Dresden Database
Systems Group

# Next-Gen Hardware for Data Management

## More a Blessing than a Curse?

Wolfgang Lehner – EDBT 2015, Brussels

# Modern Hardware – All over the place…

Dresden Database Systems Group

**Transaction Logging Unleashed with NVRAM***

Tianzheng Wang Ryan Johnson
University of Toronto
{tzwang, ryan.johnson}@cs.toronto.edu

**Utilization Wall: Dark Silicon's Effect on Multicore Scaling**

Spectrum of tradeoffs between # of cores and frequency

Example:
65 nm → 32 nm (S = 2)

2x4 cores @ 1.8 GHz
(8 cores dark, 8 dim)

4 cores @ 1.8 GHz

65 nm

**SGI Scales Up HANA On UV NUMA Syste**

June 3, 2014 by Timothy Prickett Morgan

In-memory processing is the hot new thing in the enterprise d
with decades of experience building shared memory supercc
leverage that expertise and catch the in-memory wave. For t
the company has been designing its next-generation NUMA
the UV Gen3 systems that will make use of it, and is now pre
Box" implementation of that future machine, tuned up specific

**Data-Oriented Transaction Exe**

Ippokratis Pandis[1,2] Ryan Johnson[1,2] Nikos Hardavell
ipandis@ece.cmu.edu ryanjohn@ece.cmu.edu nikos@northweste

[1]Carnegie Mellon University
Pittsburgh, PA, USA

[2]École Polytechnique Fédérale de Lausanne
Lausanne, VD, Switzerland

**ABSTRACT**

While hardware technology has undergone major advancements over the past decade, transaction processing systems have remained largely unchanged. The number of cores on a chip grows exponentially, following Moore's Law, allowing for an ever-increasing number of transactions to execute in parallel. As the number of concurrently-executing transactions increases, contended critical sections become scalability burdens. In typical transaction processing systems the centra
often the first contended component and sc

In this paper, we identify the c
ansaction assignment policy as the prima
hen, we design DORA, a system t
ansaction to smaller actions and assigns
n which data each action is about to a
llows each thread to mostly access threa
inimizing interaction with the contention
anager. Built on top of a conventional
aintains all the ACID properties. Eval
mplementation of DORA on a multicore sy
ORA attains up to 4.8x higher throughpu
orage engine when running a variety of s
LTP workloads.

**Fast Updates on Read-Optimized Databases Using Multi-Core CPUs**

Jens Krueger†, Changkyu Kim*, Martin Grund†, Nadathur Satish*, David Schwalb†, Jatin Chhugani*, Hasso Plattner†, Pradeep Dubey*, Alexander Zeier†

†Hasso-Plattner-Institute, Potsdam, Germany
Contact: jens.krueger@hpi.uni-potsdam.de

*Parallel Computing Lab, Intel Corporation
Contact: changkyu.kim@intel.com

**ABSTRACT**

Read-optimized columnar databases use differential updates to handle writes by maintaining a separate write-optimized delta partition which is periodically merged with the read-optimized and compressed main partition. This merge process introduces significant overheads and unacceptable downtimes in update intensive systems, aspiring to combine transactional and analytical workloads into one system.

In the first part of the paper, we report data analyses of 12 SAP Business Suite customer systems. In the second half, we present an optimized merge process reducing the merge overhead of current systems by a factor of 30. Our linear-time merge algorithm exploits the underlying high compute and bandwidth resources of modern multi-core CPUs with architecture-aware optimizations and efficient parallelization. This enables compressed in-memory column stores to handle the transactional update rate required by enterprise applications, while keeping properties of read-optimized databases for analytic-style queries.

partition. Inserting into the write-optimized structure can be p
formed fast if the size of the structure is kept small enough. As
additional benefit, this also ensures that the read performance d
not degrade significantly. However, keeping this size small imp
merging frequently, which increases the overhead of updates.

To the best of our knowledge we are not aware of any sophi
cated implementation and therefore compare against a naïve imp
mentation. Based on the result of analyzing 12 SAP Business S
customer systems, we found that current systems would merge
prox. 20 hours every month, while supporting a maximum of ~1,
updates per second (see Section 2 for more detail). In read-mos
scenarios this limitation is not a major problem since the wo
load can be stopped during reload, modifications are invisible u
applied in batch or performance degradation is acceptable. H
ever, when engineering a system for both transactional and ana
ical workloads as described in [22, 17, 13], it becomes essen
to reduce the merge overhead and to support the required sin
update rates for handling transactional workloads. Systems un
o cope with even longer times for merging or f

**Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores**

Xiangyao Yu
MIT CSAIL
yxy@csail.mit.edu

George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu

**ABSTRACT**

Computer architectures are moving towards an era dominated by many-core machines with dozens or even hundreds of cores on a single chip. This unprecedented level of on-chip parallelism introduces a new dimension to scalability that current database management systems (DBMSs) were not designed for. In particular, as the number of cores increases, the problem of concurrency control becomes extremely challenging. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will likely diminish the gains from increased core counts.

To better understand just how unprepared current DBMSs are for future CPU architectures, we performed an evaluation of concurrency control for on-line transaction processing (OLTP) workloads on many-core chips. We implemented seven concurrency control algorithms on a main-memory DBMS and using computer simula-

that instruction-level parallelism and single-threaded performance will give way to massive thread-level parallelism.

As Moore's law continues, the number of cores on a single chip is expected to keep growing exponentially. Soon we will have hundreds or perhaps a thousand cores on a single chip. The scalability of single-node, shared-memory DBMSs is even more important in the many-core era. But if the current DBMS technology does not adapt to this reality, all this computational power will be wasted on bottlenecks, and the extra cores will be rendered useless.

In this paper, we take a peek at this dire future and examine what happens with transaction processing at one thousand cores. Rather than looking at all possible scalability challenges, we limit our scope to concurrency control. With hundreds of threads running in parallel, the complexity of coordinating competing accesses to data will become a major bottleneck to scalability, and will likely dwin-

**LLAMA: A Cache/Storage Subsystem for Modern Hardware**

Justin Levandoski
Microsoft Research
One Microsoft Way
Redmond, WA 98052
justinle@microsoft.com

David Lomet
Microsoft Research
One Microsoft Way
Redmond, WA 98052
lomet@microsoft.com

Sudipta Sengupta
Microsoft Research
One Microsoft Way
Redmond, WA 98052
sudipta@microsoft.com

**ABSTRACT**

LLAMA is a subsystem designed for new hardware environments that supports an API for page-oriented access methods, providing both cache and storage management. Caching (CL) and storage (SL) layers use a common mapping table that separates a page's logical and physical location. CL supports data updates and management updates (e.g., for index re-organization) via latch-free compare-and-swap atomic state changes on its mapping table. SL uses the same mapping table to cope with page location changes produced by log structuring on every page flush. To demonstrate LLAMA's suitability, we tailored our latch-free Bw-tree implementation to use LLAMA. The Bw-tree is a B-tree style index. Layered on LLAMA, it has higher performance and scalability using real workloads compared with BerkeleyDB's B-tree, which is known for good performance.

## 1 INTRODUCTION

### 1.1 Modern Architectures

Modern computer platforms have changed sufficiently that it is

We believe there are fundamental problems posed by current hardware that impact all access methods: B-trees, hashing, multi-attribute, temporal, etc. Further, these problems can be solved with general mechanisms applicable to most access methods.

1. Good processor utilization and scaling with multi-core processors via latch-free techniques.
2. Good performance with multi-level cache based memory systems via delta updating that reduces cache invalidations.
3. Write limited storage in two senses: (1) limited performance of random writes; (2) flash write limits; via log structuring.

The Bw-tree [16], an index resembling B-trees [4, 7], is an example of a DC or key-value store that exploits these techniques. Indeed, it is an instance of a paradigm for how to achieve latch-freedom and log structuring more generally. In this paper, we describe a new architecture where the latch-free and log-structure techniques of the Bw-tree are implemented in a cache/storage subsystem capable of supporting multiple access methods, in the same way that a traditional cache/storage subsystem deals with latched access to fixed size pages that are written back to disks as in-place updates.

# A Look at Hardware Trends – 201x

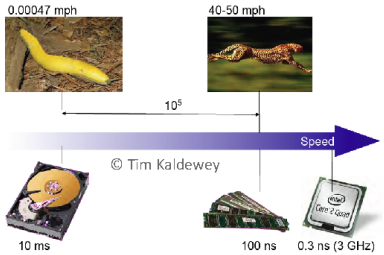
Dresden Database Systems Group

## > A Look at Hardware Trends

Database Technology Group

### Increasing Main Memory Capacity*

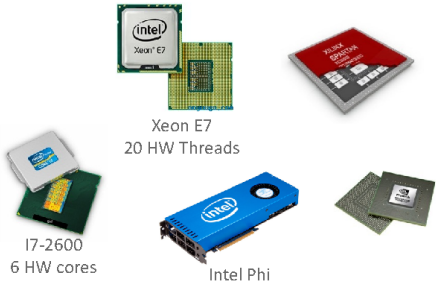
„Main Memory“ is the new disk!(?)




### Increasing Number of Cores

- CPU/GPU, hybrids
- FPGA (**Field Programmable Gate Array**)

„Parallelism“ is the name of the game!




* stable RAM will be an additional game changer

TECHNISCHE UNIVERSITÄT DRESDEN

11

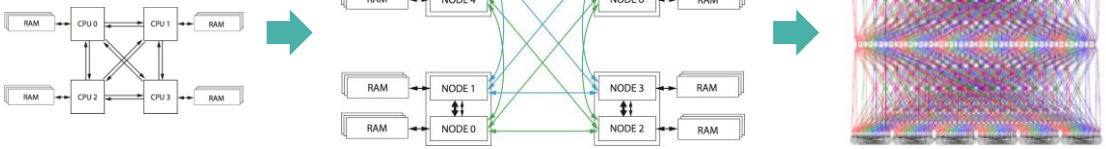
# A Look at Hardware Trends - 2015

Dresden Database Systems Group

System Level

Component Level

## Extreme NUMA Effects




1

„Main Memory“ is the new disk!(?)

„Parallelism“ is the name of the game!

## Storage-Class Memory

3

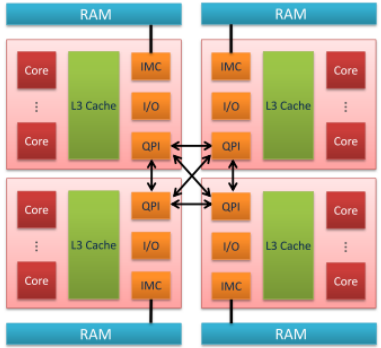
## Application-Specific Instruction Sets

2

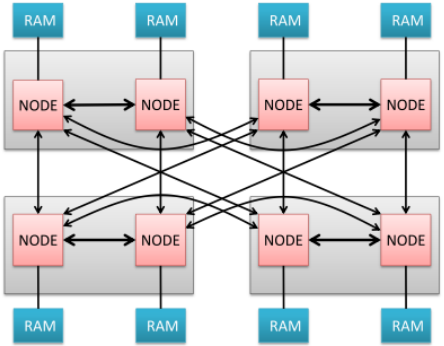
# Extreme NUMA Awareness

# NUMA Awareness

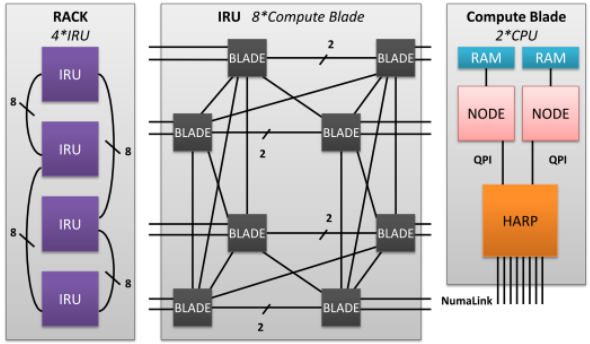
Dresden Database Systems Group




(a) Intel Machine (Detailed).




(b) AMD Machine (Topology View).




(c) SGI Machine (Topology View).

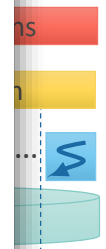
| Intel machine | | | AMD machine | | | SGI machine | | |
|---|---|---|---|---|---|---|---|---|
| distance | bandwidth (GB/s) | latency (ns) | distance (link width) | bandwidth (GB/s) | latency (ns) | distance | bandwidth (GB/s) | latency (ns) |
| local | 26.7 | 129 | local | 16.4 | 85 | local | 36.2 | 81 |
| 1 hop QPI | 10.7 | 193 | 1 hop HT (full link) | 5.8 | 136 | 2nd processor | 9.5 | 400 |
| | | | 1 hop HT (split,single) | 4.2 | 152 | 1 hop NUMALink | 7.5 | 505 - 515 |
| | | | 1 hop HT (split,dual) | 2.9 | 152 | 2 hop NUMALink | 7.5 | 625 - 635 |
| | | | 2 hop HT (split,single) | 3.7 | 196 | 3 hop NUMALink | 7.1 | 745 - 755 |
| | | | 2 hop HT (split,dual) | 1.8 | 196 | 4 hop NUMALink | 6.5 | 870 |

# TA versus Data-oriented Architecture (DORA)

Dresden Database Systems Group

***Which Architecture ?***

hitecture
*shared-nothing*



***Pros & Cons***

rallel systems

ndirection required

nal by design

***Challenges***

g indirection to work

***ry*** systems

oriented architecture

## Data-Oriented Transaction Execution


Ippokratis Pandis[1,2] ipandis@ece.cmu.edu
Ryan Johnson[1,2] ryanjohn@ece.cmu.edu
Nikos Hardavellas[3] nikos@northwestern.edu
Anastasia Ailamaki[2,1] natassa@epfl.ch

[1]Carnegie Mellon University Pittsburgh, PA, USA
[2]École Polytechnique Fédérale de Lausanne Lausanne, VD, Switzerland
[3]Northwestern University Evanston, IL, USA



**ABSTRACT**

While hardware technology has undergone major advancements over the past decade, transaction processing systems have remained largely unchanged. The number of cores on a chip grows exponentially, following Moore's Law, allowing for an ever-increasing number of transactions to execute in parallel. As the number of concurrently-executing transactions increases, contended critical sections become scalability burdens. In typical transaction processing systems the centralized lock manager is often the first contended component and scalability bottleneck.

In this paper, we identify the conventional thread-to-transaction assignment policy as the primary cause of contention. Then, we design DORA, a system that decomposes each transaction to smaller actions and assigns actions to threads based on which data each action is about to access. DORA's design allows each thread to mostly access thread-local data structures, minimizing interaction with the contention-prone centralized lock manager. Built on top of a conventional storage engine, DORA maintains all the ACID properties. Evaluation of a prototype implementation of DORA on a multicore system demonstrates that DORA attains up to 4.8x higher throughput than a state-of-the-art storage engine when running a variety of synthetic and real-world OLTP workloads.

chip equipped with 8 cores[1], while multicores targeting specialized domains find market viability at even larger scales. With experts in both industry and academia forecasting that the number of cores on a chip will follow Moore's Law, an exponentially-growing number of cores will be available with each new process generation.

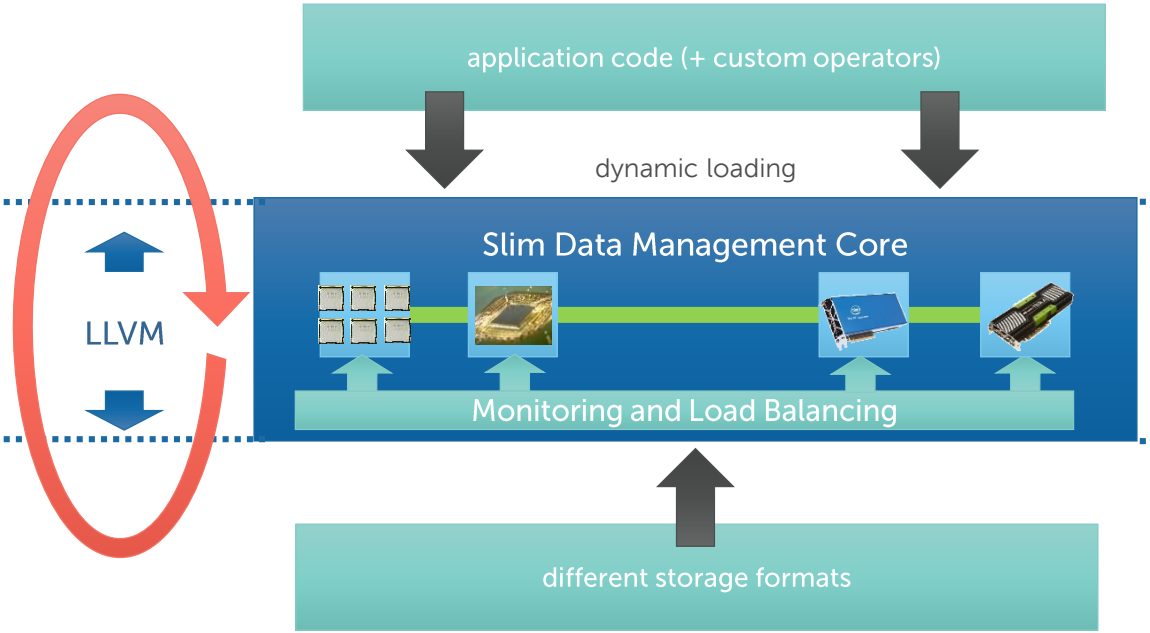
As the number of hardware contexts on a chip increases exponentially, an unprecedented number of threads execute concurrently, contending for access to shared resources. Thread-parallel applications, such as online transaction processing (OLTP), running on multicores suffer of increasing delays in heavily-contended critical sections, with detrimental performance effects [14]. To tap the increasing computational power of multicores, the software systems must alleviate such contention bottlenecks and allow performance to scale commensurately with the number of cores.

OLTP is an indispensable operation in most enterprises. In the past decades, transaction processing systems have evolved into sophisticated software systems with codebases measuring in the millions of lines. Several fundamental design principles, however, have remained largely unchanged since their inception. The execution of transaction processing is full of critical sections [14]. Consequently, these systems face significant performance and scalability problems on highly-

# ERIS Data Management Core

Dresden Database Systems Group

... an academic playground for modern DB techniques




## Characteristics

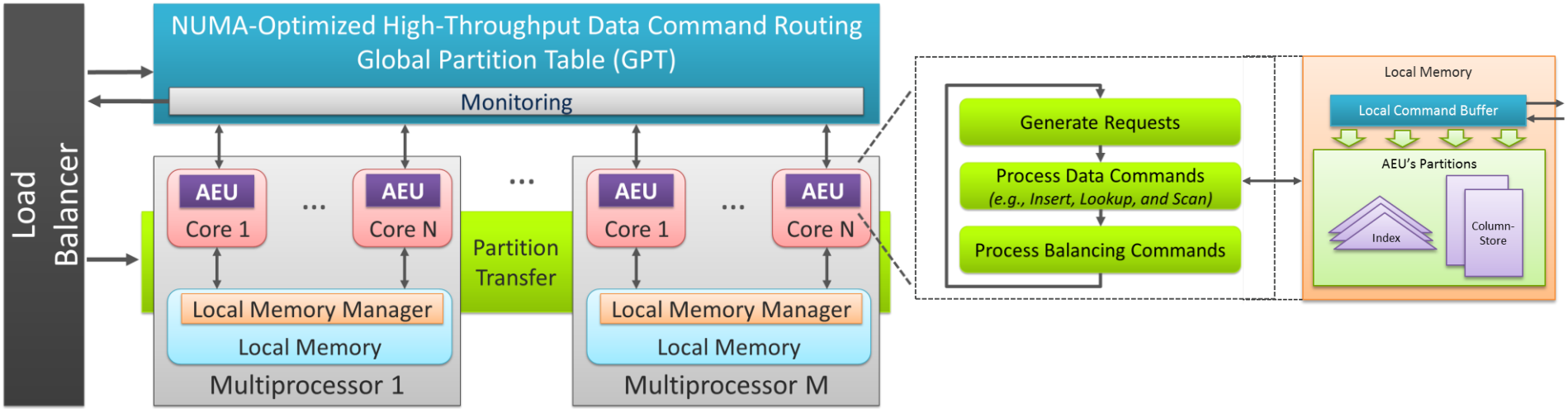
- data-oriented architecture (via message passing)
- NUMA awareness
- heterogeneous hardware
- aggressive elasticity strategies
- dynamic data placement policies

# ERIS Overall Architecture

Dresden Database Systems Group

## Data-oriented Architecture

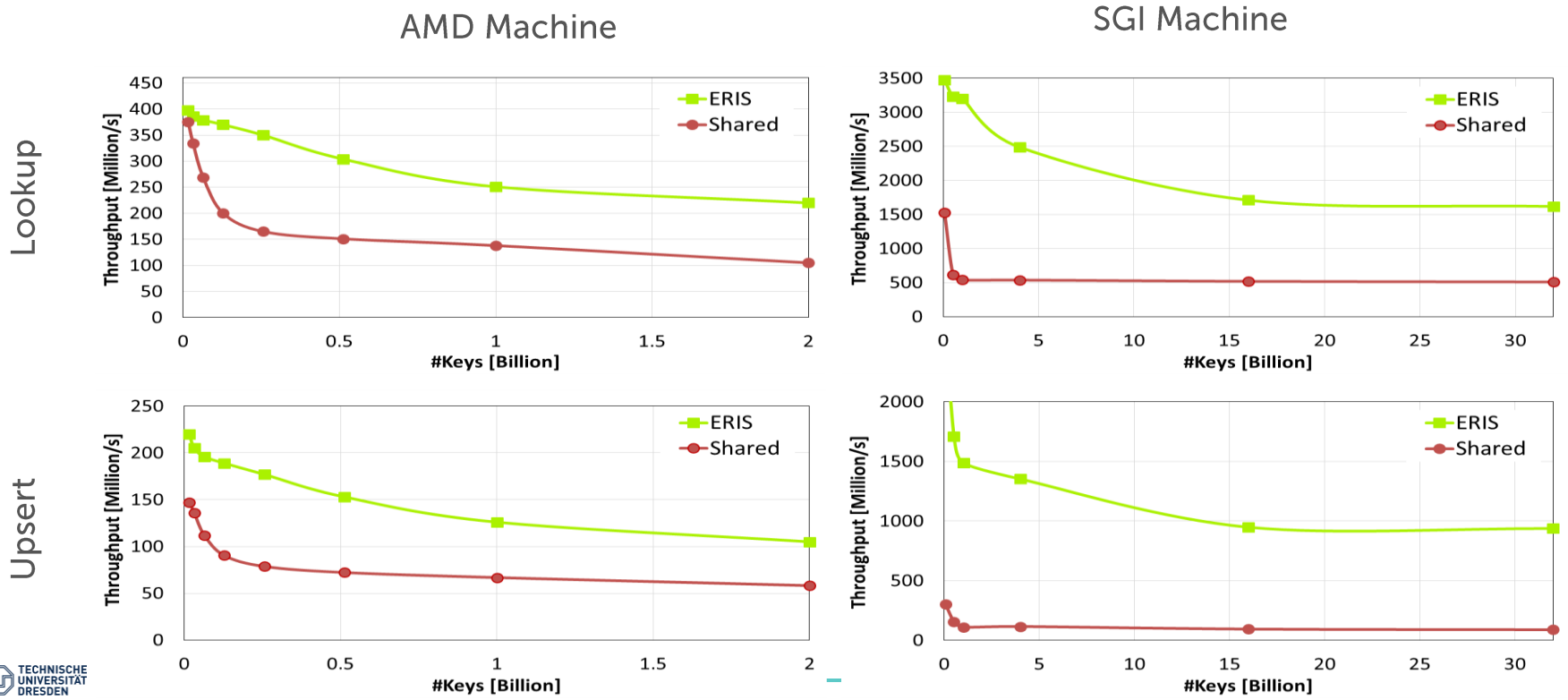
- Follows MVCC principle
- Distribution based on logical partitioning
- Aggressive re-partioning using copy as well as link strategies



# Evaluation: Some MicroBenchmarking

Dresden Database Systems Group

## Lookup/Upsert Throughput Depending on Index Size

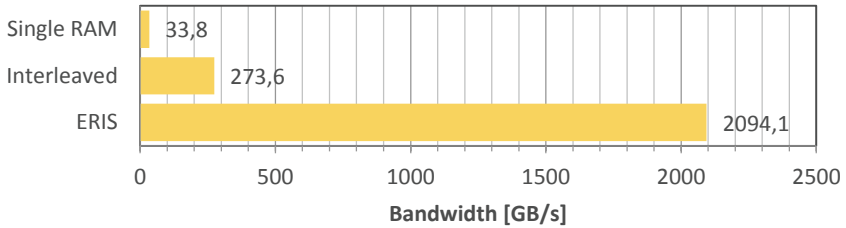


# Evaluation: Scan Throughput

Dresden Database Systems Group

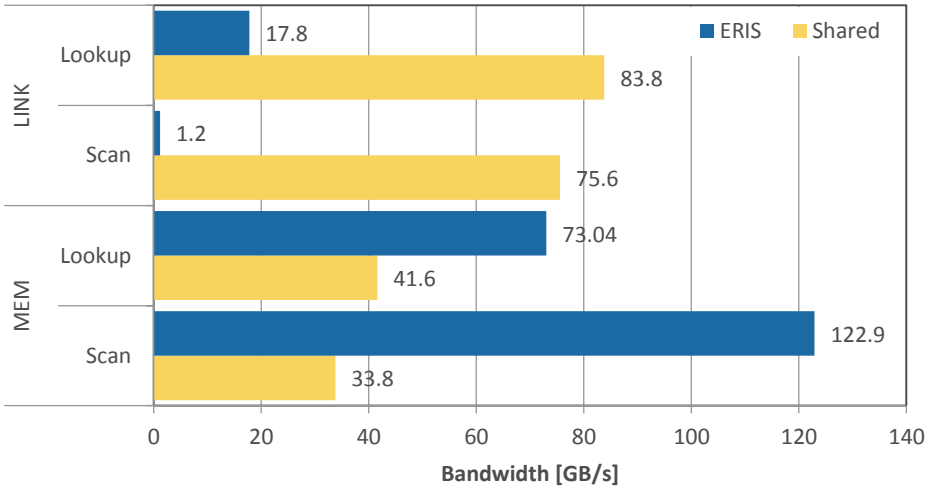
## Scan Performance

- SGI Machine
- 488 cores – parallel scan
- 8 billion entries in the column store




## Link and Memory Controller Activity

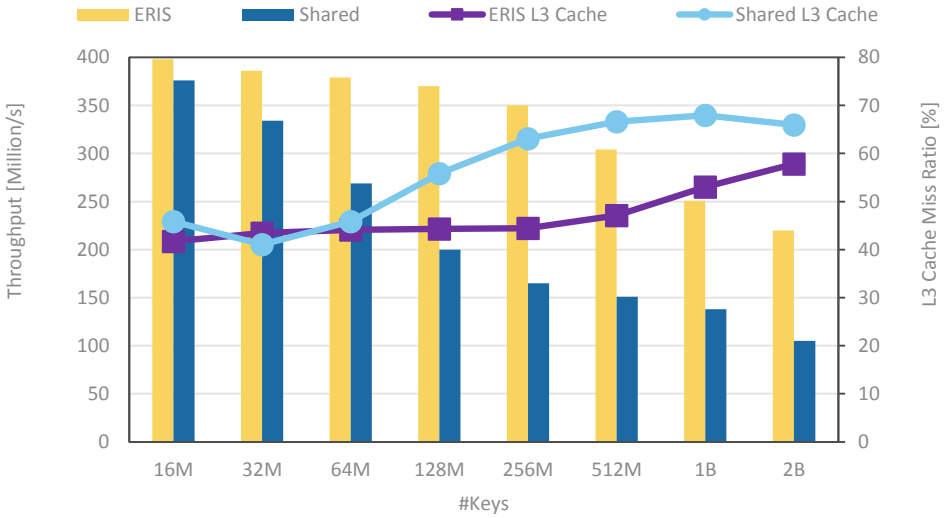
- AMD Machine
- Scan: 8B Keys
- Lookup: 1B Keys



# Evaluation: L3 Cache Usage

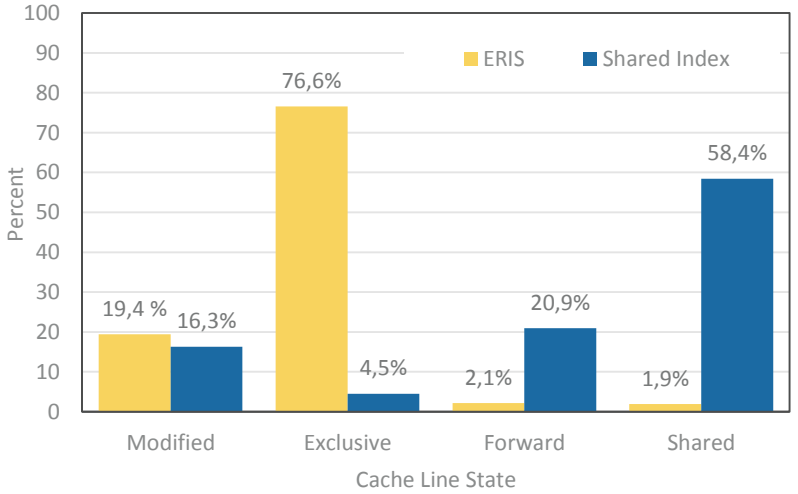
Dresden Database Systems Group

## L3 Cache Usage – Index Lookup




## L3 Cache Line State – Index Lookup

- Percentage of all hits
- 1B keys



# Key Component: Load Balancer

Dresden Database Systems Group

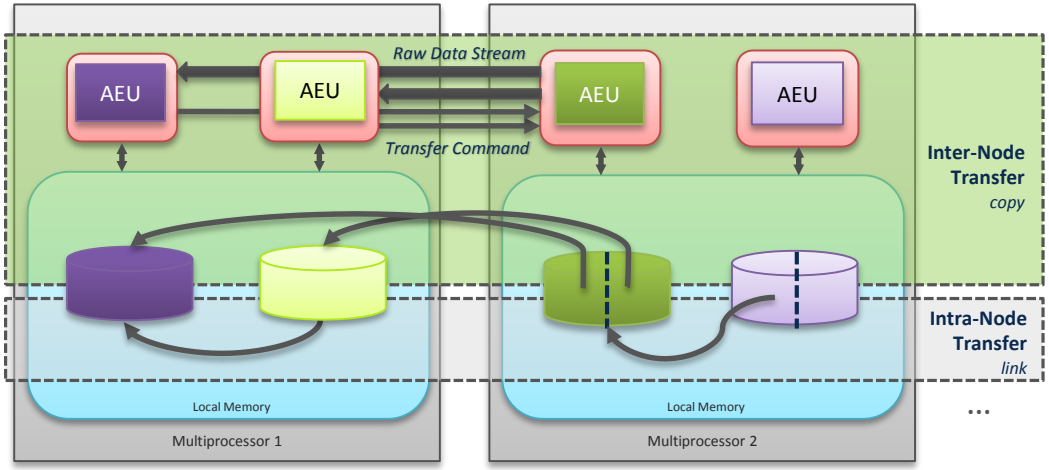
## Balancing Strategies

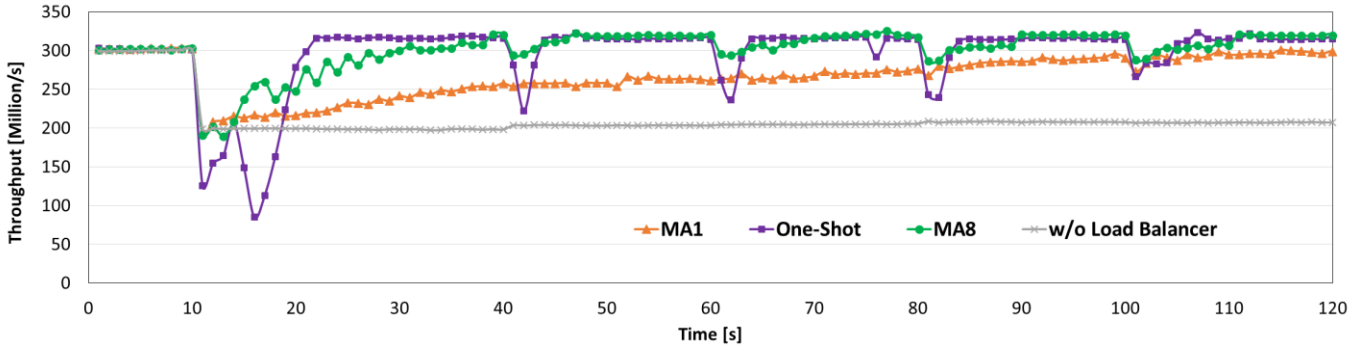
- Copy Strategy:

  copy data within NUMA systems between different sockets

- Link Strategy:

  delegate pointer, delay data movment



# What's Next? Wireless Interconnects!

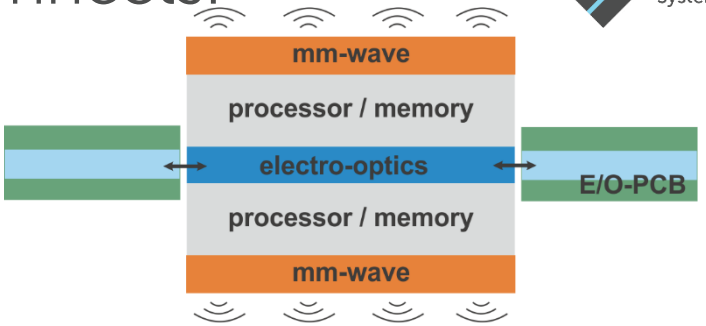
Dresden Database Systems Group

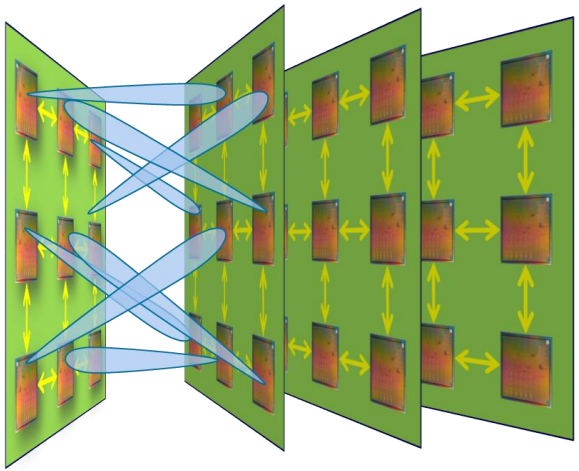
## Advances in wired interconnects

- Optical interconnects, …

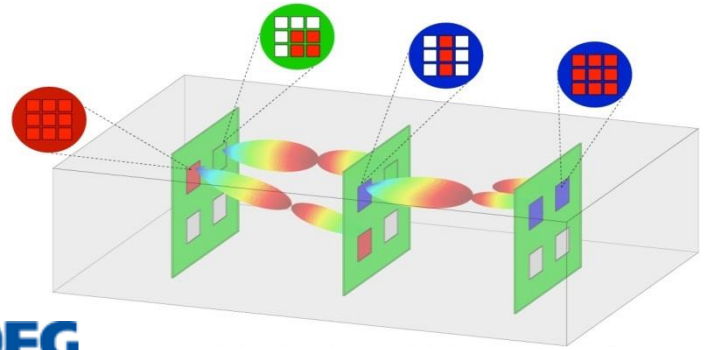
## …but: the breakthrough may come with

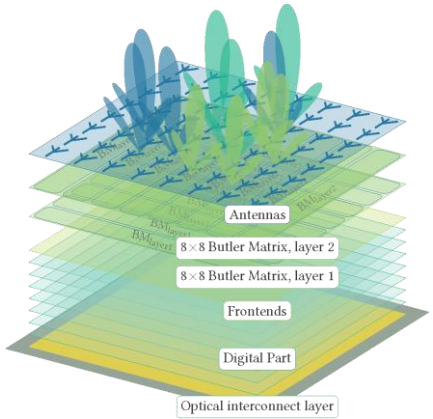
- High-speed, short-range wireless links



**Antennas and Wave Propagation for Adaptive Wireless Backplane Communication**



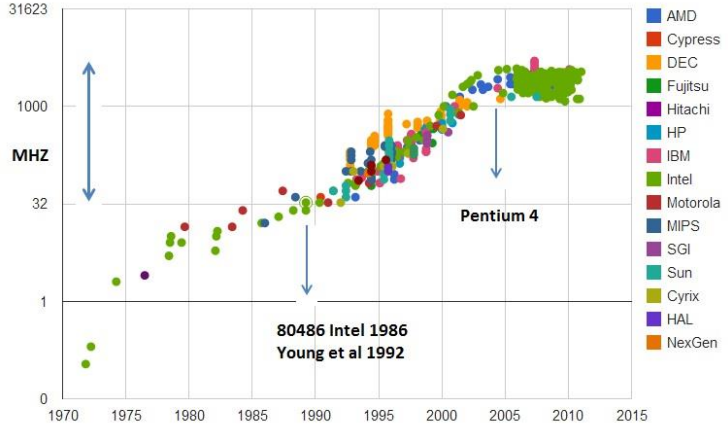



DFG Deutsche Forschungsgemeinschaft

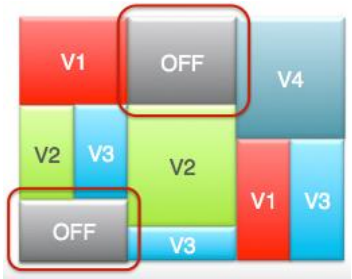
# Database-specific Instructions

# xPU Developments and Consequences

Dresden Database Systems Group



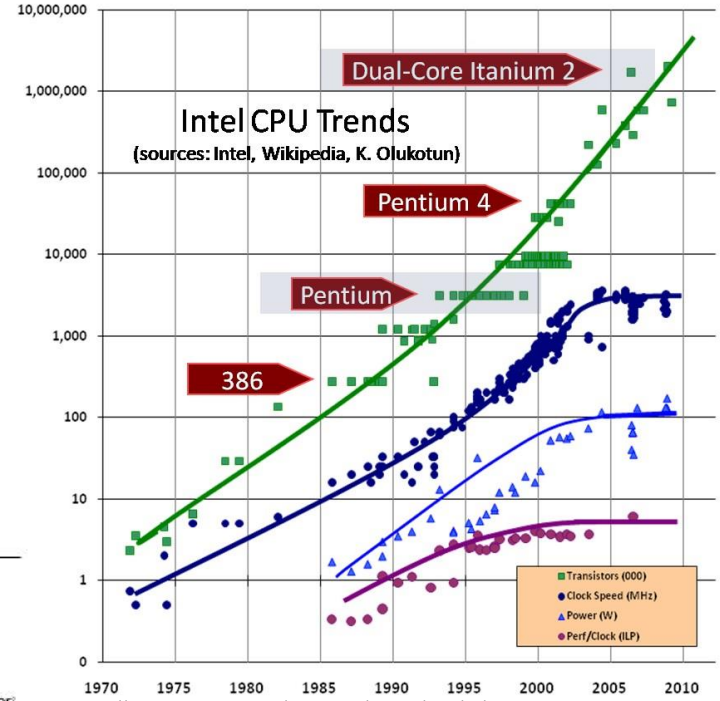

http://upload.wikimedia.org/wikipedia/en/c/ce/Clock_CPU_Scaling.jpg




*Appears in the Proceedings of the 38th International Symposium on Computer Architecture (ISCA '11)*

**Dark Silicon and the End of Multicore Scaling**

Hadi Esmaeilzadeh[†] Emily Blem[‡] Renée St. Amant[§] Karthikeyan Sankaralingam[‡] Doug Burger[*]
[†]University of Washington [‡]University of Wisconsin-Madison
[§]The University of Texas at Austin [*]Microsoft Research
hadianeh@cs.washington.edu blem@cs.wisc.edu stamant@cs.utexas.edu karu@cs.wisc.edu dburger@microsoft.com




http://www.extremetech.com/wp-content/uploads/2012/02/CPU-Scaling.jpg

**ABSTRACT**

Since 2005, processor designers have increased core counts to exploit Moore's Law scaling, rather than focusing on single-core per-

ture, and compiler advances, Moore's Law, coupled with Dennard scaling [11], has resulted in commensurate exponential performance increases. The recent shift to multicore designs has aimed to in-

# Motivation of „DB Processor“

Dresden Database Systems Group

## Today's Database Servers

- Fat cores (area & power)
- Few HW adaptions
- CMOS scaling


## Database Machines (Ancient)

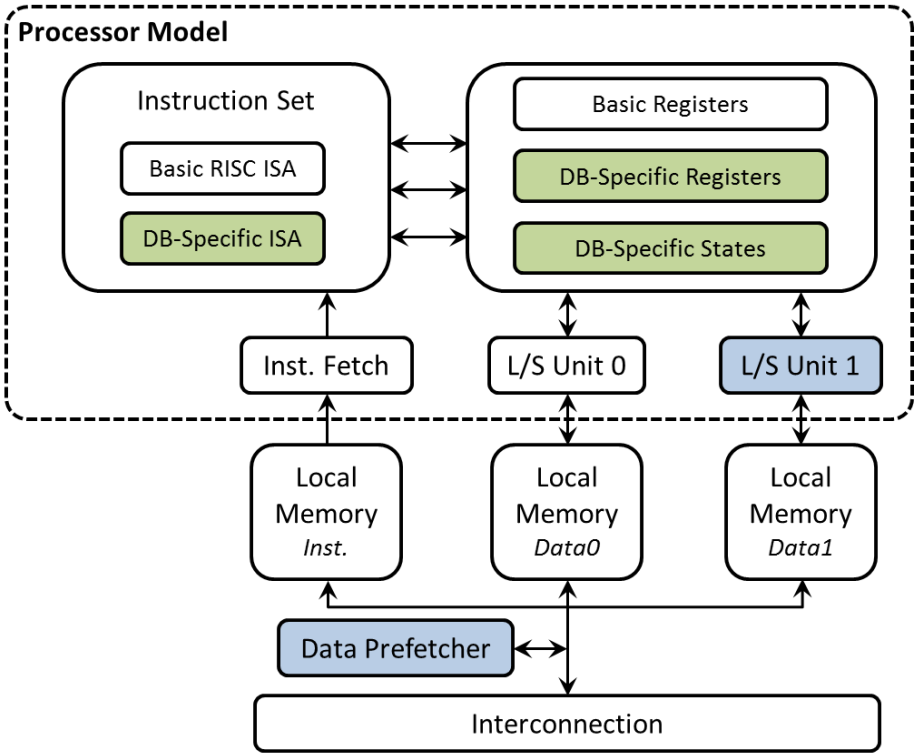
- Processors build from scratch
- Long development cycles
- High development costs

## Our Approach

- HW/SW Co-design
- Customizable processor
- Application-specific ISA extensions
- Tool flow & short HW development cycles

# Customizable Processor Model

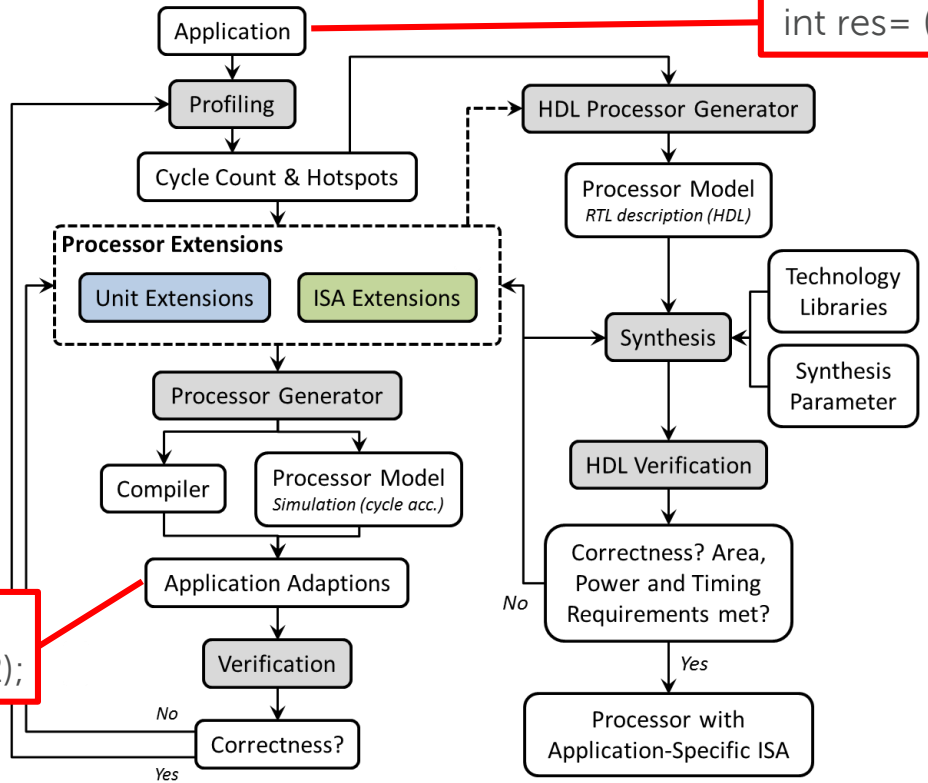





Basic Core:
Tensilica LX4

# Tool Flow






```
int res= (v0 + v1 + v2) >> shift8;
```

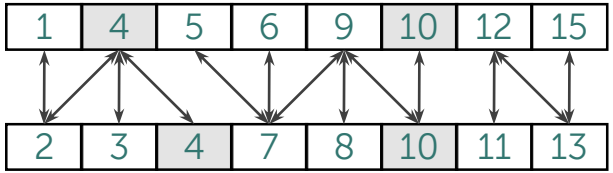
```
// shift8 -> internal state
int res=add3_shift(v0, v1, v2);
```

# Application Scenario

Dresden Database Systems Group

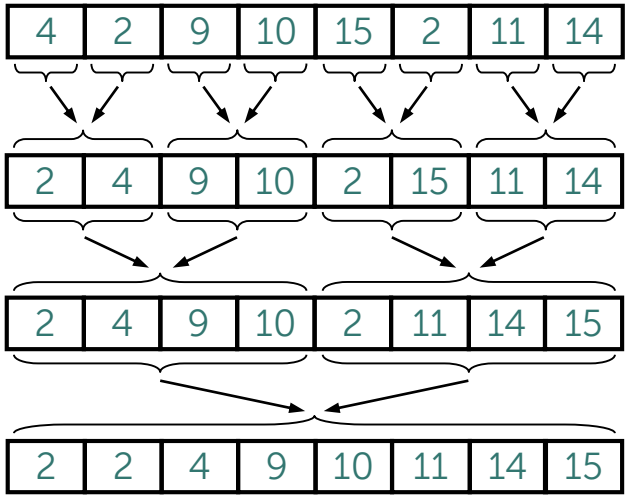
## Set Operations

- Intersection
- Difference
- Union




## Sorting

- Merge Sort




## Hash Operations

- Integer Hashing
- String Hashing
- Hash Table Management

# Sorted-Set Intersection

Dresden Database Systems Group

```
int intersect (int* A, int* B, int l_a, int l_b, int* C) {
        int pos_a = 0, pos_b = 0, pos_c = 0;
        while( pos_a < l_a && pos_b < l_b ) {
                if( A[ pos_a ] == B[ pos_b ] ) {
                        C[ pos_c++ ] = A[ pos_a ];
                        pos_a++;
                        pos_b++;
                }
                else if( A[ pos_a ] < B[ pos_b ] )
                        pos_a++;
                else
                        pos_b++;
        }
        return pos_c;
}
```

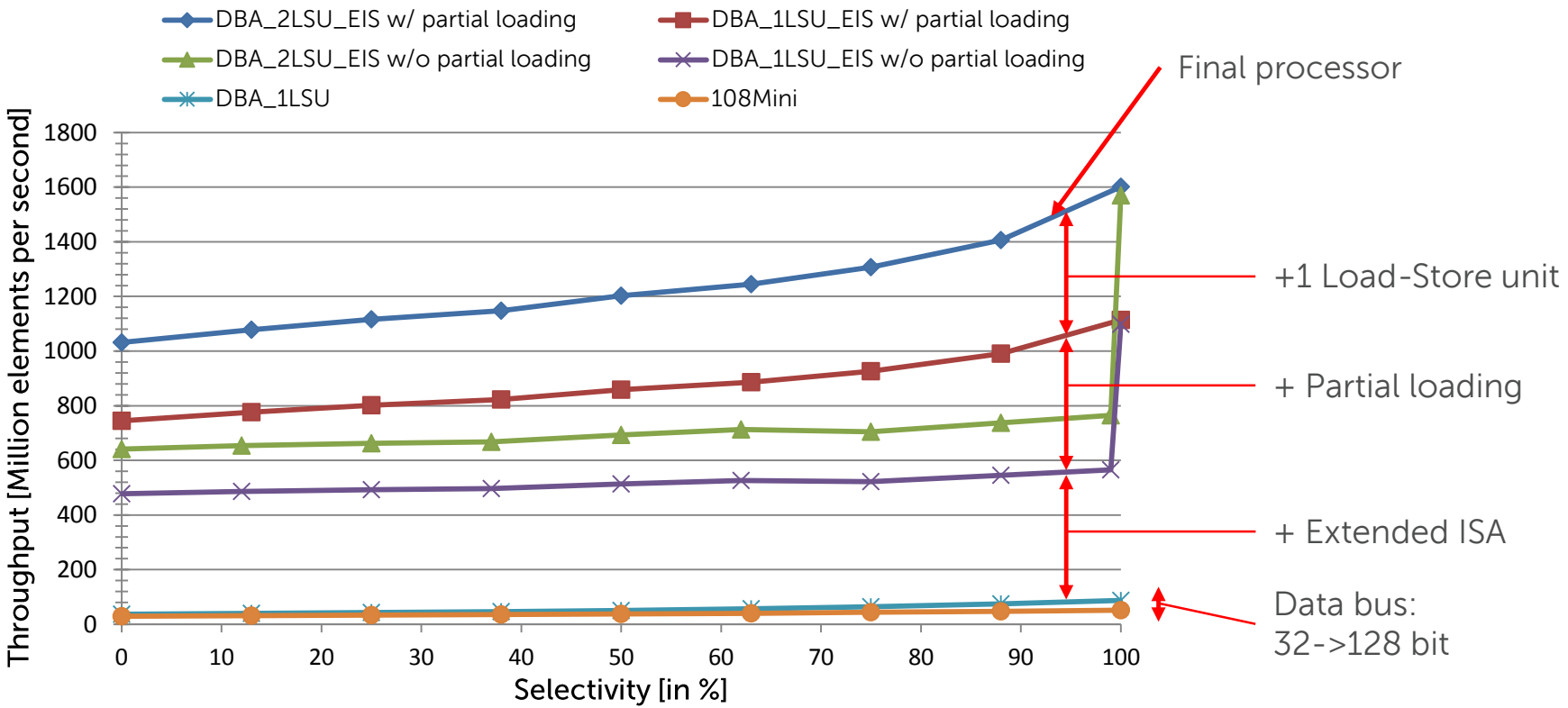
internal states

merged in one inst.

➔ + 2x 128 bit data busses + explicit load-store + SIMD + ...

# Selectivity: Intersection

Dresden Database Systems Group



# Throughput [Mib/s]

Dresden Database Systems Group

Selectivity: 50%

+1 Load-Store unit

| Processor | Partial Load–Store | f[MHz] | Intersection | Union | Difference | Merge–Sort |
|---|---|---|---|---|---|---|
| 108Mini[1] | - | 442 | 31.3 | 26.4 | 35.7 | 1.7 |
| DBA_1LSU | - | 435 | 50.7 | 47.7 | 50.4 | 3.2 |
| DBA_1LSU_EIS | no | 424 | 513.4 | 665.0 | 658.8 | 29.3 |
| DBA_2LSU_EIS | no | 410 | 693.0 | 643.0 | 637.0 | 28.3 |
| DBA_1LSU_EIS | yes | 424 | 859.0 | 574.2 | 859.0 | 29.3 |
| DBA_2LSU_EIS | yes | 410 | 1203.0 | 780.4 | 1192.6 | 28.3 |

Data bus: 32->128 bit

Final processor

➔ DBA_2LSU_EIS vs. 108MINI: 38x 30x 33x 17x

# Comparison

Dresden Database Systems Group

## Sorted-Set Intersection

| | Intel i7-920 | DBA_2LSU_EIS | |
|---|---|---|---|
| Throughput (elements/s) | 1,100 mio | 1,203 mio | |
| Clock frequency | 2.67 GHz | 0.41 GHz | → 7x improvement |
| Max. TDP | 130 W | 0.135 W | → 963x improvement |
| Cores/Threads | 4/8 | 1/1 | |
| Feature size | 45 nm | 65 nm | |
| Area (logic & memory) | 263 $mm^2$ | 1.5 $mm^2$ | → 175x improvement |

# Timing and Area

Dresden Database Systems Group

| Technology | Processor | $A_{LOGIC}\,[mm^2]$ | $A_{MEM}\,[mm^2]$ | $f_{MAX}$ [MHz] | P[mW] @$f_{MAX}$ |
|---|---|---|---|---|---|
| 65 nm | 108Mini | 0.220[1] | - | 442[1] | 27.4[1] |
| | DBA_1LSU | 0.177 | 0.874 | 435 | 56.6 |
| | DBA_2LSU | 0.177 | 0.870 | 429 | 57.1 |
| | DBA_1LSU_EIS | 0.523 | 0.874 | 424 | 123.5 |
| | DBA_2LSU_EIS | 0.645 | 0.870 | 410 | 135.1 |
| 28 nm | DBA_2LSU_EIS | 0.169 | 0.232 | 500 | 47.0 |

[1] http://www.tensilica.com/uploads/pdf/108Mini.pdf

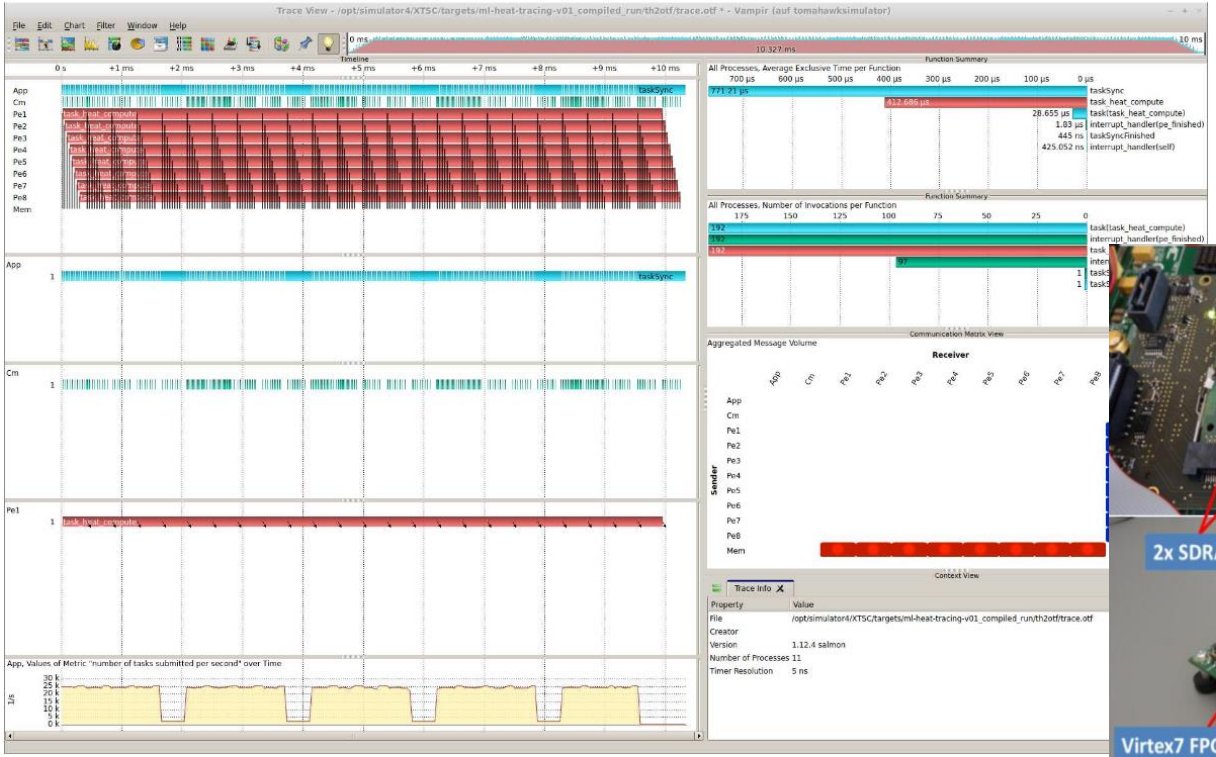
Final processor

Relative Area Consumption(DBA_2LSU_EIS)

| Part | Area[%] |
|---|---|
| Basic Core | 20.5 |
| Decoding/Muxing | 14.4 |
| States | 14.7 |
| Op: All | 11.3 |
| Op: Intersection | 6.8 |
| Op: Difference | 9.0 |
| Op: Union | 17.6 |
| Op: Merge-Sort | 5.7 |
| SUM | 100 |

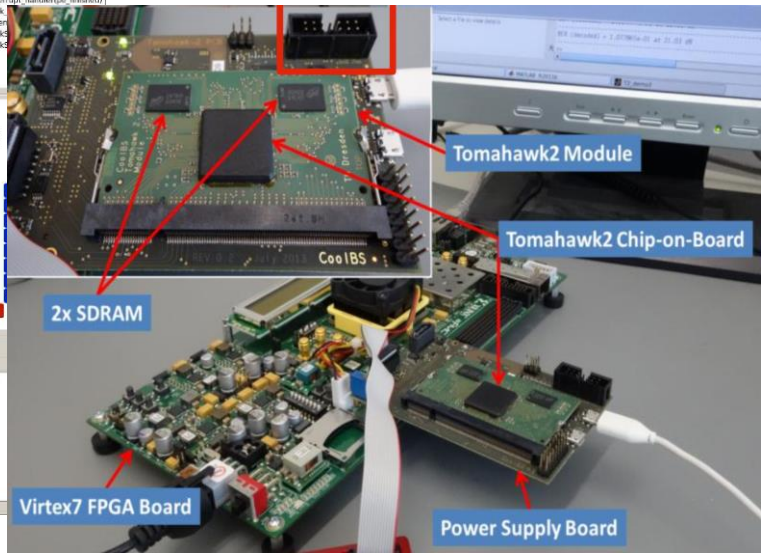
# Tomahawk2 Programming

Dresden Database Systems Group



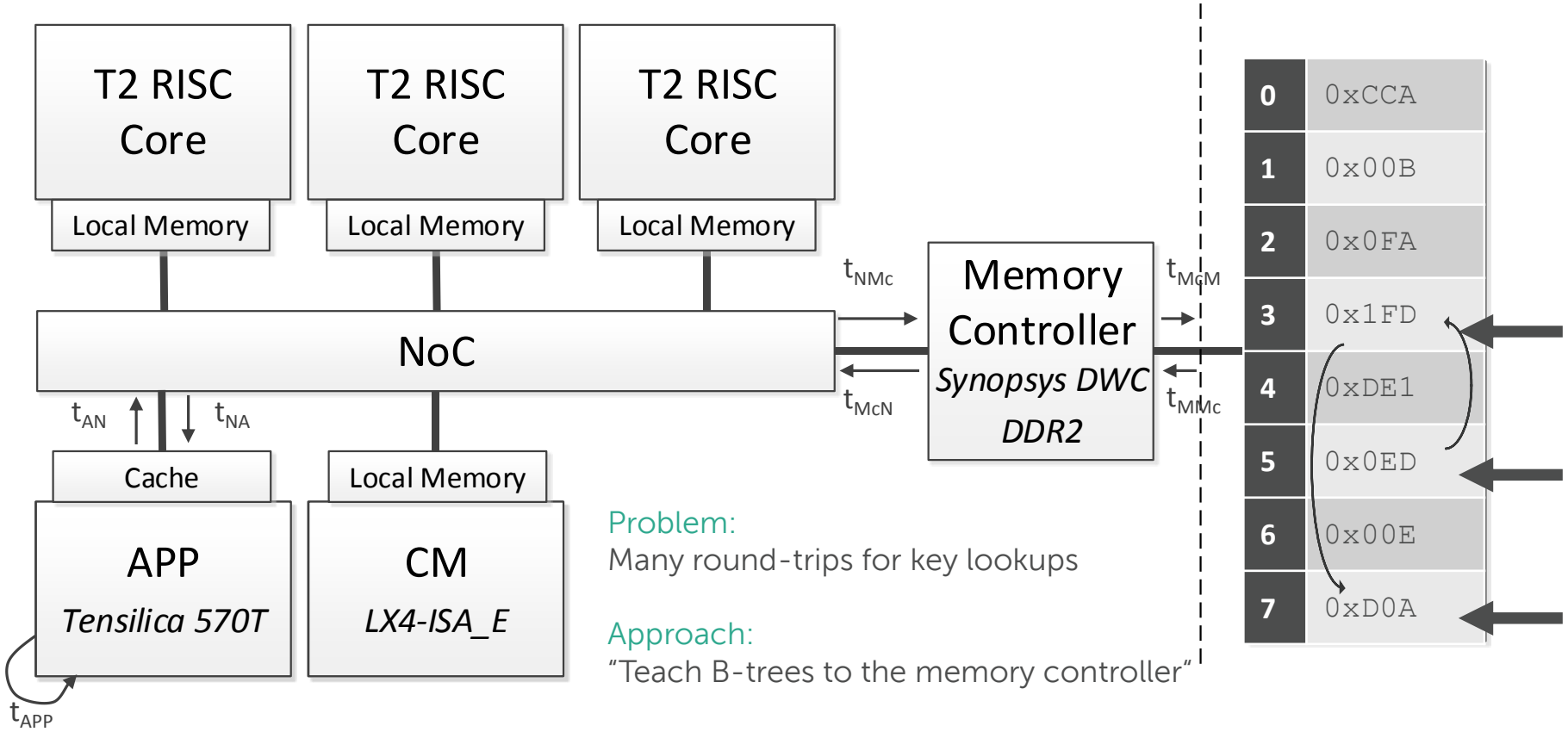

Profiling Software

Hardware Setup



# What's next? → Pointer Chasing

Dresden Database Systems Group

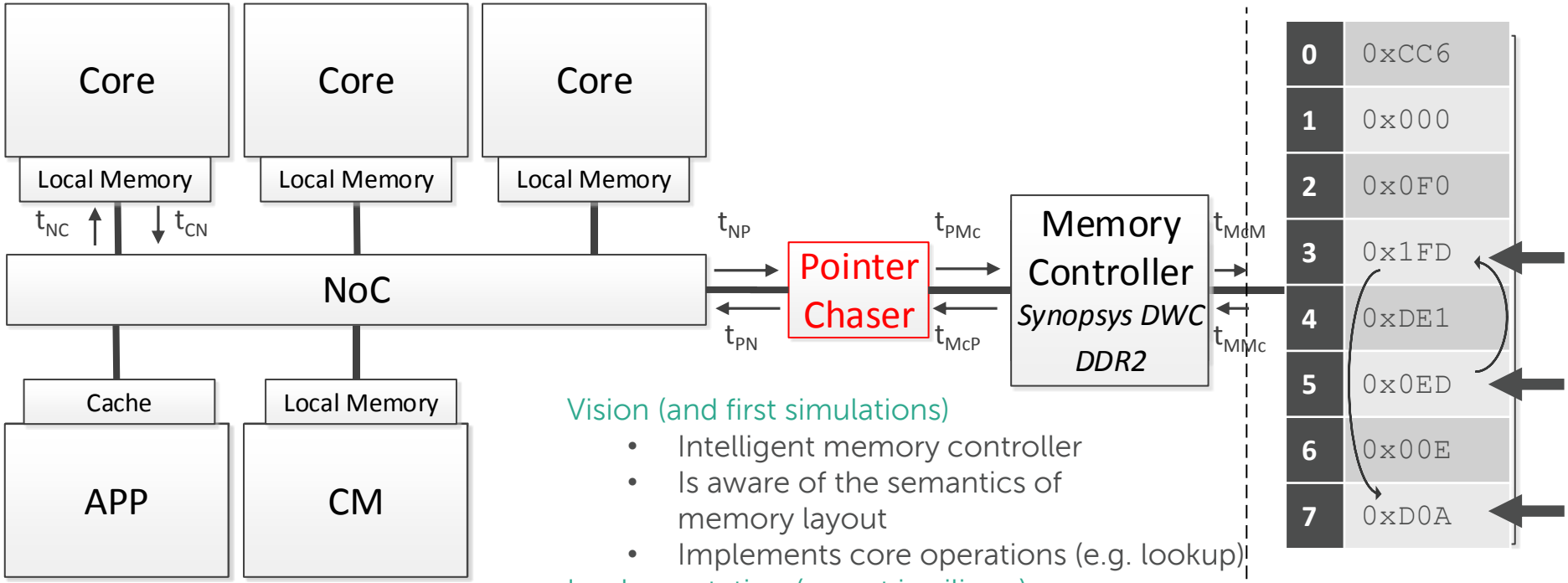



Problem:
Many round-trips for key lookups

Approach:
"Teach B-trees to the memory controller"

# Basic Principle

Dresden Database Systems Group




Vision (and first simulations)

- Intelligent memory controller
- Is aware of the semantics of memory layout
- Implements core operations (e.g. lookup)

Implementation (no yet in silicon)

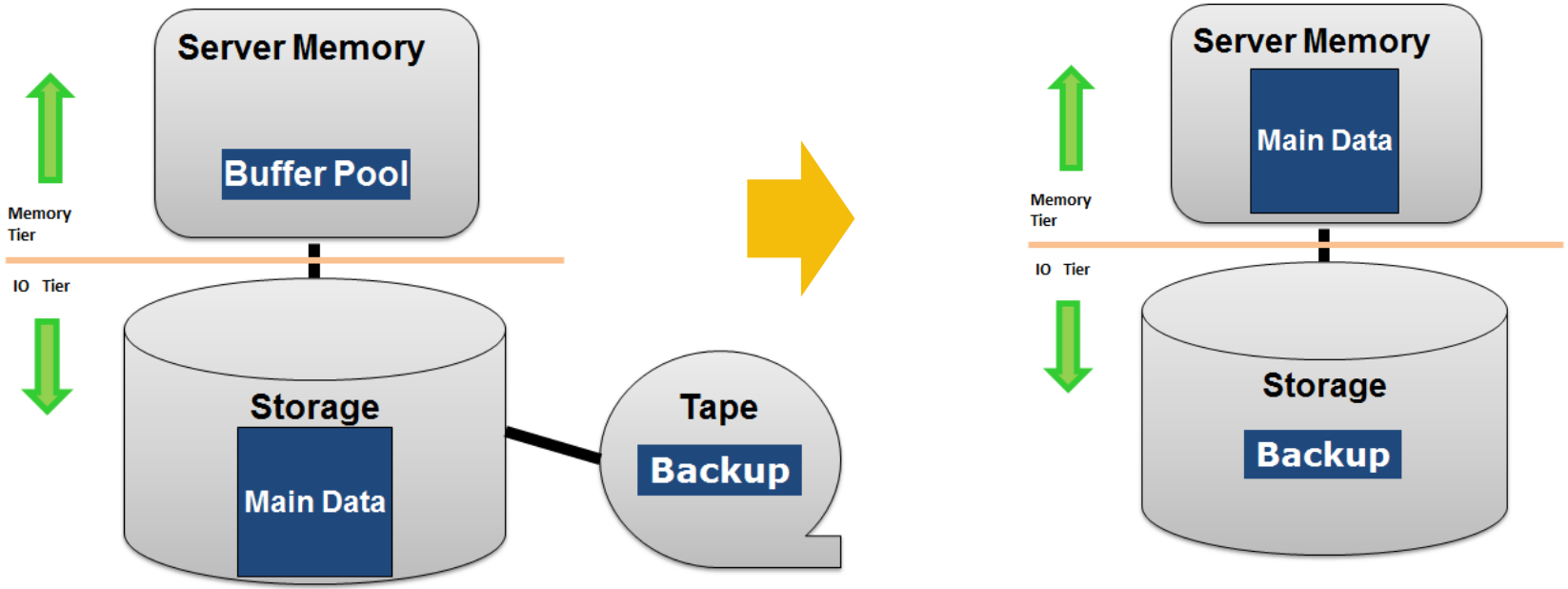
- 0,183mm$^2$ PE with 200Mhz

# Storage Class Memory

Project „SOFORT“  

# Database Evolution (10 years ago...)

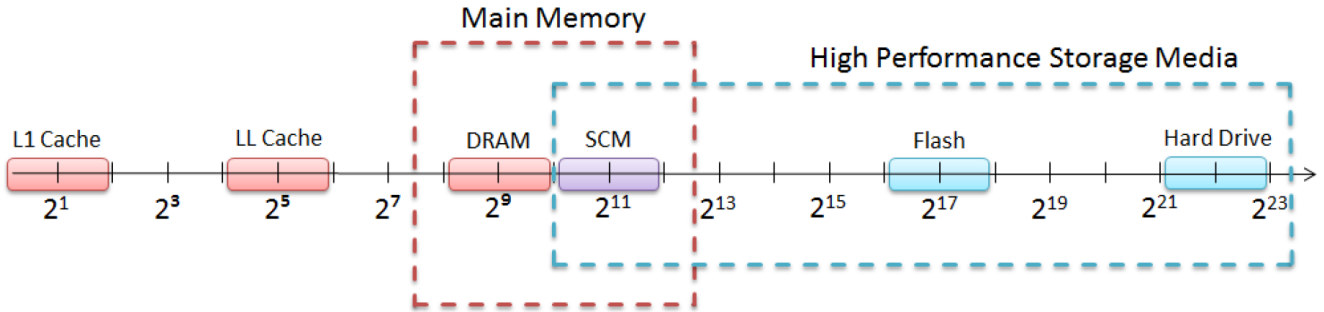
Dresden Database Systems Group




DRAM is hitting costs and power consumption limits!

# Storage Class Memory / Non-Volatile RAM

Dresden Database Systems Group




Adapted from: M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In ISCA 2009

## Emerging Technology

- Examples: MRAM(IBM), FRAM (Ramtron), PCM(Samsung)
- Merging Point between storage and memory

## Advantages

- ~4x denser than DRAM
- SCM does not consume energy if not used
- SCM is cheaper, persistent, byte-addressable

## Drawbacks

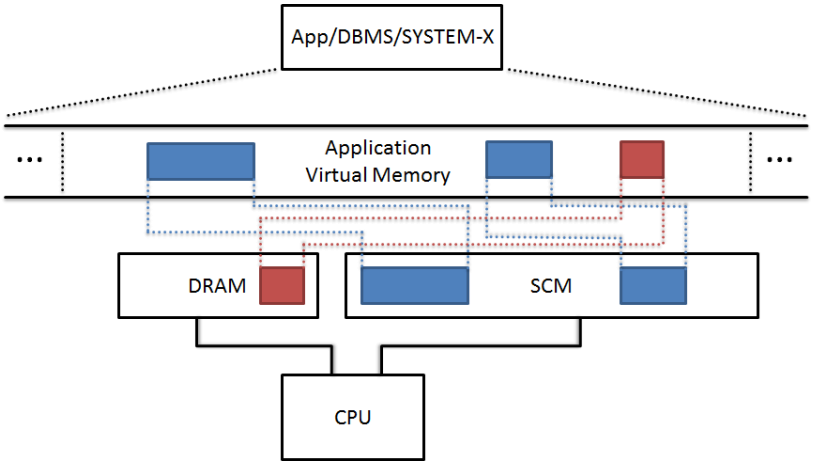
- Number of writes is limited (life expectancy 2-10 years)
- SCM has higher latency than DRAM
  - Read latency ~2x slower than DRAM
  - Write latency ~10x slower than DRAM

# SCM Access // File Level

Dresden Database Systems Group

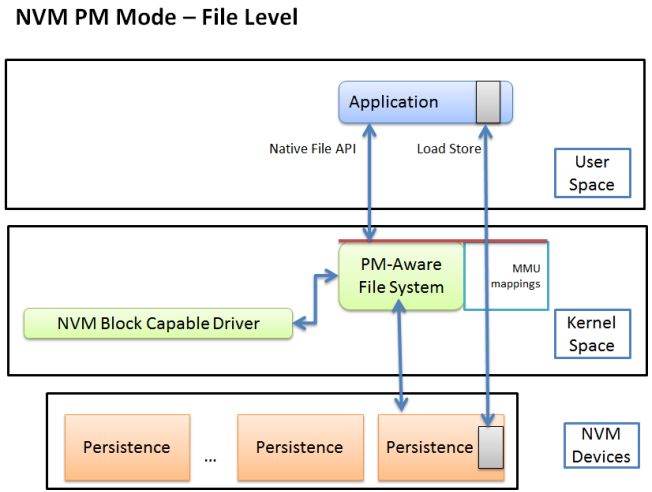
## Hybrid Storage Model

- Application may distinguish between
  - Traditional memory
  - Persistent memory blocks
- Files names are used as identification handle

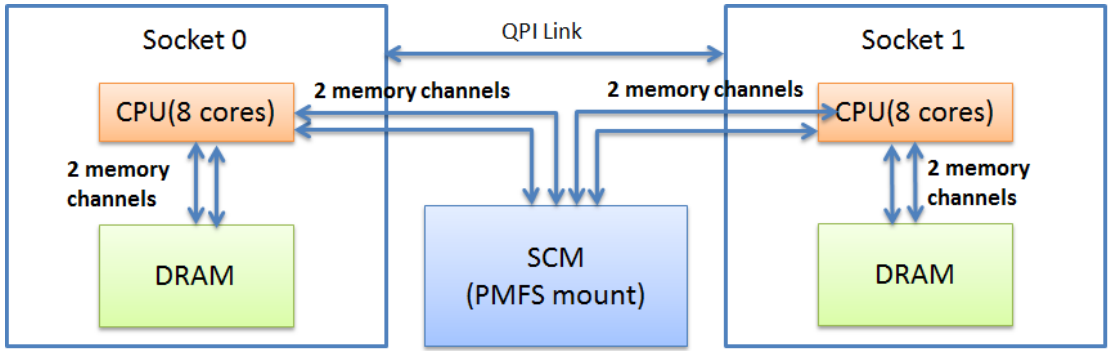



## NVM PM Mode – File Level (PMFS)

- Shows how to get persistent memory to the application level
- Requires a persistent memory-aware file system
- Direct access to regions of persistent memory inside the application



# SCM Hardware Simulator




## Experimental Setup

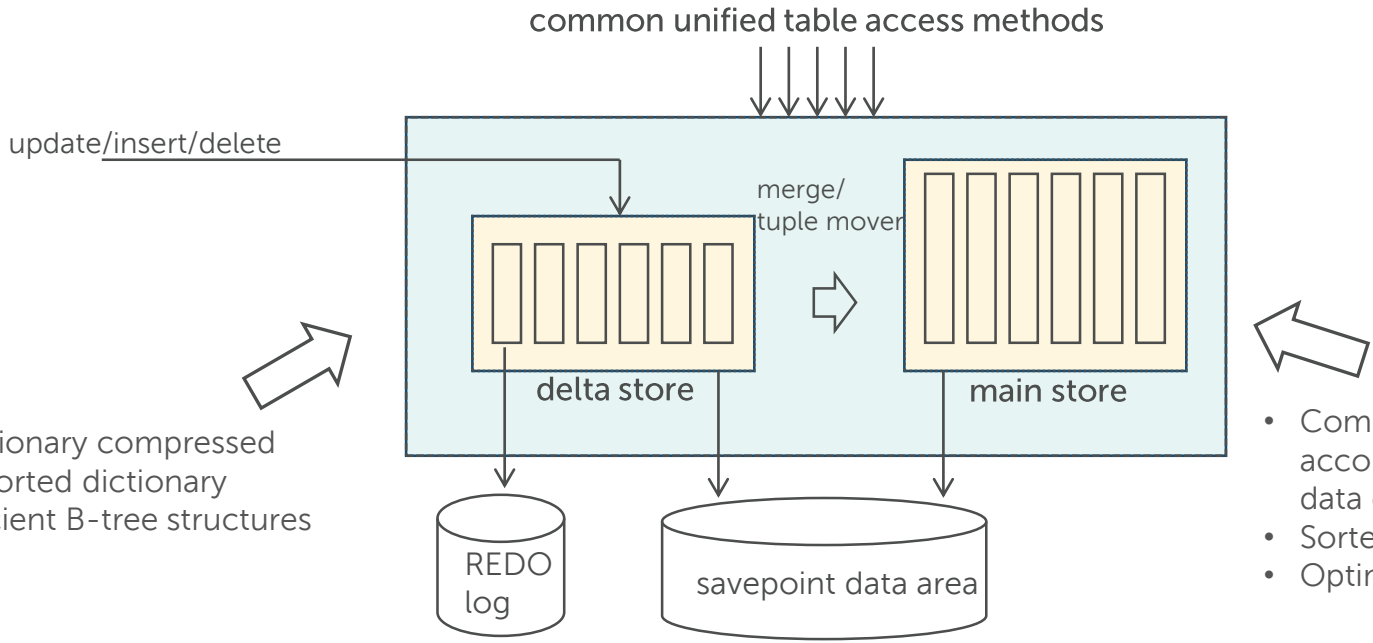
- hybrid architecture with both DRAM and SCM, software can allocate in SCM through a seperate memory interface
- Special BIOS allows simulation of different latency by the means of a microcode patch
  - Conceptually, local DRAM is attached to each socket, whereas the SCM is attached to both sockets emulating a uniform SCM region by interleaving memory on a cache-line level.
- SCM latency is fixed to 200ns by default. DRAM latency is 90ns.

# Hybrid Storage Architecture for Column Stores

Dresden Database Systems Group

## Use of compression implies two stores

- Write optimized store (WOS)
- Read optimized (compressed) store (ROS)




- Dictionary compressed
- Unsorted dictionary
- Efficient B-tree structures

- Compression schemes according to existing data distribution
- Sorted dictionary
- Optimized for HW-scans

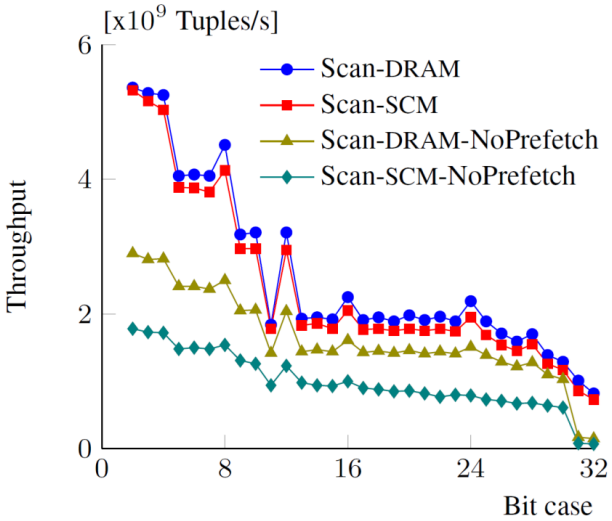
# NVRAM for ROS-Structure

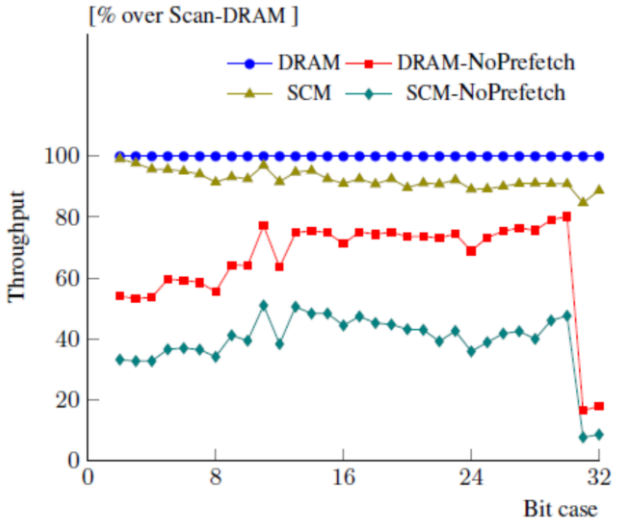
Dresden Database Systems Group

## Micro Experiments: SIMD Scan performance on DRAM and SCM

- With prefetching: average penalty for using SCM instead of DRAM is only 8%.
- Without prefetching: average penalty for using SCM instead of DRAM is 41%.

## NOTE For operators with sequential memory access patterns, SCM performs almost as good as DRAM



# NVRAM for WOS-Structure

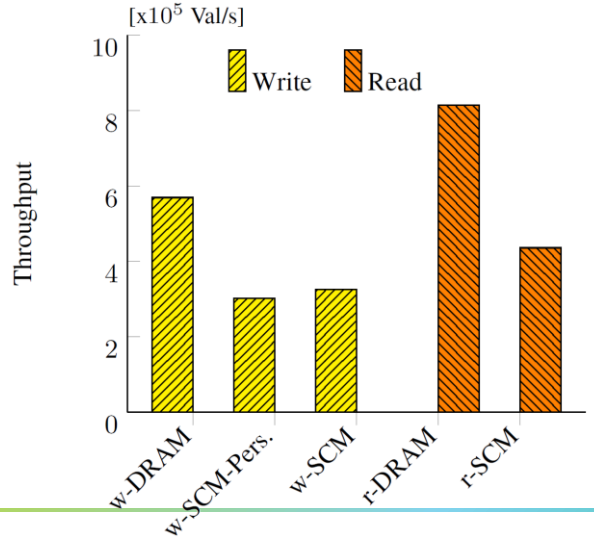
Dresden Database Systems Group

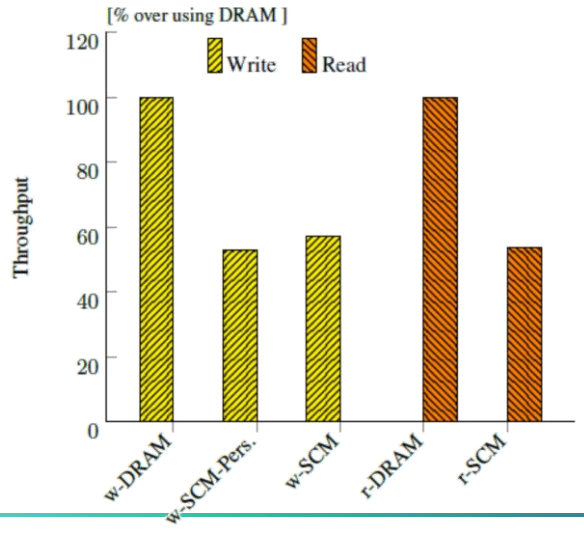
## Micro Experiment

- Skip List read/write performance on DRAM and SCM
- 47% penalty for reads, and 43-47% penalty for writes for using SCM instead of DRAM.

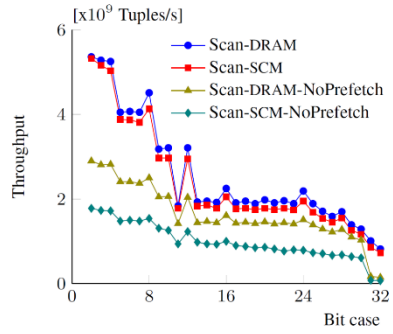
## Operations with random memory access patterns are expensive in SCM

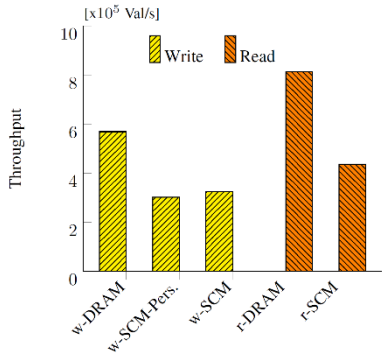
- Writing persistent and concurrent data structures is NOT trivial

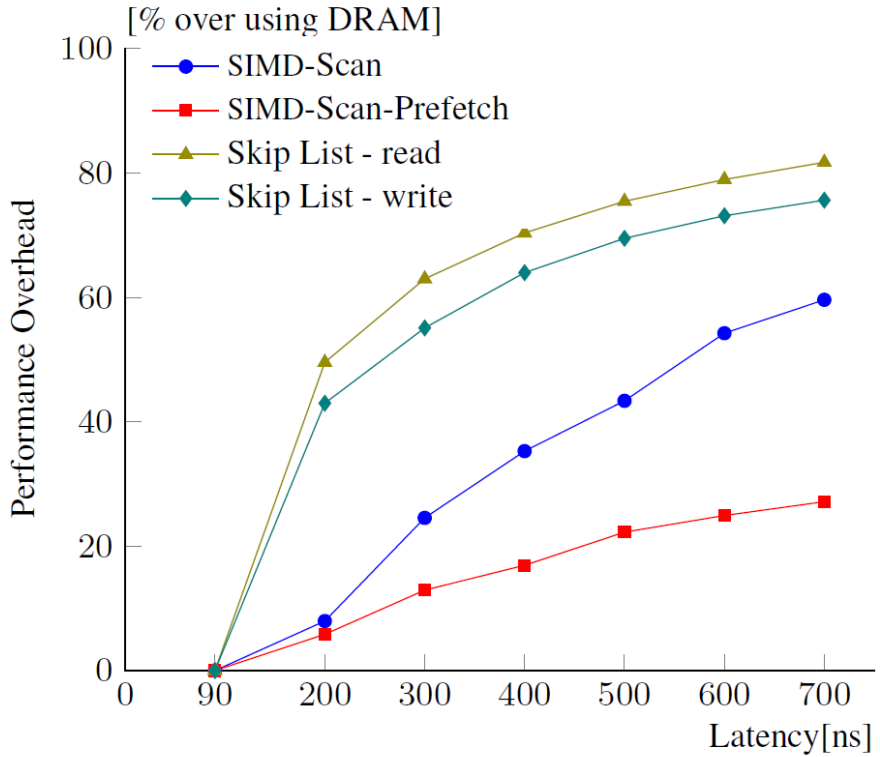


# Impact of NVRAM latency




SIMD: 16bit case / SL: 64Byte

# SCM Recovery

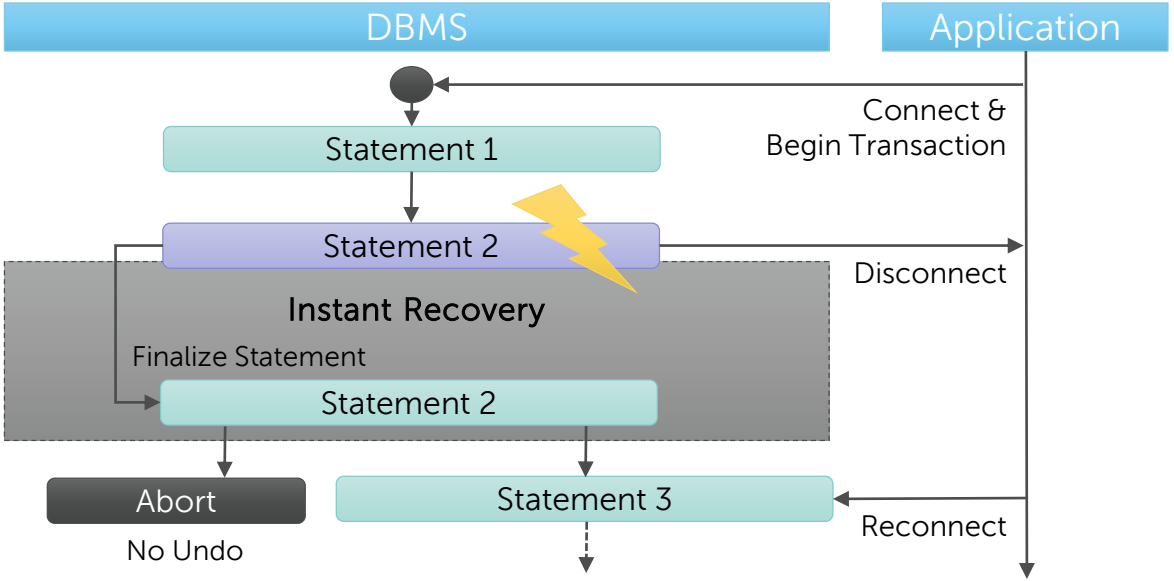
Dresden Database Systems Group

## Drawback of traditional approach with large main-memories

- System is operational after recovery of all database objects is performed
- Recovery may take a significant amount of time

## Goal: Instant Recovery

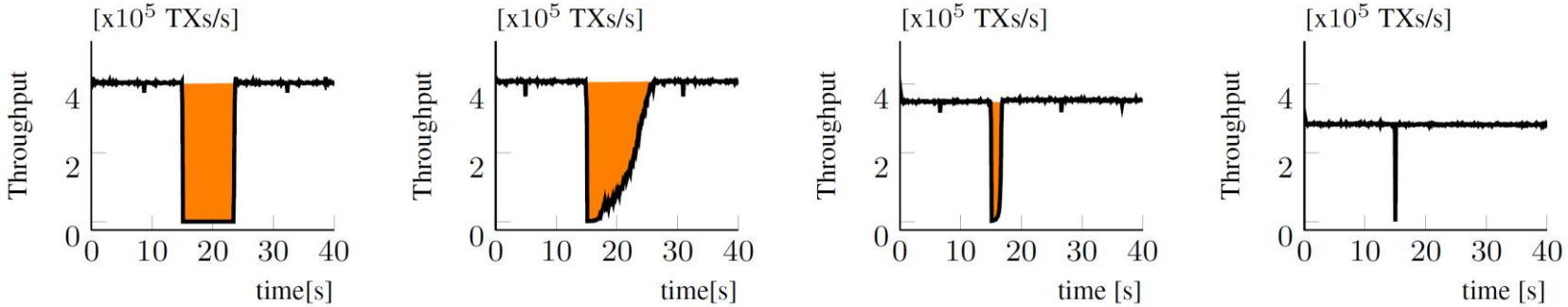
- Each executed statement is guaranteed to have persisted its changes in SCM
- The transaction array is persistent, allowing unfinished transactions at crash time to continue
- No need for system-triggered rollback



# Experiment: Recovery Performance

Dresden Database Systems Group

Different recovery schemes. TATP scale factor 500, 4 users. The database is crashed at second 15.




(a) Scenario 1

(b) Scenario 2 no indexes in SCM

(c) Scenario 2 40% indexes in SCM

(d) Scenario 3

- Scenario 1: rebuild all secondary data structures before starting answering queries.
- Scenario 2: rebuild secondary data structures in the background and start immediately answering queries using primary, persistent data. Recovery area decreased by 16%.
- Scenario 3: similar to scenario 2, with 40% persistent secondary data structures. Recovery area decreased by 82%. Throughput decreased by 14%.
- Scenario 4: all secondary data structures are persistent. Recovery area decreased by 99.8%. Throughput decreased by 30%.

# Recovery vs. Query Performance

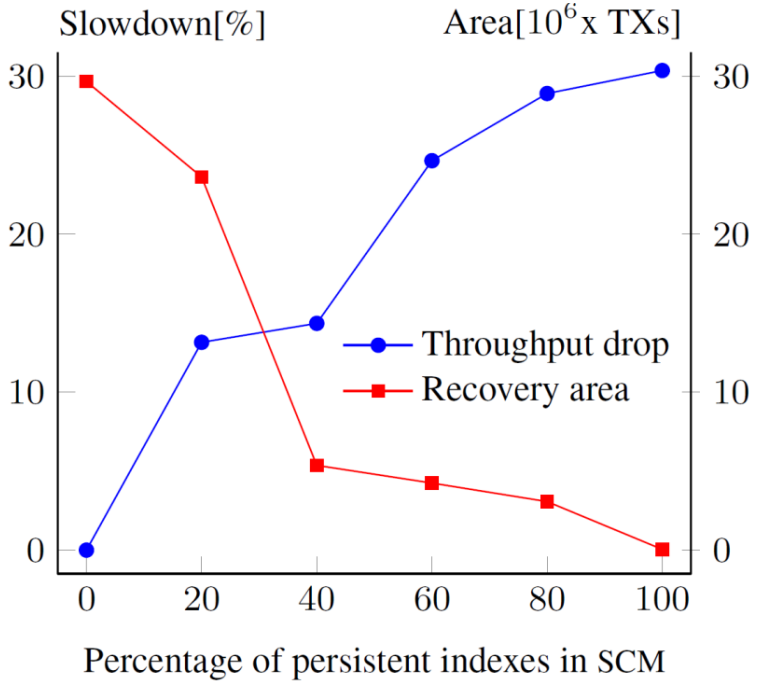
Dresden Database Systems Group

## Experiment

- Impact of persistent indexes in SCM on throughput during usual processing and recovery area. TATP scale factor 500, 4 users.

## Observation

- The more secondary data structures we persist in SCM, the smaller the recovery area and the lower the throughput during usual processing.
- The throughput drop is limited to 30%.
- Both curves are not linear: not all secondary data structures are equally important.



# Response Time during Recovery

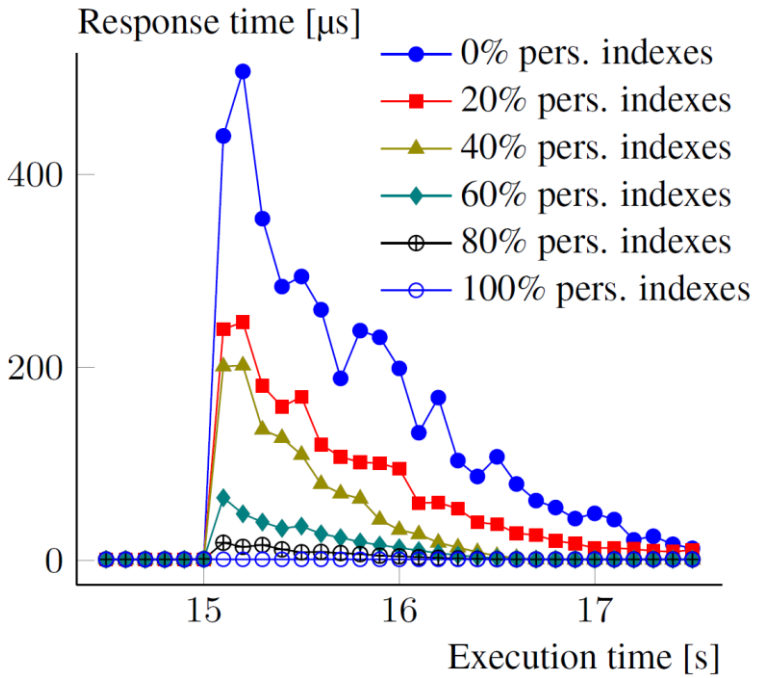
Dresden Database Systems Group

## Experiment

- average recovery response time. TATP scale factor 500, 4 users.

## Observation

- The more secondary data structures we put in SCM, the lower the average response time during recovery
- Maximum response time for the two extreme cases:
  - All in DRAM: 506 µs
  - All in SCM: 2 ms

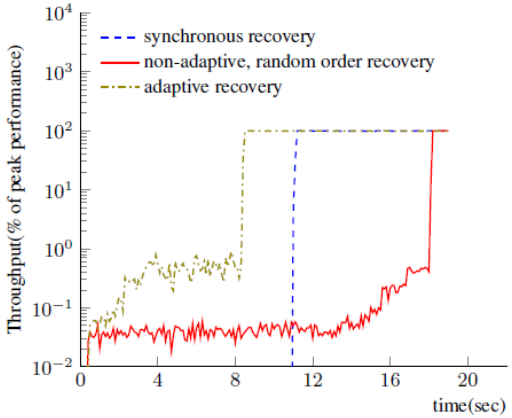


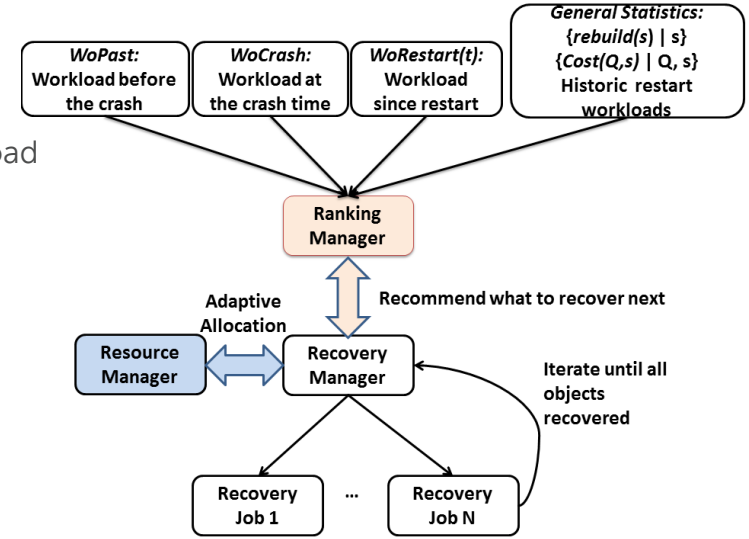
# Recovery Optimization

Dresden Database Systems Group

## Optimize the sequence of recovery objects

- Not all index structures show the same relevance for a given workload
- Workload may change due to recovery procedures




$$Benefit(WoPast, WoCrash, s) = \beta * Benefit(WoPast, s) + (1 - \beta) * Benefit(WoCrash, s)$$

$$rank(0, s) = Benefit(WoPast, WoCrash, s) - rebuild(s)$$

$$n = sizeof(WoRestart(t))$$

$$rank(t, s) = \alpha(n) * Benefit(WoPast, WoCrash, s) + (1 - \alpha(n)) * Benefit(WoRestart(t), s) - rebuild(s)$$

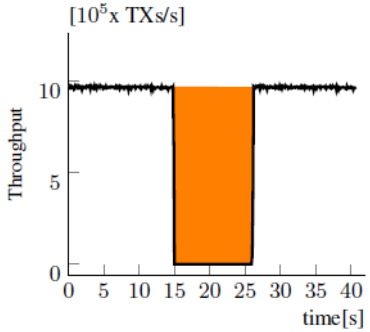
## Ranking Manager <> Recovery Manager

## Ressource Allocation for Recovery and query Processing

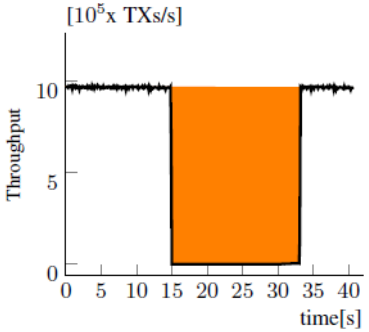
# Recovery Optimization (2)

Dresden Database Systems Group

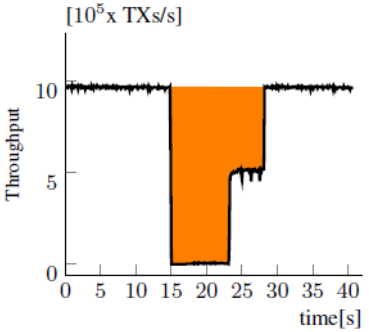
## Experimental Results



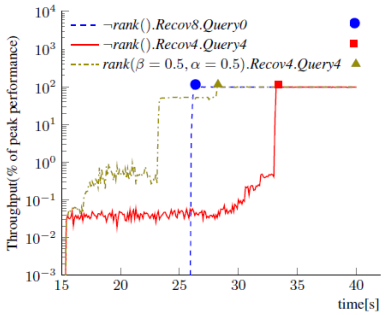

(a) $\neg rank().Recov8.Query0$



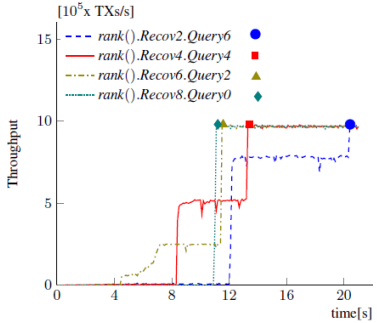

(b) $\neg rank().Recov4.Query4$



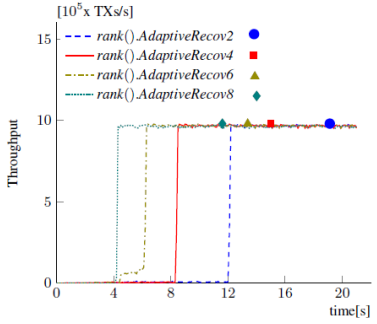

(c) $rank(\beta = 0.5, \alpha = 0.5).Recov4.Query4$



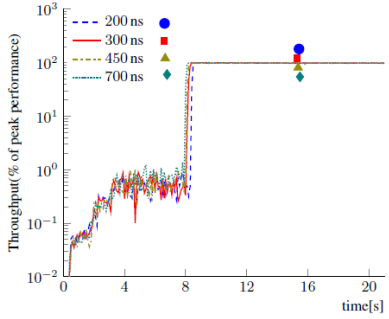

Static resource allocation




Static resource allocation



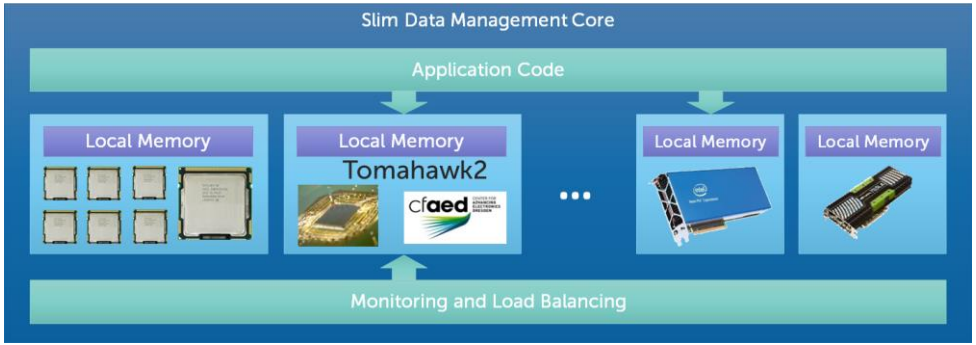

Dynamic resource allocation




Impact of SCM latency

# Summary and Conclusion

# Conclusion




- MODERN HARDWARE DEMANDS SIGNIFICANT RE-THINKING OF DB ARCHITECTURES
  - extreme NUMA Systems: „distributed system" with common address space
  - customizable processors („dark silicon"): revival of the database machine?
  - communication: optical / wireless & RMDA:
    blurring the boundaries between scale out and scale-up
  - Non-Volatile RAM: will be a game-changer!
- SLIM AND EXTENSIBLE DATA MANAGEMENT CORE TO
  - efficiently support today's applications requirements (mixed OLTP/OLAP workloads etc.)
  - embrace and exploit capabilities of modern hardware

Dresden Database Systems Group

HAEC
CRC 912
Highly Adaptive Energy-Efficient Computing

ScaDS DRESDEN LEIPZIG

DRESDEN concept

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

5G Lab Germany

Dresden Database
Systems Group

# Next-Gen Hardware for Data Management

## More a Blessing than a Curse?

Wolfgang Lehner – EDBT 2015, Brussels