

Chapter 4:

Memory Management

Contents

- Introduction to Memory Management
 - Basic Memory Management Mechanism
 - Memory Allocation Techniques
 - Swapping and Paging
 - Segmentation with Paging
-

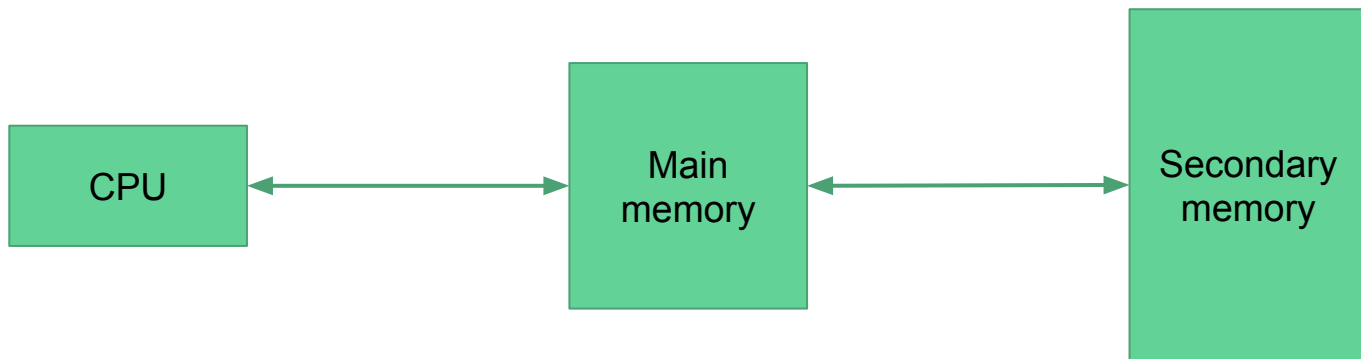
Background

Memory Hierarchy



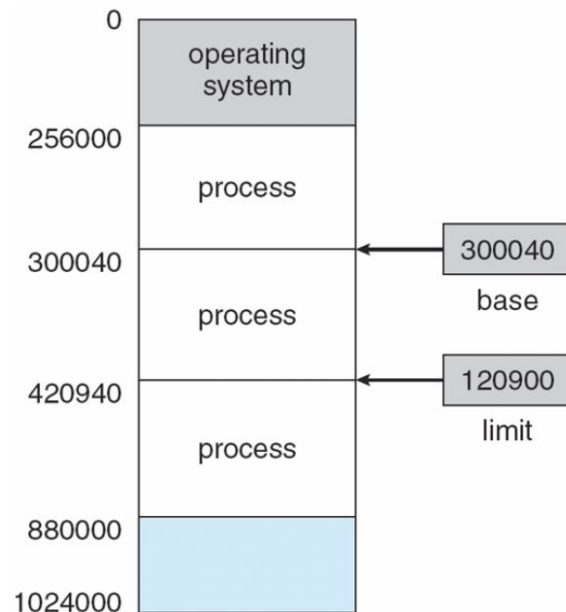
Background

- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly



Background

- Memory is partitioned into two parts:
 - One for the OS
 - The other for the user area, which needs to be further divided into multiple parts for various user processes
- This division of memory for processes needs proper management, including efficient allocation and protection.



Memory management

- The part of the operating system that manages (part of) the memory hierarchy is called the memory manager
- Keeps track of which parts of memory are in use
- Allocates memory to processes when they need it
- Deallocates memory when the processes are complete

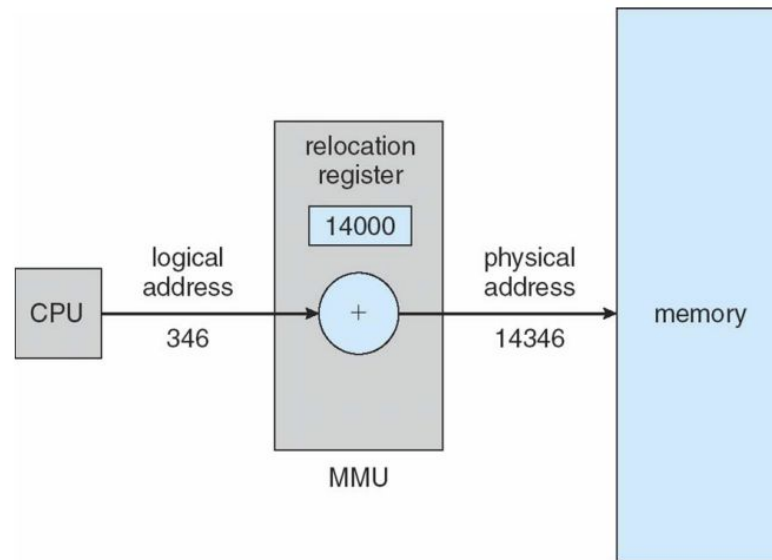
Basic concepts

Static and dynamic memory allocation

- In static allocation, the allocation is done before the execution of a process
- In dynamic allocation, the memory allocation is deferred till the process starts executing
 - Required for multi-processing

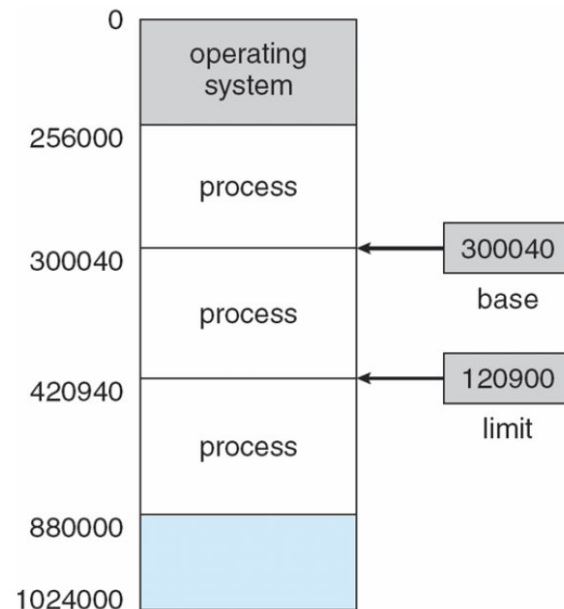
Basic concepts

- In dynamic memory allocation, the process is loaded in memory initially with all the memory references in relative form. The absolute addresses in memory are calculated as an instruction in the process is executed.
- Those addresses in relative form generated by the CPU are known as **logical addresses**
- The absolute addresses in memory are called **physical addresses**

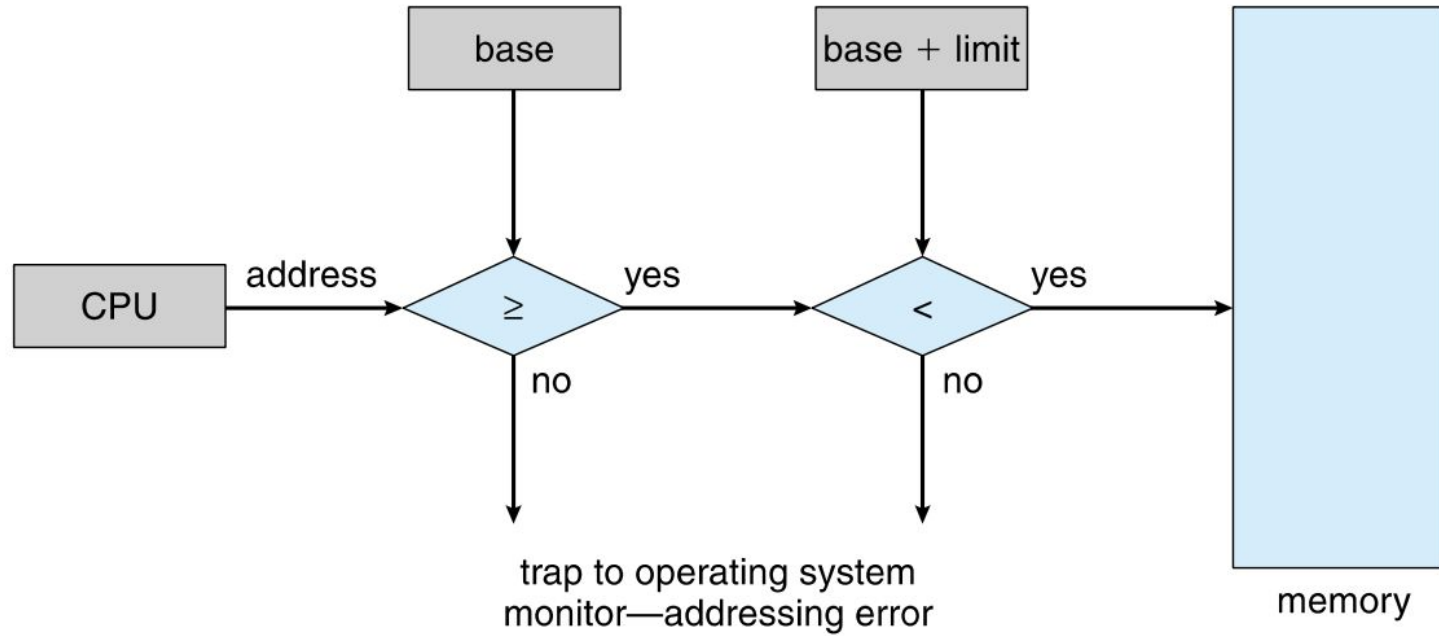


Relocation

- Translation of logical addresses to physical addresses
- Classical solution is to equip each CPU with two special hardware registers, called **base and limit registers**
- Base register holds the starting address in the main memory from where the process needs to be relocated
- Limit register holds the ending location of the process in the main memory



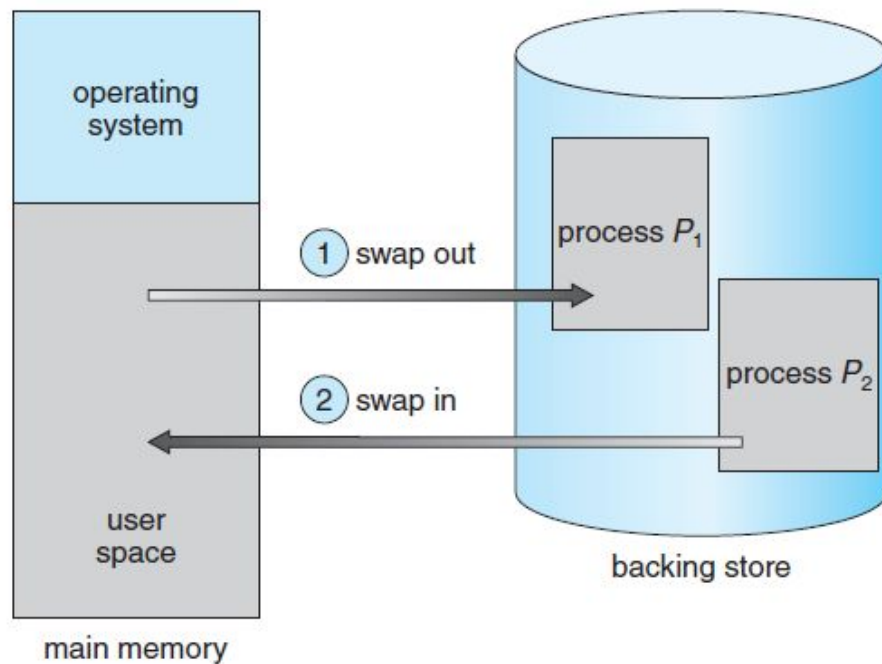
Relocation



Swapping

Taking a process out of memory temporarily (e.g., when there is no memory for executing a new process) to a backing store = **swap out**

Bringing the process back into memory from the backing store for continued execution = **swap in**



Memory allocation

- Contiguous memory allocation
- Non-contiguous memory allocation

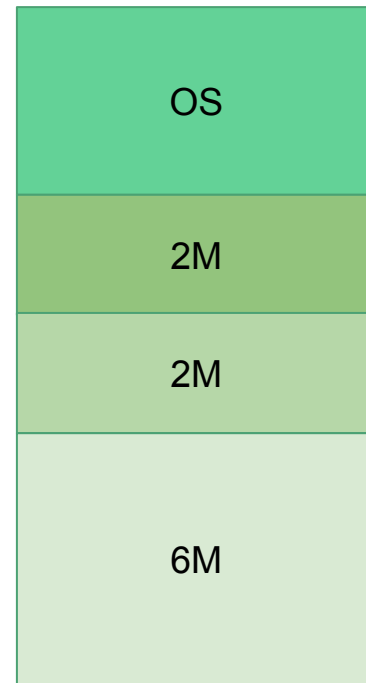


Contiguous memory allocation

- A process is allocated a single contiguous area in memory
- Memory can be partitioned in two ways:
 - Fixed memory partitioning
 - Variable memory partitioning

Contiguous allocation with fixed partitioning

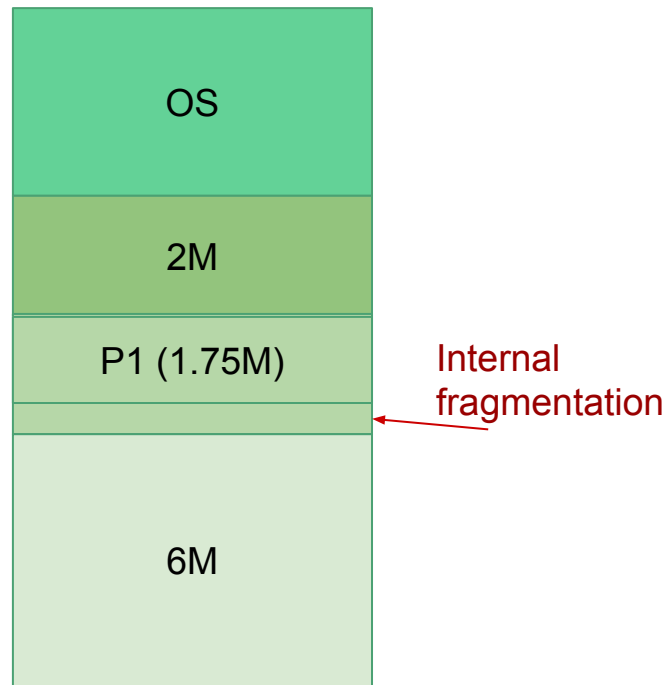
- Fixed partitioning method generates memory partitions of fixed size
- To allocate memory to a process, the memory partition that fits the process (called a **hole**) is searched and is allocated to the process
- When the process terminates, the occupied memory is released and the hole is available again



Contiguous allocation with fixed partitioning

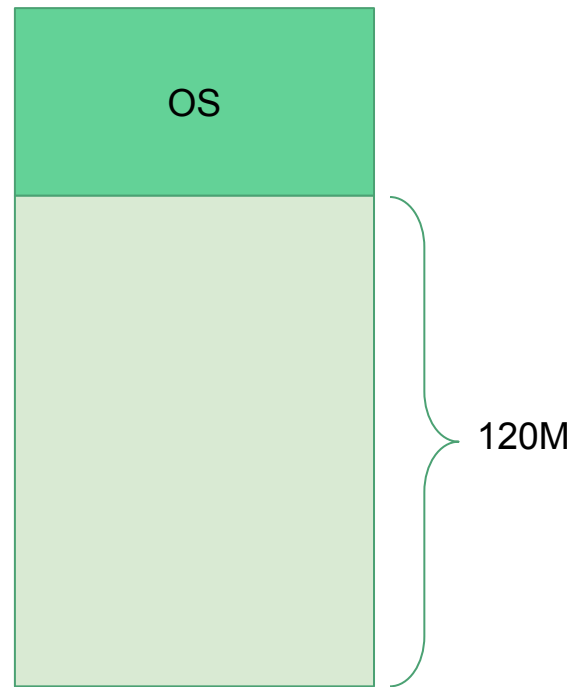
- **Internal Fragmentation**

- When a process is allocated to a partition, it may be possible that its size is less than the size of the partition, leaving a space after the allocation, which is unusable by any other process
- This wastage of memory, internal to a partition, is known as internal fragmentation

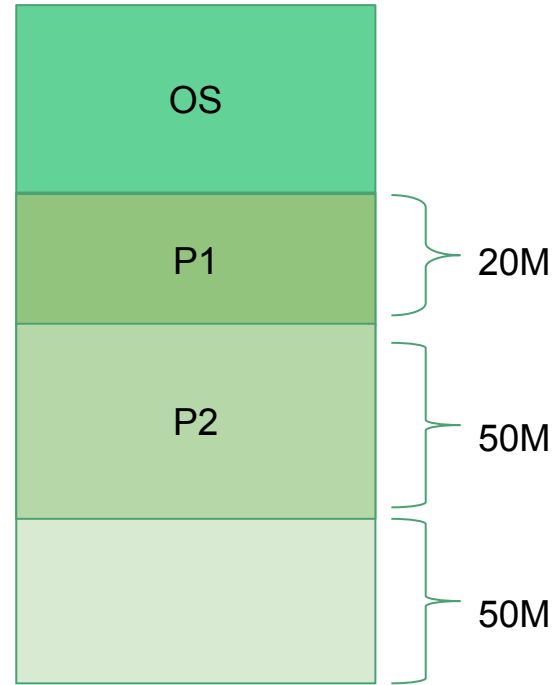
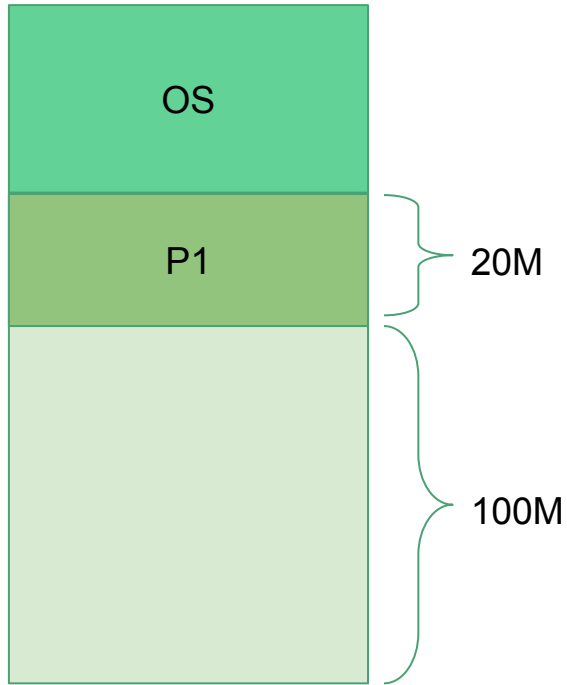


Contiguous allocation with variable partitioning

- In variable partitioning, the number and the size of partitions are variable and created at run time, i.e., when a process is brought to the ready queue
- Like with fixed partitioning, a hole is searched for and allocated to a process
- Initially, there is only a single large hole



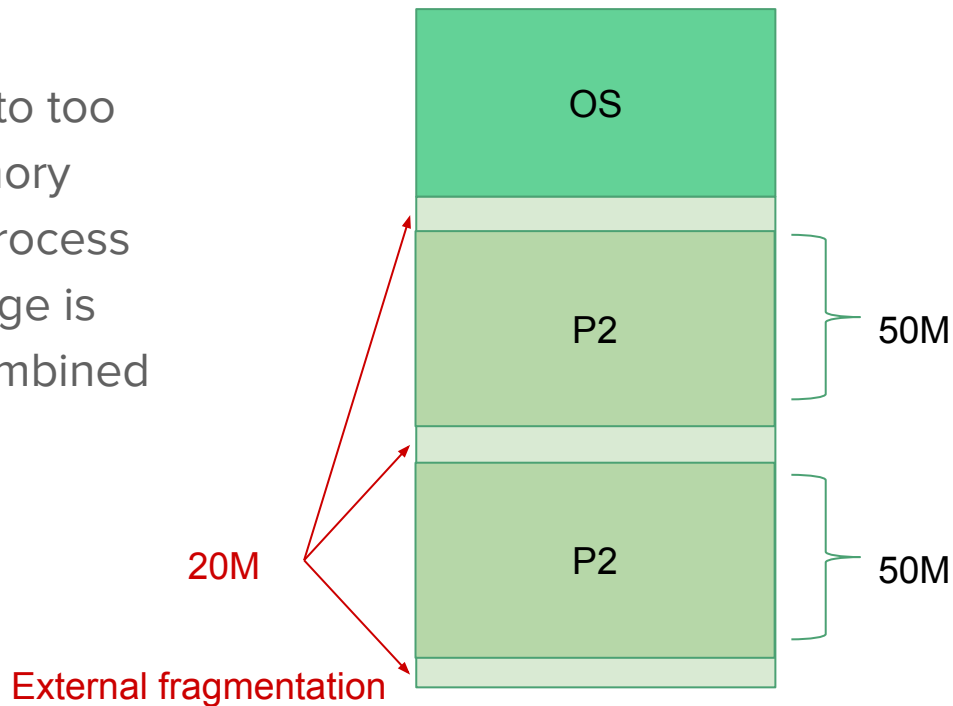
Contiguous allocation with variable partitioning



Contiguous allocation with variable partitioning

- **External Fragmentation**

- When the memory is divided into too small pieces such that the memory cannot be allocated to a new process even though enough free storage is available if these pieces are combined



Memory allocation techniques

- First-fit allocation
- Best-fit allocation
- Worst-fit allocation



Memory allocation techniques

Algorithms that decide which hole from the list of free holes must be allocated to the process

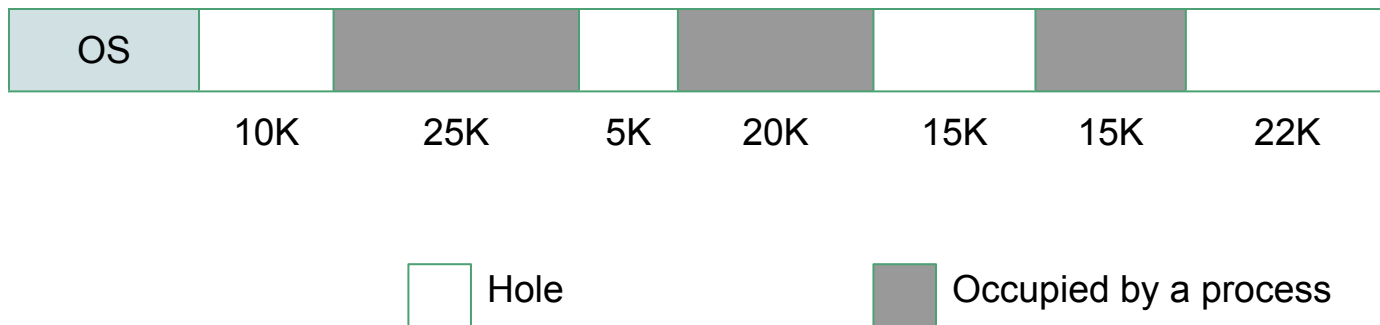
First-fit: Allocate the first hole that is big enough

Best-fit: Allocate the smallest hole that is big enough; Produces the smallest leftover hole

Worst-fit: Allocate the largest hole; Produces the largest leftover hole

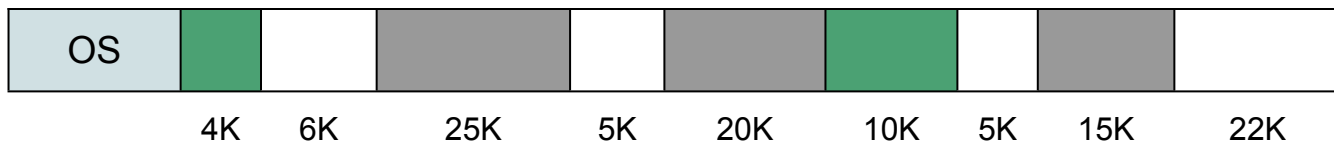
Example




Consider the memory allocation scenario as shown below. Allocate memory for additional requests of 4K and 10K (in this order) using first-fit, best-fit and worst-fit allocation methods.



First-fit allocation

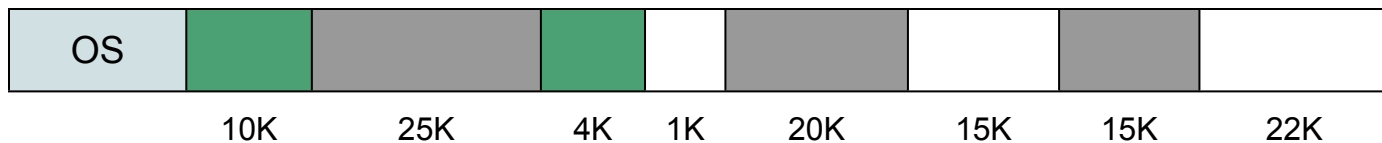
Allocate the first hole that is big enough






 Hole  Occupied by a process  Occupied by new processes

Best-fit allocation

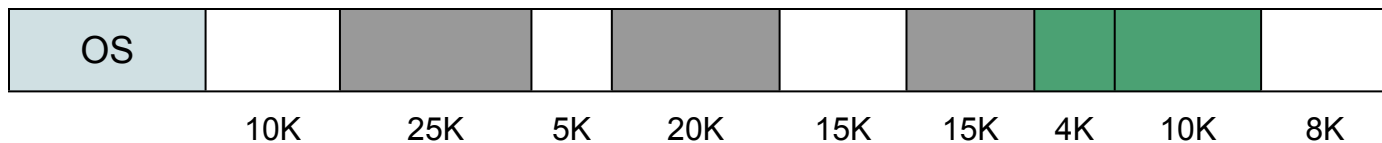
Allocate the smallest hole that is big enough






 Hole  Occupied by a process  Occupied by new processes

Worst-fit allocation

Allocate the largest hole



 Hole  Occupied by a process  Occupied by new processes

Problems with contiguous allocation

Problems with fixed partitions:

- Limited degree of multiprogramming
- Internal fragmentation

Problems with variable partitions:

- External fragmentation, compaction.

Alternative: Non-contiguous allocation

Non-contiguous memory allocation

- Paging
- Segmentation



Non-contiguous memory allocation

Divides the logical address space of a process into several blocks (pages or segments) which are loaded into the different area of memory space according to the availability of the memory.

Non-contiguous memory allocation with fixed partitioning = **Paging**

Non-contiguous memory allocation with variable partitioning = **Segmentation**

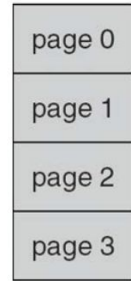
Paging

Eliminates external fragmentation but internal fragmentation may exist

The idea:

- Divide main memory into equal-size partitions, called **frames**
- Divide the logical memory of a process into blocks, called **pages** of a process, which are of the same size as frames
- Keep track of all free frames
- To run a program of size N pages, find N free frames and load program
- Set up a **page table** (one page table for each process) to translate logical to physical addresses

Paging

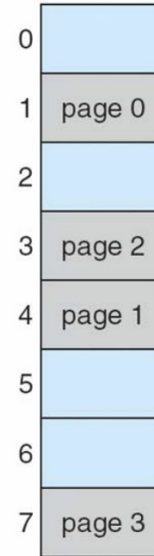


logical
memory

0	1
1	4
2	3
3	7

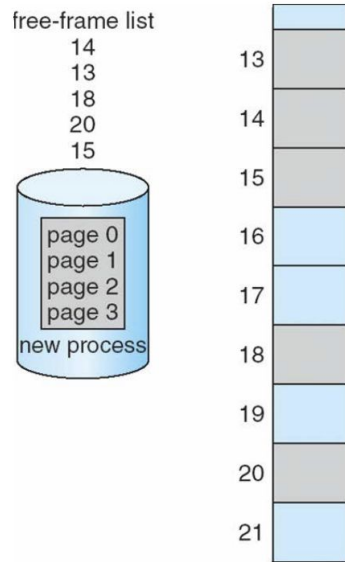
page table

frame
number



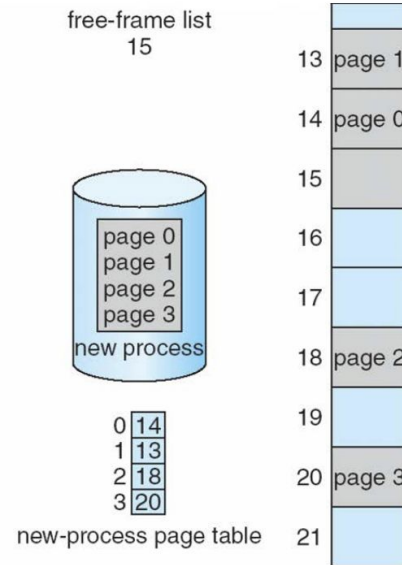
physical
memory

Paging



(a)

Before allocation



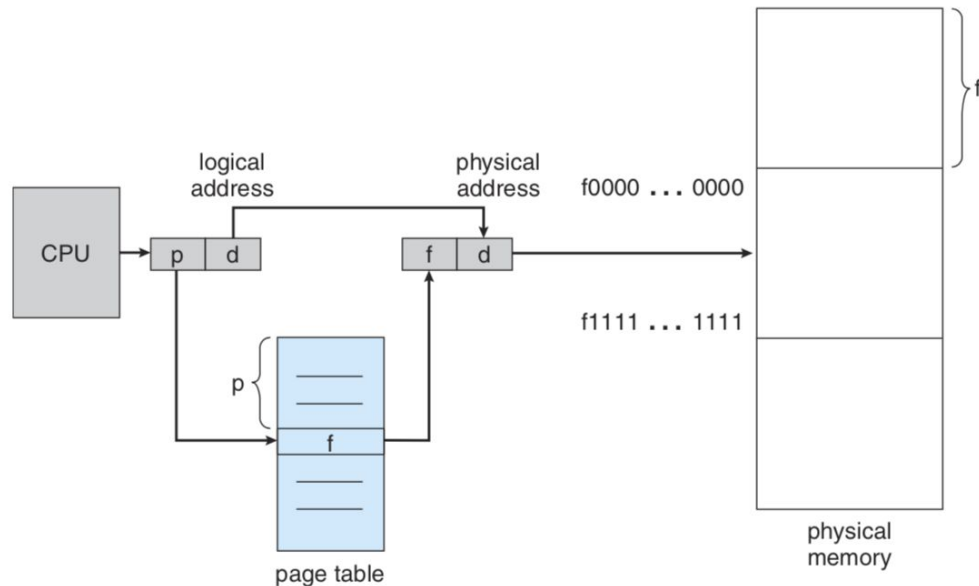
(b)

After allocation

Paging

Address translation

1. The processor generates a logical address in the form: **(page number p, offset d)**
2. Using p, the base address of the corresponding frame, f, is retrieved from the page table
3. f is added to d to get the corresponding physical address



Segmentation

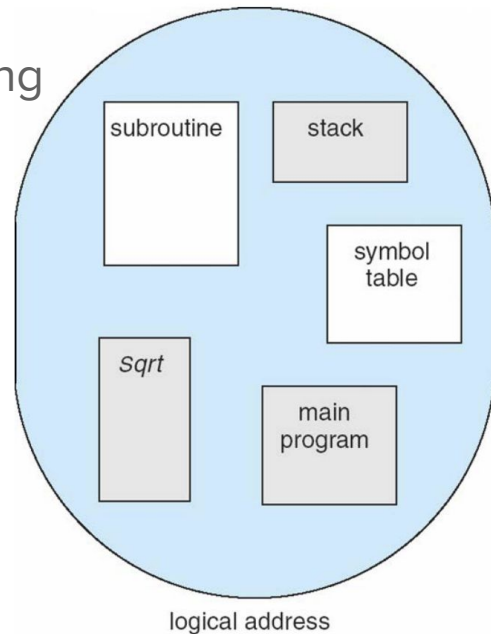
Non-contiguous memory allocation with variable partitioning

The logical address space is divided into variable-sized **segments**

Segments are logical divisions of a program, such as main program, procedure, function, method, object, local variables, global variables etc.

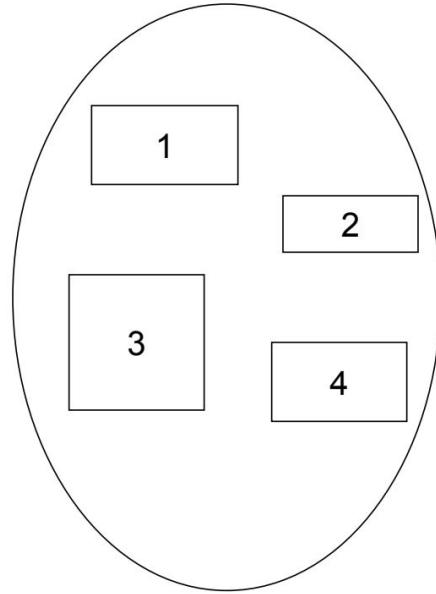
(paging = physical divisions of a program)

Segments may be of different size

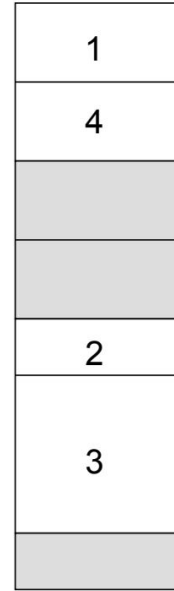


User's view of a program

Segmentation



user space



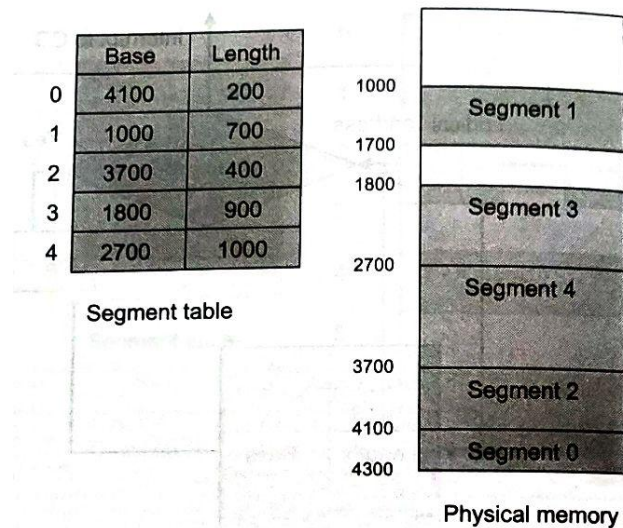
physical memory space

Segmentation

To translate logical address to physical address, a data structure, called **segment table**, is maintained. Each table entry has:

- **base** – contains the starting physical address where the segments reside in memory
- **limit** – specifies the length of the segment

Segment table resides in main memory



Segmentation

Segment-table base register (STBR) points to the segment table's location in memory

Segment-table length register (STLR) indicates number of segments used by a program

segment number s is legal if $s < \text{STLR}$

Address translation in segmentation

1. Logical address consists of two parts: segment number s and offset d
2. s is used to retrieve from segment table the base address of the segment in memory and the length of the segment
3. If d is less than the segment length, d is added to the base address to generate the physical address. Otherwise, the address is not valid.

