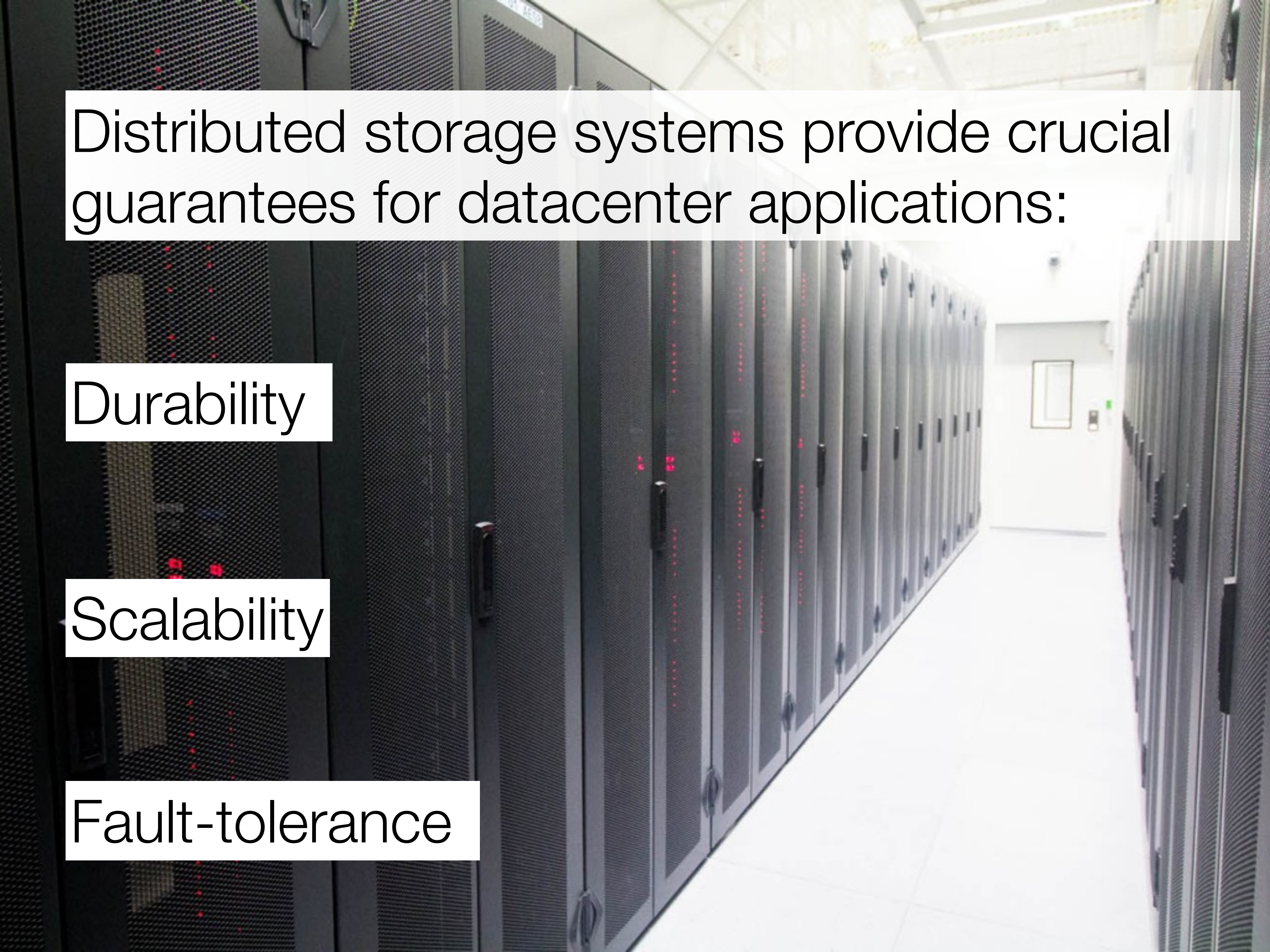
Building Consistent Transactions with Inconsistent Replication

Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres,
Arvind Krishnamurthy, Dan R. K. Ports
University of Washington



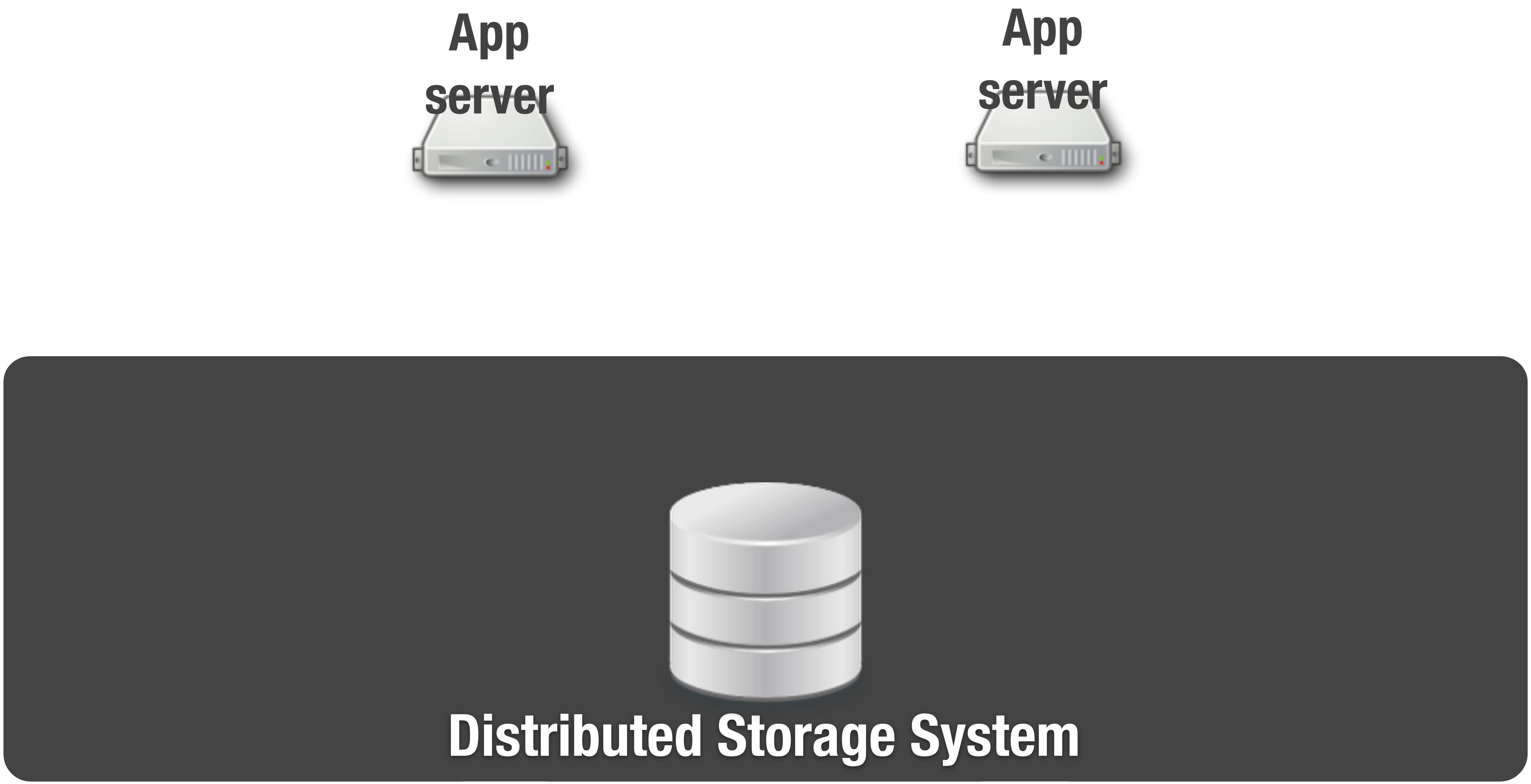
Distributed storage systems provide crucial guarantees for datacenter applications:

Durability

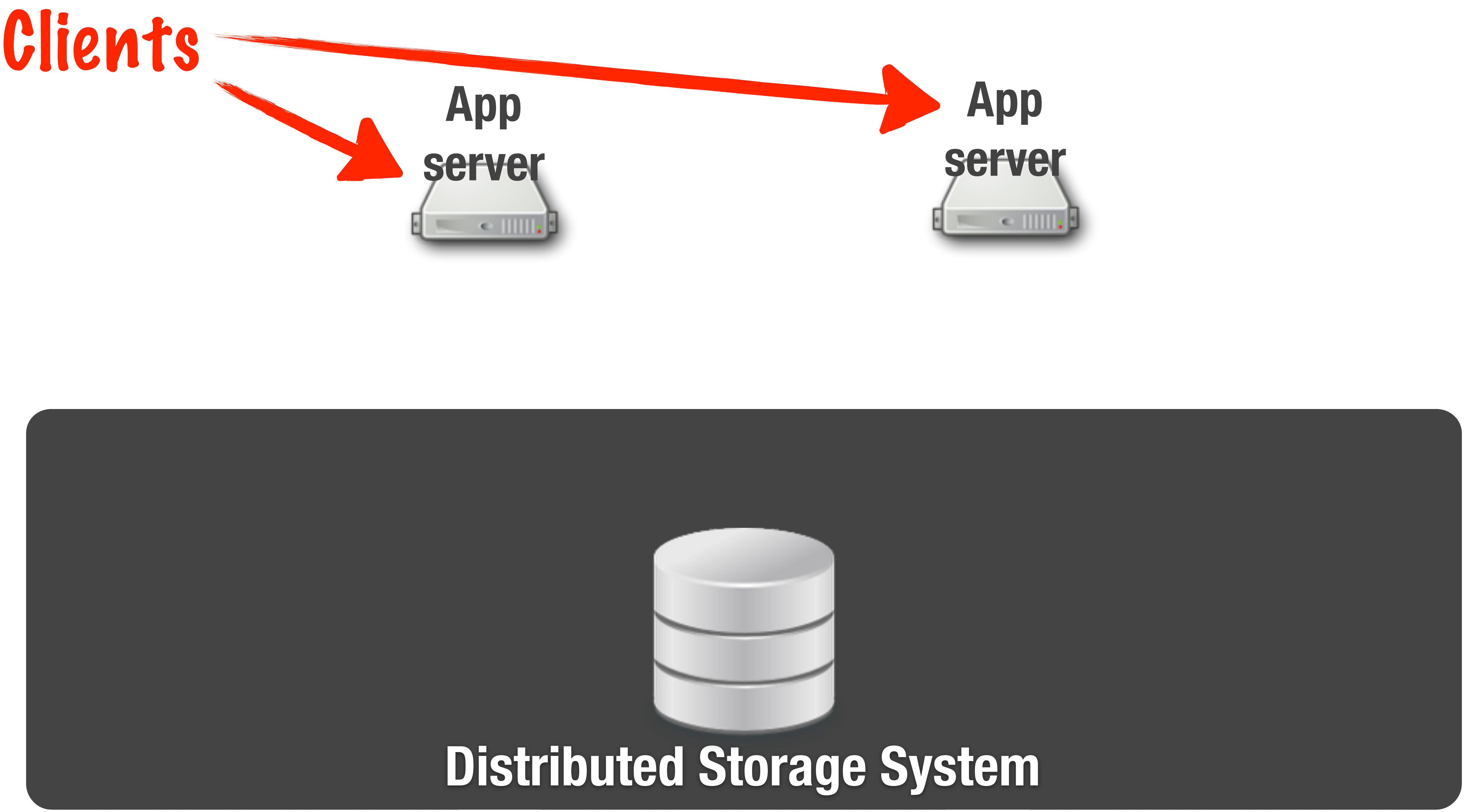
Scalability

Fault-tolerance

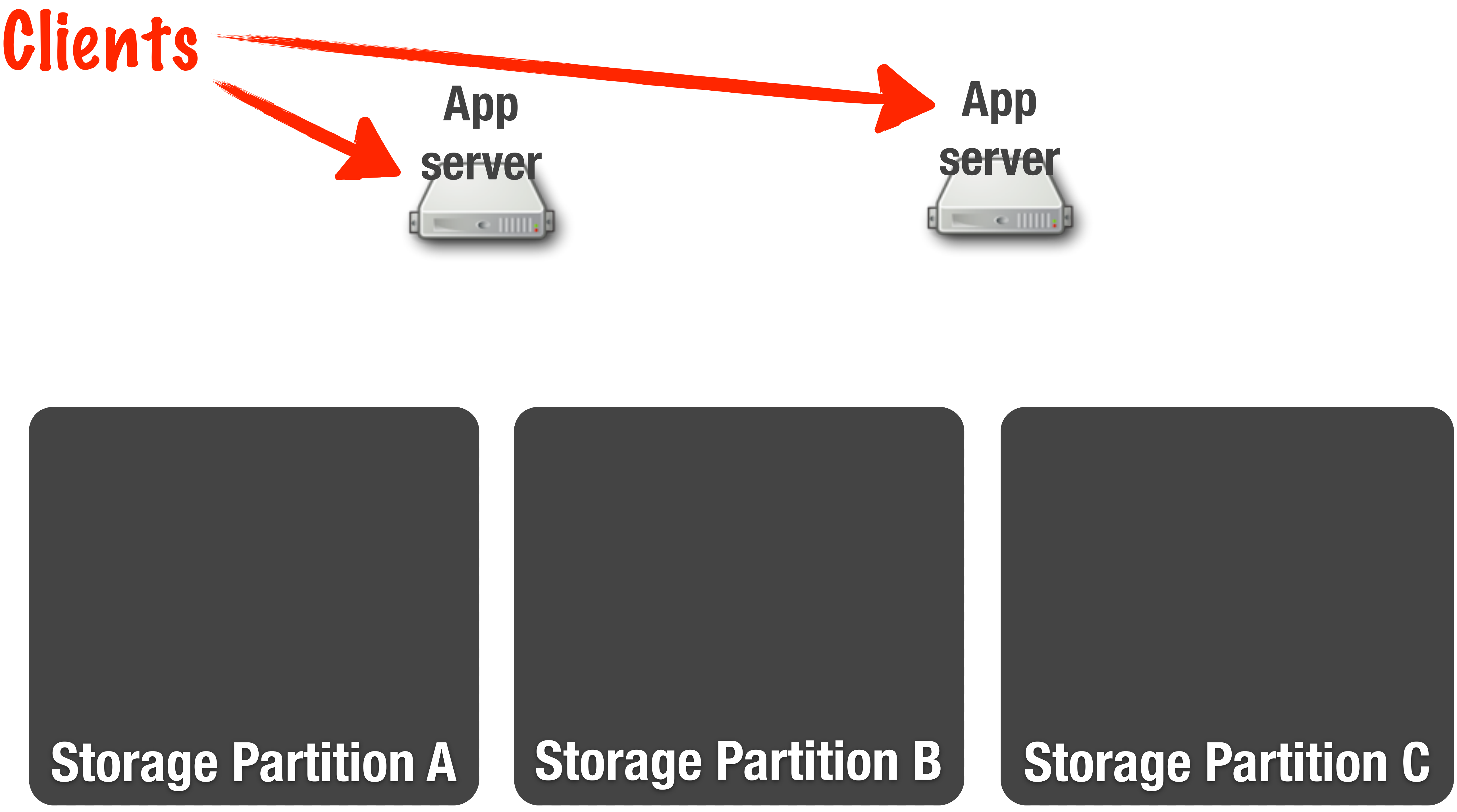
Distributed Storage Architecture



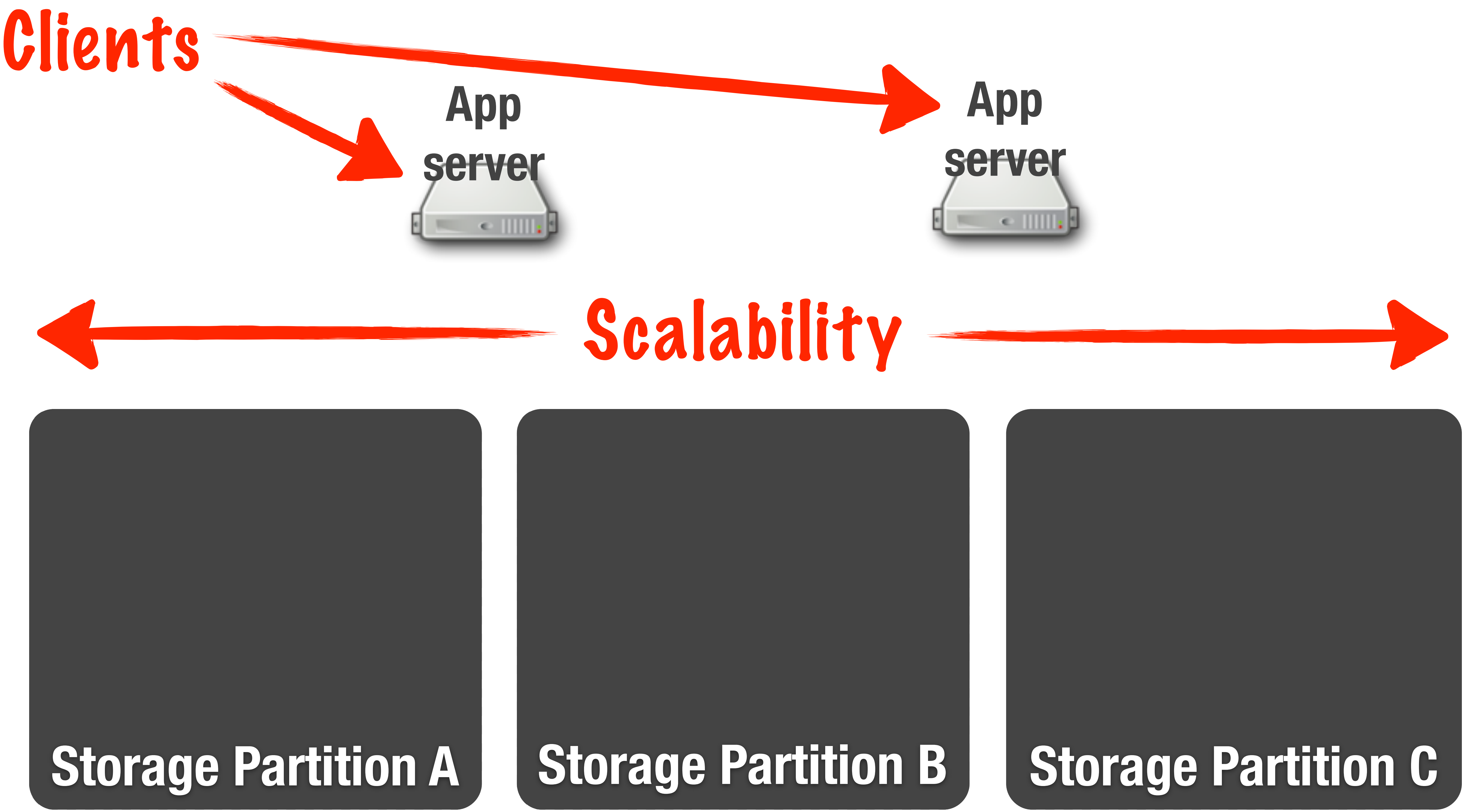
Distributed Storage Architecture



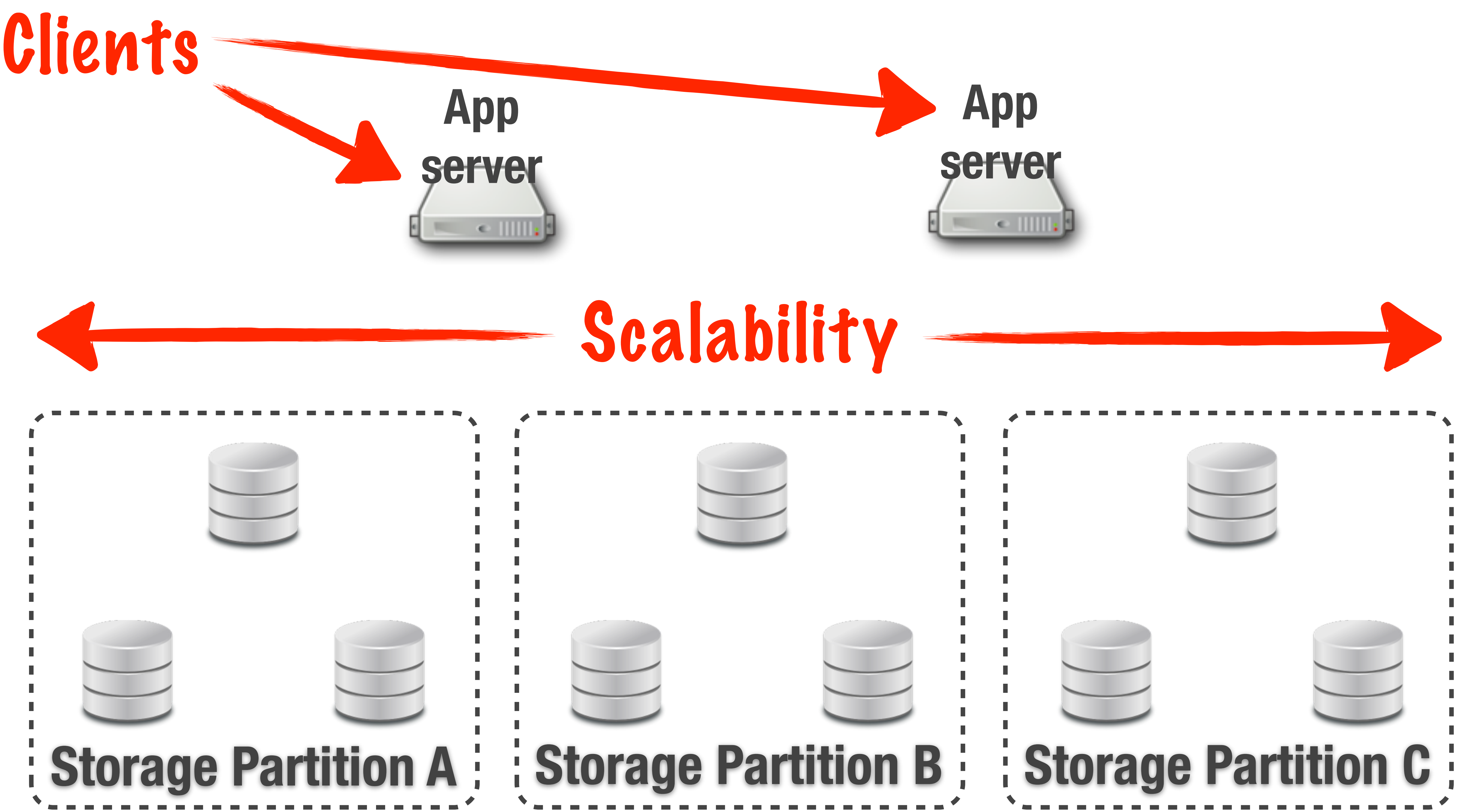
Distributed Storage Architecture



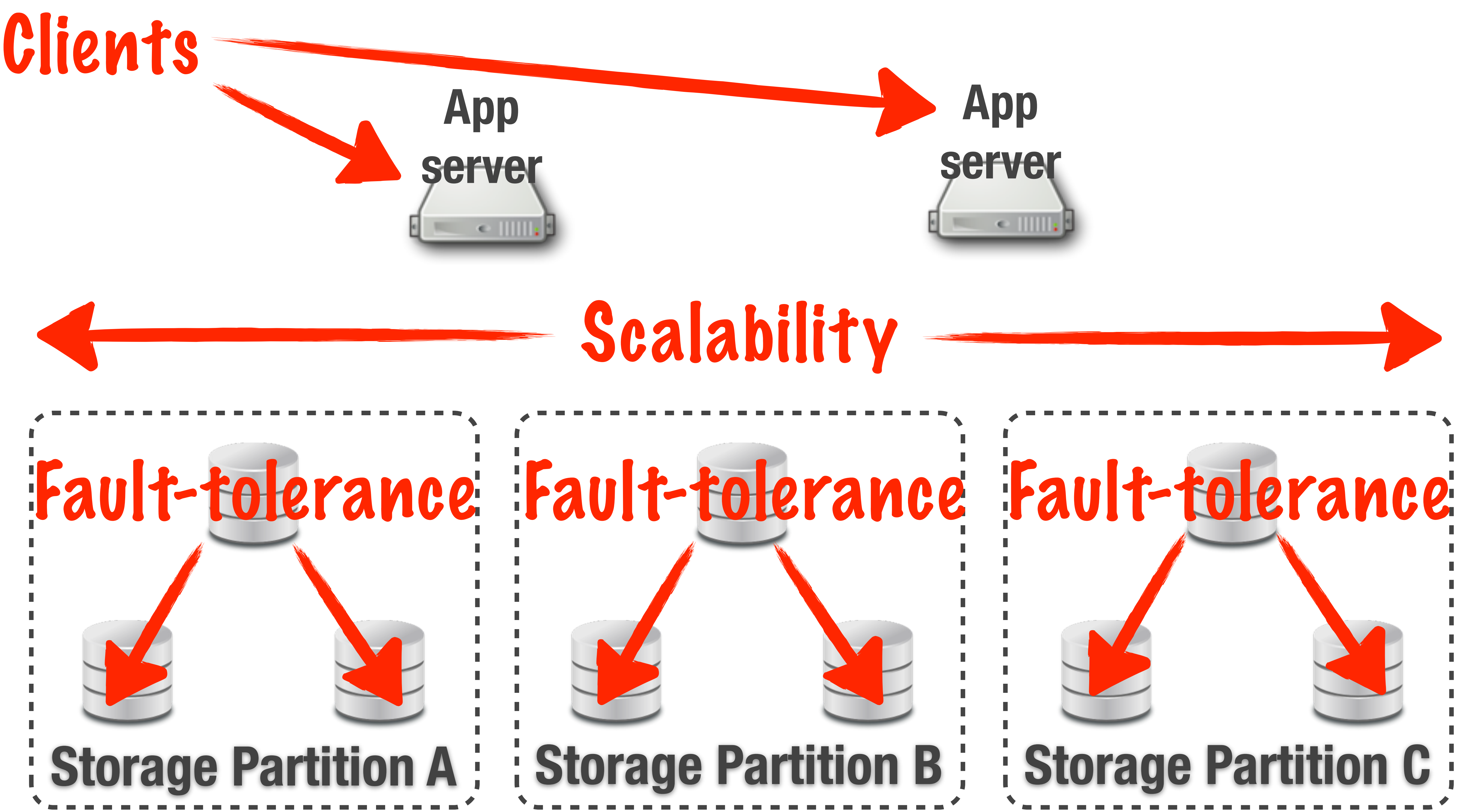
Distributed Storage Architecture



Distributed Storage Architecture



Distributed Storage Architecture



Consistency guarantees are important in a distributed system.

Guides programmer reasoning about:

- application state (i.e., what is a valid state, what invariants can I assume)
- concurrency (i.e., what happens when two writes happen at the same time)
- failures (i.e., what happens when the system fails in the middle of an operation)

Some systems have weaker consistency guarantees.

- Eventual consistency - eventual ordering of operations and applications resolve conflicts
- No atomicity or concurrency control - applications use versioning and explicit locking
- Examples: Dynamo, Cassandra, Voldemort

Some systems have strong consistency guarantees.

- ACID distributed transactions - help applications manage concurrency
- Strong consistency/linearizable isolation - strict serial ordering of transactions
- Examples: Spanner, MegaStore

Distributed transactions are expensive in a replicated system.

- Distributed transactions with strong consistency require replication with strong consistency.
- Replication with strong consistency imposes a high overhead.

Distributed transactions are expensive in a replicated system.

- Distributed transactions with strong consistency require replication with strong consistency.
- Replication with strong consistency imposes a high overhead.

Lots of cross-replica coordination = higher latency + lower throughput

Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.

Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.

Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.

Programmers face a choice.

- Strong consistency guarantees are easier to use but have limited performance.
- Weak consistency guarantees are harder to use but have better performance.

Our Goal

Make transactional storage <u>cheaper</u> to use while maintaining <u>strong guarantees</u>.

Our Goal

Improve latency and throughput for r/w transactions

Make transactional storage cheaper to use while maintaining strong guarantees.

Our Goal

Improve latency and throughput for r/w transactions

Make transactional storage <u>cheaper</u> to use while maintaining <u>strong guarantees</u>.

Strong Consistency
General Transaction Model

Our Approach

Provide distributed transactions with strong consistency using a replication protocol with no consistency.

Our Approach

Provide distributed transactions with strong consistency using a replication protocol with no consistency.

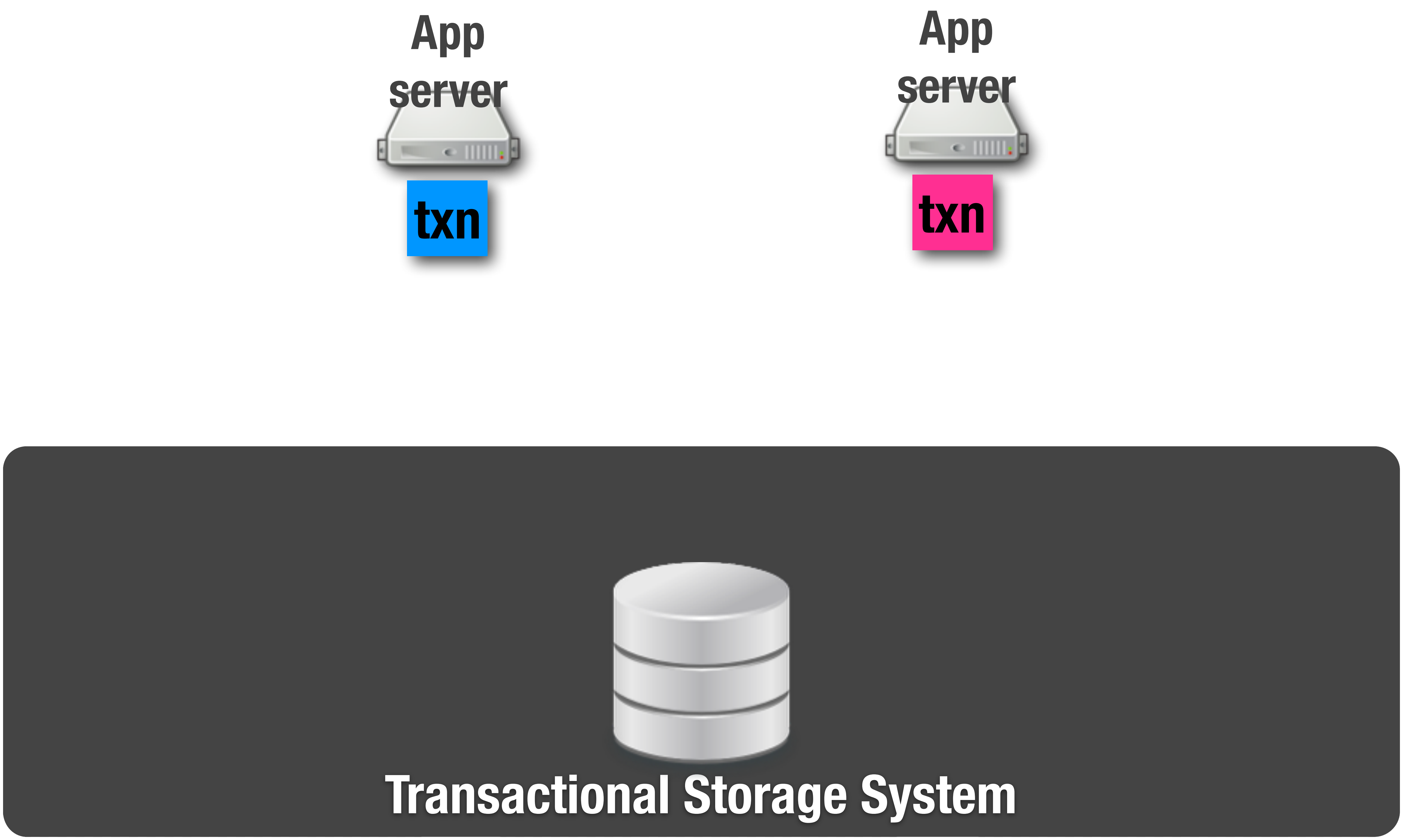
Our Approach

Provide distributed transactions with strong consistency using a replication protocol with no consistency.

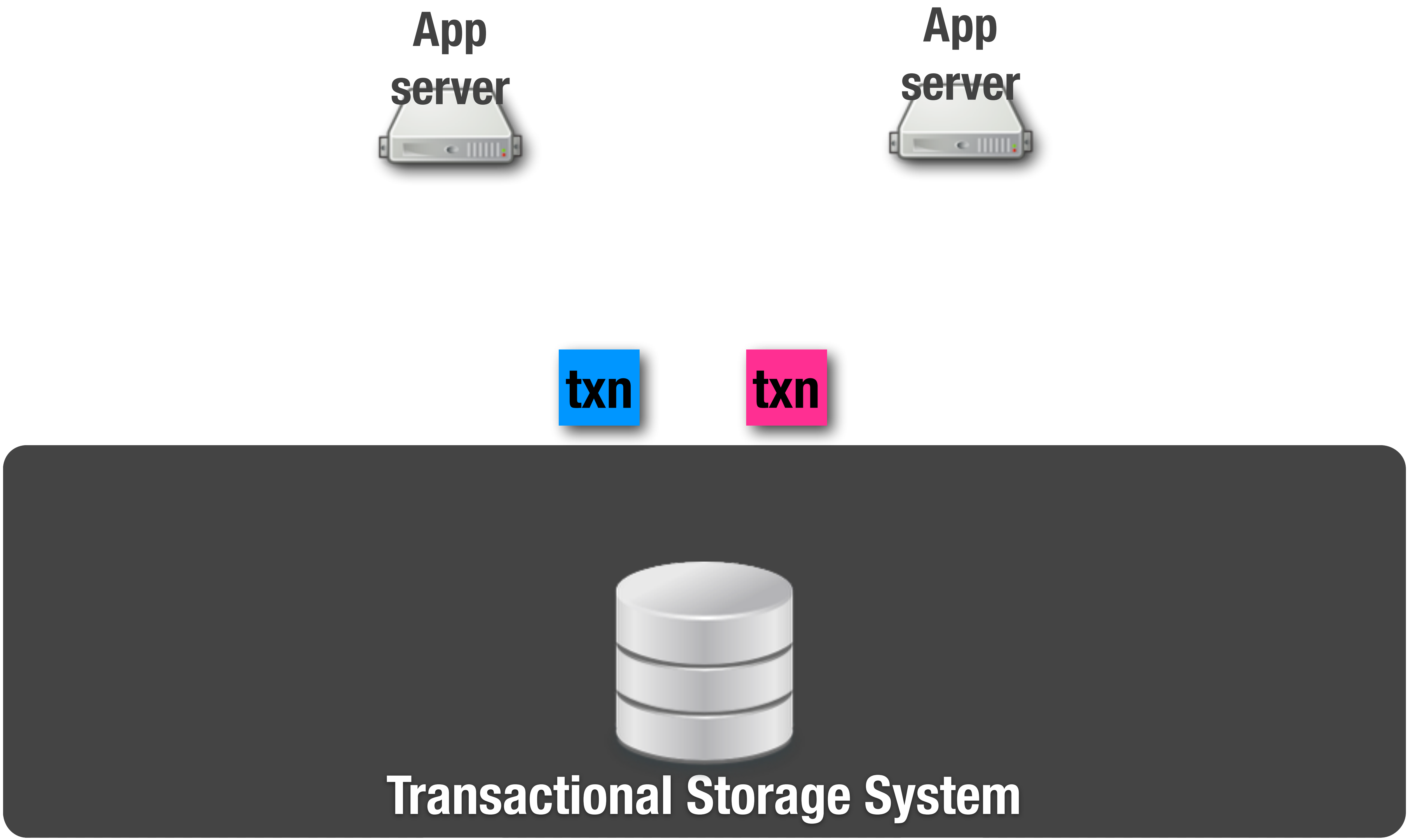
Rest of this talk

1. The cost of strong consistency
2. TAPIR - the Transactional Application Protocol for Inconsistent Replication
3. Evaluation
4. Summary

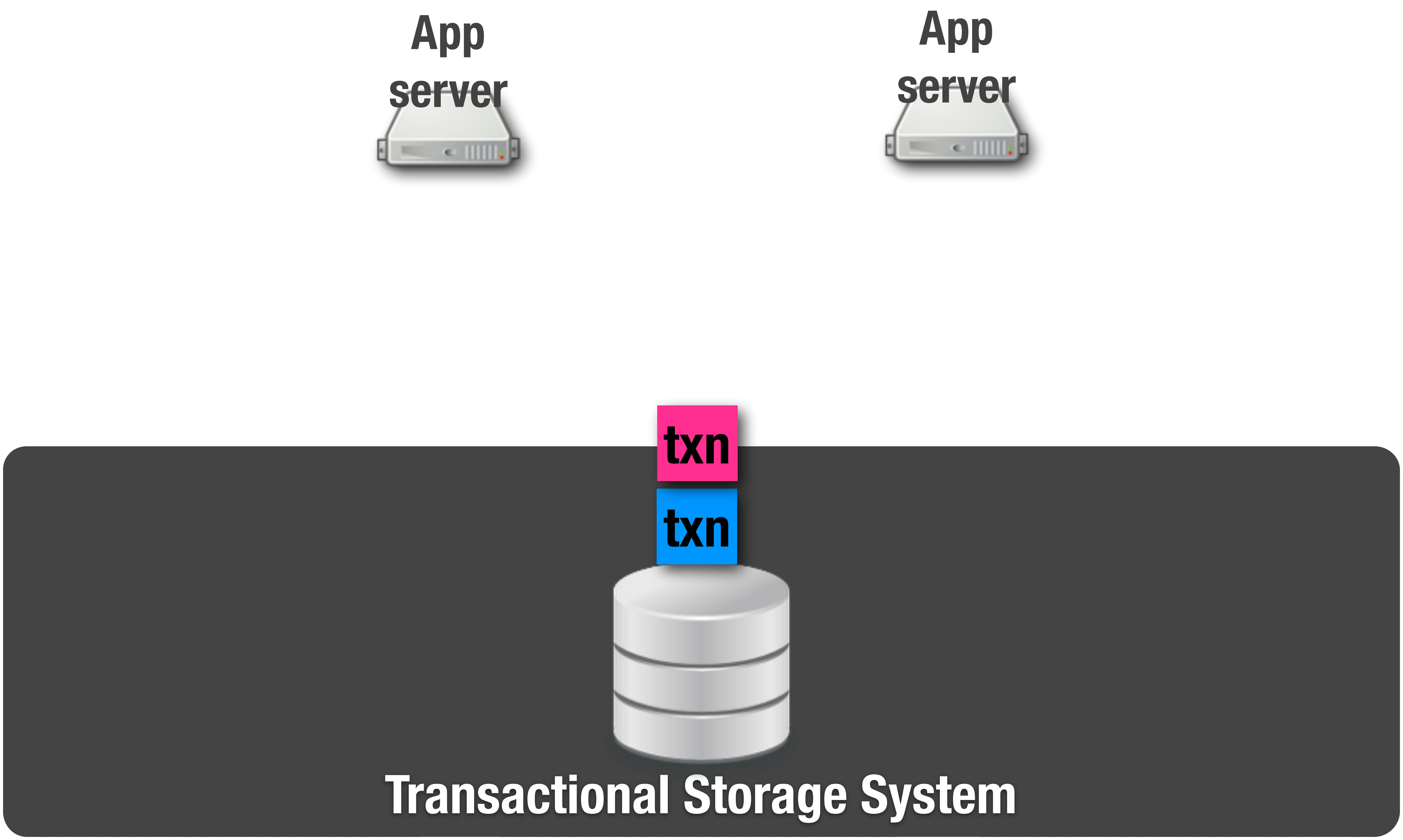
Why is consistency so expensive?



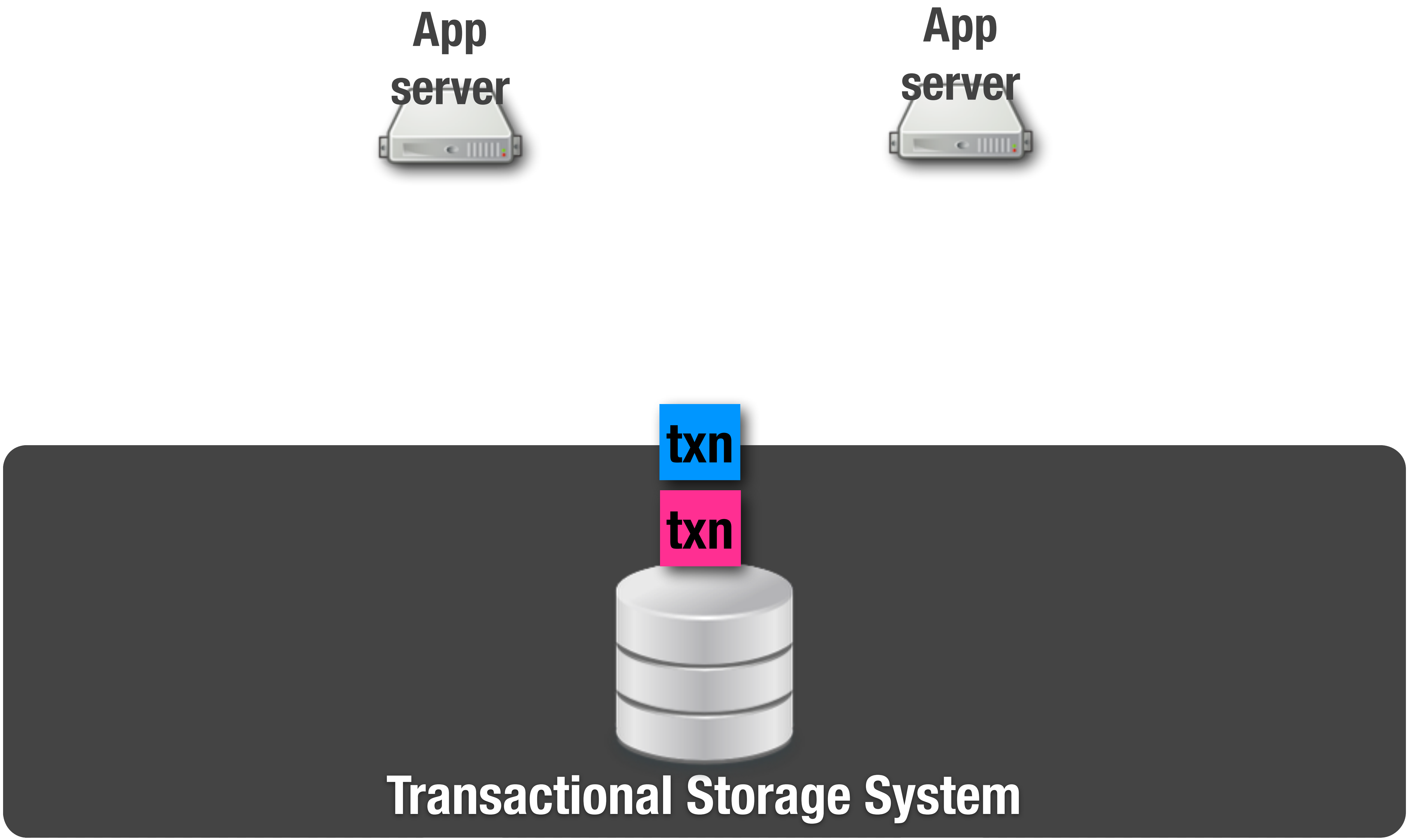
Why is consistency so expensive?



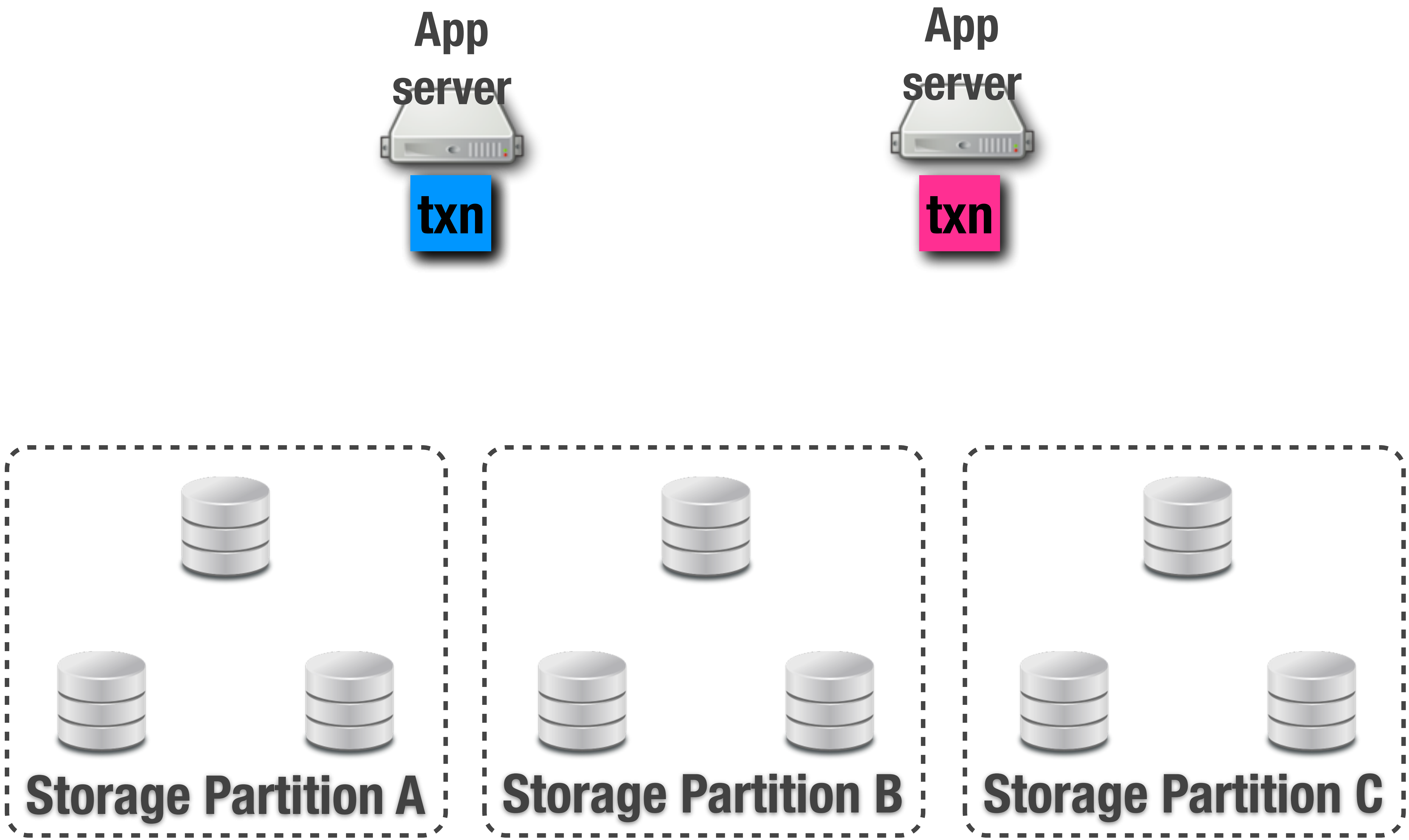
Why is consistency so expensive?



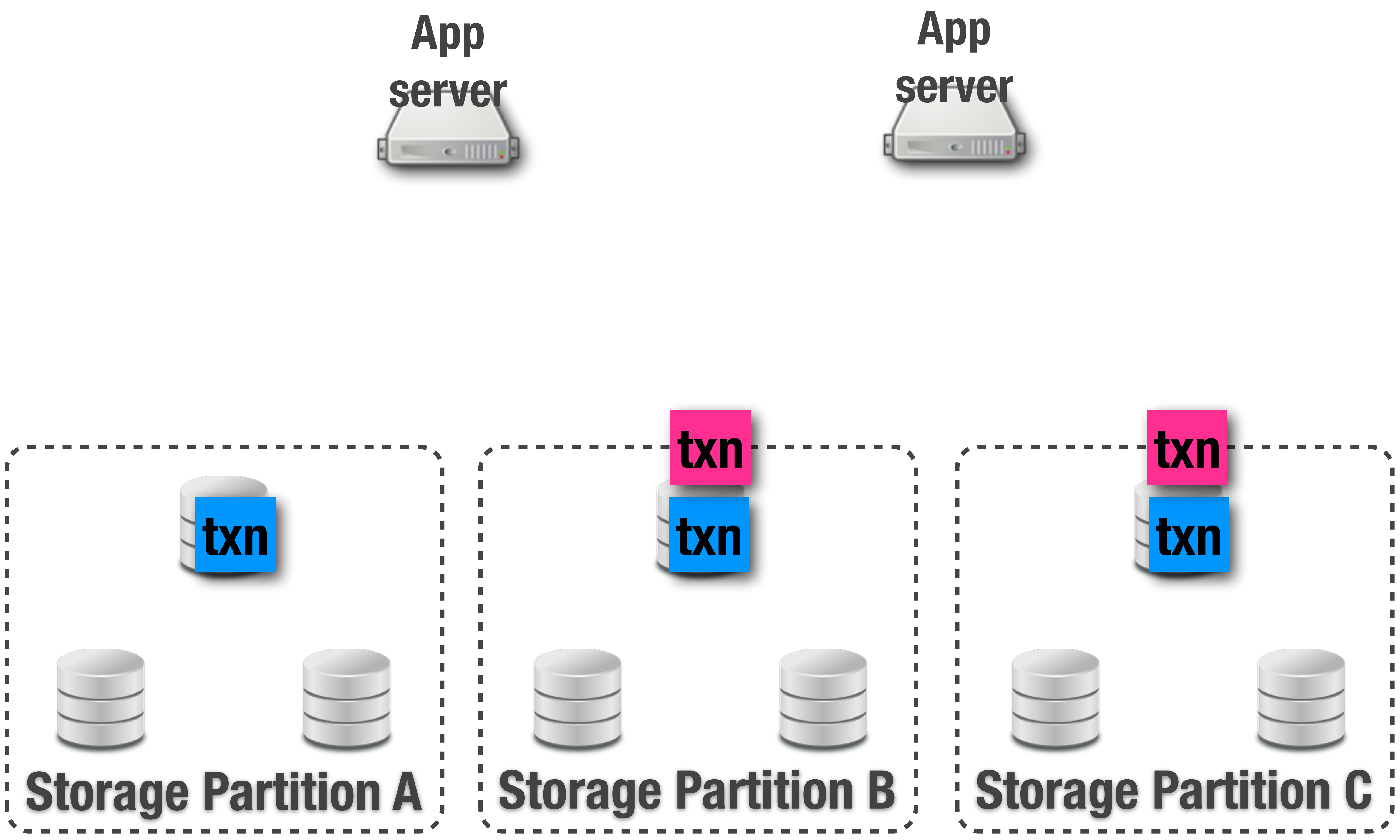
Why is consistency so expensive?



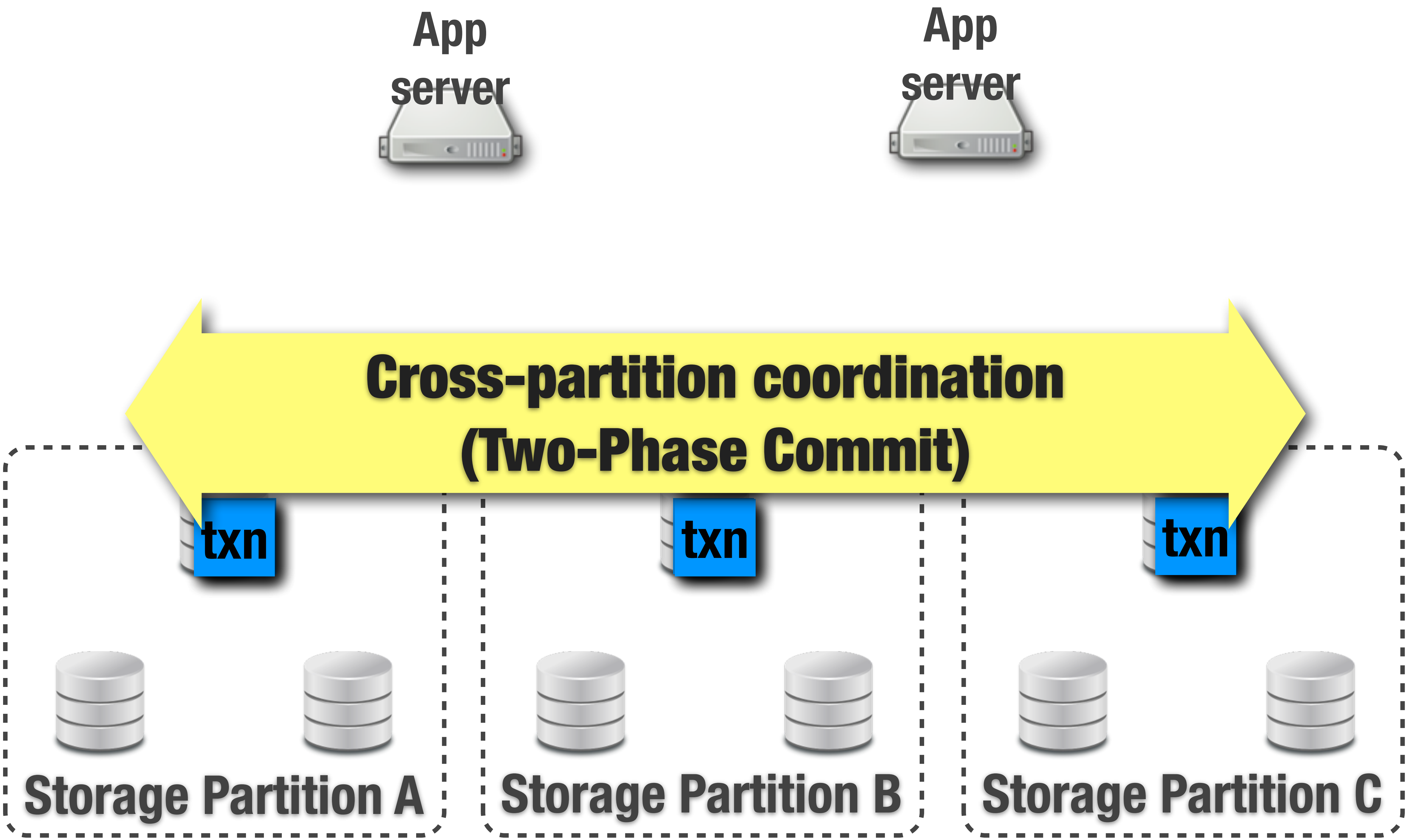
Why is consistency so expensive?



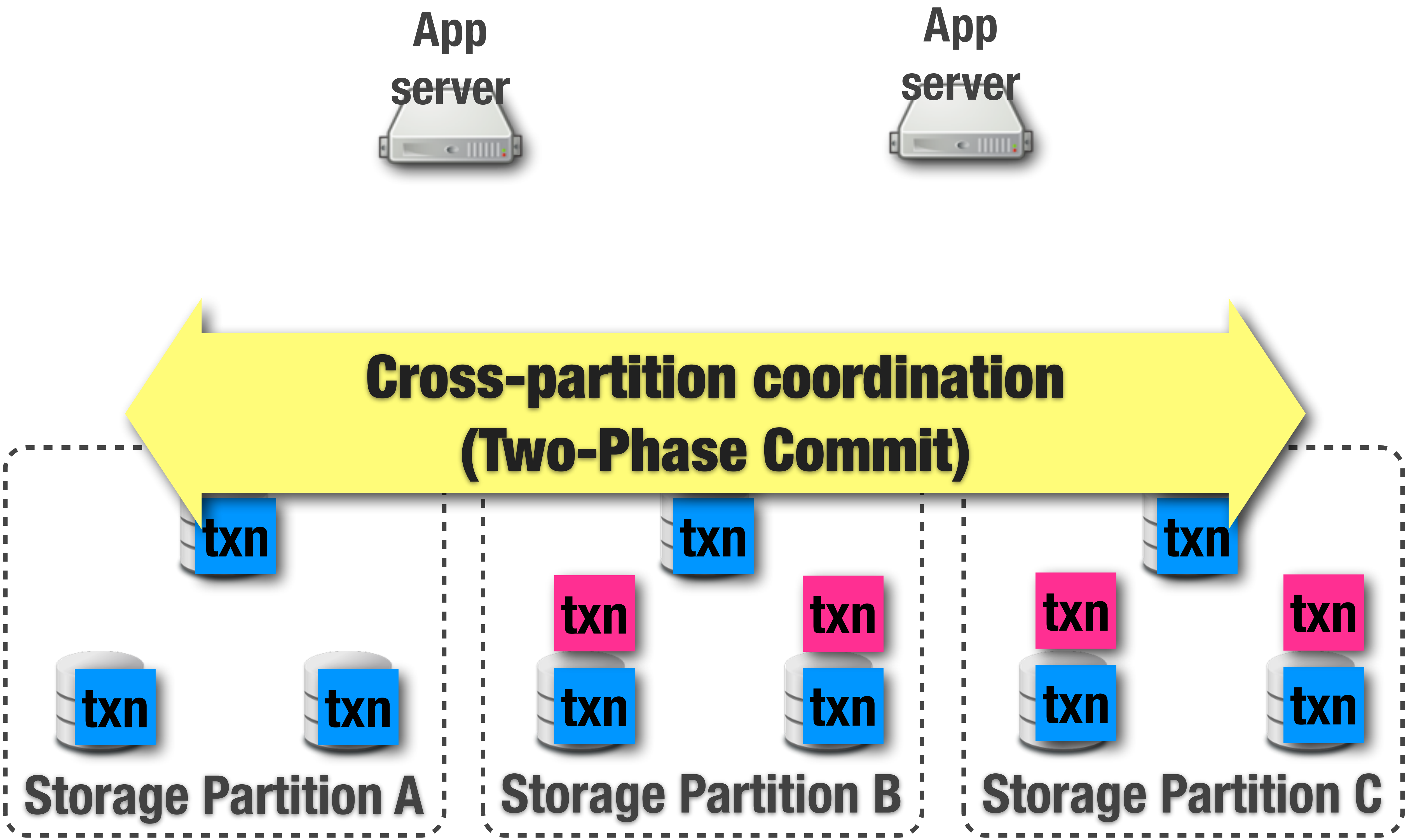
Why is consistency so expensive?



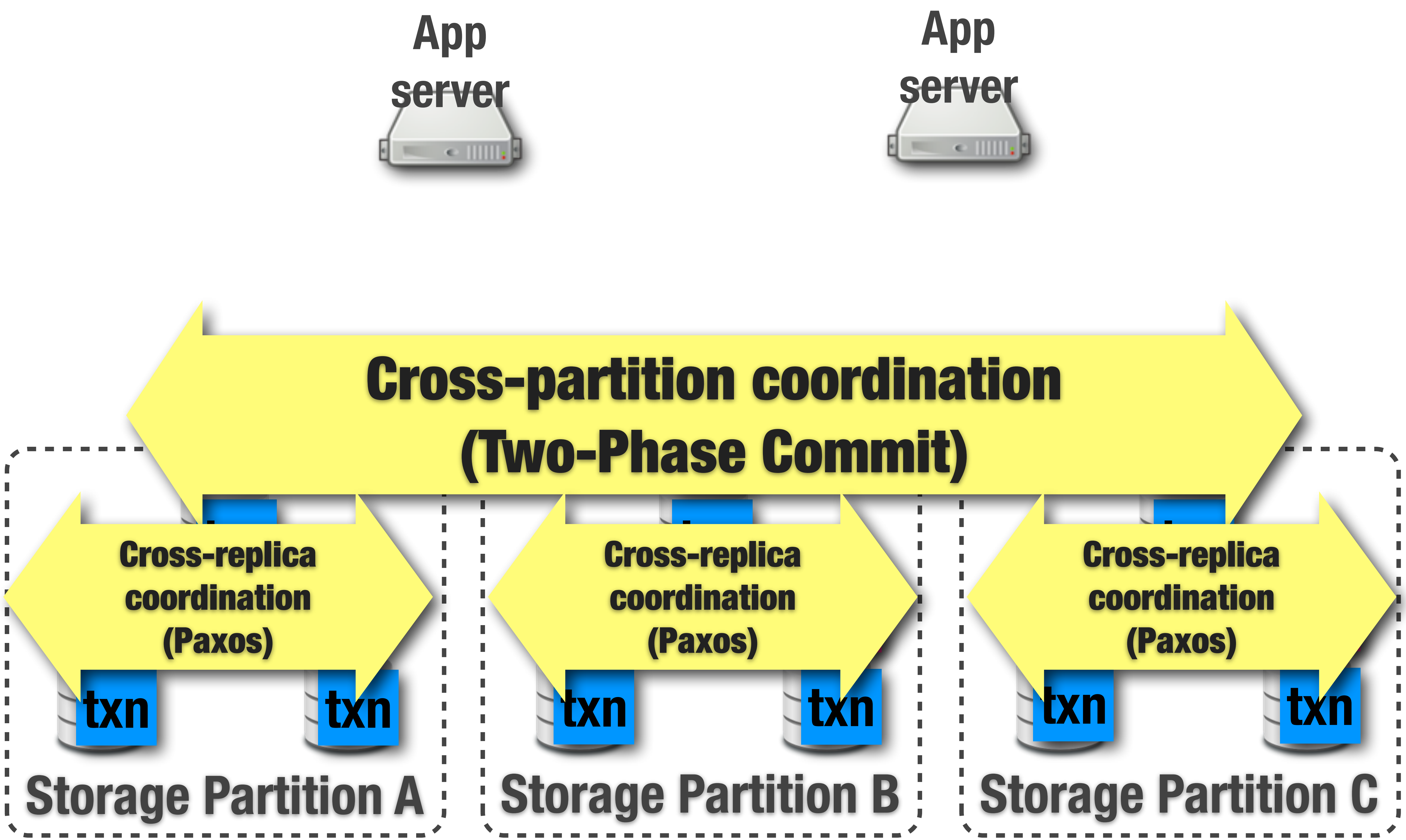
Why is consistency so expensive?



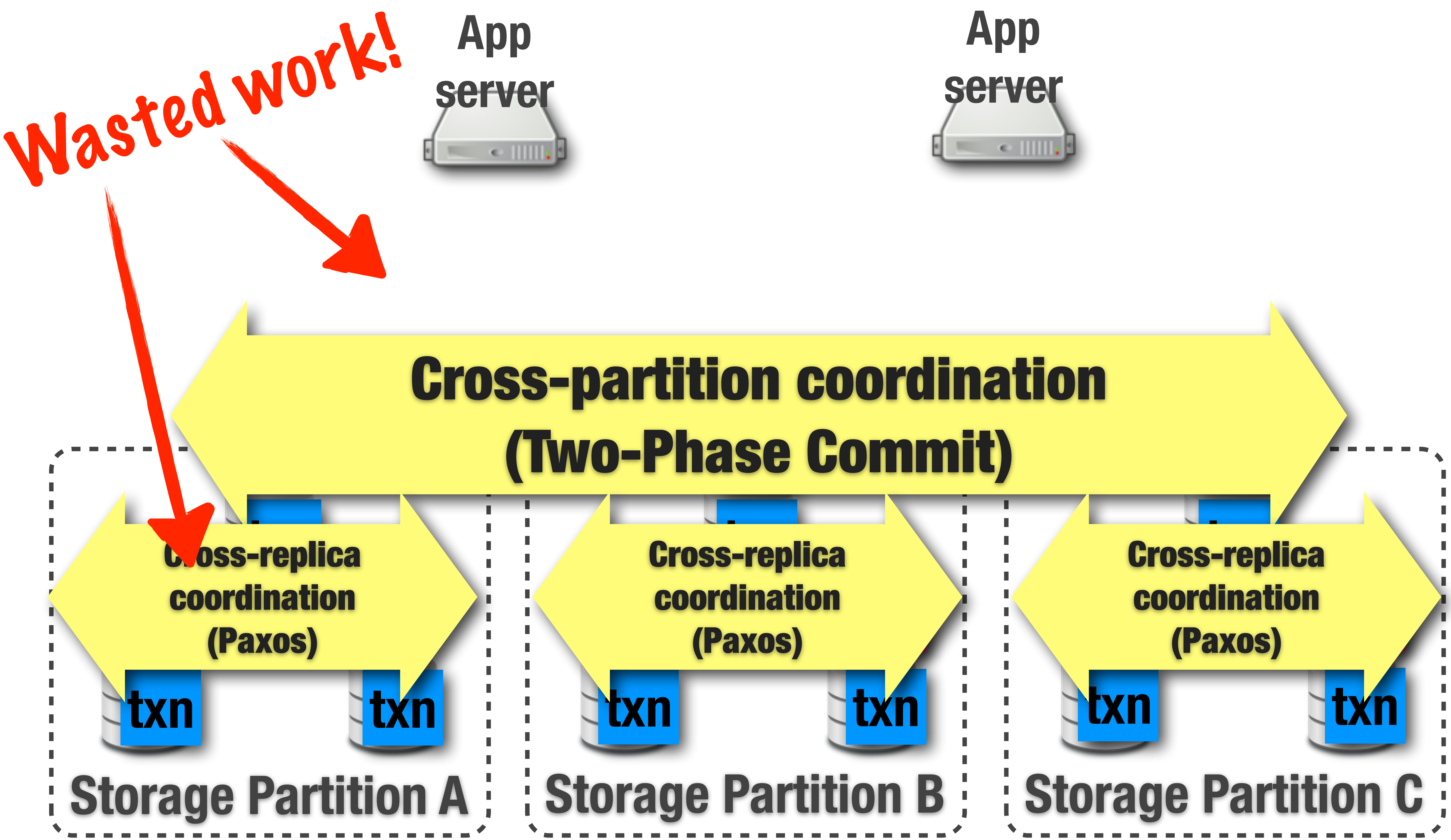
Why is consistency so expensive?



Why is consistency so expensive?



Why is consistency so expensive?



Why is consistency so expensive?

Existing transactional storage systems use a transaction protocol and a replication protocol that **both** enforce strong consistency.

Rest of this talk

1. The cost of strong consistency
2. <u>TAPIR - the Transactional Application Protocol for Inconsistent Replication</u>
3. Evaluation
4. Summary

TAPIR

Transactional Application Protocol for Inconsistent Replication

TAPIR

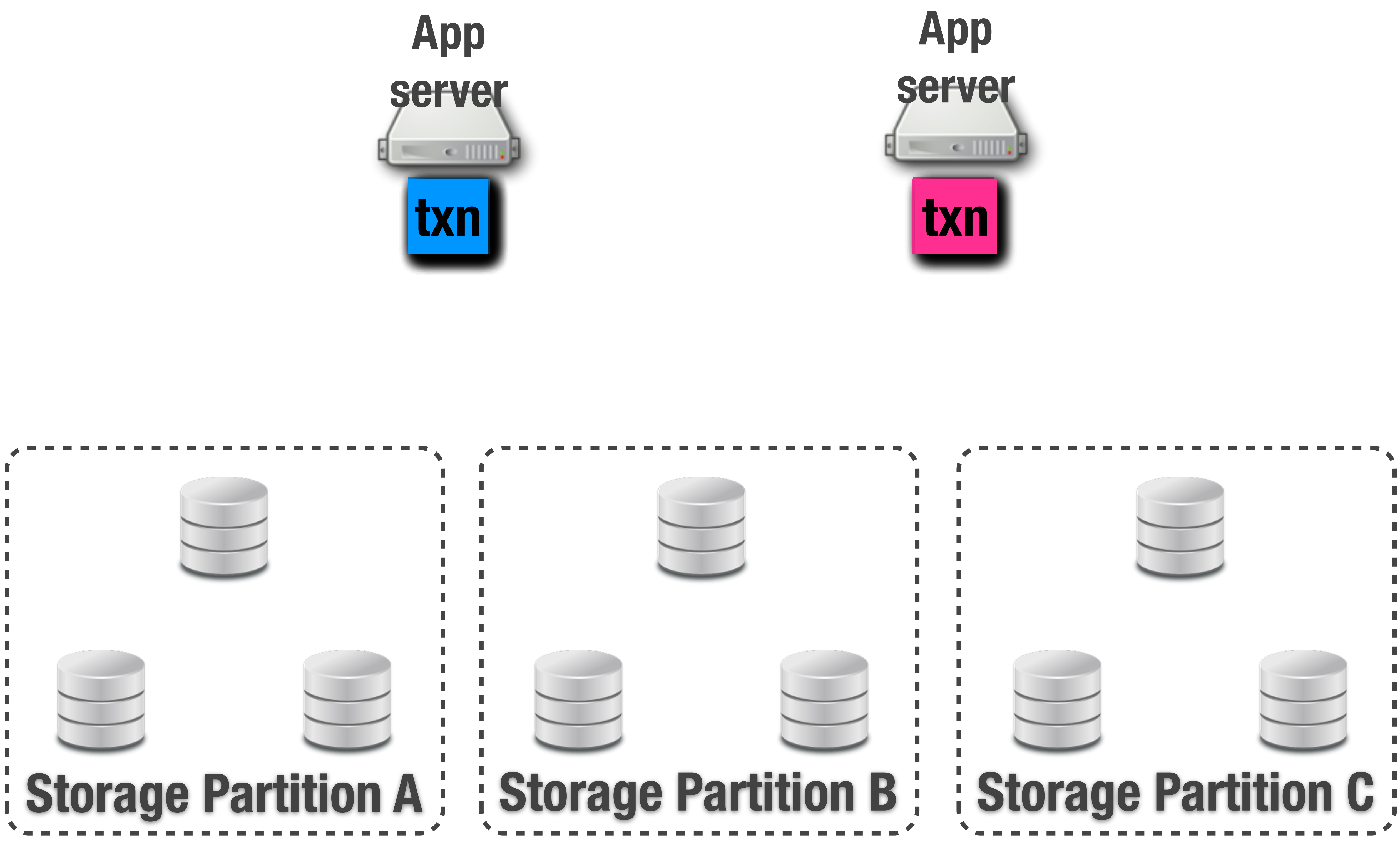
The first transaction protocol to provide distributed transactions with strong consistency using a replication protocol with no consistency.

Inconsistent Replication

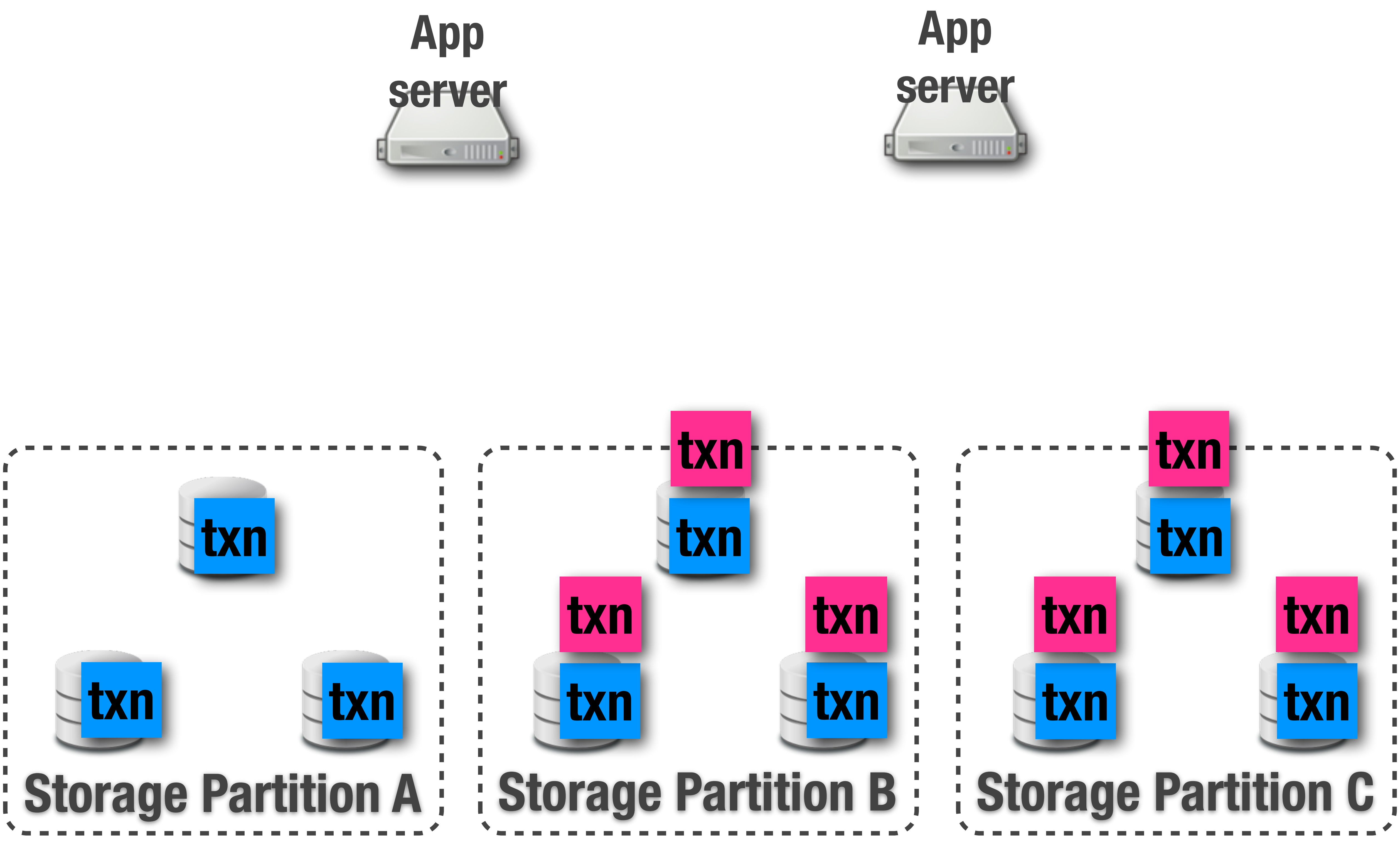
A new replication protocol that:

- Provides fault-tolerance without consistency
- Supports unordered record, instead of ordered log
- Requires no cross-replica coordination
- Does not rely on synchronous disk writes

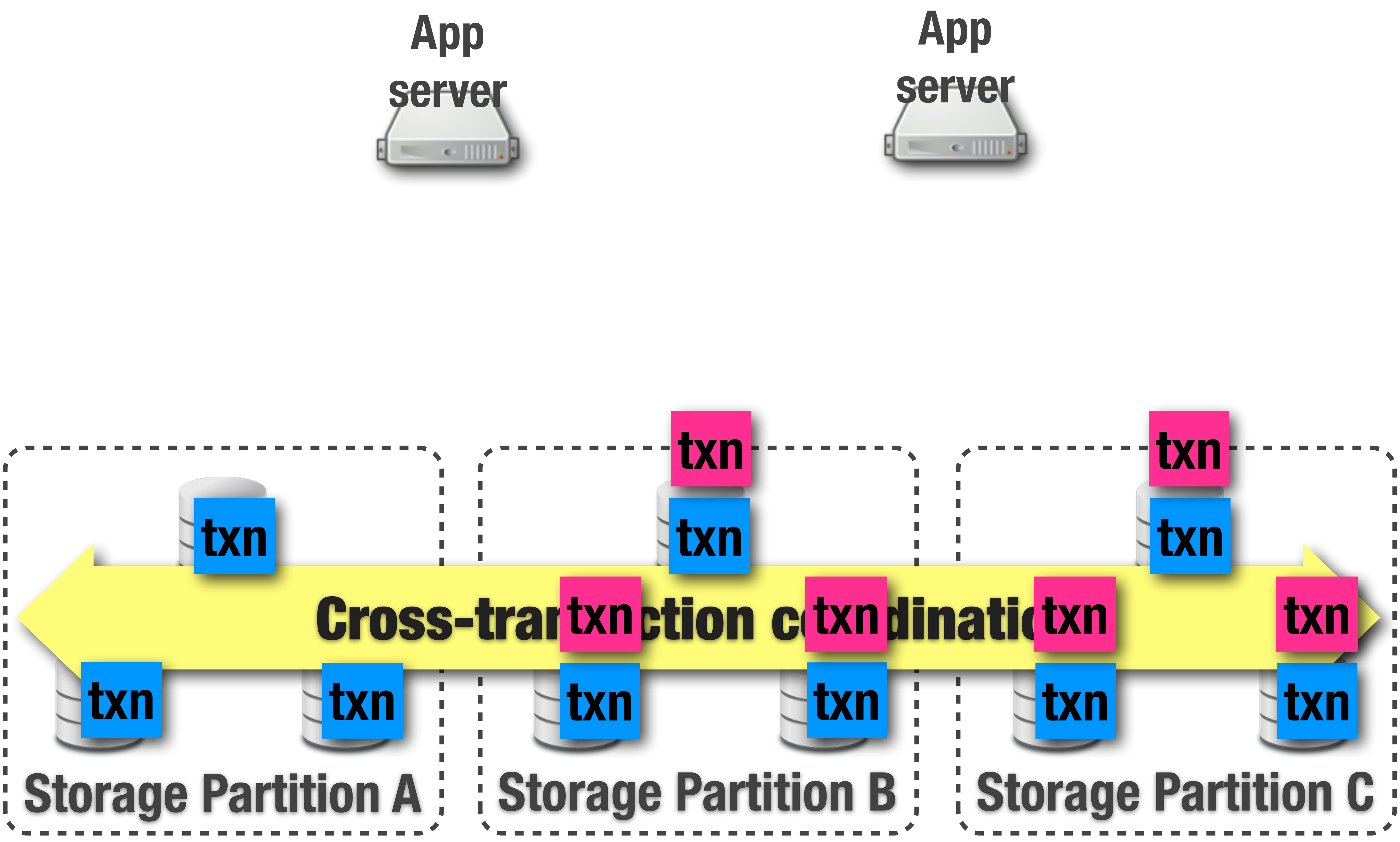
TAPIR


App server
txn
App server
txn
Storage Partition A
Storage Partition B
Storage Partition C

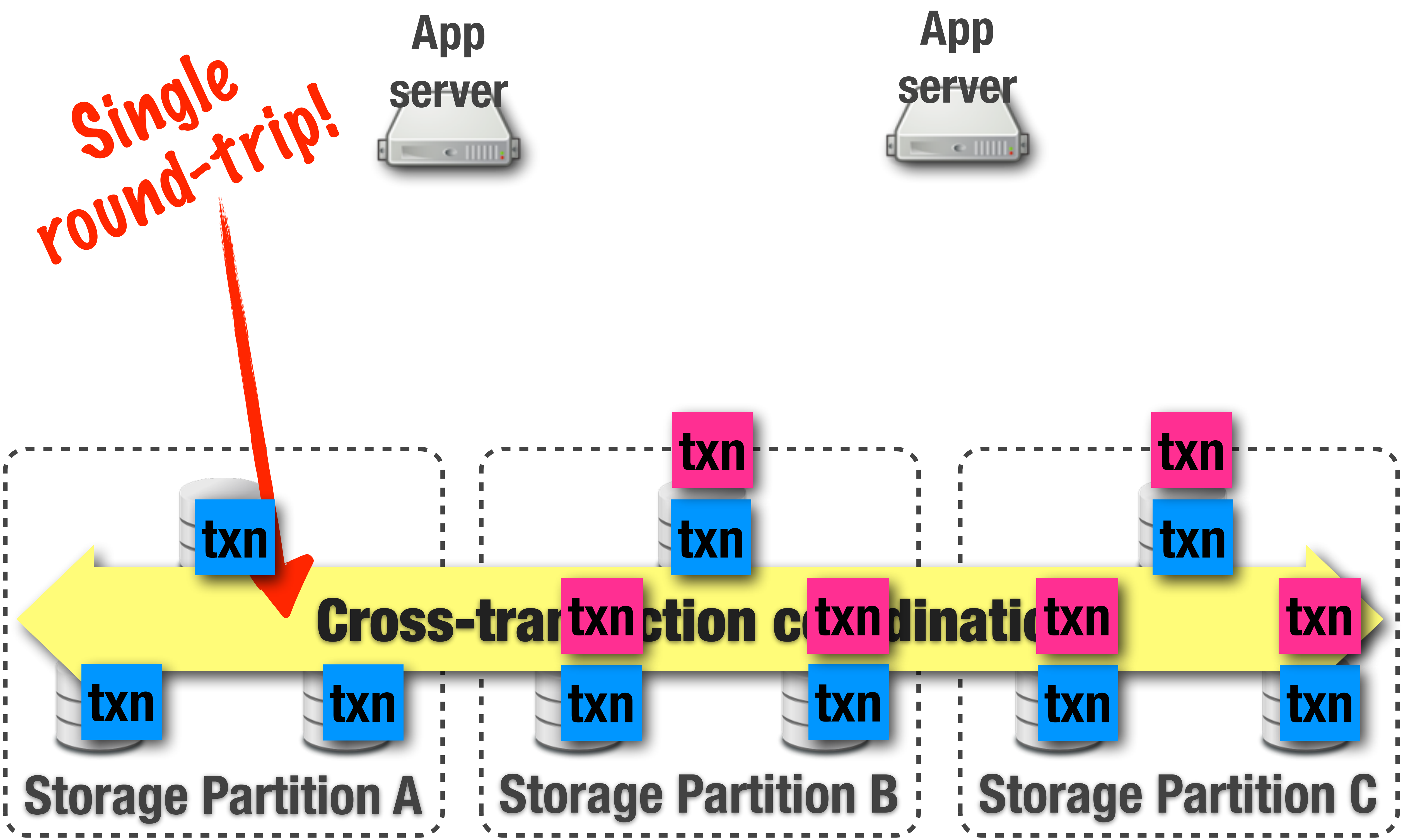
TAPIR


App server
App server
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
Storage Partition A
Storage Partition B
Storage Partition C

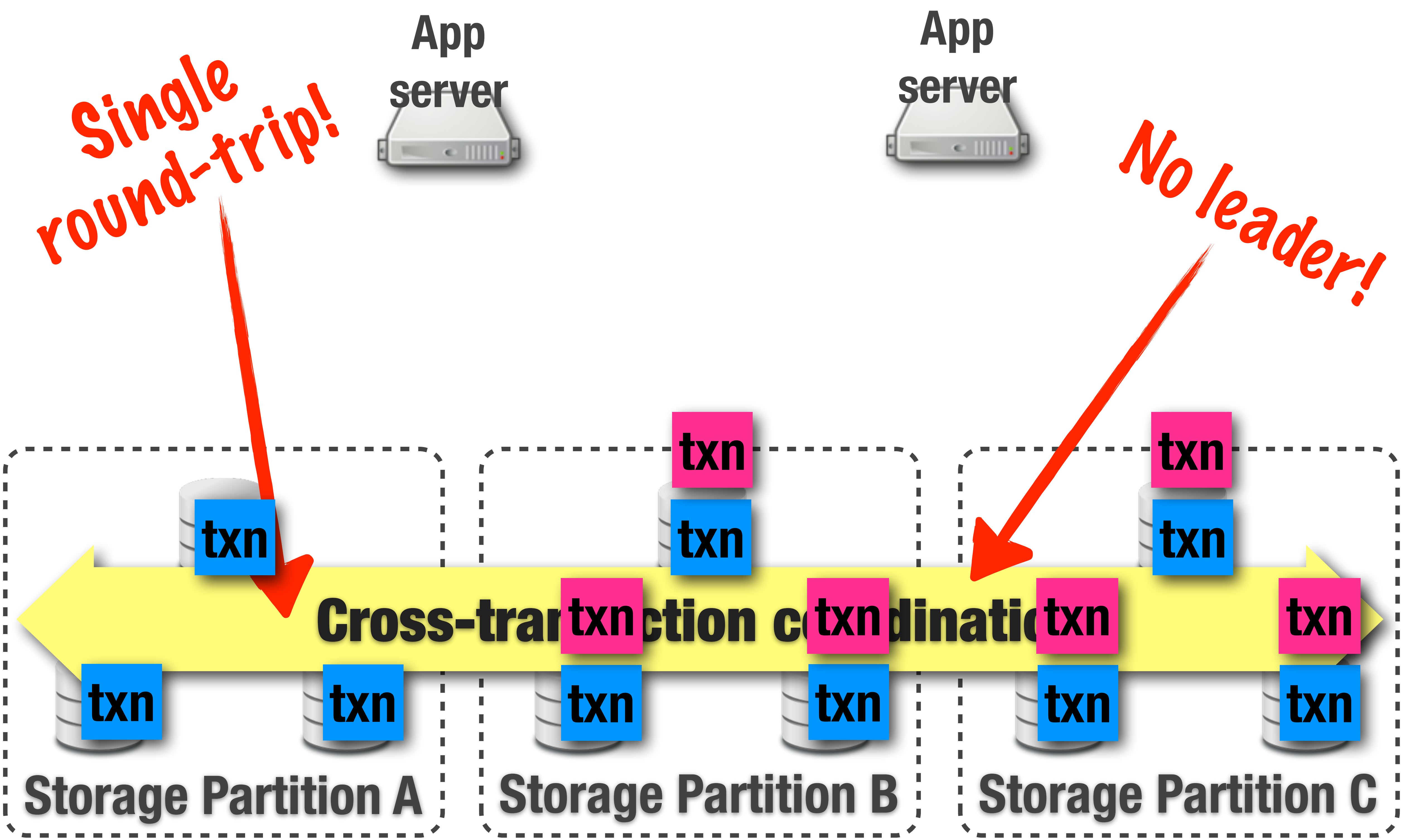
TAPIR


App server
App server
txn
txn
txn
txn
txn
Cross-tran txn ction c txn dinatio txn
txn
txn
txn
txn
txn
txn
txn
txn
Storage Partition A
Storage Partition B
Storage Partition C

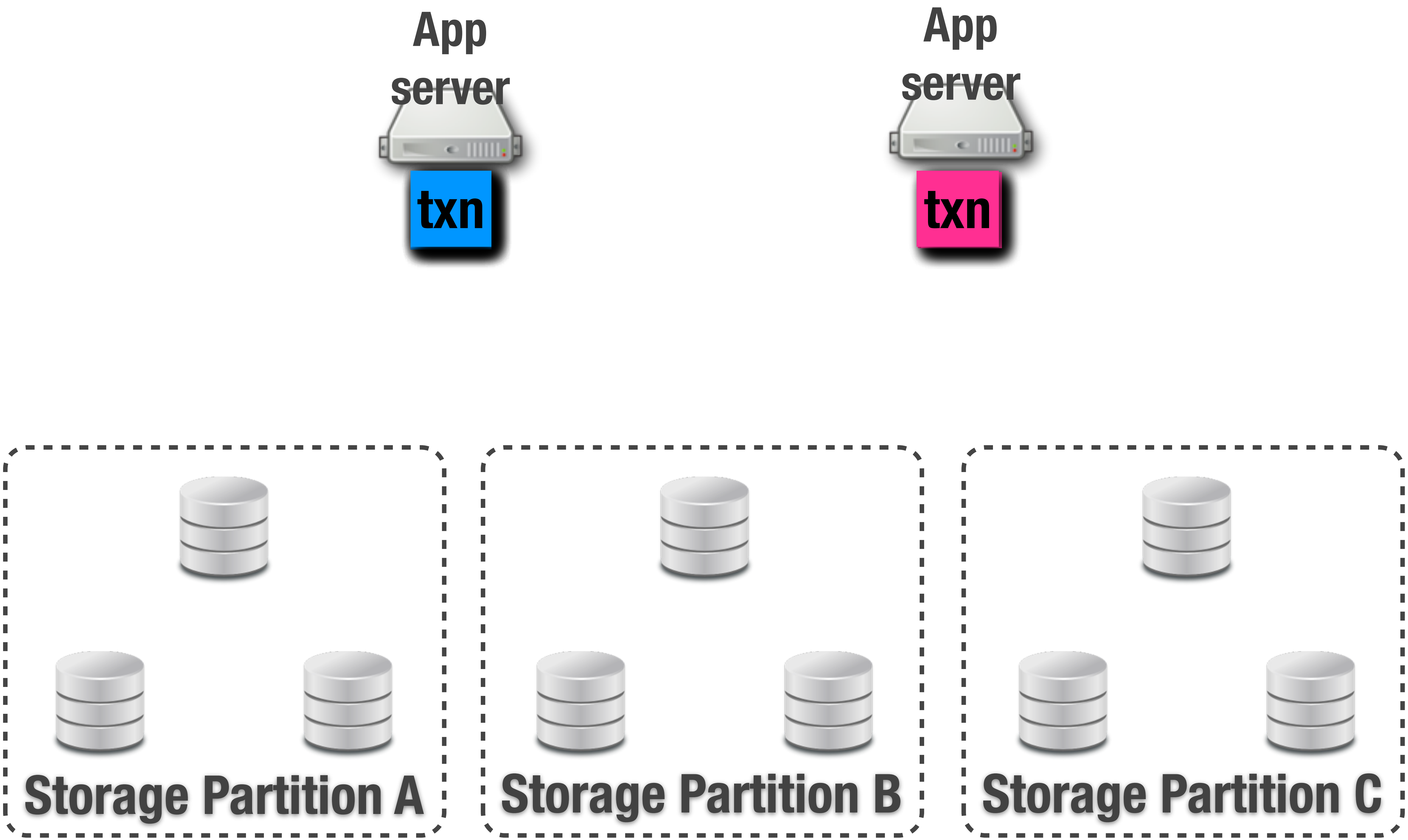
TAPIR


App server
App server
Single round-trip!
txn
Storage Partition A
Storage Partition B
Storage Partition C

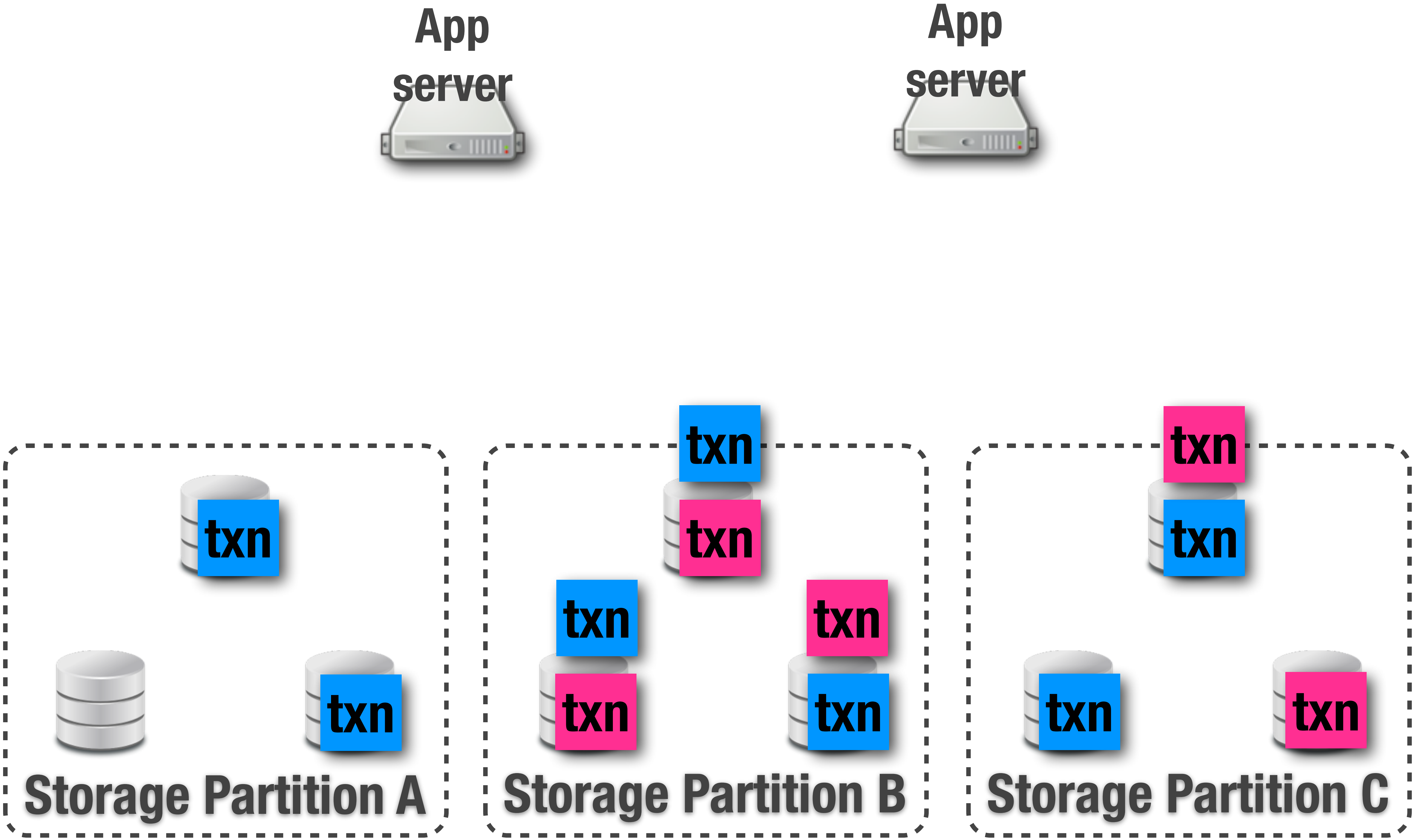
TAPIR


App server
App server
Single round-trip!
No leader!
txn
Cross-transaction coordination
Storage Partition A
Storage Partition B
Storage Partition C

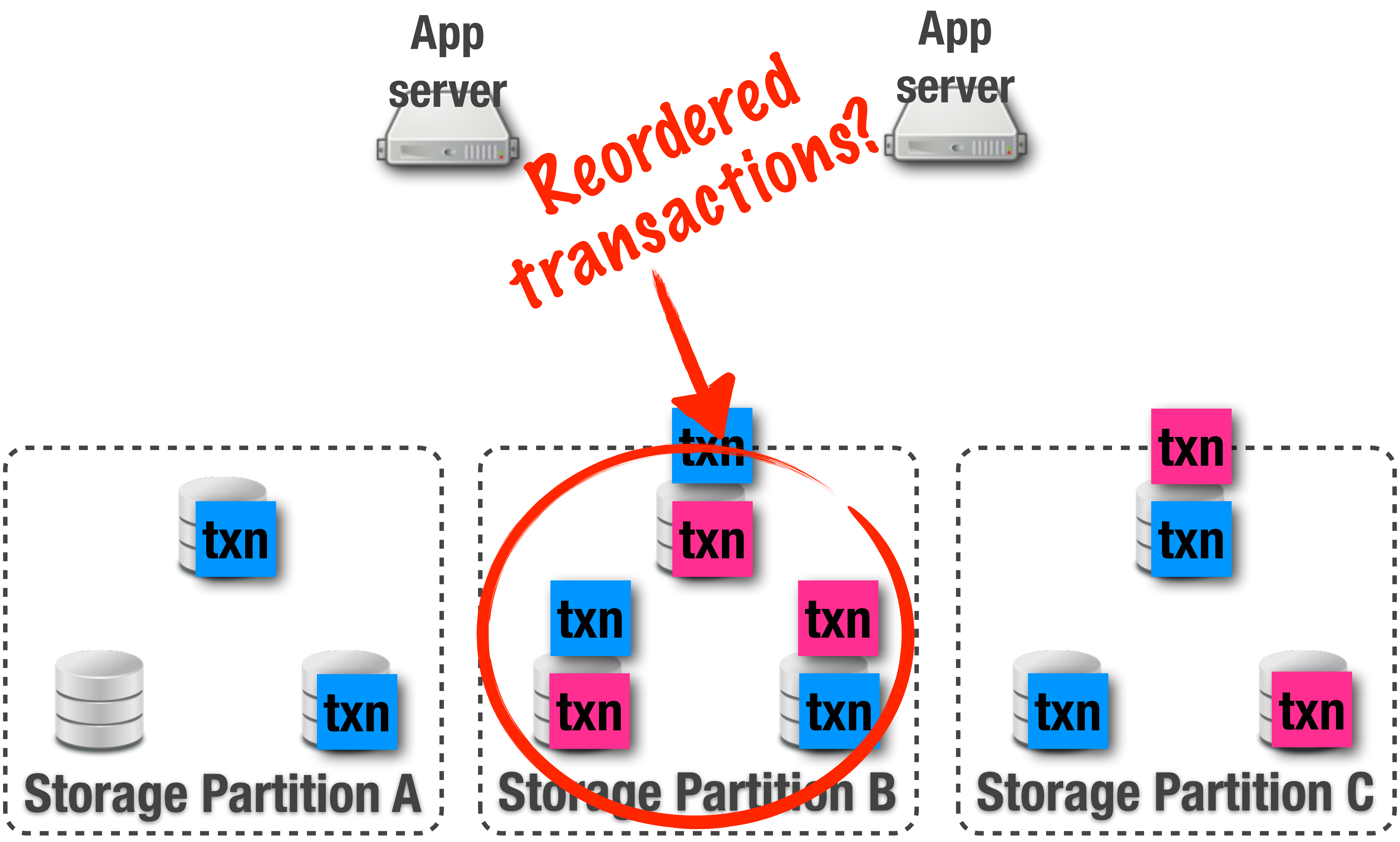
TAPIR


App server
txn
App server
txn
Storage Partition A
Storage Partition B
Storage Partition C

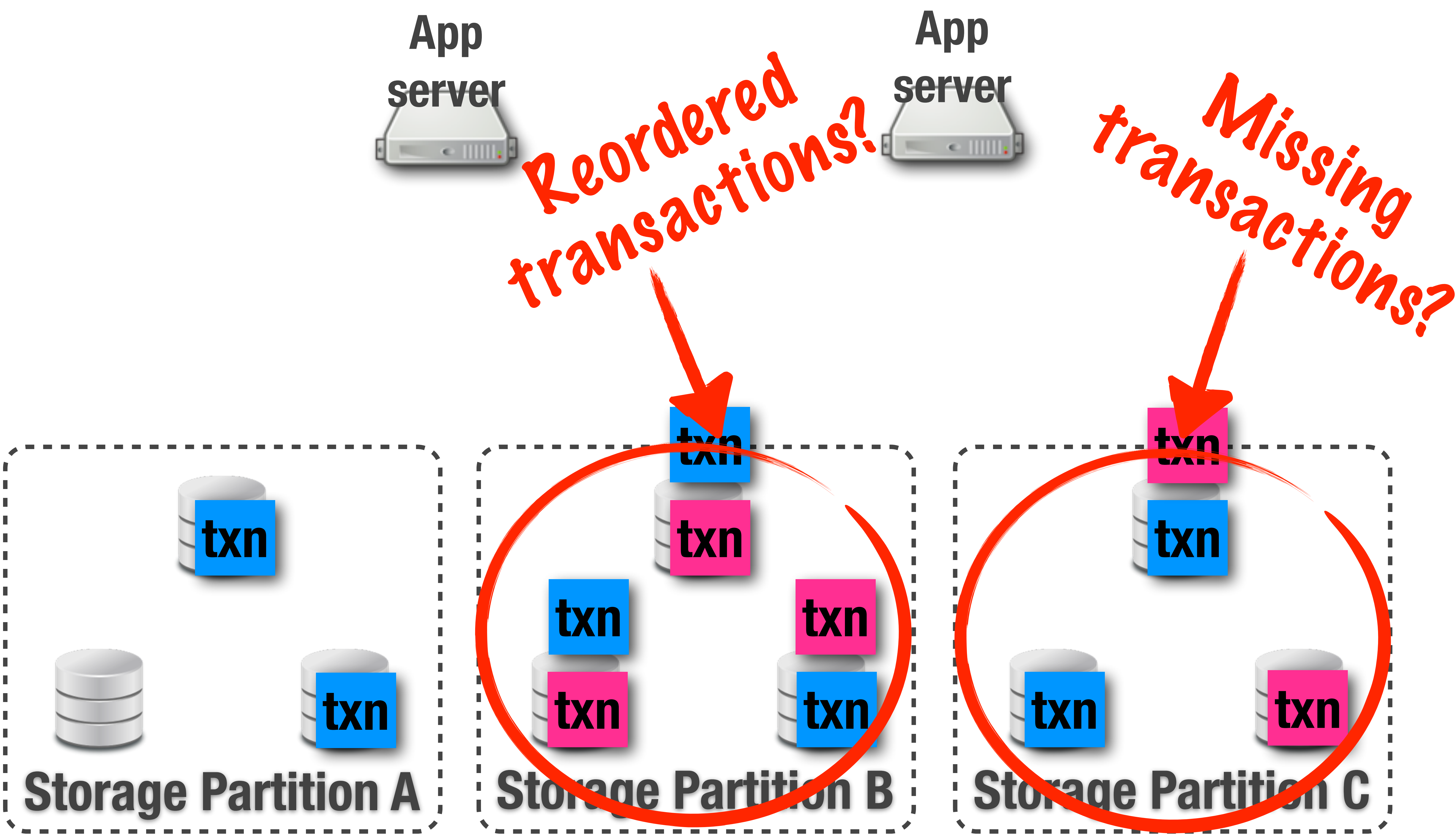
TAPIR


App server
App server
txn
txn
txn
Storage Partition A
txn
txn
txn
txn
txn
txn
Storage Partition B
txn
txn
txn
txn
Storage Partition C

TAPIR


App server
App server
Reordered transactions?
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
Storage Partition A
Storage Partition B
Storage Partition C

TAPIR



App server
App server
Reordered transactions?
Missing transactions?
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
txn
Storage Partition A
Storage Partition B
Storage Partition C

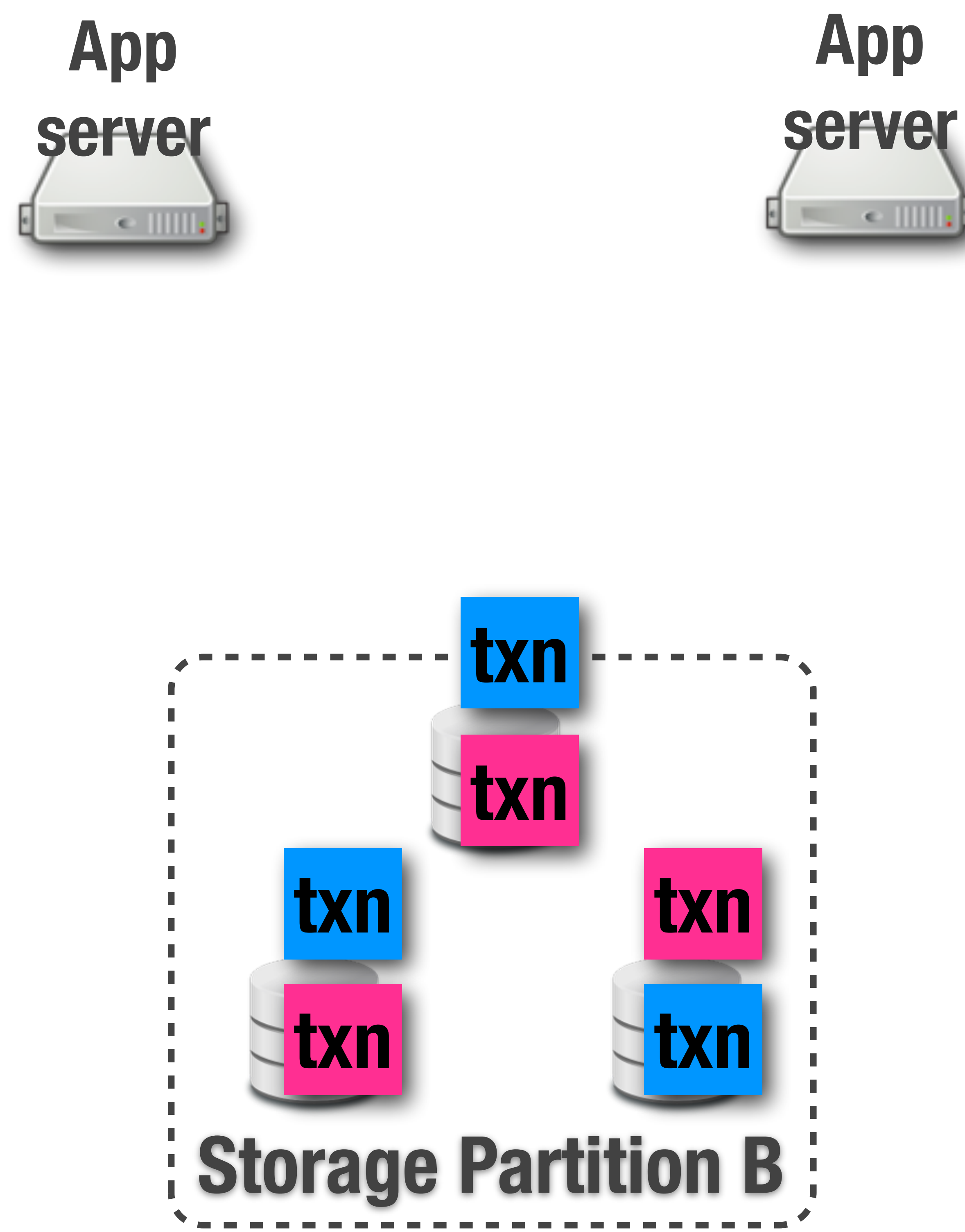
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for transaction ordering.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.

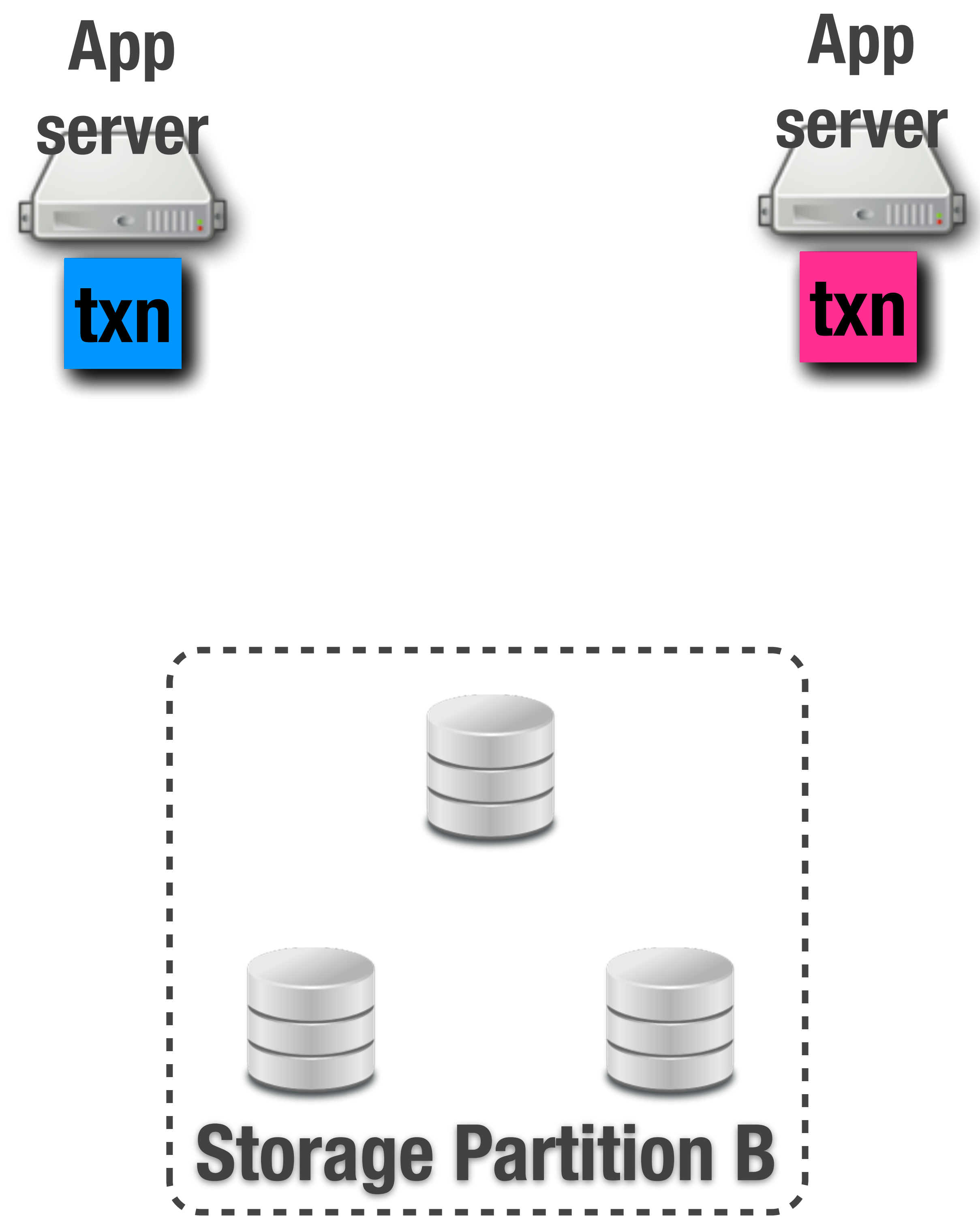
TAPIR Technique: Transaction ordering with **loosely synchronized clocks**

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.



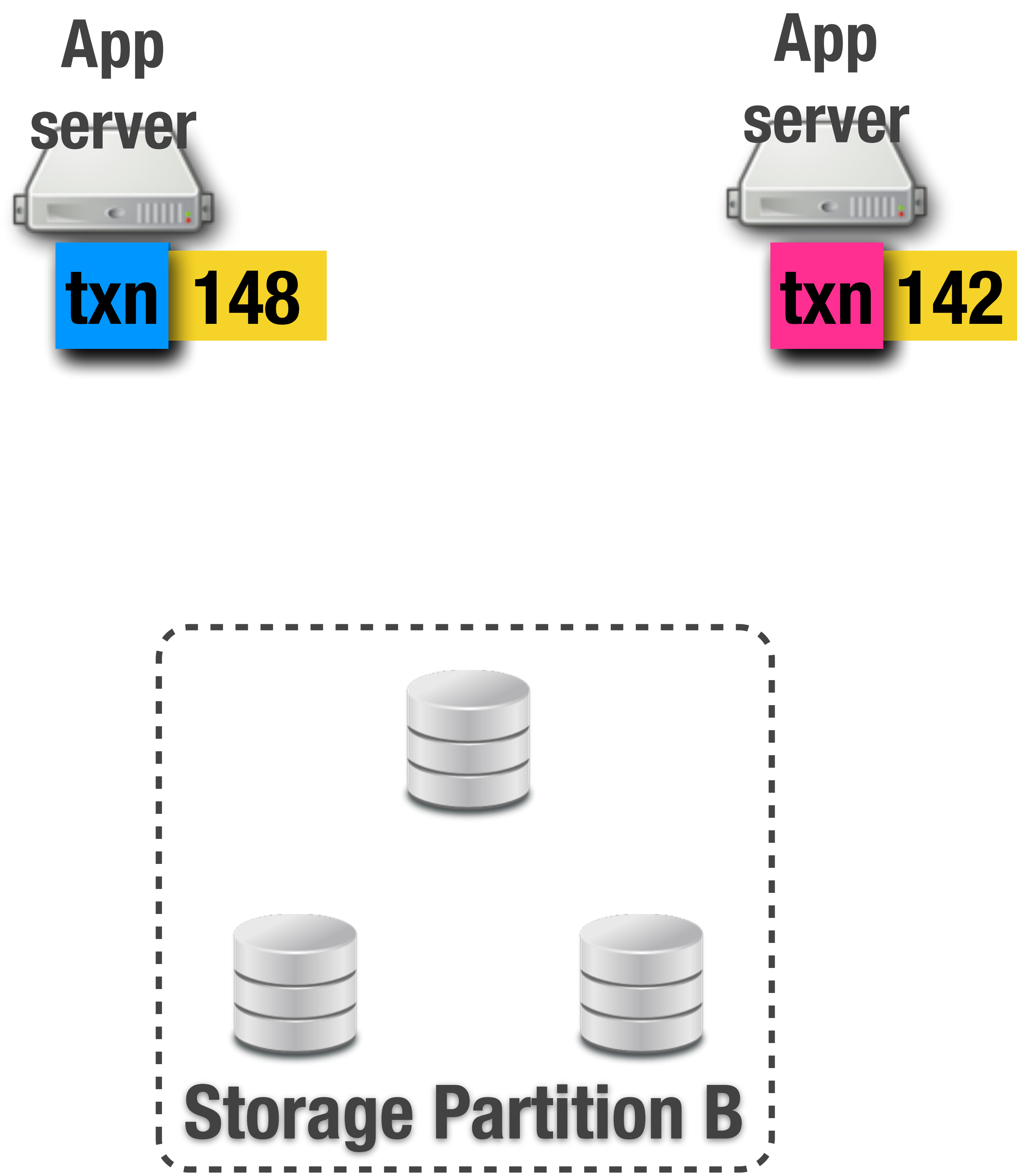
TAPIR Technique: Transaction ordering with **loosely synchronized clocks**

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.



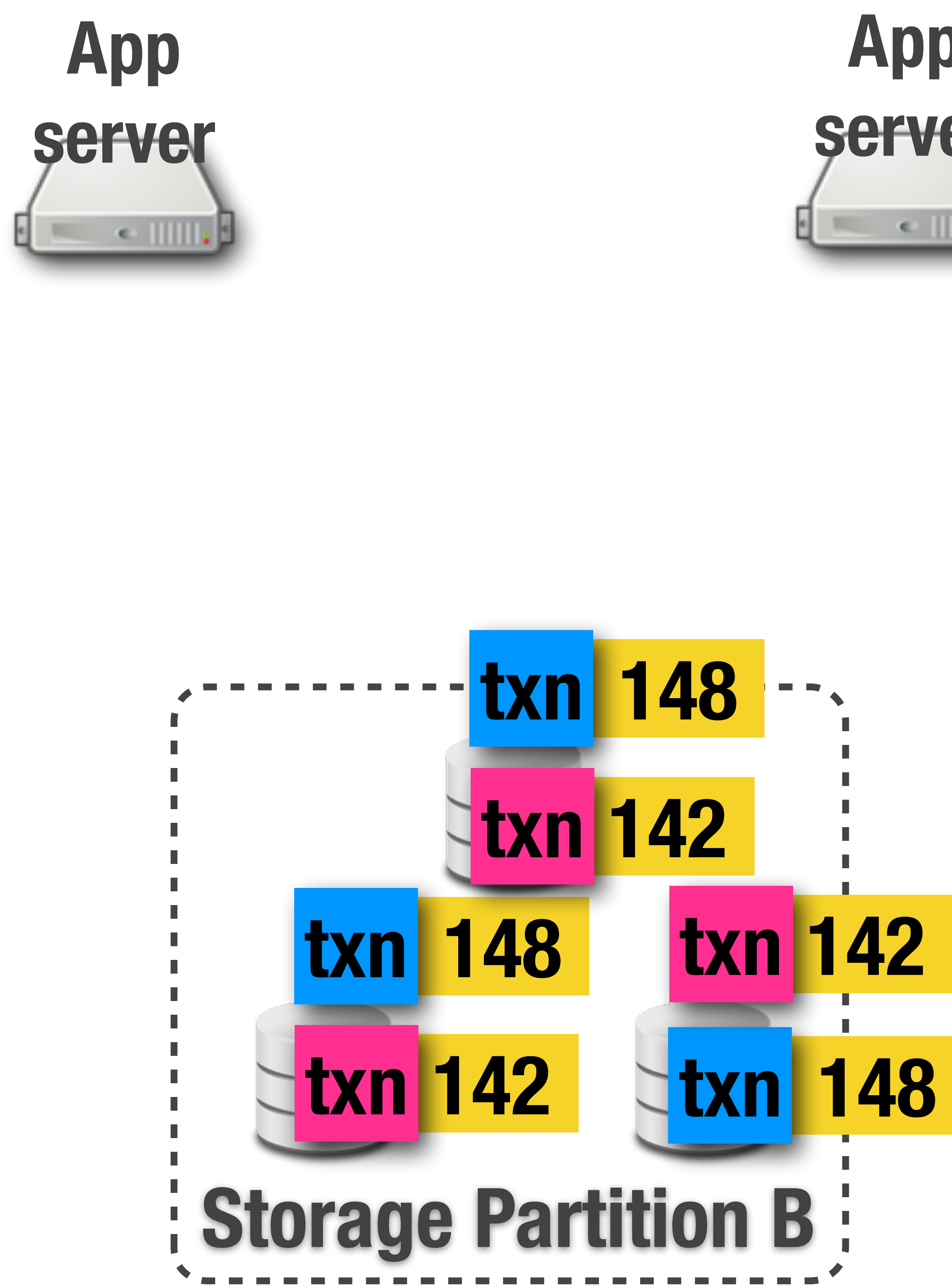
TAPIR Technique: Transaction ordering with **loosely synchronized clocks**

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.



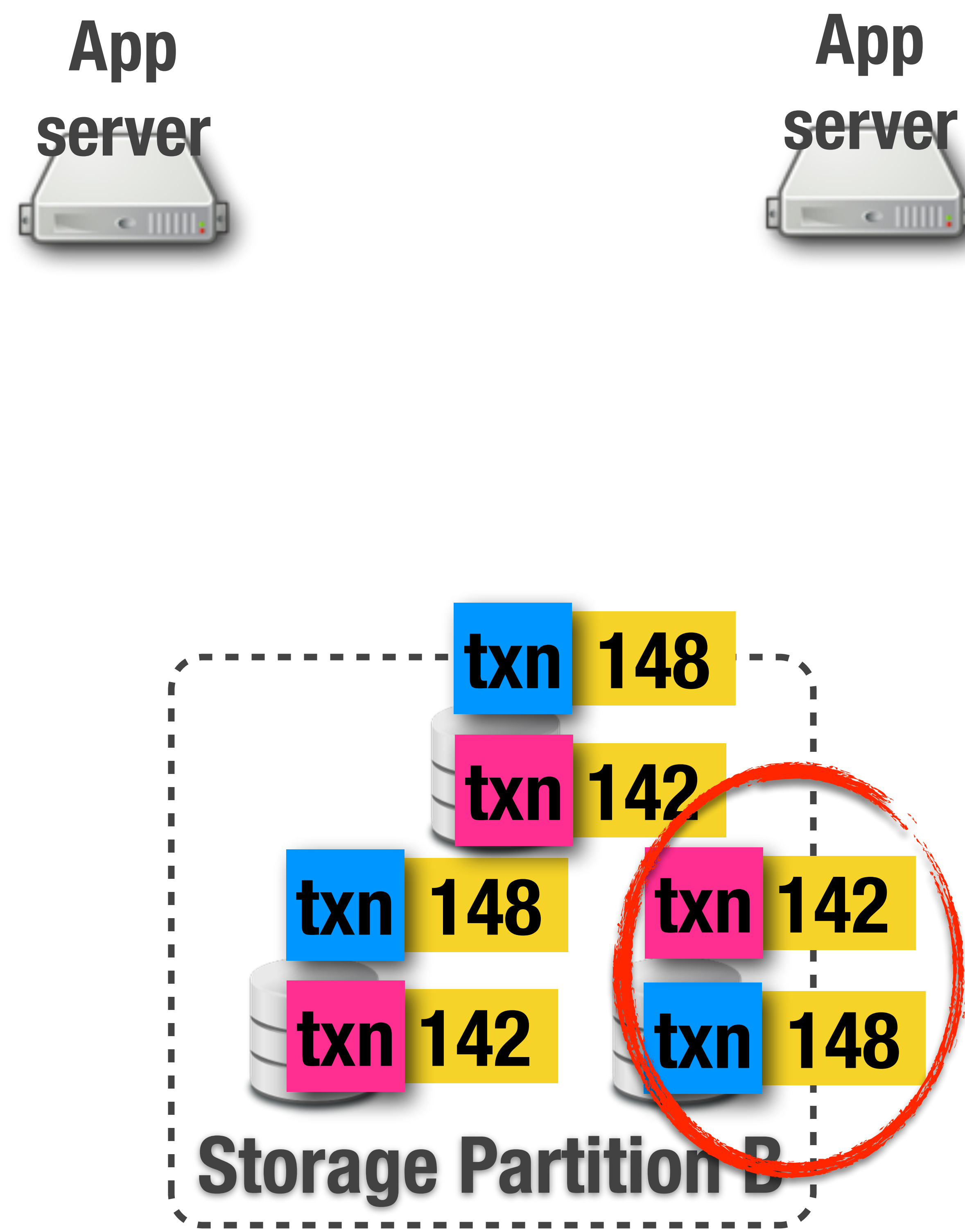
TAPIR Technique: Transaction ordering with **loosely synchronized clocks**

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.



TAPIR Technique: Transaction ordering with **loosely synchronized clocks**

- Clients pick transaction timestamp using local clock.
- Replicas validate transaction at timestamp, regardless of when they receive the transaction.
- Clock synchronization for performance, not correctness.



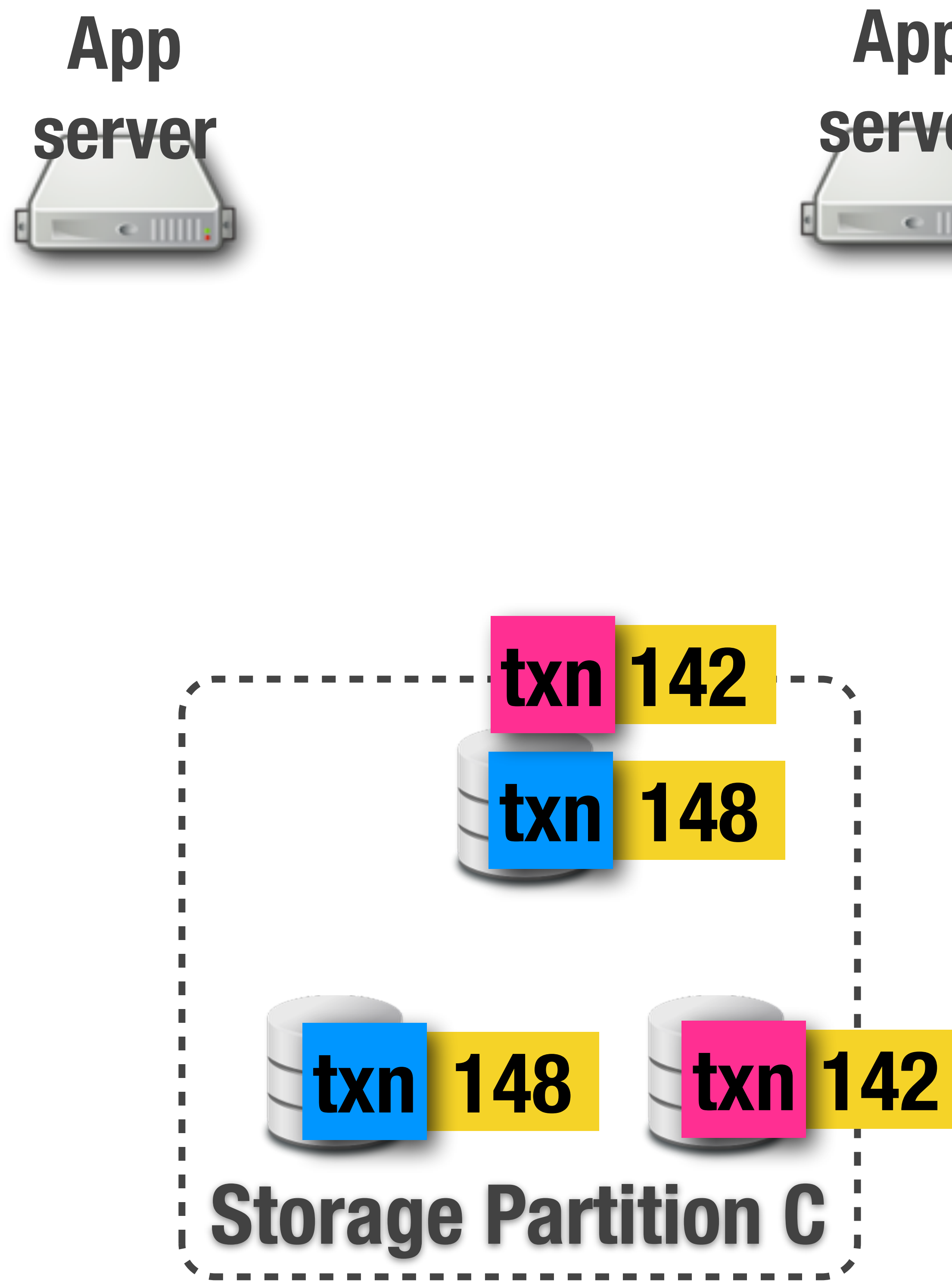
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for optimistic transaction ordering at clients.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.

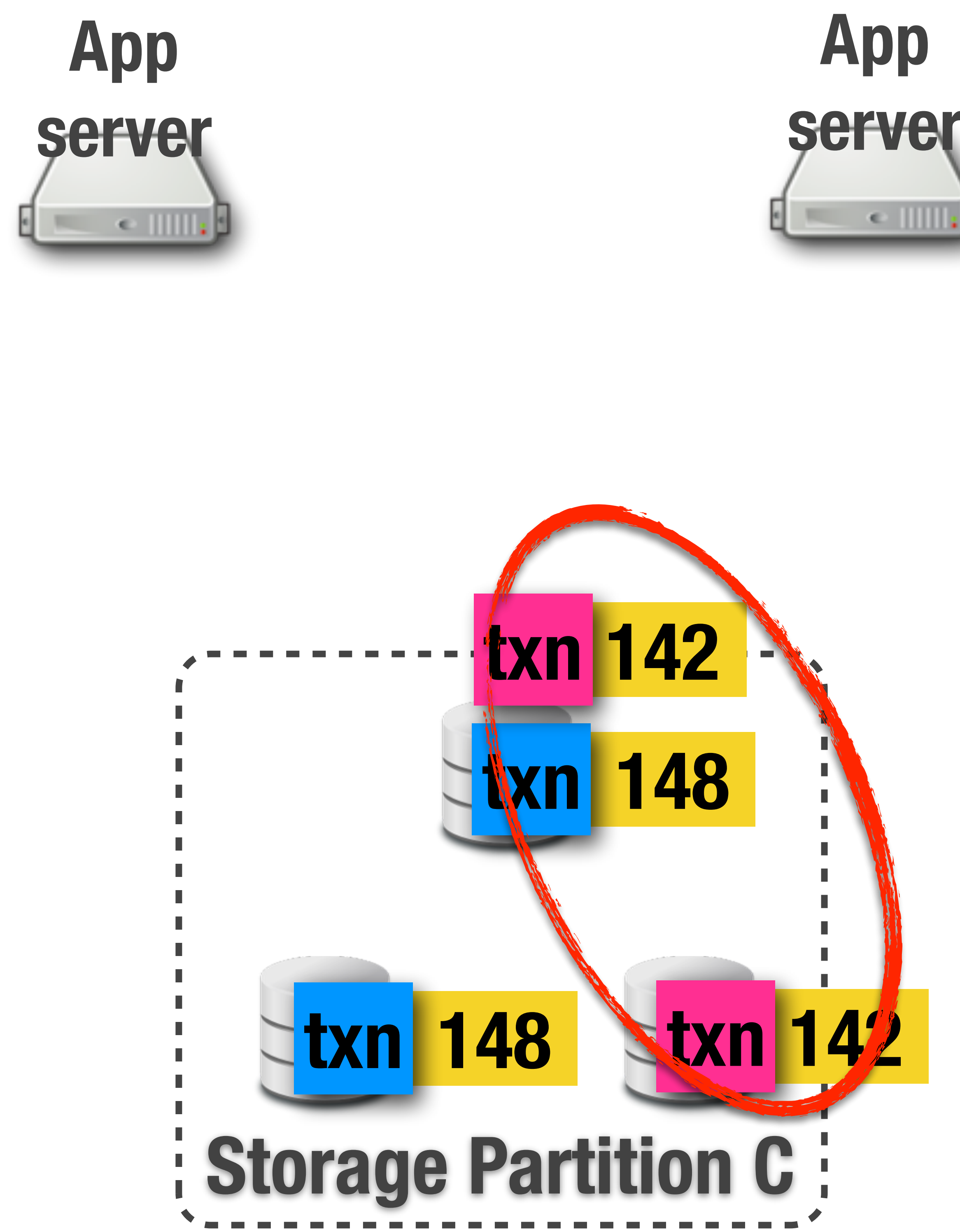
TAPIR Technique: Conflict detection with **optimistic concurrency control**

- OCC checks just one transaction at a time, so a full transaction history is not necessary.
- Every transaction committed at a majority.
- Quorum intersection ensures every transaction is checked.



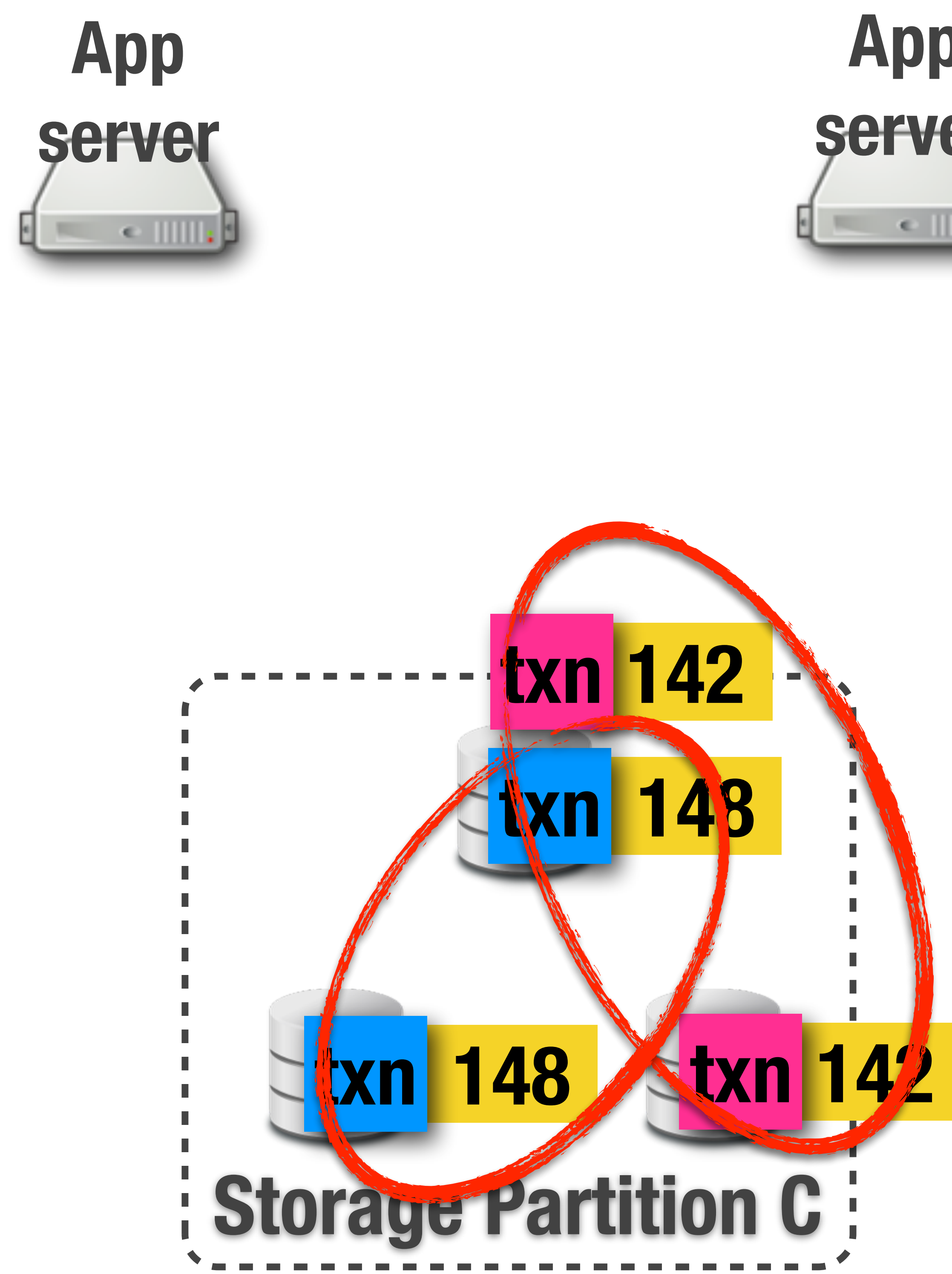
TAPIR Technique: Conflict detection with **optimistic concurrency control**

- OCC checks just one transaction at a time, so a full transaction history is not necessary.
- Every transaction committed at a majority.
- Quorum intersection ensures every transaction is checked.



TAPIR Technique: Conflict detection with **optimistic concurrency control**

- OCC checks just one transaction at a time, so a full transaction history is not necessary.
- Every transaction committed at a majority.
- Quorum intersection ensures every transaction is checked.



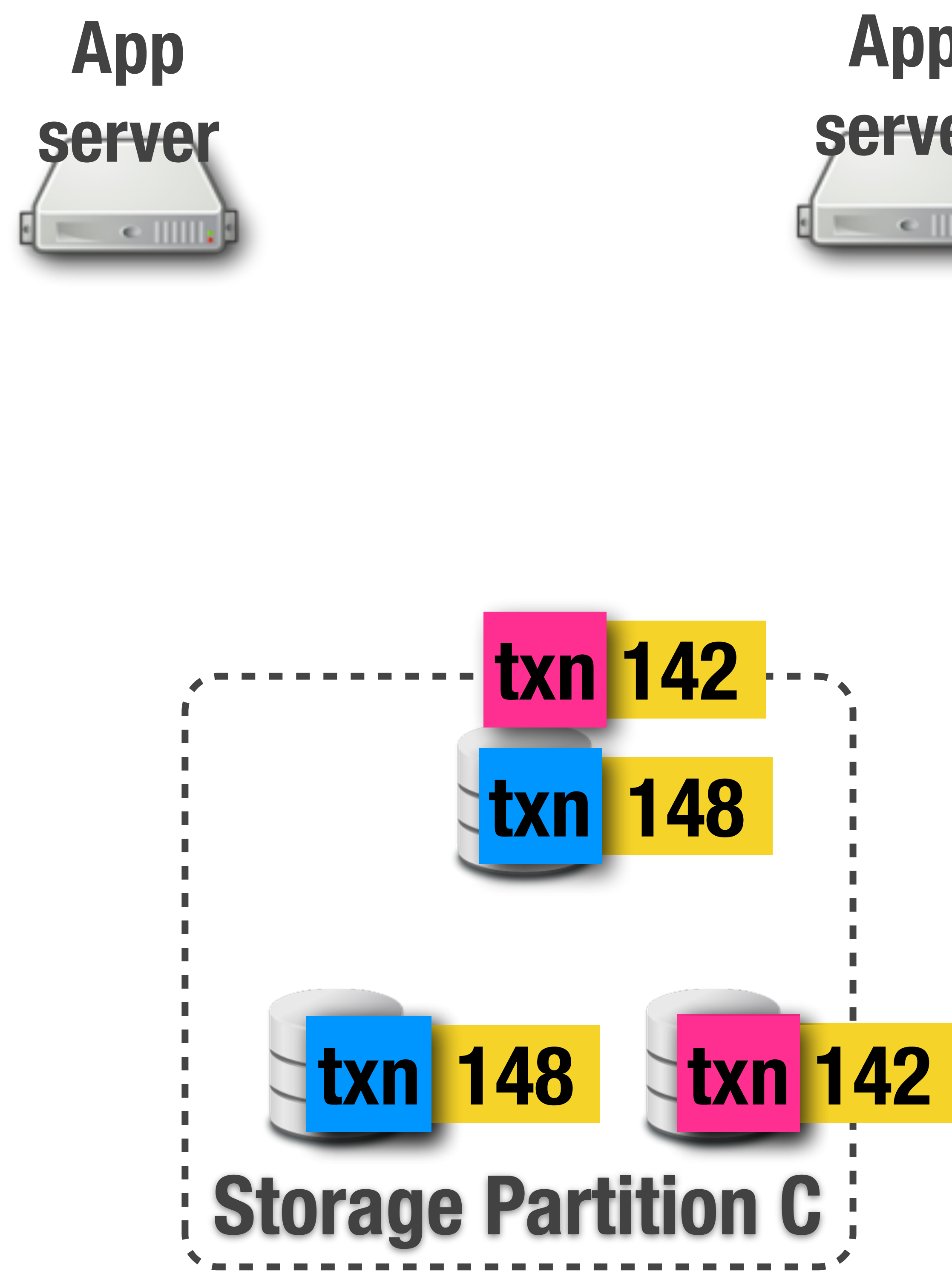
Handling Inconsistency

TAPIR uses several techniques to cope with inconsistency across replicas:

- **Loosely synchronized clocks** for optimistic transaction ordering at clients.
- **Optimistic concurrency control** to detect conflicts with a partial history.
- **Multi-versioned storage** for applying updates out-of-order.

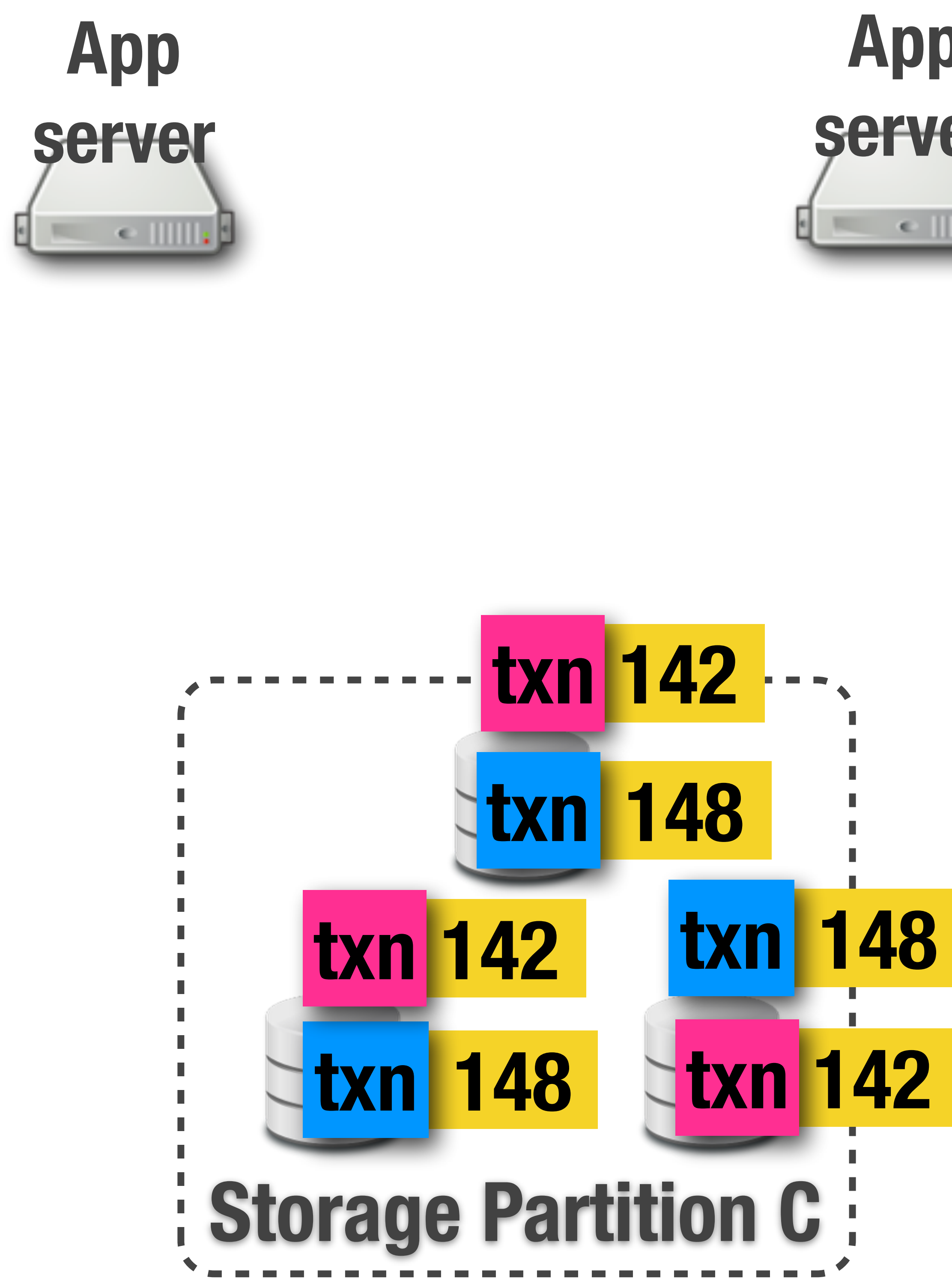
TAPIR Technique: Out-of-order updates with **multi-versioned storage**

- Backing store versioned using transaction timestamp.
- Replicas periodically synchronize to find missed transactions.
- Backing store converges to same state, regardless of when the updates are applied.



TAPIR Technique: Out-of-order updates with **multi-versioned storage**

- Backing store versioned using transaction timestamp.
- Replicas periodically synchronize to find missed transactions.
- Backing store converges to same state, regardless of when the updates are applied.



Rest of this talk

1. The cost of strong consistency
2. TAPIR - the Transactional Application Protocol for Inconsistent Replication
3. <u>Evaluation</u>
4. Summary

Experimental Questions

Does TAPIR improve **latency**?

- In a single cluster?
- Across datacenters?

Does TAPIR improve **throughput**?

- For low contention workloads?
- For high contention workload?

Deployment

Cluster

- Servers connected via 12 switch fat-tree topology
- Average clock skew: ~6us
- Average RTT: ~150us

Wide-area

- Google Compute Engine VMs in Asia, Europe and US
- Average clock skew: ~2ms
- Average RTT: (Eu-A)~260 (Eu-US)~110 (US-As)~166

Workload

Microbenchmark

- Single key read-modify-write transaction
- 1 shard, 3 replicas
- Uniform access distribution over 1 million keys

Retwis benchmark

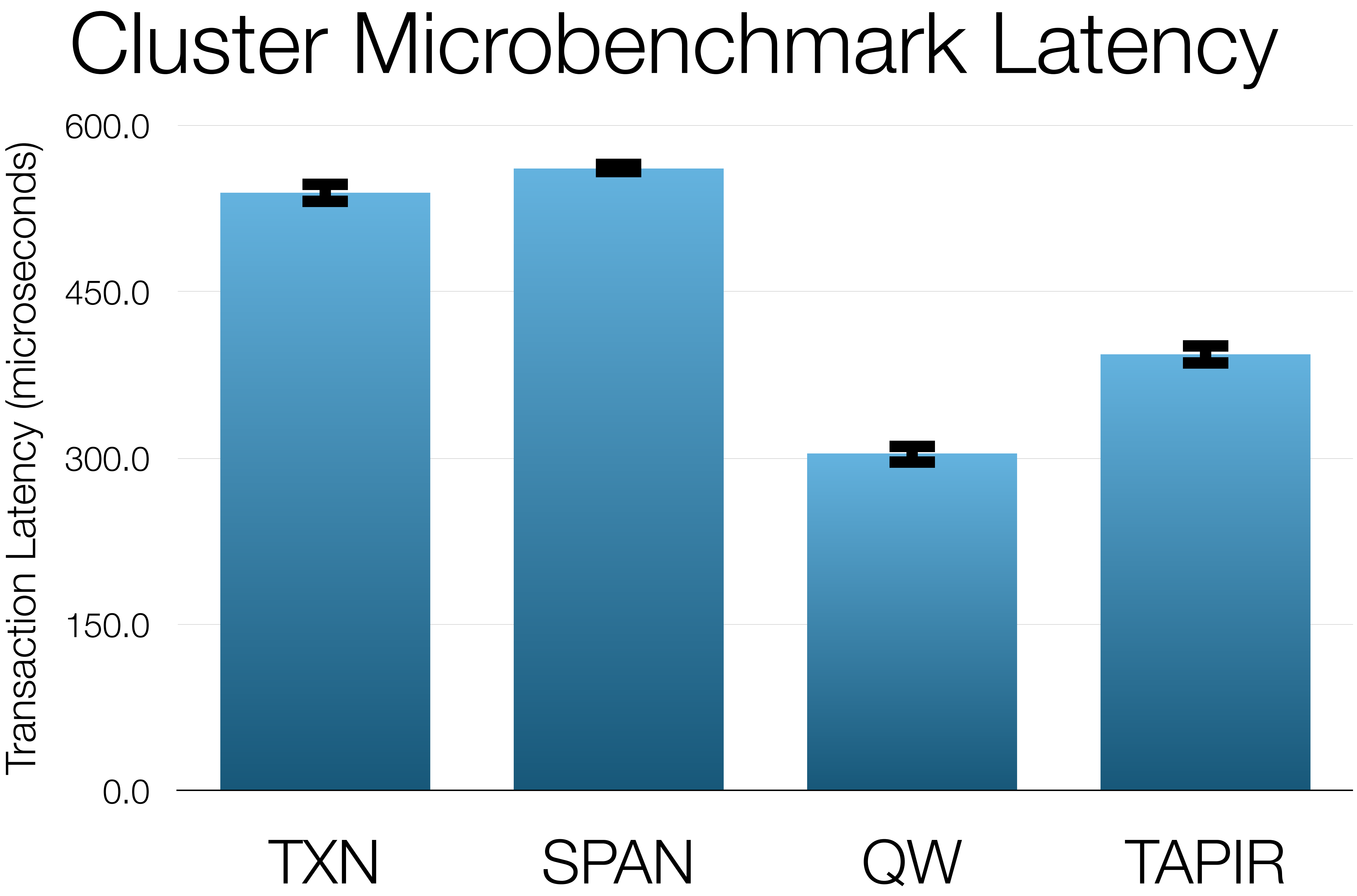
- Read-write transactions based on Retwis
- 5 shards, 3 replicas
- Zipf distribution (co-efficient=0.6) over 1 million keys

Systems

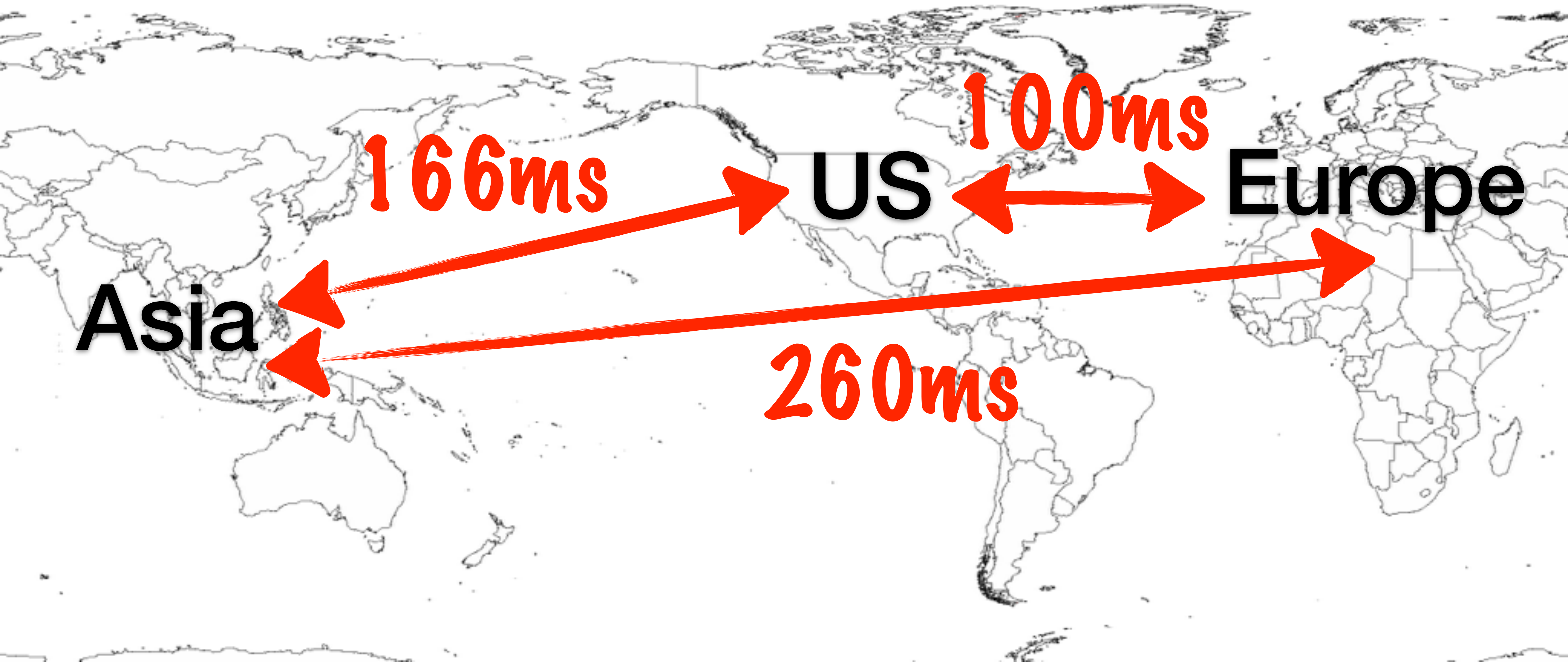
- **TAPIR:** Transactional storage with strong consistency with inconsistent replication
- **TXN:** Transactional storage with strong consistency
- **SPAN:** Spanner read-write protocol
- **QW:** Non-transactional storage with weak consistency with write everywhere, read anywhere policy

System Comparison

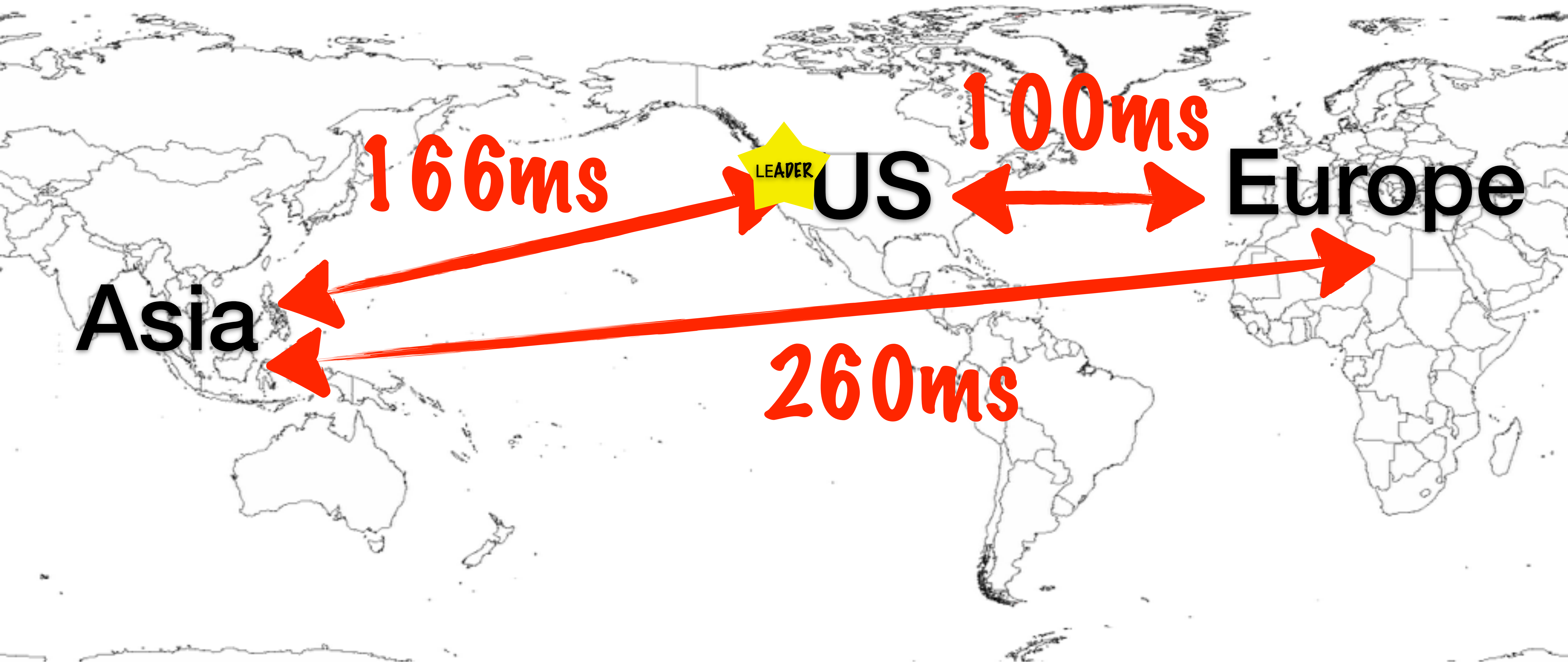
	Transaction Protocol	Replication Protocol	Concurrency Control
TAPIR	2PC	Inconsistent Replication	OCC
TXN	2PC	Paxos	OCC
SPAN	2PC	Paxos	Strict 2-Phase Locking
QW	None	Write everywhere, Read anywhere	None


Cluster Microbenchmark Latency
Transaction Latency (microseconds)
600.0
450.0
300.0
150.0
0.0
TXN
SPAN
QW
TAPIR

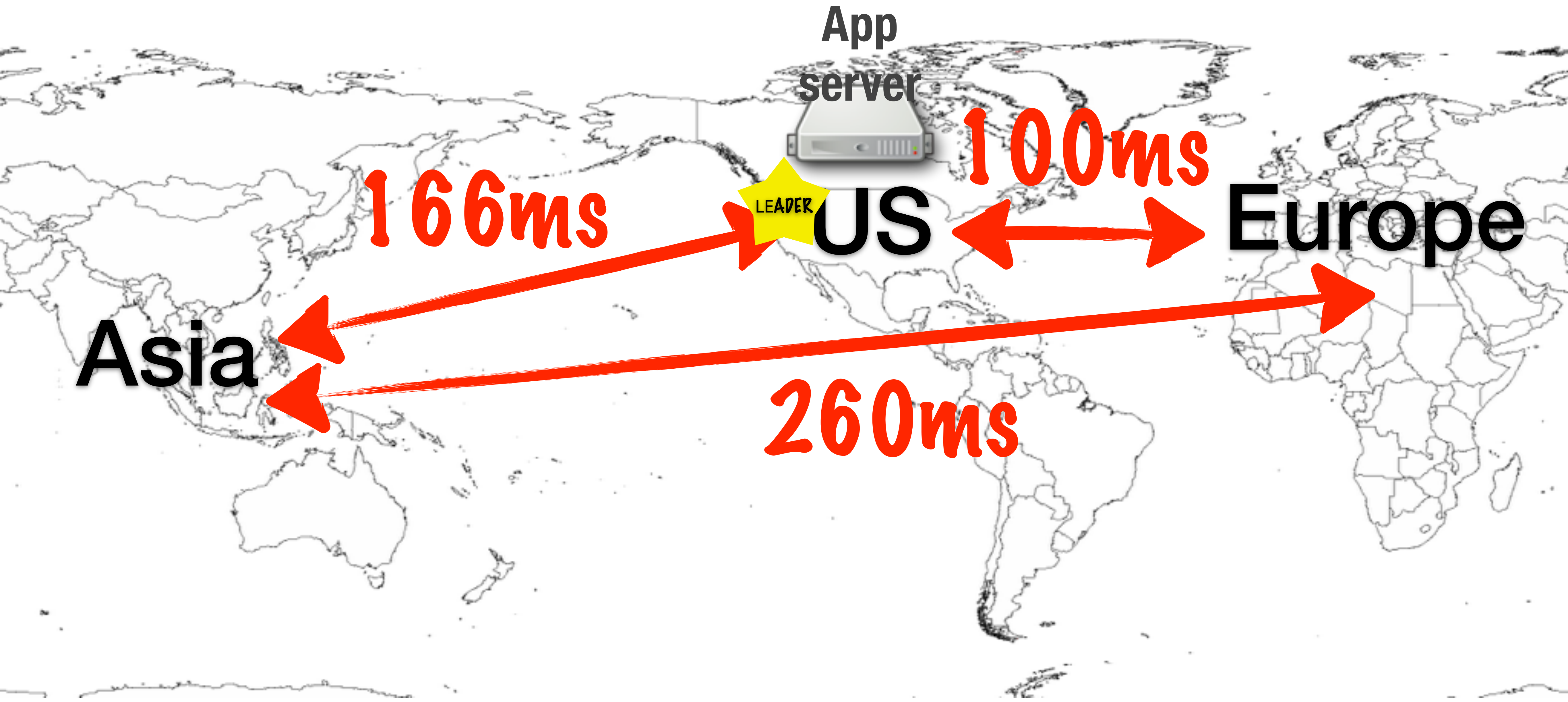
Wide-area Deployment



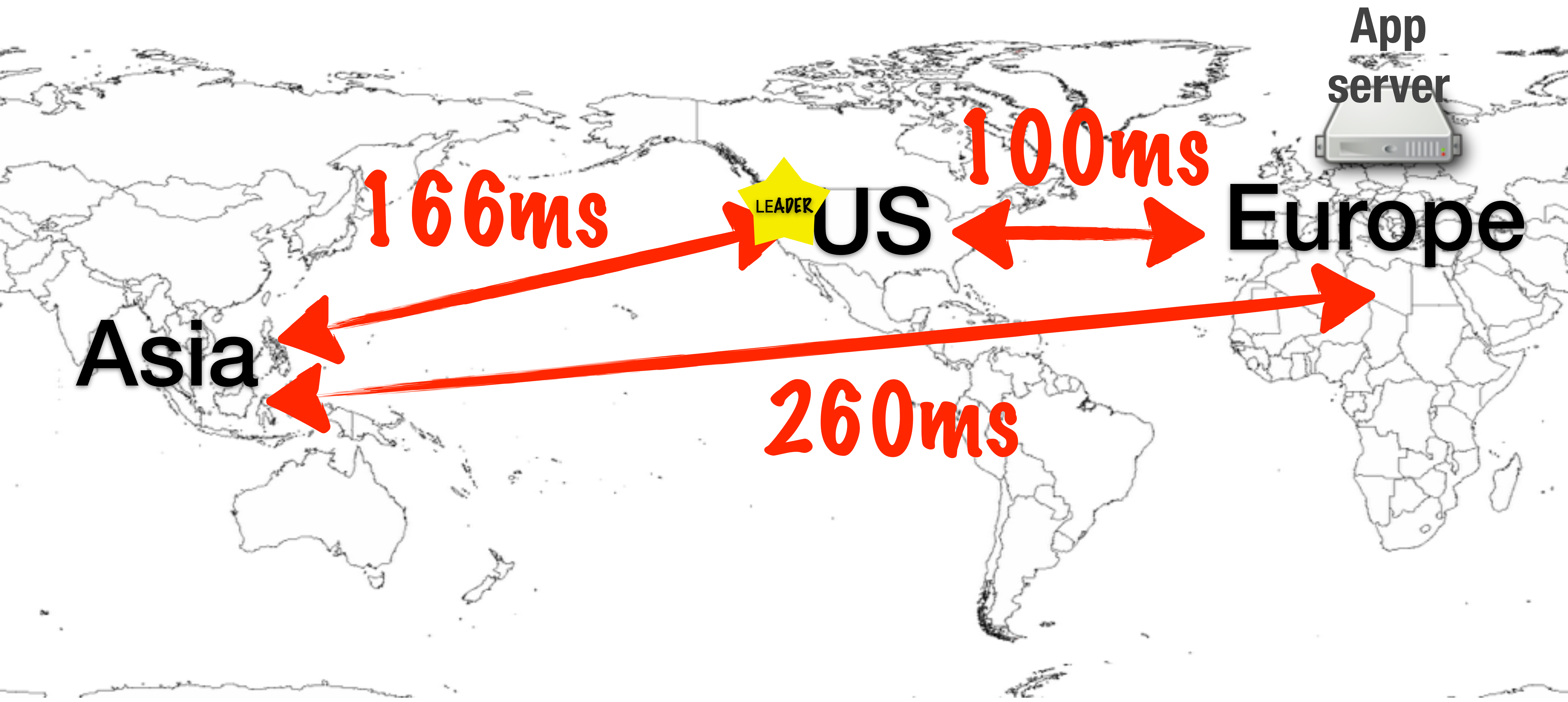
Wide-area Deployment



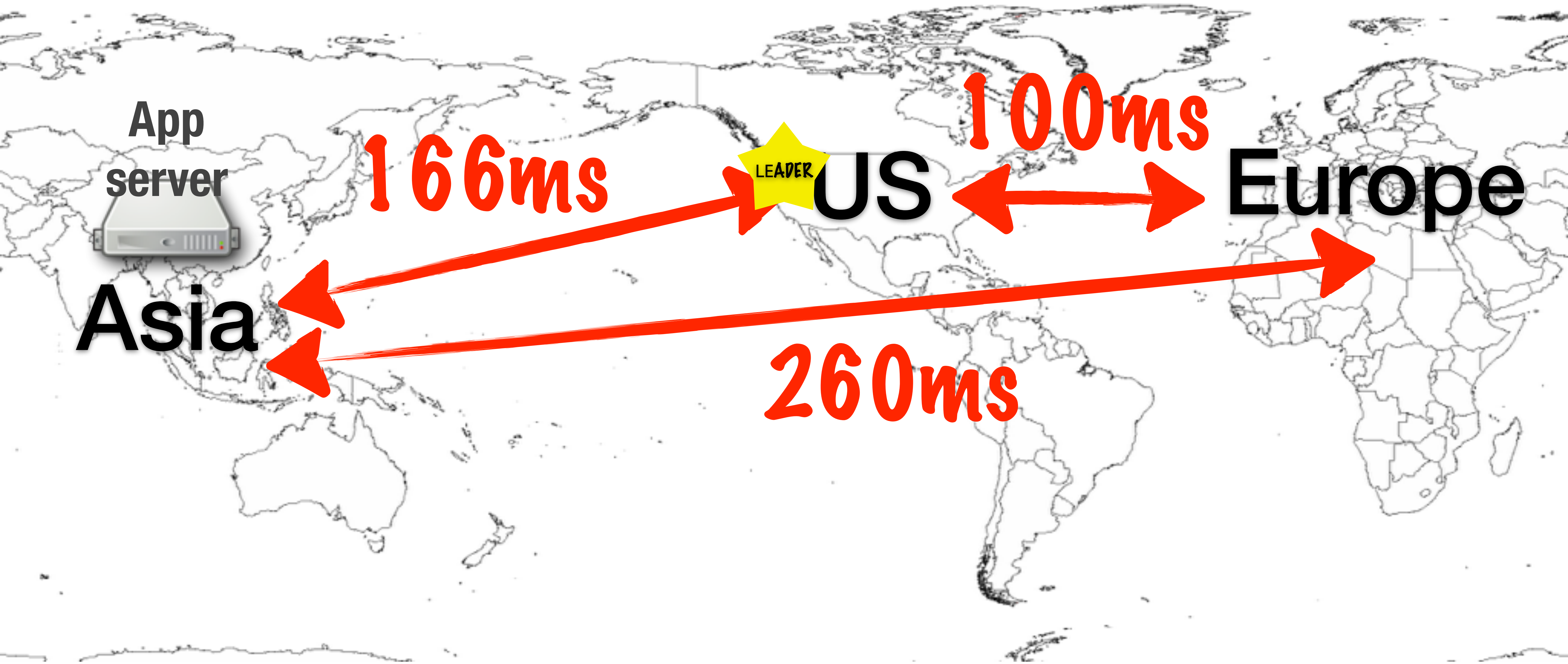
Wide-area Deployment



Wide-area Deployment

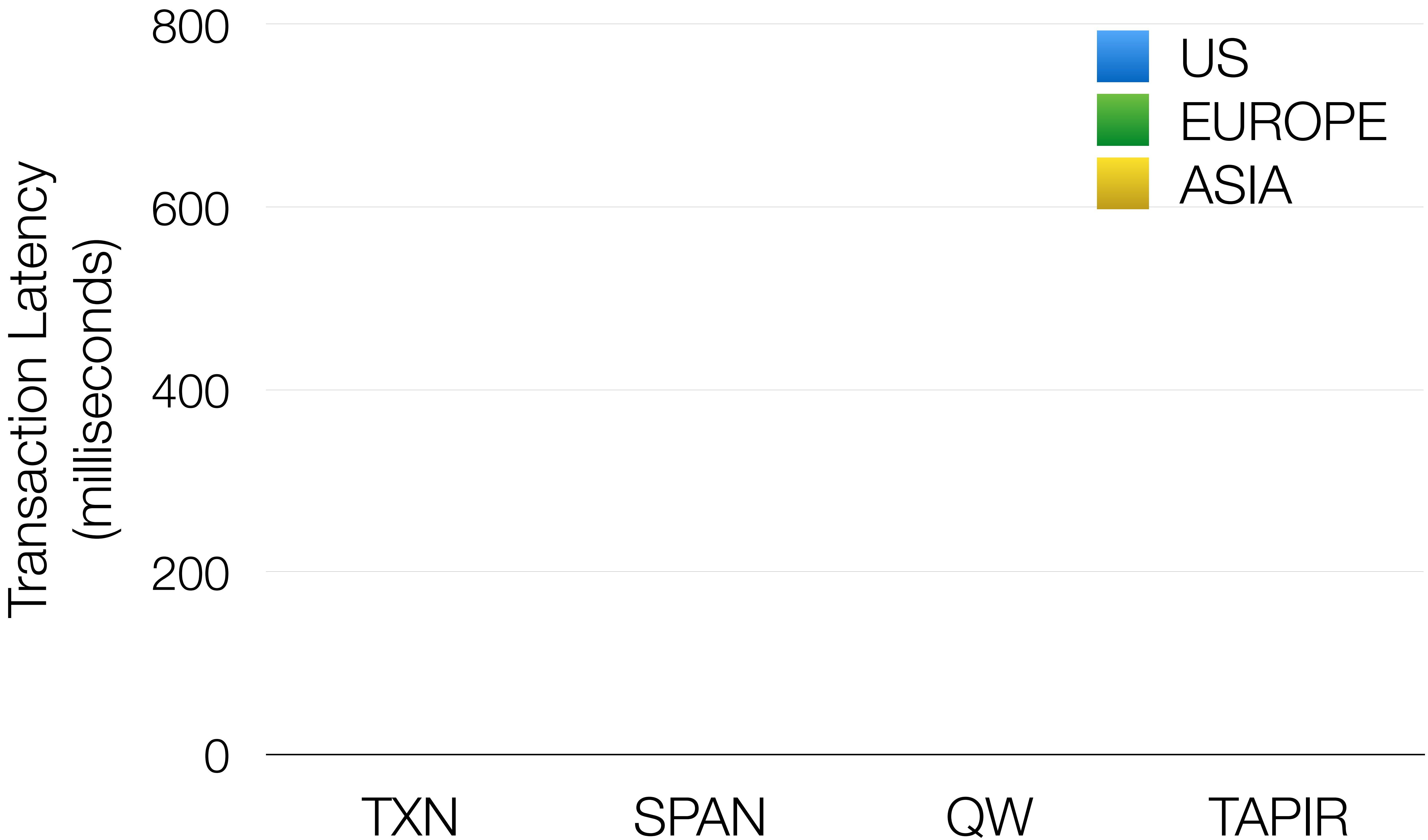


Wide-area Deployment

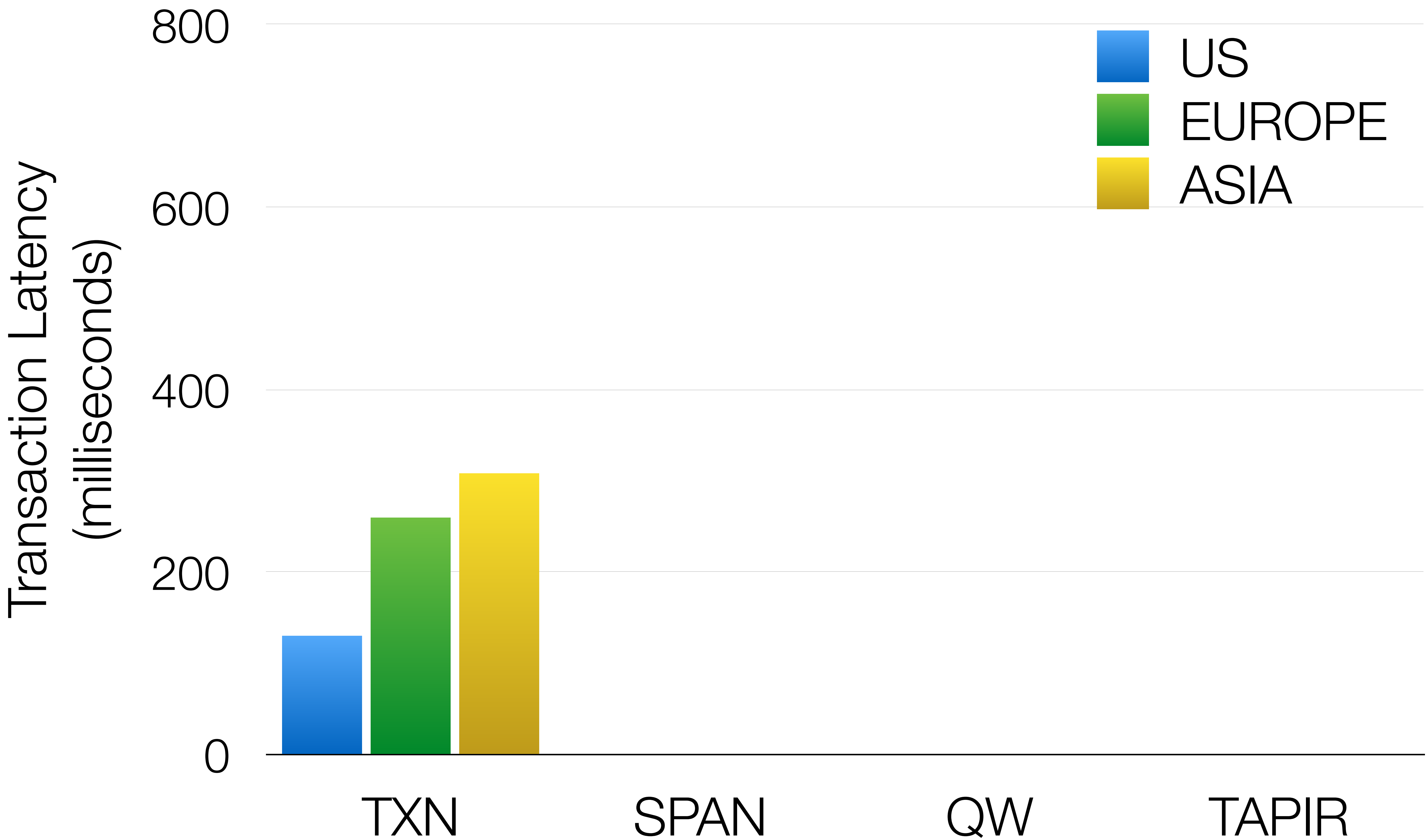


Wide-area Retwis Latency

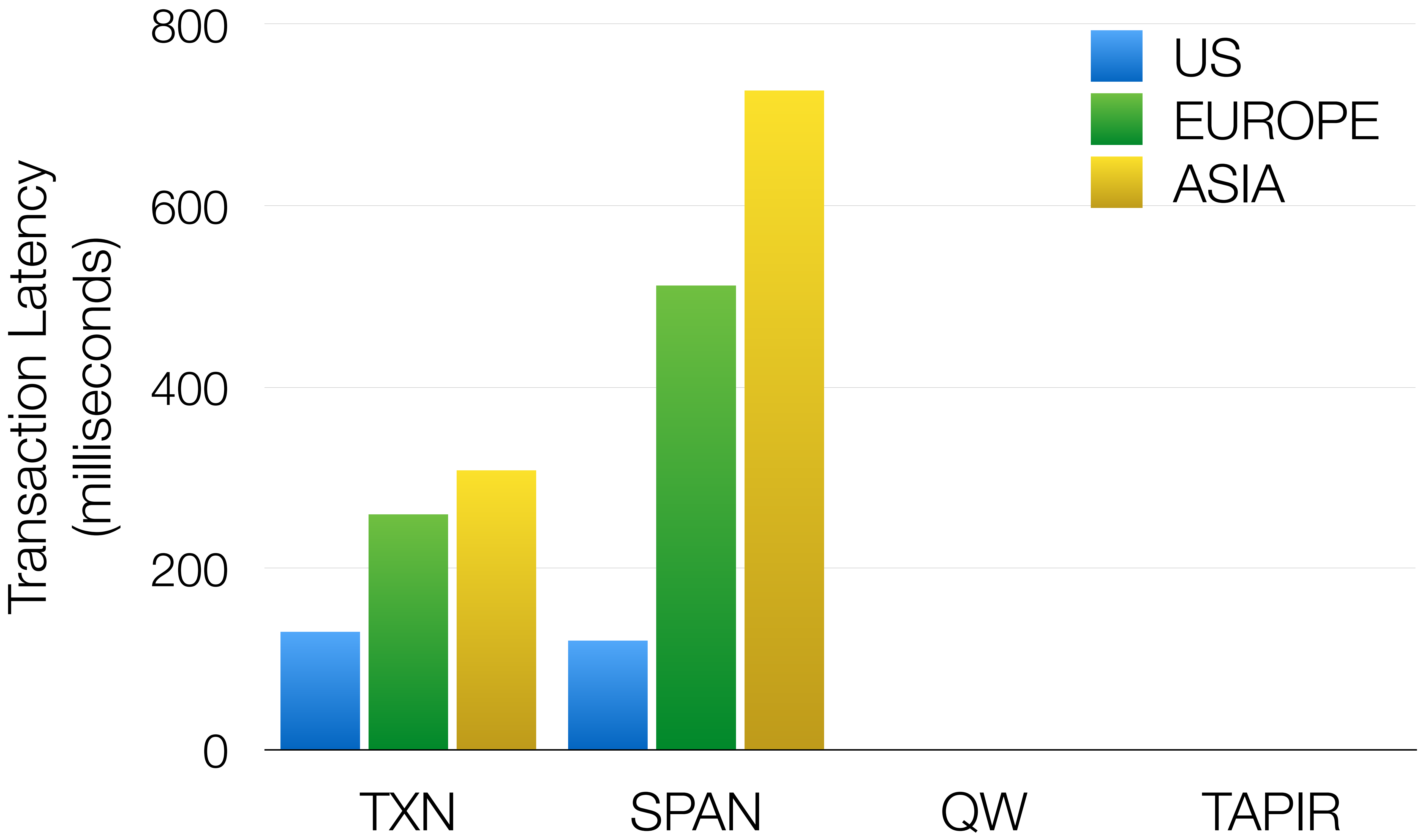
Wide-area Retwis Latency



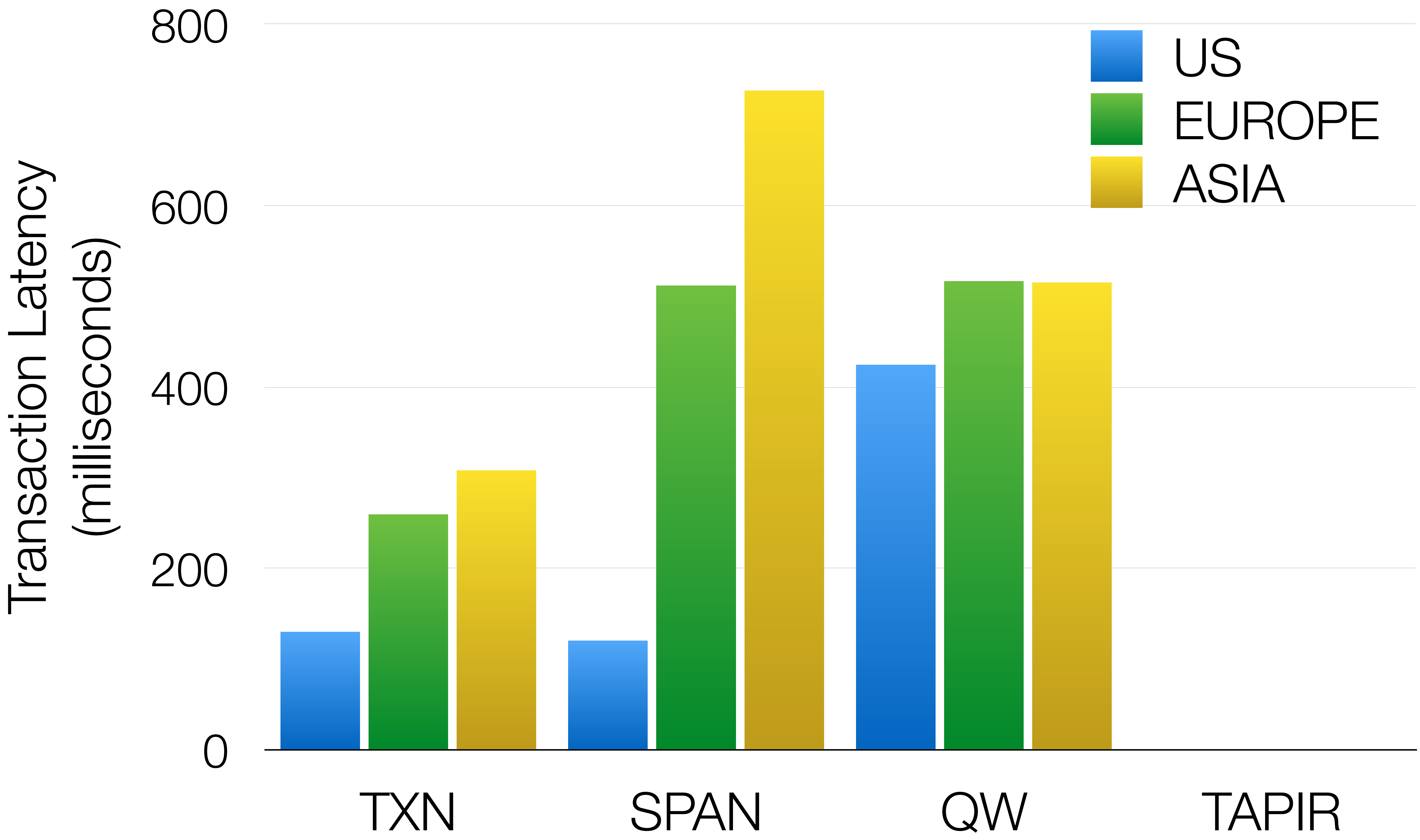
Wide-area Retwis Latency



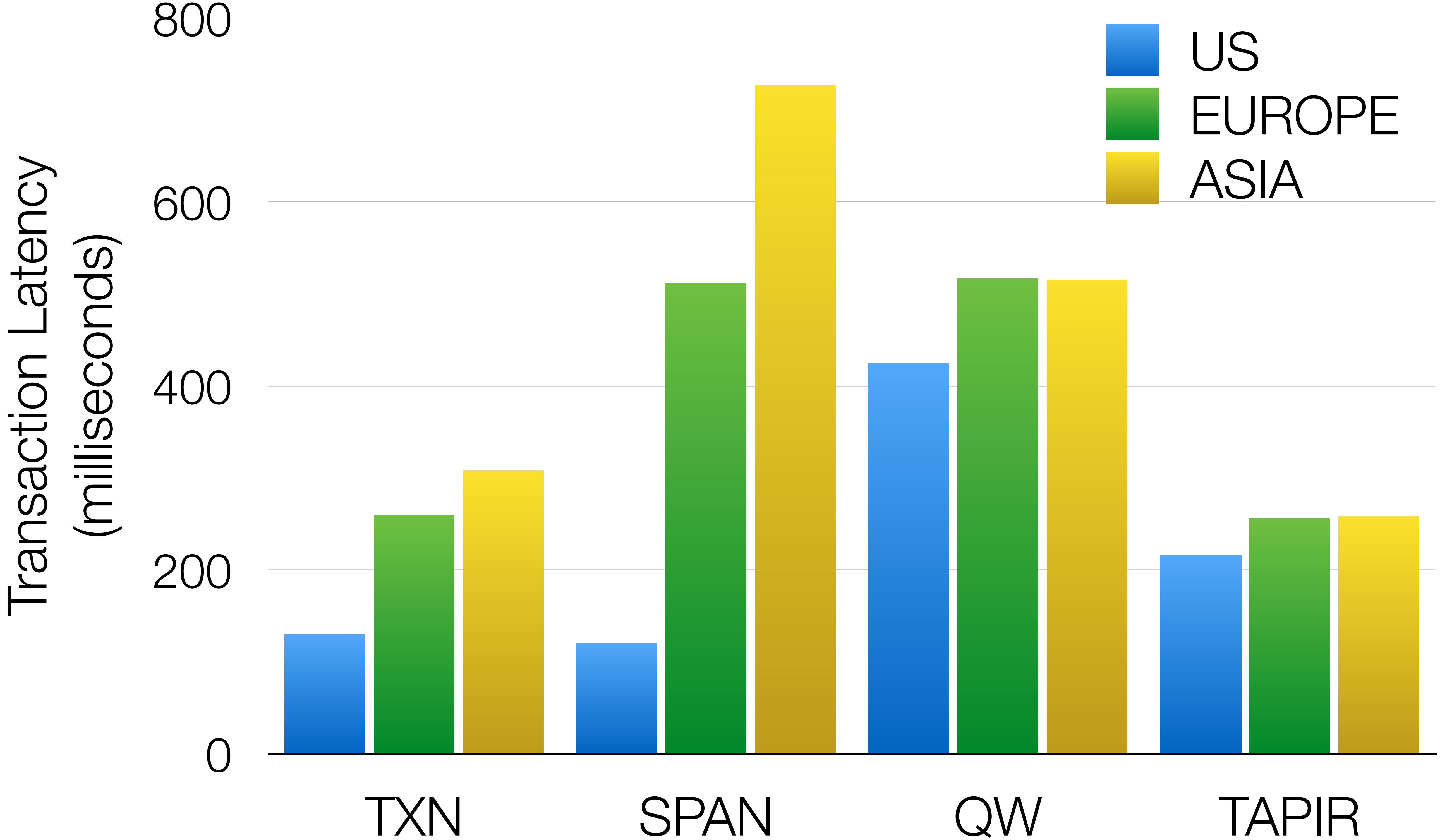
Wide-area Retwis Latency



Wide-area Retwis Latency

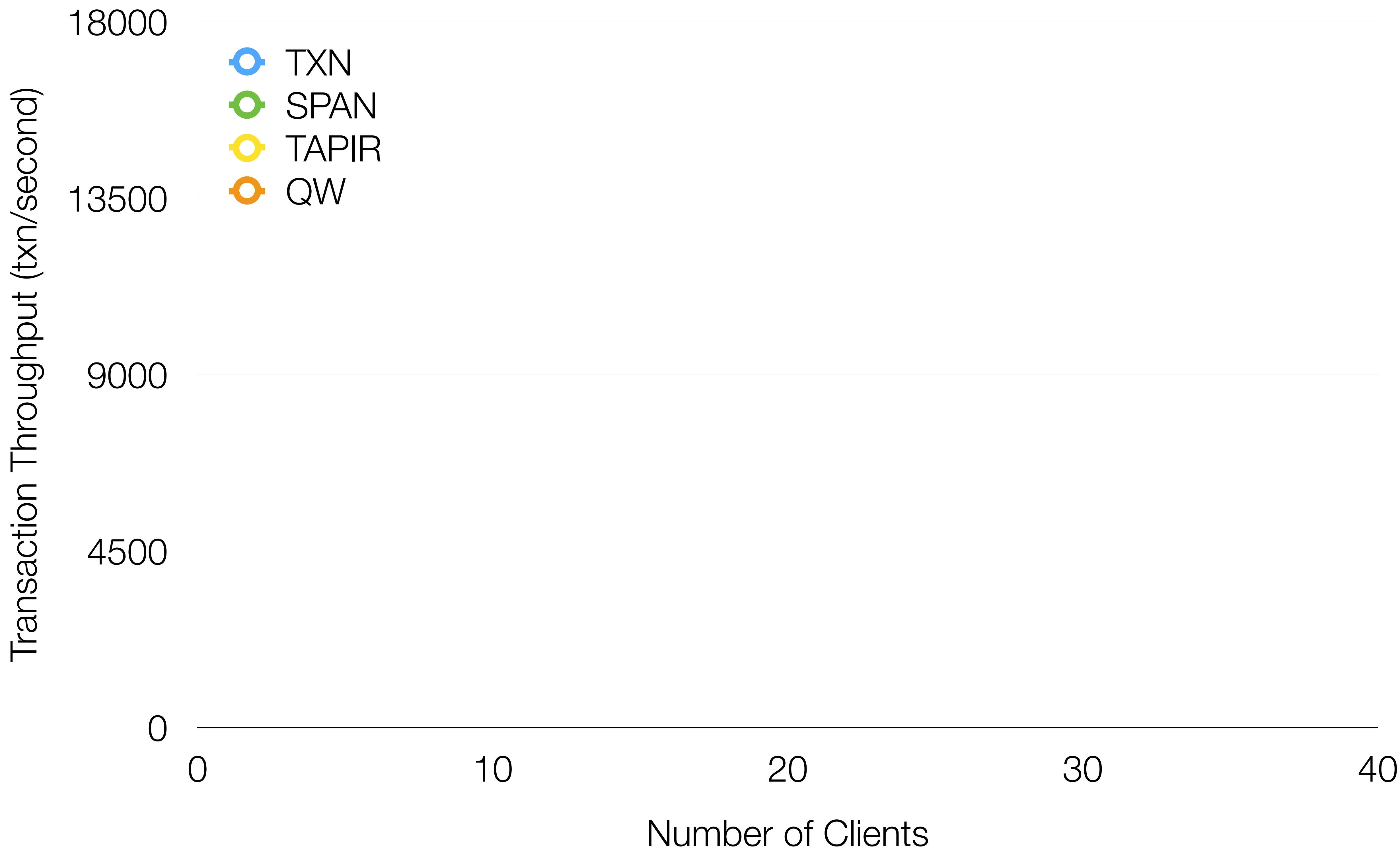


Wide-area Retwis Latency

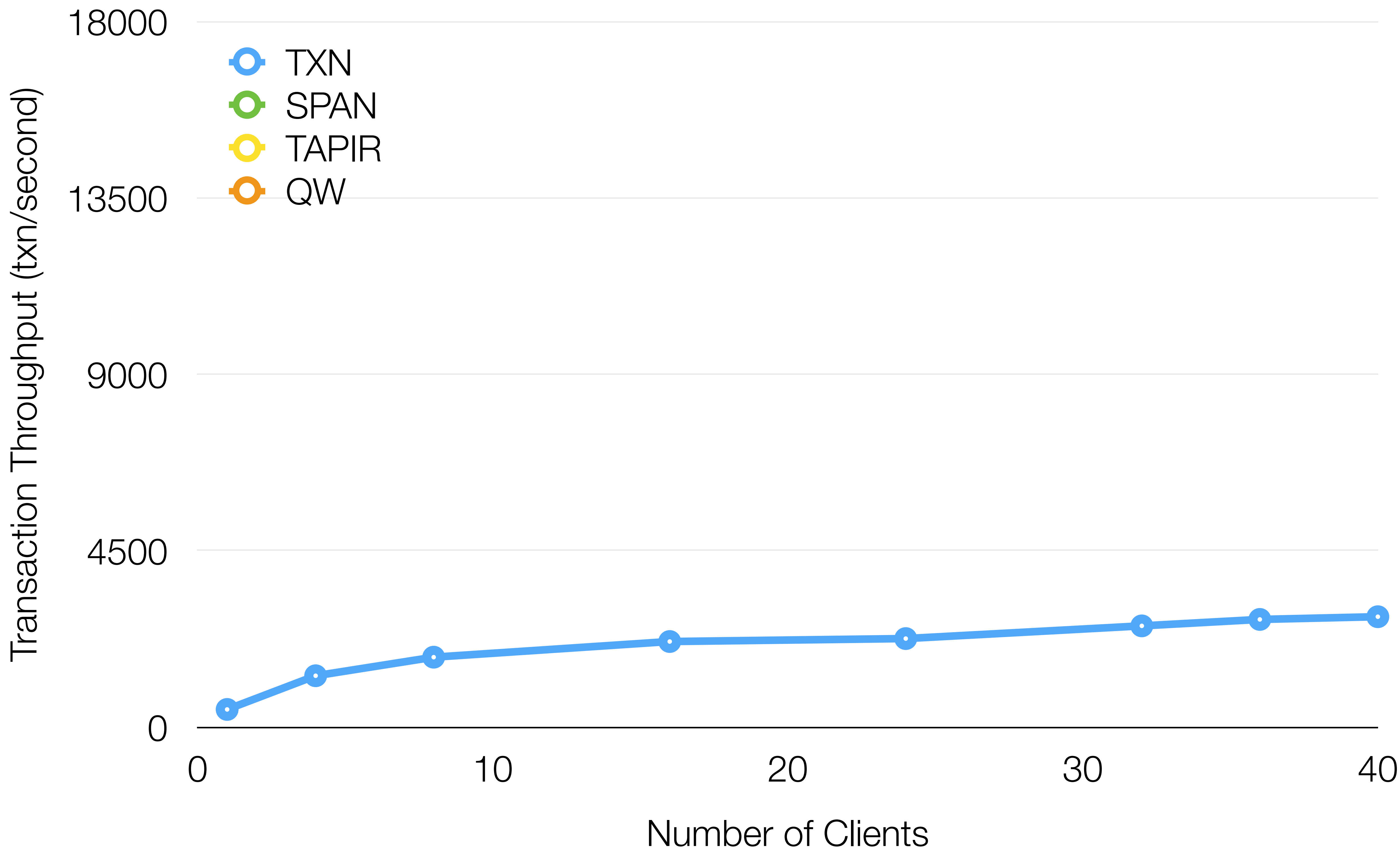


Microbenchmark Throughput

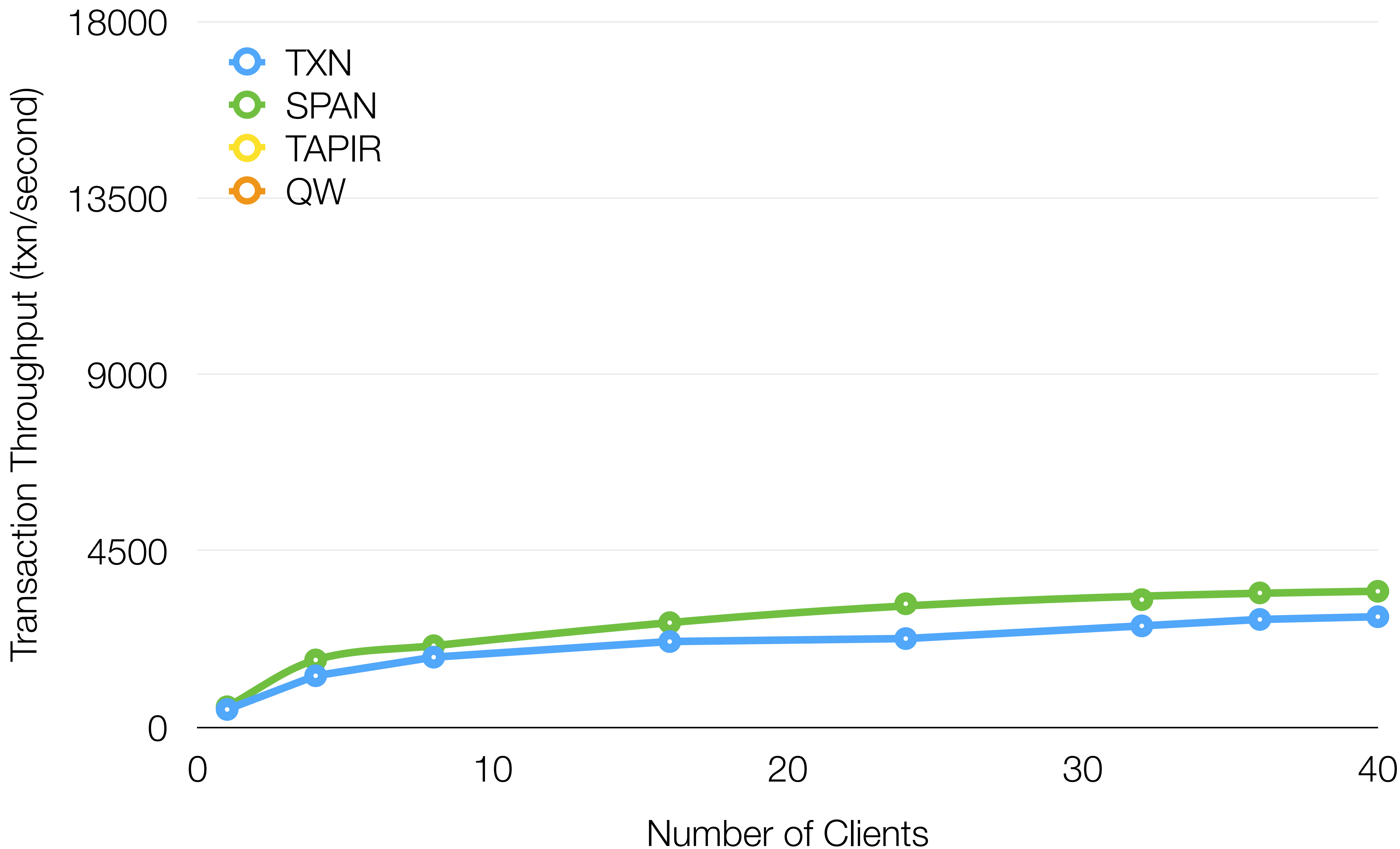
Microbenchmark Throughput



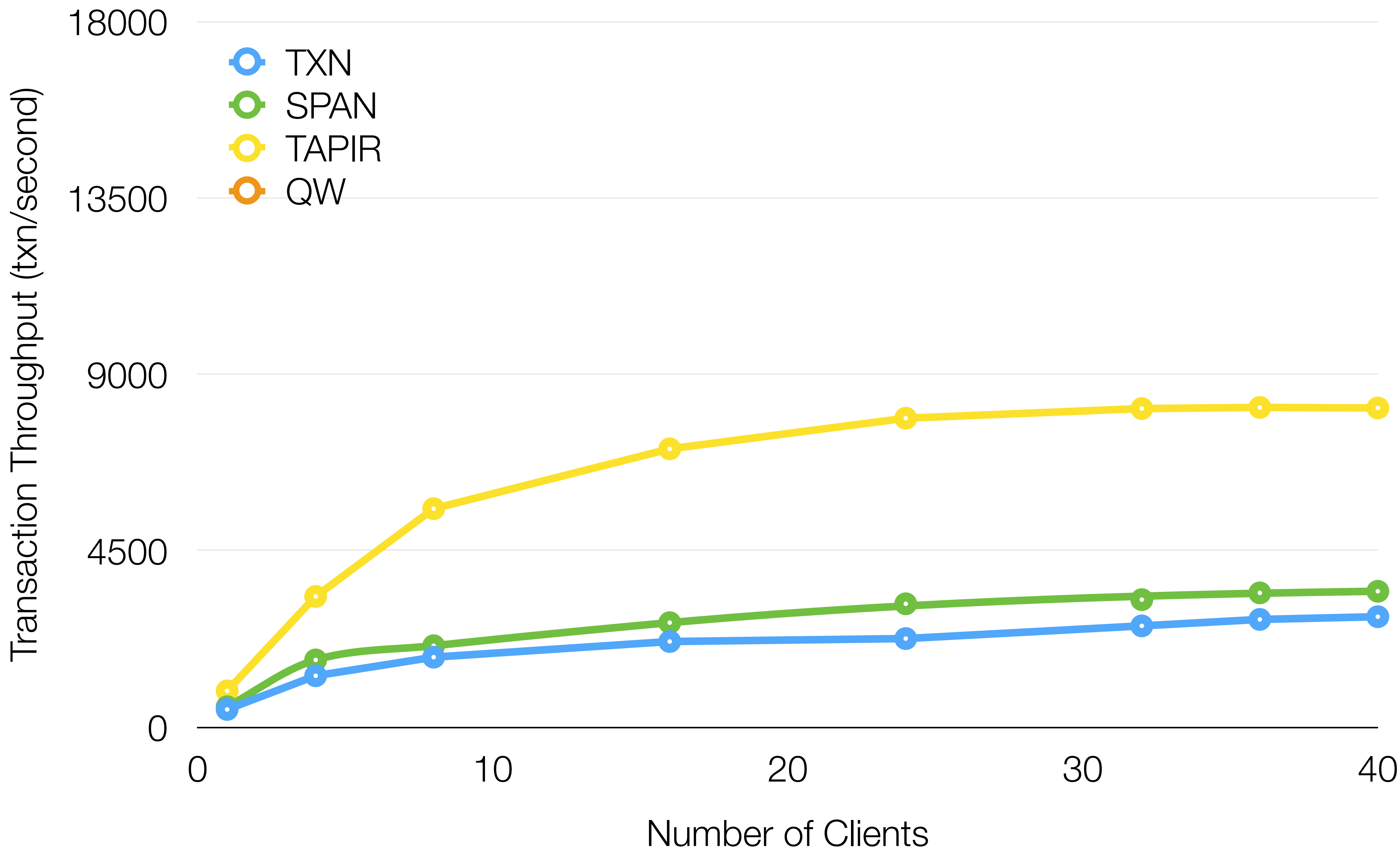
Microbenchmark Throughput



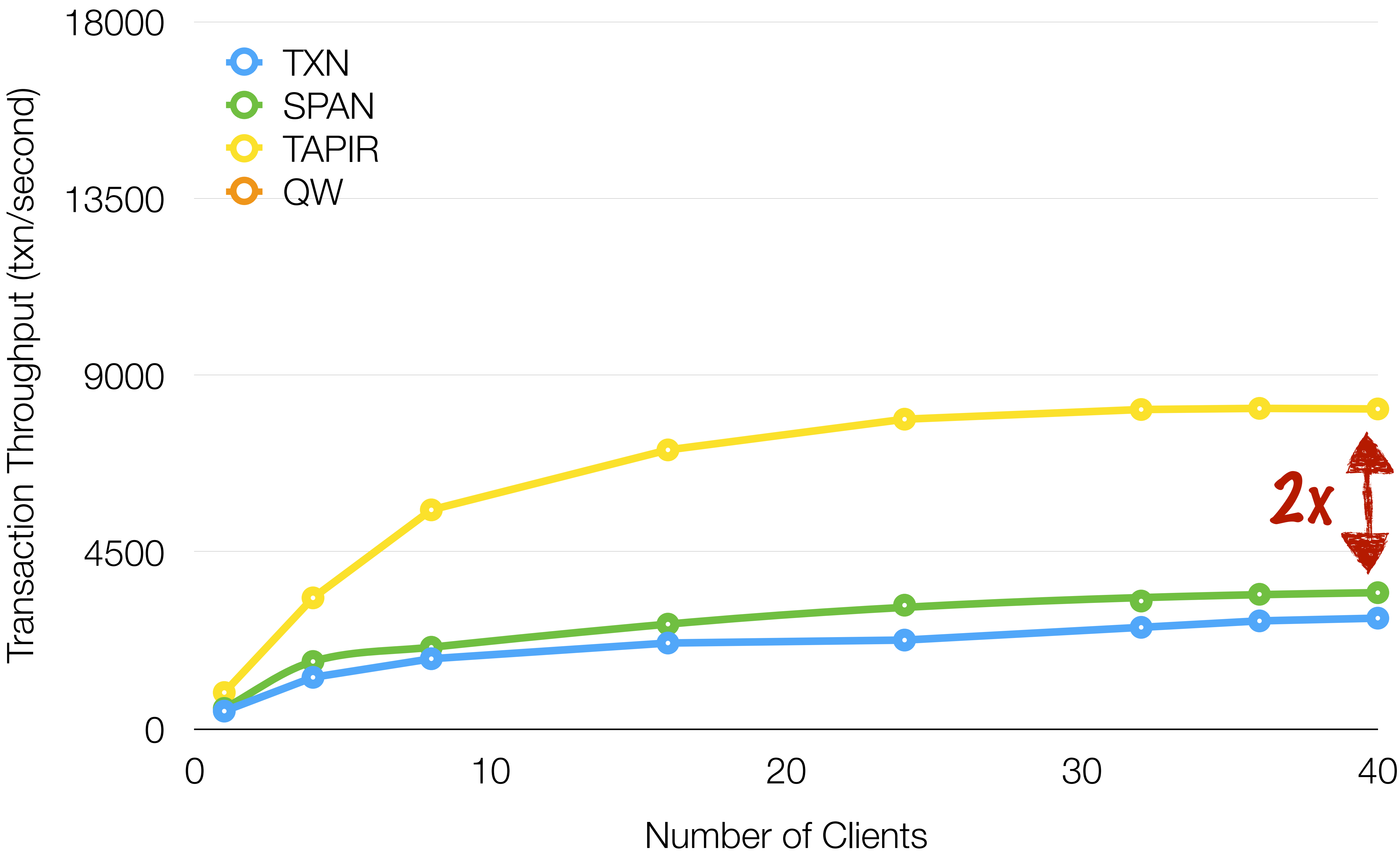
Microbenchmark Throughput



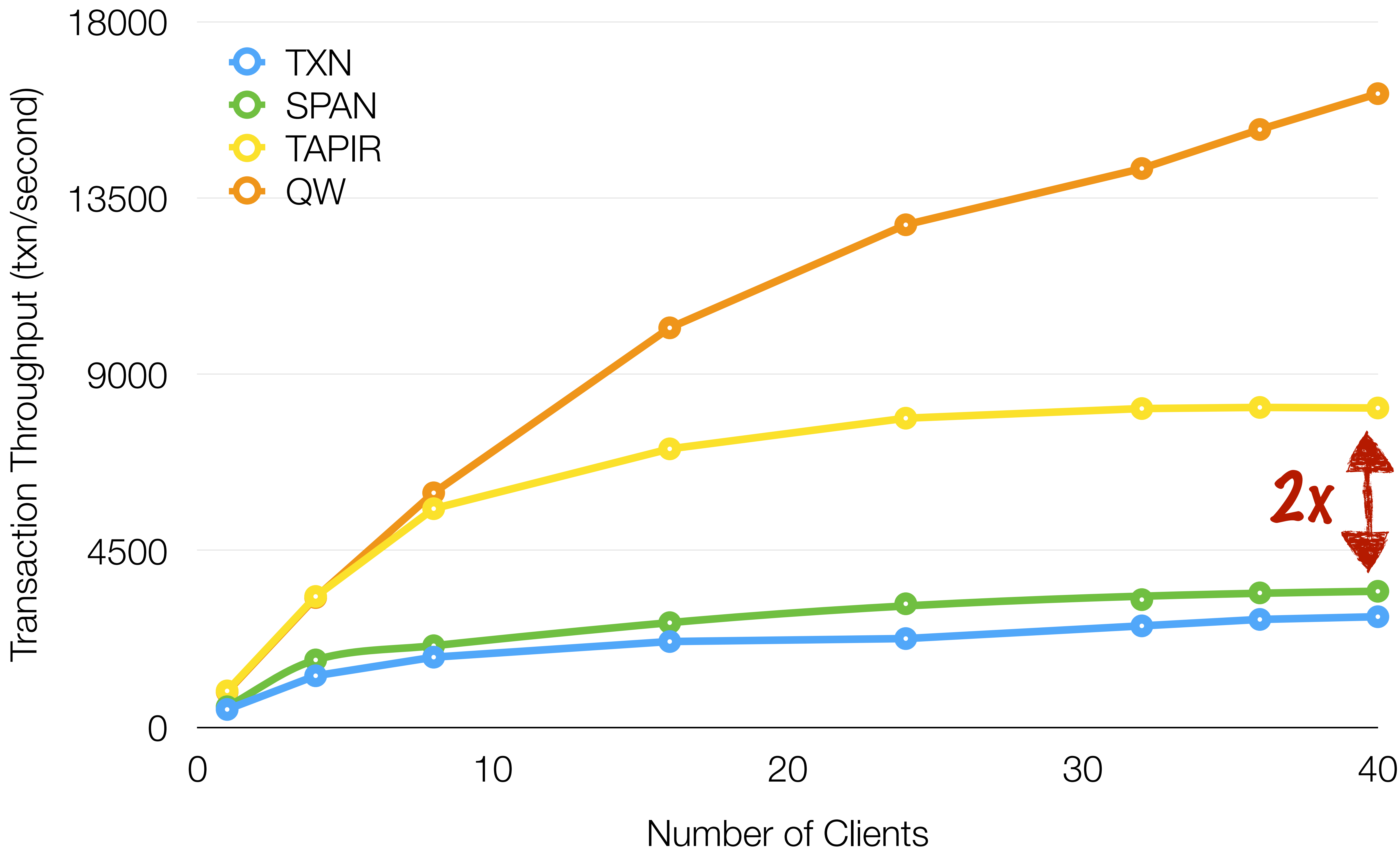
Microbenchmark Throughput



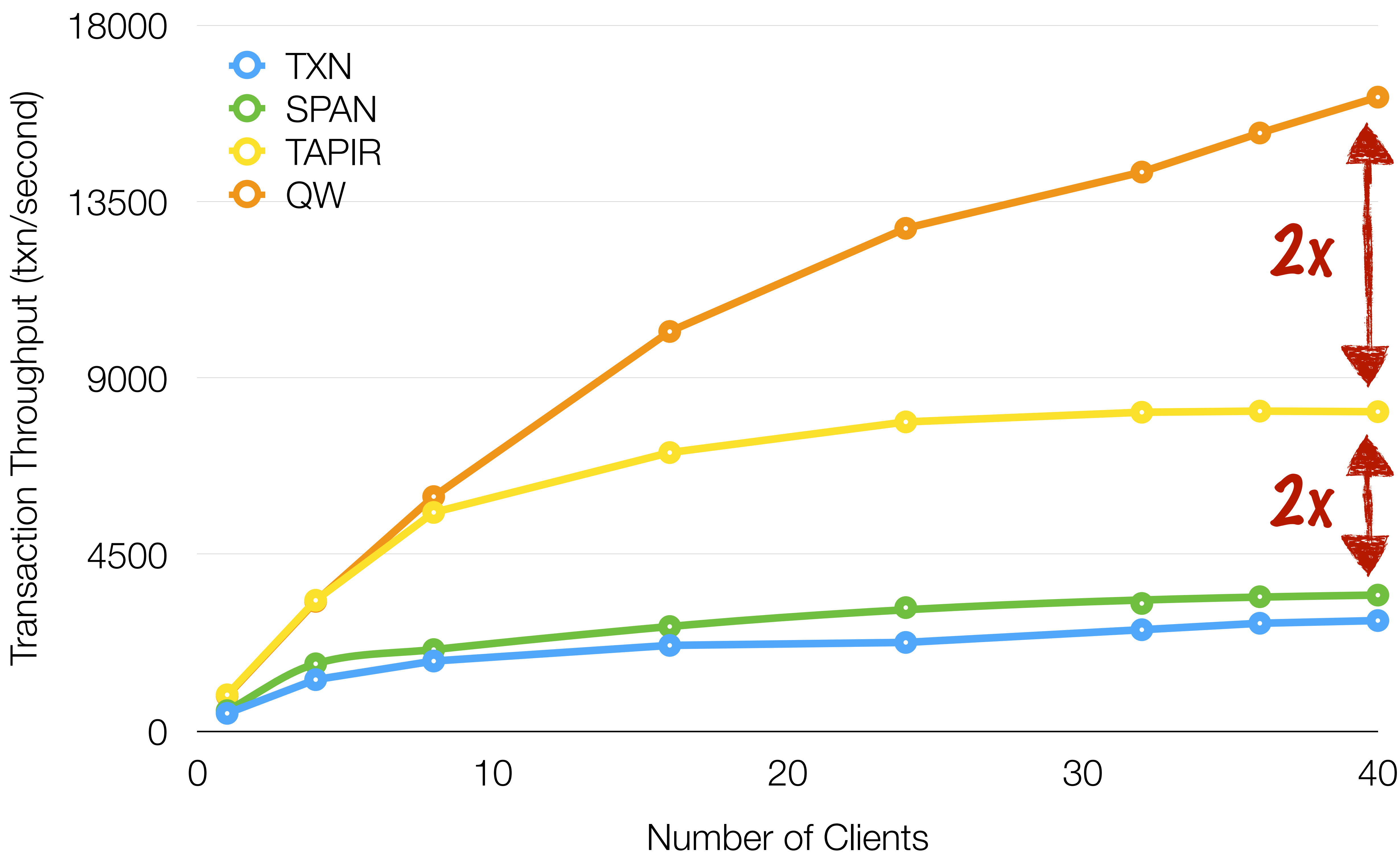
Microbenchmark Throughput



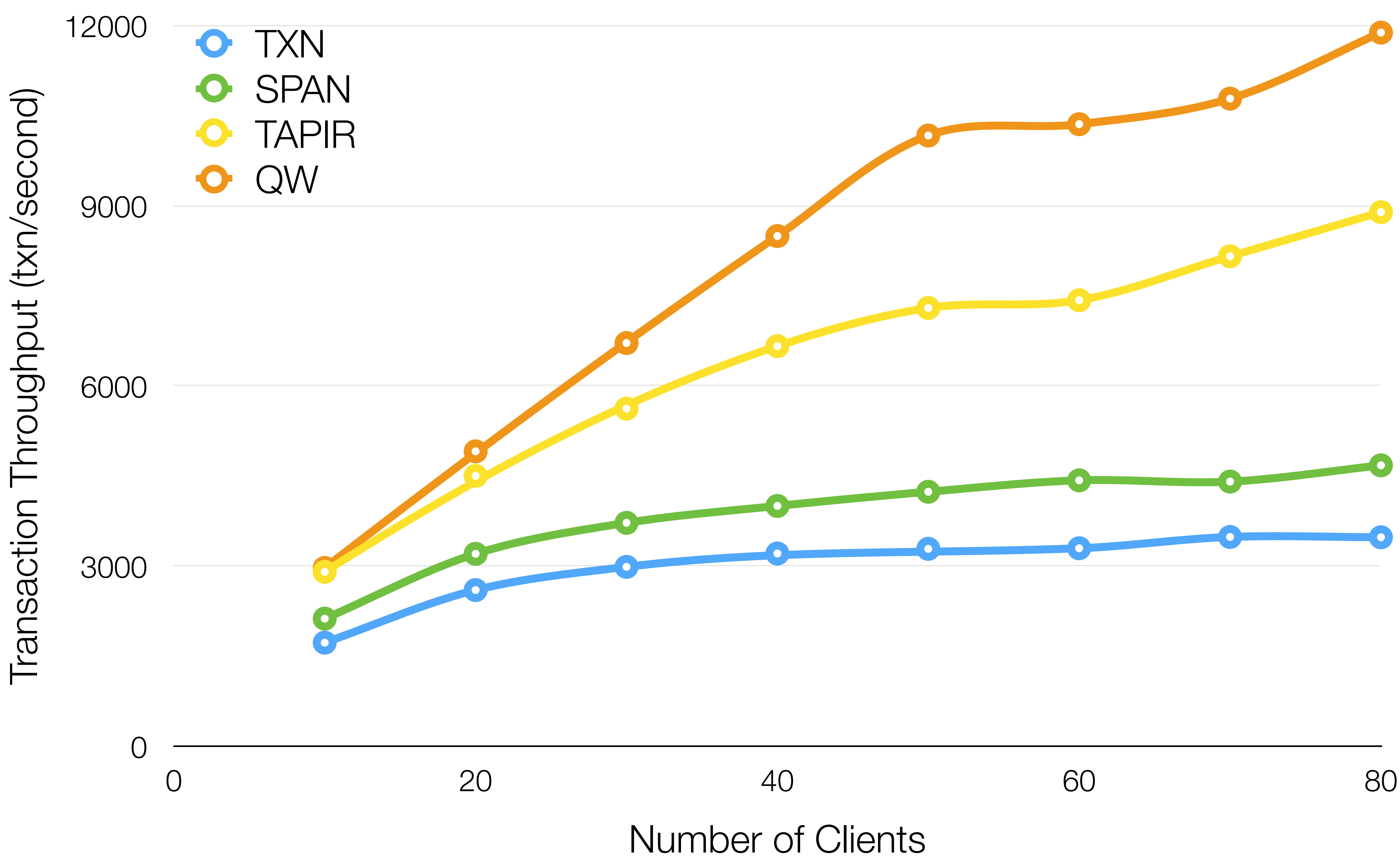
Microbenchmark Throughput



Microbenchmark Throughput



Retwis Throughput



Summary

- TAPIRs are surprisingly fast.
- Replication does not have to be consistent for transactions to be.
- Transactions do not have to be expensive.