15CI06 – Operating Systems

Memory Management

Dr.M.Rajalakshmi
Associate Professor
Department of Information Technology
Coimbatore Institute of Technology
Coimbatore

Memory Management

- Introduction
- Memory Partitioning
- Basic requirements of Memory Management
- Basic blocks of memory management
  - Paging
  - Segmentation

Memory Management

- OS acting as a resource manager
- Mainly it is concerned with the management of main memory

Memory Management Functions

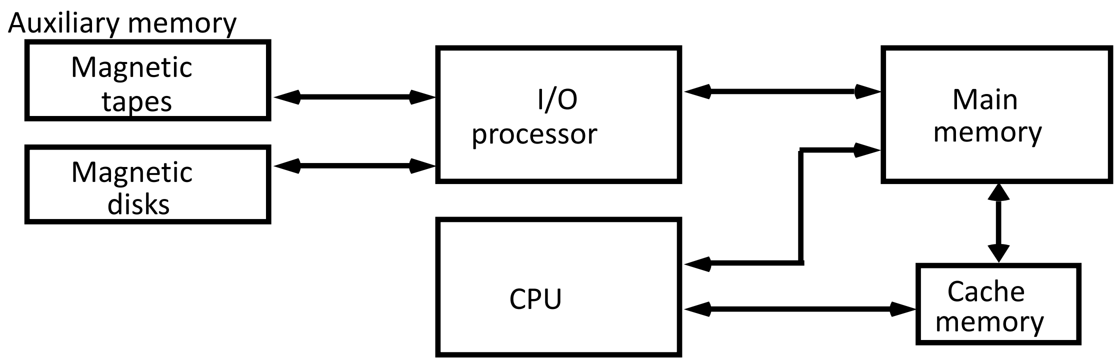
- Keep track of how each region of memory is being used. (allocated or free)
- Allocate and de-allocate memory regions in response to explicit and implicit requests.
- Manage movement of the contents of memory between primary and secondary storage

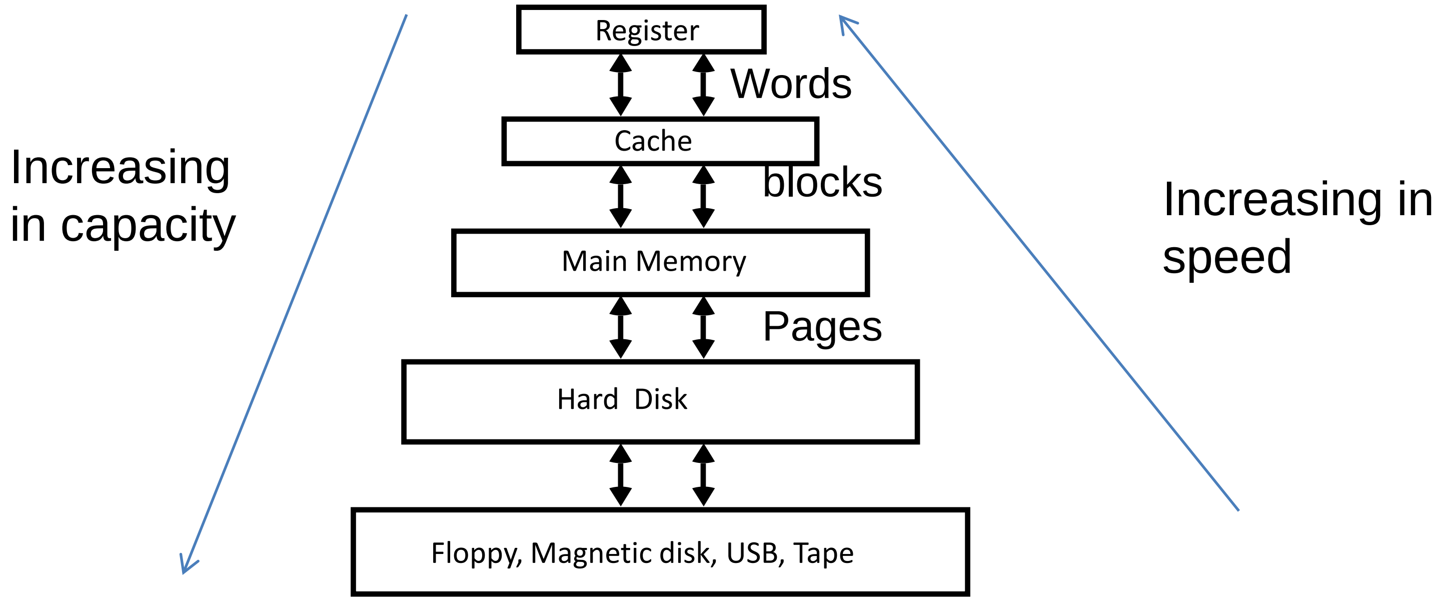
Key Characteristics of Computer Memory Systems

Capacity Number of words Number of bytes	**Unit of Transfer** Word Block
Access Method Sequential Direct Random Associative	**Performance** Access time Cycle time Transfer rate
Physical Type Semiconductor Magnetic Optical Magneto-optical	**Physical Characteristics** Volatile/nonvolatile Erasable/non-erasable

Memory Hierarchy

Memory Management must deal with storage hierarchy present in modern machines



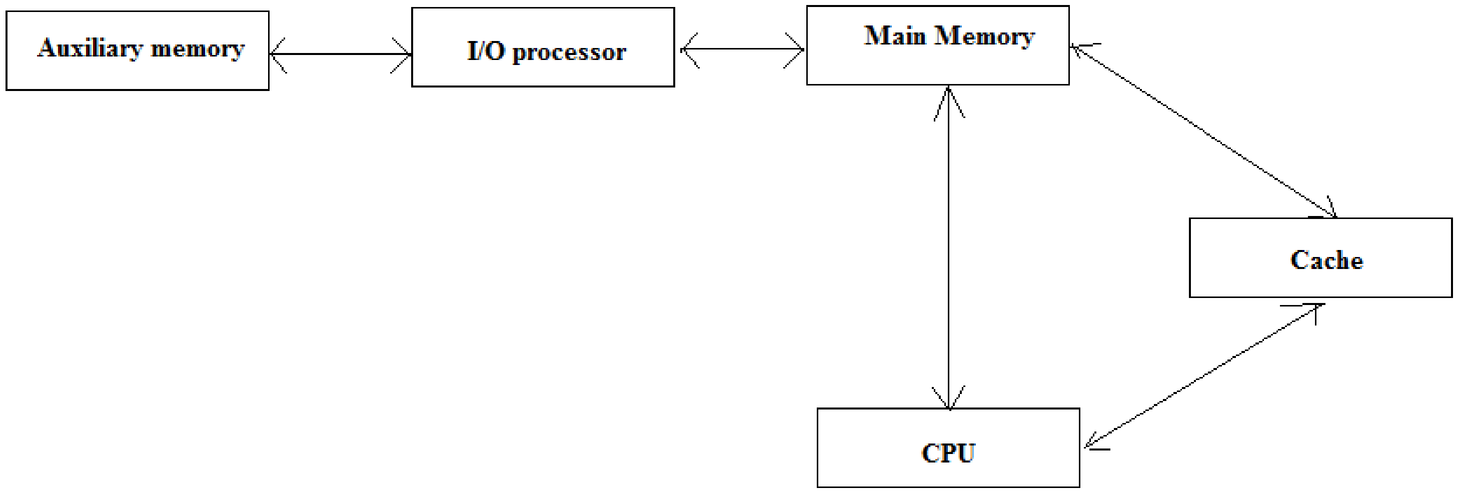


Memory Hierarchy

- Computer Memory Hierarchy is a pyramid structure that is commonly used to illustrate the significant differences among memory types.
- A hierarchical memory can be used to close the speed gap.
- The memory unit that directly communicate with CPU is called the *main memory*
- Devices that provide backup storage are called *auxiliary memory*
- The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory

Memory Hierarchy

- The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor
- A special very-high-speed memory called **cache** is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate



Memory Hierarchy

- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
- The typical access time ratio between cache and main memory is about 1 to 7
- Auxiliary memory access time is usually 1000 times that of main memory

Basic Concepts

- The maximum size of the memory that can be used in any computer is determined by the addressing scheme
- For example, a 16-bit computer that generates 16-bit addresses is capable of addressing up to 2^{16}= 64K memory locations. (1024 KB = 1 GB)
- Similarly, machines whose instructions generate 32-bit addresses can utilize a memory that contains up to 2^{32}=4G memory locations
- **Word**: The natural size with which a processor is handling data (the register size). The most common **word** sizes are 8, 16, 32 and 64 bits, but other sizes are possible. For examples, there were a few 36 bit machines, or even 12 bit machines. The **byte** is the smallest addressable unit for a CPU.
- Most modern computers are byte addressable
- Data transfer between the memory and processor takes place through the use of two processor registers, MAR and MDR

Basic Concepts

- A useful measure of the speed of memory units is the time that elapses between the initiation of an operation and the completion of that operation. This is referred to as the memory access time.

- Another important measure is the memory cycle time, which is the minimum time delay required between the initiation of two successive memory operations.

- A memory unit is called random-access memory(RAM) if any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location's address. The memory cycle time is the bottleneck in the system

Basic Concepts

- One way to reduce the memory access time is to use a cache memory. Cache memory is a small, fast memory that is inserted between the larger, smaller main memory and the processor.
- Virtual memory is used to increase the apparent size of the physical memory. Data are addressed in a virtual address space that can be as large as the addressing capability of the processor. But at any given time, only the active portion of this space is mapped onto locations in the physical memory. The remaining virtual addresses are mapped onto the bulk storage devices used, such as magnetic disks.

Memory Partitioning - Types

- Contiguous Memory Allocation Scheme (User programs are placed in main memory in continuous manner)
  - Single contiguous allocation scheme
  - Fixed Partitioning
  - Dynamic Partitioning

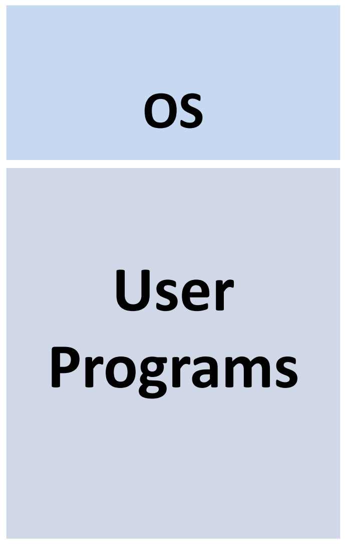
- Non-Contiguous Memory Allocation Scheme (A program is divided into several blocks and that may be placed throughout the main memory in pieces not necessarily adjacent to one another)
  - Simple Paging
  - Simple Segmentation
  - Virtual Memory Paging (Demand Paging)
  - Virtual Memory Segmentation (Demand Segmentation)

Single Contiguous Allocation scheme

- Simplest technique
- Only one process running at a time
- Incoming process is loaded in the available space

<u>Disadvantages</u>

- Part of memory space may be wasted
- Insufficient usage of memory
- Processor is idle during I/O operation

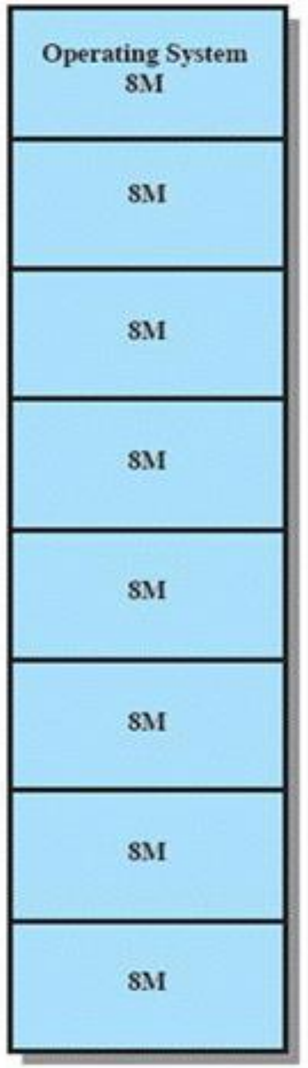


Fixed Partitioning

- Suitable for multiprogramming
- Can load more than one program
- Memory is divided into regions with fixed boundaries.
- We should know size and number of processes previously
- Used when designer knows about size and number of processes
- Partitions are done during setup.
- Two Types
    1. Equal-sized partitions
    2. Un-equal sized partitions

Fixed Partitioning – Equal sized

- Equal-size partitions (see fig a)
  - Any process whose size is less than or equal to the partition size can be loaded into an available partition
- The operating system can swap a process out of a partition
  - If none are in a ready or running state



(a) Equal-size partitions

Fixed Partitioning Unequal Sized

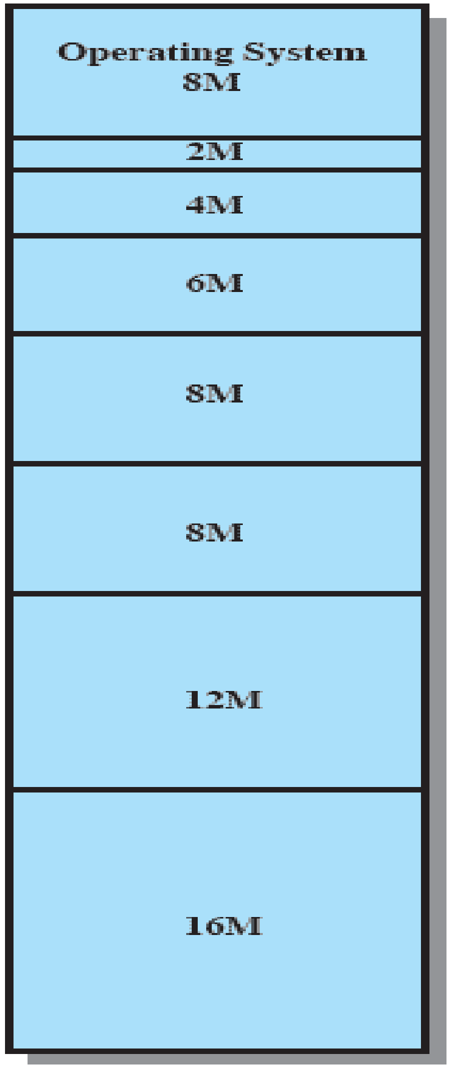
In Fig b

– Programs up to 16M can be accommodated

Smaller programs can be placed in smaller partitions, reducing internal fragmentation

Advantages of fixed partitioning

- Supports Multiprogramming
- Relatively simple



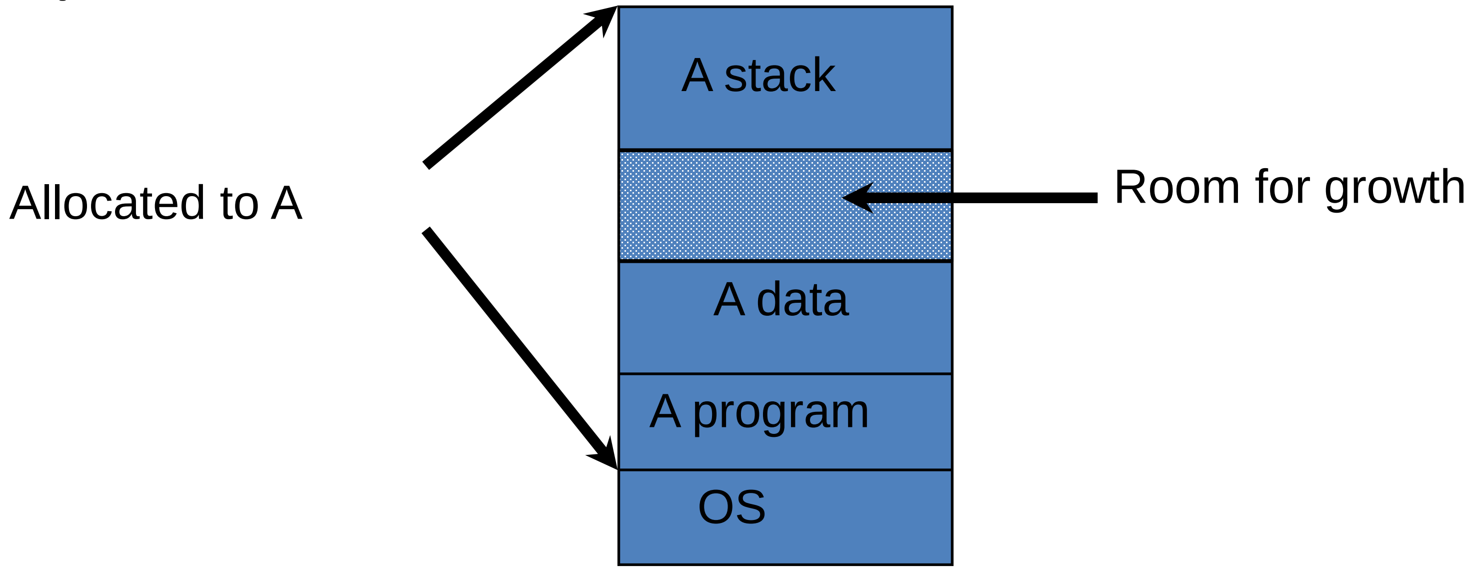
(b) Unequal-size partitions

Fixed Partitioning - Disadvantages

- A program may not fit in a partition
  - The programmer must split the program, so that only a portion of the program need to be in main memory
- Main memory utilization is extremely inefficient
  - Any program, no matter how small, occupies an entire partition results in ***internal fragmentation***
  - There is a wasted space internal to the partition due to the fact that the block of data loaded is smaller than the portion is called ***internal fragmentation***.

Internal Fragmentation

- Have some “empty” space for each processes



- Internal Fragmentation - allocated memory may be slightly larger than requested memory and not being used.

Fixed Partitioning

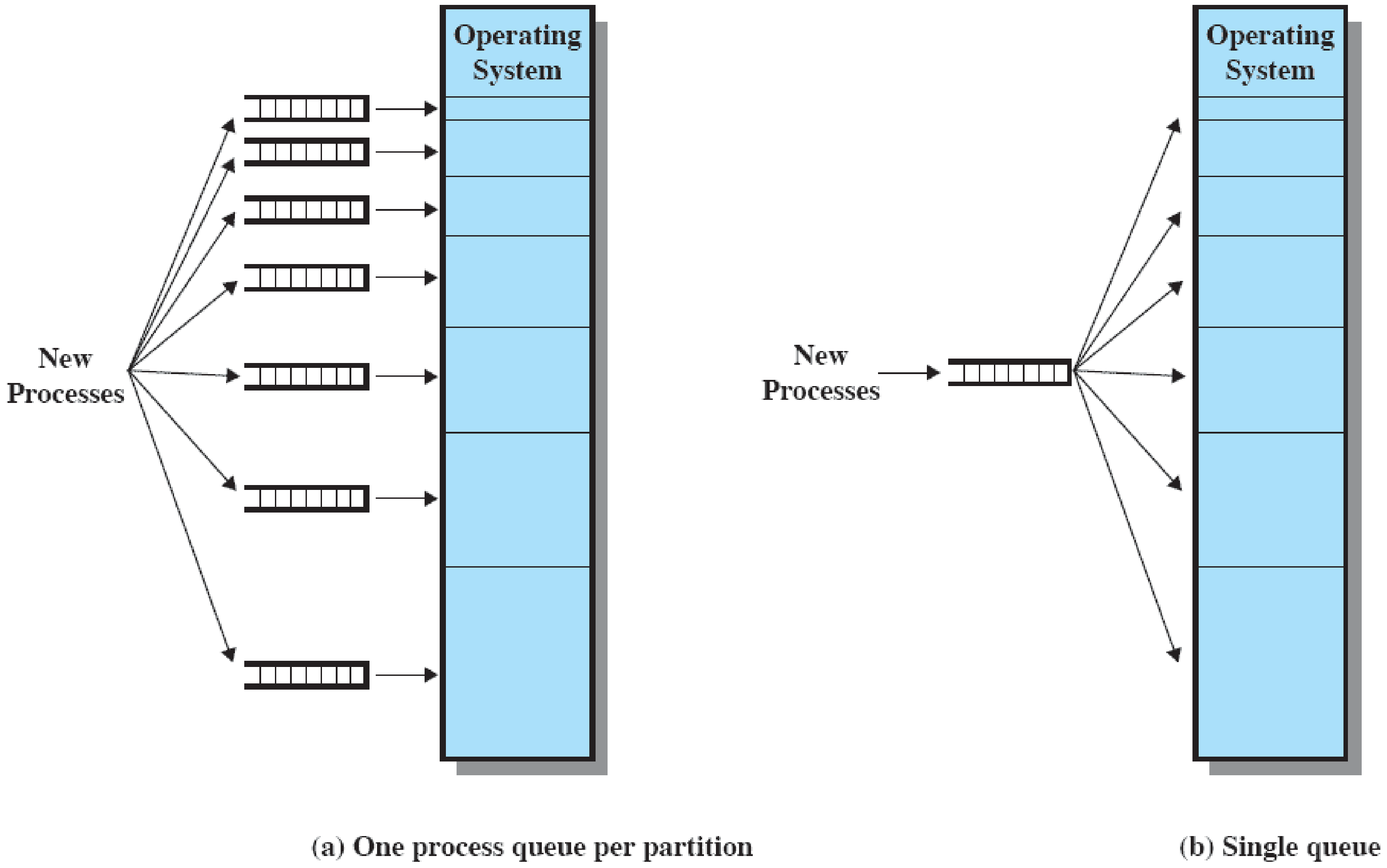
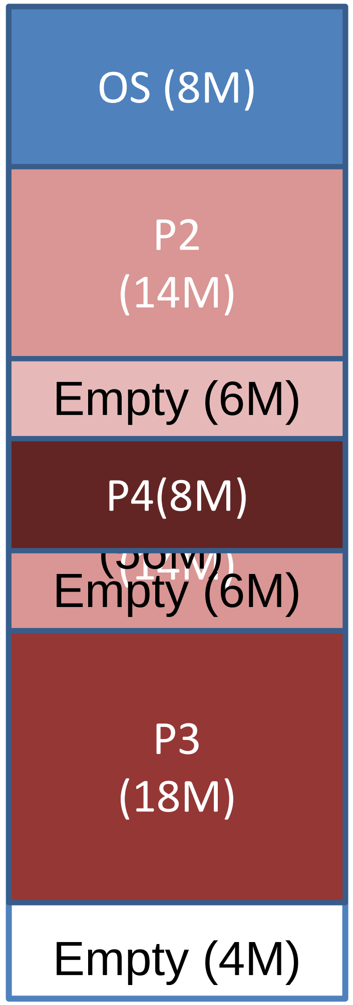


Figure 7.3 Memory Assignment for Fixed Partitioning

Dynamic Partitioning

- Partitions are of variable length and number
- Partition size is not fixed
- During the load time, i.e when the process is brought into the main memory ,Process is allocated exactly as much memory as required
- There will not be any wastage of memory
- All the processes are loaded in contiguous allocation

Dynamic Partitioning Example

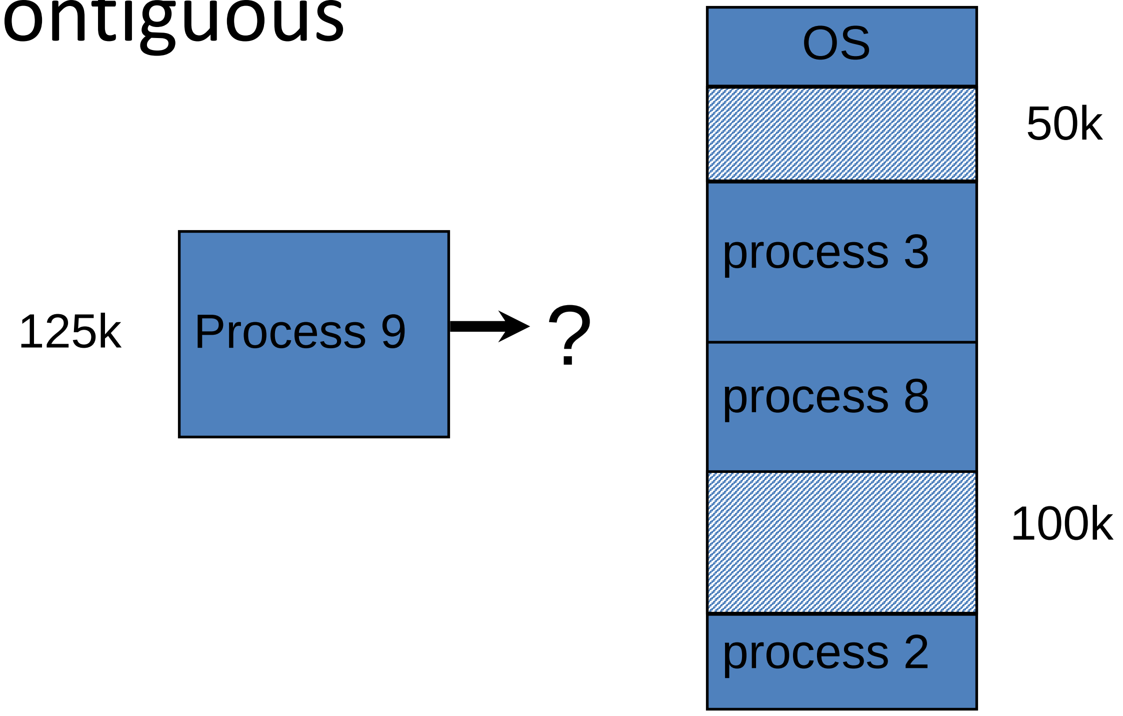


Refer to Figure 7.4

- ***External Fragmentation***
- Memory external to all processes is fragmented
- Can resolve using ***compaction***
- ***Compaction :*** Collecting the unused spaces by reshuffling the processes.
  - From time to time, the OS shifts the processes so that all of the free memory is together in one block (contiguous)
  - Time consuming and wastes CPU time

External Fragmentation

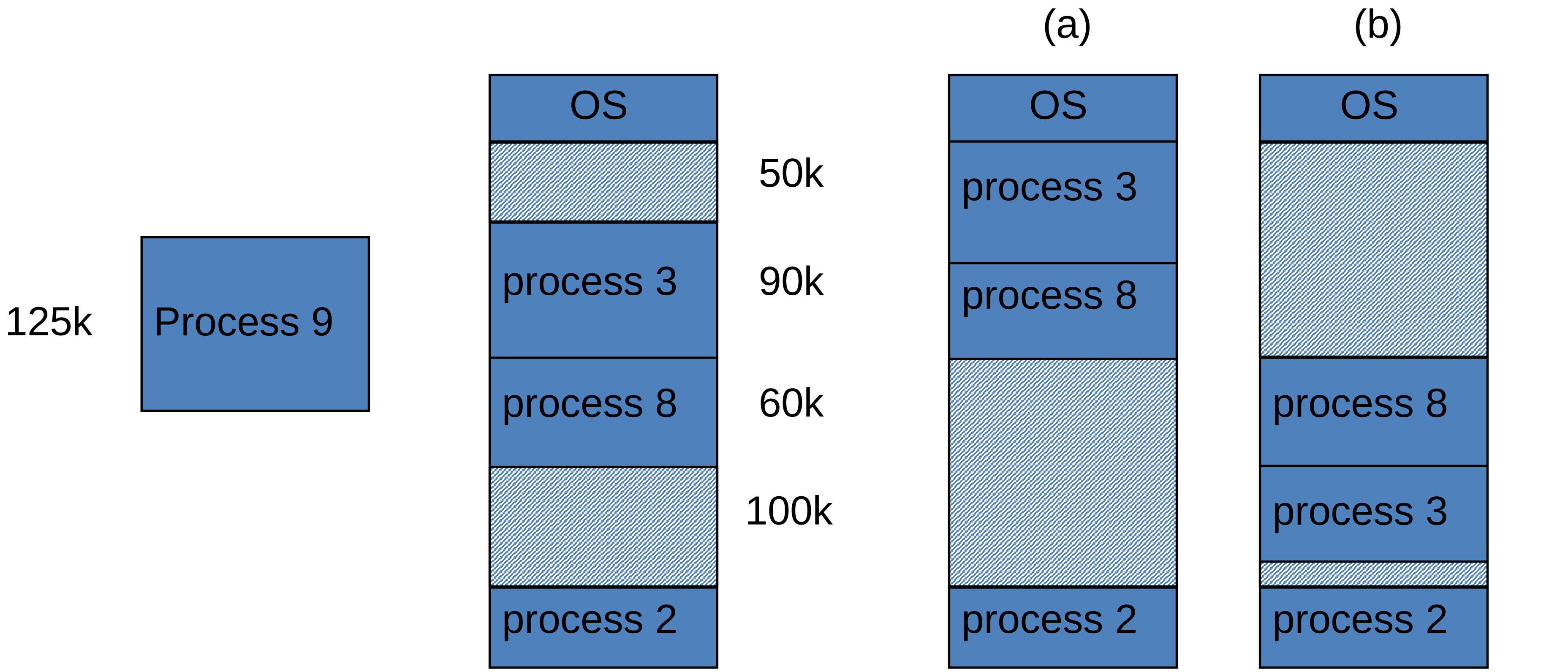
- External Fragmentation - total memory space exists to satisfy request but it is not contiguous



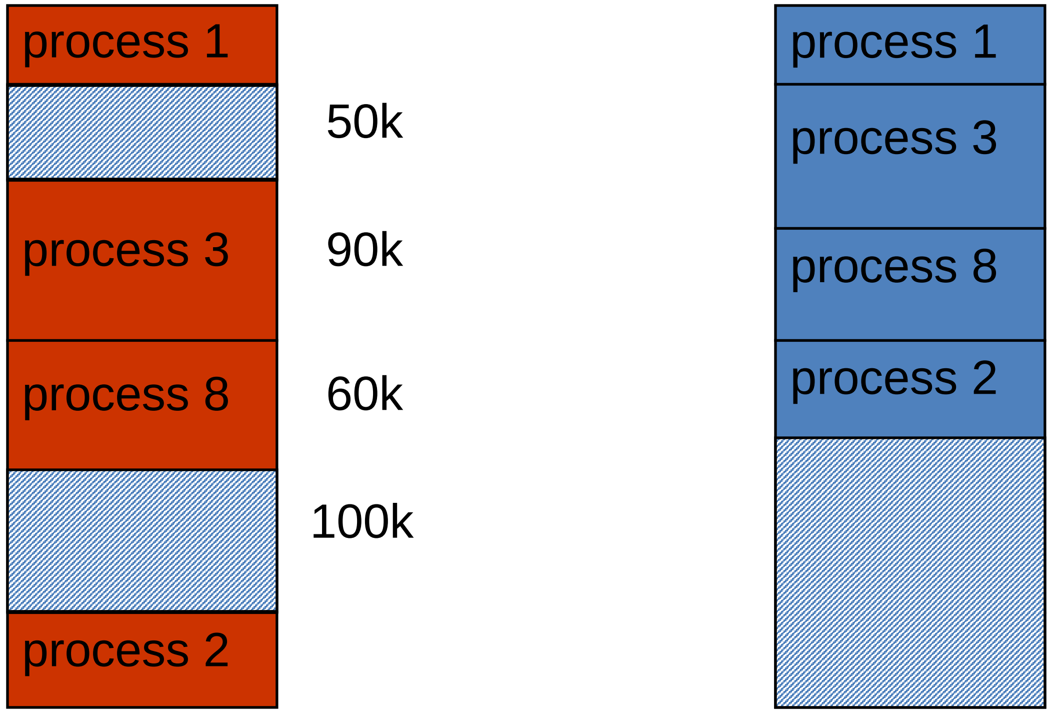
"But, how much does this matter?"

Compaction

- Shuffle memory contents to place all free memory together in one large block
- Only if relocation dynamic!
- Same I/O DMA problem



Cost of Compaction



- 2 GB RAM, 10 nsec/access (cycle time)

  ➔ 5 seconds to compact!
- Disk much slower!

Buddy System

- Entire space available is treated as a single block of 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
  - Process continues until smallest block greater than or equal to *s* is generated

Example of Buddy System

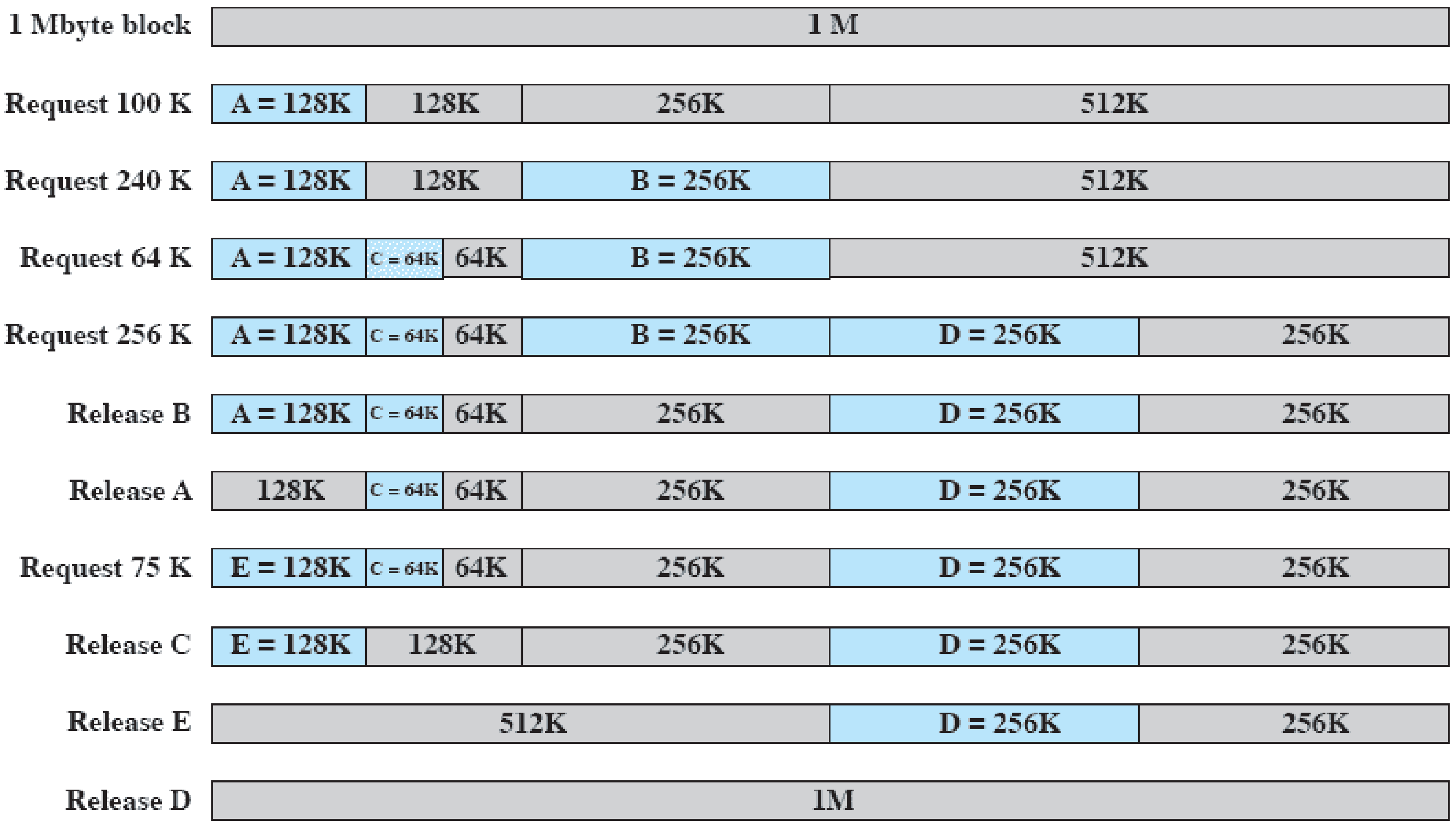


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

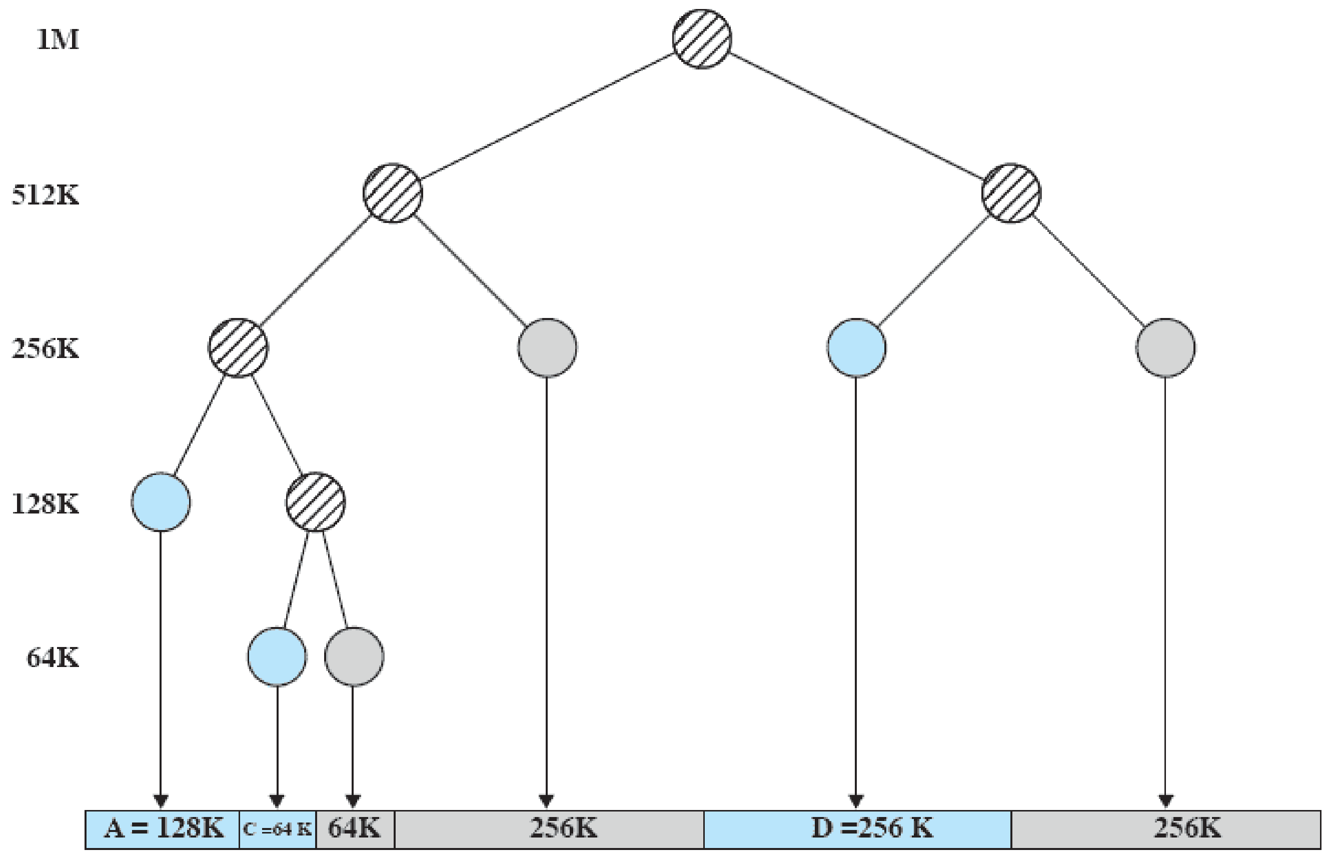


Figure 7.7 Tree Representation of Buddy System

Memory Management Requirements

- Relocation
- Protection
- Sharing
- Logical organisation
- Physical organisation

Relocation

- When a program is loaded into memory the actual (absolute) memory locations are determined
- A process may occupy different partitions which means different absolute memory locations during execution.
  - Swapping
  - Compaction
- The programmer does not know where the program will be placed in memory when it is executed,
  - it may be swapped to disk and return to main memory at a different location (relocated)
- Memory references must be translated to the actual physical memory address

Addressing

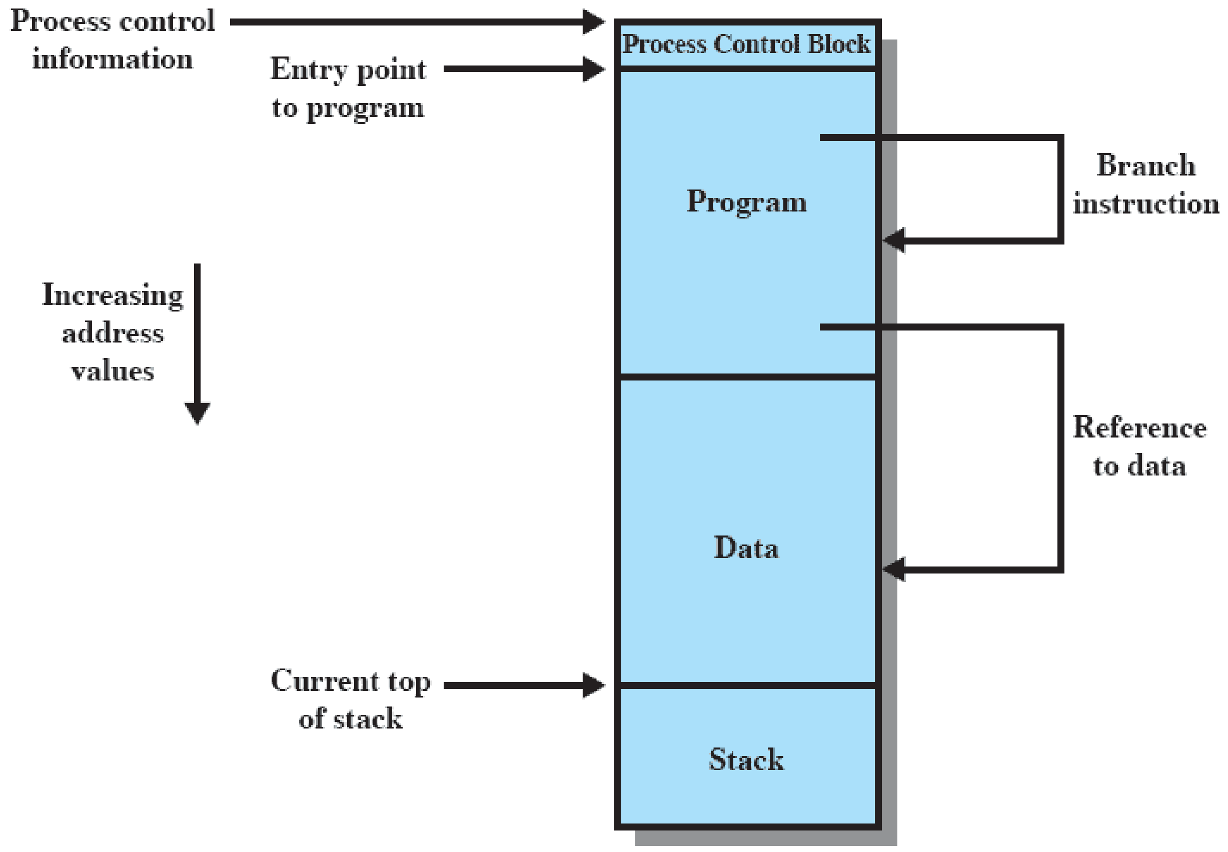


Figure 7.1 Addressing Requirements for a Process

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

Sharing

- Allow several processes to access the same portion of memory
- Better to allow each process access to the same copy of the program rather than have their own separate copy

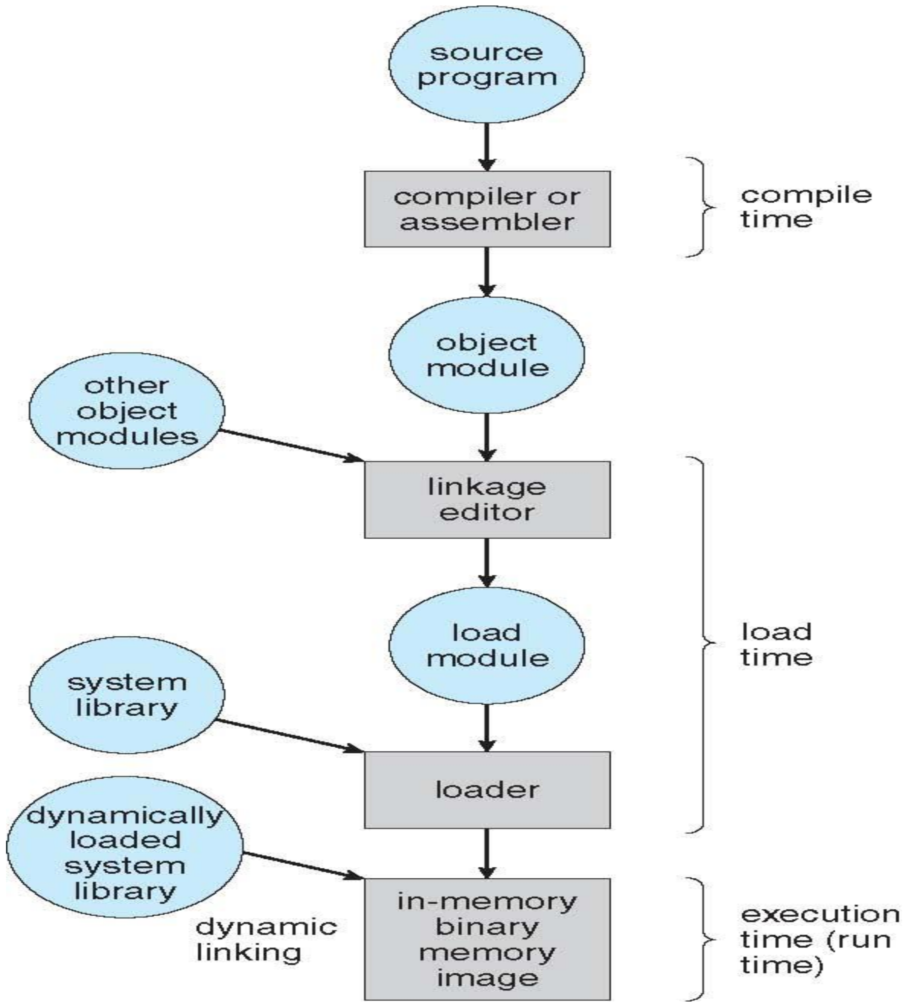
Logical Organization

- Memory is organized linearly (usually)
- 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
- Segmentation helps here

Physical Organization

- Cannot leave the programmer with the responsibility to manage memory
- Memory available for a program plus its data may be insufficient
  - Overlaying allows various modules to be assigned the same region of memory but is time consuming to program
- Programmer does not know how much space will be available

Multistep Processing of a User Program



Addresses

- Logical Address
  - Reference to a memory location independent of the current assignment of data to memory.
  - generated by the CPU; also referred to as **virtual address**
- Relative Address (Loader)
  - Address expressed as a location relative to some known point.
- Physical or Absolute Address
  - The absolute address or actual location in main memory.
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Address Binding

- Where a symbolic label/name is translated (bound) to an actual address

- The actual binding can be specified in the program, or resolved at compile time, link time, load time, or run time.

Types of loading

- Three ways a program can get loaded
  - Absolute loading – Load program at the same address (virtual and/or physical) every time
  - Relocatable loading – Load program at different addresses based on what is available
  - Dynamic run-time loading – Load and reload the program at different addresses while the program is running. In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format.

Types of Linking

- Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed.
- **Static Linking :** Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking. Libraries linked at compile time, so program code size becomes bigger
- **Dynamic Linking** : Linking is done at the time of execution, it is called as dynamic linking. Program code size remains smaller.

Types of Dynamic Linking

- Load time Dynamic Linking

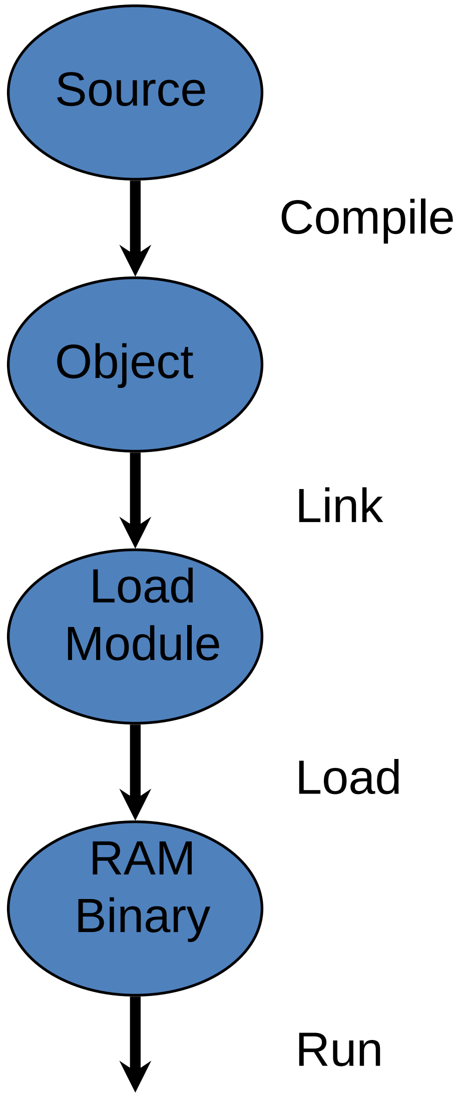
  The load module is loaded into main memory. Any reference to an external module causes the loader to find the module, load it and alter the addresses from the beginning of the application module

- Runtime time Dynamic Linking

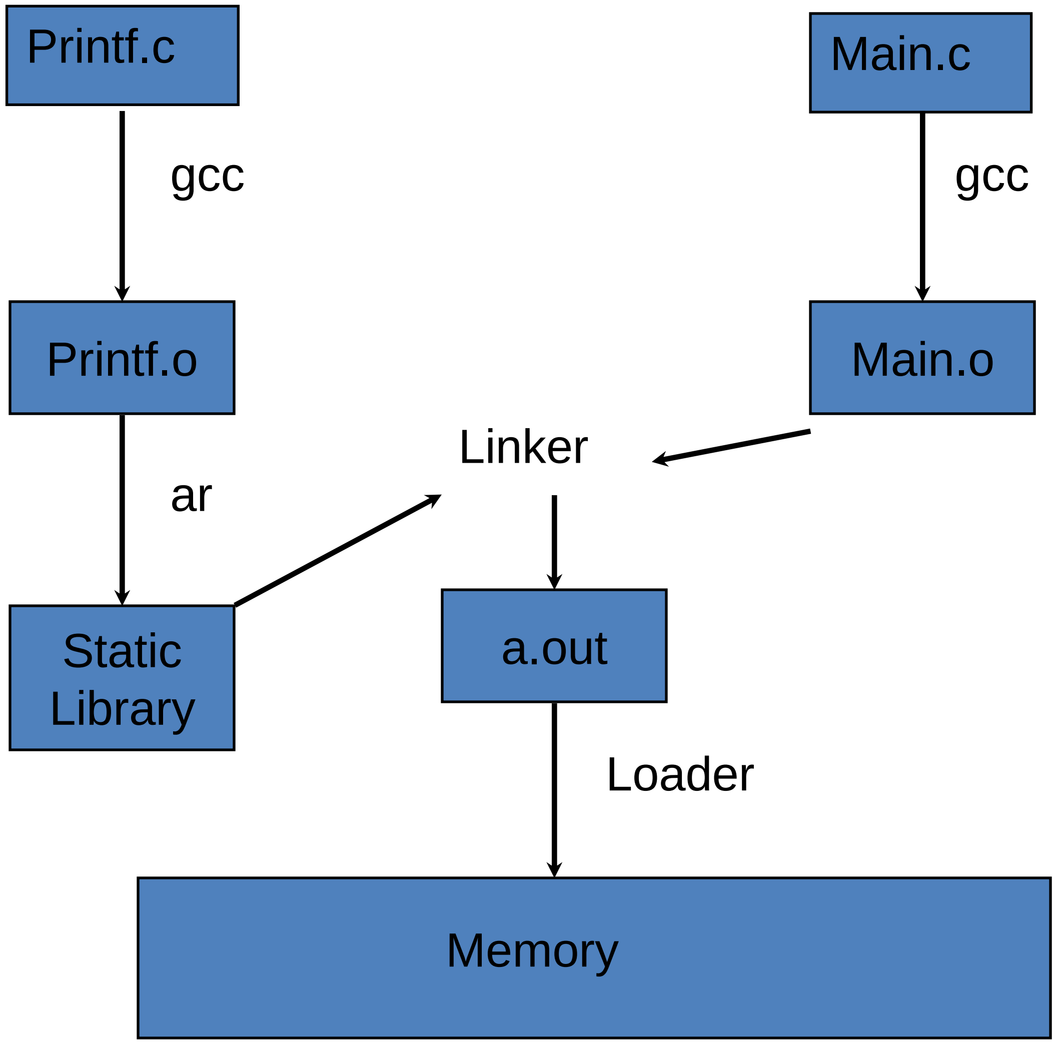
  Some of the linking is postponed to until execution time. When a call is made to the absent module, OS locates the module, loads it and links it to the calling module

Address Binding

- Compile Time
  - maybe absolute binding (`.com`)
- Link Time
  - dynamic or static libraries
- Load Time
  - relocatable code
- Run Time
  - relocatable memory segments
  - overlays
  - paging

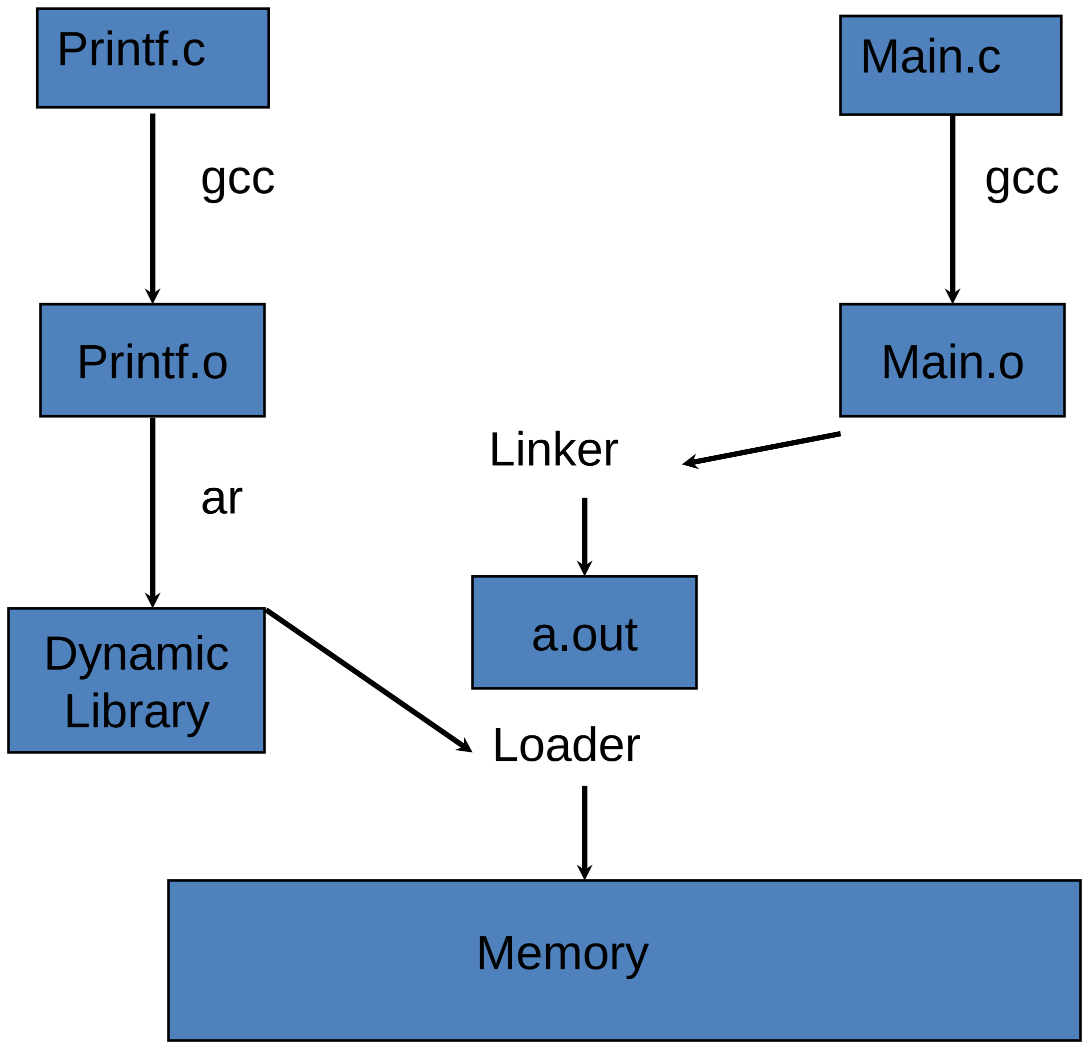


Normal Linking and Loading



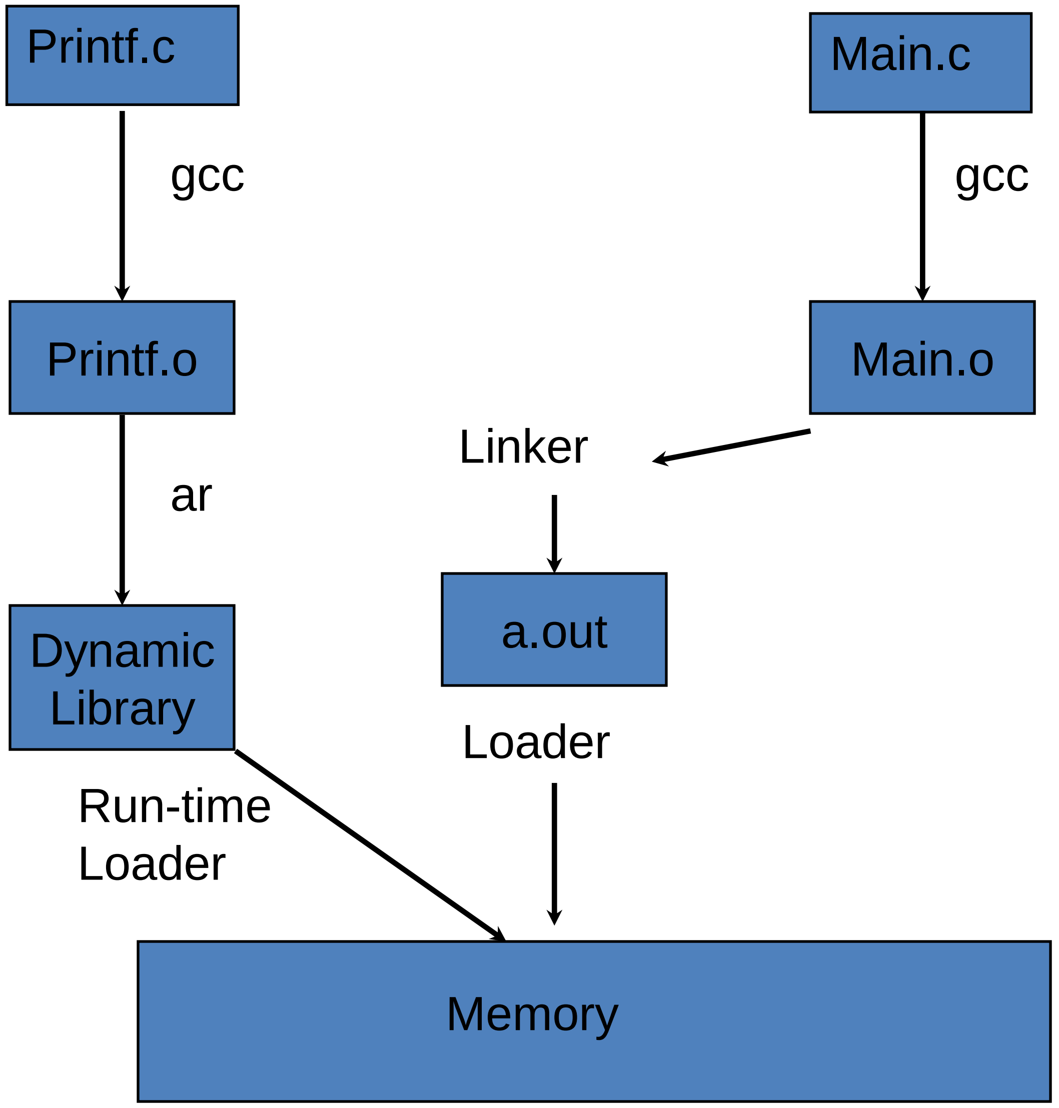
X Window code:
- 500K minimum
- 450K libraries

Load Time Dynamic Linking



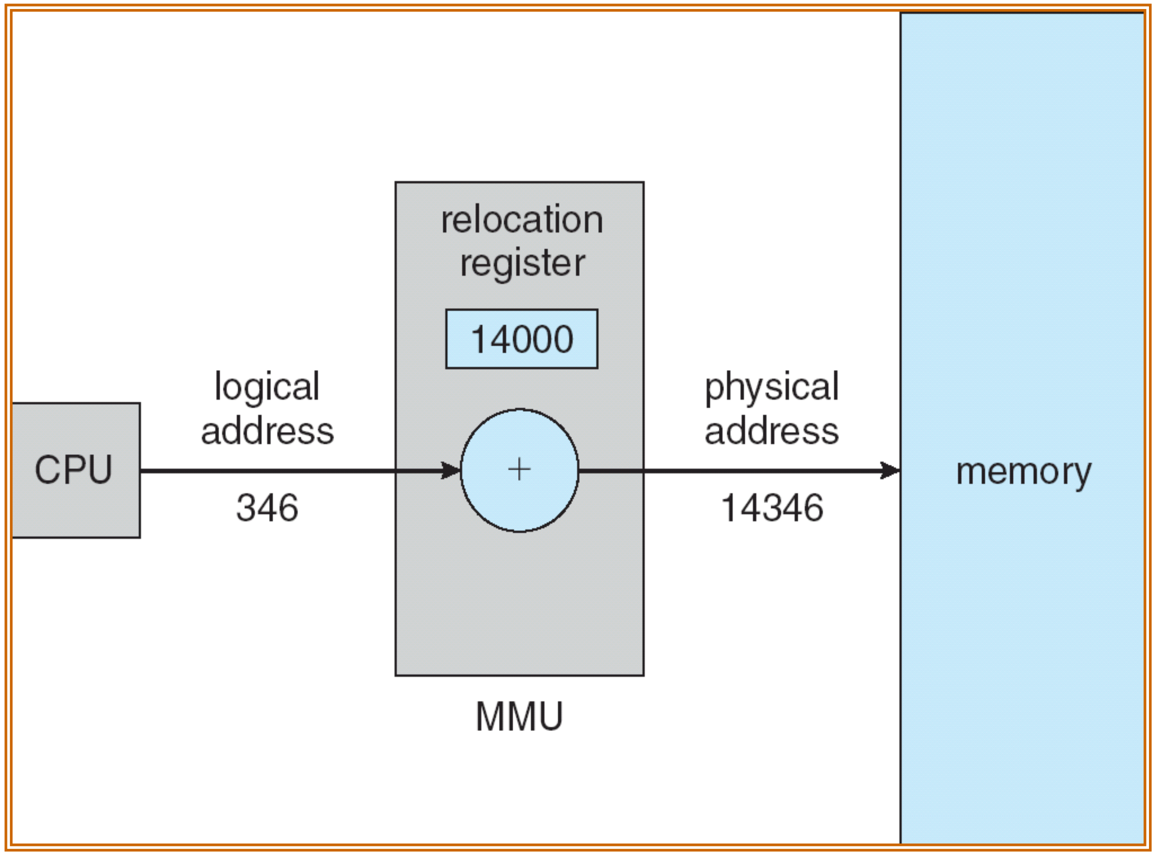
- Save disk space.
- Libraries move?
- Moving code?
- Library versions?
- Load time still the same.

Run-Time Dynamic Linking



Save disk space.
Startup fast.
Might not need all.

Dynamic relocation using a relocation register



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

Registers Used during Execution

- 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

Relocation

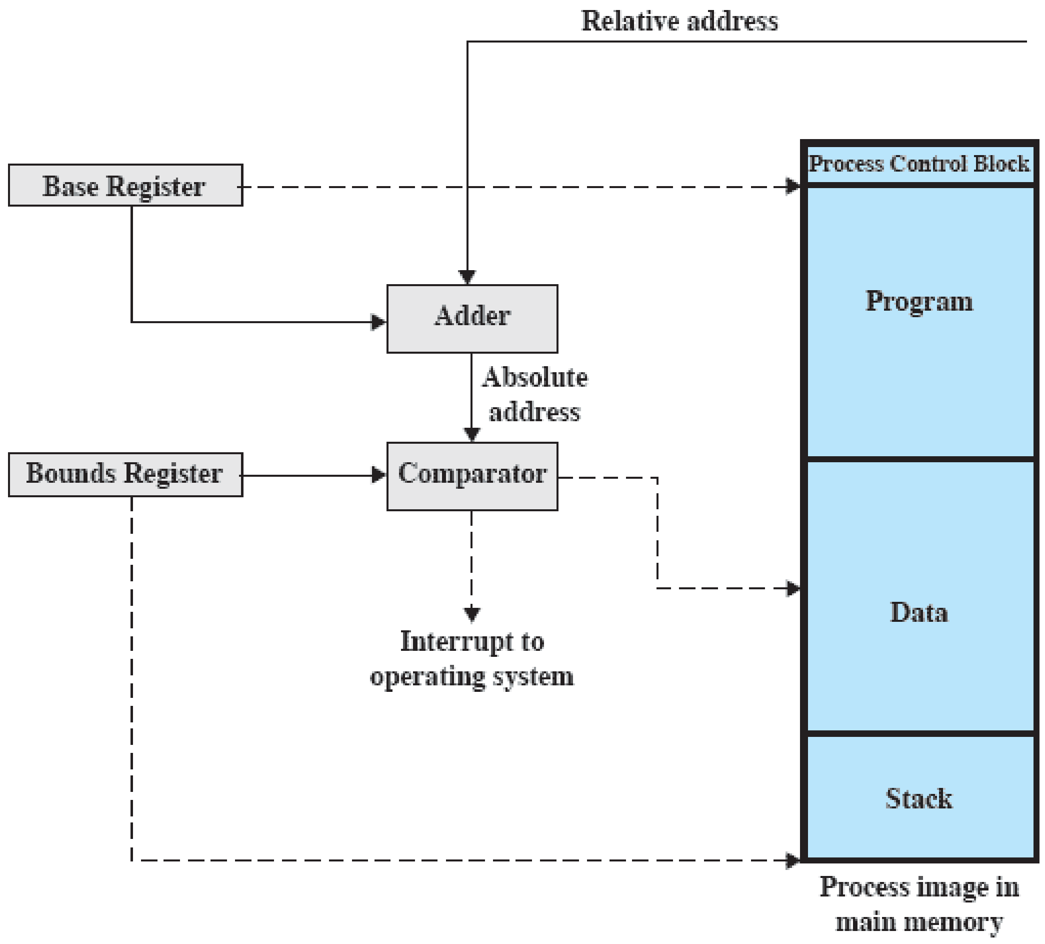


Figure 7.8 Hardware Support for Relocation

Non-Contiguous Memory Allocation Scheme

Paging

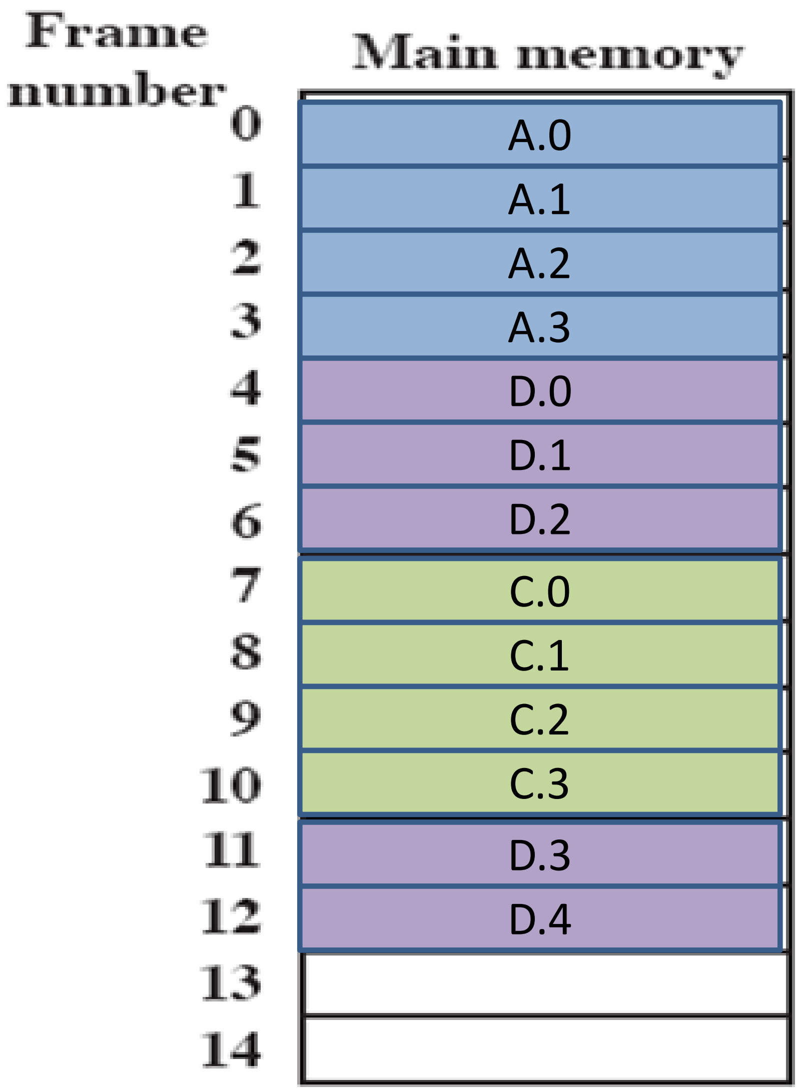
- Paging is similar to fixed partitioning but here a process may occupy more than one partition and these partitions need not be contiguous.
- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called ***pages***
- The chunks 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

Memory Management Terms

Table 7.1 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.

Processes and Frames



Page Table

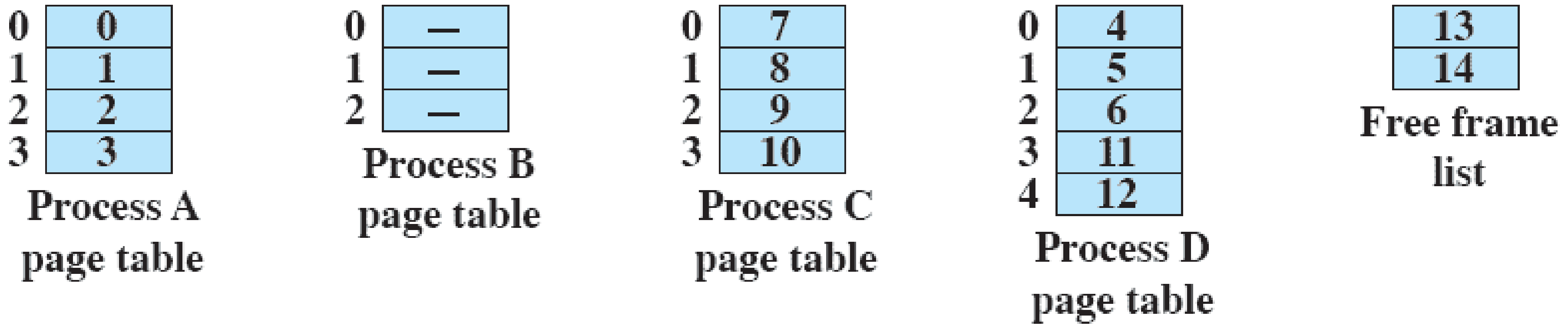


Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Paging Hardware

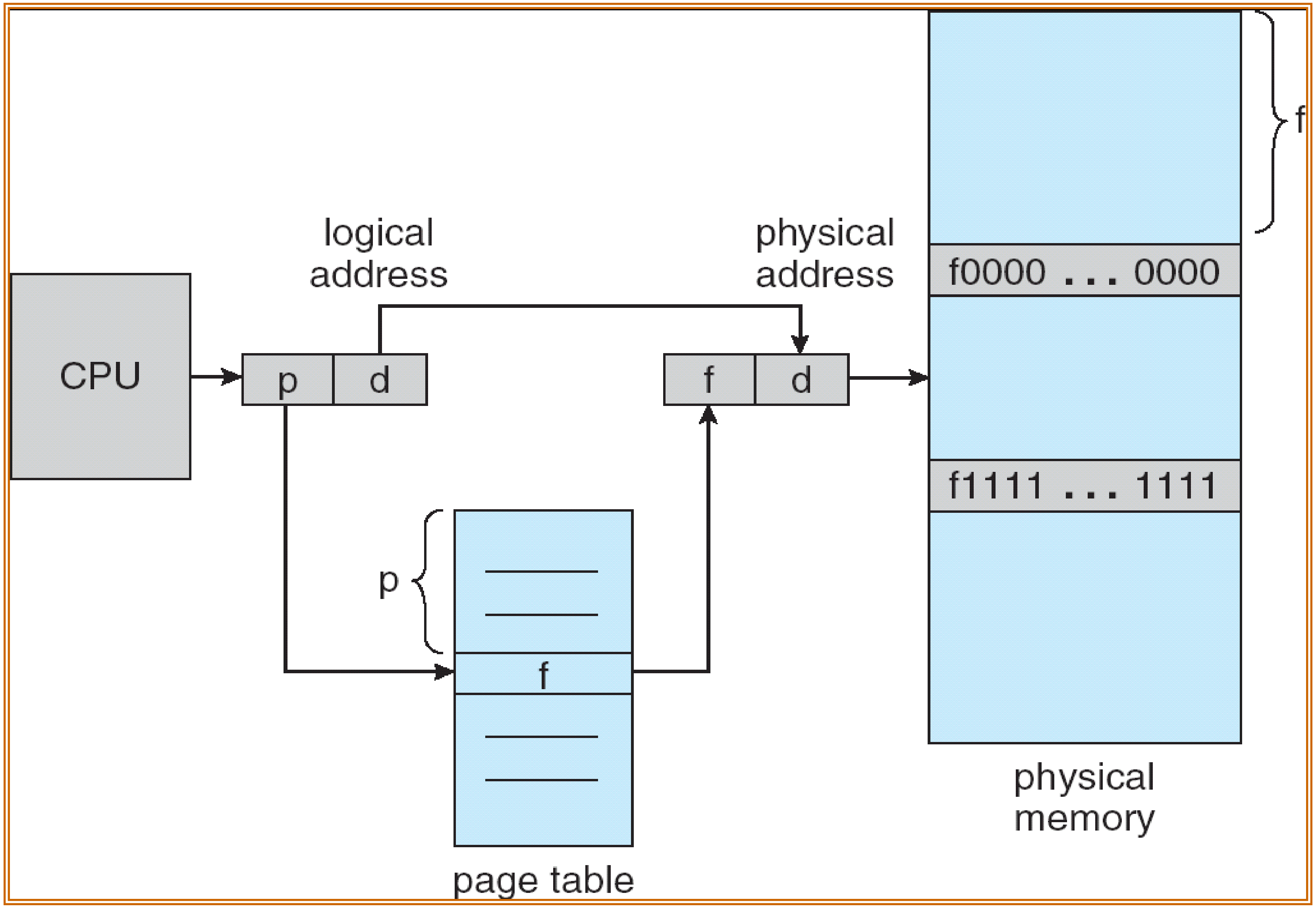
To make the paging scheme convenient let us take page size and frame size as power of 2.

Each address generated by the CPU is divided into 2 parts.

1. Page number
2. page offset (displacement)

Displacement locate the object from the starting address of memory

PAGING - HARDWARE



Logical Addresses

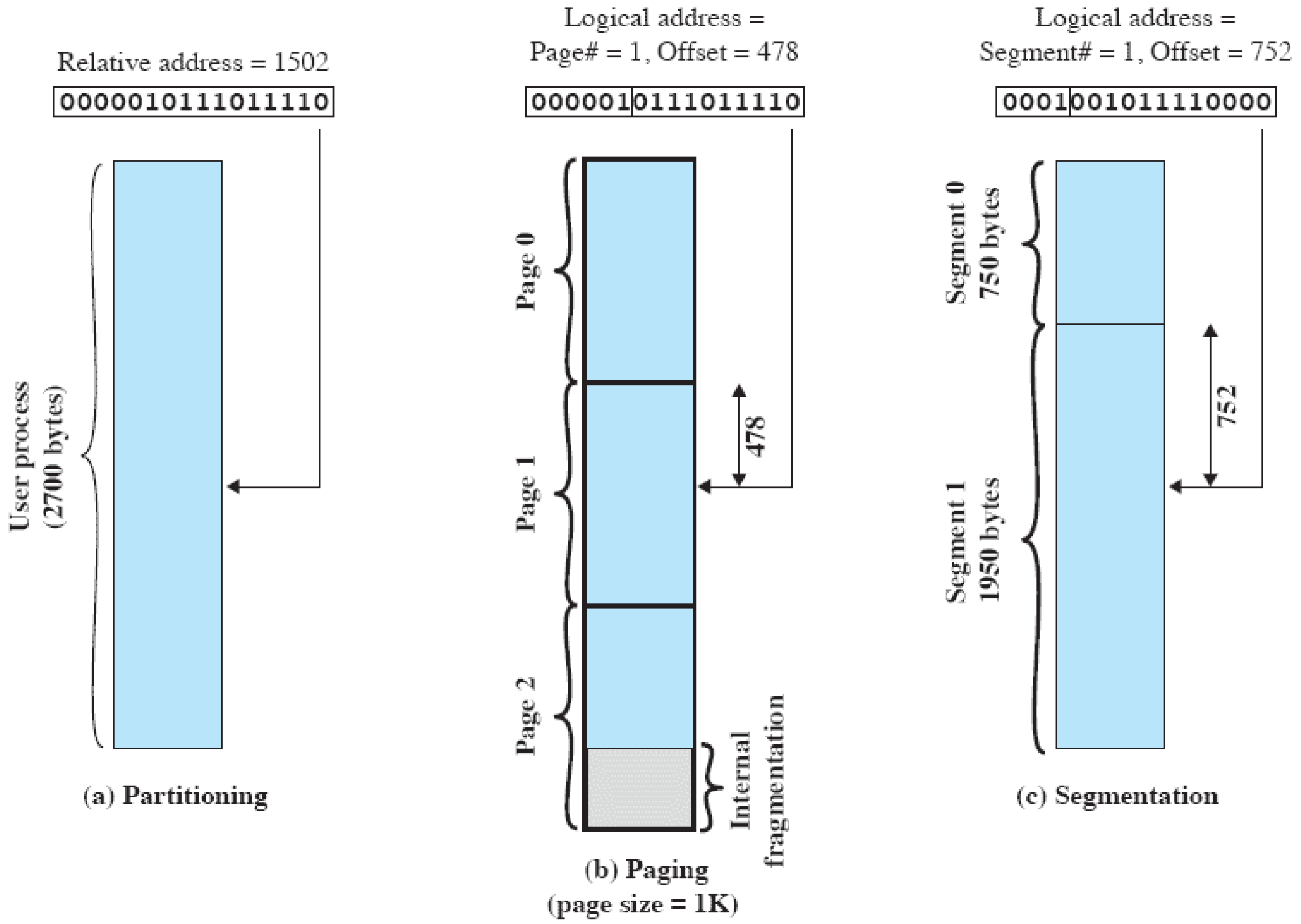


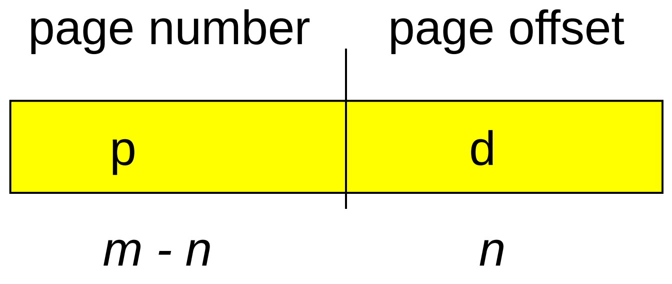
Figure 7.11 Logical Addresses

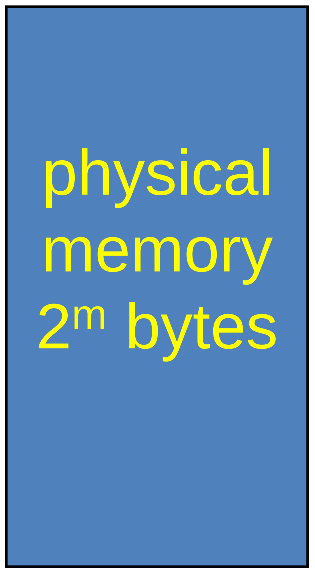
Address Translation Scheme

Page number (*p*) – used as an index into a *page table* . Page frame address is extracted from the page table. (base address of each page in physical memory)

Page offset (d) – gives the physical address within the frame. (combined with base address to define the physical memory address that is sent to the memory unit)

- address space 2^m
- page offset 2^n
- page number 2^{m-n}





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

Address Translation Scheme

- Consider an address of n + m bits, where the leftmost n bits are the page number and the rightmost m bits are the offset.

The following steps are needed for address translation:

1. Extract the page number as the leftmost n bits of the logical address.
2. Use the page number as an index into the process page table to find the frame number.
3. The starting physical address of the frame is k x 2^m and the physical address of the referred byte is that number plus the offset. (Eg. Starting address of the 6th frame is 6 x 2^{10} = 6144)

This physical address need not be calculated. It is easily constructed by appending the frame number to the offset.

Starting address of 0th frame : 0 to 1023

Starting address of 1st frame : 1024 to 2047

Etc.

Address Translation Scheme

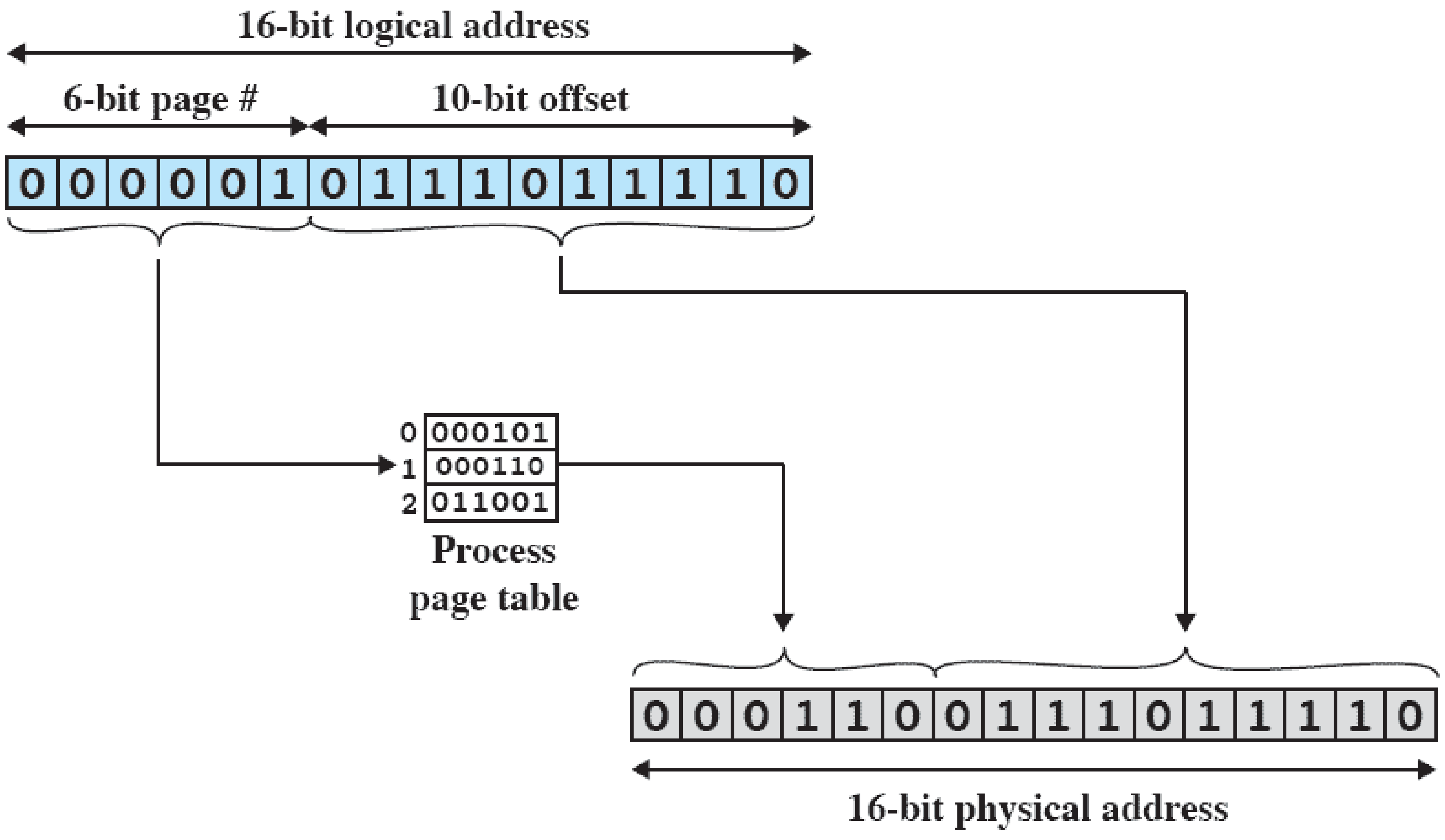
- In this example, 16-bit addresses are used, and the page size is 1K =1024 bytes.
- The relative address 1502, in binary form, is 0000010111011110.
- Page size is 1 KB i.e . 2^{10} bytes. So 10 bits are needed to address the page.

- 6 bits are for page number .
- Thus a program consists of 2^6 pages i.e 64 pages of 1 KB each.

  So the logical address 000001 0111011110 corresponds to

  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 = 0001100111011110

Paging



(a) Paging

Segmentation

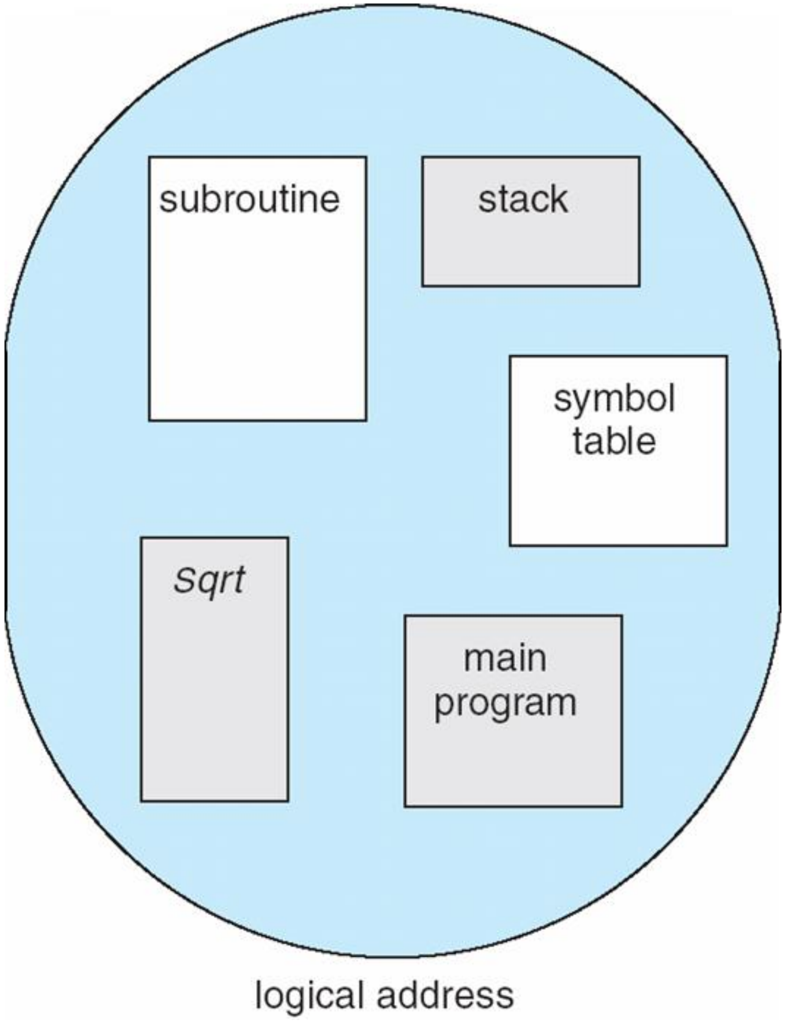
- Memory-management scheme that supports user view of memory
- A program is a collection of segments
  - A segment is a logical unit such as:

main program, procedure, function,
method, object, local variables,
global variables, common block
stack, symbol table, arrays etc.

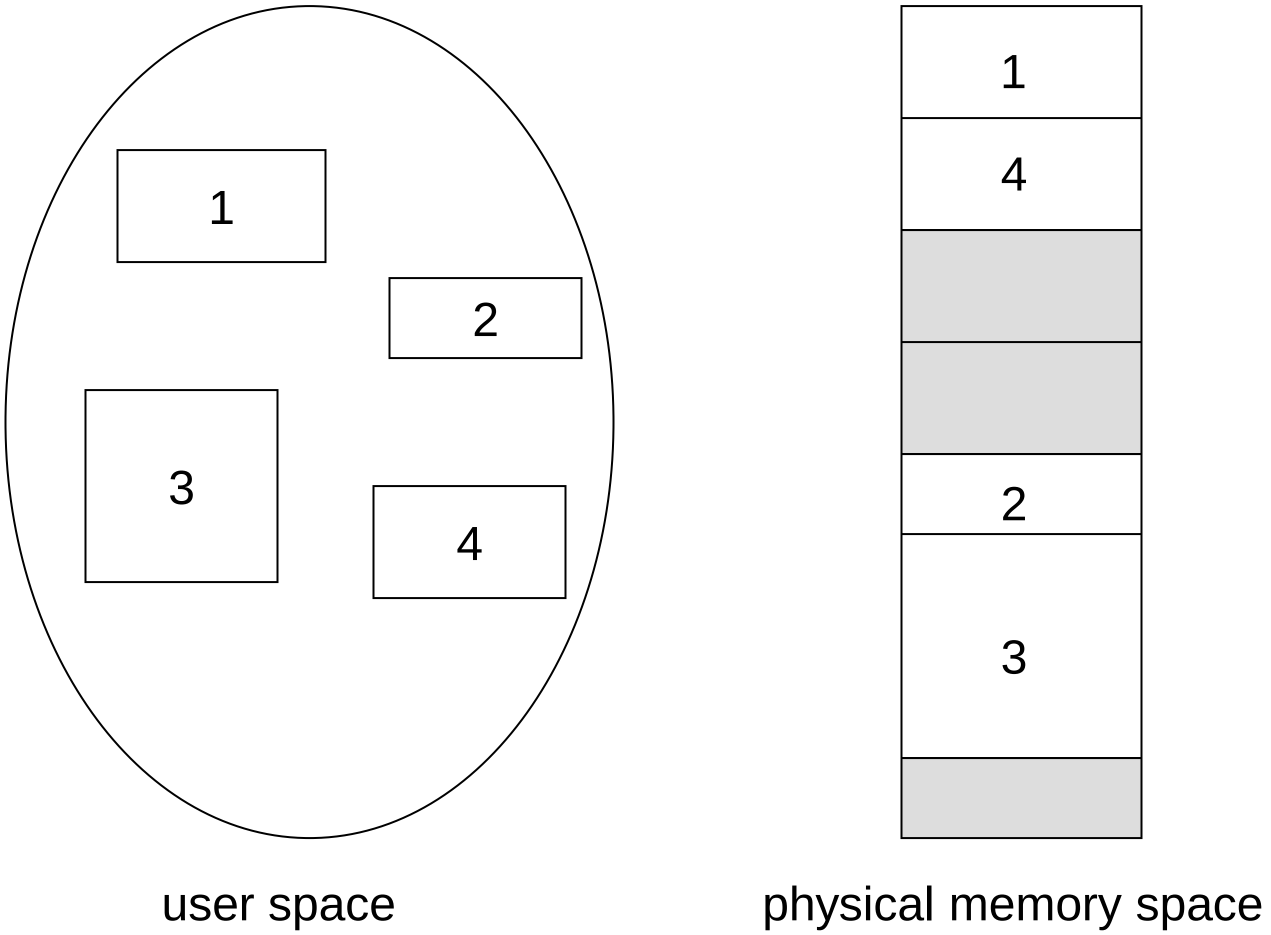
Segmentation

- A process can be subdivided into segments
  - Segments may vary in length
  - There is a maximum segment length
- Segmentation is similar to dynamic partitioning, but a program may occupy more than one partition and those partition need not be contiguous
- Provides more utilization of memory
- Paging is invisible to the programmer but segmentation is usually visible and is provided as a convenience for organizing programs and data.

User's View of a Program



Logical View of Segmentation



Segmentation Architecture

- Logical address space is collection of segments. Each segment has a name (number) and length.
- Logical address consists of a two tuple:

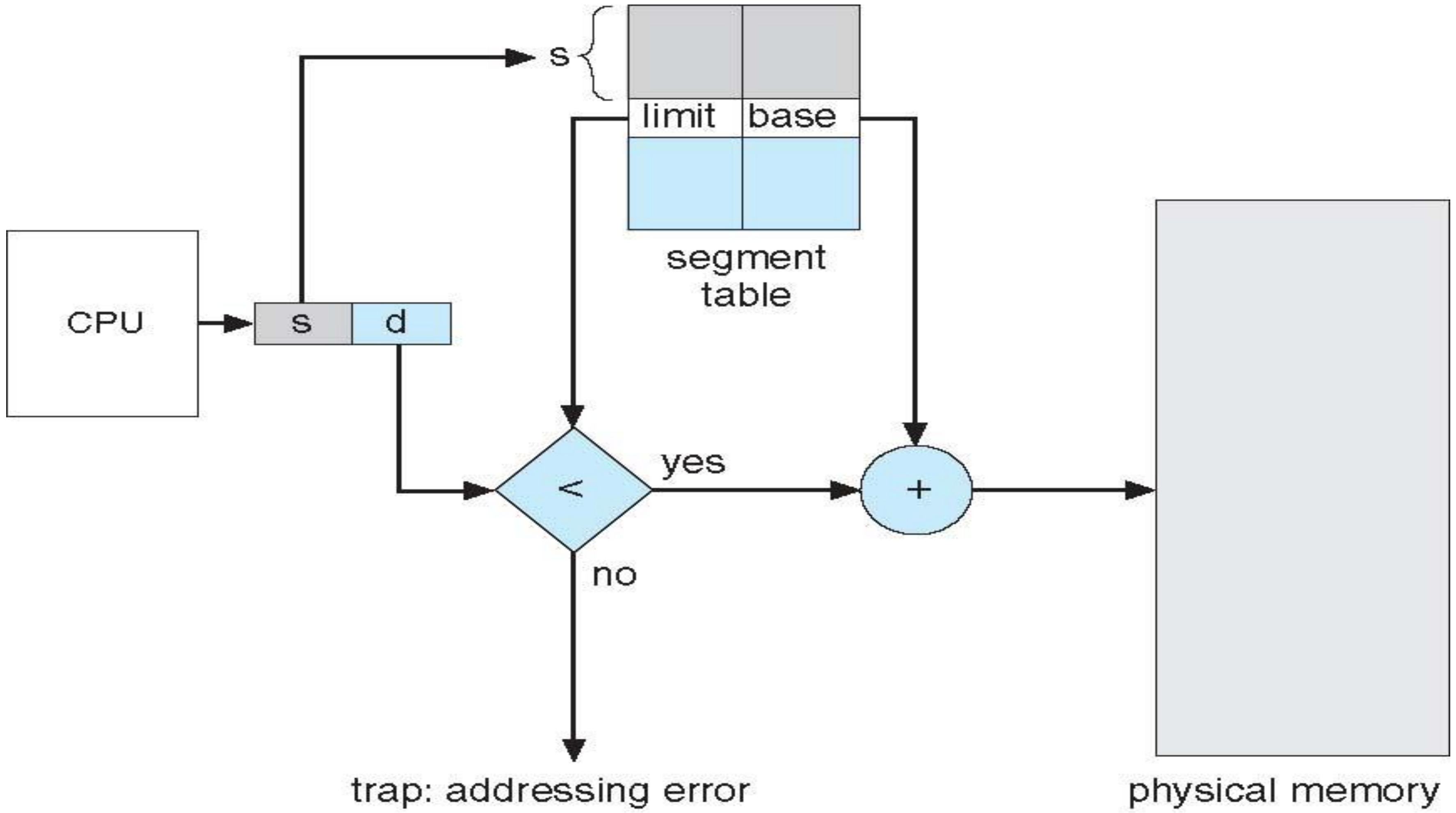
  <segment-number, offset>

  Seg-Number : used as an index into the segment table.

  Offset : must be between 0 and segment length.

  Segment table for each process and a list of free blocks of main memory
- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

Segmentation Hardware



Segmentation (Cont.)

- Address translation

  i. Extract the segment number as the leftmost ‘n’ bits of logical address.

  ii. Use the segment number as an index into the process segment table to find the starting physical address of the segment

  iii. Compare offset expressed in the rightmost m bits to the length of the segment. If the offset is greater than the length, the address is invalid.

  iv. The desired physical address is the sum of the starting physical address of the segment and the offset.

- 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 our example, we have the logical address 0001001011110000, which is segment number 1, offset 752.

- In the example on the slide
  - n = 4 bits
  - m =12 bits
- Thus the maximum segment size is 2^{12} = 4096.

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

  Then the physical address is 0010000000100000 + 001011110000 = 0010001100010000

Segmentation

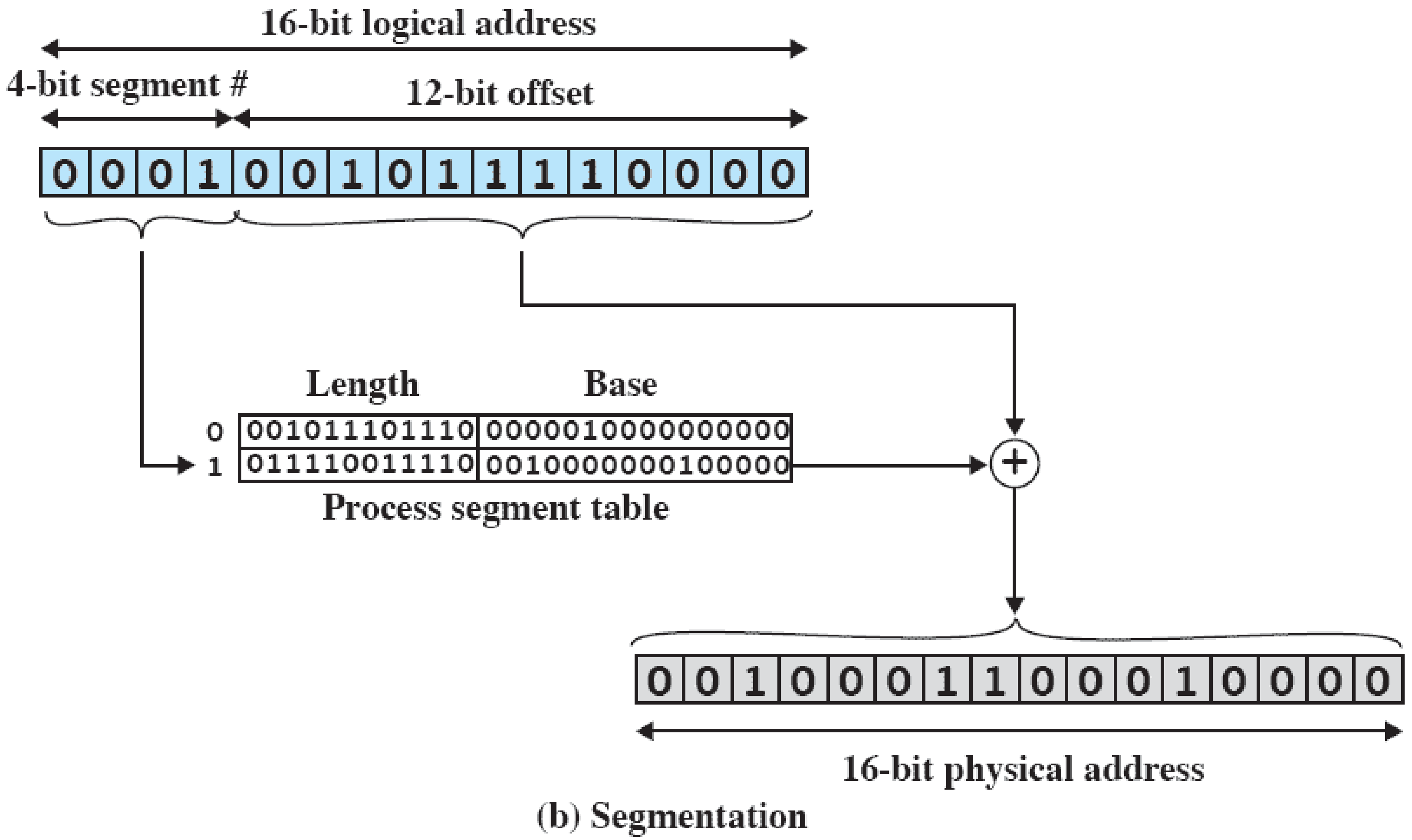


Figure 7.12 Examples of Logical-to-Physical Address Translation

Segmentation

- Advantages
  - Segments are easily relocatable. So it provides efficient utilization of memory
  - Dynamic relocation
  - Sharing of segments are easier
- Disadvantages
  - External fragmentation

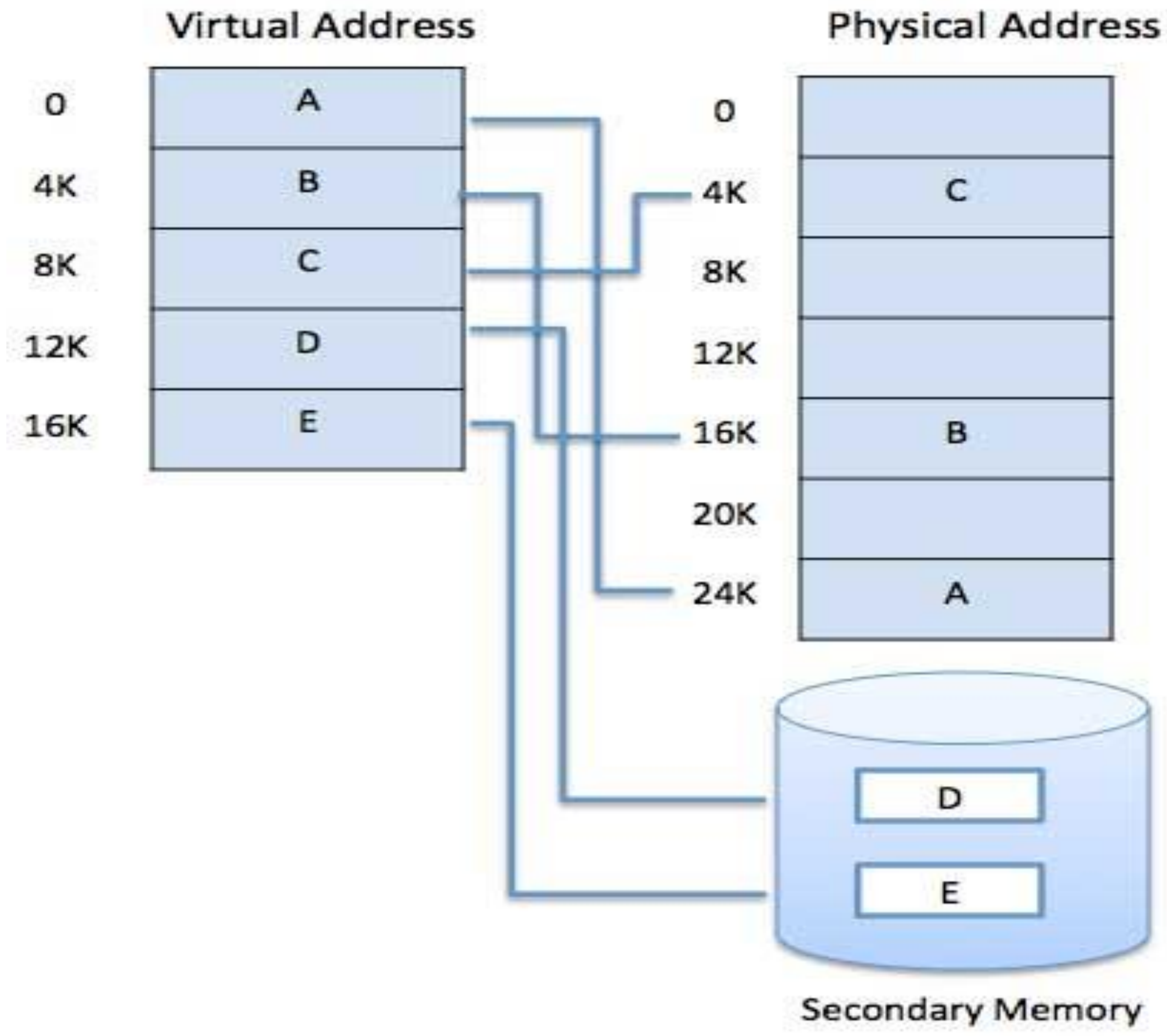
Virtual Memory

Virtual Memory

- Real, or physical, memory exists on RAM chips inside the computer. Virtual memory, as its name suggests, doesn't physically exist on a memory chip. It is an *optimization technique* and is implemented by the operating system in order to give an application program the impression that it has more memory than actually exists.

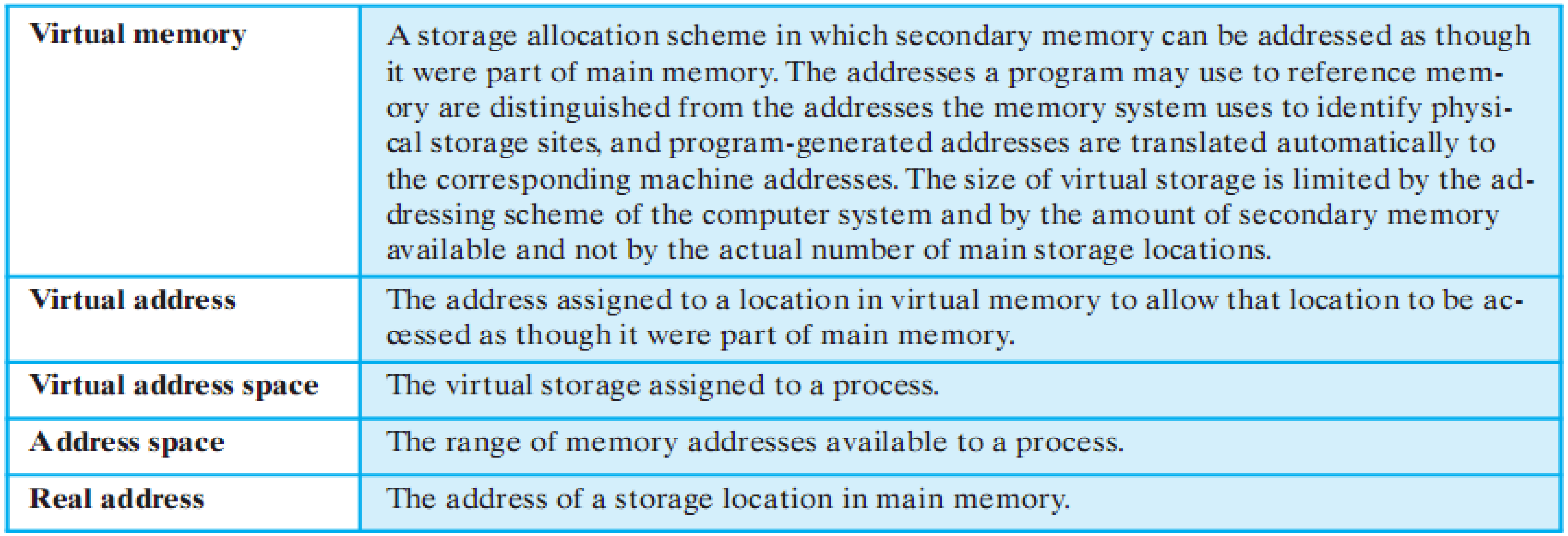
- Virtual memory is implemented by various operating systems such as Windows, Mac OS X, and Linux.
- So how does virtual memory work? Let's say that an operating system needs 120 MB of memory in order to hold all the running programs, but there's currently only 50 MB of available physical memory stored on the RAM chips.

- The operating system will then set up 120 MB of virtual memory, and will use a program called the virtual memory manager (VMM) to manage that 120 MB. The VMM will create a file on the hard disk that is 70 MB (120 – 50) in size to account for the extra memory that's needed. The O.S. will now proceed to address memory as if there were actually 120 MB of real memory stored on the RAM, even though there's really only 50 MB. So, to the O.S., it now appears as if the full 120 MB actually exists. It is the responsibility of the VMM to deal with the fact that there is only 50 MB of real memory.


Virtual Address
Physical Address
0
4K
8K
12K
16K
A
B
C
D
E
0
4K
8K
12K
16K
20K
24K
C
B
A
D
E
Secondary Memory

Terminology

Table 8.1 **Virtual Memory Terminology**



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

Advantages

More Processes are maintained in main memory

A Process mat be larger than all of 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

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

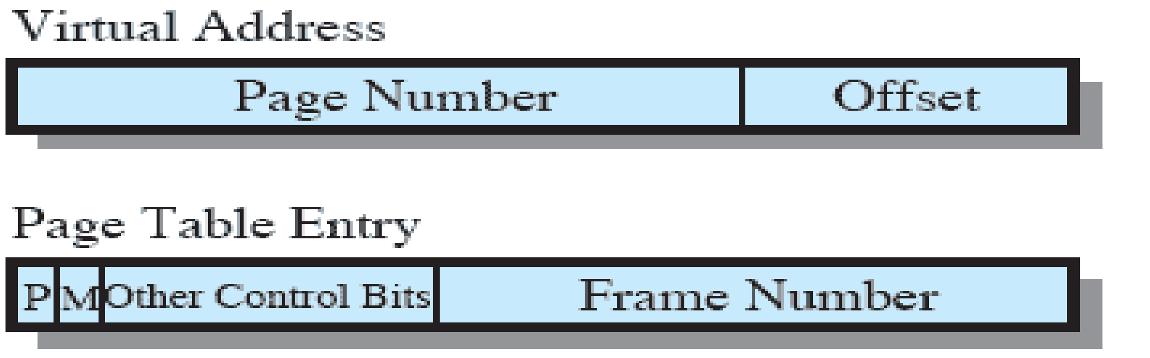
Execution of a Process

- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory. It is called page fault or segmentation fault.
- Operating system places the process in a blocking state
- Piece of process that contains the logical address is brought into main memory
  - Operating system issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the operating system to place the affected process in the Ready state

Virtual Paging / Demand Paging

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

Paging Table



(a) Paging only

P – indicates whether the corresponding page is present in memory or not

P = 1 , Page is in main memory and it denotes the page frame address.

P = 0, it indicates the page fault, page frame address gives the secondary memory address. Page fault handler brings the faulted page from the sec. memory to main memory and changes P = 1 by changing the frame number field with the new page frame address.

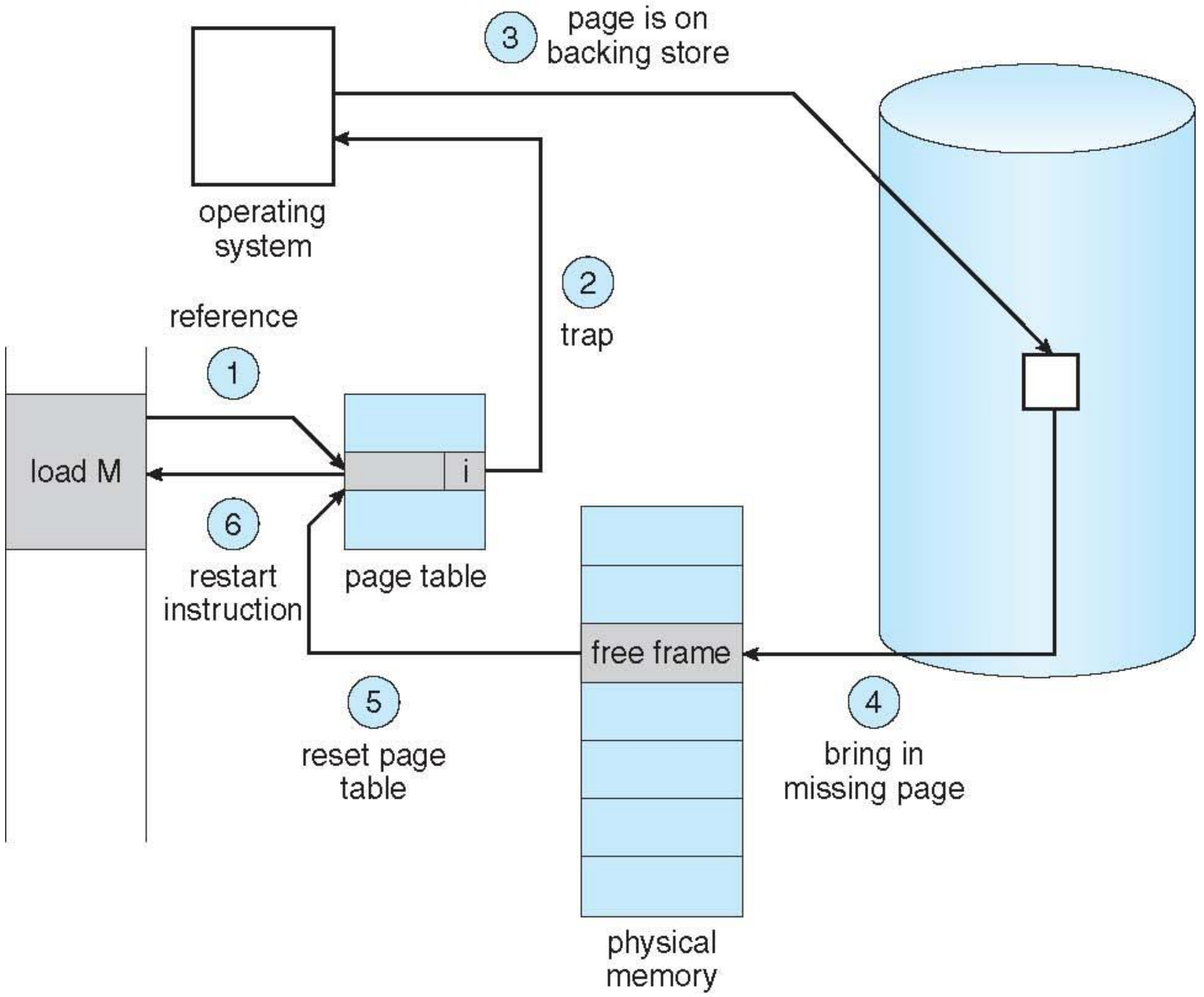
M (Modify bit) – indicating whether the contents of the corresponding page have been altered since the page was last loaded into main memory

M = 1, it indicates that the page has been modified and the copy in the lower levels of memory has to be updated.

Protection bits - used to grant permission for some level of users like supervisor, user, kernel etc.

R/W/X - Read / write / execute - Access rights for particular page.

Steps in Handling a Page Fault



Address Mapping

- Direct Mapping
  - Multilevel paging
  - Inverted page table method

- Associative Mapping

Address Translation

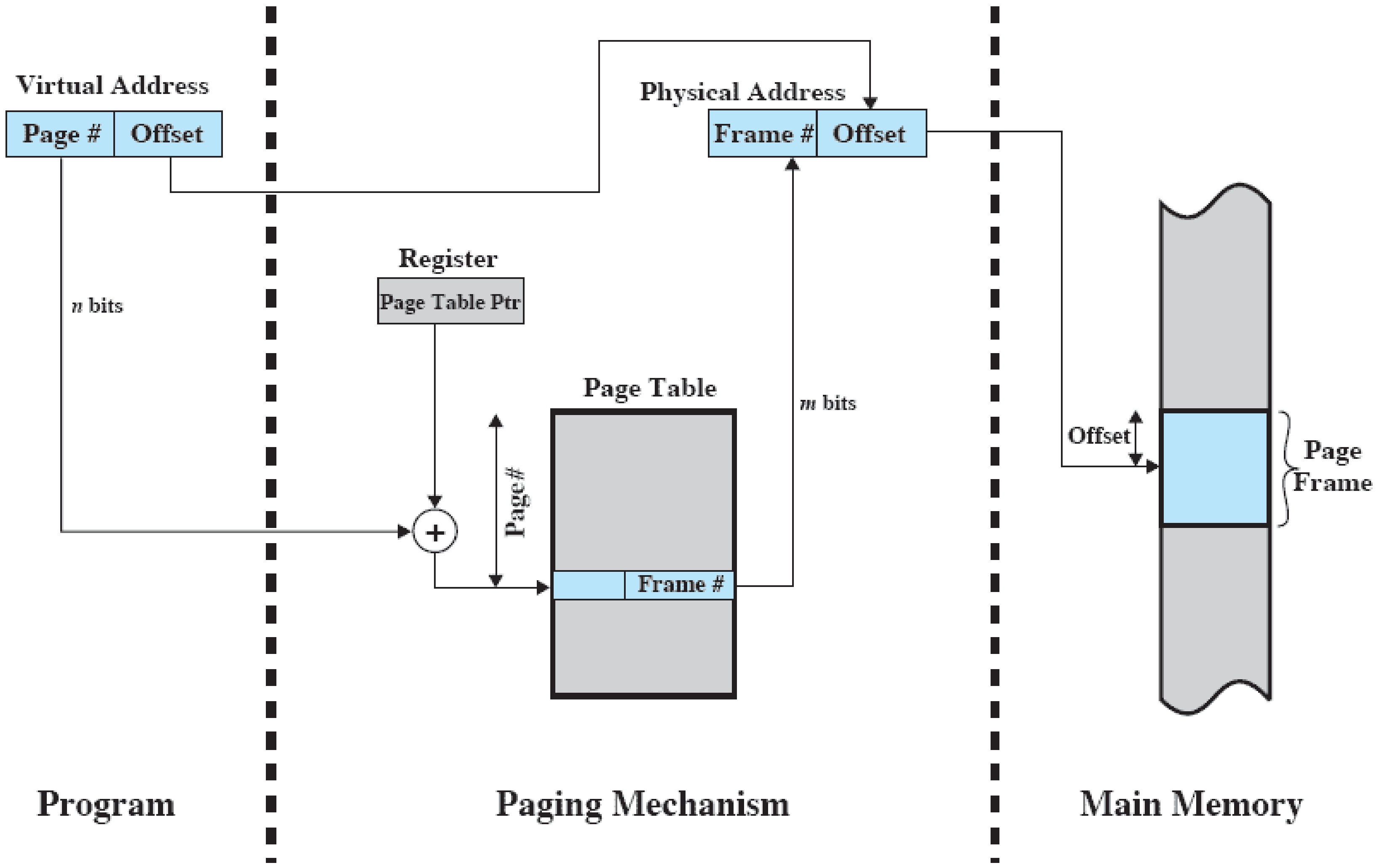


Figure 8.3 Address Translation in a Paging System

Page Tables

- Page tables are also stored in virtual memory
- When a process is running, part of its page table is in main memory

Two-Level Hierarchical Page Table

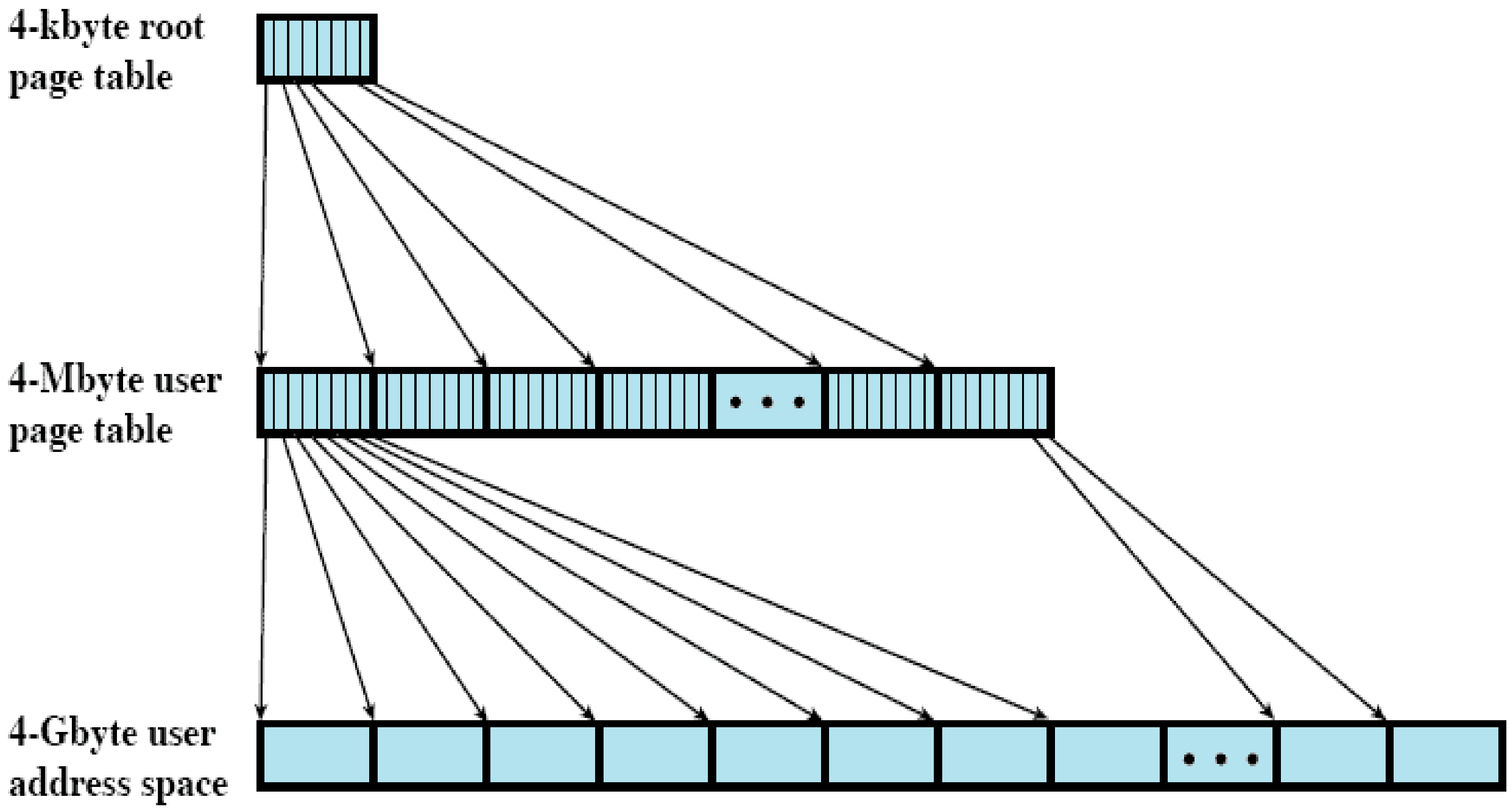


Figure 8.4 A Two-Level Hierarchical Page Table

Multi-Level Paging

- Modern computer systems support a large logical address space (2^{32} to 2^{64}), in such environment, the page table itself becomes excessively large.

- Eg. Consider a system with 32 bit logical address space. If we assume byte level addressing and if the page size is 4 KB (2^{12} pages) then a page table may consists of upto 1 million entries (2^{32} / 2^{12}) = 2^{20} .
- To overcome this problem multi-level paging scheme is used.
- In 2- level scheme if the length of the page directory is X and if the Max. length of a page table is Y then process can consists of upto XY pages.

Address Translation for Hierarchical page table

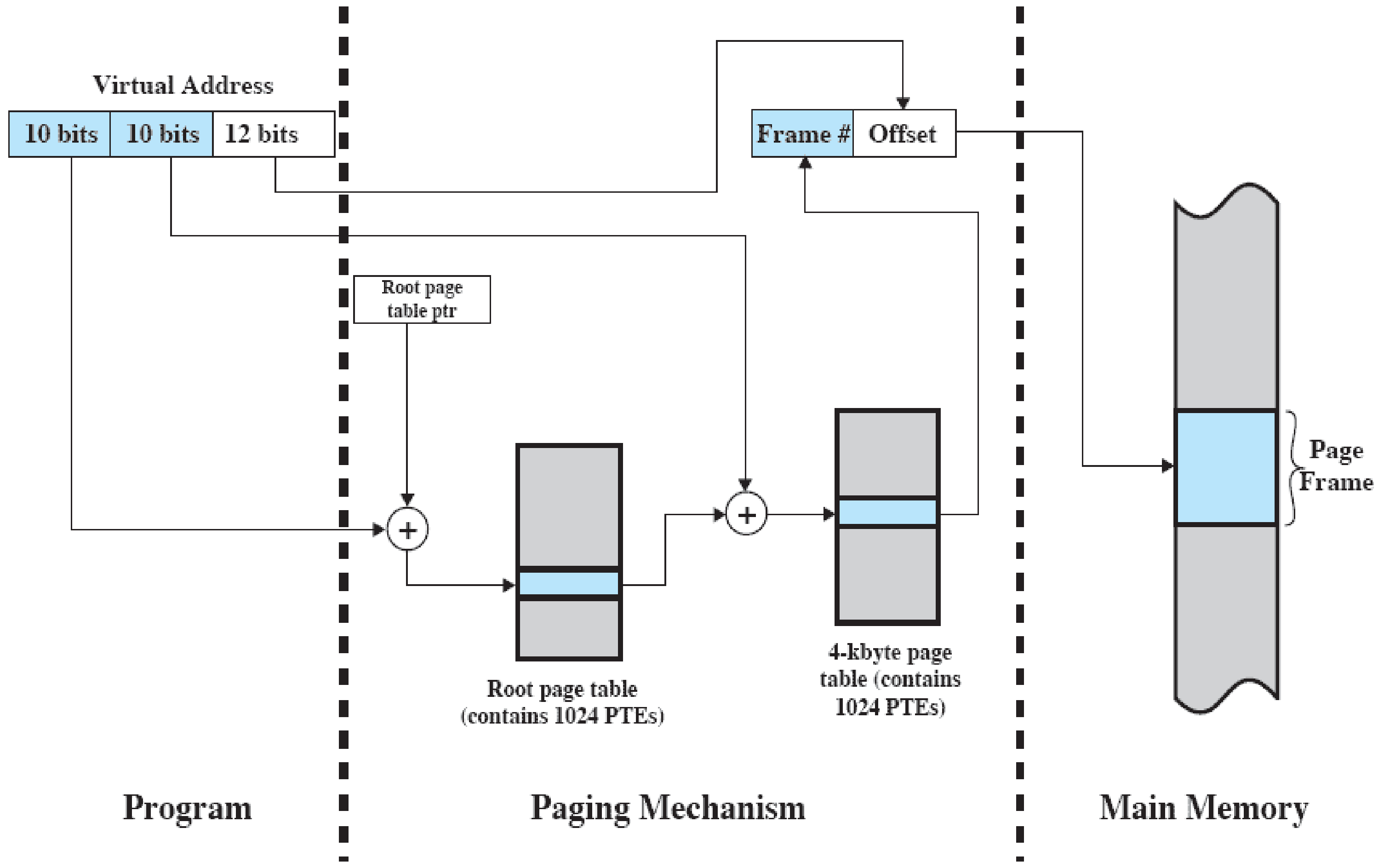


Figure 8.5 Address Translation in a Two-Level Paging System

Page tables grow proportionally

- A drawback of the type of page tables just discussed is that their size is proportional to that of the virtual address space.

- An alternative is Inverted Page Tables

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
- More than one virtual address may map into same hash table entry.
- A chaining technique is used to manage the overflow.
- The page table structure is called inverted because it indexes page table entries by frame number rather than by virtual page number.

Inverted Page Table

Each entry in the page table includes:

- Page number
- Process identifier
  - The process that owns this page.
- Control bits
  - includes flags, such as valid, referenced, etc
- Chain pointer
  - the index value of the next entry in the chain.

Inverted Page Table

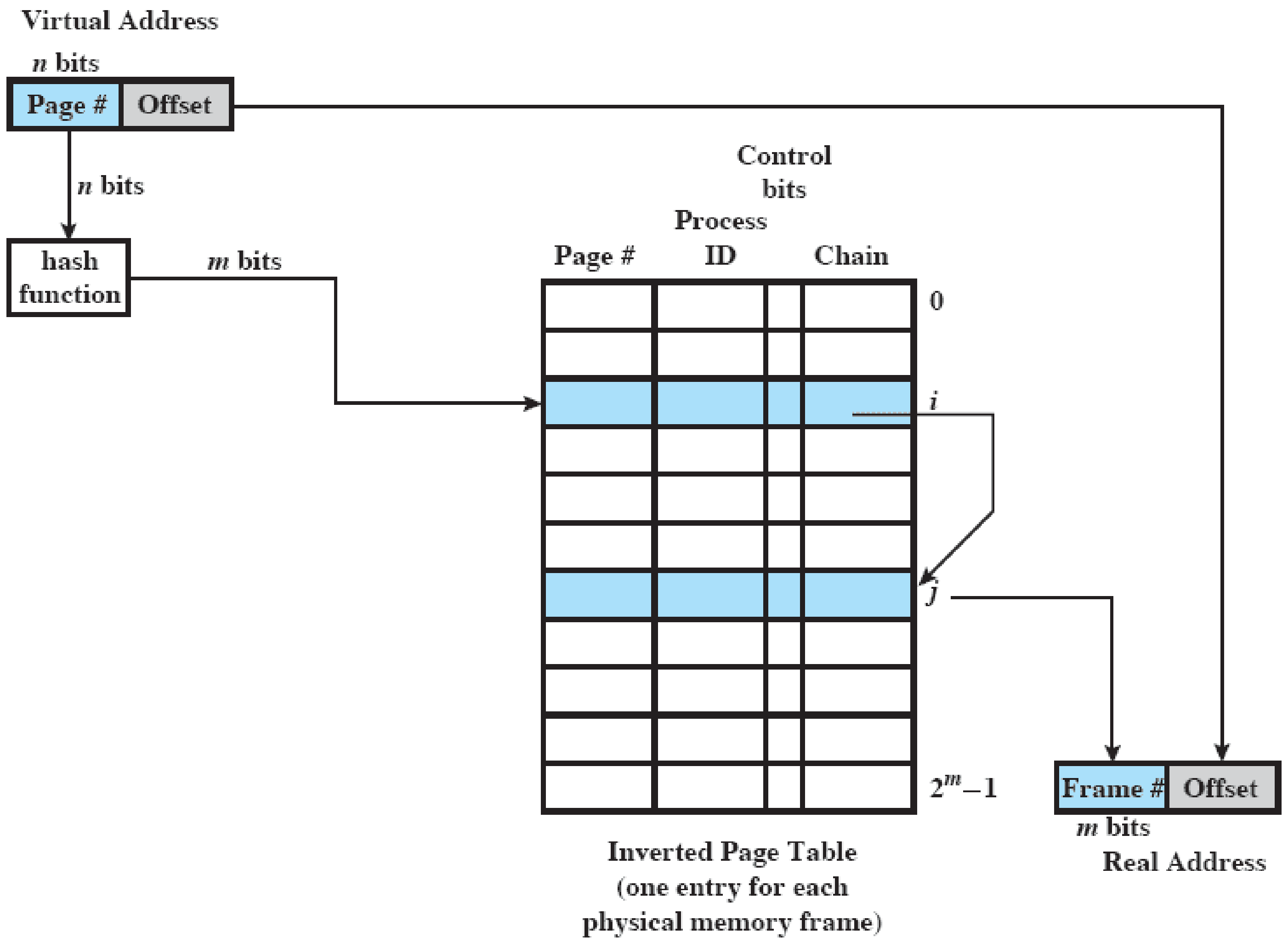


Figure 8.6 Inverted Page Table Structure

Associative Mapping - Translation Lookaside Buffer

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- 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 not in main memory a page fault is issued
- The TLB is updated to include the new page entry

Associative Mapping

- As the TLB only contains some of the page table entries, we cannot simply index into the TLB based on the page number
  - Each TLB entry must include the page number as well as the complete page table entry

- The process is able to simultaneously query numerous TLB entries to determine if there is a page number match

Translation Lookaside Buffer (TLB)

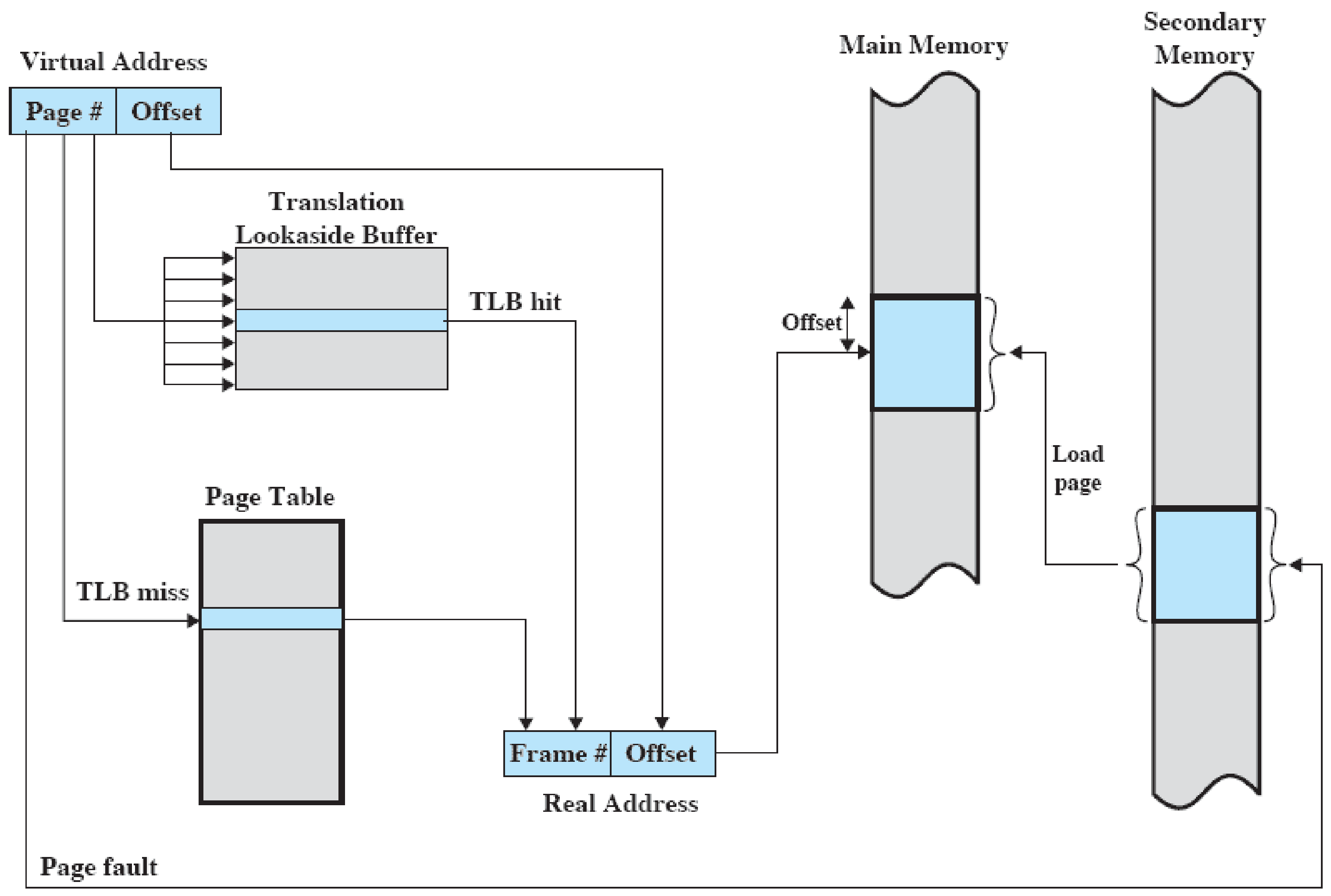


Figure 8.7 Use of a Translation Lookaside Buffer

TLB operation

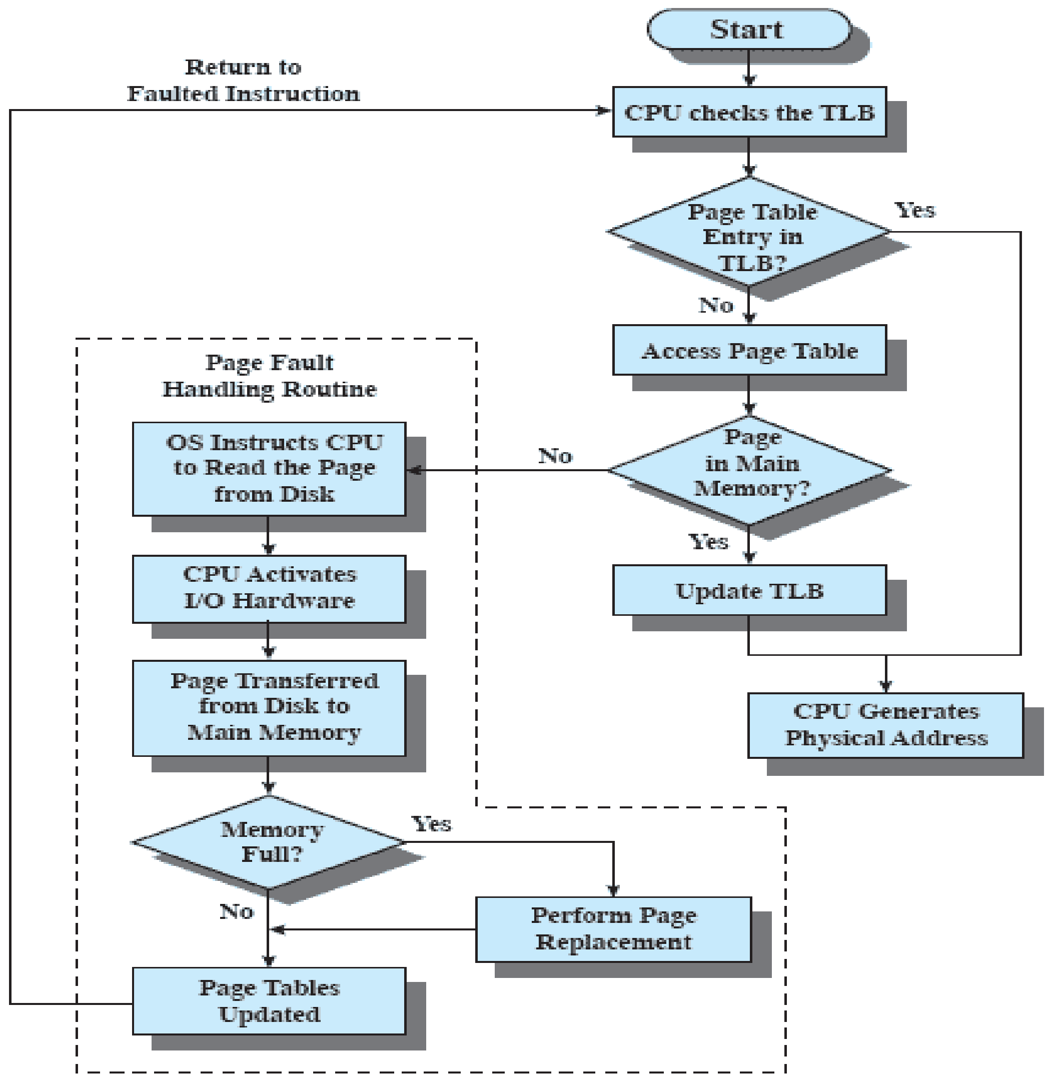

Start
CPU checks the TLB
Page Table Entry in TLB?
Yes
No
Access Page Table
Page in Main Memory?
No
Yes
Update TLB
CPU Generates Physical Address
Return to Faulted Instruction
Page Fault Handling Routine
OS Instructs CPU to Read the Page from Disk
CPU Activates I/O Hardware
Page Transferred from Disk to Main Memory
Memory Full?
Yes
No
Perform Page Replacement
Page Tables Updated

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

Translation Look aside Buffer

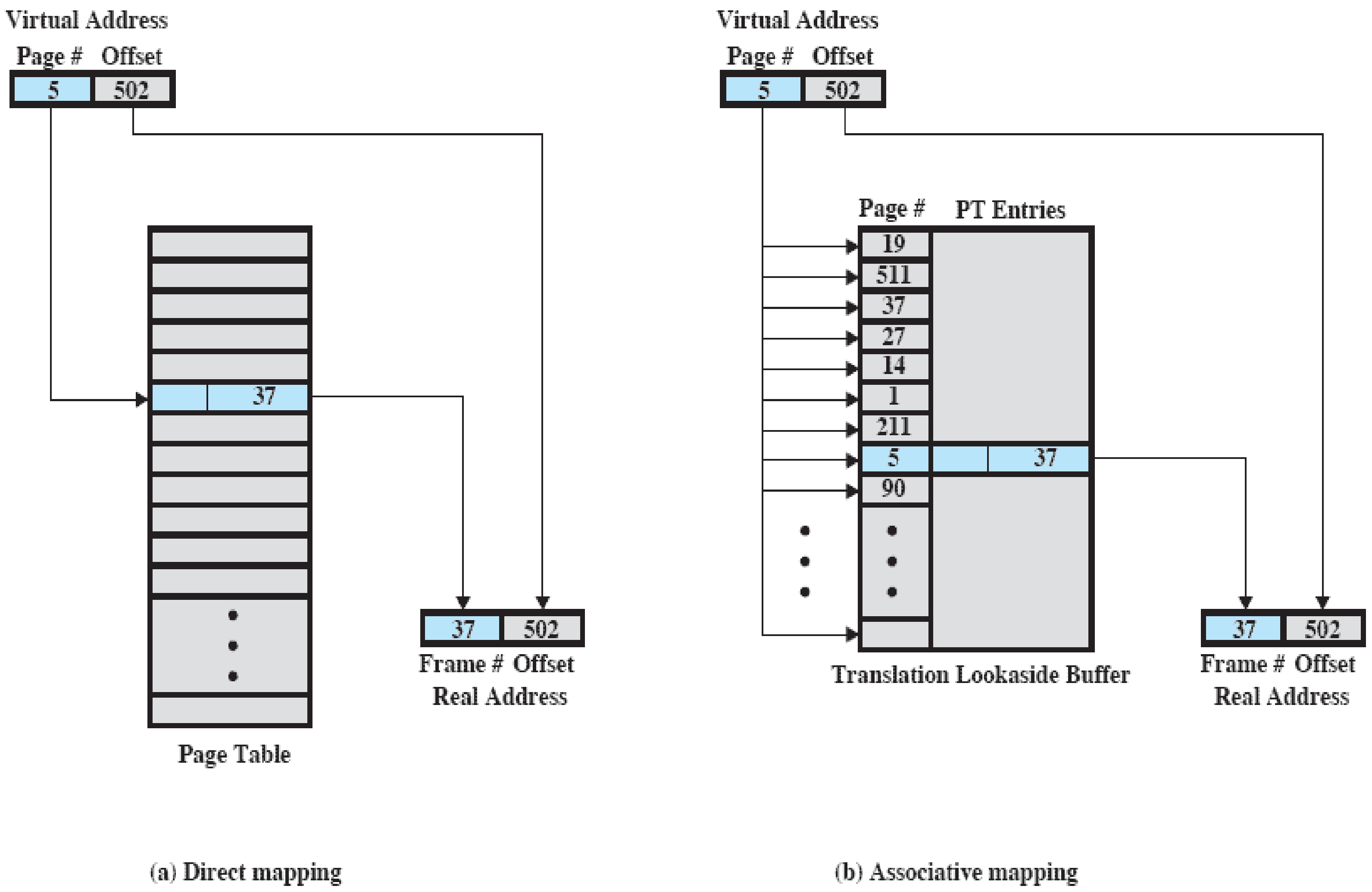


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

TLB and Cache Operation

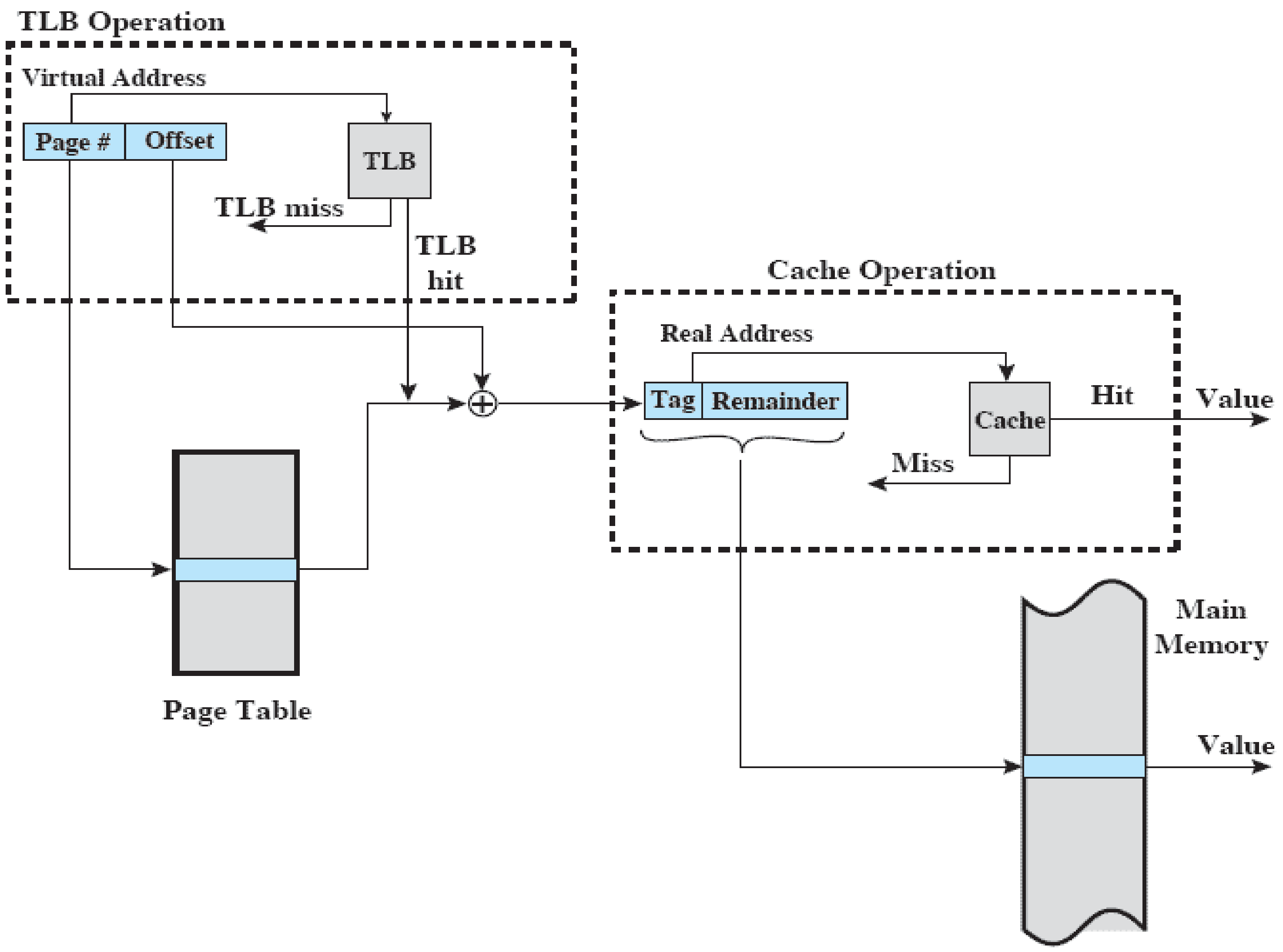


Figure 8.10 Translation Lookaside Buffer and Cache Operation

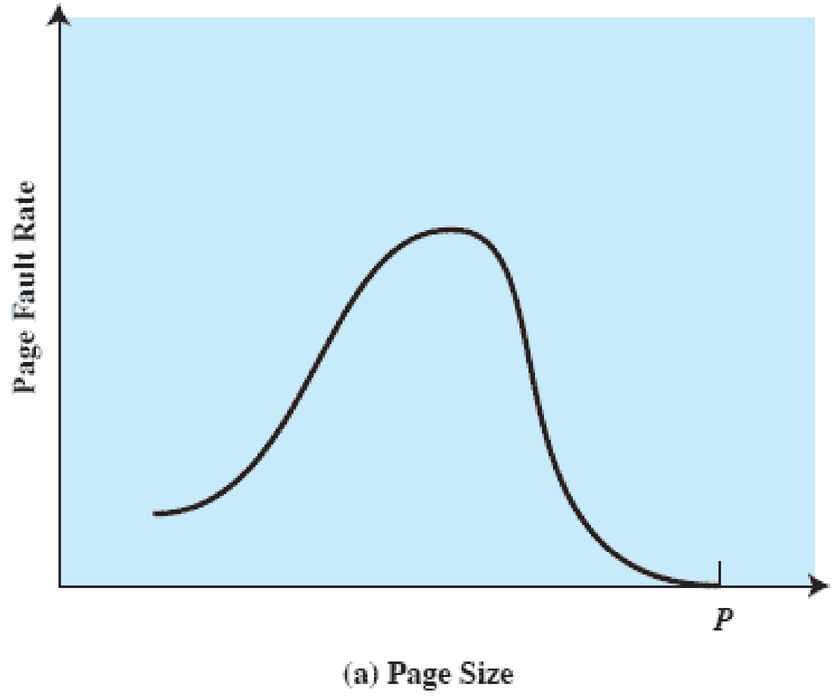
Page Size

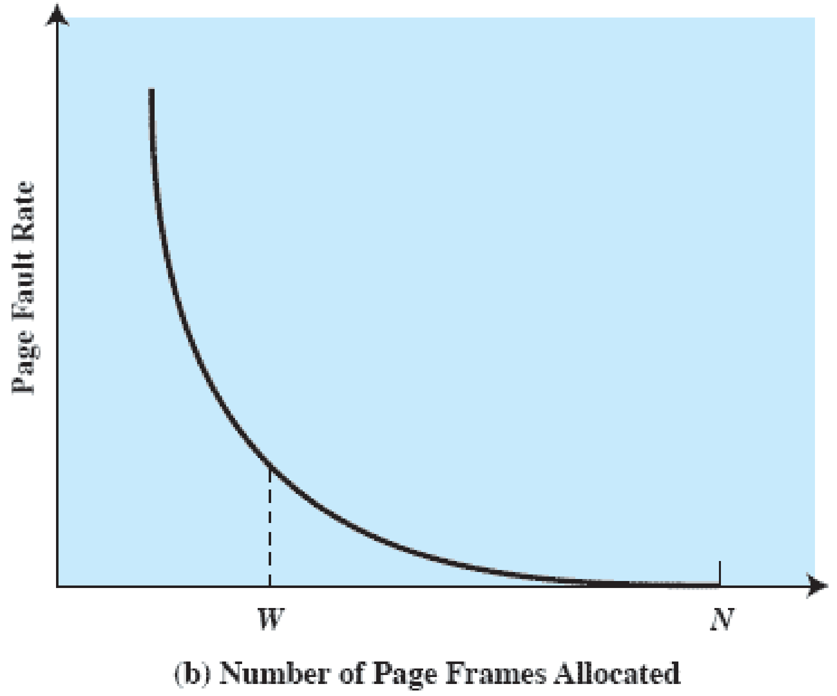
- Smaller page size, less amount of internal fragmentation
- But Smaller page size, more pages required per process
  - More pages per process means larger page tables
- Larger page tables means large portion of page tables in virtual memory
- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Further complications to Page Size

- 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.

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

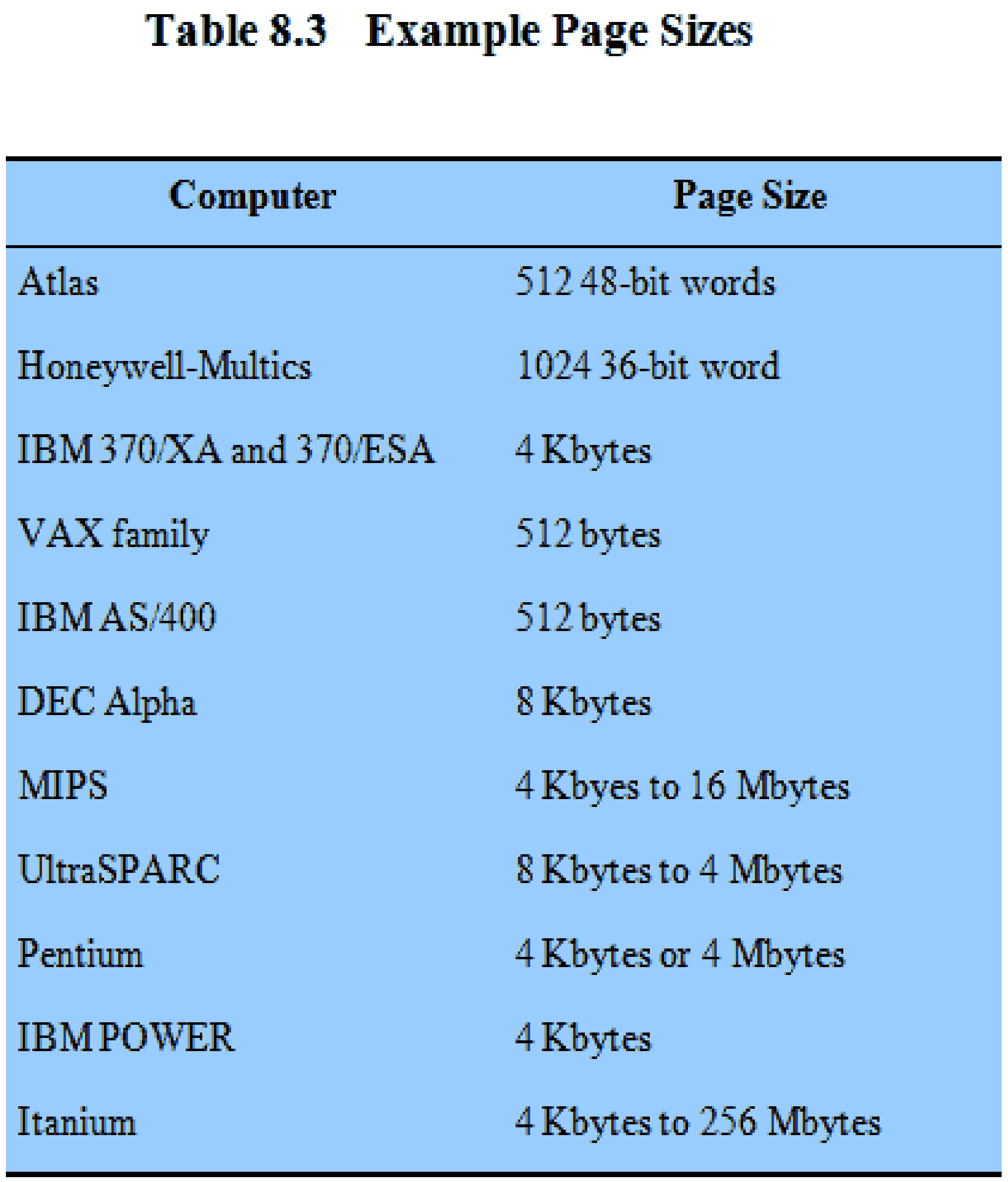


Table 8.3 Example Page Sizes

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

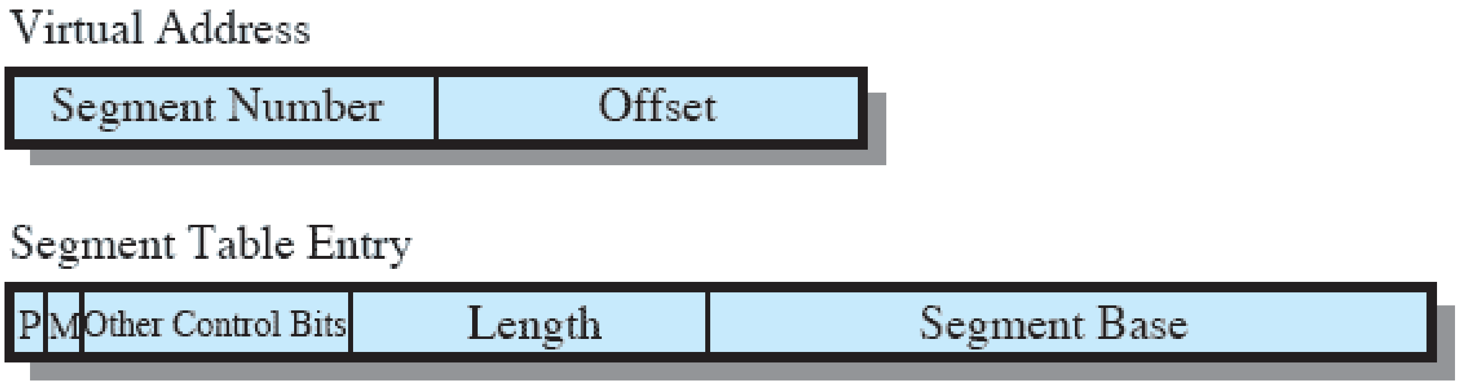
Virtual Segmentation / Demand Segmentation

- Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments.
  - May be unequal, dynamic size
  - Simplifies handling of growing data structures
  - Allows programs to be altered and recompiled independently
  - Lends itself to sharing data among processes
  - Lends itself to protection

Segment 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

Segment Table Entries



(b) Segmentation only

Address Translation in Segmentation

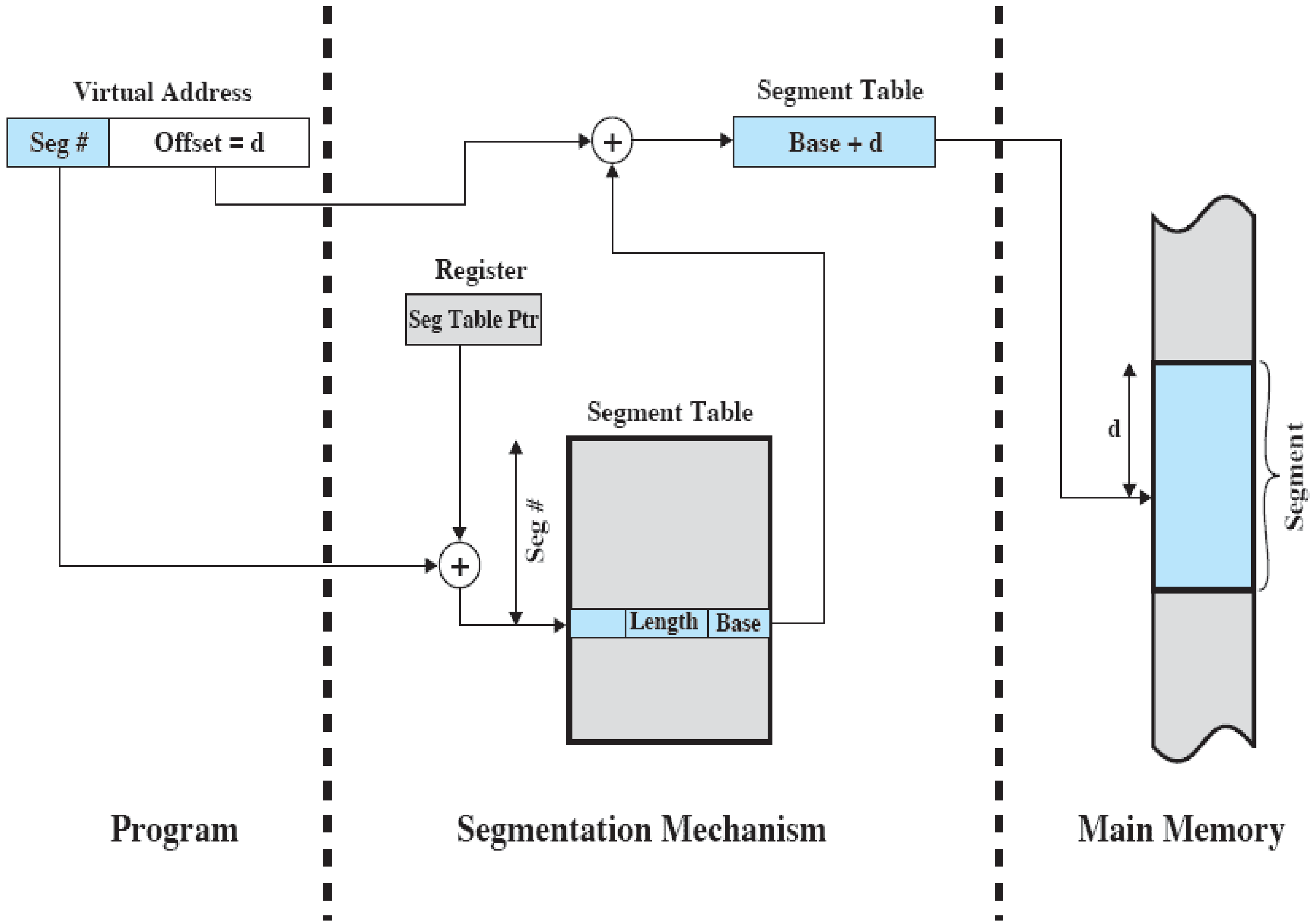


Figure 8.12 Address Translation in a Segmentation System

Combined Paging and Segmentation

<u>Paging Advantages</u>

- Address Mapping is easier

<u>Paging disadvantages</u>

- Paging is transparent to the programmer
- Internal Fragmentation

<u>Segmentation Advantages</u>

- Segmentation is visible to the programmer
- Support modular programming

<u>Segmentation disadvantages</u>

- External fragmentation
- Superfluity - small fraction of segment is used

Combined Paging and Segmentation

To combine the advantages of both paging and segmentation, a new scheme have been developed.

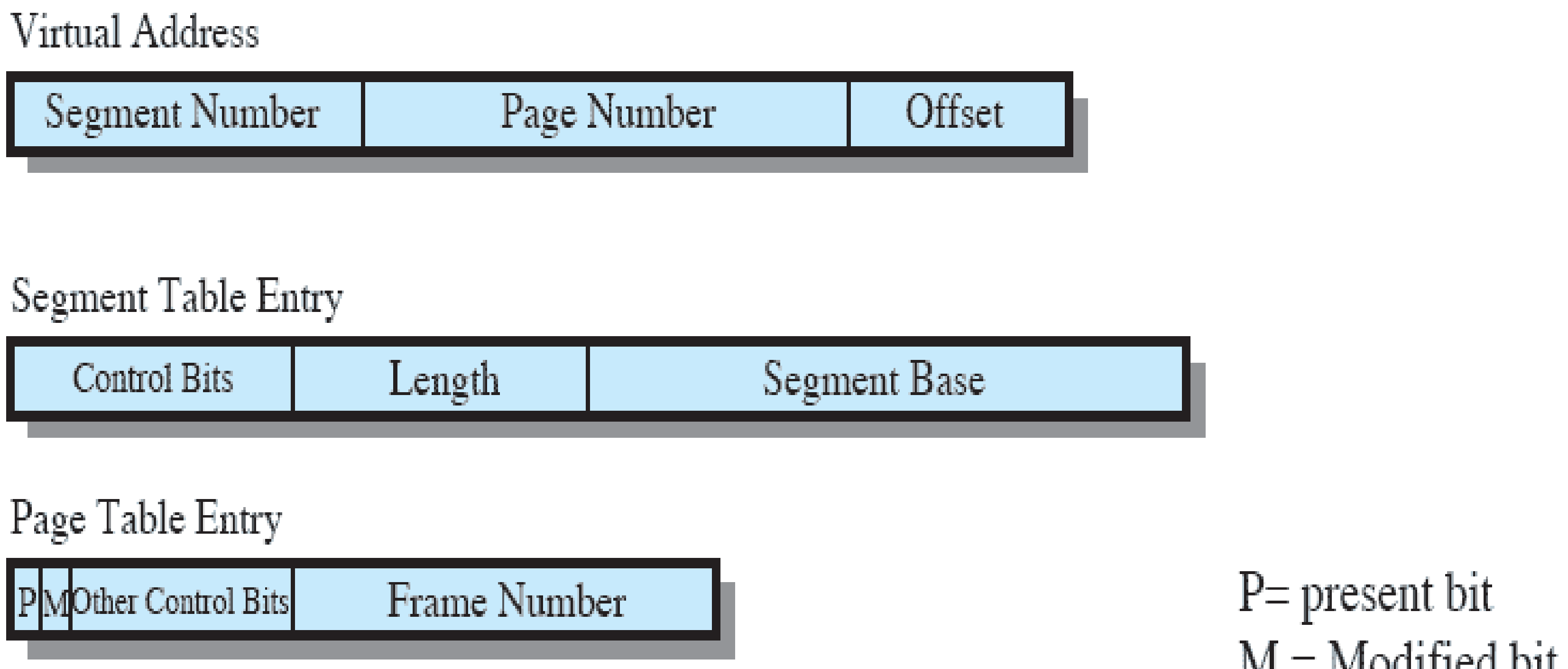
1. Paged segmentation – Paging characteristics dominates

A User's address space is divided into number of segments. Each segment is broken up into number of fixed size pages.

2. Segmented paging – Segmentation characteristics dominates

A User's address space is divided into number of pages. Each page is broken up into number of segments.

Combined Paging and Segmentation



(c) Combined segmentation and paging

Address Translation

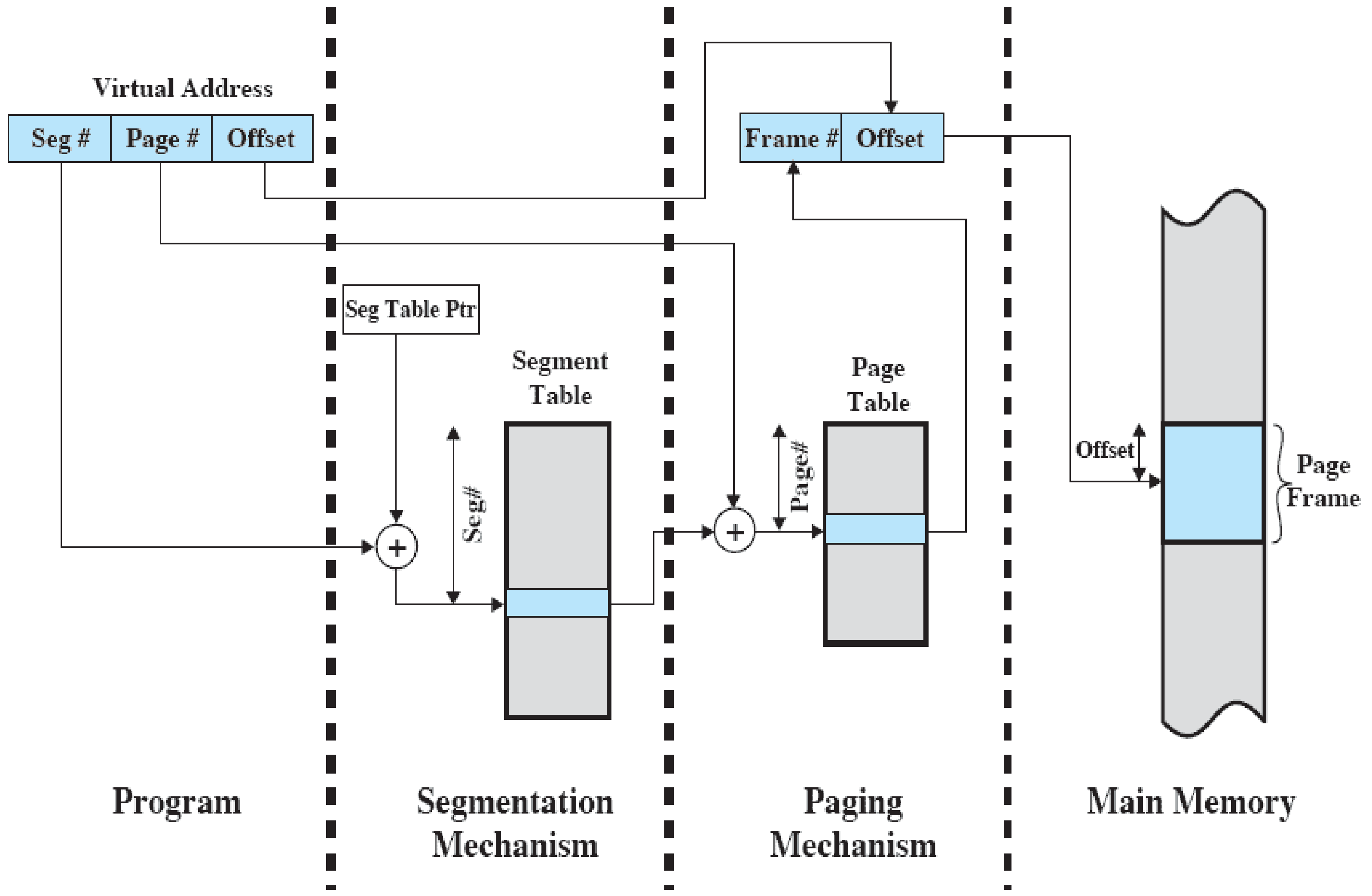


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection and sharing

- Page and Segment can be shared among the processes.
- If the code is **Re-entrant code**, it can be shared easily. Re-entrant code is non-self modifiable code. It never changes during execution.
- Segmentation lends itself to the implementation of protection and sharing policies.
- As each entry has a base address and length, inadvertent memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes

Protection Relationships

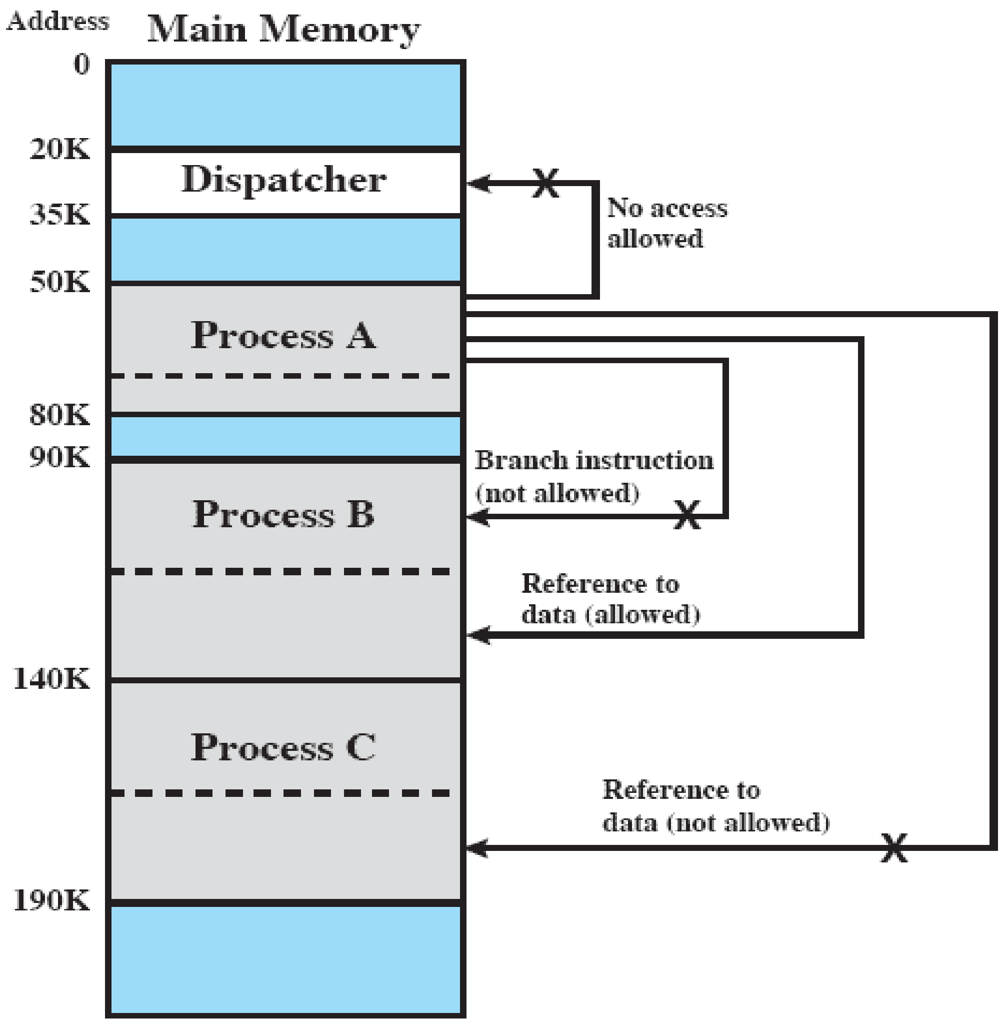


Figure 8.14 Protection Relationships Between Segments

Thrashing

- A state in which the system spends most of its time swapping pieces rather than executing instructions.
- To avoid this, the operating system tries to guess which pieces are least likely to be used in the near future.
    - The guess is based on recent history

Principle of Locality

- Program and data references within a process tend to cluster
- Only a few pieces of a process will be needed over a short period of time
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

Operating System Software

The design of the memory management portion of an operating system depends on three fundamental areas of choice :

1. Whether or not to use virtual memory techniques
2. The use of paging and segmentation or both
3. The algorithm employed for various aspects of memory management

The choice of first two areas depend on the hardware platform available. The choice related to third item are the domain of the OS software.

Operating System Software – Memory Management Policies

1. Fetch Policy - decides when a page should be brought into main memory from the lower levels of memory
2. Placement Policy - decides where in main memory a process piece to reside
3. Replacement Policy - decides which piece has to be removed or replaced to accommodate newly fetched processes.

Fetch Policy

Two Types

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

Placement Policy (Algorithms)

- Operating system must decide which free block to allocate to a process. Four methods.
- First-fit
- Next-fit
- Best-fit
- Worst-fit

Placement Policy (Algorithms)

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

Placement Policy (Algorithms)

- 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

Placement Policy (Algorithms)

- Best-fit algorithm
  - 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

Placement Policy (Algorithms)

- Worst-fit algorithm
  - Chooses the block that is largest in size among all the blocks to the request
  - Best performer overall
  - Since largest block is found for process, the reasonable amount of fragmentation is left. So that next incoming process may be accommodated.

Allocation

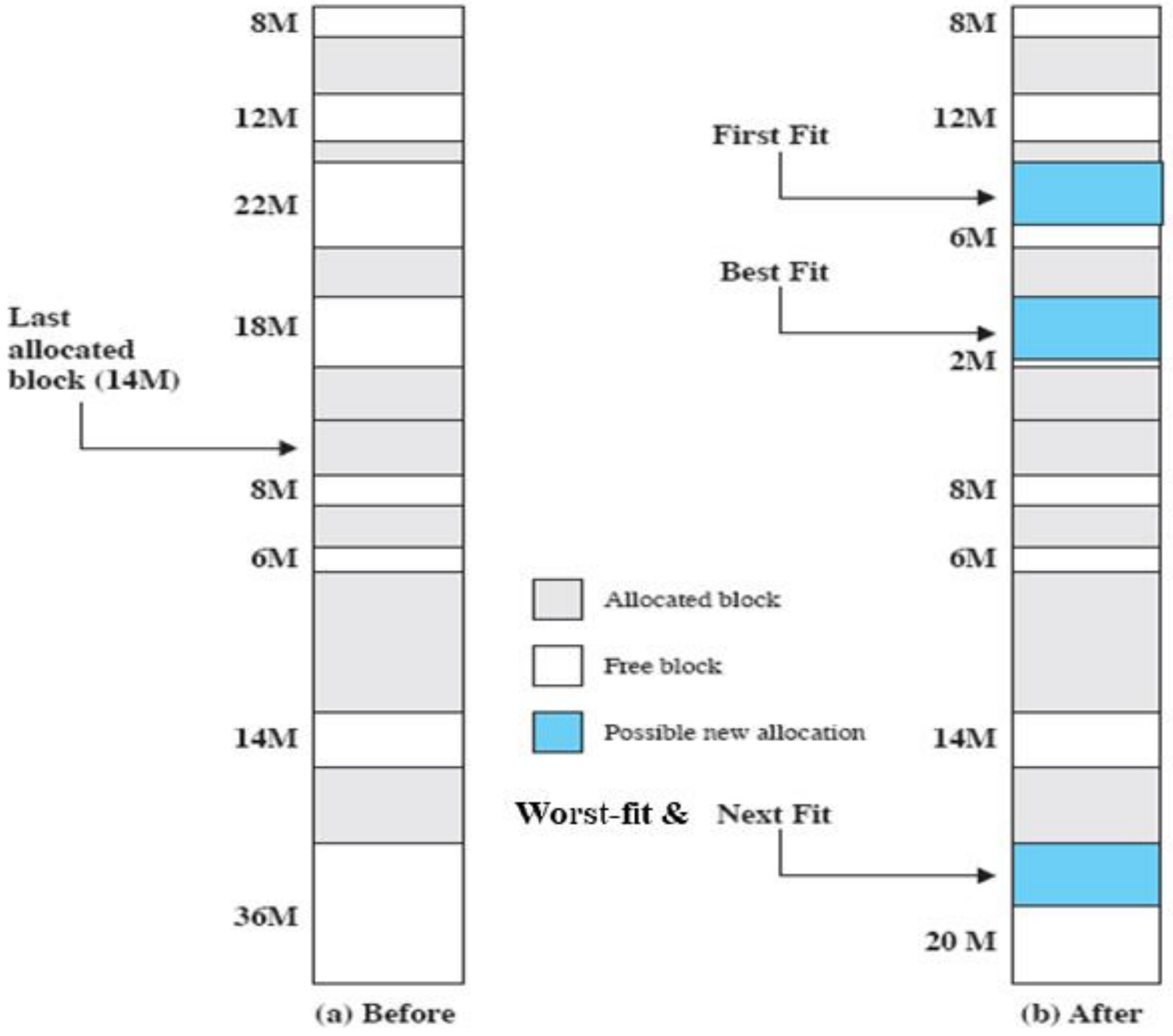


Figure Example Memory Configuration before and after Allocation of 16-Mbyte Block

Replacement Policy (Algorithms)

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

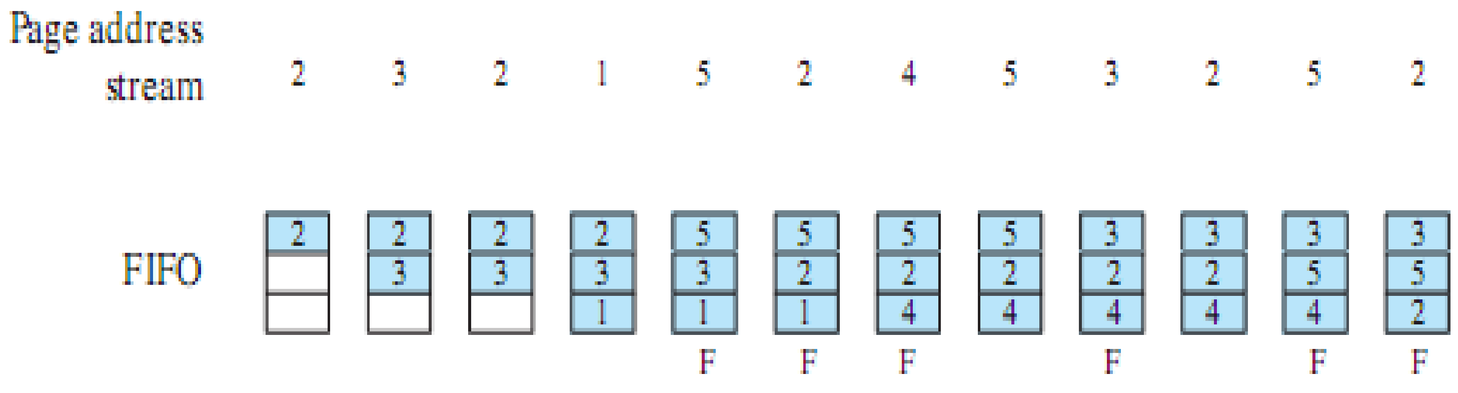
Examples

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

First-in, first-out (FIFO)

- It Replaces the oldest frame that was brought into memory i.e Page that has been in memory the longest is replaced
  - But, these pages may be needed again very soon if it hasn’t truly fallen out of use
- It associates time with each page.
- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
  - Simplest replacement policy to implement

FIFO Example



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

Figure 8.15 Behavior of Four Page Replacement Algorithms

- The FIFO policy results in six page faults.
  – Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

Example

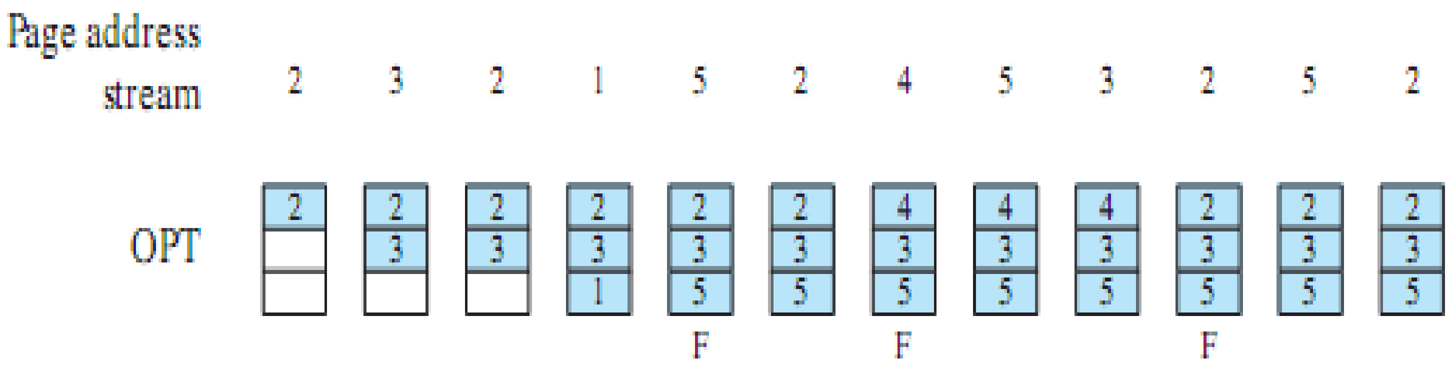
2 3 2 1 5 2 4 5 3 2 5 2

2	**2**	**2**	**2**	**5**	**5**	**5**	**5**	**3**	**3**	**3**	**3**
	3	3	3	3	2	2	2	2	2	5	5
			1	1	1	4	4	4	4	4	2
				F	F	F	NO	F	No	F	F

Optimal policy

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

Optimal Policy Example



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

Figure 8.15 Behavior of Four Page Replacement Algorithms

- The optimal policy produces three page faults after the frame allocation has been filled.

Example

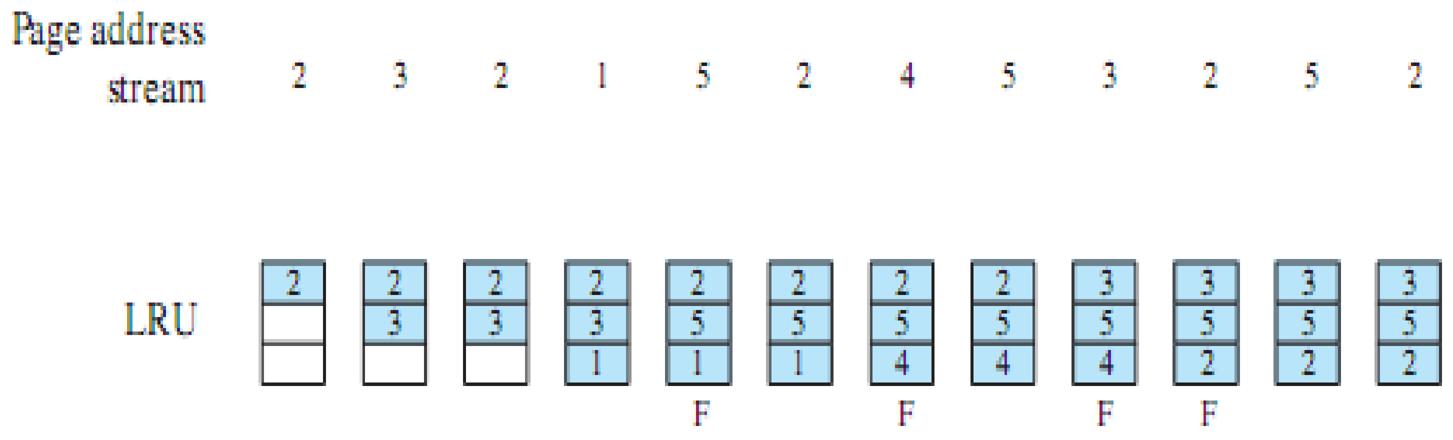
2 3 2 1 5 2 4 5 3 2 5 2

2	**2**	**2**	**2**	**2**	**4**	**4**	**4**	**2**			
	3	3	3	3	3	3	3	3			
			1	5	5	5	5	5			
			F	No	F	No	No	F			

Least Recently Used (LRU)

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement
  - One approach is to tag each page with the time of last reference.
  - This requires a great deal of overhead.

LRU Example



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

Figure 8.15 Behavior of Four Page Replacement Algorithms

optimal policy.

– In this example, there are four page faults

Example

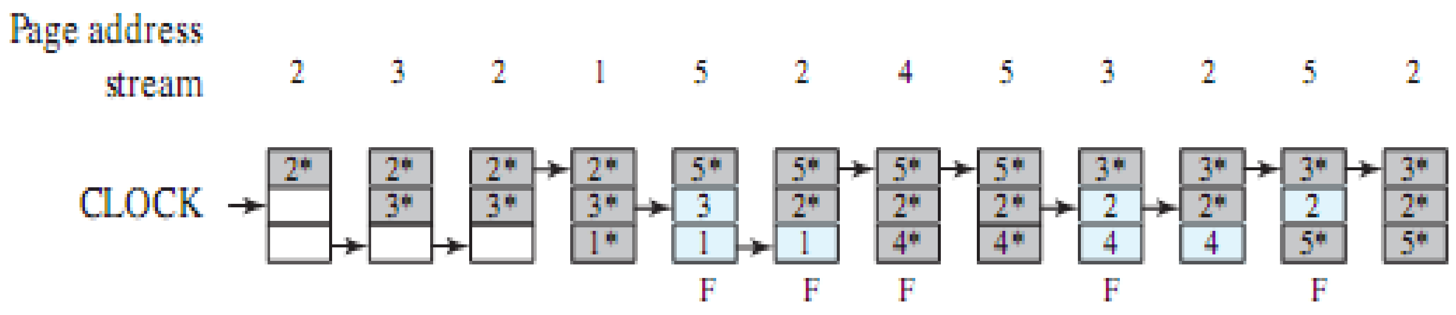
2 3 2 1 5 2 4 5 3 2 5 2

2	**2**	**2**	**2**	**2**	**2**	**2**	**2**	**3**	**3**		
	3	3	3	5	5	5	5	5	5		
			1	1	1	4	4	4	2		
				F	No	F	No	F	F		

Clock Policy

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

Clock Policy Example

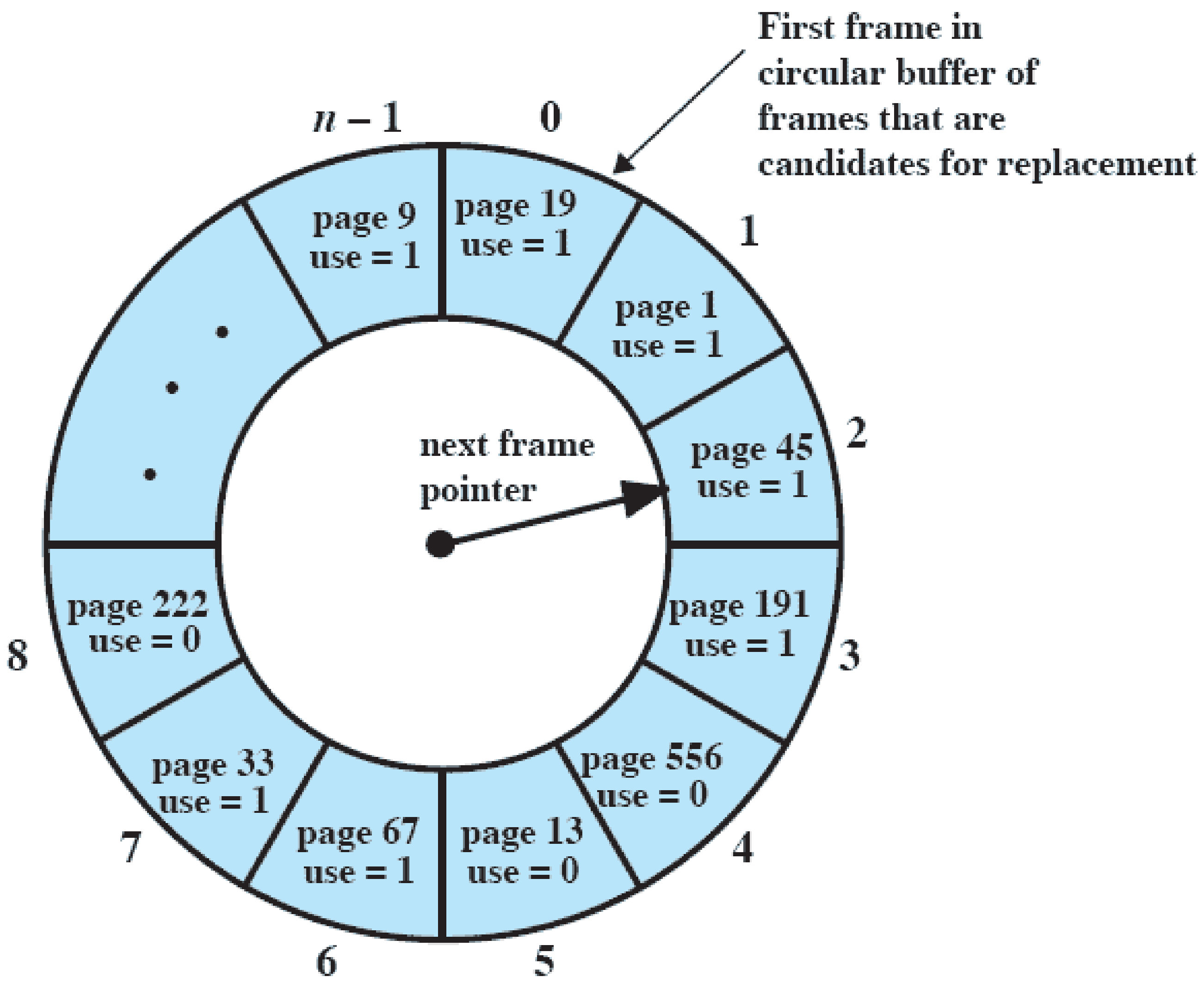


F= page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page Replacement Algorithms

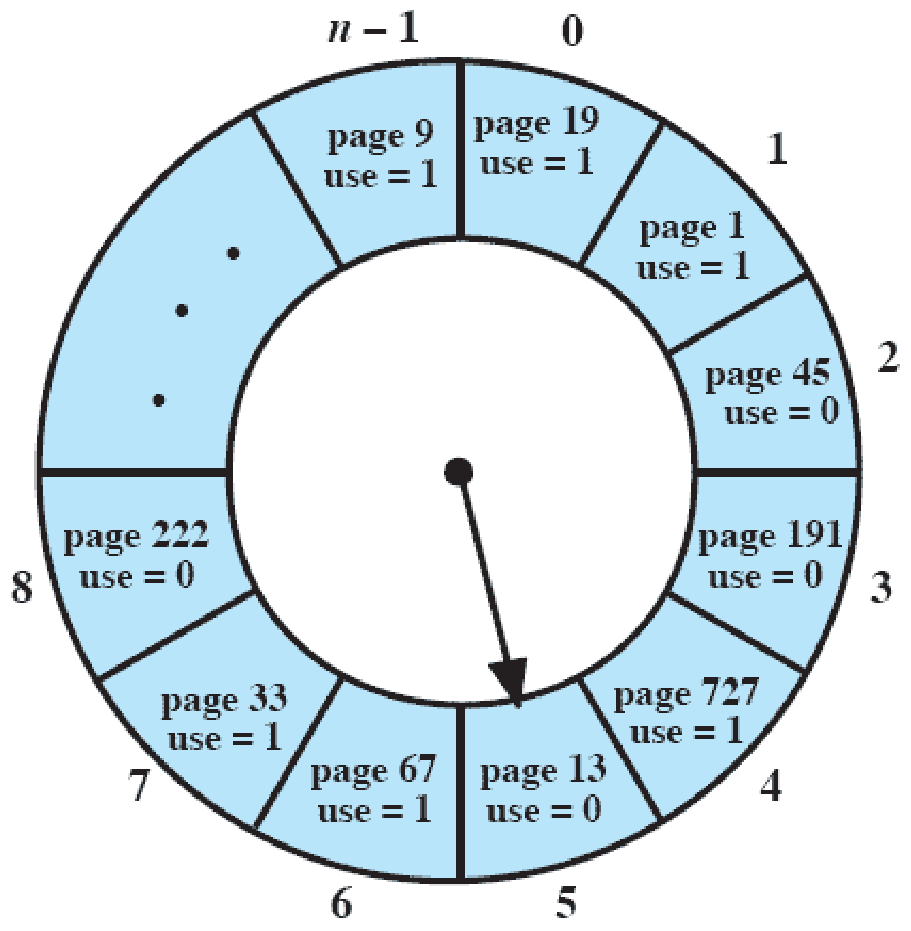
- Note that the clock policy is adept at protecting frames 2 and 5 from replacement.

Clock Policy



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

Clock Policy



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

Figure 8.16 Example of Clock Policy Operation

Clock Policy

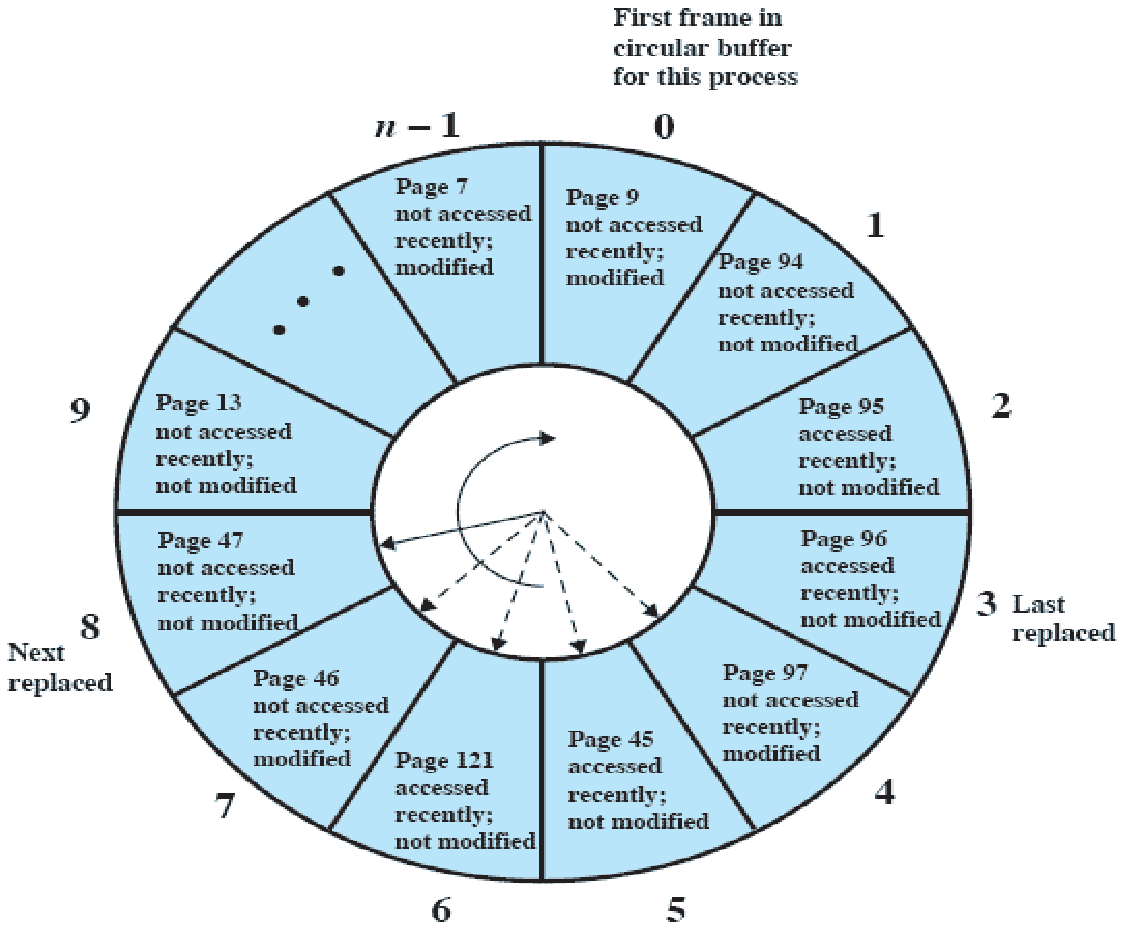
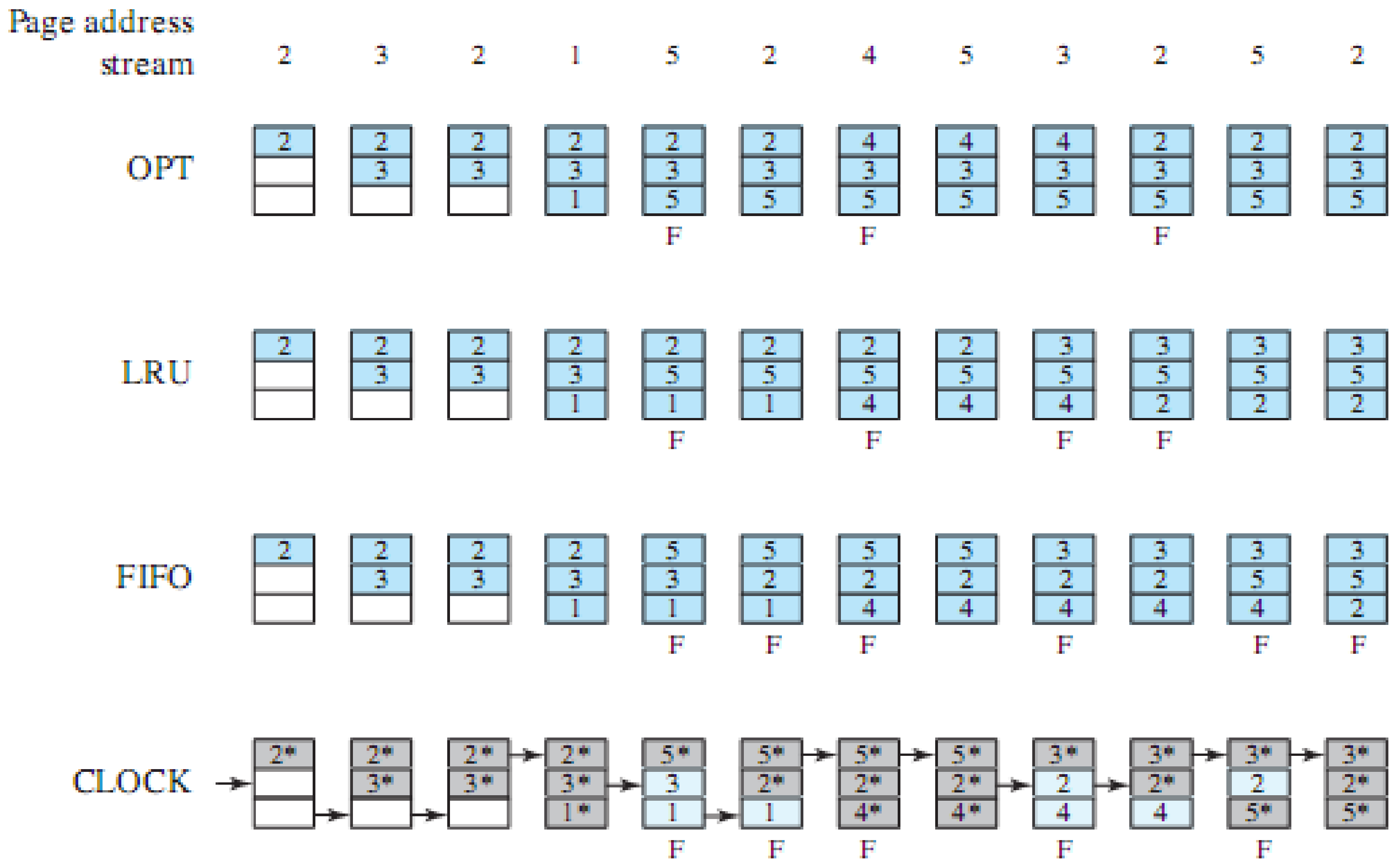

First frame in circular buffer for this process
0
Page 9 not accessed recently; modified
1
Page 94 not accessed recently; not modified
2
Page 95 accessed recently; not modified
3 Last replaced
Page 96 accessed recently; not modified
4
Page 97 not accessed recently; modified
5
Page 45 accessed recently; not modified
6
Page 121 accessed recently; not modified
7
Page 46 not accessed recently; modified
8 Next replaced
Page 47 not accessed recently; not modified
9
Page 13 not accessed recently; not modified
n – 1
Page 7 not accessed recently; modified

Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Combined Examples



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

Figure 8.15 Behavior of Four Page Replacement Algorithms

Comparison

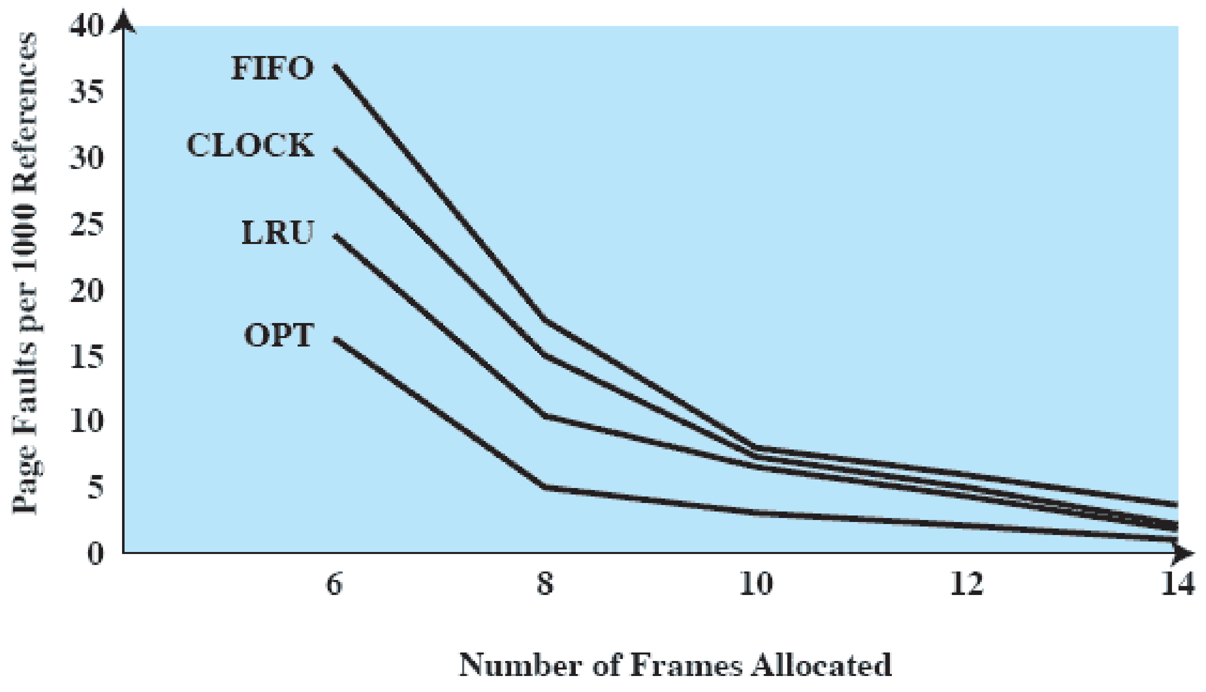


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Page Buffering

- LRU and Clock policies both involve complexity and overhead
  - Also, replacing a modified page is more costly than unmodified as needs written to secondary memory
- Solution: Replaced page is added to one of two lists
  - Free page list if page has not been modified
  - Modified page list

Replacement Policy and Cache Size

- Main memory size is getting larger and the locality of applications is decreasing.
  - So, cache sizes have been increasing
- With large caches, replacement of pages can have a performance impact
  - improve performance by supplementing the page replacement policy with a with a policy for page placement in the page buffer

Resident Set Management

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

Resident Set Management

- Fixed-allocation
  - Gives a process a fixed number of pages within which to execute
  - When a page fault occurs, one of the pages of that process must be replaced
- Variable-allocation
  - Number of pages allocated to a process varies over the lifetime of the process

Replacement Scope

- The scope of a replacement strategy can be categorized as *global* or *local*.
  - Both types are activated by a page fault when there are no free page frames.
  - A local replacement policy chooses only among the resident pages of the process that generated the page fault
  - A global replacement policy considers all unlocked pages in main memory

Fixed Allocation, Local Scope

- Decide ahead of time the amount of allocation to give a process
- If allocation is too small, there will be a high page fault rate
- If allocation is too large there will be too few programs in main memory
  - Increased processor idle time or
  - Increased swapping.

Variable Allocation, Global Scope

- Easiest to implement
  - Adopted by many operating systems
- Operating system keeps list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no free frame, replaces one from another process
  - Therein lies the difficulty … which to replace.

Variable Allocation, Local Scope

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

Resident Set Management Summary

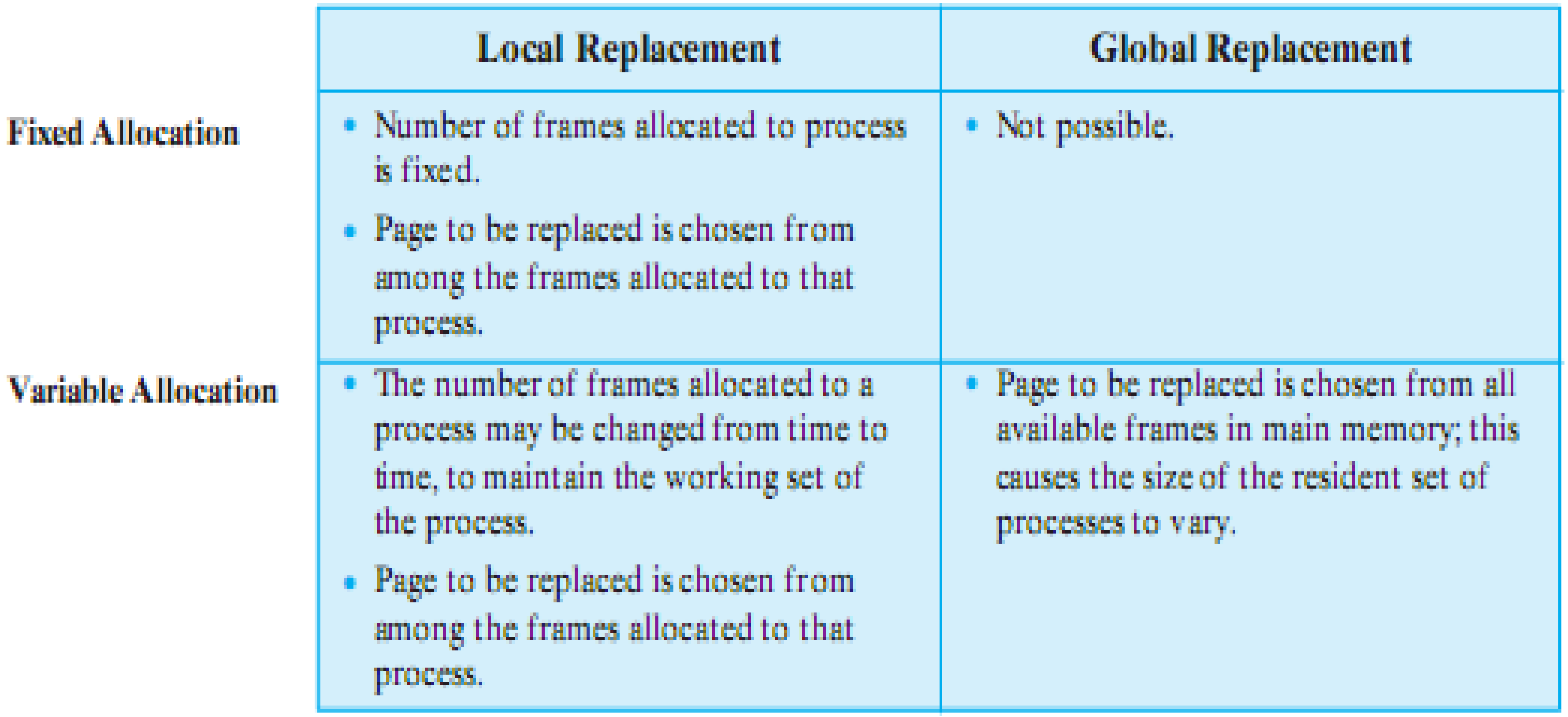


Table 8.5 Resident Set Management

	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.

Cleaning Policy

- A cleaning policy is concerned with determining when a modified page should be written out to secondary memory.
- Demand cleaning
  – A page is written out only when it has been selected for replacement
- Pre-cleaning
  – Pages are written out in batches

Cleaning Policy

- Best approach uses page buffering
- Replaced pages are placed in two lists
  - Modified and unmodified
- Pages in the modified list are periodically written out in batches
- Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Load Control

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