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

J Arulraj, et el.

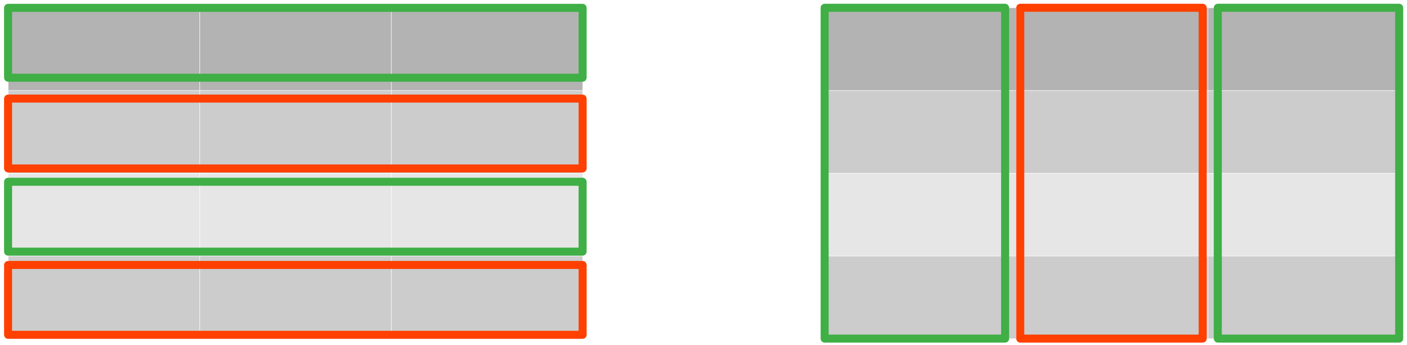
SIGMOD ’20

Speaker: Yu-Cheng Huang

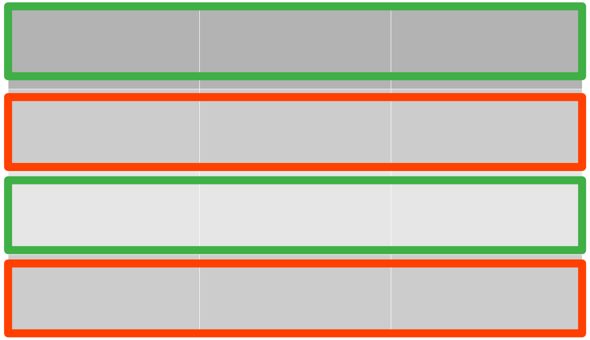
Introduction

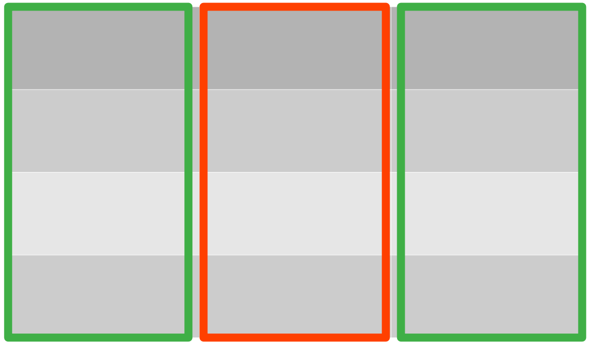
What are the two basic memory layout?

Physical Memory layout



Physical Memory layout

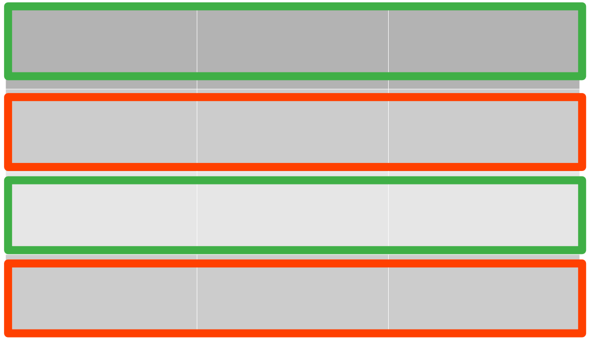




?

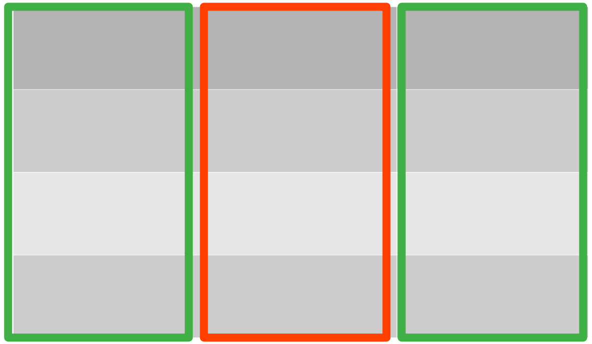
?

Physical Memory layout



Row Storage

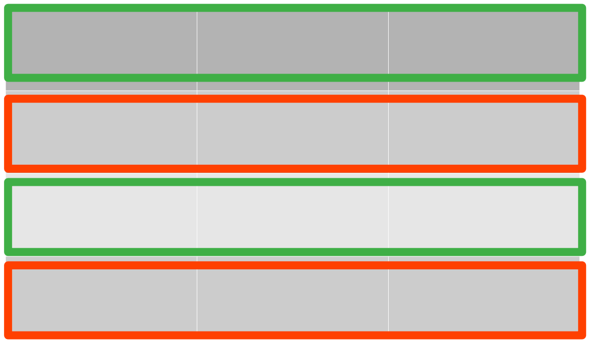
tuple-centric



Column Storage

Columnar

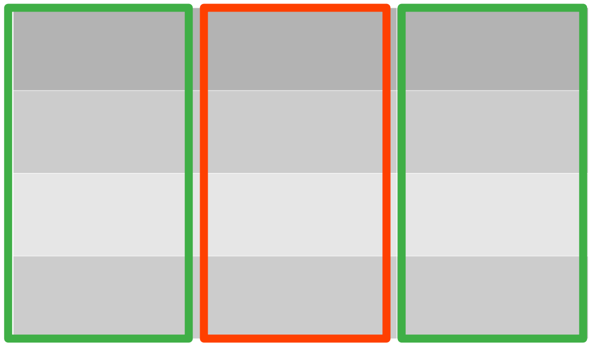
Physical Memory layout



Row Storage

tuple-centric

NSM

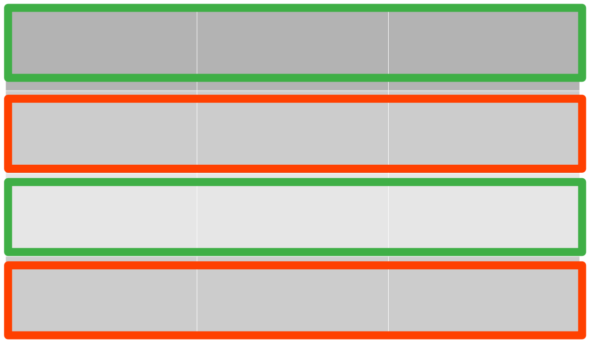


Column Storage

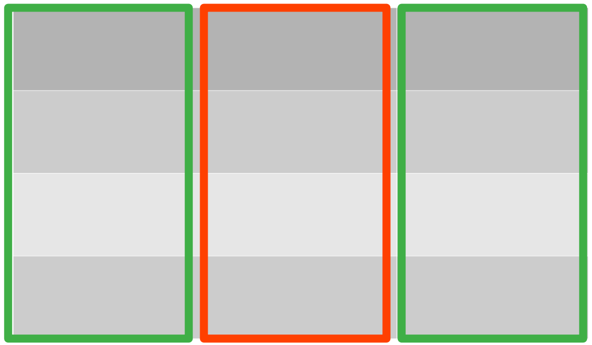
Columnar

DSM

Physical Memory layout



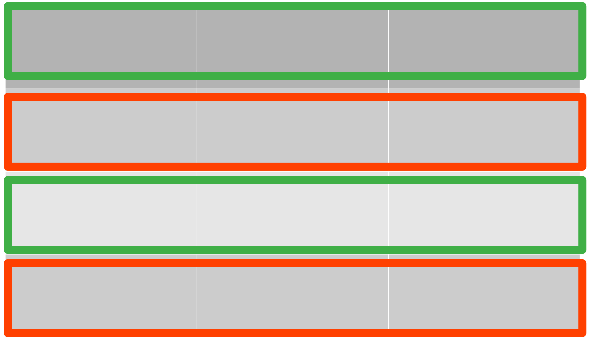
Row Storage



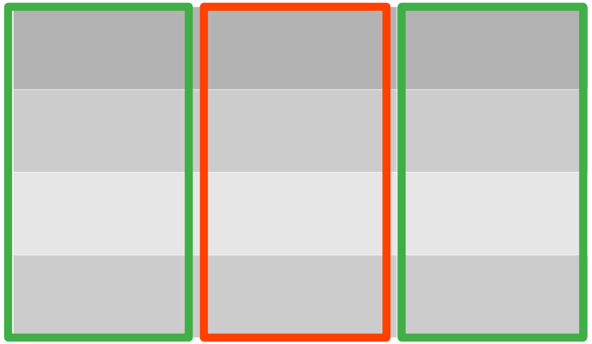
Column Storage

When?

Physical Memory layout



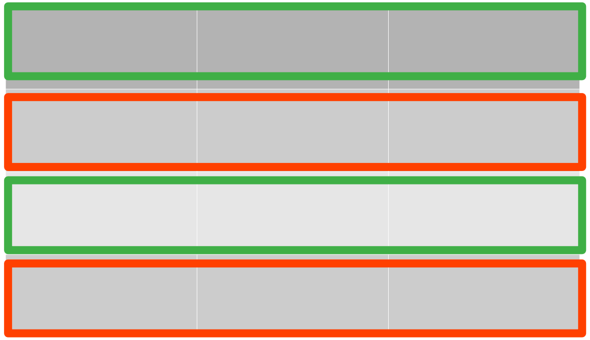
Row Storage



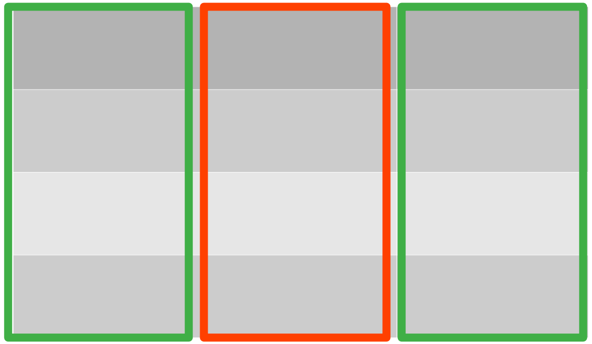
Column Storage

When? Insert, Update Search for an attribute

Physical Memory layout



Row Storage

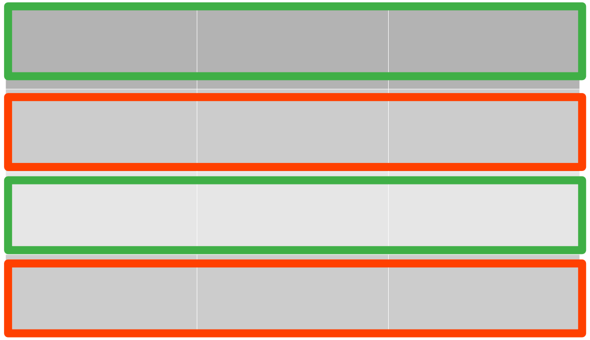


Column Storage

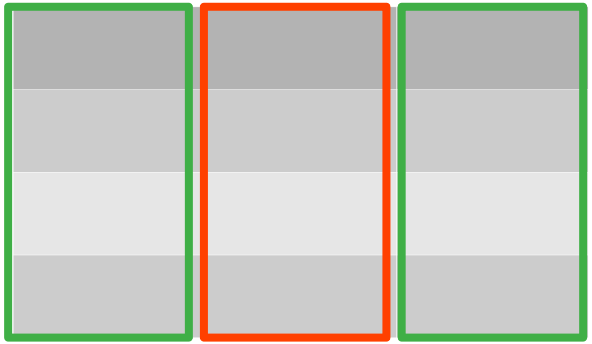
When? Insert, Update Search for an attribute

Data?

Physical Memory layout



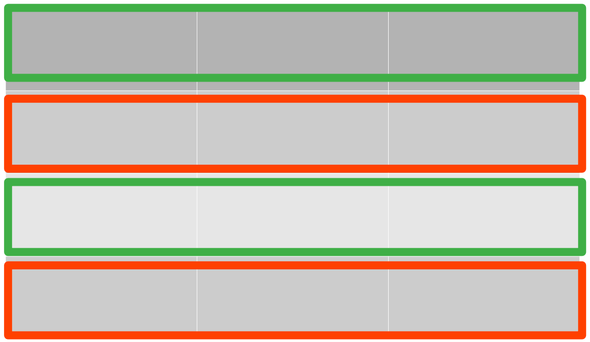
Row Storage



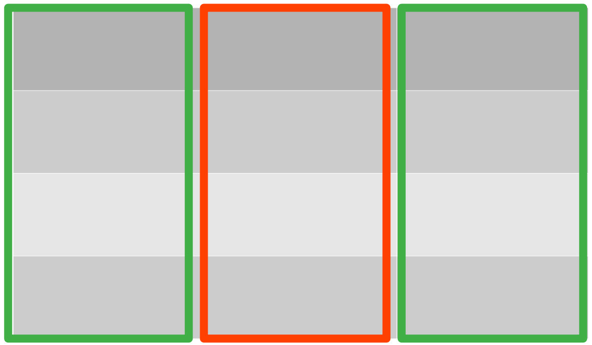
Column Storage

When?	Insert, Update	Search for an attribute
Data?	Hotly updated	Coldly stored

Physical Memory layout



Row Storage

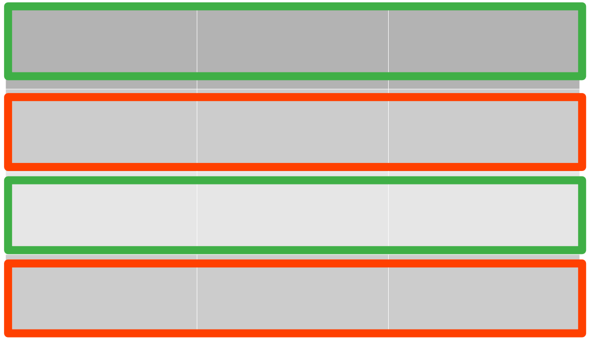


Column Storage

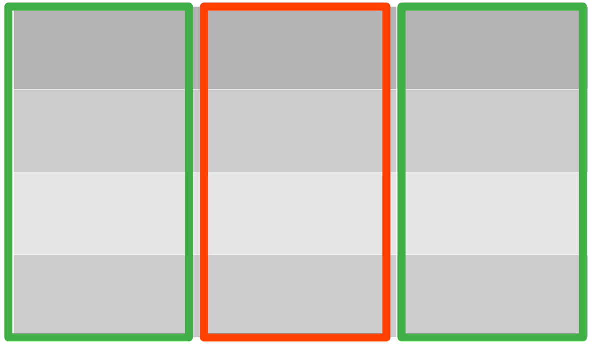
When? Insert, Update Search for an attribute

process?

Physical Memory layout



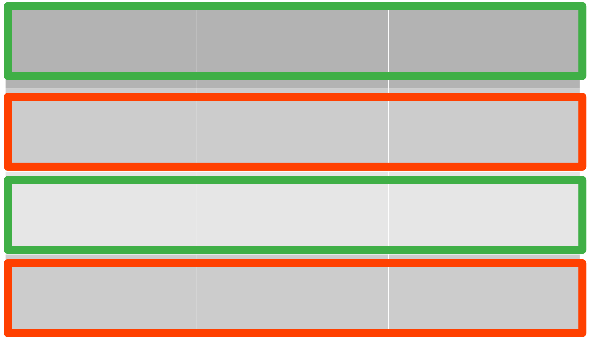
Row Storage



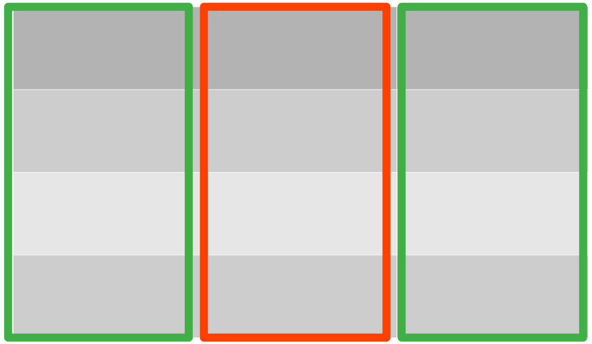
Column Storage

When?	Insert, Update	Search for an attribute
process?	Transactional	Analytical

Physical Memory layout



Row Storage

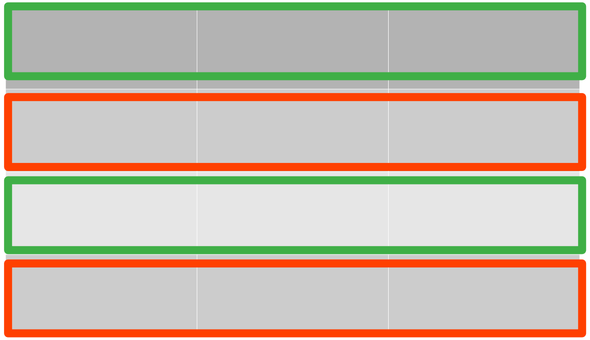


Column Storage

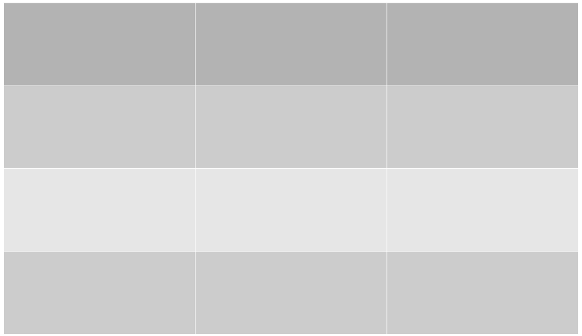
When? Insert, Update | Search for an attribute

process? OLTP | OLAP

Physical Memory layout



Row Storage

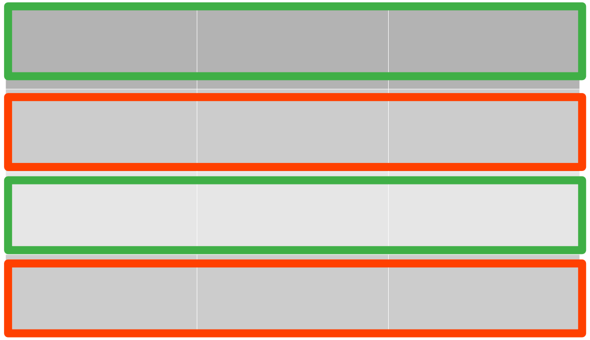


When? Insert, Update

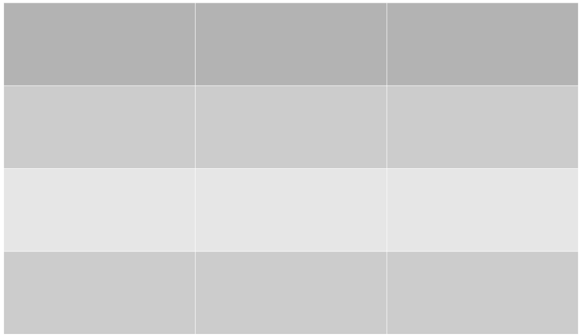
OLTP

Search for an attribute

Physical Memory layout



Row Storage

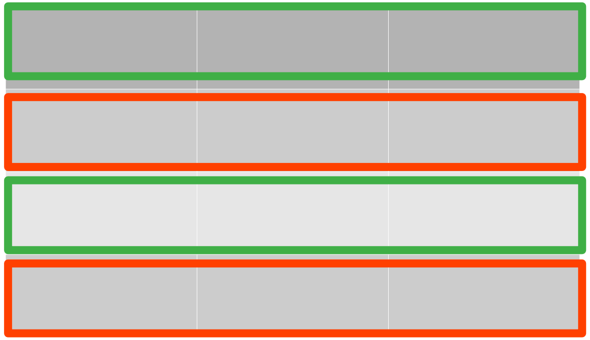


When? Insert, Update

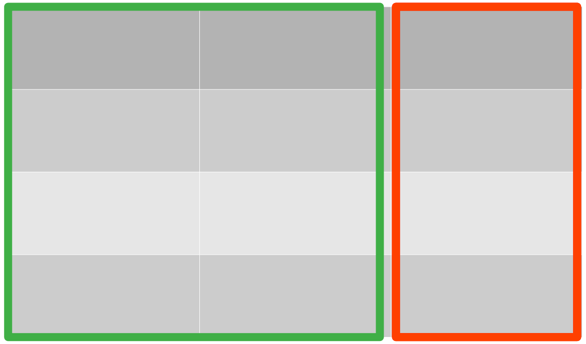
OLTP

Search for **(a1,a2)** attribute

Physical Memory layout



Row Storage

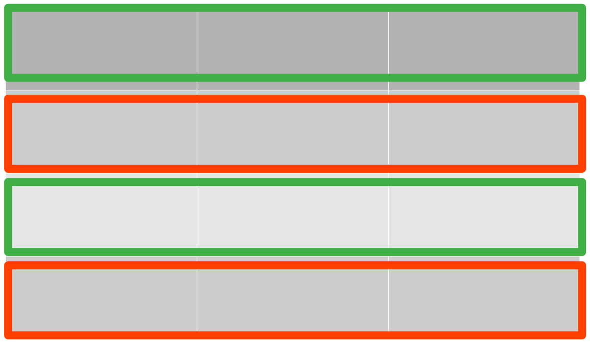


When? Insert, Update

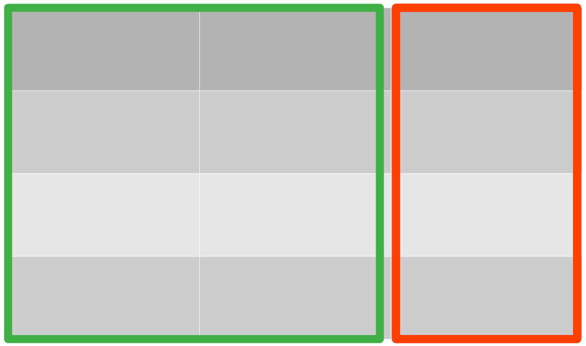
OLTP

Search for **(a1,a2)** attribute

Physical Memory layout



Row Storage



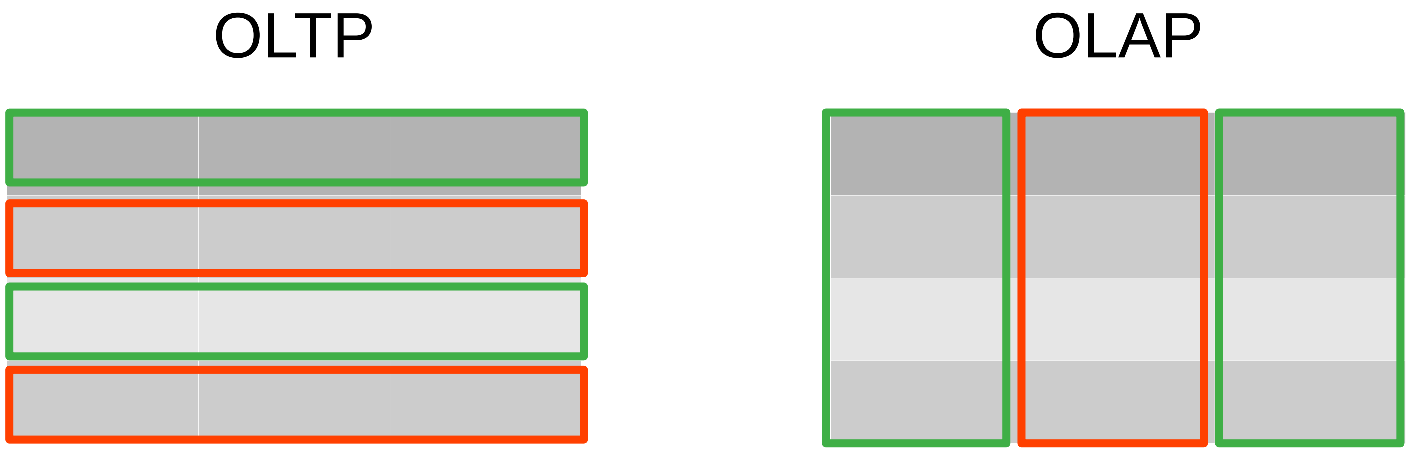
??

When? Insert, Update

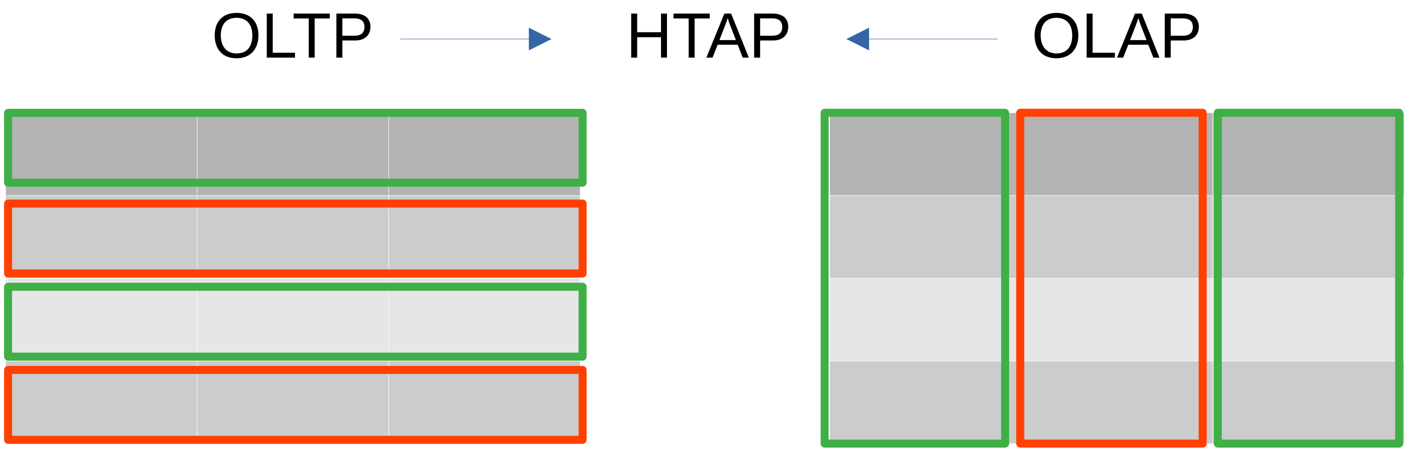
OLTP

Search for **(a1,a2)** attribute

Physical Memory layout



Physical Memory layout



Physical Memory layout

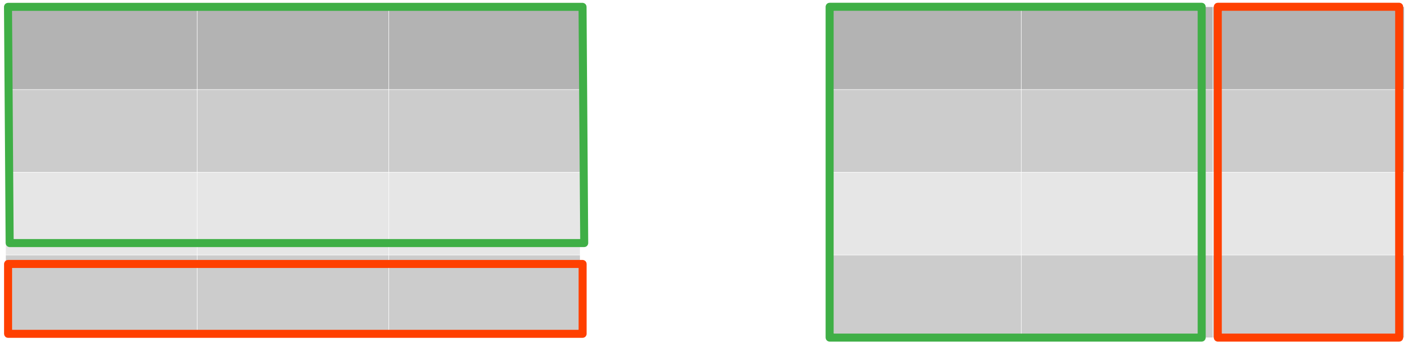
OLTP → HTAP ← OLAP

Physical Memory layout

HTAP

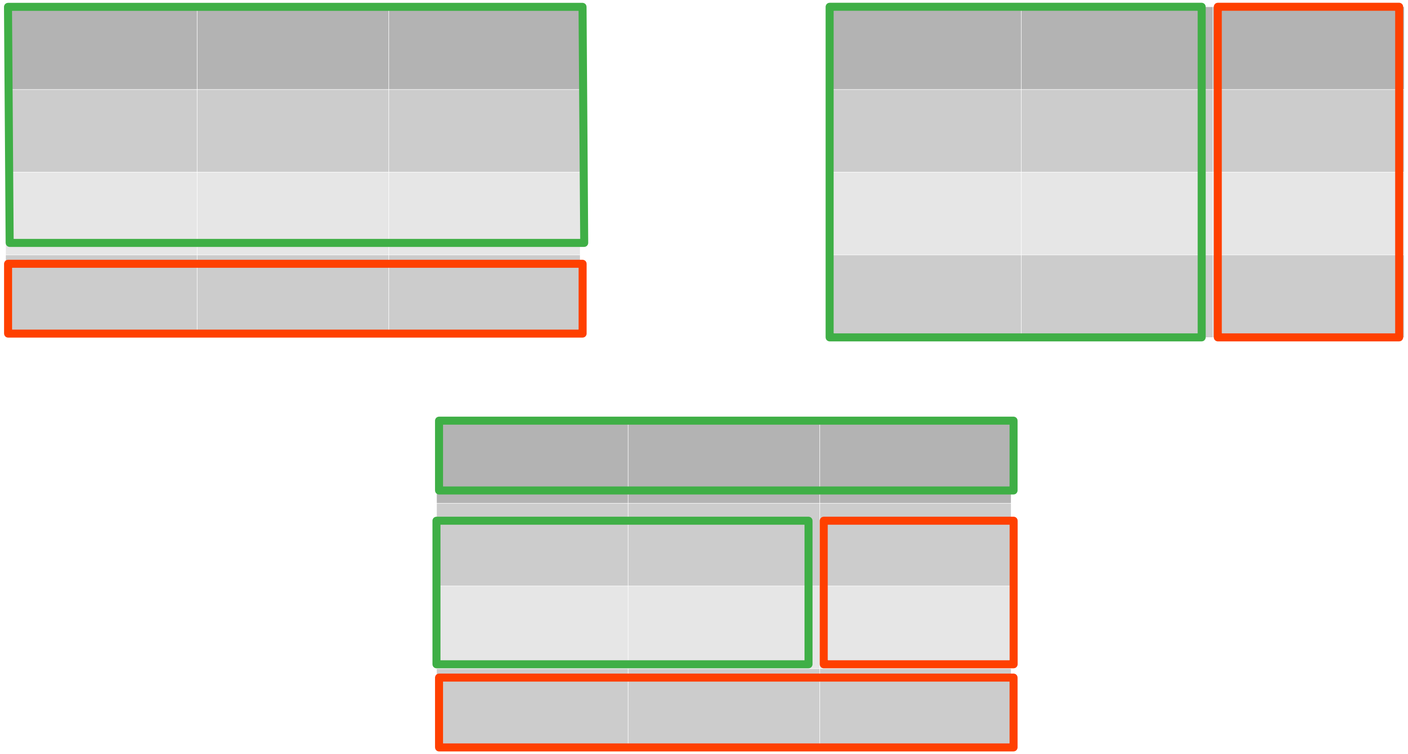
Physical Memory layout

HTAP



Physical Memory layout

HTAP



Which Memory layout

Which?

Which Memory layout

Which?

Do we have any observation?

Which Memory layout

Which?

Do we have any observation?

Which Memory layout

Hot data → row

Cold data → column

Which Memory layout

Hot data → row

Cold data → column

To do such data distribution, what should be done?

Which Memory layout

Record Query types?

Which Memory layout

Record Query types?

If we know what types of queries there are, then we definitely can make a good design.

Which Memory layout

Record Query types?

If we know what types of queries there are, then we definitely can make a good design.

But ...

Which Memory layout

Record Query types?

If we know what types of queries there are, then we definitely can make a good design.

But ...

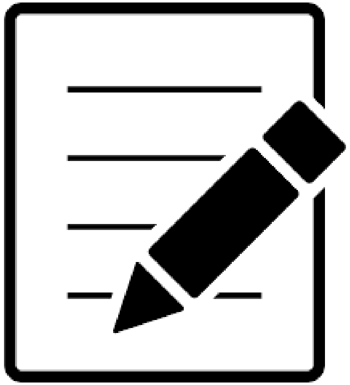
Can we be smarter?

Which Memory layout

Record Query types?

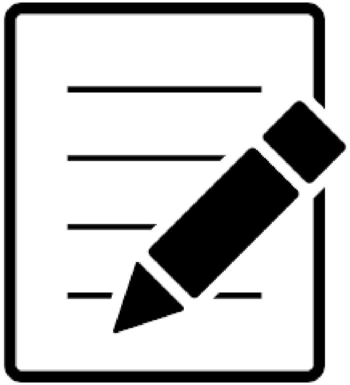
Self-adaptive?

Self-adaptive algorithm



Record

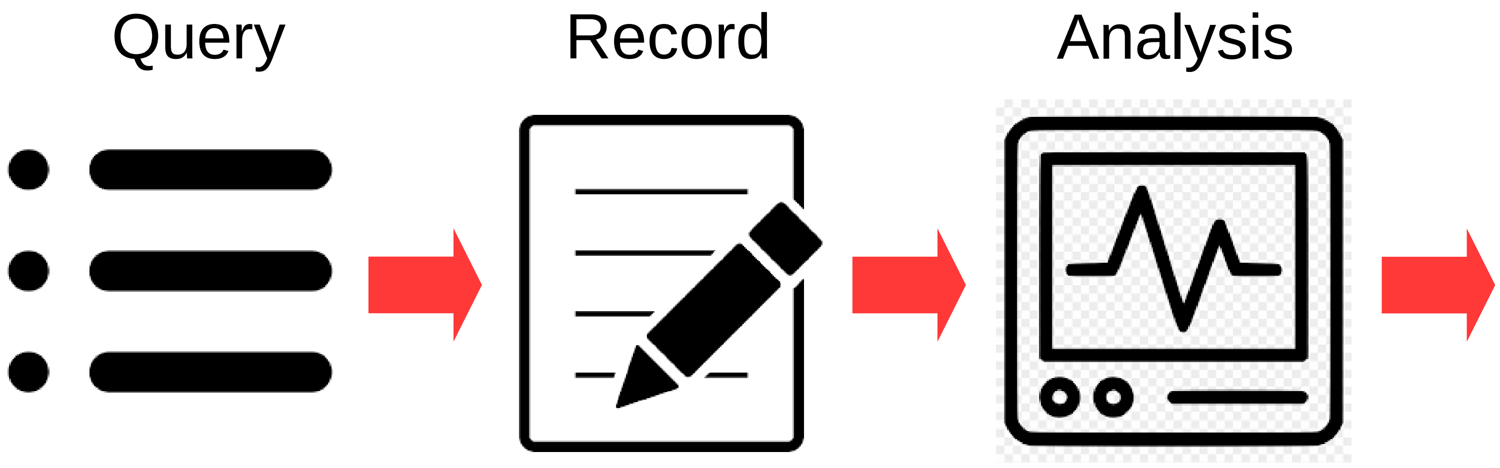
Self-adaptive algorithm



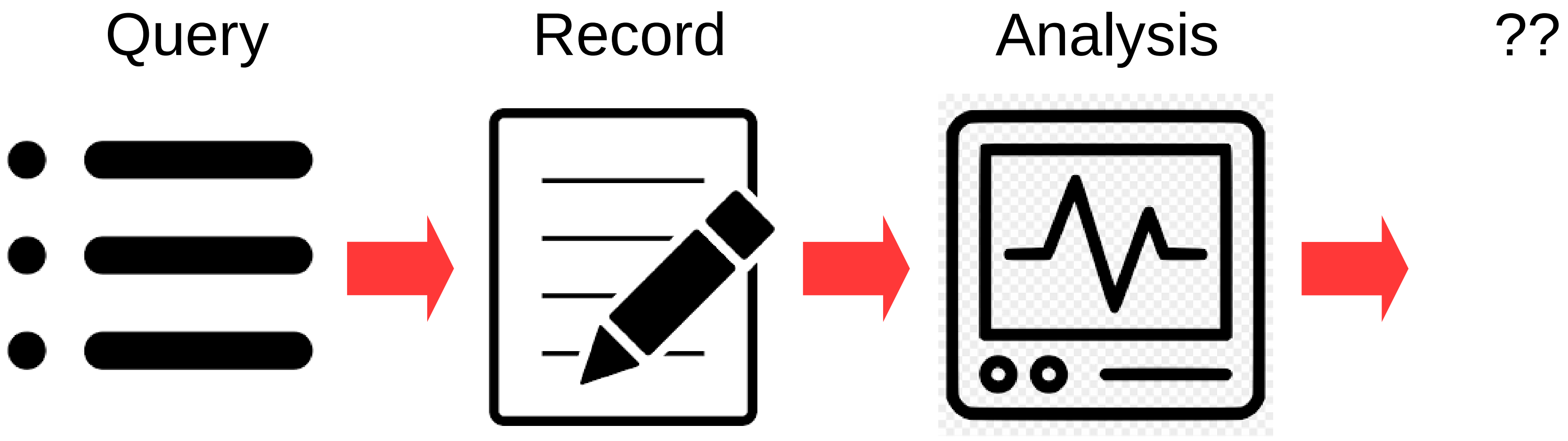
Record

Analysis

Self-adaptive algorithm

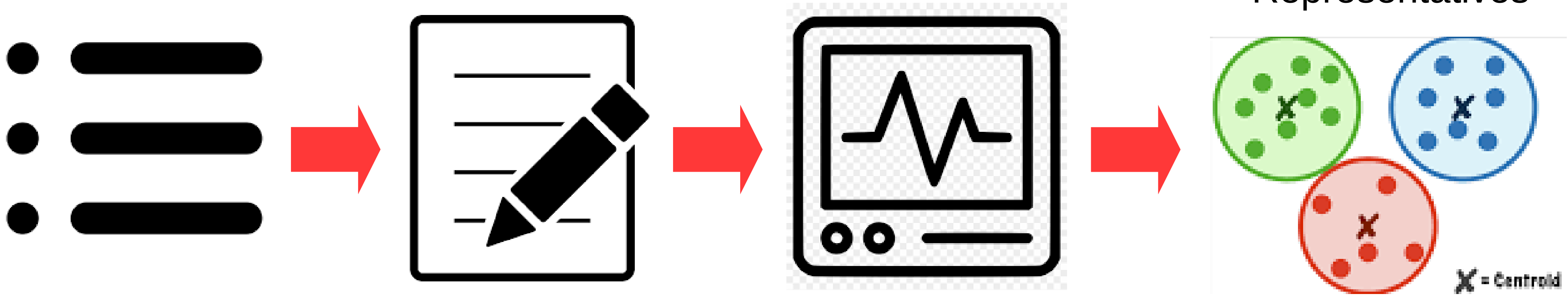


Self-adaptive algorithm



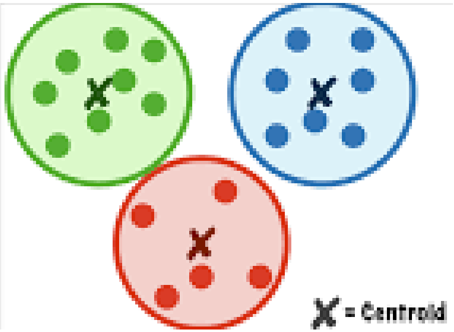
Self-adaptive algorithm

Query | Record | Analysis | Find Representatives



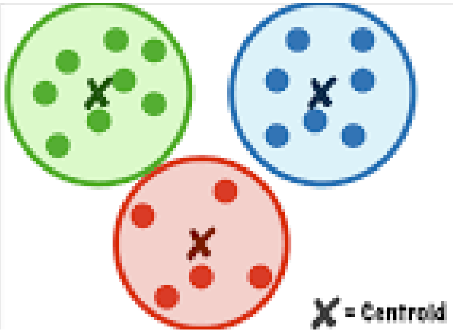
On-line k-means algorithm

Find
Representatives



On-line k-means algorithm

Find
Representatives



1^{st} step:

Using recent n Queries,

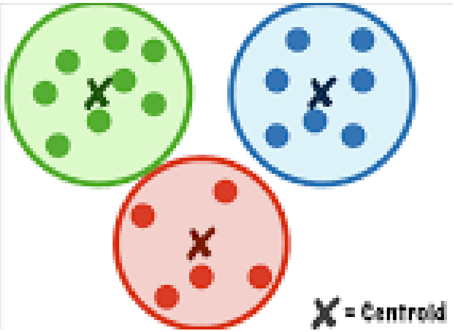
Find k Representatives R

2^{nd} step:

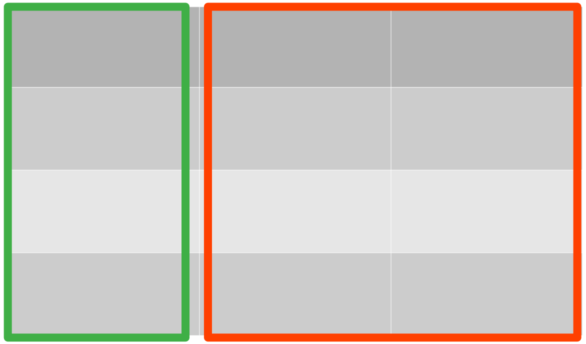
Generate vertical partitioned layout using R with greedy algorithm. (Largest cluster first)

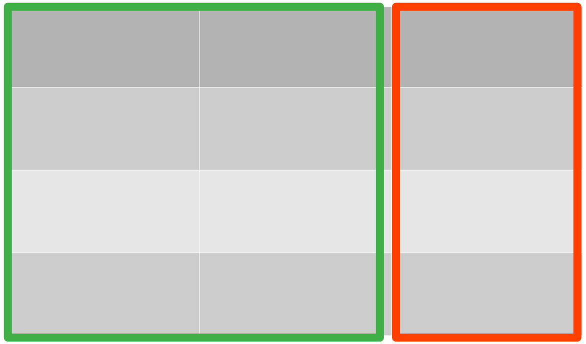
On-line k-means algorithm

Find
Representatives



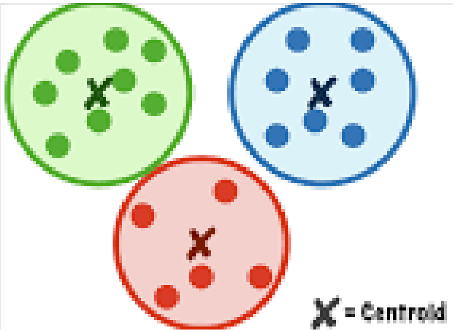
FSM (Flexible Storage Model)



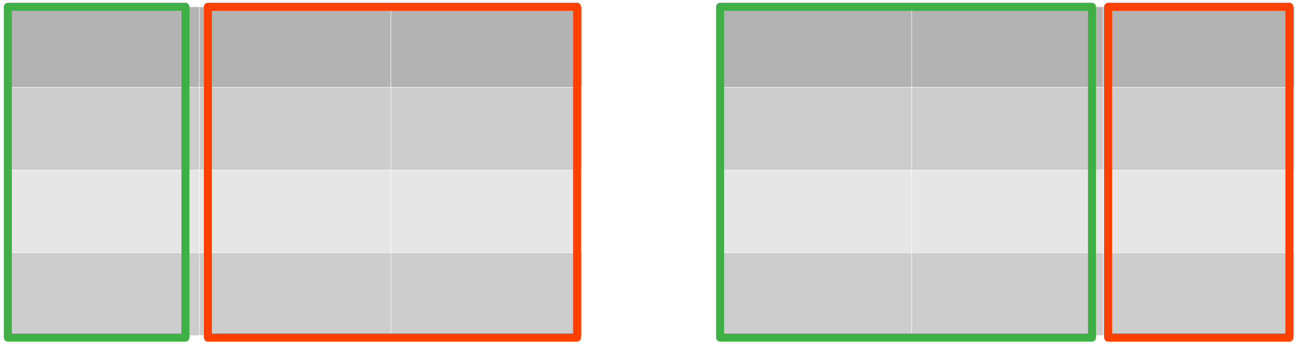


On-line k-means algorithm

Find
Representatives



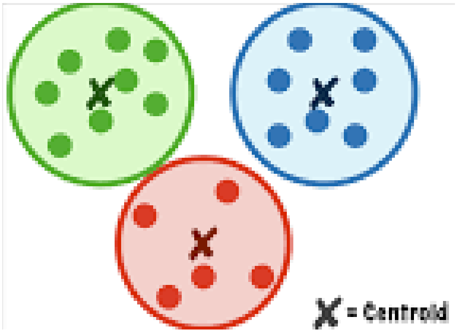
FSM (Flexible Storage Model)



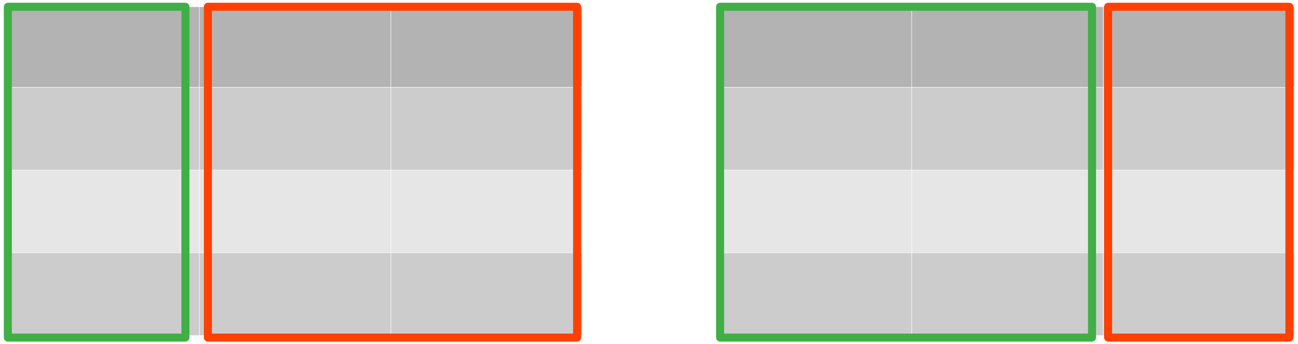
Different tables, different vertical layouts

On-line k-means algorithm

Find
Representatives



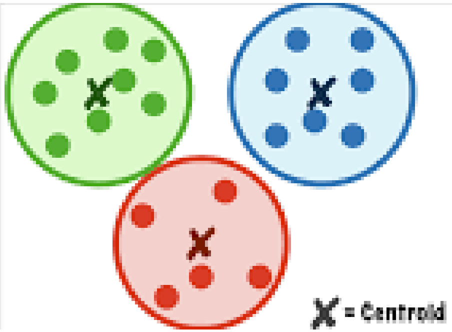
FSM (Flexible Storage Model)



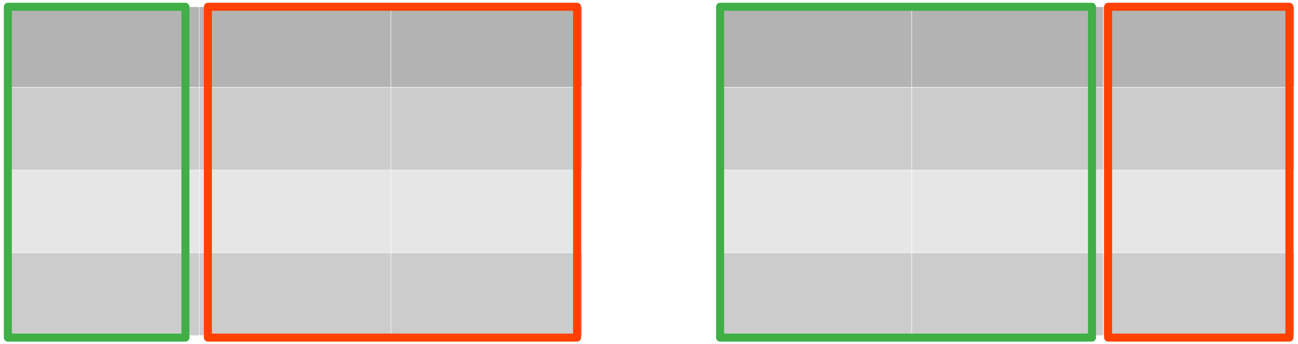
Different layouts → different access methods?

On-line k-means algorithm

Find
Representatives



FSM (Flexible Storage Model)

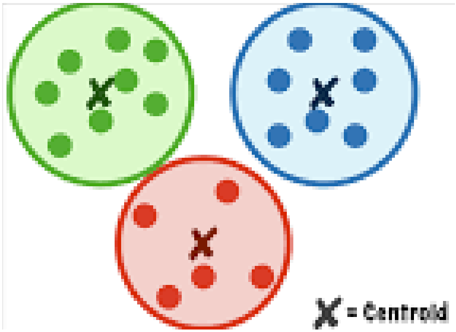


Different layouts → different access methods?

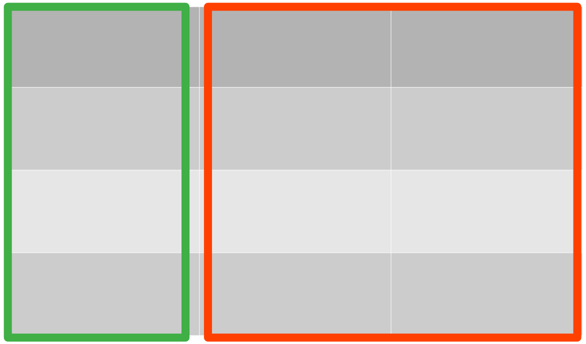
Inefficient !!

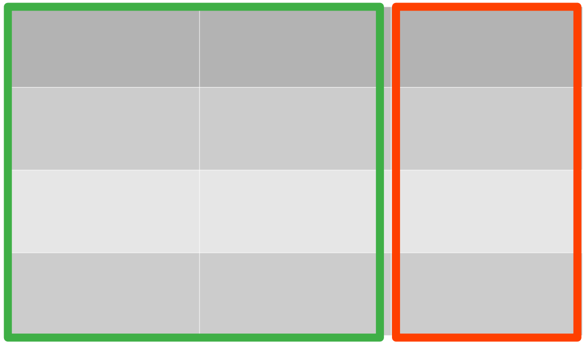
On-line k-means algorithm

Find
Representatives



FSM (Flexible Storage Model)

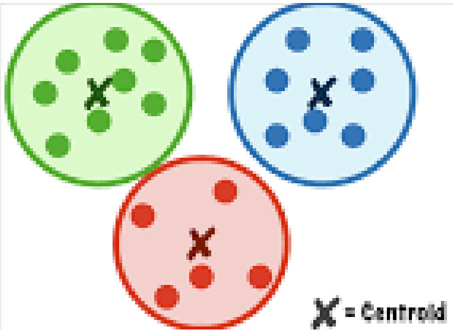




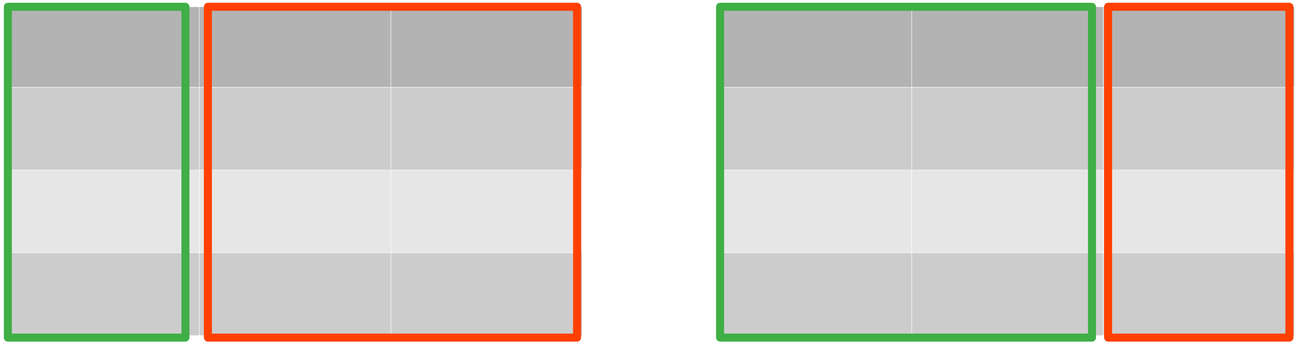
Provide Abstract layer !!

On-line k-means algorithm

Find
Representatives



FSM (Flexible Storage Model)



Provide Abstract layer !!

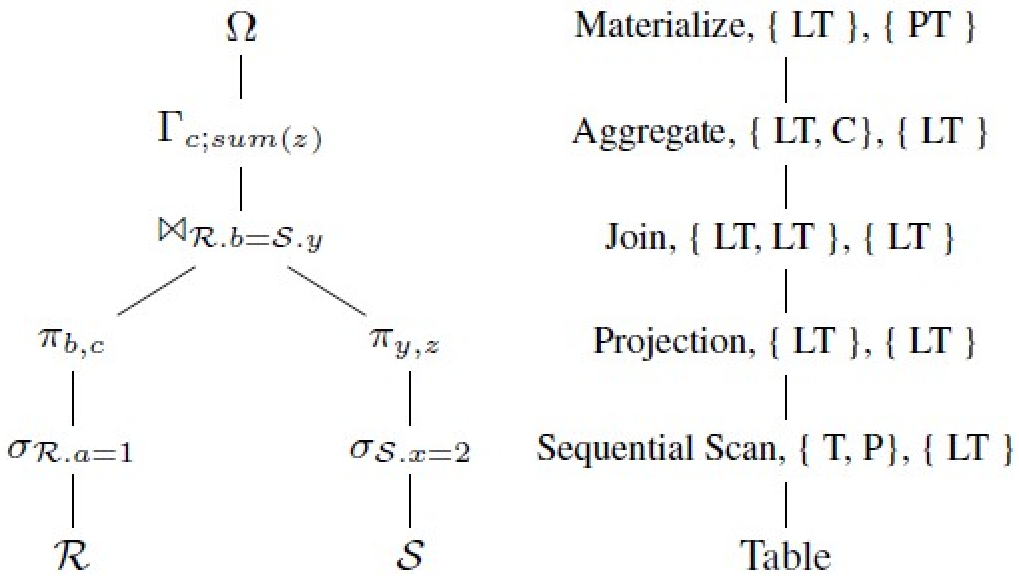
One mutual access interface.

Abstraction

```
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;
```

Abstraction

```
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;
```



LT : logical tile

PT : physical tile

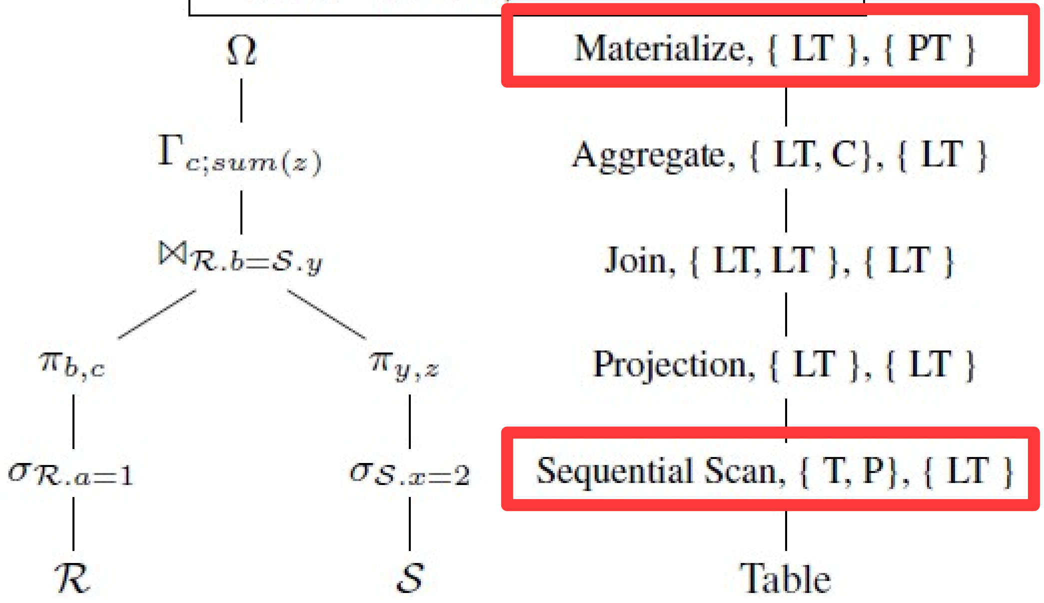
T : table

Attributes : C

Predicate : P

Abstraction

```
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;
```



LT : logical tile

PT : physical tile

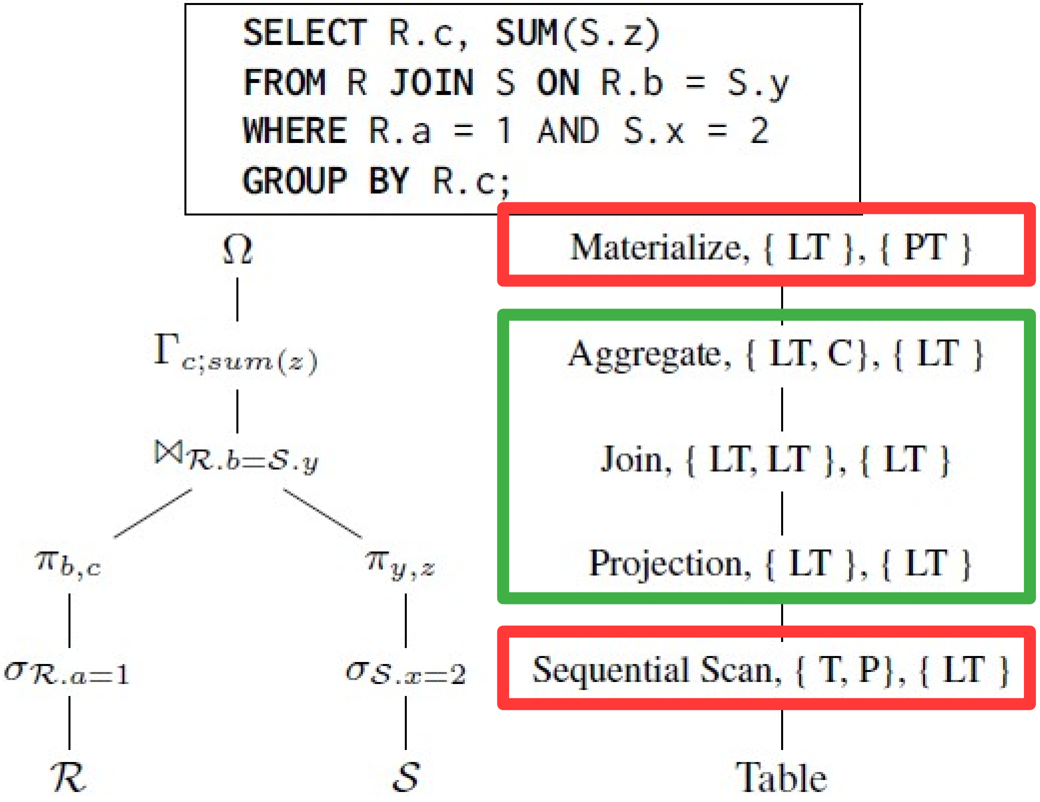
T : table

Attributes : C

Predicate : P

Physical data

Abstraction



LT : logic tile

PT : physical tile

T : table

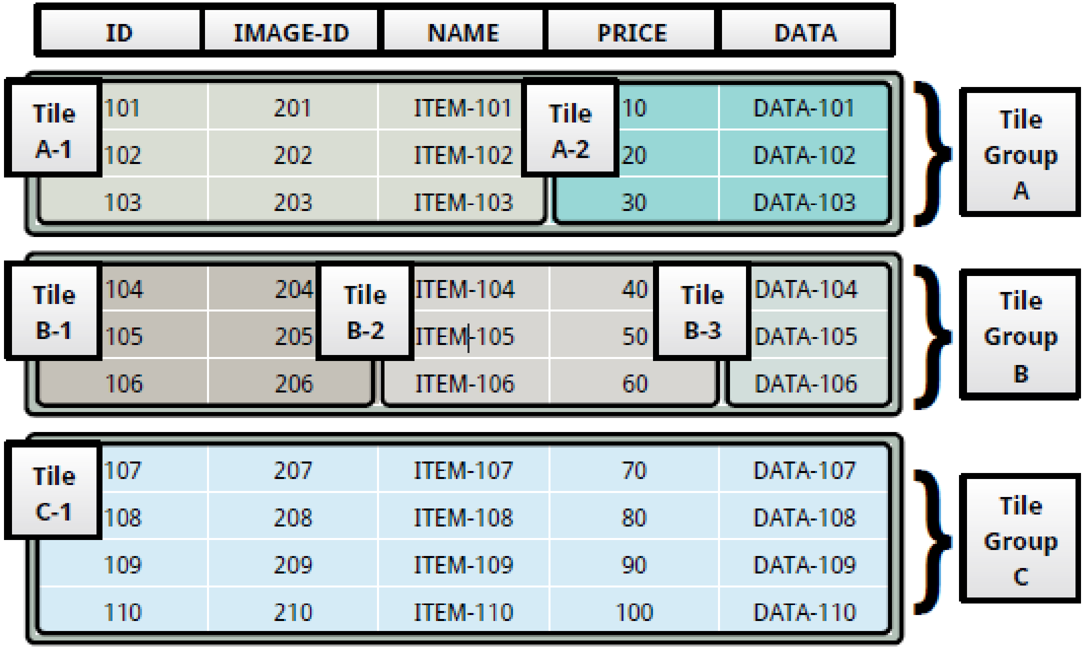
Attributes : C

Predicate : P

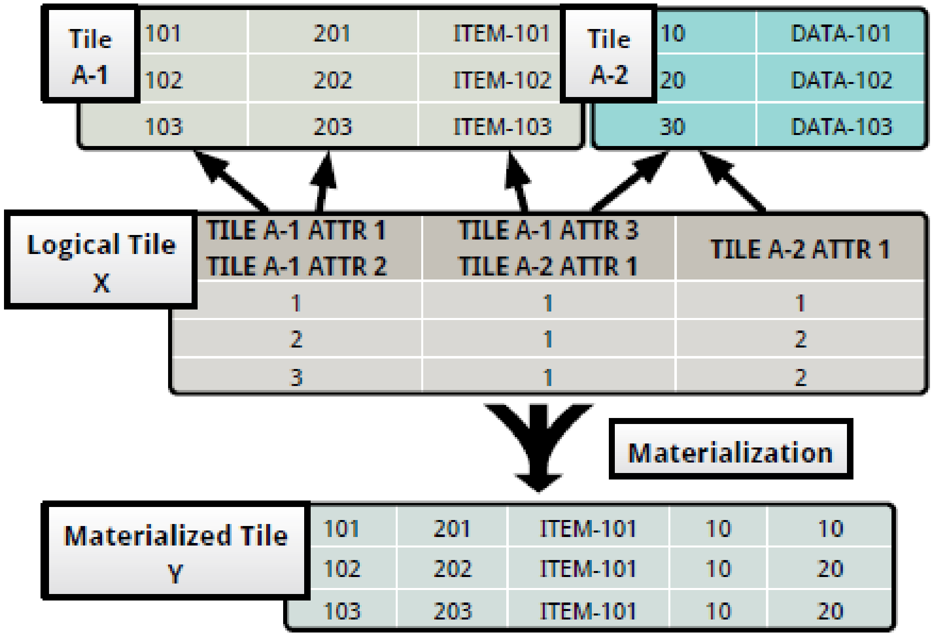
logical data +

Flexible materialization

Tiles



Physical Tile



Logical Tile

Parallelism

Parallelism

Definitely we do not want stalling.

Parallelism

Definitely we do not want stalling.

Write while reconfigure memory layout

Parallelism

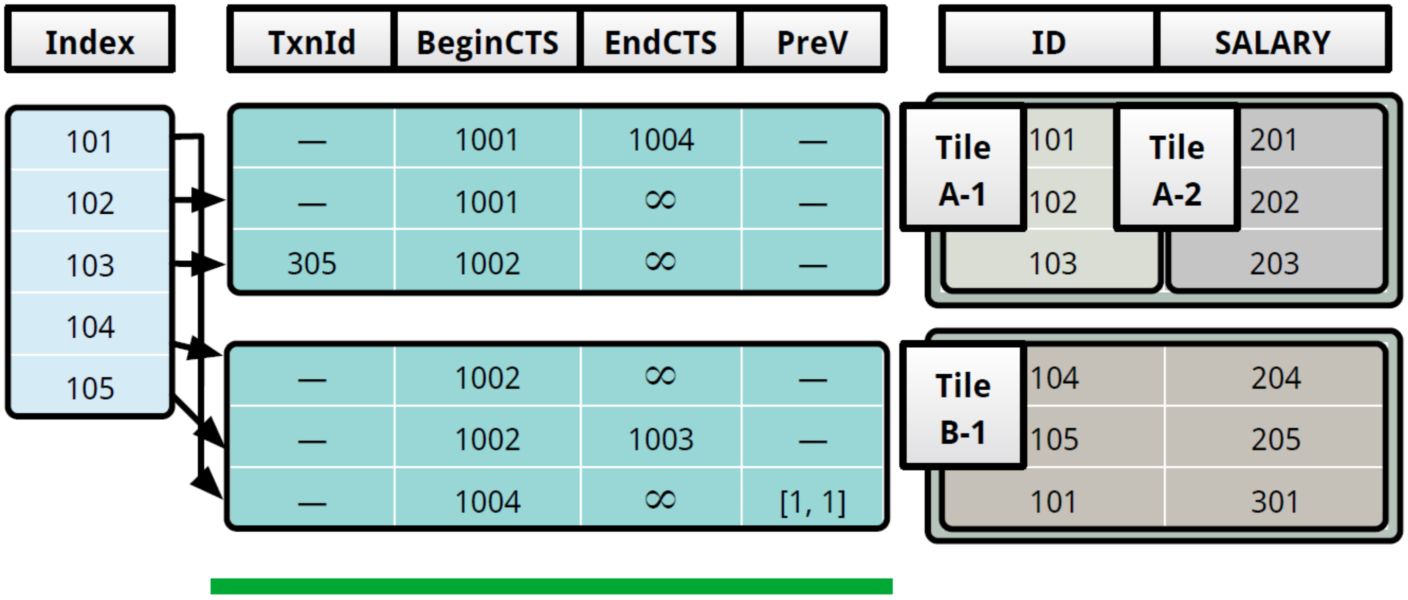
Definitely we do not want stalling.

Write while reconfigure memory layout

$\rightarrow$ MVCC

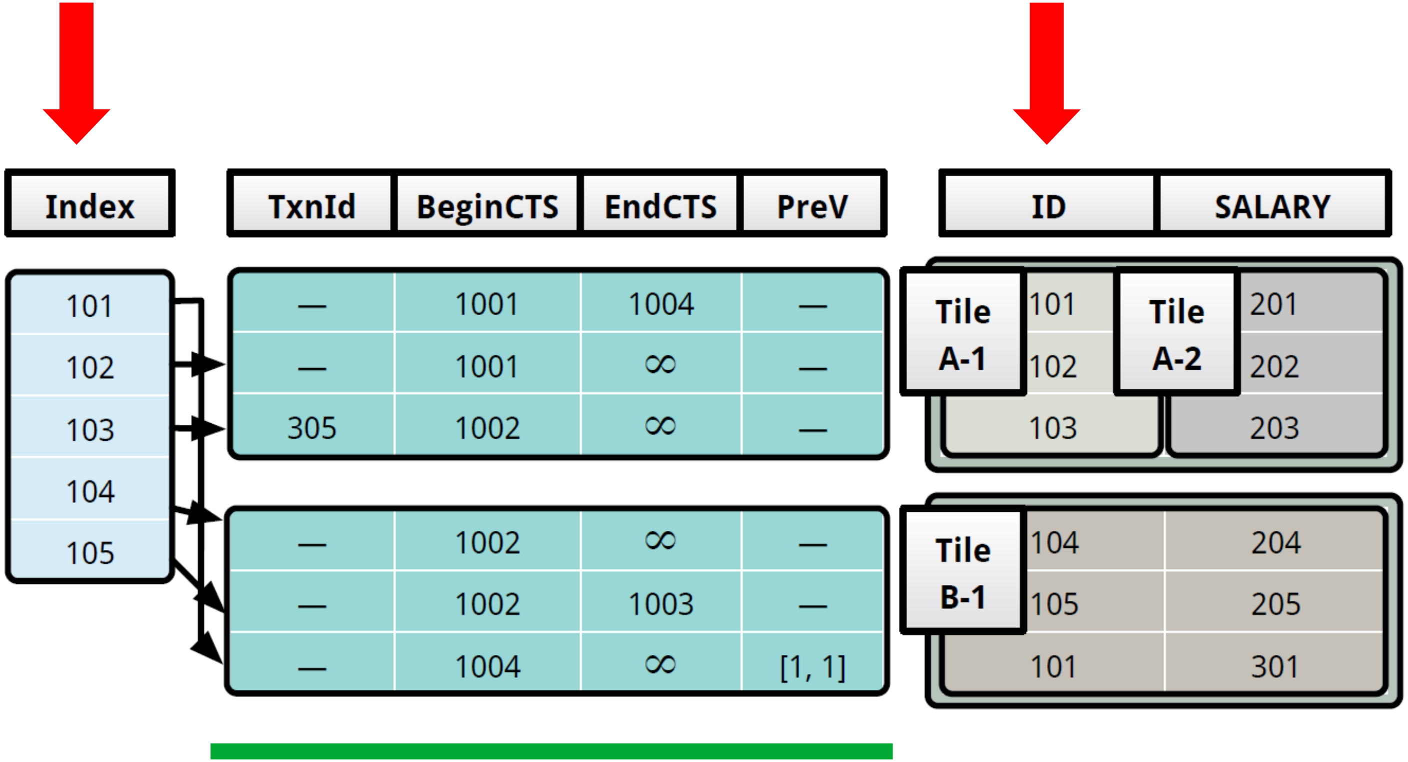
(Multi-Version Concurreny Control)

MVCC



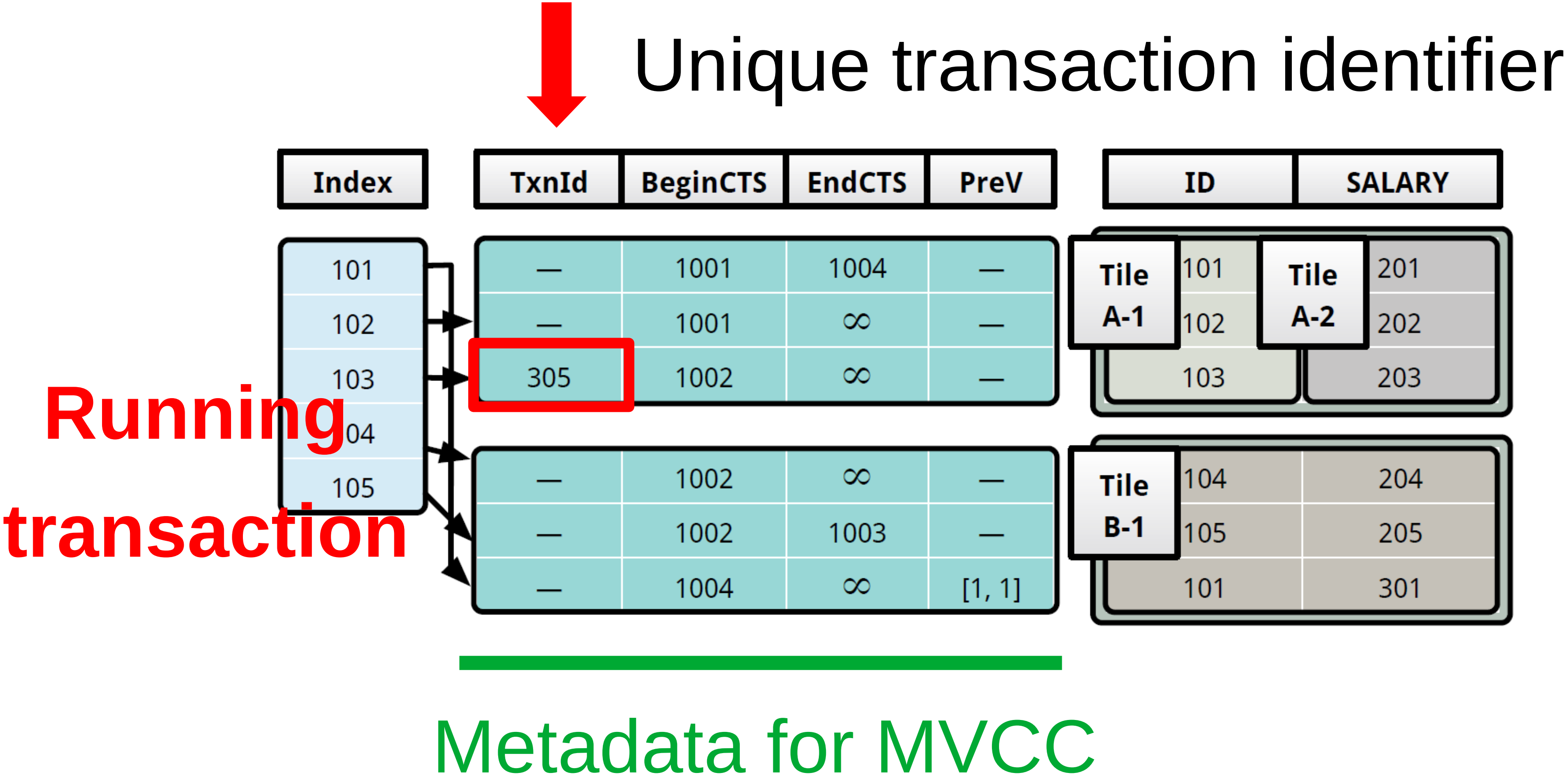
Metadata for MVCC

MVCC

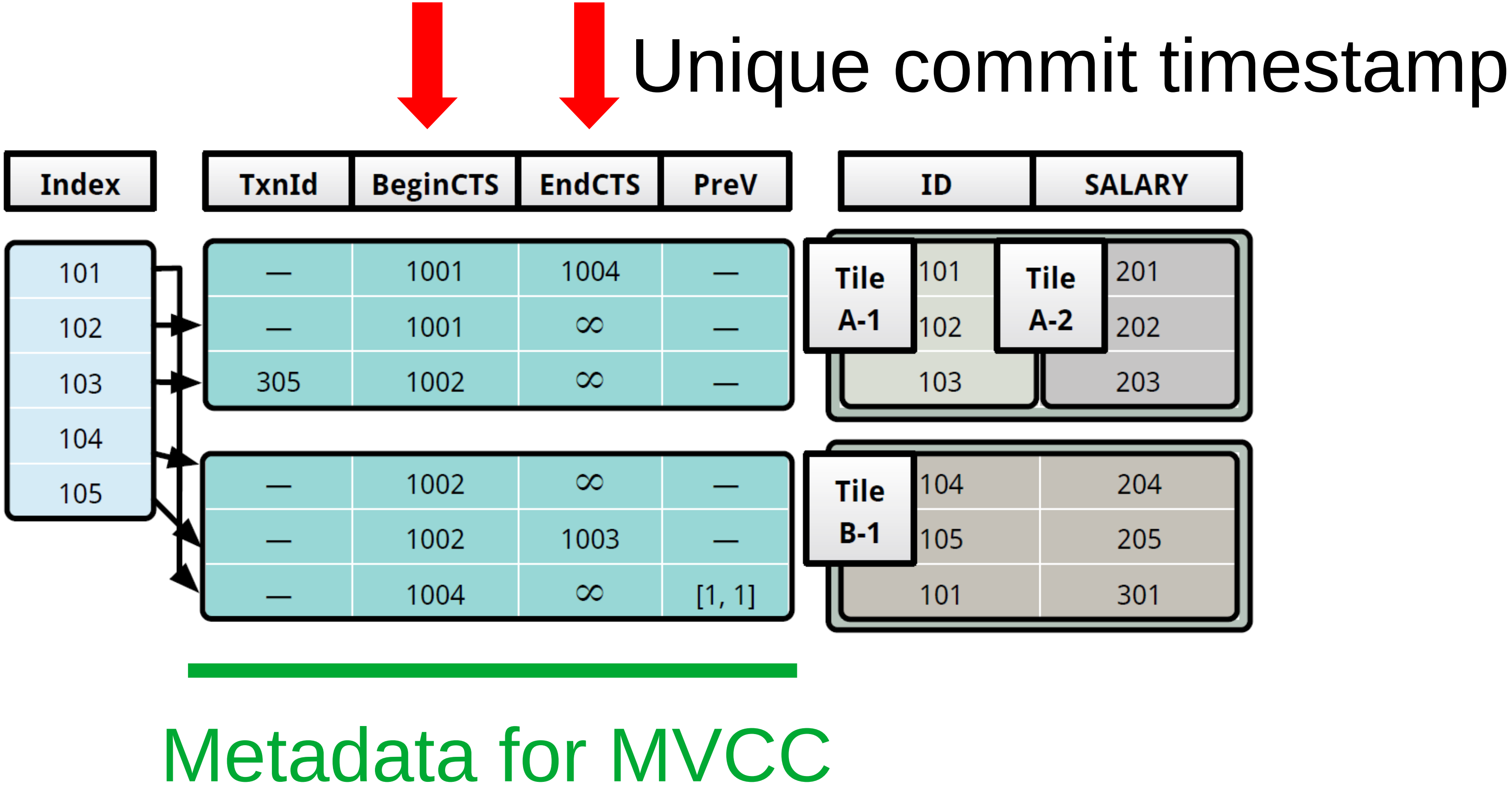


Metadata for MVCC

MVCC

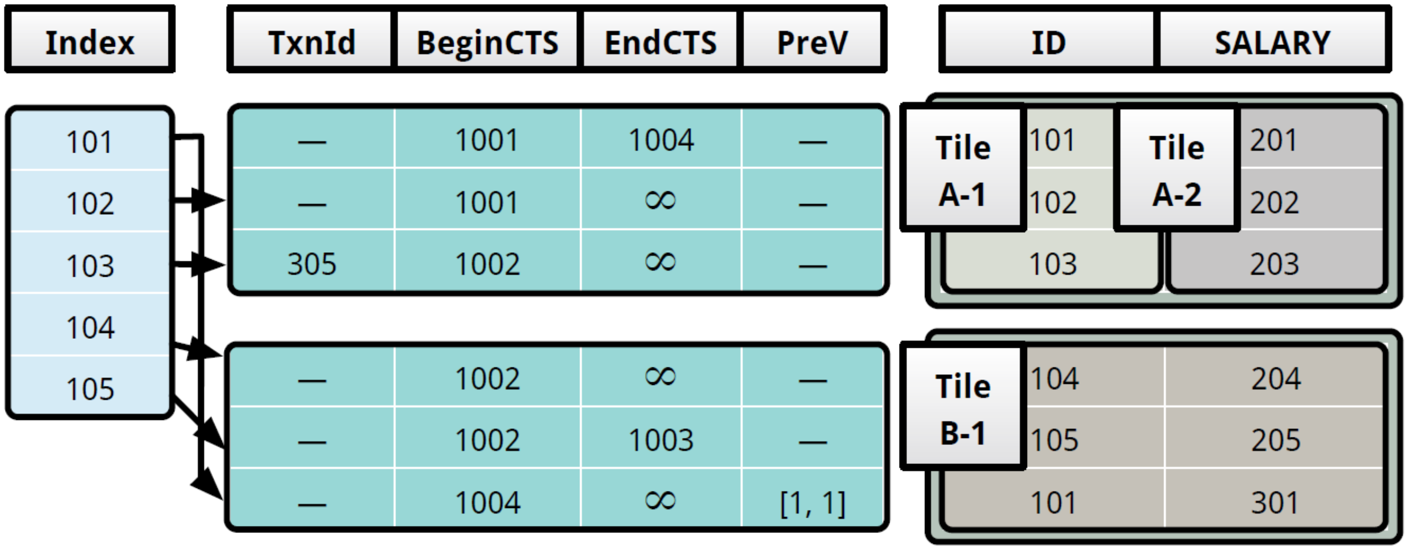

Unique transaction identifier
Index
TxnId
BeginCTS
EndCTS
PreV
ID
SALARY
101
102
103
104
105
—
1001
1004
—
—
1001
∞
—
305
1002
∞
—
—
1002
∞
—
—
1002
1003
—
—
1004
∞
[1, 1]
Tile A-1
101
102
103
Tile A-2
201
202
203
Tile B-1
104
105
101
204
205
301
Running transaction
Metadata for MVCC

MVCC


Unique commit timestamp
Index
TxnId
BeginCTS
EndCTS
PreV
ID
SALARY
101
102
103
104
105
—
305
1001
1002
1003
1004
∞
[1, 1]
Tile A-1
Tile A-2
Tile B-1
201
202
203
204
205
301
Metadata for MVCC

MVCC

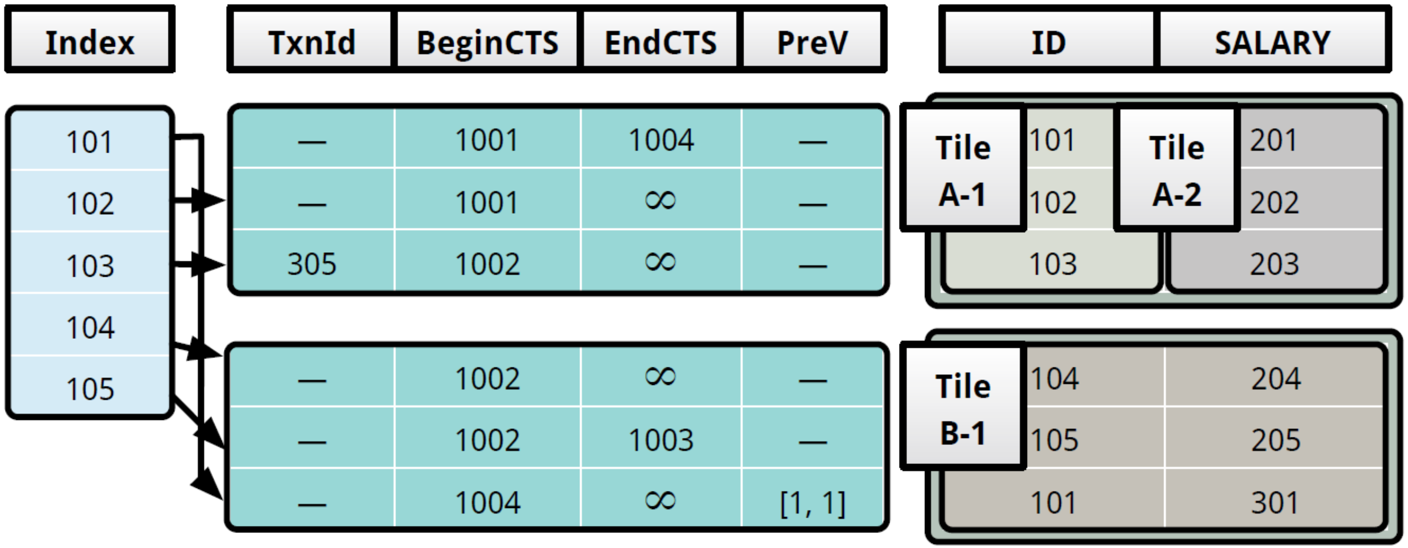
i) Insert (atomic)



Metadata for MVCC

MVCC

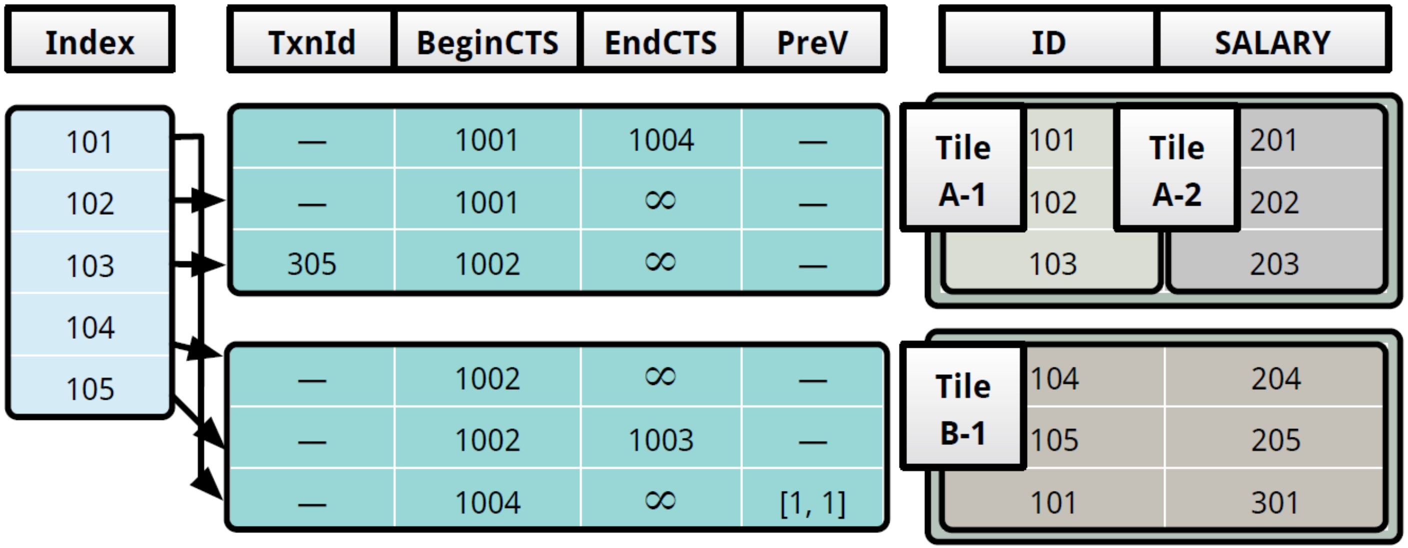
i) Insert (after commit)



Metadata for MVCC

MVCC

ii) Delete (atomic)

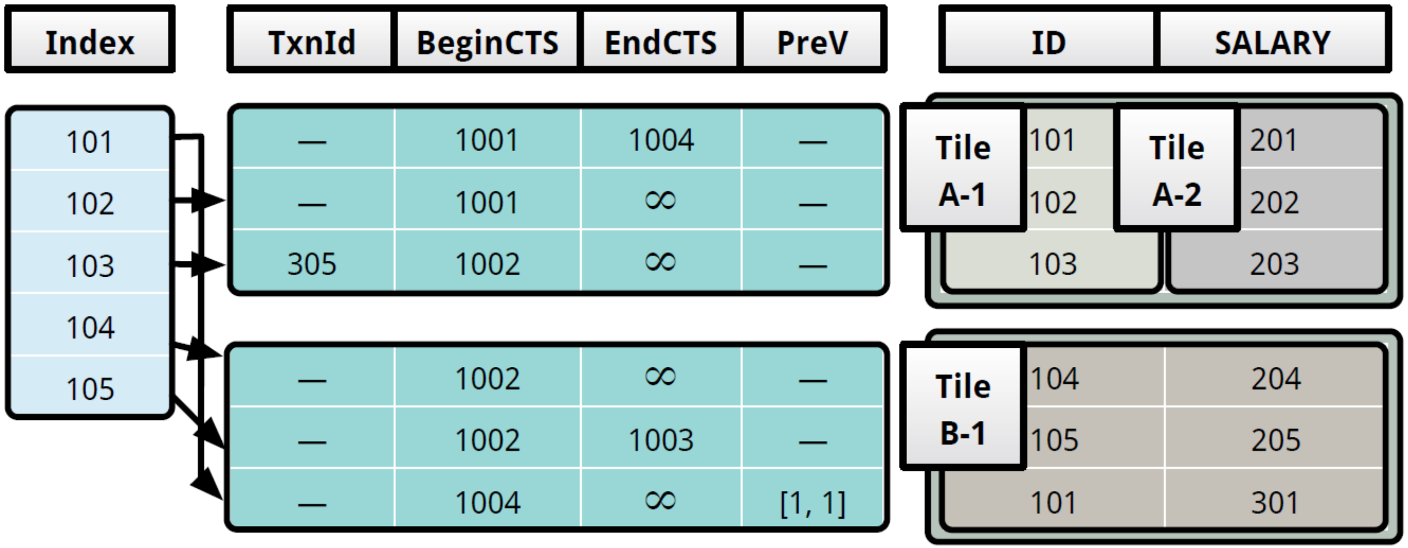


Metadata for MVCC

MVCC

ii) Delete

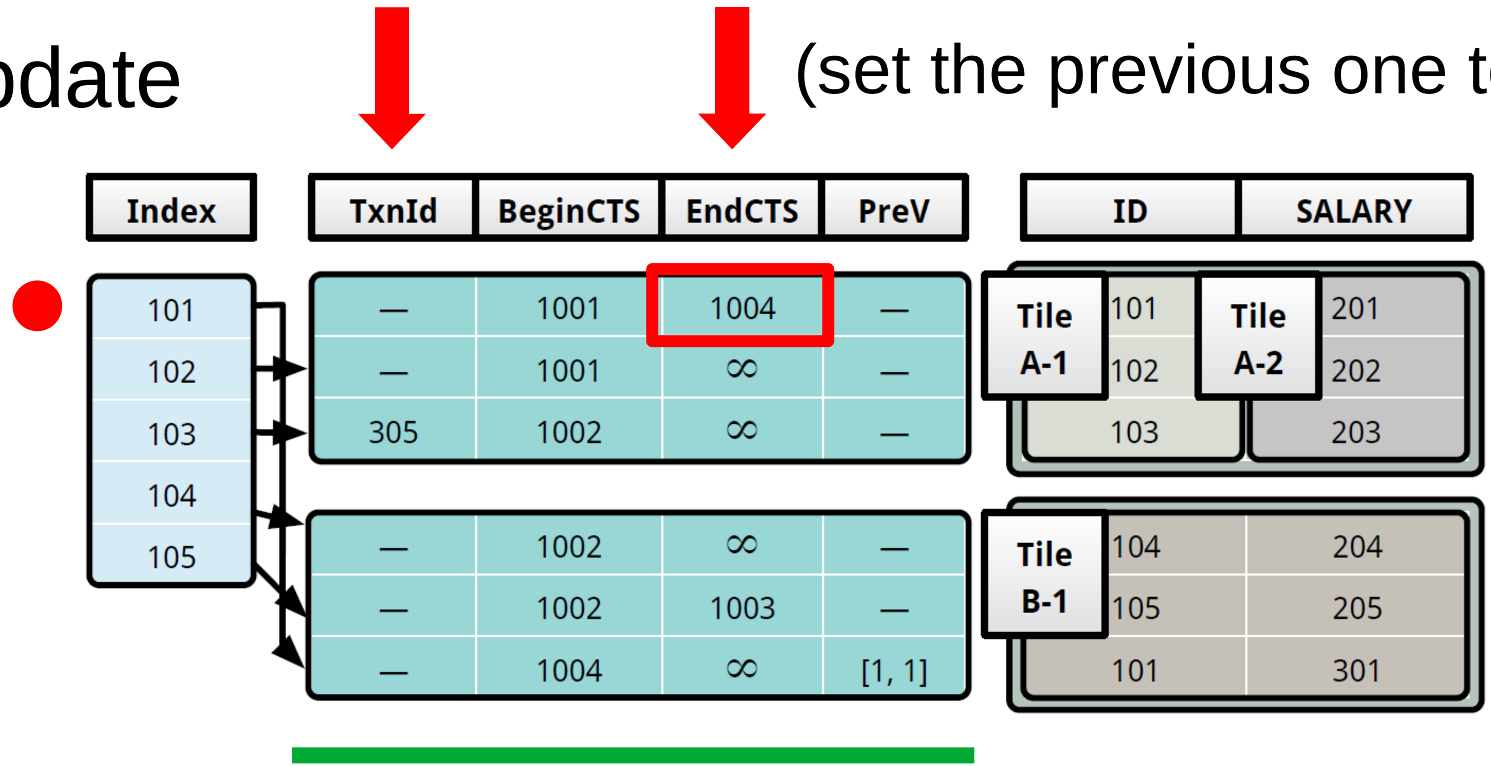
(after commit)



Metadata for MVCC

MVCC

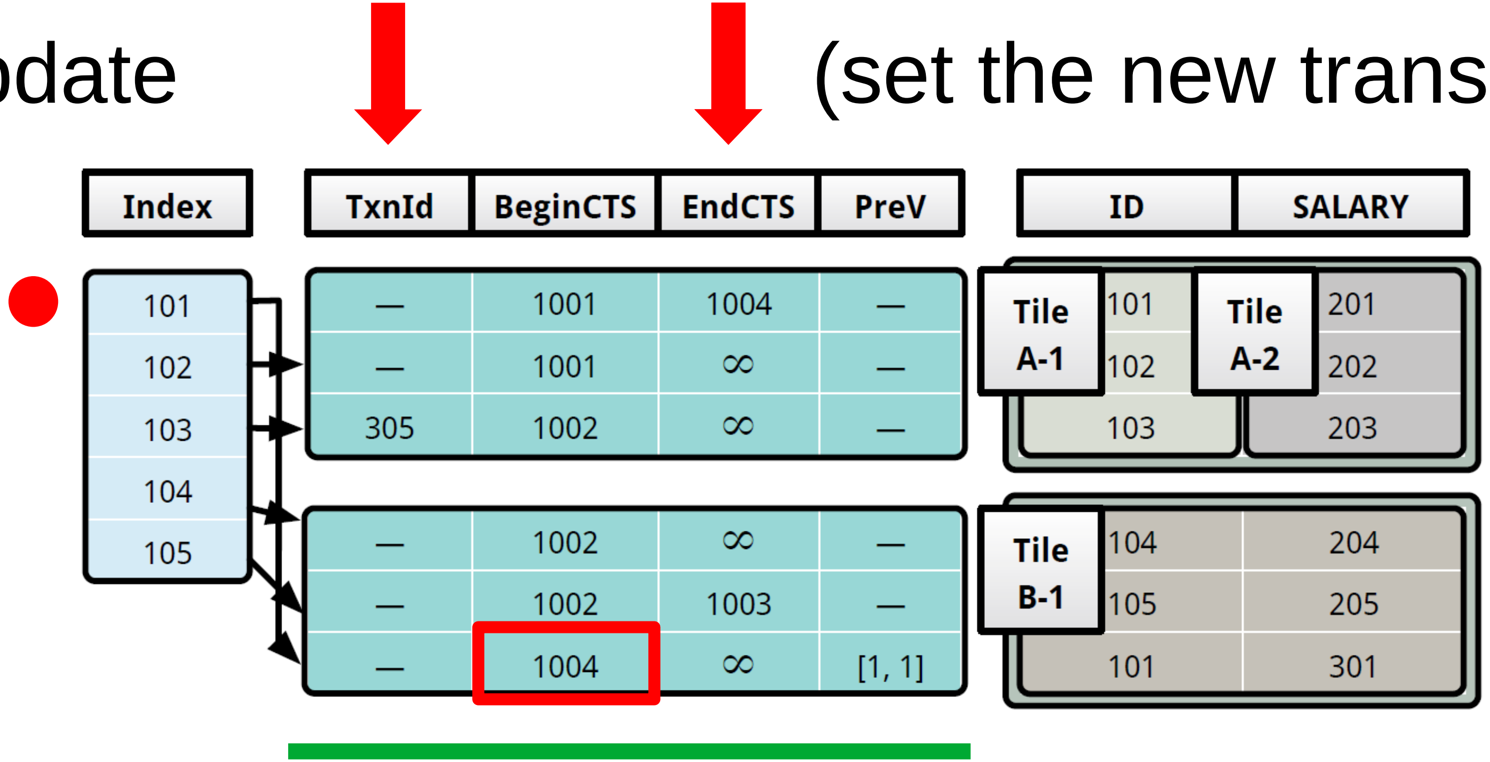
iii) Update (set the previous one to invalid)



Metadata for MVCC

MVCC

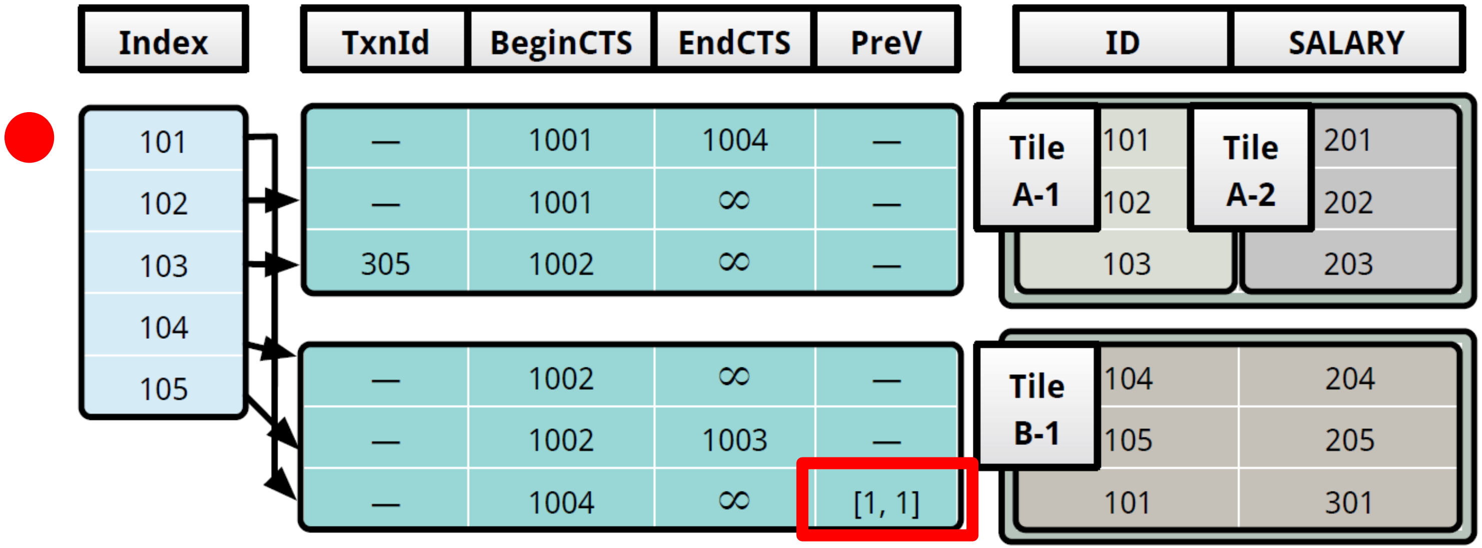
iii) Update (set the new transaction)



Metadata for MVCC

MVCC

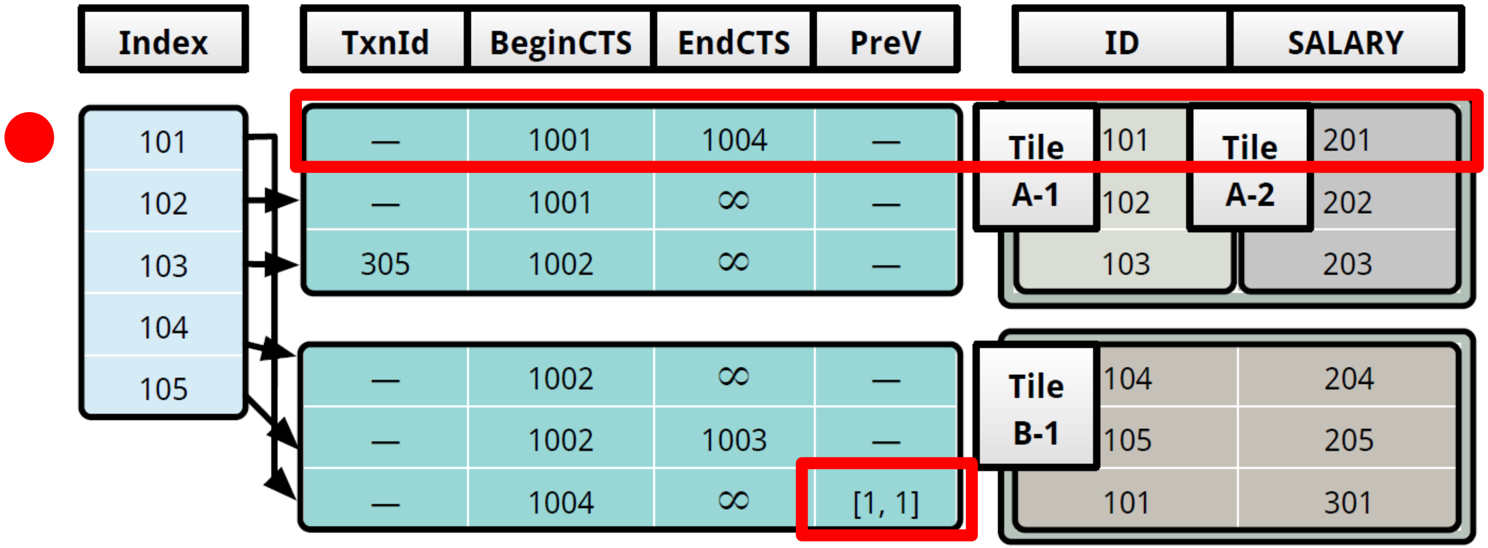
iii) Update (point to the previous one)



Metadata for MVCC

MVCC

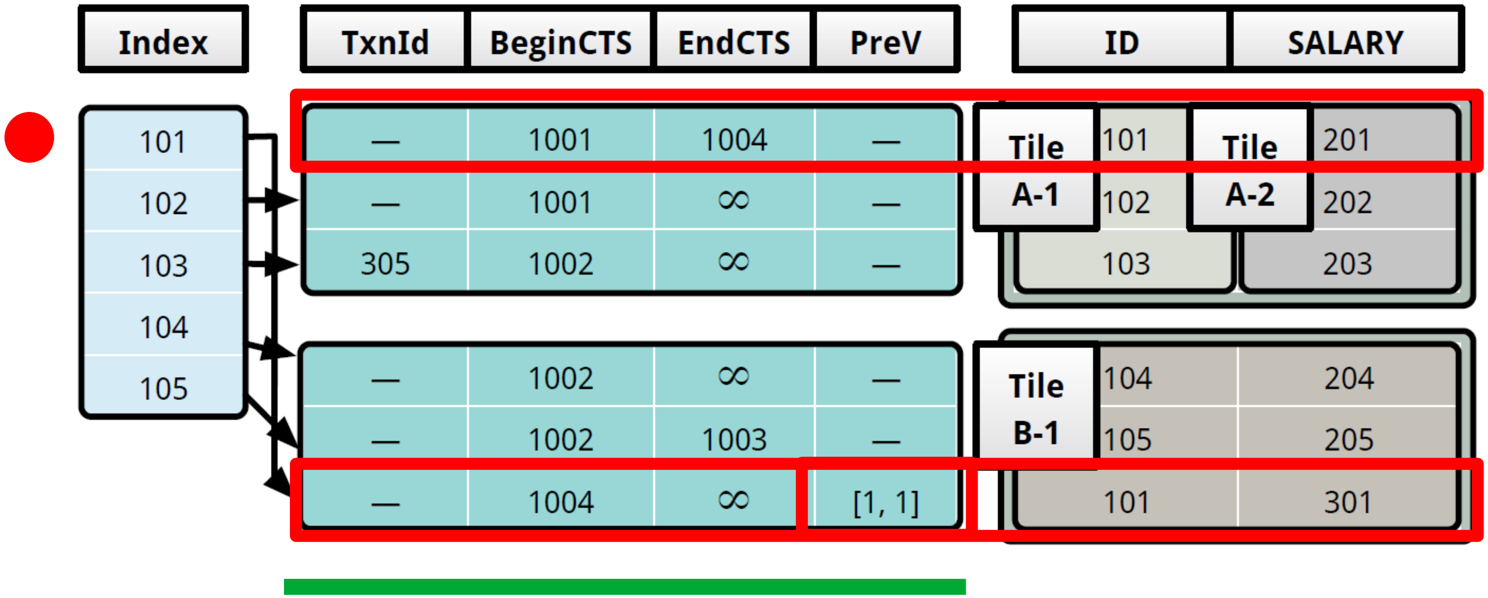
iii) Update (point to the previous one)



Metadata for MVCC

MVCC

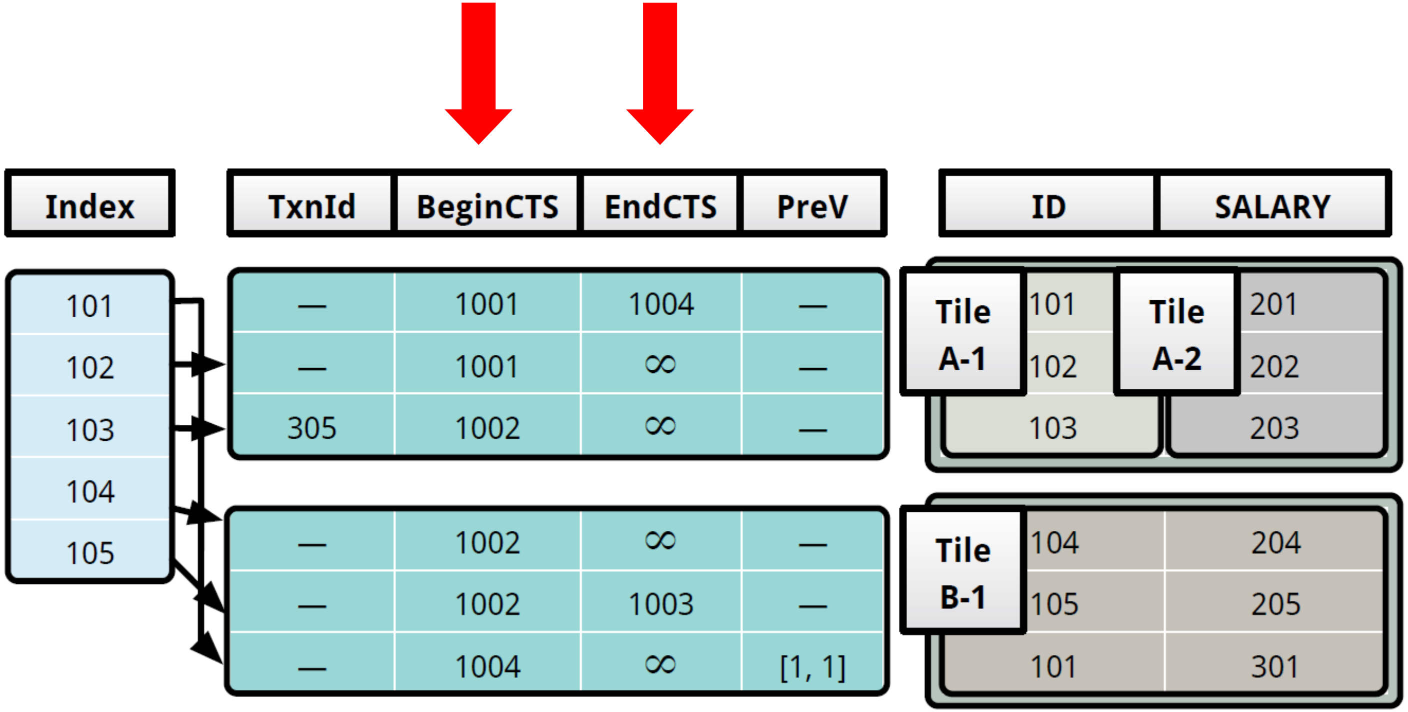
iii) Update (point to the previous one)



Metadata for MVCC

MVCC

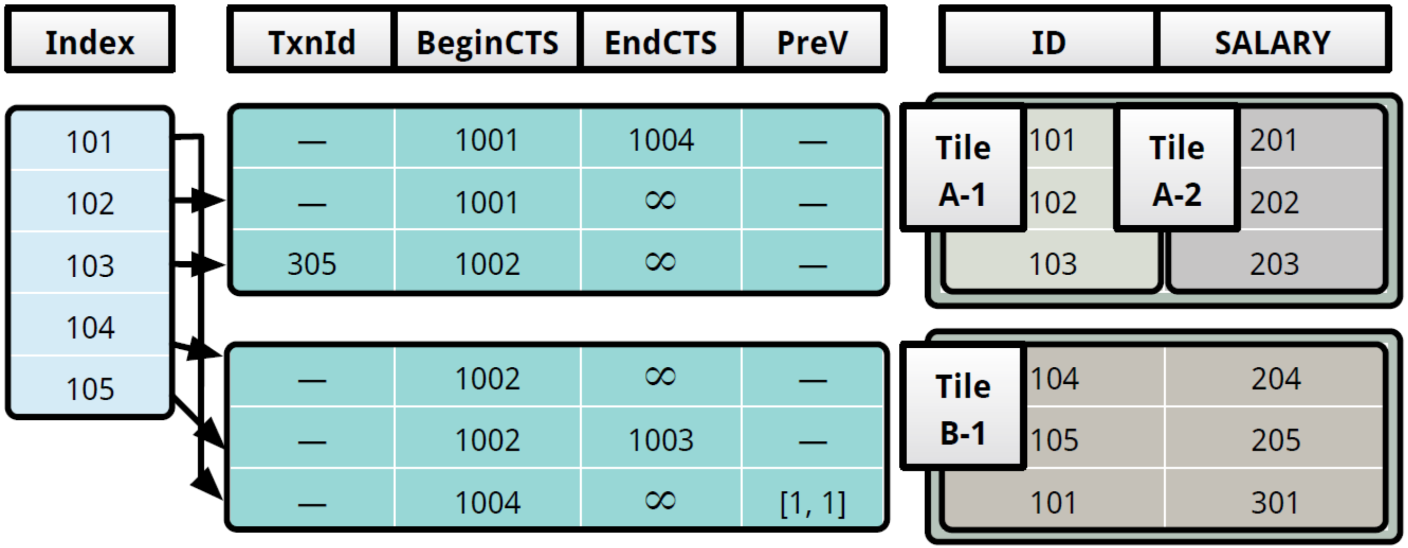
Search



Looking through the metadata, find the one within

Its visibility (BeginCTS < transaction ID < EndCTS)

MVCC



Garbage Collector claims those old spaces back

Performance

What about the overhead?

Is this method practical?

Performance

What about the overhead?

Is this method practical?

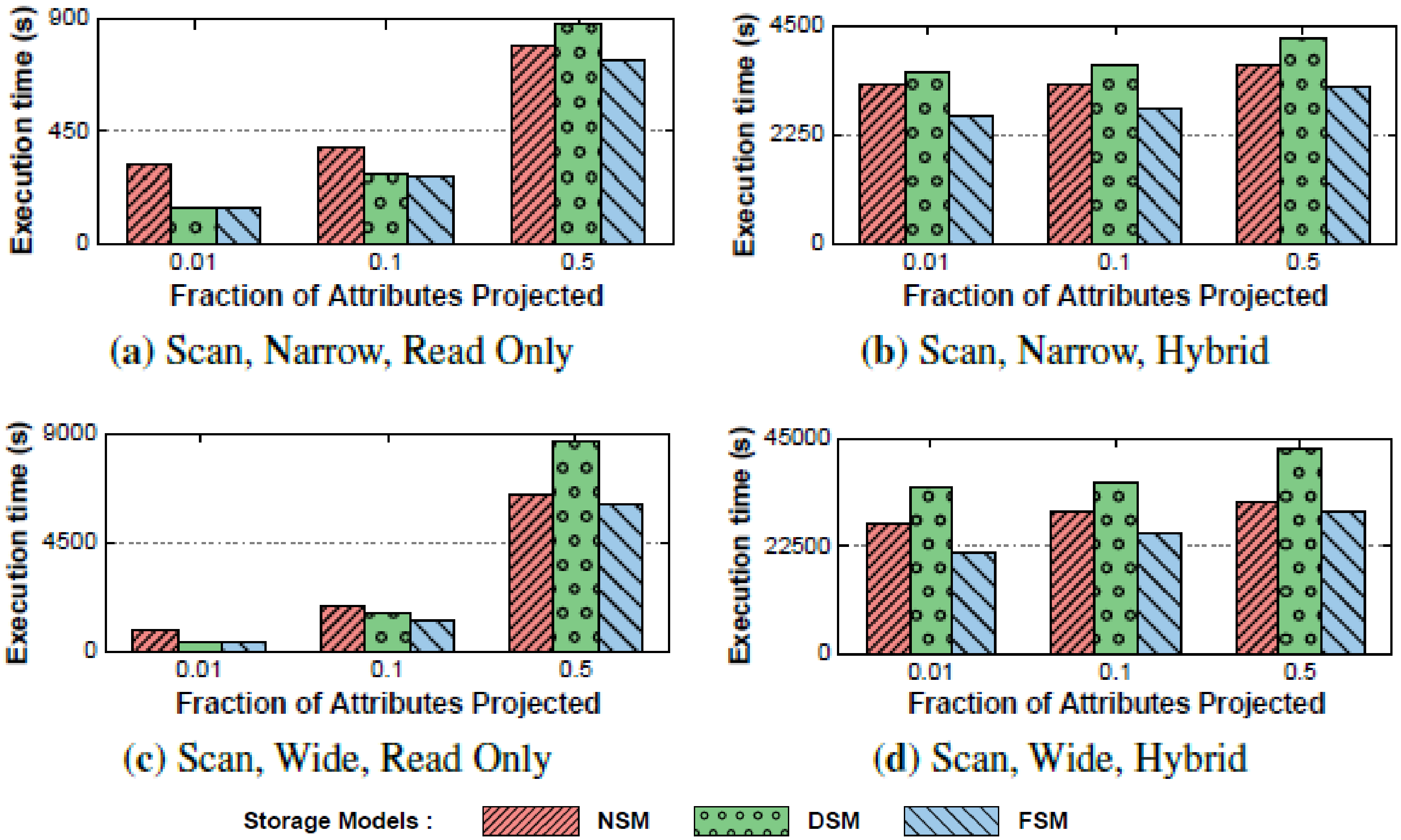
Let us see the results.

Evaluation

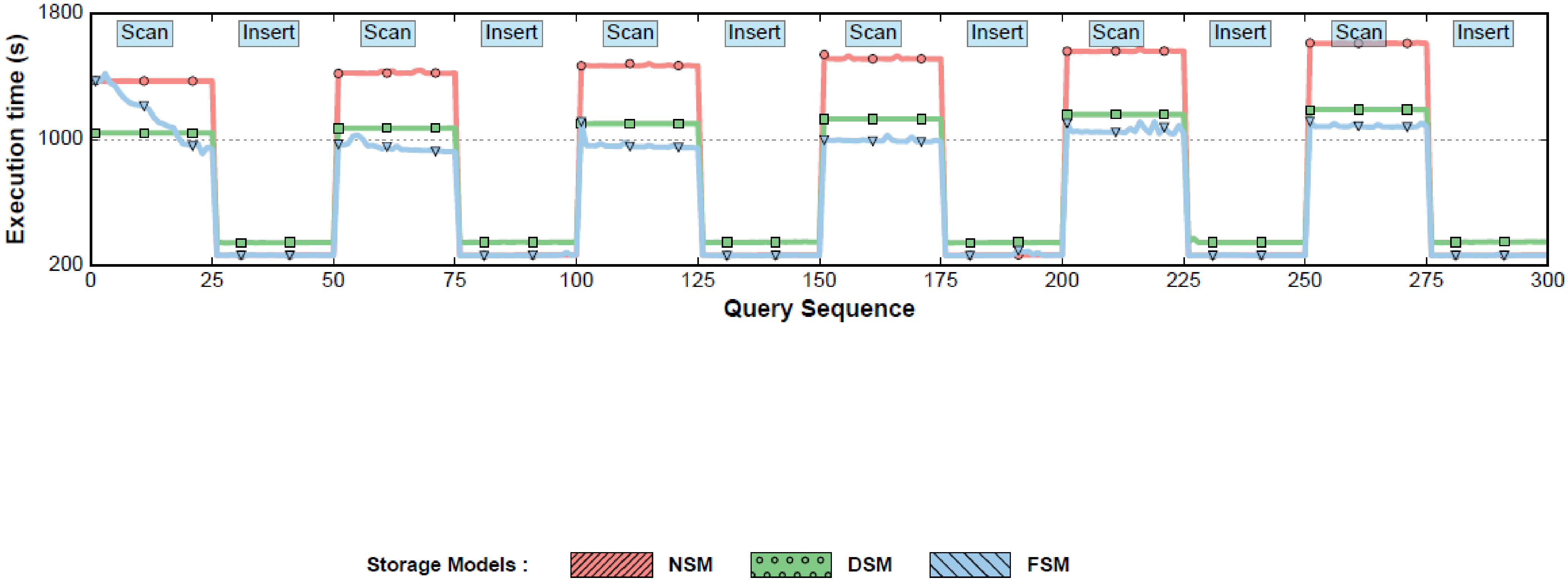
Experiemnt settings

- CPU: dual-socket Intel Xeon E5-4620 server
- OP: Ubuntu 14.04 (64-bit)
- Core: 2.6 GHz * 8 / socket
- DRAM: 128 GB
- L3 cache size: 20 MB

NSM / DSM / FSM SCAN



Adaptation



Scan
Insert
Scan
Insert
Scan
Insert
Scan
Insert
Scan
Insert
Scan
Insert
1800
1000
200
Execution time (s)
0
25
50
75
100
125
150
175
200
225
250
275
300
Query Sequence
Storage Models :
NSM
DSM
FSM

Conclusion

Conclusion

- Memory Layout (column storage, row storage)
- + Adaptation → FSM
- + Abstraction → Logical Tiles
- + Concurrency control → MVCC

Discussion

Discussion

- When the reorganization benefits the queries and whet it does not?

Discussion

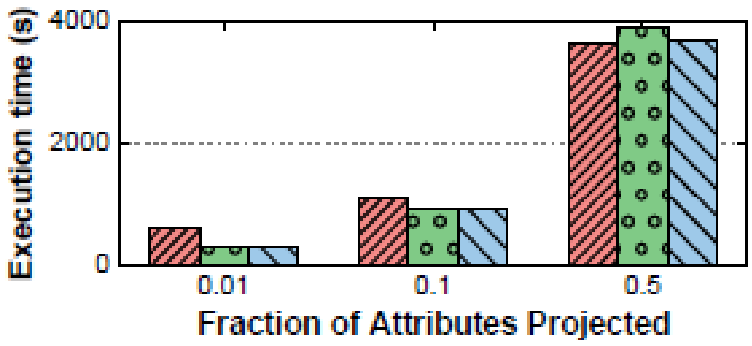
- When the reorganization benefits the queries and whet it does not?
- Is k-means a good algorithm for the layout reorganization task?

END

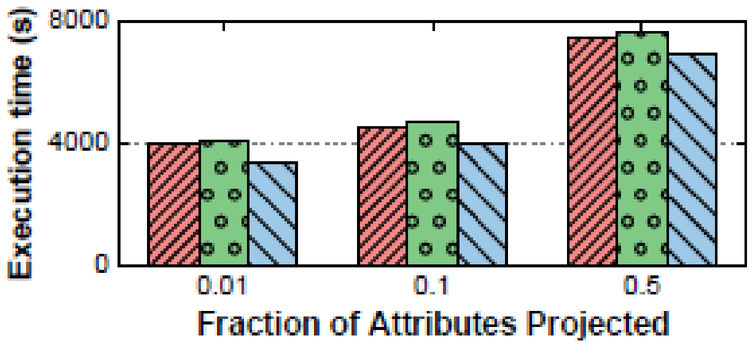
- 1 beginning, main contribution of the paper
- 2. before conclusion, whether it supports its idea
- 3. Conclude, open question
- 4.Experiement settings
- 5.mentioned NSM, DSM in the beginning part

  6. k-means → not really finding the right clusters (CS565)
- 7. Asking questions

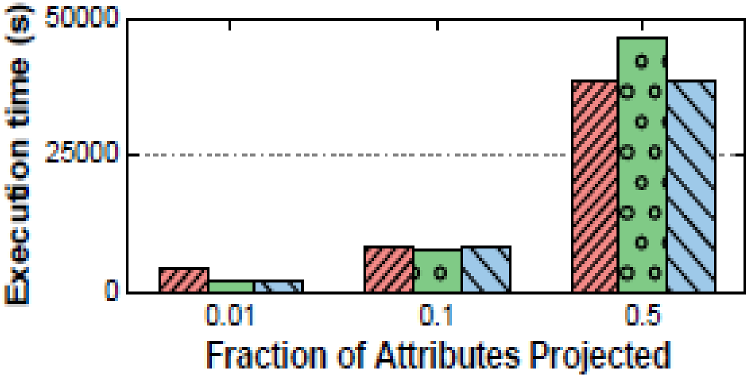
NSM / DSM / FSM Aggregate



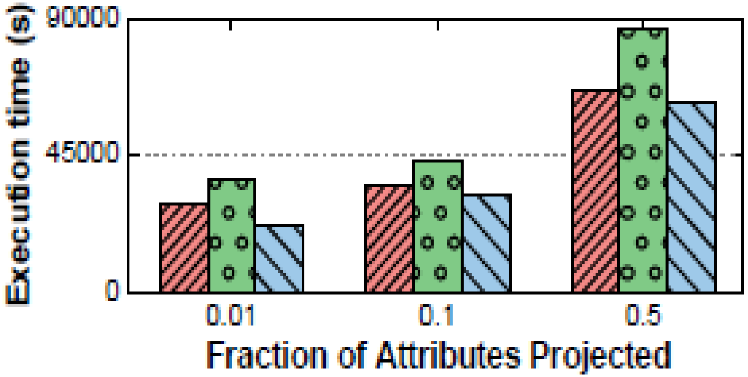
(e) Aggregate, Narrow, Read Only



(f) Aggregate, Narrow, Hybrid



(g) Aggregate, Wide, Read Only



(h) Aggregate, Wide, Hybrid

Horizontal Separation

(a) Scan, Narrow, Read Only

(b) Scan, Narrow, Hybrid

(c) Scan, Wide, Read Only

(d) Scan, Wide, Hybrid