Why are Disruptive Memory Technologies relevant for the SAP HANA database?

Dr. Norman May,
SAP SE

Agenda

Background on me and SAP HANA

Evolution: Challenges on Large NUMA Systems

- SAP HANA NUMA-aware task scheduling in a Nutshell
- Problem 1: Scalability on MANY core machines
- Problem 2: Performance of initial memory allocations
- Problem 3: SAP HANA Crash performance
- Problem 4: DRAM is not reliable

Evolution: Persistent Memory in SAP HANA

- Byte-addressable, but still adoption needed
- Problem: hyperscaler support, PMEM roadmap discontinued

Future: Use Cases for Compute Express Link

I am a Database Guy

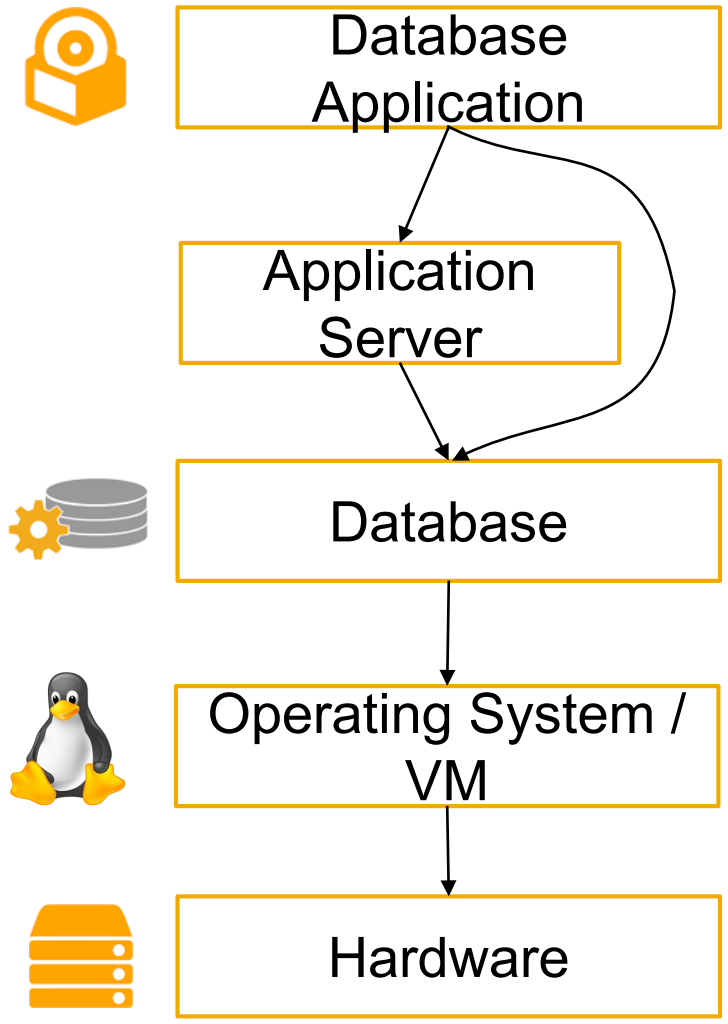
Chief Architect for the SAP HANA database core

Focus on workload management (CPU, memory) and query processing

Involved with hardware certification of SAP HANA

Adviser for PhD student in SAP HANA Campus

System Architecture Perspective



SAPUI5 SAP ERP SAP BW on HANA tableau SOFTWARE SAP Data Warehouse Cloud

SAP NetWeaver node.js JBoss

SAP HANA

SUSE vmware docker

redhat IBM intel

sgi Hewlett Packard Enterprise Lenovo

The „Genesis“ of SAP HANA as In-memory Database

SAP HANA (aka NewDB) pioneering …

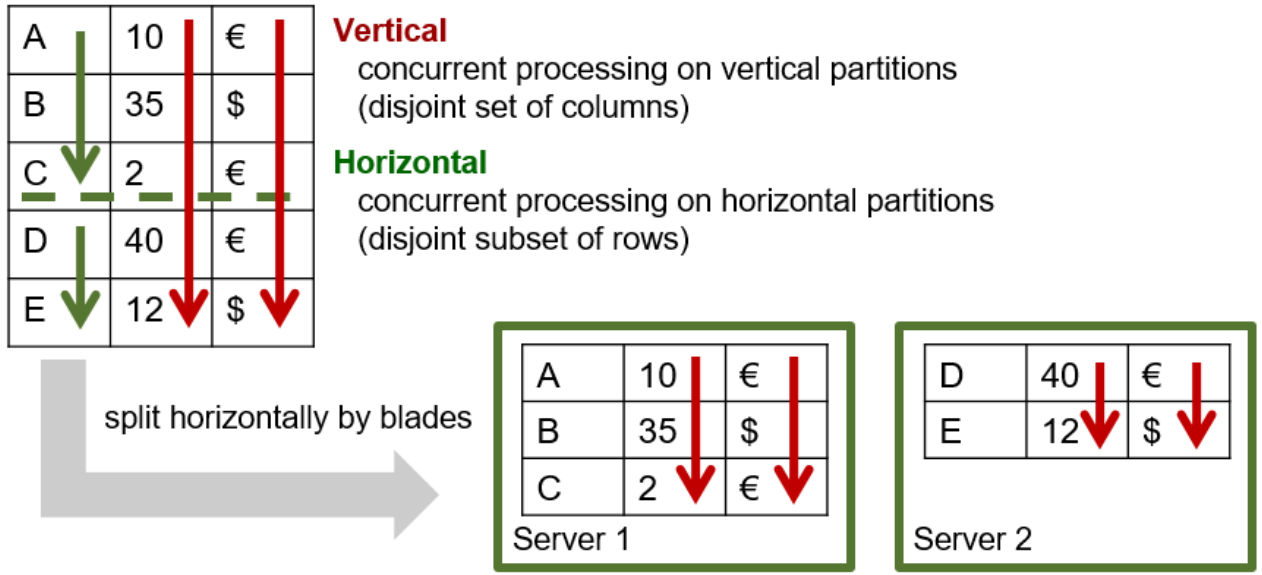
- query processing on multi-core
- Vectorized query processing
- In-memory processing in a column store
- Hybrid analytical and transactional data management (HTAP)a

Challenges

- At startup:
  - (Almost) all data must be loaded – terabytes
  - Large amounts of memory allocated (~50%)
- Performance of queries suffers until data loaded
  - No results until all required tables loaded
- Single-process architecture
  - Restarts very costly

SAP NewDB Technology

Exploit multi-core architectures by parallelization of operations



Evolution: Challenges on Large NUMA Systems

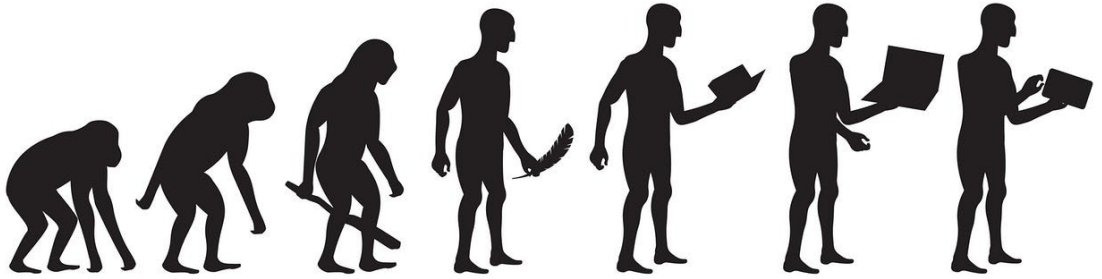
SAP HANA NUMA-aware task scheduling in a Nutshell

Problem 1: Scalability on MANY core machines

Problem 2: Performance of initial memory allocations

Problem 3: Process destruction time (SAP HANA crash performance)

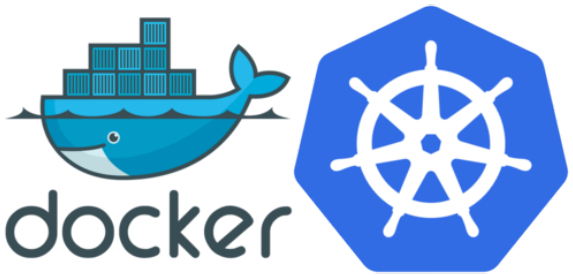
Problem 4: DRAM is not reliable



Scaling up SAP HANA

Small
- 32GB-4TB DRAM
- 1-4 CPU sockets (Intel Xeon)
- 2 (v)CPU cores to >50 physical cores per socket
- A **single process** using all resources

Typical SAP HANA systems

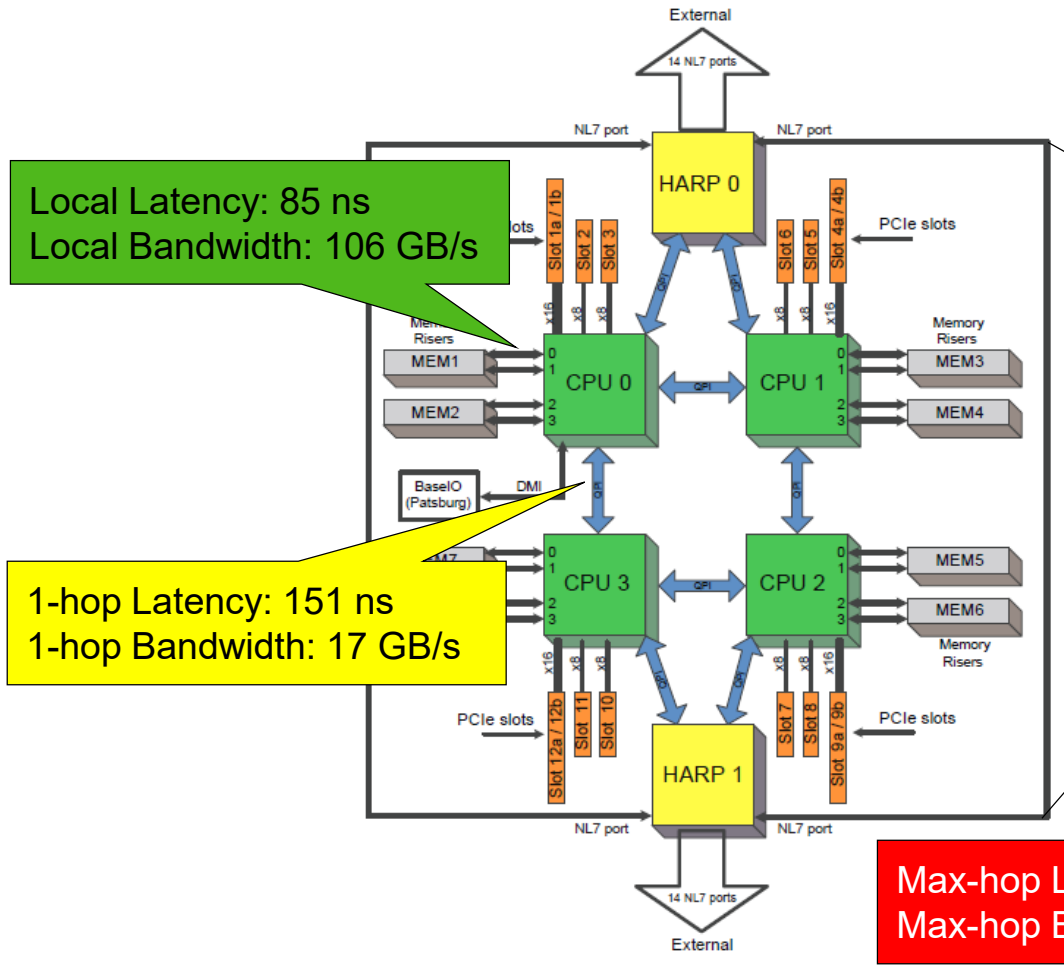


Large
- 12-48TB DRAM (more planned)
- 8-32 CPU sockets (Intel Xeon, IBM Power)
- up to ~1792 logical cores
- Still a **single process** using all resources

Real challenge

NUMA Topology of a 32-Socket System
(NUMA = Non-Uniform Memory Access)

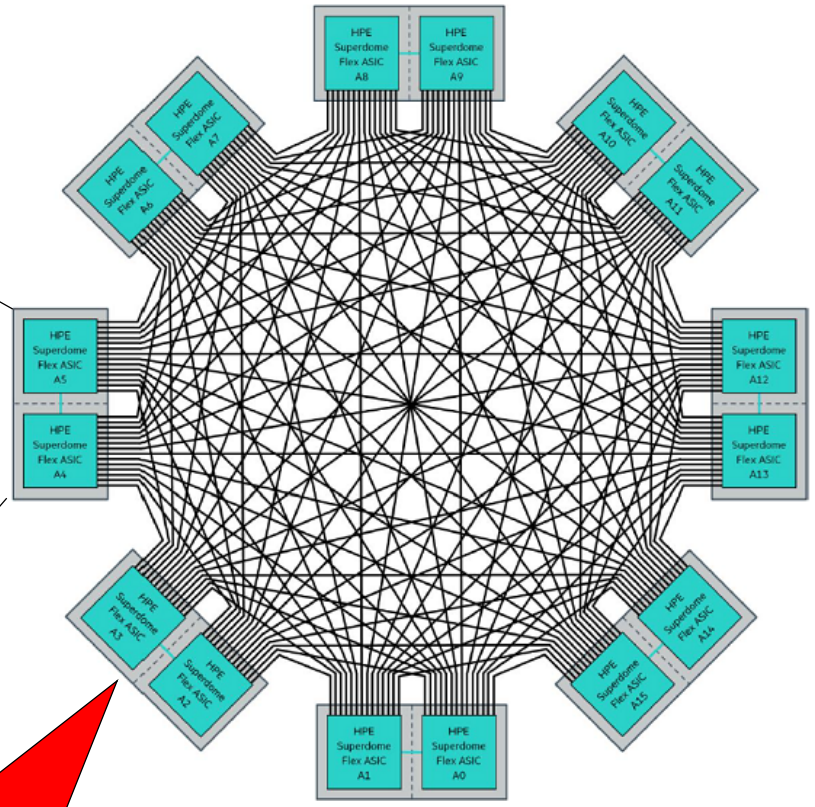
1 Chassis



Local Latency: 85 ns
Local Bandwidth: 106 GB/s

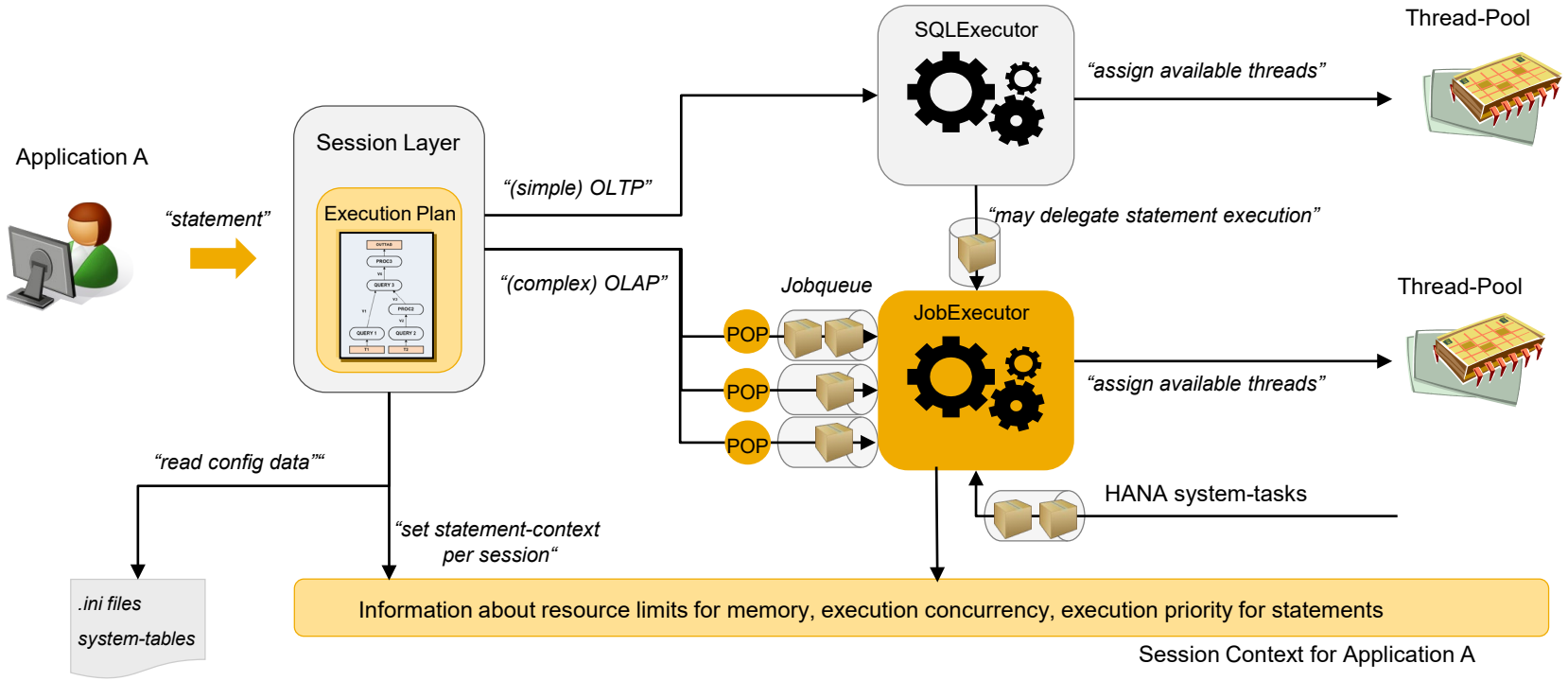
1-hop Latency: 151 ns
1-hop Bandwidth: 17 GB/s

HPE NUMA link topology



Max-hop Latency: 360 ns
Max-hop Bandwidth: 14 GB/s

Understanding NUMA-aware Statement Execution in SAP HANA



Custom-built memory management

NUMA-aware data placement:

- Table data allocated on certain NUMA node
- Hash-based default or manual via DDL

NUMA-aware task scheduling:

- Table access operations scheduled where the data is allocated
- Job queues per NUMA node
- Job worker threads can be bound to NUMA nodes
- Task stealing to balance NUMA node contention

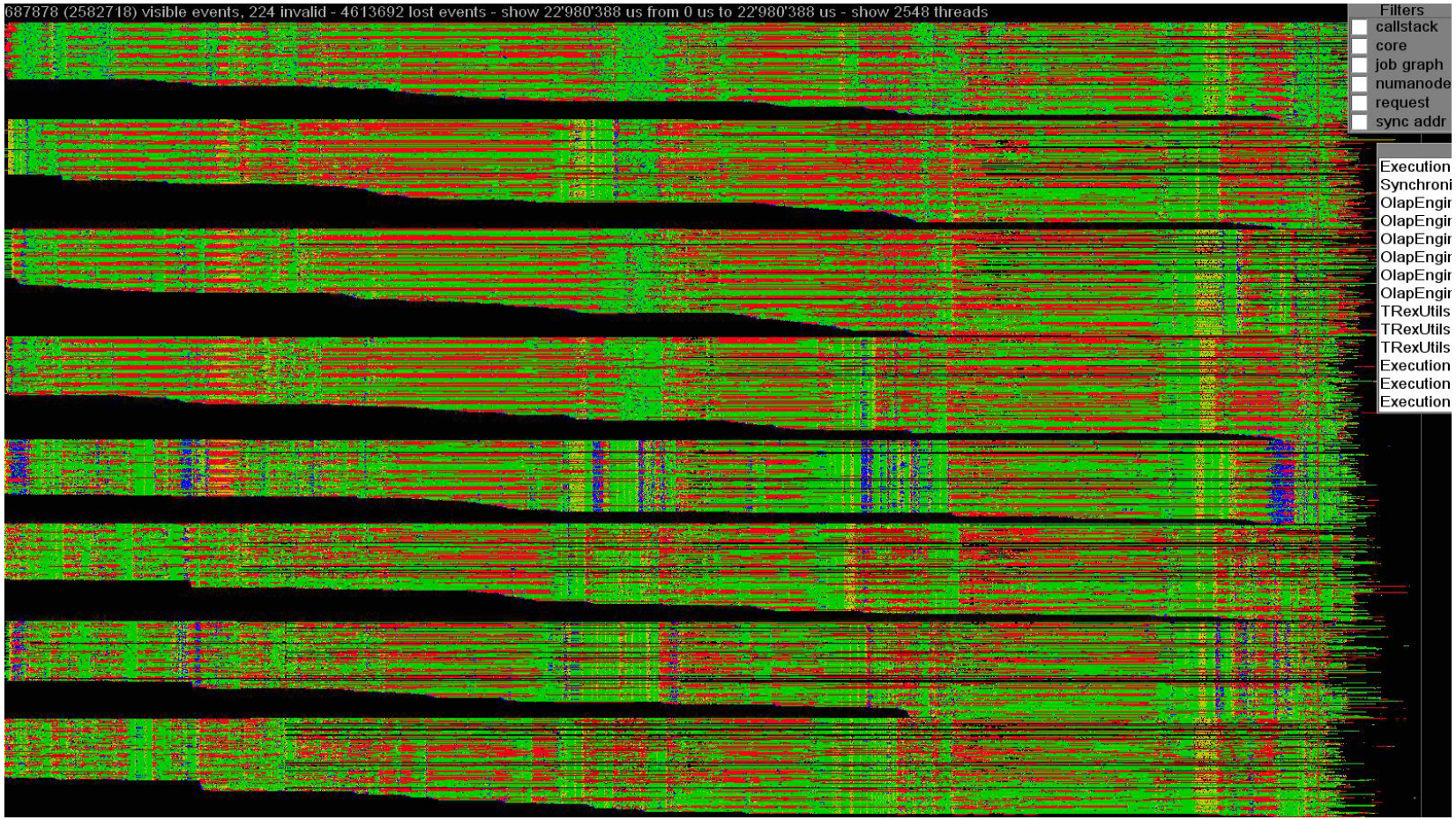
SAP-internal benchmark on PPC (608 cores on 8 sockets)
before optimization

Red: Locking, Waiting

- In SAP HANA leads to thread creation (staircase)

Blue: Task scheduling overheads

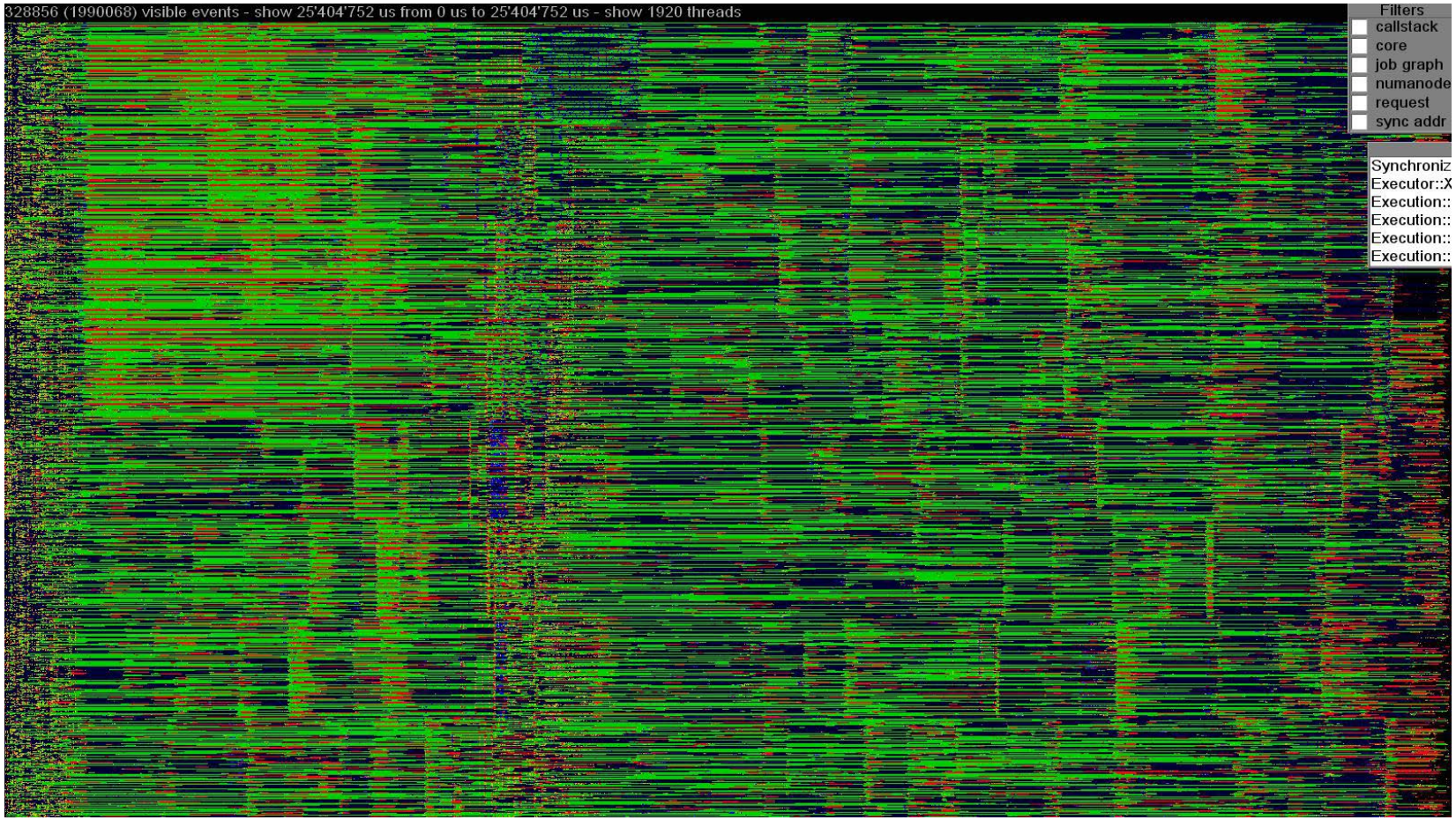
Green: threads perform useful work



SAP-internal benchmark on PPC (608 cores on 8 sockets)
after optimization

Reduced locking
→ fewer threads

More useful work performed



Problem 1: Scalability on MANY core machines

We meet the same scalability issues whenever number of cores increases:

- Latches
- Data structures like B-tree
- Operations like metadata lookup

Question: How to evolve a database in an evolutionary way to avoid optimization cycles every 2-3 years?

- Can we avoid a fundamental rewrite of the database engine?
- Identify data structures and algorithms with good performance and scalability trade-off
- Choice of locking primitives?
- Scalability and supportability of lock-free data structures and hardware-transactional memory?

Problem 2: Initial Allocation (1/2)

Simulation experiment:

- Allocate and touch memory in *n* threads in parallel:

```
allocated = 0;
while (allocated < SIZE) {
        ptr = mmap(0, BLOCKSIZE, …, FLAGS, …);
        touch_every_cacheline(ptr, BLOCKSIZE);
        touch_every_cacheline(ptr, BLOCKSIZE);
        blocks.push_back(ptr);
        allocated += BLOCKSIZE;
}
```

- Deallocate memory (optional):

```
for (ptr : blocks)
        munmap(ptr, BLOCKSIZE);
```

mmap() flags:

- `MAP_PRIVATE | MAP_ANONYMOUS`
- `MAP_PRIVATE | MAP_ANONYMOUS | MAP_POPULATE`
- The above with `MAP_HUGE_2MB`

Block sizes:

- 1 to 256K pages (4KB-1GB), powers of two

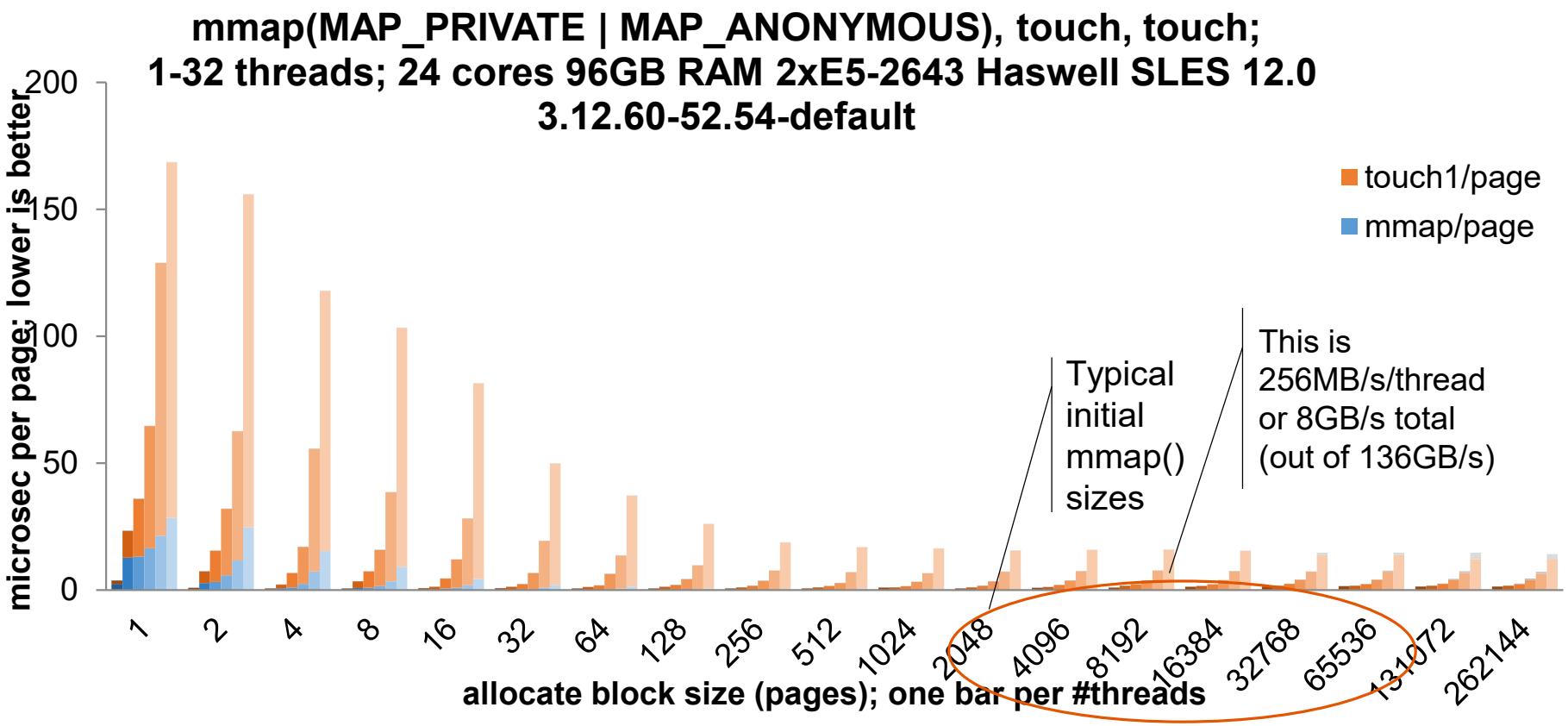
Thread counts:

- 1-32, powers of two

Test Systems:

- 96GB DDR4-2133 DRAM, 2 sockets, 24 log. cores
- 16TB RAM, 32 sockets, 1152 logical cores

Problem 2: Initial Allocation (2/2): On-Demand Allocation



On-demand allocation: Scalability problems

- In mmap() itself
- In page fault

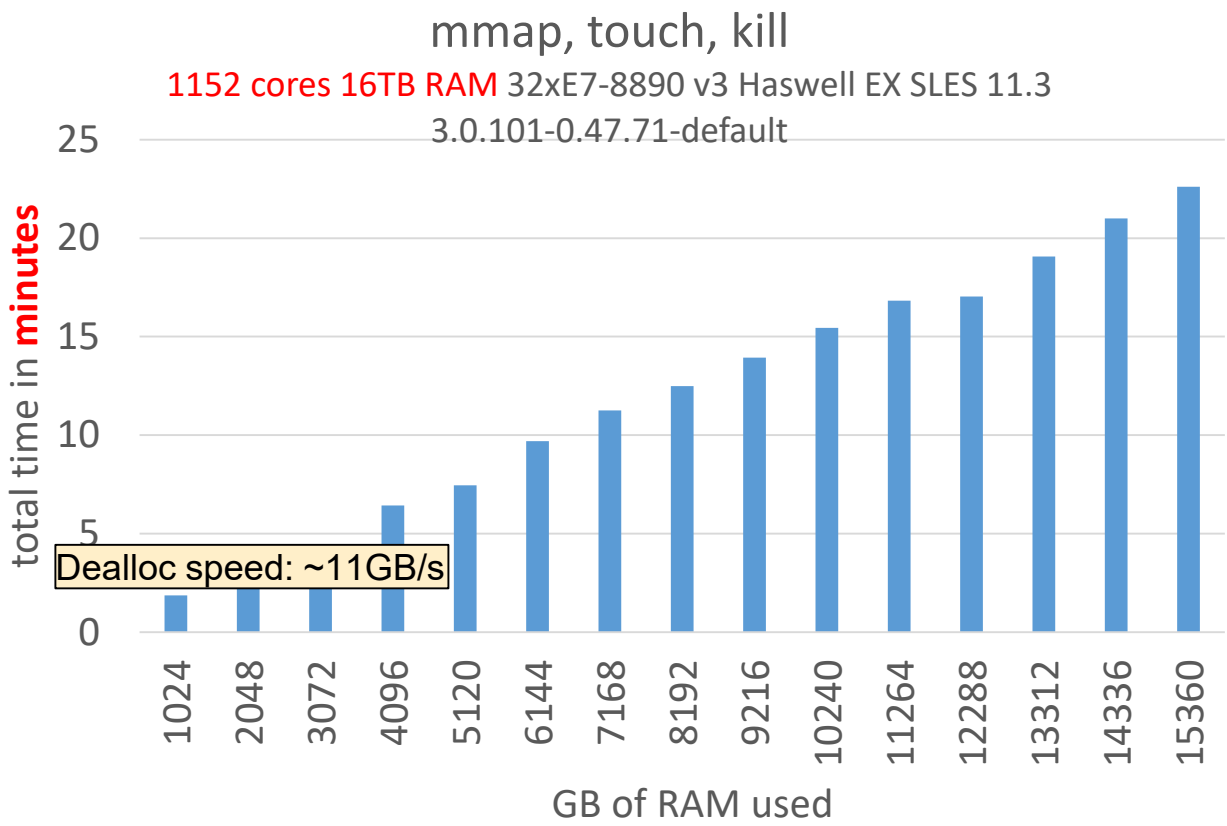
Problem 3: Process Destruction Time (1/2)

Simple Experiment:

```
int child_pid = fork();
if (child_pid) {          // parent
          wait_for_signal_from_child();
          ts1 = timestamp();
          kill(child_pid, SIGKILL);
          waitpid(child_pid, NULL, 0);
          ts2 = timestamp();
          print_time(ts2 - ts2);
} else {  // child
          ptr = mmap(n * GB);
          touch(ptr, n * GB);
          send_signal_to_parent();
          while (true) pause();
}
```

Observation:

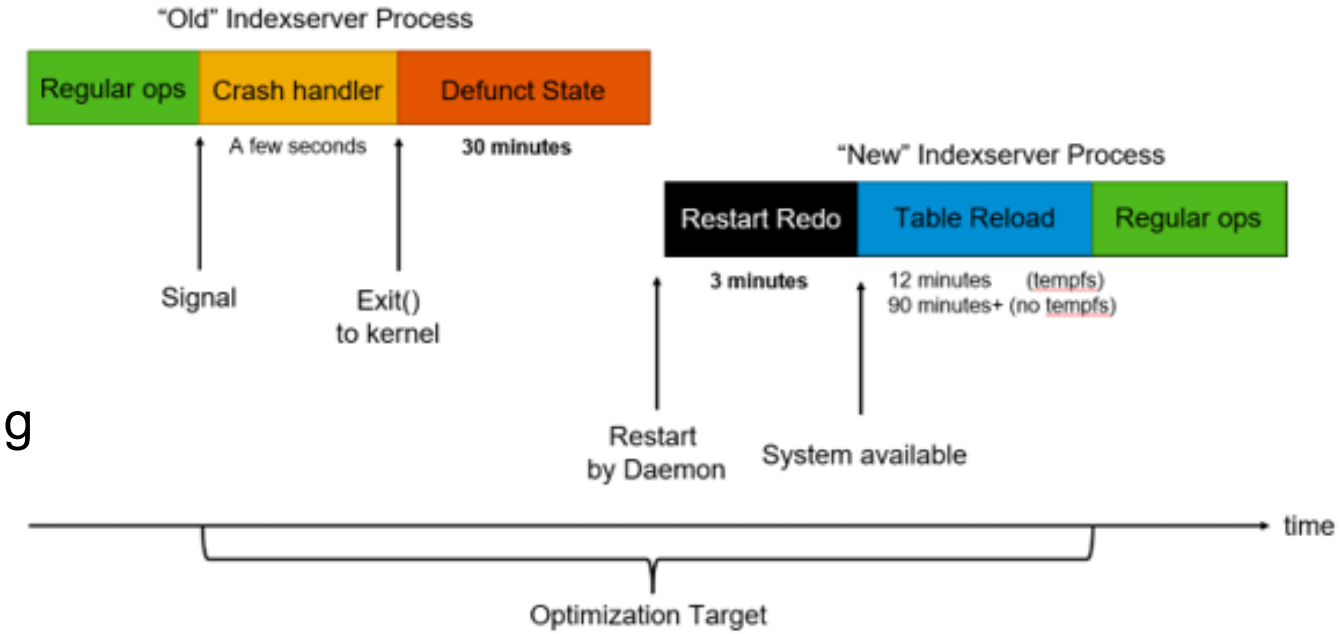
- Process exit time linear to allocated memory
- Processed on single core
- 2MB hugepages speed up ~200x, but not used in SAP HANA



Problem 3: Process Destruction Time (2/2)

System downtime induced by crashes & restart

- Cluster managers don't help usually
- Failover not even automated in some cases
- Cloud: False positives / negatives due to ping service



Restart time dominated by

- Memory deallocation in defunct state of old process because of locks in the Linux kernel
- Loading data into memory & memory allocation

- Parallel memory deallocation in separate process virtually removes the defunct-state issue
- Selective pre-load of tables relevant for application improves perceived availability

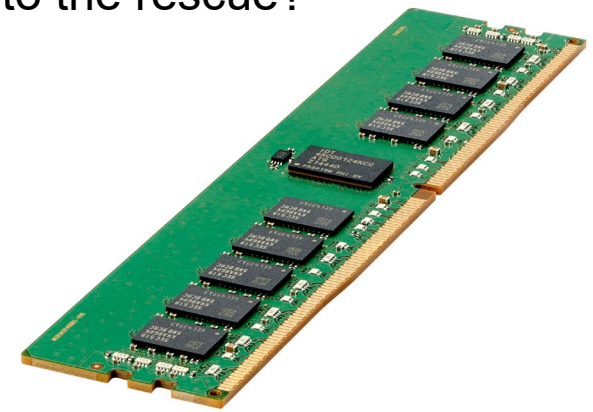
Problem 4: Many DRAM DIMMs Fail

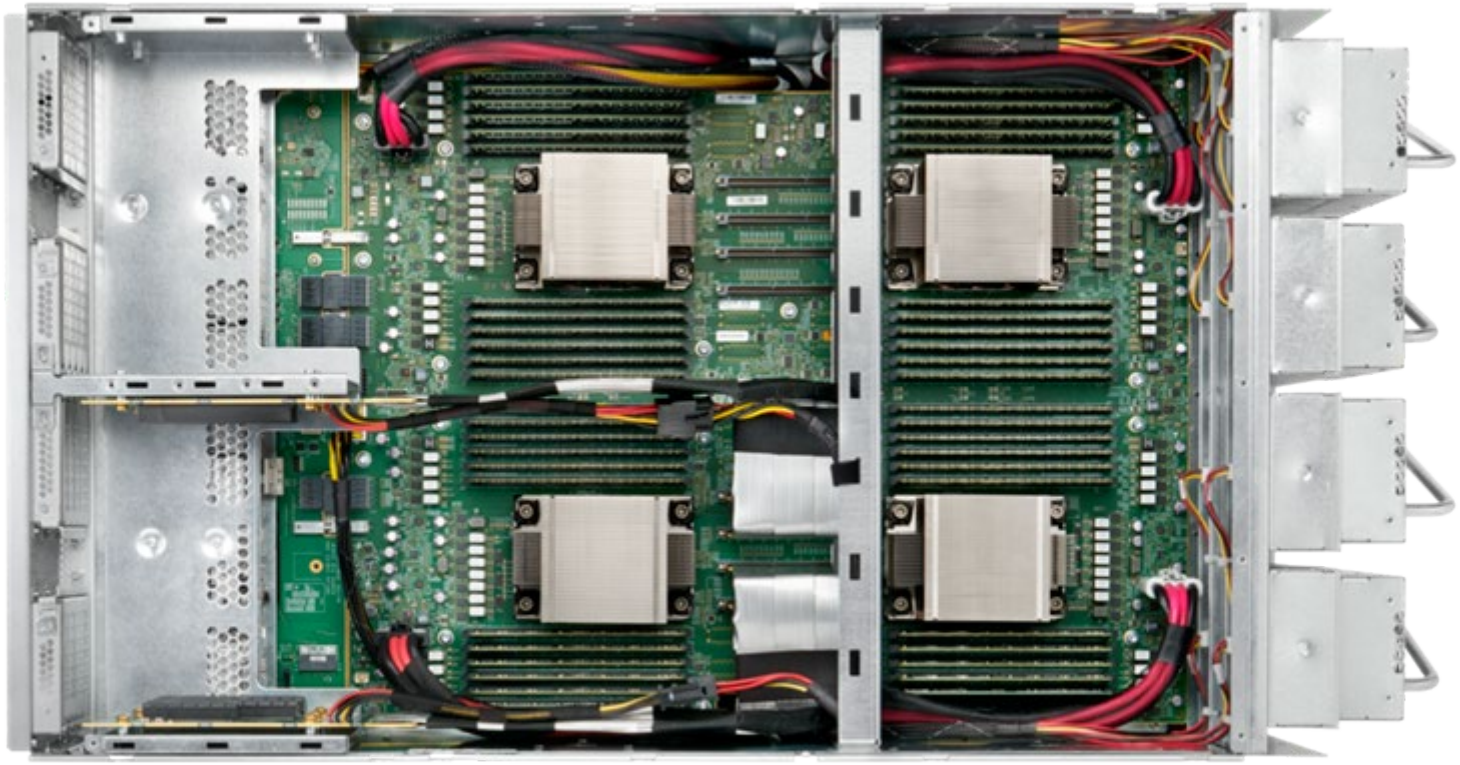
Do the math:

- 32 sockets x 12 DRAM banks
  = 384 DRAM DIMMs
- 384 DRAM SIMMs x 10 chips
  = 3840 chips
- Chances increase for DIMM failures

➔DRAM failures lead to crashes

➔RAS features to the rescue?



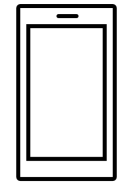


Summary: Memory Management Pain Points

Memory management in Linux does not scale to large processes:

- Virtual memory allocation does not scale
- Reallocating memory by scraping up free chunks causes fragmentation in mappings

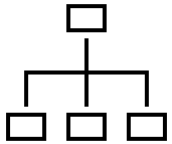
→Performance problems seem to be related to process address space/page table locks
→SAP HANA is not the main target for Linux



Performance tuning on large scale-up systems is painful:

- Resolving lock contention is very tedious

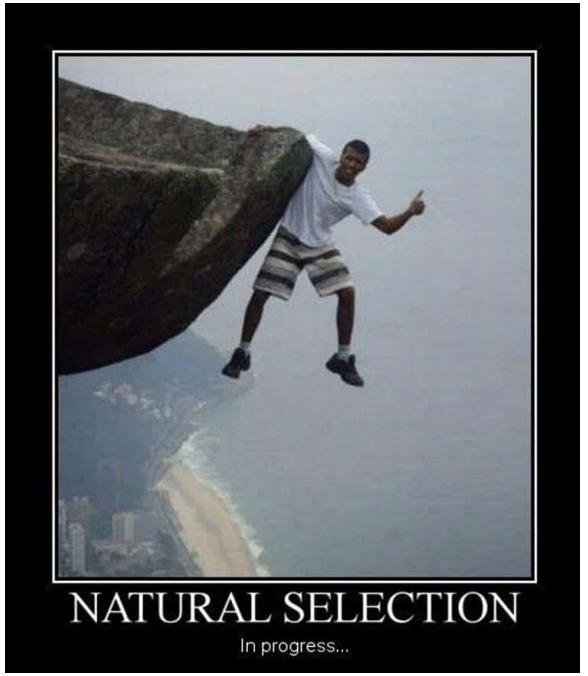
→What are good programming abstractions?



Evolution: Persistent Memory in SAP HANA

Byte-addressable, but adoption needed

Problem: hyperscaler support, PMEM roadmap discontinued

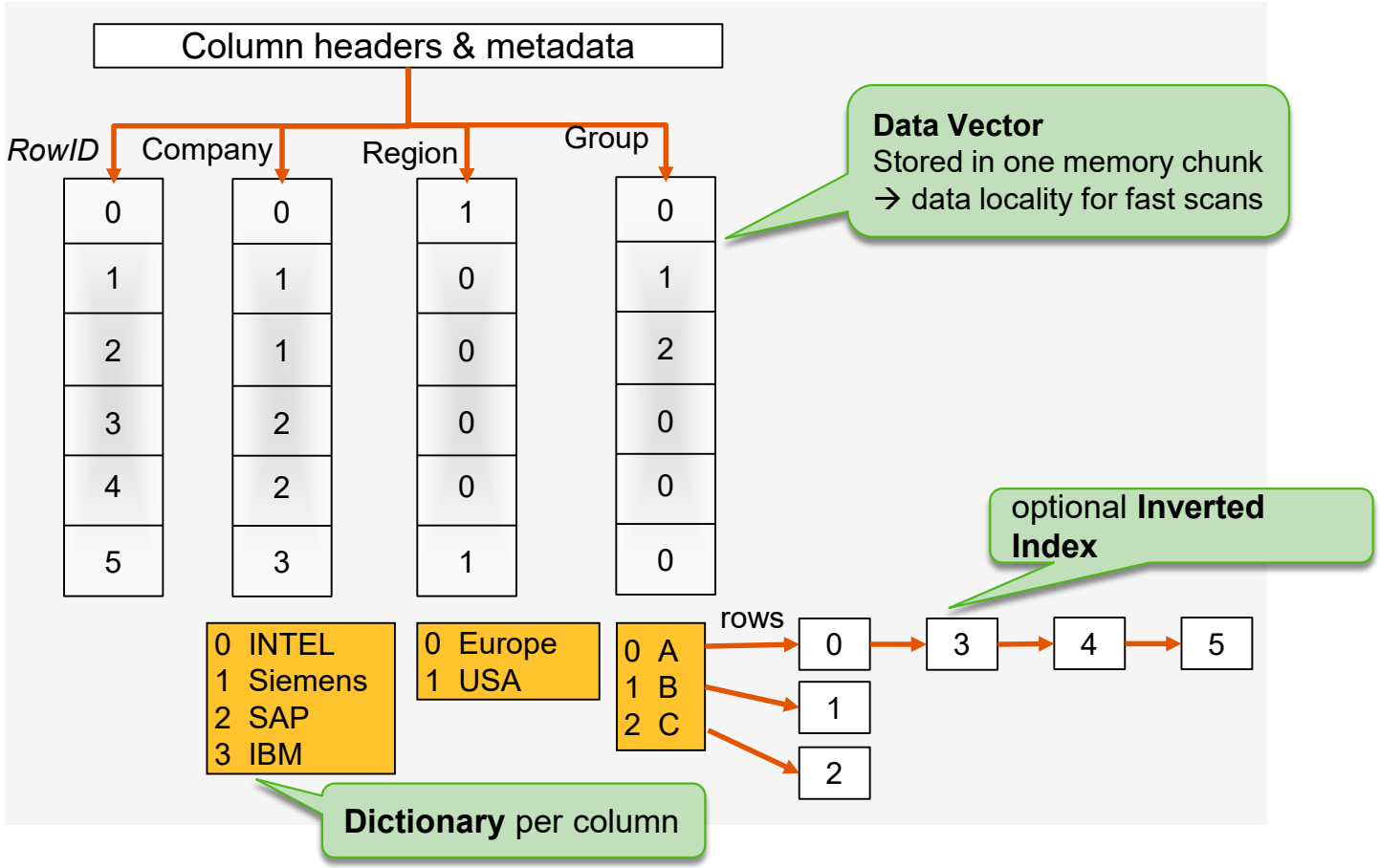


Adopting In-Memory Column Store to Persistent Memory

Row-oriented table „Company“

Company [CHAR50]	Region [CHAR30]	Group [CHAR5]
INTEL	USA	A
Siemens	Europe	B
Siemens	Europe	C
SAP	Europe	A
SAP	Europe	A
IBM	USA	A

Column-oriented table „Company“



➔ Need to replace pointers in DRAM by offsets …

Design Decisions for PMEM in SAP HANA

HANA uses PMEM for the large read-optimized main fragment, i.e.

- The write-optimized delta fragment remains in DRAM
- DRAM is used for transient data, e.g. for query processing

HANA still writes the table data to disk

- because used by disk-based replication tools
- as additional layer of reliability
- At restart no loading from disk needed, just follow the offsets in PMEM → faster restart times

HANA uses dedicated allocators for managing PMEM

- PMEM „base path“ assigned to PMEM devices using DAX-fs – usually per socket
- Keep common byte-addressable interface for memory-based data structures
- Careful programming needed, e.g. memory barriers 1st generation. Extended asynchronous DRAM refresh (eADR) with Intel Optane PMem 200 series handles flushing caches.

Problem: Hyperscaler support, PMEM roadmap discontinued

PMEM was not adopted by hyperscalers

- No good match for Docker-based environments because PMEM sticks with the machine …
- Speculation: price strategy by Intel hindered broader adoption

PMEM to be discontinued

- Intel Optane 300 series canceled, i.e. for SaphireRapids and later need to find alternative …
- Intel Optane 200 series and earlier still supported by Intel and SAP on Ice Lake / CooperLake an earlier for SAP HANA on-premise

The investments were not in vein: pointer-less main fragment

- Used in SAP HANA for „fast restart option“ where the data is stored in shared memory
- Useful for use cases with memory pools in CXL – see later

Future: Compute Express Link

How to leverage PMEM investments?

Source: Samsung

Opportunities for CXL-attached memory

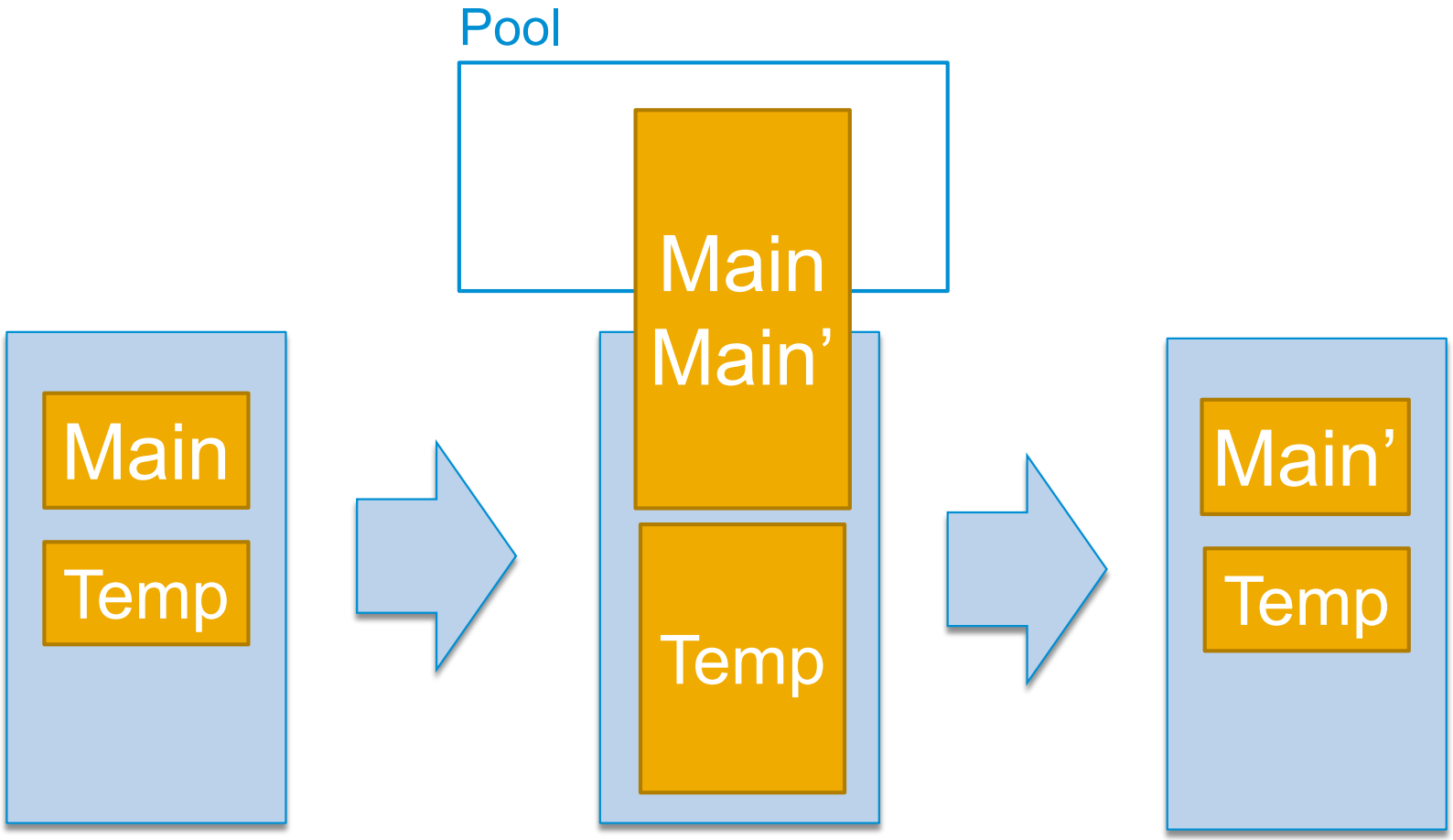
Broad interest by hyperscalers and HW vendors ☺

Mainly interested in memory pools (CXL 2.0) – see use cases

- Open question: Linux APIs for (de-) registering memory from the pool

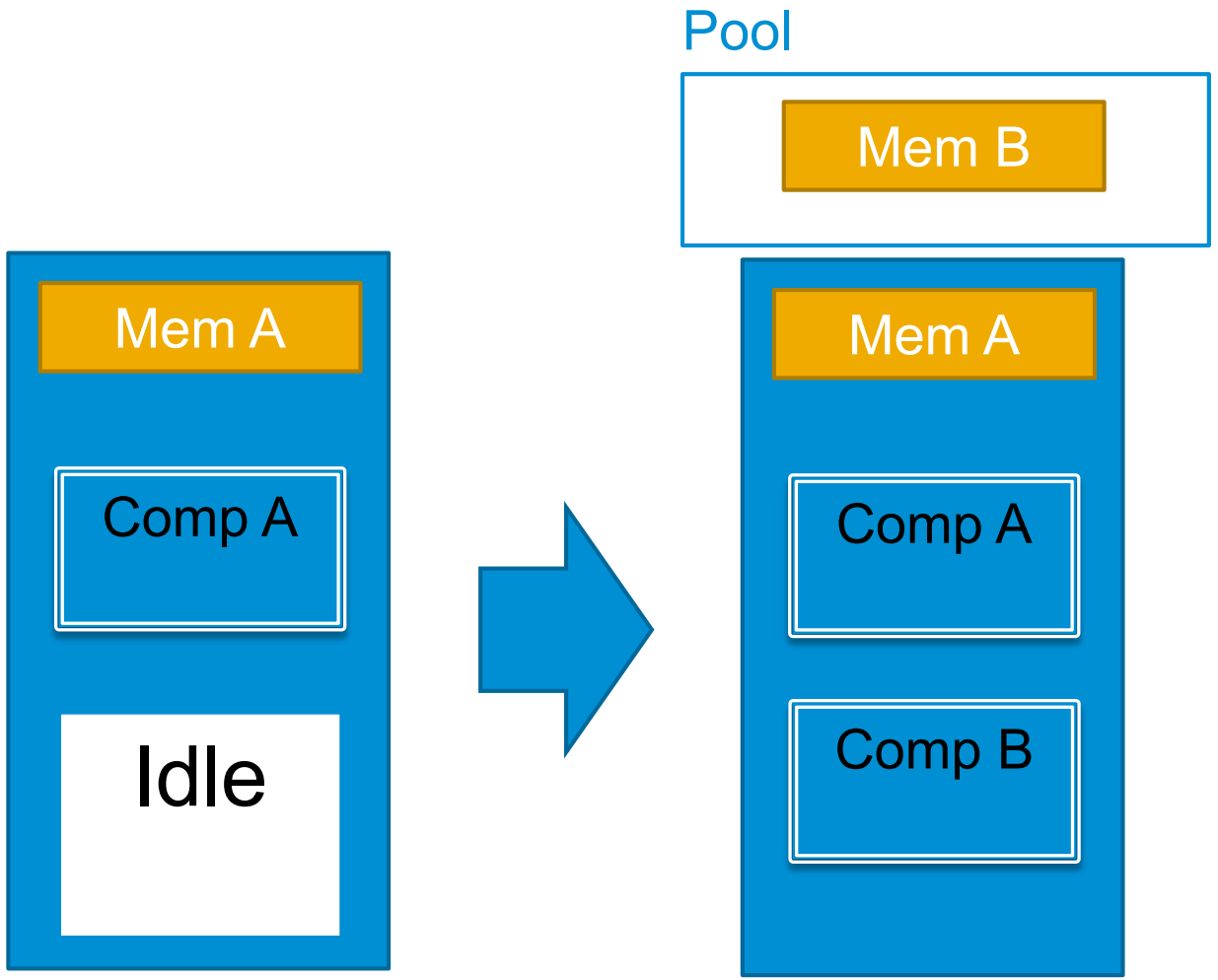
Investments for PMEM can be leveraged

CXL Scenario 1: “Upgrade”



Temporarily use more memory from a pool during software upgrade, e.g. S/4 HANA

CXL Scenario 2: “Use idle compute”



Comp A could be a SAP HANA indexserver

The idle compute capacity could be used by an Elastic Read Node, i.e. another SAP HANA process.

The additional memory required is provided from a memory pool, e.g. binding the memory of Comp B to the CXL-based memory pool

Performance Evaluation

System Configuration & Single-Node Test Configurations

System Configuration

- CPU: Sapphire Rapids C0 stepping
- Host DRAM: 256 GB per socket
- Memory BW: ~200 GB/s (3200 MHz * 8 channels)
- DAX using PMEM feature:
  - No additional overhead when reading the main storage

Workloads

- TPC-C (OLTP): 100 warehouses
- TPC-DS (OLAP): Scale factor = 100

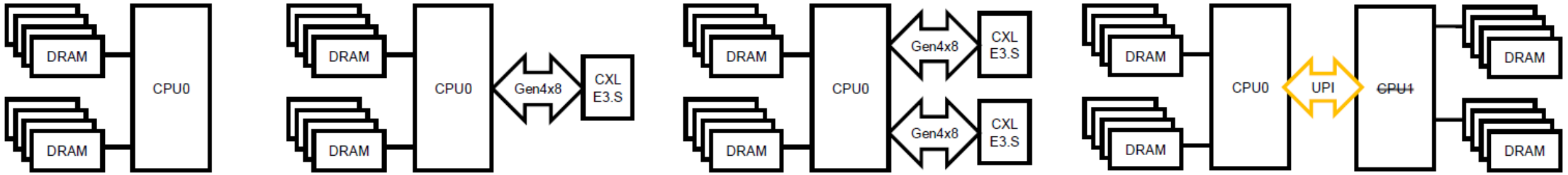
Single-Node Test Configurations

(a) Baseline
No CXL memory expansion
Main storage in the host DRAM

(b) 1CXL
1 CXL memory
Main storage in CXL Memory

(c) 2CXL
2 CXL memory
Main storage in CXL Memory

(d) CXL Emulation
Affinity to CPU0 only
Main storage in the remote socket



Performance Evaluation

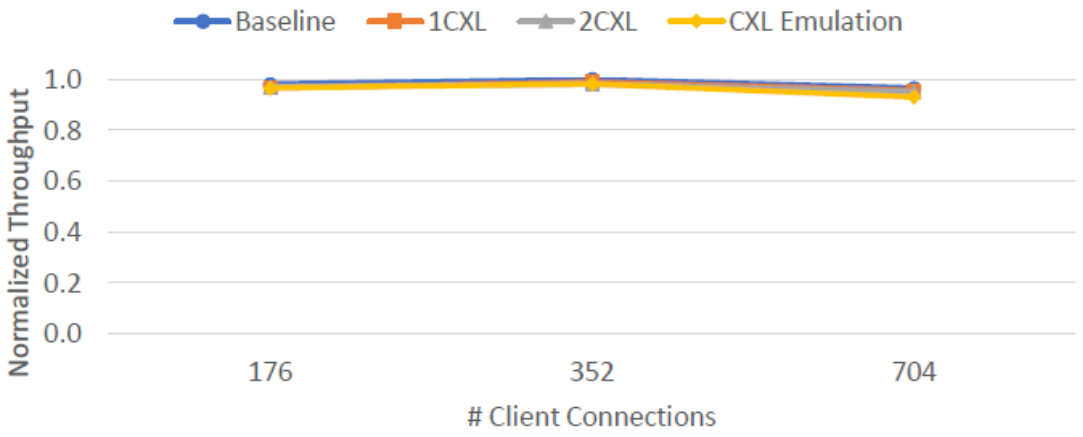
Single-Node Test Results

TPC-C

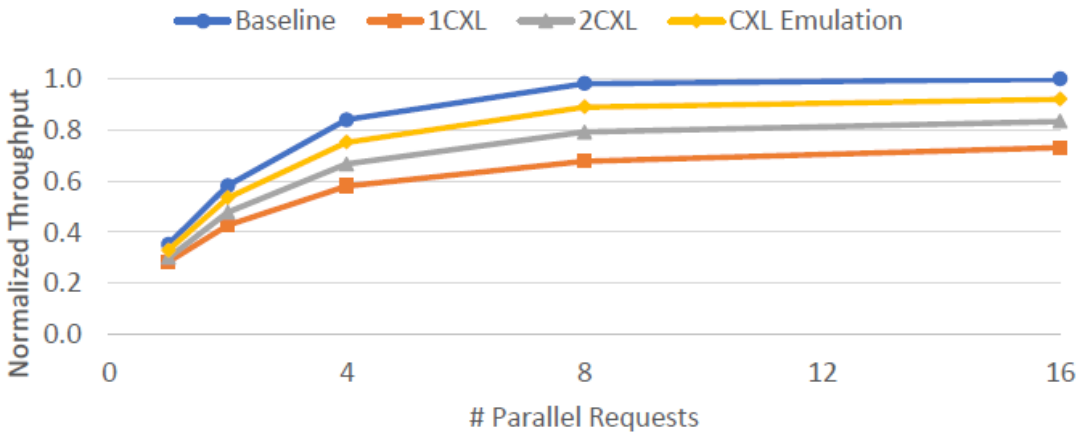
- No significant perf drop (~1%)
- Low memory bandwidth bound

TPC-DS

- Performance degradation
  – 1CXL: 27%, 2CXL: 18%
- CXL emulation in SPR: 8%
  – UPI: ~64 GB/s
- Speculation in ASIC implementation: ~8%
  – PCIe Gen5x16: Theoretically 64 GB/s



(a) TPC-C



(b) TPC-DS

Conclusion: Lot‘s of open questions on OS level

Effective task scheduling on large machines

Scalable synchronization primitives

How to „fix“ Linux memory management

- Scalable memory allocation and de-allocation
- Bulk operations, e.g. for memory remapping

Life beyond Intel Optane persistent memory

- Memory pools with Compute Express Link (CXL)
  - Coherent OS APIs
  - Feasible database use cases

Norman May
norman.may@sap.com
https://www.linkedin.com/in/normanmay/