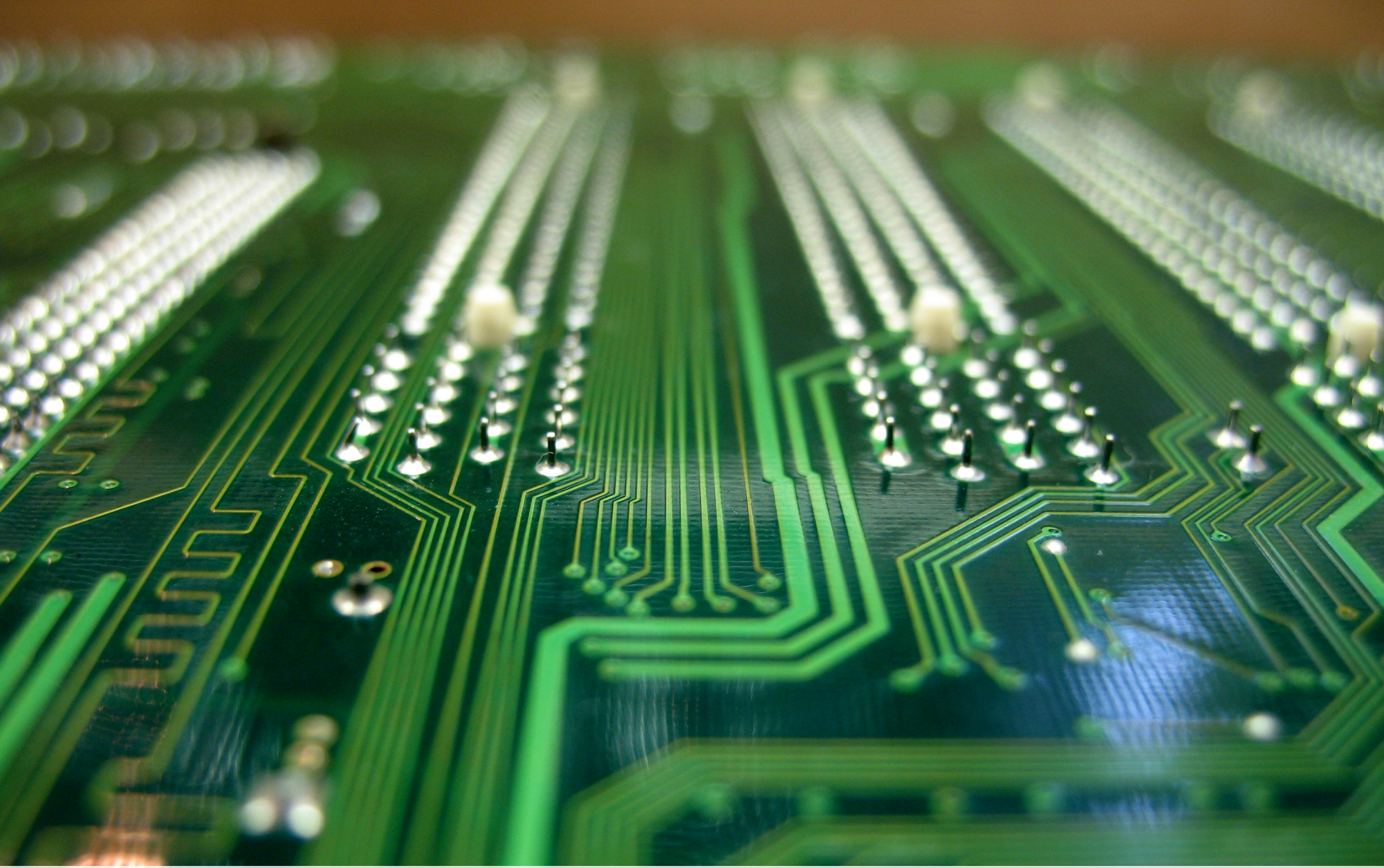


In-Memory Database Systems

Jens Teubner · TU Dortmund University

technische universität
dortmund

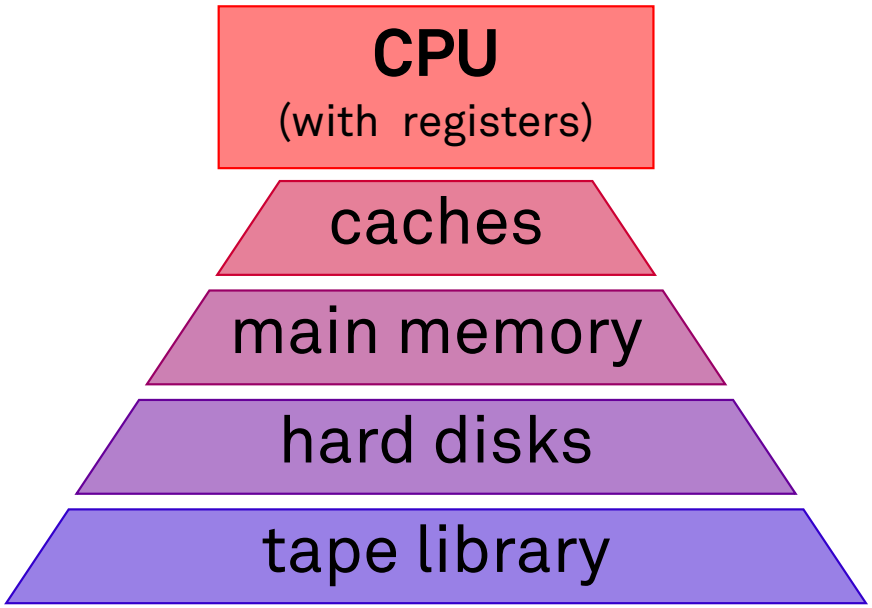
A Few Words About Me

Jens Teubner
DBIS Group
Otto-Hahn-Strasse 14
jens.teubner@cs.tu-dortmund.de

Databases and Information Systems Group:

- Databases on modern Hardware
  - in-memory, cache awareness, multi-core
  - data processing on graphics processors (GPUs)
  - data processing on FPGAs (programmable hardware)
- Energy-efficient database software
- Large-scale data processing ($\rightsquigarrow$ physics)

Database Folklore (Memory Hierarchy)



	capacity	latency
CPU (with registers)	bytes	< 1 ns
caches	kilo-/megabytes	< 10 ns
main memory	gigabytes	70–100 ns
hard disks	terabytes	3–10 ms
tape library	petabytes	varies

RAM vs. disk is like

- Grab the beer in front of you (cheers!) (2 sec)

versus

- Fly to Australia and get it from there (2½ days)

Database Technology

Folklore:

- Avoid disk access **“at any cost”**.
- Use **DRAM to cache disk data**.
  - → Database “buffer manager” manages caching.

Hmmm…

- Today’s systems can host terabytes of RAM.

Database Systems with Large Memories

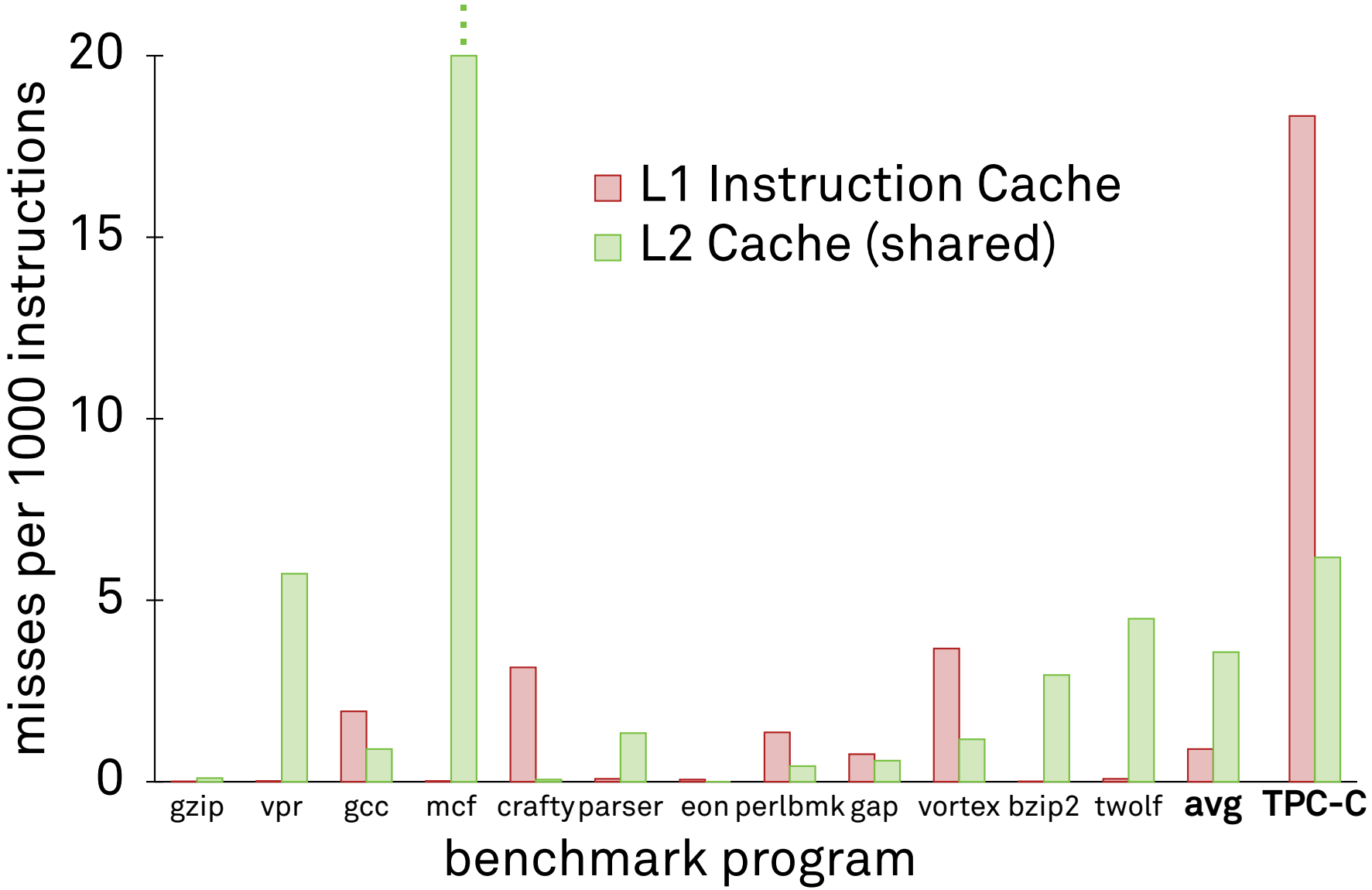
Databases, a concept that had its day?

- Buffer management $\rightarrow$ overhead
  - $\rightarrow$ "Hidden" behind disk latency in classical systems.
- Transaction management, persistence?

Actually...

- RAM $\leftrightarrow$ disk gap $\longrightarrow$ **cache $\leftrightarrow$ RAM gap**
- Buffer manager $\longrightarrow$ hardware

Effectiveness of Hardware Caches



A conventional database with a large memory will not do the trick.

E.g., SAP HANA

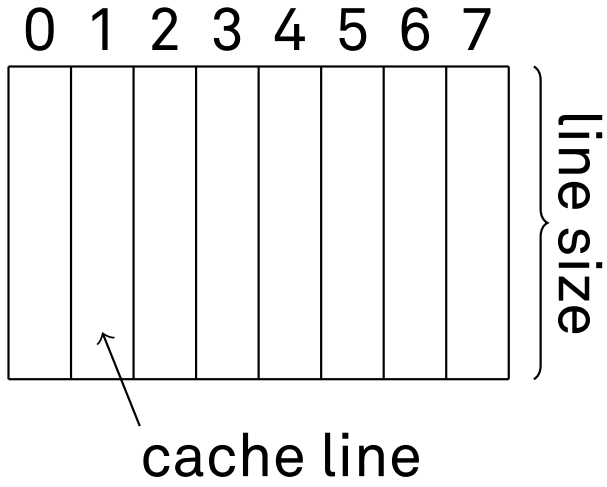
SAP HANA basiert auf der In-Memory-Technologie mit spaltenbasierter Datenspeicherung, die die Vorteile der Multi-Core-Verarbeitung und des SIMD-Instruktionssatzes (Single Instruction, Multiple Data) optimiert. Die Plattform ist vollständig kompatibel mit den Anforderungen des ACID-Standards (Atomarität, Konsistenz, Isolation und Dauerhaftigkeit). Mit intuitiven Modellierungswerkzeugen, Entscheidungsta-

- column-oriented storage?
- multi-core processing?
- SIMD instruction sets?

How CPU Caches Work

For high speed, the **overhead** of caching must be kept low.

- Organize cache in **cache lines**.
- Only load/evict **full cache lines**.
- Typical **cache line size**: 64 bytes.



- Block-wise transfers are well-supported by DRAM chips.
- Organization in cache lines $\leftrightarrow$ principle of locality

Database ↔ Cache Interaction

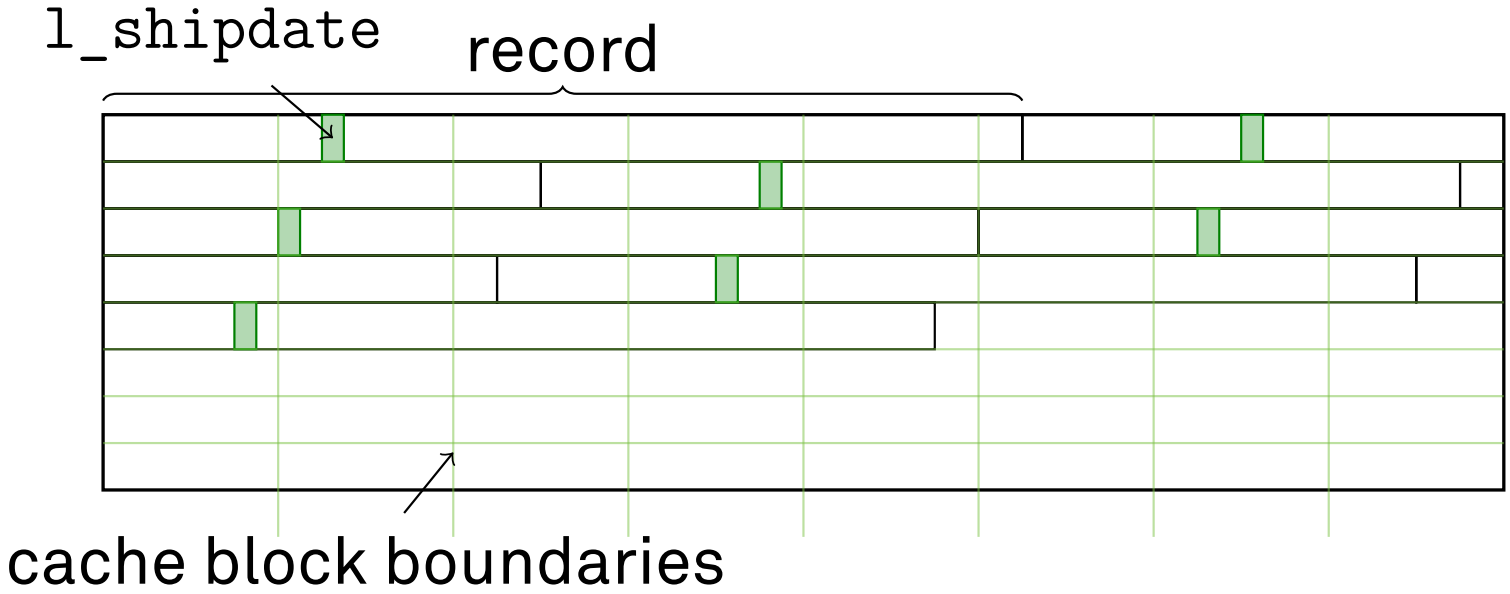
Consider, *e.g.*, a selection query:

```
SELECT COUNT(*)
  FROM lineitem
 WHERE l_shipdate = "2009-09-26"
```

- This query typically involves a **full table scan**.

Table Scans and Caches

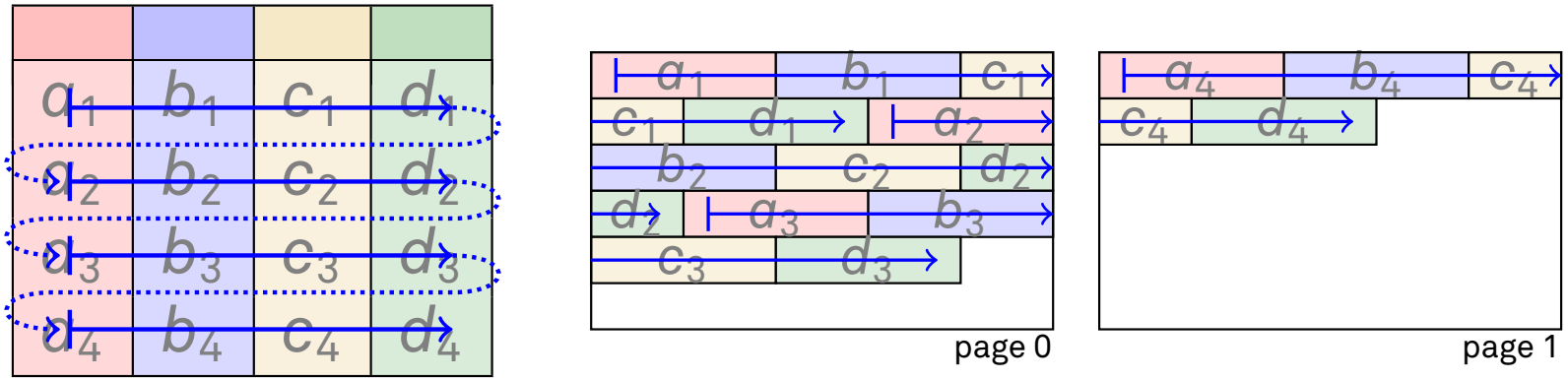
Tuples: **records** stored sequentially on a database page.



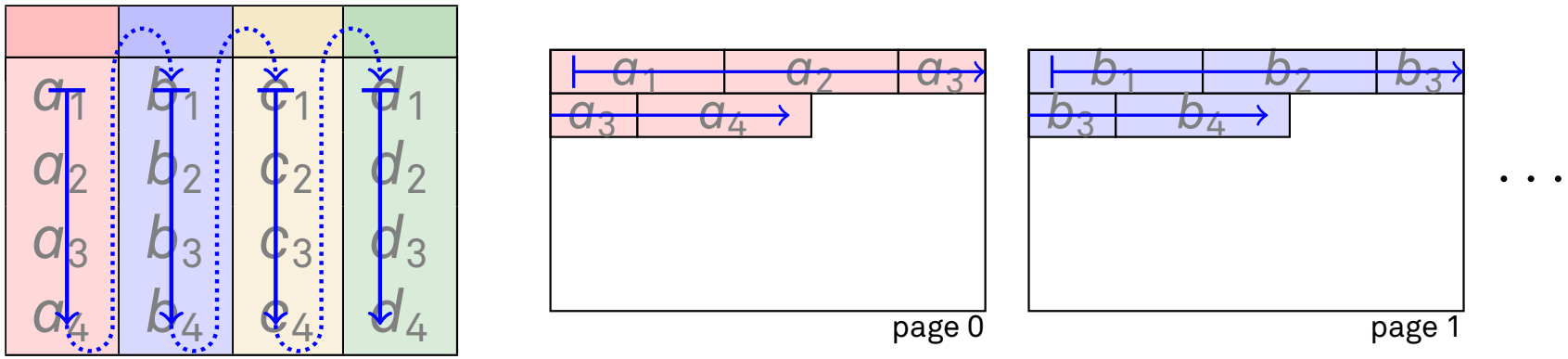
- With every access to a `l_shipdate` field, we load a large amount of **irrelevant** information into the cache.
- 4 bytes for `l_shipdate`, 64 bytes cache line size
  → 94 % garbage fetched!

Row-Wise vs. Column-Wise Storage

This is when we assume **row-wise** storage (also: *n*-ary storage model, NSM).



Column-wise storage (decomposition storage model, DSM):



Column Stores in Commercial DBMSs

Commercial databases have added column-store functionality to their engines:

- **Microsoft SQL Server**:
  - Represented as “Column Store Indexes”
  - Available since SQL Server 11
  - see Larson *et al.*, SIGMOD 2011
- **IBM DB2**:
  - IBM DB2 “BLU Accelerator”, a column store that ships with DB2 10.5.
  - BLU stands for “Blink Ultra”; Blink was developed at IBM Almaden (↗ Raman et al., *ICDE 2008*).

Why aren’t all databases built as row stores then?

MonetDB: Binary Association Tables

MonetDB makes this explicit in its data model.

- **All** tables in MonetDB have two columns (“head” and “tail”).

oid	NAME	AGE	SEX
o_1	John	34	m
o_2	Angelina	31	f
o_3	Scott	35	m
o_4	Nancy	33	f

$\rightarrow$

oid	NAME
o_1	John
o_2	Angelina
o_3	Scott
o_4	Nancy

oid	AGE
o_1	34
o_2	31
o_3	35
o_4	33

oid	SEX
o_1	m
o_2	f
o_3	m
o_4	f

- Each column yields one **binary association table (BAT)**.
- **Object identifiers** (`oids`) identify matching entries (BUNs).
- Oftentimes, `oids` can be implemented as **virtual** `oids` (`voids`).
  - $\rightarrow$ Not explicitly materialized in memory.

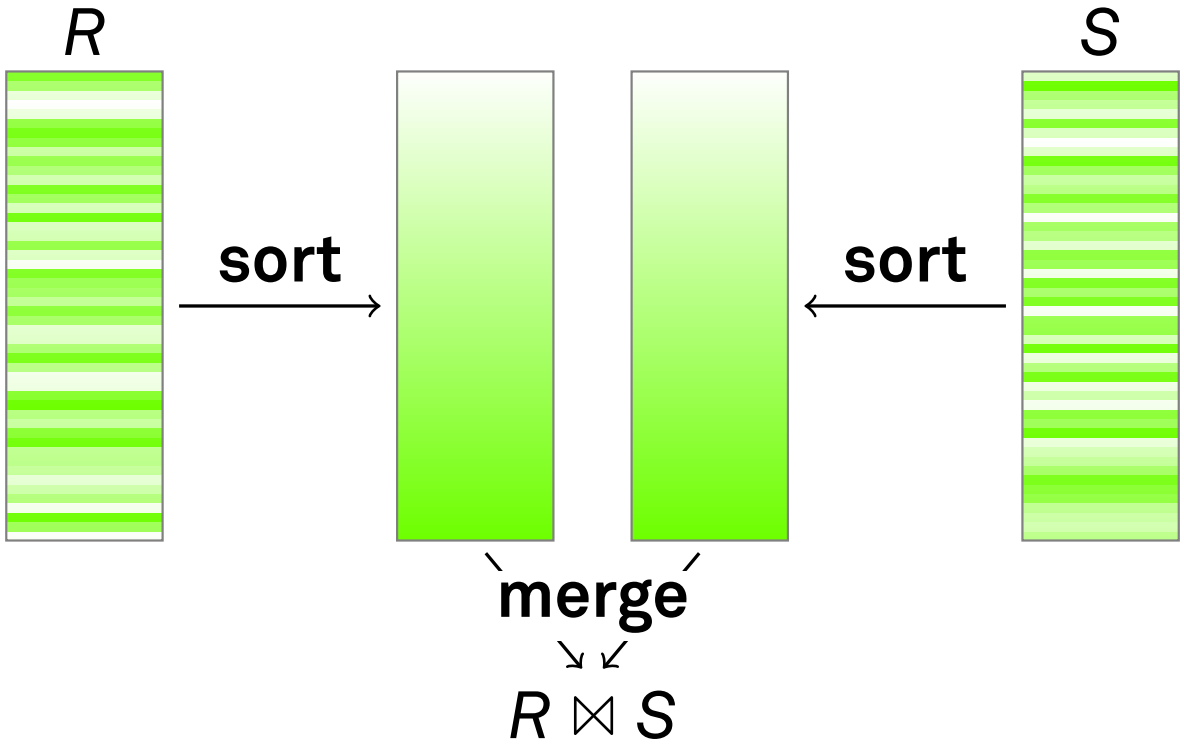
Binary Association Tables

Simplicity of two-column setting → **aggressive optimization**.

Example: `batval_int_add(···)` (*i.e.*, `[+](int,BAT[any,int])`)

```
⋮
if (vv != int_nil) {
    for (; bp < bq; bp++, bnp++) {
        REGISTER int bv = *bp;
        if (bv != int_nil) {
            bv = (int) OP(bv,+,vv);
        }
        *bnp = bv;
    }
} else {
    for (; bp < bq; bp++, bnp++) {
        *bnp = vv;
    }
}
⋮
```

Sort-Merge Join

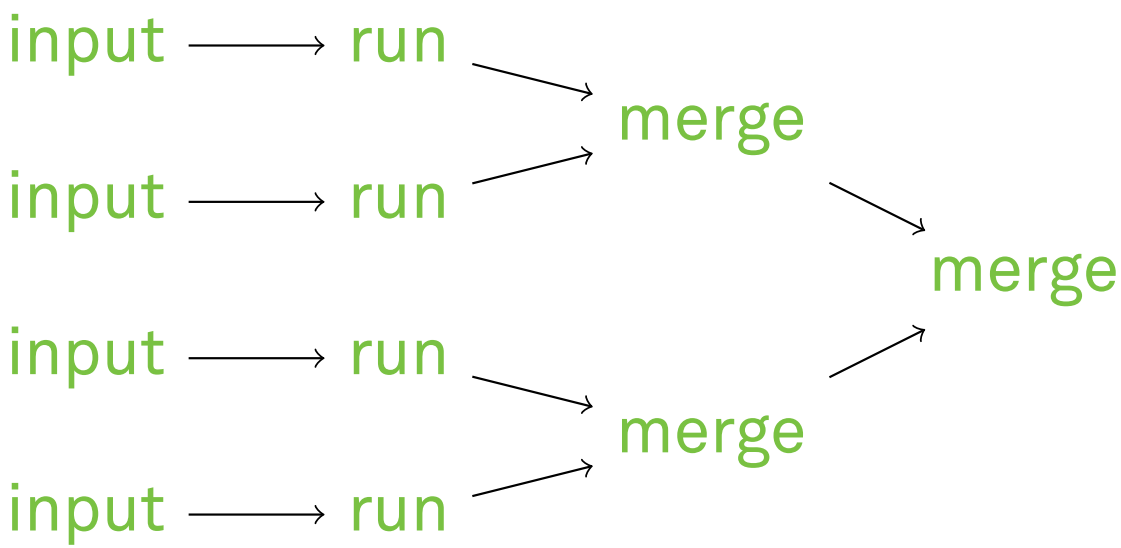


- ✓ Can be done as **external sort**
- ✓ $\mathcal{O}(N \log N)$

Sorting in Databases

The critical part of sort-merge join is **sorting**.

- Method of choice: **merge sort**
  - → two parts: **run generation** and **merging**

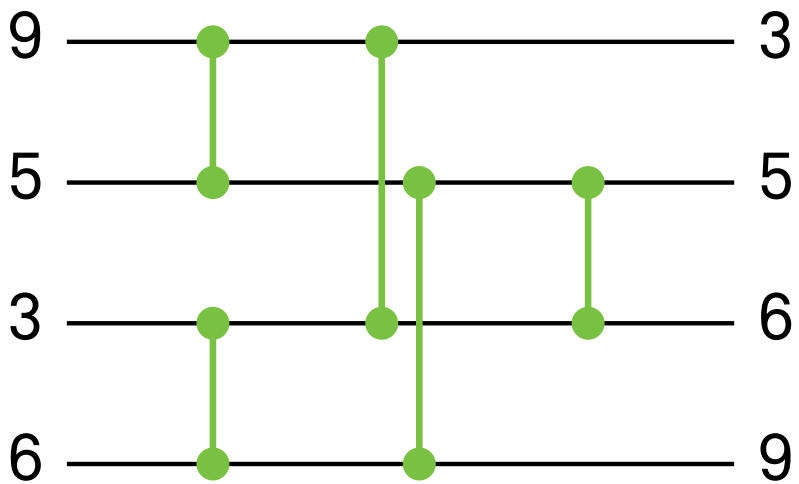


→ Both are good candidates for **SIMD acceleration**

Sorting networks

→ **branch free**, supports **data parallelism**

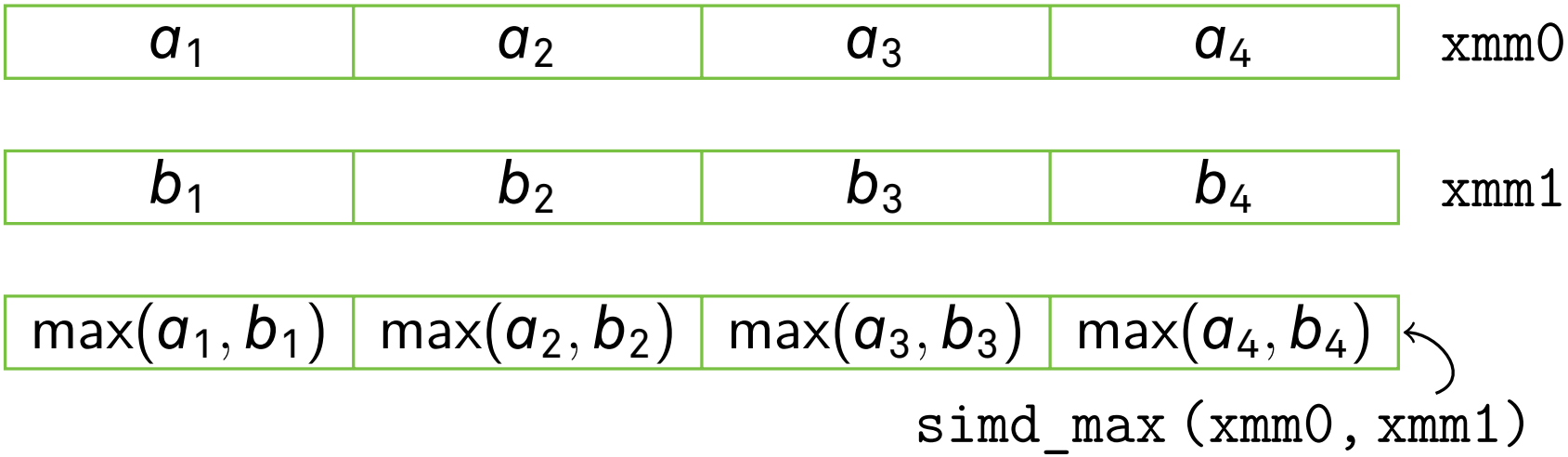
E.g., network for four elements (“even-odd network”):



→ Build larger networks by **merging** sorted runs.

SIMD instructions

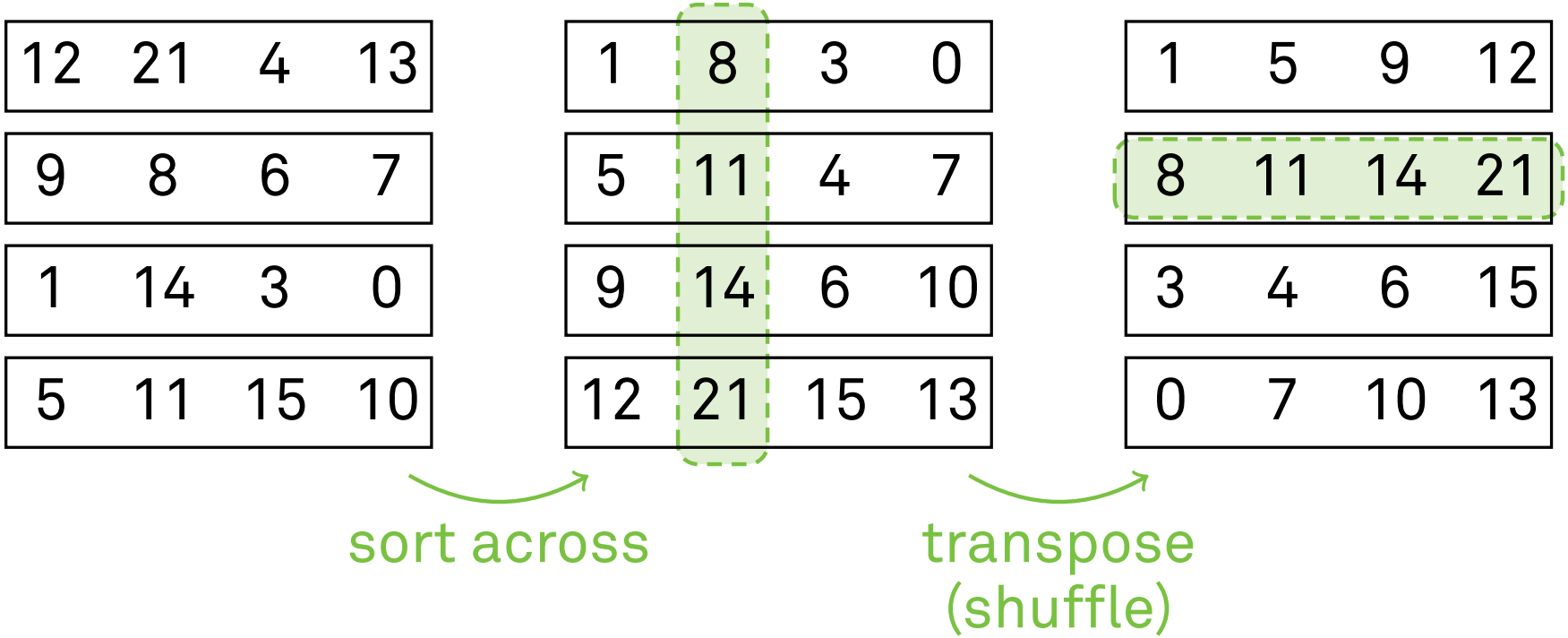
E.g., four words per SIMD register:



Operations **across** registers, **not within**

→ **But:** Can **shuffle** across and within

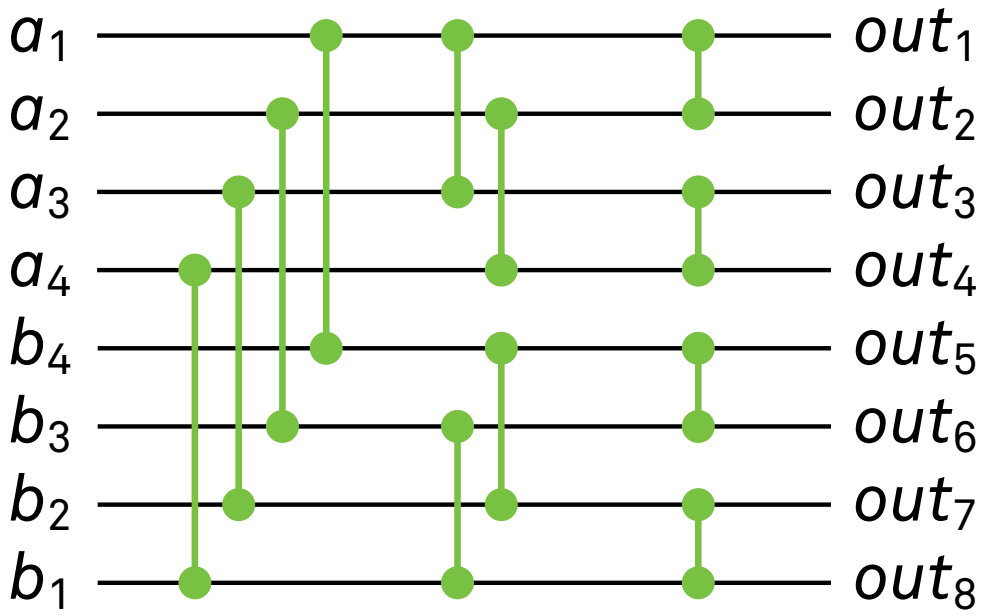
Run generation



- 10 min/max, 8 shuffle, 8 load/store
- 64 bytes in, 64 bytes out (128-bit SIMD)

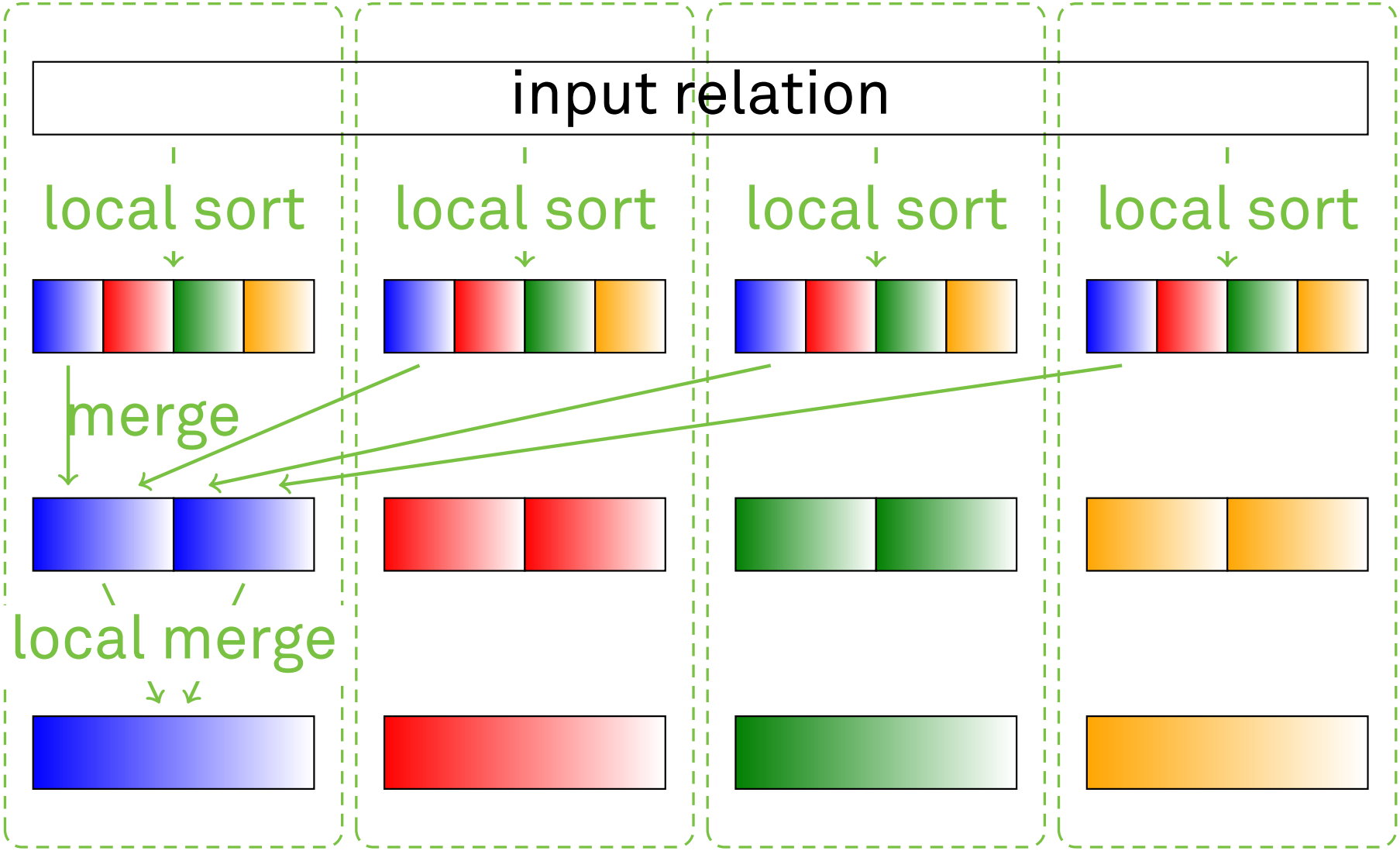
Merging

Two sorted runs, four items each:

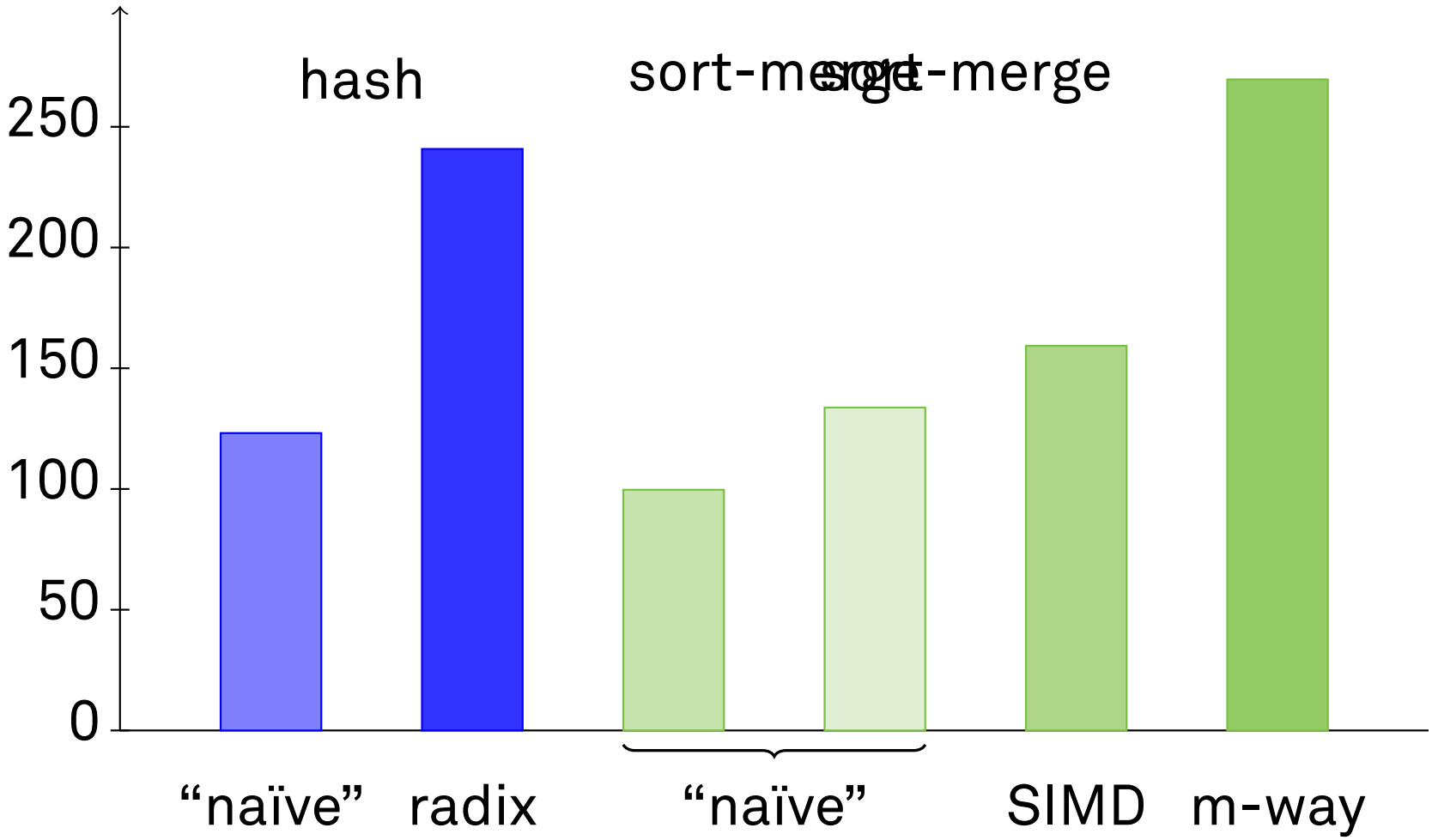


- Input: two SIMD registers **a** and **b**, sorted
- 6 min/max, 10 shuffle, 4 load/store

Sorting and NUMA



Sort or Hash?



- 12.8 GB ⋈ 12.8 GB
- Intel E5-4640 (“Sandy Bridge”), 2.4 GHz, 32 cores

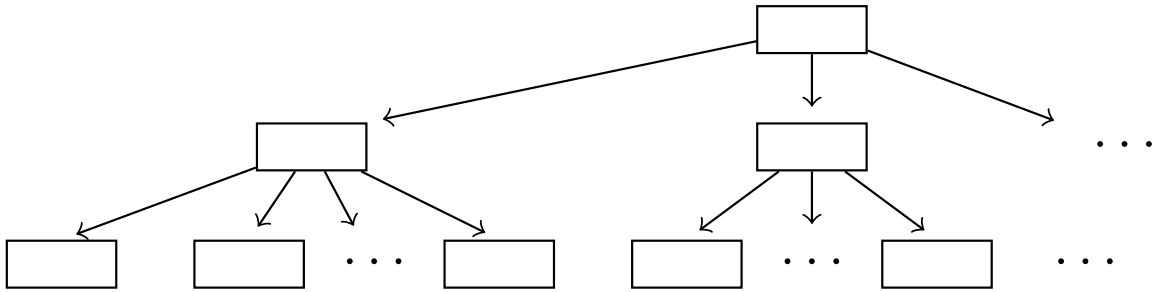
Concurrency

fast in-memory processing
high degrees of hardware parallelism

increased **concurrency** in the system
correctness? scalability?

E.g., B-Trees

B-trees are **the** indexing mechanism used in databases today.



Challenge:

- Update operations may change **structure** of the B-tree.
- Concurrent transactions must not see inconsistent tree.

Defensive solution:

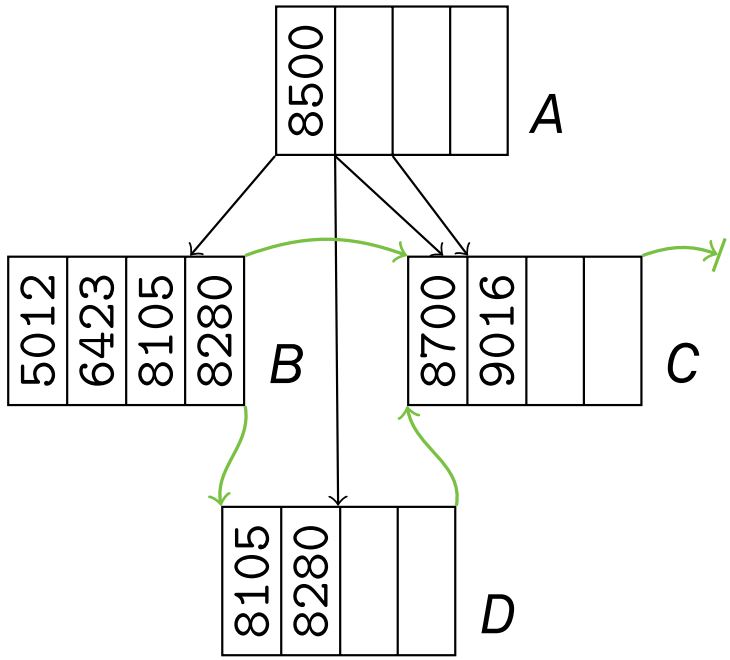
- Generously **lock** (“latch”) parts of the tree.

B^{link}-Trees: “Single-Latch B-Trees”

B-tree variant with **sibling pointers.**[1]

```
1  latch & read page B;
2  create and latch new page D;
3  populate page D;
4  set next pointer D → C;
5  un-latch D;
6  set next pointer B → D;
7  adjust content of B;
8  un-latch B;

9  latch & read A;
10 adjust content of A;
11 un-latch A;
```



→ Lines 9–11 can be deferred to a later time.

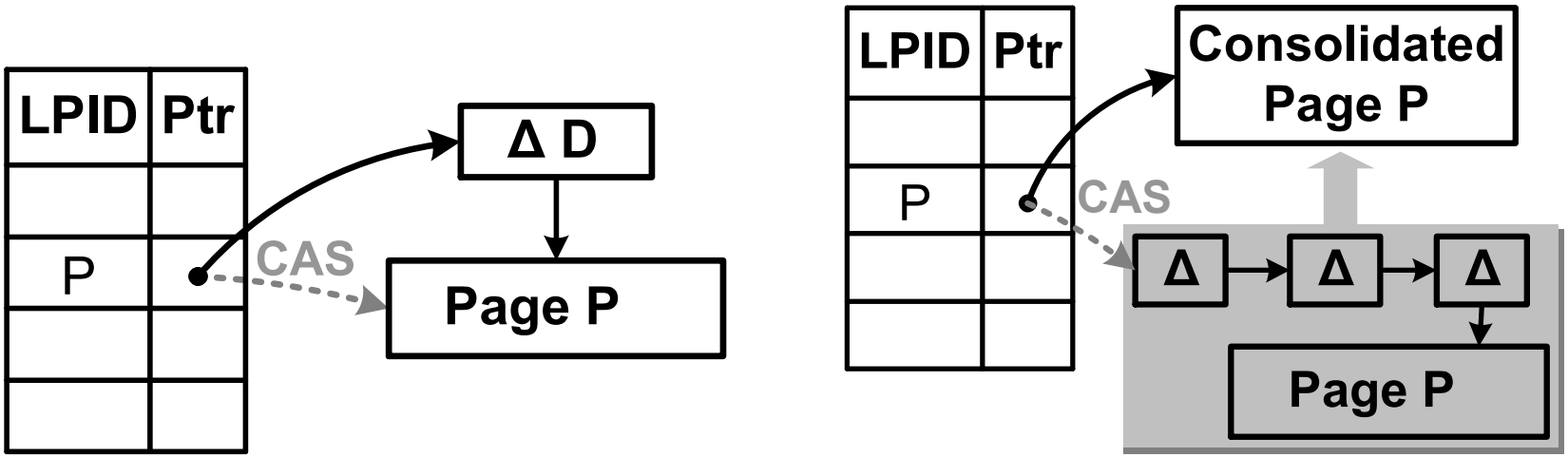
[1] PostgreSQL uses B^{link}-trees.

(Truly) Latch-Free B-Trees?

Avoid latching altogether?

Bw-tree:[2]

- **Compare-and-swap** instead of latching
- **Delta records** to reduce update volume
- **Logical addressing** to update nodes atomically.



[2] Levandoski *et al.* The Bw-Tree: A B-tree for New Hardware Platforms. ICDE 2013.

Bw-Trees

- Implementation heavily optimized ($\rightsquigarrow$ caches).
- Part of Microsoft SQL Server (“Hekaton”).
- Ready for **flash-based storage**.
- Ready for **non-volatile memories**.

Wrap-Up

“In-Memory Databases”

- **Not** just a database and a large RAM!
- Software tailored toward
  - memory/cache efficiency
  - CPU efficiency (including SIMD, etc.)
  - hardware parallelism
  - ...

Specifically, I discussed

- cache awareness $\rightarrow$ **column stores**
- CPU efficiency $\rightarrow$ **SIMD**, **multi-core**, **NUMA**
- concurrency $\rightarrow$ **latch-free indexing, Bw-tree**