# OS7 - Memory Management and Deadlocks

Monday, 17 July 2023 6:18 PM

CPU → Registers

CPU → L1 L2 L3 caches } Non-volatile

CPU → RAM

CPU → Persistent Storage

HDD | SSD (Volatile)

Program (SSD, 60 GB) → RAM (8 GB)

60 GB into 8 GB?

How ☹

Memory Management

1. **Continuous Memory Allocation**

   Cons:

   A lot of unallocated memory gets wasted which could otherwise be used for larger processes.

1 GB

3 GB

2 GB

4 GB

3 GB unallocated

7 GB only allocated

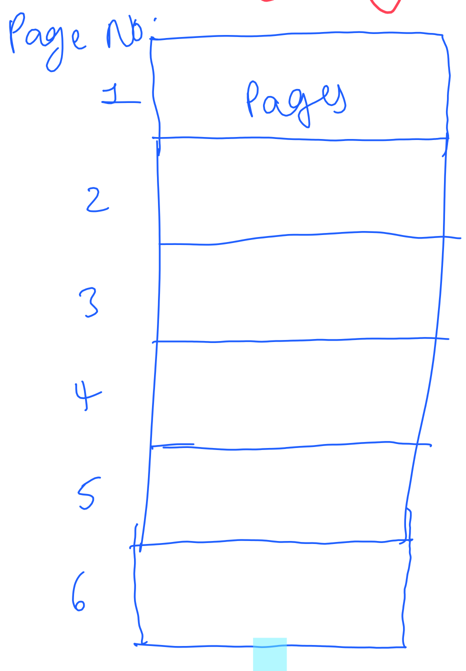
As the processes come and go out of the RAM, eventually some memory gaps can arise that remain unallocated.
This is called **Memory Fragmentation**

Deframentation programs are used to **defragment** these unoccupied memory chunks or **holes**.

Pros:

When the data of the program is fetched, all of the data is fetched at once because it is continuous. This makes the program faster.

# Paging → unit of memory of RAM




At any given moment, not all the features of a program are used. Hence the program's process memory is divided into pages and only the useful pages are loaded into the RAM for execution. This is called **Paging**.

Similarly, the physical memory (volatile + non-volatile) is fragmented into what is known as **Frames**.

The page size and the frame size is generally equal.

From ChatGPT:
In some operating systems and memory management schemes, the size of one page in virtual memory can be the same as the size of one frame in physical memory.
This memory management technique is known as "Fixed-size Page/Frame Mapping" or "Equal-size Page/Frame Mapping."

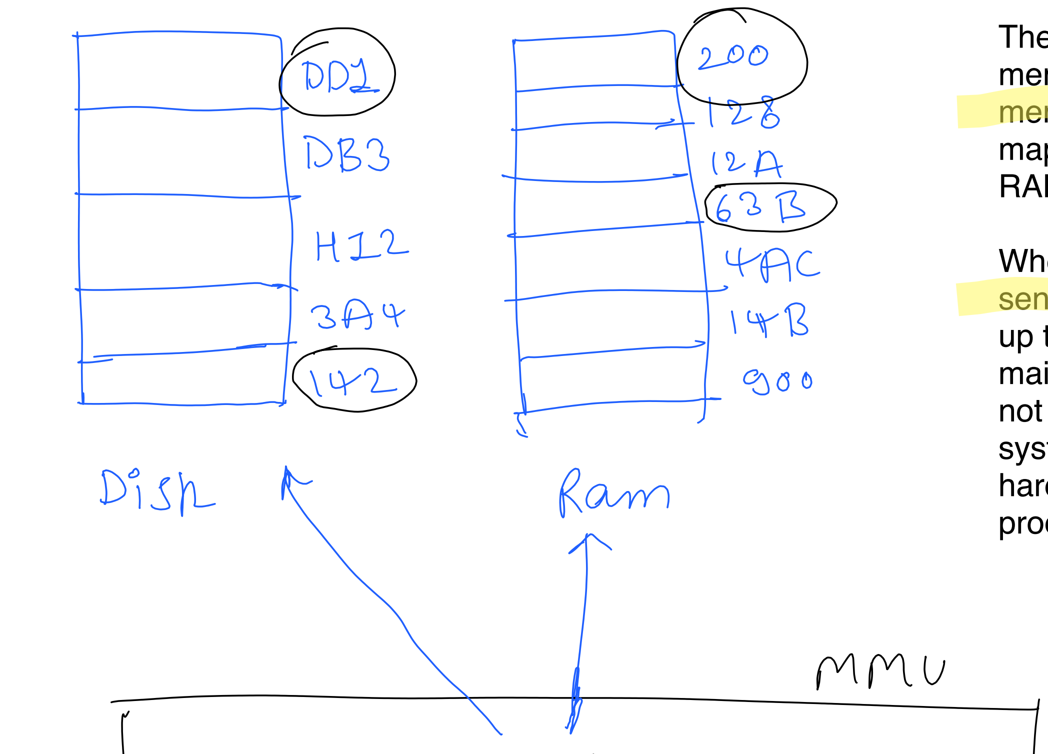
In this approach, the virtual memory space is divided into fixed-size chunks called pages, and the physical memory is divided into corresponding fixed-size chunks called frames. Each page in virtual memory is mapped to a frame in physical memory, and this mapping is maintained by the operating system in a data structure called the "Page Table."

More details on Paging.

Memory Address

Logical (conceptual addr.)

Physical (real addr.)




A Memory Management Unit (MMU) is a hardware component present in most modern computer systems, including CPUs and microprocessors. Its primary function is to handle the virtual memory to physical memory translation, providing an essential part of memory management in the operating system.

The MMU is responsible for translating virtual memory addresses used by processes into physical memory addresses in RAM or other storage devices. This translation is crucial because modern operating systems use virtual memory to enable processes to use more memory than what is physically available in RAM.

The MMU works in conjunction with the operating system's memory management system, which divides the virtual memory into fixed-size units called pages and maintains a mapping between virtual pages and physical frames in RAM or secondary storage.

When a process accesses a memory address, the CPU sends the virtual address to the MMU. The MMU then looks up the corresponding physical address in the page table maintained by the operating system. If the required page is not present in physical memory (a page fault), the operating system retrieves the page from secondary storage (e.g., a hard disk) and updates the page table before allowing the process to access the data.

[https://www.scaler.com/topics/operating-system/virtual-memory-in-os/](https://www.scaler.com/topics/operating-system/virtual-memory-in-os/)

When there is not enough memory (RAM) for a

| | |
|---|---|
| 1 | DD1 |
| 2 | 63B |
| 3 | 200 |
| 4 | 142 |

Page Table

program to load, some pages of unused program in the memory are transferred into the disk and enough space is made for loading the program. This is called Swap Memory and swapping.

# Page Fault

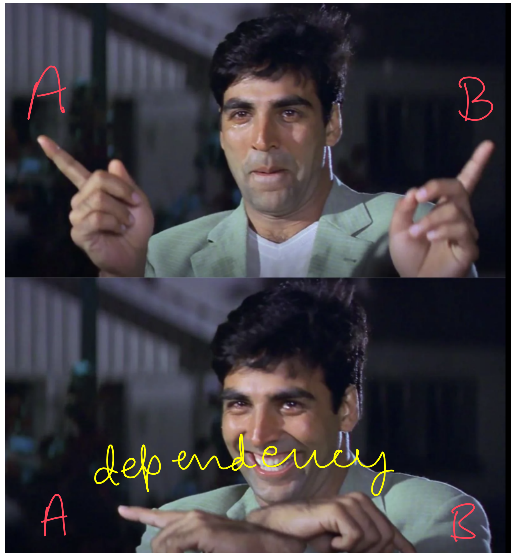
If the process wants to access a page but it is not available in RAM bcos it's in disk.

In order to handle page fault, page replacement algorithms are used.

1) FIFO - First In First Out
2) LRU - Least Recently Used
3) LFU - Least Frequently Used

# DEADLOCK

Deadlock is a situation in concurrent programming where two or more threads are unable to proceed with their execution because each of them is waiting for a resource that is held by another thread. This creates a circular dependency among the threads, and none of them can release the resources they hold, resulting in a state of deadlock.




1. Thread A acquires Resource 1.
2. Thread B acquires Resource 2.
3. Thread A attempts to acquire Resource 2 but must wait for Thread B to release it.
4. Thread B attempts to acquire Resource 1 but must wait for Thread A to release it.

## Overcoming Deadlocks

1) Deadlock Avoidance

2) Deadlock Recovery

3) Deadlock Ignorance

## Deadlock Avoidance

* → Create Resource Allocation Graph
  → Detect a cycle
  → If a cycle is present, then the program is prone to deadlock

* Or set a timeout within a thread to wait for a resource only upto a certain time period.