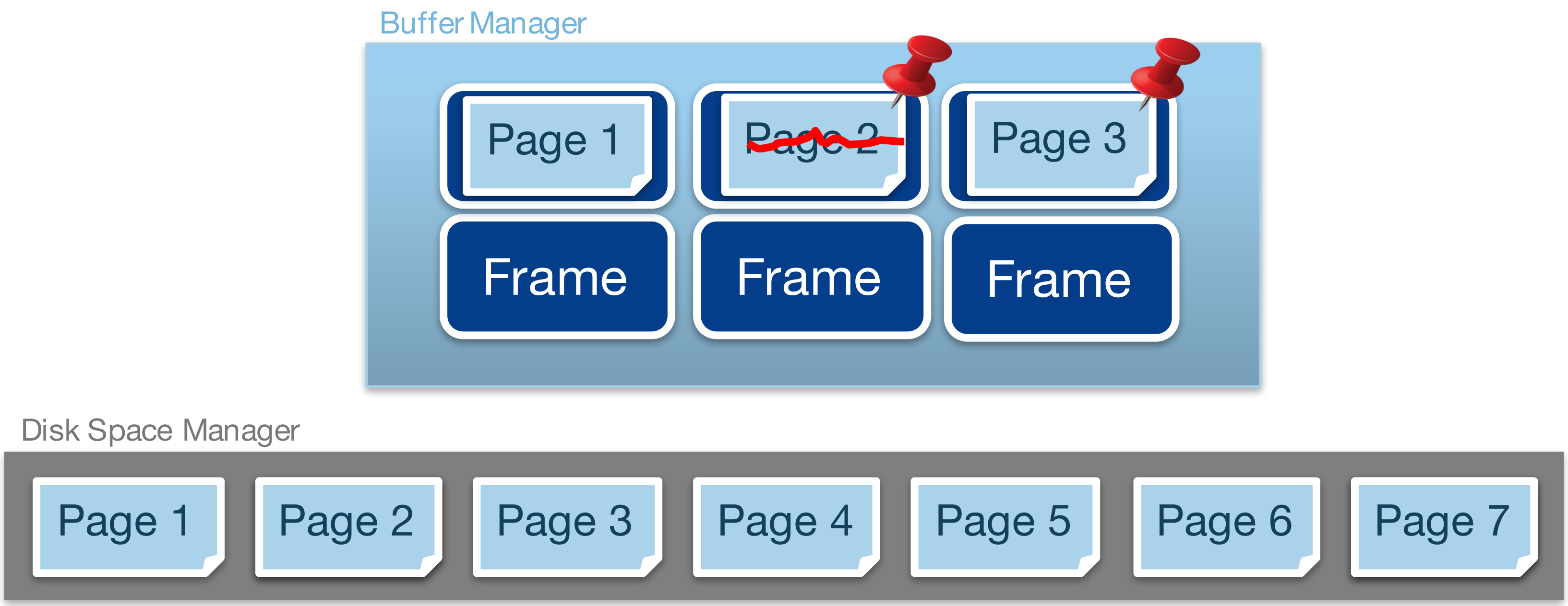
Project3

Buffer management

Scalable Computing Systems Laboratory
Hanyang University

Buffer Management

- Current disk-based b+tree doesn't support buffer management.
- Our goal is to implement **in-memory buffer manager** to cache on-disk pages.



Project Specification

- Define the buffer block structure, which should contain at least those fields.
  - **Physical frame**: containing up to date contents of target page. (you may define this field as a pointer to 4KB aligned frame)
  - **Table id:** the unique id of table (per file)
  - **Page offset**: the position of target page within a file.
  - **Is dirty:** if the buffer is modified, write this page during eviction.
  - **Pin count:** if the buffer is pinned (pin count > 0), do not evict this page.
  - **LRU list next (prev)** : buffer blocks are managed by LRU list.
- Note that this design is not mandatory. You can implement your own buffer manager with your design. (but, **DO NOT CHANGE** the disk format for compatibility)

Buffer Structure

frame (page size : 4096 bytes)
table_id
page_offset
is_dirty
pin_count
next/prev of LRU

Project Specification

➢ Implement database initialization function.

- **int init_db (int num_buf);**
- Allocate the buffer pool (array) with the given number of entries.
- Initialize other fields (state info, LRU info..) with your own design.
- If success, return 0. Otherwise, return non-zero value.

➢ Modify previous **open_db** interface to **open_table**

- **int open_table (char *pathname);**
- Open existing data file or create one if not existed.
- If success, return the **unique table id**, which represents the own table in this database. (Return negative value if error occurs)
- You have to maintain a table id once open_table() is called, which is matching file descriptor or file pointer depending on your previous implementation. (table id $\geq$ 1 and maximum allocated id is set to 10)

Project Specification

- Table id is also used in previous **insert, delete, find** interfaces as well. Modify those to table APIs.
  - **int insert (int table_id, int64_t key, char * value);**
  - **char * find (int table_id, int64_t key);**
  - **int delete (int table_id, int64_t key);**
- Your existing APIs (insert, delete, find) must work with implemented buffer manager first before accessing to disk. (more details in next slides)
  - If the page is not in buffer pool (cache-miss), read page from disk and maintain this page in buffer block.
  - Page modification only occurs in-memory buffer. If the page frame in buffer is updated, mark the buffer block as dirty.
  - According to LRU policy, least recently used buffer is the victim for page eviction. Writing page to disk occurs during LRU page eviction.

Project Specification

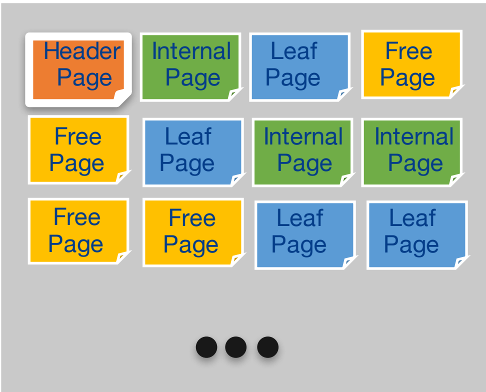
- Implement **close_table** interface.
  - **int close_table(int table_id);**
  - Write all pages of this table from buffer to disk and discard the table id.
  - If success, return 0. Otherwise, return non-zero value.
- Implement database shutdown function.
  - **int shutdown_db();**
  - Flush all data from buffer and destroy allocated buffer.
  - If success, return 0. Otherwise, return non-zero value.

Project Specification

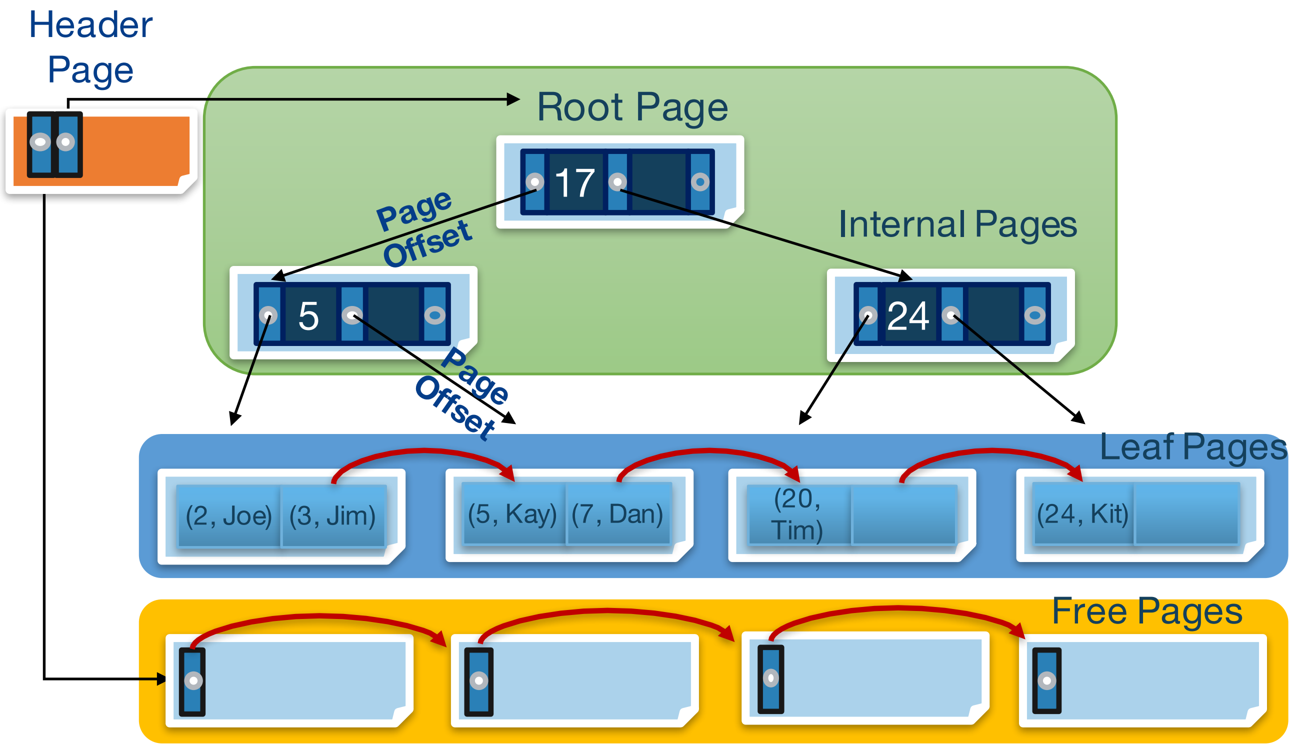
- Your library (libbpt.a) should provide those API services.

1. **int init_db (int buf_num);**
   - Initialize buffer pool with given number and buffer manager.
   - Below table APIs should be called after init_db() is called.
2. **int open_table (char * pathname);**
   - Open existing data file using 'pathname' or create one if not existed. If success, return **table_id**.
3. **int insert (int table_id, int64_t key, char * value);**
4. **char * find (int table_id, int64_t key);**
5. **int delete (int table_id, int64_t key);**
6. **int close_table(int table_id);**
   - Write the pages relating to this table to disk and close the table.
7. **int shutdown_db(void);**
   - Destroy buffer manager.

So far..

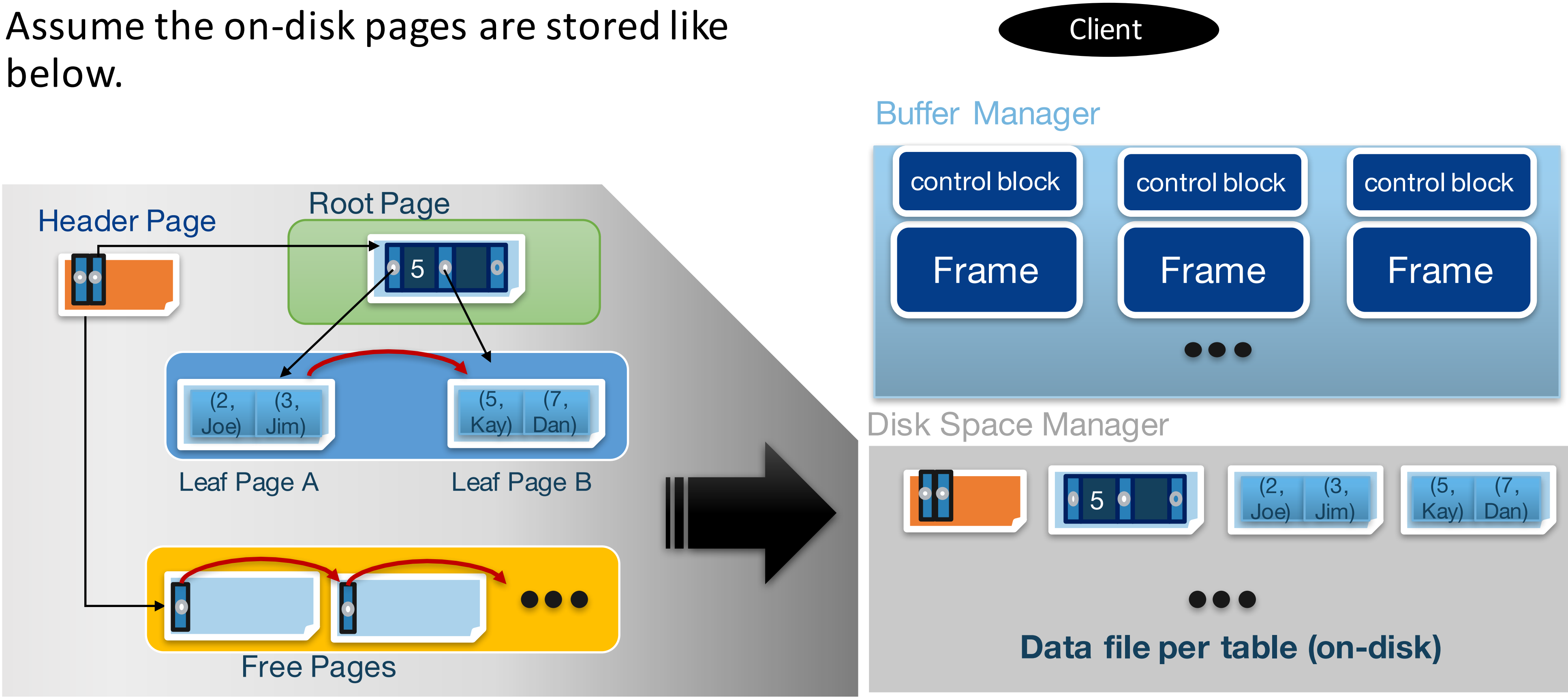


Your Data File (Example)



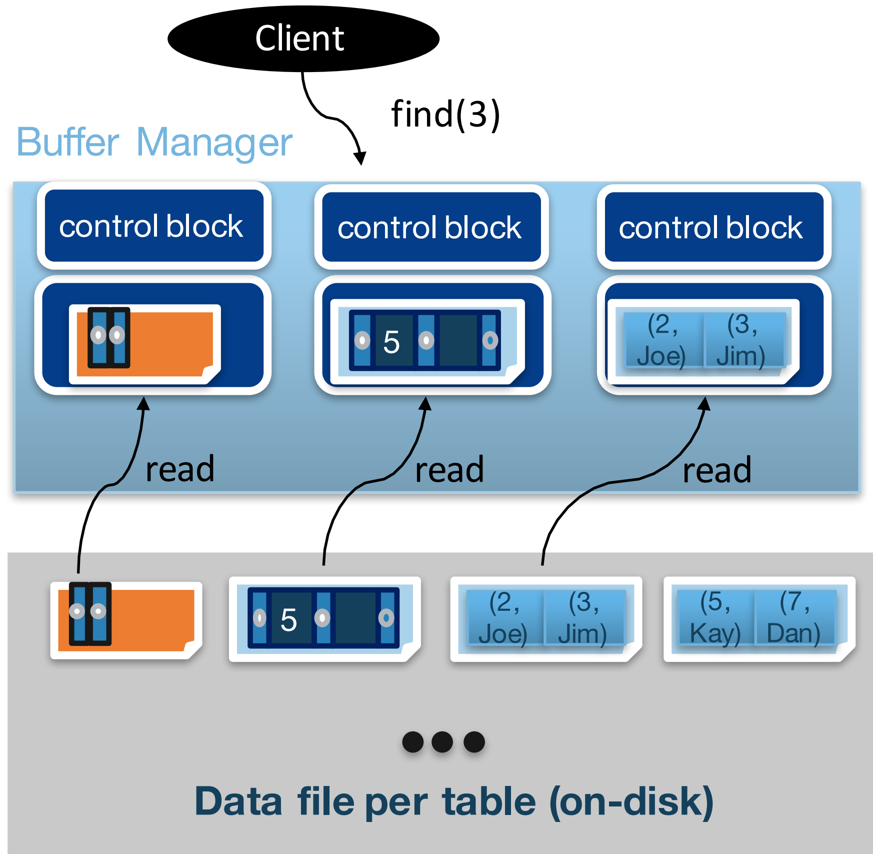
Buffer Management Example

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



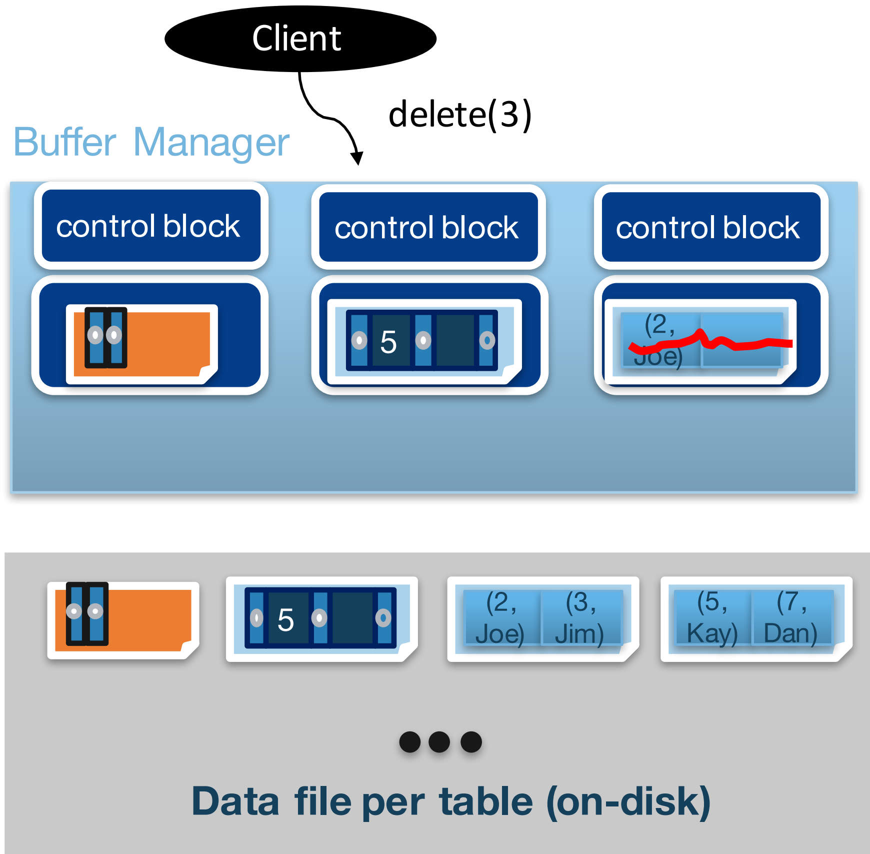
Buffer Management Example

- First, search the page from the buffer pool.
- If the page is not in the buffer pool (i.e, cache-miss occurs), read the page from disk and maintain this page in buffer block.
- While indexing from root to leaf page A (where key 3 is located), header page and root page (internal page) are also read by buffer manager.



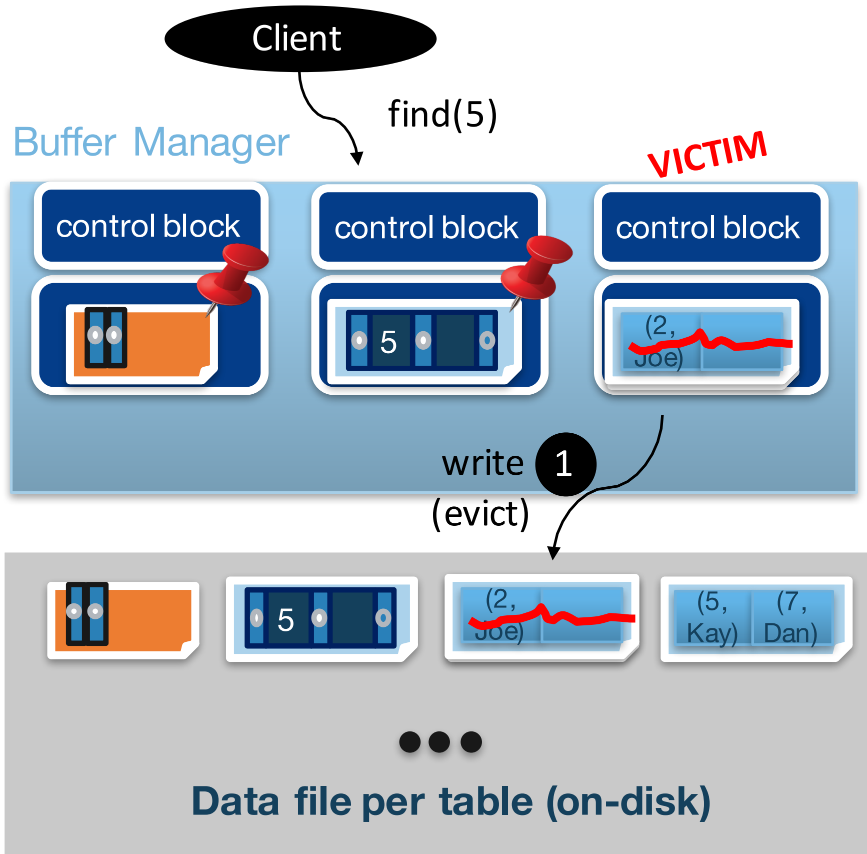
Buffer Management Example

- After reading page to buffer, update operation can be handled in buffer (in memory).
- So delete key 3 operation occurs in buffer, which makes that page marked to dirty.



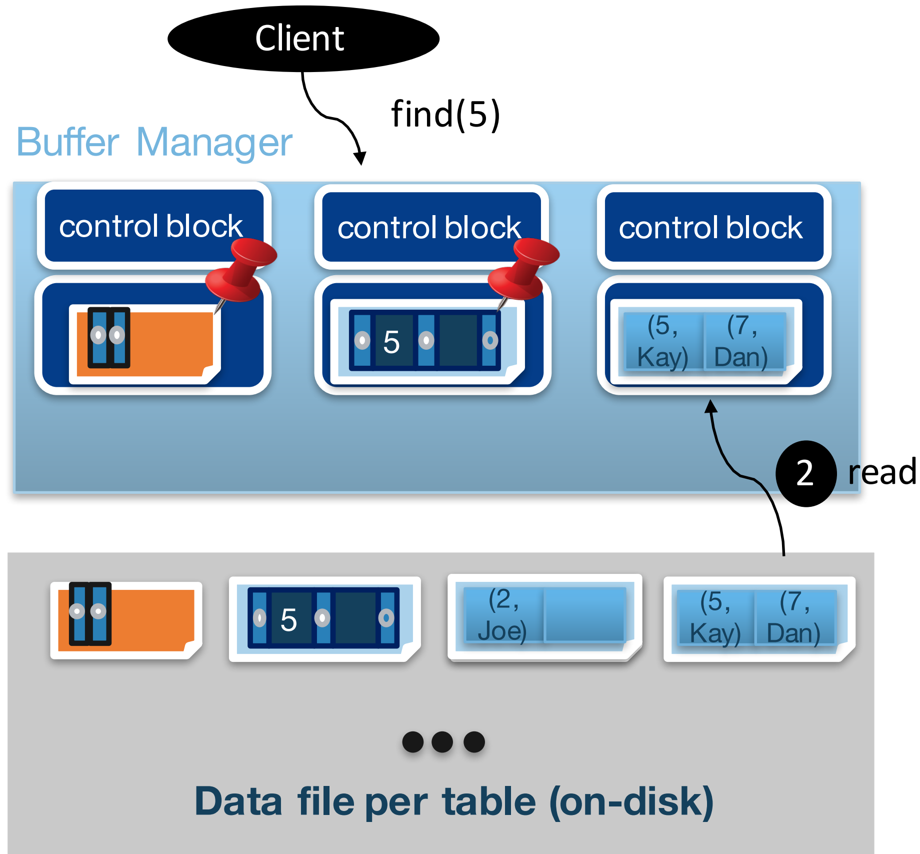
Buffer Management Example

- Dirty page is written to disk when this page is selected to the victim of LRU policy.
- Assuming example shown left, find(5) tries to read leaf page B which triggers page eviction. (Pinned page should not be the victim of eviction.)
- If the victim page is marked as dirty, write data to disk first. 1



Buffer Management Example

- Dirty page is written to disk when those page is selected to the victim of LRU policy.
- Assuming example shown left, find(5) tries to read leaf page B which triggers page eviction. (Pinned page should not be the victim of eviction.)
- If the victim page is marked as dirty, write data to disk first. 1
- Then read another page from disk. 2



Buffer Management Example

- close_table() or shutdown_db() writes out all dirty buffer block to disk.
- close_table() writes out the pages which are relating to given table_id.
- This command can provide synchronous semantic (durability) to user, but loses performance.