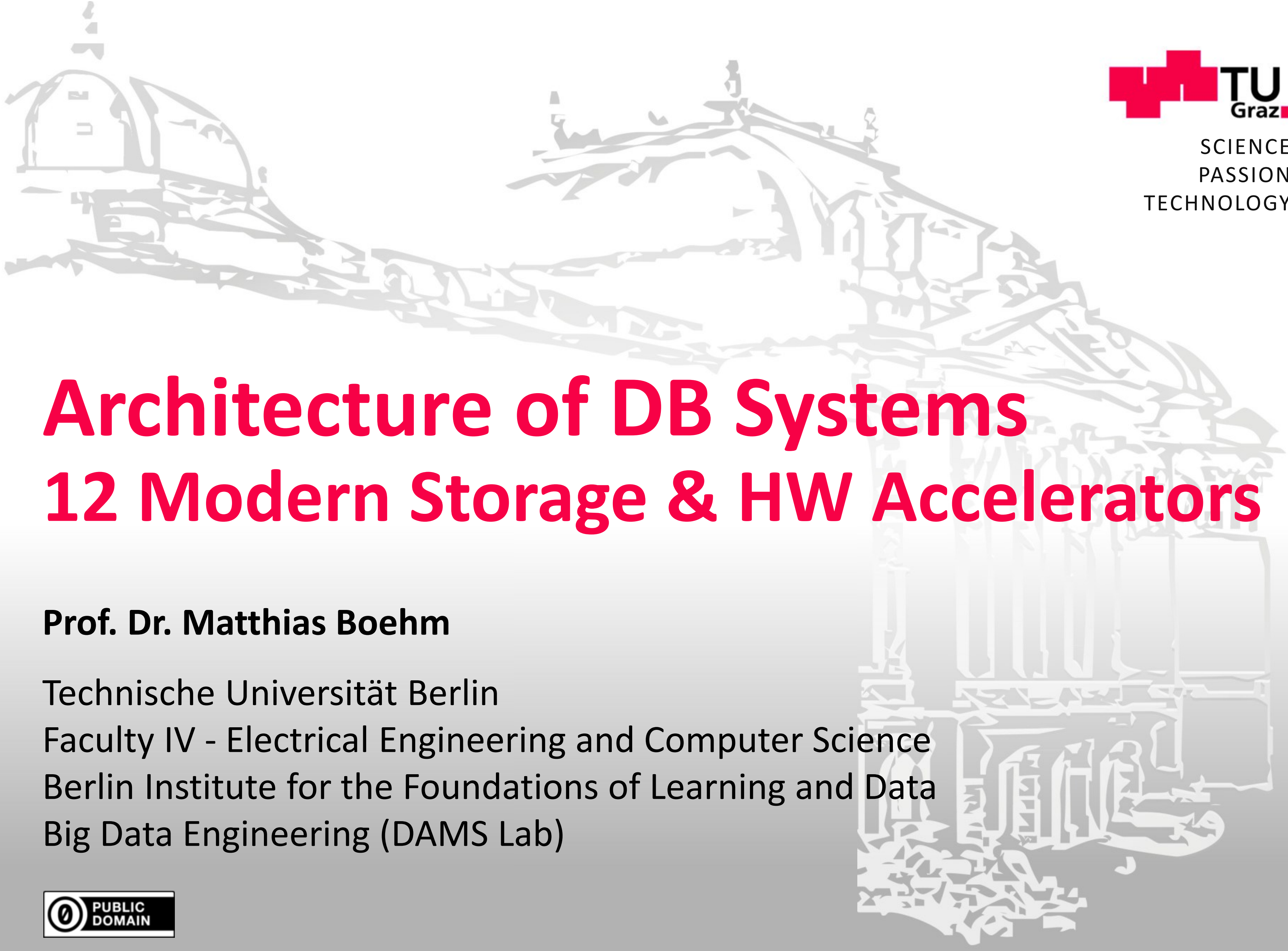


TU Graz

SCIENCE
PASSION
TECHNOLOGY

Architecture of DB Systems
12 Modern Storage & HW Accelerators

Prof. Dr. Matthias Boehm

Technische Universität Berlin
Faculty IV - Electrical Engineering and Computer Science
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: Dec 06, 2023

ISDS

Announcements/Org

- **#1 Lecture Format**
  - Introduction virtual, remaining lectures blocked **Dec 04 - Dec 07**
  - Optional attendance
  - **Hybrid**, in-person but live-streaming / video-recorded lectures
    - **HS i10** + Zoom: https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09

Agenda

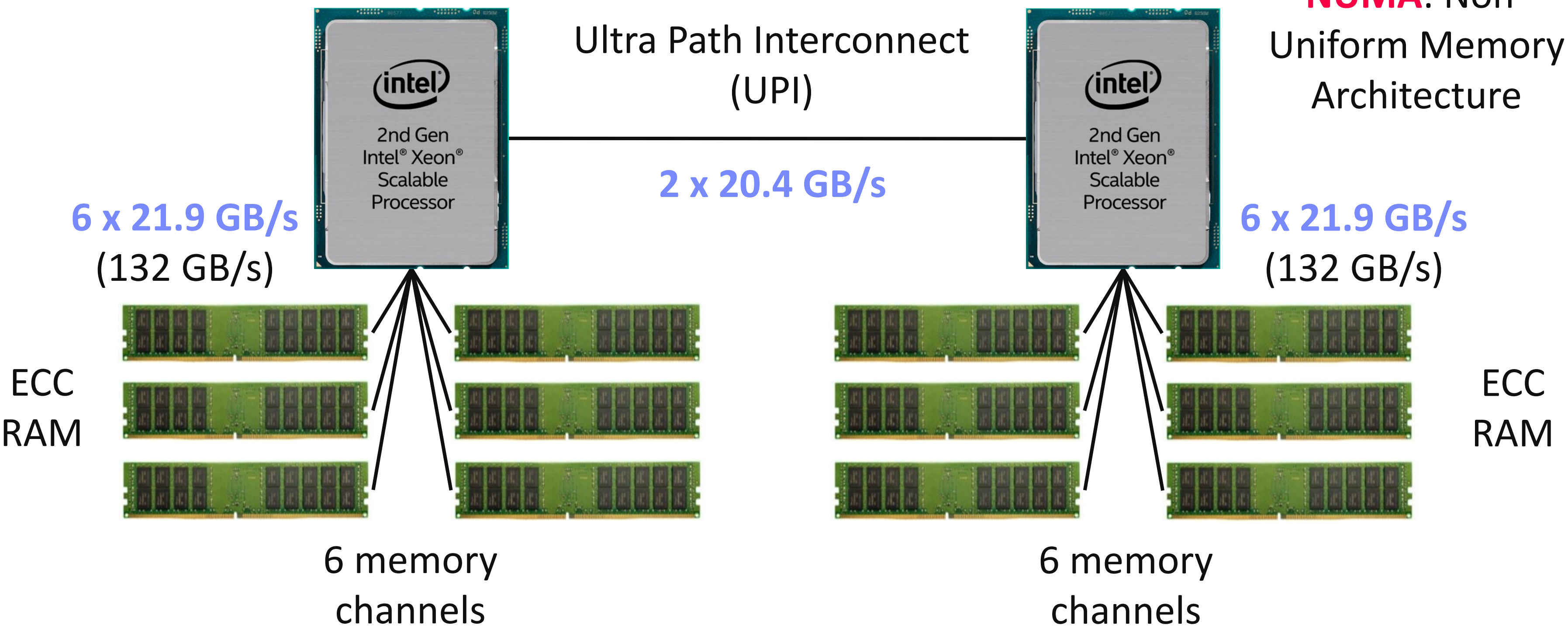
- **Recap: Basic HW Background**
- **Compute: DBMS on GPUs, FPGAs, ASICs**
- **Memory: DBMS on Non-volatile Memory**
- **Storage: DBMS on Computational Storage**

Recap: Basic HW Background

Basic CPU/Memory Architecture

[https://en.wikichip.org/wiki/intel/xeon_gold/6238r]

- **Example DM Cluster** (scale-up)
  - Scale-up Intel Xeon Gold 6238R @ 2.2-4 Ghz (2 x 28 pcores, **2 x 56 vcores**)
  - **768 GB** HPE DDR4 RAM @ 2.933 GHz (12 x 64GB 2Rx4 PC4-2933Y-R)


intel
2nd Gen Intel® Xeon® Scalable Processor
intel
2nd Gen Intel® Xeon® Scalable Processor

Ultra Path Interconnect (UPI)

2 x 20.4 GB/s

6 x 21.9 GB/s (132 GB/s)

6 x 21.9 GB/s (132 GB/s)

ECC RAM

ECC RAM

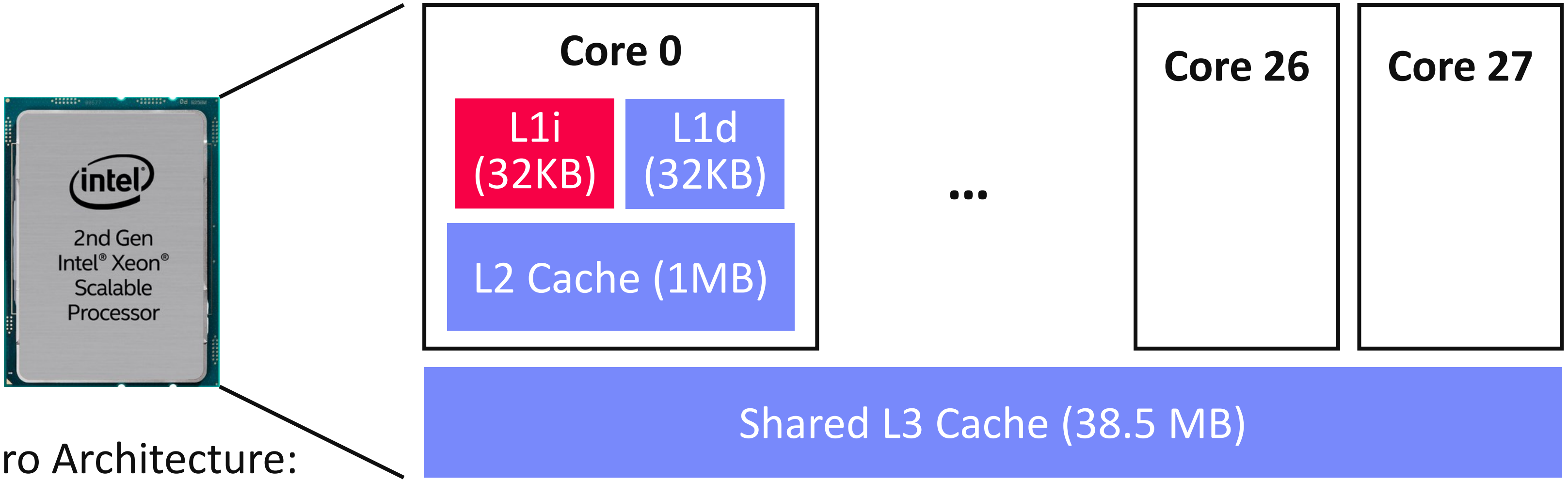
6 memory channels

6 memory channels

NUMA: Non-Uniform Memory Architecture

Basic CPU/Memory Architecture, cont.

- **Example DM Cluster**
  - Scale-up Intel Xeon Gold 6238R @ 2.2-4 GHz (2 x 28 pcores, **2 x 56 vcores**)
  - **768 GB** HPE DDR4 RAM @ 2.933 GHz (12 x 64GB 2Rx4 PC4-2933Y-R)



Core 0

L1i (32KB)

L1d (32KB)

L2 Cache (1MB)

…

Core 26

Core 27

Shared L3 Cache (38.5 MB)

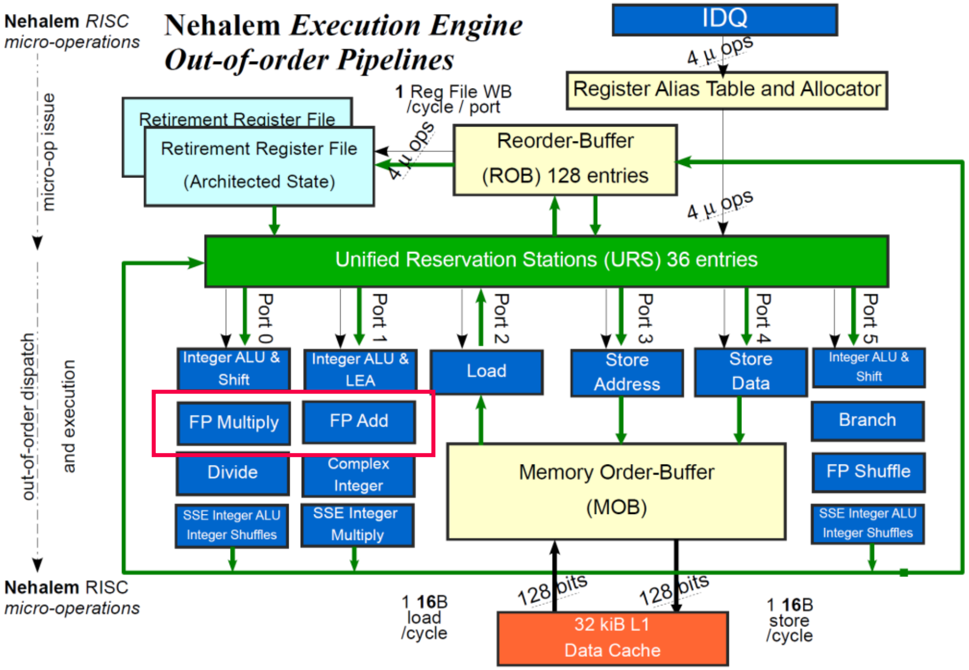
Micro Architecture:
Cascade Lake '19

Why do we need a cache hierarchy?

- **Cache Coherence Protocols** (e.g., dictionary, snooping)

CPU (Core) Microarchitecture

- **Example Nehalem**
  - **Frontend:** Instruction Fetch, Pre-Decode, and Decode
  - **Backend:** Rename/Allocate, Scheduler, Execute
  - Out-of-Order Execution Engine (128b FP Mult/Add)

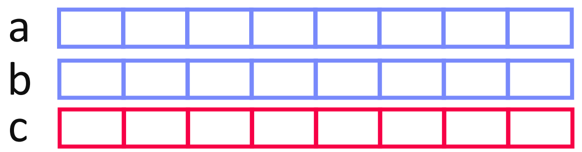

Nehalem RISC micro-operations
Nehalem Execution Engine Out-of-order Pipelines
IDQ
4 µ ops
Register Alias Table and Allocator
1 Reg File WB /cycle / port
Retirement Register File
Retirement Register File (Architected State)
4 µ ops
Reorder-Buffer (ROB) 128 entries
4 µ ops
micro-op issue
Unified Reservation Stations (URS) 36 entries
Port 0
Port 1
Port 2
Port 3
Port 4
Port 5
Integer ALU & Shift
Integer ALU & LEA
Load
Store Address
Store Data
Integer ALU & Shift
FP Multiply
FP Add
Branch
Divide
Complex Integer
Memory Order-Buffer (MOB)
FP Shuffle
SSE Integer ALU Integer Shuffles
SSE Integer Multiply
SSE Integer ALU Integer Shuffles
out-of-order dispatch and execution
Nehalem RISC micro-operations
1 16B load /cycle
128 bits
128 bits
1 16B store /cycle
32 kiB L1 Data Cache

[M. E. Thomadakis: The Architecture of the Nehalem Processor and Nehalem EP SMP Platforms, Report, 2010]

- **SIMD Processing**
  - Single-instruction, multiple data
  - Process the same operation on multiple elements at a time
  - Data/instruction parallelism
  - Example: **VFMADD132PD**

2009 Nehalem: **128b** (2xFP64)
2012 Sandy Bridge: **256b** (4xFP64)
2017 Skylake: **512b** (8xFP64)

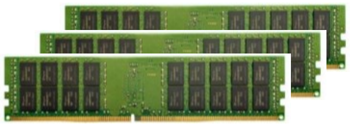
```
c = _mm512_fmadd_pd(a, b);
```



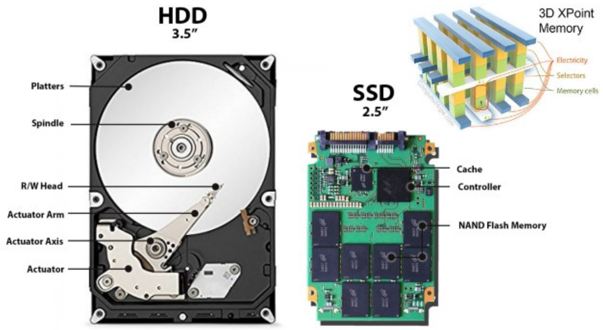
Basic Storage Architecture

Perf ←→ Cost per GB

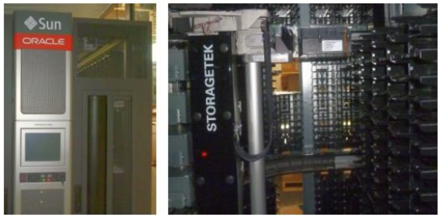
- **Primary Storage**
  - Main Memory (volatile, often charge-based)
  - Capacitors leak charge → periodic refresh (**~64ms**)



- **Secondary Storage** (non-volatile storage)
  - **HDD:** hard disk drive (magnetic, rotating platters)
  - **SSD:** solid-state drives (flash memory)
  - **NVM:** non-volatile memory (flash/resistive)



- **Tertiary Storage** (archival mass storage)
  - Optical disks (special materials), Magneto-optical disks
  - Tape drives: magnetic tape w/ high capacity cartridges



50PB tape library

Why do we need a storage hierarchy?

[Thomas Hahmann, Hans Weber, Erhard Diedrich, Gunter Schreier: SENTINEL-1 AND SENTINEL-3-OLCI PAC AT **DLR**, ESA-SP 722, **2013**]

HW Challenges

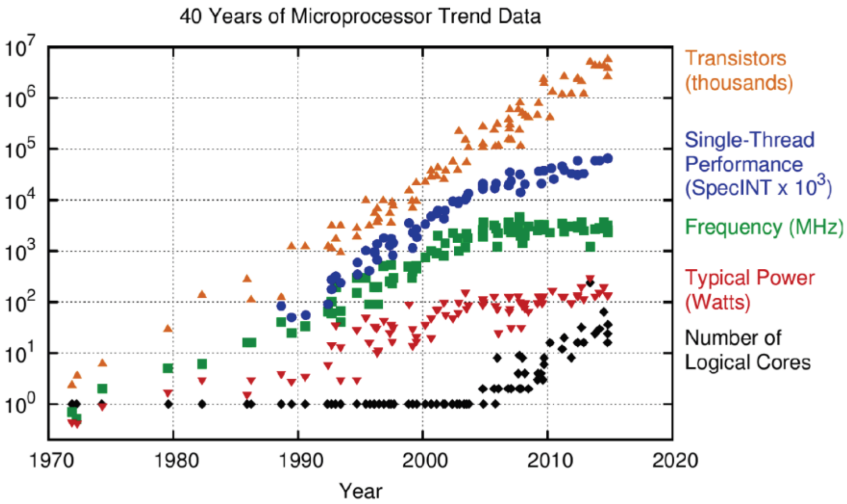
[S. Markidis, E. Laure, N. Jansson, S. Rivas-Gomez and S. W. D. Chien: Moore's Law and Dennard Scaling]

Moore's Law and Dennard Scaling

- **#1 End of Dennard Scaling** (~2005)
  - **Law:** power stays proportional to the area of the transistor
  - Ignored leakage current / threshold voltage → **increasing power density S^2** (power wall, heat) → stagnating frequency

$P = \alpha\, CFV^2$ (**power density 1**)
(P .. Power, C .. Capacitance, F .. Frequency, V .. Voltage)

- **#2 End of Moore's Law** (~2010-20)
  - **Law:** #transistors/performance/ CPU frequency doubles every 18/24 months
  - Original: # transistors per chip doubles **at constant costs**
  - Now increasing costs (10/7/5nm)



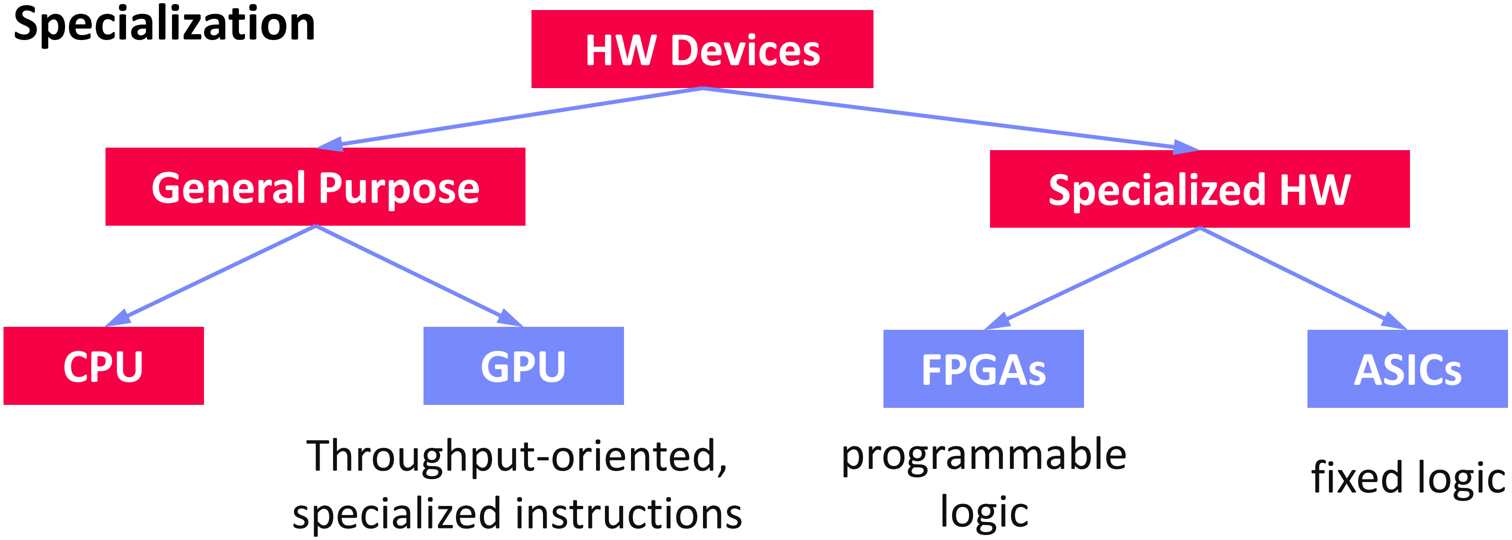
- **#3 Amdahl's Law** (speedup limitations)

→ **Consequences: Dark Silicon and Specialization**

Compute:
DBMS on GPUs, FPGAs, ASICs

Towards Specialized Hardware

- **HW Specialization**

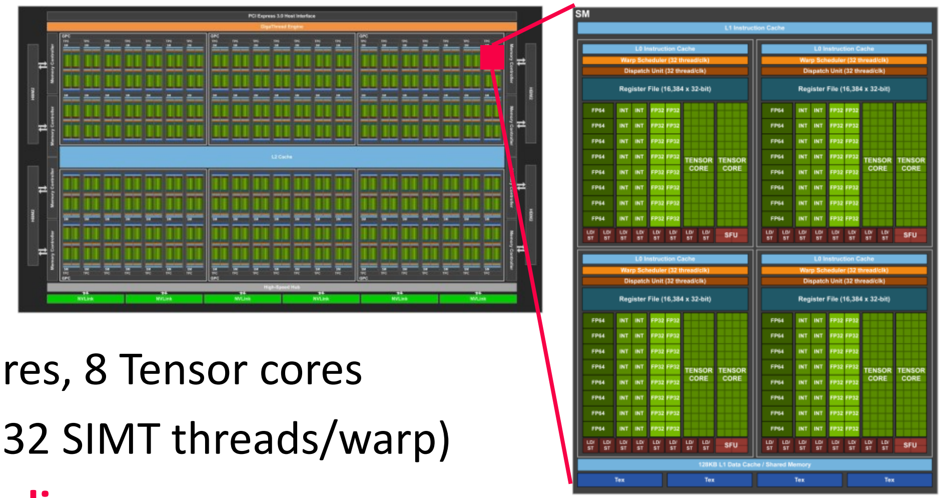


- **Interconnect**
  - PCI Express (PCI-e): v3 x16: **16GB/s**, v4 x16: **32GB/s**
  - New link technologies: GPUs via NVLink, FPGAs via QPI/UPI, all via OpenCAPI

- **Additional Specialization**
  - Data Transfer & Types: e.g., compressed representations
  - Near-Data Processing: e.g., operations in memory or storage

Graphics Processing Units (GPUs)

- **GPU Characteristics**
  - Separate (PCIe, NVLink) or integrated devices
  - High compute throughout (e.g., NVIDIA V100 FP64: **7.8 TFLOPs**, FP32: **15.7 TFLOPs,** DL FP16: **125 TFLOPs**)
  - High bandwidth memory (e.g., NVIDIA V100 16/32 GB (**900 GB/s**)



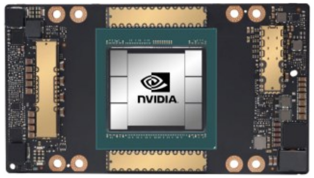
PCI Express 3.0 Host Interface
GigaThread Engine
L2 Cache
NVLink
High-Speed Hub
SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

- **V100 Architecture**
  - 6 GPU Processing Clusters with 7x2 SMs
  - Streaming Multiprocessors
    - 32 FP64 cores
    - 64 FP32 cores, 64 INT32 cores, 8 Tensor cores
    - Thread blocks → N warps (32 SIMT threads/warp)
    - Control flow causes **thread divergence** (V100 new **`__syncwarp()`** primitive)

[NVIDIA Tesla V100 GPU Architecture, Whitepaper, **Aug 2017**]

Graphics Processing Units (GPUs), cont.

- **NVIDIA Ampere A100**
  - 7nm, 8 GPC x 8 TPC * 2 SM = 128 SMs, 40GB HBM
  - FP32 **19.5 TFLOPs**, FP16: 78 TFLOPs, BF16: 39 TFLOPs
  - New generation of “TensorCores” (FP64, **new data types:** TF32, BF16)
  - Fine-grained **sparsity exploitation**
  - Multi-instance **GPU (MIG) virtualization**: up to 7 virtual GPU instances
  - Link technologies: **NVLink 3** (25GB/s bidirectional) x 12 links = 600GB/s



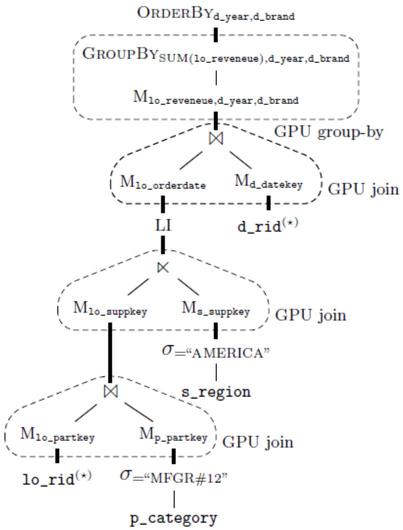
- **NVIDIA Hopper H100**
  - 7nm, 7/8 GPC x 9 TPC * 2 SM = 114/144 SMs, 80GB HBM
  - FP32 **66.9 TFLOPs**, FP16: 134 TFLOPs, BF16: 134 TFLOPs
  - **FP8 TensorCore** 1979 TFLOPs / 3958 TFLOPs (sparse), INT8
  - **Dedicated Transformer Engine** (hybrid FP8 and FP16)
  - Improved link technologies (NVLink 4, NVSwitch 3, PCIe 5)

GPU Join Processing

- **Data Transfer**
  - Transfer often ignored although dominating
  - **Overlapped transfer and compute** to reach PCIe max throughput (non-trivial)
  - **CUDA Unified Virtual Addressing** (UVA)
- **Join Implementation**
  - Column-oriented (ID, RID) representation
  - Partitioned hash join (**06 Query Processing**)
- **Query Plan Decomposition**
  - Most time spent in joins and group-by
  - GPU kernels for **binary join** and **group-by** (only one hash table on small GPU device mem)
  - Other operators on host CPU

[Tim Kaldewey, Guy M. Lohman, René Müller, Peter Benjamin Volk: GPU join processing revisited. **DaMoN@SIGMOD 2012**]

[René Müller, Tim Kaldewey, Guy M. Lohman, John McPherson: WOW: what the world of (data) warehousing can learn from the World of Warcraft. **SIGMOD 2013**]


$\text{ORDERBY}_{\texttt{d\_year,d\_brand}}$
$\text{GROUPBY}_{\texttt{SUM(lo\_reveneue),d\_year,d\_brand}}$
$\text{M}_{\texttt{lo\_reveneue,d\_year,d\_brand}}$
GPU group-by
$\bowtie$
$\text{M}_{\texttt{lo\_orderdate}}$
$\text{M}_{\texttt{d\_datekey}}$
GPU join
LI
$\texttt{d\_rid}^{(*)}$
$\ltimes$
$\text{M}_{\texttt{lo\_suppkey}}$
$\text{M}_{\texttt{s\_suppkey}}$
GPU join
$\sigma_{=\text{“AMERICA”}}$
s_region
$\bowtie$
$\text{M}_{\texttt{lo\_partkey}}$
$\text{M}_{\texttt{p\_partkey}}$
GPU join
$\texttt{lo\_rid}^{(*)}$
$\sigma_{=\text{“MFGR\#12”}}$
p_category

GPU Query Compilation

- **Recap: Query Compilation**
  - Modern in-memory DBMS → **CPU/memory efficiency crucial**
  - Generation of data-centric tuple-at-a-time pipelines (often LLVM)
  - How to **generate code for GPUs** and HW accelerators in general?

- **Voodoo**
  - **Vector algebra** w/ knobs for parallelization (e.g., load, add/mult, scatter/gather, zip, fold)
  - OpenCL/interpreter backends for CPU/GPU

[Holger Pirk, Oscar R. Moll, Matei Zaharia, Sam Madden: Voodoo - A Vector Algebra for Portable Database Performance on Modern Hardware. **PVLDB 9(14) 2016**]

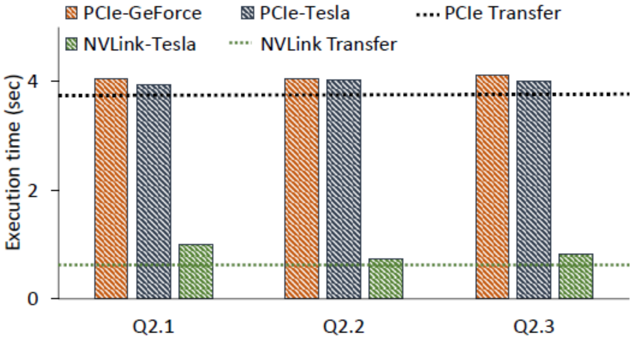
- **HorseQC**
  - **Vectorized execution** of fused operators
  - Data-parallel prefix sums for write positions
  - Single data pass w/ exploitation of GPU **memory hierarchy** (CPU, PCIe, global, shared)

[Henning Funke, Sebastian Breß, Stefan Noll, Volker Markl, Jens Teubner: Pipelined Query Processing in Coprocessor Environments. **SIGMOD 2018**]

GPU Query Processing on new Link Technologies

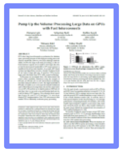
- **New Link Technologies**
  - NVLink for CPU-mem/GPU and GPU-GPU communication
  - OpenCAPI for cache-coherent accelerator integration

- **#1 H²TAP (EPFL)**
  - HTAP system (OLTP on CPU, OLAP on GPU)
  - LLVM code generation for operator fusion
  - Lazy transfers (UVA) and transfer sharing

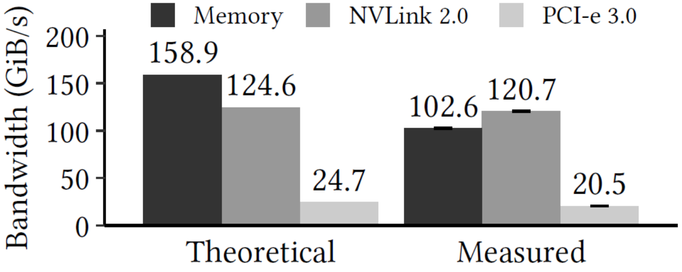


[Aunn Raza et al: GPU-accelerated data management under the test of time. **CIDR 2020**]

- **#2 Experimental Analysis (TU Berlin)**
  - In-depth analysis of NVLink 2 vs PCIe 3



[Clemens Lutz et al.: Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. **SIGMOD 2020** **(best paper award)**]

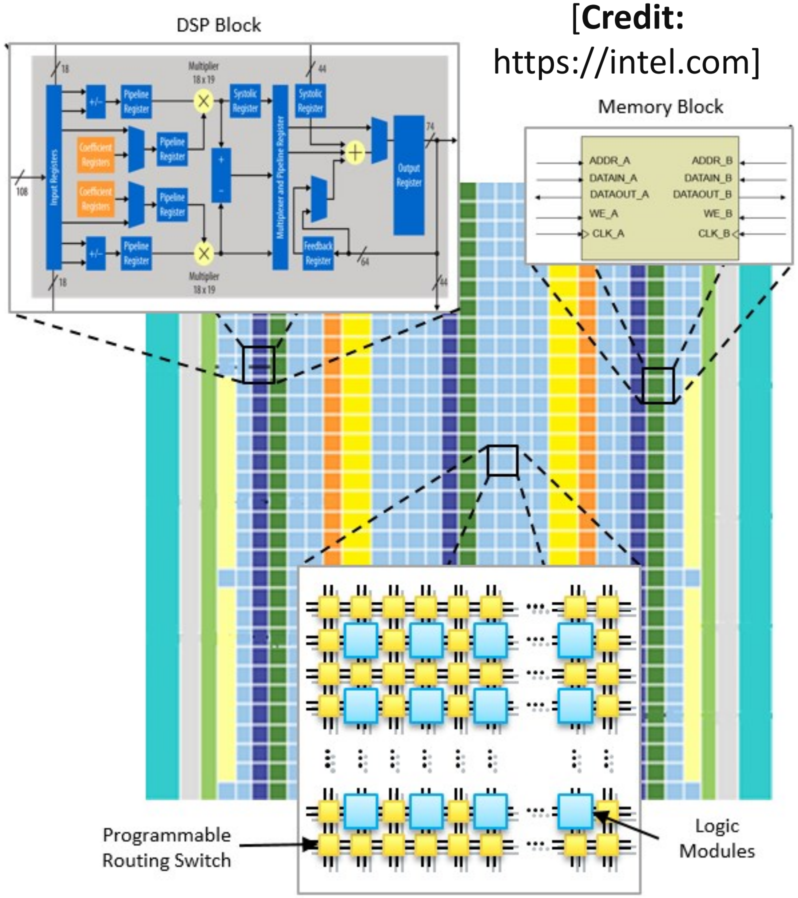


Field-Programmable Gate Arrays (FPGAs)

- **FPGA Characteristics**
  - Integrated circuit that allows **configuring custom hardware designs**
  - Reconfiguration in <1s
  - HW description language: e.g., VHDL, Verilog (RTL)

- **FPGA Components**
  - **#1 lookup table** (LUT) as logic gates
  - **#2 flip-flops** (registers)
  - **#3 interconnect network**
  - Additional memory and DSP blocks

- **Examples**
  - Intel (Altera) Stratix 10 SoC FPGA
  - Xilinx Virtex UltraSCALE+

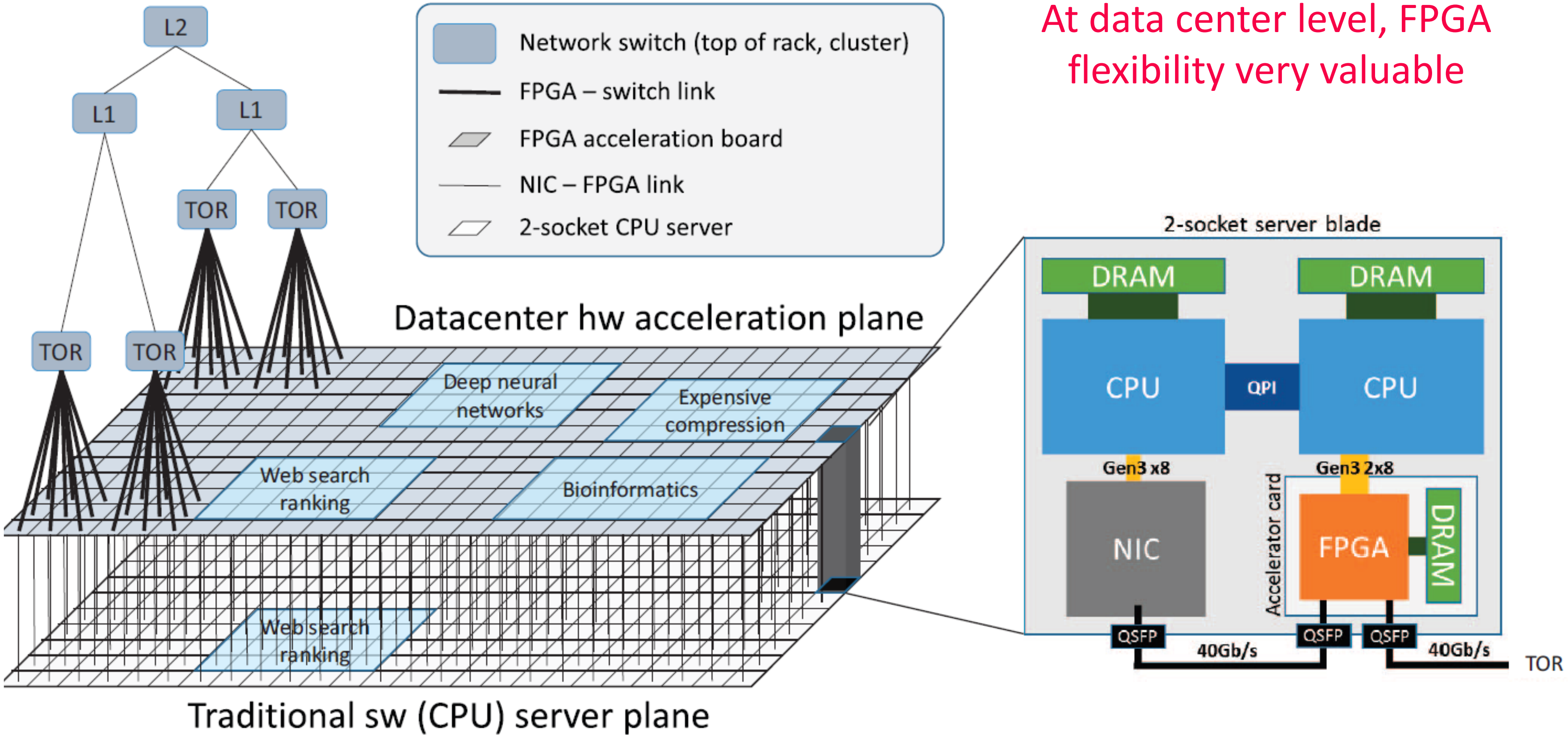


[Credit: https://intel.com]

FPGAs in Microsoft’s Data Centers

- **Microsoft Catapult**
  - Dual-socket Xeon w/ PCIe-attached FPGA
  - Pre-filtering neural networks, compression, and other workloads

[Adrian M. Caulfield et al.: A cloud-scale acceleration architecture. **MICRO 2016**]



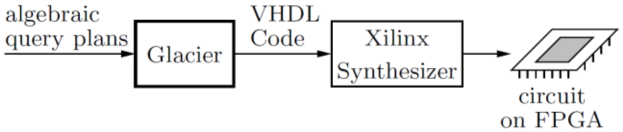
At data center level, FPGA flexibility very valuable

FPGA Query Processing

- **Motivation**
  - FPGA on data path from network/disk/stream to CPU for offloading
  - Performance and energy efficiency (e.g., Netezza → IBM, discontinued)

- **FPGAs for Data Processing**
  - Specialized stream operators (e.g., window median – sorting network)
  - **Glacier:** Library and query compiler for continuous (streaming) queries

[René Müller, Jens Teubner, Gustavo Alonso: Data Processing on FPGAs. **PVLDB 2(1) 2009**]



[René Müller, Jens Teubner, Gustavo Alonso: Streams on Wires - A Query Compiler for FPGAs. **PVLDB 2(1) 2009**]

- **Many Specialized Operators**
  - Frequent item set mining
  - Regular expression matching
  - Complex event processing

[Louis Woods, Jens Teubner, Gustavo Alonso: Complex Event Detection at Wire Speed with FPGAs. **PVLDB 3(1) 2010**]

Application-Specific Integrated Circuit (ASICs)

- **Gamma Database Machine**
  - Long history of HW-accelerated DB workloads
  - Unsuccessful due to need for continuous improvements (Moore’s Law)

[David J. DeWitt: DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems. **IEEE Trans. Computers 28(6) 1979**]

- **#1 TUD Tomahawk**
  - Specialized ops (**sorted set intersection**)

[Oliver Arnold et al: An application-specific instruction set for accelerating set-oriented database primitives. **SIGMOD 2014**]

- **#2 Oracle DAX**
  - M7 chip with 8 Data Analytics engines
  - Specialized ops (**decompress**, **selections**)

[Kathirgamar Aingaran et al.: M7: Oracle's Next-Generation Sparc Processor. **IEEE Micro 35(2) 2015**]

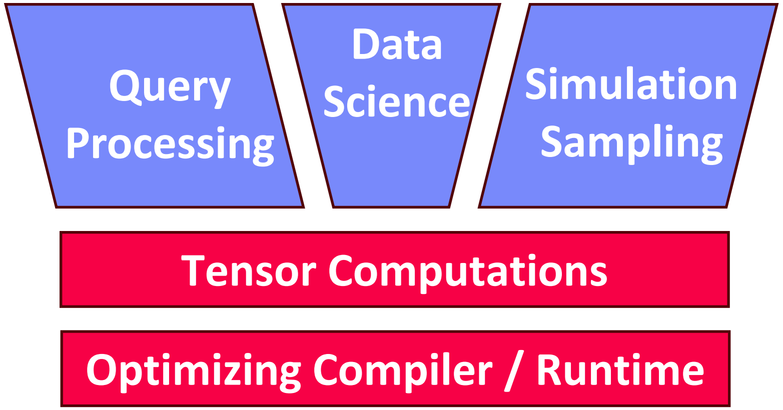
- **#3 Oracle DPU (RAPID**, discontinued**)**
  - Data Processing Unit (shared-mem many core)
  - Data movement system, atomic TX engine
  - Partitioning and **partitioned hash join** (kernels)

[Cagri Balkesen et al.: RAPID: In-Memory Analytical Query Processing Engine with Extreme Performance per Watt. **SIGMOD 2018**]

Optimizing Tensor Computations

[SIGMOD'23 Tutorial]

- **#1 Simplicity**
  - Coarse-grained frame/tensor data structures and operations
  - Reduced system infrastructure complexity (boundary crossing)



Query Processing

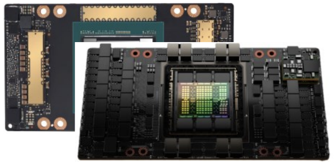
Data Science

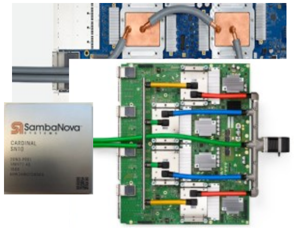
Simulation Sampling

Tensor Computations

Optimizing Compiler / Runtime

- **#2 Reuse of Compiler/Runtime Techniques**
  - Reuse of commonly used techniques
  - Generality over hand-crafted, specialized systems and algorithms





- **#3 Performance and Scalability**
  - Leverage HW Accelerators and distributed backends
    ➔ Increasing specialization and rapid evolution
  - Homogeneous arrays and simple parallelization

Build Libraries for Tensor Ops on HW X once and reuse

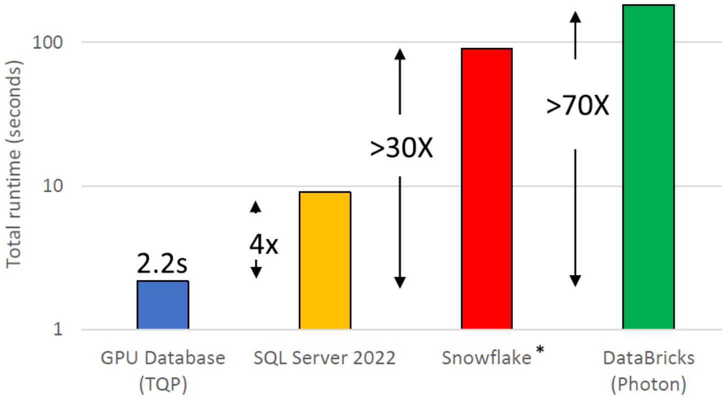
Example: Tensor Query Processor (TQP)

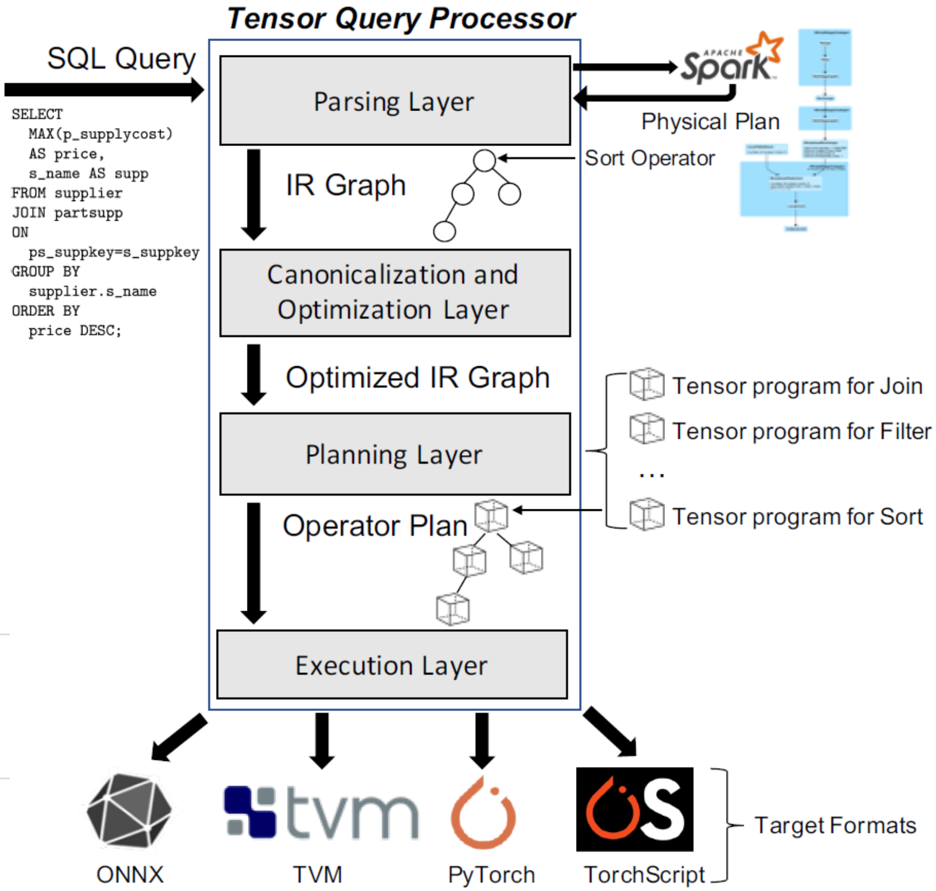
- **System Architecture**

[Dong He, […], Konstantinos Karanasos, Matteo Interlandi: Query Processing on Tensor Computation Runtimes. **PVLDB 2022**]

- **Preliminary Performance**

[https://mboehm7.github.io/resources/sigmod2023_tutorial.pdf]





```
SELECT
  MAX(p_supplycost)
  AS price,
  s_name AS supp
FROM supplier
JOIN partsupp
ON
  ps_suppkey=s_suppkey
GROUP BY
  supplier.s_name
ORDER BY
  price DESC;
```

Memory:
DBMS on Non-volatile Memory

HOW TO BUILD A
NON-VOLATILE MEMORY
DATABASE SYSTEM

JOY ARULRAJ & ANDY PAVLO
CARNEGIE MELLON UNIVERSITY

SIGMOD 2017, CHICAGO

[Joy Arulraj, Andrew Pavlo: How to Build a Non-Volatile Memory Database Management System. **SIGMOD 2017**]

Data Structure Engineering for Byte-Addressable Non-Volatile Memory

Ismail Oukid (TU Dresden & SAP SE), Wolfgang Lehner (TU Dresden)

SIGMOD Tutorial, Chicago, IL, May 14th, 2017

[Ismail Oukid, Wolfgang Lehner: Data Structure Engineering For Byte-Addressable Non-Volatile Memory. **SIGMOD 2017**]

Overview Non-Volatile Memory (NVM)

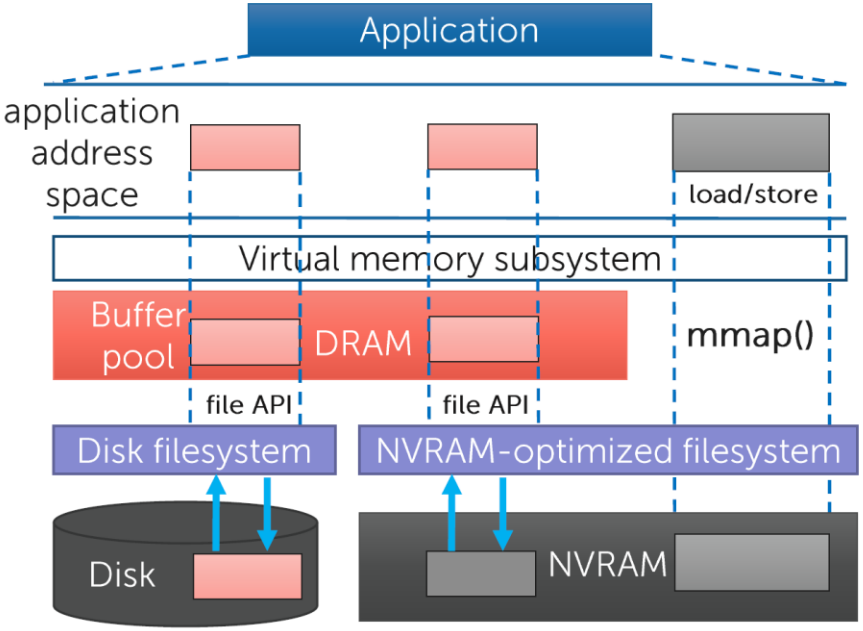
- **DRAM Scaling Limits**
  - Decreasing feature sizes → **increasing failure rates** (e.g., RowHammer)
  - DRAM DIMM **cost per GB** scales super-linearly (~ 9.5x for 4x capacity)
  - DRAM **major energy consumer** (charge-based, requires refresh ~64ms)

- **Non-Volatile Memory (NVM)**
  - Aka Storage-Class Memory (SCM) → non-volatile storage
  - **Higher capacity**, lower energy, and cheaper than DRAM (~3x bandwidth)
  - **Byte-addressable** read and write (unlike SSD/HDD)
  - **High random write throughput**
  - **Read/write asymmetry** and **wear leveling**

Overview Non-Volatile Memory (NVM), cont.

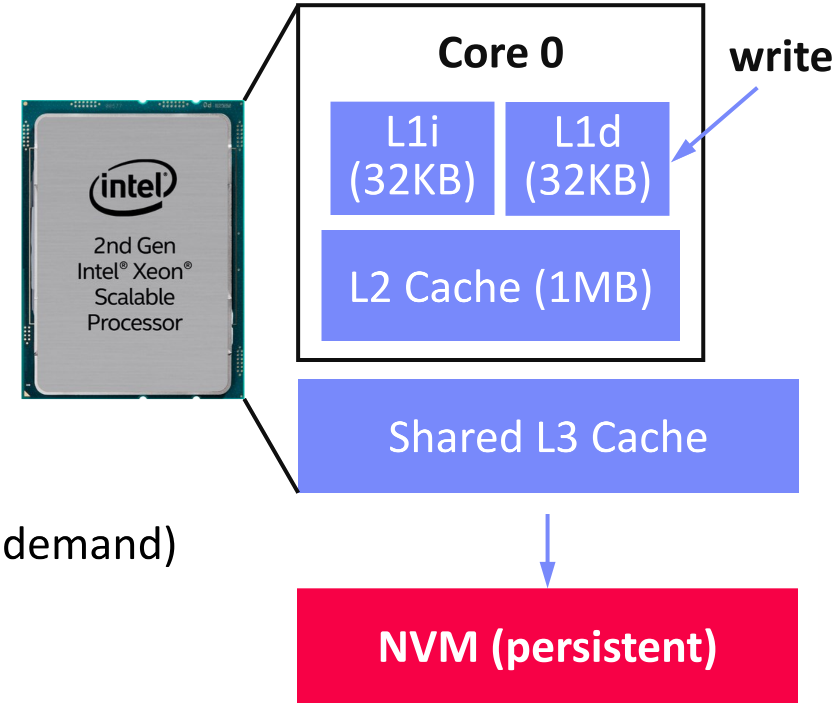
- **Different System Integration Approaches**
  - NVDIMM-N: DIMM with flash storage and DRAM
  - NVDIMM-F: DIMM with flash storage
  - NVDIMM-P: DIMM with NVM technologies (e.g., PCM)

- **File System Support**
  - Access NVRAM via mmap (zero-copy via bypassing OS page cache)
  - Dedicated file systems (NOVA, PMFS, SCMFS)



NVM Durability Guarantees

- **NVM Durability Challenges**
  - Little control when data is persisted
  - **Examples:** CPU Cache evictions, memory reordering, partial writes, (at cache line granularity)



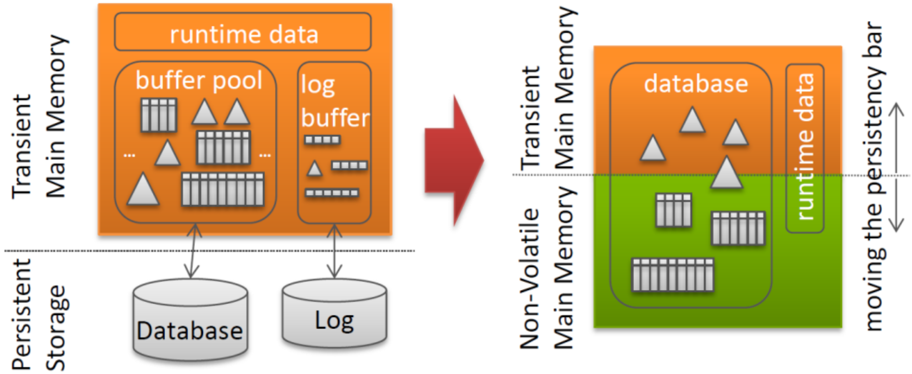
- **NVM-relevant Instructions**
  - **CLFLUSH** (flush cache line from every level of the cache hierarchy, write on demand)
  - **CLWB** (write back cache line)
  - **MFENCE** (serializes global visibility), **SFENCE** (store fence), **LFENCE** (load fence)
  - **MOVNT** (write bypassing the caches)
  - Special handling for draining store buffers

Additional Challenges
Persistent Memory Leaks
Persistent Data Corruption
Persistent Memory Management

NVM Logging and Recovery

[Ismail Oukid, Wolfgang Lehner, Thomas Kissinger, Thomas Willhalm, Peter Bumbulis: Instant Recovery for Main Memory Databases. **CIDR 2015**]

- **SOFOFT: DB Recovery on NVM**
  - Simulated DBMS on NVM
  - Instant recovery by trading TX throughput vs recovery time (**% of data structures on SCM**)
  - No need for logging (everything is durable)
  - Continue TXs on recovery



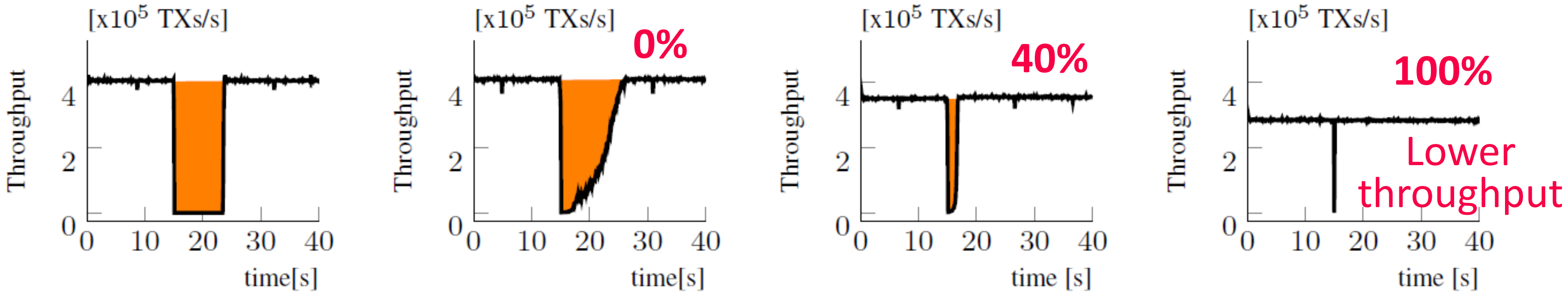
a) Traditional Architecture

b) SCM-enabled Architecture

- **Recovery Simulation**

Wait for Rebuild

Continue TX, Async Rebuild



0%

40%

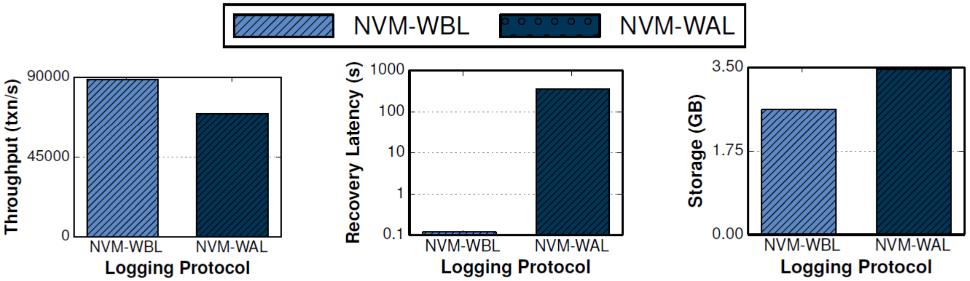
100%

Lower throughput

NVM Logging and Recovery, cont.

[Joy Arulraj, Matthew Perron, Andrew Pavlo: Write-Behind Logging. **PVLDB 10(4) 2016**]

- **Motivation**
  - NVM with **higher write throughput**
  - **Smaller gap sequential vs random** write throughput
  - **Byte-addressable NVM** (no need for pages)

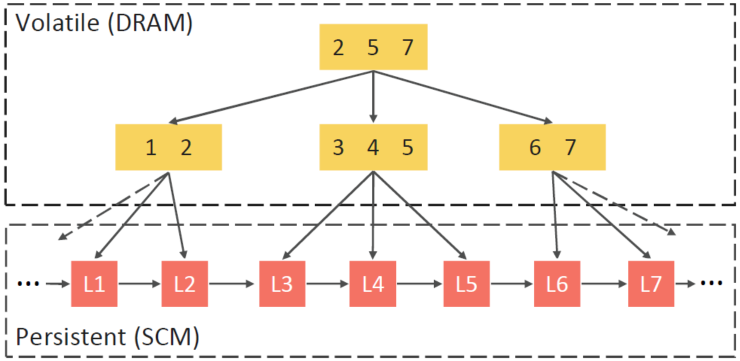


- **Write-Behind Logging**
  - Leverage byte-addressable NVM to reduce amount of logged data on changes
  - Dirty tuple table (DTT) for tracking changes → never written to NVM
  - Update persistent data (SCM) + DTT on commit, log change metadata
    - c_p timestamp – TXs durable on NVM → log entry
    - c_d timestamp – TX commit timestamp
  - Recovery: Ignore and GC TX changes in (c_p,c_d)

NVM Index Structures

- **FPTree** (Fingerprinting Persistent Tree)
  - **Selective persistence:** Hybrid B$^+$-Tree with volatile inner and durable leaf nodes
  - **Unsorted leaves** for reduce # of writes
  - **Fingerprinting** (1B hashes of in-leaf keys)
  - Recovery: rebuild inner nodes

[Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, Wolfgang Lehner: FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. **SIGMOD 2016**]



Inner nodes in DRAM for better performance

Leaves in SCM to ensure durability

[BTW'2019]

- **BzTree**
  - **Latch-free** B-tree for NVM
  - Multi-word compare-and-swap w/ persistence guarantees (PMwCAS)

[Joy Arulraj, Justin J. Levandoski, Umar Farooq Minhas, Per-Åke Larson: BzTree: A High-Performance Latch-free Range Index for Non-Volatile Memory. **PVLDB 11(5) 2018**]

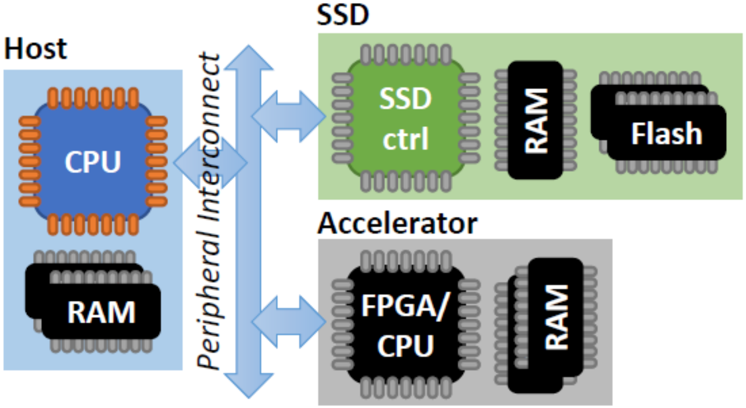
Storage:
DBMS on Computational Storage

Overview Computational Storage

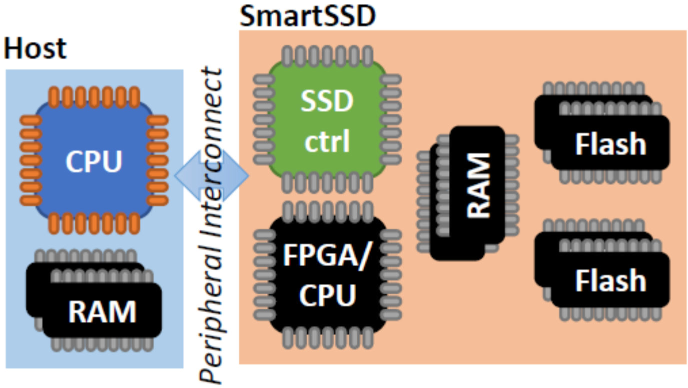
[Antonio Barbalace, Jaeyoung Do: Computational Storage: Where Are We Today?, **CIDR 2021**]

- **Different SSD Architectures**
  - (a) SSDs (solid-state drive)
  - (b) Smart SSDs
  - (c) Accelerators in the SSD I/O path (e.g., Netezza)

(a) SSD w/ controller



(b) SSD w/ co-processors



- **Background: FTL (flash translation layer)**
  - SW or HW (controller) FTL for mapping of logical to physical addresses, wear-leveling, ECC, and bad block management

Query Processing on (Smart)SSDs

- **Query Processing on SSDs**
  - Wide spectrum of work (index structures, out-of-core operations)
  - Handle **read/write asymmetry** and other properties → perf/energy efficiency

- **OLAP Query Processing on SmartSSDs**
  - Session-based protocol for SATA/SAS (PCIe) interface OPEN/CLOSE/GET (results, 10ms poll)
  - Coordinator/worker threads
  - Flash pages pinned into DRAM
  - **Projection, selection and aggregation**

[Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, David J. DeWitt: Query processing on smart SSDs: opportunities and challenges. **SIGMOD 2013**]

- **OLTP Query Processing on SmartSSDs**
  - Small random writes cause huge write overhead on SSDs (**write amplification**)
  - Flash-append features, **delta record area** via controlled data placement

[Sergey Hardock, Ilia Petrov, Robert Gottstein, Alejandro P. Buchmann: From In-Place Updates to In-Place Appends: Revisiting Out-of-Place Updates on Flash. **SIGMOD 2017**]

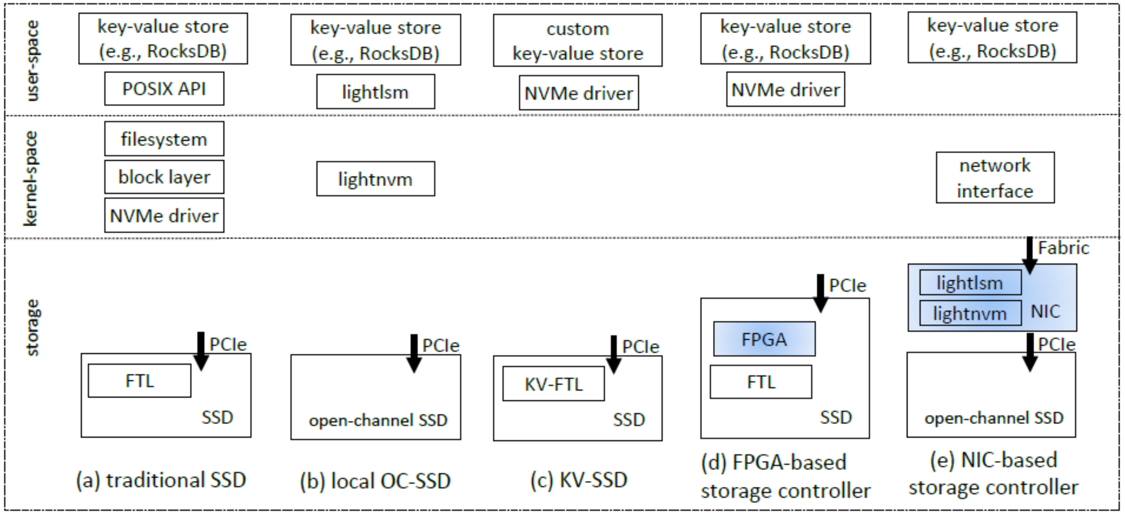
KV Stores on SSDs

- **Open Channel SSDs**
  - Storage devices that let hosts control data placement and I/O scheduling

[Ivan Luiz Picoli, Niclas Hedam, Philippe Bonnet, Pinar Tözün: Open-Channel SSD (What is it Good For). **CIDR 2020**]

- **FTL Specialization for Log-structured Merge Trees (LSM)**

[Ivan Luiz Picoli, Philippe Bonnet, Pinar Tözün: LSM Management on Computational Storage. **DaMoN@SIGMOD 2019**]



- **Batched Writes @ MS**
  - Log structuring in SSD controller

[Jaeyoung Do, David B. Lomet, Ivan Luiz Picoli: Improving CPU I/O Performance via SSD Controller FTL Support for Batched Writes. **DaMoN@SIGMOD 2019**]

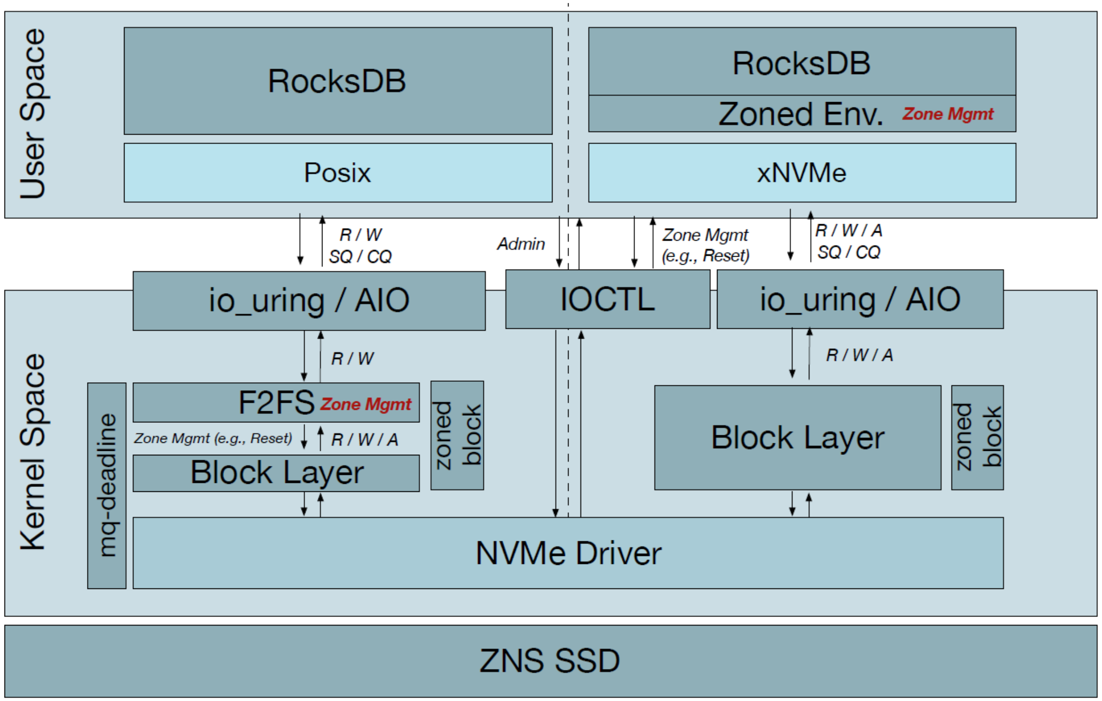
Zoned Namespaces (ZNS)

[Javier González: Zoned Namespaces - Use Cases, Standard and Linux Ecosystem, **SDC SNIA EMEA 20**]

Zoned Namespaces
Use Cases, Standard and Linux Ecosystem

- **Zoned Namespace**
  - **Motivation:** log-structured merge trees (KV-store), log-structured FS, with user-space data placement, garbage collection, metadata
  - Divide logical block device (LBA) into fixed-size zones, sequential write per zone

- **Linux Stack**
  - Zoned FS (e.g., F2FS)
  - ZNS-specific features (append, ZRWA, Simple Copy)
  - 2 I/O paths
    - Zoned FS
    - Zoned Apps

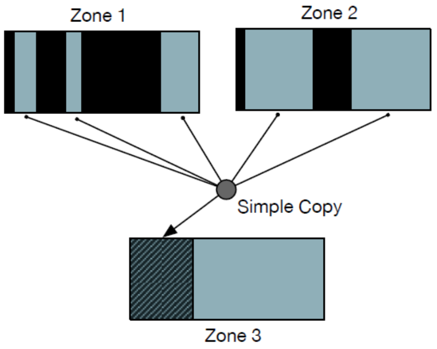


[Dev Purandare et al.: Append is Near: Log-based Data Management on ZNS SSDs, **CIDR 2022**]

Copy in Memory / Storage

[Javier González: Zoned Namespaces - Use Cases, Standard and Linux Ecosystem, **SDC SNIA EMEA 20**]

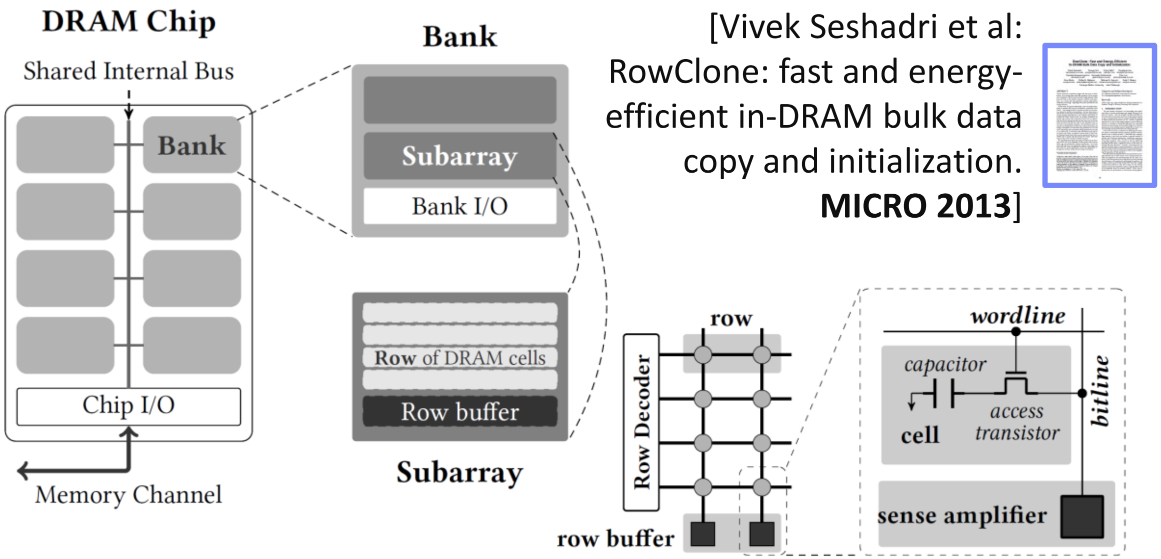
- **ZNS Simple Copy**
  - Offload copy operation to SSD
  - Data moved directly by SSD controller
  - No PCIe data transfer, no CPU cycles spent
  - Applications: garbage collection
  - Under discussion in NVMe



multiple sources

single target

- **DRAM RowClone**
  - Copy memory rows via row buffer
  - Within/across banks
  - Also used for row initialization (e.g., `memset(0)`)
  - Low-cost HW ext.



[Vivek Seshadri et al: RowClone: fast and energy-efficient in-DRAM bulk data copy and initialization. **MICRO 2013**]

Problems and Challenges

- **#1 Caching**
  - Buffer pools and distributed caching unaware of push-down
  - Prefiltering in storage destroys caching and reuse → cold analysis tasks

- **#2 Granularity**
  - Page-oriented (fixed size) buffer pool for sequential I/O
  - Pre-filtering in storage destroys blocks → variable length

- **#3 Practicability**
  - Programmability/usability
  - Multi-tenant environments
  - Security concerns

[Antonio Barbalace, Jaeyoung Do: Computational Storage: Where Are We Today?, **CIDR 2021**]

Summary and **Q&A**

- **Recap: Basic HW Background**
- **Compute: DBMS on GPUs, FPGAs, ASICs**
- **Memory: DBMS on Non-volatile Memory**
- **Storage: DBMS on Computational Storage**

Thanks

- **Project Submission**
  - Submission deadline: Jan 21 → **Jan 31 EOD**, oral exams Feb 05-14 / Mar 25-29
  - Submission link: https://tinyurl.com/44w4kdk7 (incl exam registration)
  - **Perf target: score >= # physical-cores / 2**