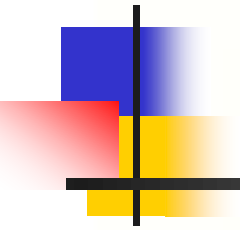


SWENG3043

Operating Systems

Software Engineering Department

AASTU

Dec 2018

Chapter Six

Memory Management

Chapter Objectives

- To provide a detailed description of various ways of organizing memory hardware.
- To discuss various memory-management techniques, including paging and segmentation.

Topics Included

- Introduction
- Basic Memory Management
- Multiprogramming with variable partitions – swapping
- Virtual Memory
- Paging
- Page Replacement Algorithms
- Segmentation

Introduction

- Memory is one of the most important resources of the computer system that is used to store data and programs temporarily.
- Program must be brought into memory and placed within a process for it to be run.
- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Operating system coordinate how these memories are used.

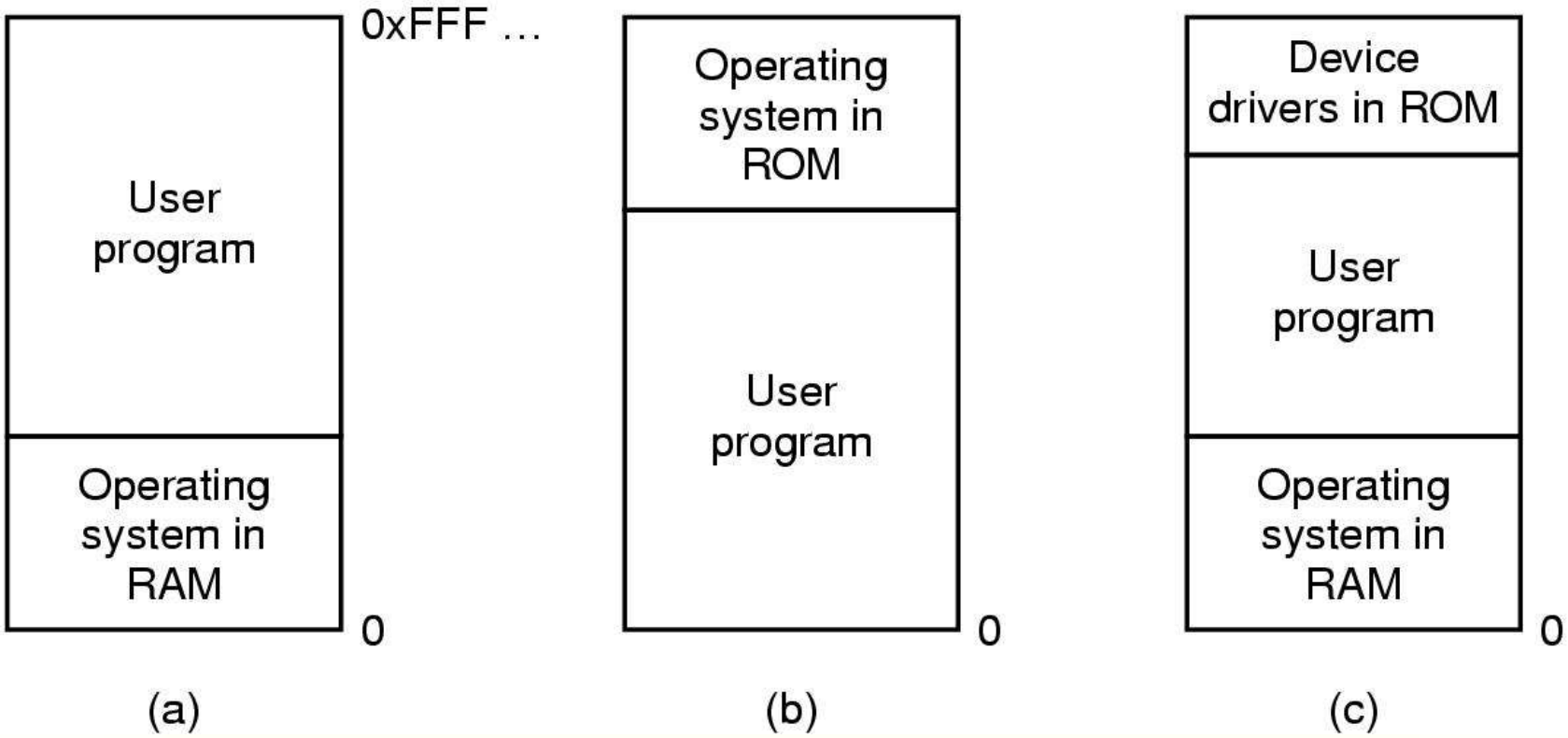
Introduction(cont'd)

- Part of the operating system that manages memory is called the memory manager (MM).
- The main functions of the MM are the following:
  - Keeping track of which part of memory are in use and which parts are free
  - Allocating and de-allocating memory to processes
  - Managing swapping between memory and disk when memory is not big enough to hold all the processes
- A good memory manager has the following attributes
  - It allows all processes to run
  - Its memory (space used for the management activity) overhead must be reasonable
  - Its time overhead (time required for the management activity) is reasonable.

Basic Memory Management

- Memory management systems can be divided into two basic classes:
  - those that move processes back and forth between main memory and disk during execution (swapping and paging),
  - and those that do not:
- Mono-programming – Only one process will be in the memory in addition to the operating system
- Multiprogramming - It allows multiple programs to run in parallel. Divides the memory among concurrently running processes and the operating system.

Monoprogramming without Swapping or Paging



Three simple ways of organizing memory

(a) formerly used on mainframes and minicomputers but is rarely used any more

(b) used on embedded systems

(c) used by early personal computers

Memory Partitioning

- In order to run multiple programs in parallel a memory should be partitioned.
- There are two partitioning options – fixed partitioning and dynamic partitioning
  - Fixed partitioning – partitioning is done before the processes comes to memory
  - Dynamic partitioning – partitioning is done when processes request memory space.

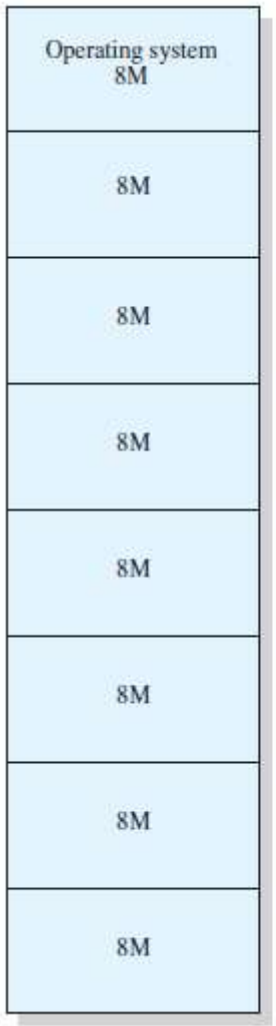
1. Fixed partitioning

- It is the simplest and oldest partitioning technique
- Variations of fixed partitioning
  - Equal sized fixed partitions
  - Unequal sized fixed partitions

Equal sized fixed partitions

- The memory is partitioned into equal-sized partitions
- Any process whose size is less than or equal to the partition size can be loaded into any available partition
- If all partitions are occupied with processes that are not ready to run, then one of these processes is swapped-out and a new process is loaded or a suspended process is swapped in and loaded
- Advantage: It is simple to implement and requires minimal management overhead
- Problems:
  - A program may be too big to fit into a partition
  - Inefficient use of memory due to internal fragmentation.
    - Can happen if there is hole of size 15,000 bytes and a process needs 14,900 bytes; Keeping a hole of size 100 bytes is not worth the effort so the process is allocated 15,000 bytes.
    - The size difference of 100 bytes is memory internal to a partition, but not being used

Example of Fixed Partitioning



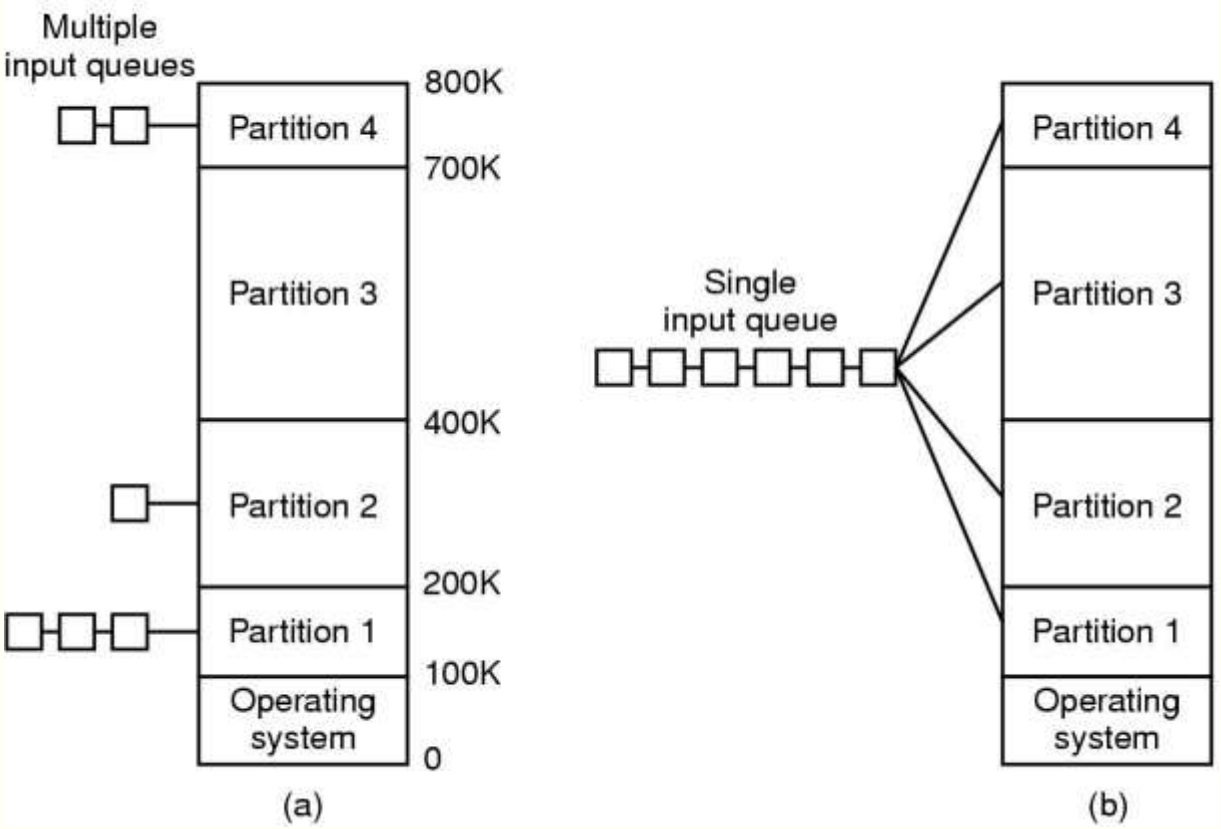
(a) Equal-size partitions

(b) Unequal-size partitions

Unequal sized fixed partitioning

- Memory is partitioned into unequal-sized fixed partitions
- In this case there are two memory allocation schemes:
  - Using multiple queues
  - Using a single queue
- **Using multiple queues**
  - Each partition has its own queue
  - A process is assigned to the smallest partition within which it will fit and it can only be loaded into the partition it belongs
  - **Advantage**: Minimize internal fragmentation
  - **Disadvantage** processes are waiting in some other queues: Some partitions may remain unused.

Unequal sized fixed partitioning(cont'd)



- separate input queues for each partition
  - (a) Multiple input queue
  - (b) Single input queue

Unequal sized fixed partitioning(cont'd)

- **Using single queue**
  - The smallest available partition that can hold the process is selected
  - Whenever a partition becomes free, the process closest to the front of the queue that fits in it could be loaded into the empty partition.
  - **Advantage**: Partitions are used all the time
  - **Disadvantage**: When best fit is searched, it will not give small jobs best service, as they are interactive. A possible remedy for this problem is not to allow small processes not to be skipped more than k – times.
- **Summary**
  - Advantage of fixed partitioning
    - It is simple to implement and requires minimal management overhead
  - Disadvantages of fixed partitioning
    - Inefficient use of memory due to internal fragmentation
    - Limits number of active processes
    - A program may be too big to fit in any of the partitions.

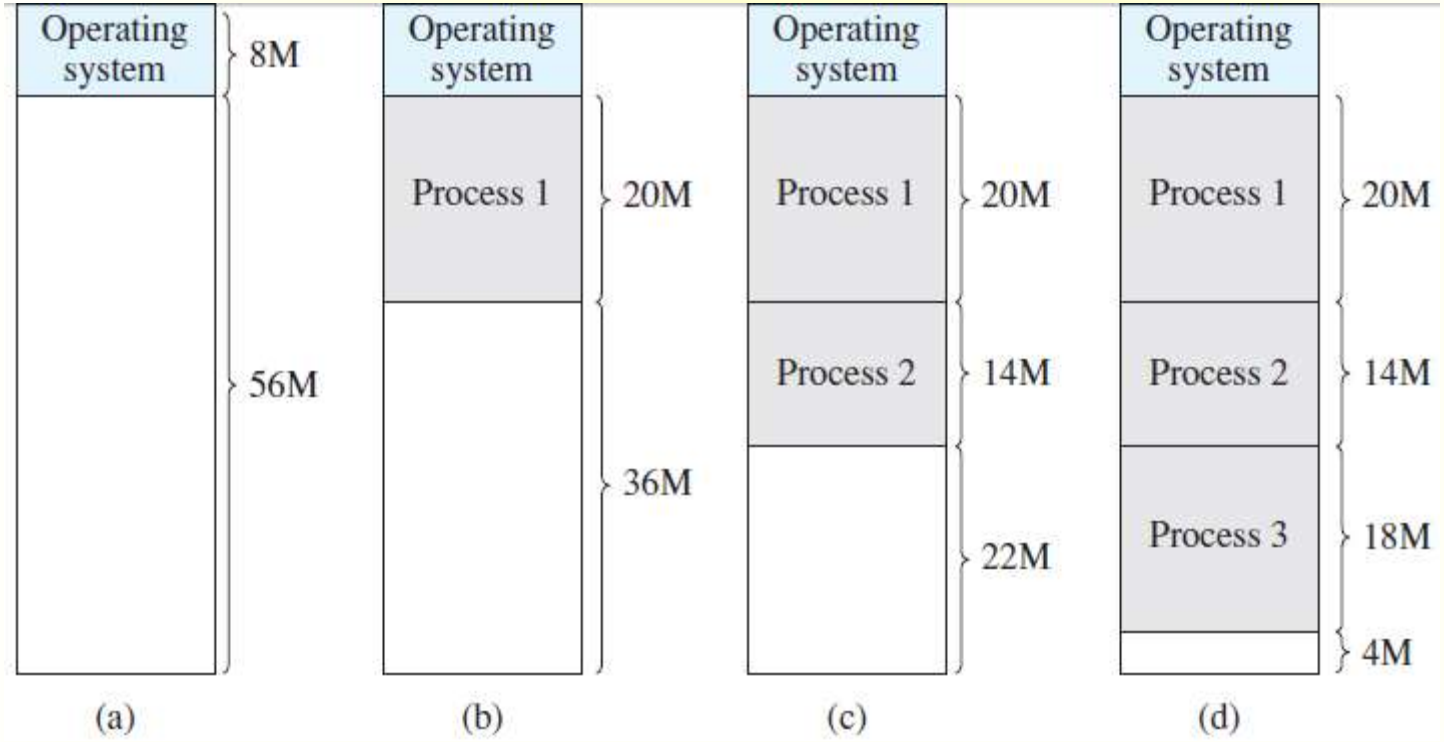
2. Dynamic partitioning

- Partitions are created dynamically during the allocation of memory.
- The size and number of partitions vary throughout of the operation of the system.
- Processes will be allocated exactly as much memory they require.
- If there is not enough main memory to hold all the currently active processes, so excess processes must be kept on disk and brought in to run dynamically.

Dynamic partitioning(cont'd)

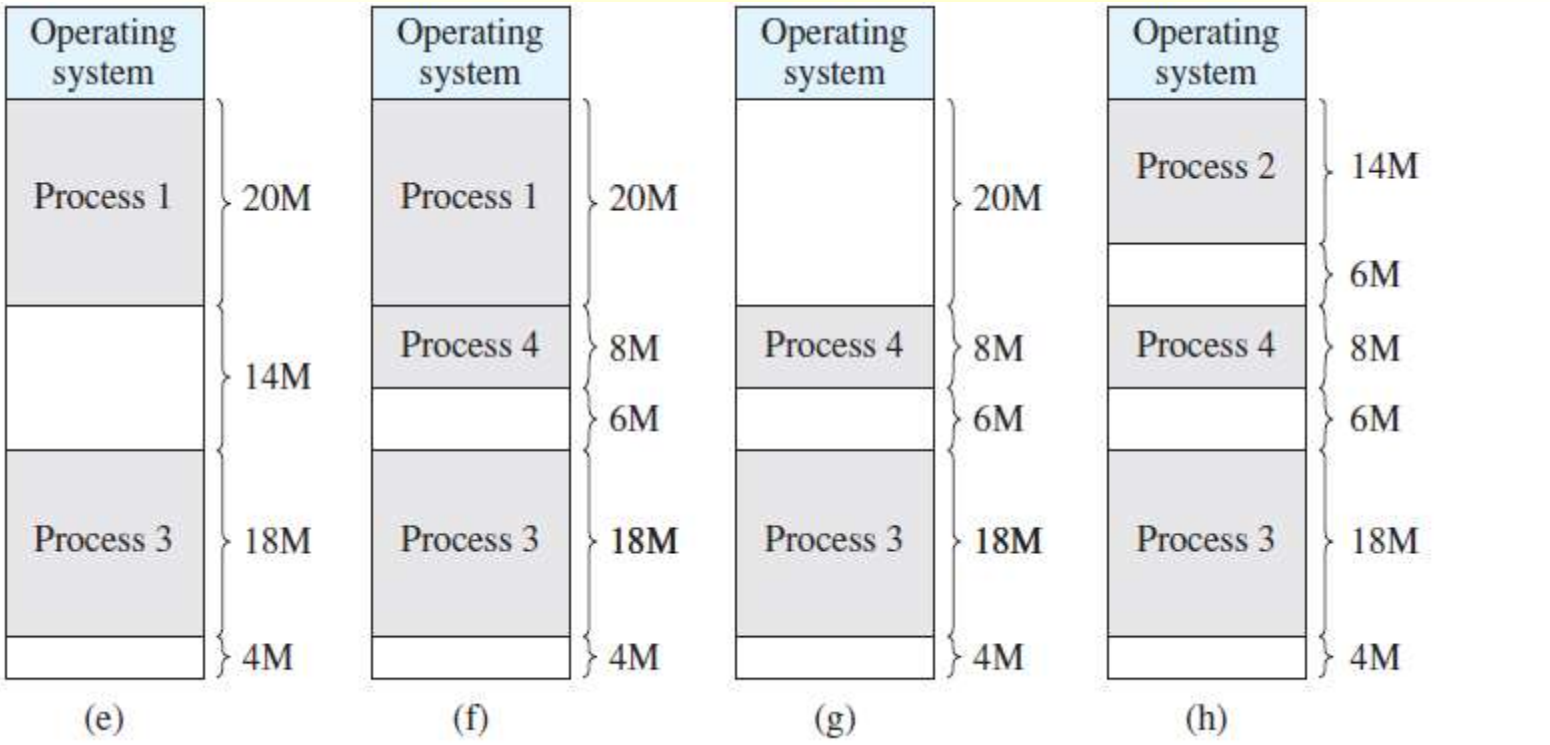
- Dynamic partitioning leads to a situation in which there are a lot of small useless holes in memory. This phenomenon is called external fragmentation.
- Solution to external fragmentation is compaction – combining the multiple useless holes in to one big hole.
  - Problem: it requires a lot of CPU time. For instance, on a 1-GB machine that can copy at a rate of 2 GB/sec (0.5 nsec/byte) it takes about 0.5 sec to compact all of memory.

Dynamic partitioning(cont'd)



The Effect of Dynamic Partitioning

Dynamic partitioning(cont'd)



The Effect of Dynamic Partitioning

Dynamic Memory Summary

- **Advantage**
  - Dynamic memory allocation provides a better memory utilization
- **Disadvantage**
  - Management is more complex
  - Includes memory compaction overhead

Relocation and Protection

- Multiprogramming introduces two essential problems that must be solved – **protection and relocation**
- **Protection**
  - Programs in other processes should not be able to reference memory locations in a process for reading or writing purposes without permission.
  - A user program shouldn't be able to address a memory area which is not in its partition.
    - A user process must not access any portion of the OS.
    - Usually a program in one process can not branch to an instruction in another process.
    - Without permission, a program in one process cannot access the data area of another process.

Relocation and Protection(cont'd)

- **Relocation**
  - Location of a program in main memory is unpredictable due to loading, swapping and compaction.
  - To solve this problem, a distinction is made among several types of addresses.
  - There are two types of addresses:
    - Logical address
      - It is a reference to a memory location independent of the current assignment of data to memory.
    - Physical/absolute address
      - An actual locator in the main memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.
- The memory-mapping hardware converts logical addresses into physical addresses.

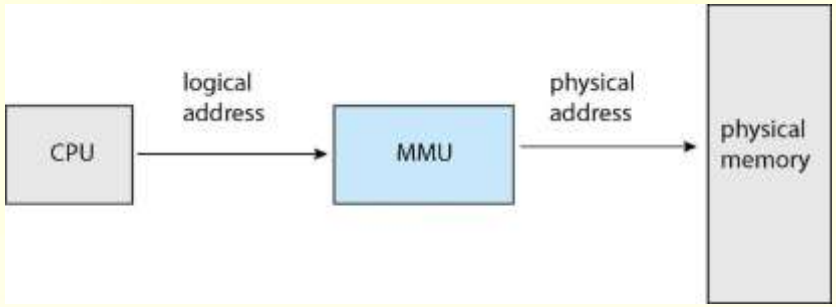
Logical vs. Physical Address Space

- The concept of a **logical address space** that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU.
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are:
  - The same in compile-time and load-time address-binding schemes;
  - They differ in execution-time address-binding scheme. In that case the logical address is referred to as **virtual address.**

  We use Logical address and virtual address interchangeably
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses corresponding to a given logical address space.

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual addresses to physical address



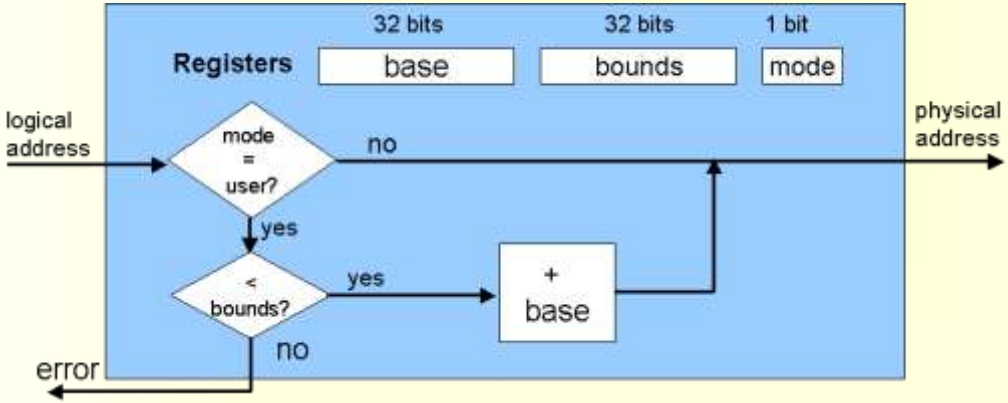
Relocation techniques

- There are two possible solutions for relocation problems – *static relocation* and *dynamic relocation.*
- **Static Relocation**
  - When a process is first loaded, all relative memory references are replace by an absolute addresses based on the base address of the loaded process
  - It can be applied for processes which are always assigned to the same partition, for instance in the case of unequal size fixed partitions scheme with multiple process queue.
  - It doesn't solve the protection problem. A malicious program can always construct a new instruction and jump to it.

Relocation techniques(cont'd)

- **Dynamic Relocation**
  - When a process is assigned for running, a special processor register (base register) is loaded with the starting address of the program and a limit register is loaded with ending address of the partition
  - Limit and base registers are protected by the hardware not to be modified by user programs
  - Hardware mechanisms are used for translating relative addresses to physical addresses at the time of execution of the instructions that contains the reference.

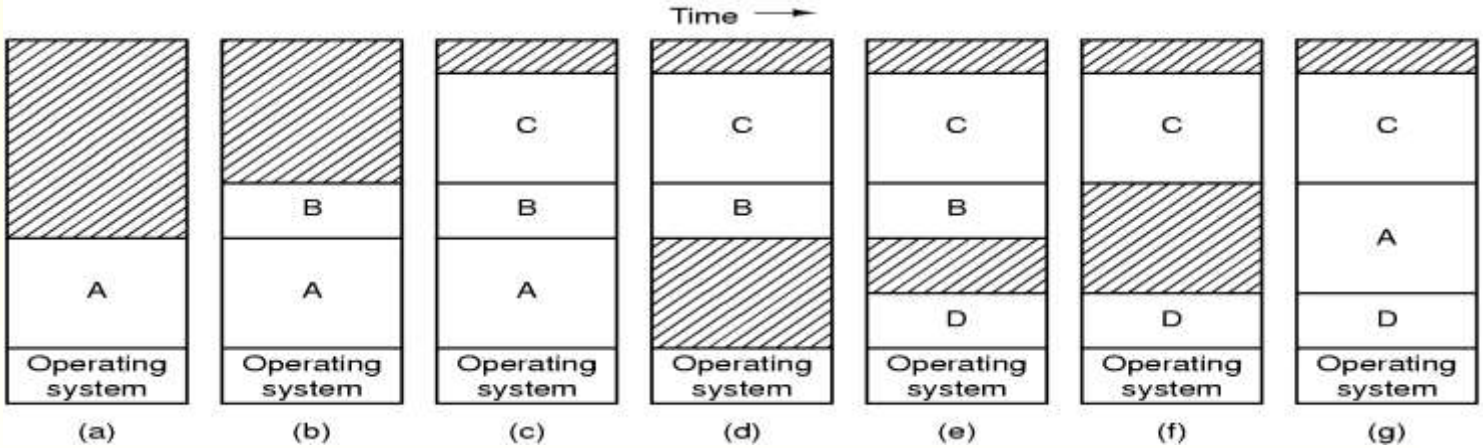
Relocation techniques(cont'd)



- The value in the base register is added to the relative address to produce an absolute address
- The absolute address is compared to the valued in the limit/bound register
- If the address is within bounds, the instruction execution can proceed, otherwise an interrupt is generated to the OS
- Dynamic relocation provides solution to both relocation and protection problems.

Swapping

- A process can be **swapped** temporarily out of memory to a backing store and then brought back into memory for continued execution
  - Total physical memory space of all processes can exceed the real physical memory of the system.
- Two general approaches to memory management can be used
  - **Swapping**: bringing in each process in its entirety, running it for a while, then putting it back on the disk.
  - **virtual memory** allows programs to run even when they are only partially in main memory.

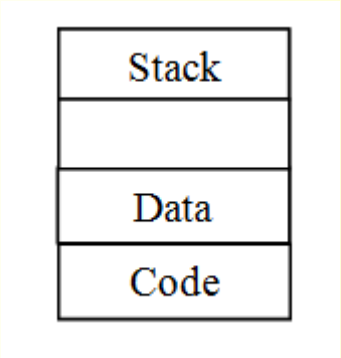


- Memory allocation changes as
  - processes come into memory
  - leave memory
- Shaded regions are unused memory

Swapping(cont'd)

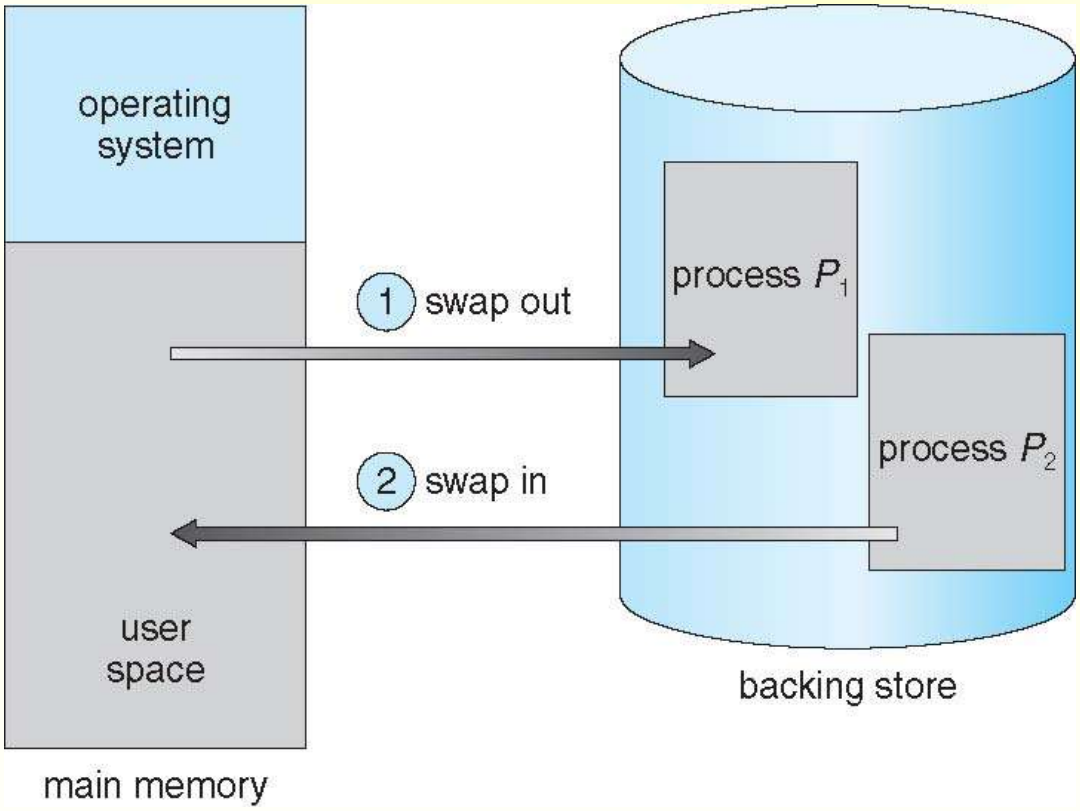
- A process has three major sections - *stack*, *data* and *code*
- With fixed partitions, the space given to process is usually larger than it asked for and hence the process can expand if it needs.
- With dynamic partitions since a process is allocated exactly as much memory it requires, it creates an overhead of moving and swapping processes whenever the process grows dynamically.
- It is expected that most processes will grow as they run, so it is a good idea to allocate a little extra memory whenever a process is loaded into the memory.
- As shown in the following diagram the extra space is located between the stack and data segments because the code segment doesn't grow.

Swapping(cont'd)



- If a process runs out of the extra space allocated to it, either
  - It will have to be moved to a hole with enough space or
  - Swapped out of memory until a large enough hole can be created or
  - Kill the process

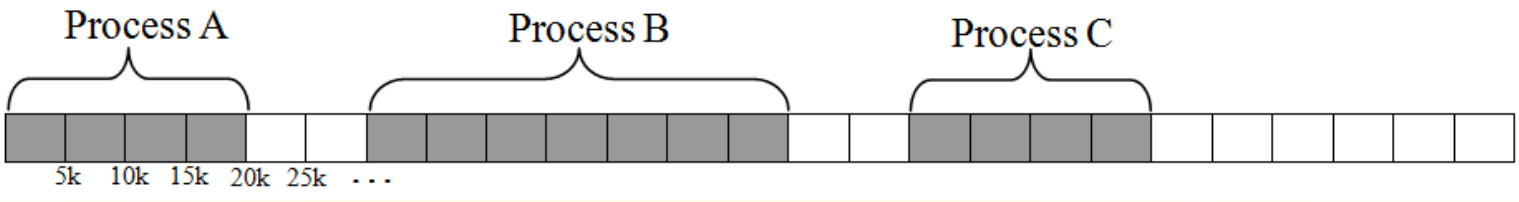
Schematic View of Swapping



Memory Usage Management

- When memory is assigned dynamically, the operating system must manage it.
- There are two ways to keep track of memory usage – bitmap and linked list
- **Bitmaps**
  - The memory is divided into fixed size allocation units.
  - Each allocation unit is represented with a bit in the bit map. If the bit is 0, it means the allocation unit is free and if it is 1, it means it is occupied.
  - **Example**: Look at the following portion of a memory. Let the size of the allocation units is 5k as shown in the diagram.

Memory Management with Bitmaps



- The corresponding bitmap for the above memory can be as follows:

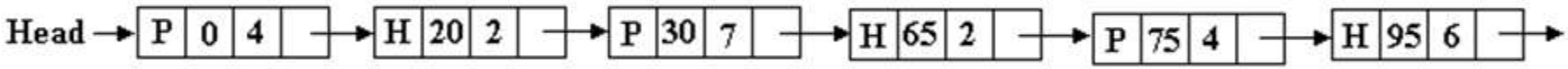
11110011
11111001
11100000
...

Memory Management with Bitmaps(cont'd)

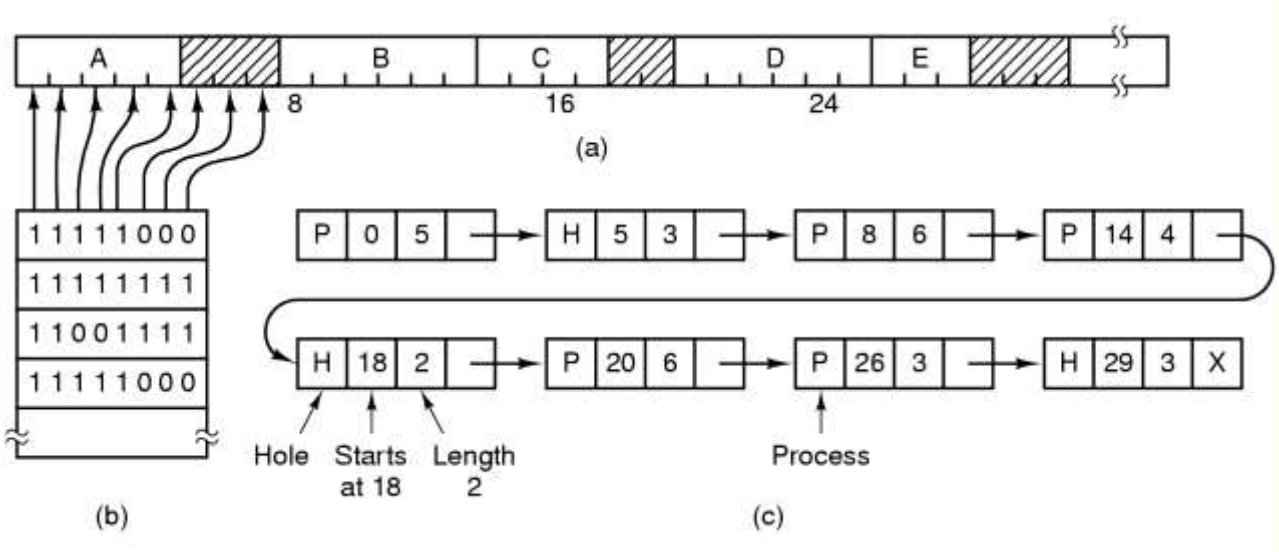
- The size of the allocation unit is an important design issue.
- The smaller the allocation unit, the larger the bitmap size, which will result to memory and CPU overheads. Searching will take time.
- **E.g.** Assume an allocation unit of 4bytes and assume the computer has 800KB of memory
  - **Number of bits required =** 800KB/4B = 200Kbits
  - **Size of bitmap =** 200Kbits/8 = 25KB
  - **Percentage of the memory used for the bitmap = (**25/800)*100% ≈ 3%
- What if the allocation unit is large?
  - The larger the allocation unit, the larger the internal fragmentation.

Memory Management with Linked Lists

- Maintain a linked list of allocated and free memory segments
- Each segment is either a process(P) or a hole(H) between two processes and contains a number of allocation units
- Each entry in the list consists of
  - Segment type: P/H (1bit)
  - The address at which it starts
  - The length of the segment
  - A pointer to the next entry
- **Example 1:** The memory in the bitmap example can be represented using linked list as follows:



Example 2: bitmap and linked list

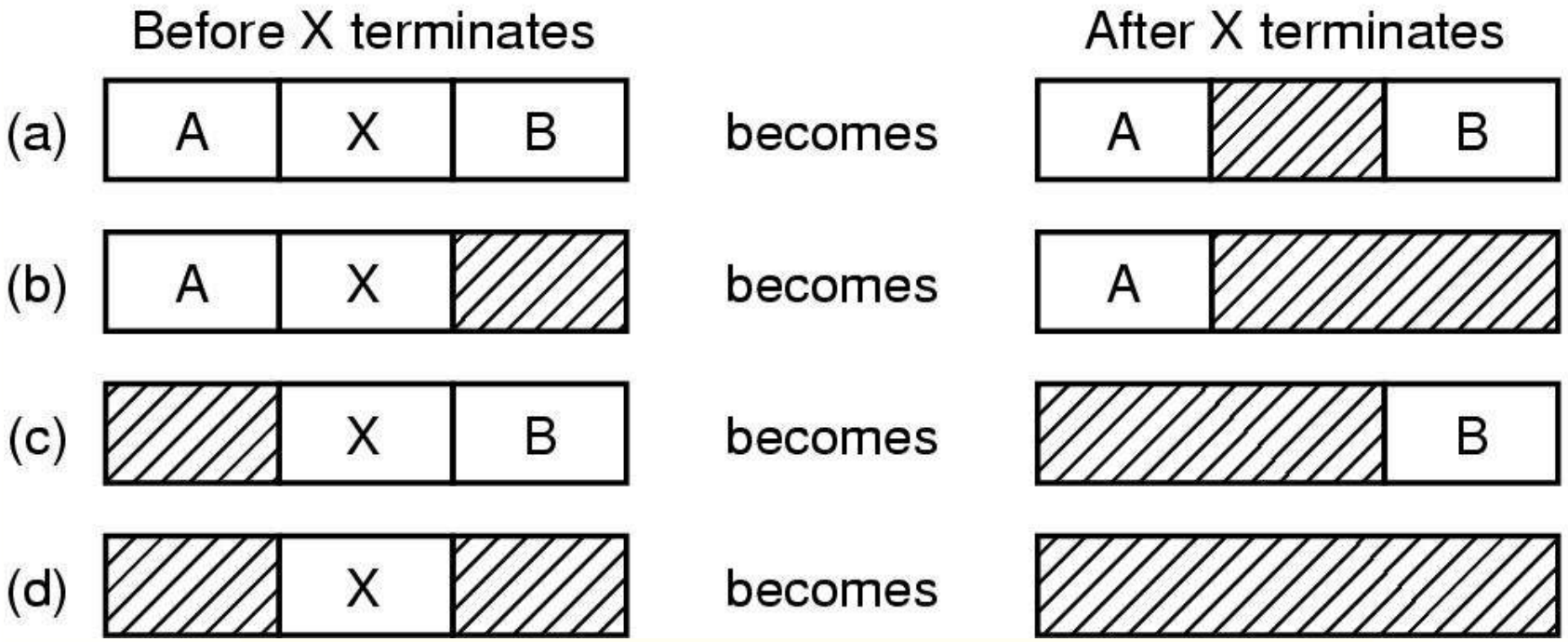


- (a) Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- (b) Corresponding bit map
- (C) Same information as a list

Memory Management with Linked Lists(cont'd)

- The list is kept sorted by address. Sorting this way has the advantage of easily updating when processes terminates or is swapped out.
- A terminating process normally has two neighbors, except when it is at the very beginning or end of the list.
- The neighbors can be either process or holes, leading to four combinations as shown on the next slide:

Memory Management with Linked Lists(cont'd)



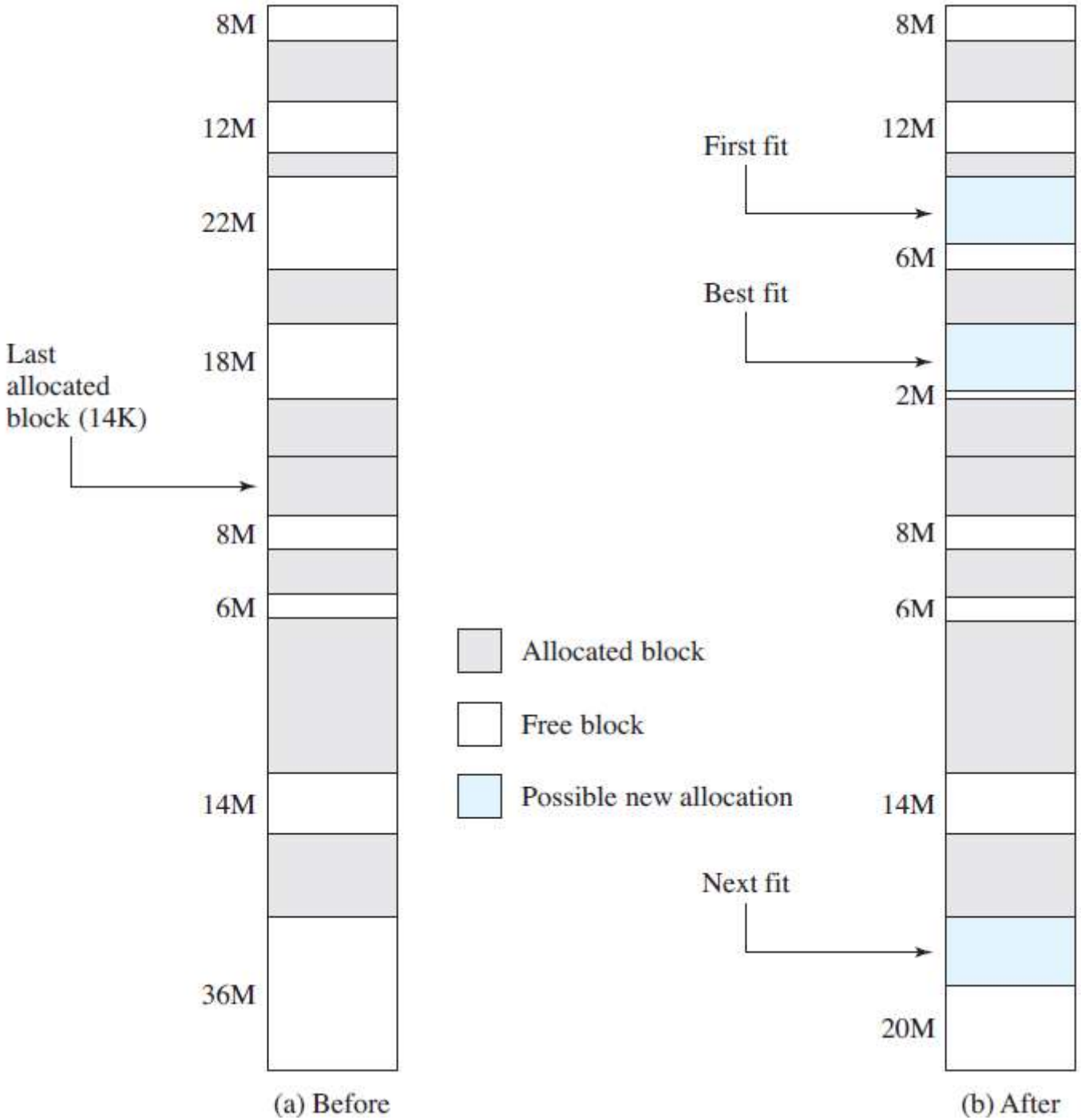
- When a process and holes are kept on a list sorted by address, several algorithms can be used to allocate memory for a newly created process.

Memory Allocation Algorithms

- **Memory Allocation algorithms (Placement algorithms)**
- When a new process is created or swapped in, the OS must decide which free block to allocate.
- The following are some of the placement algorithms
- **First-Fit**
  - It scans the memory from the beginning and chooses the first available block that is large enough
  - It is the simplest and fastest algorithm
  - **Disadvantage**: Searching from the beginning always may result in slow performance

Memory Allocation Algorithms(cont'd)

- **Next Fit**
  - It starts scanning the memory from the location of the last placement and chooses the next available block that is large enough
  - **Disadvantage**: It may cause the last section of memory to be quickly fragmented
- **Best Fit**
  - It searches the entire list of available holes and chooses the lock that is closest in size to the request
  - **Disadvantage**: External fragmentation and slower because it scans the whole list .
- **Example:** Memory Configuration before and after Allocation of 16-Mbyte Block


8M
12M
22M
18M
Last allocated block (14K)
8M
6M
14M
36M
(a) Before
Allocated block
Free block
Possible new allocation
8M
12M
First fit
6M
Best fit
2M
8M
6M
14M
Next fit
20M
(b) After

Memory Allocation Algorithms(cont'd)

- **Worst Fit**
  - It searches for the largest available hole, so that the hole broken off will be big enough to be useful
  - **Disadvantage**: It is as slow as best fit
- **Quick Fit**
  - It maintains separate list for some of the more common size requests
  - **Advantage**: Finding a hole of the required size is extremely fast
  - **Disadvantage**: When a process terminates or is swapped out, finding its neighbors to see if a merge is possible is expensive.

Virtual Memory

- A process may be larger than the available main memory
- In the early days **overlays** were used to solve this problem
  - The programmer will divide the program into modules and the main program is responsible for switching the modules in and out of memory as needed.
  - Although the actual work of swapping overlays in and out was done by the system, the decision of how to split the program into pieces had to be done by the programmer.
  - **Drawback**: The programmer must know how much memory is available and it wastes the programmer time.
- In virtual memory the OS keeps those parts of the program currently in use in main memory and the rest on the disk.

Virtual Memory(cont'd)

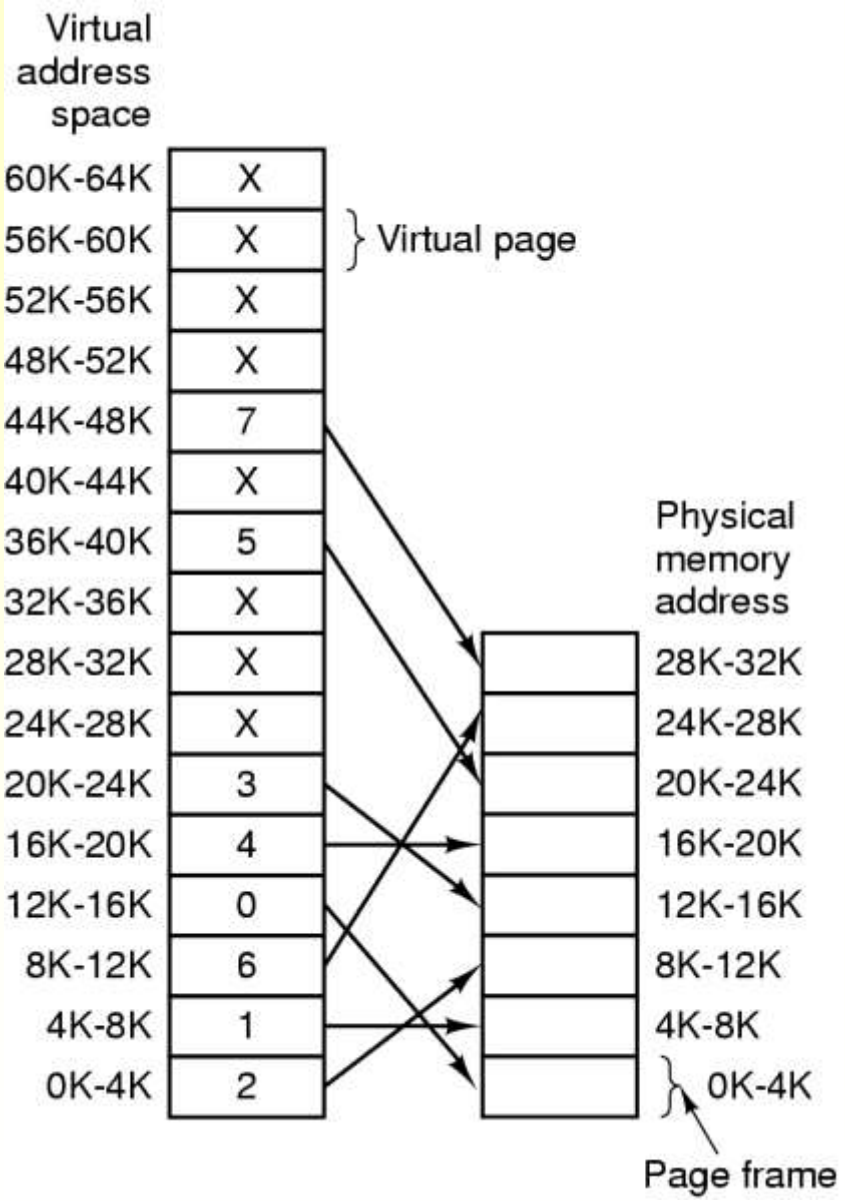
- Piece of programs are swapped between disk and memory as needed.
- **Example**, a 512-MB program can run on a 256-MB machine by carefully choosing which 256 MB to keep in memory at each instant, with pieces of the program being swapped between disk and memory as needed.
- Program generated addresses are called virtual addresses and the set of such addresses form the **virtual address space**.
- The virtual address will go to the **Memory Management Unit (MMU)** that maps the virtual addresses into physical addresses.

Paging

- There are two virtual memory implementation techniques
  - Paging
  - Segmentation
- **Paging**
  - The virtual address space is divided into units called **pages.** The corresponding units in the physical memory are called **page frames**.
    - Size of a frame is power of 2 between 512 bytes and 16 Mbytes
    - Pages and page frames should have the same size
    - Transfer between memory and disks are always in units of pages
    - The operation of dividing a process into pages is done by the compiler or MM
  - **Example**: Assume a 64KB virtual address and each process is divided into 4K pages and the physical memory size is 32KB.

Virtual Memory

The relation between virtual addresses and physical memory addresses given by page table



Paging(cont'd)

- **Examples:**

1. When the program tries to access address 0 using the instruction - MOVE REG, 0
   - The MMU sees that this virtual address falls in page 0 (0-4095), which according to its mapping is page frame 2 (8192 to 12287). It thus transforms the address to 8192 and outputs address 8192 onto the bus.

2. MOVE REG, 8196 is effectively transformed in to MOVE REG, 24580

3. MOVE REG, 4250, ?

In to: 4250

4. Move REG, 20580, ?

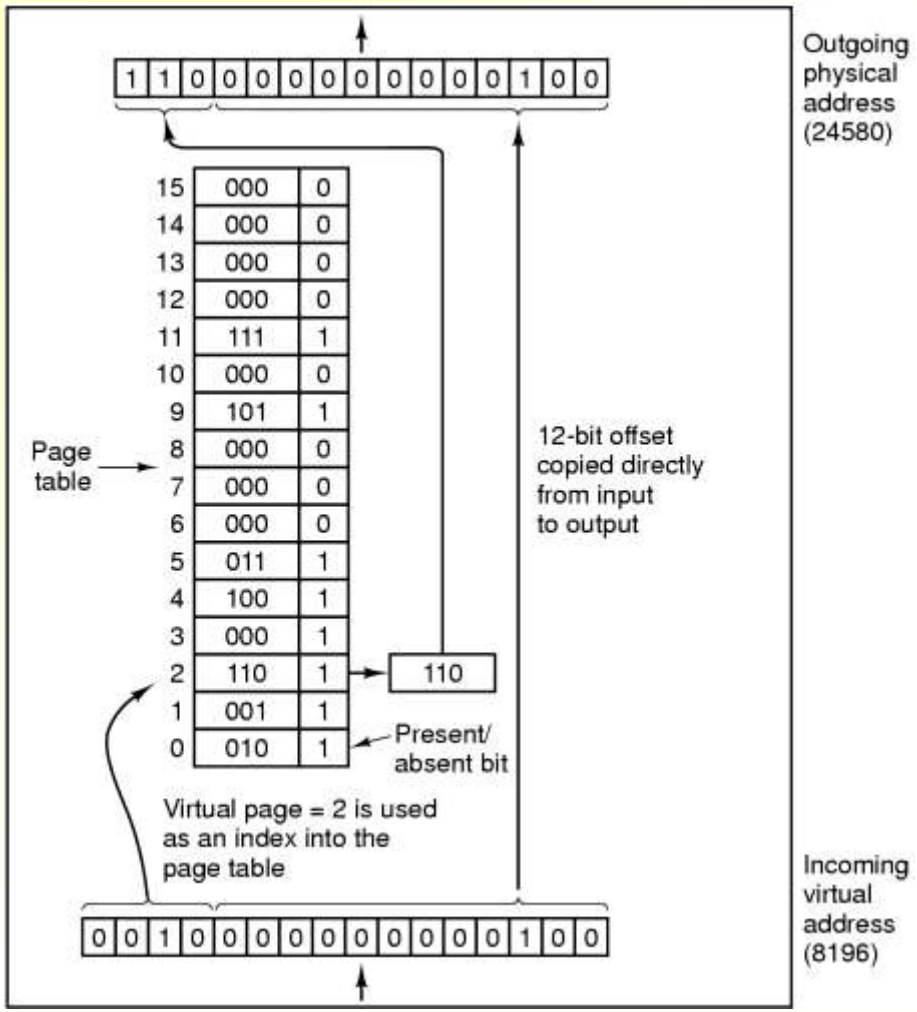
In to: 12388

- Address

0. 0-4095
1. 4096- 8191
2. 8192 – 12287
3. 12288 – 16383
4. 16384 – 20479
5. 20480 – 24575
6. 24576 – 28671
7. 28672 - 32767
8. 32767-36863
9. 36864- 40959
10. 40960– 45055
11. 45056– 49151
12. 49152– 53247
13. 53248– 57343
14. 57344– 61439
15. 61440- 65535

Paging(cont'd)

- **Example**: Address Bits = 16, Page size = 4K, Logical address = 8196. The following figure shows the internal operation of the MMU with 16 4K pages. split into a 4-bit page number and a 12-bit offset.



Internal operation of MMU with 16 4 KB pages

Paging(cont'd)

- **Page Tables**
  - The OS maintains page table for mapping virtual addresses to physical addresses.
  - Page tables are stored in memory.
  - The OS also maintains all free-frames, list of all frames in main memory that are currently unoccupied.
  - Each page table entry contains: **present-bit**, **modified-bit**, and **frame number.**
    - **Present-bit**: Whether the page is present in main memory or not
    - **Modified-bit:** whether the contents of the corresponding page have been altered since the page was last loaded into main memory. If it is not altered, then it is not necessary to write the page out when it is to replace the page in the frame it occupies
    - **Frame number:** Indicates the corresponding page frame number in memory.

Paging(cont'd)

- **Page Size**
- There are two issues:
  - The smaller the page size, the less number of internal fragmentation which optimizes the use of main memory.
  - The smaller the page size, the greater the number of pages and thus the larger the page table.

Paging(cont'd)

- If an instruction in a process tries to access an address in unmapped page
  - The MMU causes the CPU to generate an interrupt, called **page fault**
  - The OS puts the interrupted process in a blocking state and takes control
  - The OS chooses a page(page frame) to remove from memory and updates its contents back to the disk if it is modified since its last load
  - The OS issues a disk I/O read request to fetch the page just referenced to the page frame just freed and the OS can dispatch another process to run while the disk I/O is performed.

Page Replacement Algorithms

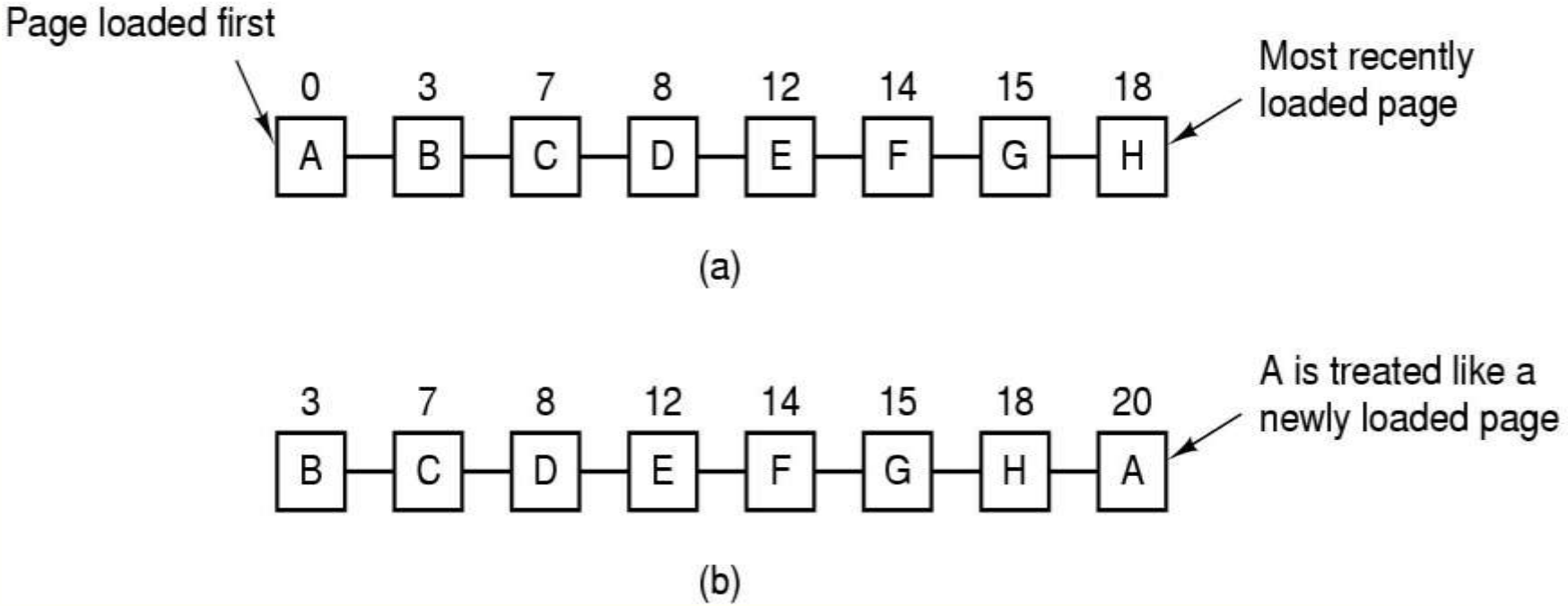
- **Page Replacement Algorithms (PRA)**
  - When a page fault occurs, the OS has to choose a page to remove from memory to make room for the page that has to be brought in.
- **Optimal PRA**
  - It says replace the page that will not be referenced soon
  - Drawback – impossible to know whether the page will be referenced or not in the future (its unrealizable)
- **First In First Out PRA**
  - Remove the oldest page
  - The FIFO page-replacement algorithm is easy to understand and program.
  - However, its performance is not always good.
    - page in memory the longest may be often used

Page Replacement Algorithms(cont'd)

- FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be used.
- **Least Recently Used (LRU) PRA**
  - Remove pages that have less used in the past.
  - We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.

Page Replacement Algorithms(cont'd)

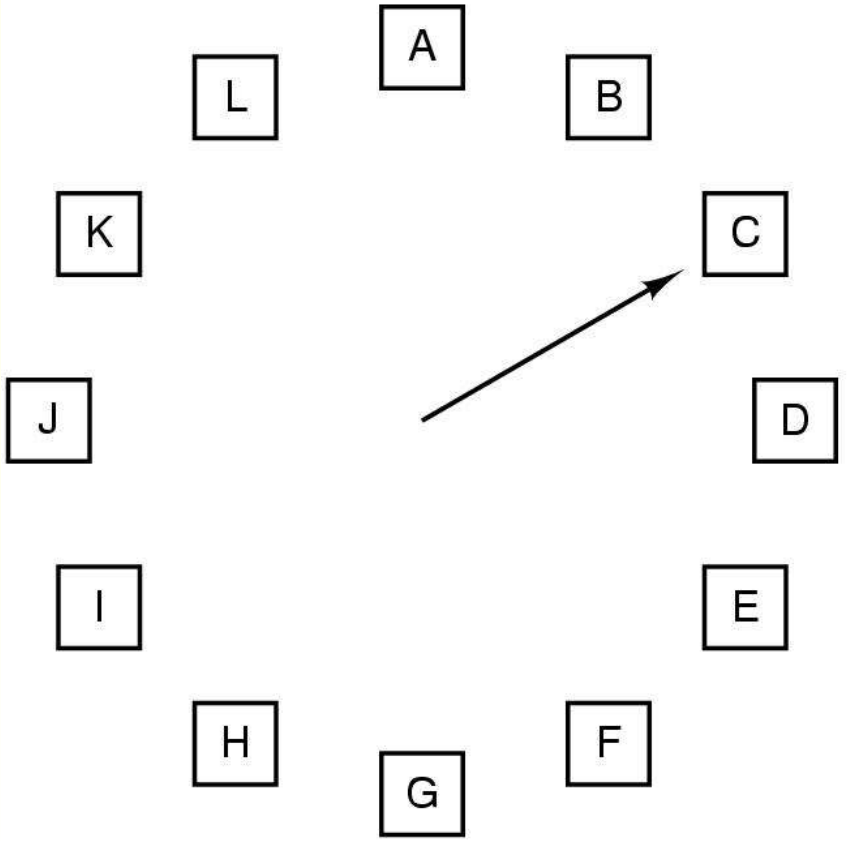
- Second Chance PRA



- Operation of a second chance
  - Pages sorted in FIFO order. Inspect reference bit. If the value is 0, replace this page; but if the reference bit is set to 1, give the page a second chance and move on to select the next FIFO page.
  - Page list if fault occurs at time 20, *A* has *R* bit set

Page Replacement Algorithms(cont'd)

- Clock PRA
  - It differs from second chance only in the implementation.



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

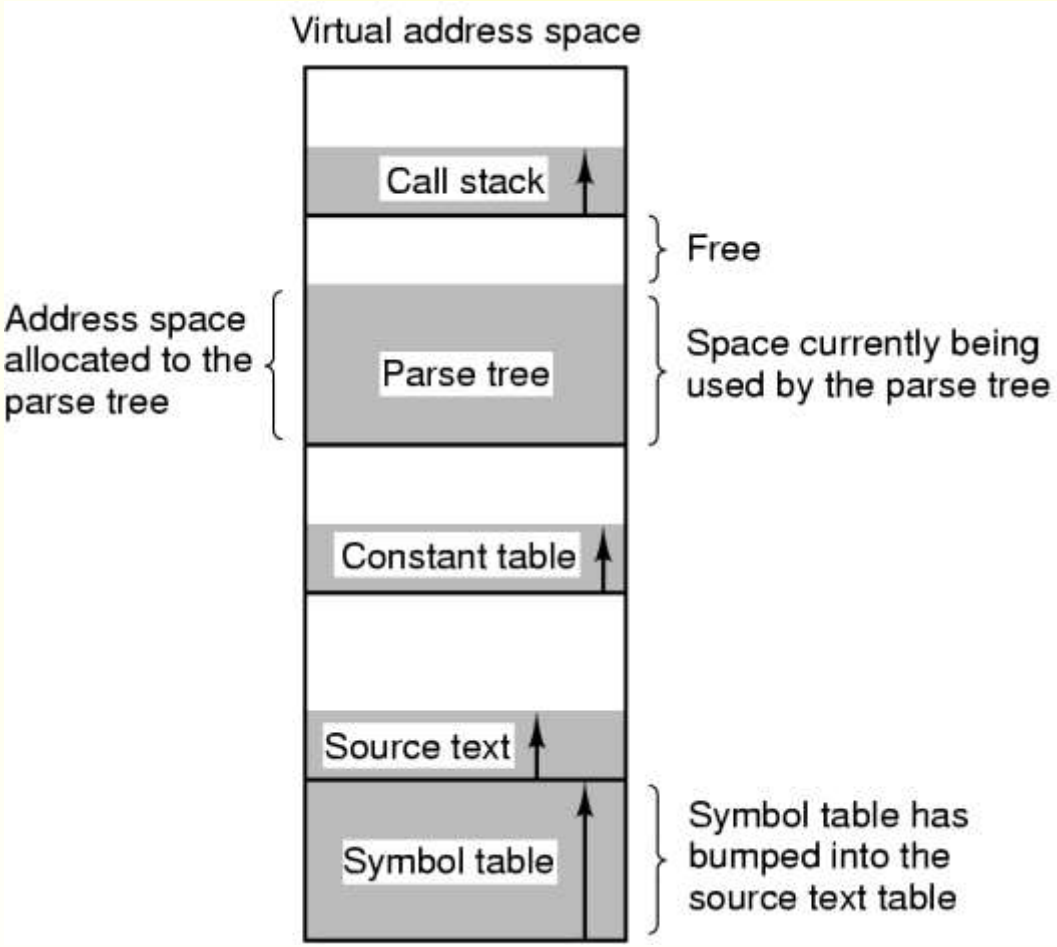
R = 0: Evict the page

R = 1: Clear R and advance hand

Segmentation

- Memory-management scheme that supports user's view of memory
- A program is a collection of segments -- a logical unit such as:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table
  - arrays
- Each segment can to reside in different parts of memory. Way to circumvent the contiguous allocation requirement.
- In a one-dimensional memory, these segments would have to be allocated contiguous chunks of virtual address space

Segmentation(cont'd)

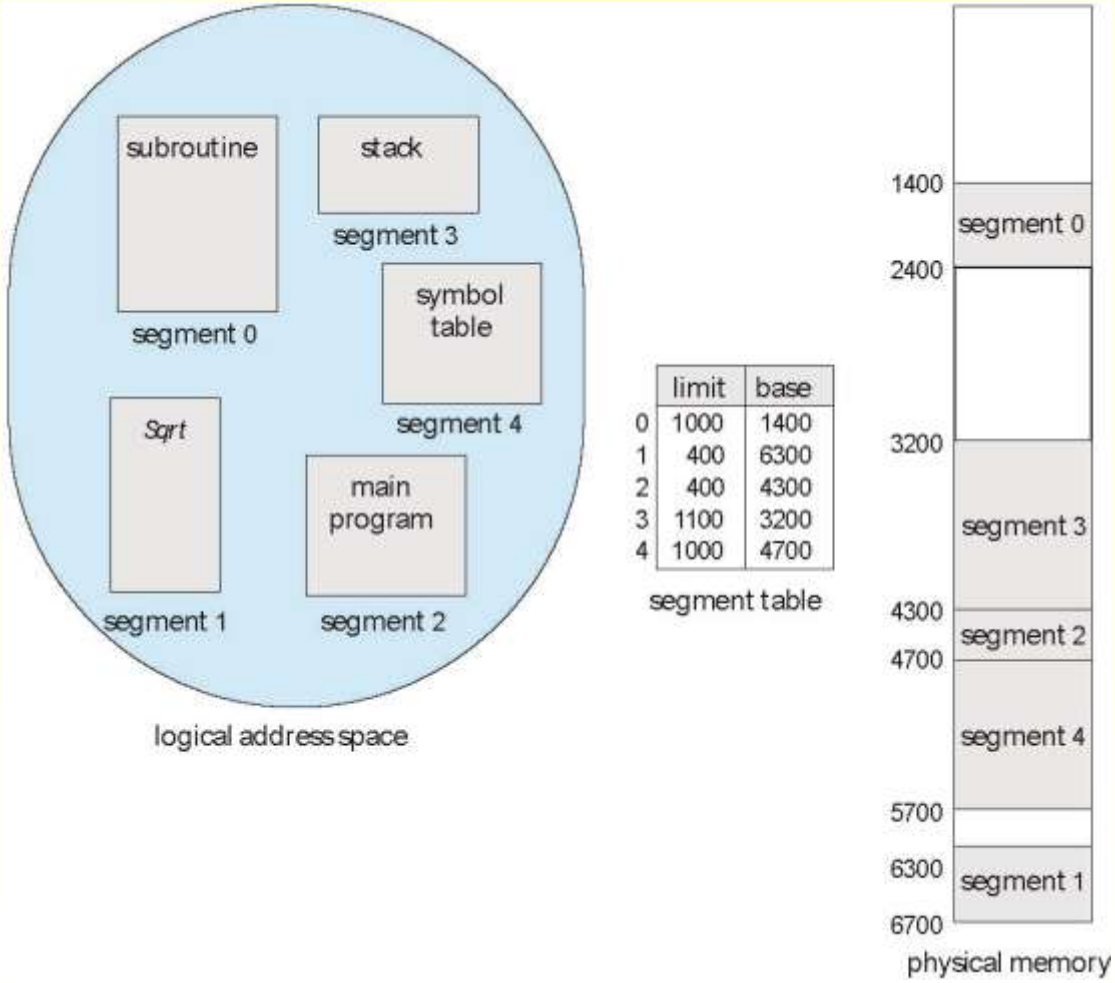


- One-dimensional address space with growing tables
- One table may bump into another

Segmentation(cont'd)

- The virtual space is divided into units called segments.
- Segments can have unequal and dynamic size.
- Each segment has its own virtual address space.
- Each segment consists of a linear sequence of addresses, from 0 to some maximum.
- The OS maintains a segment table for each process and also a free-segment list.
- It simplifies the handling of growing data structures.
- When segmentation is used there is no internal fragmentation.

Example of Segmentation



Paging Vs Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

Comparison of paging and segmentation