Bridging the Archipelago between Row-Stores and Column-Stores for Hybrid Workloads

PRESENTATION BY
Adit Mehta
Amara Nwigwe
Satha Kitirattragarn

Introduction - Problem

- Usage of Database Management Systems (DBMSs) concerns processing raw data into analytical insight. = “Hybrid Transactional-Analytical Processing (HTAP)”
- HTAP workloads are often segmented into two parts:
  - 1 DBMS optimized for On-Line Transactional Processing (OLTP) workload
  - 1 DBMS optimized for On-Line Analytical Processing (OLAP) workload

PROBLEMS

- Time - Inability for immediate data utilization at the application level
- Cost - Costly Administrative Overhead, estimated at ~50% of the total operating cost

Introduction - Solution?

- **ONE SOLUTION** → Single HTAP DBMS capable of both:
  - Transactional (OLTP) benefits: high throughput + low latency transactions performed
  - Analytical (OLAP) benefits: complex, long-running queries to operate on “hot” (transactional, more recent, more frequented) and “cold” (historical, less recent, less frequented) data

↳ **MAIN CHALLENGE**: Heavily-affected performance when performing transactional (OLTP) and analytical (OLAP) processes concurrently

↳ **CURRENT FIX**: implement separate query execution engines for different workloads

- OLTP execution engines process transactions for row-oriented data
- OLAP execution engines process analyses for column-oriented data

Introduction - Better Solution?

CHALLENGES WITH THE CURRENT FIX:

- Increase in undue DBMS complexity
- Worse performance with the required maintenance on additional overhead
- Only a limited range of queries possible

A BETTER SOLUTION?

The proposed “Flexible Storage Model”

Motivating Illustration

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(a) OLTP-oriented N-ary Storage Model (NSM)

ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(b) OLAP-oriented Decomposition Storage Model (DSM)

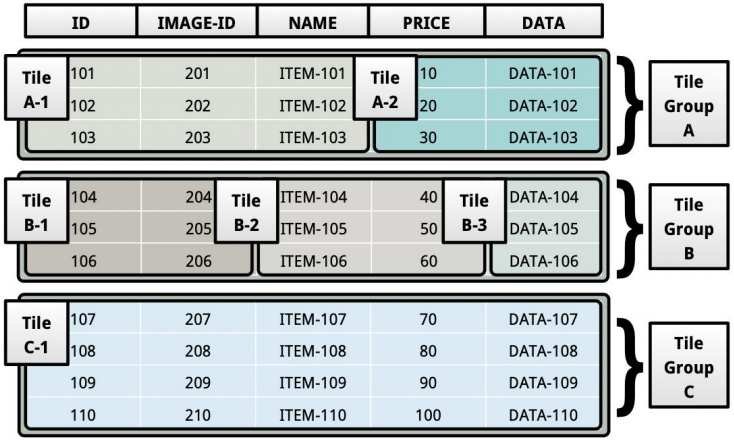
ID	IMAGE-ID	NAME	PRICE	DATA
101	201	ITEM-101	10	DATA-101
102	202	ITEM-102	20	DATA-102
103	203	ITEM-103	30	DATA-103
104	204	ITEM-104	40	DATA-104

(c) HTAP-oriented Flexible Storage Model (FSM)

Figure 1: Storage Models – Different table storage layouts work well for OLTP, OLAP, and HTAP workloads. The different colored regions indicate the data that the DBMS stores contiguously.

TILE BASED ARCHITECTURE

TILE TUPLES / PHYSICAL TILES / TILE GROUPS



WHAT ARE THEY?

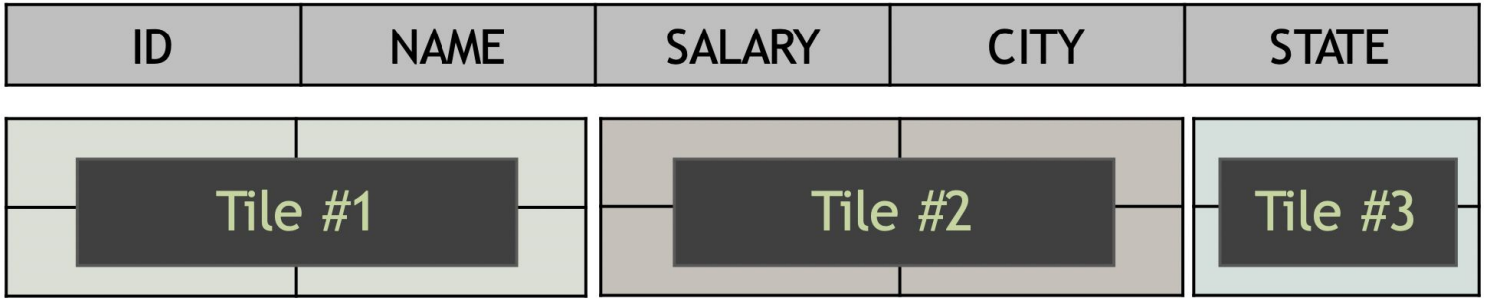
A set of tile tuples form a physical tile. The collection of physical tiles is a *tile group*. The physical tiles belonging to a tile group contain the same number of tile tuples.

TILE BASED ARCHITECTURE

LOGICAL TILES

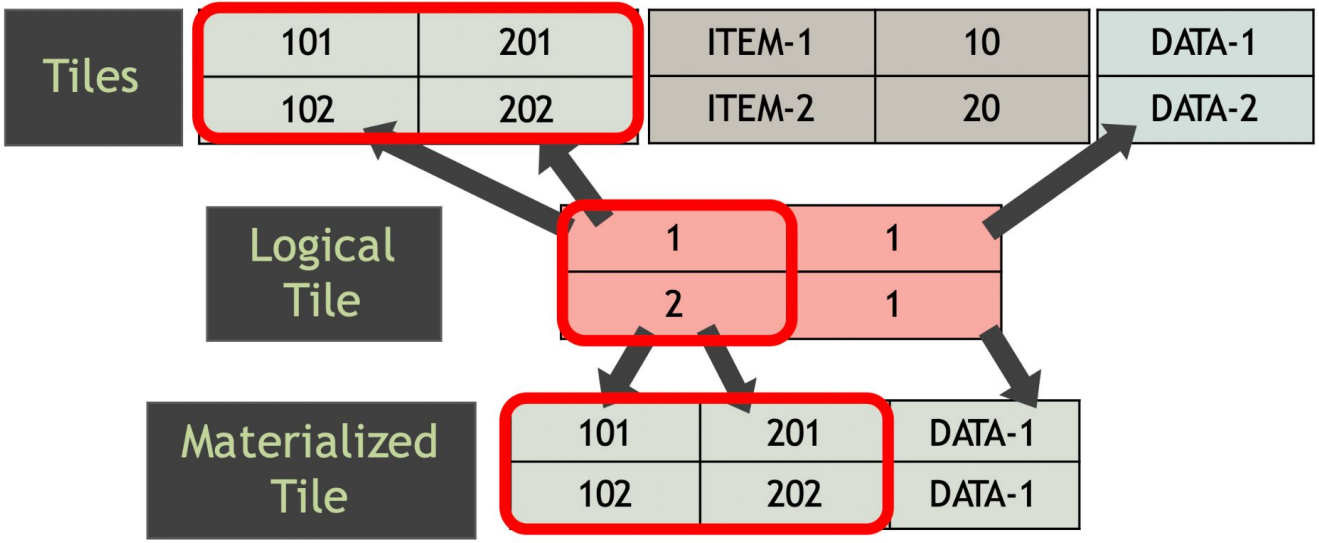
WHAT ARE THEY?

These are representative of values spread across a collection of physical tiles from one or more tables. Abstracts hybrid layout specifics without affecting performance.



TILE BASED ARCHITECTURE

LOGICAL TILES



TILE BASED ARCHITECTURE

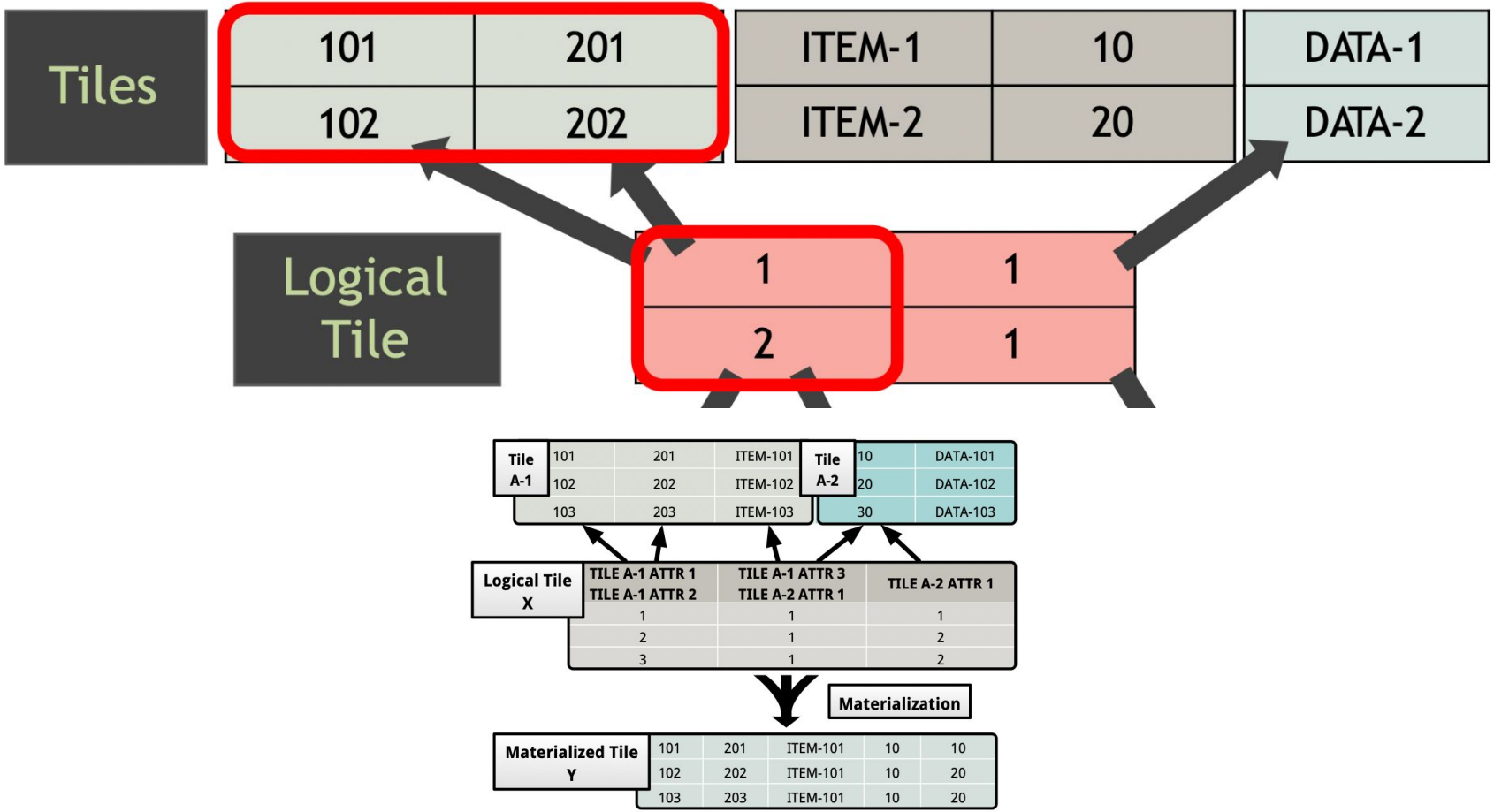
LOGICAL TILE ALGEBRA

Bridge Operators

- ➔ Connect logical and physical tiles (using table accessing methods)
  - ◆ Sequential scan
    - Makes a logical tile for every tile group in the table
  - ◆ Index scan
    - Constructs one or more logical tiles that contain matching tuples (that match predicate using index)

Metadata Operators

- ➔ Modifies data of operators, not the data it represents
  - ◆ Projection
    - Modifies list of attributes to remove those not needed in the final result
  - ◆ Selection
    - Modifies metadata, marks rows that don’t satisfy the predicate as not part of the physical tile


Tiles
101
201
102
202
ITEM-1
10
ITEM-2
20
DATA-1
DATA-2
Logical Tile
1
1
2
1
Tile A-1
101 201 ITEM-101
102 202 ITEM-102
103 203 ITEM-103
Tile A-2
10 DATA-101
20 DATA-102
30 DATA-103
Logical Tile X
TILE A-1 ATTR 1
TILE A-1 ATTR 2
TILE A-1 ATTR 3
TILE A-2 ATTR 1
TILE A-2 ATTR 1
1 1 1
2 1 2
3 1 2
Materialization
Materialized Tile Y
101 201 ITEM-101 10 10
102 202 ITEM-101 10 20
103 203 ITEM-101 10 20

TILE BASED ARCHITECTURE

LOGICAL TILE ALGEBRA

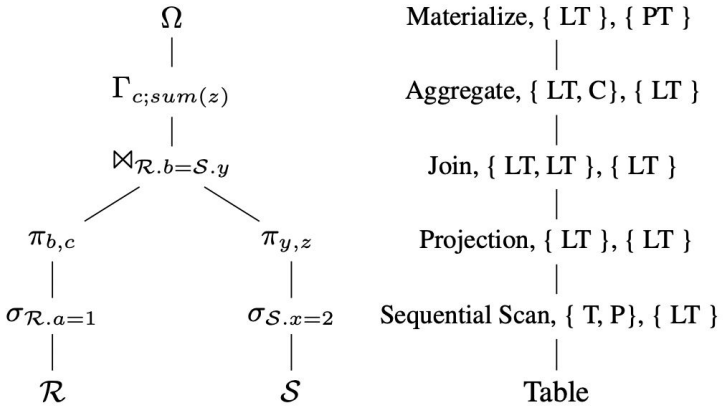
Mutators

- ➔ Modify data stored in table
  - ◆ Insert
  - ◆ Delete
  - ◆ Update

Pipeline Breakers

- ➔ Consume the logical tiles produced by their children in the plan tree
- ➔ Block execution of upper level operators while they wait for children's output

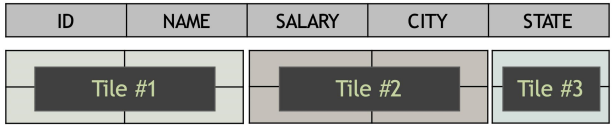
```
SELECT R.c, SUM(S.z)
FROM R JOIN S ON R.b = S.y
WHERE R.a = 1 AND S.x = 2
GROUP BY R.c;
```



TILE BASED ARCHITECTURE

LOGICAL TILE

BENEFITS?



- complexity is reduced
  - due to abstraction
  - operators need not be specialized for all storage layouts
- reduces interpretation overhead through vectorized processing
- flexible materialization (can happen at any time)
- complex intermediate query execution is easier to process

CONCURRENCY CONTROL

LOGICAL TILE

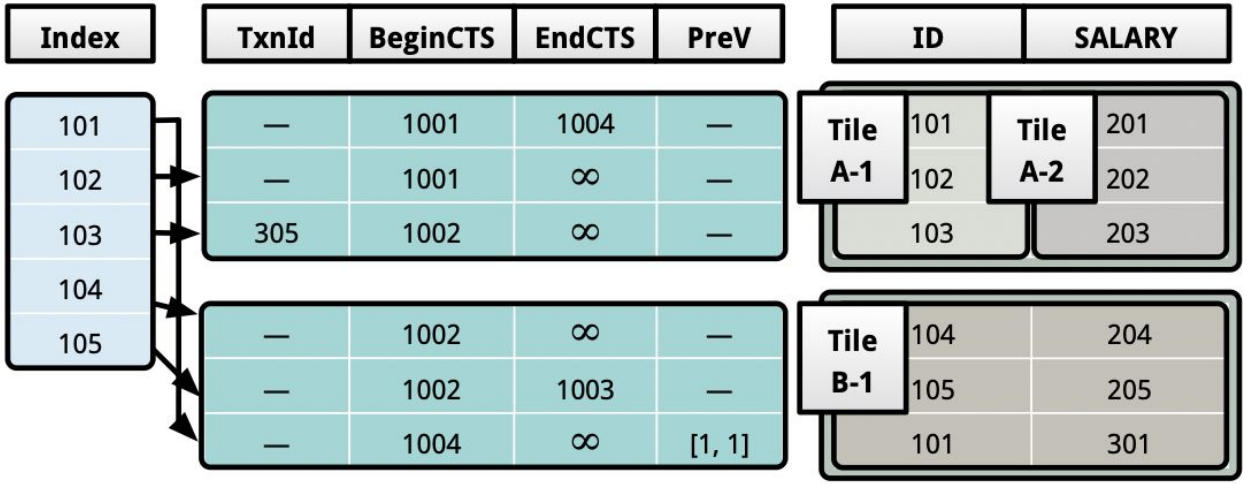
WHY?

Multi-version Concurrency control is implemented so OLAP (on-line analytical processing) queries do not see the effects of transactions that start after they begin and the readers should not block on writers.

- **TxnId:** A placeholder for the identifier of the transaction that currently holds a latch on the tuple.
- **BeginCTS:** The commit timestamp from which the tuple becomes visible.
- **EndCTS:** The commit timestamp after which the tuple ceases to be visible.
- **PreV:** Reference to the previous version, if any, of the tuple.

CONCURRENCY CONTROL

LOGICAL TILES

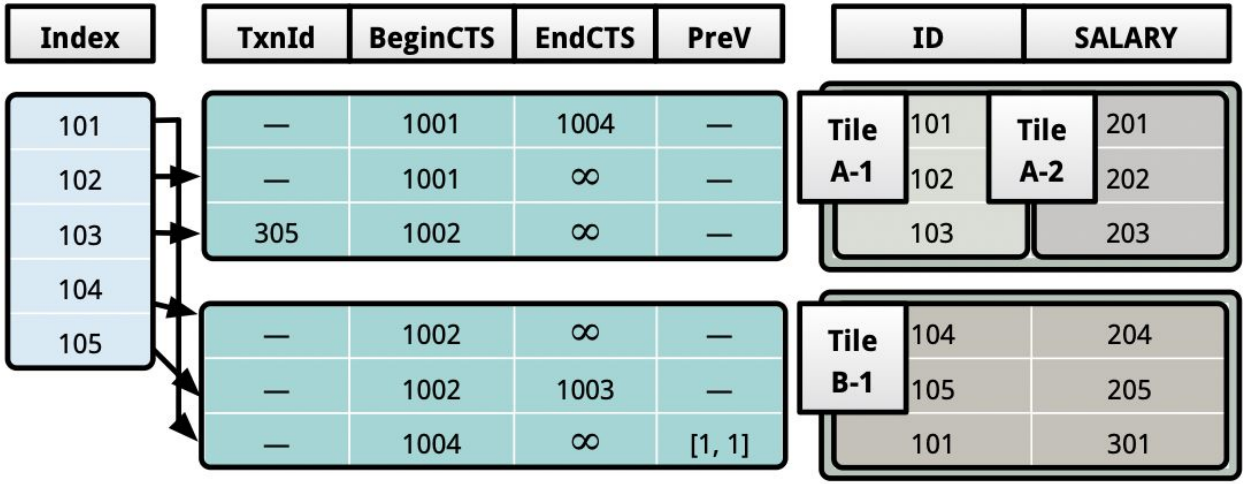


Mutators

Insert, delete, update

CONCURRENCY CONTROL

LOGICAL TILES



Bridge Operators

sequential scan, index scan

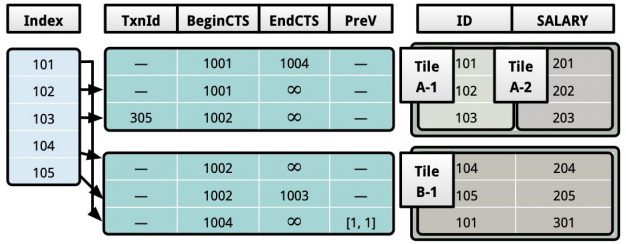
CONCURRENCY CONTROL

LOGICAL TILES & INDEXES

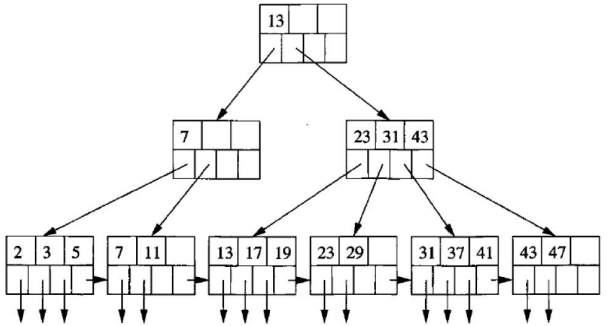
An order-preserving in-memory index is used (e.g., B+tree) for primary and secondary indexes.

The key value is a logical location of the latest version of a tuple.

An operator might encounter a version of the tuple that is not visible to its current transaction. When this occurs, it uses the PreV field to traverse the version chain to find the newest version of the tuple that is visible to the transaction.



EXAMPLE: B+-TREE



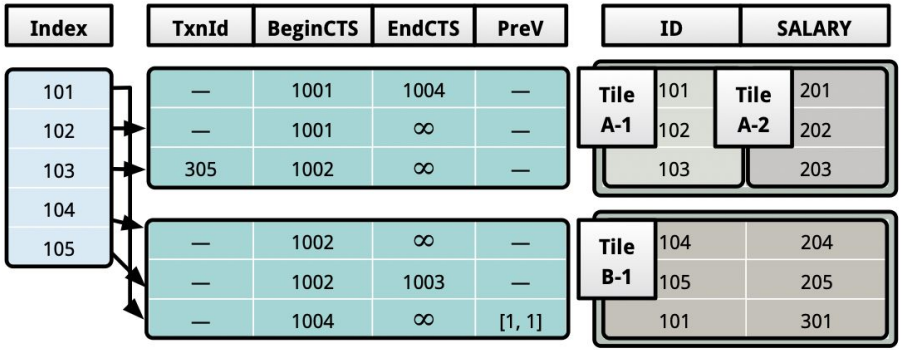
CONCURRENCY CONTROL

RECOVERY

The DBMS rebuilds all of the tables' indexes during recovery to ensure that they are consistent with the database.

Recovery employs a variant of the canonical ARIES recovery protocol that is adapted for in-memory DBMSs.

Uncommitted changes made at the time of a crash are not propagated to the database.



LAYOUT REORGANIZATION

CLUSTERING & GREEDY ALGORITHM

Clustering and Greedy Algorithm are phases of the two part Vertical Partitioning Algorithm.

ON-LINE QUERY MONITORING

The goal is to determine which attributes should be stored in the same physical tile in the new tile group layout.

Done through the monitor selecting info using the SELECT and WHERE clauses. Statistics taken from a random subset of queries to reduce overhead & allows for no bias from frequently observed transactions.

Algorithm 1 Vertical Partitioning Algorithm

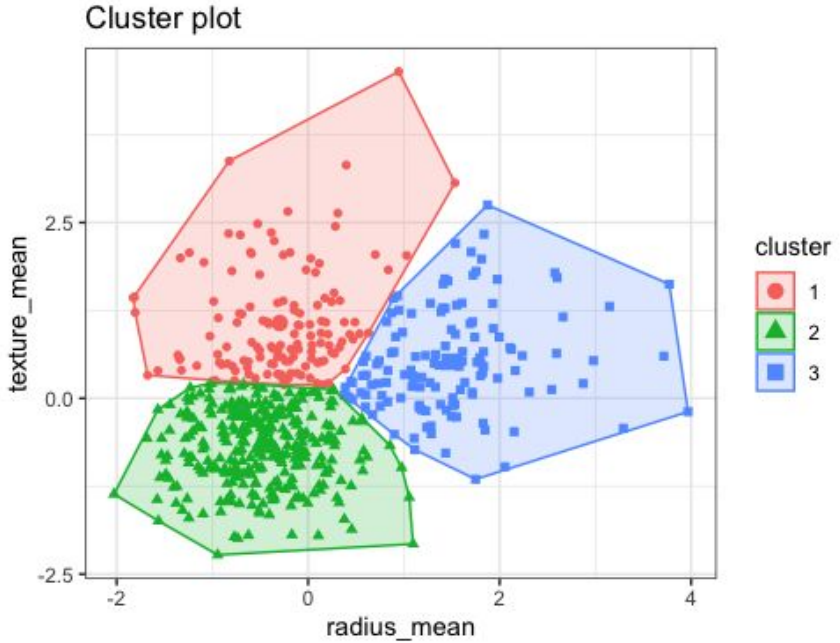
Require: recent queries Q, table T, number of representative queries k
function UPDATE-LAYOUT(Q, T, k)
 # Stage I : Clustering algorithm
for all queries q appearing in Q **do**
for all representative queries r_j associated with T **do**
if r_j is closest to q **then**
$r_j \leftarrow r_j + \mathrm{w} \times (\mathrm{q} - r_j)$
end if
end for
end for
 # Stage II : Greedy algorithm
Generate layout for T using r
end function

LAYOUT REORGANIZATION

CLUSTERING & GREEDY ALGORITHM

For each table T in the database, the DBMS maintains statistics about the recent queries Q that accessed it. For each q ∈ Q, the DBMS extracts its metadata, such as the attributes it accessed.

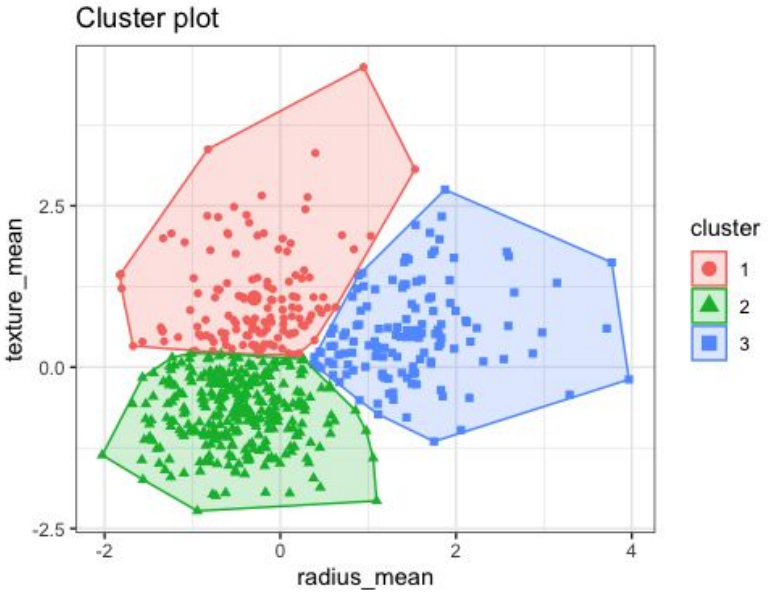
Important attributes determined by k-means clustering and the storage layout of the table for these queries is optimized if the DBMS can recognize these attributes.



LAYOUT REORGANIZATION

CLUSTERING & GREEDY ALGORITHM

- For each query q, the clustering algorithm observes the referenced attributes, and assigns it to the jth cluster, whose mean representative query rj is the most similar to q.
- The distance metric between two queries is defined as the number of attributes which are accessed by one and exactly one of the two queries divided by the number of attributes in T .
- Queries that access a lot of common attributes of T , therefore, tend to belong to the same cluster. After assigning the query to a cluster, it updates rj to reflect the inclusion of the query q.



LAYOUT REORGANIZATION

CLUSTERING & GREEDY ALGORITHM

This algorithm iterates over the representative queries in the descending order based on the weight of their associated clusters. For each cluster, the algorithm groups the attributes accessed by that cluster's representative query together into a tile. It continues this process until it assigns each attribute in the table to some tile. In this manner, the DBMS periodically computes a layout for the table using the recent query statistics.

Algorithm 1 Vertical Partitioning Algorithm

Require: recent queries Q, table T, number of representative queries k
function UPDATE-LAYOUT(Q, T, k)
 # Stage I : Clustering algorithm
for all queries q appearing in Q **do**
for all representative queries r_j associated with T **do**
if r_j is closest to q **then**
$r_j \leftarrow r_j + \mathrm{w} \times (\mathrm{q} - r_j)$
end if
end for
end for
 # Stage II : Greedy algorithm
Generate layout for T using r
end function

LAYOUT REORGANIZATION

DATA LAYOUT REORGANIZATION

An incremental approach.

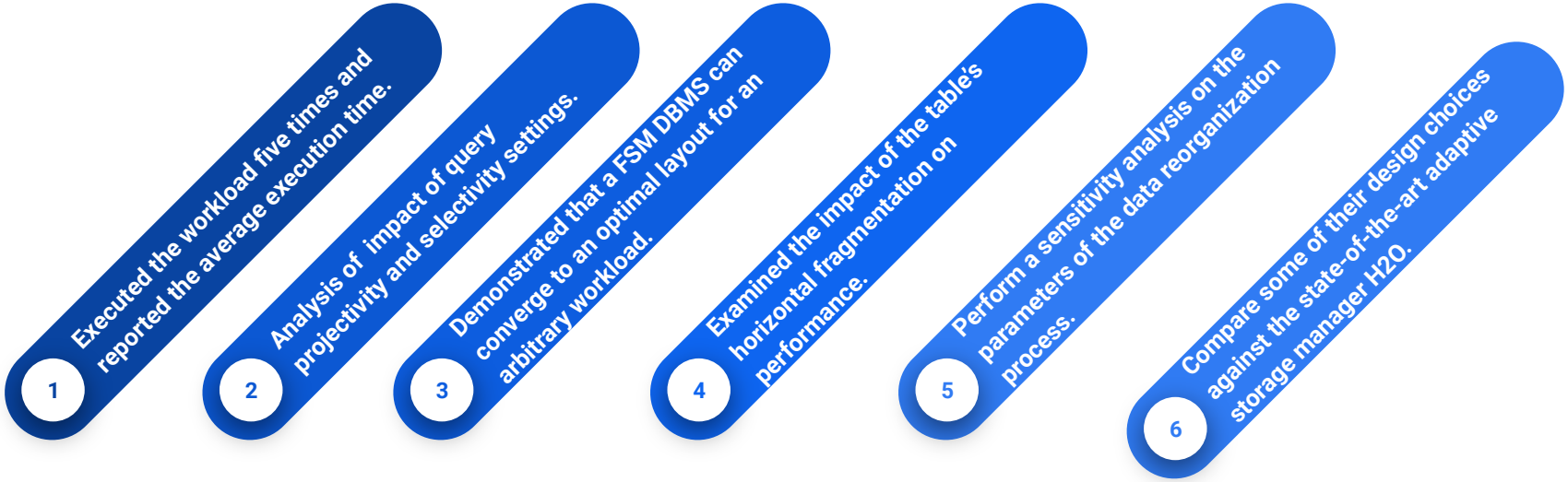
Data is copied to a new layout, then a newly constructed tile group is swapped into the table. Does not apply to transactional (hot) data, but to historical (cold) data. Starts out as a tuple centric layout (similar to row store), then into an OLAP (on-line analytical processing, similar to column-store) optimized layout from the greedy algorithm.

Algorithm 1 Vertical Partitioning Algorithm

Require: recent queries Q, table T, number of representative queries k
function UPDATE-LAYOUT(Q, T, k)
 # Stage I : Clustering algorithm
for all queries q appearing in Q **do**
for all representative queries r_j associated with T **do**
if r_j is closest to q **then**
$r_j \leftarrow r_j + \mathrm{w} \times (\mathrm{q} - r_j)$
end if
end for
end for
 # Stage II : Greedy algorithm
Generate layout for T using r
end function

Does this paper support it's claims?

YES!!



The ADAPT Benchmark

Q_1: **INSERT INTO** R **VALUES** $(a_0, a_1, \ldots, a_p)$

Q_2: **SELECT** $a_1, a_2, \ldots, a_k$ **FROM** R **WHERE** $a_0 < \delta$

Q_3: **SELECT** MAX$(a_1), \ldots,$MAX(a_k) **FROM** R **WHERE** $a_0 < \delta$

Q_4: **SELECT** $a_1 + a_2 + \ldots + a_k$ **FROM** R **WHERE** $a_0 < \delta$

Q_5: **SELECT** X.$a_1, \ldots,$X.a_k, Y.$a_1, \ldots,$ Y.a_k
FROM R **AS** X, R **AS** Y **WHERE** X.a_i < Y.a_j

Analysis of Storage Models under Different Projectivity

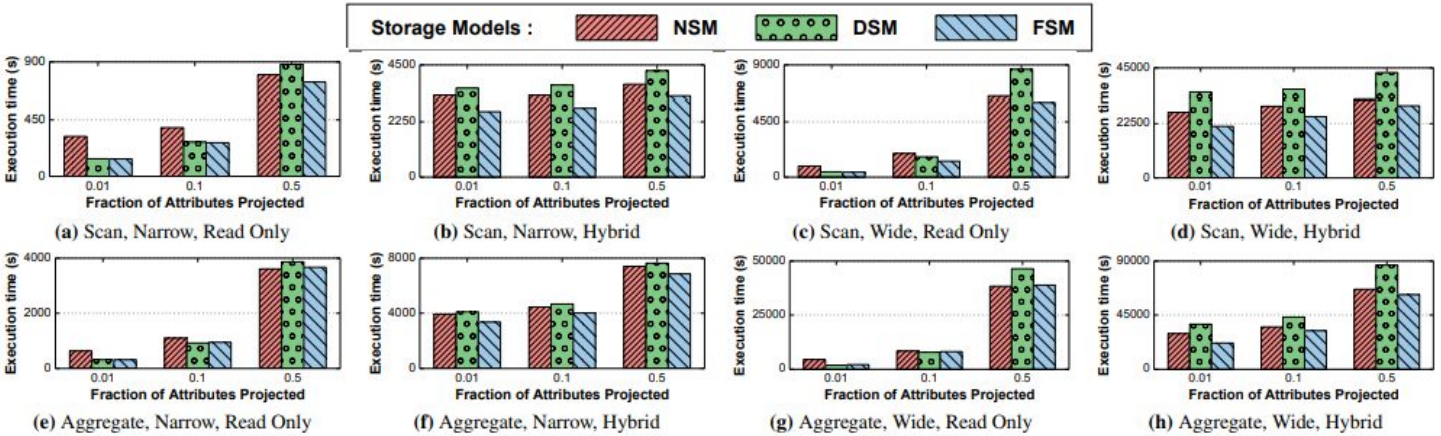


Figure 7: Projectivity Measurements – The impact of the storage layout on the query processing time under different projectivity settings. The execution engine runs the workload with different underlying storage managers on both the narrow and the wide table.

A Workload Aware Adaptation of the Earlier Experiment

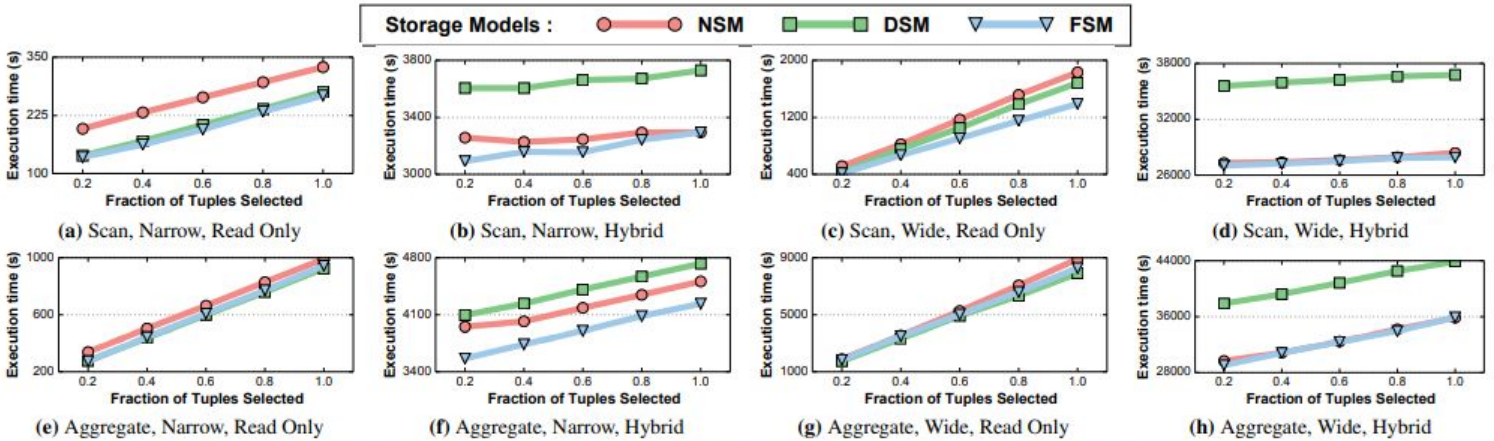
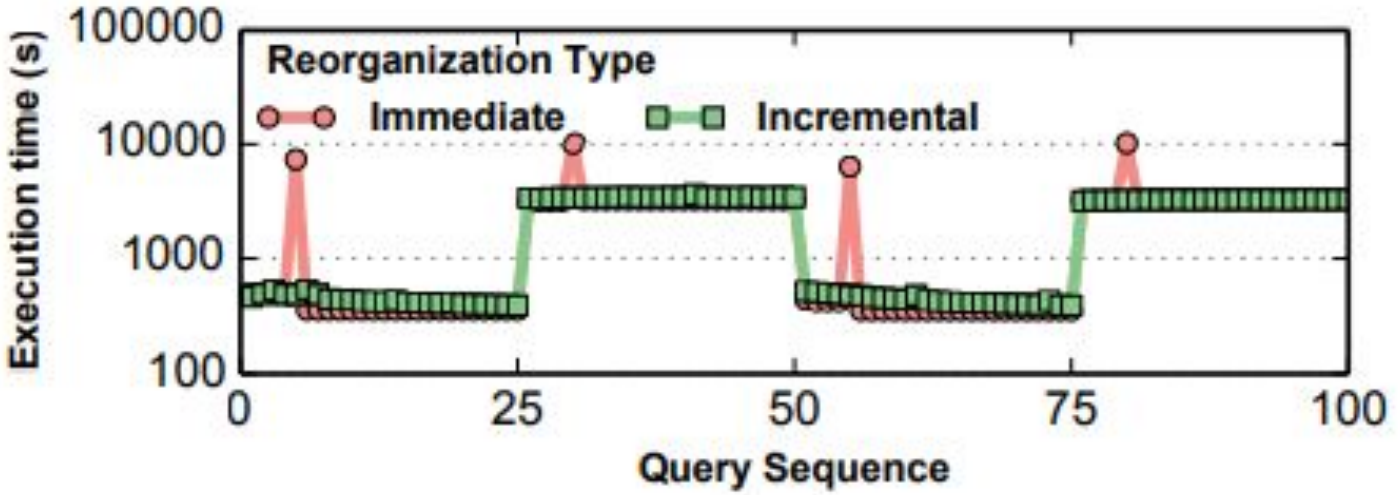


Figure 8: Selectivity Measurements – The impact of the storage layout on the query processing time under different selectivity settings. The execution engine runs the workload with different underlying storage managers on both the narrow and the wide table.

Immediate vs Incremental Reorganization



Next Steps:

- Setting up the DBMS so that it should automatically adjust Peloton knobs for the layout reorganization process based on the HTAP workload.
- Investigate the design of a self-driving module within the DBMS that dynamically adjusts these knobs to simplify the tuning process.
- Explore code generation and data compression techniques for optimizing query execution

slido

How understandable did you find the article and what ideas interested you from it?

ⓘ Start presenting to display the poll results on this slide.

Discussion Questions

- How understandable did you find the article and what ideas interested you from it?
- What real world examples of pipelines could benefit from this hybrid DBMS?
- Now let’s discuss… what are your questions!

Thank you for listening!