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

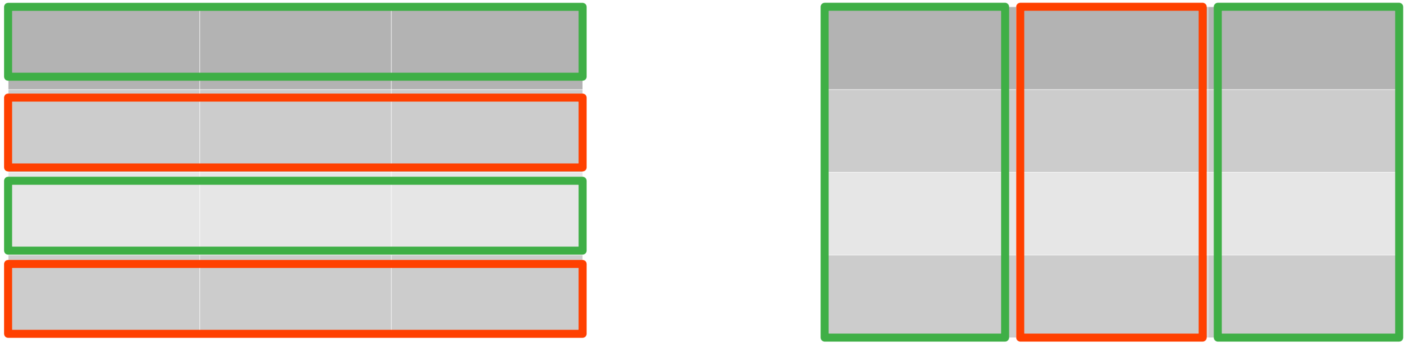
J Arulraj, et el.

SIGMOD ’20

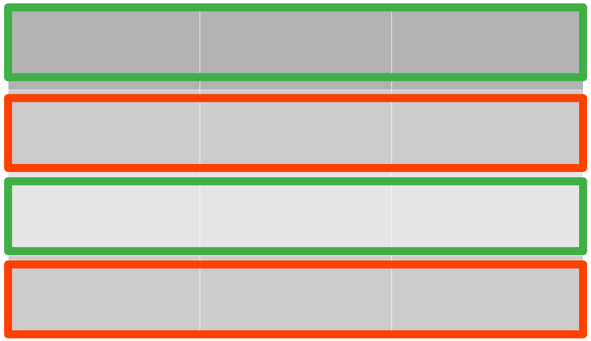
Speaker:

Introduction

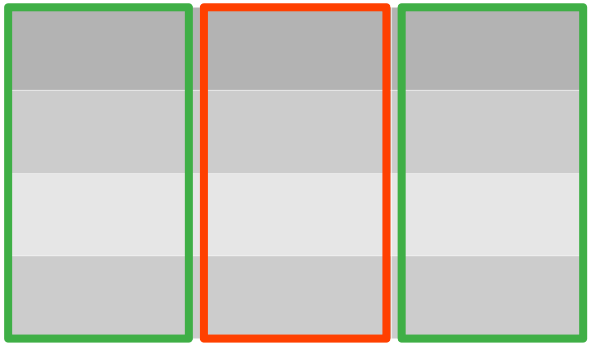
Physical Memory layout



Physical Memory layout

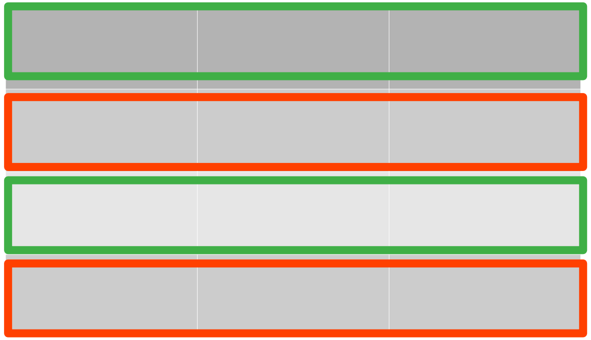


?



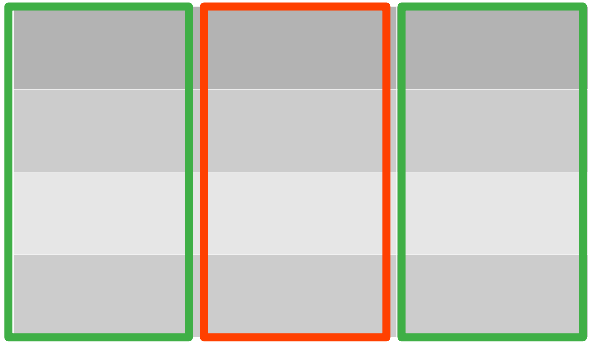
?

Physical Memory layout



Row Storage

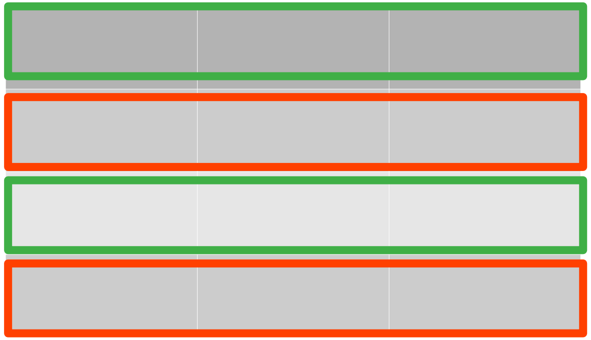
tuple-centric



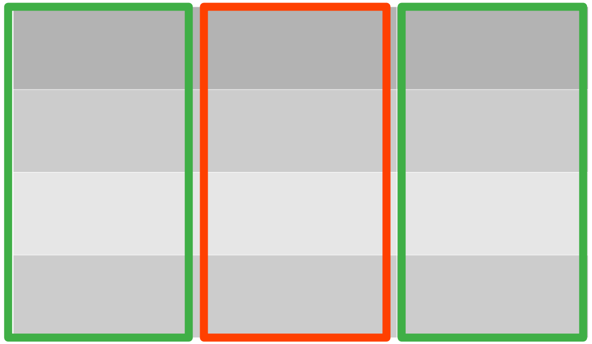
Column Storage

Columnar

Physical Memory layout



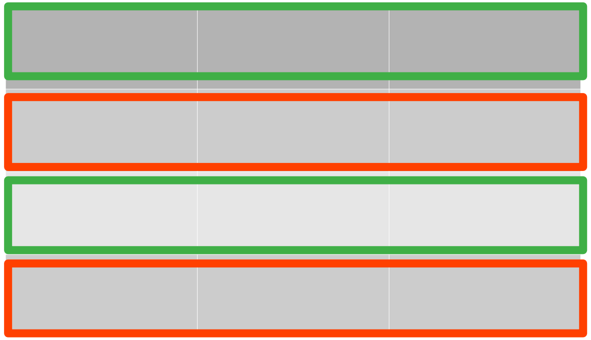
Row Storage



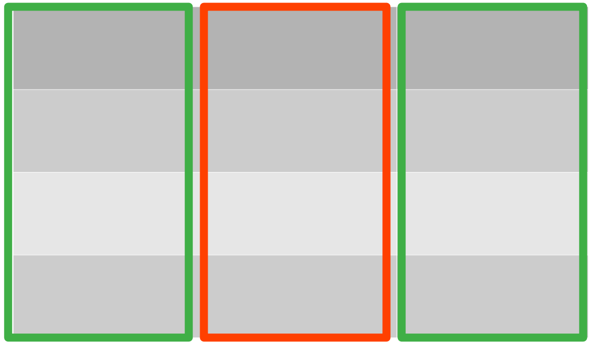
Column Storage

When?

Physical Memory layout



Row Storage

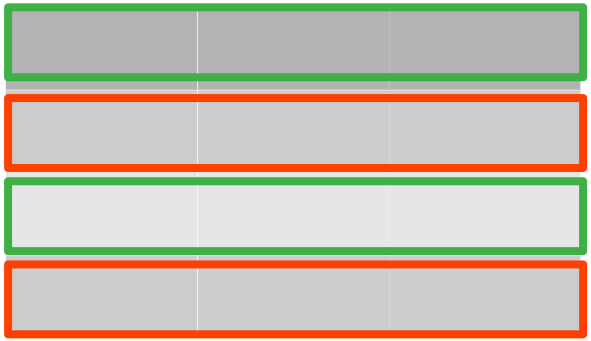


Column Storage

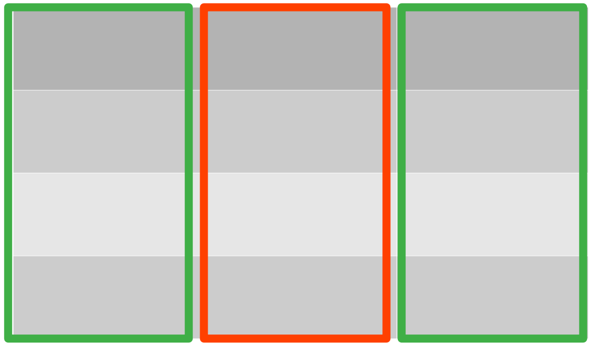
When? Insert, Update

Search for an attribute

Physical Memory layout



Row Storage

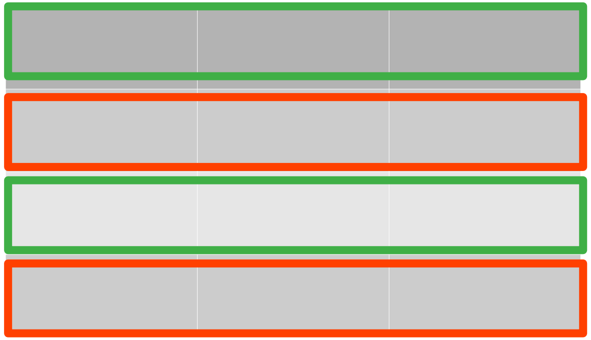


Column Storage

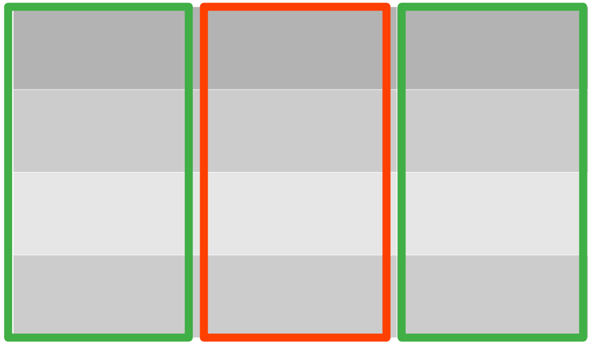
When? Insert, Update Search for an attribute

Hotly updated Coldly stored

Physical Memory layout



Row Storage

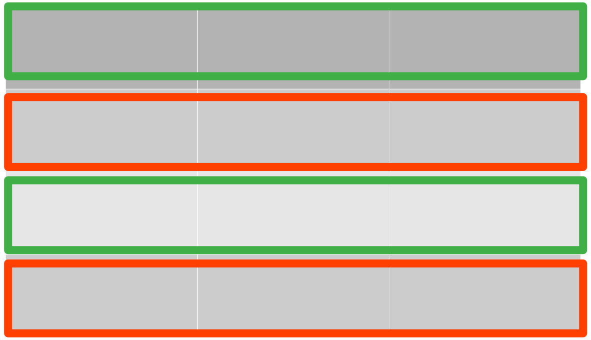


Column Storage

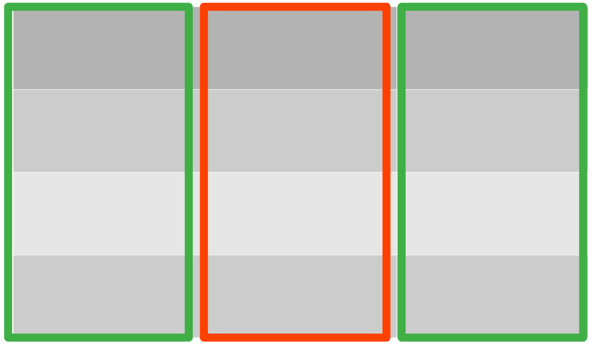
When? Insert, Update

Search for an attribute

Physical Memory layout



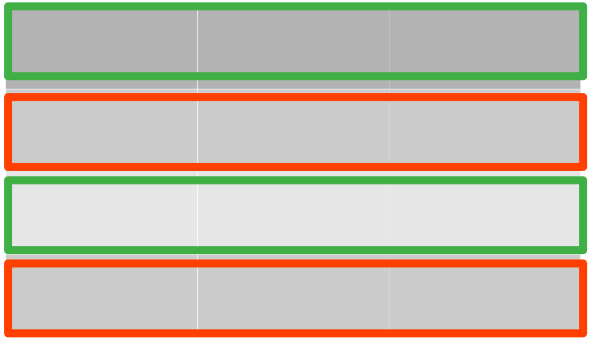
Row Storage



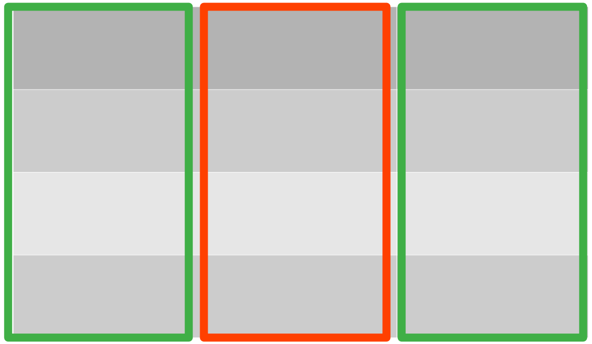
Column Storage

When?	Insert, Update	Search for an attribute
	Transactional	Analytical

Physical Memory layout



Row Storage

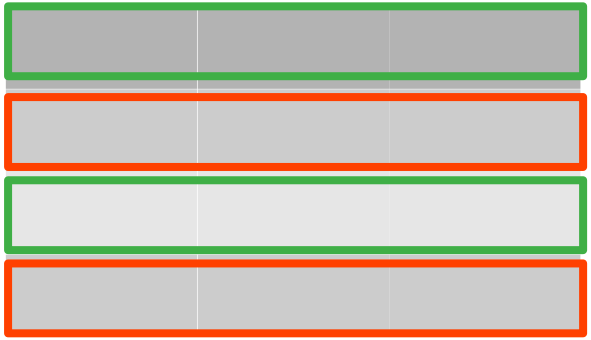


Column Storage

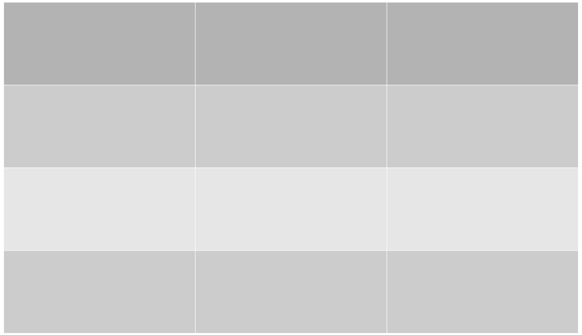
When? Insert, Update Search for an attribute

OLTP OLAP

Physical Memory layout



Row Storage

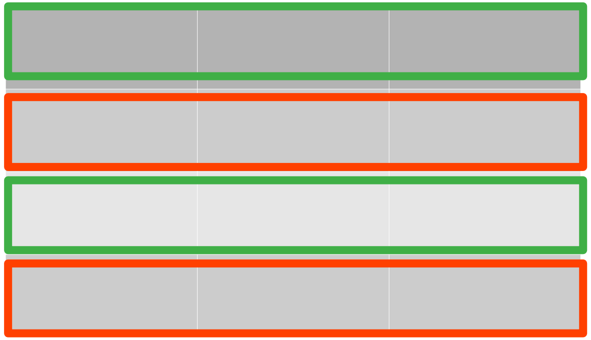


When? Insert, Update

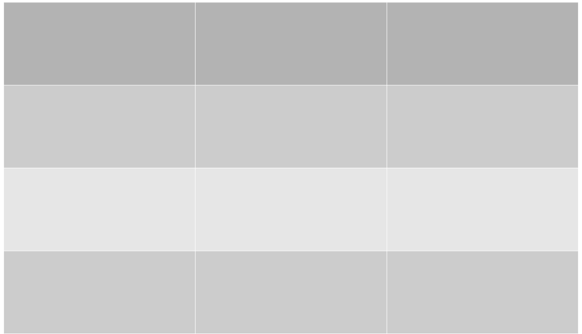
Search for an attribute

OLTP

Physical Memory layout



Row Storage

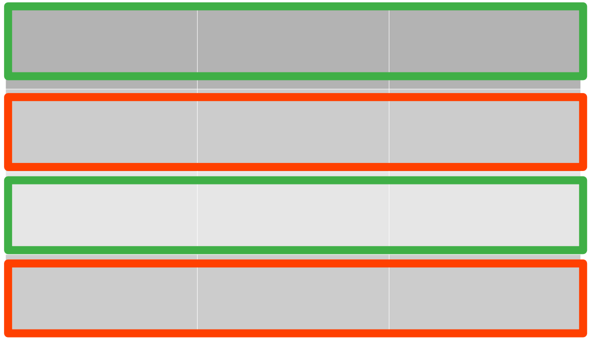


When? Insert, Update

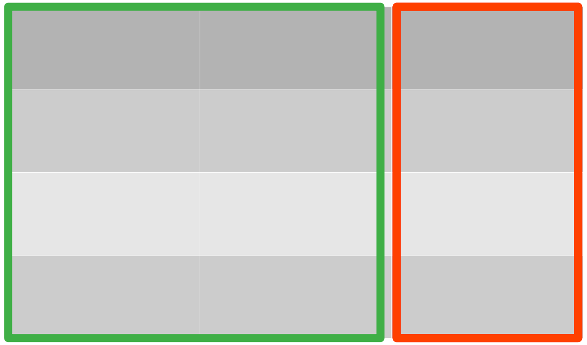
OLTP

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

Physical Memory layout



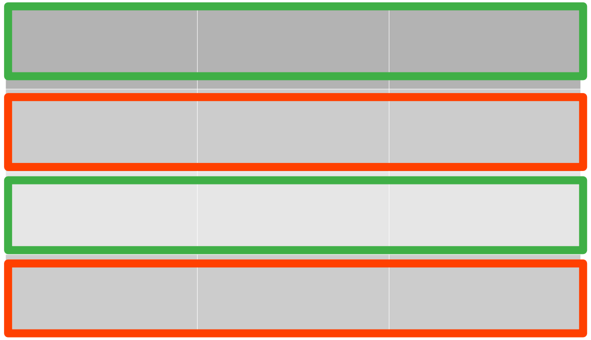
Row Storage



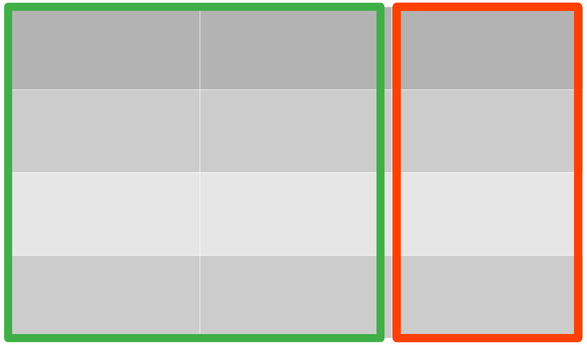
When? Insert, Update Search for **(a1,a2)** attribute

OLTP HTAP

Physical Memory layout



Row Storage



??

When? Insert, Update

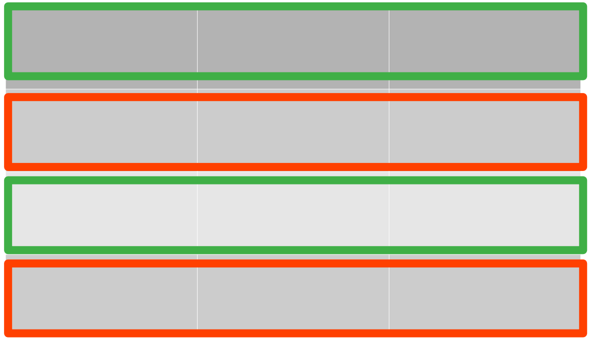
Search for **(a1,a2)** attribute

OLTP

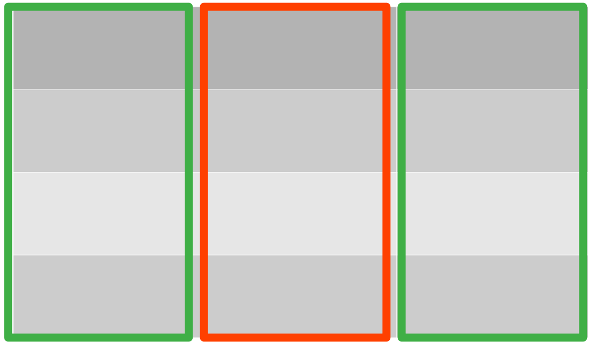
HTAP

Physical Memory layout

OLTP

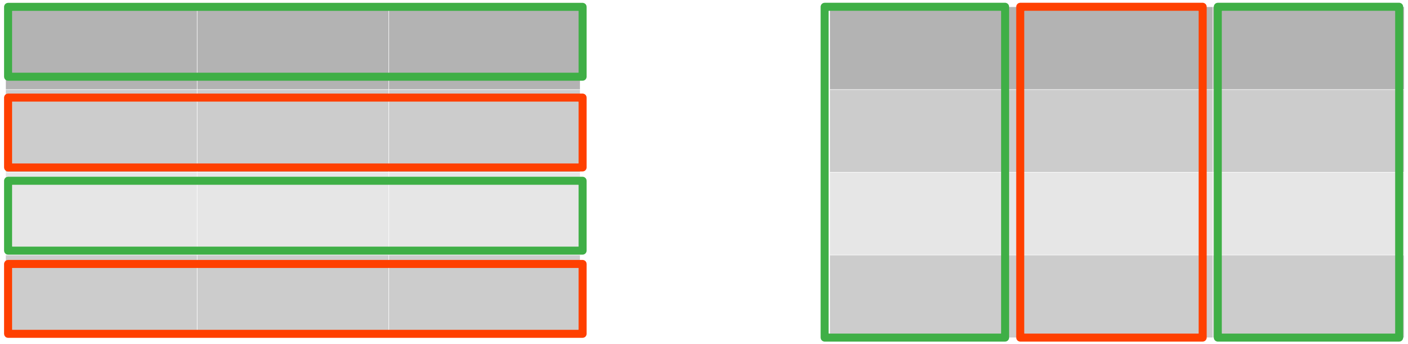


OLAP



Physical Memory layout

OLTP → HTAP ← OLAP



Physical Memory layout

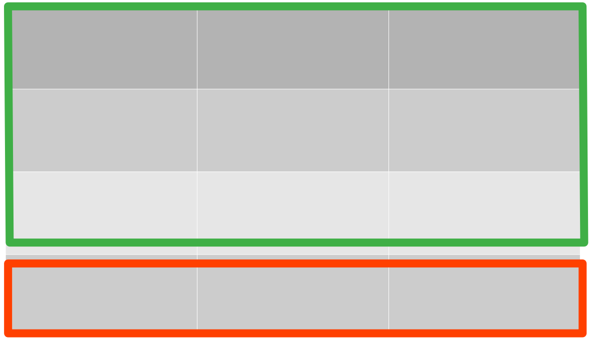
OLTP → HTAP ← OLAP

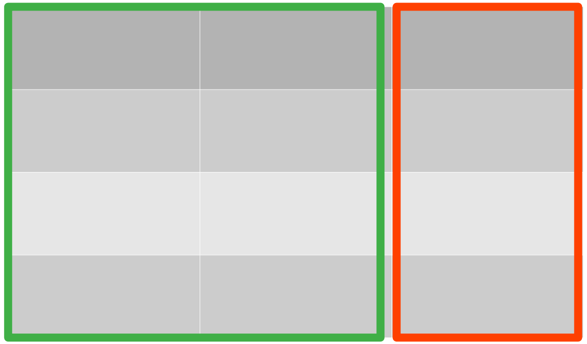
Physical Memory layout

HTAP

Physical Memory layout

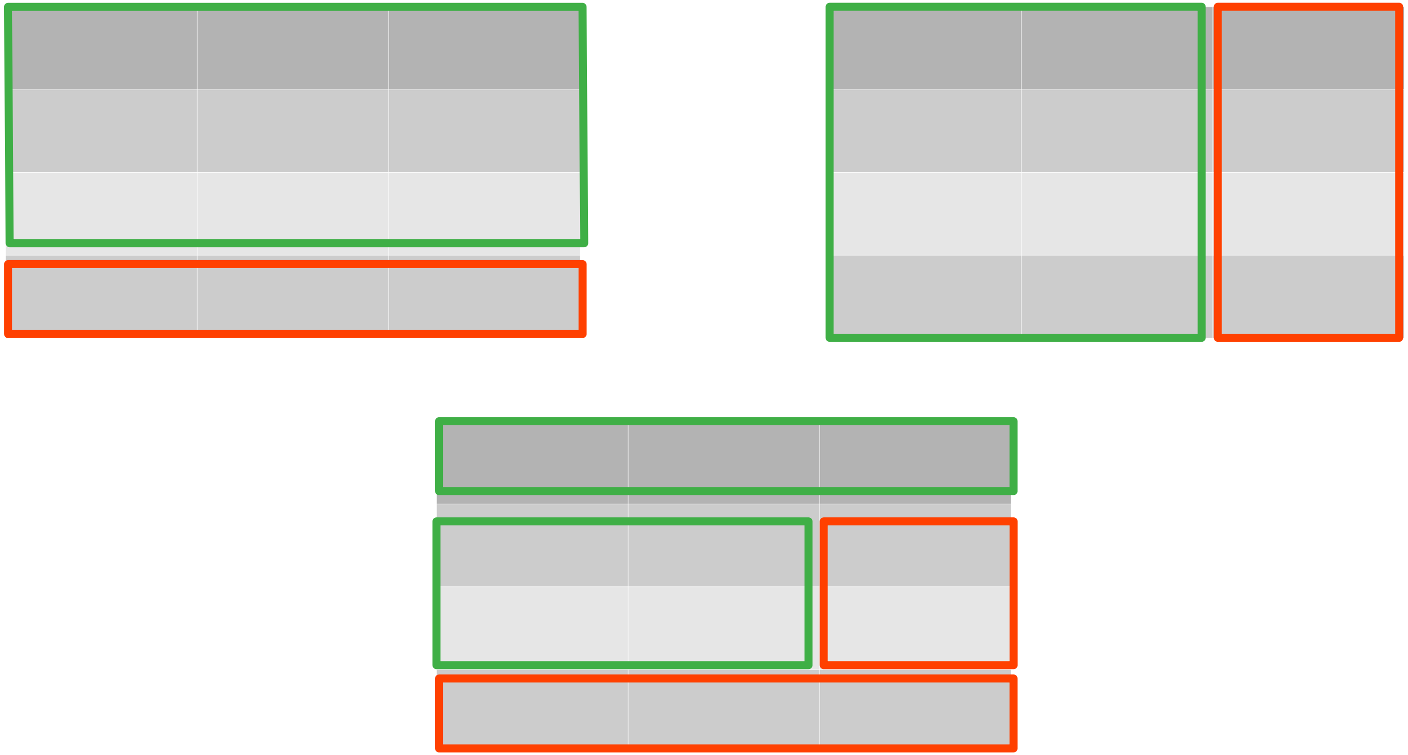
HTAP





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

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

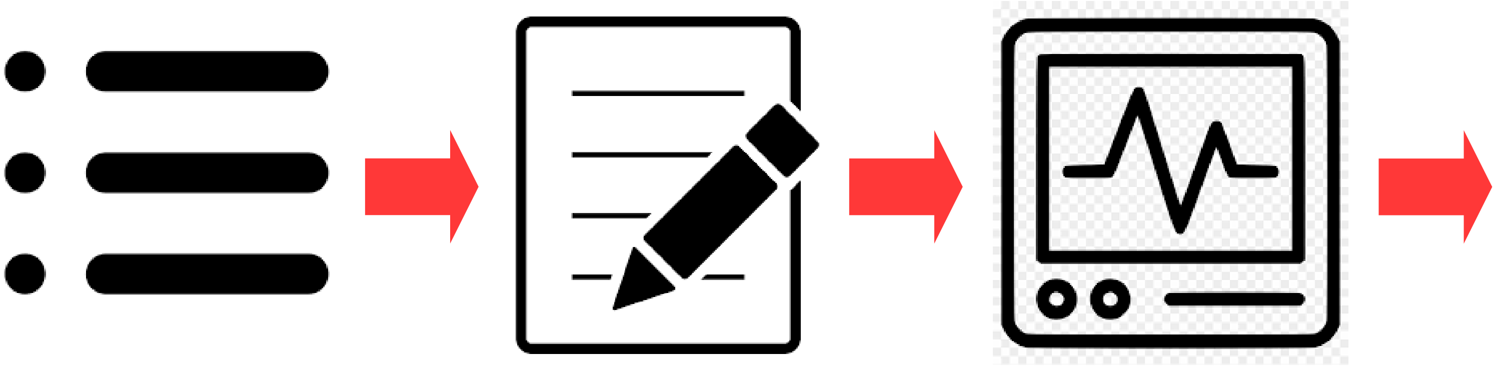
Record

Self-adaptive algorithm

Record

Analysis

Self-adaptive algorithm



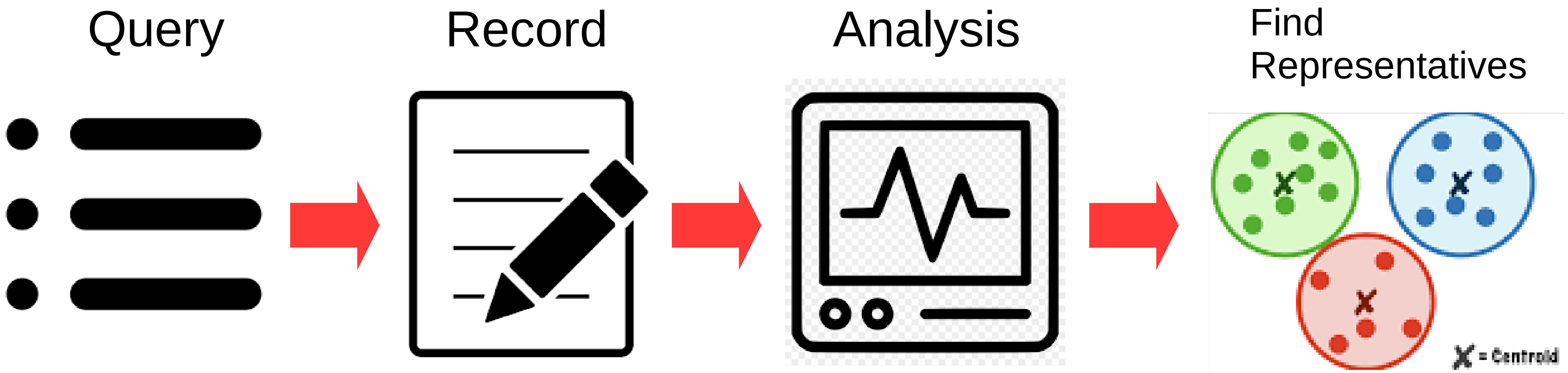
Query

Record

Analysis

??

Self-adaptive algorithm



Query

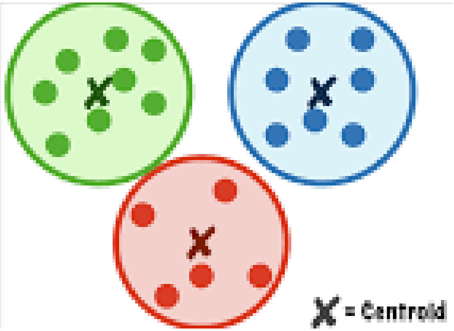
Record

Analysis

Find Representatives

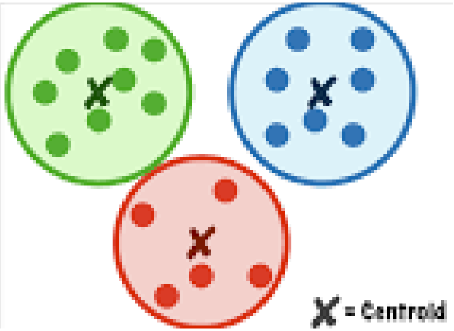
On-line k-means algorithm

Find
Representatives



On-line k-means algorithm

Find Representatives



1st step:

Using recent n Queries,

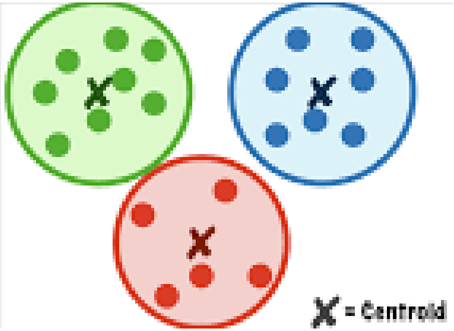
Find k Representatives R

2nd step:

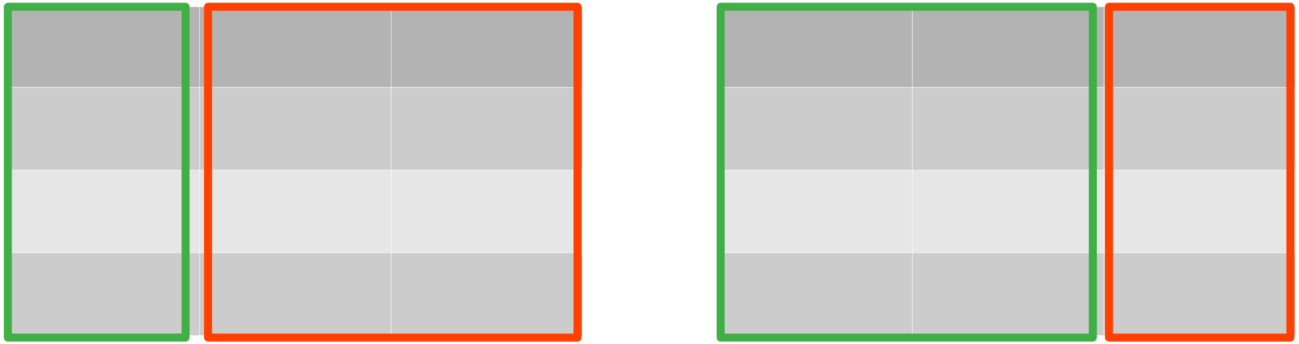
Generate vertical partitioned layout using R with greedy algorithm. (Largest cluster first)

On-line k-means algorithm

Find Representatives



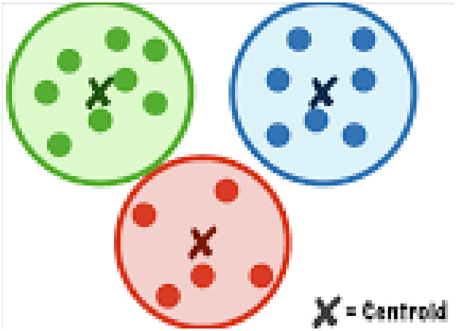
FSM (Flexible Storage Model)



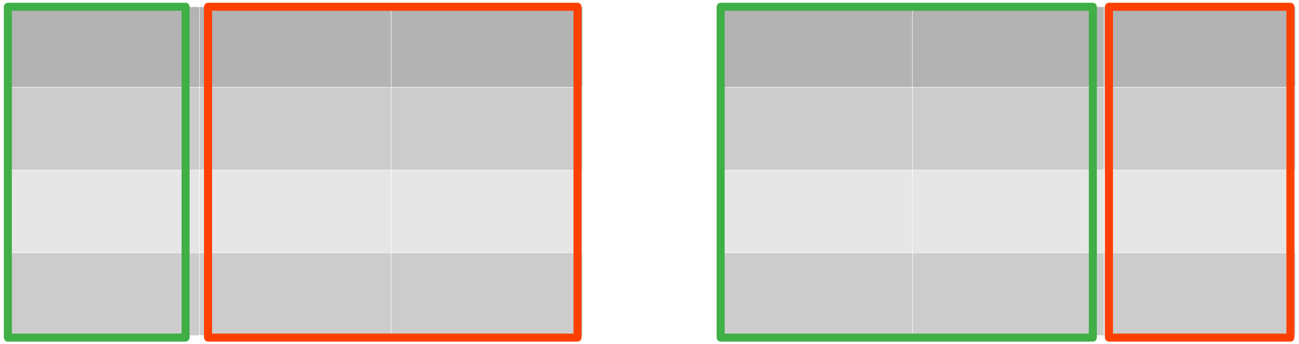
Different tables, different vertical layouts

On-line k-means algorithm

Find
Representatives



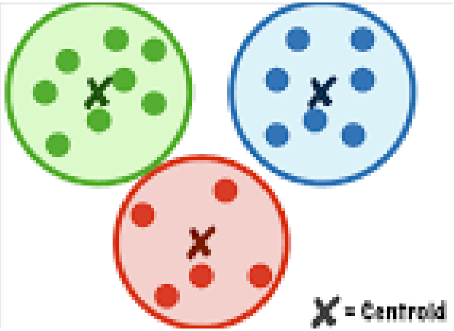
FSM (Flexible Storage Model)



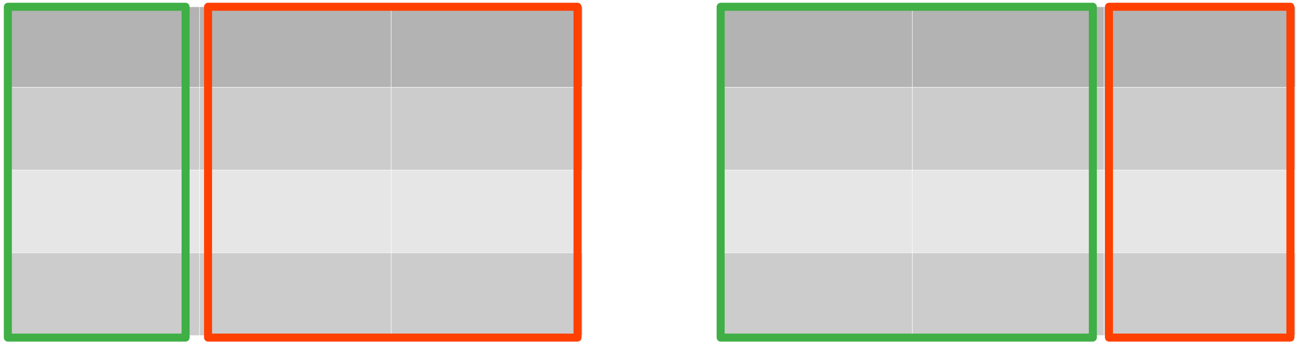
Different layouts → different access methods?

On-line k-means algorithm

Find Representatives



FSM (Flexible Storage Model)

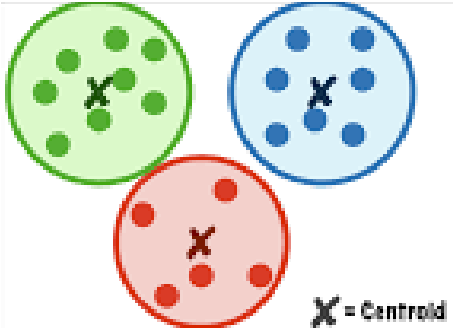


Different layouts → different access methods?

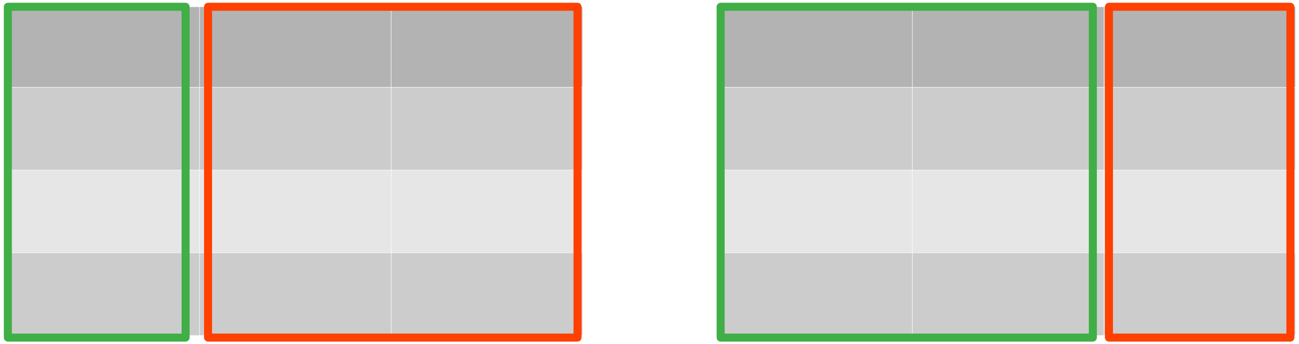
Inefficient !!

On-line k-means algorithm

Find Representatives



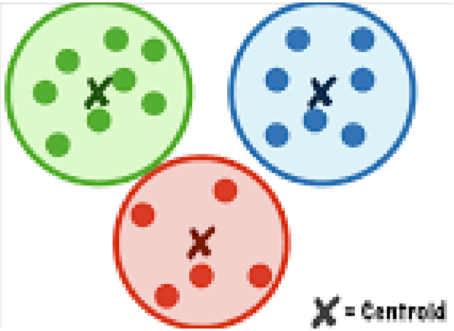
FSM (Flexible Storage Model)



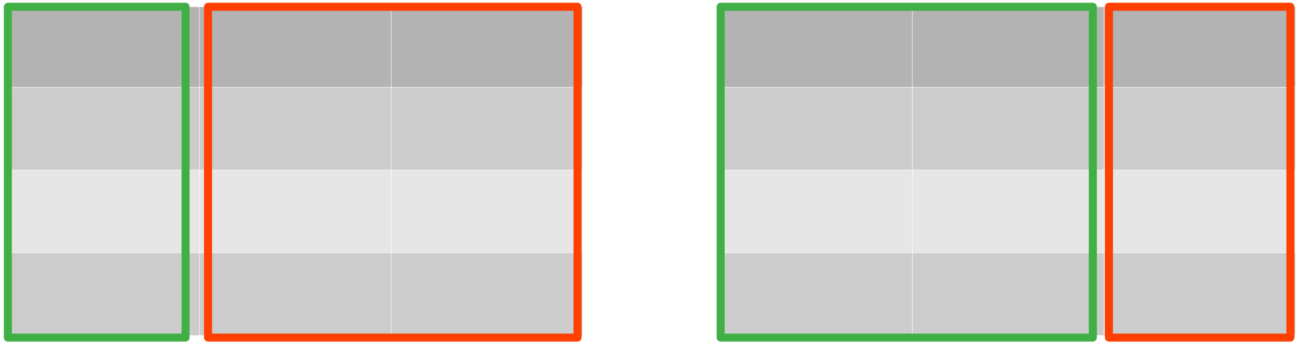
Provide Abstract layer !!

On-line k-means algorithm

Find Representatives



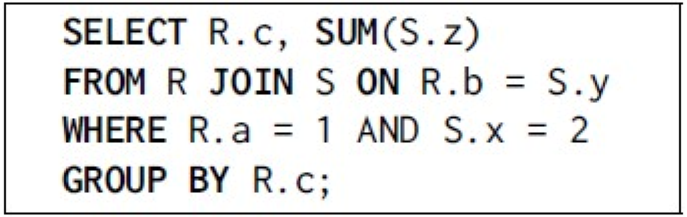
FSM (Flexible Storage Model)



Provide Abstract layer !!

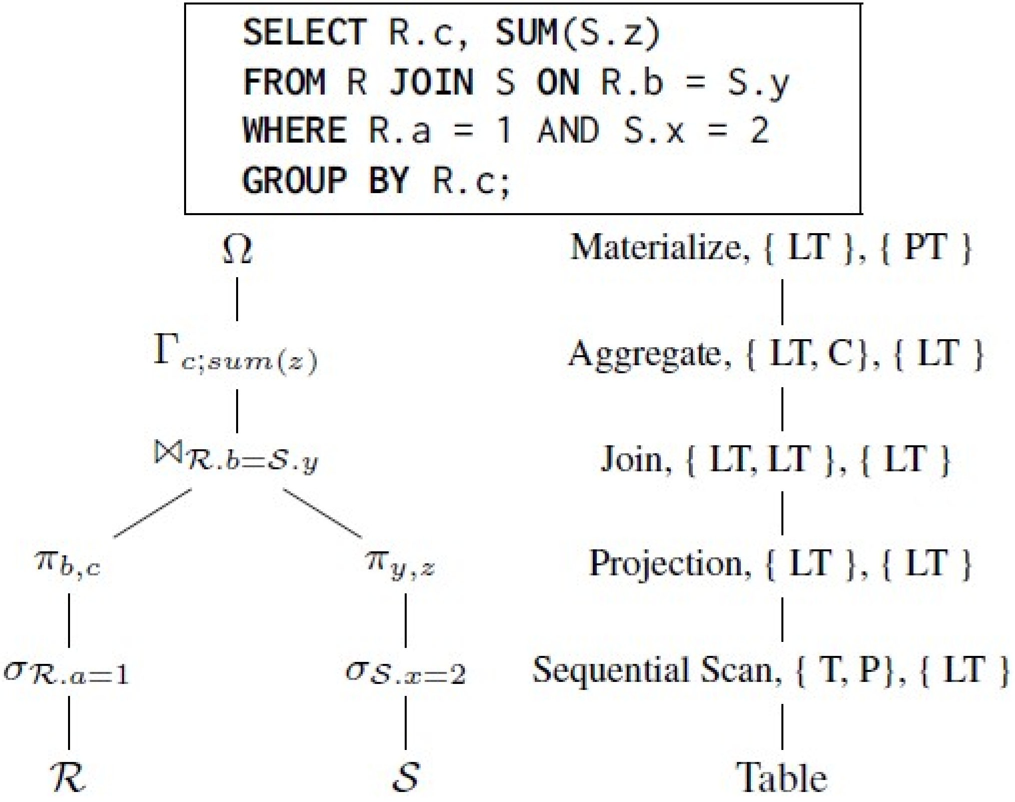
One mutual access interface.

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;
```

Ω
$\Gamma_{c;sum(z)}$
$\bowtie_{\mathcal{R}.b=\mathcal{S}.y}$
$\pi_{b,c}$
$\pi_{y,z}$
$\sigma_{\mathcal{R}.a=1}$
$\sigma_{\mathcal{S}.x=2}$
$\mathcal{R}$
$\mathcal{S}$
Materialize, { LT }, { PT }
Aggregate, { LT, C}, { LT }
Join, { LT, LT }, { LT }
Projection, { LT }, { LT }
Sequential Scan, { T, P}, { LT }
Table

LT : logical tile

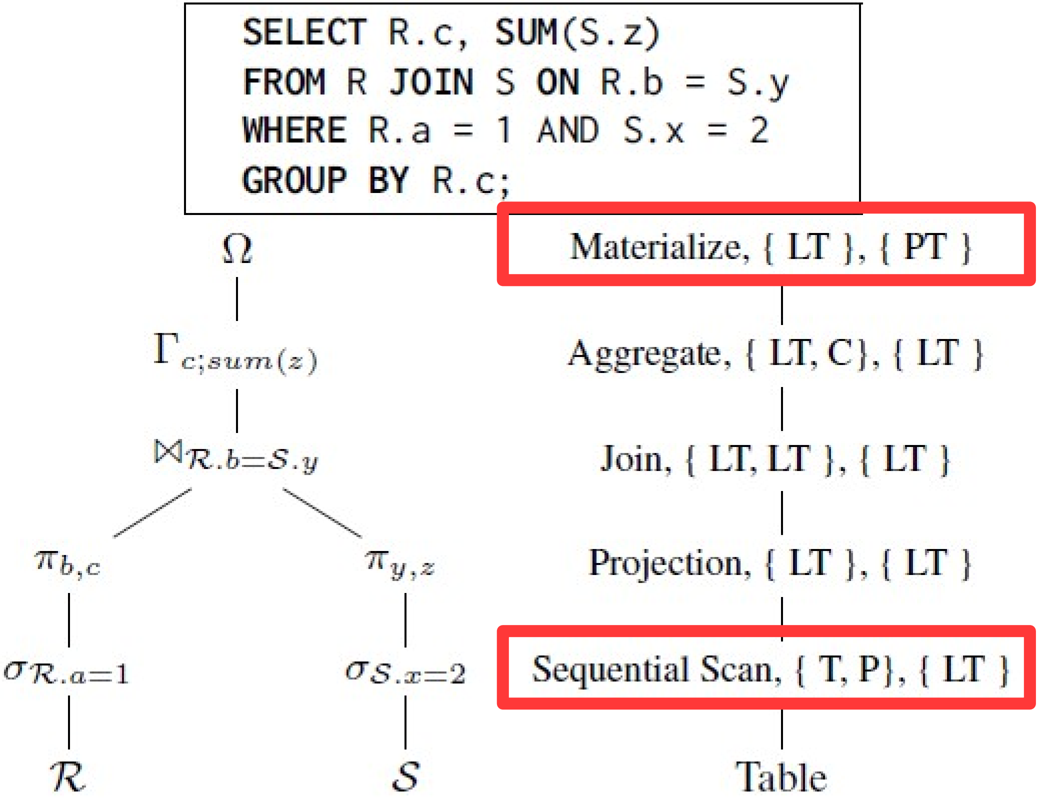
PT : physical tile

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;
```

Ω
$\Gamma_{c;sum(z)}$
$\bowtie_{\mathcal{R}.b=\mathcal{S}.y}$
$\pi_{b,c}$
$\pi_{y,z}$
$\sigma_{\mathcal{R}.a=1}$
$\sigma_{\mathcal{S}.x=2}$
$\mathcal{R}$
$\mathcal{S}$
Materialize, { LT }, { PT }
Aggregate, { LT, C}, { LT }
Join, { LT, LT }, { LT }
Projection, { LT }, { LT }
Sequential Scan, { T, P}, { LT }
Table

LT : logical tile

PT : physical tile

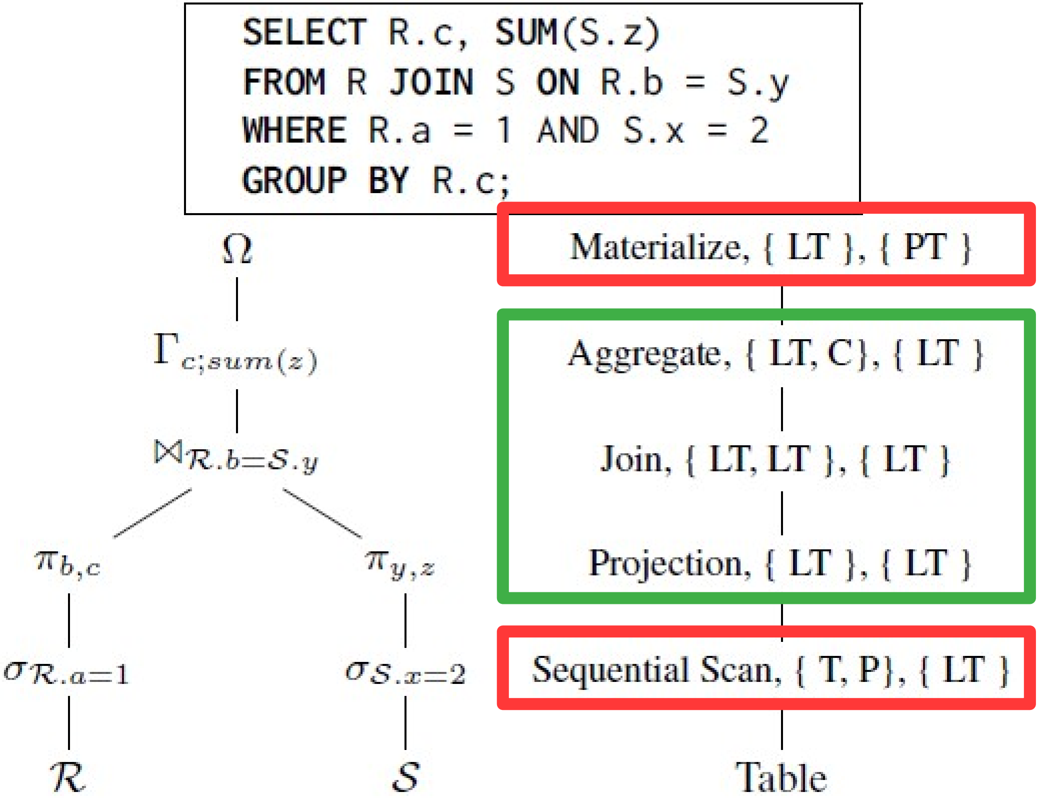
T : table

Attributes : C

Predicate : P

Physical data

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;
```

Ω
$\Gamma_{c;sum(z)}$
$\bowtie_{\mathcal{R}.b=\mathcal{S}.y}$
$\pi_{b,c}$
$\pi_{y,z}$
$\sigma_{\mathcal{R}.a=1}$
$\sigma_{\mathcal{S}.x=2}$
$\mathcal{R}$
$\mathcal{S}$
Materialize, { LT }, { PT }
Aggregate, { LT, C}, { LT }
Join, { LT, LT }, { LT }
Projection, { LT }, { LT }
Sequential Scan, { T, P}, { LT }
Table

LT : logic tile

PT : physical tile

T : table

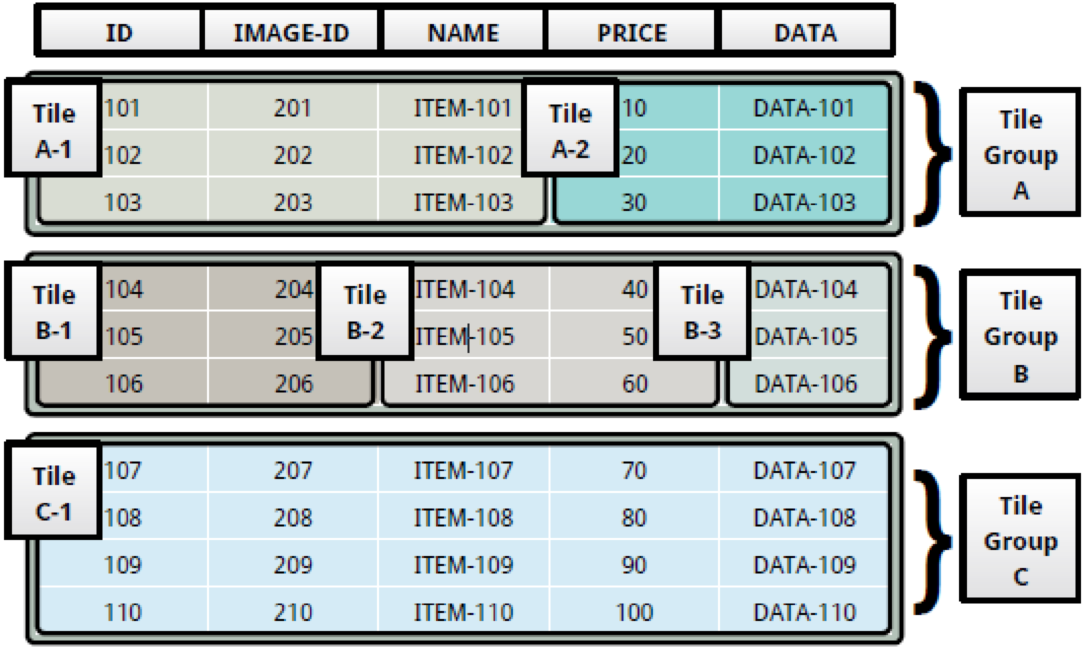
Attributes : C

Predicate : P

logical data +

Flexible materialization

Tiles



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
105	205	ITEM-105	50	DATA-105
106	206	ITEM-106	60	DATA-106
107	207	ITEM-107	70	DATA-107
108	208	ITEM-108	80	DATA-108
109	209	ITEM-109	90	DATA-109
110	210	ITEM-110	100	DATA-110

Tile A-1

Tile A-2

Tile Group A

Tile B-1

Tile B-2

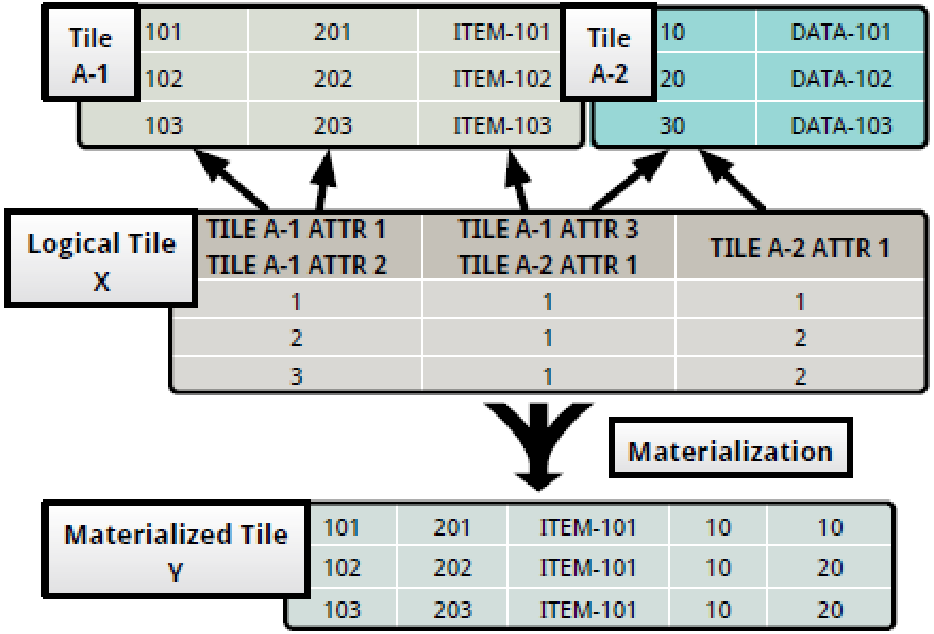
Tile B-3

Tile Group B

Tile C-1

Tile Group C

Physical Tile



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

Logical Tile

Parallelism

Definitely we do not want stalling.

Parallelism

Definitely we do not want stalling.

Write while reconfigure memory layout

Parallelism

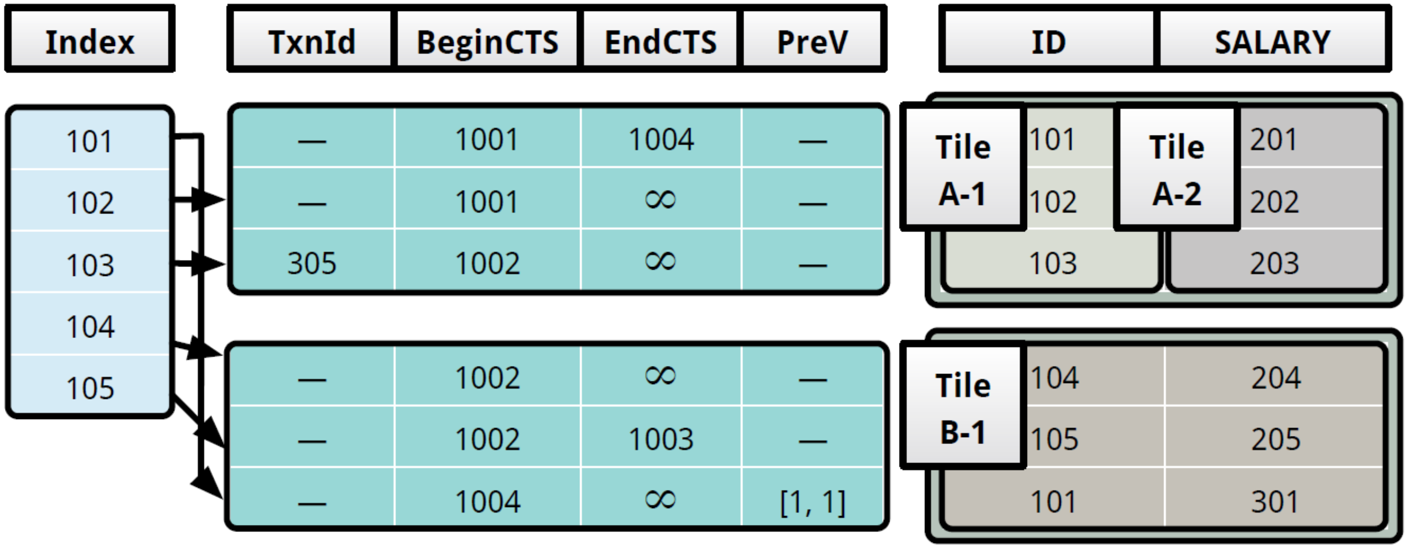
Definitely we do not want stalling.

Write while reconfigure memory layout

→ MVCC

(Multi-Version Concurreny Control)

MVCC



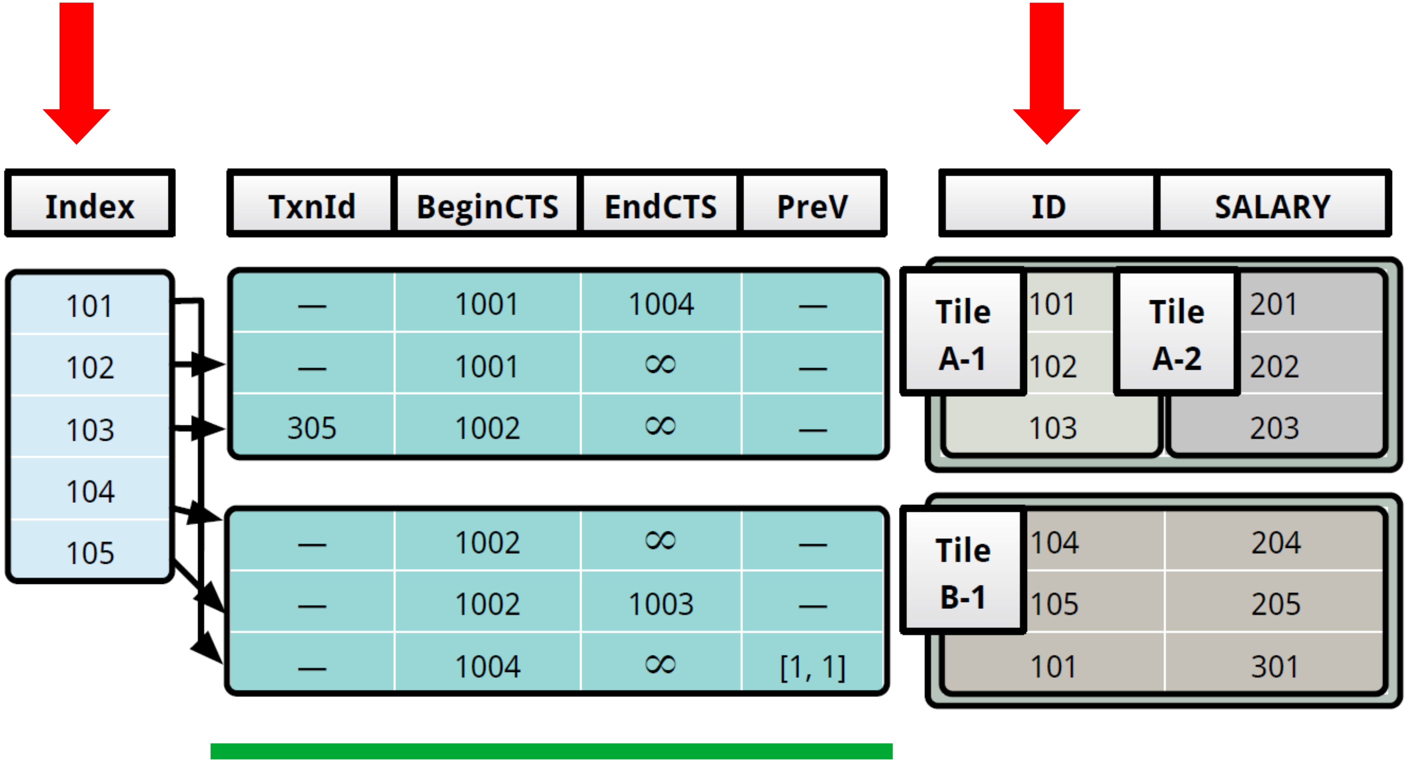
Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

	ID		SALARY
Tile A-1	101	Tile A-2	201
	102		202
	103		203
Tile B-1	104		204
	105		205
	101		301

Metadata for MVCC

MVCC



Index
101
102
103
104
105

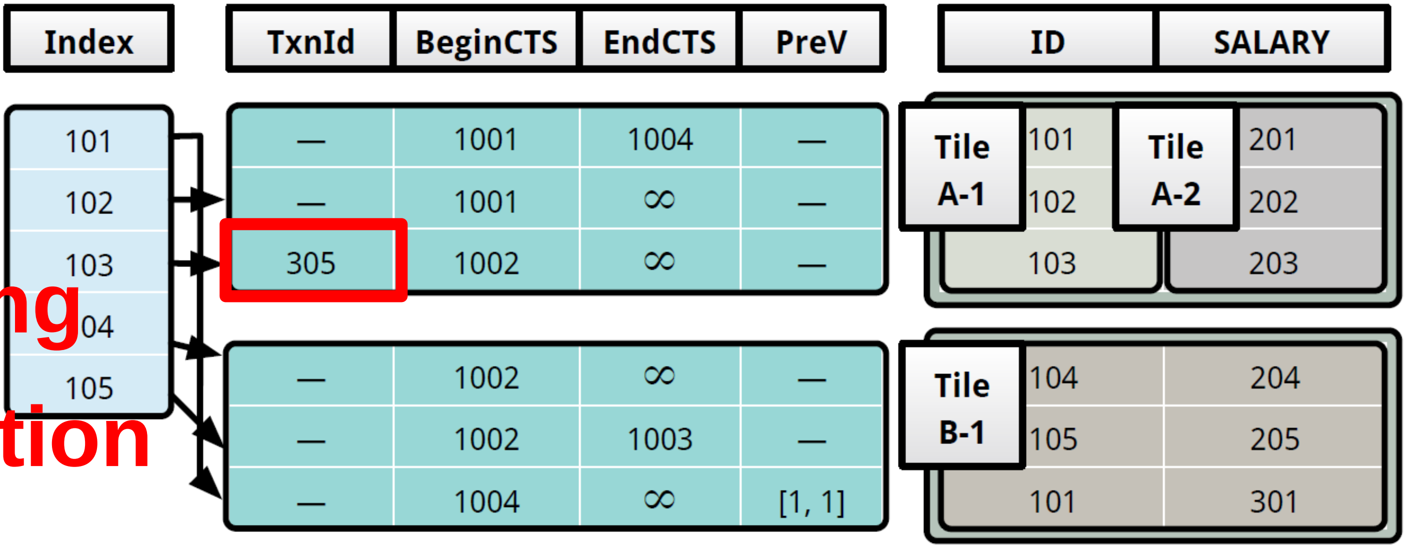
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

Unique transaction identifier



Index
101
102
103
104
105

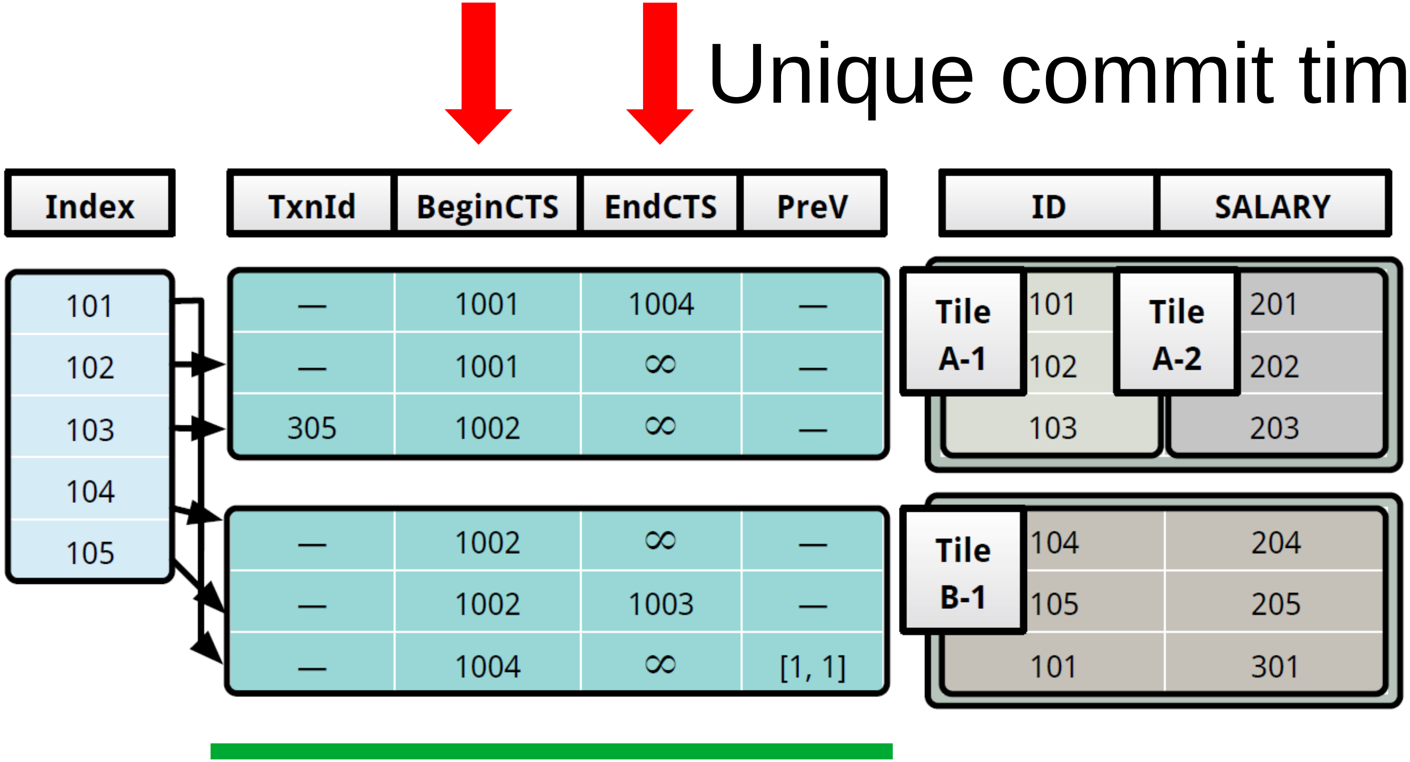
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Running transaction

Metadata for MVCC

MVCC



Unique commit timestamp

Index
101
102
103
104
105

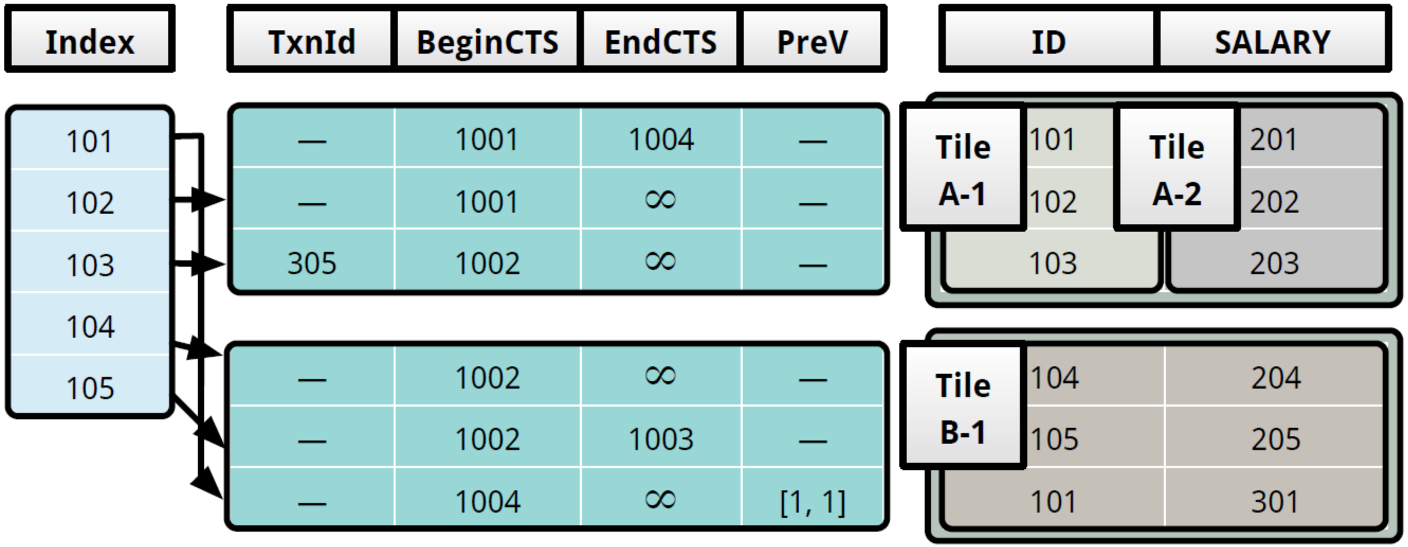
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

i) Update (atomic)



Index
101
102
103
104
105

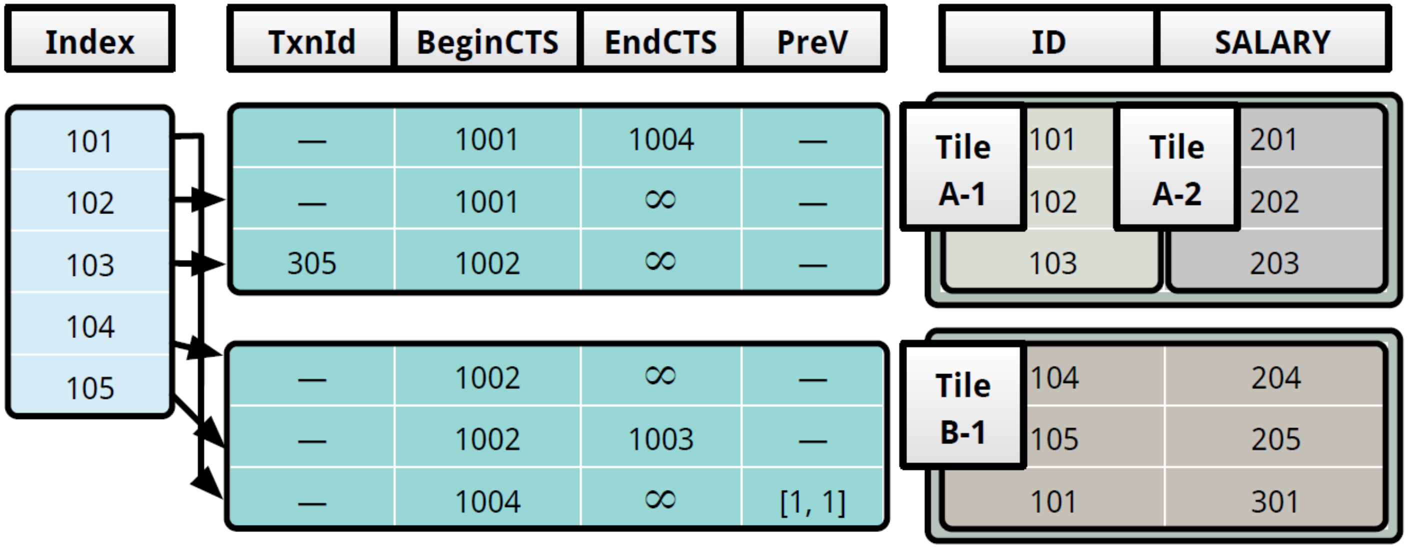
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

	ID	SALARY
Tile A-1 / Tile A-2	101	201
	102	202
	103	203
Tile B-1	104	204
	105	205
	101	301

Metadata for MVCC

MVCC

i) Update (after commit)



Index
101
102
103
104
105

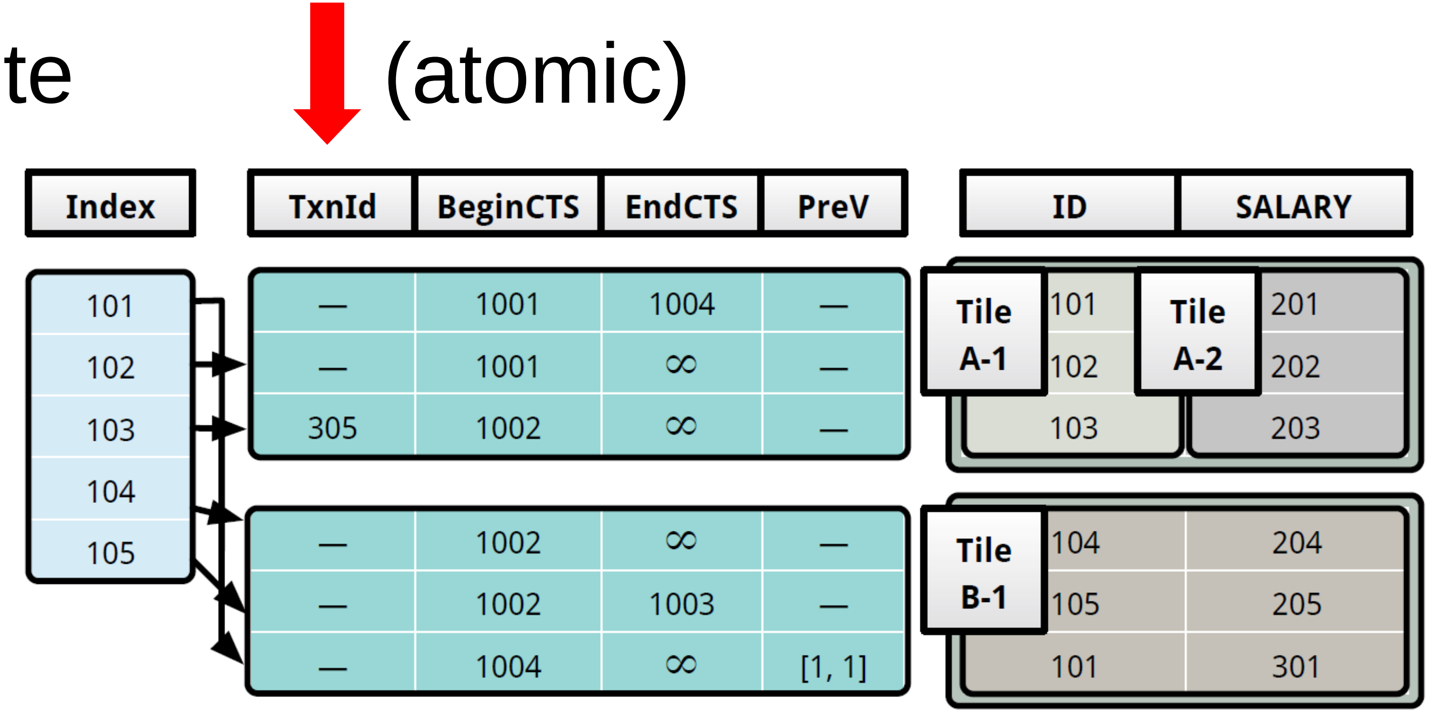
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

ii) Delete (atomic)



Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

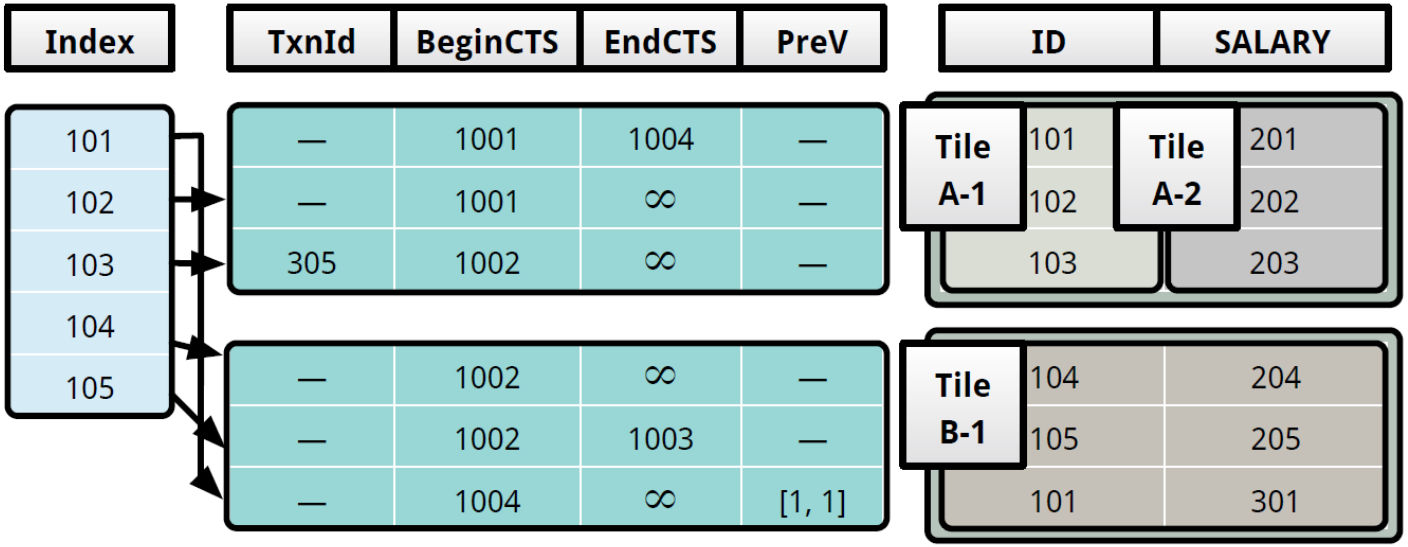
ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

ii) Delete

(after commit)



Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

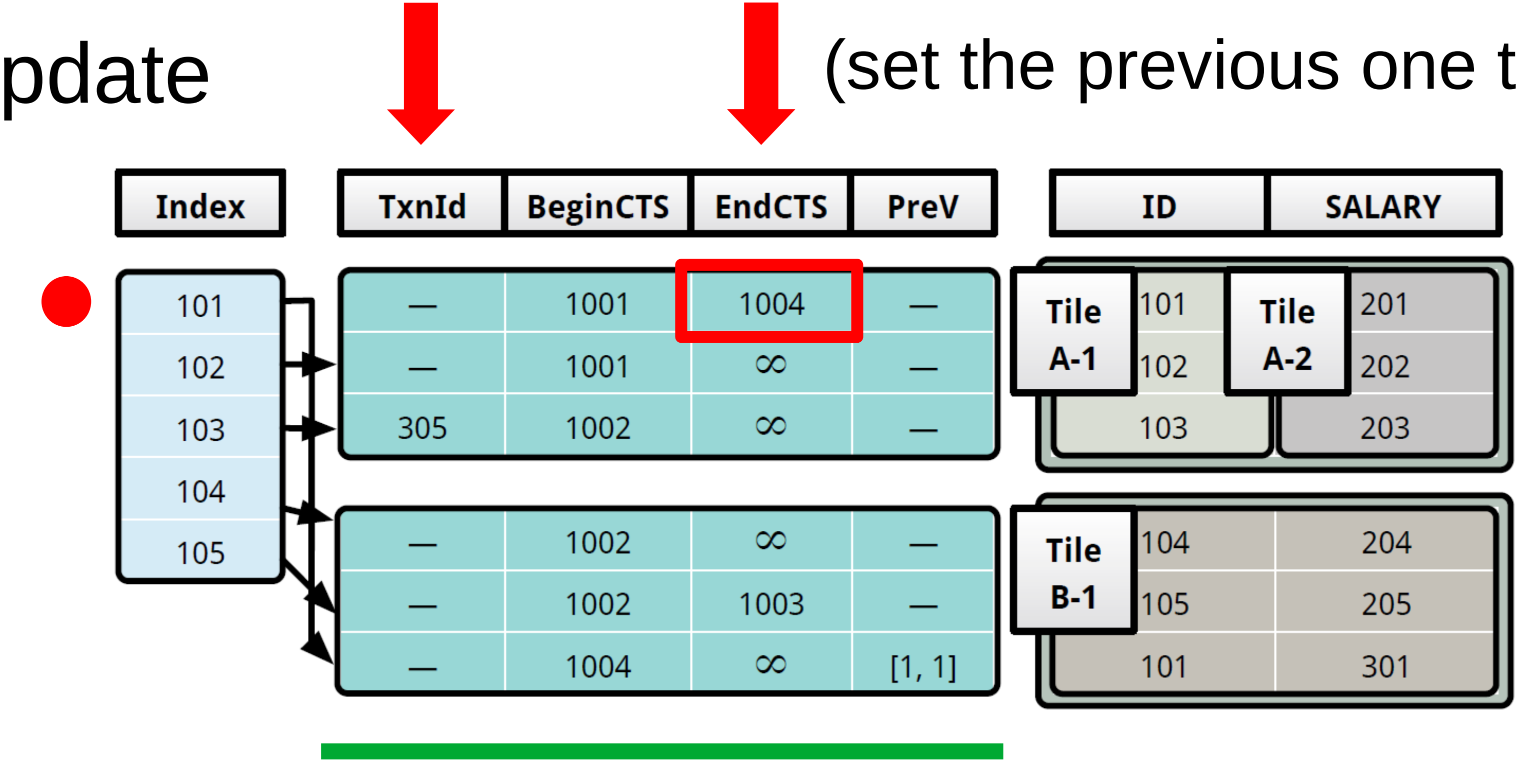
ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

iii) Update

(set the previous one to invalid)



Index
101
102
103
104
105

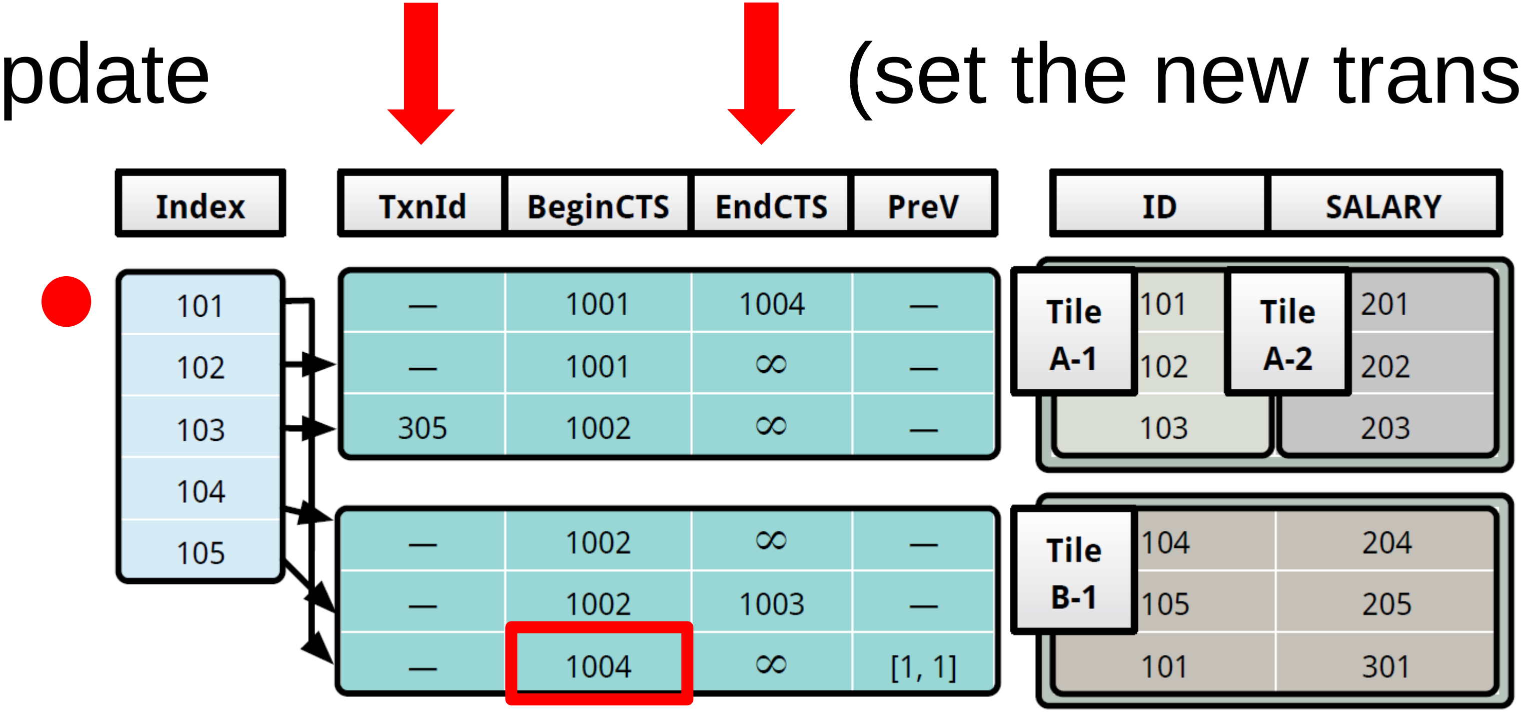
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

iii) Update (set the new transaction)



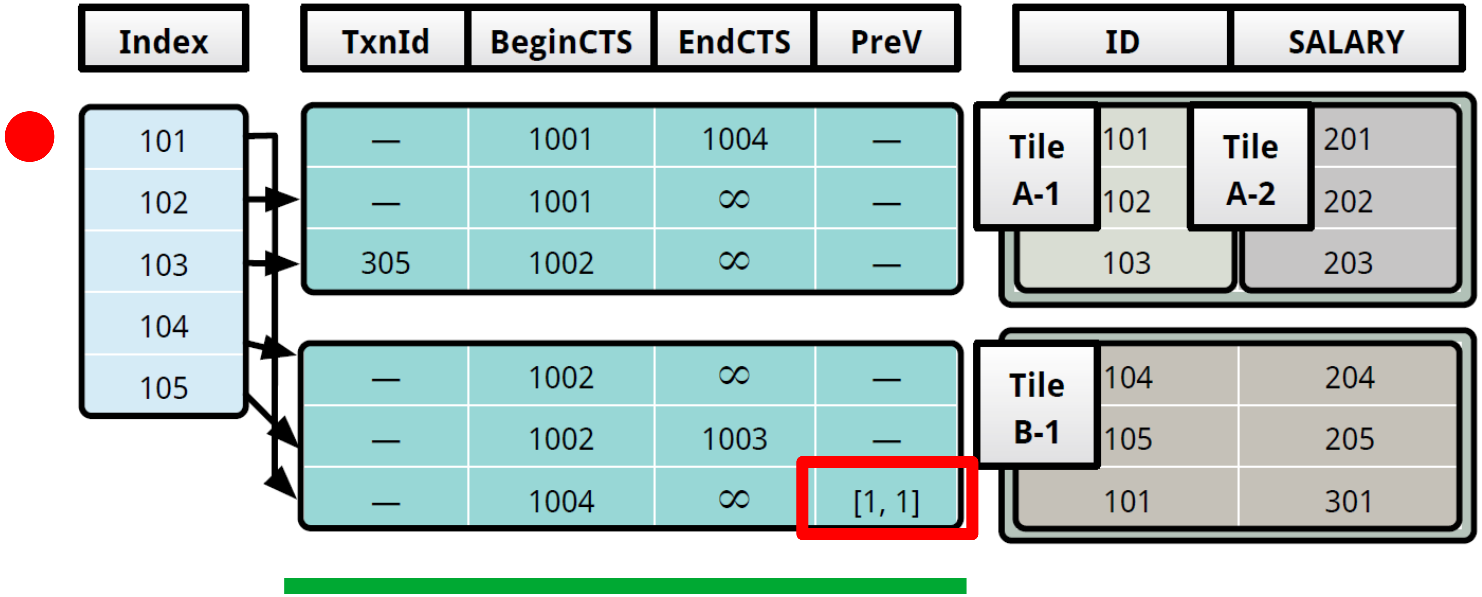
Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV	ID	SALARY
—	1001	1004	—	101	201
—	1001	∞	—	102	202
305	1002	∞	—	103	203
—	1002	∞	—	104	204
—	1002	1003	—	105	205
—	1004	∞	[1, 1]	101	301

Metadata for MVCC

MVCC

iii) Update (point to the previous one)



Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

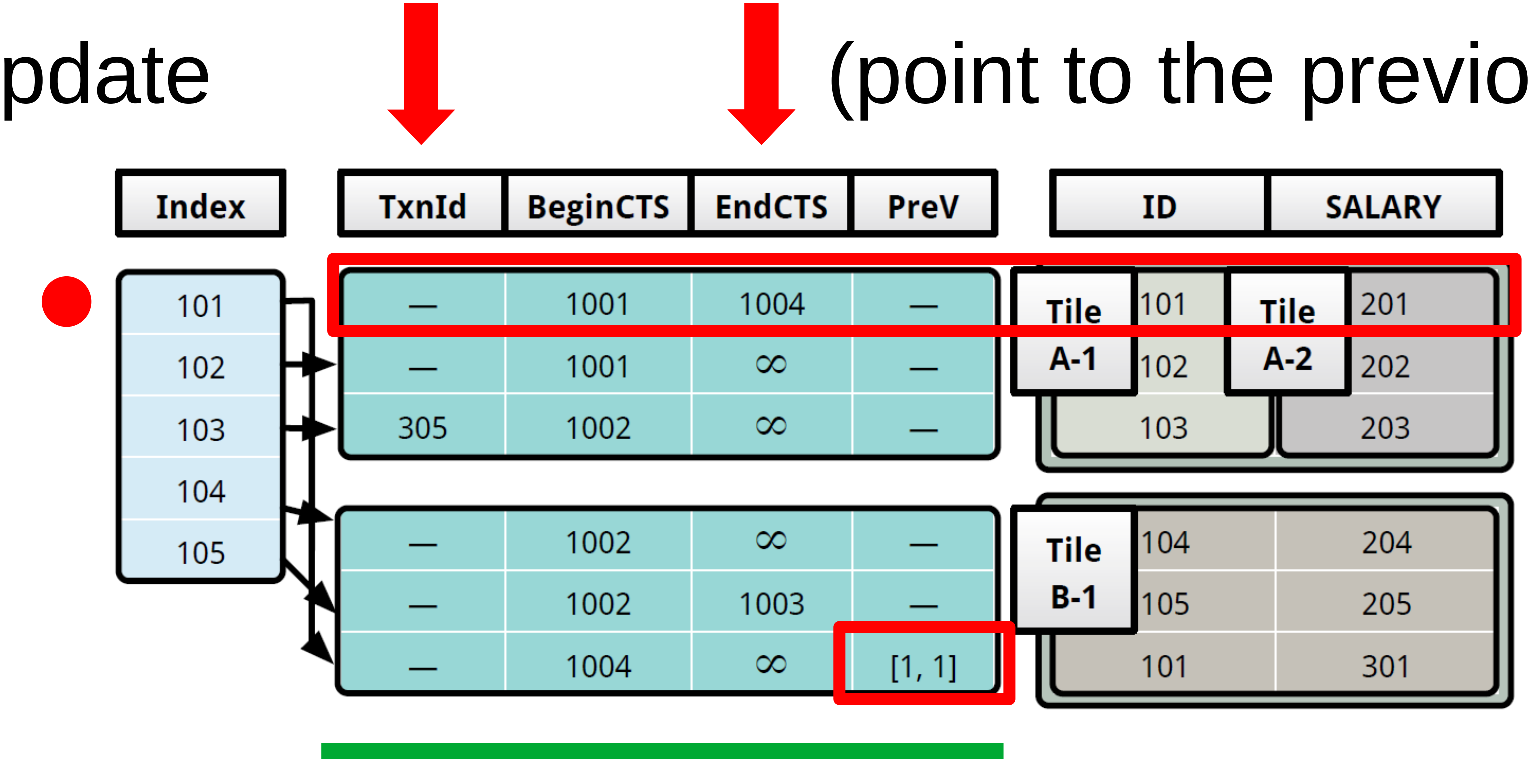
ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

iii) Update

(point to the previous one)



Index
101
102
103
104
105

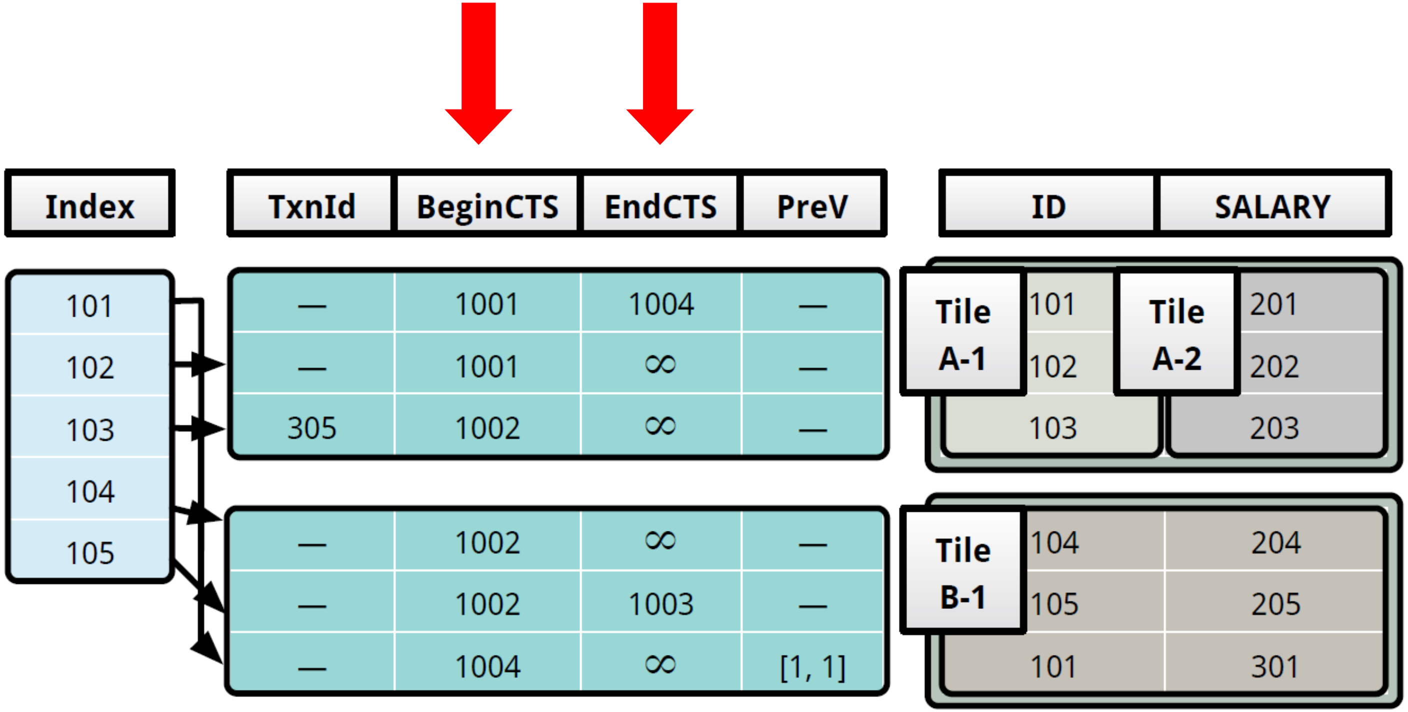
TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Metadata for MVCC

MVCC

Search



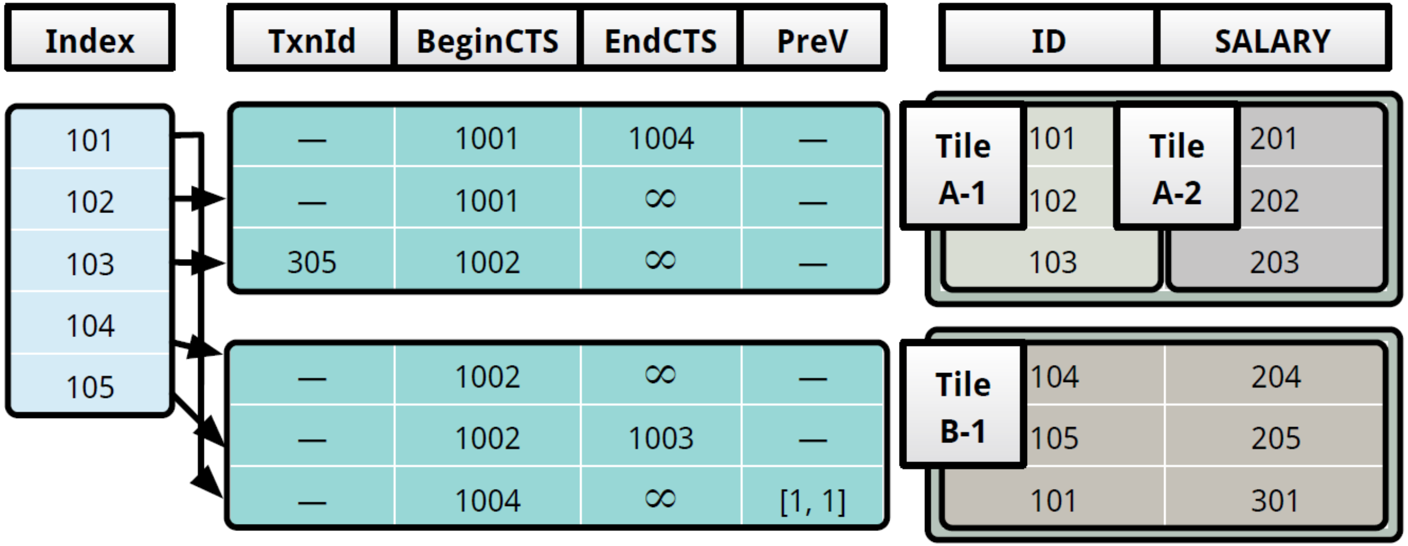
Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Looking through the metadata, find the one within Its visibility (BeginCTS < transaction ID < EndCTS)

MVCC



Index
101
102
103
104
105

TxnId	BeginCTS	EndCTS	PreV
—	1001	1004	—
—	1001	∞	—
305	1002	∞	—
—	1002	∞	—
—	1002	1003	—
—	1004	∞	[1, 1]

ID	SALARY
101	201
102	202
103	203
104	204
105	205
101	301

Garbage Collector claims those old spaces back

Performance

What about the overhead?

Is this method practical?

Performance

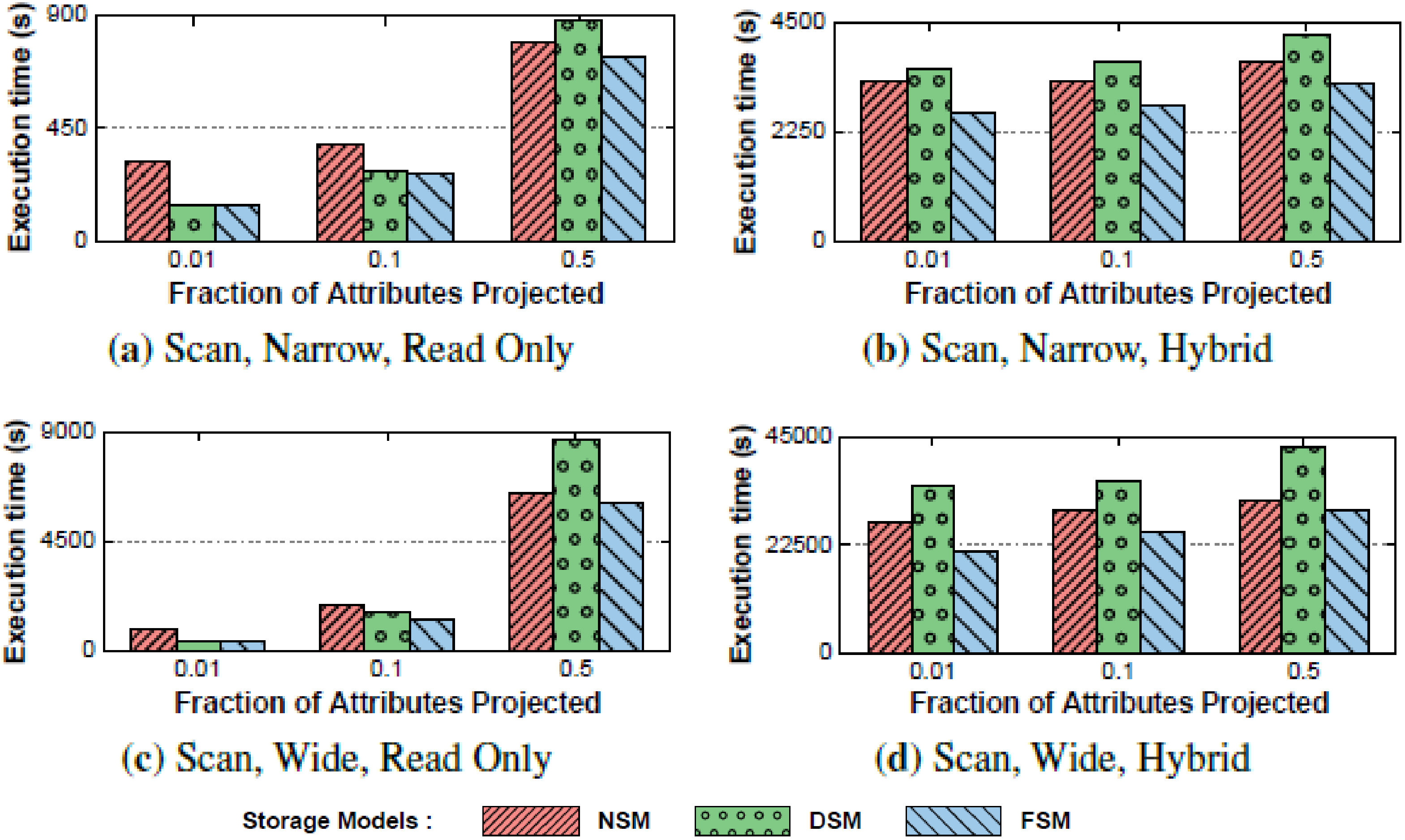
What about the overhead?

Is this method practical?

Let us see the results.

Evaluation

NSM / DSM / FSM SCAN



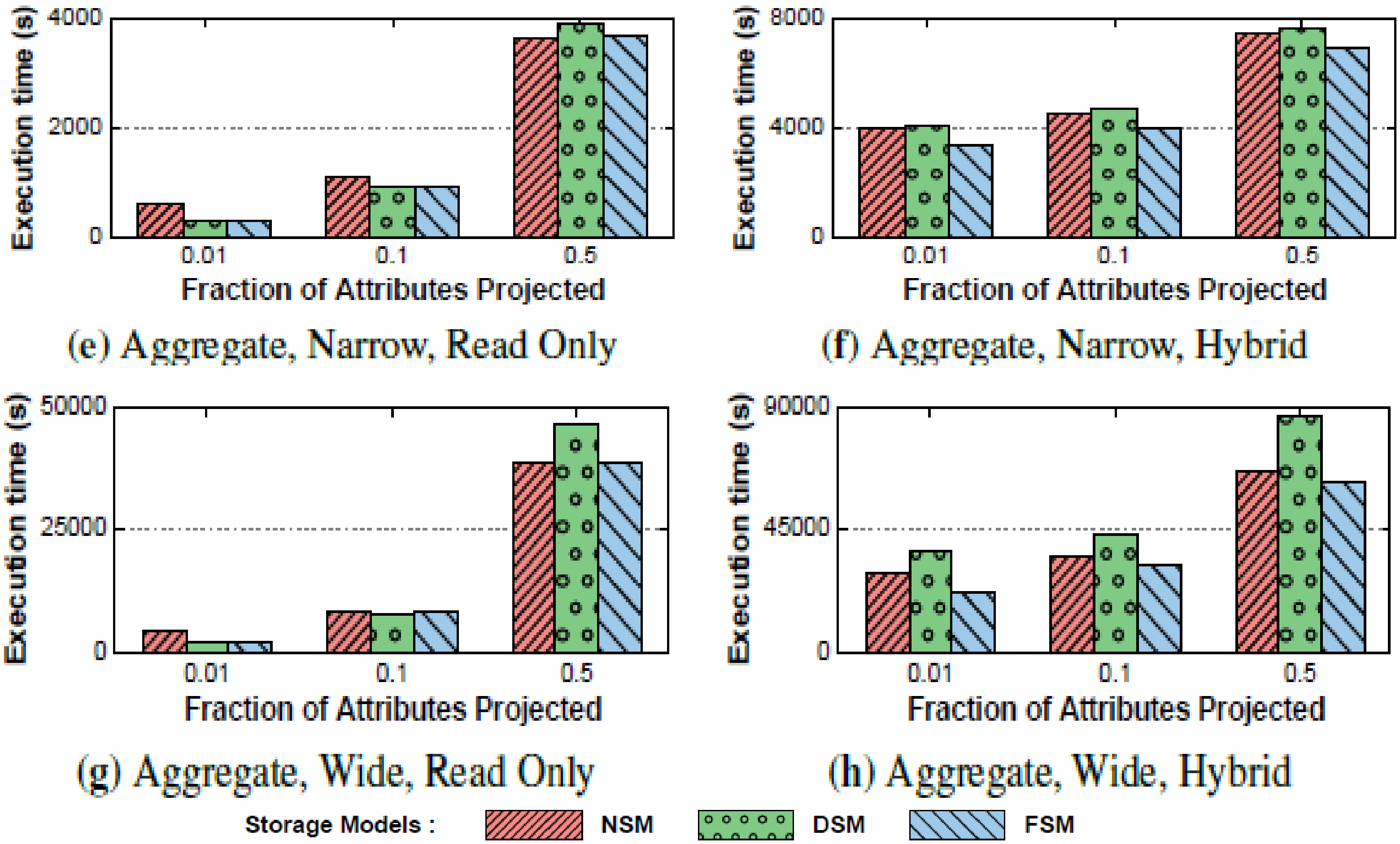
(a) Scan, Narrow, Read Only

(b) Scan, Narrow, Hybrid

(c) Scan, Wide, Read Only

(d) Scan, Wide, Hybrid

NSM / DSM / FSM Aggregate



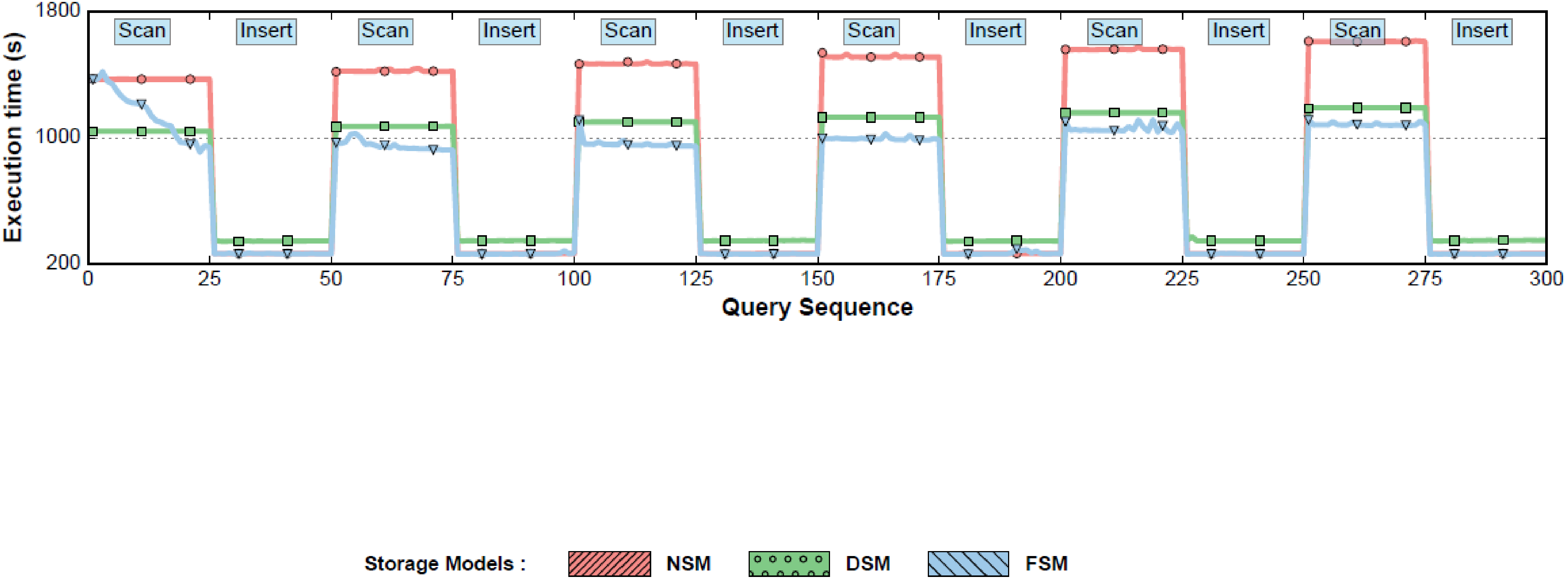
(e) Aggregate, Narrow, Read Only

(f) Aggregate, Narrow, Hybrid

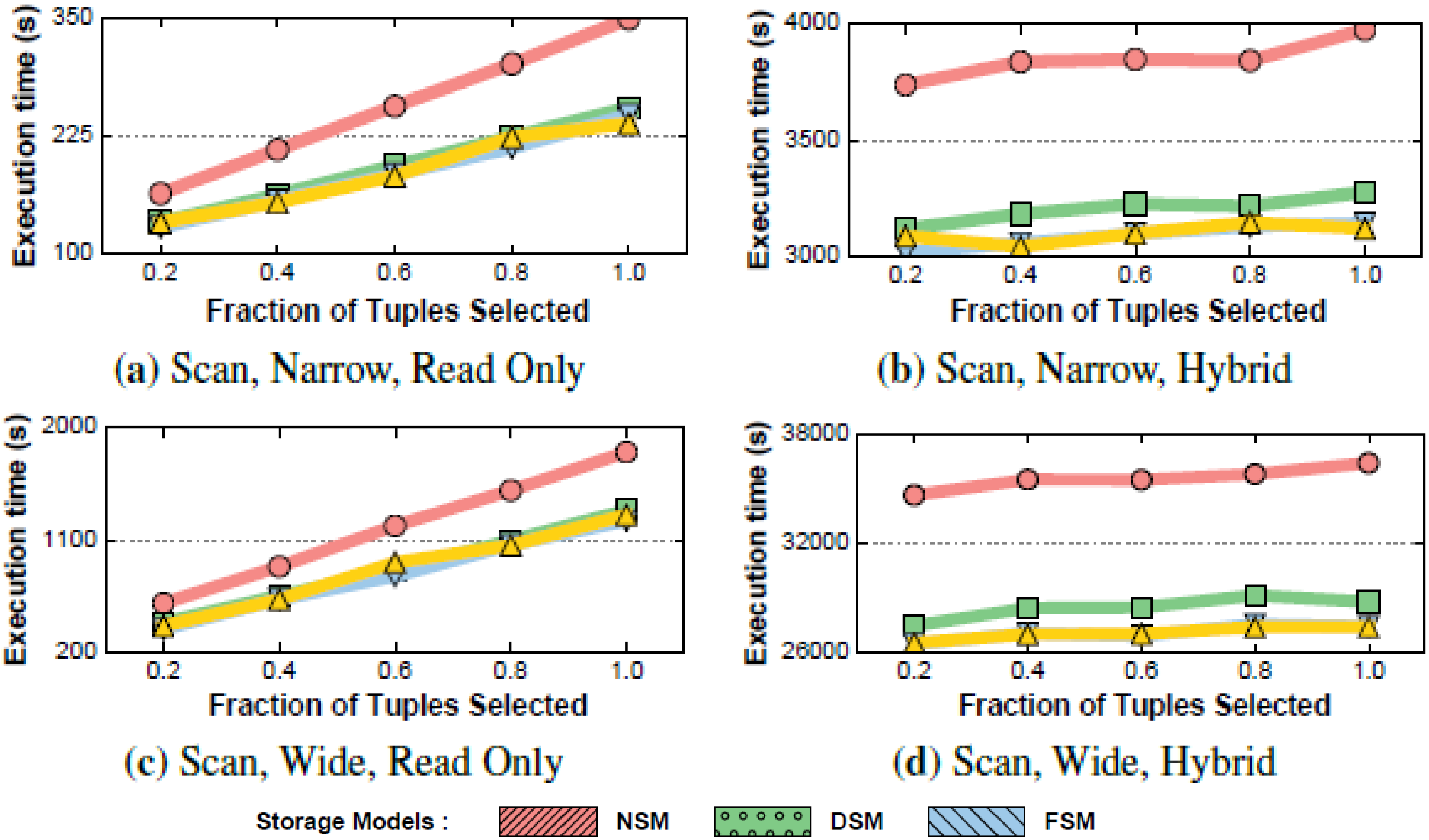
(g) Aggregate, Wide, Read Only

(h) Aggregate, Wide, Hybrid

Adaptation


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

Logical tiles



(a) Scan, Narrow, Read Only

(b) Scan, Narrow, Hybrid

(c) Scan, Wide, Read Only

(d) Scan, Wide, Hybrid

Conclusion

Conclusion

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