RDMA-enable Transaction Processing

XINDA WEI, JIAXIN SHI, YANZHE CHEN,
RONG CHEN, HAIBO CHEN

Institute of Parallel and Distributed Systems
Shanghai Jiao Tong University, China

Transaction: Key Pillar for Many Systems

$9.3 billion/day

Demand Speedy Distributed Transaction Over Large Data Volumes

9.56 million tickets/day

PayPal™

11.6 million payments/day

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:

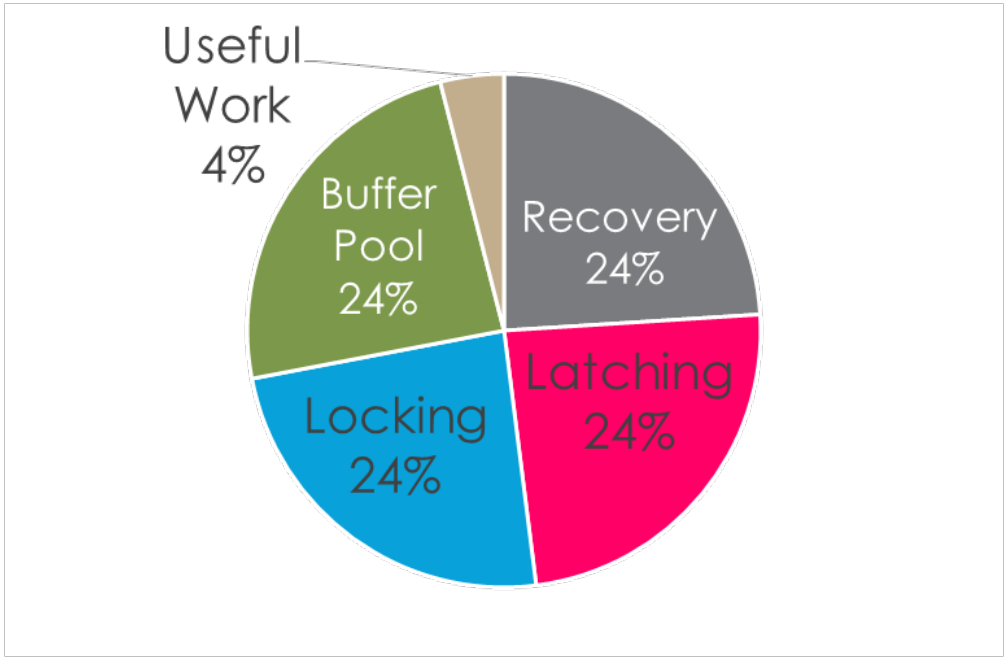
single-node perf. *vs.* **scale-out**

[1] Salability! 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]

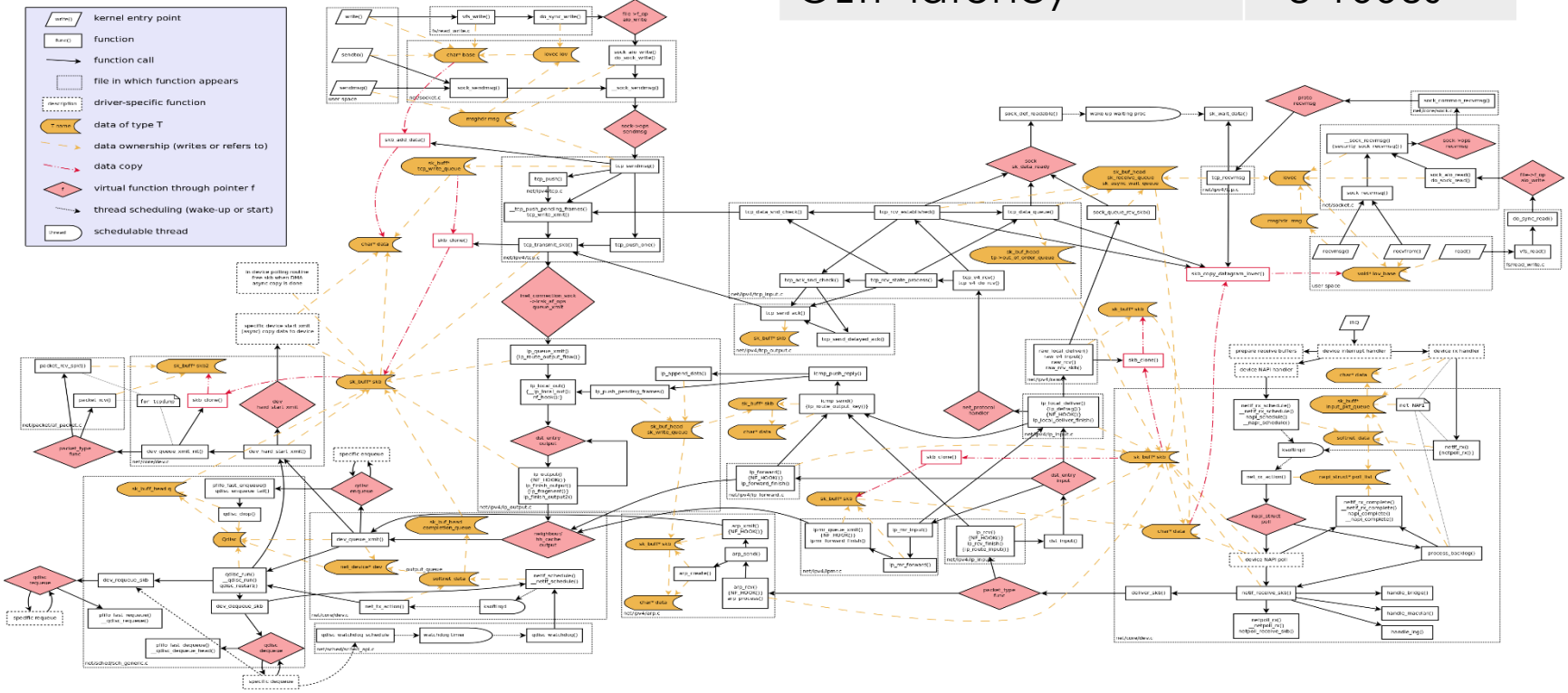
-- Michael *Stonebraker*



[1] "The Traditional RDBMS Wisdom is All Wrong"

Networking costs

Physical limitx	~1-10us
Linux/TCP/IP Stack	~50-200us
OLTP latency	~8-100us



http://www.linuxfoundation.org/images/1/1c/Network_data_flow_through_kernel.png

RDMA: a "Not So New" Feature

Widely used in HPC area for decades

Recently seen increasing adoption in server computing

- Lower price due to massive production and technique advances

RDMA: Remote Direct Memory Access

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

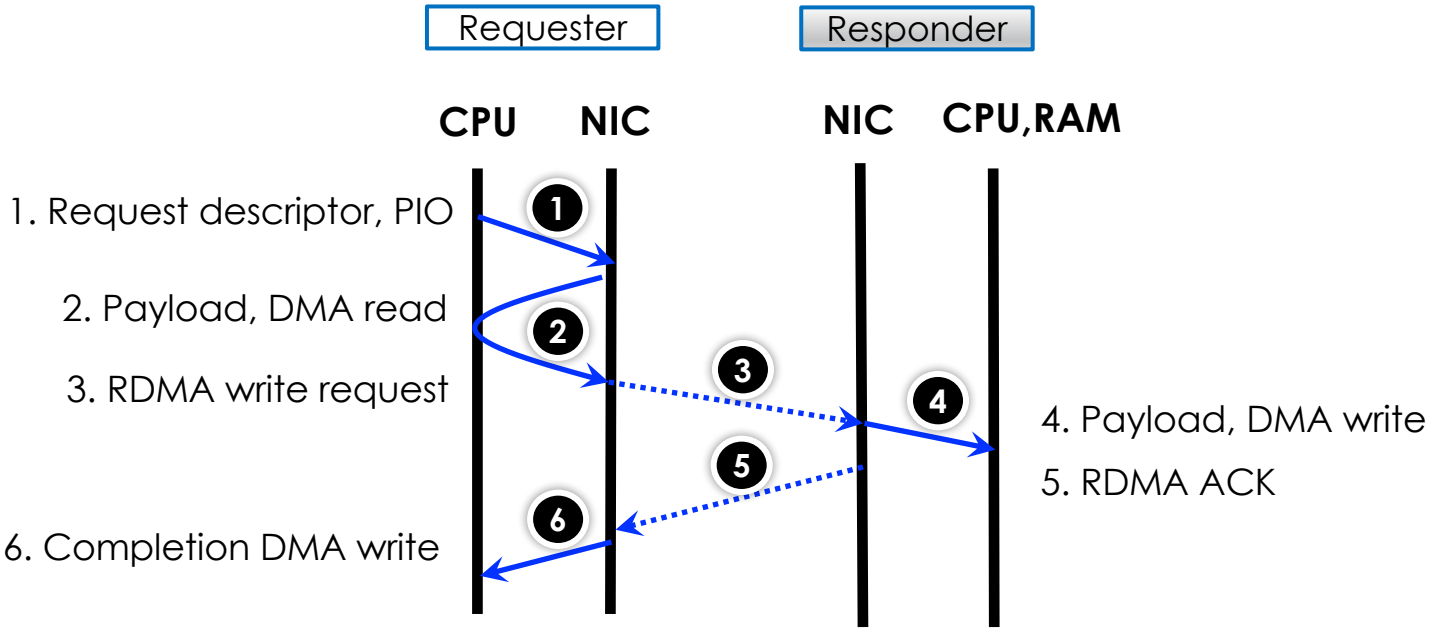
High speed, low latency & low CPU overhead

- Interface: SEND/RECV Verbs, and one-sided RDMA (READ/WRITE/CAS), IPoIB, etc.
- Round-trip time: one-sided/~3μs, verb msg/~7μs, IPoIB/~100 μs
- Bypasses OS kernels: Zero copy

One-sided DMA Primitives

Incl. RDMA READ, WRITE, CAS and XADD

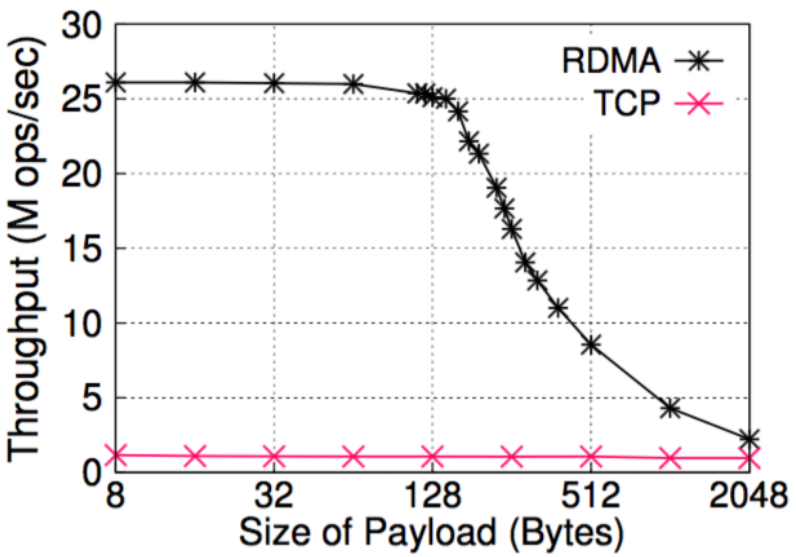
Life-cycle of an RDMA WRITE

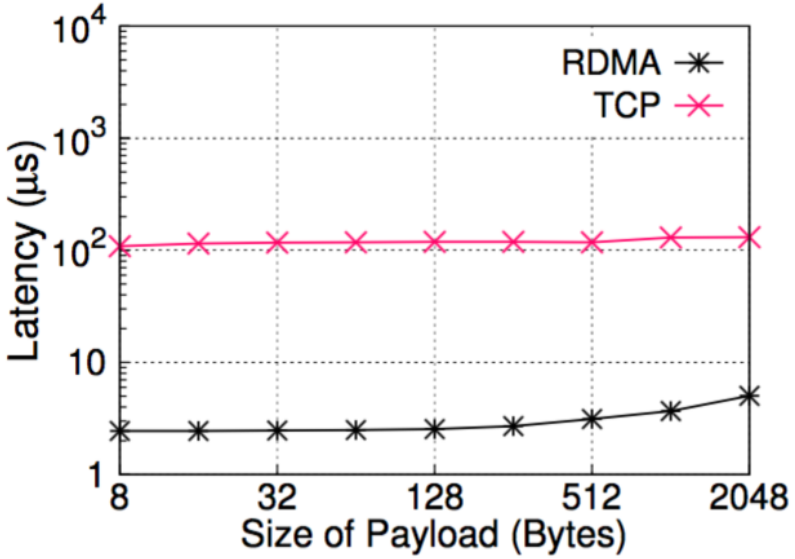


Credit: Anuj Kalia's SIGCOMM talk

One-sided RDMA Performance

Perf. of Random Read[1]





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

Opportunities with HTM & RDMA

HTM: **H**ardware **T**ransaction **M**emory

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

Strong Atomicity

RDMA: **R**emote **D**irect **M**emory **A**ccess

Opportunities with HTM & RDMA

HTM: **H**ardware **T**ransaction **M**emory

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

Strong Atomicity

RDMA: **R**emote **D**irect **M**emory **A**ccess

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

Strong Consistency

Opportunities with HTM & RDMA

HTM: **H**ardware **T**ransaction **M**emory

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

RDMA: **R**emote **D**irect **M**emory **A**ccess

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

HTM Strong Atomicity + **RDMA Strong Consistency** ⇒ **RDMA ops will abort conflicting HTM TX**

Opportunities with HTM & RDMA

HTM: **H**ardware **T**ransaction **M**emory

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

RDMA: **R**emote **D**irect **M**emory **A**ccess

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

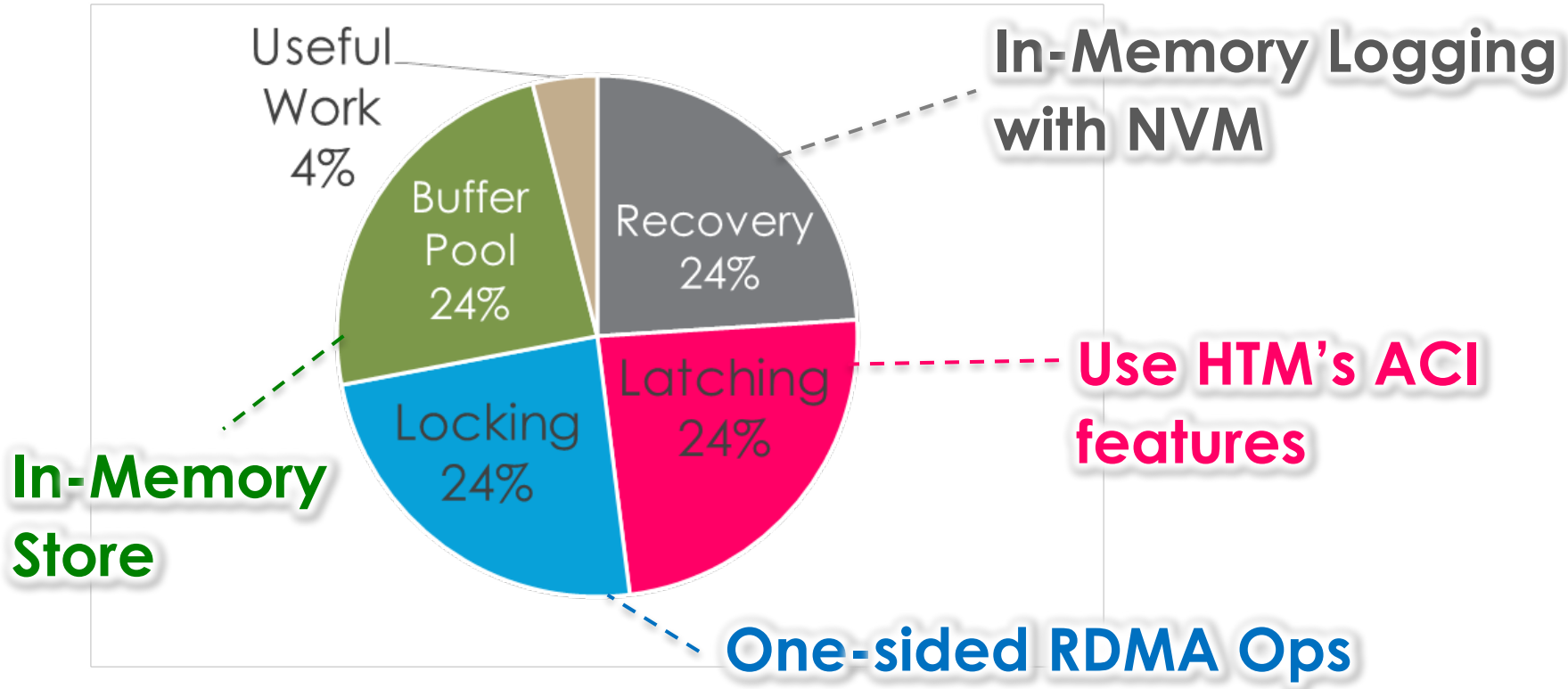
HTM Strong Atomicity + **RDMA Strong Consistency** ⇒ **RDMA ops will abort conflicting HTM TX**

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

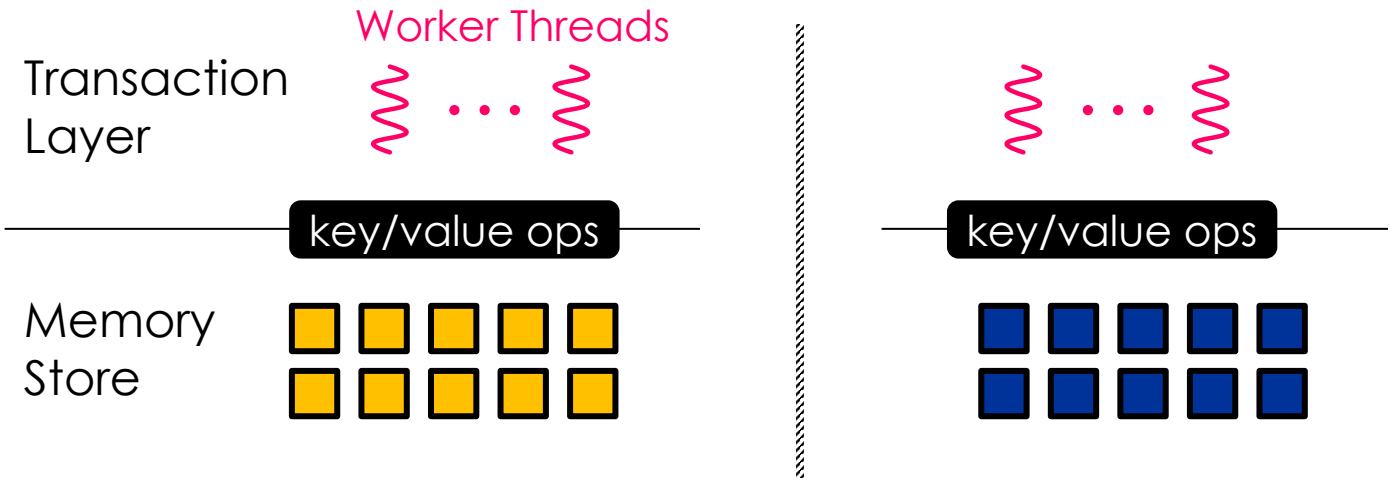


System Overview

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



Agenda

- Transaction Layer
- Memory Storage
- Implementation
- Evaluation

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

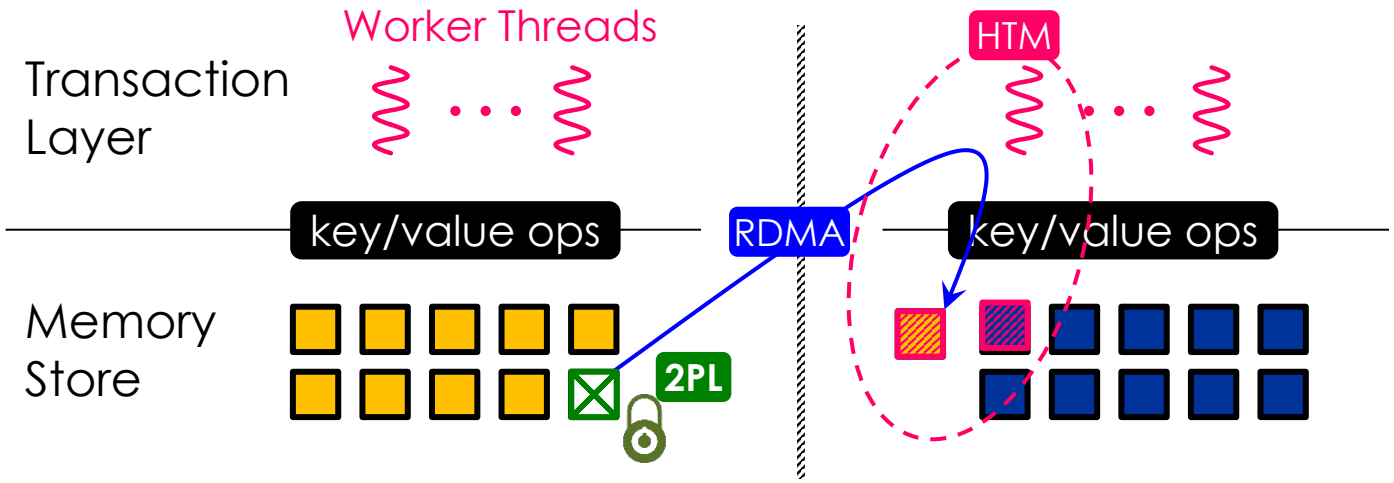
- Like any RDMA ops: READ/WRITE, CAS, SEND/RECV

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 transactions[1]



[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

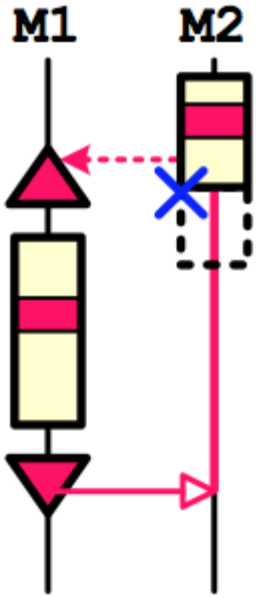
Local TX vs. Local TX: HTM

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

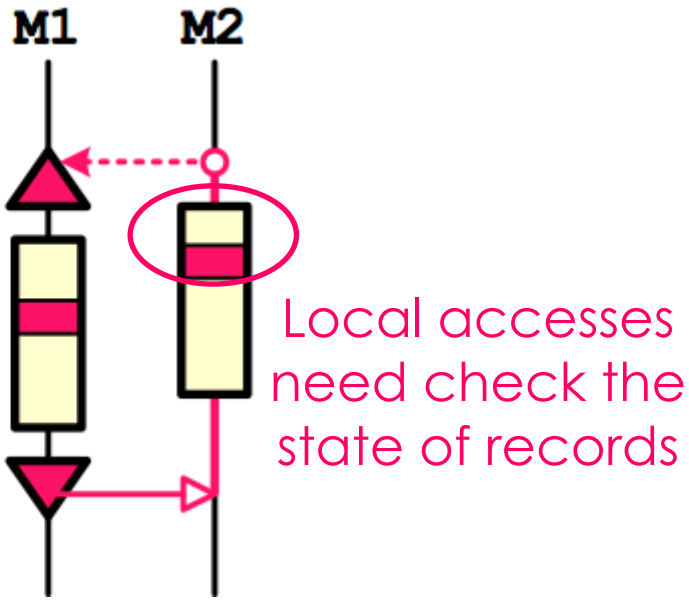
Local TX vs. Local TX: HTM

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)

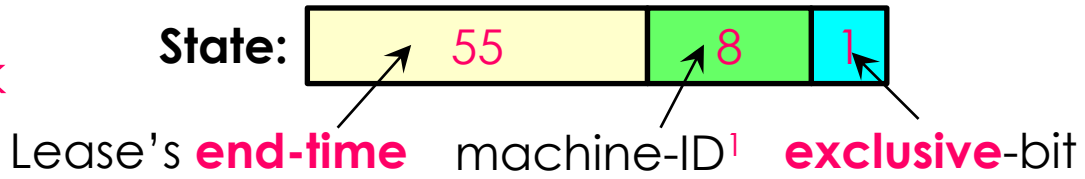
1. DrTM's **exclusive** lock
   - Spinlock: use RDMA CAS to acquire & release
2. DrTM's **shared** lock
   - Lease-based protocol

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



$000...000_2$ unlocked

$000...yy1_2$ exclusive locked

$xxx...000_2$ shared locked

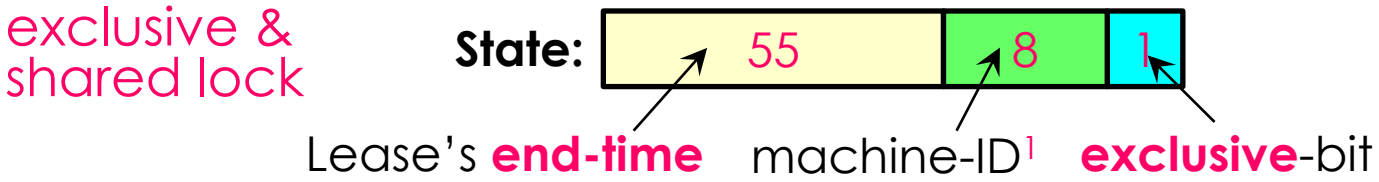
State is atomically compare and swap using RDMA CAS

[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 PTP[2]



$000...000_2$ unlocked

$000...yy1_2$ exclusive locked

$xxx..000_2$ shared locked

EXPIRED: if now > end-time + DELTA

INVALID: if now < end-time - DELTA

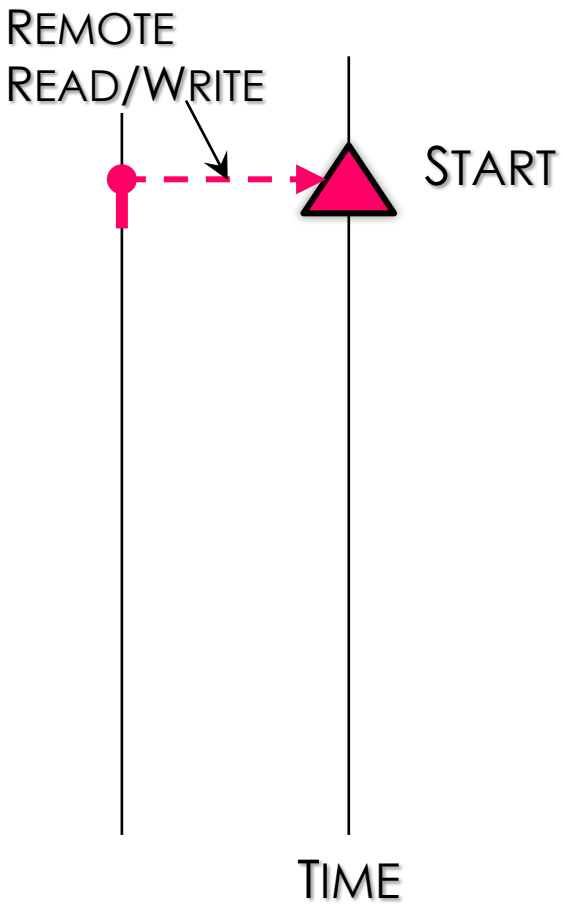
DELTA is used to tolerate the time bias among machines

[1] Machine ID is only used by recovery

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

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**

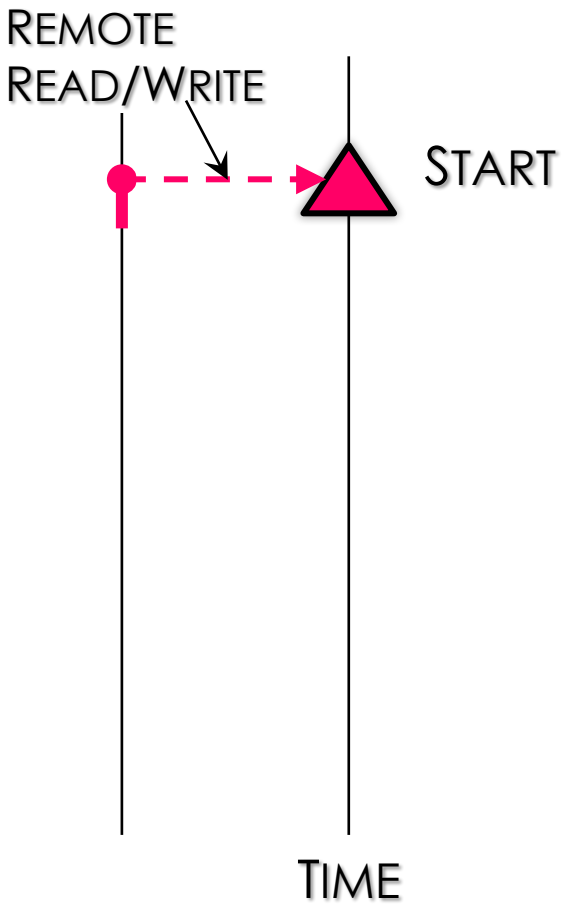



```
START (remote_writeset,remote_readset)
  foreach key in remote_writeset
    value = Exclusive_lock_fetch(key)
    cache[key] = value
  foreach key in remote_readset
    value = Shared_lease_fetch(key)
    cache[key] = value

  XBEGIN()
```

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**

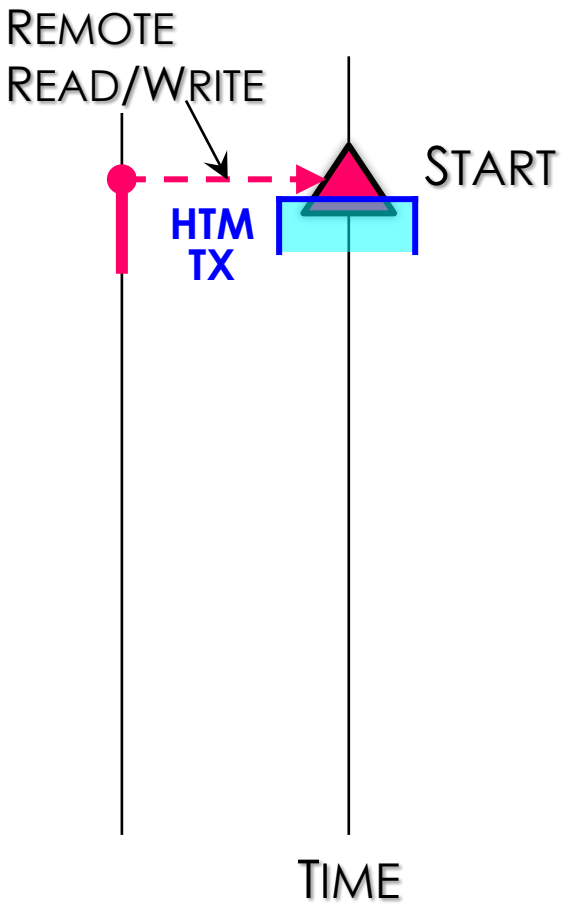



```
START(remote_writeset,remote_readset)
  foreach key in remote_writeset
    value = Exclusive_lock_fetch(key)
    cache[key] = value
  foreach key in remote_readset
    value = Shared_lease_fetch(key)
    cache[key] = value

  XBEGIN()
```

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**

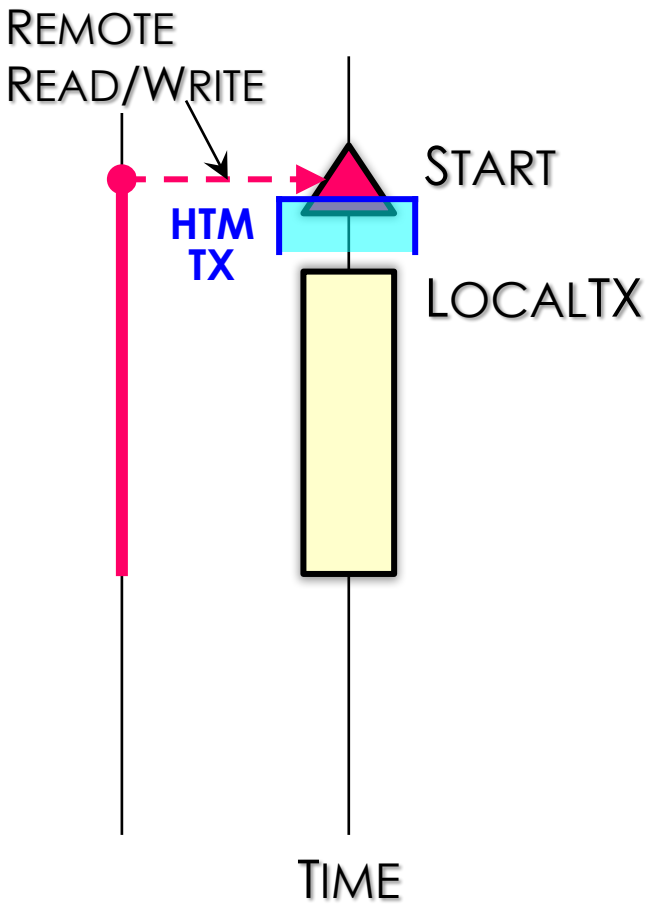



```
START(remote_writeset,remote_readset)
  foreach key in remote_writeset
    value = Exclusive_lock_fetch(key)
    cache[key] = value
  foreach key in remote_readset
    value = Shared_lease_fetch(key)
    cache[key] = value

  XBEGIN()
```

Transactional Read & Write

DrTM's Transaction: START + LOCALTX + COMMIT

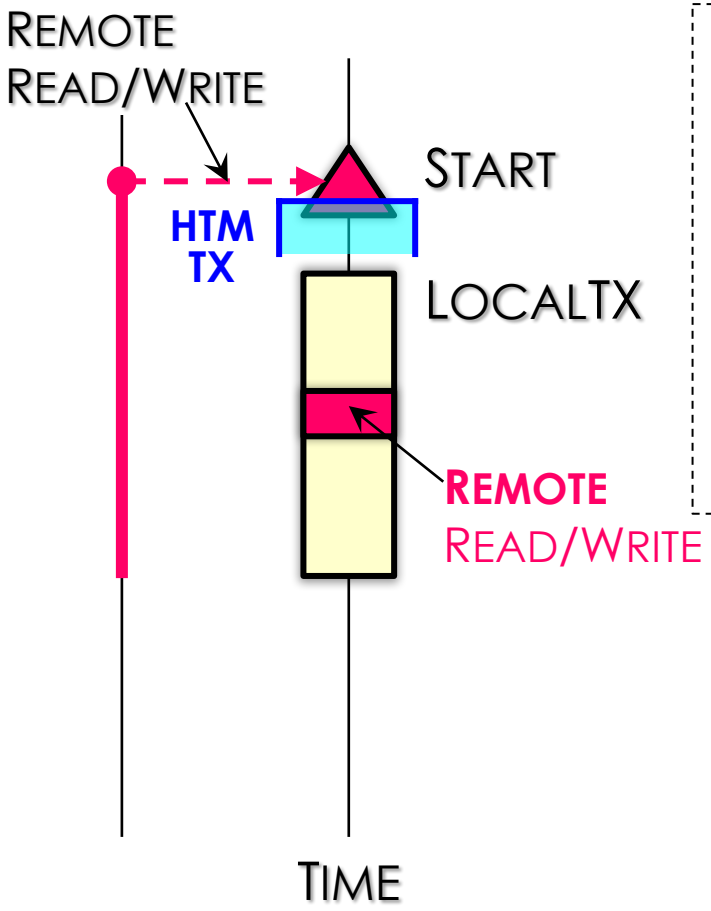



```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

Transactional Read & Write

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**

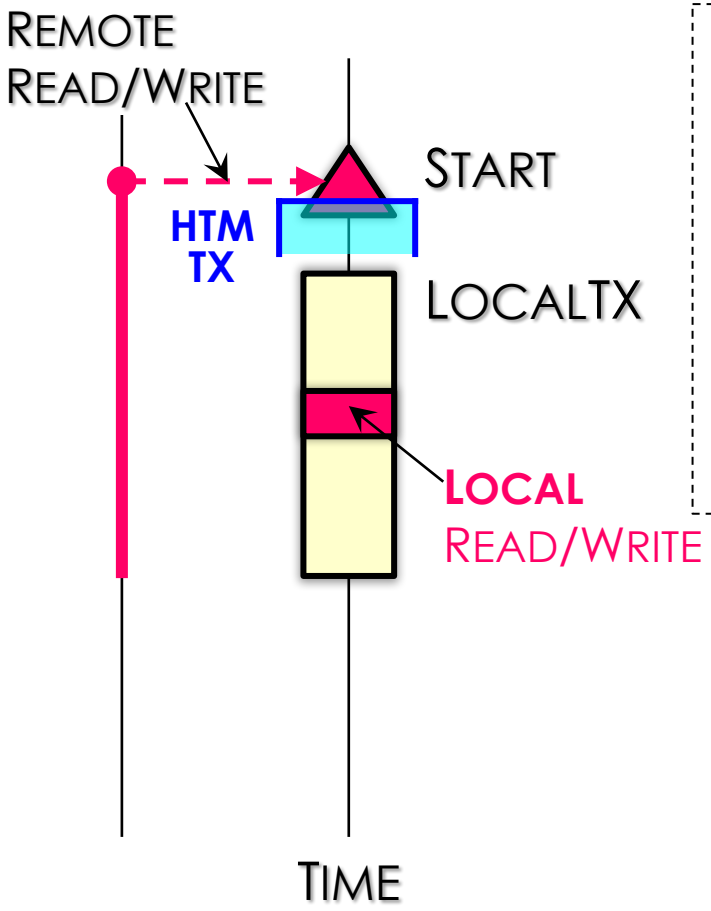



```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

Transactional Read & Write

DrTM's Transaction: START + LOCALTX + COMMIT

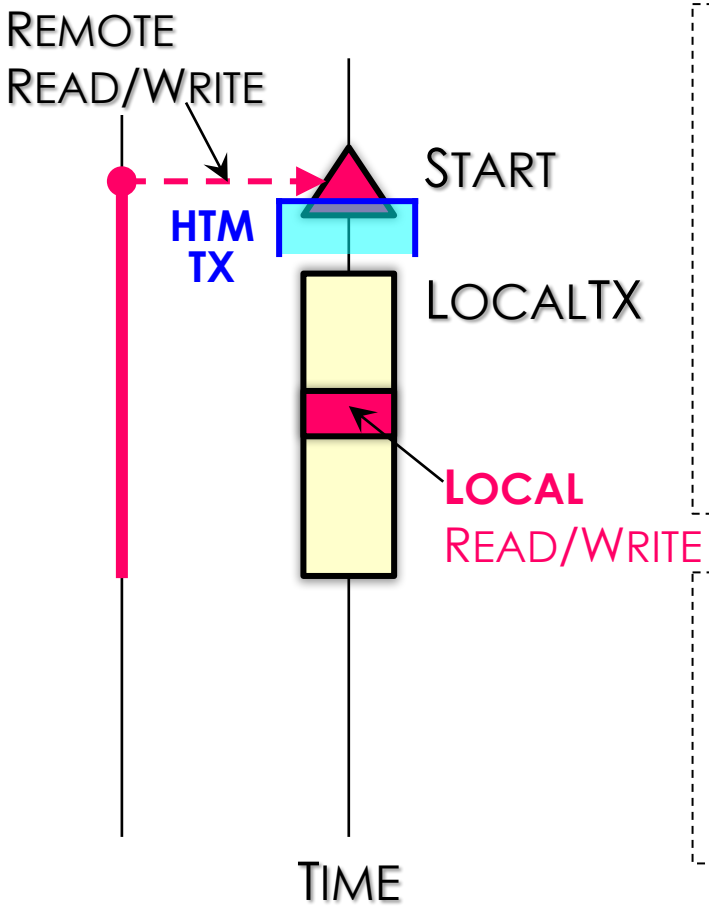



```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

Transactional Read & Write

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**



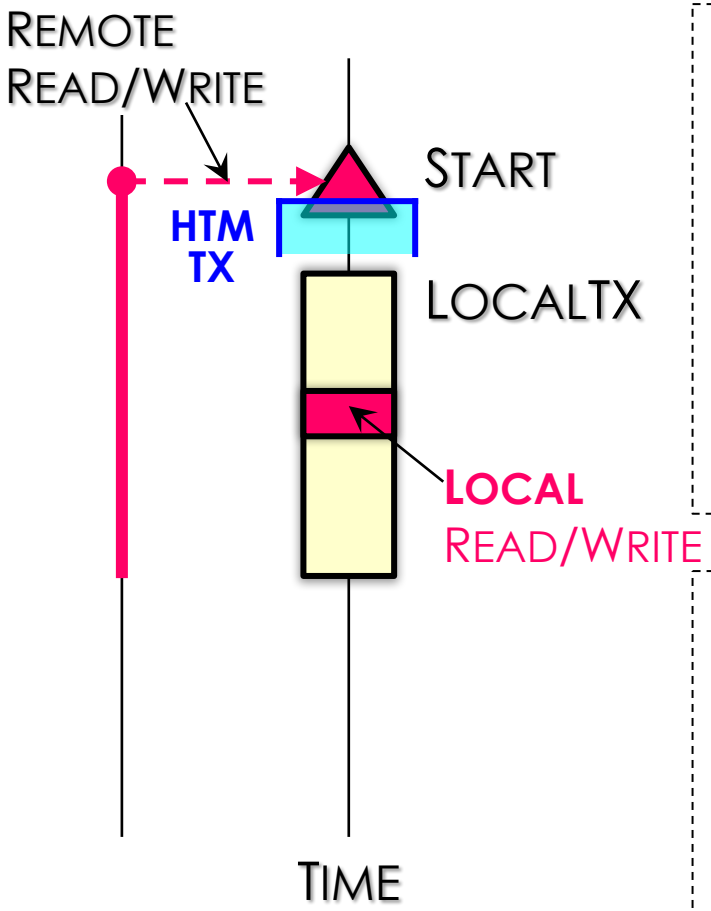

```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

```
LOCAL_READ(key)
  if states[key].w_lock == W_LOCKED
    ABORT()
  else
    return values[key]
```

Transactional Read & Write

DrTM's Transaction: START + LOCALTX + COMMIT



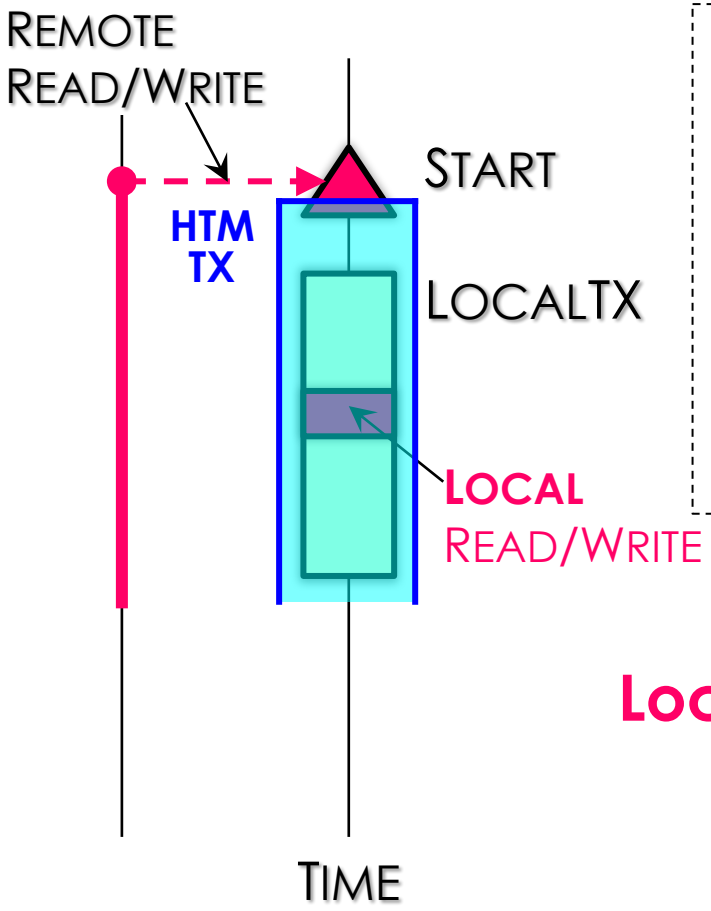

```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

```
LOCAL_WRITE(key, value)
  if states[key].w_lock == W_LOCKED
    ABORT()
   else if EXPIRED(END_TIME(states[key]))
    values[key] = value
  else ABORT()
```

Transactional Read & Write

DrTM's Transaction: START + LOCALTX + COMMIT



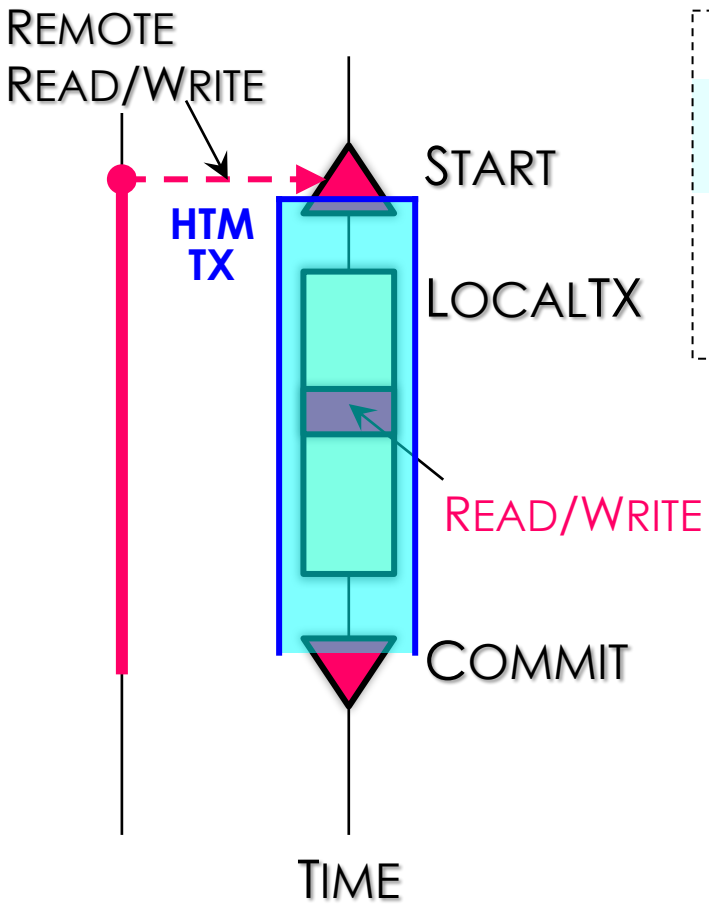

```
READ(key)
  if key.is_remote() == true
    return cache[key]
  else return LOCAL_READ(key)

WRITE(key, value)
  if key.is_remote() == true
    cache[key] = value
  else LOCAL_WRITE(key, value)
```

Local conflicts are detected by **HTM**

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**



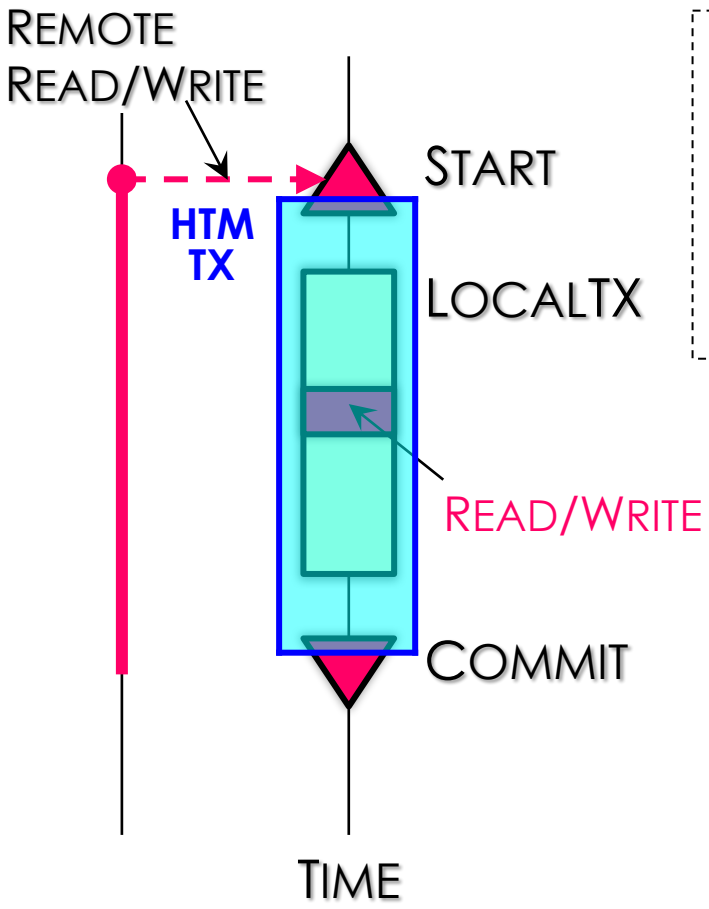

```
COMMIT(remote_writeset,remote_readset)
  if !VALID(end_time)
    ABORT()
  XEND()
  foreach key in remote_writeset
    RELEASE_WRITE_BACK(key,cache[key])
```

2PL: all **shared locks** must be released in shrinking phase

- Insert validation to all leases just before HTM commit

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**

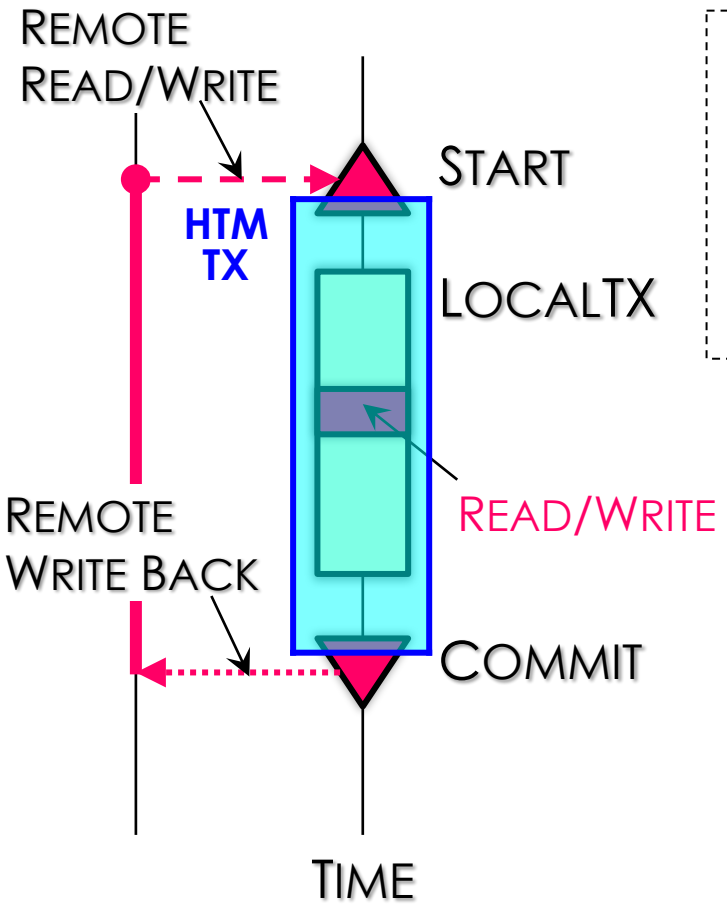



```
COMMIT(remote_writeset,remote_readset)
  if !VALID(end_time)
    ABORT()
  XEND()
  foreach key in remote_writeset
    RELEASE_WRITE_BACK(key,cache[key])
```

Commit **local** updates by **HTM**

Transaction Execution Flow

DrTM's Transaction: **START** + **LOCALTX** + **COMMIT**




```
COMMIT(remote_writeset,remote_readset)
  if !VALID(end_time)
    ABORT()
  XEND()
  foreach key in remote_writeset
    RELEASE_WRITE_BACK(key,cache[key])
```

2PL & HTM → Serializability

Commit **local** updates by **HTM**

Commit **remote** updates by **RDMA**

Agenda

Transaction Layer

Memory Storage

Implementation

Evaluation

Other Specific Implementation

Transaction chopping: reduce HTM working set

Fine-grained RTM's fallback handler

Atomicity Issues: RDMA CAS vs. Local CAS

Horizontal scaling across socket: logical node

Avoiding remote range query

Durability: Cooperative Logging

Platform: *Intel E5-2650 v3 RTM-enabled*
Mellanox ConnectX-3 56GB InfiniBand

Agenda

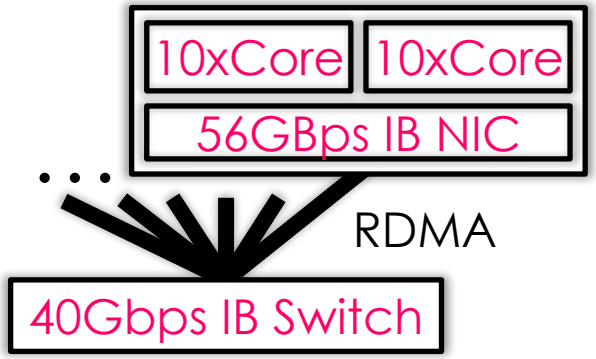
Transaction Layer

Memory Storage

Implementation

Evaluation

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]

Benchmark[2]

- TPC-C

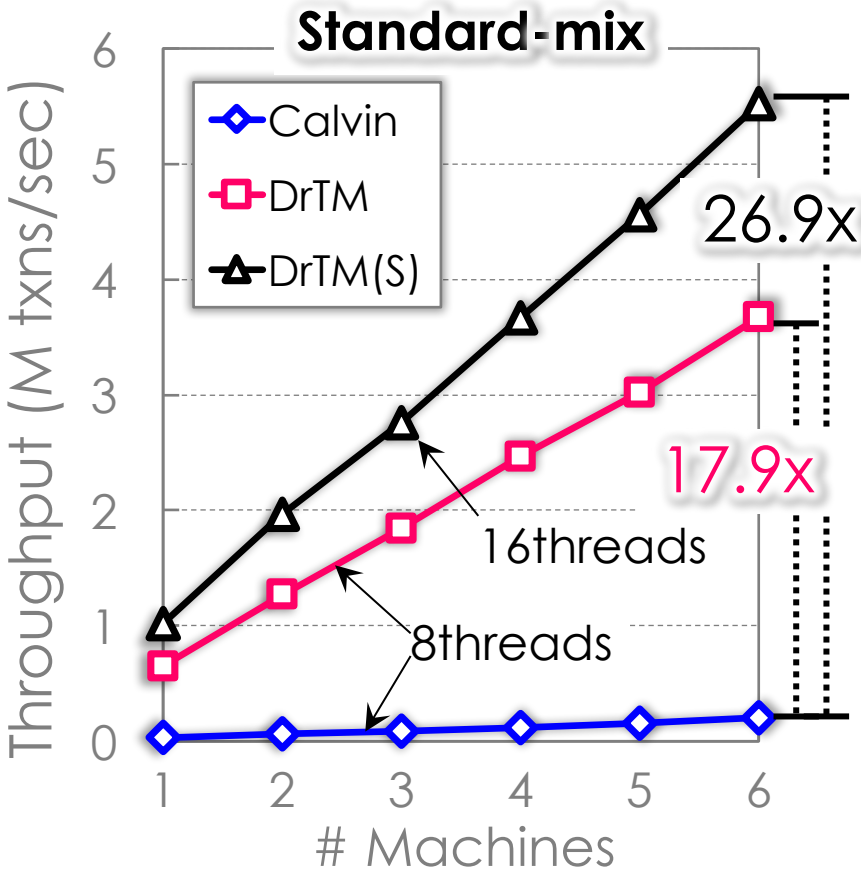
TPC-C	NEW	PAY	DLY	OS	SL
Ratio	45%	43%	4%	4%	4%
Type	d+rw	d+rw	l+rw	l+ro	l+ro

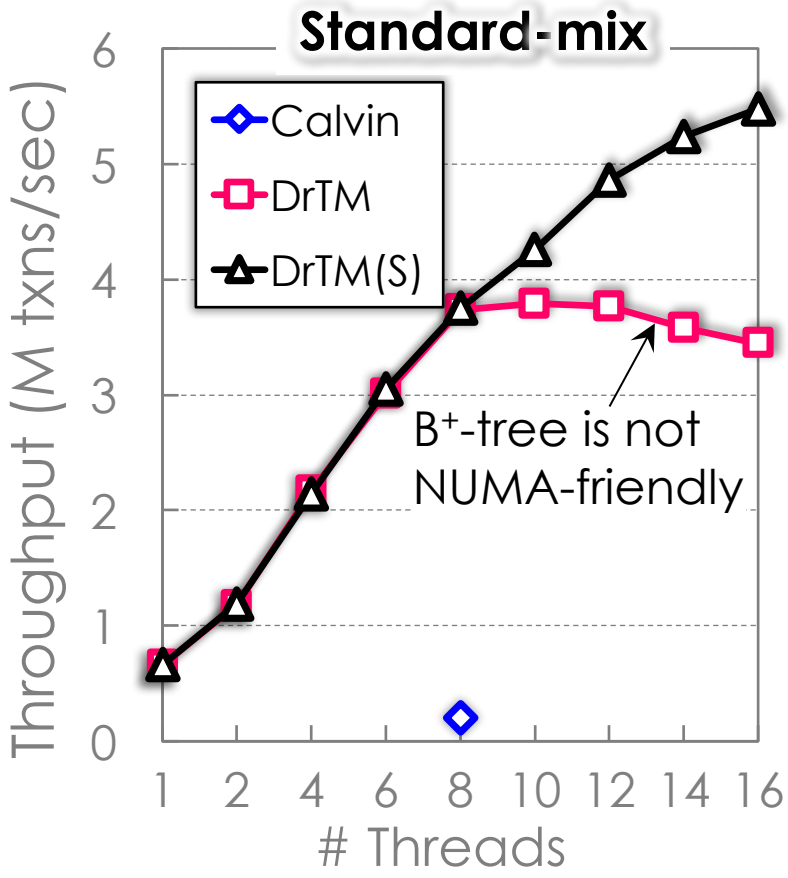
[1] All machines run Ubuntu 14.04 with Mellanox OFED v3.0-2.0.1 stack.
[2] **d** and **l** stand for distributed and local. **rw** and **ro** stand for read-write and read-only.

Performance on TPC-C

New-order TX
≈ Standard-mix x45%

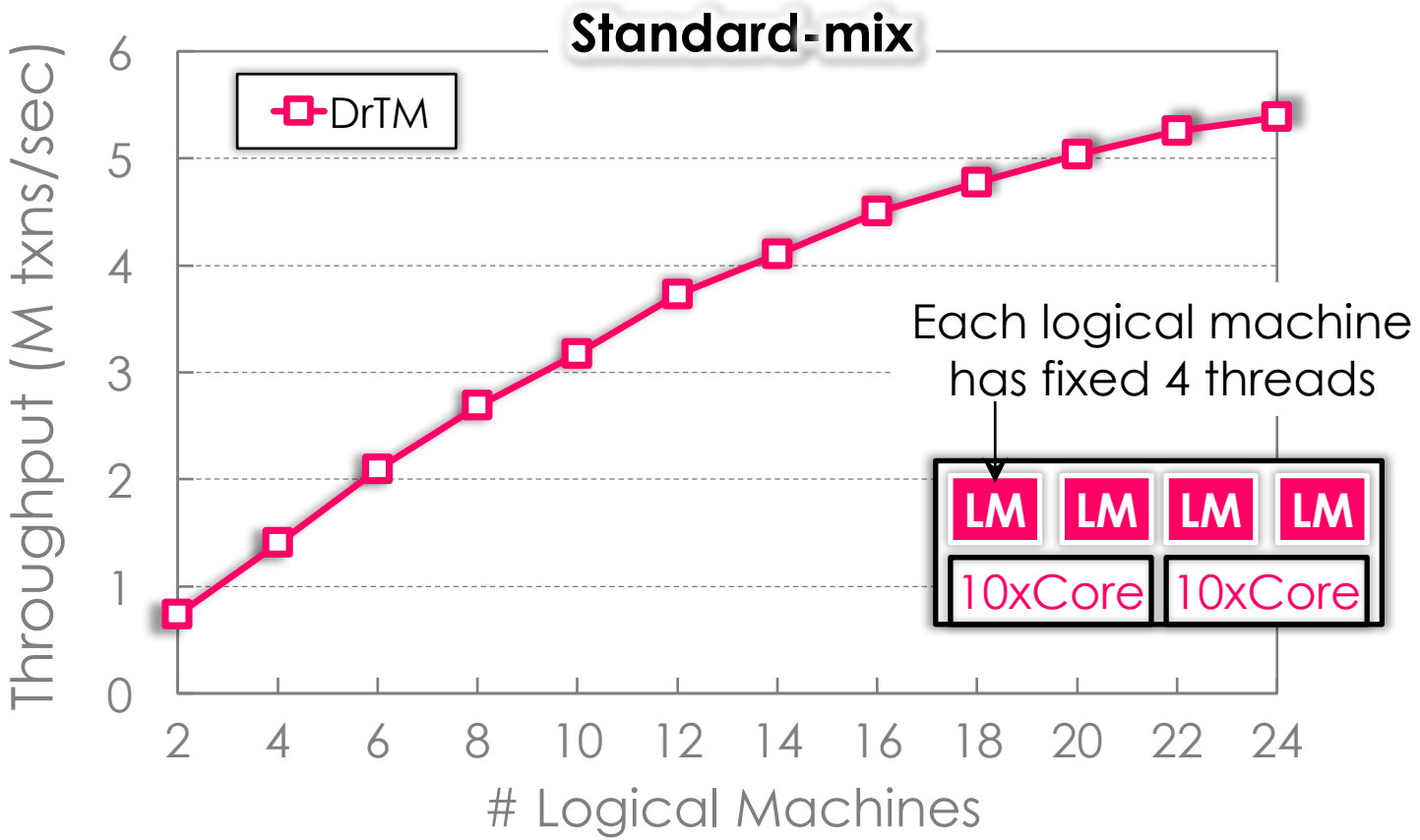
DrTM(S): run a separate logical node on each socket





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

Limitations of DrTM

Require advance knowledge of read/write sets of transactions

Provide only an HTM/RDMA-friendly hash table for unordered stores, w/o B^+-tree support

Preserve durability rather than availability in case of machine failures

Performance is related to locality of TXs

Local ops are much faster than remote ones

Conclusion

High COST of concurrency control in distributed transactions calls for new designs

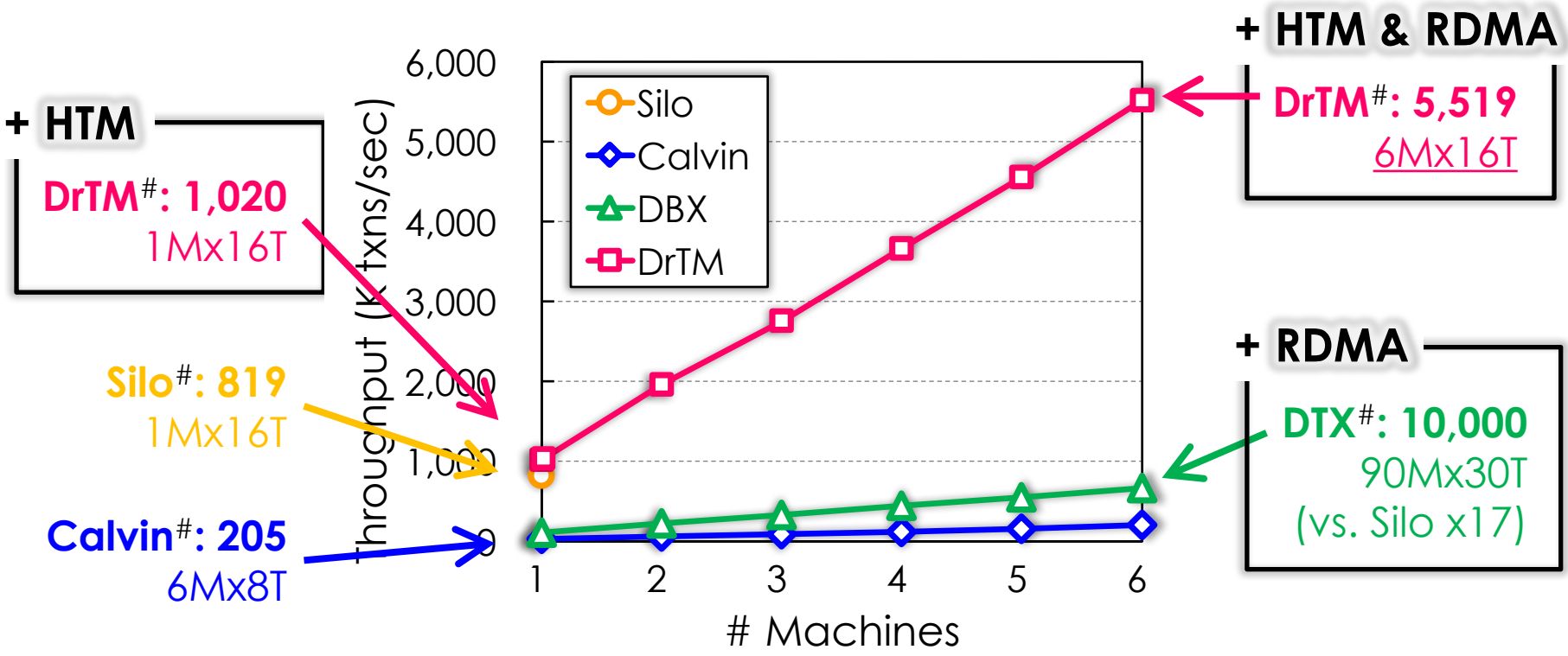
New hardware technologies open opportunities

DrTM : The first design and impl. of combining HTM and RDMA to boost in-memory transaction system

Achieving orders-of-magnitude higher throughput and lower latency than prior general designs

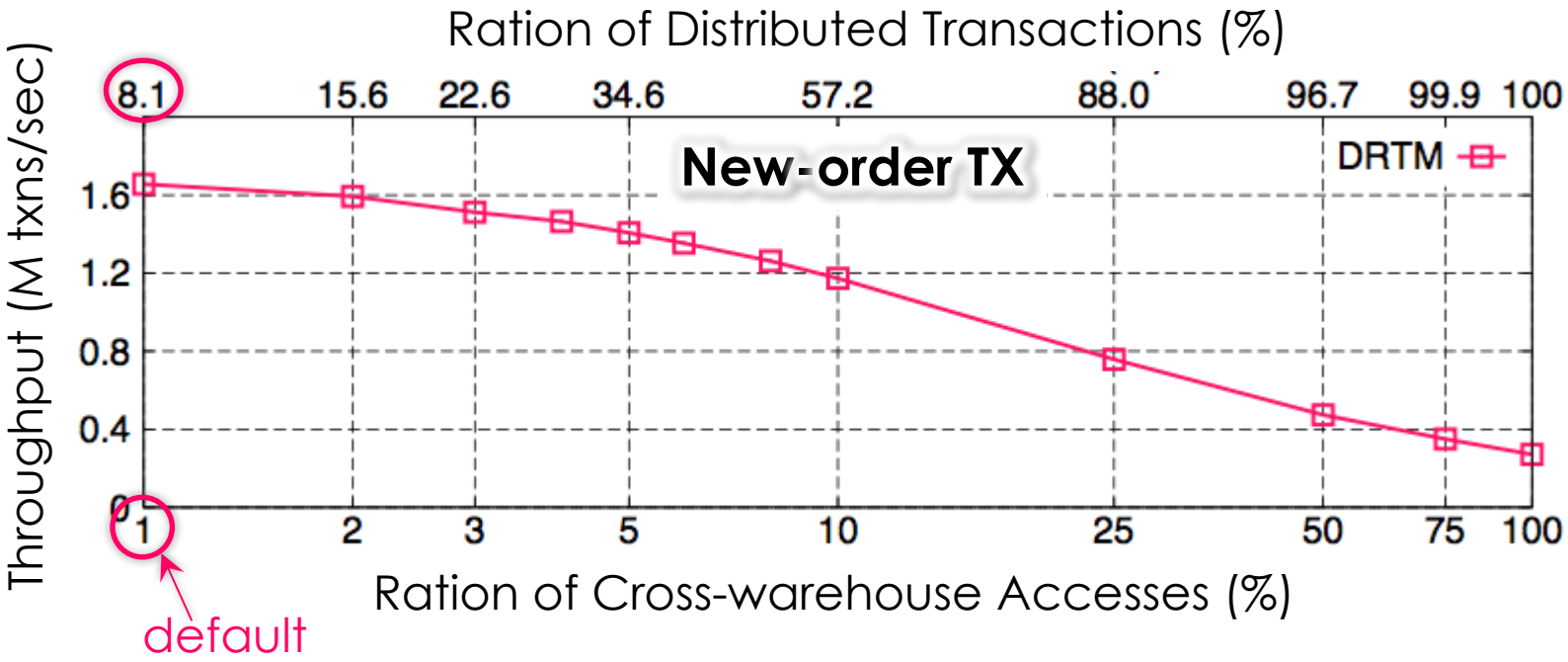
Backup

End-to-end Comparison of TPC-C



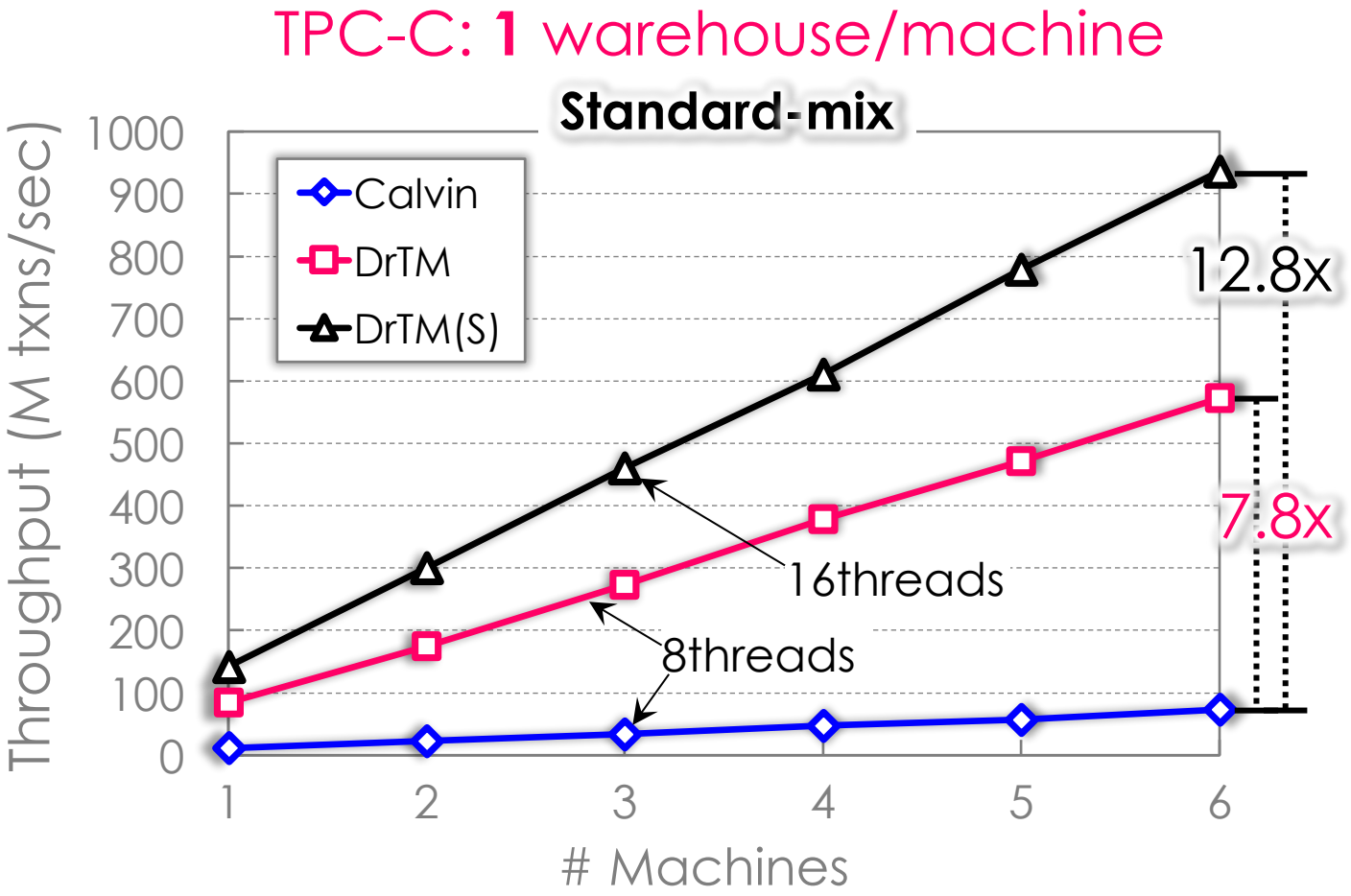
Each machine: 20-core Intel E5-2650 w/o HT, Mellanox ConnextX-3 56Gbps IB, 1 warehouse/T
$ Each machine: 16-core Intel E5-2650 w/ HT, Mellanox ConnectX-3 56Gbps IB, 240 warehouse/M

Impact from Distributed Transaction



High Contention

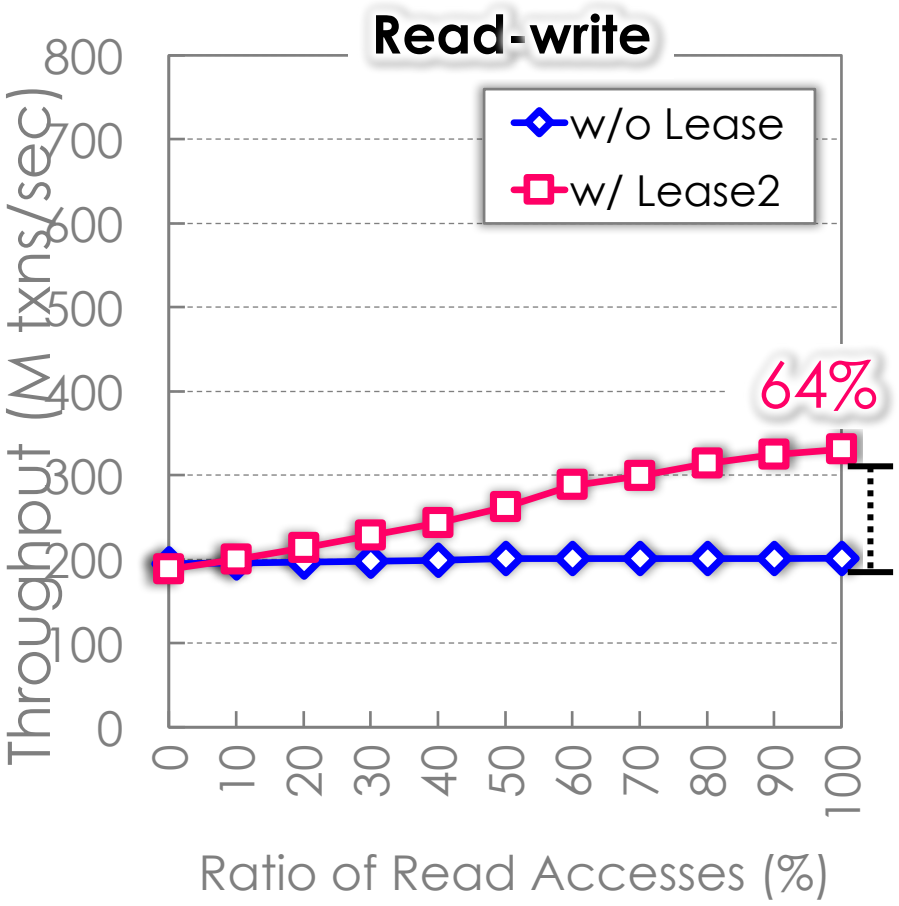
New-order TX
≈ Standard-mix x45%



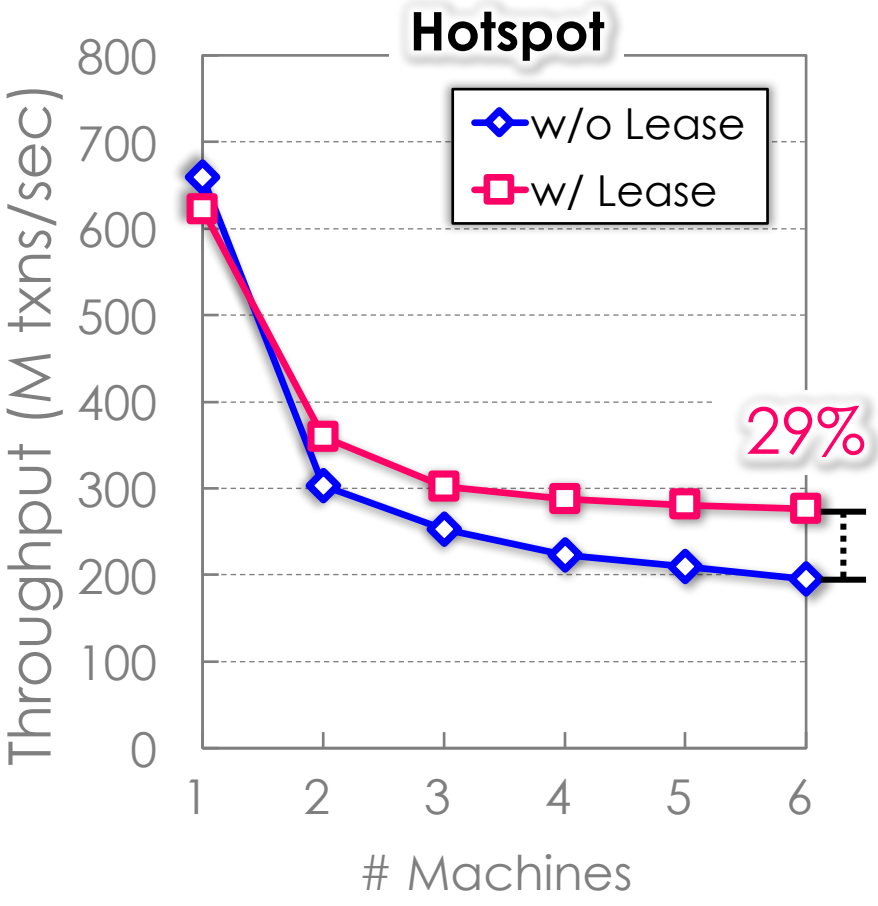
DrTM(S): run a separate logical node on each socket

Lease

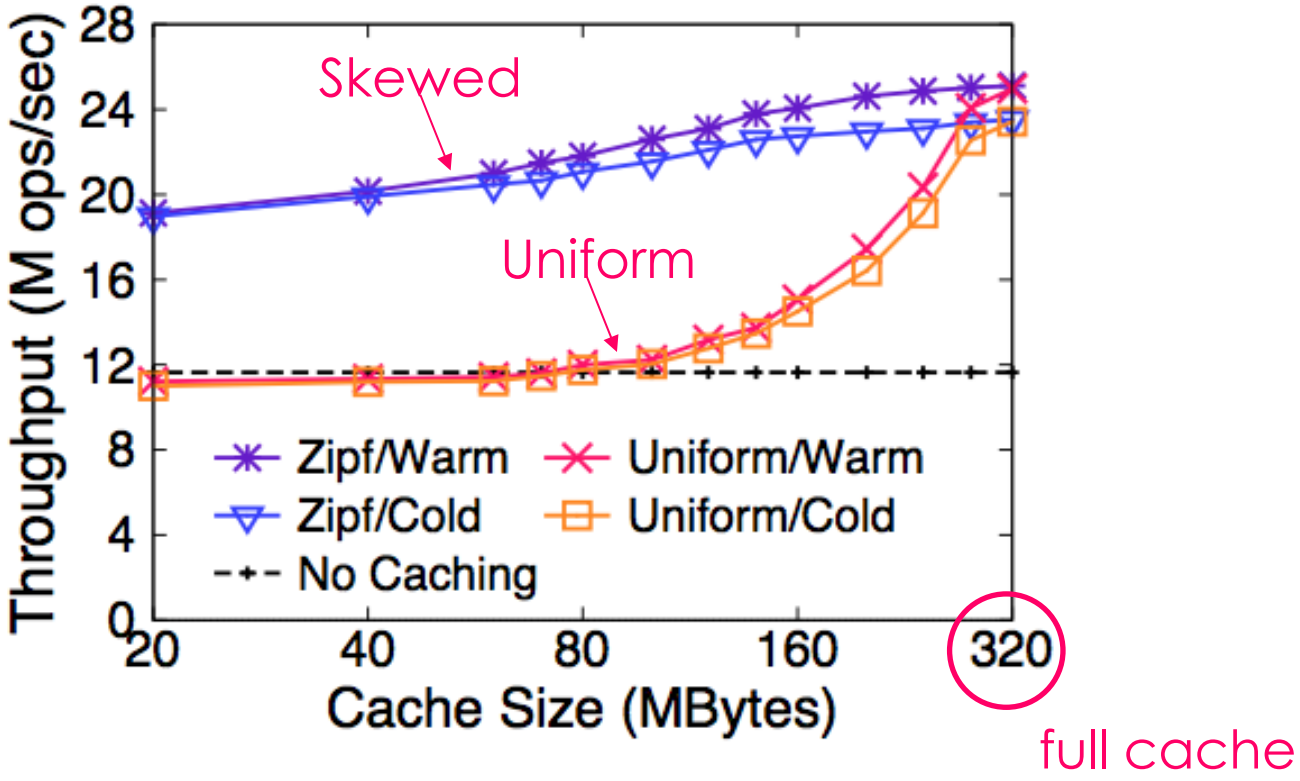
Parts of records (0%-100%) does not write back (read)



1 of 10 records is chosen from 120 hotpot records



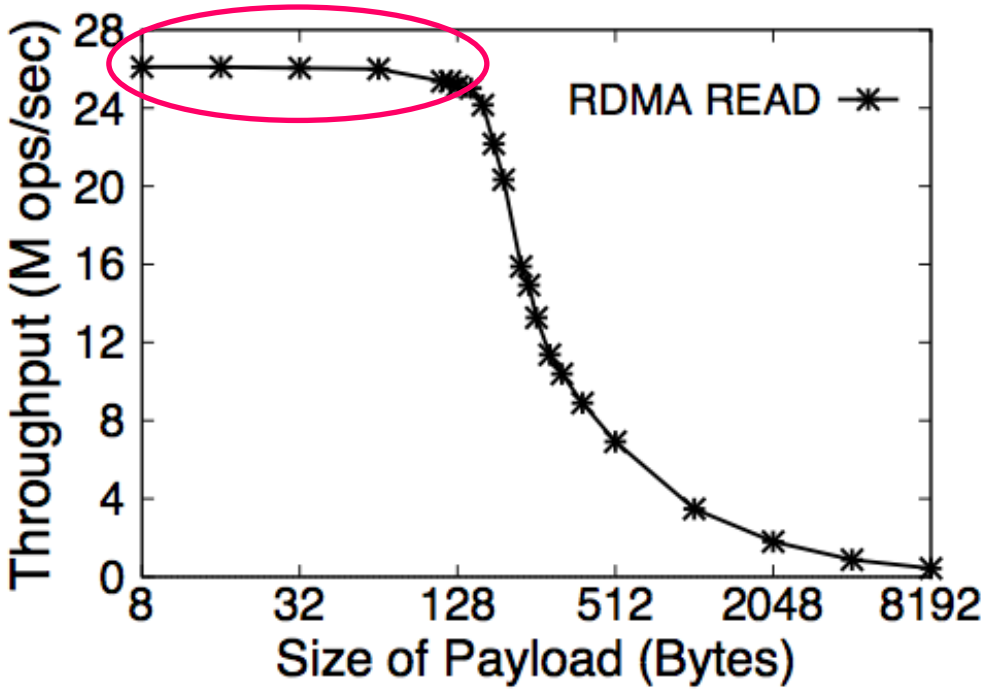
Location-based Cache



Setting: 1 server and 5 clients (up to 8 threads), 20 million k/v pairs
Traditional replacement policy (i.e., LRU)

RDMA READ

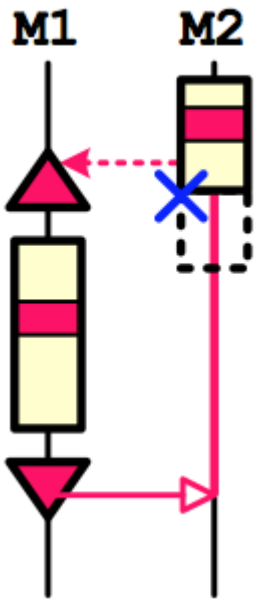
Peak throughput ≈ 26 Mops/sec



Testbed: Mellanox ConnectX-3 MCX353A
56Gbps InfiniBand NIC w/ RDMA

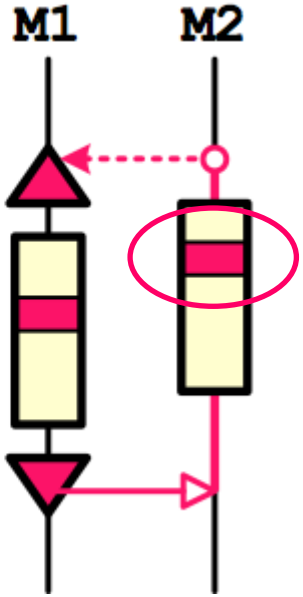
Local TX vs. Distributed TX

Local TX prior to Distributed TX



RDMA op will abort local TX

Distributed TX prior to Local TX



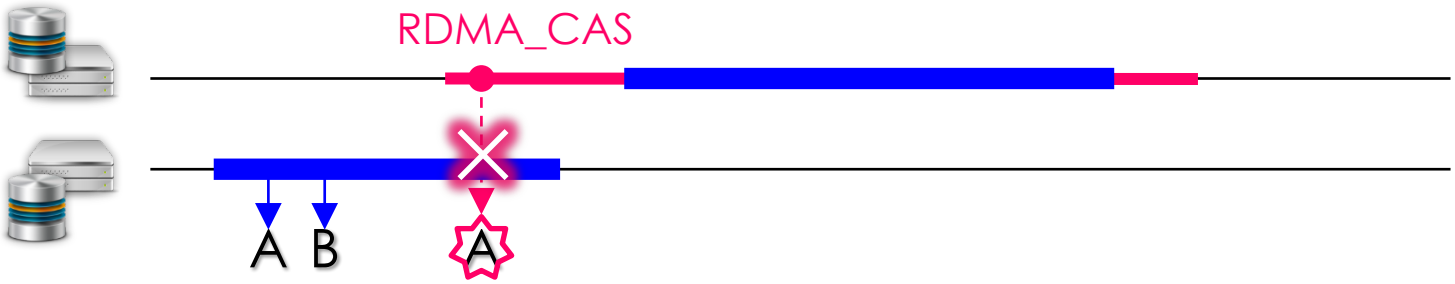
Local accesses need check the state of records

False Conflict

write locked

read locked

expired




```
TXN:
  read A
  write B
```

```
REMOTE_READ(key, end_time)
  _s = INIT
L:s = RDMA_CAS(key, _s, R_LEASE(end_time))
  if s == _s //SUCCESS: init
    read_cache[key] = RDMA_READ(key)
    return end_time
  else if s.w_lock == W_LOCKED
    ABORT() //ABORT: write locked
  else
    if EXPIRED(END_TIME(s))
      _s = s
      goto L //RETRY: correct s
    else //SUCCESS: unexpired leased
      read_cache[key] = RDMA_READ(key)
      return s.read_lease
```

L_: local
R_: remote

RD: read
WR: write
WB: write-back

	L_RD	L_WR	R_RD	R_WR	R_WB
State	RS	RS	WR	WR	WR
Value	RS	WS	RD	RD	WR

RS: read-set
WS: write-set

RD: read
WR: write

False conflict only impacts little performance not correctness

DrTM's Failure Model

Failure model

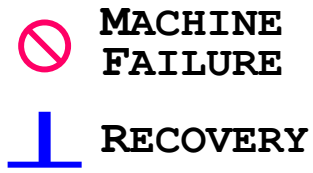
- Similar to WSP[ASPLOS'12] and DTX[SOSP'15]
- Assume flush-on-failure policy

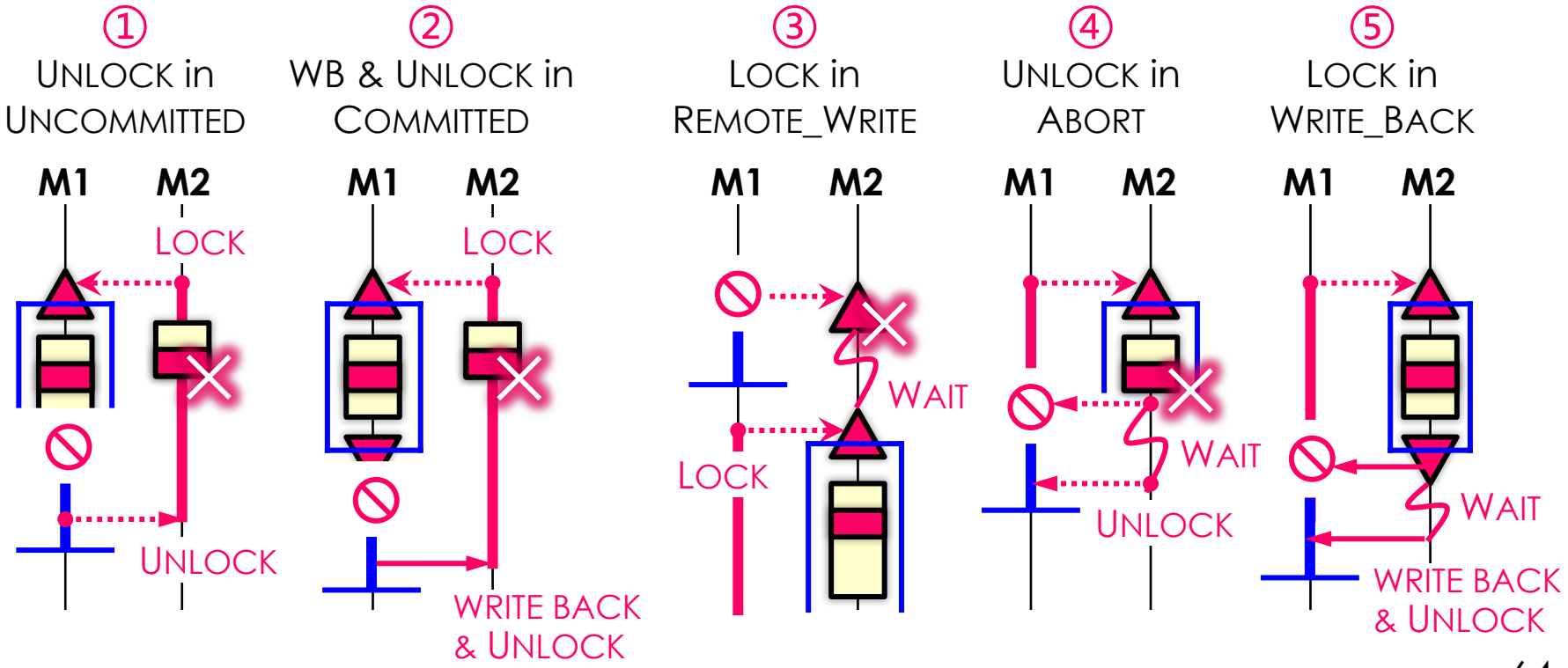
> Flush any transient state in registers and cache lines to non-volatile DRAM (NVRAM) and finally to a persistent storage (SSD) upon a failure by the power from UPS

- Fail-stop crash instead of arbitrary failures (e.g., BFT)
- Zookeeper
  - Detect machine failures by a heartbeat mechanism
  - Notify surviving machines to assist the recovery of crashed machines

Cooperative Recovery

1. Crashed machine: recovery from logs
2. Surviving machine: suspend & redo





Related Work

In-memory Transaction Processing

- **General**: Spanner[OSDI'12], Calvin[SIGMOD'12], Silo[SOSP'13], Lynx[SOSP'13], Hekaton[SIGMOD'13], Salt[OSDI'14], Doppel[OSDI'14], and ROCOCO[OSDI'14]
- **HTM**: DBX[EuroSys'14], TSO[ICDE'14] and DBX-TC[TR'15]
- **RDMA**: FaRM[NSDI'14] and DTX[SOSP'15]

Key-value Store with RDMA

- Pilaf[ATC'13], FaRM[NSDI'14], HERD[SIGCOMM'14], and C-Hint[SoCC'14]

Distributed Transactional Memory

- Ballistic[DISC'05], DMV[PPoPP'06], and Cluster-STM[PPoPP'08]

Lease

- Megastore[CIDR'11], Spanner[OSDI'12], and Quorum leases[SoCC'14]

Performance on Smallbank