Systems Software Research Group

HiperTM: High Performance Fault-Tolerant Transactional Memory

Sachin Hirve, Roberto Palmieri, and **Binoy Ravindran**

Systems Software Research Group
Virginia Tech
ssrg.ece.vt.edu

Motivation

- **STM: A promising programming model for general purpose concurrency control**
- Ensures Atomicity, Consistency and Isolation properties
- In-memory transaction processing provides high throughput

- Fault-tolerance is highly desirable for such systems
  - Node failure or system crash results in loss of data and service interruption
- Fault-tolerance through data replication [Guerraoui , 96]
  - Immunity to faults, as failure of one node is tolerated by other replicas

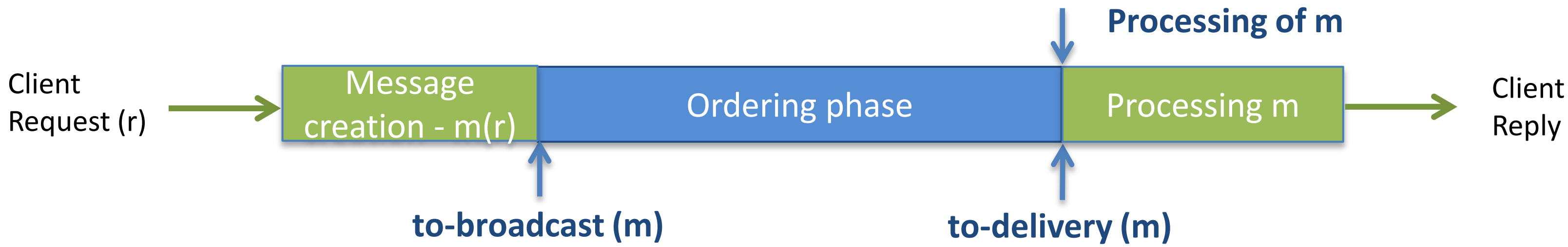
Taxonomy of Replication Models

- **Partial replication:** Data is replicated on subset of nodes [Serrano, 07]
  - Amount of data and system size can scale
  - Only a subset of nodes takes part in co-ordination phase
  - Remote communication for retrieving and committing objects
- **Full replication:** Data is replicated on all nodes [Schneider, 90]

Certification-based replication [Kemme, 98]	Active replication [Schneider, 93]
– With low conflicts, high scalability and performance (within ordering layer's scalability bottleneck) – Compatible with legacy TM/DB programming model – Performance is conflict-dependent – Performance is impacted by message size; batching is out of scope	– Performance is conflict-independent – Local transaction execution – Full failure masking

Active Replication

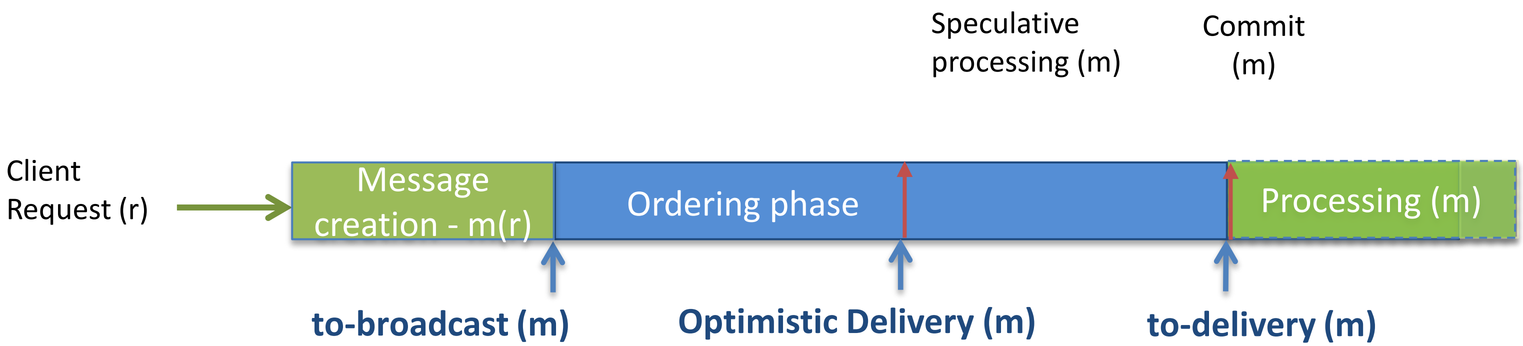
- Transaction execution is post-ordering
- Each node/replica executes same set of requests in same order
  - Same sequence of updates on objects, despite failures
- Benefits
  - High performance: Local execution of requests
  - Full failure masking
- Drawbacks
  - Scalability hampered due to ordering layer
  - Co-ordination phase and execution phase are serialized



Optimistic Atomic Broadcast (OAB)

[Pedone, 03]

- With high probability, messages broadcast in a LAN are received totally ordered
  - Exploit broadcast message to maximize concurrent processing of ordering and processing phase
- Final order can differ from earlier broadcast order (message re-ordering)
  - E.g., if the sequencer crashes mid-consensus and new sequencer creates a new order based on previously broadcast message



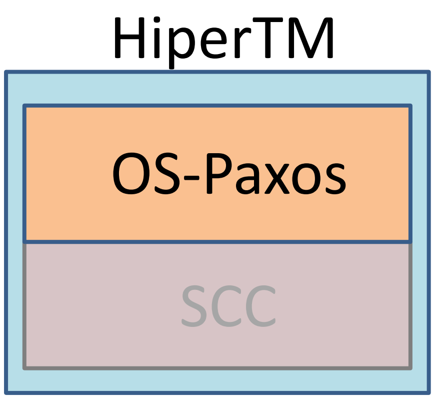
Design Goals

- Maximize the overlap of ordering and execution phases
  - Exploit knowledge of probable order during ordering phase
- Eliminate message re-ordering in failure-free executions

- Building a Concurrency Control (CC) such that it:
  - Enforces the request order received from AB
  - Is independent from contention level
  - Ensures abort-free processing of read-only transactions

Building blocks of HiperTM

- **OS-Paxos**
  - Optimistic ordering layer built on S-Paxos
- **SCC**
  - Speculative Concurrency Control (a transaction processing layer)

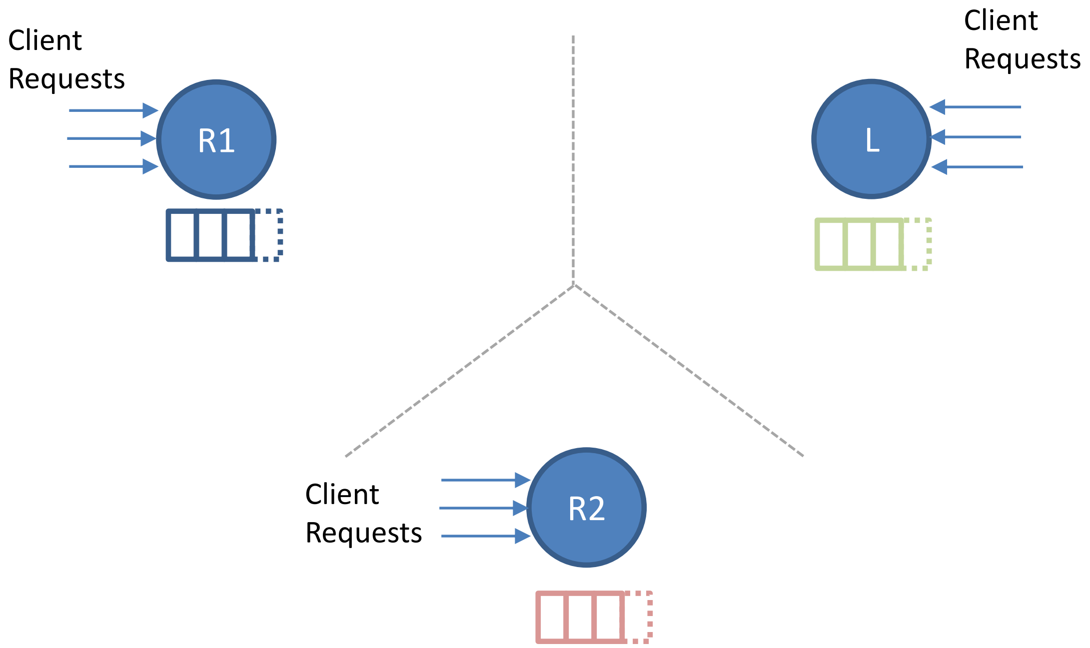


OS-Paxos

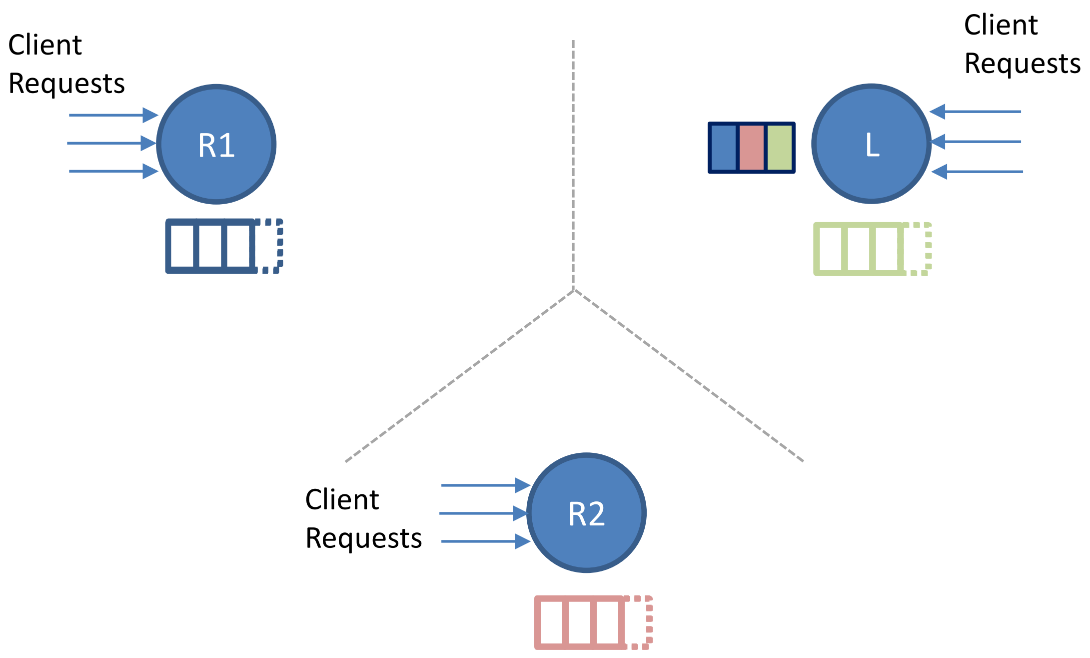
- **Protocol overview**
  - Replicas[1] receive client requests and creates batches
  - Request batches are uniform broadcast to other replicas
  - Leader creates an order for received batches and gathers consensus from other replicas
  - Optimistic delivery (*oDeliver*) is issued on to-broadcast of the order
  - Final delivery (*aDeliver*) is issued on to-delivery of the order

[1] number of replicas is 2*f*+1, and at most *f* replicas may crash

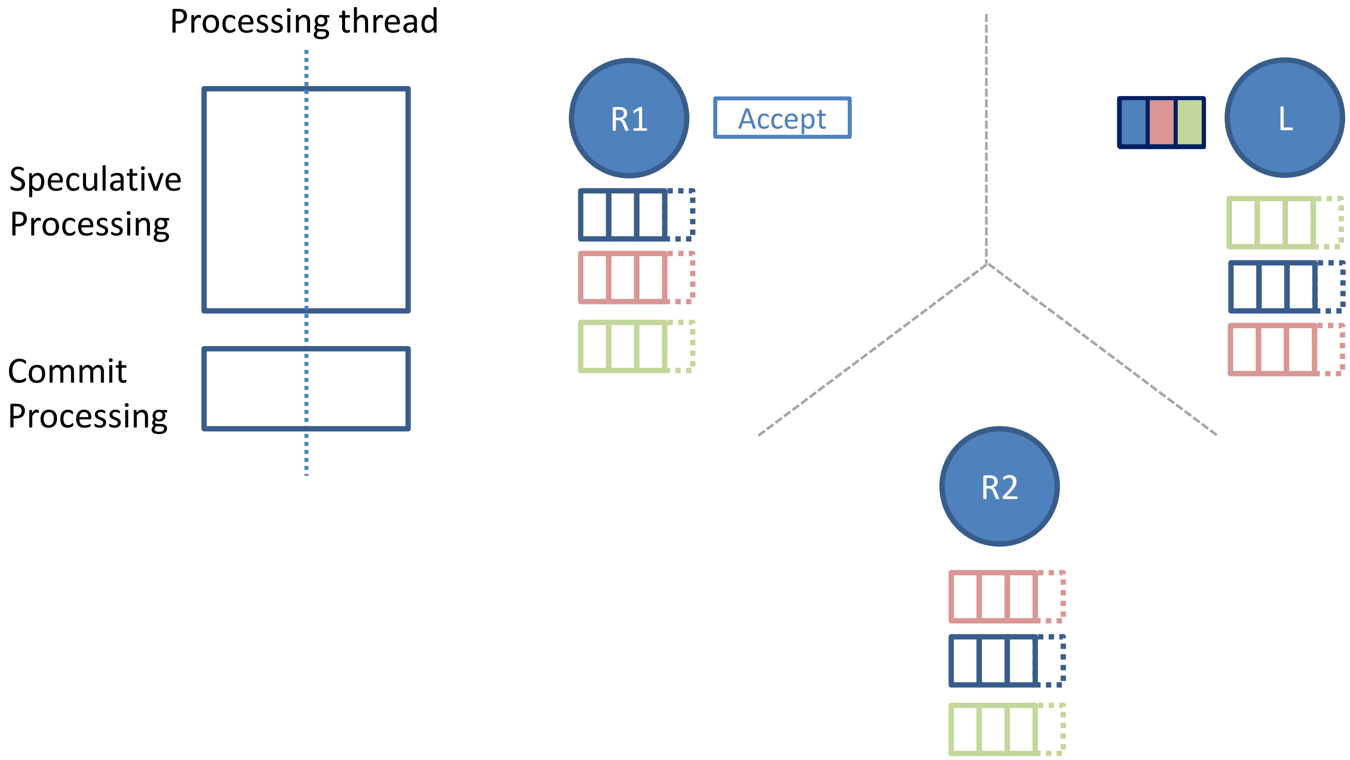
OS-Paxos Illustration: Request batch formation



OS-Paxos Illustration: Batch propagation and order proposal

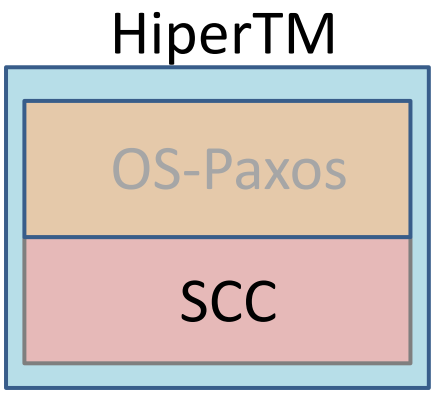


OS-Paxos Illustration: Optimistic delivery

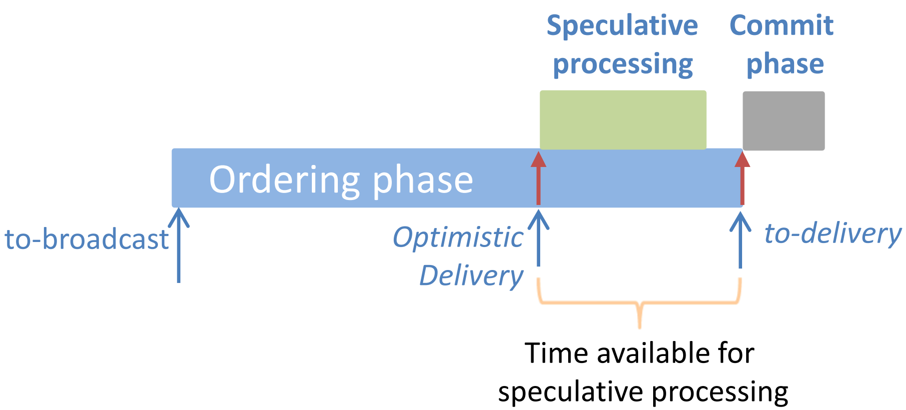


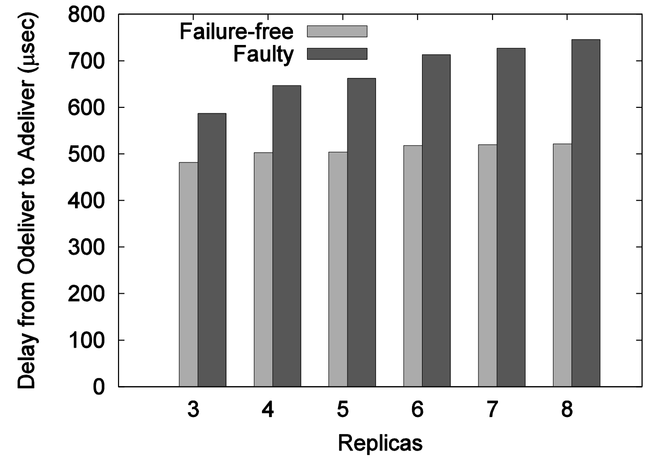
Building blocks of HiperTM

- OS-Paxos
  - Optimistic ordering layer built on S-Paxos
- **SCC**
  - Speculative Concurrency Control (a transaction processing layer)



SCC: Speculative Concurrency Control





- Limited delay between optimistic delivery and final order
  - Expensive synchronization for concurrent processing of optimistically delivered order
- Design:
  - Single-threaded processing for write transactions
  - Local multi-threaded processing for read transactions

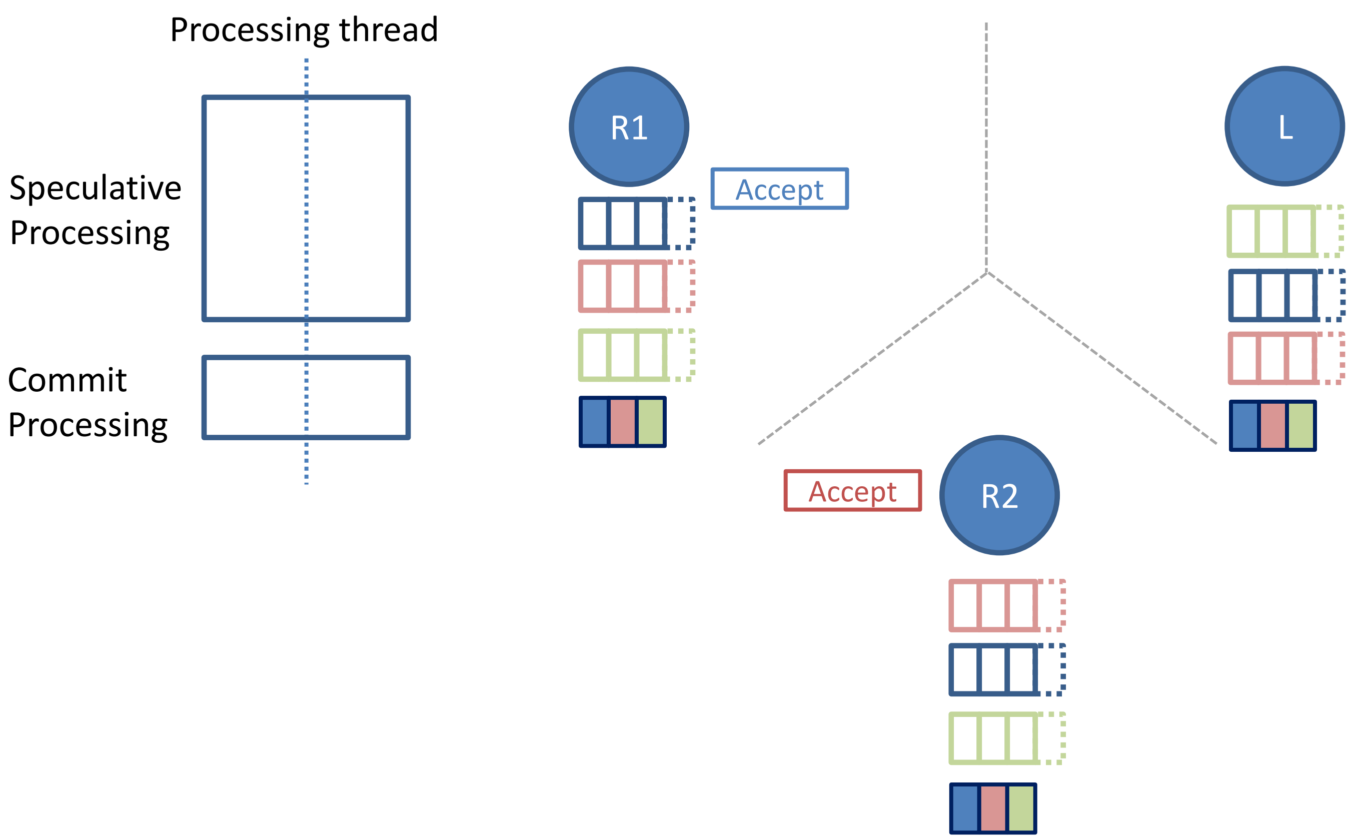
SCC: continued…

- Objects stored in a multi-version data –structure
- Replica timestamp is incremented by committing transaction

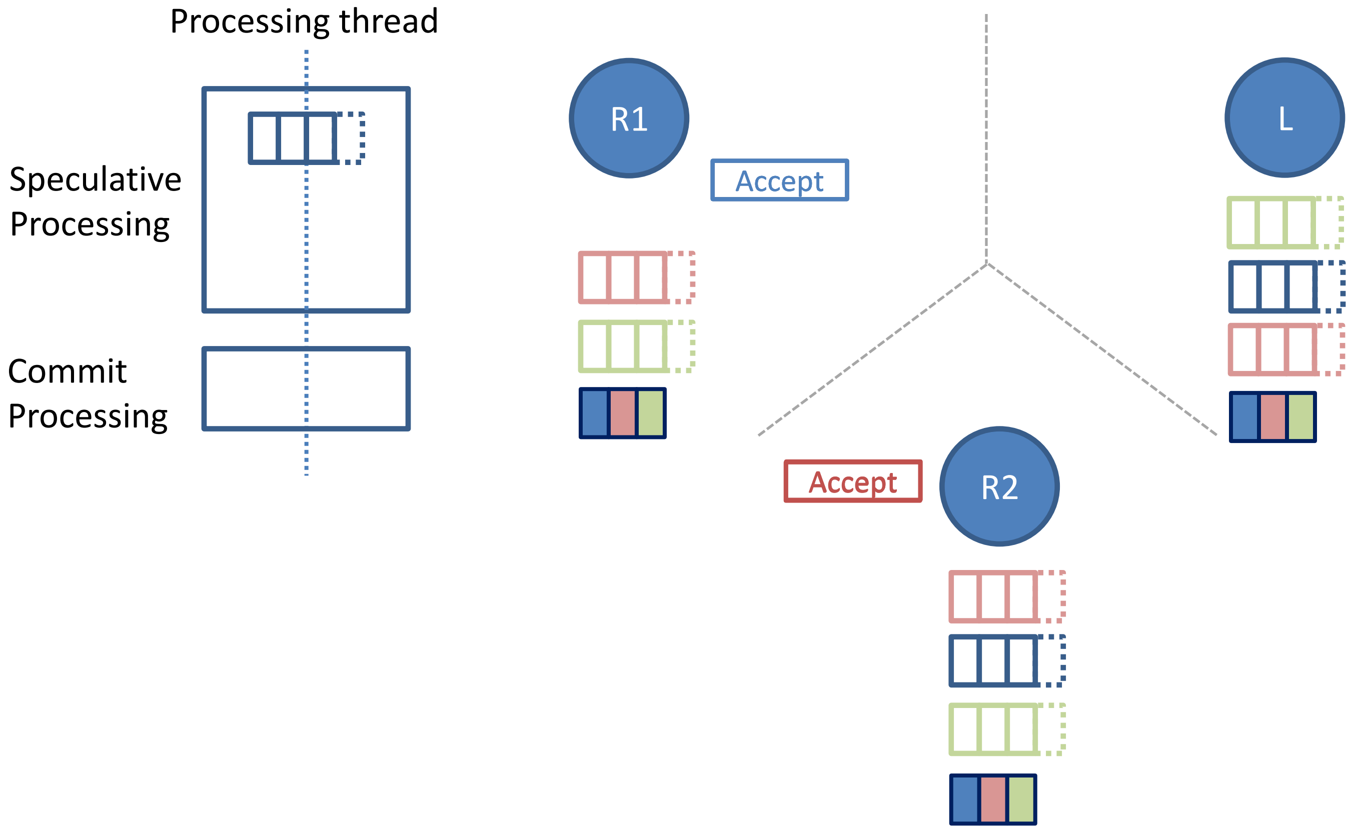
- **Execution of write transactions**
  - Arrive through OS-Paxos layer
  - Single thread processing:
    - Speculative processing on *oDeliver*
    - Commit of write-set on *aDeliver*
  - On commit, a new timestamp is attached to committing objects

- **Execution of read requests**
  - Execution using thread pool:
    - Acquires replica timestamp at start
    - Latest objects are accessed w.r.t. transaction-timestamp

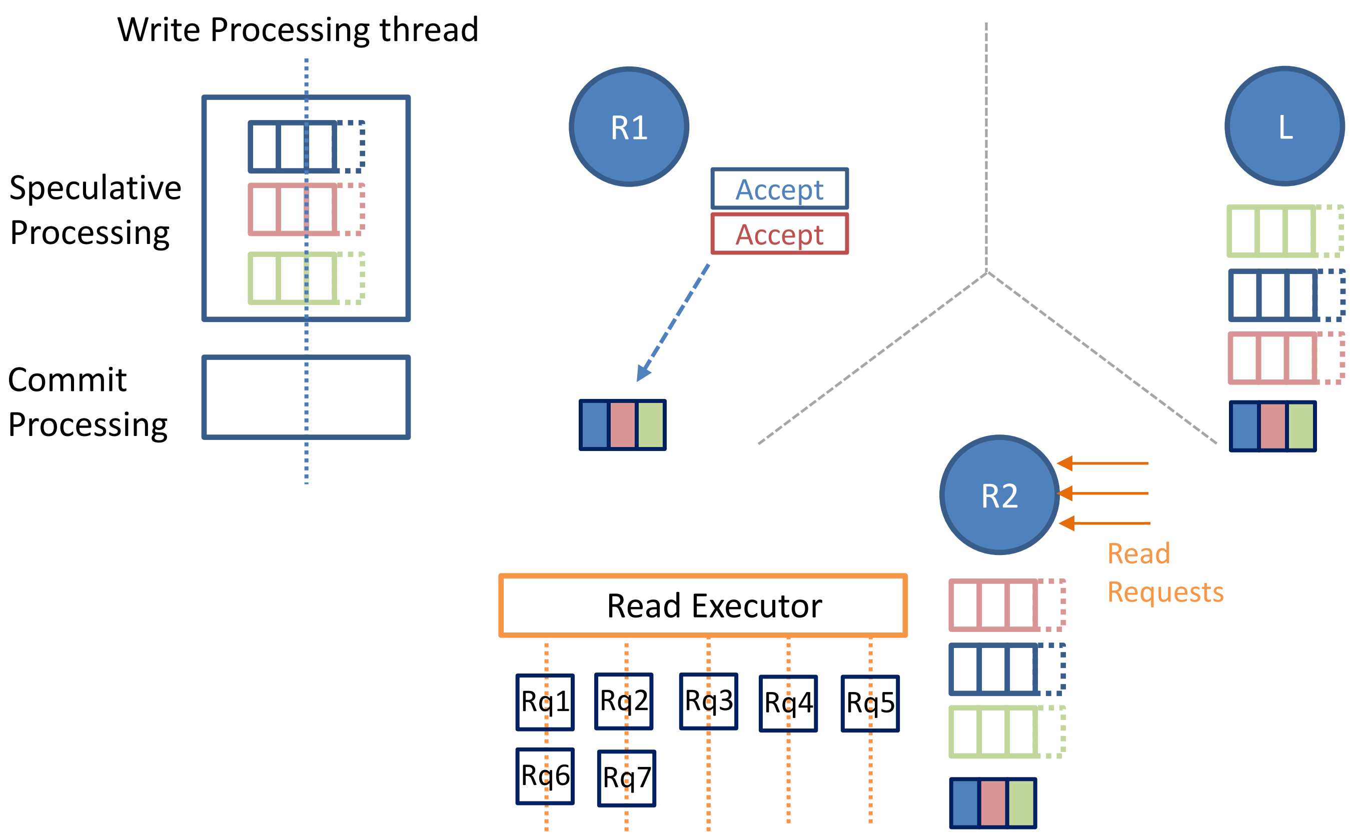
SCC Illustration: speculative processing and consensus



SCC Illustration: consensus in progress



SCC Illustration: Committing write and read processing



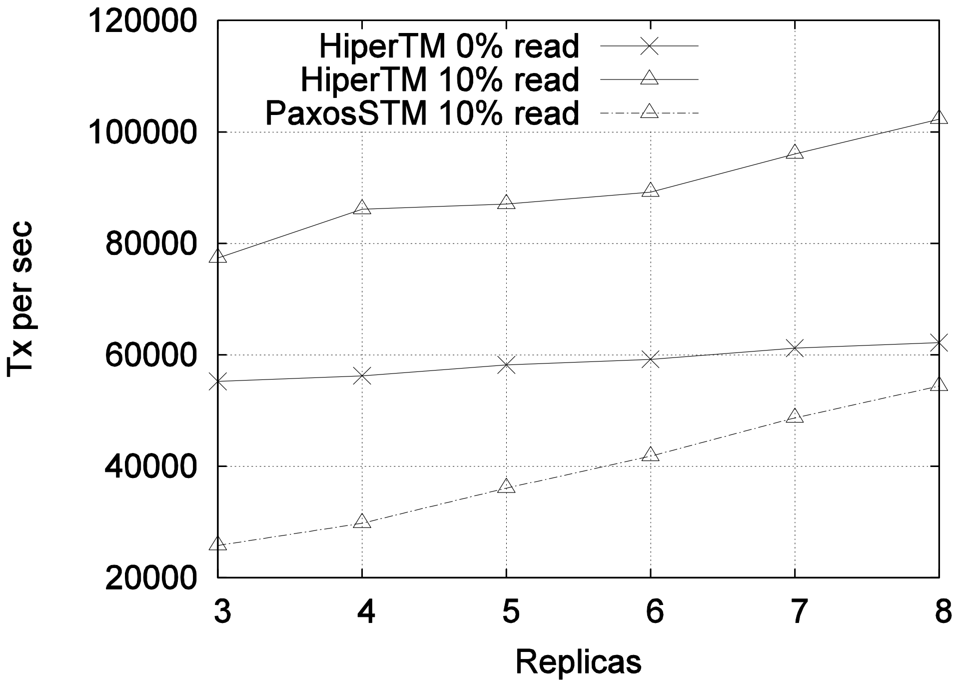
Properties

- 1-copy serializability
- Opacity
- Lock-freedom
- Abort-freedom for read-only transactions

Evaluation

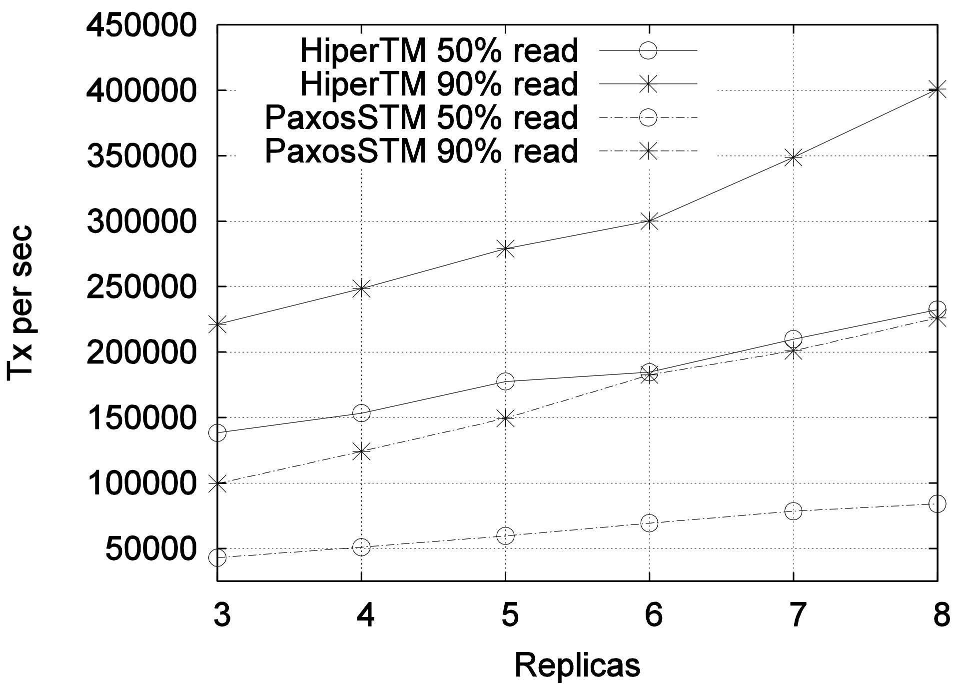
- **Test-bed consists of 8 nodes**
  - AMD Opteron machines
    - 4 nodes with 64-cores and 2.3GHz speed
    - 4 nodes with 48-cores and 1.7GHz speed
  - 1Gb/s switched network
- **Benchmarks**
  - Bank: A micro-benchmark emulating a bank application
  - TPC-C: A well known OLTP benchmark
- **Competitors**
  - PaxosSTM [Kobus, 12]: Certification-based with full replication
  - Score [Peluso, 12]: A partial replication-based DTM protocol ensuring abort-freedom of read-only transactions

Evaluation – Bank Benchmark



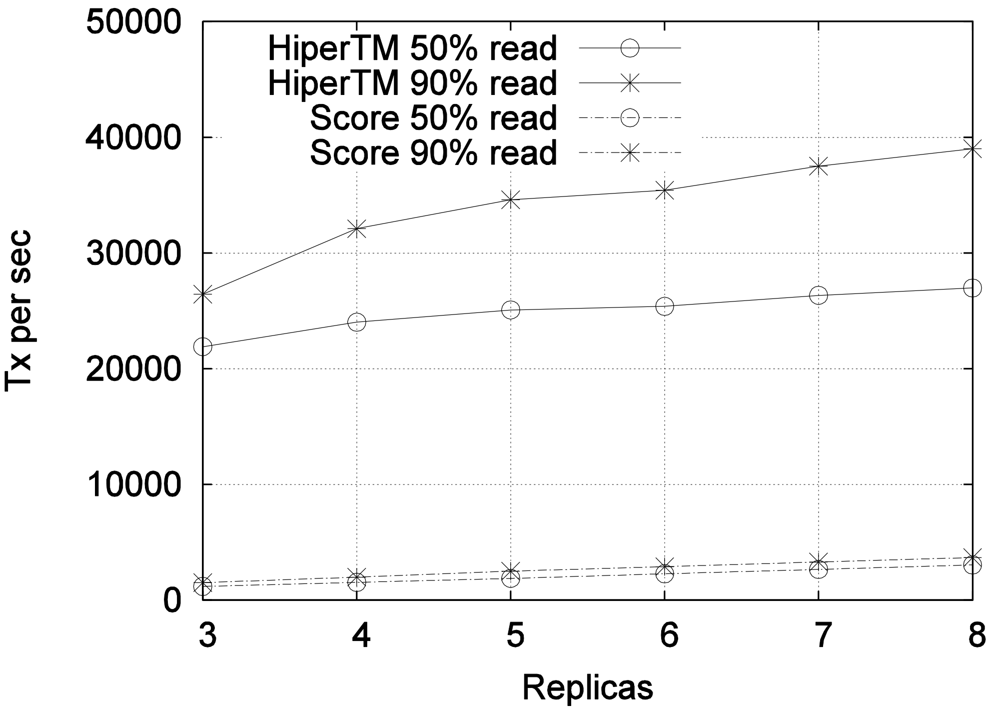
- 1000 bank accounts (conflict-intensive)
- Speculative processing is effective
  - Key is leveraging optimistic delivery
- Single-thread processing is effective
  - Better performance with less implementation complexity

Evaluation – Bank Benchmark



- Performance and system scalability increases as read-only transactions increase from 10% -to- 90%
- Maximum speed-up: ~1.2x

Evaluation – TPC-C Workload



- HiperTM (with 8 replicas) outperforms SCORe by up to 10x
  - SCORe's object look-ups degrades performance
- (Experiments with failures show up to 30% performance degradation before system stabilizes again)

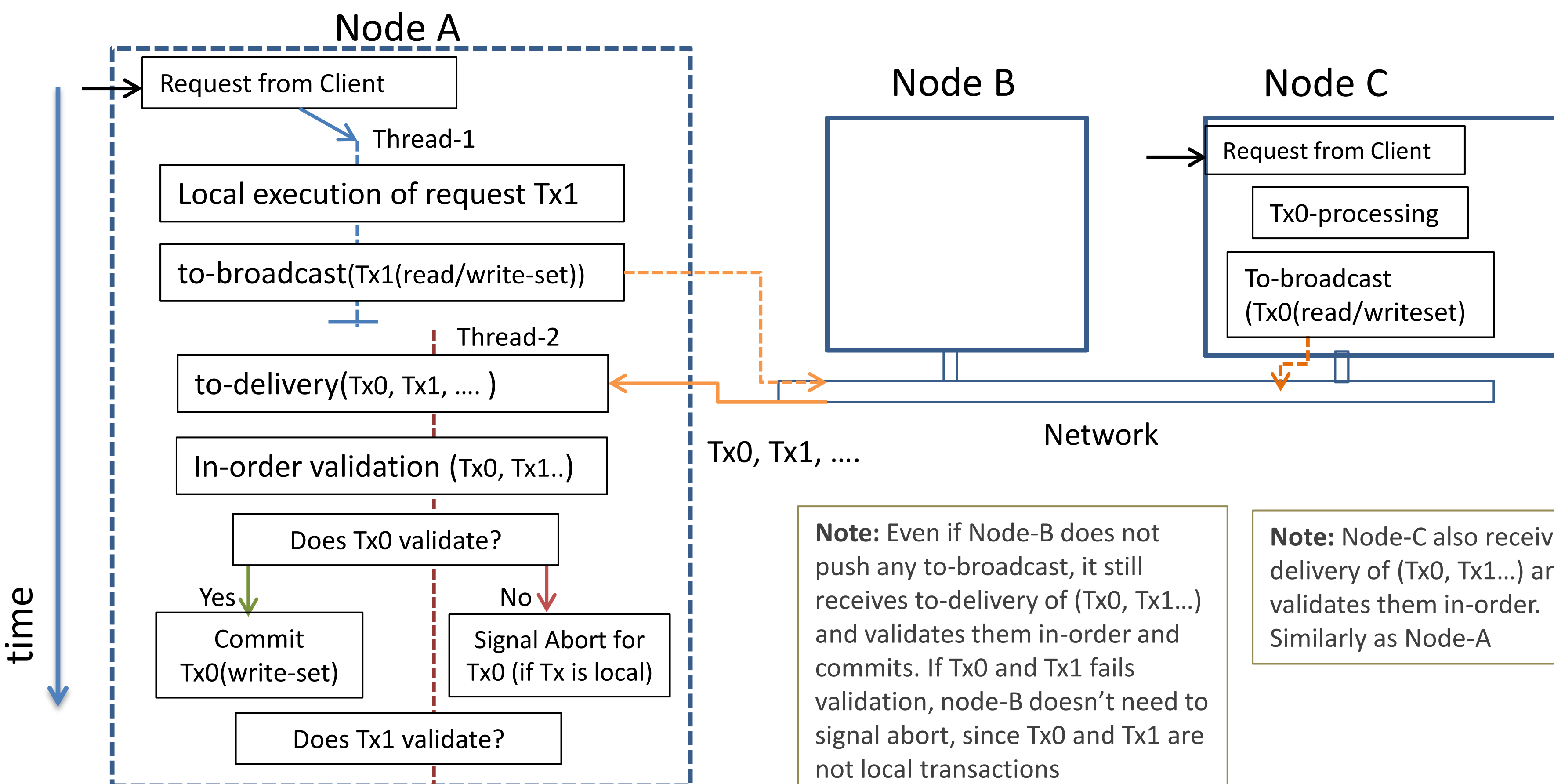
Conclusions

- **Optimism pays off**
  - Speculative transaction execution partially hides total-order latency
  - Serial execution of writes is effective
  - Multi-versioning needed for abort-freedom of read-only

- **Implementation matters**
  - Important insights; pre-requisite for any transitions
  - Number of design decisions affect performance; involve tradeoffs
  - E.g., avoid costly synchronization mechanisms; optimizations to counter network non-determinism

Certification-based replication

How it works?



Note: Even if Node-B does not push any to-broadcast, it still receives to-delivery of (Tx0, Tx1…) and validates them in-order and commits. If Tx0 and Tx1 fails validation, node-B doesn't need to signal abort, since Tx0 and Tx1 are not local transactions

Note: Node-C also receives to-delivery of (Tx0, Tx1…) and validates them in-order. Similarly as Node-A