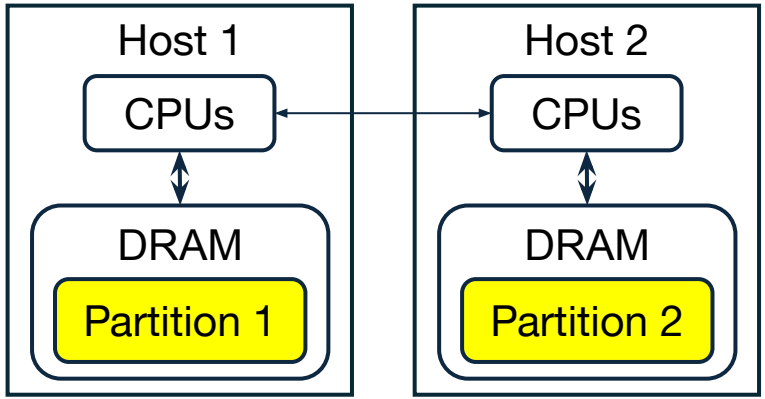
Tigon: A Distributed Database for a CXL Pod

Yibo Huang, Haowei Chen, Newton Ni, Yan Sun[1],
Vijay Chidambaram, Dixin Tang, Emmett Witchel

The University of Texas at Austin
[1]*University of Illinois Urbana-Champaign*

Scaling Out a Transactional Database is Hard…



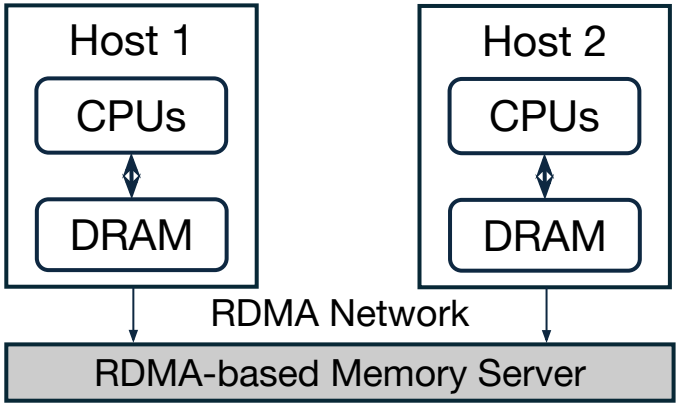
Partition-based Shared-nothing DB

Pros:

- Single-partition transactions scale well

Cons:

- Numerous message exchange
- Require two-phase commit (2PC)



RDMA-based Shared-memory DB

Pros:

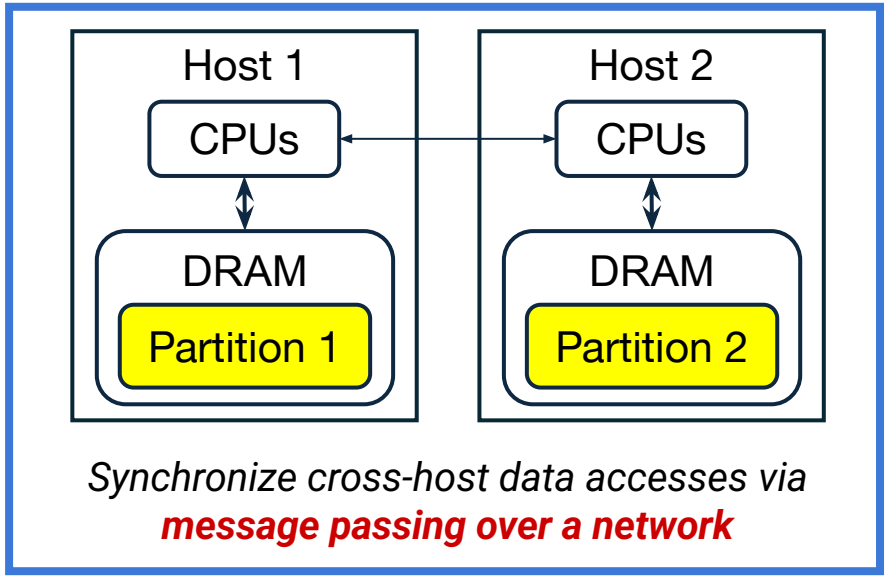
- No message exchange
- No two-phase commit (2PC)

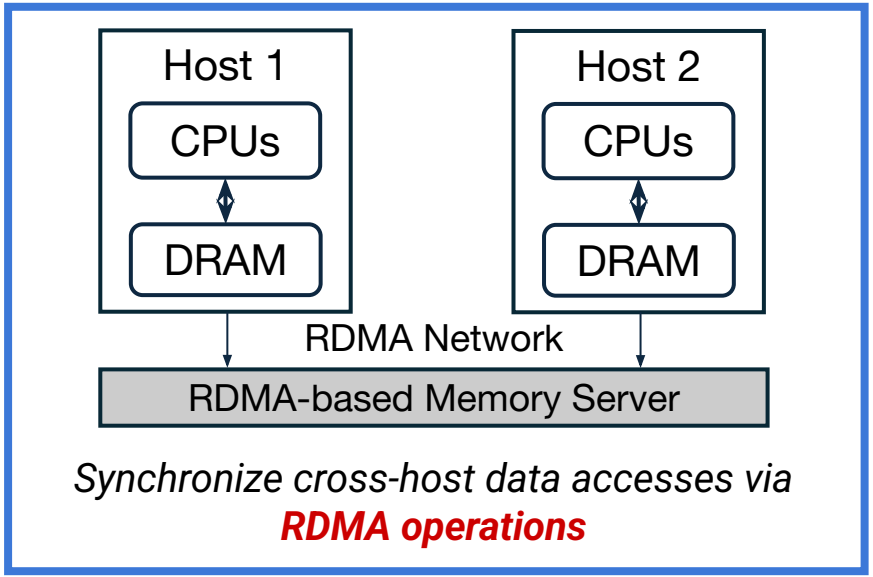
Cons:

- RDMA has microsecond latency
- High programming complexity

Scaling Out a Transactional Database is Hard…

Root cause: use of a **network** for synchronizing cross-host data accesses





Existing DBs either suffer from numerous message exchanges or high latency memory access, **both introduced by the network**

Scaling Out a Transactional Database is Hard…

Can we avoid using a network for cross-host data synchronization?

Compute Express Link (CXL)

A fast **interconnect** between CPUs and devices based on PCIe 5.0 and 6.0

CXL memory is a memory module connected to one or multiple hosts via CXL

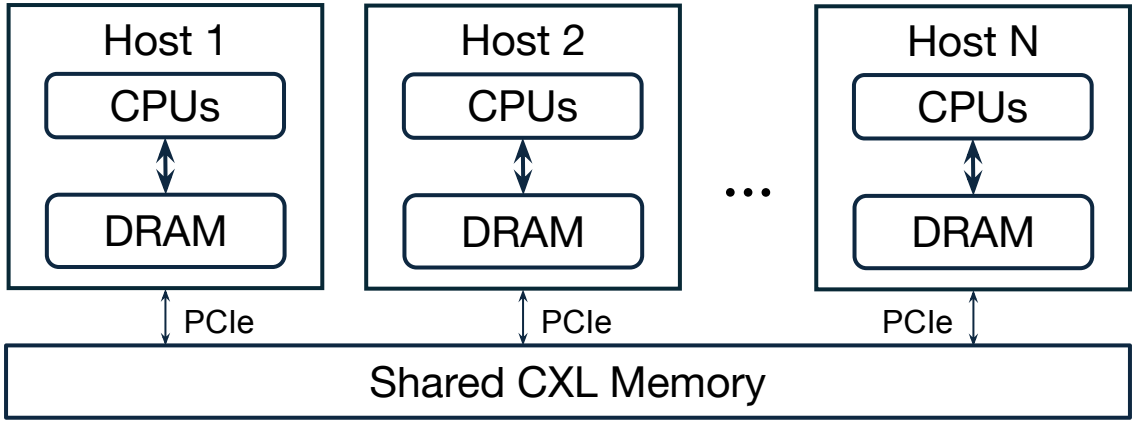
Memory tiering and pooling (1.0 & 2.0):

TPP (ASPLOS 23), Pond (ASPLOS 23), Nomad (OSDI 24), Memstrata (OSDI 24), Colloid (SOSP 24), Soar & Alto (OSDI 25)

Memory sharing (3.0): **Our focus!**

CXL-SHM (SOSP 23), HydraRPC (ATC 24), TrEnv (SOSP 24), CXLfork (ASPLOS 25), CtXnL (ASPLOS 25), PolarDB DMP (SIGMOD 25)

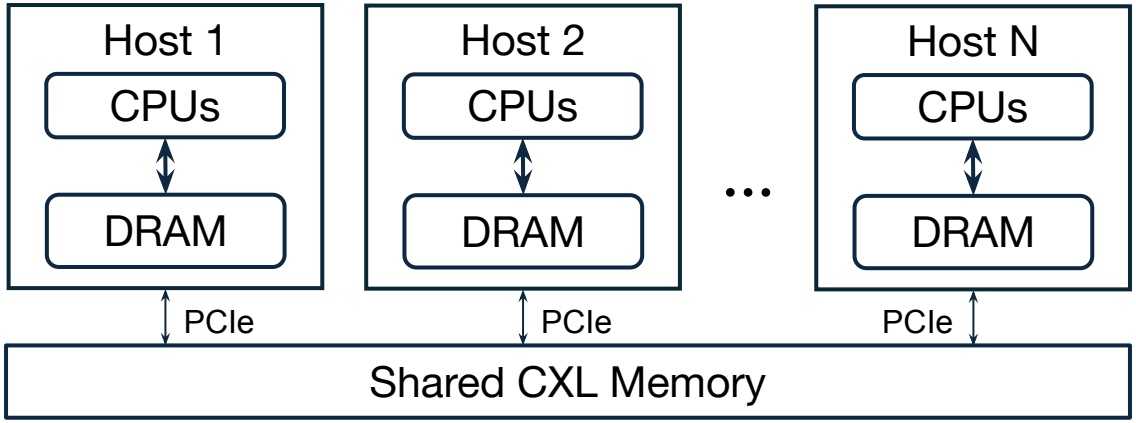
CXL Pod: A New Opportunity to Scale Databases



- CXL pod: 8-16 hosts connected to a shared CXL memory module
  - Inter-host hardware cache coherence (HWcc) based on CXL 3.0-3.2 spec
  - Hardware prototype exists - e.g., Niagara 2.0 from SK Hynix (no HWcc)
- Advantages over RDMA:
  - Lower latency – hundreds of nanoseconds
  - Supports load/store instructions
  - HWcc

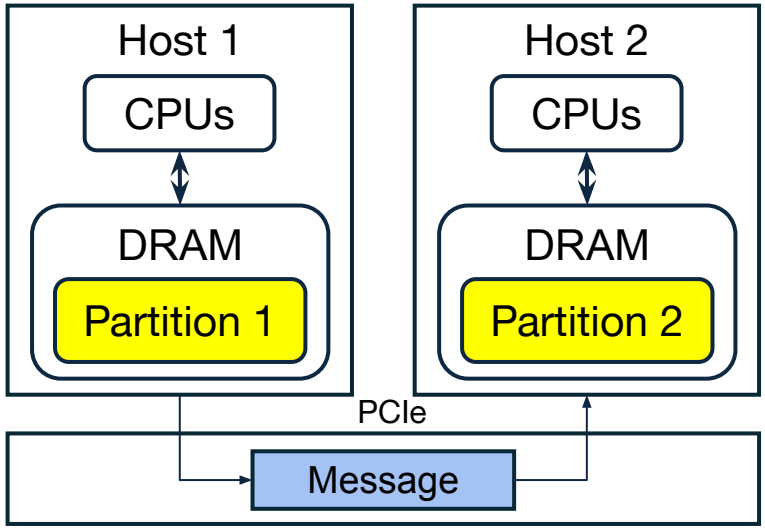
CXL Pod: A Distributed System w/ Shared Memory

Distributed System: Run a database on multiple nodes



Shared Memory: Instead of synchronizing cross-host data accesses over a network, let's do it over memory!

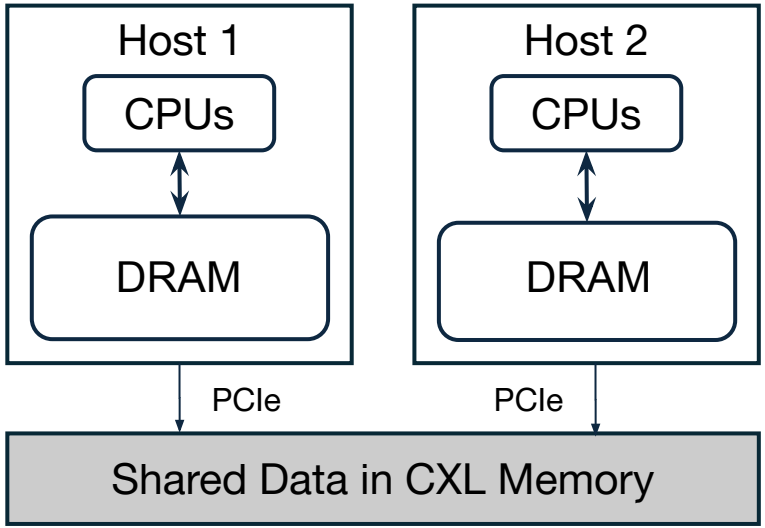
How to Scale Out a Database on a CXL Pod?



Partitioning Data + CXL Transport

- At most 2.0× faster
- Numerous message exchange
- Require two-phase commit (2PC)

2.0× faster but still bad



Sharing everything in CXL memory

- No message exchange
- No two-phase commit (2PC)

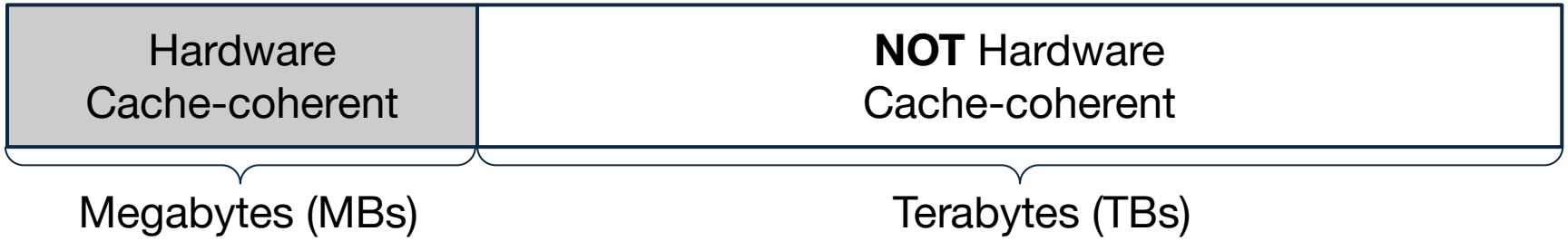
Does this work?

CXL Pod: Challenges

- Higher latency than DRAM (250-400 ns[1])
- Lower bandwidth than DRAM (9-11 GB/s single channel[1])

Takeaway 1: Performance suffers if all data is in CXL memory

- Limited-size hardware cache coherence[2]

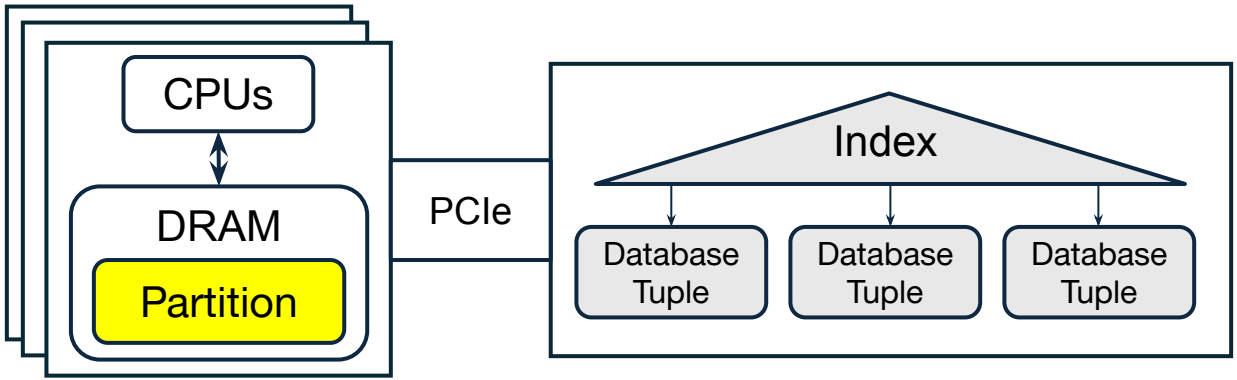


Takeaway 2: Must minimize hardware cache coherence usage

[1] Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices, *MICRO 2023*
[2] Memory Sharing with CXL: Hardware and Software Design Approaches, *HCDS 2024*

Tigon[1]: The First Transactional DB for a CXL Pod

Pasha Architecture[2]: Partitioned and Shared

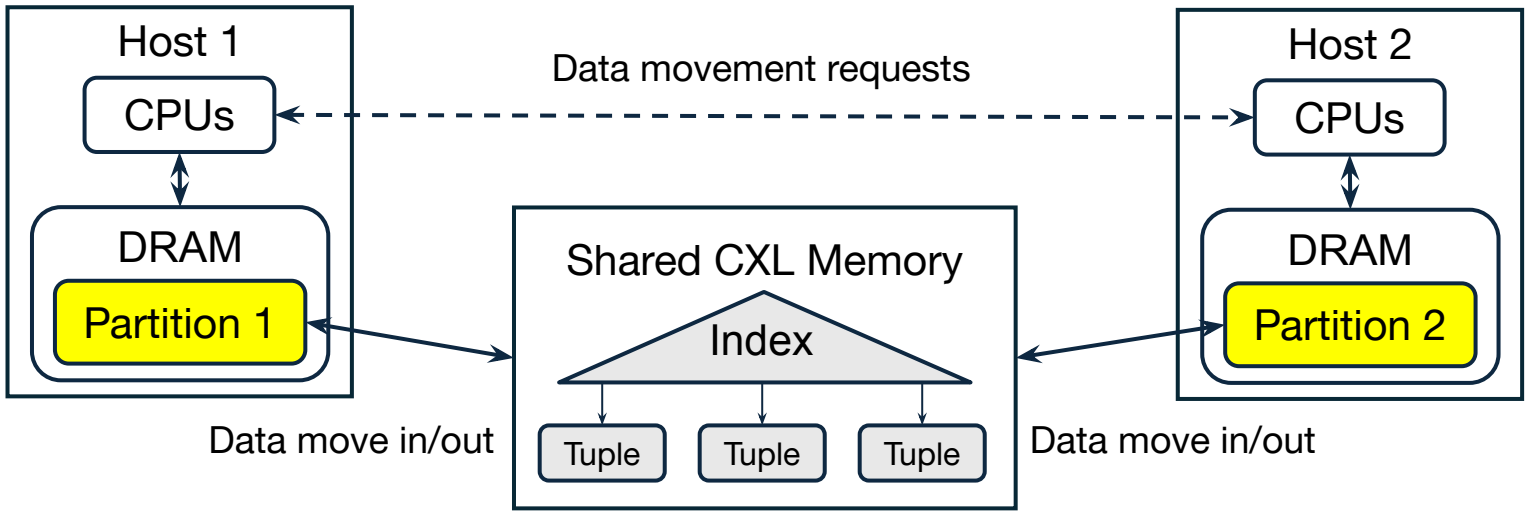


Idea 1: **Partition** the data and store them in local DRAM to leverage its high performance

Idea 2: **Share only** the data that will be accessed by multiple hosts in CXL to avoid message exchange

[1] A tigon is a hybrid of a male tiger and a female lion
[2] Pasha: An Efficient, Scalable Database Architecture for CXL Pods, *CIDR 2025*

Tigon: The First Transactional DB for a CXL Pod

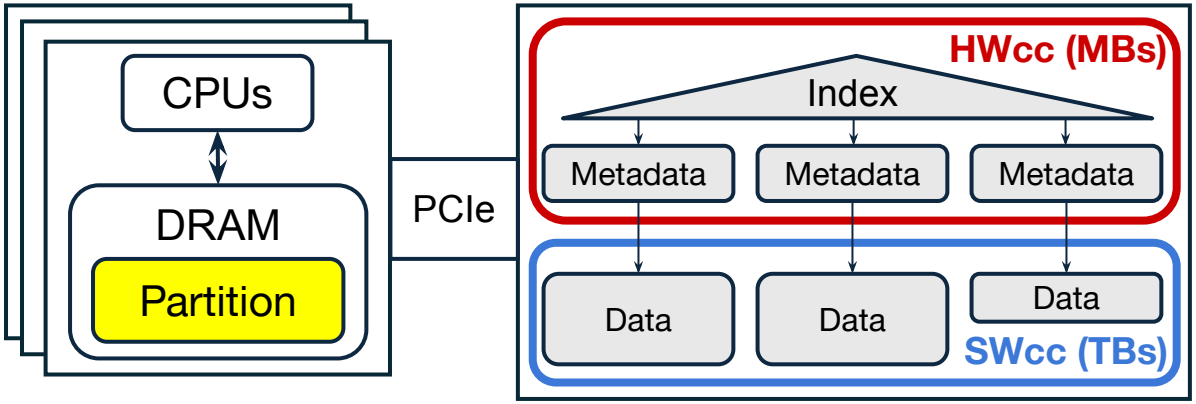


- **Initially** partition the data and store them in local DRAM
- Move data to CXL memory **upon the request of non-owner hosts**
  - Messages only exchanged for data movement
  - No two phase commit
- Move data back to its original partition when CXL memory is full

Reducing Data Movement Frequency

Problem: Oversubscribed HWcc memory incurs frequent data movement

Solution: Store data in the non-HWcc region and enable cacheable access using a Software Cache Coherence (SWcc) protocol



Key idea 1: Compact **sync-heavy** and **SWcc** metadata into **8 bytes** and store them in **HWcc**

Key idea 2: Use metadata in **HWcc** to enable **SWcc**

5× higher YCSB throughput compared with using HWcc only

Avoiding Two-Phase Commit (2PC)

Problem: How to let each host **EXECUTE** and **LOG** all transaction operations on its own?

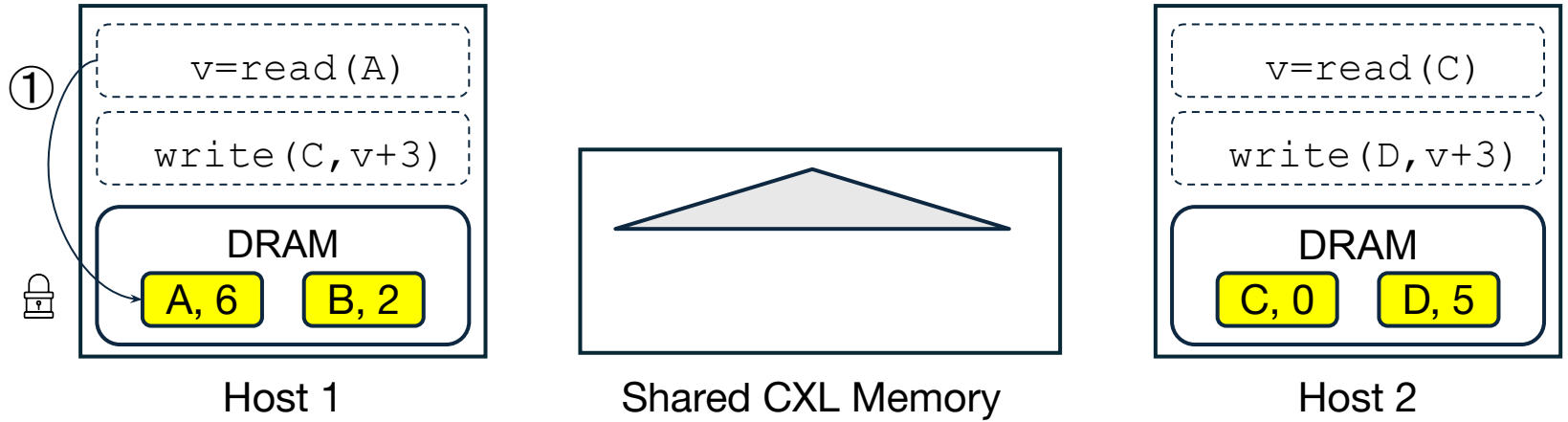
- Tuple modifications
- Index operations
- Data movements

Insight: DB internal state modifications can be reconstructed or discarded upon recovery - logging not needed

Solution:

- Let the remote host move data – each host can **execute all tuple modifications**
- Adapt value logging that **logs only tuple modifications**
- **Reconstruct** index and data movement metadata upon recovery

Example Transaction Workflow



① *Txn1* locks *A* and reads it (*A* = *6*)

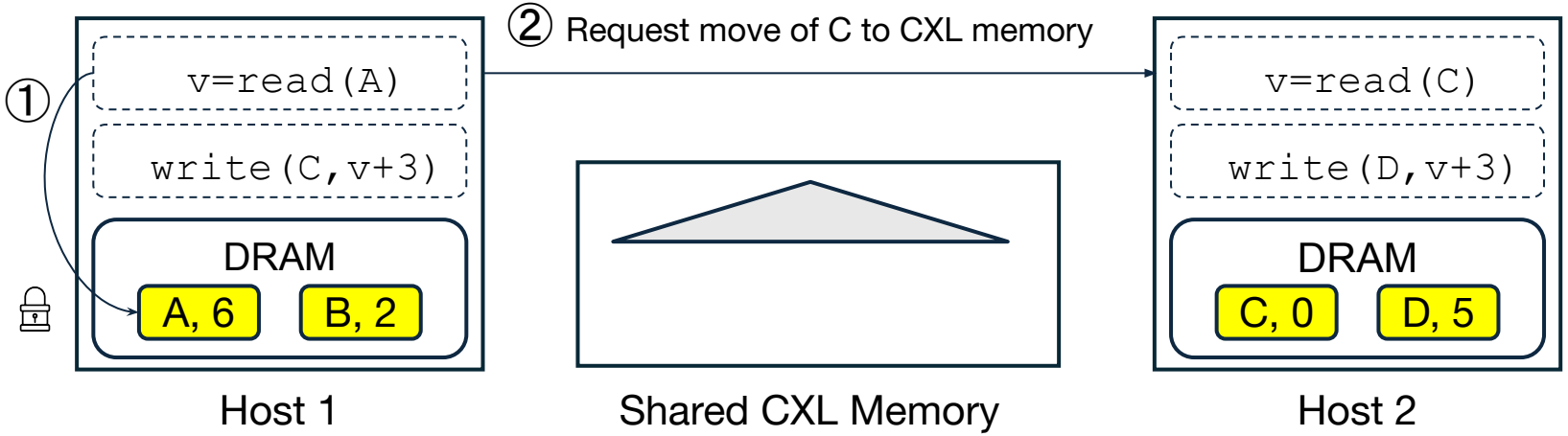
② *Txn1* messages to *Host 2* about *C*

③ *Host 2* moves *C* to CXL memory

④ *Txn1* locks *C* and writes it (*C* = *9*)

⑤ *Txn2* read of *C* is denied

Example Transaction Workflow



① *Txn1* locks *A* and reads it (*A* = *6*)

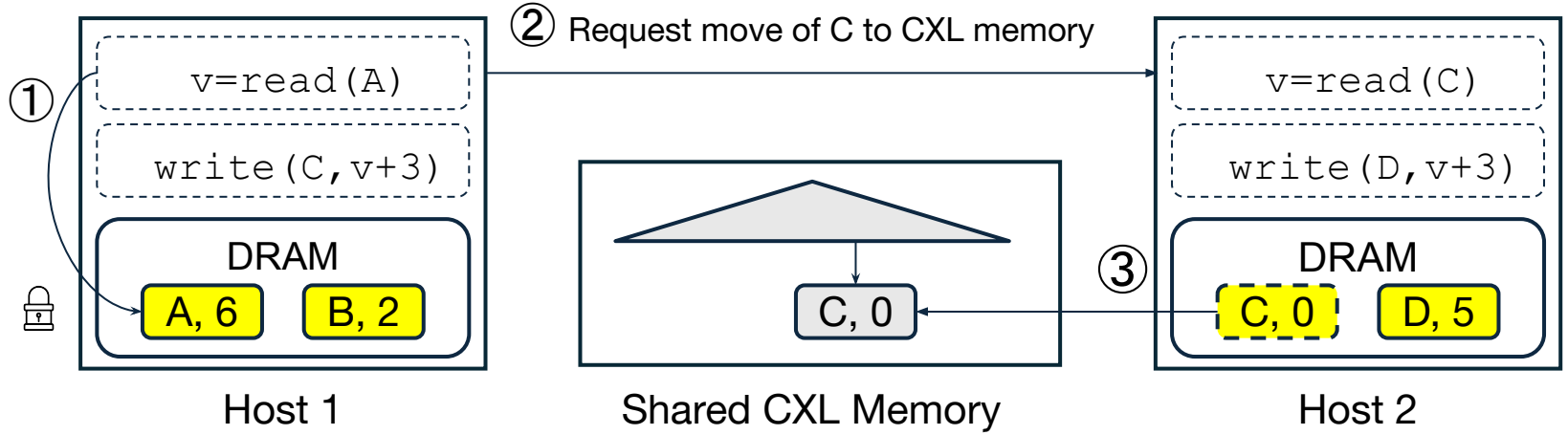
② *Txn1* messages to *Host 2* about *C*

③ *Host 2* moves *C* to CXL memory

④ *Txn1* locks *C* and writes it (*C* = *9*)

⑤ *Txn2* read of *C* is denied

Example Transaction Workflow



① *Txn1* locks *A* and reads it (*A* = *6*)

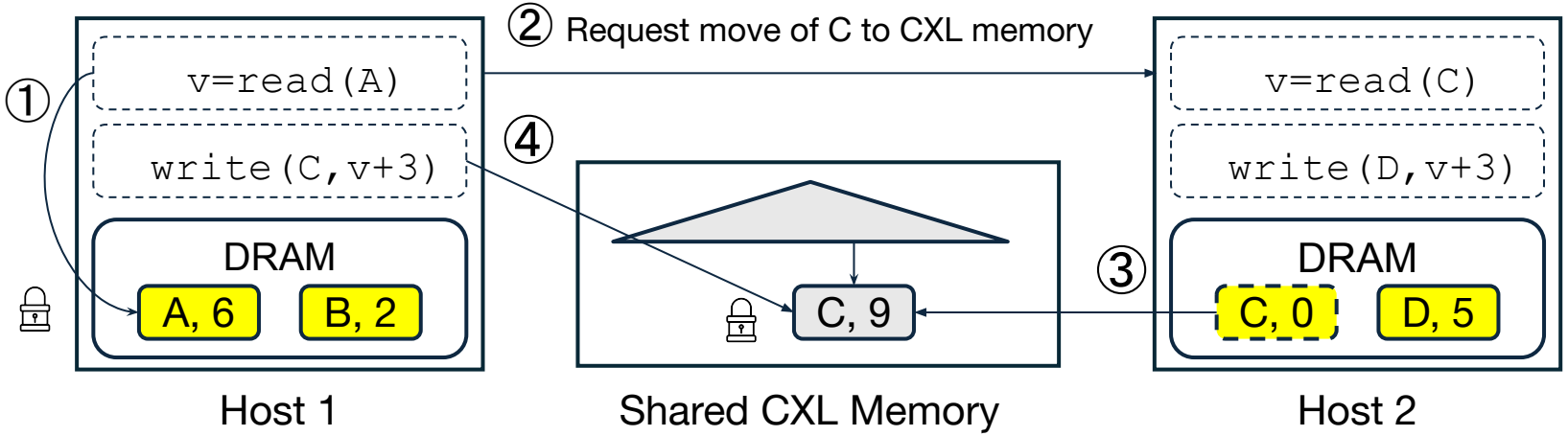
② *Txn1* messages to *Host 2* about *C*

③ *Host 2* moves *C* to CXL memory

④ *Txn1* locks *C* and writes it (*C* = *9*)

⑤ *Txn2* read of *C* is denied

Example Transaction Workflow



① *Txn1* locks *A* and reads it (*A* = *6*)

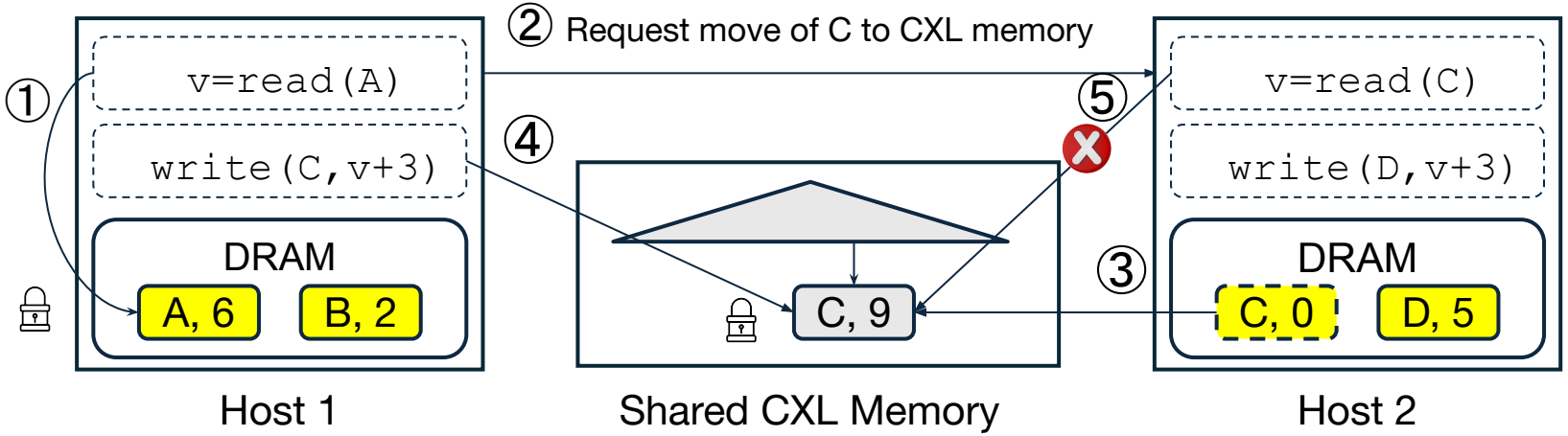
② *Txn1* messages to *Host 2* about *C*

③ *Host 2* moves *C* to CXL memory

④ *Txn1* locks *C* and writes it (*C* = *9*)

⑤ *Txn2* read of *C* is denied

Example Transaction Workflow



① *Txn1* locks *A* and reads it (*A* = *6*)

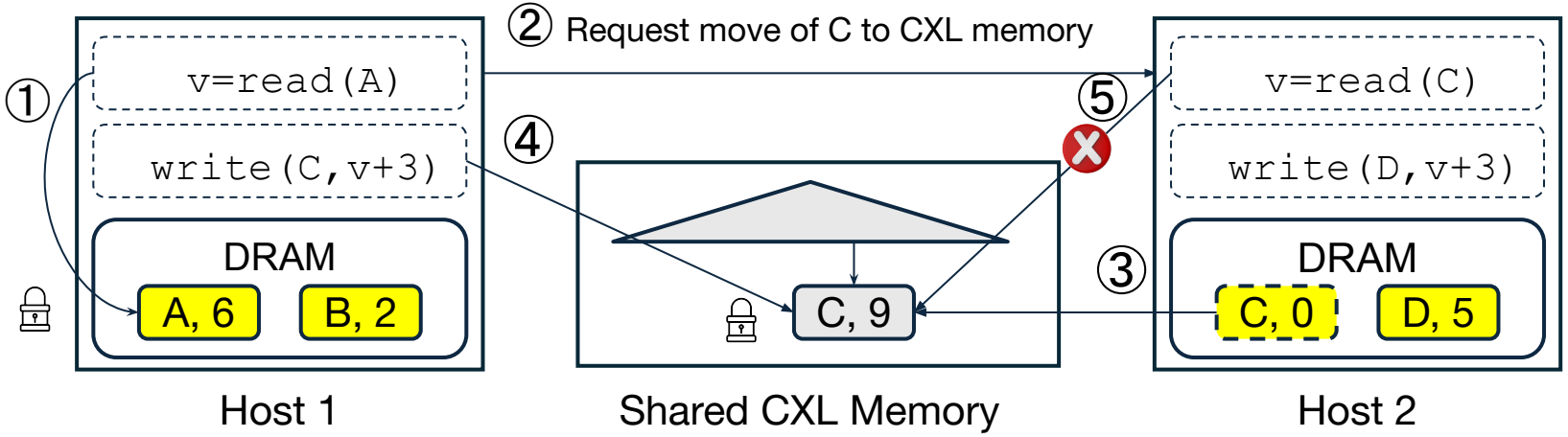
② *Txn1* messages to *Host 2* about *C*

③ *Host 2* moves *C* to CXL memory

④ *Txn1* locks *C* and writes it (*C* = *9*)

⑤ *Txn2* read of *C* is denied

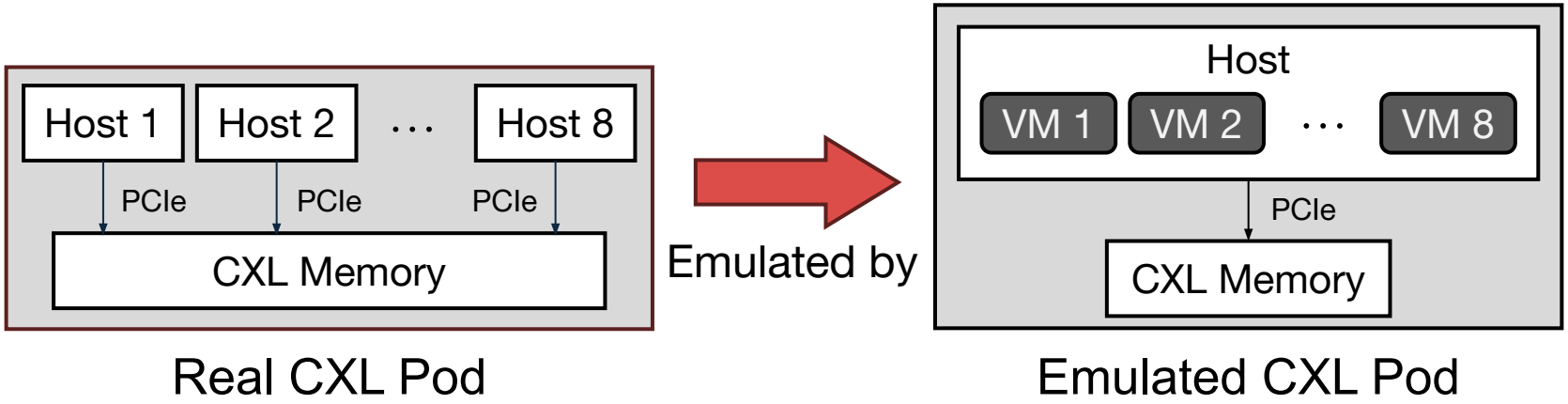
Example Transaction Workflow



Takeaway 1: Each host can execute all tuple modifications for a single transaction, avoiding two-phase commit

Takeaway 2: Tigon maintains only data that is accessed by multiple hosts in shared CXL memory, minimizing HWcc

Evaluation: CXL Pod Emulation



Real CXL Pod

Emulated CXL Pod

- Not commercially available

- 40-core physical machine
- CXL 1.1 memory module
- Run 8 VMs each with 5 cores

Evaluation: Baselines

Existing partition-based distributed in-memory DB

- Sundial[1] - optimistic concurrency control (OCC)
- DS2PL - pessimistic concurrency control (2PL)

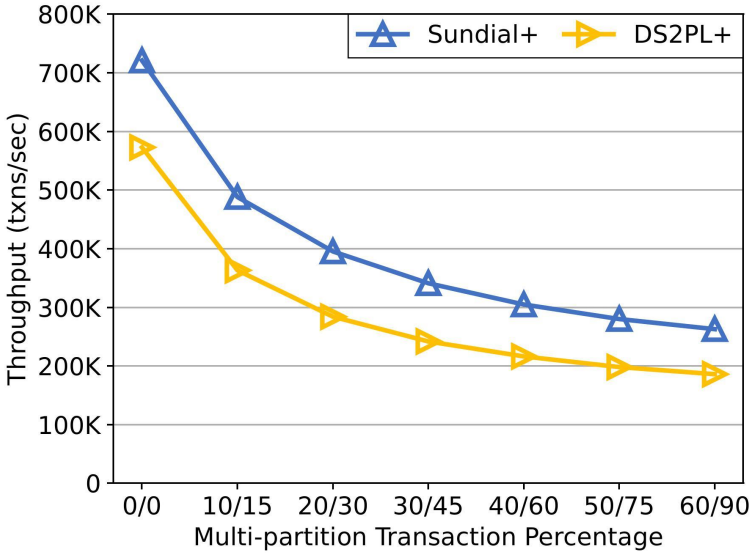
Improved baselines

- Improvements
  1. Replace network transport with CXL message queues
  2. Repurpose a network thread for transaction execution
- Improved baselines
  - **Sundial+** - Sundial with improvements 1 & 2
  - **DS2PL+** - DS2PL with improvements 1 & 2

RDMA-based shared-memory DB - **Motor[2]**

[1] Sundial: Harmonizing Concurrency Control and Caching in a Distributed OLTP Database Management System, *VLDB 2018*
[2] Motor: Enabling Multi-Versioning for Distributed Transactions on Disaggregated Memory, *OSDI 2024*

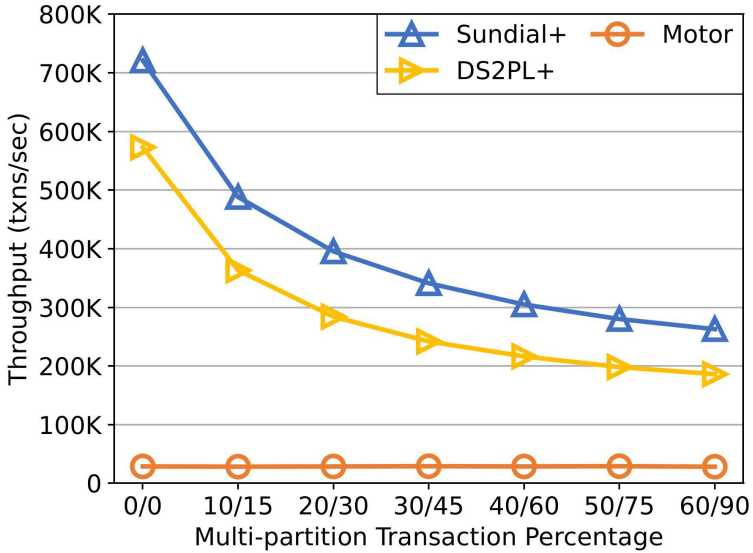
Evaluation: TPC-C



TPC-C Throughput (10/15: 10% NewOrder and 15% Payment transactions are remote)

- Sundial+ and DS2PL+ (our improved baselines) suffer from the overhead of message exchanges

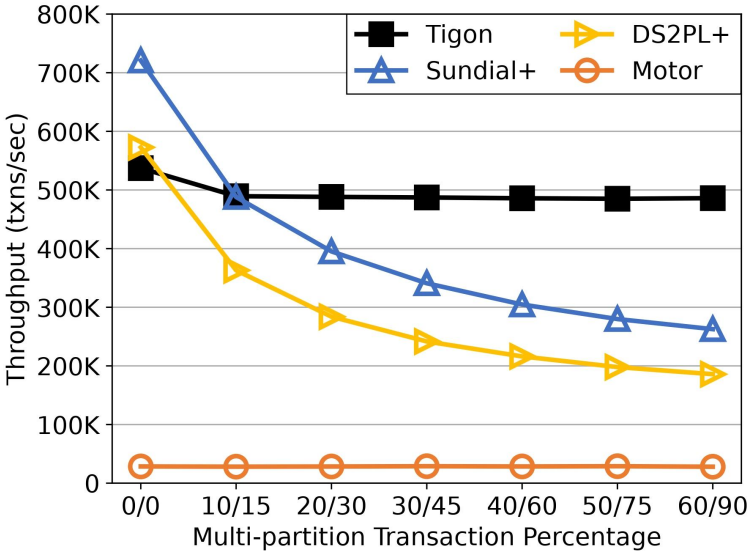
Evaluation: TPC-C



TPC-C Throughput (10/15: 10% NewOrder and 15% Payment transactions are remote)

- Sundial+ and DS2PL+ (our improved baselines) suffer from the overhead of message exchanges
- Motor suffers from the overhead of high-latency RDMA operations

Evaluation: TPC-C

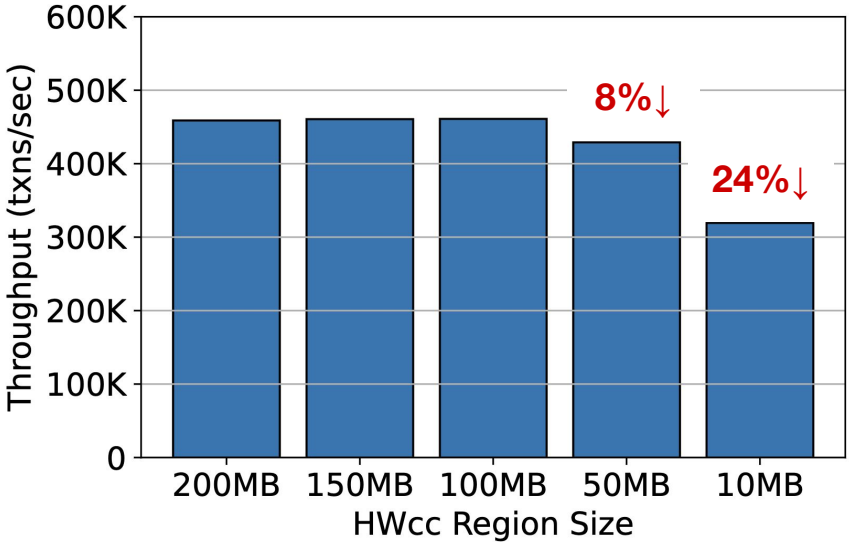


TPC-C Throughput (60/90: 60% NewOrder and 90% Payment transactions are remote)

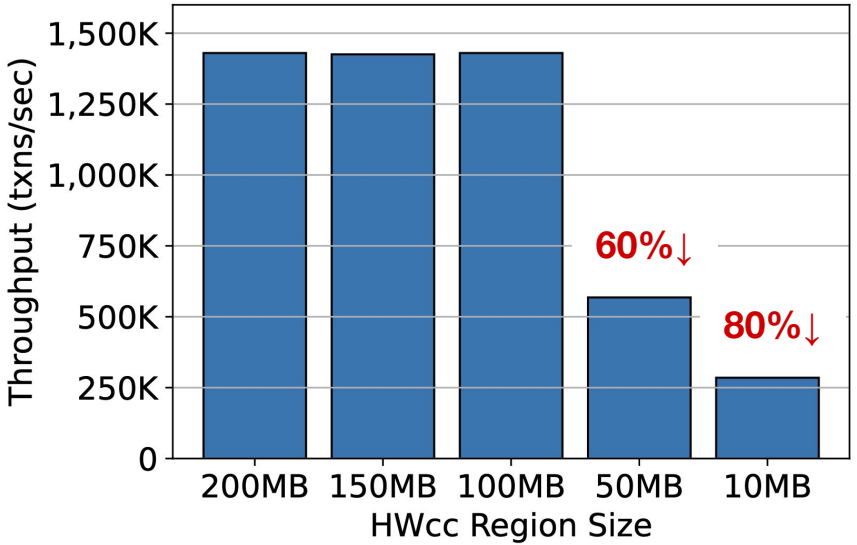
- Sundial+ and DS2PL+ (our improved baselines) suffer from the overhead of message exchanges
- Motor suffers from the overhead of high-latency RDMA operations
- Tigon
  - At most 75% faster than Sundial+
  - At most 2.4× faster than DS2PL+
  - 11.9×-14.4× faster than Motor

Evaluation: Varying HWcc Budgets

TPC-C with
60/90 remote transaction ratio



YCSB (95%R/5%W) with
100% remote transaction ratio



- TPCC 60/90: Drop within 8% for 50-150MB, 24% for 10MB
- YCSB 100%: Drop by 60% for 50MB, 80% for 10MB

This Talk

A new direction for building distributed databases:

- *Instead of doing distributed synchronization over a network, let's do it **over CXL memory**, utilizing the emerging CXL technology*

Tigon is the first distributed transactional database for a CXL pod

- Adopts the **Pasha** (partitioned and shared) architecture
- Utilizes the non-HWcc region using **software cache coherence**
- Avoids 2PC by adopting **value logging** and **reconstructing** internal state upon recovery

github.com/ut-datasys/tigon
ybhuang@cs.utexas.edu