CS 839: Design the Next-Generation Database

Lecture 16: High Availability

Xiangyao Yu

3/12/2020

Announcements

Online lectures

- Until at least April 10

Canvas (canvas.wisc.edu)

–> Courses

–> COMPSCI839: Core Topics in Computing (005) SP20

–> BBCollaborate Ultra

Learning to Use BBCollaborate Ultra

Blackboard Collaborate Ultra

- Mute your audio by default
- Raise hand
- Breakout groups

Discussion Highlights

RDMA for transaction execution phase

- No need to partition or replicate indexes
- Centralized locking and replication can be faster
- Centralized logging service
- Accessing and locking remote data using RDMA
- Prefetch data into local memory

DB components significantly affected by a faster network

- Two phase commit, Consensus, Replication
- Less worry about locality
- Breakdown program into multiple microservices
- Data shuffling in the network
- Distributed system becomes NUMA is network is sufficiently fast

Opportunities and challenges of memory disaggregation

- Opportunities: Independent scaling of CPU and memory; simplifying scheduling; larger aggregated memory capacity; potentially faster fault tolerance
- Challenges: independent failure of compute and memory; consistency and coherency; data placement and partitioning

Today's Paper

Rethinking Database High Availability with RDMA Networks

Erfan Zamanian[1], Xiangyao Yu[2], Michael Stonebraker[2], Tim Kraska[2]
[1] Brown University [2] Massachusetts Institute of Technology
erfanz@cs.brown.edu, {yxy, stonebraker, kraska}@csail.mit.edu

ABSTRACT

Highly available database systems rely on data replication to tolerate machine failures. Both classes of existing replication algorithms, active-passive and active-active, were designed in a time when network was the dominant performance bottleneck. In essence, these techniques aim to minimize network communication between replicas at the cost of incurring more processing redundancy; a trade-off that suitably fitted the conventional wisdom of distributed copy propagate to all the backup copies synchronously such that any failed primary server can be replaced by a backup server.

The conventional wisdom of distributed system design is that the network is a severe performance bottleneck. Messaging over a conventional 10-Gigabit Ethernet within the same data center, for example, delivers 2–3 orders of magnitude higher latency and lower bandwidth compared to accessing the local main memory of a server [3]. Two dominant high availability approaches, *active-*

VLDB 2019

Key Idea

Common wisdom: **Network** is a severe **performance bottleneck** in a distributed database

Key Idea

Common wisdom: **Network** is a severe **performance bottleneck** in a distributed database

No longer true for the **next-generation high bandwidth RDMA-enabled** network

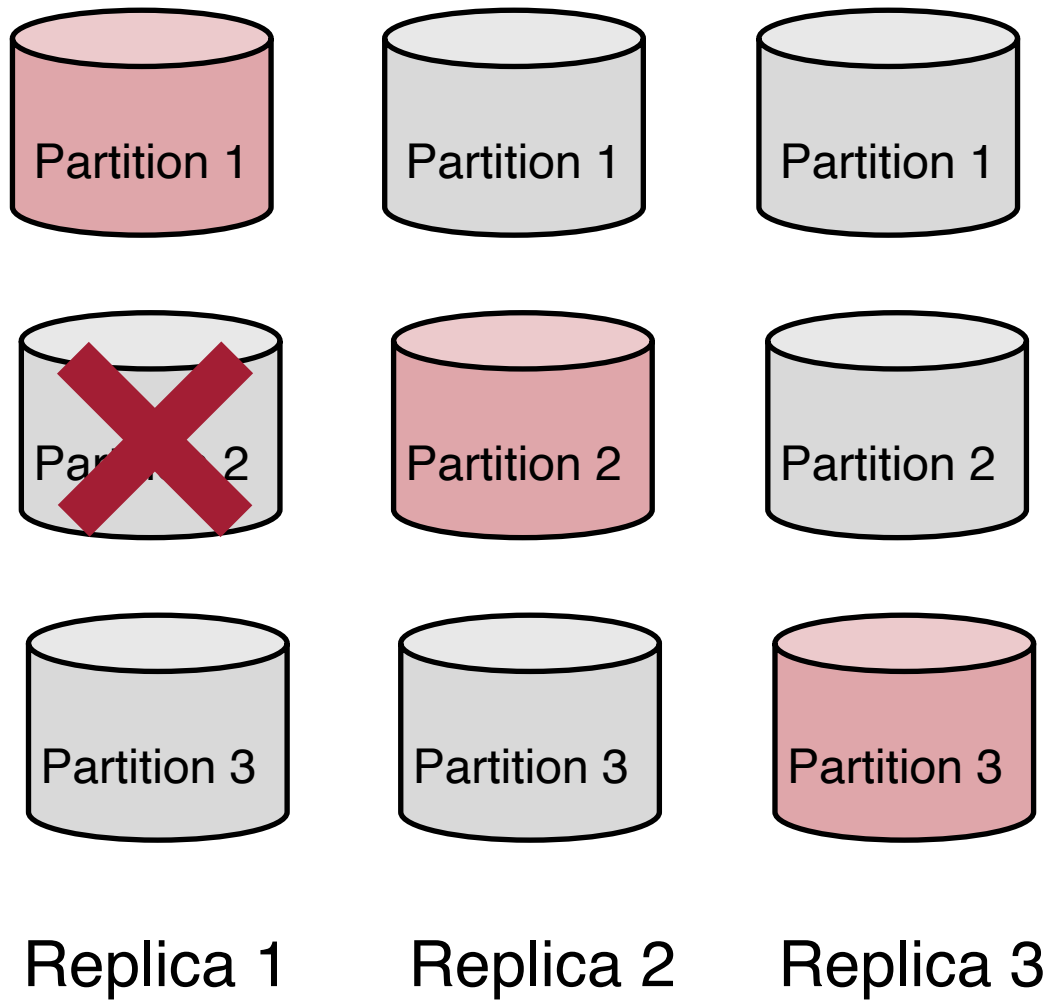
Key Idea

Common wisdom: **Network** is a severe **performance bottleneck** in a distributed database

No longer true for the **next-generation high bandwidth RDMA-enabled** network

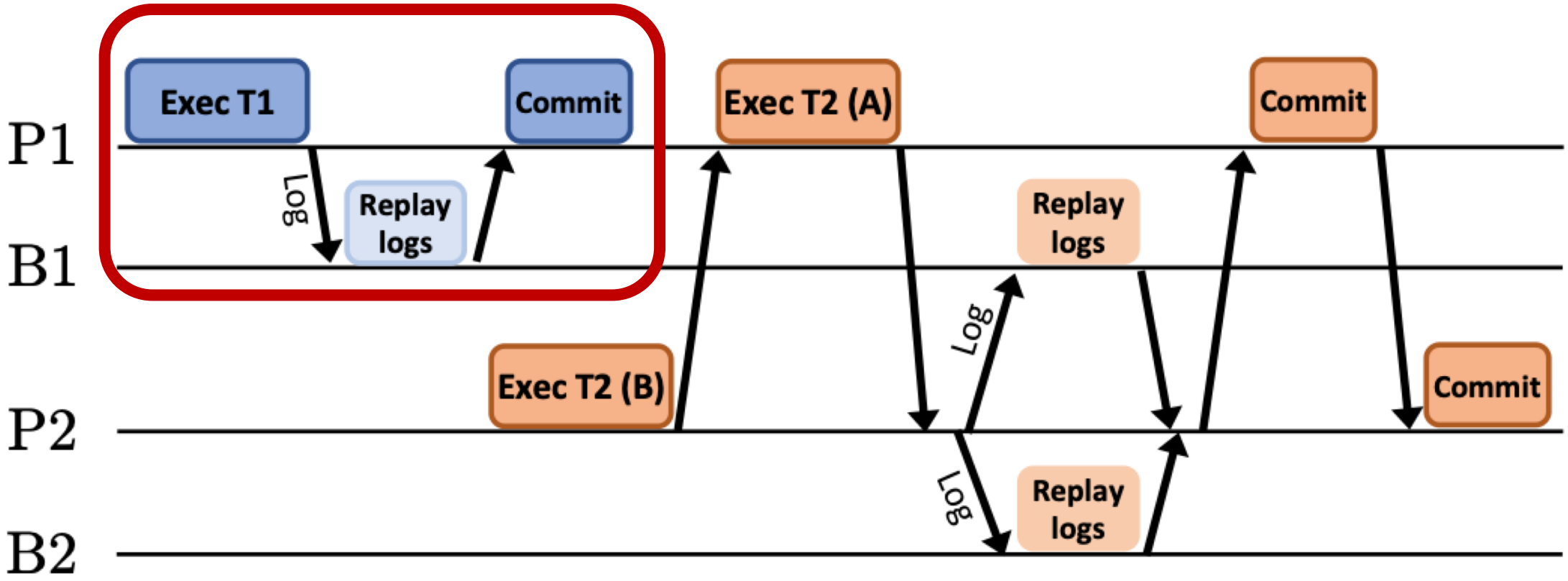
This paper: **Rethinking high-availability** (HA) protocols in the context of RDMA-based network

High Availability



- Replicate data across multiple servers
- Data is available if at least one partition is still alive
- If the primary node fails, failure over to a secondary node

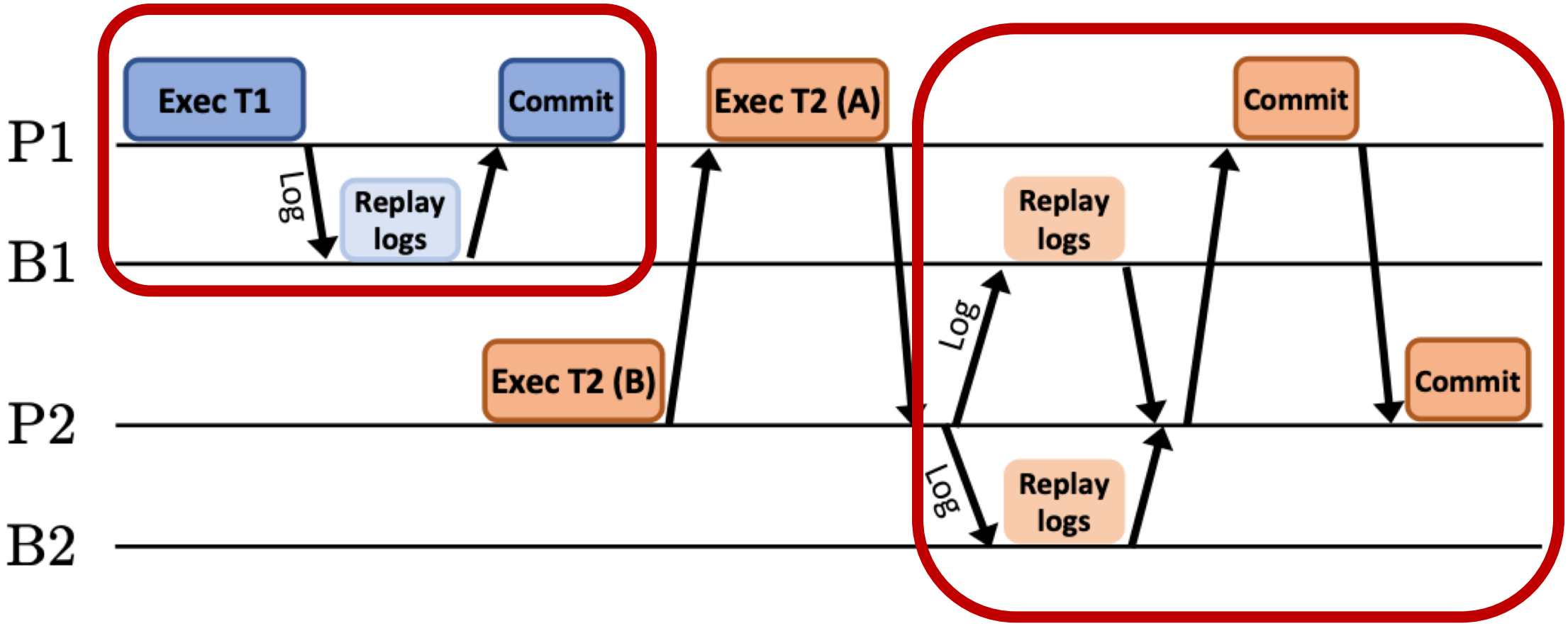
Active Passive



Log shipping: primary nodes send log to backup nodes.

- Network traffic for log
- CPU cycles for log replay

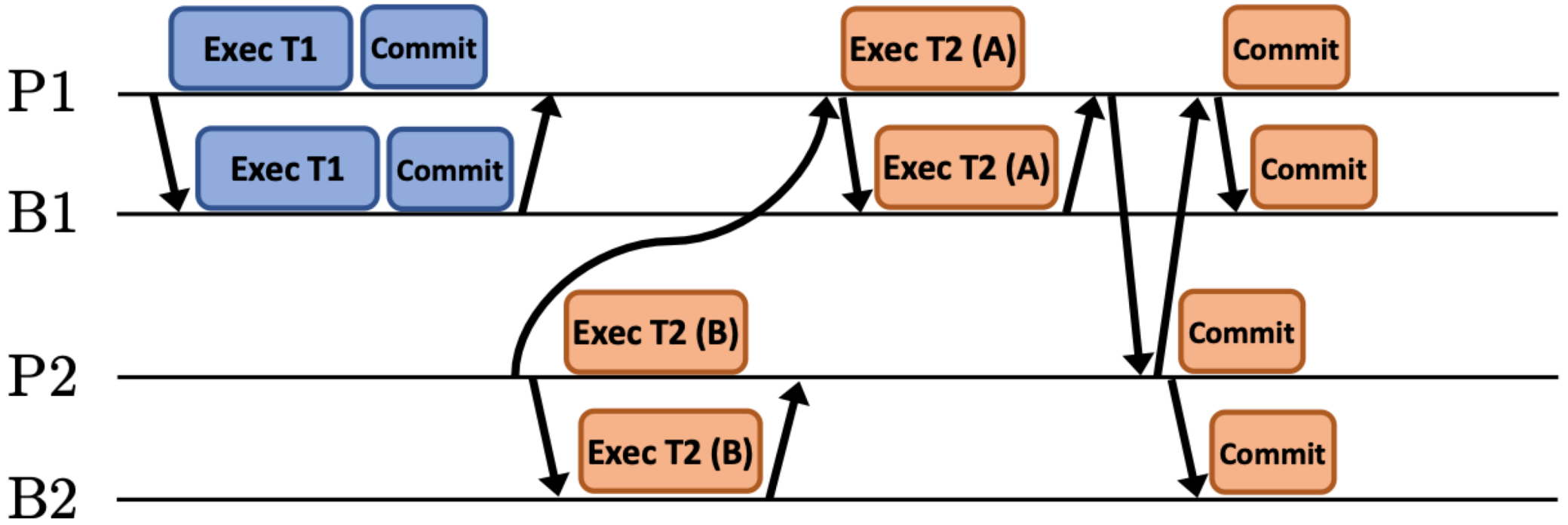
Active Passive



Log shipping: primary nodes send log to backup nodes.

- Network traffic for log
- CPU cycles for log replay

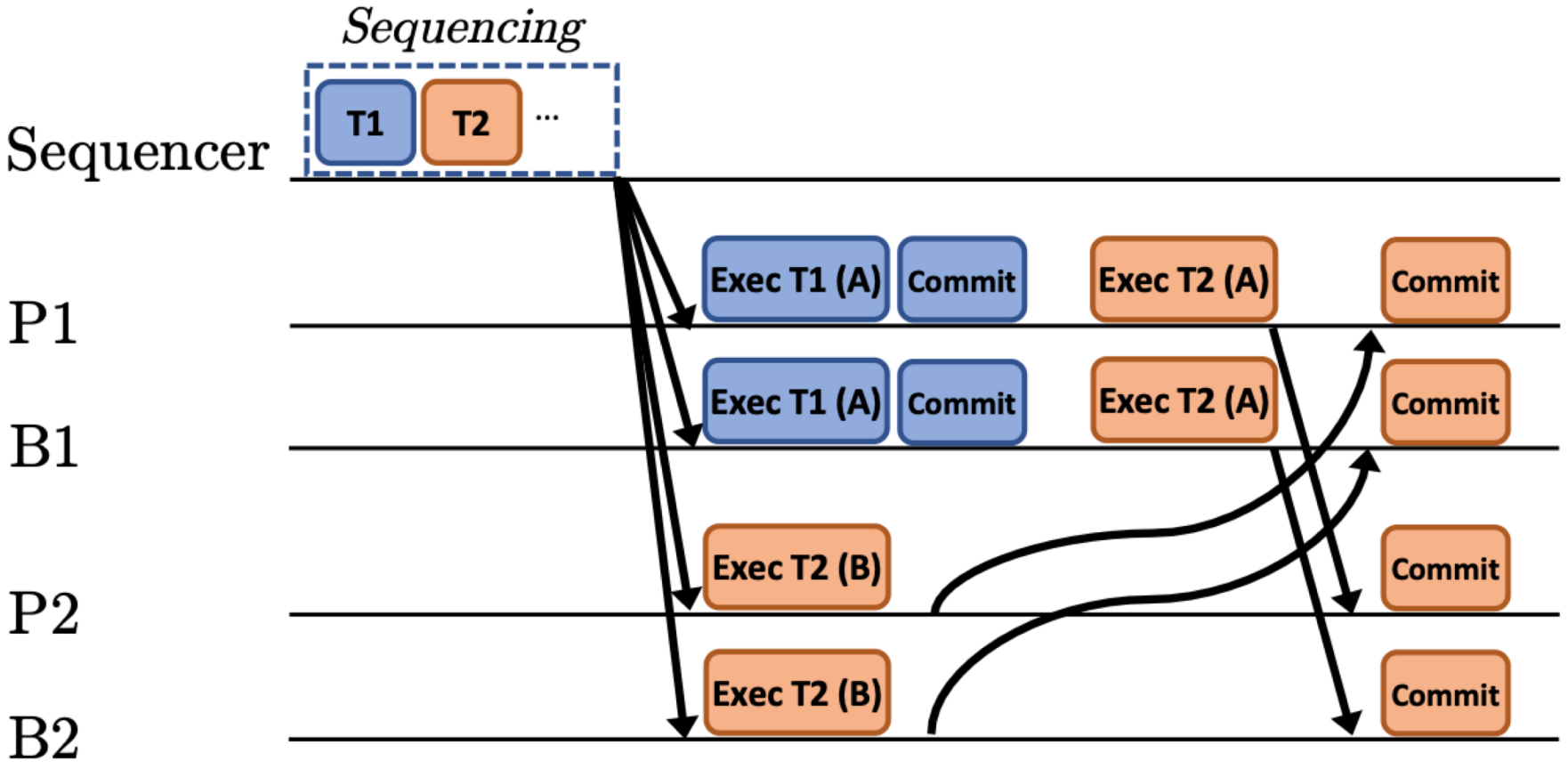
Active Active (H-Store/VoltDB)



Partition-based locking

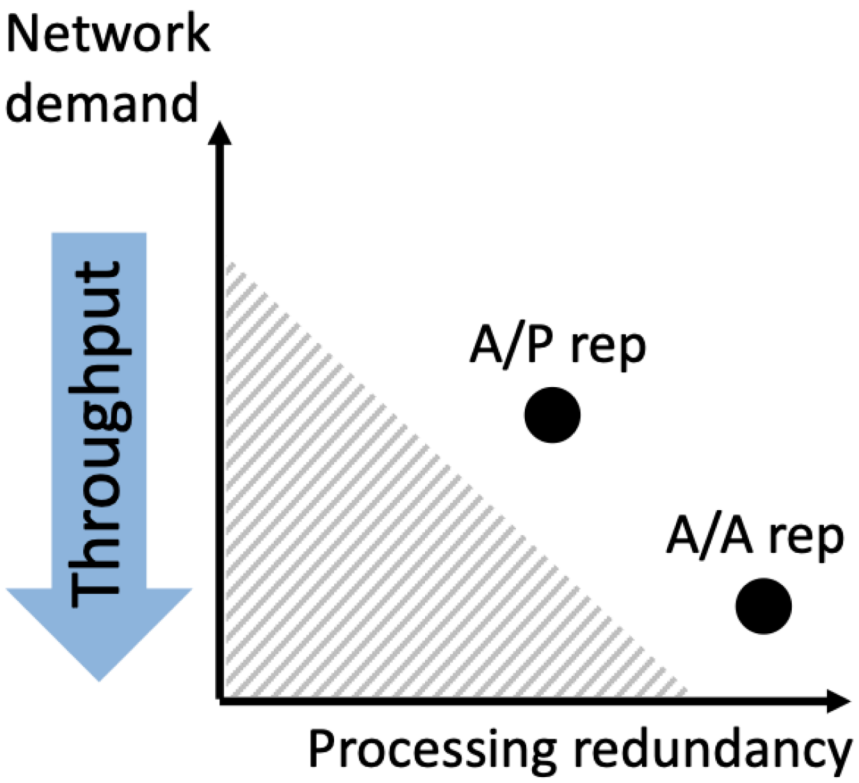
- After locking a partition, primary and backup run the same code
- No concurrency within a partition

Active Active (Calvin)



Deterministic execution following pre-assigned sequential order

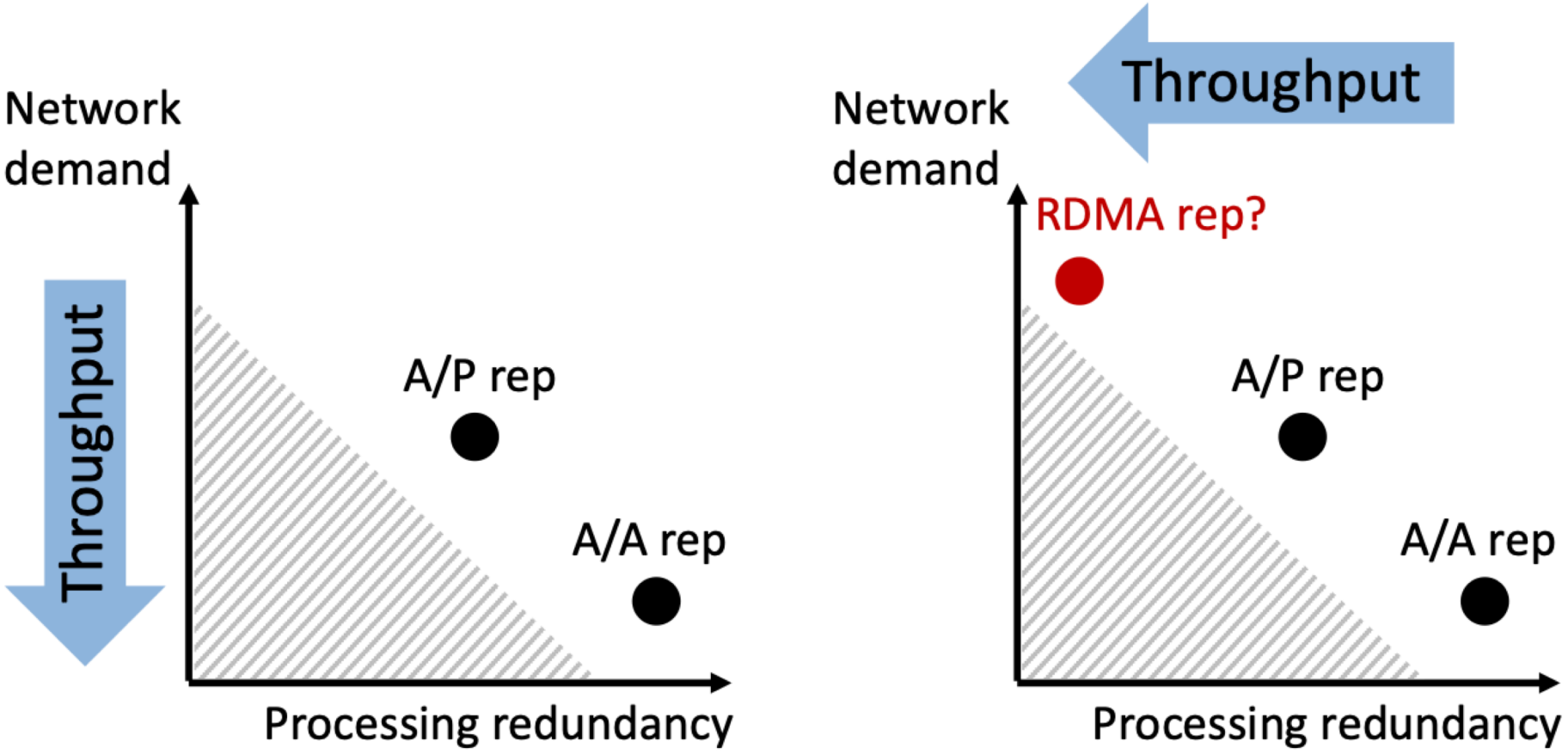
Active-Active vs. Active-Passive



Network is the bottleneck in a conventional distributed database

Performance improves when network activity decreases

Active/Active & Active/Passive



CPU is the bottleneck in an RDMA-based distributed database

Performance improves when CPU activity decreases

Bottleneck Shifts from Network to CPU

Old optimization goal: **Reduce network demand**

New optimization goal: **Reduce CPU demand**

CPU demand

Network demand

Bottleneck Shifts from Network to CPU

	Active-Passive		Active-Active
CPU Demand	**Replay logs**	**<**	**Duplicate execution**
Network Demand	**Send logs**	**>**	**Send input**

CPU demand

Network demand

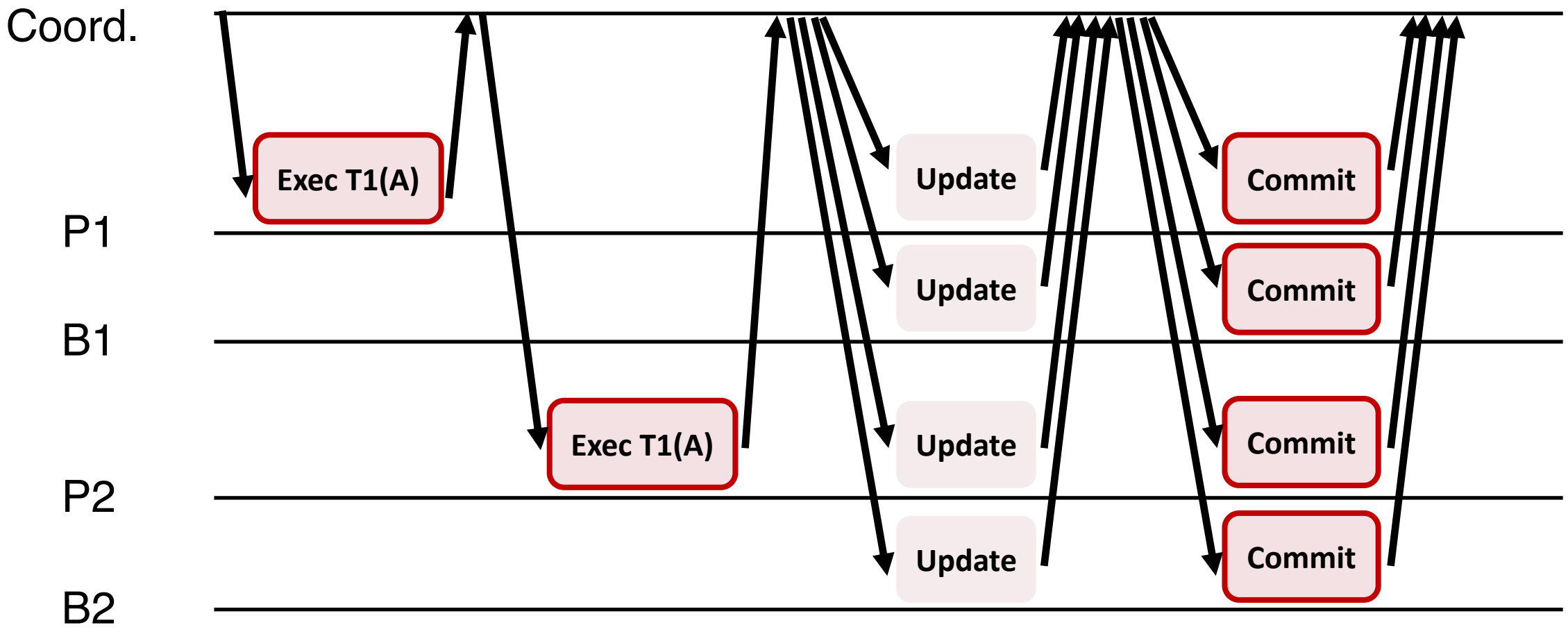
Bottleneck Shifts from Network to CPU

Key idea: Coordinator **directly updates** memory states of backup nodes using **one-sided RDMA**

CPU demand

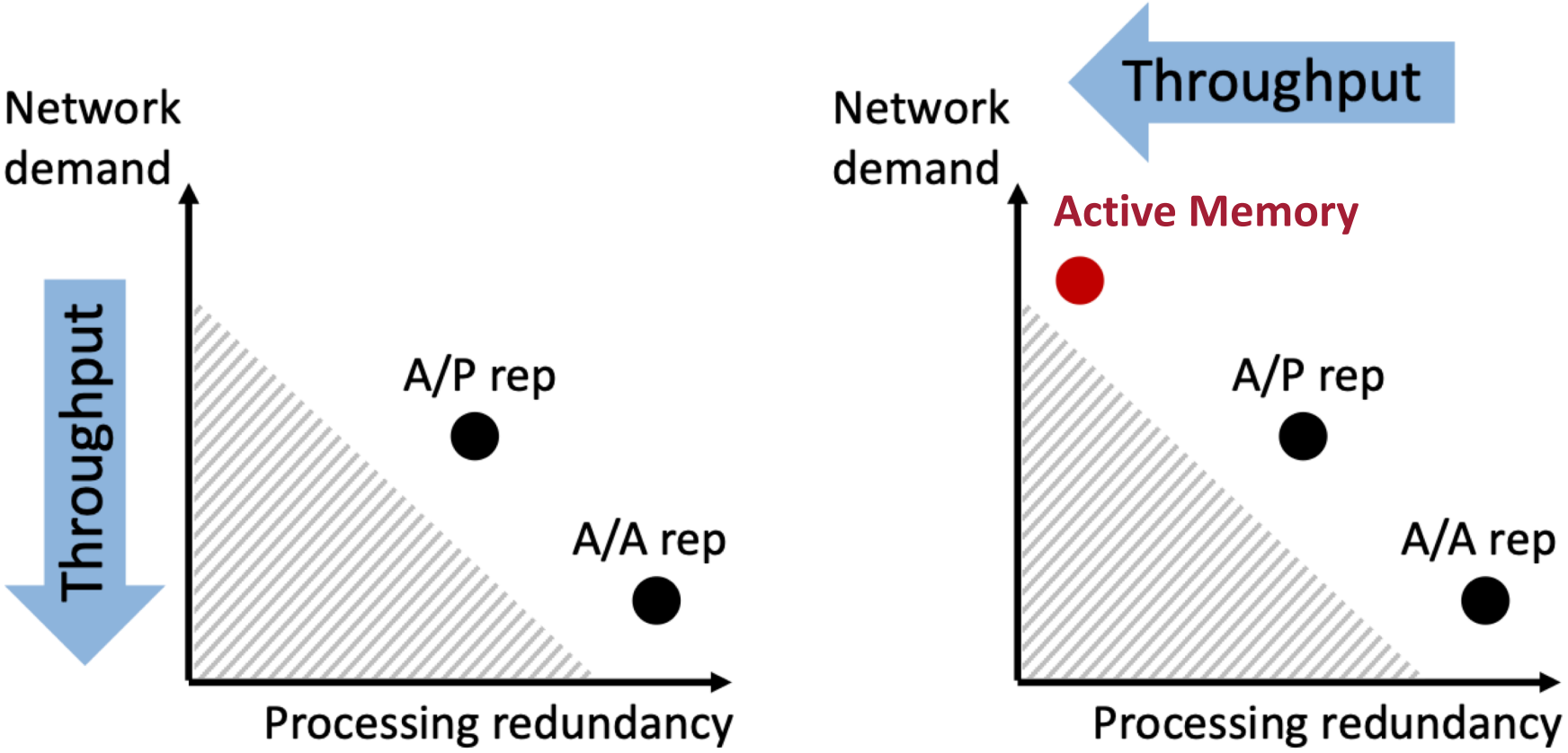
Network demand

Active Memory



Coordinator directly updates memory states of all nodes using one-sided RDMA

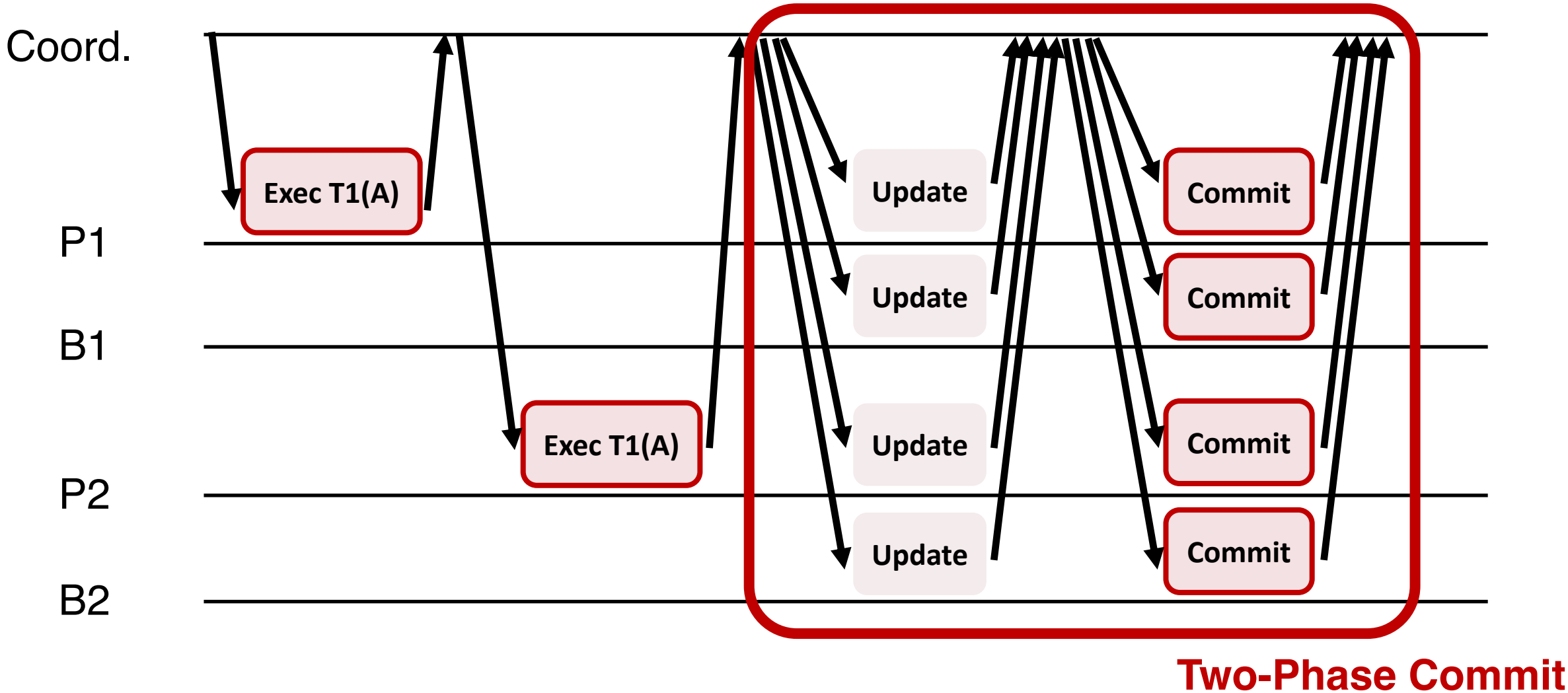
Active Memory



Less CPU consumption: no log replay, no redundant execution

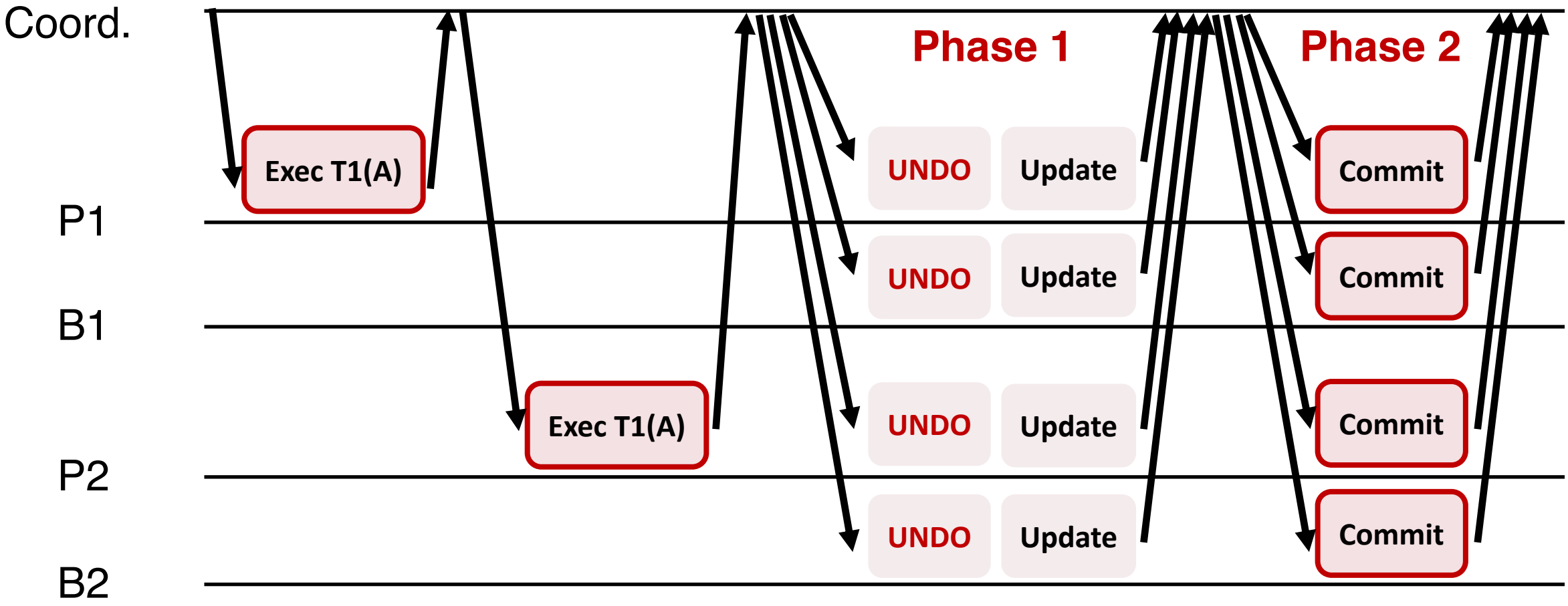
More network traffic: direct updates consume higher bandwidth than log shipping or inputs replication

Challenge: Fault Tolerance



Coordinator must unilaterally guarantee fault tolerance properties

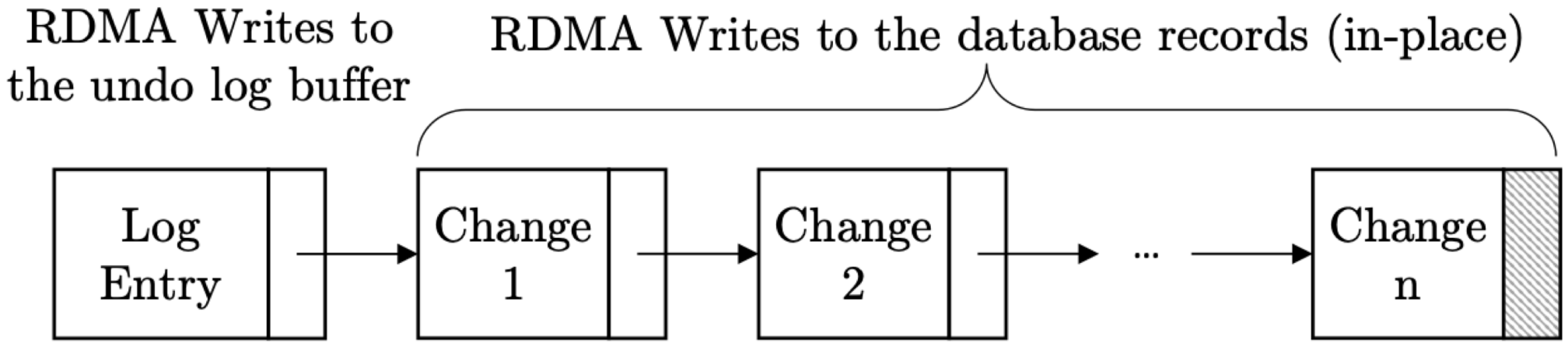
Undo Logging in Active Memory



Write UNDO entry before directly updating memory

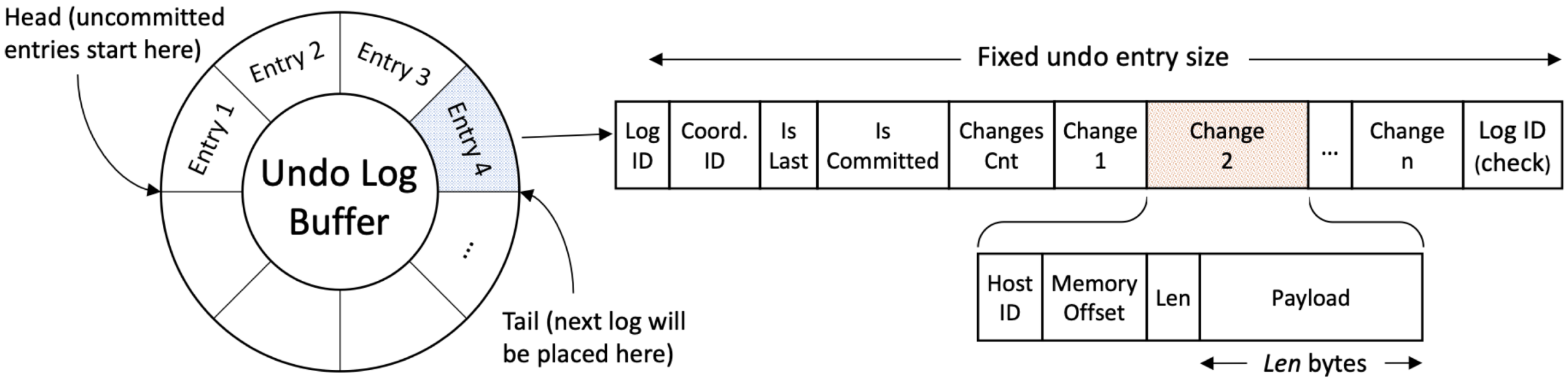
Set the commit bit in UNDO log entry to 1

Phase1 Message Format



RDMA guarantees in-order processing of network messages

Undo Log Buffer



- A dedicated Undo log buffer for each remote node
- Each Undo Log Buffer has a fixed number of entries
- An entry is reclaimed after the transaction commits

Fault Tolerance

Need to handle only transactions in the UNDO buffer

- Otherwise the transaction must have already committed

If primary replica fails, promote a backup to be the new primary

All nodes broadcast UNDO buffers

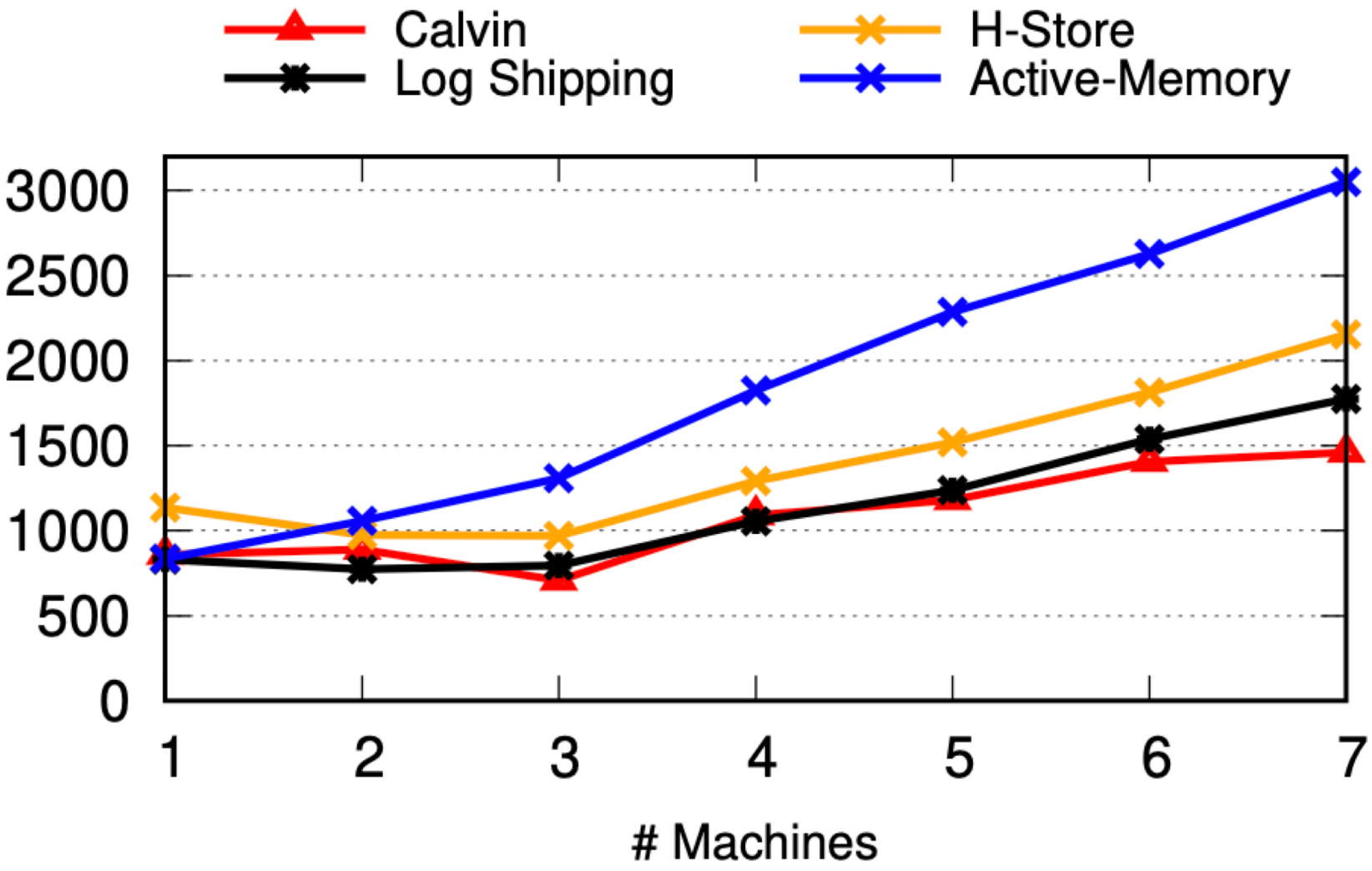
If a transaction T has commit bit set in all UNDO buffers

-> commit the transaction

Otherwise

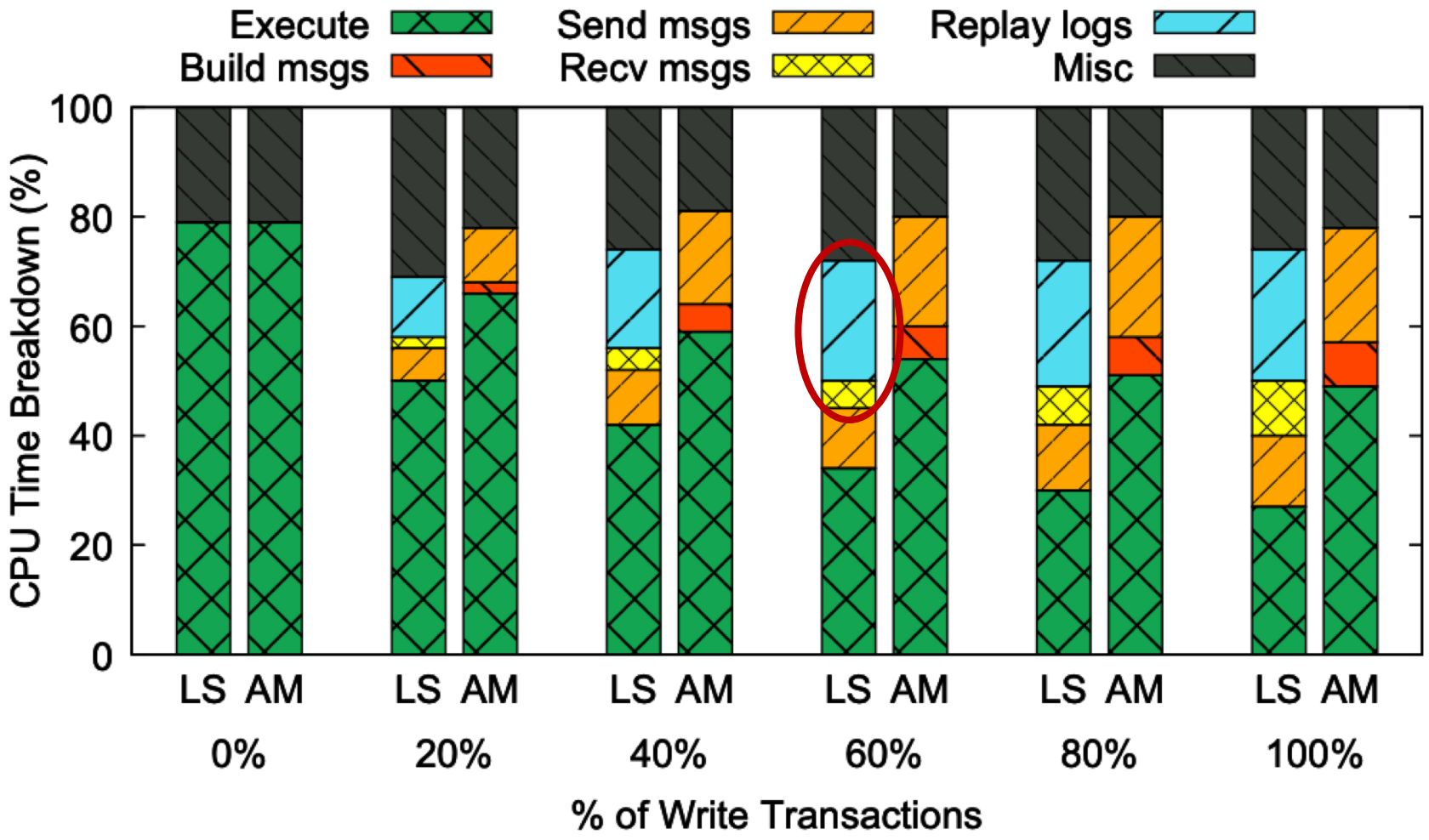
-> rollback and abort the transaction

Evaluation – Scalability



Evaluation – Scalability

LS: Log Shipping **AM**: Active Memory

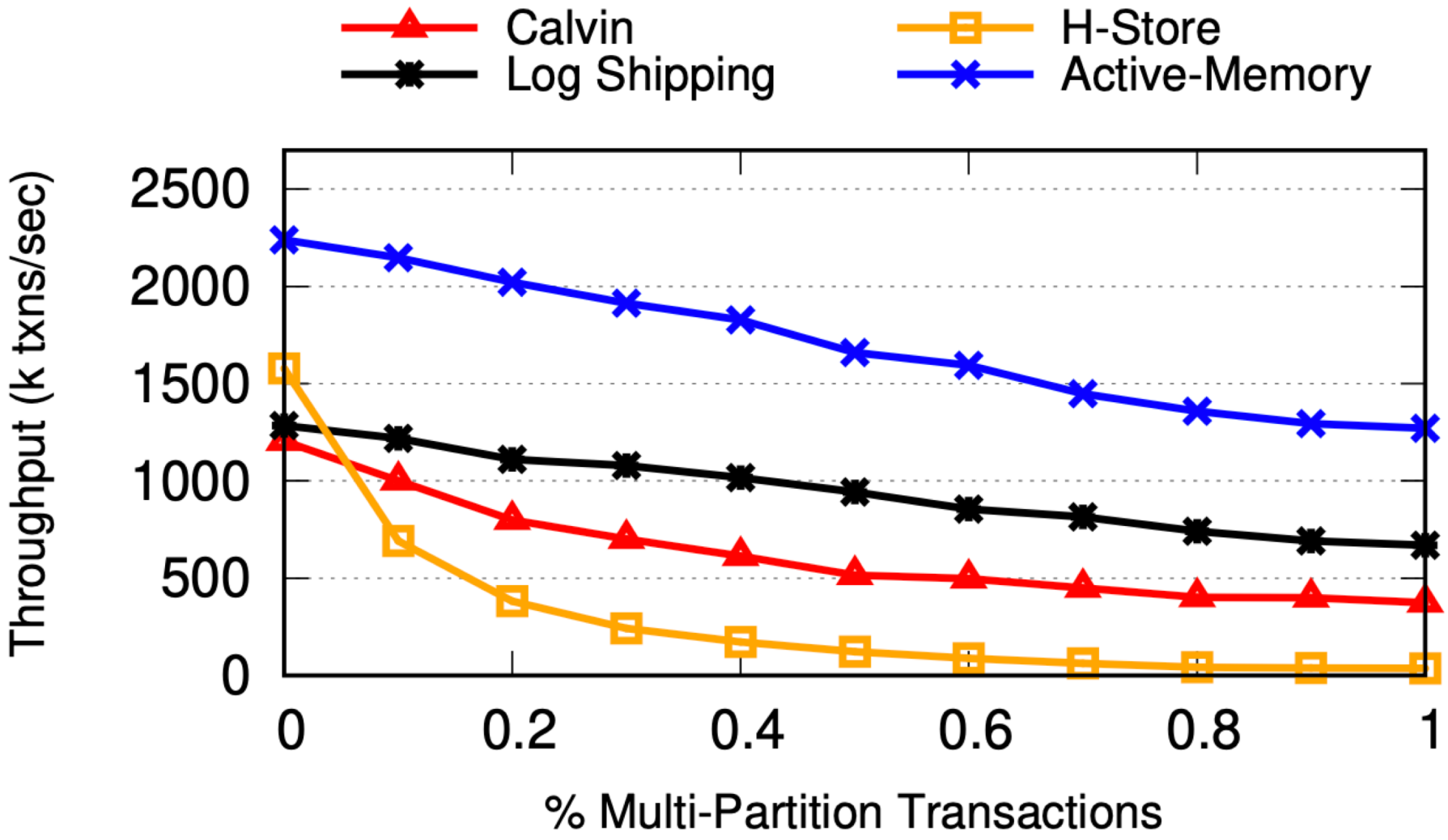


Evaluation – Latency

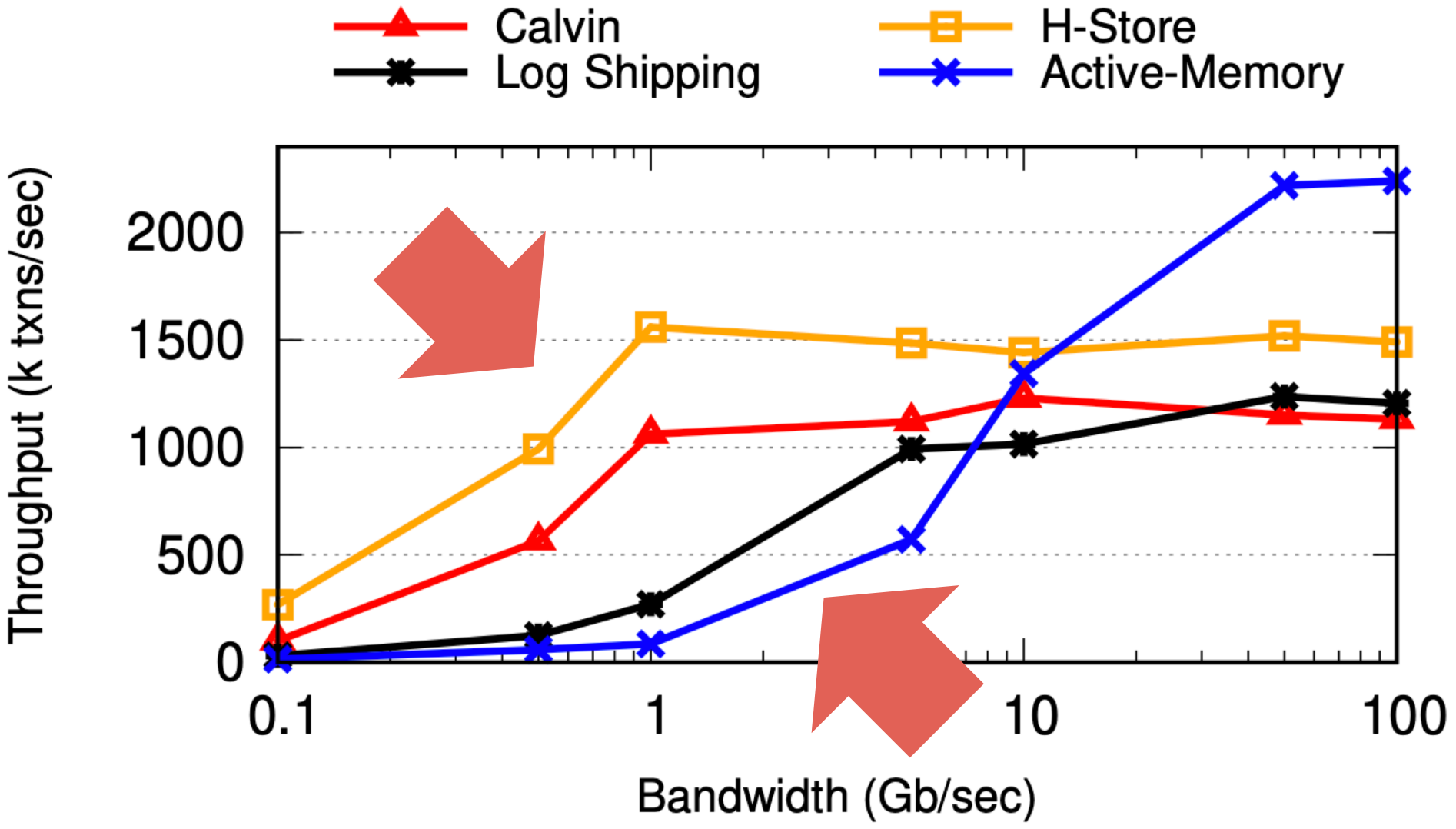
Table 2: The median latency per transaction.

H-Store	Calvin	Log shipping	Active-Memory
$85\mu s$	$1253\mu s$	$142\mu s$	$121\mu s$

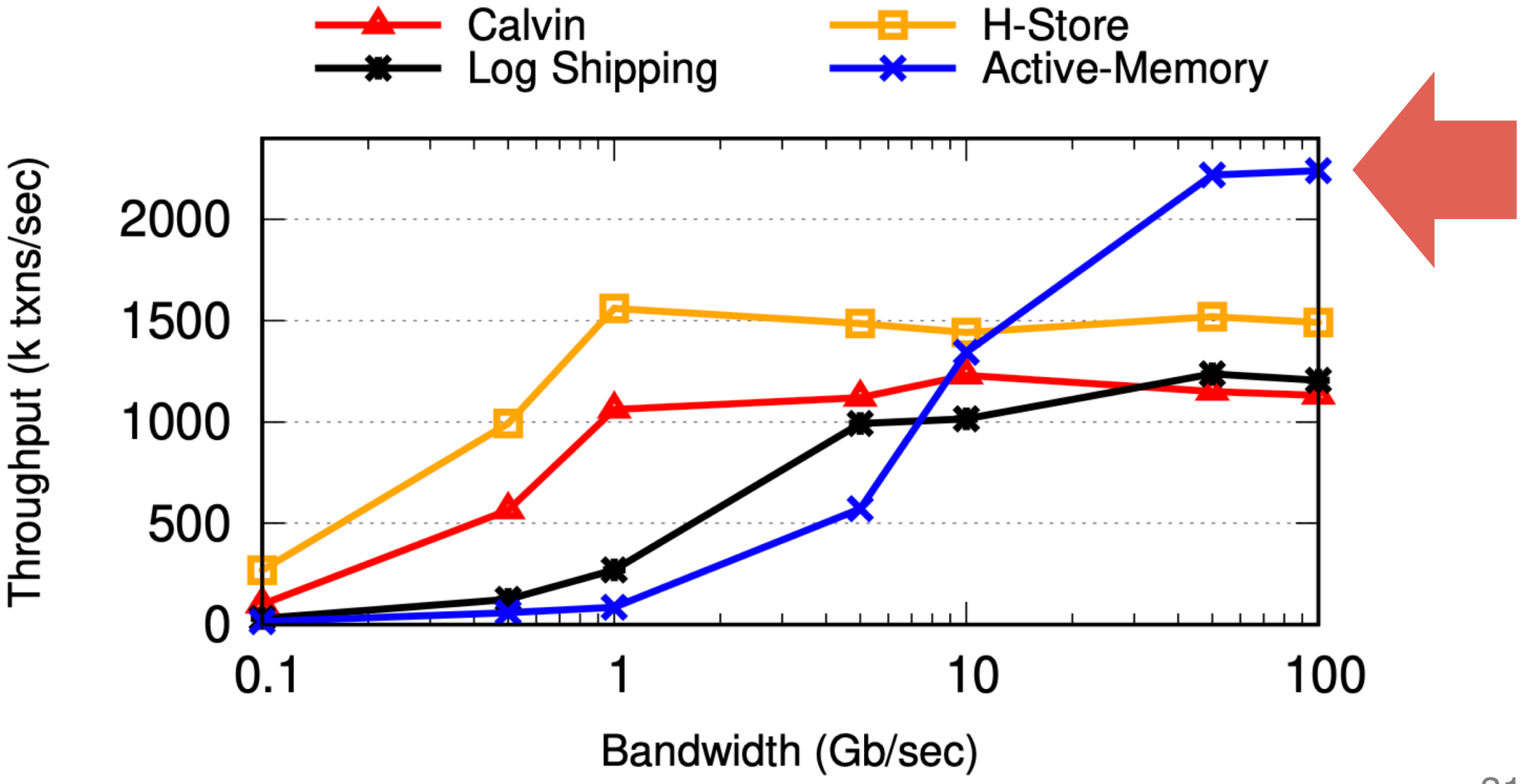
Evaluation – Multi-Partition Transactions



Evaluation – Network Bandwidth



Evaluation – Network Bandwidth

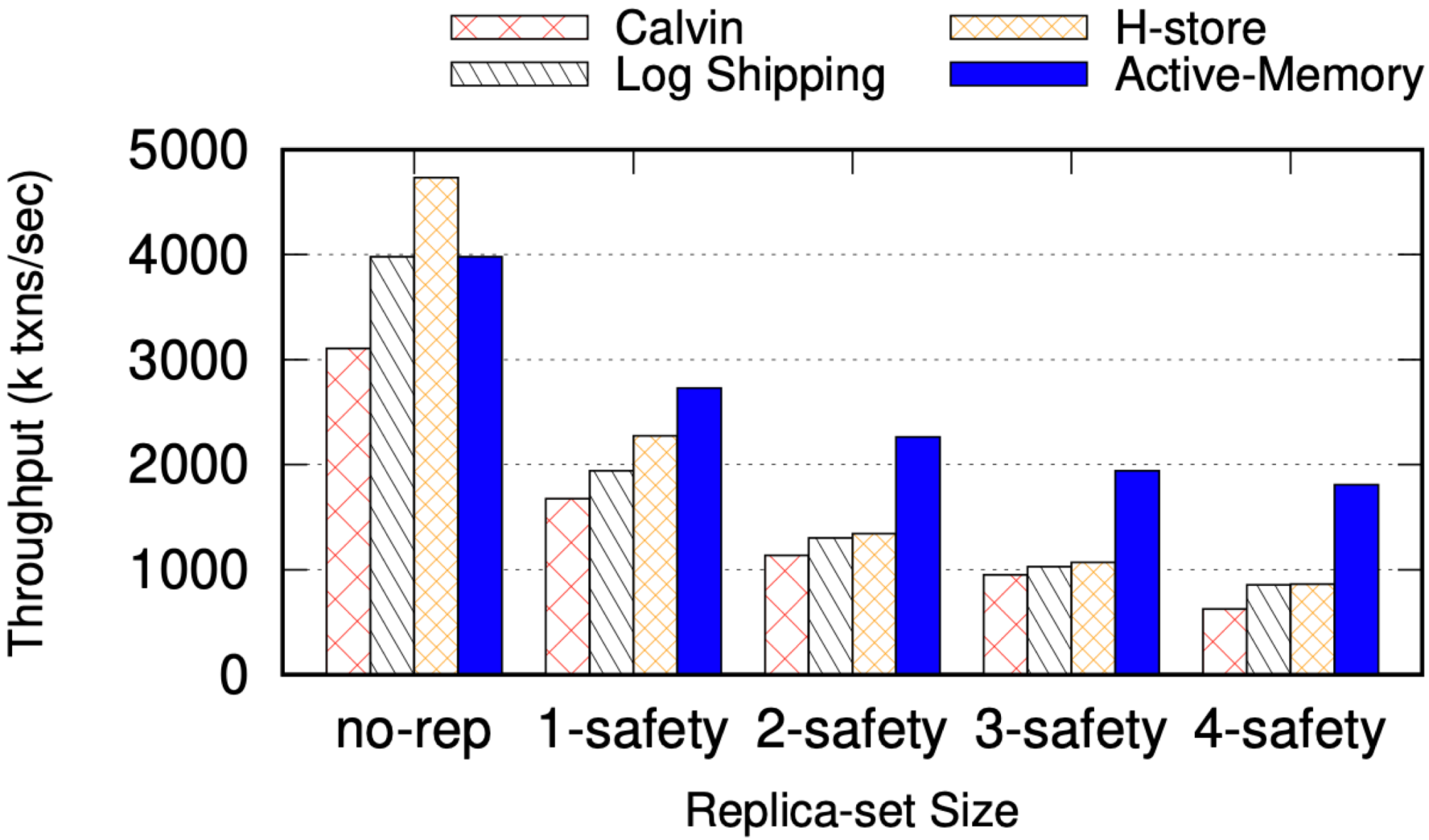


Evaluation – Network Traffic

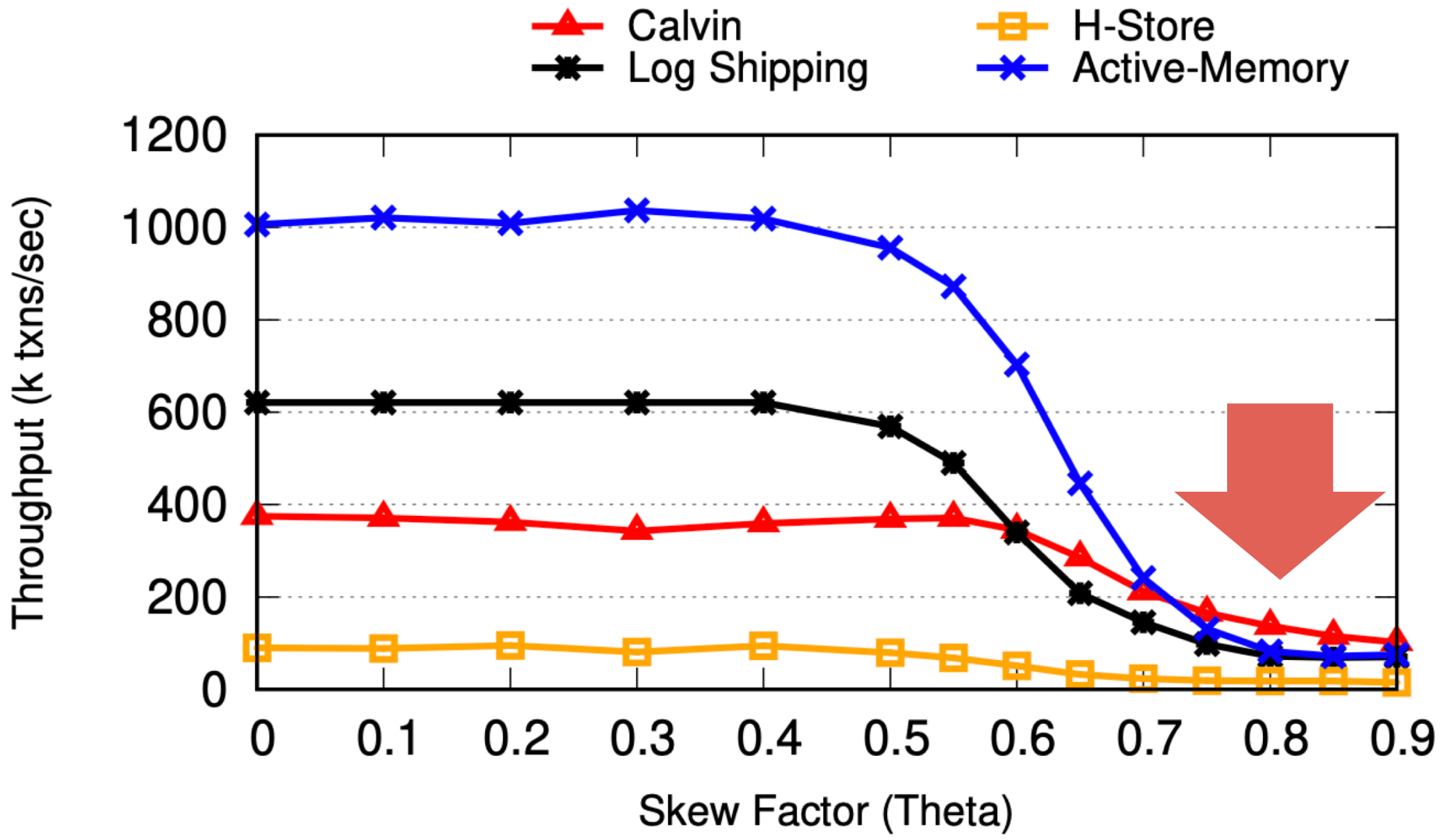
Table 3: The network traffic per transaction in each replication protocol in our unified platform.

	Single-partition Transactions	**Multi-partition Transactions**
H-Store	~ 0.3KB	~ 1.5KB
Calvin	~ 0.5KB	~ 0.6KB
Log shipping	~ 2.5KB	~ 3.7KB
Active-Memory	~ 4.5KB	~ 6KB

Evaluation – Replication Factor



Evaluation – Contention



Summary

- **RDMA shifts the bottleneck from network to CPU**
- Conventional HA protocols (i.e., active-passive and active-active) are optimized for reducing network demand and is thus are no longer optimal for RDMA
- Active-memory is optimized to reduce CPU demand
- **Active-memory achieves 2x performance improvement**

High Availability – Q/A

How long does recovery take?

Extend the idea to UC or UD queue pairs?

- In-order delivery of UNDO and updates

Exhaust entries in the circular UNDO log buffer?

Concurrency control other than NO-WAIT?

Eventual consistency? (RDMA for consensus)

Crash recovery? (NVM)

Log entry cannot fit in UNDO buffer?

Undo buffer overhead (w.r.t. cluster size, # of connections, etc.)

Backups unanimously decide to abort the current transaction?

Group Discussion

How to make Active Memory work when the network does not support in-order delivery?

What is the similarity between Active Memory and Write Behind Logging (discussed in Lecture 11)? Can they be combined?

List other examples in computer systems that increase CPU computation to reduce network overhead. How can RDMA change the design tradeoff in these cases?

Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com

- **Deadline: Friday 11:59pm**

Have a great Spring Break!