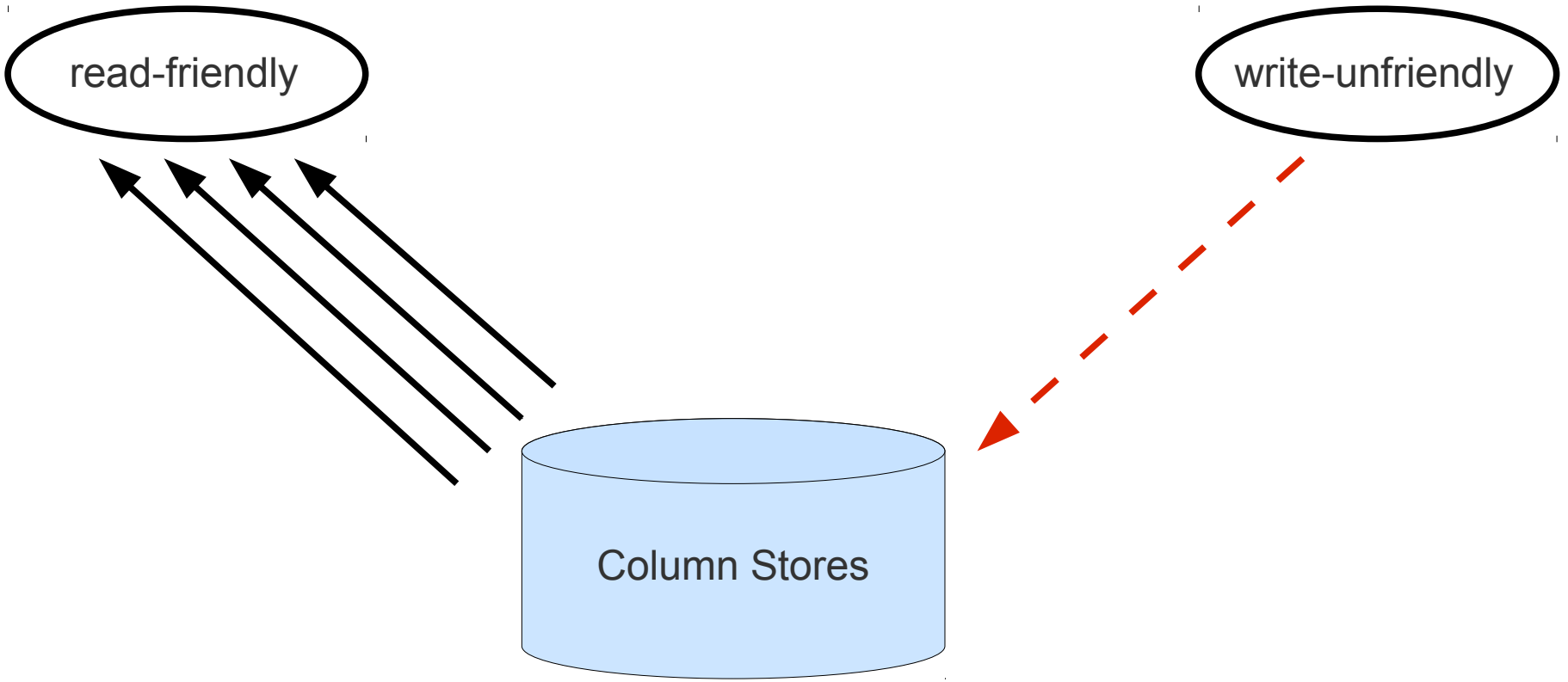
Seminar: Turn Head 90º — Column Store Databases

Positional update handling in column stores

Jonas Schnee
17.12.2010

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Motivation



- Efficent update functionality ?

Updates in column stores

- Splitting the architecture (i.e. C-Store)
  - Read-store

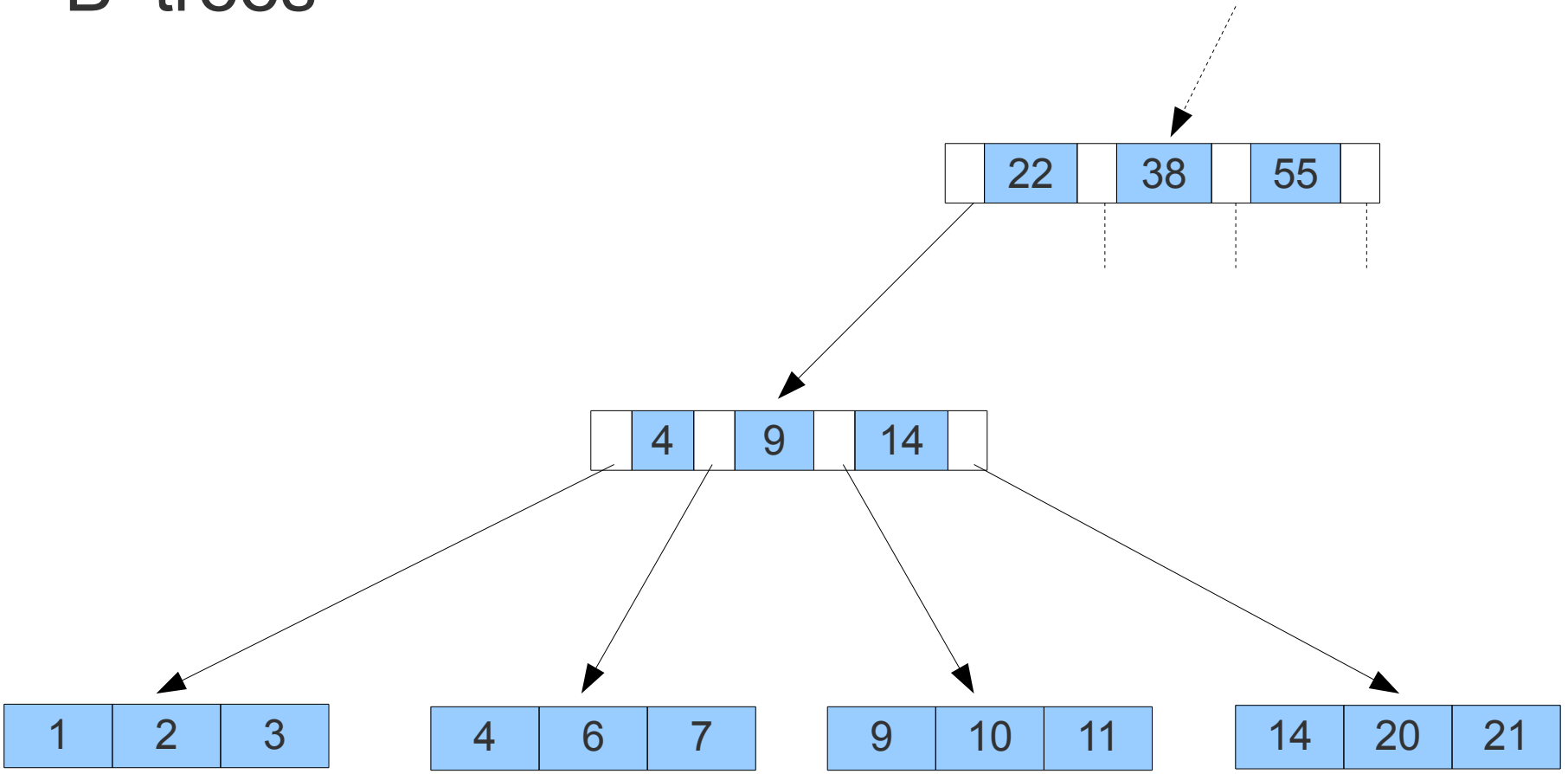
    → manages all data
  - Write-store

    → manages recent updates

- Write-store implementation
  - Value-based
  - Position-based

Detour

B$^+$-trees



Positional Delta Tree (PDT)

- New data structure:
  - Similar to ordered B-trees
  - Based on tuple positions
  - Differential update-structure
  - Difference-merge: no need to look at SK-values
    - Less I/O
    - Less CPU intensive

- Simple concept – complex realization

Positional Delta Tree (PDT)

- Stable-id (SID):

  $SID(\tau)$ *is position of tuple* τ *within the stable table*

- Row-id (RID):

  $RID(\tau)_t$ *is position of tuple* τ *at time* t

- Δ:
  - $\Delta_t(\tau) = RID(\tau)_t - SID(\tau)$
  - |inserts| - |deletes| before τ

PDT by example

$TABLE_0$:

$\overbrace{\hspace{10em}}^{Sort-Key}$ (store, product)

SID	store	product	new	qty	RID
0	London	chair	N	30	0
1	London	stool	N	10	1
2	London	table	N	20	2
3	Paris	rug	N	1	3
4	Paris	stool	N	5	4

PDT by example

Queries

→ INSERT INTO inventory VALUES ('Berlin','table','Y',10)

→ INSERT INTO inventory VALUES ('Berlin','cloth','Y',5)

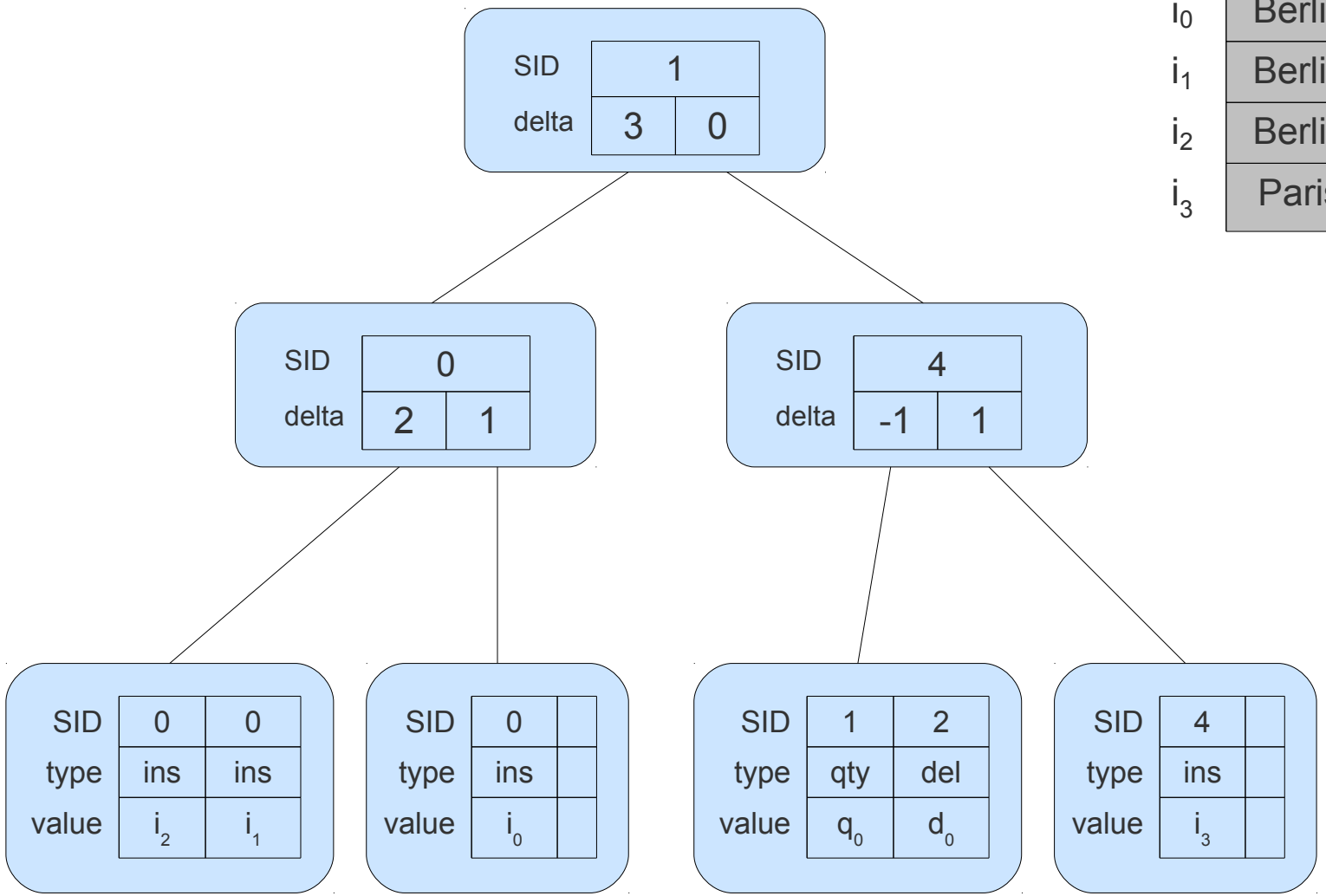
→ INSERT INTO inventory VALUES ('Berlin','chair','Y',20)

→ INSERT INTO inventory VALUES ('Paris','sofa','Y',3)

→ UPDATE inventory SET qty=13 WHERE store='London' AND
product='stool'

→ DELETE FROM inventory WHERE store='London' AND
product='table'

PDT by example



ins

store	prod	new	qty
Berlin	table	Y	10
Berlin	cloth	Y	5
Berlin	chair	Y	20
Paris	sofa	Y	3

i_0 i_1 i_2 i_3

del

store	prod
London	table

d_0

qty

qty
13

q_0

new

new

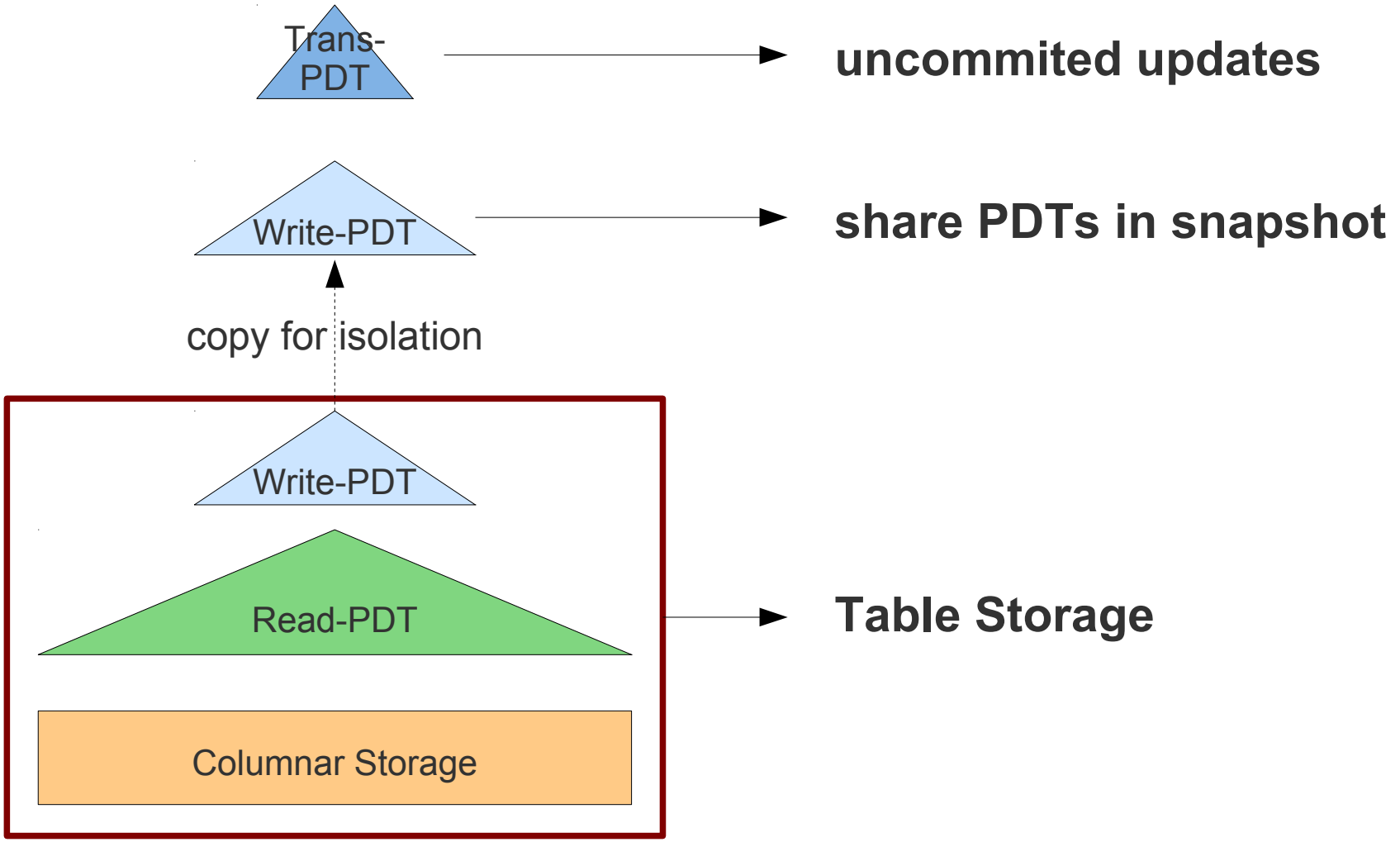
PDT by example

Table$_1$

SID	store	prod	Neu	Menge	RID
0	Berlin	chair	Y	20	0
0	Berlin	cloth	Y	5	1
0	Berlin	table	Y	10	2
0	London	chair	N	30	3
1	London	stool	N	13	4
2	London	table	N	20	5
3	Paris	rug	N	1	5
4	Paris	sofa	N	3	6
4	Paris	stool	N	5	7

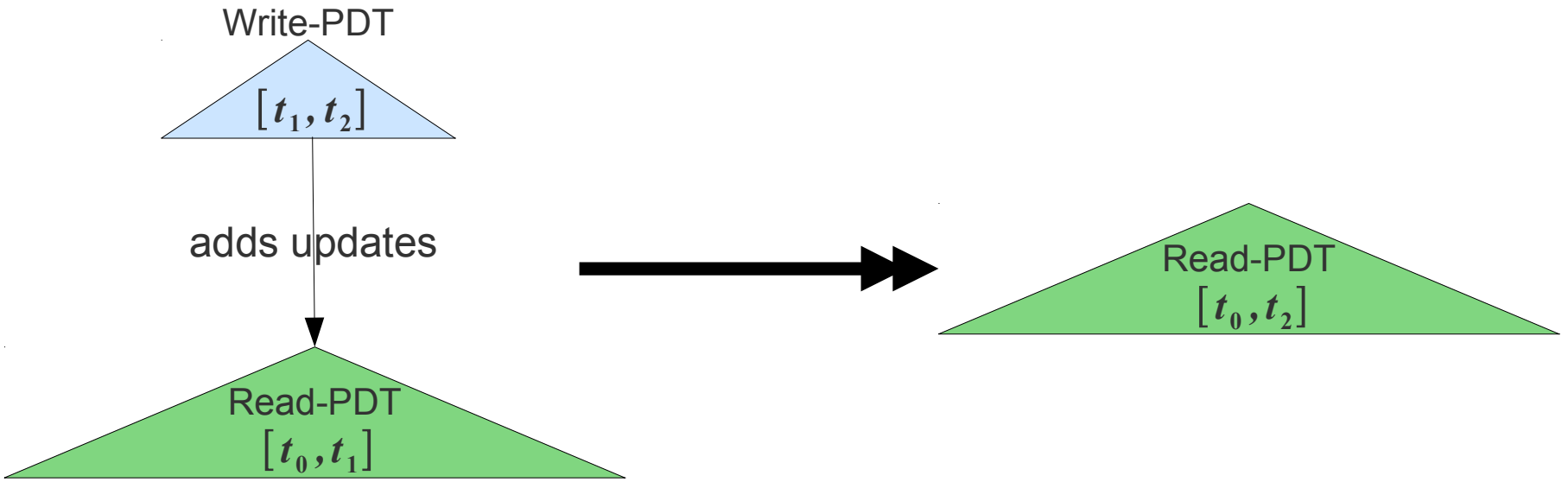
Transactions

Snapshot Isolation with Layered PDTs



Transactions

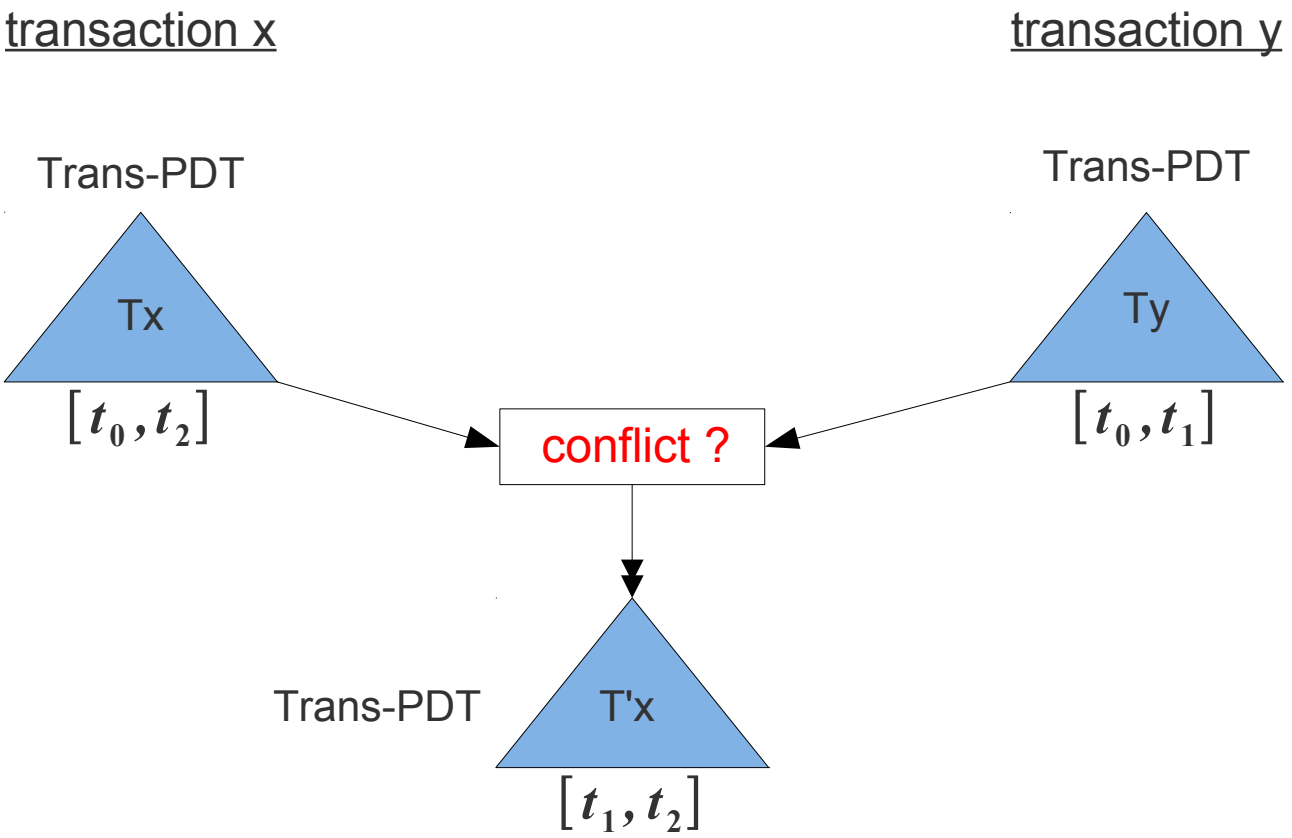
Propagate: $R_{t2}^{t0} \leftarrow R_{t1}^{t0}.Propagate(W_{t2}^{t1})$



$$TABLE_t = TABLE_0.Merge(R_r^0).Merge(W_w^r).Merge(T_t^w)$$

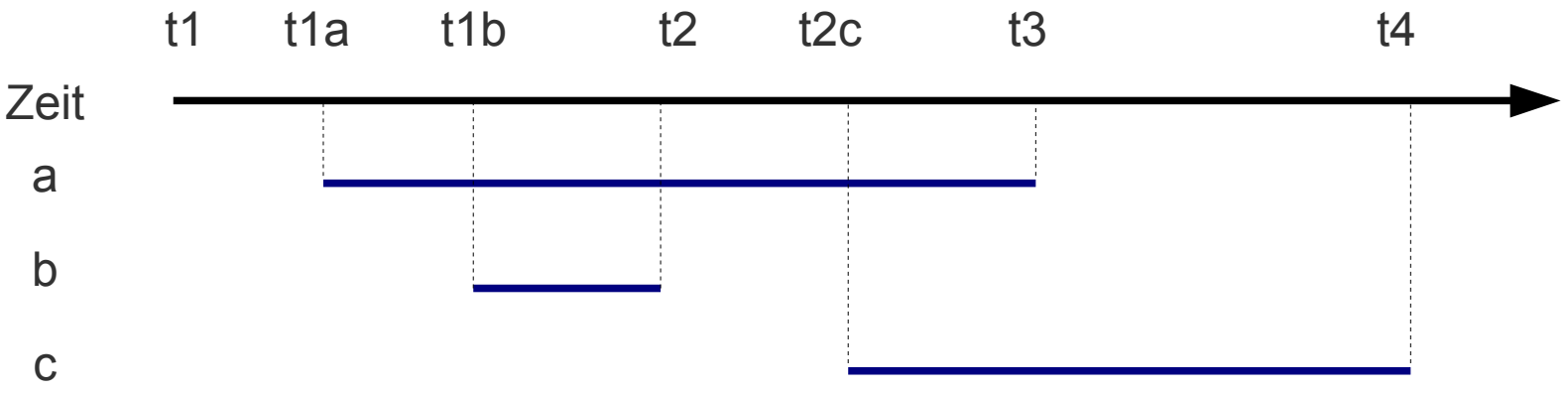
Transactions

Serialize: $T'x_{t2}^{t1} \leftarrow Tx_{t2}^{t0}.Serialize(Ty_{t1}^{t0})$

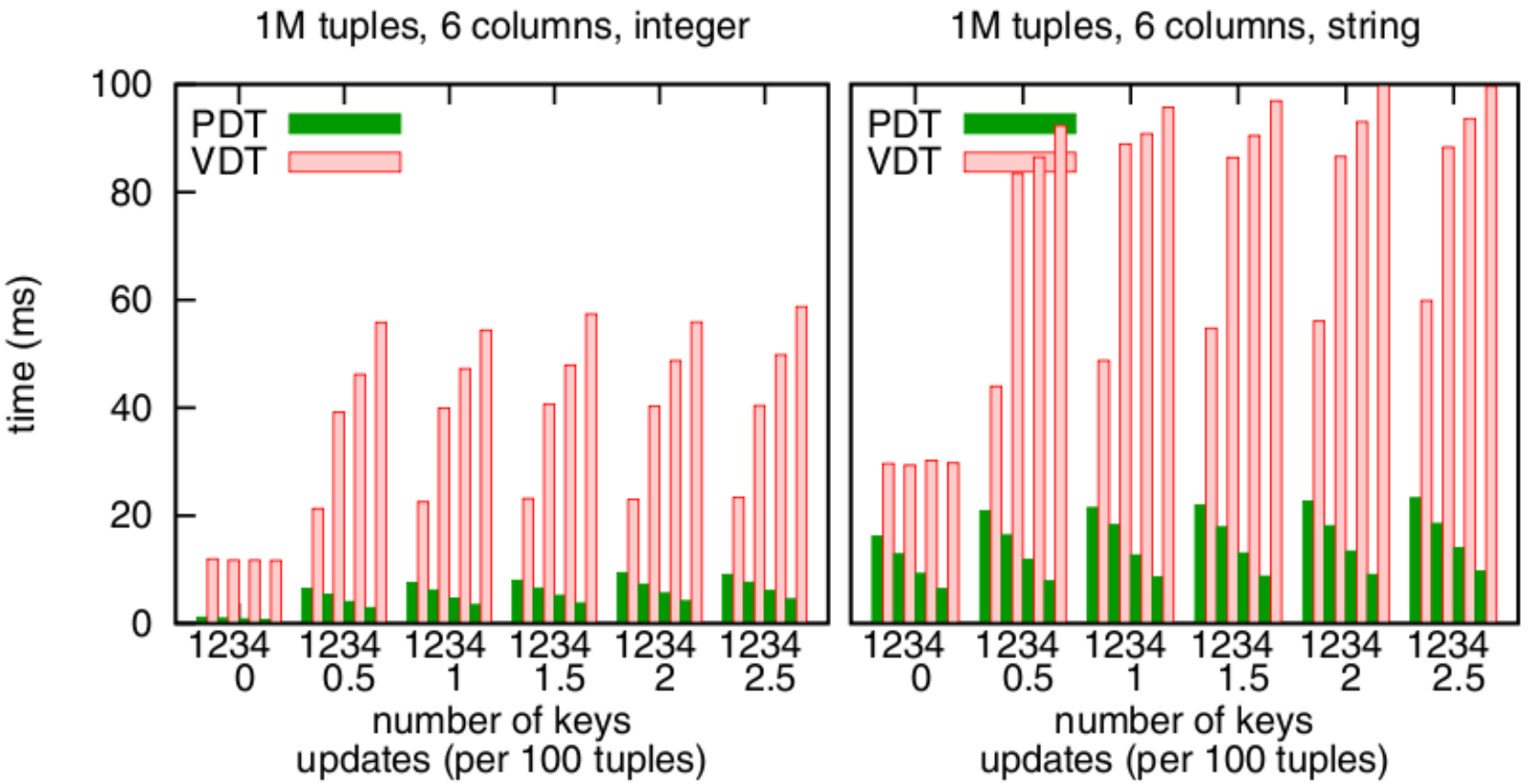


Transactions

Example:



Experimental Evalution



Summary

- New positional approach → PDT
- PDT more efficient
- Read-performance not compromised
- Efficient lock-free transaction management

- Future work:
  - Information retrieval systems
  - Flash memory
  - Row stores

Thank you for your attention

PDT Algorithmen

Blätter finden:

- (SID,RID) ist Schlüssel

PDT.FindLeafByRid(*rid*)

```
1:  node = this; δ = 0
2:  while is_leaf(node) = false do
3:      for i=0 to node_count(node) do
4:          δ = δ + node.delta[i]
5:          if rid < node.sids[i] + δ then
6:              δ = δ - node.delta[i]
7:              break from inner loop
8:      node = node.child[i]
9:  return (node,δ)
```

PDT Algorithmen

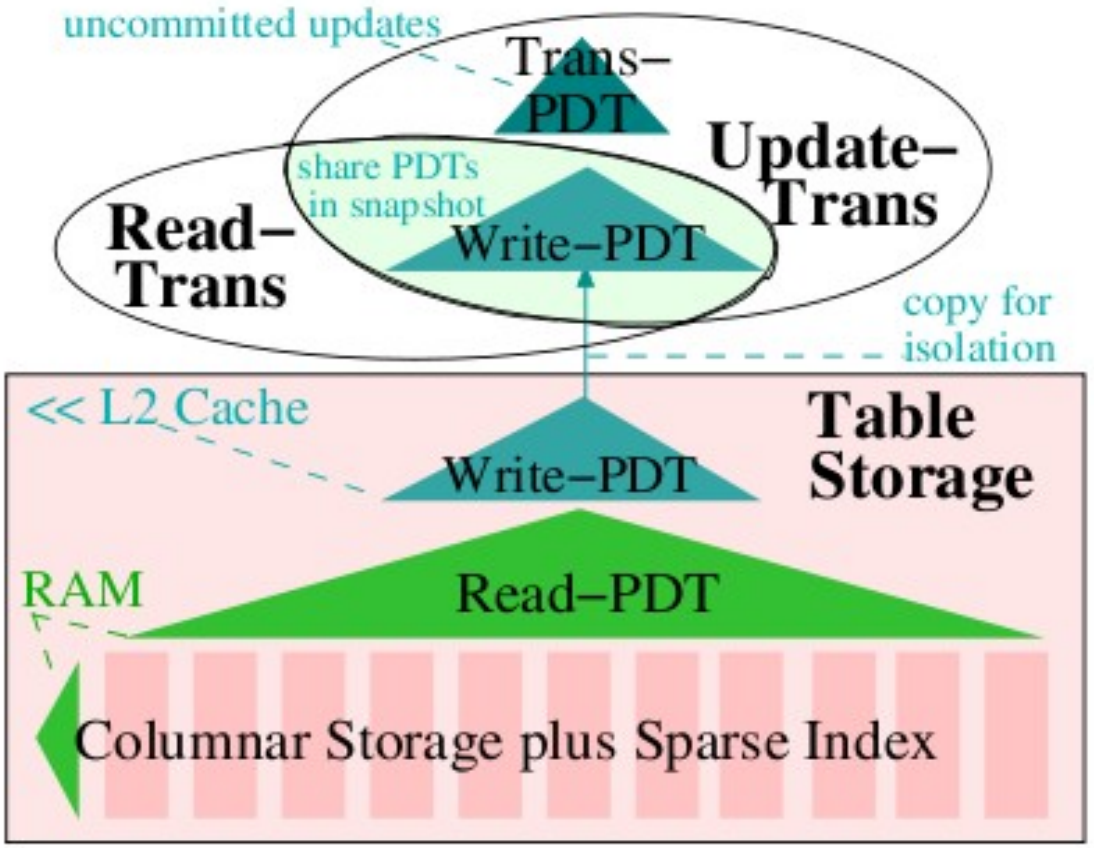
Einfügen von Tupel

PDT.AddInsert(*sid,rid,tuple*)

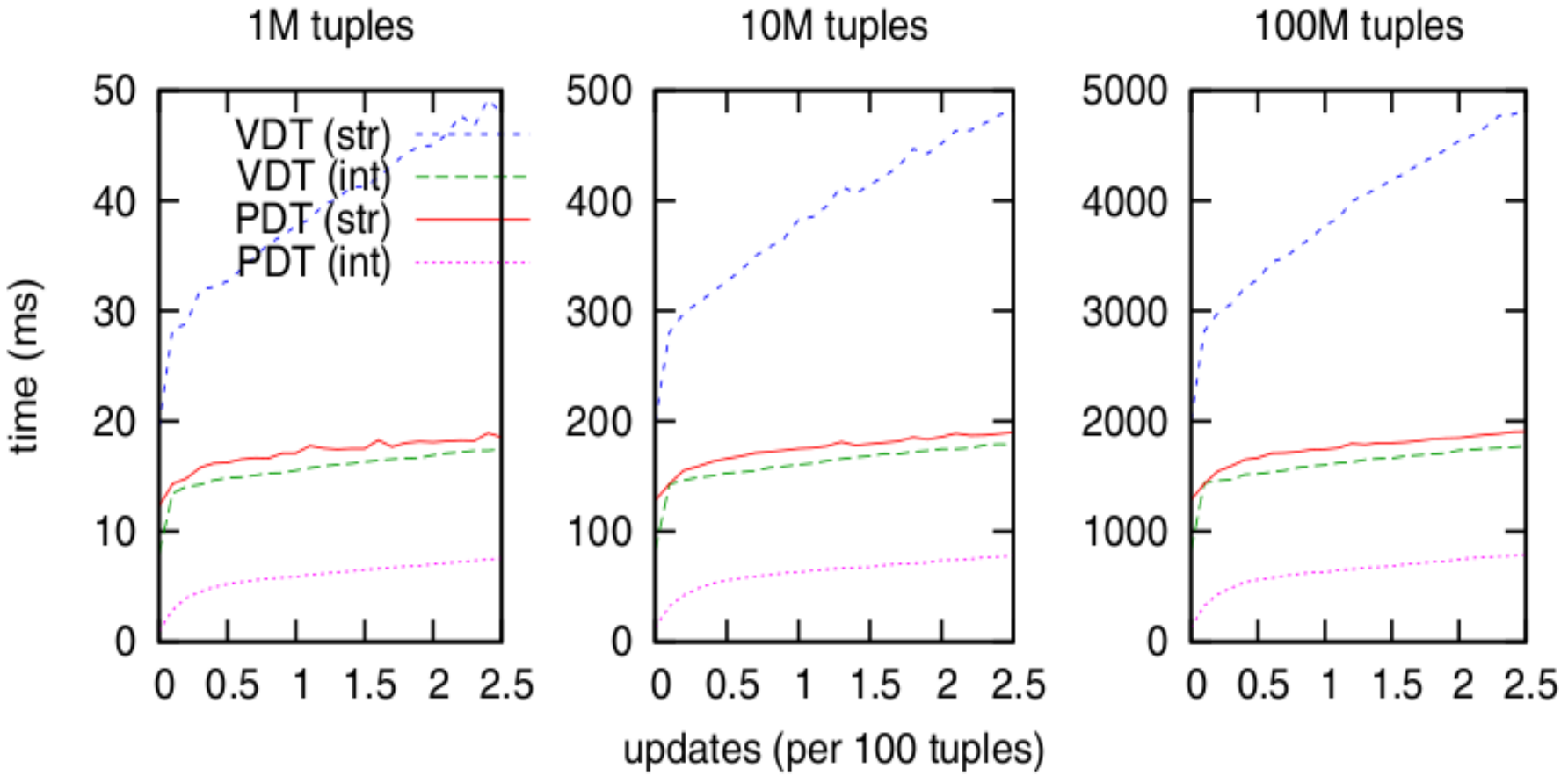
```
1:    (leaf,δ) = this.FindLeafBySidRid(sid,rid)
2:    while leaf.sid[pos] < sid or leaf.sid[pos] + δ < rid  do
3:        if leaf.type[pos] ≡ INS then
4:            δ = δ + 1
5:        else if leaf.type[pos] ≡ DEL then
6:            δ = δ - 1
7:        pos = pos + 1
8:    this.ShiftLeafEntries(leaf,pos,1)
9:    leaf.type[pos] = INS
10:   leaf.sid[pos] = rid - δ
11:   offset = this.AddToInsertSpace(tuple)
12:   leaf.value[pos] = offset
13:   this.AddNodeDeltas(leaf,1)
```

Transaktionen

Gestapelte PDTs als Transaktionsbaustein



Experimental Evaluation



Positional Delta Tree (PDT)

- Definition:

$$MINUS_{t2}^{t1} = \{ \tau \in TABLE_{t1} : \neg \exists \gamma \in TABLE_{t2} : \tau . SK = \gamma . SK \}$$

- Dann gilt:

$$\Delta_{t2}^{t1}(\tau) = |\{ \gamma \in MINUS_{t1}^{t2} : RID(\gamma)_{t2} < RID(\tau)_{t2} \}| - |\{ \gamma \in MINUS_{t2}^{t1} : SID(\gamma) < SID(\tau) \}|$$

#Inserts - #Deletes