15-721 DATABASE SYSTEMS

Lecture #20 – Scan Sharing

Andy Pavlo // Carnegie Mellon University // Spring 2016

COURSE ANNOUNCEMENTS

Mid-term Grades

MemSQL Project Machines

Extra Credit

SpliceMachine Tech Talk

→ April 15th @ 12:00pm

→ GHC 6115

→ http://db.cs.cmu.edu/events/monte-zweben-splice-machine/

PROJECT #3 UPDATES

No class on **April 6th.**

Each team will sign up to meet with me for 20 minutes during the class period.

In-class project update presentations will be on **April 13th.**

TODAY'S AGENDA

Background
Scan Sharing
Continuous Scanning
Work Sharing

OBSERVATION

Running multiple OLAP queries concurrently can saturate the I/O bandwidth (disk, memory) if they access different parts of a table at the same time.

SCAN SHARING

Queries are able to reuse data retrieved from storage or operator computations.

→ This is different from result caching.

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

→ Queries do not have to be exactly the same.

→ Can also share intermediate results.

DISK-BASED SCAN SHARING

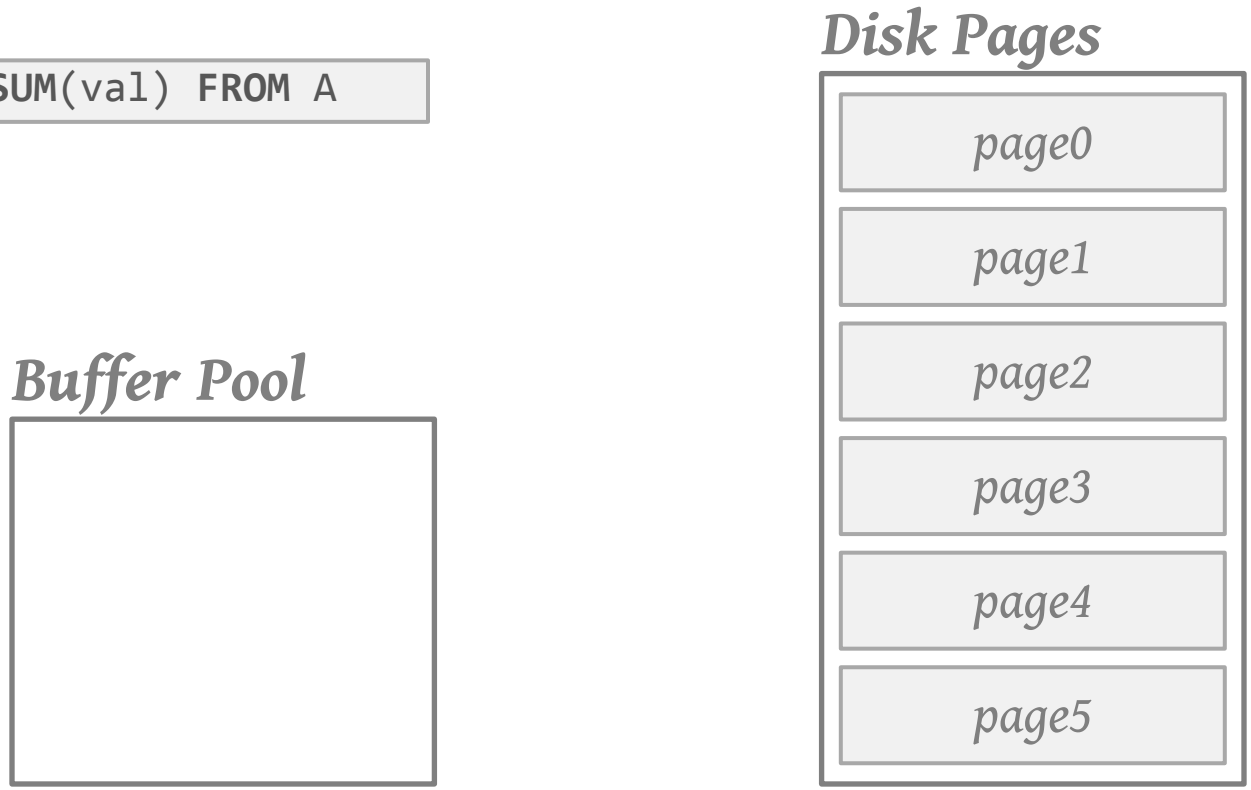
If a query needs to perform a scan and if there one already doing this, then the DBMS will attach the second query to that scan cursor.

→ The DBMS keeps track of where the second query joined with the first so that it can finish the scan when it reaches the end of the data structure.

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

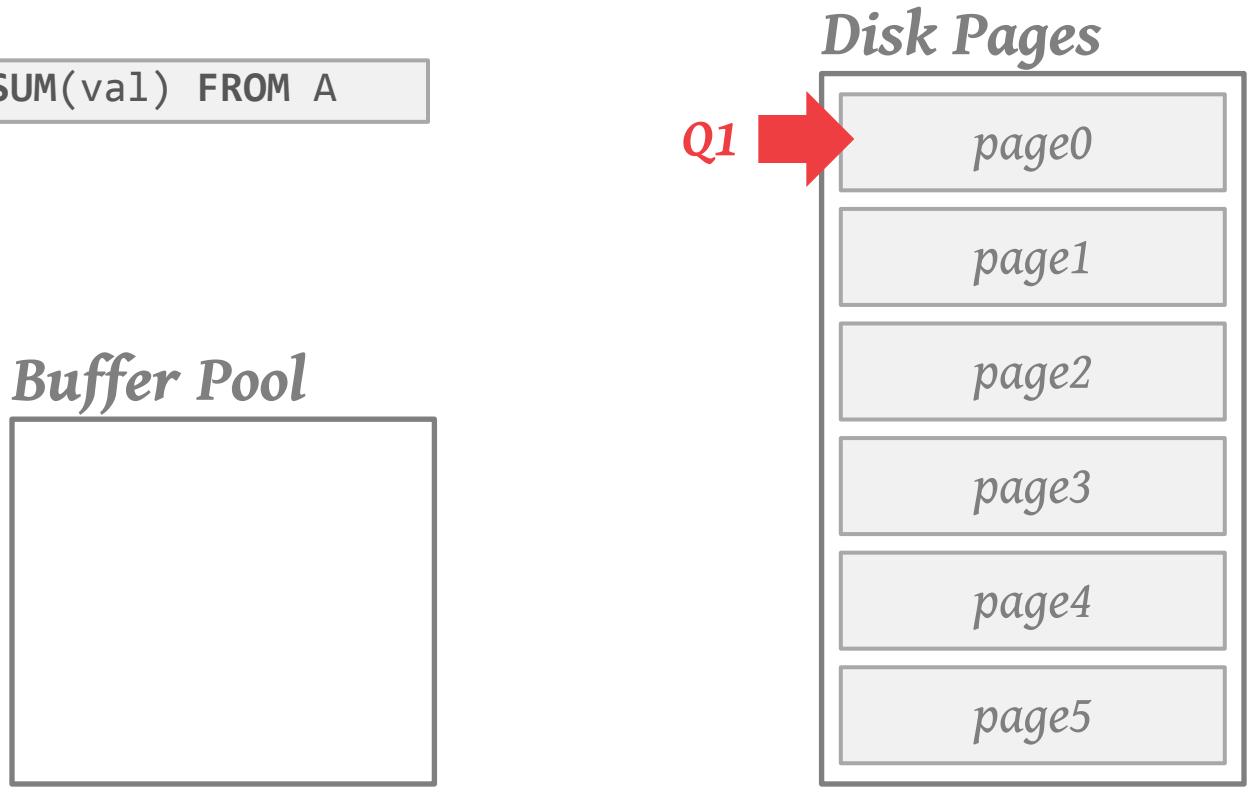
DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



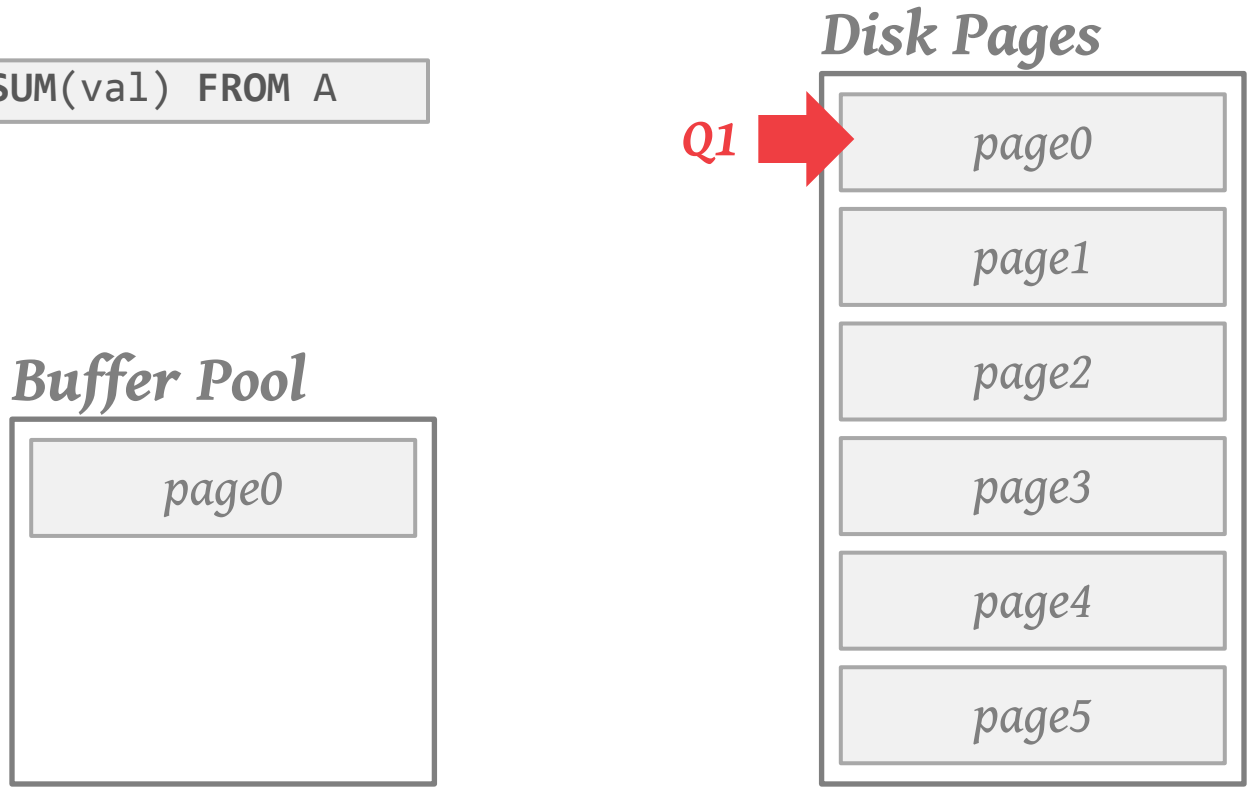
DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



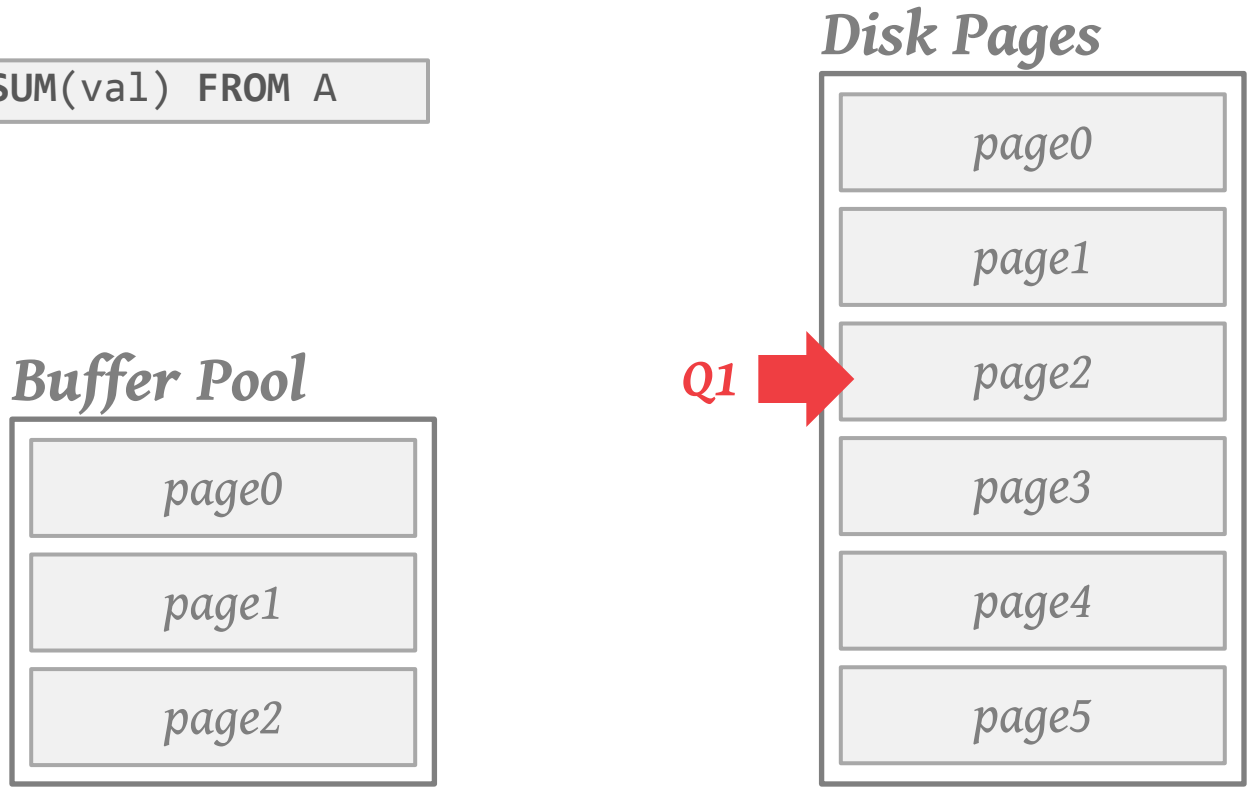
DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



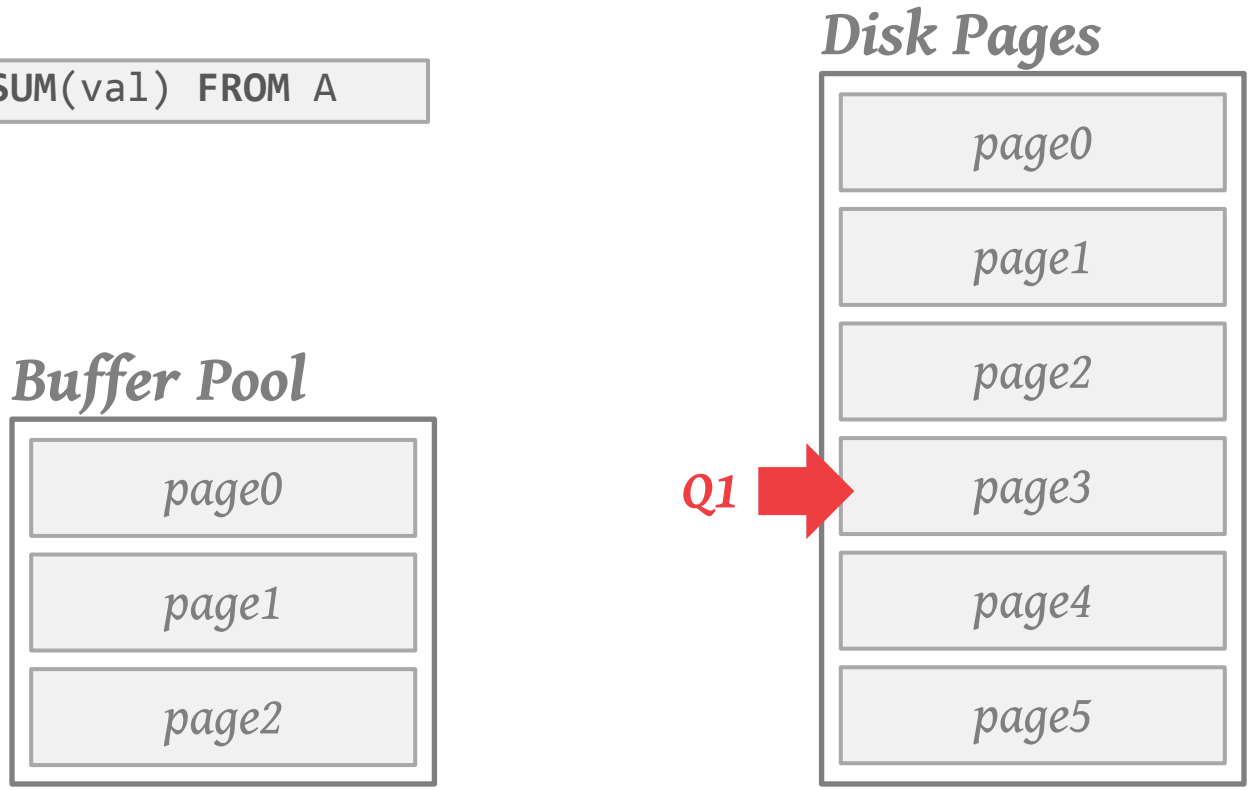
DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



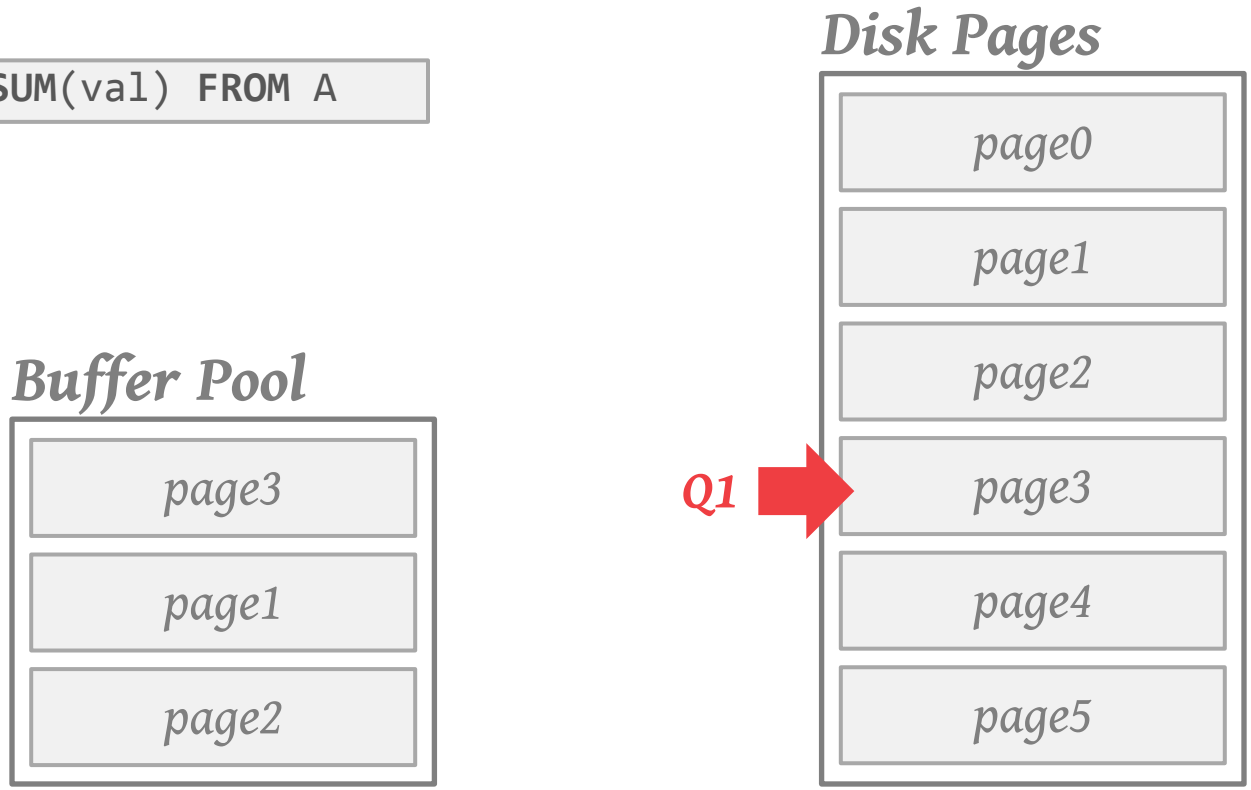
DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`



DISK-BASED SCAN SHARING

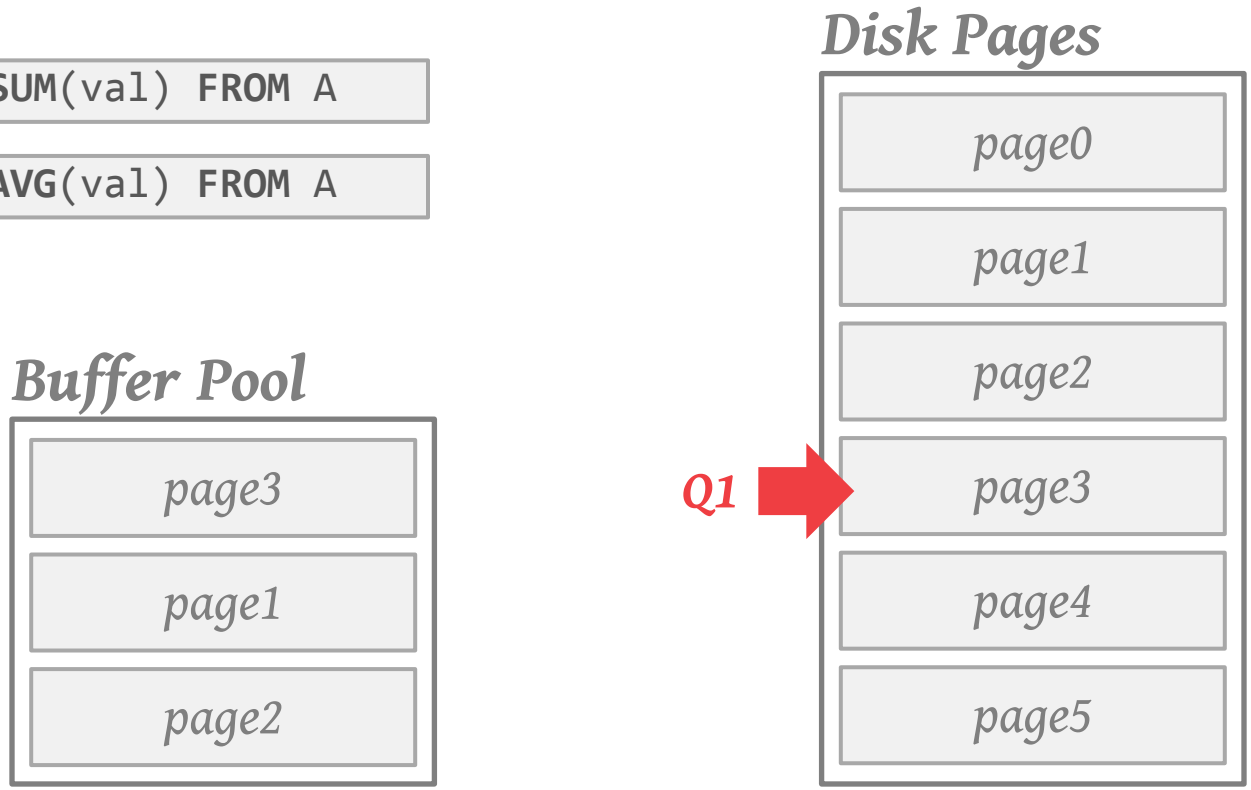
Q1 `SELECT SUM(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

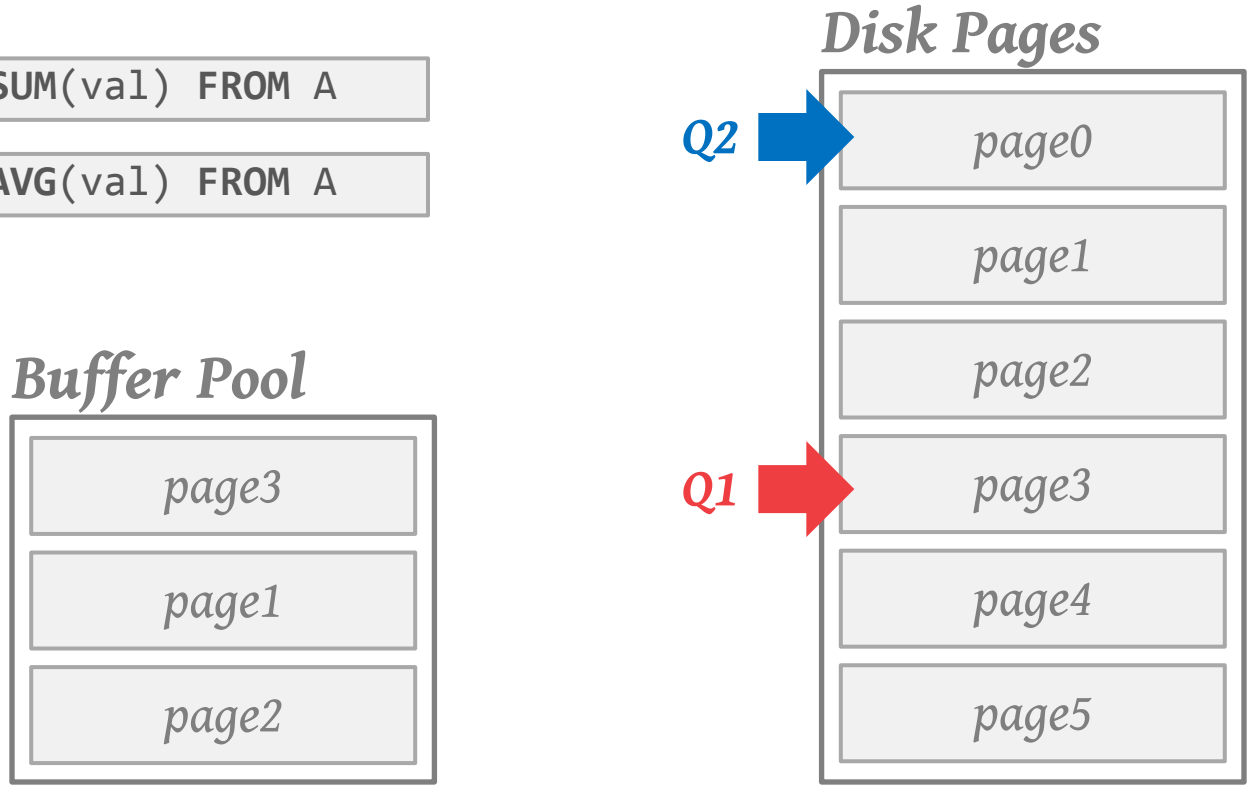
Q2 `SELECT AVG(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

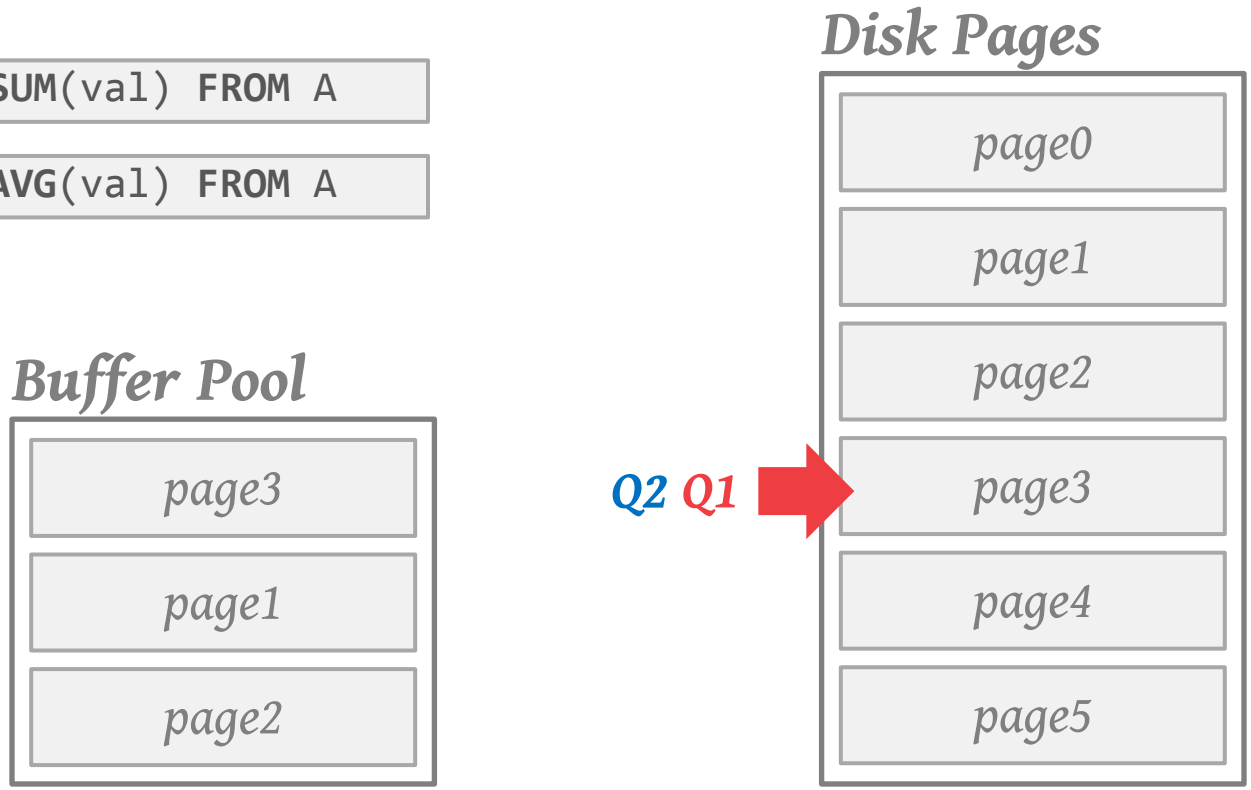
Q2 `SELECT AVG(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

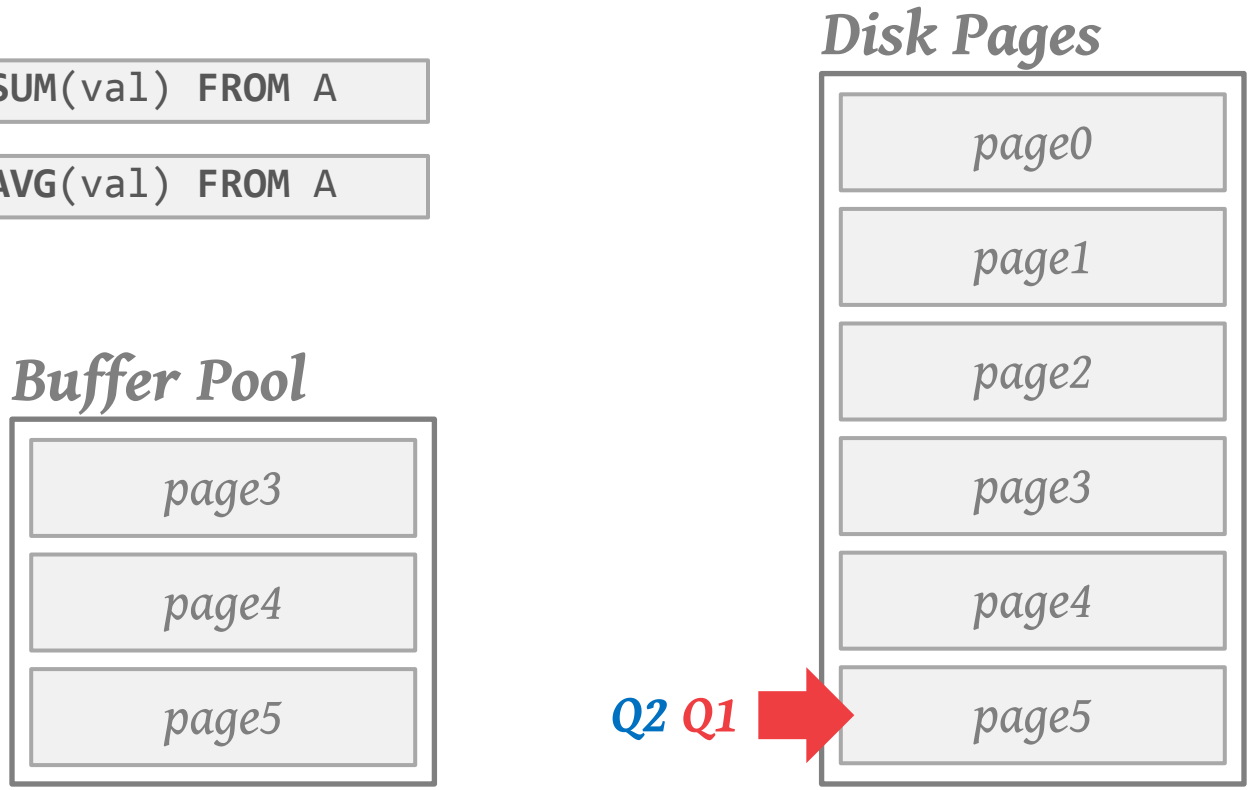
Q2 `SELECT AVG(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

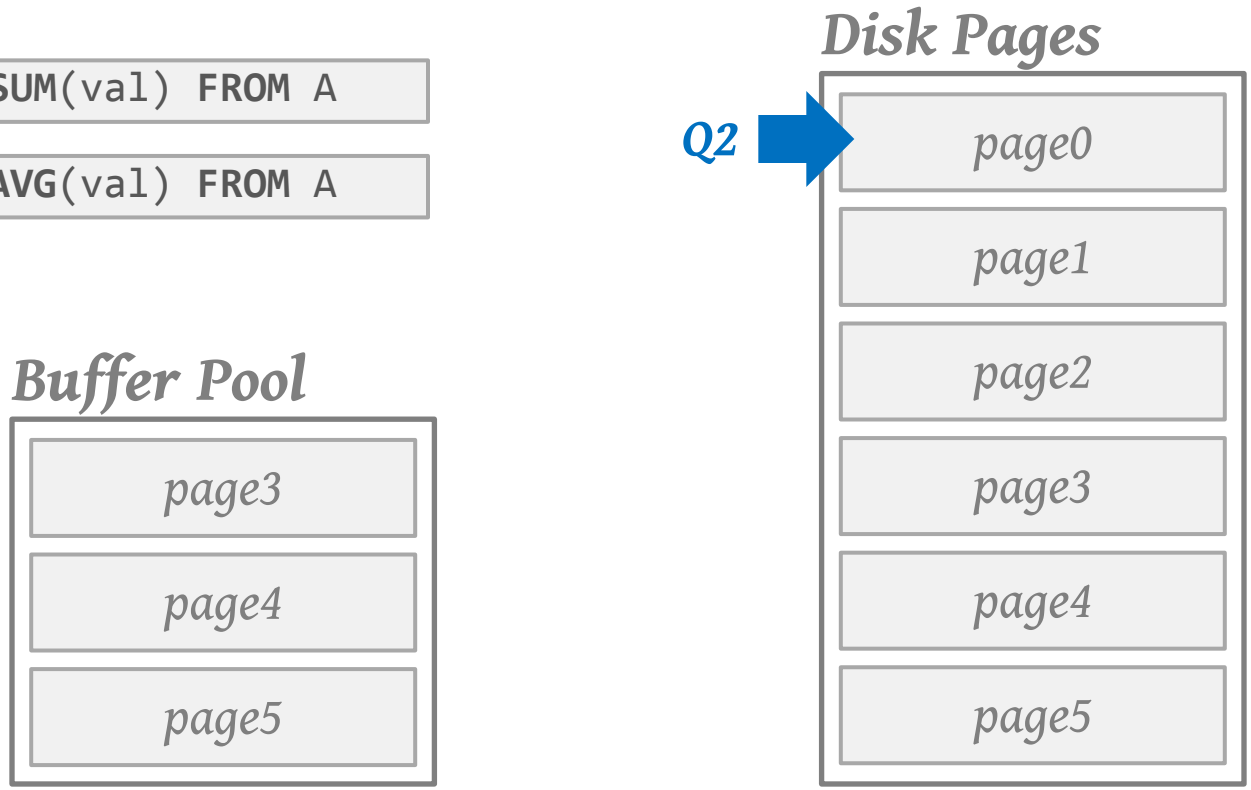
Q2 `SELECT AVG(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

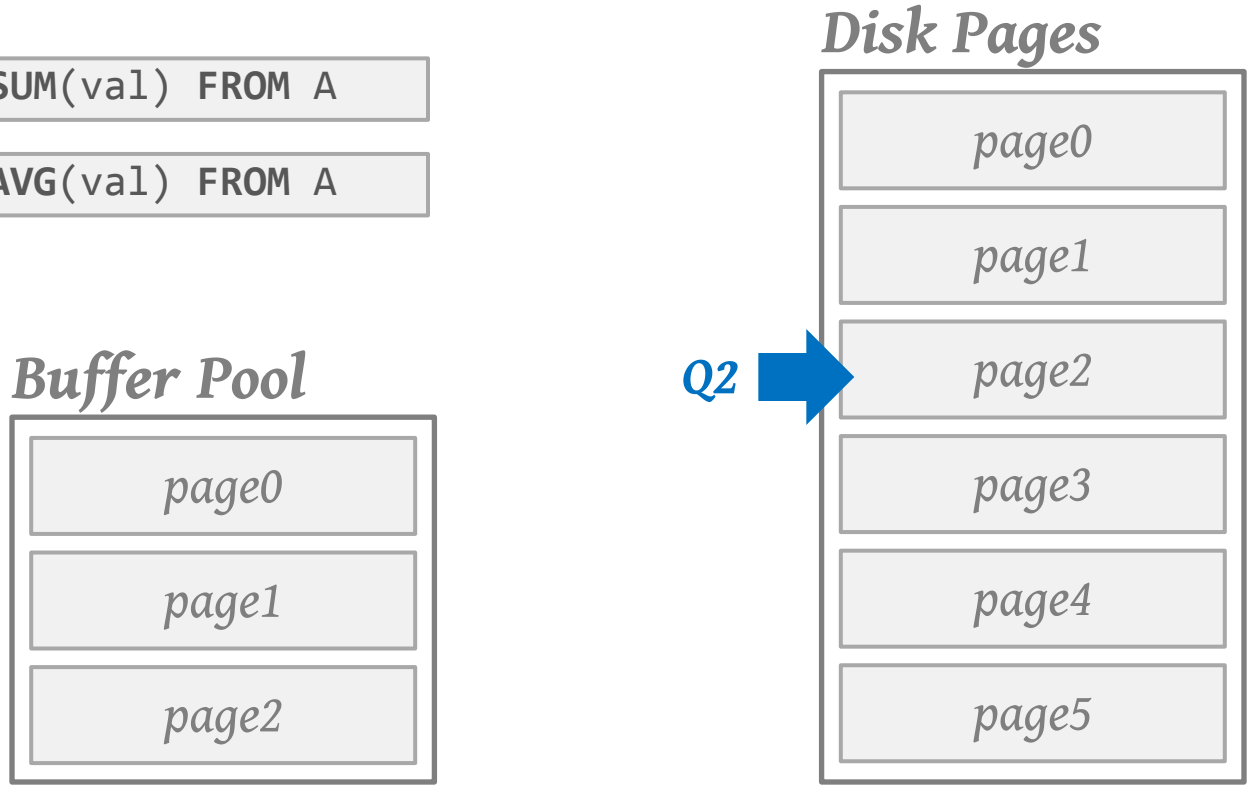
Q2 `SELECT AVG(val) FROM A`



DISK-BASED SCAN SHARING

Q1 `SELECT SUM(val) FROM A`

Q2 `SELECT AVG(val) FROM A`



OBSERVATION

Scan sharing is easier to implement in a disk-based DBMS because they support programmable buffer pools and dedicated threads for I/O.

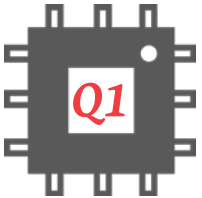
In-memory DBMSs do not have full control of CPU caches and those caches are much smaller than DRAM.

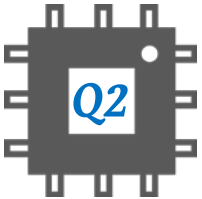
IN-MEMORY SCAN SHARING

The goal of scan sharing in an in-memory DBMS is to reduce the number of cache misses.

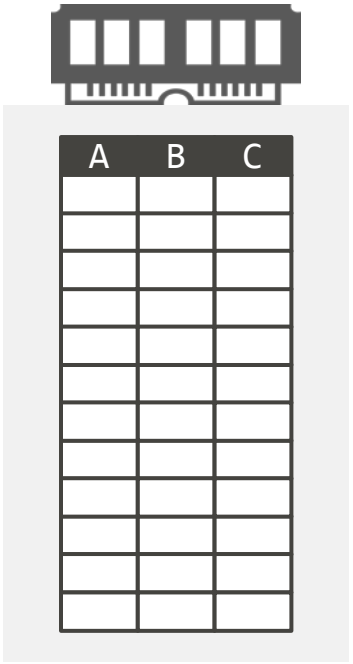
Have to be careful about not running too many queries at once because the intermediate results and auxiliary data structures will exceed cache sizes.

CONVOY PHENOMENON

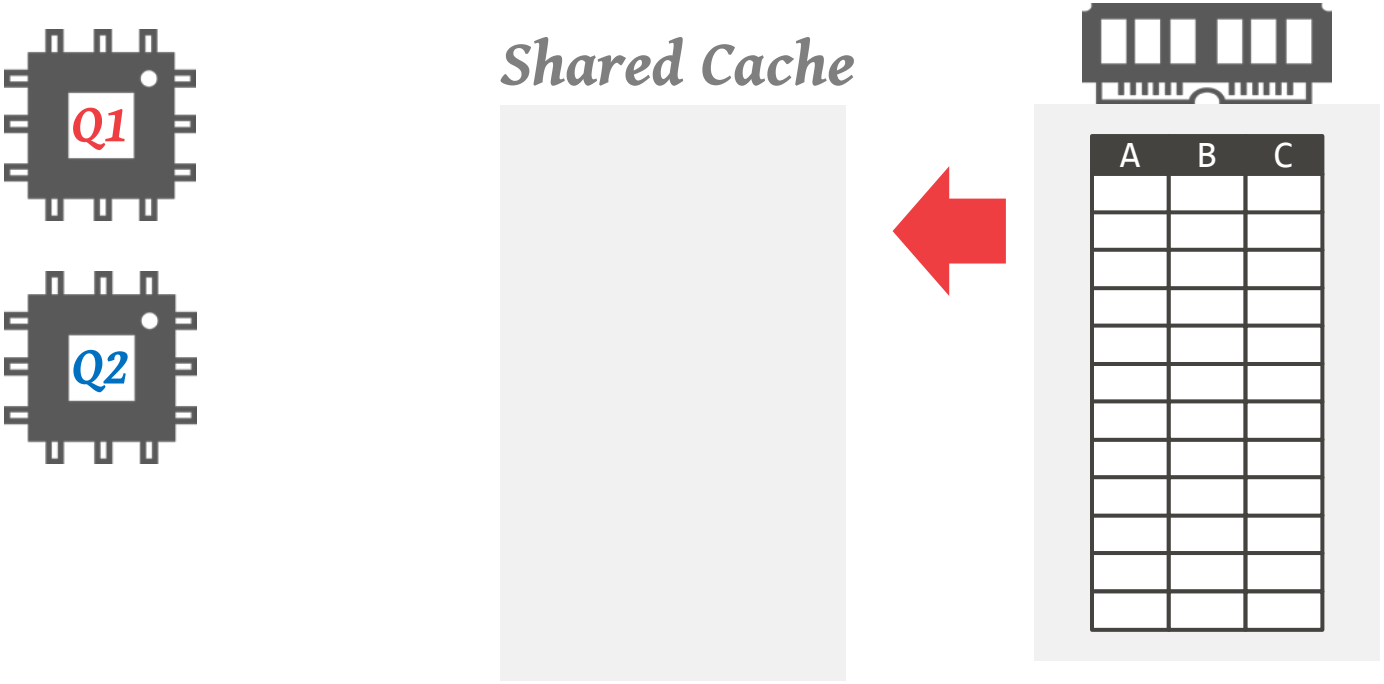




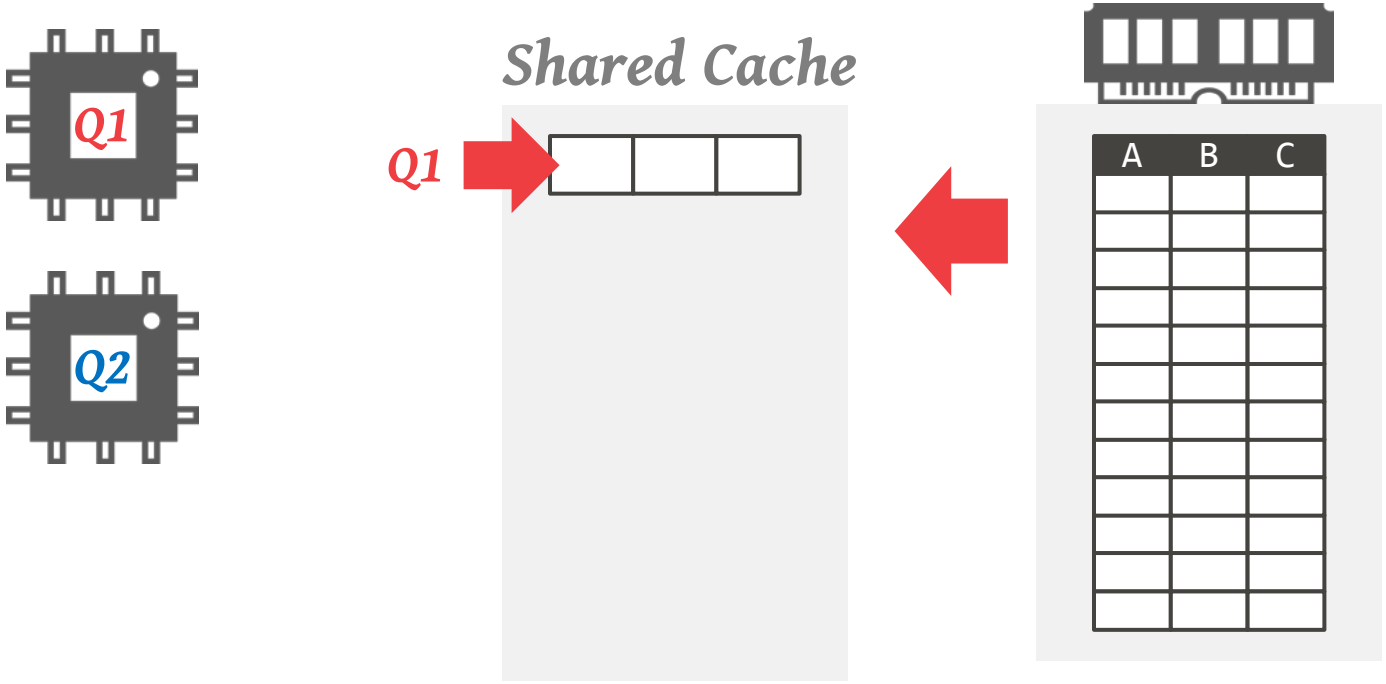
Shared Cache



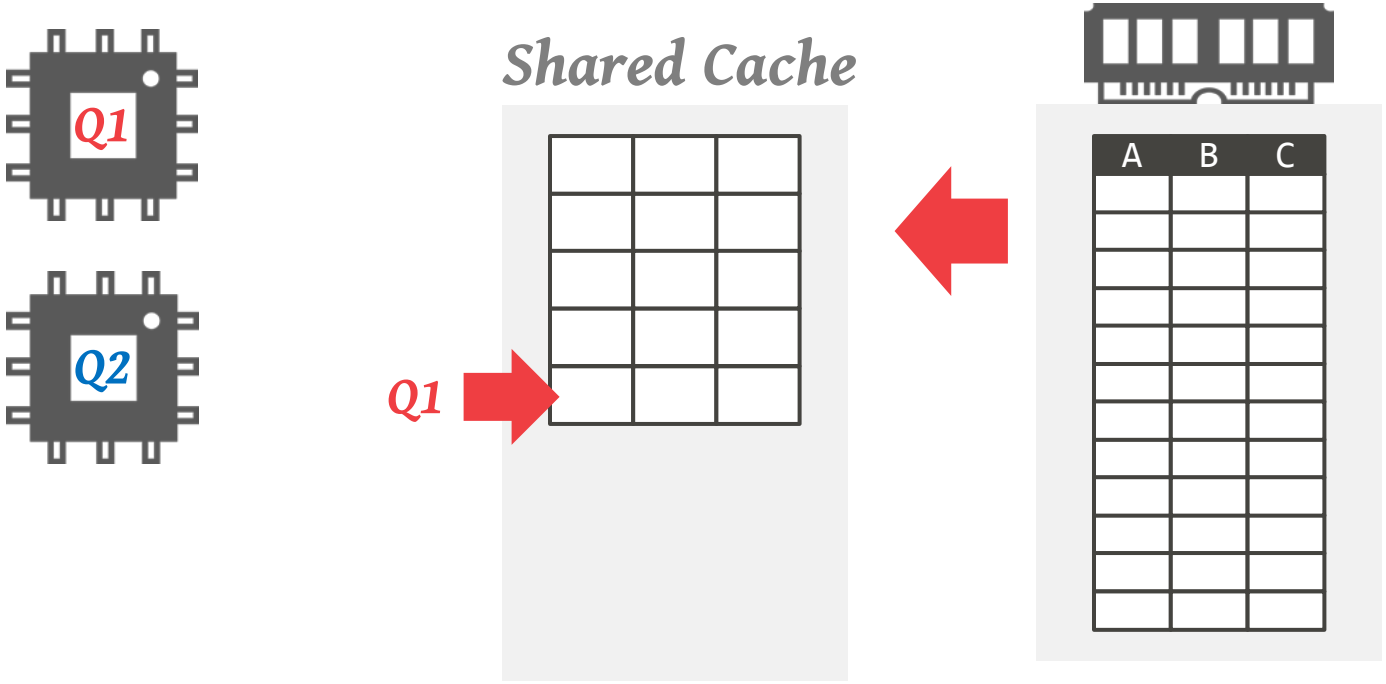
CONVOY PHENOMENON



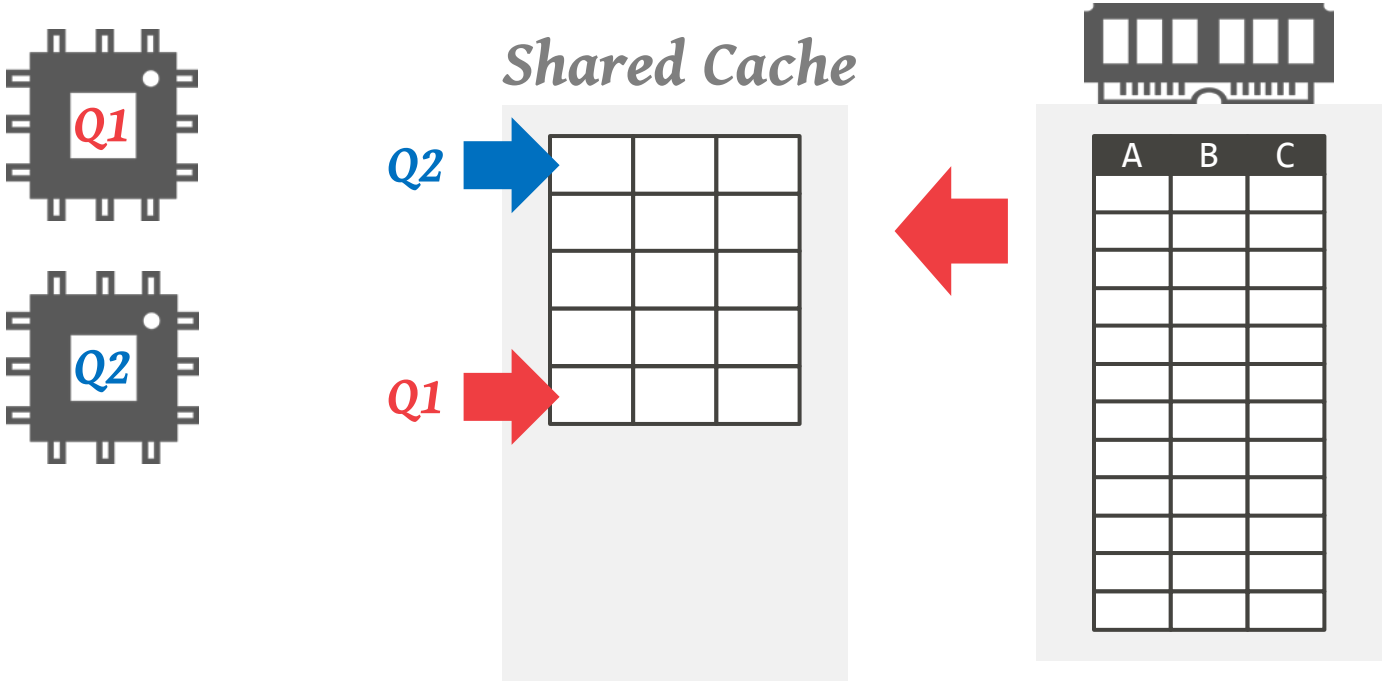
CONVOY PHENOMENON



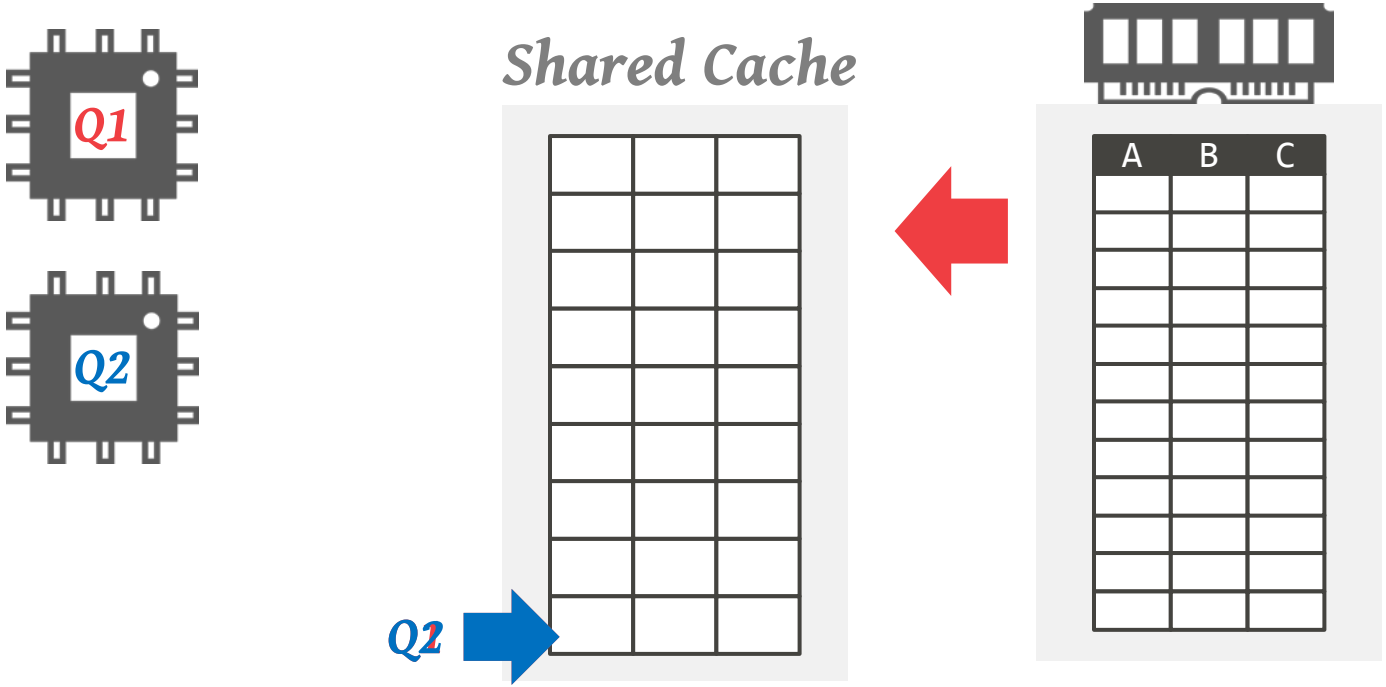
CONVOY PHENOMENON



CONVOY PHENOMENON



CONVOY PHENOMENON



IBM BLINK

In-memory query accelerator for IBM DB2 with support for explicit scan sharing.

→ Maintains a separate read-only DSM copy.
→ Dictionary compression.
→ Similar to Oracle's fractured mirrors.

Newer version is called **IBM DB2 BLU**.

→ Can now manages the primary copy of data (DSM).
→ Similar to Microsoft's Hekaton/Apollo extensions.

MAIN-MEMORY SCAN SHARING FOR MULTI-CORE CPUS
VLDB 2008

FULL-SHARING SCANS

Each worker thread will execute a separate table scan operations.

A dedicated ***reader thread*** feeds blocks of tuples to the worker threads.

→ Load a block of tuples in the cache and share it among multiple queries.

→ All of the worker threads have to acknowledge they are finished with the current block before the reader can move on to the next one.

BATCHED-SHARING SCANS

If too many worker threads share a scan, then their ***working sets*** can overflow the cache.

Prevent thrashing by grouping together smaller queries into batches.

But it is hard to figure out what queries to combine in a batch at runtime.

RESULT ESTIMATION

In order to pick the right number of queries to batch together, the DBMS needs to estimate the intermediate result size of operators.

→ Same problem as with query optimization.

Classify queries based on operator selectivity

→ **Always Share:** low selectivity (<0.1%)

→ **Never Share:** high selectivity

→ **Could Share:** have to estimate working set size

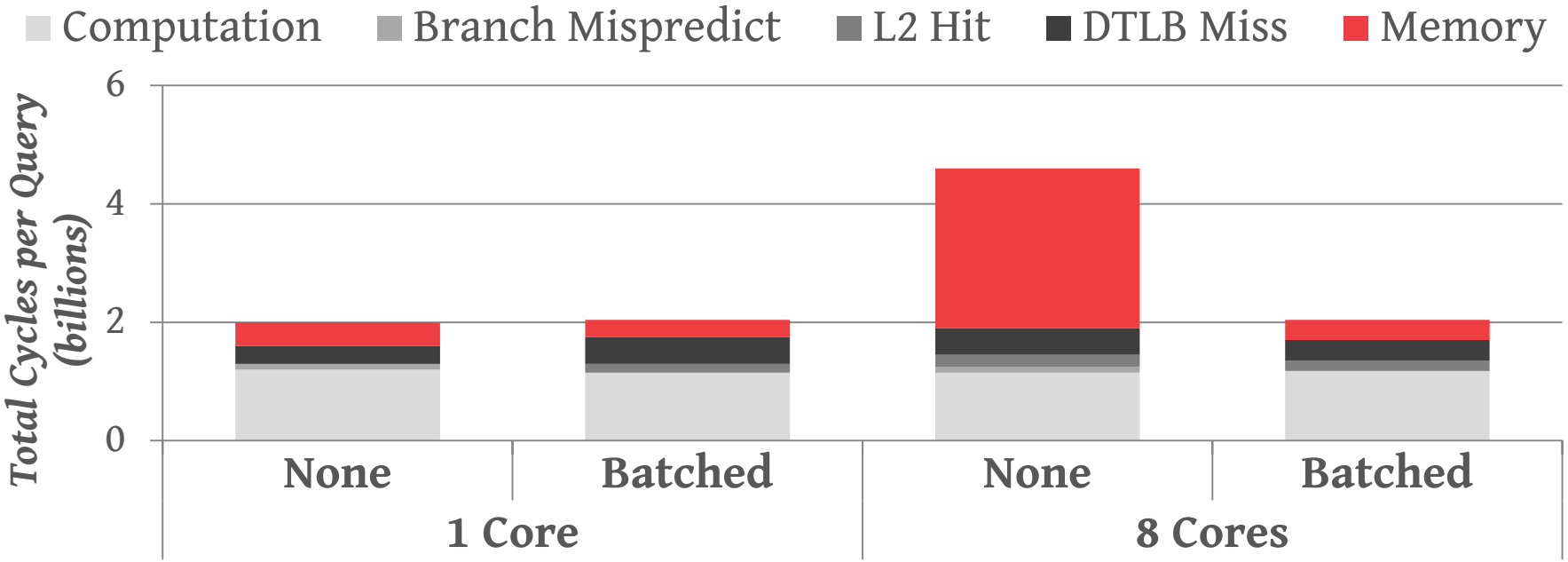
EVALUATION

Compare “naïve sharing” with batched-sharing using a OLAP/BI query from an IBM customer.

Industrial research paper so the performance numbers are all relative.

CPU CYCLES BREAKDOWN

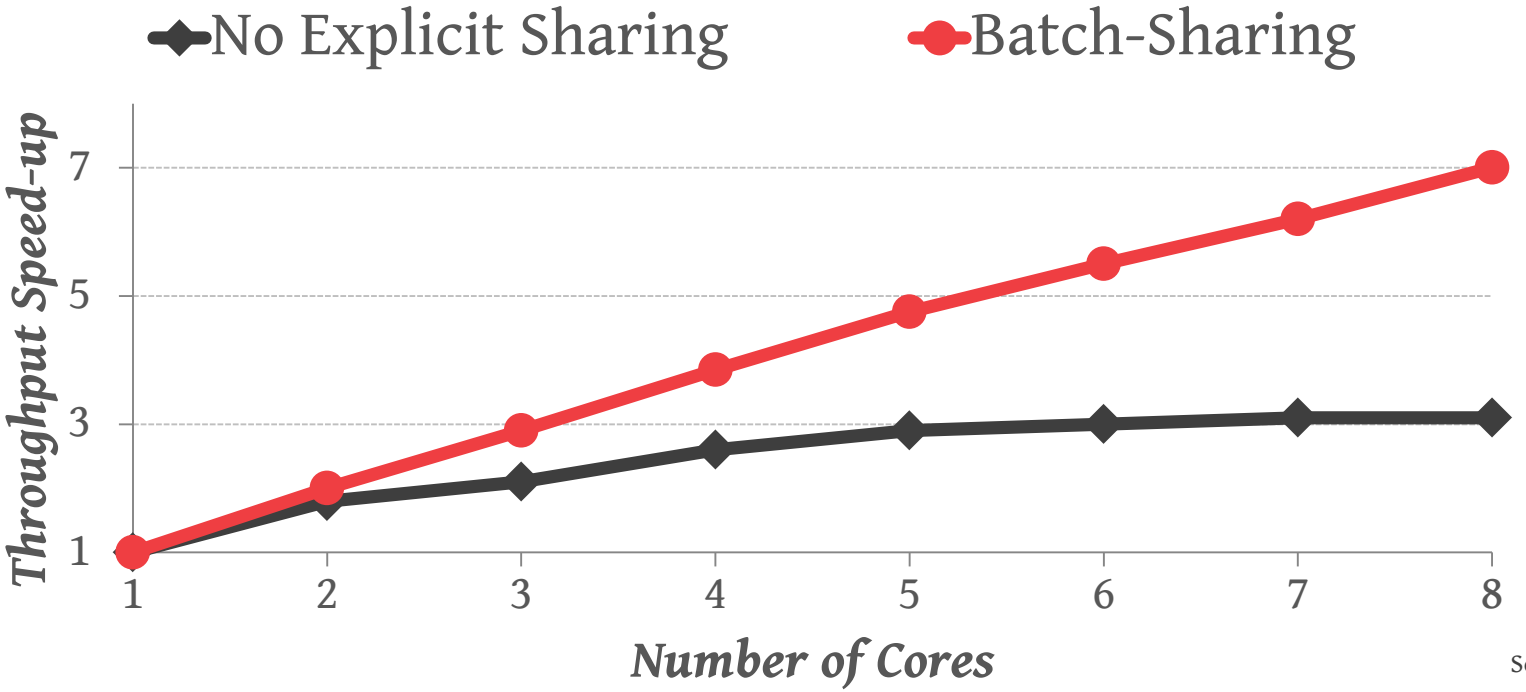
Dual-socket Intel Xeon @ 2.66 GHz
1 OLAP query per core



Source: Lin Qiao

MULTI-CORE SCALING

Dual-socket Intel Xeon @ 2.66 GHz
1 OLAP query per core



Source: Lin Qiao

ALTERNATIVE TECHNIQUES

Continuous Scanning

Work Sharing

CONTINUOUS SCANNING

Instead of trying to be “smart” about sharing scans across queries, just have a reader thread continuously scanning a partition of a table.

Queries attach/detach themselves to the readers as needed.
→ Think of this like a pub/sub interface.

CRESCANDO

Continuous scan in-memory DBMS with batched updates and queries.

→ Indexes are built on the fly.
→ All queries are guaranteed to take roughly the same amount of time no matter how many updates.

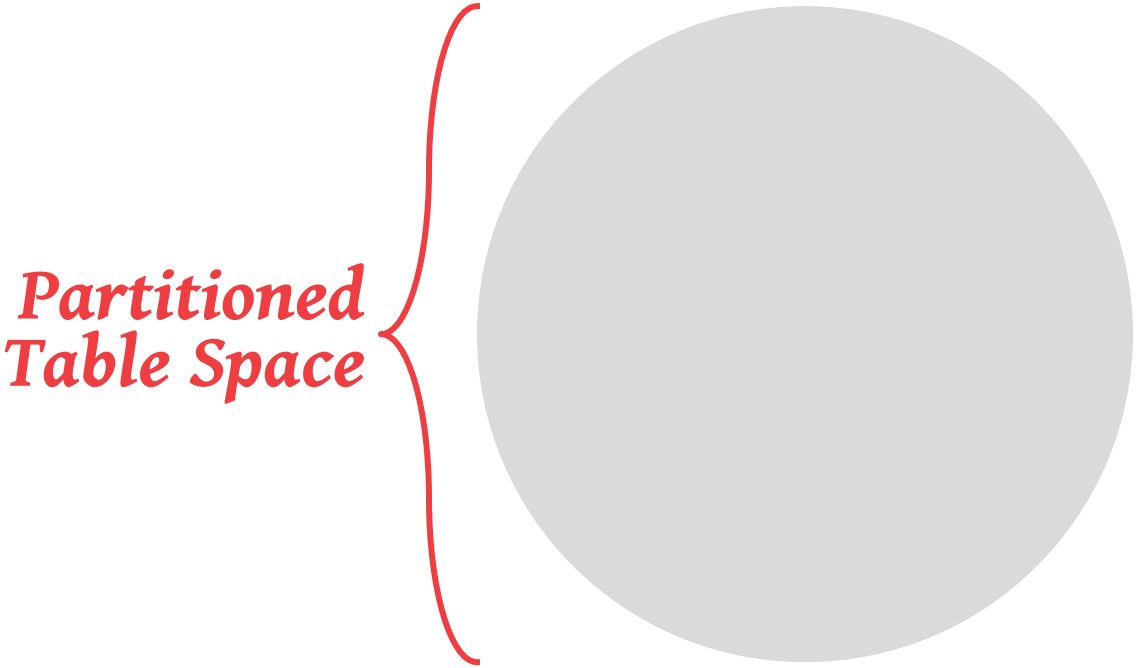
Use two different cursors to continuously scan and update the table at the same time.

→ **Write Cursor:** Apply batch updates to table.
→ **Read Cursor:** Runs behind writer.

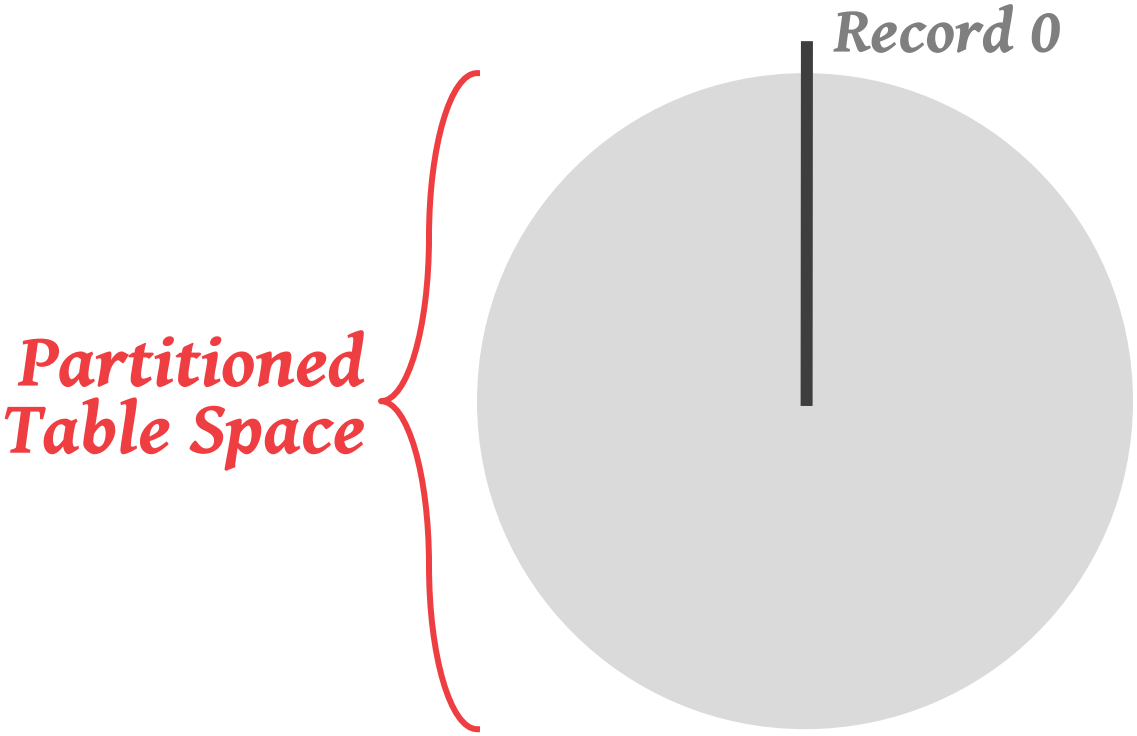
PREDICTABLE PERFORMANCE FOR UNPREDICTABLE WORKLOADS
VLDB 2009

CRESCANDO – CLOCK SCAN

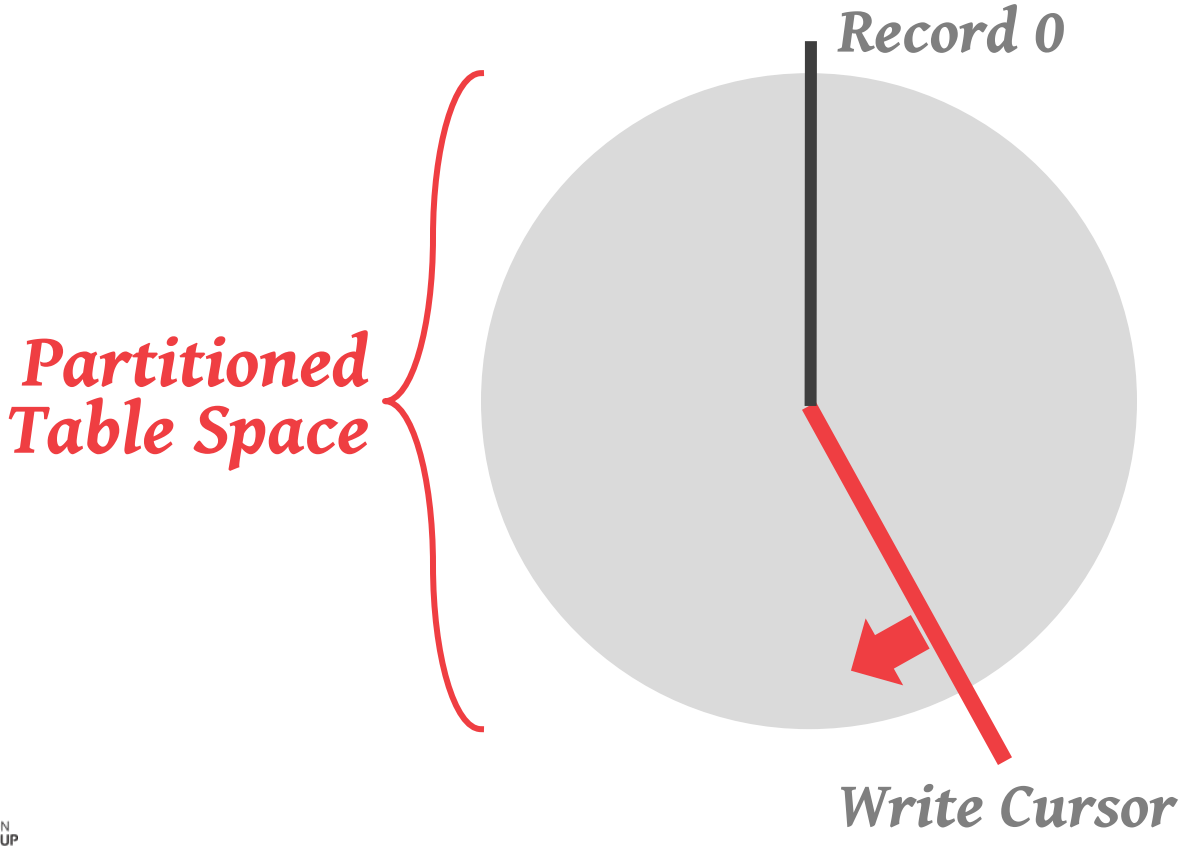
CRESCANDO – CLOCK SCAN



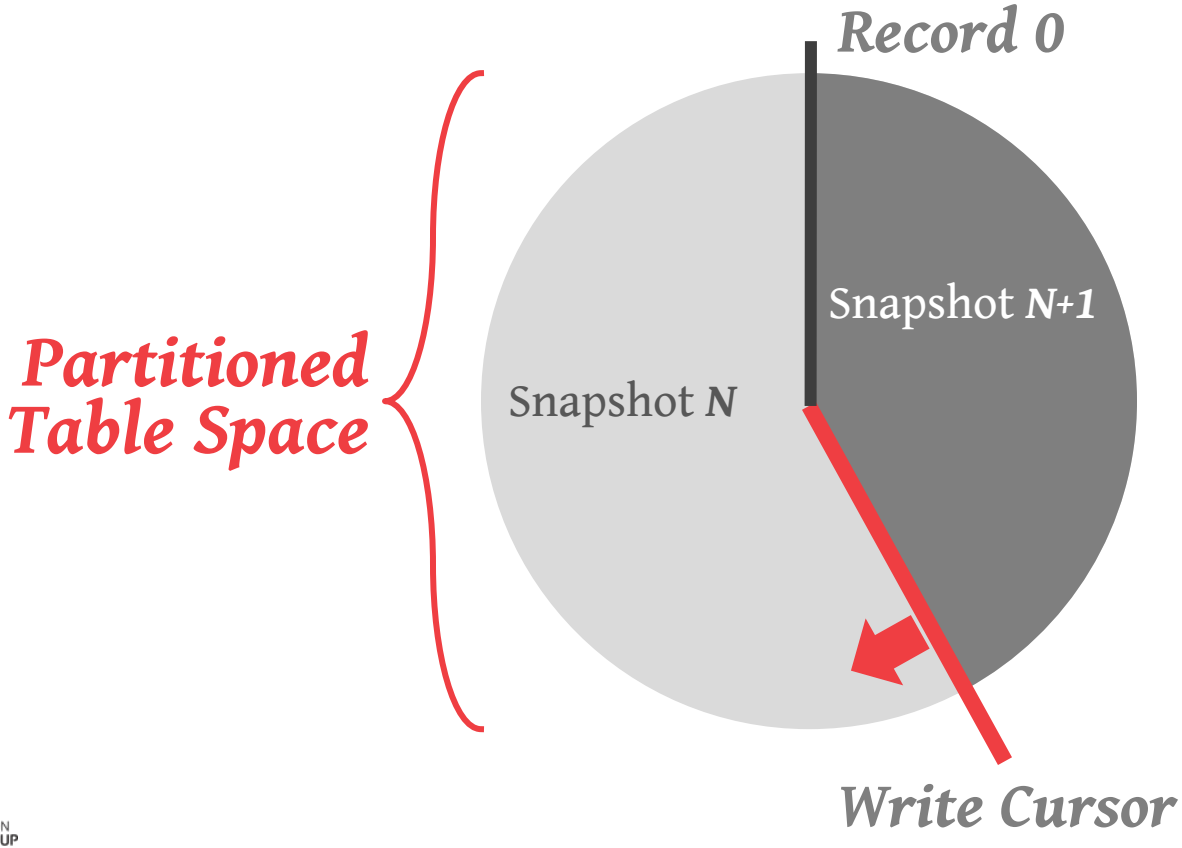
CRESCANDO – CLOCK SCAN



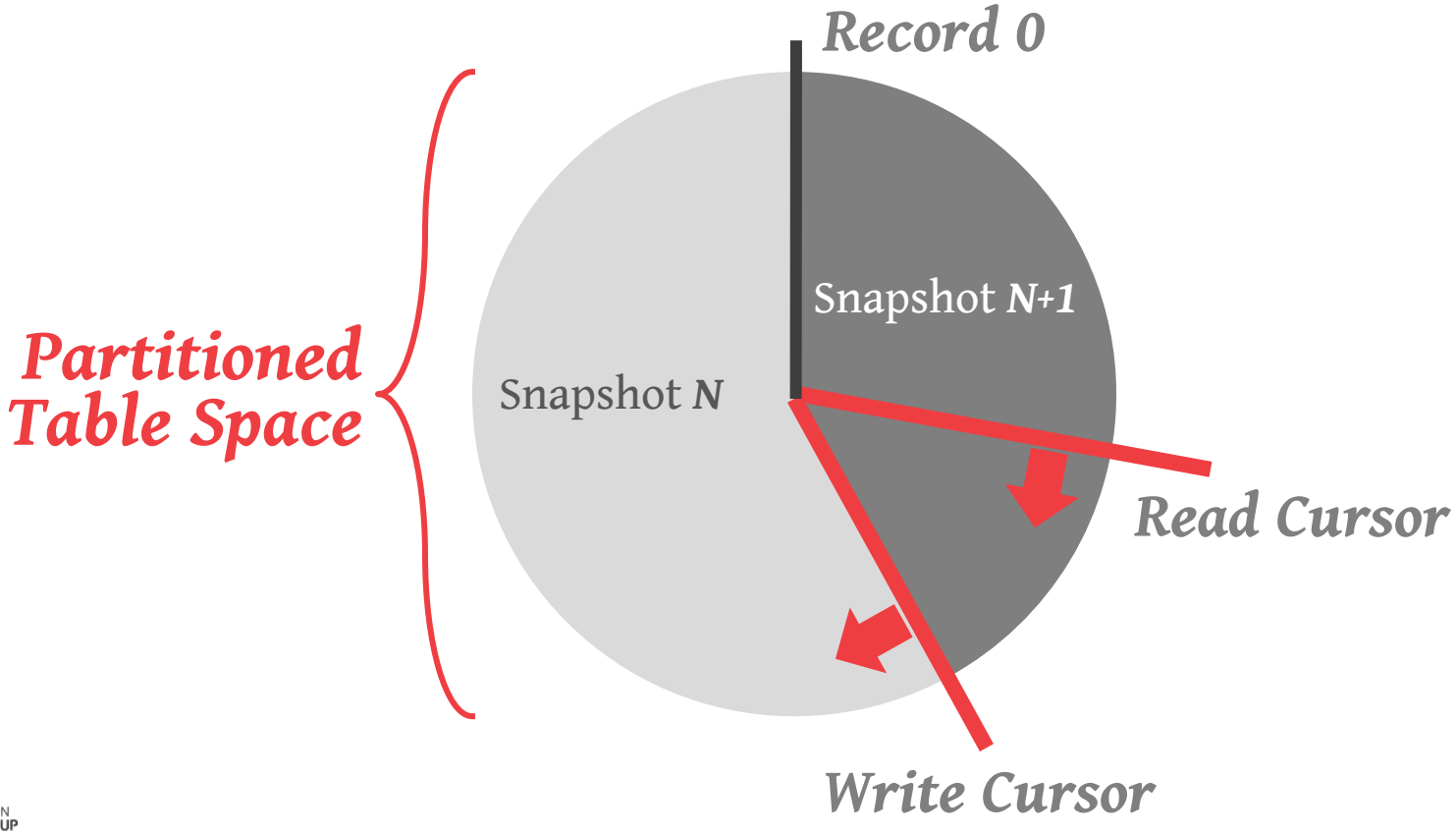
CRESCANDO – CLOCK SCAN



CRESCANDO – CLOCK SCAN



CRESCANDO – CLOCK SCAN



CRESCANDO IMPLEMENTATION

At the beginning of every scan, decide which indexes to build for current batch of queries.

→ Entire scan has to run once every 1s.
→ Use different data structures for different predicates.
→ Different batches may need different indexes.

DBMS architecture works well with NUMA.

→ Allows explicit control of threads read+write patterns
→ Scales up linearly with the number of cores
→ Partitioning the table space means that most of these indexes may fit in L2 cache.

Source: Donald Kossman

WORK SHARING

Worker threads can also share their intermediate results to avoid redundant computations.

It is up to the optimizer finds operators that can be shared across queries.

→ Multi-Query Optimization (MQO)
→ It may choose a plan that is bad for the individual query but better for the entire system.

SHARED-DB

Query planning and execution front-end for Crescando that generates a global plan for all pre-defined queries.

Queries are batched together. Predicates are combined into smallest disjunctive clause.

SHAREDDB: KILLING ONE THOUSAND QUERIES WITH ONE STONE
VLDB 2012

SHARED-DB – JOIN EXAMPLE

Q1

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.city = ?
   AND B.date = ?
```

Q2

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.name = ?
   AND B.price < ?
```

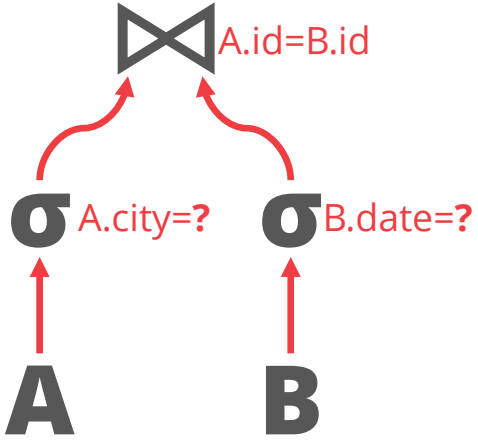
Q3

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.addr = ?
   AND B.date > ?
```

SHARED-DB – JOIN EXAMPLE

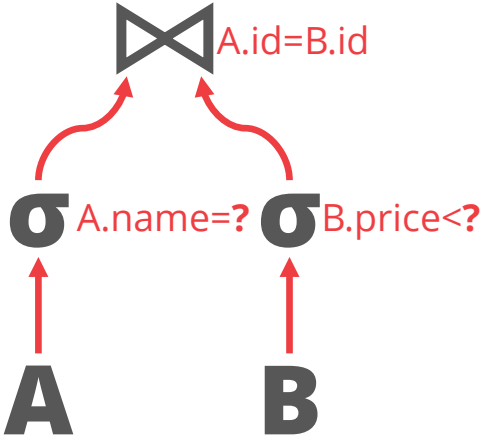
Q1

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.city = ?
   AND B.date = ?
```



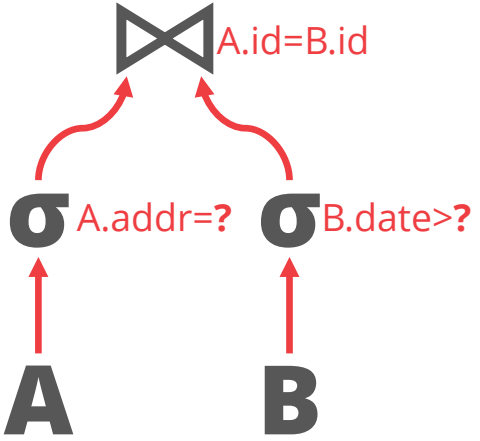
Q2

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.name = ?
   AND B.price < ?
```



Q3

```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.addr = ?
   AND B.date > ?
```

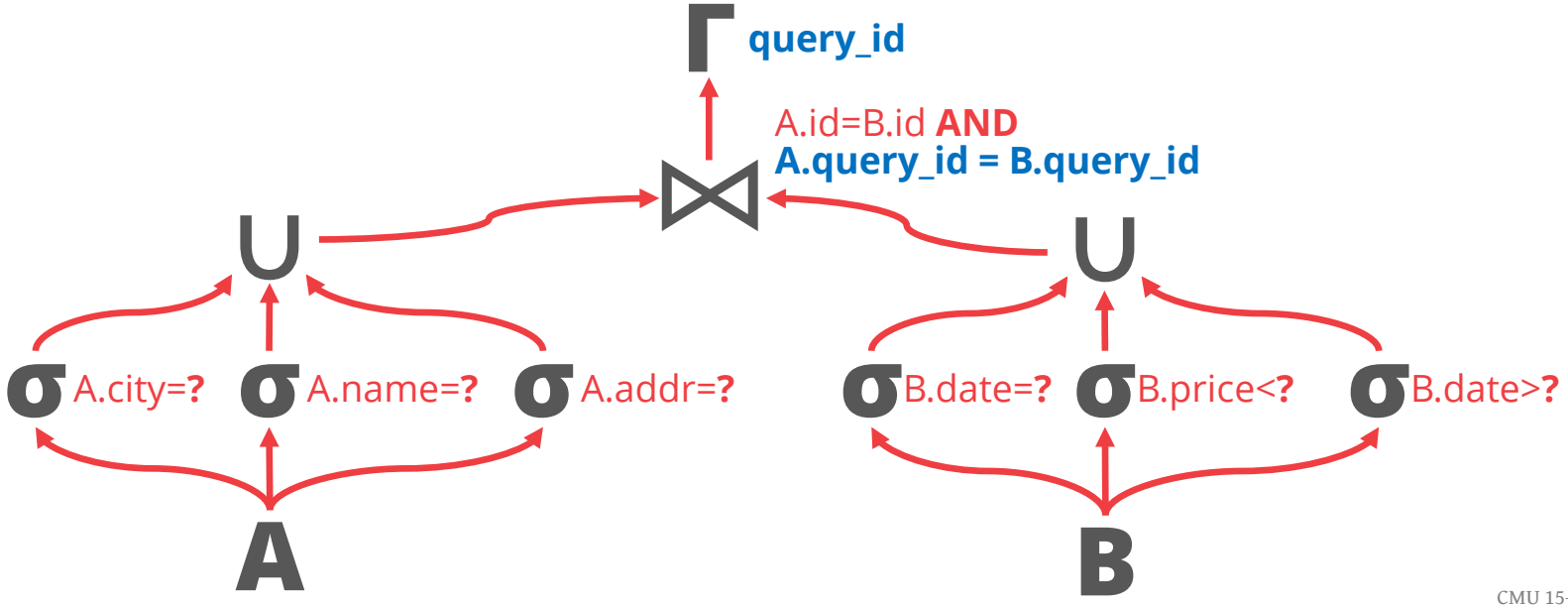


SHARED-DB – JOIN EXAMPLE

Q1
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.city = ?
   AND B.date = ?
```

Q2
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.name = ?
   AND B.price < ?
```

Q3
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.addr = ?
   AND B.date > ?
```

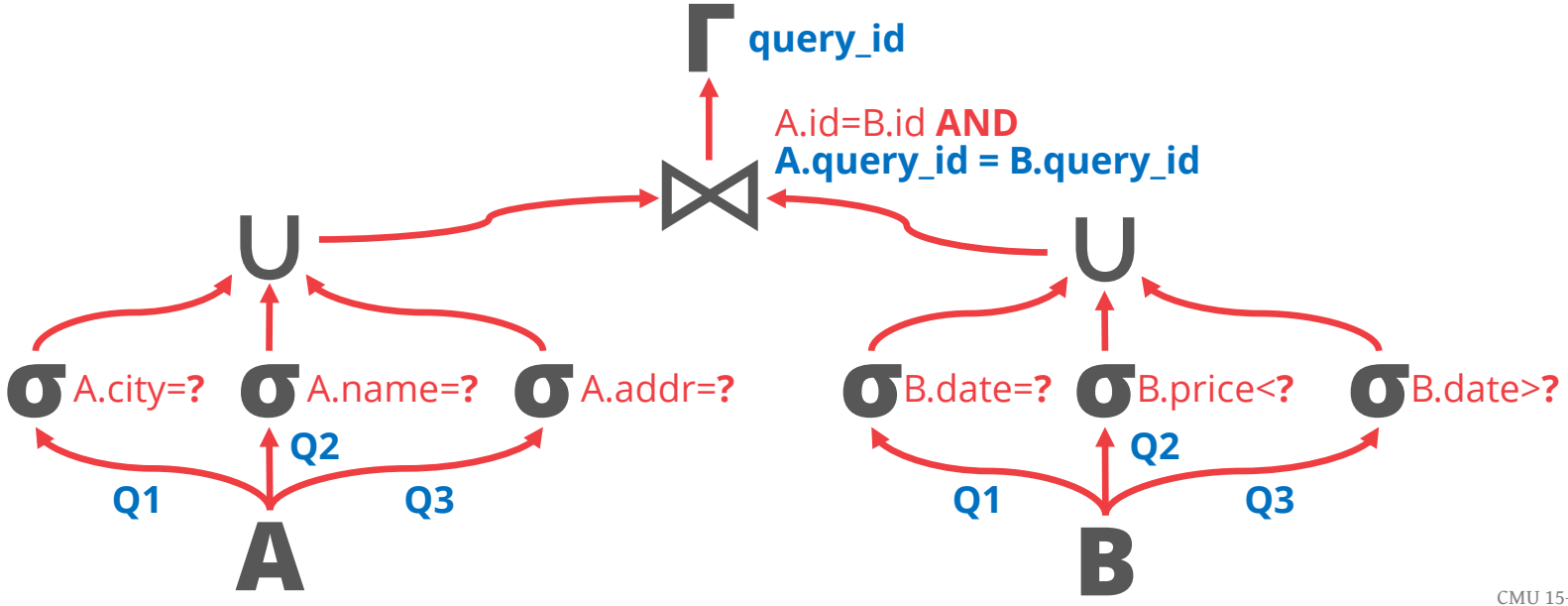


SHARED-DB – JOIN EXAMPLE

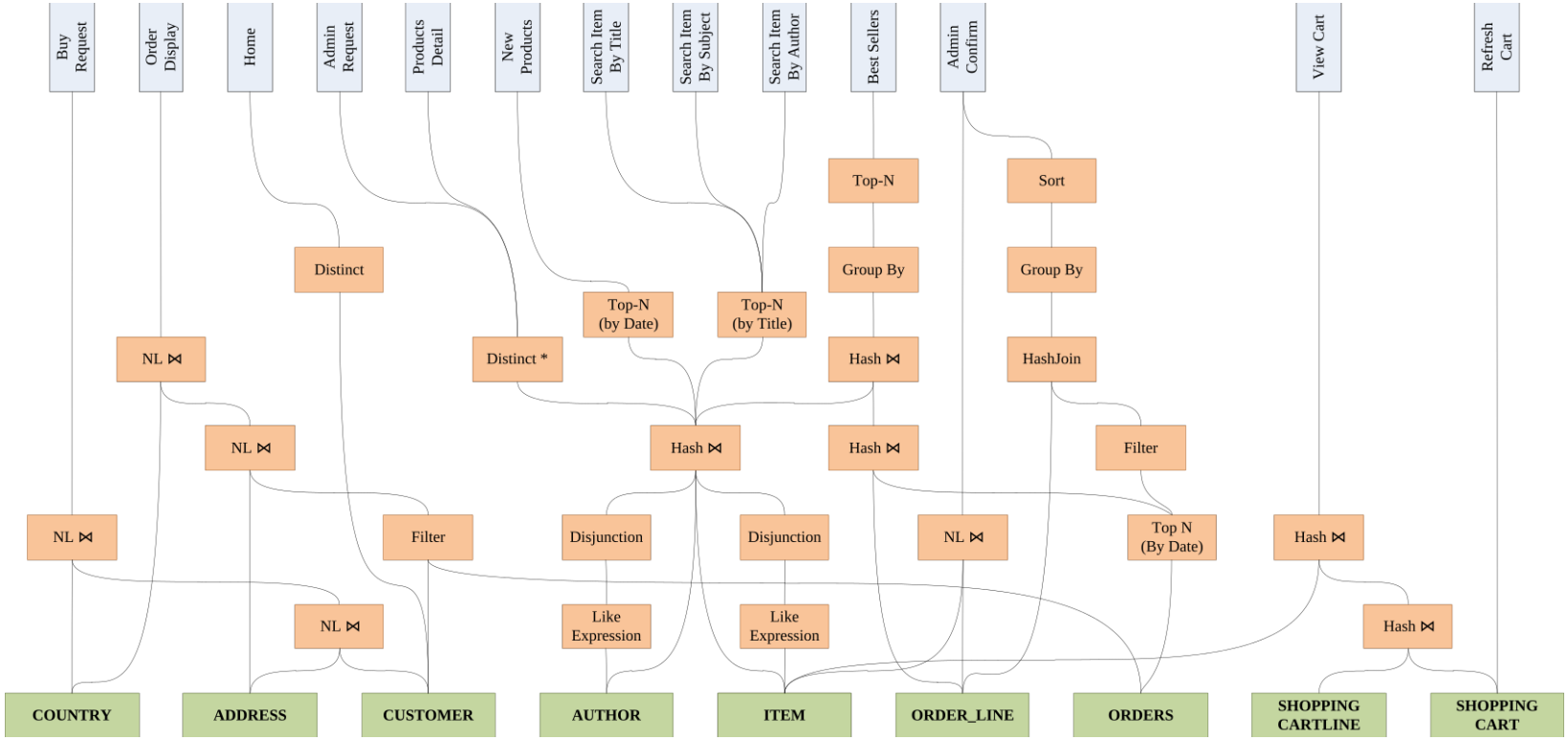
Q1
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.city = ?
   AND B.date = ?
```

Q2
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.name = ?
   AND B.price < ?
```

Q3
```
SELECT * FROM A, B
 WHERE A.id = B.id
   AND A.addr = ?
   AND B.date > ?
```

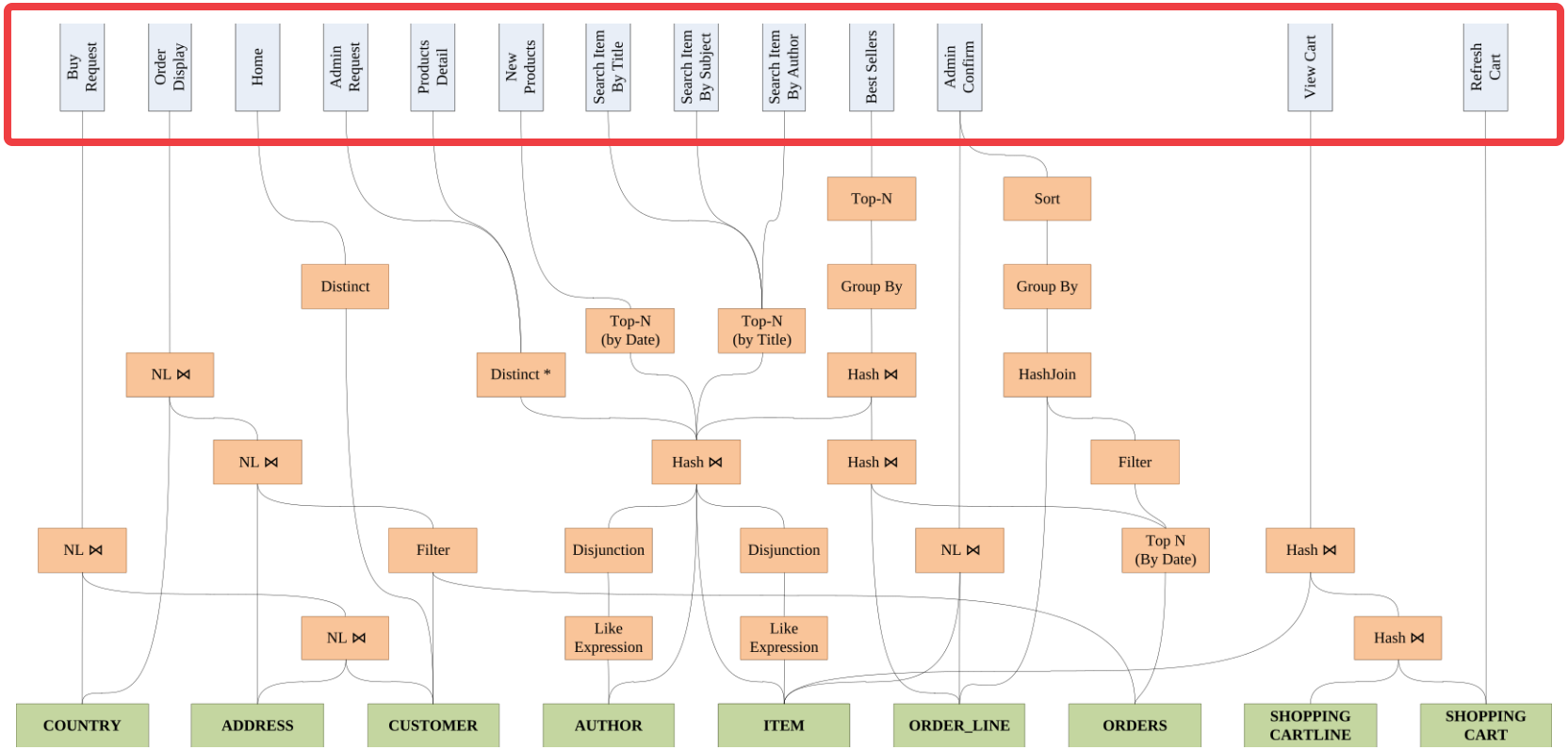


SHARED-DB – TPC-W QUERY PLAN



Source: Philipp Unterbrunner

SHARED-DB – TPC-W QUERY PLAN



Source: Philipp Unterbrunner

PARTING THOUGHTS

The benefits of scan sharing depend on the application's workload.

How to support scan sharing in an HTAP DBMS is an open research problem.

Continuous scanning is a cool idea but I think is only practical in a DBMS with batch updates.

NEXT CLASS

Vectorized Execution / SIMD

Prison Food Recipes