

Hauptspeicher- Datenbanksysteme

- **Hardware-Entwicklungen**
- **Column- versus Row-Store**
- **...**

Hauptspeicher-Datenbanksysteme

- Disk is Tape, Tape is dead ... Jim Gray
- Die Zeit ist reif für ein Re-engineering der Datenbanksysteme
- Man kann heute für 25000 Euro einen Datenbankserver mit 1 TeraByte Hauptspeicher und 32 Rechenkernen kaufen

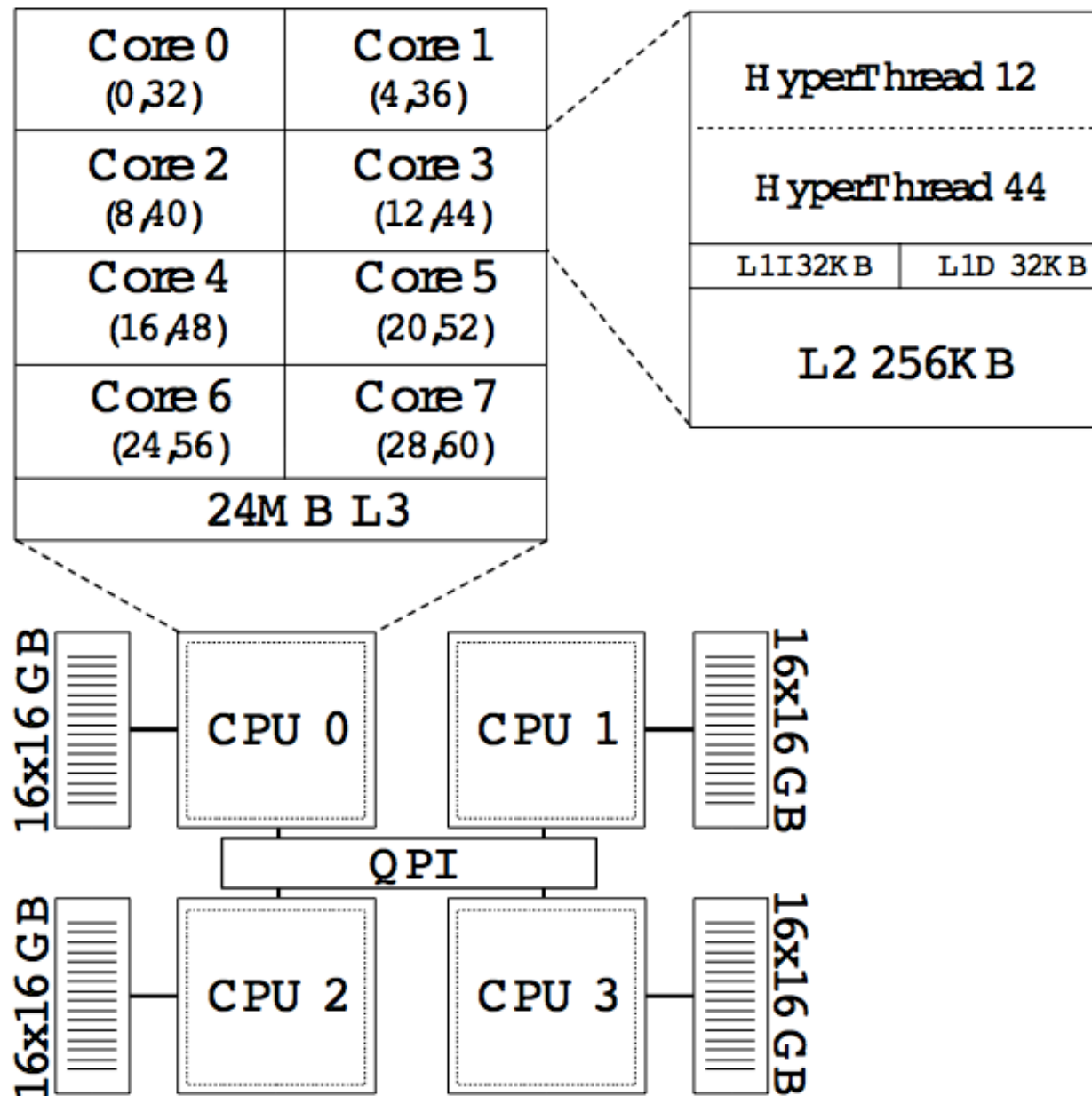
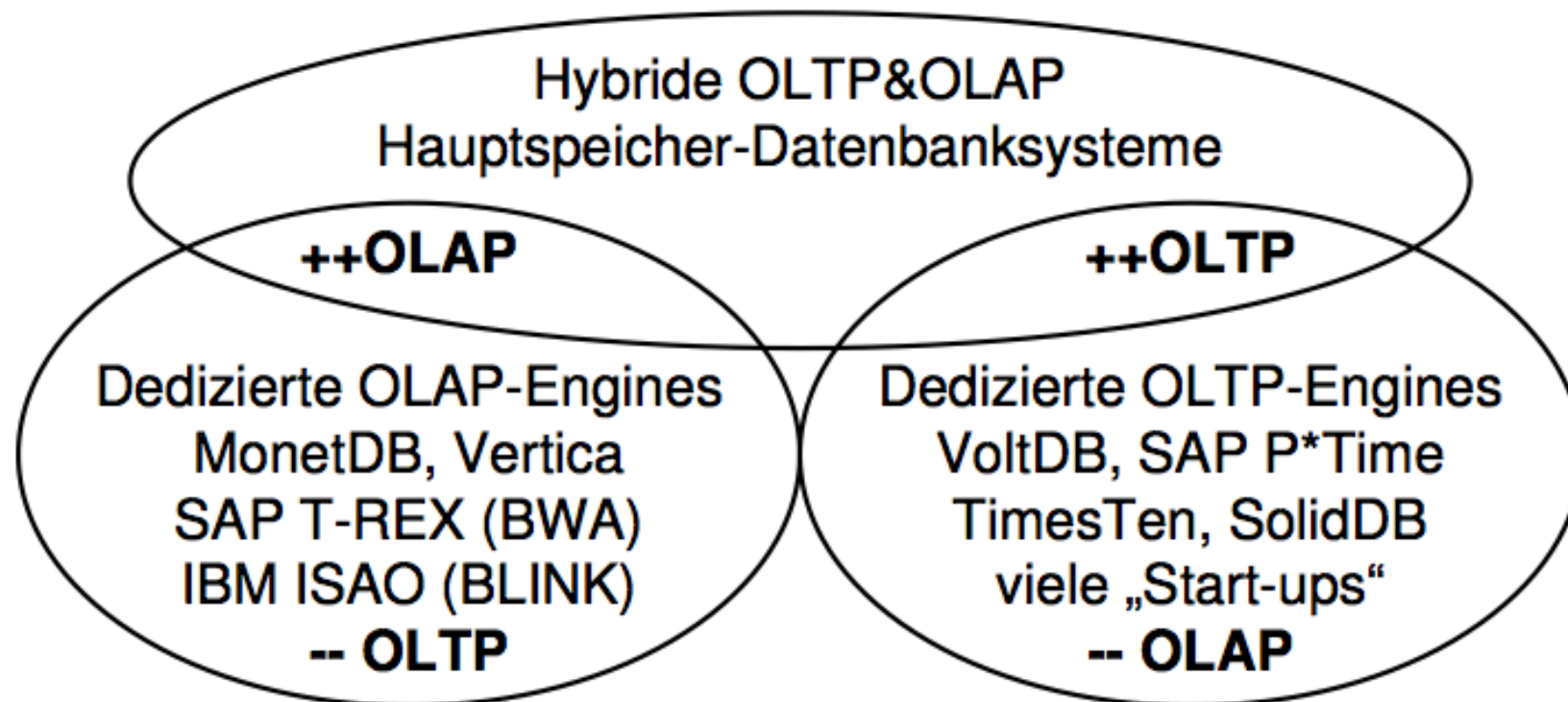


Abbildung 18.1: Architektur eines Mehrkern-Rechners mit NUMA-Hauptspeicher

Einsatz von Hauptspeicher-Datenbanksystemen



Feasibility: Main Memory DBMS

● Amazon

● Data Volume

- Revenue: 15 billion Euro
- Avg. Item Price: 15 Euro
- 1 billion order lines per year
 - 54 Bytes per order line
 - 54 GB per year
 - + additional data
 - - compression

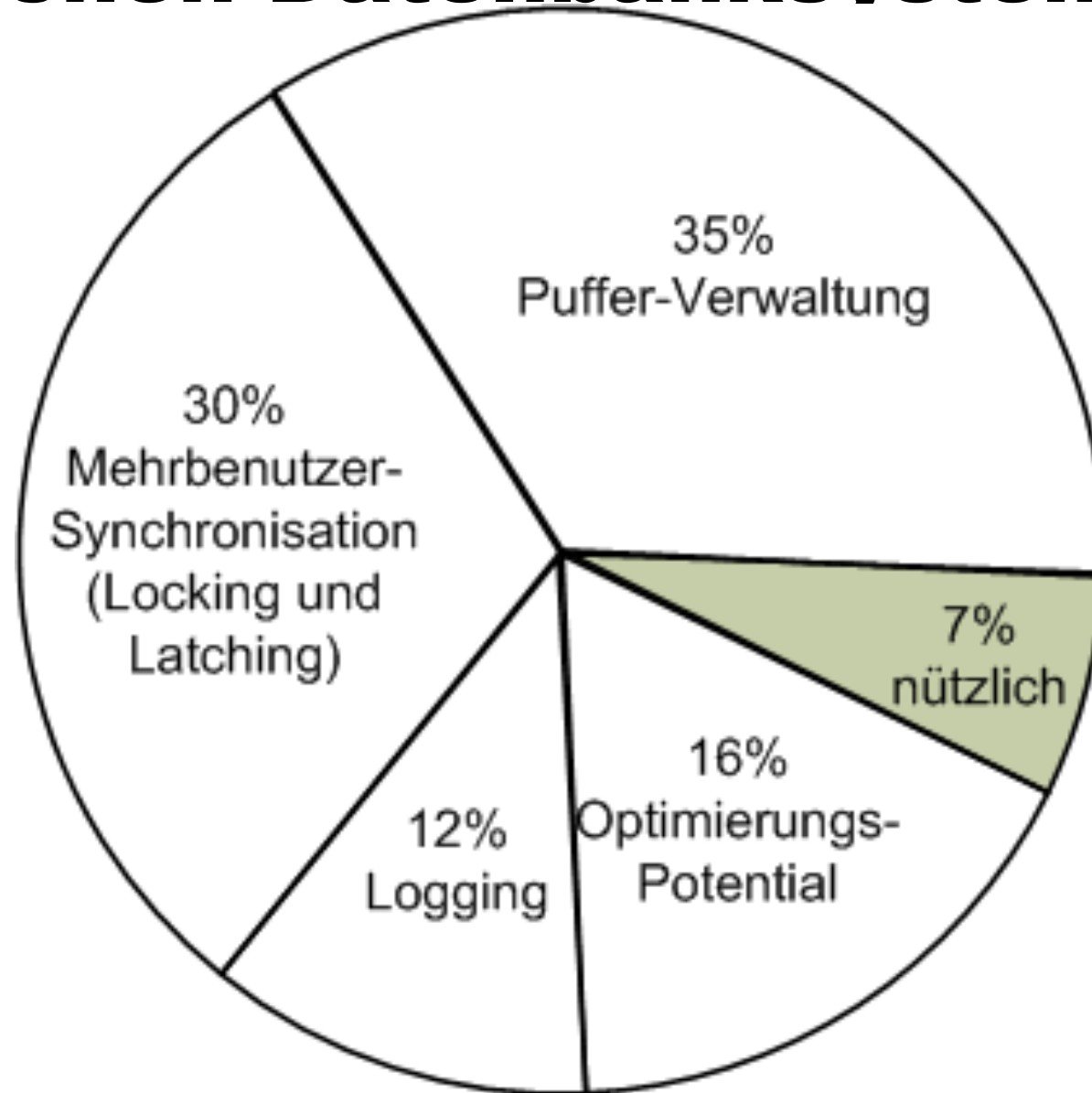
● Transaction Rate

- Avg: 32 orders per s
- Peak rate: Thousands/s
- + inquiries

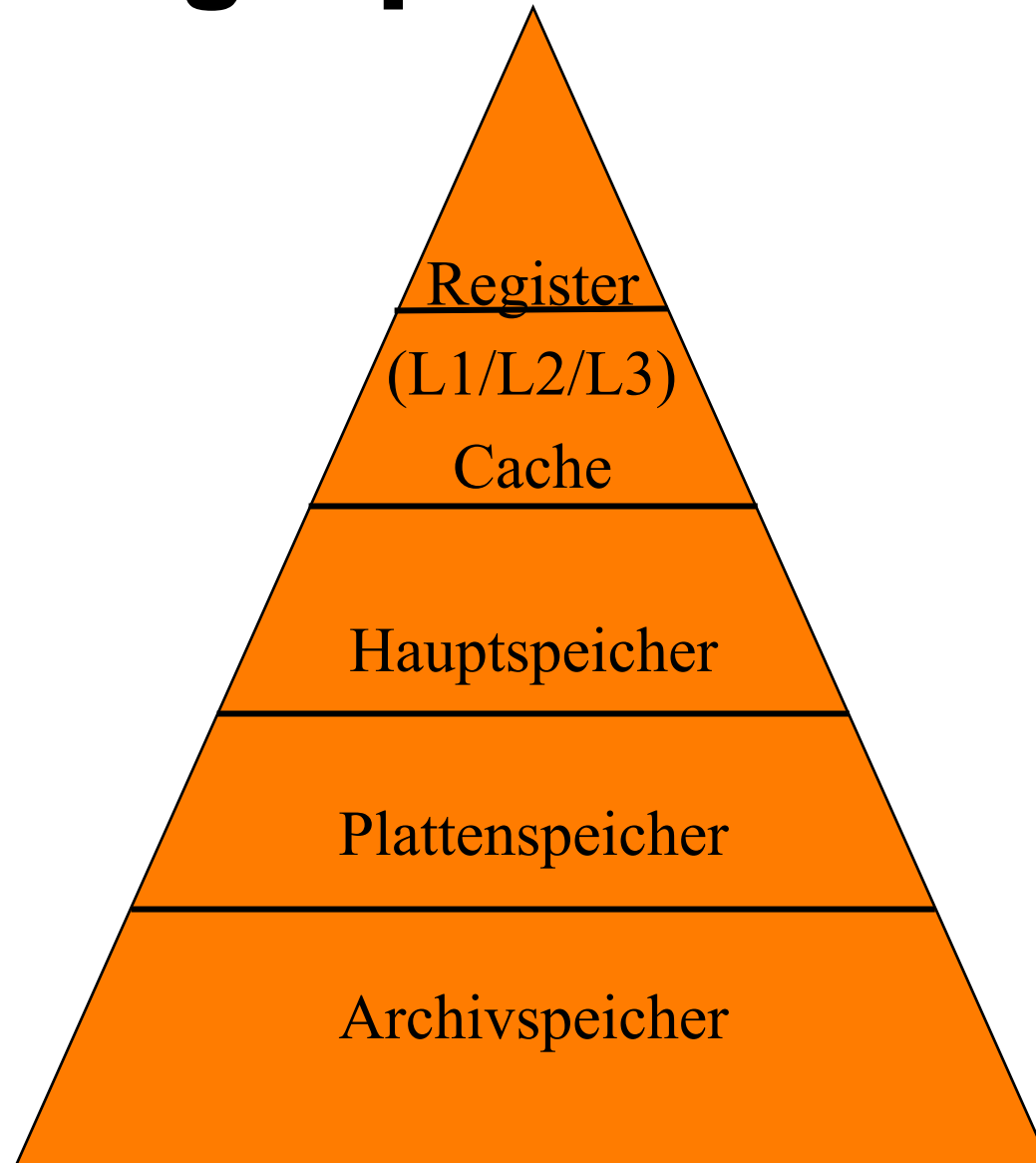
● Intel

- Tera Scale Initiative
- Server with several TB main memory
- We just ordered one from Dell for 49 K Euro
- Main Memory capacity will grow faster than Customers' Needs
- Cf. RAMcloud-project at Stanford
 - Ousterhoud et al.

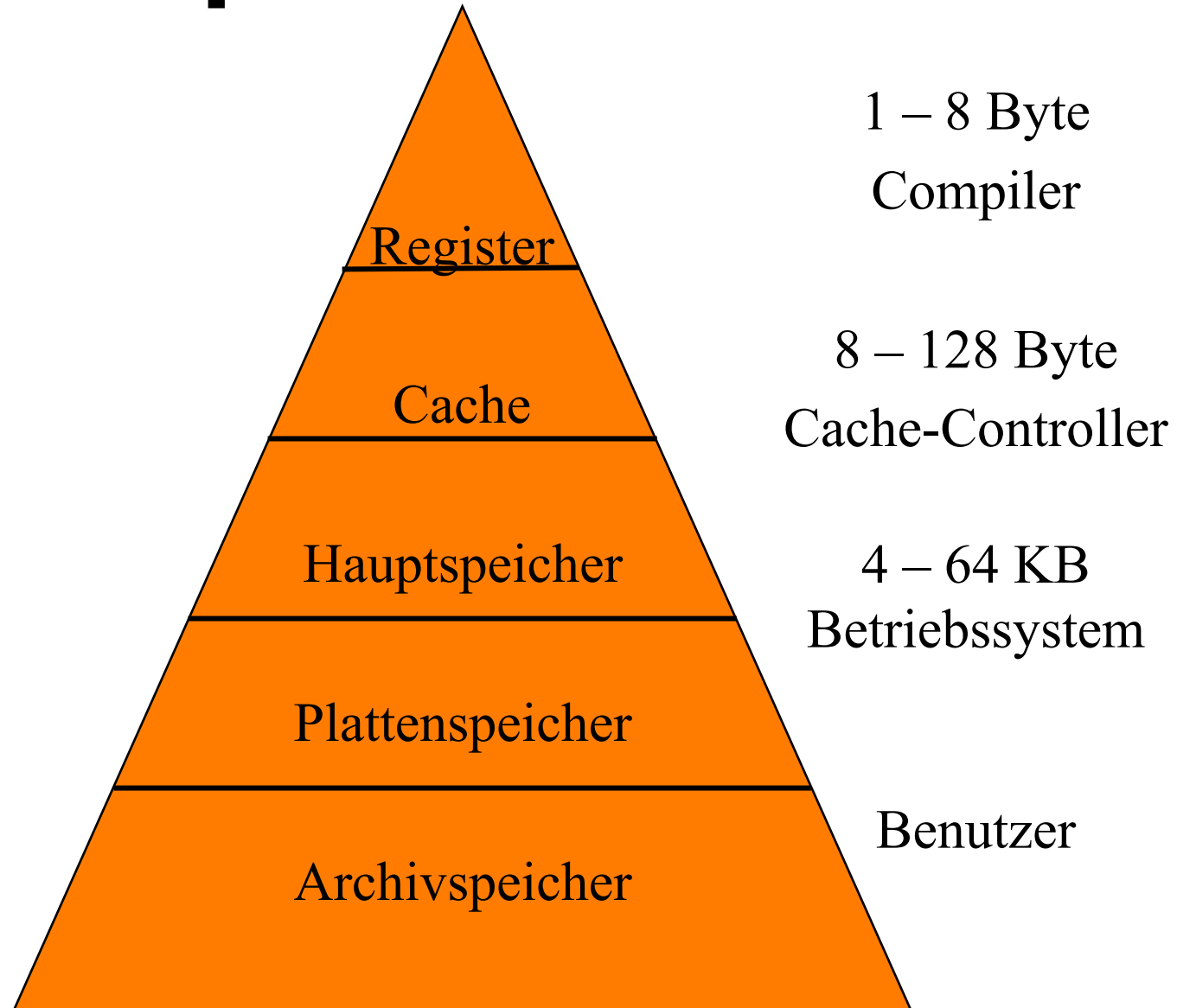
Leistungsengpässe: Profiling eines klassischen Datenbanksystems



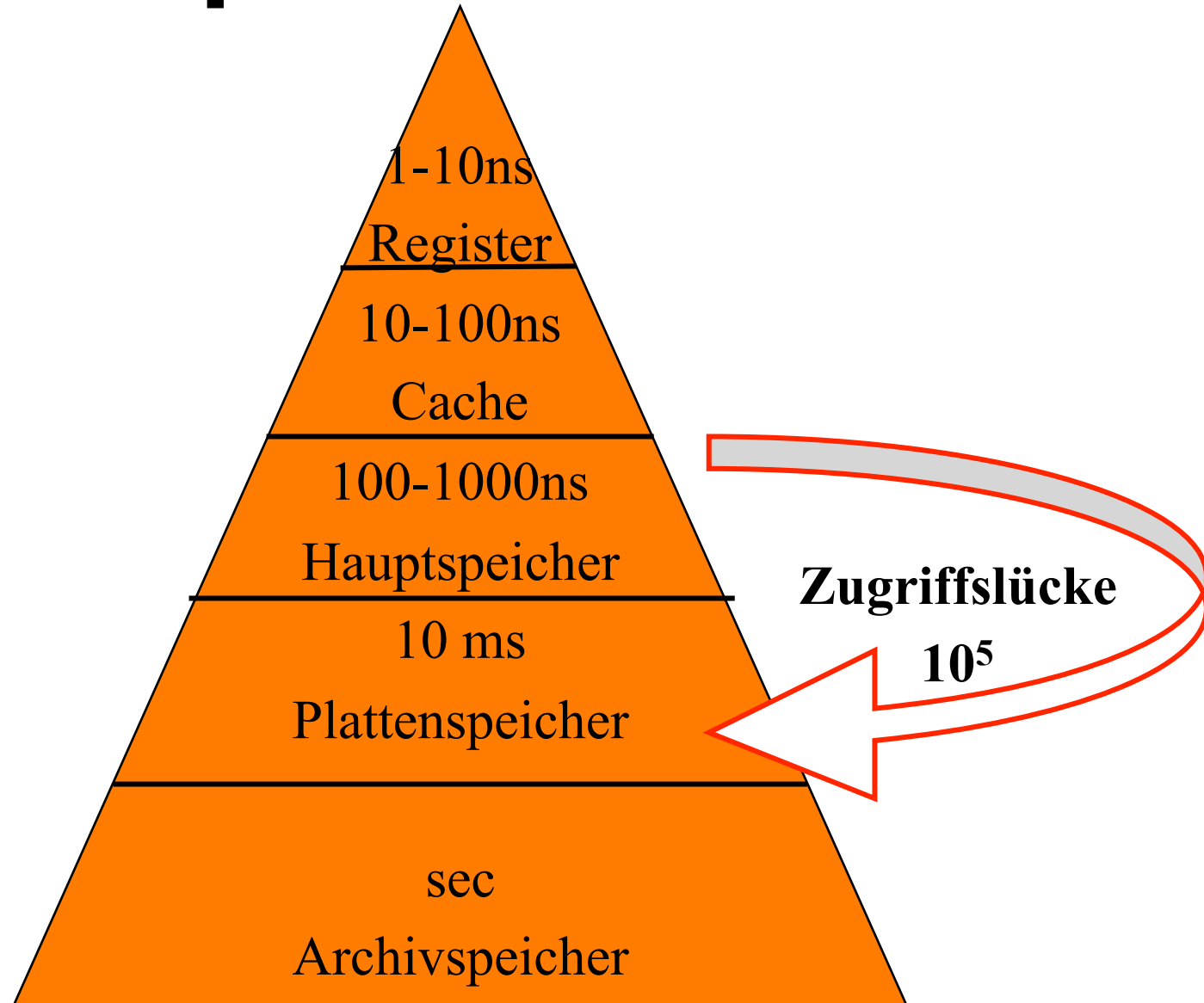
Widerholung: Speicherhierarchie



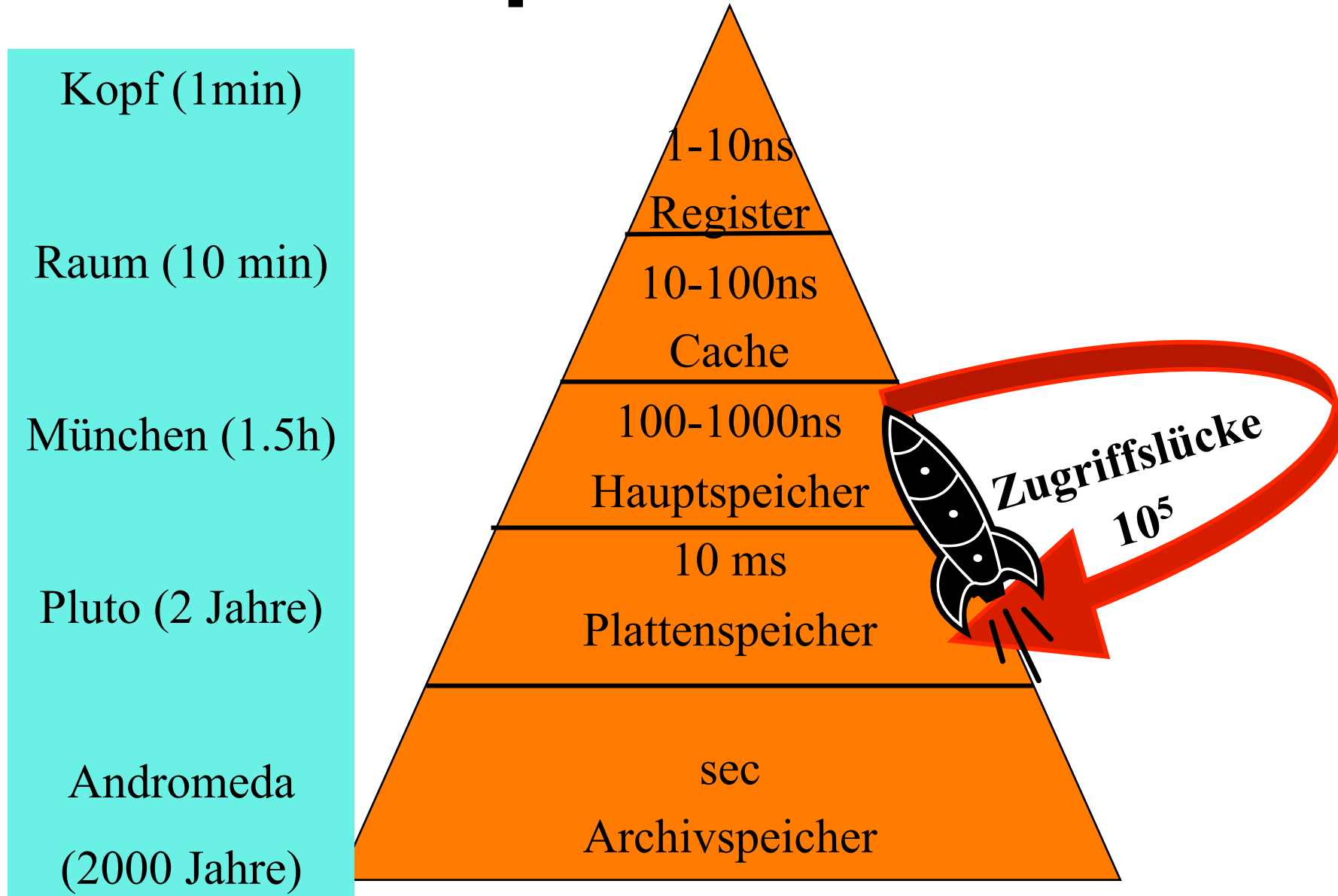
Überblick: Speicherhierarchie



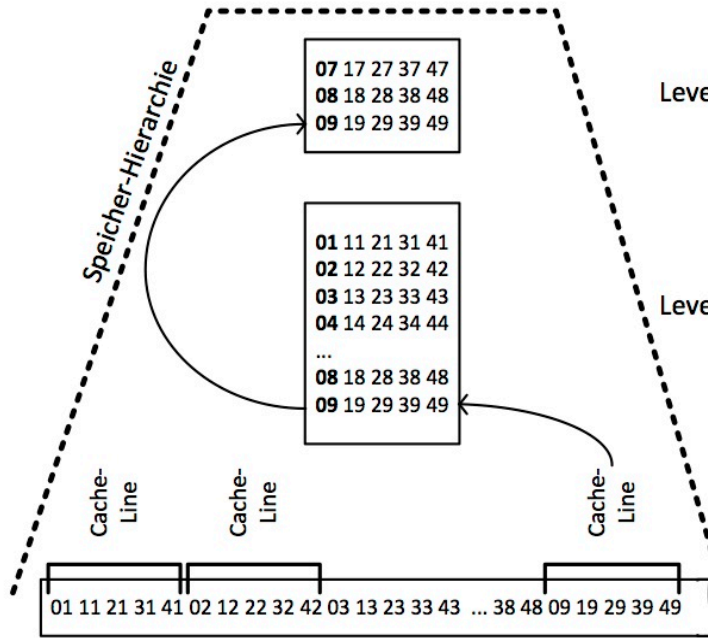
Überblick: Speicherhierarchie



Überblick: Speicherhierarchie



**select sum(A)
from R**



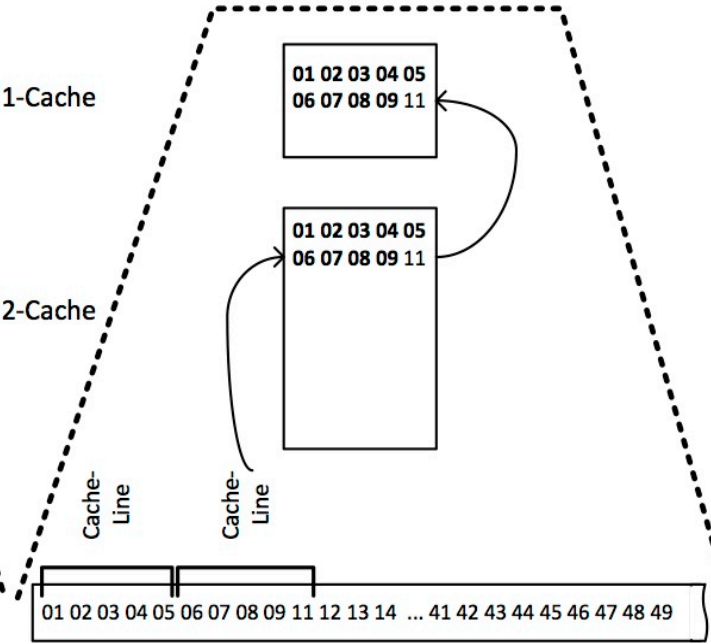
Hauptspeicher („ein-dimensional“)

R

A	B	C	D	E
01	11	21	31	41
02	12	22	32	42
03	13	23	33	43
04	14	24	34	44
05	15	25	35	45
06	16	26	36	46
07	17	27	37	47
08	18	28	38	48
09	19	29	39	49

Logischer Row-Store

**select sum(A)
from R**



Hauptspeicher („ein-dimensional“)

R

A	B	C	D	E
01	11	21	31	41
02	12	22	32	42
03	13	23	33	43
04	14	24	34	44
05	15	25	35	45
06	16	26	36	46
07	17	27	37	47
08	18	28	38	48
09	19	29	39	49

Logischer Column-Store

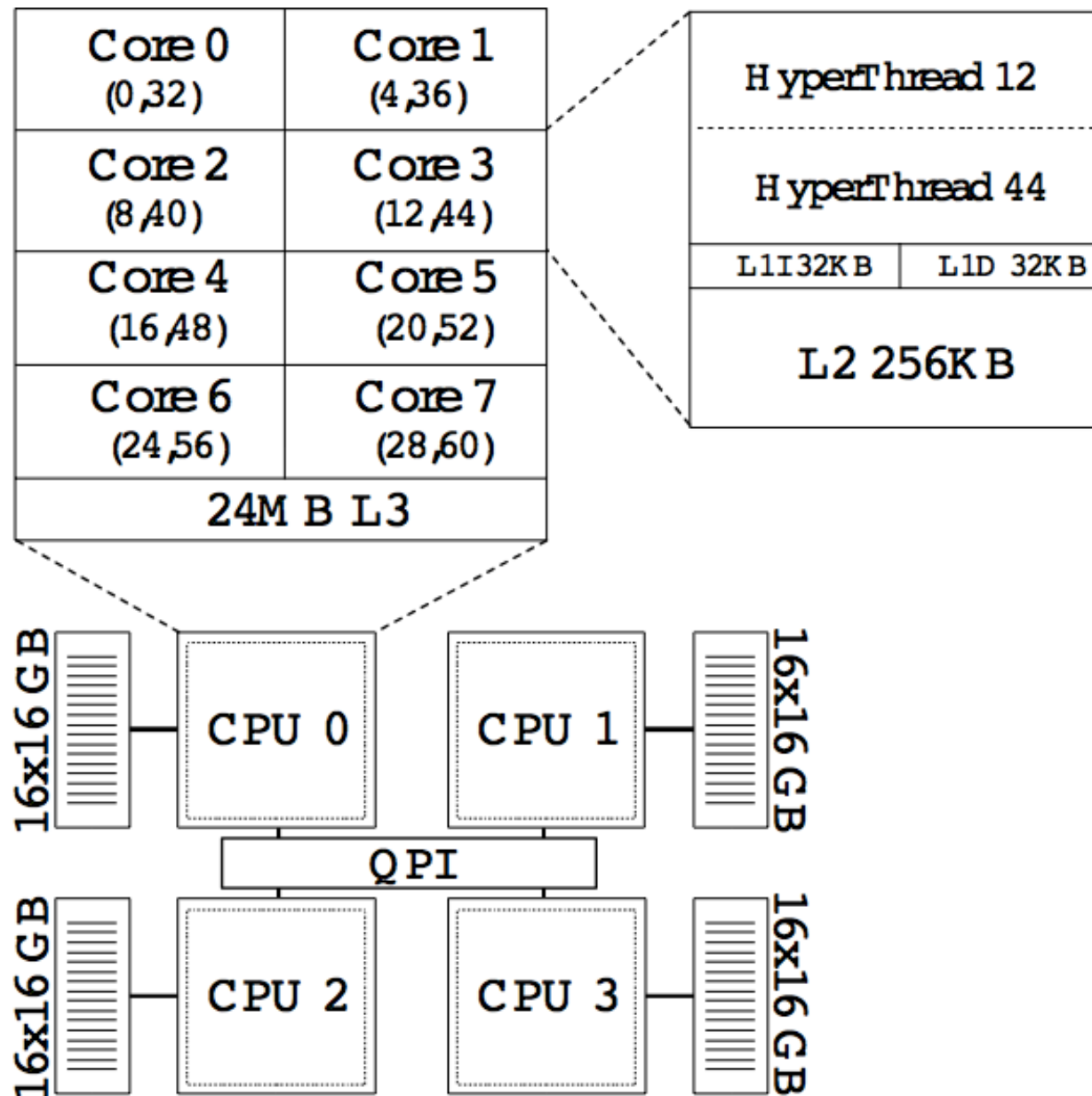


Abbildung 18.1: Architektur eines Mehrkern-Rechners mit NUMA-Hauptspeicher

Row Store versus Column Store

Verkäufe				
Produkt	Kunde	Preis	Filiale	...
Handy	Kemper	345	Schwabing	...
Radio	Mickey	123	Bogenhausen	...
Handy	Minnie	233	Schwabing	...
Kühlschrank	Urmel	240	Augsburg	...
Beamer	Bond	740	London	...
Handy	Lucie	321	Bogenhausen	...

Row Store versus Column Store

Produkt	
ID	Produkt
0	Handy
1	Radio
2	Handy
3	Kühlschrank
4	Beamer
5	Handy

Kunde	
ID	Kunde
0	Kemper
1	Mickey
2	Minnie
3	Urmel
4	Bond
5	Lucie

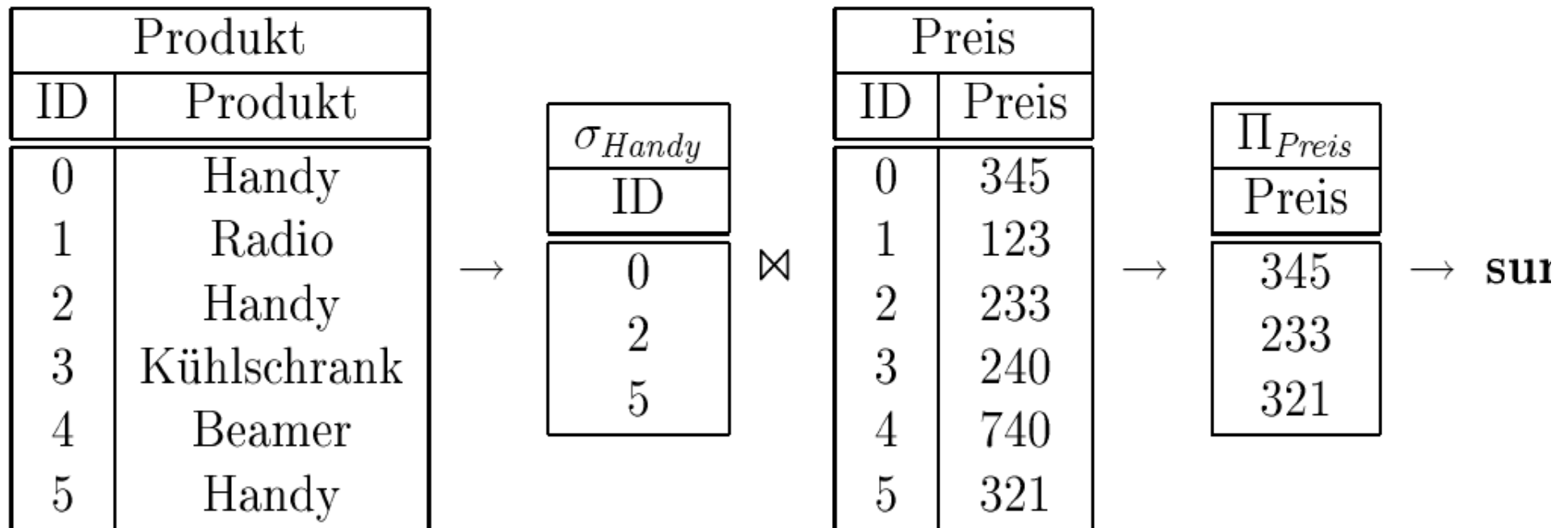
Preis	
ID	Preis
0	345
1	123
2	233
3	240
4	740
5	321

Filiale	
ID	Filiale
0	Schwabing
1	Bogenhausen
2	Schwabing
3	Augsburg
4	London
5	Bogenhausen

Anfragebearbeitung

```
select sum(Preis)
from Verkäufe
where Produkt = 'Handy'
```

Die schrittweise Abarbeitung dieser Anfrage ist nachfolgend illustriert



Komprimierung

Dictionary	
ID	Wort
0	Augsburg
1	Beamer
2	Bogenhausen
3	Handy
4	Kühlschrank
5	London
6	Radio
7	Schwabing
...	...

Produkt	
ID	Produkt
0	3
1	6
2	3
3	4
4	1
5	3

Filiale	
ID	Filiale
0	7
1	2
2	7
3	0
4	5
5	2

Datenstrukturen einer Hauptspeicher-Datenbank

- Kunden: {[id: int, name: char(30), rabatt: double, land: int]}
- Laender: {[id: int, name: char(30), steuern: double]}
- Produkte: {[id: int, name: char(30), preis: double]}
- Verkaeufe: {[id: int, kunde: int, produkt: int, datum: int, preis: double]}

```
create table Kunden
```

```
( id int, name char(30), rabatt double, land int )
```

Row-Store-Format

```
/// Ein Kunde
struct Kunde { unsigned id; char name[30]; double rabatt; unsigned land; };
/// Ein Land
struct Land { unsigned id; char name[30]; double steuern; };
/// Ein Produkt
struct Produkt { unsigned id; char name[30]; double preis; };
/// Ein Verkauf
struct Verkauf { unsigned id; unsigned kunde; unsigned produkt;
                unsigned datum; double preis; };

/// Die Datenbank im Row-Format
vector<Kunde> Kunden;
vector<Land> Laender;
vector<Produkt> Produkte;
vector<Verkauf> Verkaeufe;
```

Column-Store-Format

```
/// Template für Strings fester Länge -- ohne Indirektion
template <unsigned len> struct Char { char data[len]; };

/// Ein Kunde
struct Kunde { unsigned id; Char<30> name; double rabatt; unsigned land; };
///Alle Kunden in Column-Format
struct Kunden {
    vector<unsigned> data_id; vector<Char<30>> data_name;
    vector<double> data_rabatt; vector<unsigned> data_land;

    void insert(Kunde&& kunde);
};

/// Ein Land
struct Land { unsigned id; char name[30]; double steuern; };
/// Alle Länder in Column-Format
struct Laender {
    vector<unsigned> data_id; vector<Char<30>> data_name;
    vector<double> data_steuern;

    void insert(Land&& land);
};
```

Column-Store-Format (cont'd)

```
/// Ein Produkt
struct Produkt { unsigned id; char name[30]; double preis; };
/// Alle Produkte in Column-Format
struct Produkte {
    vector<unsigned> data_id; vector<Char<30>> data_name;
    vector<double> data_preis;

    void insert(Produkt&& produkt);
};

/// Ein Verkauf
struct Verkauf { unsigned id; unsigned kunde; unsigned produkt;
                unsigned datum; double preis; };
/// Alle Verkäufe in Column-Format
struct Verkaeufe {
    vector<unsigned> data_id; vector<unsigned> data_kunde;
    vector<unsigned> data_produk; vector<unsigned> data_datum;
    vector<double> data_preis;

    void insert(Verkauf&& verkauf);
};
```

Einfügeoperation eines Tupels

Insert into Verkaeufe values (12, 007, 4711, 27.50)



```
void Verkaeufe::insert(Verkauf&& verkauf)
{
    data_id.push_back(verkauf.id);
    data_kunde.push_back(verkauf.kunde);
    data_produkt.push_back(verkauf.produkt);
    data_datum.push_back(verkauf.datum);
    data_preis.push_back(verkauf.preis);
}
```

Anfragen

```
select sum(v.preis) from Verkaeufe v
where v.datum >= 20130101
```



```
double umsatz(Verkaeufe& v)
{
    double summe = 0.0;
    for (unsigned i = 0; i < v.data_datum.size(); i++) {
        if (v.data_datum[i] >= 20130101) {
            summe += v.data_preis[i];
        }
    }
    return summe;
}
```

Hybrides Speichermodell

```
select datum, sum(preis)
from Verkaeufe
where datum >= 20130101
group by datum
```

```
/// Ein Verkauf
struct Verkauf { unsigned id; unsigned kunde; unsigned produkt;
                 unsigned datum; double preis; };
struct VerkaufsDatumPreis { unsigned datum; double preis; };
/// Alle Verkäufe in hybridem Format
struct Verkaeufe {
    vector<unsigned> data_id; vector<unsigned> data_kunde;
    vector<unsigned> data_produkt;
    vector<VerkaufsDatumPreis> data_datum_preis;

    void insert(Verkauf&& verkauf);
};
```



Anfragebearbeitung

```
unordered_map<unsigned, double> umsatzProDatum(Verkaeufe& verkaeufe)
{
    unordered_map<unsigned, double> groupBy;
    for (VerkaufsDatumPreis datum_preis : verkaeufe.data_datum_preis) {
        if (datum_preis.datum >= 20130101) {
            groupBy[datum_preis.datum] += datum_preis.preis;
        }
    }
    return groupBy;
}
```


Anwendungsoperationen in der Datenbank: Stored Procedures

```
create procedure newOrder (w_id integer not null, d_id integer not null,
    c_id integer not null, table positions(line_number integer not null,
    supware integer not null, itemid integer not null, qty integer not null),
    datetime timestamp not null) // note the TABLE-valued parameter above
{
    select w_tax from warehouse w where w.w_id=w_id; // w_tax value used later
    select c_discount from customer c // c_discount used in orderline insert
        where c_w_id=w_id and c_d_id=d_id and c.c_id=c_id;
```

```

select d_next_o_id as o_id,d_tax from district d // get the next o_id
      where d_w_id=w_id and d.d_id=d_id;
update district set d_next_o_id=o_id+1 // increment the next o_id
      where d_w_id=w_id and district.d_id=d_id;

select count(*) as cnt from positions; // how many items are ordered
select case when count(*)=0 then 1 else 0 end as all_local
      from positions where supware<>w_id;

insert into "order" values (o_id,d_id,w_id,c_id,datetime,0,cnt,all_local);
insert into neworder values (o_id,d_id,w_id); // insert reference to order

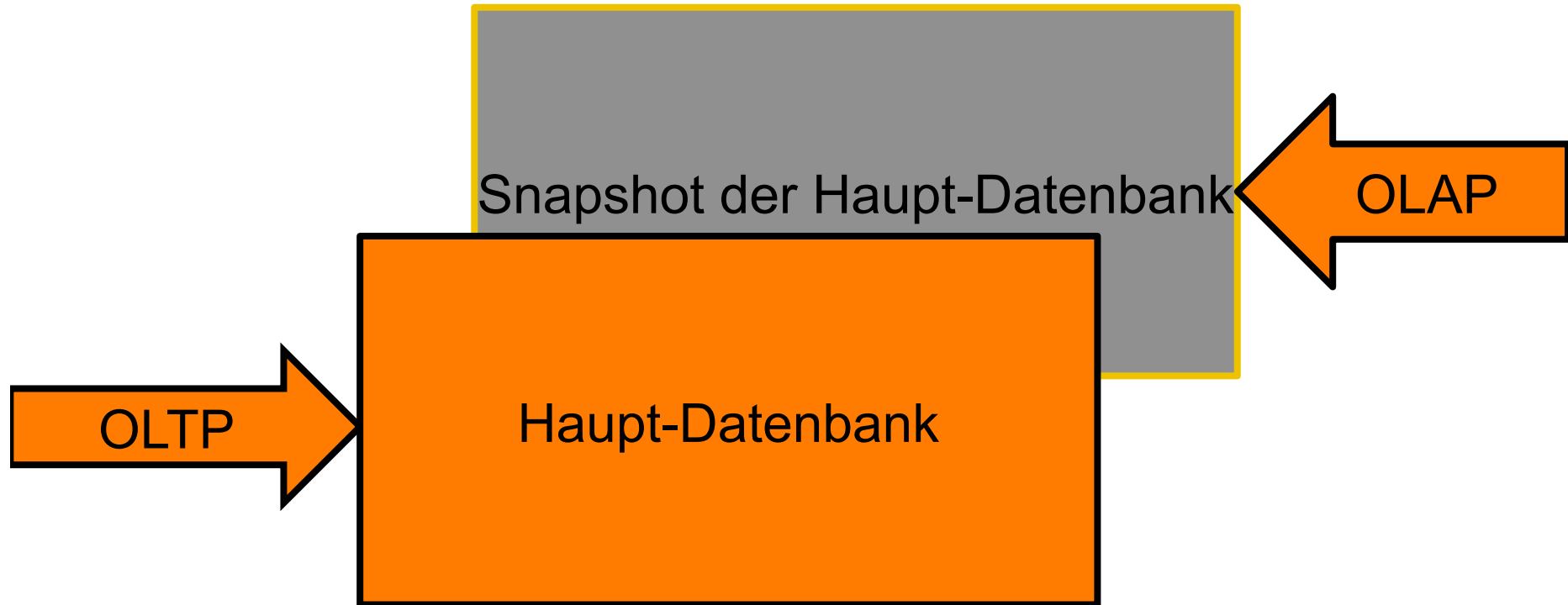
update stock
set s_quantity=case when s_quantity>qty then s_quantity-qty
                    else s_quantity+91-qty end,
    s_remote_cnt=s_remote_cnt+case when supware<>w_id then 1 else 0 end,
    s_order_cnt=s_order_cnt+case when supware=w_id then 1 else 0 end
from positions
where s_w_id=supware and s_i_id=itemid;

insert into orderline // insert all the order positions
      select o_id,d_id,w_id,line_number,itemid,supware,null,qty,
            qty*i_price*(1.0+w_tax+d_tax)*(1.0-c_discount),
            case d_id when 1 then s_dist_01 when 2 then s_dist_02
                    when 3 then s_dist_03 when 4 then s_dist_04
                    when 5 then s_dist_05 when 6 then s_dist_06
                    when 7 then s_dist_07 when 8 then s_dist_08
                    when 9 then s_dist_09 when 10 then s_dist_10 end
      from positions, item, stock
      where itemid=i_id and s_w_id=supware and s_i_id=itemid
returning count(*) as inserted; // how many were inserted?

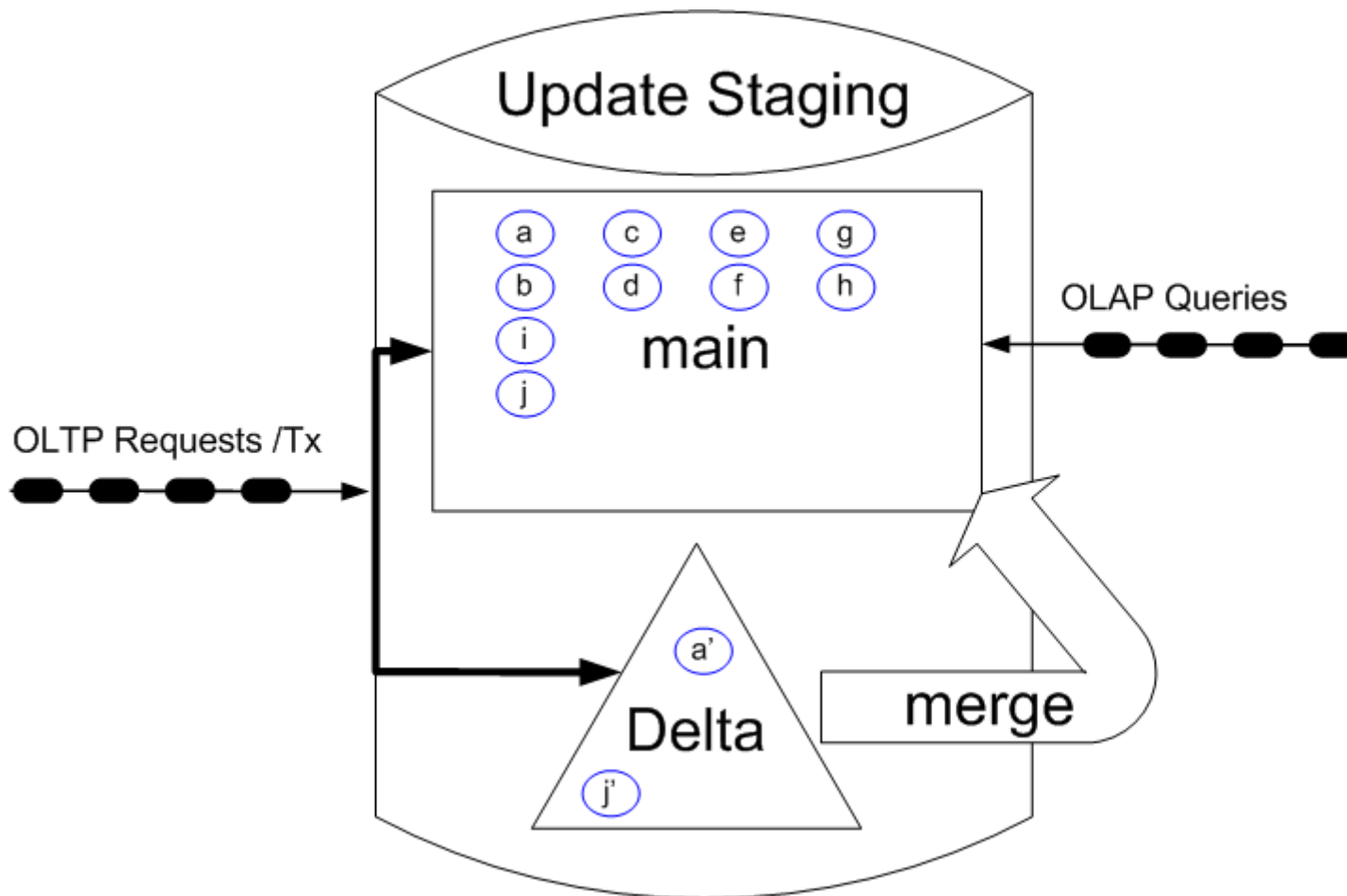
if (inserted<cnt) rollback; // not all ==> invalid item ==> abort
};

```

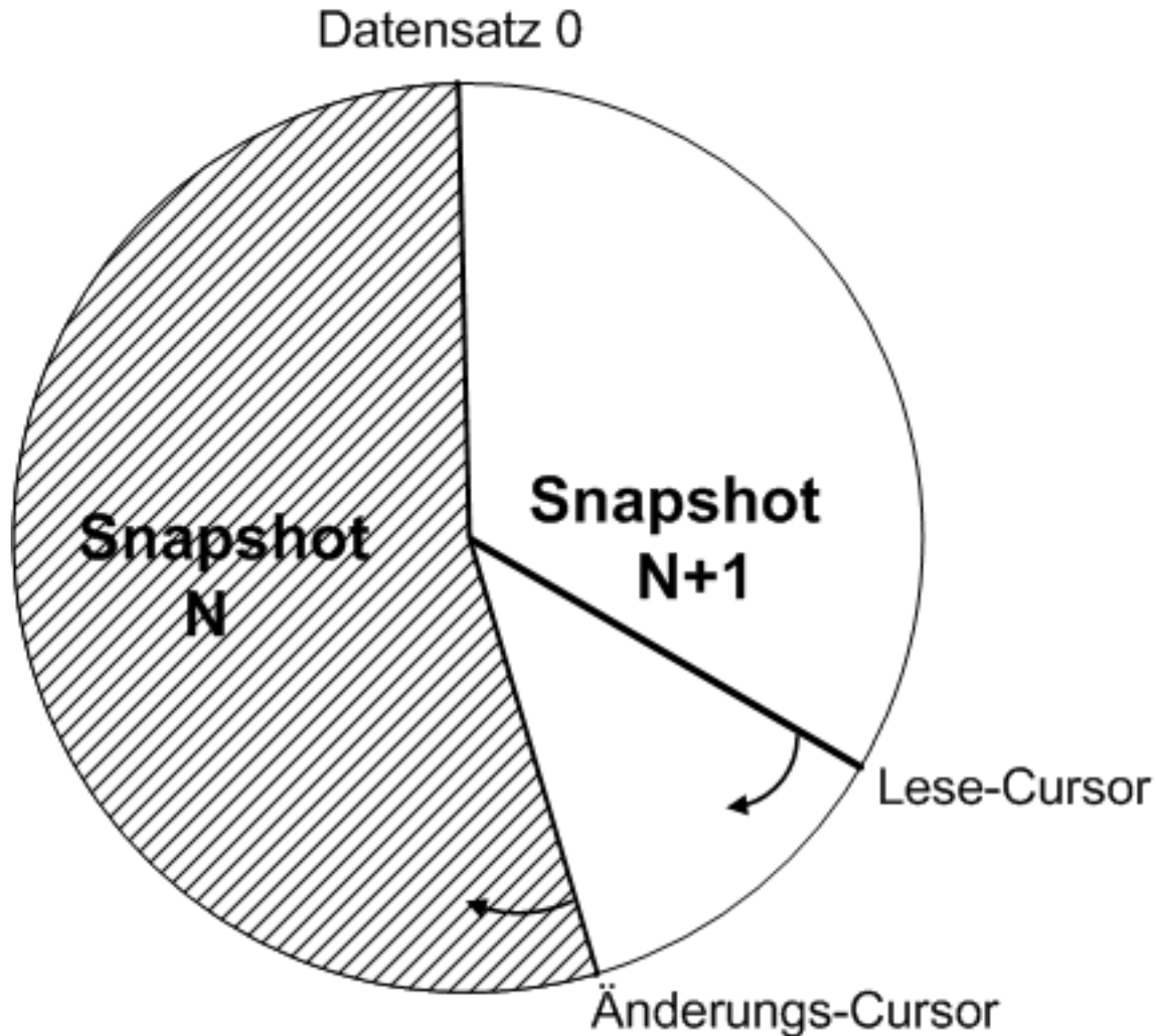
Snapshots für Anfragen



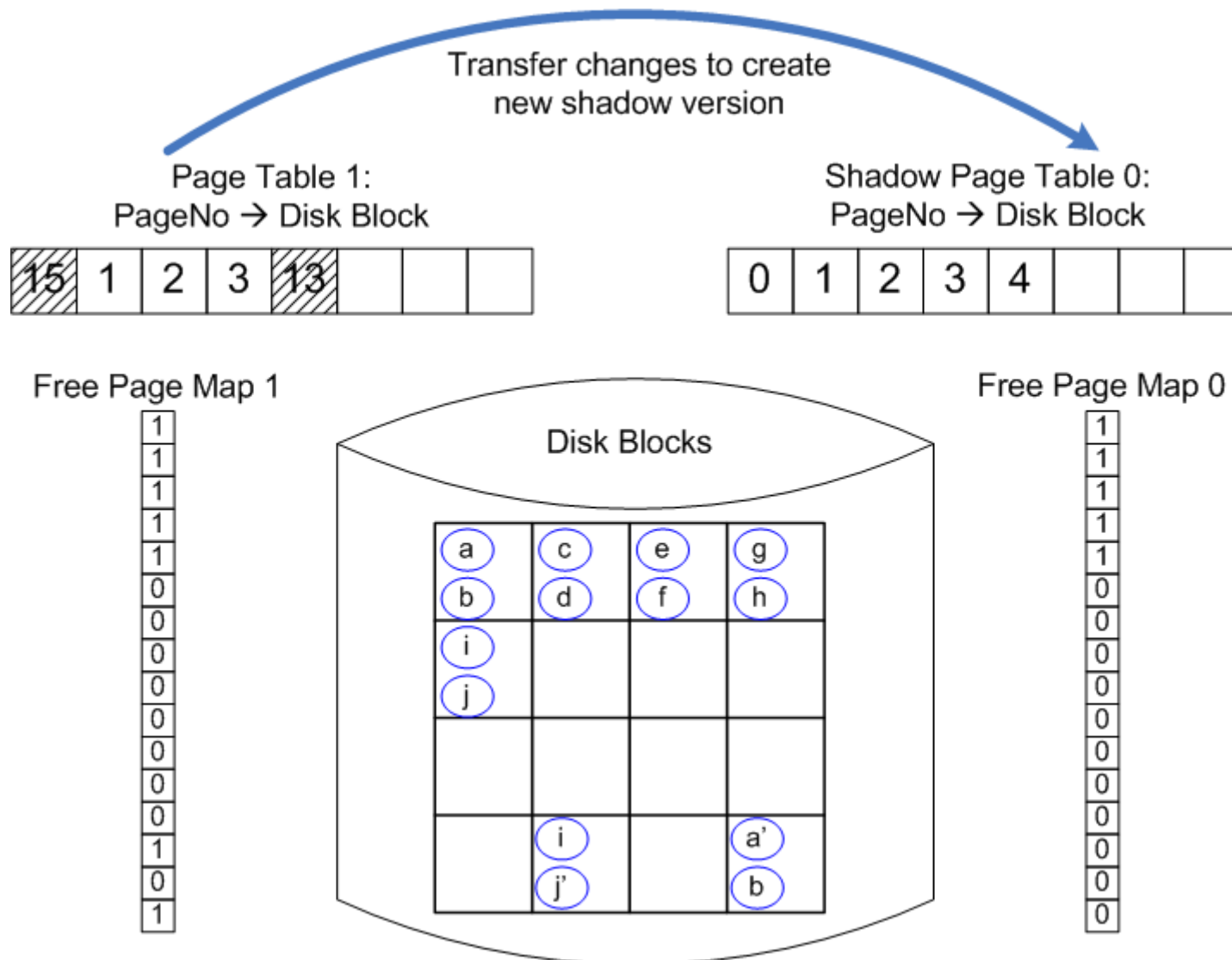
Update Staging: In vielen Systemen verwendet, zB. NewDB von SAP

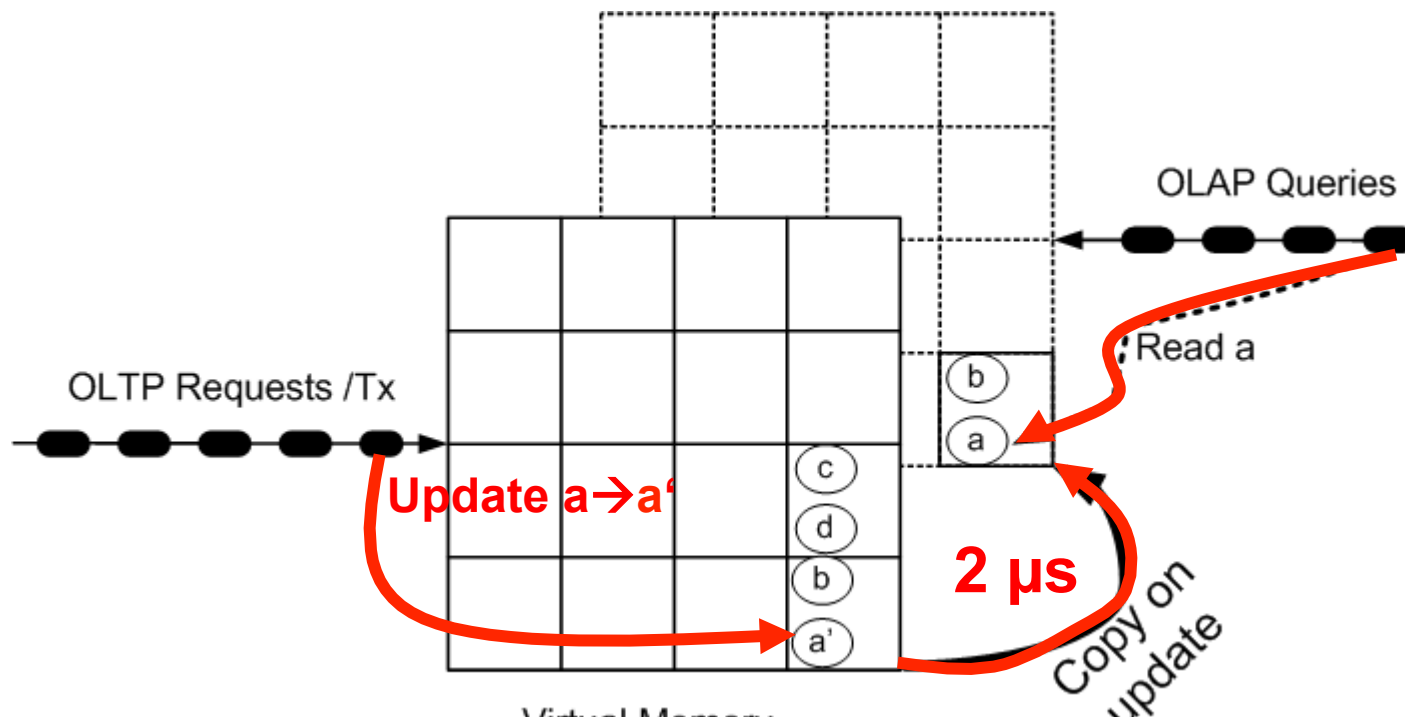
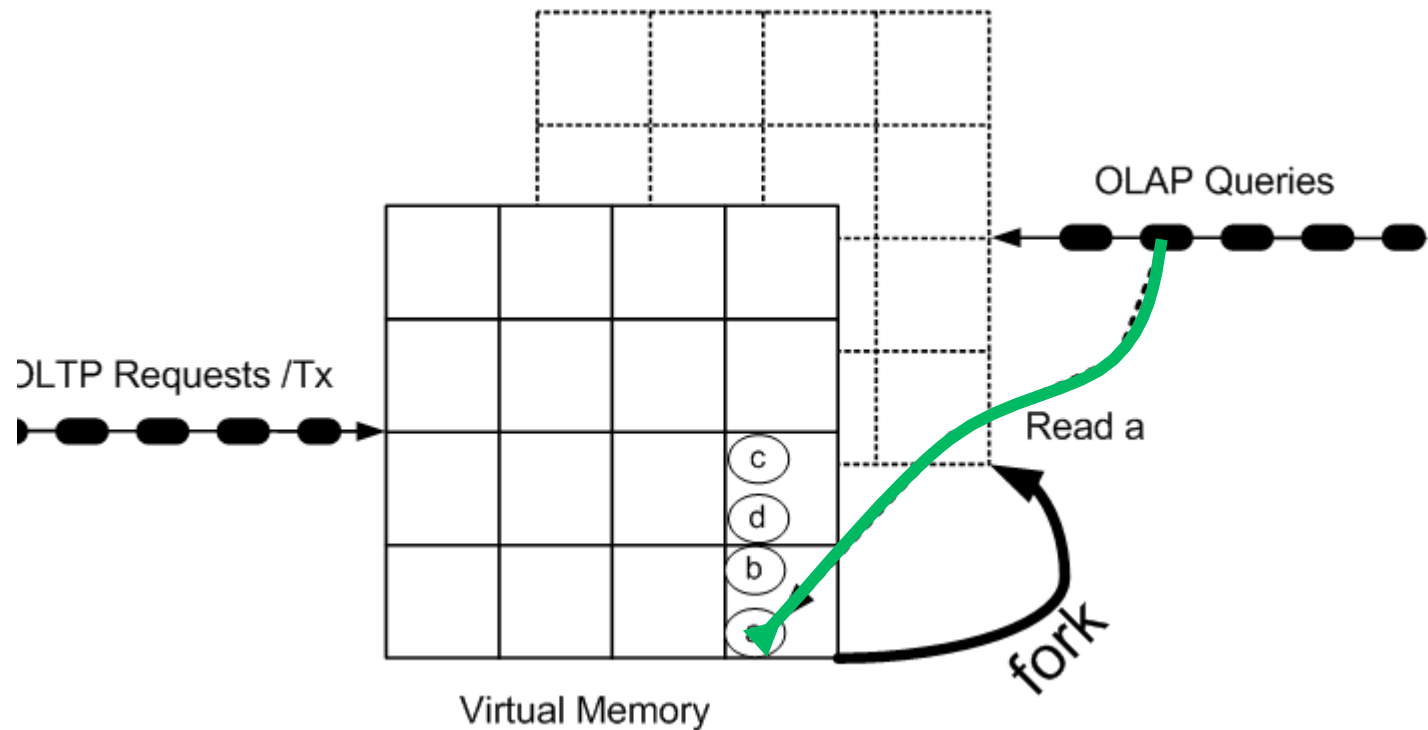


Scan-only Datenbanken: ISA0 von IBM oder Crescando von der ETHZ

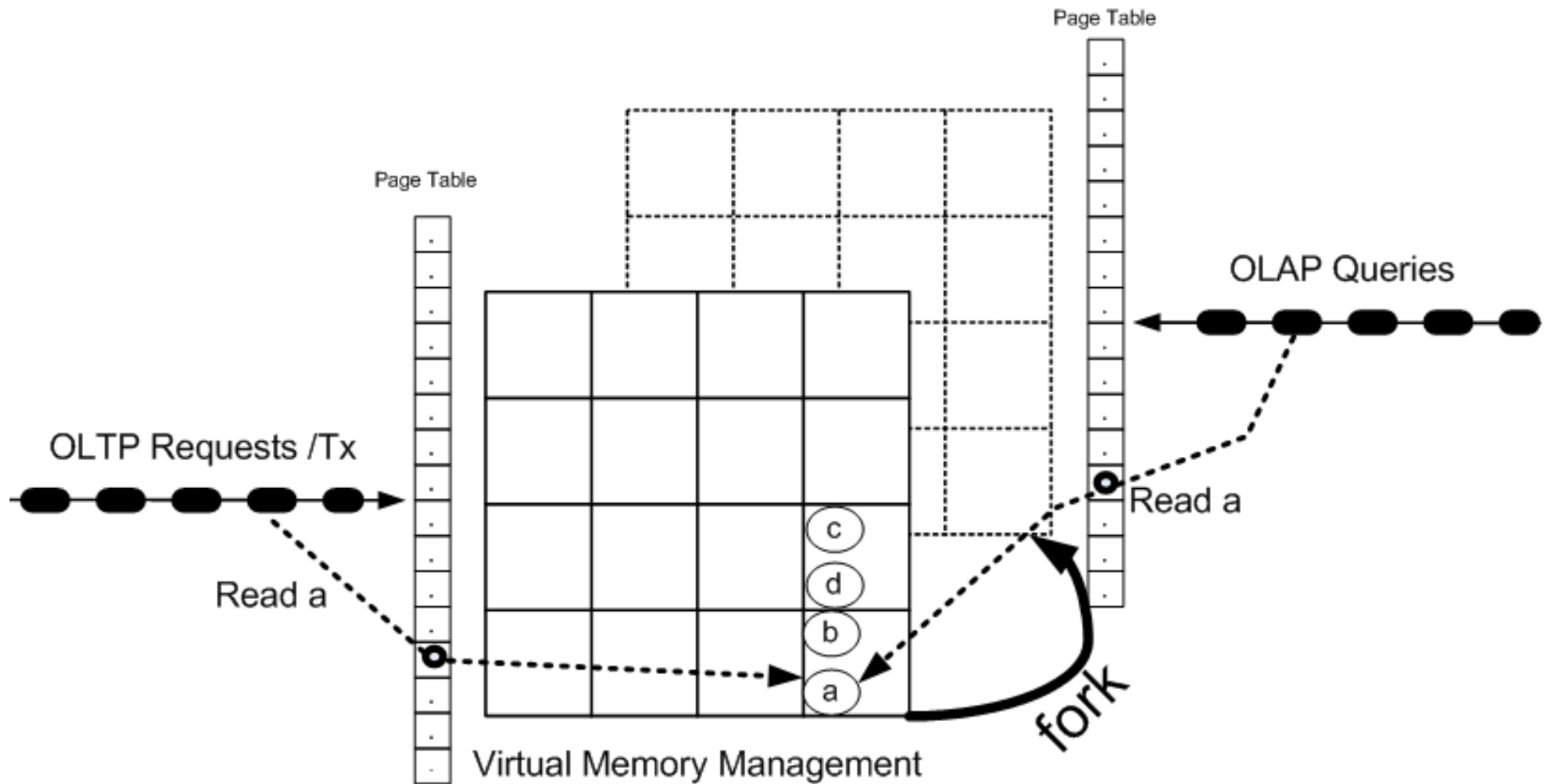


Ursprüngliches Schattenspeicher-Verfahren: Lorie77 für IBM System R

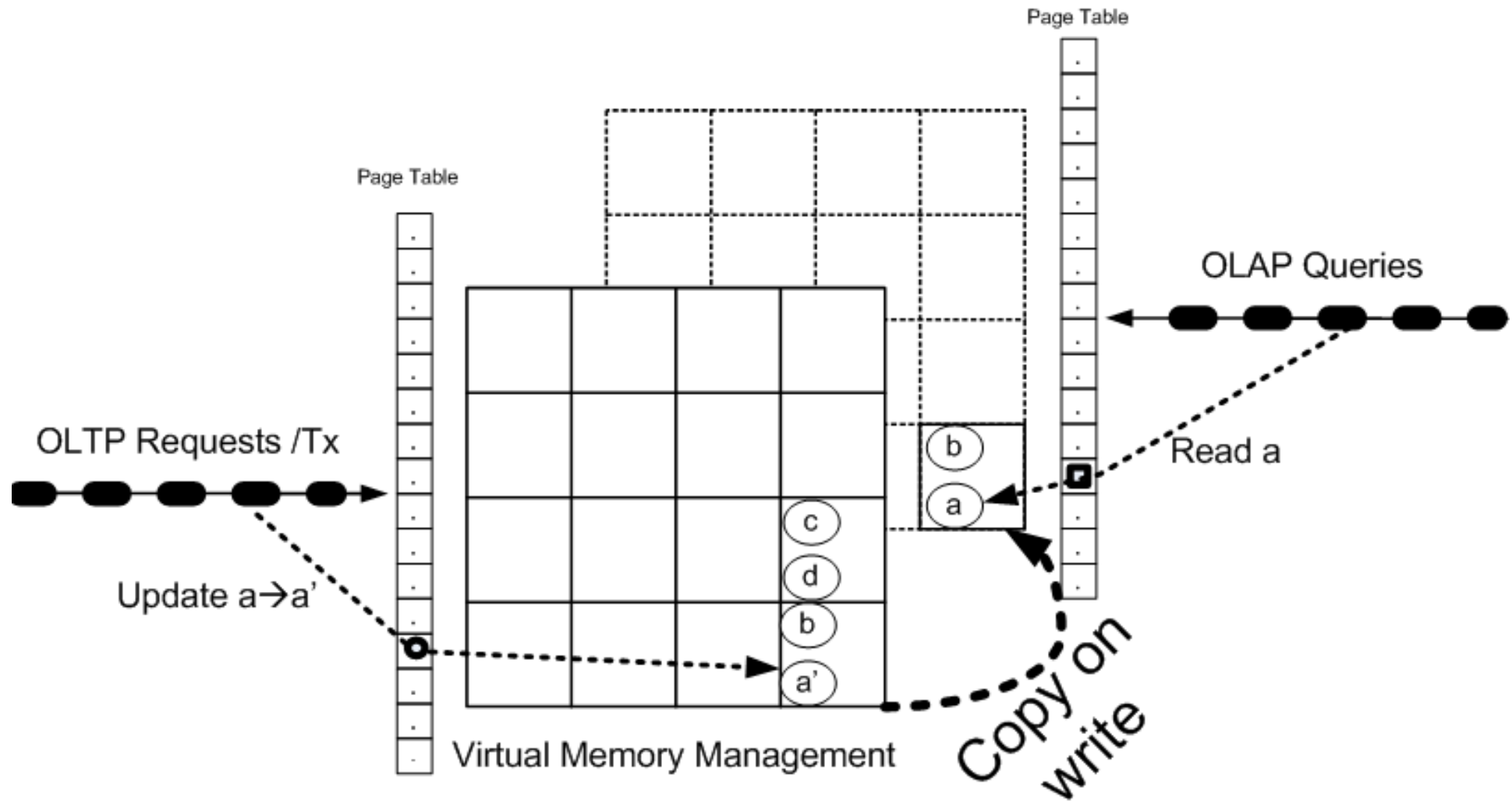




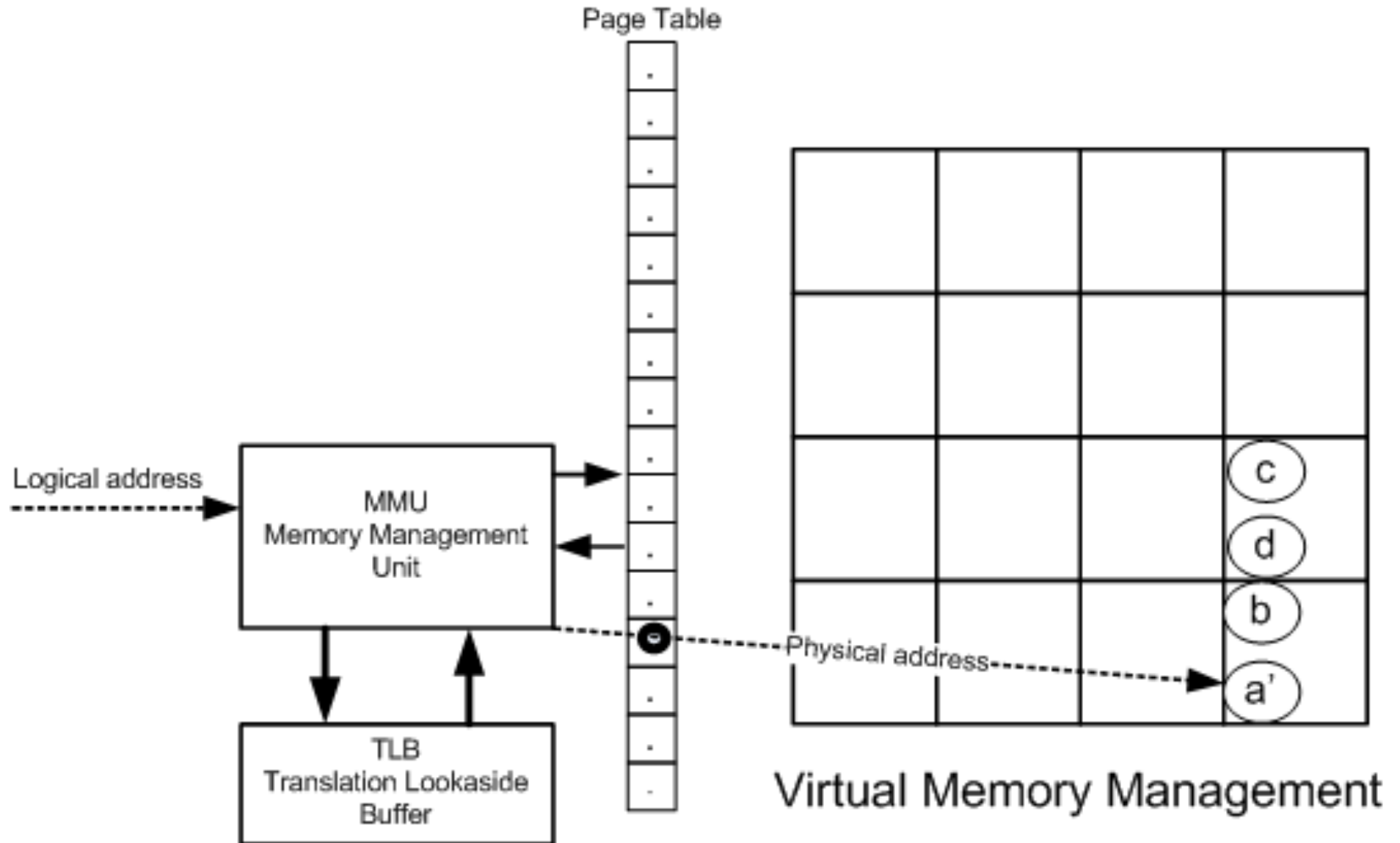
Snapshotting via fork-ing: Details



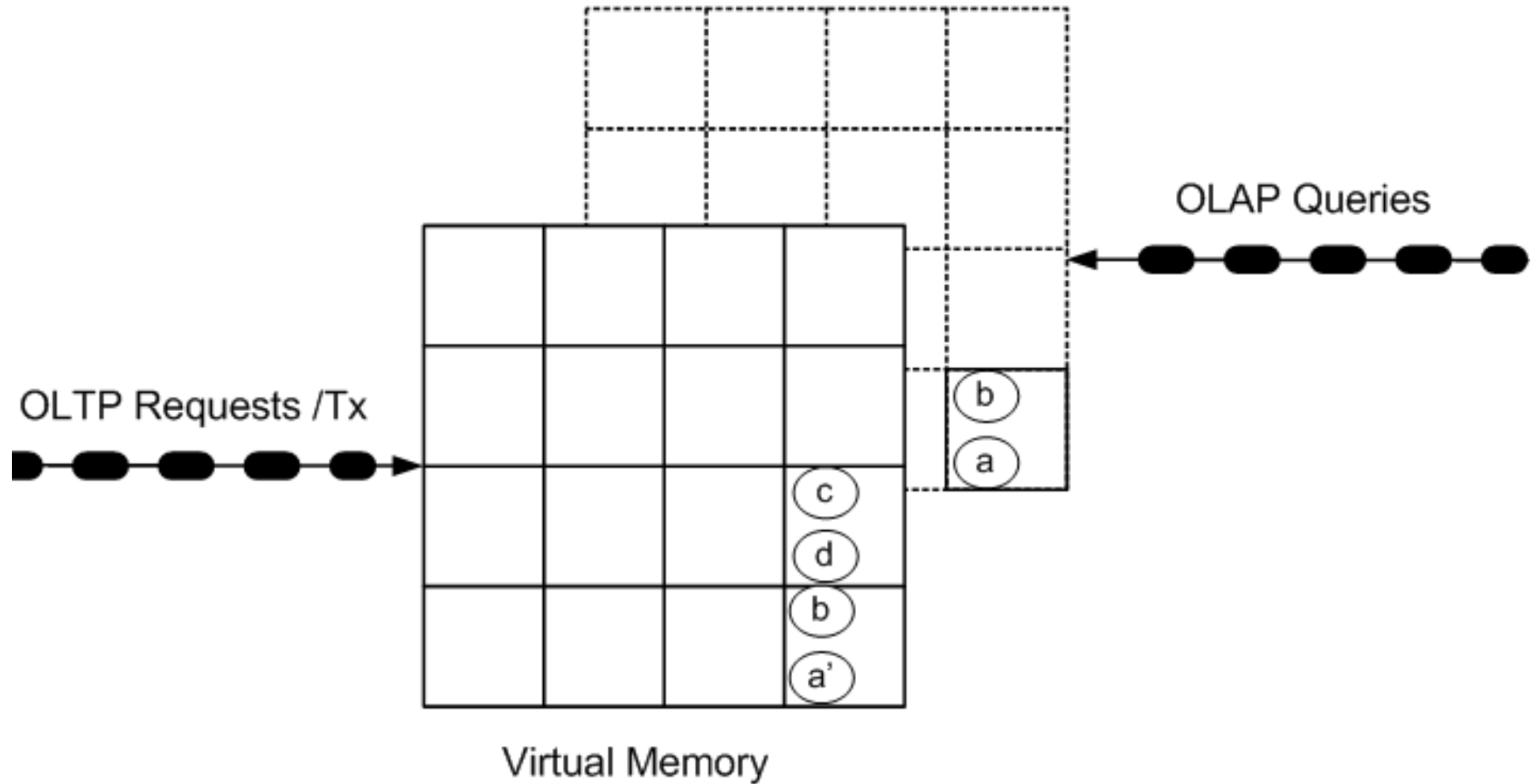
Snapshot Maintenance: copy on write



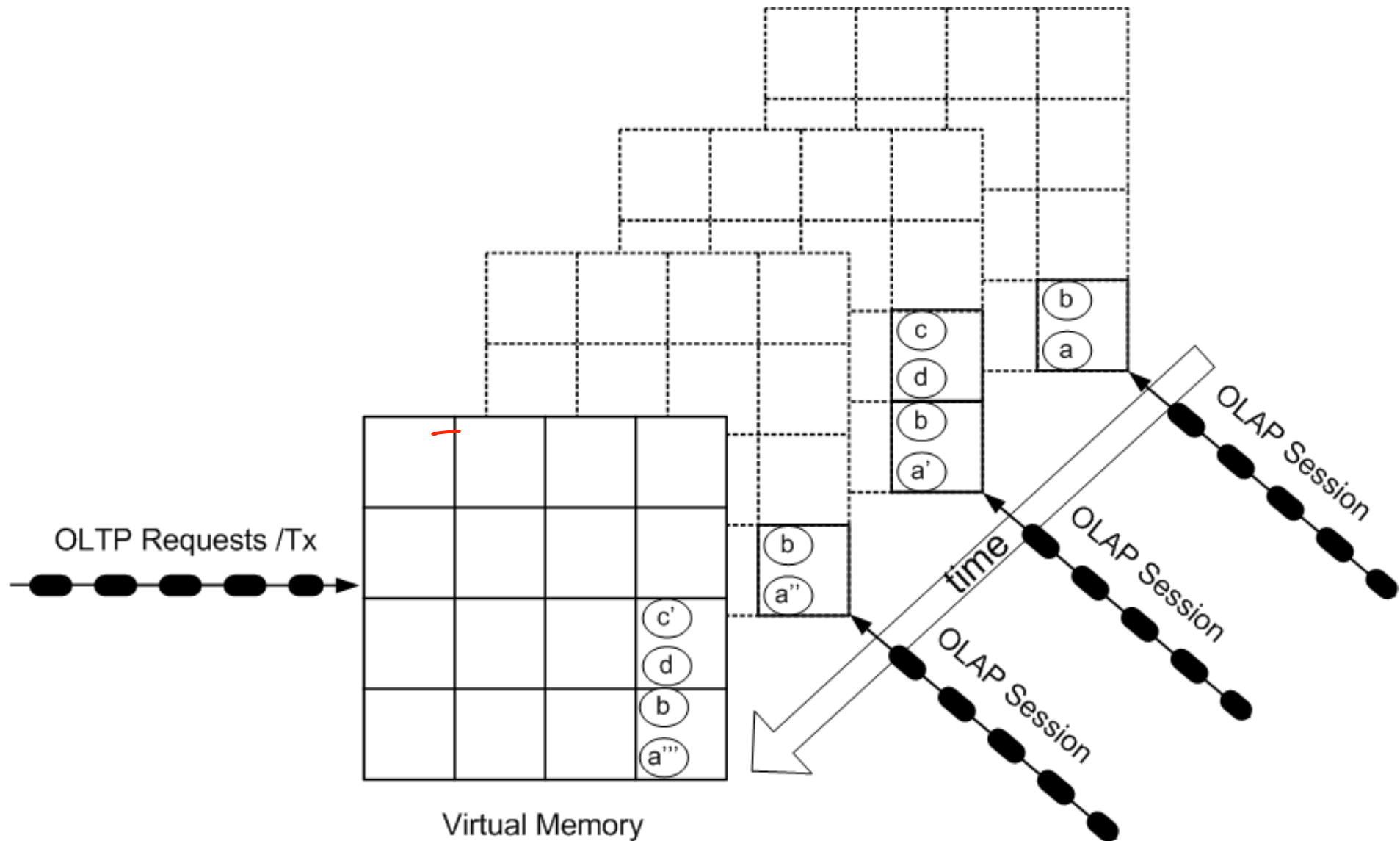
Fast because of Hardware-Support: MMU



OLAP Queries on Tx-Consistent Snapshots



Multiple Query Sessions



Synchronization-Assertions

- Serializability of the OLTP Transactions
 - What else if executed serially
 - We support full ACID → see coming slides
- Snapshot isolation of the OLAP queries
 - Multi-version mixed synchronization method
 - Several OLAP queries form one Tx = OLAP Session
 - Bernstein, Hadzilacos, Goodman: Chapter 5.5

Kompaktifizierung: Motivation

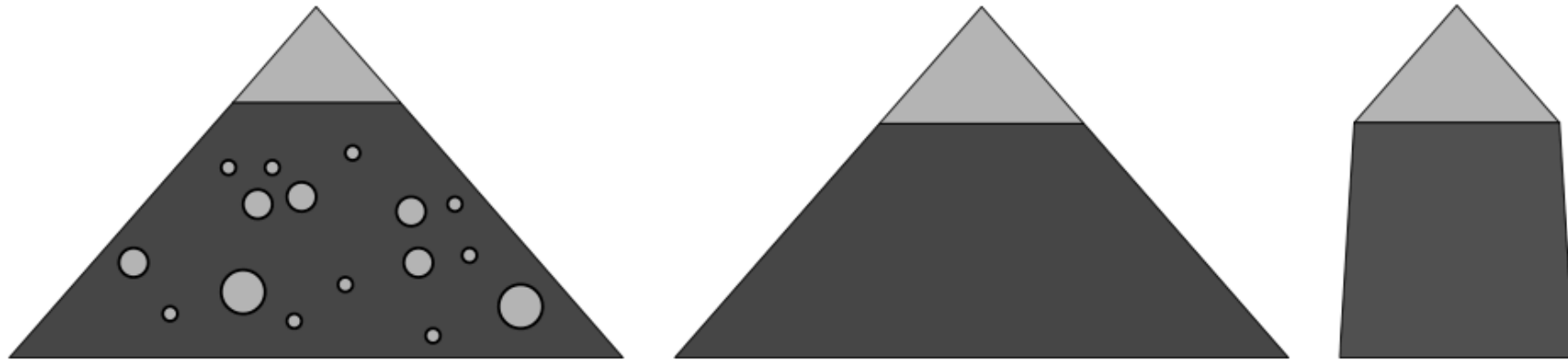
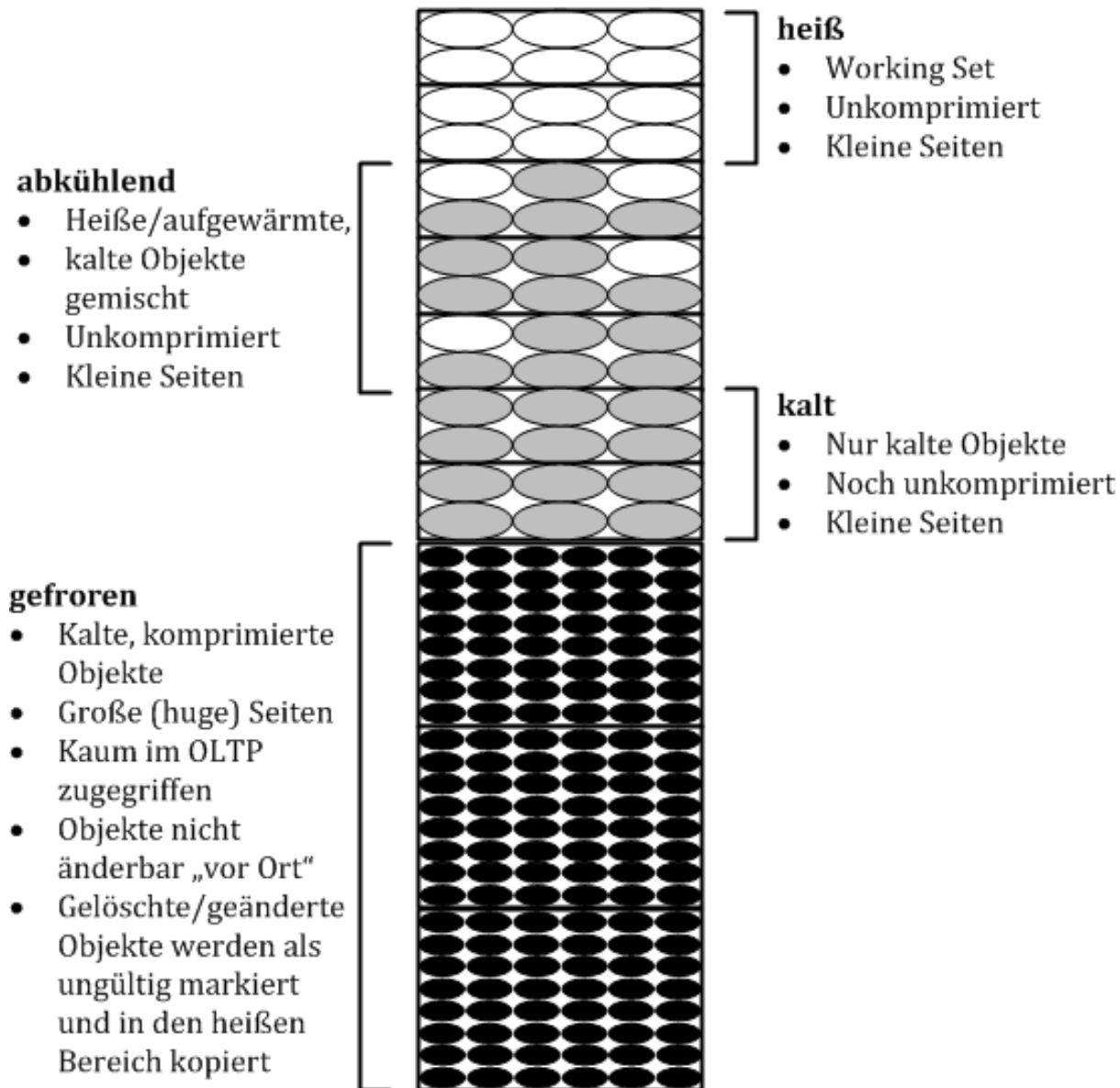
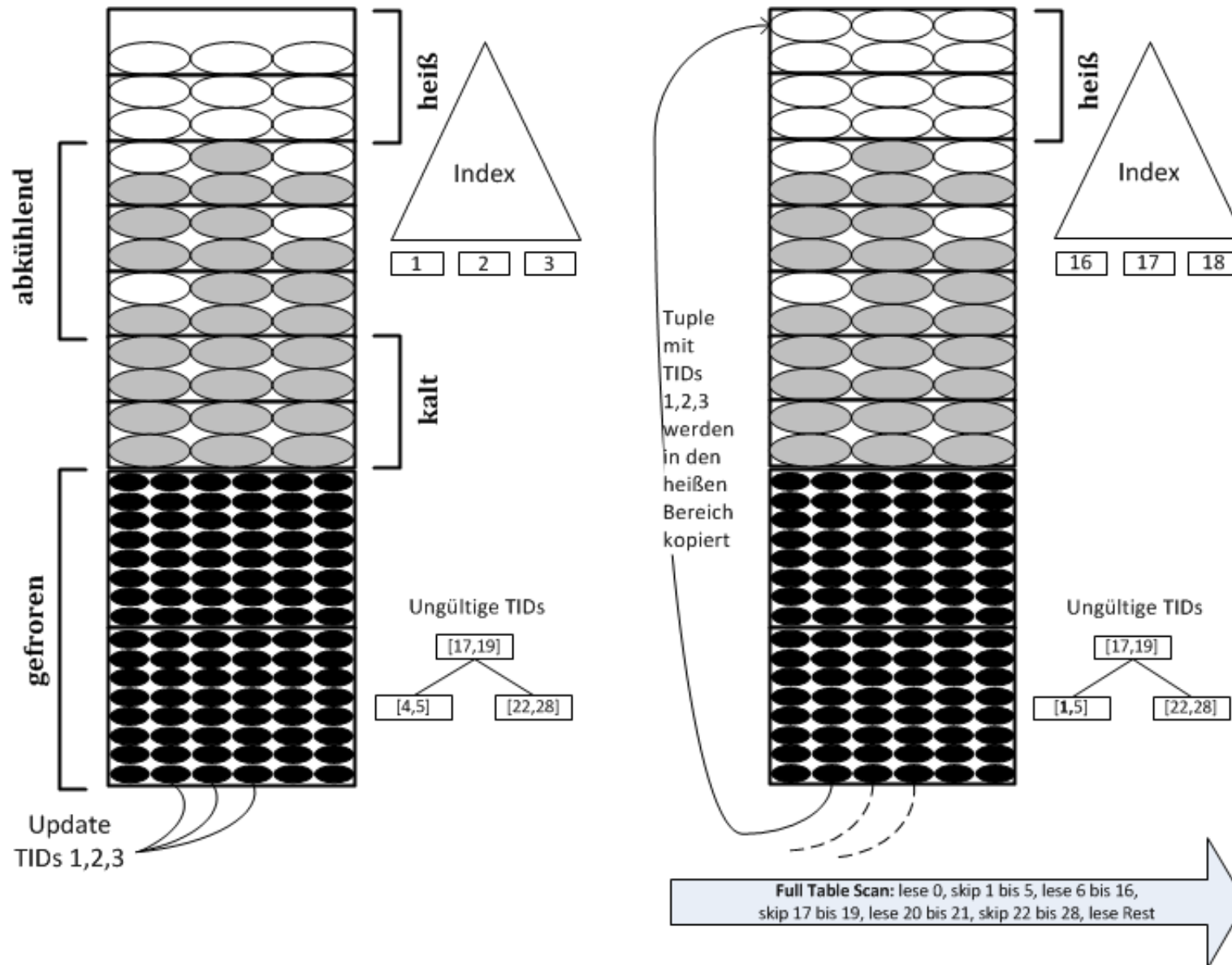


Abbildung 18.12: Der Working Set einer Datenbank: (links) verstreut über die DB, (mittig) nach Reorganisation der heißen Objekte in einen Bereich, (rechts) nach zusätzlicher Kompression der kalten Datenbank

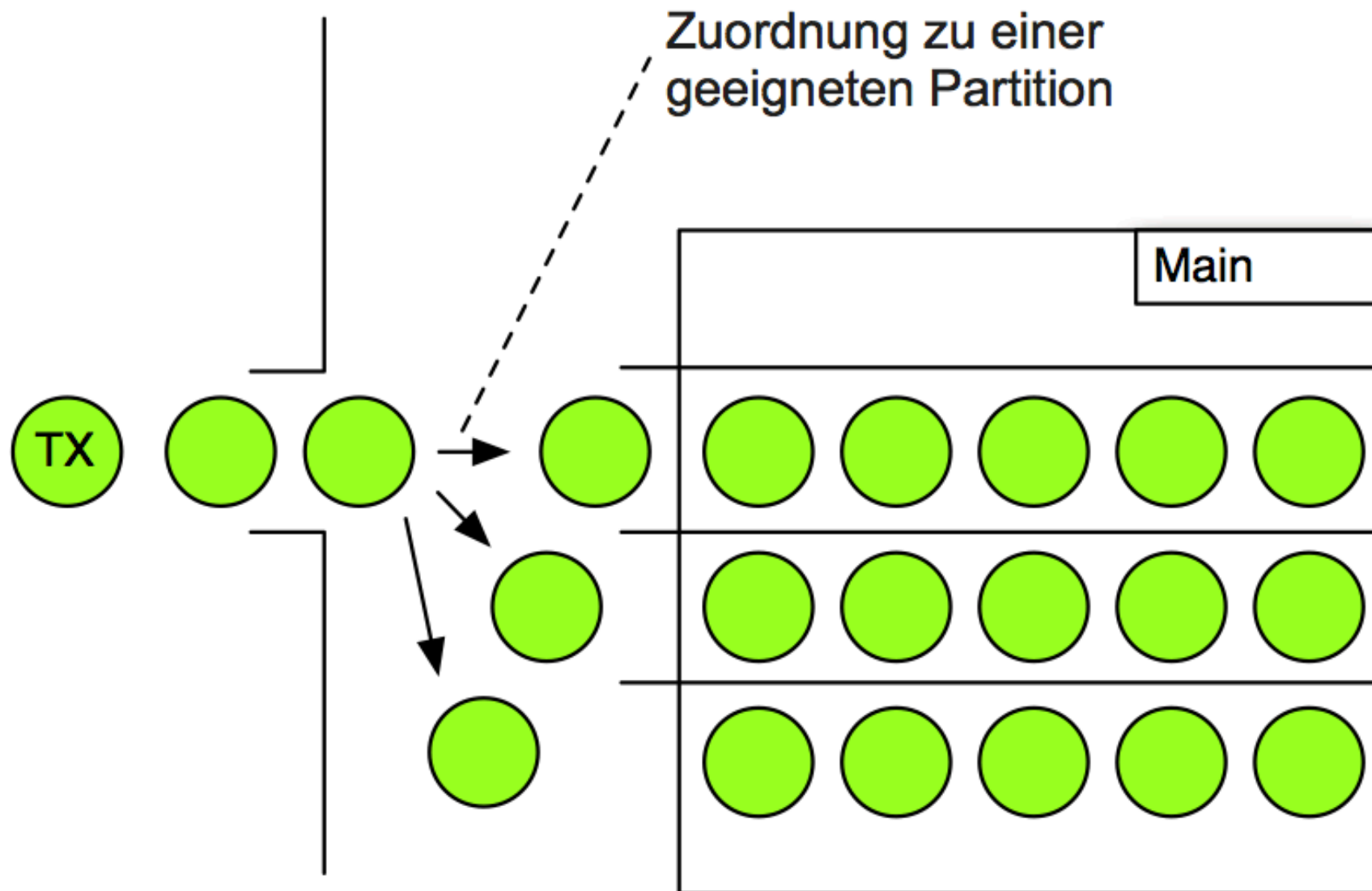
Kompaktifizierung der Datenbank



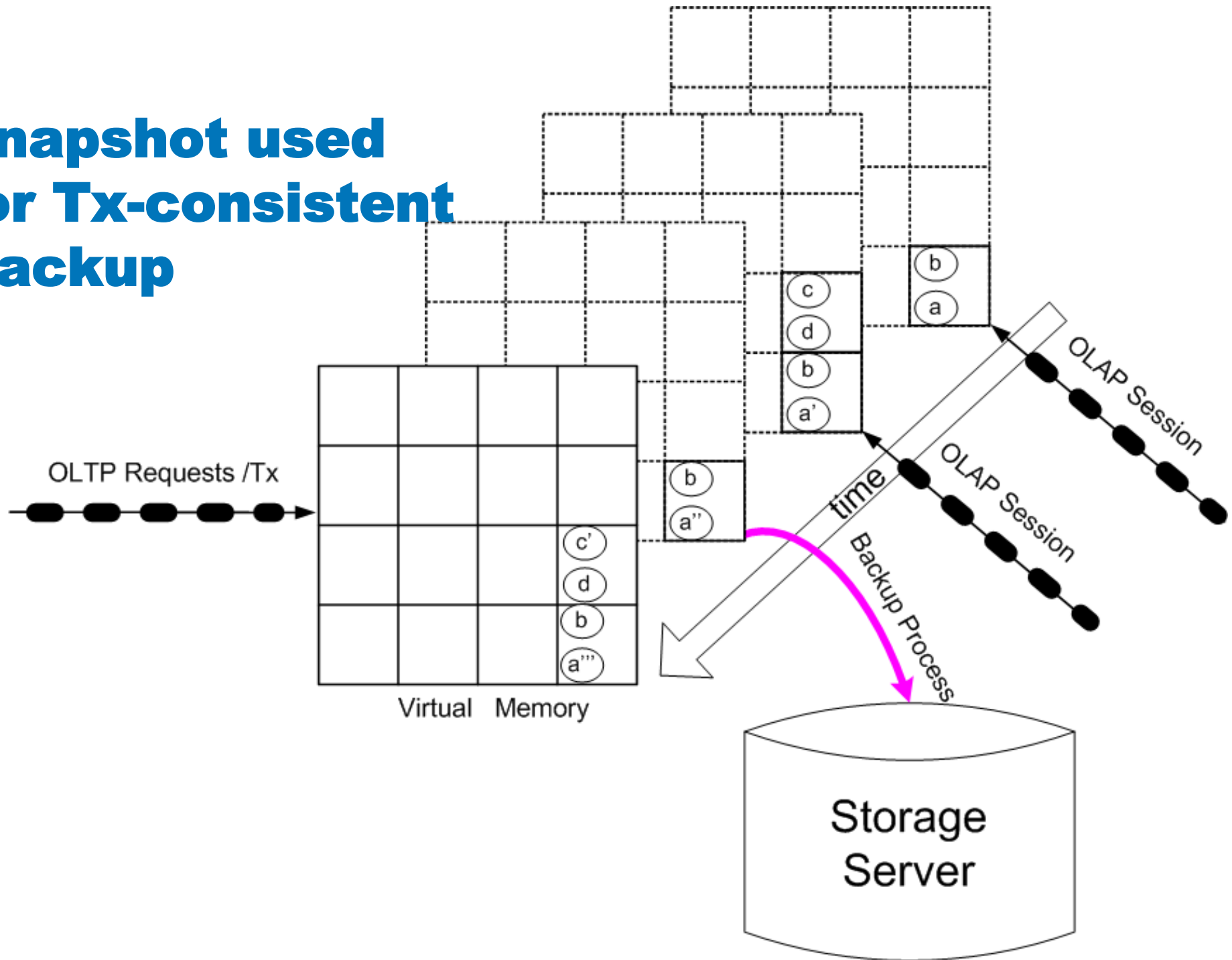
Invalidierung gefrorener Datenobjekte



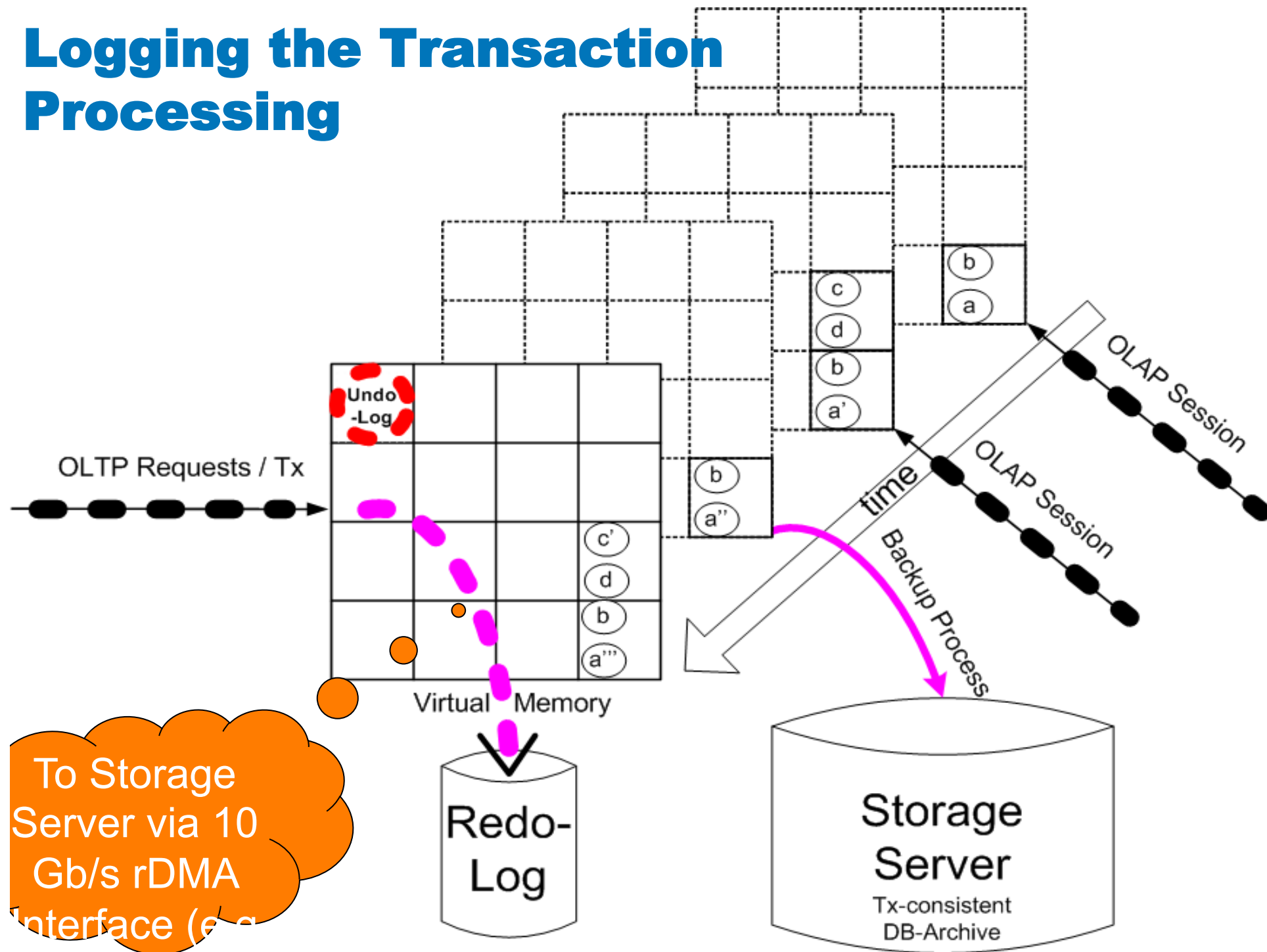
Transaktionsverwaltung: serielle Ausführung auf Partitionen



Snapshot used for Tx-consistent Backup

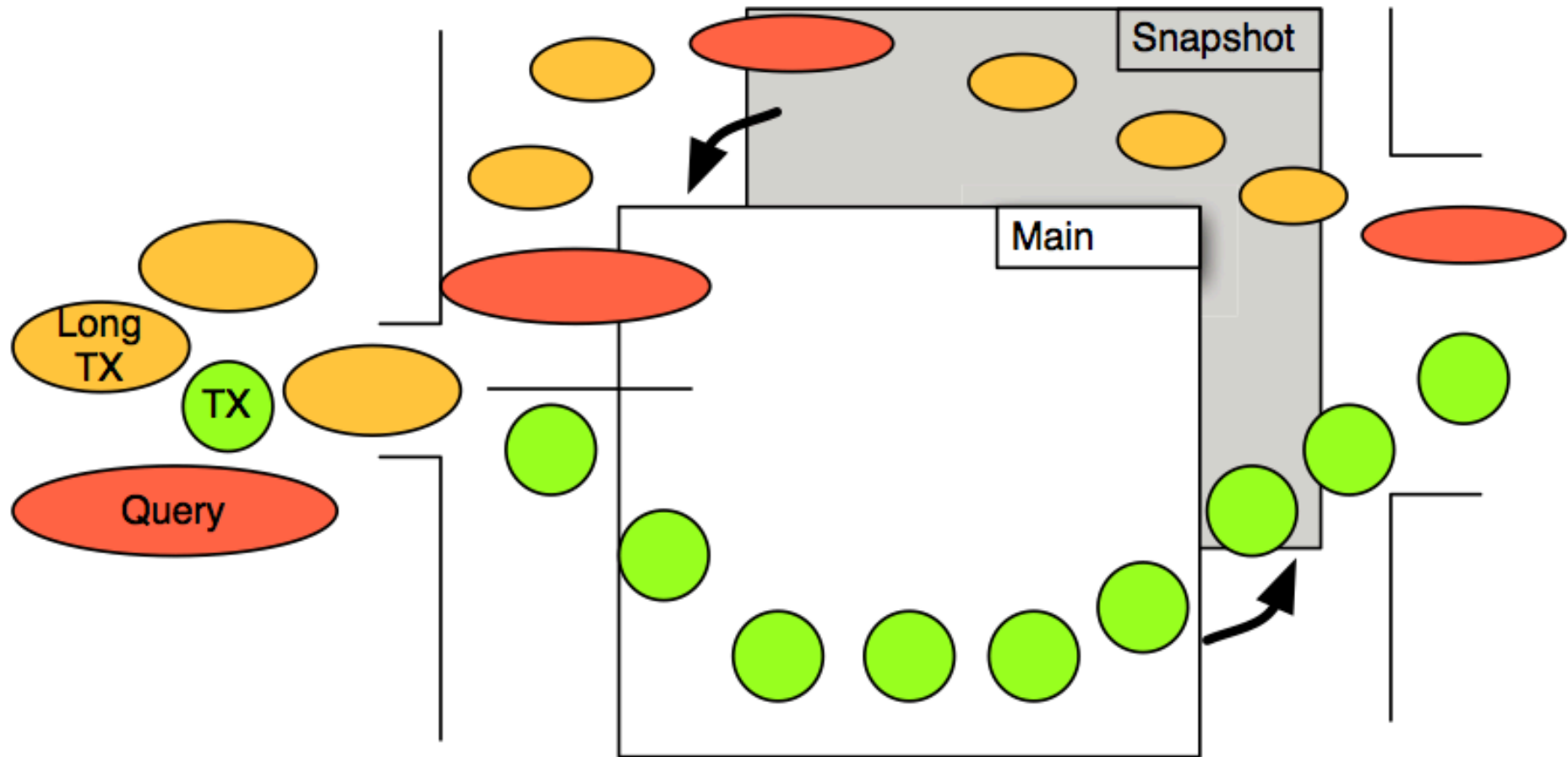


Logging the Transaction Processing

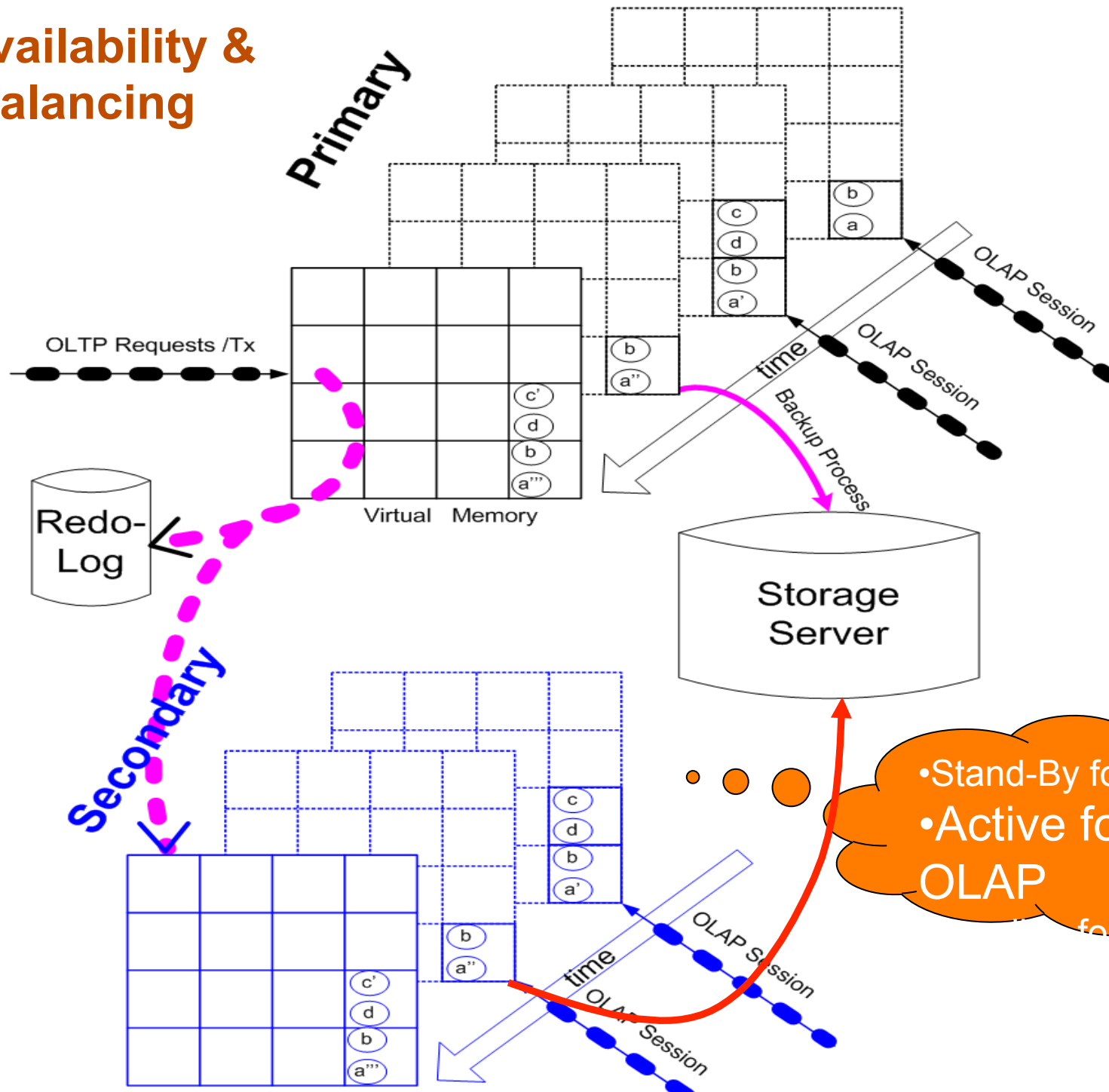


To Storage Server via 10 Gb/s rDMA Interface (e.g.)

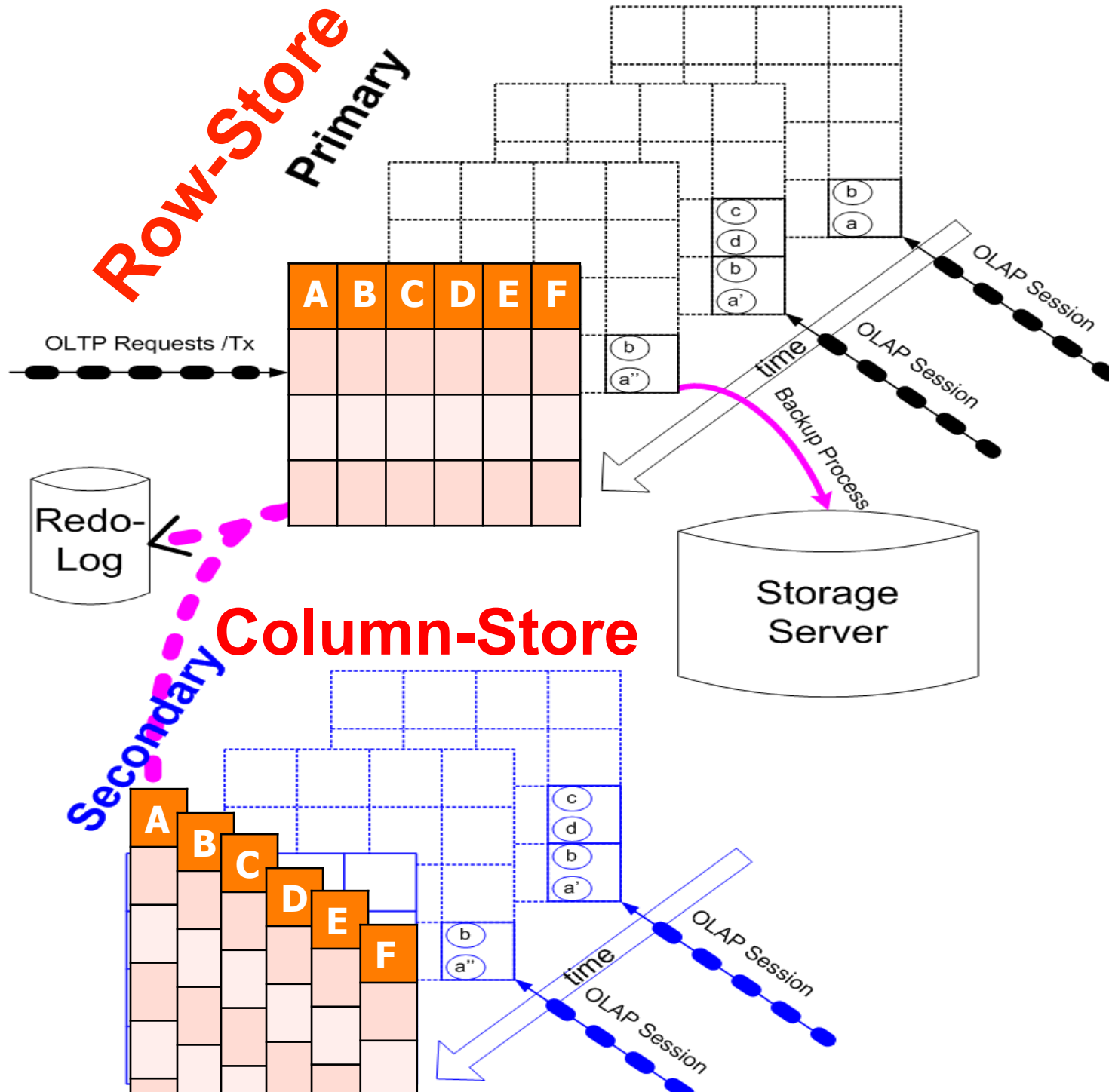
Tentative Ausführung langer Transaktionen



High Availability & Load Balancing

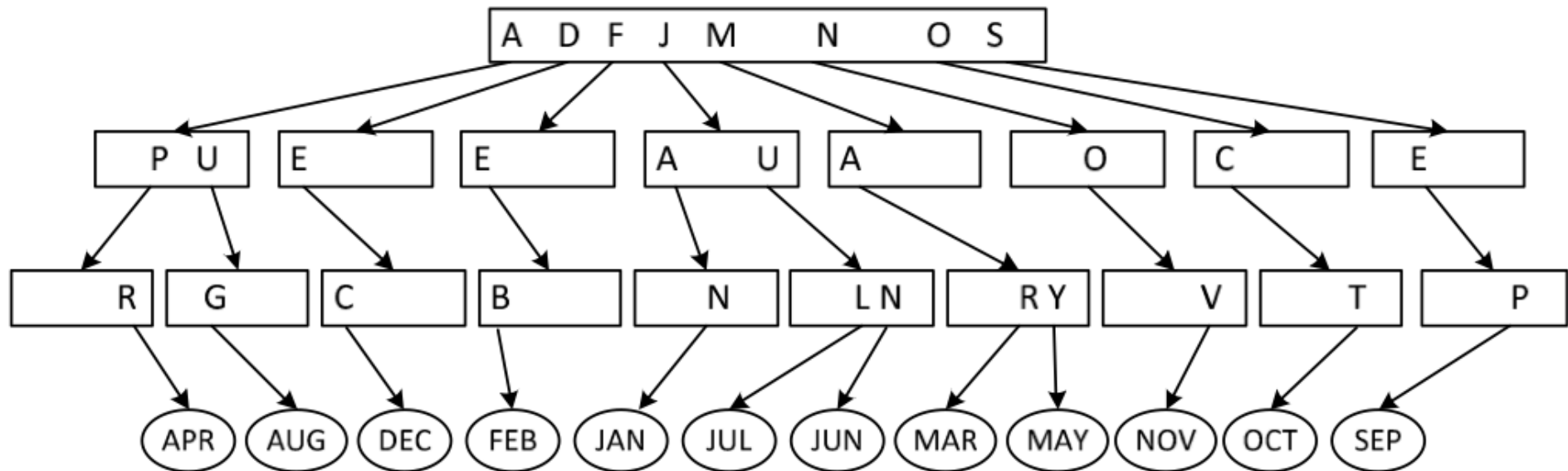


- Stand-By for OLTP
- Active for OLAP

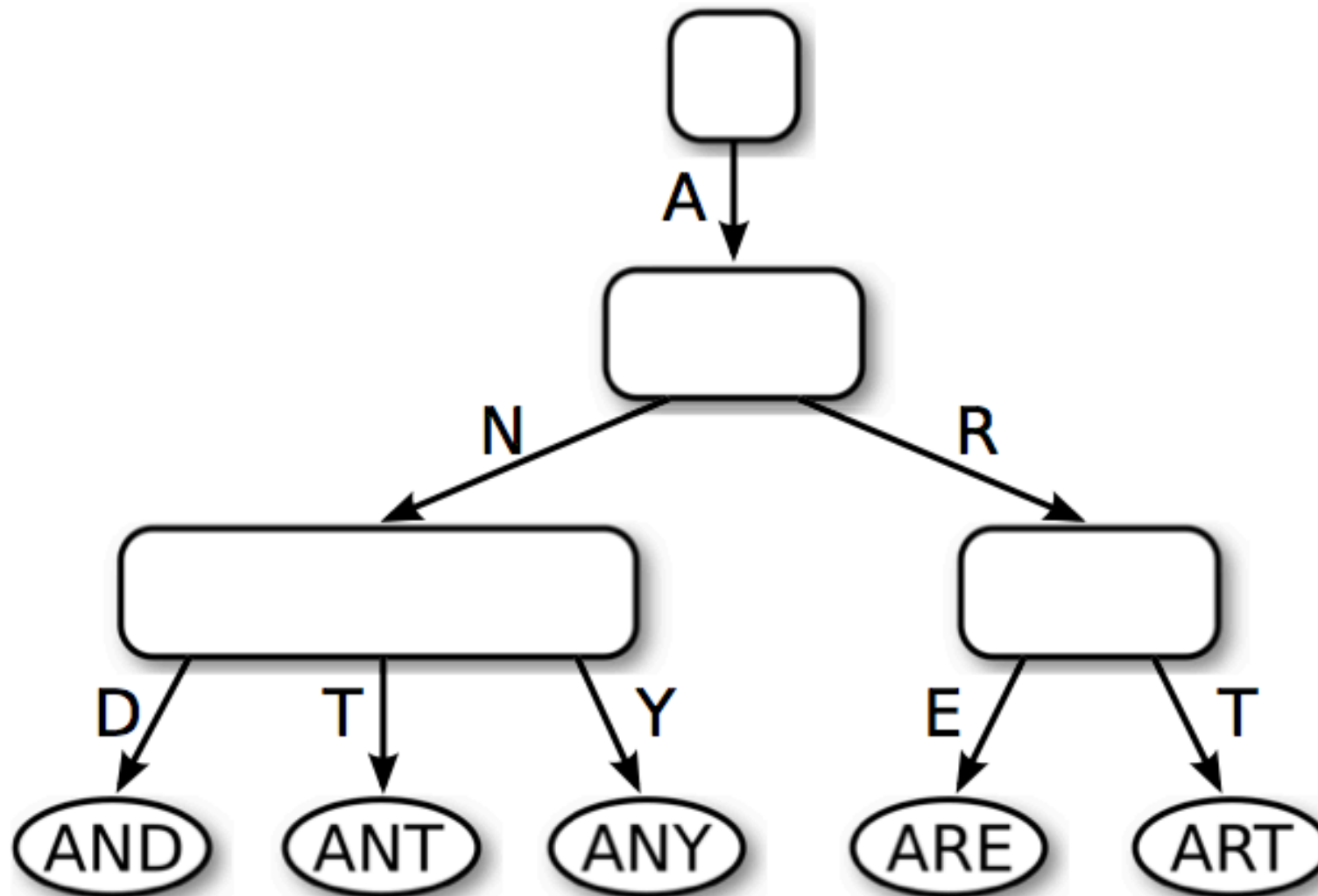


Indexstrukturen für Hauptspeicher-Datenbanken

- Radix-Baum / Trie / Präfixbaum



Idee des Adaptiven Radix-Baums ART

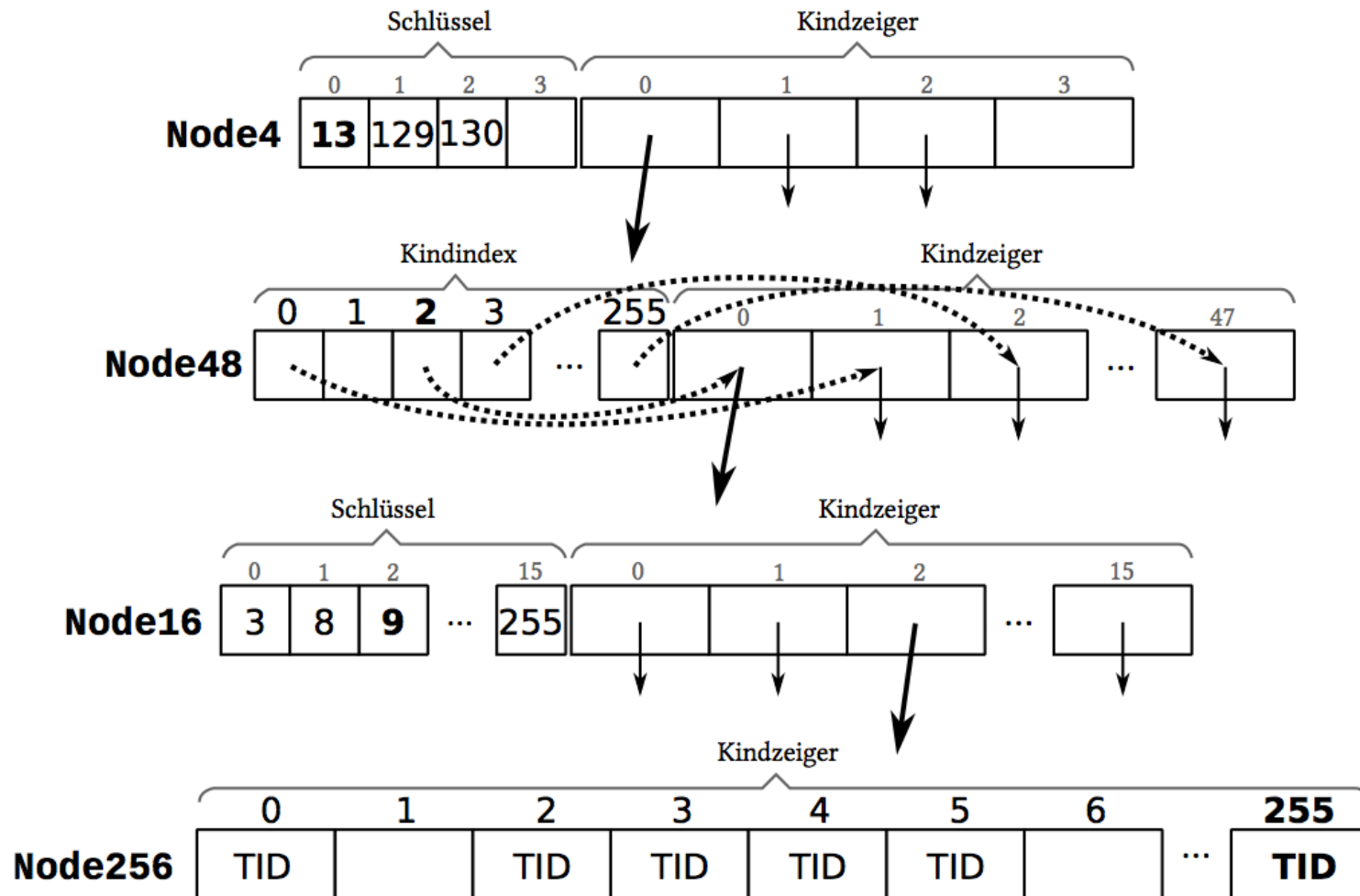


Adaptive Knoten des ART-Baums

Integer Schlüssel Bit-Repräsentation (32 bit, ohne Vorzeichen) Byte-Repräsentation

+218237439 00001101 00000010 00001001 11111111

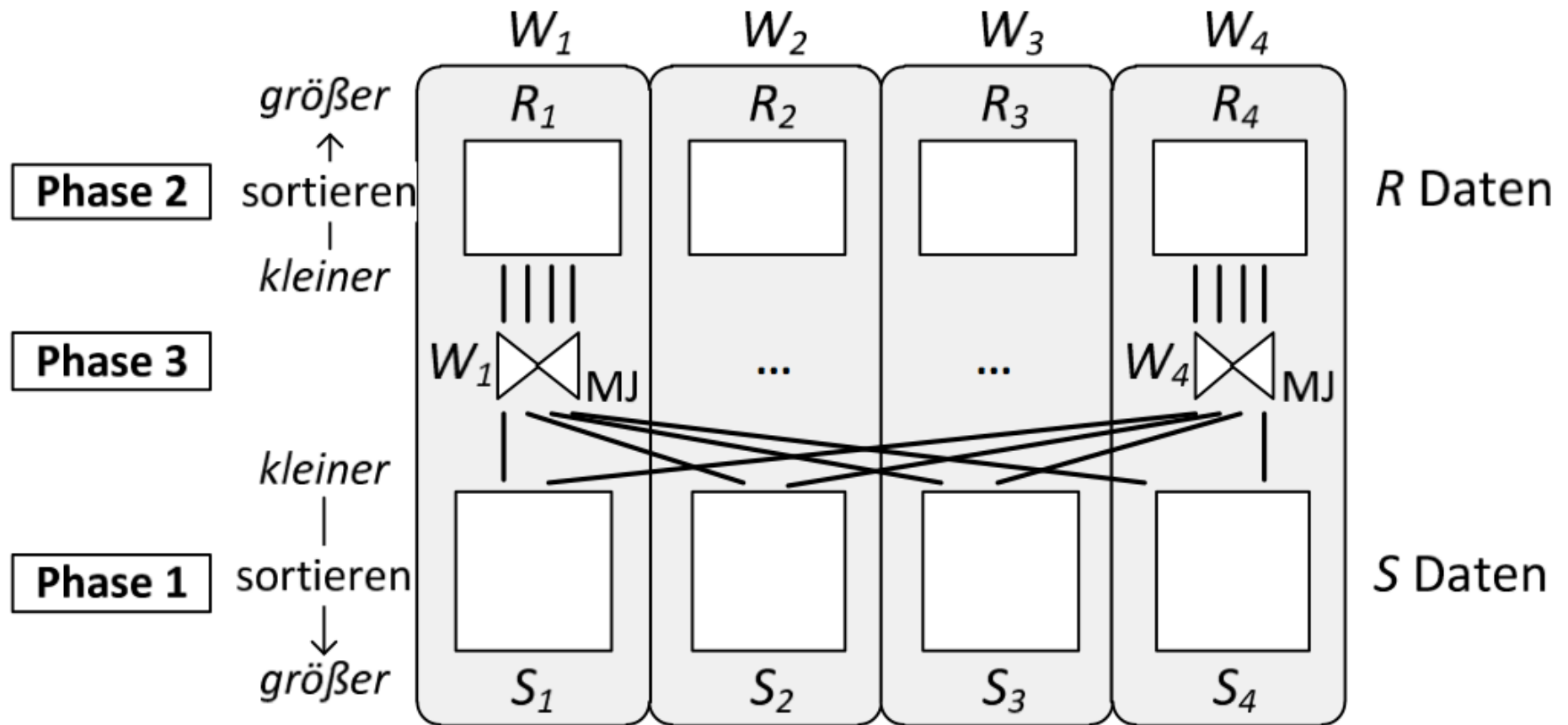
13	2	9	255
-----------	----------	----------	------------



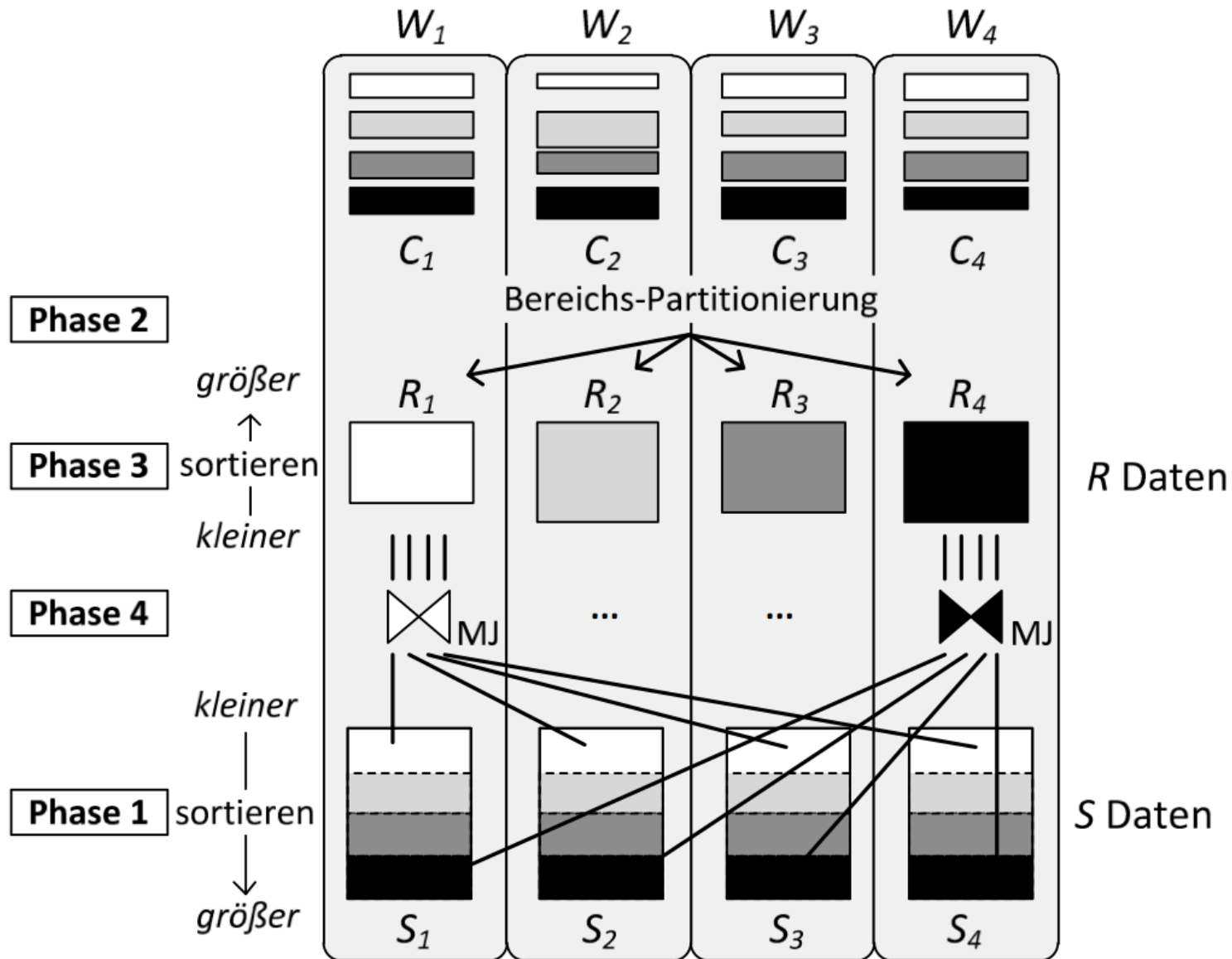
Join-Berechnung

- Cache-Lokalität
- Mehrkern-Parallelität
- NUMA-Berücksichtigung
- Synchronisations-freie Parallelität

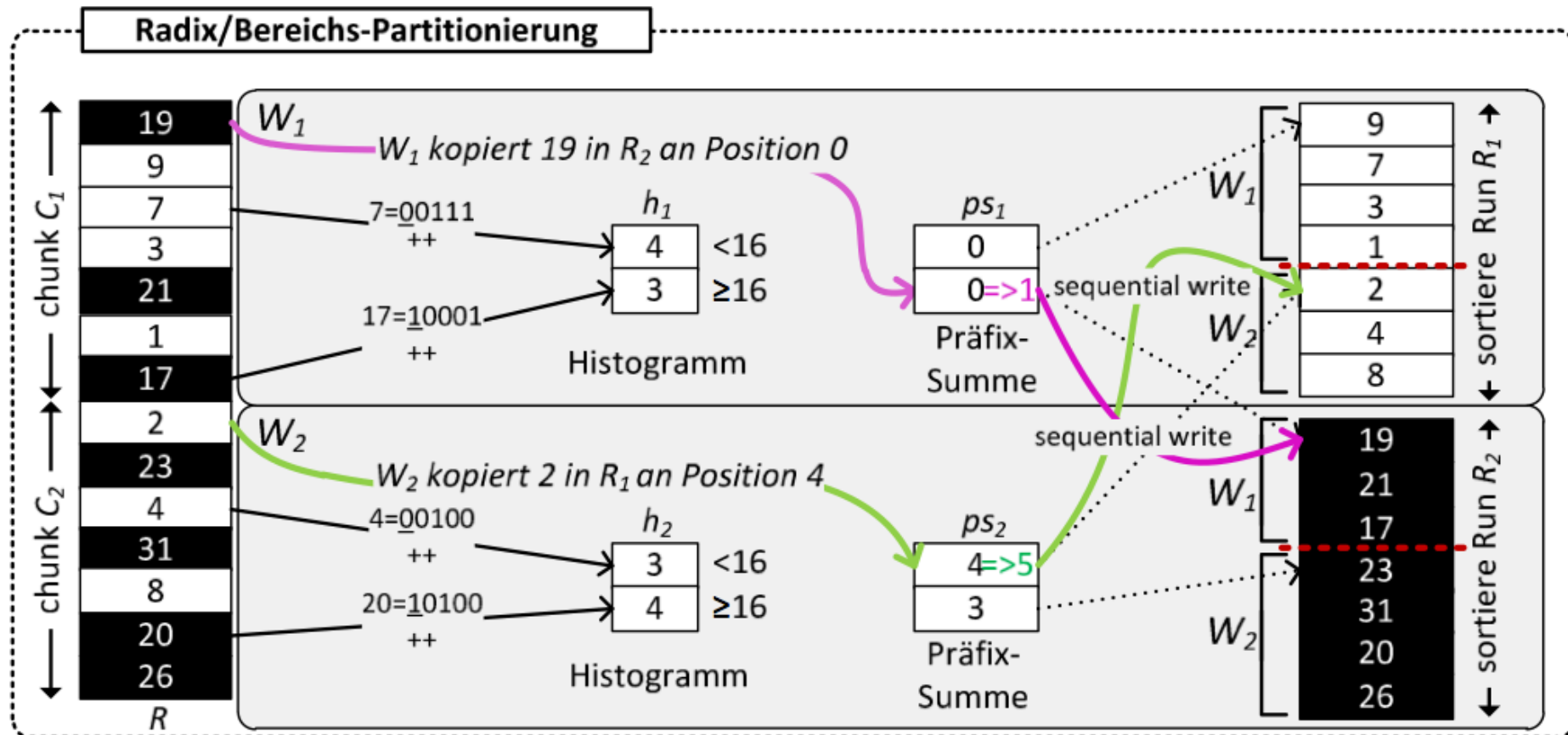
Grundidee des hoch-parallelen Sort/ Merge-Joins



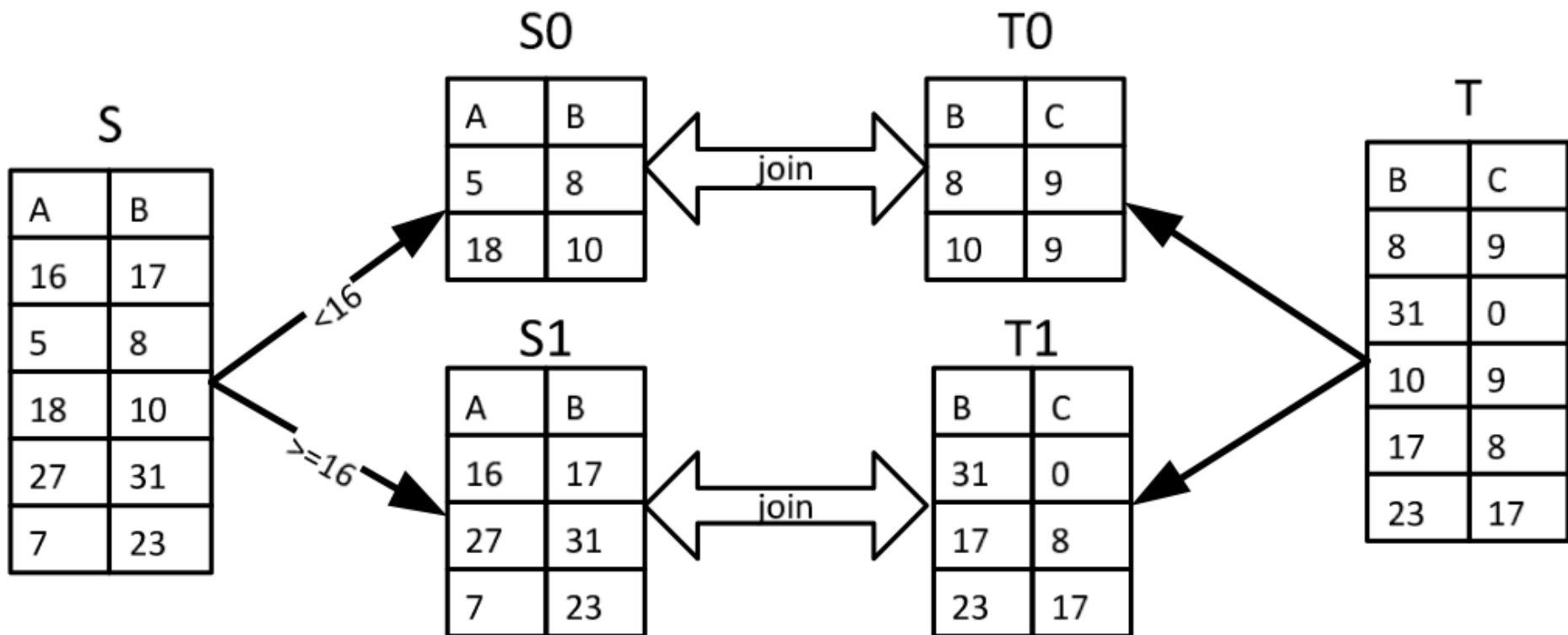
Bereichspartitionierung



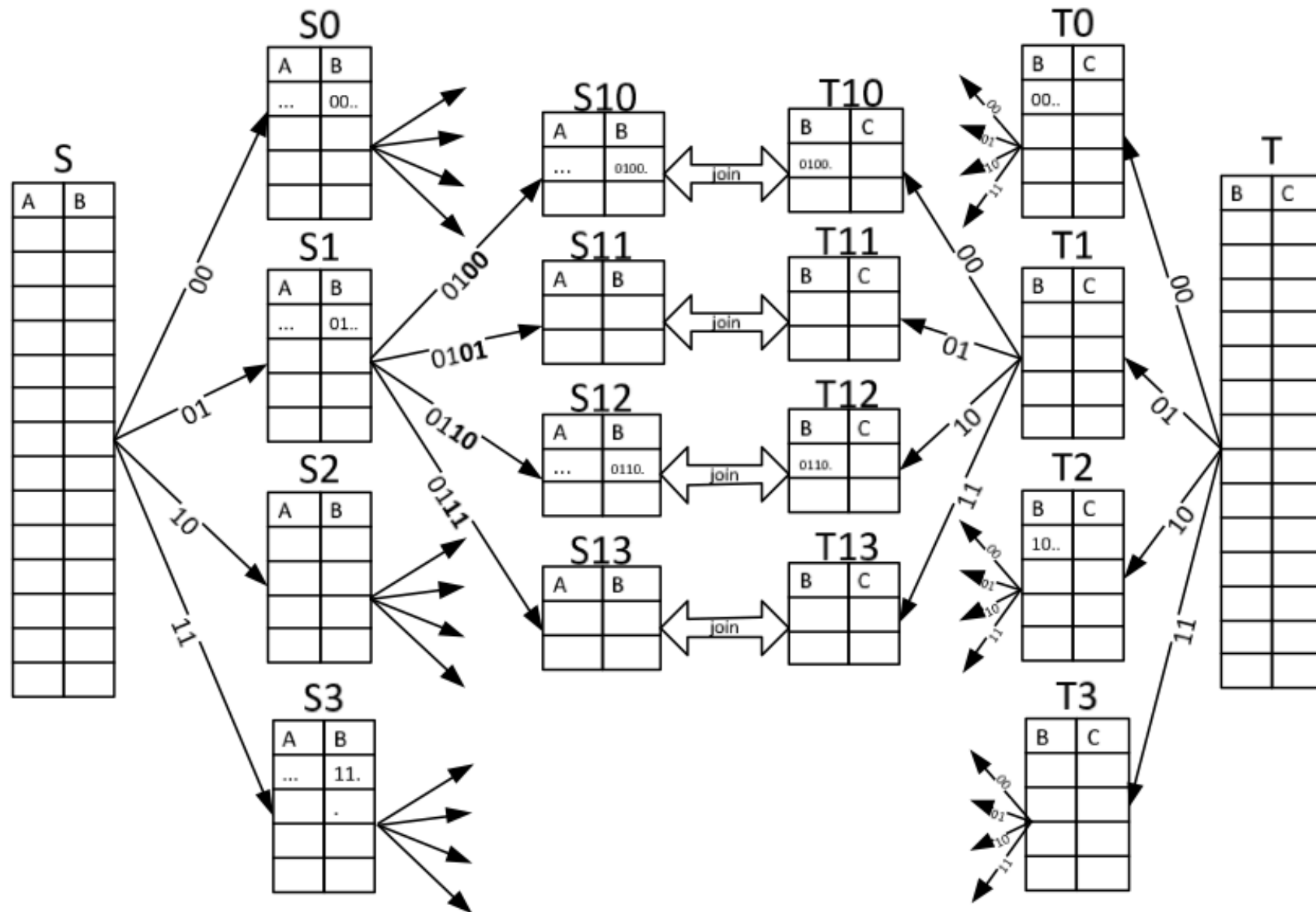
Hochparallel Bereichs/Radix-Partitionierung



Parallel Radix-Join



Mehrfache Partitionierung des Radix-Joins: Cache-Lokalität



Hash-Join-Teams: Globale Hashtabelle

