Contents

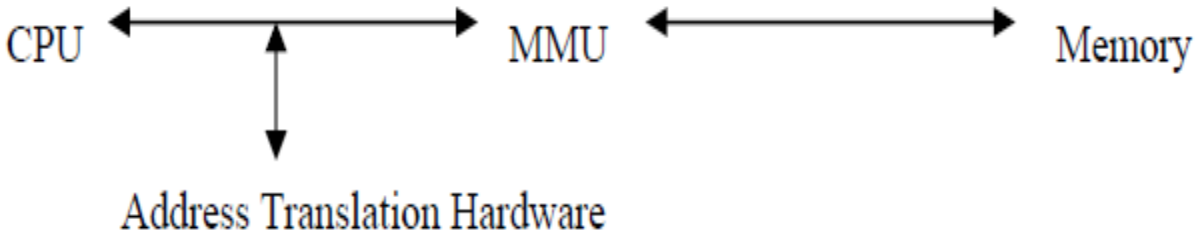
- Memory Management requirements
- Memory Partitioning: Fixed and Variable Partitioning,
- Allocation Strategies (First Fit, Best Fit, and Worst Fit), Fragmentation, Swapping.
- Paging, Segmentation, Address Translation,
- Page Replacement Policies (FIFO, LRU, Optimal, Other Strategies),
- Thrashing.

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

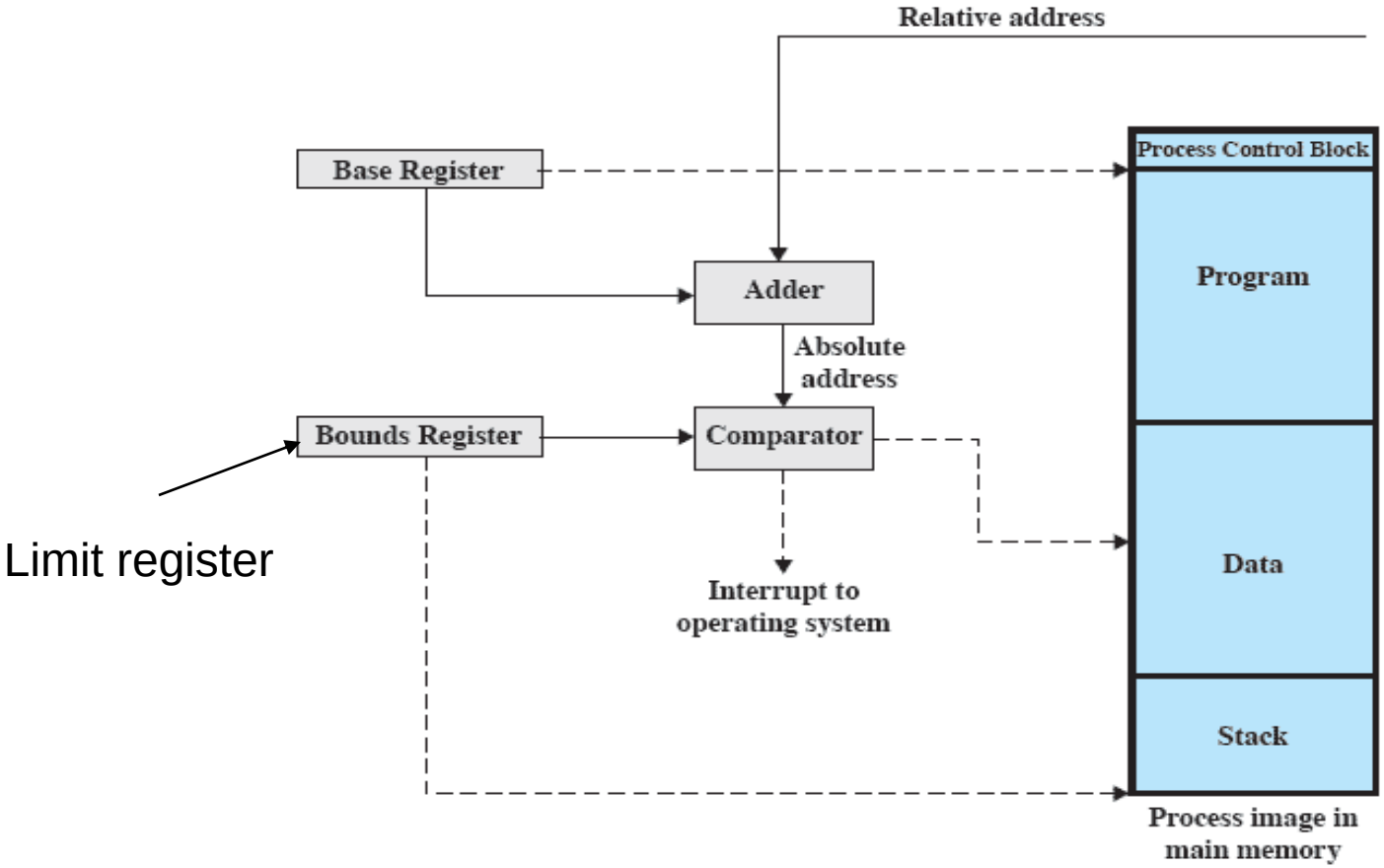


Figure 7.8 Hardware Support for Relocation

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

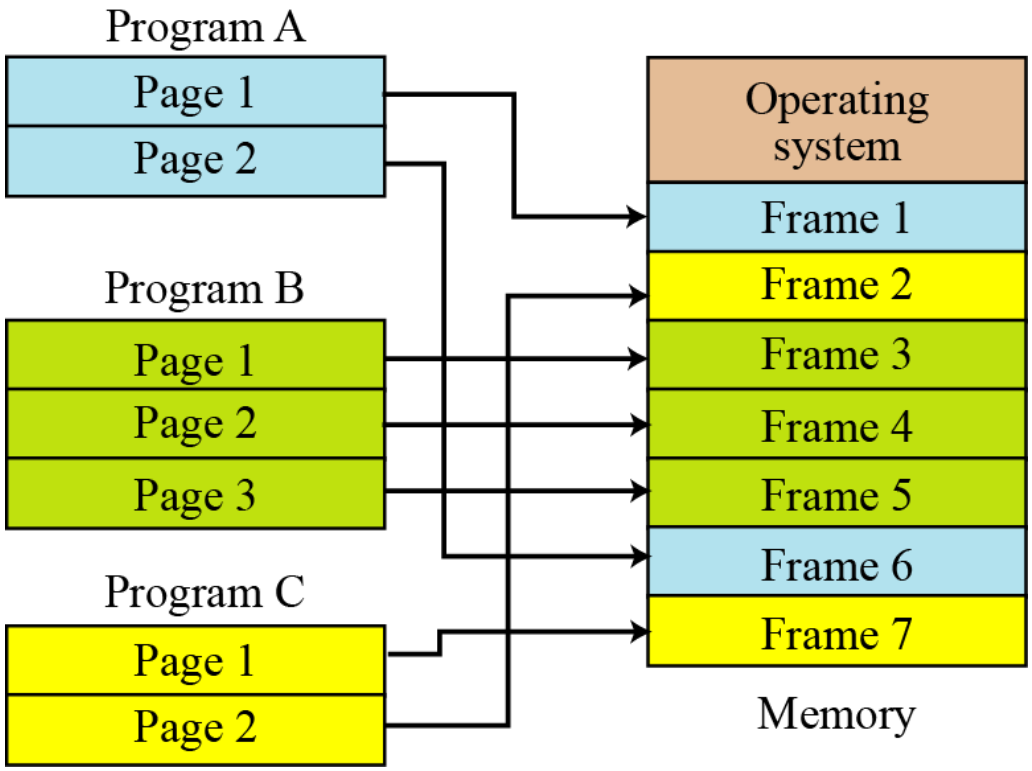
- Fixed & variable size partitions are insufficient as involves internal / external fragmentation.→ contiguous
- What were the two problems with equal sized fixed partitions?
  - Program too large for a partition
  - Program too small for a partition
- Partition memory into small equal fixed-size chunks 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

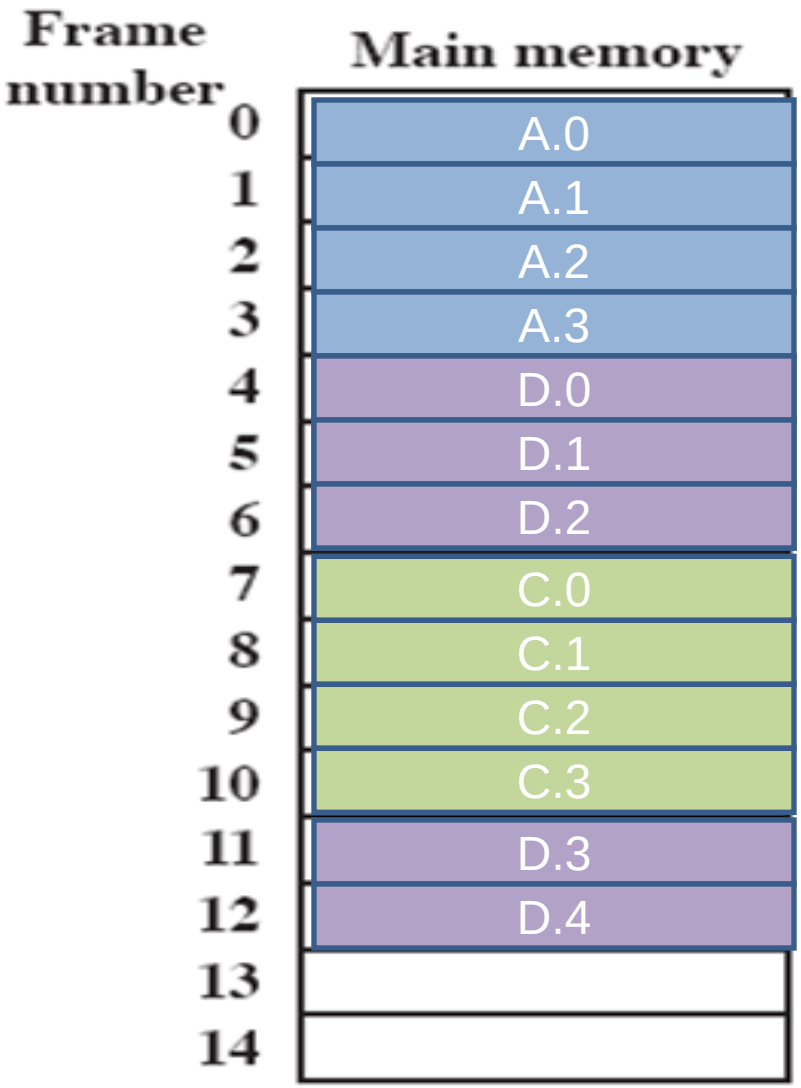
- Each process has its own page table.
- Each page table entry contains the frame number of the corresponding page in main memory
- A bit is needed to indicate whether the page is in main memory or not.
- No external fragmentation
- all frames (physical memory) can be used by processes
- Possibility 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

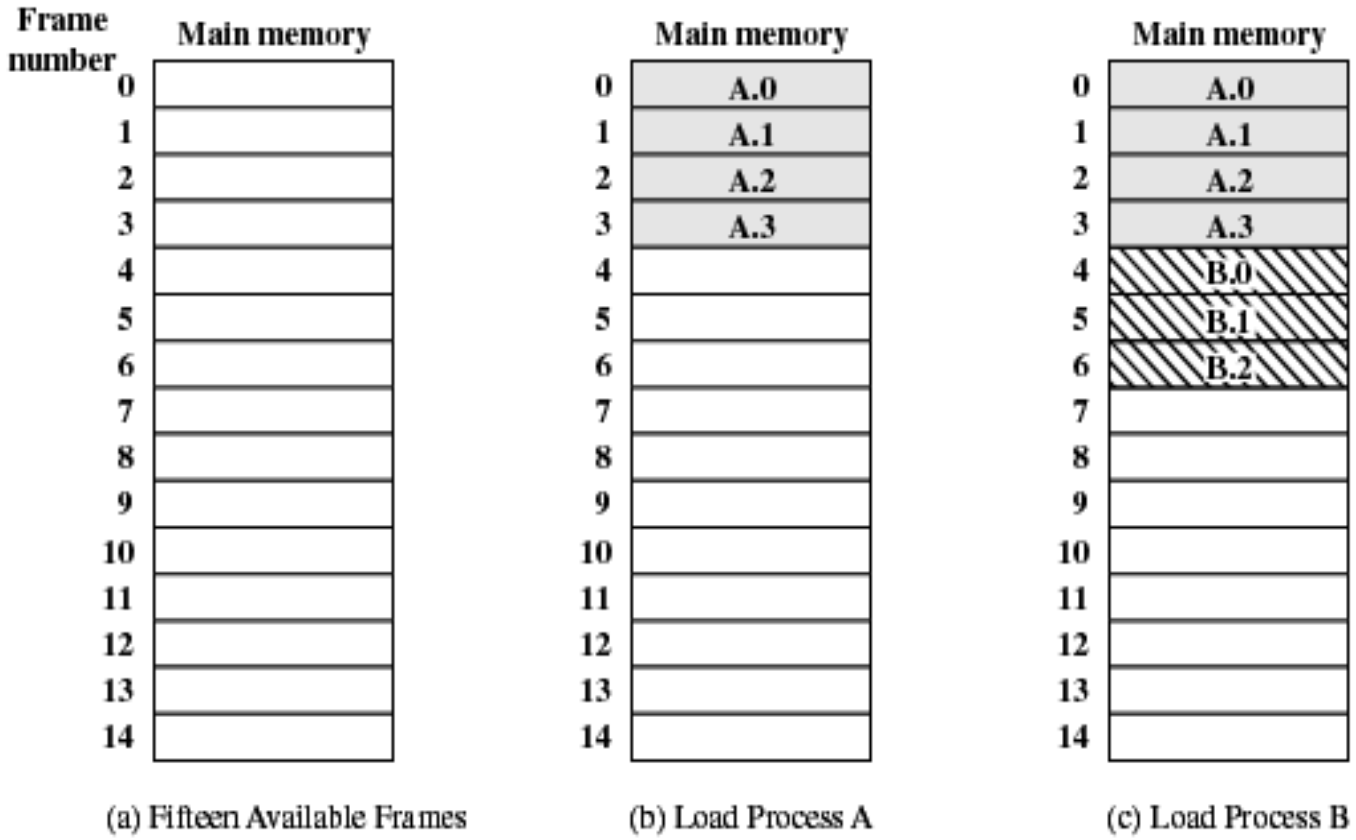
- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory


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

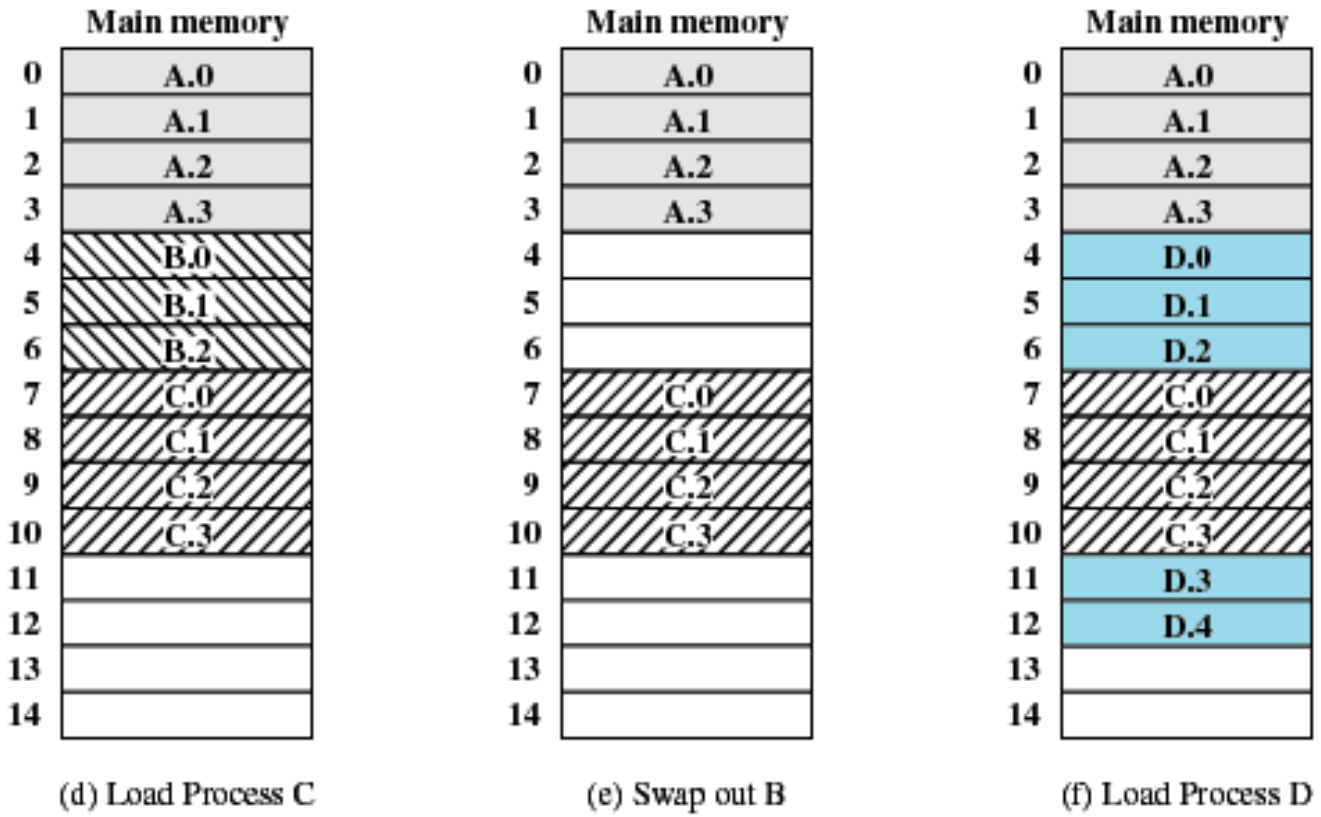


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

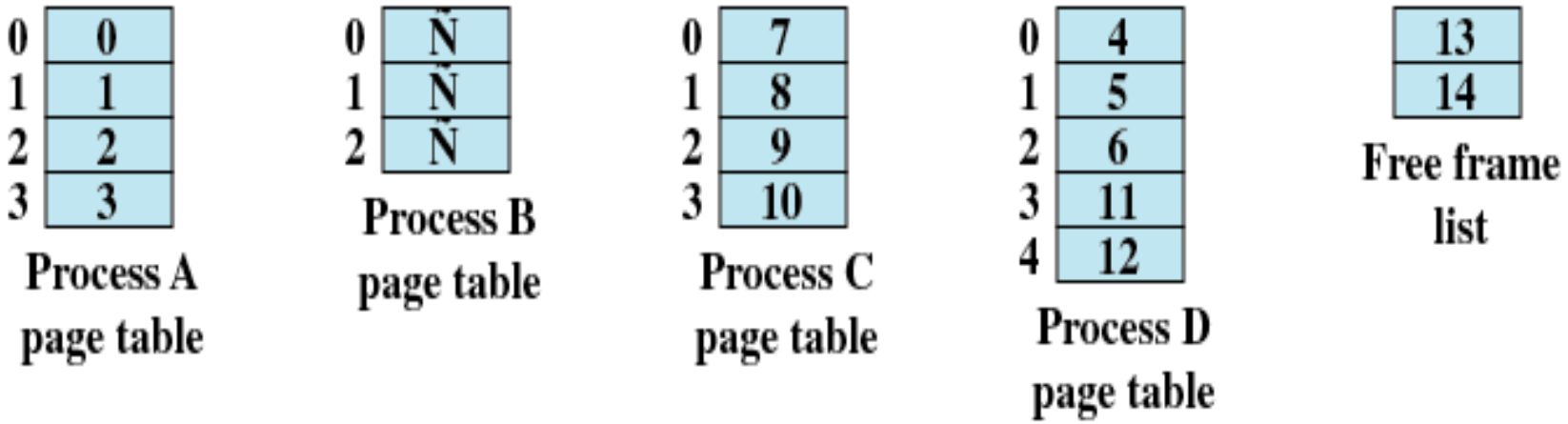
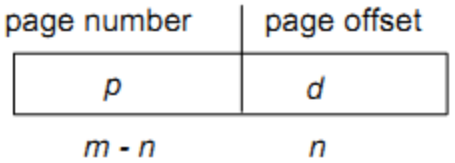


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging

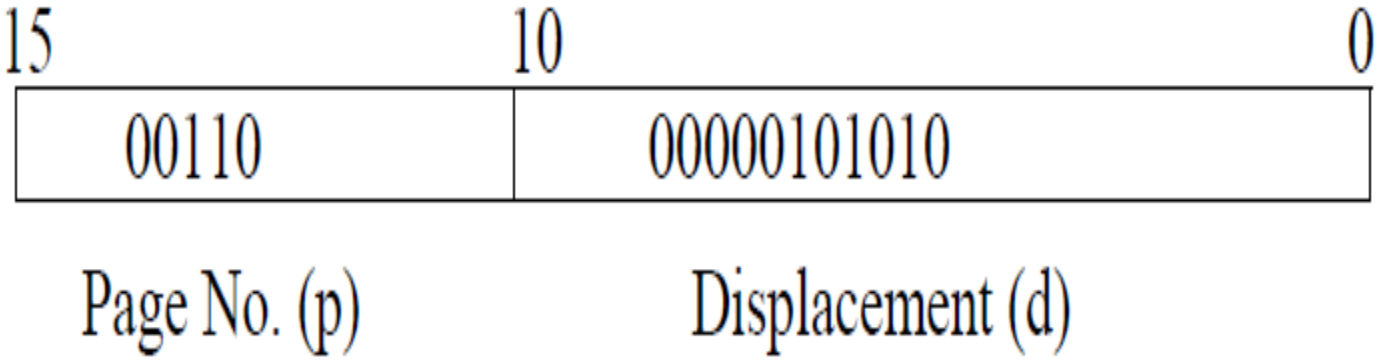
- The page size is defined by the hardware.
- The page size is typically a power of 2.The size can be 512 bytes to 16 MB.
- The selection of power of 2 as a page size makes translation of logical address into a page number and page offset easy.
- Address generated by CPU is divided into:
- Page number (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

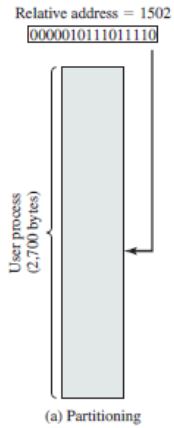
Assume 16 bit address



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.

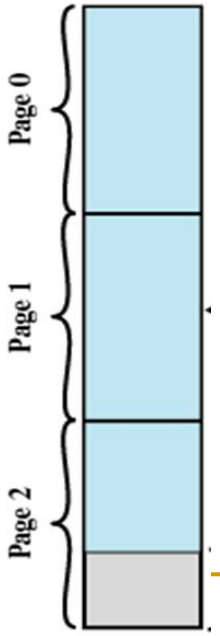
10111011110

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,

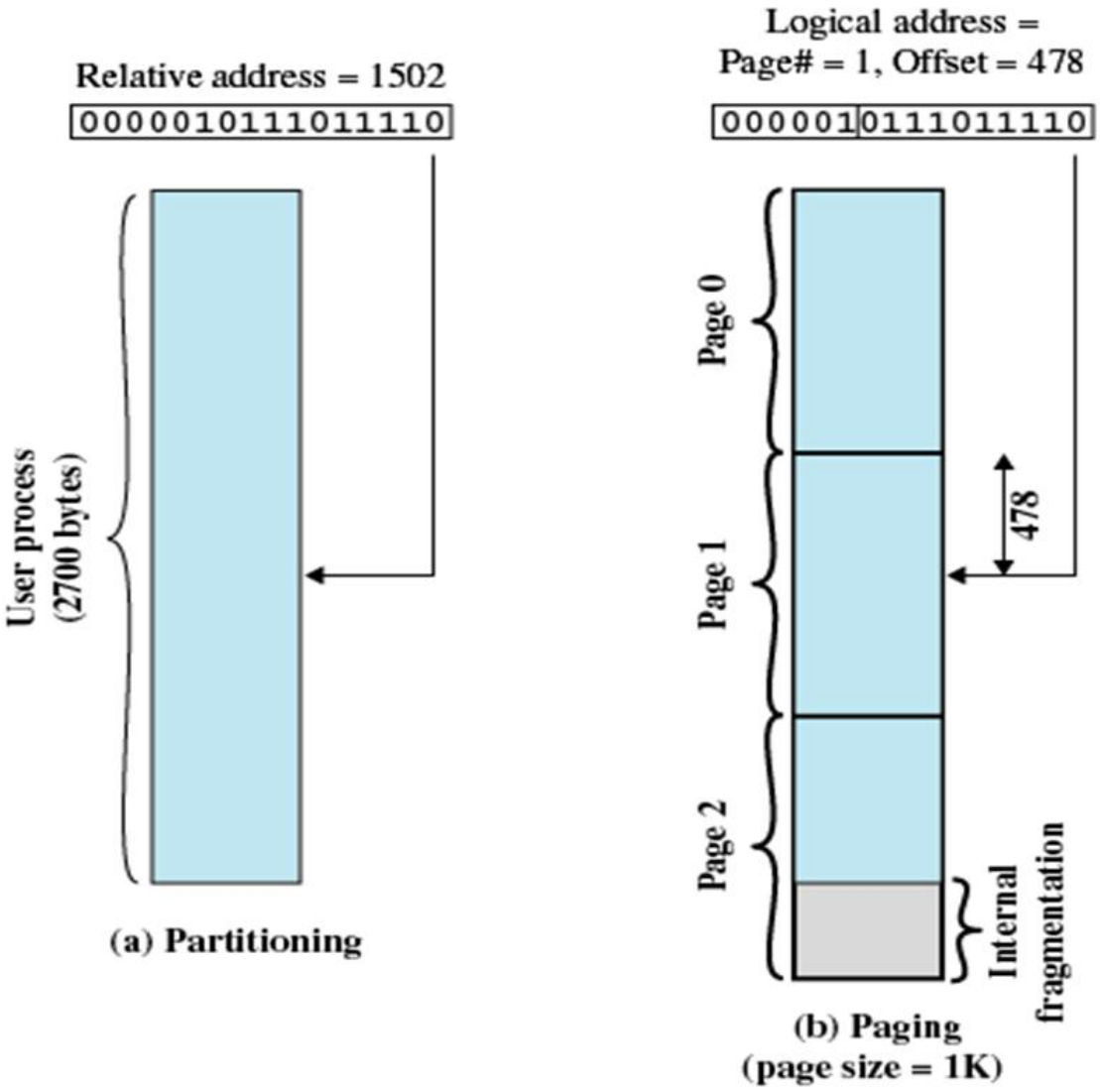
leaving 6 bits for the page number.

- 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),

which yields the same 16-bit number, 0000010111011110.

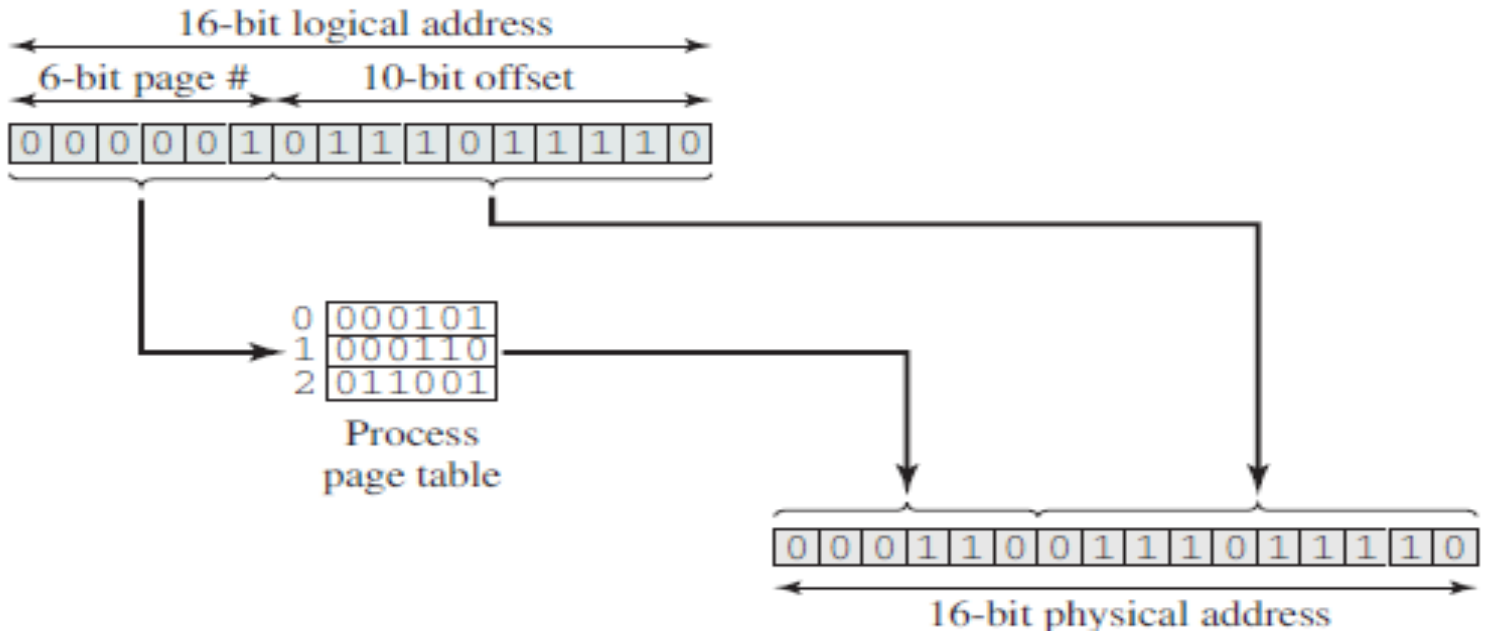
Illustration



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

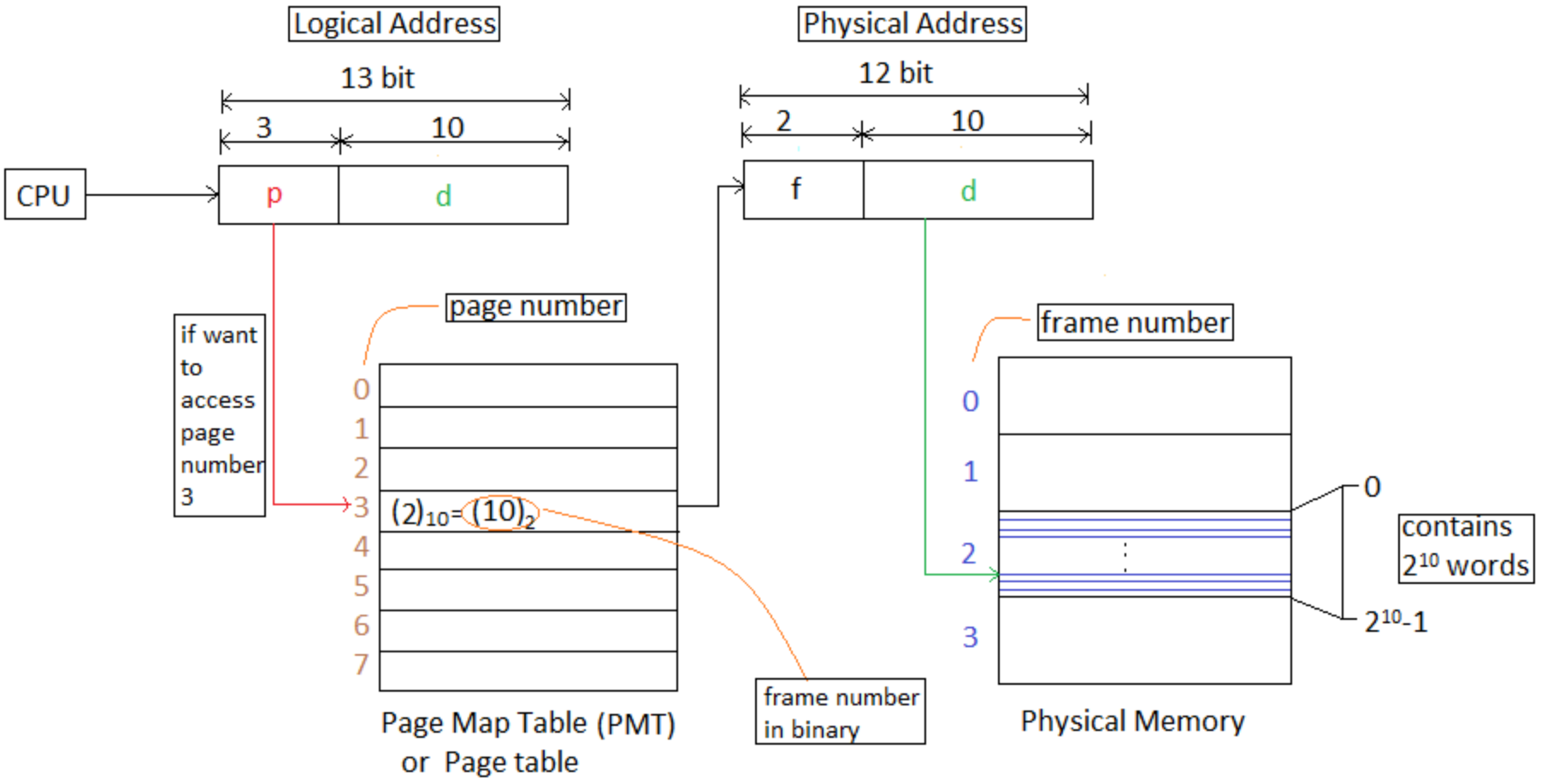
(6*2^10=6144+478=6622 (000110111011110)

Let us consider an example:

- Physical Address = 12 bits, then Physical Address Space = 4 K words
- Logical Address = 13 bits, then Logical Address Space = 8 K words
- Page size = frame size = 1 K words (assumption)=2^10

Number of frames = Physical Address Space / Frame size = 4 K / 1 K = 4 = 2^2

Number of pages = Logical Address Space / Page size = 8 K / 1 K = 8 = 2^3



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

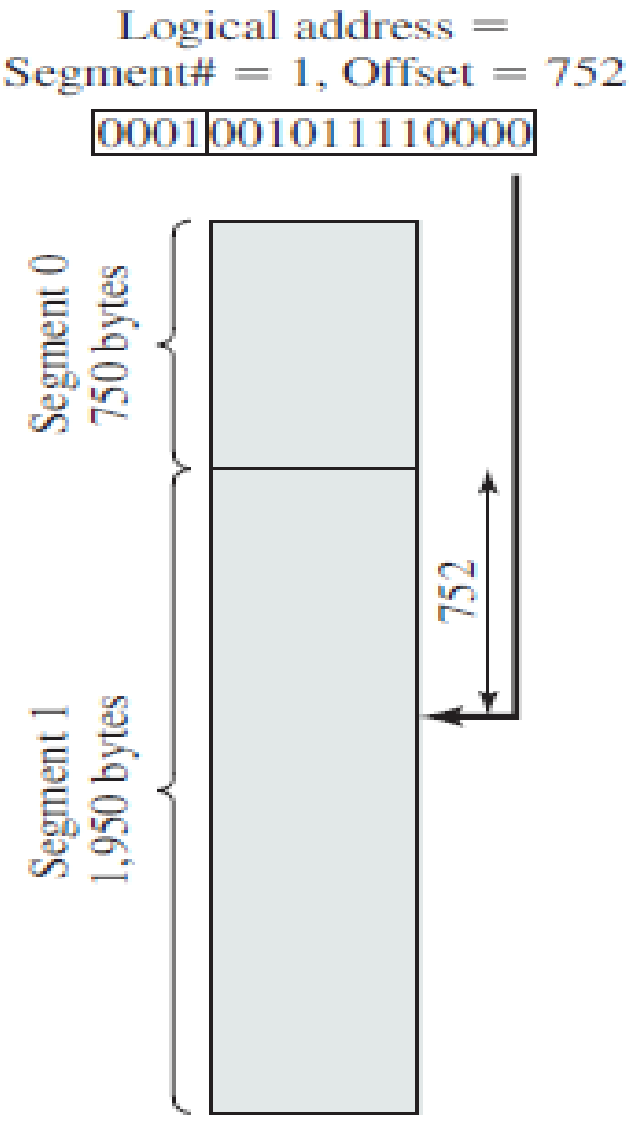
Segment Organization

- 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

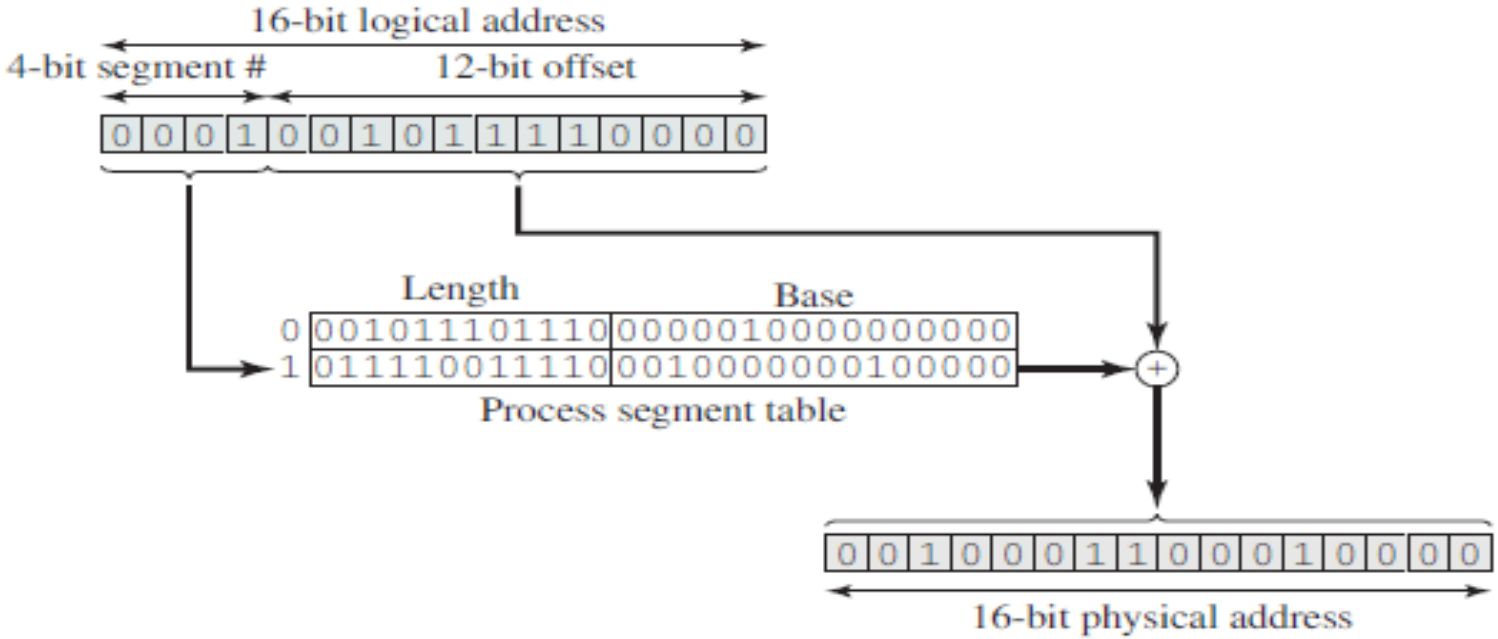
And m =12.



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 +

001011110000 =0010001100010000

Example

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

a) How many bits are there in the logical address?

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

a) 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.

a) 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*2^10=2^12
- 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

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

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

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

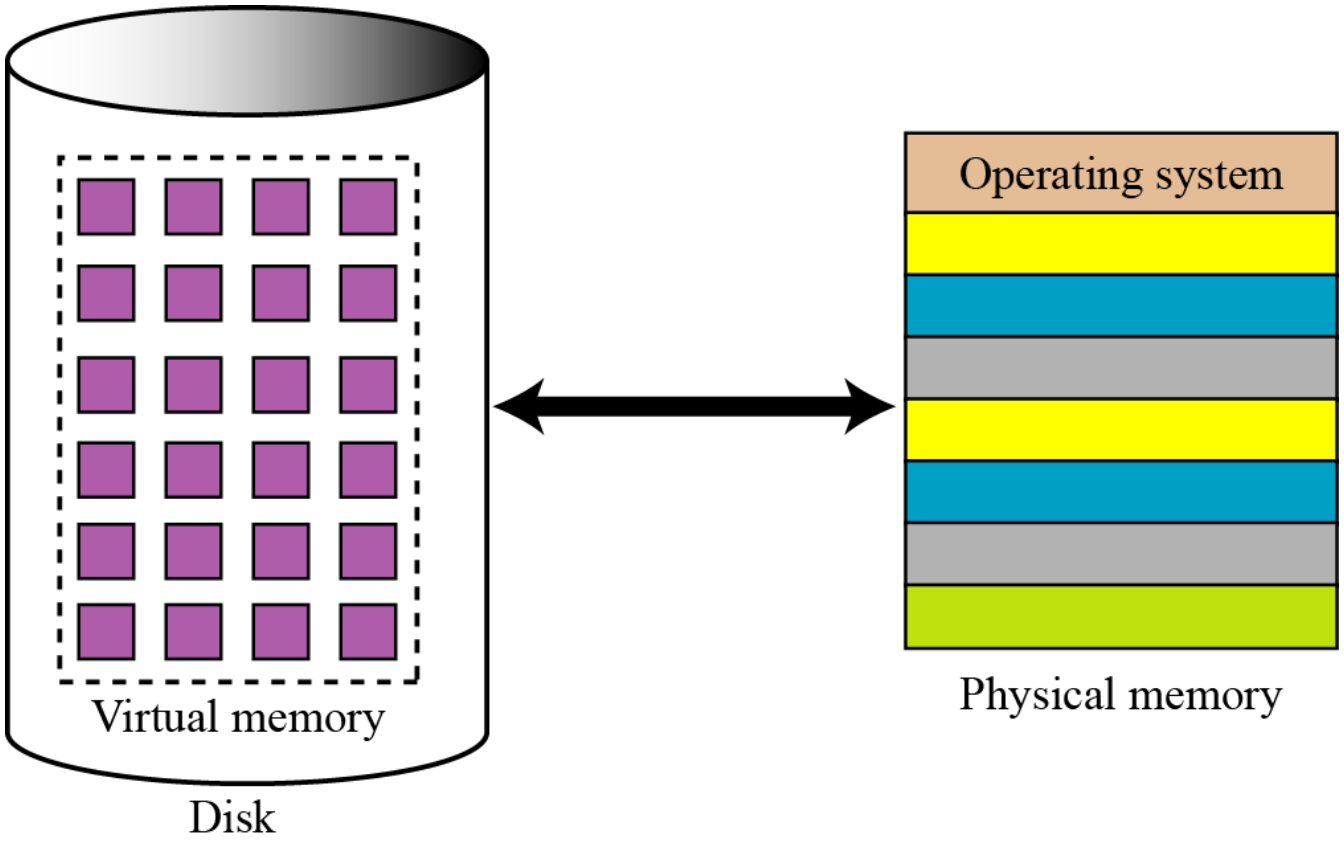
Virtual Memory

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.

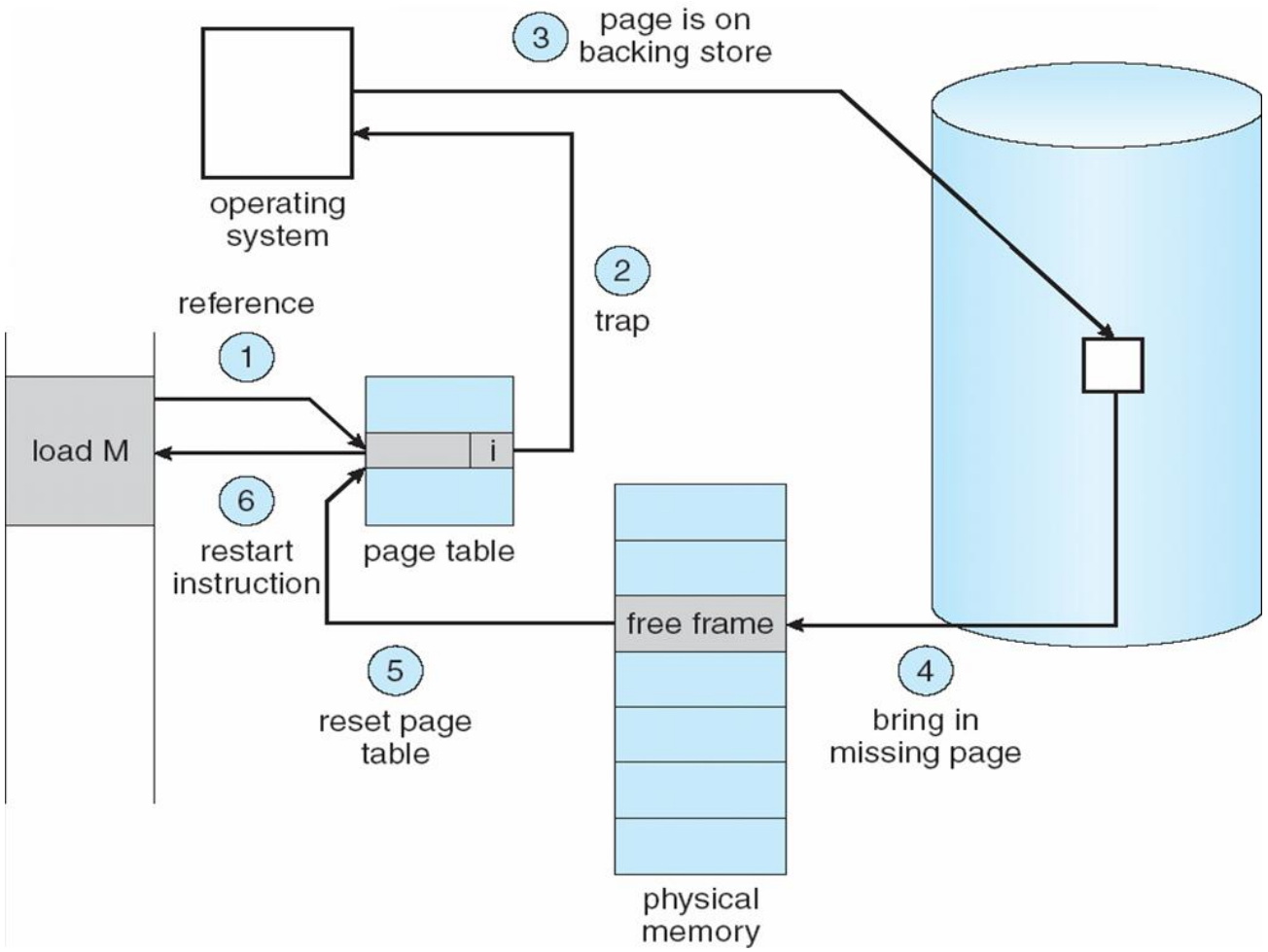
Virtual Memory



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

Steps in handling a page fault



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

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.

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

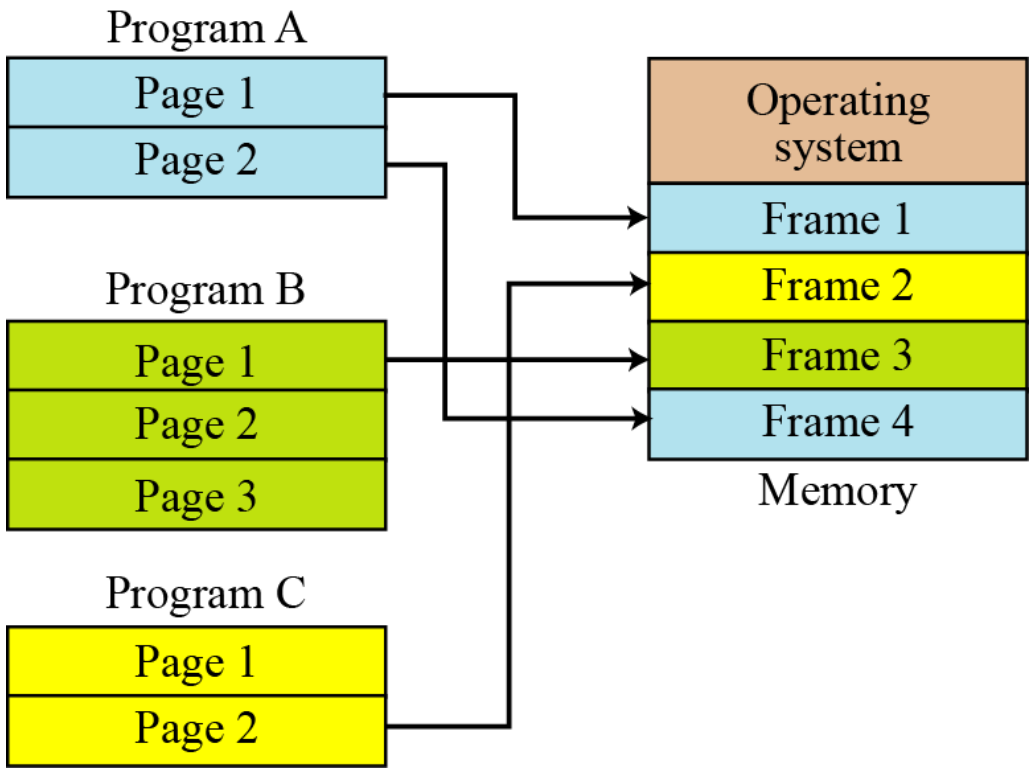
Virtual Memory Requirements

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

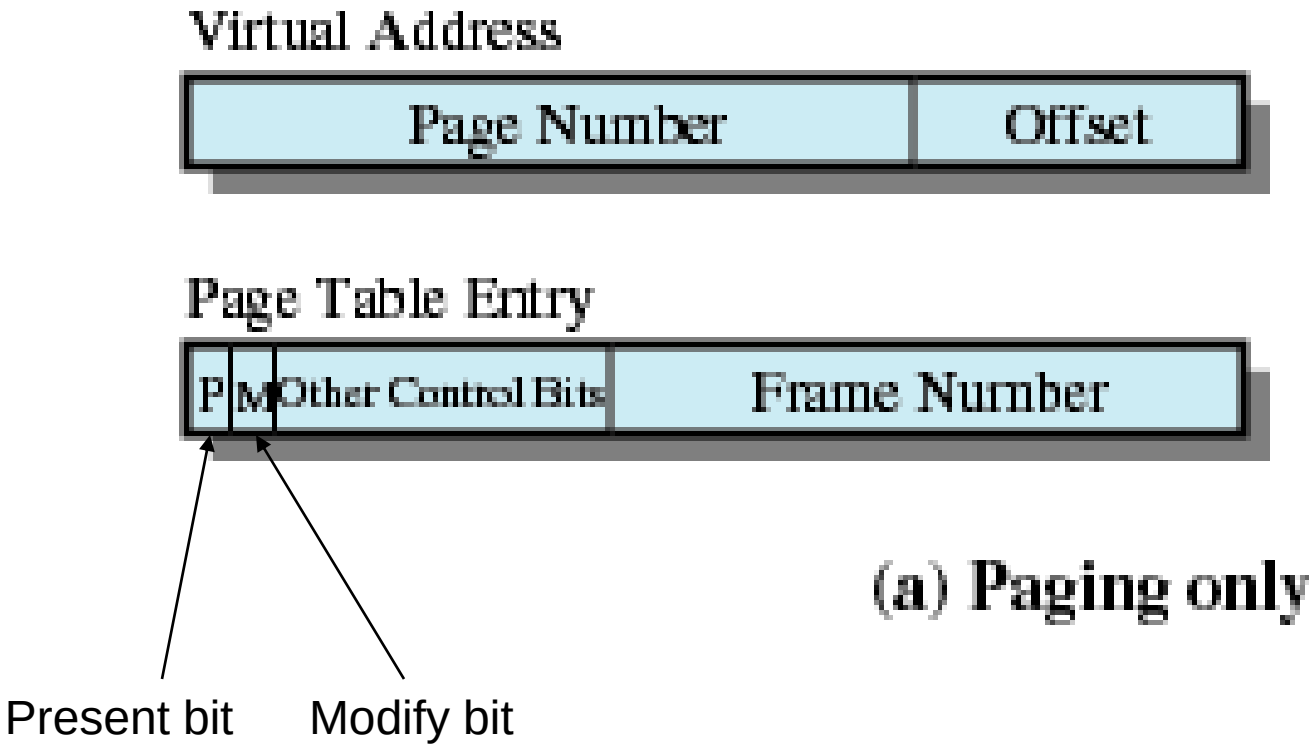
Virtual Memory + Paging

- 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



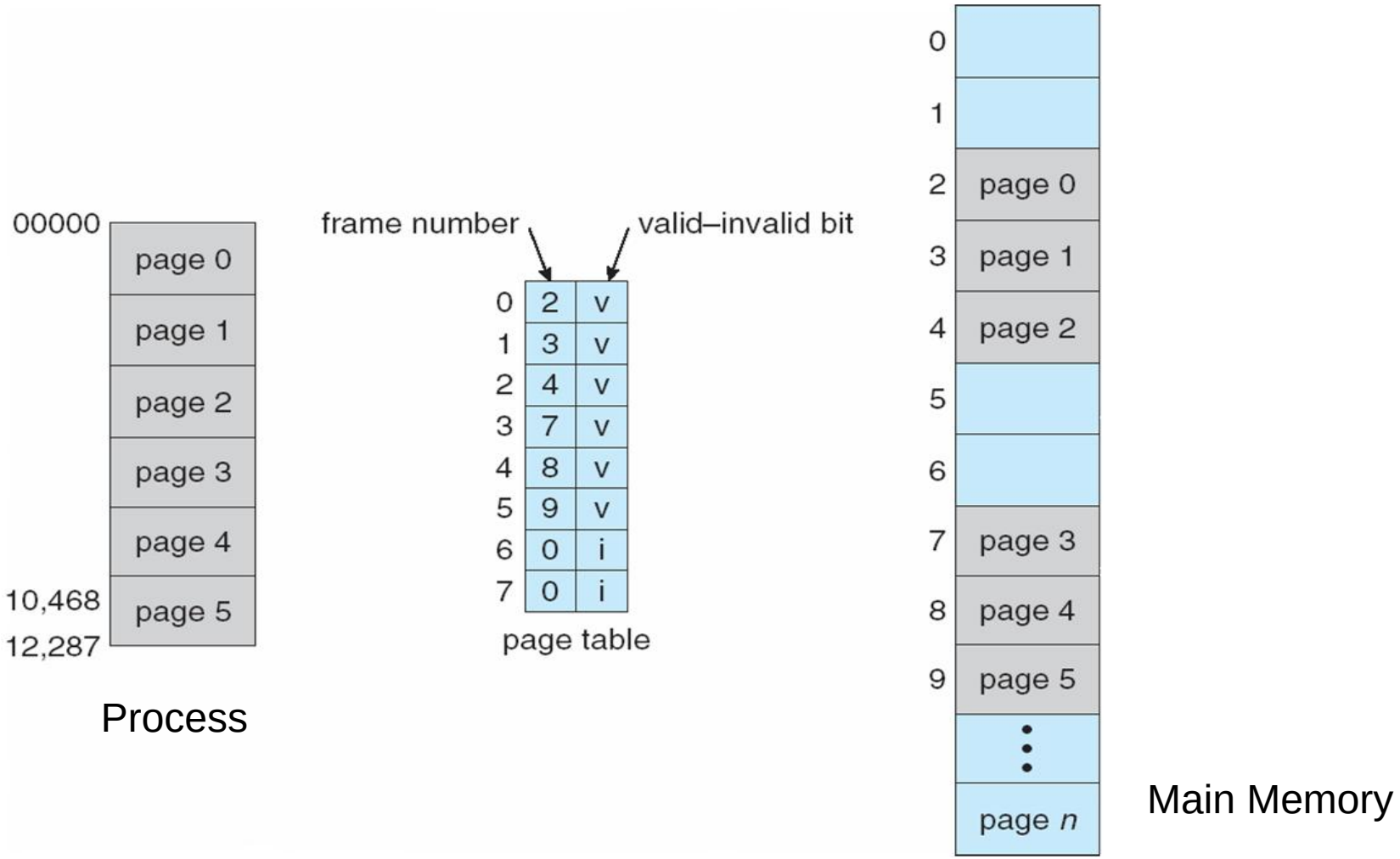
Page Table Entry



Page protection

- Implemented by associating protection bits with each virtual page in page table
- Protection bits
1. present bit: map to a valid physical page?
2. read/write/execute bits: can read/write/execute?
3. user bit: can access in user mode?
- x86: PTE_P, PTE_W, PTE_U
- Checked by MMU on each memory access

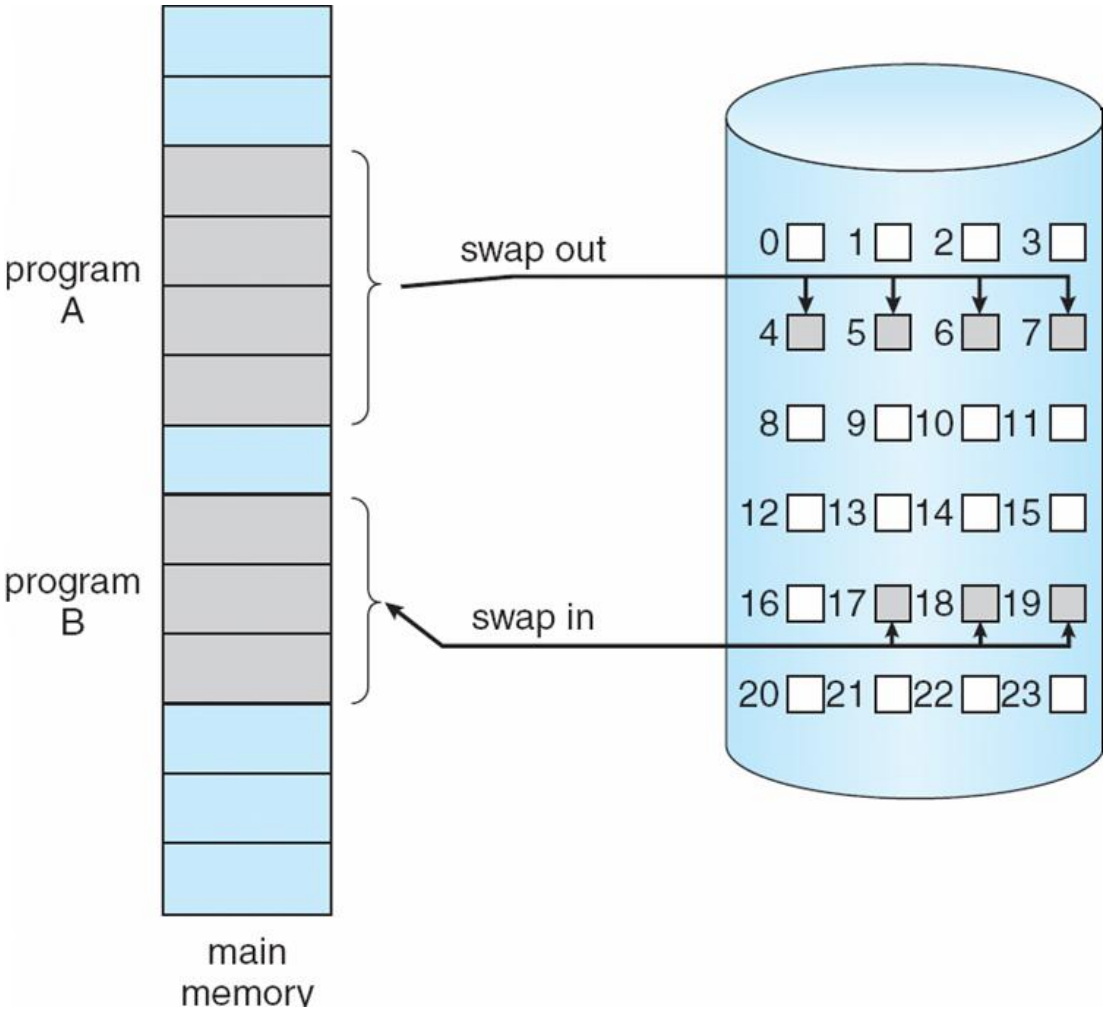
Example:Valid (v) or Invalid (i) Bit In A Page Table



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

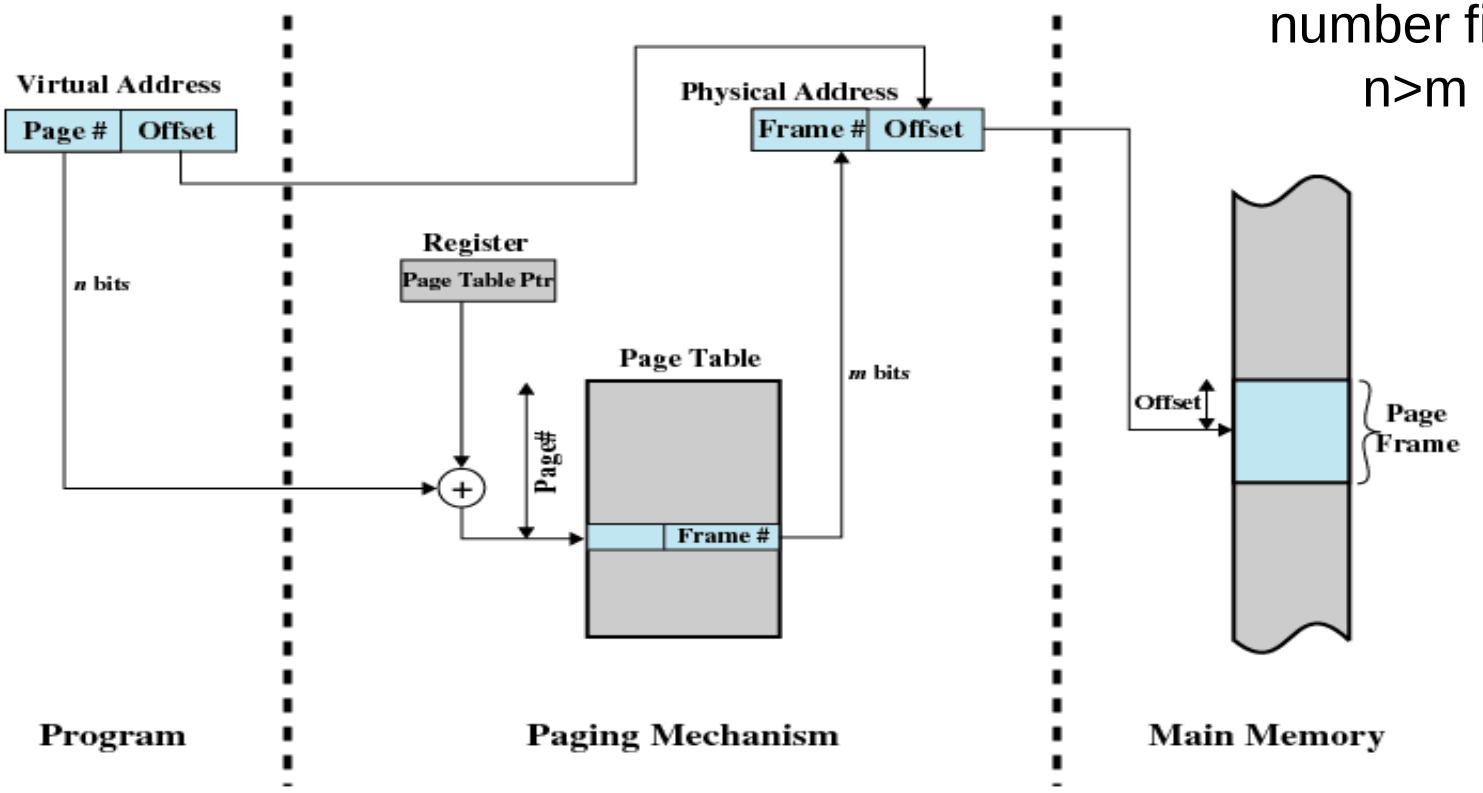


Figure 8.3 Address Translation in a Paging System

Page Tables

- Generally one page table per process
- Let us say a process requires 2GB virtual memory
- How many 512 bytes pages will it contain?
- 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.

2^31 pages /process

2^31/2^9=2^22 pages /process

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

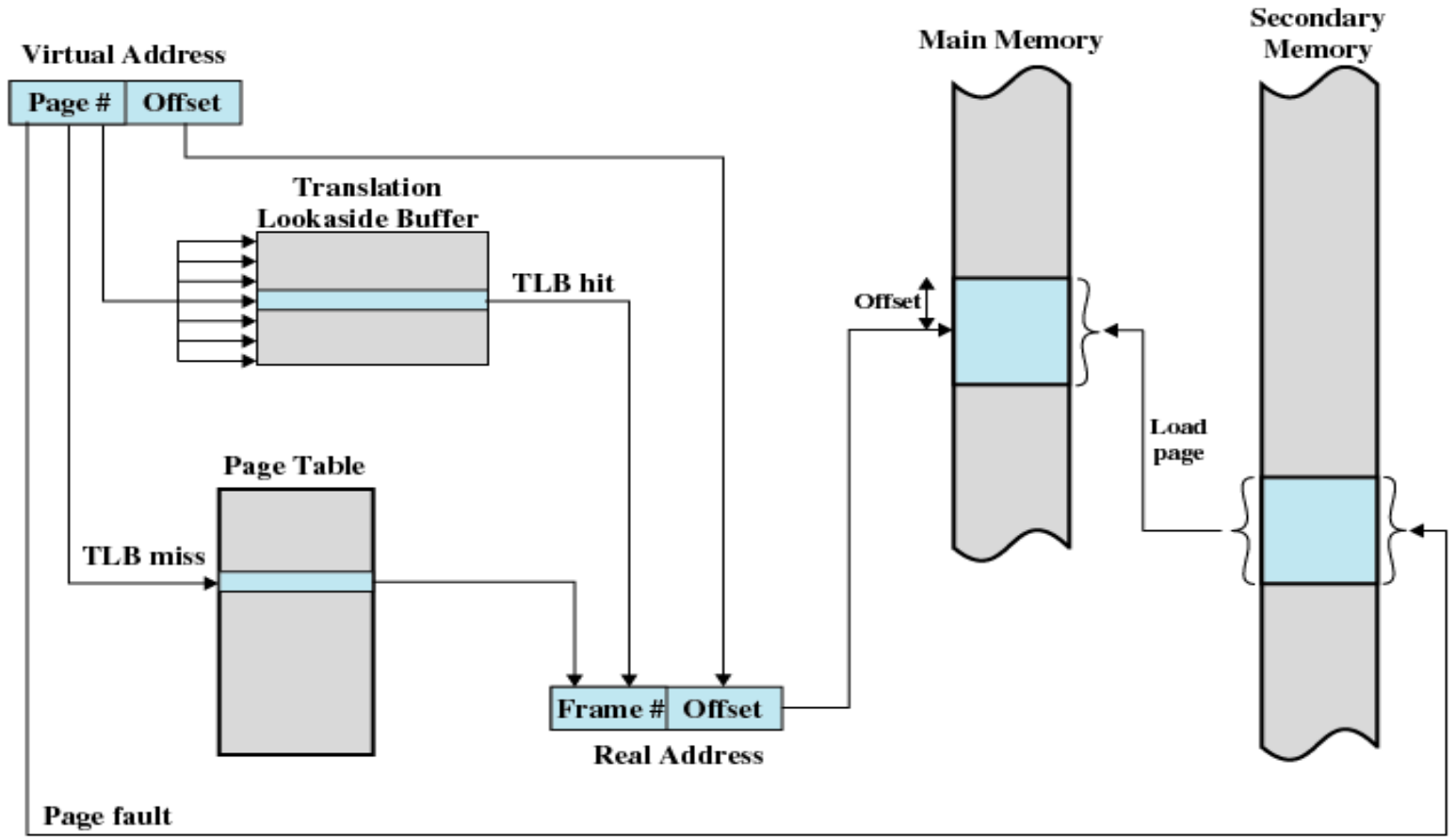


Figure 8.7 Use of a Translation Lookaside Buffer

TLB Operation

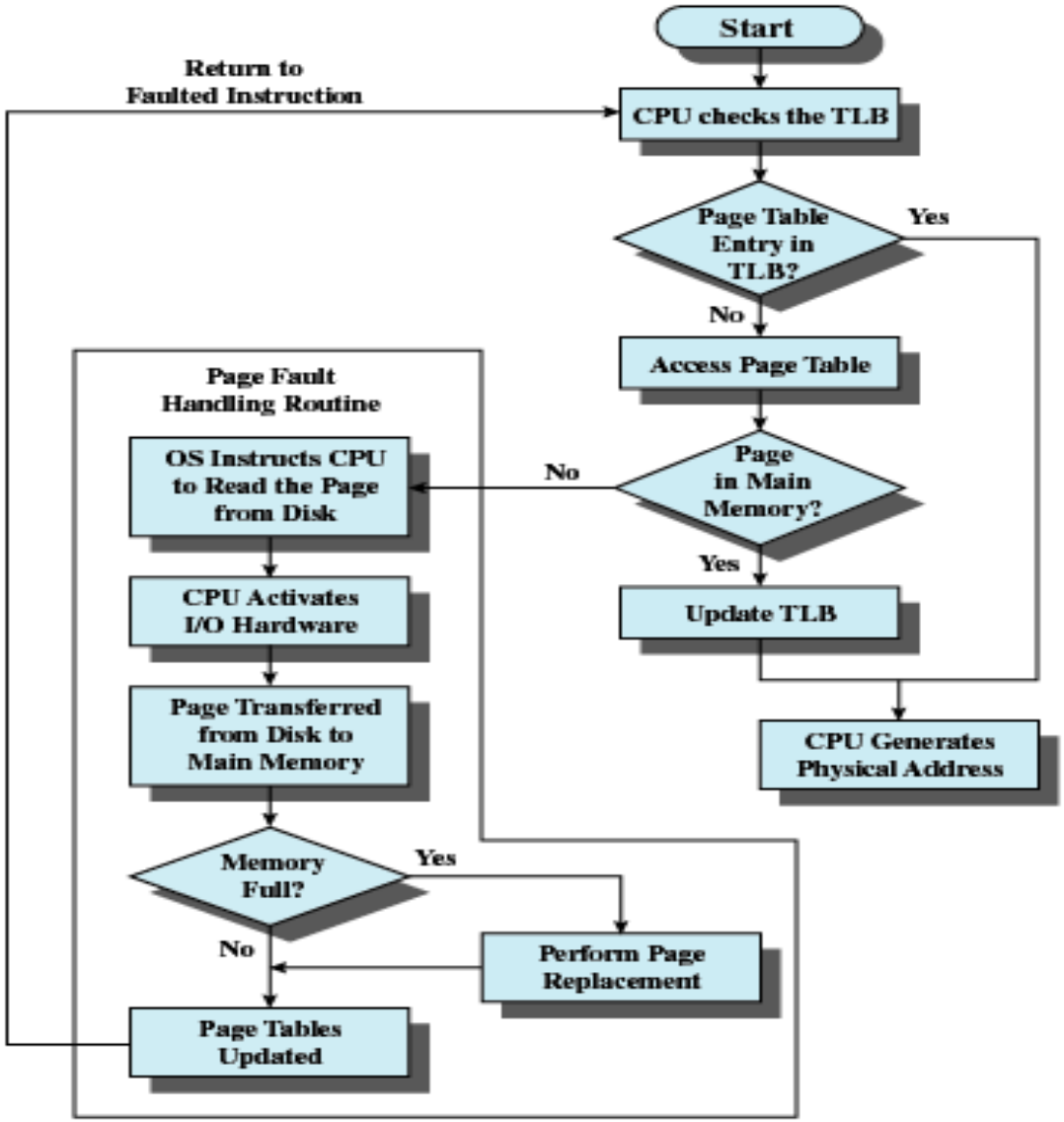


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Translation Lookaside Buffer

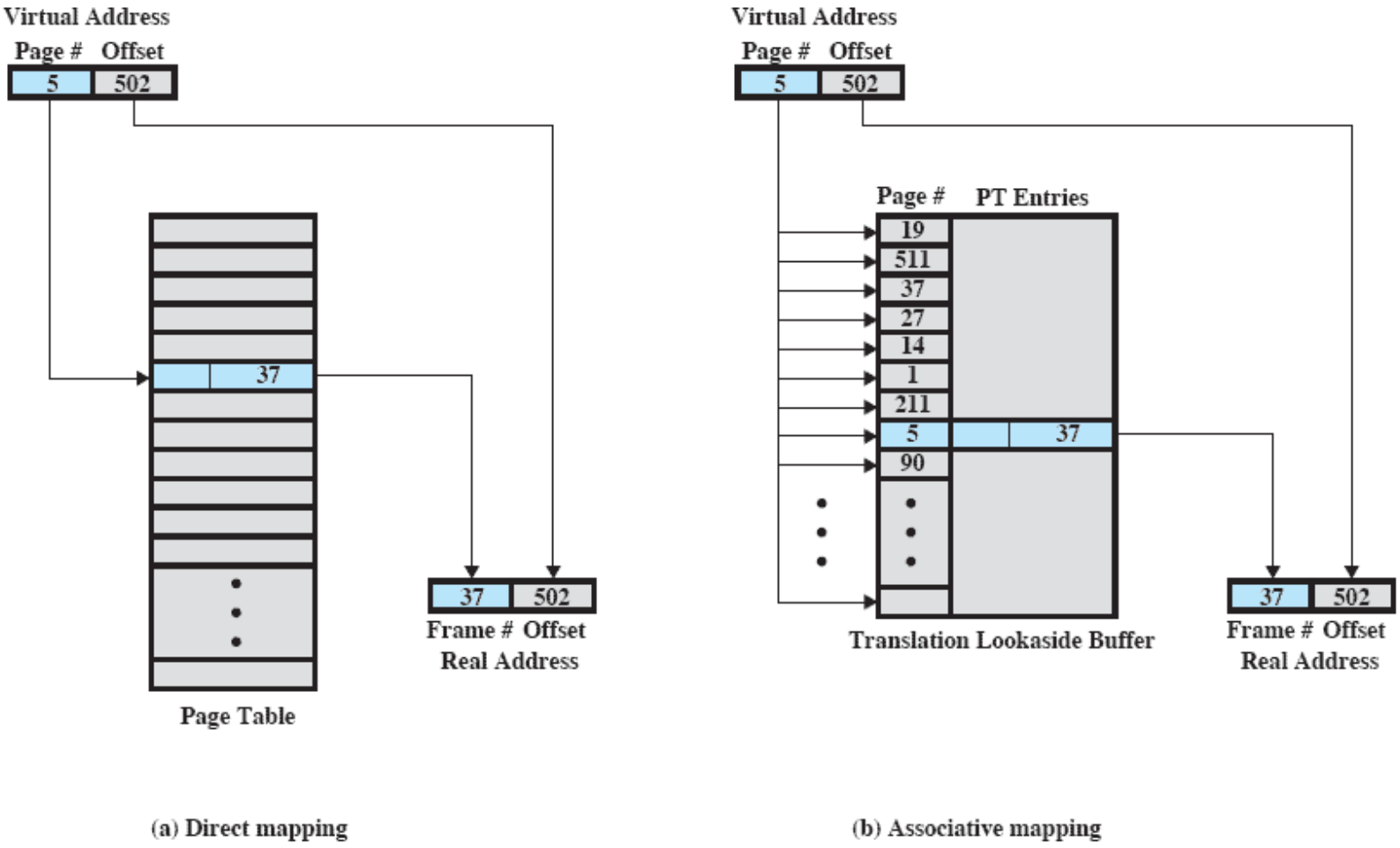


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

Translation Lookaside Buffer

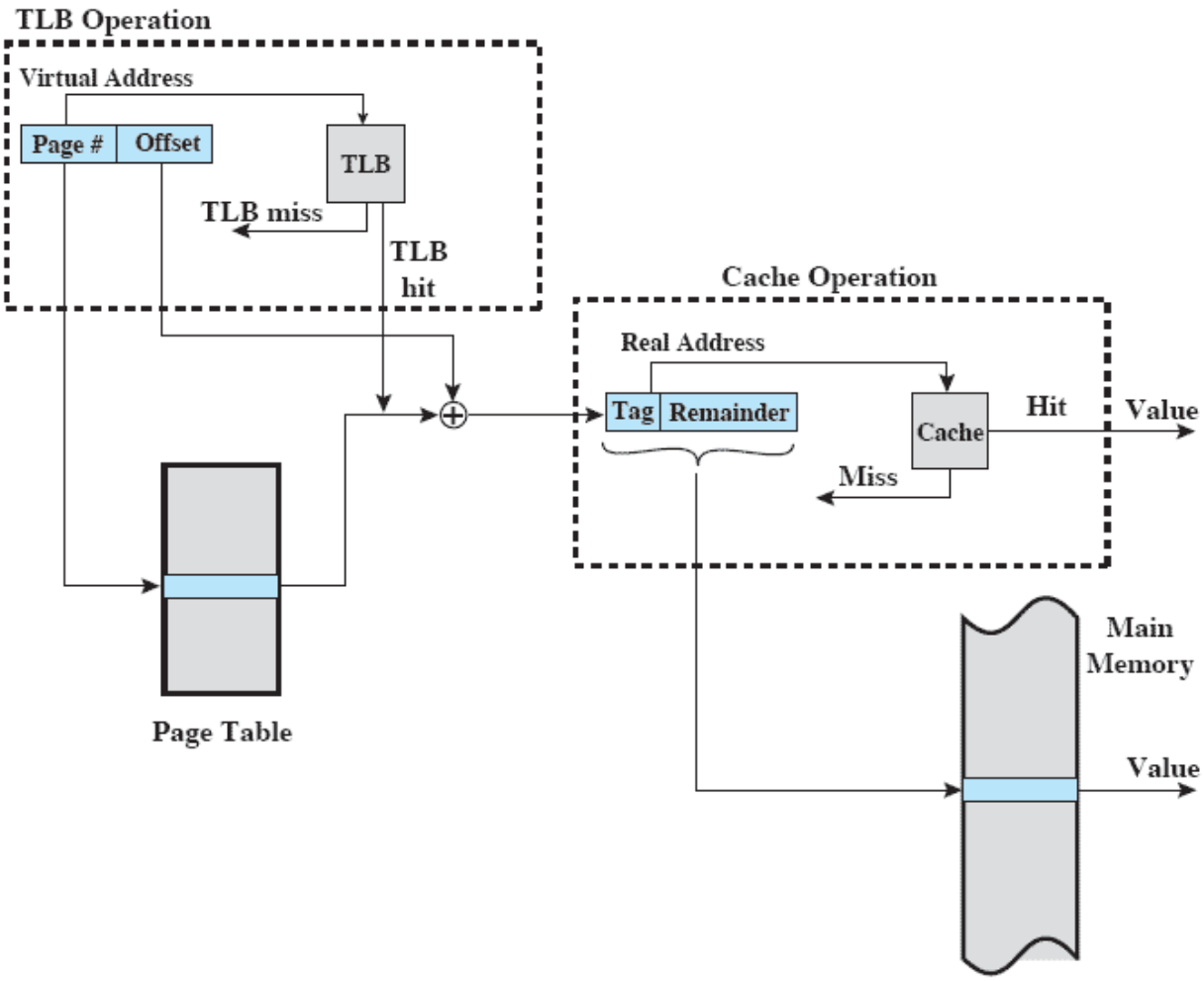


Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page 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

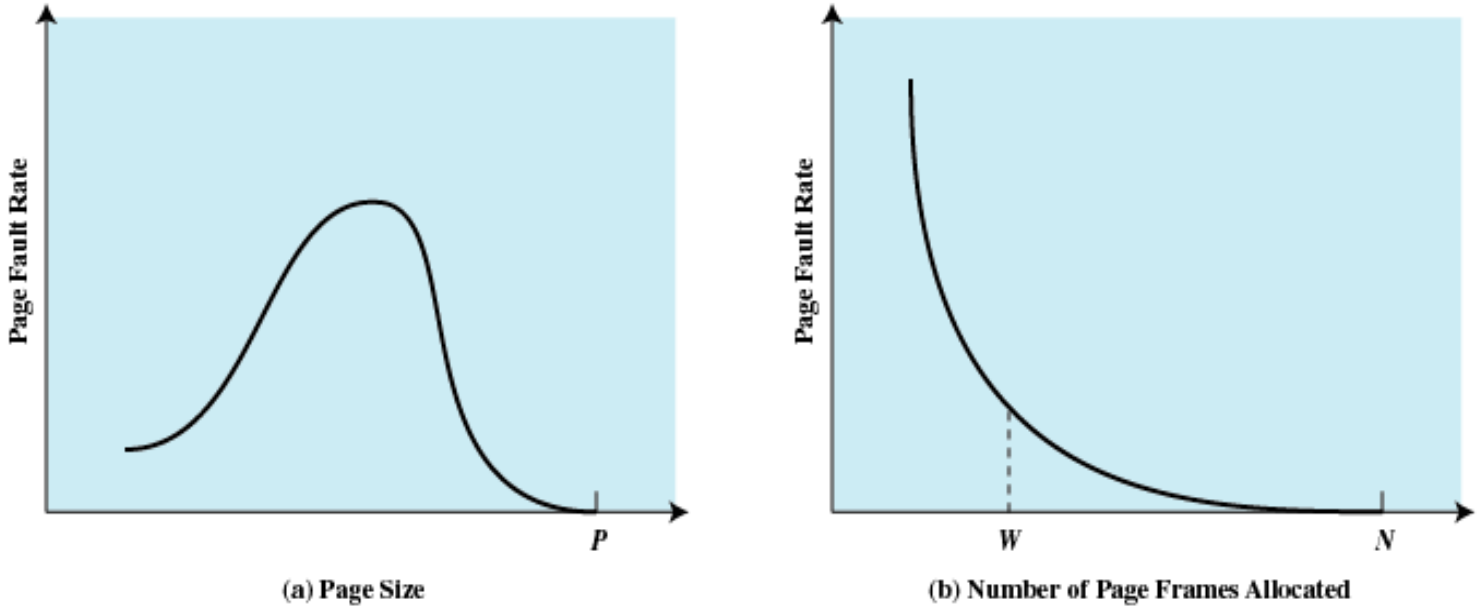
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better
- Small page size, large number of pages will be found in main memory
- As time goes on during execution, the pages in memory will all contain portions of the process near recent references => Page 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

1. Easier to relocate segment than entire program
2. Simplifies handling of growing data structures
3. Allows programs to be altered and recompiled independently
4. Sharing among processes
5. Provides protection
6. Avoids allocating unused memory (no internal fragmentation)
7. Efficient translation -> Segment table small (fit in MMU)

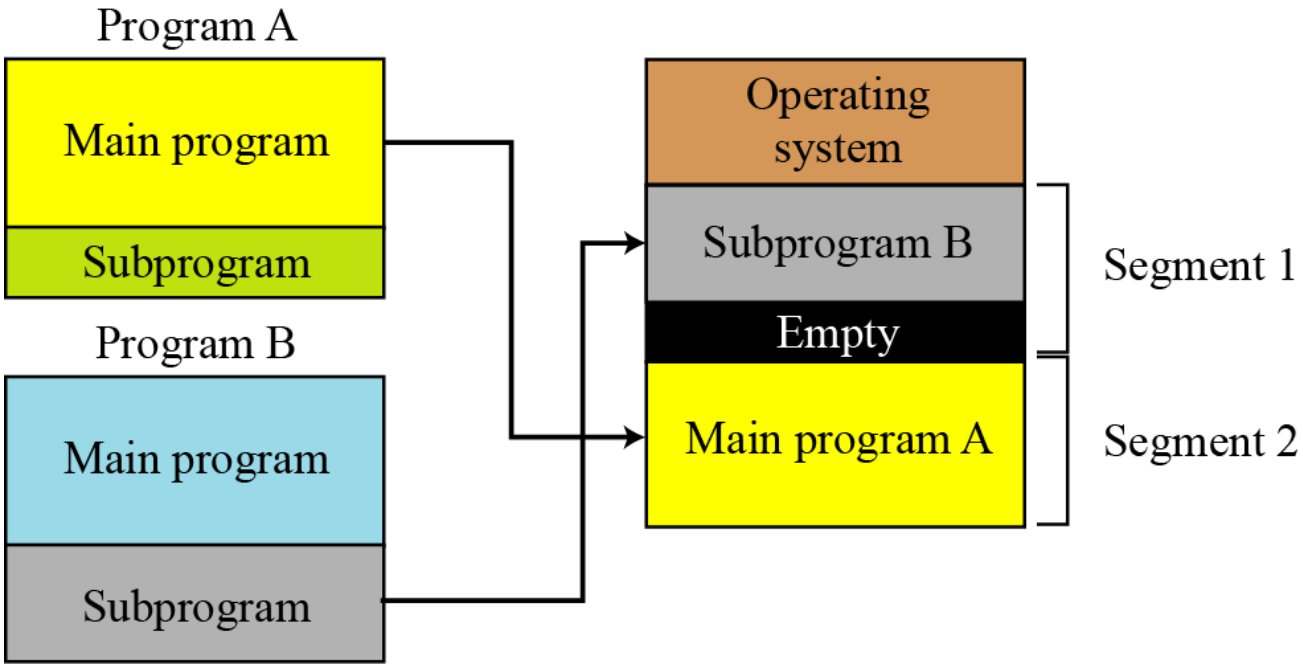
- Disadvantages

1. Segments have variable lengths -> how to fit?
2. Segments can be large -> external fragmentation

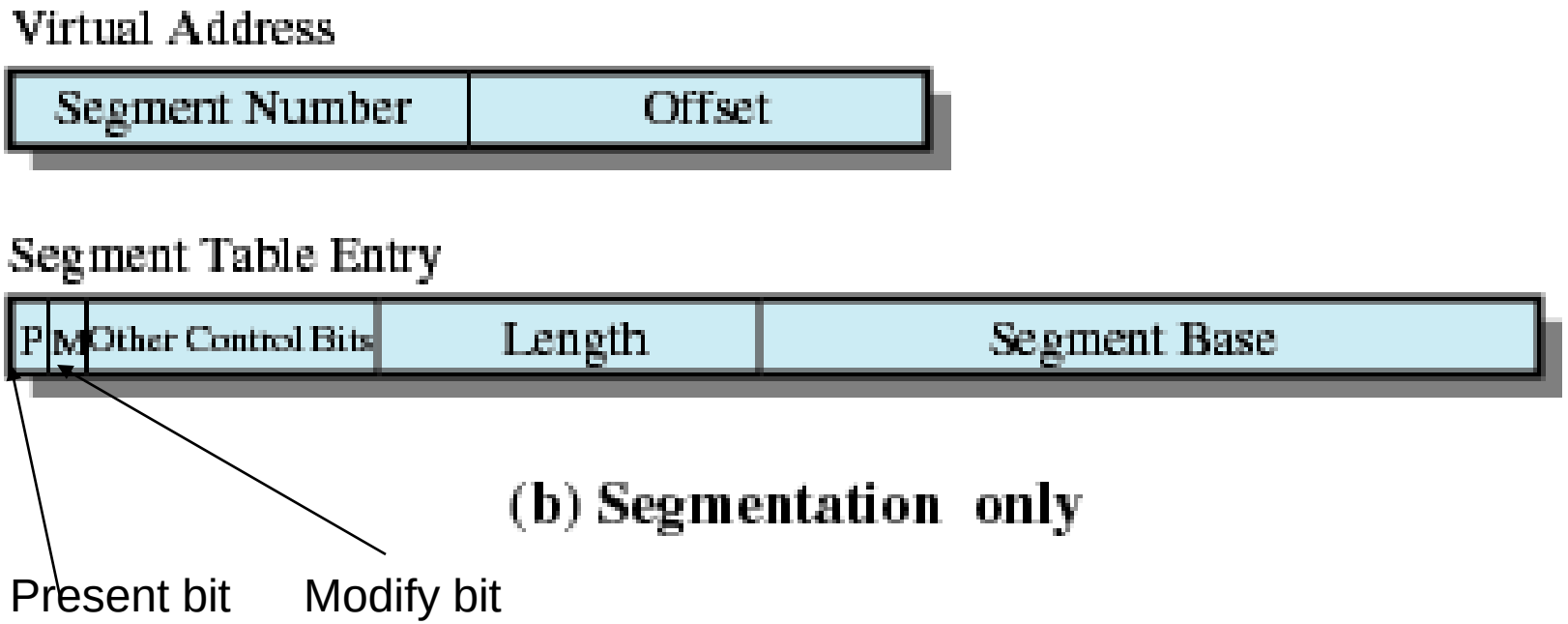
VM + Segmentation

- 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

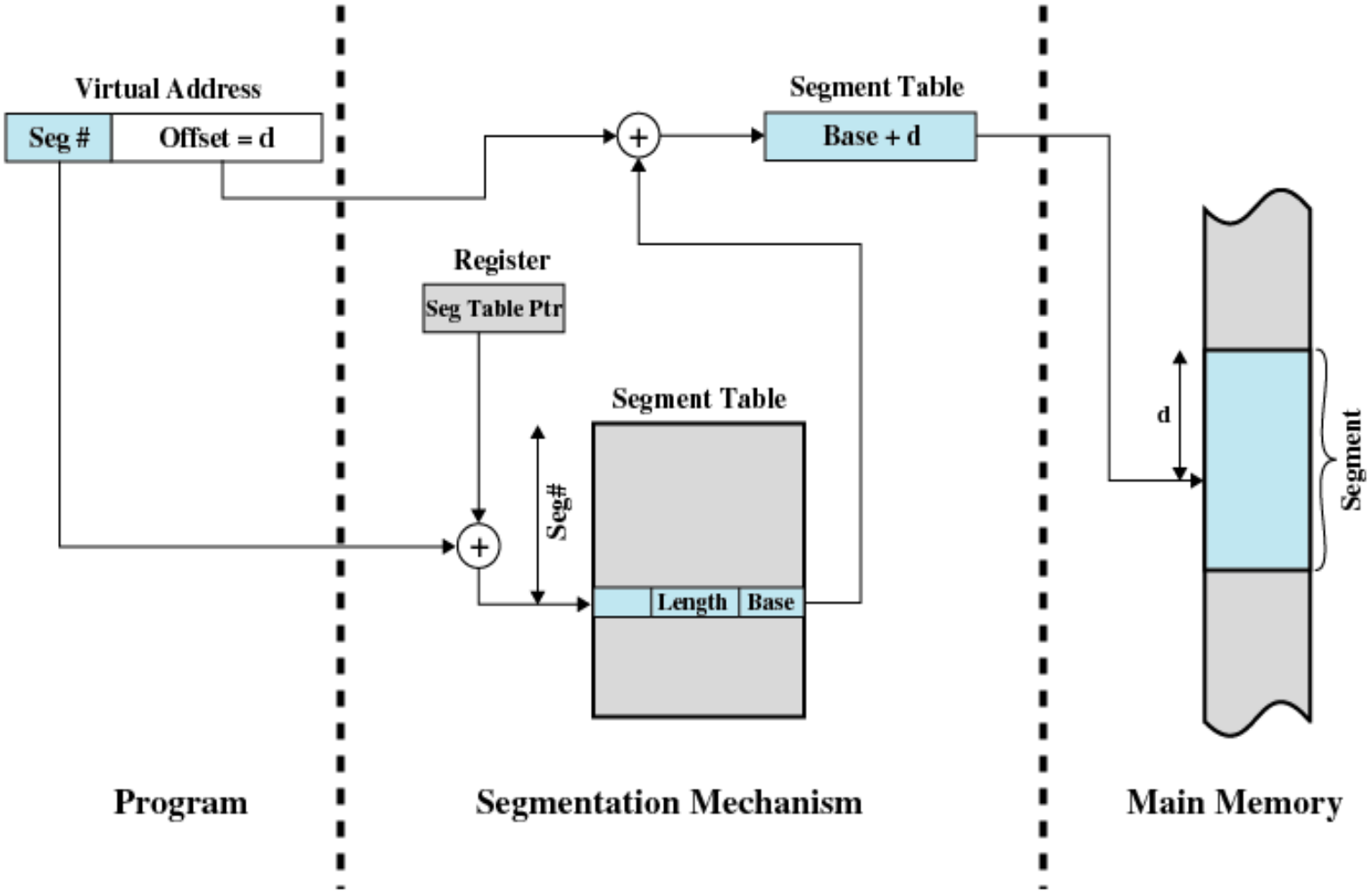


Figure 8.12 Address Translation in a Segmentation System

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