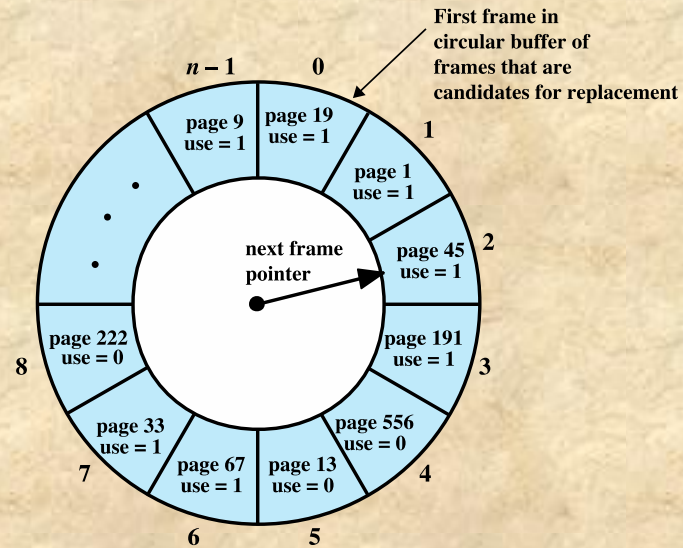
(a) State of buffer just prior to a page replacement
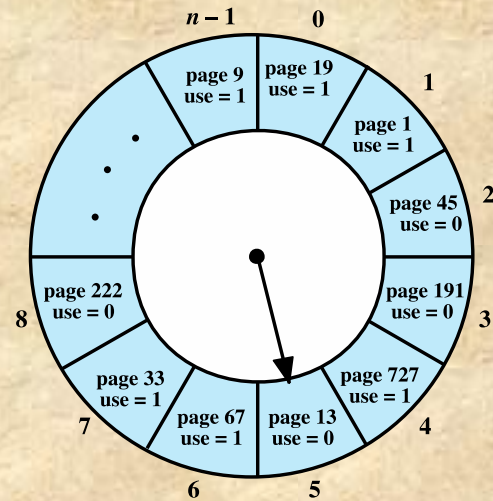
(b) State of buffer just after the next page replacement

Figure 8.15   Example of Clock Policy Operation
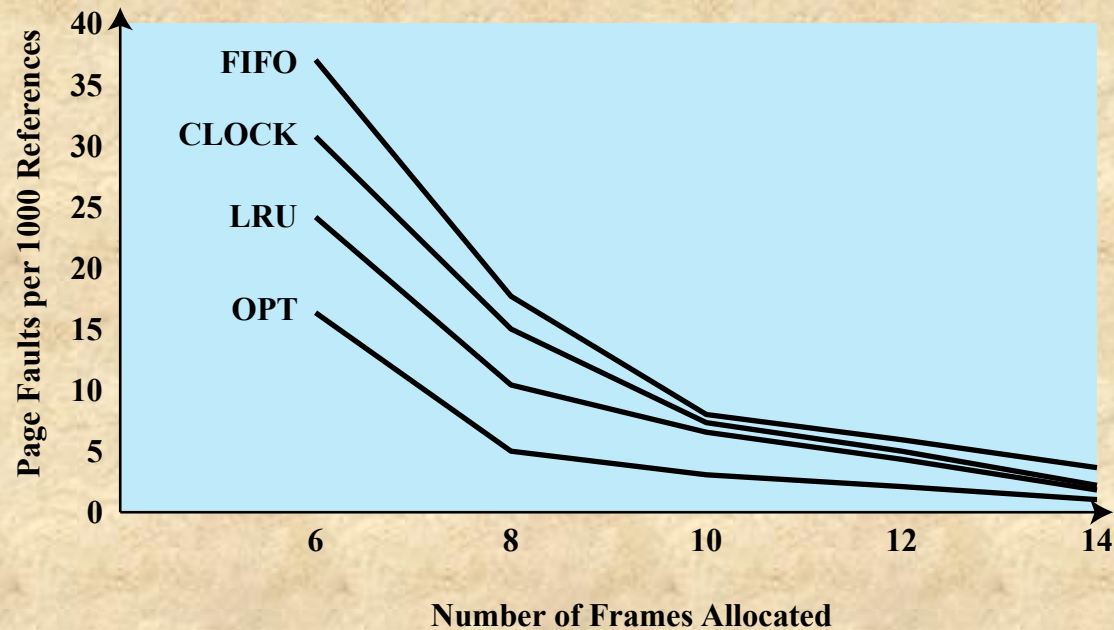
Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Page Buffering

- Improves paging performance and allows the use of a simpler page replacement policy

A replaced page is not lost, but rather assigned to one of two lists

Free page list

Modified page list

list of page frames available for reading in pages

pages are written out in clusters
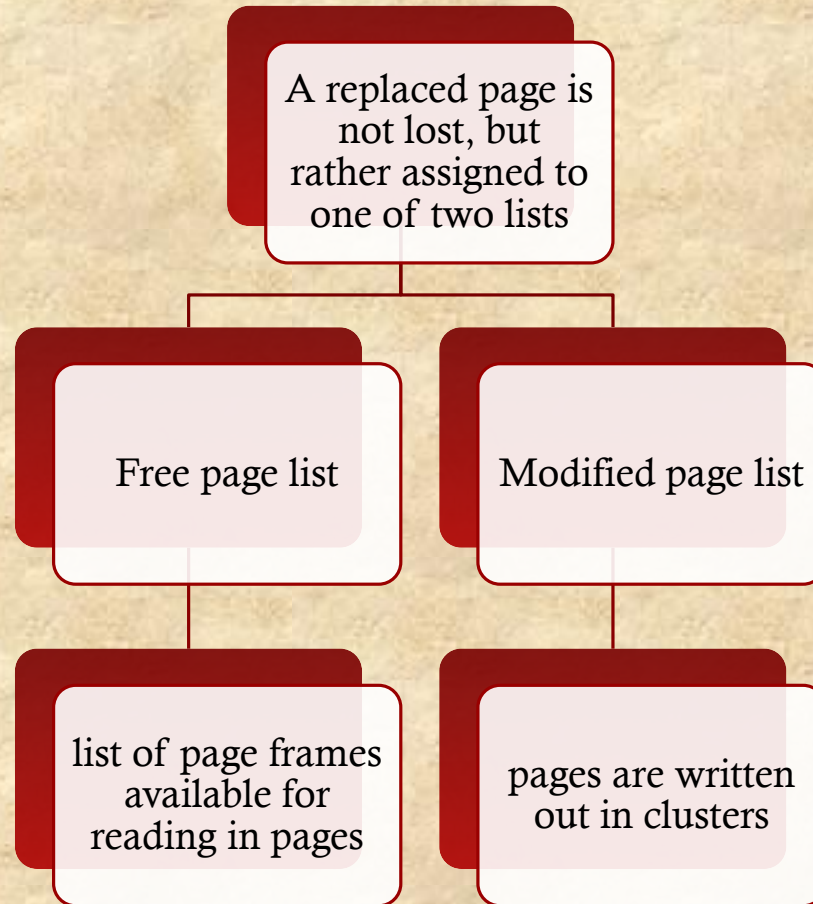
Replacement Policy and Cache Size

- With large caches, replacement of pages can have a performance impact
  - if the page frame selected for replacement is in the cache, that cache block is lost as well as the page that it holds
  - in systems using page buffering, cache performance can be improved with a policy for page placement in the page buffer
  - most operating systems place pages by selecting an arbitrary page frame from the page buffer
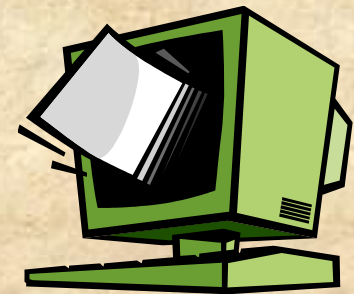
Resident Set Management

- The OS must decide how many pages to bring into main memory
  - the smaller the amount of memory allocated to each process, the more processes can reside in memory
  - small number of pages loaded increases page faults
  - beyond a certain size, further allocations of pages will not effect the page fault rate

Resident Set Size

Fixed-allocation

- gives a process a fixed number of frames in main memory within which to execute

  - when a page fault occurs, one of the pages of that process must be replaced

Variable-allocation

- allows the number of page frames allocated to a process to be varied over the lifetime of the process

Replacement Scope

- The scope of a replacement strategy can be categorized as *global* or *local*
  - both types are activated by a page fault when there are no free page frames

Local

- chooses only among the resident pages of the process that generated the page fault

Global

- considers all unlocked pages in main memory

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. •Page to be replaced is chosen from among the frames allocated to that process.	•Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Table 8.5 Resident Set Management

Fixed Allocation, Local Scope

- Necessary to 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

- increased processor idle time
- increased time spent in swapping

Variable Allocation Global Scope

- Easiest to implement
    - adopted in a number of operating systems

- OS maintains a list of free frames

- Free frame is added to resident set of process when a page fault occurs

- If no frames are available the OS must choose a page currently in memory

- One way to counter potential problems is to use page buffering

Variable Allocation Local Scope

- When a new process is loaded into main memory, allocate to it a certain number of page frames as its resident set

- When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault

- Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance

Variable Allocation Local Scope

- Decision to increase or decrease a resident set size is based on the assessment of the likely future demands of active processes

Key elements:

- criteria used to determine resident set size
- the timing of changes

Sequence of Page References	Window Size, D			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Figure 8.17 Working Set of Process as Defined by Window Size

Figure 8.18 Typical Graph of Working Set Size [MAEK87]

Page Fault Frequency (PFF)

- Requires a use bit to be associated with each page in memory

- Bit is set to 1 when that page is accessed

- When a page fault occurs, the OS notes the virtual time since the last page fault for that process

- Does not perform well during the transient periods when there is a shift to a new locality

Variable-Interval Sampled Working Set (VSWS)

- Evaluates the working set of a process at sampling instances based on elapsed virtual time

- Driven by three parameters:

| the minimum duration of the sampling interval | the maximum duration of the sampling interval | the number of page faults that are allowed to occur between sampling instances |

Cleaning Policy

- Concerned with determining when a modified page should be written out to secondary memory

Demand Cleaning

a page is written out to secondary memory only when it has been selected for replacement

Precleaning

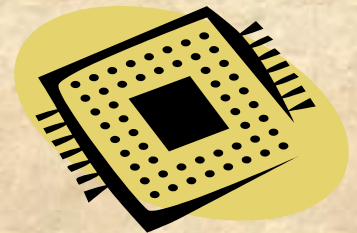allows the writing of pages in batches

Load Control

- Determines the number of processes that will be resident in main memory
    - *multiprogramming* level

- Critical in effective memory management

- 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
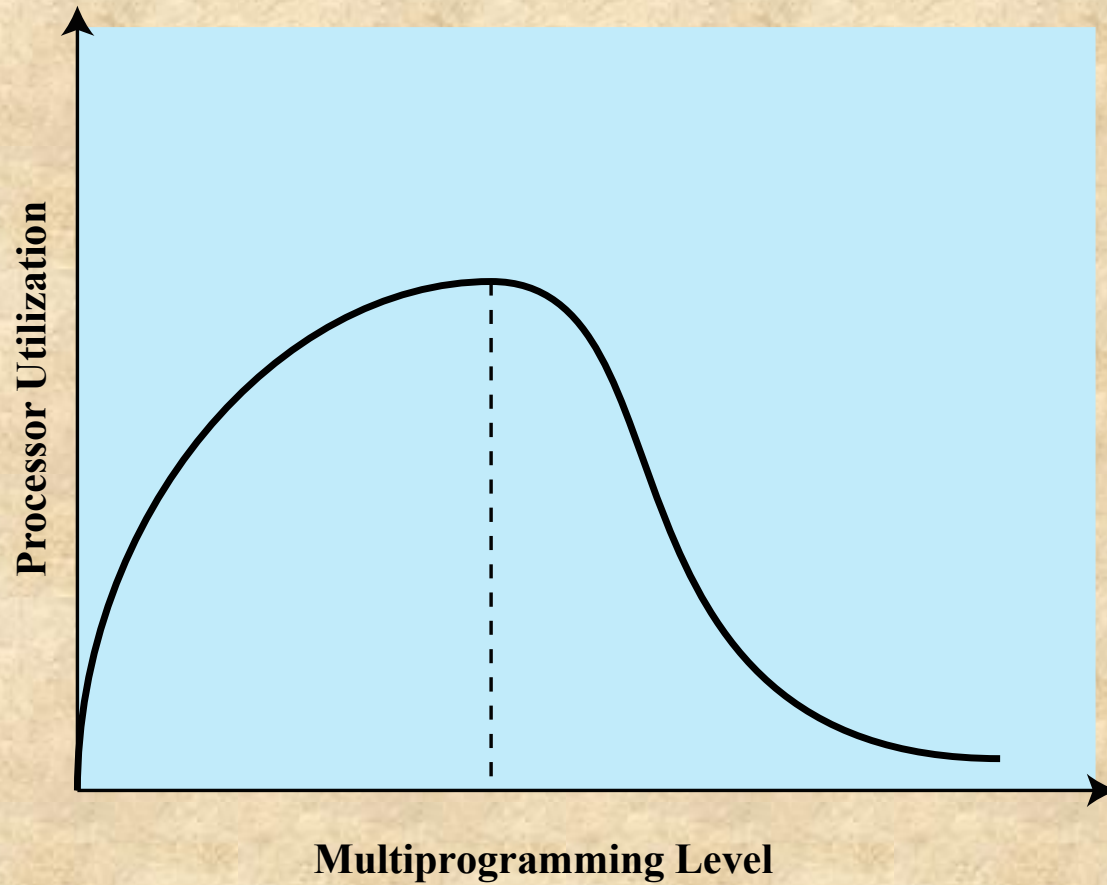
Figure 8.19 Multiprogramming Effects

Process Suspension

■ If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out

Six possibilities exist:

- lowest-priority process
- faulting process
- last process activated
- process with the smallest resident set
- largest process
- process with the largest remaining execution window

UNIX

- Intended to be machine independent so its memory management schemes will vary
  - early UNIX: variable partitioning with no virtual memory scheme
  - current implementations of UNIX and Solaris make use of

SVR4 and Solaris use two separate schemes:

- paging system
- kernel memory allocator

Paging System and Kernel Memory Allocator

Paging System

provides a virtual memory capability that allocates page frames in main memory to processes

allocates page frames to disk block buffers

Kernel Memory Allocator

allocates memory for the kernel

Page frame number	Age	Copy on write	Mod-ify	Refe-rence	Valid	Pro-tect

(a) Page table entry

Swap device number	Device block number	Type of storage

(b) Disk block descriptor

Page state	Reference count	Logical device	Block number	Pfdata pointer

(c) Page frame data table entry

Reference count	Page/storage unit number

(d) Swap-use table entry

Figure 8.20 UNIX SVR4 Memory Management Formats

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 0 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 8.6

UNIX SVR4 Memory Management Parameters
(page 1 of 2)

(Table can be found on page 379 in the textbook)

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.

Table 8.6

UNIX SVR4 Memory Management Parameters
(page 2 of 2)

(Table can be found on page 379 in the textbook)

Page Replacement

- The page frame data table is used for page replacement

- Pointers are used to create lists within the table
  - all available frames are linked together in a list of free frames available for bringing in pages
  - when the number of available frames drops below a certain threshold, the kernel will steal a number of frames to compensate

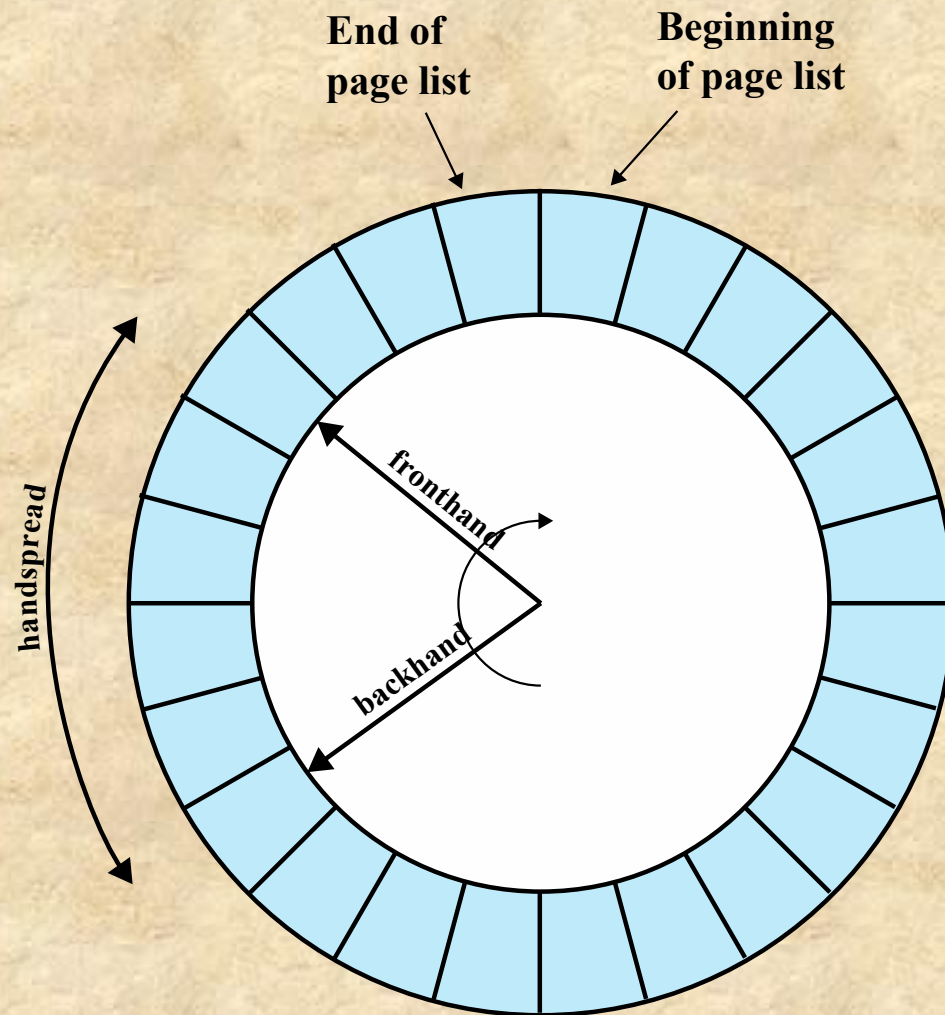**Figure 8.21 Two-Handed Clock Page-Replacement Algorithm**

Kernel Memory Allocator

- The kernel generates and destroys small tables and buffers frequently during the course of execution, each of which requires dynamic memory allocation.

- Most of these blocks are significantly smaller than typical pages (therefore paging would be inefficient)

- Allocations and free operations must be made as fast as possible

Lazy Buddy

- Technique adopted for SVR4

- UNIX often exhibits steady-state behavior in kernel memory demand
    - i.e. the amount of demand for blocks of a particular size varies slowly in time

- Defers coalescing until it seems likely that it is needed, and then coalesces as many blocks as possible
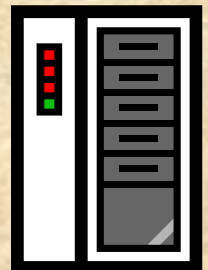
Initial value of D_i is 0

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 2^i and free it globally; coalesce if possible
      $D_i := 0$

Figure 8.22 Lazy Buddy System Algorithm

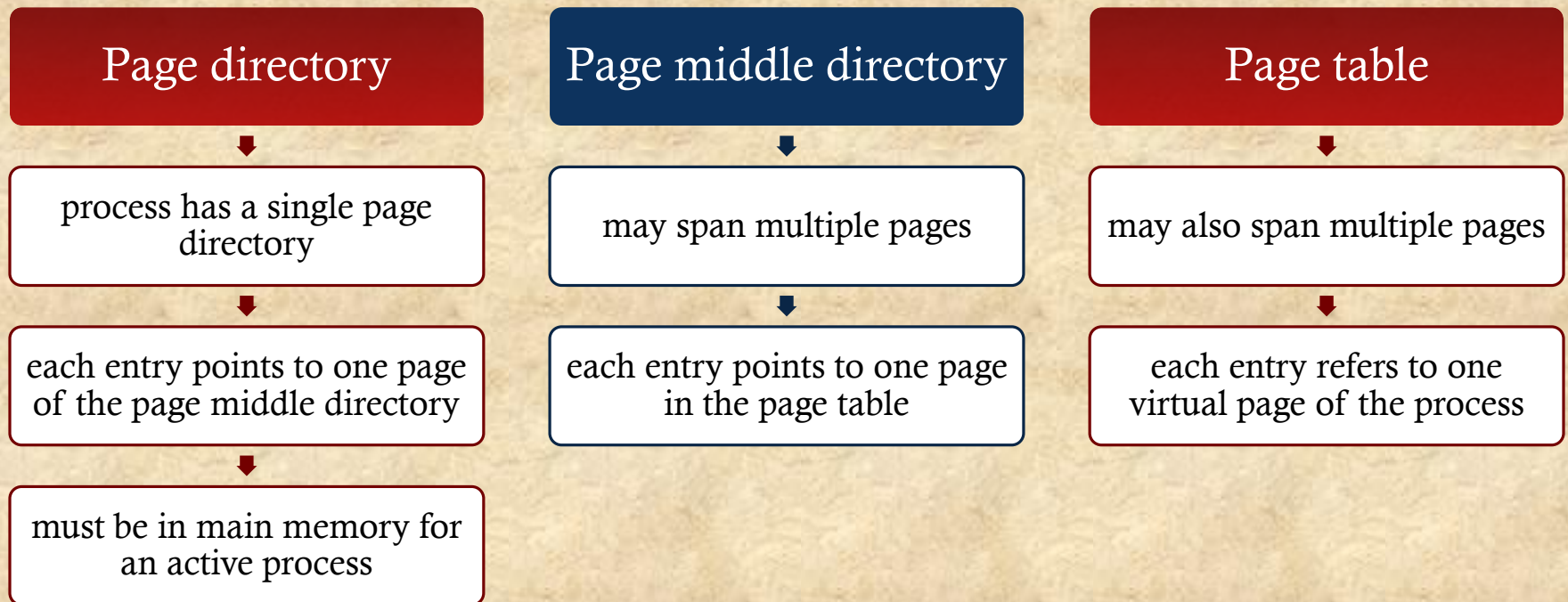# Linux
Memory Management

- Shares many characteristics with UNIX

- Is quite complex

Two main aspects
- process virtual memory
- kernel memory allocation

Linux Virtual Memory

- Three level page table structure:

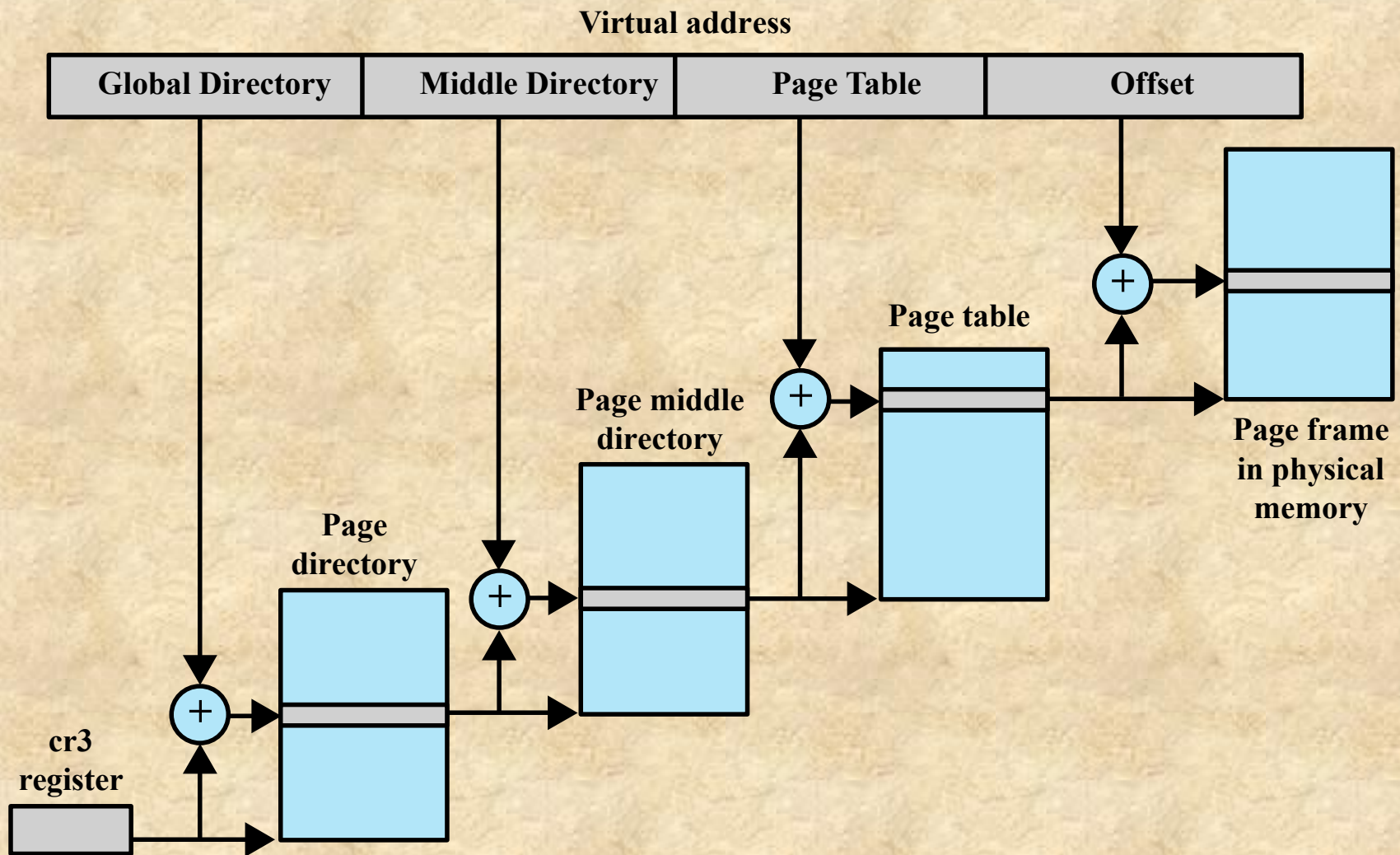| Page directory	Page middle directory	Page table
process has a single page directory	may span multiple pages	may also span multiple pages
each entry points to one page of the page middle directory	each entry points to one page in the page table	each entry refers to one virtual page of the process
must be in main memory for an active process		

Figure 8.23   Address Translation in Linux Virtual Memory Scheme

Linux Page Replacement

- Based on the clock algorithm

- The use bit is replaced with an 8-bit age variable
    - incremented each time the page is accessed

- Periodically decrements the age bits
    - a page with an age of 0 is an "old" page that has not been referenced in some time and is the best candidate for replacement

- A form of least frequently used policy

Inactive

Active

timeout

PG_active = 0
PG_referenced = 0

timeout

PG_active = 1
PG_referenced = 0

used

timeout

used

used

timeout

PG_active = 0
PG_referenced = 1

PG_active = 1
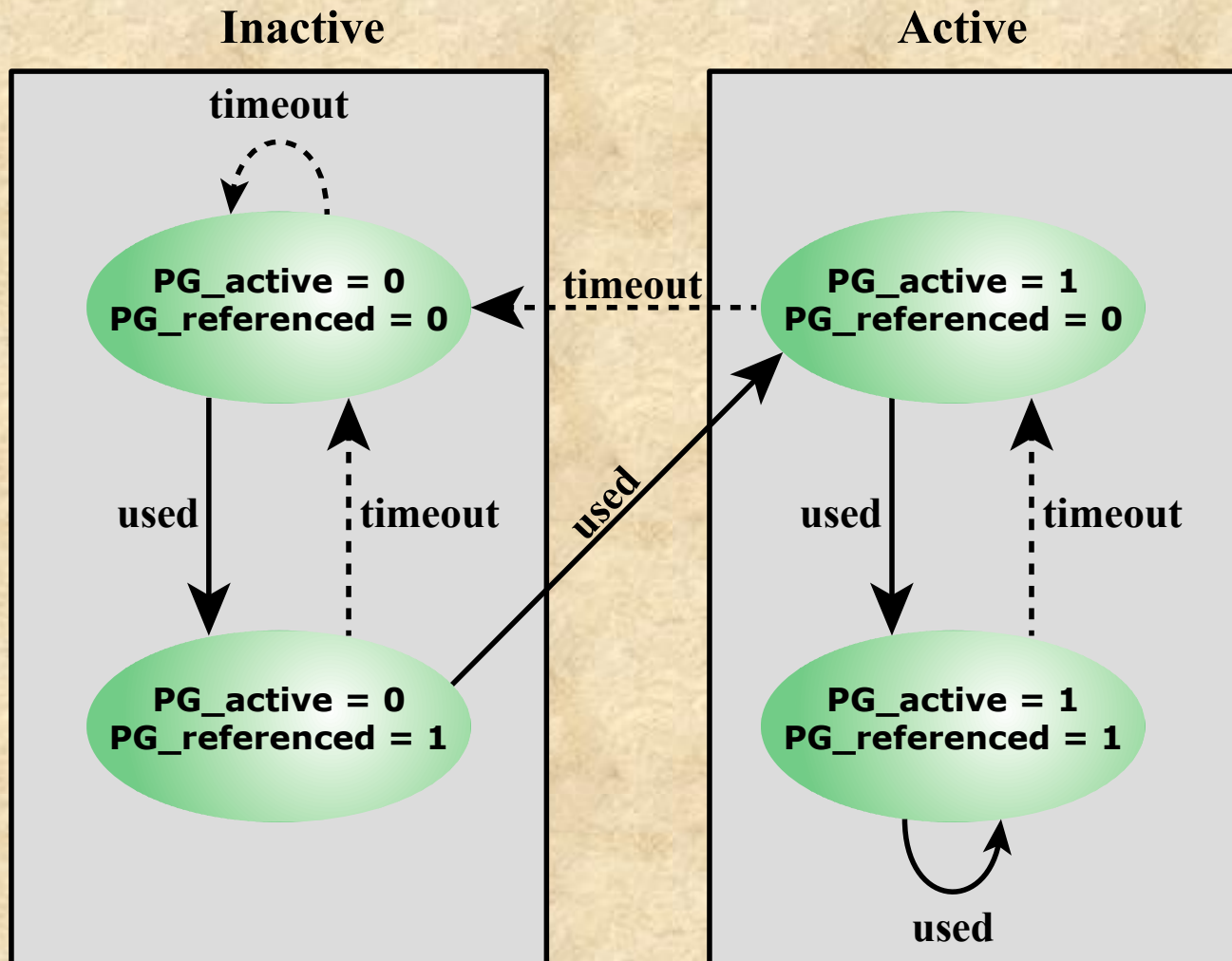PG_referenced = 1

used

Figure 8.24 Linux Page Reclaiming

Kernel Memory Allocation

- Kernel memory capability manages physical main memory page frames
  - primary function is to allocate and deallocate frames for particular uses

Possible owners of a frame include:

- user-space processes
- dynamically allocated kernel data
- static kernel code
- page cache

- A buddy algorithm is used so that memory for the kernel can be allocated and deallocated in units of one or more pages

- Page allocator alone would be inefficient because the kernel requires small short-term memory chunks in odd sizes

- Slab allocation
  - used by Linux to accommodate small chunks

Windows Memory Management

- Virtual memory manager controls how memory is allocated and how paging is performed

- Designed to operate over a variety of platforms

- Uses page sizes ranging from 4 Kbytes to 64 Kbytes
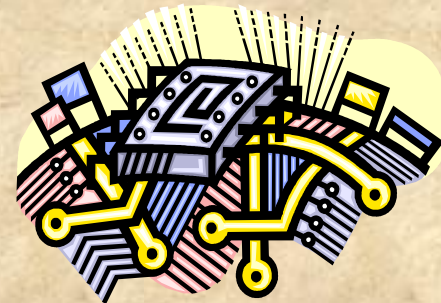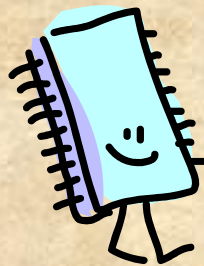
Windows Virtual Address Map

- On 32 bit platforms each user process sees a separate 32 bit address space allowing 4 Gbytes of virtual memory per process
  - by default half is reserved for the OS
- Large memory intensive applications run more effectively using 64-bit Windows
- Most modern PCs use the AMD64 processor architecture which is capable of running as either a 32-bit or 64-bit system

64-Kbyte region for NULL-pointer assignments (inaccessible)

0

2-Gbyte user address space (unreserved, usable)

64-Kbyte region for bad pointer assignments (inaccessible)

2-Gbyte region for the operating system (inacessible)
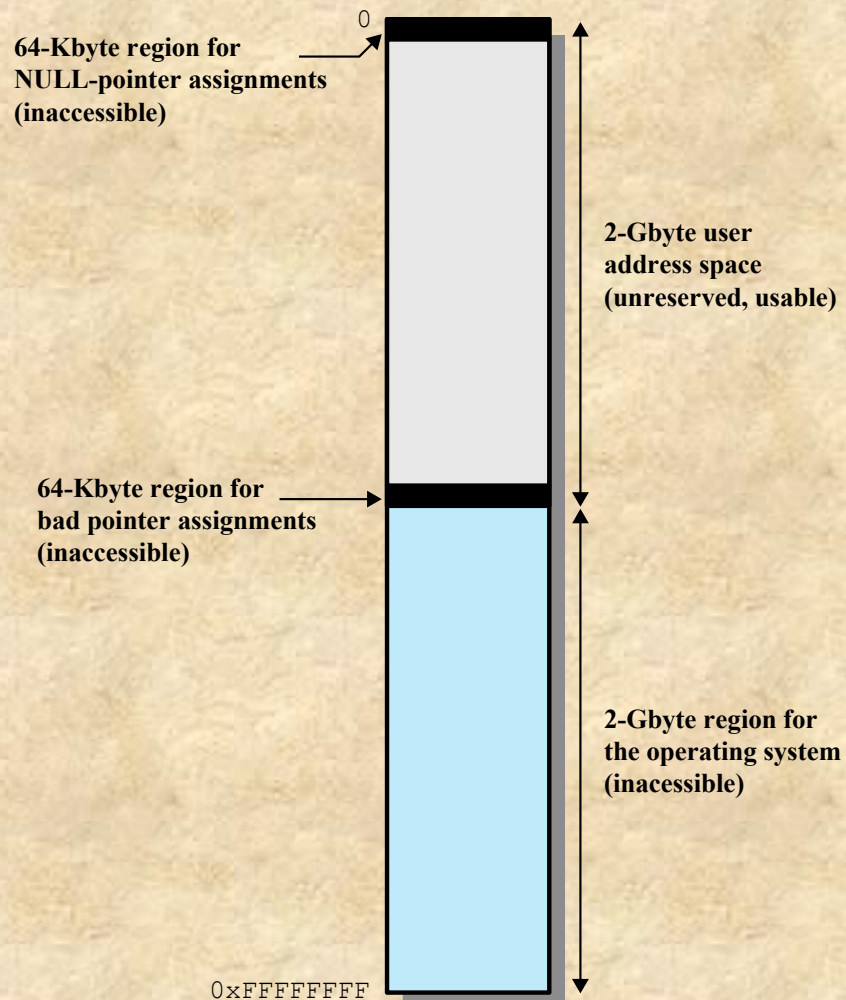
0xFFFFFFFF

**Figure 8.25 Windows Default 32-bit Virtual Address Space**

Windows Paging

- On creation, a process can make use of the entire user space of almost 2 Gbytes

- This space is divided into fixed-size pages managed in contiguous regions allocated on 64 Kbyte boundaries

- Regions may be in one of three states:

available → reserved → committed

Resident Set Management System

- Windows uses variable allocation, local scope

- When activated, a process is assigned a data structure to manage its working set

- Working sets of active processes are adjusted depending on the availability of main memory
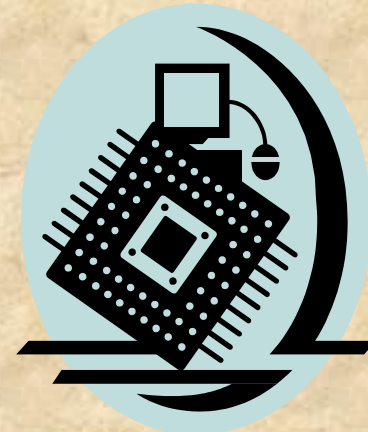
Android Memory Management

- Android includes a number of extensions to the normal Linux kernel memory management facility

- These include:
  - ASHMem
    - this feature provides anonymous shared memory, which abstracts memory as file descriptors
    - a file descriptor can be passed to another process to share memory
  - Pmem
    - this feature allocates virtual memory so that it is physically contiguous
    - this feature is useful for hardware that does not support virtual memory
  - Low Memory Killer
    - this feature enables the system to notify an app or apps that they need to free up memory
    - if an app does not cooperate, it is terminated

questions

1. What is memory management in an operating system?
2. What are the different types of memory in a computer system, and how are they used in memory management?What is the difference between physical memory (RAM) and virtual memory?
3. What is the purpose of a memory manager in an operating system?
4. What is fragmentation, and how does it impact memory management?.
5. What are the advantages and disadvantages of paging and segmentation?
6. What is a page table, and how does it work in paging?
7. What is a TLB (Translation Lookaside Buffer), and how does it improve address translation performance?
8. Explain the concept of page replacement algorithms in paging. Provide examples of such algorithms.
9. What is thrashing in the context of memory management, and how can it be prevented or mitigated?
10. What is segmentation fault (segfault), and why does it occur?
11. What is virtual memory swapping, and when does it occur?
12. how does memory protection work in an operating system?
13. Discuss the concept of demand paging and its benefits.
14. What is the role of the MMU (Memory Management Unit) in memory management?

Summary

- Hardware and control structures
    - Locality and virtual memory
    - Paging
    - Segmentation
    - Combined paging and segmentation
    - Protection and sharing

- OS software
    - Fetch policy
    - Placement policy
    - Replacement policy
    - Resident set management
    - Cleaning policy
    - Load control

- UNIX and Solaris memory management
    - Paging system
    - Kernel memory allocator

- Linux memory management
    - Linux virtual memory
    - Kernel memory allocation

- Windows memory management
    - Windows virtual address map
    - Windows paging
    - Windows 8 swapping

- Android memory management