```
[<c219ec5f>] security_sk_free+0xf/0x2
[<c2451efb>] __sk_free+0x9b/0x120
[<c25ae7c1>] ? _raw_spin_unlock_irqre
[<c2451ffd>] sk_free+0x1d/0x30
[<c24f1024>] unix_release_sock+0x174/
```

Optimizing Distributed Transactions: Speculative Client Execution, Certified Serializability, and High Performance Run-Time

Thesis Defense—Master of Science

<u>Utkarsh Pandey</u>

Advisor: Binoy Ravindran

Systems Software Research Group

Virginia Tech

ece Bradley Department of Electrical & Computer Engineering

VirginiaTech Invent the Future

Overview

- Introduction
- Motivation
- Contributions
  - PXDUR
  - TSAsR
  - Verified Jpaxos
- PXDUR
- Experimental Results
- Conclusions

Transactional Systems

- Back end - online services.
- Usually backed by one or more Database Management Systems (DBMS).
- Support multithreaded operations.
- Require concurrency control.
- Employ transactions to execute user requests.
- Transactions – Unit of atomic operation.

Replication in services

- Replication – Increased availability, fault tolerance.
- Service replicated on a set of server replicas.
- Distributed algorithms – Co-ordination among distributed servers.
- State Machine Replication (SMR) –
  - All replicated servers run command in a common sequence.
  - All replicas follow the same sequence of states.

Distributed Transactional Systems

- Distributed system:
  - Service running on multiple servers (replicas).
  - Data replication (full or partial).
  - Transactional systems - support multithreading.
- Deferred Update Replication (DUR):
  - A method to deploy a replicated service.
  - Transactions run locally, followed by ordering and certification.
- Fully partitioned data access:
  - A method to scale the performance of DUR based systems.
  - No remote conflicts.
  - The environment studied here.
- Bottlenecks in fully-partitioned DUR systems:
  - Local conflicts among application threads.
  - Rate of certification post total order establishment.

SMR algorithms

- Distributed algorithms:
  - Backbone of replicated services.
  - Based on State Machine Replication (SMR).
- Optimization of SMR algorithm:
  - Potential of huge benefits.
  - Involve high verification cost.
- Existing methods to ease verification:
  - Functional languages lending easily to verification – EventML, Verdi.
  - Frameworks for automated verification – PSYNC.
- Modeled algorithms - low performance.

Centralized Database Management Systems

- Centralized DBMS:
  - are standalone systems.
  - Employ transactions for DBMS access.
  - Support multithreading - exploit multicore hardware platforms.
- Concurrency control:
  - Prevent inconsistent behavior.
  - Serializability - Gold standard isolation level.
- Eager-locking protocols:
  - Used to enforce serializability.
  - Too conservative for many applications.
  - Scale poorly with increase in concurrency.

Motivation for Transactional Systems Research

Problems

- Alleviate local contention in distributed servers(DUR based) through speculation and parallelism.
- Low scalability of centralized DBMS with increased parallelism.
- Lack of high performance SMR algorithms which lend themselves easily to formal verification.

Research Goals

- **Broad**: Improve system performance while ensuring ease of deployment.
- **Thesis**: Three contributions – PXDUR, TSAsR and Verified JPaxos.

Research Contributions

- **PXDUR:**
  - DUR based systems suffer from local contention and limited by committer's performance.
  - Speculation can reduce local contention.
  - Parallel speculation improves performance.
  - Commit optimization provides added benefit.

- **TSAsR :**
  - *Serializability*: Transactions operate in isolation.
  - Too conservative requirement for many applications.
  - Ensure serializability using additional meta-data while keeping the system's default isolation relaxed.

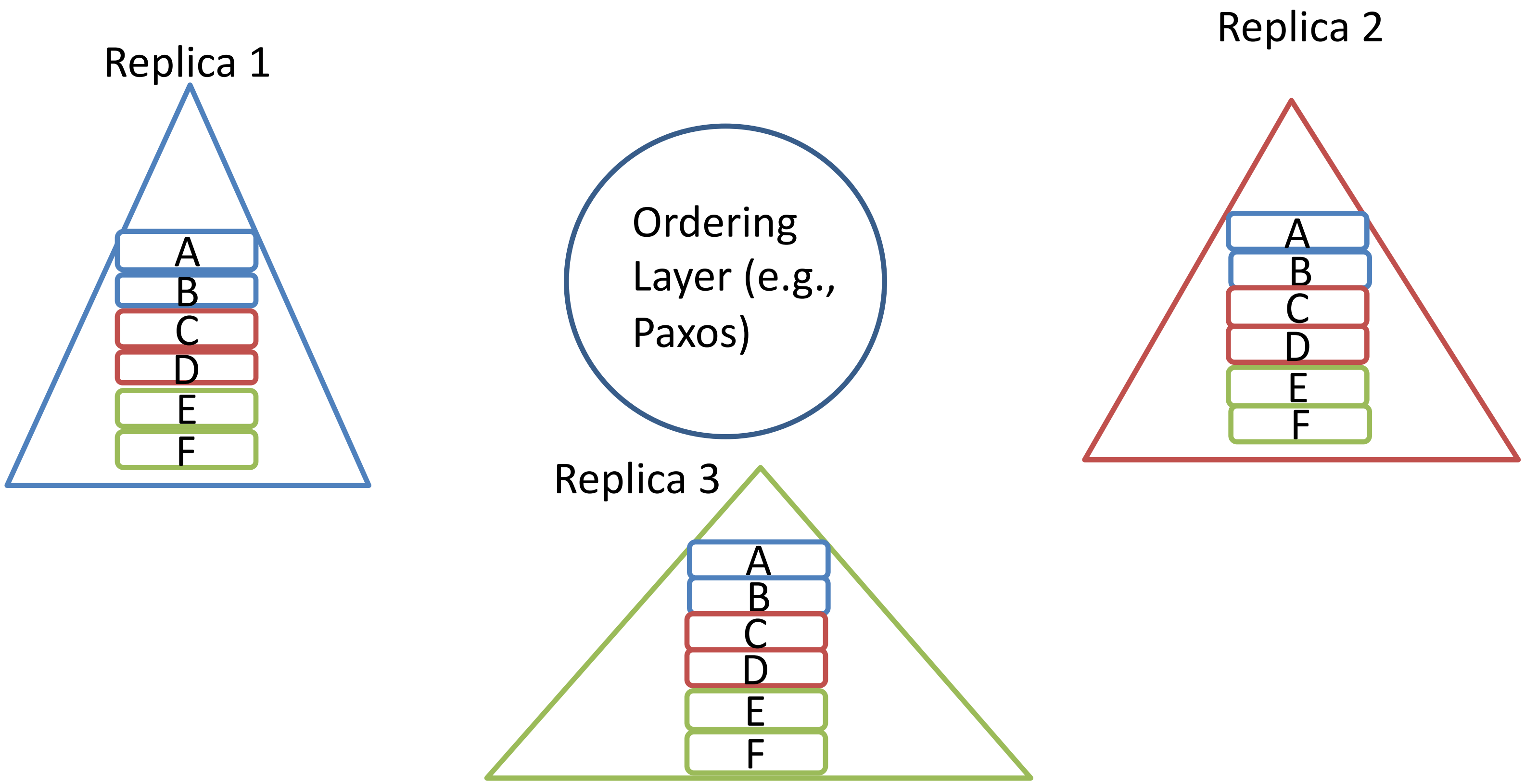
Research Contributions

- **Verified JPaxos**
  - SMR based algorithms not easy to verify.
  - Algorithms produced by existing verification frameworks perform poorly.
  - JPaxos based run-time for easy to verify Multipaxos algorithm, generated from HOL specification.

PXDUR : Related Work

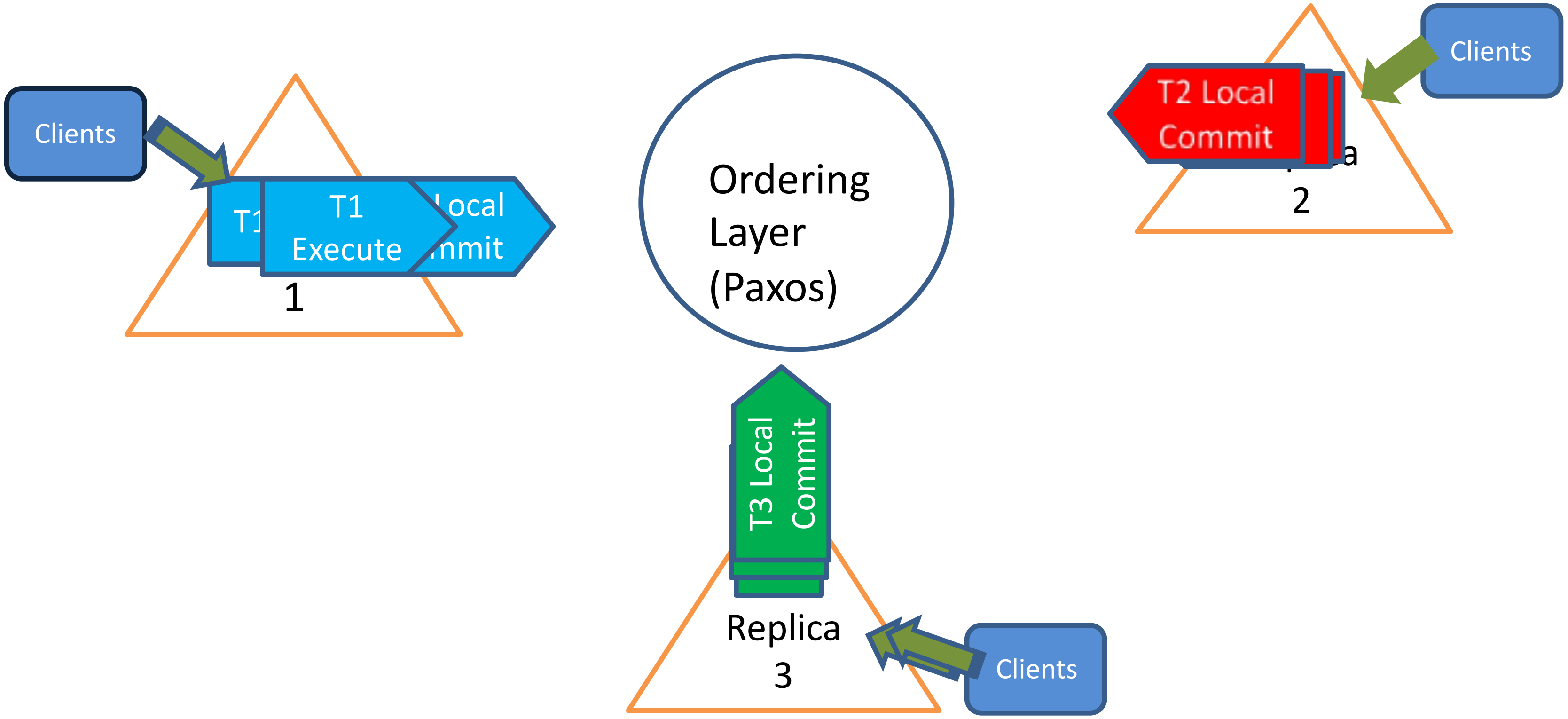
- DUR :
  - Introduced as an alternative to immediate update synchronization.
- SDUR:
  - Introduces the idea of using fully partitioned data access.
  - Significant improvement in performance.
- Conflict aware load balancing:
  - Reduce local contention by putting grouping conflicting requests on replicas.
- XDUR :
  - Alleviate local contention by speculative forwarding.

Fully Partitioned Data Access



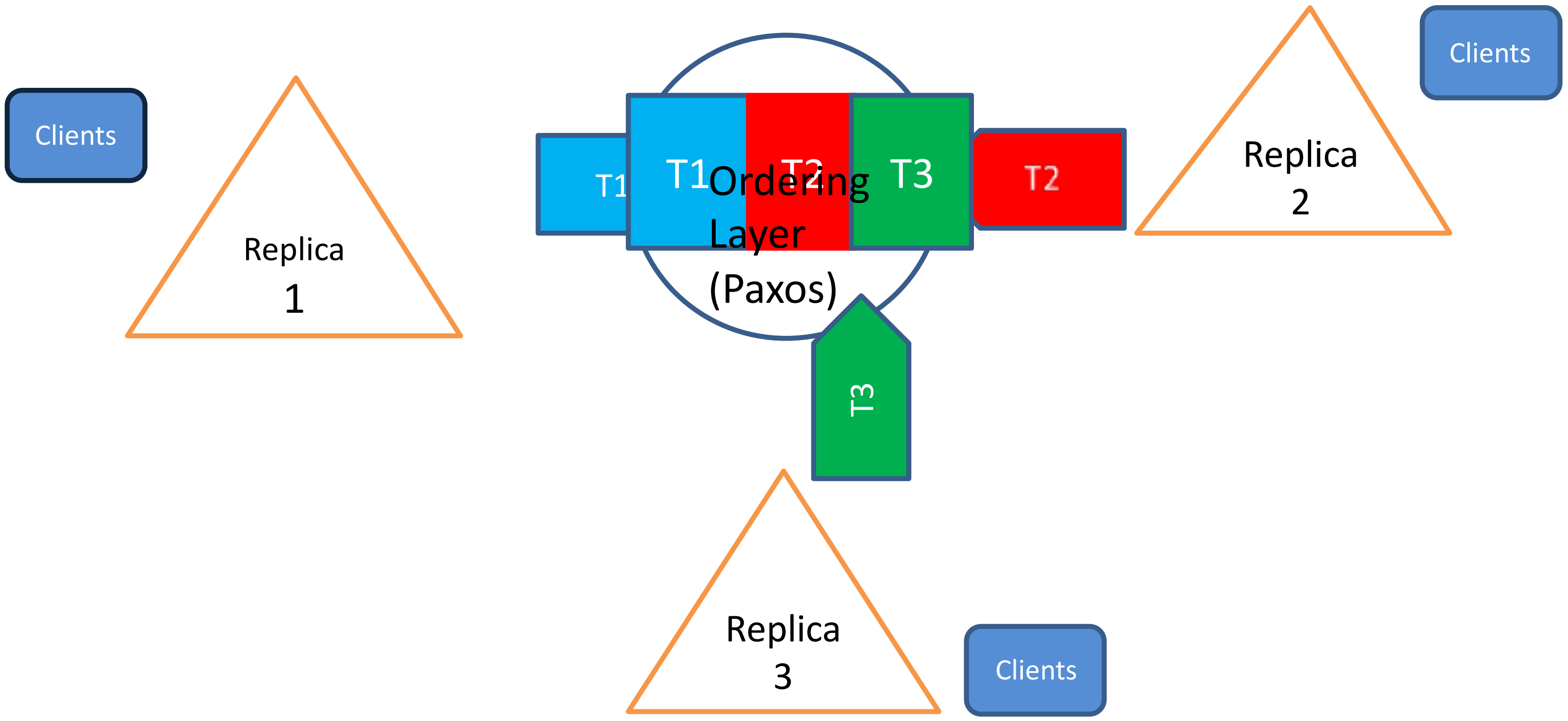
Deferred Update Replication

Local Execution Phase



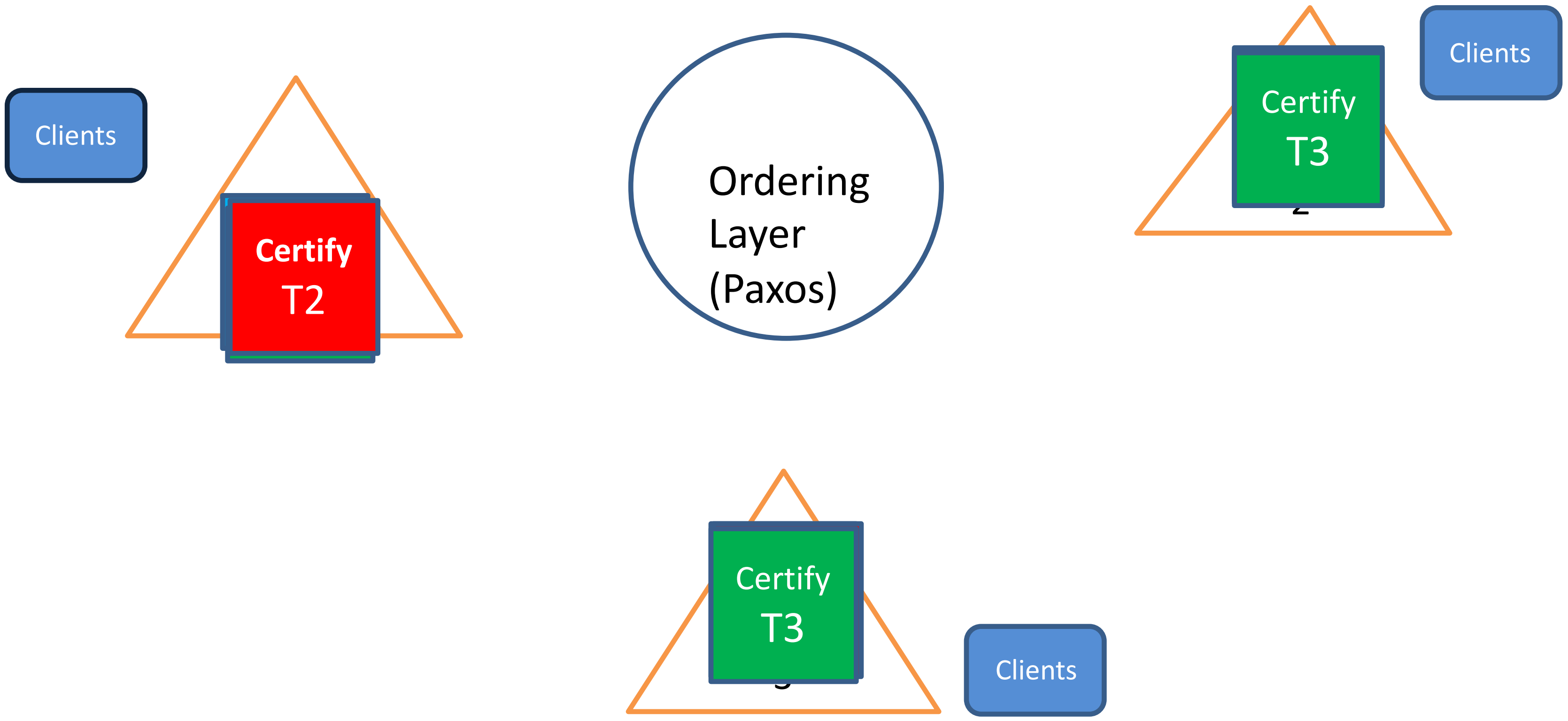
Deferred Update Replication

Global Ordering Phase



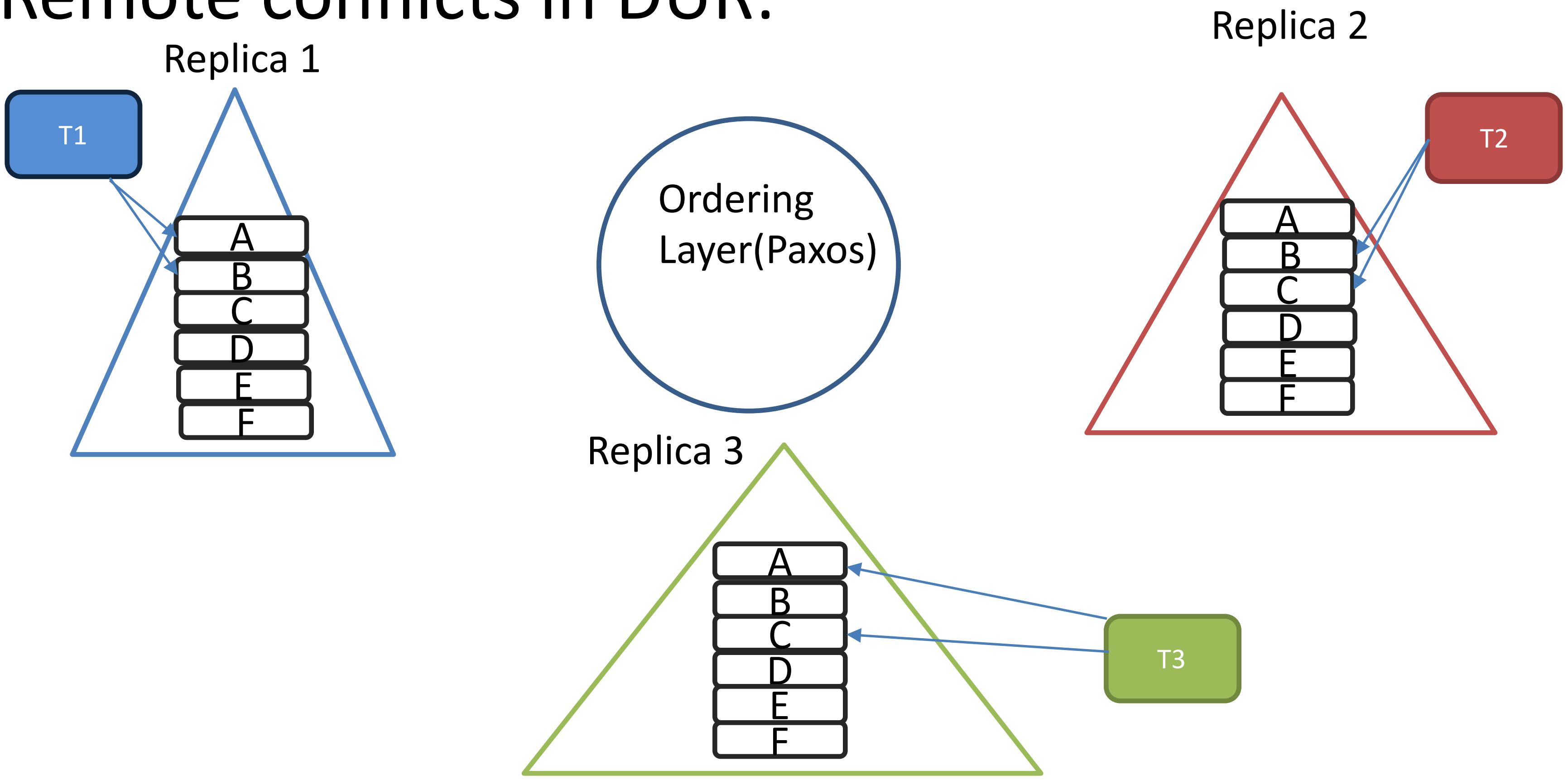
Deferred Update Replication

Certification Phase:



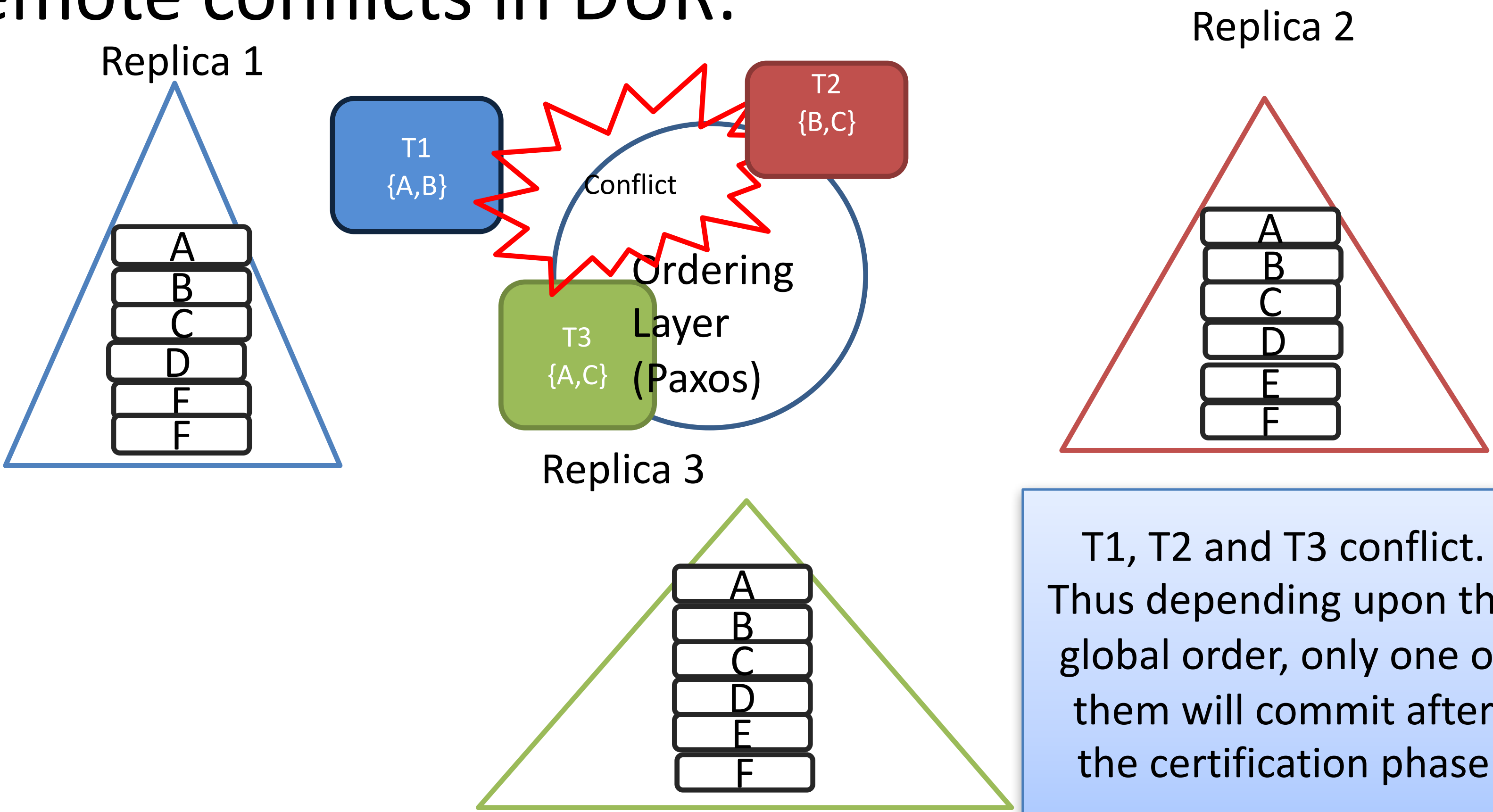
Deferred Update Replication

Remote conflicts in DUR:



Deferred Update Replication

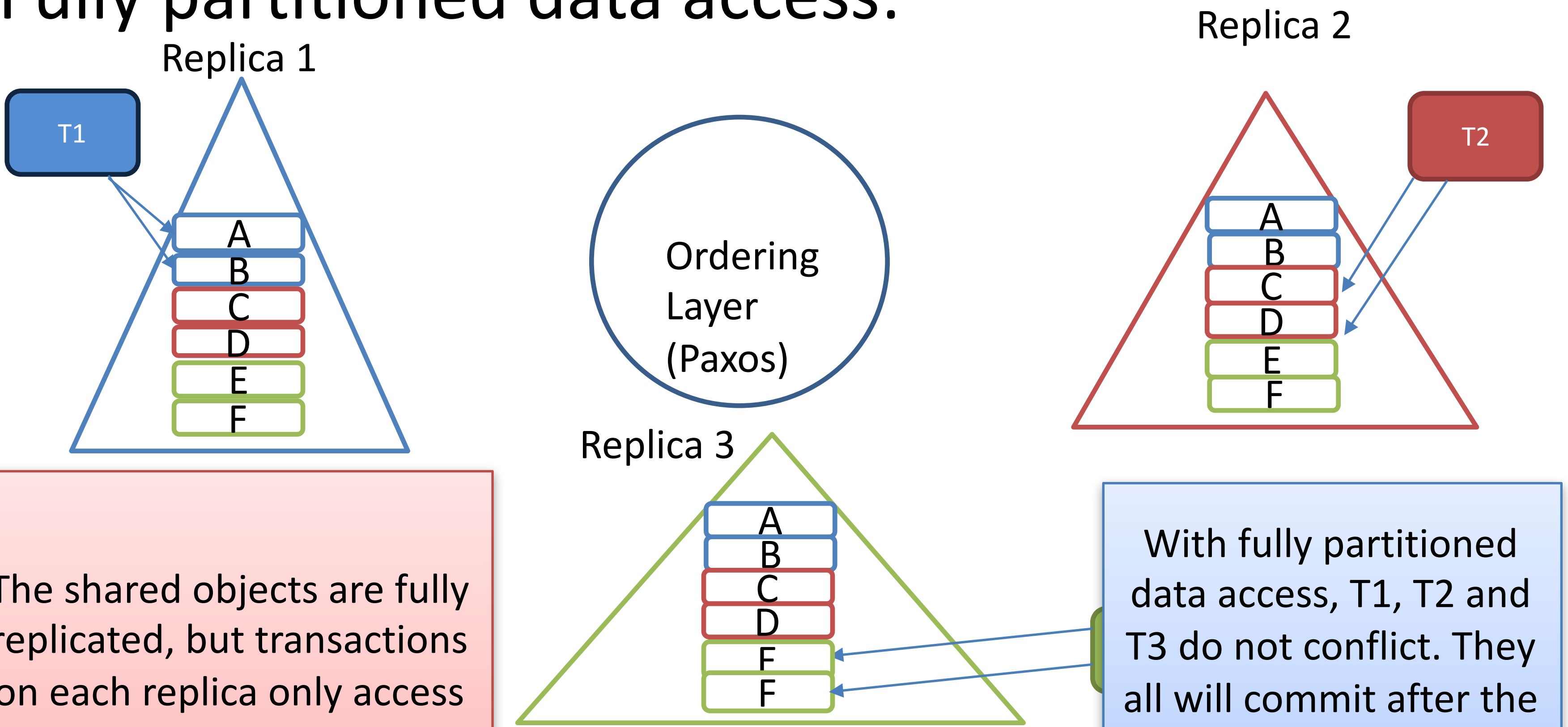
Remote conflicts in DUR:



T1, T2 and T3 conflict. Thus depending upon the global order, only one of them will commit after the certification phase

Deferred Update Replication

Fully partitioned data access:



The shared objects are fully replicated, but transactions on each replica only access a mutually exclusive subset of objects.

With fully partitioned data access, T1, T2 and T3 do not conflict. They all will commit after the total order is established.

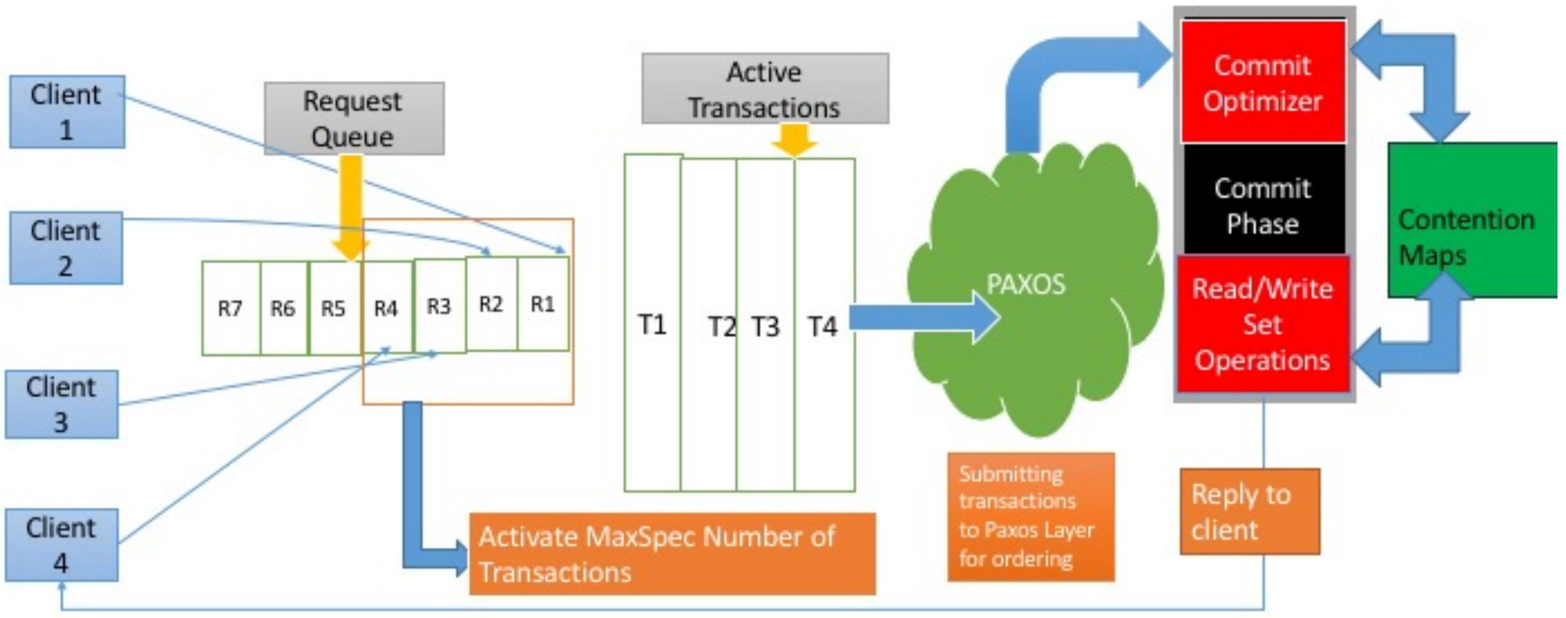
Bottlenecks in fully partitioned DUR systems

- Fully partitioned access - Prevents remote conflicts.
- Other factors which limit performance:
  - Local contention among application threads.
  - Rate of post total-order certification.

PXDUR

- PXDUR or Parallel XDUR.
- Addresses local contention through speculation.
- Allows speculation to happen in parallel:
  - Improvement in performance.
  - Flexibility in deployment.
- Optimizes the commit phase:
  - Skip the read-set validation phase, when safe.

PXDUR Overview

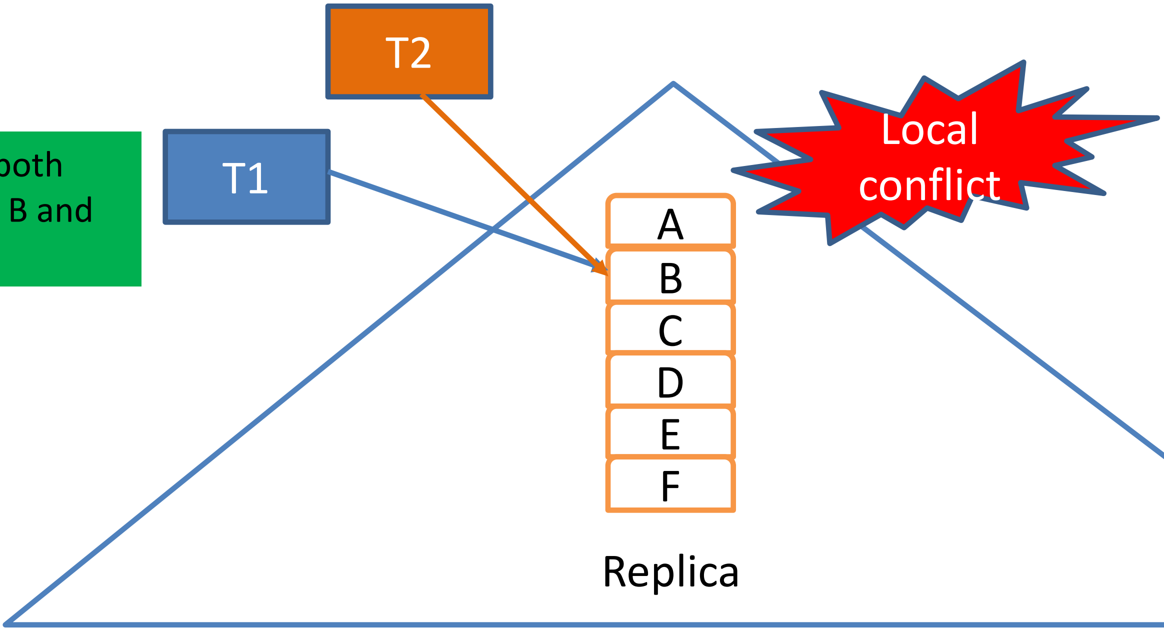


Reducing local contention

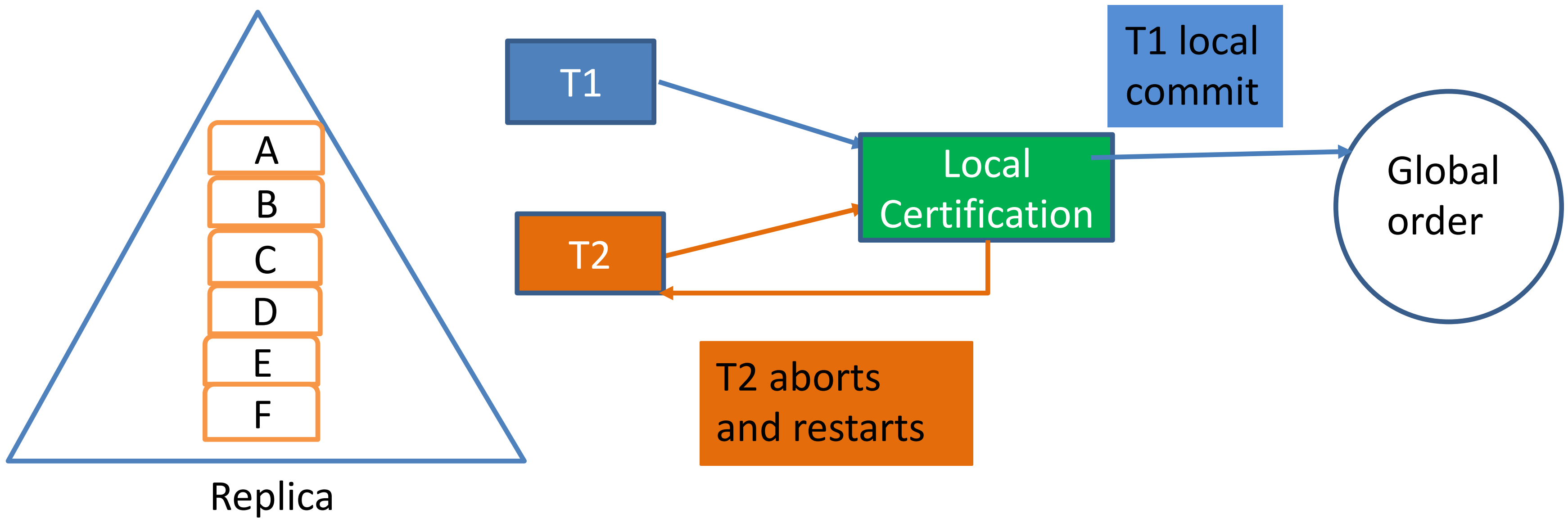
- Speculative forwarding : Inherited from XDUR.
- Active transactions - Read from the snapshot generated by completed local transactions, awaiting global order.
- Ordering protocol respects the local order:
  - Transactions are submitted in batches respecting the local order.

Local contention in DUR

T1 and T2 both read object B and modify it.



Local contention in DUR



Speculation in PXDUR

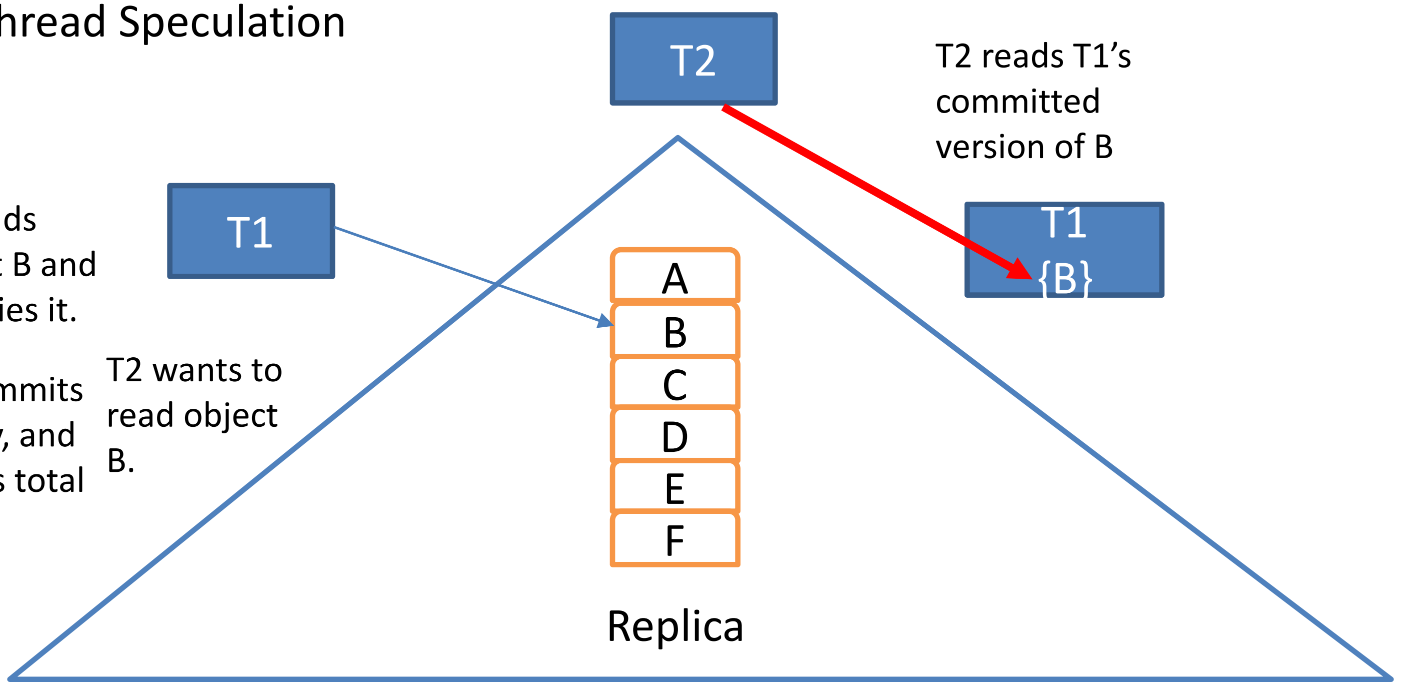
Single thread Speculation

T1 reads object B and modifies it.

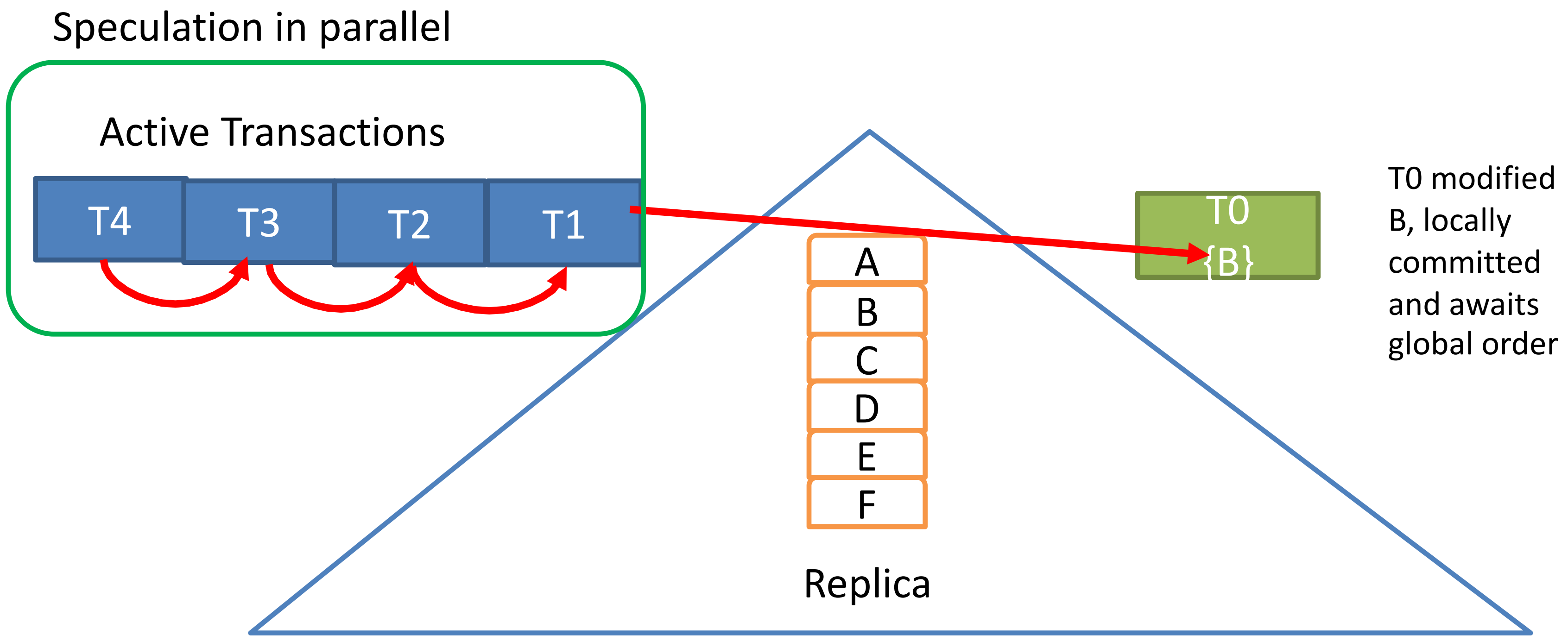
T1 commits locally, and awaits total order.

T2 wants to read object B.

T2 reads T1's committed version of B



Speculation in PXDUR



T0 modified B, locally committed and awaits global order

Speculation in parallel

- Concurrent transactions speculate in parallel.
- Concurrency control employed to prevent inconsistent behavior:
  - Extra meta-data added to objects.
- Transactions:
  - Start in parallel.
  - Commit in order.
- Allows for scaling of single thread XDUR.

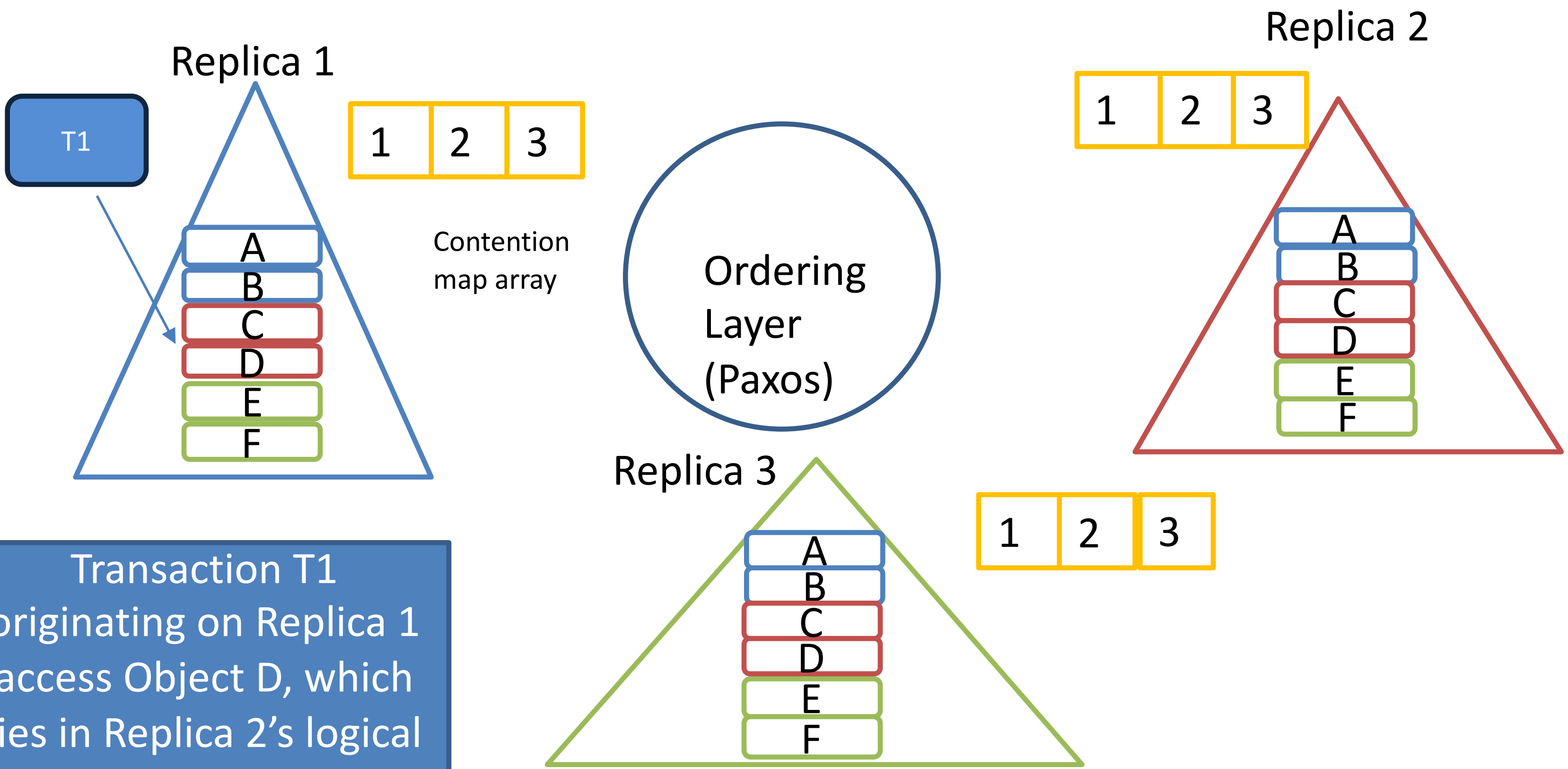
Commit Optimization

- Fully partitioned data access:
  - Transactions never abort during final certification.
- We use this observation to optimize the commit phase.
- If a transaction does not expect conflict:
  - Skip the read-set validation phase of the final commit.

Commit Optimization

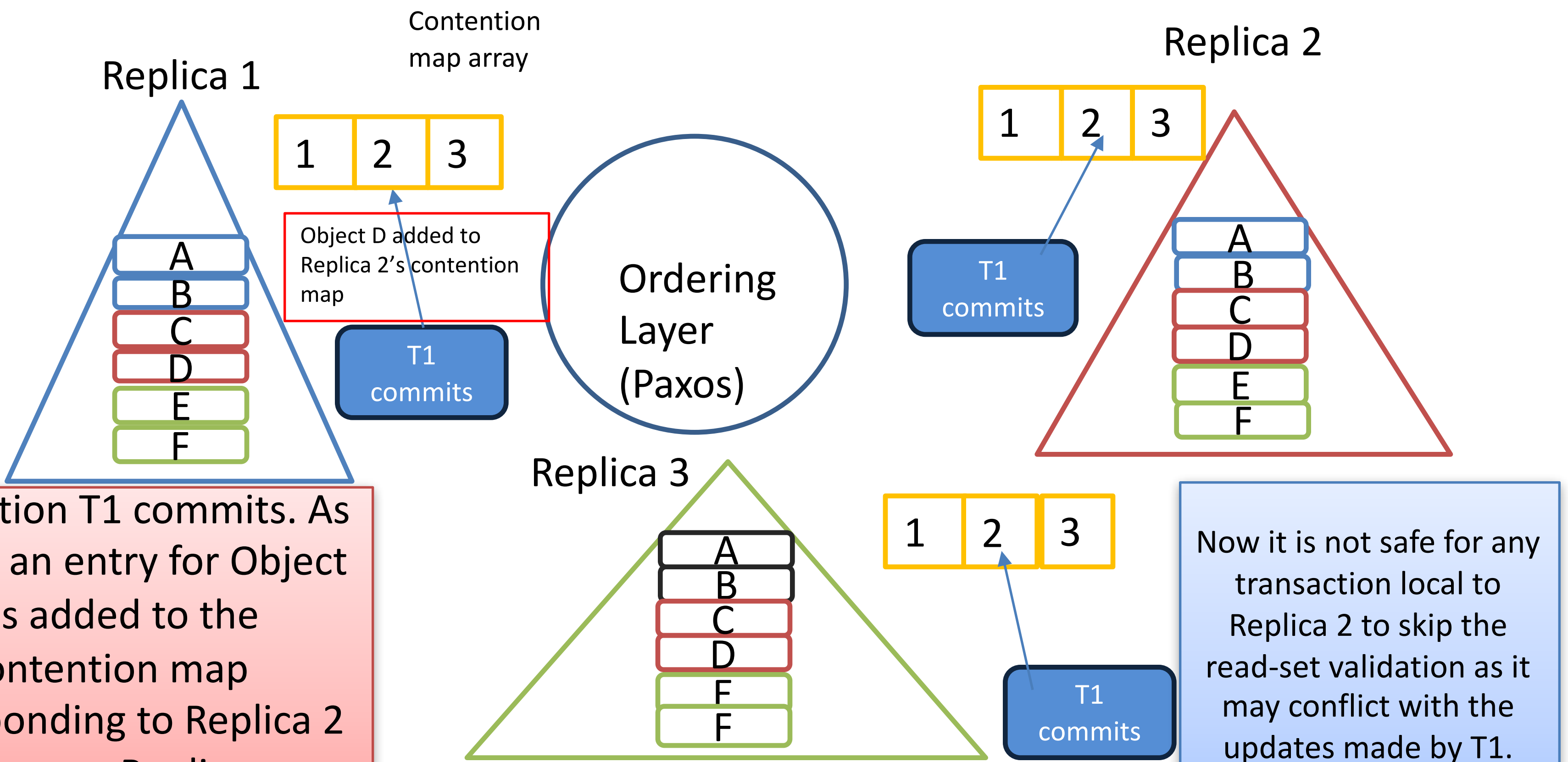
- Array of contention maps present on each replica:
  - Each array entry corresponds to one replica.
  - Contention maps contain the object IDs which are suspected to cause conflicts.
- A transaction cannot skip the read-set validation if:
  - It performed cross-partitioned access.
  - The contention map corresponding to its replica of origination is not empty.
- Contention maps fill when:
  - A transaction doing cross-partition access commits.
  - A local transaction aborts.

Commit Optimization



Transaction T1 originating on Replica 1 access Object D, which lies in Replica 2's logical partition.

Commit Optimization



Transaction T1 commits. As a result an entry for Object D is added to the contention map corresponding to Replica 2 on every Replica.

Now it is not safe for any transaction local to Replica 2 to skip the read-set validation as it may conflict with the updates made by T1.

Evaluation Results

PRObE Cluster.
AMD Opteron, 64 core, 2.1 GHz CPU.
128 GB of memory, 40Gb Ethernet.

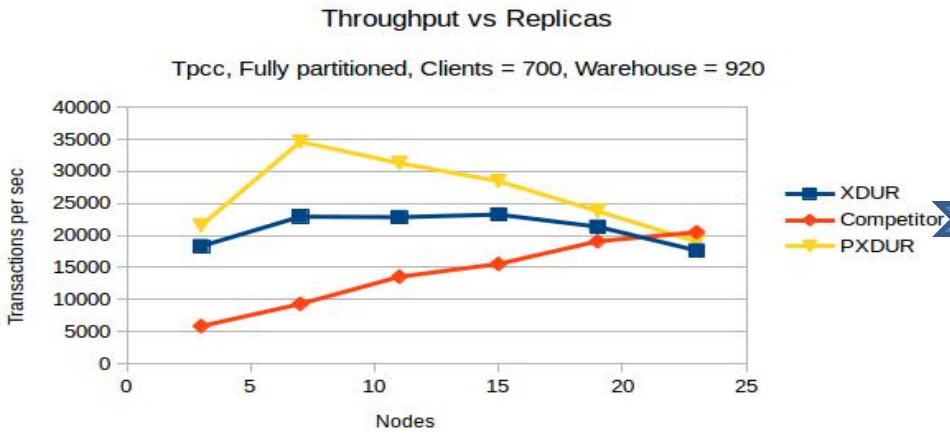
Benchmarks: Bank, TPC-C.

Configuration:

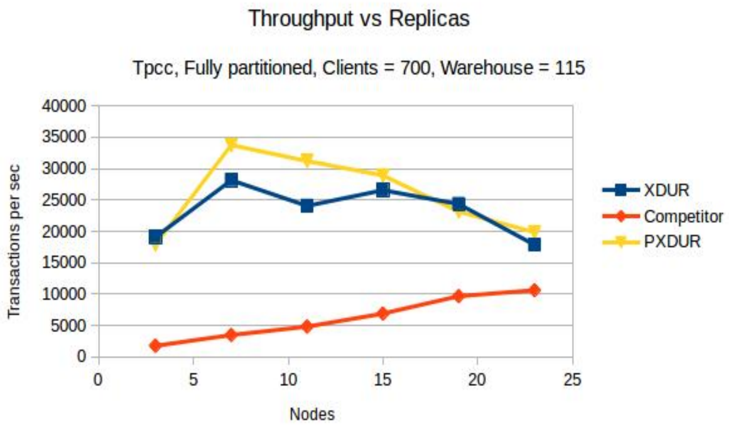
- Each benchmark studied under fully partitioned data access.
- Experiments conducted for low, medium and high local contention.
- Up to 23 replicas were used.

.

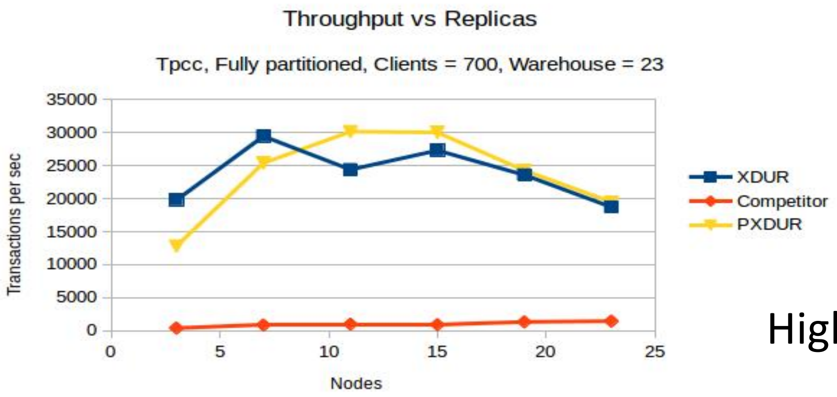
TPC-C



Low contention

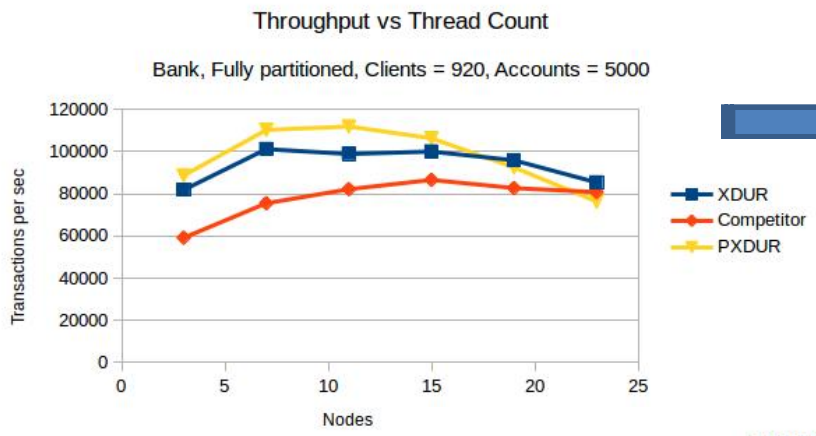


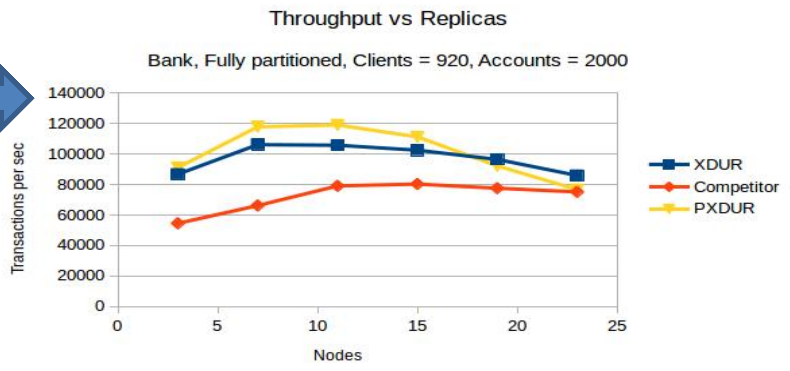
Medium contention



High contention

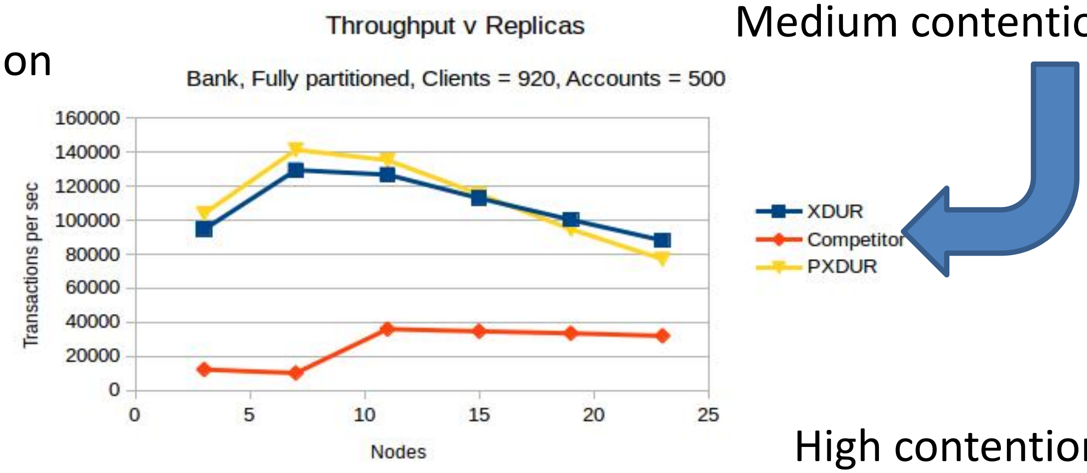
Bank





Low contention

Medium contention



High contention

Evaluation results

- PXDUR reaps the benefit of both parallelism and speculation for low and medium contention scenarios.
- For high contention scenarios, it still gives good performance due to speculation.

Conclusion

- Contributions:
  - PXDUR
  - TSAsR
  - Verified Jpaxos
- Significant performance improvement.
- Ease of usability.
- Improved performance scalability with the increase in cores.