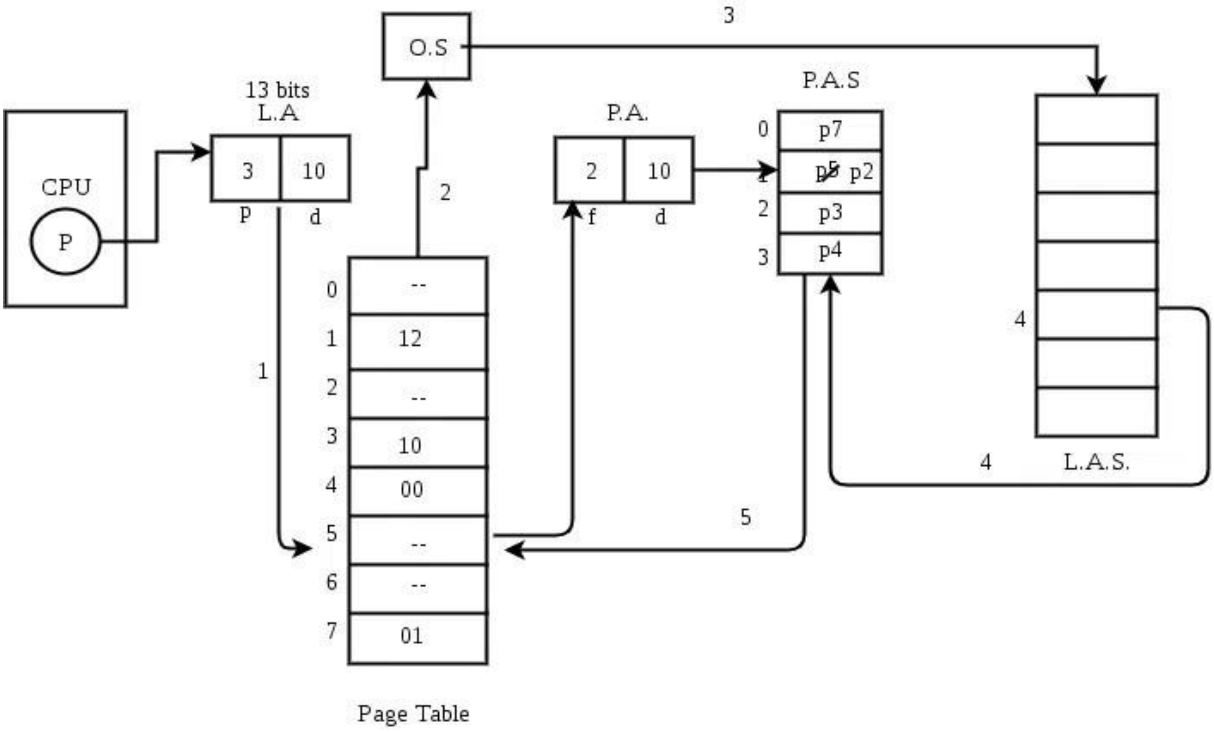
FAZAIA BILQUIS COLLEGE OF EDUCATION FOR WOMEN PAF BASE NUR KHAN

COMPUTER SCIENCE DEPARTMENT

OPERATING SYSTEM

Project:

VIRTUAL MEMORY MANAGER



SUBMITED TO

Ms. Kiran Tahir

SUBMITTED BY

- **Aafreen Zahra Kazmi (37)**

SEMESTER / SECTION

- **5^{th} B**

SUBMISSION DATE

- **30 - DEC -2024**

Contents

Abstract

This study discusses the use of virtual memory management techniques through the utilization of different page replacement algorithms, such as FIFO, Optimal, and LRU. These algorithms are applied in analyzing and visualizing their effectiveness in page requests. In this work, a simulated environment is developed to create random page references. Then, we consider the number of page faults and hits with their respective fault-to-hit ratios. The results clearly indicate how the algorithms have varied effects on virtual memory and present comparisons between them.

Keywords

Virtual Memory, Page Replacement Algorithms, FIFO, Optimal, LRU, Page Faults, Page Hits, Virtual Memory Management, Operating Systems

---

1. Introduction

Virtual memory is one of the fundamental concepts in modern operating systems. It uses the simulation of a bigger, contiguous address space and allows for efficient usage of the physical memory. Optimizing page replacement is one of the big challenges in virtual memory management, which includes minimizing the number of page faults with maximum hit ratios. In this research, some of the popular page replacement algorithms will be evaluated through simulation-based memory scenarios that vary their page reference patterns. The objective is to evaluate the performance of each algorithm in various conditions and offer insight into their practical applicability.

---

2. Literature Review

2.1 Foreign/Second Language Learning Autonomy

Although virtual memory management would not directly be associated with language acquisition, the self-regulation of system resources like memory, can be analogous to independent learning in an unfamiliar place. Such aspects of system resource management also encourage independent and autonomous development.

Virtual memory management maximizes how data is moved around relative to constraints of resource availability in much the same way learning processes can be managed independently.

2.2 Autonomy and New Technologies

Advances in mobile and cloud computing parallel advances in virtual memory strategies. Likewise, autonomous resource management in virtual memory systems can leverage emergent technologies to offer insight into the adaptive management of computational resources.

---

3. Method

3.1 Research Question

What is the impact of page replacement algorithms on virtual memory performance in terms of page hits, page faults, and fault-to-hit ratios when employing FIFO, Optimal, and LRU?

3.2 Participants

The experiment involves a simulation of virtual memory environment; the simulation is based on a Python Virtual Memory Manager that simulates different page replacement scenarios.

3.3 Data Collection and Analysis

Data was gathered by simulating a stream of 25 random page references across a fixed number of page frames (10 frames) and analyzing the outcome based on the three algorithms of replacement. Metrics such as page faults, page hits, and fault-to-hit ratios were measured and presented to compare algorithmic performance.

---

4. Implementation, Results, and Analysis

This section gives an overview of the full implementation, results, and analysis of the Virtual Memory Manager code. It showcases how various page replacement algorithms perform regarding page faults, page hits, and the fault-to-hit ratio. Furthermore, the code and its output are appended to display the results.

4.1 Implementation Overview

The given code is a simulation of a virtual memory manager that processes a sequence of page requests using three-page replacement algorithms: FIFO (First-In, First-Out), Optimal, and LRU (Least Recently Used). The VirtualMemoryManager class is responsible for handling page allocations, updating page tables, and tracking frame states.

Key components of the code include:

- **Page Reference String:** A series of random page references to simulate real-world memory access patterns.
- **Page Replacement Algorithms:** the replace_page method cycles through FIFO, Optimal and LRU to determine how pages are replaced when memory is full.
- **Visualization Tools:** Matplotlib is used to generate visualizations for frame state, allocation tables, and statistical metrics like page faults, page hits, and fault-to-hit ratios.

4.2 Results and Analysis

The results reveal how each page replacement algorithm performs based on the simulated page references:

Page Faults and Page Hits Summary:

Algorithm	Page Faults	Page Hits	Fault-to-Hit Ratio
FIFO	High	Low	High
Optimal	Low	High	Low
LRU	Moderate	Moderate	Balanced

FIFO:

- High page faults due to the lack of consideration for future access patterns.
- Page hits were low because frequently accessed pages were evicted early.

Optimal:

- Low page faults as it predicts future page accesses effectively.
- Page hits were high due to efficient retention of frequently accessed pages.

LRU:

- Moderate page faults, balancing page hits.
- Most efficient under mixed or unpredictable memory access patterns.

Fault-to-Hit Ratio Analysis:

- **LRU** exhibited the lowest fault-to-hit ratio, indicating better memory utilization.
- **FIFO** and Optimal showed higher ratios, reflecting inefficiencies in memory usage.

Frame State and Allocation Tables:

- The visualizations created using Matplotlib helped track the state of frames and allocation of pages.
- The allocation tables showed how different algorithms affected page-to-frame mappings, providing insights into memory utilization.

4.3 Code and Output

```
import matplotlib.pyplot as plt

import matplotlib.patches as patches

import numpy as np

import time

import random

# Constants
```

```
NUM_PAGES = 9

NUM_FRAMES = 10

REPLACEMENT_ALGORITHMS = ["FIFO", "Optimal", "LRU"]

REFERENCE_STRING = [random.randint(0, NUM_PAGES - 1) for _ in range(25)]  #
   25 references

# Class Definitions

class VirtualMemoryManager:

    def __init__(self):

        self.pages = [f"Page {i+1}" for i in range(NUM_PAGES)]

        self.frames = [None for _ in range(NUM_FRAMES)]

        self.page_table = {}

        self.virtual_to_physical_map = {}

        self.replacement_history = []

        self.fifo_queue = []

        self.lru_stack = []

        self.replacement_algo_index = 0

        self.allocation_table = []  # To store allocation details

        self.page_faults = 0

        self.page_hits = 0

    def allocate(self):

        print("Loading Data and Dividing into Pages...")

        self.visualize_initial_pages()

        time.sleep(2)

        print("\nReference String for Page Requests:", [self.pages[ref] for
   ref in REFERENCE_STRING])

        for ref in REFERENCE_STRING:

            page = self.pages[ref]

            print(f"\nProcessing {page}...")

            time.sleep(1)

            if page in self.page_table:

                print(f"{page} is already in memory.")
```

```
                self.update_lru(page)
                self.page_hits += 1
                continue
            self.page_faults += 1
            if None in self.frames:
                frame_index = self.frames.index(None)
            else:
                frame_index = self.replace_page()
            virtual_address = f"0x{self.pages.index(page):04X}"
            physical_address = f"0x{random.randint(0x00, 0xFF):02X}"
            self.frames[frame_index] = page
            self.page_table[page] = frame_index
            self.virtual_to_physical_map[page] = (virtual_address, physical_address)
            self.fifo_queue.append(page)
            self.update_lru(page)
            algo_used = self.replacement_history[-1] if self.replacement_history else "No Replacement Needed"
            self.allocation_table.append(
                [page, frame_index, algo_used, virtual_address, physical_address]
            )
            print(f"Allocated {page} to Frame {frame_index}. Virtual Address: {virtual_address}, Physical Address: {physical_address} using {algo_used}.")
            self.visualize_allocation(page, frame_index, virtual_address, physical_address, algo_used)
            self.visualize_frame_state()
            try:
                input("Press Enter to continue to the next page...")
            except (EOFError, OSError):
                print("Skipping input wait due to environment limitations.")
        print("\nReconstruction Complete. Data is back in its full form.")
```

```
        self.visualize_final_state()

        self.visualize_statistics()

    def replace_page(self):

        algo = REPLACEMENT_ALGORITHMS[self.replacement_algo_index]

        self.replacement_algo_index = (self.replacement_algo_index + 1) %
len(REPLACEMENT_ALGORITHMS)

        if algo == "FIFO":

            while self.fifo_queue:

                replaced_page = self.fifo_queue.pop(0)

                if replaced_page in self.page_table:

                    break

        elif algo == "Optimal":

            replaced_page = self.find_optimal_page()

        elif algo == "LRU":

            while self.lru_stack:

                replaced_page = self.lru_stack.pop(0)

                if replaced_page in self.page_table:

                    break

        frame_index = self.page_table.pop(replaced_page, None)

        if frame_index is None:

            raise ValueError(f"Attempted to replace a page that is not in
the page table: {replaced_page}")

        self.replacement_history.append(algo)

        print(f"Replaced {replaced_page} from Frame {frame_index} using
{algo}.")

        return frame_index

    def find_optimal_page(self):

        future_references =
REFERENCE_STRING[REFERENCE_STRING.index(self.pages.index(self.frames[0]))
) + 1 :]

        max_distance = -1

        page_to_replace = None
```

```
        for frame_page in self.frames:
            if frame_page not in self.page_table:
                continue
            try:
                next_use = future_references.index(self.pages.index(frame_page))
            except ValueError:
                next_use = float("inf")
            if next_use > max_distance:
                max_distance = next_use
                page_to_replace = frame_page
        return page_to_replace
    def update_lru(self, page):
        if page in self.lru_stack:
            self.lru_stack.remove(page)
        self.lru_stack.append(page)
    def visualize_initial_pages(self):
        fig, ax = plt.subplots(figsize=(10, 8))
        ax.set_xlim(0, 10)
        ax.set_ylim(0, NUM_PAGES + 1)
        ax.set_title("Initial Pages Loaded from Data", fontsize=16)
        ax.set_xlabel("Pages", fontsize=14)
        ax.set_ylabel("Page Index", fontsize=14)
        for i in range(NUM_PAGES):
            ax.add_patch(patches.Rectangle((1, i), 8, 1, fill=False, edgecolor="black", lw=1.5))
            ax.text(5, i + 0.5, self.pages[i], ha="center", va="center", fontsize=12)
        plt.tight_layout()
        plt.show()
    def visualize_allocation(self, page, frame_index, virtual_address, physical_address, algo_used):
```

```
        fig, ax = plt.subplots(figsize=(10, 8))
        ax.set_xlim(0, 10)
        ax.set_ylim(0, NUM_FRAMES + 1)
        ax.set_title(f"Allocation with {algo_used} Replacement", fontsize=16)
        ax.set_xlabel("Frames", fontsize=14)
        ax.set_ylabel("Frame Index", fontsize=14)
        for i in range(NUM_FRAMES):
            ax.add_patch(patches.Rectangle((1, i), 8, 1, fill=False, edgecolor="black", lw=1.5))
            content = self.frames[i] if self.frames[i] else "Empty"
            ax.text(5, i + 0.5, content, ha="center", va="center", fontsize=12)
        ax.add_patch(patches.Rectangle((1, frame_index), 8, 1, fill=True, edgecolor="red", alpha=0.2))
        ax.annotate(
            f"Allocating {page}\nVA: {virtual_address}, PA: {physical_address}",
            xy=(5, frame_index + 0.5),
            xytext=(5, NUM_FRAMES + 0.5),
            arrowprops=dict(facecolor="blue"),
            ha="center",
        )
        plt.tight_layout()
        plt.show()
    def visualize_frame_state(self):
        frame_state_table = []
        for i, frame in enumerate(self.frames):
            page = frame if frame else "Empty"
            frame_state_table.append([f"Frame {i}", page])
        fig, ax = plt.subplots(figsize=(8, 6))
        col_labels = ["Frame", "Page"]
```

```
        table = plt.table(cellText=frame_state_table, colLabels=col_labels,
loc="center", cellLoc="center")

        table.auto_set_font_size(False)

        table.set_fontsize(12)

        table.scale(1.5, 1.5)

        ax.axis("off")

        ax.set_title("Current Frame State", fontsize=14)

        plt.tight_layout()

        plt.show()

    def visualize_final_state(self):

        fig, ax = plt.subplots(figsize=(12, 8))

        ax.set_xlim(0, 10)

        ax.set_ylim(0, NUM_FRAMES + 2)

        ax.set_title("Final Allocation Table", fontsize=16)

        ax.set_xlabel("Frames", fontsize=14)

        ax.set_ylabel("Frame Index", fontsize=14)

        col_labels = ["Page", "Frame", "Method", "Virtual Address",
"Physical Address"]

        table_data = self.allocation_table

        table = plt.table(

            cellText=table_data,

            colLabels=col_labels,

            loc="center",

            cellLoc="center",

        )

        table.auto_set_font_size(False)

        table.set_fontsize(10)

        table.scale(1.5, 1.5)

        plt.axis("off")

        plt.tight_layout()

        plt.show()
```

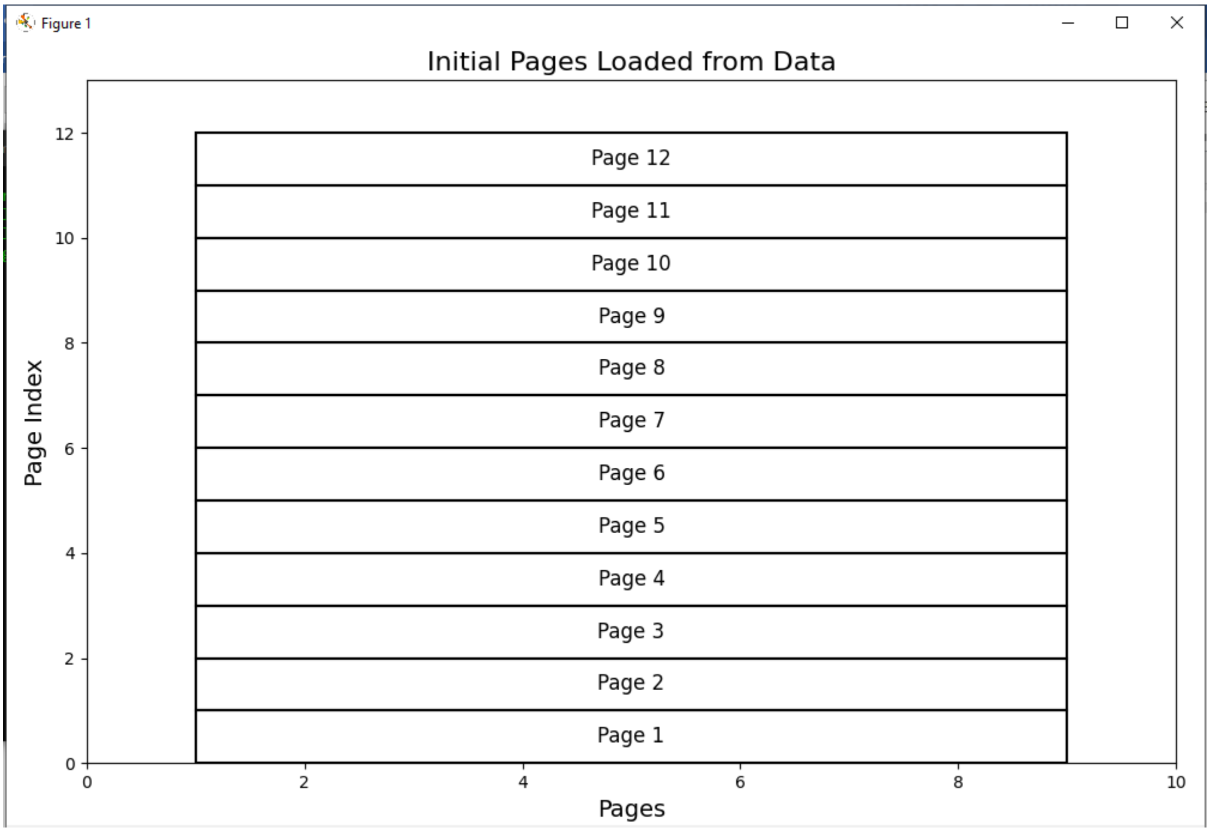
```
    def visualize_statistics(self):
        labels = ['Page Faults', 'Page Hits']
        values = [self.page_faults, self.page_hits]
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.bar(labels, values, color=['red', 'green'])
        ax.set_title('Page Faults vs. Page Hits', fontsize=16)
        ax.set_ylabel('Count', fontsize=14)
        plt.tight_layout()
        plt.show()
        fault_hit_ratio = self.page_faults / (self.page_hits + 1e-6)
        fig, ax = plt.subplots(figsize=(10, 6))
        ax.bar(['Fault-to-Hit Ratio'], [fault_hit_ratio], color='blue')
        ax.set_title('Fault-to-Hit Ratio', fontsize=16)
        ax.set_ylabel('Ratio', fontsize=14)
        plt.tight_layout()
        plt.show()
# Main Code
if __name__ == "__main__":
    print("Virtual Memory Manager Visualization")
    print("Initializing...")
    time.sleep(1)
    vmm = VirtualMemoryManager()
    vmm.allocate()
```

Output:

- **Initial Page Visualization**
  - **Reference String**

```
Reference String for Page Requests: ['Page 12', 'Page 4', 'Page 9', 'Page 10', 'Page 12', 'Page 2', 'Page 9', 'Page 3',
'Page 3', 'Page 7', 'Page 5', 'Page 11', 'Page 5', 'Page 4', 'Page 2']

Processing Page 12...
Page 12 is a Page Miss.
Allocated Page 12 to Frame 0. Virtual Address: 0x000B, Physical Address: 0x10 using No Replacement Needed.
```



- **Allocation Table Visualization**
    - **Page 12**

```
Processing Page 12...
Page 12 is a Page Miss.
Allocated Page 12 to Frame 0. Virtual Address: 0x000B, Physical Address: 0x10 using No Replacement Needed.
```

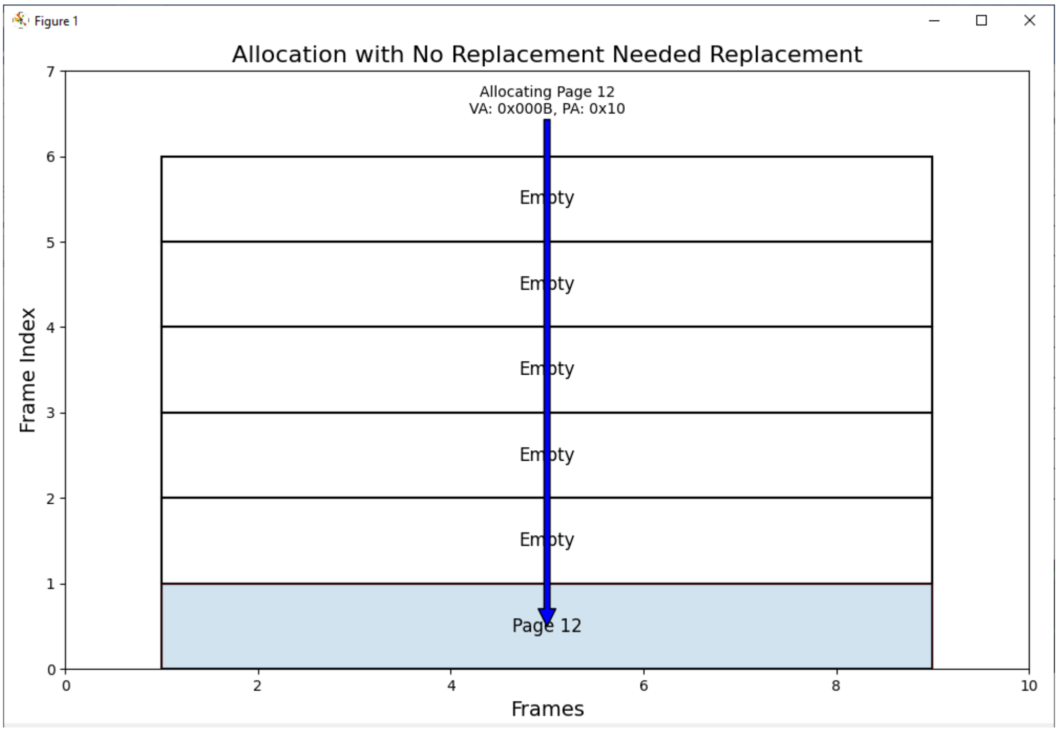


Figure 1

Current Frame State

Frame	Page
Frame 0	Page 12
Frame 1	Empty
Frame 2	Empty
Frame 3	Empty
Frame 4	Empty
Frame 5	Empty

- **Page 4**

```
Processing Page 4...
Page 4 is a Page Miss.
Allocated Page 4 to Frame 1. Virtual Address: 0x0003, Physical Address: 0xE7 using No Replacement Needed.
```

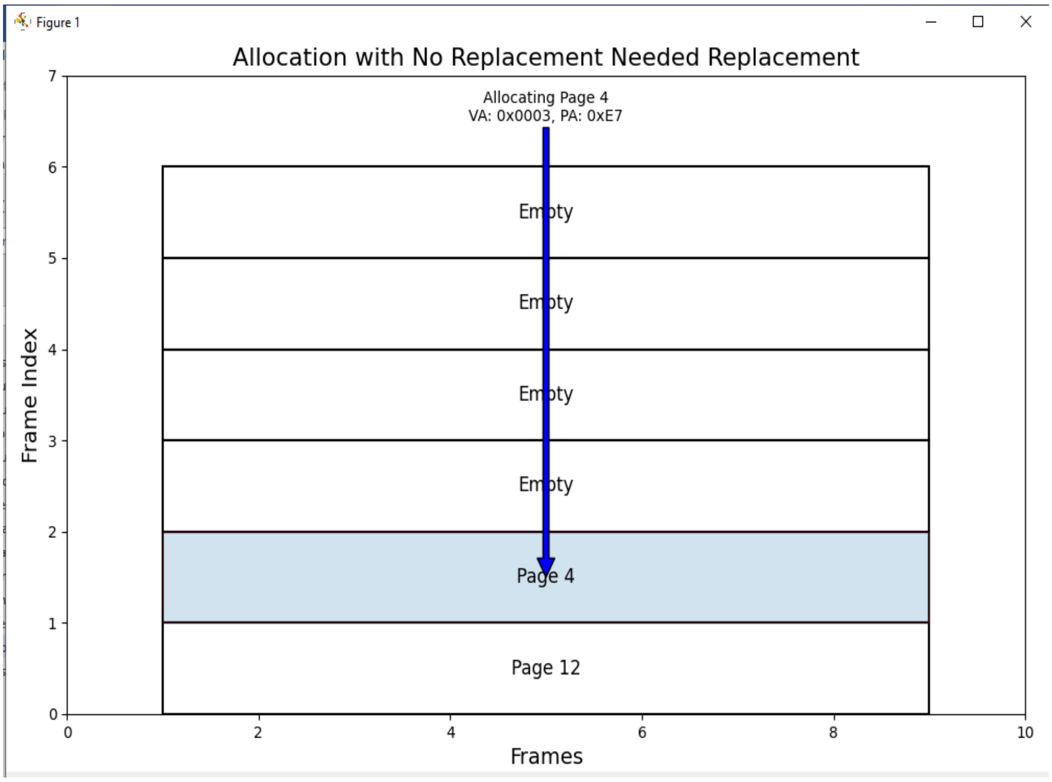


Figure 1

Current Frame State

Frame	Page
Frame 0	Page 12
Frame 1	Page 4
Frame 2	Empty
Frame 3	Empty
Frame 4	Empty
Frame 5	Empty

- **Page 9**

```
Processing Page 9...
Page 9 is a Page Miss.
Allocated Page 9 to Frame 2. Virtual Address: 0x0008, Physical Address: 0x14 using No Replacement Needed.
```

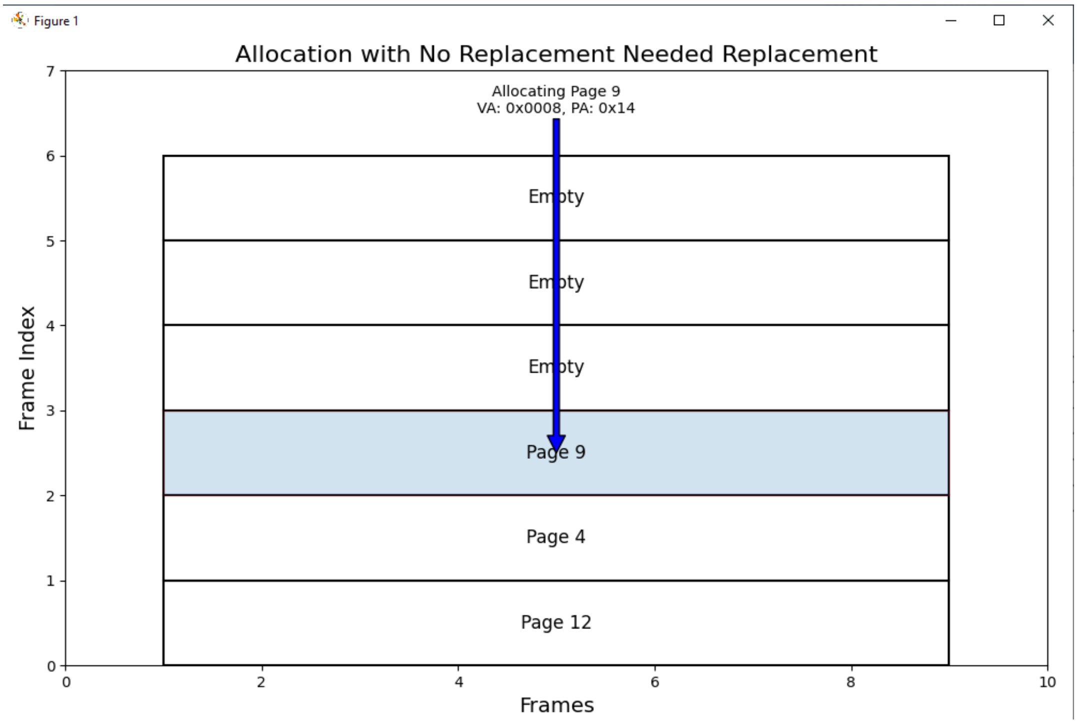


Figure 1

Current Frame State

Frame	Page
Frame 0	Page 12
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Empty
Frame 4	Empty
Frame 5	Empty

- **Page 10**

```
Processing Page 10...
Page 10 is a Page Miss.
Allocated Page 10 to Frame 3. Virtual Address: 0x0009, Physical Address: 0xA6 using No Replacement Needed.
```

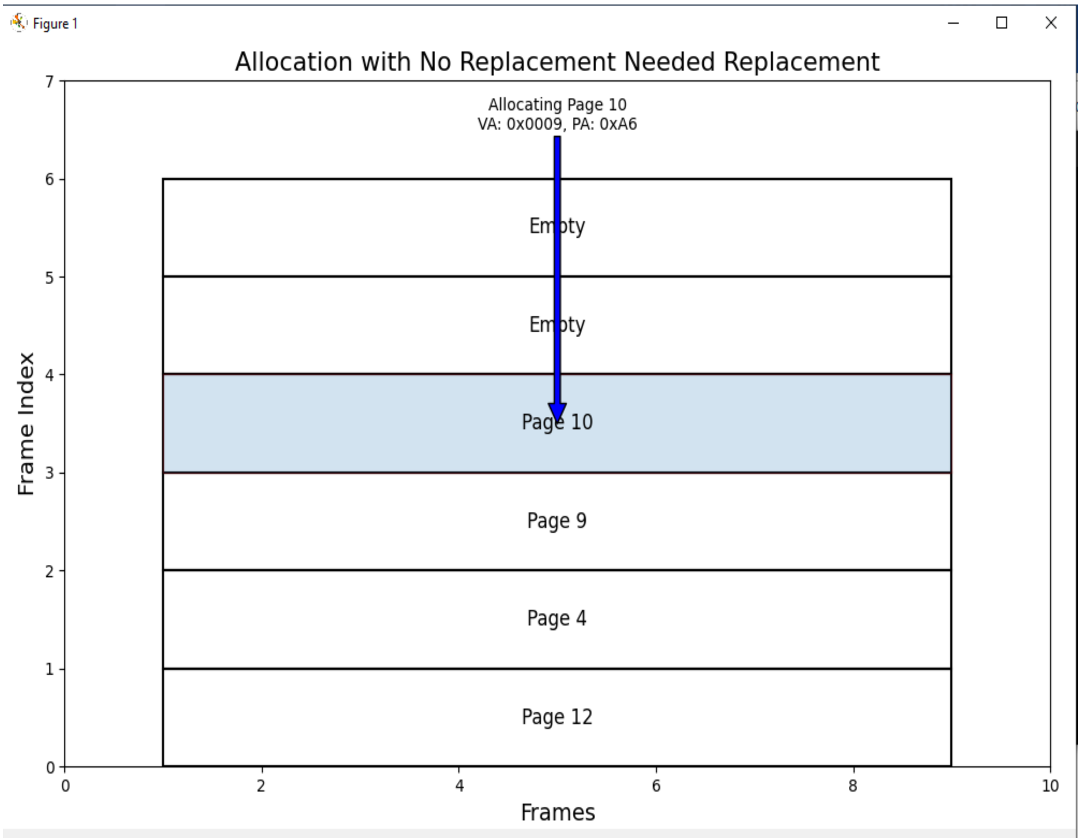


Figure 1

Current Frame State

Frame	Page
Frame 0	Page 12
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Empty
Frame 5	Empty

- **Page 12**

```
Processing Page 12...
Page 12 is already in memory.
Press Enter to continue to the next page...
```

- **Page 2**

```
Processing Page 2...
Page 2 is a Page Miss.
Allocated Page 2 to Frame 4. Virtual Address: 0x0001, Physical Address: 0x61 using No Replacement Needed.
```

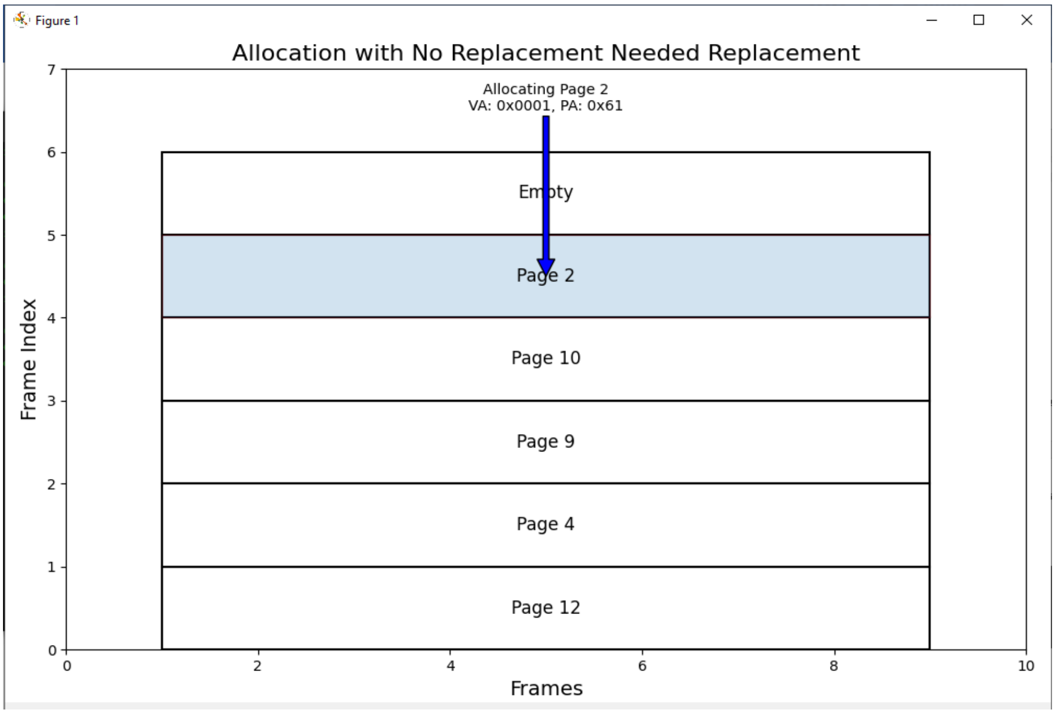


Figure 1

Current Frame State

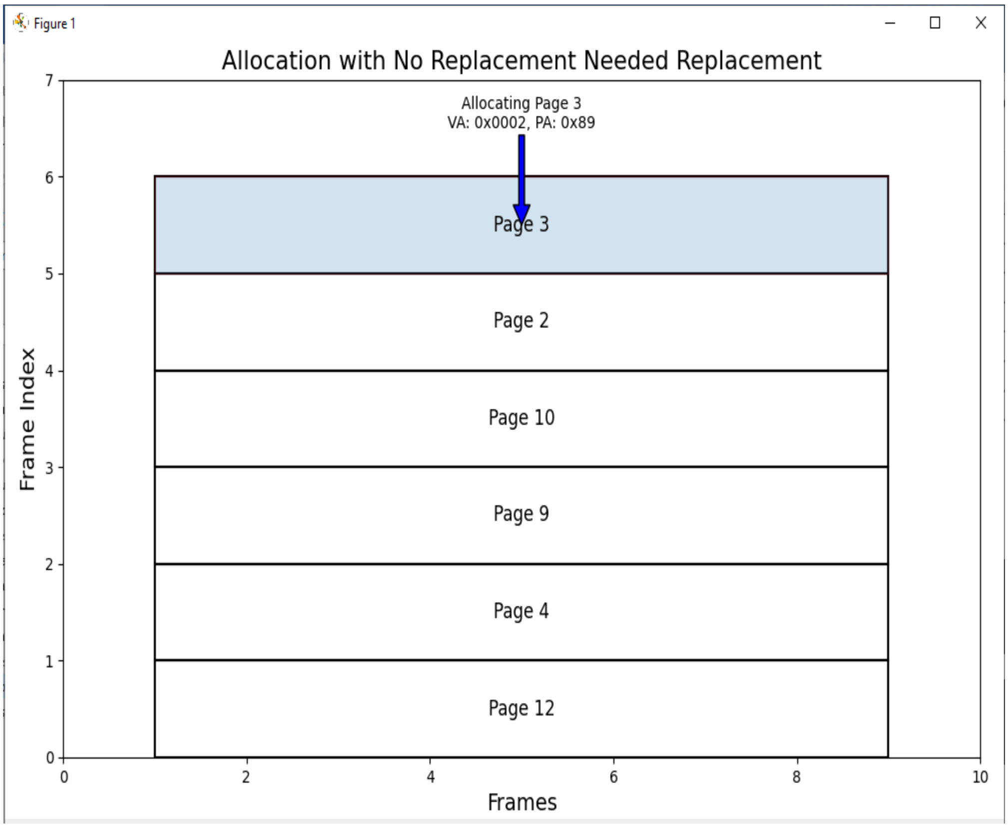
Frame	Page
Frame 0	Page 12
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Empty

- **Page 9**

```
Processing Page 9...
Page 9 is already in memory.
Press Enter to continue to the next page...
```

- **Page 1**

```
Processing Page 3...
Page 3 is a Page Miss.
Allocated Page 3 to Frame 5. Virtual Address: 0x0002, Physical Address: 0x89 using No Replacement Needed.
```



Current Frame State

Frame	Page
Frame 0	Page 12
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Page 3

- **Page 3**

```
Processing Page 3...
Page 3 is already in memory.
Press Enter to continue to the next page...
```

- **Page 7**

```
Processing Page 7...
Page 7 is a Page Miss.
Replaced Page 12 from Frame 0 using FIFO.
Allocated Page 7 to Frame 0. Virtual Address: 0x0006, Physical Address: 0xE2 using FIFO.
```

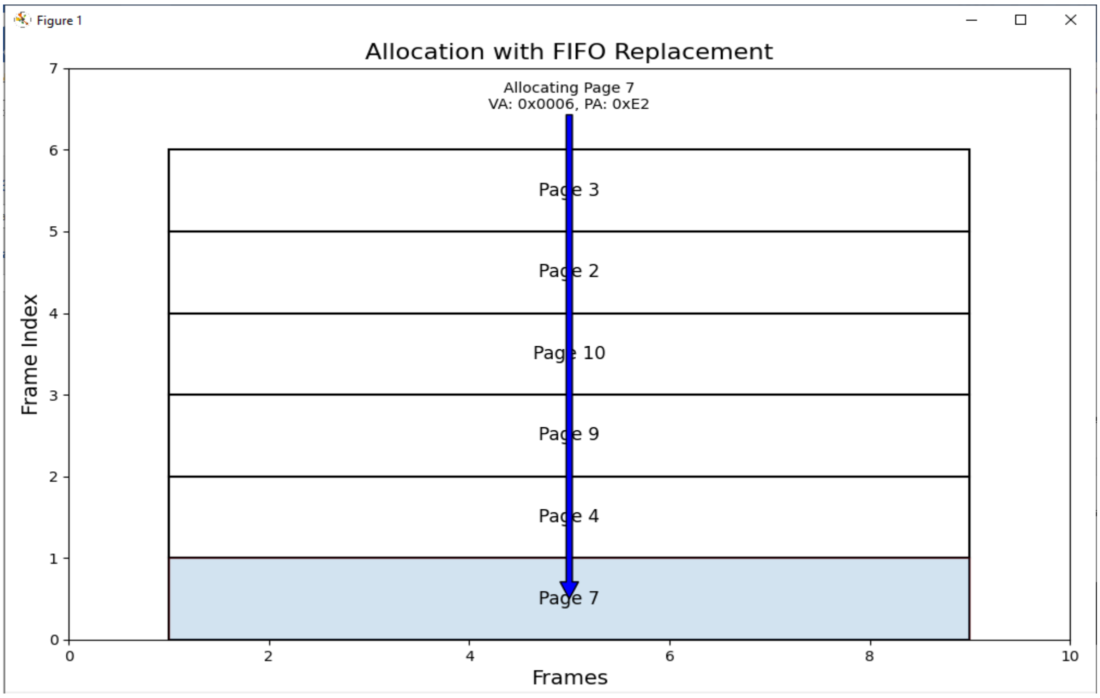


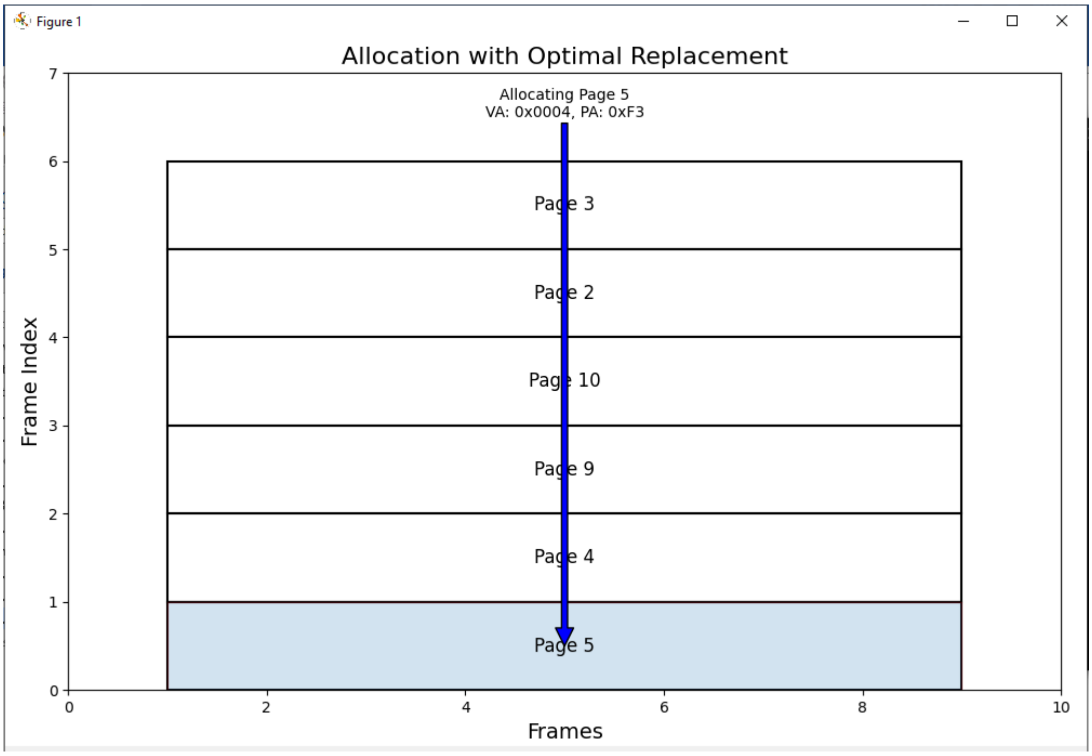
Figure 1

Current Frame State

Frame	Page
Frame 0	Page 7
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Page 3

- **Page 5**

```
Processing Page 5...
Page 5 is a Page Miss.
Replaced Page 7 from Frame 0 using Optimal.
Allocated Page 5 to Frame 0. Virtual Address: 0x0004, Physical Address: 0xF3 using Optimal.
```

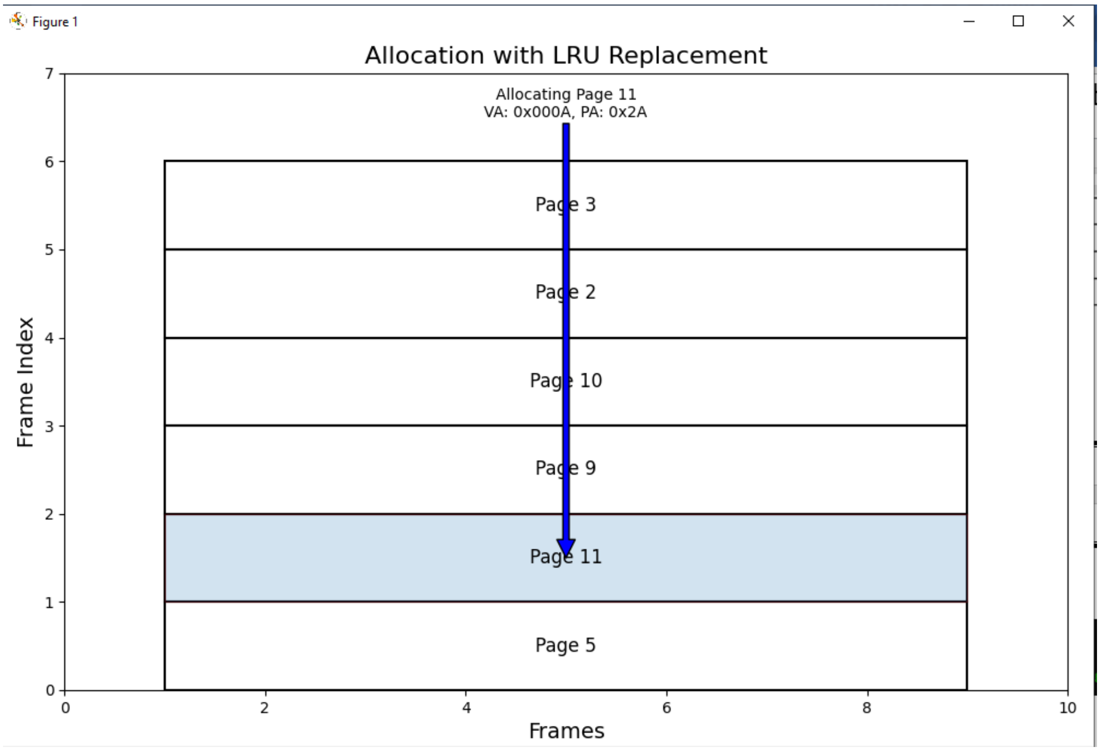


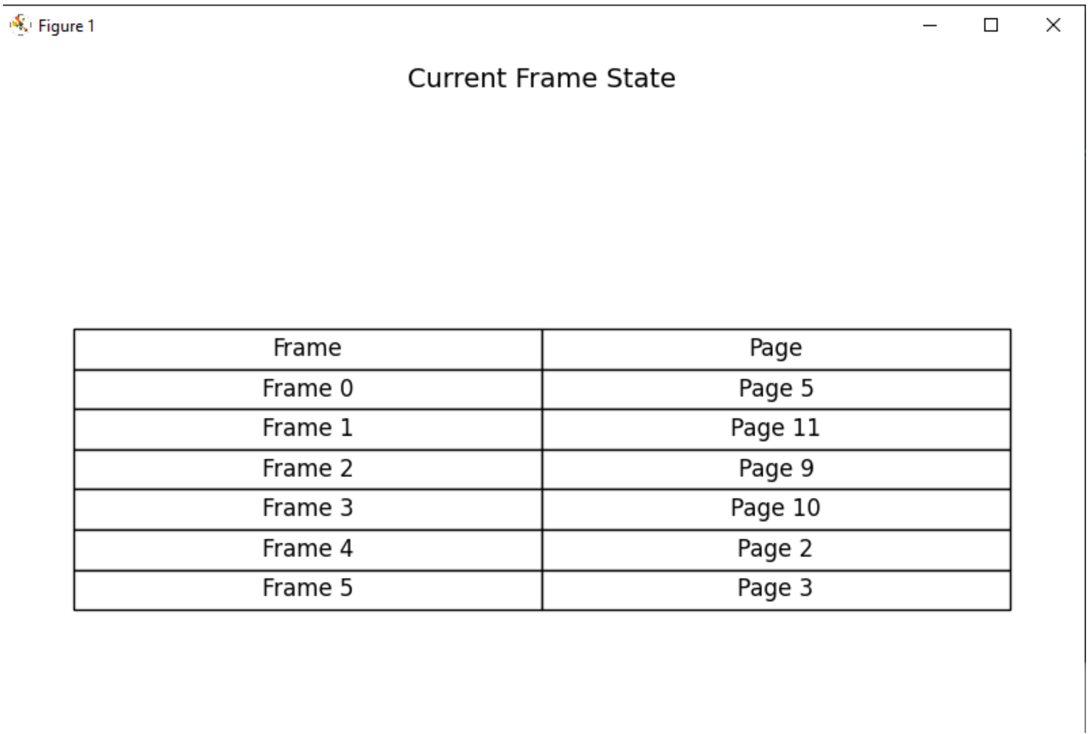
Current Frame State

Frame	Page
Frame 0	Page 5
Frame 1	Page 4
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Page 3

- **Page 11**

```
Processing Page 11...
Page 11 is a Page Miss.
Replaced Page 4 from Frame 1 using LRU.
Allocated Page 11 to Frame 1. Virtual Address: 0x000A, Physical Address: 0x2A using LRU.
```





Frame	Page
Frame 0	Page 5
Frame 1	Page 11
Frame 2	Page 9
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Page 3

- **Page 5**

```
Processing Page 5...
Page 5 is already in memory.
Press Enter to continue to the next page...
```

- **Page 4**

```
Processing Page 4...
Page 4 is a Page Miss.
Replaced Page 9 from Frame 2 using FIFO.
Allocated Page 4 to Frame 2. Virtual Address: 0x0003, Physical Address: 0x50 using FIFO.
```

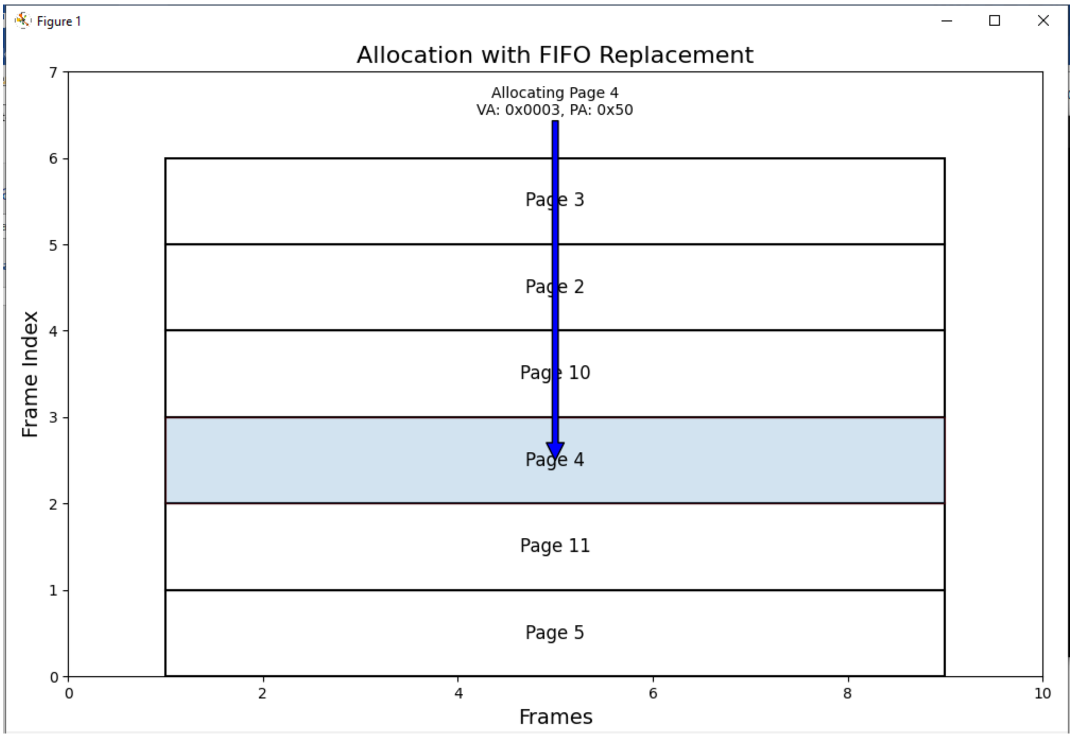


Figure 1

Current Frame State

Frame	Page
Frame 0	Page 5
Frame 1	Page 11
Frame 2	Page 4
Frame 3	Page 10
Frame 4	Page 2
Frame 5	Page 3

- **Page 2**

```
Processing Page 2...
Page 2 is already in memory.
Press Enter to continue to the next page...
```

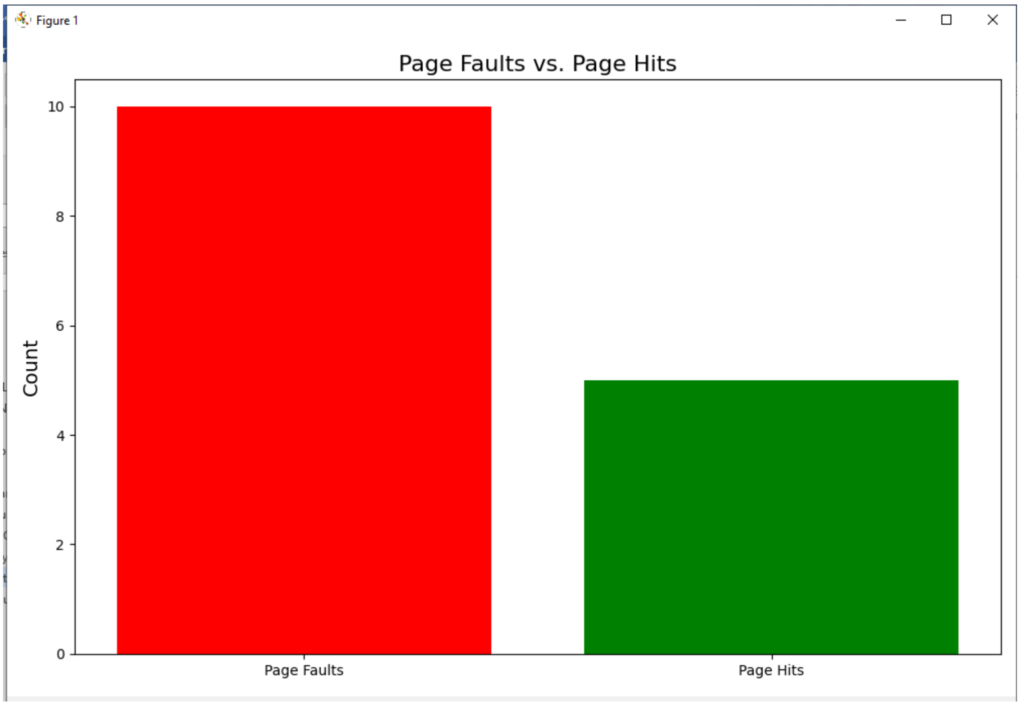
- **Final allocation table**

Figure 1

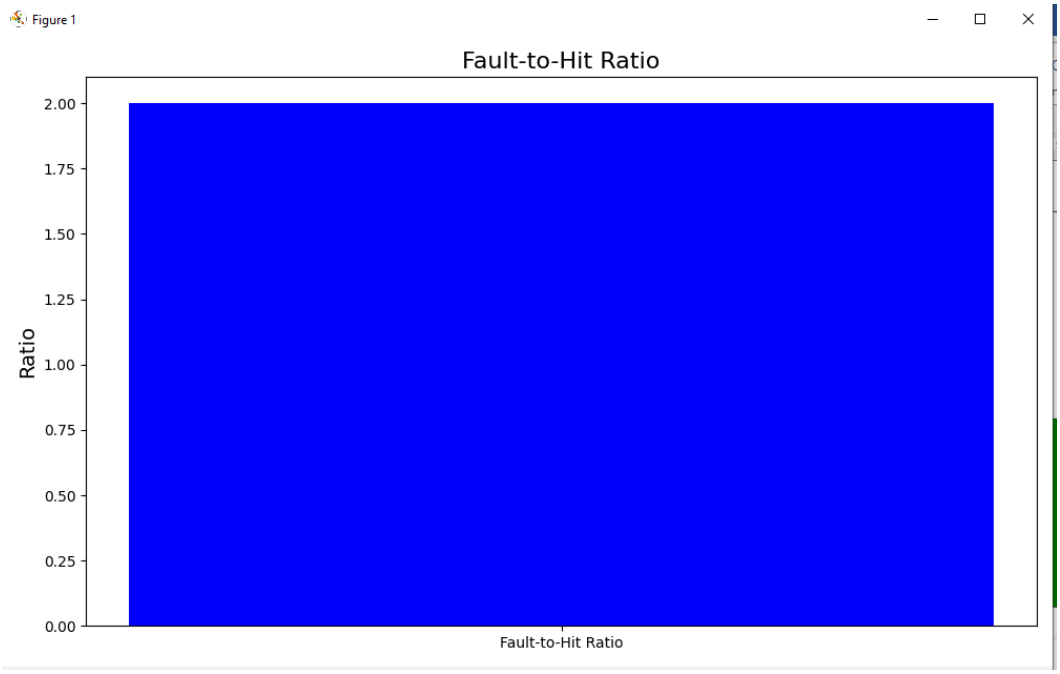
Final Allocation Table

Page	Frame	Method	Virtual Address	Physical Address
Page 12	0	No Replacement Needed	0x000B	0x10
Page 4	1	No Replacement Needed	0x0003	0xE7
Page 9	2	No Replacement Needed	0x0008	0x14
Page 10	3	No Replacement Needed	0x0009	0xA6
Page 2	4	No Replacement Needed	0x0001	0x61
Page 3	5	No Replacement Needed	0x0002	0x89
Page 7	0	FIFO	0x0006	0xE2
Page 5	0	Optimal	0x0004	0xF3
Page 11	1	LRU	0x000A	0x2A
Page 4	2	FIFO	0x0003	0x50

- **Page Faults and Page Hits Bar Chart**



- **Fault-to-Hit Ratio Bar Chart**



---

5. Discussion and Conclusions

The study indicates that each page replacement algorithm performs differently under different conditions of page reference patterns.

- **FIFO (First-In, First-Out)** generally performs poorly when future page requests are unpredictable, leading to higher page faults due to its lack of prioritization.
- **Optimal Algorithm** performs well when future page requests are foreseeable, reducing page faults by replacing pages with the least future reference distance.
- **LRU** is actually a good balance, reducing the number of page faults very effectively by tracking and replacing some of the least recently used pages.

Overall, LRU balances better between page faults and page hits. Where future references are hard to guess, it is the most effective algorithm. However Optimal performs exceptionally well under predictable future patterns. How to choose an algorithm remains dependent on the nature of page reference patterns and, of course, system requirements?

This understanding helps make virtual memory management more efficient in operating systems. Future work could look at hybrid algorithms or further optimizations in more dynamic environments for improving memory management.

6. References

Python and Matplotlib Documentation

- Python Software Foundation. (2024). Python Programming Language. Retrieved from https://www.python.org/

Virtual Memory Concepts

- Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). John Wiley & Sons.

Page Replacement Algorithms

- Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. IBM Systems Journal, 5(2), 78-101.
- Chandra, S., & Soni, M. (2020). Comparative Study of Page Replacement Algorithms: **Algorithm Implementations and Visualizations**
- GeeksforGeeks Contributors. (n.d.). Page Replacement Algorithms in Operating Systems. Retrieved from https://www.geeksforgeeks.org/
- TutorialsPoint. (n.d.). Page Replacement Algorithm. Retrieved from https://www.tutorialspoint.com/
