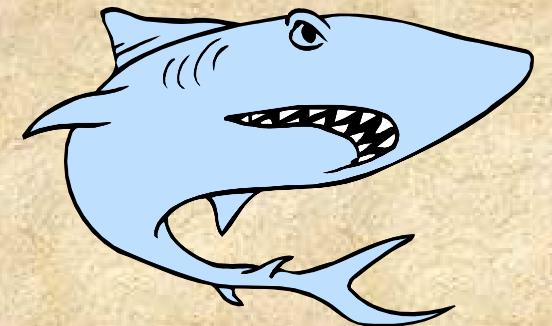
Operating Systems: Internals and Design Principles

Chapter 8
Virtual Memory

Seventh Edition
William Stallings

Operating Systems: Internals and Design Principles

You're gonna need a bigger boat.



— Steven Spielberg,

JAWS, 1975

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

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.

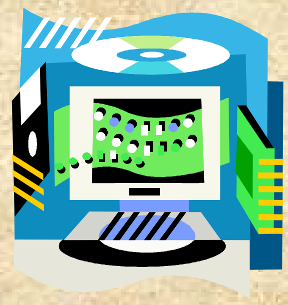
Execution of a Process

- 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

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 in the Ready state



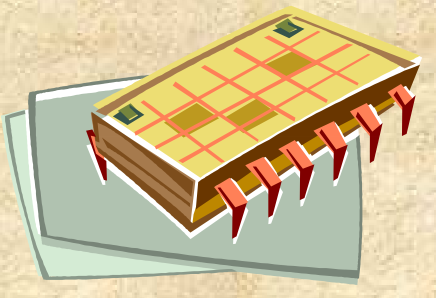
Implications

- 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

Table 8.2

Characteristics of Paging and Segmentation

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	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)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	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 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 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 free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment 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

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

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

Paging Behavior

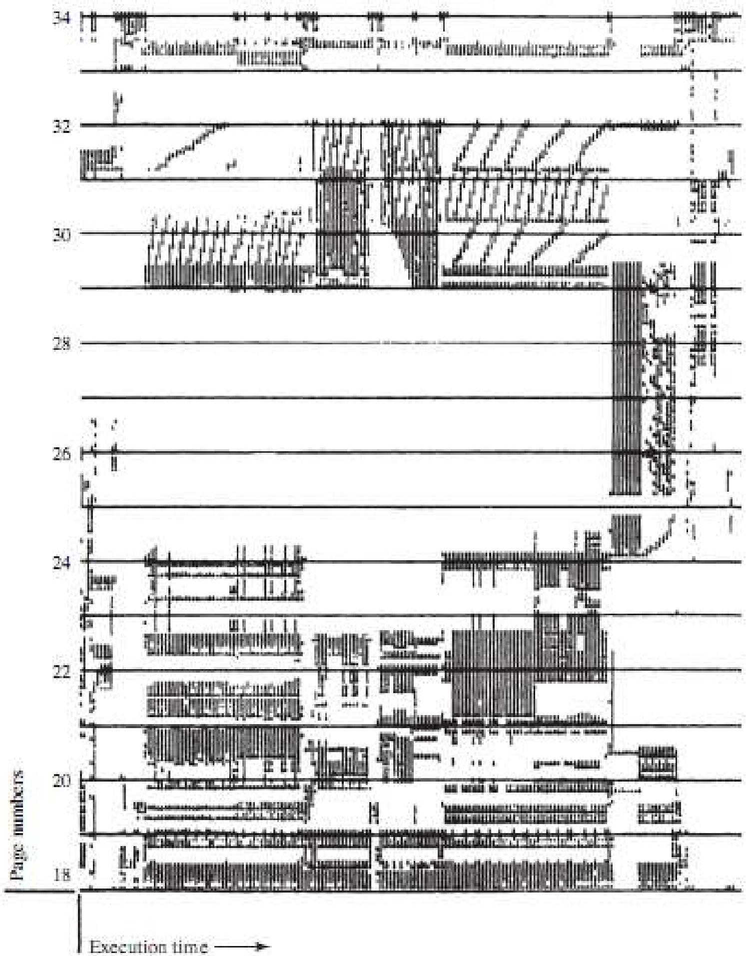


Figure 8.1 Paging Behavior

- During the lifetime of the process, references are confined to a subset of pages

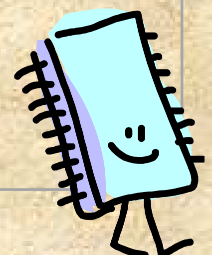
Support Needed for Virtual Memory

For virtual memory to be practical and effective:

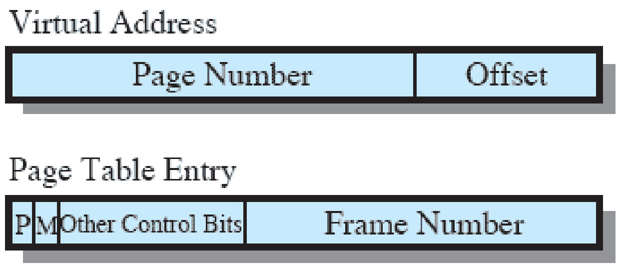
- hardware must support paging and segmentation
- operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

Paging

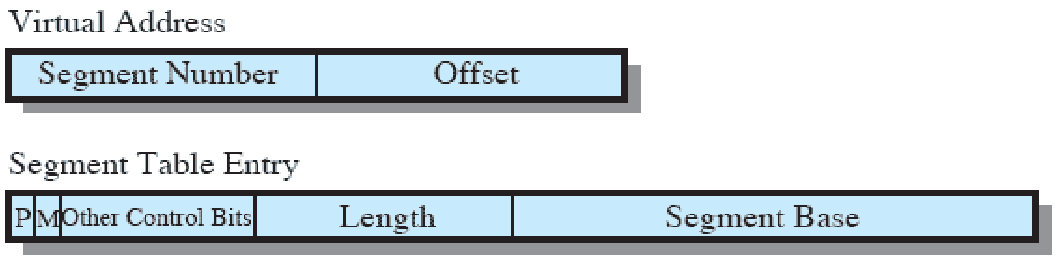
- The term *virtual memory* is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the Atlas computer
- Each process has its own page table
  - each page table entry contains the frame number of the corresponding page in main memory



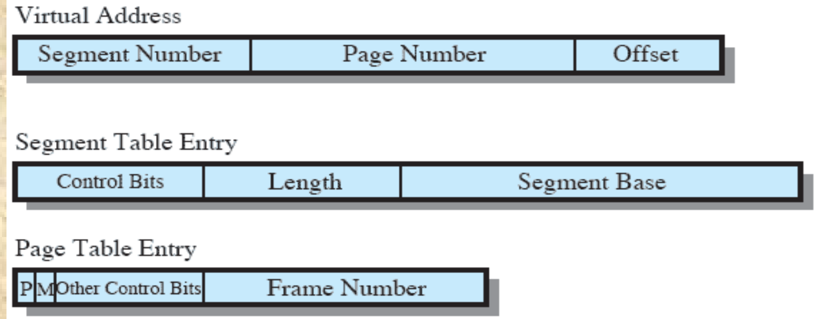
Memory Management Formats



(a) Paging only



(b) Segmentation only



P= present bit
M = Modified bit

(c) Combined segmentation and paging

Address Translation

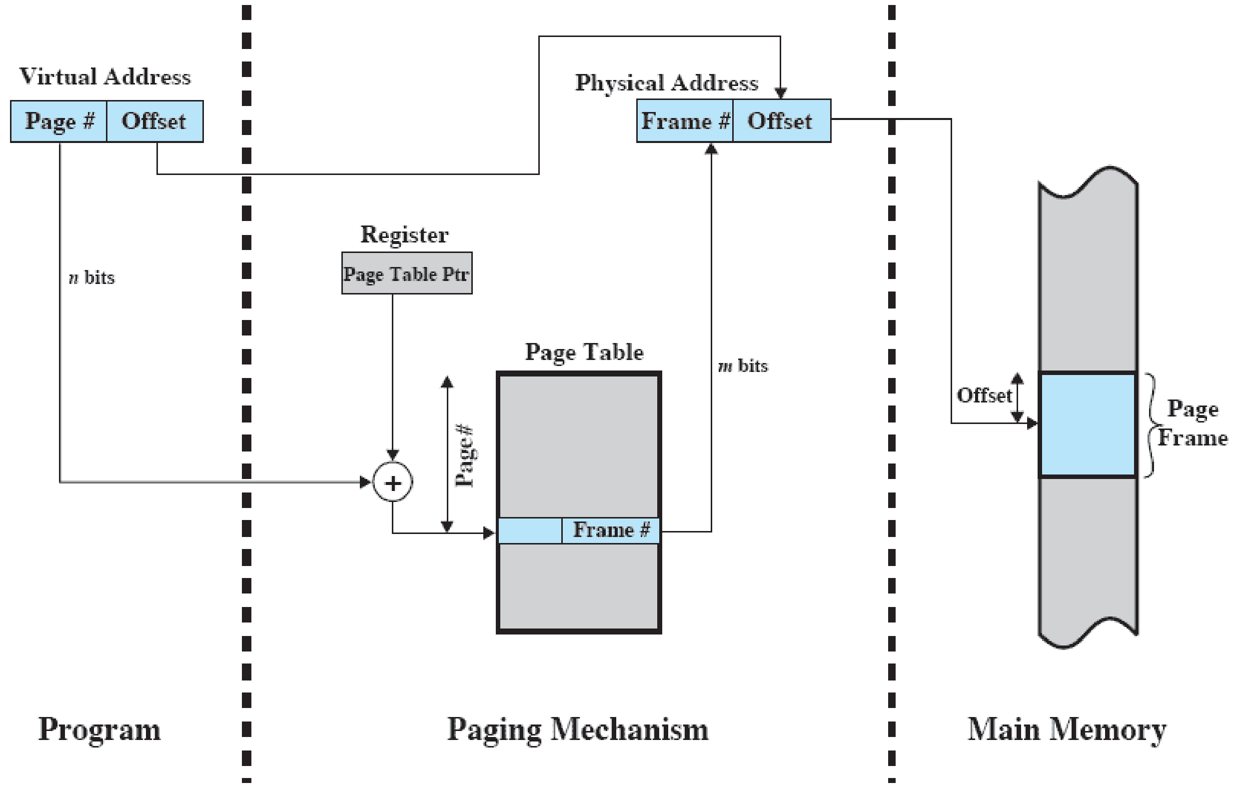


Figure 8.3 Address Translation in a Paging System

Two-Level Hierarchical Page Table

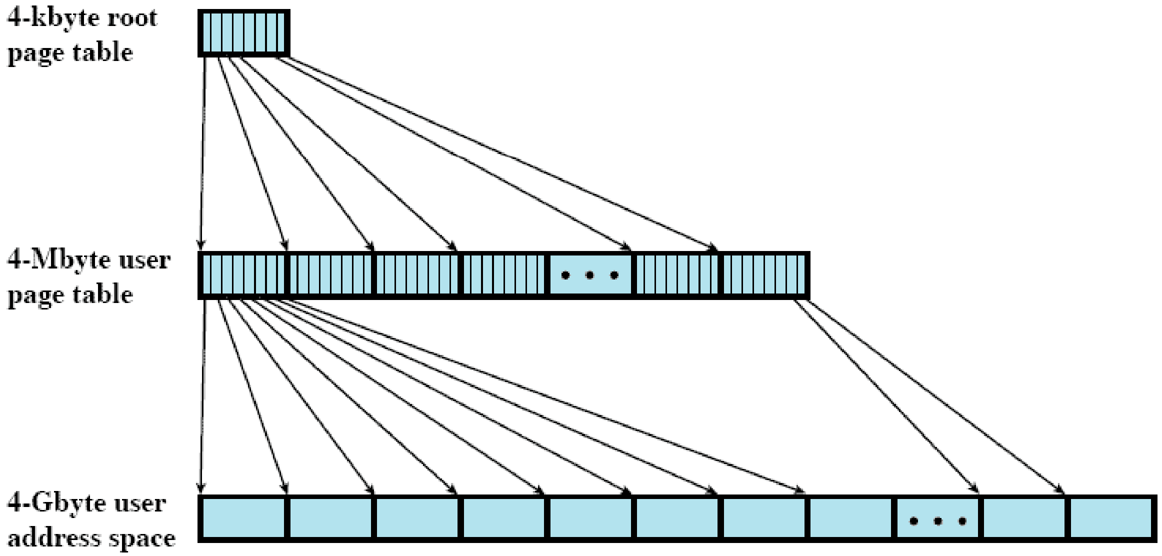


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation

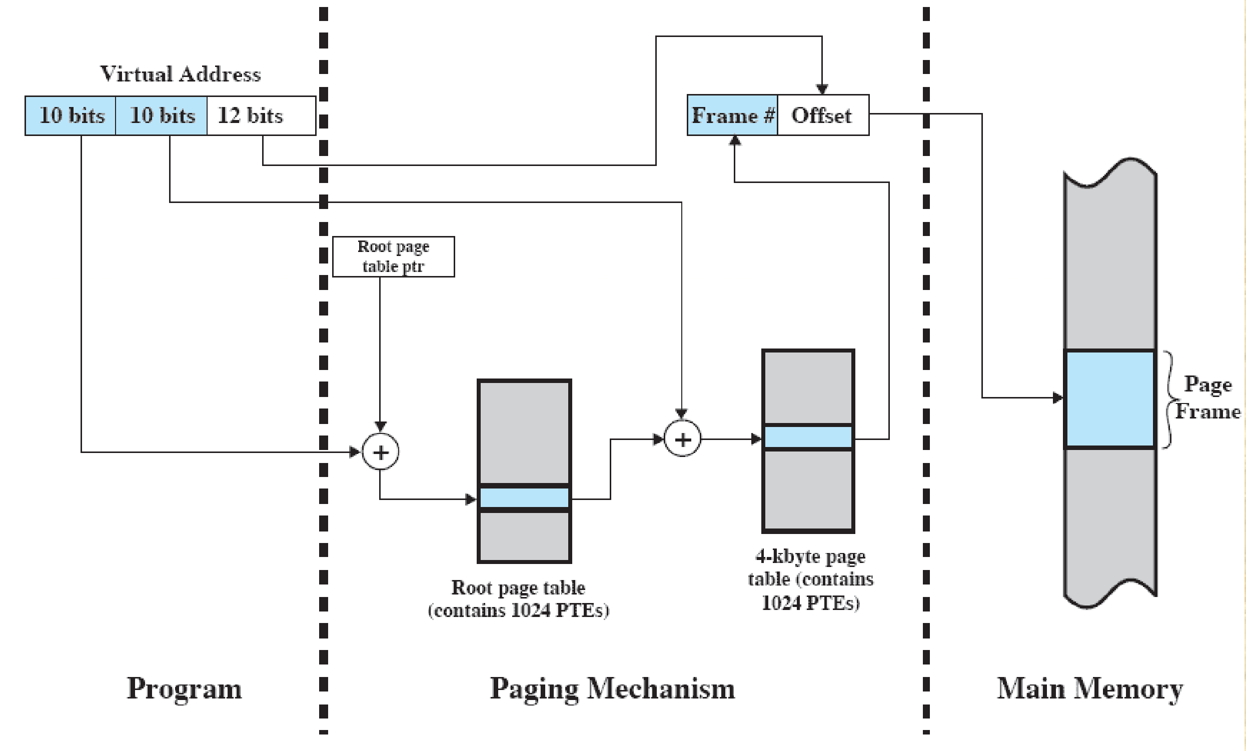
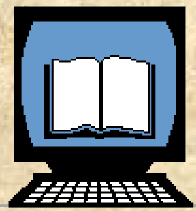


Figure 8.5 Address Translation in a Two-Level Paging System

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



Inverted Page Table

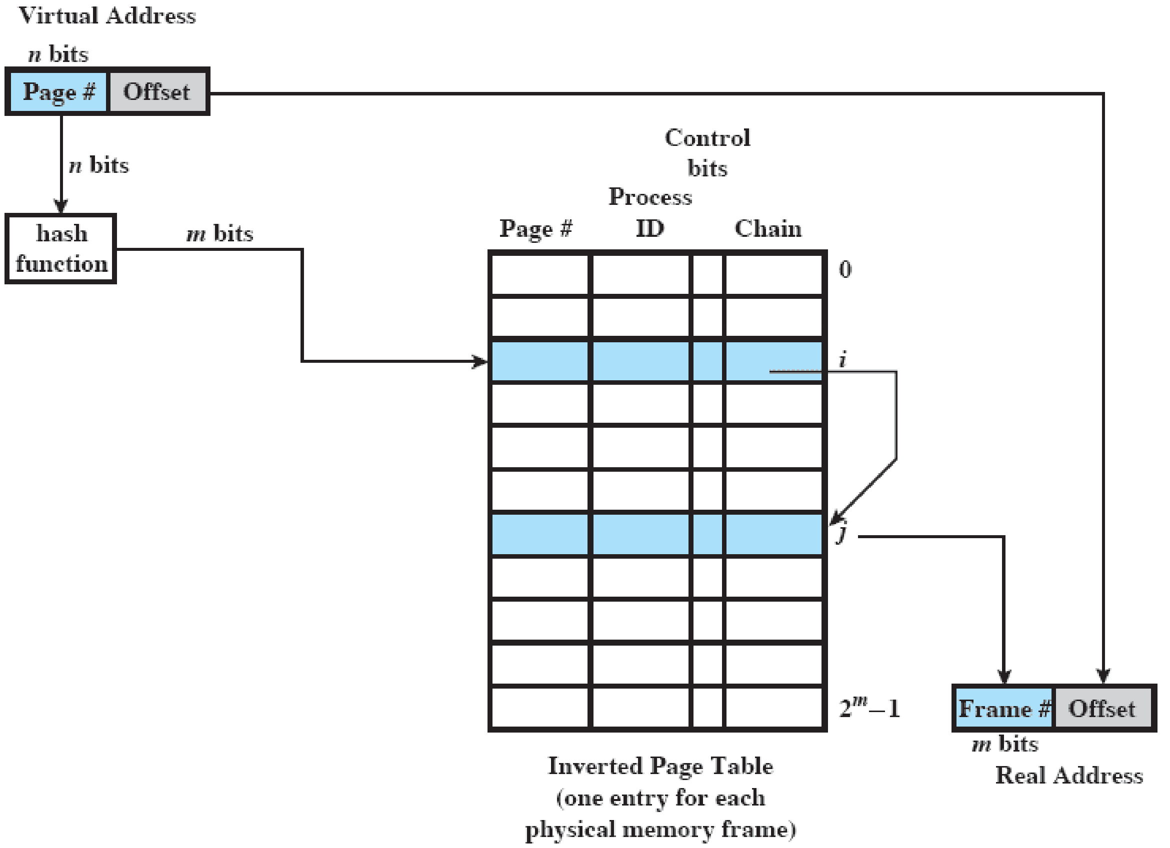


Figure 8.6 Inverted Page Table Structure

Inverted Page Table

Each entry in the page table includes:

Page number

Process identifier
- the process that owns this page

Control bits
- includes flags and protection and locking information

Chain pointer
- the index value of the next entry in the chain

Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

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***

Use of a TLB

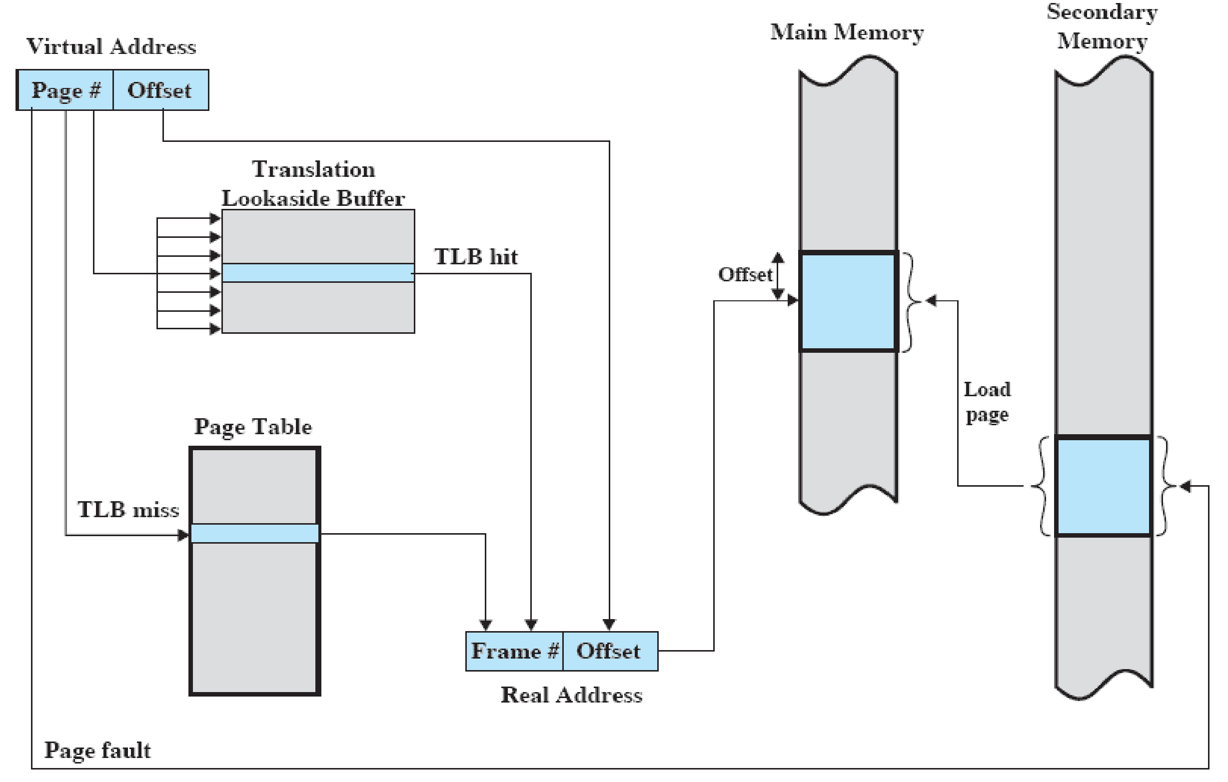


Figure 8.7 Use of a Translation Lookaside Buffer

TLB Operation

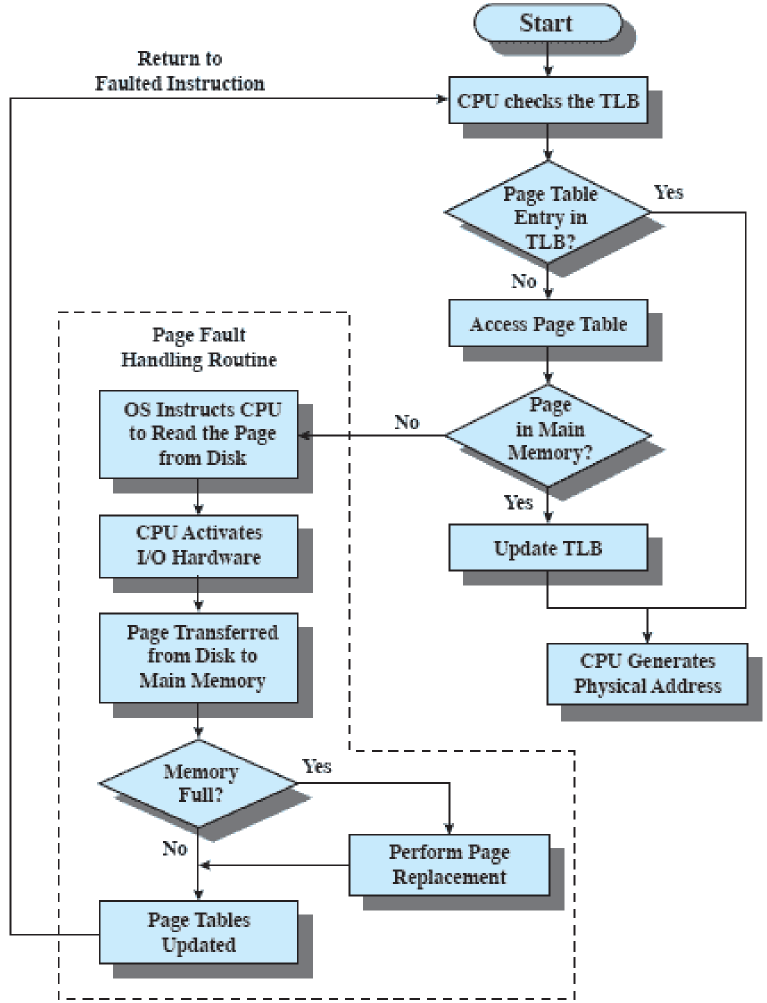
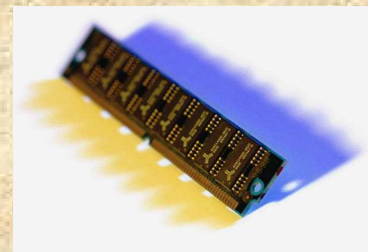


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

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



Direct Versus Associative Lookup

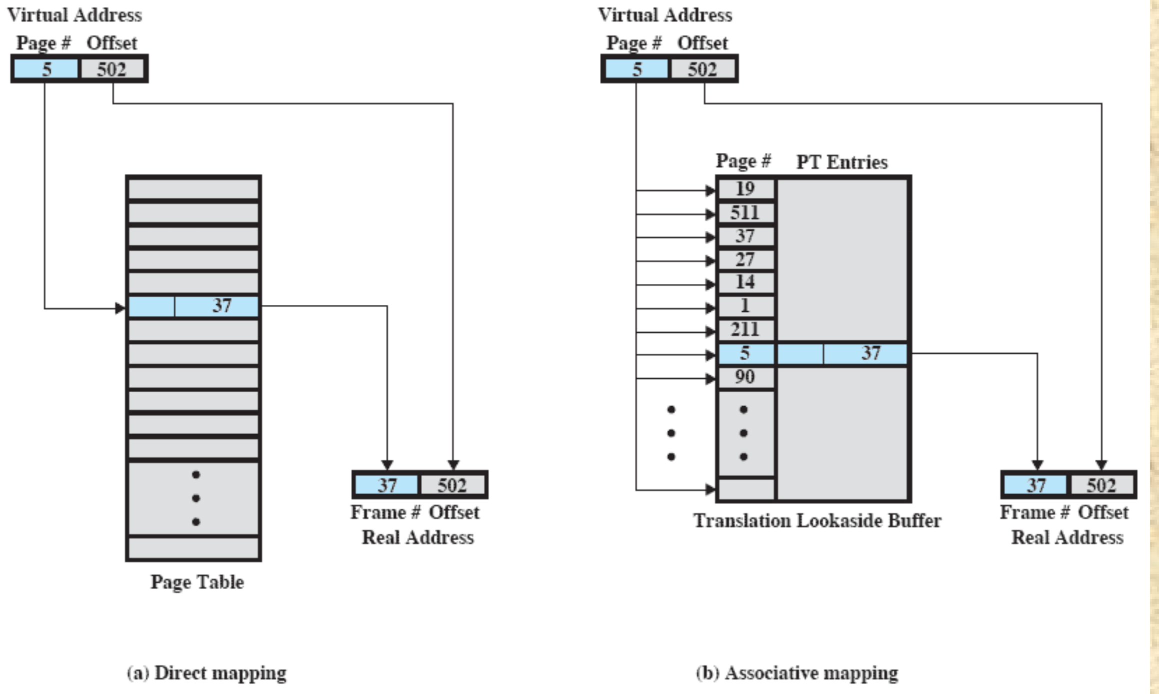


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

TLB and Cache Operation

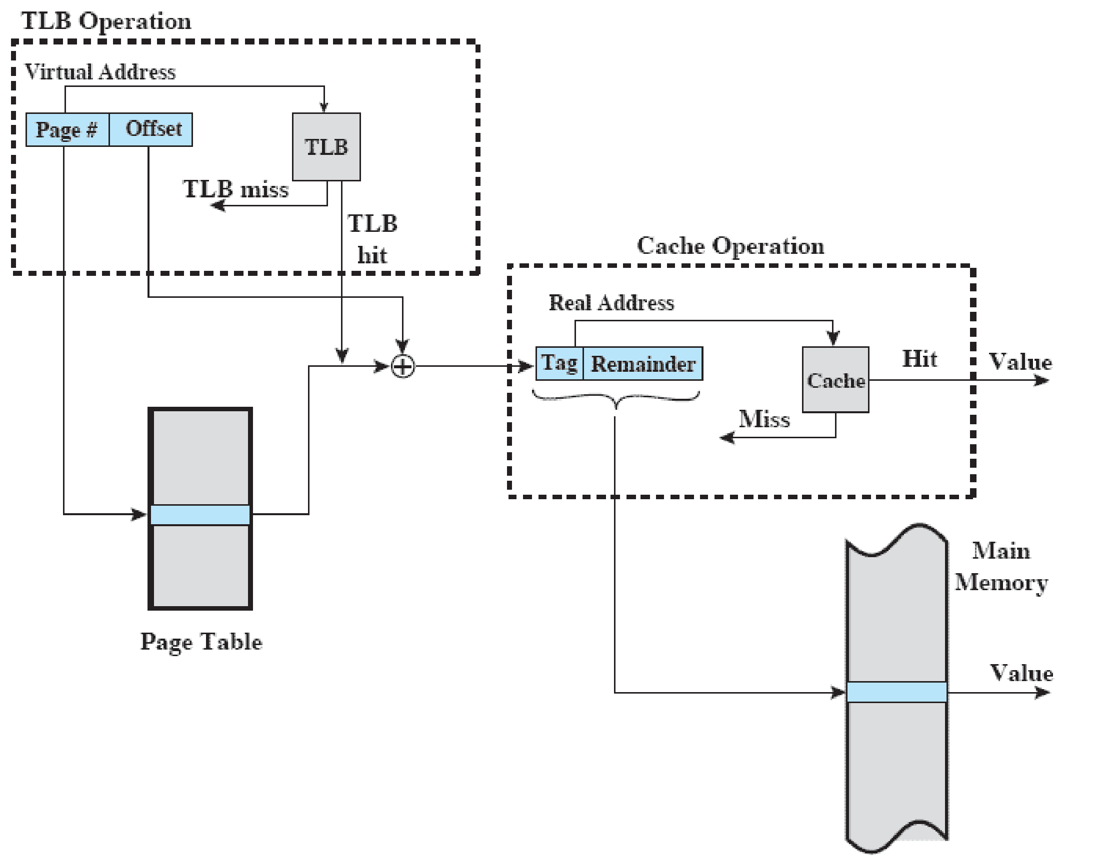


Figure 8.10 Translation Lookaside Buffer and Cache Operation

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

Paging Behavior of a Program

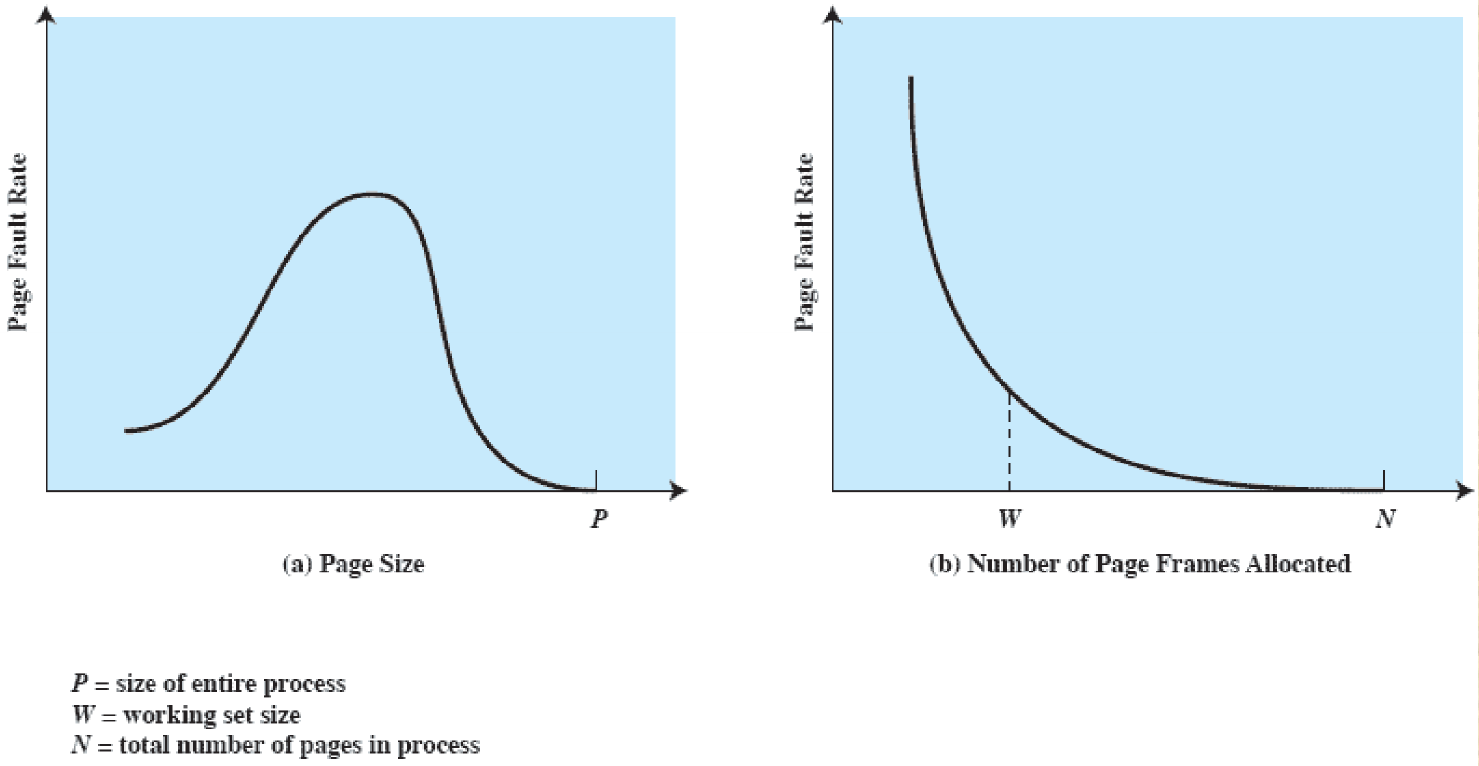


Figure 8.11 Typical Paging Behavior of a Program

Example: Page Sizes

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

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

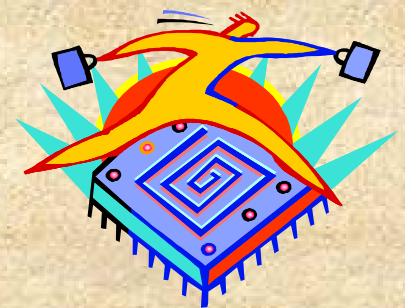
- Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments

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



Address Translation

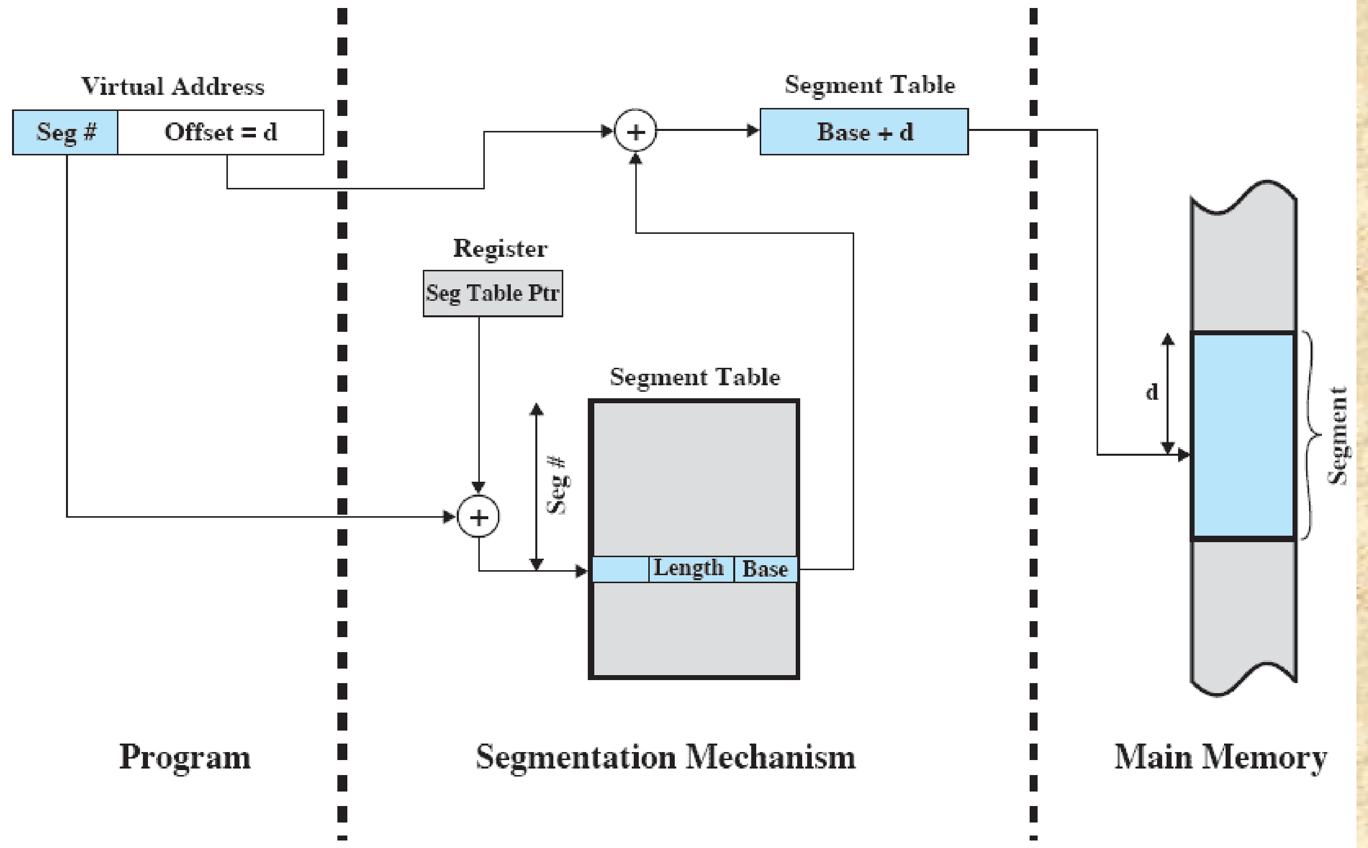


Figure 8.12 Address Translation in a Segmentation System

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

Address Translation

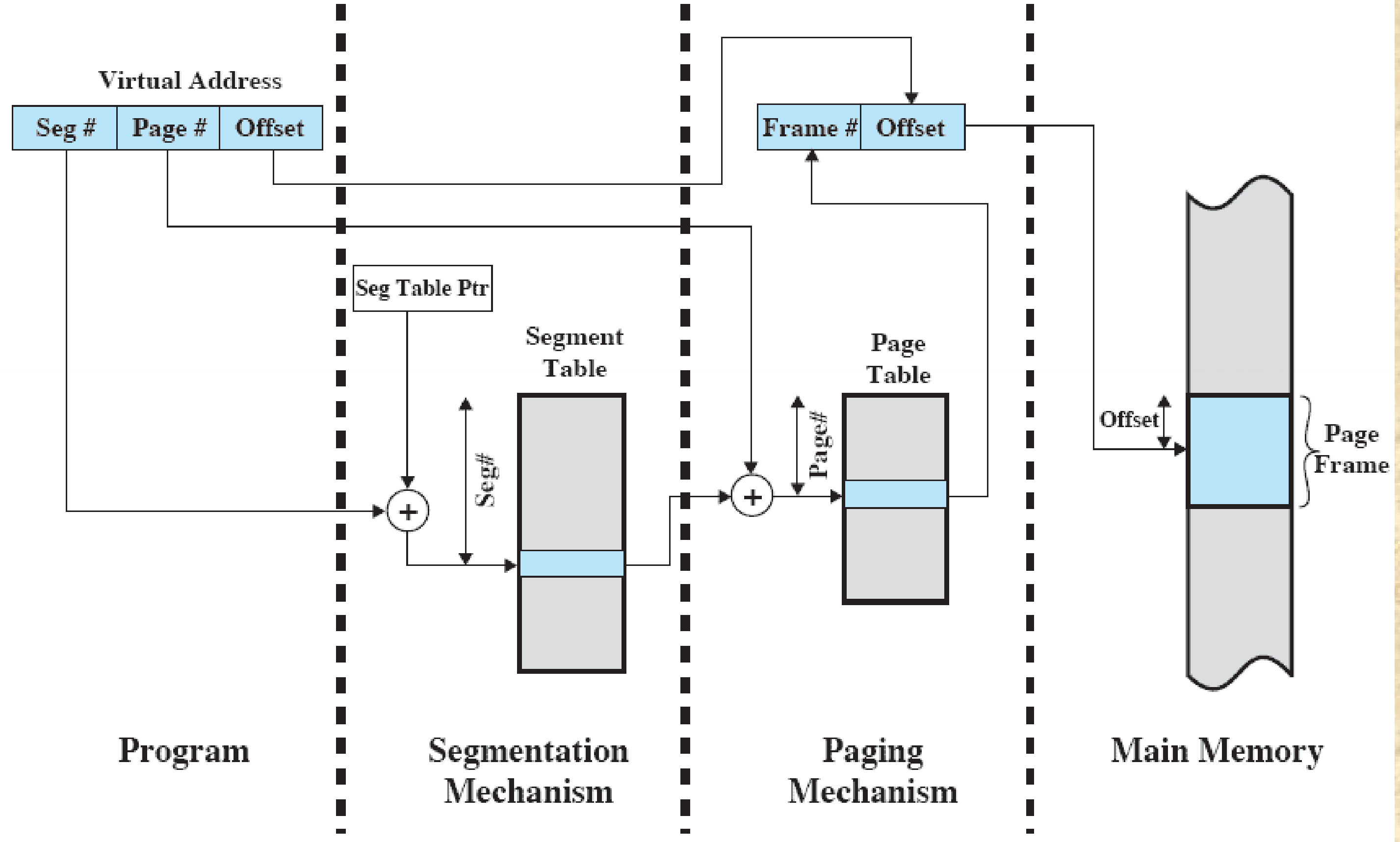
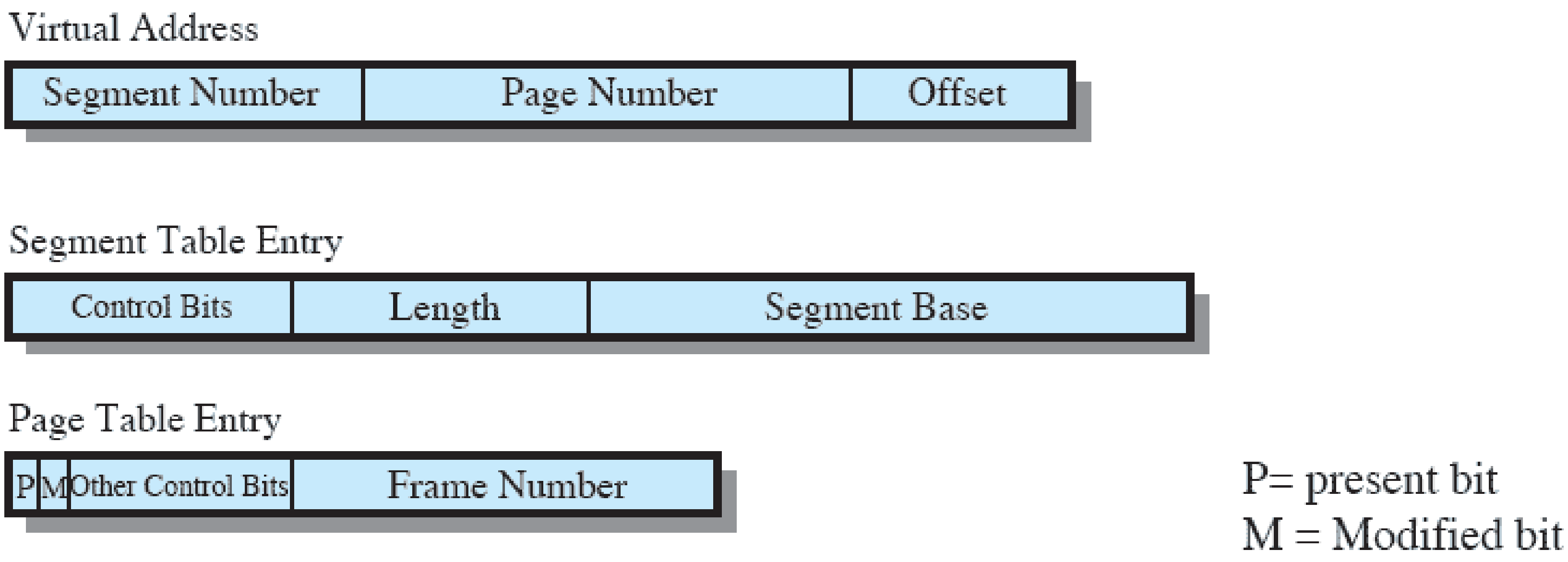


Figure 8.13 Address Translation in a Segmentation/Paging System

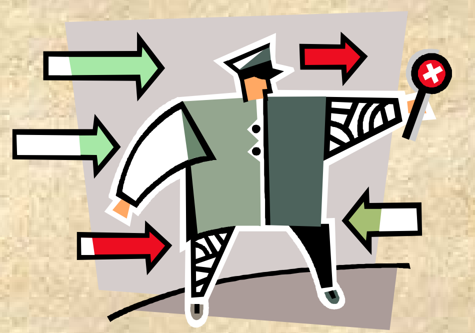
Combined Segmentation and Paging



(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



Protection Relationships

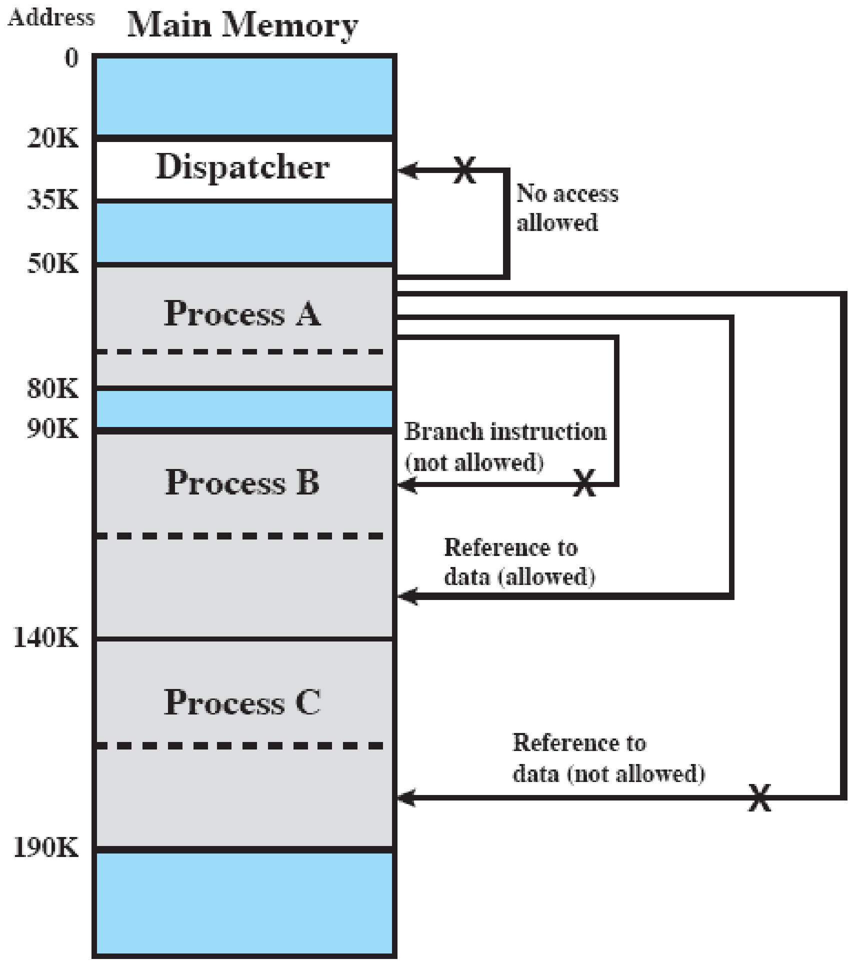
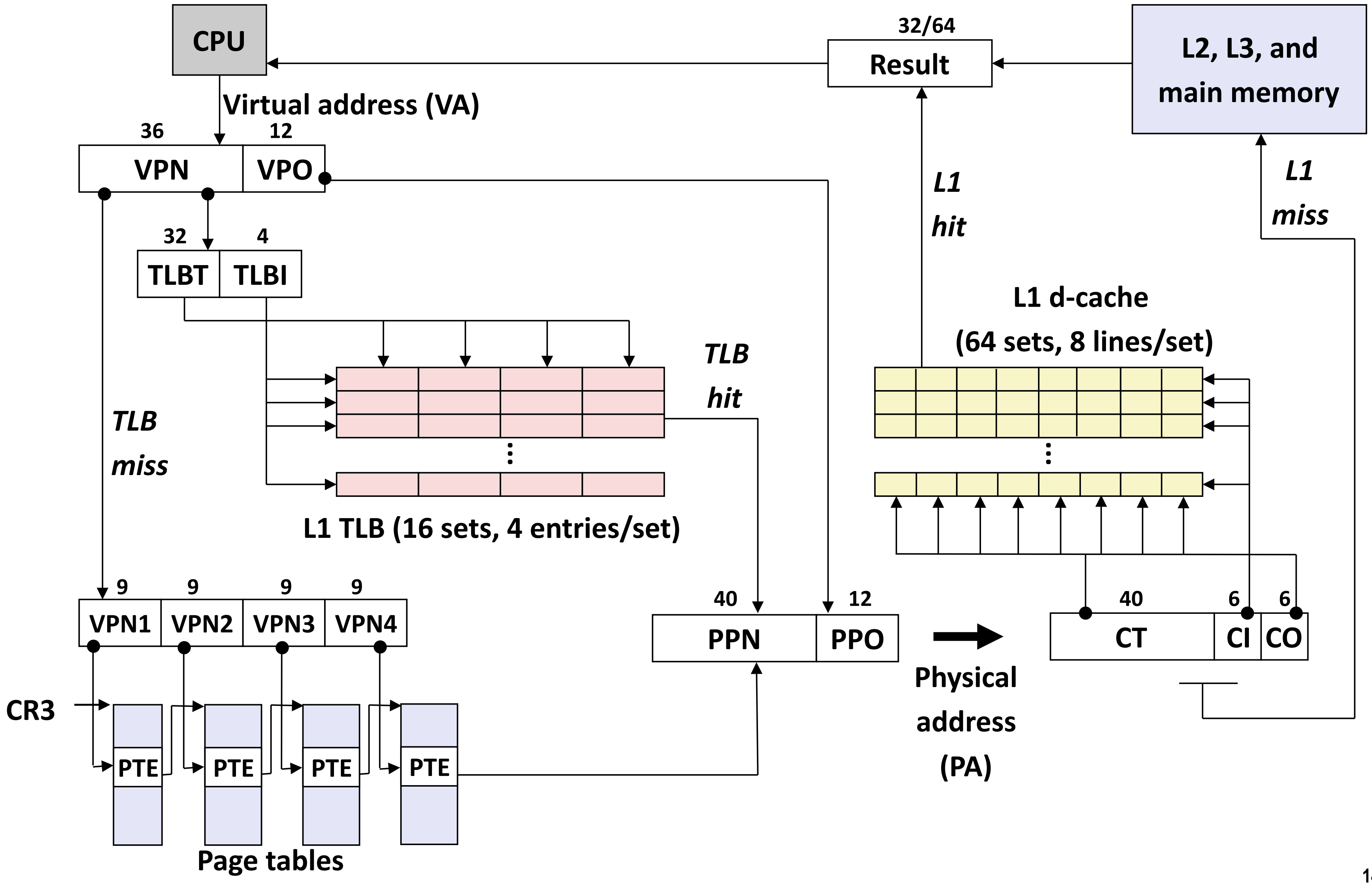


Figure 8.14 Protection Relationships Between Segments

End-to-end Core i7 Address Translation



Core i7 Level 1-3 Page Table Entries

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page table physical base address	Unused	G	PS		A	CD	WT	U/S	R/W	P=1

Available for OS (page table location on disk)	P=0

Each entry references a 4K child page table

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

CD: Caching disabled or enabled for the child page table.

A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

G: Global page (don't evict from TLB on task switch)

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

Core i7 Level 4 Page Table Entries

63	62 … 52	51 … 12	11 … 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address	Unused	G		D	A	CD	WT	U/S	R/W	P=1

Available for OS (page location on disk)	P=0

Each entry references a 4K child page

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for child page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

CD: Cache disabled (1) or enabled (0)

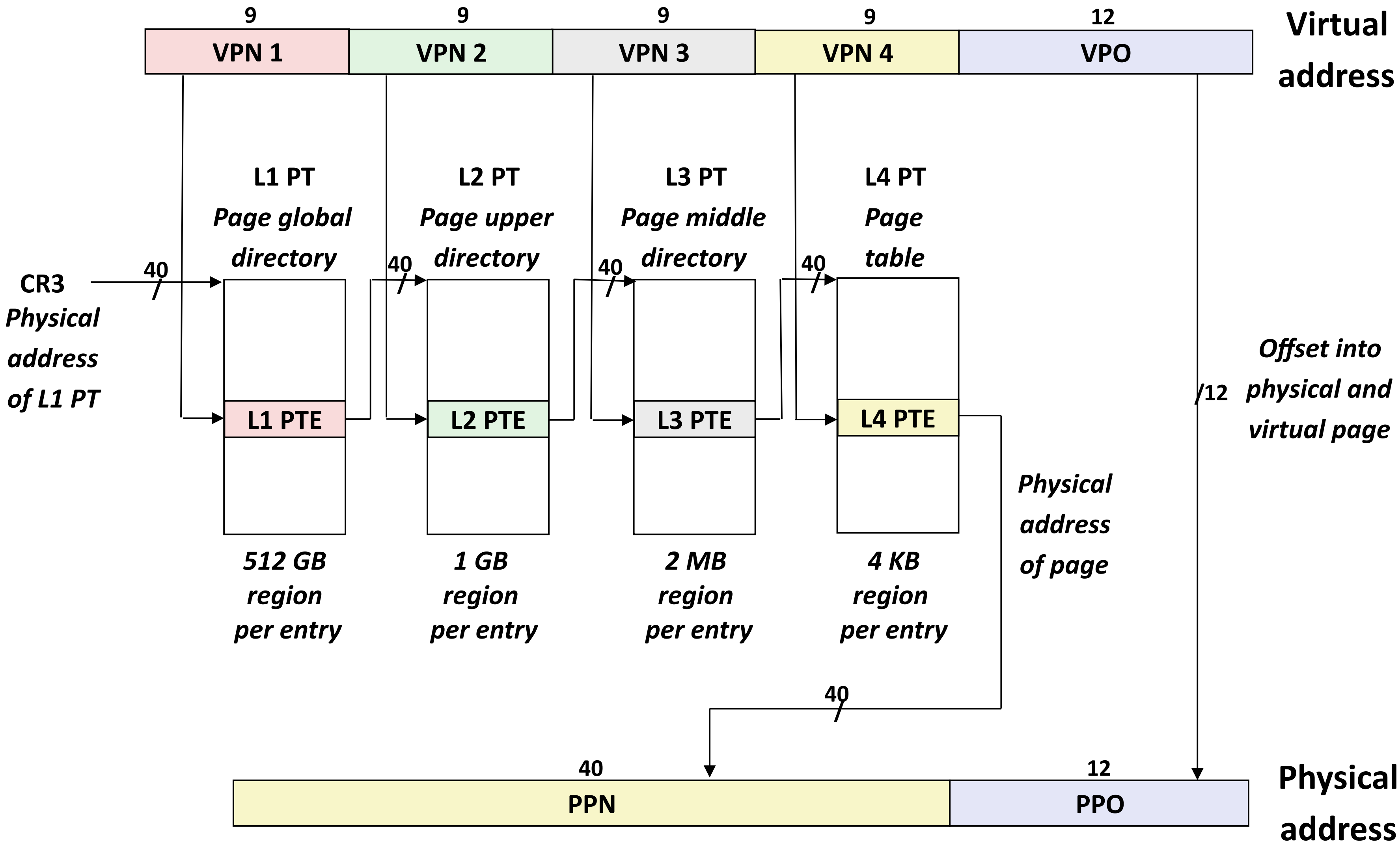
A: Reference bit (set by MMU on reads and writes, cleared by software)

D: Dirty bit (set by MMU on writes, cleared by software)

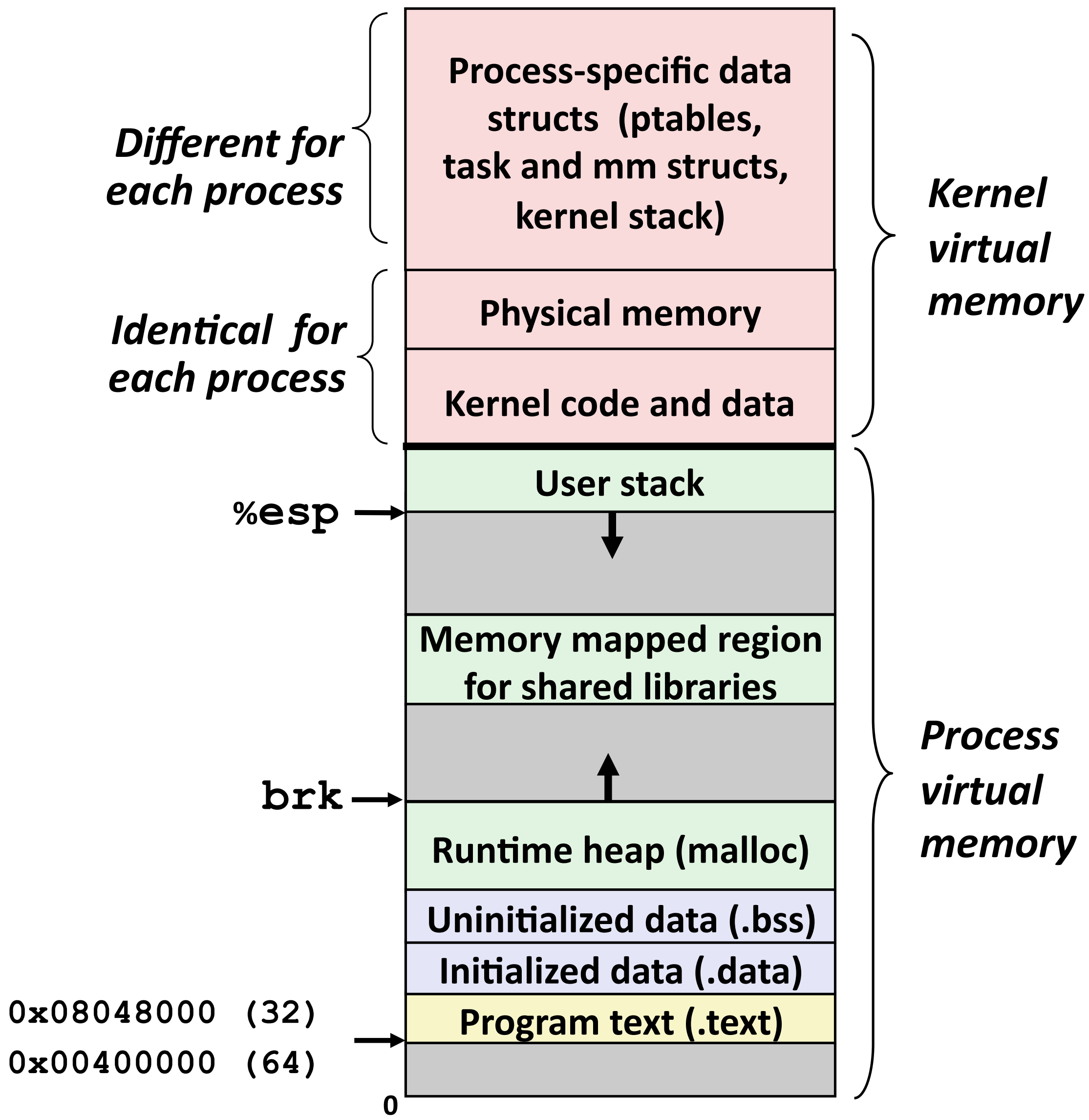
G: Global page (don't evict from TLB on task switch)

Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

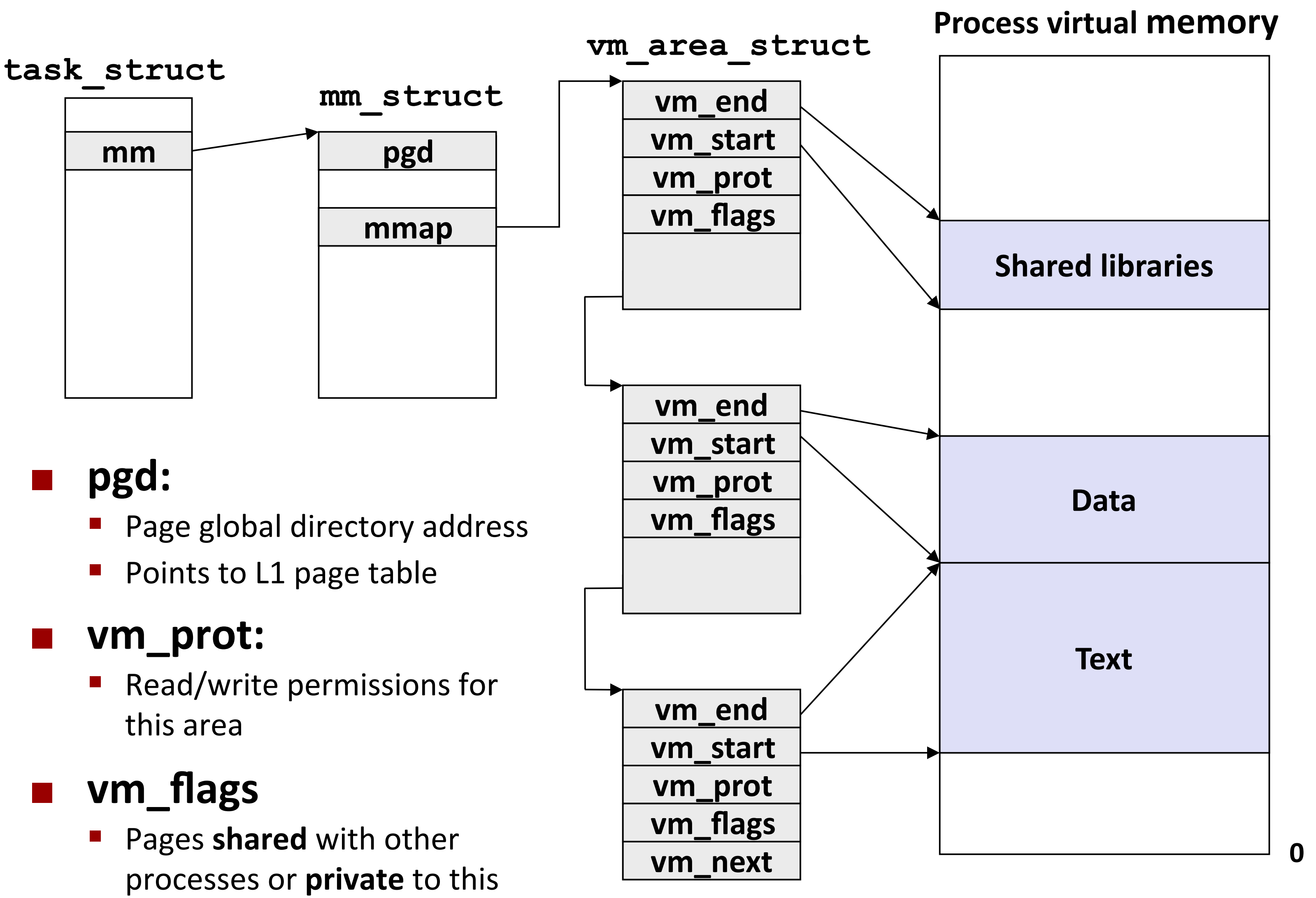
Core i7 Page Table Translation



Virtual Memory of a Linux Process

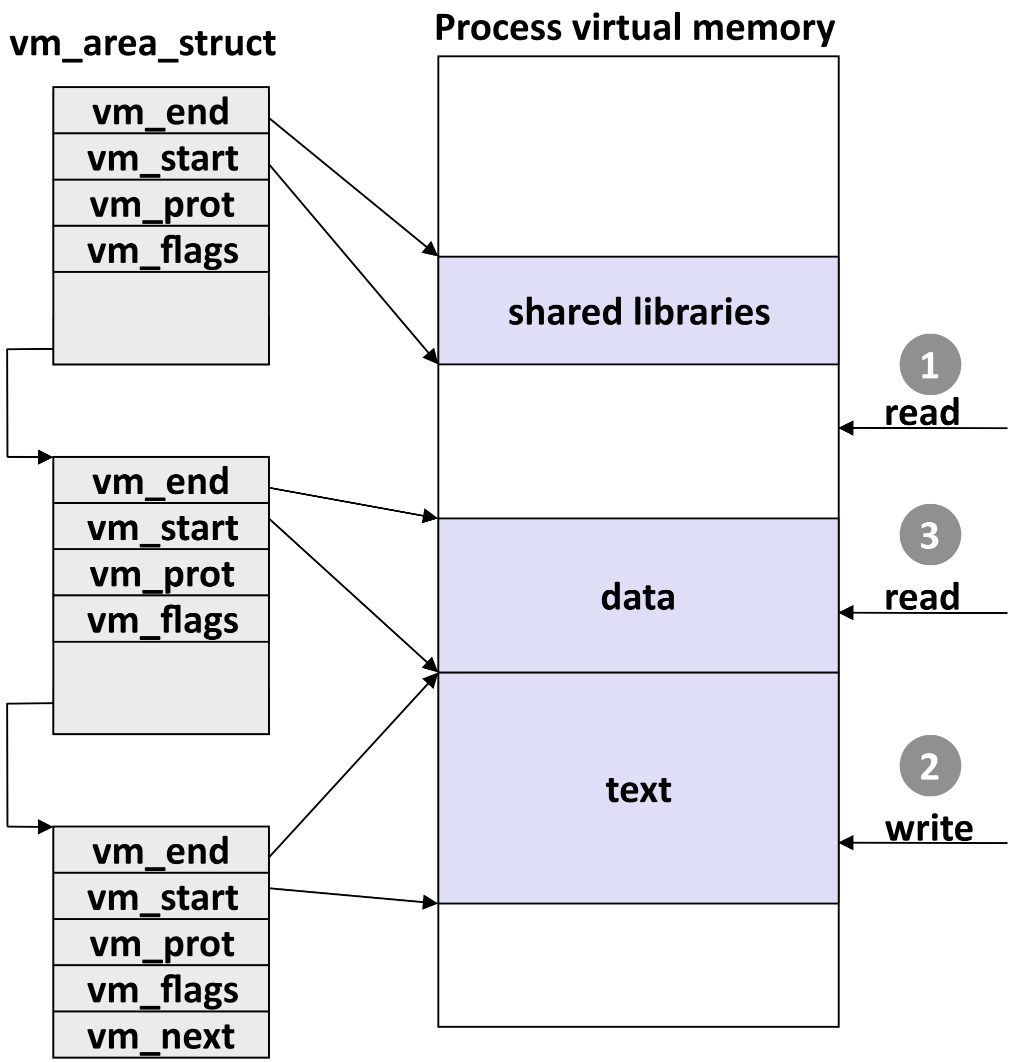


Linux Organizes VM as Collection of "Areas"



- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Linux Page Fault Handling



Segmentation fault:
accessing a non-existing page

Normal page fault

Protection exception:
e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

Memory Mapping

- **VM areas initialized by associating them with disk objects.**
  - Process is known as ***memory mapping***.

- **Area can be backed by (i.e., get its initial values from) :**
  - ***Regular file*** on disk (e.g., an executable object file)
    - Initial page bytes come from a section of a file
  - ***Anonymous file*** (e.g., nothing)
    - First fault will allocate a physical page full of 0's (***demand-zero page***)
    - Once the page is written to (***dirtied***), it is like any other page

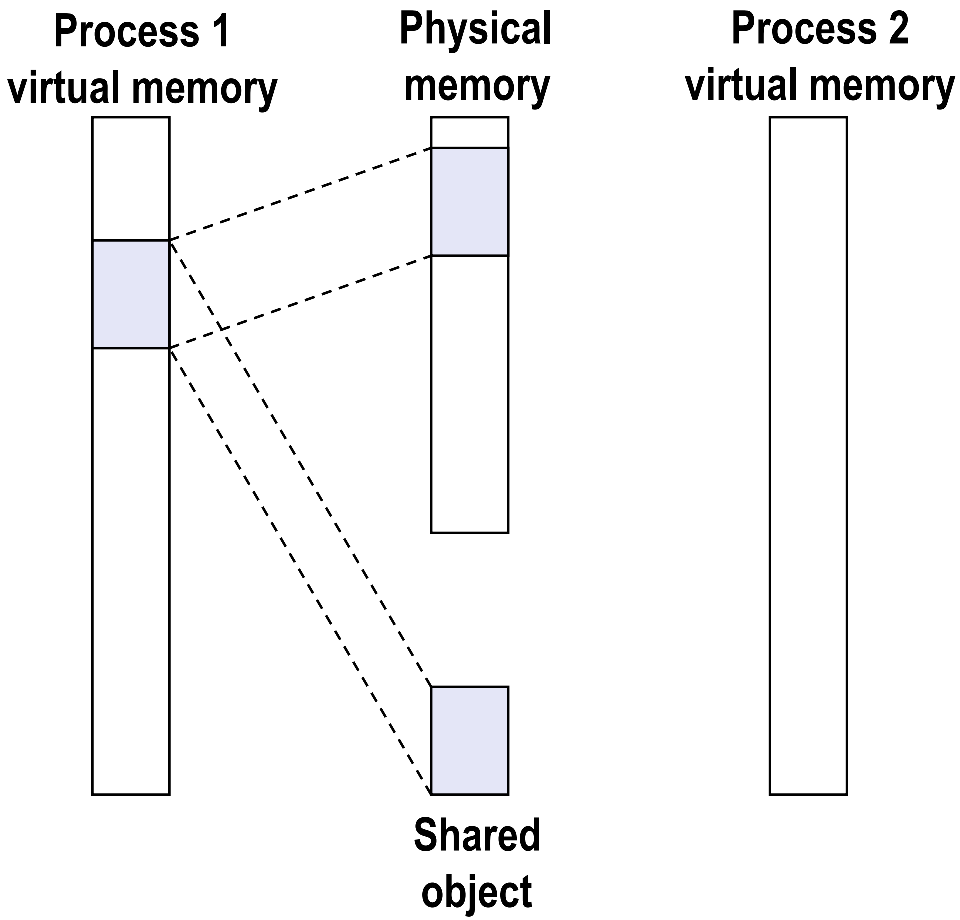
- **Dirty pages are copied back and forth between memory and a special *swap file*.**

Demand paging

- ***Key point:*** **no virtual pages are copied into physical memory until they are referenced!**
  - Known as ***demand paging***

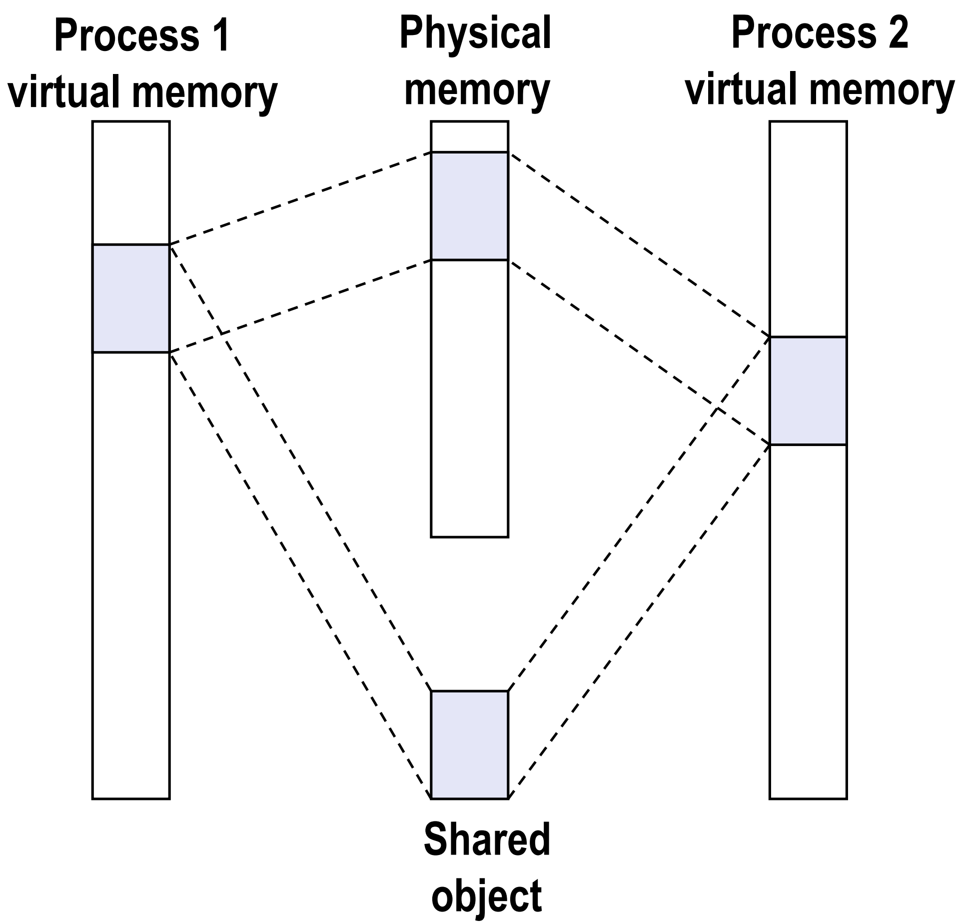
- **Crucial for time and space efficiency**

Sharing Revisited: Shared Objects



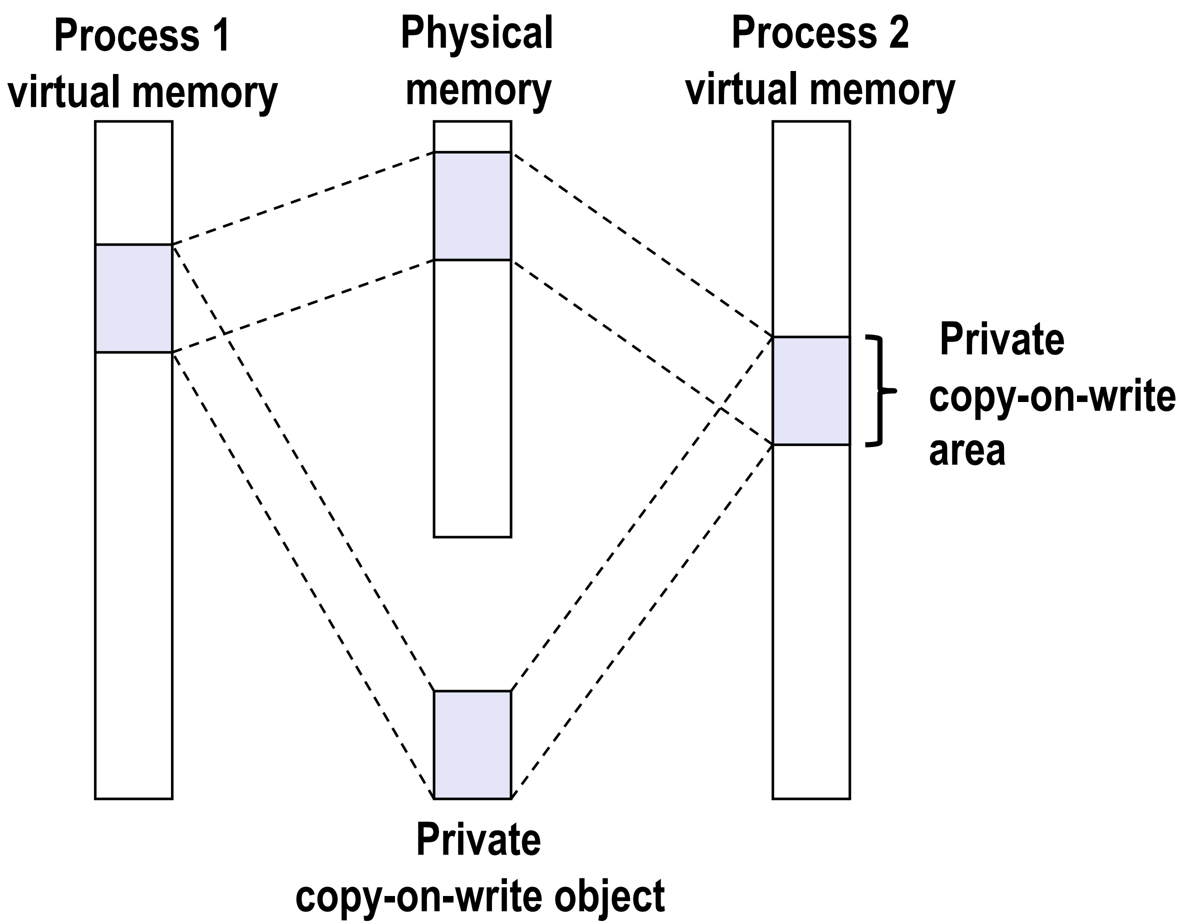
- Process 1 maps the shared object.

Sharing Revisited: Shared Objects



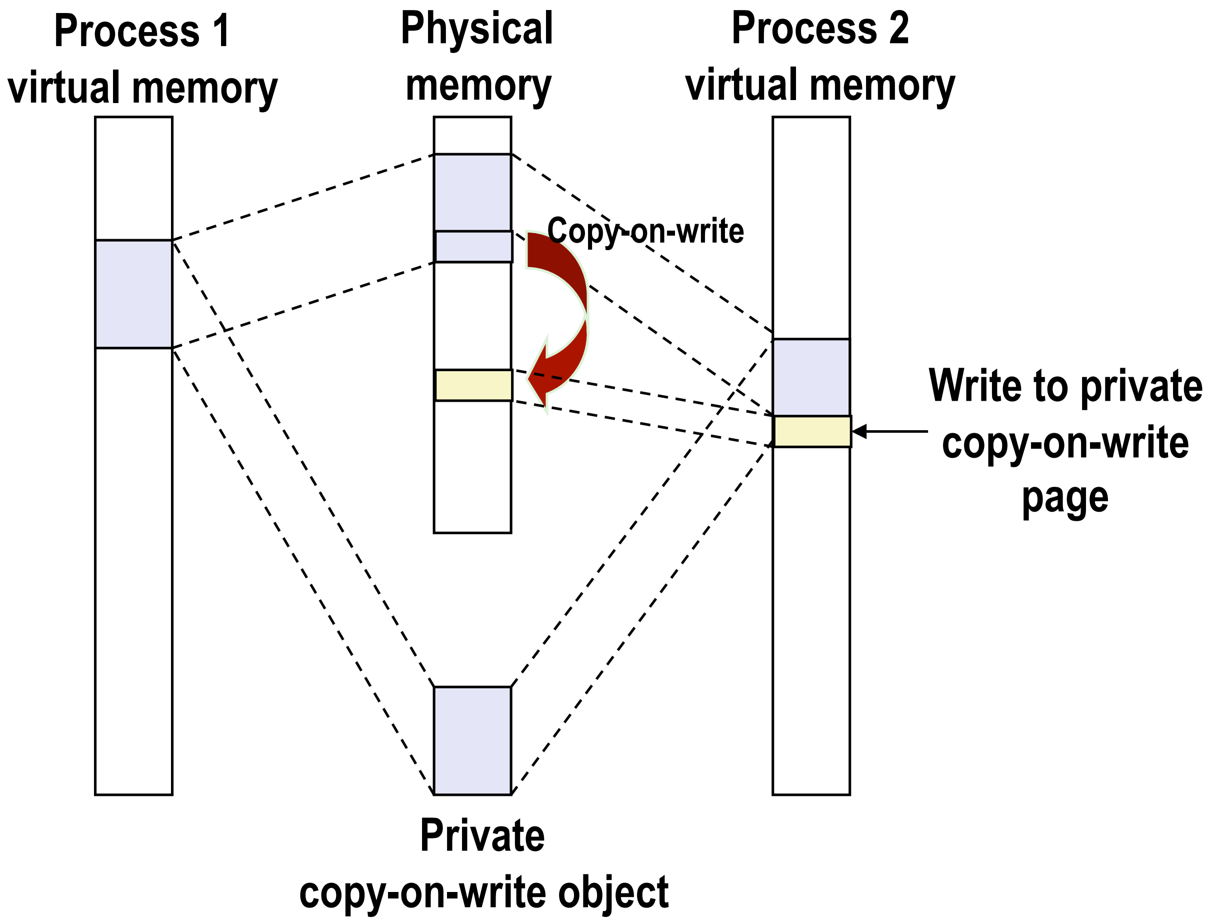
- Process 2 maps the shared object.
- Notice how the virtual addresses can be different.

Sharing Revisited: Private Copy-on-write (COW) Objects



- Two processes mapping a ***private copy-on-write (COW)*** object.
- Area flagged as private copy-on-write
- PTEs in private areas are flagged as read-only

Sharing Revisited: Private Copy-on-write (COW) Objects



- Instruction writing to private page triggers protection fault.
- Handler creates new R/W page.
- Instruction restarts upon handler return.
- Copying deferred as long as possible!

The `fork` Function Revisited

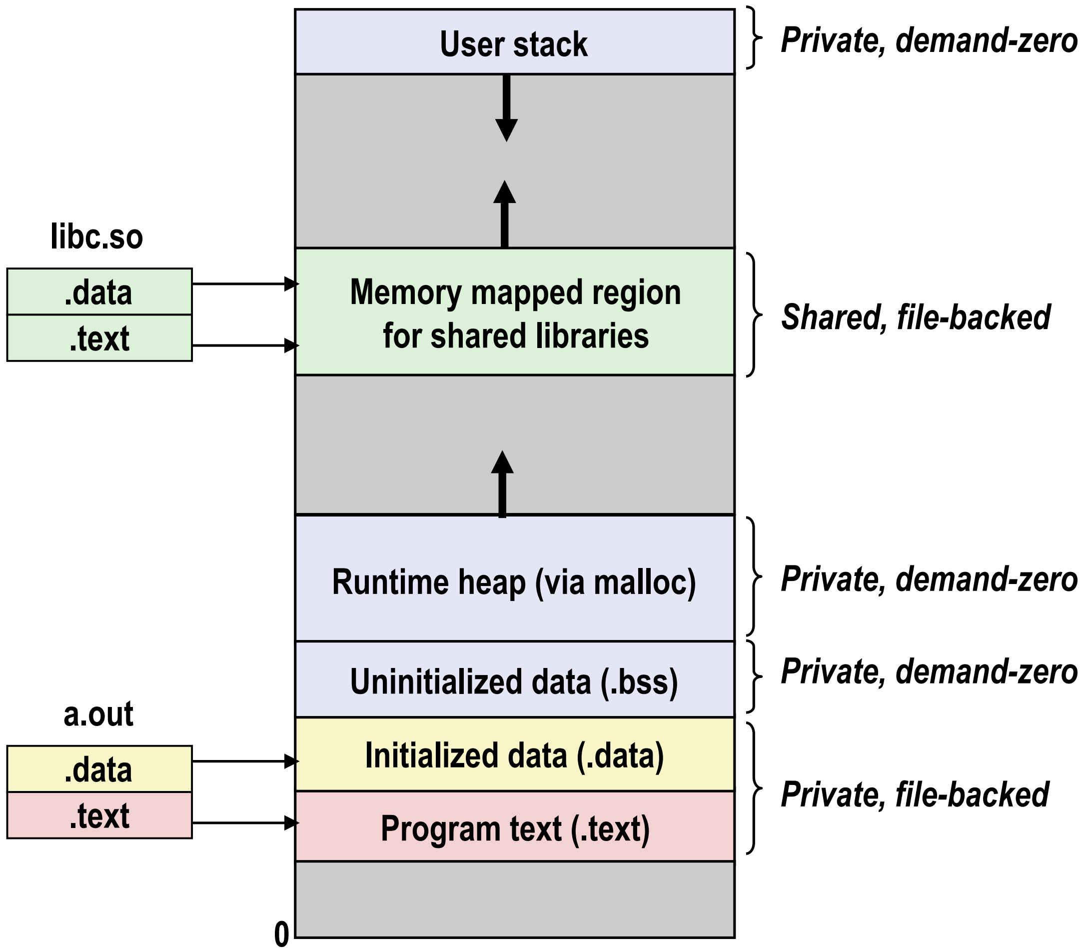
- **VM and memory mapping explain how `fork` provides private address space for each process.**

- **To create virtual address for new new process**
  - Create exact copies of current `mm_struct`, `vm_area_struct`, and page tables.
  - Flag each page in both processes as read-only
  - Flag each `vm_area_struct` in both processes as private COW

- **On return, each process has exact copy of virtual memory**

- **Subsequent writes create new pages using COW mechanism.**

The `execve` Function Revisited



- **To load and run a new program `a.out` in the current process using `execve`:**
- **Free `vm_area_struct`'s and page tables for old areas**
- **Create `vm_area_struct`'s and page tables for new areas**
  - Programs and initialized data backed by object files.
  - `.bss` and stack backed by anonymous files .
- **Set PC to entry point in `.text`**
  - Linux will fault in code and data pages as needed.

User-Level Memory Mapping

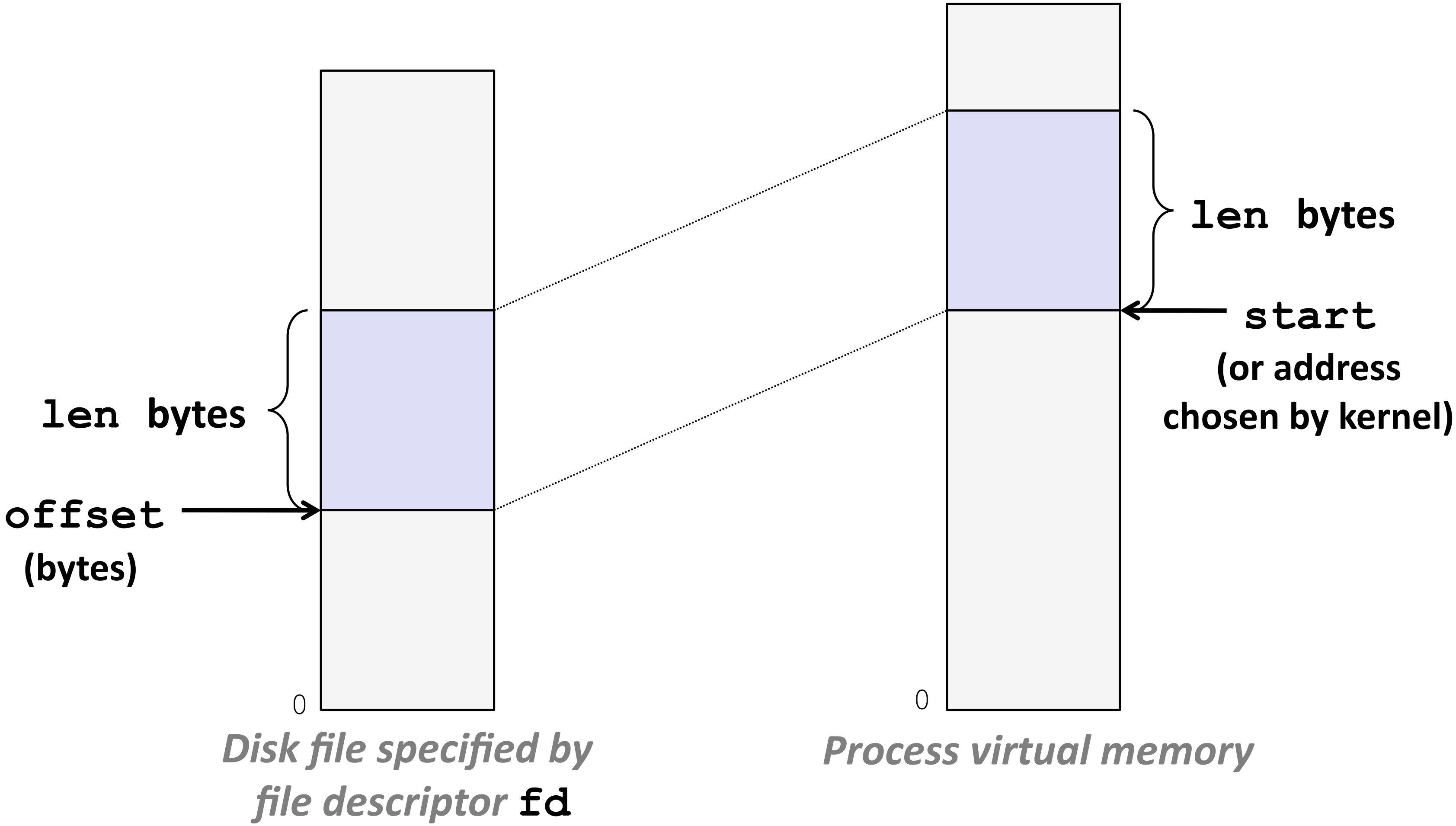
```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

- **Map `len` bytes starting at offset `offset` of the file specified by file description `fd`, preferably at address `start`**
  - `start`: may be 0 for "pick an address"
  - `prot`: PROT_READ, PROT_WRITE, ...
  - `flags`: MAP_ANON, MAP_PRIVATE, MAP_SHARED, ...

- **Return a pointer to start of mapped area (may not be `start`)**

User-Level Memory Mapping

```
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```



Using mmap to Copy Files

- **Copying without transferring data to user space .**

```
#include "csapp.h"

/*
 * mmapcopy - uses mmap to copy
 *            file fd to stdout
 */
void mmapcopy(int fd, int size)
{

    /* Ptr to mem-mapped VM area */
    char *bufp;

    bufp = Mmap(NULL, size,
                PROT_READ,
                MAP_PRIVATE, fd, 0);
    Write(1, bufp, size);
    return;
}
```

```
/* mmapcopy driver */
int main(int argc, char **argv)
{
    struct stat stat;
    int fd;

    /* Check for required cmdline arg */
    if (argc != 2) {
        printf("usage: %s <filename>\n”,
                argv[0]);
        exit(0);
    }

    /* Copy the input arg to stdout */
    fd = Open(argv[1], O_RDONLY, 0);
    Fstat(fd, &stat);
    mmapcopy(fd, stat.st_size);
    exit(0);
}
```

Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by **mapping** a disk block to a page in memory
- A file is initially read using demand paging
  - A page-sized portion of the file is read from the file system into a physical page
  - Subsequent reads/writes to/from the file are treated as ordinary memory accesses
- Simplifies and speeds file access by driving file I/O through memory rather than `read()` and `write()` system calls
- Also allows several processes to map the same file allowing the pages in memory to be shared
- But when does written data make it to disk?
  - Periodically and / or at file `close()` time
  - For example, when the pager scans for dirty pages

Memory-Mapped File Technique for all I/O

- Some OSes uses memory mapped files for standard I/O
- Process can explicitly request memory mapping a file via `mmap()` system call
  - Now file mapped into process address space
- For standard I/O (`open(), read(), write(), close()`), mmap anyway
  - But map file into kernel address space
  - Process still does read() and write()
    - Copies data to and from kernel space and user space
  - Uses efficient memory management subsystem
    - Avoids needing separate subsystem
- COW can be used for read/write non-shared pages
- Memory mapped files can be used for shared memory (although again via separate system calls)

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

Policies for Virtual Memory

- Key issue: Performance
    - minimize page faults

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

Fetch Policy

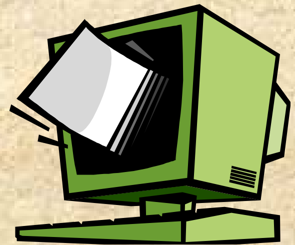
- Determines when a page should be brought into memory

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"

Non-Uniform Memory Access

- So far all memory accessed equally
- Many systems are **NUMA** – speed of access to memory varies
  - Consider system boards containing CPUs and memory, interconnected over a system bus
- Optimal performance comes from allocating memory "close to" the CPU on which the thread is scheduled
  - And modifying the scheduler to schedule the thread on the same system board when possible
  - Solved by Solaris by creating **lgroups**
    - Structure to track CPU / Memory low latency groups
    - Used my schedule and pager
    - When possible schedule all threads of a process and allocate all memory for that process within the lgroup

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

Optimal Policy

- Selects the page for which the time to the next reference is the longest
- Produces three page faults after the frame allocation has been filled

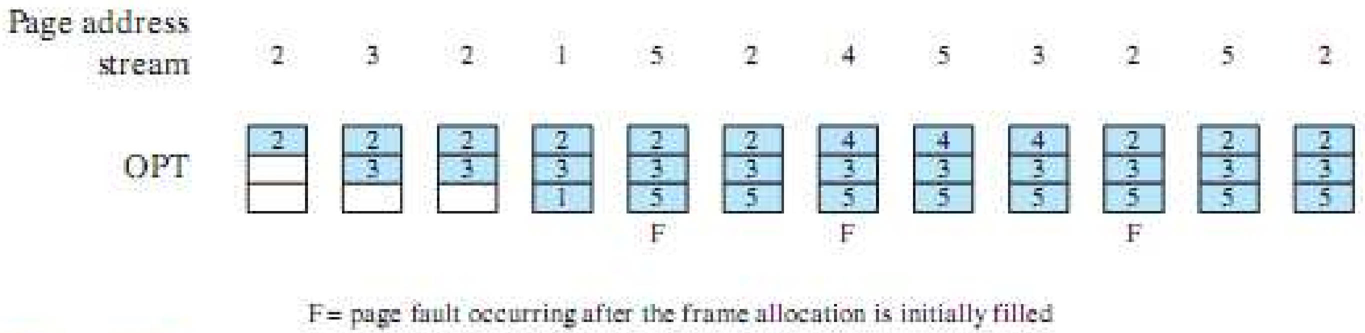


Figure 8.15 **Behavior of Four Page Replacement Algorithms**

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

LRU Example

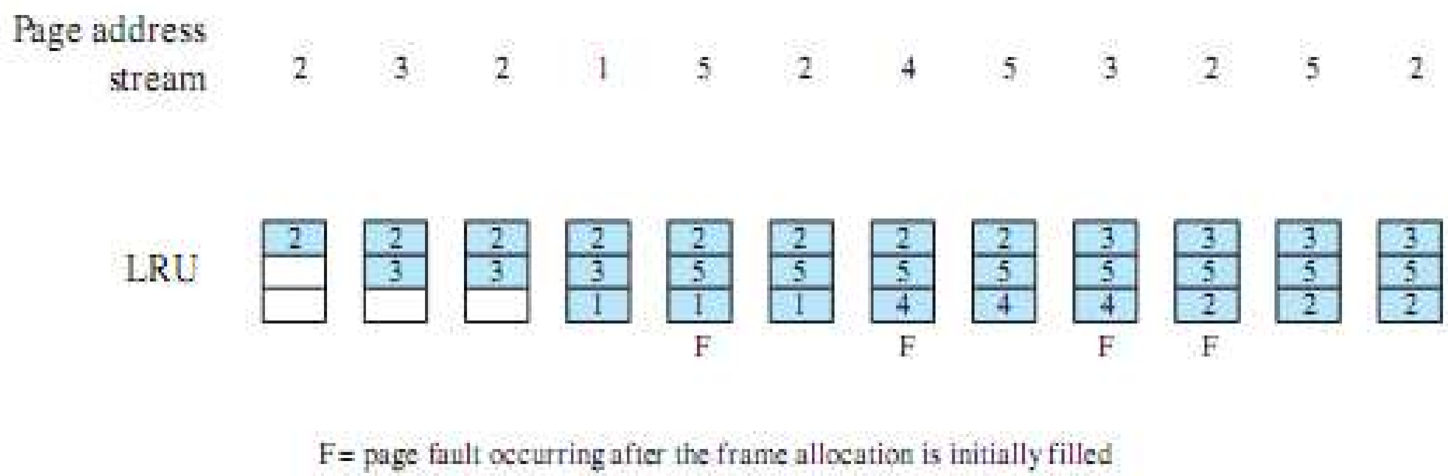
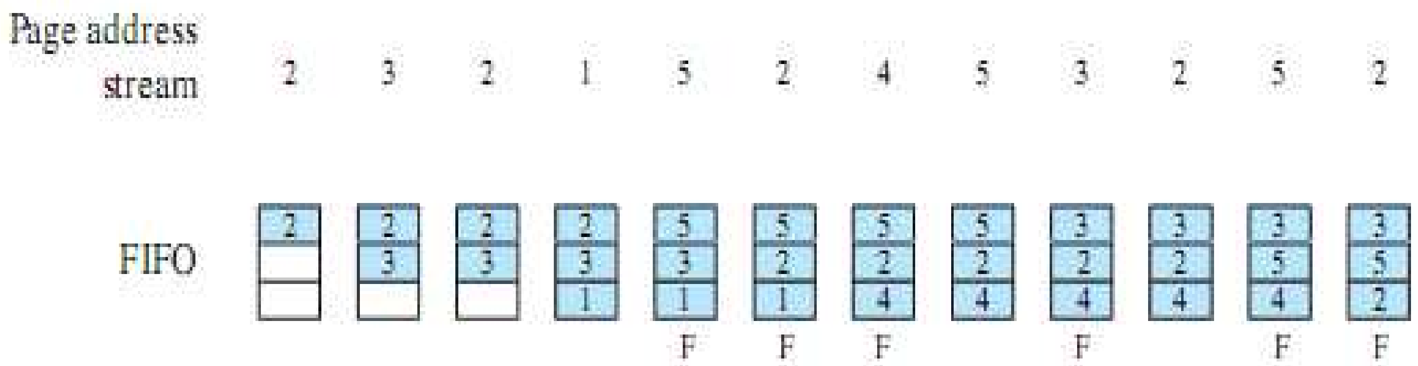


Figure 8.15 Behavior of Four Page Replacement Algorithms

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

FIFO Example

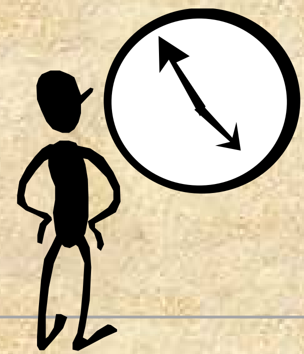


F= page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

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



Clock Policy Example

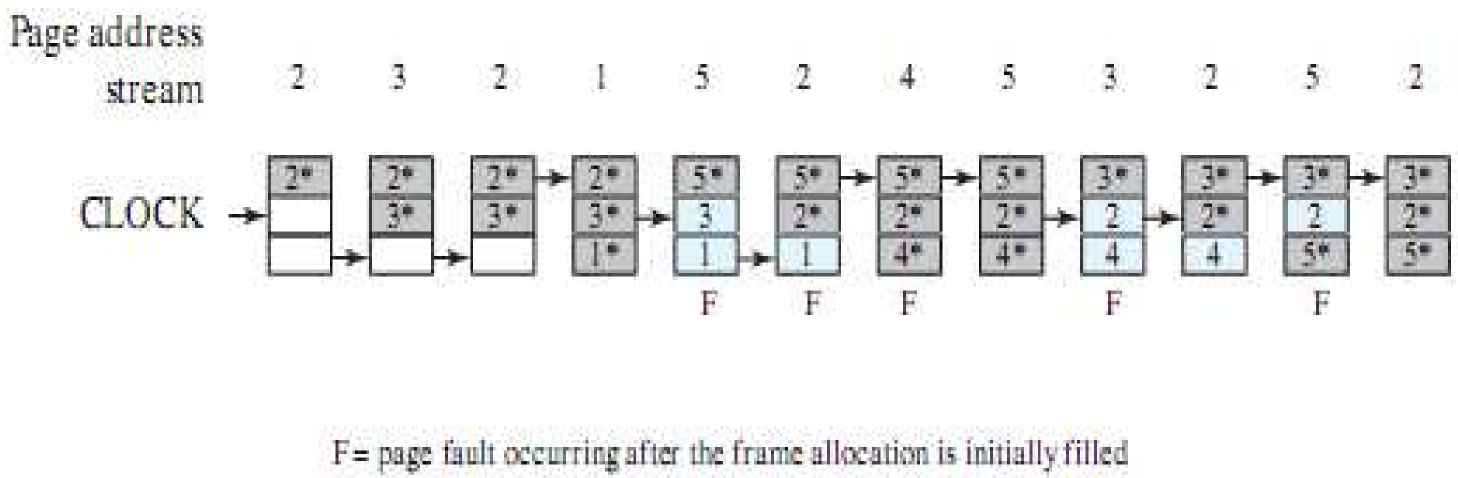


Figure 8.15 **Behavior of Four Page Replacement Algorithms**

Clock Policy

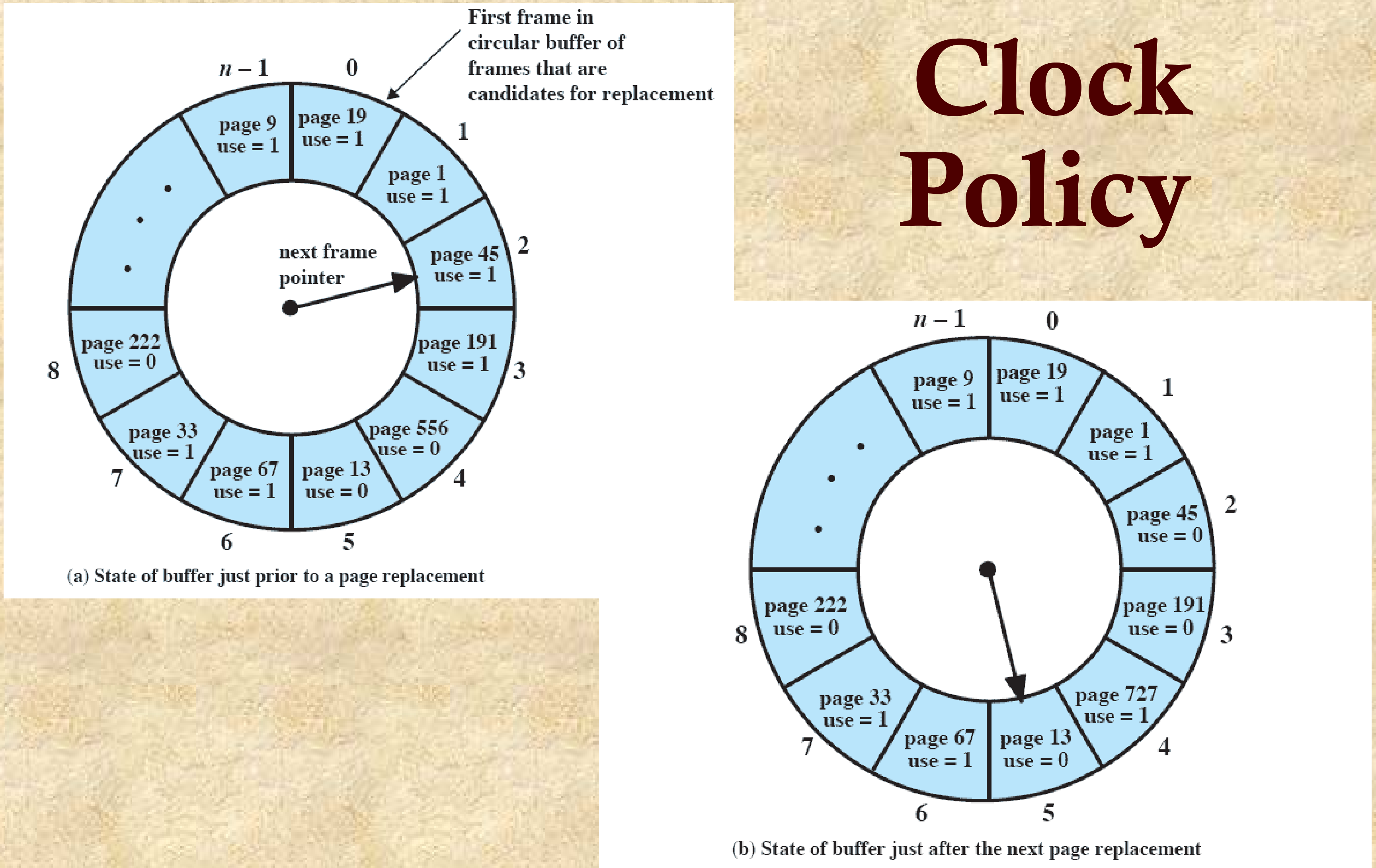


Figure 8.16 Example of Clock Policy Operation

Comparison of Algorithms

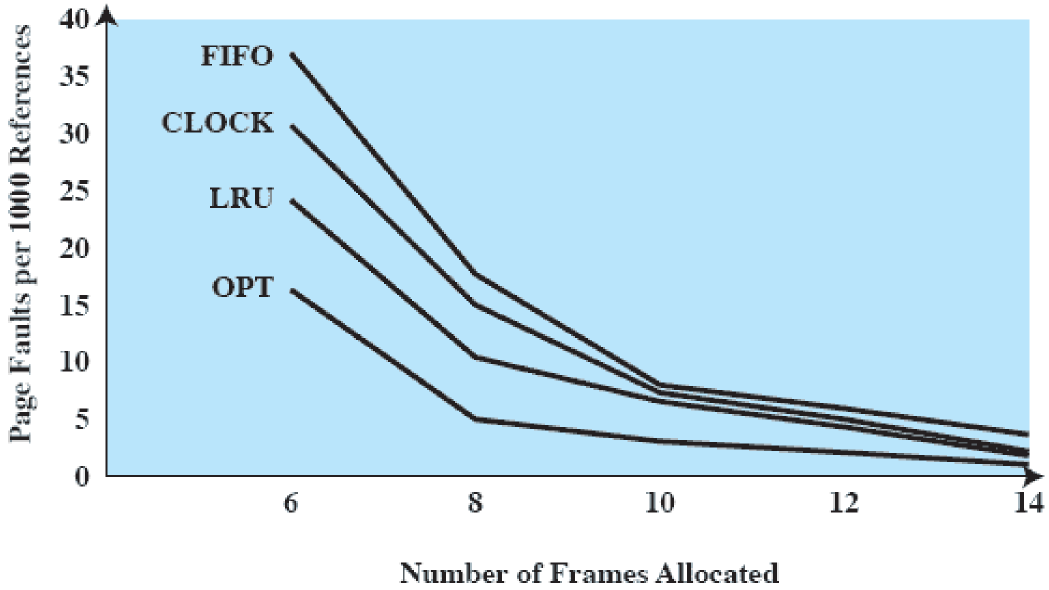


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

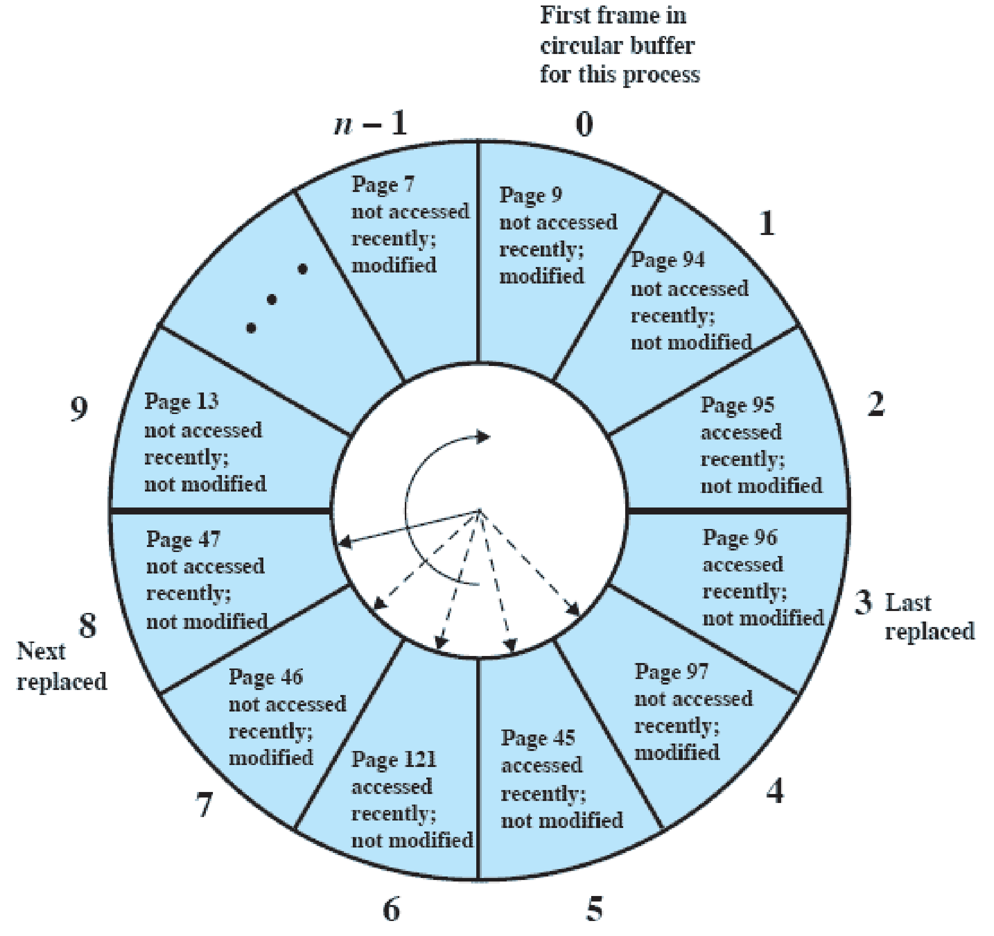


Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Clock Policy

Combined Examples

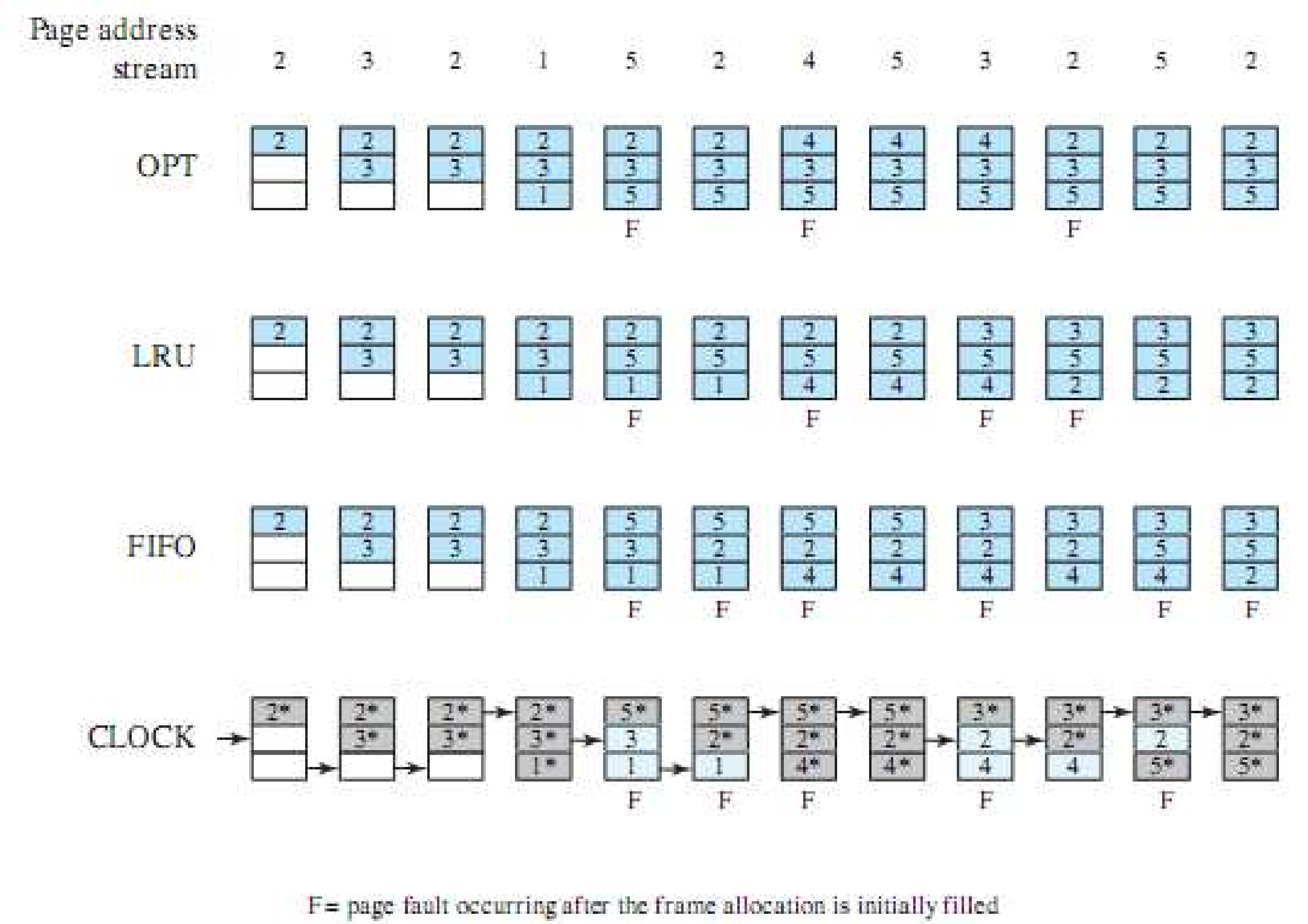
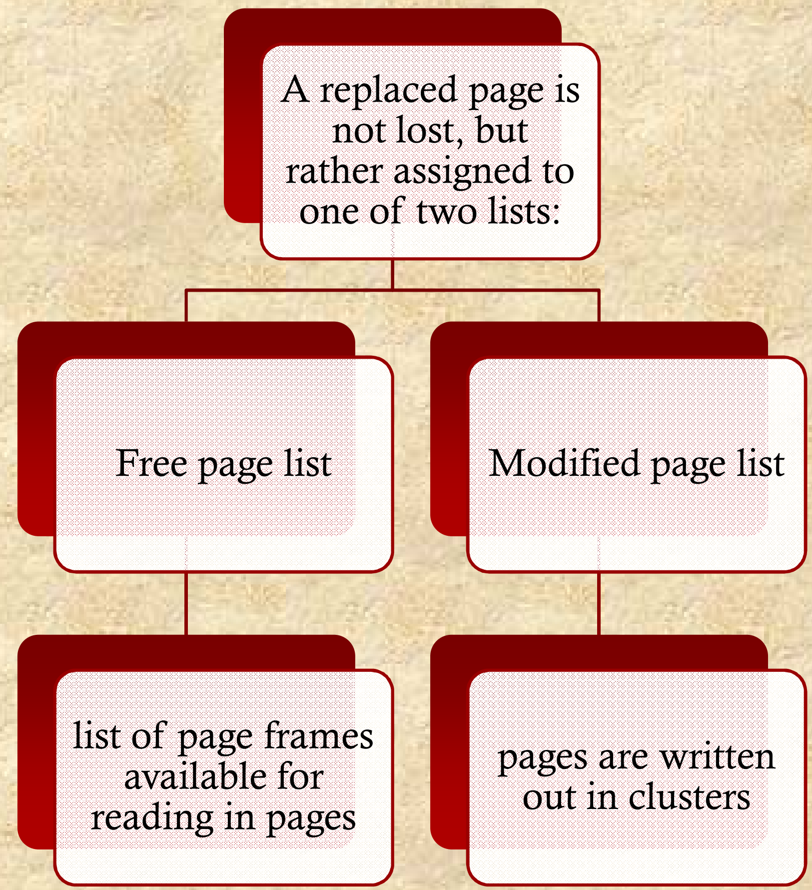


Figure 8.15 **Behavior of Four Page Replacement Algorithms**

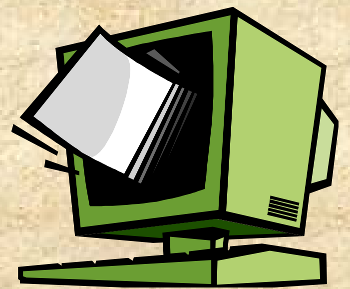
Page Buffering

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



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

Resident Set Management Summary

	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.

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

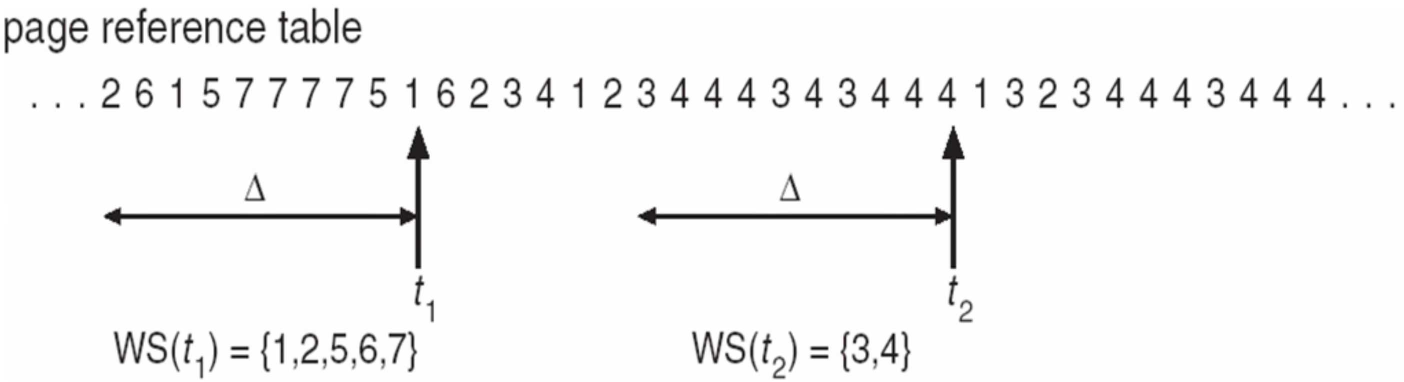
Sequence of Page References	Window Size, Δ			
	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	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 8.19

Working Set of Process as Defined by Window Size

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\, WSS_i \equiv$ total demand frames
  - Approximation of locality
- if $D > m \Rightarrow$ Thrashing
- Policy if D > m, then suspend or swap out one of the processes



Keeping Track of the Working Set

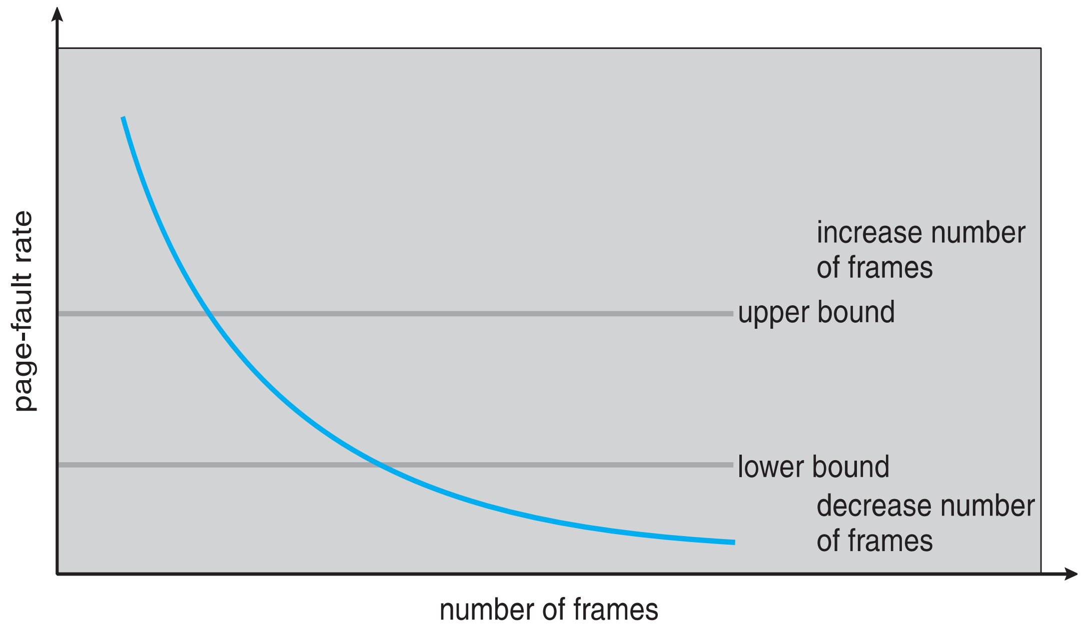
- Approximate with interval timer + a reference bit
- Example: Δ = 10,000
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

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

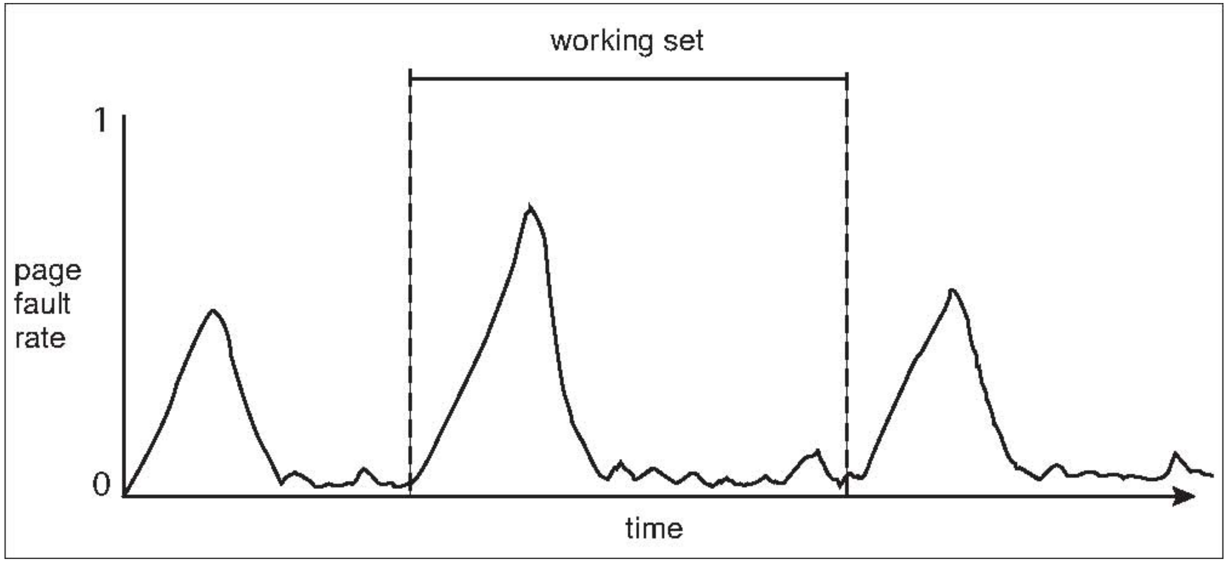
Page-Fault Frequency

- More direct approach than WSS
- Establish "acceptable" **page-fault frequency** (**PFF**) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Working Sets and Page Fault Rates

- Direct relationship between working set of a process and its page-fault rate
- Working set changes over time
- Peaks and valleys over time



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

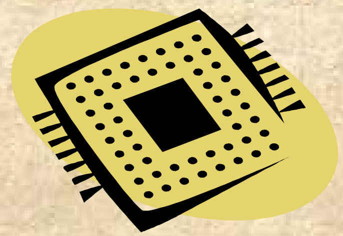
a page is written out to secondary memory only when it has been selected for replacement

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



Multiprogramming

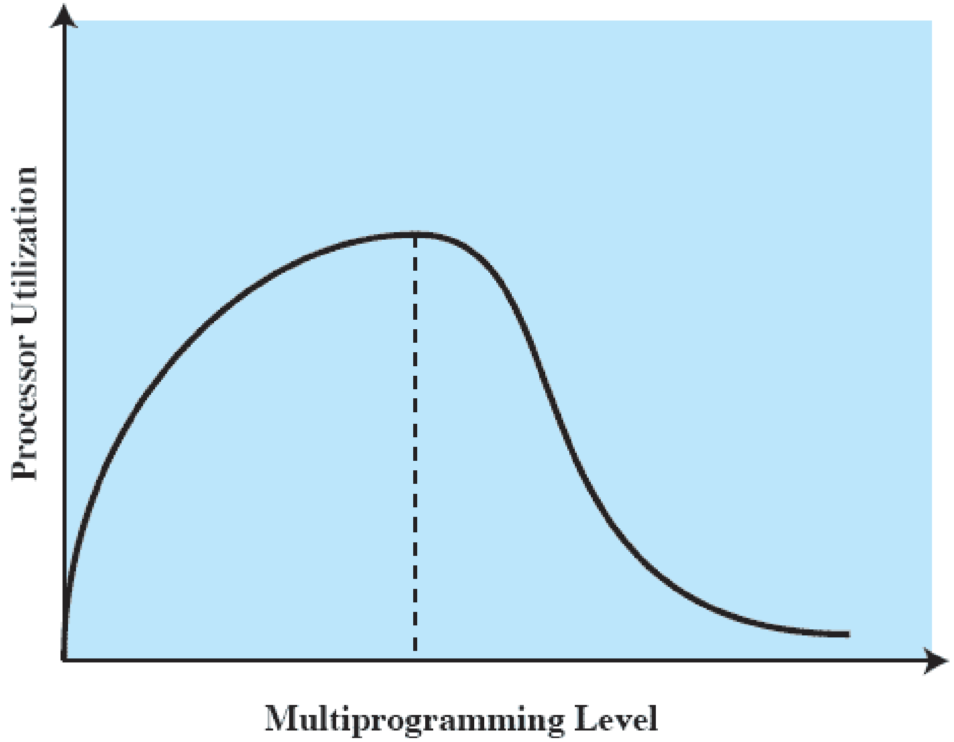


Figure 8.21 Multiprogramming Effects

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	Kernel Memory Allocator
provides a virtual memory capability that allocates page frames in main memory to processes	allocates memory for the kernel
allocates page frames to disk block buffers	

UNIX SVR4 Memory Management Formats

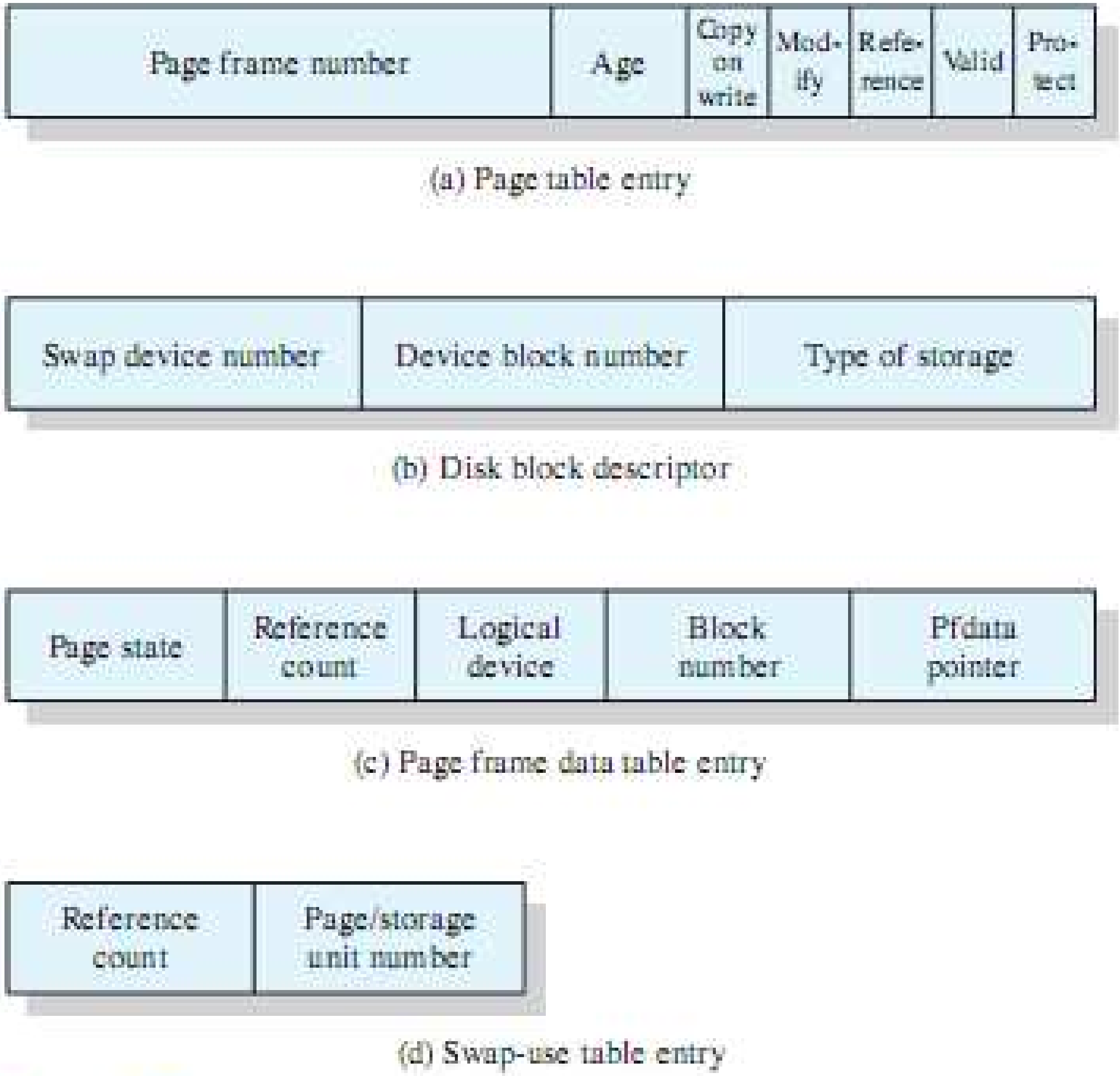


Figure 8.22 UNIX SVR4 Memory Management Formats

Table 8.6

UNIX SVR4 Memory Management Parameters (page 1 of 2)

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 2 of 2)

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.

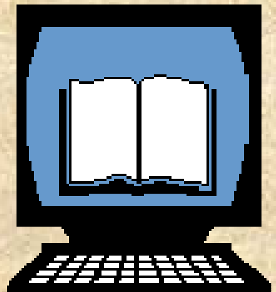
Swap-Use Table Entry

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.

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



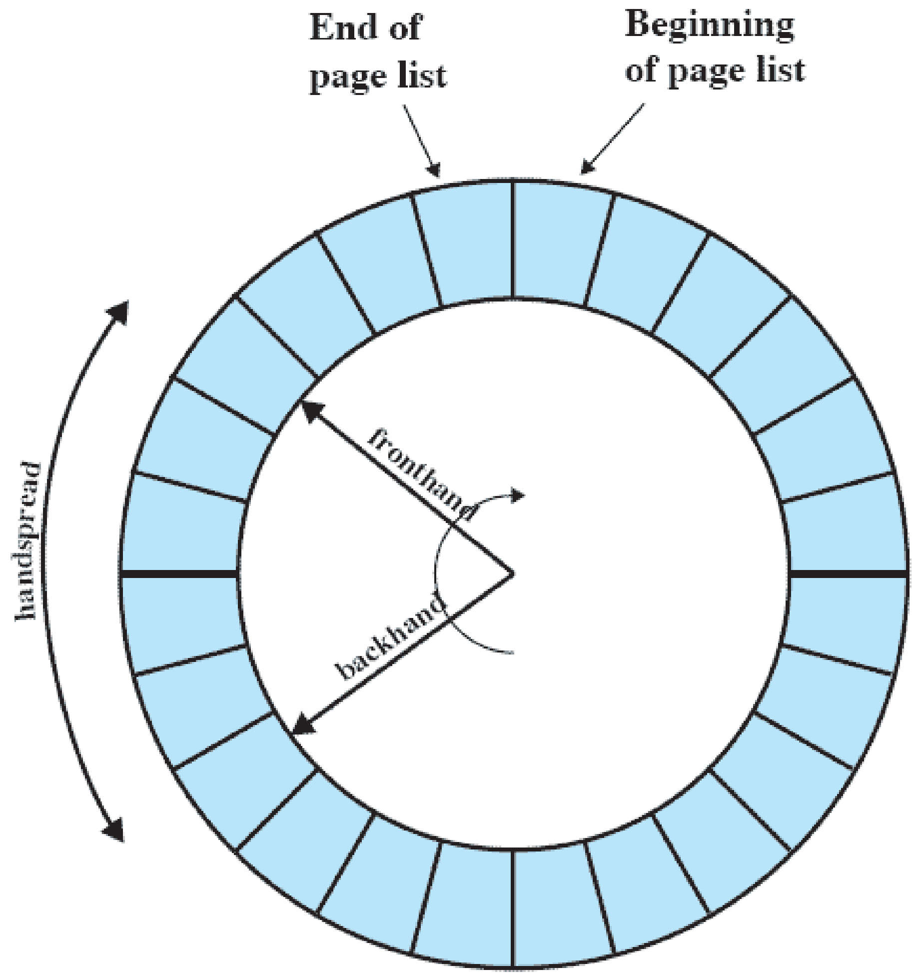


Figure 8.23 Two-Handed Clock Page-Replacement Algorithm

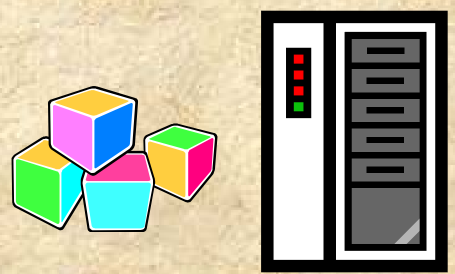
“Two Handed” Clock Page Replacement

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



Lazy Buddy System Algorithm

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 8.24 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
- process has a single page directory
- each entry points to one page of the page middle directory
- must be in main memory for an active process

Page middle directory
- may span multiple pages
- each entry points to one page in the page table

Page table
- may also span multiple pages
- each entry refers to one virtual page of the process

Address Translation

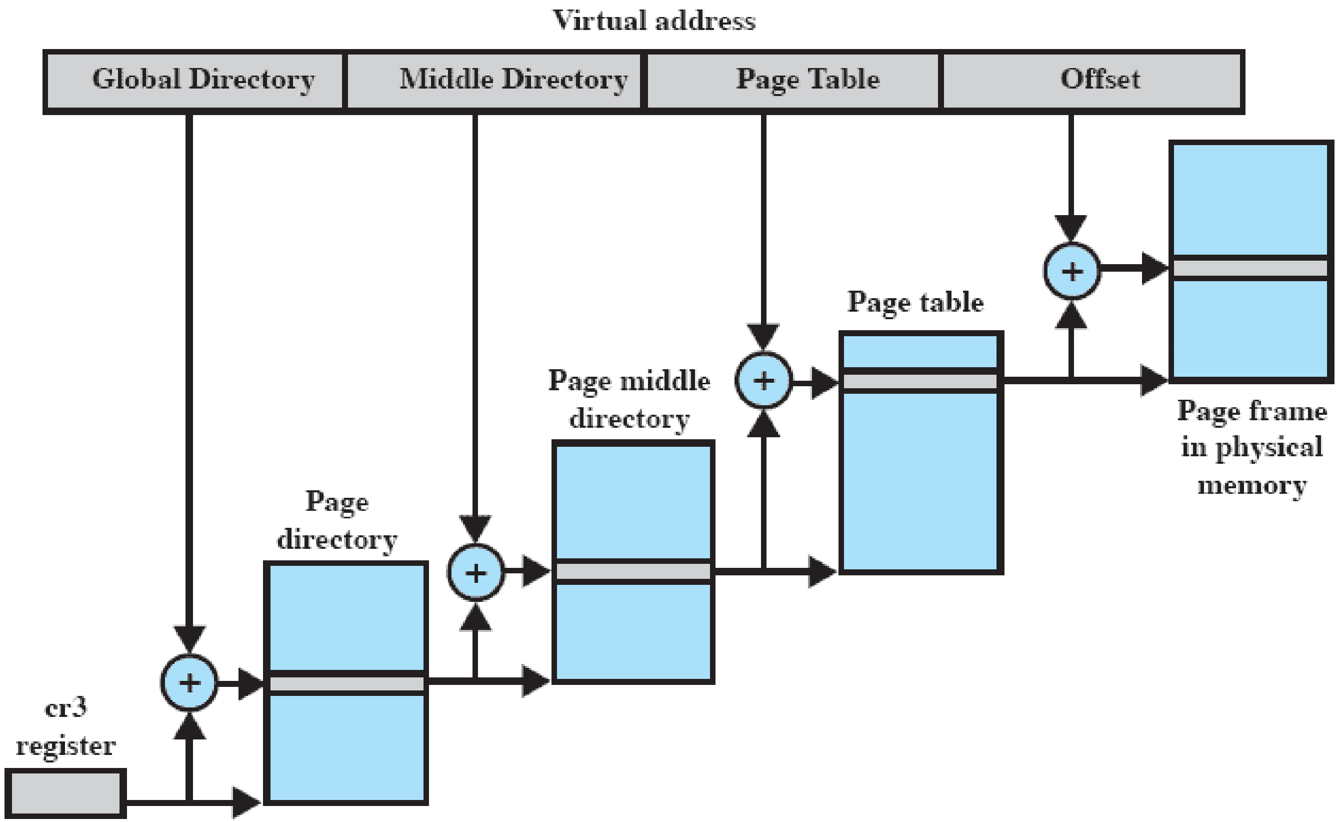


Figure 8.25 Address Translation in Linux Virtual Memory Scheme

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 is some time and is the best candidate for replacement
- A form of least frequently used policy

Kernel Memory Allocation

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

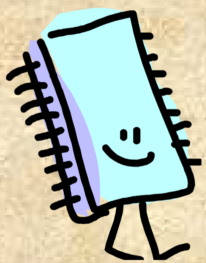
Possible owners of a frame include:

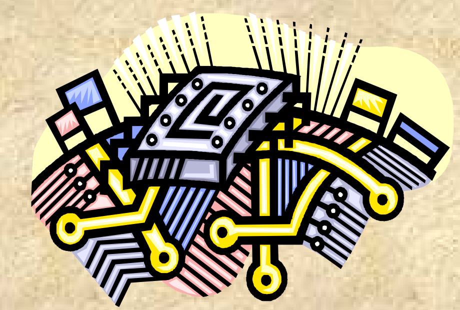
- user-space processes
- dynamically allocated kernel data
- static kernel code
- page cache

- 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

32-Bit Windows Address Space

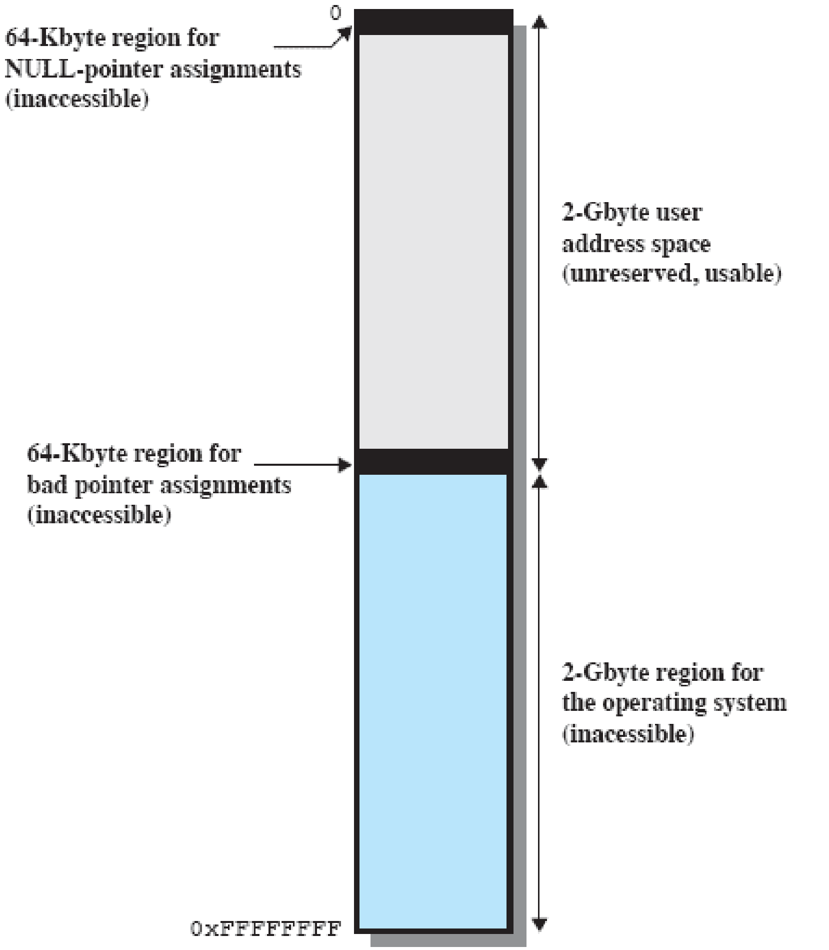


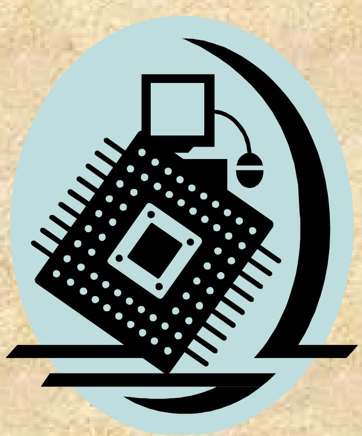
Figure 8.26 Windows Default 32-bit Virtual Address Space

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:

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



Summary

- Desirable to:
  - maintain as many processes in main memory as possible
  - free programmers from size restrictions in program development
- With virtual memory:
  - all address references are logical references that are translated at run time to real addresses
  - a process can be broken up into pieces
  - two approaches are paging and segmentation
  - management scheme requires both hardware and software support