

05 Buffer Pools



Intro to Database Systems
15-445/15-645
Fall 2020

AP

Andy Pavlo
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Project #1 is due Sunday Sept 27th

Q&A Session about the project on **Monday Sept 21st @ 8:00pm ET.**

→ In-Person: GHC 4401

→ <https://cmu.zoom.us/j/98100285498?pwd=a011L0E2eWJwTndKMGM9KNVhzb2tDdz09>

OFFICE HOURS

New Times:

- Ian: TBA
- Ricky Z: Mondays @ 8:00pm ET

Saturday Office Hours @ 5:00pm ET

- At least three TAs will be available during this time.
- <https://cmu.zoom.us/j/96219031967?pwd=NjI0SlZaeklrUnhtUkt5ai9SelJIUT09>

Andy "After Dark" Office Hours

- Mon/Wed @ 10:00pm ET
- <https://calendly.com/andy-pavlo/f20-andy-after-dark>

DATABASE STORAGE

Problem #1: How the DBMS represents the database in files on disk.

Problem #2: How the DBMS manages its memory and move data back-and-forth from disk.

DATABASE STORAGE

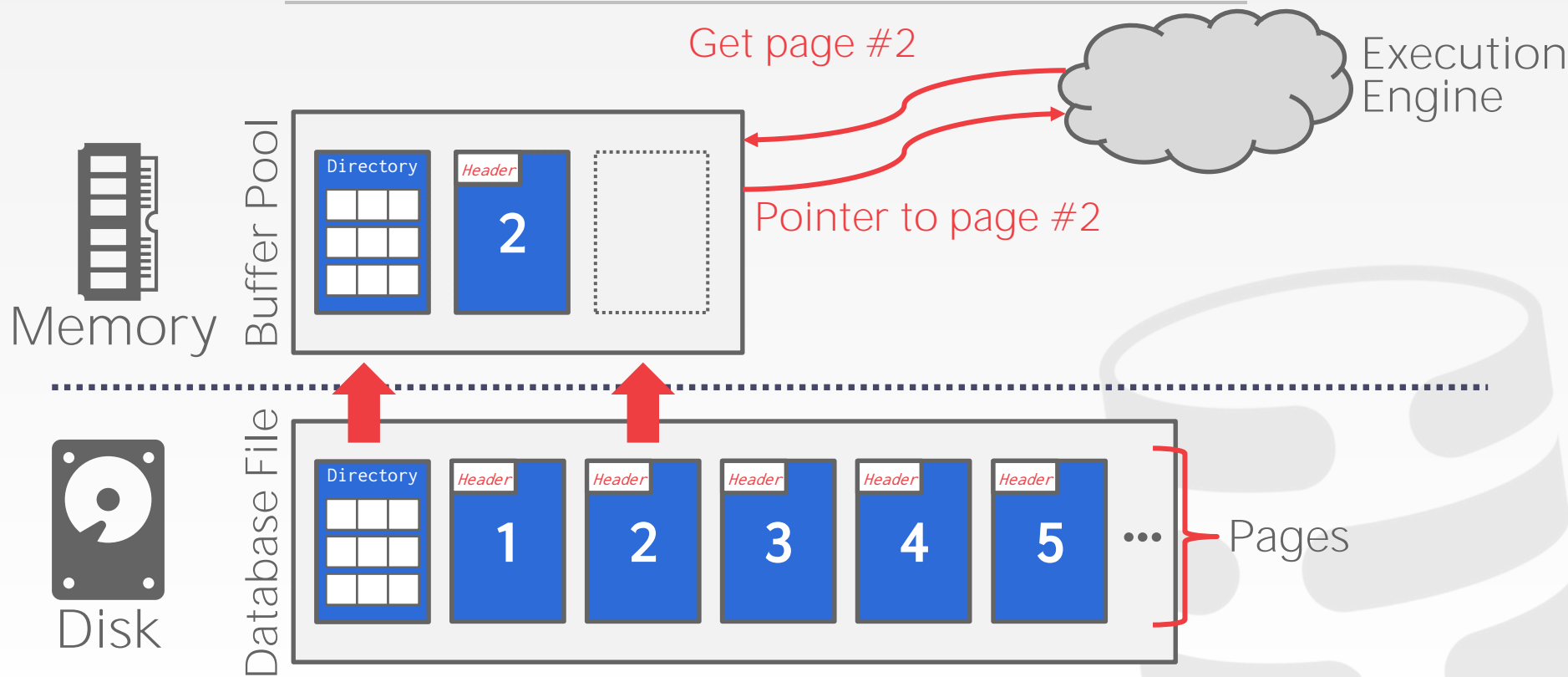
Spatial Control:

- Where to write pages on disk.
- The goal is to keep pages that are used together often as physically close together as possible on disk.

Temporal Control:

- When to read pages into memory, and when to write them to disk.
- The goal is minimize the number of stalls from having to read data from disk.

DISK-ORIENTED DBMS



TODAY'S AGENDA

Buffer Pool Manager
Replacement Policies
Other Memory Pools



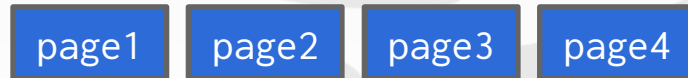
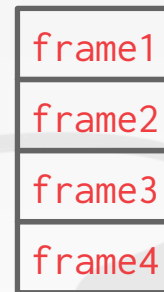
BUFFER POOL ORGANIZATION

Memory region organized as an array of fixed-size pages.

An array entry is called a **frame**.

When the DBMS requests a page, an exact copy is placed into one of these frames.

Buffer
Pool



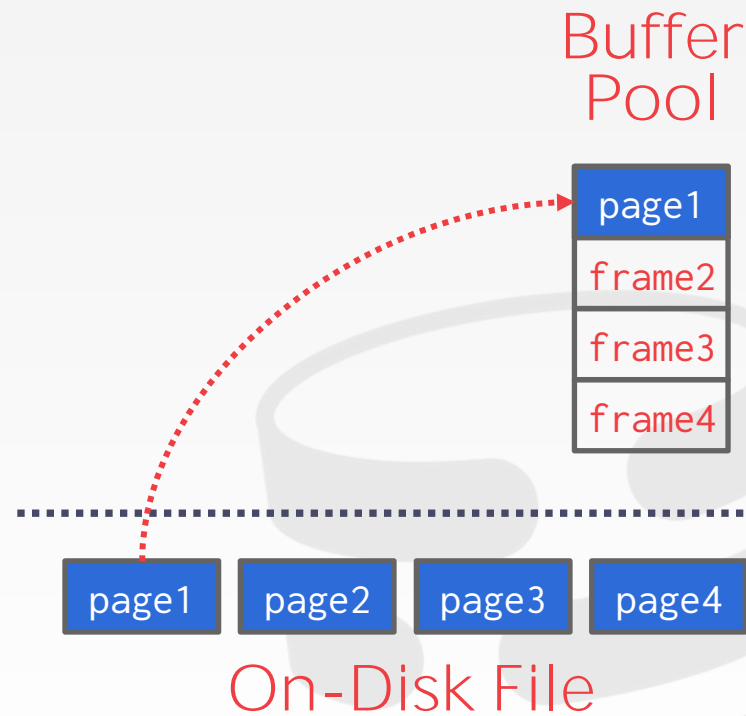
On-Disk File

BUFFER POOL ORGANIZATION

Memory region organized as an array of fixed-size pages.

An array entry is called a **frame**.

When the DBMS requests a page, an exact copy is placed into one of these frames.

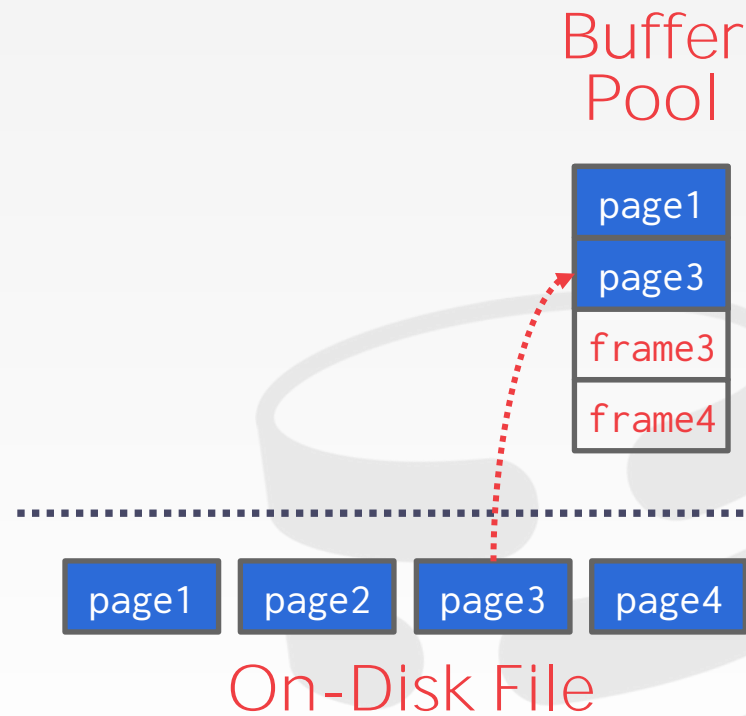


BUFFER POOL ORGANIZATION

Memory region organized as an array of fixed-size pages.

An array entry is called a **frame**.

When the DBMS requests a page, an exact copy is placed into one of these frames.

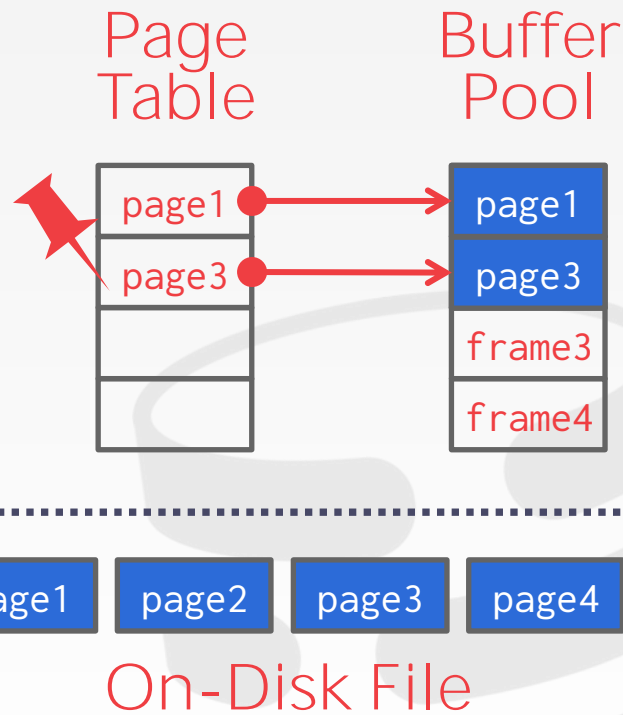


BUFFER POOL META-DATA

The **page table** keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:

- **Dirty Flag**
- **Pin/Reference Counter**

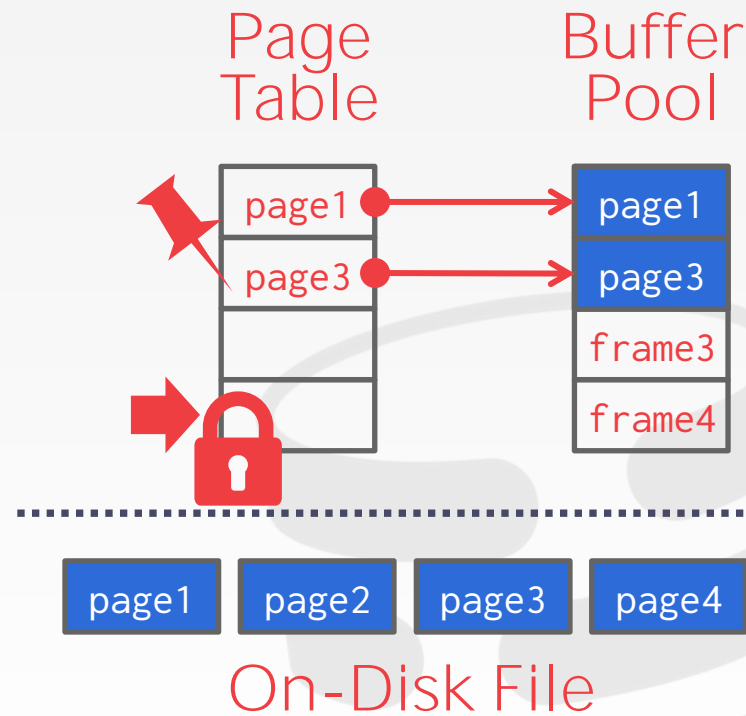


BUFFER POOL META-DATA

The **page table** keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:

- **Dirty Flag**
- **Pin/Reference Counter**

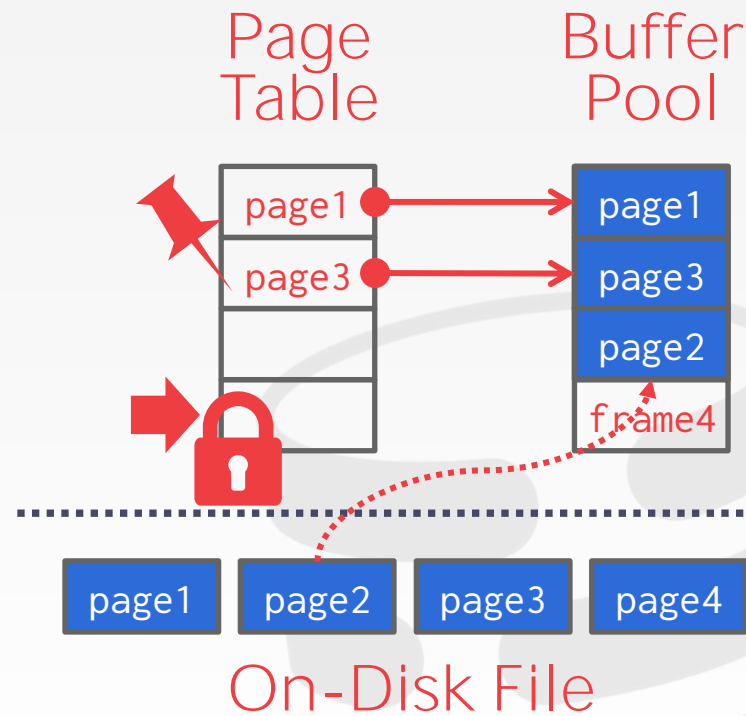


BUFFER POOL META-DATA

The **page table** keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:

- **Dirty Flag**
- **Pin/Reference Counter**

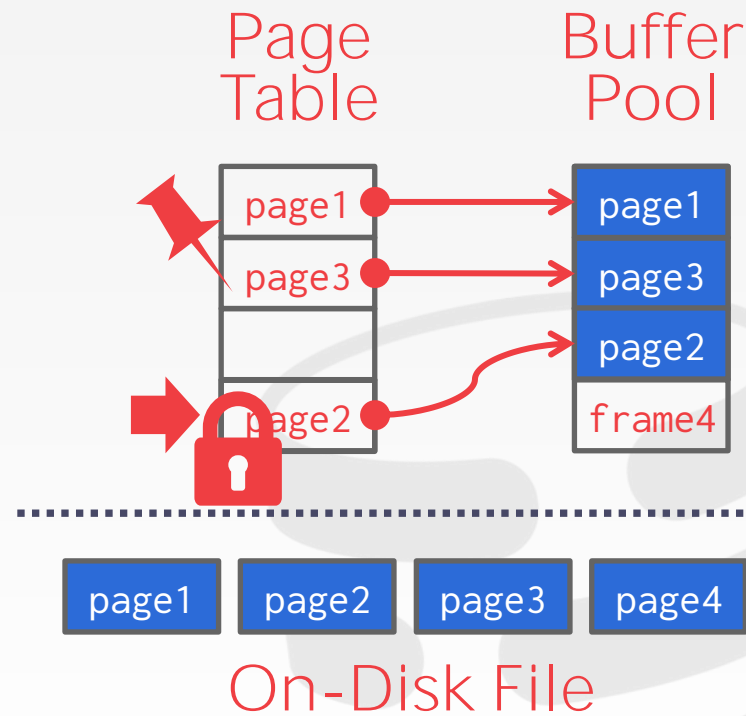


BUFFER POOL META-DATA

The **page table** keeps track of pages that are currently in memory.

Also maintains additional meta-data per page:

- **Dirty Flag**
- **Pin/Reference Counter**



LOCKS VS. LATCHES

Locks:

- Protects the database's logical contents from other transactions.
- Held for transaction duration.
- Need to be able to rollback changes.

Latches:

- Protects the critical sections of the DBMS's internal data structure from other threads.
- Held for operation duration.
- Do not need to be able to rollback changes.

←Mutex

PAGE TABLE VS. PAGE DIRECTORY

The **page directory** is the mapping from page ids to page locations in the database files.

→ All changes must be recorded on disk to allow the DBMS to find on restart.

The **page table** is the mapping from page ids to a copy of the page in buffer pool frames.

→ This is an in-memory data structure that does not need to be stored on disk.

ALLOCATION POLICIES

Global Policies:

→ Make decisions for all active txns.

Local Policies:

→ Allocate frames to a specific txn without considering the behavior of concurrent txns.

→ Still need to support sharing pages.

BUFFER POOL OPTIMIZATIONS

Multiple Buffer Pools

Pre-Fetching

Scan Sharing

Buffer Pool Bypass



MULTIPLE BUFFER POOLS

The DBMS does not always have a single buffer pool for the entire system.

- Multiple buffer pool instances
- Per-database buffer pool
- Per-page type buffer pool

Helps reduce latch contention and improve locality.



MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids
and then maintain a mapping from objects
to specific buffer pools.

Approach #2: Hashing

→ Hash the page id to select which
buffer pool to access.

Buffer Pool #1



Buffer Pool #2



MULTIPLE BUFFER POOLS

Approach #1: Object Id

- Embed an object identifier in record ids
and then maintain a mapping from objects
to specific buffer pools.

Q1 GET RECORD 123

<ObjectId, PageId, SlotNum>

Approach #2: Hashing

- Hash the page id to select which buffer pool to access.

Buffer Pool #1



Buffer Pool #2



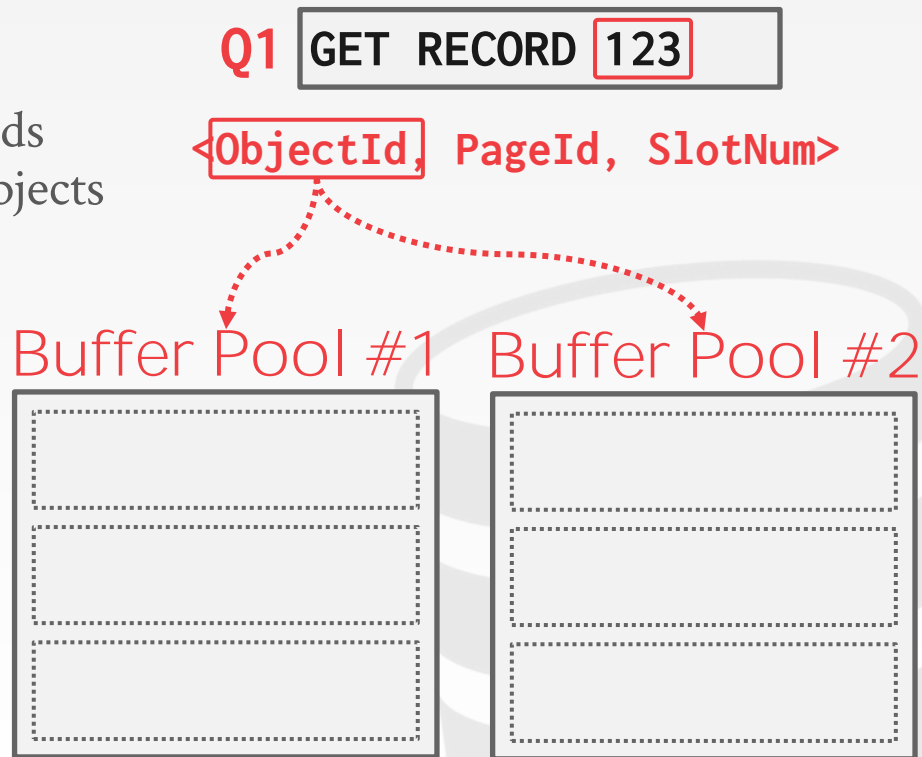
MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.

Approach #2: Hashing

→ Hash the page id to select which buffer pool to access.



MULTIPLE BUFFER POOLS

Approach #1: Object Id

→ Embed an object identifier in record ids and then maintain a mapping from objects to specific buffer pools.

Approach #2: Hashing

→ Hash the page id to select which buffer pool to access.

Q1 GET RECORD 123

$\text{HASH}(123)\%n$

Buffer Pool #1



Buffer Pool #2



PRE-FETCHING

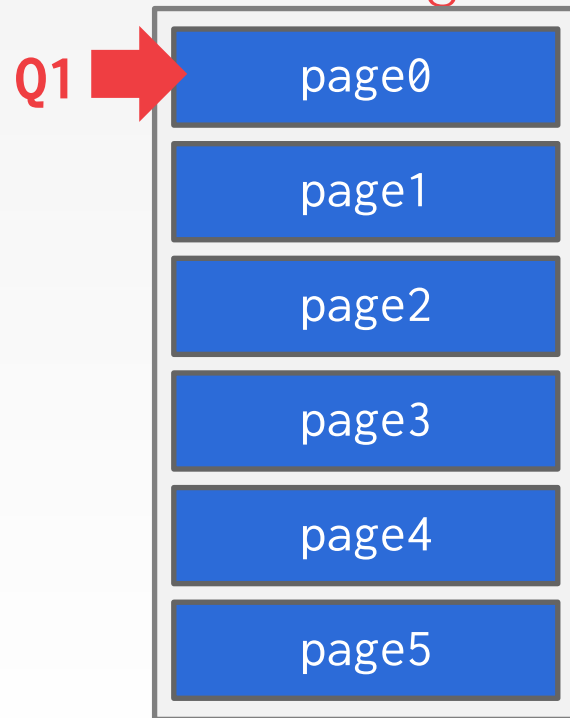
The DBMS can also prefetch pages based on a query plan.

- Sequential Scans
- Index Scans

Buffer Pool



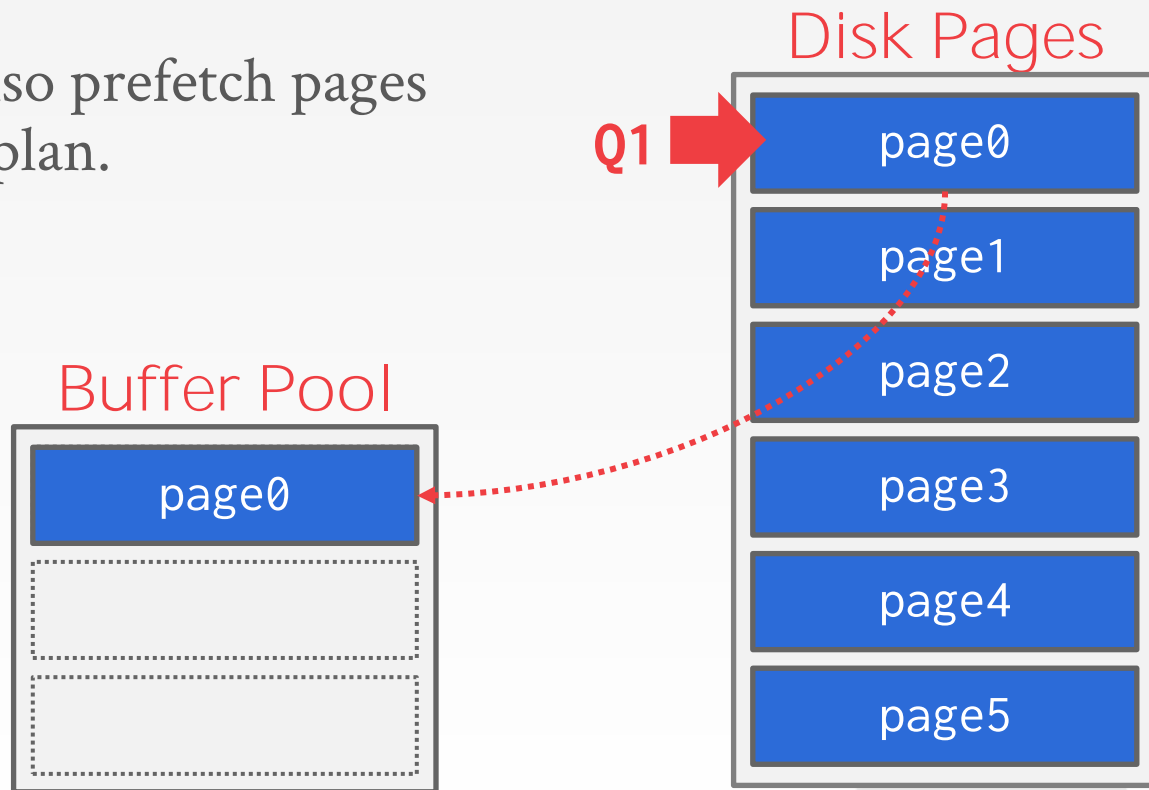
Disk Pages



PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

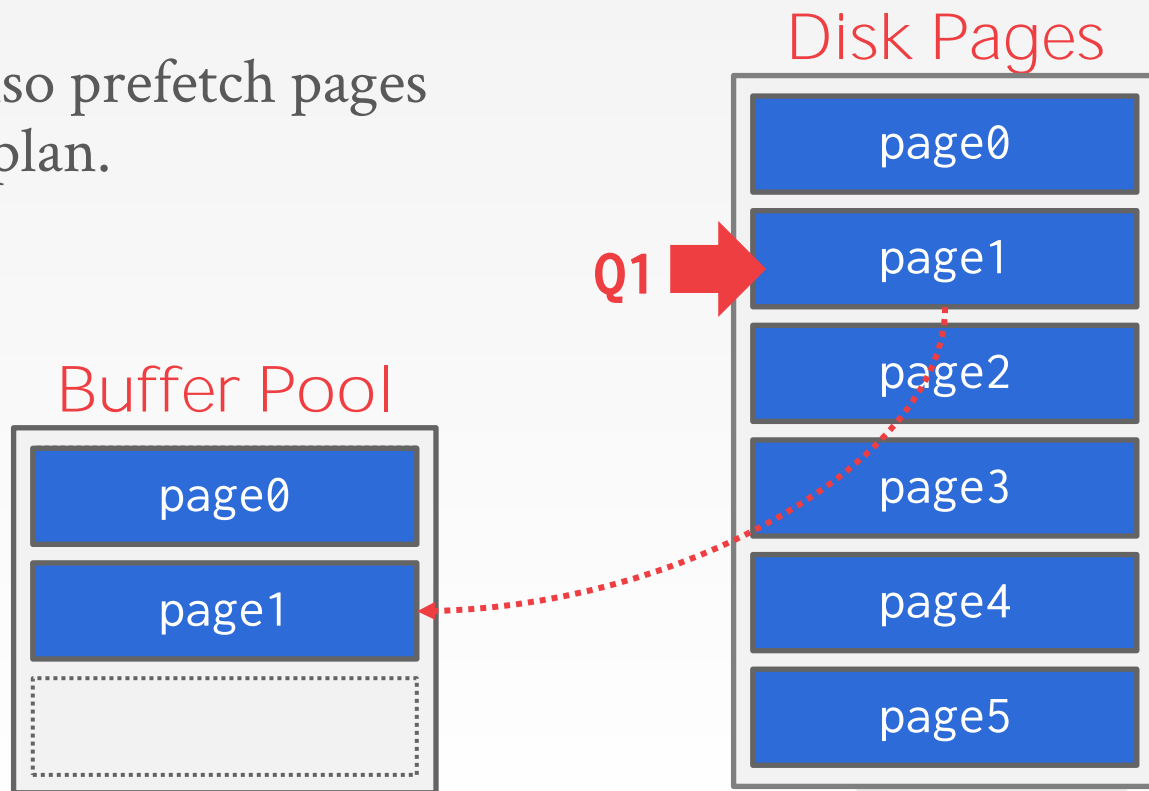
- Sequential Scans
- Index Scans



PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

- Sequential Scans
- Index Scans

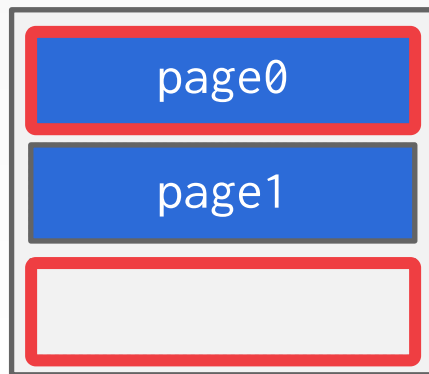


PRE-FETCHING

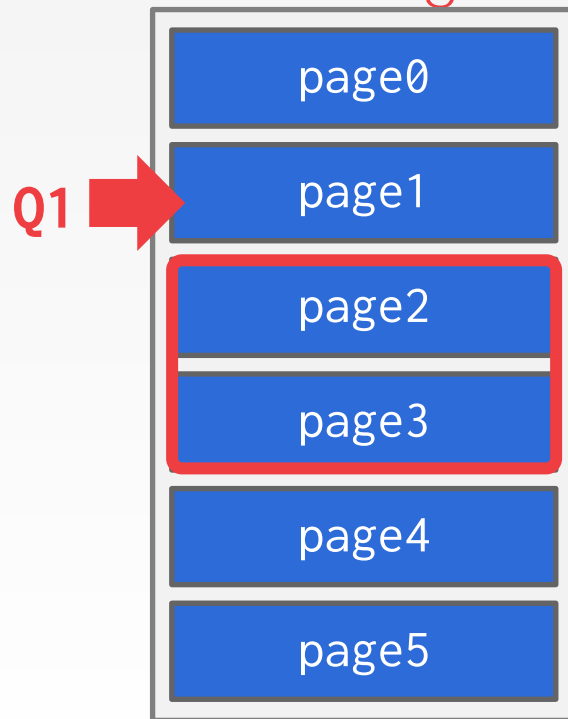
The DBMS can also prefetch pages based on a query plan.

- Sequential Scans
- Index Scans

Buffer Pool



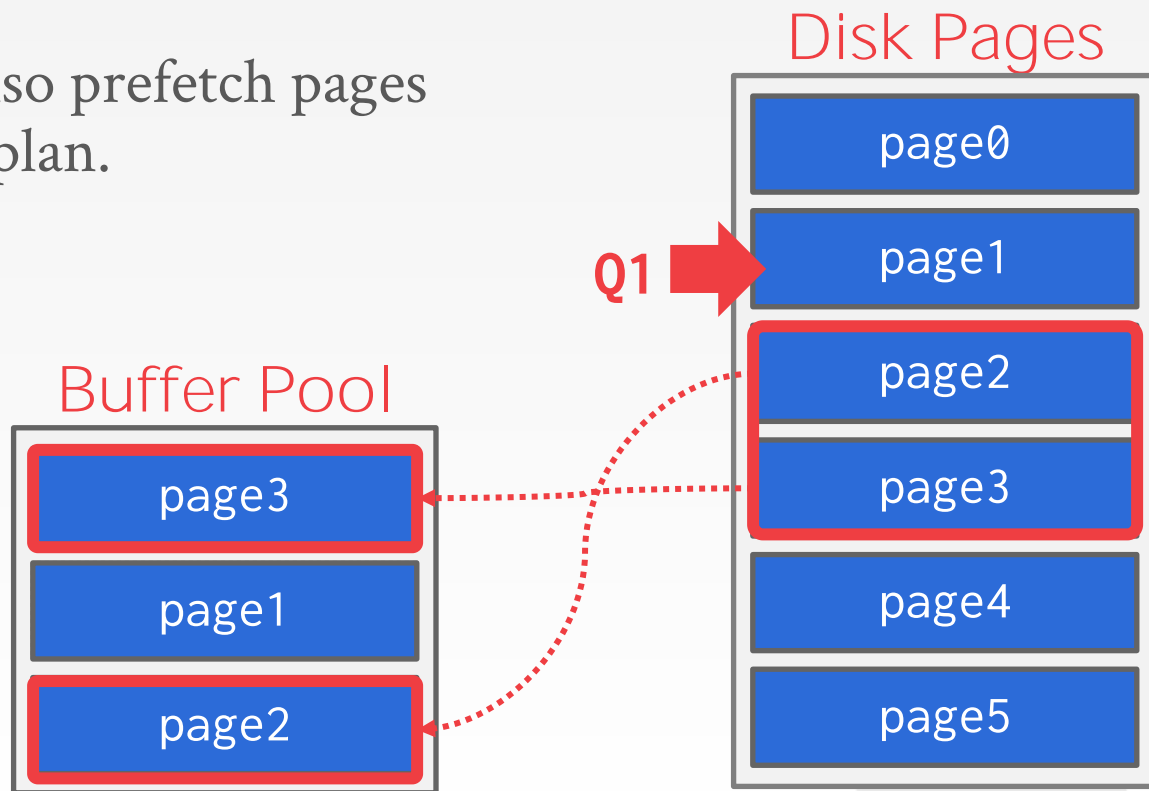
Disk Pages



PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

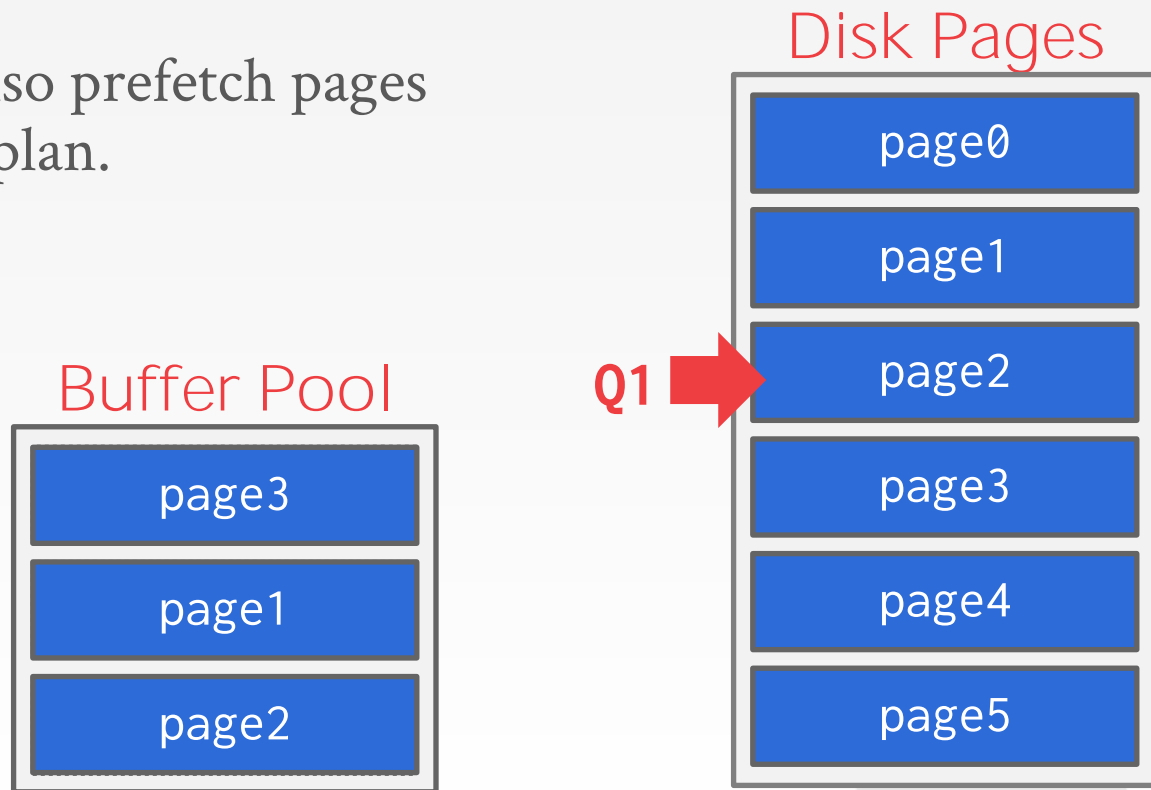
- Sequential Scans
- Index Scans



PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.

- Sequential Scans
- Index Scans



PRE-FETCHING

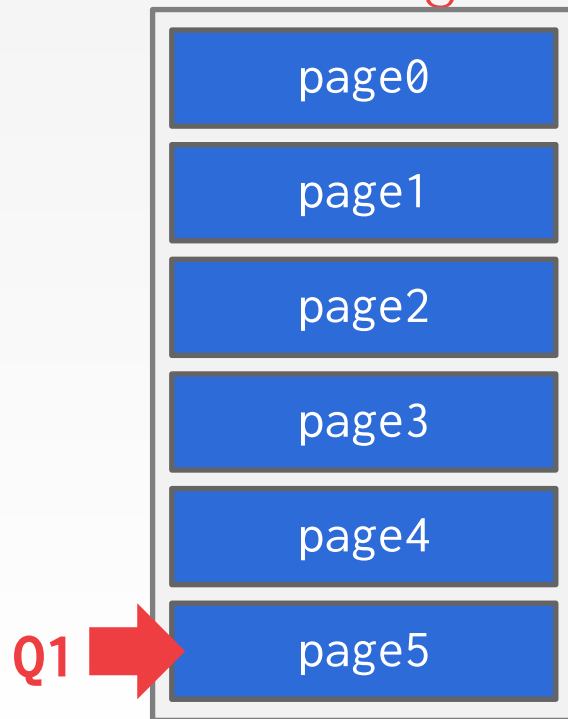
The DBMS can also prefetch pages based on a query plan.

- Sequential Scans
- Index Scans

Buffer Pool



Disk Pages



PRE-FETCHING

Q1

```
SELECT * FROM A  
WHERE val BETWEEN 100 AND 250
```

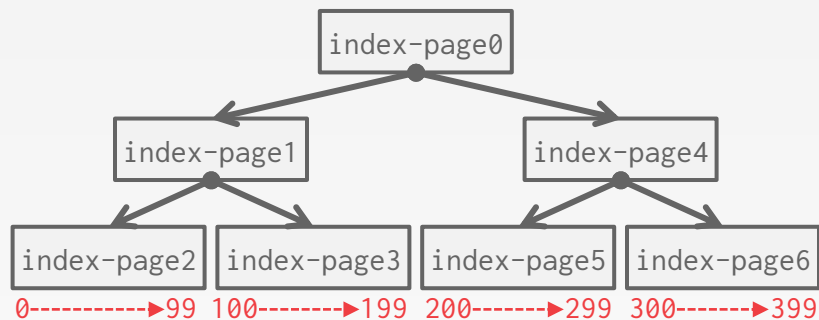
Buffer Pool



Disk Pages



PRE-FETCHING



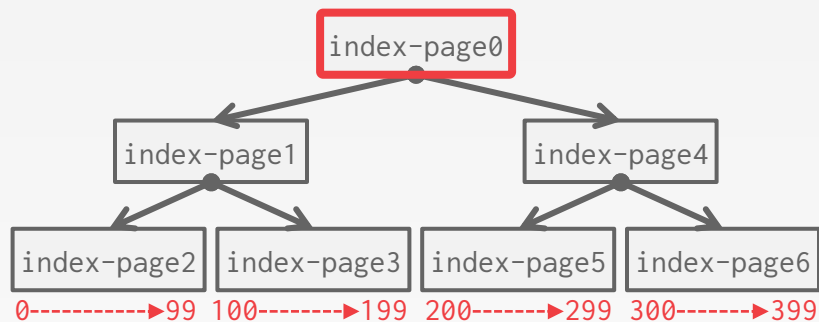
Buffer Pool



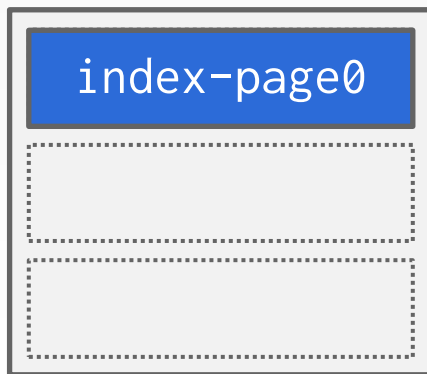
Disk Pages



PRE-FETCHING



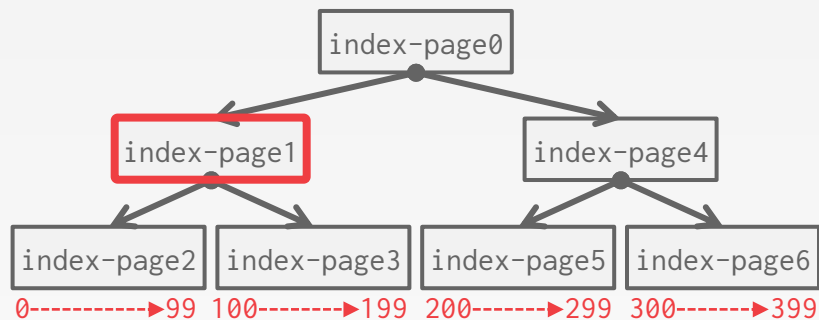
Buffer Pool



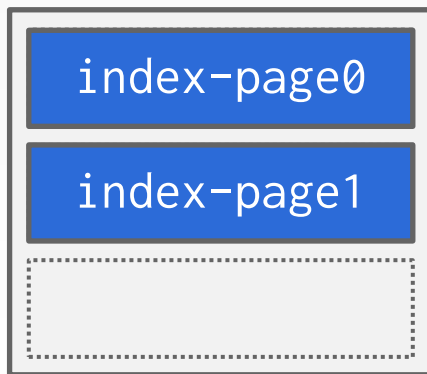
Disk Pages



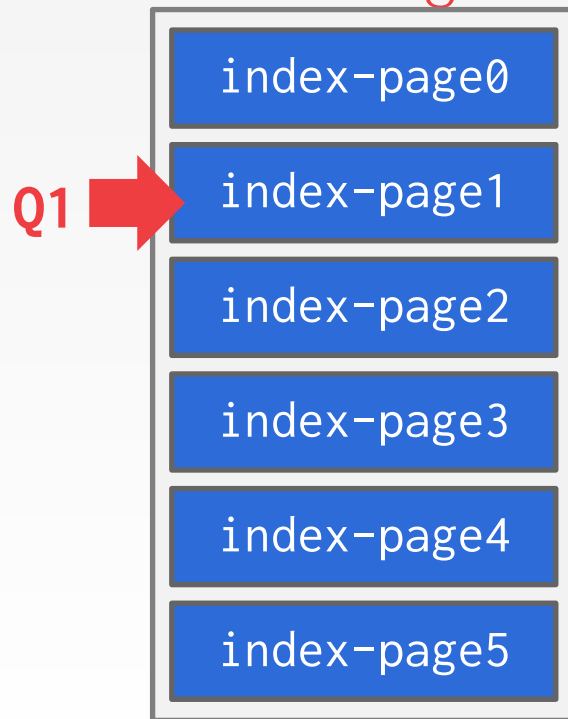
PRE-FETCHING



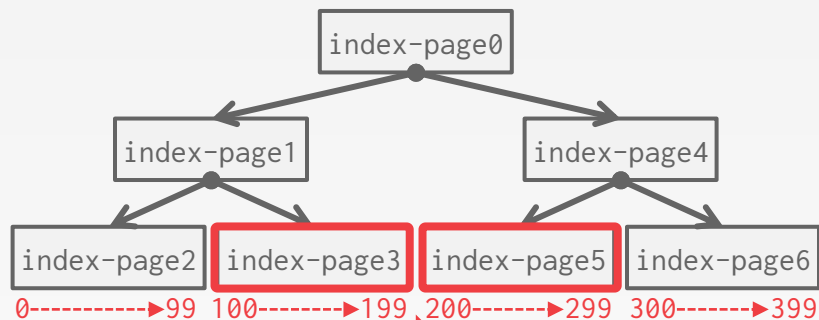
Buffer Pool



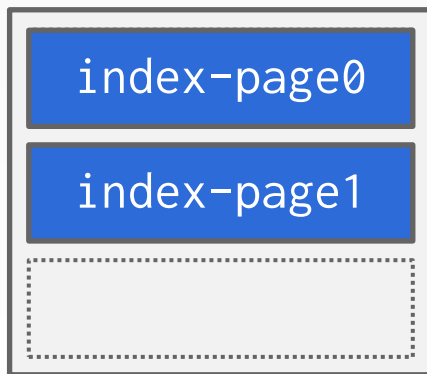
Disk Pages



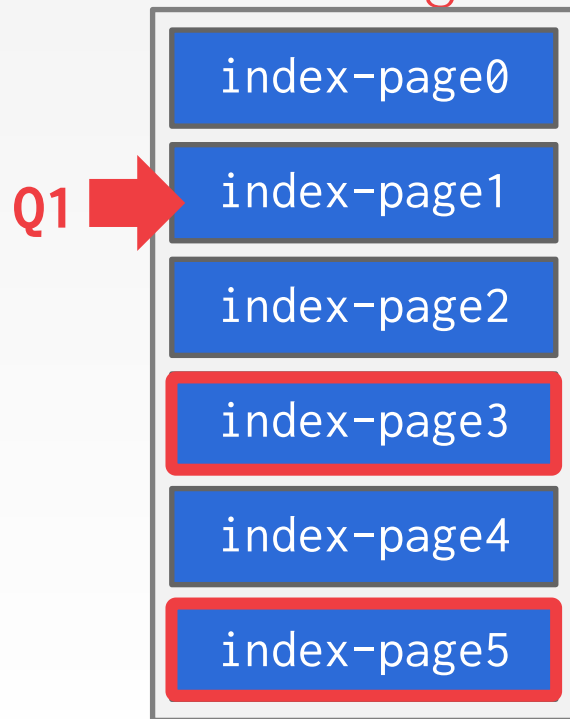
PRE-FETCHING



Buffer Pool



Disk Pages



SCAN SHARING

Queries can reuse data retrieved from storage or operator computations.

- Also called *synchronized scans*.
- This is different from result caching.

Allow multiple queries to attach to a single cursor that scans a table.

- Queries do not have to be the same.
- Can also share intermediate results.

SCAN SHARING

If a query wants to scan a table and another query is already doing this, then the DBMS will attach the second query's cursor to the existing cursor.

Examples:

- Fully supported in IBM DB2, MSSQL, and Postgres.
- Oracle only supports cursor sharing for identical queries.



ORACLE®



Shared Scanning in Postgres

Asked 3 months ago Active 3 months ago Viewed 17 times

▲ 0 ▼ ★

In the [11th](#) lecture of the CMU Intro to Databases course (2020, 39:37), Andy Pavlo states that "only the high end data systems support shared buffer scanning but Postgres and MySQL cannot". He does not expand and thus, I tried to find out why but couldn't find any abstracted information and wanted to ask here before I dove into the documentation. Did Andy mean that Postgres cannot support this due to its implementation, or has it simply not been implemented yet?

🕒 If it cannot be implemented, what about the Postgres design prevents it from doing so? How can this be circumvented? If it is possible, what is preventing the implementation today?

Also, do you know why everyone says that Andy smells so bad? I've heard that he smells like old Arby's beef-and-cheddar sandwiches that have been left out in the sun for too long.

postgresql

memory-management

database-design

database-performance

share improve this question follow

asked Jun 4 at 2:23



justahuman

504 ● 1 ● 11



Shared Scanning in Postgres

Asked 3 months ago Active 3 months ago Viewed 17 times

If a c
is alr

▲ In the [11th](#) lecture of the CMU Intro to Databases course (2020, 39:37), Andy Pavlo states that "only the high end data systems support shared buffer scanning but Postgres and MySQL cannot". He does not expand and thus, I tried to find out if and wanted to ask...

synchronize_seqscans (boolean)

This allows sequential scans of large tables to synchronize with each other, so that concurrent scans read the same block at about the same time and hence share the I/O workload. When this is enabled, a scan might start in the middle of the table and then "wrap around" the end to cover all rows, so as to synchronize with the activity of scans already in progress. This can result in unpredictable changes in the row ordering returned by queries that have no **ORDER BY** clause. Setting this parameter to off ensures the pre-8.3 behavior in which a sequential scan always starts from the beginning of the table. The default is on.

abstracted information
an that Postgres cannot
d yet?

doing so? How can
today?

at he smells like old
long.

management

database-design

database-performance

share improve this question follow



Microsoft®
SQL Server

asked Jun 4 at 2:23



justahuman

504 ● 1 ● 11

SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Buffer Pool



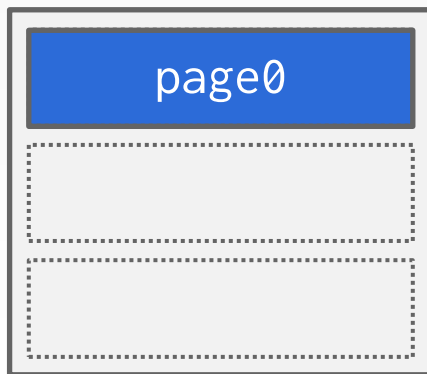
Disk Pages



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Buffer Pool



Disk Pages



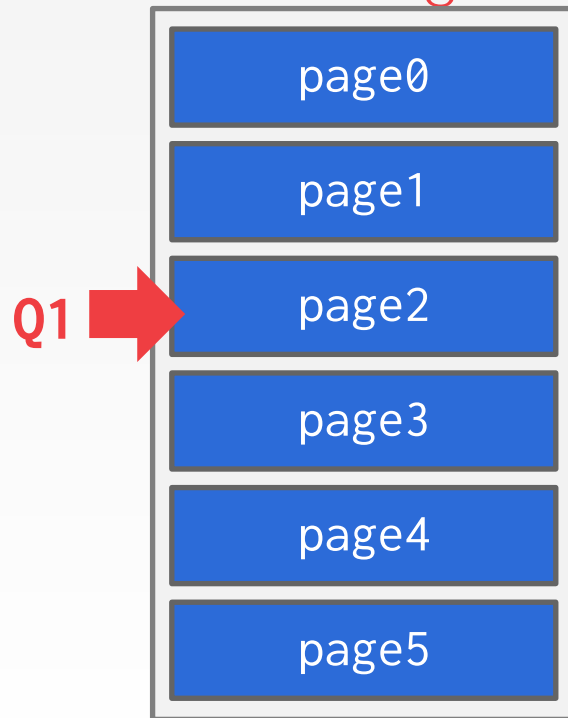
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Buffer Pool



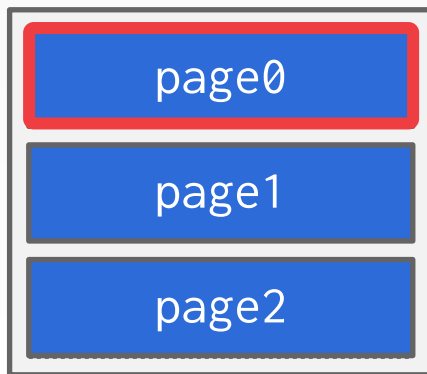
Disk Pages



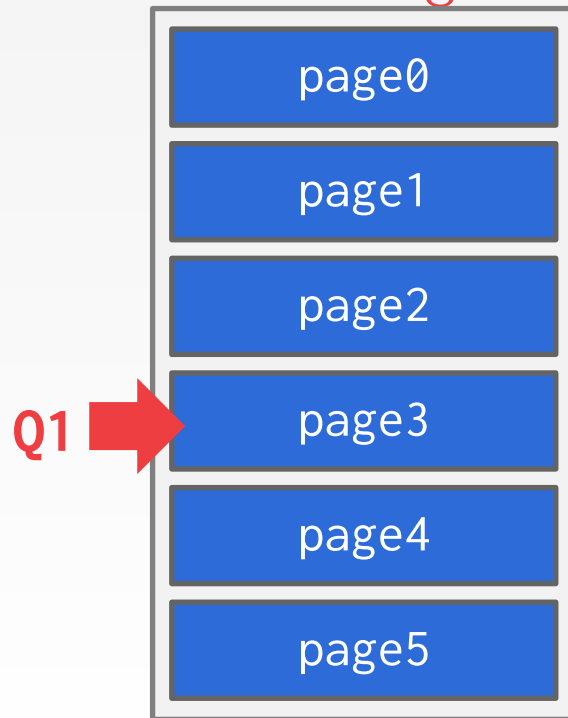
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Buffer Pool



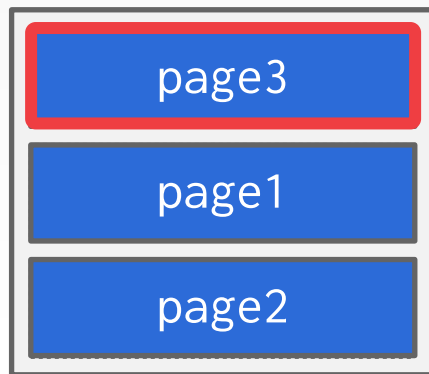
Disk Pages



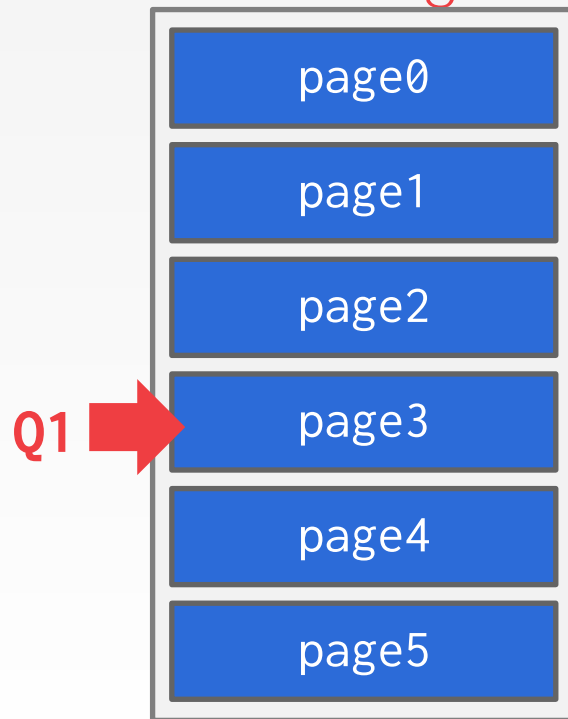
SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Buffer Pool



Disk Pages



SCAN SHARING

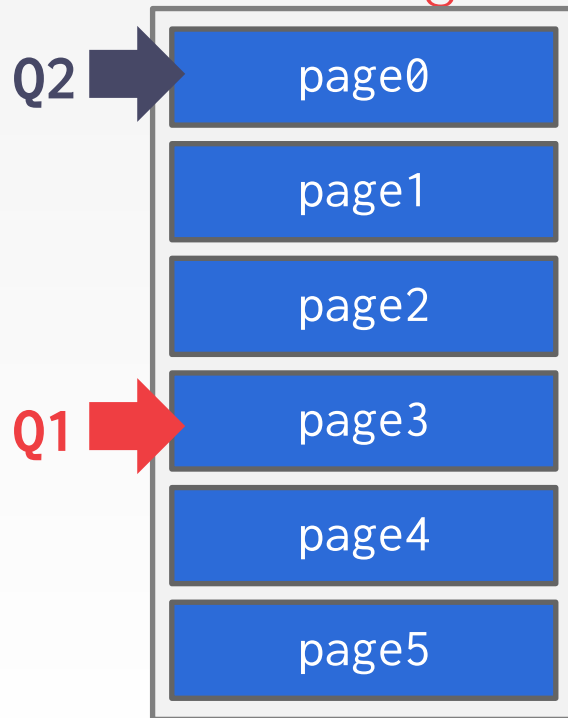
Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



Q2 Q1 →

SCAN SHARING

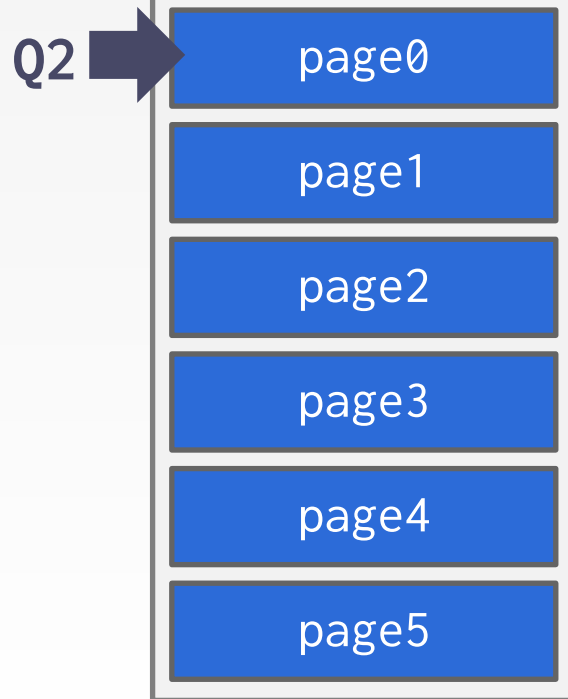
Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



SCAN SHARING

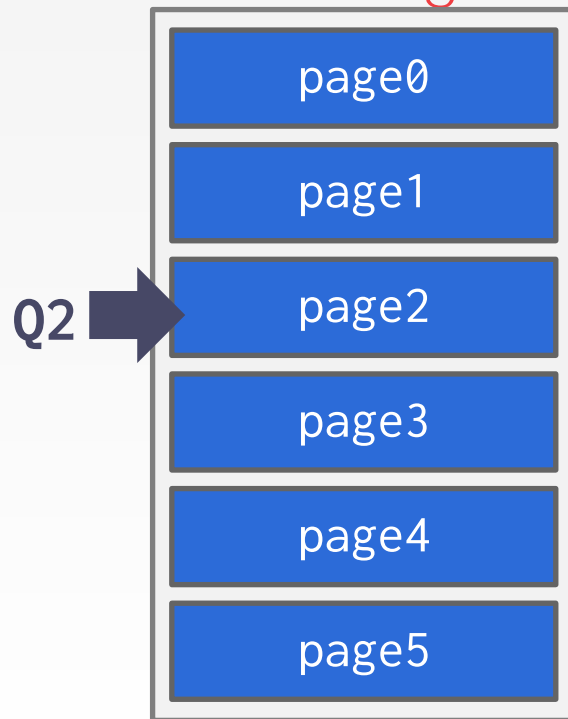
Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages

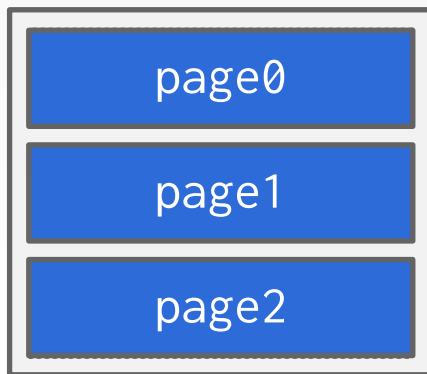


SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A LIMIT 100`

Buffer Pool



Disk Pages



BUFFER POOL BYPASS

The sequential scan operator will not store fetched pages in the buffer pool to avoid overhead.

- Memory is local to running query.
- Works well if operator needs to read a large sequence of pages that are contiguous on disk.
- Can also be used for temporary data (sorting, joins).

Called "Light Scans" in Informix.

ORACLE®

Microsoft®
SQL Server®

PostgreSQL

Informix®

OS PAGE CACHE

Most disk operations go through the OS API.

Unless you tell it not to, the OS maintains its own filesystem cache.

Most DBMSs use direct I/O (O_DIRECT) to bypass the OS's cache.

- Redundant copies of pages.
- Different eviction policies.

Demo: Postgres

BUFFER REPLACEMENT POLICIES

When the DBMS needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool.

Goals:

- Correctness
- Accuracy
- Speed
- Meta-data overhead



LEAST-RECENTLY USED

Maintain a single timestamp of when each page was last accessed.

When the DBMS needs to evict a page, select the one with the oldest timestamp.

→ Keep the pages in sorted order to reduce the search time on eviction.

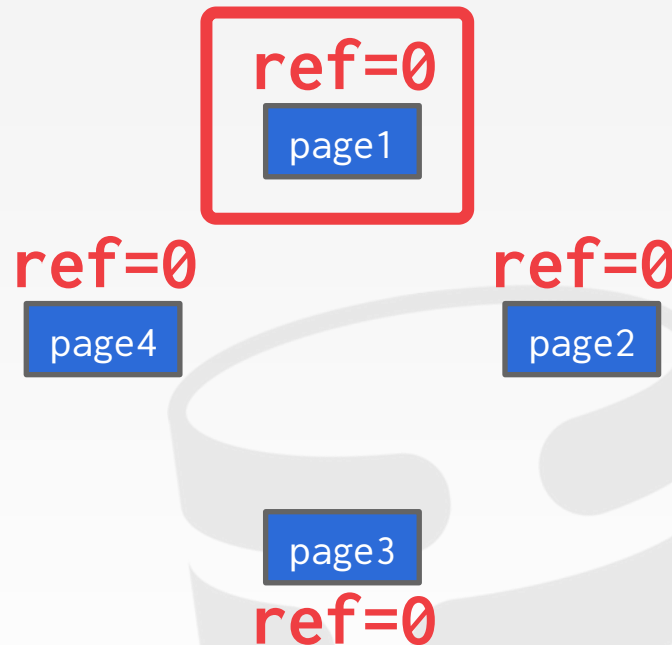
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



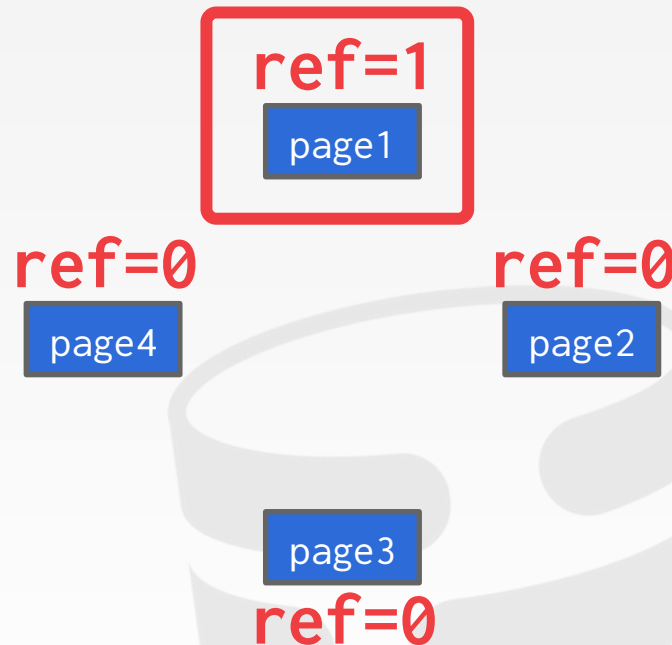
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



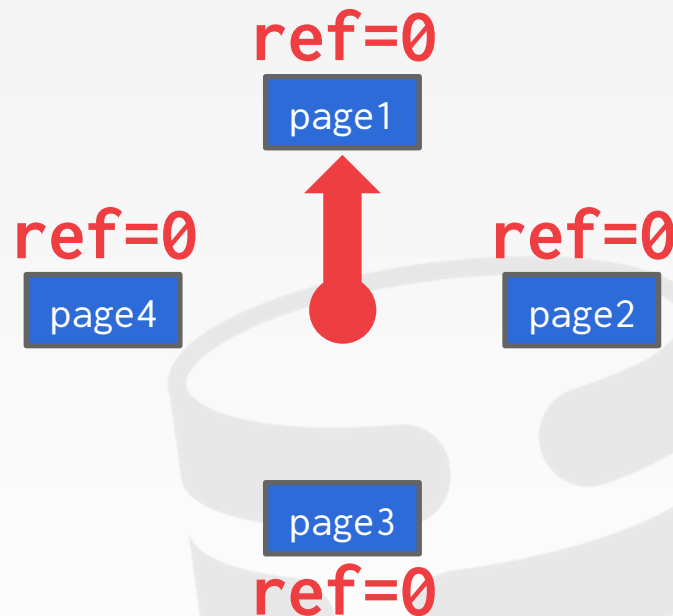
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



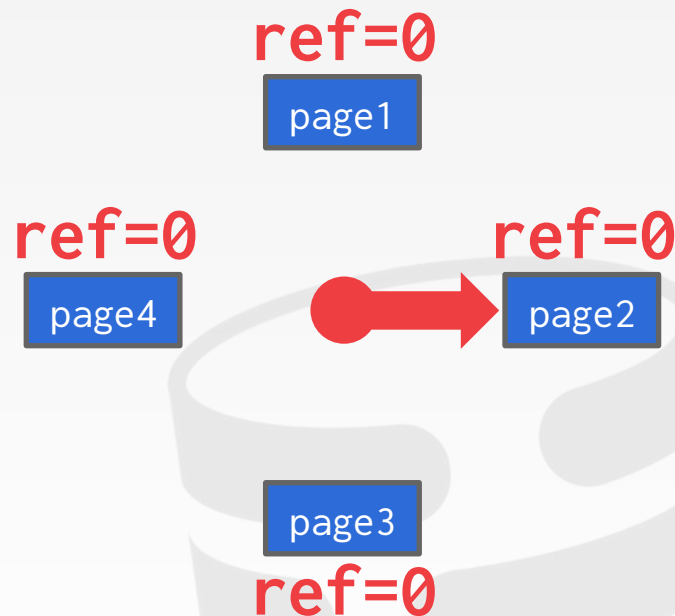
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



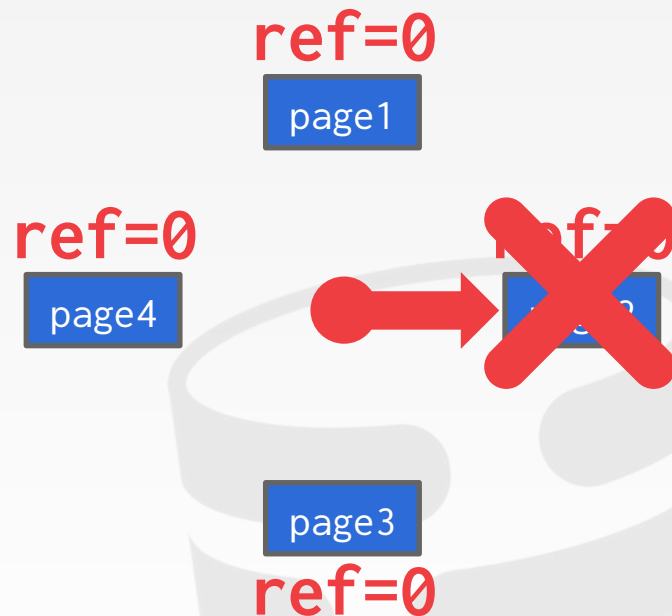
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



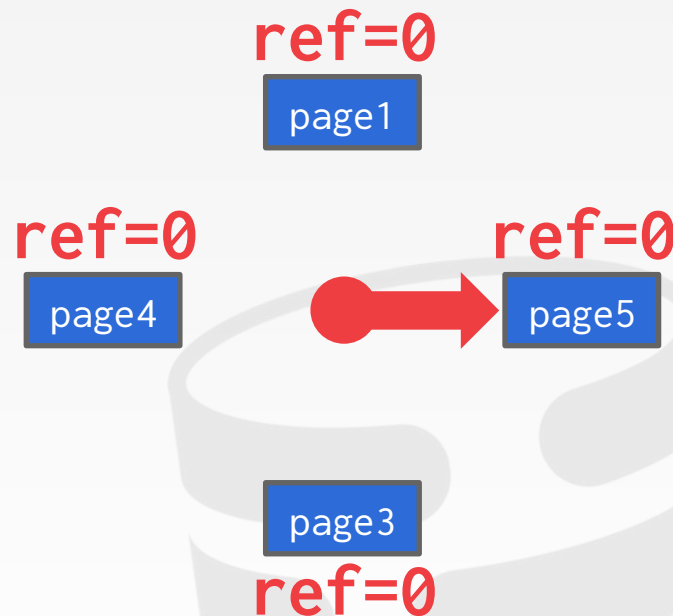
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



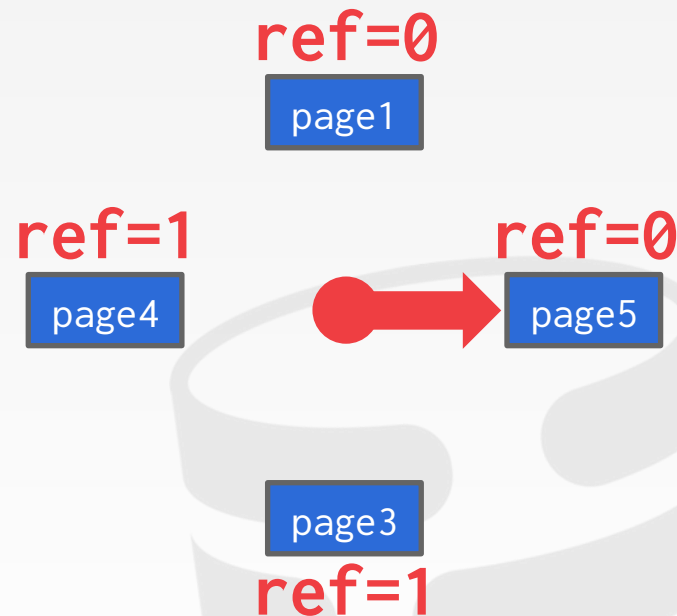
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



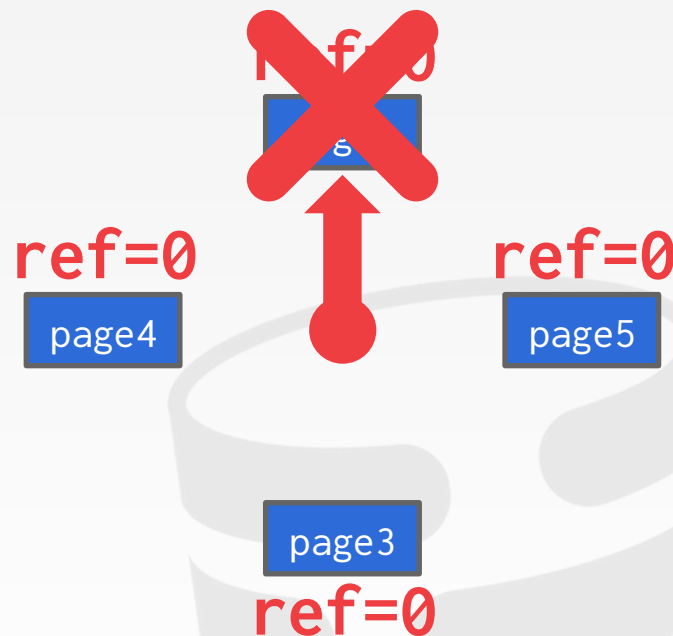
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



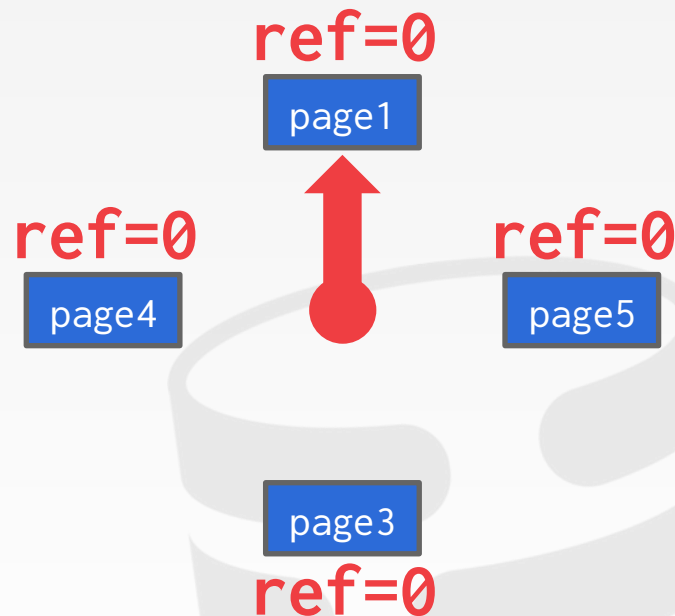
CLOCK

Approximation of LRU that does not need a separate timestamp per page.

- Each page has a reference bit.
- When a page is accessed, set to 1.

Organize the pages in a circular buffer with a "clock hand":

- Upon sweeping, check if a page's bit is set to 1.
- If yes, set to zero. If no, then evict.



PROBLEMS

LRU and CLOCK replacement policies are susceptible to sequential flooding.

- A query performs a sequential scan that reads every page.
- This pollutes the buffer pool with pages that are read once and then never again.

In some workloads the most recently used page is the most unneeded page.

SEQUENTIAL FLOODING

Q1 `SELECT * FROM A WHERE id = 1`

Buffer Pool



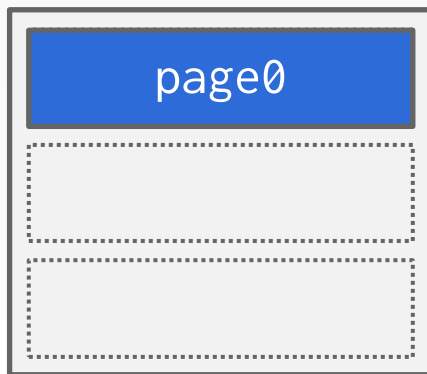
Disk Pages



SEQUENTIAL FLOODING

Q1 `SELECT * FROM A WHERE id = 1`

Buffer Pool



Disk Pages

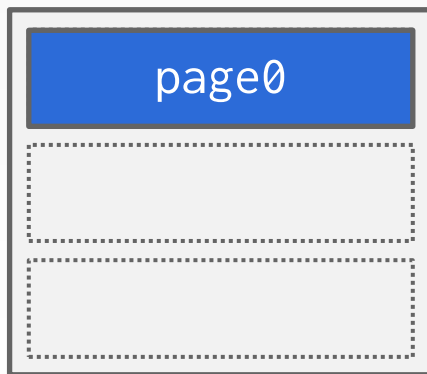


SEQUENTIAL FLOODING

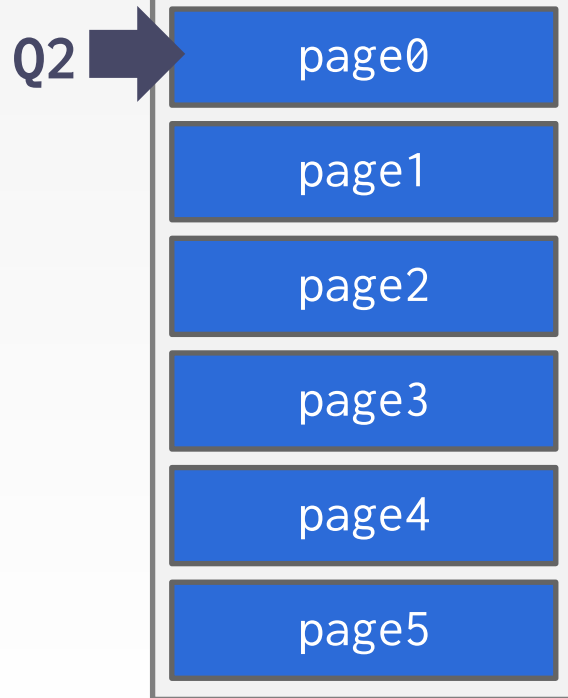
Q1 `SELECT * FROM A WHERE id = 1`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages

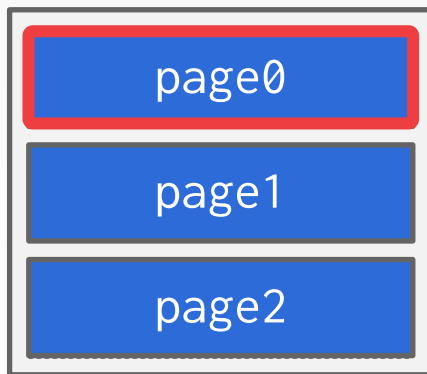


SEQUENTIAL FLOODING

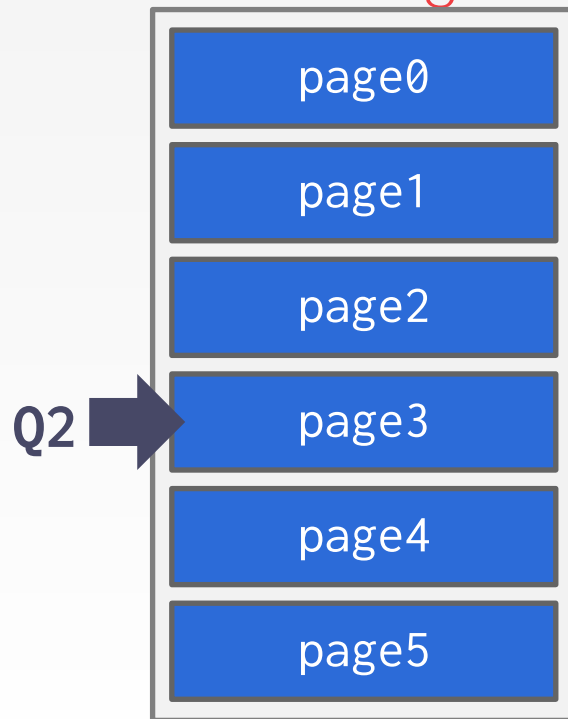
Q1 `SELECT * FROM A WHERE id = 1`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



SEQUENTIAL FLOODING

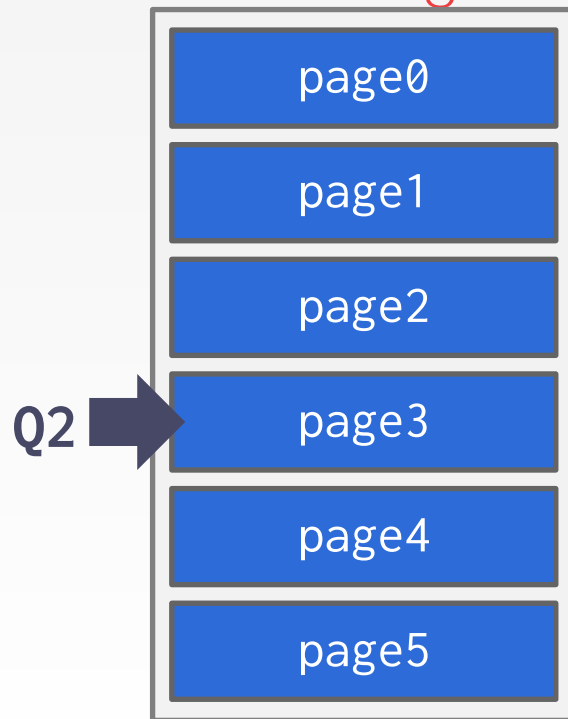
Q1 `SELECT * FROM A WHERE id = 1`

Q2 `SELECT AVG(val) FROM A`

Buffer Pool



Disk Pages



SEQUENTIAL FLOODING

Q1 `SELECT * FROM A WHERE id = 1`

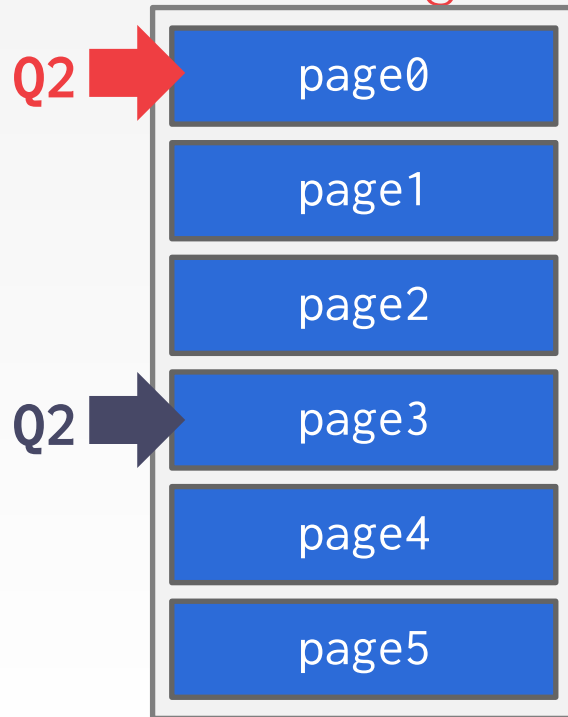
Q2 `SELECT AVG(val) FROM A`

Q3 `SELECT * FROM A WHERE id = 1`

Buffer Pool



Disk Pages



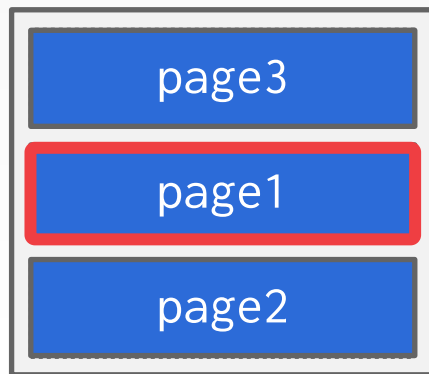
SEQUENTIAL FLOODING

Q1 `SELECT * FROM A WHERE id = 1`

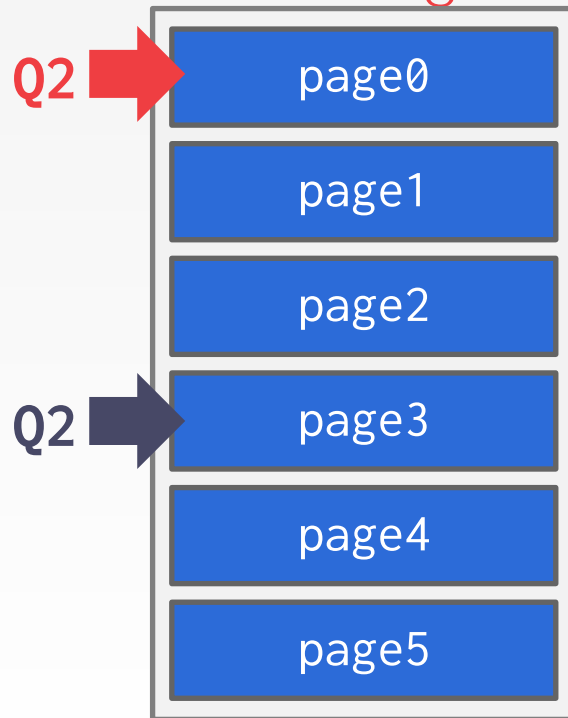
Q2 `SELECT AVG(val) FROM A`

Q3 `SELECT * FROM A WHERE id = 1`

Buffer Pool



Disk Pages



BETTER POLICIES: LRU-K

Track the history of last K references to each page as timestamps and compute the interval between subsequent accesses.

The DBMS then uses this history to estimate the next time that page is going to be accessed.

BETTER POLICIES: LOCALIZATION

The DBMS chooses which pages to evict on a per txn/query basis. This minimizes the pollution of the buffer pool from each query.

→ Keep track of the pages that a query has accessed.

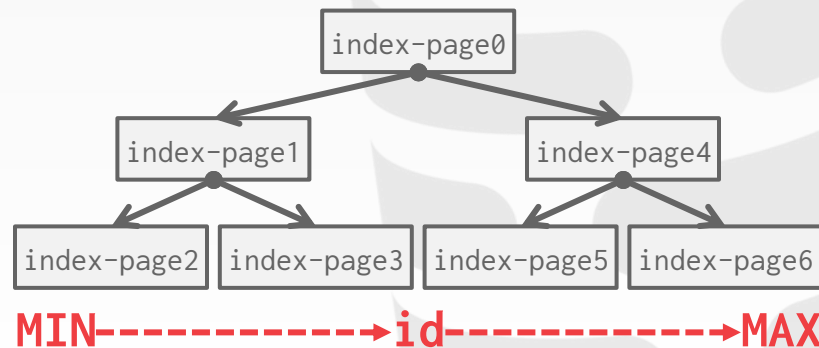
Example: Postgres maintains a small ring buffer that is private to the query.

BETTER POLICIES: PRIORITY HINTS

The DBMS knows what the context of each page during query execution.

It can provide hints to the buffer pool on whether a page is important or not.

Q1 INSERT INTO A VALUES (*id++*)

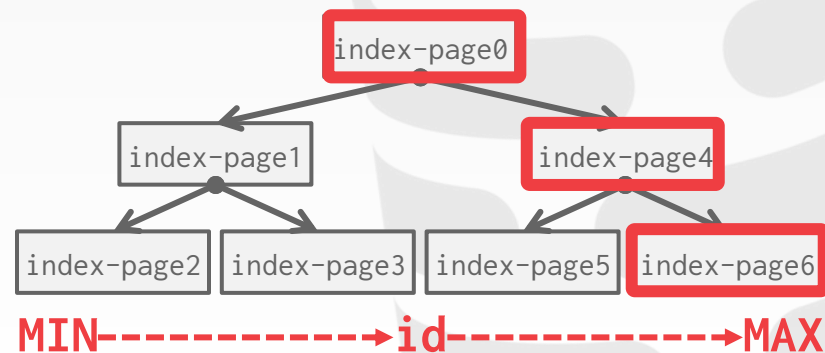


BETTER POLICIES: PRIORITY HINTS

The DBMS knows what the context of each page during query execution.

It can provide hints to the buffer pool on whether a page is important or not.

Q1 INSERT INTO A VALUES (*id*++)



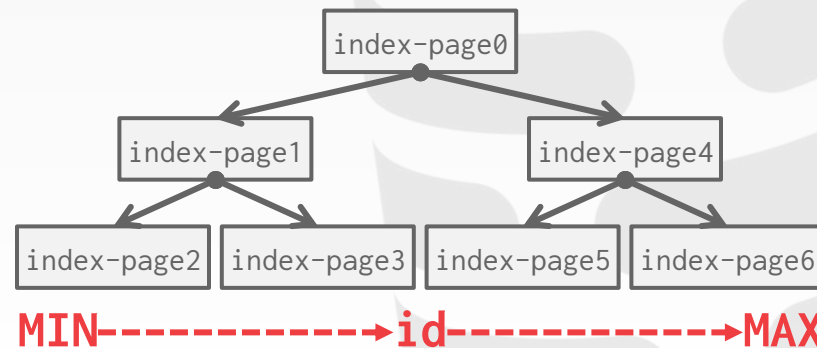
BETTER POLICIES: PRIORITY HINTS

The DBMS knows what the context of each page during query execution.

It can provide hints to the buffer pool on whether a page is important or not.

Q1 INSERT INTO A VALUES (*id++*)

Q2 SELECT * FROM A WHERE id = ?



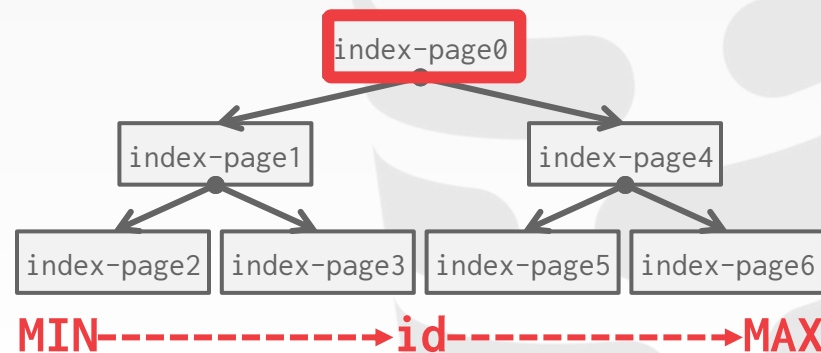
BETTER POLICIES: PRIORITY HINTS

The DBMS knows what the context of each page during query execution.

It can provide hints to the buffer pool on whether a page is important or not.

Q1 INSERT INTO A VALUES (*id++*)

Q2 SELECT * FROM A WHERE id = ?



DIRTY PAGES

FAST: If a page in the buffer pool is not dirty, then the DBMS can simply "drop" it.

SLOW: If a page is dirty, then the DBMS must write back to disk to ensure that its changes are persisted.

Trade-off between fast evictions versus dirty writing pages that will not be read again in the future.

BACKGROUND WRITING

The DBMS can periodically walk through the page table and write dirty pages to disk.

When a dirty page is safely written, the DBMS can either evict the page or just unset the dirty flag.

Need to be careful that we don't write dirty pages before their log records have been written...

OTHER MEMORY POOLS

The DBMS needs memory for things other than just tuples and indexes.

These other memory pools may not always be backed by disk. Depends on implementation.

- Sorting + Join Buffers
- Query Caches
- Maintenance Buffers
- Log Buffers
- Dictionary Caches



CONCLUSION

The DBMS can manage that sweet, sweet memory better than the OS.

Leverage the semantics about the query plan to make better decisions:

- Evictions
- Allocations
- Pre-fetching



NEXT CLASS

HASH TABLES!

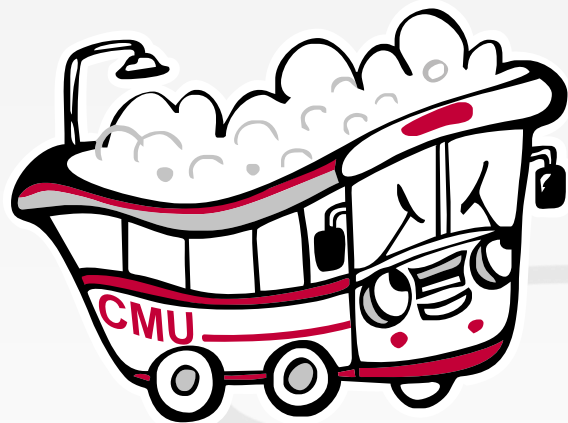


PROJECT #1

You will build the first component of your storage manager.

- LRU Replacement Policy
- Buffer Pool Manager

We will provide you with the disk manager and page layouts.



BusTub

Due Date:
Sunday Sept 27th @ 11:59pm

TASK #1 – LRU REPLACEMENT POLICY

Build a data structure that tracks the usage of **frame_ids** using the LRU policy.

General Hints:

- Your **LRUReplacer** needs to check the "pinned" status of a **Page**.
- If there are no pages touched since last sweep, then return the lowest page id.

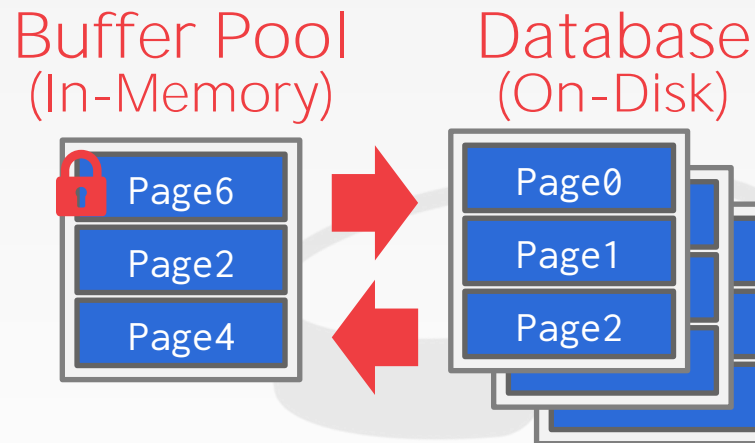
TASK #2 – BUFFER POOL MANAGER

Use your LRU replacer to manage the allocation of pages.

- Need to maintain an internal data structures of allocated + free pages.
- We will provide you components to read/write data from disk.
- Use whatever data structure you want for the page table.

General Hints:

- Make sure you get the order of operations correct when pinning.



GETTING STARTED

Download the source code from [GitHub](#).

Make sure you can build it on your machine.

- We have tested Ubuntu, OSX, and Windows (WSL2).
- We are also providing a Docker and Vagrant file to setup your environment.
- It does **not** compile on the Andrews machines. Please contact me if this is a problem.

THINGS TO NOTE

Do **not** change any file other than the four that you must hand in.

The projects are cumulative.

We will **not** be providing solutions.

Post your questions on Piazza or come to our office hours. We will **not** help you debug.

CODE QUALITY

We will automatically check whether you are writing good code.

- [Google C++ Style Guide](#)
- [Doxygen Javadoc Style](#)

You need to run these targets before you submit your implementation to Gradescope.

- **make format**
- **make check-lint**
- **make check-censored**
- **make check-clang-tidy**

EXTRA CREDIT

Gradescope Leaderboard runs your code with a specialized in-memory version of BusTub.

The top 20 fastest implementations in the class will receive extra credit for this assignment.

- **#1:** 50% bonus points
- **#2–10:** 25% bonus points
- **#11–20:** 10% bonus points

Student with the most bonus points at the end of the semester will receive a BusTub shirt!

PLAGIARISM WARNING

Your project implementation must be your own work.

- You may not copy source code from other groups or the web.
- Do not publish your implementation on GitHub.

Plagiarism will not be tolerated.
See [CMU's Policy on Academic Integrity](#) for additional information.

