

Chapter 8 – Virtual Memory

Lecture 6

Roadmap

- **Hardware and Control Structures**
- Operating System Software

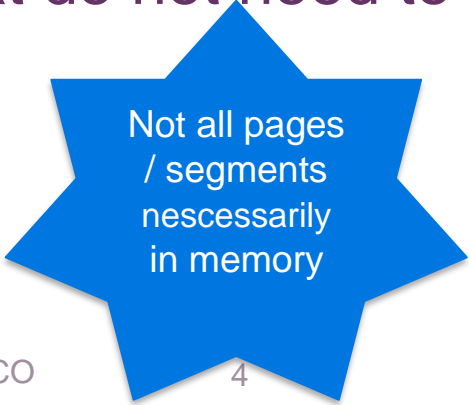
Terminology

Table 8.1 Virtual Memory Terminology

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

Key points in Memory Management

- 1) Memory references are logical addresses dynamically translated into physical addresses at run time
 - A process may be swapped in and out of main memory occupying different regions at different times during execution
- 2) A process may be broken up into pieces that do not need to be located contiguously in main memory



Not all pages
/ segments
necessarily
in memory

What happens when a process is selected for first time execution?

Execution of a Process (1)

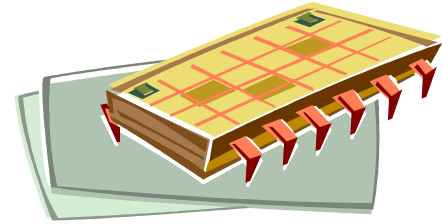
- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
 - Operating system places the process in a blocking state

Execution of a Process (2)

- Piece of process that contains the logical address is brought into main memory
 - Operating system issues a disk I/O Read request
 - Another process is dispatched to run while the disk I/O takes place
 - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Types of Memory

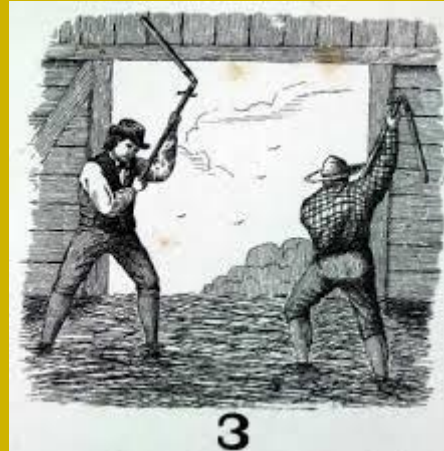
- Real memory
 - Main memory, the actual RAM
- Virtual memory
 - Memory on disk
 - Allows for effective multiprogramming and relieves the user of tight constraints of main memory



How many processes shall we bring into memory?

Thrashing

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



Principle of locality



Figure 8.1 Paging Behavior

Support Needed for Virtual Memory

Virtual Memory needs to be effective and practical:

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

Paging

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

(see next slide)

Paging Table

Virtual Address



Page Table Entry



Virtual Address



Segment Table Entry



Virtual Address



Segment Table Entry



Page Table Entry



P = present bit
M = Modified bit

Address Translation – Using pages

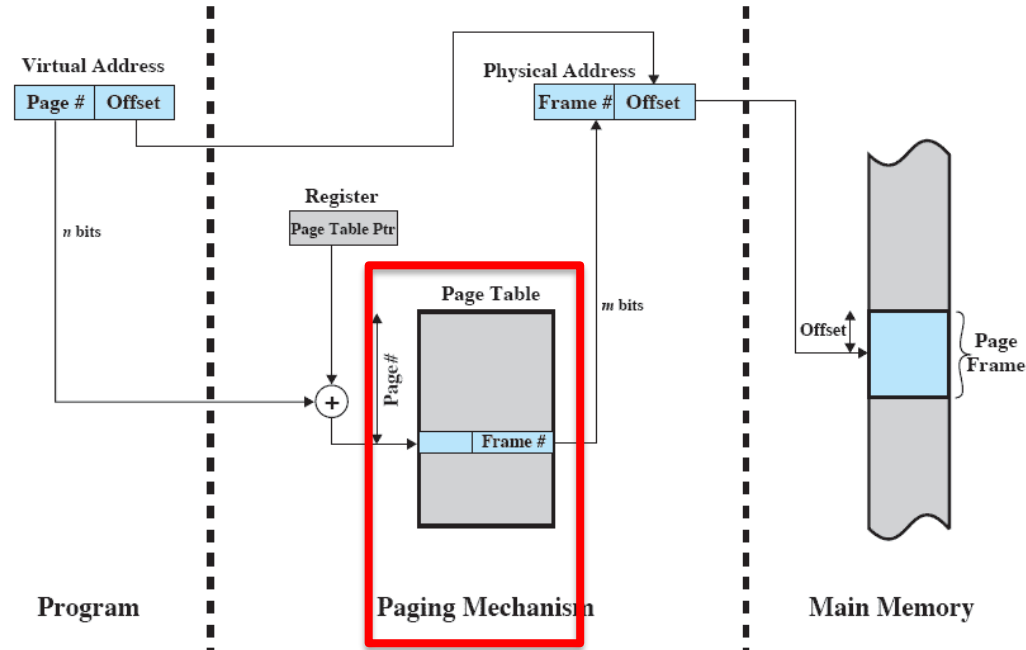


Figure 8.3 Address Translation in a Paging System OSCO

Page table size

- User space: 4Gb
 - Page size: 4 KBytes
 - How large is the page table?
- $4\text{Gb} = 2^{32}$ Userspace
 - $4\text{Kb} = 2^{12}$ Page size
 - So: 2^{20} pages = 1 M pages
 - Size pagetable = $2^{20+2} = 4\text{Mb}$
 - Is: $2^{10} = 1024$ pages in itself!
PER PROCESS!

Two-Level Hierarchical Page Table

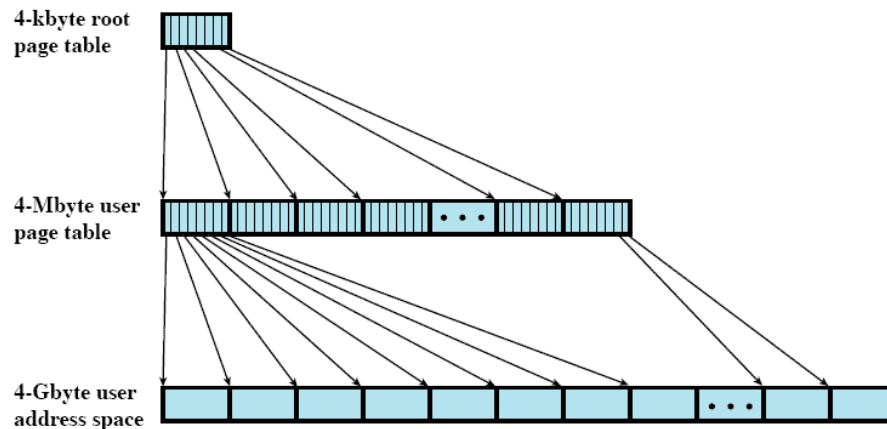


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation for Hierarchical page table

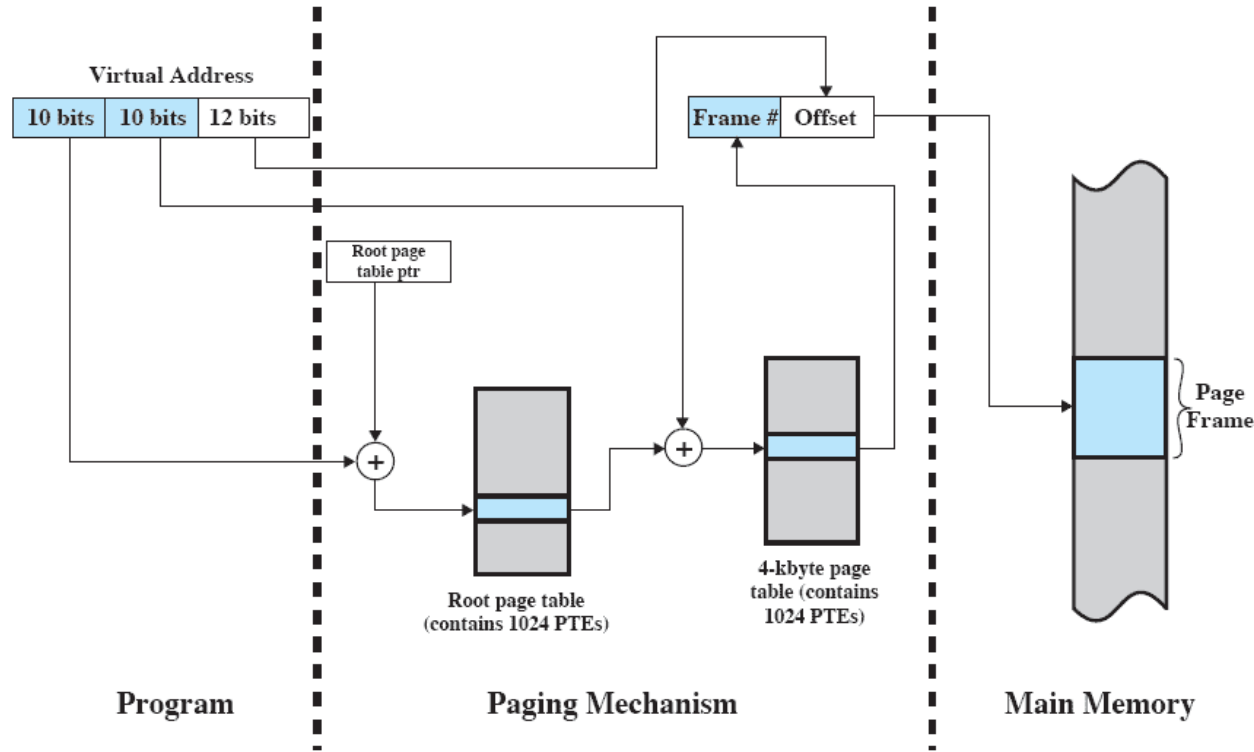


Figure 8.5 Address Translation in a Two-Level Paging System

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses
 - One to fetch the page table
 - One to fetch the data

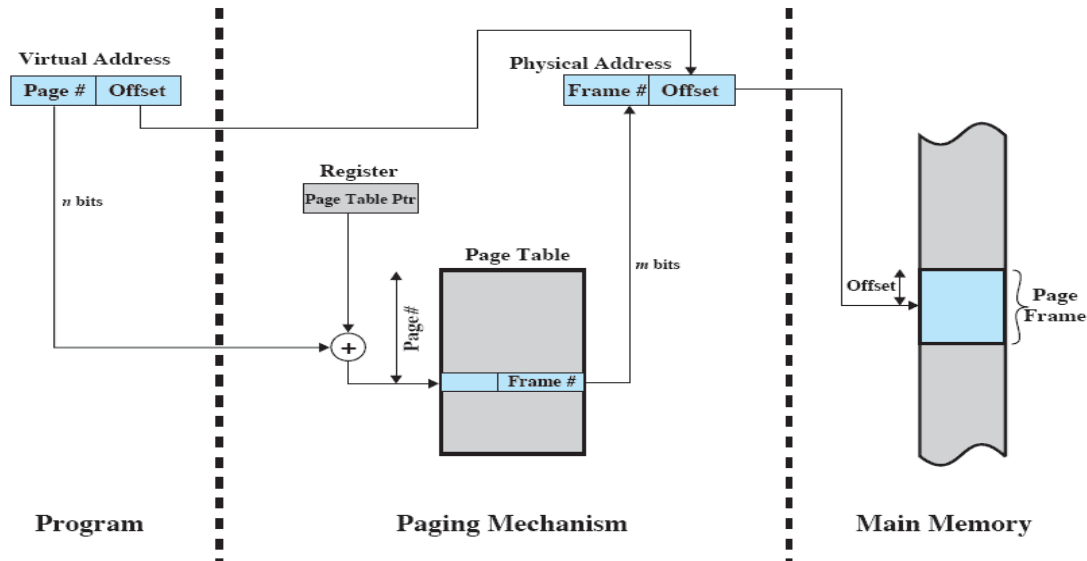
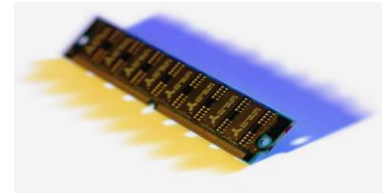


Figure 8.3 Address Translation in a Paging System

Looking into the Process Page Table

- First checks if page is already in main memory
 - If not in main memory a page fault is issued
- The TLB is updated to include the new page entry



Translation Lookaside Buffer

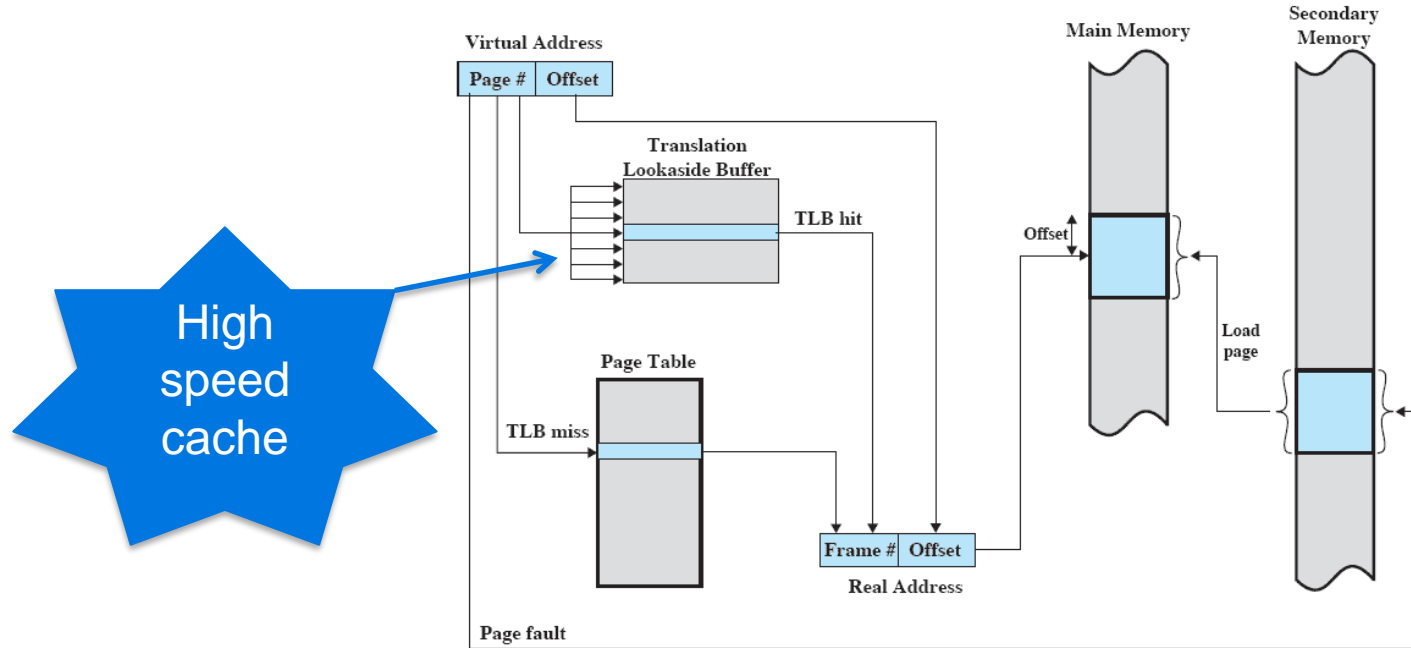
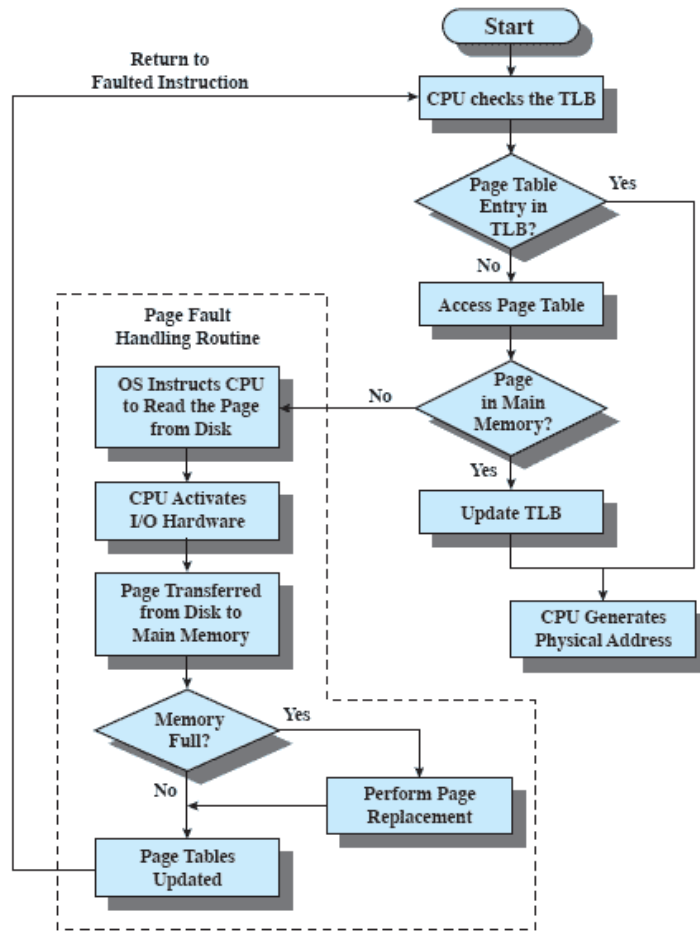


Figure 8.7 Use of a Translation Lookaside Buffer

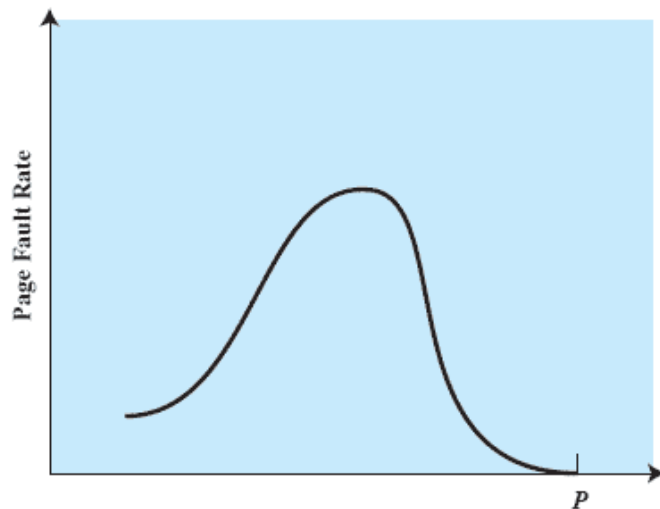
TLB operation



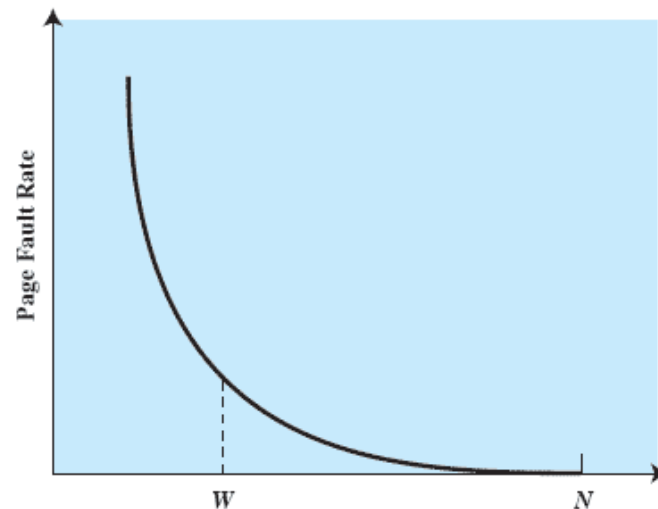
Page Size

- Smaller page size, less amount of internal fragmentation
 - But Smaller page size, more pages required per process
 - More pages per process means larger page tables
 - Larger page tables means large portion of page tables in virtual memory (not in main memory)
 - Secondary memory is designed to efficiently transfer large blocks of data

Page Size



(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 8.11 Typical Paging Behavior of a Program

Table 8.3 Example Page Sizes

Example Page Size

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBMAS/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
IBMPowerPC	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Intel core i7 4Kbytes to 1Gbyte

Segmentation

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

- May be unequal, dynamic size
- Simplifies handling of growing data structures
- Allows programs to be altered and recompiled independently
- Lends itself to sharing data among processes
- Lends itself to protection

Segment Table Entries

Virtual Address

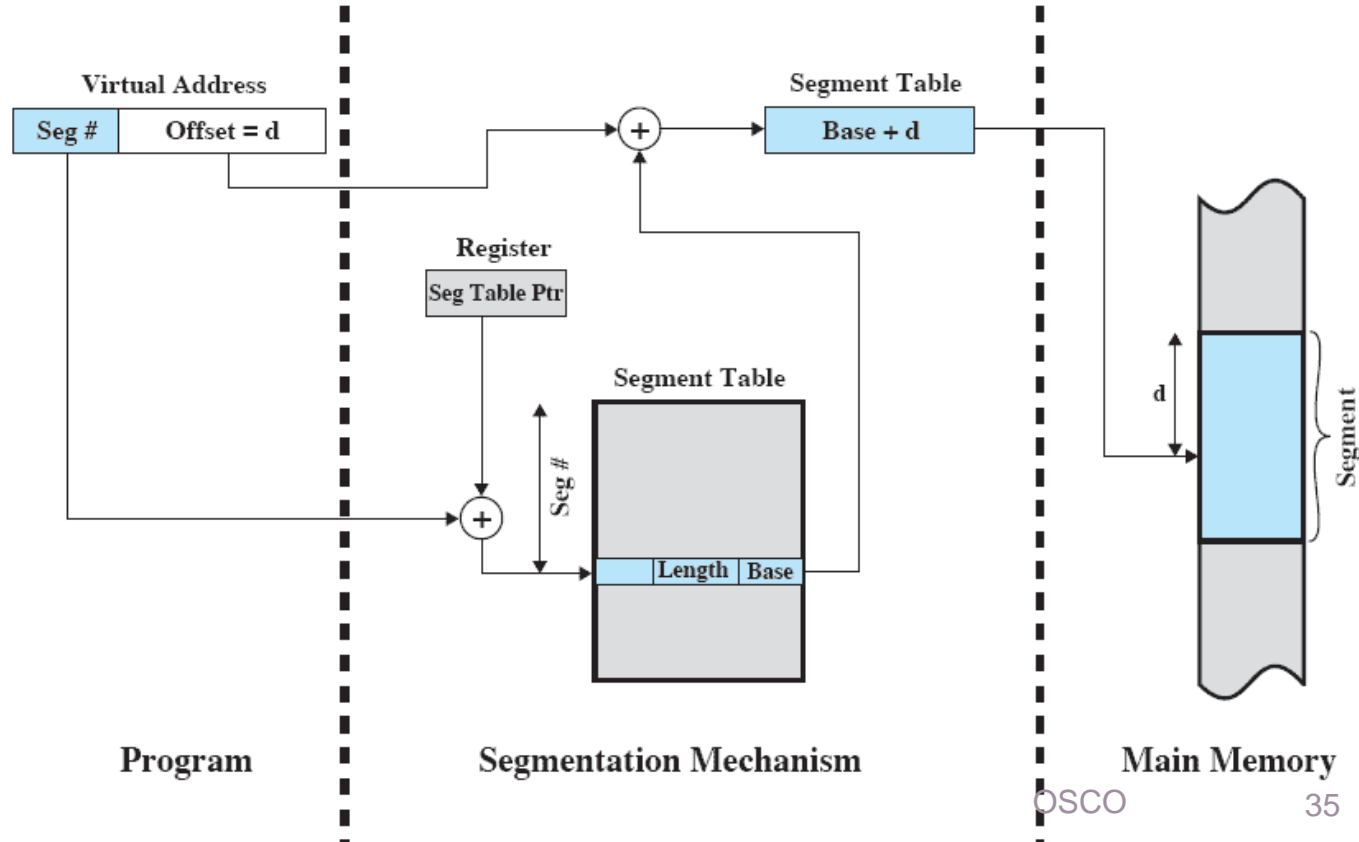


Segment Table Entry



(b) Segmentation only

Address Translation in Segmentation



Combined Paging and Segmentation

- Paging is transparent to the programmer
- Paging eliminates external fragmentation
- Segmentation is visible to the programmer
- Segmentation allows for growing datastructures, modularity, and support for sharing and protection
- Each segment is broken into fixed-size pages

Combined Paging and Segmentation

Virtual Address



Segment Table Entry



Page Table Entry



P= present bit
M = Modified bit

(c) Combined segmentation and paging

Address Translation

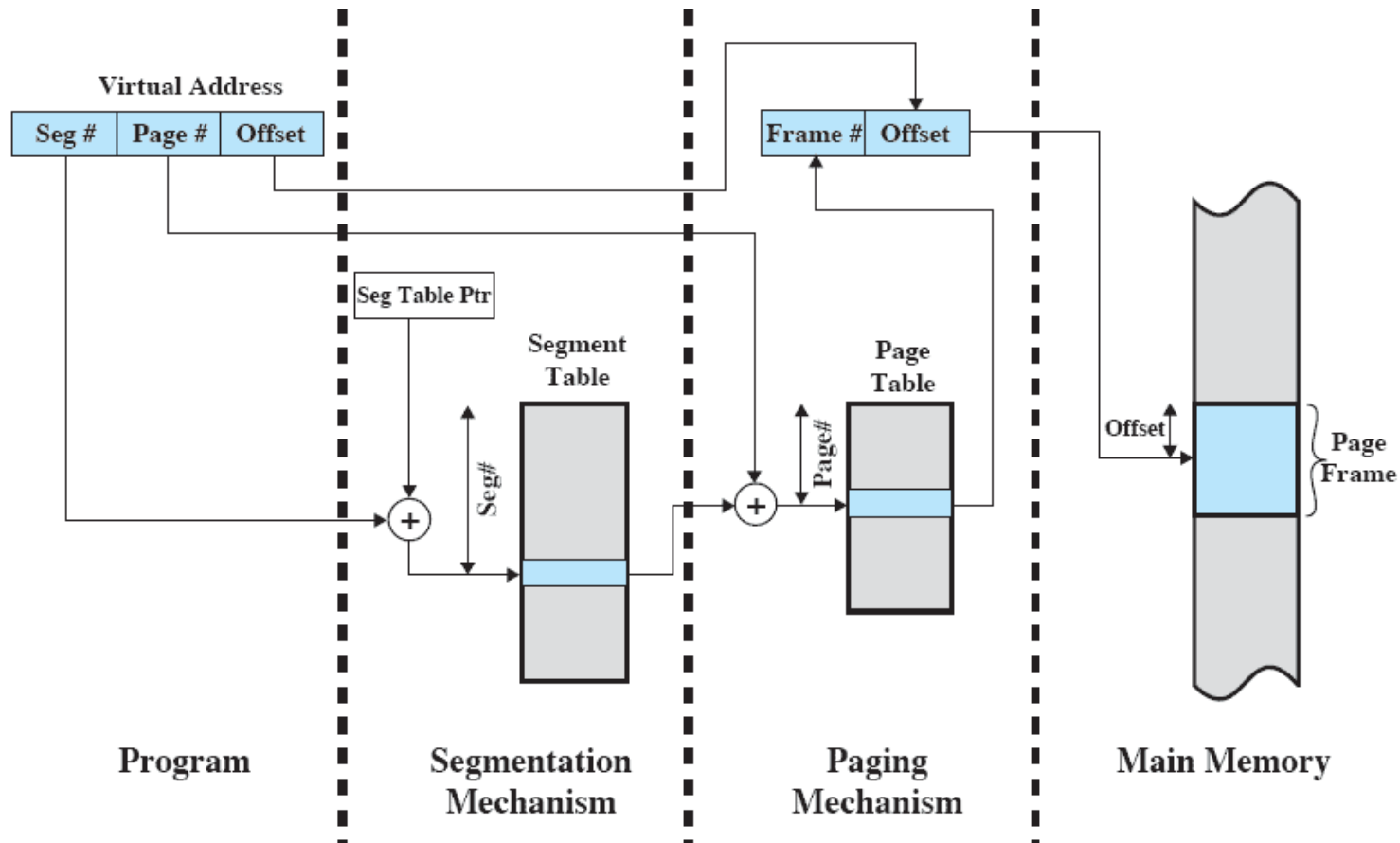
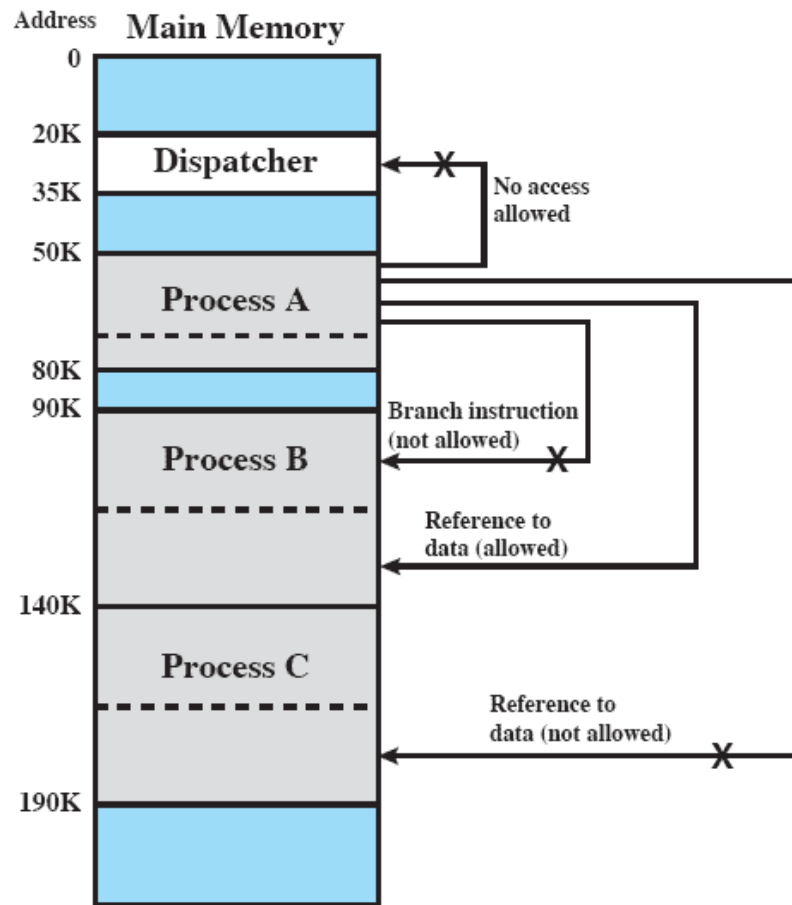


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection Relationships



Roadmap

- Hardware and Control Structures
- **Operating System Software**

Memory Management Decisions

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

Key Design Elements

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

Table 8.4 Operating System Policies for Virtual Memory

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

Fetch Policy Demand 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
 - Prepaging

Placement Policy

- Determines where in real memory a process piece is to reside

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy Demand 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
---	--

Replacement Policy

Fetch Policy Demand 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
--	--

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

But...

- Which page is replaced?
- Page removed should be the page least likely to be referenced in the near future

Replacement Policy: Frame Locking

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

Basic Replacement Algorithms

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

Examples

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

Optimal policy

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

Page address
stream

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

OPT

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

F

F

F

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

Figure 8.15 Behavior of Four Page Replacement Algorithms

Only
3
faults

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

- The LRU policy does nearly as well as the optimal policy.

Page address
stream

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

LRU

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

F

F

F

F

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

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
 - Simplest replacement policy to implement
- Page that has been in memory the longest is replaced
 - But, these pages may be needed again very soon if it hasn't truly fallen out of use

FIFO Example

6
page
faults

Page address
stream

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

FIFO

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

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

Figure 8.15 Behavior of Four Page Replacement Algorithms

- Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.



Clock Policy

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

Clock Policy Example

5
page
faults

Page address
stream

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

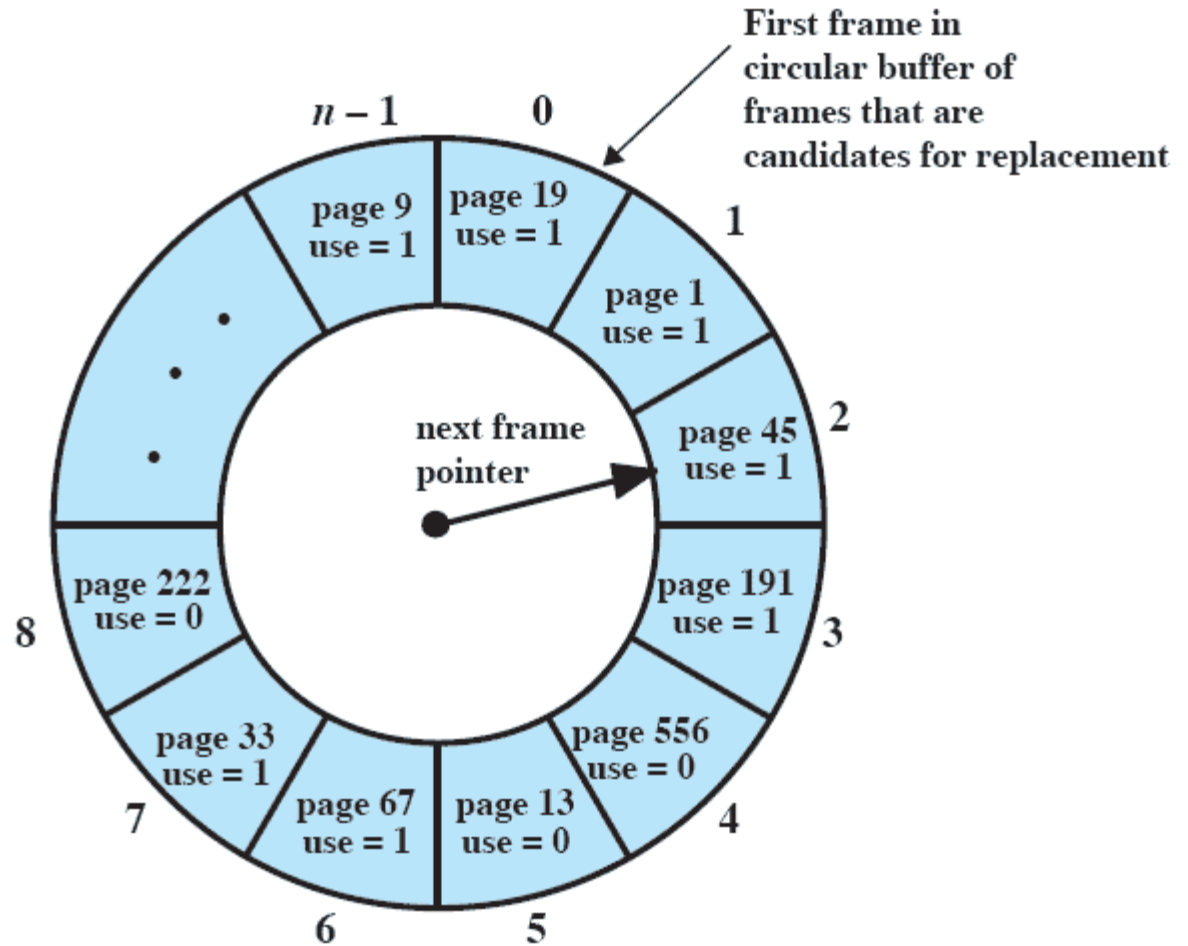


F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

- Note that the clock policy is adept (==highly skilled) at protecting frames 2 and 5 from replacement.

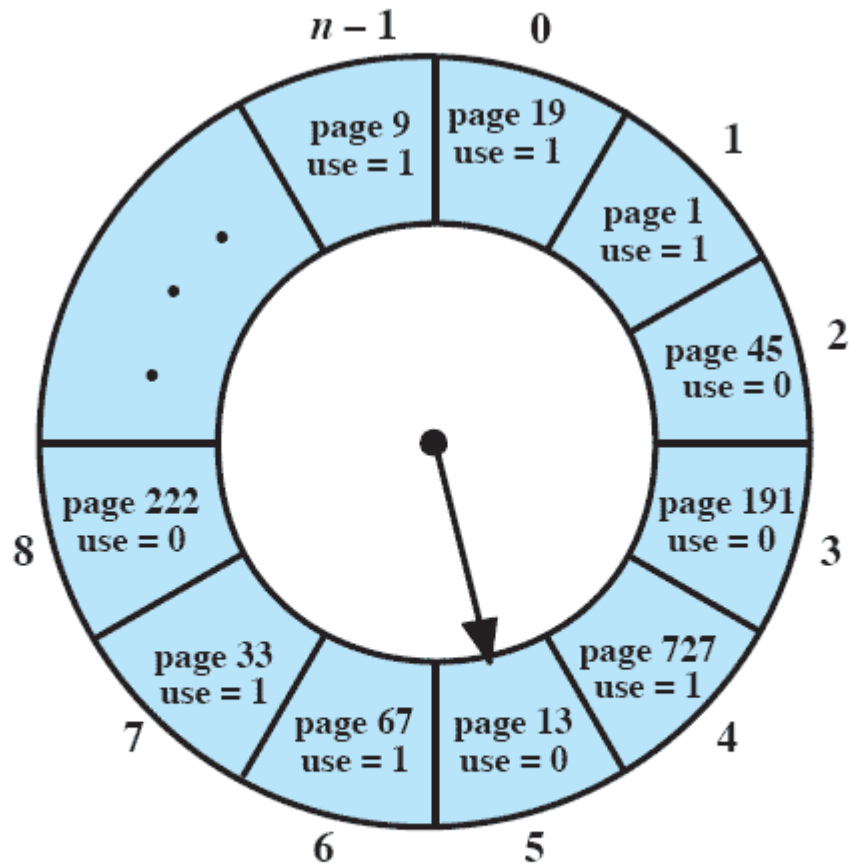
Clock Policy



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

OSCO

Clock Policy



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

Comparison

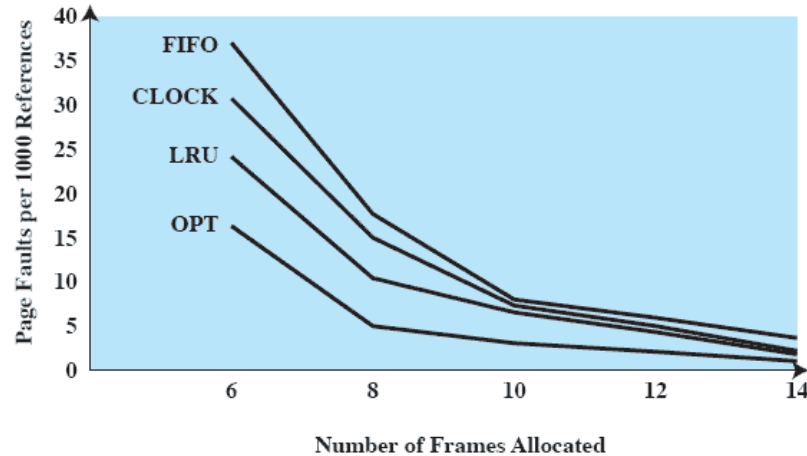


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy Demand 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
---	--

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 that can reside in memory.
 - Small number of pages loaded increases page faults.
 - Beyond a certain size, further allocations of pages will not affect the page fault rate.

Fetch Policy Demand 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
--	--

Resident Set Size

- Fixed-allocation
- Variable-allocation



Replacement Scope

Table 8.4 Operating System Policies for Virtual Memory	
Fetch Policy Demand Prepaging	Resident Set Management Resident set size Fixed Variable Replacement Scope Global Local
Placement Policy Replacement Policy Basic Algorithms Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page buffering	Cleaning Policy Demand Precleaning
	Load Control Degree of multiprogramming

- The scope of a replacement strategy can be categorized as:
 - *global* or
 - *local*



Fixed Allocation, Local Scope

- Decide ahead of time the amount of allocation to give a process
 - What if too small?
 - What if too big?

Variable Allocation, Global Scope

- Easiest to implement
 - Adopted by many operating systems
- Operating system keeps list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no free frame, replaces one from another process
 - That gives you difficulties ... which one to replace?

Variable Allocation, Local Scope

- When new process added, allocate number of page frames based on application type, program request, or other criteria
- When page fault occurs, select page from among the resident set of the process that suffers the fault
- Reevaluate allocation from time to time

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy Demand 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
--	--

Cleaning Policy

- A cleaning policy is concerned with determining when a modified page should be written out to secondary memory.
 - Demand cleaning
 - Precleaning

Fetch Policy Demand 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	

Load Control

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

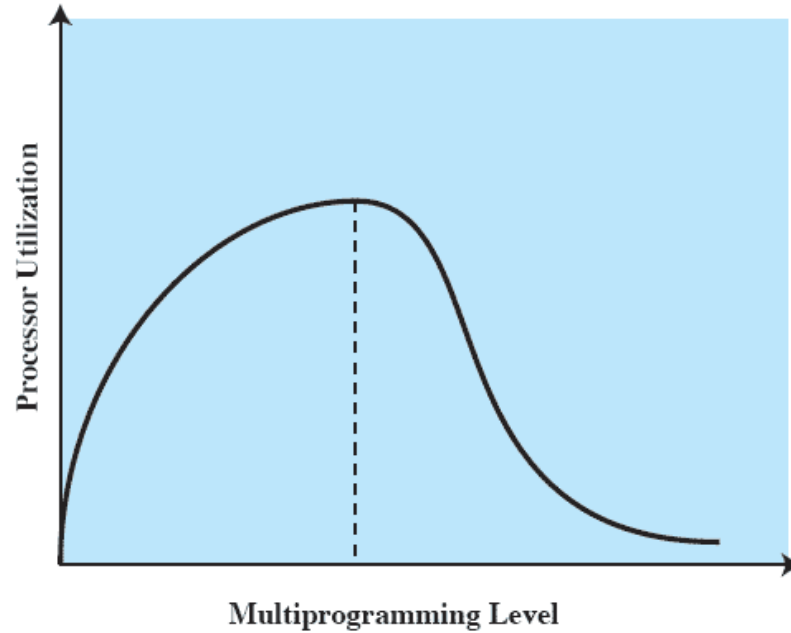


Figure 8.21 Multiprogramming Effects

OSCO

Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be suspended (swapped out).
- Six possibilities exist...

Suspension policies

- 1) Lowest priority process
- 2) Faulting process
 - This process does not have its working set in main memory so it will be blocked anyway
- 3) Last process activated
 - This process is least likely to have its working set resident

Suspension policies cont.

- 4) Process with smallest resident set
 - This process requires the least future effort to reload
- 5) Largest process
 - Obtains the most free frames
- 6) Process with the largest remaining execution window

Random selection & Practical assignment explanation

Questions?

