Intro to Database Systems (15-445/645)

06 Memory Management

Carnegie Mellon University

SPRING 2023

Charlie Garrod

ADMINISTRIVIA

Homework 1 due last Friday (Feb 3rd).

Homework 2 available today, due February 17th.

Project 1 available, due February 19th.

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

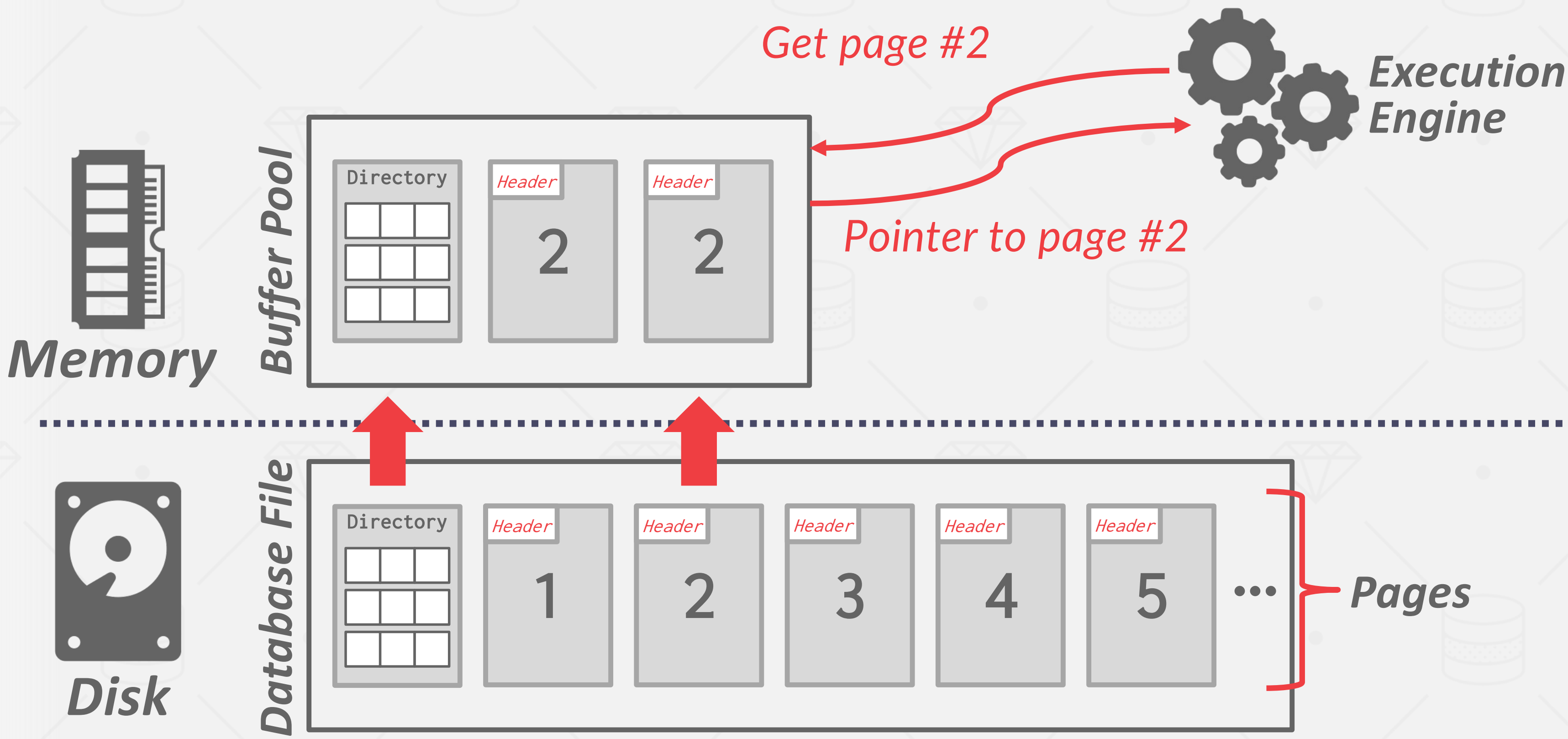
Spatial Control:

→ 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 to minimize the number of stalls from having to read data from disk.

DISK-ORIENTED DBMS



TODAY'S AGENDA

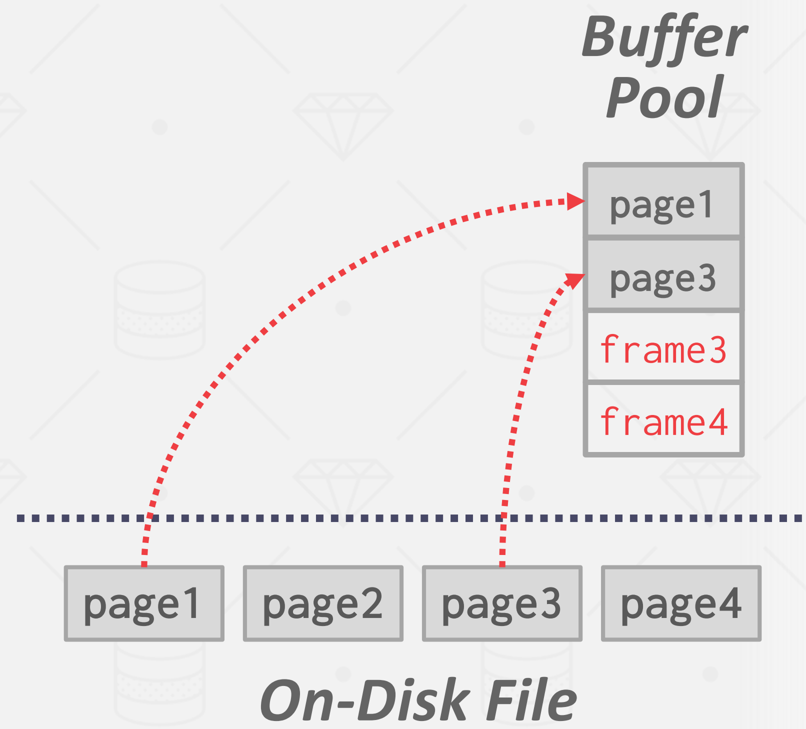
Buffer Pool Manager
Replacement Policies
Other Memory Pools

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.

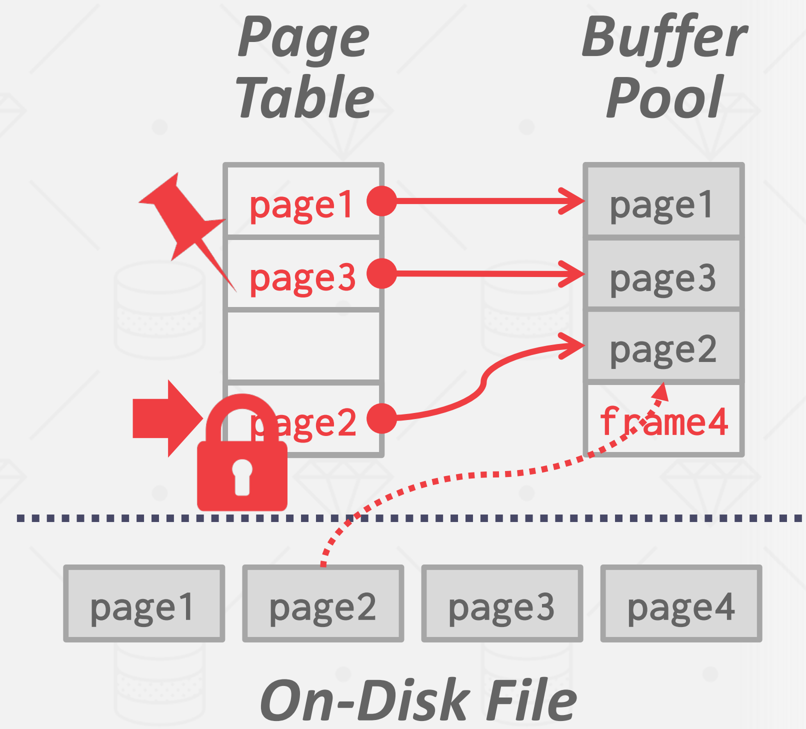
Dirty pages are buffered and not written to disk immediately
→ Write-Back Cache



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:

→ Prote

transa

→ Held

→ Need

Latche

→ Prote

struc

→ Held

→ Do n

utex

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 queries.

Local Policies:

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

→ 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

Partitioning memory across multiple pools helps reduce latch contention and improve locality.

MULTIPLE BUFFER POOLS

DB2

```
CREATE BUFFERPOOL custom_pool
   SIZE 250 PAGESIZE 8k;

CREATE TABLESPACE custom_tablespace
   PAGESIZE 8k BUFFERPOOL custom_pool;

CREATE TABLE new_table
   TABLESPACE custom_tablespace ( ... );
```

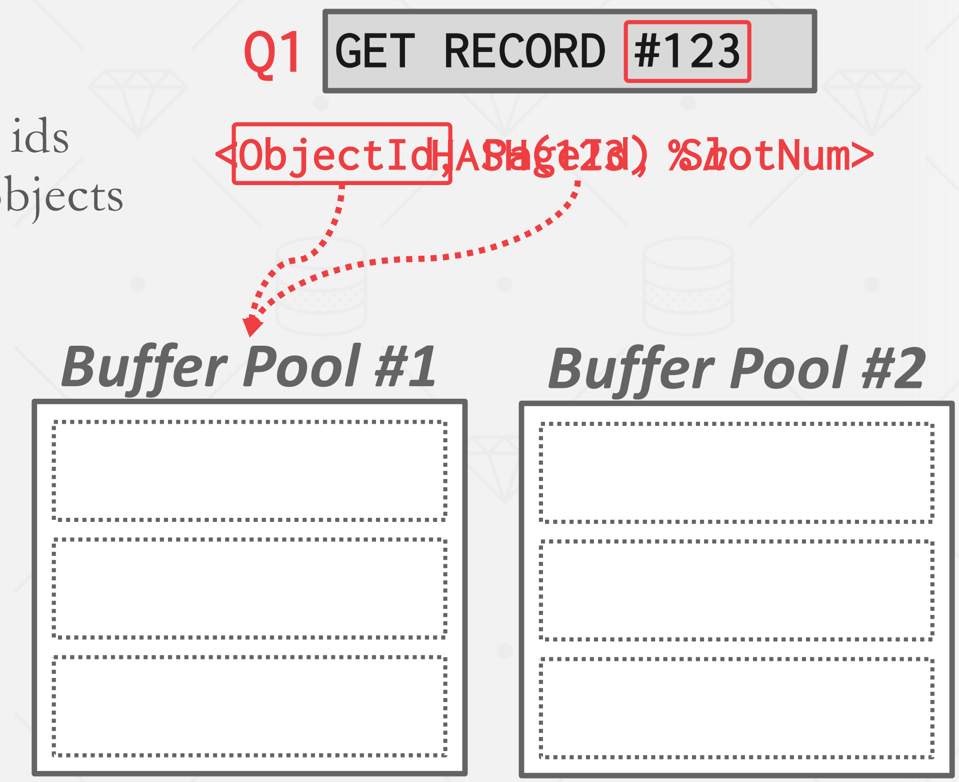
MULTIPLE BUFFER POOLS

Approach #1: Object Id

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

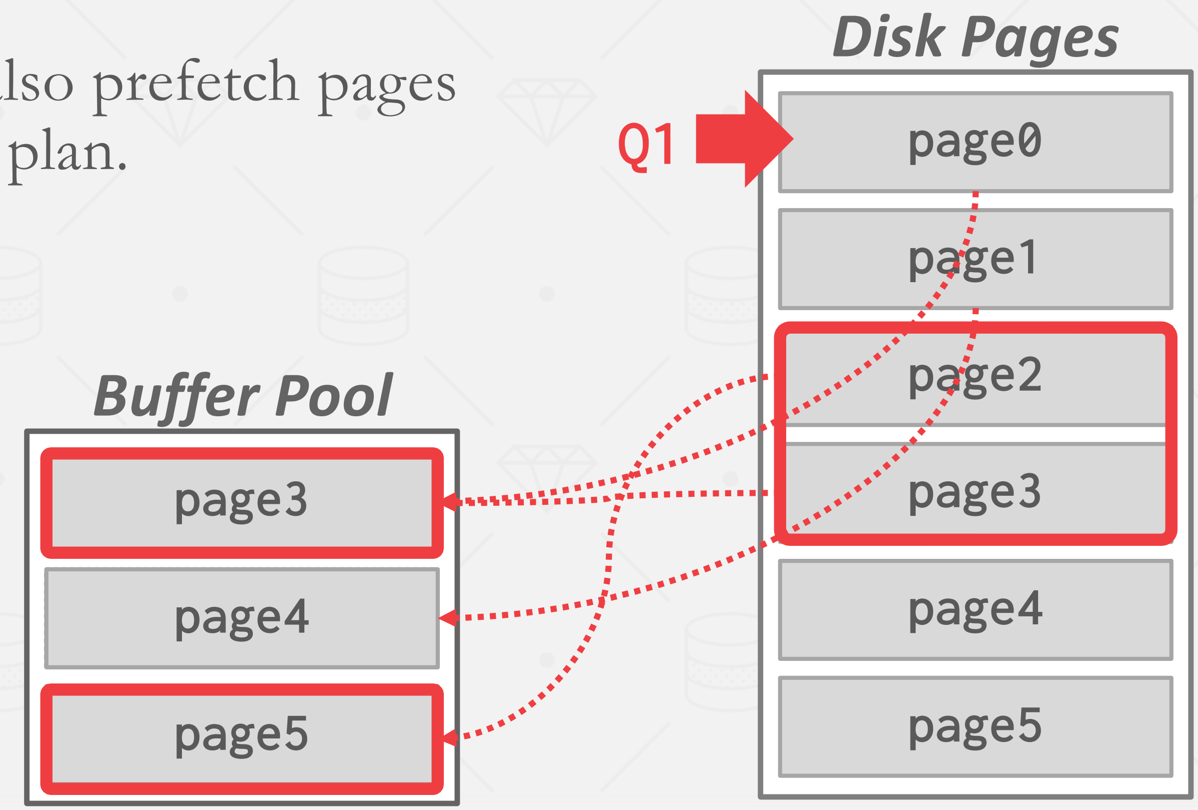
Approach #2: Hashing

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

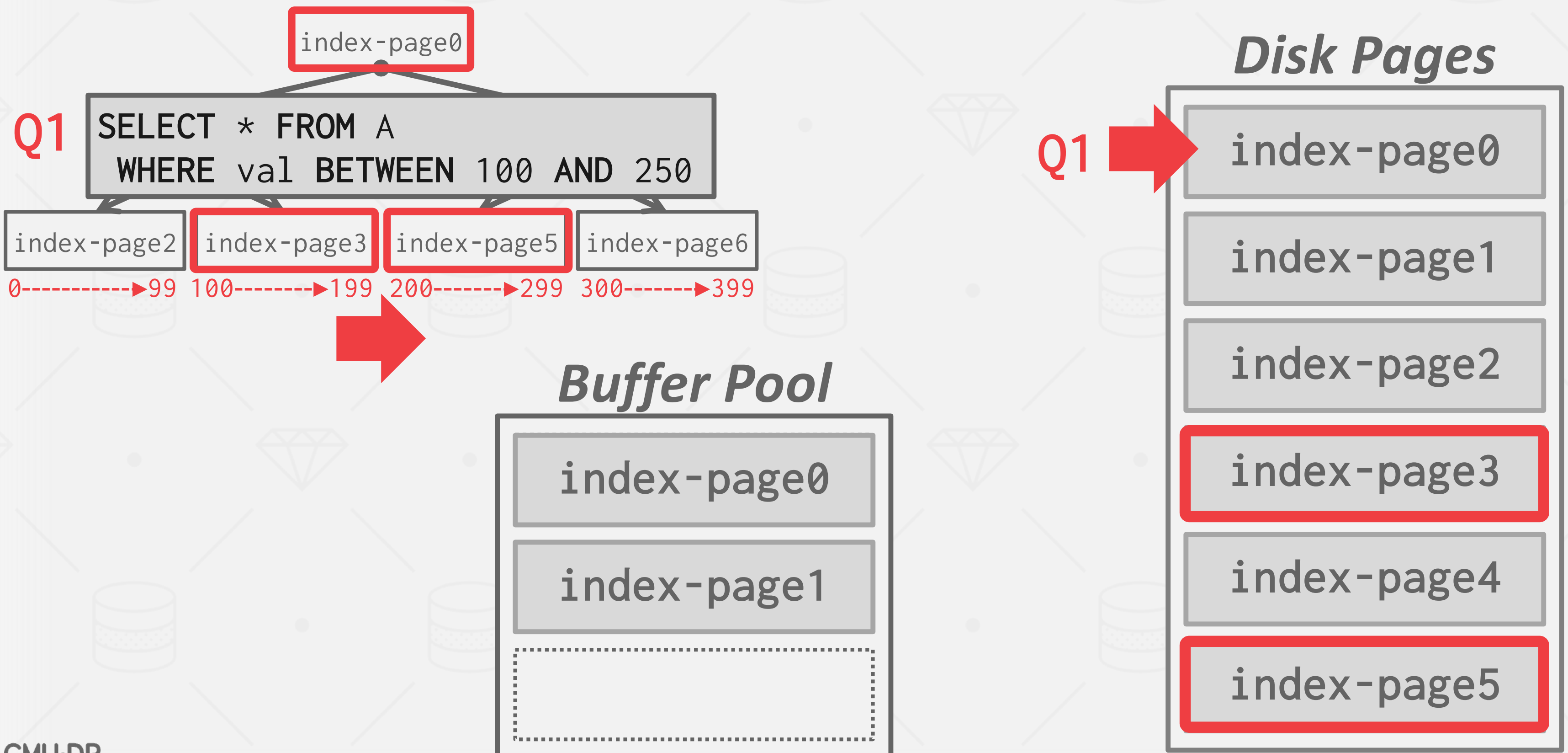


PRE-FETCHING

The DBMS can also prefetch pages based on a query plan.
→ Sequential Scans
→ Index Scans



PRE-FETCHING



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:

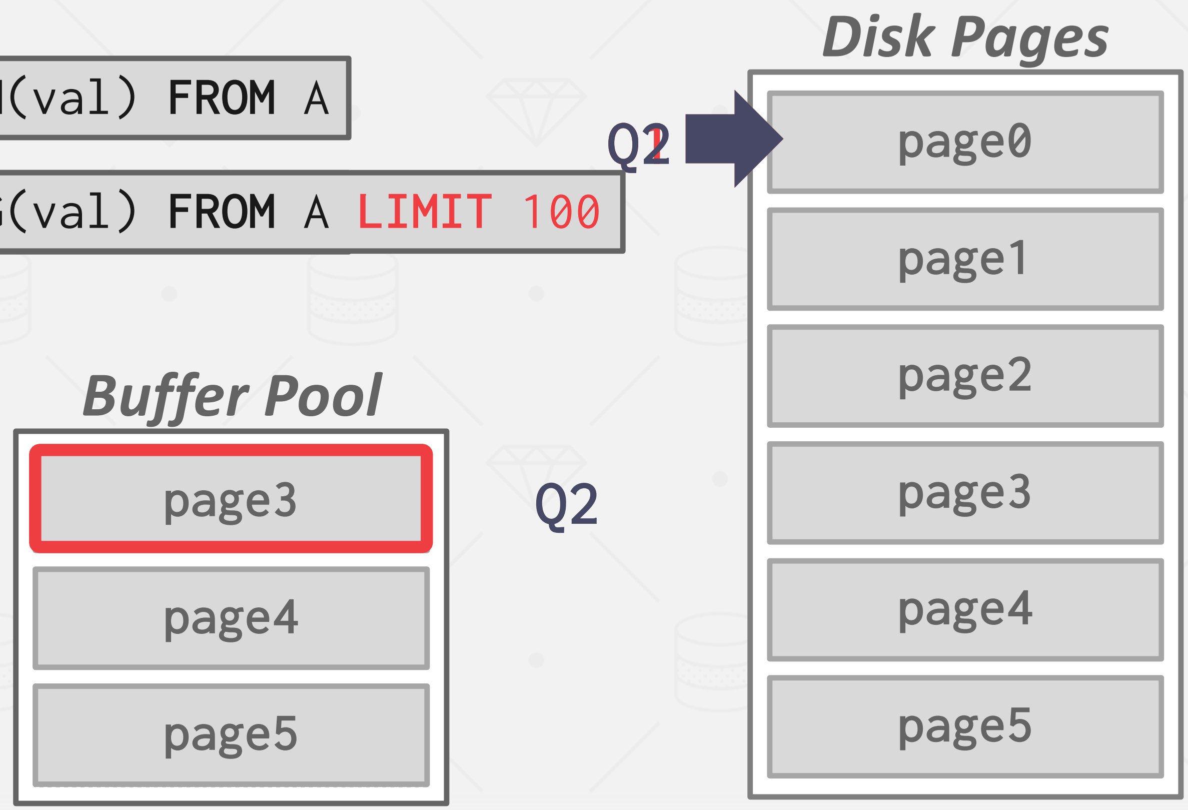
→ Fully supported in IBM DB2, MSSQL, and Postgres.

→ Oracle only supports cursor sharing for identical queries.

SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

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



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).

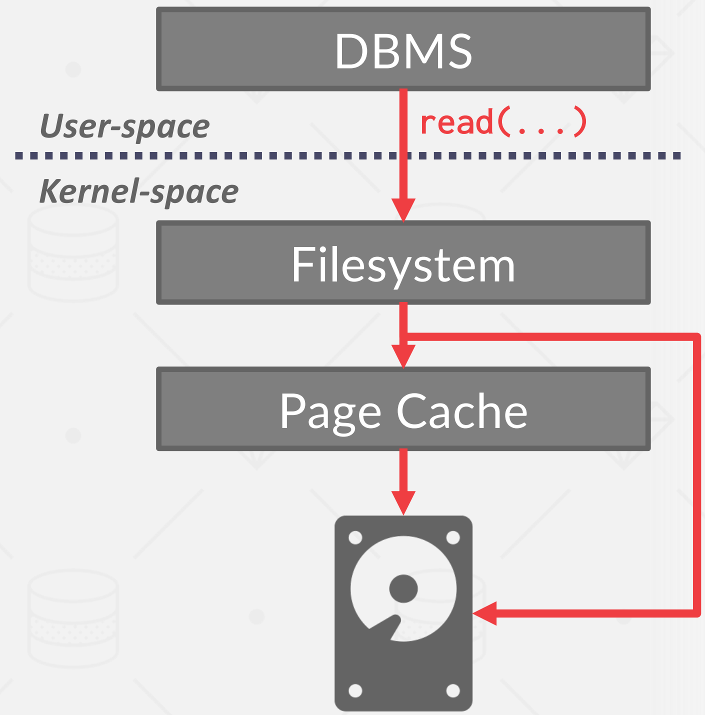
Called "Light Scans" in Informix.

OS PAGE CACHE

Most disk operations go through the OS API. Unless the DBMS tells it not to, the OS maintains its own filesystem cache (aka page cache, buffer cache).

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

→ Redundant copies of pages.
→ Different eviction policies.
→ Loss of control over file I/O.



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.

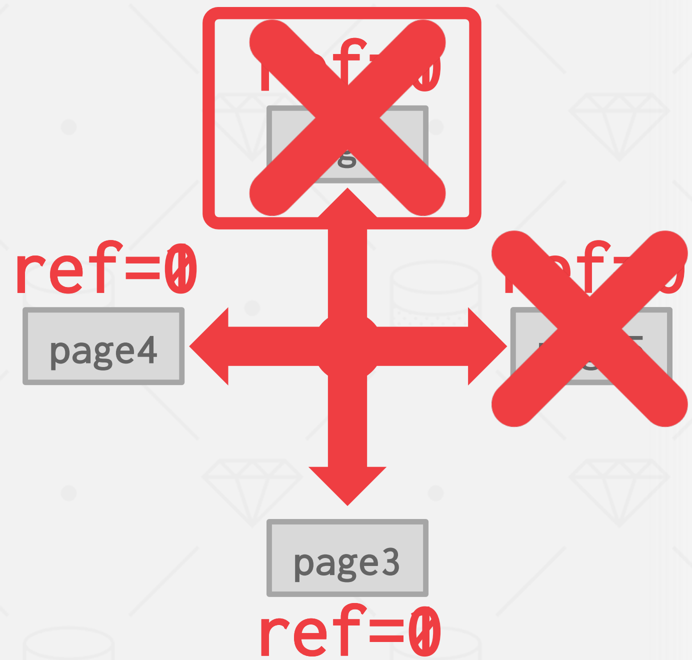
When the DBMS needs to evict a page, select the one with the oldest timestamp.

→ 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.



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.

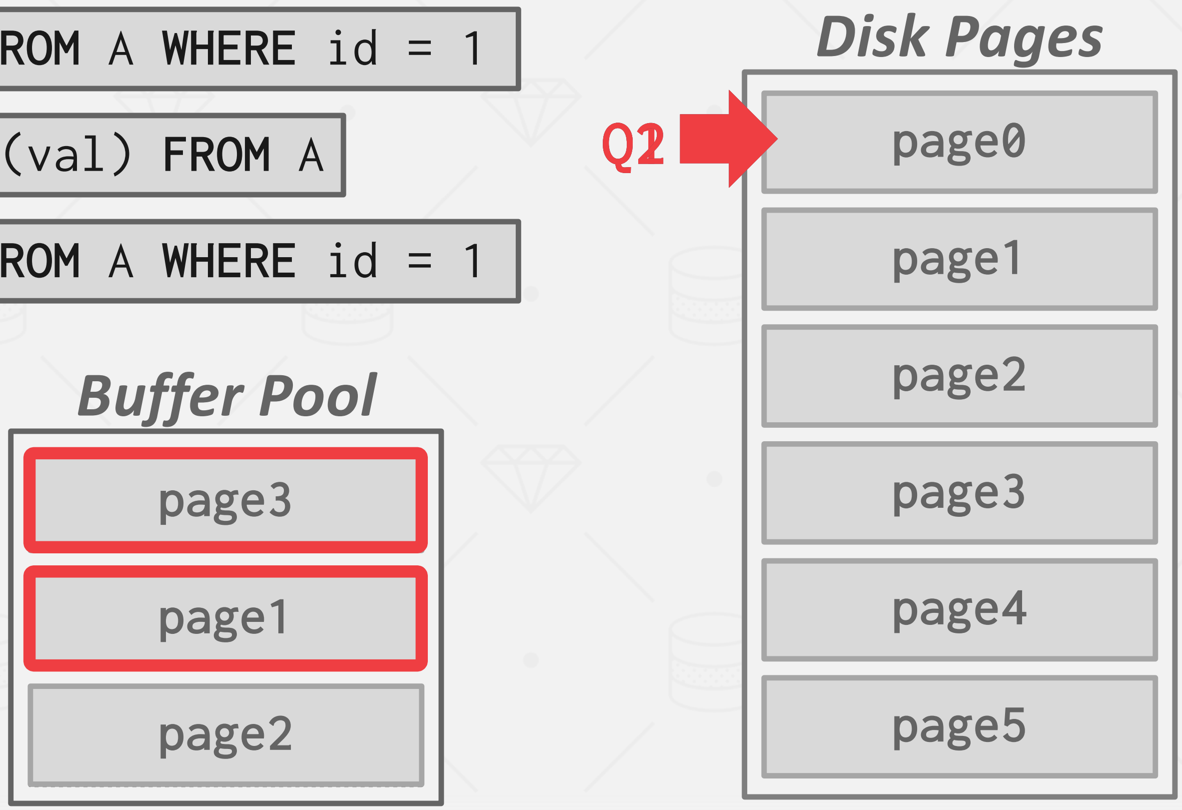
In some workloads the most recently used page is the most unneeded page.

SEQUENTIAL FLOODING

Q1 `SELECT * FROM A WHERE id = 1`

Q2 `SELECT AVG(val) FROM A`

Q3 `SELECT * FROM A WHERE id = 1`



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 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.

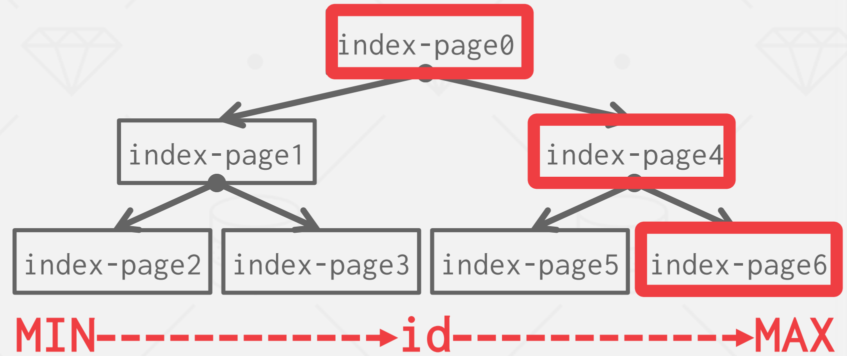
BETTER POLICIES: PRIORITY HINTS

The DBMS knows about 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 Path: If a page in the buffer pool is <u>not</u> dirty, then the DBMS can simply "drop" it.

Slow Path: 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 the system doesn't write dirty pages before their log records are written…

OTHER MEMORY POOLS

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

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

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

CONCLUSION

The DBMS can almost always manage memory better than the OS.

Leverage the semantics about the query plan to make better decisions:
→ Evictions
→ Allocations
→ Pre-fetching

NEXT CLASS

Hash Tables