Memory Management

Unit 5: Memory Management (7 Hours)

Memory Management concepts: Memory Management requirements, Memory Partitioning: Fixed, Dynamic Partitioning, Buddy Systems, Fragmentation, Paging, Segmentation, Address translation.
Placement Strategies: First Fit, Best Fit, Next Fit and Worst Fit.
Virtual Memory: Concepts, Swapping, VM with Paging, Page Table Structure, Inverted Page Table, Translation Lookaside Buffer, Page Size, VM with Segmentation,VM with combined paging and segmentation.
Page Replacement Policies: FIFO, LRU, Optimal, Clock.
Swapping issues: Thrashing

Memory Management

- In a uniprogramming system, main memory is divided into two parts:
  - one part for the operating system (resident monitor, kernel) and
  - one part for the program currently being executed.
- In a multiprogramming system, the “user” part of memory must be further subdivided to accommodate multiple processes.
- The task of subdivision is carried out dynamically by OS is known as **Memory Management.**
- Memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.
- Memory Management, involves swapping blocks of data from secondary storage

Execution of a Program

- 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.

Memory Management Requirements

Relocation

Protection

Sharing

Logical organisation

Physical organisation

Memory Management Terms

Term	Description
Frame	***Fixed***-length block of main memory.
Page	***Fixed***-length block of data in secondary memory (e.g. on disk).
Segment	***Variable-length*** block of data that resides in secondary memory.

Memory Management Requirements

1. Relocation:

- Programmer does not know where the program will be placed in memory when it is executed.
- While the program is executing, it may be swapped to disk and returned to main memory at a different location (relocated).
- Memory references must be translated in the code to actual physical memory address.

Addressing

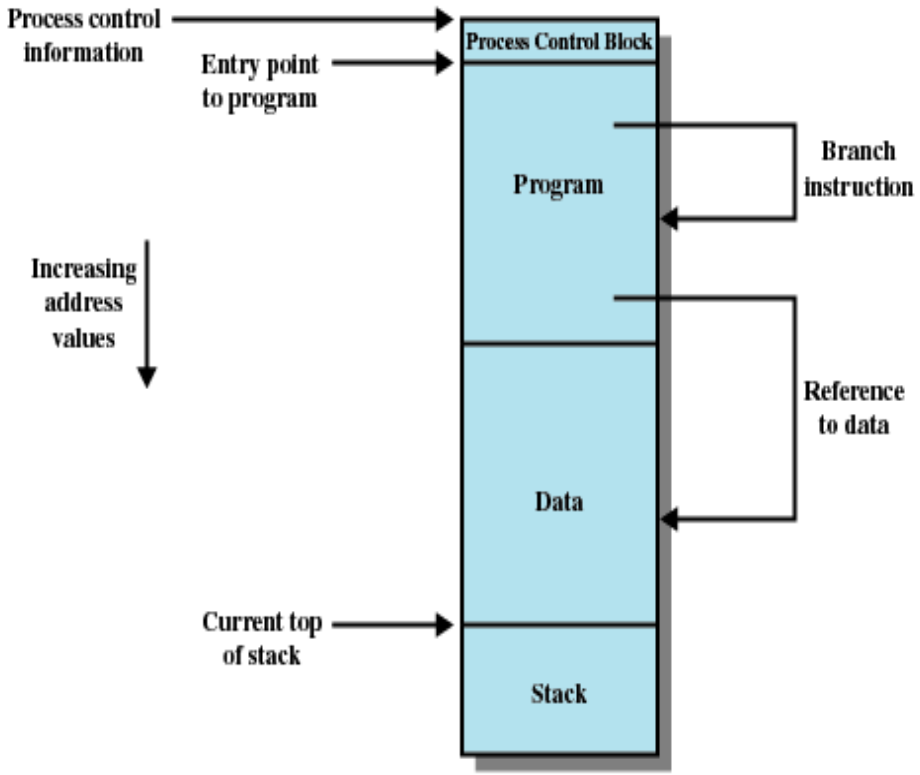


Figure 7.1 Addressing Requirements for a Process

The OS needs to know the location of:

- process control information
- the execution stack,
- the entry point to begin execution of the program for this process.

There will be memory references within the program

Eg. BT, GD, PD

Processor HW and OS should be able to translate these into actual physical memory addresses

Changes every time

Memory Management Requirements

2. Protection

- Processes should not be able to reference memory locations in another process without permission.
- Impossible to check absolute addresses at compile time.
- Must be checked at run time.
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
  - Operating system cannot anticipate all of the memory references a program will make

Memory Management Requirements

3. Sharing

- Allow several processes to access the same portion of memory
  - Ex. Number of processes executing same program
- Better to allow each process access to the same copy of the program rather than have their own separate copy
- Protection mechanism must have flexibility to allow several processes to access the same portion of memory
- Processes that are cooperating on some task may need to share access to the same data structure.

Memory Management Requirements

4. Logical Organization

- Memory is organized linearly (usually)

  - Main memory is usually organized as a linear, or 1-D address space, consisting of a sequence of bytes or words.

  - Secondary memory, at its physical level, is similarly organized.

  If memory can also be dealt with in form of modules, there are several advantages(Programs are written in modules)

  - Modules can be written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
  - Share modules among processes.

- Ex: use of segmentation uses module concept

Memory Management Requirements

5. Physical Organization

- Memory is organized into Two –levels
- Main memory :fast , expensive, volatile
- Secondary memory : Slow , less expensive, non-volatile
- Flow of information b/n main & secondary memory is major concern
- Can it be programmer's responsibility?
- No
- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory
  - Programmer dependent
  - Wastage of time of programmer
  - Programmer does not know how much space will be available and where that space will be.

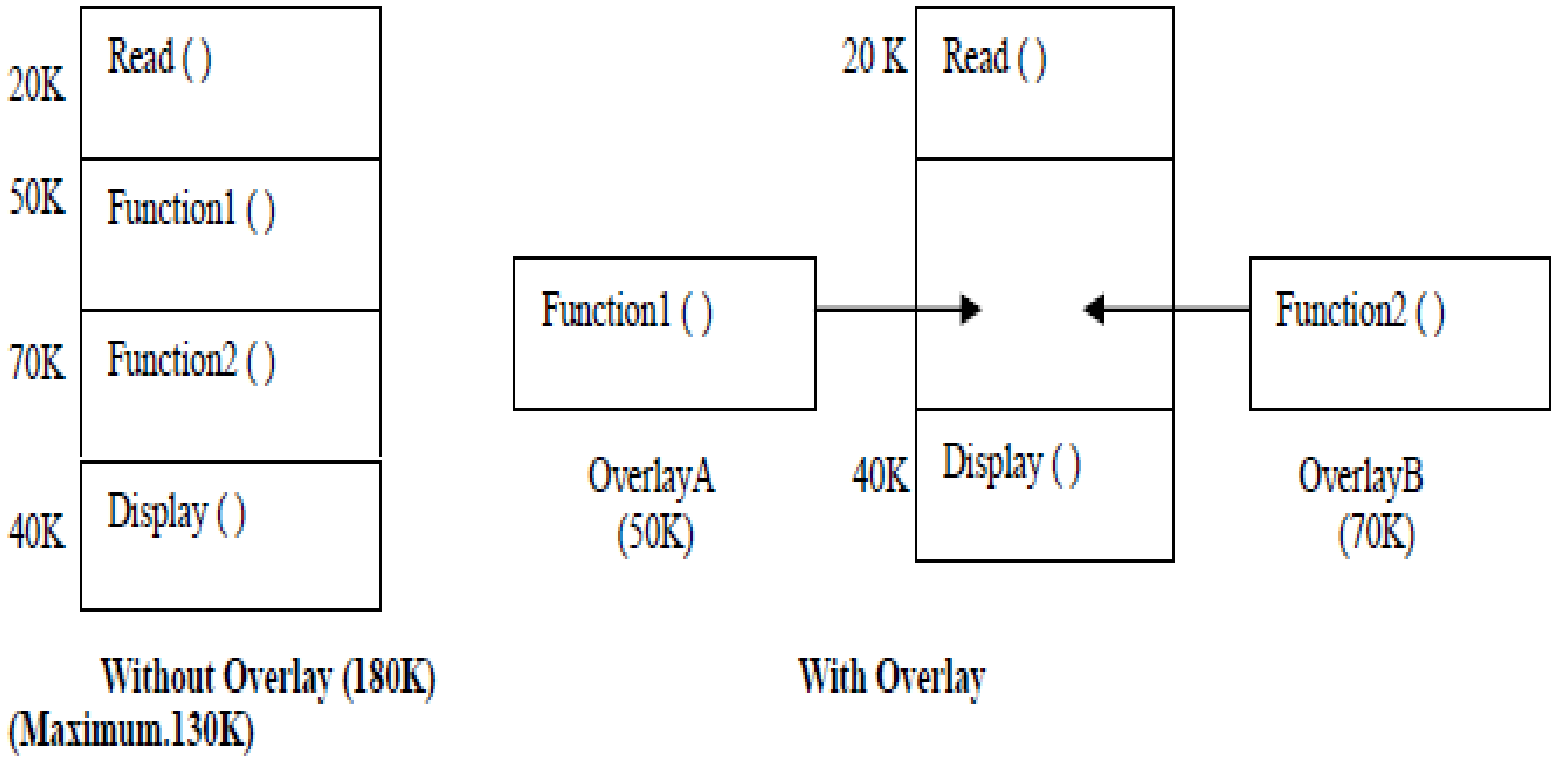
Overlays

- Program & its related data is loaded in physical memory for execution.
- What happens if process is larger that the amount of memory allocated to it?
- Use overlays.
- Overlays can be implemented by users without OS support
- The entire program /application is divided into instructions & data sets such that when one instruction set is needed it is loaded in memory and after its execution is over, the space is released.
- As & when requirement for other instruction arises, it is loaded into space that was released previously by the instructions that are no longer needed.
- The program based on overlays scheme mainly consist of following:

  A root piece which is always memory resident

  Set of overlays

Overlays example



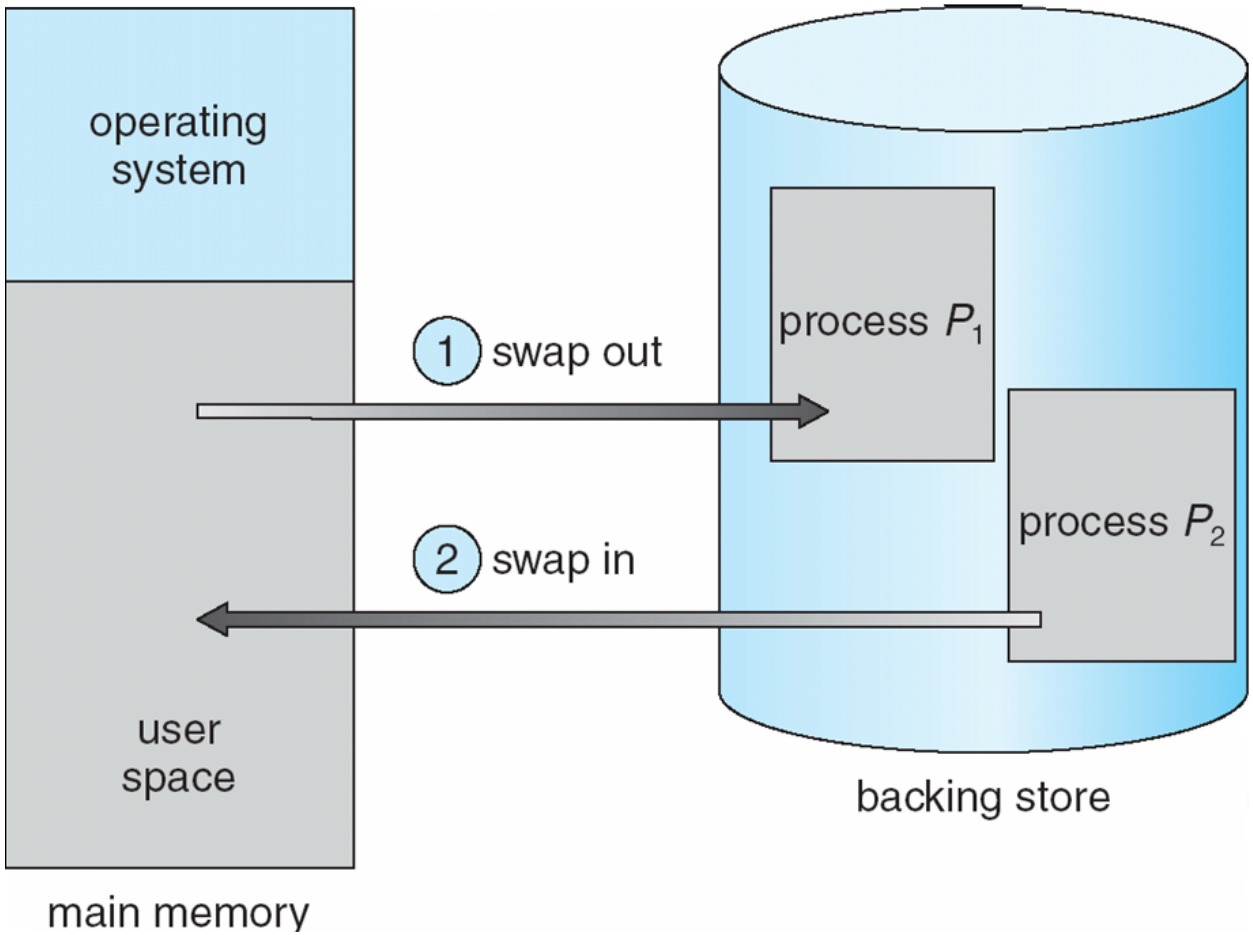
Overlays limitations

- Requires careful and time consuming planning
- Programmer is responsible for designing overlays structure of program
- OS provides the facility to load files into overlays region.

Swapping

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Swapping

- Benefits:

Allows higher degree of multiprogramming

Allow dynamic relocation

Better memory utilization

Less wastage of CPU time on compaction

Can easily be applied on priority based scheduling algorithms to improve performance

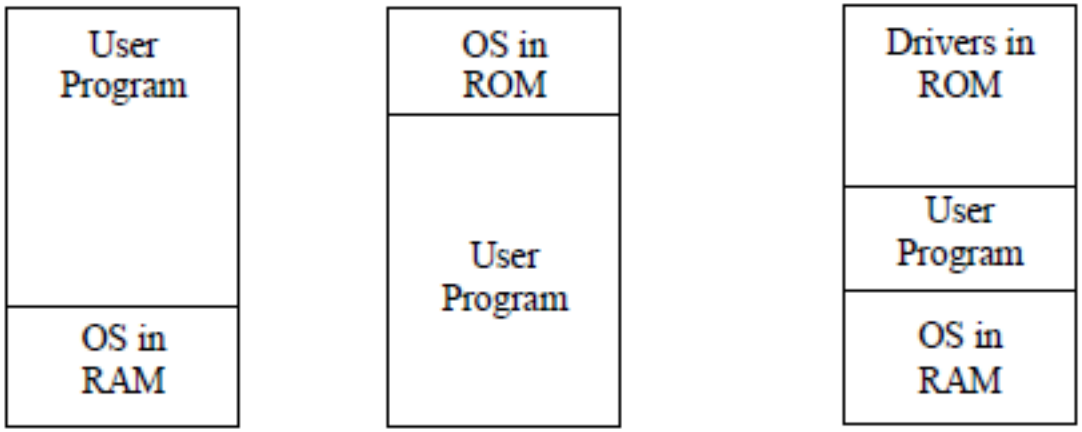
- Limitations:

Entire program must be resident in store when it is executing

Processes with changing memory requirements will need to issue system calls for requesting & releasing memory.

Single process monitor(Monoprogramming)

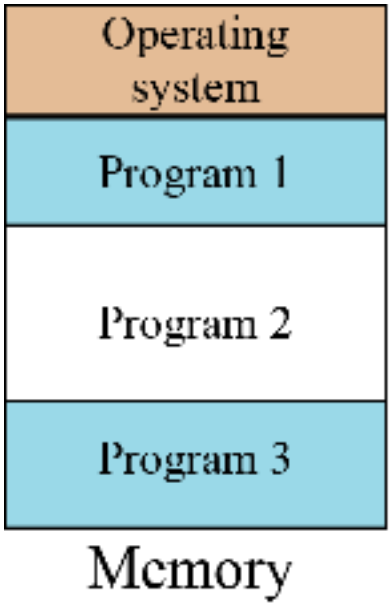
- Only one process in main memory
- No address translation during execution of program
- Protection of OS
- Ex: MS-DOS
- Limited capacity & performance



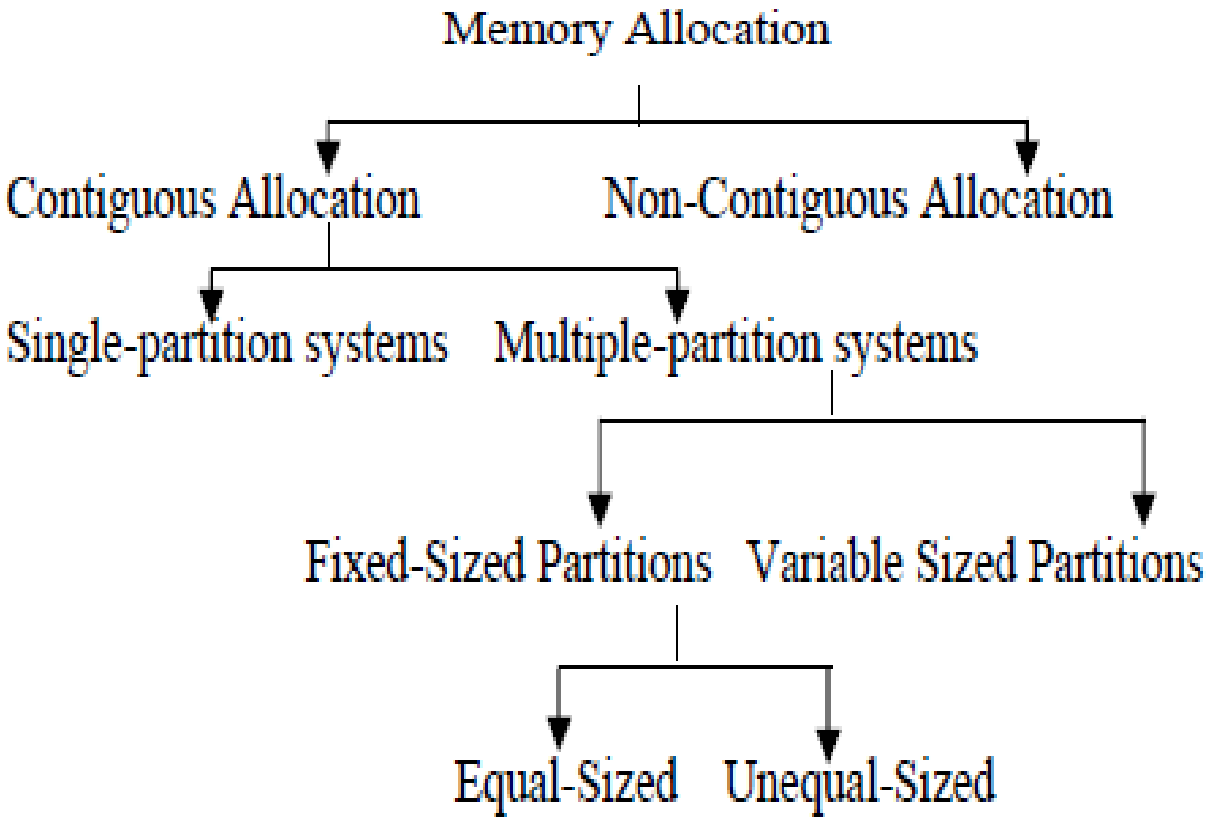
Contents

- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

Multiprogramming

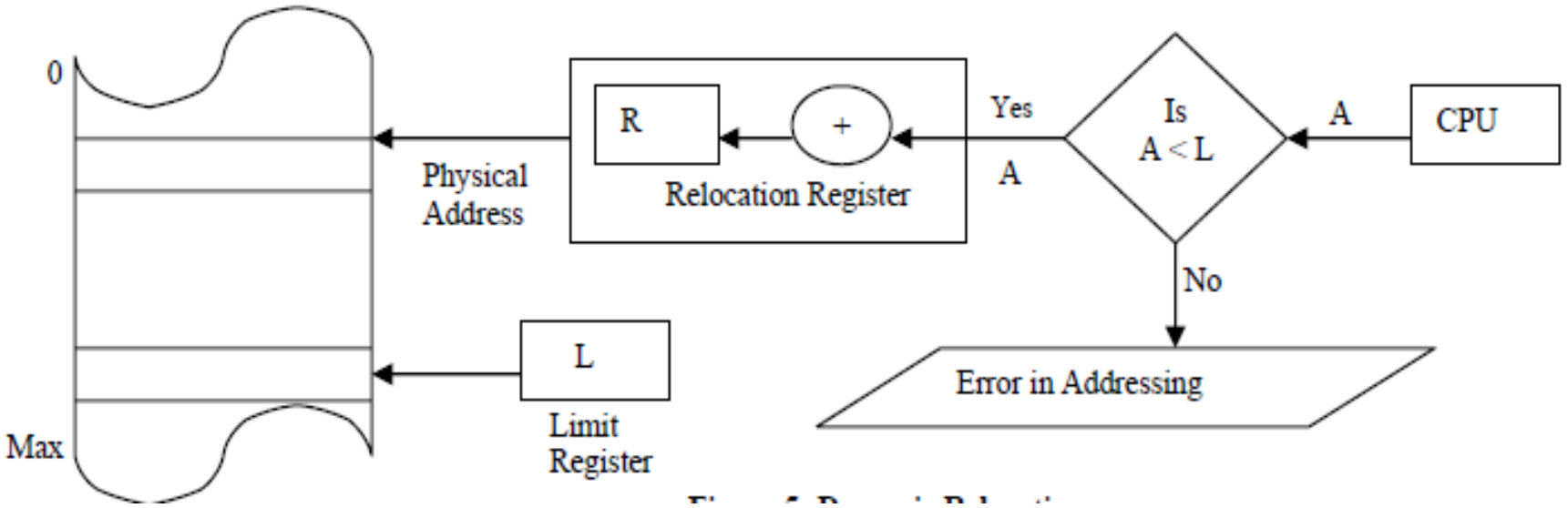


Memory Allocation Methods



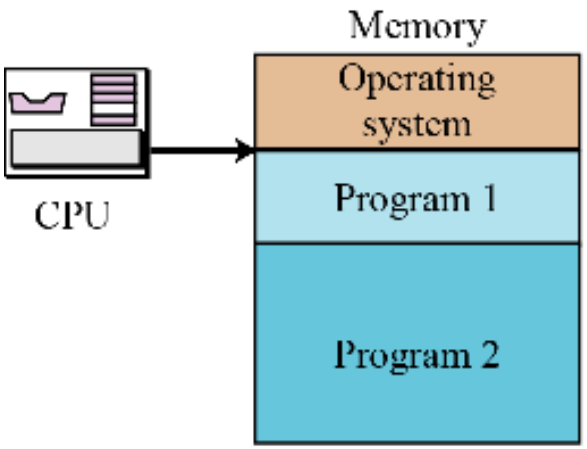
Single partition System

- OS can be protected by keeping it in lower part and user processes in upper part of memory
- Dynamic relocation is used using relocation register & base register
- Relocation register contains smallest physical address
- Limit register contains logical address range
- If logical address space is 0 to MAX then physical

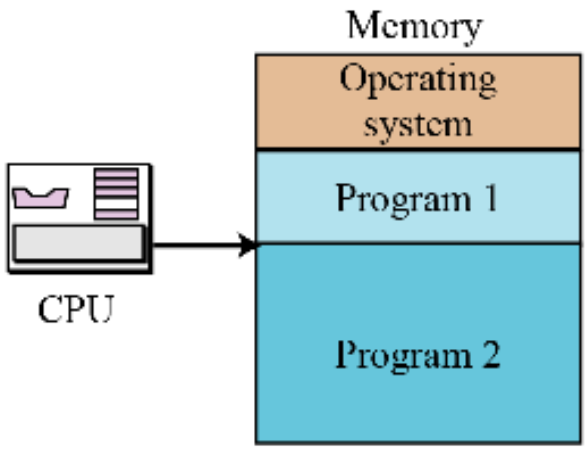


Memory Management

- Principal operation of MM is to bring process from secondary memory to main memory
- This typically involves Virtual Memory which in turn involves segmentation and/or paging
- Prior to Virtual Memory, other simpler techniques were used
  - Partitioning
    - Fixed
    - Variable / Dynamic
  - Concept of paging
  - Concept of segmentation



a. CPU starts executing program 1



b. CPU starts executing program 2

Contiguous Memory Partitioning

- Partition the available memory into regions with fixed boundaries.

    1. **Fixed Partitioning:**

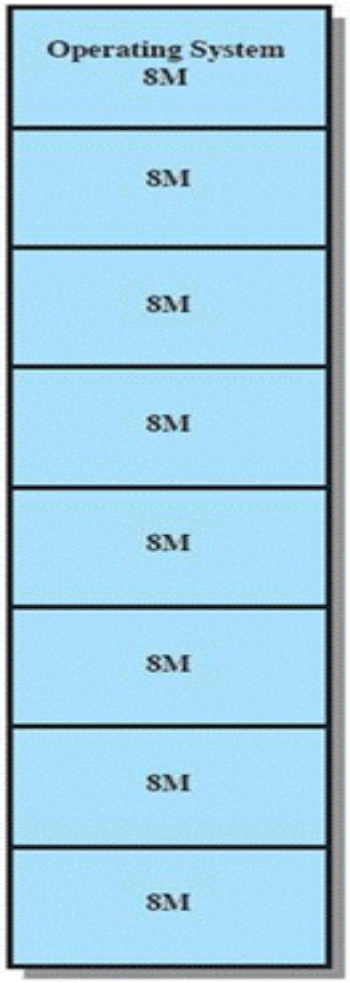
        Equal-size Partitions

        Unequal-size Partitions

    2. **Variable Partitioning**

Fixed Partitioning

- Equal-size partitions
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
  - If all partitions are full, the operating system can swap a process out of a partition



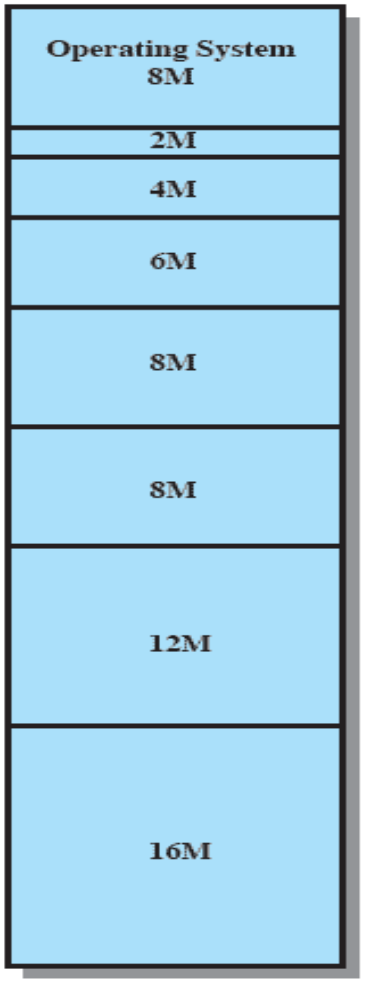
(a) Equal-size partitions

Fixed Partitioning

- Problems:
- A program may be too big to fit in any partition.
  - Use of overlays is compulsory.
- Main memory use is inefficient.
  - Any program, no matter how small, occupies an entire partition.
  - Wasted space internal to the partition called internal fragmentation

Solution: Unequal Size Partitions

- Lessens both problems
  - but doesn't solve completely
- In Figure
  - Programs up to 16M can be accommodated without overlay
  - Smaller programs can be placed in smaller partitions, reducing internal fragmentation



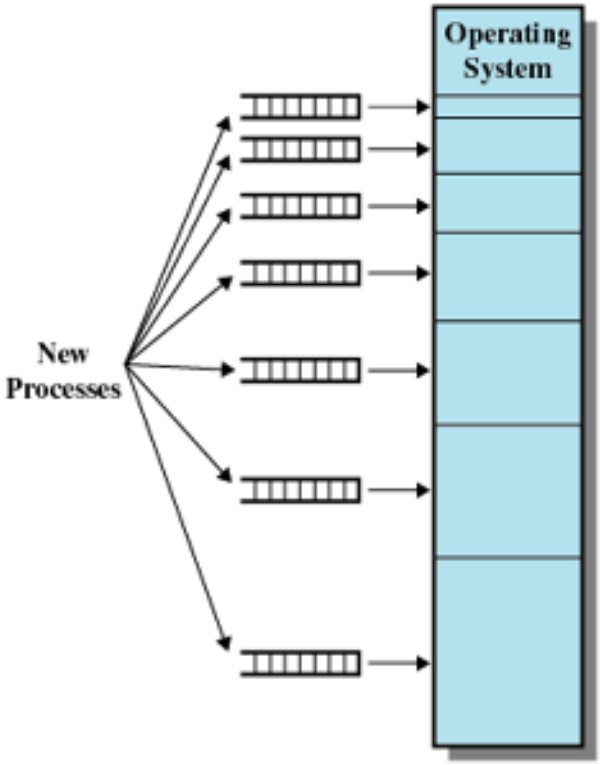
(b) Unequal-size partitions

Placement Algorithm with Partitions

- Equal-size partitions
  - Because all partitions are of equal size, it does not matter which partition is used.
  - If all partitions occupied with processes not ready to run, swap one out and load this guy in -Which one to swap out is a scheduling decision
- Unequal-size partitions
  - Can assign each process to the smallest partition within which it will fit.
  - Queue for each partition.
  - Processes are assigned in such a way as to minimize wasted memory within a partition.
  - Two possible strategies

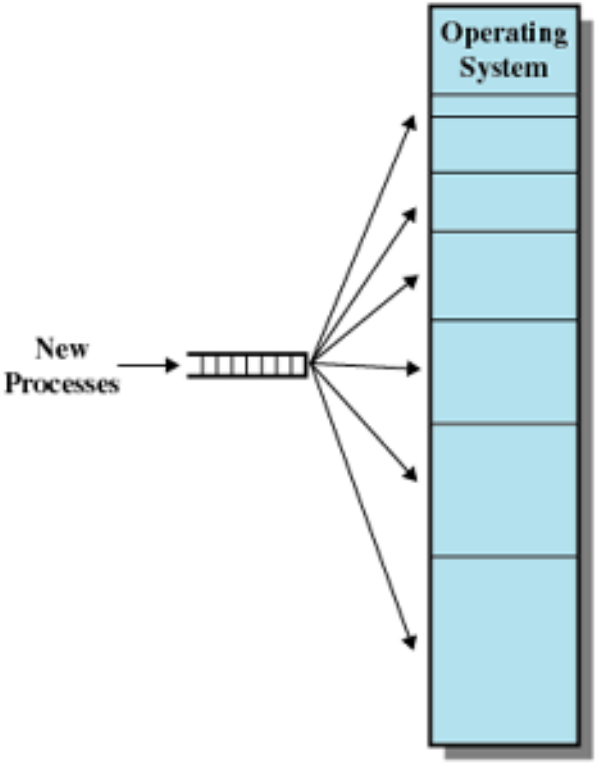
Strategy 1

- Assign to smallest partition possible
  - May have to wait because somebody else is already there
- Needs a scheduling queue for each partition to hold swapped out processes destined for that partition
- If memory requirement is not clear, overlaying or virtual memory are the only solutions
- Advantage: minimum internal fragmentation
- Disadvantage?
  - Small process cannot go to bigger partition even if available



Strategy 2

- Single queue
- Smallest available partition is selected when needed
- If all are occupied, then swap out somebody
  - Smallest partition process that is enough to hold this guy
  - Priority
  - Process state



Fixed Partitioning

- Advantages
  - Simple, require minimal OS and processing overhead
- Disadvantages
  - Number of partitions specified at system generation time limits the number of active processes system can support
  - Internal fragmentation cannot be completely eliminated
    - It is always possible to get small jobs which do not utilize partitions fully
- Fixed partitioning is no where to be seen today
- Example: IBM Mainframe's OS/MFT(Multiprogramming with fixed number of tasks)

Dynamic Partitioning

- Purpose:to overcome the difficulties of fixed partitioning
- Partitions are of variable length and number.
- Process is allocated exactly as much memory as required.
- Eventually get holes in the memory. This is called external fragmentation
- Example:I BM Mainframe's OS/MVT OS/MFT(Multiprogramming with variable number of tasks)

Dynamic Partitioning Example

- ***External Fragmentation***
- Memory external to all processes is fragmented
- Can resolve using **coalescing holes** & ***compaction***

Dynamic Partitioning Example

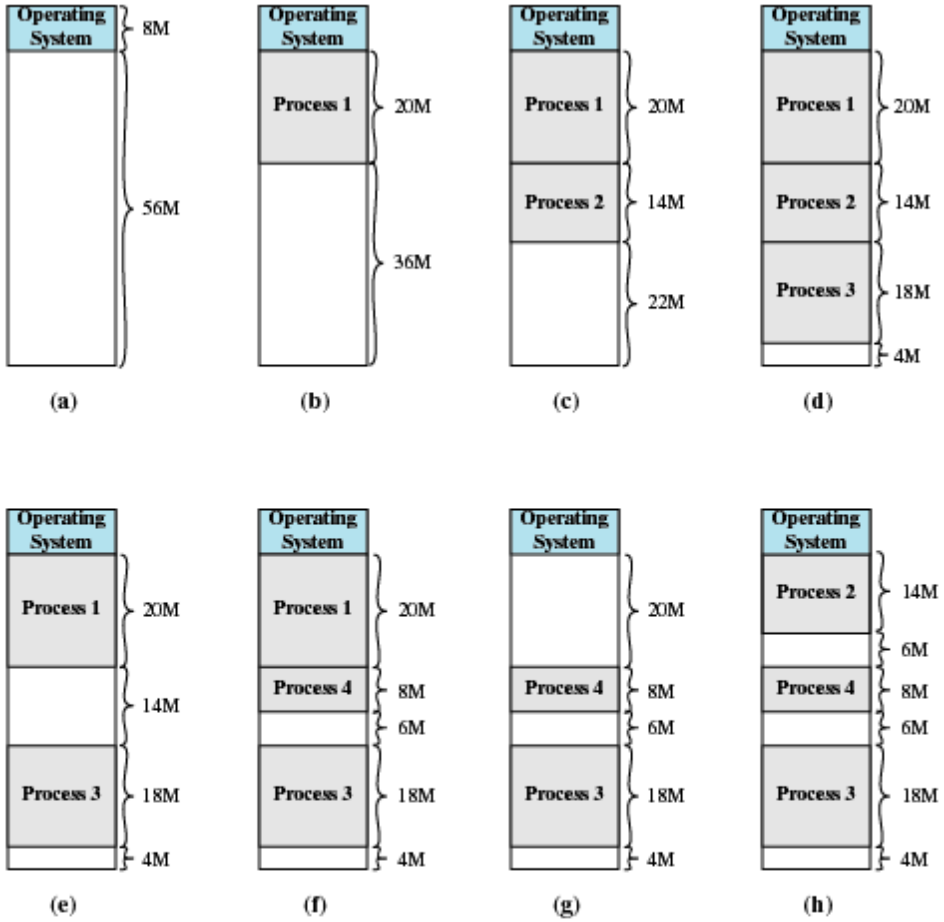


Figure 7.4 The Effect of Dynamic Partitioning

Dynamic Partitioning

- Coalescing holes is the process of merging existing hole adjacent to a process that will terminate and free its allocation space. So that new adjacent hole & existing holes can be viewed as a single large hole and can be efficiently used.
- In Compaction, shuffle all occupied areas of memory to one end and leave all free memory space as a large block

- Problem with compaction?
  - Extra overhead in terms of resource utilization & large response time
  - Expensive
  - Needs dynamic relocation of processes in memory

Contents

- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.
- OS Services layer in the Mobile OS: Multimedia and Graphics Services, Connectivity Services.

Dynamic Partitioning Placement Algorithm

- Purpose: To overcome on problem of compaction
- Operating system must decide which free block to allocate to a process
- When more than one choice available, OS must decide cleverly which hole to fill
- Hole must be big enough to accommodate the to-be-loaded process
- Placement algorithms:

First fit

Best fit

3. Next fit
4. Worst fit

Dynamic Partitioning Placement Algorithm

- First-fit algorithm
  - Scans memory from the beginning and chooses the first available block that is large enough
  - Simplest, Fastest
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Dynamic Partitioning Placement Algorithm

Best-fit algorithm

- scan all holes to see which is best
- Chooses the block that is closest in size to the request
- Worst performer overall
- Since smallest block is found for process, the smallest amount of fragmentation is left
- Memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

- Next-fit
  - Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

- The last block that was used was a 22-Mbyte block from which a 14-Mbyte partition was created.

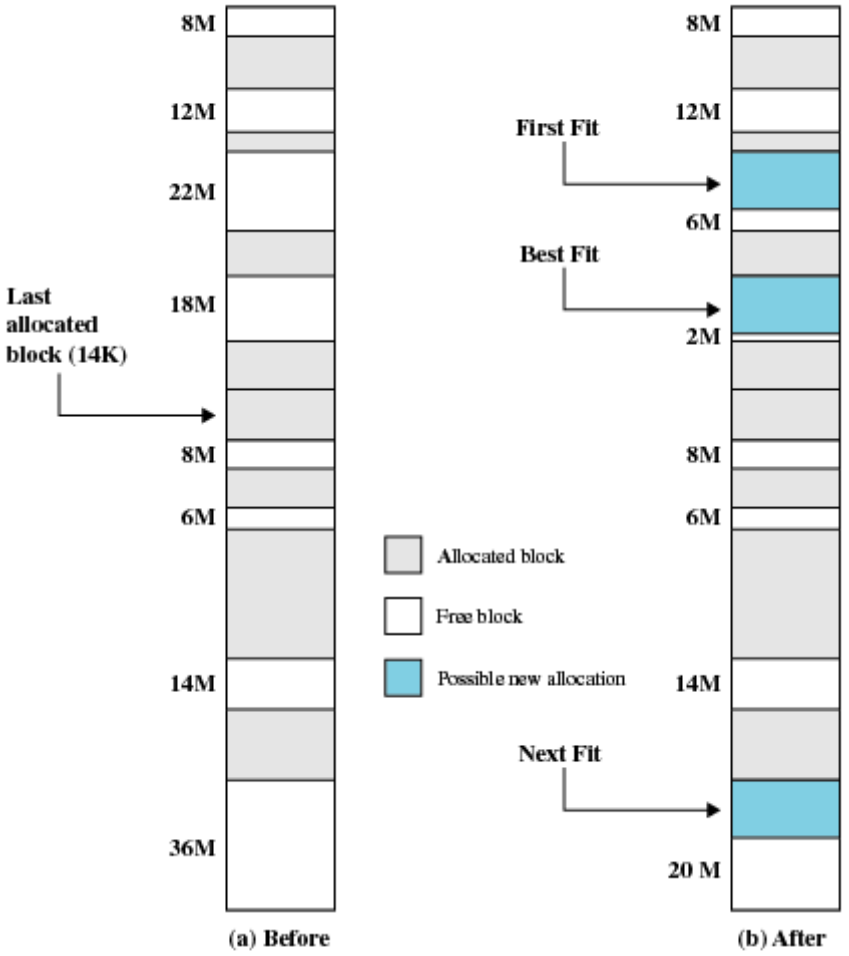


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

Comparison

- Depends on exact sequence of process swapping and size of processes
- First fit
  - Simplest and usually the best and fastest
  - Splits regions towards beginning requiring more searches
- Next fit produces slightly worse results than first fit
  - Tends to allocate more frequently towards the end of the memory, thus largest block of free memory which usually appears at the end is quickly fragmented requiring more frequent compaction
- Best fit is usually the worst performer
  - Every allocation leaves behind smallest fragment of no good use
  - Requires compaction even more frequently
- How about worst-fit strategy?
- Worst-fit: Allocate the largest hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB. How would each of the First fit, Best-Fit and Worst-Fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB ?

2. Given memory partition of 100 KB, 500 KB, 200 KB and 600 KB (in order). Show with neat sketch how would each of the first-fit, best-fit and worst fit algorithms place processes of 412 KB, 317 KB, 112 KB and 326 KB (in order). Which algorithm is most efficient in memory allocation?

Fragmentation

- **External Fragmentation** – Total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference in memory is internal to a partition, but not being used.
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers

Buddy System

- To overcome on the drawbacks of Fixed & Variable partitioning scheme.
- Memory blocks are available of size 2^K words, where L<=K<=U
- 2^L Smallest size block that is allocated
- 2^U largest block that is allocated
- Generally 2U is the size of entire memory available for allocation
- Entire space available is treated as a single block of size 2^U
- If a request of size *s* where $2^{U-1} < s <= 2^U$
  - entire block is allocated
- Otherwise block is split into two equal buddies of size 2^{U-1}.
  - Process continues until smallest block greater than or equal to *s* is generated

Buddy System

- The buddy system of partitioning relies on the fact that space allocations can be conveniently handled in sizes of power of 2.
- There are two ways in which the buddy system allocates space.
- Suppose we have a hole which is the closest power of two. In that case, that hole is used for allocation.
- In case we do not have that situation then we look for the next power of 2 hole size, split it in two equal halves and allocate one of these.
- Because we always split the holes in two equal sizes, the two are “buddies”. Hence, the name buddy system.
- The buddy system has the advantage that it minimizes the internal fragmentation.
- In practice, some Linux flavors use it.

Example of Buddy System

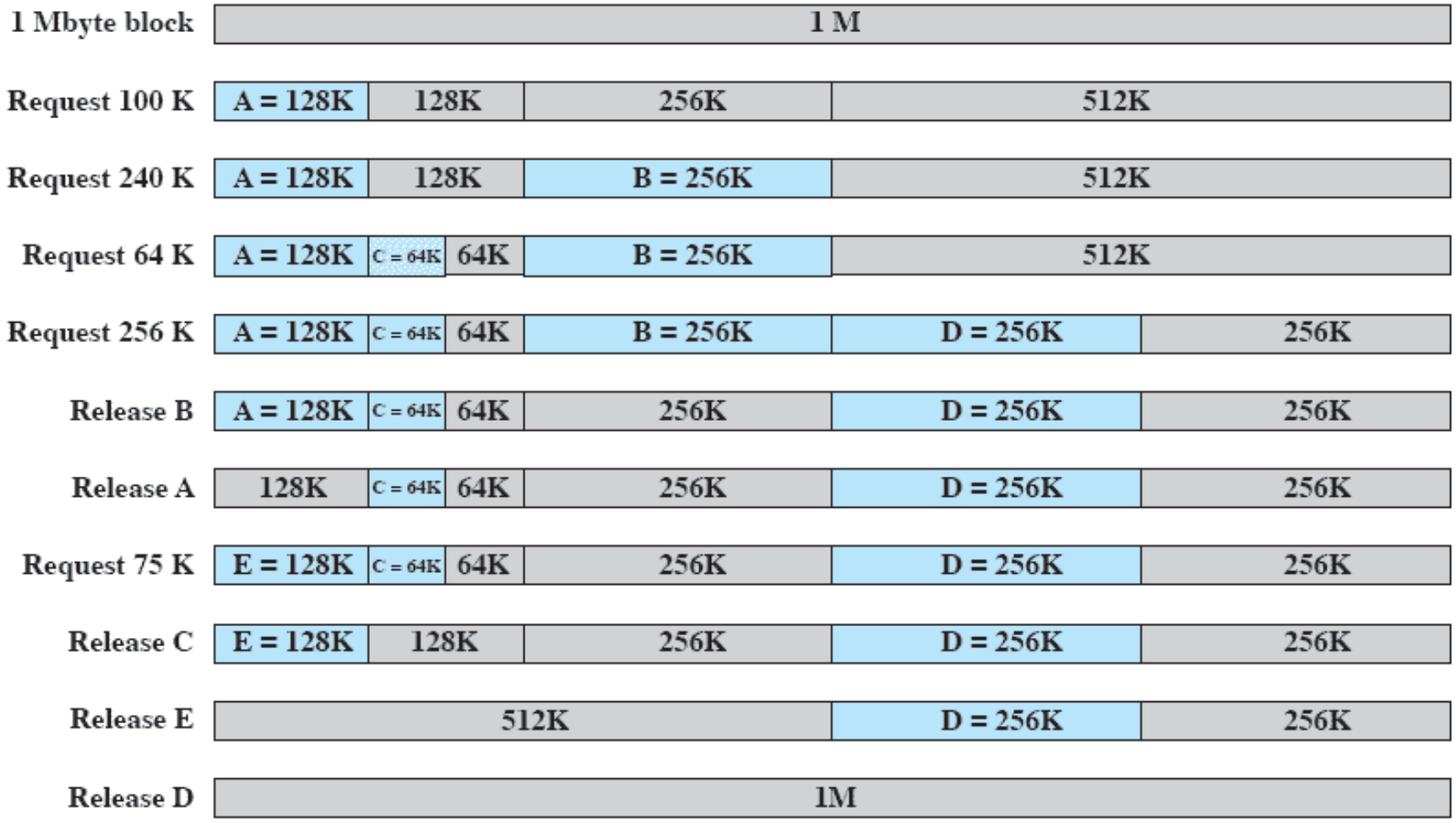


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

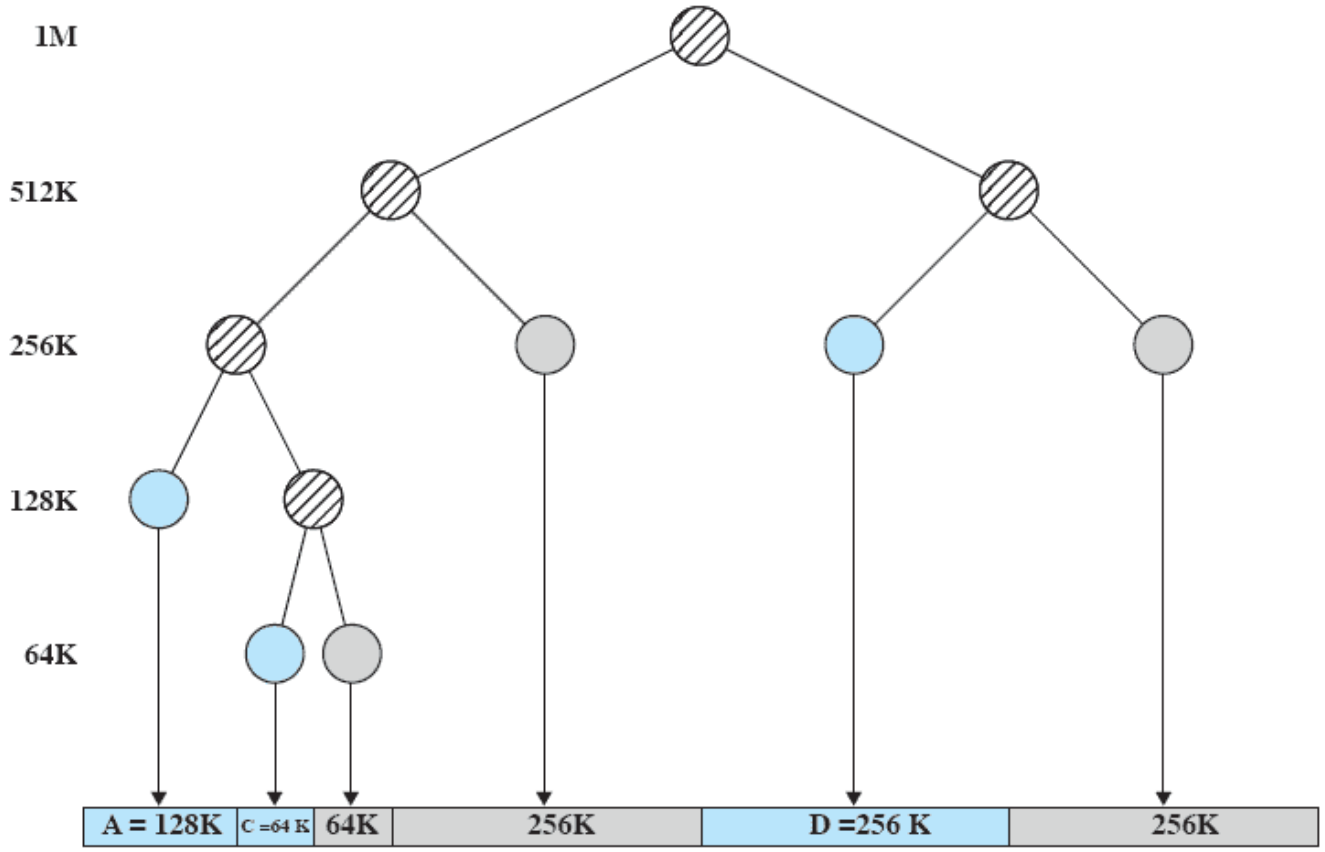


Figure 7.7 Tree Representation of Buddy System

Tree Representation of Buddy System

- Figure 7.7 shows a binary tree representation of the buddy allocation immediately after the Release B request.
- The leaf nodes represent the current partitioning the memory.
- If two buddies are leaf nodes, **then at least one must be allocated;**
  - otherwise they would be coalesced into a larger block.
- The buddy system is a reasonable compromise to overcome the disadvantages of both the fixed and variable partitioning schemes,
- But in contemporary operating systems, virtual memory based on paging and segmentation is superior.
- However, the buddy system has found application in parallel systems as an efficient means of allocation and release for parallel programs. A modified form of the buddy system is used for UNIX kernel memory allocation

Example on Buddy system

- A minicomputer uses the buddy system for memory management. Initially it has one block of 256K at address 0. After successive requests of 7K, 26K, 34K and 19K come in, how many blocks are left and what are their sizes and addresses?

Example on Buddy system

- Solution:
- 7K: We recursively break the address space into 2 halves until we have:

  8K - 8K - 16K - 32K - 64K -128 K
- The first segment is used to satisfy the 7K request.
- 26K: We use the 32K block to allocate the request.

  8K - 8K - 16K - 32K - 64K -128 K
- 34K: We use the 64K block to satisfy the request:

  8K - 8K - 16K - 32K - 64K -128 K
- 19K: Since 8K and 16K cannot satisfy the request on their own, we need to break the big
- 128K block recursively until we get the size we need. The blocks will look like:
- 8K - 8K - 16K - 32K - 64K - 32K - 32K - 64K

Example on Buddy system

- A 1MB block of memory is allocated using the buddy system.Show the result of the following sequence:Request A 70,request B 35,request C 80, return A,request D 60,return B,return D,return C.
- Show the binary tree representation following Return B

Contents

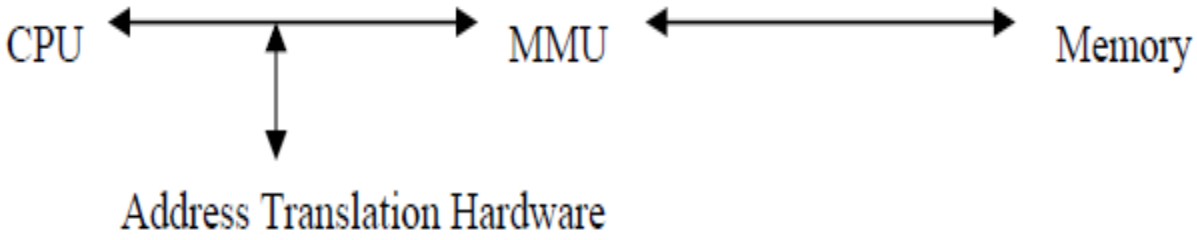
- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Paging, Segmentation, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

Addresses

- Logical
  - Reference to a memory location independent of the current assignment of data/instruction to memory. Generated by CPU
- Relative
  - Type of logical address wherein address is specified as a location relative to some known point, say a value in a register
- Physical or Absolute
  - The absolute address or actual location in main memory.
- Typically, all of the memory references in a loaded process are relative to the base address
  - Hardware mechanism is used to translate logical/relative to physical at the time of execution of instruction that contains the reference

Memory Management Unit

- MMU is the hardware device that maps logical address to physical address



Relocation

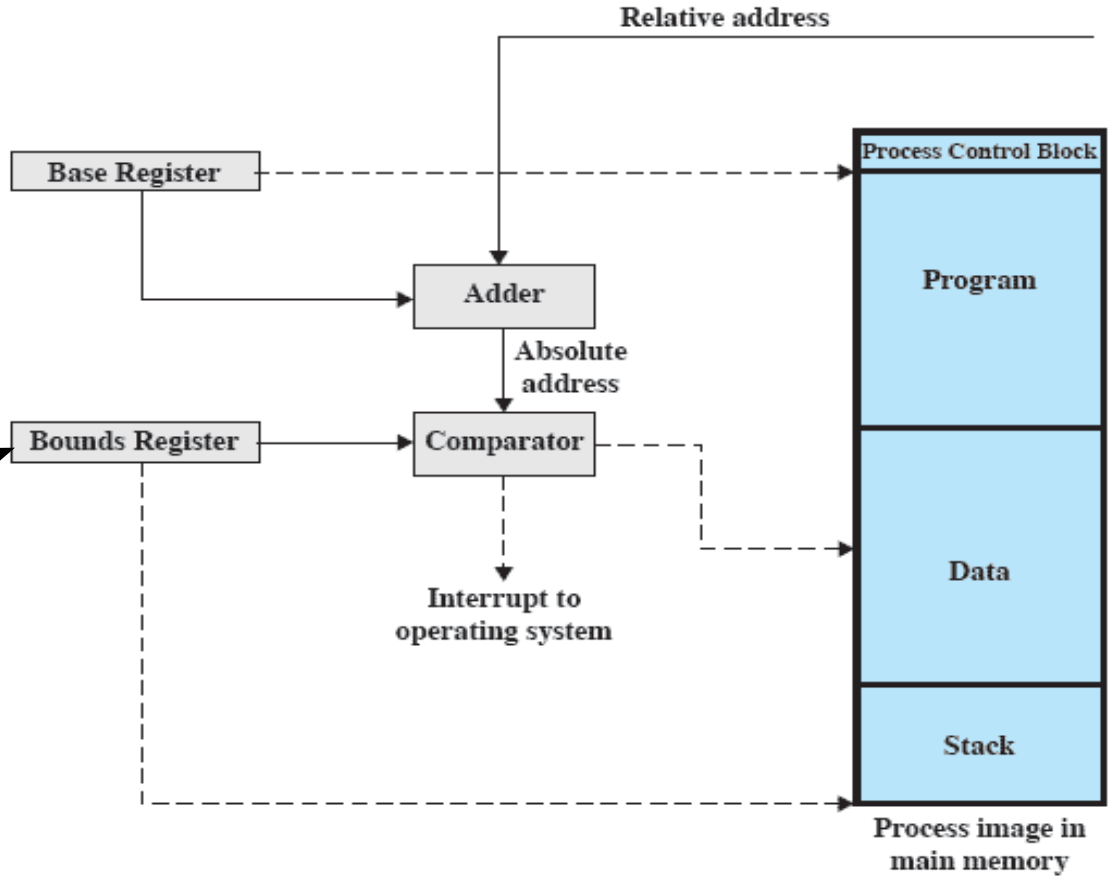


Figure 7.8 Hardware Support for Relocation

Registers Used during Execution

- Base register: Starting address for the process.
- Bounds register: Ending location of the process.
- These values are set when the process is loaded or when the process is swapped in.
- The value of the base register is added to a relative address to produce an absolute address.
- The resulting address is compared with the value in the bounds register.
- If the address is not within bounds, an interrupt is generated to the operating system otherwise instruction gets executed.

Paging

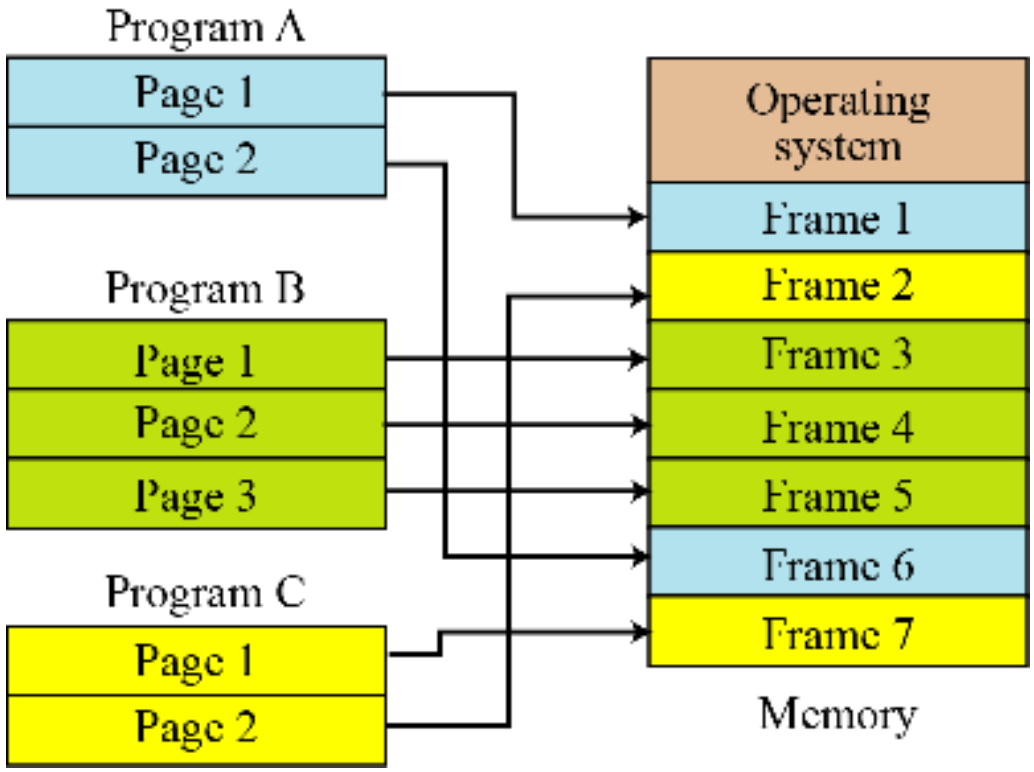
- Fixed & variable size partitions are insufficient as involves internal / external fragmentation. contiguous
- What were the two problems with equal sized fixed partitions?
  - Program too large for a partition
  - Program too small for a partition
- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks.
- The chunks of a process are called pages and chunks of memory are called frames.
- Operating system maintains a page table for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page
  - Page number is used as an index to page table.

Paging

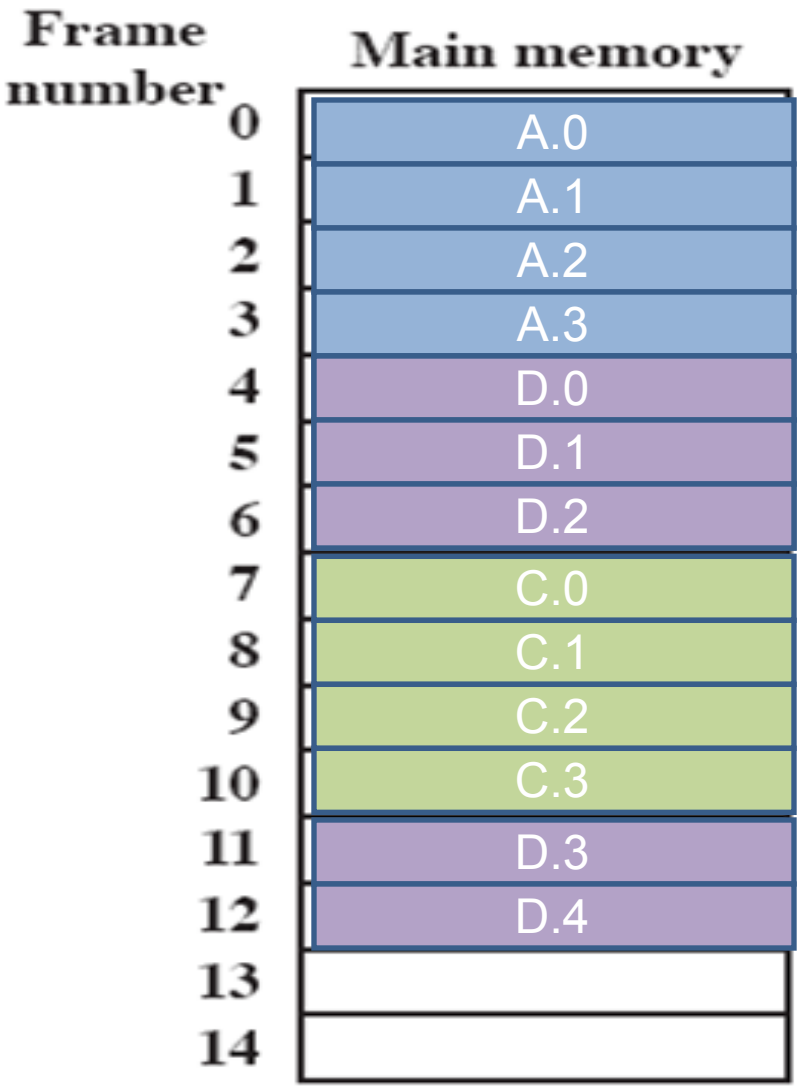
- Each process has its own page table.
- Each page table entry contains the frame number of the corresponding page in main memory
- A bit is needed to indicate whether the page is in main memory or not.
- No external fragmentation
- all frames (physical memory) can be used by processes
- Possibility of Internal fragmentation
- The physical memory used by a process is no longer contiguous
- The logical memory of a process is still contiguous
- The logical and physical addresses are separated
- the process does not see the translation or the difference to having physical memory

Advantages of Breaking up a Process

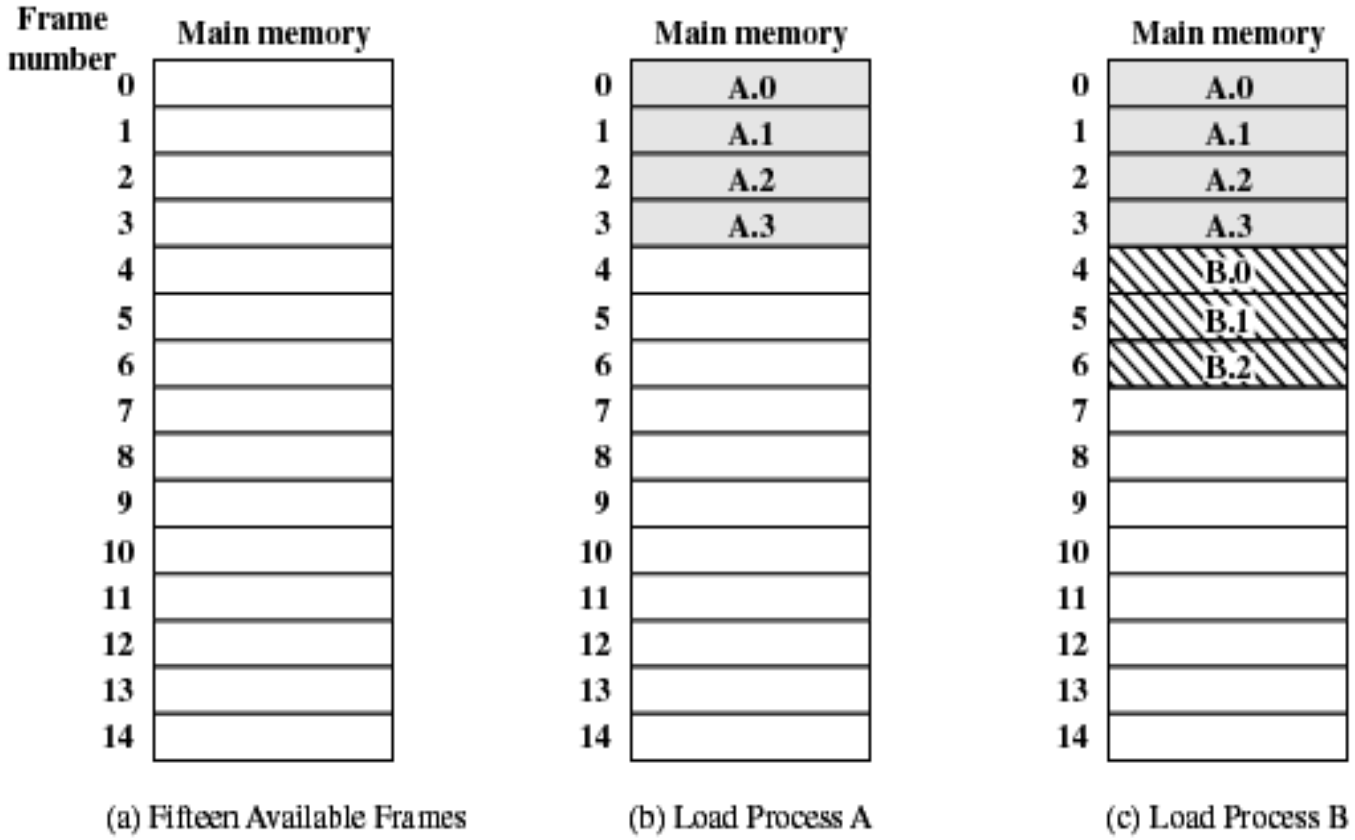
- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory


Program A
Page 1
Page 2
Program B
Page 1
Page 2
Page 3
Program C
Page 1
Page 2
Operating system
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Memory

Processes and Frames



Assignment of Process Pages to Free Frames



Assignment of Process Pages to Free Frames

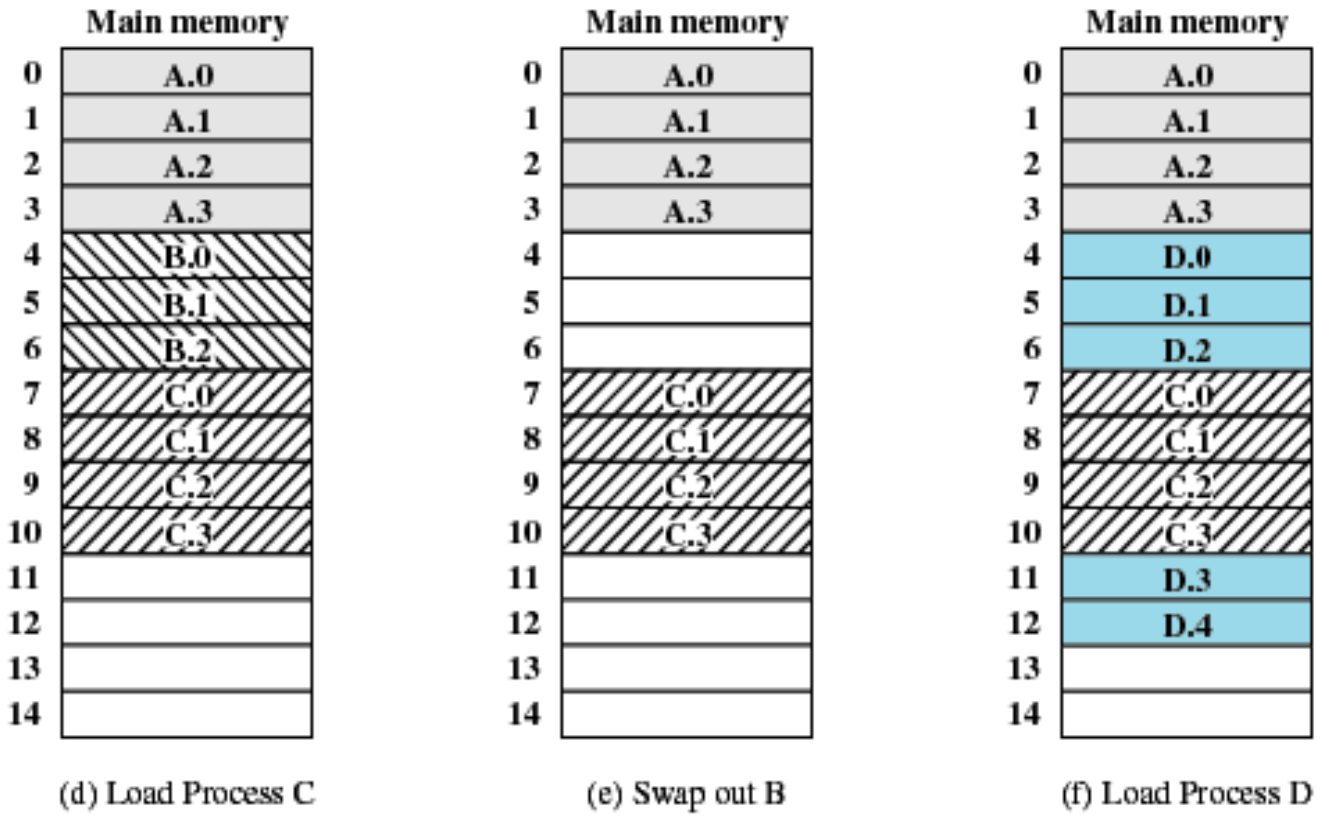


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

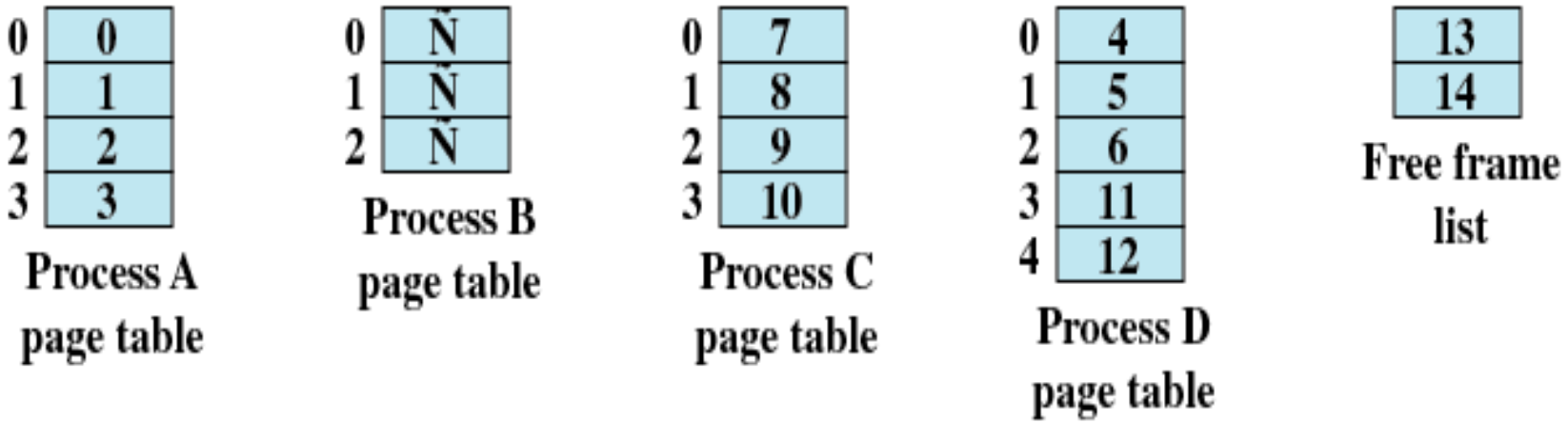
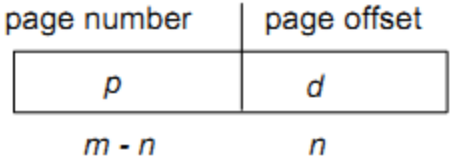


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging

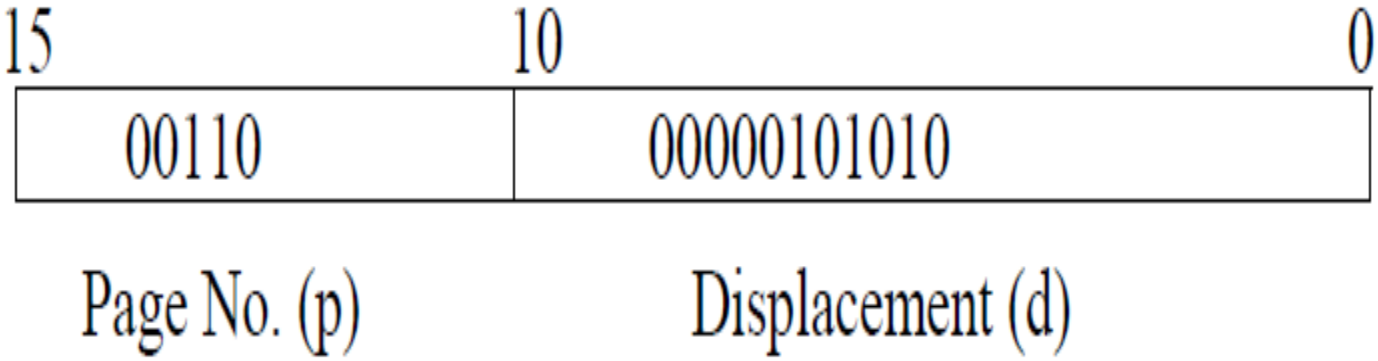
- The page size is defined by the hardware.
- The page size is typically a power of 2.The size can be 512 bytes to 16 MB.
- The selection of power of 2 as a page size makes translation of logical address into a page number and page offset easy.
- Address generated by CPU is divided into:
- Page number (p) – used as an index into a page table which contains base address of each page in physical memory
- Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit.

For given logical address space 2^m and page size 2^n



Paging

Assume 16 bit address

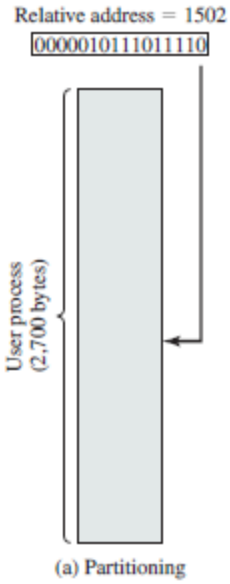


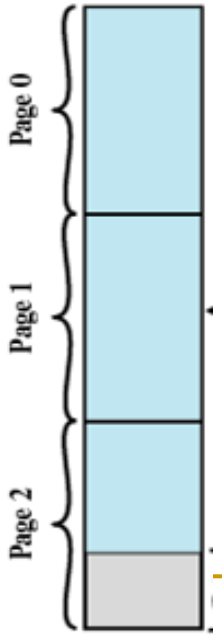
Here, as page number takes 5bits, so range of values is 0 to 31(i.e. 2^5-1). Similarly, offset value uses 11-bits, so range is 0 to 2023(i.e., 2^{11}–1). Summarizing this we can say paging scheme uses 32 pages, each with 2024 locations.

Logical Address vs Relative Address

- What does relative address 1502 mean?
  - Start from base relative address of the process
- What logical address would this correspond to?
  - Let page size be 1000 bytes
  - Logical address = page 1, offset 502
- That is, relative address 1502 corresponds to logical address (1,502)
- Expressed as 16 bit values, relative address is 0000010111011110 and

  logical address is (000001, 0111100110)
- **They are different**

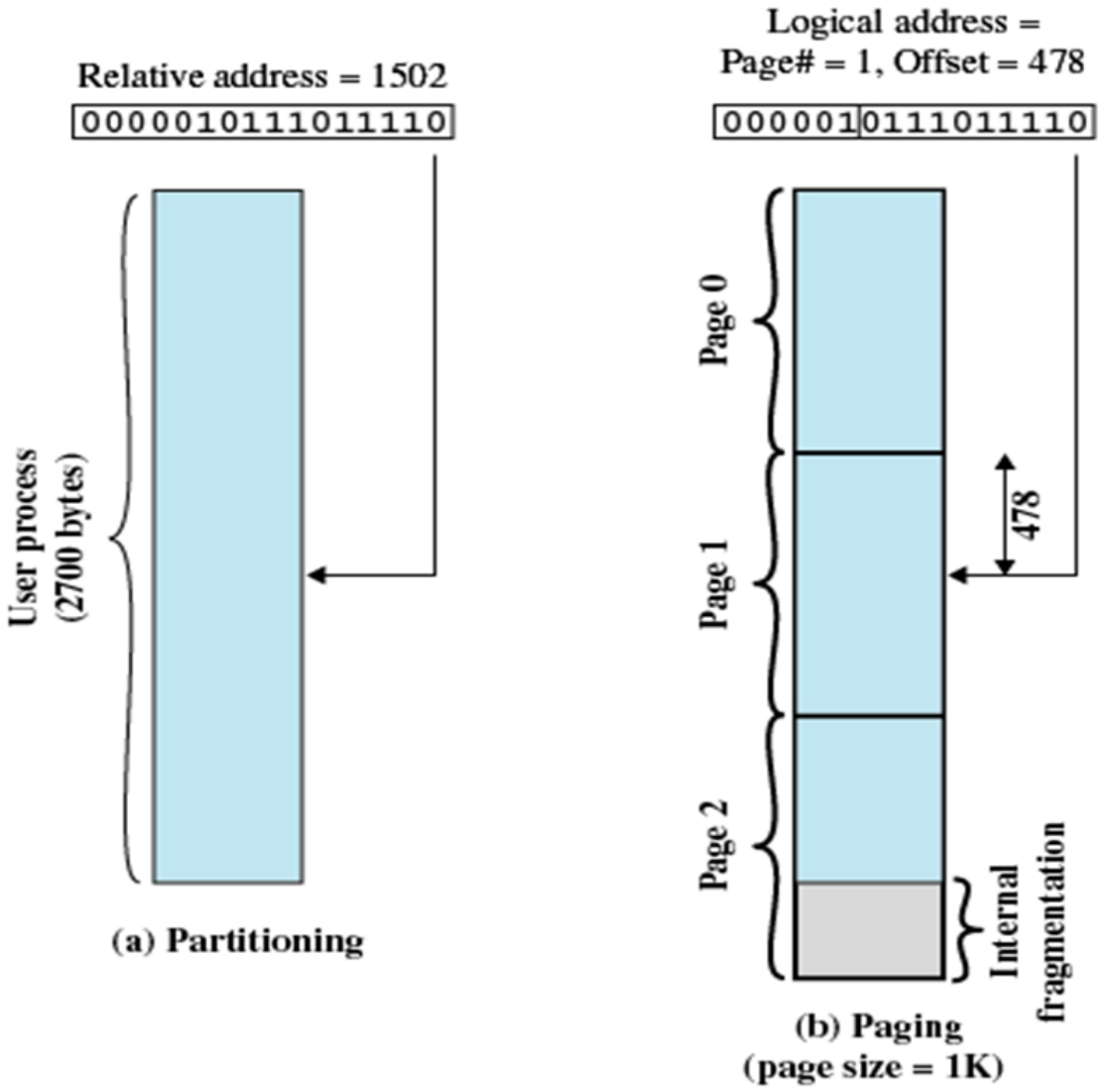




- In this example, 16-bit addresses are used, and the page size is 1K =1024 bytes=2^{10}
- The relative address 1502, in binary form, is 0000010111011110.
- From page size, we understand how many bits the offset would need
- Remaining bits are then available for page number
- Logical address simply corresponds to the two portions of relative address
- With a page size of 1K, an offset field of 10 bits is needed, leaving 6 bits for the page number.
- Can you compute maximum number of pages a program can contain?
  - 2^6 =64 pages of 1K bytes each.
- As Figure 7.11b shows, relative address 1502 corresponds to

an offset of 478 (0111011110) on page 1 (000001),

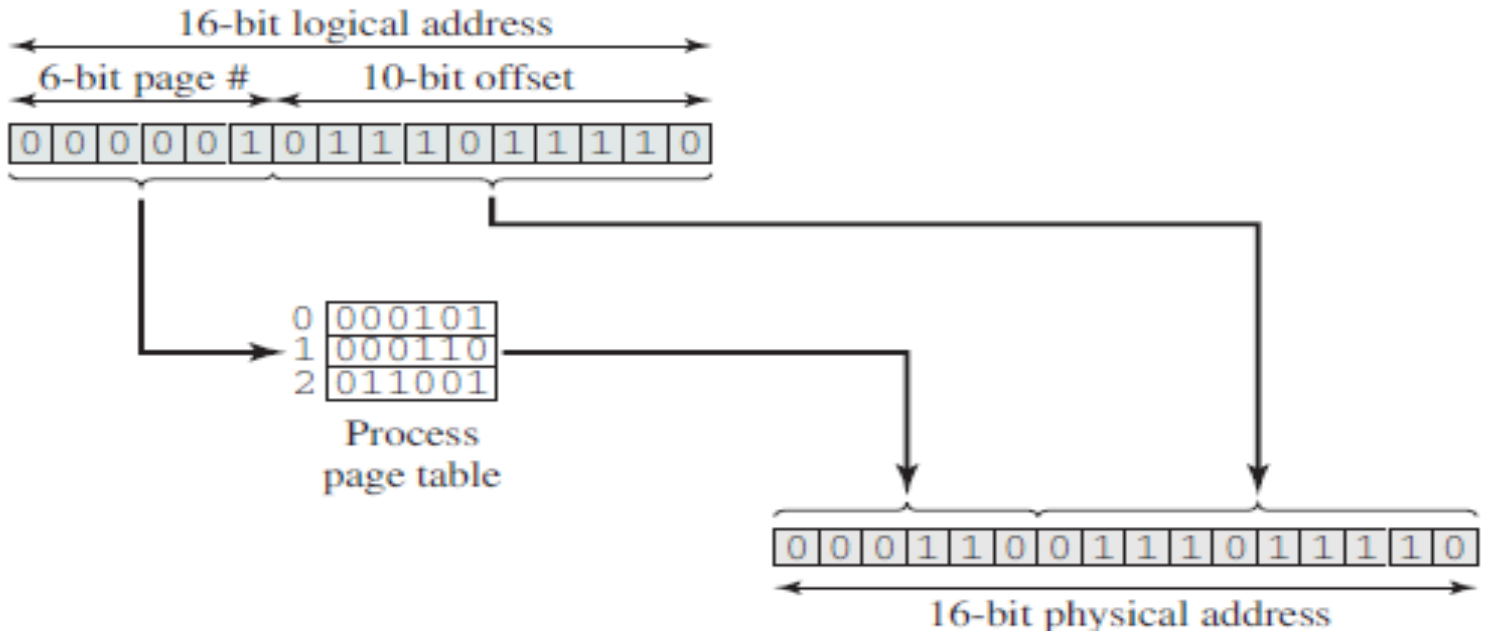
Illustration



Logical to physical address translation

- Consider a logical address of n+m bits
  - First n bits = page number
  - Last m bits = offset
- Use the page number as an index into process page table to find frame number, k
- Starting physical address of that frame then would be k X 2^m
- Desired physical address = this + offset, m
  - Again, doesn't need to be calculated
  - Just append offset to the frame number k

Logical to Physical Address Translation in Paging



The logical address is 000001**0**111011110, which is page number 1, offset 478.

Suppose that this page is residing in main memory frame 6 = binary 000110. Then the physical address is frame number 6, offset 478 = 000110111011110

(6*2^10=6144+478=6622 (000110111011110)

Example

- Consider a logical address space of 64 pages of 1024 words each, mapped onto a physical memory of 32 frames.

How many bits are there in the logical address?

Sol:64 pages=64*1024=2^6*2^10=2^16=16 bits

How many bits are there in the physical address?

Sol: 32 frames= 32 *1024=2^5*2^10=2^15=15 bits

Example

- Consider a logical address space of 32 pages of 1024 words per page, mapped onto a physical memory of 16 frames.

How many bits are there in the logical address?

Sol:32 pages=32*1024=2^5*2^10=2^15=15 bits

b) How many bits are there in the physical address?

Sol:16 pages=16*1024=2^4*2^10=2^14=14 bits

Example

- Consider a computer system with a 32 bit logical address and 4KB page size. The system supports up to 512 MB of physical memory. How many entries are there in a conventional page table?
- Logical address =32 bit
- Page size=4KB=4*1024=2^2*10^2=10^12
- page number =32-12=20 bits, offset=12 bits
- Physical address=512 MB=512*1024*1024=2^9*2^20=2^29
- Page table entry=total frames=2^20

Example

Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of logical address space.

a. How many bits are in a logical address? ⟶ 26 BITS

b. How many bytes in a frame? ⟶ 2^10 bytes:1Kb

c. How many bits in the physical address specify the frame? ⟶ 22 BITS

d. How many entries in the page table? ⟶ 2^16

e. How many bits in each page table entry? Assume each page table entry contains a valid/invalid bit. ⟶ 22+1 BITS

Solve same by replacing 2^ 32 bytes with 2^32 frames of physical memory

Consider a simple paging system with the following parameters: 2^{32} bytes of physical memory; page size of 2^{10} bytes; 2^{16} pages of logical address space.
How many bits are in a logical address?

b. How many bytes in a frame?

c. How many bits in the physical address specify the frame?

d. How many entries in the page table?

e. How many bits in each page table entry? Assume each page table entry contains a valid/invalid bit.

Segmentation

- User program and associated data now divided not into pages, but segments which could be of unequal size
- All segments of all programs do not have to be of the same length, May be unequal, dynamic size
- There is a maximum segment length, Simplifies handling of growing data structures
- Logical Address consist of two parts - a segment number and an offset
- Similar to dynamic partitioning

A program however can now occupy more than one partitions
Partitions need not be contiguous

- No internal fragmentation?
- No external fragmentation?

Segmentation

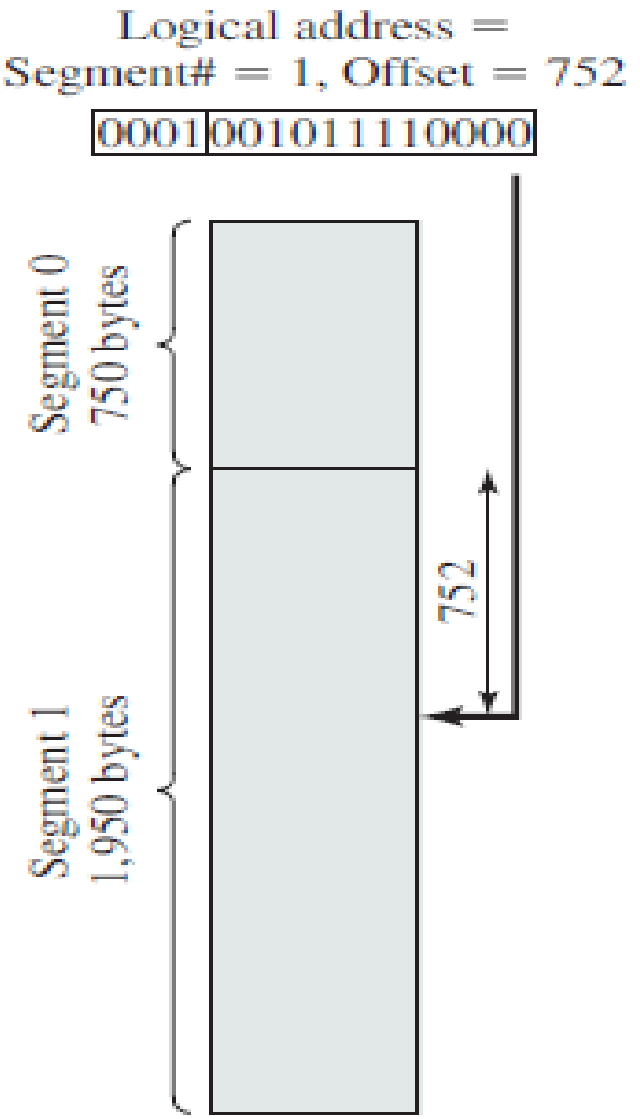
- Paging is invisible to the programmer, segmentation is usually visible and is provided as a convenience for organizing programs and data
  - Compiler or programmer assigns programs and data to different segments
  - One program may be further broken down into multiple segments for purposes of modular programming
- Allows programs to be altered and recompiled independently
- Lends itself well to sharing data among processes
- Lends itself well to protection
- Simplifies handling of growing data structures
- Logical address to physical address translation is now little complicated but similar
  - Segment table
    - Length of segment and starting physical address

Segment Organization

- Starting address corresponding segment in main memory
- Each entry contains the length of the segment
- A bit is needed to determine if segment is already in main memory
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory

Logical to Physical Address Translation in Segmentation

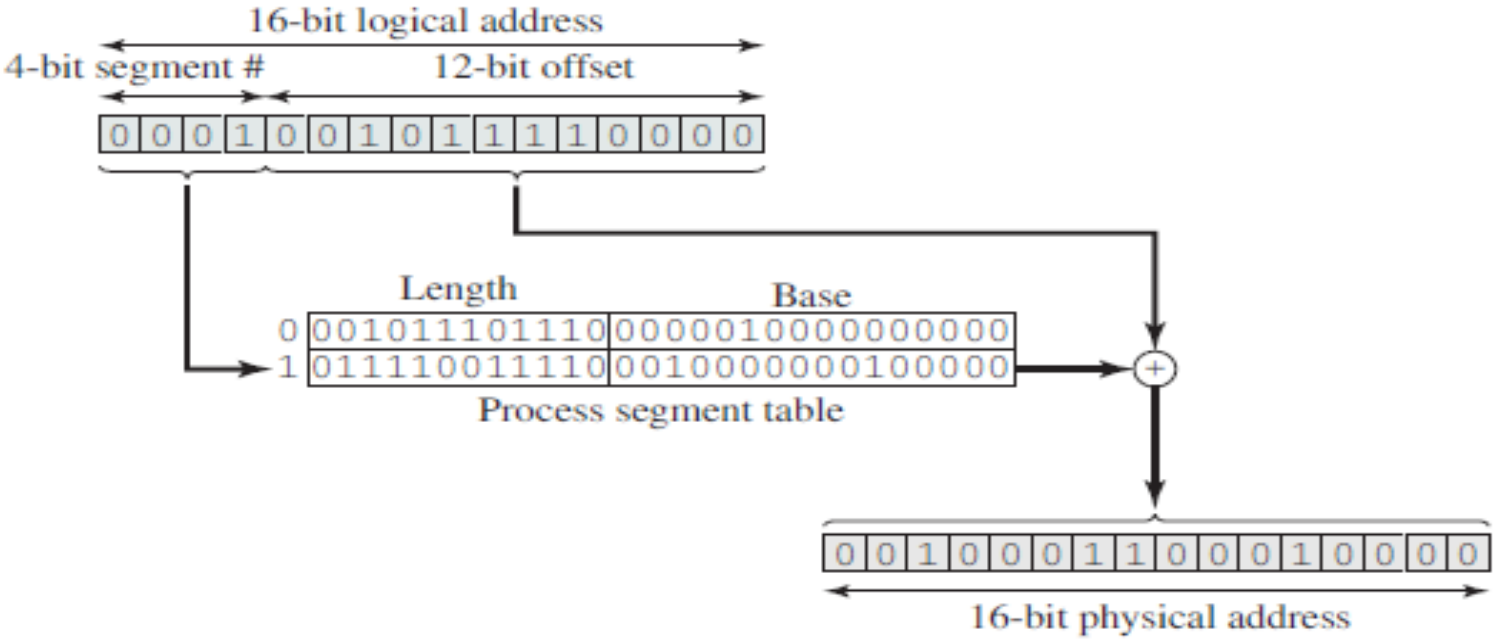
- In this example, 16-bit addresses are used,
- The relative address 1502, in binary form, is 0000010111011110
- Consider an address of n + m bits where the leftmost n bits are the segment number and the rightmost m bits are the offset.
- In the example on the slide n = 4 And m =12.



Address translation

- Extract segment number from logical address (left most n bits)
- Find base address of this segment from segment table
- Compare offset (rightmost m bits) with segment length
- If ok, desired physical address = base address + offset

Logical to Physical Address Translation in Segmentation



The logical address is 0001001011110000, which is segment number 1, offset 752.

Suppose that this segment is residing in main memory starting at physical address 0010000000100000.

Then the physical address(base addess+offset):
0010000000100000 +

Examples

Consider a simple segmentation system that has the following segment table:

Starting Address	Length (bytes)
660	248
1,752	422
222	198
996	604

For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:

a. 0, 198
b. 2, 156
c. 1, 530
d. 3, 444
e. 0, 222

Segment no	Start	Length	End (start+length)
S2	222	198	420
S0	660	248	908
S3	996	604	1600
S1	1752	422	2174

- (0,198): 660+198=858
- (2,156)=222+156=377
- (1,530)=invalid,as offset > length
- (3,444)=996+444=1440
- (0,222)=Invalid as offset > length

Example

- 8-bit virtual address, 10-bit physical address, and each page is 64 bytes.
- How many virtual pages? 4 pages
- How many physical pages? 16 frames
- How many entries in page table? 4 PTE
- Given page table = [2, 5, 1, 8], what's the physical address for virtual address 241? 561

Example

Given memory partitions of 100 KB, 500 KB, 200 KB, 300 KB and 600 KB (in order), how would each of the first-fit, best-fit and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB and 426 KB (in that order) ? Which algorithm makes the most efficient use of memory?

Virtual Memory : Paging & Segmentation

Addresses

Table 8.1 Virtual Memory Terminology

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

Key points in Memory Management

1) Memory references are logical addresses dynamically translated into physical addresses at run time

   - A process may be swapped in and out of main memory occupying different regions at different times during execution

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

- **Portion** of a process that is in memory at any given time is called **resident set** of the process

Breakthrough in Memory Management

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

Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

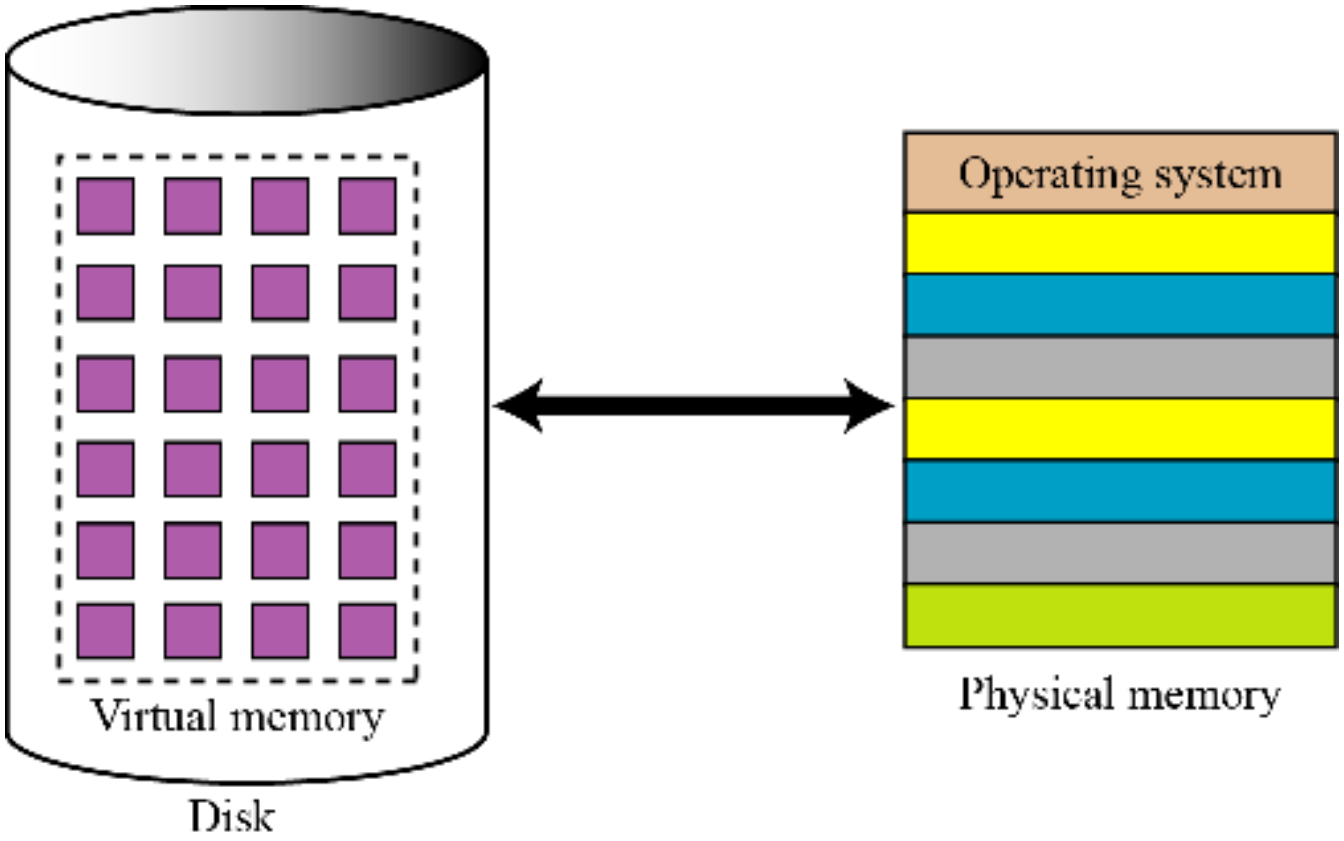
Virtual Memory

Demand paging and demand segmentation mean that, when a program is being executed, part of the program is in memory and part is on disk.

This means that, for example, a memory size of 10 MB can execute 10 programs, each of size 3 MB, for a total of 30 MB.

At any moment, 10 MB of the 10 programs are in memory and 20 MB are on disk. There is therefore an actual memory size of 10 MB, but a virtual memory size of 30 MB. Figure 7.11 shows the concept. **Virtual memory**, which implies demand paging, demand segmentation or both, is used in almost all operating systems today.

Virtual Memory



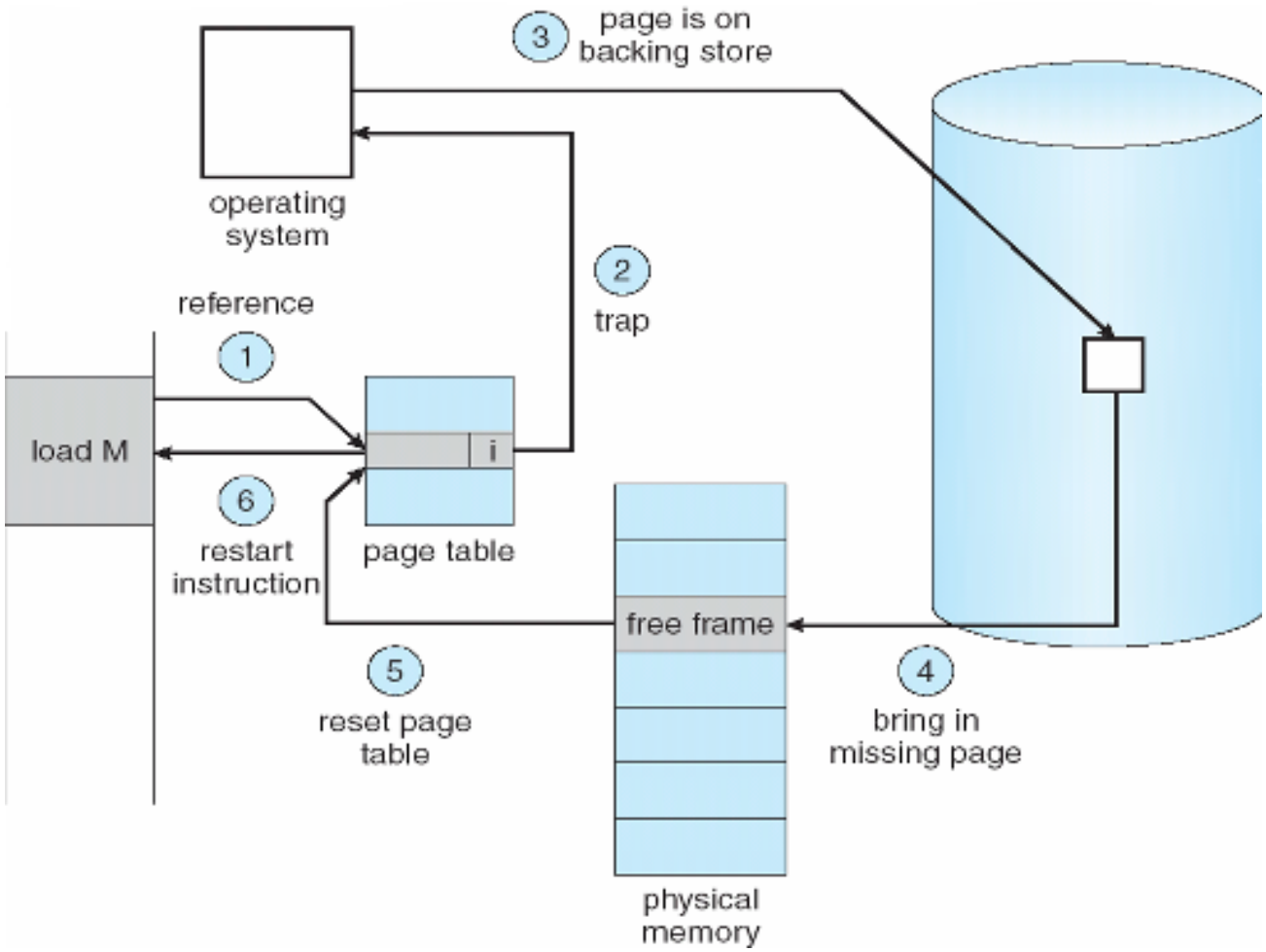
Virtual Memory

- If a ‘piece’ is not in memory and is required, processor generates interrupt indicating memory access fault
  - OS takes charge, puts process in blocked state
  - OS issues disk I/O to bring in the desired piece
  - Schedules another process in the mean time
  - Once brought in, I/O interrupt is raised, OS takes control again and puts the process in Ready queue
- VM Virtual Memory, Virtual Machine

Advantages of having only a portion

- More processes in memory at a given time with increase in CPU utilization, throughput but no increase in response time & turnaround time
- Processes can now be larger than main memory without any tension on part of the programmer (overlaying)
- Why to waste memory with portions of program/data which are being used only rarely
- Time is saved as unused pieces are not being swapped in/out
- Less I/O needed to load or swap user programs into memory, so each user program would run faster.

Steps in handling a page fault



Issues

- To bring in a piece, some other piece needs to be thrown out
  - Piece is thrown out just before it is used
  - Go get that piece again almost immediately
  - Leads to **thrashing**
    - System spends to much time swapping the pieces rather than executing instructions
    - Problem is worsened if OS mistakes it to be an indicator to increase the level of multi programming
- Solution: OS tries to guess which pieces are least likely to be used in the near future

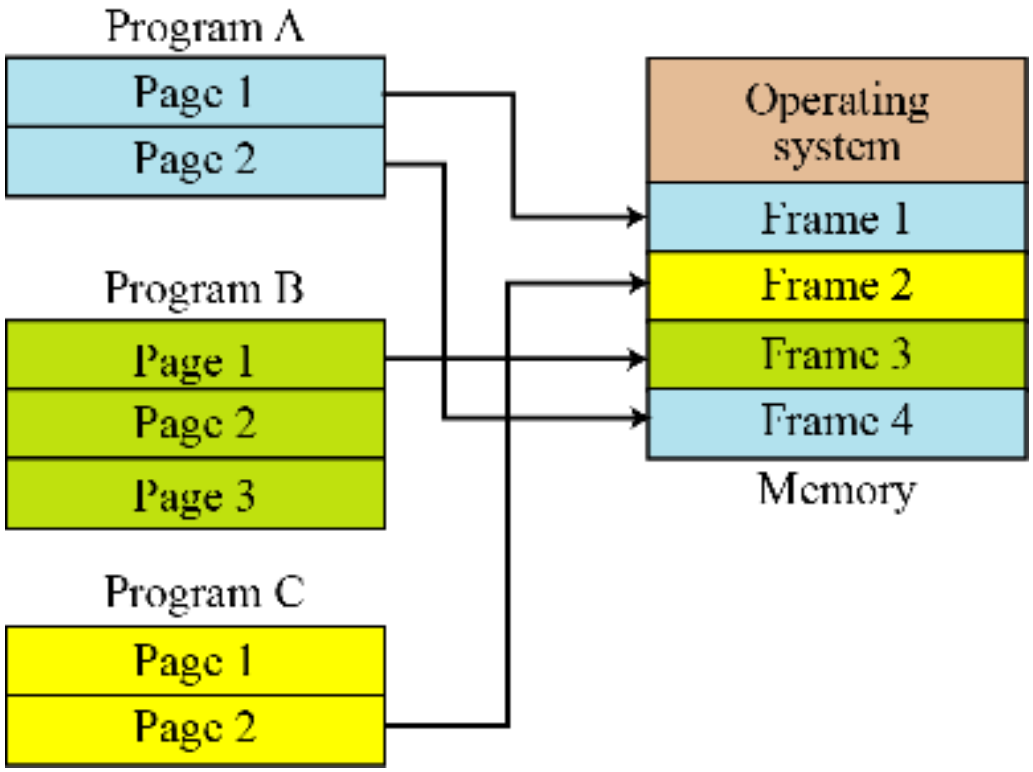
Virtual Memory Requirements

- Hardware must support paging and/or segmentation
- OS must support swapping of pages and/or segments

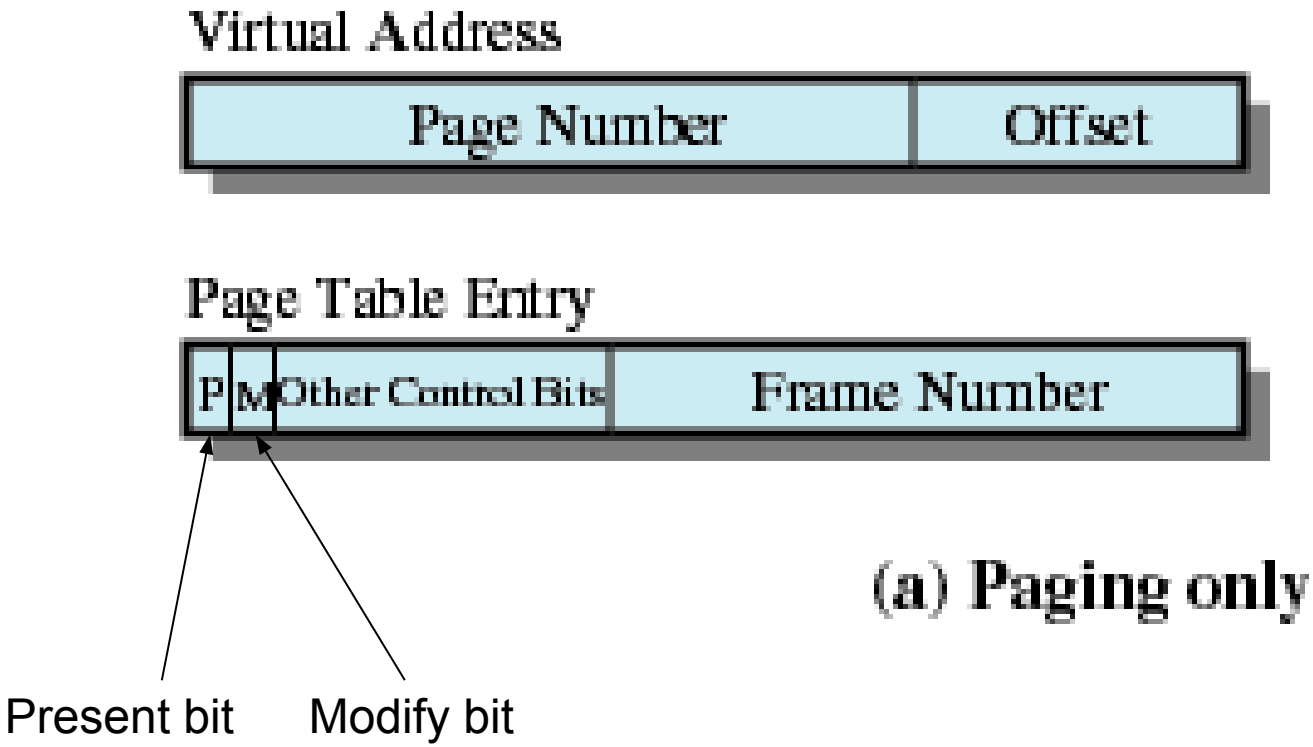
Virtual Memory + Paging

- Earlier paging: when all pages of a process are loaded into memory, its page table is created and loaded
- Page Table Entry (PTE) now needs to have
  - An extra bit to indicate whether page is in memory or not (P)
  - Another bit to indicate whether it is modified since it was last loaded (M)
    - When somebody else comes to replace me, I will not need to be written back to disk
  - Some control bits
    - For example protection or sharing at page level

Virtual Memory + Paging /Demand Paging



Page Table Entry



Page protection

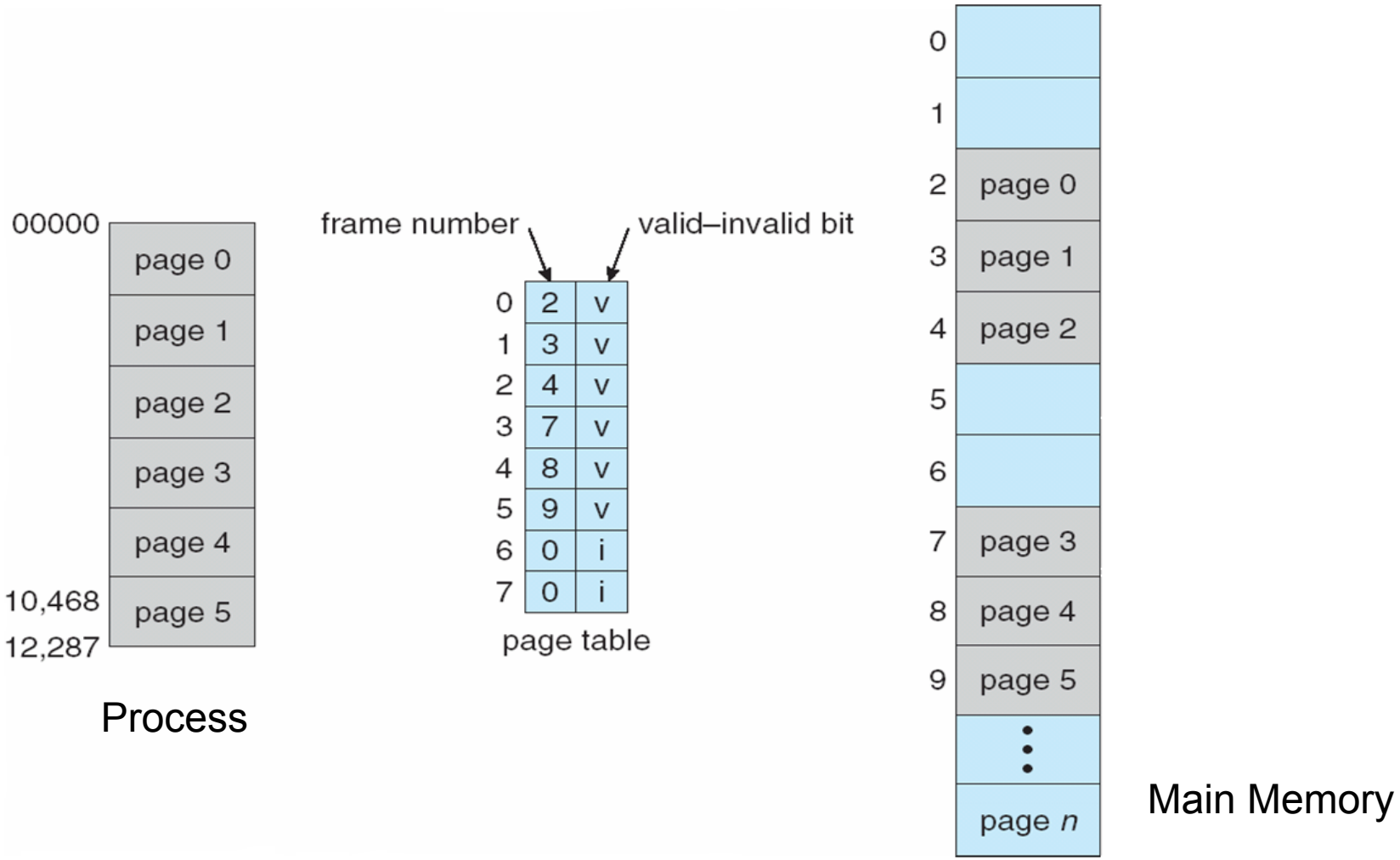
- Implemented by associating protection bits with each virtual page in page table
- Protection bits

  present bit: map to a valid physical page?

  read/write/execute bits: can read/write/execute?

  user bit: can access in user mode?
- x86: PTE_P, PTE_W, PTE_U
- Checked by MMU on each memory access

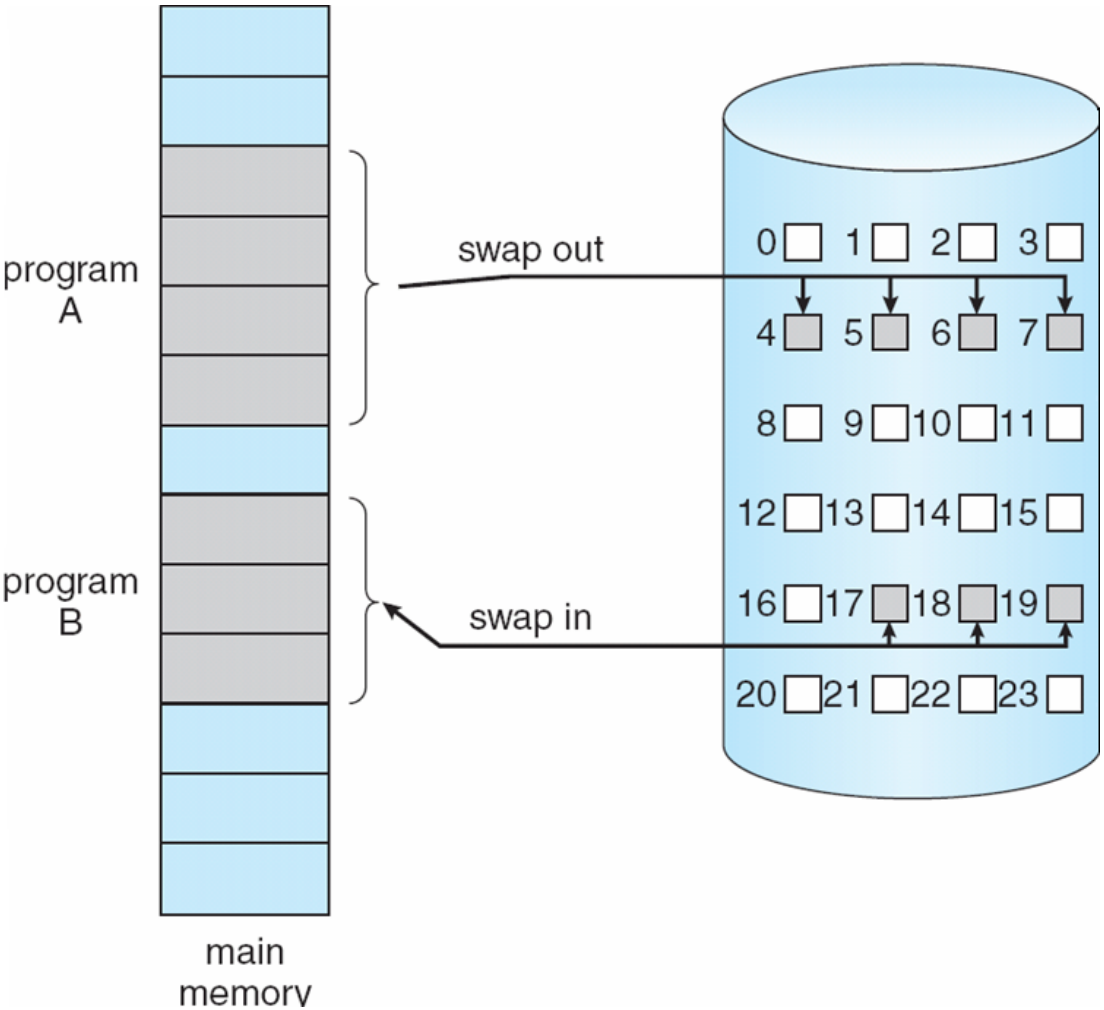
Example:Valid (v) or Invalid (i) Bit In A Page Table



Demand paging

- Page is needed reference to it
  - invalid reference abort
  - not-in-memory bring to memory
- Lazy swapper – never swaps a page into memory unless page will be needed
- Swapper that deals with pages is a pager

Transfer of a Paged Memory to Contiguous Disk Space



Address Translation in Paging

Page number field > frame number field n>m

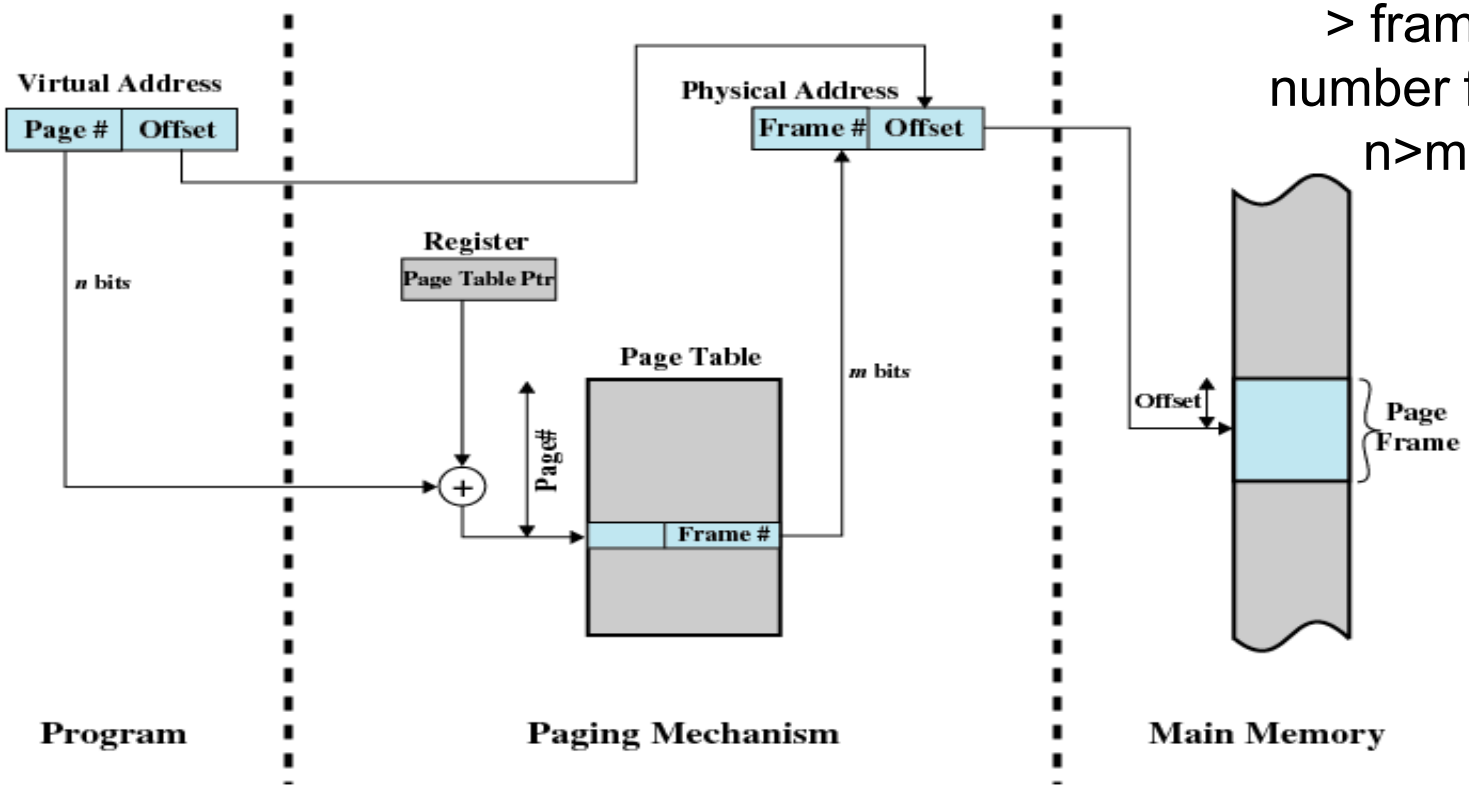


Figure 8.3 Address Translation in a Paging System

Page Tables

- Generally one page table per process
- Let us say a process requires 2GB virtual memory 2^31 pages /process
- How many 512 bytes pages will it contain? 2^31/2^9=2^22 pages / process
- As the size of page table increases, amount of memory required by it could be unacceptably high
- Page tables are also stored in virtual memory
- Page tables are subject to paging as other pages
- When a process is running, part of its page table must be in main memory.
- Page-table base register (PTBR) points to the page table
- Page-table length register (PRLR) indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

Translation Lookaside Buffer

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch appropriate page table
  - One to fetch appropriate data
- Effect of doubling the memory access time!
- To overcome this problem a high-speed cache is set up for page table entries
  - Called a Translation Lookaside Buffer (TLB)
  - Contains page table entries that have been most recently used

TLB Operation

- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table
  - First checks if page is already in main memory
    - If yes, go ahead, update TLB to include this new entry
    - If no, a page fault is issued to get the page (OS takes over), page table is updated, instruction is re-executed
- It have entries between 64 to 1024

Translation Lookaside Buffer

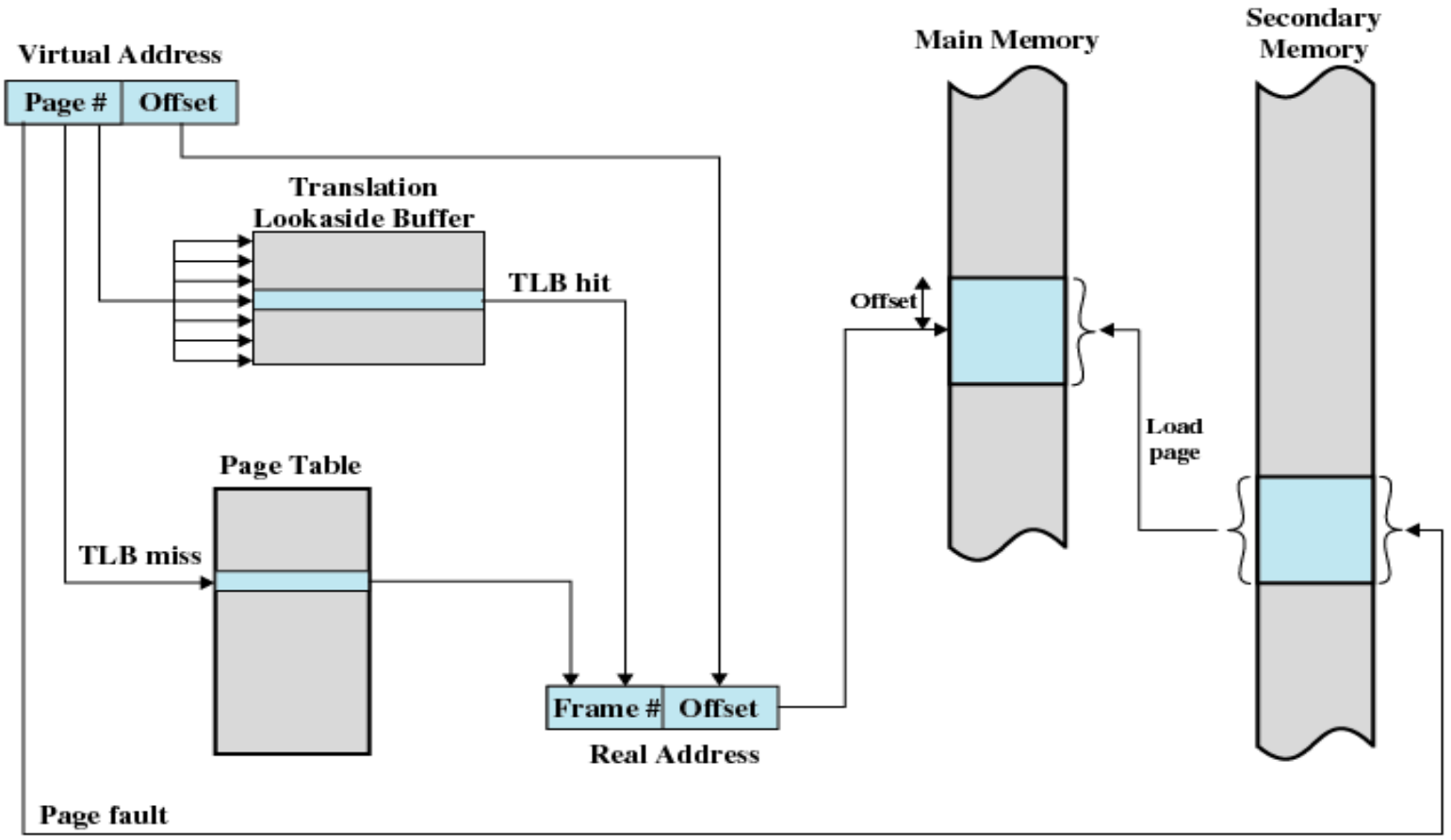


Figure 8.7 Use of a Translation Lookaside Buffer

TLB Operation

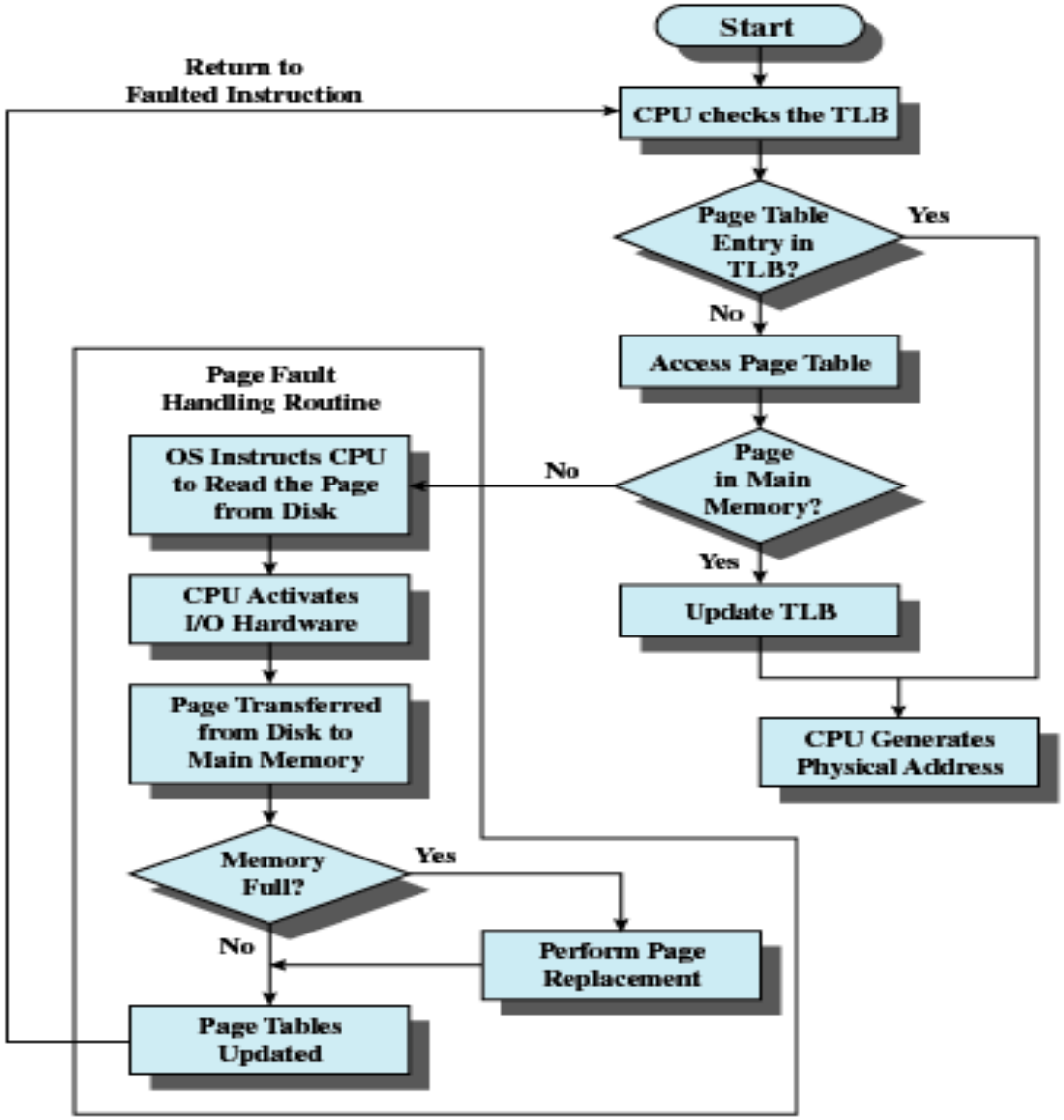


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Translation Lookaside Buffer

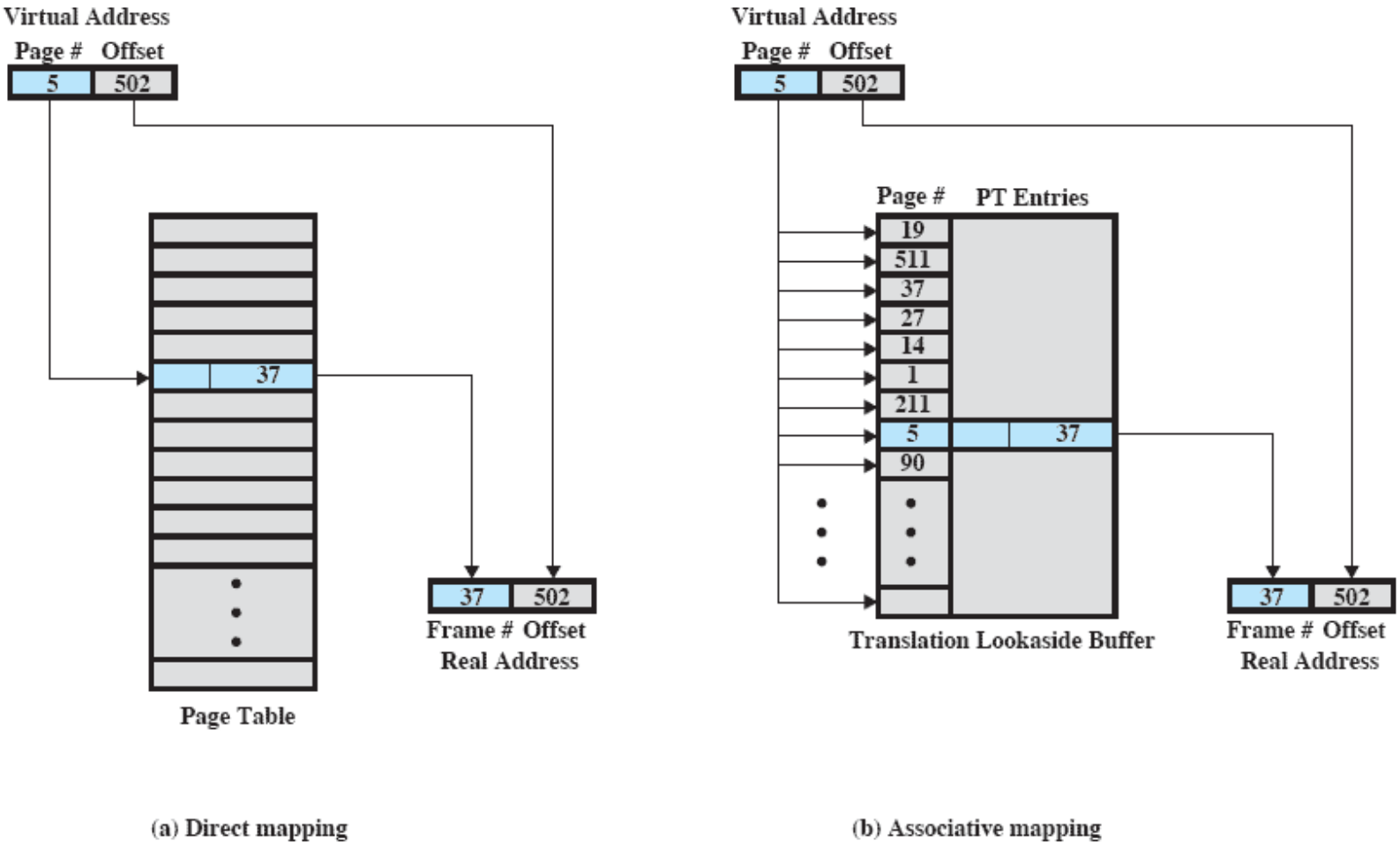


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

Translation Lookaside Buffer

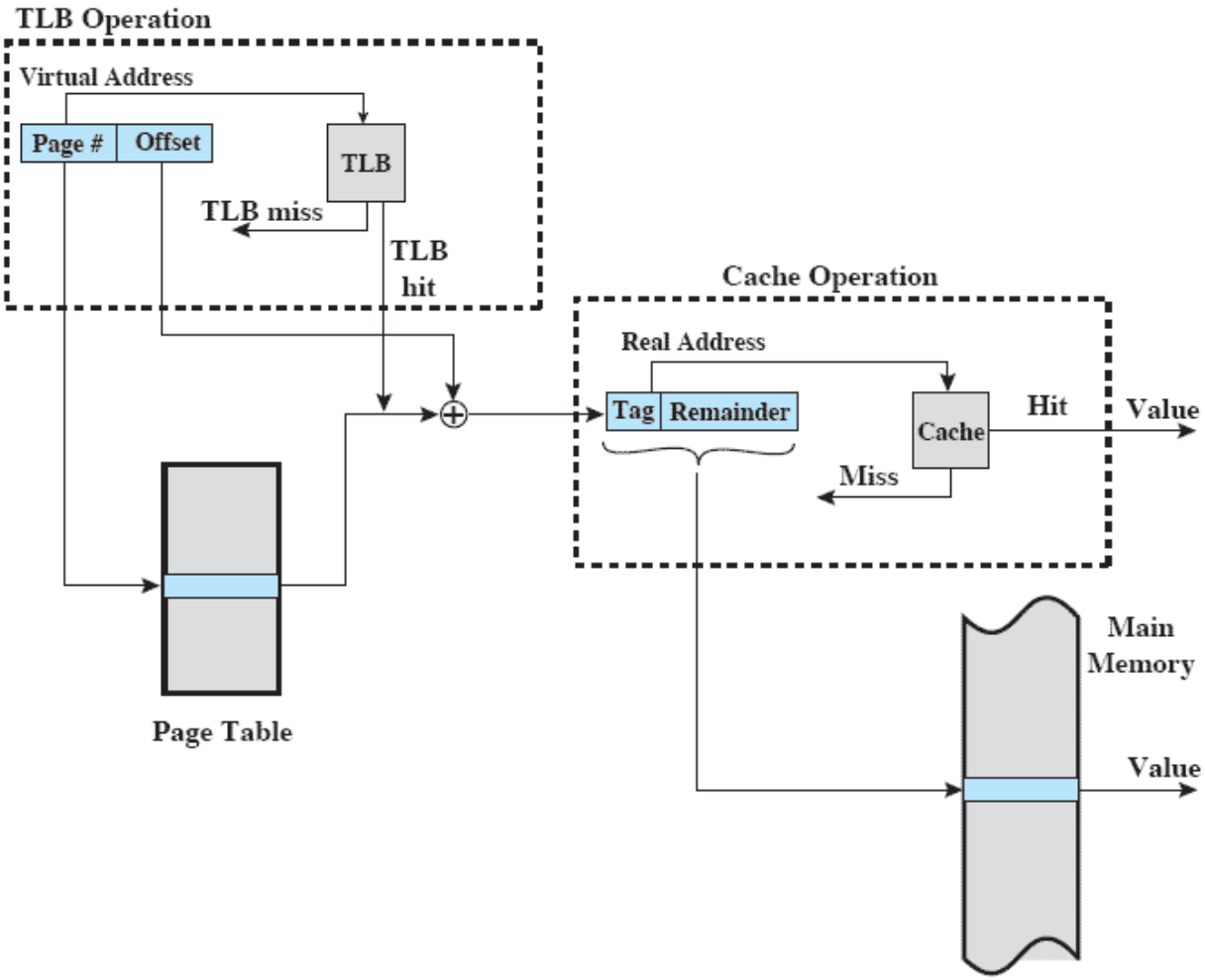


Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page table size issues

- Given:
  - A 32 bit address space (4 GB)
  - 4 KB pages
  - A page table entry of 4 bytes
- Implication: page table is 4 MB per process!
- Observation: address space are often sparse
  - Few programs use all of 2^32 bytes
- Change page table structures to save memory
  - Trade translation time for page table space

Solution: Hierarchical page table

Two Level Hierarchical Page Table

- Some processor makes use of two-level scheme to organize large page tables
- There is a page directory
  - Each entry points to a page table
  - If length of page directory is X & maximum length of a page table is Y,
    - A process can consist of up to X*Y pages.
    - Maximum length of a page table is restricted to be equal to one page

Two Level Hierarchical Page Table

4kbyte=2^12=page size

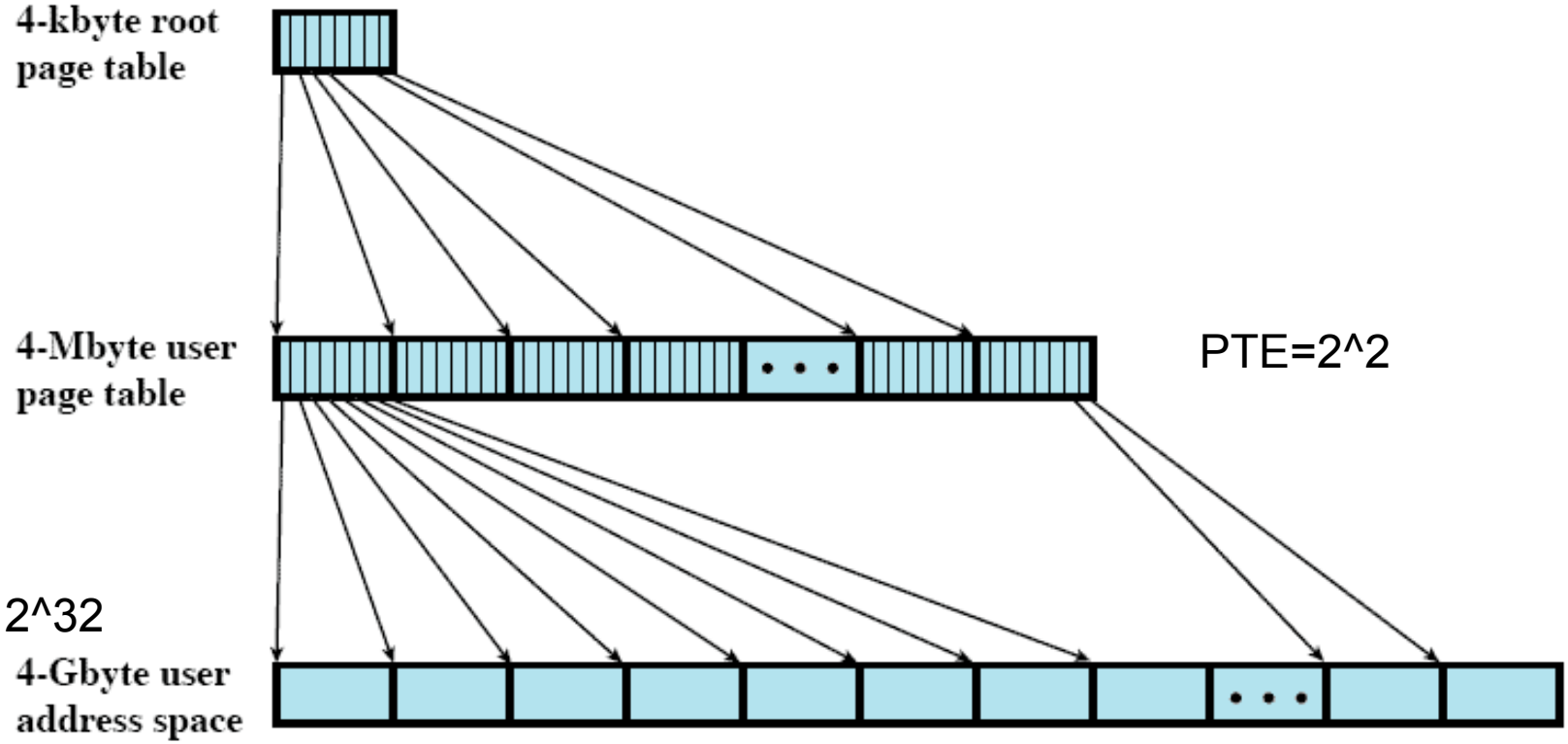


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation in Two-Level Paging System

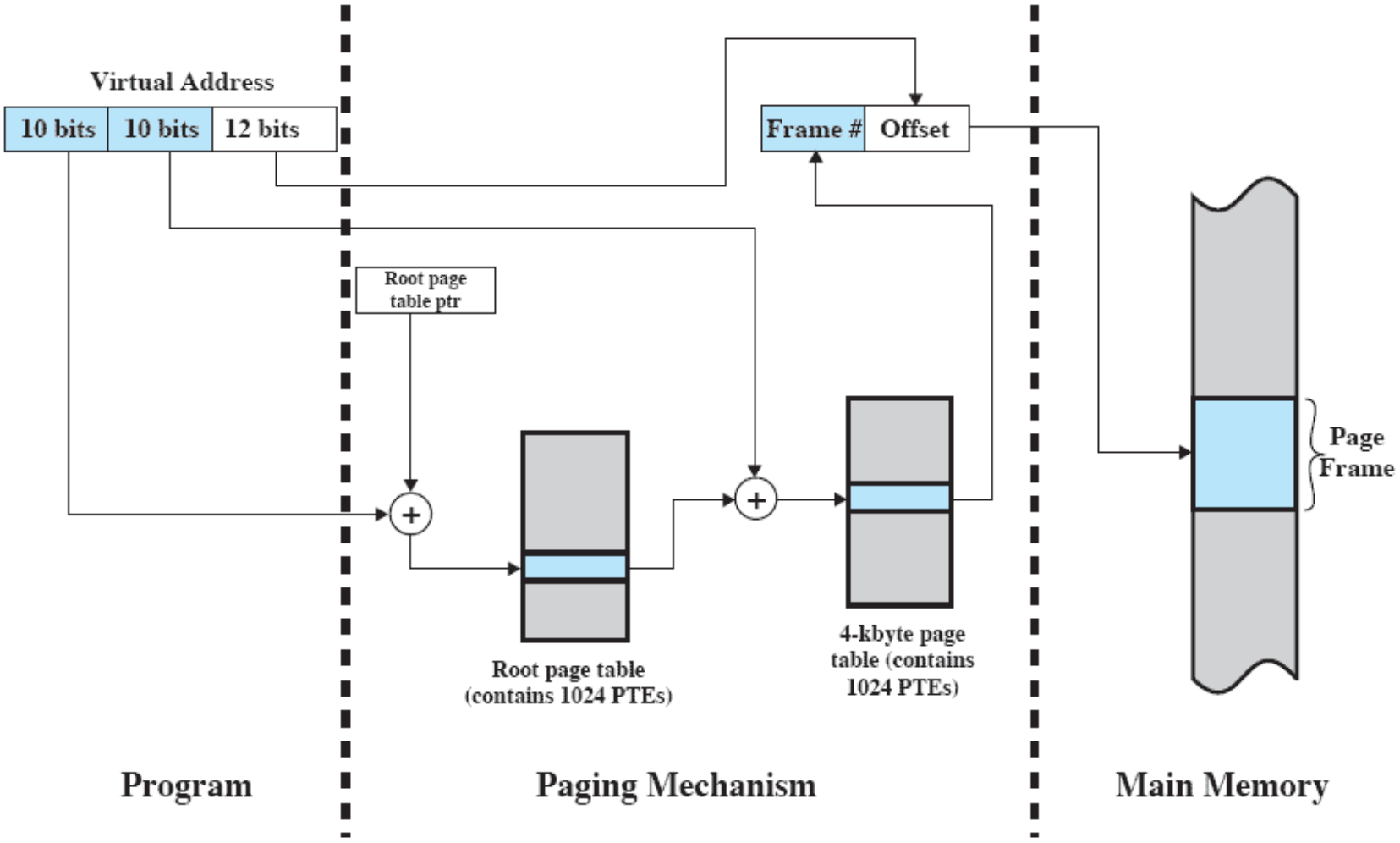


Figure 8.5 Address Translation in a Two-Level Paging System

Inverted Page Table

- Used on PowerPC, UltraSPARC, and IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes

Inverted Page Table

- Page number
- Process identifier
- Control bits
- Chain pointer

Inverted Page Table

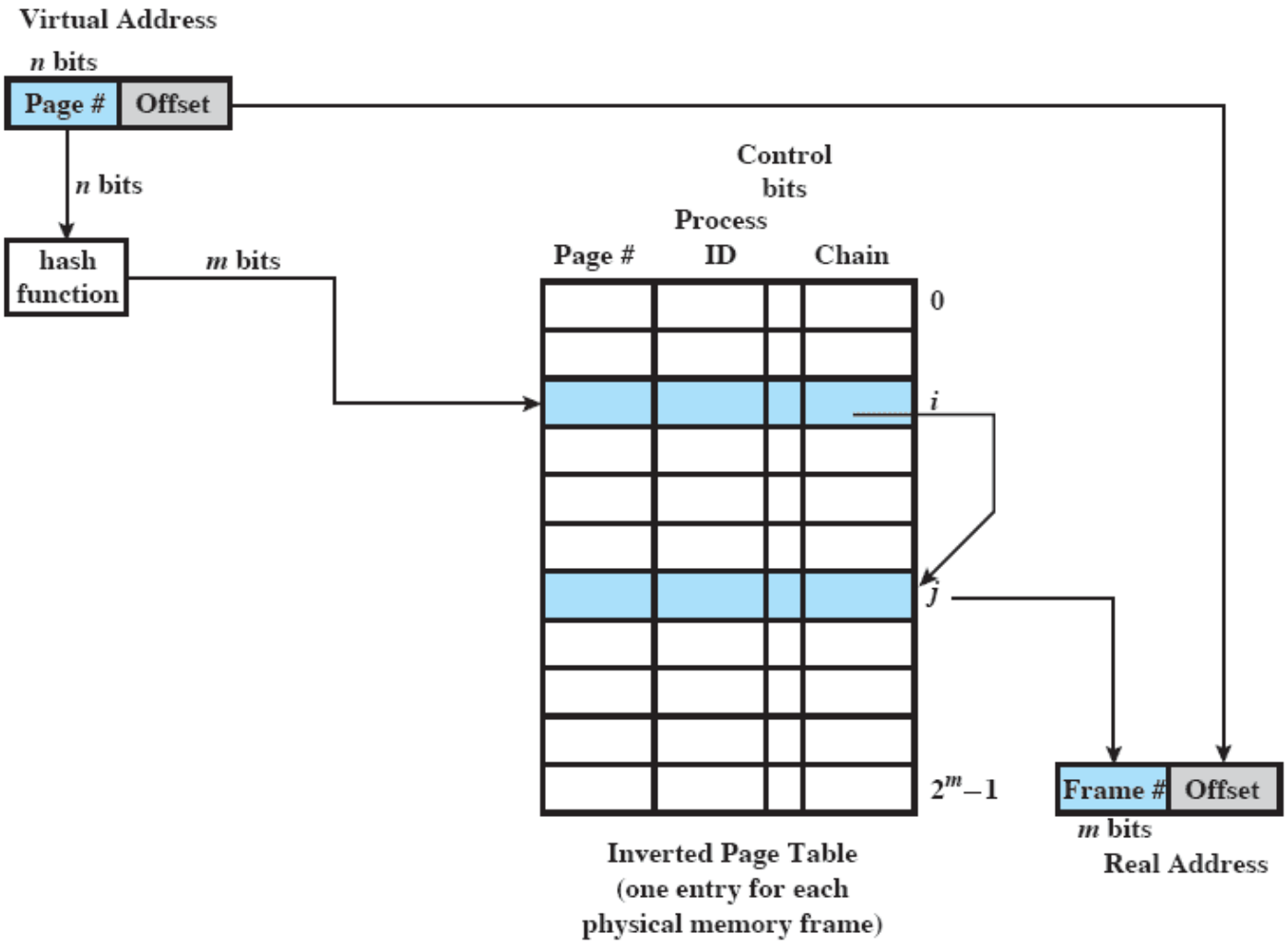


Figure 8.6 Inverted Page Table Structure

Page Size

- Effect on internal fragmentation?
  - Smaller less internal fragmentation
- optimal memory utilization-We want less internal fragmentation
  - Keep page size small
- But small page size more number of pages per process larger page tables using virtual memory for page table potential double page fault!
- Further, larger page size more efficient block transfer of data
  - Due to physical characteristics of rotational devices

Page Size

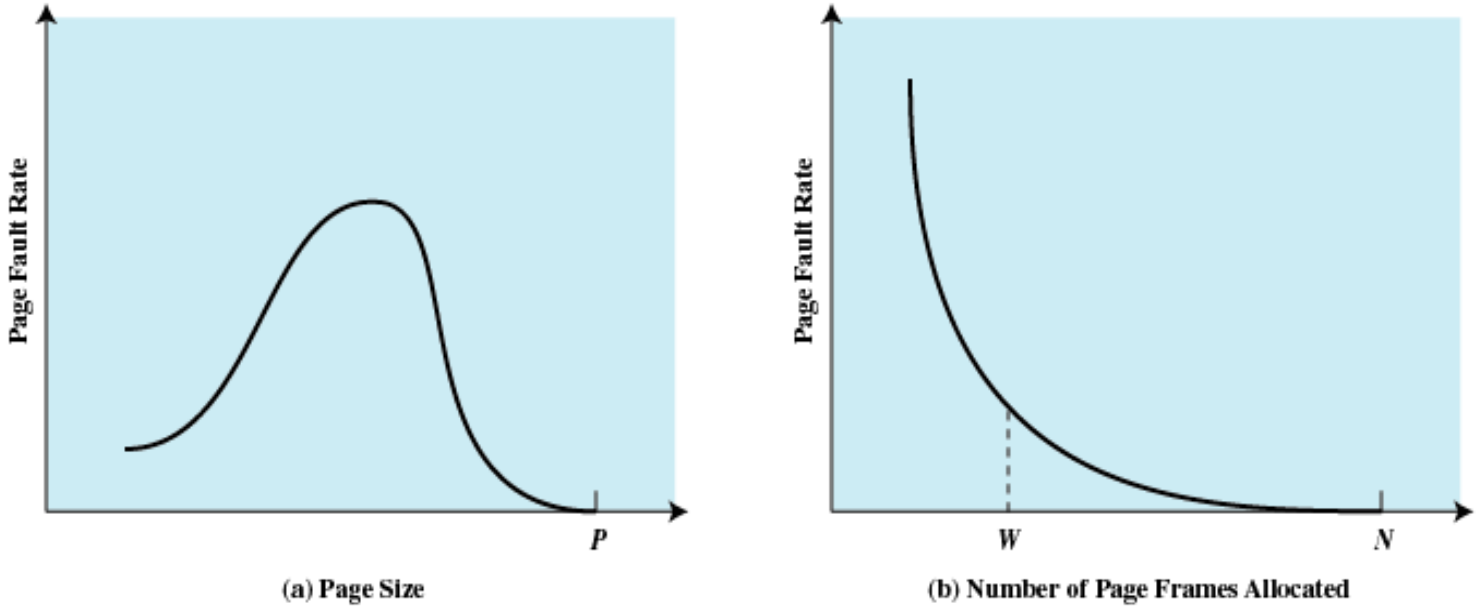
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better
- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references => Page faults low.
- Increased page size causes pages to contain locations further from any recent reference => Page faults rise.

Effect of page size on number of page faults

- With very small page size
  - More pages in memory, thus lesser page faults
  - Greater effect of principle of locality
    - One page refers to nearby locations
- As page size increases
  - Lesser pages in memory, thus more page faults
  - Effect of locality reduced
    - Each page will contain locations further and further away from recent references
- Page fault rate is also determined by the number of frames allocated to a process.

3. Size of physical memory & program size:

Page Size



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

Example Page Size

Table 8.3 Example Page Sizes

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
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbyes 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

Ordinary Paging vs VM Paging

Paging	Virtual memory paging
Main memory partitioned small into fixed - size chunks called frames	Main memory partitioned into small fixed - size chunks called frames
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system
Internal fragmentation within frames	Internal fragmentation within frames
No external fragmentation	No external fragmentation
OS must maintain page table for each process showing which frame each page occupies	OS must maintain page table for each process showing which frame each page occupies
OS must maintain a free frame list	OS must maintain a free frame list
Processor uses page number, offset to calculate absolute address	Processor uses page 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 the pages of a process must be in main memory for process. Page may be read in as needed
	Reading a page into main memory may require writing a page out to disk

Segmentation

- Memory consist of multiple address space/segments
- Segments may be of unequal/dynamic in size
- Memory reference consist of segment number, offset to form address.
- Have advantages to programmer over non segmented memory

Easier to relocate segment than entire program

Simplifies handling of growing data structures

Allows programs to be altered and recompiled independently

Sharing among processes

Provides protection

Avoids allocating unused memory (no internal fragmentation)

Efficient translation -> Segment table small (fit in MMU)

- Disadvantages

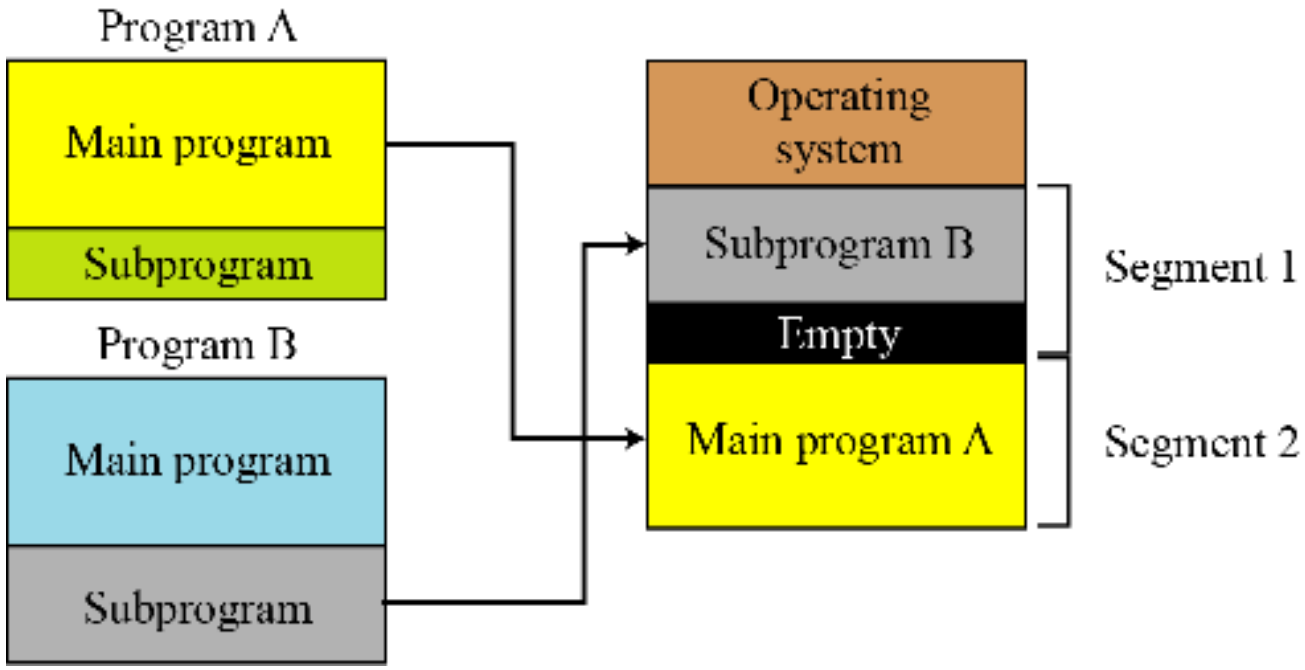
Segments have variable lengths -> how to fit?

Segments can be large -> external fragmentation

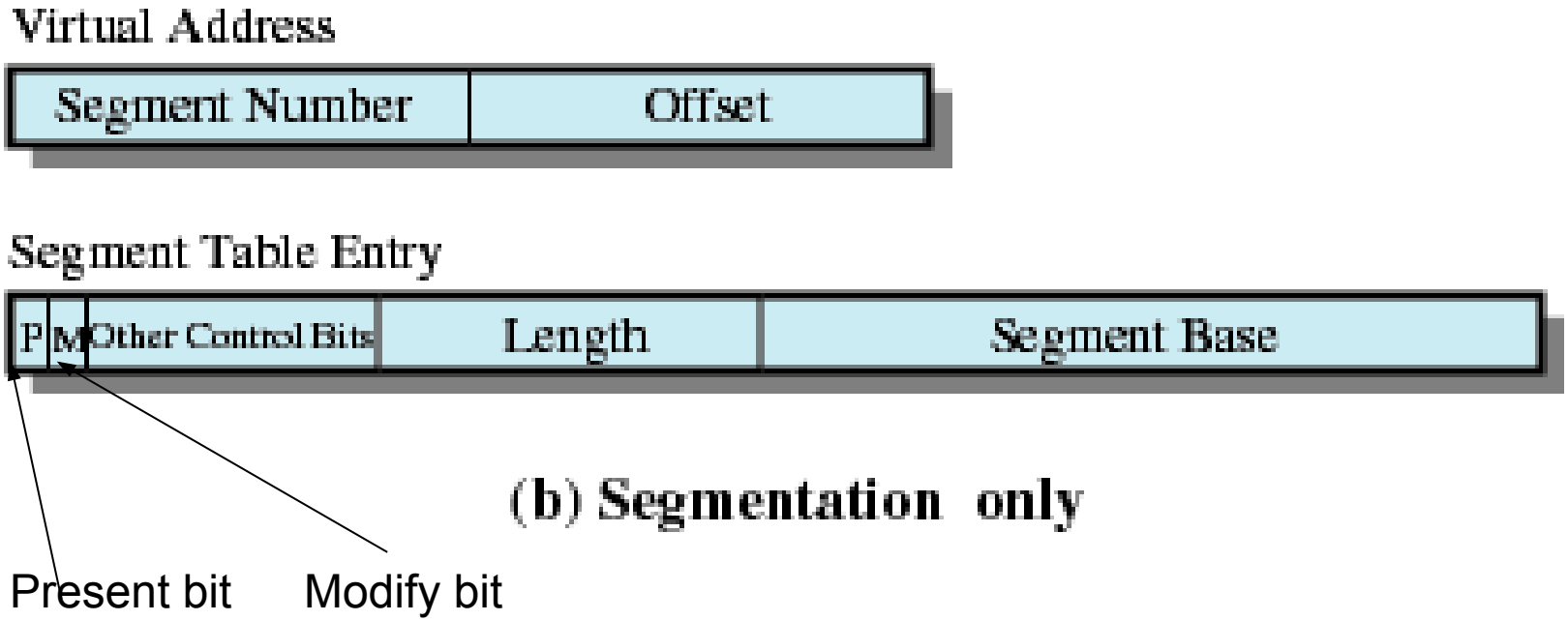
VM + Segmentation

- Earlier segmentation: each process has its own segment table
  - When all of its segments are loaded into main memory, segment table is created and loaded
- With VM, STEs become more complex
  - P bit (present/absent)
  - M bit (modify)
  - Other control bits
    - For example to support sharing or protection at segment level

VM + Segmentation



Segment Table Entries



Address Translation in Segmentation

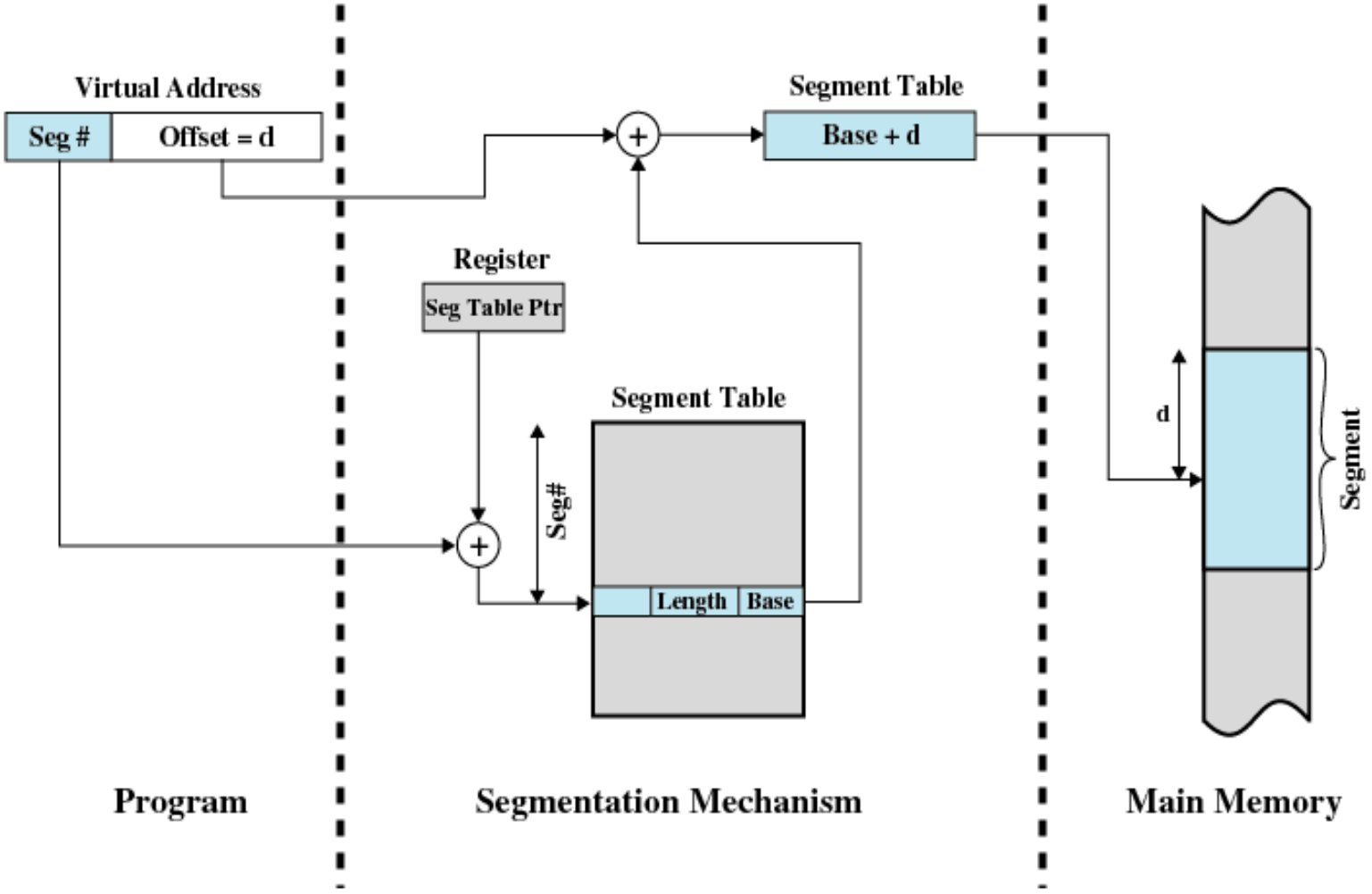


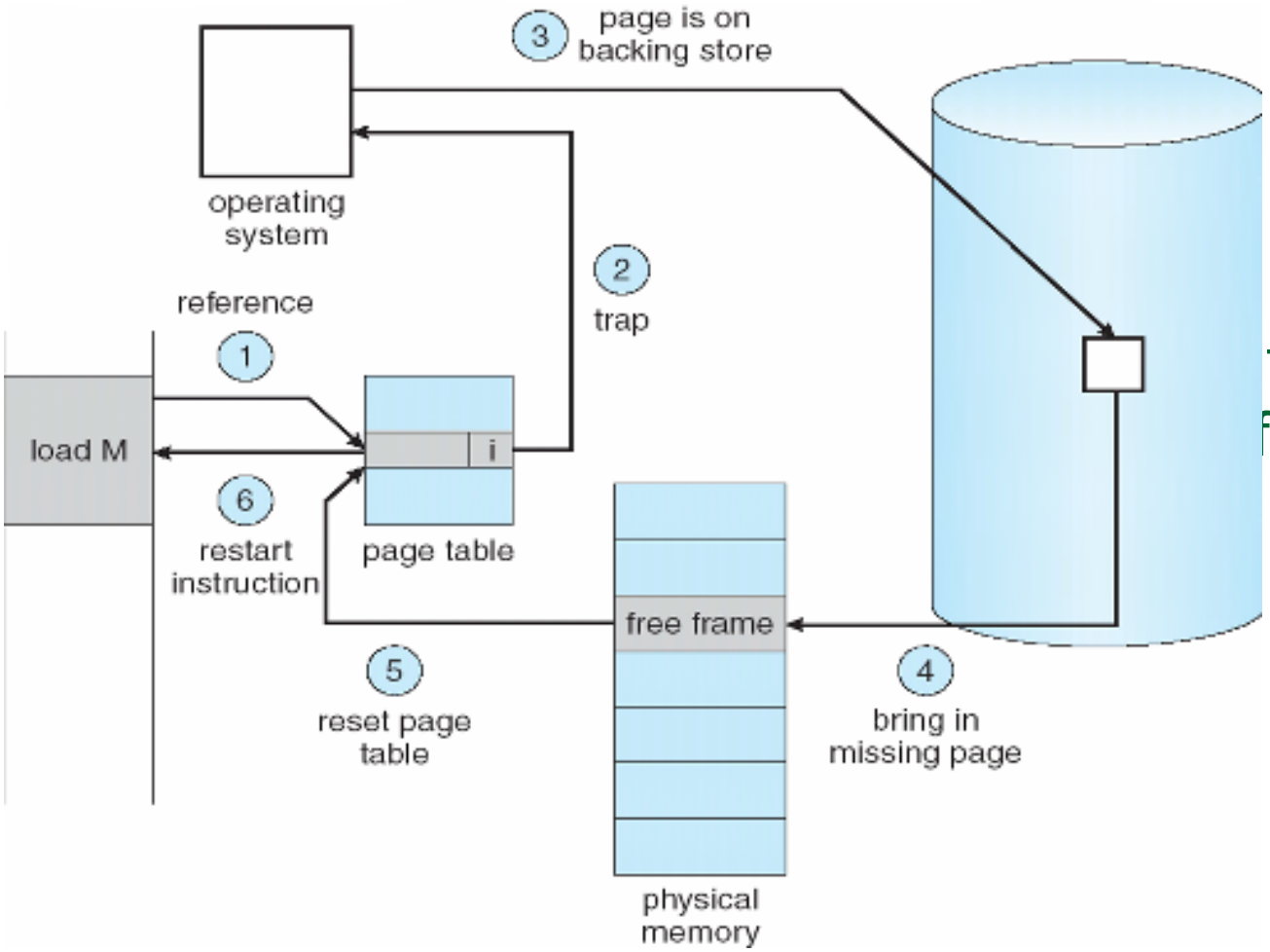
Figure 8.12 Address Translation in a Segmentation System

Ordinary Segmentation vs VM Segmentation

Simple Segmentation	Virtual memory segmentation
Main memory not partitioned	Main memory not partitioned
Program segments specified by the programmer to the compiler	Program segments specified by the programmer to the compiler
No Internal fragmentation	No Internal fragmentation
External fragmentation	External fragmentation
OS must maintain segment table for each process showing the load address and length of each segment	OS must maintain segment table for each process showing the load address and length of each segment
OS must maintain a list of holes	OS must maintain a list of holes
Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the segments of a process must be in main memory for process to run, unless overlays are used	Not all the segments of a process must be in main memory for process to run. Segment s may be read in as needed
	Reading a segment into main memory may require writing one/more segments out to disk

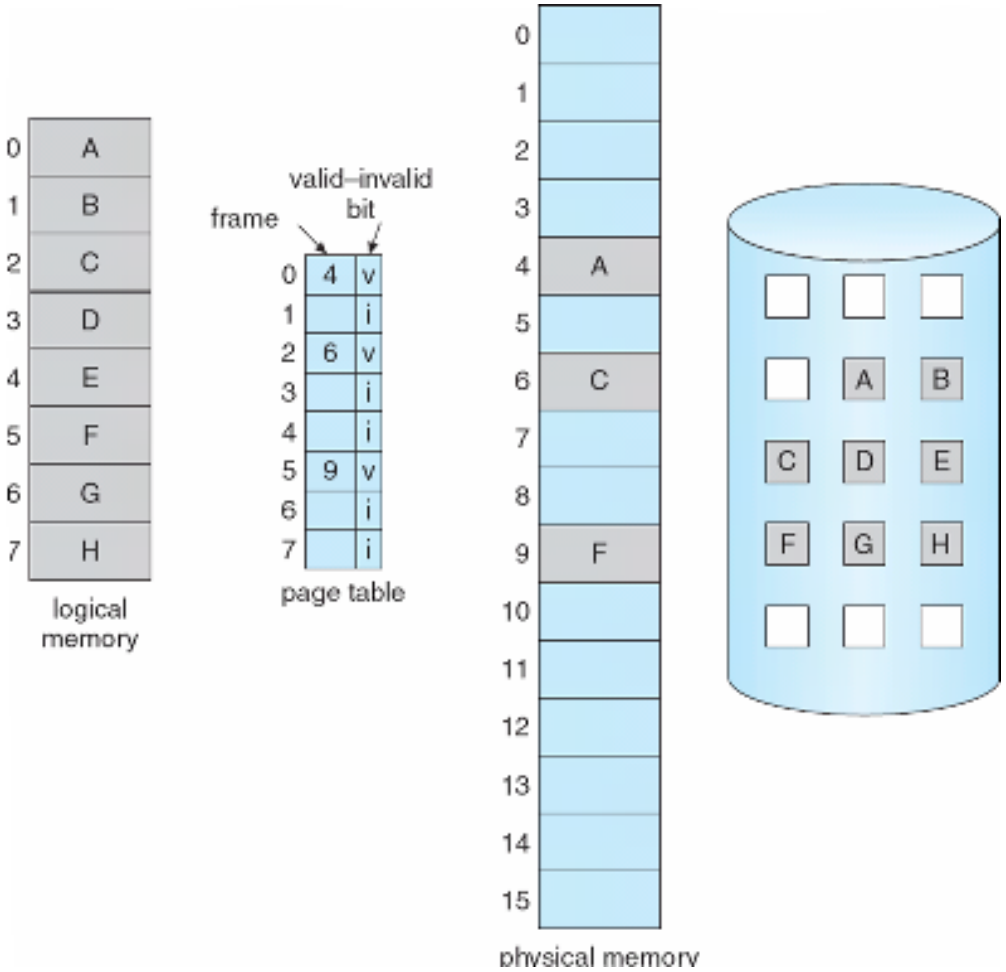
Contents

- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.
- OS Services layer in the Mobile OS: Multimedia and Graphics Services, Connectivity Services.



What happens when there is no free frame?

Page Table When Some Pages Are Not in Main Memory



Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:

  page fault

Operating system looks at another table to decide:

- Invalid reference ⇒ abort
- Just not in memory

Get empty frame

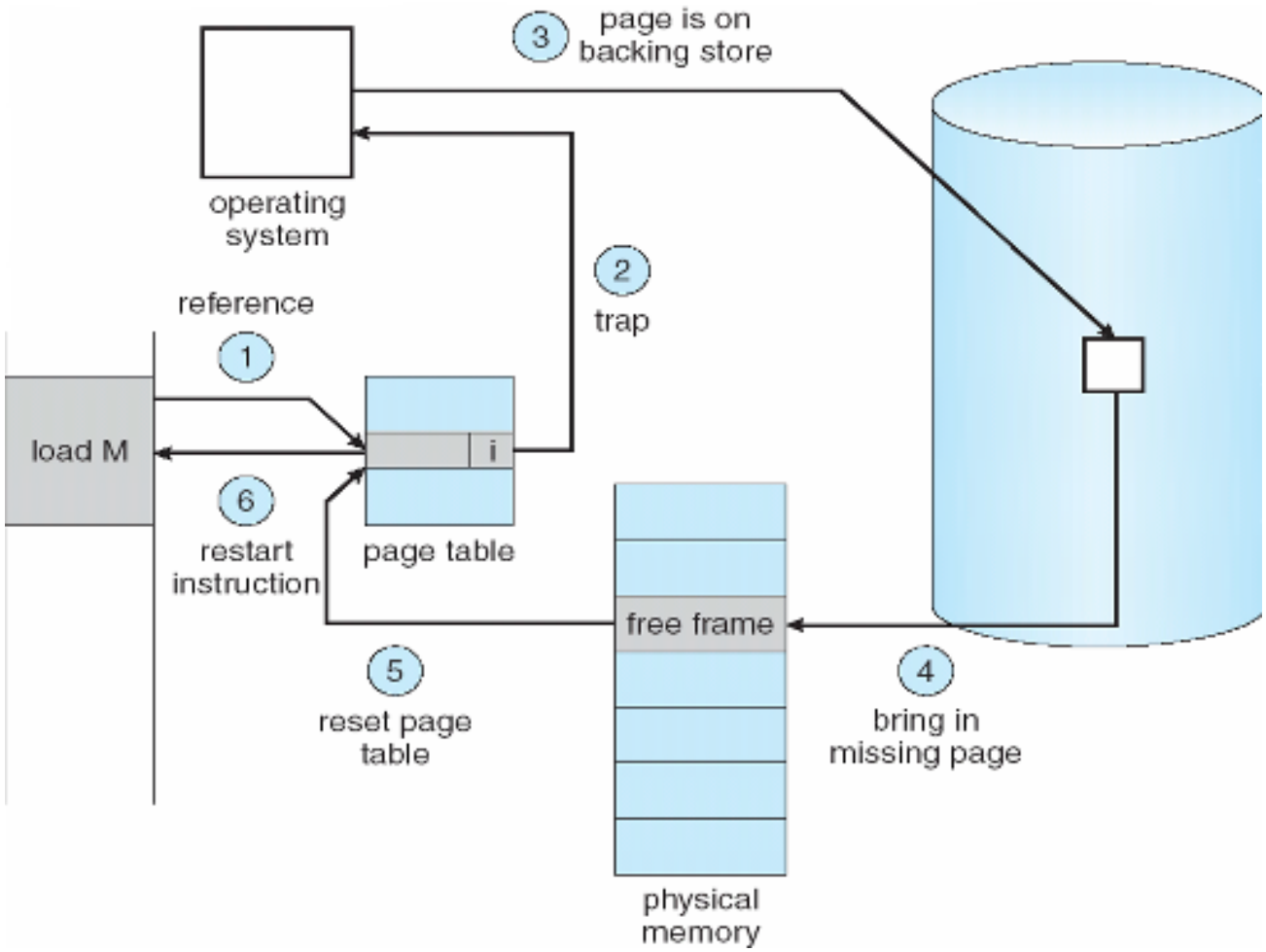
Swap page into frame

Reset tables

Set validation bit = **v**

Restart the instruction that caused the page fault

Steps in handling a page fault



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out
  - Different algorithms, different performance
  - Want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement

- Use **modify (dirty) bit** to reduce overhead of page transfers – only modified pages are written to disk

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Basic Page Replacement

Find the location of the desired page on disk

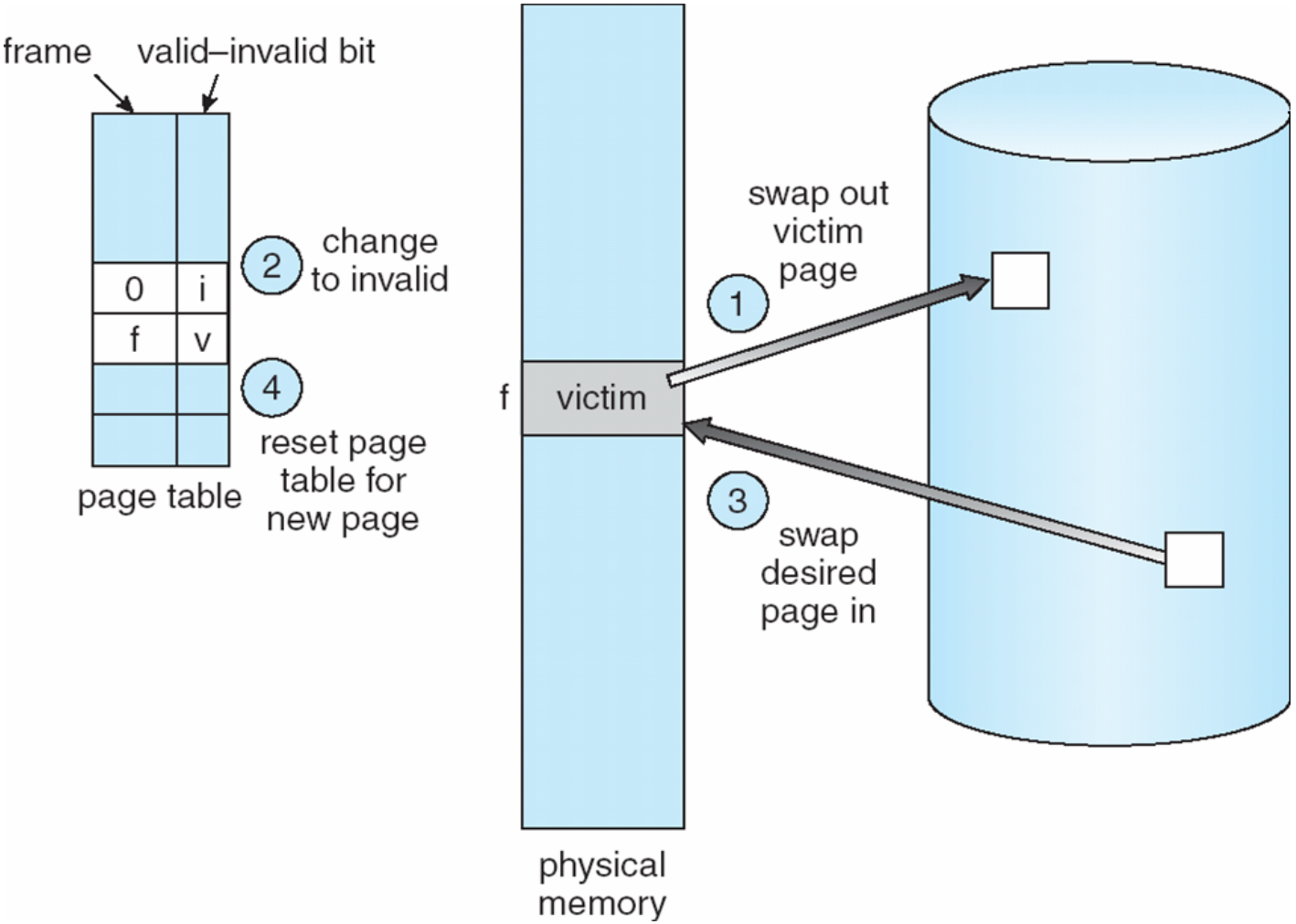
Find a free frame:

- If there is a free frame, use it
- If there is no free frame, use a page replacement algorithm to select a **victim** frame

Bring the desired page into the (newly) free frame; update the page and frame tables

Restart the process

Page Replacement



Page-Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is

A B C A B D A D B C B

Basic 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
  - These pages may be needed again very soon
  - How would you implement FIFO strategy?
    - Keep a queue and do round robin

Example: FIFO

- Suppose we have 3 page frames, 4 virtual pages, and following reference stream:
  - A B C A B D A D B C B
- Consider FIFO Page replacement:

Ref: / Page:	A	B	C	A	B	D	A	D	B	C	B
1	A					D				C	
2		B					A				
3			C						B		

  - FIFO: 7 faults.
  - When referencing D, replacing A is bad choice, since need A again right away

Basic Replacement Algorithms

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

Example: Optimal(MIN)

- Suppose we have the same reference stream:
  - A B C A B D A D B C B
- Consider Optimal Page replacement:

Ref: / Page:	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

  - MIN: 5 faults
  - Where will D be brought in? Look for page not referenced farthest in future.

Basic 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
  - Each page could be tagged with the time of last reference. This would require a great deal of overhead.

Example: LRU

- Suppose we have the same reference stream:
  - A B C A B D A D B C B
- Consider LRU Page replacement:

Ref: / Page:	A	B	C	A	B	D	A	D	B	C	B
1	A									C	
2		B									
3			C			D					

  - LRU: 5 faults
- What will LRU do?
  - Same decisions as MIN here, but won't always be true!

LRU Algorithm (Cont.)

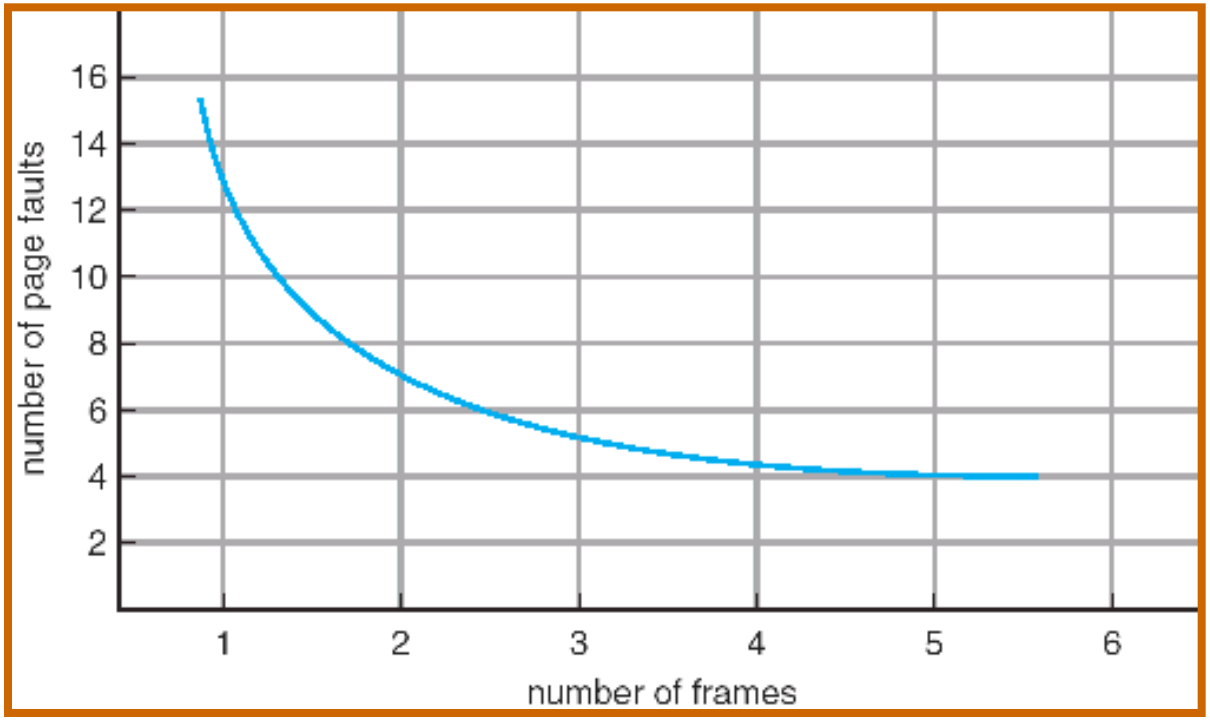
- How would you implement LRU strategy?
- Stack implementation – keep a stack of page numbers in a double link form:
  - Any time Page is referenced move it to the top
  - Replace page from the bottom of the stack
  - No search for replacement

- RANDOM:
  - Pick random page for every replacement
  - Typical solution for TLB's. Simple hardware
  - Pretty unpredictable – makes it hard to make real-time guarantees

Global vs. Local Allocation

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- Local replacement – each process selects from only its own set of allocated frames.

Graph of Page Faults Versus The Number of Frames



- One desirable property: When you add memory the miss rate goes down
  - Does this always happen?
  - Seems like it should, right?
- No: BeLady's anomaly
  - Certain replacement algorithms (FIFO) don't have this obvious property!

Adding Memory Doesn't Always Help Fault Rate

- Does adding memory reduce number of page faults?
  - Yes for LRU and MIN
  - Not necessarily for FIFO! (Called Belady's anomaly)

Ref: / Page:	A	B	C	D	A	B	E	A	B	C	D	E
1	A			D			E					
2		B			A					C		
3			C			B					D	

Ref: / Page:	A	B	C	D	A	B	E	A	B	C	D	E
1	A						E				D	
2		B						A				E
3			C						B			
4				D						C		

Comparison of Algorithms

	FIFO	OPT/MIN	LRU
Data structure for implementation	Queue	NA	DLL using counter or stack method
Traversal of reference string	Forward	Forward	Backward
Implementation	Easy to understand n imple.	Diff. to impl.	Approximation of OPT ,impl is possible
Performance	Not good	Mainly used for comparison study	Good
Time to be considered	Uses the time when page was brought into memory	Uses the time when a page is to be used	Approximation of OPT , Uses the time when a page is used

Decisions involved in VM implementation

- Whether or not to use virtual memory techniques?
  - Depend on hardware support
- The use of paging or segmentation or both
  - Depend on hardware support
- The algorithms employed for various aspects of memory management
  - Depends on OS

Table 8.4 Operating System Policies for Virtual Memory

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 page should be brought into main memory.

1. Demand paging-page is brought into memory only when a reference is made to a location on that page.
2. Prepaging-pages other than the one demanded by a page fault are brought in.

Fetch policy

- Prepaging exploits the characteristics of most secondary memory devices,such as disks.
- If the pages of a process are stored contiguously in secondary memory, then it is more efficient to bring in a number of contiguous pages at one time rather than bringing them in one at a time over an extended period.
- this policy is ineffective if most of the extra pages that are brought in are not referenced.

Placement Policy

- determines where in real memory a process piece is to reside.
- In a pure segmentation system, the placement policy is an important design issue;policies such as best-fit, first-fit, and so on

Replacement policy

How many page frames are to be allocated to each active process

Global/local replacement

Victim page selection(Page Replacement).

- First two are resident set management

Frame locking

- Some of the frames in main memory may be locked.
- If frame is locked ,page stored in that may not be replaced.
- Kernel code ,control structures,i/o buffers are stored in locked frames.
- Associate lock bit with each frame.

Page Buffering

- LRU is better than FIFO, but is more complex
- Cost of replacing a page that has been modified is greater than for one that has not
- Interesting strategy: Page Buffering
- Page replacement algorithm is simple FIFO
- A replaced page is not lost but rather is assigned to one of two lists
  - Free page list if the page has not been modified, or the modified page list if it has
- Page is not physically moved about in main memory
  - Entry in the page table for this page is removed and placed in either the free or modified page list

Page Buffering

- When an unmodified page is to be replaced, it remains in memory and its page frame is added to the tail of the free page list
- Free page list is a list of page frames available for reading in pages
- When a page is to be read in, the page frame at the head of the list is used, destroying the page that was there
- When a modified page is to be written out and replaced, its page frame is added to the tail of the modified page list

Advantage?

- Page to be replaced remains in memory
- If the process references that page, it is returned to the resident set of that process at little cost
  - In effect, the free and modified page lists act as a cache of pages
- Modified pages are written out in clusters rather than one at a time

Decisions taken by OS

- Resident Set Management
  - How many frames to be allocated to each active process
  - Victim should be from the same process or could be page from any process?(Local / Global)

Resident Set Management

- OS must decide how many pages to bring in
  - how much main memory to allocate to a particular process
- Factors
  - Less memory more multi programming
  - Less pages more page faults
  - Beyond a certain size, further allocations of pages will not affect the page fault rate due to principle of locality
- Two policies
  - Fixed Allocation
  - Variable Allocation

Fixed Allocation Policy

- Gives a process a fixed number of frames in main memory within which to execute.
- Number is decided at initial load time (process creation time)
- Based upon
  - Type of process (interactive, batch, type of application)
  - Guidance from the programmer or system manager
- Page fault one of the pages of that process must be replaced by the needed page.

Variable Allocation Policy

- Number of page frames allocated to a process to be varied over the lifetime of the process.
- A process suffering from high page fault rate will be given more frames and vice versa.
- Related with the concept of replacement scope.
- Obviously better, but requires OS to assess the behavior of active processes.

Replacement Scope

- Could be global or local
- Activated when there is a page fault and no free frame is available
- Local replacement policy
  - Chooses only among the resident pages of the process that generated the page fault as victim
  - Easier to analyze
- Global replacement policy
  - Considers all unlocked page frames in main memory as candidates for replacement, regardless of which process owns a particular page
  - Simple implementation, minimal overhead

Valid Combinations

- Fixed allocation local replacement
- Variable global or local

Fixed Allocation, Local Scope

- Decide ahead of time the amount of allocation to give a process
- Allocation could depend upon type of application or amount requested by programmer.
- Drawbacks:
  - If allocations tend to be too small, then there will be a high page fault rate, causing the entire multiprogramming system to run slowly.
  - If allocations tend to be unnecessarily large, then there will be too few programs in main memory and there will be either considerable processor idle time or considerable time spent in swapping.

Variable Allocation, Global Scope

- Easiest to implement
  - Adopted by many operating systems
- Operating system keeps list of free frames
- When a page fault occurs, a free frame is added to the resident set of a process and the page is brought in
  - A process experiencing page faults will gradually grow in size, which should help reduce overall page faults in the system
- If no free frame, replaces one from another process
  - Therein lies the difficulty … which to replace.
  - Use different page replacement algorithms

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
  - Based on some factors
  - Use either prepaging or demand paging to fill up the allocation
- When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault
- From time to time, reevaluate the allocation provided to the process, and increase or decrease it to improve overall performance
  - Based on an assessment of the likely future demands of active processes

Variable Allocation,Local Scope

- More complex than variable, global
- May yield better performance
- Key parameters
  - Criteria used to determine resident set size and the timing of changes
  - One such strategy is **working set** strategy

Resident Set Management Summary

Table 8.5 Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	• Number of frames allocated to 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.

Decisions taken by OS

- Cleaning Policy
  - When a modified page should be written out to secondary memory?
  - Demand cleaning: page is written out to secondary memory only when it has been selected for replacement
    - Process blocked for more time
  - Pre-cleaning: writes modified pages before their page frames are needed so that pages can be written out in batches
    - Problem: writing could be waste if modified again while in memory
  - Best approach is page buffering
- Load Control
  - Determining the number of processes that will be resident in main memory – called multi programming level

Contents

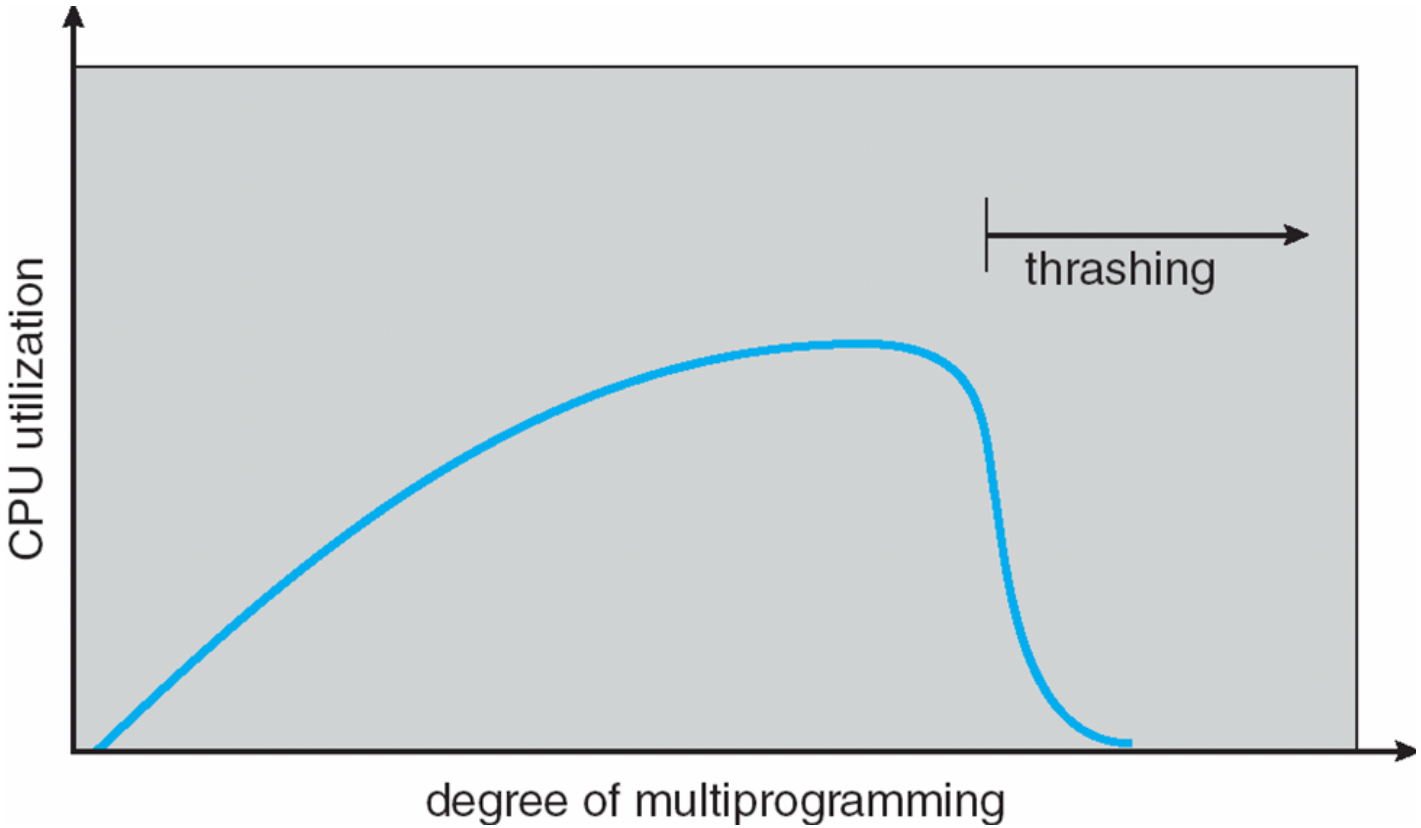
- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Virtual Memory: Concepts, Segmentation, Paging, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- **Thrashing** ≡ a process is busy swapping pages in and out
- Swapping out a piece of a process just before that piece is needed
- The processor spends most of its time swapping pieces rather than executing user instructions

Thrashing (Cont.)



- If global page replacement algorithm is used then thrashing occurs
- To avoid this use local page replacement algorithm

Thrashing and Global Replacement Policy

- Global scope increases thrashing
- The process scheduler see that CPU utilization is low.
  - Increases the degree of multiprogramming by introducing a new process into the system
- One process now needs more frames
  - It starts faulting and takes away frames from other processes. (i.e., global-page replacement).
  - These processes need those pages and thus they start to fault, taking frames from other processes.
  - These faulting processes must use the paging device to swap pages in and out.
  - As they queue up on the paging device, the ready-queue empties.
- However, as processes wait on the paging device, CPU utilization decreases
- The process scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming !!!

Thrashing and Local Replacement Policy

- Only partially solves the problem
- If one process starts thrashing, it cannot steal frames of another process and cause the later to thrash
- However, the thrashing processes will be in the paging device queue which will increase the time for a page fault to be serviced
  - Effective access time will increase even for those processes not thrashing

How to avoid thrashing?

- Provide processes with as many frames as they need
- How to determine this?
  - Look at how many frames a process actually "uses“
- Locality model
- Working set model
- Page Fault Frequency Model

Self Study

- Combined Paging and segmentation
- Working Set Model

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
- Possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references

  Example: 10,000 instruction

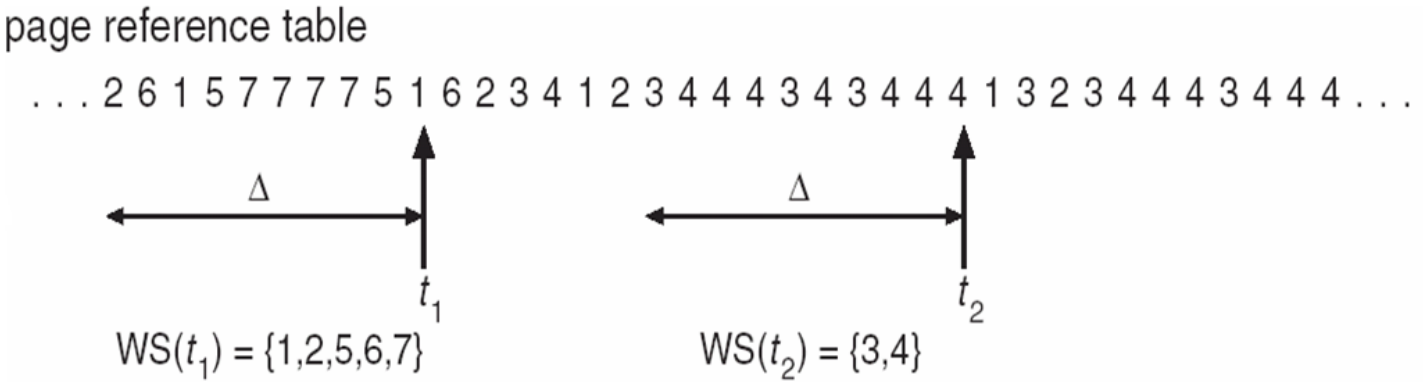
- WSS_i (working set of Process P_i) =

  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
- if $D > m \Rightarrow$ Thrashing
- Policy if $D >$ m, then suspend one of the processes

Working-set model

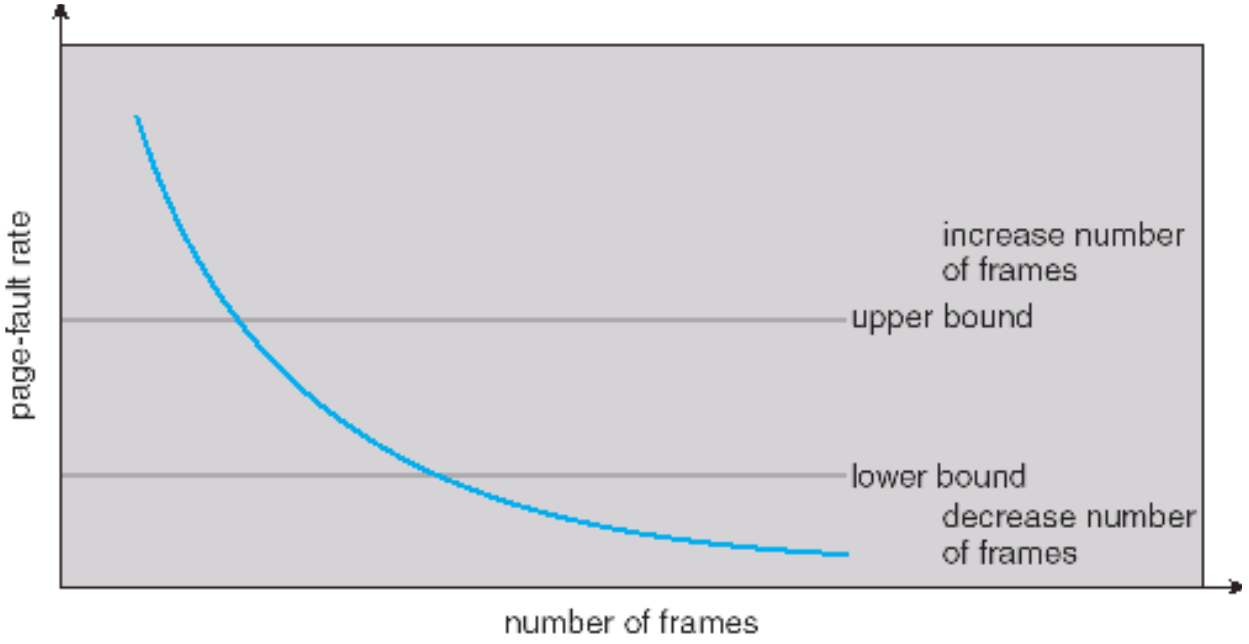


Keeping Track of the Working Set

- 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 Scheme

- Establish “acceptable” page-fault rate
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



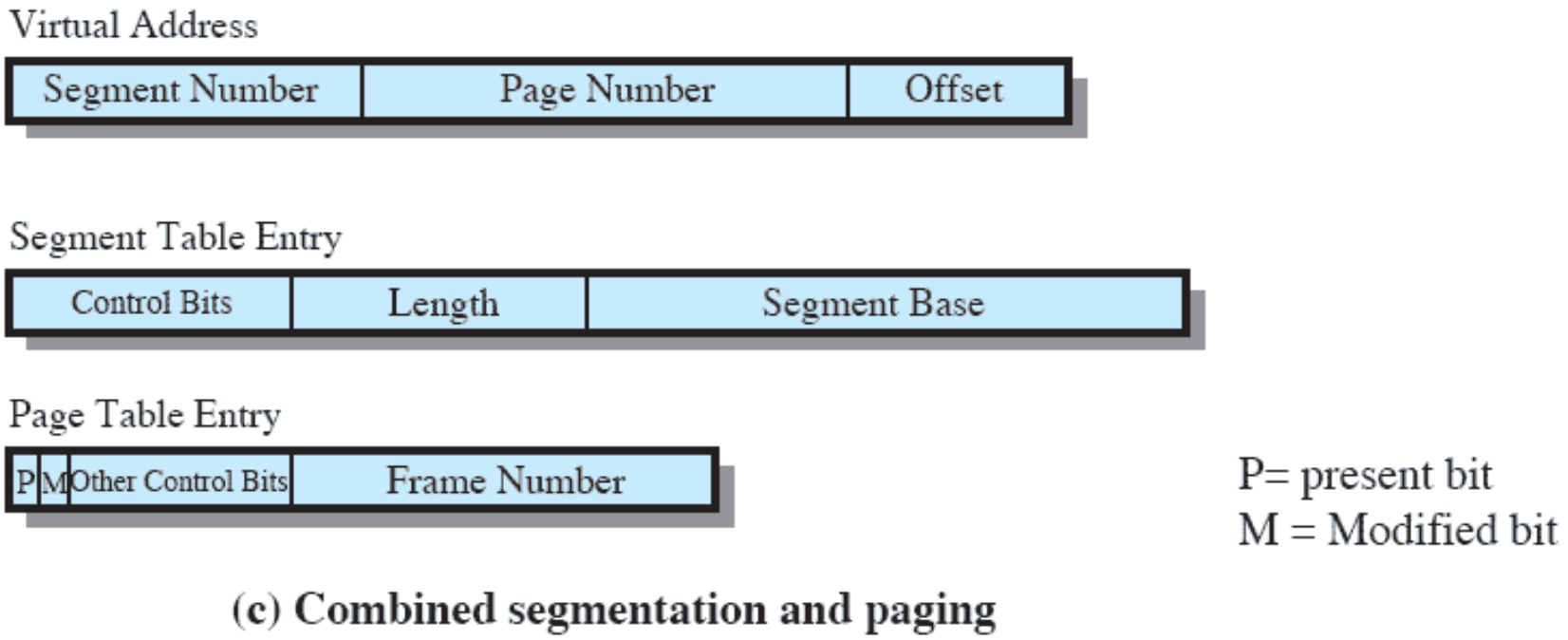
Combined Segmentation and Paging

- Single segment table
  - pointer to page table(base address -> page table)
  - Present & modify bits are needed as they are handled at page level.
  - Other control bits are used for sharing & protection
- Multiple page tables
  - Frame number for respective page number
  - Present ,modify ,sharing & protection bits are present.

Combined Segmentation and Paging

- User's address space is broken up into a number of segments, at the discretion of the programmer
- Each segment is then broken up into a number of fixed size pages
  - If segment is lesser, it just occupies one page
- Programmer's view logical address
  - Segment number and offset
- System's view
  - Segment offset = page number and page offset for a page within the specified segment.

Combined Paging and Segmentation



(c) Combined segmentation and paging

Address Translation

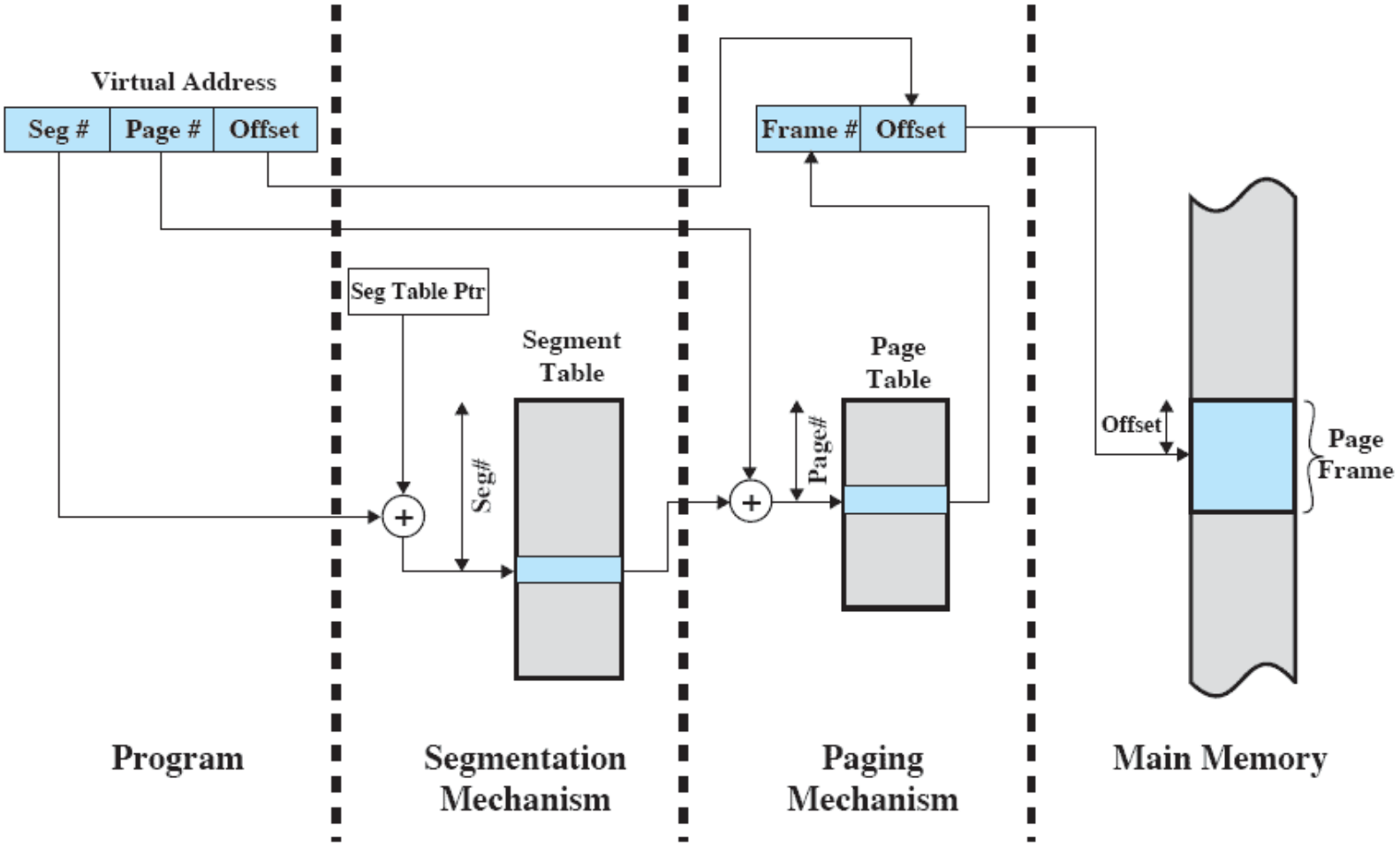


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection Relationships

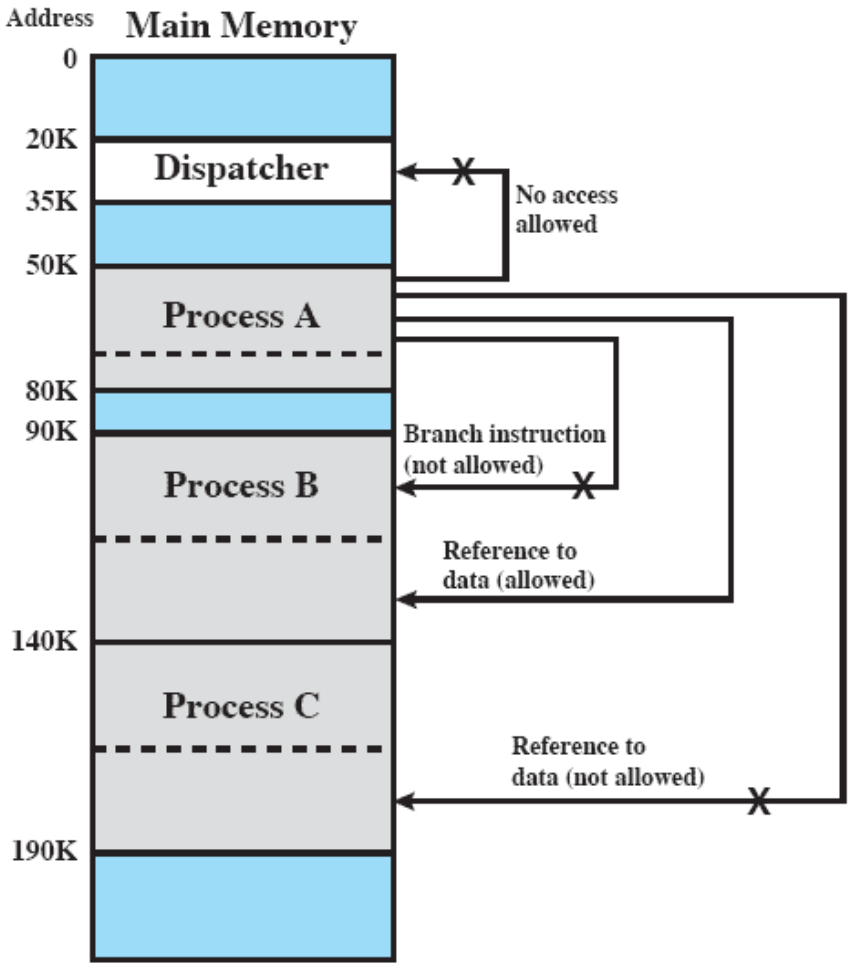


Figure 8.14 Protection Relationships Between Segments