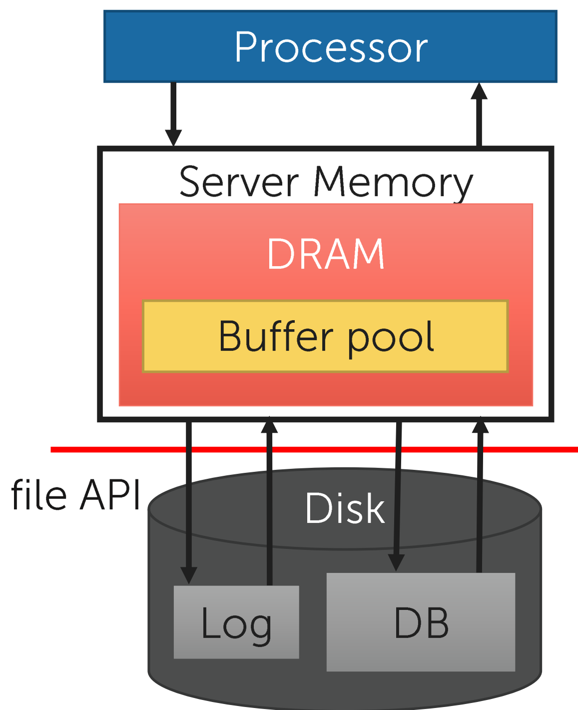
Towards a Single-Level Database Architecture on Byte-Addressable Non-Volatile Memory

Ismail Oukid (TU Dresden & SAP SE)

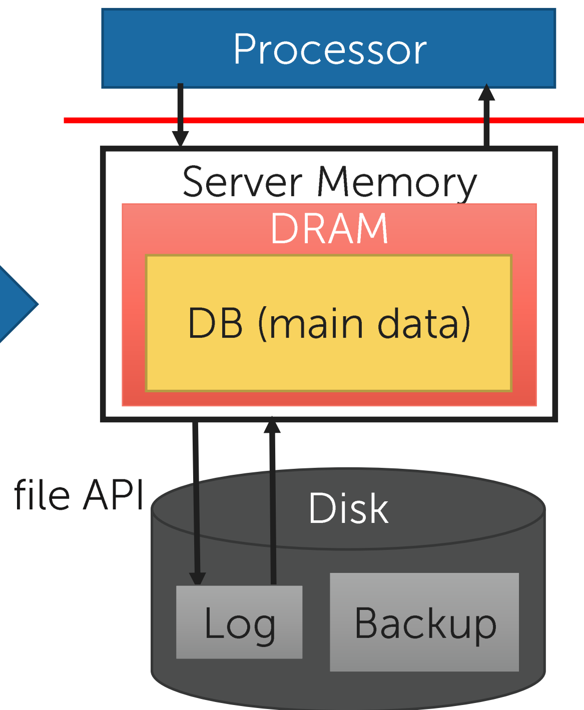
NVRAM Workshop, Paris, May 30th, 2017

From Disk to Main Memory

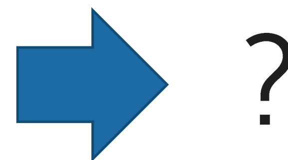
...in ancient times



...10 years back

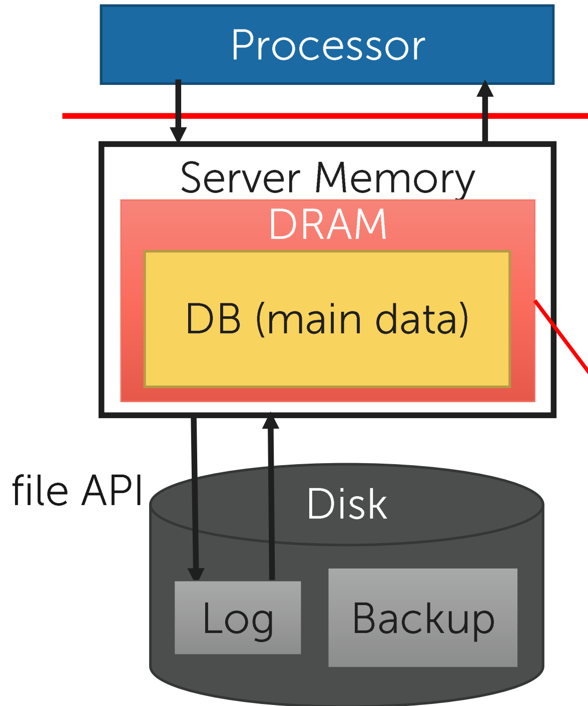


...today?

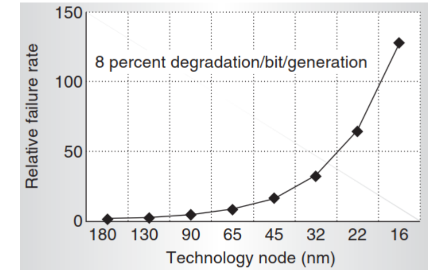


?

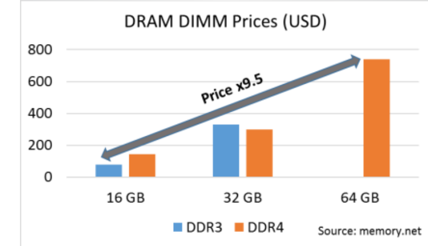
From Disk to Main Memory



Intrinsically hard to further increase DRAM's density

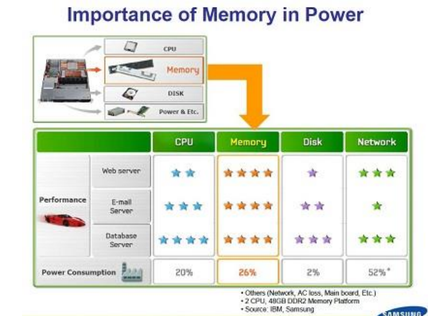


Cost per GB does not scale
→ 9,5x price for 4x capacity



Ever-increasing need for more main memory

Core count increasing faster than DIMM capacity

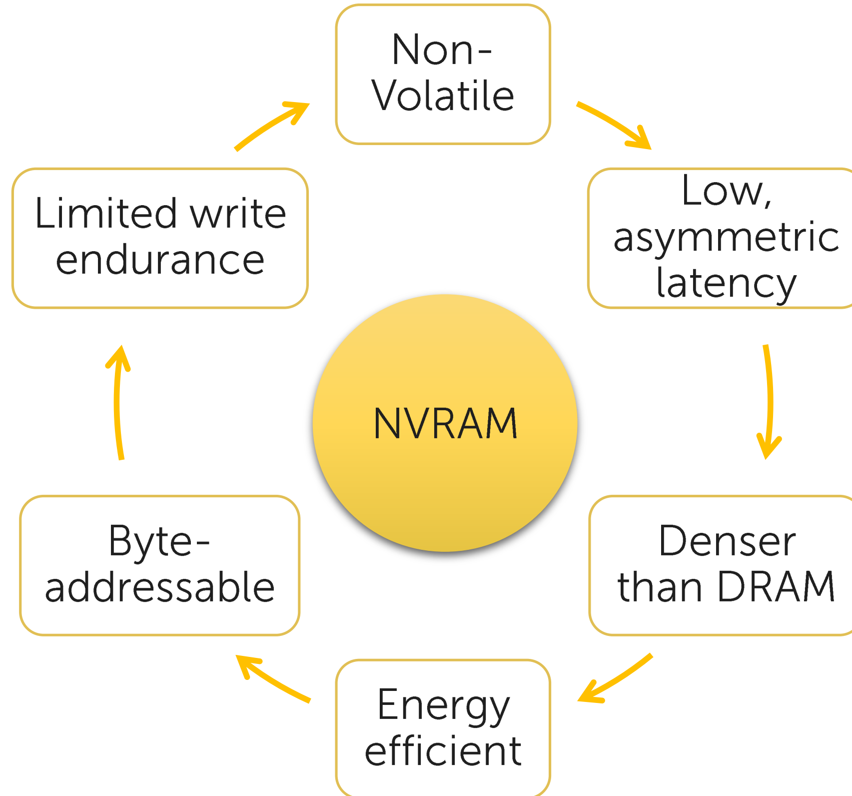


DRAM is hitting its scalability limits

Byte-Addressable Non-Volatile Memory

SAP Dresden Database Systems Group

We assume hardware-based wear-leveling

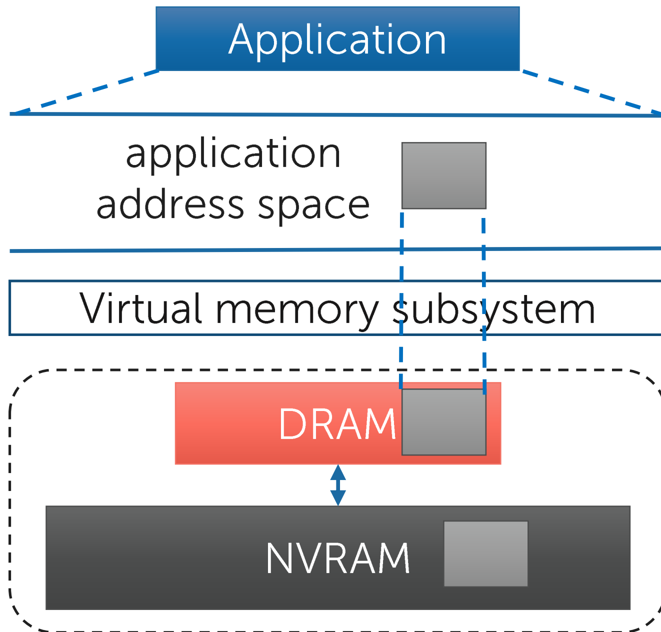


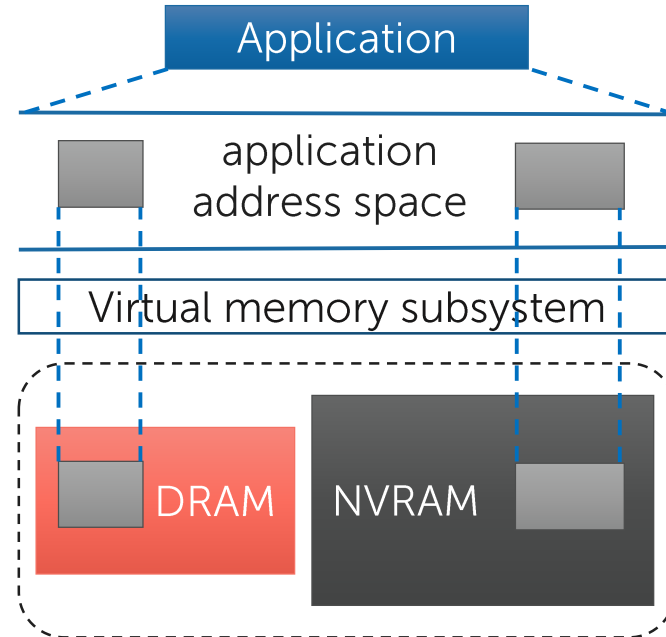
Writes noticeably slower than reads

More capacity and cheaper than DRAM
→ 3 TB per socket for first-gen 3D XPoint

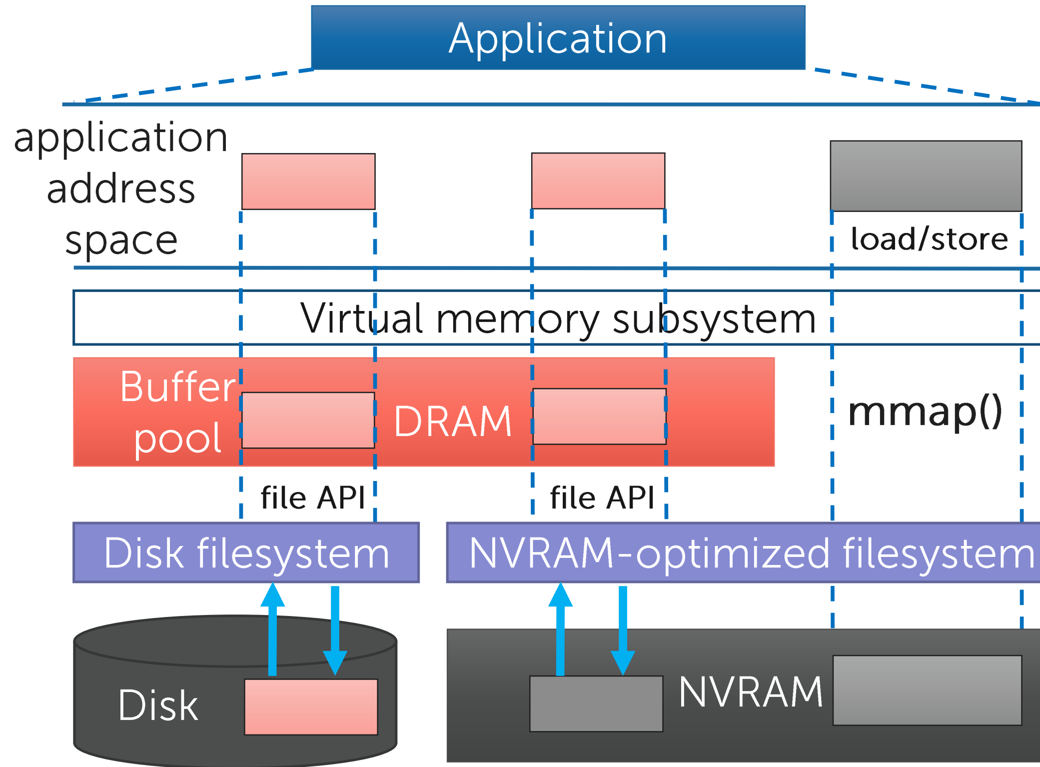
NVRAM as Transient Main Memory

Dresden Database Systems Group





NVRAM as Persistent Main Memory



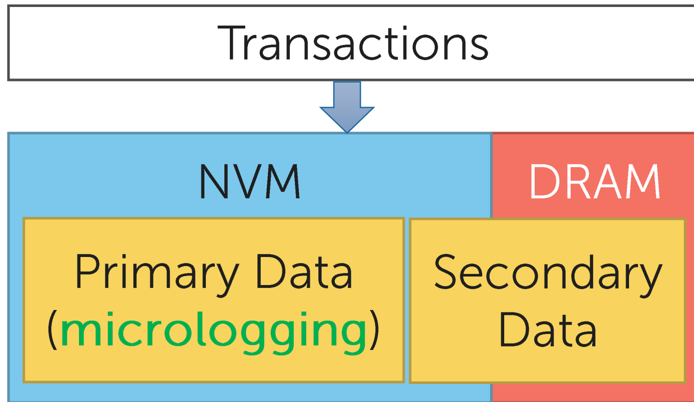
- SNIA recommends to access NVRAM via file mmap
- An NVRAM-aware filesystem provides zero-copy mmap, bypassing the OS page cache

→ Several filesystem proposals: NOVA, PMFS, SCMFS, etc.

→ Linux ext4 and xfs already provide Direct Access support

NVRAM may serve as memory and storage at the same time

Towards a Single-Level Database Architecture*



Hybrid NVM-DRAM architecture

- ➢ Primary data in NVM
- ➢ Secondary data in DRAM or NVM

→ Near-instant recovery
→ No WAL log
→ Low TX latency

*Instant Recovery for Main-Memory Databases. Oukid et al. In CIDR 2015.

NVM can revolutionize database architecture

Is it a *free lunch*?

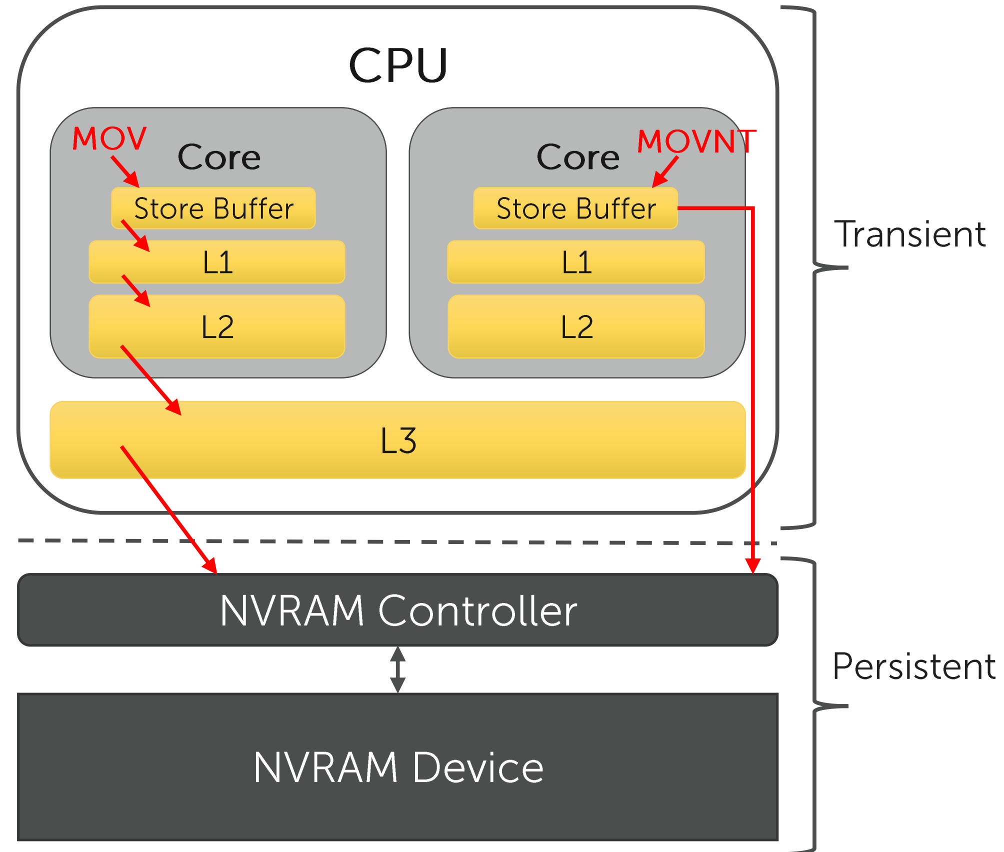
NO

Data Durability

Little control over when data is persisted
- CPU Cache eviction policy
- Memory reordering

Enforce order & durability of stores
- CLFLUSH, CLFLUSHOPT, CLWB
- MFENCE, SFENCE, LFENCE
- Non-temporal stores (MOVNT)

New primitives are being researched
- e.g., HOPS and its OFENCE and DFENCE barriers

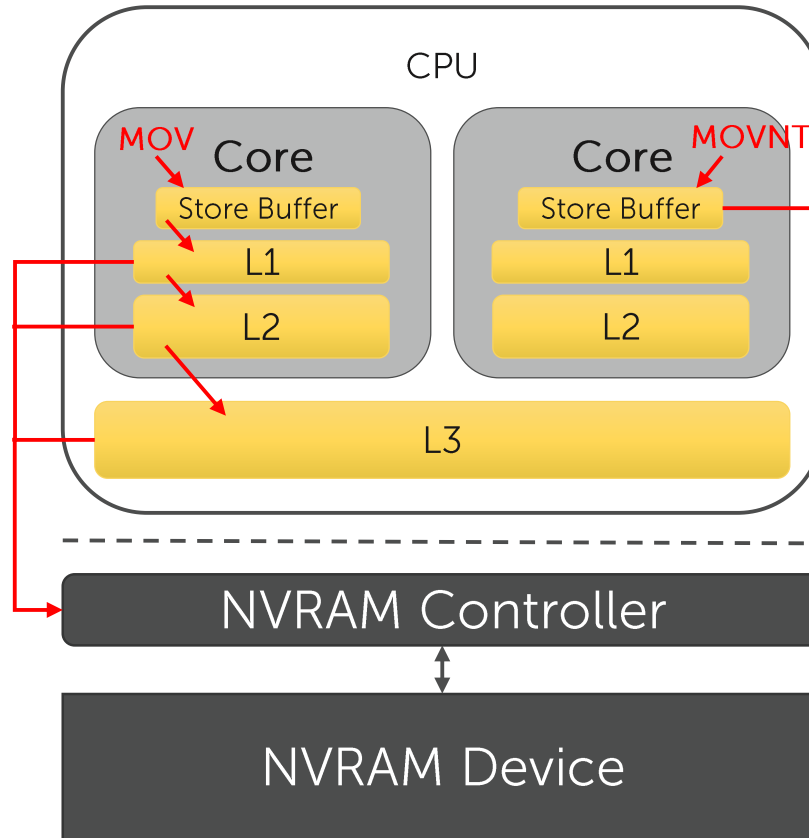


Data Durability

Ensure preceding writes made it to the store buffer → guarantee that the latest data is flushed

SFENCE + CLWB + SFENCE

Ensure CLWB finishes executing



SFENCE

Ensure the NT store buffer is drained to NVRAM

Data Durability: Example

Dresden Database Systems Group

Simplified array append operation

```
void push_back(int val){
    m_array[m_size] = val;
    sfence();
    clwb(&m_array[m_size]);
    sfence();
    m_size++;
    sfence();
    clwb(&m_size);
    sfence();
}
myArray.push_back(2017);
```

What is in NVRAM after the insertion?

m_size	m_array				
0					✗
1					✗ Corrupt!
0	2017				✗
1	2017				✓

Need to enforce write ordering and durability at cache-line granularity

Partial Writes

Persist = sfence + clwb + sfence

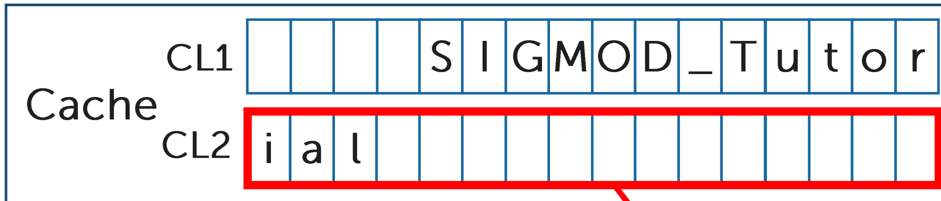
p-atomic store → executes in a one CPU cycle

Currently only 8-Byte stores are p-atomic on Intel x86

```
strcpy(ptr, "SIGMOD Tutorial");
persist(ptr, 15);
flag = true;
persist(&flag);
```

What is in NVRAM?

1. ""
2. "SIGM"
3. "SIGMOD T"
4. "SIGMOD Tutor"
5. "SIGMOD Tutorial"
6. "\0\0\0\0\0\0\0\0\0\0\0\0ial"



CL2 evicted before CL1, e.g., due to a context switch

Need software-built p-atomicity for writes > 8 bytes

Persistent Memory Leaks

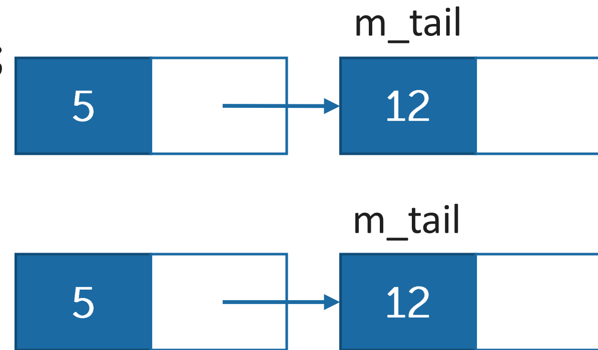
SAP Dresden Database Systems Group

New class of memory leaks resulting from failures

Example: crash during a linked-list insertion

```
void append(int val){
    node *newNode = new node();
    newNode->value = val;
    persist(&(newNode->value));
    m_tail->next = newNode;
    persist(m_tail);
    m_tail = newNode;
    persist(&m_tail);
}
List.append(9);
```

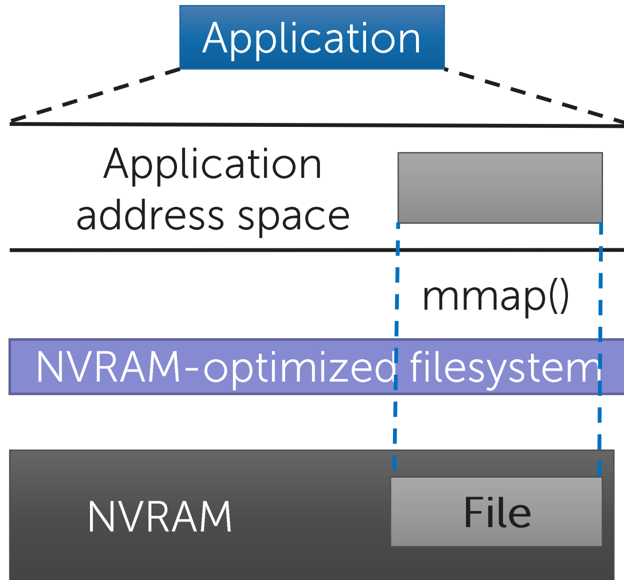
Persistent allocation



Failure-induced persistent memory leak!

Avoiding memory leaks is a requirement

Data Recovery



Address space lost upon restart
→ stored virtual pointers become invalid

Filesystem provides a naming scheme

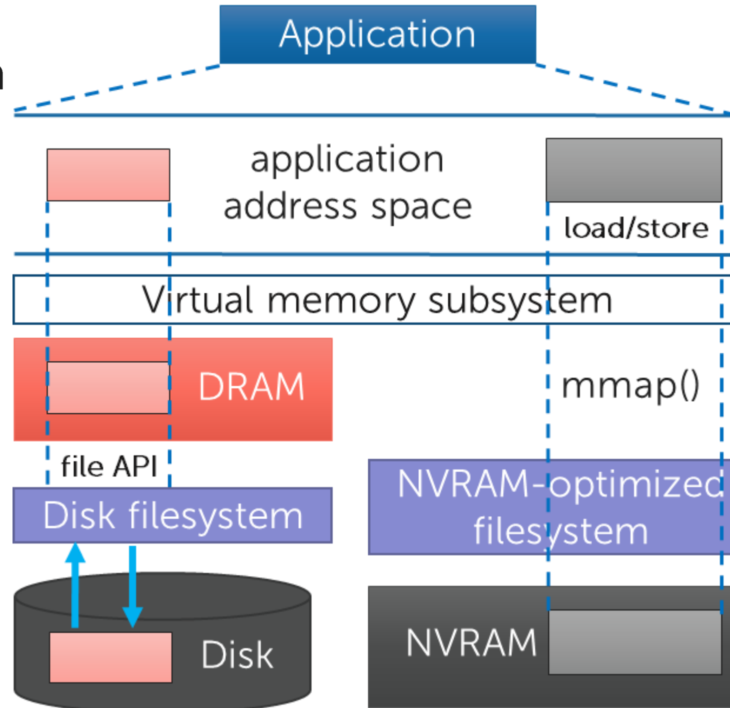
One file per object not realistic
→ How to recover objects?

Need persistent, recoverable NVRAM addressing scheme

Testing of NVRAM-Based Software

Traditional storage media accessed via DRAM → Data corruption risks minimized

Corruption happens first in DRAM → catch the corruption before it propagates to disk



NVRAM directly exposed to the user space → **more corruption risks**

Dangling pointer → Persistent data corruption

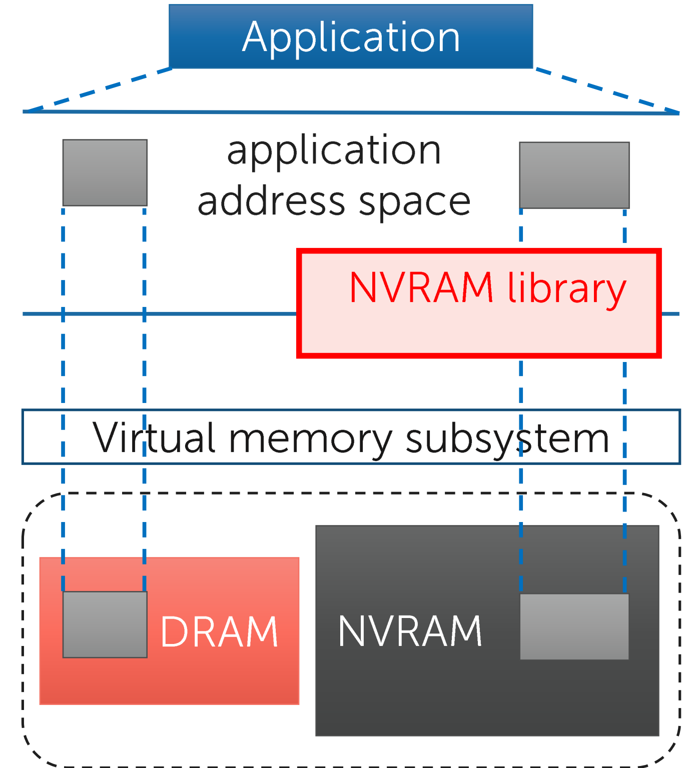
Missing or misplaced persistence primitives; wrong store order, etc.

Need testing and validation tools for NVRAM-based software

NVRAM Programming Models

We look at the following NVRAM programming challenges:

1. How to provide a recoverable addressing scheme?
2. How to avoid persistent memory leaks?
3. How to ensure data consistency?

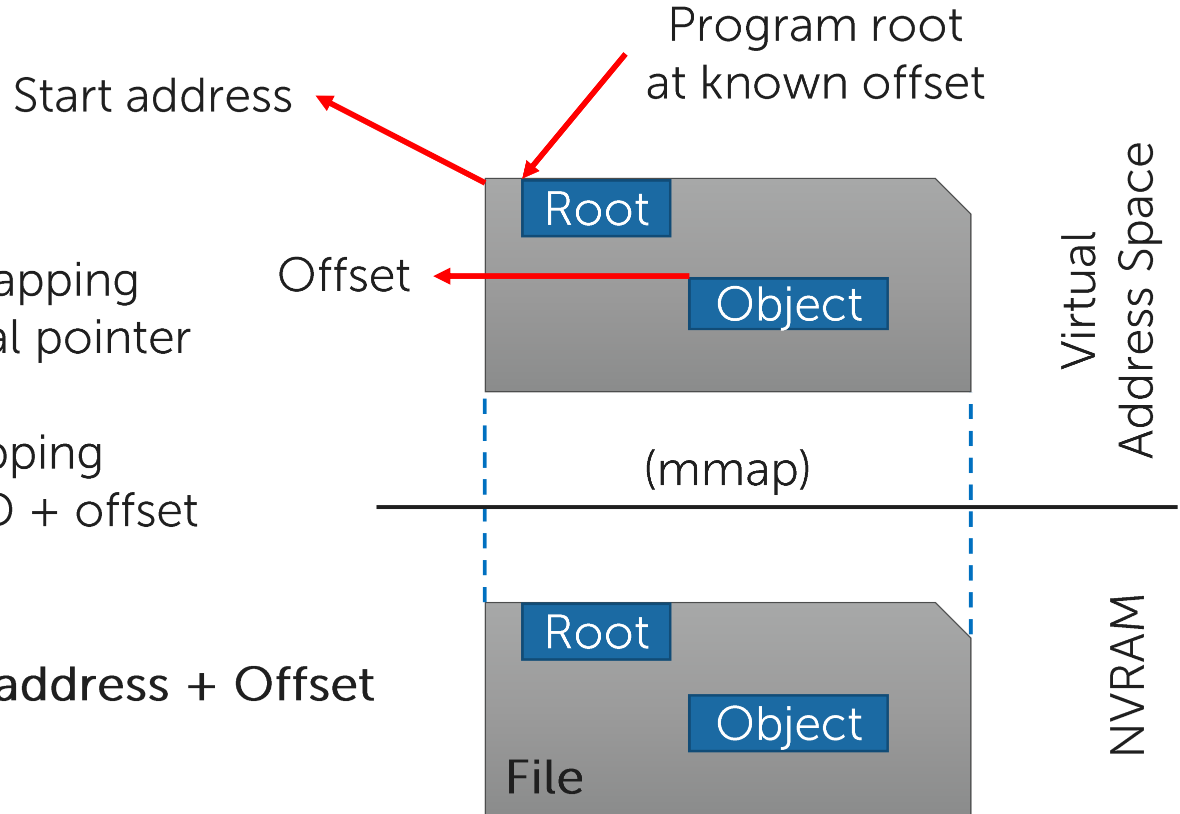


Recoverable Addressing Scheme

Two alternatives

- Fixed-address memory-mapping
  Persistent pointer → virtual pointer
- Unrestricted memory-mapping
  Persistent pointer → file ID + offset

Volatile pointer = File start address + Offset



Recoverable Addressing Scheme

SAP Dresden Database Systems Group

Fixed-address memory-mapping	Unrestricted memory-mapping

Fixed-address memory-mapping

Pros:
- Familiar interface
- No runtime overhead

Cons:
- Fixed address is a security issue
- Can unmap existing mappings

Unrestricted memory-mapping

Pros:
- Safe, easy-to-implement, and portable approach

Cons:
- Potential overhead for converting to regular pointer

Unrestricted memory-mapping the safest way to go

Preventing Memory Leaks

Dresden Database Systems Group

```
pptr = allocate(size);
persist(&pptr);
```

→ Traditional interface has a “blind spot”

Three alternatives

- Reference passing
  → `allocate(PPtr &pptr, size_t allocSize)`
  `pptr` is owned by the data structure

- Transactional logging
  → Wrap operation involving allocation within fail-atomic transaction
  `BEGIN_TX {pptr = allocate(size); persist(&pptr);} END_TX`

- Offline garbage collection
  → Scan allocated blocks upon recovery to detect memory leaks

Preventing Memory Leaks

Reference Passing

Pros:
- Explicit memory management
- No runtime overhead

Cons:
- Data structure must be aware of memory leaks

Transactional Logging

Pros:
- Data structure can be leak-oblivious

Cons:
- Runtime overhead due to write-ahead log

Offline Garbage Collection

Pros:
- Catch existing memory leaks upon restart
- No runtime overhead

Cons:
- Restricts programming language
- Slow recovery

Reference passing closer to becoming the standard

Consistency Handling

SAP Dresden Database Systems Group

Transactional Model

Provide durable transaction semantics for NVRAM programming

```
void push_back(int val){
    TXBEGIN {
        m_array[m_size] = val;
        m_size++;
    } TXEND
}
```

At least 4 writes

Lightweight Primitives

Provide basic functionality, e.g., memory allocation, leak avoidance etc.

```
void push_back(int val){
        m_array[m_size] = val;
        persist(&m_array[m_size]);
        m_size++;
        persist(&m_size);
}
```

Only 2 writes

Consistency Handling

Transactional Model	Lightweight Primitives

Transactional Model

Pros:
- Easy to use and to reason about

Cons:
- Overhead due to systematic logging
- Low-level optimizations not possible

Lightweight Primitives

Pros:
- Low-level optimizations possible

Cons:
- Programmer must reason about the application state
  → Harder to use and error prone

High Performance → Lightweight Primitives

Existing NVRAM Libraries

PPtr → Persistent Pointer

Approach	Consistency Handling	Addressing Scheme	Leak Prevention	Compiler support	Source
Mnemosyne	Transactional & Lightweight primitives	**PPtr**: file offset **Recovery**: new mmap in reserved address space	Reference passing | Transactional logging	Yes	ASPLOS'11
NV-Heaps	Transactional	**PPtr**: file Id + offset **Recovery**: new mmap	Transactional logging	No	ASPLOS'11
Intel NVML	Transactional & Lightweight primitives	**PPtr**: file Id + offset **Recovery**: new mmap	Reference passing | Transactional logging	No	http://pmem.io/
Atlas	Transactional (sections determined by locks)	**PPtr**: volatile pointer **Recovery**: fixed mmap	Transactional logging	Yes	OOPSLA'14
REWIND	Transactional	Undefined, hints → **PPtr**: volatile pointer **Recovery**: fixed mmap	Transactional logging	Yes	VLDB'15
PAllocator	Lightweight primitives	**PPtr**: file Id + offset **Recovery**: new mmap	Reference passing	No	To appear

Recommended starting point: NVML → rich, open source, actively developed

Fundamental Building Blocks

Persistent Memory Management*	Persistent Data Structures+	Testing tools for NVM-Based Software°

* Memory management techniques for large-scale persistent-memory-based system. VLDB 2017.

+ FPTree: A hybrid SCM-DRAM persistent and concurrent B-Tree for Storage Class Memory. SIGMOD 2016.

° On testing persistent-memory-based software. DaMoN 2016 (co-located with SIGMOD 2016).

Fundamental Building Blocks

Persistent Memory Management*	Persistent Data Structures+	Testing tools for NVM-Based Software°

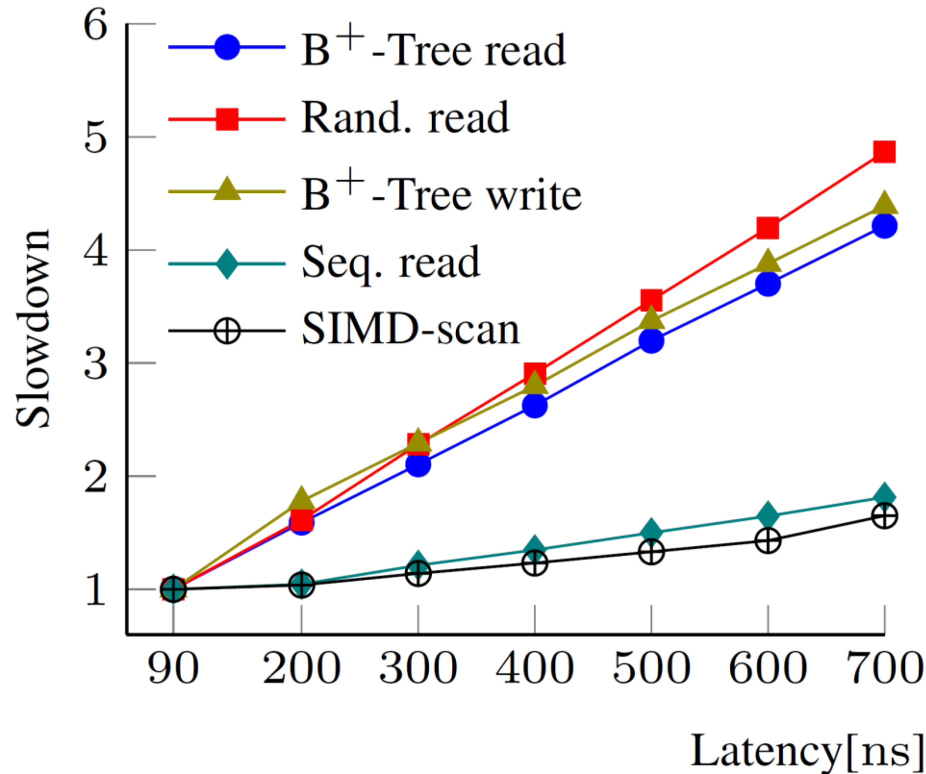
* Memory management techniques for large-scale persistent-memory-based system. VLDB 2017.

+ FPTree: A hybrid SCM-DRAM persistent and concurrent B-Tree for Storage Class Memory. SIGMOD 2016.

° On testing persistent-memory-based software. DaMoN 2016 (co-located with SIGMOD 2016).

NVRAM Performance Implications

Dresden Database Systems Group



Random memory accesses suffer from higher latencies

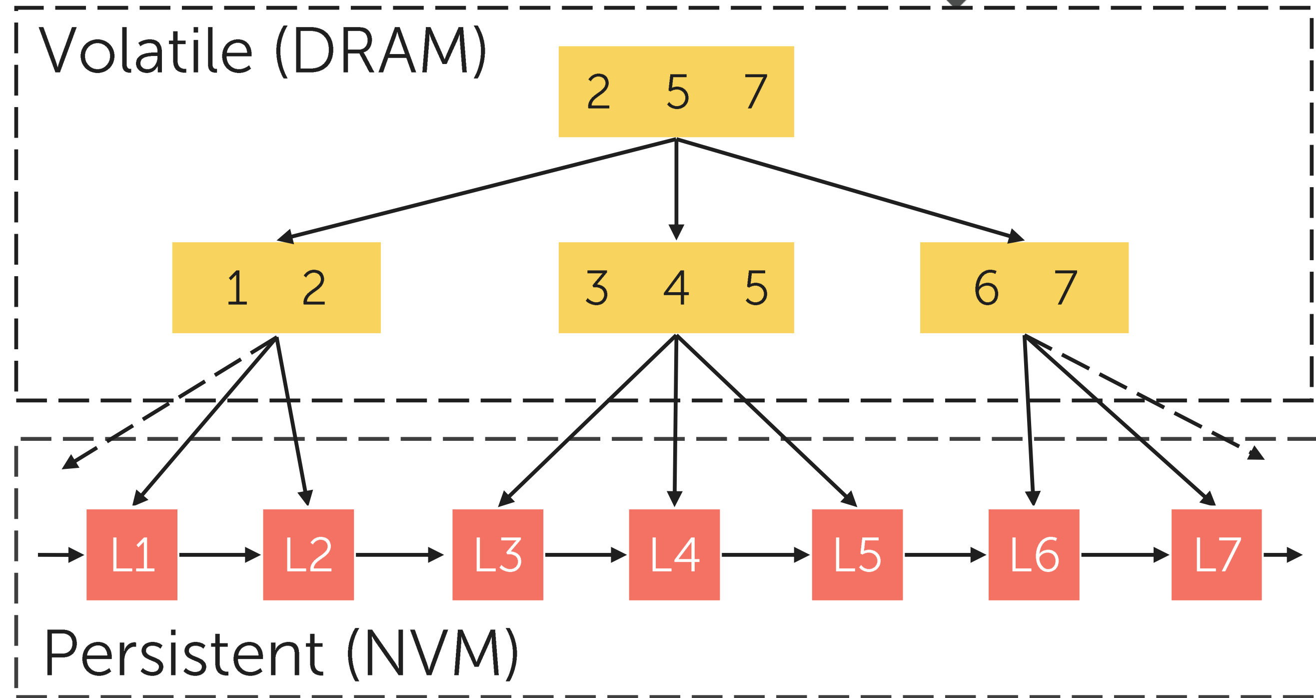
Sequential memory accesses are resilient to higher latencies

Hybrid NVM-DRAM Data Structures

SAP Dresden Database Systems Group

Inner nodes in DRAM for better performance

Leaves in NVM to ensure durability



Recovery is up to 100x faster than a full rebuild
Near-DRAM performance with only 3% of data in DRAM

Dresden Database Systems Group

Unsorted Leaves

Sorted leaf

Counter

1.

3	4	7	14	
	a	b	c	

2.

3	4		7	14
	a		b	c

3.

3	4	5	7	14
	a	d	b	c

4.

4	4	5	7	14
	a	d	b	c

! Potential corruption

! Writes slower than reads

Unsorted leaf

Bitmap

1.

7	14	12	10
e	f	g	h

2.

5	14	12	10
d	f	g	h

p-atomic

3.

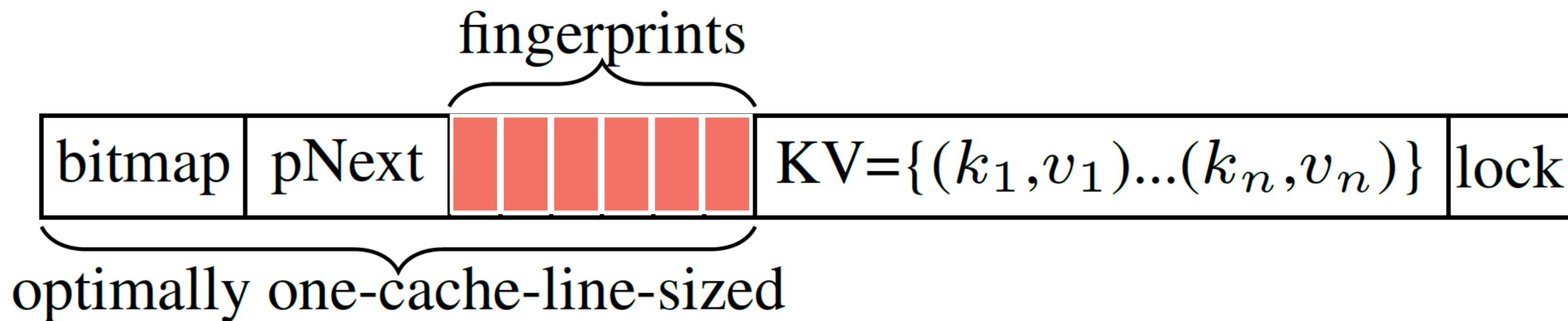
5	14	12	10
d	f	g	h

p-atomicity + decreased number of writes

TECHNISCHE UNIVERSITÄT DRESDEN

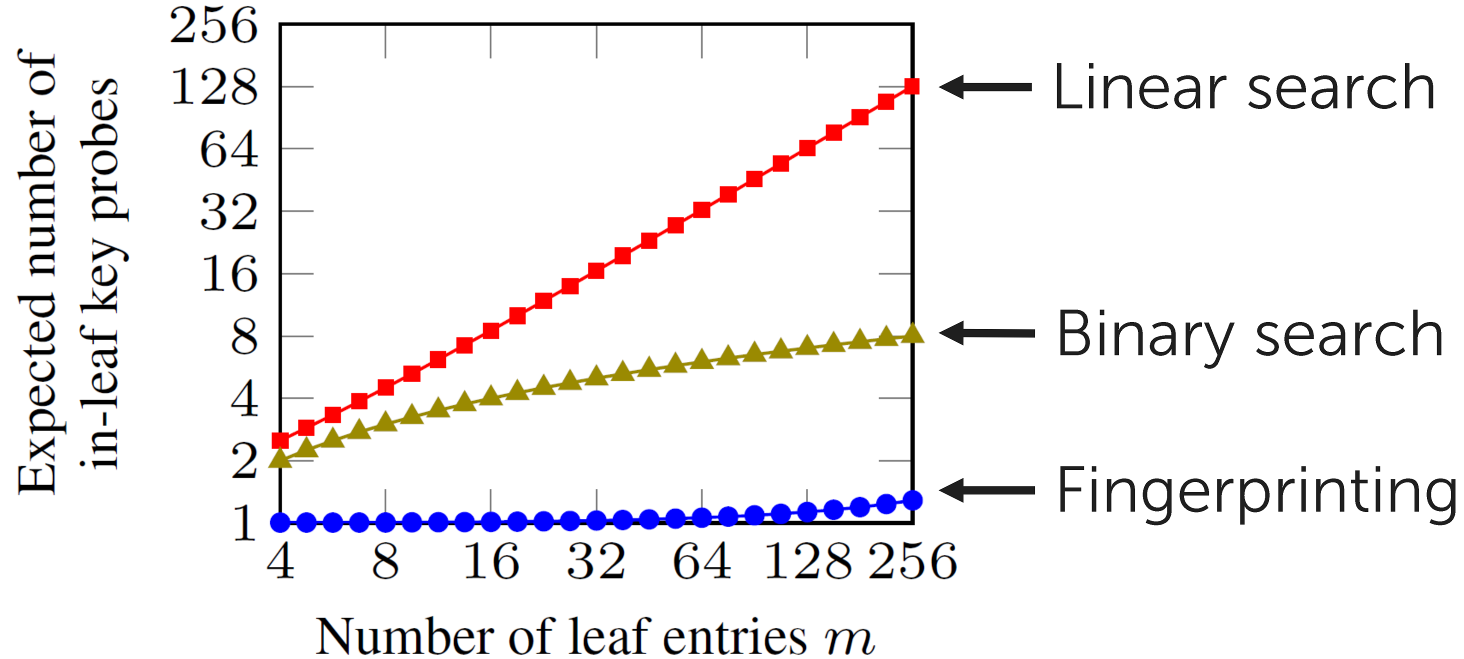
Fingerprinting

A fingerprint is a 1-byte hash of a key



Fingerprints act as a filter

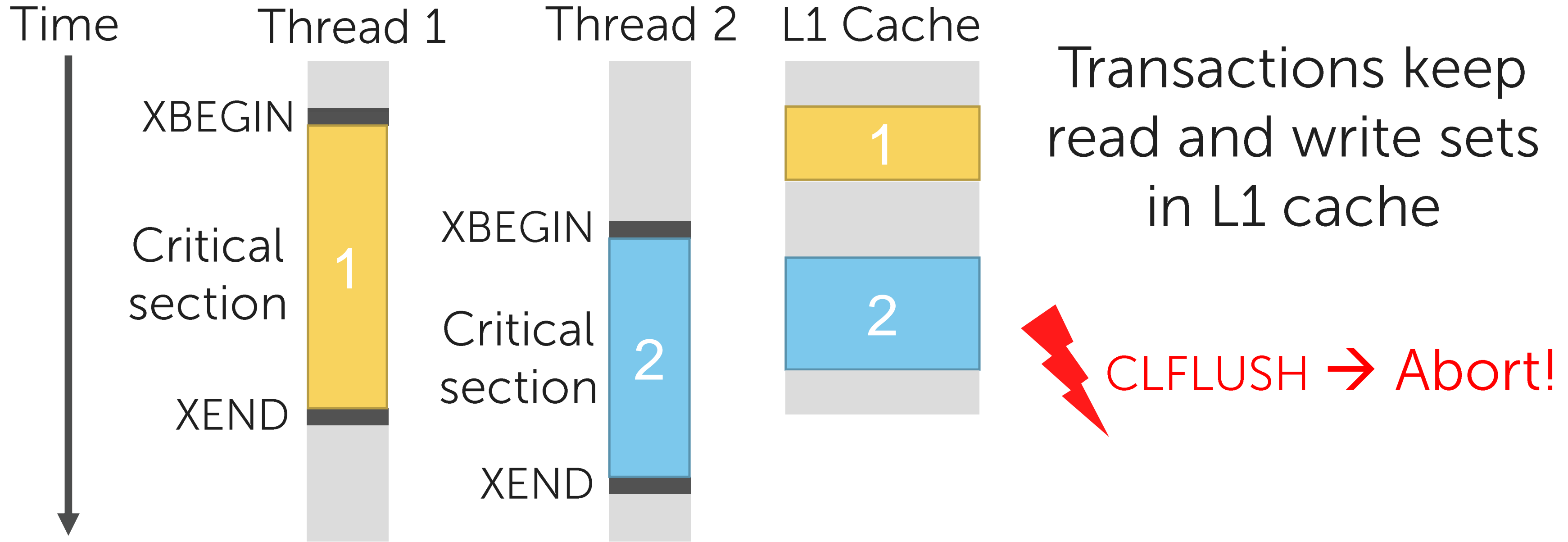
Fingerprinting



One expected key probe for up to 512 entries

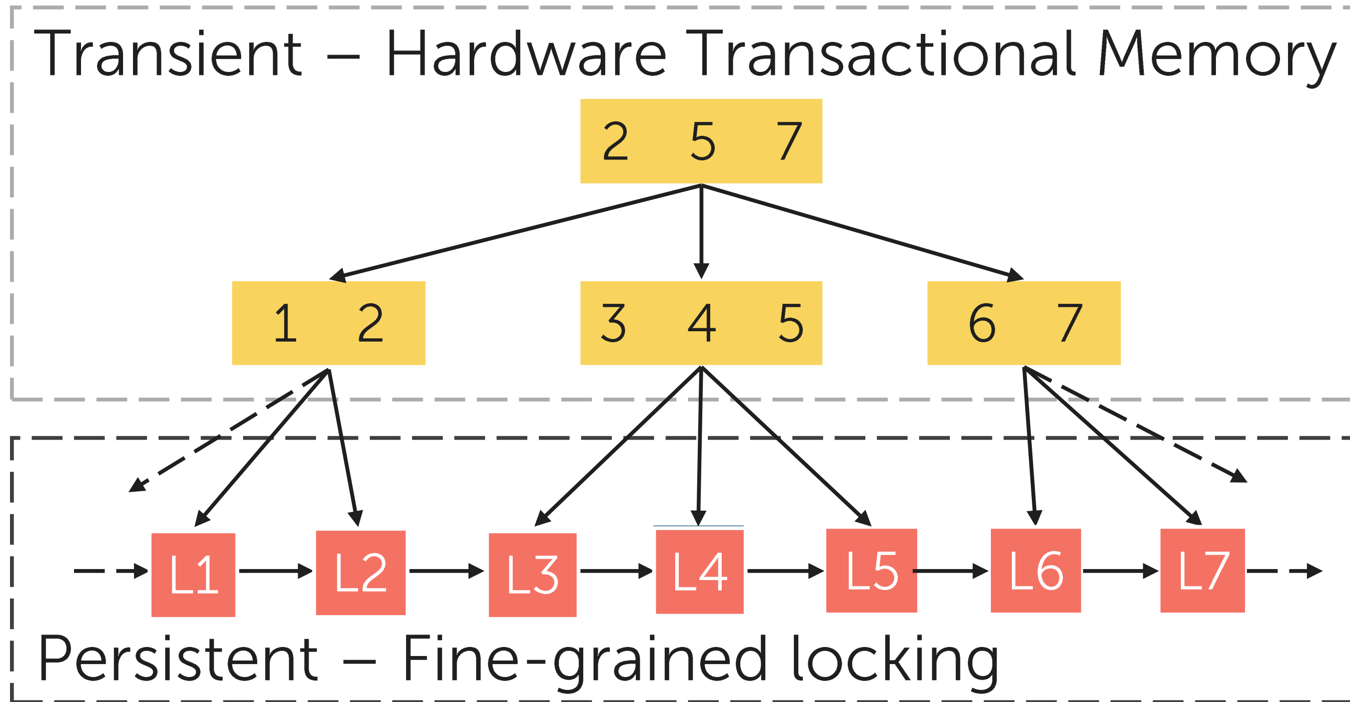
Hardware Transactional Memory

Allows optimistic execution of critical sections

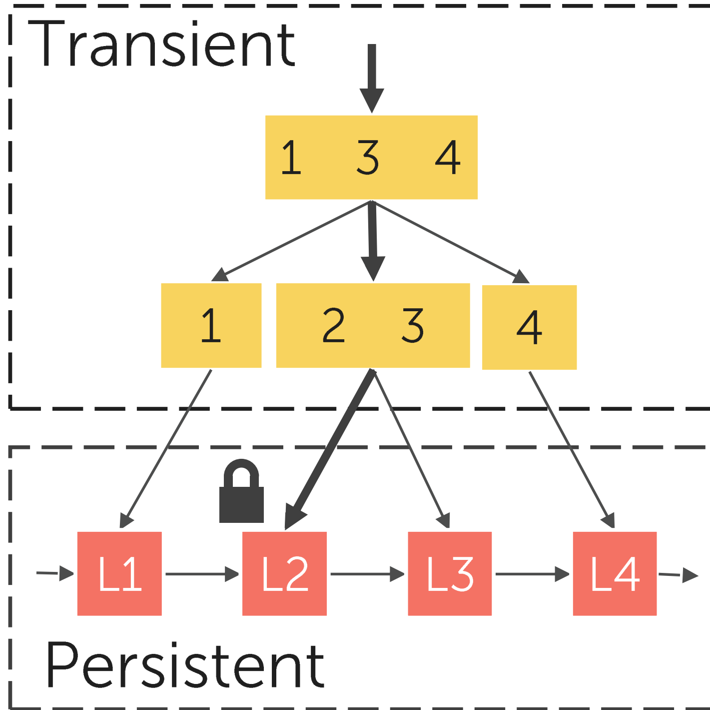


There is an incompatibility between HTM and NVM

Selective Concurrency



Selective Concurrency: Insertion



1. Find and lock leaf	2. Modify leaf	3. Update parents	4. Unlock leaf

XBEGIN CLFLUSH XEND

XEND XBEGIN

Reconcile HTM and NVM

Fundamental Building Blocks

Persistent Memory Management*

Persistent Data Structures+

Testing tools for NVM-Based Software°

* Memory management techniques for large-scale persistent-memory-based system. VLDB 2017.

+ FPTree: A hybrid SCM-DRAM persistent and concurrent B-Tree for Storage Class Memory. SIGMOD 2016.

° On testing persistent-memory-based software. DaMoN 2016 (co-located with SIGMOD 2016).

Bug Example

Simplified vector append operation:

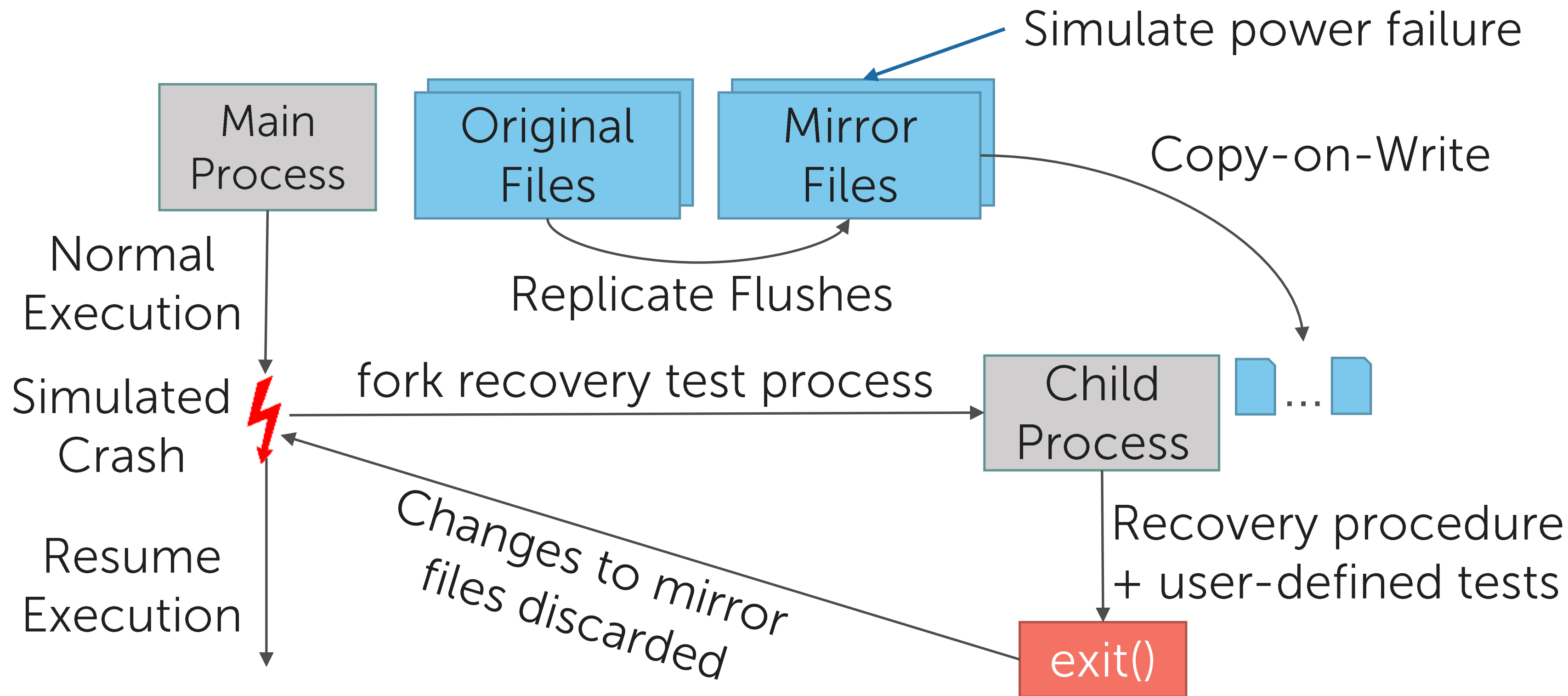
```
1 array[size] = val;
2 persist(&array[size]);
3 size++;
4 persist(&size);
```

Correct code

```
1 array[size] = val;
2 size++;
3 persist(&size);
```

Missing persist

Suspend-Test-Resume

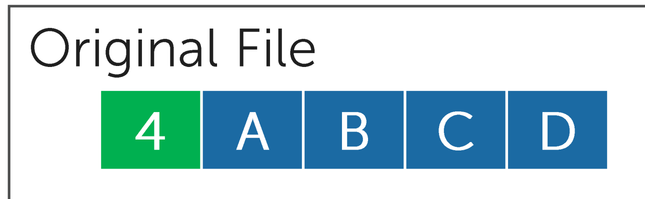


+ Fast and automated – Not exhaustive

Bug Example Revisited


```
1 array[size] = val;
2 size++;
3 persist(&size);
```

Missing persist



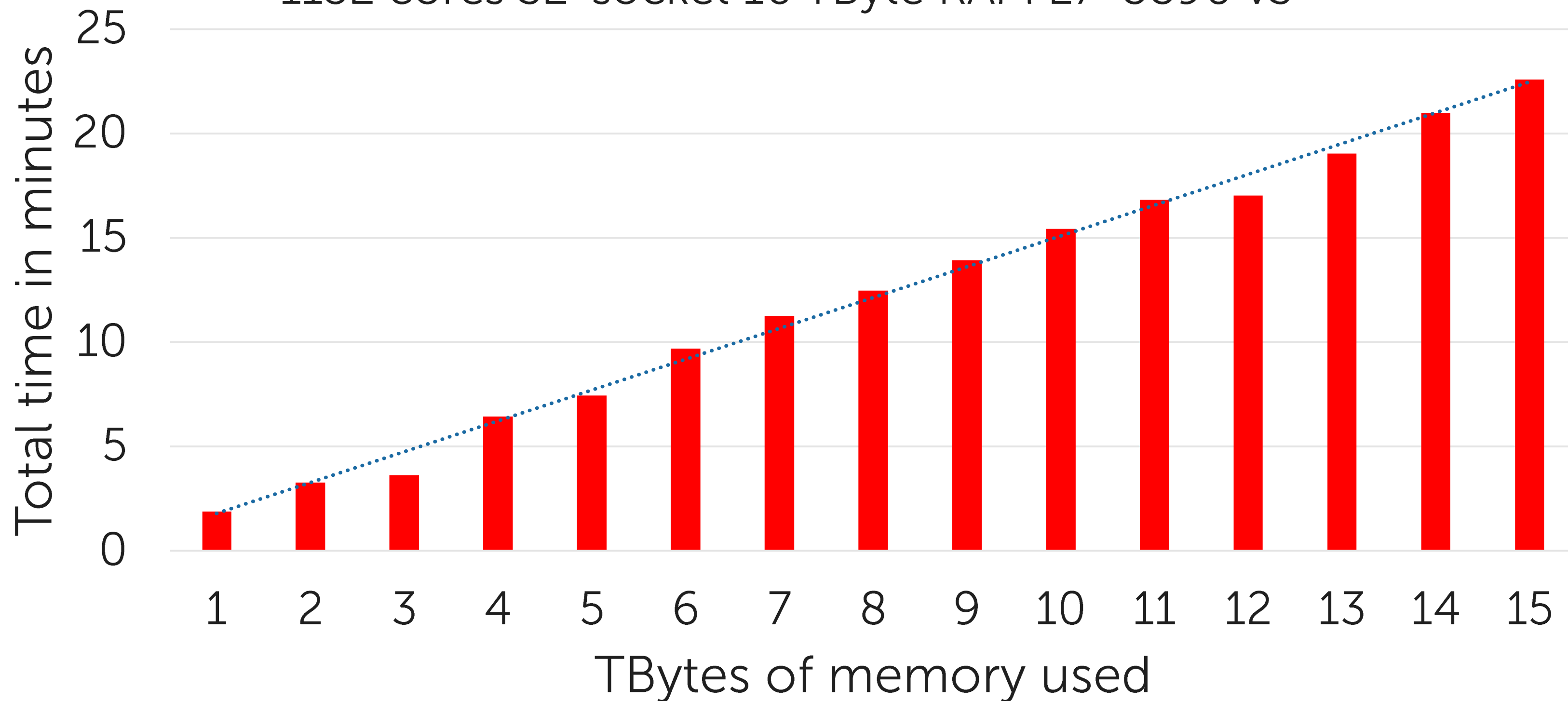
Efficiently catch missing persistence primitives

Some more challenges…

Operating System Limitations

- **Filesystems not designed to handle millions of files**
  - NVM programming model relies on file creation and mmapping
  - Single pool not an option (e.g., due to fragmentation)
- **Page Table (lack of) scalability**
  - Memory mapping millions of files upon startup a challenge
  - Slow memory reclamation upon process termination

Process Termination Duration



Courtesy of Robert Kettler and Daniel Booss

Conclusion

SAP
Dresden Database Systems Group

- NVM brings great opportunities for databases
  - Near-instant recovery
  - Remove the need to traditional write-ahead logging
  - Better transaction throughput and latency
  - Larger, cheaper, more energy-efficient machines

Hybrid SCM-DRAM data structures is a promising approach

- NVM brings great challenges as well
  - Novel programming model → novel programming challenges
  - Operating systems not ready for NVM
  - Aggravated corruption risks, novel failure scenarios
  - Exhaustive testing not feasible: theoretical guarantees a prerequisite

References

Our work

- Memory Management Techniques for Large-Scale Persistent-Main-Memory Systems. Oukid et al. In VLDB 2017.
- Data Structure Engineeting for Byte-Addressable Non-Volatile Memory. Oukid I. & Lehner W. In SIGMOD 2017 (Tutorial)
  - Slides → http://sigmod2017.org/sigmod-program/#stutorial6
- Storage Class Memory and Databases: Opportunities and Challenges. Oukid et al. In Informatoin Technology – it 2017.
- FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. Oukid et al. In SIGMOD 2016.
- On Testing Persistent-Memory-Based Software. Oukid et al. In DaMoN 2016.
- Instant Recovery for Main Memory Databases. Oukid et al. In CIDR 2015.
- SOFORT: A Hybrid SCM-DRAM Storage Engine for Fast Data Recovery. Oukid et al. In DaMoN 2014.

Further Readings

Dresden Database Systems Group

Resources

- SNIA NVM Programming Model V1.1. Technical report, 2015. http://www.snia.org/sites/default/files/NVMProgrammingModel_v1.1.pdf.
- Intel Architecture Instruction Set Extensions Programming Reference. http://software.intel.com/en-us/intel-isa-extensions.

Architecting SCM

- Scalable high performance main memory system using phase-change memory technology. Qureshi et al. In ISCA 2009.
- Architecting phase change memory as a scalable dram alternative. Lee et al. In SIGARCH Comput. Archit. News, 37(3), 2009.
- Systems and Applications for Persistent Memory. SR. Dulloor. PhD Thesis, 2016. https://smartech.gatech.edu/bitstream/handle/1853/54396/DULLOOR-DISSERTATION-2015.pdf.
- System software for persistent memory. Dulloor et al. In EuroSys 2014.

Persistent Memory Management

- Mnemosyne: Lightweight persistent memory. Volos et al. In ACM SIGPLAN Not., 47(4), 2011.
- Nv-heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. Coburn et al. In ACM SIGPLAN Not., 47(4), 2011.
- Rewind: Recovery write-ahead system for in-memory non-volatile data-structures. Chatzistergiou et al. In VLDB 2015.
- NVML: A collection of open-source libraries by Intel to manage SCM. http://pmem.io/nvml/
- Makalu: Fast Recoverable Allocation of Non-volatile Memory. Bhandari et al. In OOPSLA'16.
- Consistent, durable, and safe memory management for byte-addressable non volatile main memory. Moraru et al. In TRIOS'13.
- WAlloc: An Efficient Wear-Aware Allocator for Non-Volatile Main Memory. Yu et al. In IEEE IPCCC'15.
- nvm malloc: Memory Allocation for NVRAM. Schwalb et al. In ADMS@VLDB'15.

Further Readings

Persistent Data Structures

- Rethinking database algorithms for phase change memory. Chen et al. In CIDR 2011.
- Consistent and durable data structures for non-volatile byte-addressable memory. Venkataraman et al. In USENIX FAST, 2011.
- NV-Tree: A consistent and workload-adaptive tree structure for non-volatile memory. Yang et al. In IEEE Transactions on Computers, 2015.
- Persistent b+-trees in non-volatile main memory. Chen et al. In VLDB 2015.

Testing of Persistent Software

- An open-source extension of Valgrind for byte-addressable non-volatile memory. https://github.com/pmem/nvml
- Yat: A validation framework for persistent memory software. Lantz et al. In USENIX ATC, 2014.

NVM and Databases

- High Performance Database Logging using Storage Class Memory. Fang et al. In ICDE 2011
- Storage Management in the NVRAM-Era. Pelley et al. In VLDB 2014
- Scalable Logging through Emerging Non-Volatile Memory. Wang et al. In VLDB 2014
- NVRAM-Aware Logging in Transaction Systems. Huang et al. In VLDB 2015
- Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. Arulraj et al. In SIGMOD 2015
- FOEDUS: OLTP engine for a thousand cores and NVRAM. H. Kimura. In SIGMOD 2015
- REWIND: Recovery write-ahead system for in-memory non-volatile data-structures. Chatzistergiou et al. In VLDB 2015