Database Technology Group

TECHNISCHE UNIVERSITÄT DRESDEN

Adaptive Query Processing on Prefix Trees

Wolfgang Lehner

Fachgruppentreffen, 22.11.2012 – TU München

> Challenges for Database Systems

“Three things are important in the database world: performance, performance and performance.“
Bruce Lindsey

Extreme data

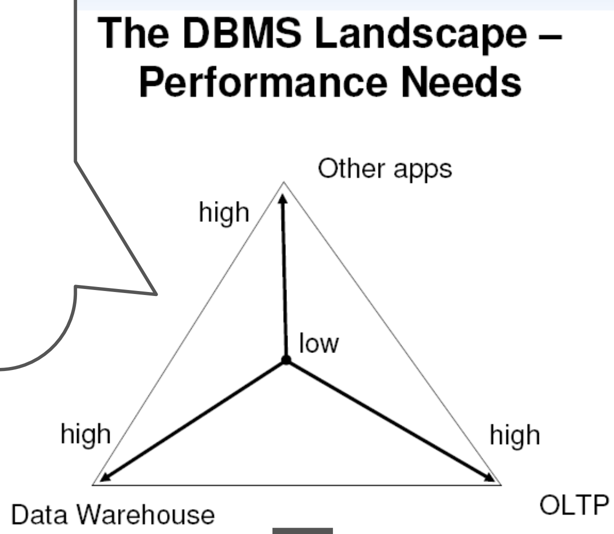
hadoop

Azure

amazon.com

Dynamo

HIVE



Extreme performance

SYBASE

ASTER

SAP

Greenplum

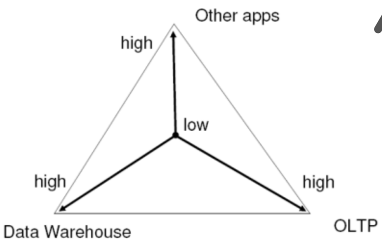
NETEZZA

> Challenges for Database Systems

Flexibility in Database Systems

- **+ during deployment time** (schema definition)
- **- during database lifetime** (schema evolution)
- **- during query runtime** (scheduling, …)

Extreme Performance



Extreme Flexibility

Extreme Data

> Flexibility from 10.000 feet

applications

data comes first, schema comes second

querying Web-Tables

role-based object models

Open Data platforms

Demand flexibility

Flexibility is cross-cutting

Database System

- *alternative query model (drill-beyond)*
- *domain-specific data models*
- *statistical operators*

data model- related

storage model- related

- record management
- logging an persistency
- HW/SW DB-CoDesign
- (MV)CC

query processing

Provide flexibility

operating system & hardware

StorageClassMemory

MegaCore systems

TeraByte-Capacity

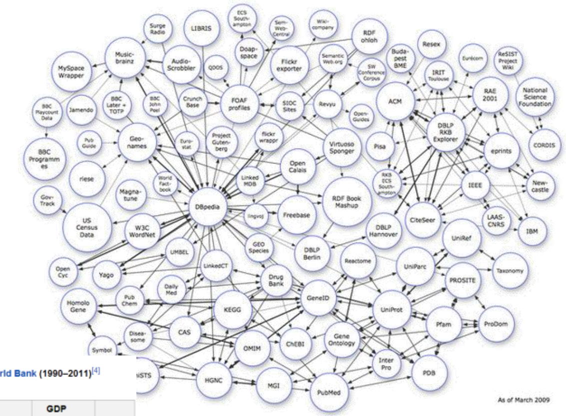
GPUs

FPGAs/FPPAs

> Example 1: Data-driven Schema Design

1998	2002	2006	2008	2012
GSM	GSM TriBand	GSM, GPRS, Bluetooth, FM	GSM, UMTS, Bluetooth, USB, WiFi	GSM, UMTS, HSPA+, Bluetooth, FM, USB, WiFi, aGPS, HDMI
Dimensions/Weight	Dimensions/Weight/ Form Factor	Dimensions/Weight/ Form Factor	Dimensions/Weight	Dimensions/Weight
DotMatrix Display	DotMatrix Display	2.0”, QVGA, Color	3.5”, 320×480, Touch	4.3”, 800×480, Touch
Standby/Talk Time	Standby/Talk Time	Standby/Talk Time	Battery Capacity	Battery Capacity
#Names, #SMS	#Names, #SMS	Memory/Card Slot	Memory	Memory/Storage/Card Slot
	Polyphonic Ringtones	MP3 Ringtones/MP3 Player/Browser	Browser/Exchange	Browser/Exchange
		3MP Camera	3MP Camera	Front/Rear Camera/HD Video
			CPU, OS	CPU, GPU, OS

Web Data: A Multitude of Data Models and Sources



List by the United Nations (2010)[2]

Rank	Country/Region	2010 GDP (millions of US$)
	World	62,633,783
1	United States	14,447,100
2	China	5,739,358
3	Japan	5,458,873
4	Germany	3,280,334
5	France	2,559,850
6	United Kingdom	2,253,552
7	Brazil	2,088,966
8	Italy	2,051,290
9	India	1,722,328
10	Canada	1,577,040
11	Russia	1,479,823
12	Spain	1,407,322
13	Australia	1,271,945
14	Mexico	1,032,224
15	South Korea	1,014,369

List by the International Monetary Fund (2011)

Rank	Country/Region	GDP (millions of US$)
	World	69,899,225[5]
	European Union	17,610,826[5]
1	United States	15,075,675
2	China	7,298,147[n2]
3	Japan	5,866,540
4	Germany	3,607,364
5	France	2,778,085
6	Brazil	2,492,907
7	United Kingdom	2,431,310
8	Italy	2,198,732
9	Russia	1,850,401
10	India	1,826,811
11	Canada	1,738,954
12	Australia	1,486,914
13	Spain	1,479,560
14	Mexico	1,153,958

List by the World Bank (1990–2011)[4]

Rank	Country/Region	GDP (millions of US$)	Year
	World	69,983,693[n12]	2011
	European Union	17,552,216[n12]	2011
1	United States	15,094,000	2011
2	China	7,318,499[n2]	2011
3	Japan	5,867,154	2011
4	Germany	3,570,556	2011
5	France	2,773,032[n4]	2011
6	Brazil	2,476,652	2011
7	United Kingdom	2,431,589	2011
8	Italy	2,194,750	2011
9	Russia	1,857,770	2011
10	India	1,847,982	2011
11	Canada	1,736,051	2011
12	Spain	1,490,810	2011
13	Australia	1,371,764	2011
14	Mexico	1,155,316	2011

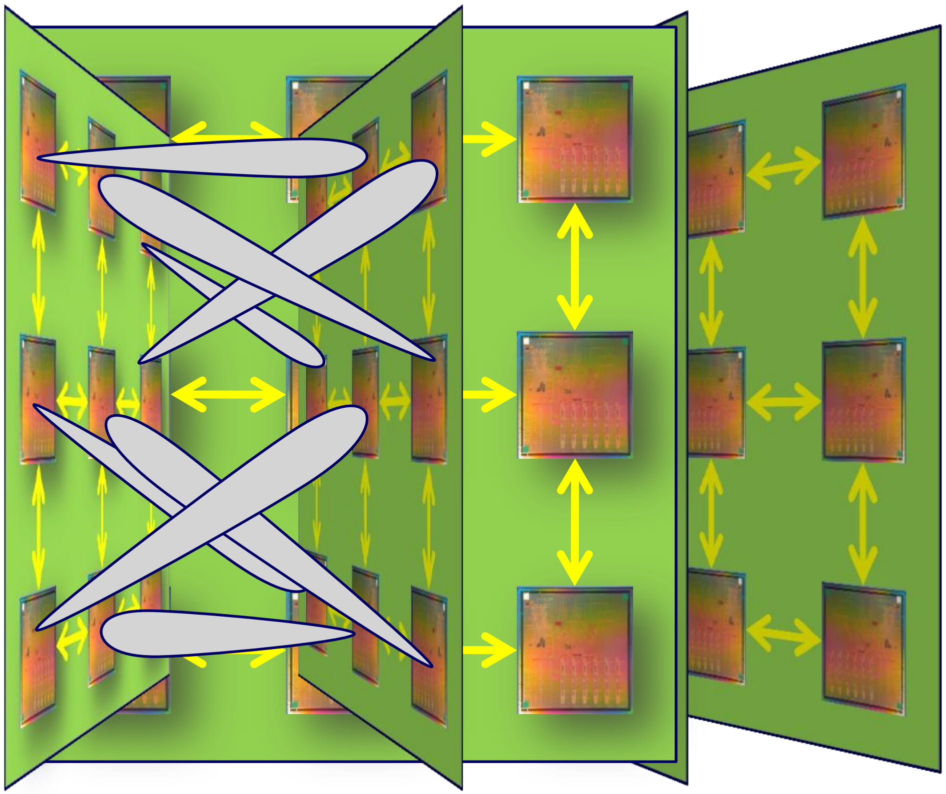
- Example 1: Queries spanning millions of data sources with as many schemata
- Example 2: Augment local datasets with (external) third-party information

select … from cust,
where nations.**GDP** > $1B

> Example 3: Energy-Adaptive High-Speed Computing Platform

Deutsche Forschungsgemeinschaft DFG

SFB912: HAEC – Highly Adaptive Energy Efficient Computing



HAEC

Optical Interconnect

- adaptive analog/digital circuits for e/o transceiver
- embedded polymer waveguide
- packaging technologies (e.g. 3D stacking of Si/III-V hybrids)
- 90° coupling of laser

Radio Interconnect

- on-chip/on-package antenna arrays
- analog/digital beamsteering and interference minimization
- 100Gb/s
- 100-300GHz channel
- 3D routing & flow management

3 Composed Operators

- Construction of composed operators, like multi-way-select-join-group-sort

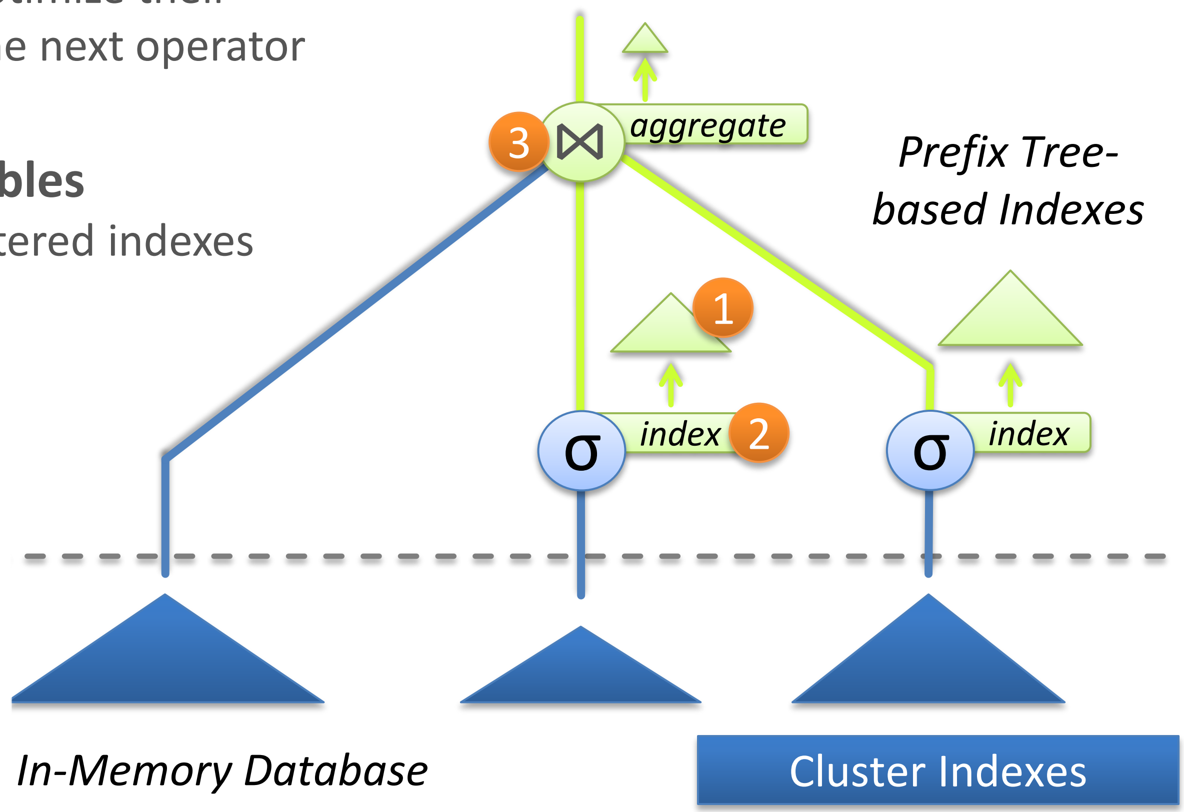
→ Avoids Intermediate Materialization

2 Cooperative Operators

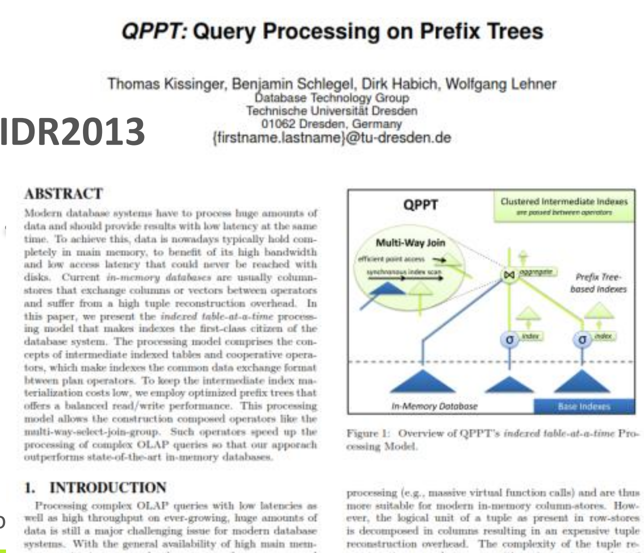
- Operators optimize their output for the next operator

1 Intermediate Indexed Tables

- Operators exchange clustered indexes



CIDR2013



***QPPT:* Query Processing on Prefix Trees**

Thomas Kissinger, Benjamin Schlegel, Dirk Habich, Wolfgang Lehner
Database Technology Group
Technische Universität Dresden
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

ABSTRACT

Modern database systems have to process huge amounts of data and should provide results with low latency at the same time. To achieve this, data is nowadays typically hold completely in main memory, to benefit of its high bandwidth and low access latency that could never be reached with disks. Current *in-memory databases* are usually column-stores that exchange columns or vectors between operators and suffer from a high tuple reconstruction overhead. In this paper, we present the *indexed table-at-a-time* processing model that makes indexes the first-class citizen of the database system. The processing model comprises the concepts of intermediate indexed tables and cooperative operators, which make indexes the common data exchange format btween plan operators. To keep the intermediate index materialization costs low, we employ optimized prefix trees that offers a balanced read/write performance. This processing model allows the construction composed operators like the multi-way-select-join-group. Such operators speed up the processing of complex OLAP queries so that our apporach outperforms state-of-the-art in-memory databases.

1. INTRODUCTION

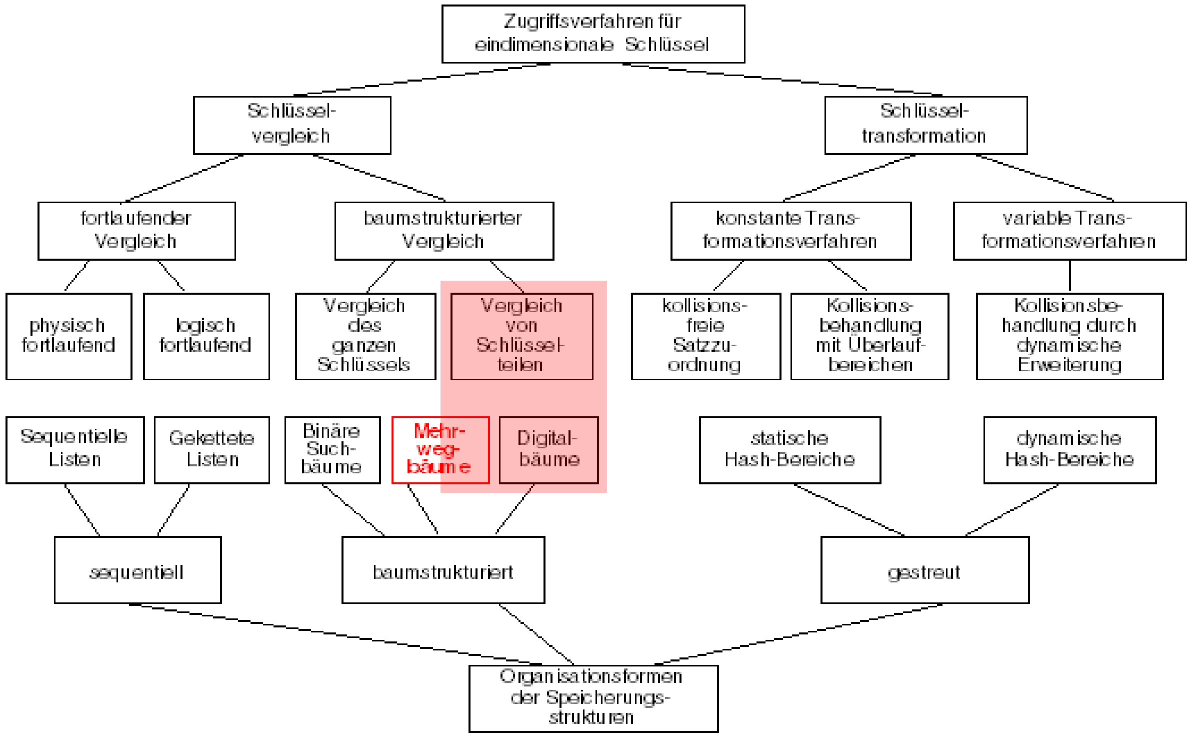
Processing complex OLAP queries with low latencies as well as high throughput on ever-growing, huge amounts of data is still a major challenging issue for modern database systems. With the general availability of high main memory capacity, in-memory database systems become more and

QPPT
Clustered Intermediate Indexes
are passed between operators
Multi-Way Join
efficient point access
synchronous index scan
aggregate
Prefix Tree-based Indexes
index
index
In-Memory Database
Base Indexes

Figure 1: Overview of QPPT's *indexed table-at-a-time* Processing Model.

processing (e.g., massive virtual function calls) and are thus more suitable for modern in-memory column-stores. However, the logical unit of a tuple as present in row-stores is decomposed in columns resulting in an expensive tuple reconstruction overhead. The complexity of the tuple reconstruction procedure grows with an increasing number of

> Prefix Trees

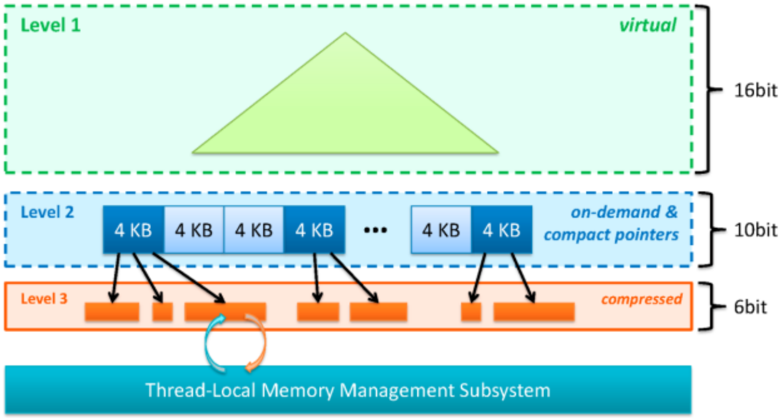


Prefix Trees	B+-Trees
▪ Unbalanced	▪ Balanced
✚ Balanced read/write performance	– Low update performance
✚ Main memory-optimized	– Disk-optimized
✚ Efficient parallel access	– Low scalability

> Index Landscape

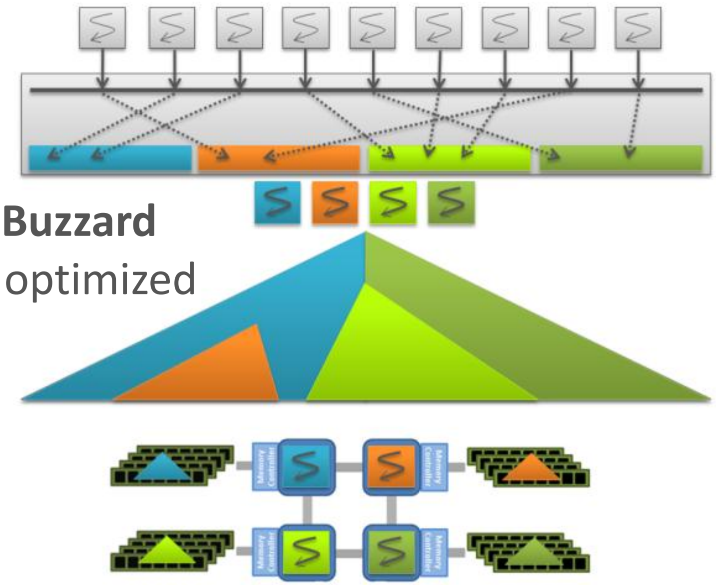
KISS-Tree

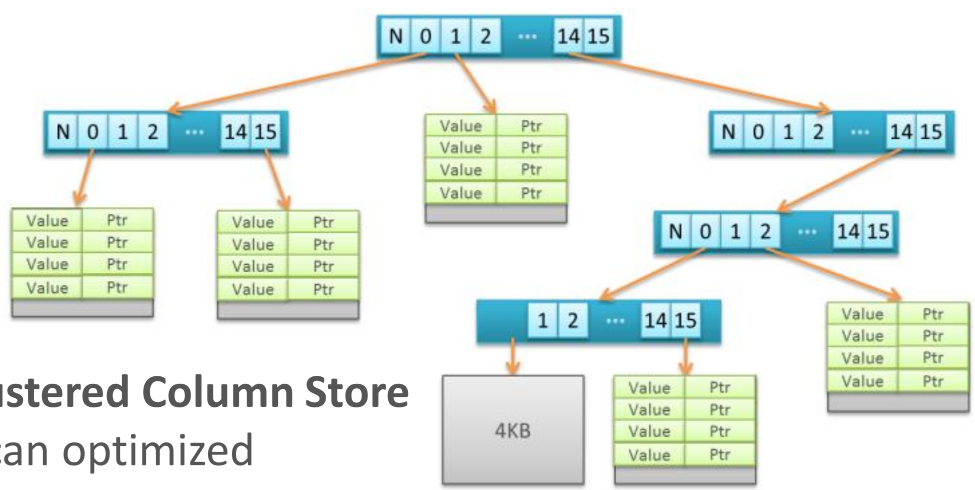
→ 32bit key index



Buzzard

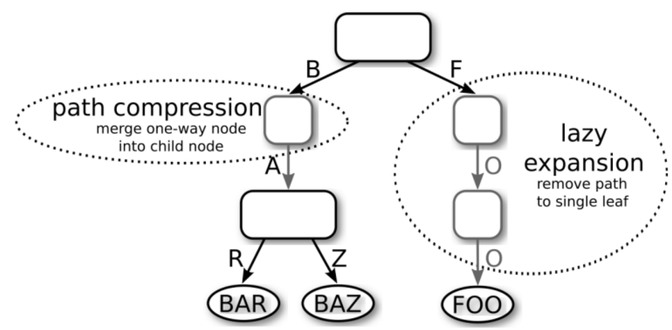
→ NUMA optimized





Clustered Column Store

→ Scan optimized



Adaptive Radix Tree (ART)

> KISS-Tree Overview

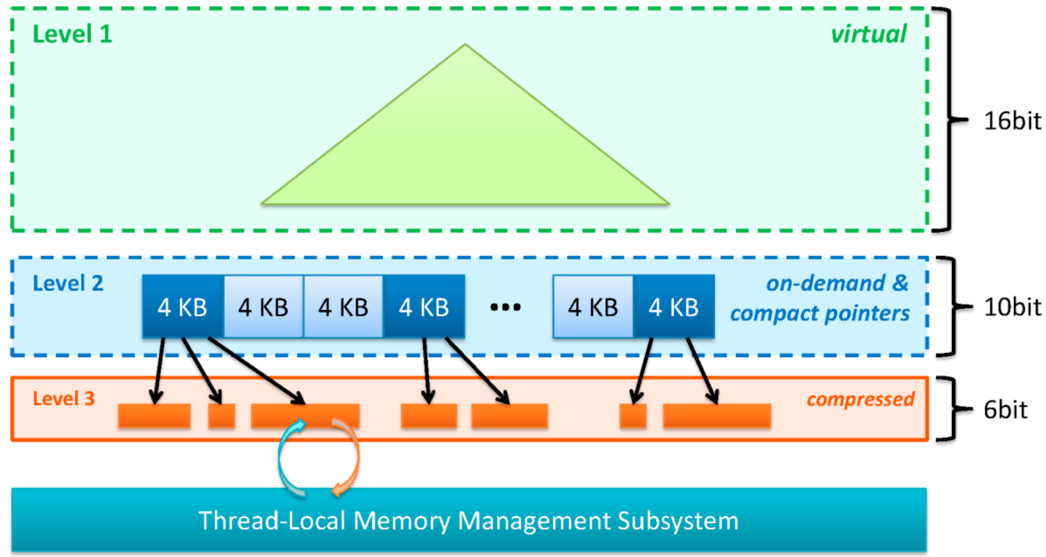
Database Technology Group

Properties

- Specialized version for 32bit keys
- **Latch-free updates**
- **Order-preserving**
- **2-3 memory accesses per key**

→ Comparable fast to reported order-preserving in-memory indexes for read access

→ BUT:
High update performance



- Heterogeneous in-memory index structure
- Combination of direct and indirect addressing
- Takes advantage of virtual memory management
- Enables different compression mechanisms

DAEMON 2012

***KISS-Tree:* Smart Latch-Free In-Memory Indexing on Modern Architectures**

Thomas Kissinger, Benjamin Schlegel, Dirk Habich, Wolfgang Lehner
Database Technology Group
Technische Universität Dresden
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

ABSTRACT

Growing main memory capacities and an increasing number of hardware threads in modern server systems led to fundamental changes in database architectures. Most importantly, query processing is nowadays performed on data that is often completely stored in main memory. Despite of a high main memory scan performance, index structures are still important components, but they have to be designed from scratch to cope with the specific characteristics of main memory and to exploit the high degree of parallelism. Current research mainly focused on adapting block-optimized B+-Trees, but these data structures were designed for secondary memory and involve comprehensive structural maintenance for updates.

In this paper, we present the *KISS-Tree*, a latch-free in-memory index that is optimized for a minimum number of memory accesses and a high number of concurrent updates. More specifically, we aim for the same performance as modern hash-based algorithms but keeping the order-preserving nature of trees. We achieve this by using a prefix tree that incorporates virtual memory management functionality and compression schemes. In our experiments, we evaluate the *KISS-Tree* on different workloads and hardware platforms and compare the results to existing in-memory indexes. The *KISS-Tree* offers the highest reported read performance on current architectures, a balanced read/write performance, and has a low memory footprint.

1. INTRODUCTION

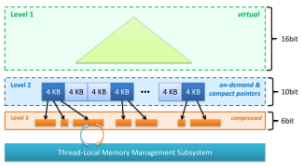


Figure 1: KISS-Tree Overview.

indexes) in-memory and use secondary memory (e.g., disks) only for persistence. While disk block accesses constituted the bottleneck for disk-based systems, modern in-memory databases shift the memory hierarchy closer to the CPU and face the "memory wall" [13] as new bottleneck. Thus, they have to care for CPU data caches, TLBs, main memory accesses and access patterns. This essential change also affects indexes and forces us to design new index structures that are now optimized for these new design goals. Moreover, the movement from disks to main memory dramatically increased data access bandwidth and reduced latency. In combination with the increasing number of cores on modern hardware, parallel processing of operations on index structures imposes a new challenges for us, because the overhead

> Level 1: Virtual Level

Decimal Key: 327914

0000 0000 0000 0101 | 0000 0000 11 | 10 1010

Split key in 3 fragments

16bit (f_1) 10bit (f_2) 6bit (f_3)

5

- Calculate corresponding 2nd node address from the first fragment

→Direct addressing

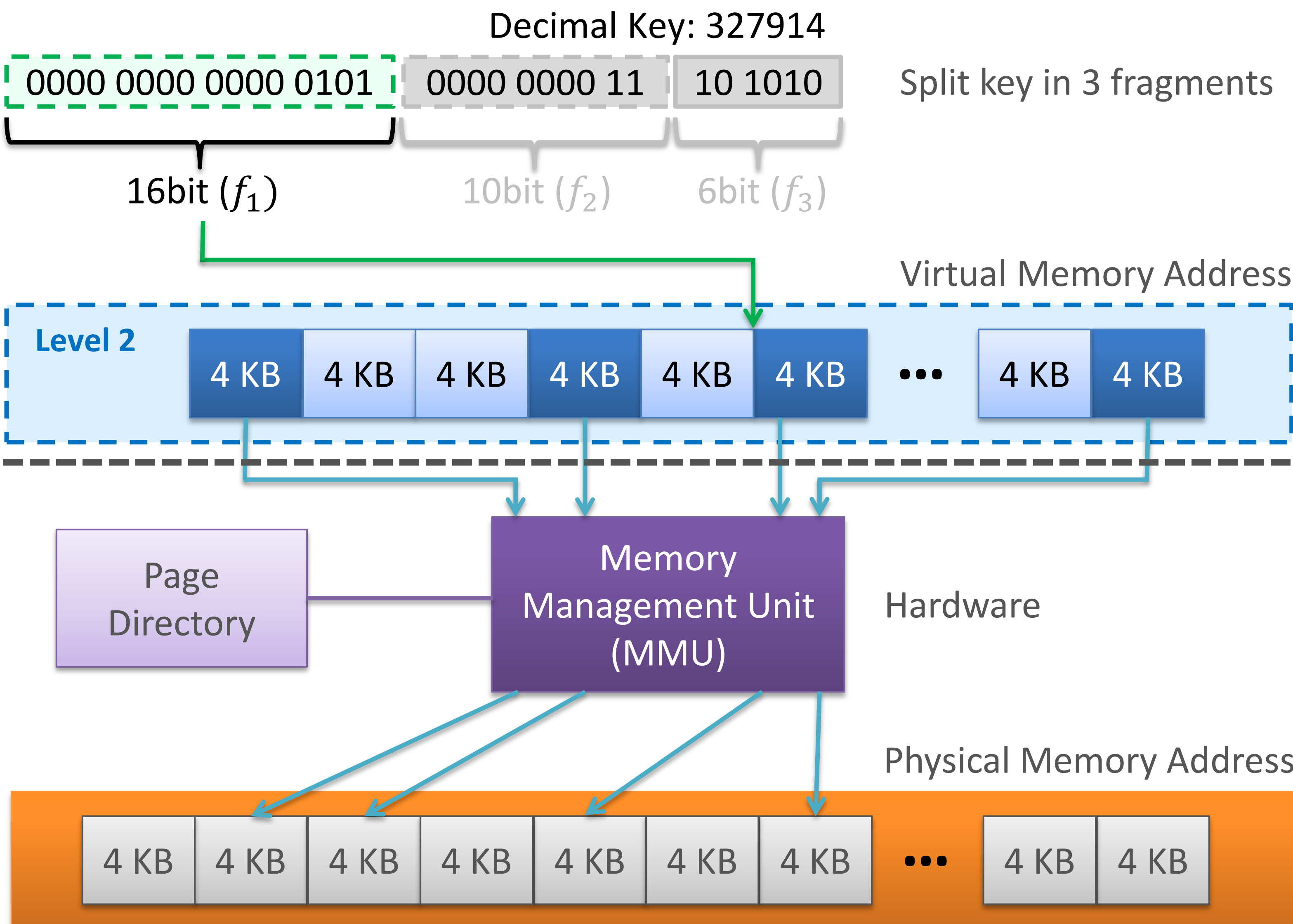
→No memory required for level 1

Level 2

Node Node Node Node Node Node ••• Node Node

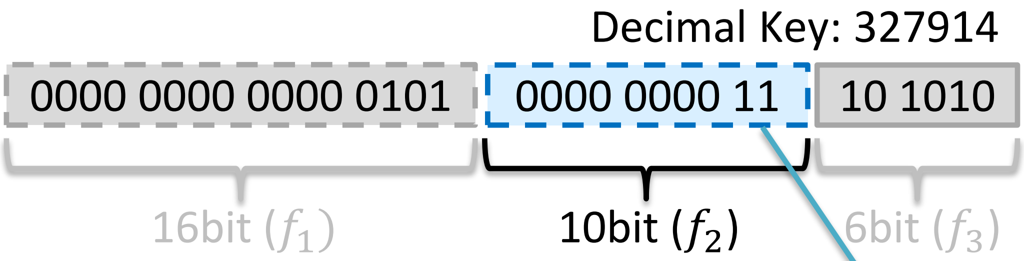
Requirement for Level 2: All nodes have to be stored sequentially in memory

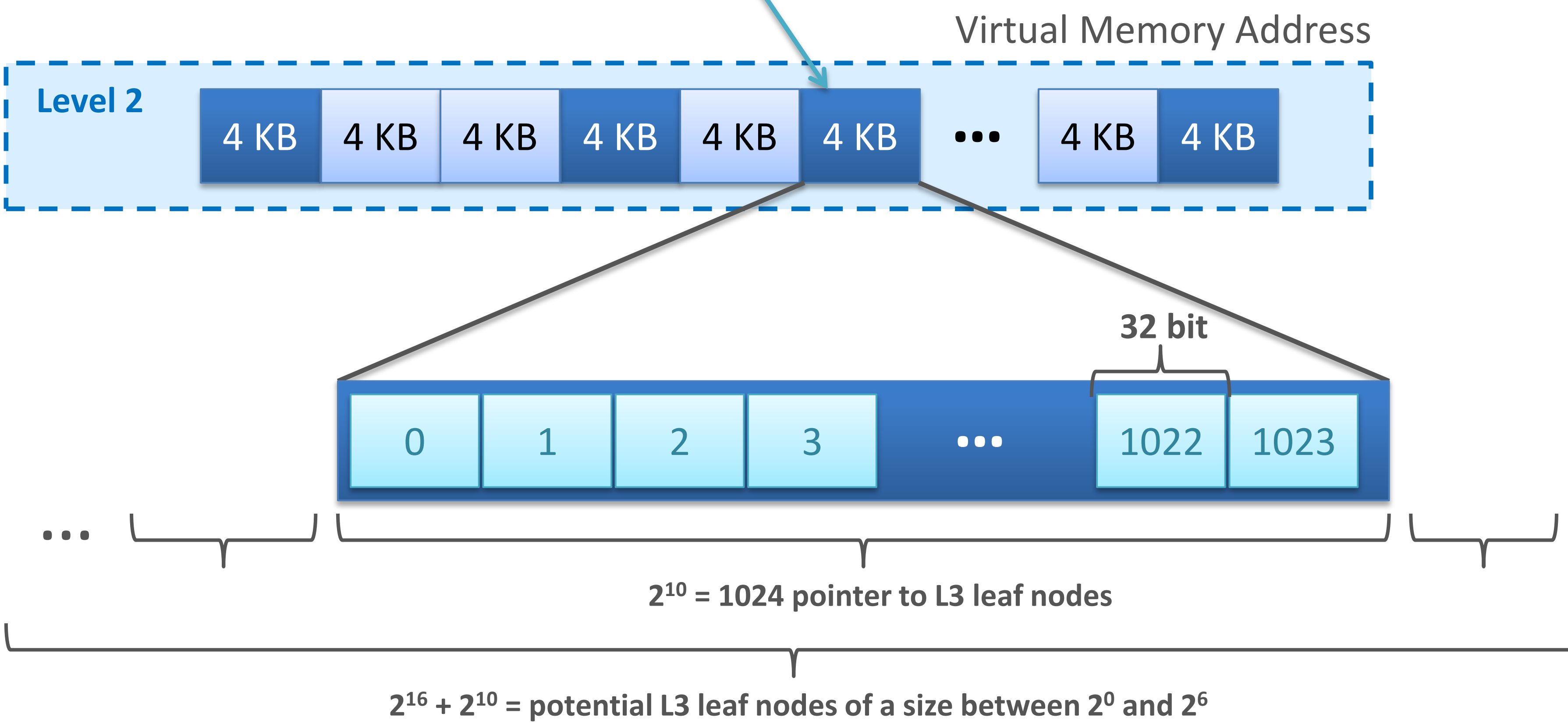
> Level 1: Virtual Level



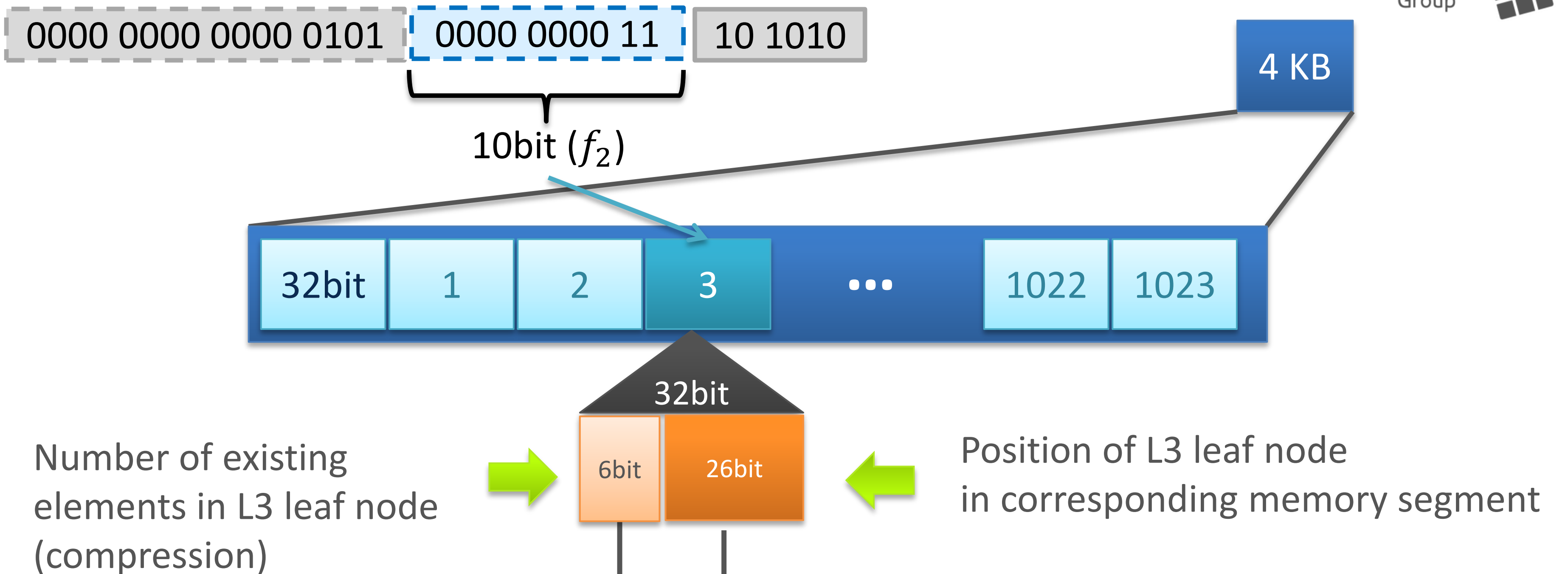
> Level 2: On-demand Level

- Change over to indirect addressing
- 1024 buckets per node containing a compact pointer to the 3rd level node
- 256 MB maximum

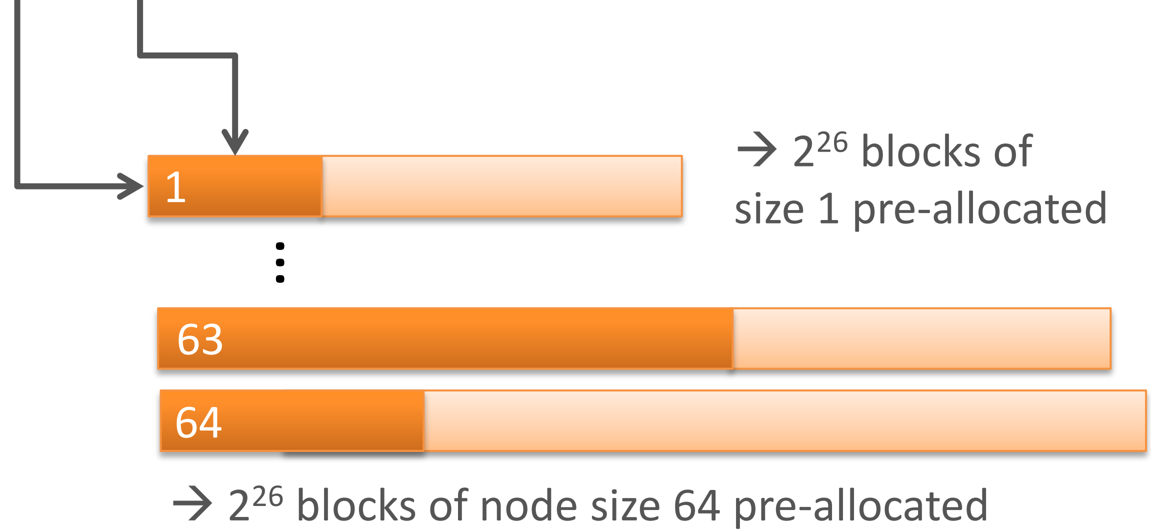




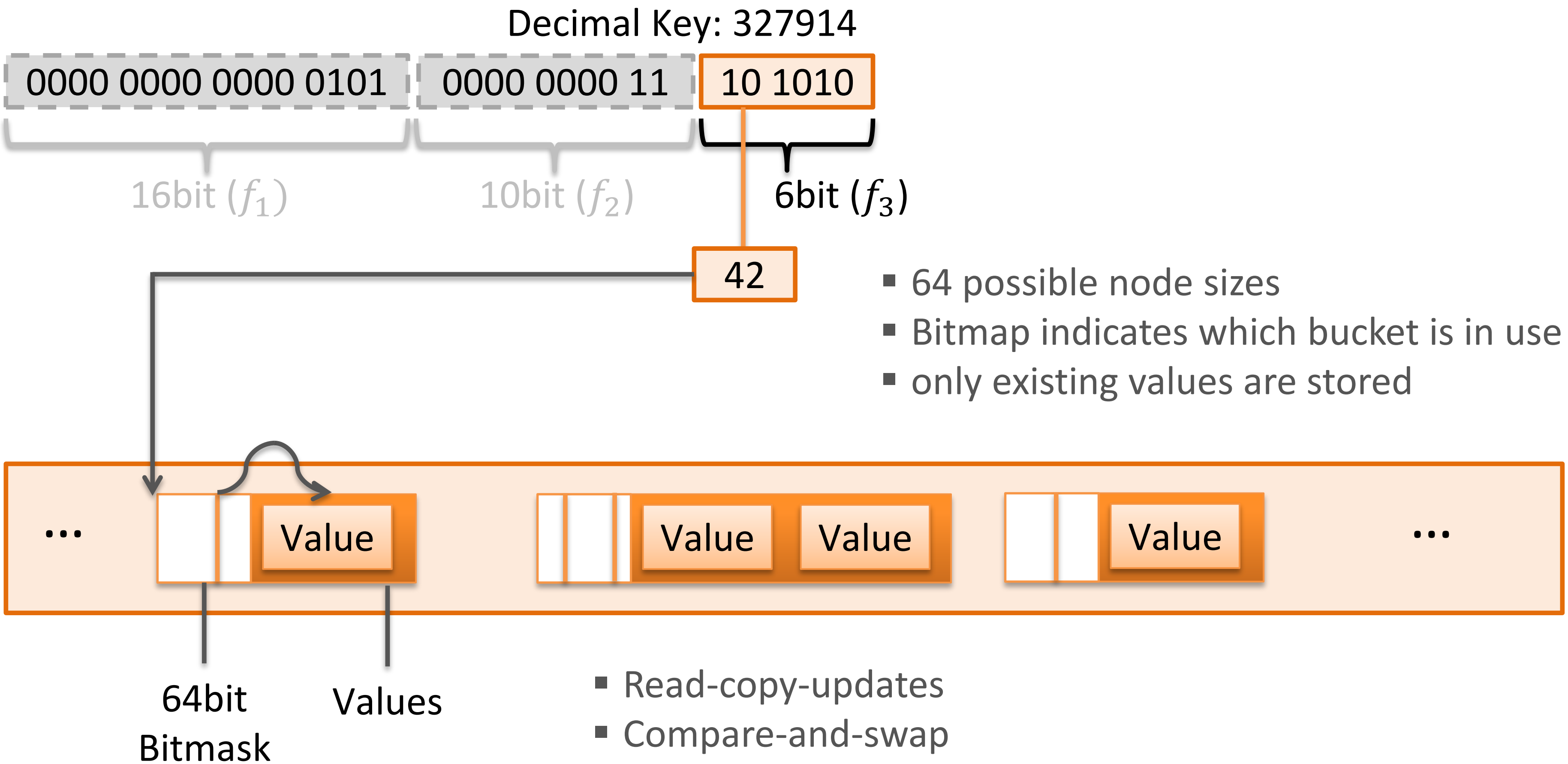
> Level 2: On-demand Level



- Allocation of consecutive virtual memory segments for each of the 64 node sizes possible on the third level
- Each segment consists of a maximum of 2^{26} blocks of the respective node size



> Level 3: Compression



- 64 possible node sizes
- Bitmap indicates which bucket is in use
- only existing values are stored

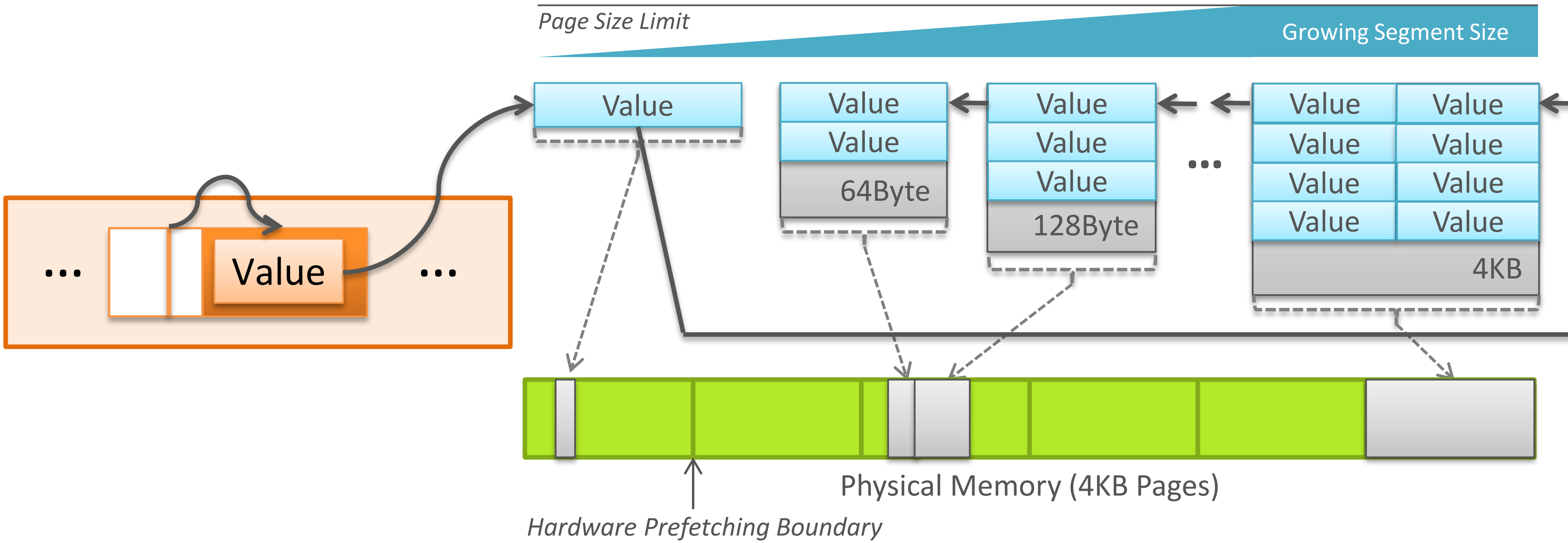
- Read-copy-updates
- Compare-and-swap

→ No in-place updates possible (lost updates)

Thread-Local Memory Management Subsystem

> Duplicate Handling

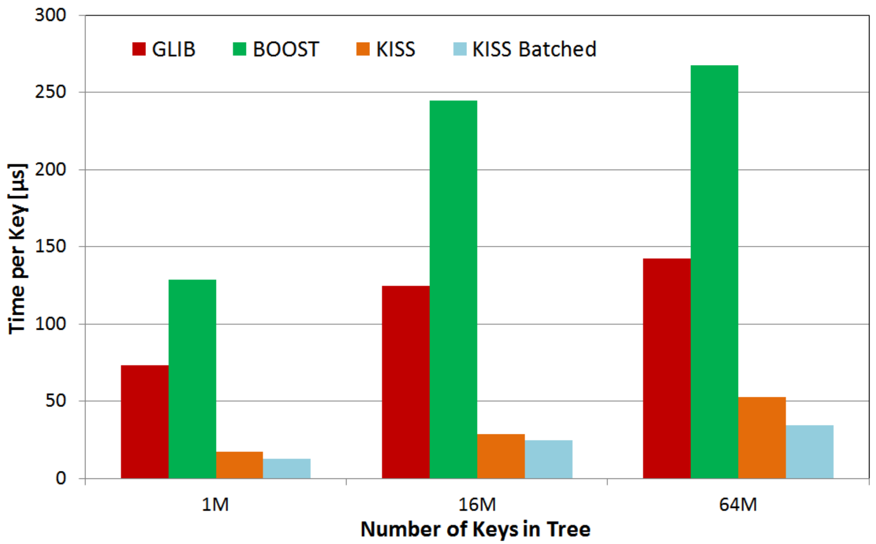
- Efficient duplicate handling necessary for query processing
  - Scanning a linked list results in random memory accesses



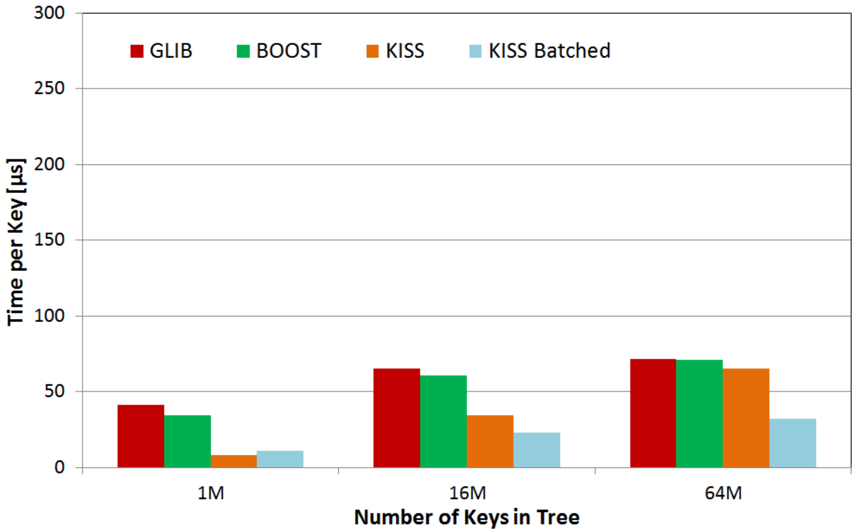
- Page boundaries are a barrier for hardware prefetchers
  - store values sequentially in 4KB blocks
  - blocks grow exponentially until reaching 4KB
    - → trade-off between scan performance and memory consumption

> Prefix Tree Performance

Insert/Update Performance



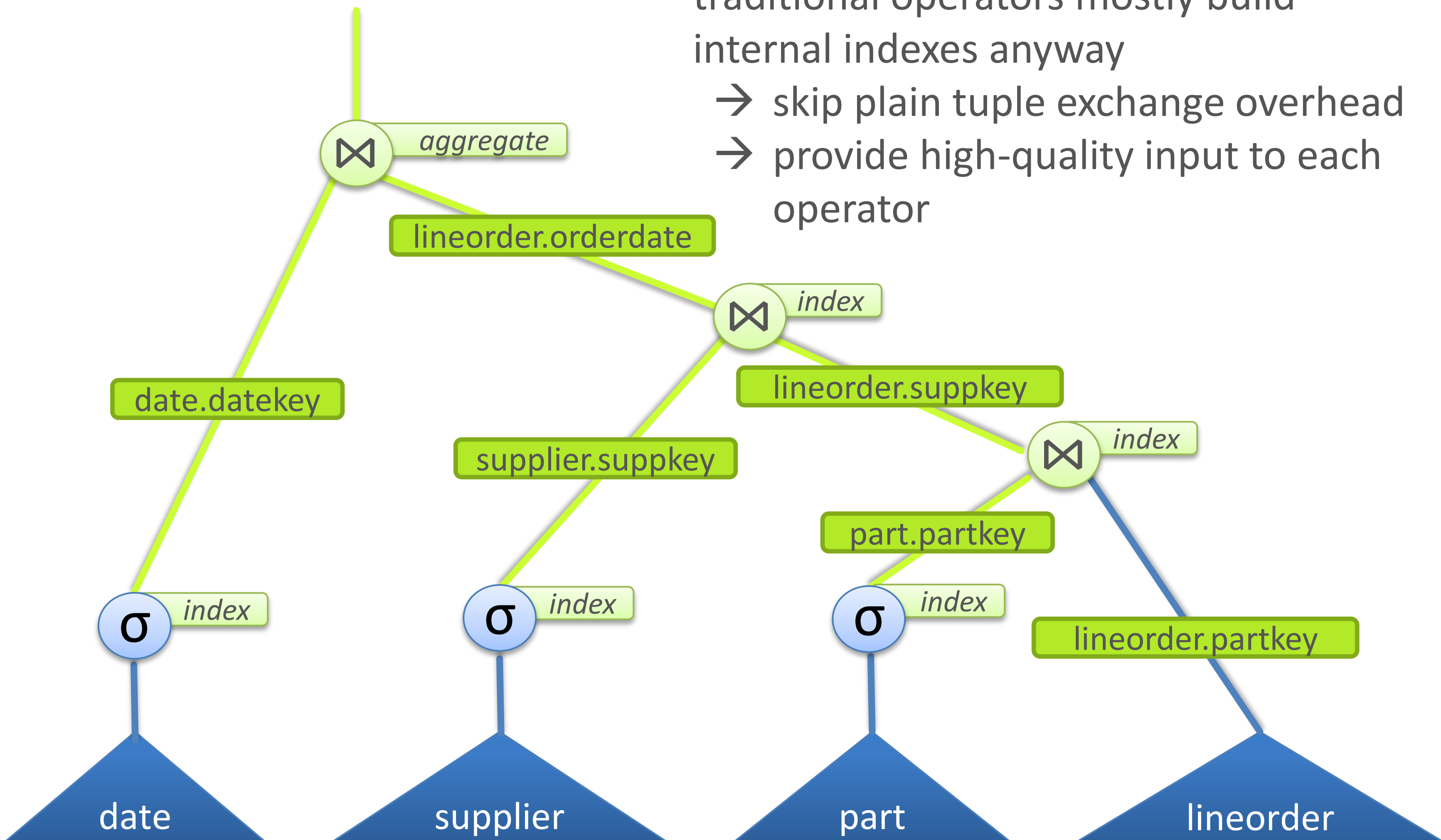
Lookup Performance



- Performance depends on the number of memory accesses per key
  - KISS-Tree: 3 memory accesses per key
  - Latency Hiding with Batch Updates/Lookups
- KISS-Tree performs better than hash table implementations

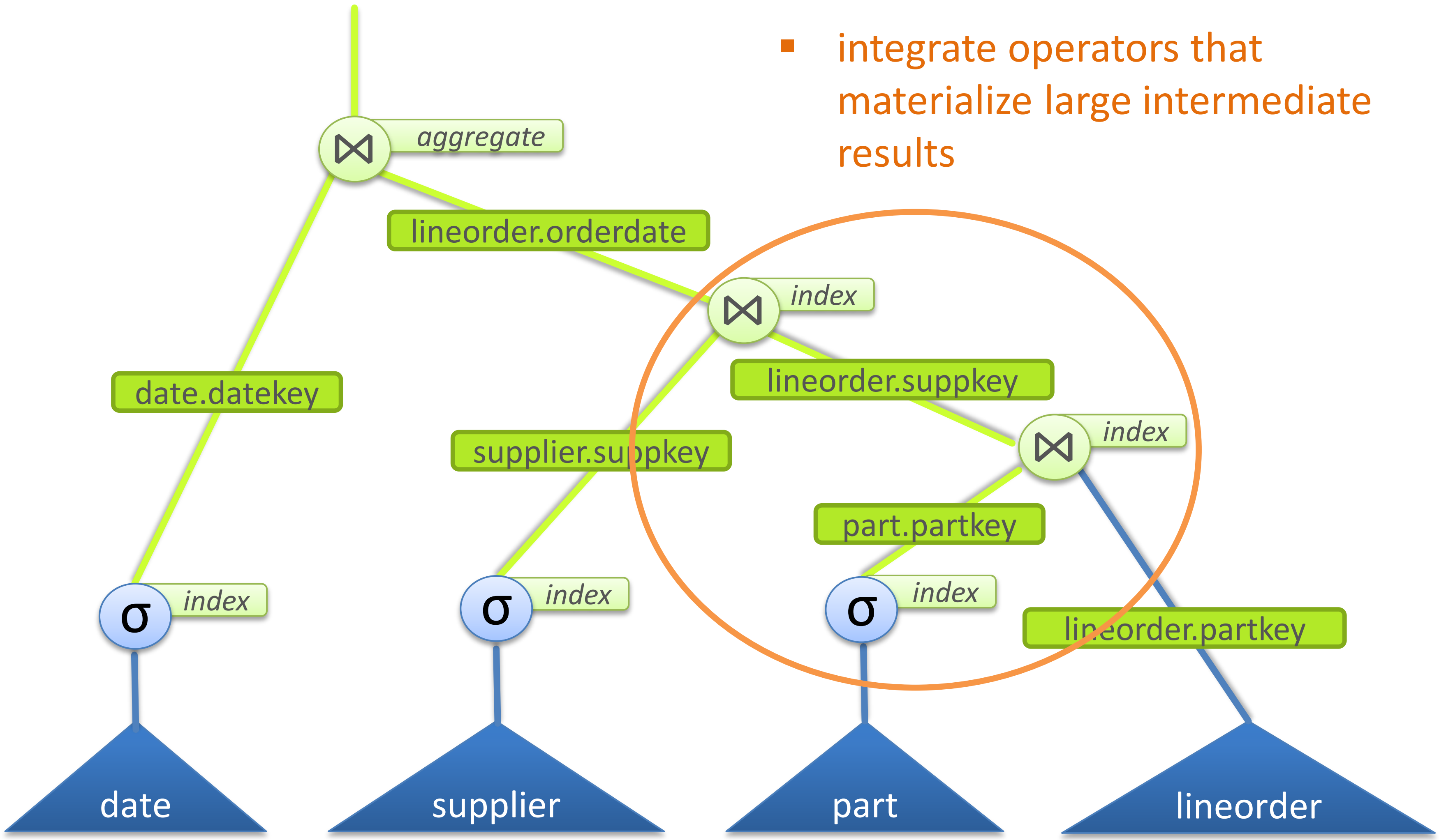
> Cooperative Operators

- Each operator adjusts its output to the requirements of the successive operator
  - traditional operators mostly build internal indexes anyway
    → skip plain tuple exchange overhead
    → provide high-quality input to each operator



> Composed Operators (1)

- Materialization is costly; thus try to completly avoid it!



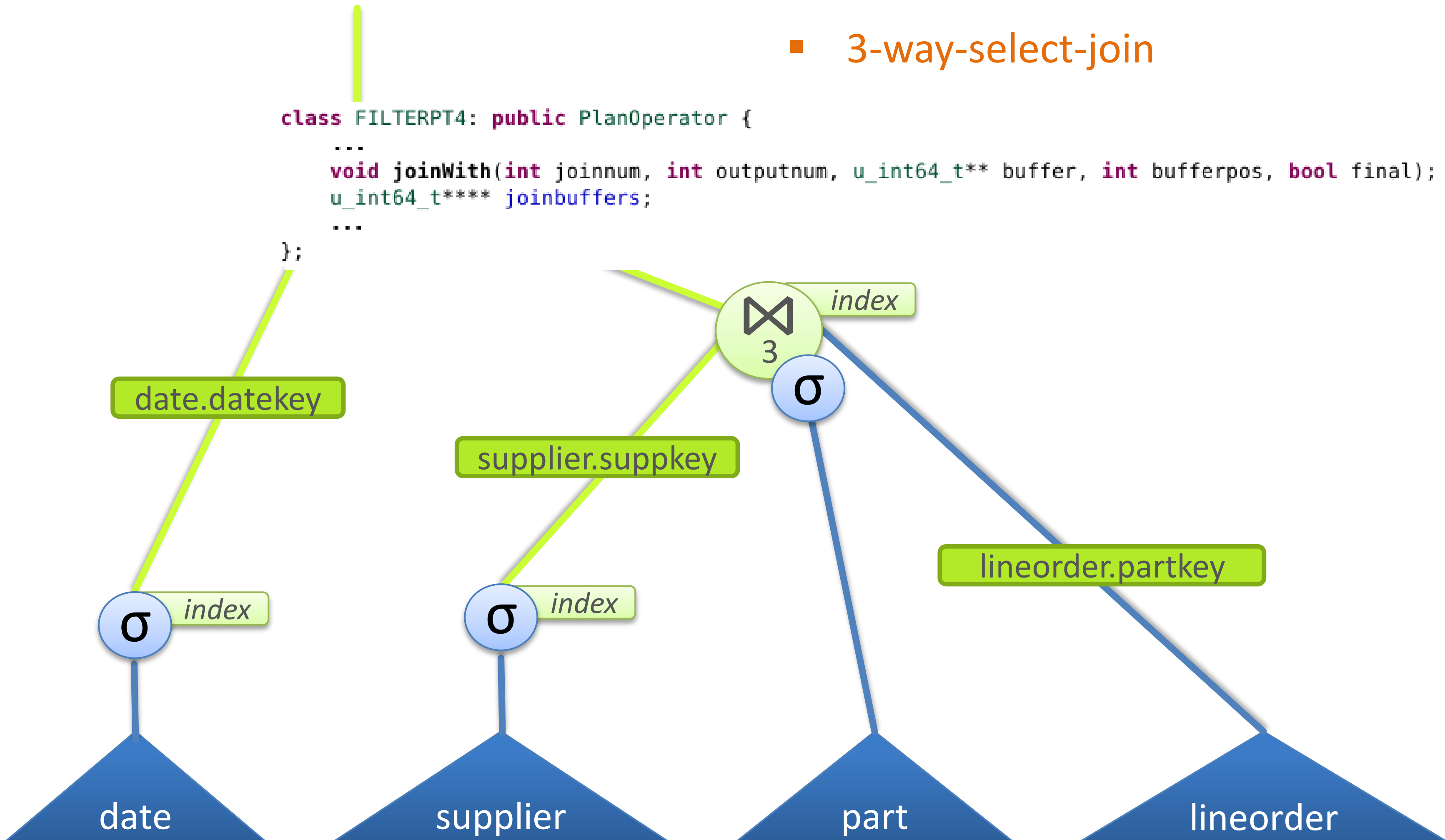
- integrate operators that materialize large intermediate results

> Composed Operators (2)

- Materialization is costly; thus try to completly avoid it!

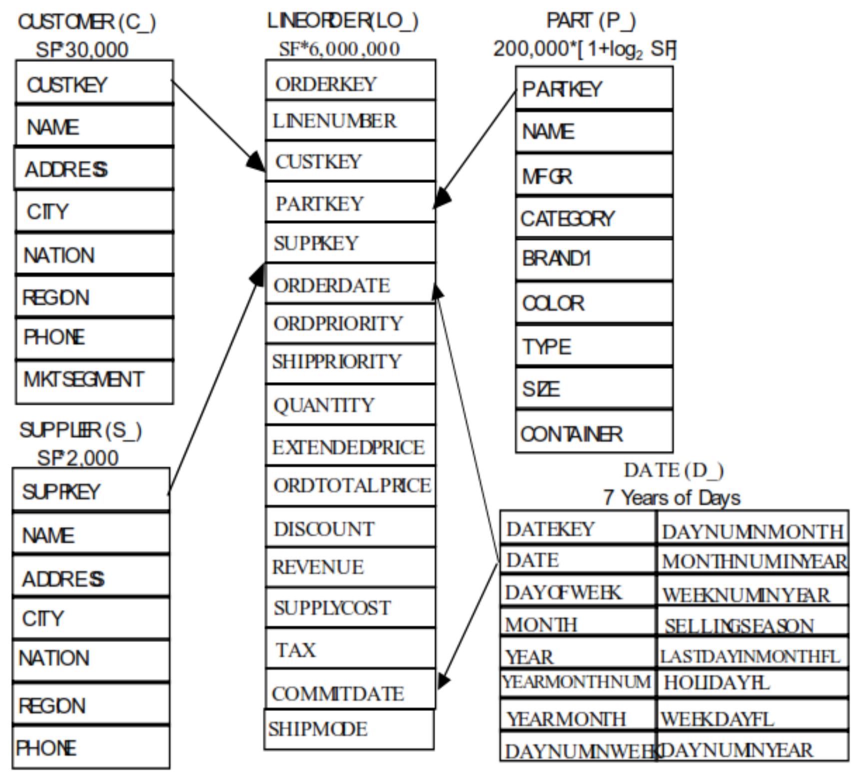
- 3-way-select-join

```
class FILTERPT4: public PlanOperator {
     ...
     void joinWith(int joinnum, int outputnum, u_int64_t** buffer, int bufferpos, bool final);
     u_int64_t**** joinbuffers;
     ...
};
```

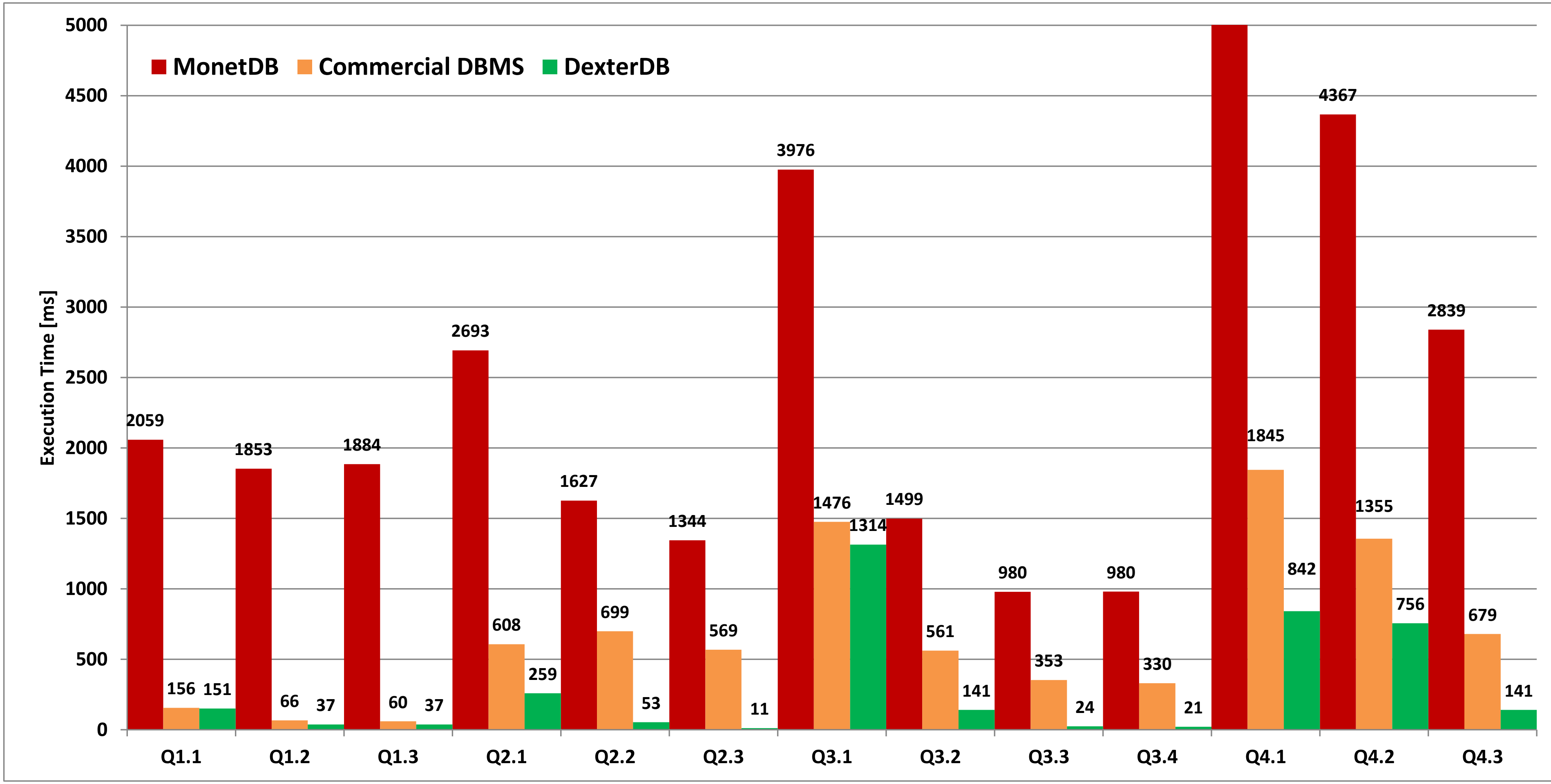


> Evaluation (1)

- Star Schema Benchmark (SSB)
  - Scale Factor = 15

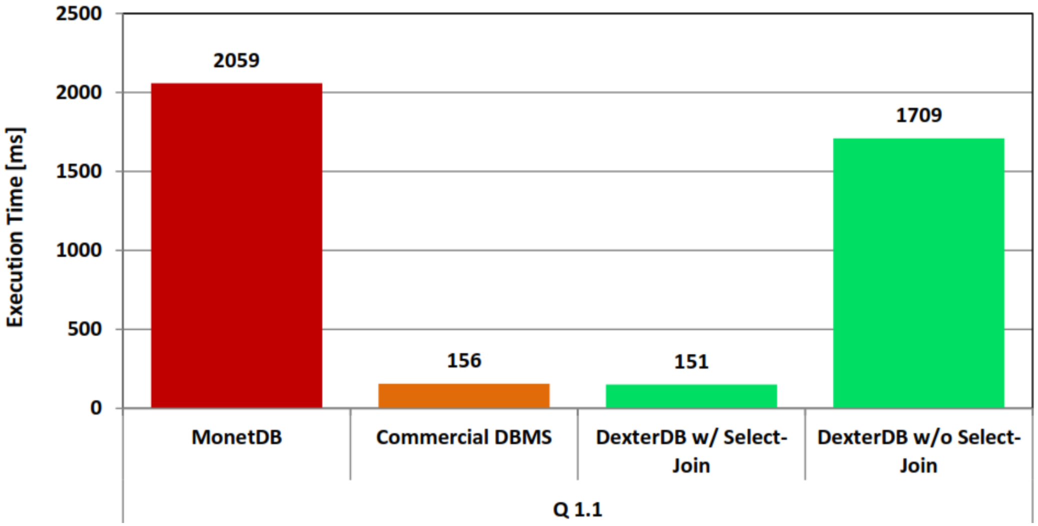


- Intel i7-3960X
  - 15 MB LLC
  - 3.3 GHz (Turbo Mode disabled)
- 32 GB Main Memory
- Ubuntu 12.04
- Single-threaded execution



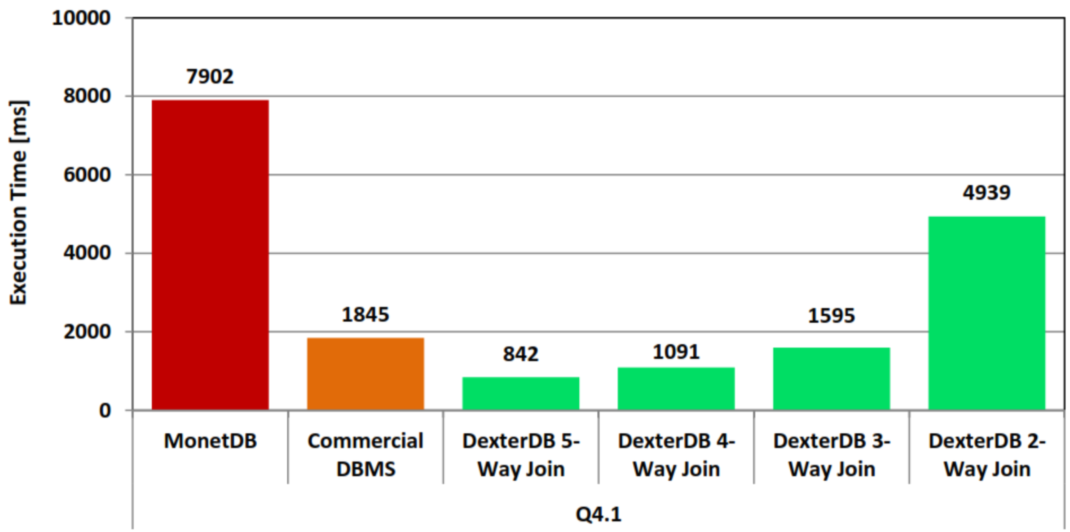
(*) : we are in touch with MonetDB; performance numbers are under investigation

> Evaluation (3)



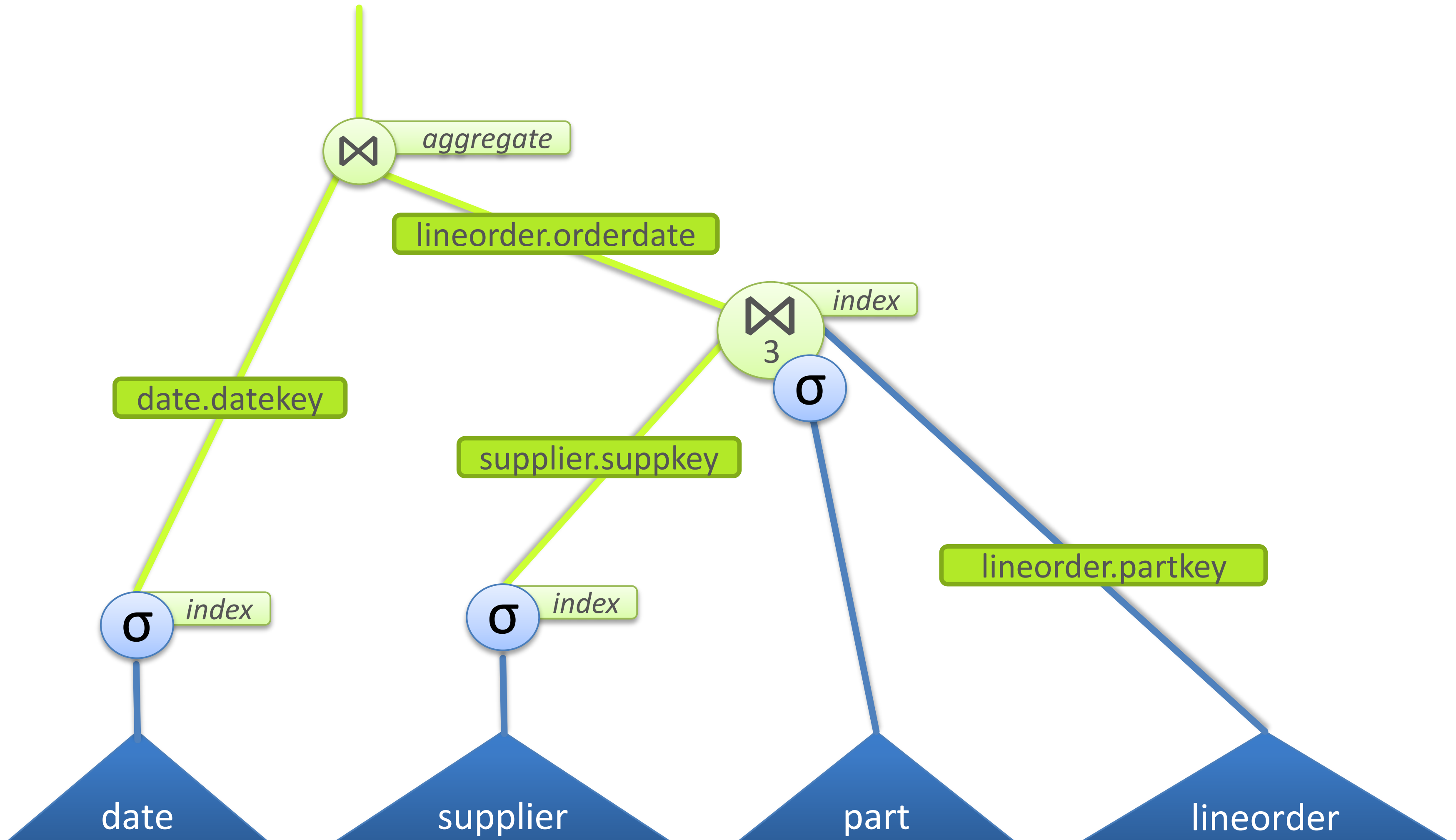
- SSB Query 1.1
- Single Join
- Select-Join operator avoids intermediate result materialization
- Batched KISS-Tree Lookups/Inserts

- SSB Query 4.1
- 4 Joins over 5 Tables
- Multi-way join operator avoids intermediate result materialization

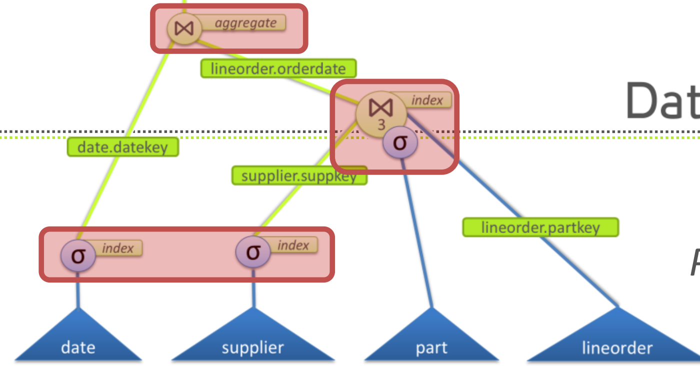


> Adaptive QPPT (1)

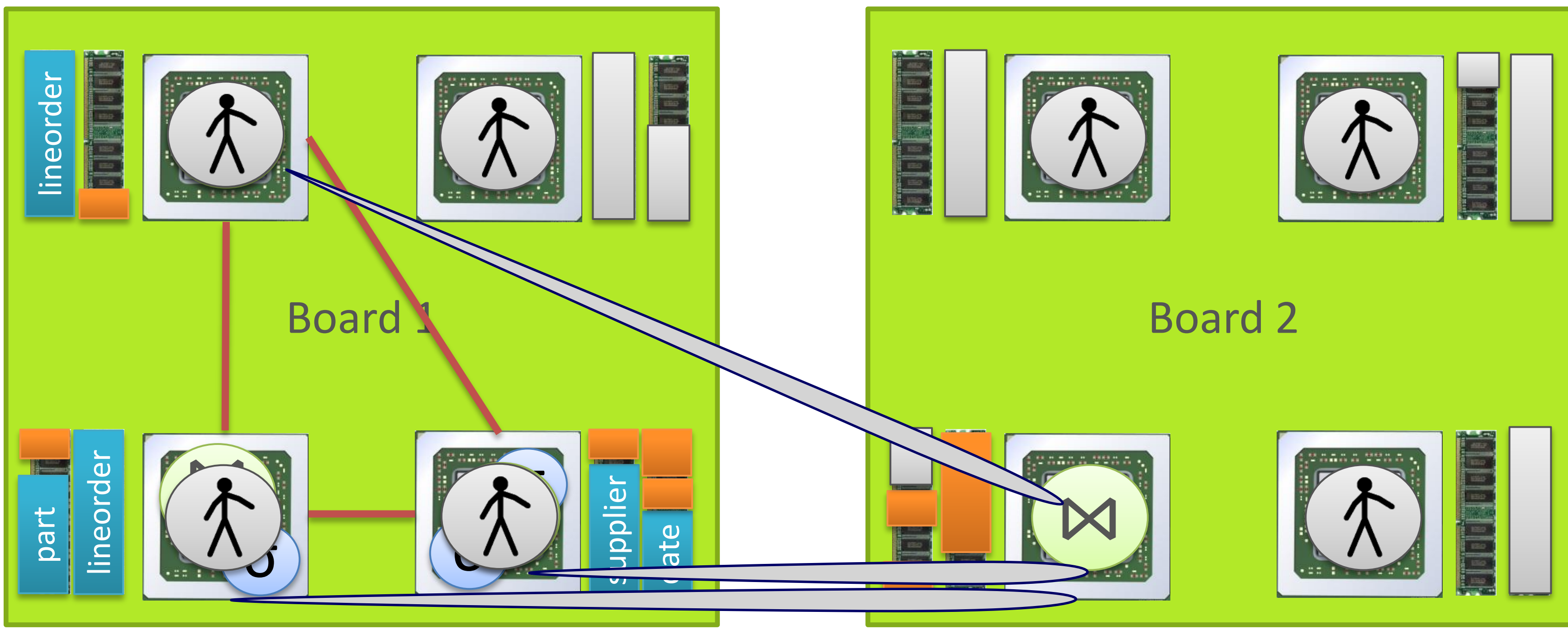
- Flexible Query Execution on Adaptive Hardware



> Adaptive QPPT (2)

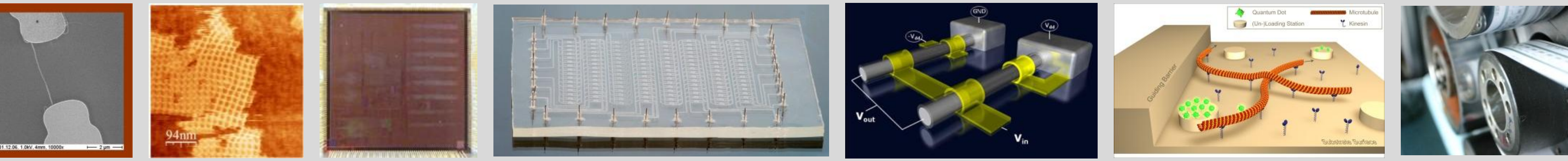


Parallel Running Query

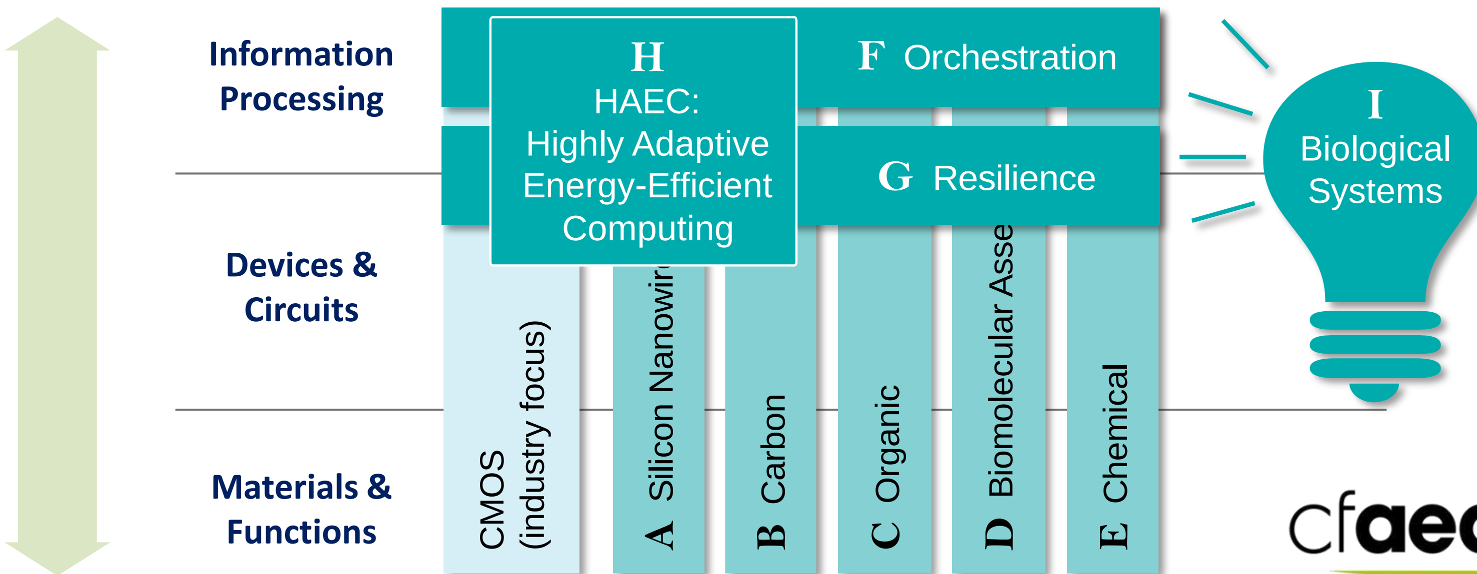


- reduce hops by wireless „point-to-point" interconnects
- operator → data / data → operator shipping
- adaptive hardware components (e.g., trade cache for compute units)

> Comprehensive Approach: 9 Paths



„More-Shots-on-Goal“



cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Database Technology Group

TECHNISCHE UNIVERSITÄT DRESDEN

Adaptive Query Processing on Prefix Trees

Wolfgang Lehner

Fachgruppentreffen, 22.11.2012 – TU München