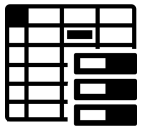
Milestone 1

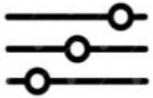
Dulce Torres, Kyle Pickle, Pranav Kode, Sean Nguyen, Ruqayyah Siddique

Our Project



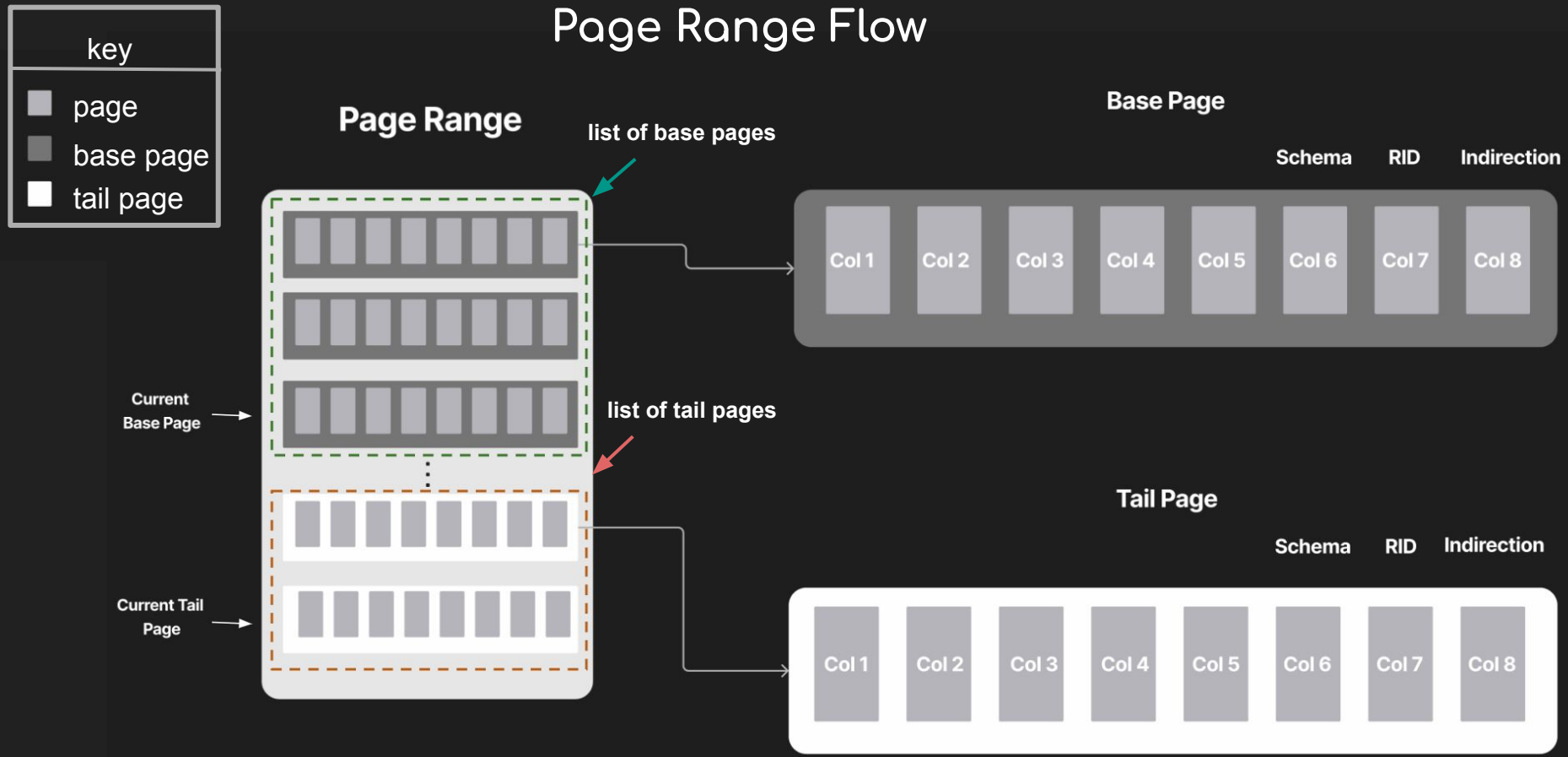
Data Model stores the Page Ranges with there base page, tail page, the schema columns in **columnar form**.

Bufferpool maintains data in memory, has page directory that **maps RIDs to pages in memory**

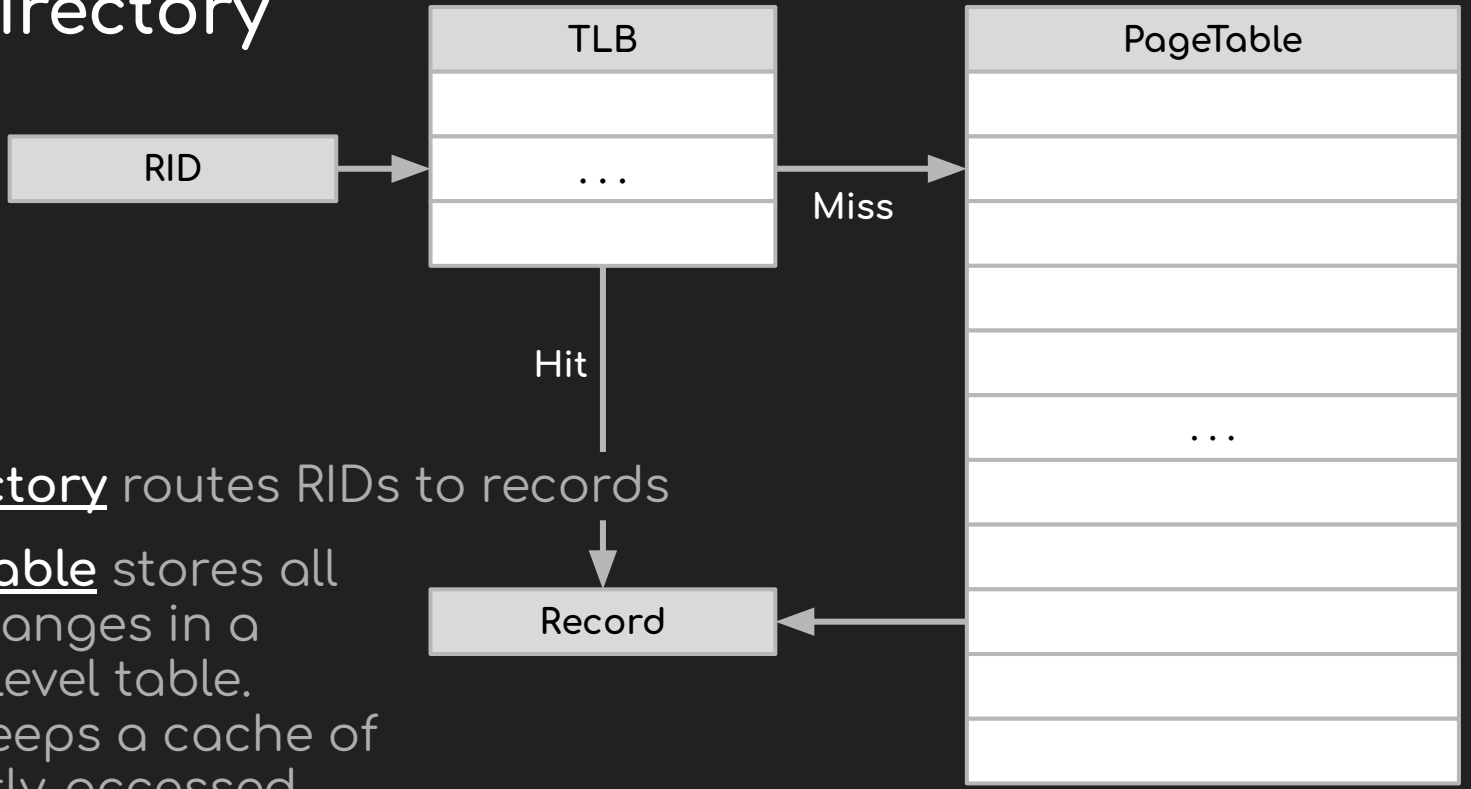


Query Interface improves discoverability of data, through querying capabilities

Page Range Flow



Page Directory



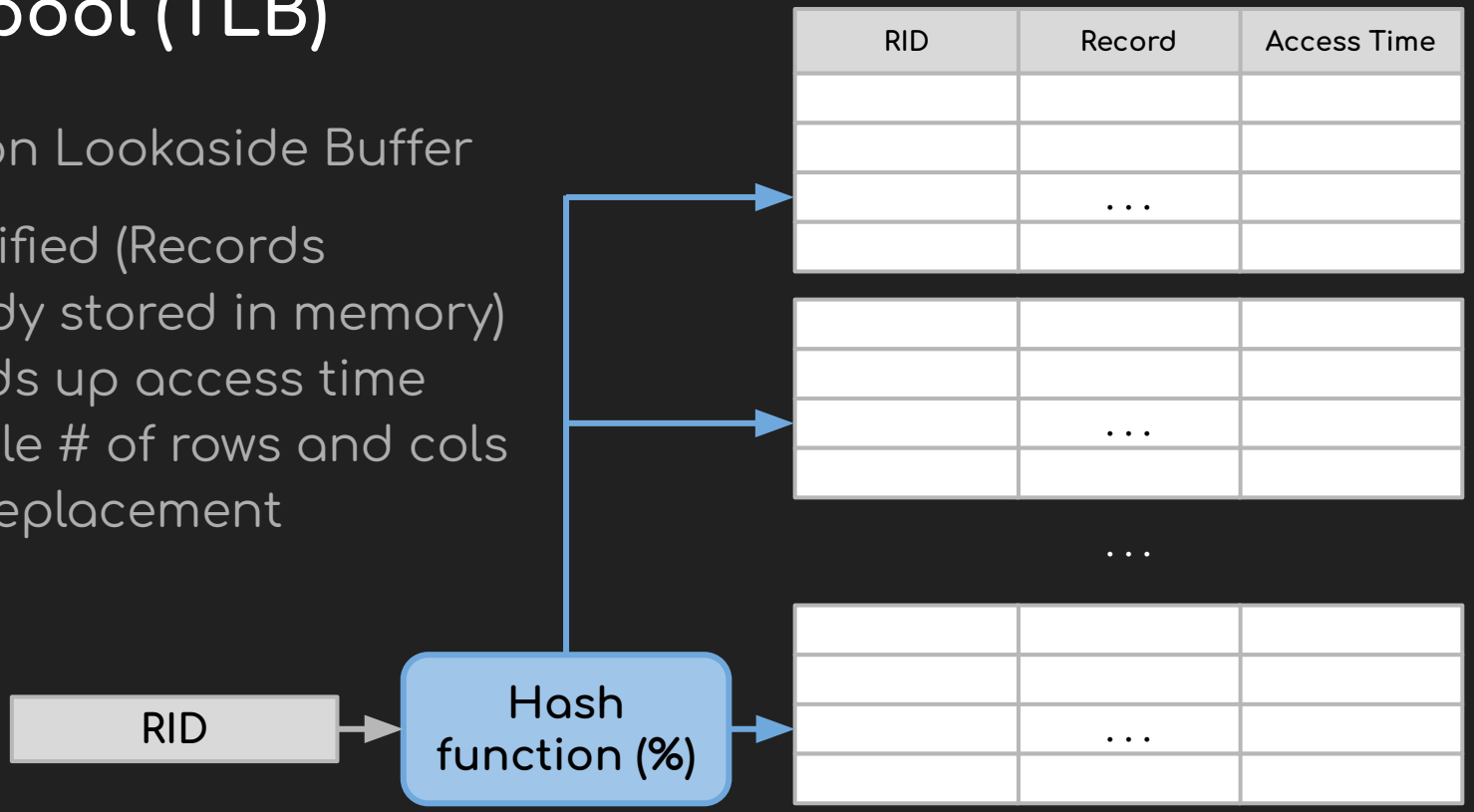
PageDirectory routes RIDs to records

- **PageTable** stores all PageRanges in a multi-level table.
- **TLB** keeps a cache of recently-accessed records for quick access.

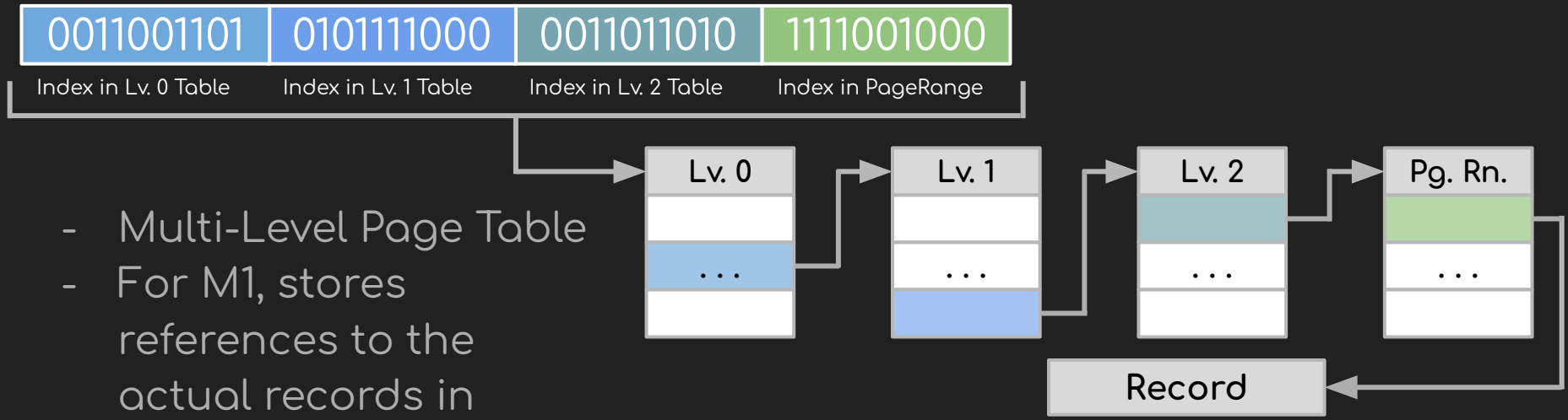
Bufferpool (TLB)

Translation Lookaside Buffer

- Simplified (Records already stored in memory)
- Speeds up access time
- Flexible # of rows and cols
- LRU replacement



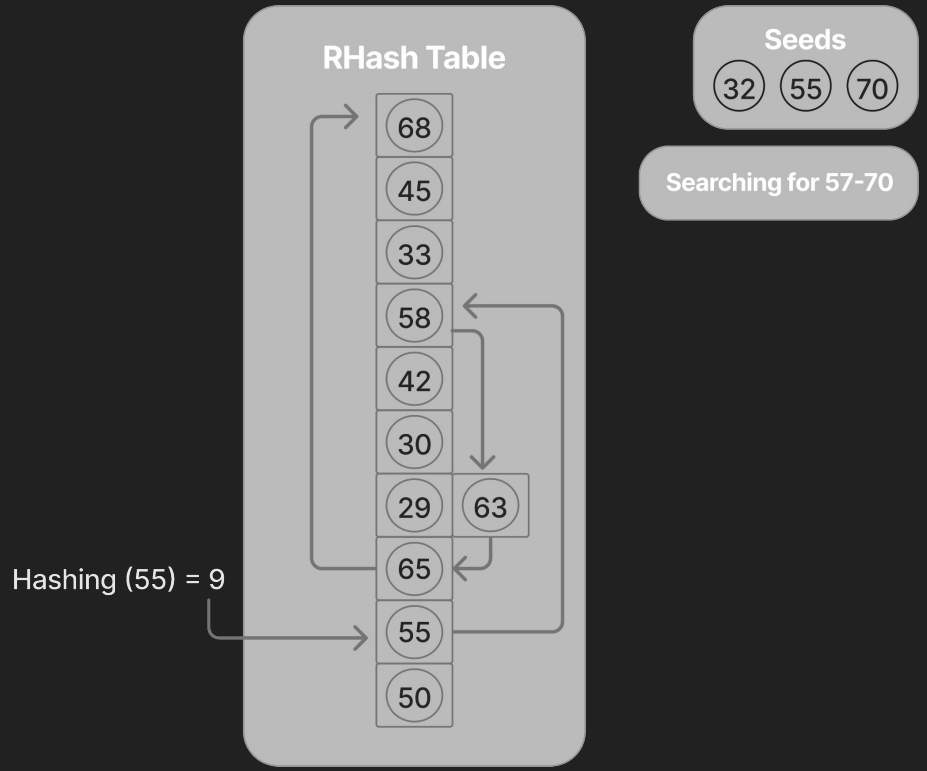
Page Table



- Multi-Level Page Table
- For M1, stores references to the actual records in memory
- Fixed-sized Pages make indexing much faster
- Multiple levels reduce storage cost with large #s of records

Index

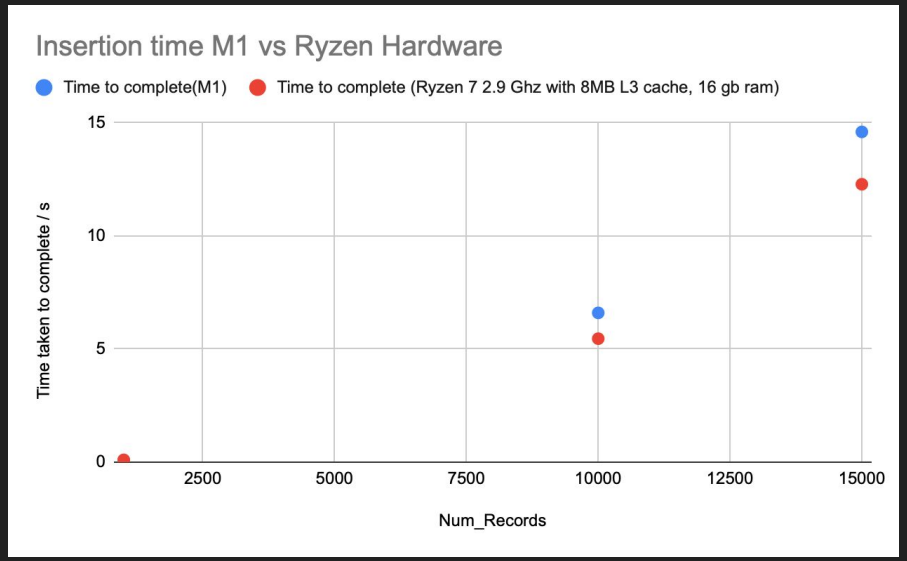
- RHash
  - Hash Table + ordered linked list
  - Map column values to Nodes
  - Node
    - RID set
    - Next value

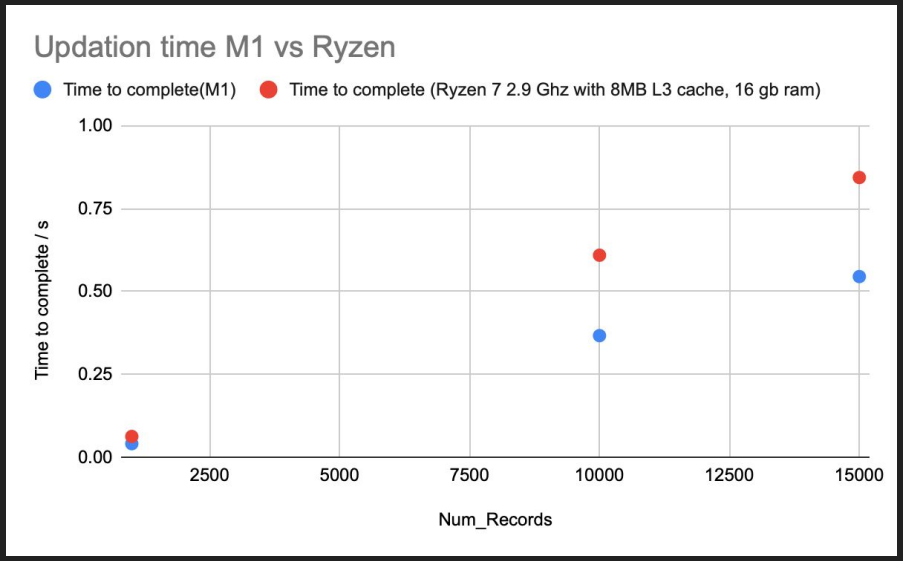


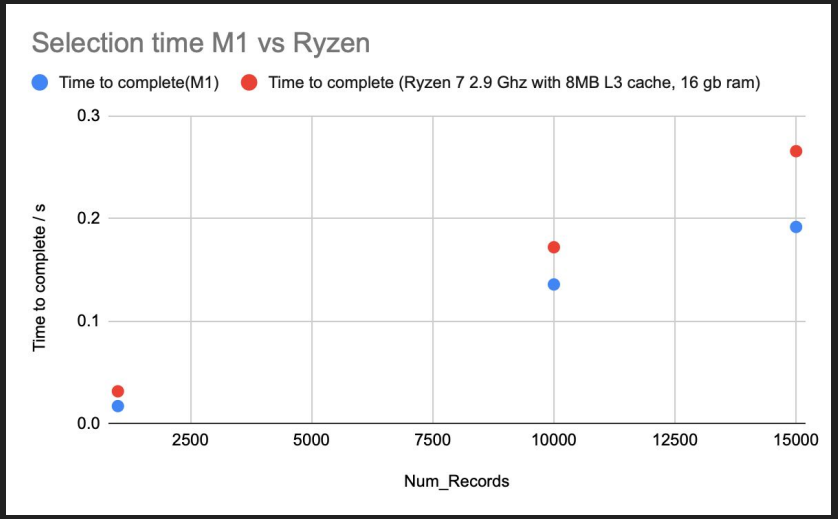
Query API

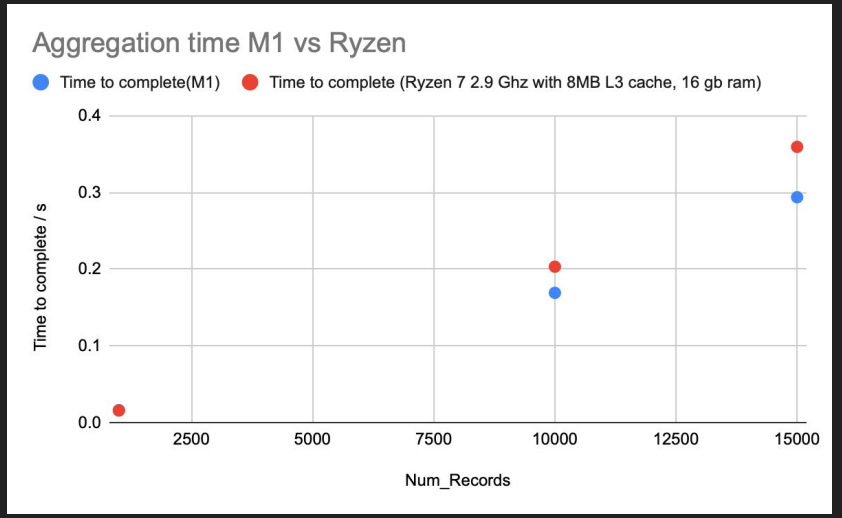
- Work through functions defined in the table class
- Delete
  - Use the primary key to get the RID, delete from the database, then update the index if necessary
- Insert
  - Check if the primary key already exists before inserting
- Select
  - Get all RIDs containing the desired value, then locate the record for each RID.
- Update
  - Check if the primary key already exists
- Sum
  - Do a range based search based on the index keys, get the appropriate column from each RID, then sum the list

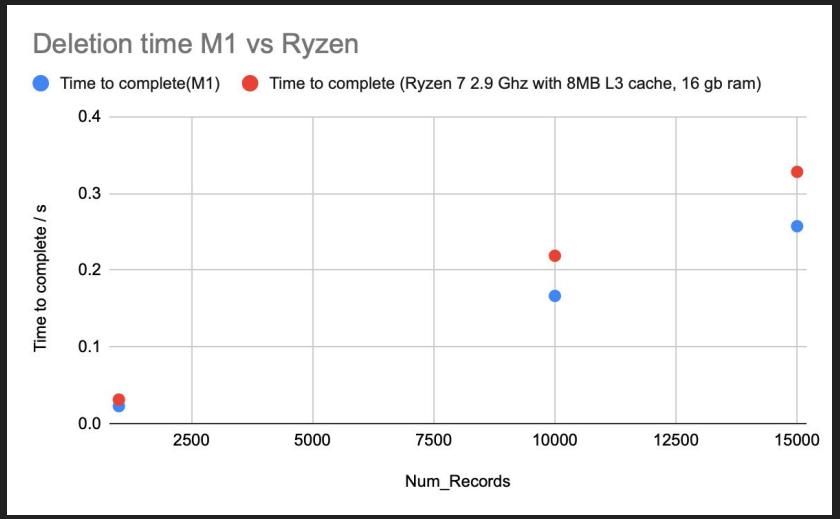
Performance on different hardwares






Selection time M1 vs Ryzen
Time to complete(M1)
Time to complete (Ryzen 7 2.9 Ghz with 8MB L3 cache, 16 gb ram)
Time to complete / s
0.0
0.1
0.2
0.3
2500
5000
7500
10000
12500
15000
Num_Records

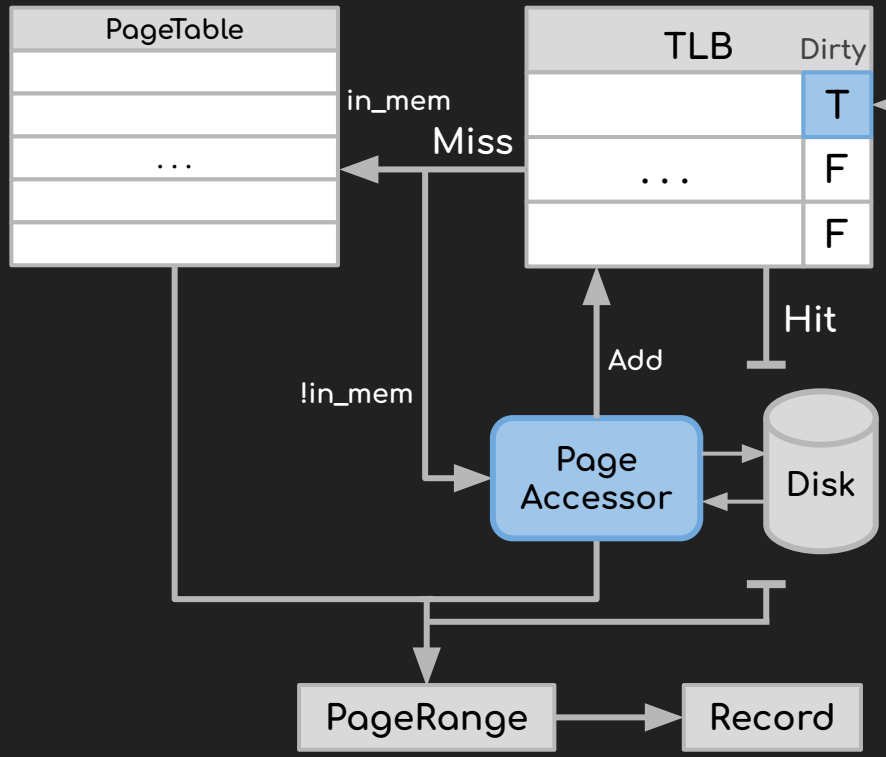

Aggregation time M1 vs Ryzen
Time to complete(M1)
Time to complete (Ryzen 7 2.9 Ghz with 8MB L3 cache, 16 gb ram)
Time to complete / s
0.0
0.1
0.2
0.3
0.4
2500
5000
7500
10000
12500
15000
Num_Records


Deletion time M1 vs Ryzen
Time to complete(M1)
Time to complete (Ryzen 7 2.9 Ghz with 8MB L3 cache, 16 gb ram)
Time to complete / s
0.0
0.1
0.2
0.3
0.4
2500
5000
7500
10000
12500
15000
Num_Records

Milestone 2

Dulce Torres, Kyle Pickle, Pranav Kode, Sean Nguyen, Ruqayyah Siddique, Wen Wei Tan

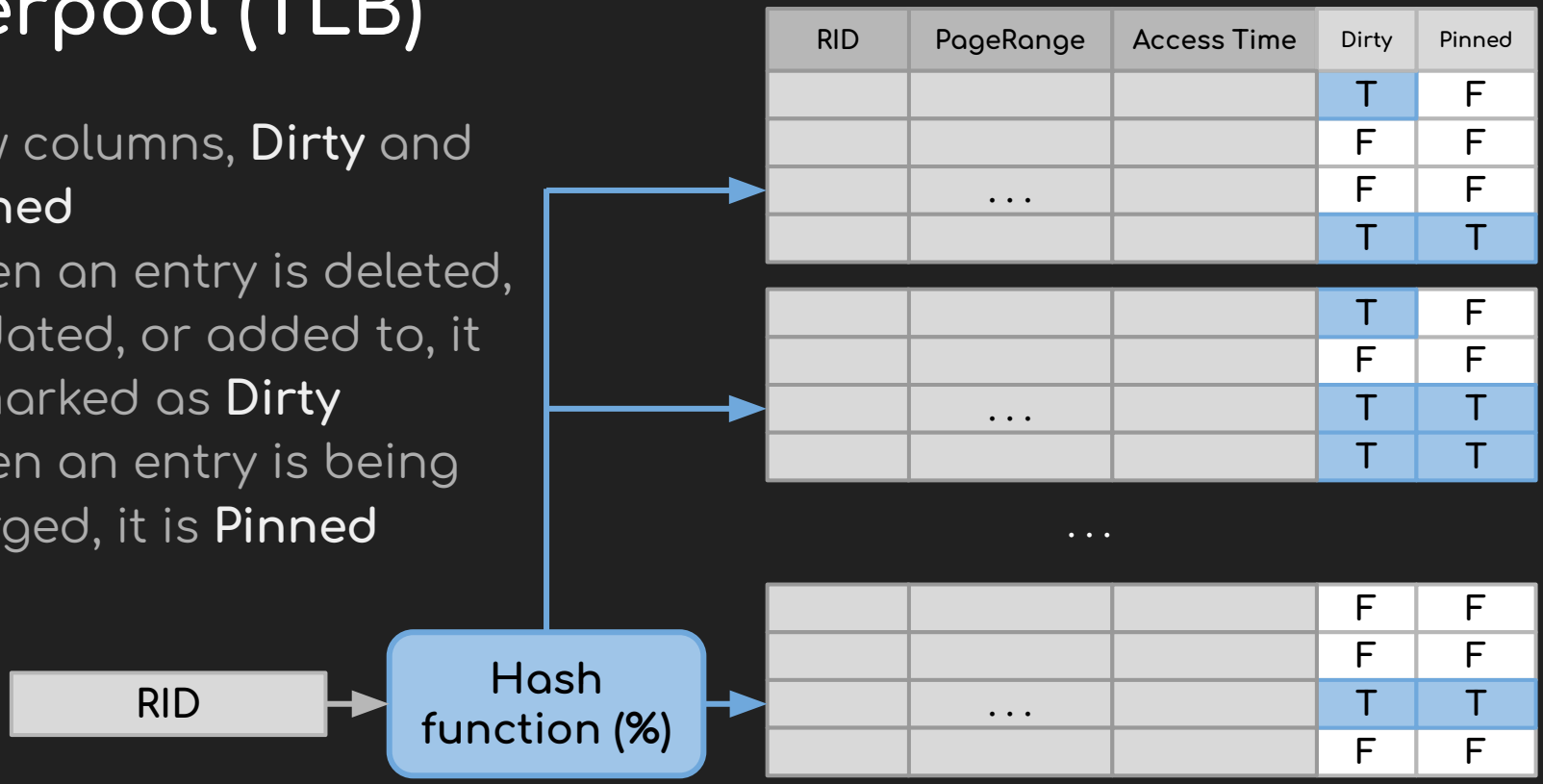
Durability & Bufferpool Extension



- New **PageDirectory** works at the granularity of **PageRanges**
- Two modes, in-memory and on-disk (automatic)
- **TLB** has been updated to work with **PageRanges**
  - **Dirty** and **Pinned** bits
- New **PageAccessor** works with **FileService**, pulling from and writing to disk

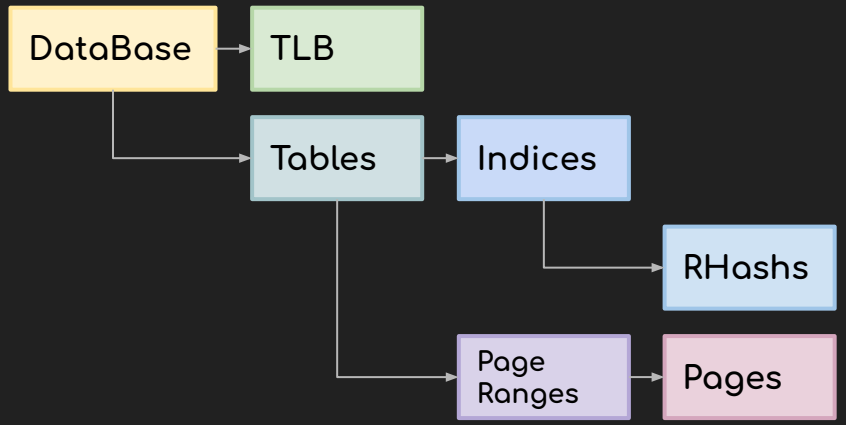
Bufferpool (TLB)

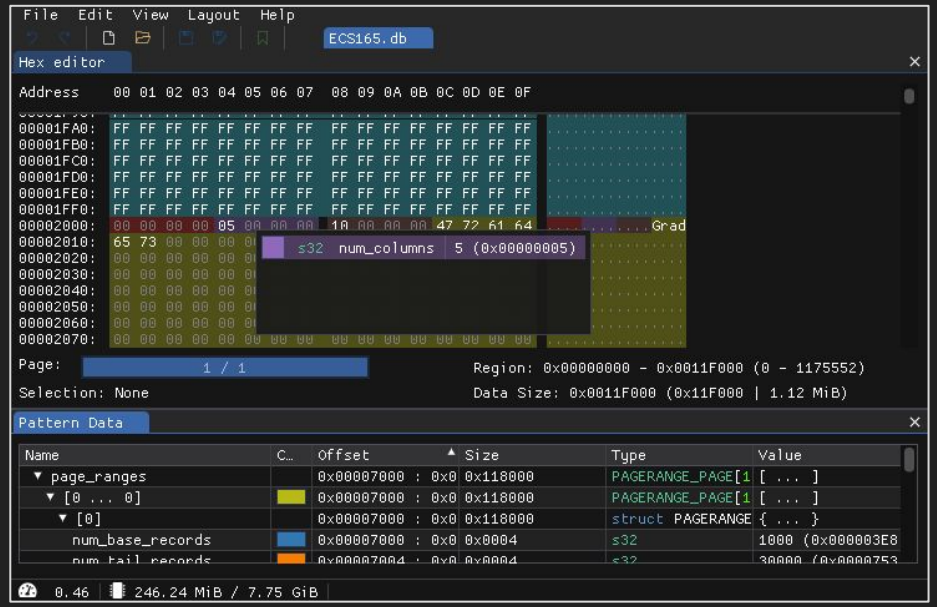
- New columns, **Dirty** and **Pinned**
- When an entry is deleted, updated, or added to, it is marked as **Dirty**
- When an entry is being merged, it is **Pinned**



FileService

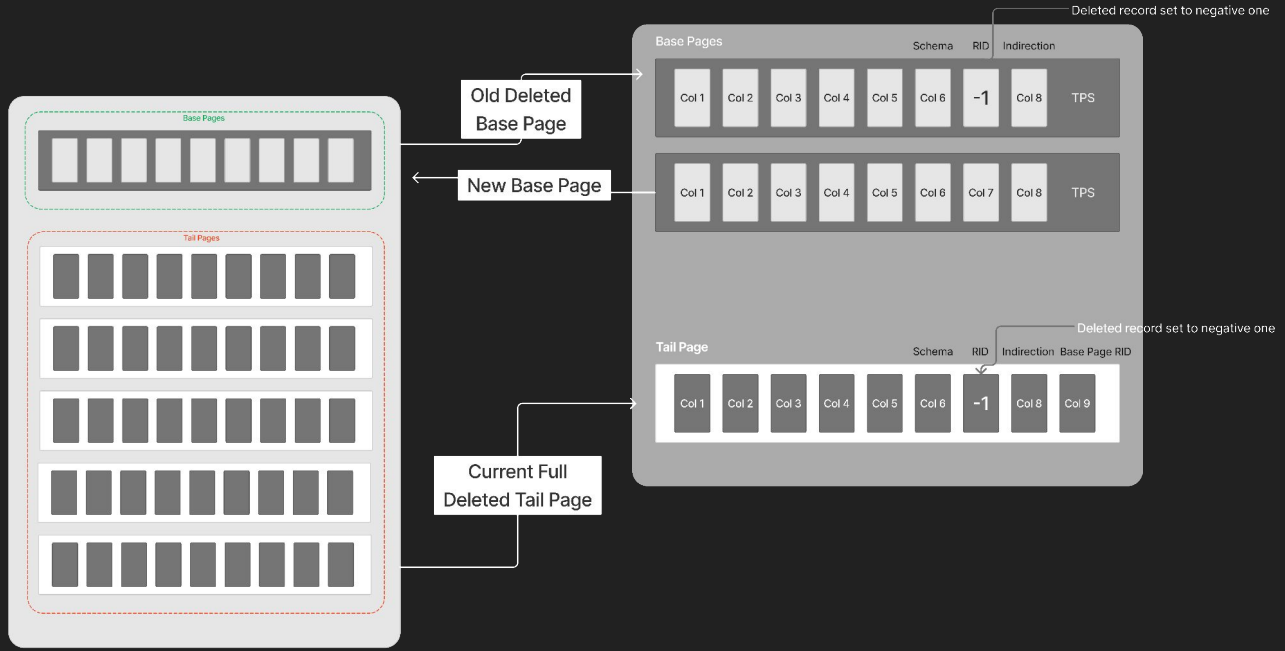
- **DataBase** is read from / written to a single .db file
  - No **Pickle** used; organized and parsed through from scratch
- **FileService** has 3 Functions:
  - load_tables
  - pull_page_range
  - merge_tables





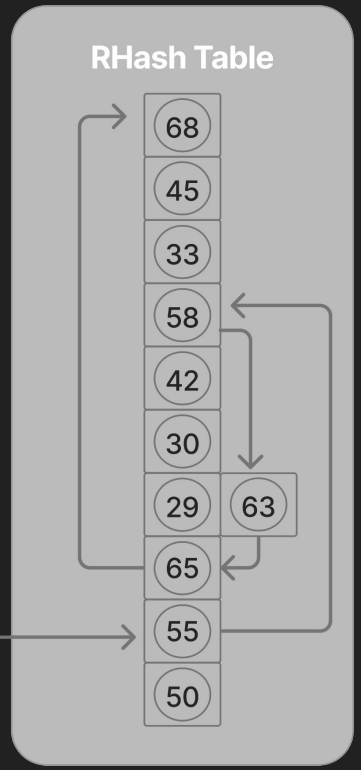
Merge

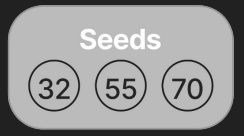
- Creates copy of new base page
- Updates the records of the old base page
- All prior tail records marked as deleted
- Old base page also marked as deleted
- Appends the new copy of the base page

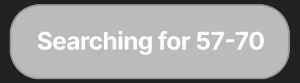


Indexing

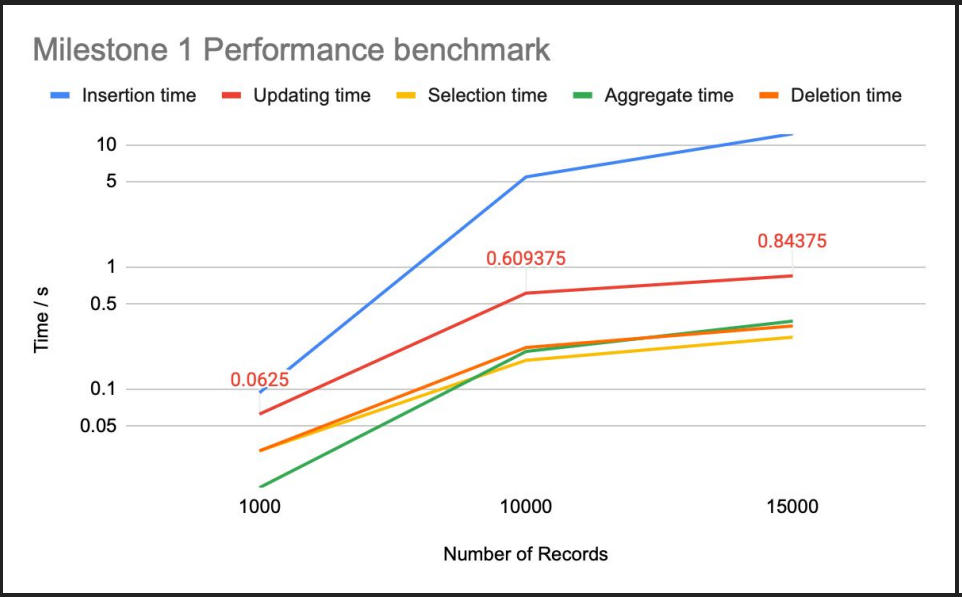
- Create_index
  - Create a new RHash object
  - Scan through all table records and call the index's insert function for the corresponding column
- Drop_index
  - Deletes the index of the specified column from memory
- New seeding scheme
  - 1% chance to add a seed after an index insert
  - Smallest value is always one of the seeds
  - Max number of seeds is 25% of the total number of records
- Indices are persisted in the .db file

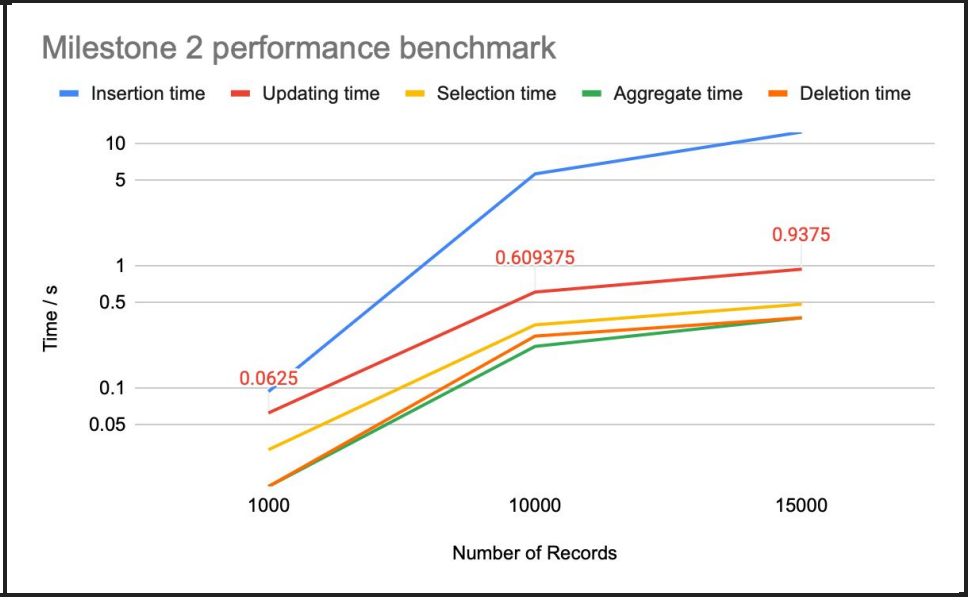






Milestone 1 vs Milestone 2





System: Ryzen 7 2.9 Ghz with 8MB L3 cache, 16GB ram
Workload: __main__.py

Milestone 3

Dulce Torres, Kyle Pickle, Pranav Kode, Sean Nguyen, Ruqayyah Siddique, Wen Wei Tan

Two-Phase Locking

- Use shared locks when data is read, allowing any other number of people to read but no one to write until released

- Use exclusive locks when data is written to, preventing anyone else from reading/writing until released locking process: **Acquire -> Execute -> Release**

- Transaction immediately aborts if it is unable to obtain a lock

- Obtains all locks before starting execution of queries

1st thread/ 2nd thread	Shared Lock (read)	Exclusive Lock (write)
Shared Lock (read)	✓	✗
Exclusive Lock (write)	✗	✗

Read-Write Lock

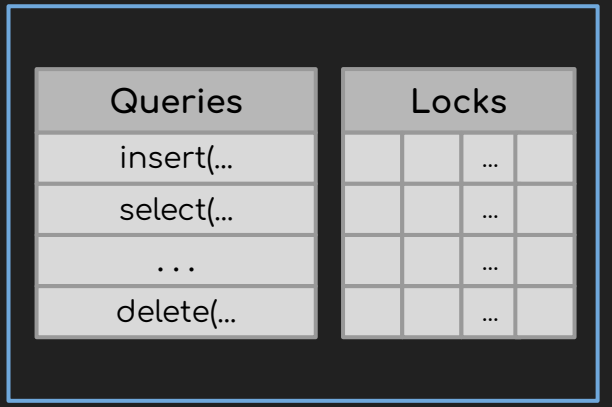
- Non-blocking and reentrant
- Shared and exclusive locks used by 2PL
- Allows many readers, but only 1 writer
- Managed by a lock-manager object, which is built on top of the Python dictionary

1 Writer OR Many readers

Transaction

- Standard 2PL using Lock class
- First tries to acquire all necessary locks
  - If one fails, release all acquired locks and abort
- Then performs all queries contention-free
- Finally releases all locks

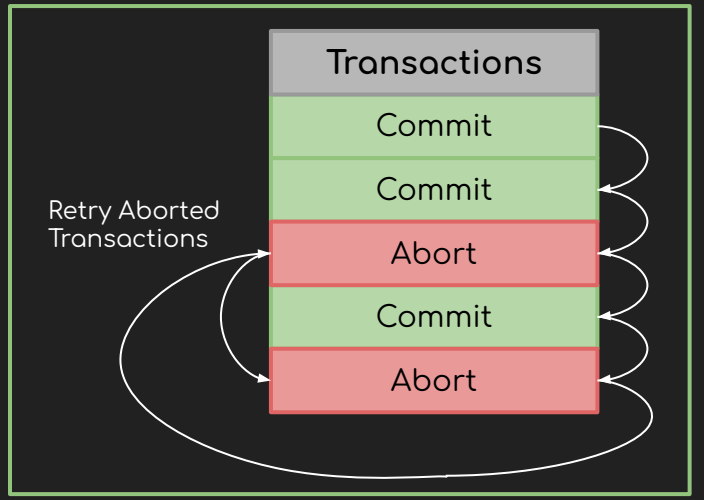
Transaction



Transaction Worker

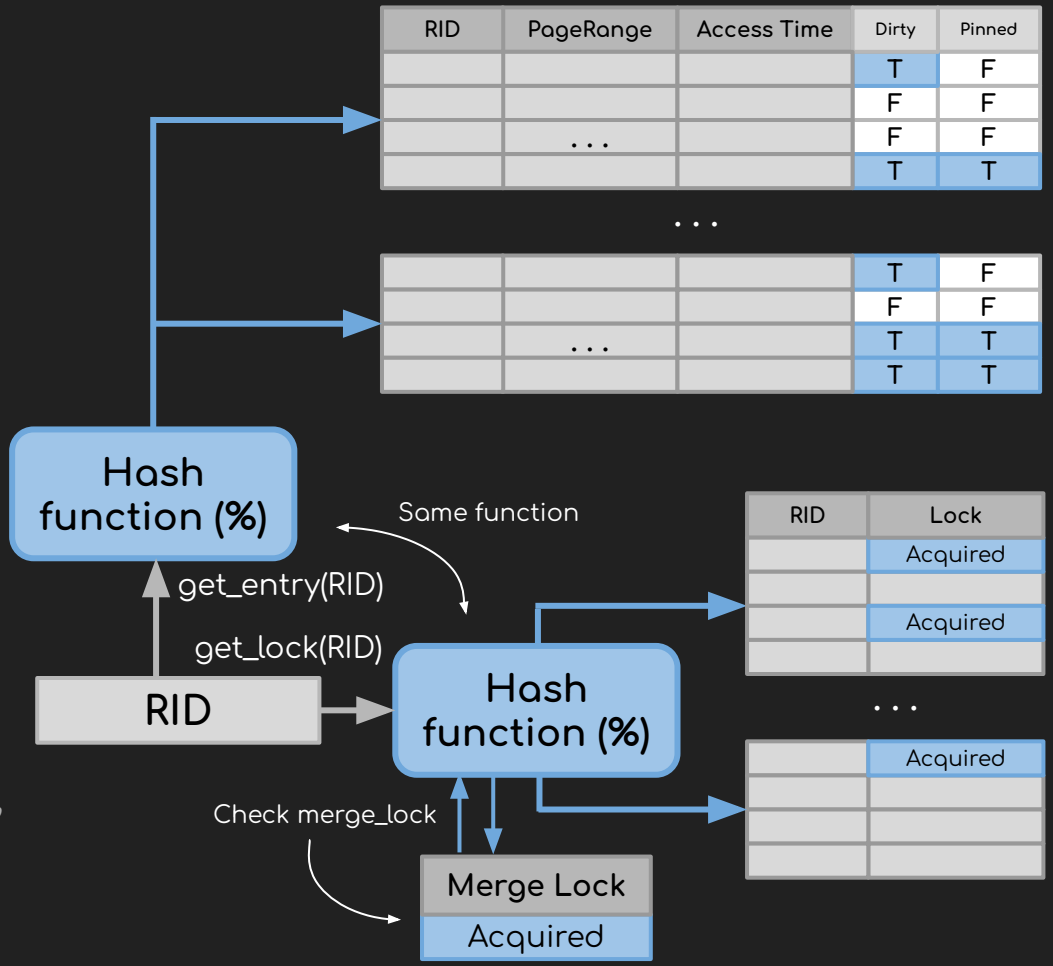
- Runs all given Transactions
- If one aborts, try it again later
- Finish when all Transactions have run successfully

Transaction Worker



Bufferpool (TLB)

- The bufferpool has been made to be contention-free
  - Corresponding table of locks
- If full of dirty PageRanges, bufferpool is merged with the disk
  - prevents more bufferpool locks from being acquired
  - When all locks are released, merge is conducted



Looking Ahead

- Faster language like C++ or Rust
- More robust logging to recover from crashes better
- Advanced concurrency control like 2VCC or QueCC

Thank You!