Milestone 2

Yingchen Gu, Glenn Chen
Rishika Roy, Kaleb Crans,
Ryan Kim

Overview

Durability and Bufferpool

Disk File Design

Read and writes to disk

Bufferpool management

Dirty Page

(Un)Pining Pages

Data Reorg

Merging

TPS

Indexing

Index across columns

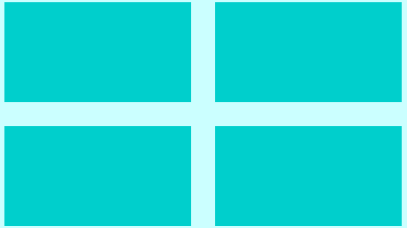
Durability and Bufferpool

Disk File Structure

Initial path

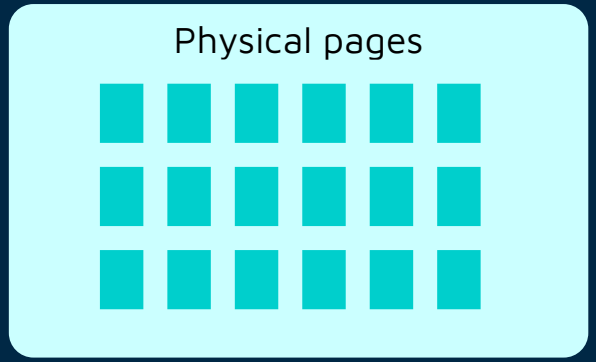
- passed to the database by the open() function
- contains the number of records and also the names of each table

Table metadata

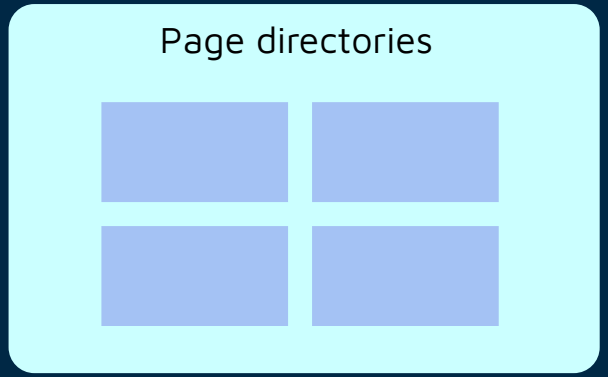


- one text file for each table
  - number of columns
  - key column
  - number of page ranges
  - number of base and tail pages for each page range
  - TPS for each base page

Disk File Structure

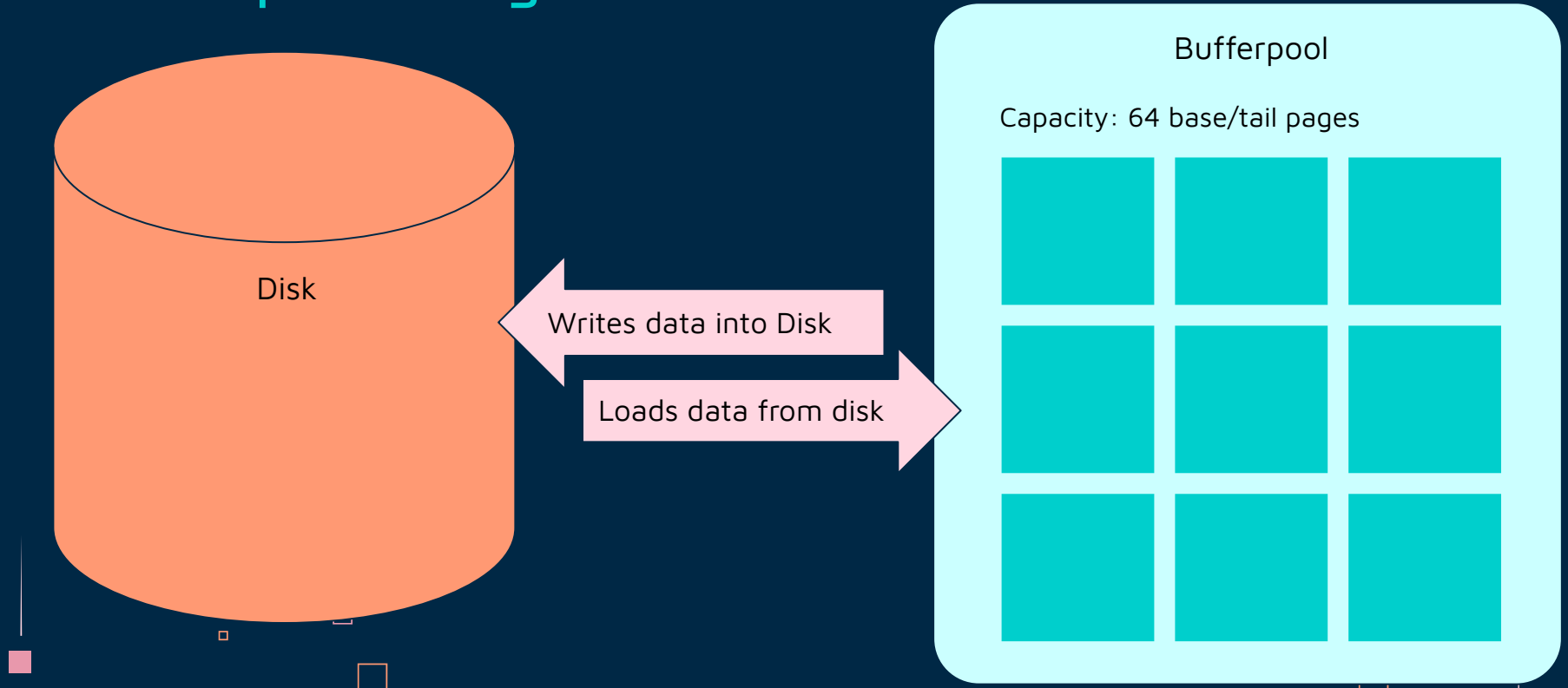


- one binary file for each physical page
- contains the byte array data
- named according to its table, page range ID, page type, base/tail page ID, and column number

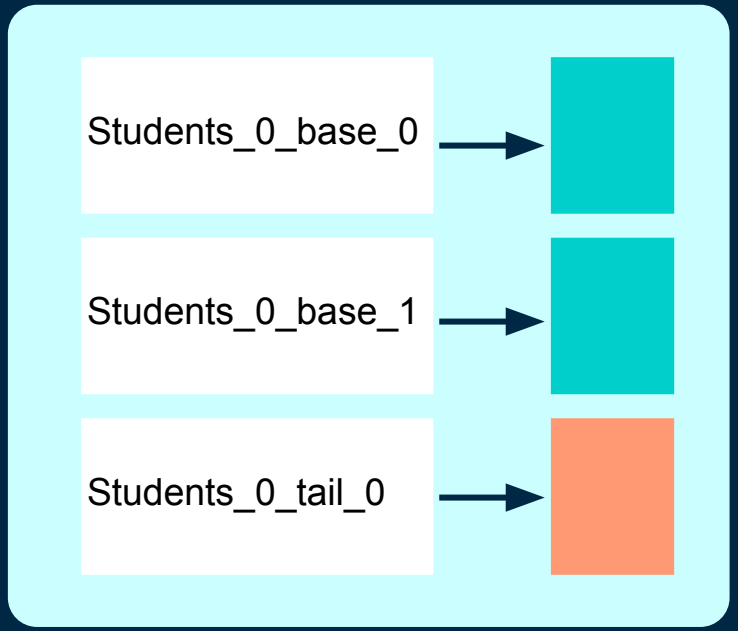


- one text file for each directory
- each line contains RID, page range ID, page type, base/tail page ID, and offset

Bufferpool Design



Bufferpool Design



Hash Table Structure

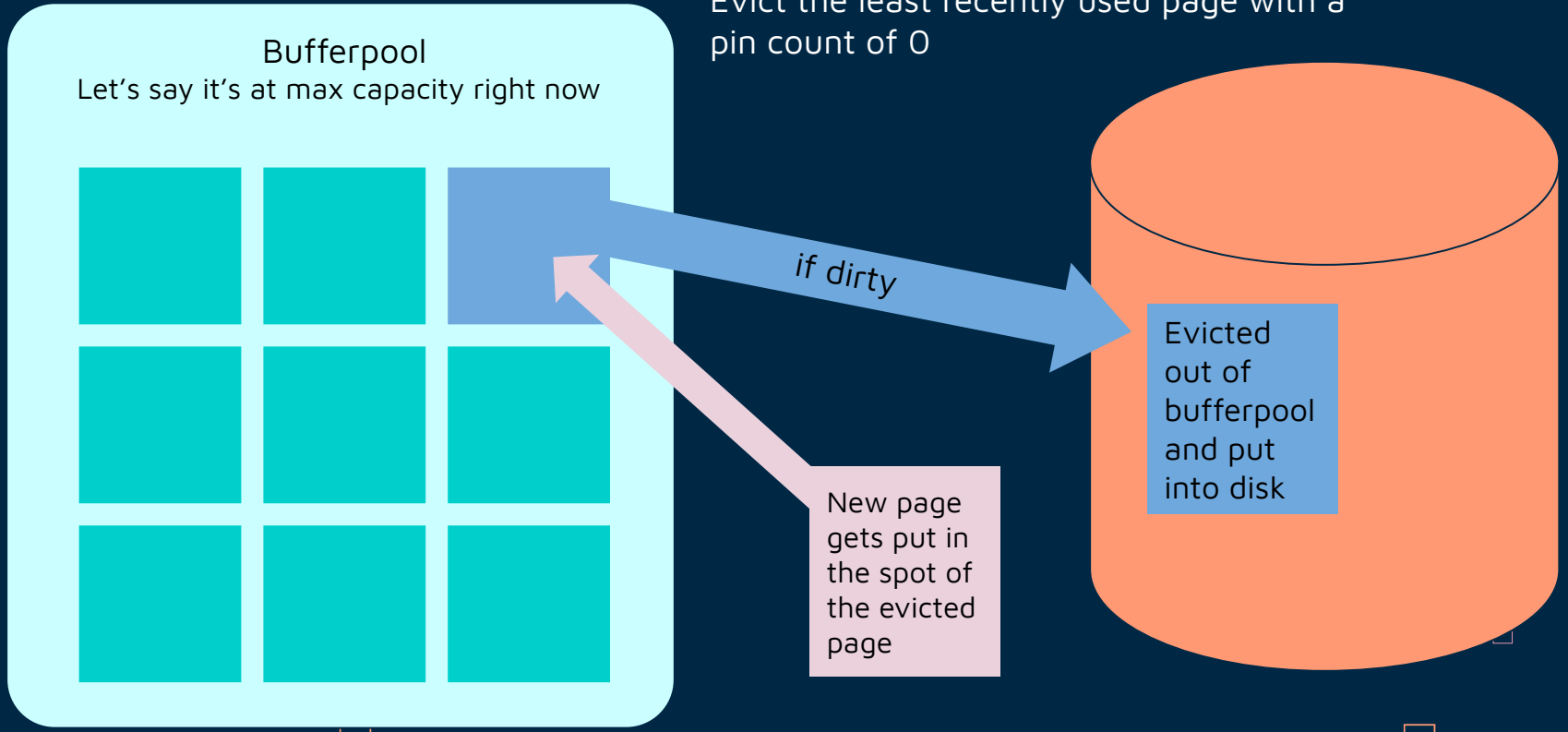
Page names used as keys to access base/tail pages

Contains functions to

- add a newly generated page to bufferpool
- retrieve a page from disk and put it in bufferpool

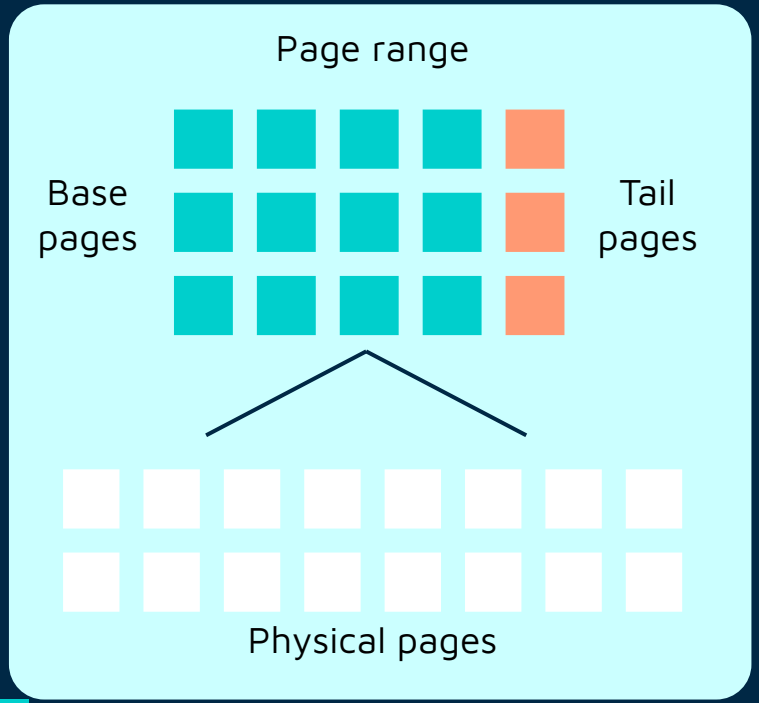
Bufferpool: Eviction

Evict the least recently used page with a pin count of 0

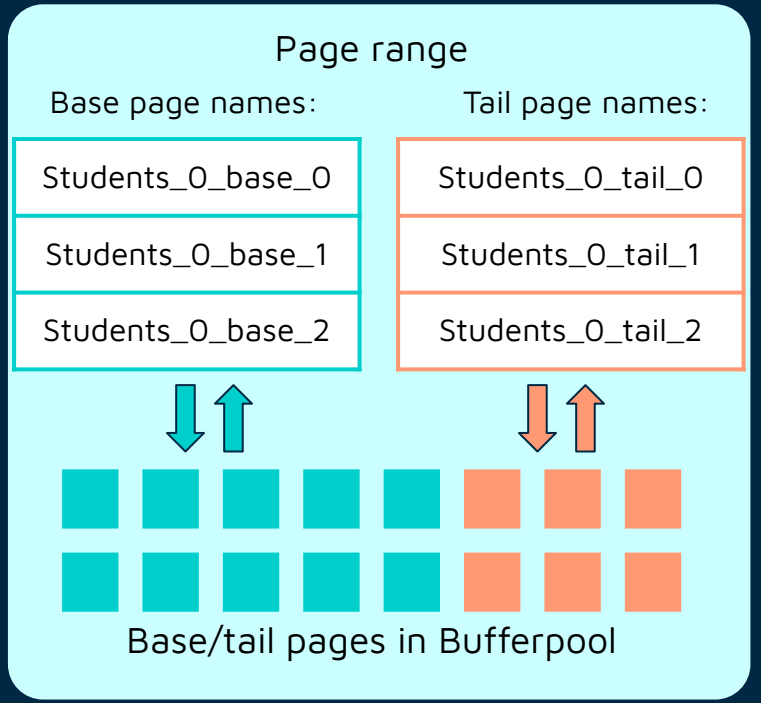


Changes to Page Range Structure

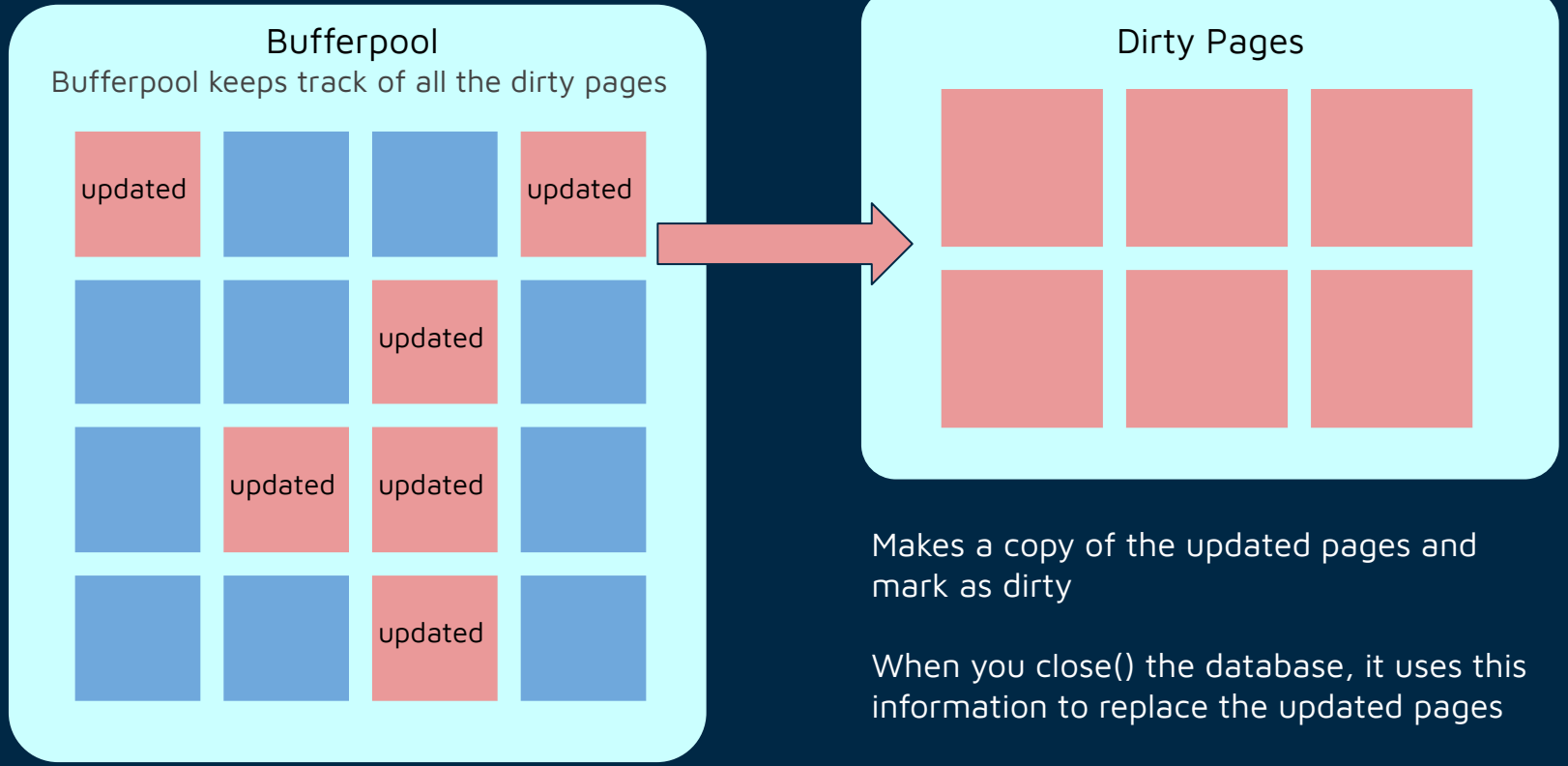
Before:



After:



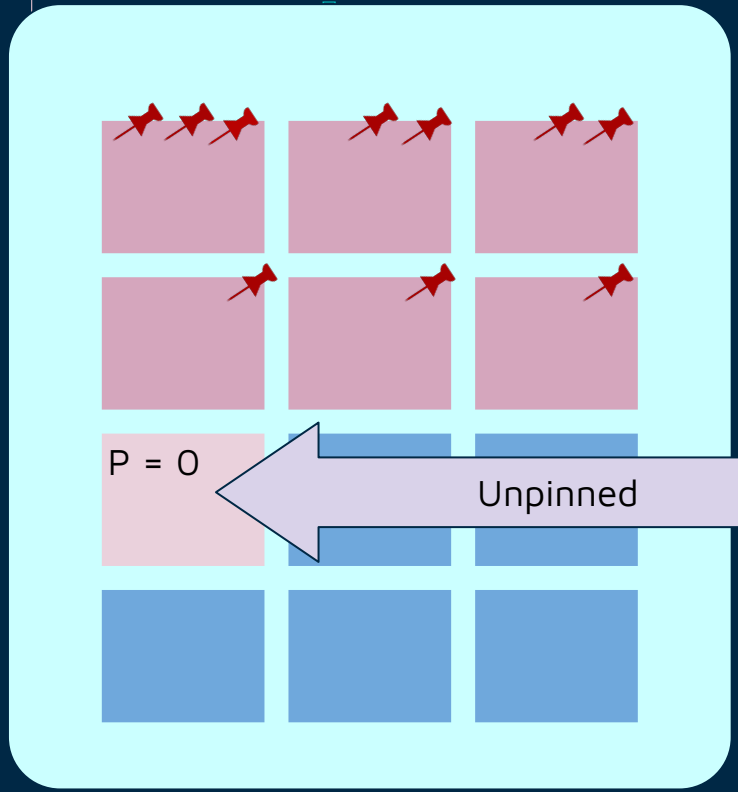
Dirty Pages



Makes a copy of the updated pages and mark as dirty

When you close() the database, it uses this information to replace the updated pages

(Un)Pinning Pages



Pinned Pages

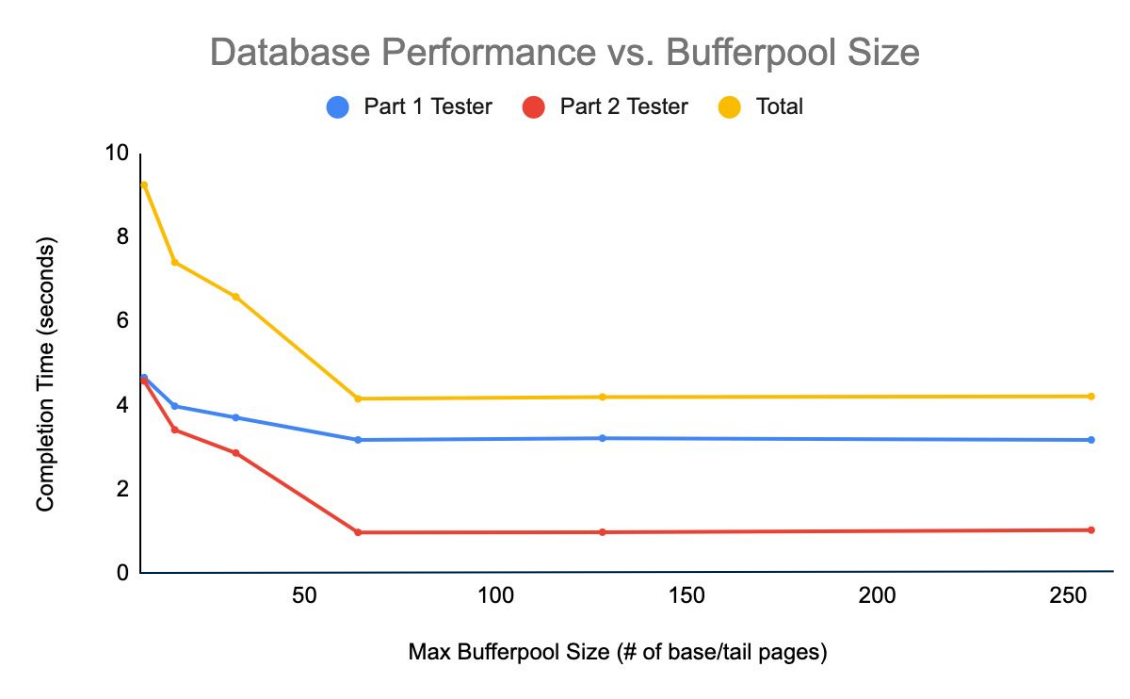
UnPinned Pages

# of transactions that need the page

When the page is not being accessed or needed anymore, it will be unpinned (pin = 0)

Only unpinned pages can be evicted from the bufferpool

Efficiency: Bufferpool Size Comparison



MacBook Pro, macOS Big Sur
Processor: GHz Dual-Core Intel Core i5
Memory: 8 GB 1867 MHz LPDDR3

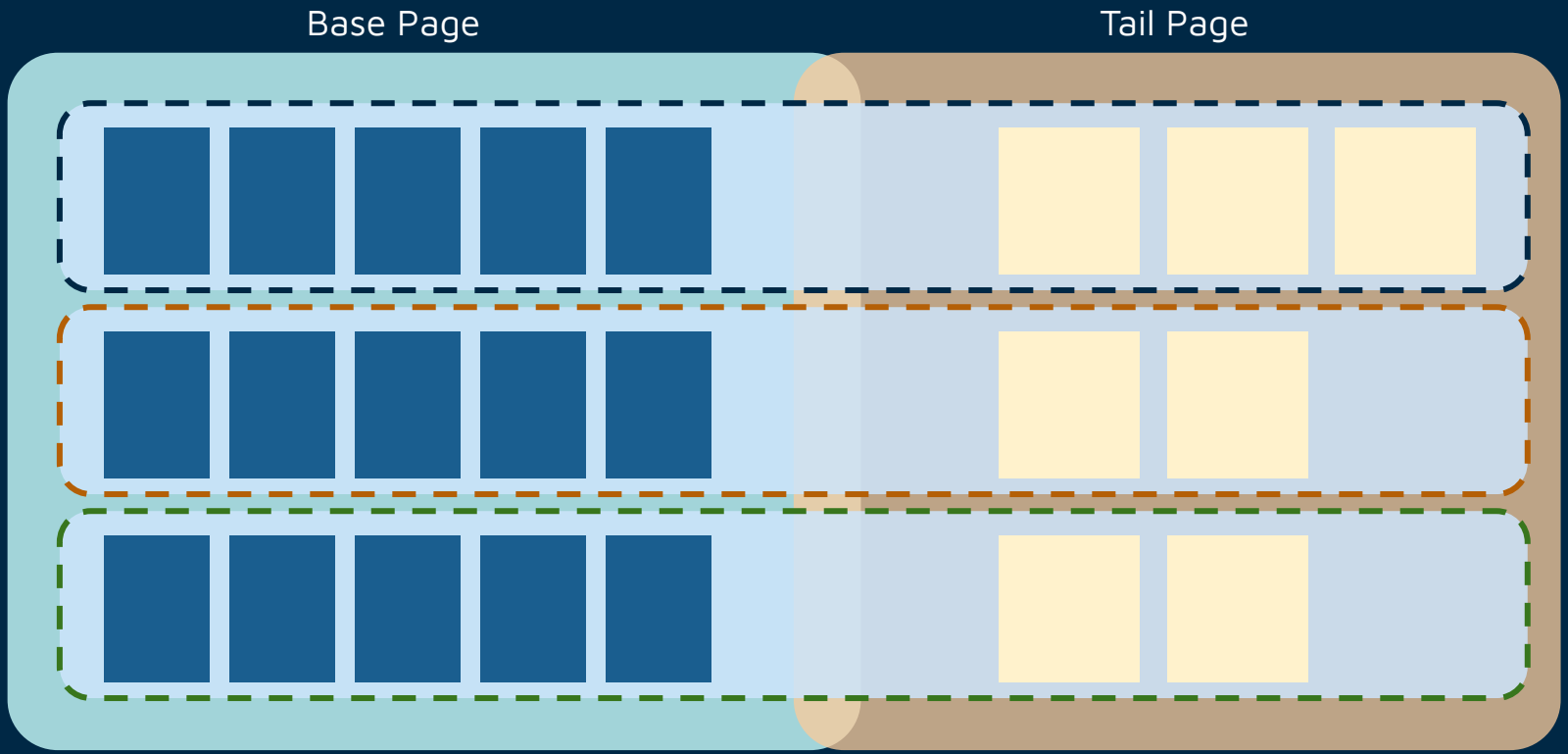
The provided milestone 2 testers were used, which opened and closed the database along with performing inserting, selecting, updating, and aggregating 1k records

We see that we reach optimal performance at a bufferpool size of around 64

Data Reorg

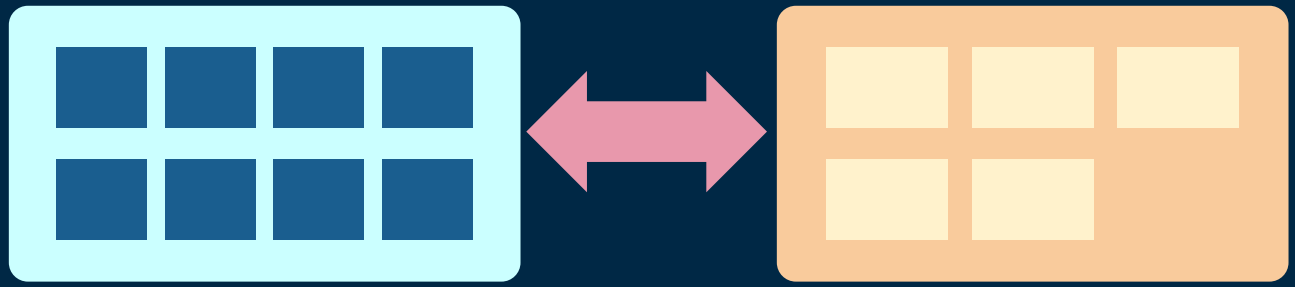
Merge

Merge combines the Base and Tail pages together



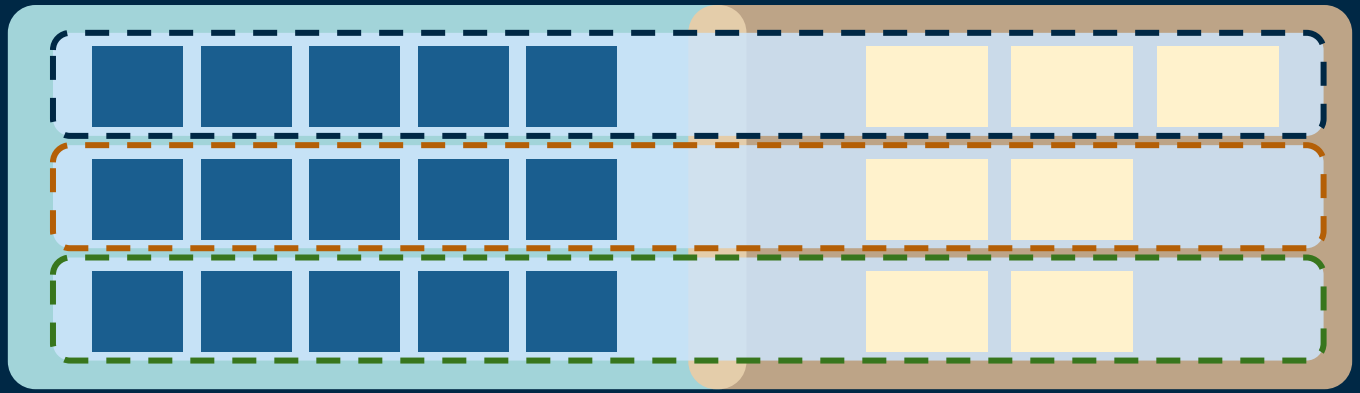
Merge in Progress...

Unmerged

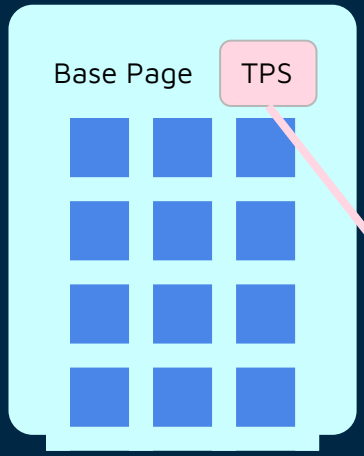


While merging, a copy of the merge page will be created
So it will have the unmerged base page and the merged based page

Merged

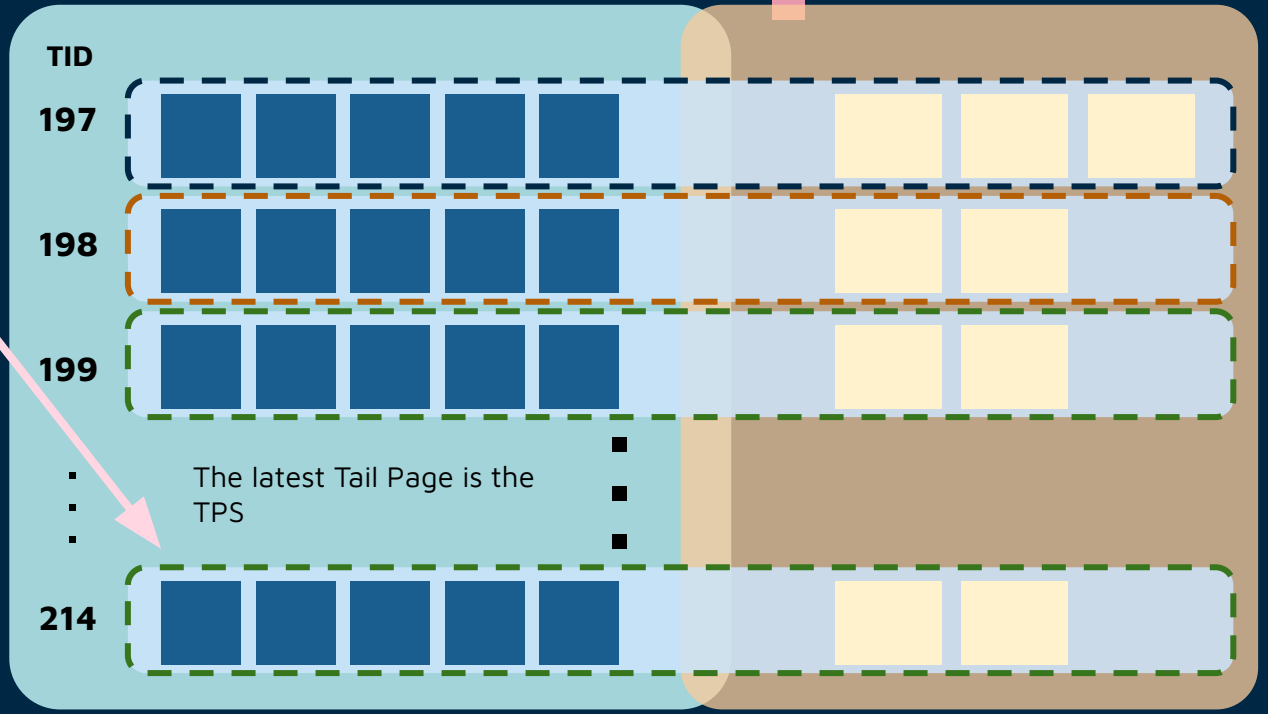


Tail-Page Sequence Number (TPS)



There's a TPS or lineage in every Base Page

The latest TPS has the smallest TID



Indexing

Indexing

Similar to the way we indexed through the data from Milestone 1 using dictionaries

When a database is opened, it will look through the data and give it an index

Game Company Dictionary

Data Value	RIDs
Riot	1,3
Hoyoverse	2
Respawn	0

Game Dictionary

Data Value	RIDs
Apex	0
League	1
Genshin	2
Valorant	3

Table

RID	Game	Game Company
0	Apex	Respawn
1	League	Riot
2	Genshin	Hoyoverse
3	Valorant	Riot

Thank You!