What is Memory Management?

Memory Management

- **Memory Management** is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.
- It is the most important function of an operating system that manages primary memory. It helps processes to move back and forward between the main memory and execution disk.
- It helps OS to keep track of every memory location, irrespective of whether it is allocated to some process or it remains free.

Why Use Memory Management?

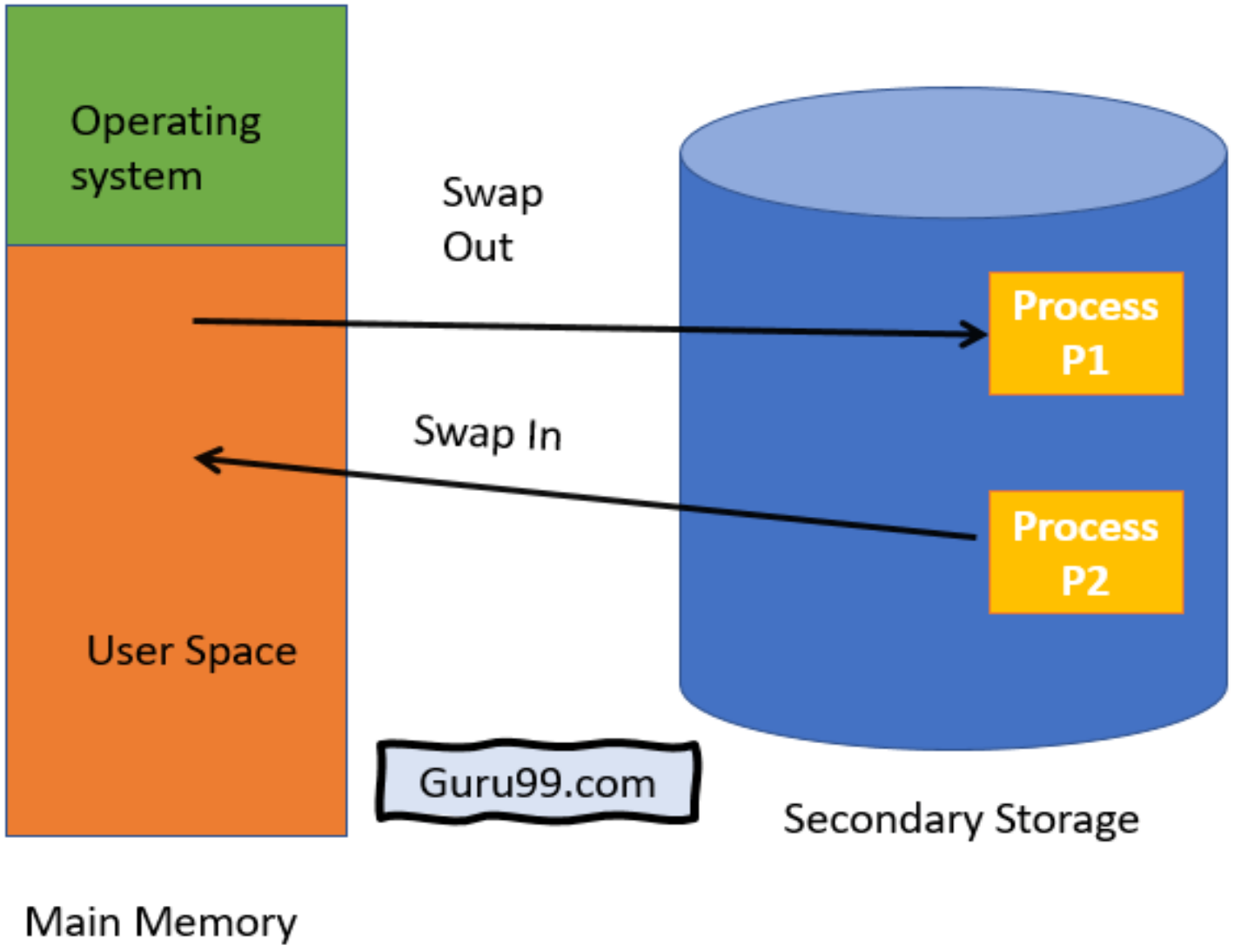
- It allows you to check how much memory needs to be allocated to processes that decide which processor should get memory at what time.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also make sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

Memory Management Techniques

- **Single Contiguous Allocation**
- It is the easiest memory management technique. In this method, all types of computer's memory except a small portion which is reserved for the OS is available for one application. For example, MS-DOS operating system allocates memory in this way. An embedded system also runs on a single application.
- **Partitioned Allocation**
- It divides primary memory into various memory partitions, which is mostly contiguous areas of memory. Every partition stores all the information for a specific task or job. This method consists of allotting a partition to a job when it starts & unallocate when it ends.

- **Paged Memory Management**
- This method divides the computer's main memory into fixed-size units known as page frames. This hardware memory management unit maps pages into frames which should be allocated on a page basis.
- **Segmented Memory Management**
- Segmented memory is the only memory management method that does not provide the user's program with a linear and contiguous address space.
- Segments need hardware support in the form of a segment table. It contains the physical address of the section in memory, size, and other data like access protection bits and status.

- **What is Swapping?**
- Swapping is a method in which the process should be swapped temporarily from the main memory to the backing store. It will be later brought back into the memory for continue execution.
- Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users. It is also capable of offering direct access to these memory images.


Operating system
Swap Out
Process P1
Swap In
Process P2
User Space
Guru99.com
Secondary Storage
Main Memory

Swapping

- A process needs to be in memory to be executed
- A process however can be swapped temporarily out of memory to a backing store.
- Assume multiprogramming environment with RR scheduling
- When the quantum expires the memory manager will start to swap the process to backing store.
- The quantum must be sufficiently large that reasonable amounts of computing are done between swaps
- A variant of this swapping policy is used for priority scheduling algorithms
- Here higher priority process swap in by memory manager and the low priority process will be swapped out

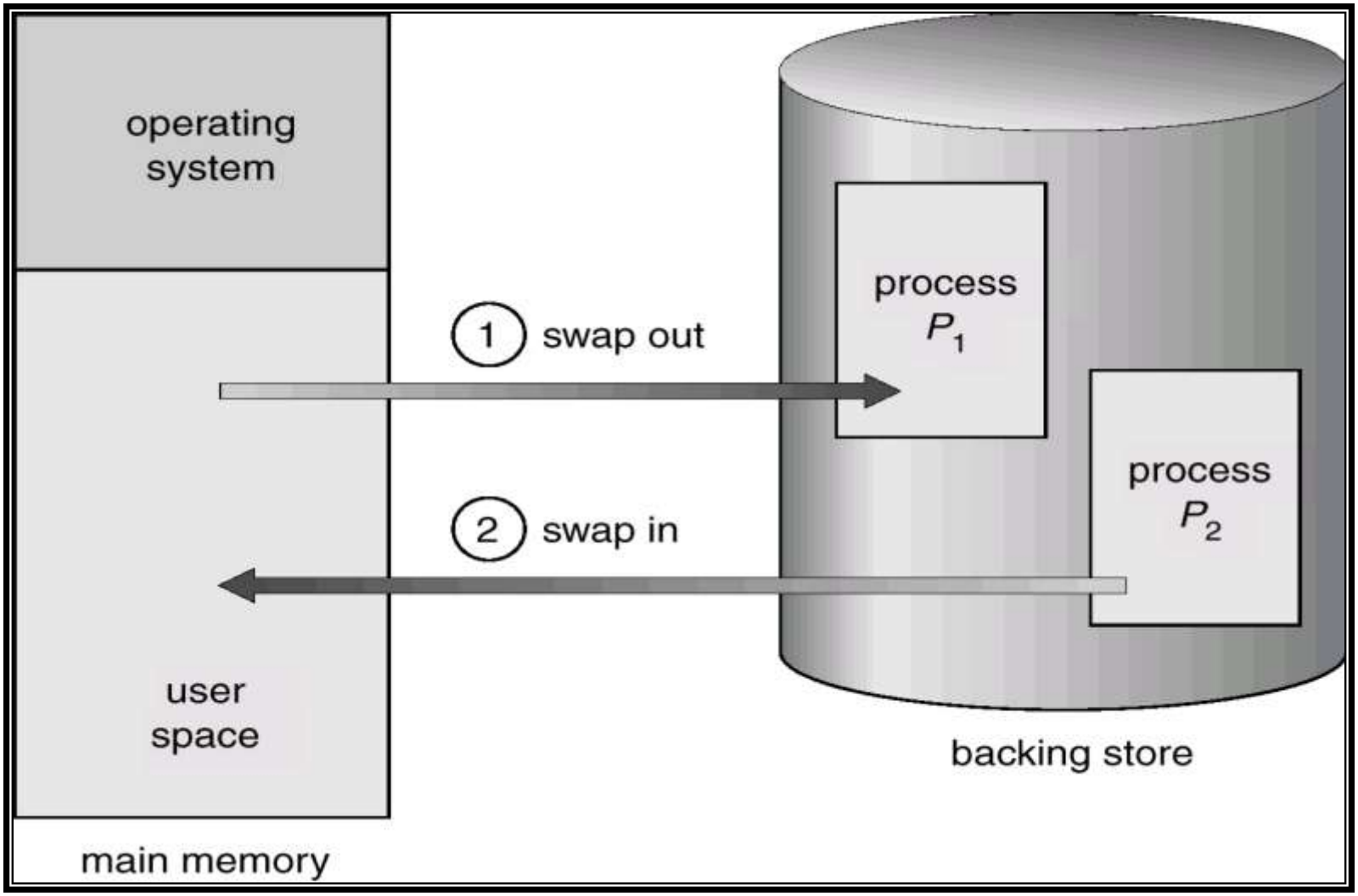
Swapping

- The context switch time in some swapping system is fairly high
- Assume a process size is 1MB and the hard disk transfer rate is 5MB per second.
- 1000KB/5000KB per second = 1/5 second => 200 millsec
- If the average head seek latency is 8 milliseconds means then that process takes 208 milliseconds
- Both swap in and swap out it takes 416 milliseconds
- For efficient CPU utilization, we want our execution time for each process must be high than swap time.

Swapping

- Swapping is constrained by the following factor
- If we want to swap a process that must be completely idle
- If a process is waiting for an I/O we can swap that process only when it is idle that is it should not accessing I/O operations
- Currently standard swapping is used in few systems
- It requires much swapping time and provides too little execution time
- Modified versions of swapping are found on many systems such as some versions of UNIX

Schematic View of Swapping

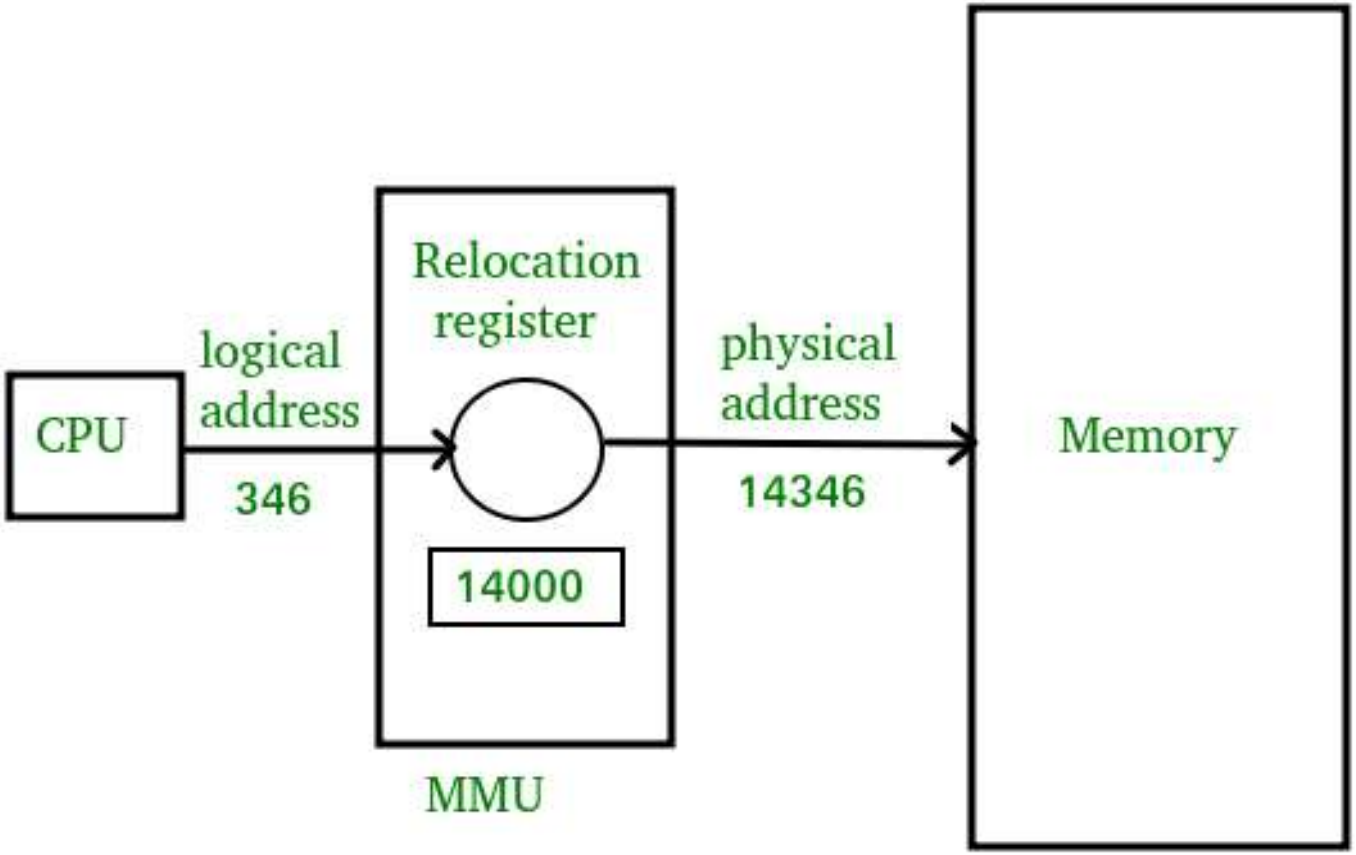


Logical and Physical address

- **Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.
- This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program's perspective.
- The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

- **Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address.
- The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used.
- The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.

Mapping Virtual Addresses to Physical Addresses



- CPU will generate logical address for eg: 346
- MMU will generate relocation register(base register) for eg:14000
- In Memory physical address is located eg:(346+14000= 14346)

Memory Allocation

- One of the simplest method for memory allocation is to divide memory into several fixed-sized partitions
- Each partition may contain exactly one process
- When a partition is free then a process is loaded into the free partition.
- When a process terminates, the partitions becomes available for another process.
- This method was originally used by the IBM OS/360 operating system.
- But it is no longer in use.
- Next method is generalization of fixed-partition scheme

Memory Allocation

- The operating system keeps a table indicating which parts of memory are available and which are occupied
- Initially all memory is available for user processes and is considered as one large block of available memory, a **hole.**
- When a process arrives and needs memory, we search for a hole large enough for this process.
- If we find one, we allocate only as much memory as is needed.
- Memory is allocated to processes until, finally, the memory requirements of the next process cannot be satisfied

Memory Allocation

- In general, a set of holes, of various sizes, is scattered throughout memory at any given time.
- When a process arrives and needs memory, the system searches this set of hole that is large enough for this process.
- If the hole is too large then it splits into two.
- When a process terminates it releases its block of memory.
- If the released block of memory is adjacent to other holes, then these holes are merged to form a one larger hole.
- How to allocate this available memory to processes?

Memory Allocation

- First fit – Allocate the first hole that is big enough. Searching start either at the beginning of the set of holes or where the previous first fit search ended.
- Best fit- Allocate the smallest hole that is big enough. We must search the entire list
- Worst fit- Allocate the largest hole
- Both first fit and best fit are better than worst fit in terms of storage utilization.
- These algorithms suffer from external fragmentation

Problem

- Given five memory partitions of 100Kb, 500Kb, 200Kb, 300Kb, 600Kb (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 212 Kb, 417 Kb, 112 Kb, and 426 Kb (in order)? Which algorithm makes the most efficient use of memory?

- First-fit:
- 212K is put in 500K partition
- 417K is put in 600K partition
- 112K is put in 288K partition (new partition 288K = 500K - 212K)
- 426K must wait

- Best-fit:
- 212K is put in 300K partition
- 417K is put in 500K partition
- 112K is put in 200K partition
- 426K is put in 600K partition

- Worst-fit:
- 212K is put in 600K partition
- 417K is put in 500K partition
- 112K is put in 388K partition
- 426K must wait