

Improving In-memory Computing with New Hardware Features

A Perspective from Systems Software

RONG CHEN

http://ipads.se.sjtu.edu.cn/rong_chen

Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University

Joint work with Haibo, Xinda, Jiaxin, Yanzhe, Youyang@IPADS

Business Demand – High Throughput



**GLOBAL SHOPPING
FESTIVAL 2016**

Peak transactions per second:

175,000 new orders

120,000 payment

Business Demand – Low Latency



Evolution and Practice: Low-latency
Distributed Applications in Finance

The finance industry has unique demands for low-latency distributed systems

- Read input message from network and parse – 5 microseconds
- Look up client profile – 3.2 milliseconds (3,200 microseconds)
- Compute client quote – 15 microseconds
- Log quote – 20 microseconds
- Serialize quote to a response message – 5 microseconds
- Write to network – 5 microseconds

How to Do It? Conventional Approach

TPC-C World Record

504,161 TXs/second
Cost: 30,528,863 USD



172,770 TXs/second
Cost: 14,276,808 USD



http://www.tpc.org/tpcc/results/tpcc_results.asp?print=false&orderby=tpm&sortby=desc

How to Do It? Today's Approach

OceanBase

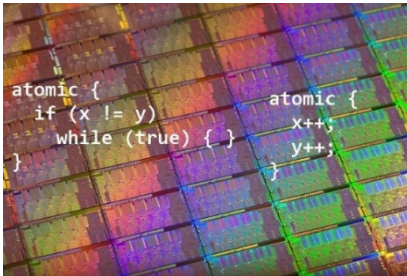
MySQL Cluster



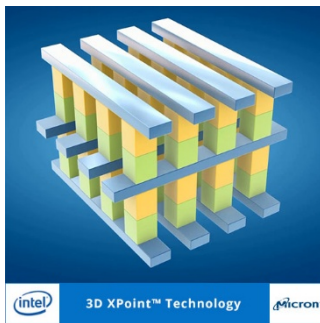
Unit :
ms

Hardware Platform

HTM

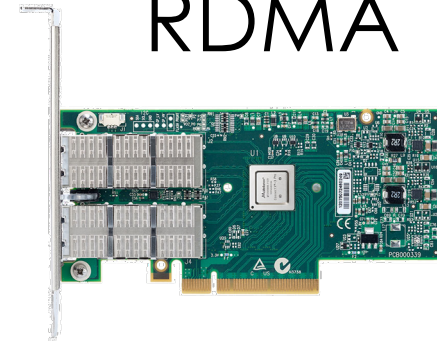


NVM



GPU

RDMA



A few rack-scale machines

Agenda

101 of HTM and RDMA

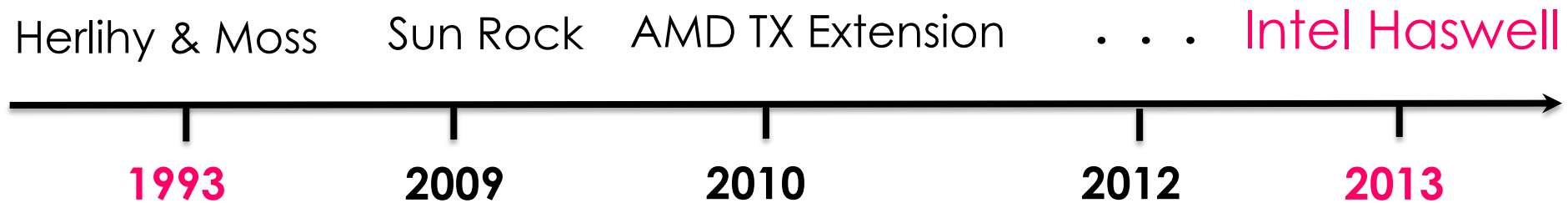
Overall Ideas

RDMA-friendly Distributed Key-value Store

Fast Distributed Transactions using RDMA & HTM

Fast and Concurrency Graph Queries using RDMA

HTM: Hardware Transactional Memory



Massively available

Intel Restricted Transactional Memory

Restricted Transactional Memory (RTM)

- Hardware transactional memory with limitations

Major limitations

- Working set is limited
- System events abort TX

New instruction set

- Xbegin, Xend, Xabort

RTM Usage

```
if _xbegin() == _XBEGIN_STARTED
    do some critical work
    _xend()
else
    fallback routine
```

Handle the
abort event

Deconstructing RTM

RTM prefers read than write

- Asymmetric Read/Write Limits
 - L1-Cache tracks writes
 - An implementation specific structure tracks reads

RTM prefers read before write

- Only eviction of cache lines in write set will abort TX

Transaction exec time affects TX abort

- Timer interrupt unconditionally abort a TX (4ms on 250hz kernel)

RDMA: Remote Direct Memory Access

A network feature that allows **direct** access to the memory of a **remote** computer

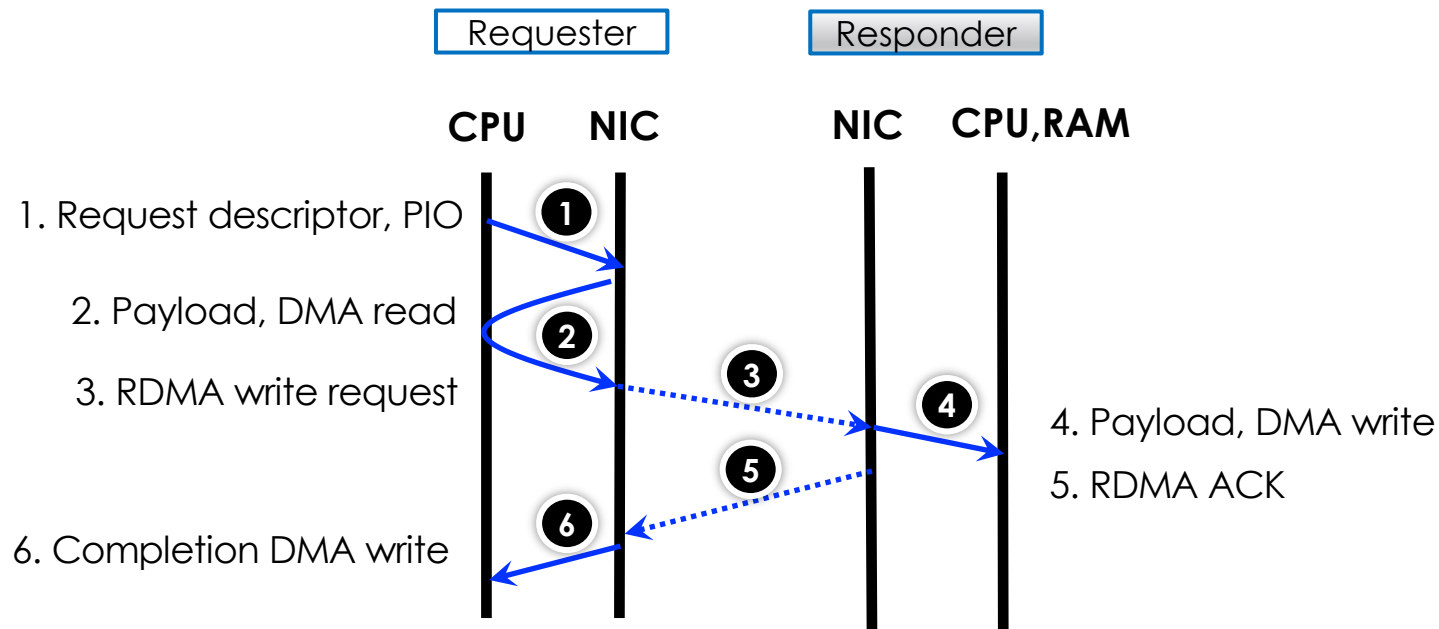
High speed, low latency & low CPU overhead

- ▶ Interface: IPoB, two-sided RDMA (SEND/RECV), and **one-sided RDMA** (READ/WRITE/CAS)
- ▶ Bypasses **OS kernels**: Zero copy
Bypasses **CPU** (one-sided)
- ▶ Round-trip time: **one-sided**/ $\sim 3\mu\text{s}$,
two-sided/ $\sim 7\mu\text{s}$, IPoB/ $\sim 100\mu\text{s}$

One-sided RDMA Primitives

RDMA read, write and CAS

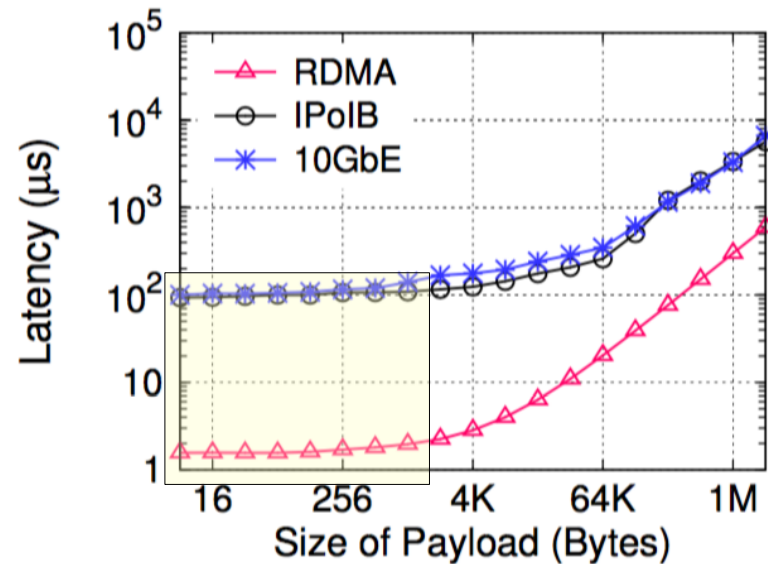
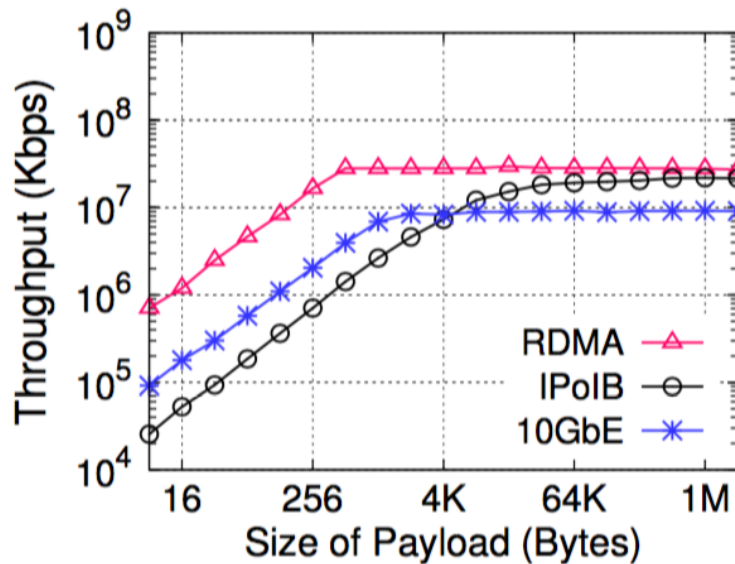
Life-cycle of an RDMA write



Credit: Anuj Kalia's SIGCOMM talk

One-sided RDMA Performance

Performance of Random Read¹



Insensitive to payload size:

High/near constant throughput/Low latency when payload is smaller than a threshold

¹ Mellanox ConnectX-3 MCX353A 56Gbps InfiniBand NIC

Overall Idea
Hardware Feature Combination

Opportunities: (not so) New HW Features

HTM: Hardware **T**ransaction **M**emory

- Allow a group of load & store instructions to execute in an **atomic**, **consistent** and **isolated** (ACI) way

RDMA: Remote **D**irect **M**emory **A**ccess

- Provide **cross-machine** accesses with high speed, low latency and low CPU overhead

Rethink the design of **low-COST**
scalable in-memory transaction systems

Opportunities with HTM & RDMA

HTM: Hardware Transaction Memory

a *non-transactional* code will unconditionally abort
a transaction when their accesses conflict

**Strong
Atomicity**

RDMA: Remote Direct Memory Access

Opportunities with HTM & RDMA

HTM: Hardware **T**ransaction **M**emory

*a **non-transactional** code will unconditionally abort a transaction when their accesses conflict*

**Strong
Atomicity**

RDMA: Remote **D**irect **M**emory **A**ccess

*one-sided RDMA operations are **cache-coherent** with local accesses*

**Strong
Consistency**

Opportunities with HTM & RDMA

HTM: Hardware **T**ransaction **M**emory

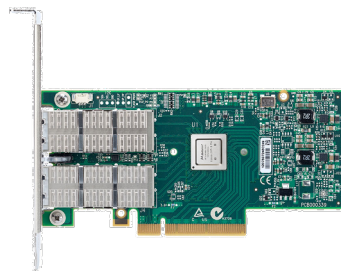
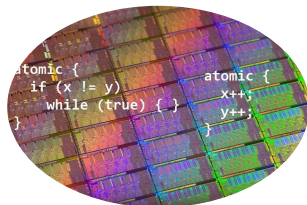
a *non-transactional* code will unconditionally abort a transaction when their accesses conflict

RDMA: Remote **D**irect **M**emory **A**ccess

one-sided RDMA operations are *cache-coherent* with local accesses

HTM Strong Atomicity + **RDMA Strong Consistency** $\Rightarrow$ **RDMA ops will abort conflicting HTM TX**

RDMA-friendly Distributed Key-value Store



In-memory KV-Stores

Interface: **GET**, **PUT**, etc.

A key pillar for many systems

- ▶ Data cache (e.g., Memcached in FB)
- ▶ In-memory database (e.g., OLTP, OLAP)
- ▶ Graph query processing

Key requirements

- ▶ Low latency
- ▶ High request rate (Throughput)

Challenges & Goals

How to maximize **RDMA** potentials?

Use **one-sided RDMA** READs and WRITEs

Little or no host **CPU involvement**

Simple/Fast KV accesses (no complex encoding or versioning)

- ▶ Otherwise, host CPU has to do complex processing during accesses

Symmetric accesses

- ▶ No trading off client resource for server efficiency

Prior systems

Prior systems (e.g. Pilaf^{ATC'13} and FaRM^{NSDI'14})

- Only leverage one-sided RDMA to read, **not write**
- Complicated INSERT: hard to leverage **HTM**
- **No RDMA-friendly caching** mechanism

Content-based caching (e.g. replication) is hard to perform strong-consistent read and write locally, especially using RDMA

	Pilaf	FaRM
Hashing	Cuckoo	Hopscotch
Race Detection	Checksum	Versioning
Remote Read	One-sided RDMA	
Remote Write	Messaging	
Caching	No	

RDMA & HTM provides a new design space

Prior Systems

	Pilaf		FaRM	HERD	DrTM
Model	Asymmetric	Symmetric	Asymmetric	Symmetric	
Hashing	Cuckoo	Hopscotch	Chaining	Chaining	
Remote Read	One-sided RDMA		Messaging	One-sided RMDA	
Remote Write	Messaging				
Race Detection	Checksum	Versioning	Exclusive Accesses	L: HTM D: Lock	
Transaction	No	Yes	No	Yes	
Caching	No	No	No	Yes	

1. A bias towards RDMA-based **remote** operations
2. Only leverage one-sided RDMA to **read**
3. No efficient RDMA-friendly **caching** scheme
4. Complicated INSERT: hard to leverage **HTM**

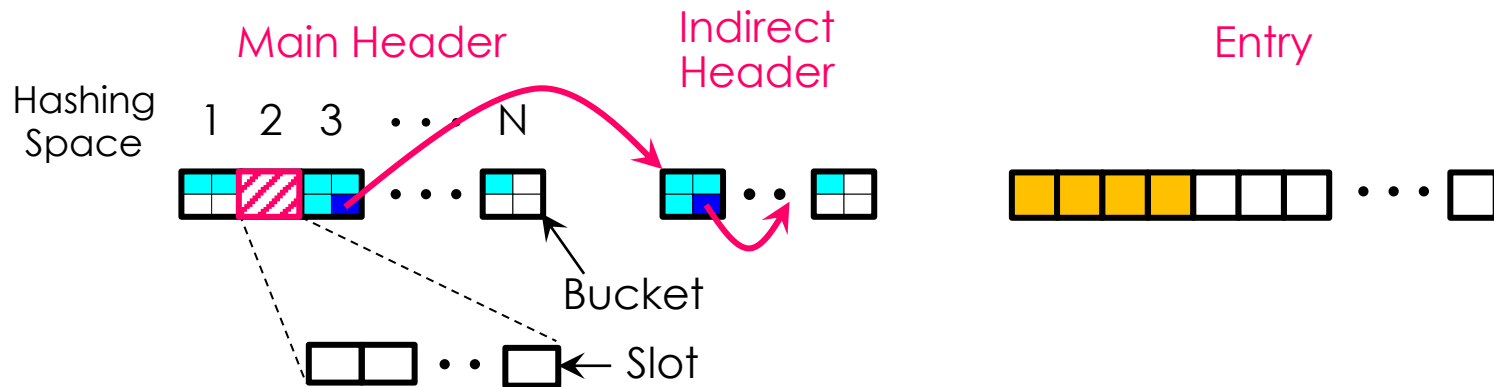
DrTM-KV's Design

- ✓ Simple hash structure to fully leverage **HTM**
- ✓ **Decouple** race detection from memory store
 - Use **HTM** to detect races
 - Use **one-sided** RDMA ops for remote read & **write**
- ✓ **Location-based** and fully transparent cache

	Pilaf		FaRM	DrTM-kV	
Hashing	Cuckoo	Hopscotch		Chaining	→ Simple
Race Detection	Checksum	Versioning		HTM	
Remote Read	One-sided RDMA			One-sided RMDA	→ Efficient
Remote Write	Messaging				
Caching	No			Yes	

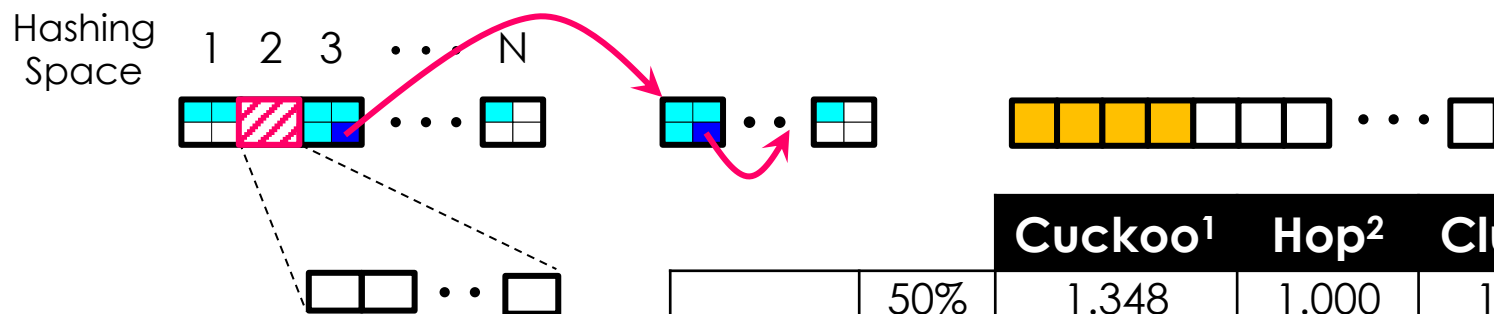
RDMA-friendly Cluster Chaining

- ✓ Similar to traditional chaining HT with associativity
- ✓ Decoupled memory region: index & data
- ✓ Shared indirect headers: high space efficiency



RDMA-friendly Cluster Chaining

- ✓ Similar to traditional **chaining** HT with **associativity**
- ✓ Decoupled memory region: **index** & **data**
- ✓ **Shared** indirect headers: high **space** efficiency



		Cuckoo ¹	Hop ²	Cluster ³
Uniform	50%	1.348	1.000	1.008
	75%	1.652	1.011	1.052
	90%	1.956	1.044	1.100
Zipf $\theta=0.99$	50%	1.304	1.000	1.004
	75%	1.712	1.020	1.039
	90%	1.924	1.040	1.091

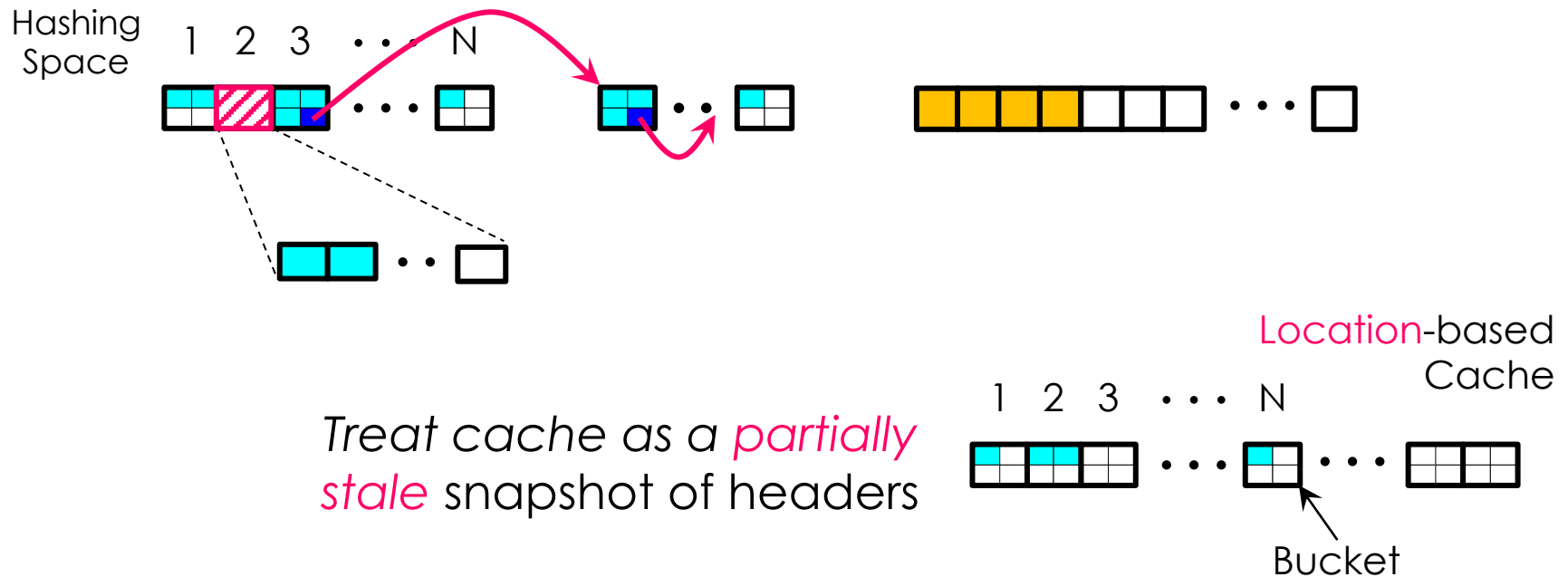
¹ Hopscotch hashing in FaRM configures the neighborhood with 8 ($H=8$).

² Cuckoo hashing in Pilaf uses 3 orthogonal hash functions and each bucket contains 1 slot.

³ Cluster hashing in DrTM configures the associativity with 8.

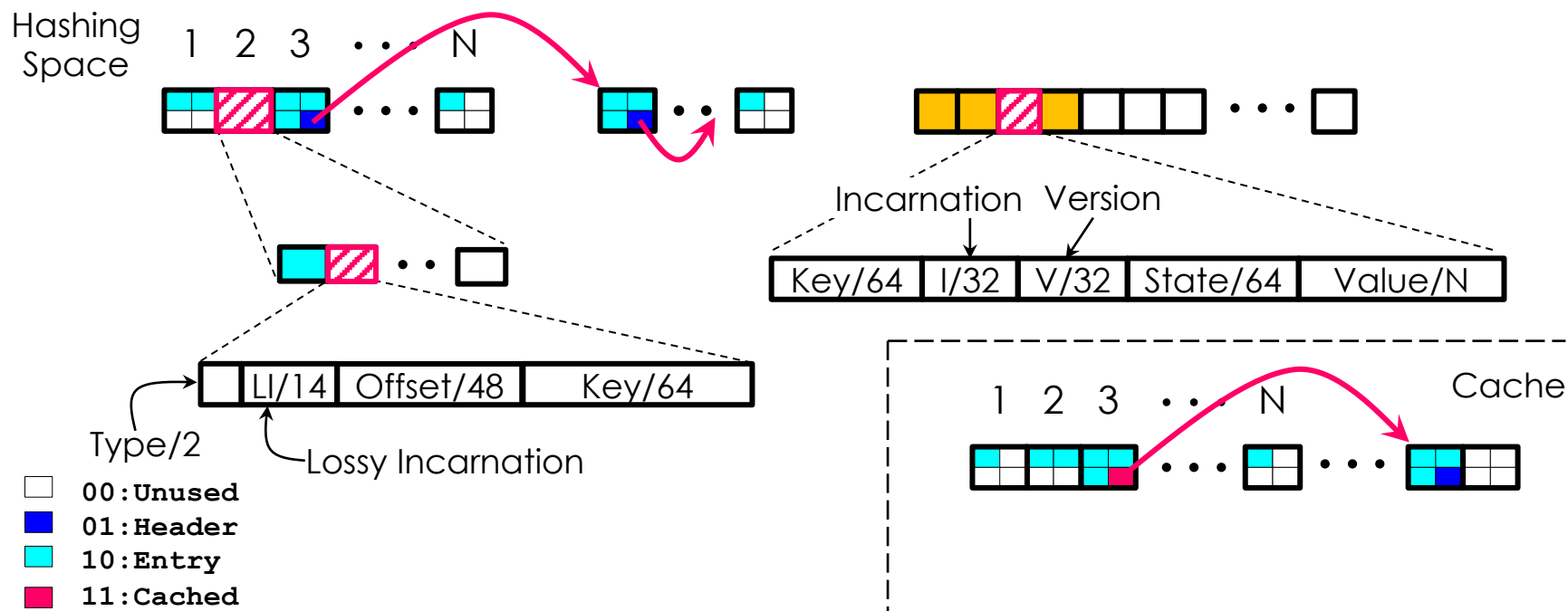
Location-based Caching

✓ **RDMA-friendly**: focus on minimizing the lookup cost



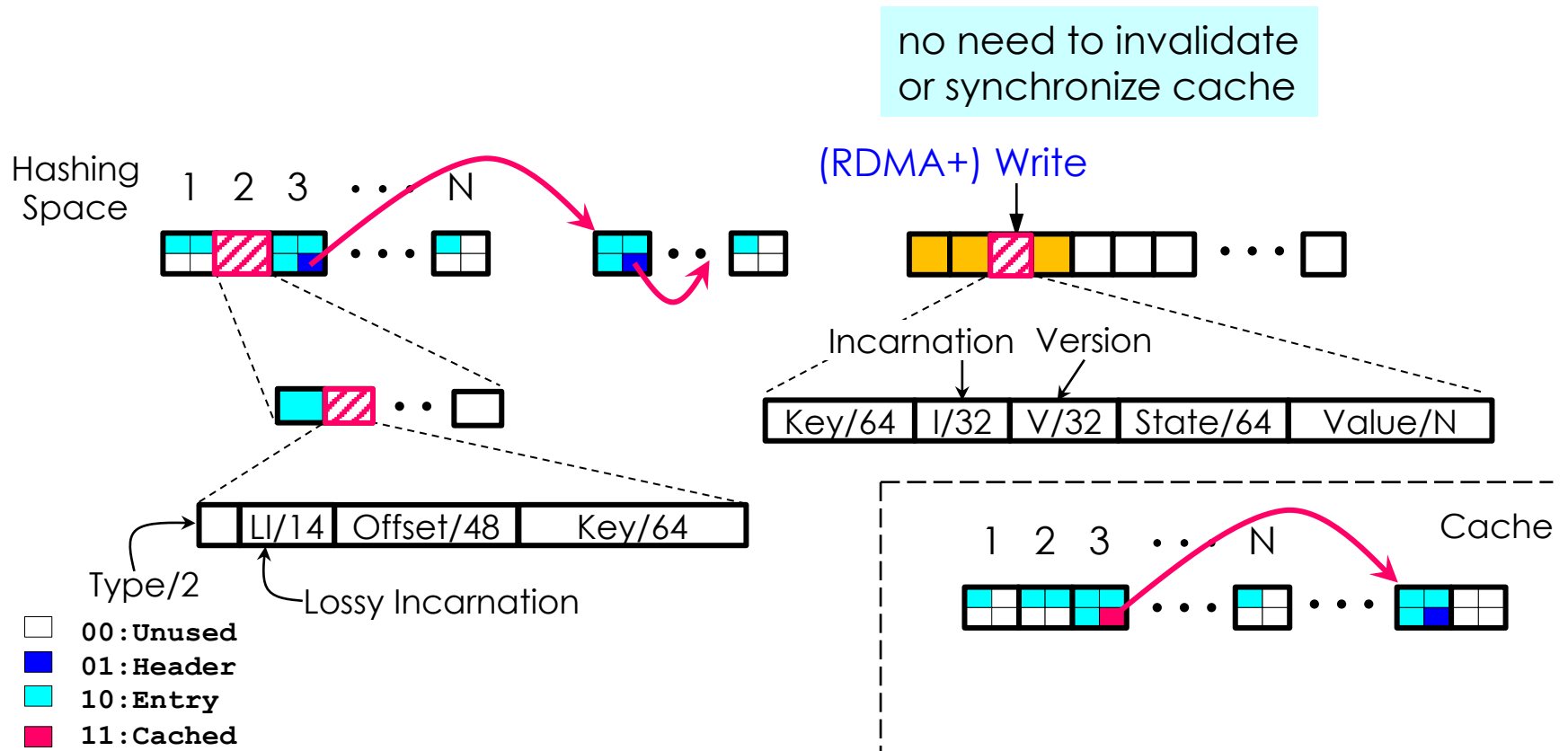
Location-based Caching

- ✓ **RDMA-friendly**: focus on minimizing the lookup cost
- ✓ Retain the full **transparency** to the host
 - All metadata used by other concurrency control mechanisms is encoded in the key-value **entry**



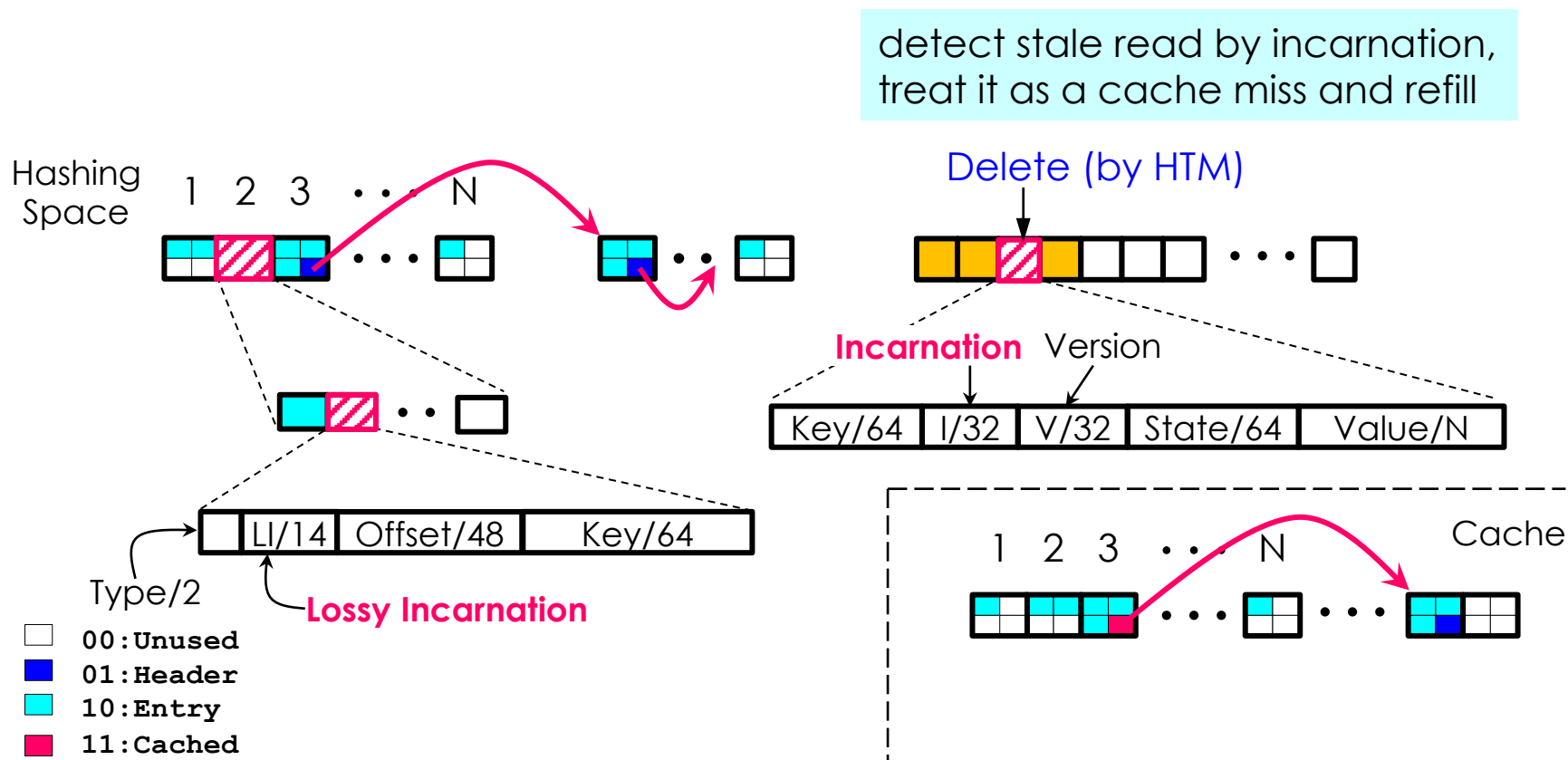
100

- ✓ **RDMA-friendly**: focus on minimizing the lookup cost
- ✓ Retain the full **transparency** to the host



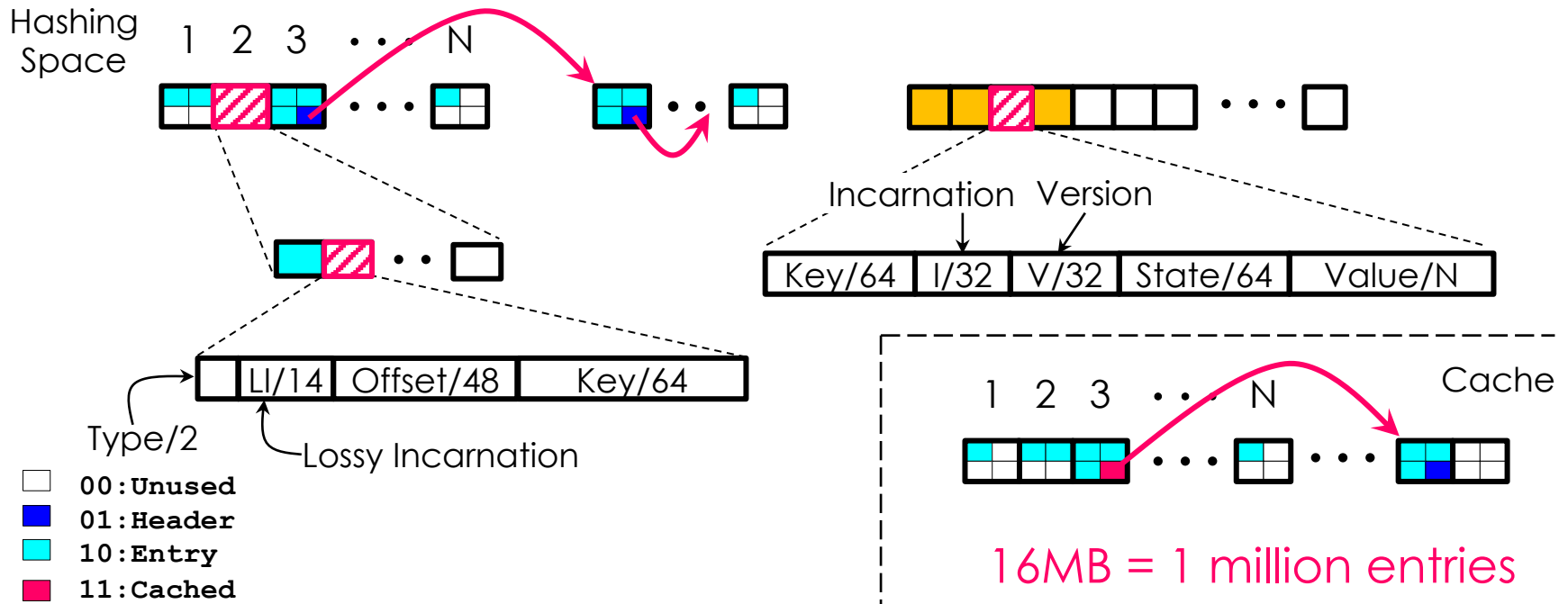
Location-based Caching

- ✓ **RDMA-friendly**: focus on minimizing the lookup cost
- ✓ Retain the full **transparency** to the host



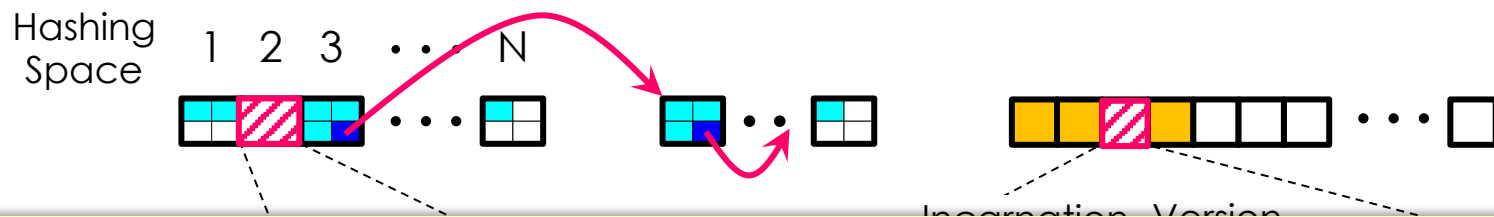
Location-based Caching

- ✓ **RDMA-friendly**: focus on minimizing the lookup cost
- ✓ Retain the full **transparency** to the host
- ✓ The size of cache for location is **small**



Location-based Caching

- ✓ **RDMA-friendly**: focus on minimizing the lookup cost
- ✓ Retain the full **transparency** to the host
- ✓ The size of cache for location is **small**
- ✓ All client threads can directly **share** the cache



The average lookup cost = **0.178**

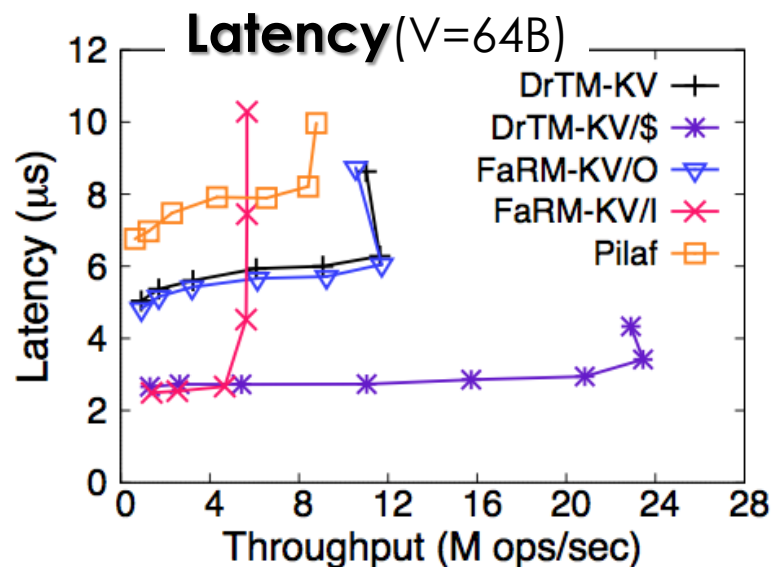
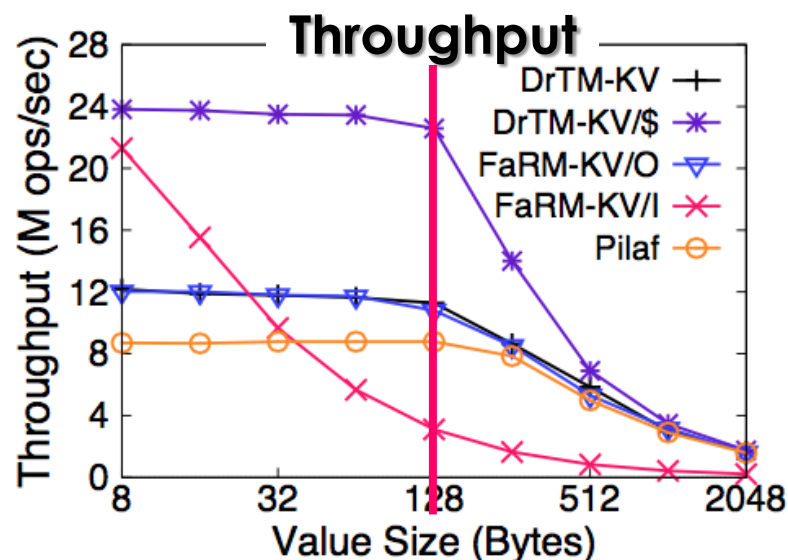
20 million kv-pairs (>40 GB), **20MB** cache (from **empty**),
8 client threads, skewed workload (**Zipf** $\theta=0.99$)

- 00: Unused
- 01: Header
- 10: Entry
- 11: Cached

Performance of DrTM-KV

- ✓ DrTM-KV w/o caching provides a comparable performance
- ✓ DrTM-KV w/ caching (DrTM-KV/\$) can achieve both lowest latency ($3.4\ \mu\text{s}$) and highest throughput ($23.4\ \text{Mops/sec}$)

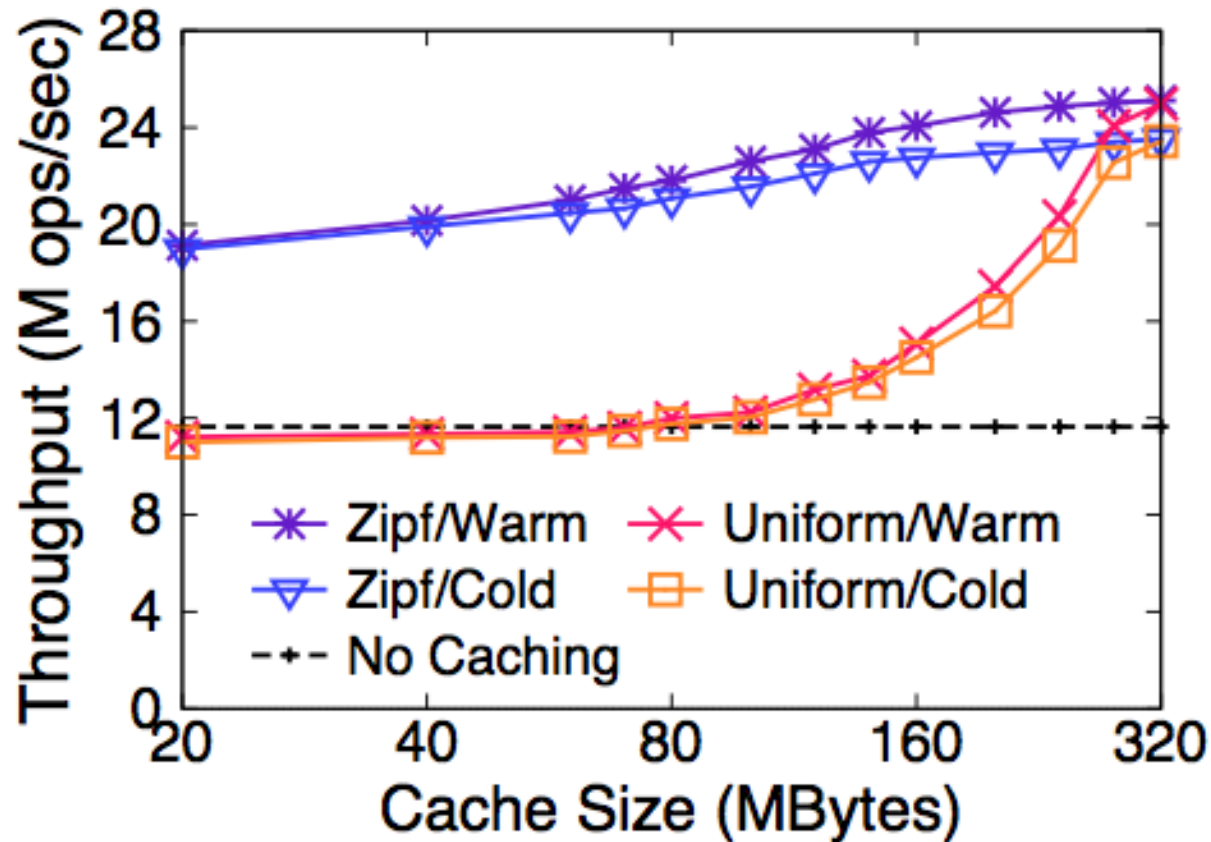
← FaRM: 2.1X, Pilaf: 2.7X



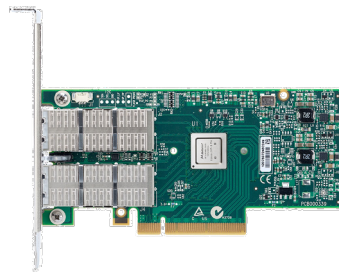
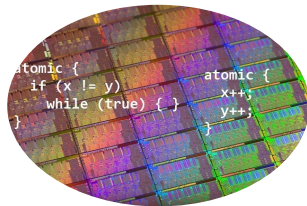
Setting: 1 server and 5 clients (up to 8 threads), 20 million k/v pairs
peak throughput of random RDMA READ $\approx 26\ \text{Mops/sec}$

Performance of Caching

Caching ($V=64B$)



Fast Distributed Transactions using RDMA & HTM



High **COST** for Distributed TX

Many scalable systems have **low performance**

- Usually 10s~100s of thousands of TX/second
- High **COST**¹ (config. that outperform single thread)
- e.g., HStore, Calvin^{SIGMOD'12}

Emerging speedy TX systems **not scale-out**

- Achieve over 100s of thousands TX/second
- e.g., Silo^{SOSP'13}, DBX^{EuroSys'14}

Dilemma:

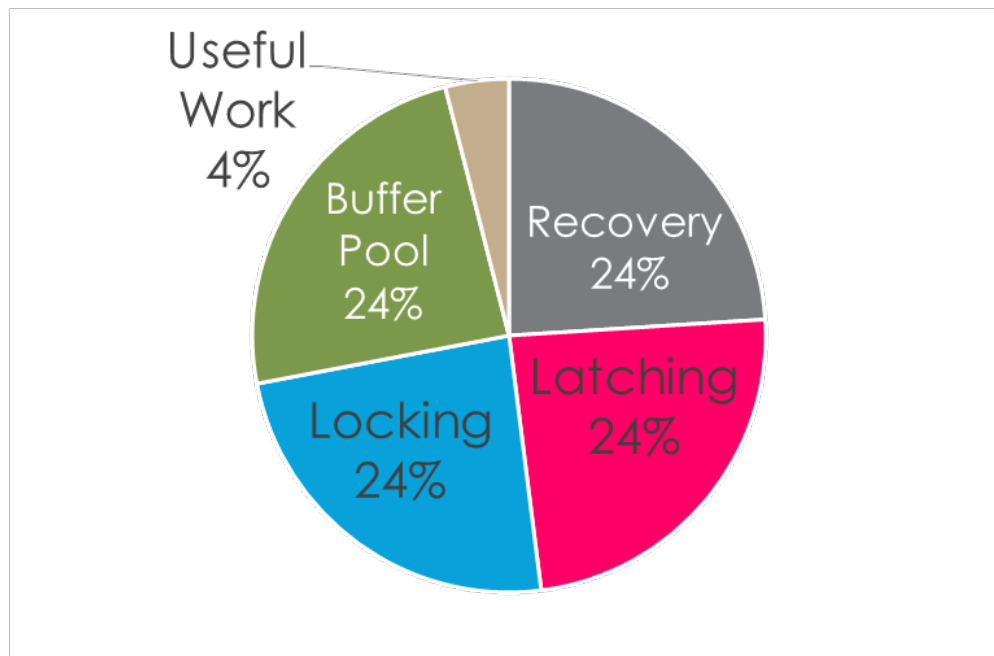
single-node perf. vs. scale-out

¹ Scalability! But at what Cost? HotOS 2015

Why (Distributed) TXs are Slow?

Only **4% of wall-clock time** spent on useful data processing, while the rest is occupied with **buffer pools, locking, latching, recovery**.¹

-- Michael Stonebraker



¹ "The Traditional RDBMS Wisdom is All Wrong"

Opportunities with HTM & RDMA

HTM: Hardware **T**ransaction **M**emory

a *non-transactional* code will unconditionally abort a transaction when their accesses conflict

RDMA: Remote **D**irect **M**emory **A**ccess

one-sided RDMA operations are *cache-coherent* with local accesses

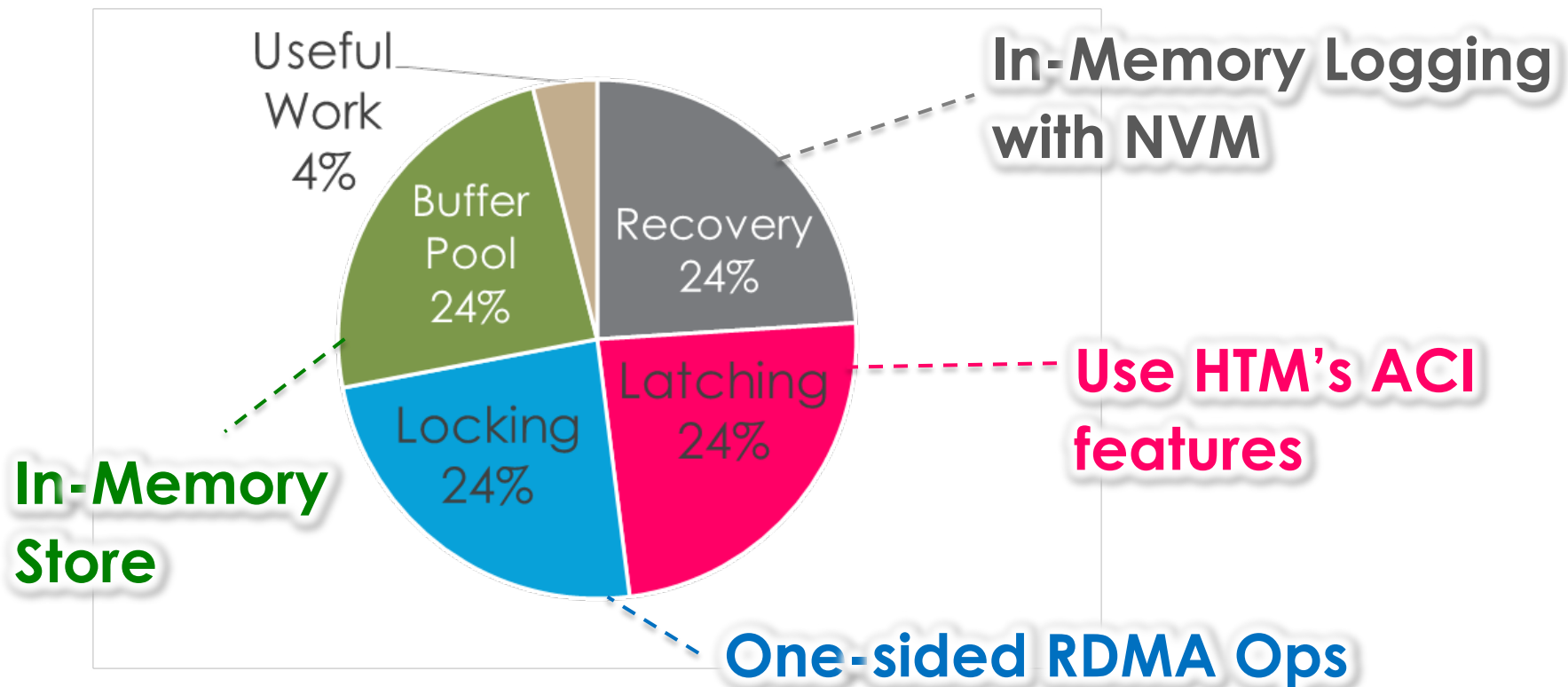
HTM Strong Atomicity + **RDMA Strong Consistency** $\Rightarrow$ **RDMA ops will abort conflicting HTM TX**

Basis for Distributed TM

DrTM: Fast In-memory Transaction Processing using RDMA and HTM

Use **HTM**'s ACI properties for local TX execution

Use one-sided **RDMA** to glue multiple HTM TXs



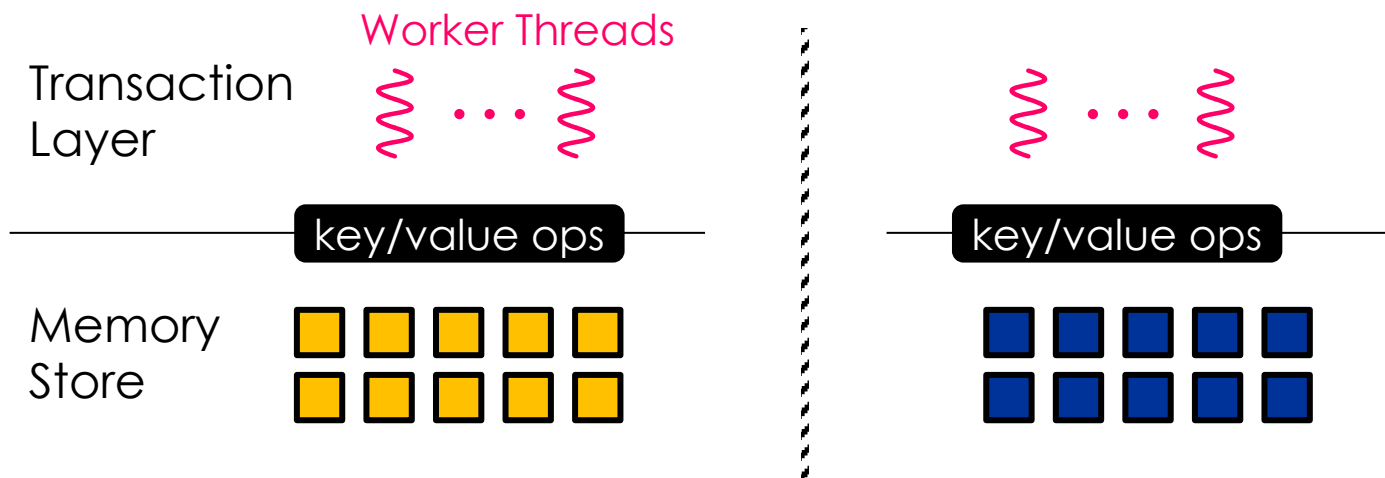
System Overview

DrTM: Distributed TX with HTM & RDMA

- Target: OLTP workloads over large volume of data
- Two independent components using HTM&RDMA

Transaction layer & memory store

- Low COST distributed TX
 - Achieve over 5.52 million TXs/sec for TPC-C on 6 nodes



Challenge#1: Restriction of HTM

HTM is only a compelling hardware feature for **single machine** platform

- **Distributed** TX cannot directly benefit from it

Some instructions & system events (e.g. network I/O) will **unconditionally** abort **HTM** transactions

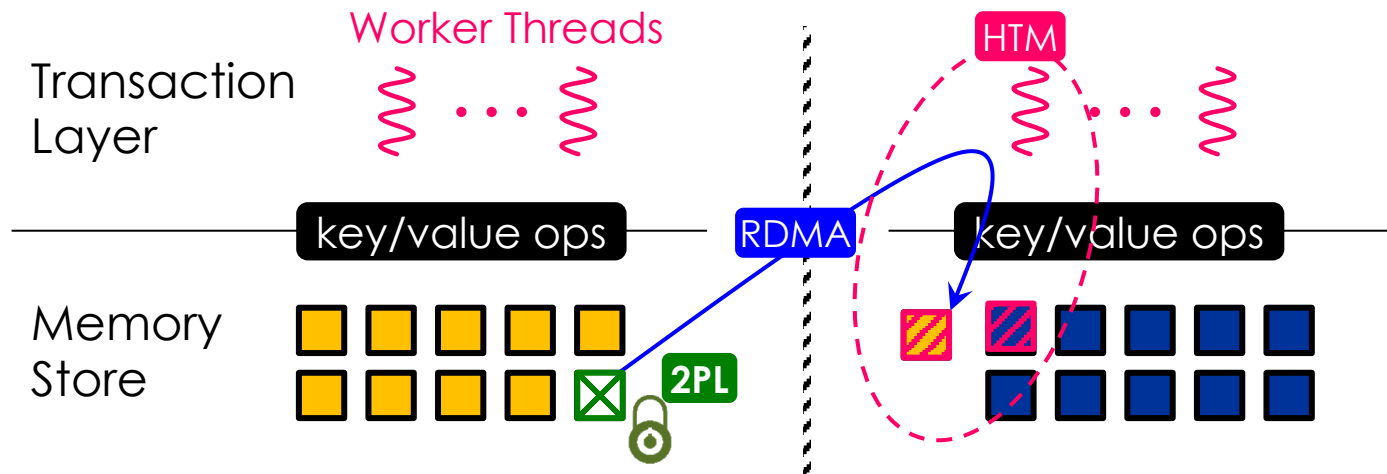
- Like any **RDMA** ops: READ/WRITE, CAS, SEND/RECV

How to glue multiple **HTM** transactions together by **RDMA** while preserving serializability?

Combining HTM with 2PL

Using **2PL** to accumulate all remote records **prior to** accesses in an HTM transaction

- Transform a distributed TX to a local one
- Limitation: require advanced knowledge of read/write sets of transactions¹



¹ This is similar with prior work (e.g. Sinfonia & Calvin) and the case for typical OLTP workloads

DrTM's Concurrency Control

Local TX vs. Local TX: **HTM**

Distributed TX vs. Distributed TX: **2PL**

Local TX vs. Distributed TX: **abort local TX**

DrTM's Concurrency Control

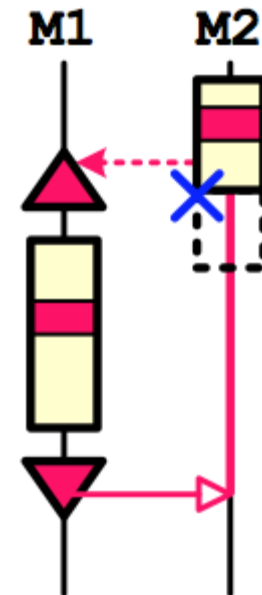
Local TX vs. Local TX: **HTM**

Distributed TX vs. Distributed TX: **2PL**

Local TX vs. Distributed TX: **abort local TX**

Local TX prior to
Distributed TX

→ **RDMA** (strong consistency)
+ **HTM** (strong atomicity)



DrTM's Concurrency Control

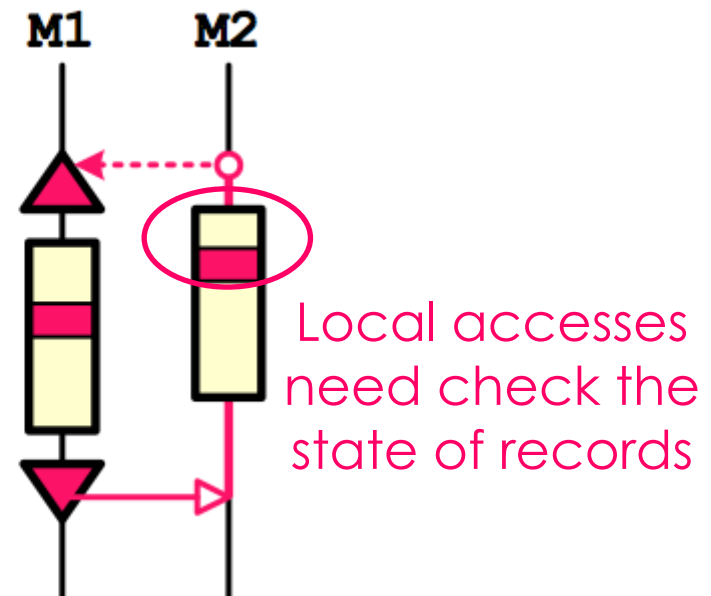
Local TX vs. Local TX: **HTM**

Distributed TX vs. Distributed TX: **2PL**

Local TX vs. Distributed TX: **abort local TX**

Distributed TX
prior to Local TX

→ Local TX **checks** (but not
locks) the records



Challenge#2: Limit of RDMA Semantics

RDMA provides three communication options

- IPoIB, SEND/RECV and one-sided RDMA ops

Good performance (e.g. latency)
and without involving the host CPU

One-sided RDMA has much limited interfaces

- READ, WRITE, CAS and XADD

How to support **exclusive** and **shared** accesses
in 2PL protocol using one-sided RDMA ops

DrTM's Lock

RDMA CAS: atomic compare-and-swap

- Similar to the semantic of normal CAS (i.e. local CAS)

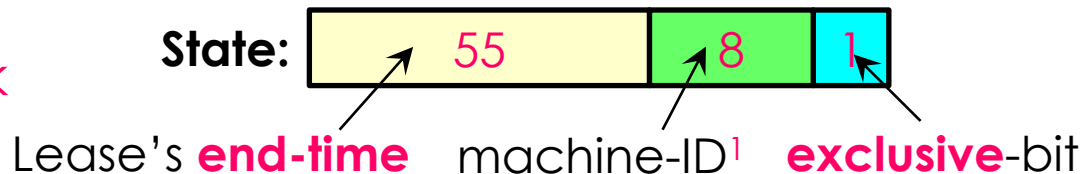
1. DrTM's **exclusive** lock
 - Spinlock: use RDMA CAS to **acquire** & **release**
2. DrTM's **shared** lock
 - **Lease-based** protocol

Shared (Read) Lock

Lease-based protocol

- Grant **read right** to the lock holder in a **time period**
- No need to **explicit** release or invalidate the lock

exclusive &
shared lock



$000\dots000_2$ **unlocked**
 $000\dots yy1_2$ **exclusive locked**
 $xxx\dots000_2$ **shared locked**

State is atomically compare and swap using RDMA CAS

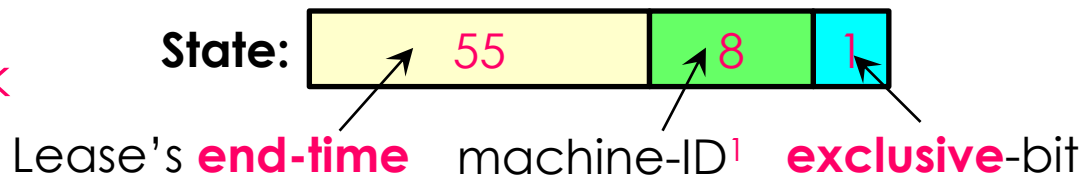
¹ Machine ID is only used by recovery

Shared (Read) Lock

Lease-based protocol

- Grant **read right** to the lock holder in a **time period**
- No need to **explicit** release or invalidate the lock
- Synchronized time is provided by PTP²

exclusive &
shared lock



000...000₂ **unlocked**
000...yy1₂ **exclusive locked**
xxx...000₂ **shared locked**

DELTA is used to tolerate the time bias among machines

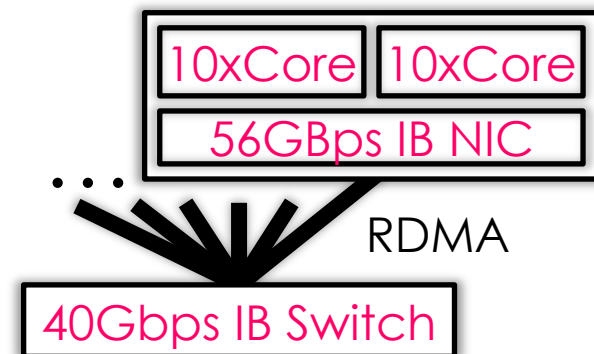
→ **EXPIRED:** if **now** > **end-time** + **DELTA**
INVALID: if **now** < **end-time** - **DELTA**

¹ Machine ID is only used by recovery

² PTP: precision time protocol, <http://sourceforge.net/p/ptpd/wiki/Home/>

Evaluation

Baseline: Latest *Calvin* (Mar. 2015)



Platforms: A small-scale 6-machine cluster

- Each: two 10-cores, *RTM-enabled* Intel Xeon E5-2650 (disabled HT), 64GB DRAM, Mellanox ConnectX-3 MCX353A 56Gbps InfiniBand NIC w/ *RDMA*¹

Benchmark²

- TPC-C

TPC-C	NEW	PAY	DLY	OS	SL
Ratio	45%	43%	4%	4%	4%
Type	d+rw	d+rw	l+rw	l+ro	l+ro

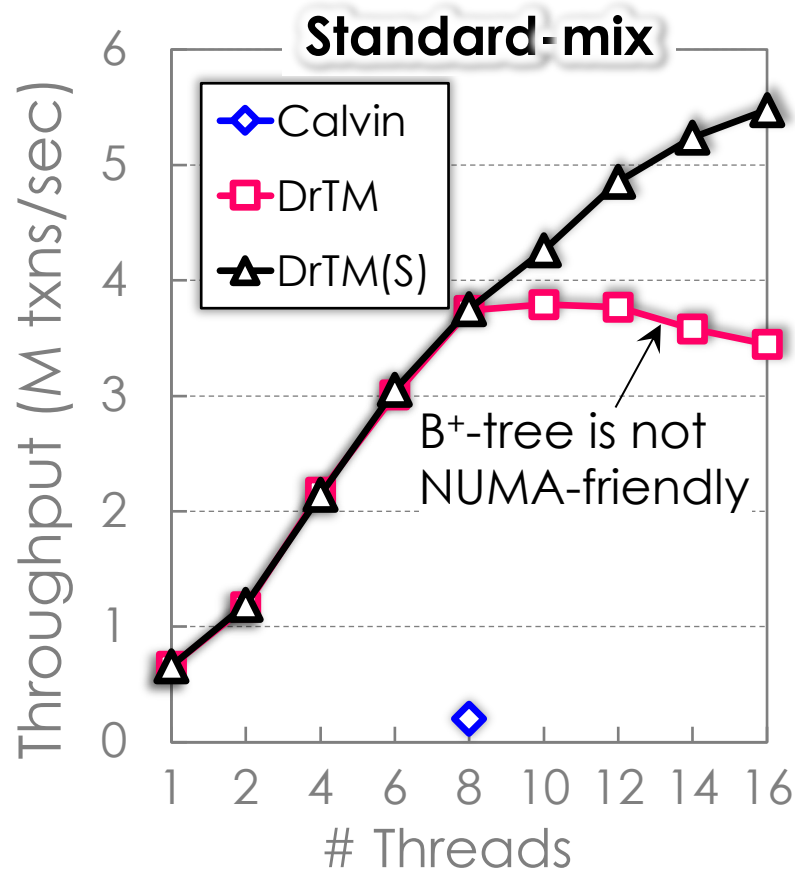
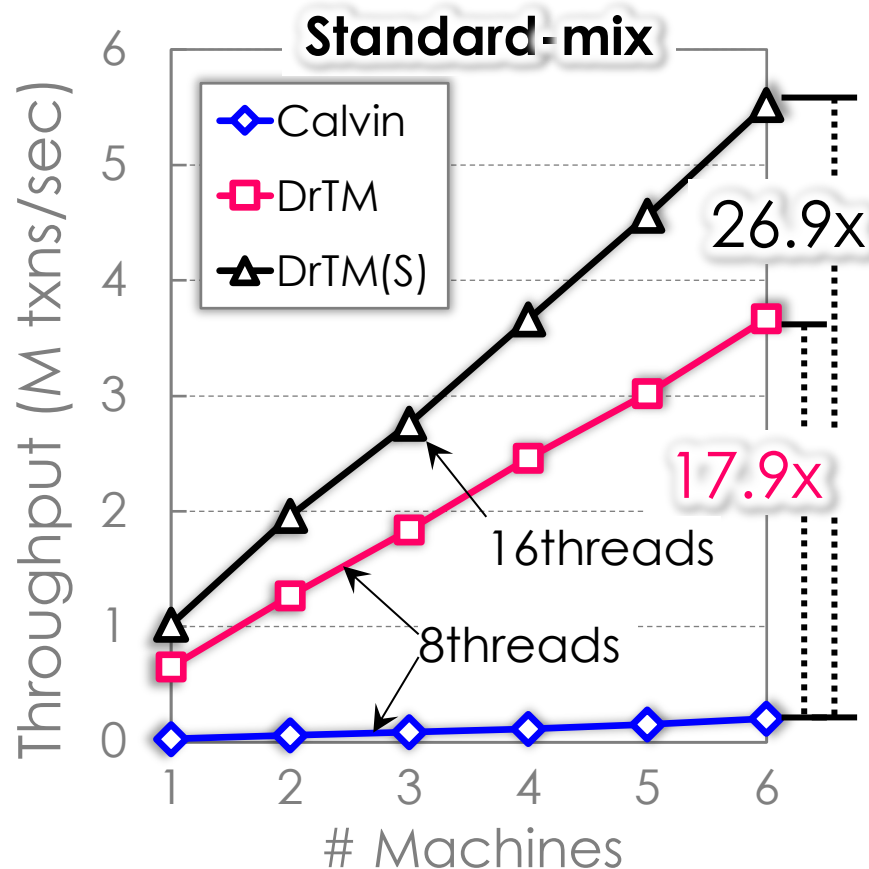
¹ All machines run Ubuntu 14.04 with Mellanox OFED v3.0-2.0.1 stack.

² **d** and **l** stand for distributed and local. **rw** and **ro** stand for read-write and read-only.

Performance on TPC-C

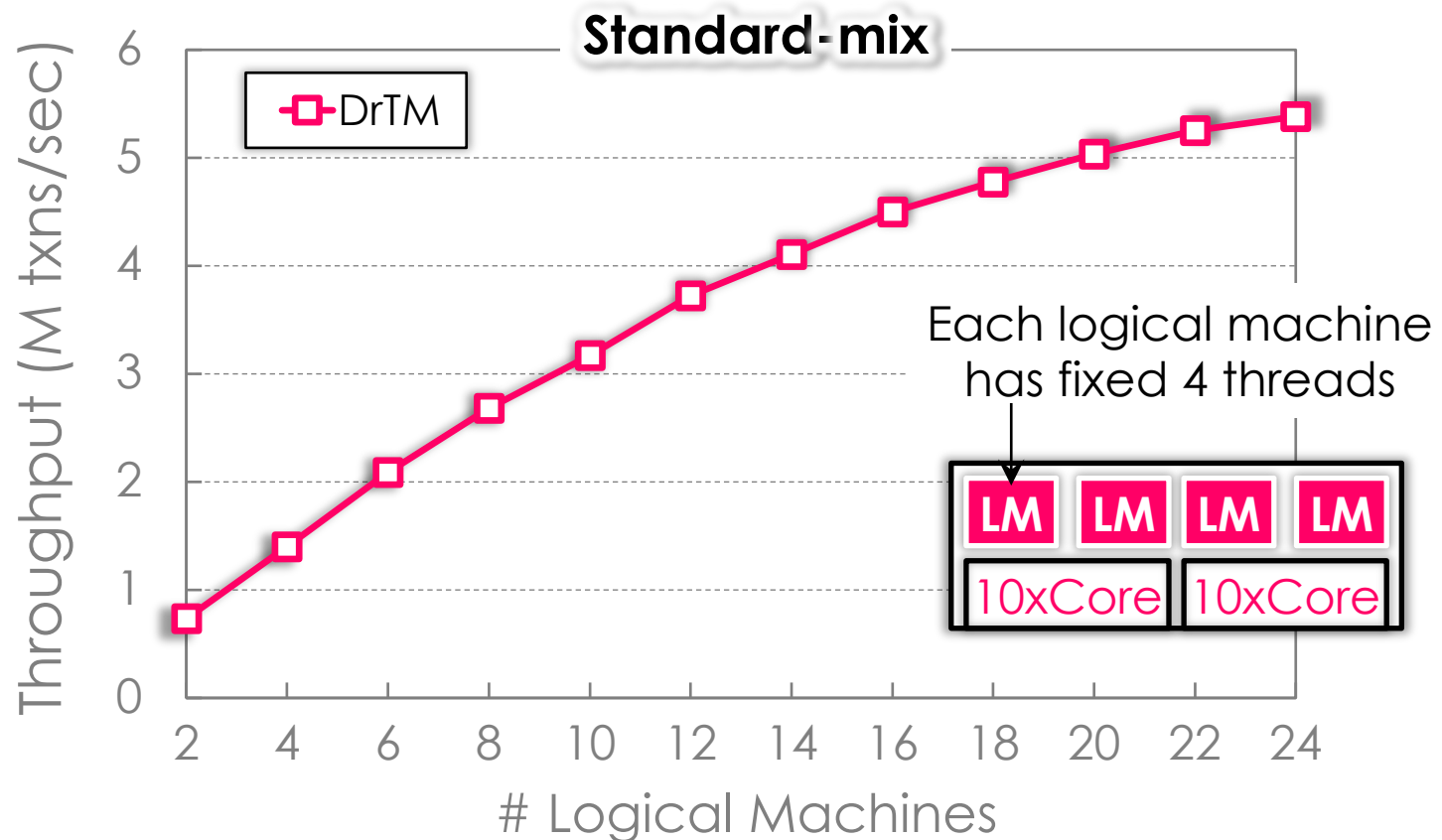
New-order TX
≈ Standard-mix x45%

DrTM(S): run a separate **logical** node on each socket



Scalability on TPC-C

New-order TX
≈ Standard-mix **x45%**



NOTE: the **interaction** btw. two logical nodes sharing the same machine still uses our **RDMA-friendly 2PL protocol**

Summary

In-memory computing demands
high throughput and **low latency**

RDMA: helps bridge the gap from incommensurate scaling for in-memory computing

- ▶ Distributed key/value store
- ▶ In-memory transactions

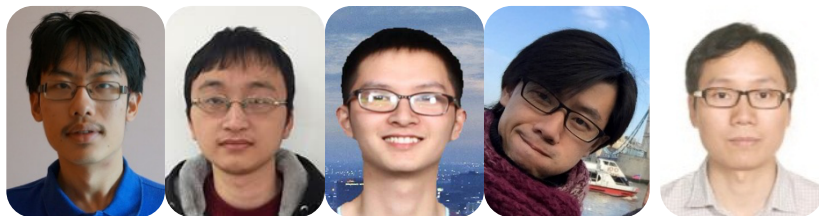
Achieving **orders-of-magnitude** lower **latency**
& higher **throughput** than prior state-of-the-art
centralized and distributed systems

Thanks



<http://ipads.se.sjtu.edu.cn/drtm>

Institute of Parallel and
Distributed Systems



Questions

