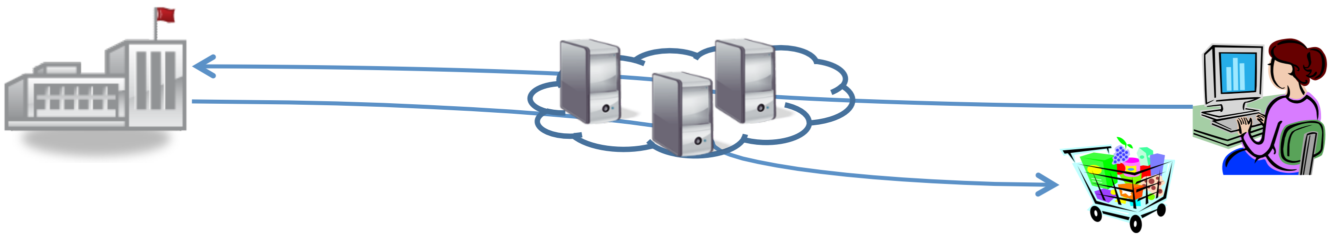

# Speculative Client Execution in Deferred Update Replication

Balaji Arun, Sachin Hirve, Roberto Palmieri, Sebastiano Peluso and Binoy Ravindran

Systems Software Research Group, Virginia Tech
http://ssrg.ece.vt.edu

# Context



- Ubiquitous nature of On Line Transaction Processing workloads
- 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 ensures high availability
  - Immunity to faults, as failure of one node is tolerated by other replicas

# Replication Models

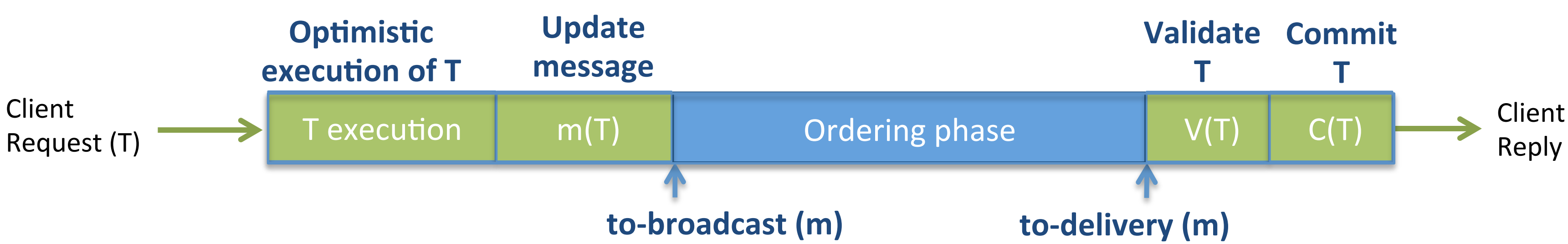
- **Partial replication:** Data is replicated on subset of nodes
  - Only a sub set of nodes takes part in co-ordination phase
  - Amount of data and system size can scale
  - Remote communication for retrieving and committing objects
- **Full replication:** Data is replicated on all nodes
  - Local transaction execution
  - Ordering layer required for ensuring replica consistency
  - Scaling of amount of data and system size is limited
  - Usual setup includes total-order based protocols, which are classified as
    - Deferred Update Replication (DUR)
    - Deferred Execution Replication (DER)

# Overview of Deferred Update Replication (DUR)

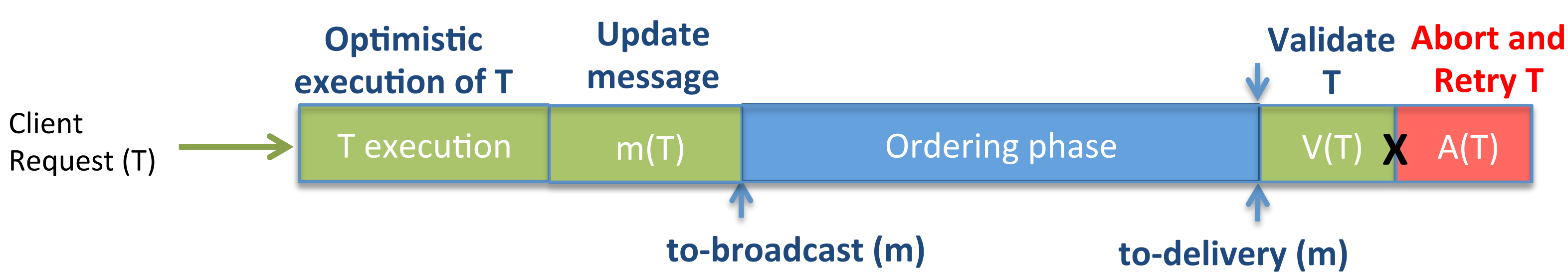
- Clients optimistically execute transactions and submit their updates to a global certification phase for commit
- Global certification phase:
  - Defines a common serialization order on all transaction updates
  - Validates the correctness of transaction execution according to serialization order
  - A transaction passes the validation if objects, it read, have not been modified by other transaction, before it commits

# DUR by Example

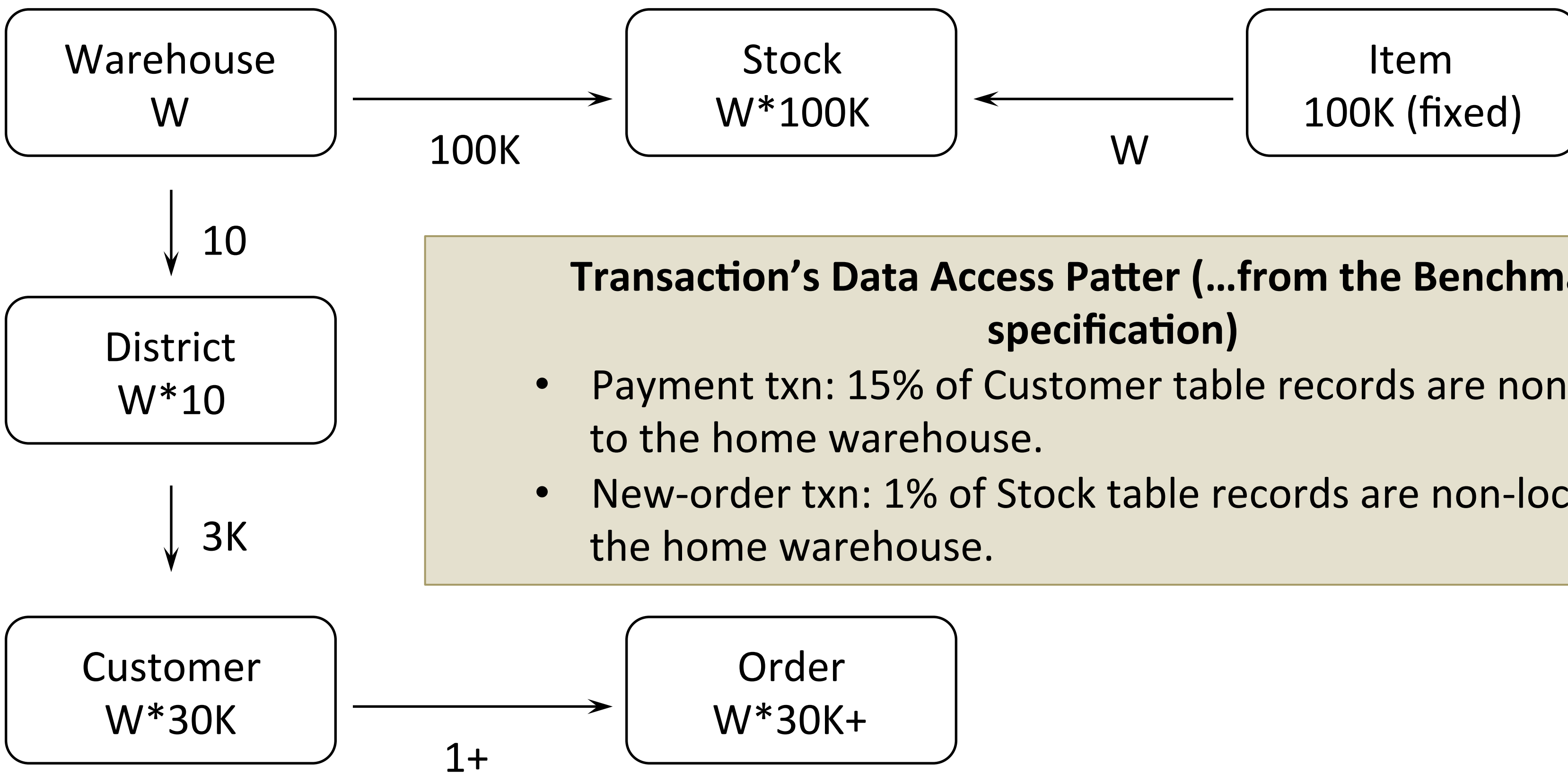
- Global certification phase (or Ordering phase):
  - On successful validation, object updates are committed




  - On failing the validation, object updates are discarded and transaction is re-executed



# The case study of TPC-C

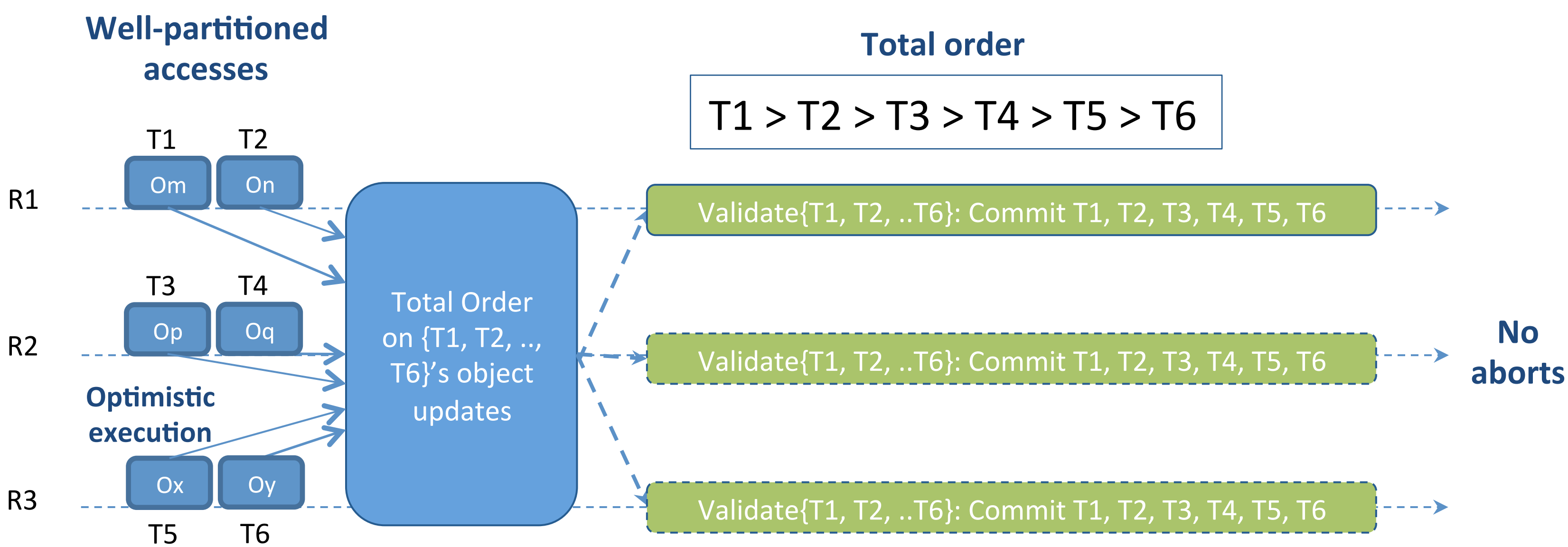



**Transaction's Data Access Patter (…from the Benchmark's specification)**

- Payment txn: 15% of Customer table records are non-local to the home warehouse.
- New-order txn: 1% of Stock table records are non-local to the home warehouse.

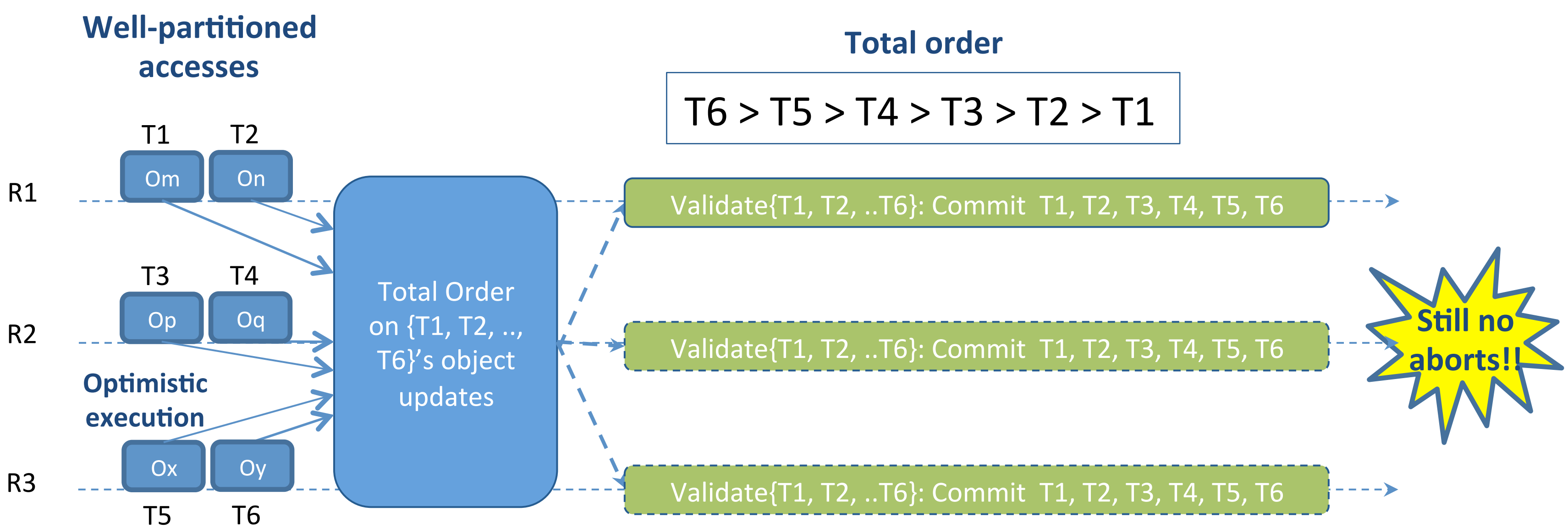
# The Best Case for DUR

- DUR benefits from massive parallelization of client threads
- In well partitioned accesses
  - Transactions running on different nodes rarely conflict



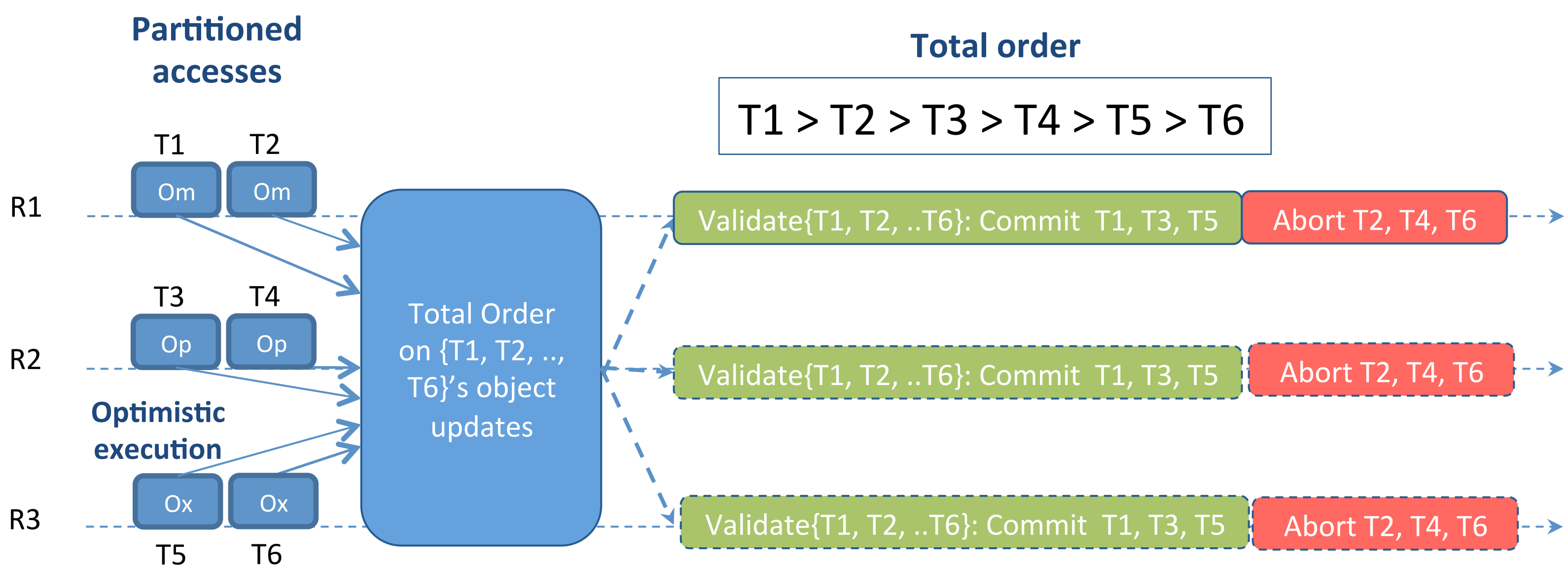
# The Best Case for DUR

- In well partitioned accesses
  - Even with different serialization order, transactions **may** not abort



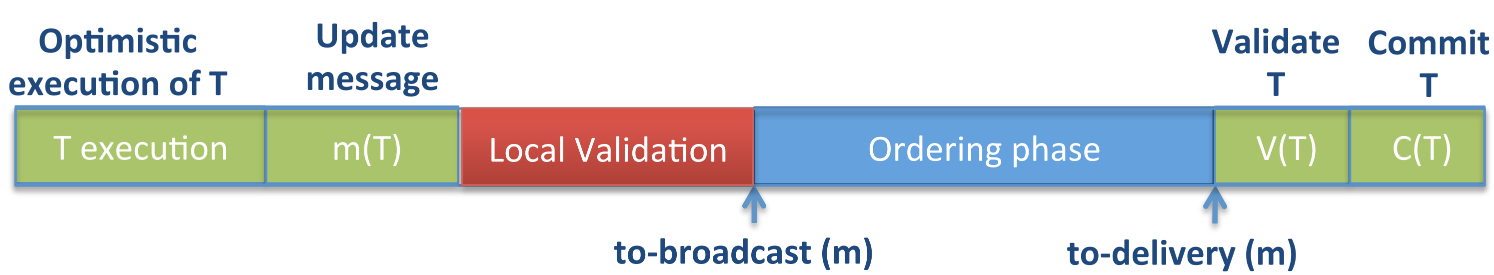
# …but even in the best case…

- In well partitioned accesses
  - Transactions running on same nodes suffer from aborts – Challenge!!!!

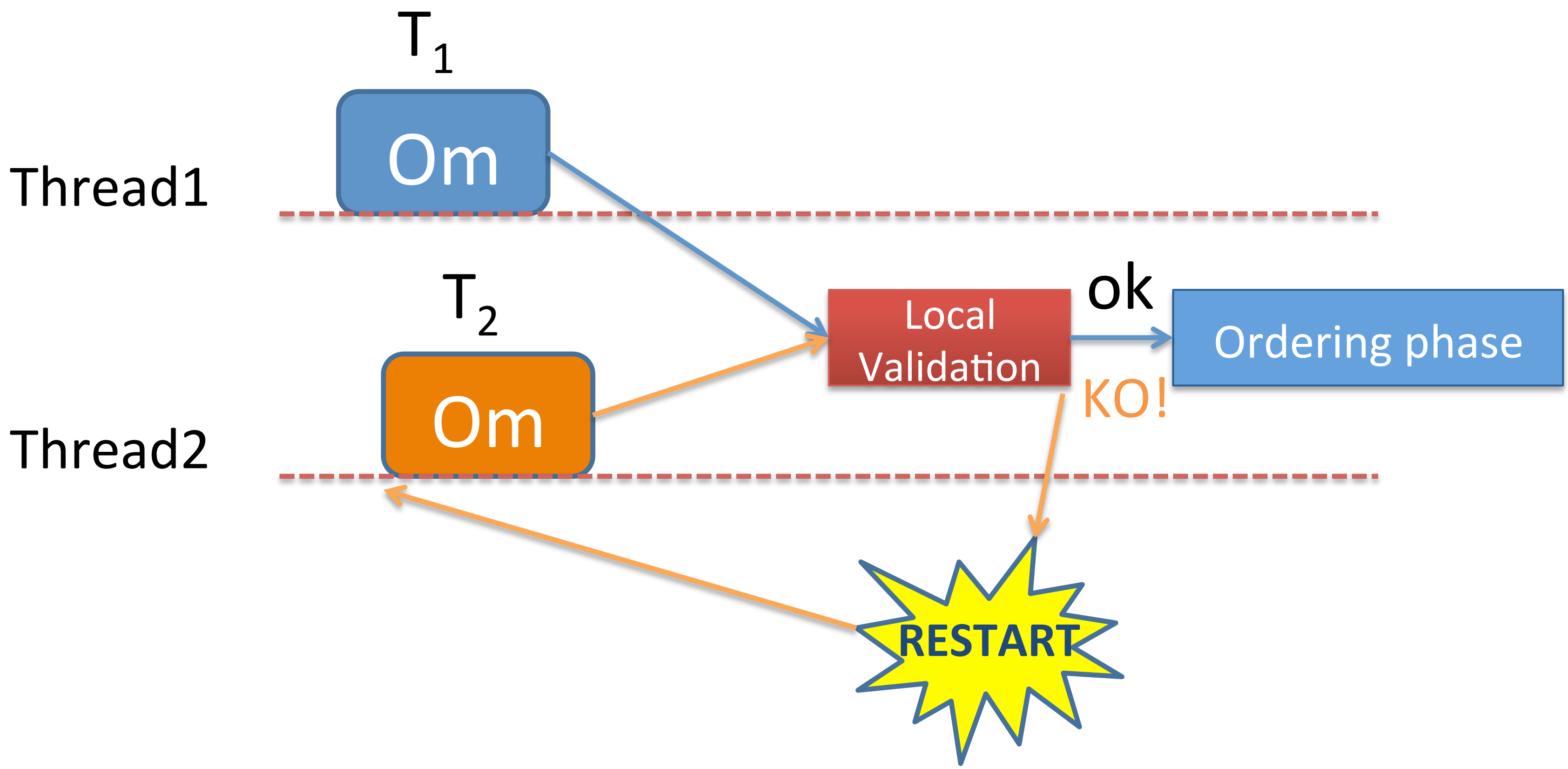


# Partial Solution

- Transaction validates against local transactions before being certified
  - Underutilization of the total order layer
  - Increased latency perceived by clients due to repeated local retries

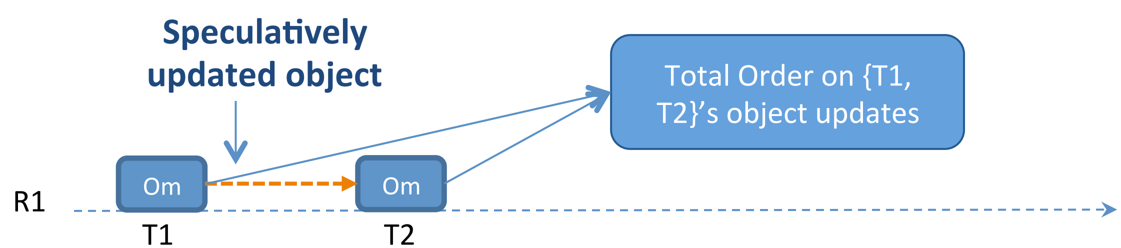


# Local Pre-Validation



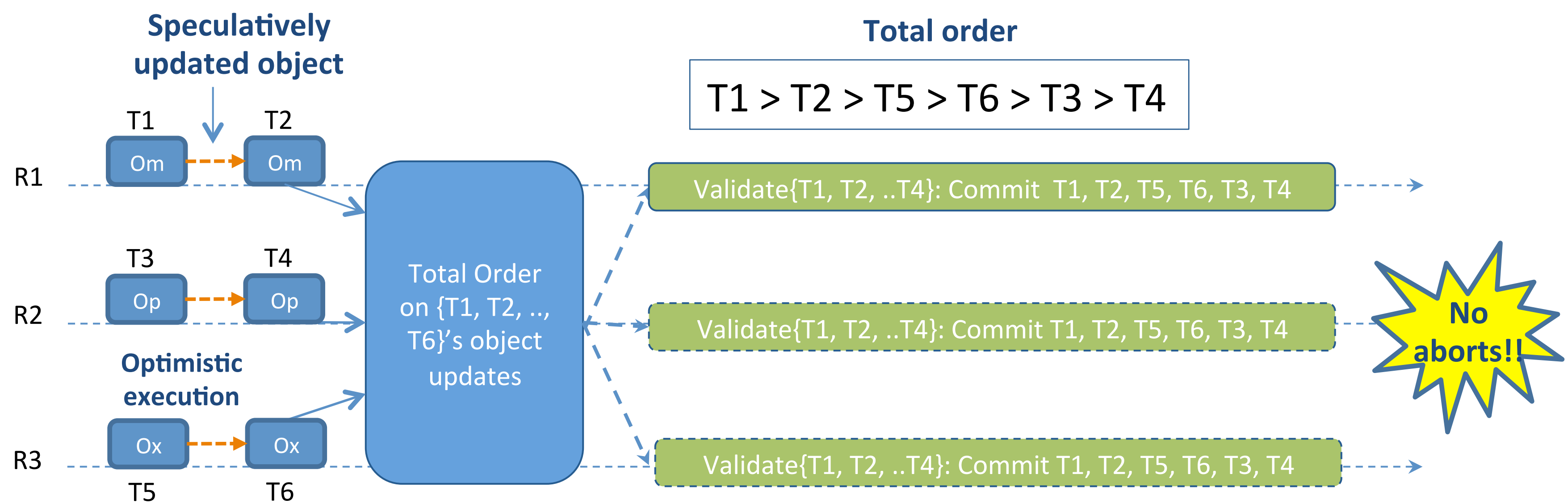
# Proposed Solution: Speculation

- Local Transaction Ordering
  - Introduction of an order for local transaction optimistic execution
  - Ordered transaction processing eliminates conflicts
- Speculative commit and read
  - Transactions commit speculatively and make their updates available to following transactions
  - Transactions read from speculative versions of objects modified by earlier transactions
  - Transactions help following transactions to commit without aborts



# Proposed Solution

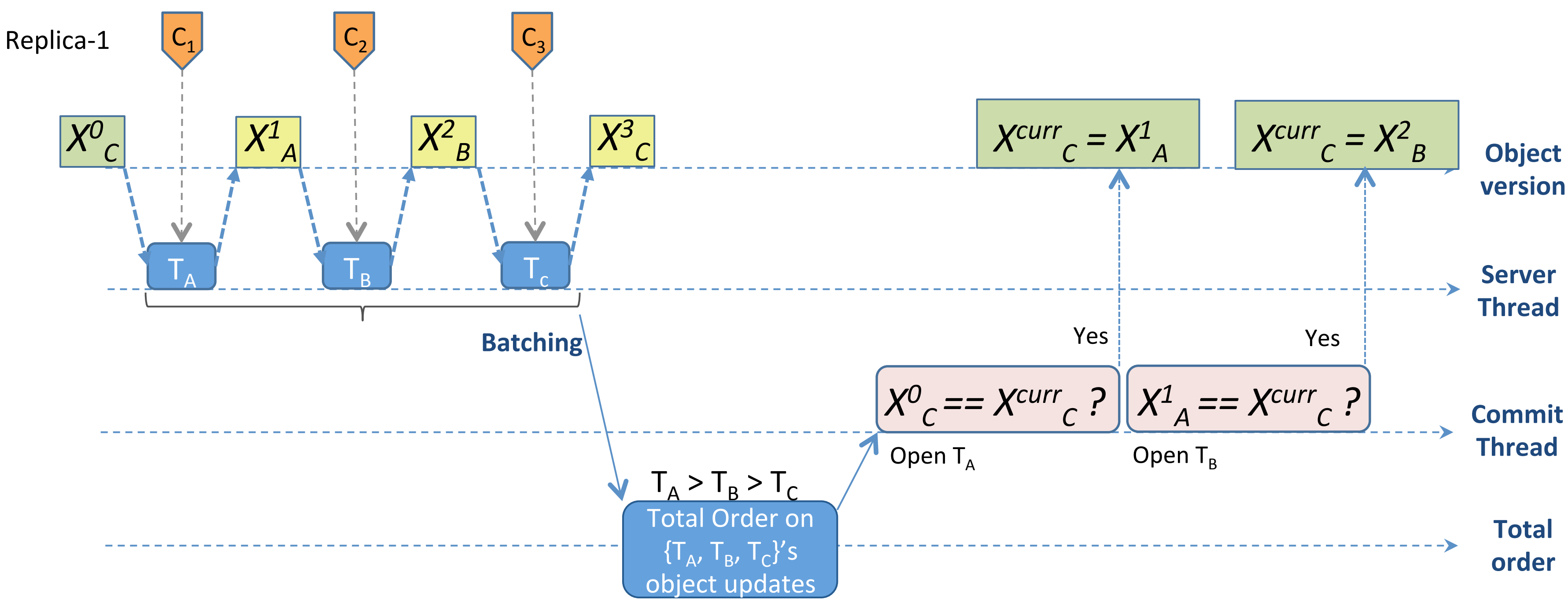
- Propagating the updates in the same order as optimistic execution order
  - Transactions from one node go to global certification phase in the same order as their execution order
  - Identical order of execution and certification reduces false conflicts



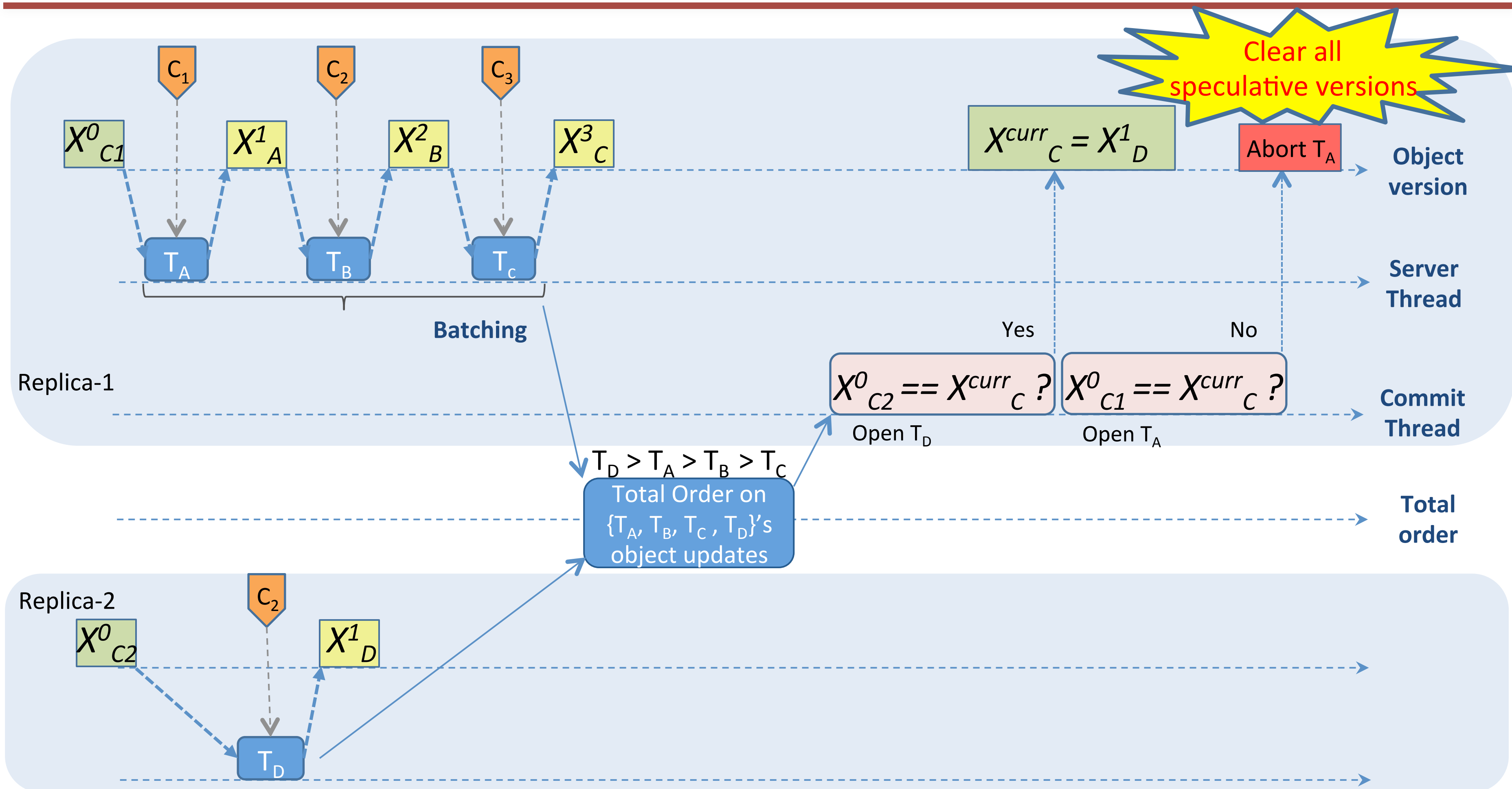
# How it works: No Abort

- Example execution on a single node
  - Counter benchmark: Each node has its own counter

$$T = \begin{bmatrix} R(x) \\ W(x) \end{bmatrix}$$



# How it works: Abort



# Evaluation

- **Prototype in Java**
- **Testbed – PRObE cluster (23 nodes)**
  - AMD Opteron 6272, 64-core, 2.1 GHz CPU
  - 128 GB RAM and 40 Gbps ethernet
- **Benchmarks**
  - TPC-C
  - Vacation (from the STAMP suite benchmark)
  - Bank
- **Competitors**
  - PaxosSTM: DUR-based approach without any speculation
  - X-DUR: our proposal

# Evaluation: TPC-C

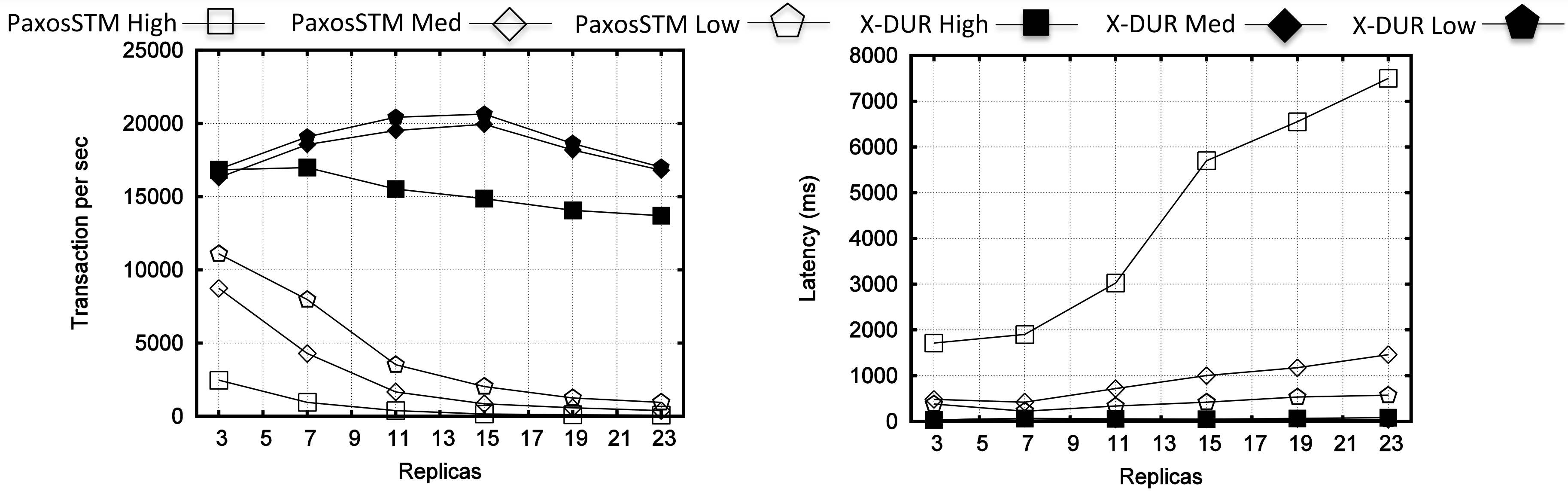



Fig. Performance plots for varying contention workload A.) Throughput, B.) Client perceived latency

- Contention settings
  - 23 warehouses (High-), 115 warehouses (Med-) and 230 warehouses (Low-conflict)
- Long transactions (OLTP) profile with 92% read-write requests
- Aborts of long transactions severely hampers PaxosSTM's performance

# Evaluation: Vacation

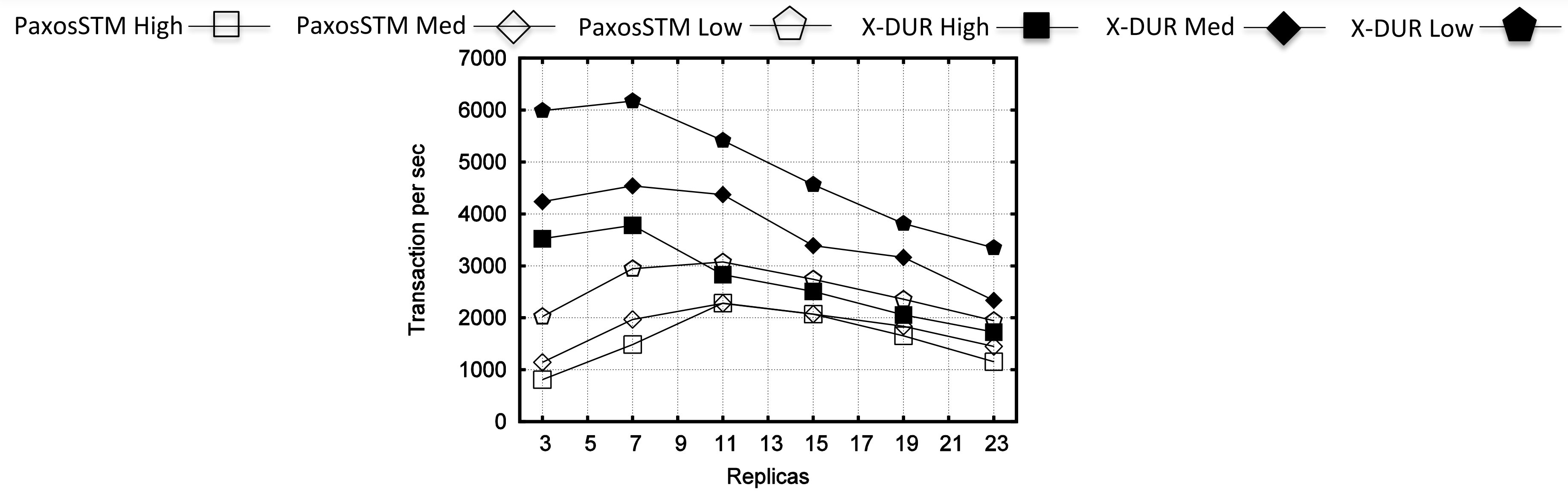



Fig. Throughput with varying contention

- Contention settings:
  - 250 relations (High-), 500 (Mid-) and 1000 (Low-conflicts)
- X-DUR out-performs PaxosSTM for all contention settings
- As system size increase, network overhead impact both X-DUR and PaxosSTM similarly

# Evaluation: Bank

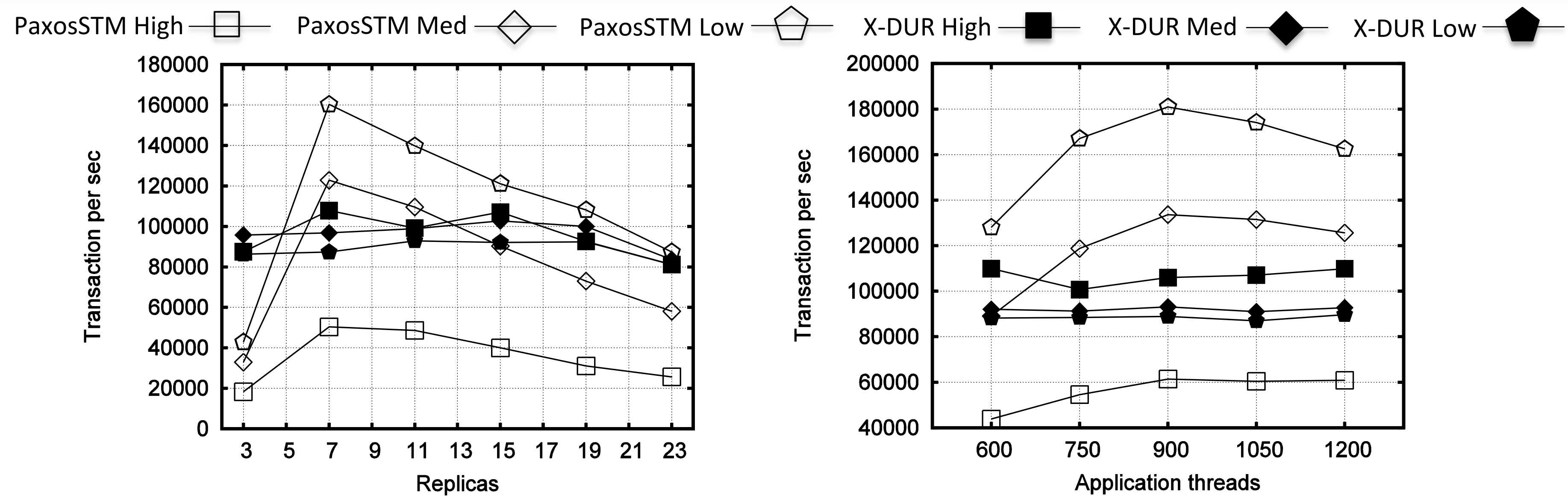



Fig. Throughput plots A.) Varying number of nodes, B.) Varying application threads on 7 nodes

- Contention settings:
  - 500 objects (High-conflict), 2000 (Medium-conflict) and 5000 objects (Low-conflict)
- PaxosSTM suffers from aborts in high % of conflicts even for partitioned accesses
- PaxosSTM benefits from massive parallelism in low and medium contention workload

# Thank You

Speculation pays off!

Questions?