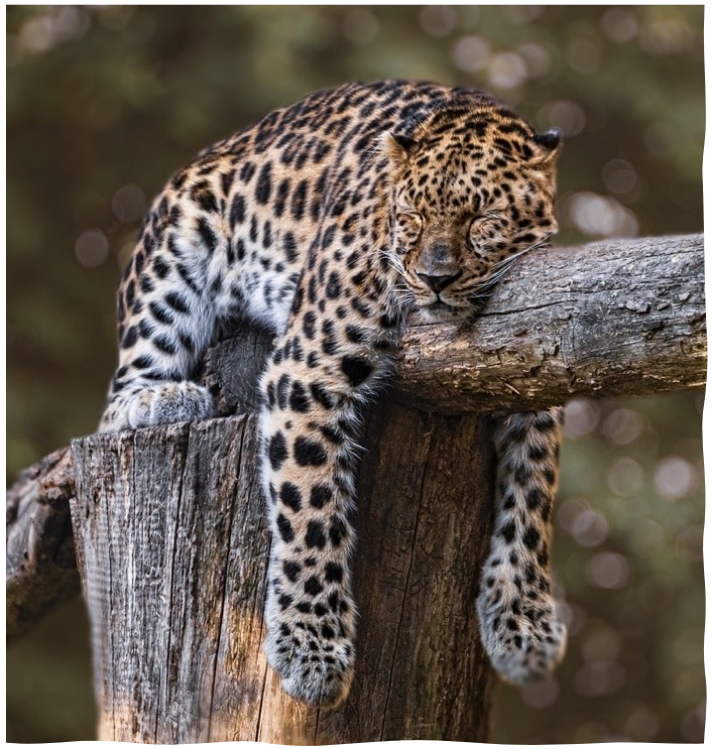
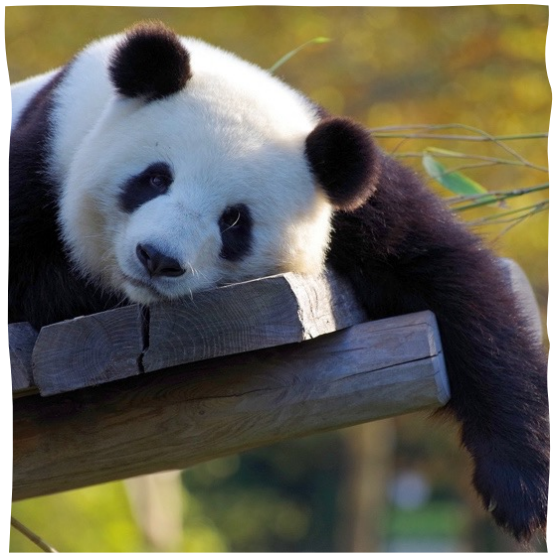




How to tame the Memory Zoo in Database Systems?

Kai-Uwe Sattler

DBIS ▪ TU Ilmenau

www.tu-ilmenau.de/dbis

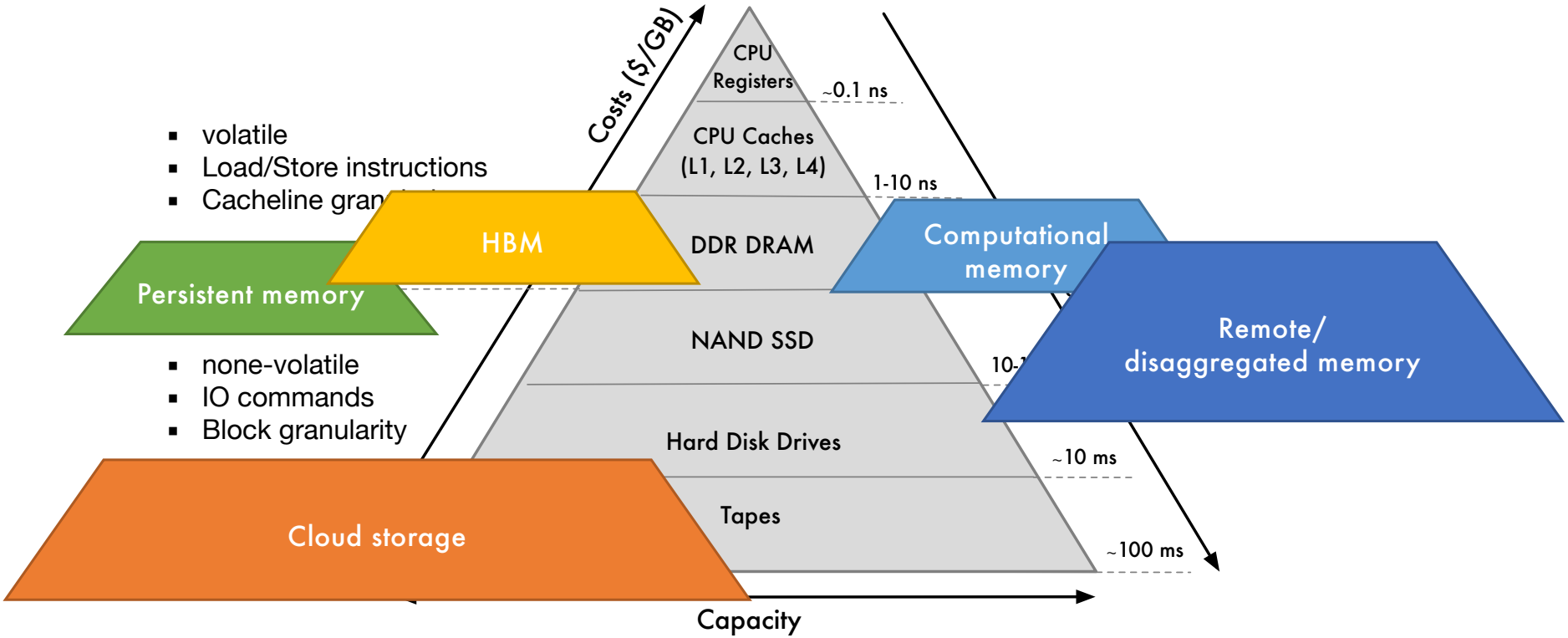
Once upon a time …

- databases were much larger than the available memory
- disk was much slower than memory (by a factor of 10^5)
- there was a memory hierarchy (cache - memory - disk - tape)
- we used a buffer pool to overcome gap in access times
- DB knows better than OS

But today …

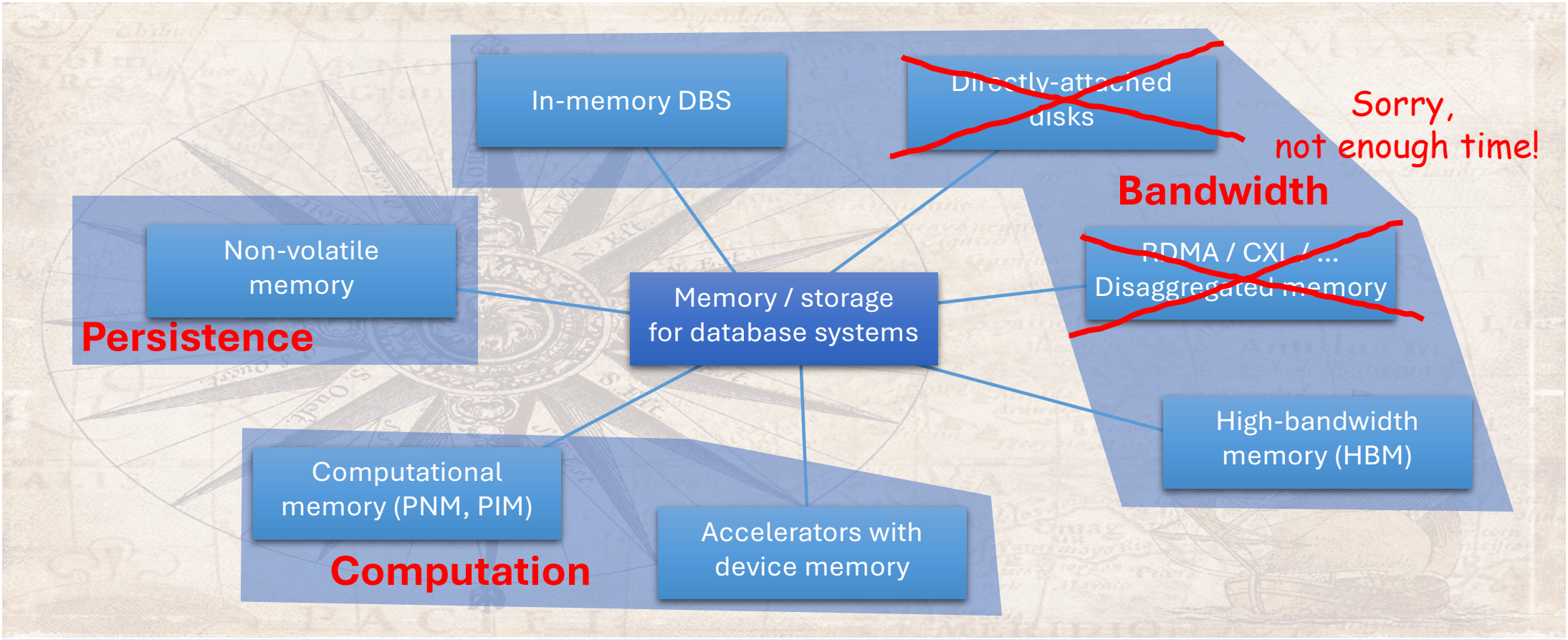
- memory in TB range
- directly-attached NVMe SSDs with 3 GB/s; as array more than 20 GB/s
- persistent, byte-addressable memory
- computational memory/storage
- GPU/FPGA-based accelerators with device memory
- disaggregated memory

From a Hierarchy to a Zoo…



Based on: Steve Scarcall: Programming Persistent Memory, Apress Open, pmem.io/book

The Memory Landscape for Databases

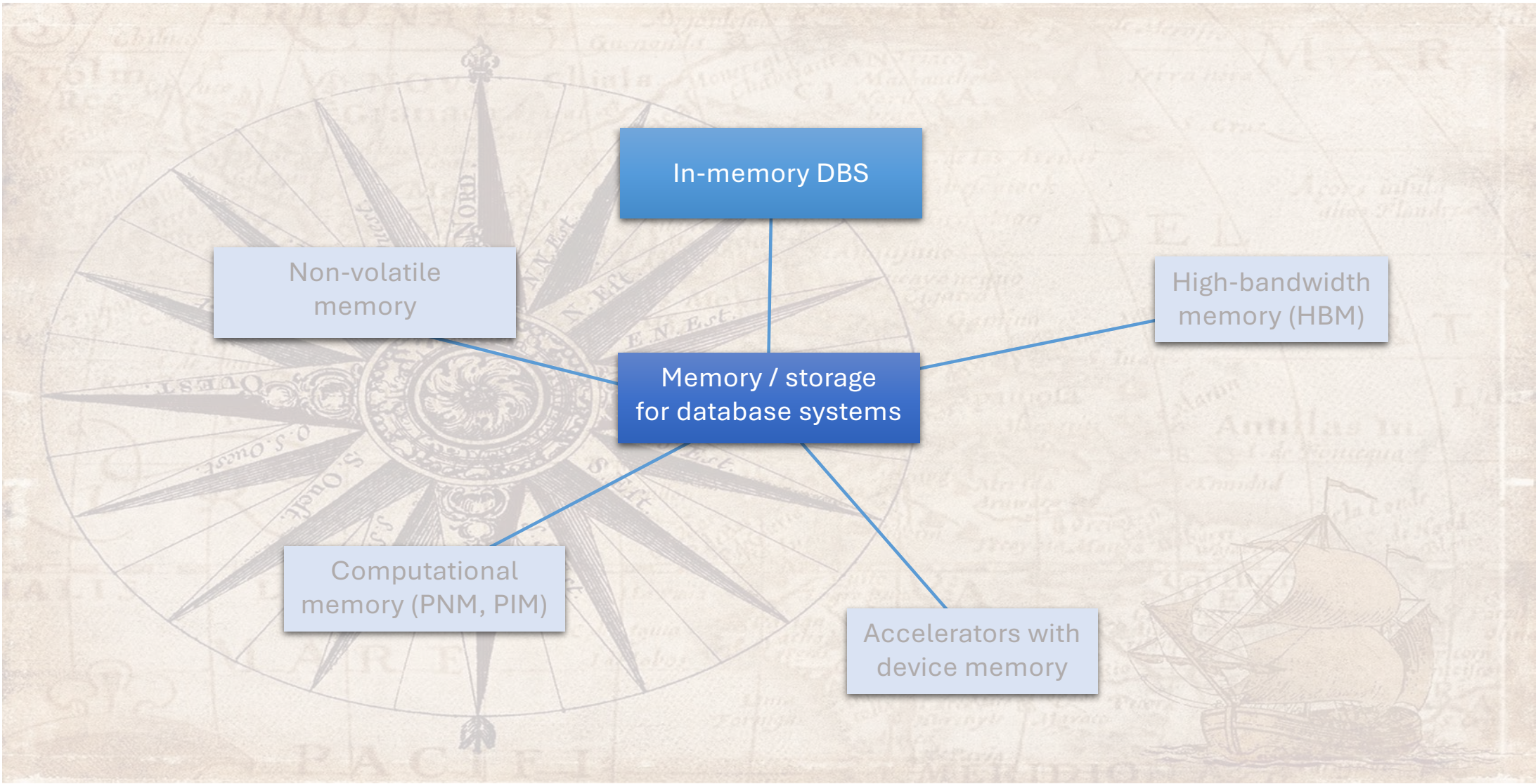


Questions for Database System Designers

- Which memory technology should we use?
- Where to place data? Or computation?
- How/when to transfer data?
- Which is the optimal data structure?
- …

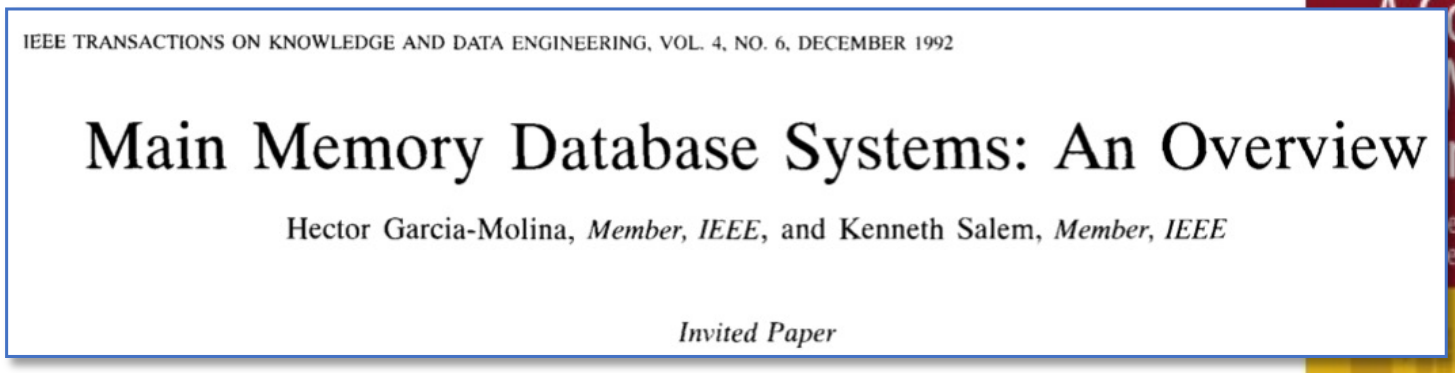
Goal of this lecture …

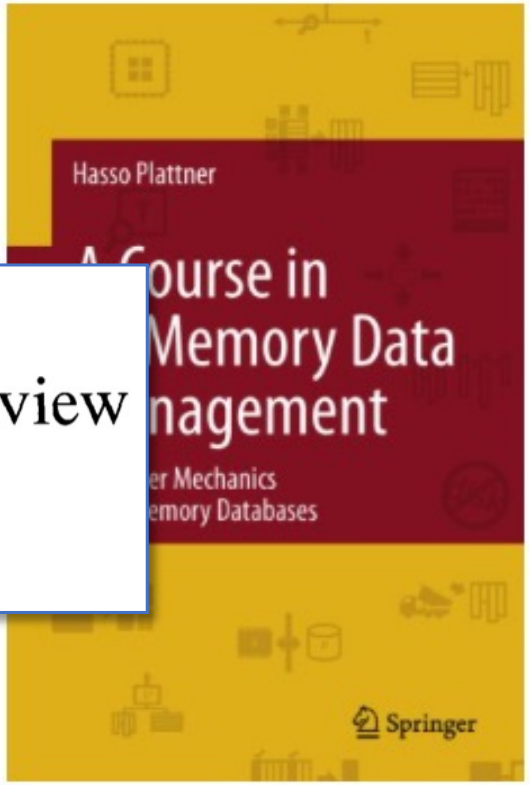
Overview of approaches to dealing with different memory technologies in databases



Main|In-Memory Databases

- database management systems that primarily rely on main memory for computer data storage





Main Memory Databases: Tasks and Use Cases

- Performance
  - Memory access as bottleneck
  - Reduce cache misses
  - Avoid page-based indirection
- Cost
  - Minimize memory footprint
- Persistence
  - with volatile main storage

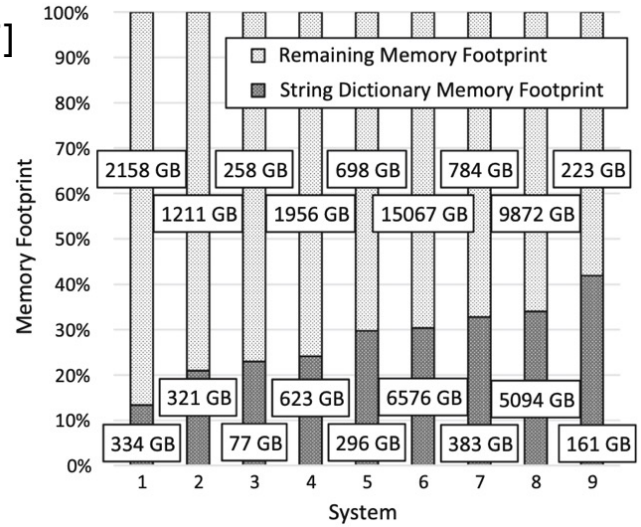
- Analytical databases
  - MonetDB, SAP HANA, Exasol, Hyper, …
- Embedded databases
  - SQlite, DuckDB, ..
- OLTP databases
  - H-Store, VoltDB, SQL Server Hekaton, …
- Stream processing engines
  - …

Research on Main Memory Databases

- Data layout
  - Column store organization [IMDB3]
  - Cache-conscious & in-memory indexes, e.g. CSB+ tree [IMDB14], Bw tree [IMDB15], ART [IMDB12], zone maps
- Persistence
  - Load & save: Sqlite, DuckDB, ..
  - Hot & cold storage [IMDB16]
  - Logging
  - Delta merge [IMDB4], positional delta trees [IMDB5]
- Memory access and query processing
  - Vectorization [IMDB20]
  - JIT query compilation [IMDB9], [IMDB11]
  - Pointer swizzling
- Compression & encoding
  - Encoding of strings (dictionary, front coding, ...) [IMDB6][IMDB7][IMDB8] [IMDB17]
  - Integer compression

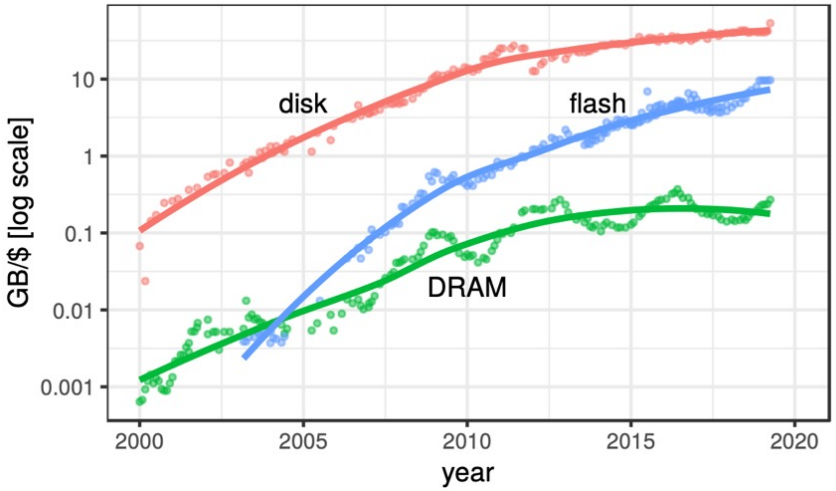
Main Memory Databases: Cost vs. Performance

[IMDB17]



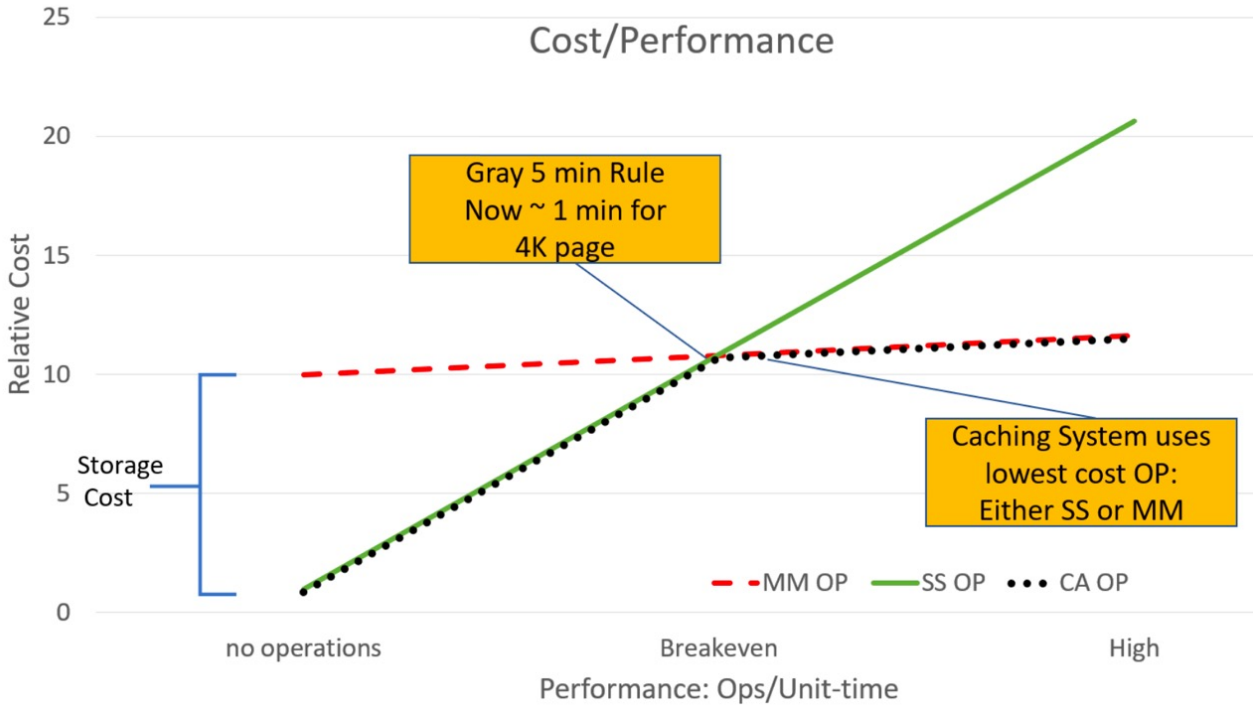
But memory is much more expensive than disk!

There is enough memory for the whole database!



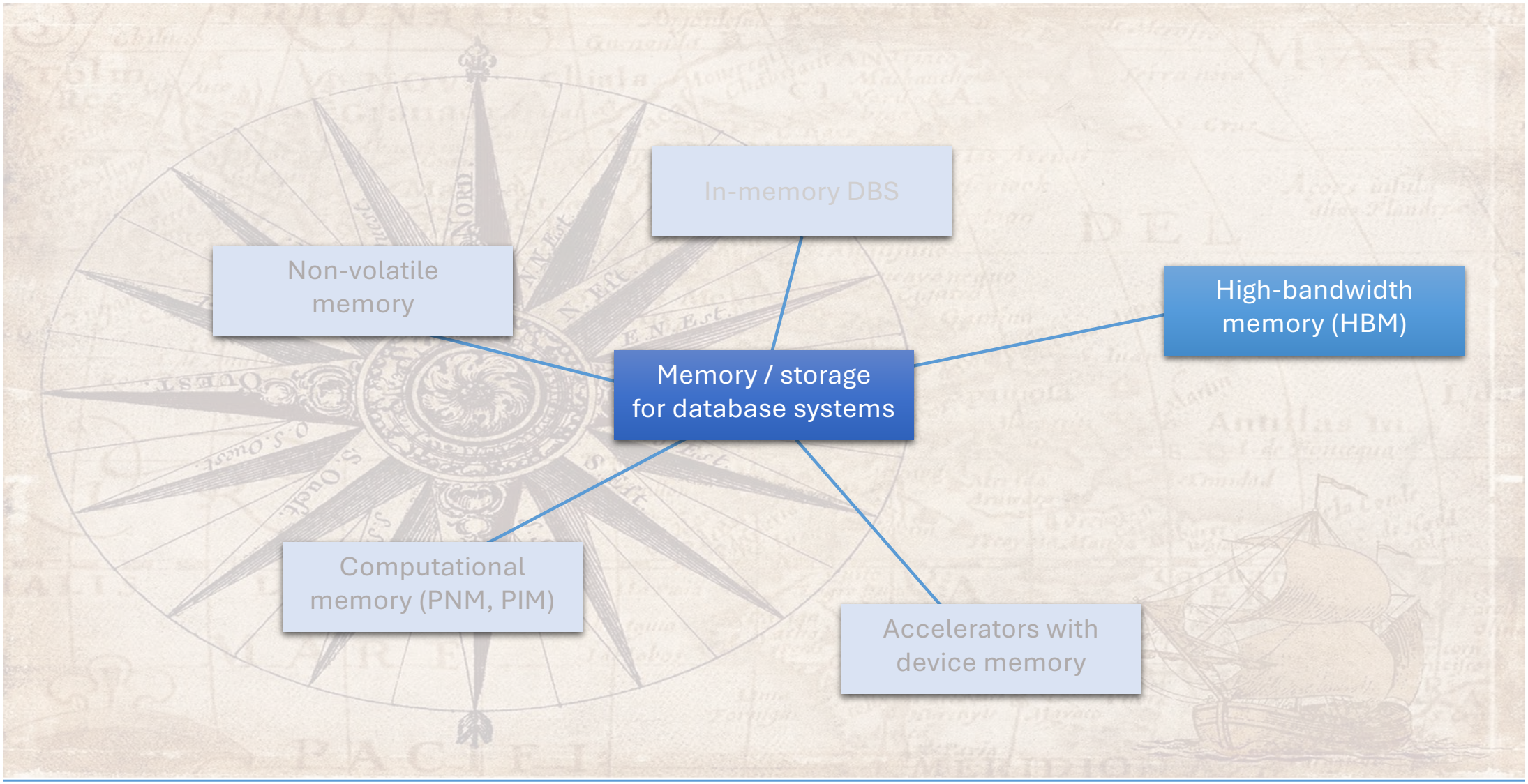
[IMDB18]

Main Memory Databases: Cost vs. Performance



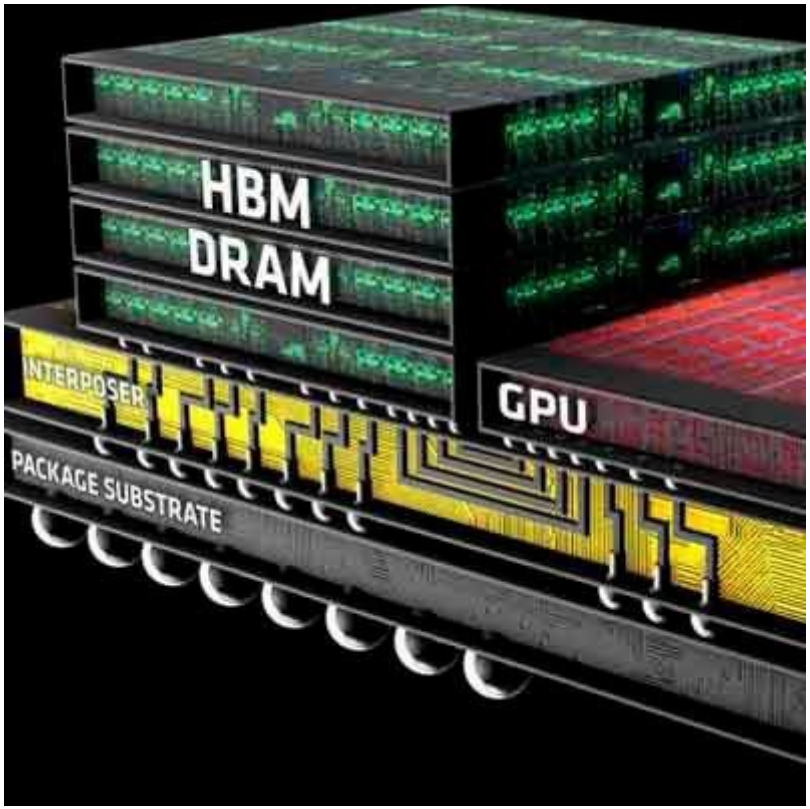
[IMDB2]

- Trend away from memory-only systems?, e.g. Umbra [IMDB19]



High-bandwidth Memory

- DRAM (dies) stacked (3D)
- Wider memory bus (1024 bit vs. 32 bit GDDR5)
- Current technology: HBM2, 2E
- HBM3 with Nvidia Hopper H100 (600 GB/s per site, 5 sites)
- HBM4 announced
- GPU related, but:
  - FPGA-based accelerators with HBM
  - Intel KNL with HBM
  - Samsung: HBM+PIM announced



Source: AMD

Database Research on HBM

- HBM + Manycore Processors (Intel KNL)
  - Hash tables [HBM2], joins [HBM1]
  - Stream processing [HBM5]
  - Sorting [HBM6]
- HBM + FPGA
  - Range selections, joins, linear model training [HBM3] [HBM4]
- HBM + GPU
  - ??

Multi-Channel DRAM: HBM for CPUs

- Used by Intel Xeon Phi „Knights Landing“ (KNL) CPU

- Hardware facts of (our) KNL 7210:
  - 64 cores à 4 threads
  - AVX512 instruction set (SIMD)
  - <1.5 GHz clock frequency
  - Memory: see table
- Feature: MCDRAM as cache

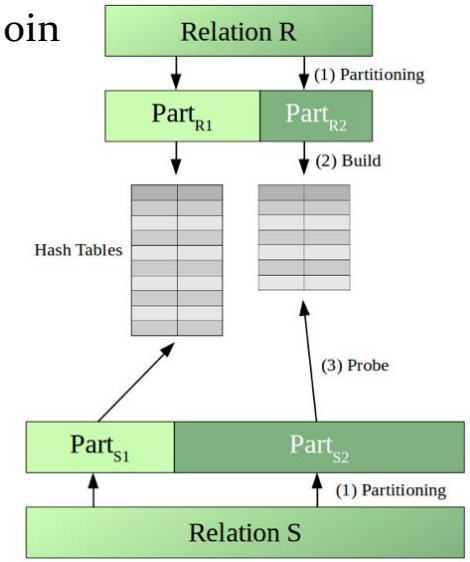
	DDR4	MCDRAM
Size	96 GB	16 GB
Latency	~140 ns	~170 ns
Peak b/w	~71 GB/s	~431 GB/s

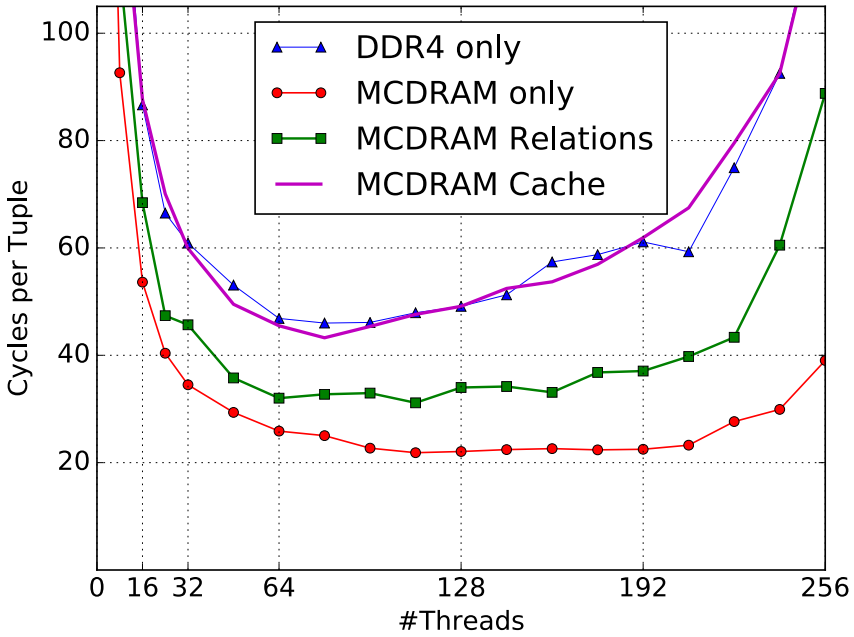
HBM and Join Processing

- Manycore architectures: high number of CPU threads might overwhelm memory controller → memory stalls → performance degradation
- Join algorithms can benefit from higher bandwidth
  - Scanning contiguous memory regions
  - Merge phases
  - How about hash joins?
- Configurable HBM allows two kinds of caching
  - Inclusive caching: lower level includes copies of higher level cache (e.g. L1/2/3 cache)
  - Exclusive caching: content is moved between levels

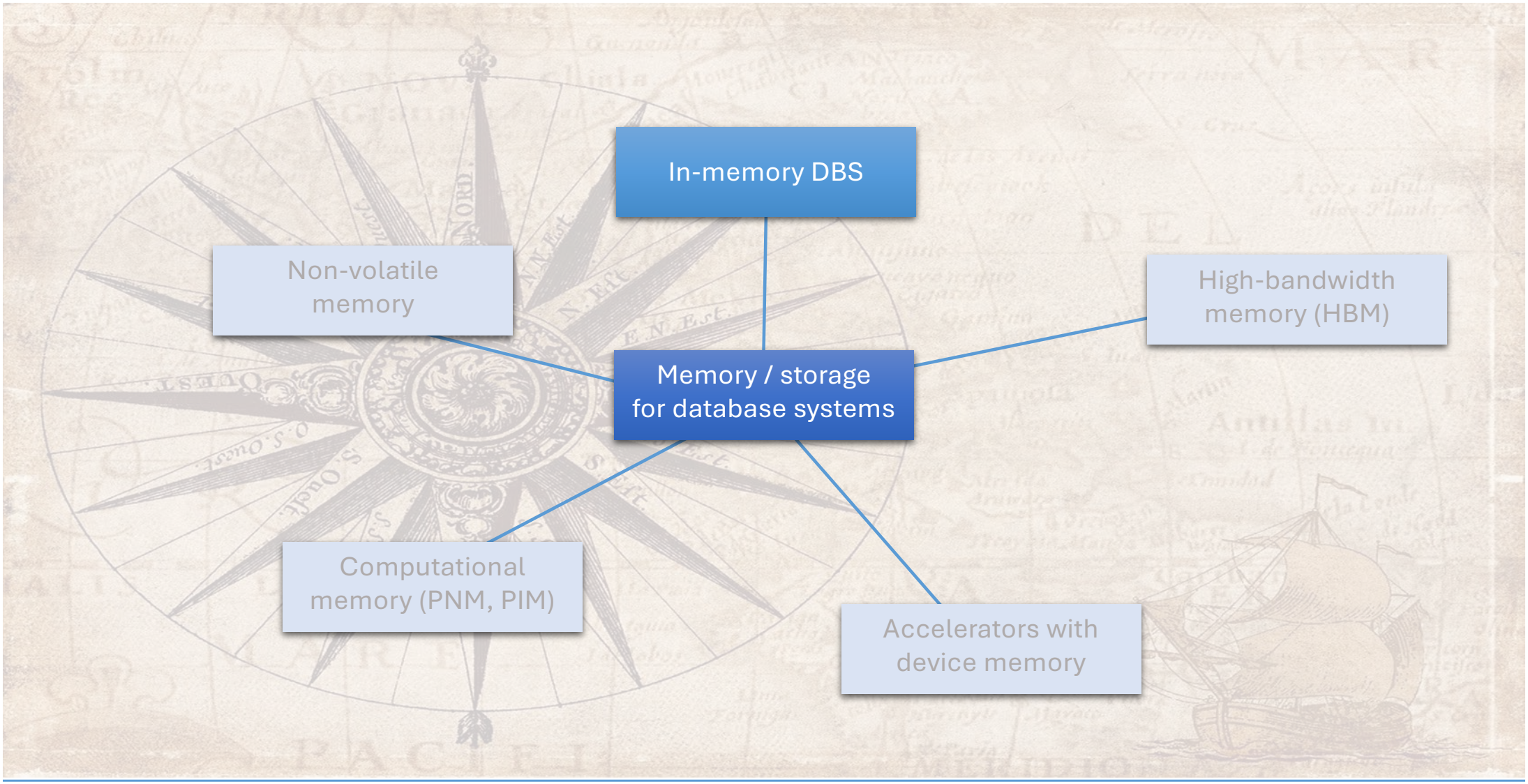
HBM: Join Algorithms and Results [HBM1]

Shared partition join



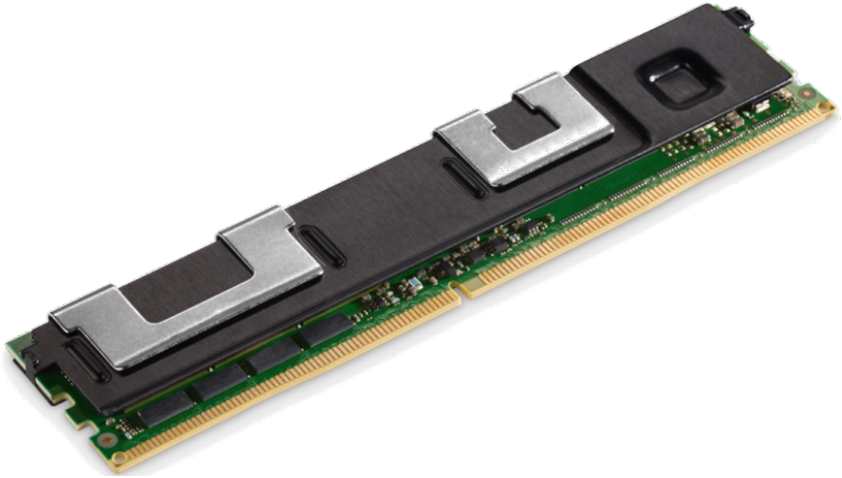


- Both relations are partitioned (hash table per partition)
- Latches for writing to the same partition
- Partitions are assigned to threads
- Partition should fit into cache

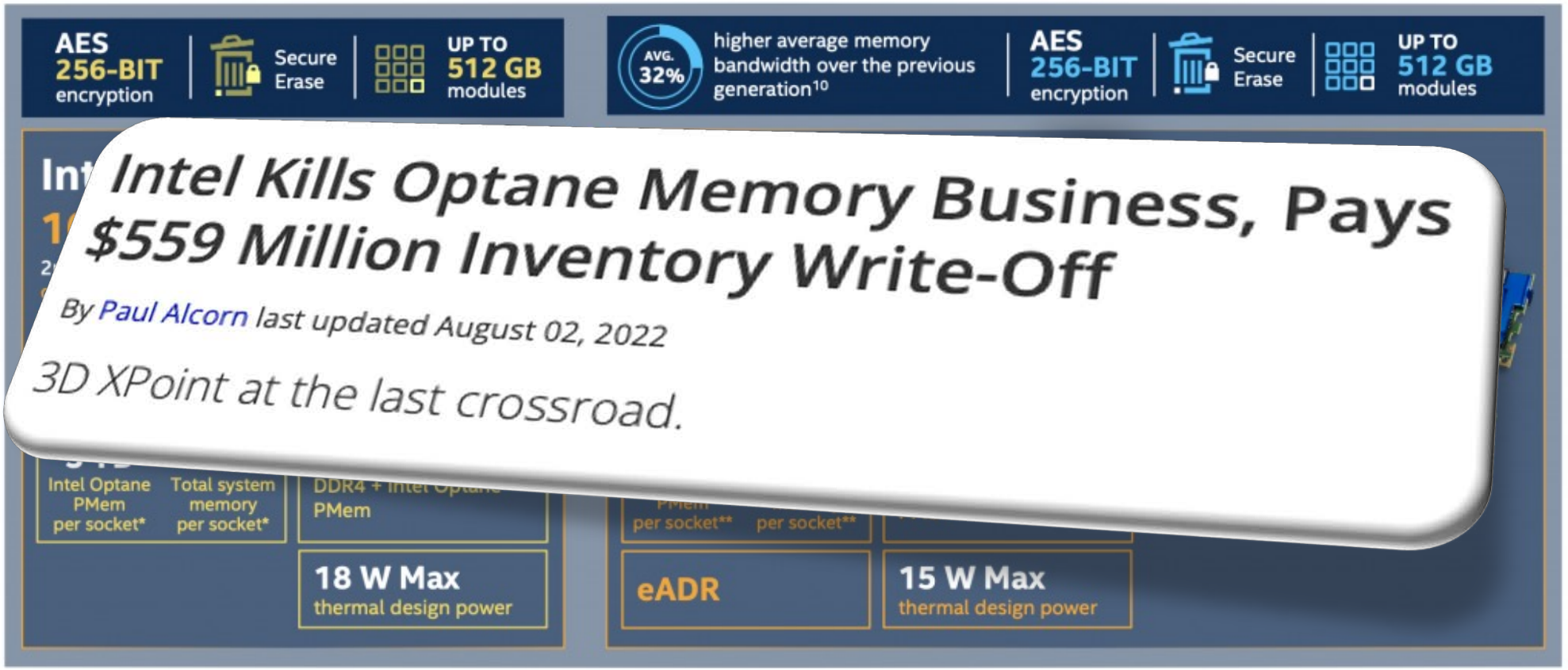


Non-volatile Memory

- also known as non-volatile RAM, persistent memory (PMem)
- In contrast to flash memory
  - byte-adressable memory (supplement/replacement DRAM)
- Different technologies:
  - Phase-Changing Memory (PCM), ferroelectric RAM (FeRAM), magnetoresistive RAM (MRAM), 3D-XPoint (Intel), ...
- with Intel Optane DC Persistent Memory Module (DCPMM) publically available since 2019



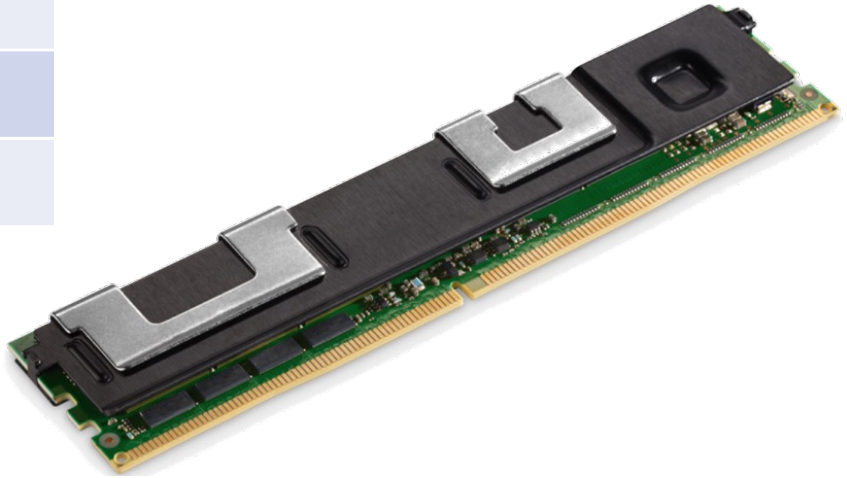
Intel Optane PMem



https://www.storagereview.com/review/intel-optane-persistent-memory-200-series-review-memverge

PMem Characteristics

	DRAM	Optane DC	NAND Flash
Read latency (seq)	80 ns	175 ns	15 µs
Read latency (rand)	90 ns	325 ns	200 µs
Read bandwidth	85 GB/s	32 GB/s	3 GB/s
Write bandwidth	46 GB/s	13 GB/s	0.6 GB/s
Write endurance	$>10^{15}$	N/A	10^4-10^5
Density	1x	2-4x	4-8x



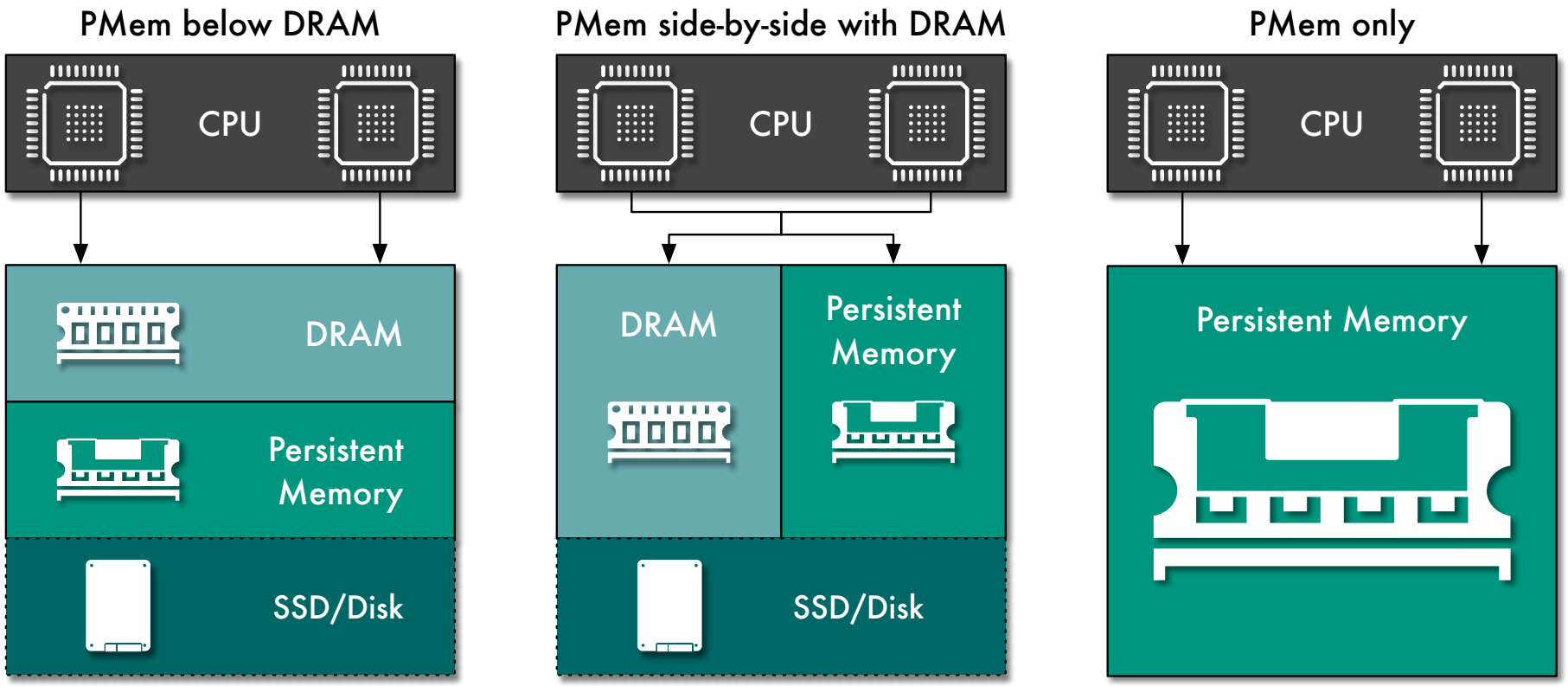
Programming Pitfalls with PMem

- Basically used as ordinary memory, but …
  - writing to PMem requires atomic updates
  - Intel CPUs provide instructions for failure atomic writes of only 8 bytes
- ACID transactions vs. cache flushes

• Controlled by programmer • AC = atomic updates of data structures • I = concurrent updates (multithreading) • D = failure atomic writes to PMem	• Controlled by CPU • Flush is done independently of transaction boundaries • …and only for data has not been flushed so far

- Persistency ist guaranteed only, when data is written to PMem (Power Fail Protection Domain)
- Data still in cache in case of system failure, instruction reordering
- Solution: explicit cache flush, memory barriers

The Role of PMem in Database Systems

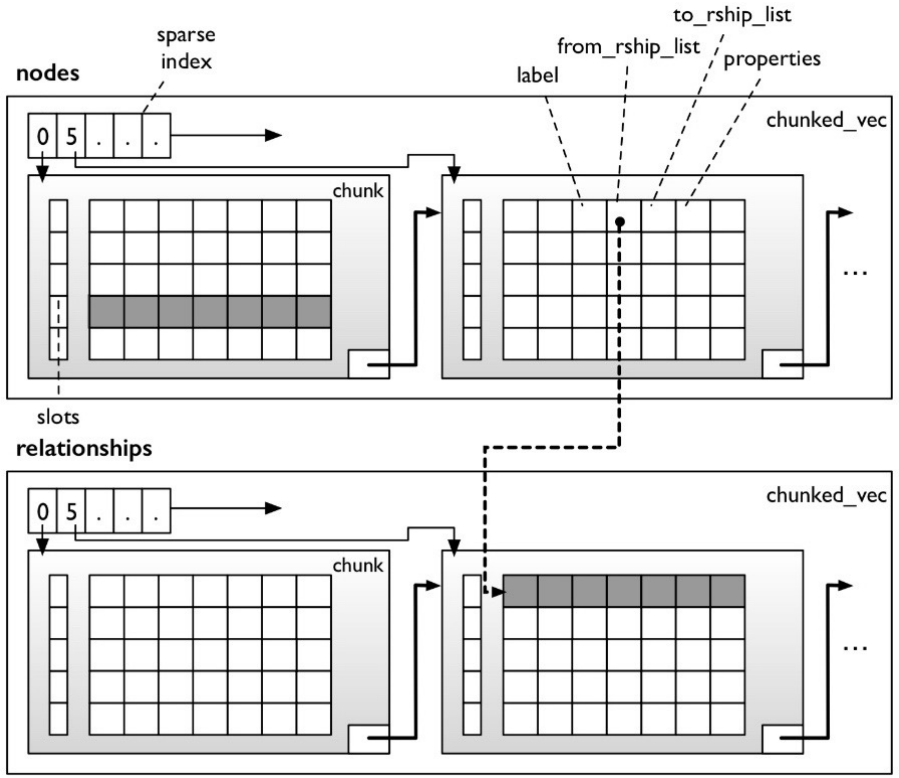


Database Research on PMem

- Index Structures
  - B+-Tree optimizations
  - Special data structures, e.g. FP tree [NVM2], LB+ tree [NVM3], log-structured hash tables [NVM10], multidimensional indexes [NVM7]
  - Data structure engineering [NVM1]
- Primitives for database operators [NVM5], [NVM6]
- Cost modeling for data placement [NVM8]
- Native DBMS [NVM9]

Poseidon: a PMem-native Graph DBMS [NVM9]

- Database system for the property graph model
- Use PMem as primary storage, i.e. maintain graph directly in PMem
  - Nodes, Relationships, and Properties stored in separate tables
  - Tables: Linked-list of chunks (multiple of 256 bytes, cache-line aligned)
  - Chunks: fixed-sized arrays of equally-size object records, properties are outsourced
  - Record access: 8-byte integer offsets instead of 16-byte persistent pointers
- Cypher-like queries, JIT query compilation
- Transactional updates, HTAP workloads (MVCC)



Design Decisions for PMem

- Higher latency and lower bandwidth than DRAM
- Reads and writes behave asymmetrically
- DCPMMs internally work on 256-byte blocks
- Failure atomicity only guaranteed for 8-byte aligned writes
- PMem allocations are expensive
- Dereferencing persistent pointers can prevent optimizations

- (DG1) Algorithmically save writes to PMem
- (DG2) Opt for a hybrid DRAM/PMem storage
- (DG3) Optimize access granularity to 256 bytes
- (DG4) Prefer failure-atomic writes over logging or shadow copying
- (DG5) Use group allocations and reuse blocks of memory instead of deallocating
- (DG6) Avoid dereferencing of persistent pointers

Indexing

- Used for lookup queries on node/relationship properties
- FPTree-like design
  - Inner nodes in DRAM + leaf nodes in PMem
  - Leaf nodes multiple of 256 bytes
  - 64 byte search header per node with bitmap and fingerprinting (1-byte hashes)
  - Fast recovery (no scanning of whole graph)

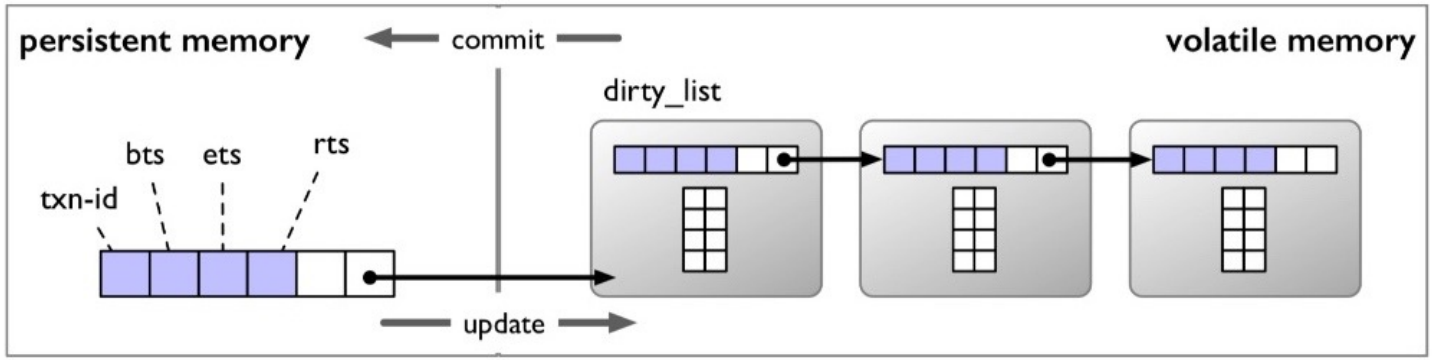

```
template<typename KeyType, typename ValueType
         int N, int M>
class FPTree {
...
struct alignas(64) LeafNode {
    /// bitmap for valid entries
    p<dbis::Bitmap<M>> bits;
    /// fingerprint array
    p<std::array<uint8_t, M>> fp;
    /// pointer to the subsequent sibling
    pptr<LeafNode> nextLeaf;
    /// pointer to the preceeding sibling
    pptr<LeafNode> prevLeaf;
    /// padding to align keys to 64 bytes
    char padding[PaddingSize];

    /// the actual keys and values
    p<std::array<KeyType, M>> keys;
    p<std::array<ValueType, M>> values;
  }
}
```

Transaction Processing

- Multi-version Timestamp Ordering (MVTO)
  - Inserts in PMem, updates on volatile versions
  - atomic copy to PMem during commit via undo log
- Dirty and old versions in DRAM
  - newest-to-oldest
- Transaction-level garbage collection
  - bitmap for reuse of space in PMem

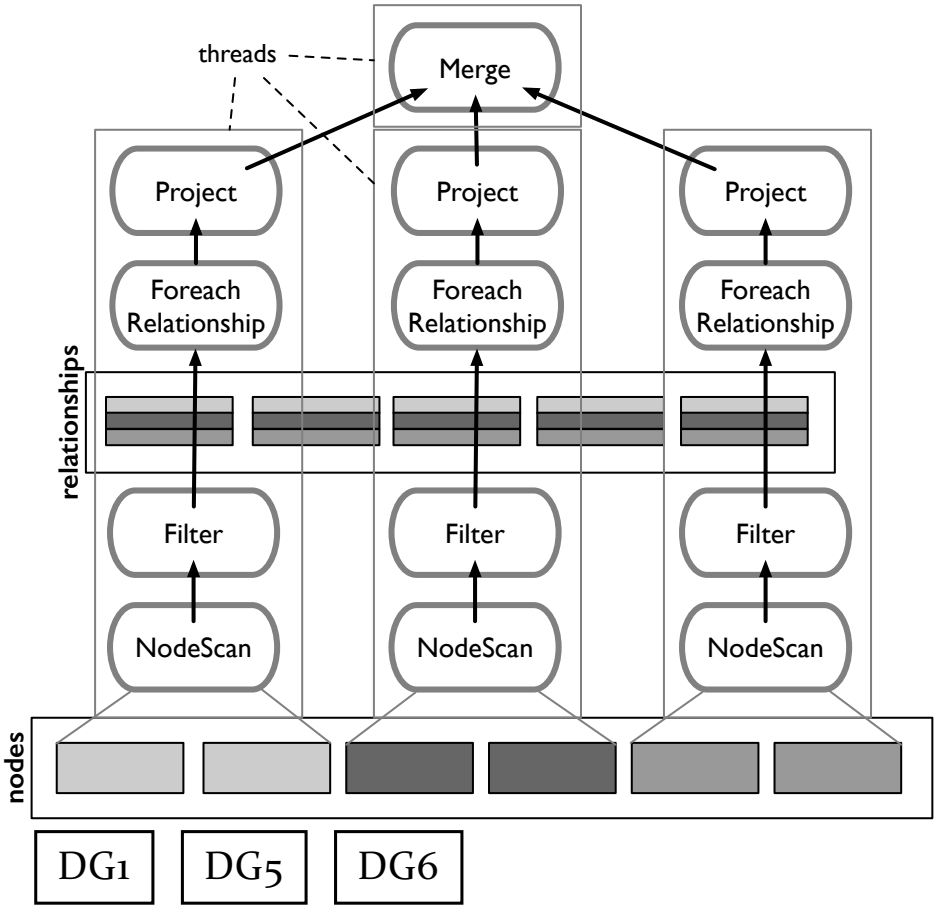
DG1 DG2 DG4



Query Processing

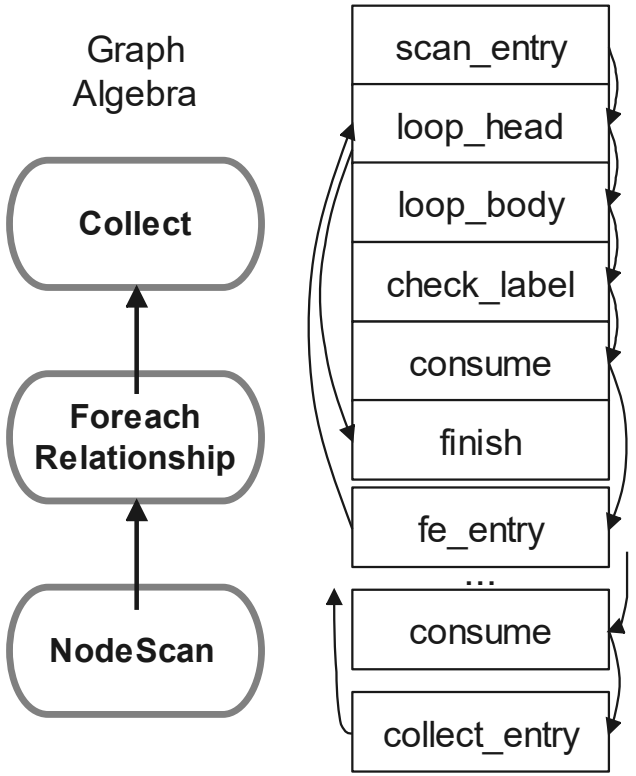
- Push-based query engine
- Graph & relational algebra operators
- Multithreaded (morsel-driven) processing
- Currently, only basic algebraic query language

```
Project([ $0.Name, $2.ID ],
  Expand(OUT, "Post",
    ForeachRelationship(FROM, ':IsLocatedIn‘,
      Filter($0.Age > 18,
        NodeScan("Person")))))
```



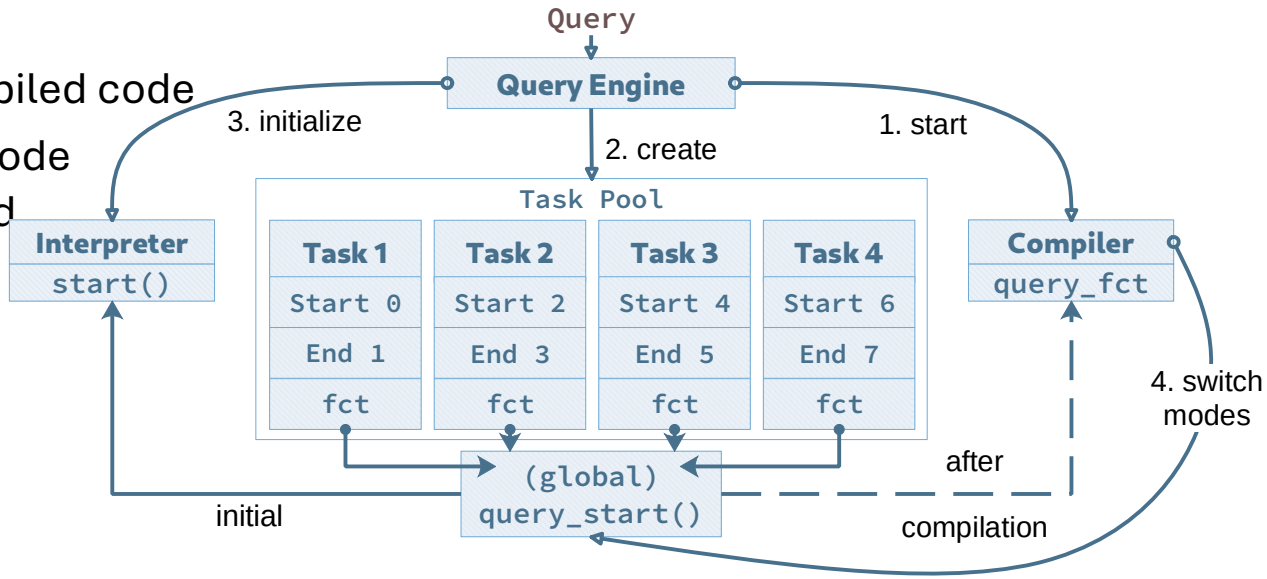
JIT Query Compilation

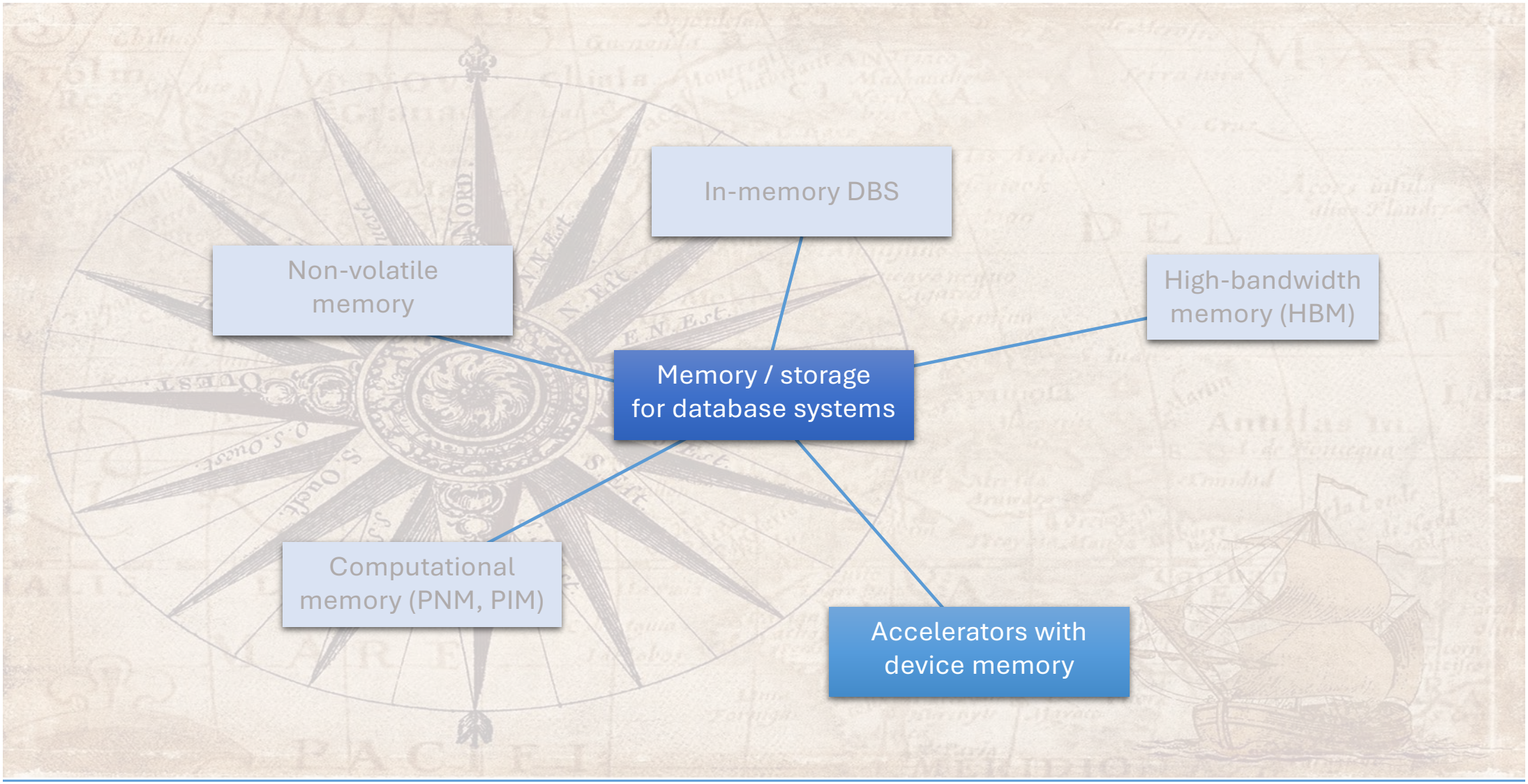
- Goal: cache-friendly query execution by Just-In-Time compilation
- Code Generation
  - graph algebra expressions to machine code
  - LLVM 11 used as backend for JIT compilation
- Requirements
  - processing tuples in registers as long as possible
  - pre-processing of initialization and space
  - type handling at code generation
  - Using IR instead of high-level C



Adaptive Query Compilation

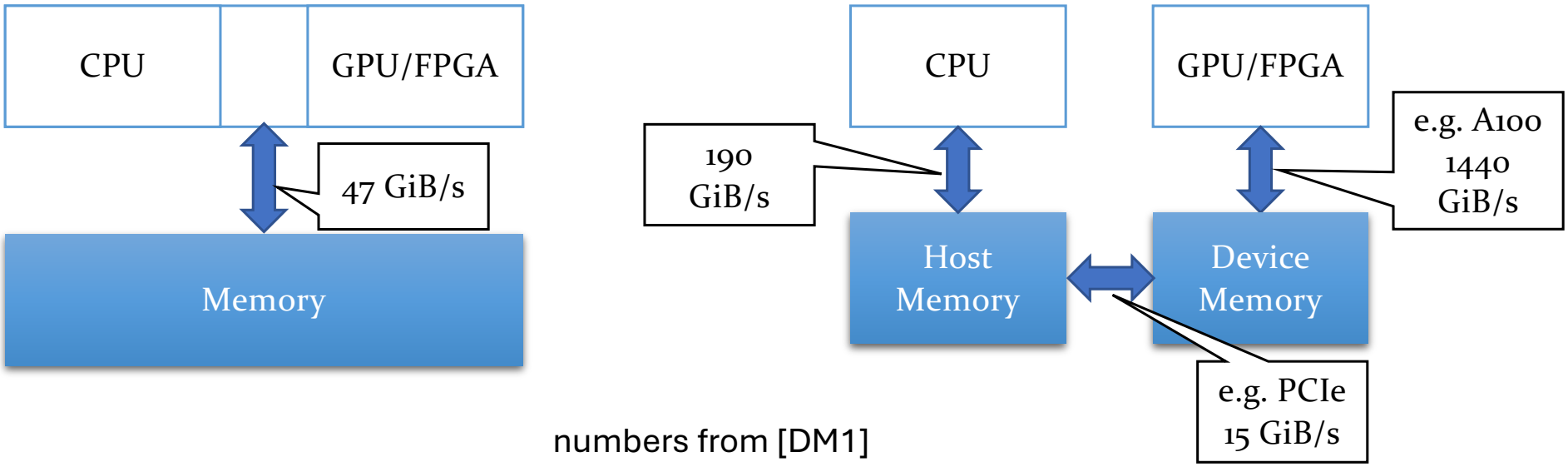
- Startup time of JIT compilation too long for short-running queries
- Idea: combine Ahead-Of-Time compiled code (interpreter) and JIT compilation
- Adaptive Execution
  - start execution on AOT-compiled code
  - run query compilation and code generation in the background
  - switch execution to compiled-code when compilation completes
- persistent code cache





Database Accelerators

- Coprocessing units for accelerating database operations
- GPU or FGPA
- integrated (shared memory) or with dedicated memory (e.g. HBM)



numbers from [DM1]

Database Research on Accelerators

GPU Processing

- Standard database operations [DM4], e.g. joins [DM3], aggregations [DM6], …
- Indexes [DM7] and index operations [DM8]
- Decompression [DM12]
- Scheduling & pipelining [DM9][DM10]
- Placement [DM5], data transfer [DM11]

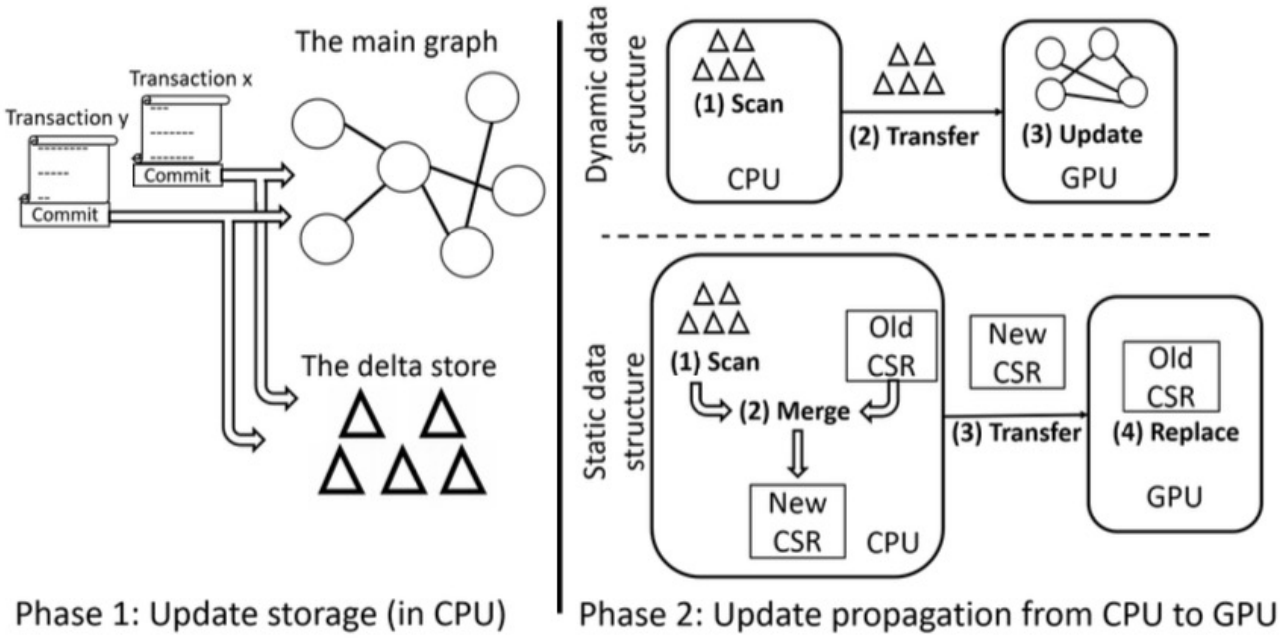
FPGA Processing

- Standard database operations [DM17][DM18], e.g. joins [DM19][DM20], aggregations, …
- Compression/decompression [DM14]
- Pipelining [DM14]
- Data partitioning [DM17]

Challenges	• Data transfer • Query compilation and execution • Placement	[DM1][DM2][DM13]

Update Handling for Heterogeneous HTAP [DM11]

- Transactional graph database Poseidon: property graph model
- GPU-powered graph analytics based on Gunrock: compressed sparse row (CSR) format
- CSR = static structure of (values, colInd, rowPtr)
- When/how to update CSR from transactional data store?



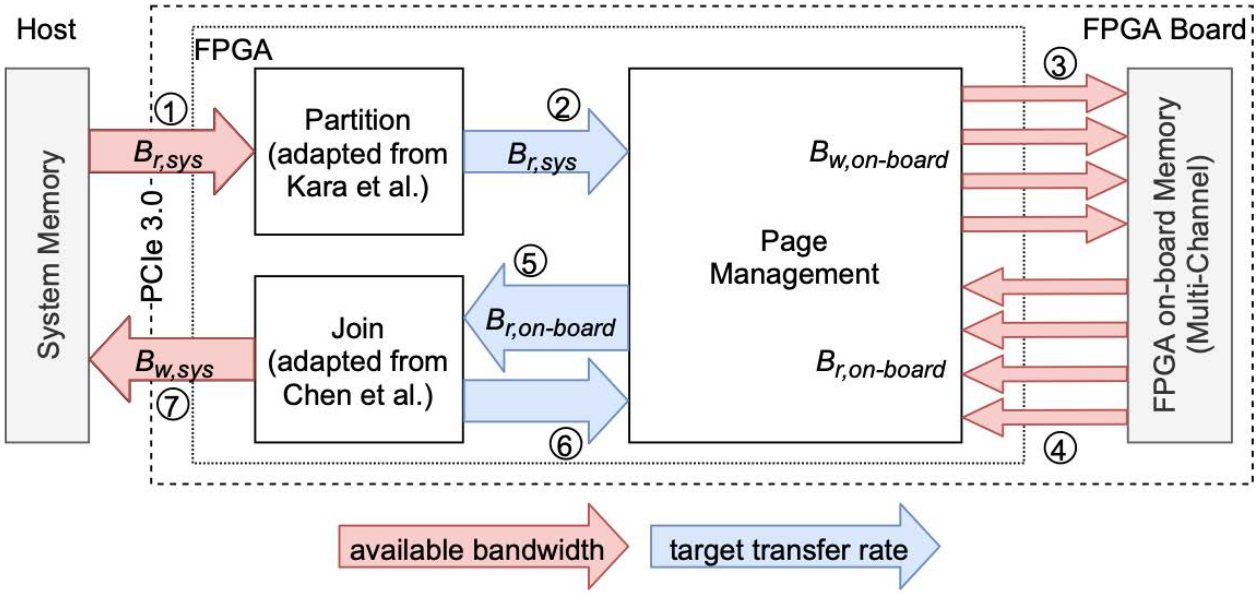
Update Handling for Heterogeneous HTAP [DM11]

- Graph 500 data at scale 24, 2M deltas
- Sortledton [P. Fuchs, J. Giceva, D. Margan. 2022. Sortledton: a universal, transactional graph data structure. Proc. VLDB Endow. 15, 6 (2022), 1173–1186.]

		BFS	PR	SSSP
Sortledton	Analytics on CPU	**1.48**	**21.34**	**57.30**
DELTA^{FE}	Update Propagation	5.38	5.38	5.38
	Analytics on GPU	0.07	0.30	0.13
	Sum	**5.45**	**5.68**	**5.51**

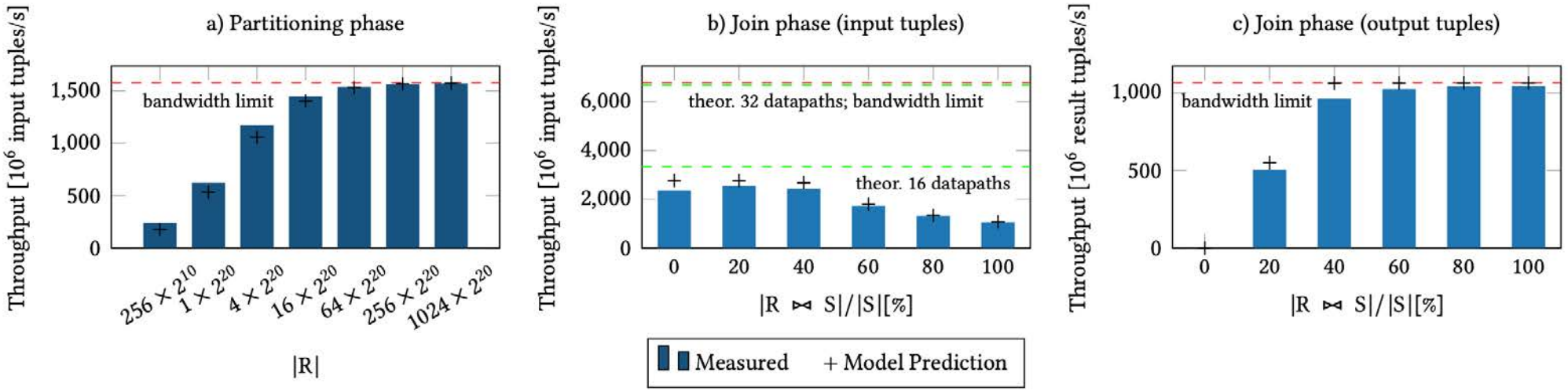
Table 1: HTAP and H^{2}TAP Analytics Latency (in seconds).

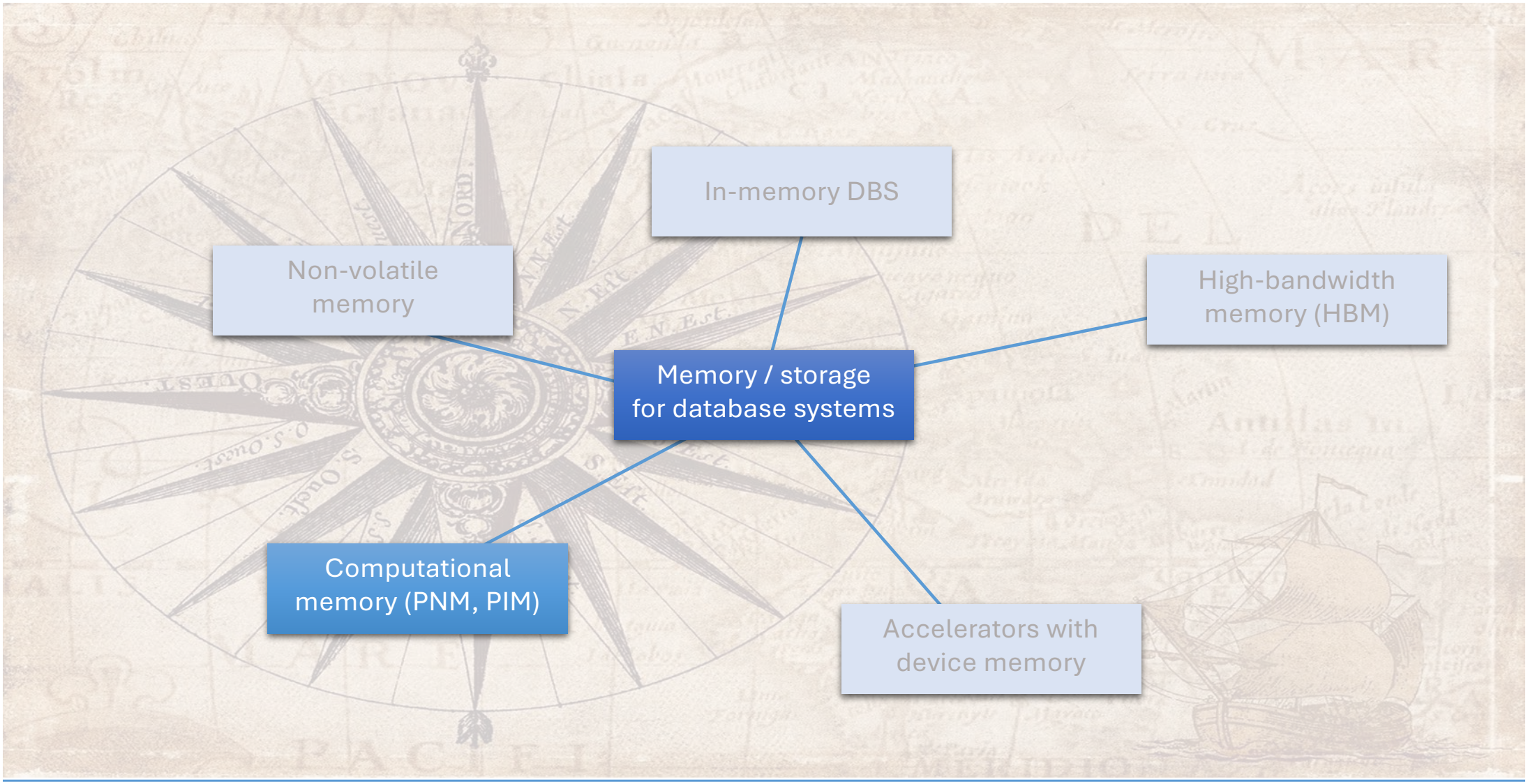
Bandwidth-optimal FPGA Join [DM20]



- Execute both phases of partitioned hash join on FPGA board
- utilizes the full available memory bandwidth without interruption for the whole duration of the join
- Partitioning by write combiners
- Page-based writing to on-board memory
- Partition-by-partition join

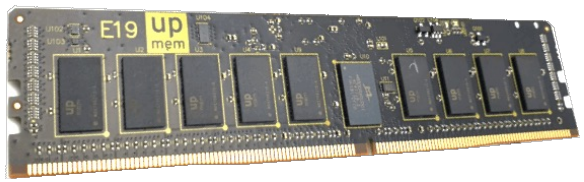
Bandwidth-optimal FPGA Join [DM20]





Offloading Computation to Memory

- Not a new idea: „Logic-in-Memory“ (1069/70)

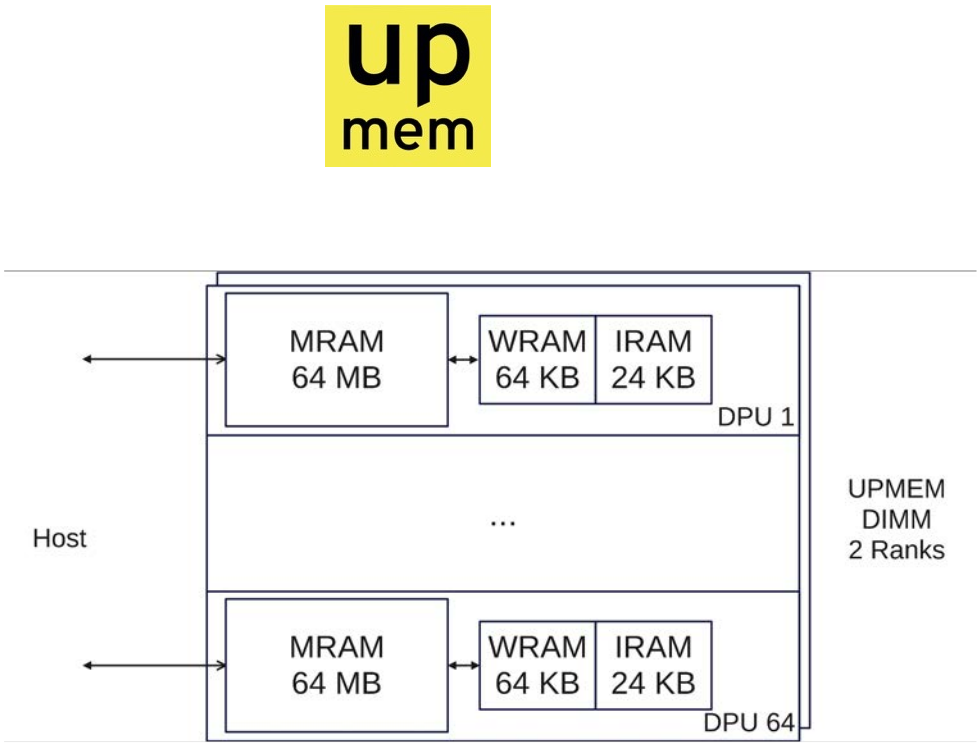


- **Processing-In-Memory:** execute operations directly in memory, e.g. UPMEM
- **Processing-Near-Memory:** processing close to memory to improve latency/ bandwidth, e.g. by FPGAs

- **Expected benefits:** reduced CPU load, increased bandwidth, reduced energy consumption(?), simplified development

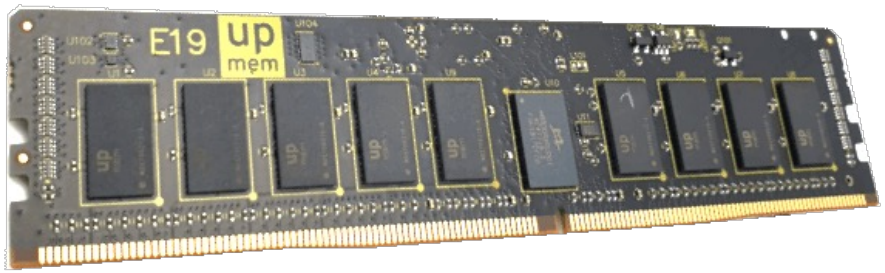
UPMEM Architecture

- First available real PIM hardware
- PIM-enabled DRAM DIMMs
  - DDR4-2400 DIMMs with additional PIM chips
  - Organized into ranks with up to 8 PIM chips
  - 8 GB RAM per DIMM
- PIM chip with up to 8 DRAM Processing Units (DPUs)
  - 128 DPUs per DIMM
- Exclusive access to
  - 64 MB MRAM (host-accessible)
  - 64 KB WRAM
  - 24 KB IRAM
- No direct communication between DPUs



UPMEM Architecture: DPU

- DRAM Processing Units (DPUs)
  - 32-bit RISC core
  - Up to 400 MHz
  - Multithreaded in-order pipeline
  - 24 hardware threads (tasklets)
  - Threads share memory on DPU



UPMEM: Programming Model

- Single Program Multiple Data
- Execution of same code on different data
- Predefined number of tasklets
- Synchronization between tasklets on DPUs
  - Mutexes
  - Barriers
  - Semaphores
- C-based SDK based on LLVM

Programming Pattern

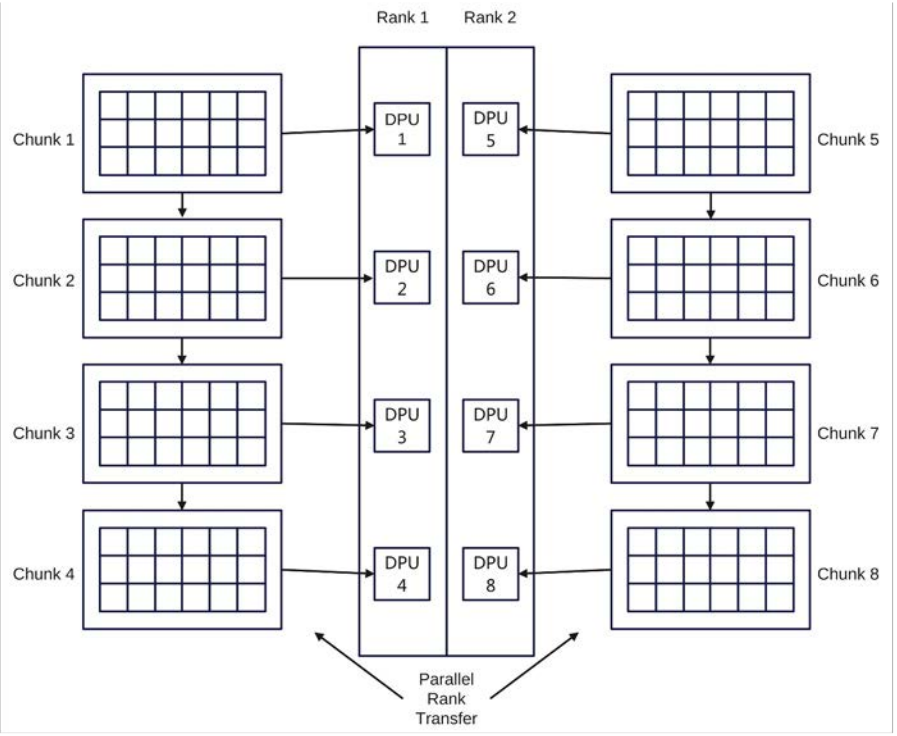
- Host allocates and initializes DPUs/ranks
- Data transfer from host to MRAM of DPUs
- Execution and control of DPU program by host
- Data transfer
  - Synchronous or asynchronous
  - 8-byte alignment
  - Same size for parallel transfer
- Results through DPU-to-host data transfer

Database Research on PIM

- Benchmarking & evaluating UPMEM
  - PrIM [PIM3], case studies [PIM4]
- Index structures
  - Skip list [PIM8], PIM tree [PIM7]
- Database operators
  - Scans, scans with filters [PIM5], [PIM6]
- Other technologies
  - AxDIMM [PIM9]

PIM-accelerated Table Scans [PIM5]

- Chunk-organized tables
  - Corresponding, optimized data structures for MRAM
- Host-to-DPU data transfer
- Exploit structure of tables
  - Chunk-DPU assignment
  - Transfer as much as possible to MRAM
  - Parallel and async transfer
- Memory exceed
  - Multiple runs
  - Execute program asynchronously



Benchmarks: Parallelism

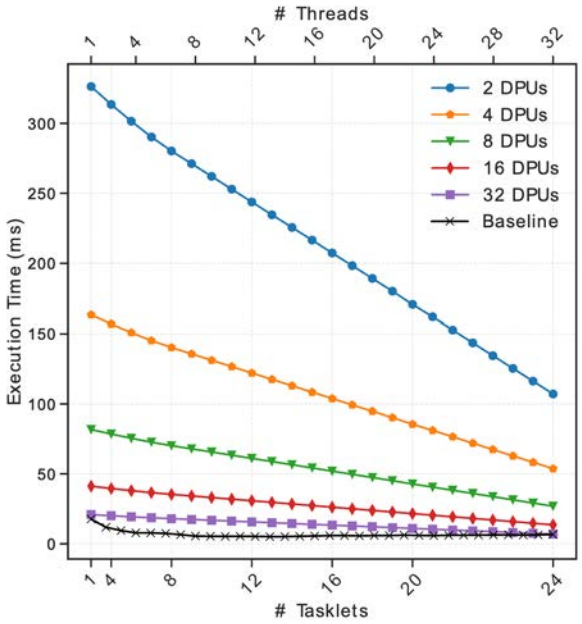


Table Scan with 2 - 32 DPUs

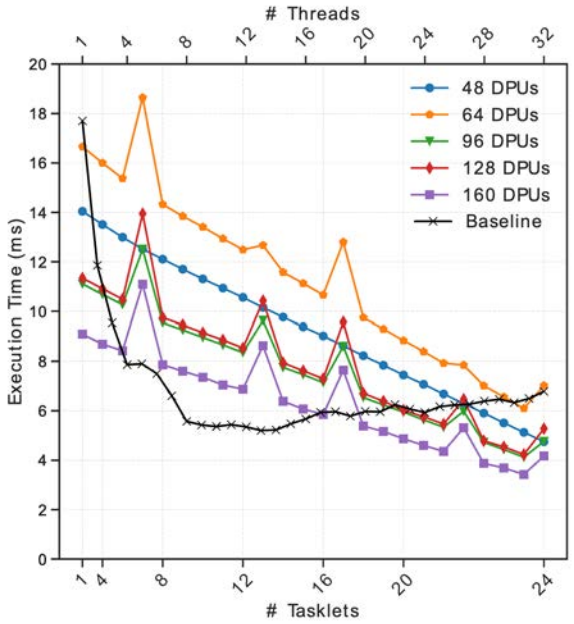


Table Scan with 48 - 160 DPUs

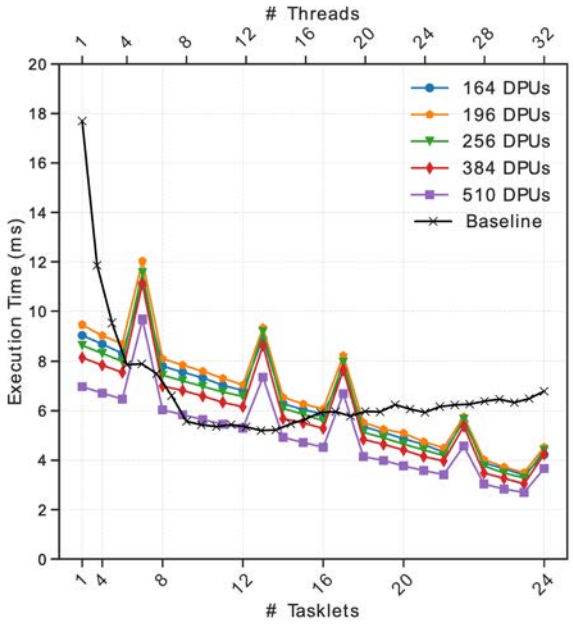
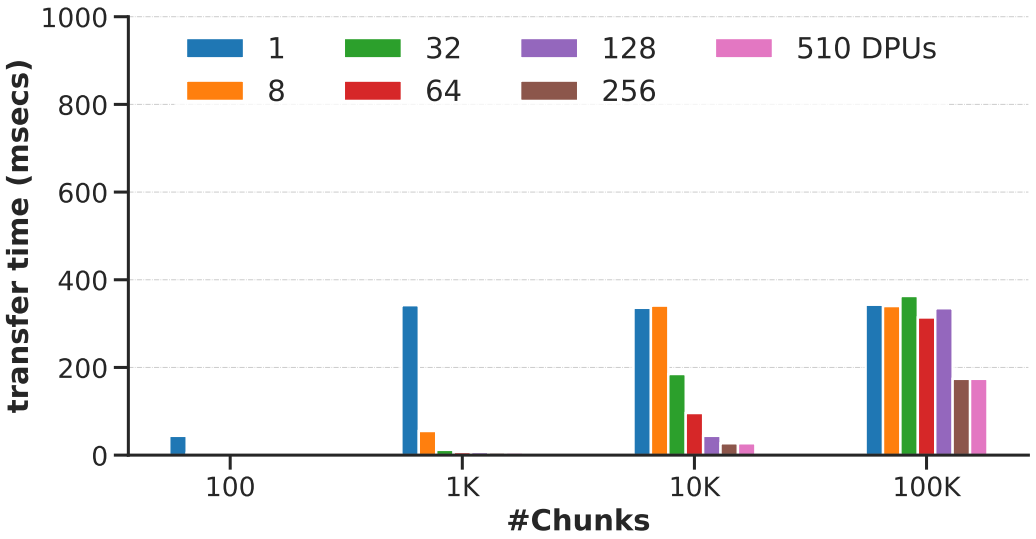
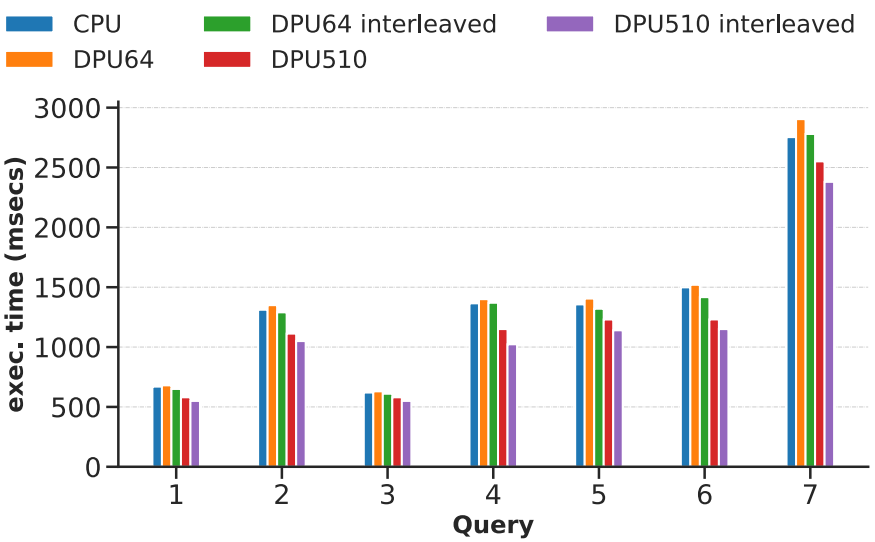


Table Scan with 164 - 510 DPUs

Benchmarks: Transfer






FUTURE
FutureRoad
80
NEXT
Learning

Conclusion

- Challenges ahead:
  - data volume, workload complexity
  - infrastructure (hardware architectures, cloud, ...)
  - energy consumption
- New exciting memory/storage technologies available
  - Dimensions: persistence, computation, bandwidth, ...

Research Questions for the Database & Systems Community

- How to utilize memory regions with different properties (persistence, latency/bandwidth, ...)?
- Can we provide abstractions, e.g. a buffer manager for non-hierarchical memory layers?
  - Variable-size pages, inclusive/exclusive caching, cache coherence, different APIs, ...
- How to utilize in/near-memory processing in databases?
- How to estimate benefits/costs of transfer?

Further reading

- [IMDB1] F. Färber, A. Kemper, P.-Å. Larson, J. Levandoski, T. Neumann, A. Pavlo: Main Memory Database Systems. Found. Trends Databases 8(1-2): 1-130 (2017)
- [IMDB2] D. Lomet: Cost/performance in modern data stores: how data caching systems succeed. DaMoN 2018: 9:1-9:10
- [IMDB3] D. Abadi, P. Boncz, S. Harizopoulos: Column oriented Database Systems. Proc. VLDB Endow. 2(2): 1664-1665 (2009)
- [IMDB4] J. Krüger, C. Kim, M. Grund, N. Satish, D. Schwalb, J. Chhugani, H. Plattner, P. Dubey, A. Zeier: Fast Updates on Read-Optimized Databases Using Multi-Core CPUs. Proc. VLDB Endow. 5(1): 61-72 (2011)
- [IMDB5] S. Héman, M. Zukowski, N.J. Nes, L. Sidirourgos, P. Boncz: Positional update handling in column stores. SIGMOD Conference 2010: 543-554
- [IMDB6] I. Müller, C. Ratsch, F. Färber: Adaptive string dictionary compression in in-memory column-store database systems. In: EDBT, pp. 283–294 (2014)
- [IMDB7] D. J. Abadi, S. Madden, M. Ferreira: Integrating compression and execution in column-oriented database systems. SIGMOD Conference 2006: 671-682
- [IMDB8] C. Lemke, K. Sattler, F. Färber, A. Zeier: Speeding Up Queries in Column Stores - A Case for Compression. DaWak 2010: 117-129
- [IMDB9] P. Menon, A. Pavlo, T. C. Mowry: Relaxed Operator Fusion for In-Memory Databases: Making Compilation, Vectorization, and Prefetching Work Together At Last. Proc. VLDB Endow. 11(1): 1-13 (2017)

Further reading

- [IMDB11] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, P. Boncz: Everything You Always Wanted to Know About Compiled and Vectorized Queries But Were Afraid to Ask. Proc. VLDB Endow. 11(13): 2209-2222 (2018)
- [IMDB12] V. Leis, A. Kemper, T. Neumann: The adaptive radix tree: ARTful indexing for main-memory databases. ICDE 2013: 38-49
- [IMDB13] F. Funke, A. Kemper, T. Mühlbauer, T. Neumann, V. Leis: HyPer Beyond Software: Exploiting Modern Hardware for Main-Memory Database Systems. Datenbank-Spektrum 14(3): 173-181 (2014)
- [IMDB14] J. Rao, K. A. Ross: Making B+-Trees Cache Conscious in Main Memory. SIGMOD Conference 2000: 475-486
- [IMDB15] J. Levandoski, D. Lomet, S. Sengupta: The Bw-Tree: A B-tree for new hardware platforms. ICDE 2013: 302-313
- [IMDB16] A. Eldawy, J. Levandoski, P. Larson: Trekking Through Siberia: Managing Cold Data in a Memory-Optimized Database. PVLDB 7(11): 931-942, (2014)
- [IMDB17] R. Lasch, I. Oukid, R. Dementiev, N. May, S.S. Demirsoy, K. Sattler: Faster & strong: string dictionary compression using sampling and fast vectorized decompression. VLDB J. 29(6): 1263-1285 (2020)
- [IMDB18] G. Haas, M. Haubenschild, V. Leis: Exploiting Directly-Attached NVMe Arrays in DBMS. CIDR 2020
- [IMDB19] T. Neumann, M. J. Freitag: Umbra: A Disk-Based System with In-Memory Performance. CIDR 2020
- [IMDB20] P. Boncz, M. Zukowski, N. Nes: MonetDB/X100: Hyper-Pipelining Query Execution. CIDR 2005

Further reading

- [HBM1] C. Pohl, K. Sattler, G. Graefe. Joins on high-bandwidth memory: a new level in the memory hierarchy, The VLDB Journal (2019), doi:10.1007/s00778-019-00546-z
- [HBM2] X. Cheng, B. He, E. Lo, W. Wang, S. Lu, X. Chen: Deploying Hash Tables on Die-Stacked High Bandwidth Memory. CIKM 2019: 239-248
- [HBM3] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, G. Alonso: High Bandwidth Memory on FPGAs: A Data Analytics Perspective. FPL 2020: 1-8
- [HBM4] R. Shi, K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, G. Alonso: Exploiting HBM on FPGAs for Data Processing. ACM Trans. Reconfigurable Technol. Syst. 15(4): 36:1-36:27 (2022)
- [HBM5] H. Miao, M. Jeon, G. Pekhimenko, K. S. McKinley, F. X. Lin: StreamBox-HBM: Stream Analytics on High Bandwidth Hybrid Memory. ASPLOS 2019: 167-181
- [HBM6] B. Bramas: Fast sorting algorithms using avx-512 on intel knights landing. arXiv preprint arXiv:1704.08579 (2017).

Further reading

- [DM1] V. Rosenfeld, S. Breß, V. Markl: Query Processing on Heterogeneous CPU/GPU Systems. ACM Comput. Surv. 55(2): 11:1-11:38 (2023)
- [DM2] J. Paul, S. Lu, B. He: Database Systems on GPUs. Found. Trends Databases 11(1): 1-108 (2021)
- [DM3] J. Paul, B. He, S. Lu, C. Tong Lau: Revisiting hash join on graphics processors: a decade later. Distributed Parallel Databases 38(4): 771-793 (2020)
- [DM4] B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, P. V. Sander: Relational query coprocessing on graphics processors. ACM Trans. Database Syst. 34(4): 21:1-21:39 (2009)
- [DM5] T. Karnagel, D. Habich, W. Lehner: Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. Proc. VLDB Endow. 10(7): 733-744 (2017)
- [DM6] T. Karnagel, R. Müller, G. M. Lohman: Optimizing GPU-accelerated Group-By and Aggregation. ADMS@VLDB 2015: 13-24
- [DM7] A. Shahvarani, H.-A. Jacobsen: A Hybrid B+-tree as Solution for In-Memory Indexing on CPU-GPU Heterogeneous Computing Platforms. SIGMOD 2016: 1523-1538
- [DM8] F. Beier, T. Kilias, K. Sattler: GiST scan acceleration using coprocessors. DaMoN 2012: 63-69

Further reading

- [DM9] S. Breß, F. Beier, H. Rauhe, K. Sattler, E. Schallehn, G. Saake: Efficient co-processor utilization in database query processing. Inf. Syst. 38(8): 1084-1096 (2013)
- [DM10] H. Funke, S. Breß, S. Noll, V. Markl, J. Teubner: Pipelined Query Processing in Coprocessor Environments. SIGMOD 2018: 1603-1618
- [DM11] M. A. Jibril, H. Al-Sayeh, A. Baumstark, K. Sattler: Fast and Efficient Update Handling for Graph H2TAP, EDBT 2023
- [DM12] E. Rozenberg, P. A. Boncz: Faster across the PCIe bus: a GPU library for lightweight decompression: including support for patched compression schemes. DaMoN 2017: 8:1-8:5

Further reading

- [DM13] J. Fang, Y. T. B. Mulder, J. Hidders, J. Lee, H. P. Hofstee: In-memory database acceleration on FPGAs: a survey. VLDB J. 29(1): 33-59 (2020)
- [DM14] M. Chiosa, F. Maschi, I. Müller, G.Alonso, N. May: Hardware Acceleration of Compression and Encryption in SAP HANA. Proc. VLDB Endow. 15(12): 3277-3291 (2022)
- [DM15] M. Owaida, G. Alonso, L. Fogliarini, A. Hock-Koon, P.-E. Melet: Lowering the Latency of Data Processing Pipelines Through FPGA based Hardware Acceleration. Proc. VLDB Endow. 13(1): 71-85 (2019)
- [DM16] K. Kara, J. Giceva, G. Alonso: FPGA-based Data Partitioning. SIGMOD Conference 2017: 433-445
- [DM17] D. Sidler, Z. István, M. Owaida, K. Kara, G. Alonso: doppioDB: A Hardware Accelerated Database. SIGMOD Conference 2017: 1659-1662
- [DM18] Lekshmi B. G., A. Becher, K. Meyer-Wegener, S. Wildermann, J. Teich: SQL Query Processing Using an Integrated FPGA-based Near-Data Accelerator in ReProVide. EDBT 2020: 639-642
- [DM19] X. Chen, Y. Chen, R. Bajaj, J. He, B. He, W.-F. Wong, D. Chen: Is FPGA Useful for Hash Joins? CIDR 2020
- [DM20] R. Lasch, M. Moghaddamfar, N. May, S. S. Demirsoy, C. Färber, K. Sattler: Bandwidth-optimal Relational Joins on FPGAs. EDBT 2022: 1:27-1:39

Further reading

- [NVM1] I. Oukid, W. Lehner: Data Structure Engineering For Byte-Addressable Non-Volatile Memory. SIGMOD Conference 2017: 1759-1764
- [NVM2] I. Oukid, J. Lasperas, A. Nica, T. Willhalm, W. Lehner: FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. SIGMOD Conference 2016: 371-386
- [NVM3] J. Liu, S. Chen, L. Wang: LB+-Trees: Optimizing Persistent Index Performance on 3DXPoint Memory. Proc. VLDB Endow. 13(7): 1078-1090 (2020)
- [NVM4] P. Götze, A. van Renen, L. Lersch, V. Leis, I. Oukid: Data Management on Non-Volatile Memory: A Perspective. Datenbank-Spektrum 18(3): 171-182 (2018)
- [NVM5] P. Götze, A. K. Tharanatha, K. Sattler: Data structure primitives on persistent memory: an evaluation. DaMoN 2020: 15:1-15:3
- [NVM6] A. van Renen, L. Vogel, V. Leis, T. Neumann, A. Kemper: Building blocks for persistent memory. VLDB J. 29(6): 1223-1241 (2020)
- [NVM7] M. A. Jibril, P. Götze, D. Broneske, K. Sattler: Selective caching: a persistent memory approach for multi-dimensional index structures. Distributed Parallel Databases 40(1): 47-66 (2022)

Further reading

- [NVM8] R. Lasch, T. Legler, N. May, B. Scheirle, K. Sattler: Cost Modelling for Optimal Data Placement in Heterogeneous Main Memory. Proc. VLDB Endow. 15(11): 2867-2880 (2022)
- [NVM9] M. A. Jibril, A. Baumstark, P. Götze, K. Sattler: JIT happens: Transactional Graph Processing in Persistent Memory meets Just-In-Time Compilation. EDBT 2021: 37-48
- [NVM10] L. Vogel, A. van Renen, S. Imamura, J. Giceva, T. Neumann, A. Kemper: Plush: A Write-Optimized Persistent Log-Structured Hash-Table. Proc. VLDB Endow. 15(11): 2895-2907 (2022)
- [NVM11] J. Arulraj, A. Pavlo: Non-Volatile Memory Database Management Systems. Synthesis Lectures on Data Management, Morgan & Claypool Publishers 2019

Further reading

- [PIM1] H. A. Du Nguyen, J. Yu, M. A. Lebdeh, M. Taouil, S. Hamdioui, F. Catthoor. A classification of memory-centric computing. J. Emerg. Technol. Comput. Syst., 16(2), January 2020.
- [PIM2] O. Mutlu, S. Ghose, J. Gómez-Luna, R. Ausavarungnirun, A modern primer on processing in memory, 2021, arXiv:2012.03112.
- [PIM3] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, O. Mutlu: Benchmarking a New Paradigm: Experimental Analysis and Characterization of a Real Processing-in-Memory System. IEEE Access 10: 52565-52608 (2022)
- [PIM4] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, A. Fedorova: A Case Study of Processing-in-Memory in off-the-Shelf Systems. USENIX Annual Technical Conference 2021: 117-130
- [PIM5] A. Baumstark, M Attahir Jibril, K. Sattler: Accelerating Large Table Scan using Processing-In-Memory Technology, 2nd Workshop on Novel Data Management Ideas on Heterogeneous Hardware Architectures (NoDMC) at BTW 2023
- [PIM6] A. Baumstark, M. Attahir Jibril, K. Sattler: Adaptive Query Compilation with Processing-in-Memory, In Proc. of Int. Workshop on Big Data Management on Emerging Hardware (HardBD), April 2023

Further reading

- [PIM7] H. Kang, Y. Zhao, G. E. Blelloch, L. Dhulipala, Y. Gu, C. McGuffey, P. B. Gibbons: PIM-tree: A Skew-resistant Index for Processing-in-Memory. Proc. VLDB Endow. 16(4): 946-958 (2022)
- [PIM8] H. Kang, P. B Gibbons, G. E Blelloch, L.n Dhulipala, Y. Gu, and C. McGuffey. 2021. The Processing-in-Memory Model. In Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures. 295–306.
- [PIM9] Donghun Lee, Jinin So, Minseon Ahn, Jong-Geon Lee, Jungmin Kim, Jeonghyeon Cho, Oliver Rebholz, Vishnu Charan Thummala, Ravi Shankar JV, Sachin Suresh Upadhya, Mohammed Ibrahim Khan, Jin Hyun Kim: Improving In-Memory Database Operations with Acceleration DIMM (AxDIMM). DaMoN 2022: 2:1-2:9