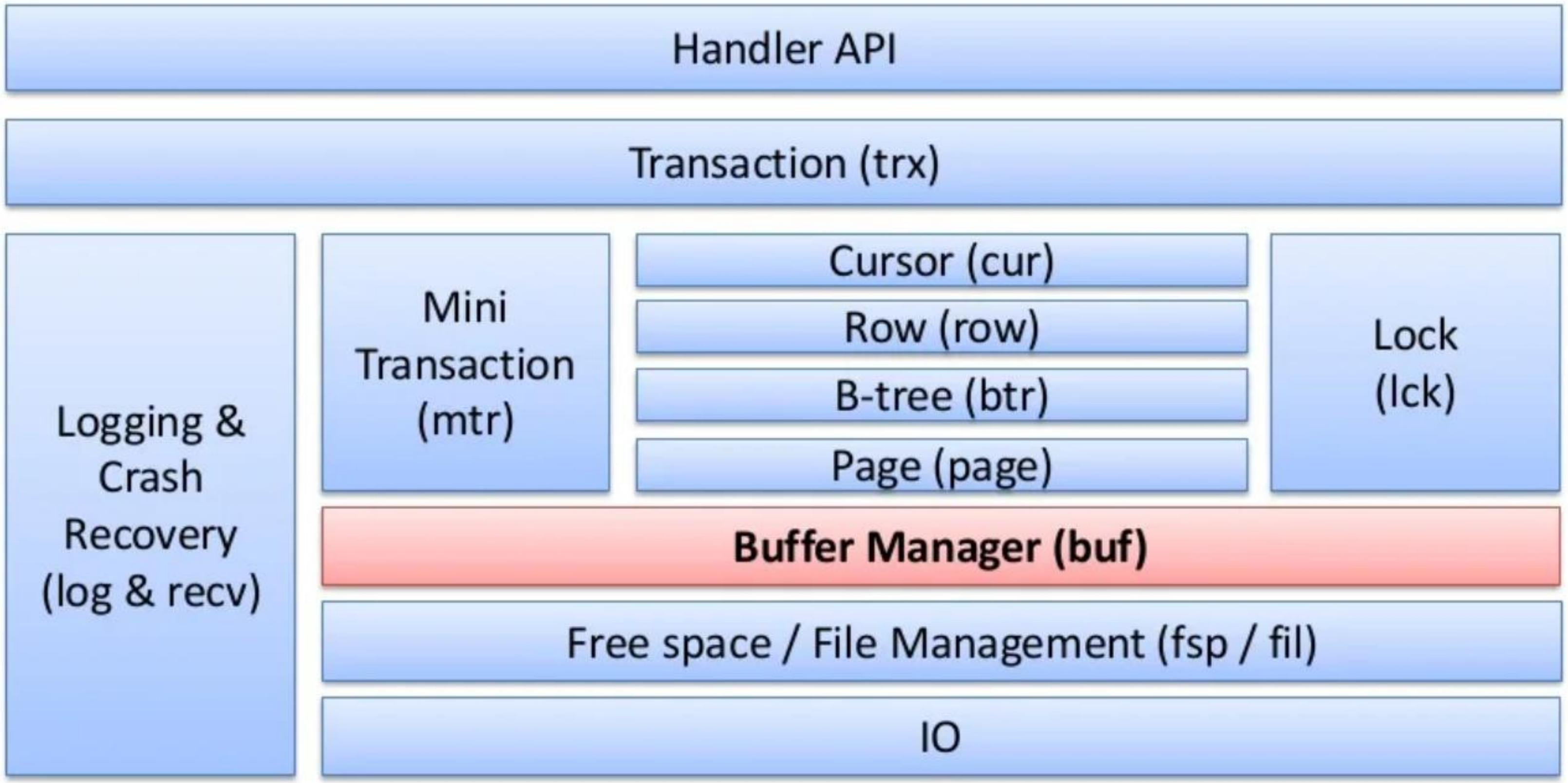
Buffer Management of MySQL/InnoDB
(1) Processing Page Read Request

Bo-Hyun Lee
lia323@skku.edu

Index

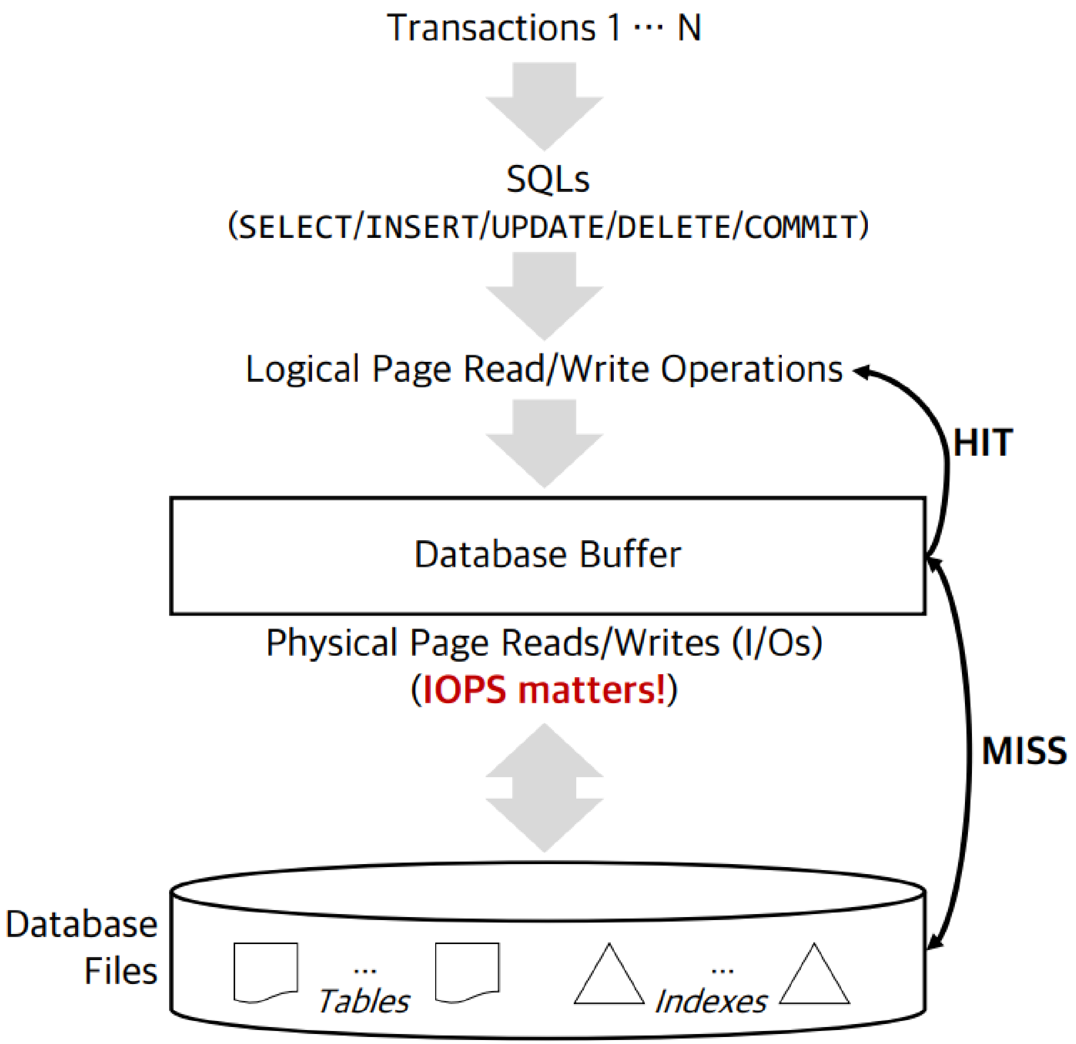
- Database I/O Architecture
- Overview of Buffer Manager in DBMS and Bufferpool Structure
- LRU List Management Upon Page Hit
- Buffer Replacement Policy of MySQL/InnoDB Upon Page Miss
- Buffer Pool Monitoring Method

InnoDB Architecture



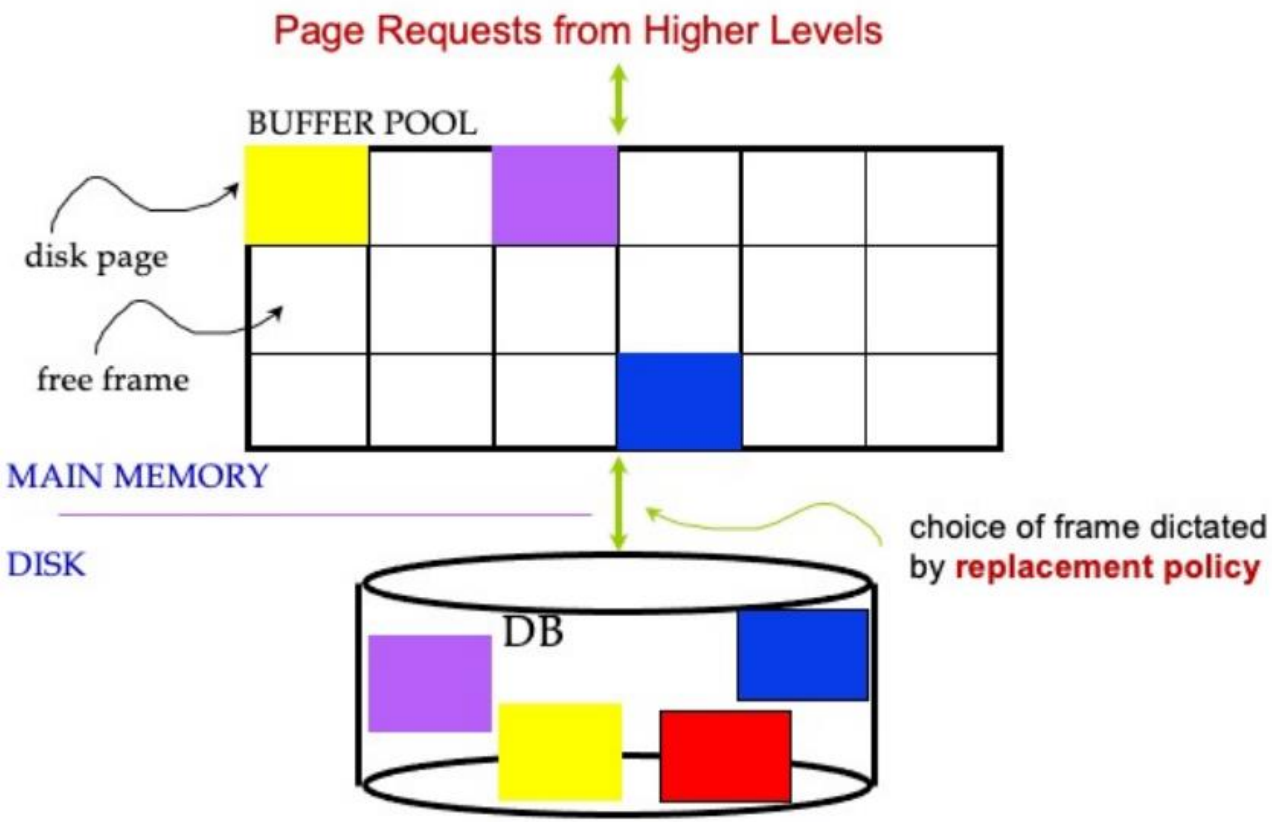
mysql-5.7.33/storage/innobase/buf

Database I/O Architecture



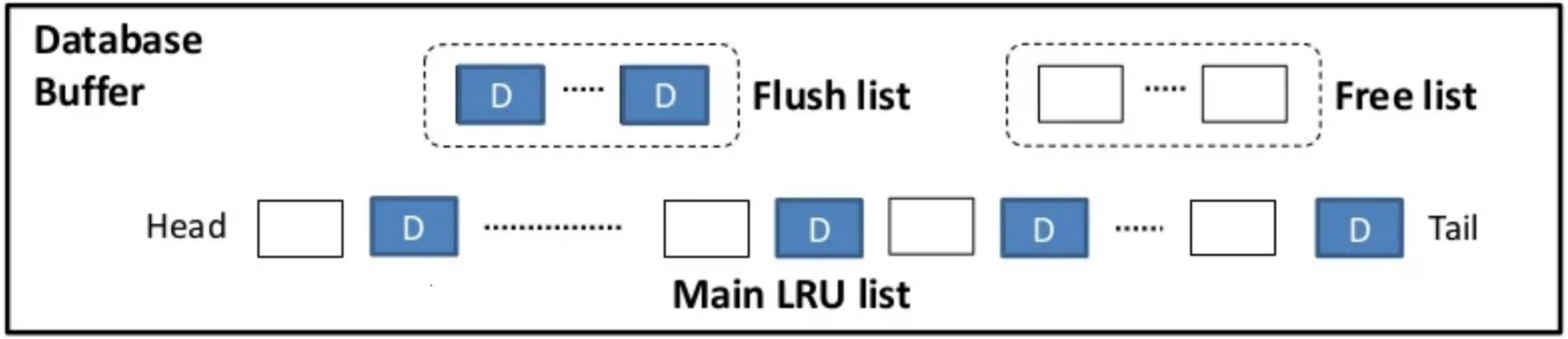
- Transaction:
  - A sequence of SQL statements
  - A sequence of *reads* & *writes*
- SQL statements:
  - `SELECT`: *reads* tuple from pages
  - `INSERT/DELETE/UPDATE`: *write* records in pages
- Buffer **HIT** & **MISS**:
  - When page(s) is in buffer (i.e., **HIT**): DRAM operation
  - Otherwise (i.e., **MISS**): Disk I/O
    - In case of dirty victim, write the page to the storage
    - Read page(s) from storage

Buffer Manager Overview



- Data must be in **DRAM** for DBMS to operate it
- The buffer manager caches frequently accessed data into DRAM to improve performance

Lists of Buffer Blocks



- **Free list**
  - Contains free (empty) buffer frames
- **LRU list**
  - Contains all the blocks holding a file page
- **Flush list**
  - Contains the blocks holding file pages that have been modified in the memory but not written to disk yet (i.e., **dirty**)

buf0buf.h: buf_pool structure

```
struct buf_pool_t{

    /** @name General fields */
    /* @{ */
    ib_mutex_t  mutex;      /*!< Buffer pool mutex of this
                    instance */
    ib_mutex_t  zip_mutex;  /*!< Zip mutex of this buffer
                    pool instance, protects compressed
                    only pages (of type buf_page_t, not
                    buf_block_t */
    ulint       instance_no;    /*!< Array index of this buffer
                    pool instance */
    ulint       old_pool_size;  /*!< Old pool size in bytes */
    ulint       curr_pool_size; /*!< Current pool size in bytes */
    ulint       LRU_old_ratio;  /*!< Reserve this much of the buffer
                    pool for "old" blocks */
…
    UT_LIST_BASE_NODE_T(buf_page_t) flush_list;
    UT_LIST_BASE_NODE_T(buf_page_t) free;
                    /*!< base node of the free
                   block list */
    UT_LIST_BASE_NODE_T(buf_page_t) LRU;
```

Flush list, free list, LRU list per buffer pool instance

When a Page Read Request Occurs…

- `buf0buf.cc → buf_page_get_gen()`
  - `block = (buf_block_t*) buf_page_hash_get_low(buf_pool, page_id);`

- If the requested page is in the buffer pool (**Buffer HIT**): `block != NULL`
  - Pin the page and return its address : `fix_block = block;`

- If the requested page is not in the buffer pool (**Buffer MISS**): `block == NULL`
  - Chooses a victim buffer frame for replacement: block: `buf_LRU_get_free_block(buf_pool);`
    - If the chosen victim is clean: simply discard the data of the frame
    - If the chosen victim is dirty, first write it to storage
  - Then, read the requested page from storage into the victim buffer frame: `buf_read_page()`
  - Pin the page and return its address

Buffer Hit Ratio & Performance

- **Hit Ratio = # of hits / # of page requests to buffer cache**
  - one MISS → one (or two) disk I/O
  - Disk access time = DRAM access time * 1000

- If the hit ratio increases from 90% to 91%
  - Miss ratio: 10% → 9%
  - how much improvement? ***1%** or **10%**?*

Device	Latency (4KB random read)	Performance Gap (compared to DRAM)
Traditional SSD	~100µs	1,000x
HDD	~10 ms	100,000x
DRAM	~100ns	1x

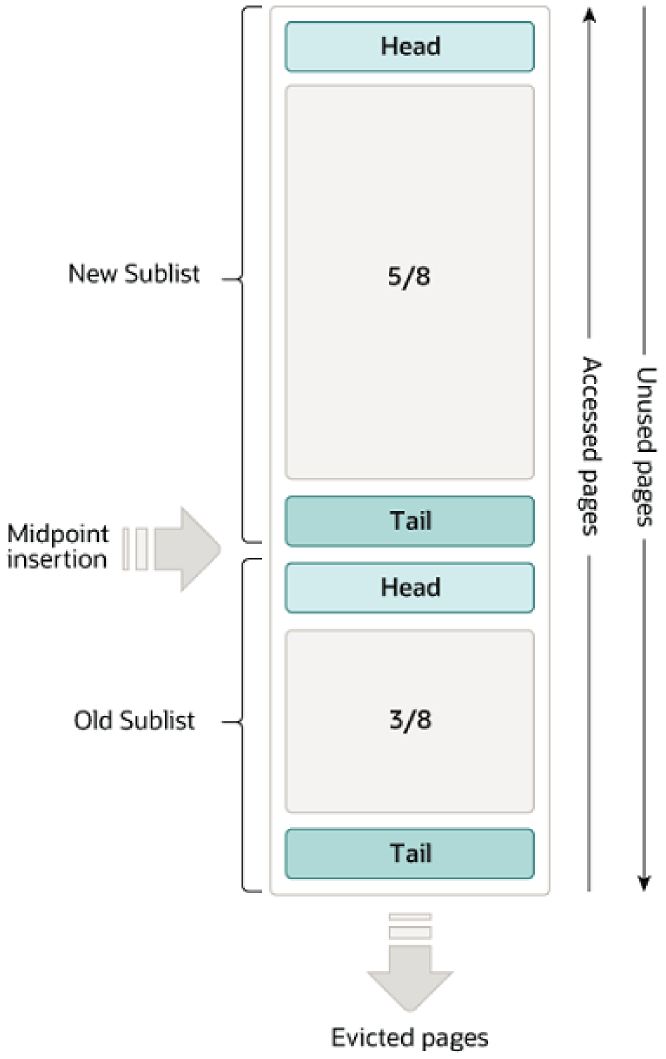
Table 1: Performance gap in storage hierarchy [1]

MySQL Buffer Pool and Replacement Policy

- Buffer Replacement Policy
  - A buffer frame is chosen for replacement by a replacement policy
  - ex. Random, FIFO, LRU, MRU, LFU, Clock, etc...
  - Replacement policy can have a big impact on # of I/Os
  - Better replacement policy will result in a higher buffer hit ratio

- Buffer Pool
  - Area in main memory where InnoDB caches data as it is accessed
  - Its buffer replacement policy is based on **LRU** **(Least Recently Used)**
    - data that is rarely used is aged out of the cache
    - Buffer pool uses linked list for cache management

Buffer Pool LRU List Management Upon Page Hit



- When MySQL reads a page into the buffer pool, it initially inserts at the midpoint
  - ✓ `buf0lru.cc buf_LRU_add_block_low():`
  - ✓ `UT_LIST_INSERT_AFTER(buf_pool->LRU, buf_pool->LRU_old, bpage);`

- Page HITs in the:
  - **Old sub-list:** makes the page young (move page to the head of the new sub-list)
    - ✓ `Buf0buf.cc buf_page_make_young_if_needed()`
  - **New sub-list:** moves the page to the head of the new sub-list only if they are a certain distance from the head
    - ✓ `buf_page_peek_if_young() then`
    - ✓ `buf_page_make_young_if_needed()`

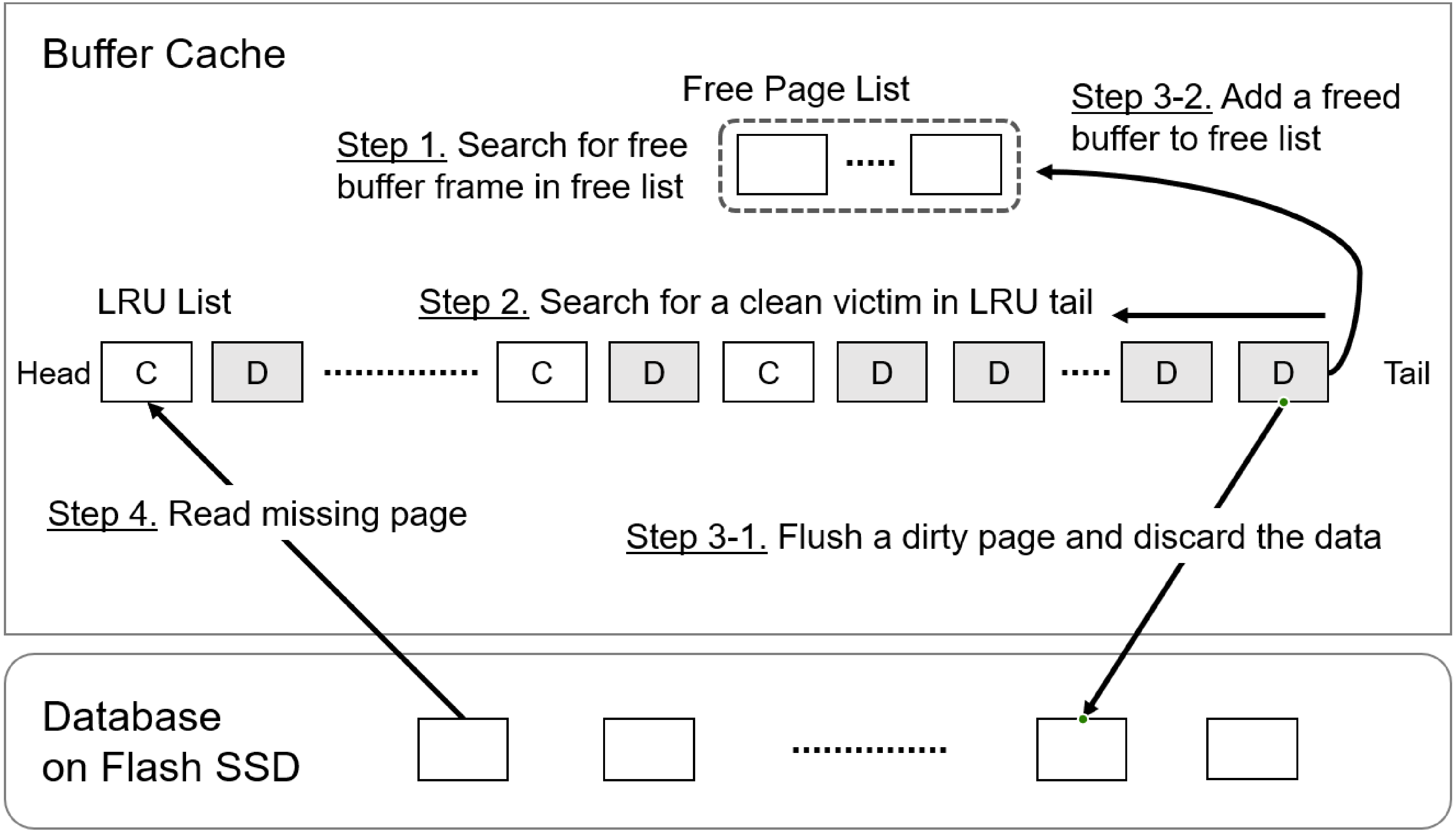
- Least recently used pages move to the tail of the list and are evicted

Making the Buffer Pool Scan Resistant

- Why is LRU list divided into two *sub-lists*?
  - To minimize the amount of data that is brought into the buffer pool and never accessed again
  - To make sure frequently accessed (“hot”) pages remain in the buffer pool
  - Keeps the buffer pool from being *churned* by read-ahead or full table/index scans

- Why midpoint insertion?
  - Read-ahead or large scans make pages inserted at the midpoint of the LRU list but they are *aged out* as they are no longer accessed and soon evicted from buffer pool

Victim Selection Upon Page Miss



Victim Selection Policy in MySQL

- `buf0lru.cc buf_LRU_get_free_block()`
- **Step 1:**
  - Search for a free buffer frame
  - `block = buf_LRU_get_free_only(buf_pool);`
- **Step 2:**
  - Scan the LRU tail to search for a clean page
  - freed = buf_LRU_scan_and_free_block(buf_pool, n_iterations > 0);
- **Step 3:**
  - Flush (write) a dirty page at the end of the LRU list
  - `buf_flush_single_page_from_LRU(buf_pool)`

Buffer Pool Monitoring Method

- InnoDB Standard Monitor output, which can be viewed using `SHOW ENGINE INNODB STATUS`, provides metrics regarding operation of the buffer pool

- Number of clean/dirty, old/young pages, *hit ratio* per buffer pool instance

```
---------------------- BUFFER POOL AND MEMORY --------------------
--
Total large memory allocated 2198863872
Dictionary memory allocated 776332
Buffer pool size 131072
Free buffers 124908
Database pages 5720
Old database pages 2071
Modified db pages 910
Pending reads 0 Pending writes: LRU 0, flush list 0, single page
0 Pages made young 4, not young 0 0.10 youngs/s, 0.00 non-
youngs/s
Pages read 197, created 5523, written 5060 0.00 reads/s,
190.89 creates/s, 244.94 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000
not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s,
Random read ahead 0.00/s
LRU len: 5720, unzip_LRU len: 0 I/O sum[0]:cur[0], unzip
sum[0]:cur[0]
```

Measure MySQL Performance by Varying the Buffer Size

- This week, you will learn to measure the hit/miss ratio in MySQL while running the TPC-C benchmark

- You will also measure the performance metrics by varying the buffer size

- Then, you will analyze the impact of different buffer sizes on the overall performance (e.g., transaction throughput, hit ratio, read/s, write/s, etc.)

- Refer to week3 https://github.com/LeeBohyun/SWE3033-S20223

References

[1] Bryan Harris and Nihat Altiparmak. 2020. Ultra-low latency SSDs' impact on overall energy efficiency. In Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'20). USENIX Association, USA, Article 2, 2.

[2] MySQL document: https://dev.mysql.com

[3] Mijin An, MySQL Buffer Management, https://www.slideshare.net/meeeejin/mysql-buffer-management