Calvin: Fast Distributed Transactions for Partitioned Database Systems

A. Thomson et al., SIGMOD'12

Presenter: Lianghong Xu

High-level Overview

- A transaction processing and replication layer
  - Based on generic, non-transactional data store
  - full ACID for distributed transactions
  - Active, consistent replication
  - Horizontal scalability

The Problem

- Distributed transactions are ***expensive***
  - Agreement protocol
    - Multiple roundtrips in 2-phase commit
    - Much longer than transaction logic itself
    - Limits scalability
  - Locking
    - Lock held during the entire transaction
      - Including network latency
    - Possible deadlock

Consistent replication

- Many systems allow inconsistent replication
  - Replicas can diverge, but are eventually consistent
  - Dynamo, SimpleDB, Cassandra…
- Consistent replication: emerging trend
  - Instant failover
  - Increased latency, especially for geo-replication

Cost is only in latency, not throughput or contention

Goals of Calvin

- Eliminate 2-phase commit (scalability)
- Reduce lock duration (throughput)
- Provide consistent replication (consistency)

Approach:

- Decoupling **"transaction logic"** from **"heavy-lifting tasks"**
  - replication, locking, disk access…

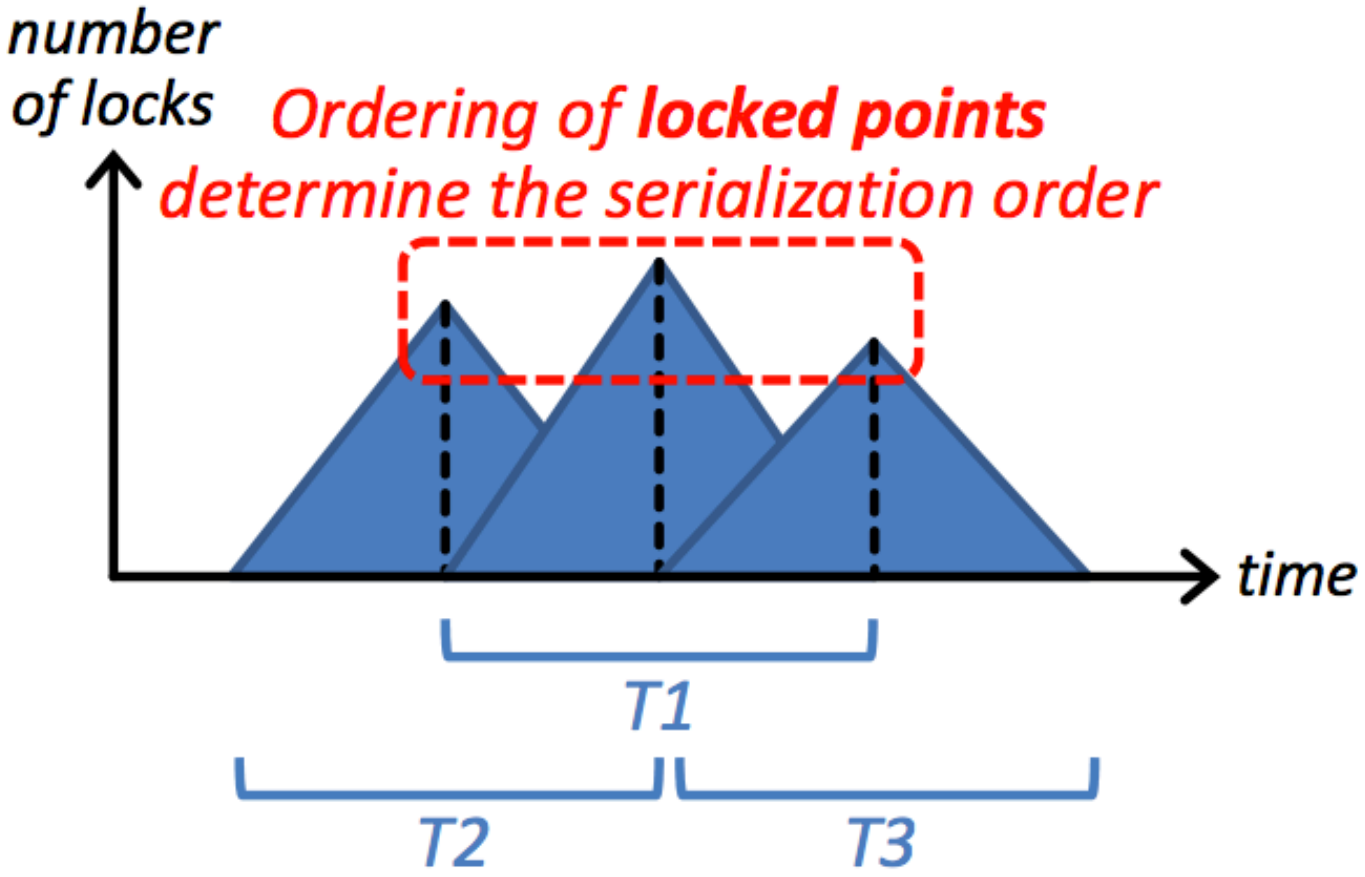
- **Deterministic concurrency control**

Non-deterministic Database Systems

- Aborts on non-deterministic events
  - E.g., node failure

- Serial ordering **cannot be pre-determined** given certain transaction inputs
  - Determined in the runtime
  - Ordering can diverge for different executions
  - Example: 2-phase locking

Serial Ordering in 2-phase Locking

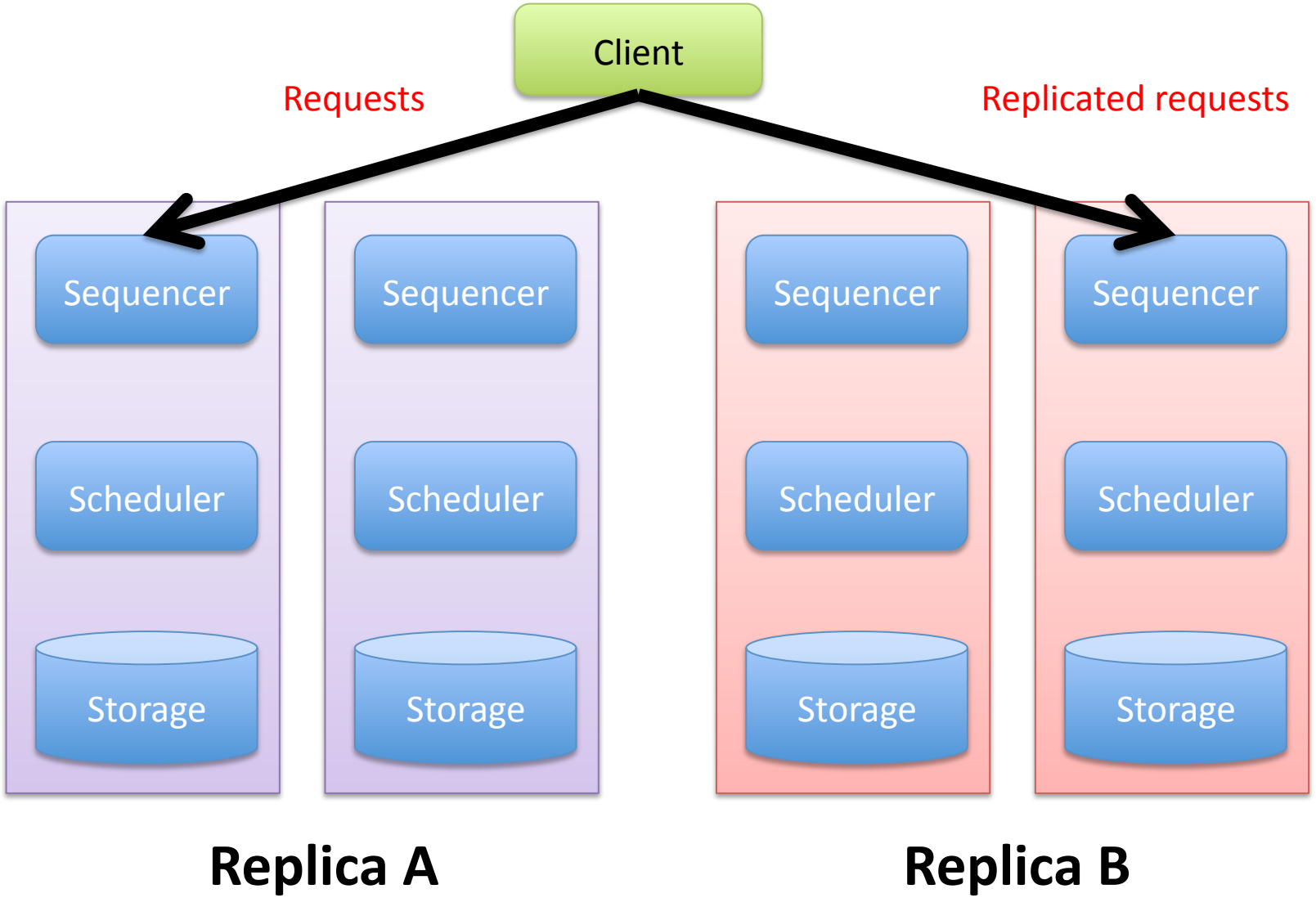


[Slide from Yoongu Kim]

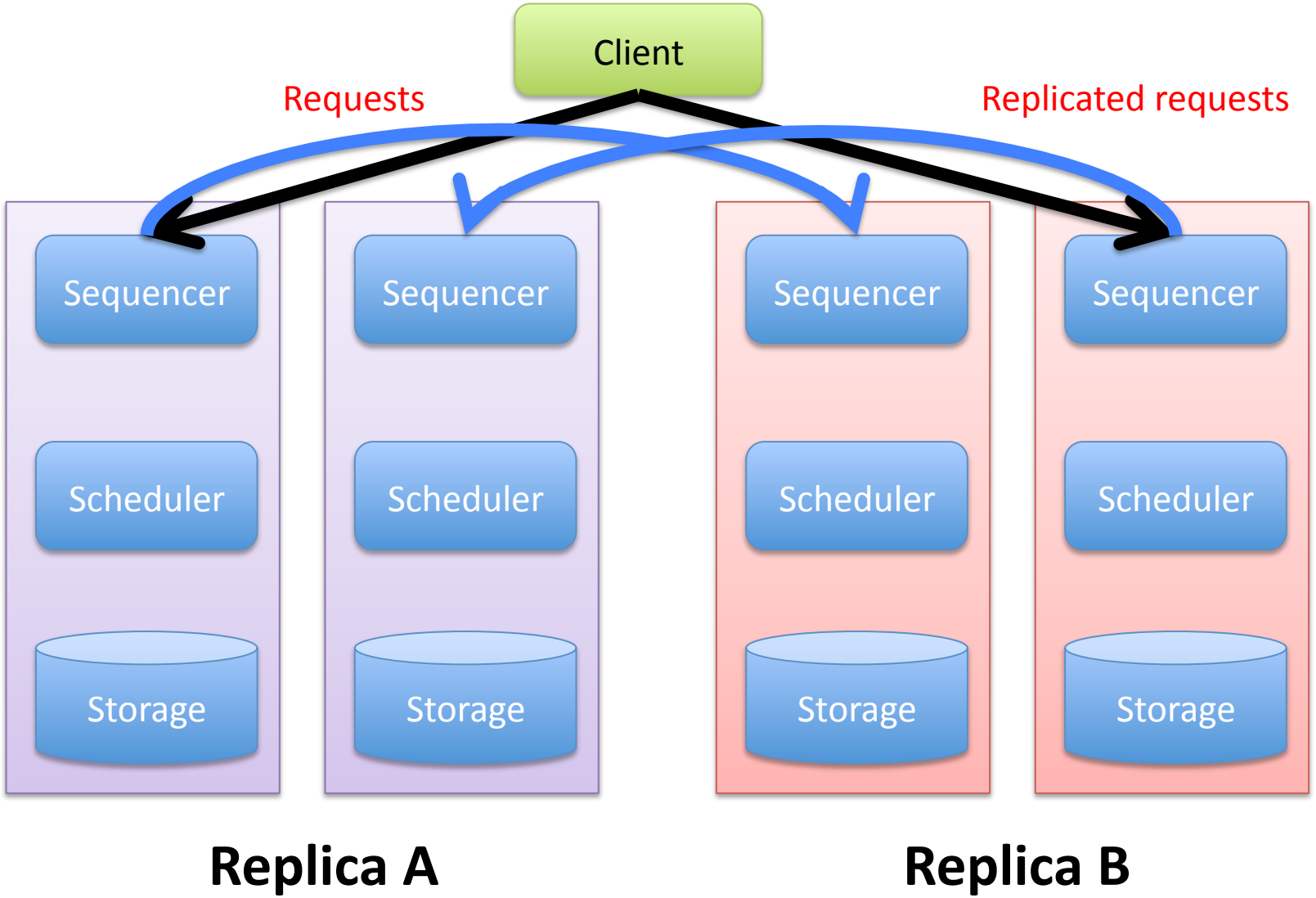
Deterministic database systems

- Given transactional inputs, serial ordering is ***pre-determined***.
- Benefits
  - No agreement protocol
    - No need to check node failures
    - Recovery from other replicas
    - ***No aborts*** due to non-deterministic events
  - Consistent replication made easier
    - Only need to replicate ***transactional inputs***
- Disadvantage
  - Reduced concurrency (potentially)

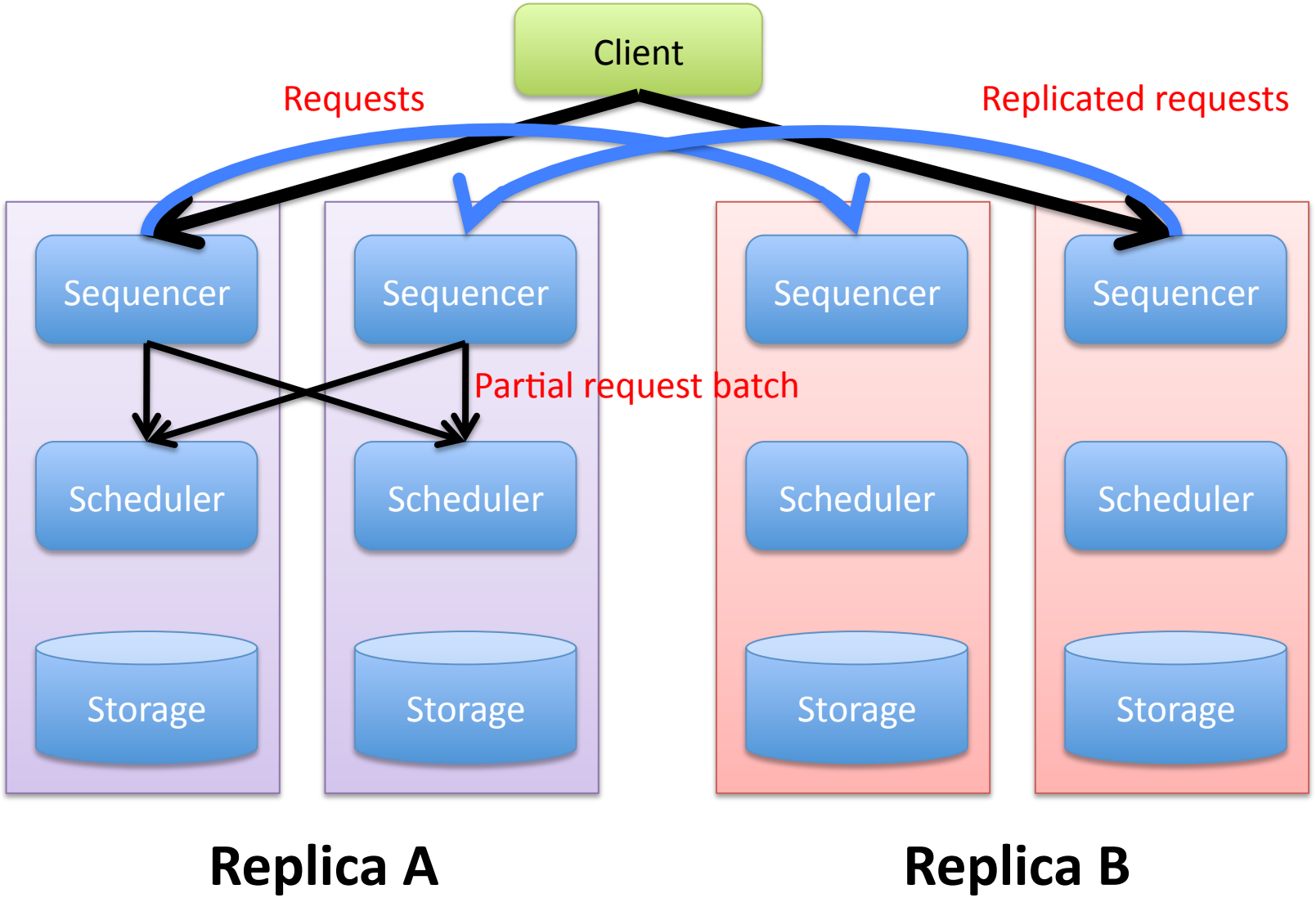
Architecture



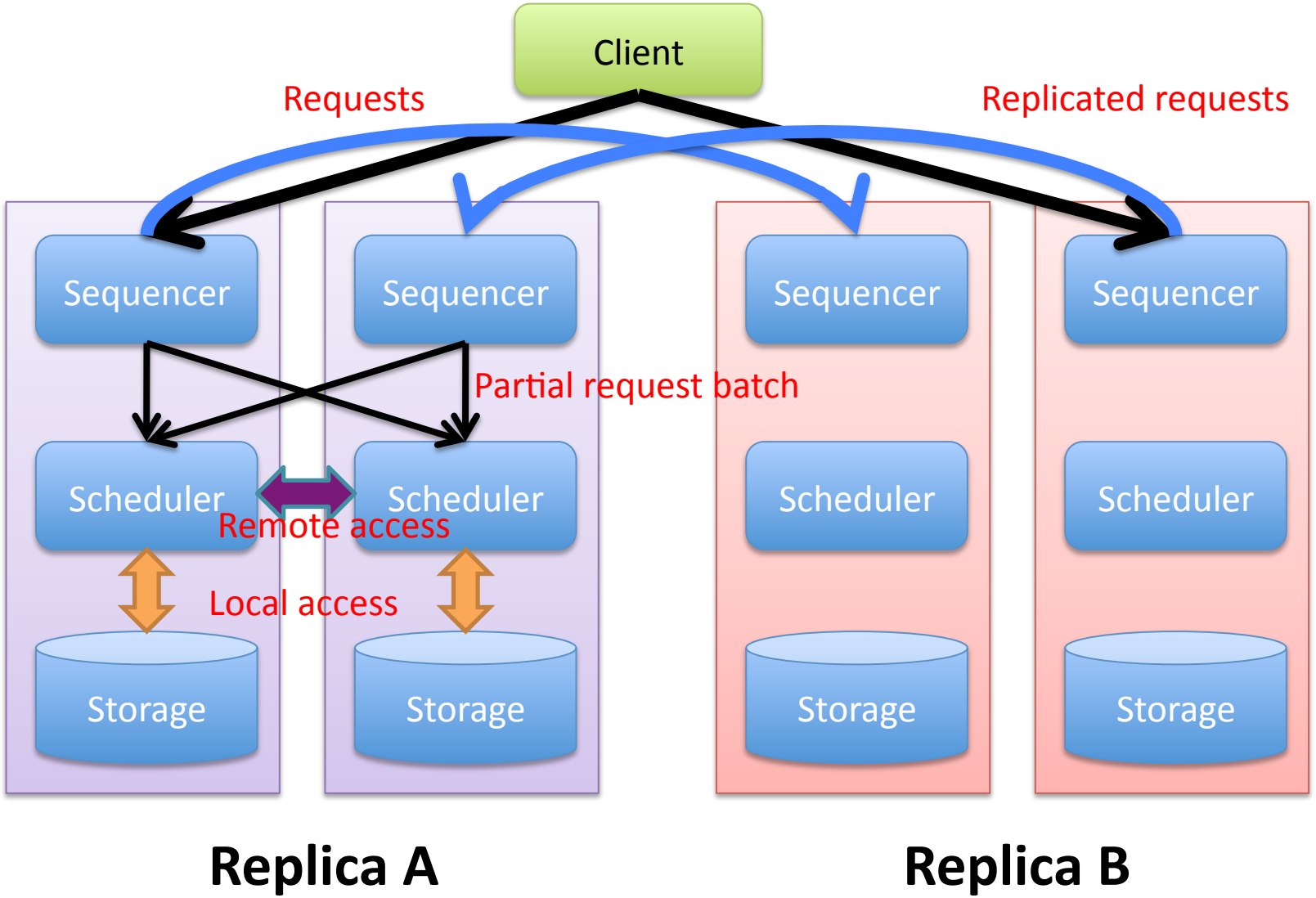
Architecture



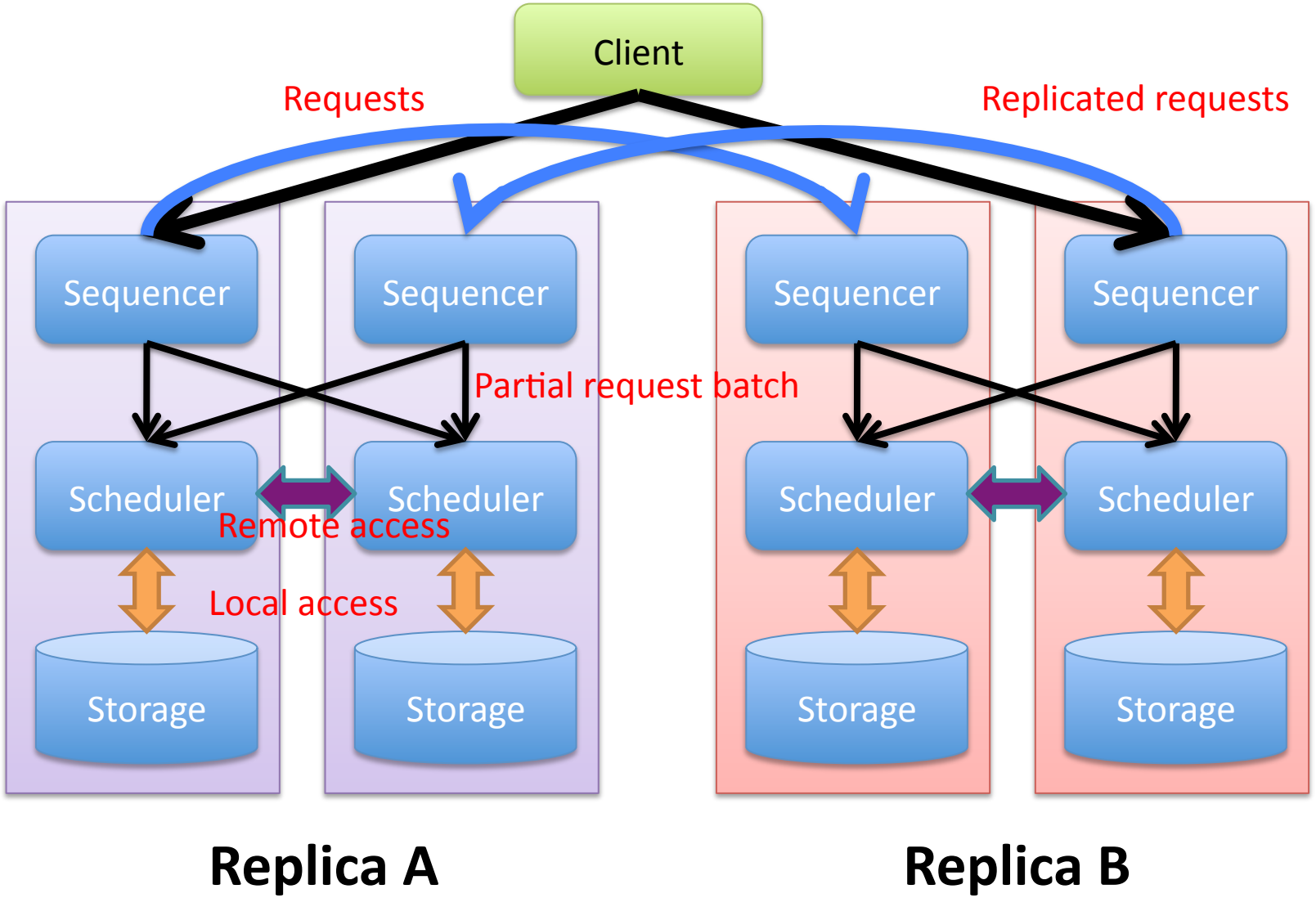
Architecture



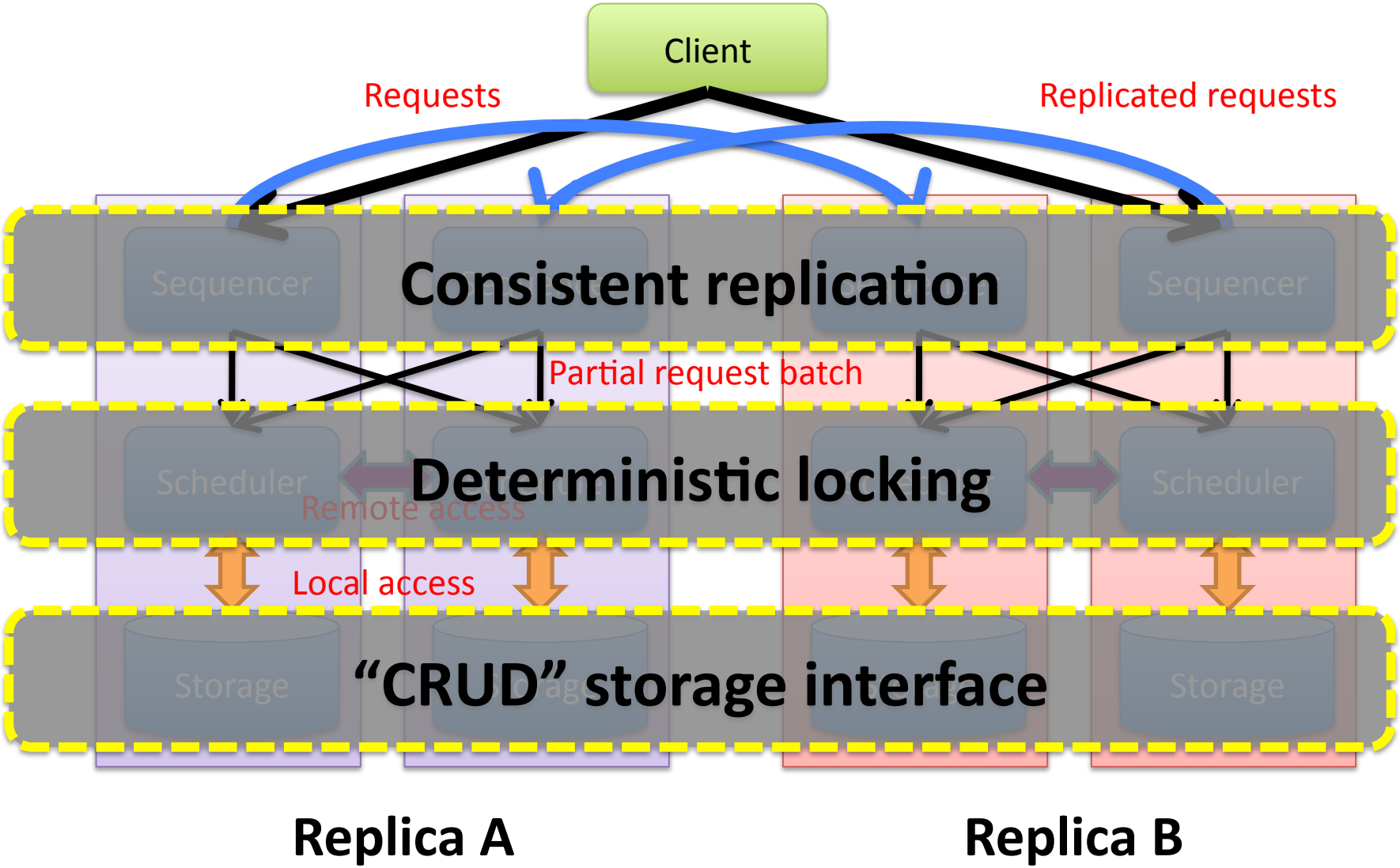
Architecture



Architecture



Architecture



Sequencer and replication

- **Only transactional inputs** need to be replicated
  - No need to worry about serial ordering (determinism)

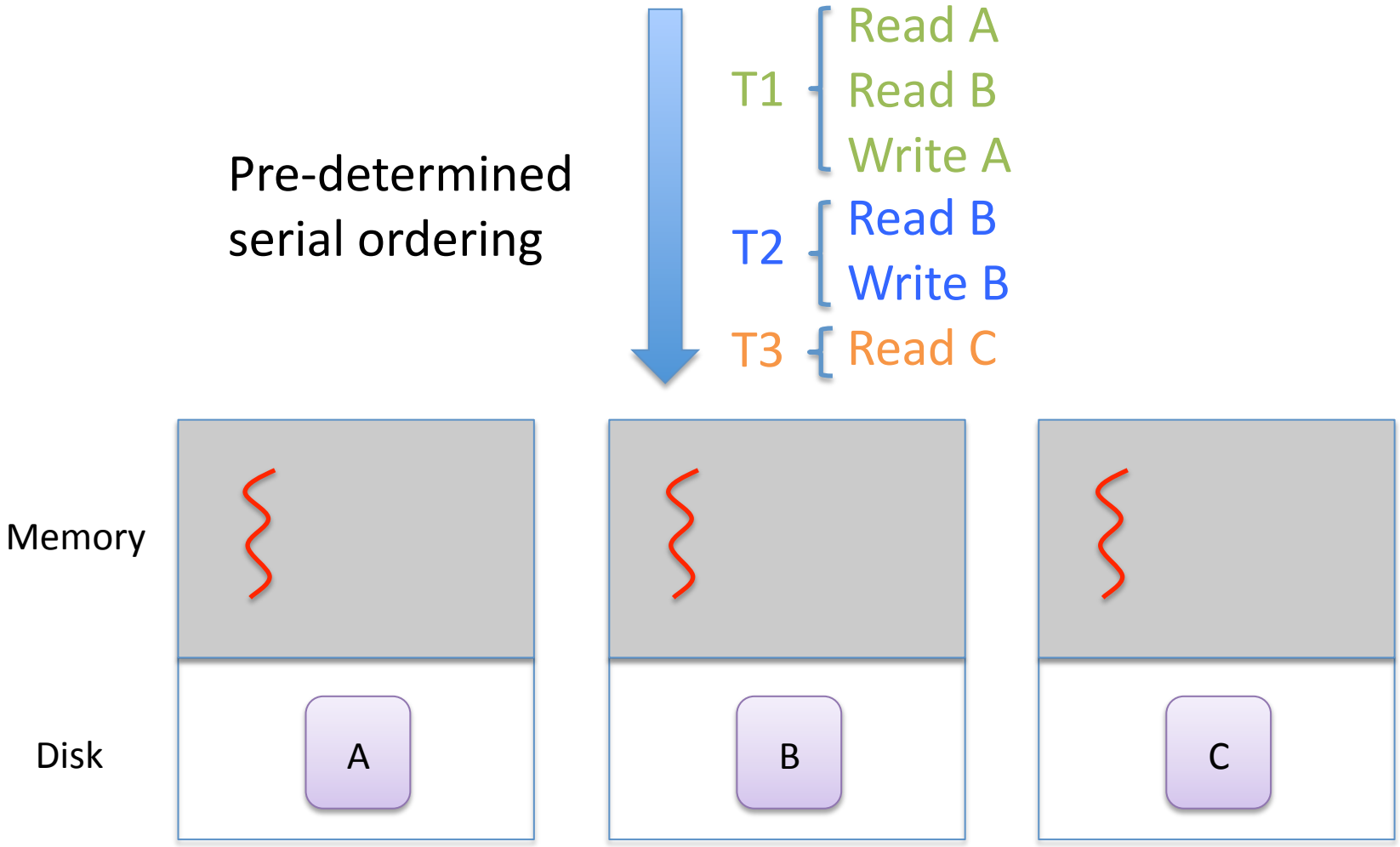
- **Asynchronous replication**
  - Master replica handles all requests
  - Propagate to slave replicas afterwards
  - **Low latency**, but **complex failure recovery**

- **Synchronous replication**
  - Based on **Paxos** (ZooKeeper)
  - Larger latency
  - **Throughput not affected**
    - Contention footprint remains the same

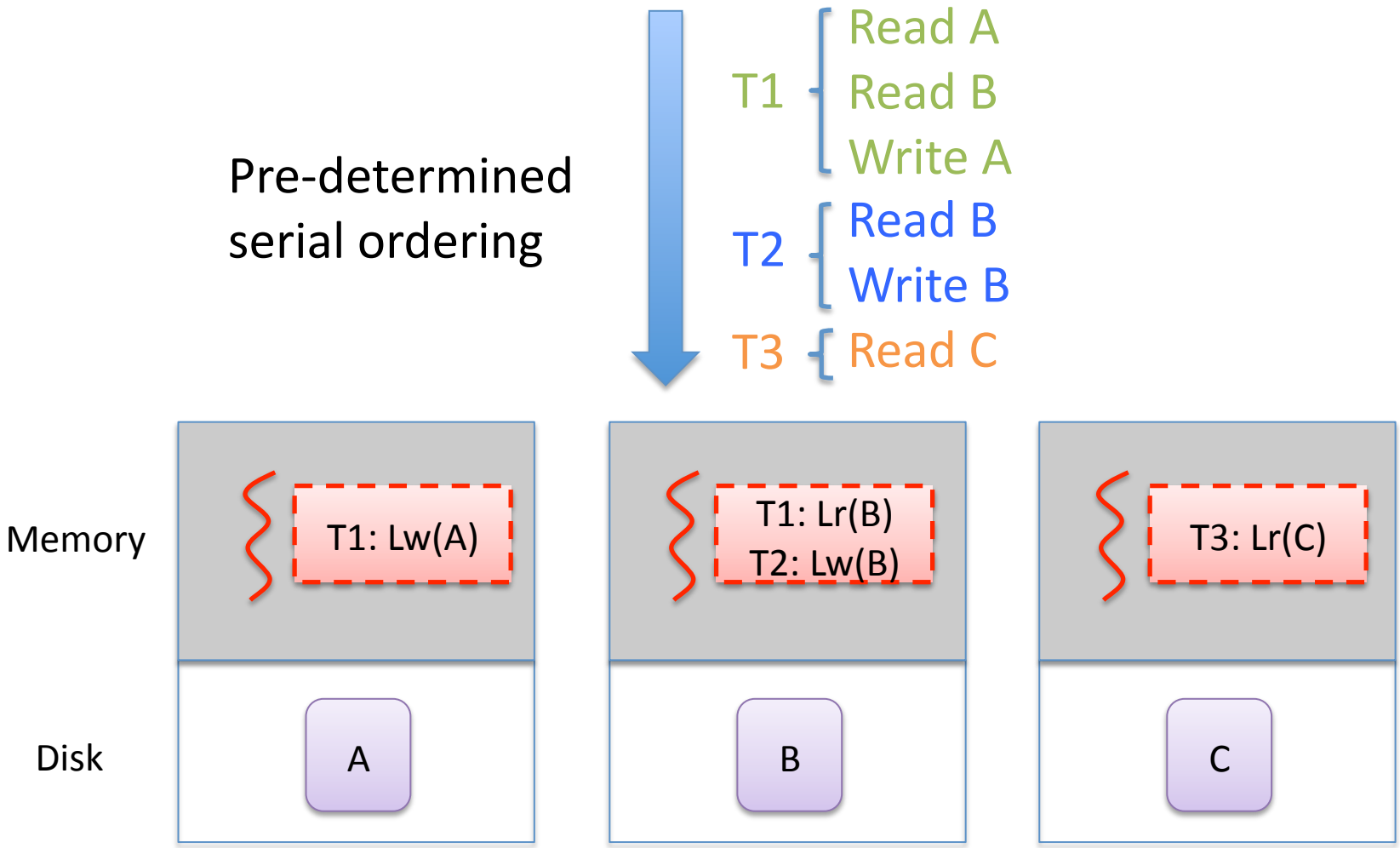
Scheduler and deterministic locking

- Logical view of records
- Global transaction order for its own share
- Only responsible for locking locally stored data
- Single-threaded locking manager
  - Resemble 2-phase locking
  - **Enforce transaction ordering from sequencers**
    - For conflicted updates
  - No deadlock

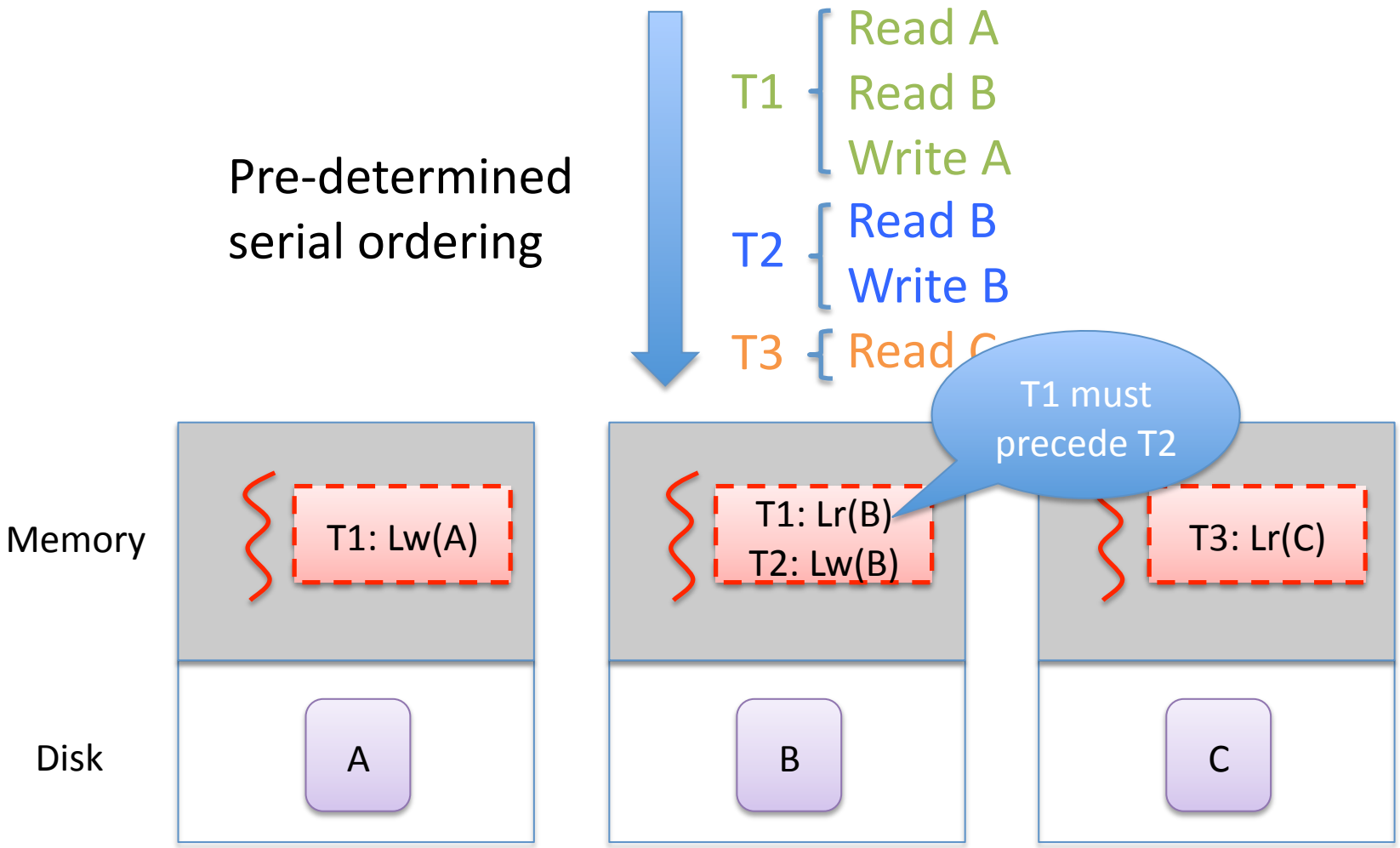
Deterministic Locking



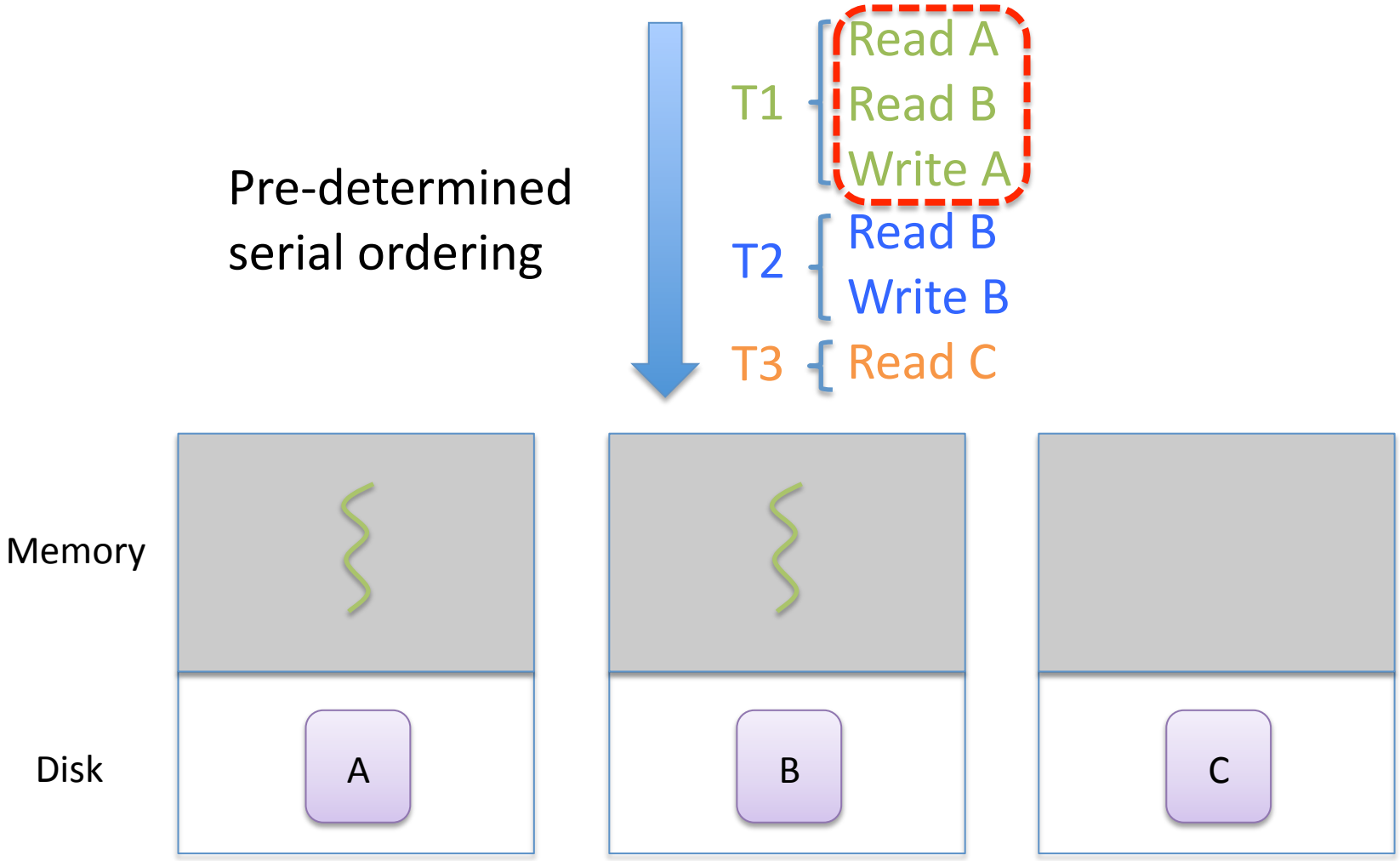
Deterministic Locking



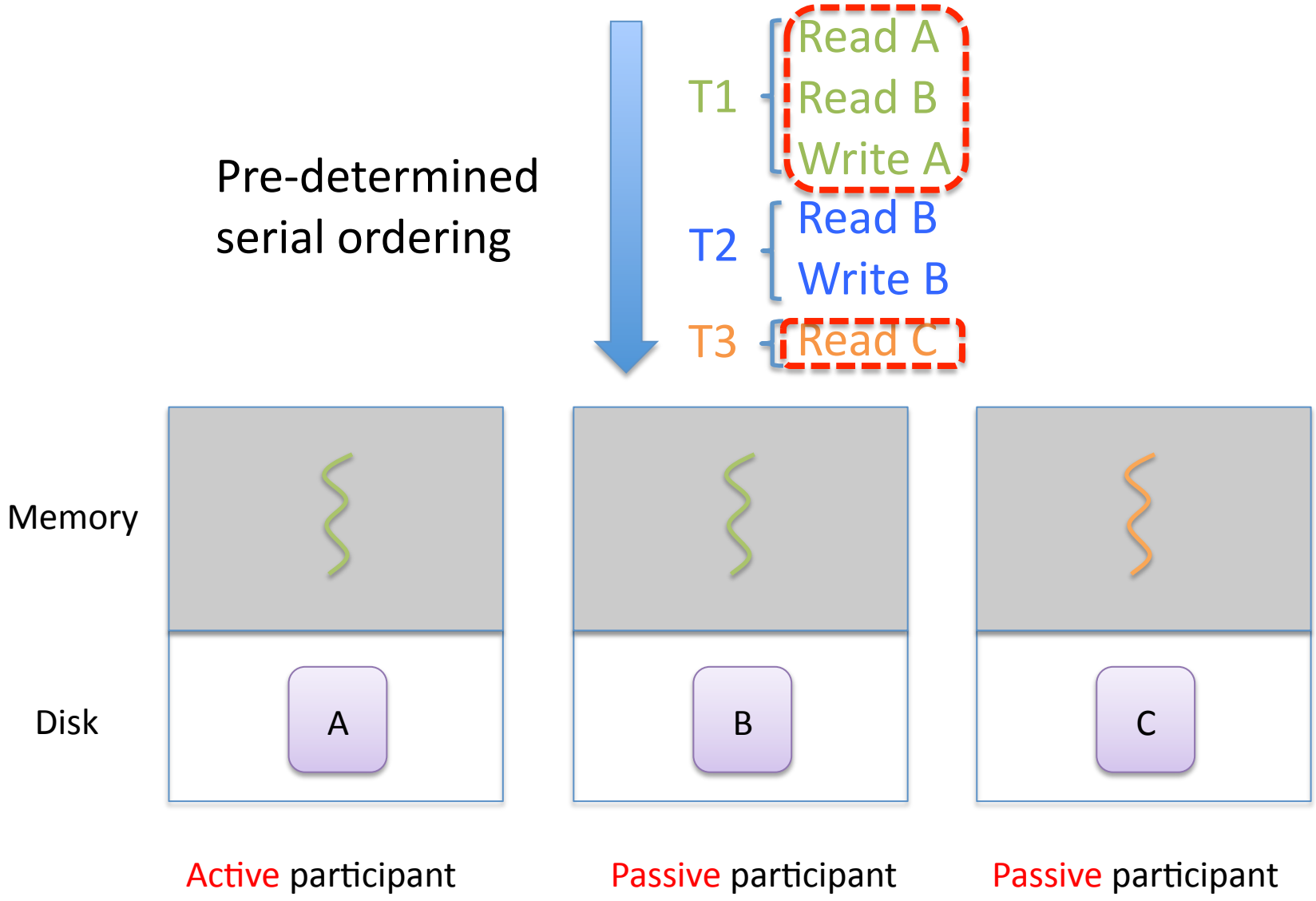
Deterministic Locking



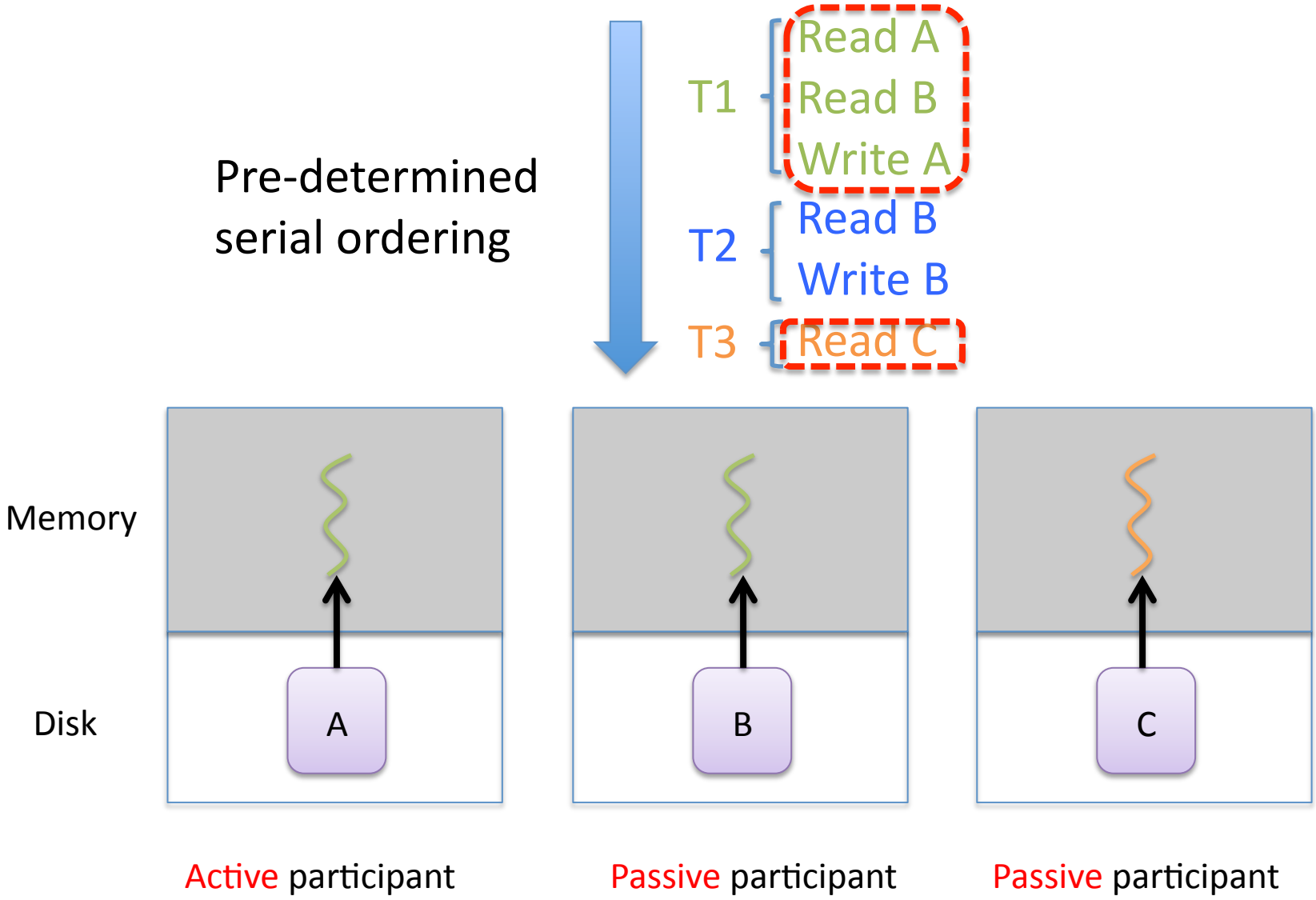
Worker execution



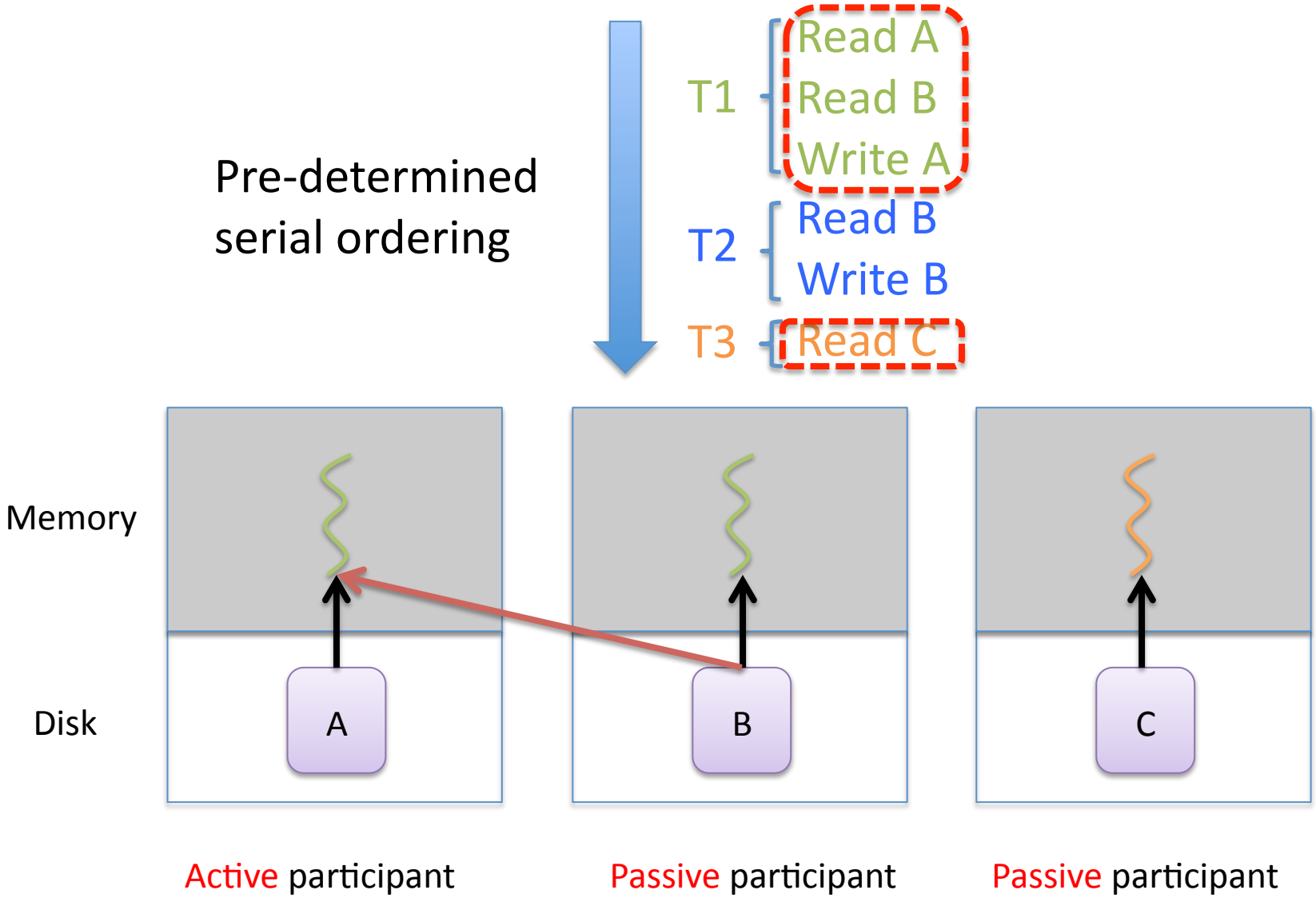
Worker execution



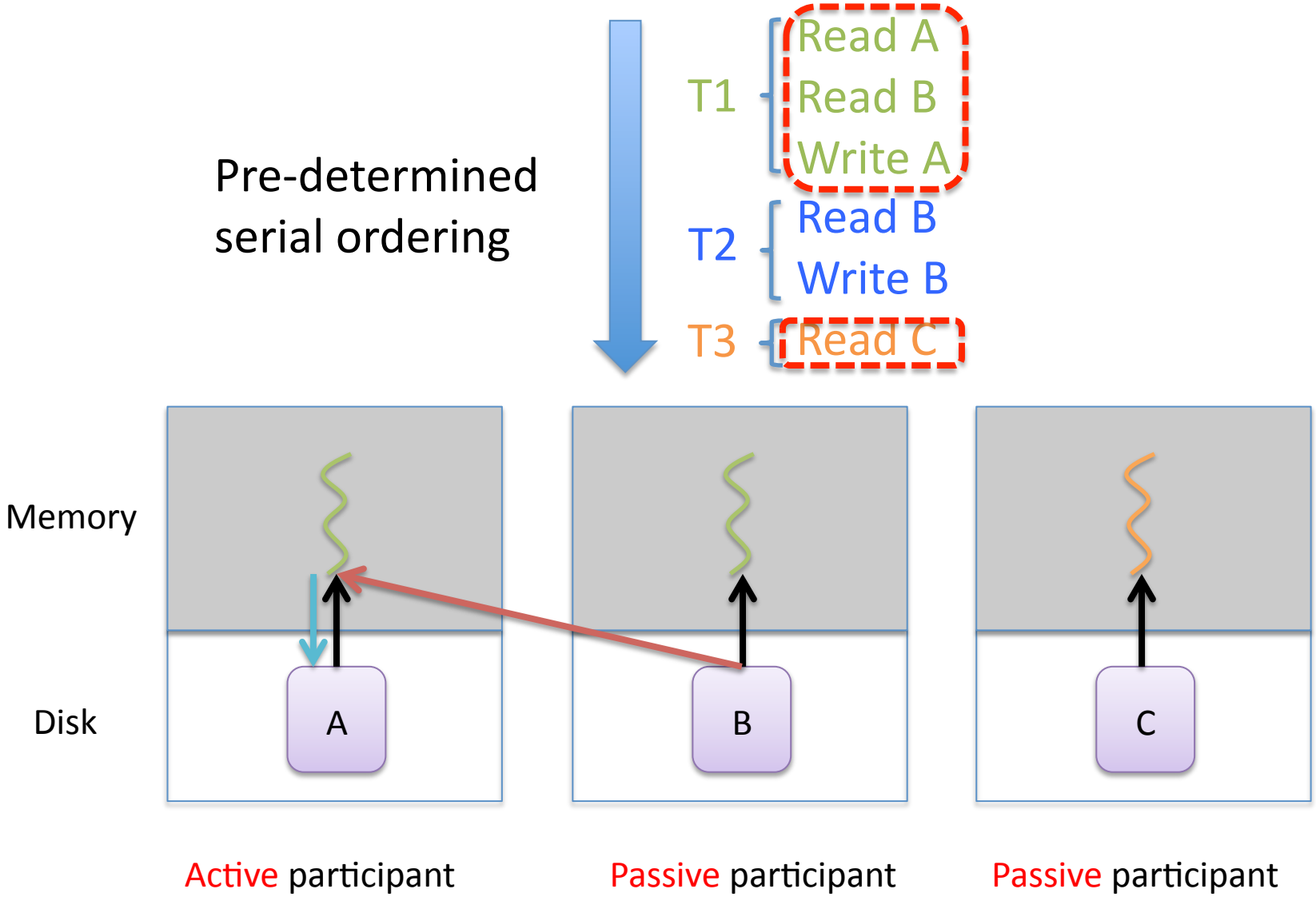
Worker execution



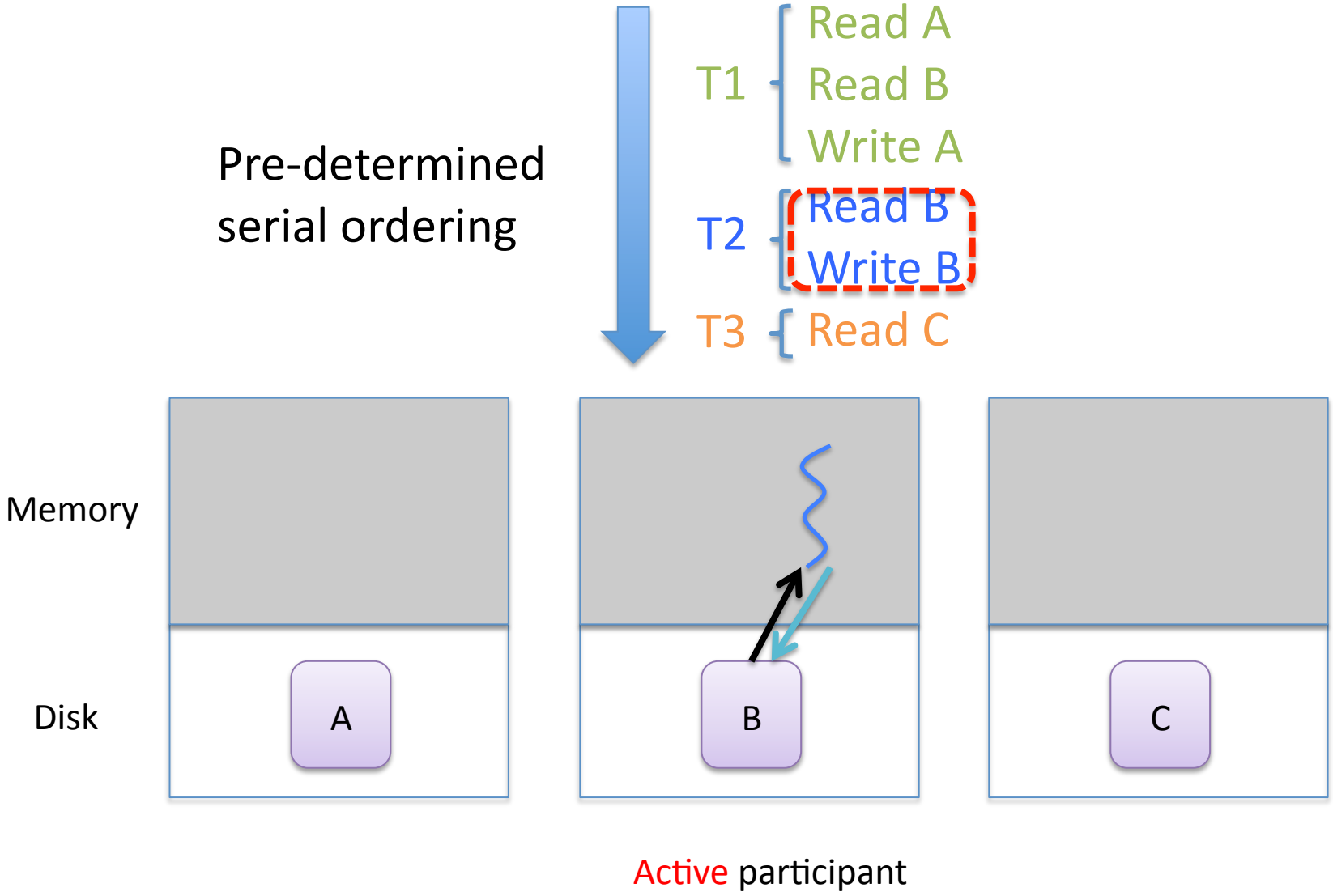
Worker execution



Worker execution



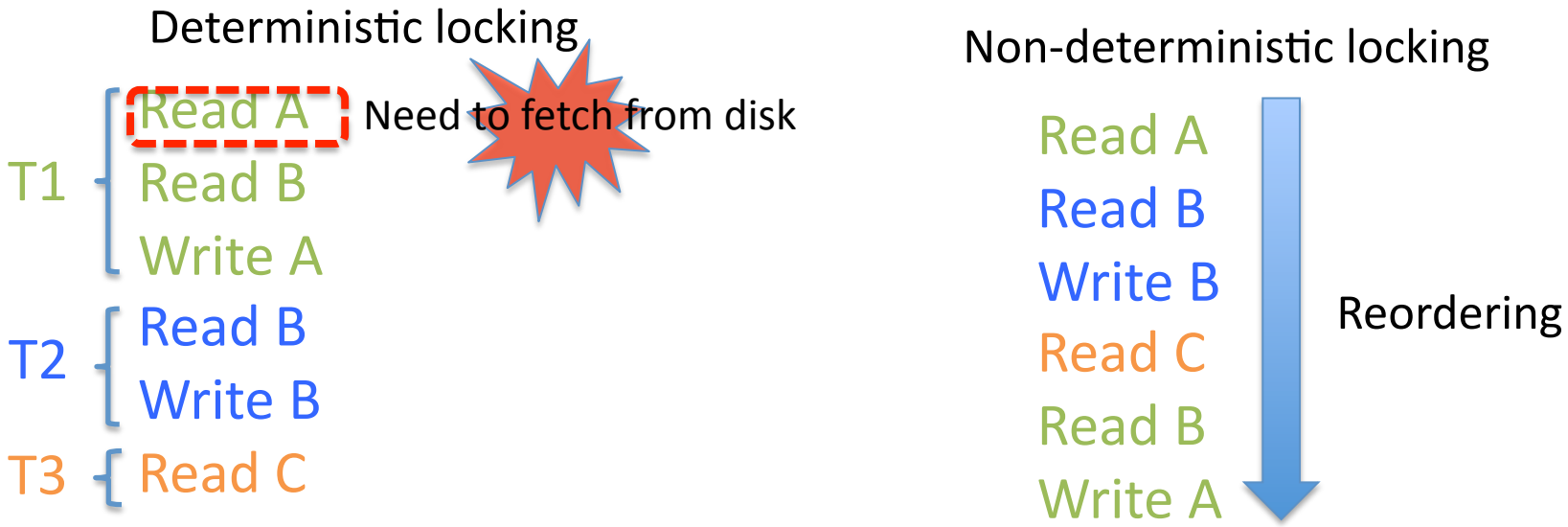
Worker execution



Problem of deterministic locking

- ***T2 cannot proceed*** because T1 is block waiting on disk access to A
- Even if T1 does NOT acquire any lock for B yet
- T3 can still proceed

Problem of deterministic locking



- ***T2 cannot proceed*** because T1 is block waiting on disk access to A
- Even if T1 does NOT acquire any lock for B yet
- T3 can still proceed

Calvin's Solution

- ***Delay*** forwarding transaction requests
  - Prefetech data from disk
  - Hope: most (99%) transactions execute without need to access disk

- Problem
  - How long to delay?
    - Need to estimate disk and network latency
  - Tracking all in-memory records
    - Need a scalable approach
  - Future work

Checkpointing

- Fault-tolerance made easier in Calvin
  - Instant failover
  - Only log transactional inputs, no REDO log

- Checkpointing for ***fast recovery***
  - Failure recovery from a recent state
  - Rather than from scratch

Checkpointing (cont.)

- **Naïve synchronous checkpointing**
  - Freeze one replica only for checkpointing
  - Difficult to bring the frozen replica up-to-date
- **Asynchronous variation of Zigzag**
  - Zigzag
    - 2 copies per replica, 2 times memory footprint
    - 1 copy is up-to-date, the other is the latest checkpoint
    - Need to stop completely to ensure checkpoint consistency
  - Calvin's optimizations
    - Pre-specified virtual consistency point, no need to stop the system
    - Copy-on-write only during checkpointing, reduce memory footprint
- **Asynchronous snapshot mode**
  - Storage layer needs to support multiversioning

Scalability

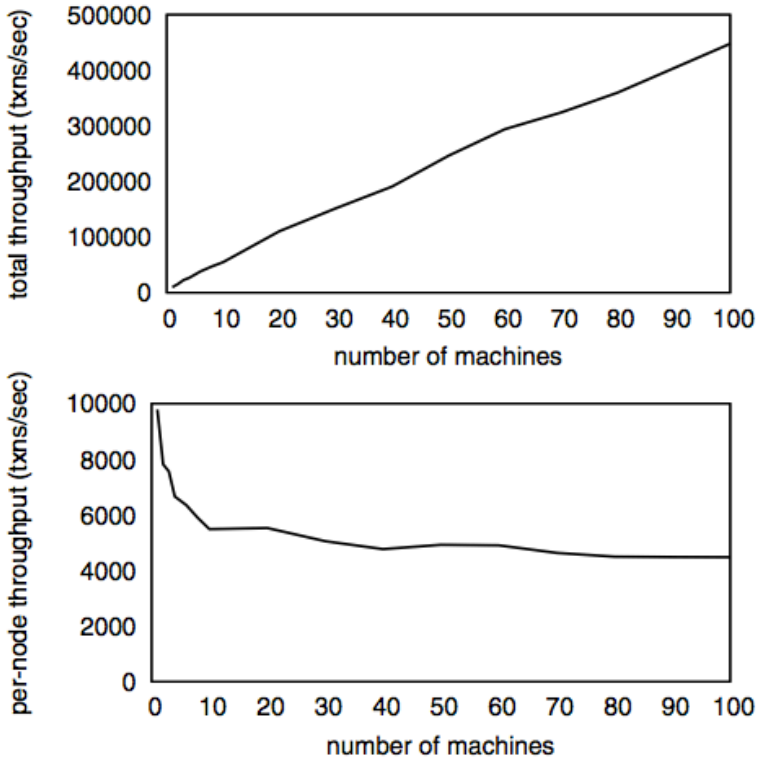
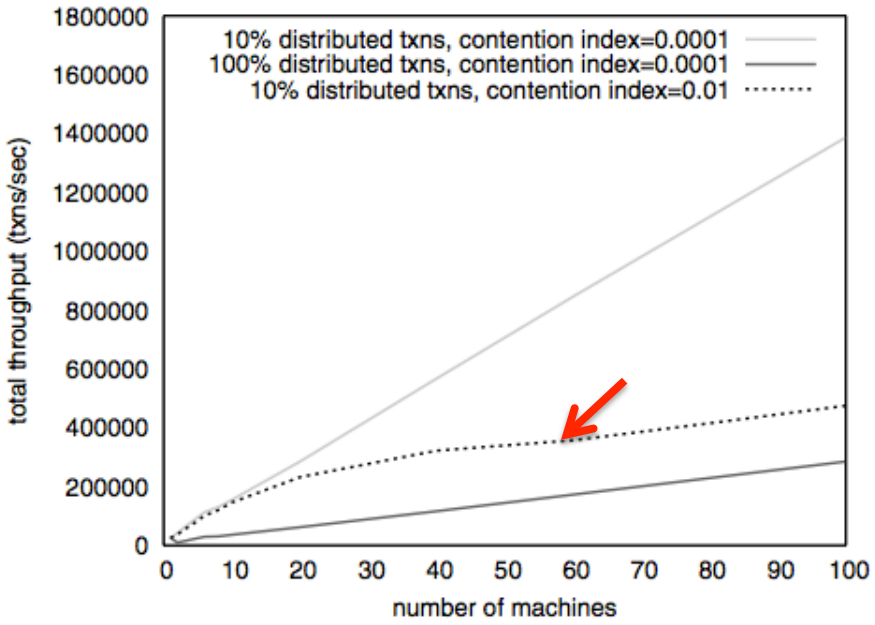
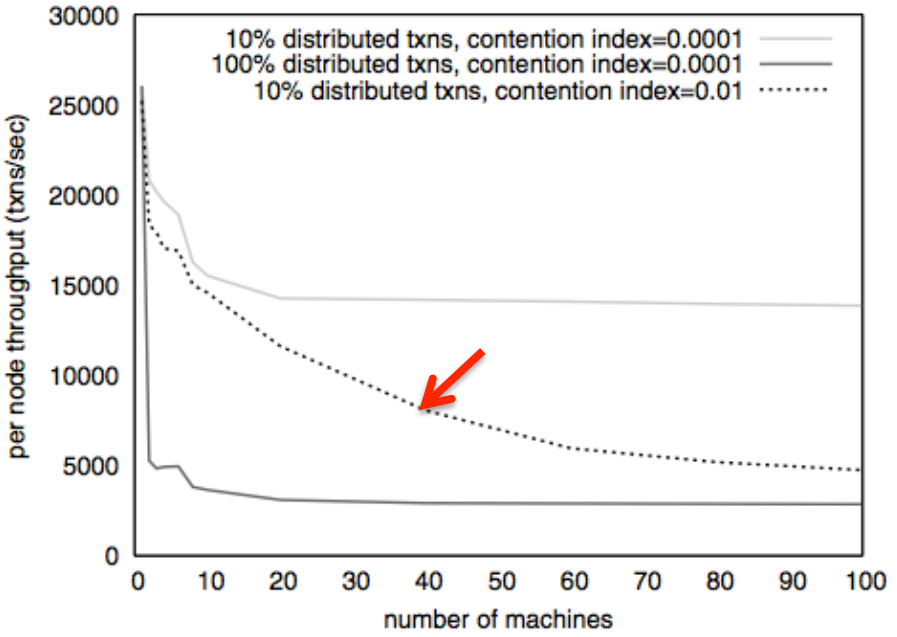


Figure 4: Total and per-node TPC-C (100% New Order) throughput, varying deployment size.

- Calvin **scales (near-)linearly**
- Throughput **comparable to world-record**
  - With much cheaper hardware

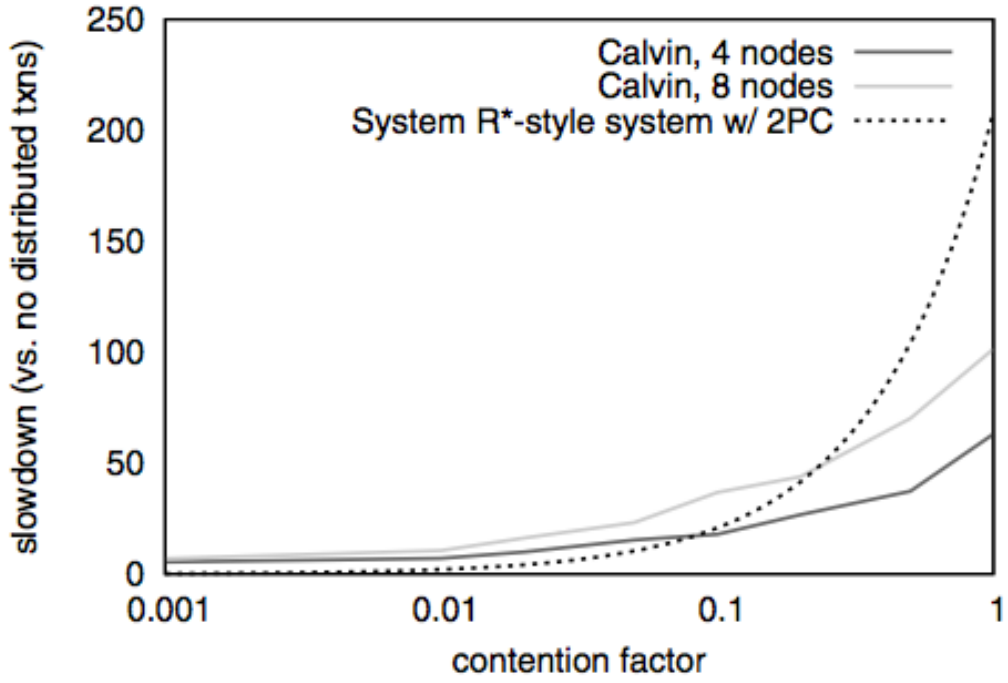
Scalability (cont.)





- Scales linearly for low contention workloads
- Scales sub-linearly when contention is high
  - **Stragglers** (slow machines, execution process skew)
  - Exacerbated with higher contention

Scalability under high contention



Calvin scales **better than 2PC** in face of high contention