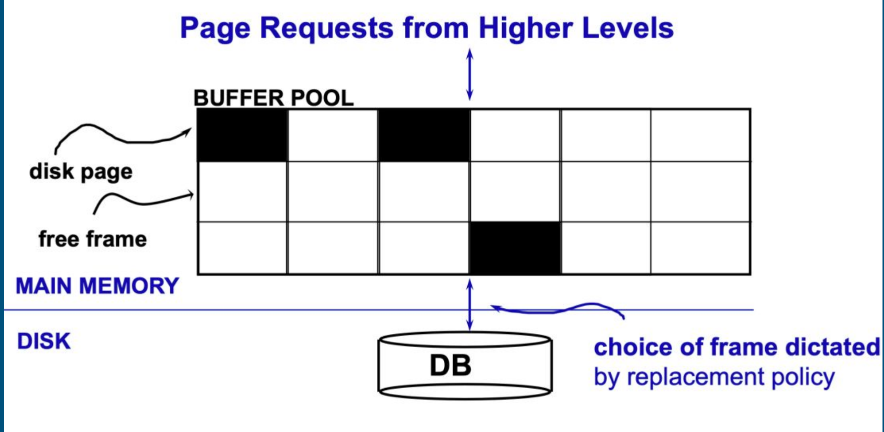
Bufferpool Implementation

Mia Li, Samir Farhat Dominguez,
Stephany Yipchoy

Introduction

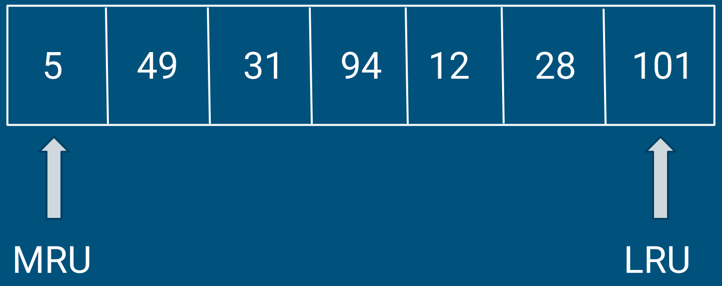
- Bufferpool overview
  - Buffer Hit
  - Buffer Miss
  - Dirty Bits
  - Eviction Strategies
- Optimize for different workloads to **maximize read and write performance** by maximizing hits



LRU Policy

- When the bufferpool becomes full, we discard the least recently used page

- The pages that have been most heavily used in the past are more likely to be used heavily in the future too



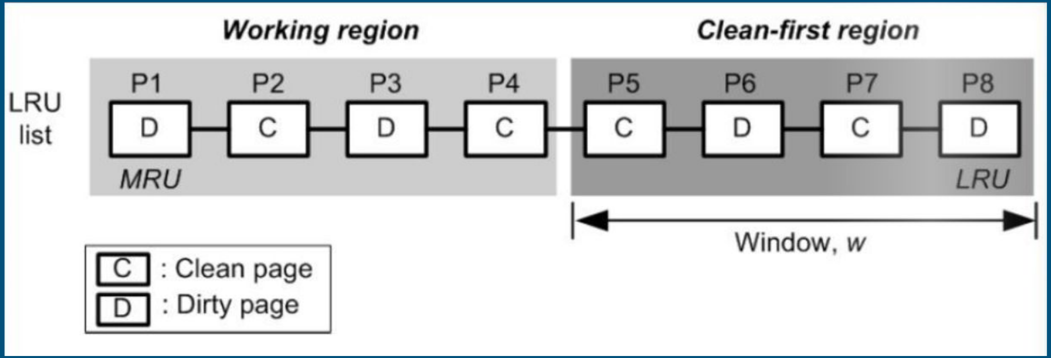
- Deque -> easy insertion/remove from both end

- Store: Page Id

- Return: Position of the least used page in bufferpool -> target evicting position

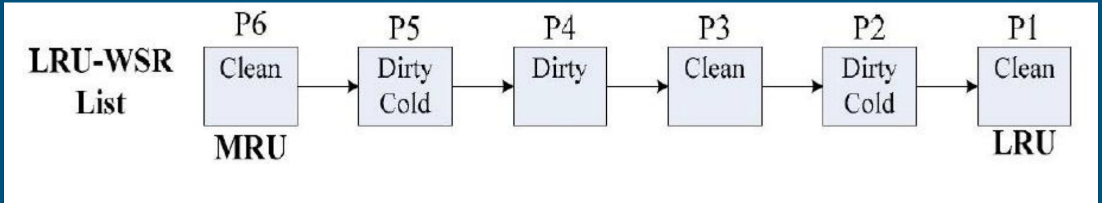
CFLRU Policy

- Separate LRU list into 2 parts
- Keep a certain amount of dirty pages in cache to reduce the number of flash write operations
- we opted for the window to be ⅓ of the current amount of pages in the buffer



LRU: p8 -> p7 -> p6-> p5
CFLRU: p7 -> p5 -> p8 -> p6

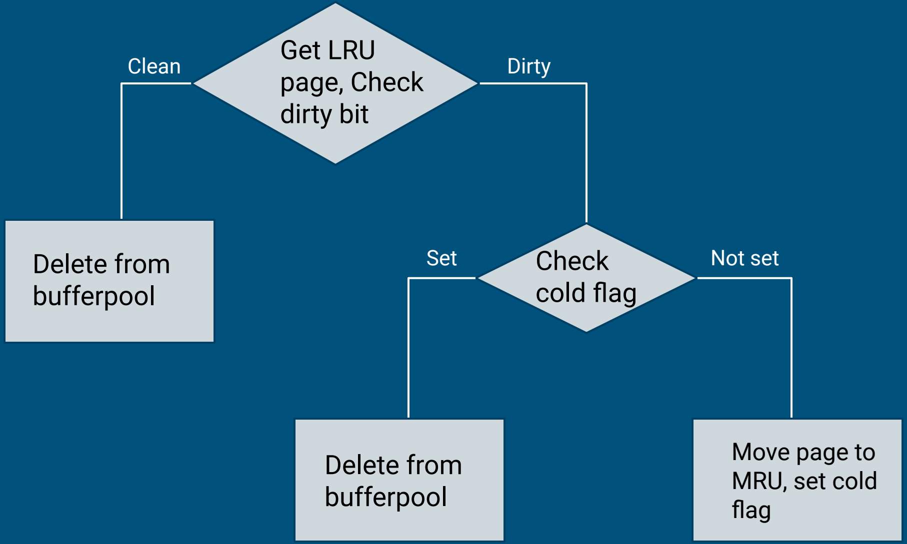
LRU-WSR Policy



- Second chance algorithm -> Cold flag can only be set on the second time (reordered writing sequences)

- The only difference between those two policies is that LRU-WSR assign each page with a bit flag called “cold-flag”

- In bufferpool we store **tuple(page id, dirty bit, cold flag)**



FIFO Policy

- First in first out, implemented by queue

- when a page hits:
  - LRU will move this page to the MRU position
  - FIFO will make no changes to the queue.

- Simple but not efficient for large number of pages → the operating system keeps track of all pages in the memory in a queue

1	2	3	4	5	1	3	1	6	3	2	3

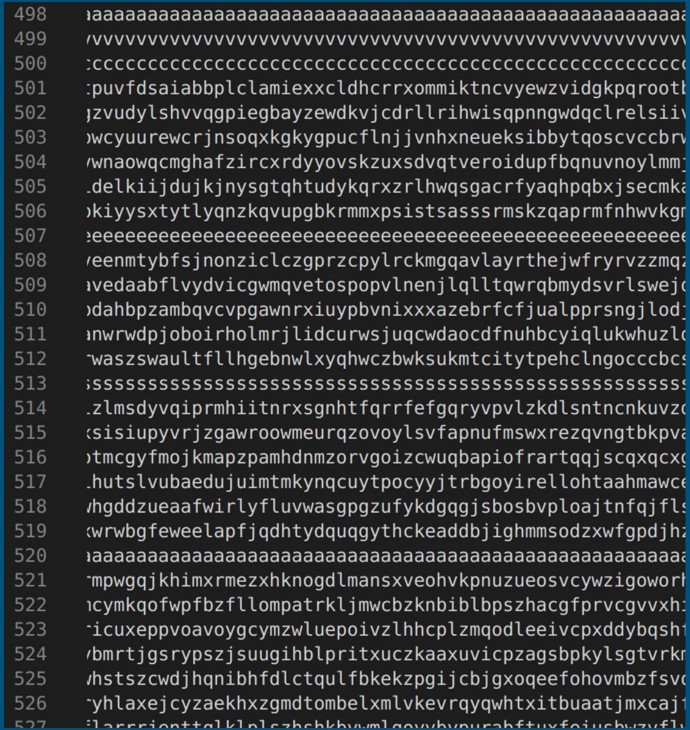
1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3

M	M	M	M	M	M	H	H	M	M	M	H

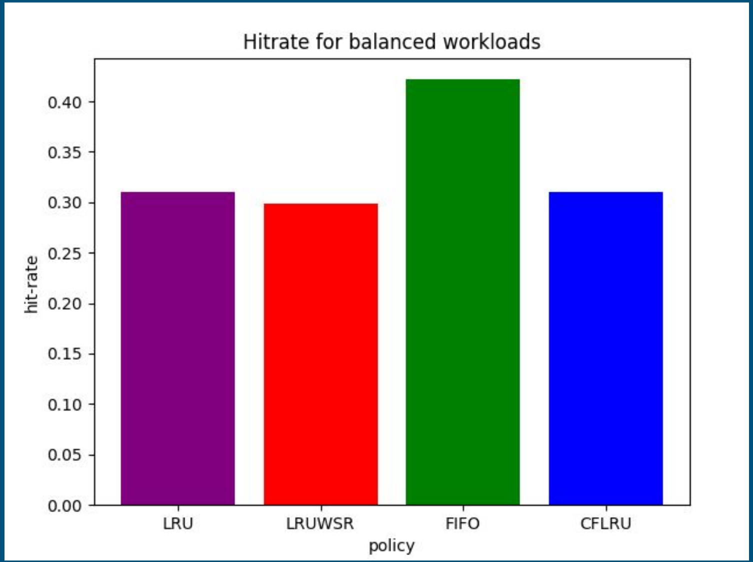
M = Miss
H = Hit

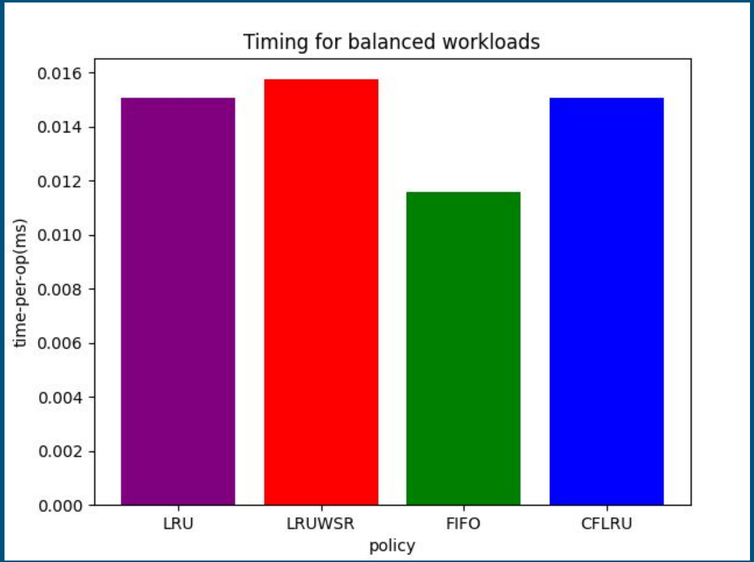
Disk Implementation

- Disk populated with arbitrary random bytes
- Writes and reads with seekg
    - One continuous character when written over(except last character is ‘\n’)
    - One page per line, no sectors because just a quick computation and isn’t productive to objectives
- Keeping file open may not be true reflection of disk access unless disk size exceeds RAM
    - We saw only a slight increase in timings
- Disk object belongs to Buffer class



Experiments





Challenges and Lessons Learned

- Disk management
  - At first only appending seemed possible without a complete overwrite
- Seg faults in overall simulation run script
- Construct the bufferpool to be more adaptive
- Learned that it’s much easier to have a small workload and know exactly how it should perform
- Read other research for guidance

Conclusions

- Implemented: FIFO, LRU, CFLRU, LRU-WSR, and Disk Functionality
- For balanced workloads, the simplest approach is also the best
- Intended extensions of this work:
    - Experiment with read heavy and write heavy workloads
    - Add more flexibility to the buffer with more parameters

```
You, 6 days ago | 1 author (You)
namespace bufmanager {
    You, 6 days ago | 1 author (You)
    class Buffer {
        // This class maintains specific property of the buffer.
        // You definitely need to modify this part
        // You need to add more variables here for your implementation. For example, curren

        private:
            Buffer(Simulation_Environment* _env);
            static Buffer* buffer_instance;

        public:
            //initizate bufferpool with <page size, dirty page bit>
            vector< pair<int, bool> > bufferpool;
            //initizate bufferpool for LRU WSR with <page size, dirty page bit, cold flag>
            vector< tuple<int, bool, bool> > bufferpool_wsr;
            //deck as the lru candidate list
            deque<int> lru_candidate;
            //for cflru: going to store either pageid of clean or -1
            deque<int> fifo_candidates;
            static long max_buffer_size;  //in pages

            static Buffer* getBufferInstance(Simulation_Environment* _env);
            // statistics to track
            static int buffer_hit;
            static int buffer_miss;
            static int read_io;
            static int write_io;
            static char disk_write_char;
            static std::chrono::duration <double, milli> timing;

            // Page replacement algorithm
            int LRU();
            int LRUWSR();
            int FIFO();
            int CFLRU();

            // Disk parameters
            std::fstream disk;
            static int pageSize;

            // Statistics collection
            static void writeResults();
            static int printBuffer();
            static int printStats();
    };

```