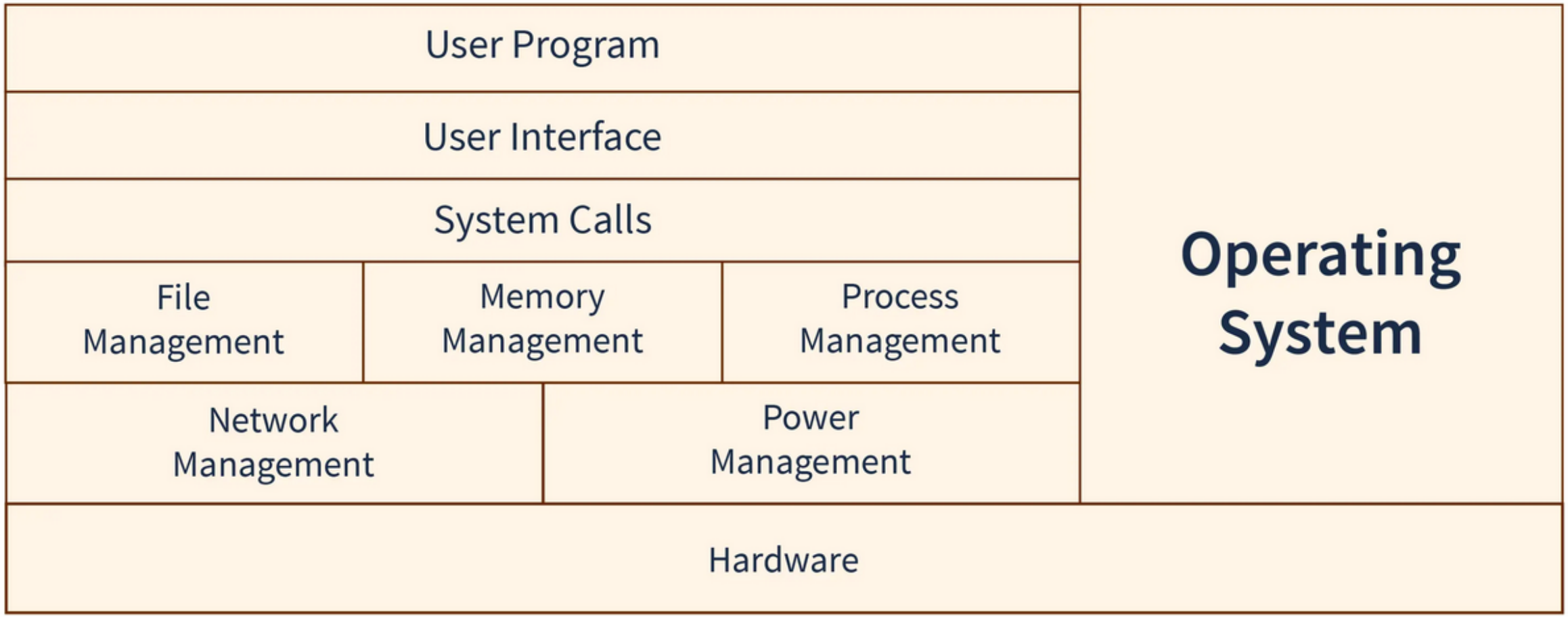
Memory Management

- The term memory can be defined as a collection of data in a specific format and used to store instructions and process data.
- The memory comprises a large array or group of words/bytes, each with its own location.
- Every program along with its information should be in main memory(**RAM**) during execution.
- CPU fetches instructions from memory according to the value of program counter.
- To achieve multiprogramming and proper utilization of memory, memory management is an important aspect.

What is Memory Management in an Operating System?

- Memory management in OS is a technique of controlling and managing the functionality of Random access memory (primary memory).
- It is used for achieving better concurrency, system performance, and memory utilization.
- Memory management moves processes from primary memory to secondary memory and vice versa. It also keeps track of available memory, memory allocation, and unallocated.



Memory Management in OS

Logical vs. Physical Address

Logical Address:

Definition:

A logical address, also known as a virtual address, is the address generated by the CPU during a program's execution. It is the address that a program uses to access its data and instructions.

Usage:

Logical addresses are used by the CPU and the program itself to refer to specific locations in memory. These addresses are typically generated by the program counter and are translated into physical addresses by the memory management unit (MMU) before actual data retrieval or storage.

Abstraction:

Logical addresses provide an abstraction layer that allows programs to run independently of the underlying hardware configuration. A program doesn't need to know the actual physical location of its data in memory; it works with logical addresses.

Dynamic Nature:

Logical addresses can be dynamic, changing as a program executes and different sections of the program are loaded into and removed from memory.

Physical Address:

Definition:

A physical address refers to the actual location in the computer's memory hardware (RAM) where data or instructions are stored. It represents a specific location on the computer's physical memory chip.

Translation:

The translation from logical address to physical address is done by the Memory Management Unit (MMU) in coordination with the operating system. The MMU performs address translation to map logical addresses to their corresponding physical addresses.

Fixed Nature:

Physical addresses are fixed and do not change during the execution of a program. Once a program is loaded into memory, the physical addresses associated with its data and instructions remain constant.

Direct Connection:

Physical addresses directly correspond to the hardware architecture and organization of the computer's memory.

Static and Dynamic Loading

Static Loading:

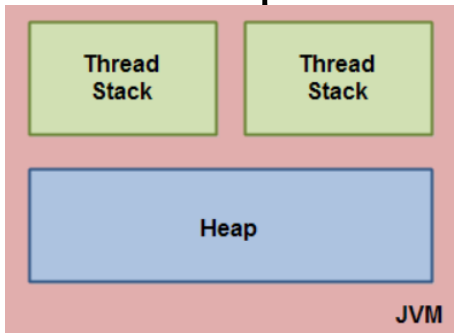
- Occurs at compile time.
- All modules are loaded into memory before execution.
- Memory allocation is fixed.
- Simpler, but potentially less memory-efficient.

Dynamic Loading:

- Occurs at runtime.
- Modules are loaded on demand.
- Memory allocation is more flexible.
- More complex, but potentially more memory-efficient.

Memory Management of Threads

The difference between threads and processes in terms of memory management is that threads within one process run in a shared memory while processes run in a separate memory.



There are two types of memory- Stack memory and Heap memory

Stack memory-

It is used for thread execution, it stores local variables, method calls and arguments within one thread. It is small and fast and always referenced in LIFO (Last-In-First-Out) order.

Heap memory-

It stores objects that are created inside heap memory until there is a reference to them from somewhere in the application. As soon as there is no reference to an object, the object is removed by the GC and heap memory is freed up. It is large and slow.

Thread: Memory Allocation

Thread memory allocation involves allocating memory for thread-specific data, including the thread's execution stack and thread-local storage.

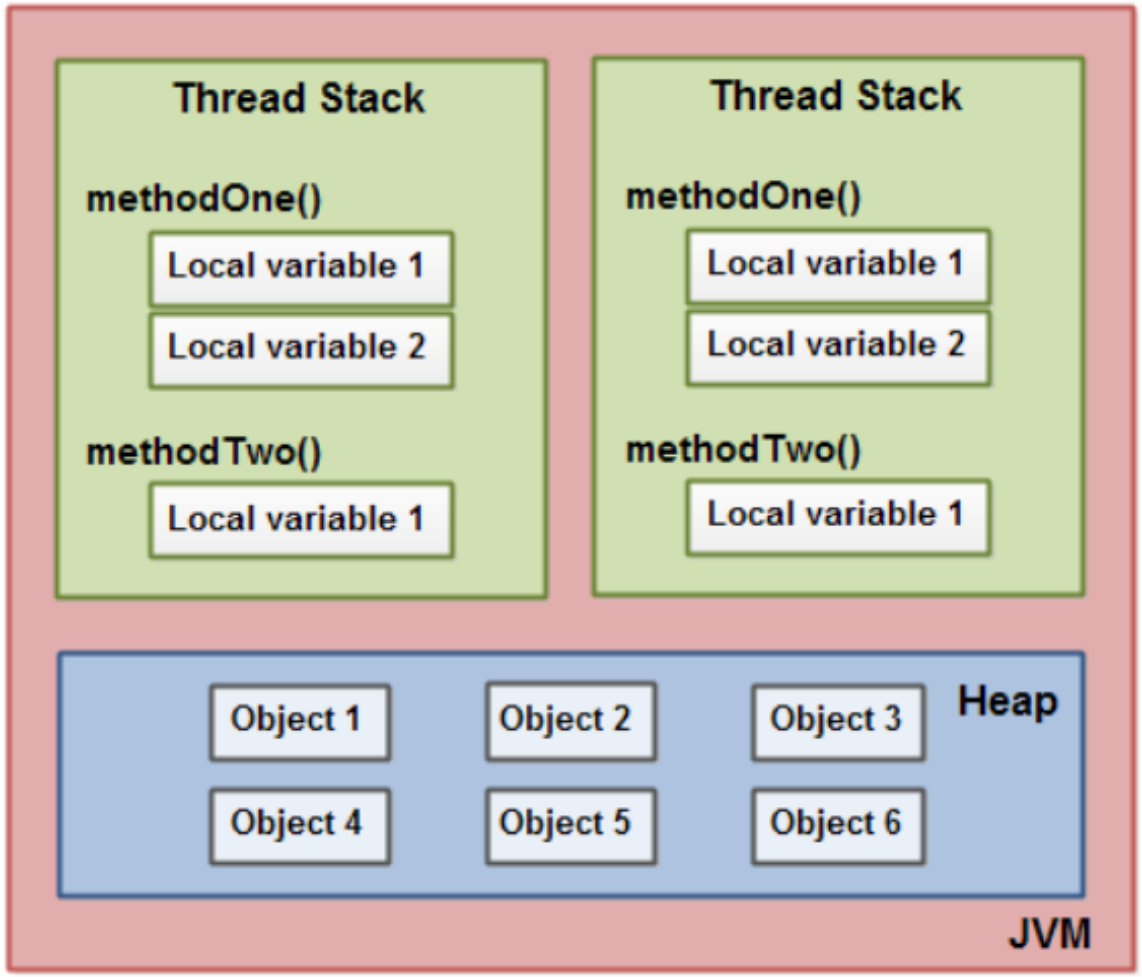
Thread Stack-
Each thread in a program has its own stack for method calls and local variables. The thread stack stores the execution context for each thread, including method call information, local variables, and thread-specific data.

Heap Memory-
Thread-related data that needs to be shared among multiple methods or threads is typically allocated on the heap, which is a region of memory separate from the thread stack. Heap memory is managed differently from stack memory and requires explicit allocation and deallocation.

Thread-Specific Storage-
Threads may also have access to thread-local storage (TLS), which allows each thread to have its own private storage for variables. TLS is typically used when data needs to be unique to each thread and remain persistent throughout the thread's execution.

Thread: Memory Allocation

Following diagram illustrates the call stack and local variables stored in thread stack and objects stored in the heap memory area-



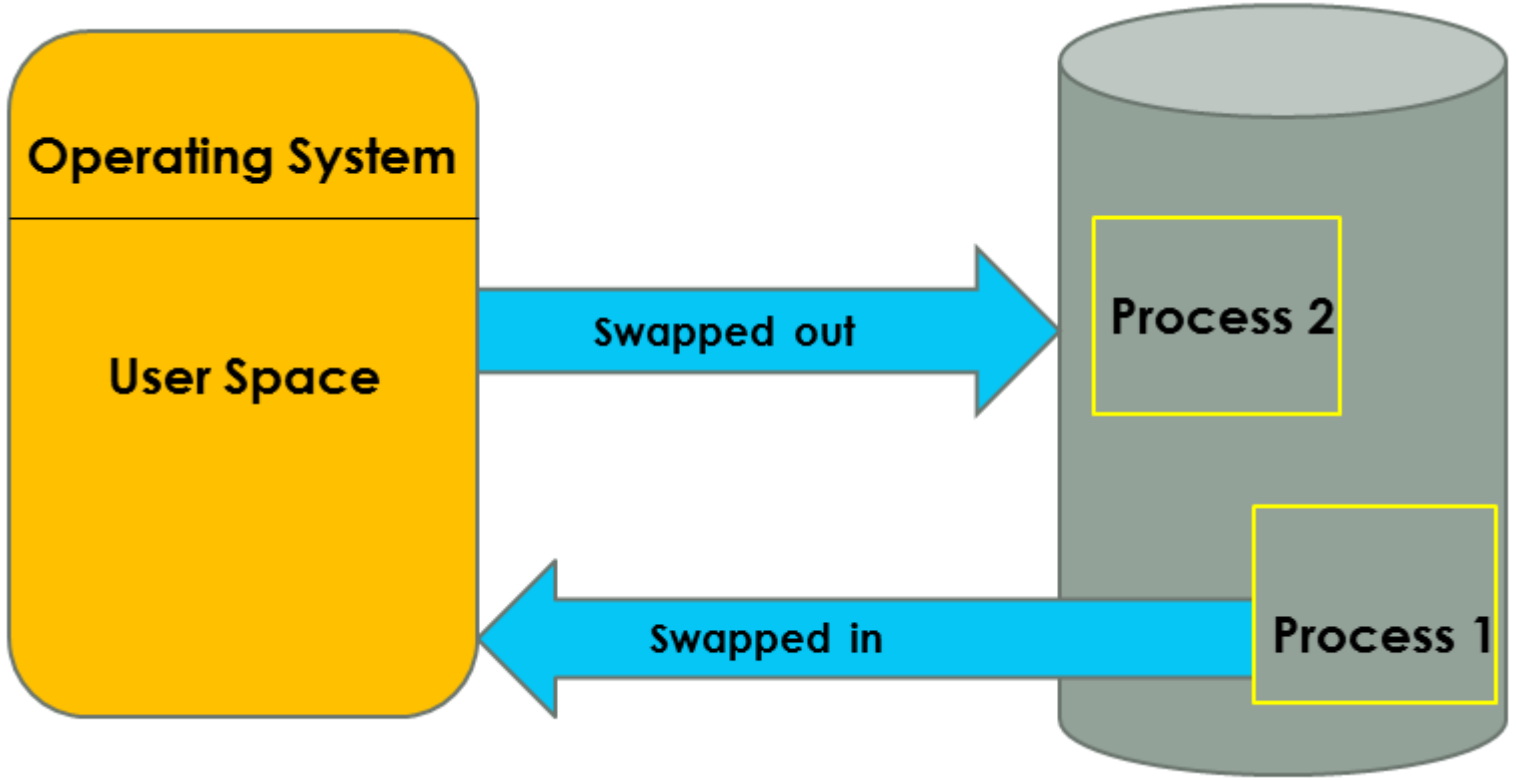
Static and Dynamic Linking

Static Loading involves loading all the necessary program components into the main memory before the program's execution begins. This means that both the executable code and data are loaded into predetermined memory locations. This allocation is fixed and does not change during the program's execution. Executable files resulting from static linking often have extensions like .exe (Windows) or have no extension (Unix-like systems).

Dynamic Loading, on the other hand, takes a more flexible approach. In this scheme, a program's components are loaded into the main memory only when they are specifically requested during execution. This results in a more efficient use of memory resources as only the necessary components occupy space. Executable files resulting from dynamic linking often have extensions like .dll (Windows) or .so (Shared Object, Unix-like systems).

Swapping

When process is to be executed then that process is taken from secondary memory to stored in RAM.But RAM have limited space so we have to take out and take in the process from RAM time to time. This process is called **swapping**. The purpose is to make a free space for other processes. And later on, that process is swapped back to the main memory.



It is further divided into two types:

Swap-in: Swap-in means removing a program from the hard disk and putting it back in the RAM.

Swap-out: Swap-out means removing a program from the RAM and putting it into the hard disk.

Paging

Paging is the memory management technique in which secondary memory is divided into fixed-size blocks called pages, and main memory is divided into fixed-size blocks called frames.

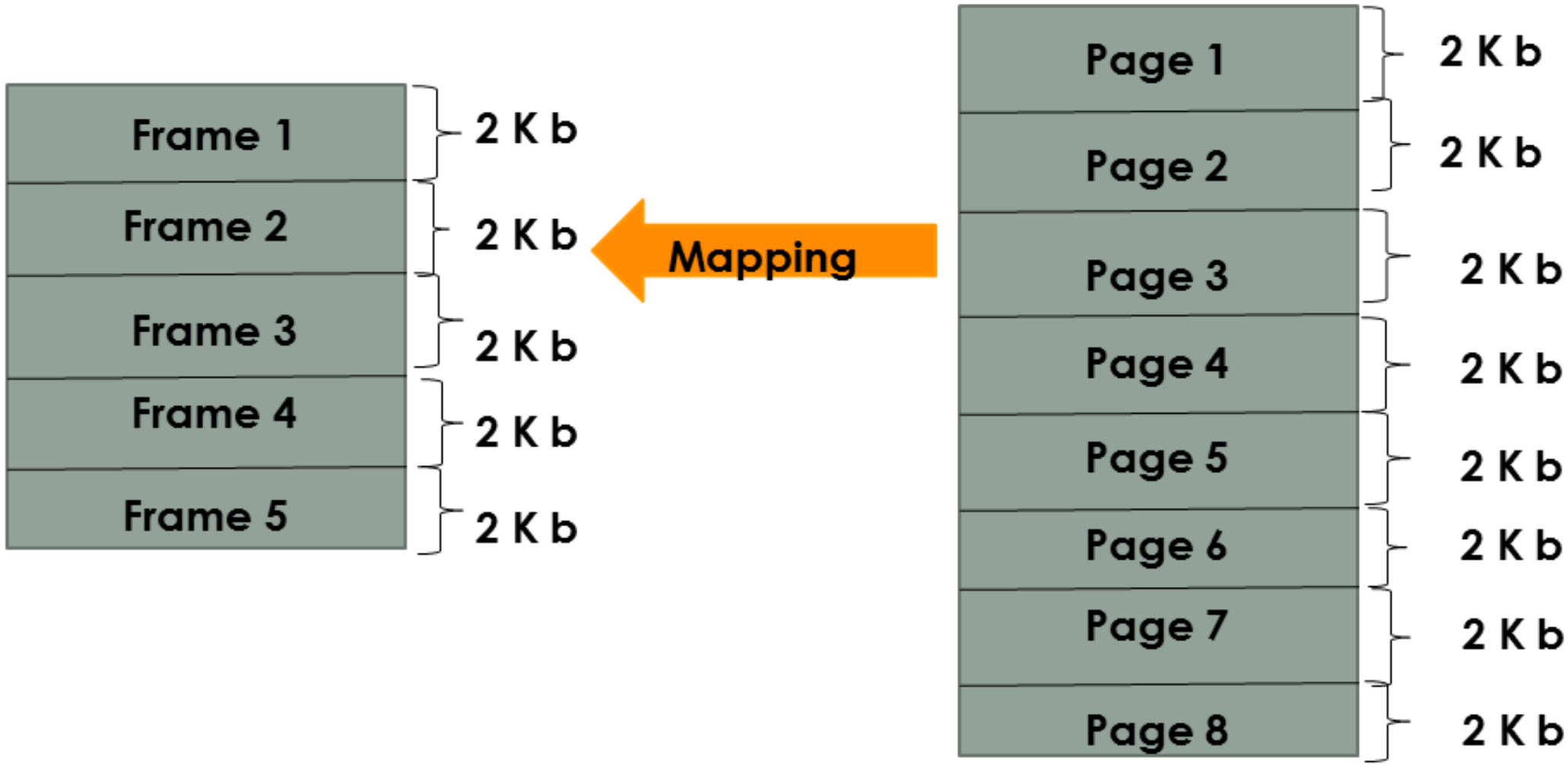
The Frame has the same size as that of a Page.

The processes are initially in secondary memory, from where the processes are shifted to main memory(RAM) when there is a requirement.

Each process is mainly divided into parts where the size of each part is the same as the page size.

One page of a process is mainly stored in one of the memory frames. Paging follows no contiguous memory allocation.

That means pages in the main memory can be stored at different locations in the memory.


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
2 K b
2 K b
2 K b
2 K b
2 K b
Mapping
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Page 8
2 K b
2 K b
2 K b
2 K b
2 K b
2 K b
2 K b
2 K b

What is Fragmentation?

When processes are moved to and from the main memory, the available free space in primary memory is broken into smaller pieces.

This happens when memory cannot be allocated to processes because the size of available memory is less than the amount of memory that the process requires. Such blocks of memory stay unused. This issue is called fragmentation.

1. External Fragmentation:
The total amount of free available primary is sufficient to reside a process, but can not be used because it is non-contiguous.

External fragmentation can be decreased by compaction or shuffling of data in memory to arrange all free memory blocks together and thus form one larger free memory block.

2. Internal Fragmentation:

Internal fragmentation occurs when the memory block assigned to the process is larger than the amount o memory required by the process.

In such a situation a part of memory is left unutilized because it will not be used by any other process. Internal fragmentation can be decreased by assigning the smallest partition of free memory that is large enough for allocating to a process.

Segmentation

Segmentation is a memory management technique used in computer systems where the main memory is divided into variable-sized segments. Each segment corresponds to a logical unit of a program, such as a function, method, or data structure. Segmentation provides a more flexible memory organization compared to contiguous memory allocation, where memory is treated as a single, large block.

Here are key concepts related to segmentation:

Segment:

A segment is a logical unit of a program, and it can represent different types of data or code. Common segments include code segments, data segments, stack segments, and heap segments.

Address Space:

In a segmented memory model, the address space of a process is divided into segments. Each segment has its own base address and length.

Segment Table:

To keep track of the segments associated with a process, a segment table is used. This table contains information such as the base address and length of each segment. The operating system uses the segment table during address translation.

Address Translation:

When a program references a memory address, the system uses the segment table to translate the logical address into a physical address. The physical address is used to access the actual location in main memory.

Segmentation Fault:

If a program attempts to access memory outside the boundaries of its segments, it results in a segmentation fault. This is a common runtime error indicating a memory access violation.

Benefits of Segmentation:

Flexibility: Segmentation allows for a more flexible memory organization, accommodating varying sizes and types of data.
Modularity: Different parts of a program (segments) can be developed and compiled independently, enhancing modularity.

Drawbacks of Segmentation:

Fragmentation: Segmentation can lead to both internal fragmentation (unused memory within a segment) and external fragmentation (unused memory scattered between segments).
Complexity: Implementing segmentation requires additional hardware support and can be more complex than contiguous memory allocation.

Segmentation with Paging:

In some systems, segmentation is combined with paging to create a two-level memory hierarchy. Each segment is divided into pages, providing advantages of both segmentation (flexibility) and paging (efficient use of memory).

Protection:

Segmentation facilitates the implementation of memory protection. Each segment can have its own access rights, controlling whether it is read-only, read-write, executable, etc.

Examples:

•The x86 architecture uses a segmented memory model, where programs are divided into segments such as code segment, data segment, and stack segment.