



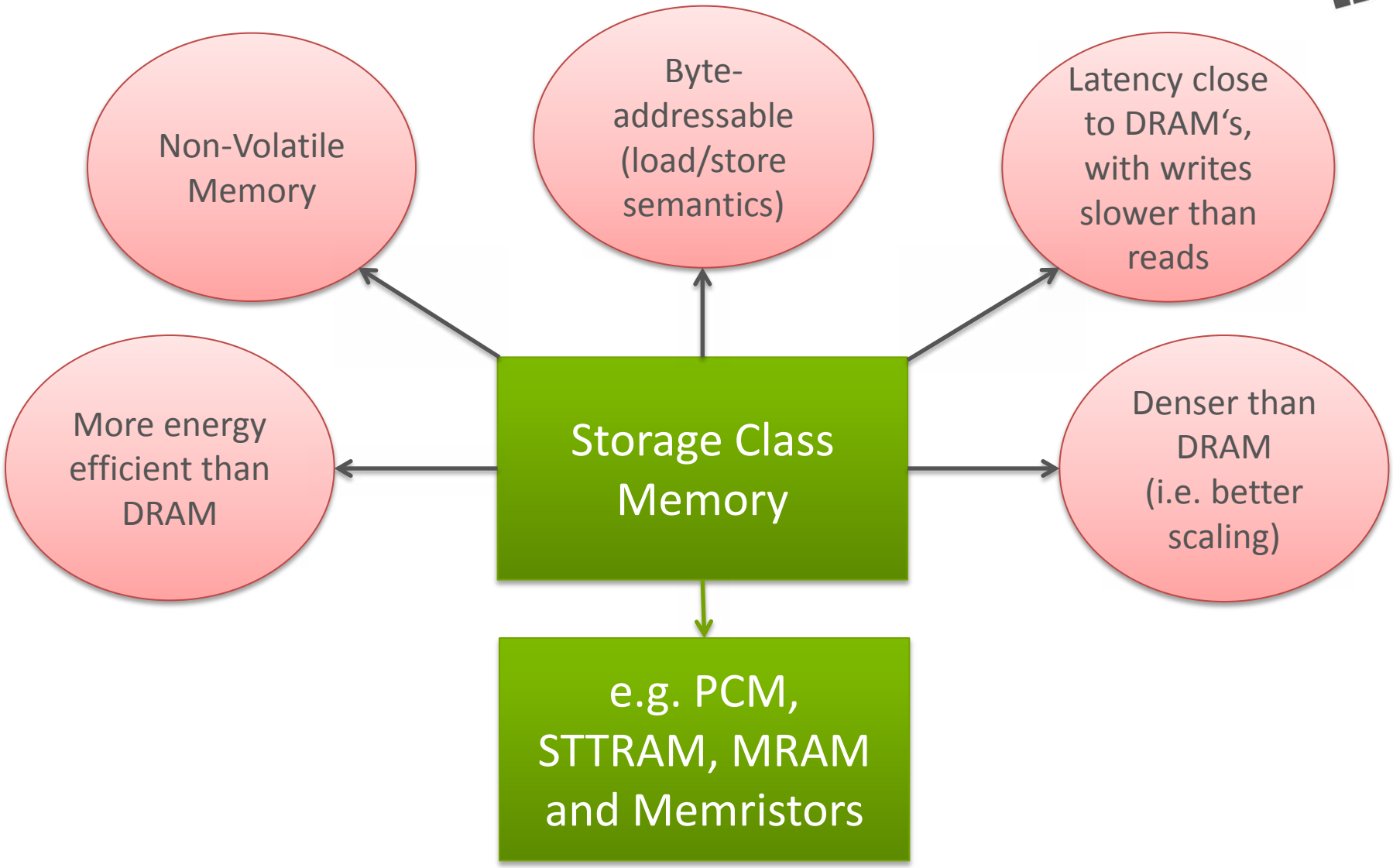
# *SOFORT: A Hybrid SCM-DRAM Storage Engine for Fast Data Recovery*

Ismail Oukid*°, Daniel Booss°, Wolfgang Lehner*, Peter Bumbulis°,
and Thomas Willhalm+

*Dresden University of Technology
°SAP AG
+ Intel GmbH

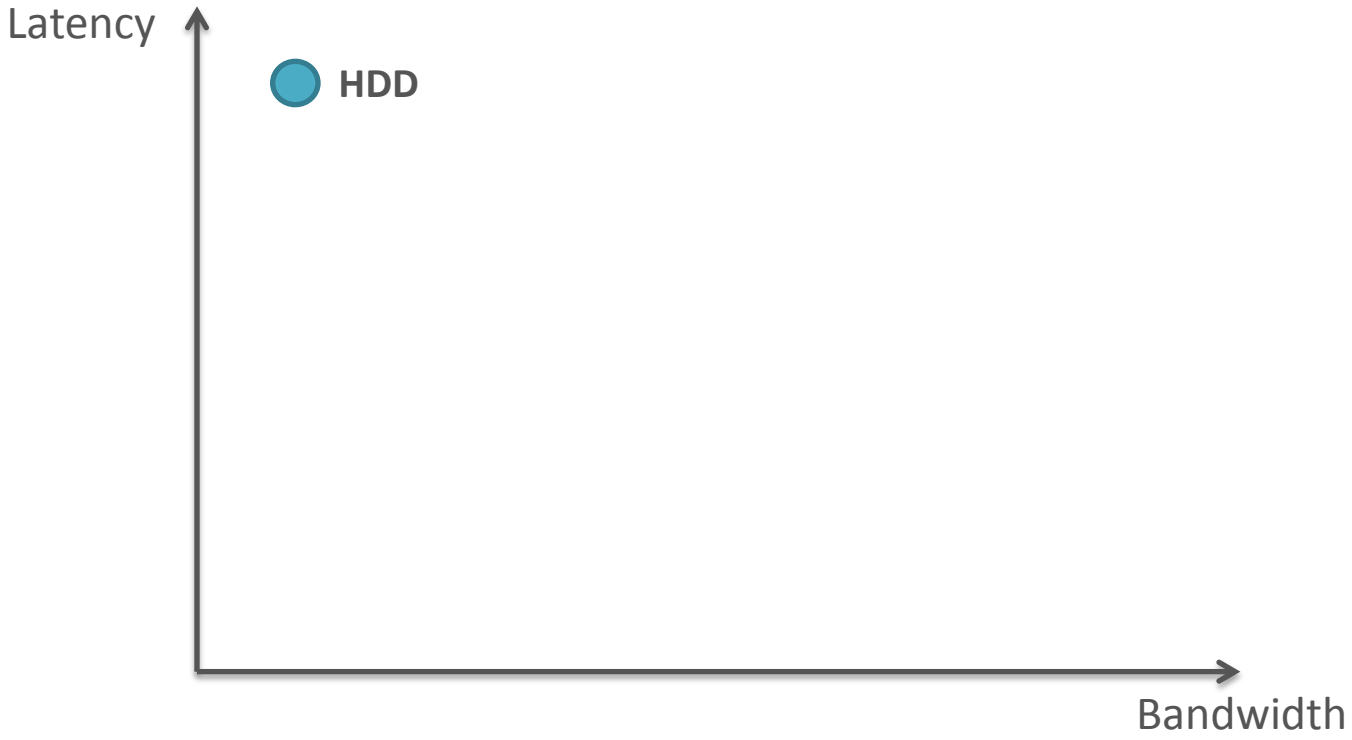
DaMoN 2014, Snowbird, Utah, USA, June 23, 2014

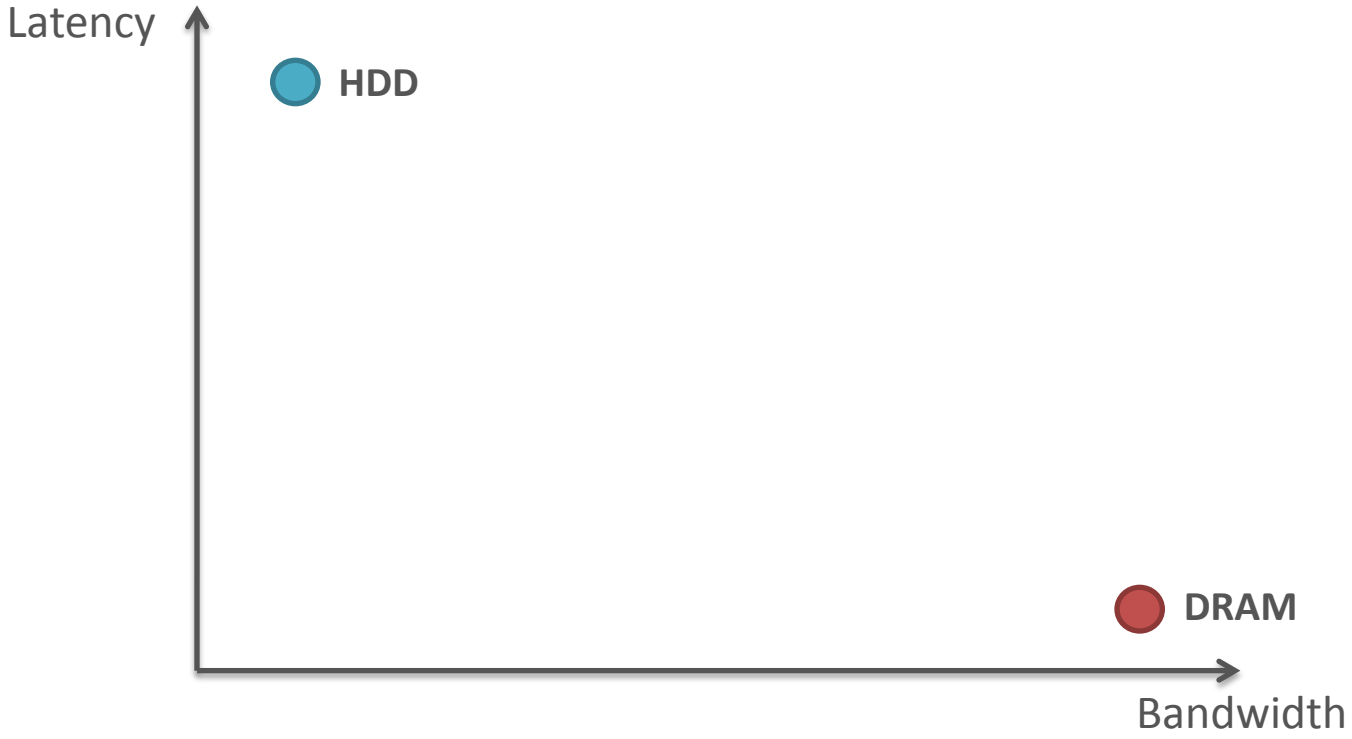
# What is Storage Class Memory?

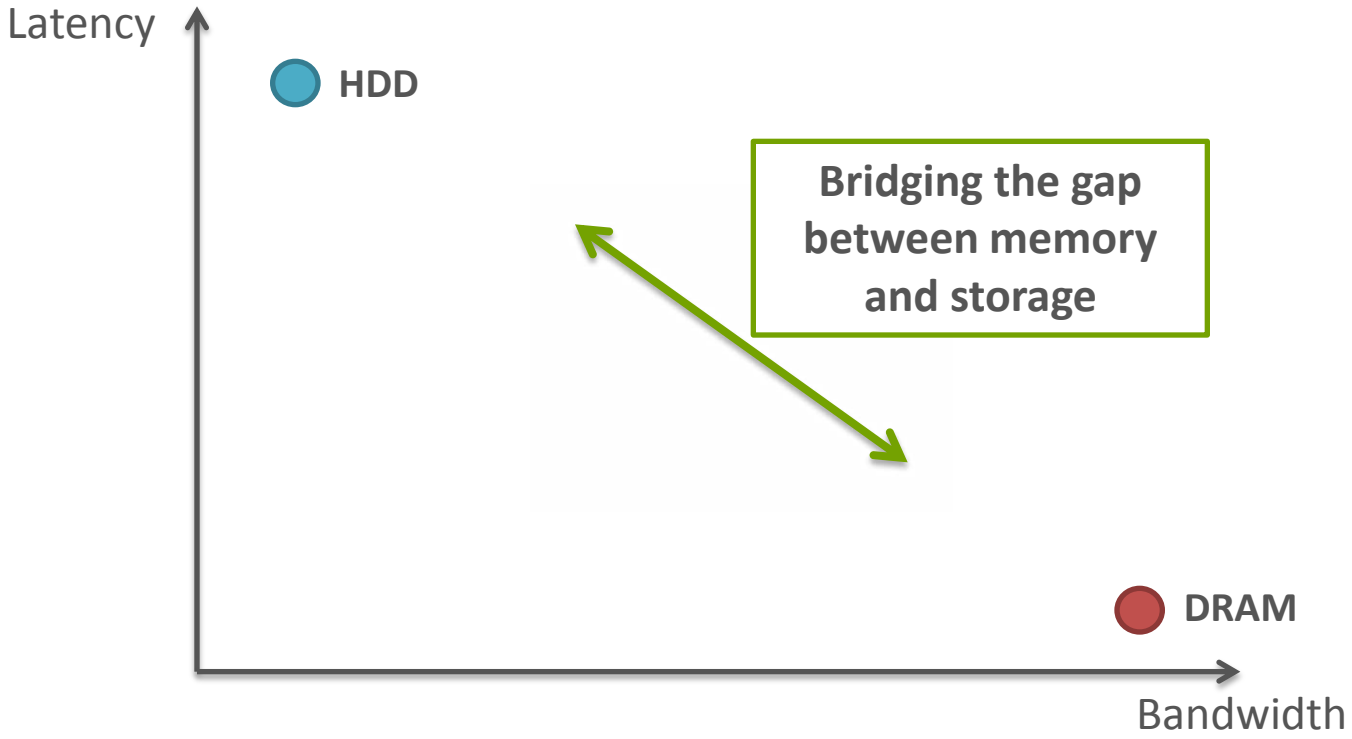


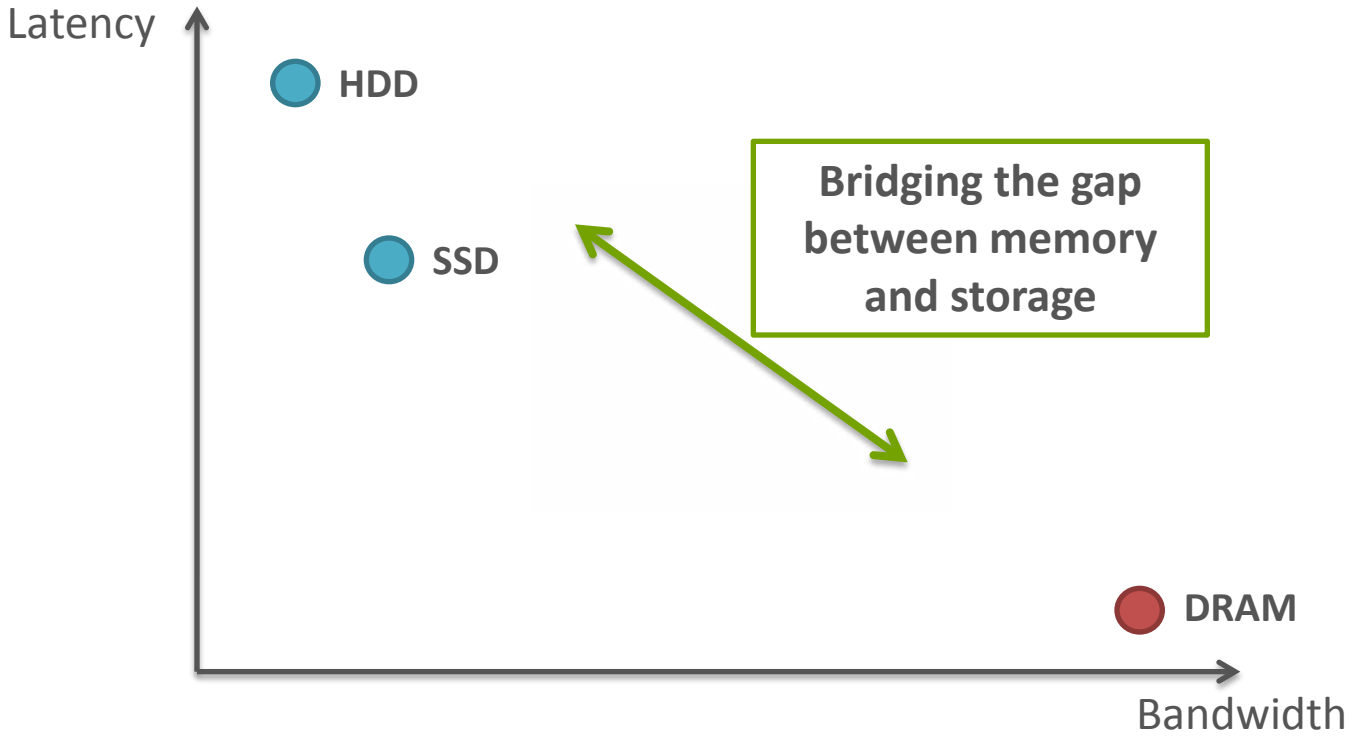

Latency
Bandwidth

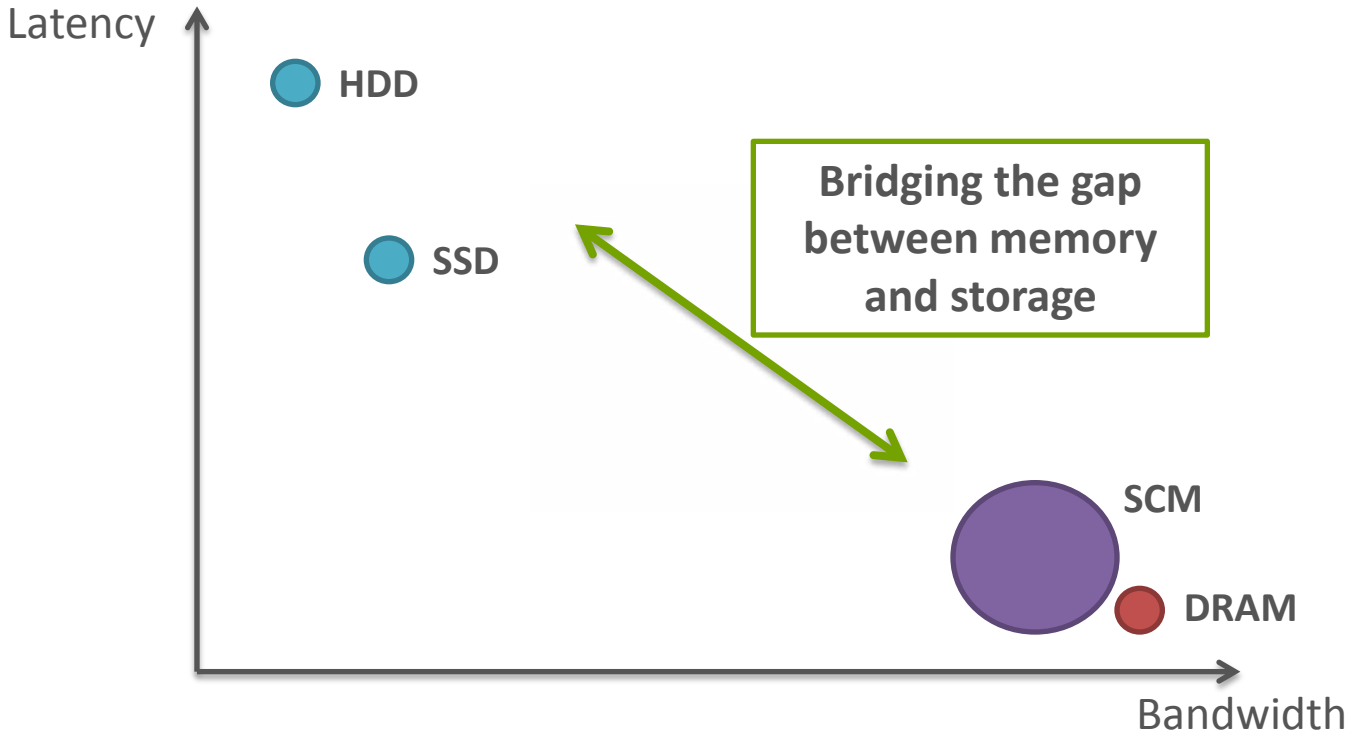
# SCM Compared to Other Technologies

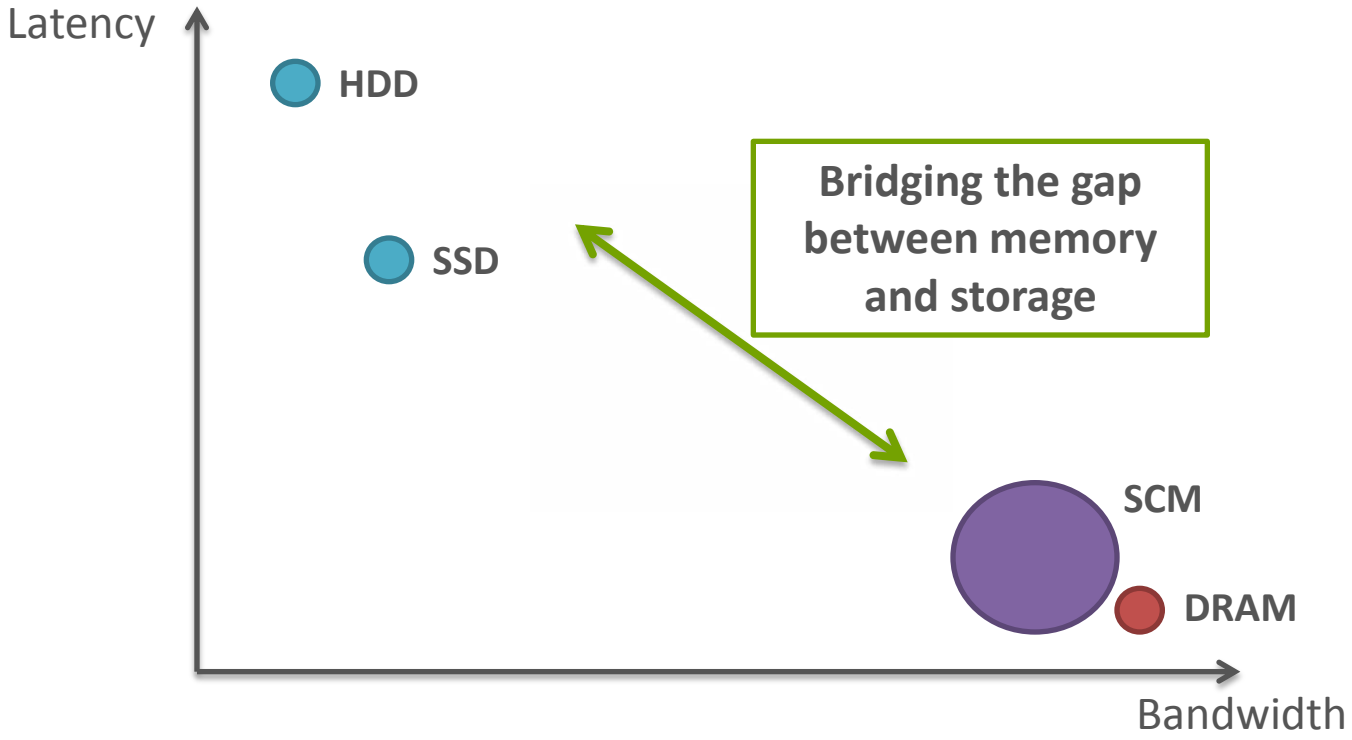



Latency
HDD
DRAM
Bandwidth


Latency
HDD
Bridging the gap between memory and storage
DRAM
Bandwidth


Latency
HDD
SSD
Bridging the gap between memory and storage
DRAM
Bandwidth


Latency
HDD
SSD
Bridging the gap between memory and storage
SCM
DRAM
Bandwidth




Storage Class Memory is a merging point between memory and storage



- *Availability guarantees are an important part of many SLAs*
- *Availability of 99% → downtime of 3.65 days/year!*

- *Many database crash conditions are transient*
- *A reasonable approach to recovery: restarting the database*
- *Database restart time impacts directly database availability*
- *Long restart times: a shortcoming of main memory databases*

***Improve database restart time using SCM***



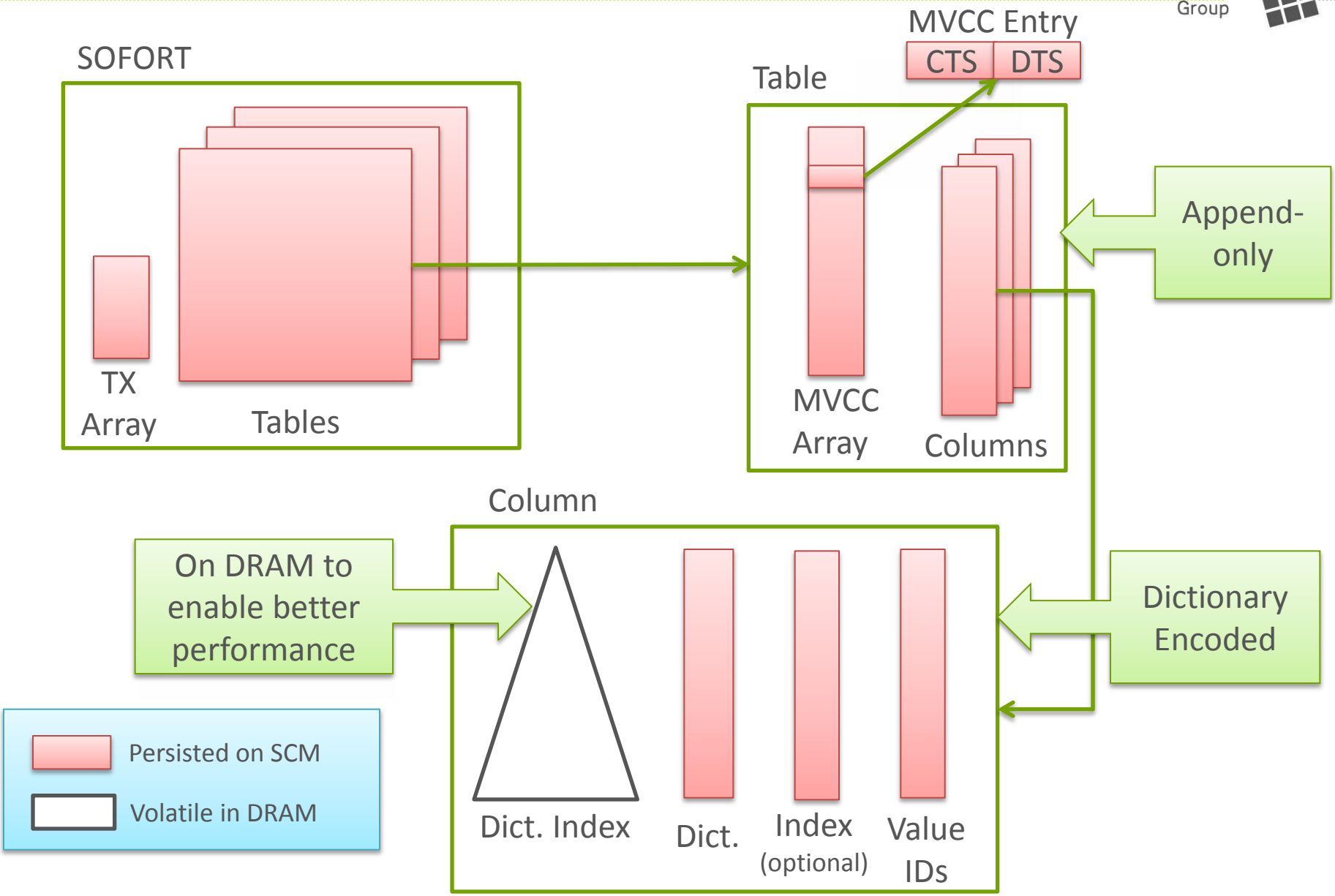
**SOFORT**: A hybrid SCM-DRAM storage engine for fast data recovery

Requirements:

- Fast data recovery
- Mixed OLTP and OLAP
- A hybrid SCM-DRAM memory environment

Architecture:

- **Single-level column-store**
- Dictionary encoded
- Multi version concurrency control (MVCC)
- No transactional log


SOFORT
TX Array
Tables
Table
MVCC Entry
CTS
DTS
MVCC Array
Columns
Append-only
Column
On DRAM to enable better performance
Dictionary Encoded
Dict. Index
Dict.
Index (optional)
Value IDs
Persisted on SCM
Volatile in DRAM



SCM accessed via the CPU cache: Writes not guaranteed to have reached SCM → **Need to flush data from cache**

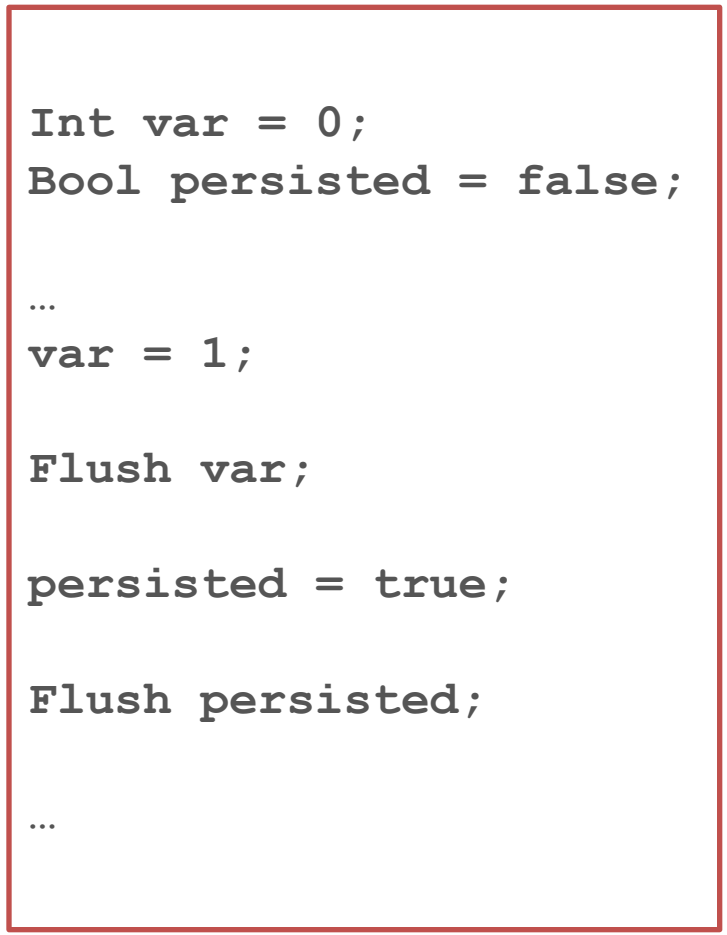
Code reordered at compilation time **&** Complex CPU out-of-order execution → **Need to order memory operations**

**Persistence primitives**: flushing instructions (CLFLUSH), memory barriers (MFENCE, SFENCE, LFENCE) and non-temporal stores (…)

Data may still be held in memory controller buffers
→ **Need to flush memory controller buffers on power failures**

# > Persisting Data on the Fly: Example





```
Int var = 0;
Bool persisted = false;

…
var = 1;

Flush var;

persisted = true;

Flush persisted;

…
```



```
Int var = 0;
Bool persisted = false;

…
var = 1;

Flush var;

persisted = true;

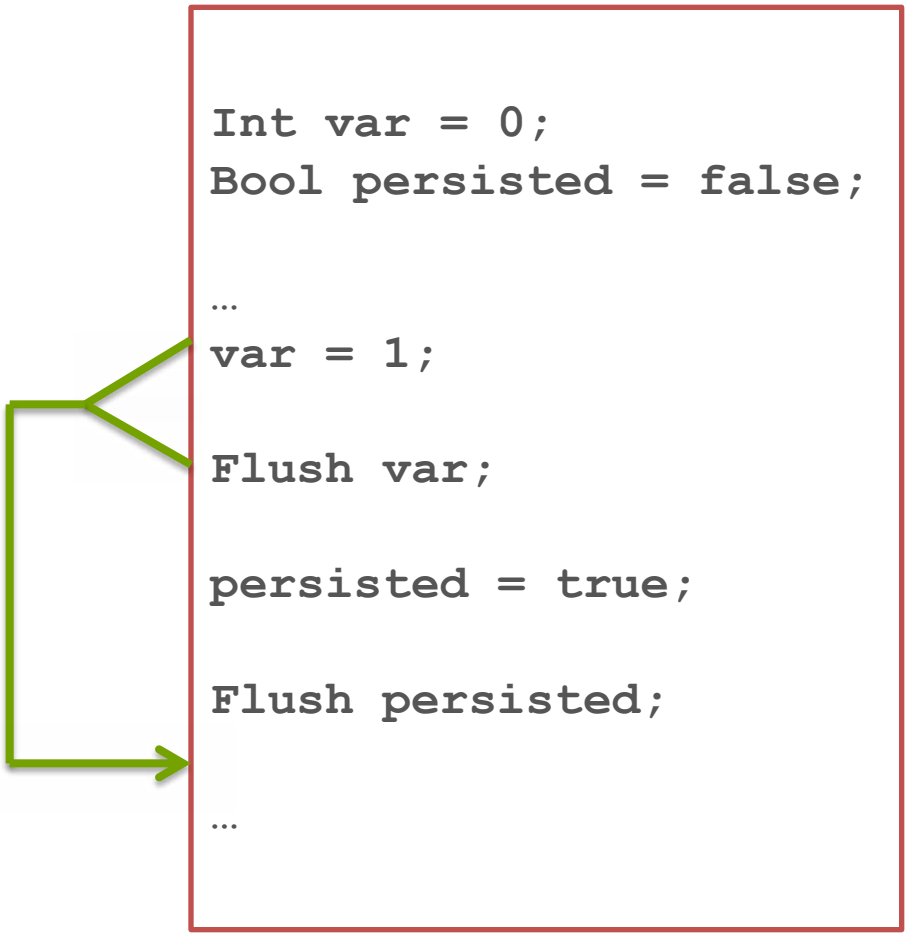
Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | False | False |

# > Persisting Data on the Fly: Example





```
Int var = 0;
Bool persisted = false;

…
var = 1;

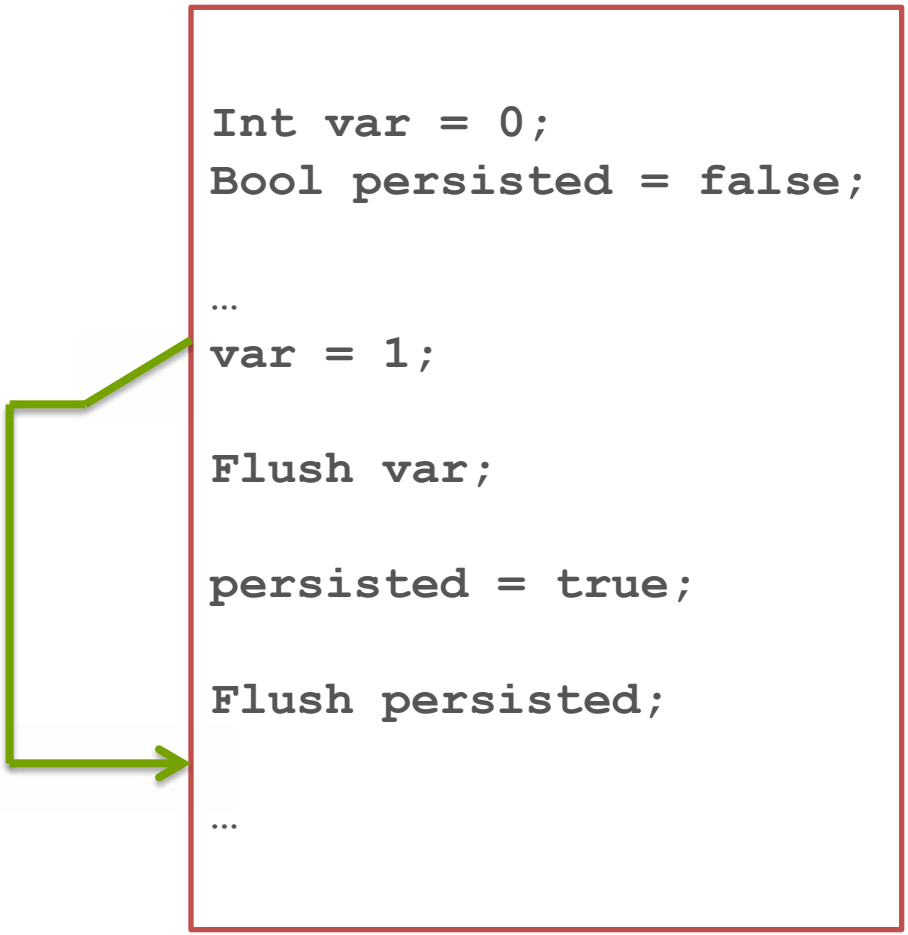
Flush var;

persisted = true;

Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | True | True |



```
Int var = 0;
Bool persisted = false;

…
var = 1;

Flush var;

persisted = true;

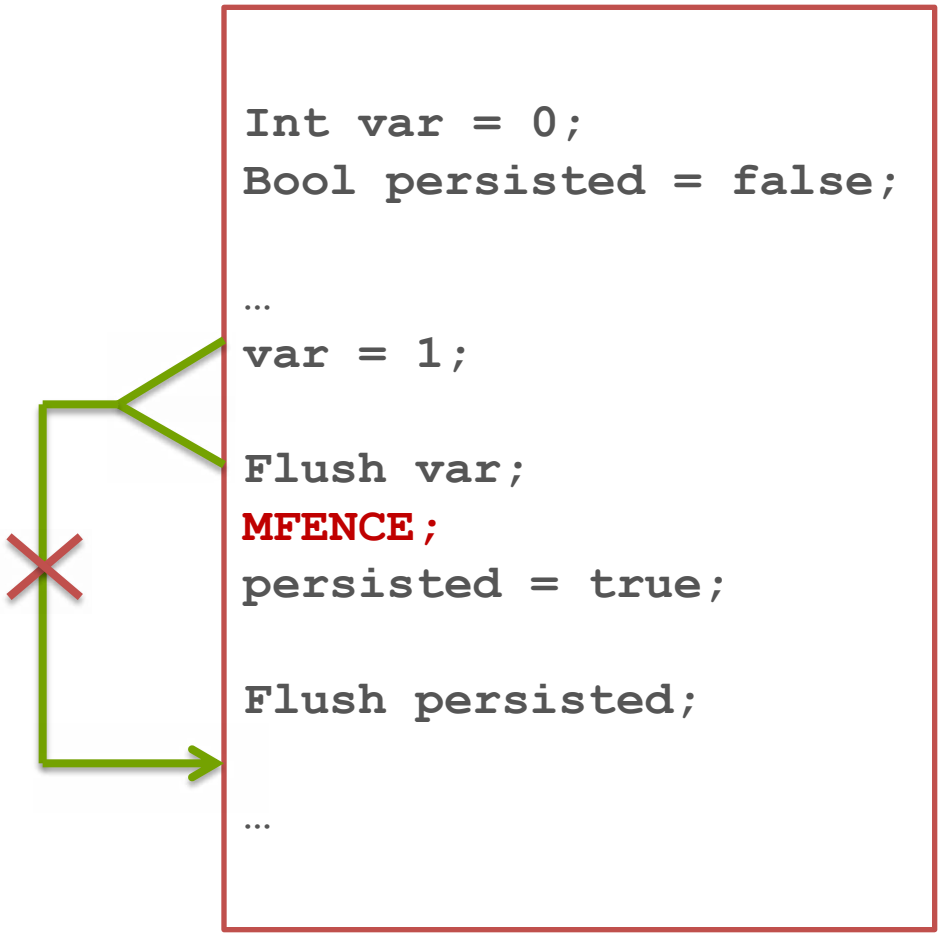
Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 1 | 0 |
| persisted | True | True |

# > Persisting Data on the Fly: Example





```
Int var = 0;
Bool persisted = false;

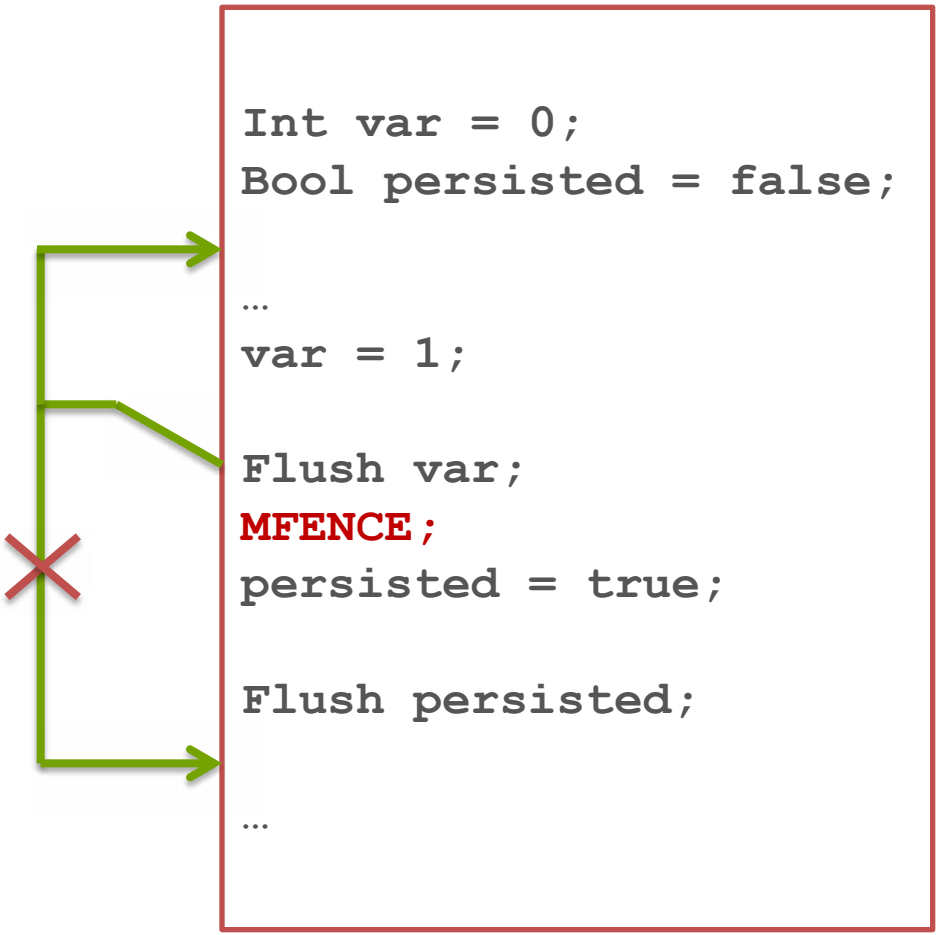
…
var = 1;

Flush var;
MFENCE;
persisted = true;

Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | False | False |



```
Int var = 0;
Bool persisted = false;

…
var = 1;

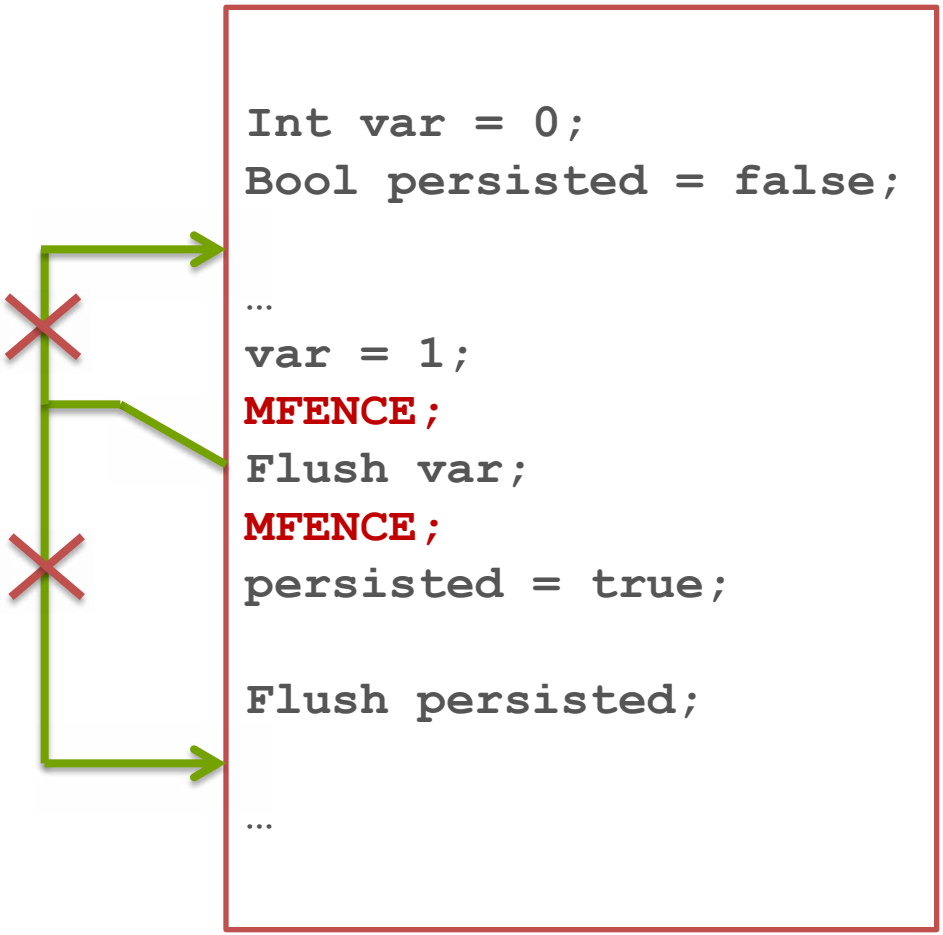
Flush var;
MFENCE;
persisted = true;

Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | False | False |

# > Persisting Data on the Fly: Example



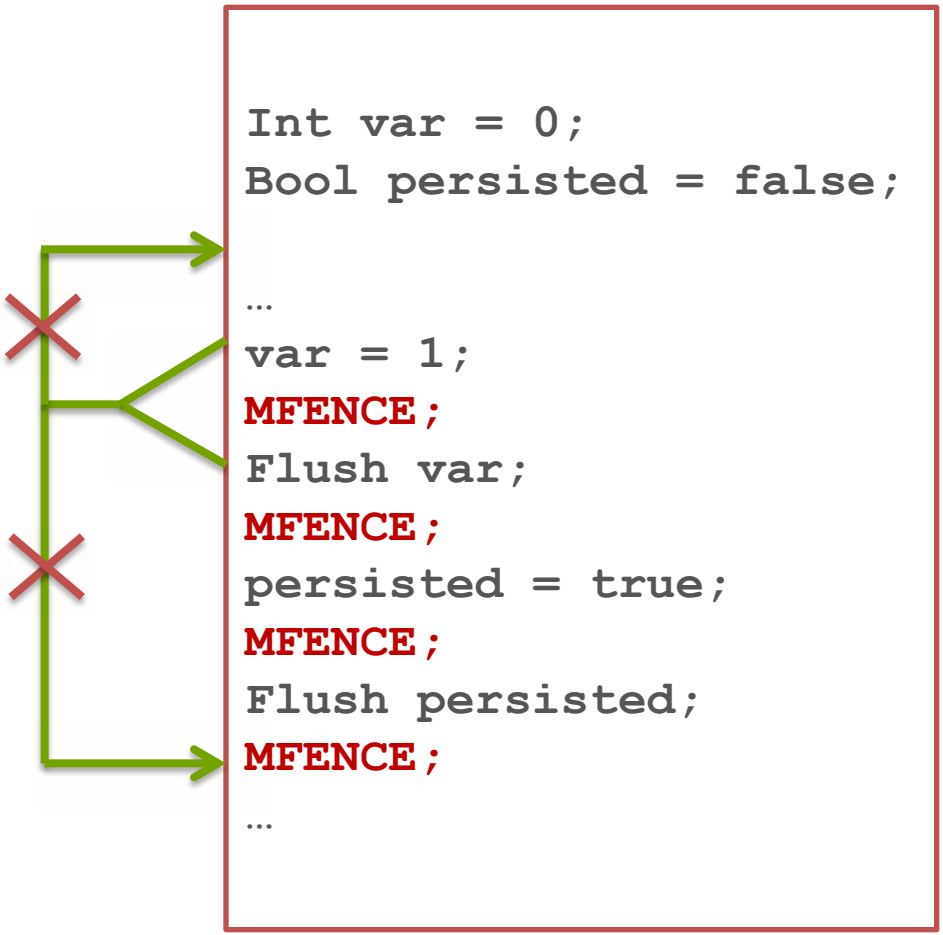
```
Int var = 0;
Bool persisted = false;

…
var = 1;
MFENCE;
Flush var;
MFENCE;
persisted = true;

Flush persisted;

…
```

| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | False | False |



```
Int var = 0;
Bool persisted = false;

…
var = 1;
MFENCE;
Flush var;
MFENCE;
persisted = true;
MFENCE;
Flush persisted;
MFENCE;
…
```

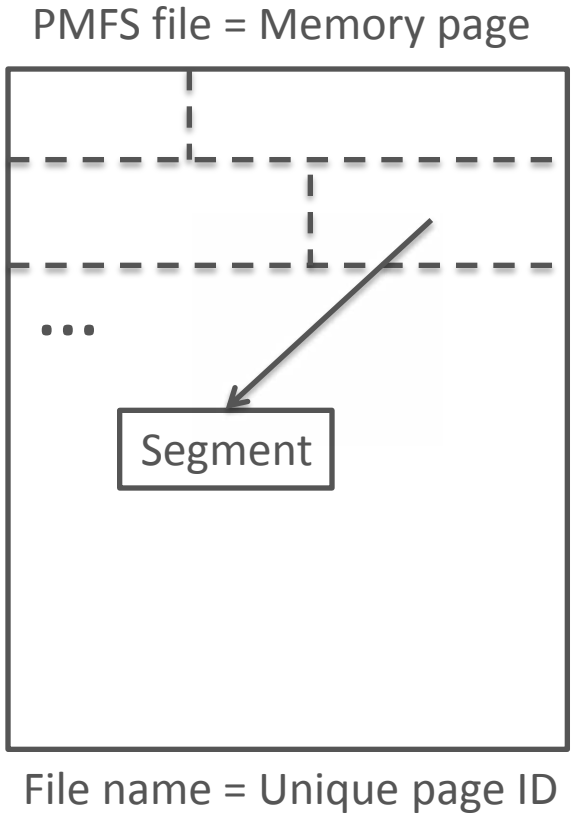
| | Cache | SCM |
|---|---|---|
| var | 0 | 0 |
| persisted | False | False |

***Persistent Memory File System (PMFS):***

- *SCM-aware file system*
- *No buffering in DRAM on mmap → direct access to SCM*

***PMAllocator:***

- *Huge PMFS files as memory pages*
- *Pages cut into segments for allocation*
- *Persistent counter of memory pages*
- *Mapping from persistent memory to VM*





Regular pointers are bound to the program‘s address space
→ Cannot be used for recovery

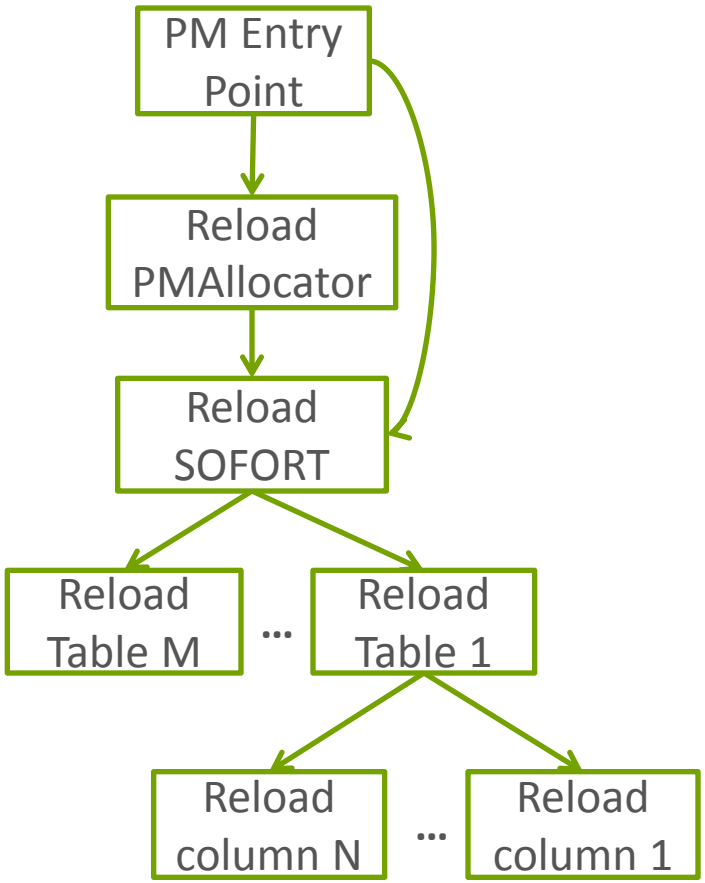
***We propose Persistent Memory Pointers (PMPtrs):***

```
Struct PMPtr {
        int64_t    m_BaseID;
        ptrdiff_t  m_Offset;
}
```

Persistent memory page ID

Offset indicating the start of the allocated block

- Can be converted (swizzled) to regular pointers
- PMPtrs stay valid across failures




- A PMPtr to the PMAllocator
- A PMPtr to the storage engine

- Reload memory pages
- Reconstruct mapping to VM

- Rollback unfinished transactions
- Allocate ressources for recovery

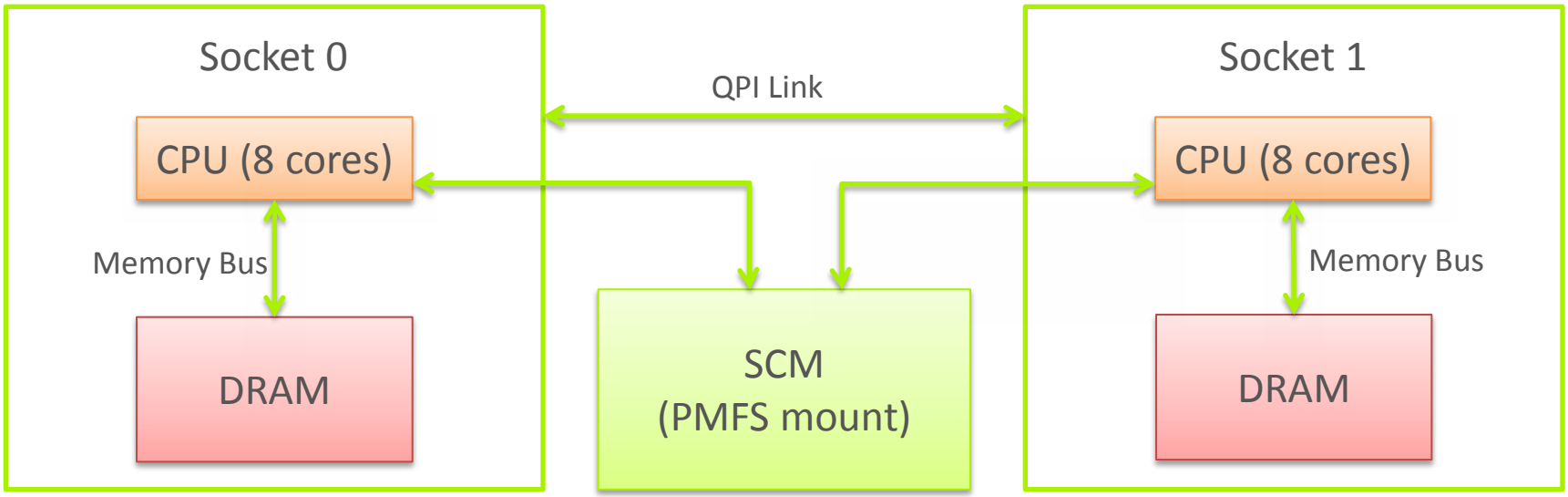
- Data structures sanity check
- Initiate columns recovery

- Data structures sanity check
- Reconstruct dictionary indexes



***Hardware-based SCM simulation:***

- *Intel Xeon E5 @2.60Ghz, 20MB L3 cache, 8 physical cores/socket*
- *Avoiding NUMA effects: benchmark run on a single socket*
- *Limitation: symmetric instead of asymmetric read/write latency*





***TATP benchmark:***

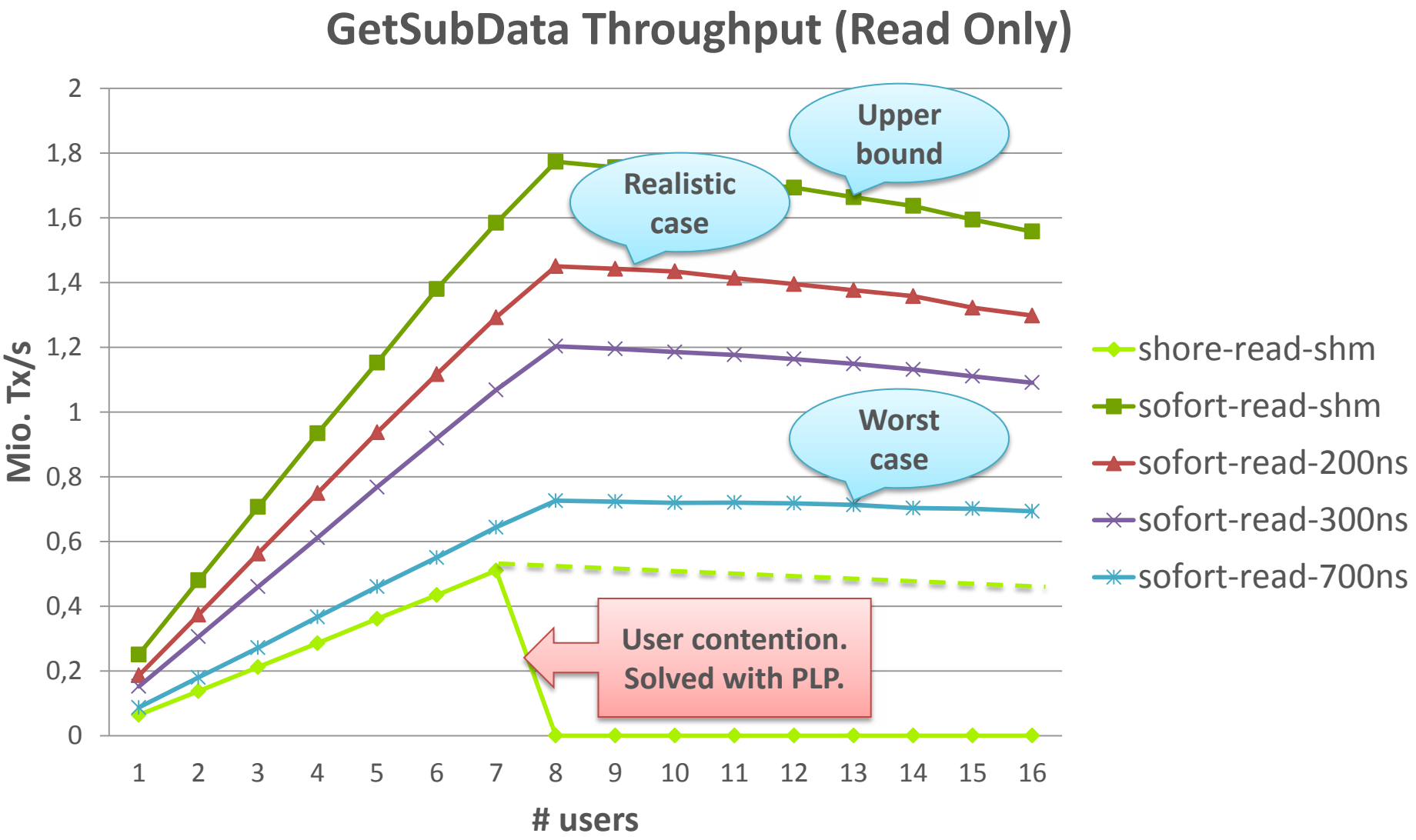
- *Simple but realistic OLTP benchmark (20% write, 80% reads)*
- *Simulates a telecommunication application*
- *Initial population of 1 million subscribers (~2GB data size)*

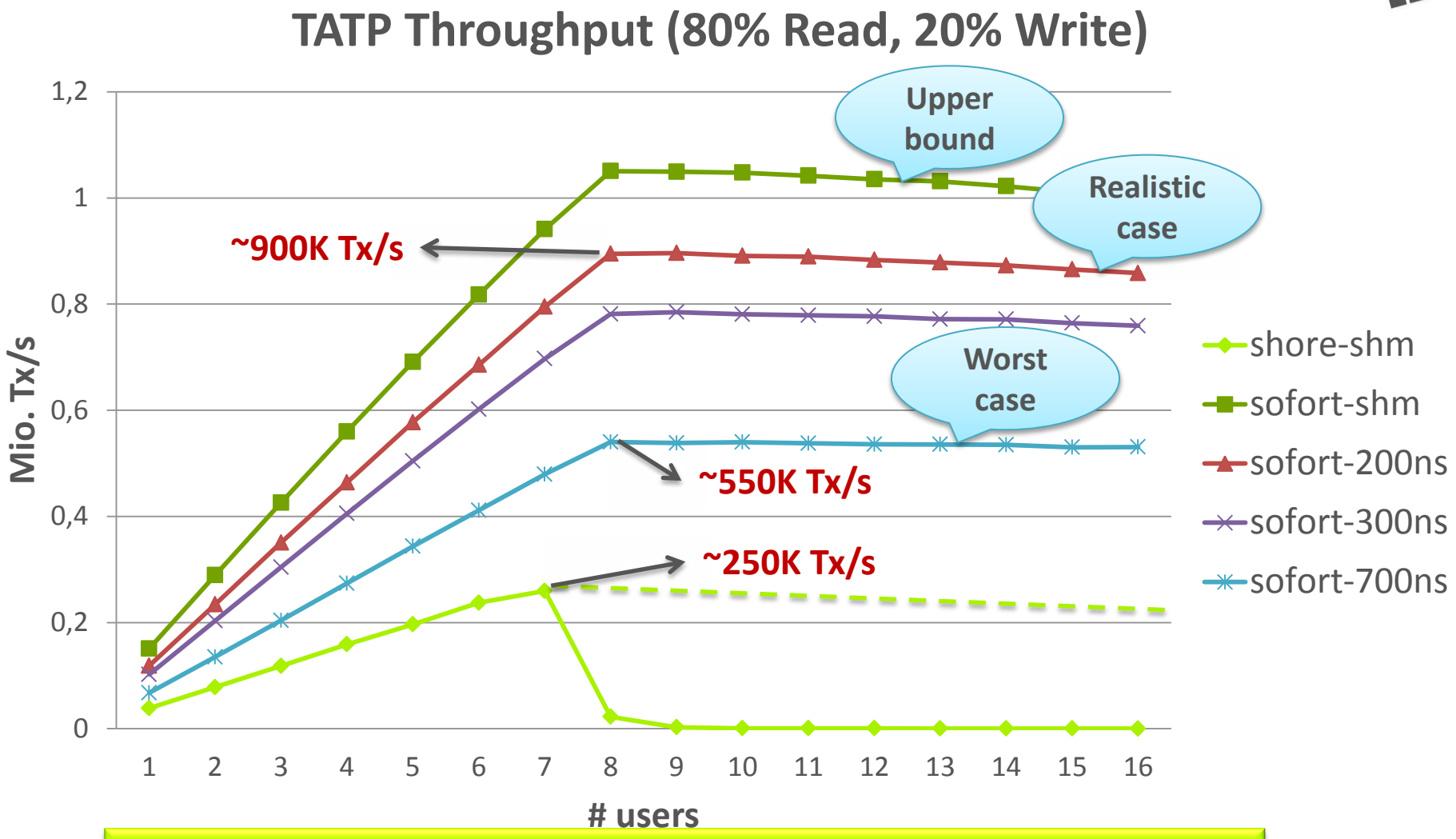
***Shore-MT:***

- *A storage manager designed for scalability in multicores*
- *Run on Ramdisk (tempFS)*

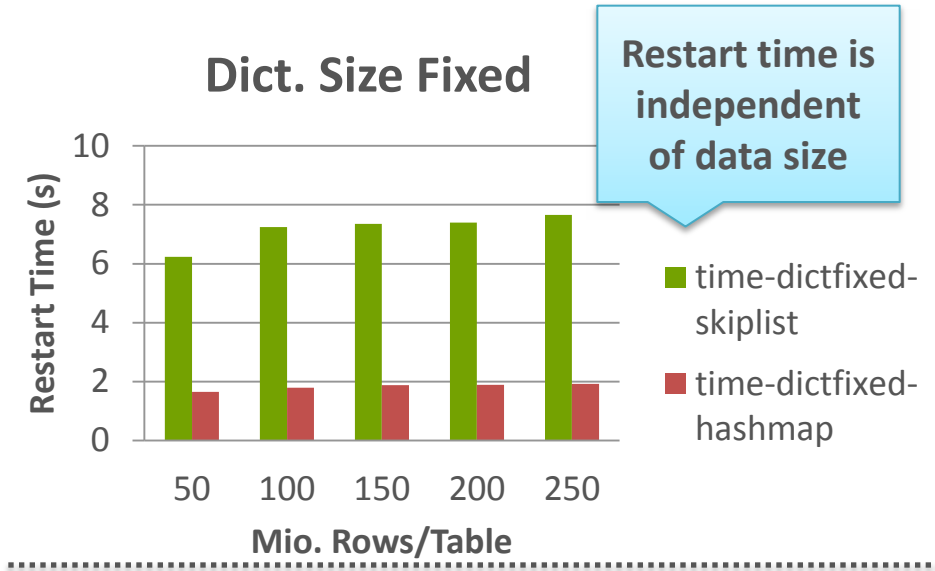
***Restart time:***

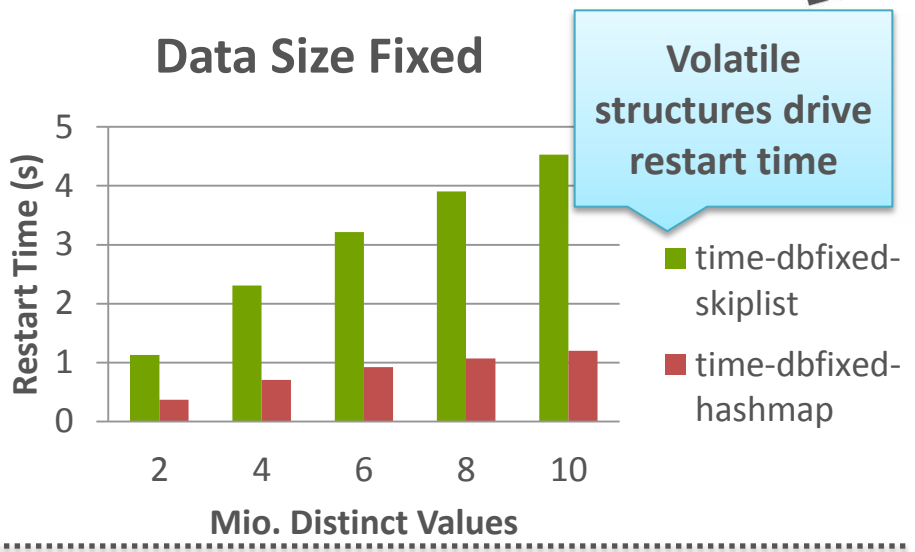
- *Single, 4 column table*
- *SCM latency set to 200ns*

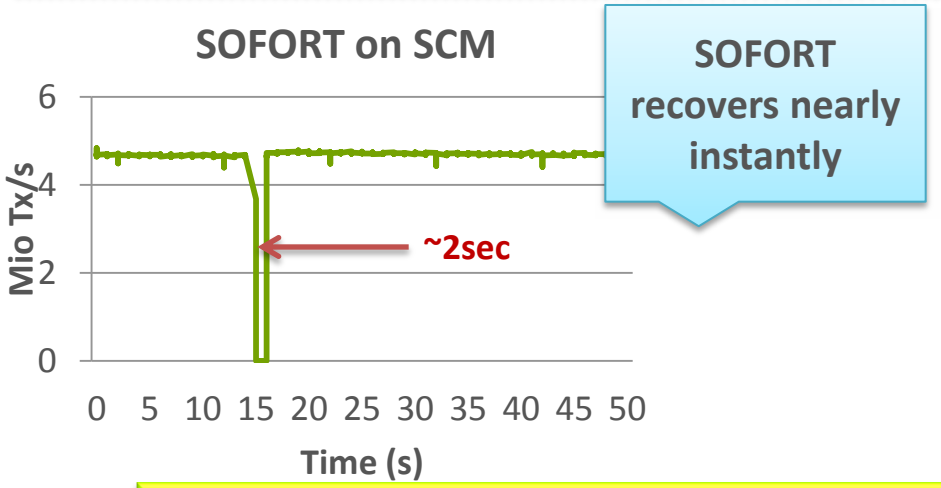

GetSubData Throughput (Read Only)
Mio. Tx/s
# users
Upper bound
Realistic case
Worst case
User contention. Solved with PLP.
shore-read-shm
sofort-read-shm
sofort-read-200ns
sofort-read-300ns
sofort-read-700ns

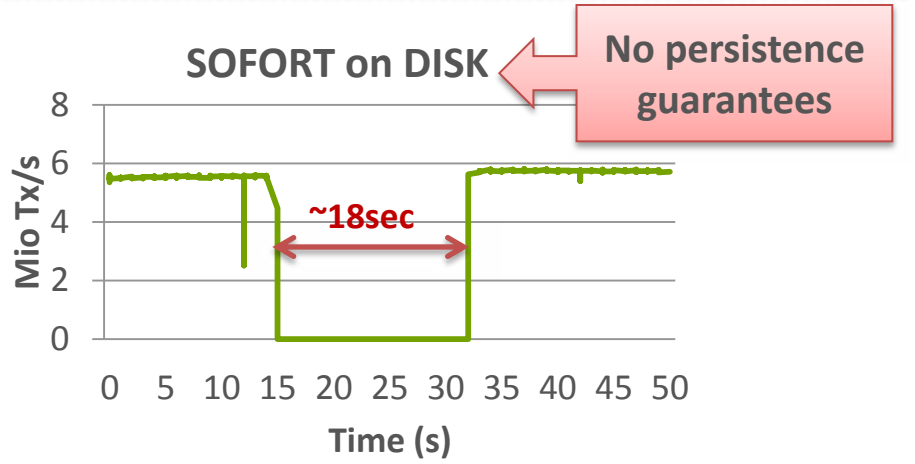



SOFORT outperforms Shore-MT even in a high latency SCM environment




**Restart time is independent of instance size and depends only on dictionaries size**



*We showed that SCM can help getting rid of the transactional log and significantly improving database restart times*

*Latency study based on hardware simulation showed that SOFORT exhibits competitive performance even in a high latency SCM environment*

*Future Work:*

- Extend SOFORT to support long transactions
- Explore new recovery schemes
- Design persistent index structures
- Experiment with write-through caching policy





**Takeaway**: SCM is a game changer in the database industry
Make the change happen!

*Thank You! Questions? Comments?*

Ismail Oukid

*SOFORT: A Hybrid SCM-DRAM Storage Engine for Fast Data Recovery*

Ismail Oukid, Daniel Booss, Wolfgang Lehner, Peter Bumbulis, and Thomas Willhalm+

DaMoN 2014, Snowbird, Utah, USA, June 23, 2014