Adaptive Recovery for SCM-Enabled Databases

Ismail Oukid (TU Dresden & SAP), Daniel Bossle* (SAP), Anisoara Nica (SAP),
Peter Bumbulis (SAP), Wolfgang Lehner (TU Dresden), Thomas Willhalm (Intel)

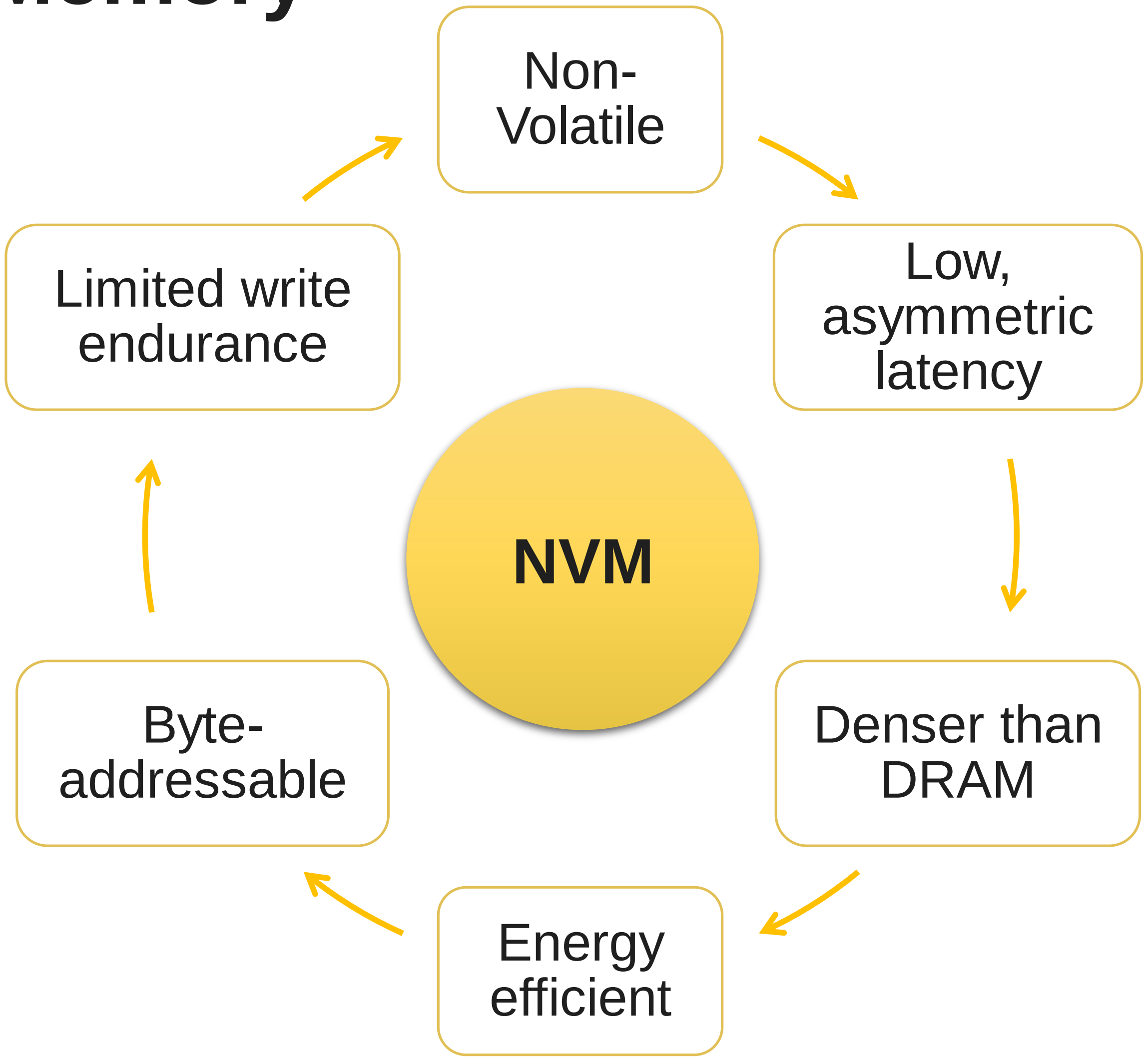
* Contributed while interning at SAP

ADMS@VLDB, September 1st, 2017

PUBLIC

Non-Volatile Memory

We assume hardware-based wear-leveling

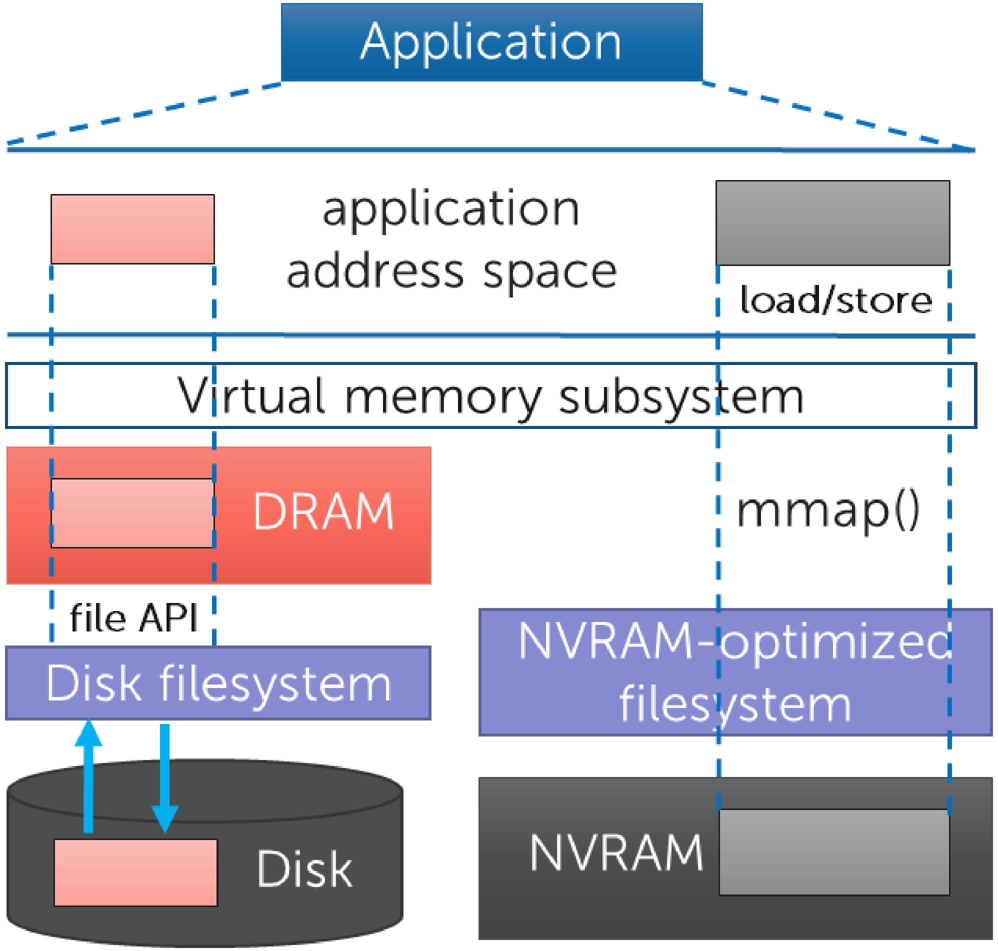


Writes noticeably slower than reads

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

NVM is a merging point between main memory and storage

Architecting NVM

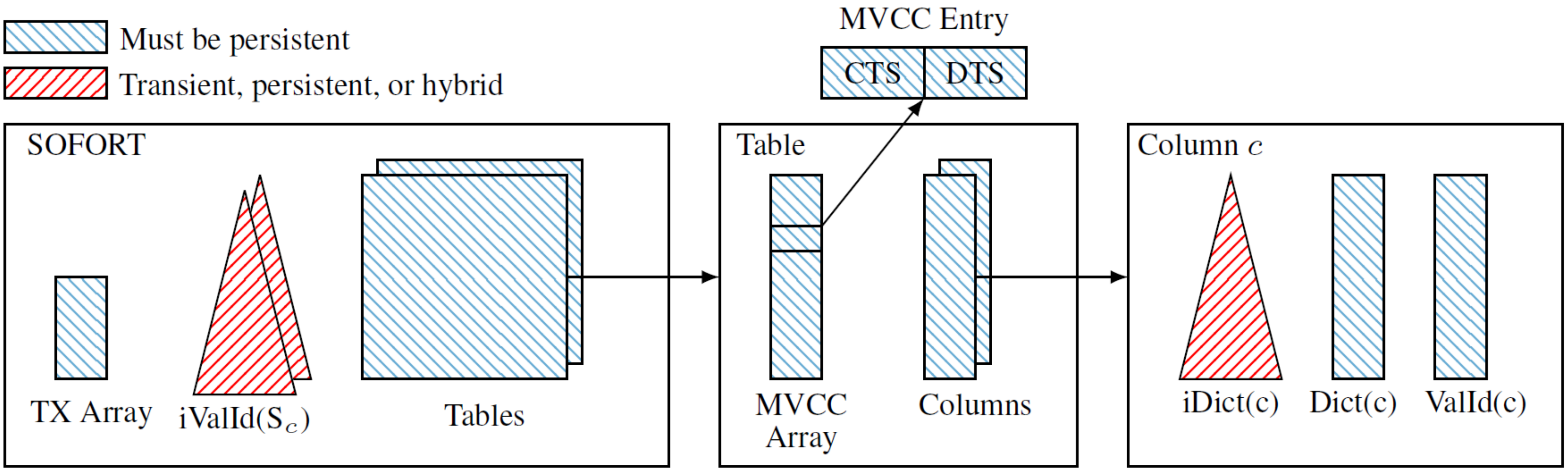


- An NVM-optimized filesystem provides zero-copy mmap
- Direct access to NVM via load/store instructions
- Several filesystem proposals: NOVA, PMFS, SCMFS, etc.
- Linux ext4 and xfs already provide Direct Access support

NVM may become a universal memory

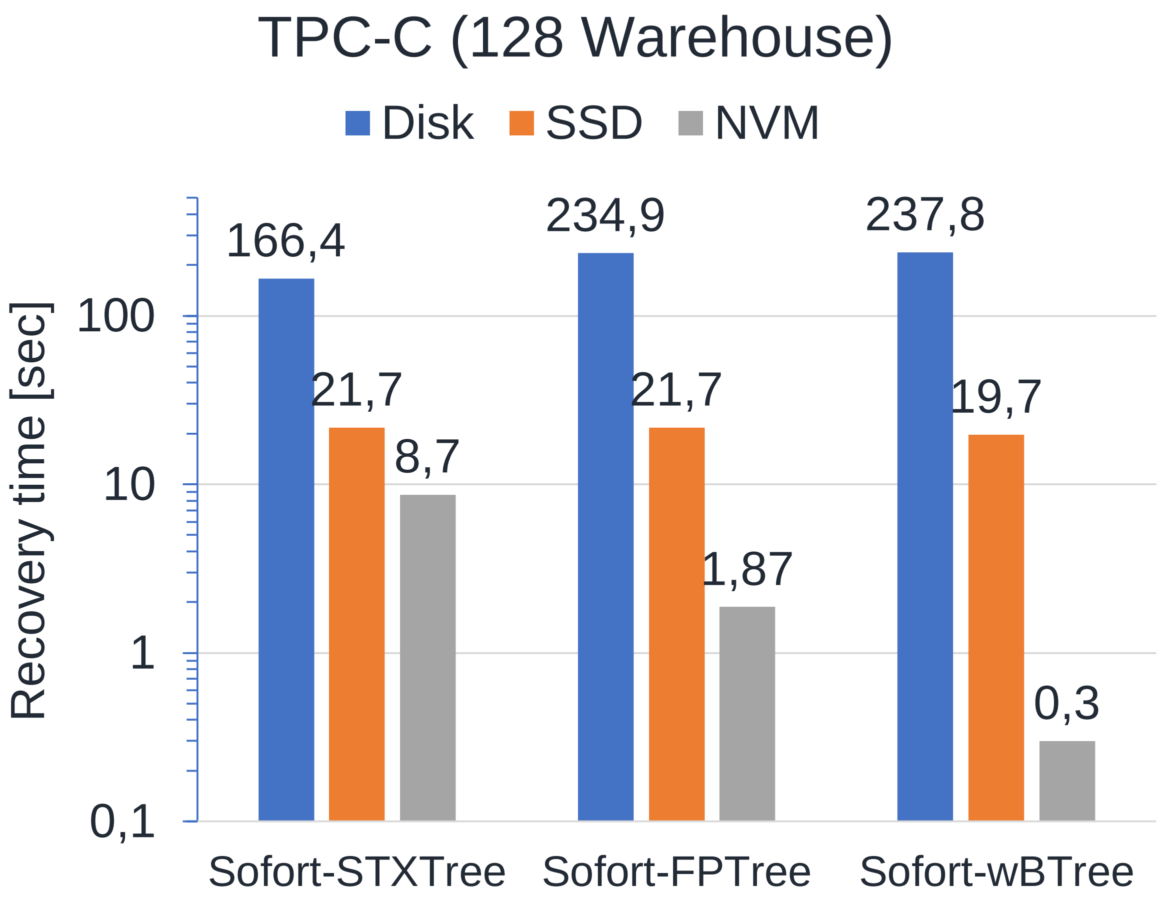
SOFORT: A Hybrid NVM-DRAM Storage Engine

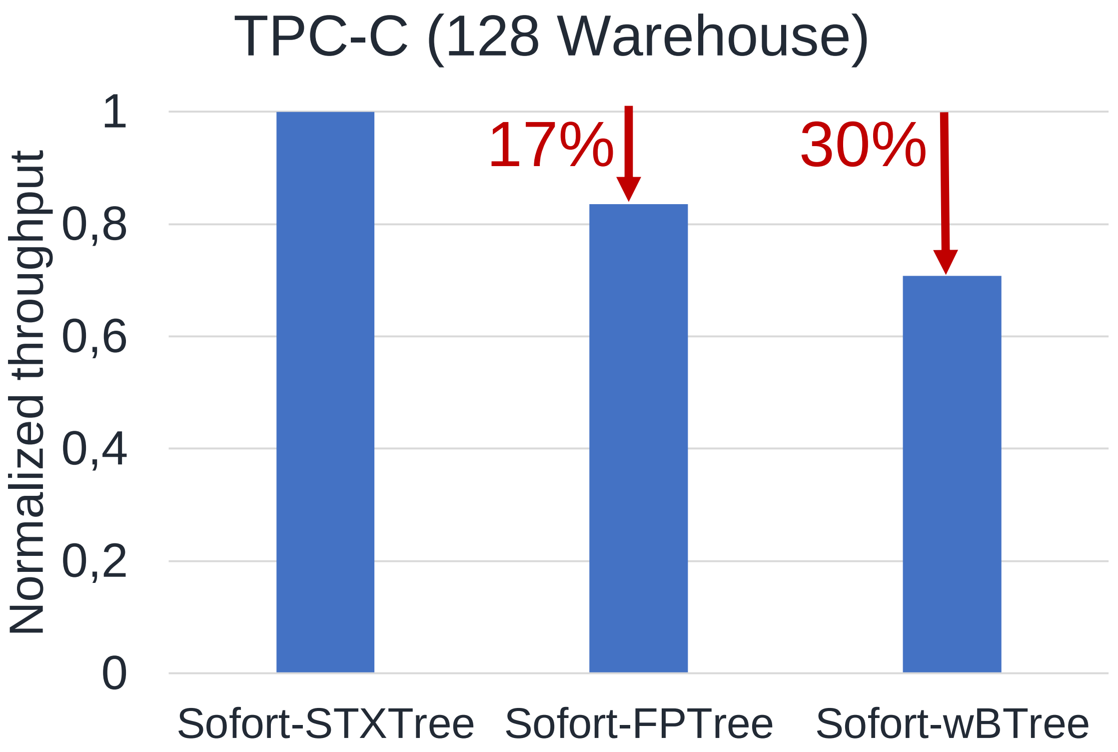
→ Primary data persisted in and accessed from NVM
→ Secondary data can be persistent, transient, or hybrid



NVM enables a single-level database storage architecture

Recovery Time





Rebuilding secondary data is the new recovery bottleneck

Goal: improve recovery without compromising query performance

Synchronous Recovery

Recovery protocol

1. Recover primary data
2. Undo in-flight TXs
3. Rebuild secondary data
4. Accept queries

\+ Secondary data rebuilt as fast as possible

\- System is not responsive during recovery

Asynchronous Recovery (aka Instant Recovery)

Intuition: Primary data is sufficient to answer queries

Accept queries right after recovering primary data

During recovery

- Dictionary index lookup replaced with dictionary array scan
- Column index lookups replaced by column scans
- CPU resources split between query processing and recovery

\+ Near-instant responsiveness of the database

\- It takes longer to reach pre-failure throughput

Adaptive Recovery

Intuition 1: Secondary data structures are not equally important

Recover indexes in the order of their importance

Intuition 2: Some secondary data structures are not useful for the currently running workload

Release recovery resources after rebuilding important indexes

How to decide the importance of secondary data structures?

Index Benefit Functions

`Benefit_indep`

Computes the benefit of an index for a query plan independently of other indexes

`Benefit_dep`

Computes the benefit of an index while captures its dependencies to currently available indexes

S: Set of all indexes s: Considered index Q: Considered query

$$Benefit(s,Q,S) = Cost(Q,S_{NVM}) - Cost(Q,S_{NVM} \cup \{s\})$$

$$Benefit(s,Q,S) = Cost(Q, S(t_Q)) - Cost(Q,S(t_Q) \cup \{s\})$$

S_{NVM}: Set of persistent indexes

$S(t_Q)$: Set of available indexes at time t_Q

Ranking of Secondary Data Structures

Workload before failure (WoPast) — Workload during recovery (WoRestart) → Time

WoPast and WoRestart can be similar or different
→ A ranking function must take both into consideration

```
rank(s,t) = α(n)*Benefit(s,WoPast,S)
            +(1-α(n))*Benefit(s,WoRestart(t),S)
            -rebuild(s)
```

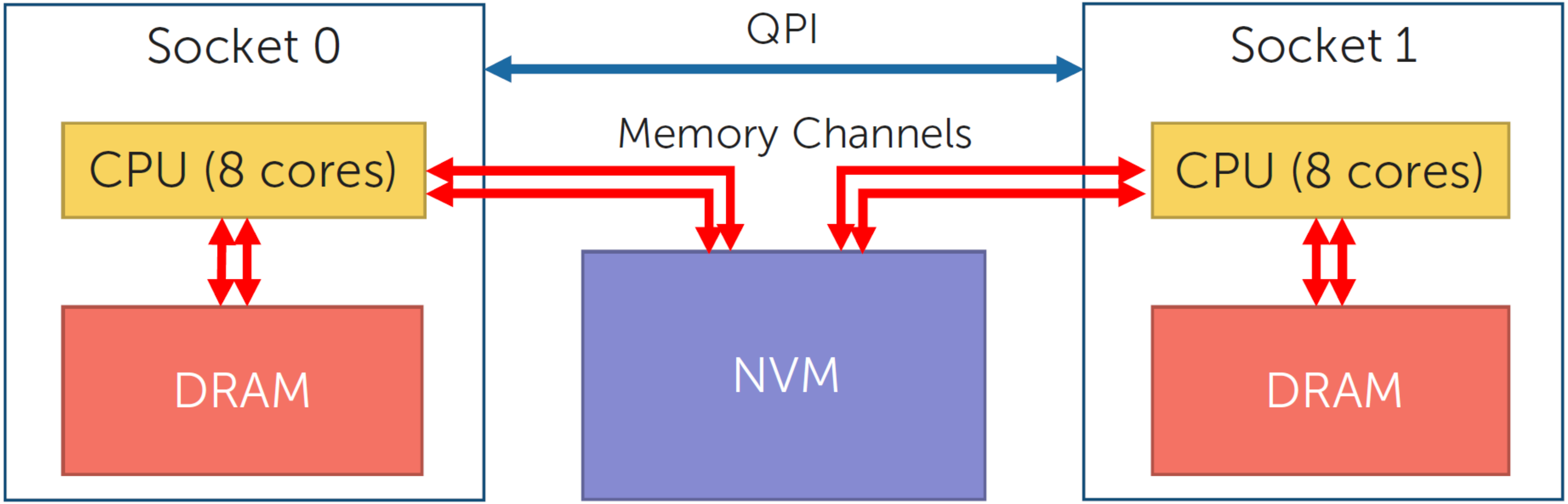
Weighted benefits on WoPast and WoRestart

Cost of index rebuild

e.g., $\alpha(n) = \alpha^n$ and $0 < \alpha < 1$
→ WoPast's weight decays with the number of statements in WoRestart

Evaluation Setup

- Intel NVM Emulator
  → Intel Xeon E5 @2.60Ghz, 20MB L3 cache, 8 physical cores
- Benchmarks run on a single socket



Experimental Setup (Cont'd)

TATP Benchmark
- 80% Get Subscriber Data
- 20% Update Location

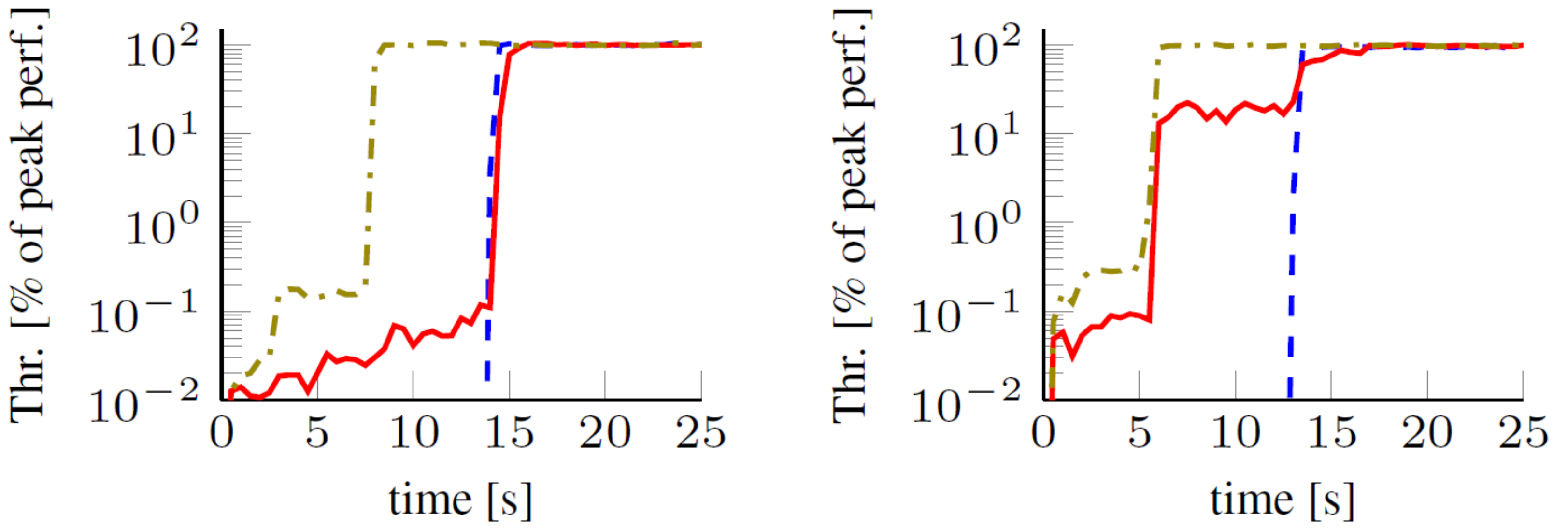
TPC-C Benchmark
- 50% Order Status
- 50% Stock Level

Sofort Configuration
- 8 users (threads)
- All indexes in DRAM

Experiment	*Query Processing Resources*	*Recovery Resources*	*Ranking*
$rk.Qx.Rz$	static x cores	static z cores	yes
$\neg rk.Qx.Rz$	static x cores	static z cores	no
$rk.Q^{ad}.Rz^{ad}$	adaptive	adaptive, start with z cores	yes

Recovery Strategies

Recovery workload same as pre-failure workload



(a) TATP (b) TPCC

Asynchronous recovery worse than synchronous recovery!

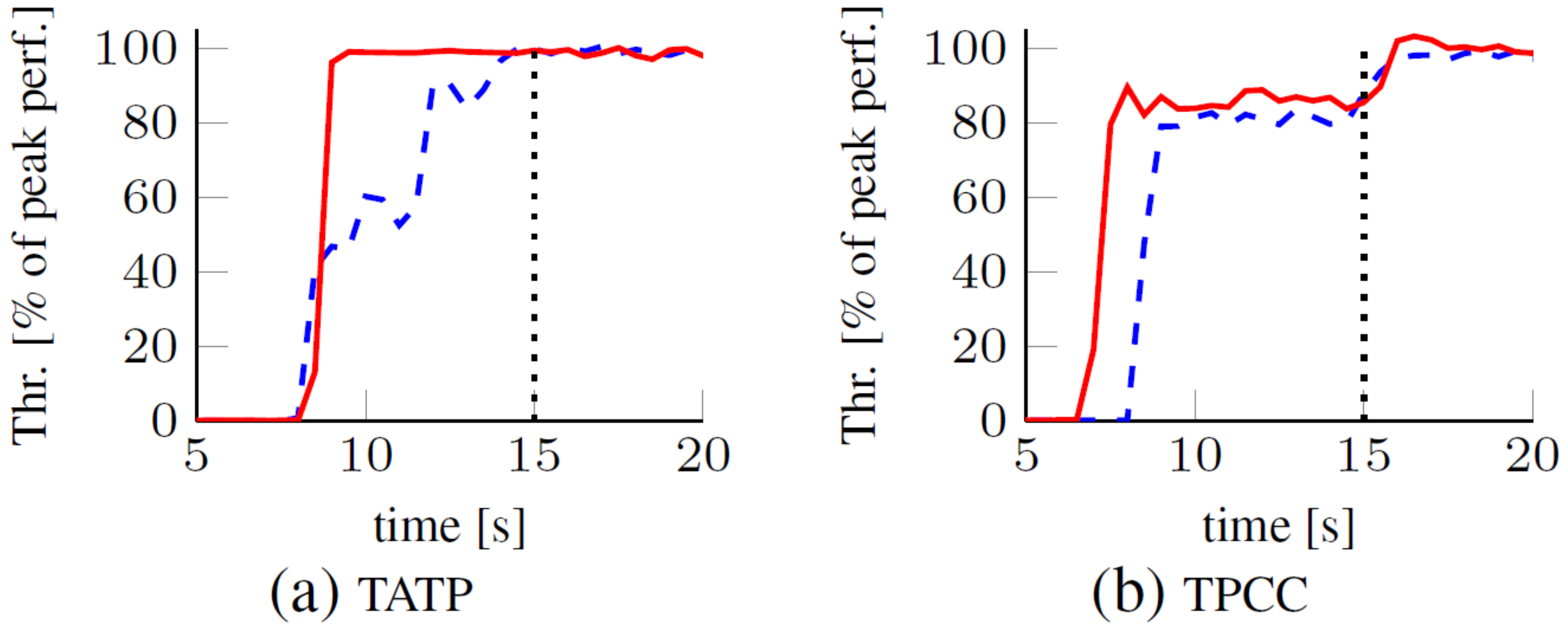
Legend Entry	TATP		TPCC	
	Rec. End	#TXs t=15 s	Rec. End	#TXs t=15 s
$\neg rk.Q0.R8$	13.9 s	0.9 M	13 s	169 K
$\neg rk.Q2.R6$	15.5 s	0.4 M	16.5 s	246 K
$rk.Q^{ad}.R6^{ad}$	63.7 s	6.4 M	65.2 s	838 K

Pre-failure throughput regained before the end of recovery

Workload Change During Recovery

Pre-failure workload: Full TATP and TPC-C mixes

Recovery workload: only TATP and TPC-C queries presented earlier



Legend Entry	TATP		TPCC	
	Rec. End	#TXs t=15 s	Rec. End	#TXs t=15 s
no adapt to WoRes.	28.1 s	4.4 M	59.5 s	498 K
adapt to WoRes.	49.4 s	5.6 M	58.3 s	624 K

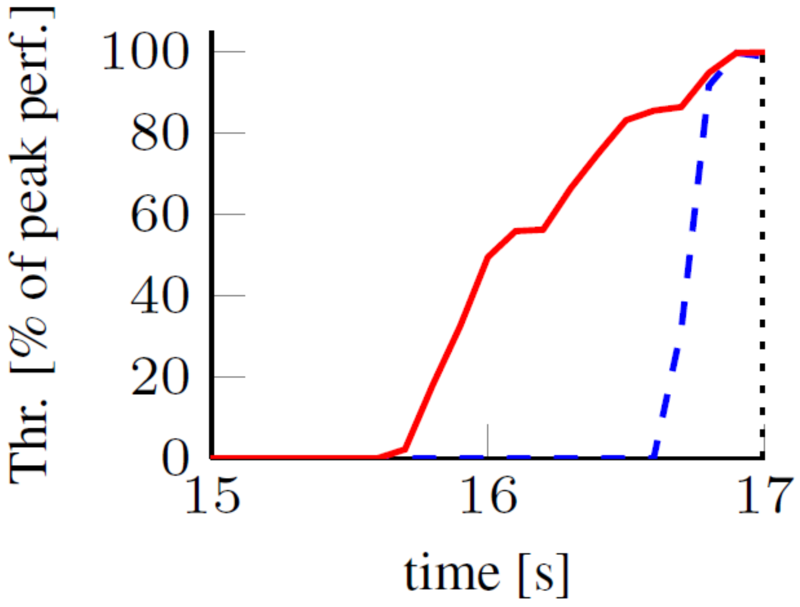
Adaptive recovery adapts well to workload changes

Worst-Case Analysis

Synthetic benchmark

- 10 tables, 10 columns and 1 Mio row each
- All columns uniformly queried

Resources are released as soon as the job queue is empty



Legend Entry	Rec. End	#TXs t=17 s
$\neg rk.Q0.R8$	16.6 s	0.29 M
$rk.Q^{ad}.R8^{ad}$	16.6 s	0.84 M

Synchronous recovery is a lower bound for adaptive recovery

Conclusion

- NVM enables a single-level database storage architecture
- Rebuilding secondary data is the new recovery bottleneck
- Regaining pre-failure throughput near-instantly possible, but at a significant query performance cost
- When query performance is paramount, adaptive recovery allows to swiftly regains pre-failure throughput

Thank you.

Contact information:

Ismail Oukid
ismail.oukid@sap.com

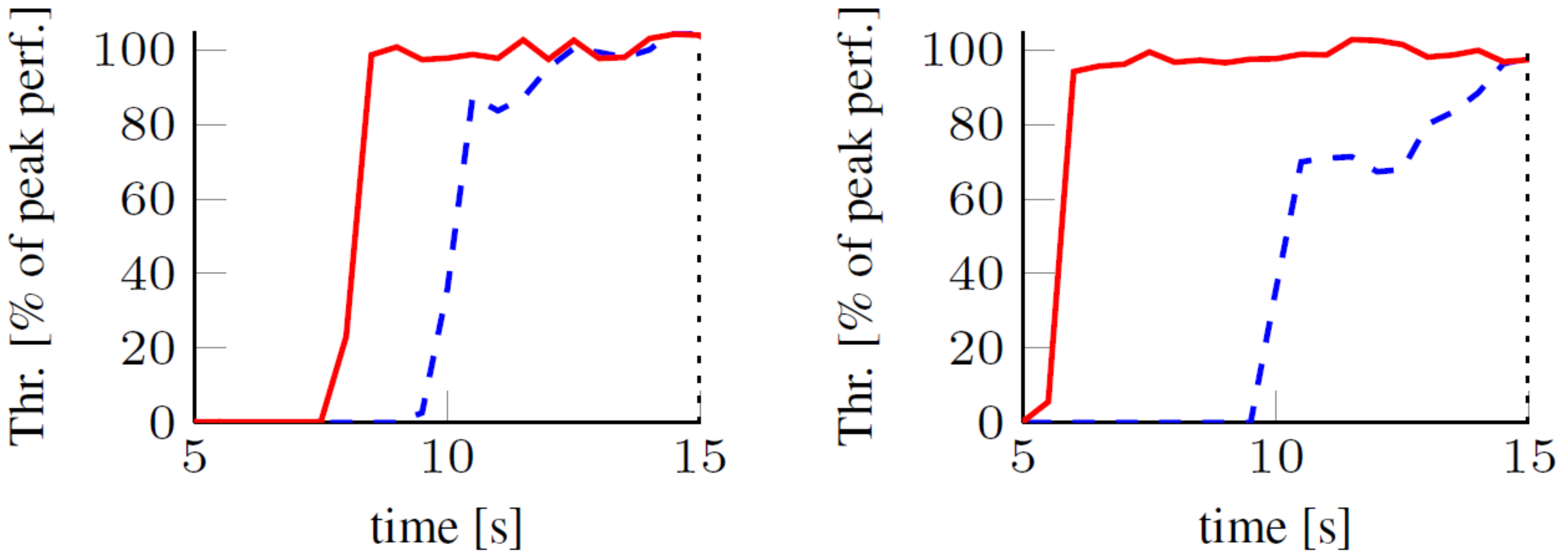
Run Simple

Dresden Database Systems Group

Resource Allocation

Recovery workload same as pre-failure workload

Only a subset of indexes is relevant to the workload



(a) TATP (b) TPCC

Legend Entry	TATP		TPCC	
	Rec. End	#TXs t=15 s	Rec. End	#TXs t=15 s
- - - $rk.Q0.R8$	11.6 s	4.3 M	13.7 s	373 K
—— $rk.Q^{ad}.R8^{ad}$	63.2 s	6.1 M	64.9 s	843 K

Adapting resources significantly improves recovery performance