GROUP 2

- Rozel Galceran
- Joeniño Cainday
- Angela Rose Mendez
- Nomeben Frietz Clarin
- John Aaron Sabio

MEMORY MANAGEMENT

OBJECTIVES

01 Basic concepts of page algorithms in memory management.

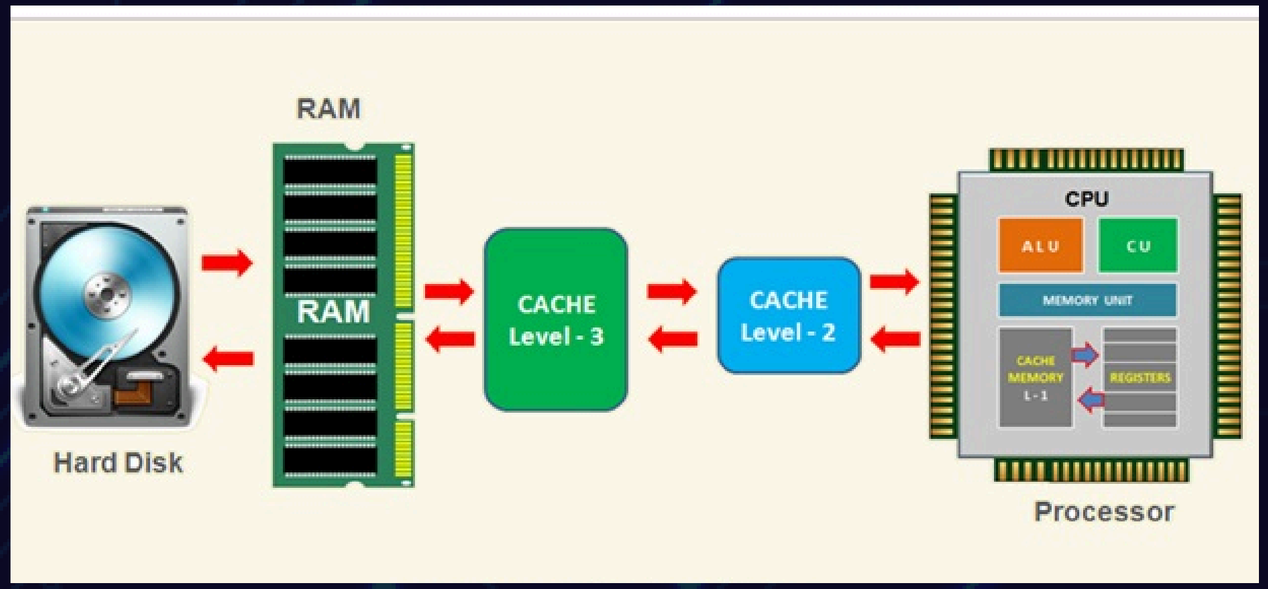
02 Algorithms such as FIFO, LRU, and Page Replacement

03 Analyze the performance with different memory access patterns.

CONCEPTS

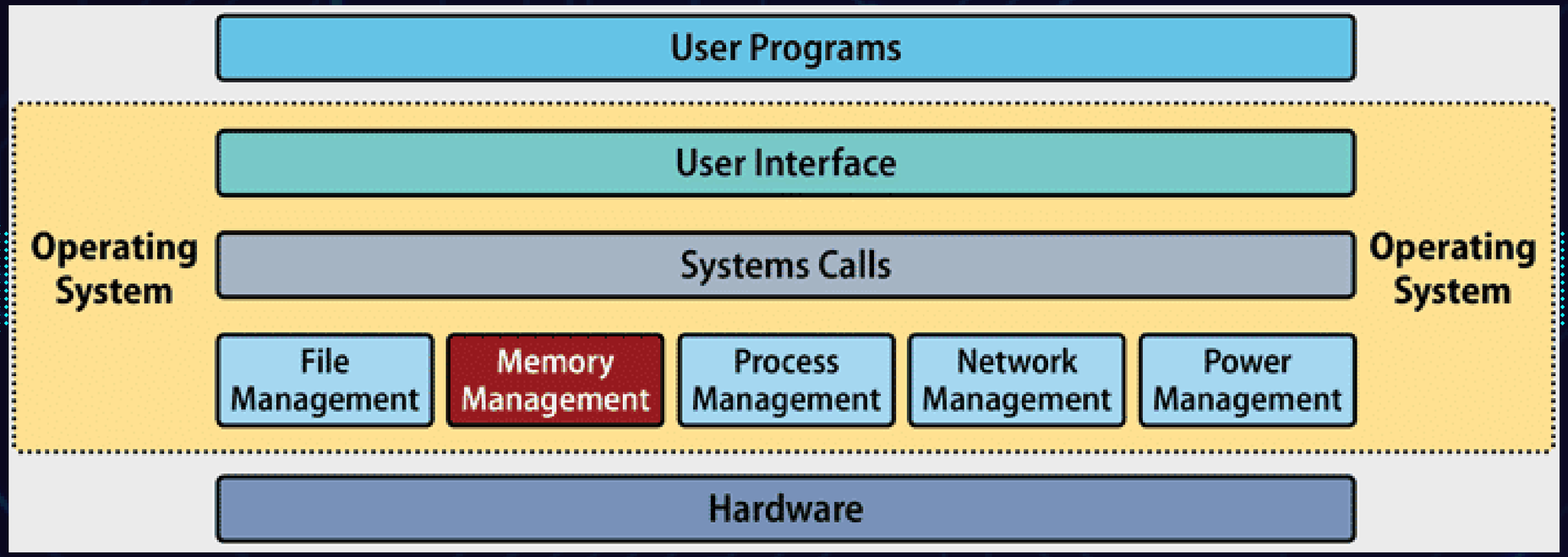
Memory Management

- a task of subdividing the memory among different processes.
- a function in the OS to manage operations between main memory and disk during process execution.
- to achieve efficient utilization of memory.



Memory Management

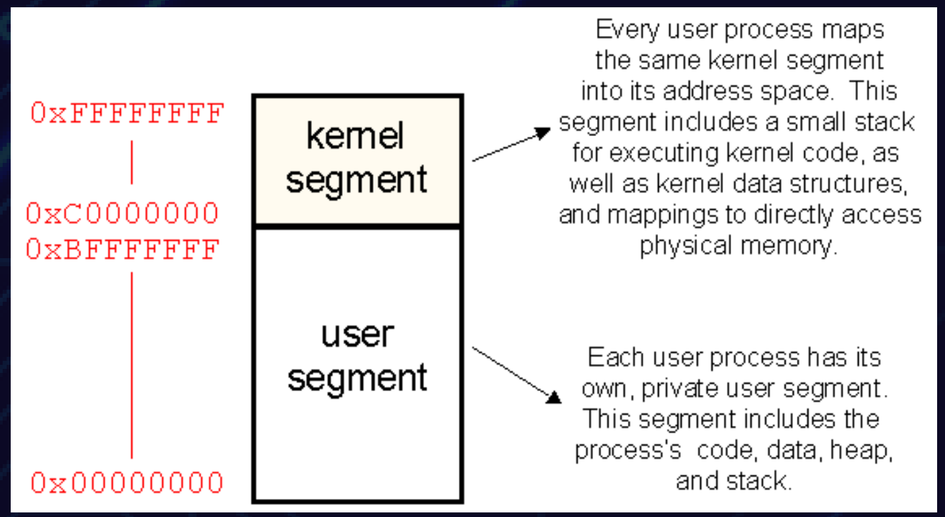
- Keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.
- Checks how much memory is to be allocated to processes.
- Decides which process will get memory at what time.
- It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.



The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program.

Process Address Space

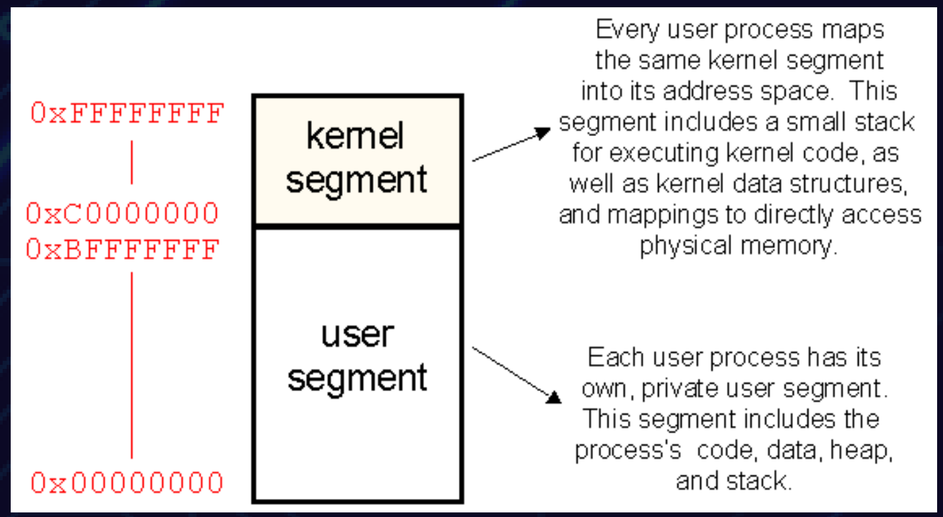
- a set of logical addresses (hex) that a process references in its code.



Process Address Space

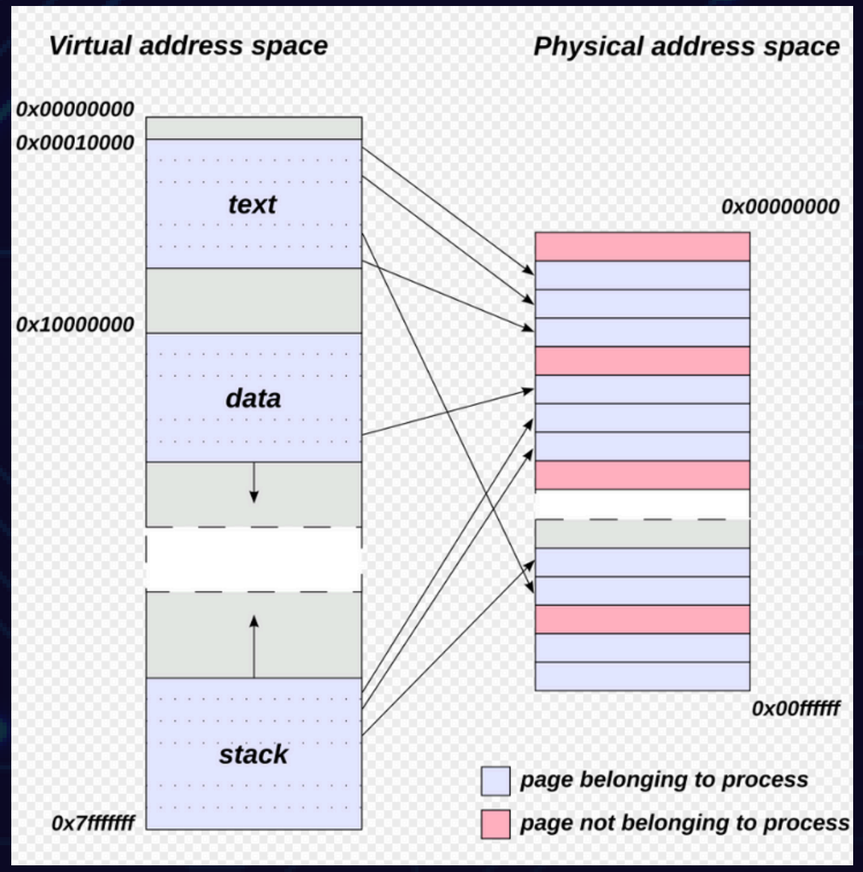
Three types of addresses before and after memory is allocated

- Symbolic Address
  - source code variables
- Relative Address
  - compiler converted symbolic
- Physical Address
  - loader to main memory



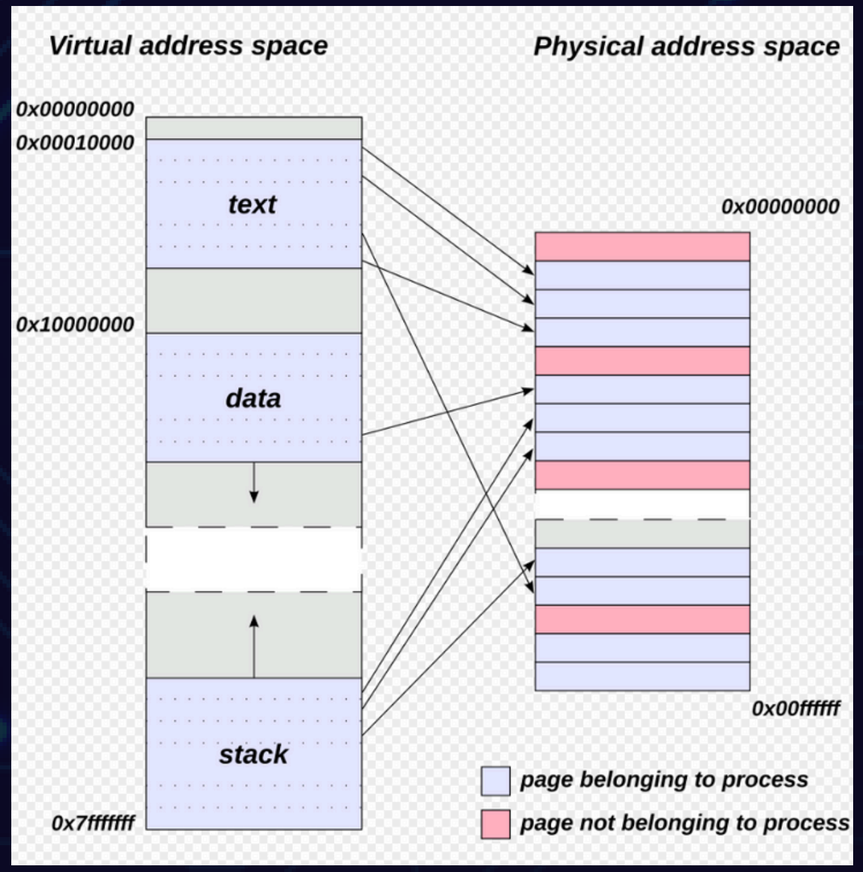
Virtual Address Space (VAS)

- the operating system set ranges of virtual addresses that makes available to a process.
- provides isolation and prevents processes from interfering with each other



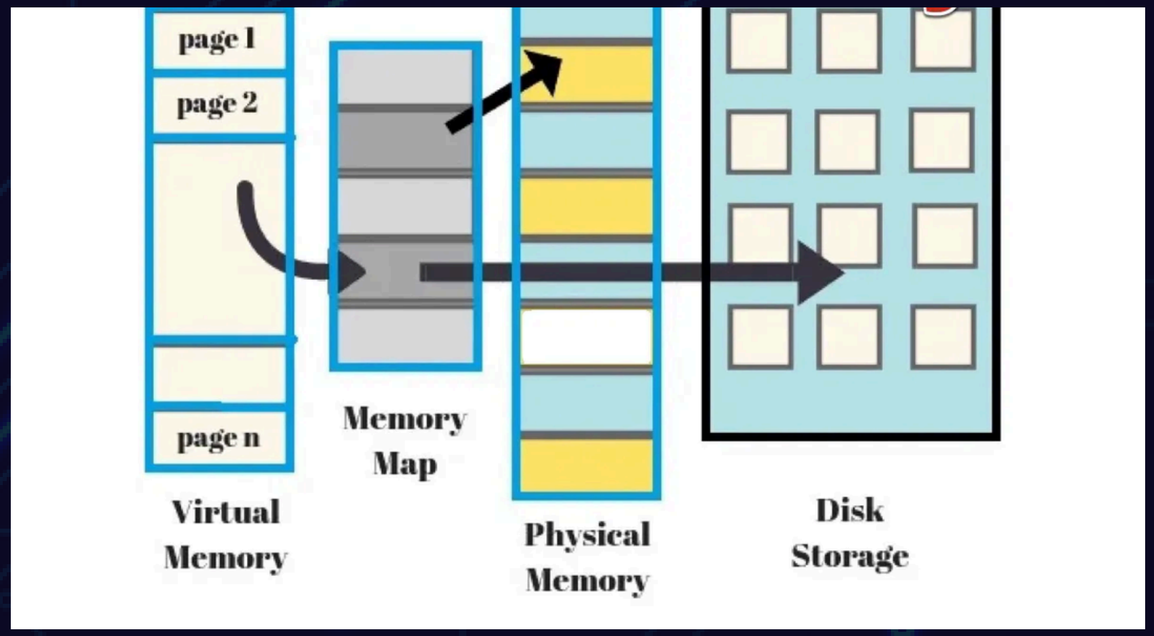
Virtual Address Space (VAS)

- aka logical address if the address generated by the CPU during the execution of a program.
- aka virtual address when the system uses virtual memory.



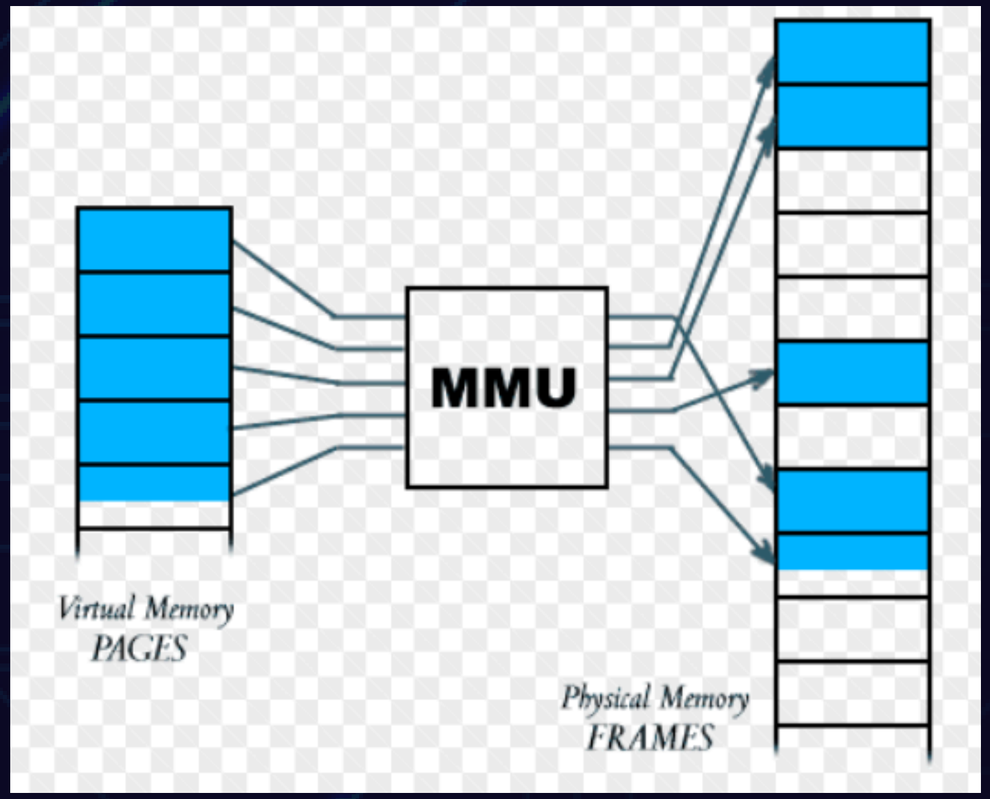
Virtual Memory

- Technique that allows the system to use disk space (secondary storage) as an extension of RAM (primary memory).
- This allows programs to run as if there is more RAM available than physically exists on the system.



Virtual Memory

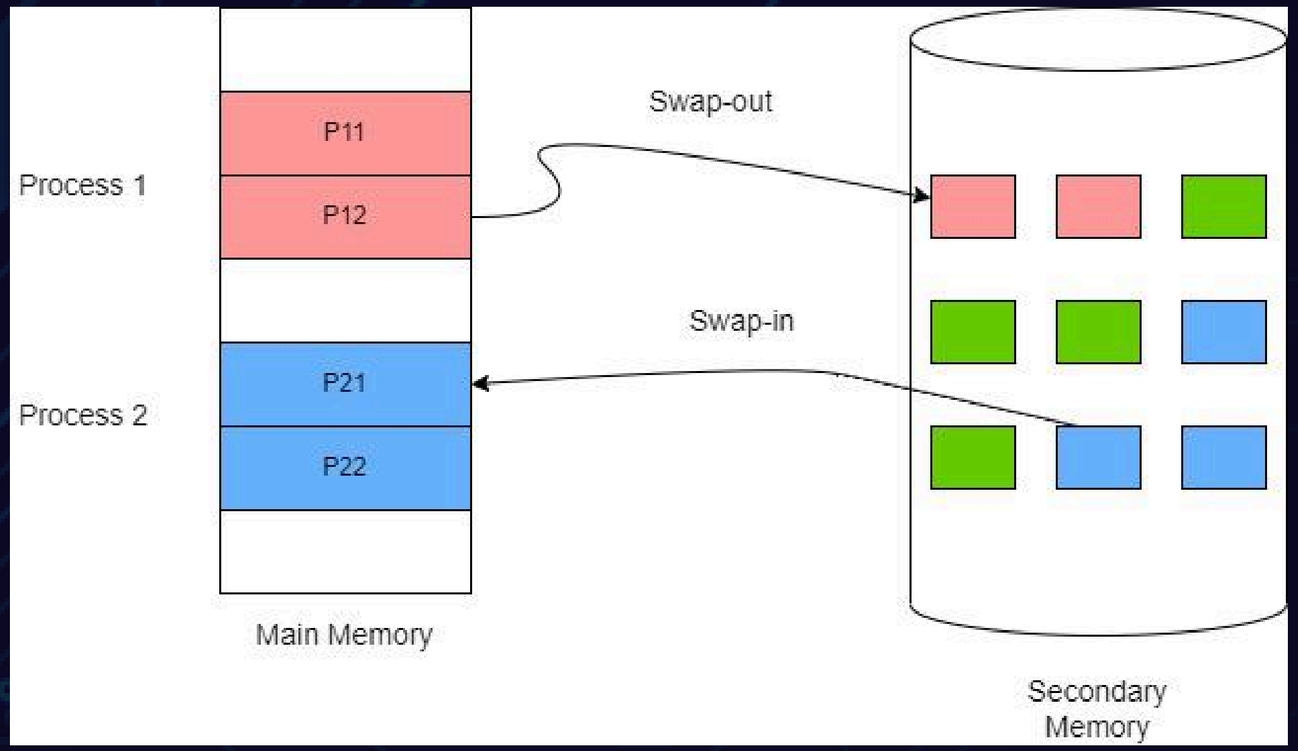
- These addresses are used by a program to access data and instructions during execution.
- However, they do not directly correspond to physical memory locations.
- Logical address is translated into a physical address by the Memory Management Unit (MMU)
- it provides an abstraction, allowing programs to think they have access to a large, continuous block of memory when, in reality, physical memory may be fragmented.



TECHNIQUES

Swapping

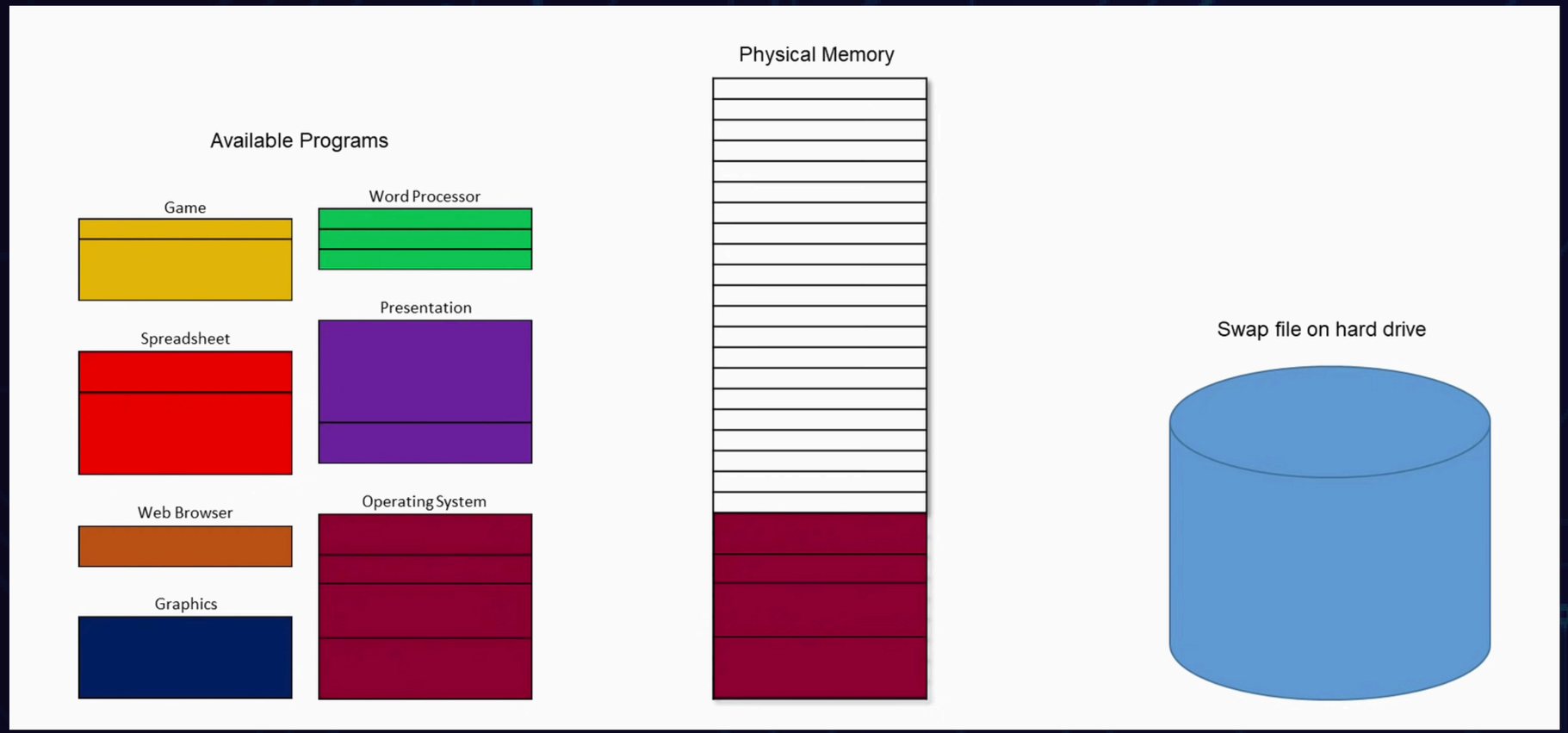
- a mechanism in which a process can be swapped temporarily out of main memory to disk.
- performance is usually affected by swapping process but it helps in running multiple and big processes in parallel.



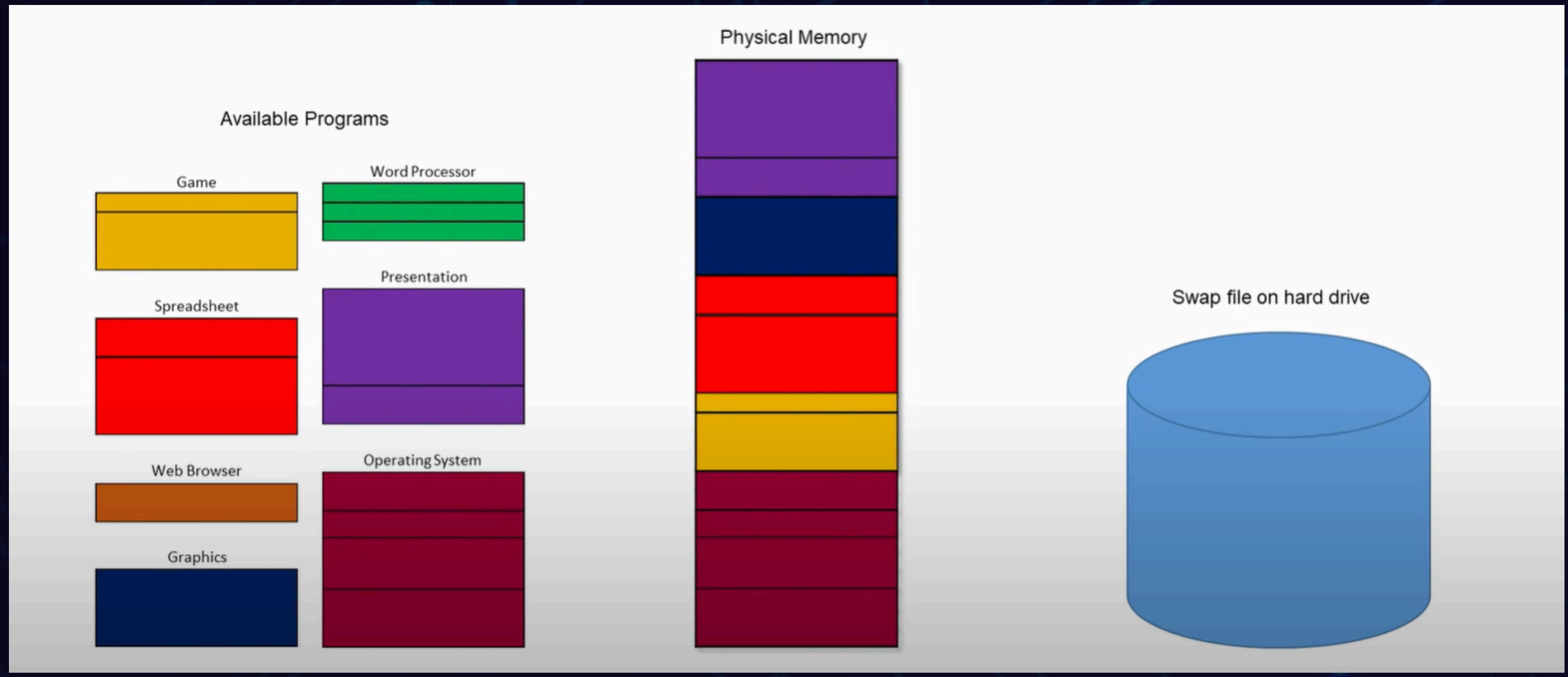
Segmentation

- A process is divided into Segments
- The chunks that a program is divided into which are not necessarily all of the exact sizes are called segments.
- Segmentation gives the user’s view of the process which paging does not provide.
- The user’s view is mapped to physical memory.

Segmentation



Segmentation

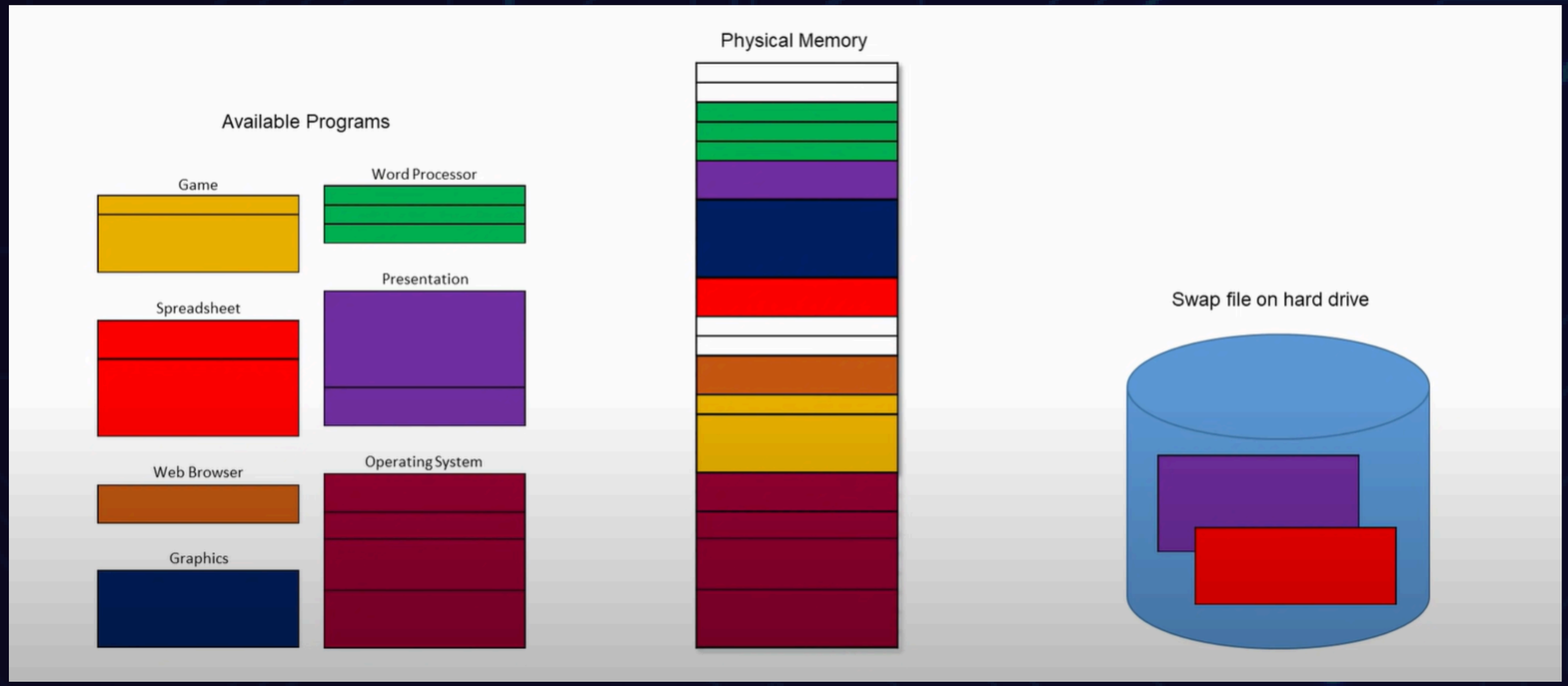


Types of Segmentation

- **Virtual Memory Segmentation:** Each process is divided into a number of segments, but the segmentation is not done all at once. This segmentation may or may not take place at the run time of the program.

- **Simple Segmentation:** Each process is divided into a number of segments, all of which are loaded into memory at run time, though not necessarily contiguously.

Fragmentation

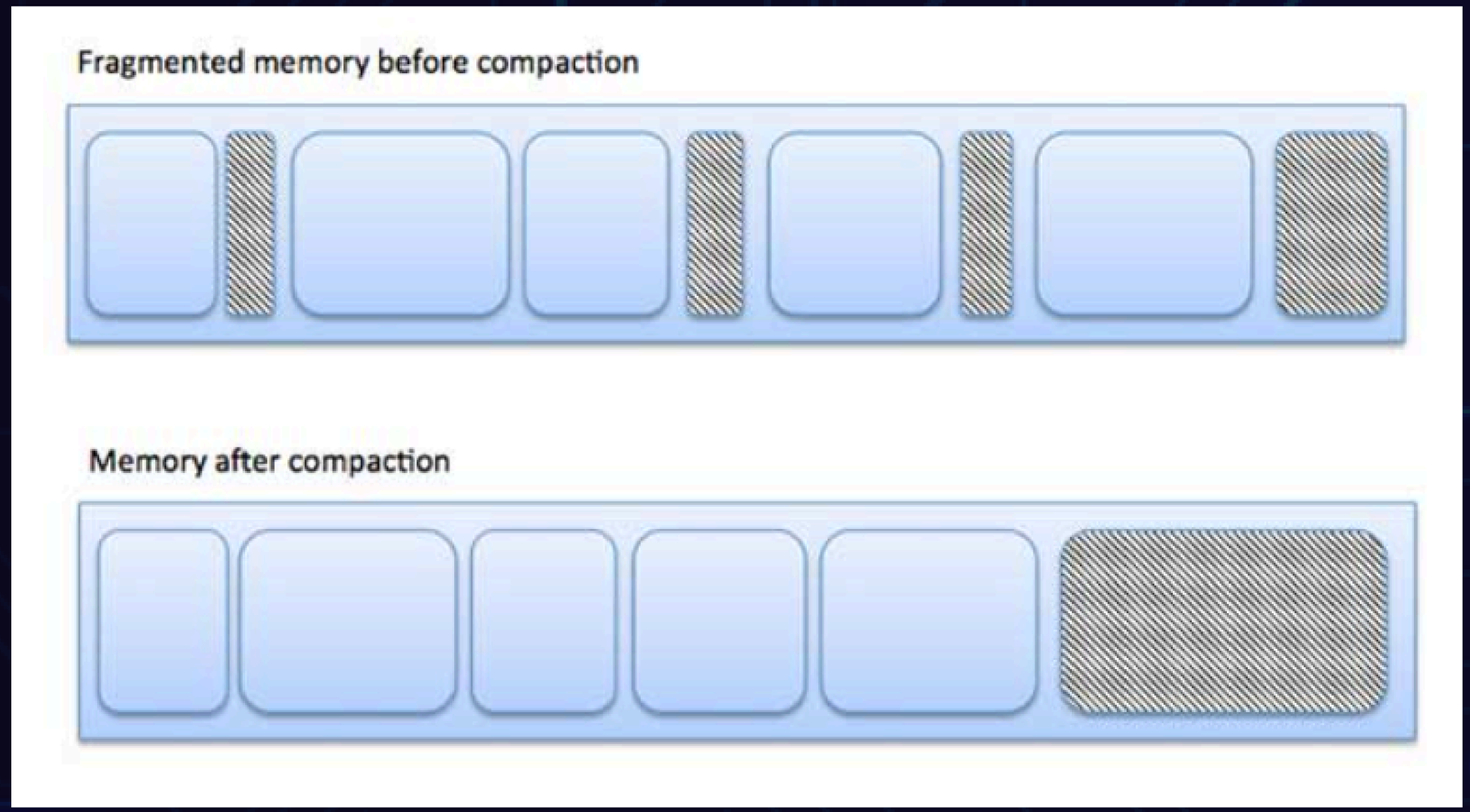


Fragmentation

- The process of dividing a computer file, such as a data file or an executable program file, into fragments that are stored in different parts of a computer's storage medium, such as its hard disc or RAM, is known as fragmentation in computing.

- When a file is fragmented, it is stored on the storage medium in non-contiguous blocks, which means that the blocks are not stored next to each other.

Compaction (after fragmentation)

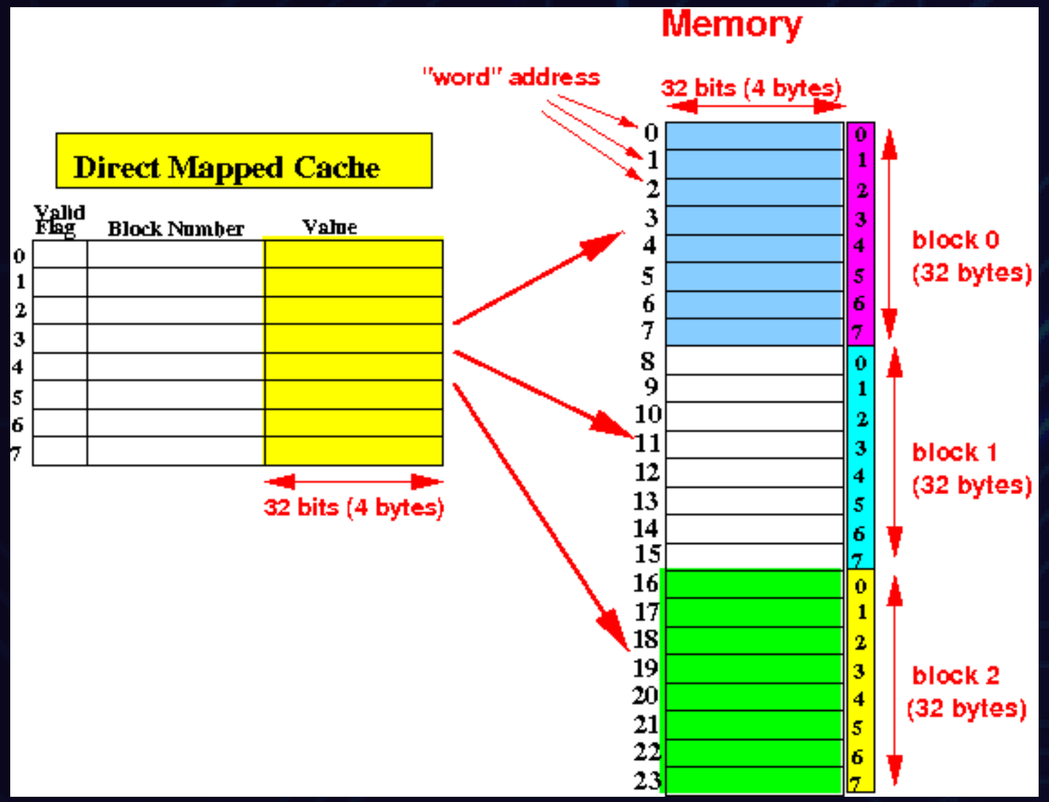


Compaction (after fragmentation)

- a technique to collect all the free memory present in the form of fragments into one large chunk of free memory, which can be used to run other processes.

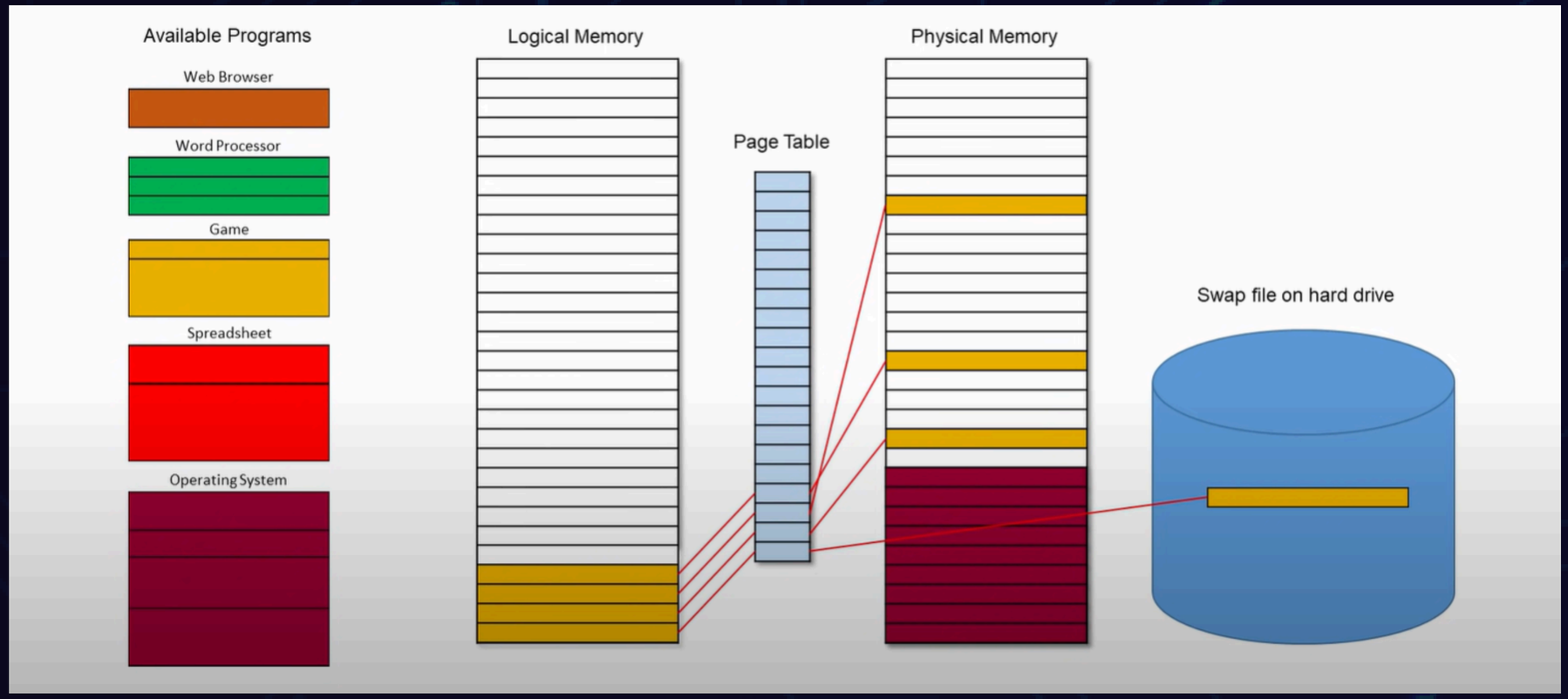
- It does that by moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous.

(Review Cache Lines and Blocks)



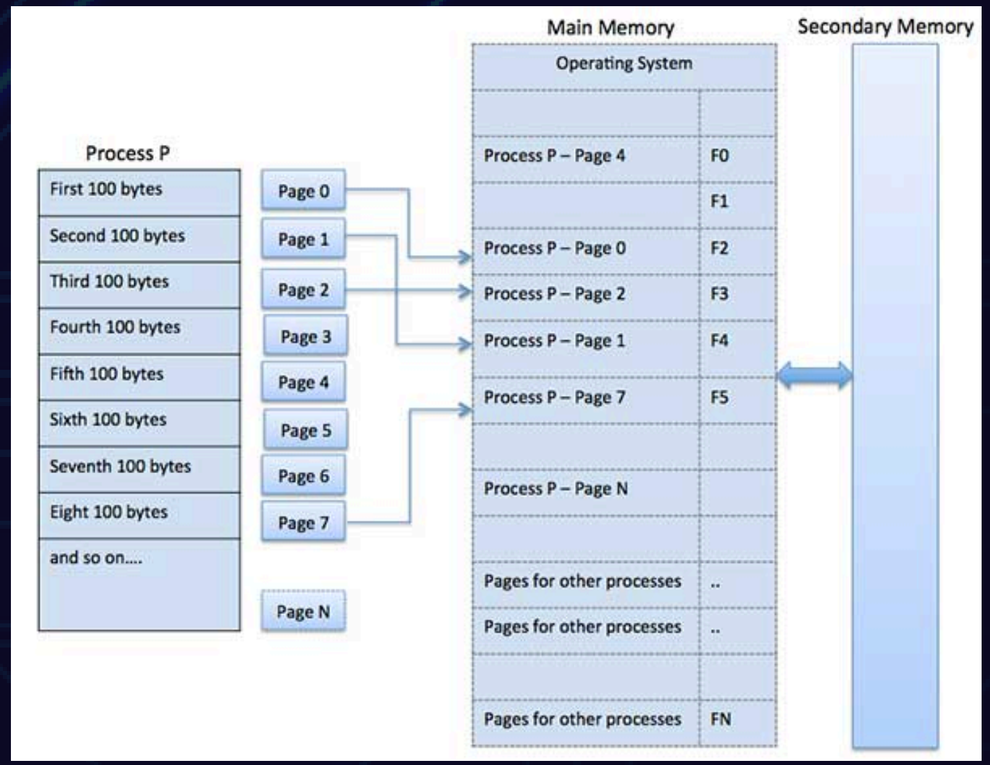
- Instead of using pages, caches operate by transferring data in smaller units called cache lines between the cache and the main memory.

Paging


Available Programs
Web Browser
Word Processor
Game
Spreadsheet
Operating System
Logical Memory
Page Table
Physical Memory
Swap file on hard drive

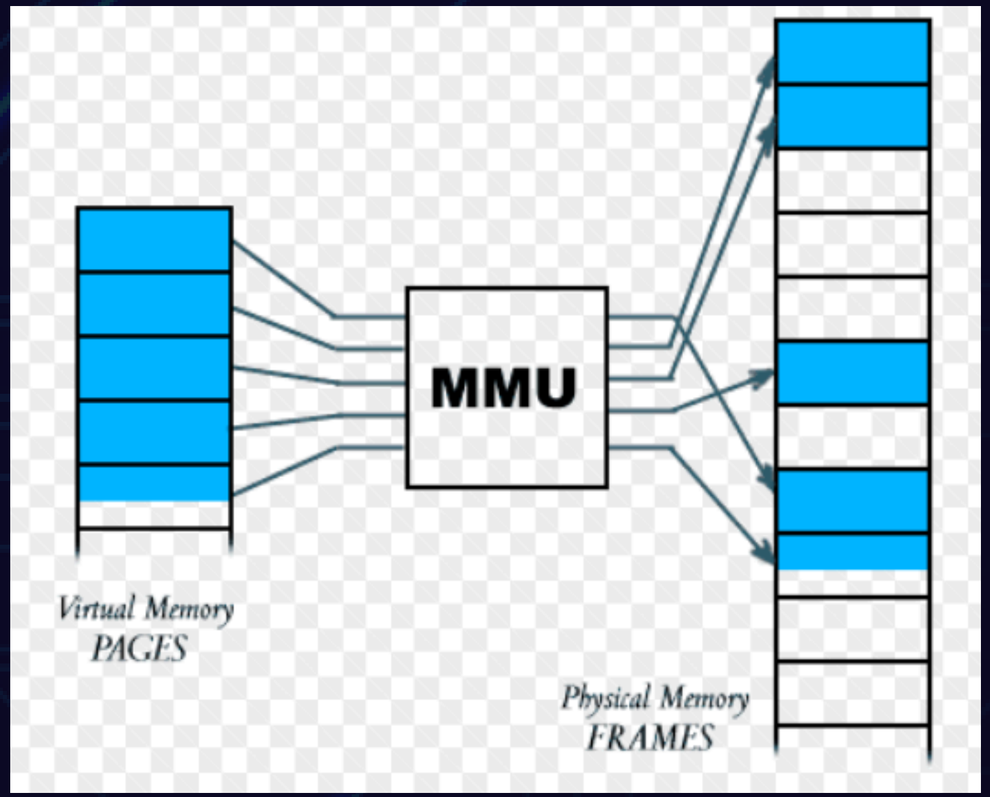
Paging

- solves the problem of segmentation and fragmentation
- **eliminates contiguous allocation** of physical memory.
- retrieving processes the form of pages from the **disk into the memory**
- to **separate each procedure** into pages.



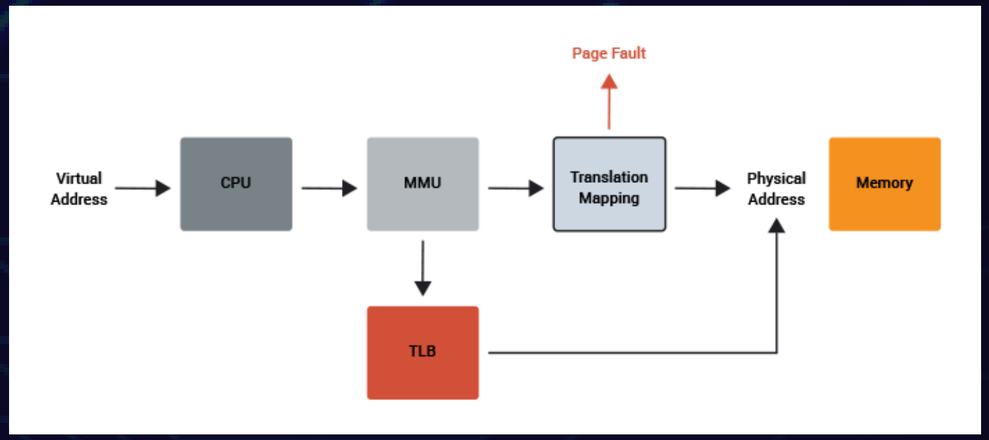
Paging

- Allows memory allocation **to be non-contiguous**
- However, they do **not directly correspond to physical** memory locations.
- Logical address is **translated** into a physical address by the Memory Management Unit (MMU)
- it **provides an abstraction**, allowing programs to think they have access to a large, continuous block of memory when, in reality, physical memory may be fragmented.



Page Fault

- when program accesses a memory page mapped into the virtual address space but not loaded in physical memory
- Since actual physical memory is much smaller than virtual memory, page faults happen.
- The Operating System might have to replace one of the existing pages with the newly needed page.
- similar to *cache miss, if you will..*



IMPLEMENTATION

Paging Algorithms

- Different page replacement algorithms **suggest different ways to decide** which page to replace.
- techniques used in operating systems to manage memory efficiently **when the virtual memory is full**
- Goal for all algorithms is **to reduce the number of page faults.**

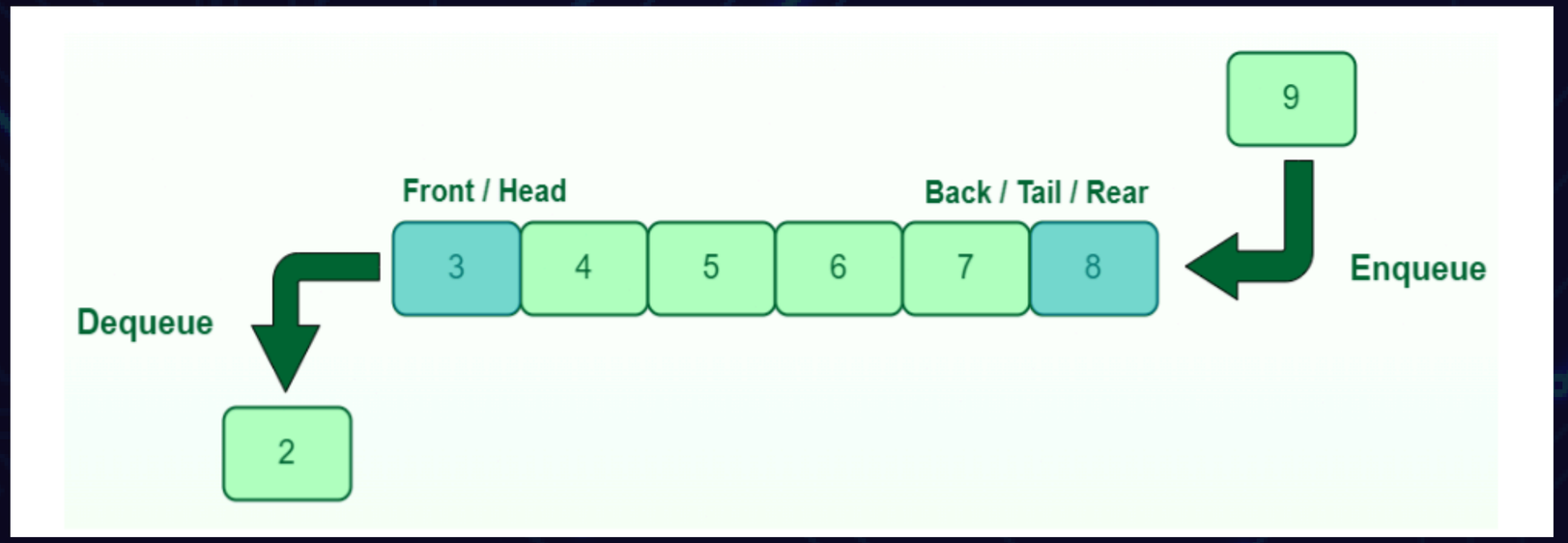
Most Common Paging Algorithms

- First In First Out (FIFO)
- Least Recently Used (LRU)
- Optimal Page replacement
- Most Recently Used (MRU)

FIFO

First-In First-Out (FIFO) Algorithm

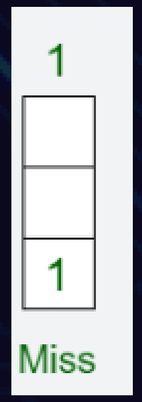
- Queue Data Structure Review



LRU

Least Recently Used (LRU) Algorithm

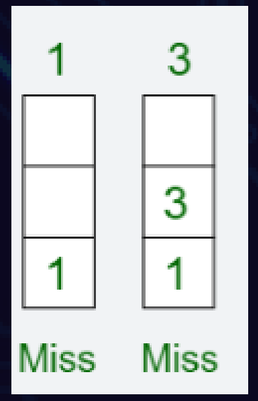
1, 3, 0, 3, 5, 6, 3



Number	Lifetime
x	x
x	x
1	1

Least Recently Used (LRU) Algorithm

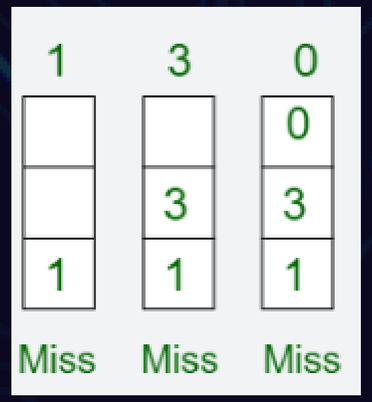
1, 3, 0, 3, 5, 6, 3



Number	Lifetime
x	x
3	1
1	2

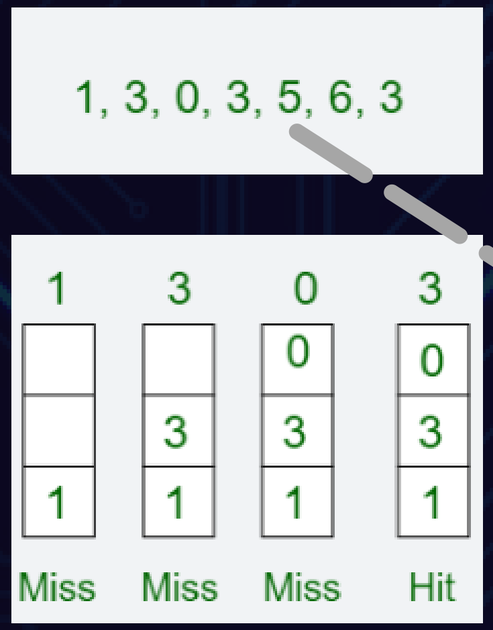
Least Recently Used (LRU) Algorithm

1, 3, 0, 3, 5, 6, 3



Number	Lifetime
0	1
3	2
1	3

Least Recently Used (LRU) Algorithm

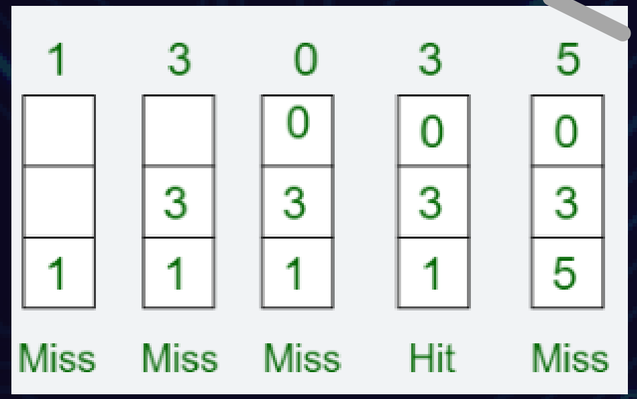


*nothing gets updated**

Number	Lifetime
0	1
3	2
1	3

Least Recently Used (LRU) Algorithm

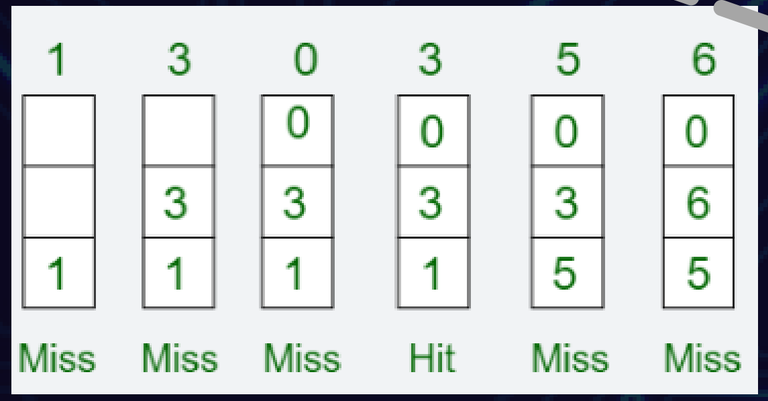
1, 3, 0, 3, 5, 6, 3



Number	Lifetime
0	2
3	3
5	1

Least Recently Used (LRU) Algorithm

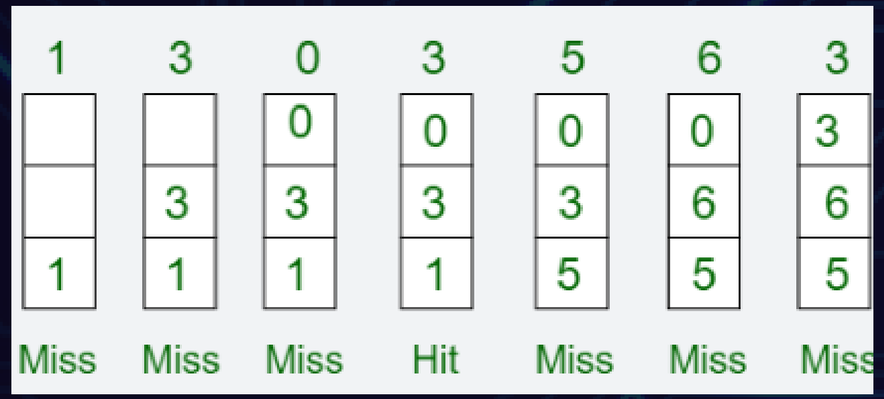
1, 3, 0, 3, 5, 6, 3



Number	Lifetime
0	3
6	1
5	2

Least Recently Used (LRU) Algorithm

1, 3, 0, 3, 5, 6, 3



done!

Number	Lifetime
3	1
6	2
5	3

Function - First In, First Out (FIFO) Algorithm

```python
def fifo_page_replacement(pages: list[int], frame_size: int) -> int:
    memory = []  # List to store pages in memory
    page_faults = 0  # Count of page faults

    for page in pages:
        if page not in memory:
            if len(memory) == frame_size:
                memory.pop(0)  # Remove the oldest page (FIFO)
            memory.append(page)  # Add the new page
            page_faults += 1
```

Function - Least Recently Used (LRU) Algorithm

```python
def lru_page_replacement(pages: list[int], frame_size: int) -> int:
    memory = []  # List to store pages in memory
    page_faults = 0  # To count page faults

    for page in pages:
        if page not in memory:
            if len(memory) == frame_size:
                memory.pop(0)  # Remove the least recently used page
            memory.append(page)  # Add the new page
            page_faults += 1
        else:
            # Refresh the page by moving it to the end
            memory.remove(page)
            memory.append(page)

    return page_faults
```

Example usage

```
# Example usage
frame_size = 3
reference_string = [1,3,0,3,5,6,3]
page_faults = fifo_page_replacement(reference_string, frame_size)
page_faults = lru_page_replacement(reference_string, frame_size)

print(f"Total page faults: {page_faults}")
```

ANALYSIS

Tests

- Run the FIFO and LRU implementations with various reference strings and frame sizes.
- Record the number of page faults for each scenario.

Tasks

- Test with different reference strings (e.g., random, sequential).
- Compare the performance of FIFO and LRU for the same reference strings.

Testing Procedure

```python
# 1. Simple Repeated Pages
pages1 = [1, 1, 1, 1, 1]
frame_sizes1 = [1, 2, 3]

# 2. Sequential Access
pages2 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
frame_sizes2 = [2, 3, 4]

# 3. Alternating Pattern
pages3 = [1, 2, 1, 2, 1, 2, 1, 2]
frame_sizes3 = [1, 2, 3]

# 4. Mixed Pattern
pages4 = [3, 2, 1, 4, 2, 5, 2, 3, 7, 6, 3, 2, 1, 4, 3, 2, 1]
frame_sizes4 = [3, 4, 5]

# 5. Worst-Case for FIFO
pages5 = [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]
frame_sizes5 = [3, 4]
```

Testing Results

Test Case	Frame Size	FIFO Faults	LRU Faults
Case 1	1	1	1
Case 1	2	1	1
Case 1	3	1	1
Case 2	2	9	9
Case 2	3	9	9
Case 2	4	9	9
Case 3	1	8	8
Case 3	2	2	2
Case 3	3	2	2
Case 4	3	15	14
Case 4	4	12	10
Case 4	5	11	9
Case 5	3	9	10
Case 5	4	10	8

Testing Function

```python
# Function to run test cases and show only page faults
def run_test_cases():
    print(f"{'Test Case':<12}{'Frame Size':<12}{'FIFO Faults':<12}{'LRU Faults':<12}")
    print("-" * 48)

    # Test Case 1: Simple Repeated Pages
    for size in frame_sizes1:
        print(f"Case 1    {size:<12}{fifo_page_replacement(pages1, size):<12}{lru_page_replacement(pages1, size):<12}")

    # Test Case 2: Sequential Access
    for size in frame_sizes2:
        print(f"Case 2    {size:<12}{fifo_page_replacement(pages2, size):<12}{lru_page_replacement(pages2, size):<12}")

    # Test Case 3: Alternating Pattern
    for size in frame_sizes3:
        print(f"Case 3    {size:<12}{fifo_page_replacement(pages3, size):<12}{lru_page_replacement(pages3, size):<12}")

    # Test Case 4: Mixed Pattern
    for size in frame_sizes4:
        print(f"Case 4    {size:<12}{fifo_page_replacement(pages4, size):<12}{lru_page_replacement(pages4, size):<12}")

    # Test Case 5: Worst-Case for FIFO
    for size in frame_sizes5:
        print(f"Case 5    {size:<12}{fifo_page_replacement(pages5, size):<12}{lru_page_replacement(pages5, size):<12}")

# Run all test cases
run_test_cases()
```

DISCUSSION

Performance Comparison

FIFO (First-In-First-Out)	LRU (Least Recently Used)
Advantages	**Advantages**
Simple to implement	Adaptive to access patterns
Less computationally expensive	Minimizes page faults in many real-world scenarios
Disadvantages	**Disadvantages**
May evict frequently accessed pages	More complex and requires tracking usage
Not adaptive to access patterns	Slightly higher overhead in memory usage

Performance Comparison

Case 1: Simple Repeated Pages

Frame Size	FIFO Faults	LRU Faults
1	1	1
2	1	1
3	1	1

Analysis:

- Since the page references are all the same (`[1, 1, 1, 1, 1]`), both FIFO and LRU algorithms incur only one page fault when the first page is accessed. Subsequent accesses do not cause any faults because the page is already in memory. Increasing the frame size does not affect the result in this case because only one unique page is being referenced.

Performance Comparison

Case 2: Sequential Access

Frame Size	FIFO Faults	LRU Faults
2	9	9
3	9	9
4	9	9

Analysis:

- The page references are sequential (`[1, 2, 3, 4, 5, 6, 7, 8, 9]`). Each new page accessed causes a fault since there's no repetition and the frame cannot hold all unique pages. Both FIFO and LRU algorithms perform identically in this case. No matter the frame size, every new page results in a fault until the end of the sequence, where the frame size is insufficient to hold all unique pages.

Performance Comparison

Case 3: Alternating Pattern

Frame Size	FIFO Faults	LRU Faults
1	8	8
2	2	2
3	2	2

Analysis:

- The pattern `[1, 2, 1, 2, 1, 2, 1, 2]` alternates between two pages.
    - With a frame size of 1, each switch between pages causes a fault, resulting in 8 faults for both algorithms.
    - With a frame size of 2 or more, the faults drop significantly to just 2, since both pages fit in the frame. Both algorithms perform equally well here as the pattern is simple and predictable.

Performance Comparison

Case 4: Mixed Pattern

Frame Size	FIFO Faults	LRU Faults
3	15	14
4	12	10
5	11	9

Analysis:

- The page references have a mix of sequential and repeated patterns (`[3, 2, 1, 4, 2, 5, 2, 3, 7, 6, 3, 2, 1, 4, 3, 2, 1]`).
  - **With frame size 3:** Both algorithms have many faults, but LRU performs slightly better, as it can better adapt to the access patterns by replacing the least recently used page.
  - **With frame size 4 or 5:** The number of faults decreases for both algorithms as the frame can hold more pages. LRU still performs better because it efficiently manages recent page accesses.

Performance Comparison

Case 5: Worst-Case for FIFO

Frame Size	FIFO Faults	LRU Faults
3	9	10
4	10	8

Analysis:

- The pattern (`[1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5]`) is chosen to demonstrate a scenario where FIFO experiences a high number of faults due to the "Belady's anomaly."
    - **With frame size 3:** FIFO has fewer faults than LRU due to the repetitive nature, where LRU mistakenly removes pages too early.
    - **With frame size 4:** LRU performs better, as it better retains recently used pages. FIFO fails to adapt, causing more faults.

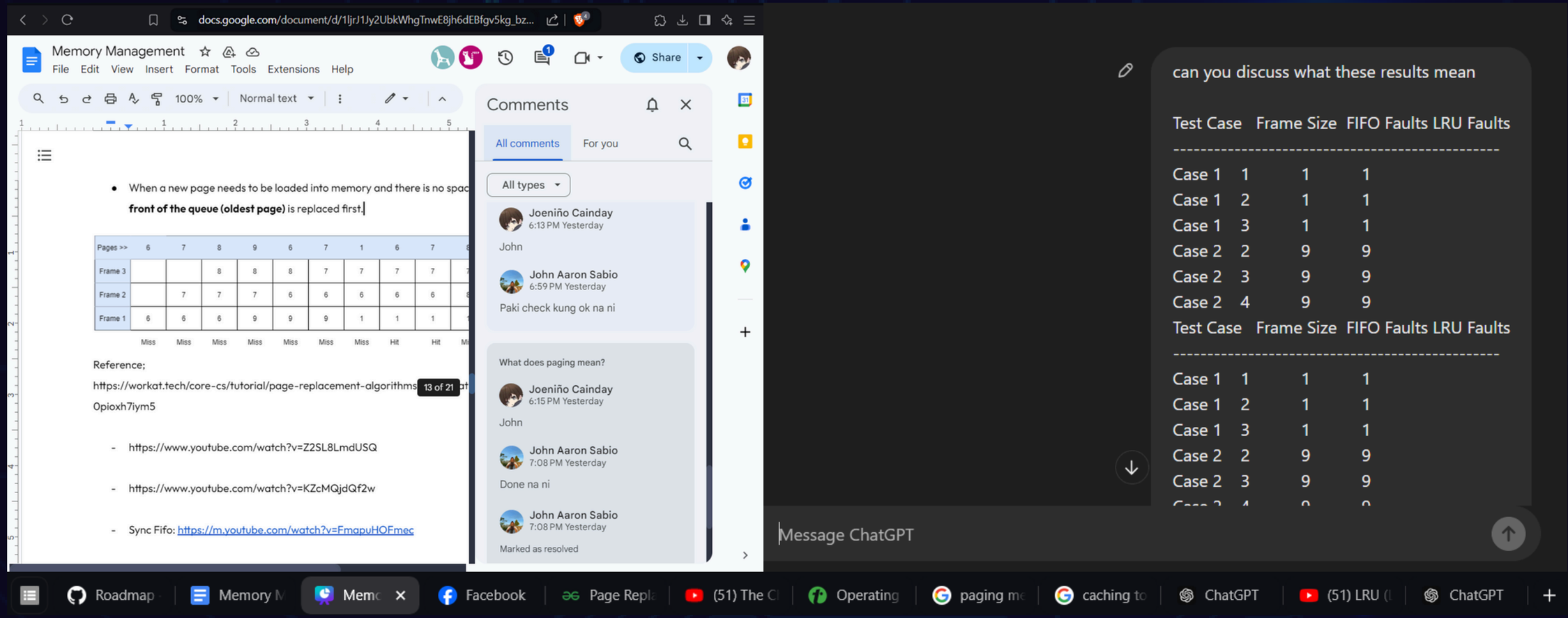
Performance Comparison

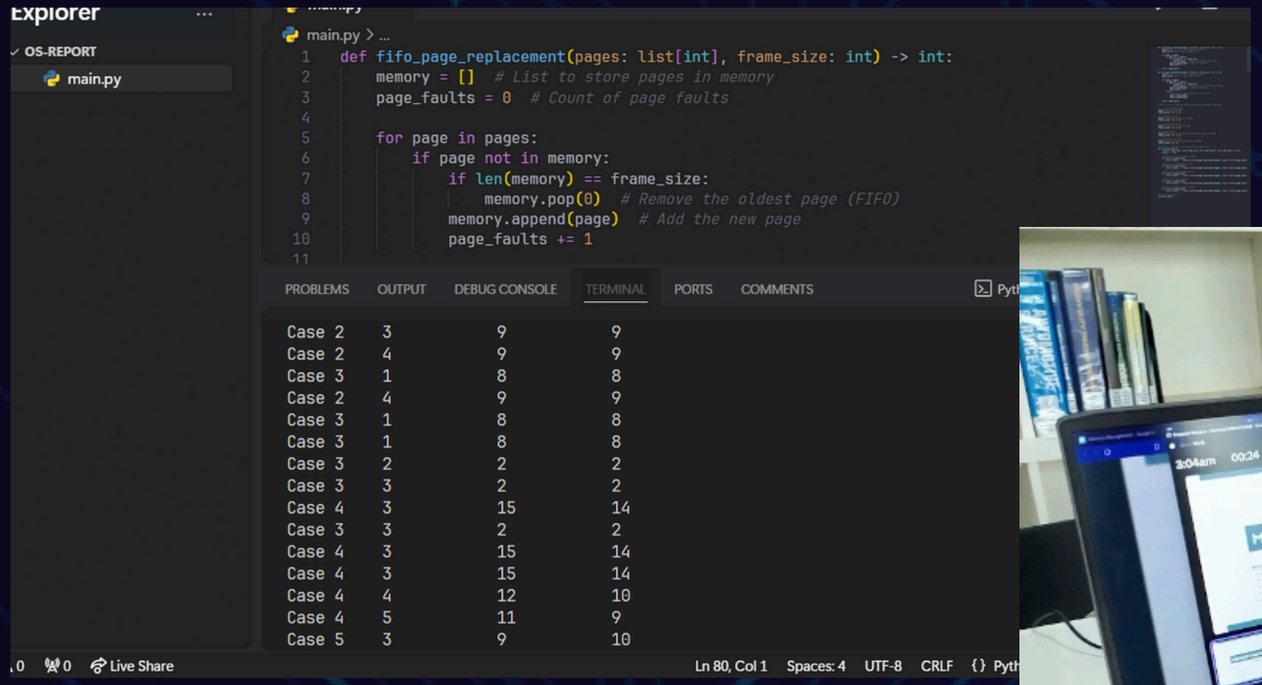
Summary

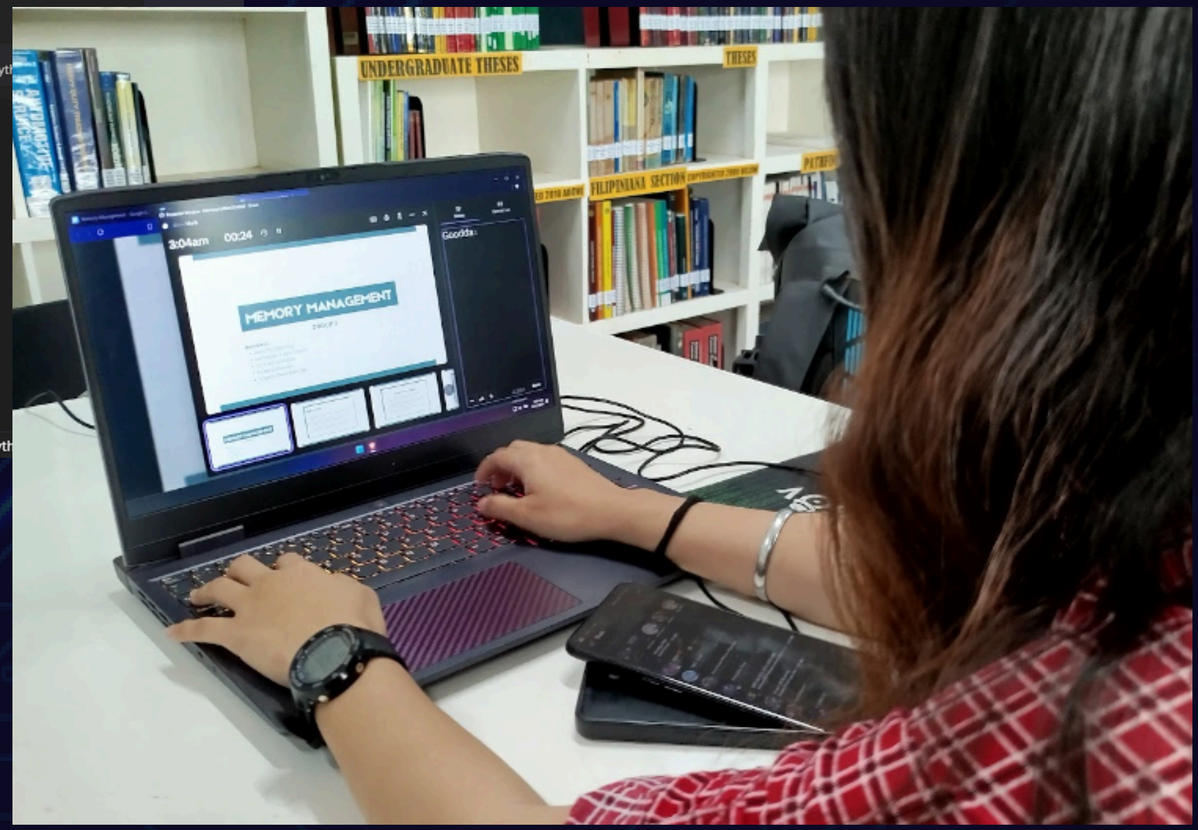
- **Case 1:** Both algorithms perform equally well, as there's no need for replacement.
- **Case 2:** Both algorithms perform poorly due to the inability to hold all unique pages.
- **Case 3:** Both perform equally, with a significant reduction in faults as frame size increases.
- **Case 4:** LRU generally performs better with larger frames, as it efficiently manages pages with more diverse patterns.
- **Case 5:** Shows a scenario where LRU can be advantageous, but sometimes FIFO can perform unexpectedly better due to specific patterns causing "Belady's anomaly."

Overall, the choice between FIFO and LRU should depend on the nature of page access patterns and the available frame size. LRU is generally better with varied patterns, while FIFO is simpler and may perform comparably in certain predictable patterns.

DOCUMENTATION

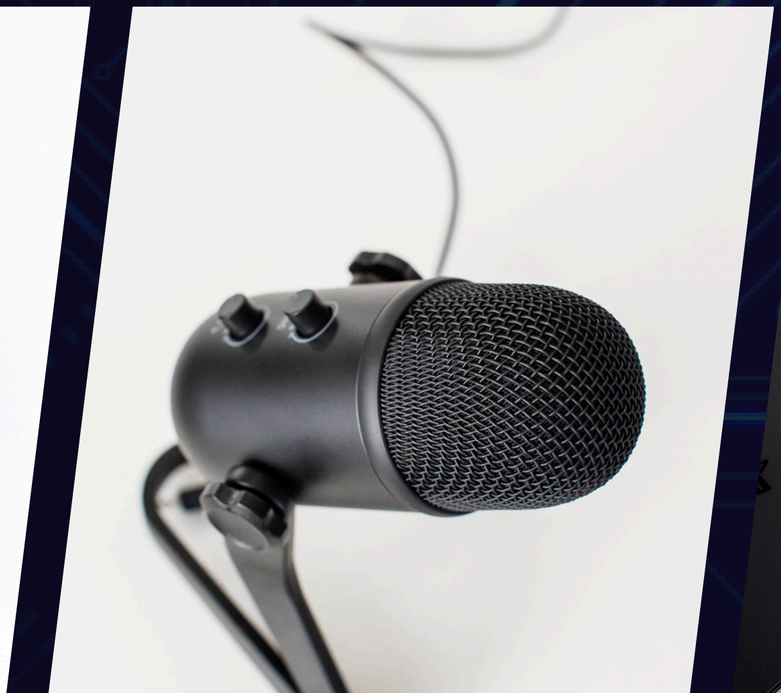

docs.google.com/document/d/1ljrJ1Jy2UbkWhgTnwE8jh6dEBfgv5kg_bz...
Memory Management
File Edit View Insert Format Tools Extensions Help
Share
100%
Normal text
Comments
All comments
For you
All types
Joeniño Cainday
6:13 PM Yesterday
John
John Aaron Sabio
6:59 PM Yesterday
Paki check kung ok na ni
What does paging mean?
Joeniño Cainday
6:15 PM Yesterday
John
John Aaron Sabio
7:08 PM Yesterday
Done na ni
John Aaron Sabio
7:08 PM Yesterday
Marked as resolved
front of the queue (oldest page) is replaced first.
Pages >> 6 7 8 9 6 7 1 6 7
Frame 3
Frame 2
Frame 1
Miss Miss Miss Miss Miss Miss Miss Hit Hit
Reference;
https://workat.tech/core-cs/tutorial/page-replacement-algorithms
13 of 21
0pioxh7iym5
https://www.youtube.com/watch?v=Z2SL8LmdUSQ
https://www.youtube.com/watch?v=KZcMQjdQf2w
Sync Fifo: https://m.youtube.com/watch?v=FmapuHOFmec
can you discuss what these results mean
Test Case Frame Size FIFO Faults LRU Faults
Case 1 1 1 1
Case 1 2 1 1
Case 1 3 1 1
Case 2 2 9 9
Case 2 3 9 9
Case 2 4 9 9
Test Case Frame Size FIFO Faults LRU Faults
Case 1 1 1 1
Case 1 2 1 1
Case 1 3 1 1
Case 2 2 9 9
Case 2 3 9 9
Message ChatGPT
Roadmap
Memory M
Mem
Facebook
Page Repl
(51) The C
Operating
paging me
caching to
ChatGPT
(51) LRU (
ChatGPT


EXPLORER
OS-REPORT
main.py
main.py > ...
def fifo_page_replacement(pages: list[int], frame_size: int) -> int:
memory = [] # List to store pages in memory
page_faults = 0 # Count of page faults
for page in pages:
if page not in memory:
if len(memory) == frame_size:
memory.pop(0) # Remove the oldest page (FIFO)
memory.append(page) # Add the new page
page_faults += 1
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
Case 2 3 9 9
Case 2 4 9 9
Case 3 1 8 8
Case 2 4 9 9
Case 3 1 8 8
Case 3 1 8 8
Case 3 2 2 2
Case 3 3 2 2
Case 4 3 15 14
Case 3 3 2 2
Case 4 3 15 14
Case 4 3 15 14
Case 4 4 12 10
Case 4 5 11 9
Case 5 3 9 10
Live Share
Ln 80, Col 1 Spaces: 4 UTF-8 CRLF


UNDERGRADUATE THESES
THESES
FILIPINIANA SECTION
MEMORY MANAGEMENT

REFERENCES

- tutorialspoint.com/operating_system/os_memory_management.htm
- wikipedia.org/wiki/Memory_management
- wikipedia.org/wiki/Memory_management_(operating_systems)
- courses.cs.washington.edu/courses/cse451/04wi/projects/mmlinux.html
- techtarget.com/searchstorage/definition/address-space
- geeksforgeeks.org/page-replacement-algorithms-in-operating-systems/
- geeksforgeeks.org/lazy-swapping-in-operating-system/

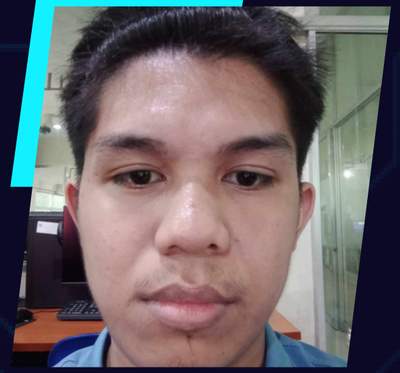




MEET OUR TEAM

Rozel Galceran

Angela Mendez

John Aaron Sabio



Nomeben Clarin

Joeninyo Cainday

THANK YOU FOR LISTENING