# SIEVE: An Efficient Eviction Policy for Bufferpool Data

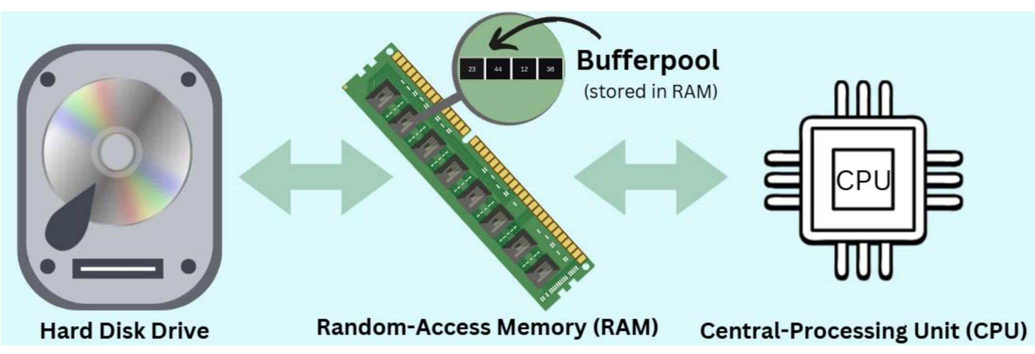

Prisha Shah[1,2], Andy Huynh[2], Tarikul Islam Papon[2], Manos Athanassoulis[2]

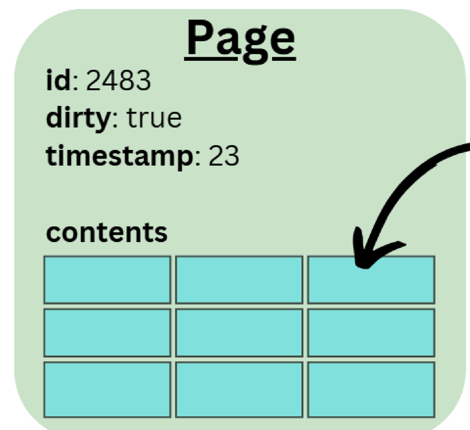
Eastlake High School (400 228th Ave NE, Sammamish, WA 98074)[1]; Boston University (665 Commonwealth Ave, Boston, MA 02215)[2]


## Introduction

### Background

- The bufferpool is a region of the computer's RAM with limited space that stores the most relevant, frequently accessed data files by a user
- Cache eviction policies are algorithms that are used to strategically determine which data files to remove when the bufferpool is full and a new page is requested
- Present-day **modern** and **classic** policies such as **ARC, FIFO, or LRU** often face a trade-off between efficiency and simplicity
- Adapting to more efficient eviction policies helps ensure **optimal energy storage**, **minimum cost**, and the **consistency of data files**




### Objective

- This research study explores a new implementation of the recently-discovered **SIEVE policy** to determine future implications of SIEVE in various workloads
- The SIEVE implementation is compared to the implementations of **LRU** (classic policy) and **CLFRU** (modern policy) to explore differences in performance metrics across workloads

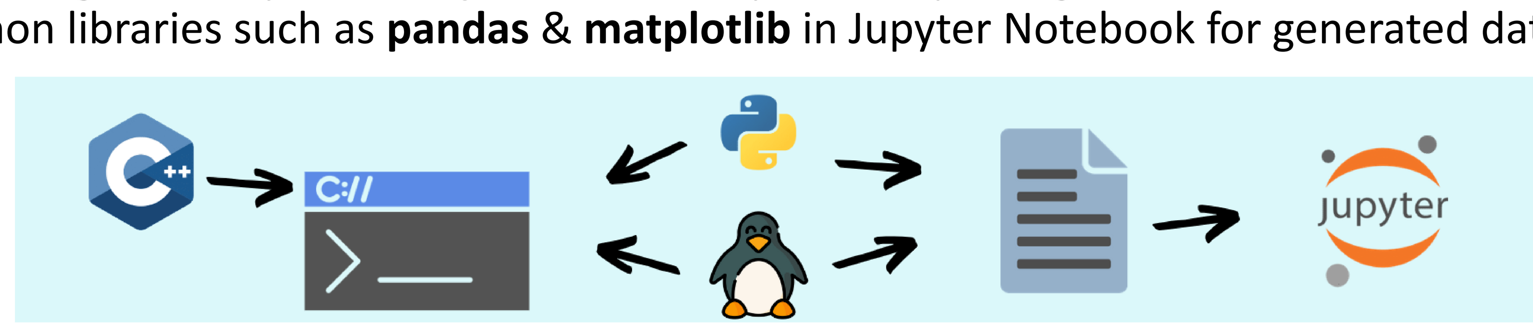
## Methods

**Part 1: Implementing Cache Eviction Policies + Read/Write Simulator (C++)**

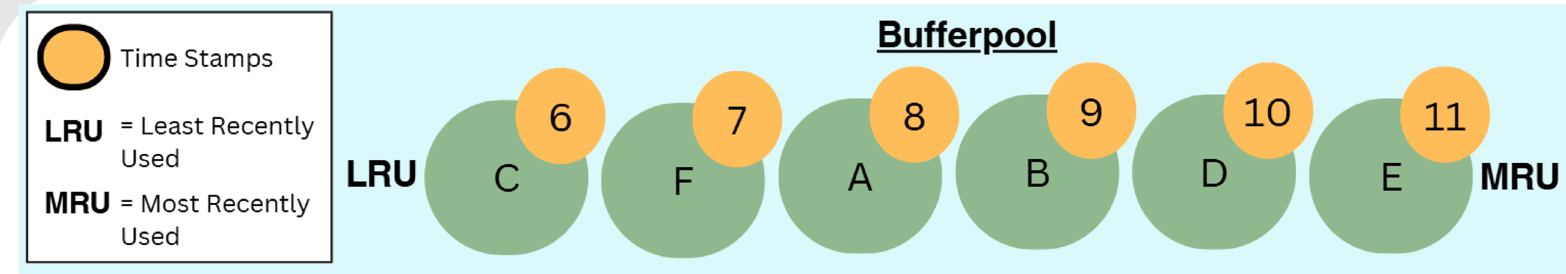
- Implemented algorithms for CFLRU, LRU, and SIEVE using VS Code + WSL
- Implementation included **workload generator**, **executor**, and **parameter** prerequisite files which were compiled using a Makefile under the target **Buffermanager**
- Executor included a **simulation** of fetching & executing read/write requests
- Each call to the Buffermanager generated metrics on the **hit rate, miss rate, read IO, write IO, and policy execution time**

**Part 2: Building a Compiler for Data Collection (Python)**

- 2 Compilers: first compiler varied **disk & bufferpool sizes** & the **number of page requests;** second compiler tracked metrics across algorithms resulting from different **workload skews** and **read/write ratios**
- Each compiler generated a CSV file with data sets for each combination of parameter values by calling repeated requests on the Buffermanager through **terminal command-prompts**
- Incorporated **Regex** and Python **Subprocess** in compilers for parsing data
- Utilized Python libraries such as **pandas** & **matplotlib** in Jupyter Notebook for generated data visualization



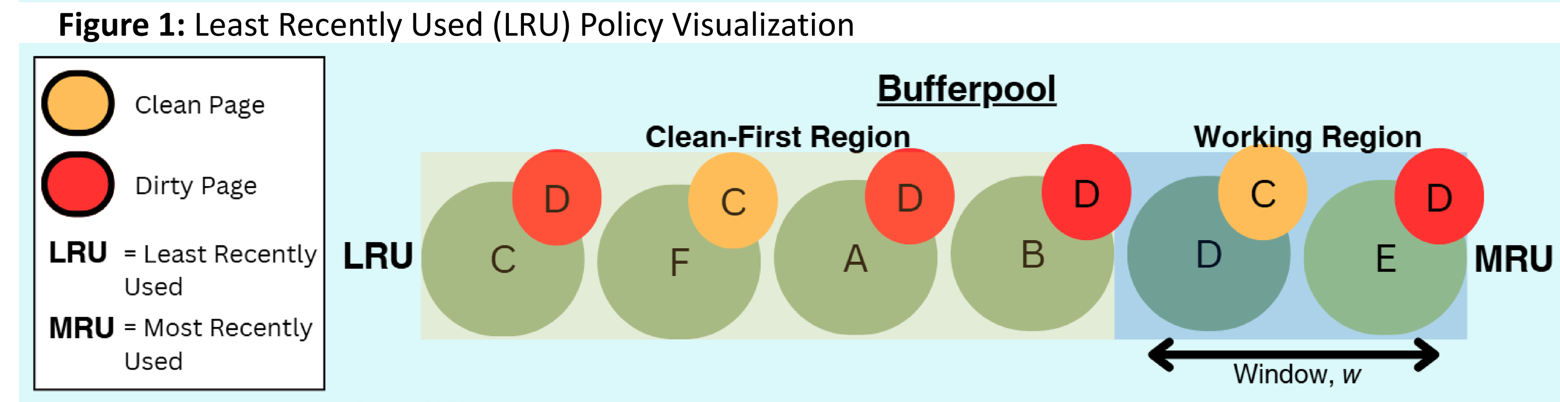

## Eviction Policies




**Each time page a is requested:**

- If the page is in the bufferpool, the page is moved to the front (MRU) and its timestamp is updated
- If the page is not in the bufferpool, the least-recently used page is evicted
  - The requested page is added in the front as the most-recently used

**Figure 1:** Least Recently Used (LRU) Policy Visualization

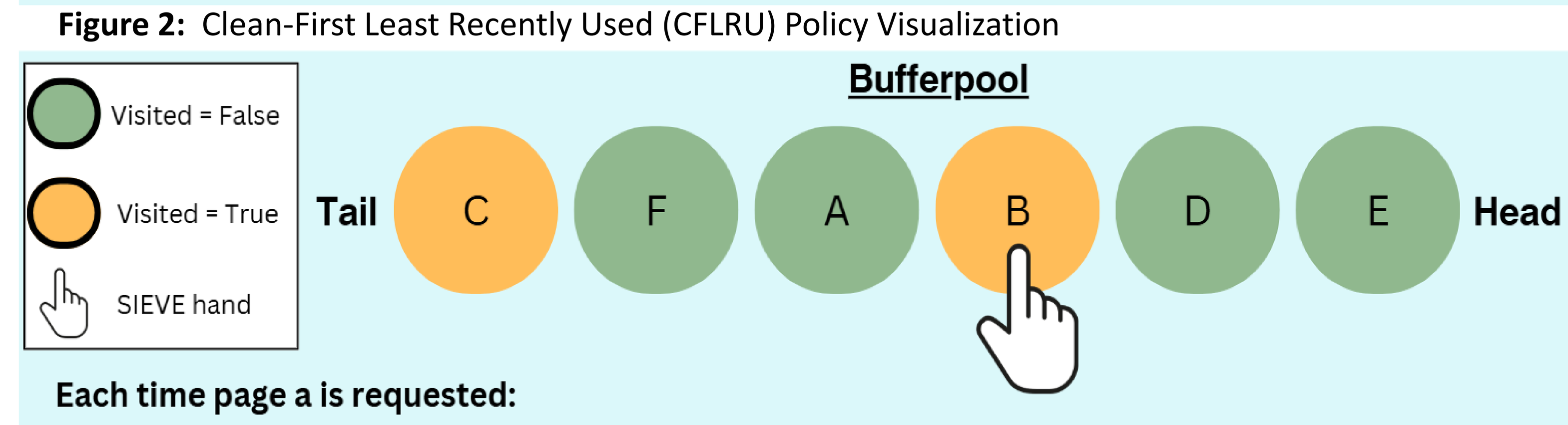



**Same as LRU with a few differences:**

- Pages are only evicted from the clean-first region
- First LRU *clean* page that is found is evicted; if clean-first region has no clean pages, the first LRU dirty page is evicted
- Window size (*w*) is determined with parameter *n*

**Window Size (*w*)** = *Buffer Size / n*

**Figure 2:** Clean-First Least Recently Used (CFLRU) Policy Visualization




**Each time page a is requested:**

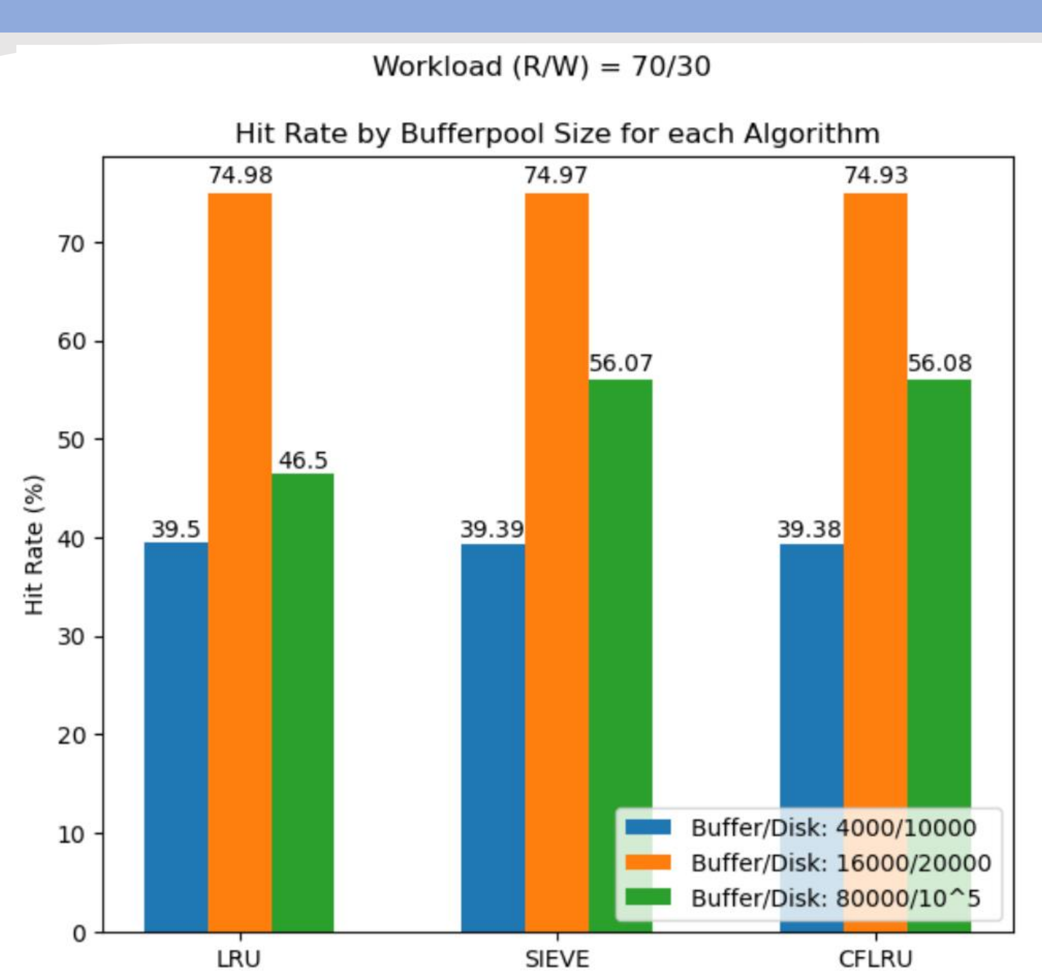
- If the SIEVE hand points to a visited page, the page is changed to false
- If hand points to an unvisited page, the page is evicted; the requested page is added to the head
- Hand moves one right (tail to head) for each request

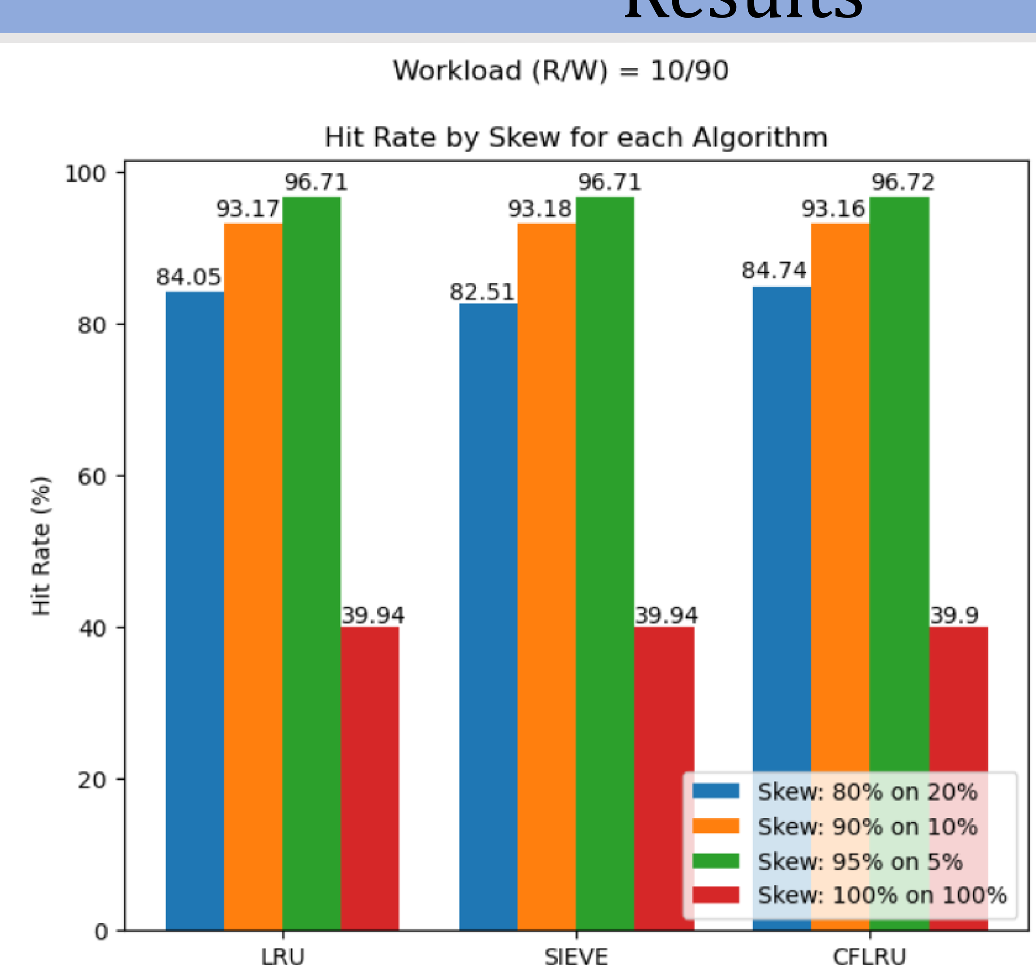
**Figure 3:** SIEVE Policy Visualization

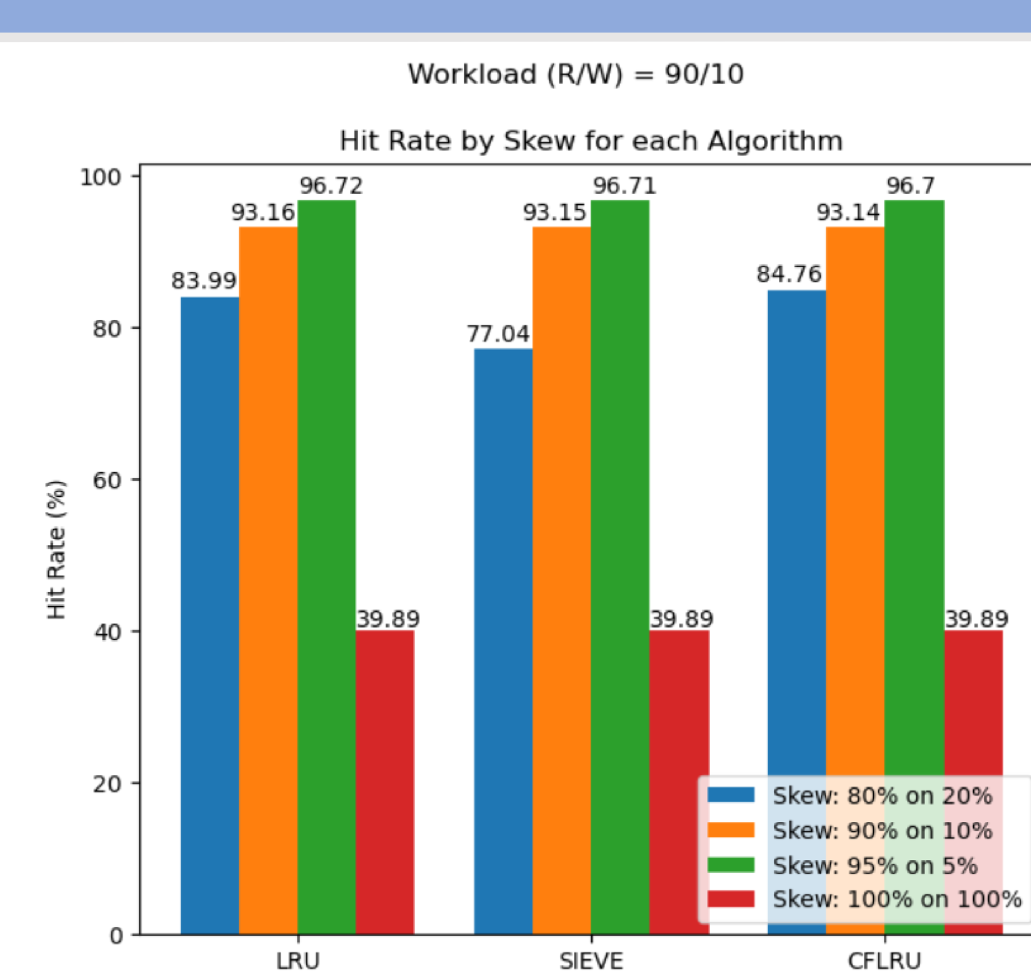
## Parameters

| Parameter | Compiler 1 | | | Compiler 2 | | | | |
|---|---|---|---|---|---|---|---|---|
| Bufferpool Size; Disk Size | 4,000; 10,000 | 16,000; 20,000 | 80,000; 10^5 | 4,000; 10,000 | | | | |
| Number of Requests | 10^5 | 2 * 10^5 | 10^6 | 10^6 | | | | |
| Read/Write Ratio | 70/30 | | | 10/90 | 30/70 | 50/50 | 70/30 | 90/10 |
| Workload Skew (% accesses on % pages) | 100%, 100% | | | 80%, 20% | 90%, 10% | 95%, 5% | 100%, 100% | |

## Results




Skewed workloads refer to the % of accesses on a % of pages in the bufferpool (e.g., 80% on 20% means 80% of accesses are done on 20% of the bufferpool data)

## Conclusions

- SIEVE performs significantly better (+9.57%) compared to LRU within larger environments that encompass larger bufferpool/disk sizes (e.g., Buffer, Disk sizes of 80,000, 10^5)
- SIEVE and CFLRU have comparable hit rates in macroenvironments; however, CFLRU takes over 161 times the processing speed of SIEVE --> SIEVE is more efficient in HDDs
- SIEVE outperforms LRU in hit rate on write-heavy workloads (R/W = 10/90); however, it underperforms when enacted in read-heavy workloads (R/W = 90/10)

**Future Implications**

- The SIEVE policy can be implemented in larger environments with a higher number of request operations, which may improve long-term efficiency for cache eviction
- Implementing the SIEVE policy can be integral in write-heavy systems such as applications for logging systems or financial transactions

## References


[1] Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon Lee. 2006. CFLRU: a replacement algorithm for flash memory. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems (CASES '06). Association for Computing Machinery, New York, NY, USA, 234–241. https://doi.org/10.1145/1176760.1176789

[2] SIEVE: an Efficient Turn-Key Eviction Algorithm for Web Caches - SIEVE is simpler than LRU. Github.io. https://cachemon.github.io/SIEVE-website/blog/2023/12/17/sieve-is-simpler-than-lru/#sieve-is-beyond-an-eviction-algorithm (accessed 2024-07-02).

[3] Zhang, Y.; Yang, J.; Yue, Y.; Vigfusson, Y.; Rashmi, K. V. SIEVE Is Simpler than LRU: An Efficient Turn-Key Eviction Algorithm for Web Caches. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24); 2023.


## Acknowledgements


I would like to thank my mentors Andy Huynh and Tarikul Islam Papon, and Professor Manos Athanassoulis for their invaluable guidance and support throughout this project. I would also like to thank Boston University for the amazing experience I had during my research.


https://github.com/s-prshah/cache_eviction_simulation