Lecture 7

Chapter 7: Memory Management

Topics:

- Memory management

- Memory management requirements

- Partitioning

- Paging

- Segmentation

Memory management

In **a uni-programming system**, the main memory is divided into 2 parts: one part for the OS (kernel) & one part for the process currently being executed.

In a multiprogramming system, the "user" part of the memory must be further subdivided to accommodate multiple processes.

Memory management: the task of subdivision of the memory that is carried out تنفذ dynamically by the OS. It involves swapping blocks of data from/to secondary storage. The OS must cleverly time the swapping to maximize the CPU's efficiency.

WHY?

Effective memory management is vital in a multiprogramming system; if only a few processes are in memory, then for much of the time all of the processes will be waiting for I/O and the processor will be idle.

Thus, memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

Memory Management Key Terms

Term	Description
Frame	**Fixed**-length block of the main memory.
Page	**Fixed**-length block of data.
Segment	**Variable-length** block of data.

Memory Management Requirements

Memory management is intended to satisfy the following requirements:

1. Relocation
2. Protection
3. Sharing
4. Physical organization
5. Logical organization

1. Relocation

- In a multiprogramming system, we would like to be able to swap active processes in and out of the main memory to maximize the processor utilization by providing a large pool of ready processes to execute.
- Once a process is swapped out to the disk,specifying that it must be placed in the same memory region when it is swapped in back would be limiting.
- Instead, we may need to **relocate** the process to a different area of the memory.

Thus, we cannot know ahead of time where a process will be placed, and we must allow for the possibility that the process may be moved in the main memory due to swapping.

The location of a process in the main memory is unpredictable.

The processor and the OS must be able to translate the memory references found in the code of the program into actual physical memory addresses reflecting the current location of the process in the main memory.

2. Protection

Each process should be protected against unwanted interference by other processes, whether accidental or intentional.

Thus, other processes should not be able to reference memory locations in a process for reading or writing purposes without permission.

Mechanisms that support relocation also support the protection requirement.

3. Sharing

Any protection mechanism must have the flexibility to allow several processes to access the same portion of the main memory. For example, if a number of processes are executing the same program, it is advantageous to allow each process to access the same copy of the program rather than having its own separate copy.

Processes that are cooperating on some task may need to share access to the same data structure.

The memory management system must therefore allow controlled access to the shared areas of the memory without compromising essential protection.

The mechanisms used to support relocation support sharing capabilities.

4. Physical Organization

Computer memory is organized into 2 levels; the main memory and the secondary memory.

The main memory provides fast access at relatively high cost. It is volatile; that is, it does not provide permanent storage.

The secondary memory is slower and cheaper than the main memory and is not volatile. Thus, the secondary memory of large capacity can be provided for long-term storage of programs and data, while a smaller main memory holds programs and data currently in use.

In this two-level scheme, the organization of the flow of information between the main and the secondary memory is a major system concern.

The task of moving information between the 2 levels of the memory should be a system responsibility. This task is the essence of memory management.

5. Logical Organization

Almost invariably, the main memory is organized as a linear address space consisting of a sequence of bytes or words. The secondary memory is similarly organized.

Most programs are organized into modules, some of which are un-modifiable (read only, execute only) and some of which contain data that may be modified. The OS and computer HW should effectively deal with user programs and data in the form of modules.

Memory Management

The principal operation of memory management is to bring processes into the main memory for execution by the processor. This involves virtual memory that is based on the use of one or both of 2 basic techniques: segmentation and paging. We will study also partitioning.

In most schemes for memory management, the OS occupies some fixed portion of the main memory and that the rest of the main memory is available for use by multiple processes.

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation.

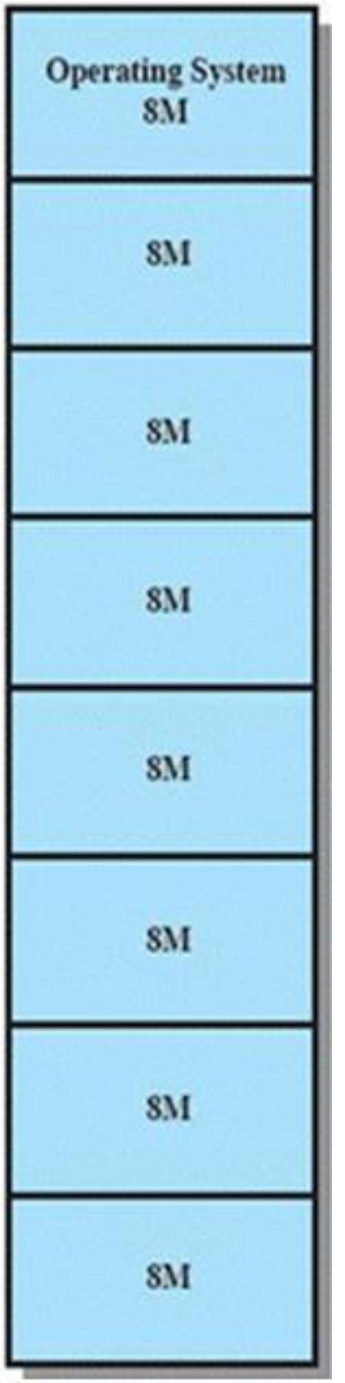
Partitioning

First: Fixed Partitioning

The simplest scheme for managing the available memory is to partition it into regions with fixed boundaries.

A) Equal-size partitioning

- Any process whose size is less than or equal to the partition size can be loaded into **any** available partition.
- If all partitions are full and no process is in the Ready or Running state, the operating system can swap a process out of any of the partitions and load in another process so that there is some work for the processor.



(a) Equal-size partitions

Placement Algorithms

As long as طالما there is **any** available partition, a process can be loaded into that partition.

Because all partitions are of equal size, it does not matter which partition is used.

If all partitions are occupied with processes that are not ready to run, then one of these processes must be swapped out to make room for a new process. Which one to swap out is a scheduling decision.

Disadvantages

1) A program may be too big to fit a partition

- A program needs to be designed with the use of **overlays.**

Overlays

-The program and data are organized in such a way that various modules can be assigned the same region of memory with a main program responsible for switching the modules in and out as needed.

-Keep in the memory only the instructions &data that are needed at any given time.

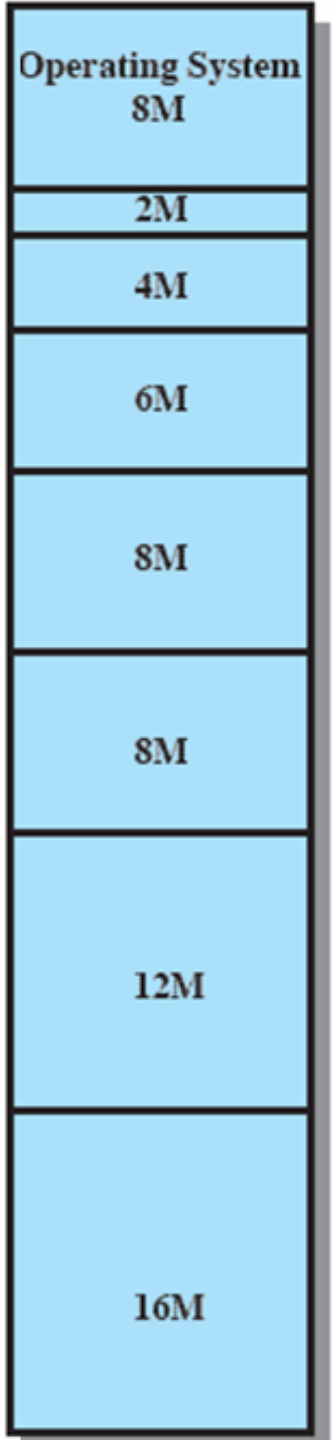
-Implemented by the programmer, no special support needed from the OS. Programming design of overlays structure is complex.

2) Main memory utilization is inefficient.

- Any process, no matter how small it is, occupies an entire partition.
- This phenomenon, in which there is **wasted space internal to a partition** because the block of data loaded is smaller than the partition, is referred to as **internal fragmentation.**

B) Unequal size partitioning

- Using unequal size partitioning helps lessen تقلل the previous problems.
  - processes up to 16M can be accommodated without overlaying.
  - partitions smaller than 8M allow smaller processes to be accommodated with less internal fragmentation.



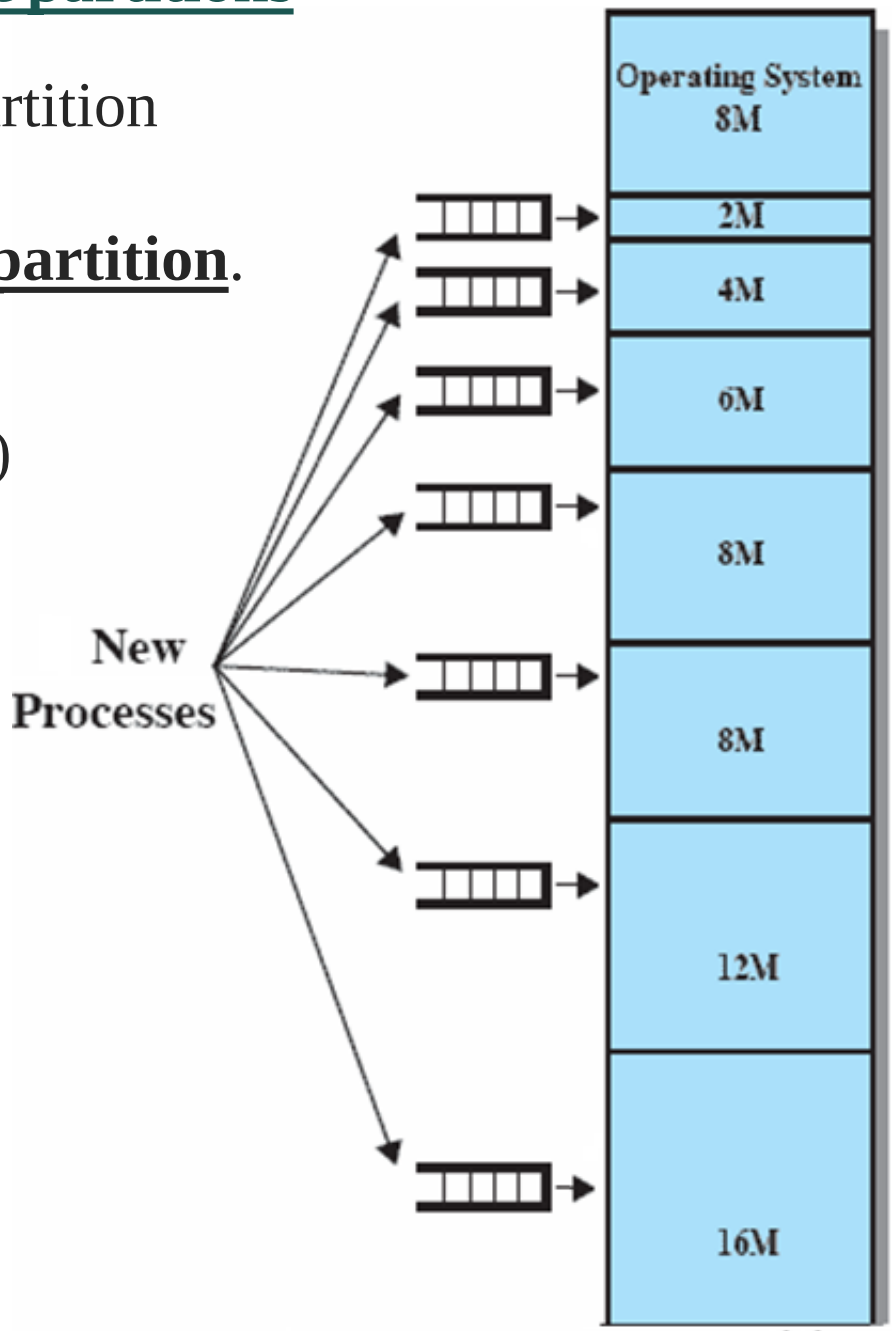
Placement algorithms for unequal-size partitions

1) assign each process to the smallest partition within which it will fit. In this case, **a scheduling queue is needed for each partition**.

Advantage: minimizes wasted memory within a partition (internal fragmentation)

Disadvantage: although this technique seems optimum **from the point of view of an individual partition, it is not optimum from the point of view of the system as a whole.** **EX**: if there are no processes with a size between 12 and 16M at a certain point in time. In this case, the 16M partition will remain unused, even though some smaller processes could have been assigned to it.

كان يمكن تخصيصها لها ولم يحدث



(a) One queue per partition

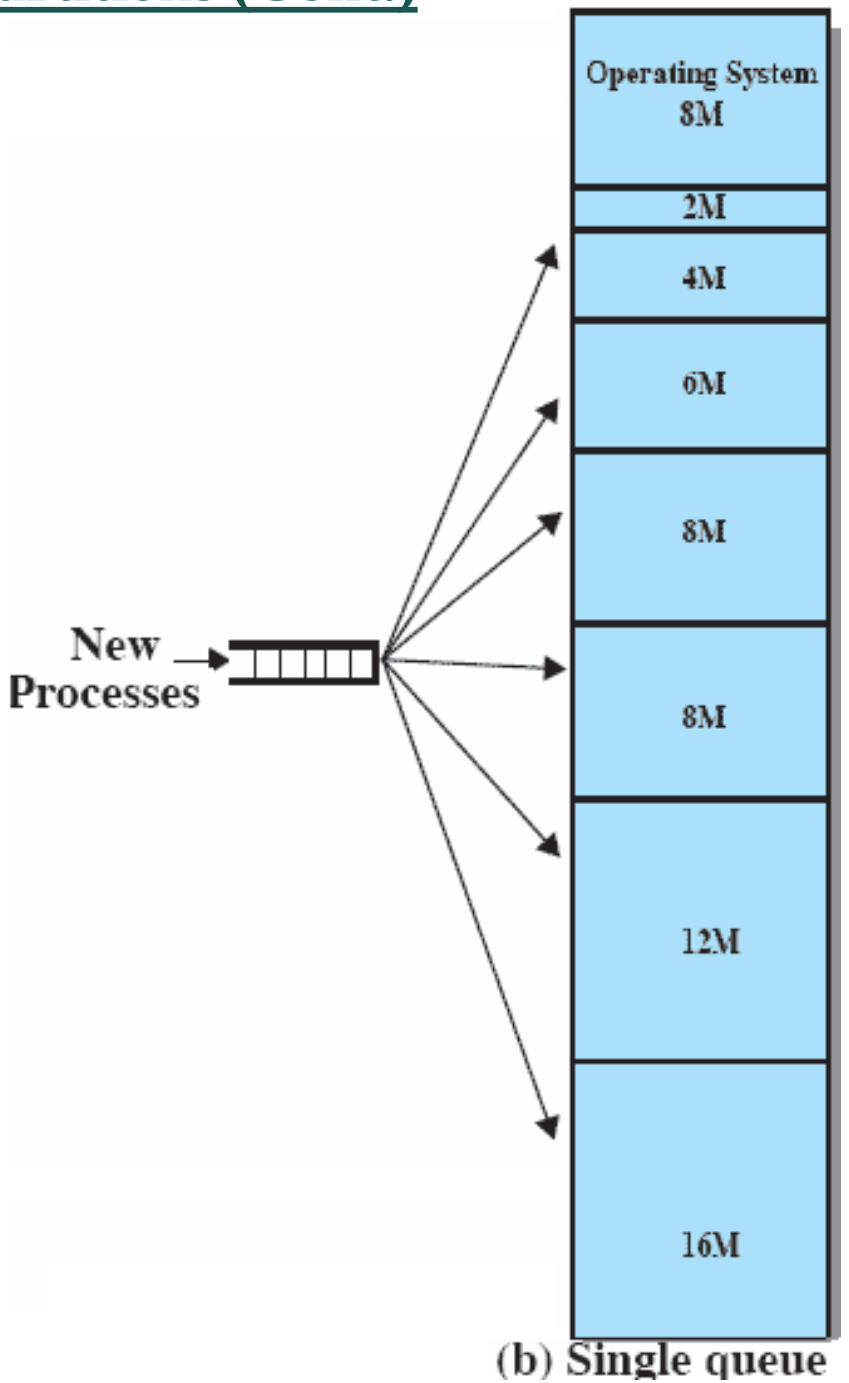
Placement algorithms for unequal-size partitions (Cont.)

2)Thus, a **preferable approach is to employ** تستخدم **a single queue for all partitions:**

When it's time to load a process into the main memory, the smallest available partition that will hold the process is selected.

If all partitions are occupied, then a swapping decision must be made.

Preference might be given to swapping out of the smallest partition that will hold the incoming process. It is also possible to consider other factors such as priority and a preference for swapping out blocked processes VS ready processes.



Advantages and Disadvantages of Fixed Partitioning

Advantages:

- unequal-size partitions provides a degree of flexibility to fixed partitioning.
- Both fixed-partitioning schemes are relatively simple and require minimal OS software and processing overhead.

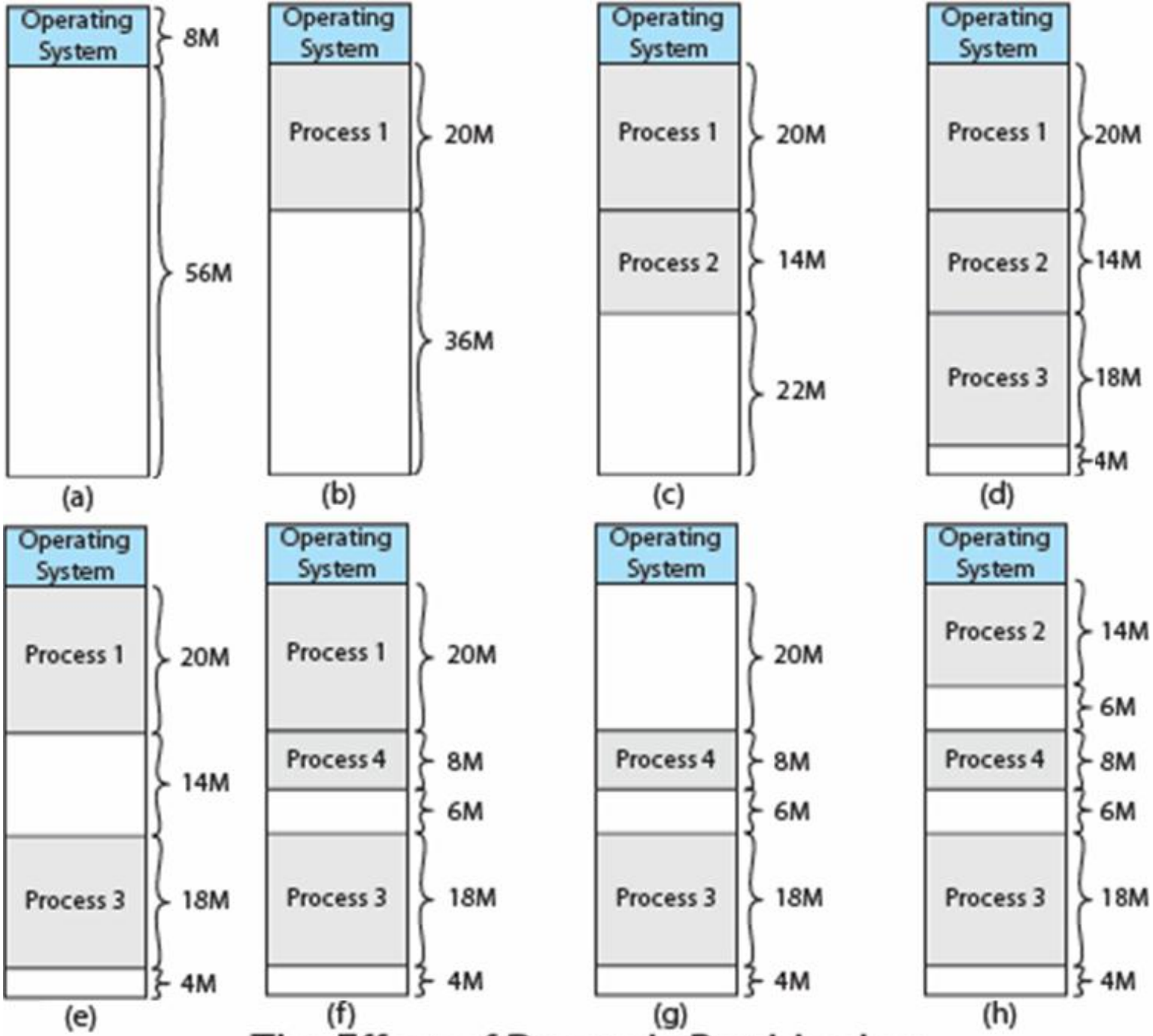
Disadvantages:

- The number of partitions specified at the system generation time: **limits** the number of the **active** processes in the system.
- Because partition sizes are preset at system generation time, small processes will not utilize partition space efficiently.

One example of the use of fixed partitioning is IBM mainframe OS: OS/MFT (Multiprogramming with a Fixed Number of Tasks).

Second: Dynamic Partitioning

- Partitions are of variable length and number.
- A process is allocated exactly as much memory as it requires.
- An important OS that used this technique was IBM's mainframe operating system, OS/MVT (Multiprogramming with a Variable Number of Tasks).



The Effect of Dynamic Partitioning

The previous example uses 64 Mbytes of the main memory. Initially, the main memory is empty, except for the OS (a). The first three processes are loaded in, starting where the OS ends and occupy just enough space for each process (b, c, d). This leaves a "**hole**" at the end of the memory that is too small for a fourth process. At some point, none of the processes in the memory is ready. The OS swaps out process 2 (e), which leaves sufficient room to load a new process, process 4 (f). Because process 4 is smaller than process 2, another small hole is created. Later, a point is reached at which none of the processes in main memory is ready, but process 2, in the Ready-Suspend state, is available. Because there is insufficient room in memory for process 2, the OS swaps process 1 out (g) and swaps process 2 back in (h).

As this example shows, this method starts out well, but eventually it leads to a situation in which there are a lot of small holes in the memory. As time goes on, memory becomes more and more fragmented, and memory utilization declines. This phenomenon is referred to as the **external fragmentation**, indicating that the memory that is external to all partitions becomes increasingly fragmented.

Dynamic Partitioning

External Fragmentation

- memory becomes more and more fragmented
- memory utilization declines

Compaction

- technique for overcoming external fragmentation
- OS shifts processes so that they are contiguous
- free memory is together in one block
- time consuming and wastes CPU time

For example, in Figure7.4h, compaction will result in a block of free memory of length 16M (6+6+4). This may be well sufficient to load in an additional process.

Note that compaction implies the need for a dynamic relocation capability. That is, it must be possible to move a process from one region to another in the main memory without invalidating the memory references in the program.

Placement Algorithms

When it is time to load or swap a process into the main memory and if there is more than one free block of memory of sufficient size, the OS must decide which free block to allocate.

Best-fit	First-fit	Next-fit
• chooses the block that is closest in size to the request	• begins to scan memory from the beginning and chooses the first available block that is large enough	• begins to scan memory from the location of the last placement and chooses the next available block that is large enough

All are limited to choosing among free blocks of the main memory that are equal to or larger than the process to be brought in.

Memory Configuration Example

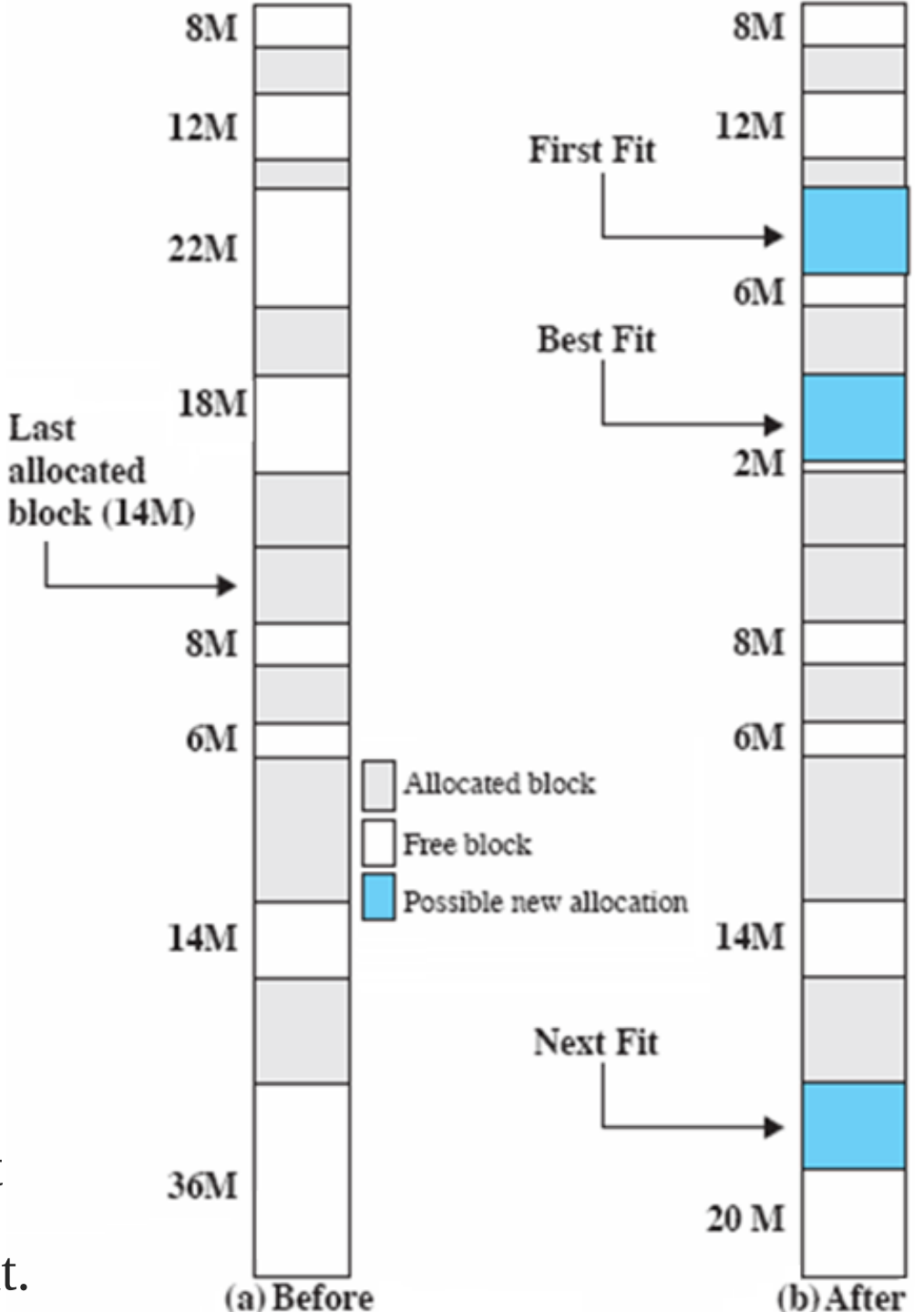
After a number of placement and swapping-out operations. The last block that was used was a 22-MB block from which a 14-MB partition was created (leading to 8MB fragment; 22-14=8).

Figure **(b)** shows the difference between the best,first,and next-fit placement algorithms in satisfying a 16MB allocation request.

Best-fit will search the entire list of available blocks and use the 18-Mbyte block, leaving a 2-MB fragment.

First-fit results in a 6-MB fragment

Next-fit results in a 20-MB fragment.



Which of these approaches is best? will depend on the exact sequence of process swapping that occurs and the size of those processes. However, **some general comments can be made:**

The first-fit algorithm isn't only the simplest but also the best & fastest.

The next-fit algorithm tends to produce slightly worse results than the first-fit. It will more frequently lead to an allocation from a free block at the end of memory. The result is that the largest block of free memory (which usually appears at the end of the memory space) is quickly broken up into small fragments. Thus, compaction may be required more frequently with the next-fit.

On the other hand, the first-fit algorithm may litter the front end with small free partitions that need to be searched over on each subsequent first-fit pass.

The best-fit is usually the worst performer. Because it looks for the smallest block that will satisfy the requirement, it guarantees that the fragment left behind is as small as possible. Although each memory request always wastes the smallest amount of memory, the result is that the main memory is quickly littered by مملوءةبـ blocks that are too small to satisfy memory allocation requests. Thus, memory compaction must be done more frequently than with the other algorithms.

Paging

- Partition memory into equal fixed-size chunks that are relatively small.
- Process is also divided into small fixed-size chunks of the same size.

Suppose that the main memory is partitioned into equal fixed-size chunks that are relatively small, and that each process is also divided into small fixed-size chunks of the same size. Then the chunks of a process, known as **pages,** could be assigned to available chunks of memory, known as **frames, or page frames.**

The wasted space in the memory for each process is due to the **internal fragmentation** consisting of only a fraction of the last page of a process.

There is no external fragmentation.

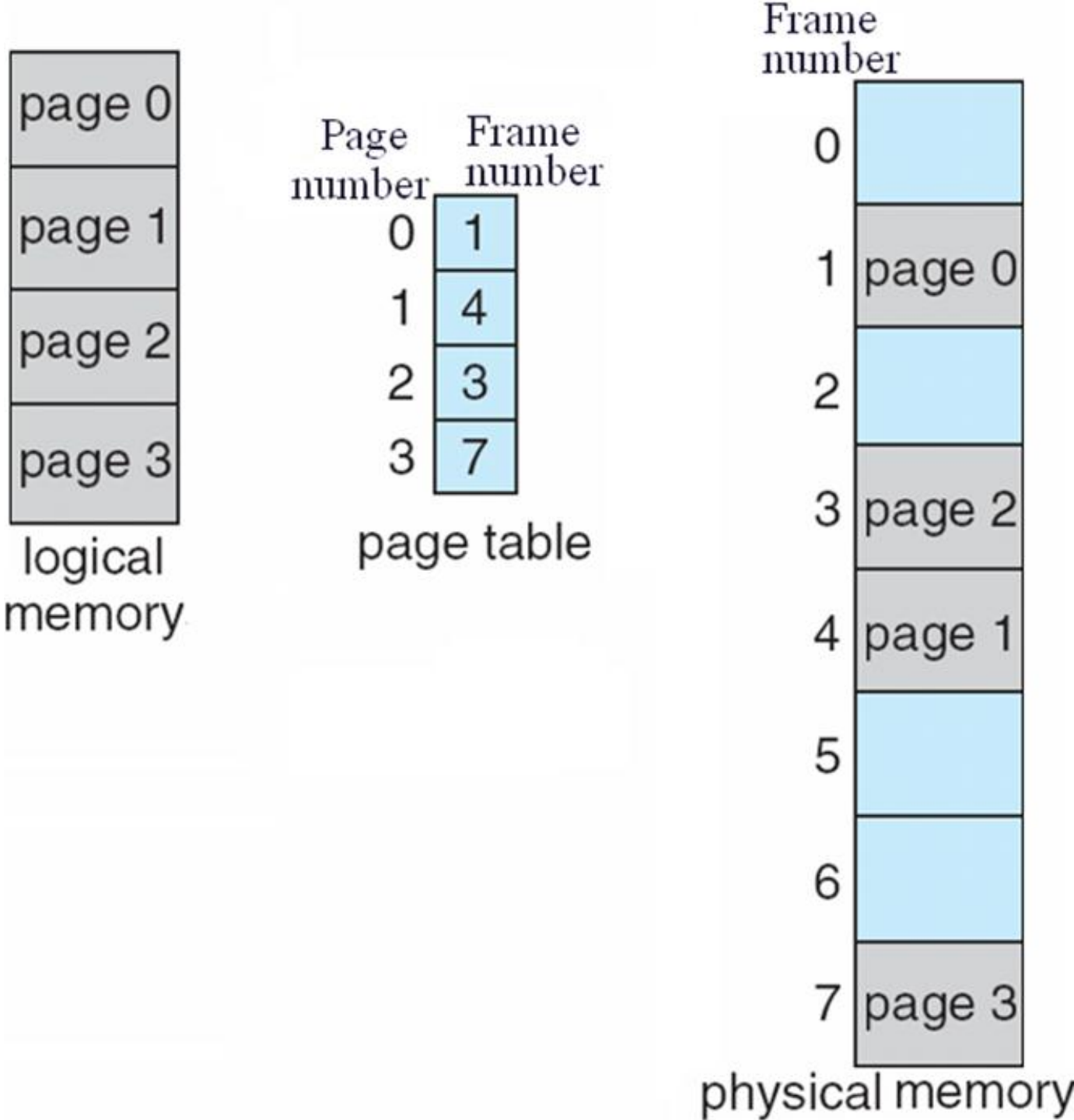
The assignment of Processes Pages to Free Frames

Some of the frames in the memory are in use and some are free. A list of free frames is maintained by the OS. Process A, stored on disk, consists of 4 pages. When it is time to load it, the OS finds 4 free frames and loads its four pages into the 4 frames. Process B, consists of 3 pages, and process C, consists of 4 pages are subsequently loaded.

Then process B is suspended & swapped out of the main memory. Later, all the processes in the main memory are blocked, and the OS needs to bring a new process that is process D(consists of 5 pages)

Page Table

- Maintained by the OS for each process.
- Contains the frame location for each page in the process.
- Used by the processor to produce a physical address.



Segmentation

A user program can be subdivided using segmentation, in which **the program & its associated data are divided into a number of segments**. It is not required that all segments of all programs be of the same length, although there is a maximum segment length.

Because of the use of unequal-size segments, segmentation is similar to dynamic partitioning. The difference (compared to dynamic partitioning) is that with segmentation a process may occupy more than one partition, and these partitions need not to be contiguous.

Segmentation **eliminates internal fragmentation.**

Segmentation (Cont.)

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

  the main program,

  a function,

  local variables,

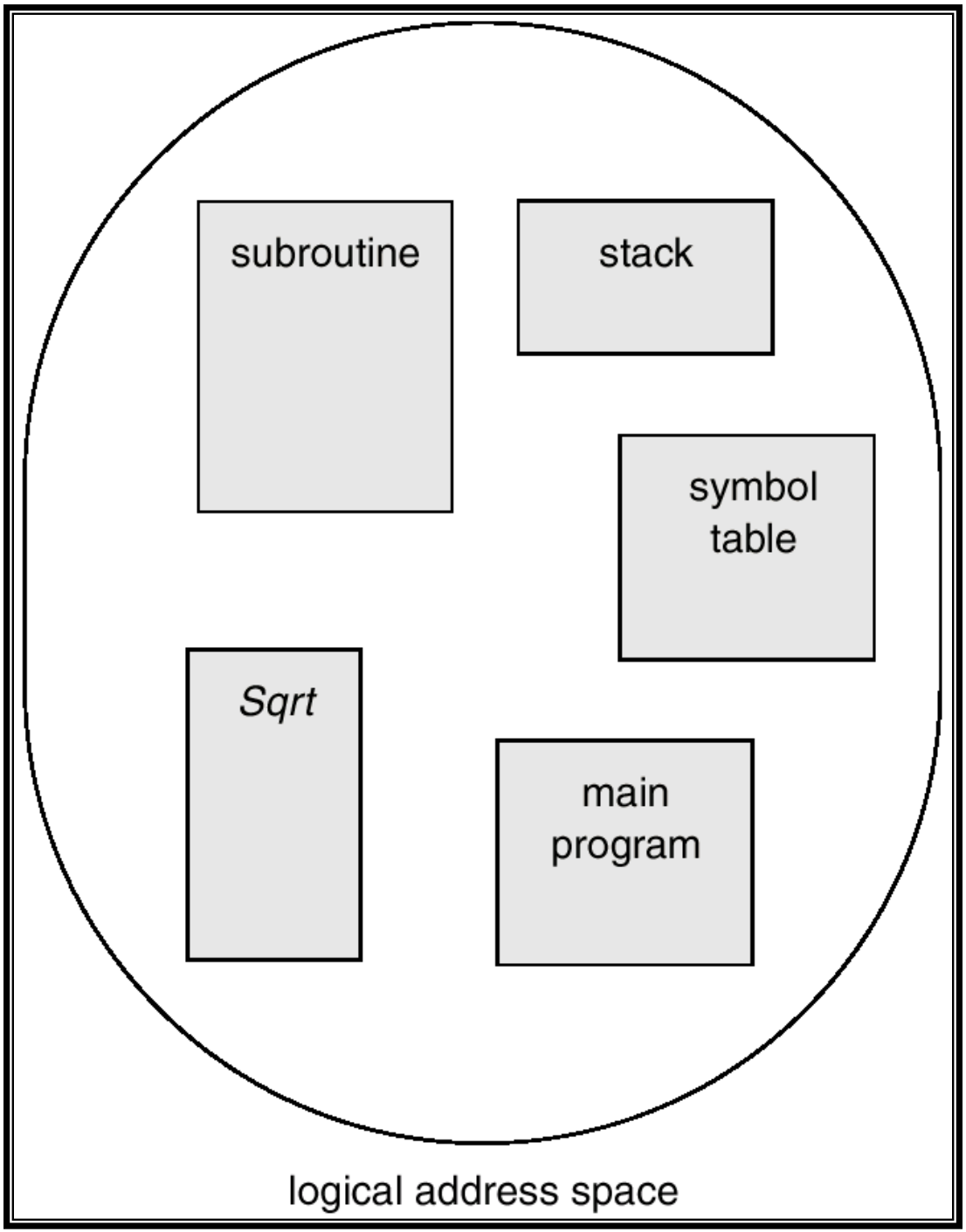
  global variables,

  a stack, and

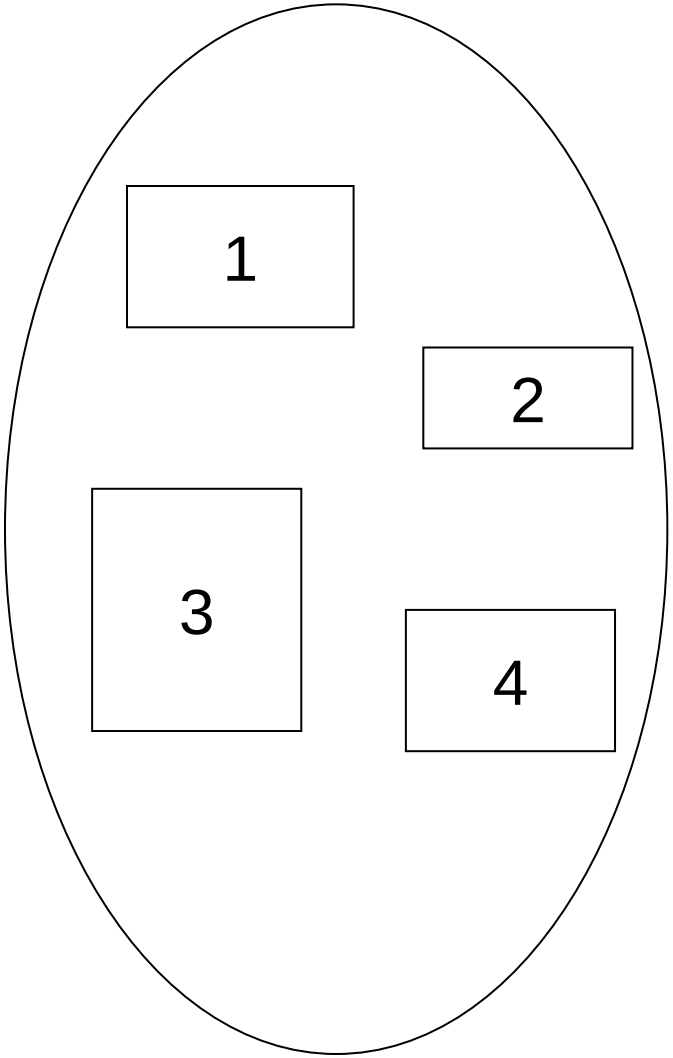
  an array.

EX:Normally,when a program is compiled, the compiler automatically constructs segments reflecting the input program. A **C compiler** may create separate segments for the following:1.The code,2.Global variables, 3. The stacks used by each thread, 4. The standard C library.
The loader will take all these segments and assign them segment numbers

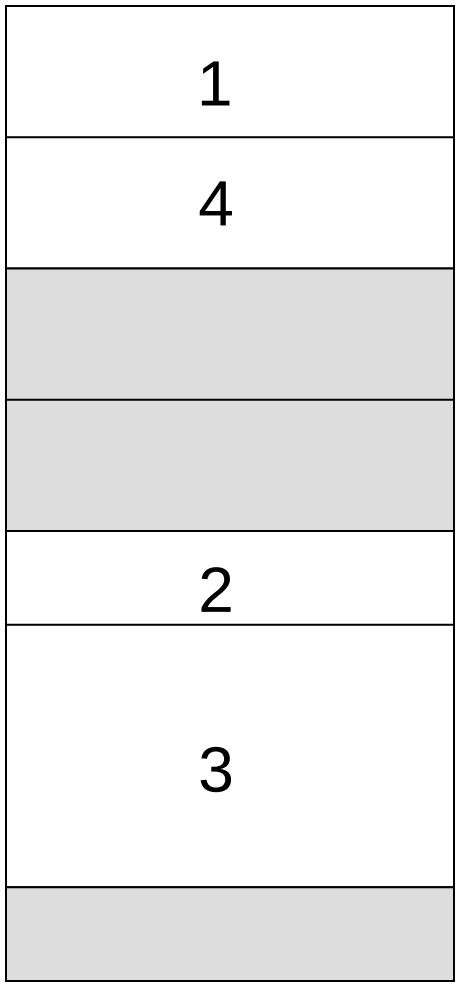
User's View of a Program



Logical View of Segmentation



The user space



The physical memory space

Questions Samples

Define: Internal fragmentation, compaction.

True or False:

- Processes must be relocated to the same area of memory.
- Paging eliminates external fragmentation.

Complete: …. partitions memory into small equal fixed-size chunks.

Do as required in each of the following:

- What is the importance of memory management.
- What are the disadvantages of equal-size partitioning?
- What is the disadvantage of having a scheduling queue for each partition with unequal-fixed partitioning?