***N**on-**V**olatile **M**emory in DBMS*

Outline

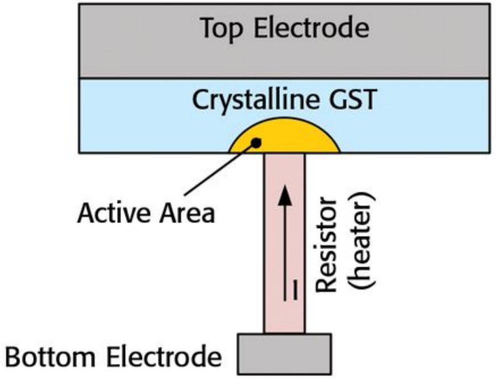
1. Introduction to NVM
2. NVM Programming Model
3. NVM-aware Database Architectures
4. Database Logging and Recovery
5. NVM-oriented Database Algorithms

<u>N</u>on-<u>V</u>olatile <u>M</u>emory

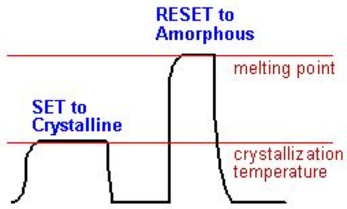
- *a.k.a.* **<u>S</u>torage <u>C</u>lass <u>M</u>emory** or **Persistent Memory**

- ***Non-volatile***
- ***Byte-addressable***
- ***Random access***
- ***Low read/write latency***

- Technologies
  - <u>P</u>hase <u>C</u>hange <u>M</u>emory
  - <u>R</u>esistive <u>RAM</u>
  - <u>M</u>agnetoresistive <u>RAM</u>
  - <u>S</u>pin-<u>T</u>ransfer <u>T</u>orque <u>RAM</u>
  - 3D XPoint ???

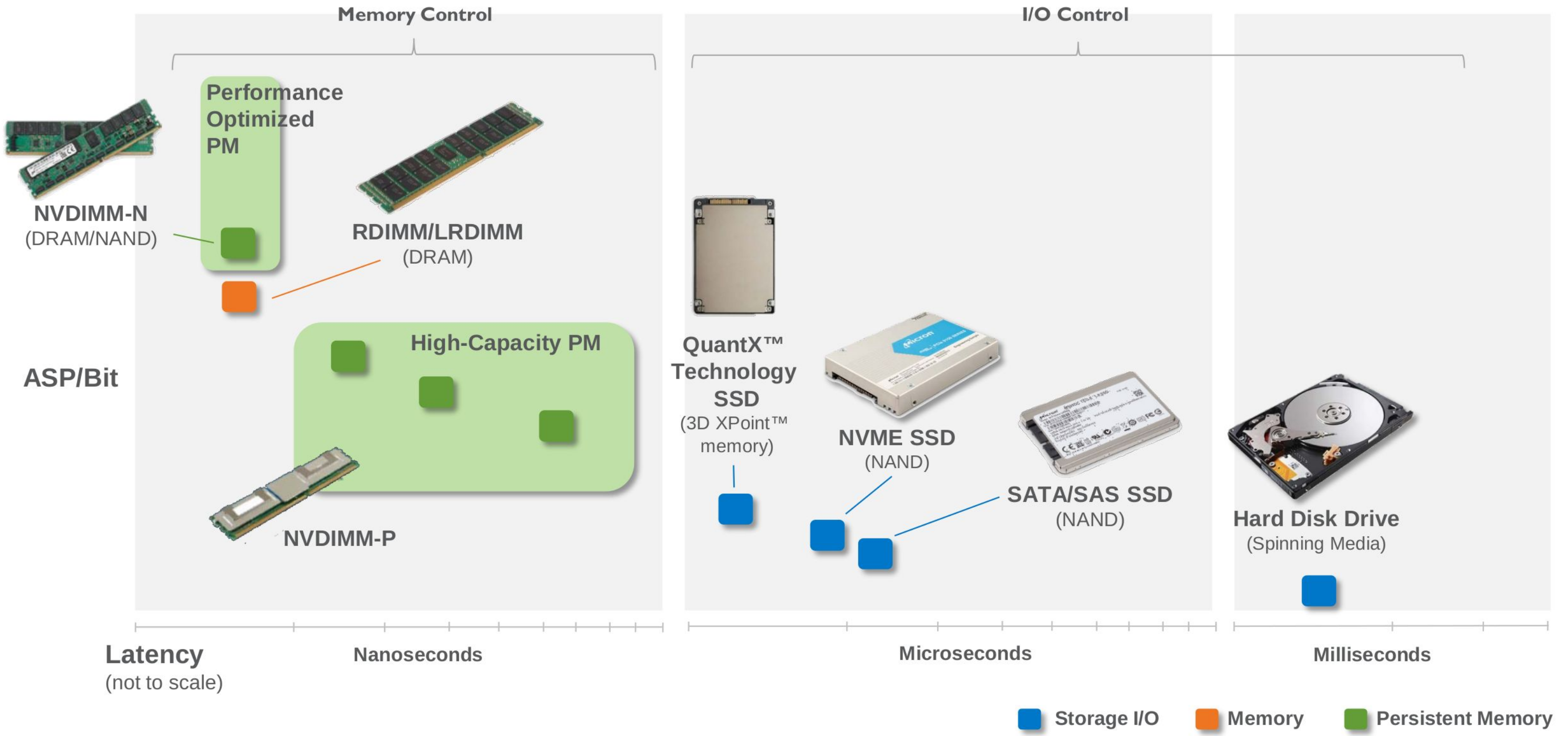


http://images.dailytech.com/nimage/Phase_Change_Memory_Diagram_Simple_Wide.png



http://img.tfd.com/cde/PCRESET.GIF

Price/Latency

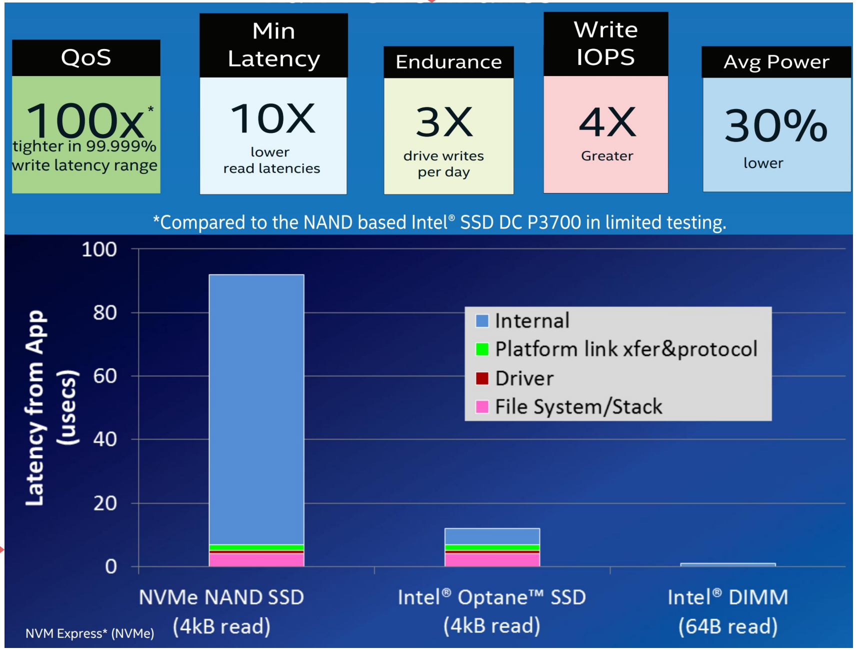


3D XPoint™

Marketing

- Intel® Optane™ SSD DC P4800X

Interface	PCIe 3.0 x4 NVMe
Capcity	375GB
Typical Latency (R/W)	<10μs
QoS(99.999%) (4K Random)	Queue Depth 1, R/W: < 60/100μs Queue Depth 16, R/W: < 150/200μs
Throughput (4K Random, QD 16)	R/W: up to 550/500k IOPS Mixed 70/30 R/W: up to 500k IOPS



DIMM version will be faster

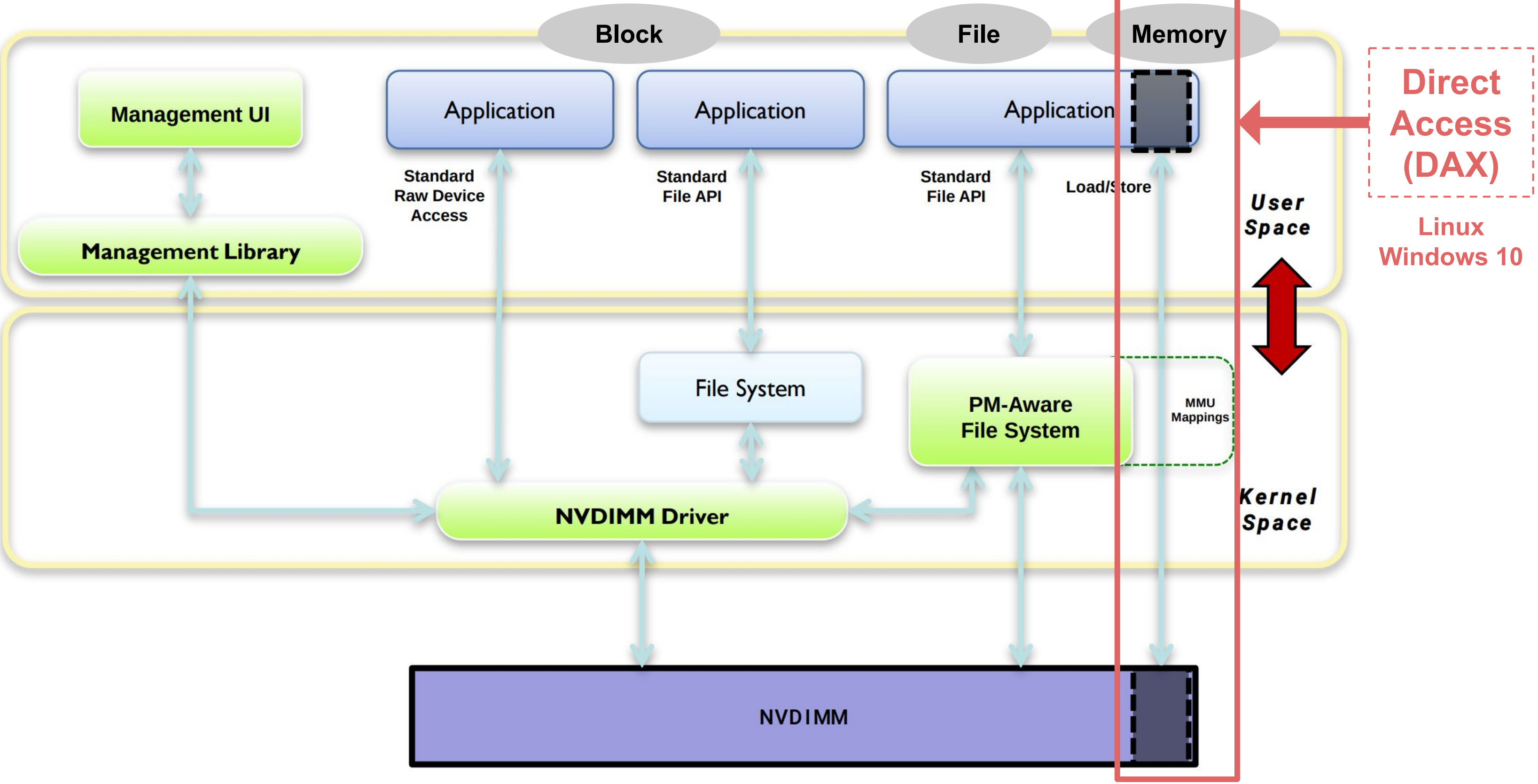
Outline

1. ~~Introduction to NVM~~
2. NVM Programming Model
3. NVM-aware Database Architectures
4. Database Logging and Recovery
5. NVM-oriented Database Algorithms

Programming Model: Challenges

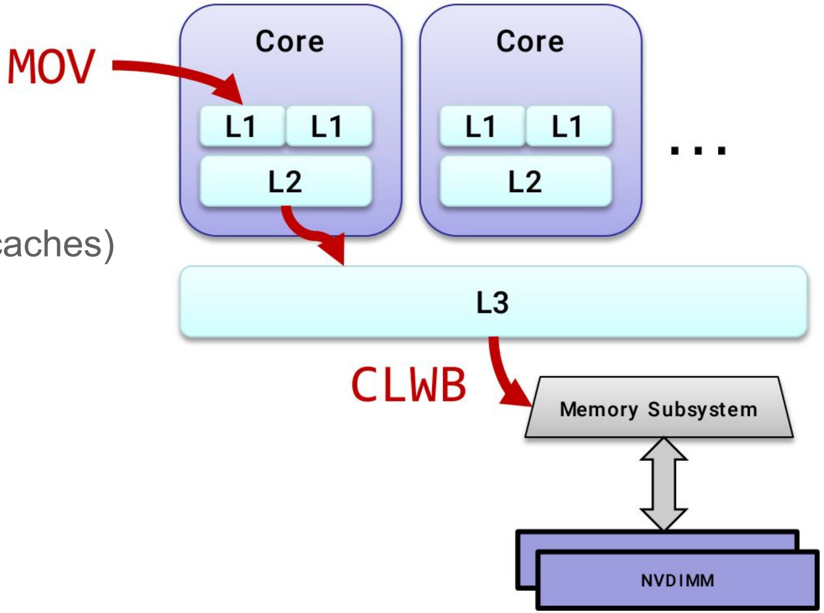
- Data Consistency
  - Between CPU and NVM: **multi-level caches/buffers**
  - Ensure writes to NVM are persisted
  - Traditional `msync` (`fsync` for `mmap`-ed file) is error-prone
- Data Recovery
  - Program restarts will invalidate all virtual pointers
  - Require persistent pointers
- Memory Leaks
  - `pmptr *p = pmalloc(pool, SIZE);` **/* power failure */** `persist(p);`
- Partial Writes
  - `persist(array);` → `store(array[0]);` **/* power failure */** `store(array[1]);`

SNIA NVM Programming Model



Related Instruction Set

- Cache line management
  - `CLFLUSH[OPT]`: write back with evicting
  - `CLWB` (new): write back without evicting
- Non-temporal write
  - `MOVNT*`: register ➞ memory (bypass L1/2/3 caches)
- Memory order
  - `MFENCE/SFENCE/LFENCE`
    - side effect: drain the CPU store buffer



Parallel Access

Be careful about multi-threading!

- Memory order

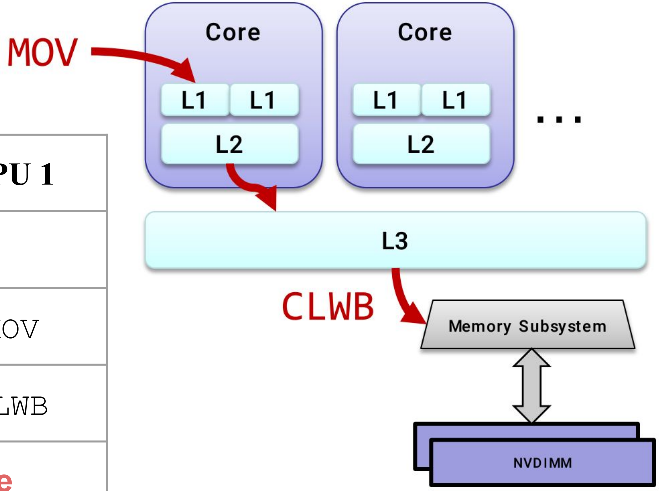
x = y = 0;	
CPU 0	**CPU 1**
x = 1; a = y;	y = 1; b = x;
assert(a == 1 || b == 1);	

CPU 0	**CPU 1**

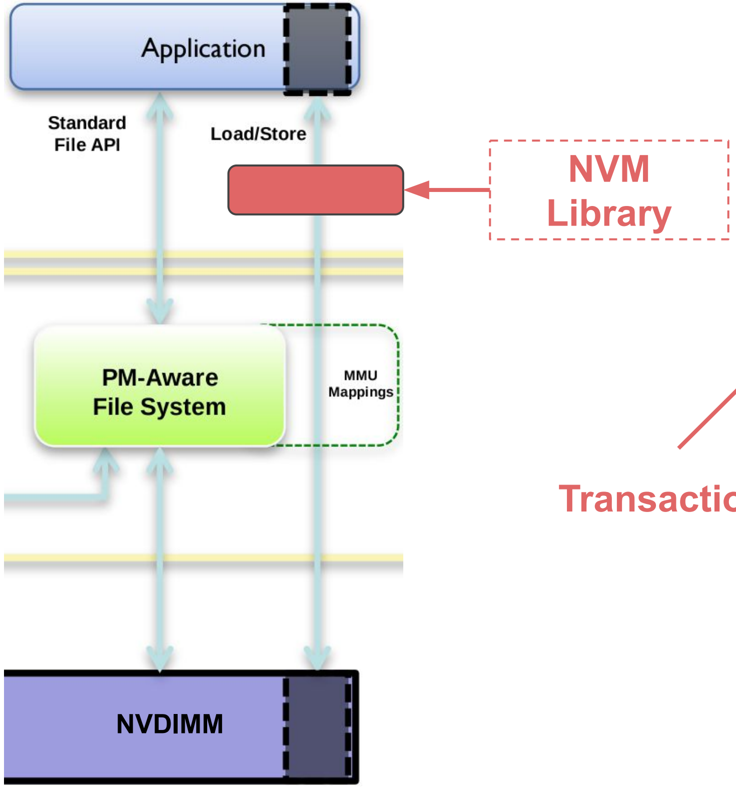
x = 1;
MFENCE
a = y;
	y = 1;
	MFENCE
	b = x;

- Premature read

CPU 0	**CPU 1**
MOV	
	MOV
	CLWB
power failure	
CLWB	



NVM Library for Memory-Mapped DAX



http://pmem.io (open-sourced by Intel)

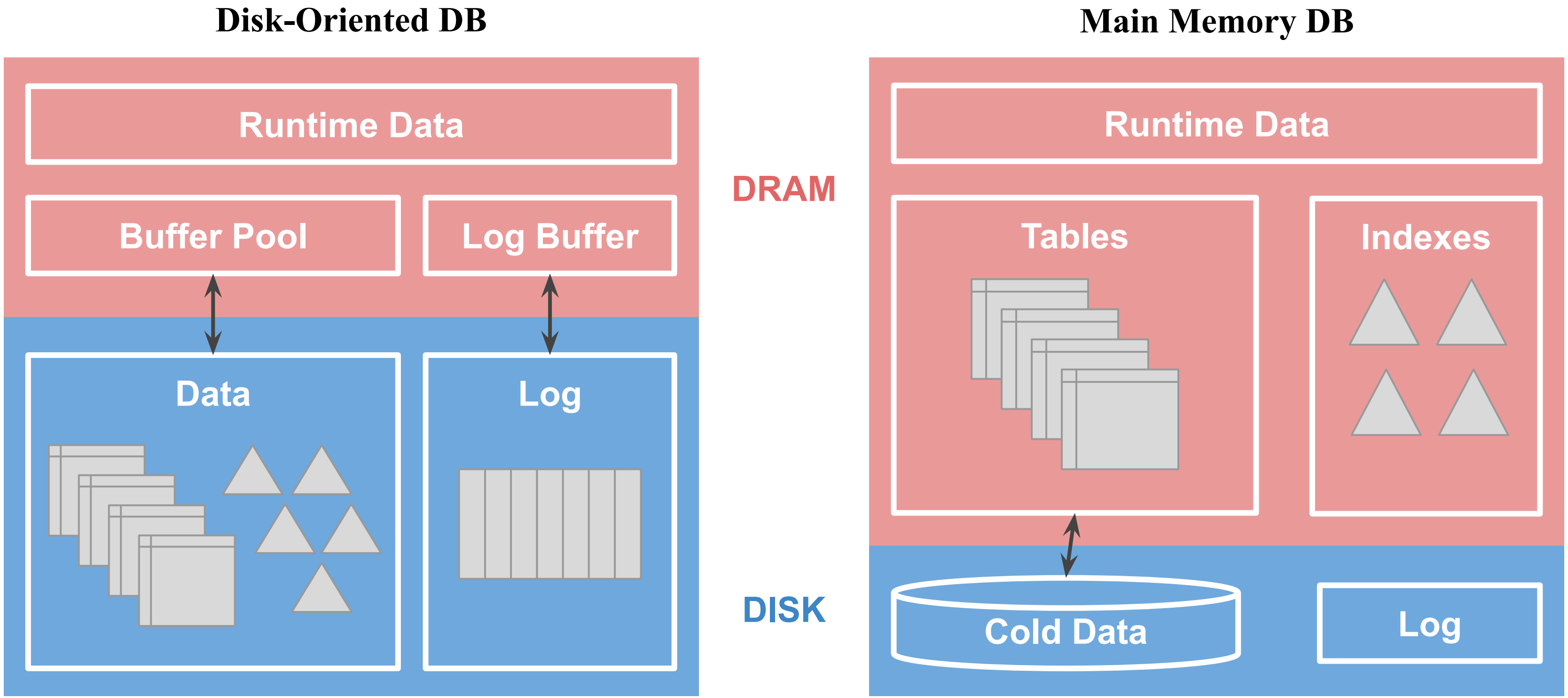
- libpmem: low level PM support
- libpmemobj: object store
- libpmemblk: array of blocks
- libpmemlog: log file
- libvmem: PM as volatile memory

Transactional

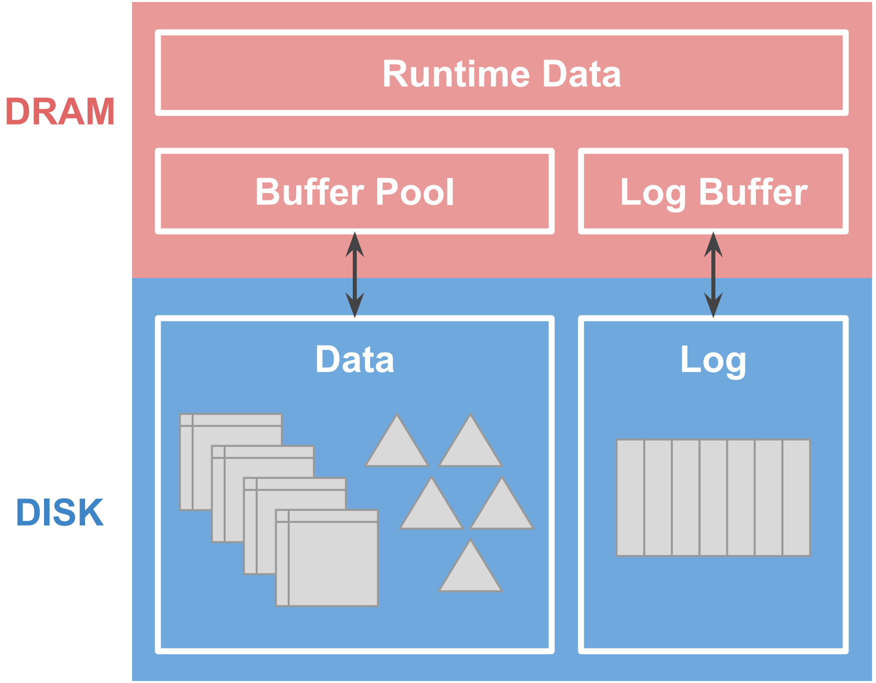
Outline

1. ~~Introduction to NVM~~
2. ~~NVM Programming Model~~
3. NVM-aware Database Architectures
4. Database Logging and Recovery
5. NVM-oriented Database Algorithms

Traditional DB Architecture



Disk-Oriented DB

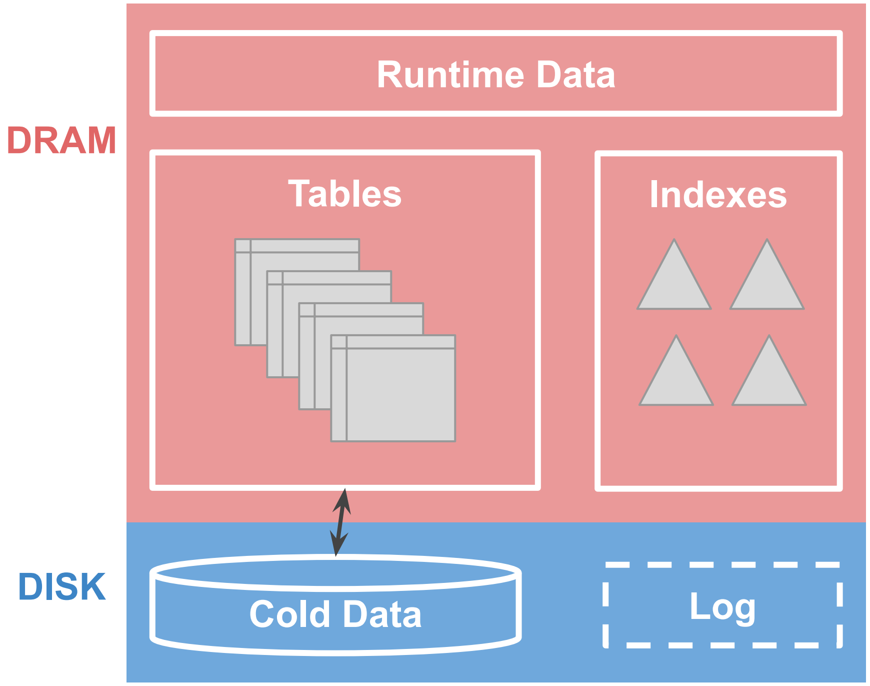


- Disk-resident **B**-trees/heap files
- **B**uffer management
- **L**ocking-based concurrency control
- **L**og-based recovery
- **L**atch-based multi-threading

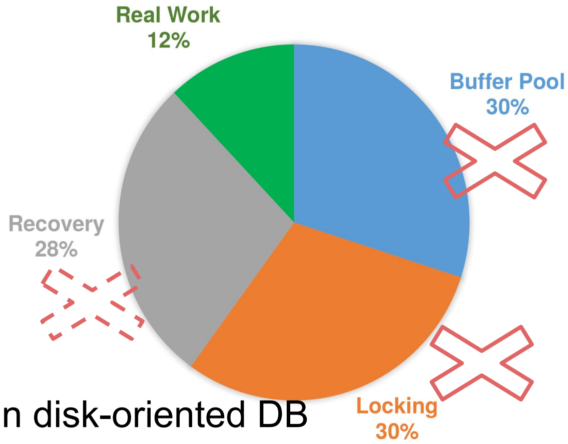
2B3L

OLTP Through the Looking Glass, and What We Found There. SIGMOD 2008

Main Memory DB

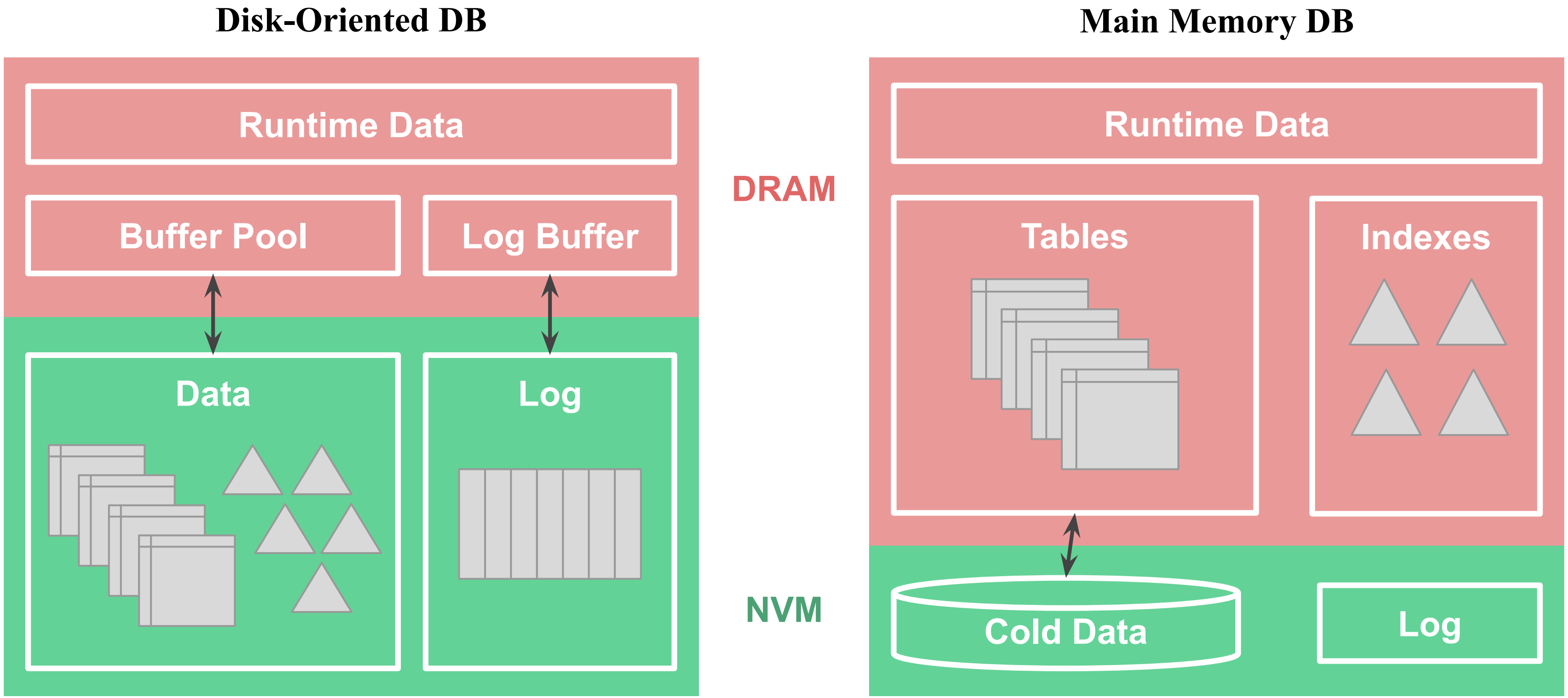


- Data lives in DRAM
  - Remove the buffer pool
  - Durability: logging + checkpoint
- Eliminate locking
  - MVCC, partitioned execution
- Memory resident indexes
  - Rebuild during the recovery



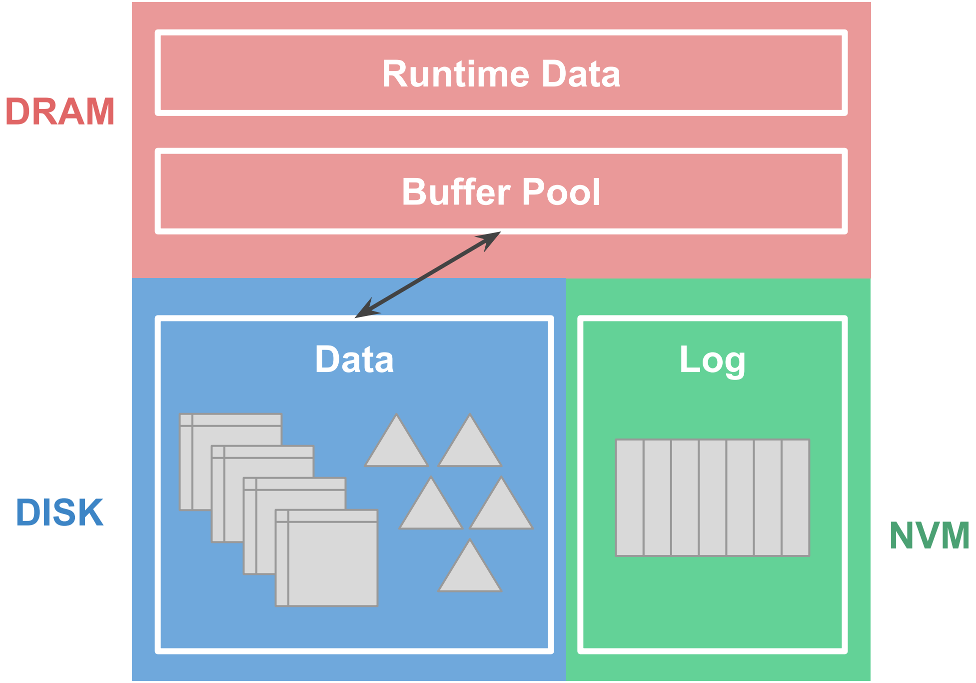
transactions in disk-oriented DB

Naïvely Replacing Disk with NVM



A Prolegomenon on OLTP Database Systems for Non-Volatile Memory. ADMS@VLDB 2014

Speeding Up Logging with NVM

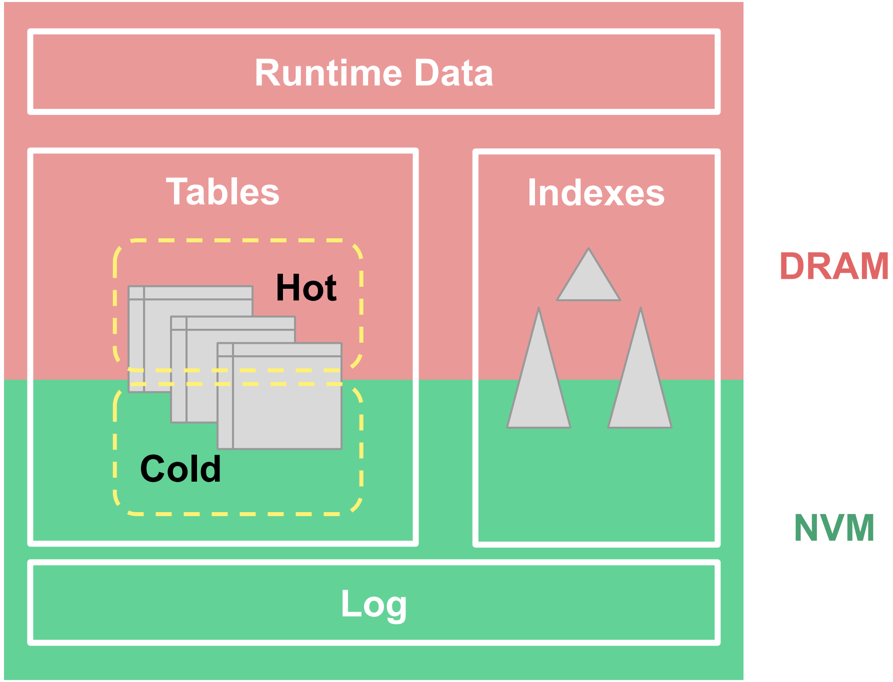


High Performance Database Logging using Storage Class Memory. ICDE 2011

Scalable Logging through Emerging Non-Volatile Memory. VLDB 2014

- Write log records directly to NVM
  - Be careful about partial writes
- Optionally, copy NVM log archives to disk asynchronously
- Distributed logging
  - Each txn writes to its own log space
  - Cheap random write & byte addressability
- Group commit can still be useful
  - Commit transactions in batches
  - Hide the latency of persist barriers

Persisting Part of the Index in NVM



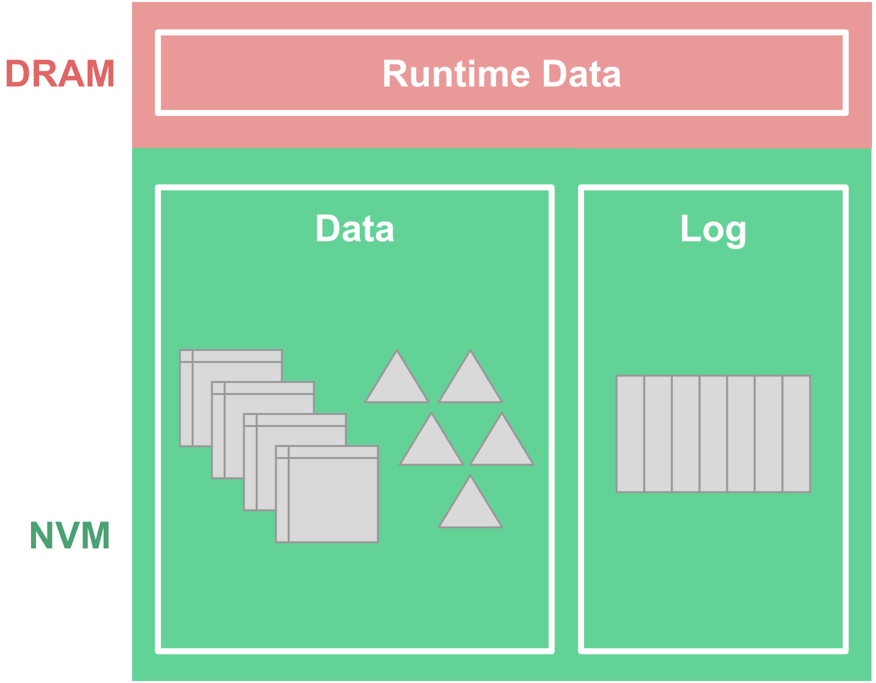
- In main memory DB, indexes reside only in memory
    - Rebuild them during the recovery
- Persisting part of the index in NVM can significantly speed up the recovery process

Instant Recovery

Instant Recovery for Main-Memory Databases. CIDR 2015

FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. SIGMOD 2016

Storing Both Data and Logs in NVM

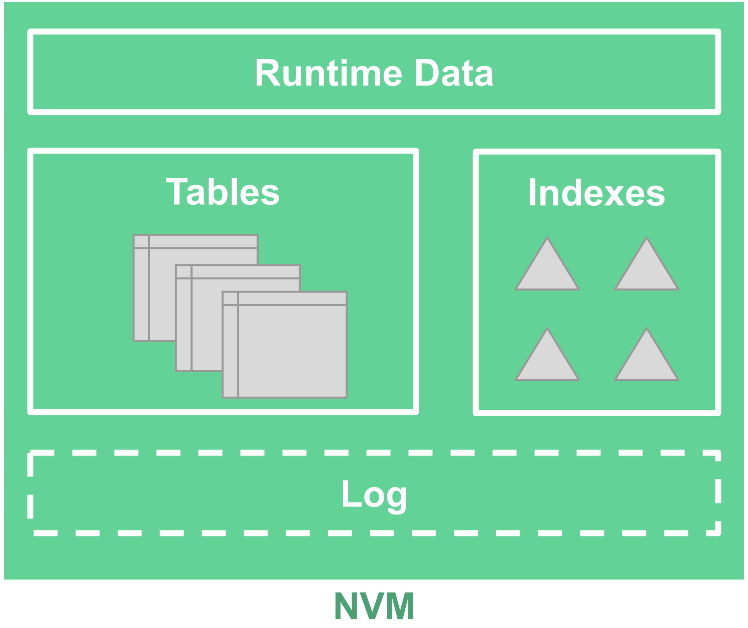


Storage Management in the NVRAM Era. VLDB 2013

Write-Behind Logging. VLDB 2016

- In-place updates to data
  - Cheap random write & byte addressability
  - No need for redo logs

NVM-only Architecture



- May eventually obviate the need for logging
- Problems:
  - In contrast to DRAM, the endurance of NVM is not infinite (10^8 cycles)
  - Some dynamic/runtime data may be more suited for DRAM
    - Locks, latches, temp data

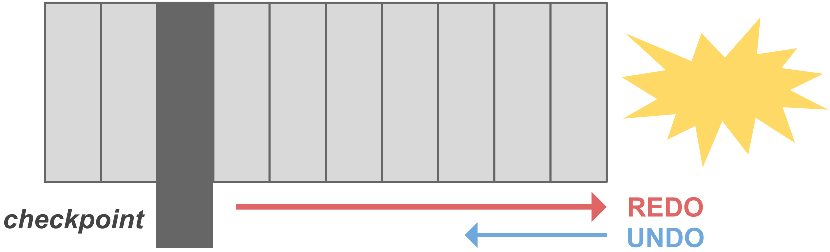
Outline

1. ~~Introduction to NVM~~
2. ~~NVM Programming Model~~
3. ~~NVM-aware Database Architectures~~
4. Database Logging and Recovery
5. NVM-oriented Database Algorithms

ARIES/WAL

- STEAL/NO-FORCE buffer policy
  - STEAL: allow updates made by an <u>uncommitted</u> txn to be written back
    - need for UNDO ⟵ **<u>A</u>**CID
  - NO-FORCE: allow updates made by an <u>committed</u> txn to be written back later
    - need for REDO ⟵ ACI**<u>D</u>**

- Guarantees of Write-Ahead Logging
  - Log records pretaining to a updated page are persisted <u>before</u> the page itself is over-written
  - A txn is not considered to be committed <u>until</u> all of its log records are persisted



Rethink ARIES

- NO-FORCE policy
    - Transform random data writes to sequential log writes
    - But random writes in NVM are cheap
- Overhead of ARIES
    - Centralized logging
        - workarounds: **group commit**, **distributed logging**
    - Data duplication
        - redo log: new version of data, undo log: old version of data
        - **MVCC** can obviate the need for undo log
        - **In-place updates** can obviate the need for redo log
        - **MVCC + in-place updates** → lightweight logging mechanism

Atomic In-place Updates for Non-volatile Main Memories with Kamino-Tx. EuroSys 2017

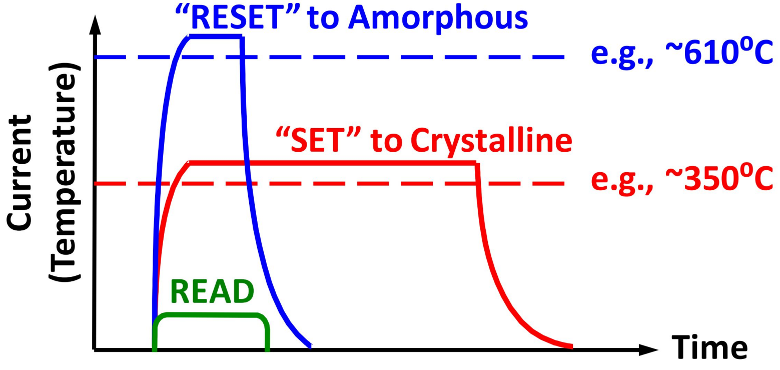
Write-Behind Logging. VLDB 2016

Outline

1. ~~Introduction to NVM~~
2. ~~NVM Programming Model~~
3. ~~NVM-aware Database Architectures~~
4. ~~Database Logging and Recovery~~
5. NVM-oriented Database Algorithms

(Possible) Properties of NVM Writes

- Limited endurance
  - Wear out quickly for hot spots
- High energy consumption
  - SET/RESET heat the material
- High latency & low bandwidth (asymmetric read/write)
  - SET/RESET time > READ time

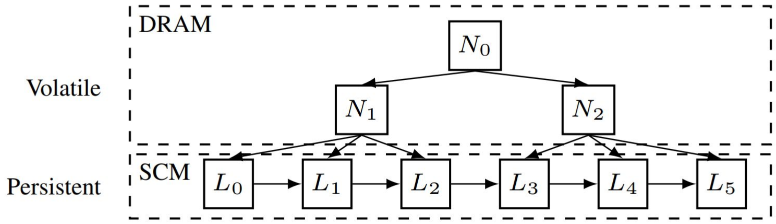


	DRAM	PCM
Read energy	0.8 J/GB	1 J/GB
Write energy	1.2 J/GB	6 J/GB
Idle power	~100 mW/GB	~1 mW/GB
Endurance	∞	$10^6 - 10^8$
Page size	64B	64B
Page read latency	20-50ns	$\sim$ 50ns
Page write latency	20-50ns	$\sim$ 1 μs
Write bandwidth	~GB/s per die	50-100 MB/s per die
Erase latency	N/A	N/A
Density	1×	2 – 4×

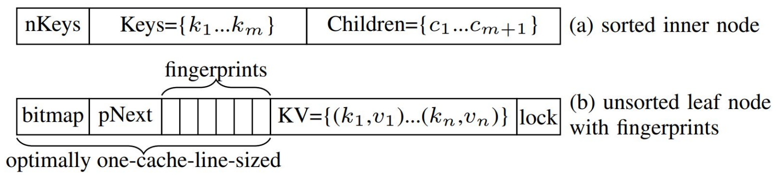
Rethinking Database Algorithms for Phase Change Memory. CIDR 2011

Write-Limited B+ Tree

- Trade expensive writes for cheaper reads
- Ideas:
  - Keeping node entries unsorted
  - Reduce the frequency of tree reorganizations
    - Allowing underflow in nodes
    - Allowing overflow chains



- Hybrid design: DRAM + NVM
  - DRAM: inner nodes, NVM: leaf nodes
- Speed up reads with lightweight hash



Making B+ -Tree Efficient in PCM-Based Main Memory. ISLPED 2014

FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. SIGMOD 2016

Beyond the B+ Tree

- Write-limited query processing
  - sort, join

- Cost model for query optimizer
  - byte-addressability: similar to main memory db
  - take the read/write asymmetry into count

Write-limited sorts and joins for persistent memory. VLDB 2014

Outline

1. ~~Introduction to NVM~~
2. ~~NVM Programming Model~~
3. ~~NVM-aware Database Architectures~~
4. ~~Database Logging and Recovery~~
5. ~~NVM-oriented Database Algorithms~~