CS 839: Design the Next-Generation Database

Lecture 6: Deterministic Database

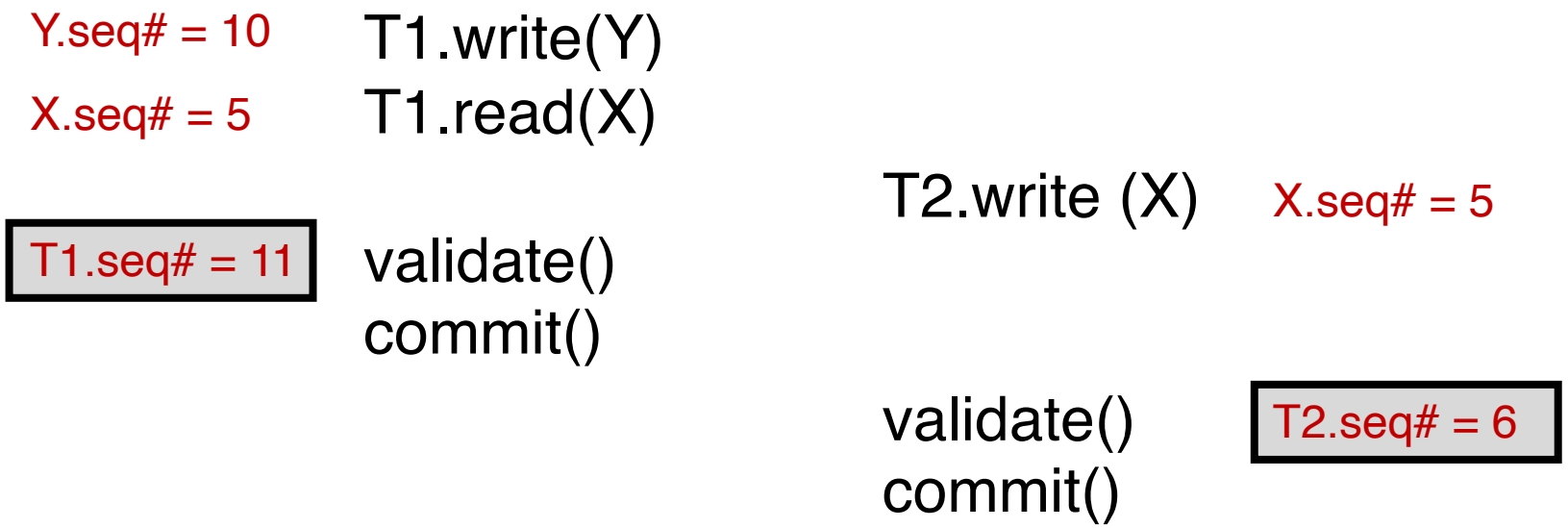
Xiangyao Yu

2/6/2020

Discussion Highlights

Silo compatible with operational logging?

No. See following example



For operational logging, must recover T1 before T2 (WAR dependency). Silo does not keep track of WAR dependency.

Discussion Highlights

Reduce transaction latency in Silo?

- Adjust epoch length based on workload or abort rate
- Soft commit vs. hard commit
- Create epoch boundary dynamically

Distributed Silo?

- Global epoch number, TID synchronization
- One extra network round trip compared to 2PL:
  Locking WS + RS validation + Write

Today's Paper

Calvin: Fast Distributed Transactions for Partitioned Database Systems

Alexander Thomson
Yale University
thomson@cs.yale.edu

Thaddeus Diamond
Yale University
diamond@cs.yale.edu

Shu-Chun Weng
Yale University
scweng@cs.yale.edu

Kun Ren
Yale University
kun@cs.yale.edu

Philip Shao
Yale University
shao-philip@cs.yale.edu

Daniel J. Abadi
Yale University
dna@cs.yale.edu

ABSTRACT

Many distributed storage systems achieve high data access throughput via partitioning and replication, each system with its own advantages and tradeoffs. In order to achieve high scalability, however, today's systems generally reduce transactional support, disallowing single transactions from spanning multiple partitions. Calvin is a practical transaction scheduling and data replication layer that

1. BACKGROUND AND INTRODUCTION

One of several current trends in distributed database system design is a move away from supporting traditional ACID database transactions. Some systems, such as Amazon's Dynamo [13], MongoDB [24], CouchDB [6], and Cassandra [17] provide no transactional support whatsoever. Others provide only limited transactionality, such as single-row transactional updates (e.g. Bigtable [11])

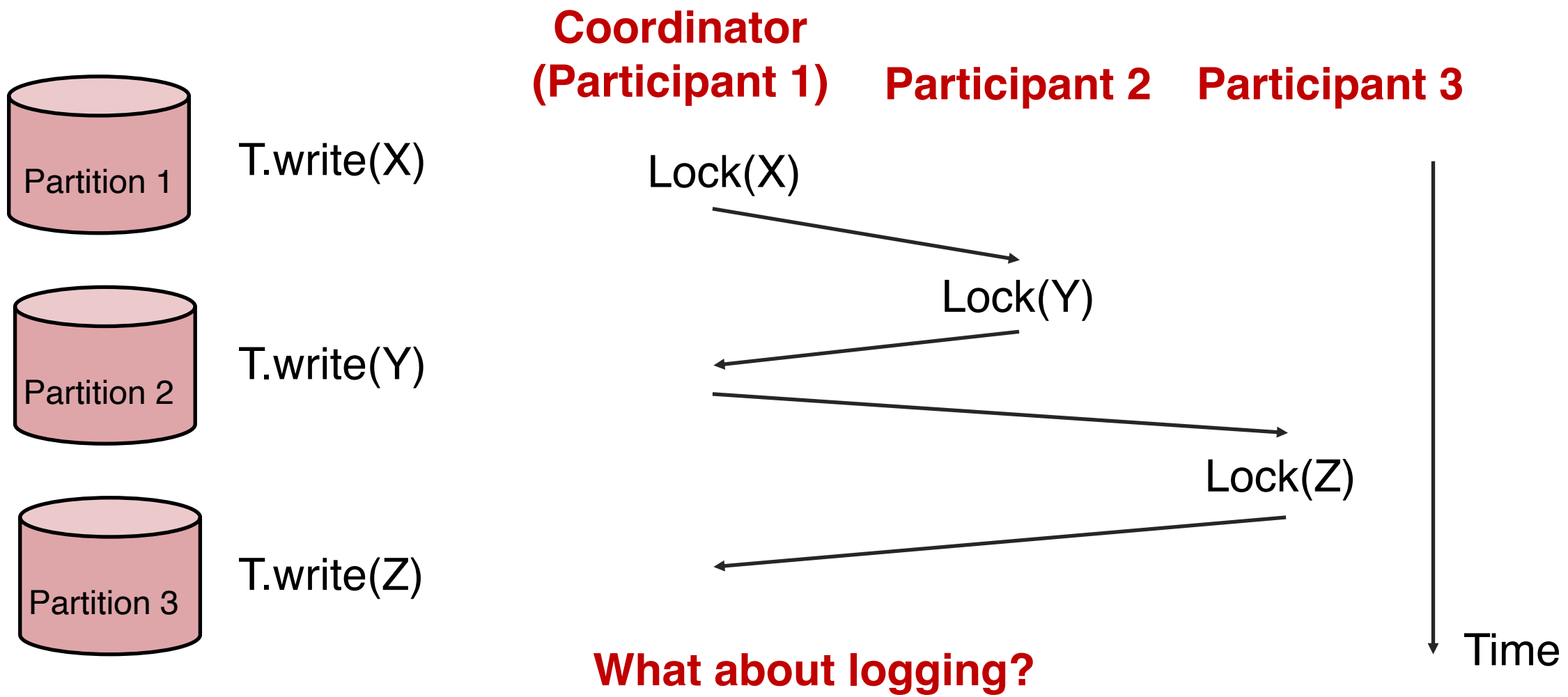
Today’s Agenda

Distributed transaction – Two-Phase Commit (2PC)

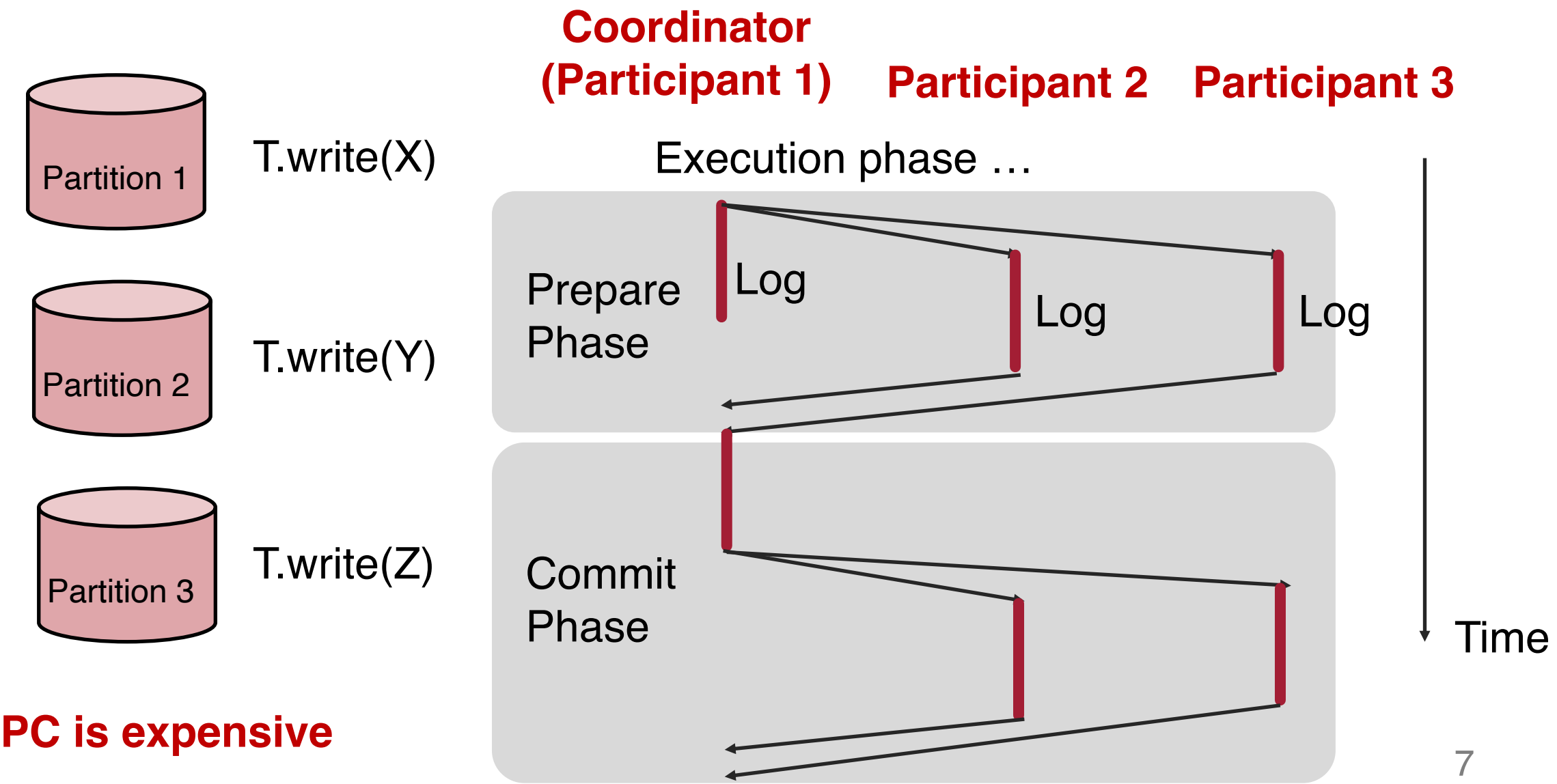
High availability

Calvin

Distributed Transaction

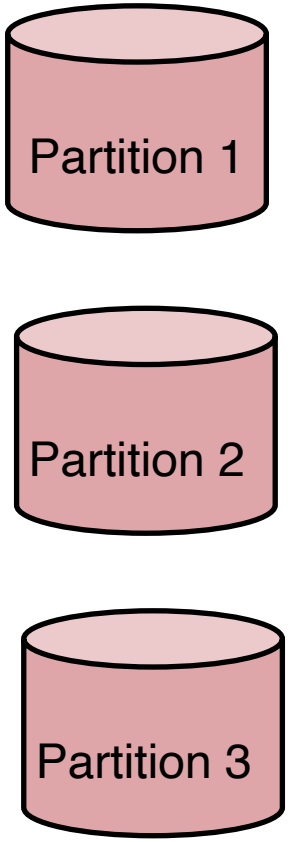


Two-Phase Commit (2PC)



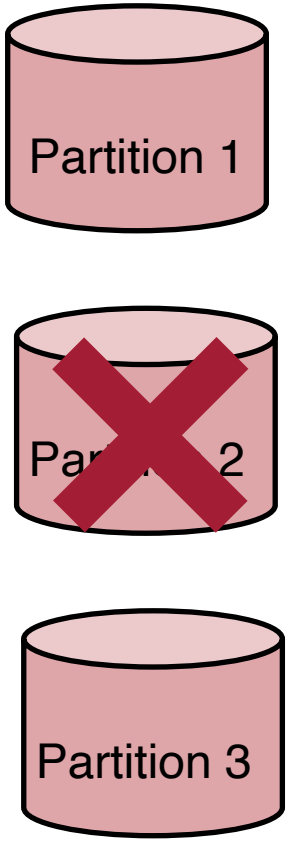
2PC is expensive

High Availability



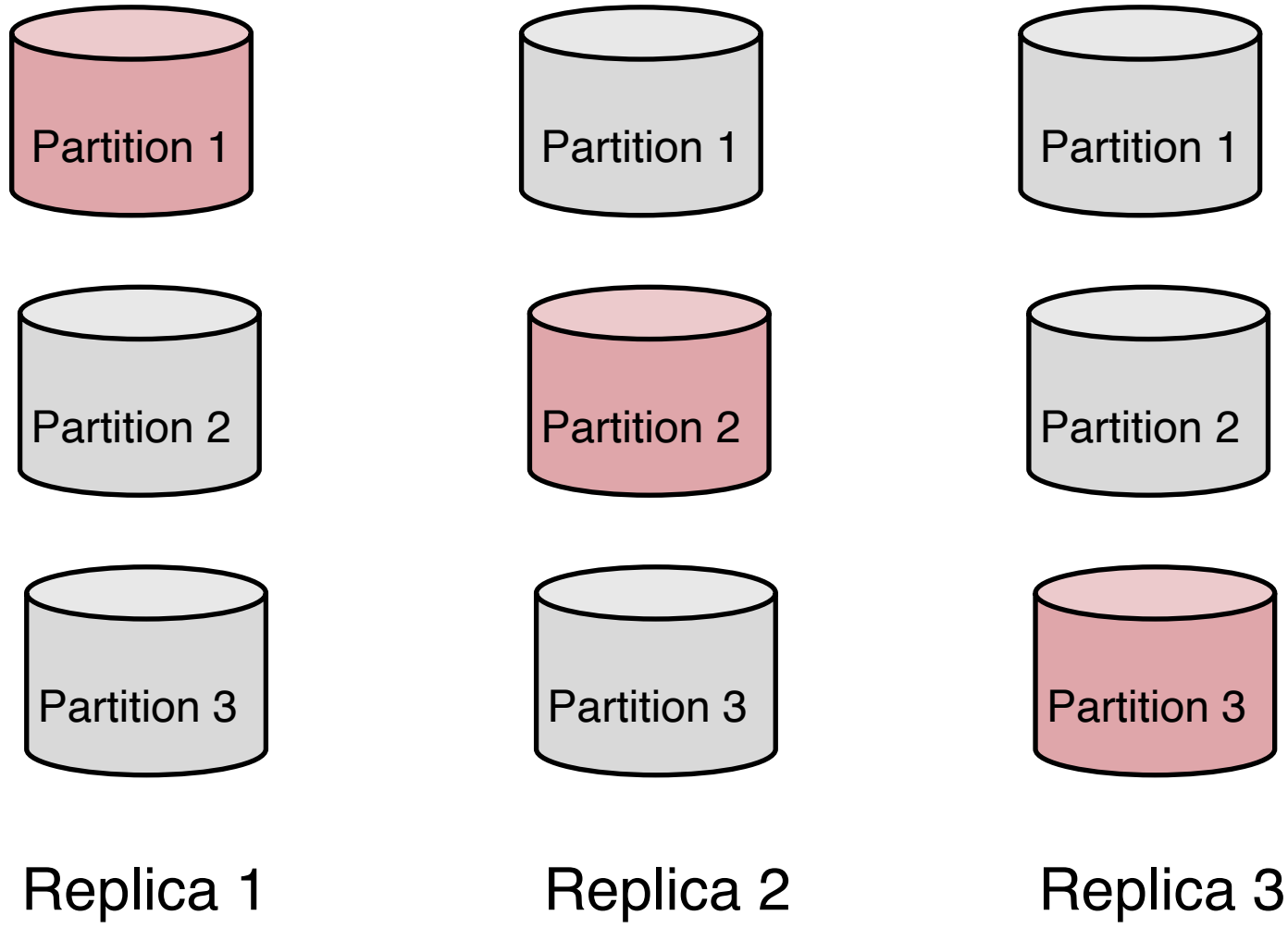
- Every tuple is mapped to one partition

High Availability



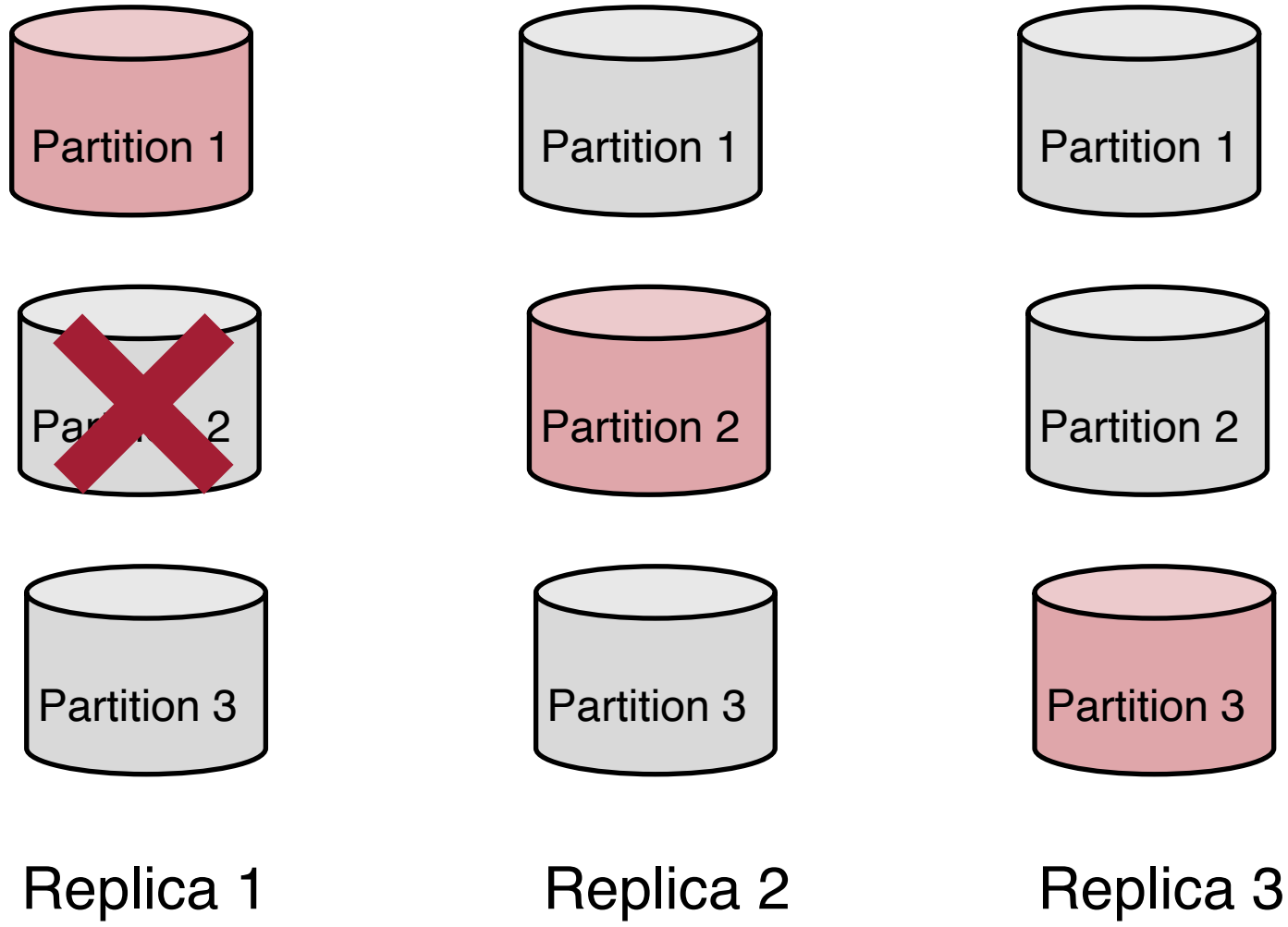
- A partition of data is unavailable if a server crashes

High Availability



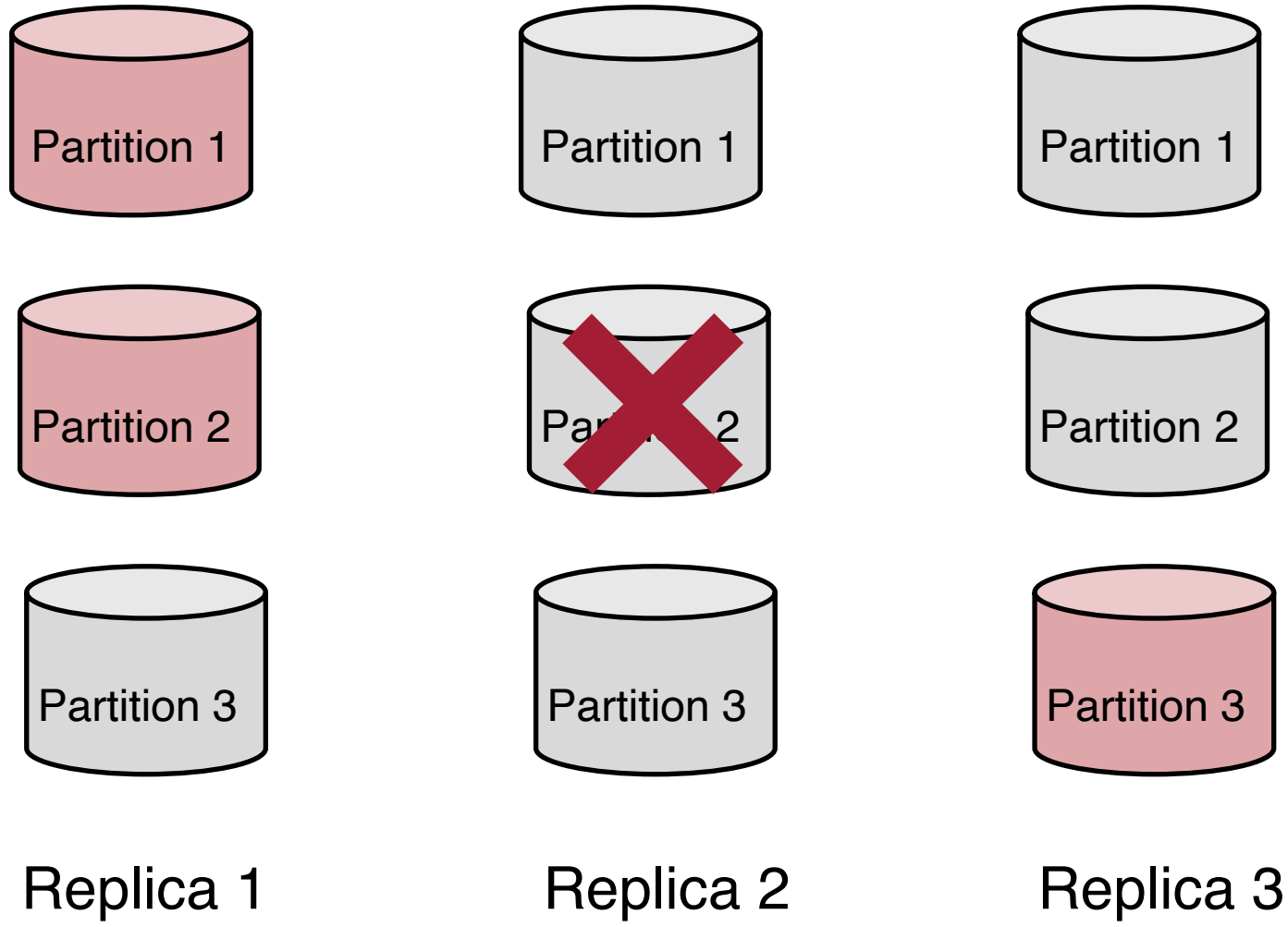
- Replicate data across multiple servers

High Availability



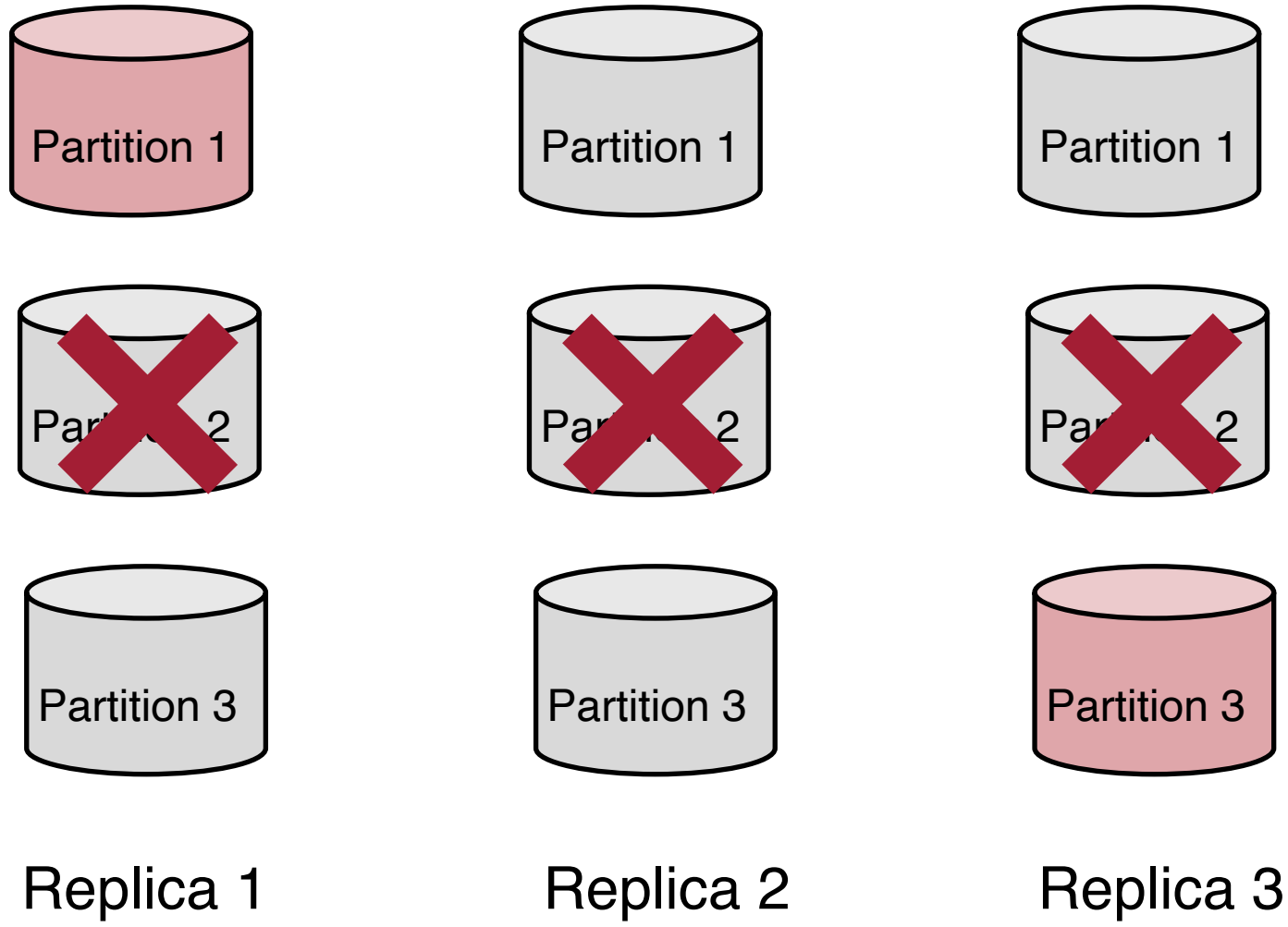
- Replicate data across multiple servers
- Data is available if at least one partition is still alive

High Availability



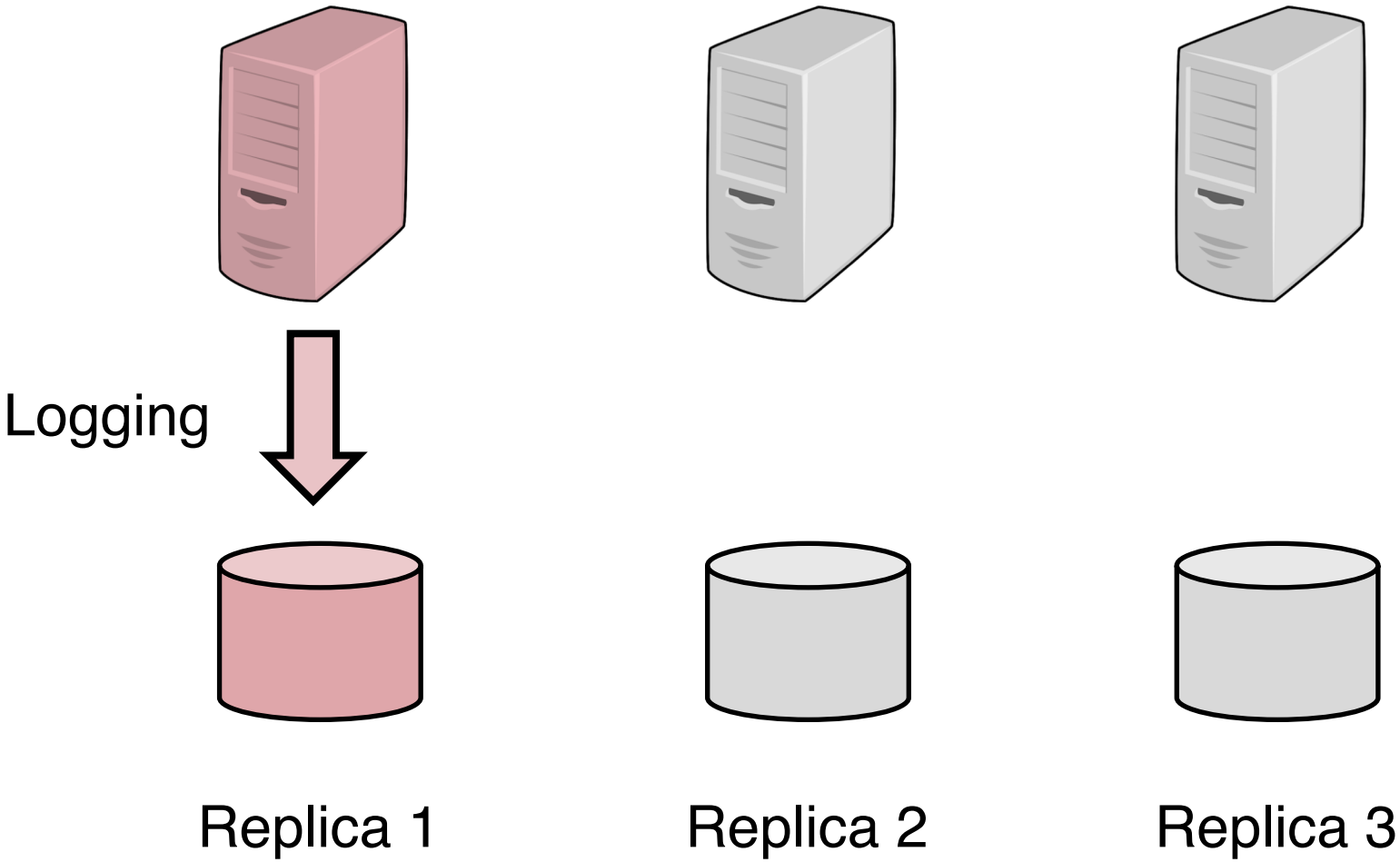
- Replicate data across multiple servers
- Data is available if at least one partition is still alive
- If the primary node fails, failure over to a secondary node

High Availability

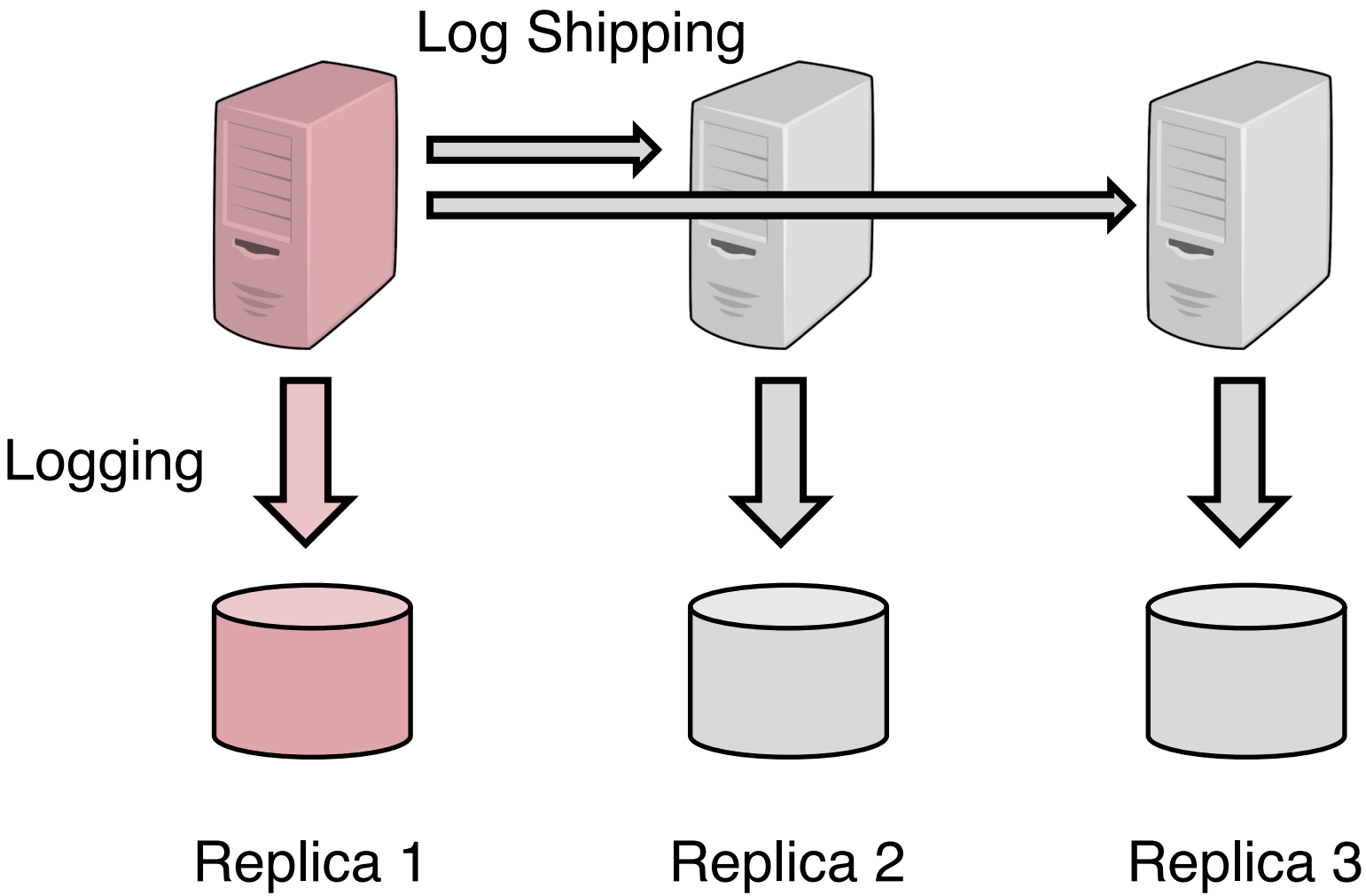


- Replicate data across multiple servers
- Data is available if at least one partition is still alive
- If the primary node fails, failure over to a secondary node
- Recovery from log if all replicas fail

Implementing High Availability

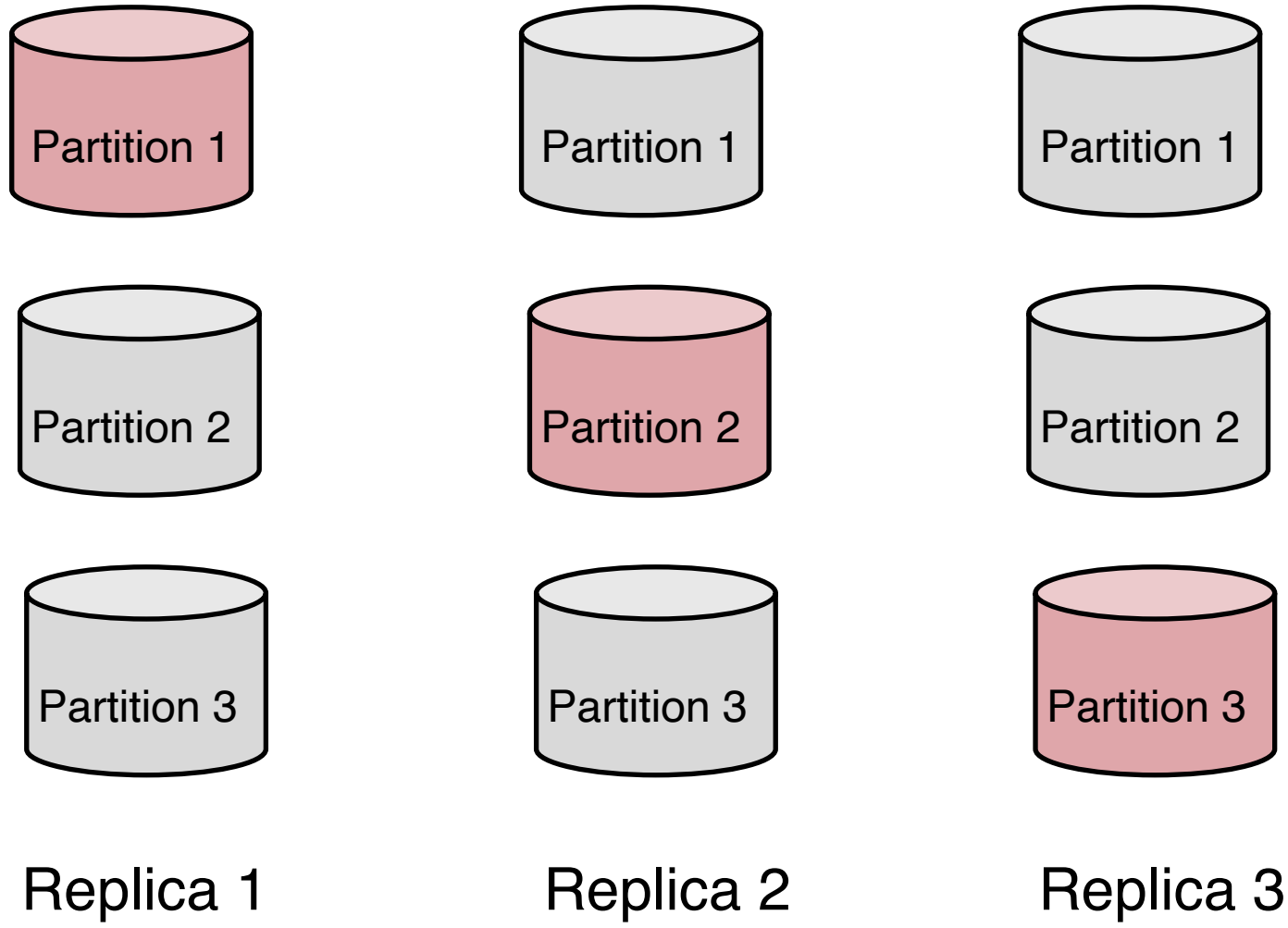


Implementing High Availability



Network can be a bottleneck for log shipping

Partition and Replication



Problem 1:
2PC is expensive

Problem 2:
Network can be a bottleneck for log shipping

Deterministic Transactions

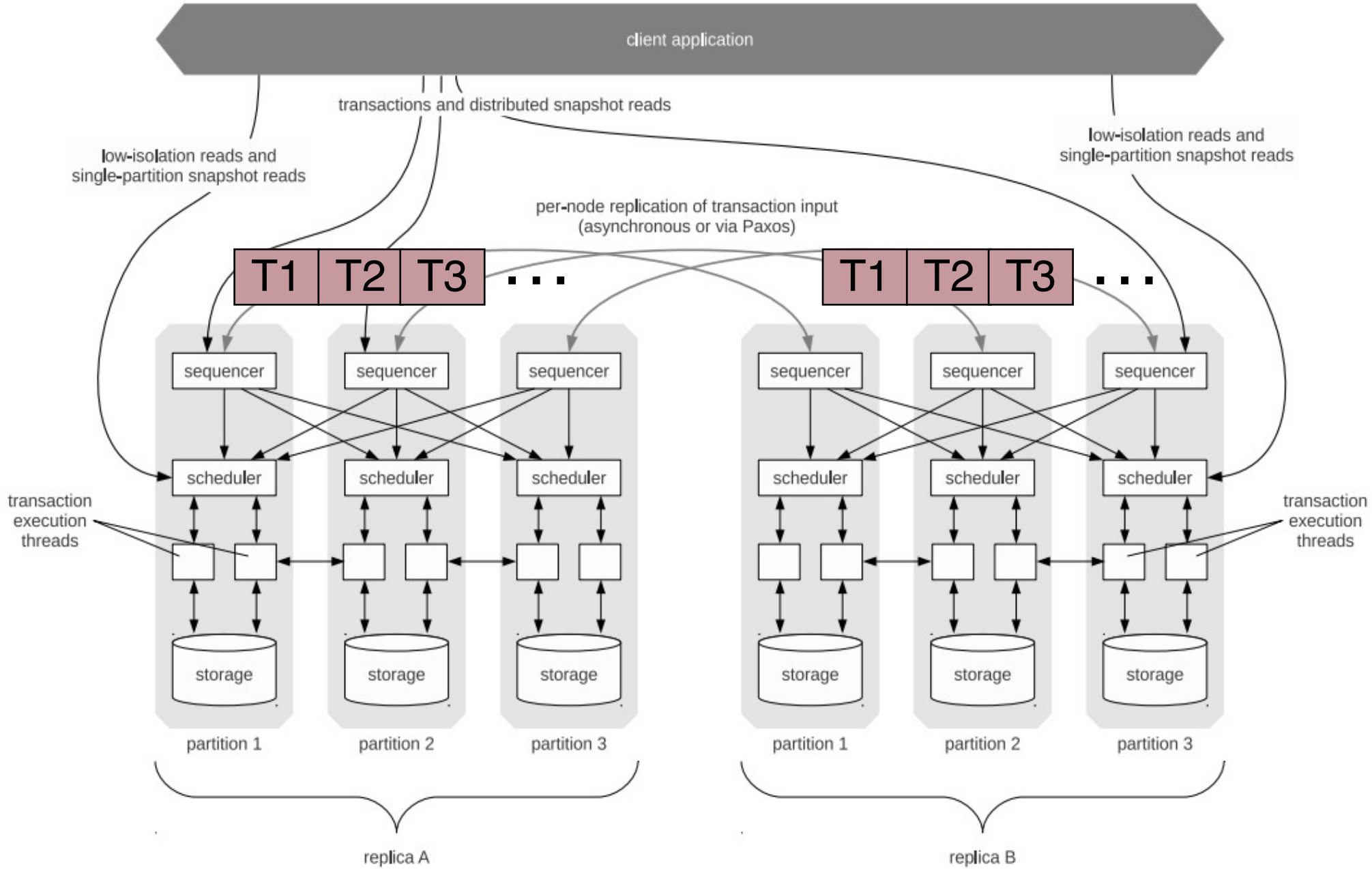
Decide the global execution order of transactions before executing them

All replicas follow same order to execute the transactions

Non-deterministic events are resolved and logged before dispatching the transactions

Log batch of inputs -> No two-phase commit

Replicate inputs -> Less network traffic than log shipping


client application
transactions and distributed snapshot reads
low-isolation reads and
single-partition snapshot reads
low-isolation reads and
single-partition snapshot reads
per-node replication of transaction input
(asynchronous or via Paxos)
T1
T2
T3
T1
T2
T3
sequencer
scheduler
transaction
execution
threads
storage
partition 1
partition 2
partition 3
replica A
replica B

Sequencer

Distributed across all nodes

- No single point of failure
- High scalability

Replicate transaction inputs asynchronously through Paxos

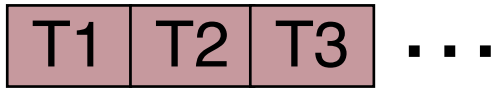
10ms batch epoch for batching

Batch the transaction inputs, determine their execution sequence, and dispatch them to the schedulers

Scheduler

All transactions have to **declare all lock requests before the transaction execution starts**

Single thread issuing lock requests



Example: T1.write(X), T2.write(X), T3.write(Y)

T1 locks X first

T3 can grab locks before T2 if T3 does not conflict with T1/T2

Transaction Execution Phases

1)Analysis all read/write sets

-Passive participants (read-only partition)

-Active participants (has write in partition)

2) Perform local reads

3) Serve remote reads

- send data needed by remote ones.

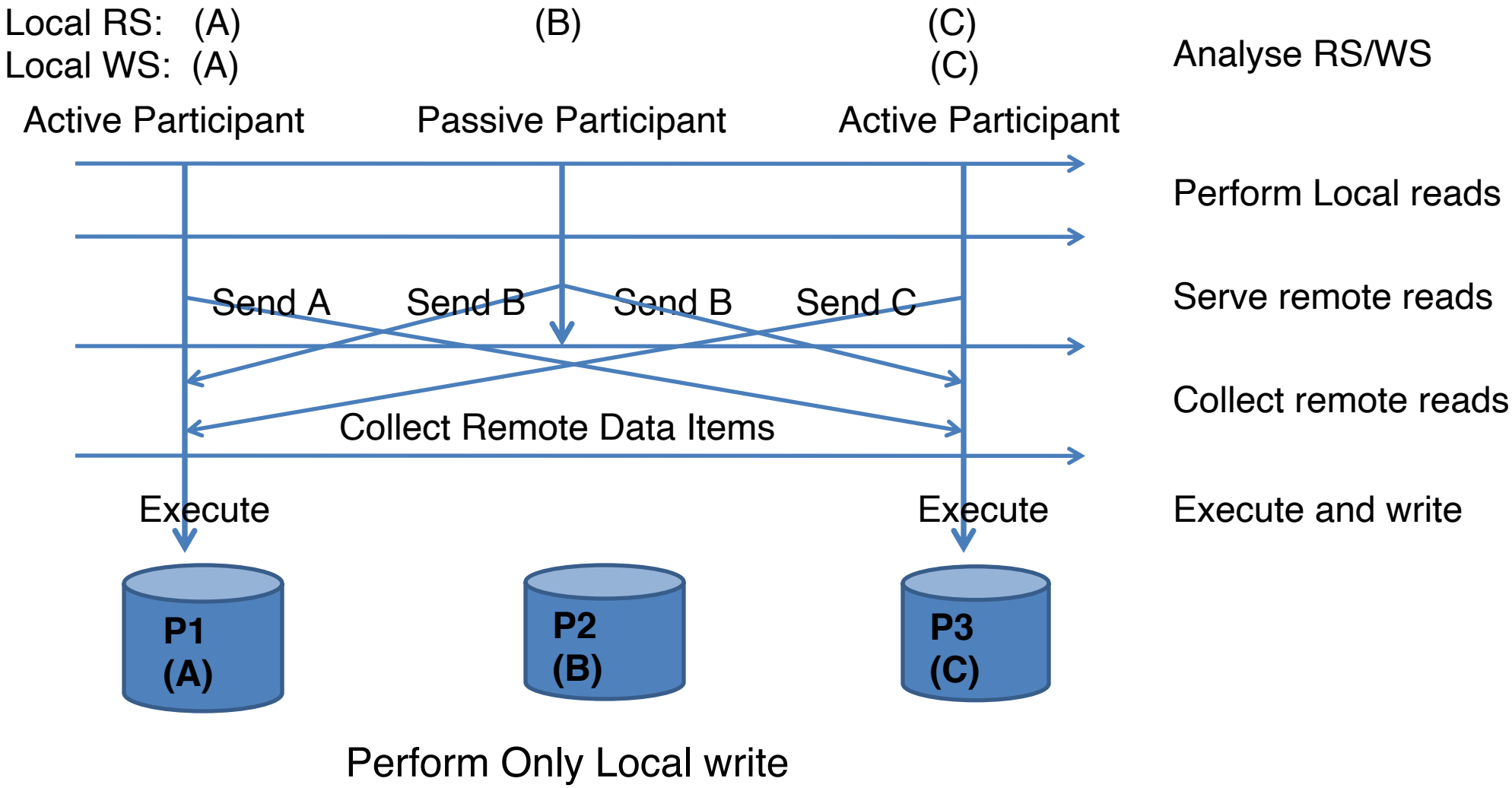
4) Collect remote read results

- receive data from remote.

5) execute transaction logic and apply writes

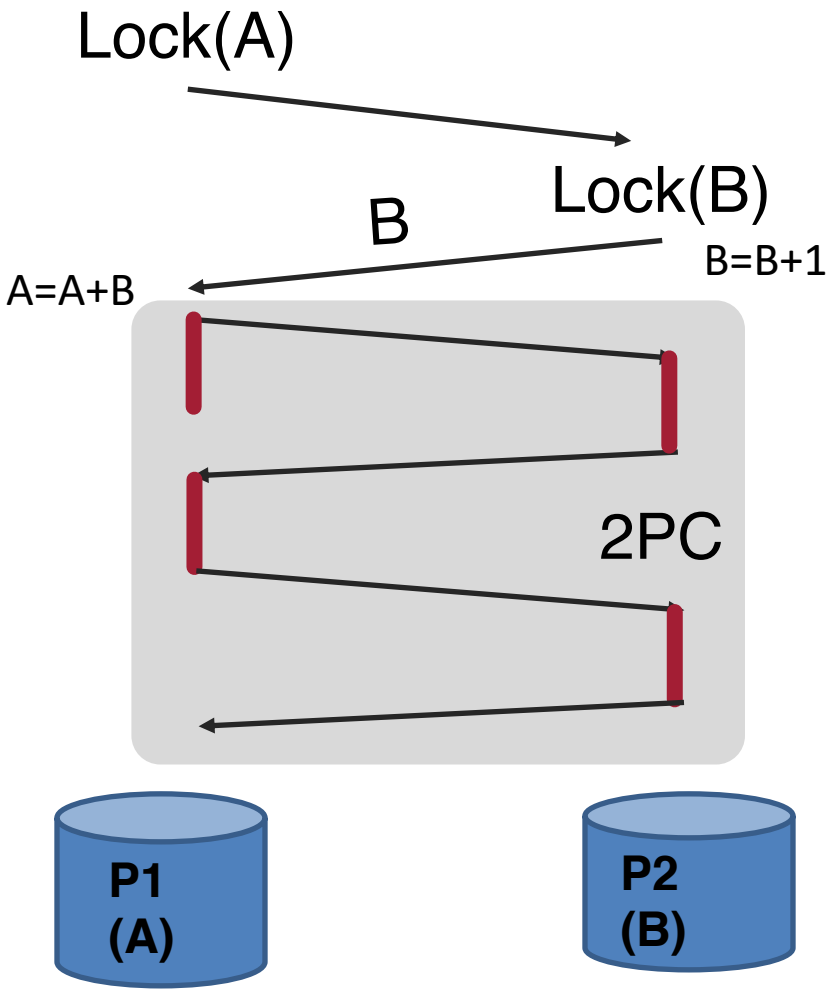
Example

T1 : A = A + B; C = C + B



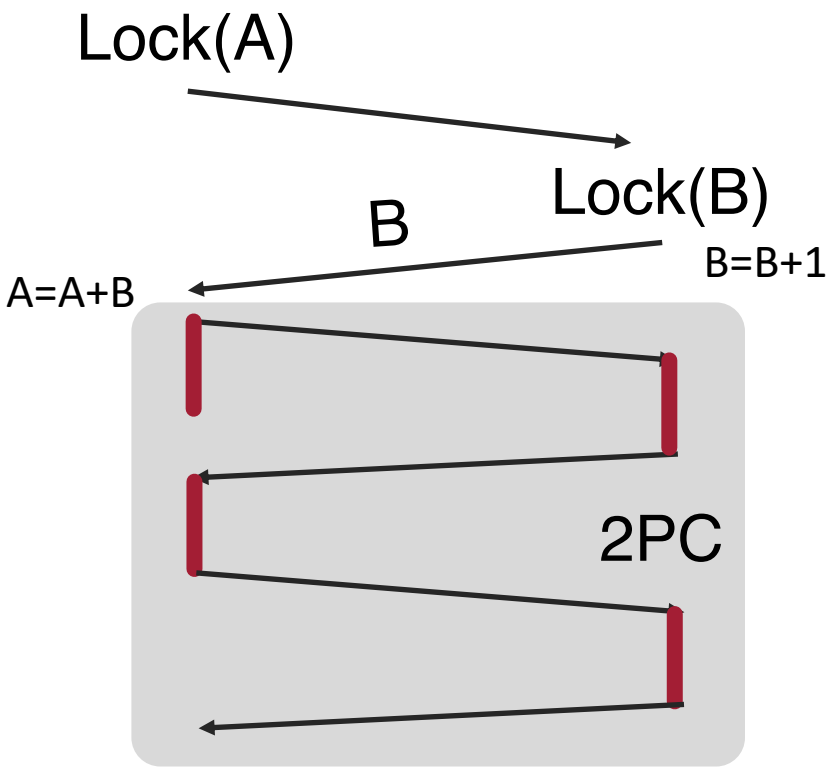
Conventional vs. Deterministic

T1: A = A + B; B = B + 1

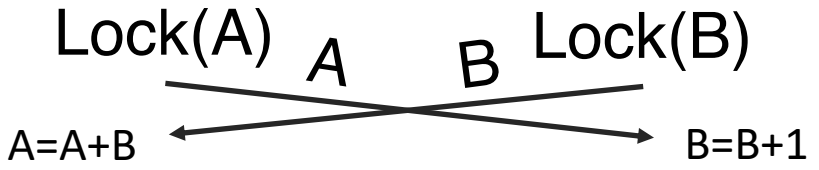


Conventional vs. Deterministic

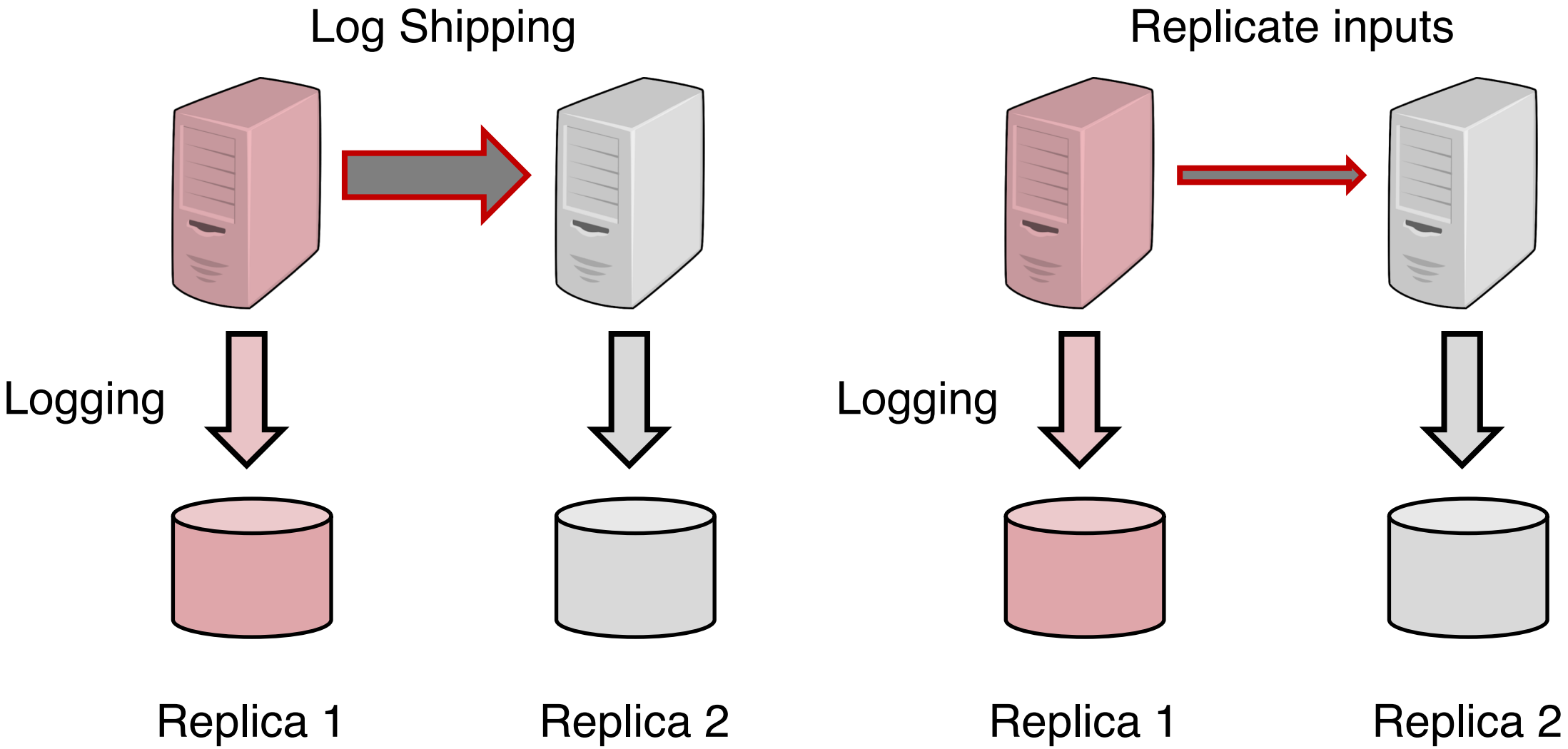
T1: A = A + B; B = B + 1



Paxos to replicate inputs



Conventional vs. Deterministic (replication)



Dependent Transactions

UPDATE table SET salary = 1.1 * salary WHERE salary < 1000

Need to perform reads to determine a transaction's read/write set

How to compute the read/write set?

- Modifying the client transaction code
- **Reconnaissance query** to discover full read/write sets
- If prediction is wrong (read/write set changes), repeat the process

Disk Based Storage

Fixed serial order leads to more blocking

- T1 write(A), write(B)
- T2 write(B), write(C)
- T3 write(C), write(D)

Solution

- Prefetch (warmup) request to relevant storage components
- Add artificial delay – equals to I/O latency
- Transaction would find all data items in memory

Checkpoint

Logs before a checkpoint can be truncated

Checkpointing modes

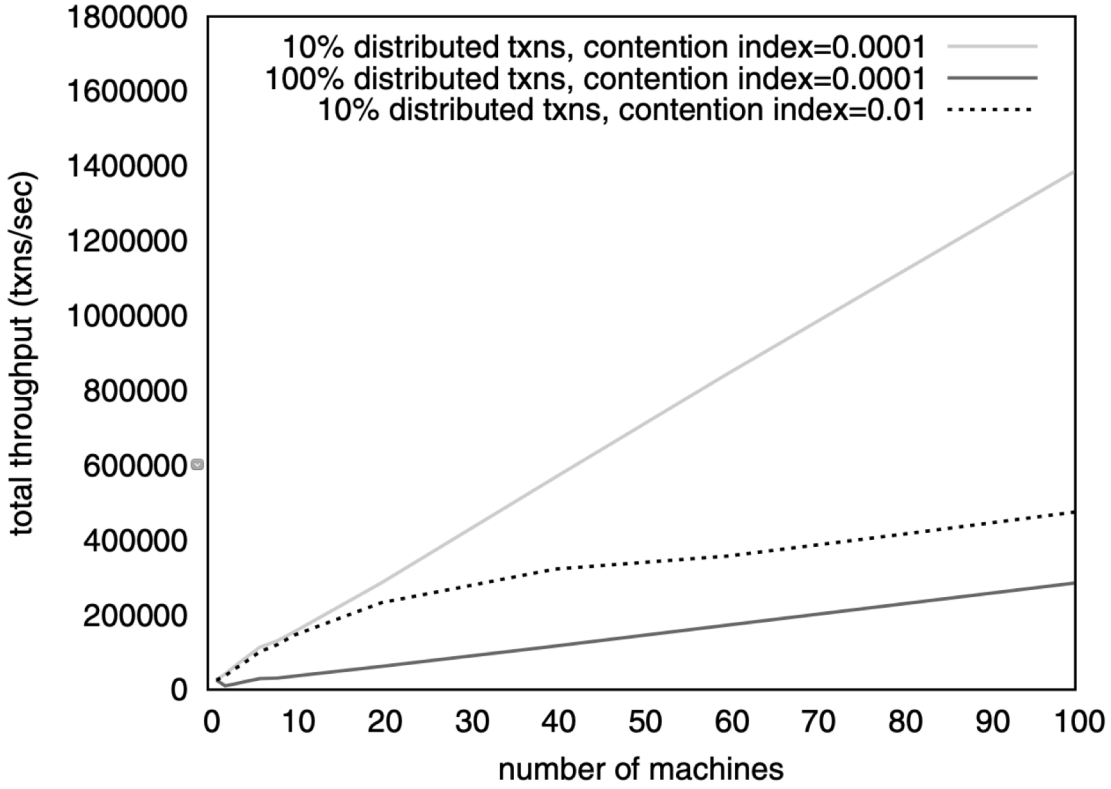
- Naïve synchronous mode:

  Stop one replica, checkpoint, replay delayed transactions
- Zig-Zag

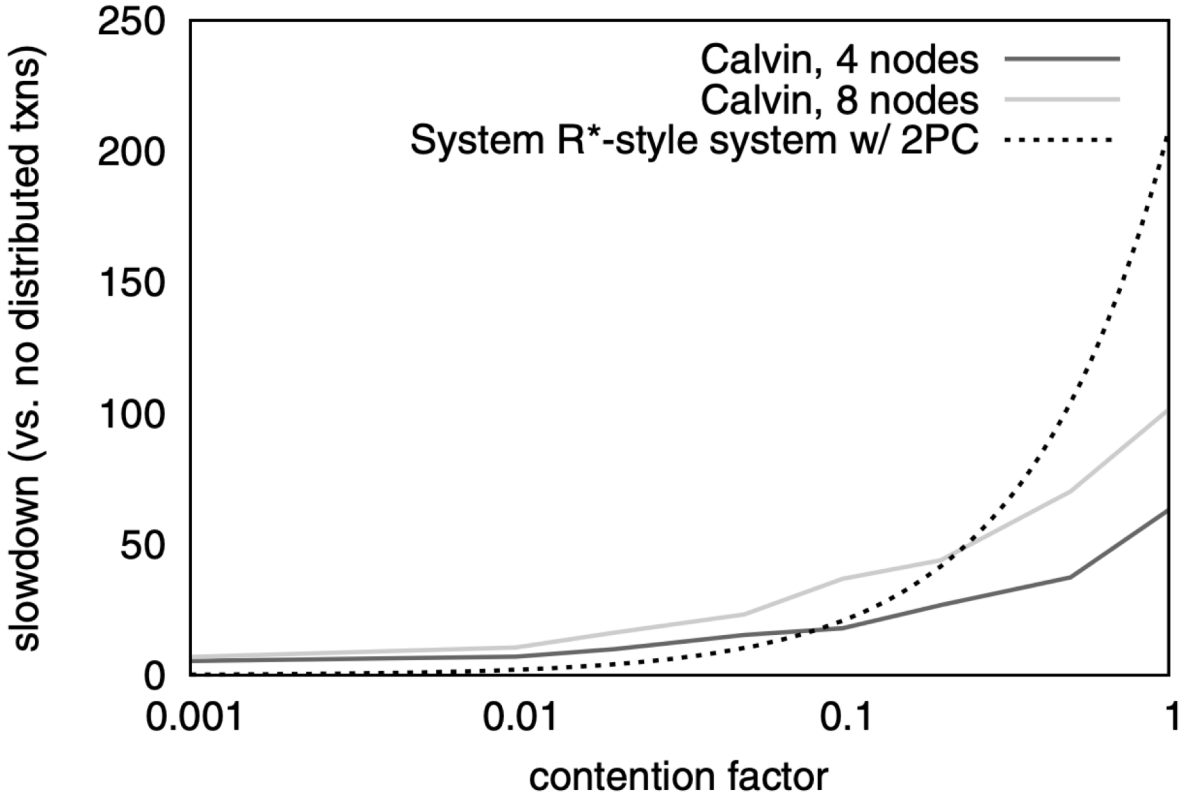
  Stores two copies of each record

Evaluation

Calvin can scale out



Calvin better than 2PC at high contention



Summary

Conventional distributed transactions

- Partition -> 2PC (network messages and log writes)
- Replication -> Log shipping (network traffic)

Deterministic transaction processing

- Determine the serial order before execution
- Replicate transaction inputs (less network traffic than log shipping)
- No need to run 2PC

Calvin – Q/A

Impact of deterministic transactions

- Series of papers from Prof. Daniel Abadi @ U Maryland
- Company: FaunaDB

Scheduler is a bottleneck for read-only workloads

Group Discussion

Is knowing read/write sets necessary for deterministic transactions? How does the protocol change if we remove this assumption?

Can you think of other optimizations if the read/write sets are known before transaction execution?

For a batch of transactions, Calvin performs a single Paxos to replicate inputs. Is it possible to amortize 2PC overhead with batch execution but not using deterministic transactions?

Before Next Lecture

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

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

Submit review for

A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics