Mnemosyne: Lightweight Persistent Memory

Haris Volos, Andres Jaan Tack, and Michael M. Swift

THE UNIVERSITY of WISCONSIN MADISON

Motivation

Technology trends: Storage-Class Memory (SCM) blurs the distinction between
- Memory (fast, expensive, volatile), and
- Storage (slow, cheap, non-volatile)

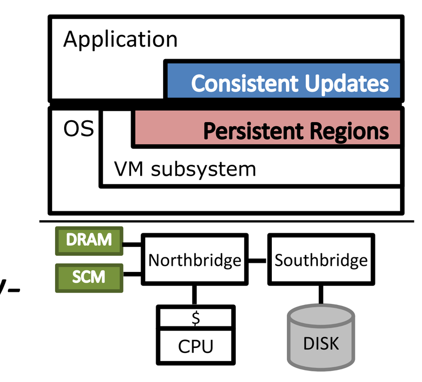
Application trends: Storage-latency drives modern applications and services
- Web applications: Facebook, Amazon,
- Desktop applications: Firefox
- Distributed systems: Paxos
- Other: High-frequency trading

Mnemosyne†

Goal: Enable flexible low-latency storage.

Challenges
- Expose storage-class memory to programmers for direct use.
- Support consistent modifications in the presence of failures.
- Design a system compatible with existing commodity processors.

Mnemosyne provides a new abstraction, *persistent memory*, that programmers can use to get direct access to durable storage.



Compared to past approaches to persistence [1,2,3], Mnemosyne enables *fine-grain* and *low-latency* updates to durable storage.

† Mnemosyne is the personification of memory in Greek mythology, and is pronounced *nee-moss-see-nee*.

Storage-Class Memory (SCM)

Features
- Byte-addressable
- Non-volatile
- Short access time (DRAM-like)

Technologies
- Phase Change Memory
- Spin Torque Transfer RAM
- Memristors

Performance Comparison

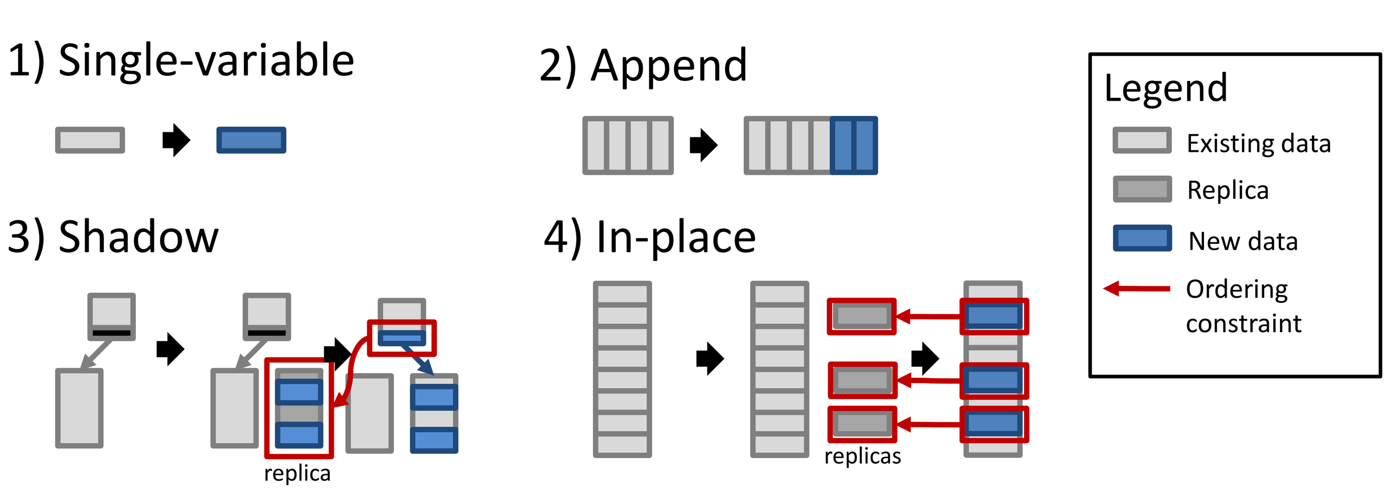
Technology	Read	Write	Endurance
DRAM	60 ns	60 ns	> 10^{16}
Spin Torque Transfer RAM	6 ns	13 ns	10^{15}
Phase Change Memory	100-300 ns	150 – 600 ns	10^{8} – 10^{12}
NAND Flash	85 µs	200 – 500 µs	10^{4} – 10^{5}

Programming Model

Persistent Memory Regions are segments of virtual memory stored in storage-class memory and backed by a disk file. Regions can be created statically to hold variables labeled with the keyword `persistent`, or allocated dynamically. The system also provides a persistent heap for allocating small blocks of memory.

Region Type	API	Conceptually similar to
Static	`persistent var`	UNIX executable data segment
Dynamic	pmap	Anonymous memory
Heap	pmalloc	Volatile heap

Consistent Updates support modifying persistent data without risking correctness after a failure. The system provides four common mechanisms for consistently updating data with varying flexibility.

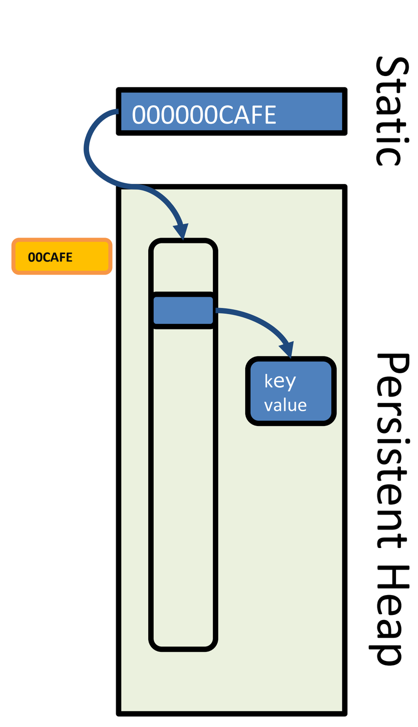


Update	API	Usage example
Single-variable	HW primitive	Set flag
Append	`log_append`	Append to journal
Shadow	`pmalloc, mov`	Update tree node
In-place	`atomic { }`	Update double linked list

Durable Memory Transactions support in-place updates. A programmer annotates a block of code `atomic` and the compiler produces code that passes all memory references to a transaction system. The system ensures all modifications are *atomic* and *durable* but provides no concurrency control.

Example: Persistent Hashtable

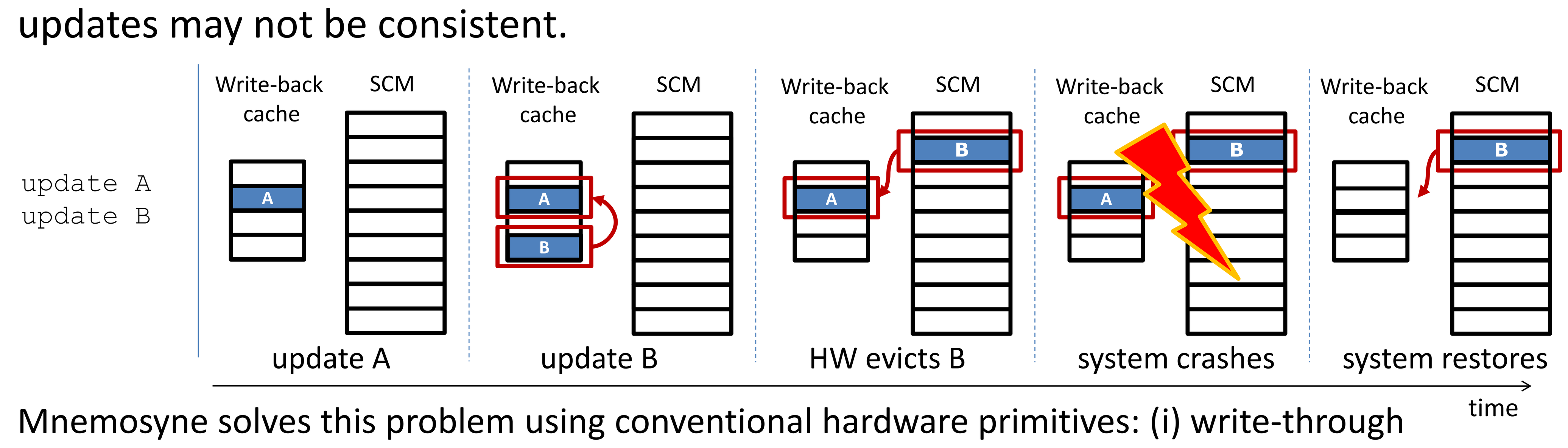
```
persistent hash = NULL;

main() {
  if (!hash) {
    pmalloc(N*sizeof(*bucket), &hash);
  }
}

update_hash(key, value) {
  lock(mutex);
  atomic {
    pmalloc(sizeof(*bucket), &bucket);
    bucket->key = key;
    bucket->value = value;
    insert(hash, bucket);
  }
  lock(mutex);
}
```



Design Challenges

Mapping Consistency onto Conventional Hardware

SCM is attached to the memory bus and is subject to caching. Caching allows reading data in SCM at almost the same latency as from DRAM. However caching may reorder updates to SCM because it may evict cached data at any time in any order. Thus, updates may not be consistent.

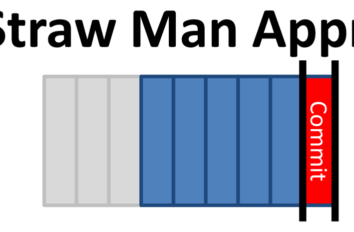


Mnemosyne solves this problem using conventional hardware primitives: (i) write-through stores, which write data directly to memory rather than to the cache, and (ii) fences, which prevent subsequent writes from completing before preceding writes.

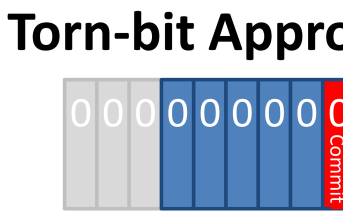
High Performance Logging

Mnemosyne provides a log to support append-only updates. To achieve high performance, the log allows later writes to complete while earlier ones did not. After a system crash, the recovery manager has to identify whether all writes that comprise an append operation completed.

Straw Man Approach: Use 2 fences



Torn-bit Approach: Use 1 fence and versioning



Per-word single-bit versioning is adequate to detect missing writes.

Performance Evaluation

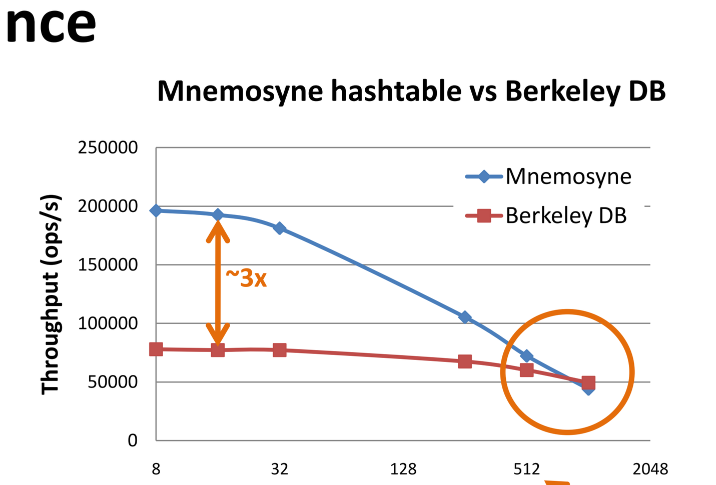
Platforms
- Mnemosyne: persistence primitives/regions + DRAM + fixed delay (150ns) after each store
- PCM-disk: ext2 file system + RAM-disk + a fixed delay when writing a block

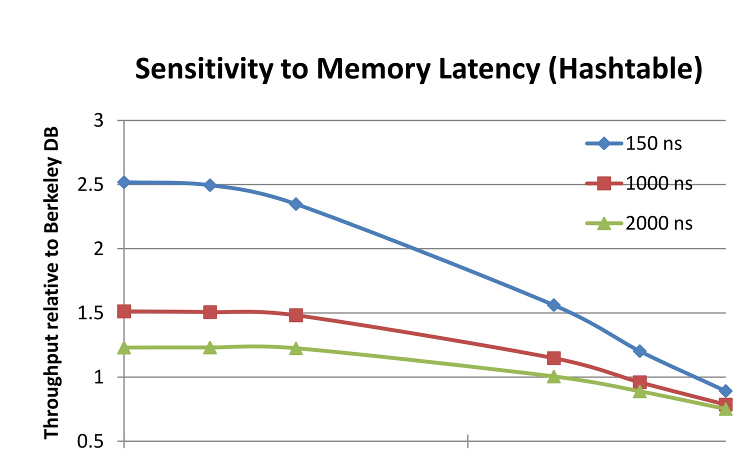
Application Performance

Application	Workload	Backend		Updates/s
TokyoCabinet	512-byte ins/del queries	PCM-disk	Syncs B-tree to a mmap-ed file	8,361
		Mnemosyne	Makes B-tree persistent	23,727
OpenLDAP	SLAMD LDIF template	PCM-disk	Stores entries in Berkeley DB	4,545
		Mnemosyne	Makes front-end cache persistent	5,560

Hashtable Performance

Compare performance of a simple hash table implemented using Mnemosyne against Berkeley DB's hash table.





References

[1] C. Lamb, G. Landis, J. Orenstein and D. WeinReb. The ObjectStore database system. Commun. ACM 34, 10 (1991)

[2] B. Liskov, A. Adya, M. Castro, M. Day, S. Ghemawat, R. Gruber, U. Maheshwari, A. C. Myers and L. Shrira. Safe and efficient sharing of persistent objects in Thor. In SIGMOD (1996)

[3] M. Satyanarayanan, H. H. Mashburn, P. Kumar, D. C. Steere and J. J. Kistler. Lightweight recoverable virtual memory. In SOSP (1993)

[4] J. Condit, E. B. Nightingale , C. Frost, E. Ipek, B. Lee, D. Burger and D. Coetzee. Better I/O through byte addressable memory. In SOSP 22 (2009)

[5] R. F. Freitas and W. W. Wilcke. Storage-class memory: the next storage system technology. IBM J. Res. Dev. 52, 4 (2008)