

# Seminar: Turn Head 90° — Column Store Databases

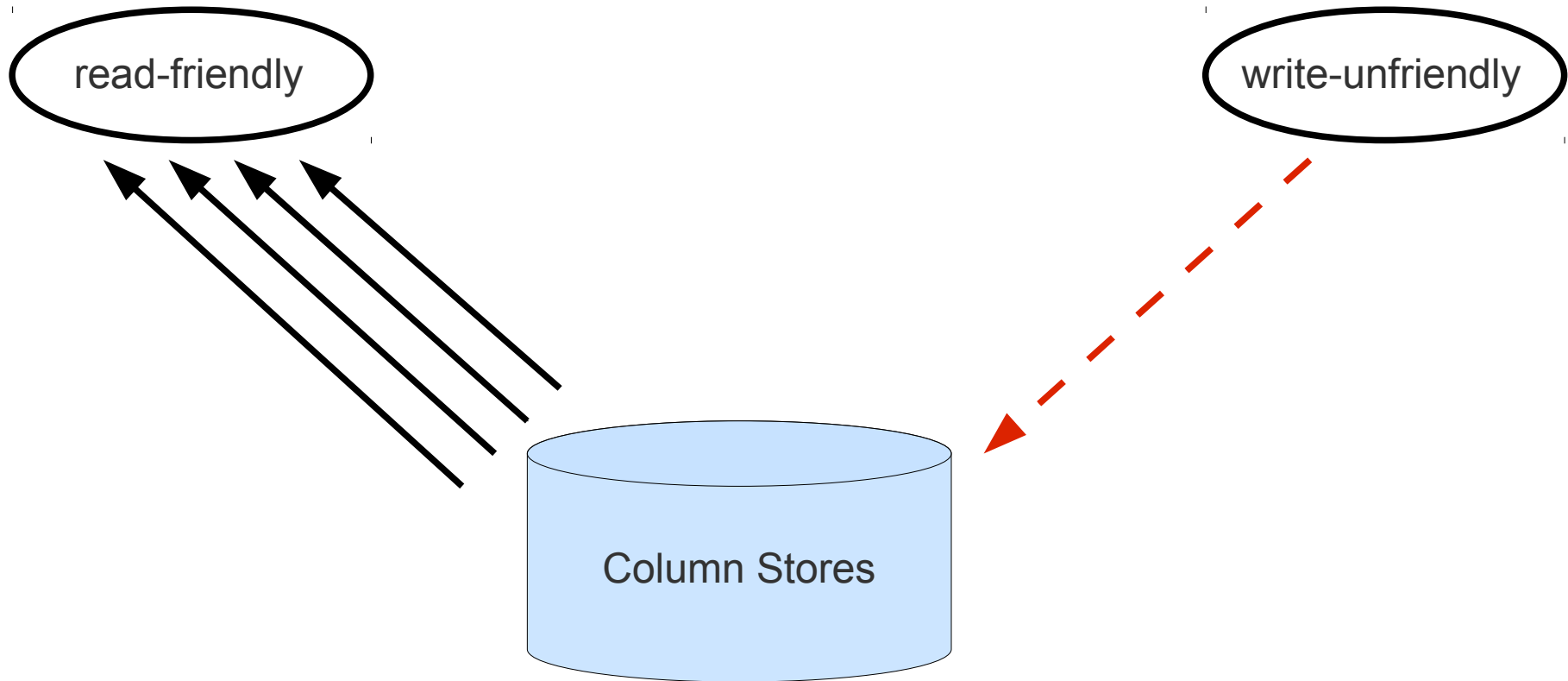
## Positional update handling in column stores

Jonas Schnee  
17.12.2010

EBERHARD KARLS  
UNIVERSITÄT  
TÜBINGEN



# Motivation



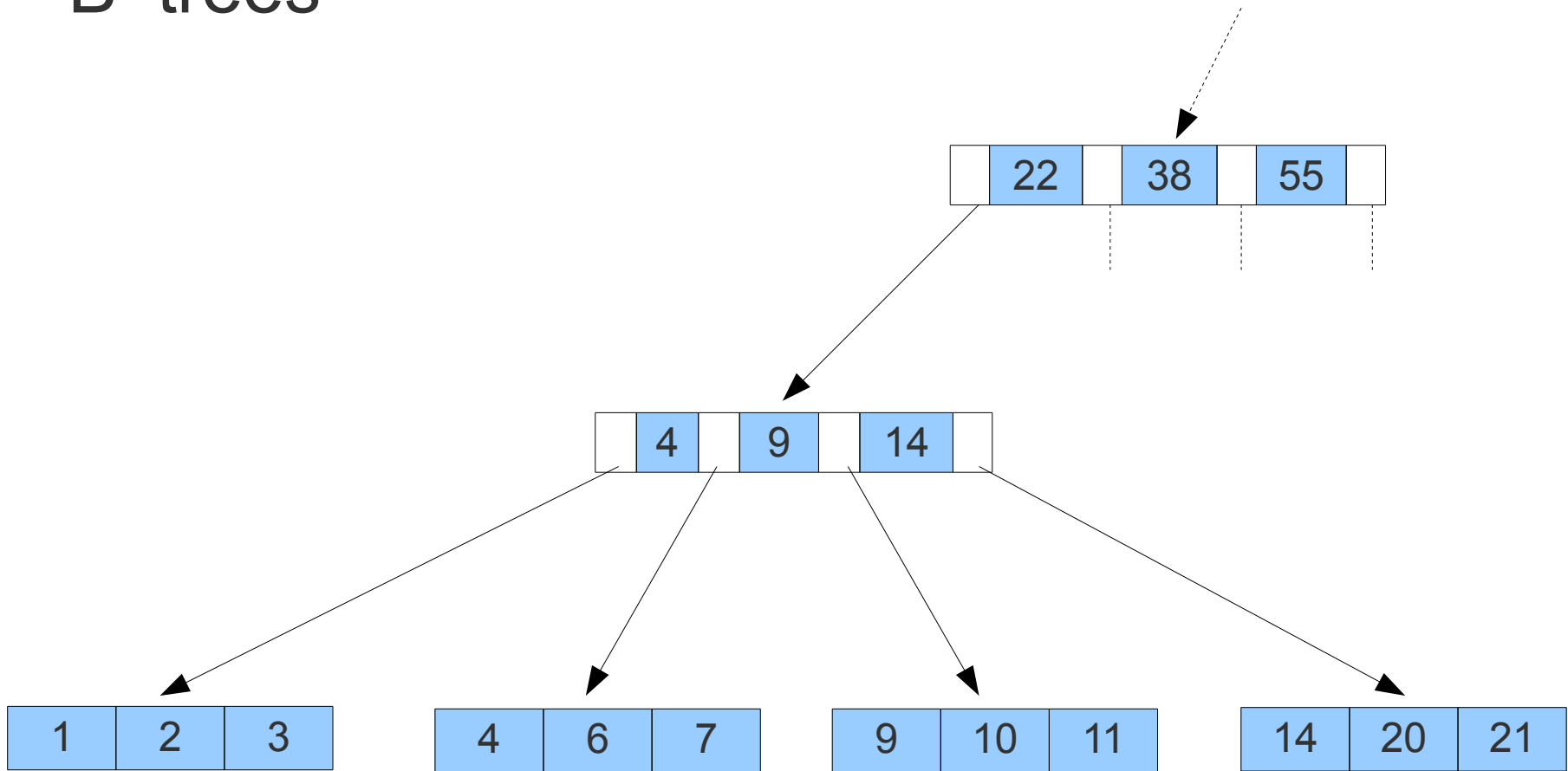
→ Efficient update functionality ?

# 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<sup>+</sup>-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  $\tau$  within the stable table

- Row-id (RID):

*RID*( $\tau$ )<sub>*t*</sub> is position of tuple  $\tau$  at time *t*

- $\Delta$ :

- $\Delta_t(\tau) = RID(\tau)_t - SID(\tau)$

- |inserts| - |deletes| before  $\tau$

# PDT by example

TABLE<sub>0</sub>:

*Sort – Key*

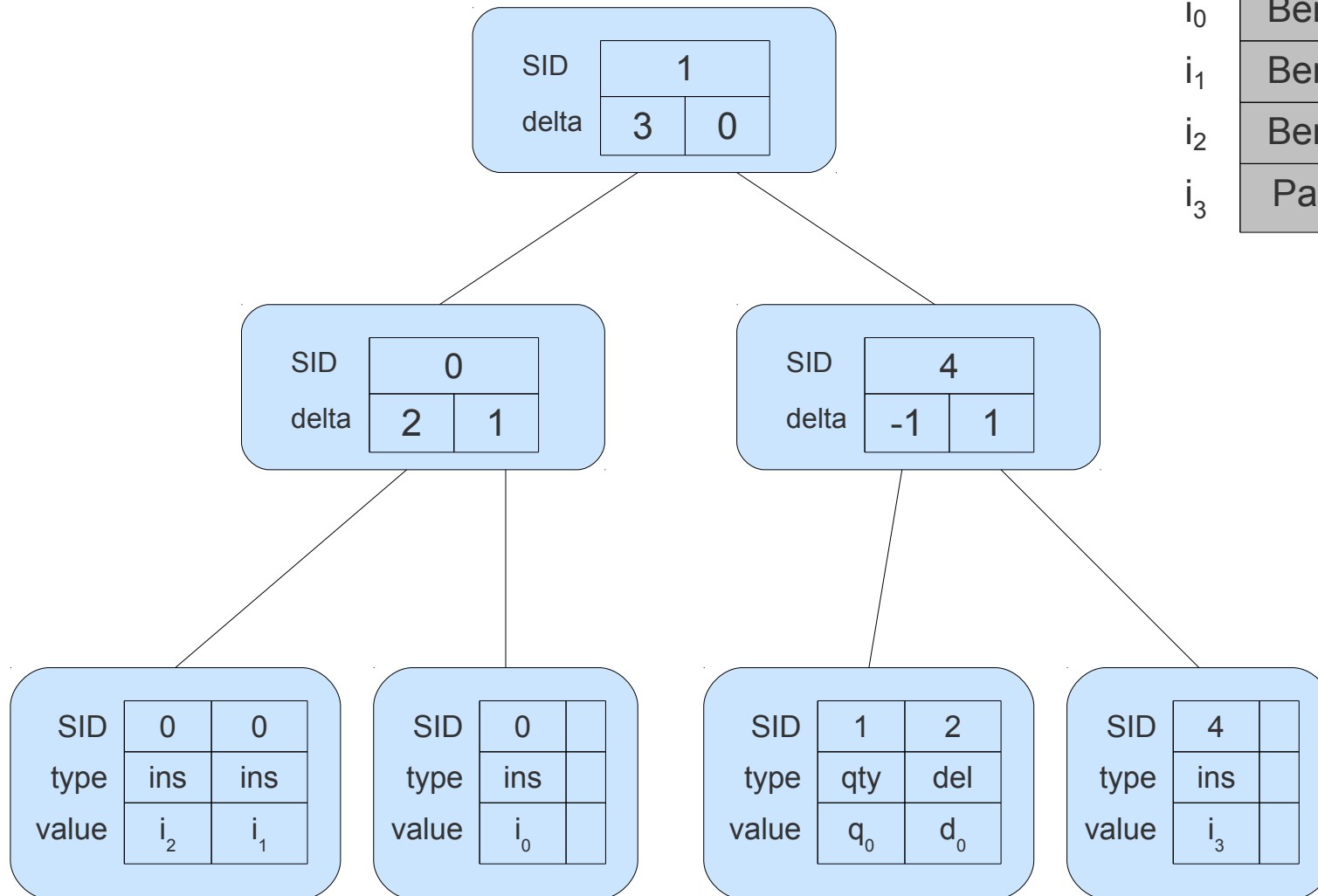
SID	store	product	new	qty	RID
0	London	chair	N	30	0
1	London	stool	N	10	1
2	London	table	N	20	2
3	Paris	rug	N	1	3
4	Paris	stool	N	5	4

# 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
$i_0$	Berlin	table	Y	10
$i_1$	Berlin	cloth	Y	5
$i_2$	Berlin	chair	Y	20
$i_3$	Paris	sofa	Y	3

del	store	prod
$d_0$	London	table

qty	qty
$q_0$	13

new	new

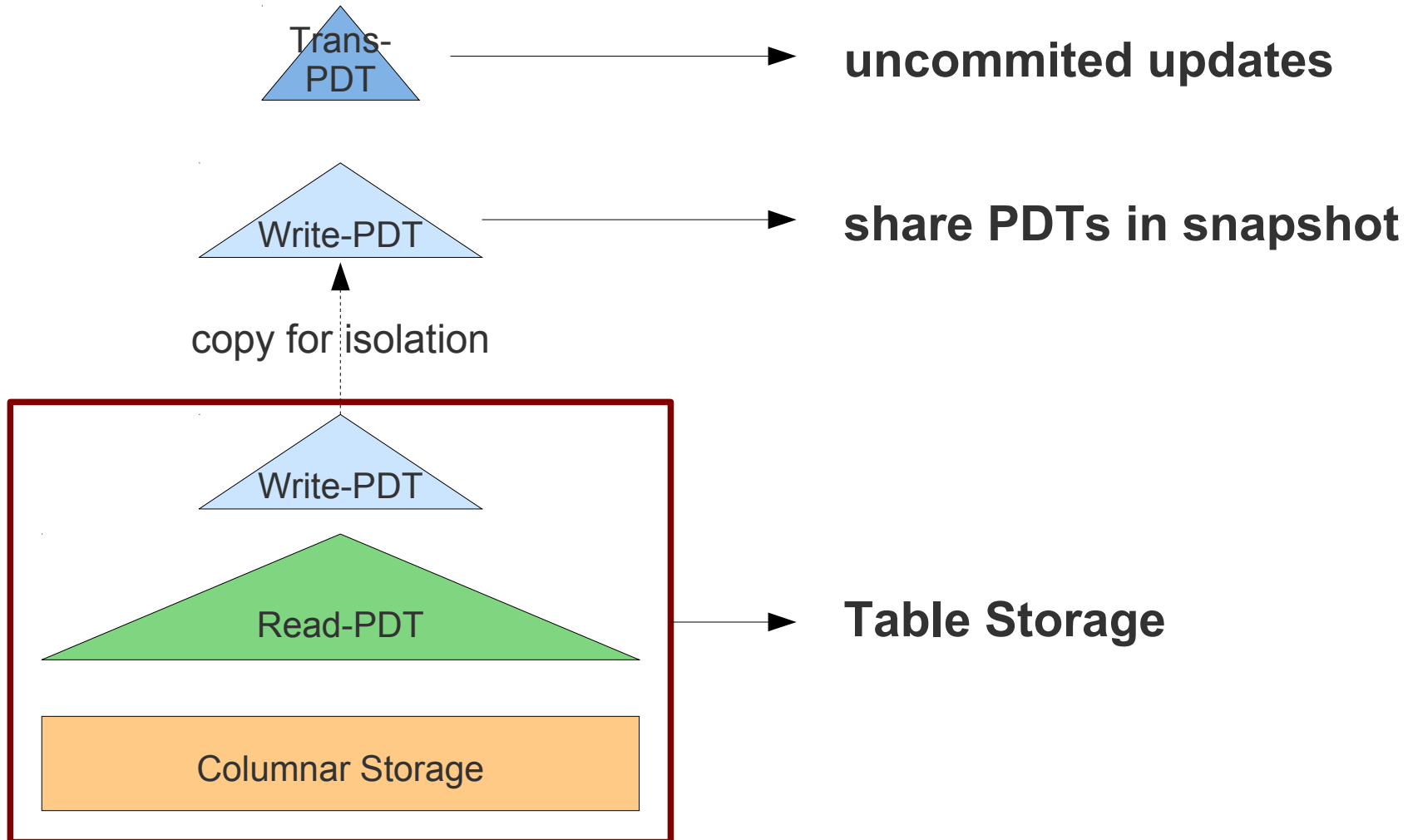
# PDT by example

Table<sub>1</sub>

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

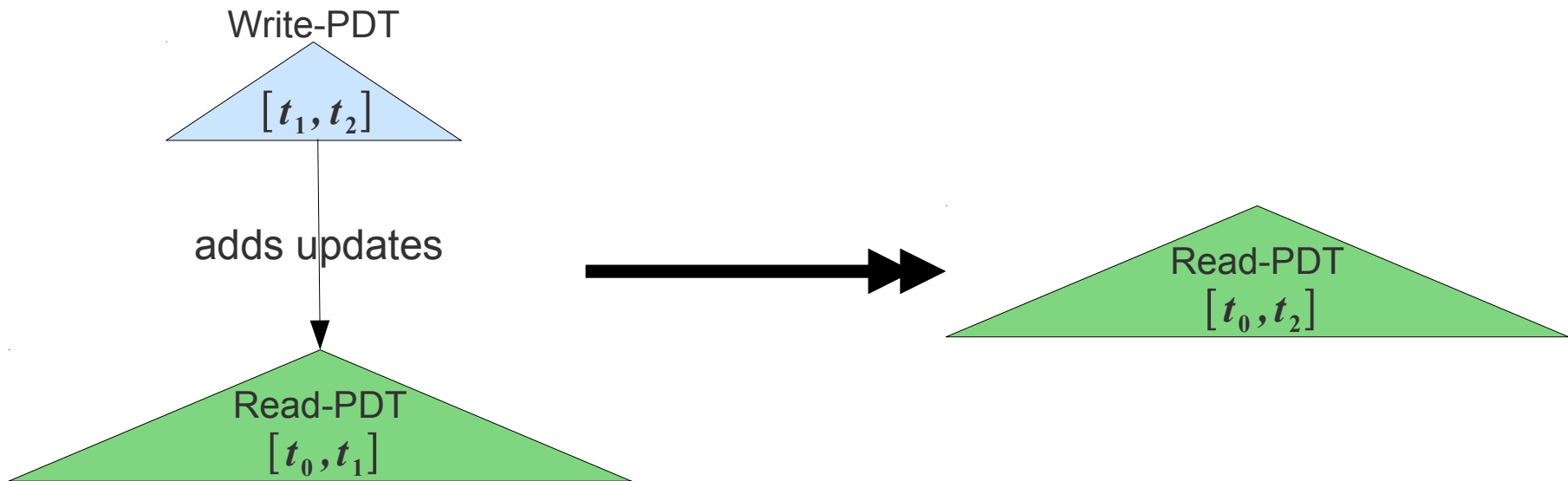
# Transactions

## Snapshot Isolation with Layered PDTs



# Transactions

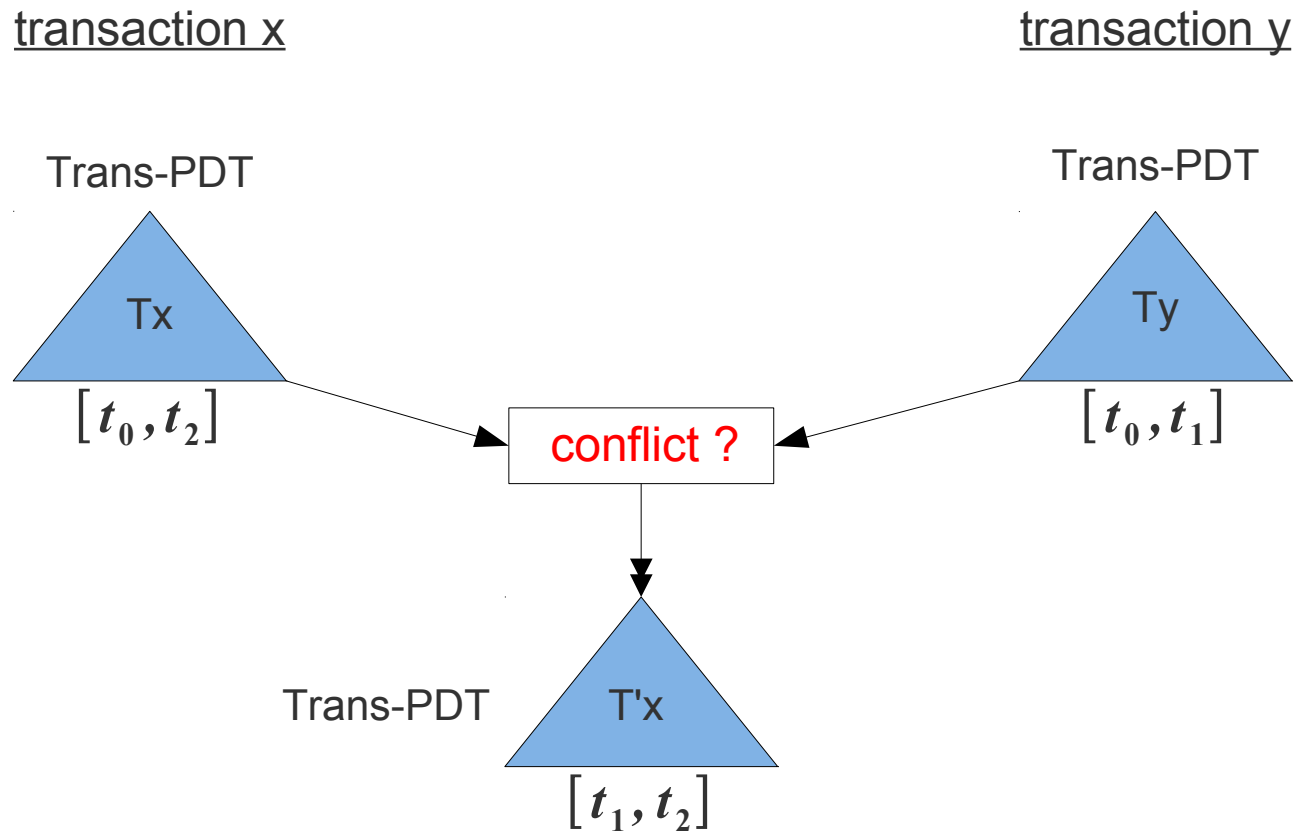
Propagate:  $R_{t_2}^{t_0} \leftarrow R_{t_1}^{t_0} . \text{Propagate}(W_{t_2}^{t_1})$



$$TABLE_t = TABLE_0 . \text{Merge}(R_r^0) . \text{Merge}(W_w^r) . \text{Merge}(T_t^w)$$

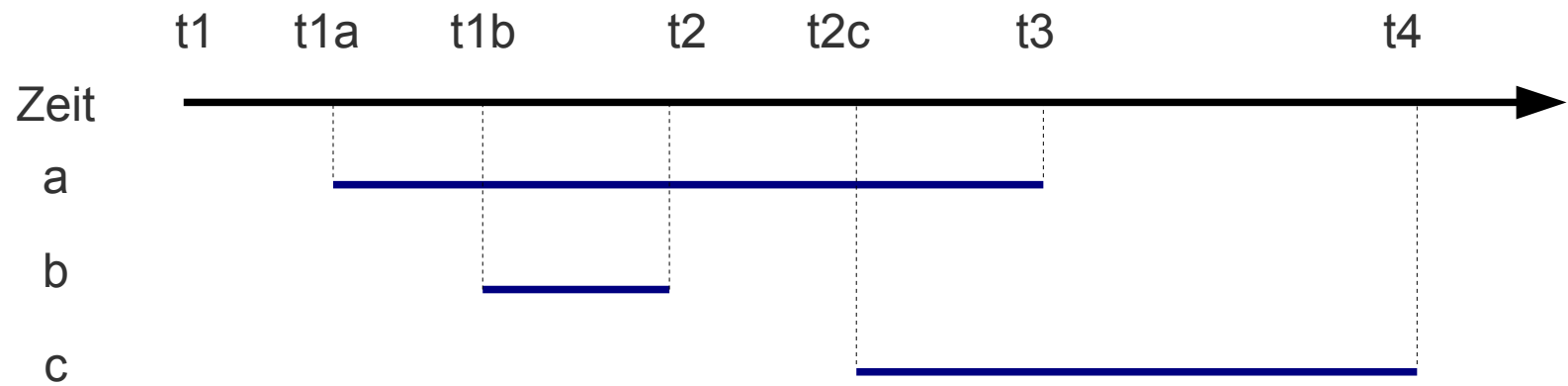
# Transactions

Serialize:  $T'x_{t_2}^{t_1} \leftarrow Tx_{t_2}^{t_0}.Serialize(Ty_{t_1}^{t_0})$

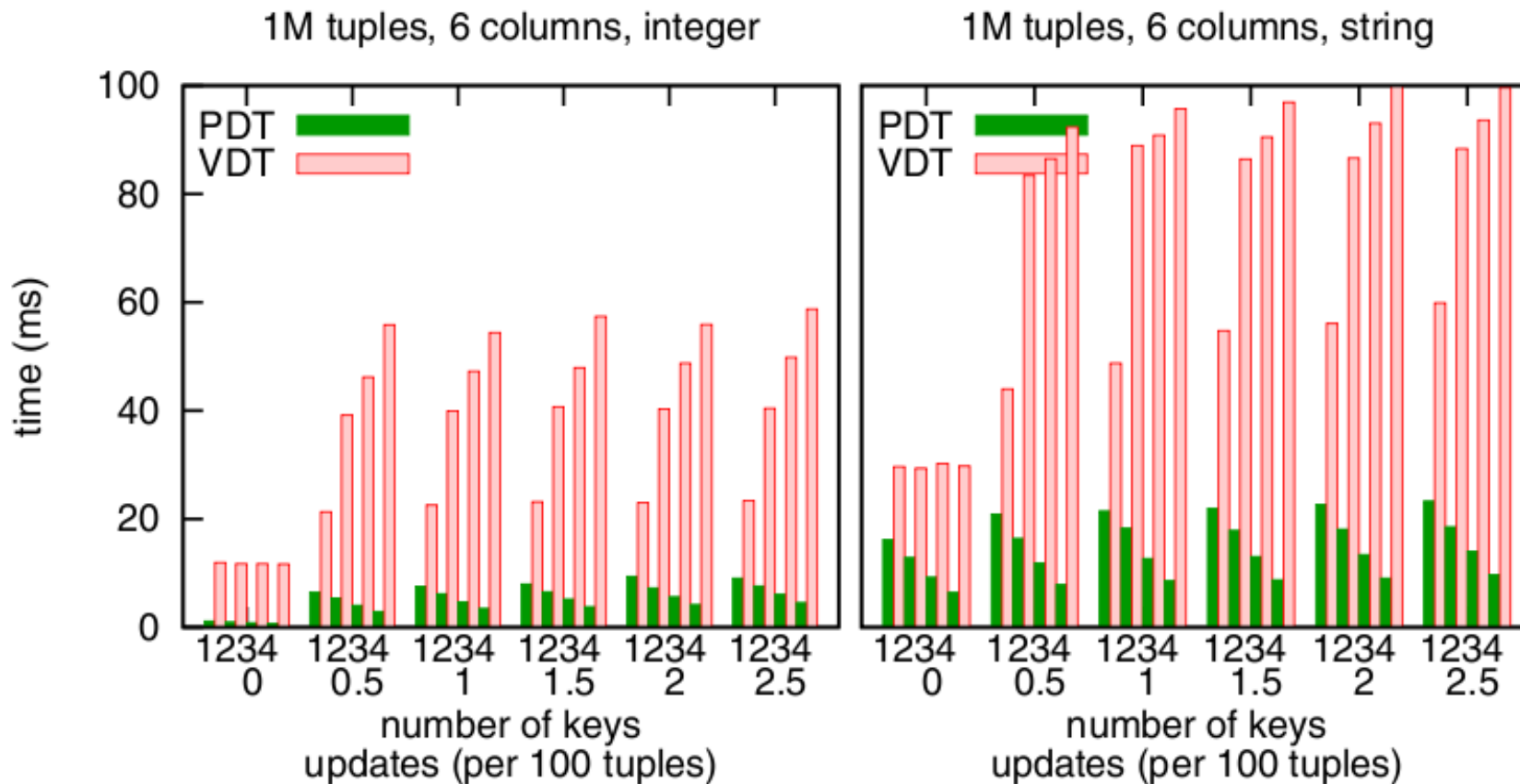


# Transactions

Example:



# Experimental Evaluation



# 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

# Seminar: Turn Head 90° — Column Store Databases

**Thank you for your attention**

# PDT Algorithmen

## Blätter finden:

→ (SID,RID) ist Schlüssel

### PDT.FindLeafByRid(*rid*)

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

# PDT Algorithmen

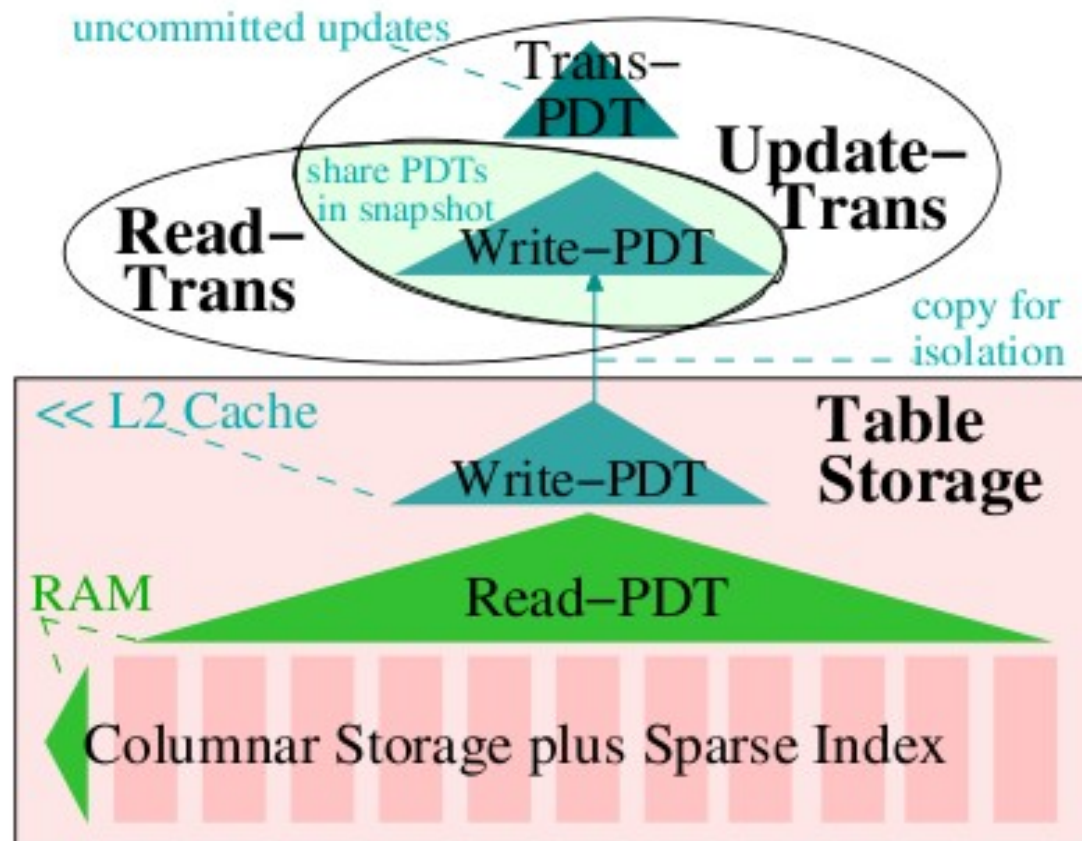
## Einfügen von Tupel

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

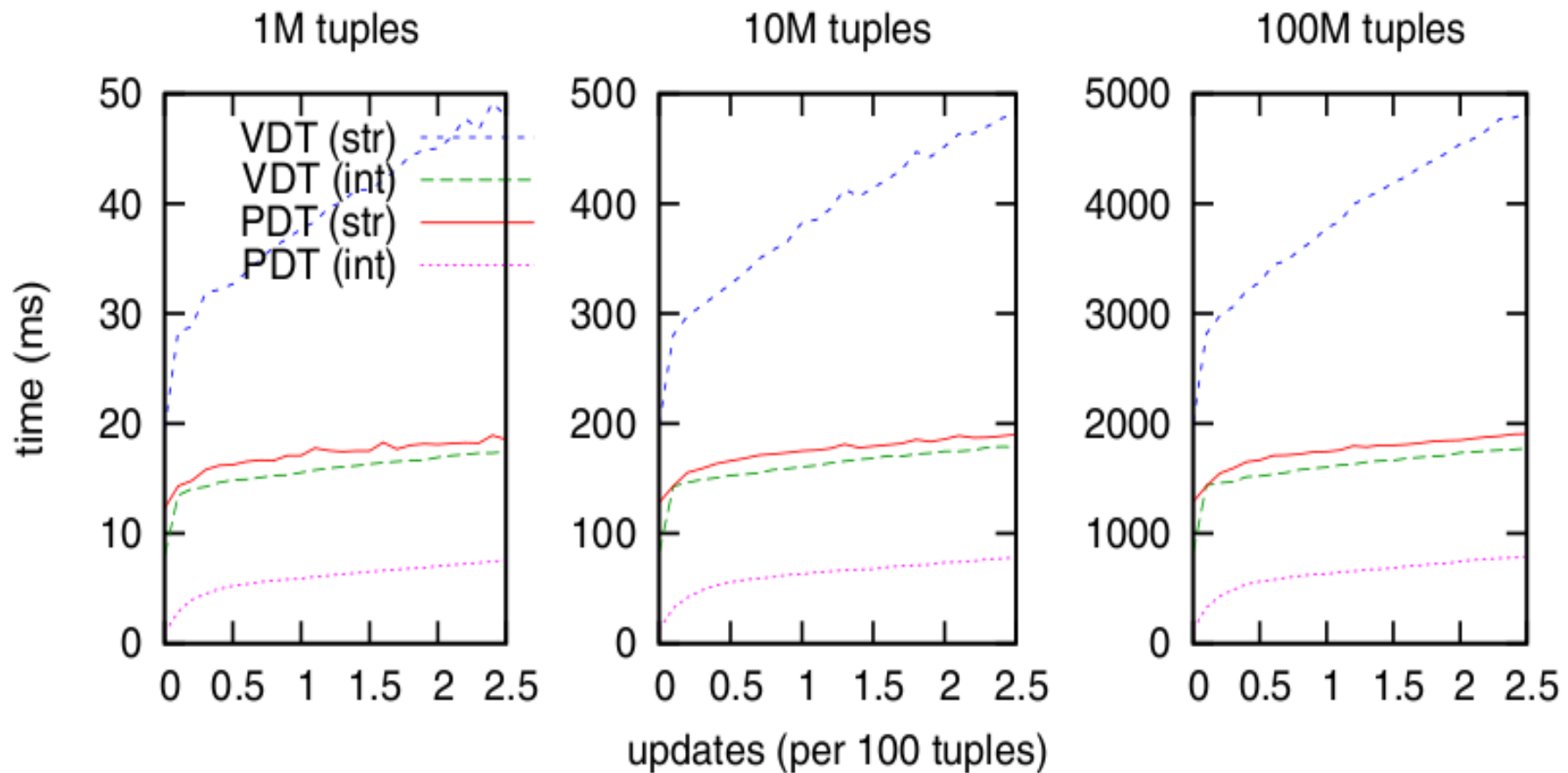
```
1:   (leaf,δ) = this.FindLeafBySidRid(sid,rid)
2:   while leaf.sid[pos] < sid or leaf.sid[pos] + δ < rid do
3:     if leaf.type[pos] ≡ INS then
4:        $\delta = \delta + 1$ 
5:     else if leaf.type[pos] ≡ DEL then
6:        $\delta = \delta - 1$ 
7:       pos = pos + 1
8:     this.ShiftLeafEntries(leaf,pos,1)
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:

$$MINUS_{t_2}^{t_1} = \{ \tau \in TABLE_{t_1} : \neg \exists \gamma \in TABLE_{t_2} : \tau.SK = \gamma.SK \}$$

- Dann gilt:

$$\Delta_{t_2}^{t_1}(\tau) = |\{ \gamma \in MINUS_{t_2}^{t_1} : RID(\gamma)_{t_2} < RID(\tau)_{t_2} \}| - |\{ \gamma \in MINUS_{t_2}^{t_1} : SID(\gamma) < SID(\tau) \}|$$

#Inserts - #Deletes