Project3

Buffer management

Scalable Computing Systems Laboratory
Hanyang University

Submission

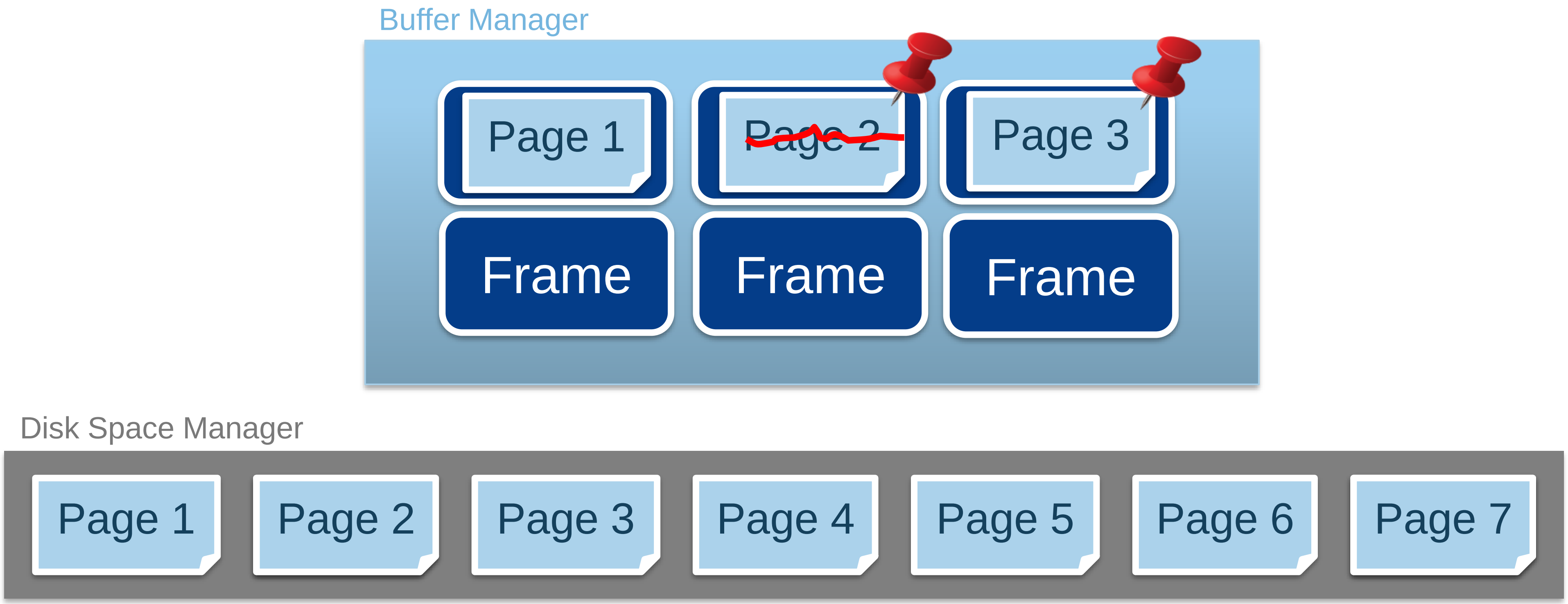
- You should submit your project in a directory structure like this: "your_repo/project3/db_project".
- Follow the db_project directory structure as before.

```
ktlee20@multicore-36:~/db2021$ ls
project1  project2  project3
ktlee20@multicore-36:~/db2021$ tree project3
project3
└── db_project
    ├── CMakeLists.txt
    ├── db
    │   ├── CMakeLists.txt
    │   ├── include
    │   │   ├── bpt.h
    │   │   ├── buffer.h
    │   │   └── file.h
    │   └── src
    │       ├── bpt.cc
    │       ├── buffer.c
    │       └── file.cc
    ├── DbConfig.h.in
    ├── main.cc
    └── test
        ├── basic_test.cc
        ├── CMakeLists.txt
        └── file_test.cc

5 directories, 13 files
```

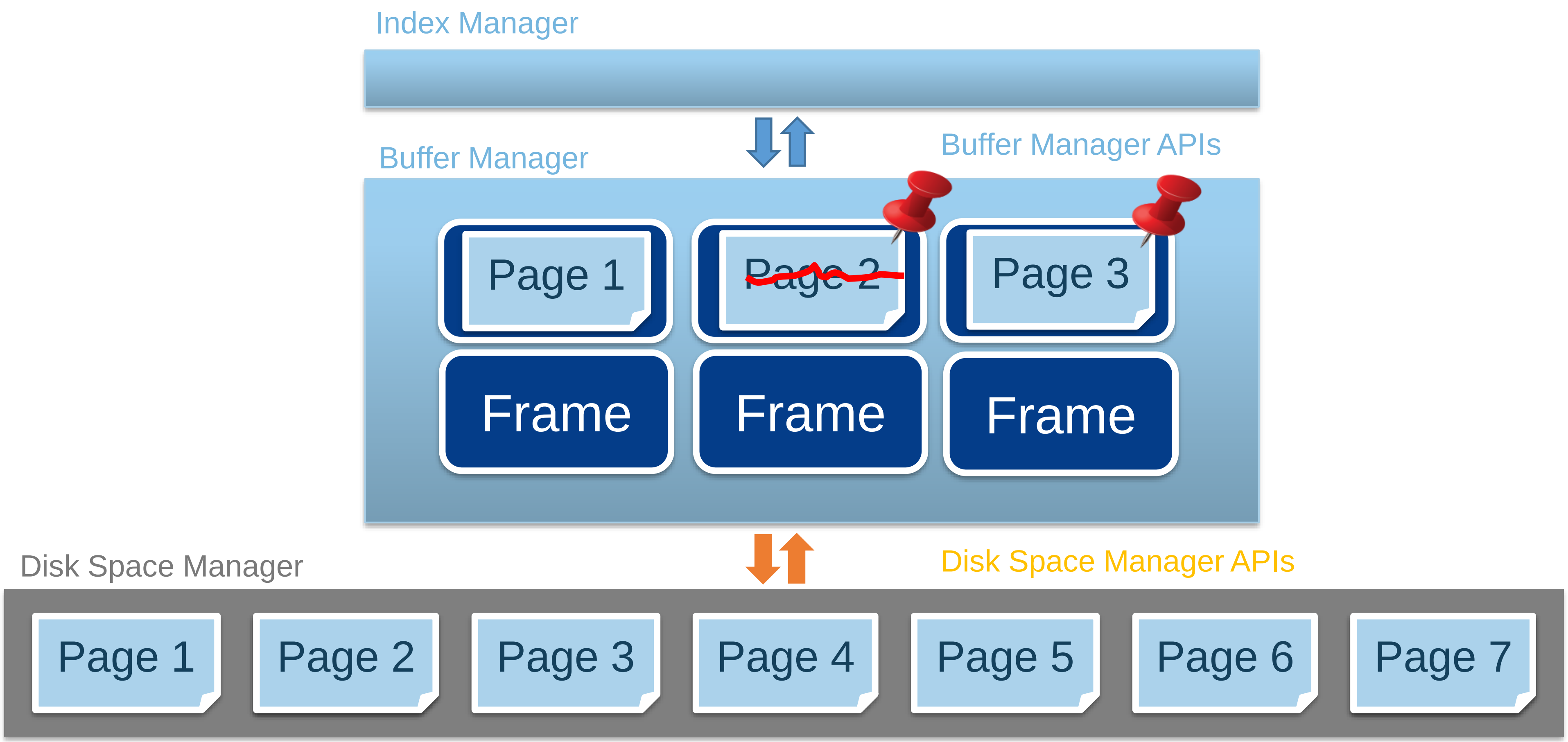
Buffer Management

- The current disk-based b+tree doesn't support buffer management.
- Our goal is to implement an **in-memory buffer manager** for caching on-disk pages.



Buffer Management

- Disk manager APIs should be called only within the buffer manager layer.



Project Specification

- Define a buffer block structure which contains the following fields.
  - **Physical frame**: containing up-to-date contents of a target page.
  - **Table id:** the unique id of a table (per file)
  - **Page number**: the target page number within a file.
  - **Is dirty:** whether this buffer block is dirty or not.
  - **Is pinned:** whether this buffer is-use(pinned) or not.
  - **LRU list next/prev** : data for representing a LRU list
  - Other data necessary for your design.

Buffer Structure

frame (page size : 4096 bytes)
table_id
page_num
is_dirty
is_pinned
next of LRU
prev of LRU

Project Specification

- Modify the database initialization function.
  - **int init_db (int num_buf);**
  - Allocate the buffer pool with the given number of entries (i.e., num_buf).
  - Initialize other fields for your own design.
  - On success, return 0. Otherwise, return a non-zero value.

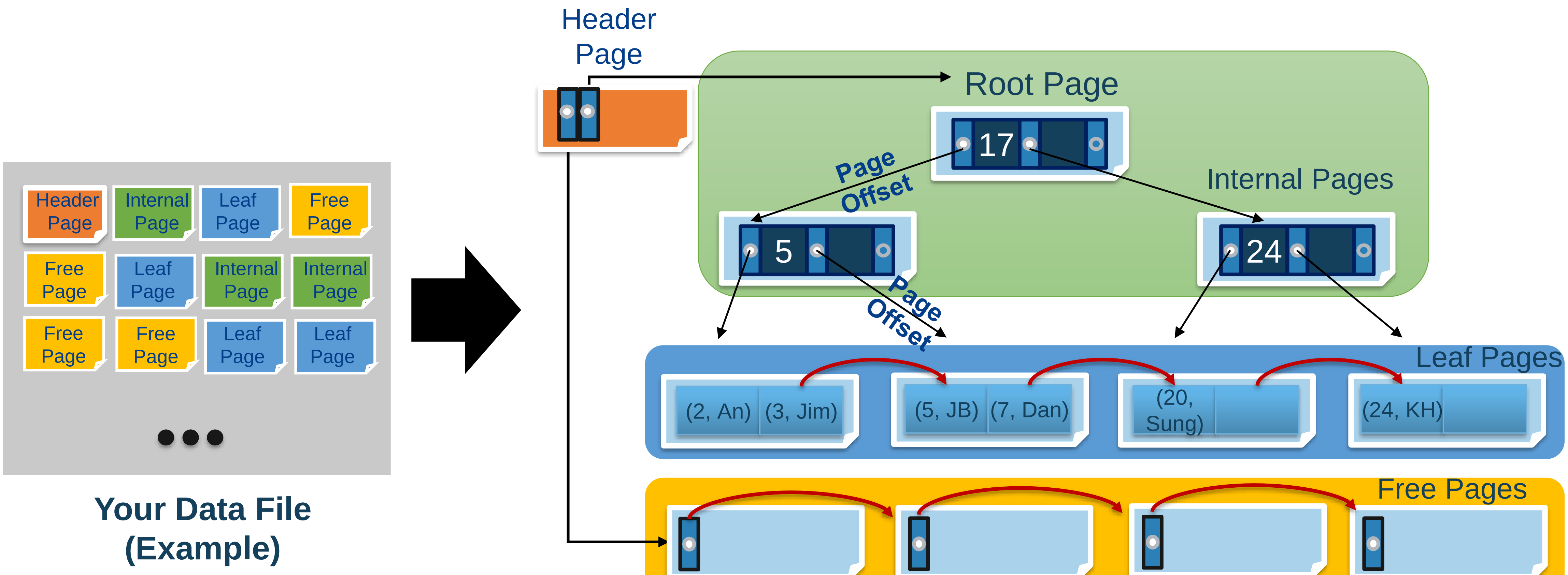
Project Specification

- Your APIs must always go through the buffer manager layer for accessing any data. (more details in the following slides)
  - If a page is not in the buffer pool (i.e., cache-miss), read the page from disk and maintain it in a buffer block.
  - Page modification should only occur within the in-memory buffer. If a page frame in memory is updated, mark the buffer block as dirty.
  - Select the victim page for eviction by following the LRU policy. Write the page to disk during the eviction process.

Project Specification

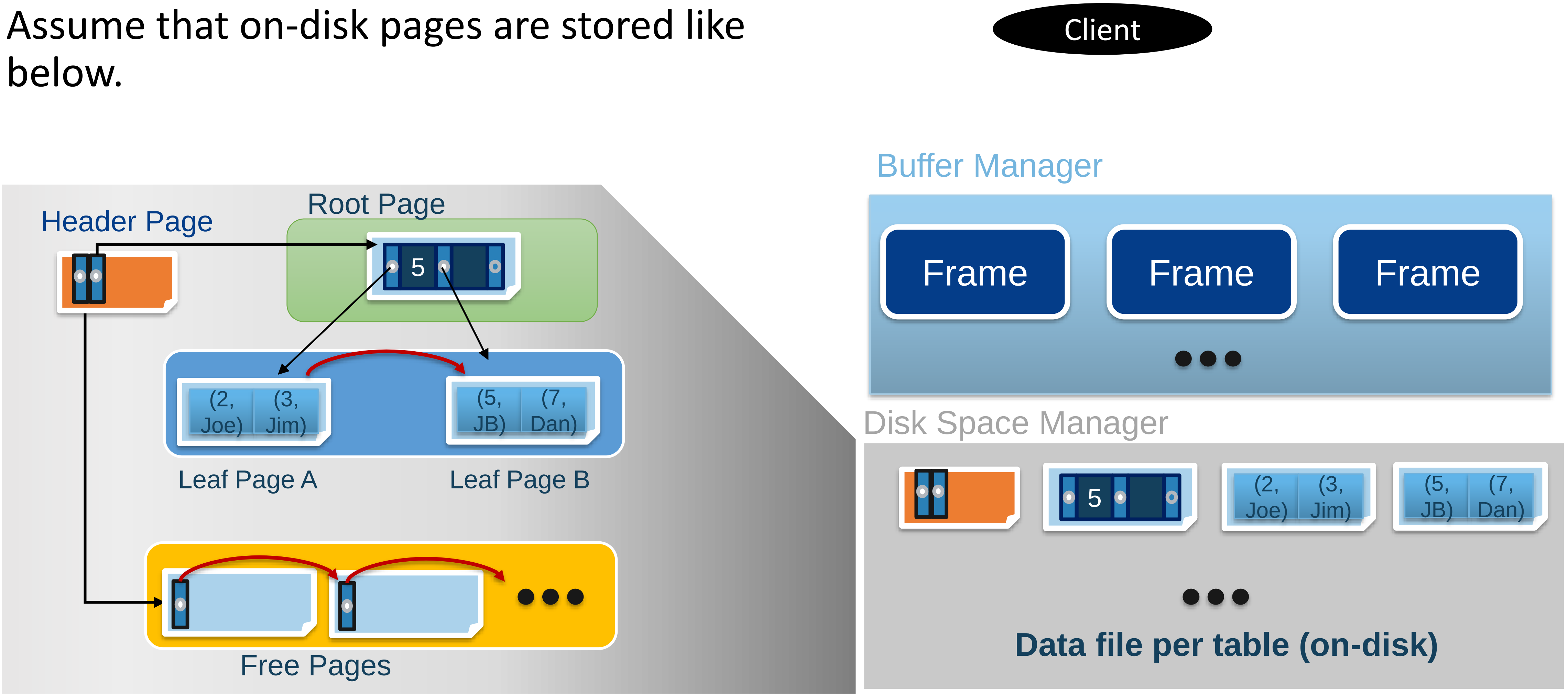
- Modify database shutdown function.
  - **int shutdown_db();**
  - Flush the entire buffer and destroy(e.g., deallocate, etc.) it.
  - On success, return 0. Otherwise, return a non-zero value.
  - Call file_close_table_files() to dereference the open files.

So far..


Header Page
Internal Page
Leaf Page
Free Page
Free Page
Leaf Page
Internal Page
Internal Page
Free Page
Free Page
Leaf Page
Leaf Page
Your Data File (Example)
Header Page
Root Page
17
Page Offset
Internal Pages
5
24
Page Offset
Leaf Pages
(2, An)
(3, Jim)
(5, JB)
(7, Dan)
(20, Sung)
(24, KH)
Free Pages

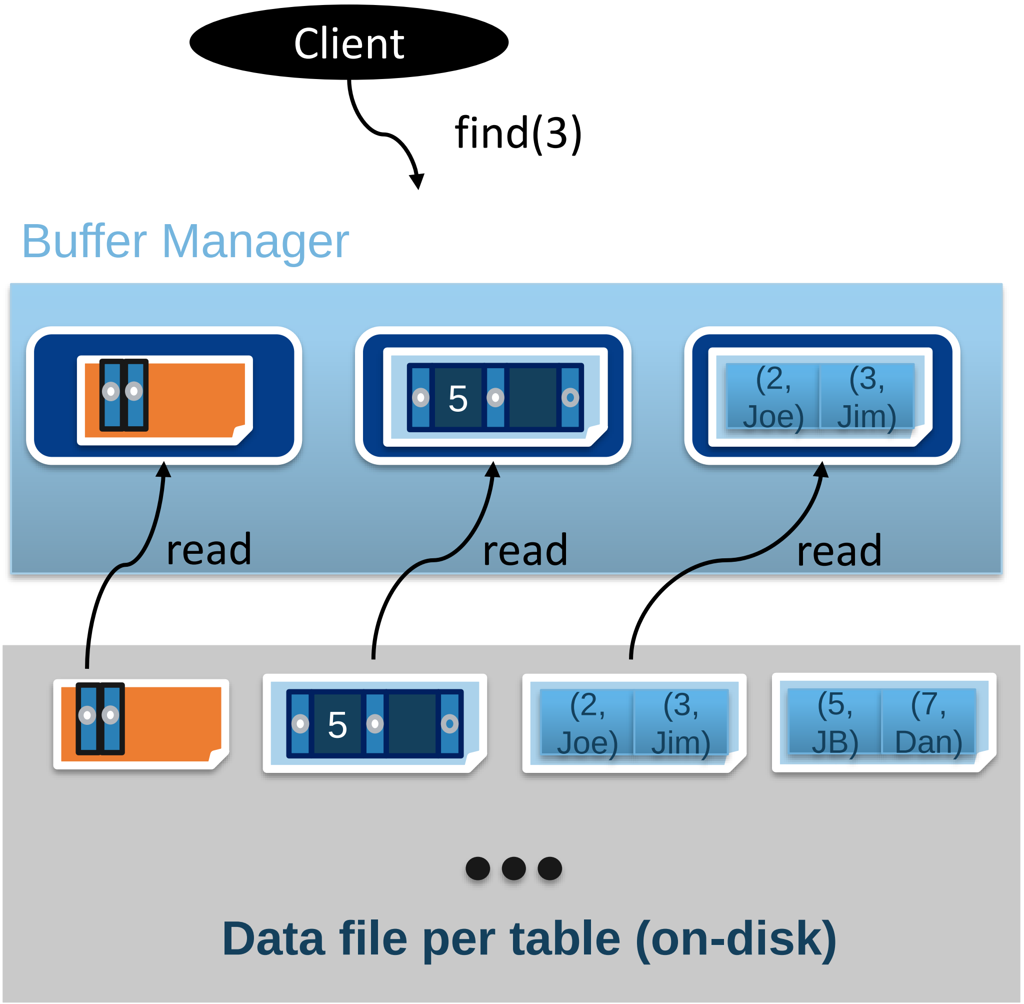
Buffer Management

- Assume that on-disk pages are stored like below.



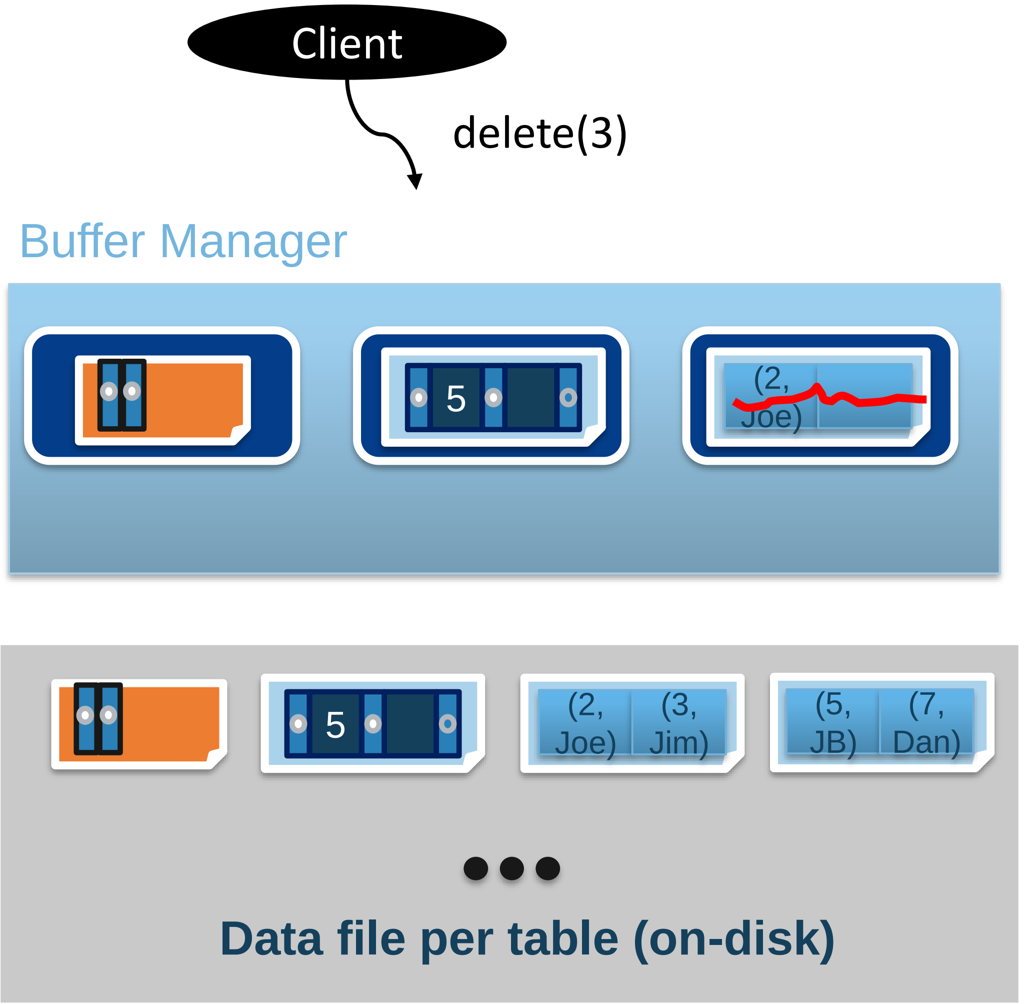
Buffer Management

- First, search for the target page within the buffer pool.
- If the desired page is not found (i.e., cache-miss occurs), read the page from disk and maintain it in a buffer block.
- Traversing the index from root to leaf page incurs access to various page types(e.g., header, root, internal, leaf, etc.). All pages are to be read/modified through the buffer manager layer.



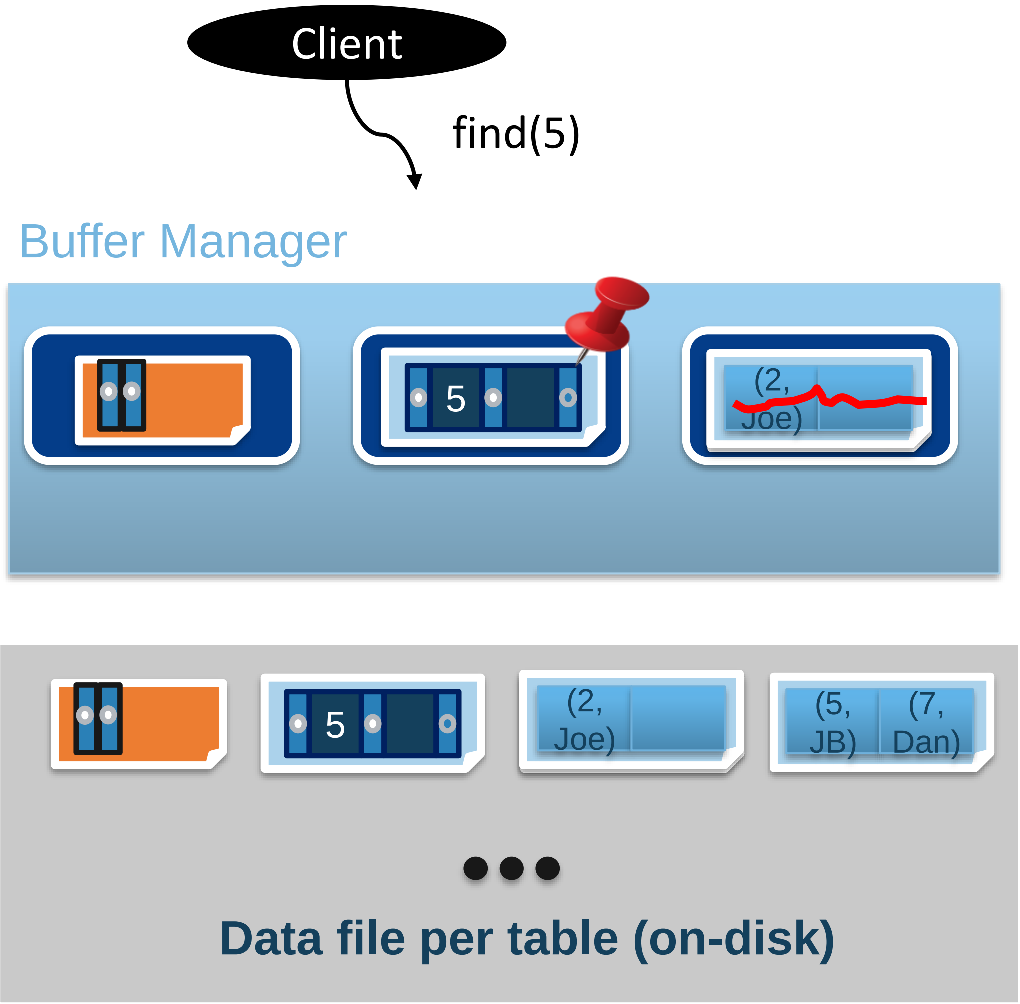
Buffer Management

- A modification operation must be handled within the in-memory buffer. You must read the page to buffer before performing any modifications.
  - Do NOT perform direct reads/writes from/to disk.
- Example(figure on the right)
  - Performing "delete key 3" will modify the page in buffer and mark the corresponding page dirty (as crossed-out in red). Notice that the on-disk page is not accessed directly.



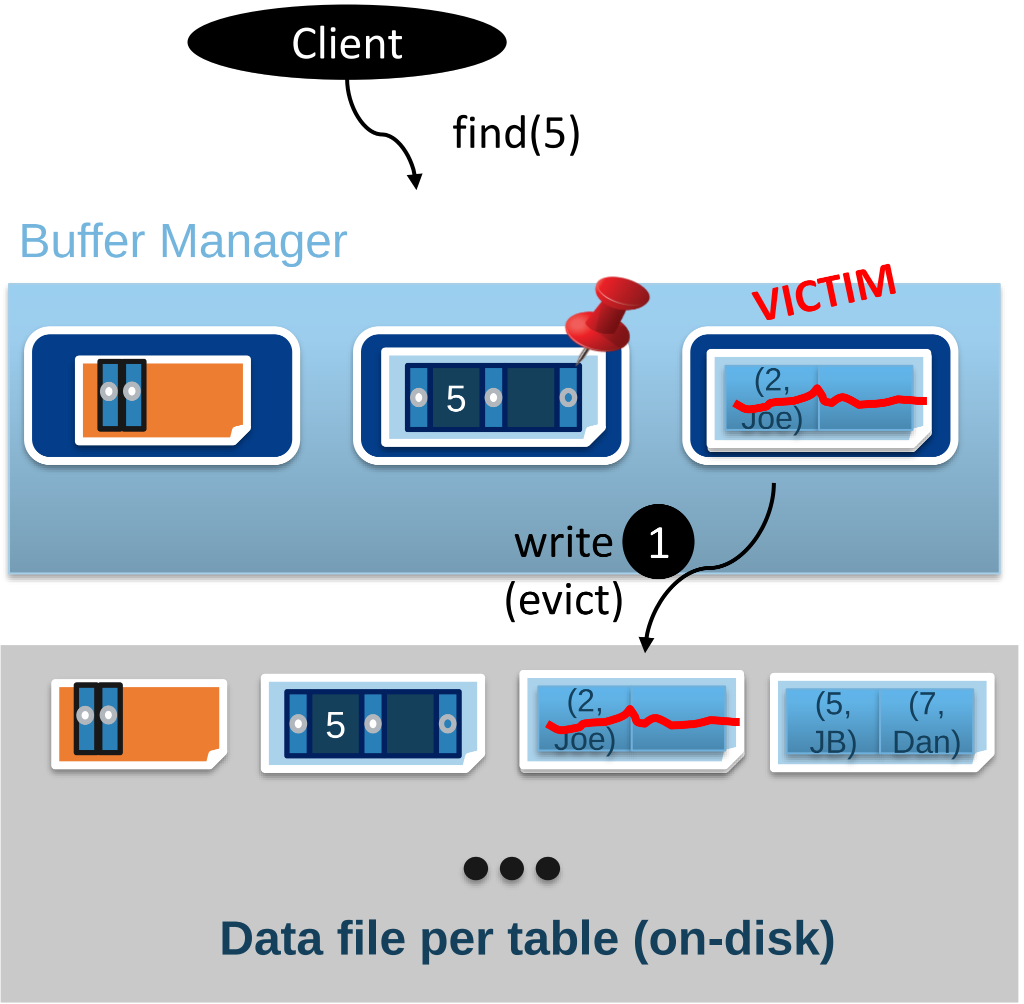
Buffer Management

- A dirty page is written to disk when the page is selected as a victim page for eviction by the LRU policy.
- Example (continue…)
  - "find key 5" attempts to read the root page.
  - The root page is found in the buffer (cache-hit)



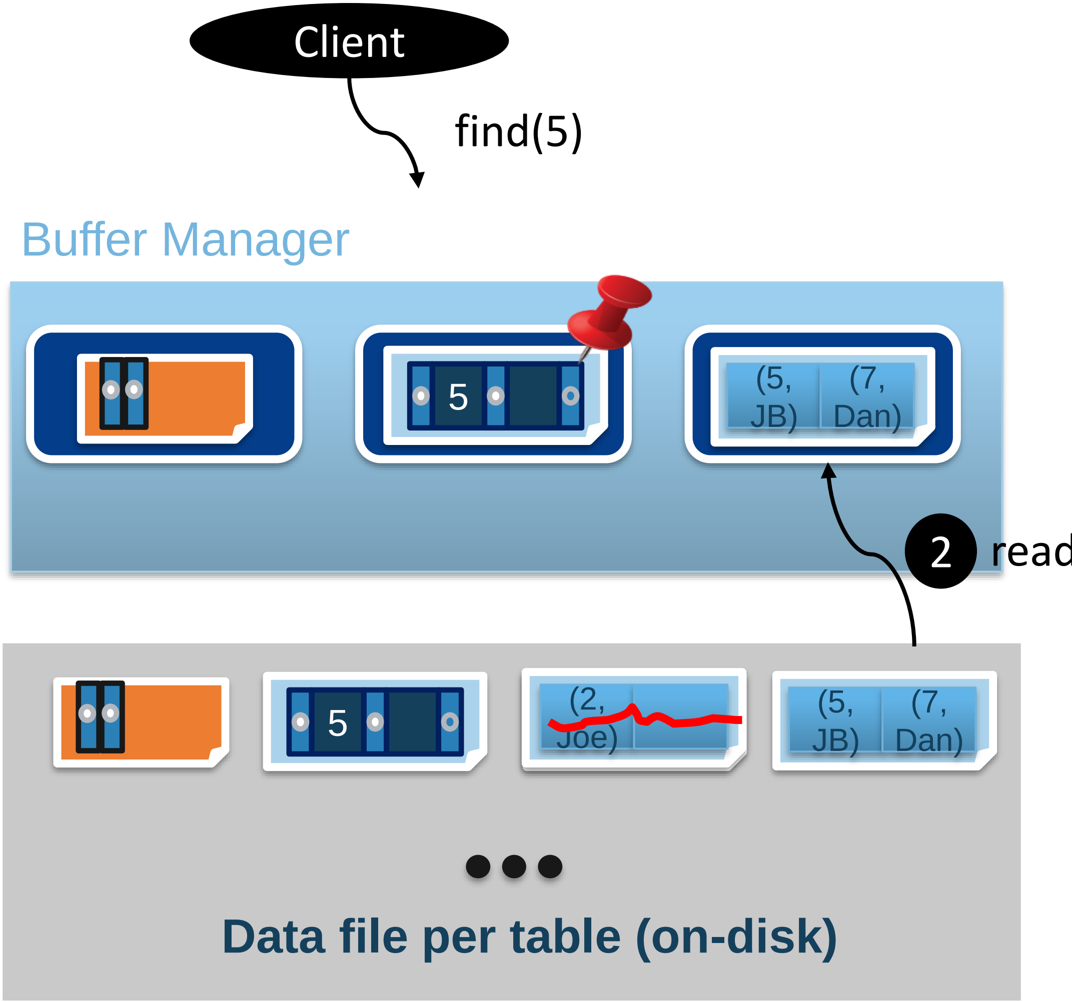
Buffer Management

- Example (continue…)
  - Traversing to the leaf, attempts to read the leaf page (with keys 5, 7). However the page is not within the buffer, cache-miss occurs.
  - The buffer manager tries to fetch the page from disk but the buffer is full.
  - The root page is evicted since it’s (A) marked as dirty from the previous delete and (B) selected by the LRU policy as victim page. 1



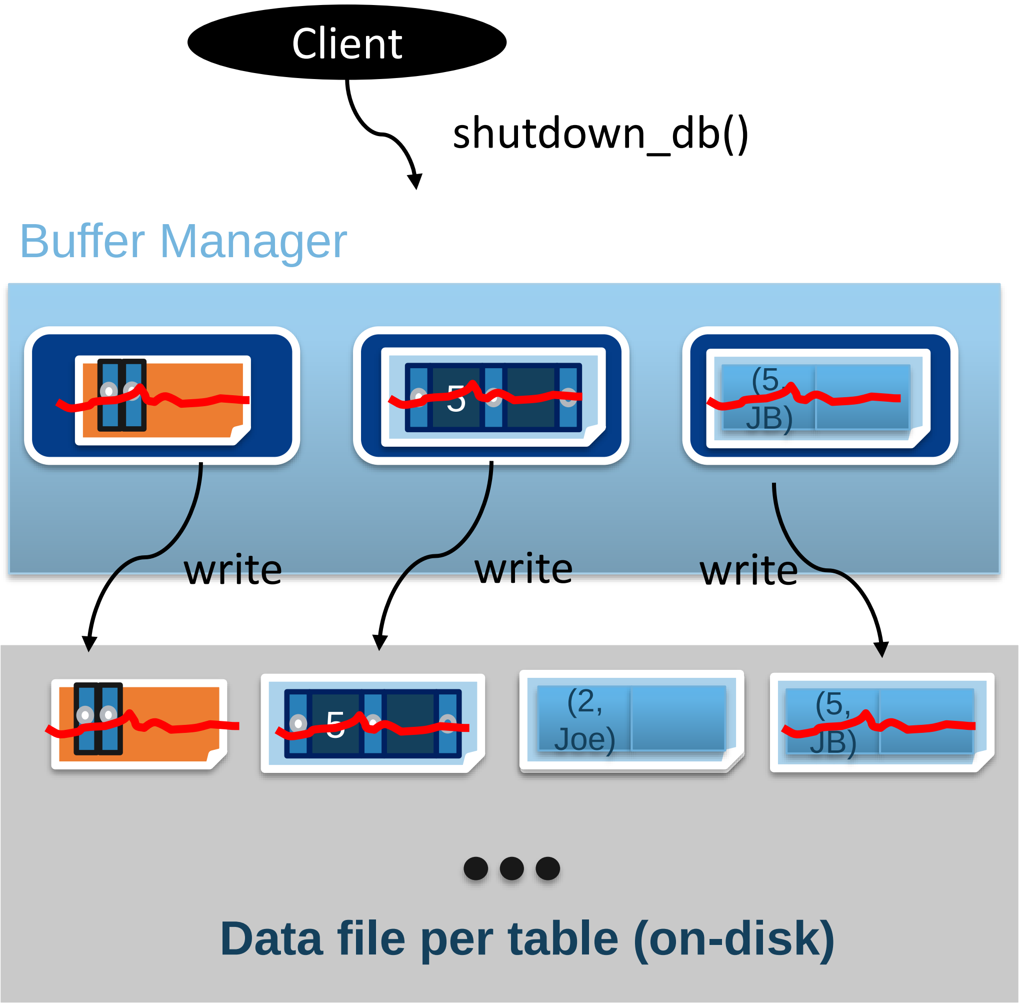
Buffer Management

- Example (continue…)
  - Traversing to the leaf, attempts to read the leaf page (with keys 5, 7). However the page is not within the buffer, cache-miss occurs.
  - The buffer manager tries to fetch the page from disk but the buffer is full.
  - The root page is evicted since it's (A) marked as dirty from the previous delete and (B) selected by the LRU policy as victim page. 1
  - Now, the corresponding leaf page can be read from disk to buffer. 2



Buffer Management

- shutdown_db() writes out all dirty buffer blocks to disk.
- This command can provide synchronous semantic (durability) to the user, but lose performance.



Submission

- Implement an in-memory buffer manager and submit a report about your design and implementation on Wiki.
  - Deadline: Nov 1 11:59pm
- We will score your project based on the last commit before deadline. Submissions afterwards will not be accepted.

Thank you