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

John C. Merfeld – 2 / 7 / 19

BACKGROUND

(getting everyone on board with jargon)

We should know these “words”

- DBMS –
- OLTP –
- OLAP –
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP –
- OLAP –
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP – On-Line Transaction Processing
- OLAP –
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP –
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line Analytical Processing
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP –

We should know these “words”

- DBMS – DataBase Management System
- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing

We should know these "words"

- DBMS – DataBase Management System
- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing

Workloads

We should know these “words”

- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing
- NSM –
- DSM –

We should know these “words”

- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing
- NSM – *n*-ary Storage Model
- DSM –

We should know these “words”

- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing
- NSM – *n*-ary Storage Model (Why is this good for writes?)
- DSM –

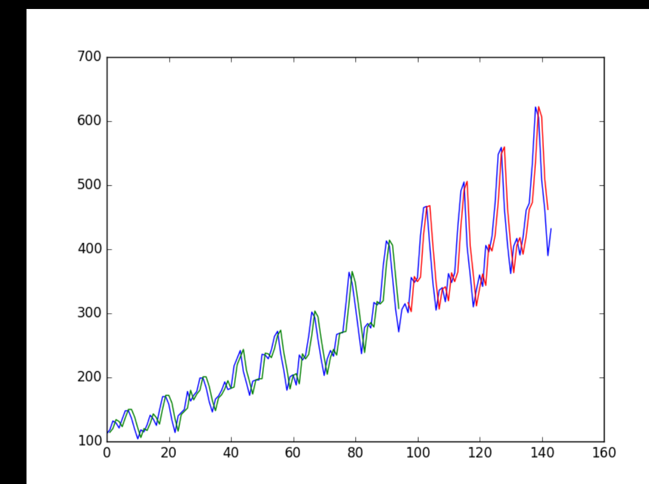
We should know these “words”

- OLTP – On-Line **Transaction** Processing (**HOT**)
- OLAP – On-Line **Analytical** Processing (**COLD**)
- HTAP – Hybrid **Transactional**-**Analytical** Processing
- NSM – *n*-ary Storage Model (Why is this good for writes?)
- DSM – Decomposed Storage Model

Storage models

Here's your pneumonic device

O L **T** P



O L **A** P

Tea is **hot**

You Analyze **history**

new data
(Updates and writes)

History is **(c)old**
(Scans and aggregations)

n-ary Storage

Decomposed Storage

THE PROBLEM

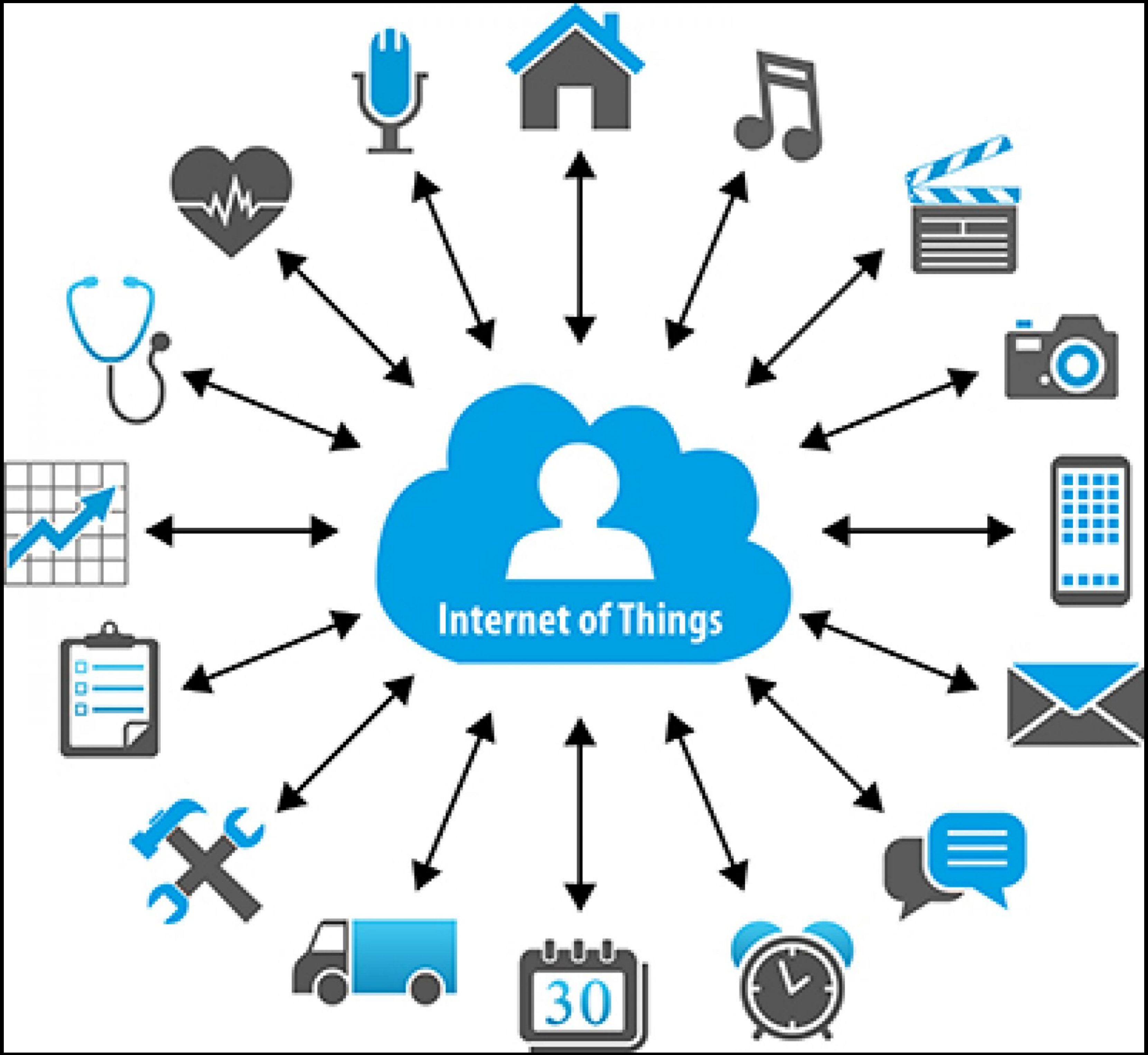
Today we’re concerned with HTAP

- Not just a “dynamic workload”
- Transactions and analytics queries running simultaneously
- Both historical and fresh data are equally relevant to analysis

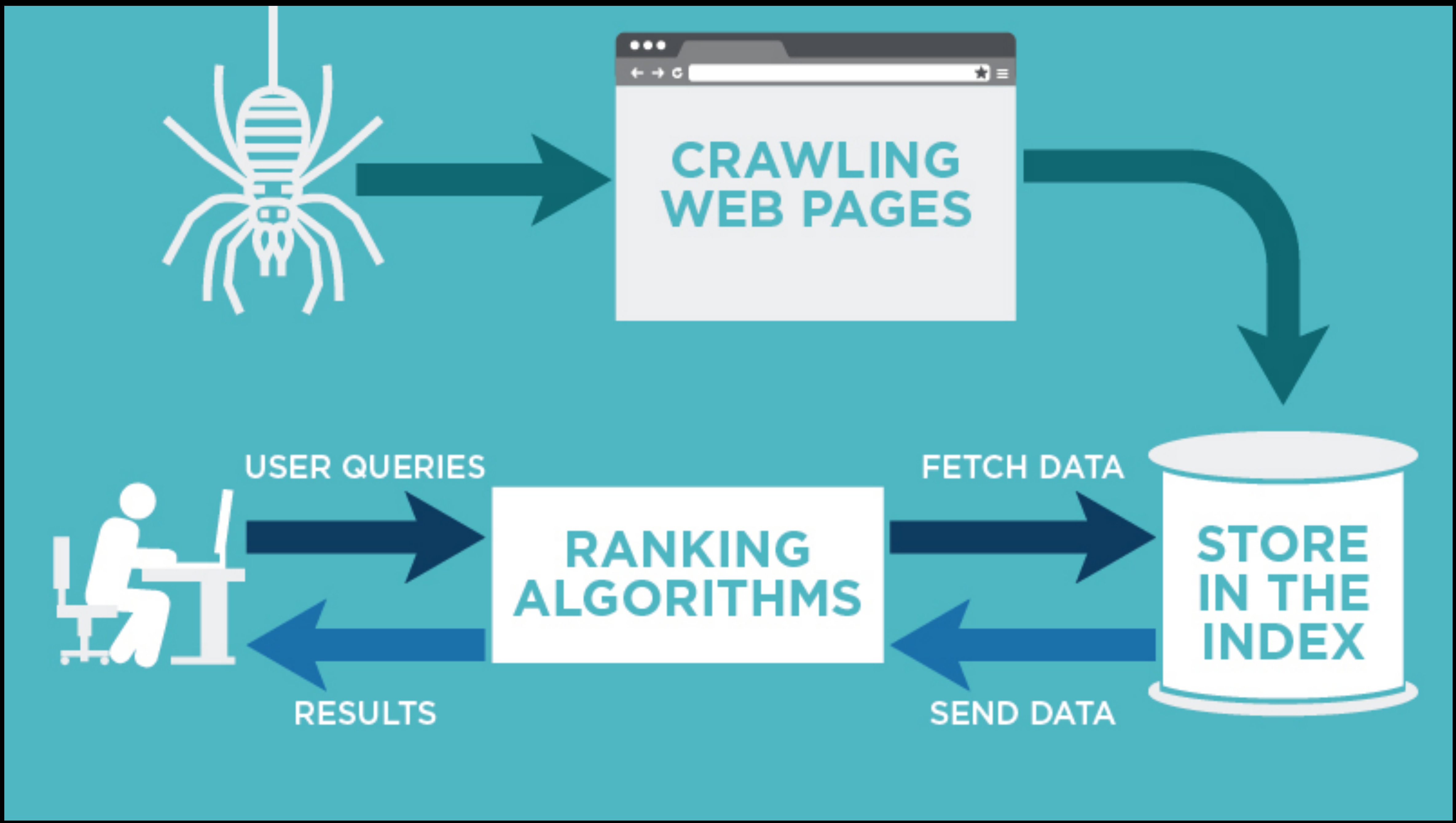
Today we’re concerned with HTAP

- Not just a “dynamic workload”
- Transactions and analytics queries running simultaneously
- Both historical and fresh data are equally relevant to analysis
- Examples?

You might work with IoT sensors



Or you might run a search engine



Or you might feel positively about the concept of money

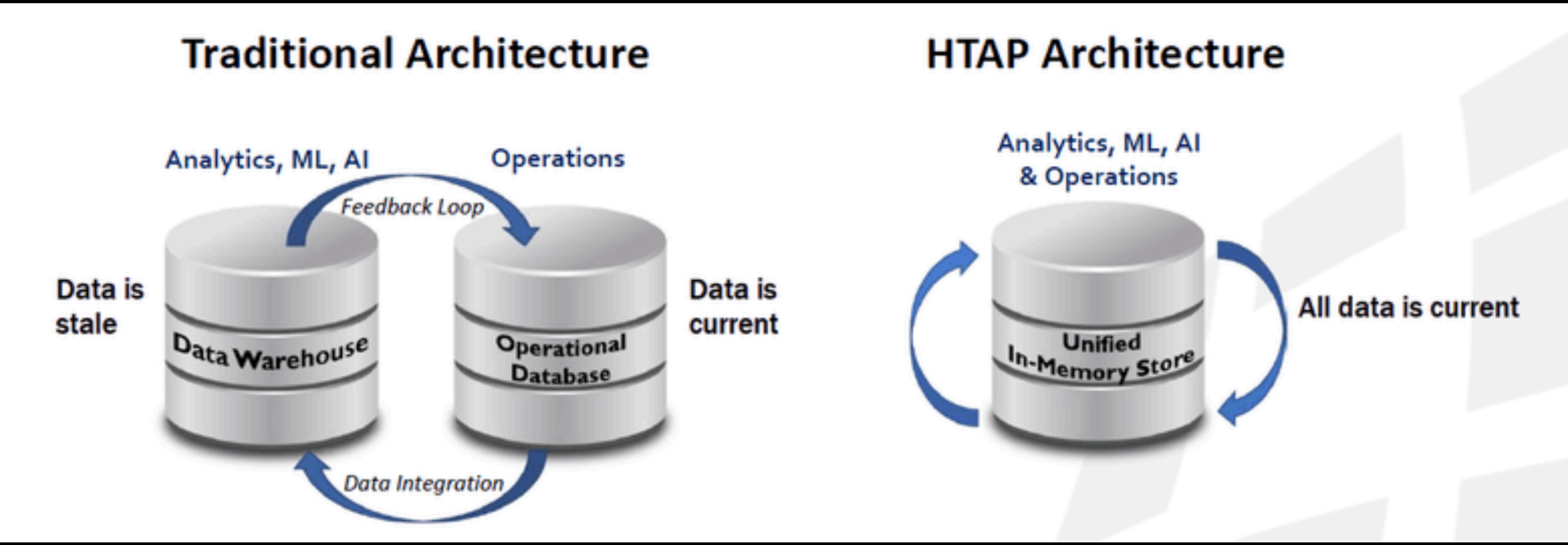
One approach is to physically separate the use cases

System A gives data to System B at… some point!	System A	System B
Workload	OLTP	OLAP
Storage Model	n-ary	Decomposed
Data stored as…	Rows	Columns
Used for…	Inserts & Updates	Reads

Is this really HTAP?

System A gives data to System B at… some point!	System A	System B
Workload	OLTP	OLAP
Storage Model	n-ary	Decomposed
Data stored as…	Rows	Columns
Used for…	Inserts & Updates	Reads

Separating the use cases defeats the purpose of HTAP



There are other reasons not to want two distinct systems

- What are they?

There are other reasons not to want two distinct systems

- What are they?
    - Two different execution engines

There are other reasons not to want two distinct systems

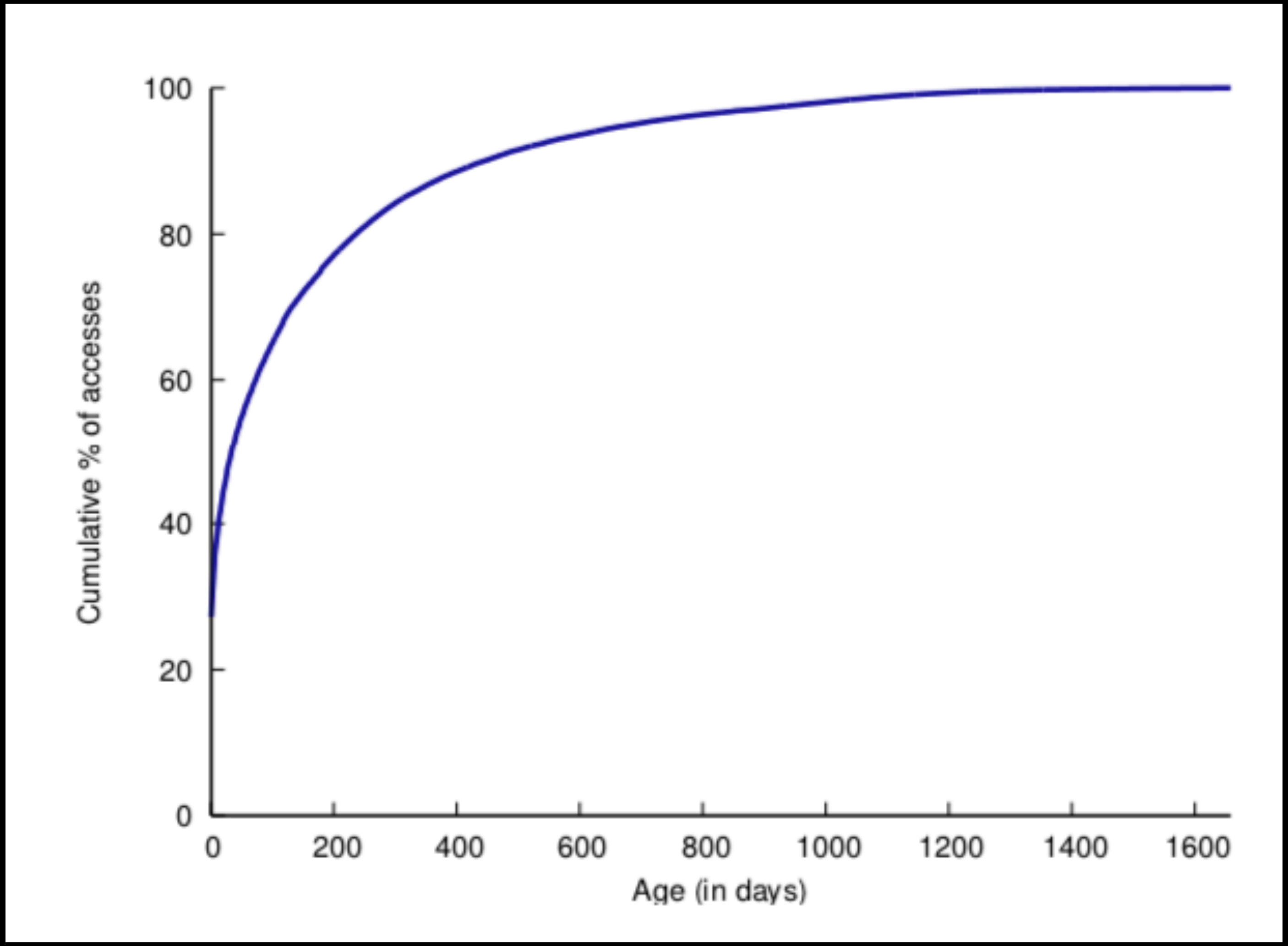
- What are they?
  - Two different execution engines
    - Twice the software

There are other reasons not to want two distinct systems

- What are they?
    - Two different execution engines
        - Twice the software
            - Twice the people!
                - (at least twice the cost…)

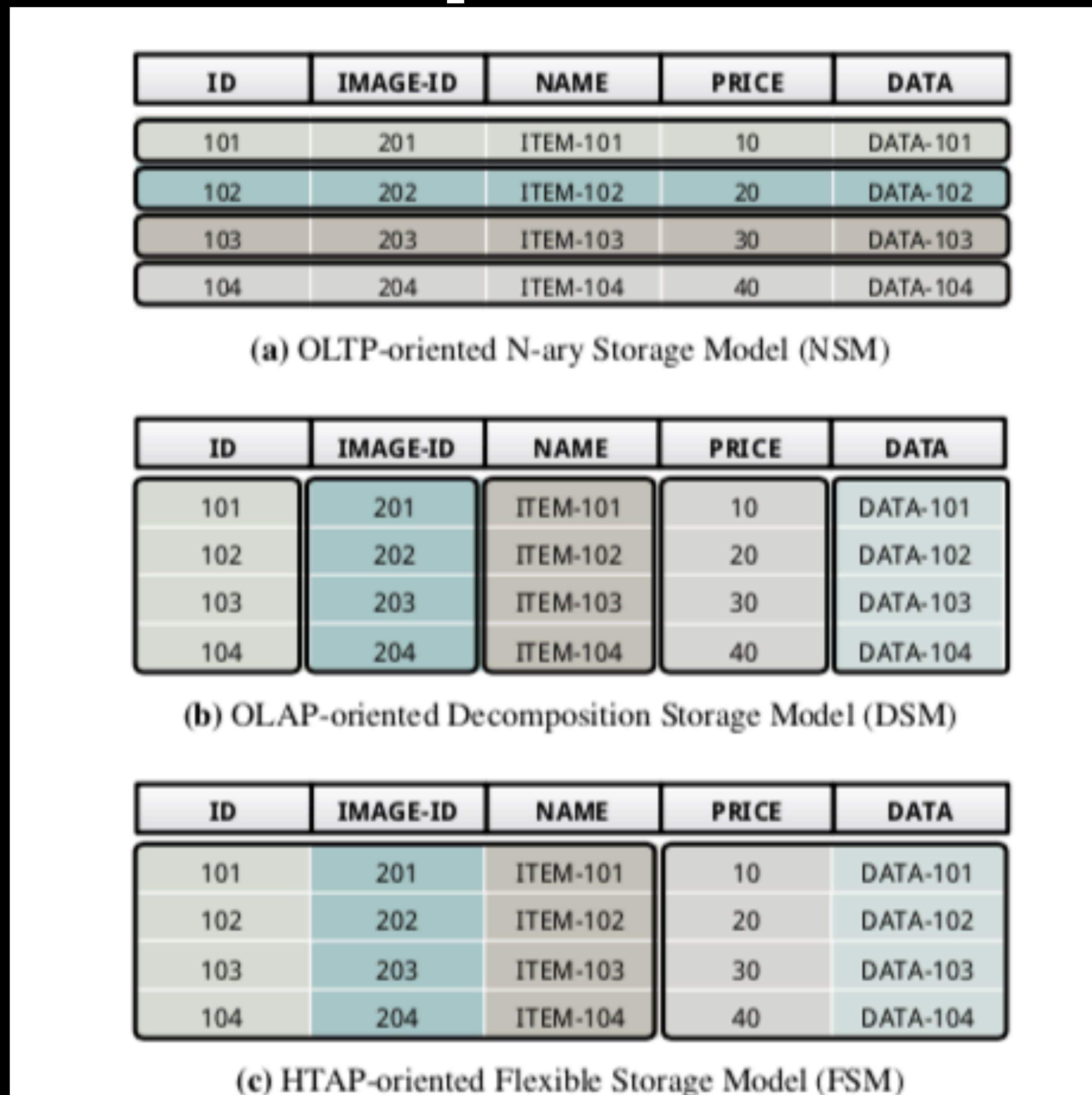
THE SOLUTION

A flexible storage model (FSM) takes the "temperature" of tuples



Source: D. Beaver, S. Kumar, H. C. Li, J. Sobel, P. Vajgel, and F. Inc. *Finding a needle in haystack: Facebook's photo storage*. In *OSDI*, 2010.

A flexible storage model (FSM) takes the "temperature" of tuples



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

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

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

A “tile” is part row, part column

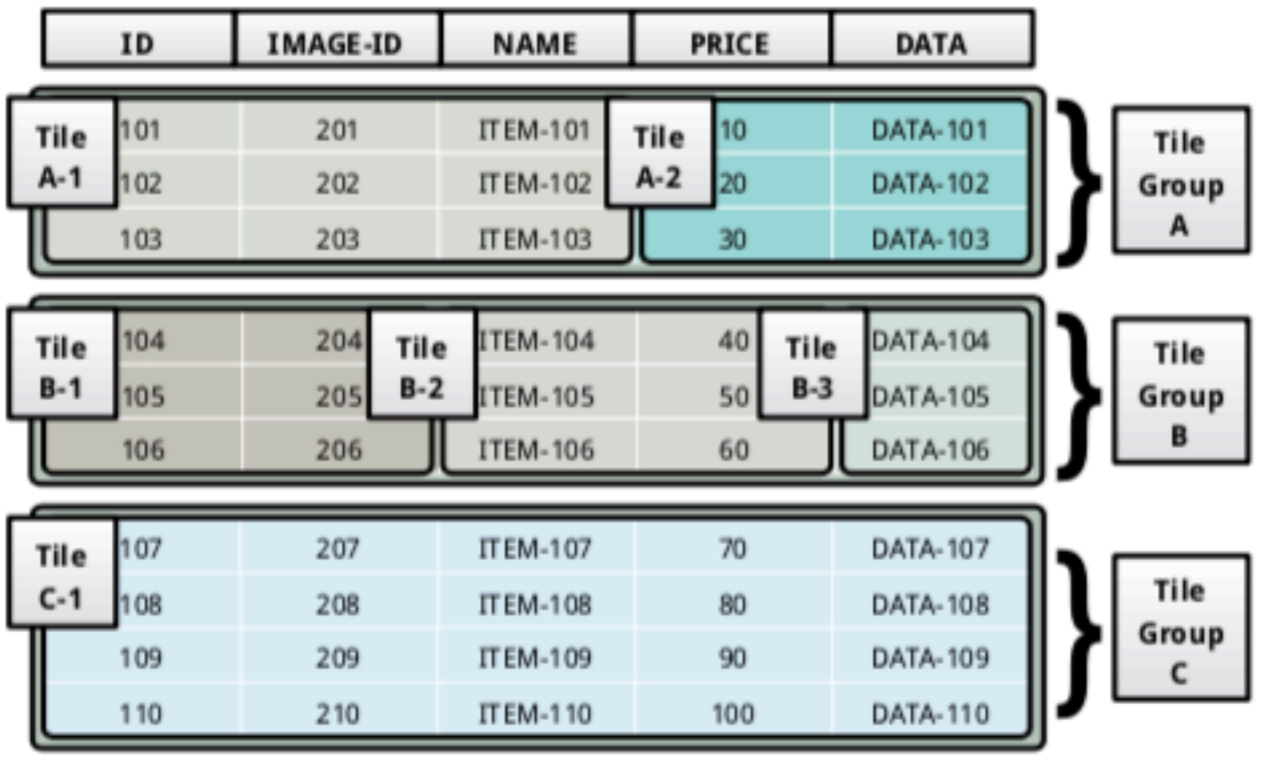


Figure 3: Physical Tile – An example storage layout of a table composed of physical tiles. This table comprises of three tile groups (A, B, C).

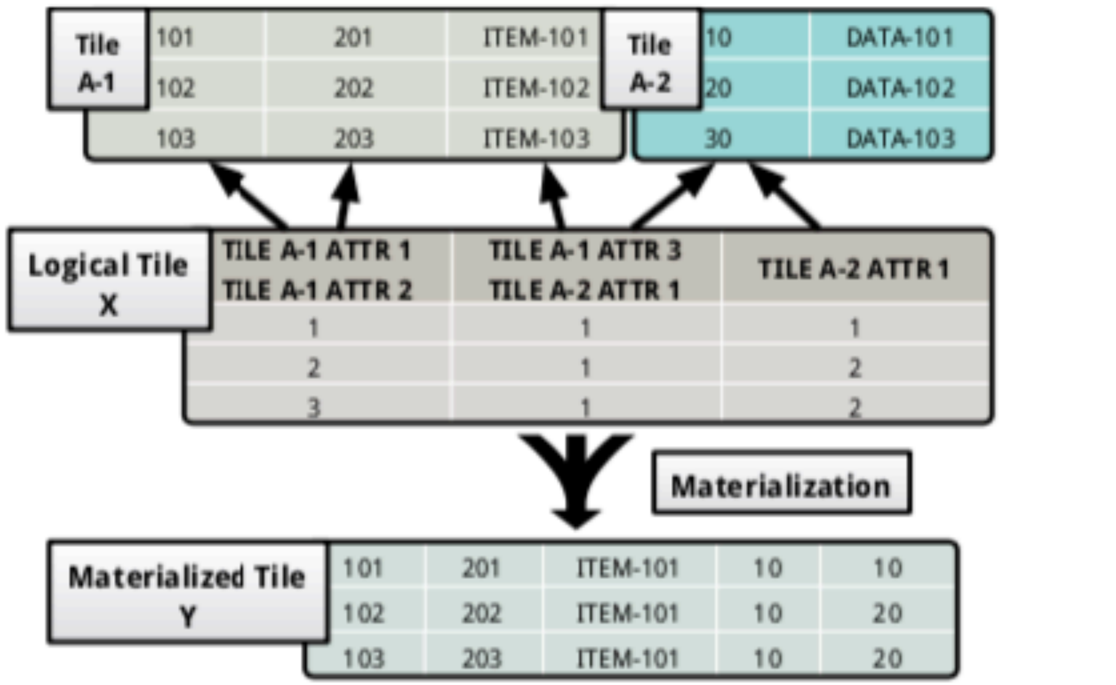
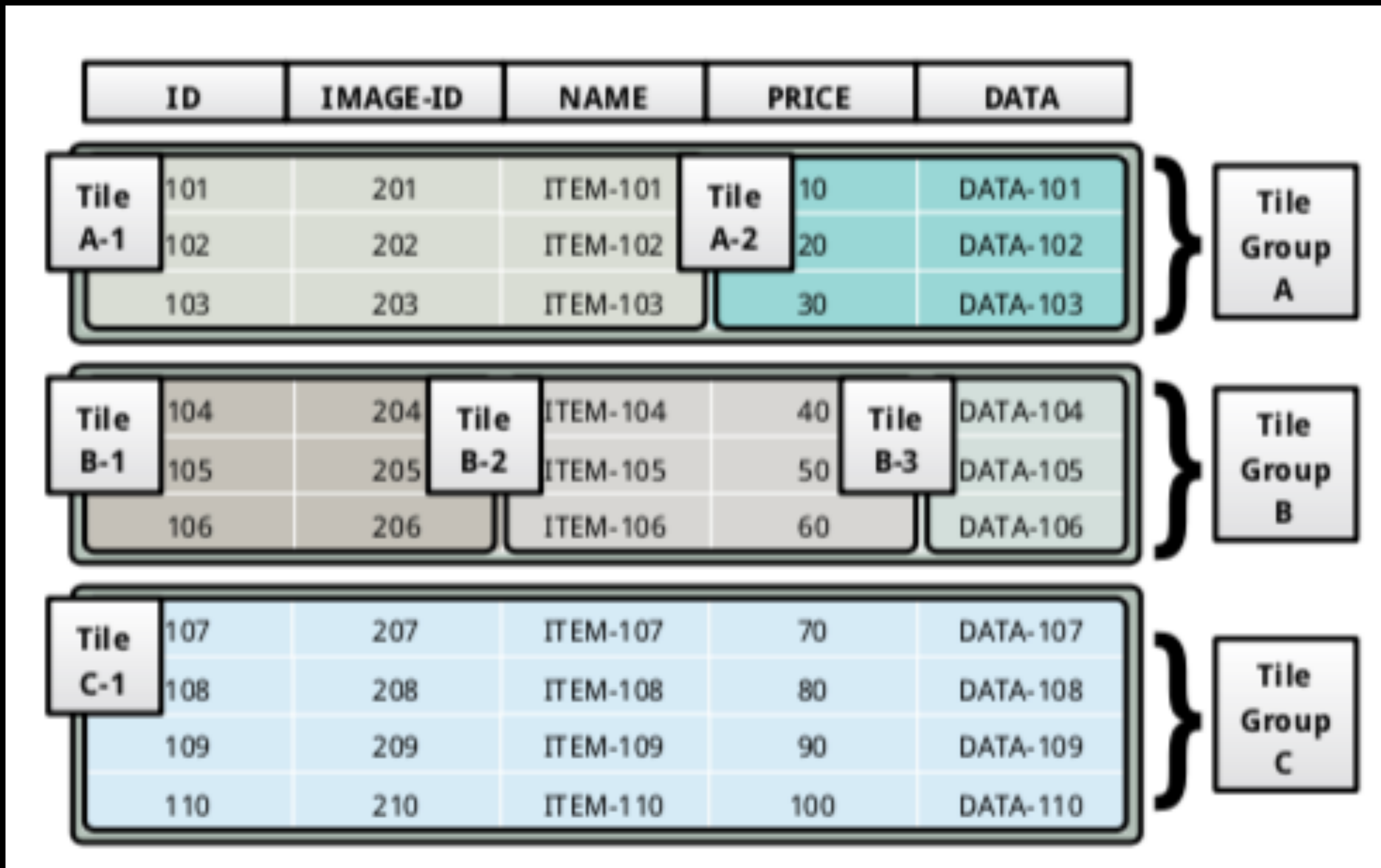


Figure 4: Logical Tile – An example of a logical tile representing data spread across a couple of physical tiles (A-1, A-2).

“Physical tiles” store subsets of tuple attributes



Great! Let’s put physical tiles in our favorite DBMS

Great! Let’s put physical tiles in our favorite DBMS

Oh, we can’t?

Why can’t we just put physical tiles in our favorite DBMS?

Why can’t we just put physical tiles in our favorite DBMS?

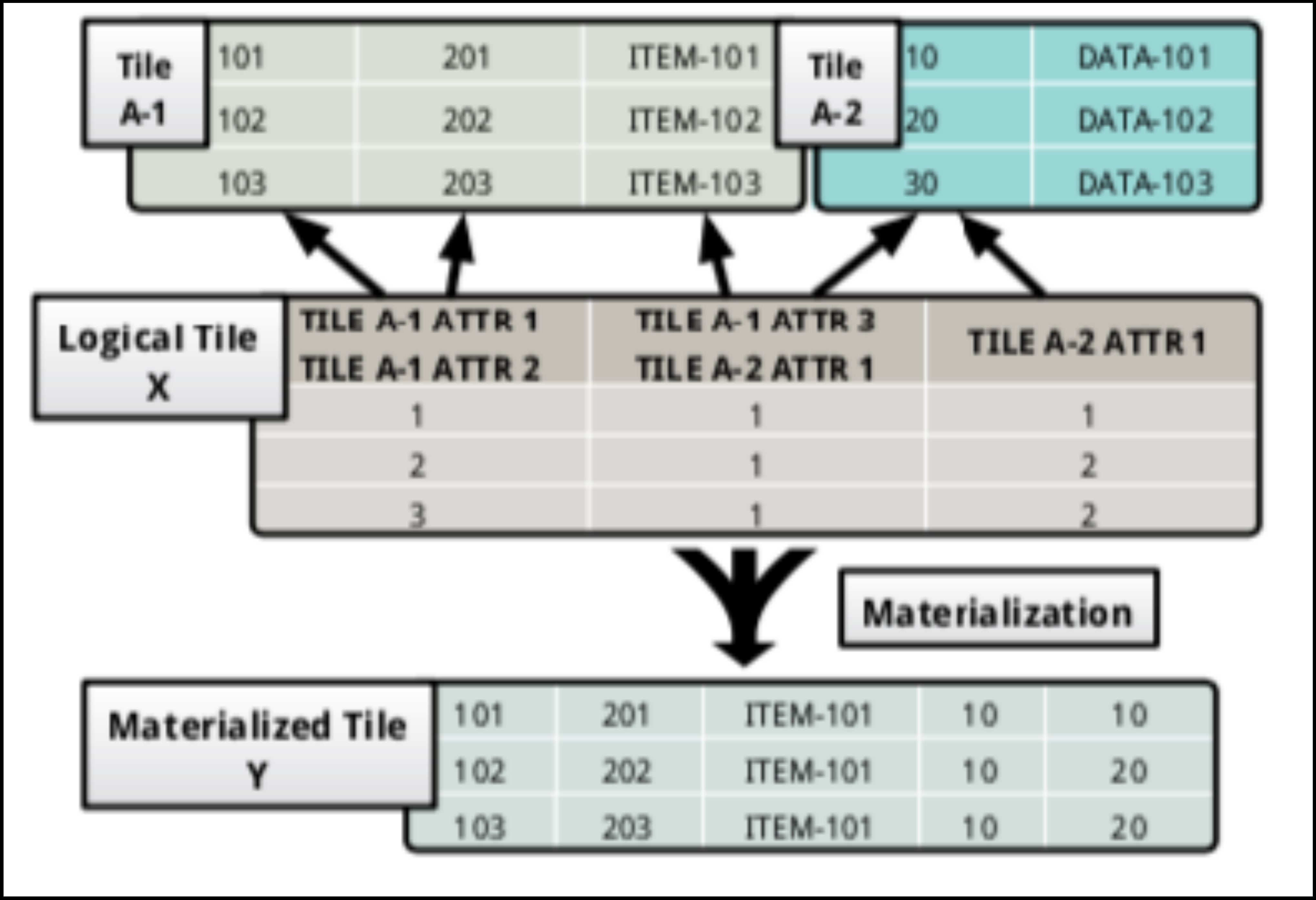
- Two words:

Why can't we just put physical tiles in our favorite DBMS?

- Two words:

Query execution!

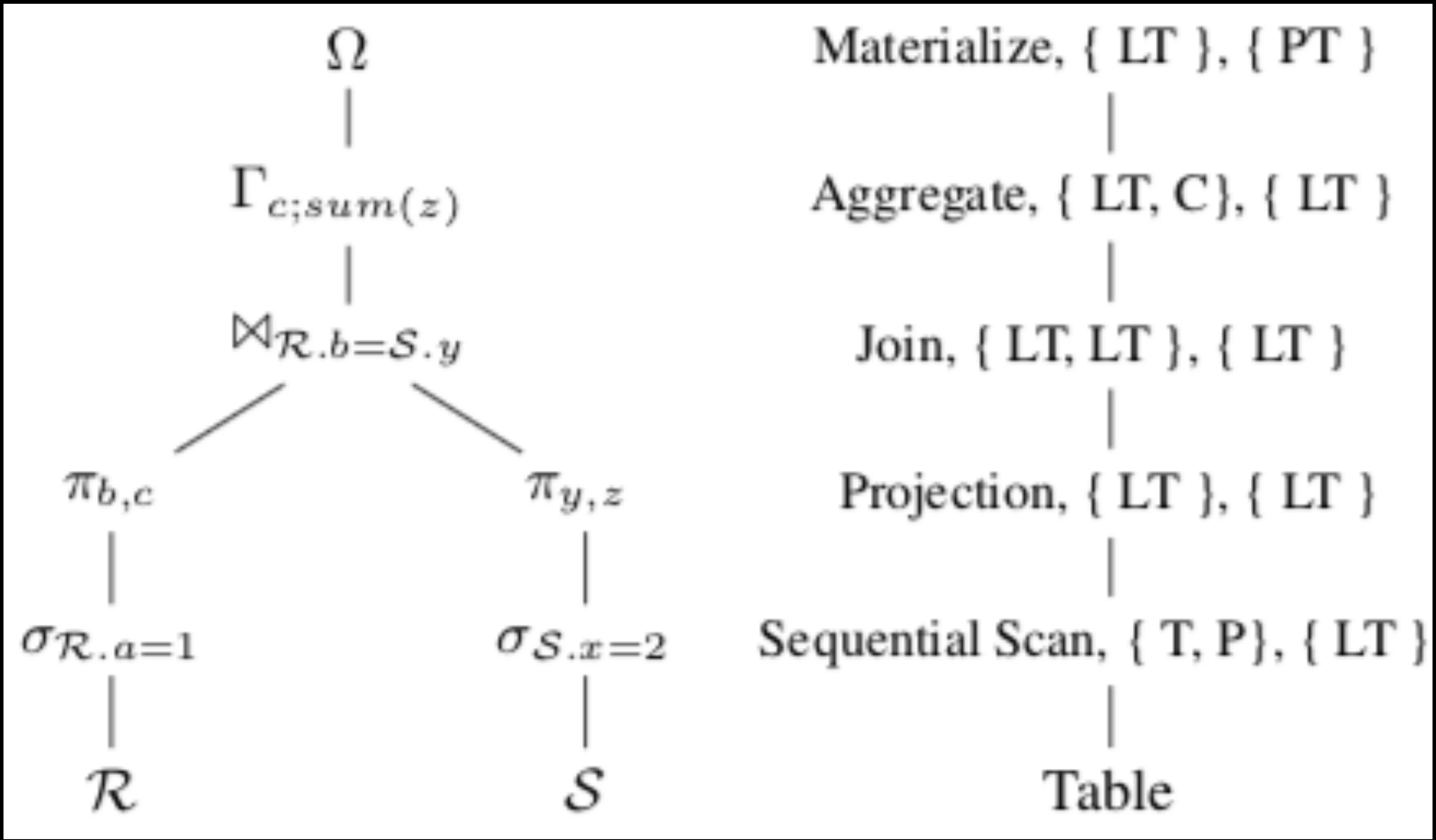
“Logical tiles” store information about multiple physical tiles



Logical tile columns contain sets of physical tiles columns

- The underlying physical data are released during materialization

“Tile algebra” is an abstracted extension of relational algebra



Tile algebra offers several advantages

Tile algebra offers several advantages

- Single execution engine
- Vectorized processing (tiles instead of tuples)
- DBMS can optimize what materializes when and what goes in the cache

The paper goes into detail about concurrency protocols

- And if you care about that, I invite you to read the paper!

None of this matters unless tile layouts can be reconfigured

- How might we do this?

None of this matters unless tile layouts can be reconfigured

- How might we do this?
    - Copy data to optimal layout before executing query?

None of this matters unless tile layouts can be reconfigured

- How might we do this?
  - ~~Copy data to optimal layout before executing query?~~

None of this matters unless tile layouts can be reconfigured

- How might we do this?
  - ~~Copy data to optimal layout before executing query?~~
  - Background process reorganize one tile at a time?

None of this matters unless tile layouts can be reconfigured

- How might we do this?
    - ~~Copy data to optimal layout before executing query?~~
    - Background process reorganize one tile at a time?

The system needs to gather statistics about incoming queries

- Record attributes found in SELECT and WHERE clauses

The system needs to gather statistics about incoming queries

- Record attributes found in SELECT and WHERE clauses

- Only do this for a random sample of queries

The system needs to gather statistics about incoming queries

- Record attributes found in SELECT and WHERE clauses
- Only do this for a random sample of queries
    - We have millions of writes and only a few big reads…

      …is that a problem?

The system needs to gather statistics about incoming queries

- Record attributes found in SELECT and WHERE clauses

- Only do this for a random sample of queries

    - We have millions of writes and only a few big reads…

      …is that a problem?

    - Record the cost of the queries too

These statistics are used to re-partition the tables into new tiles

- Clustering algorithm chooses which attributes belong together in physical tiles
- Greedy algorithm groups tiles together based on how "important" they are to workloads
- This is done incrementally to amortize the cost

EVALUATION

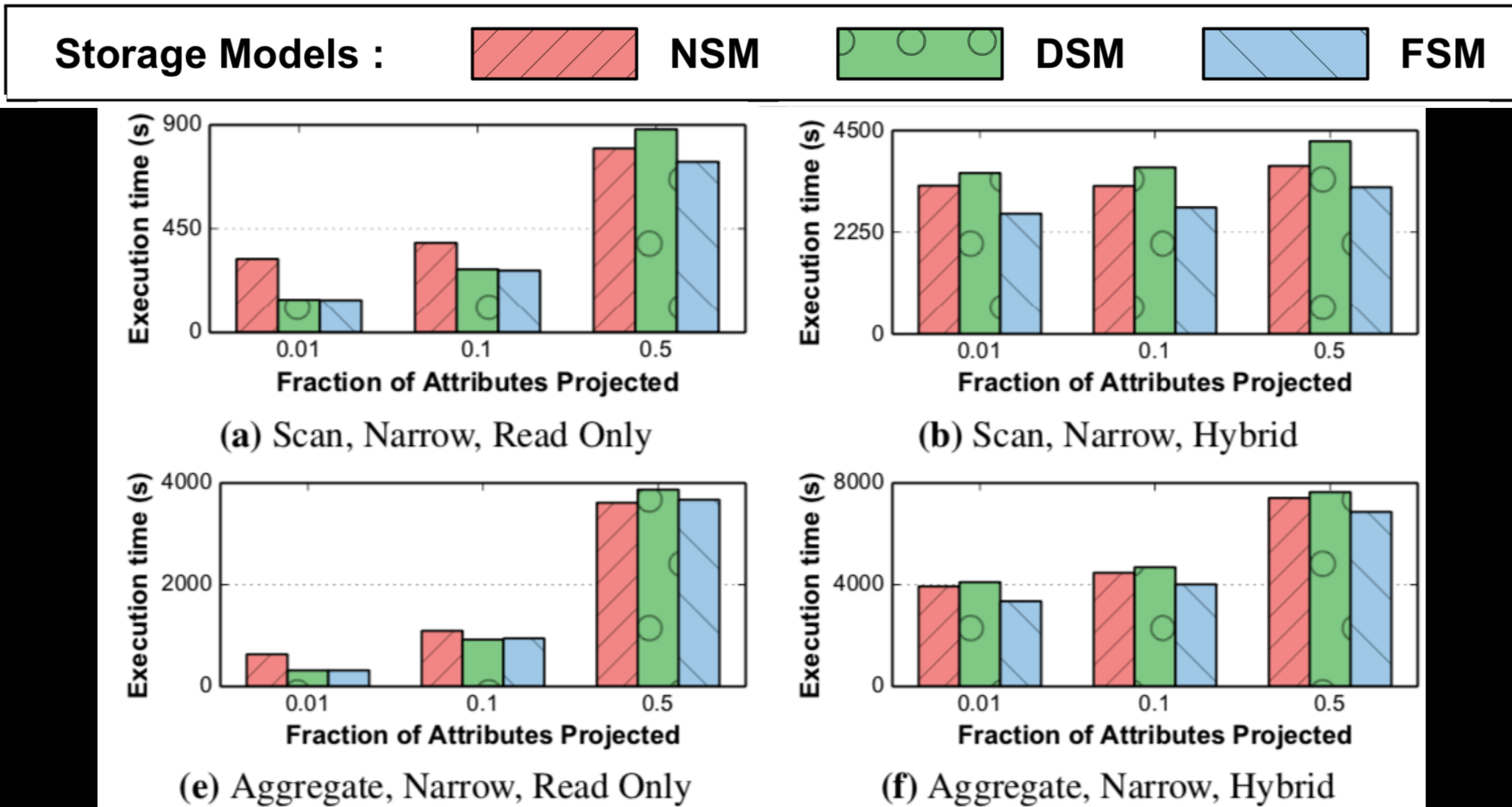
(NSM vs. DSM vs. FSM)

The system was evaluated using workloads based on these queries

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),...,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,...,X.a_k,Y.a_1,...,Y.a_k
FROM R **AS** X, R **AS** Y **WHERE** X.a_i < Y.a_j

Note that different values for k and δ alter the projectivity and the selectivity of the queries, respectively. We use different workloads comprised of these query types to evaluate the impact of the storage models on the performance of the DBMS.

Narrow => 50 attributes; Hybrid => 1M writes per read



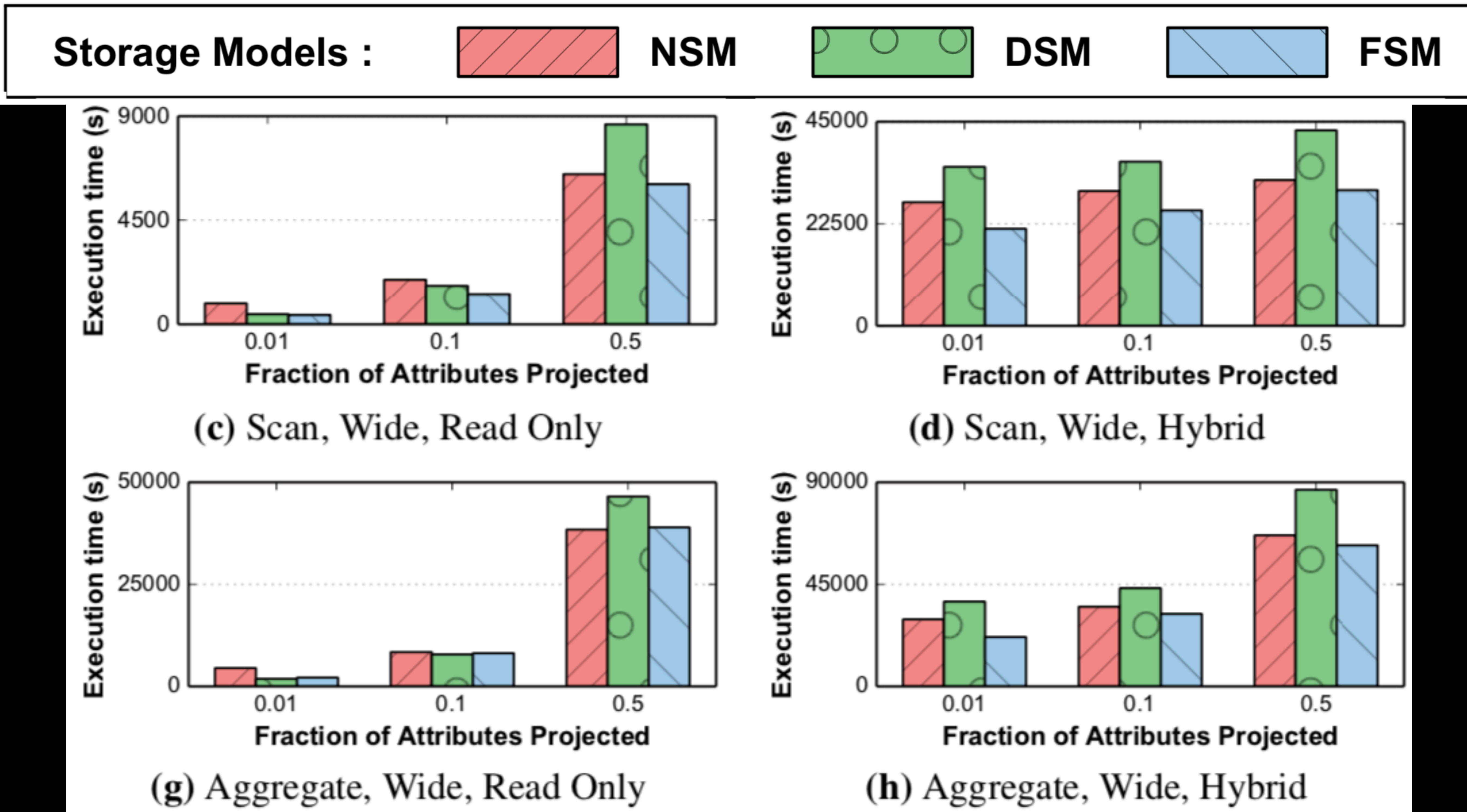
(a) Scan, Narrow, Read Only

(b) Scan, Narrow, Hybrid

(e) Aggregate, Narrow, Read Only

(f) Aggregate, Narrow, Hybrid

Wide => 500 attributes; Aggregate => MAX(x, y, z, …)



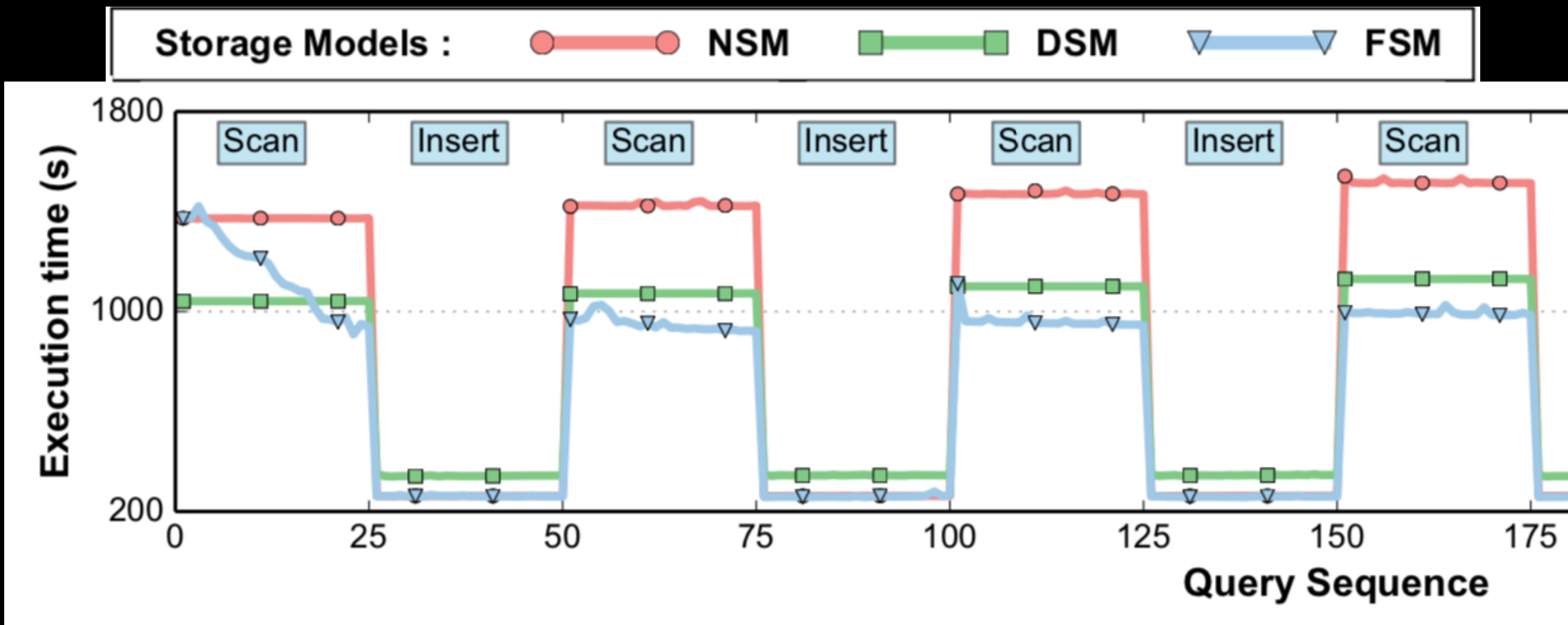
(c) Scan, Wide, Read Only

(d) Scan, Wide, Hybrid

(g) Aggregate, Wide, Read Only

(h) Aggregate, Wide, Hybrid

We can see how FSM learns over time



CONCLUSION

I think the paper did a pretty good job of…

- Demonstrating the importance of the problem and their solution
- (Usually) going into the right amount of detail
- Talking about different ways to implement each step
- Conducting a lot of difference experiments

I wish the paper had…

- Done more join queries besides a few self-joins (**major**)
- Said either way more or slightly less about tiles
  (I know they said way more in the appendix, but… yikes)
- Benchmarked their approach against another HTAP solution
  (there was some hand waving in their critiques of such systems)