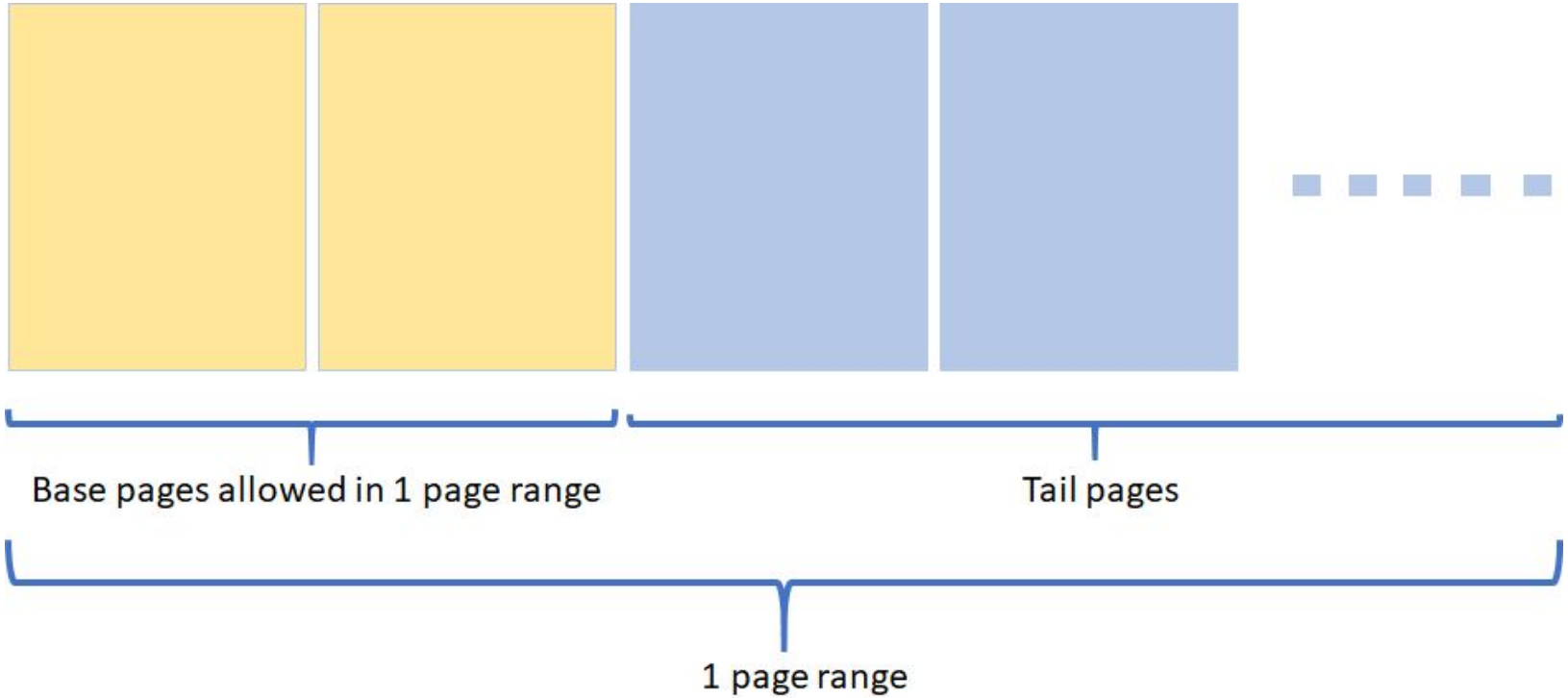
Milestone 2

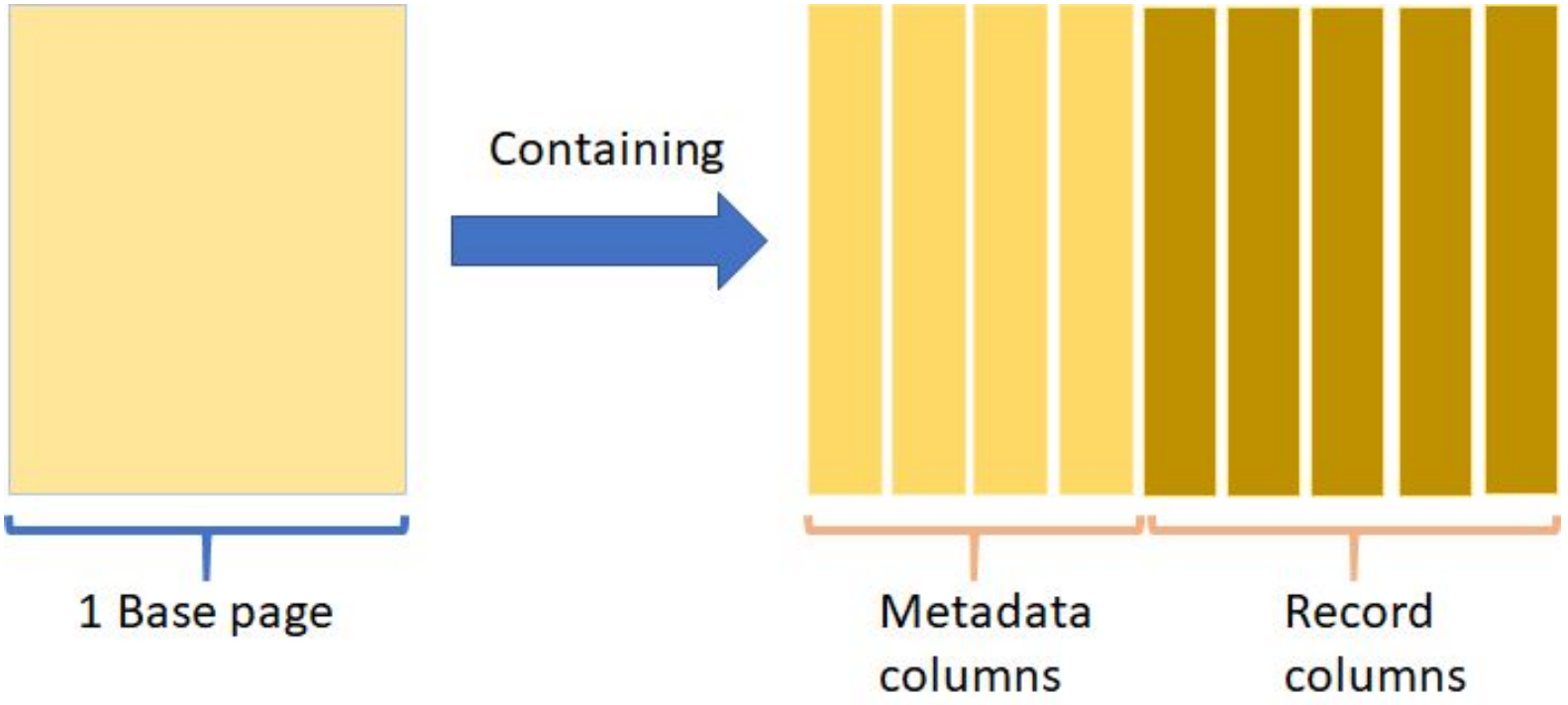
Group JeLLY-B

Jessica LaCourse
Lingxiao Meng
Lisa Malins
Yinuo Zhang
Benjamin Rausch

Page range



L-store structure



Durable Storage

All data are now pushed to the hard disk as needed

- Upon creation of a new database, a bufferpool is now initialized with the new table
- The data is pickled, or serialized, to improve space efficiency on disk
- Built-in functionality ensures enough space is allocated on disk for the new file before flushing from bufferpool to disk
- Files stored at page-level

Storage Structure

- Each page range stored in its own file
- Separation of concerns inspired by department stores: Take a Number!
  - Each “physical page” object is created with its table’s name and the index of its page range.
  - On creation, it asks the DB’s bufferpool for some space in its page range’s file.
  - The bufferpool returns the first offset in that file where a new page could be written.
- Physical page objects only store the “address” of its data on disk. It’s the bufferpool that accesses the data, going to disk if necessary.

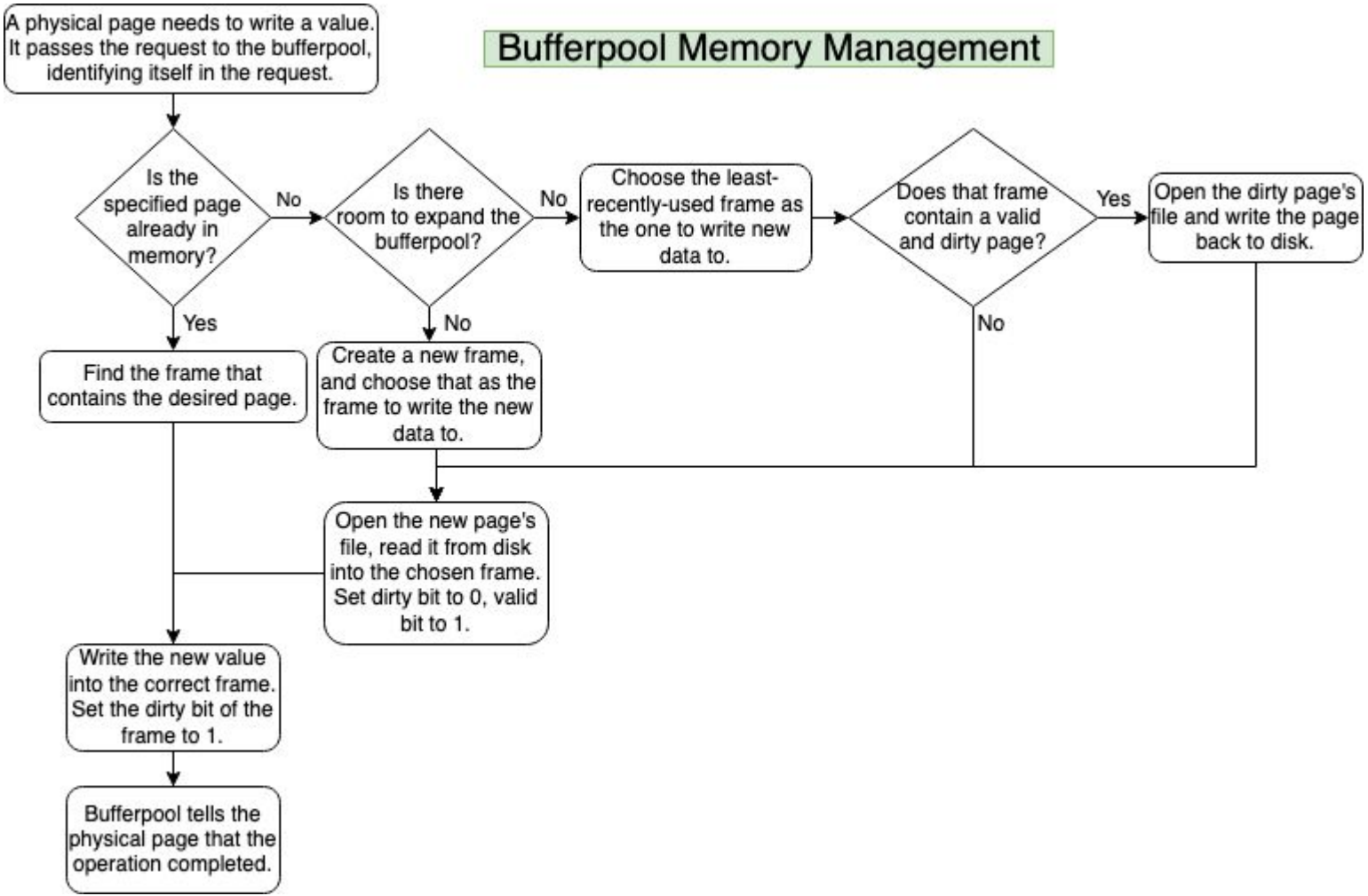
Persistence

- If the database is closed, it destroys certain unnecessary structures, asks the bufferpool to persist dirty pages to disk, then pickles its members to a “backup” file on disk.
- Whenever the database is opened, it checks for this backup file.

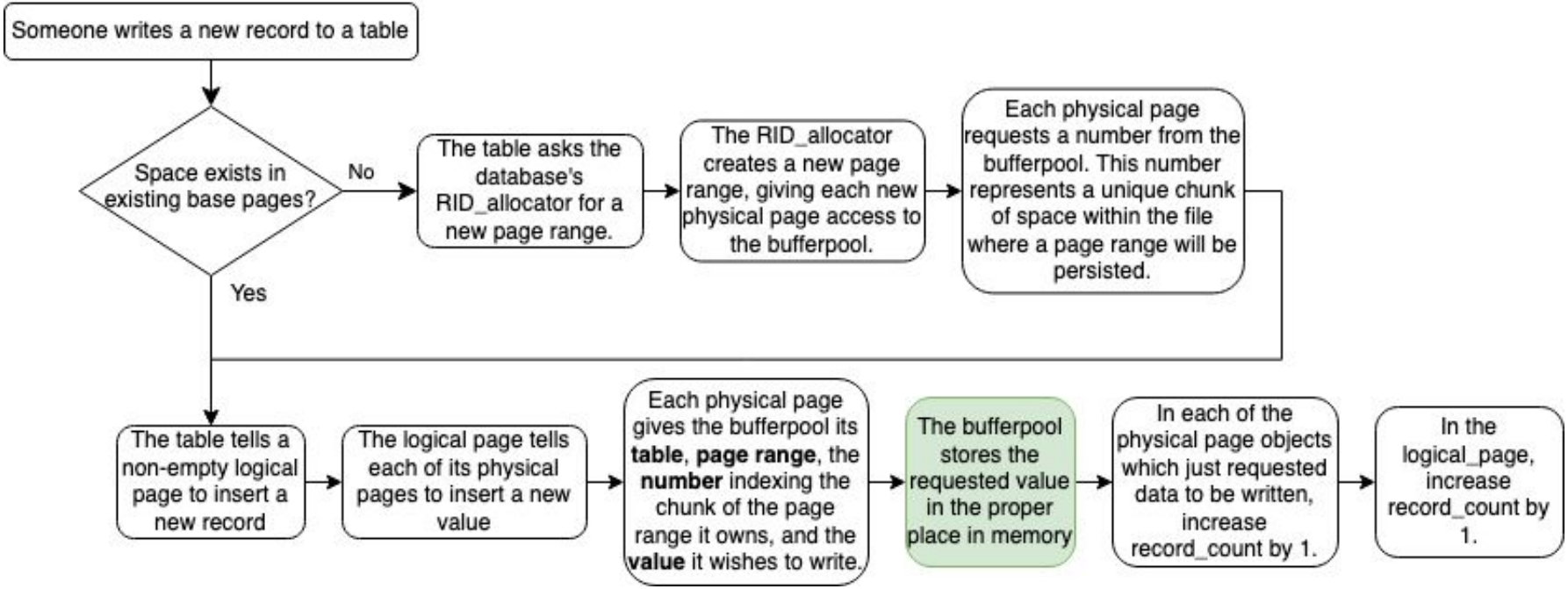
Bufferpool

- Built to improve efficiency for read/write requests
  - Cache will be much quicker than referencing hard disk
- Each page brought into the bufferpool will be assigned a frame, indexed, and pinned until the respective query (or queries) no longer need to reference the page
- Uses a Least Recently Used (LRU) Cache Implementation to evict the oldest pages in order to make space for newly requested pages
- Pages to be evicted, dirty or not, are flushed to disk

Bufferpool Memory Management



Storage Example: Insertion



Pre-Merge

- Tail records have metadata column for base RID to aid in merge
- Within the insertion function:
  Check_merge() stores page range ID's for full (complete) base pages
- Within the update function:
  Once a tail page is full, its ID will be added to merge queue
- Merge queue will pick the first page that meets the requirements:
  (1) Base pages within a given page range are full
  (2) The tail page to be merged is attached to the same page range
  (3) Otherwise tail pages are still stored in the queue (with FIFO order maintained)

Merge

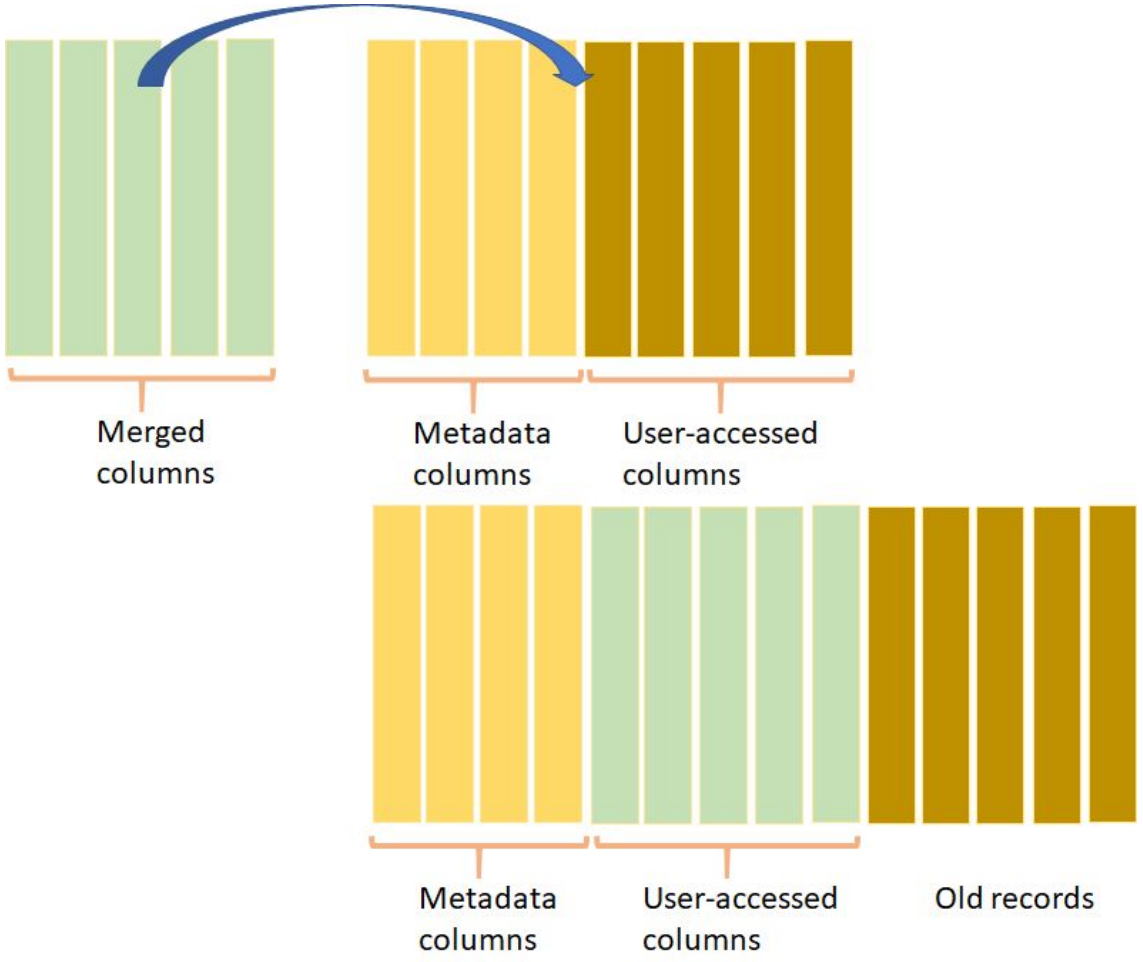
- Python threading functionality allows a merge thread in addition to the main thread
- These are core functions to ensure contention-free merges
  (1) Ensures read-only records and eliminates risk of read-write conflicts
  (2) Improves efficiency and accuracy by batching updates and pulling only the necessary pages for merge

Merge

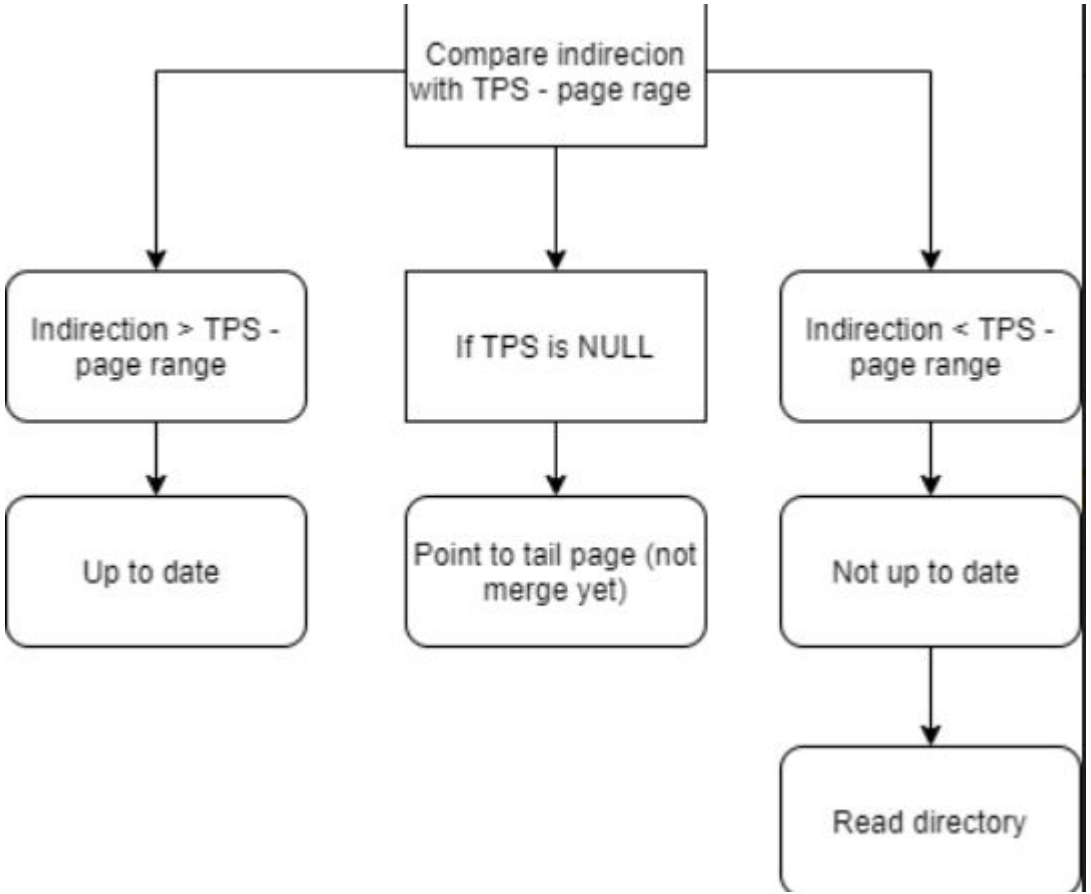
- Using cumulative updates, the latest tail entries will contain up-to-date information for their corresponding base records
- The original user data from base pages is copied
  - Metadata is left intact and untouched
  - Copy the most-up-to-date records if their base RID are marked updated
- Running concurrently with any user requests, merge() matches the updated records to their respective base record metadata
- The page directory is updated and a lock is placed while updating:
  - Users requesting information before and during the merge will receive older references
  - Users requesting information after the merge will receive up-to-date information

Page directory update

- Acquire lock during this process
- Releasing lock once finished
- Store TPS for merged page range
- (Original TPS is set to None)
- Old records are discarded



Selection


Compare indirecion with TPS - page rage
Indirection > TPS - page range
If TPS is NULL
Indirection < TPS - page range
Up to date
Point to tail page (not merge yet)
Not up to date
Read directory

Indexing on any column

Milestone 1 implemented index on primary key column only

Now able to index on any column

- Index on primary key column: only 1 RID per primary key
- Index on non-primary key column: may have multiple RIDs per key
- O(1) check to see if value exists in column
- Return all records with that value

Index

Insertion

- Let list_of_RIDs_for_this_value = self,data[column].get(value)
- If list_of_RIDs_for_this_value is none, get self.data[column][value]
- Update the insert data
- After this call, self.locate will return a list, which contains RID

Index

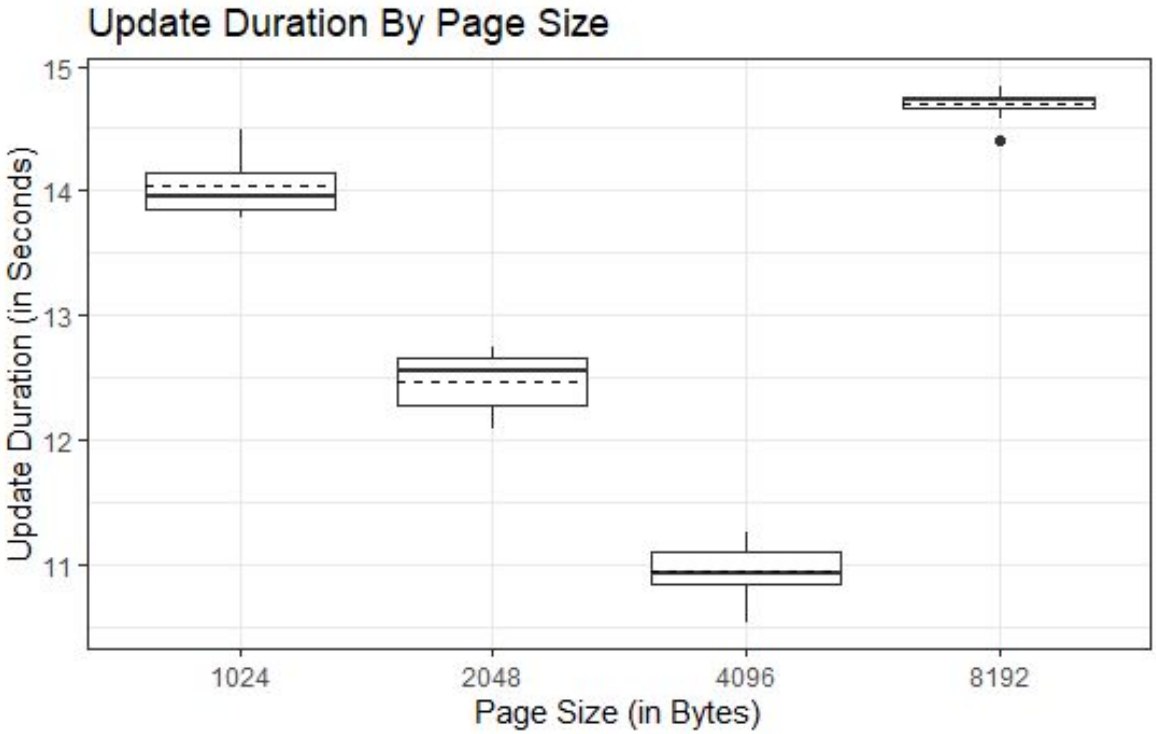
Deletion

- Use given value to get the RIDs
- Check the RIDs are associate with given value
- Remove these RIDs from index.
- After this call, self.locate will return a list, which does not contain RID

Performance

Optimal Page Size;
1,000 Records

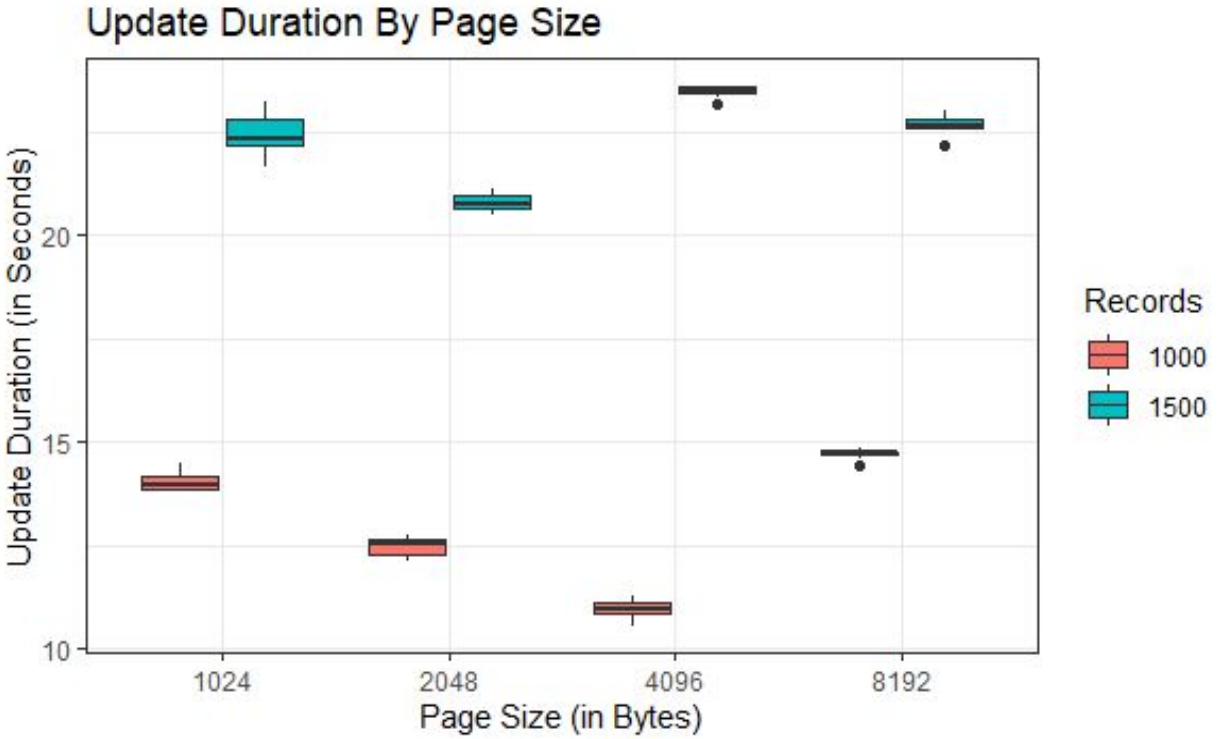
- Notable decrease in size up until 4kb/page
- Increase with 8kb/page showing impact when page(s) not entirely full



Performance

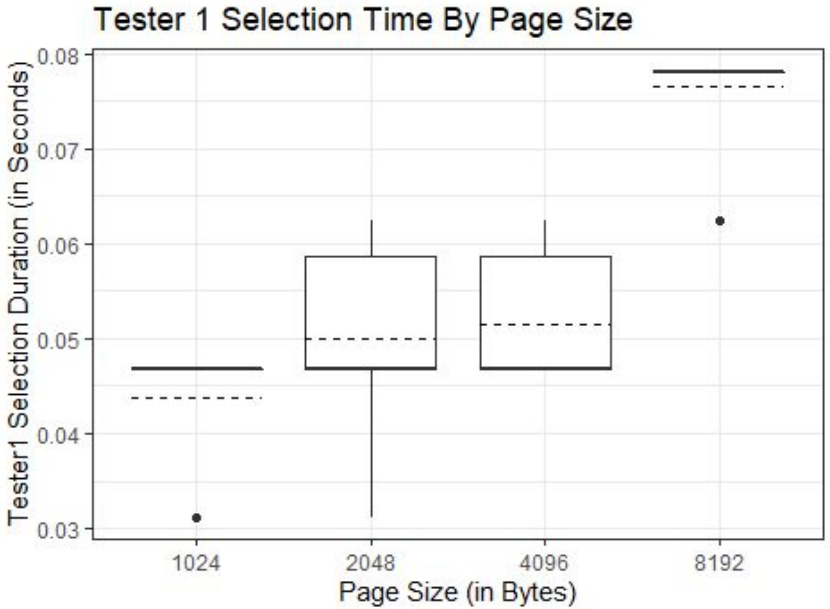
Optimal Page Size; Record Number Comparison

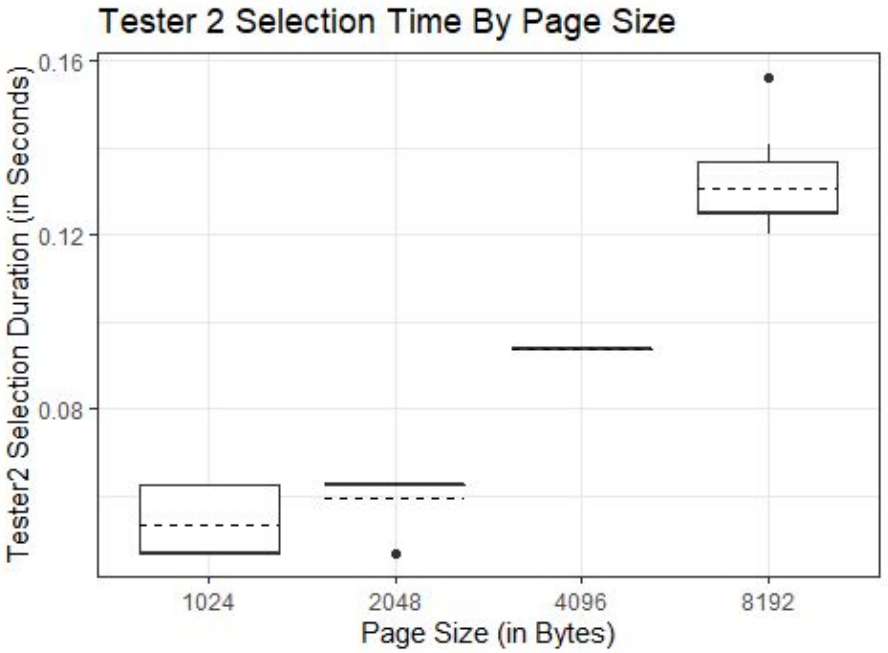
- Ratio between page size and number of records appears to directly affect performance
- Increases in the number of records (intuitively) increase processing time



Performance

Optimal Page Size; 1,000 Records





Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Installed RAM	16.0 GB