Fast Transaction Commit in Sharded Data Stores*

Gregory Chockler

Royal Holloway, University of London

Alexey Gotsman

IMDEA Software Institute

*Submitted to DISC'18

ACM Symposium on Principles of Distributed Computing

July 23 – 27, 2018, Royal Holloway, University of London, Egham, United Kingdom

The ACM Symposium on Principles of Distributed Computing (PODC), is a premier international conference on the theory, design, analysis, implementation and application of distributed systems and networks. It will be held this year at Royal Holloway, University of London. In addition to three days of cutting-edge conference research paper presentations, it will feature two full days dedicated to the following workshop topics:

- Biological distributed algorithms
- Social issues in online social networks
- Theory and practice for integrated cloud, fog and edge computing paradigms
- Blockchain technology and theory
- Large-scale distributed systems and middleware
- Tools/languages/platforms for distributed system evaluation

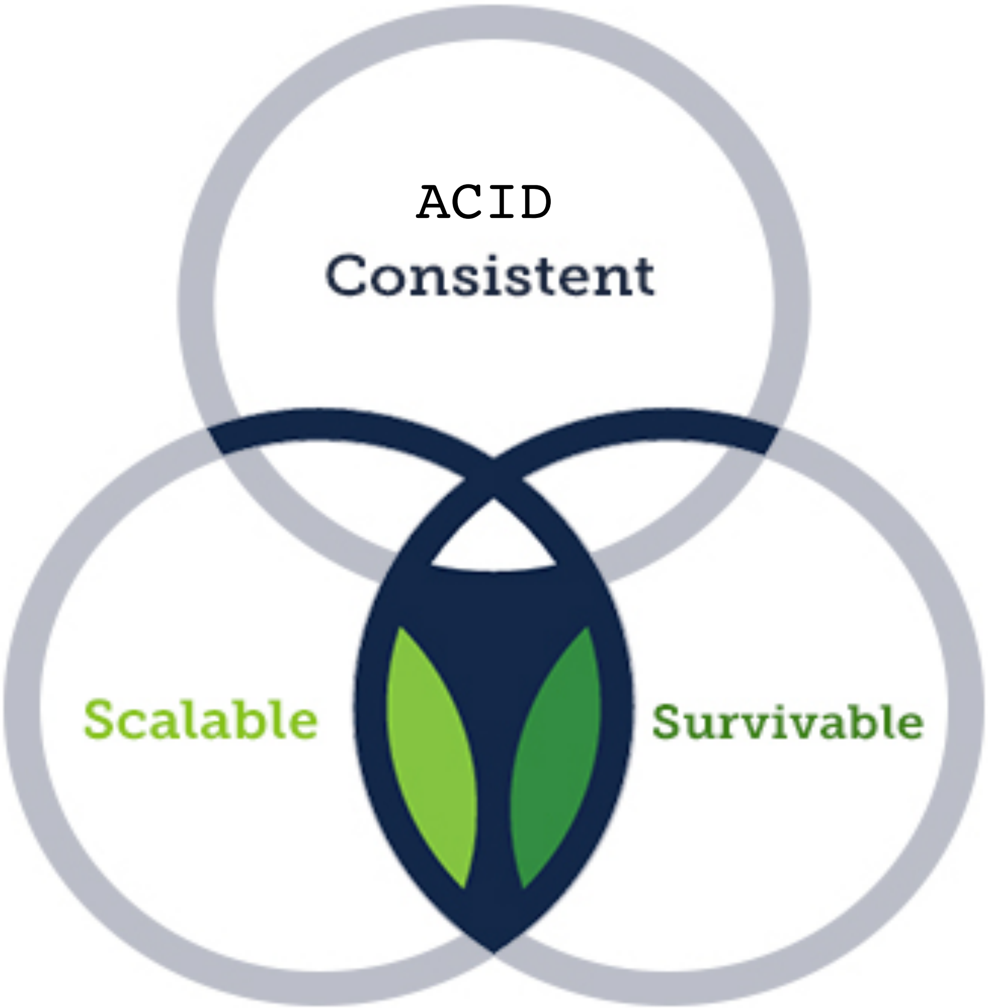
For more information and online registration: www.podc.org

Not Only SQL

No More

Not **O**nly SQL

Modern Data Stores



Modern Data Stores

Google Cloud Spanner

Clustrix

Cockroach DB

apache Ignite™

VOLTDB

Azure Cosmos DB

Blockchain

- Transactional semantics and fault-tolerance are non-negotiable

- Proof-of-Work (PoW): slow
  - Mining rate bounded by block propagation and validation delays

- Committee Consensus (PBFT)
  - Fast but bandwidth bound

Scaling Blockchains

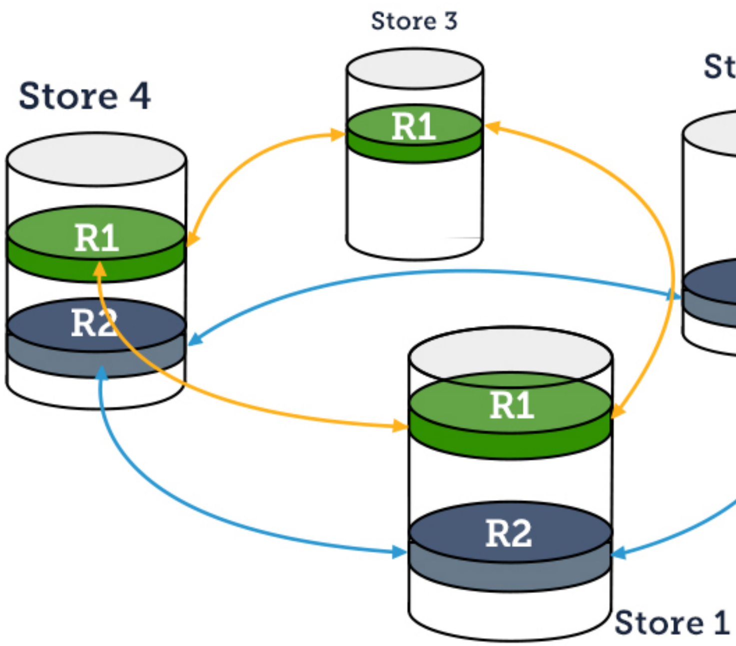
- Hybrid blockchains
  - PoW to elect and maintain a committee
  - Algorand, etc.

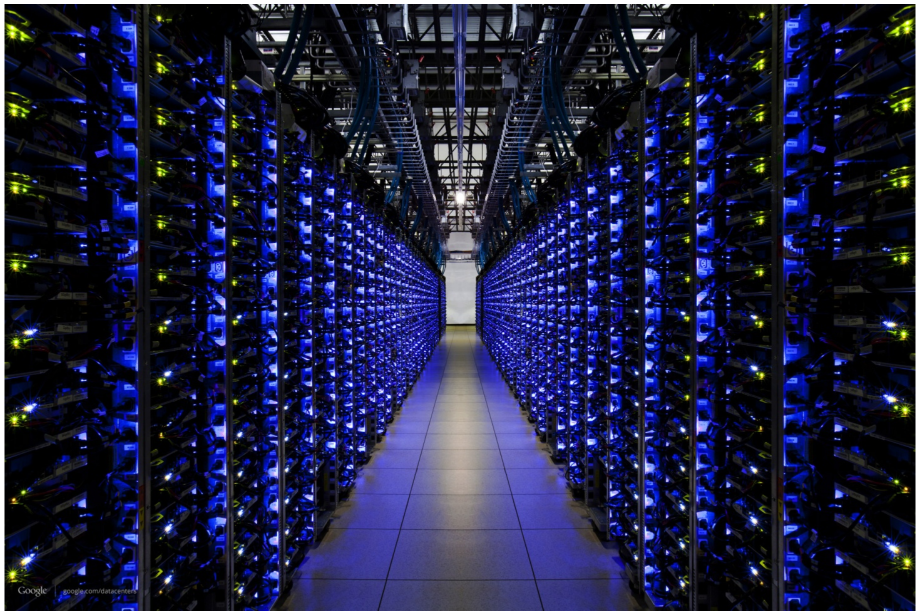
- Permissioned blockchains
  - Separate tx execution, ordering and state
  - Delegate ordering to the committee

- How to scale the committee?

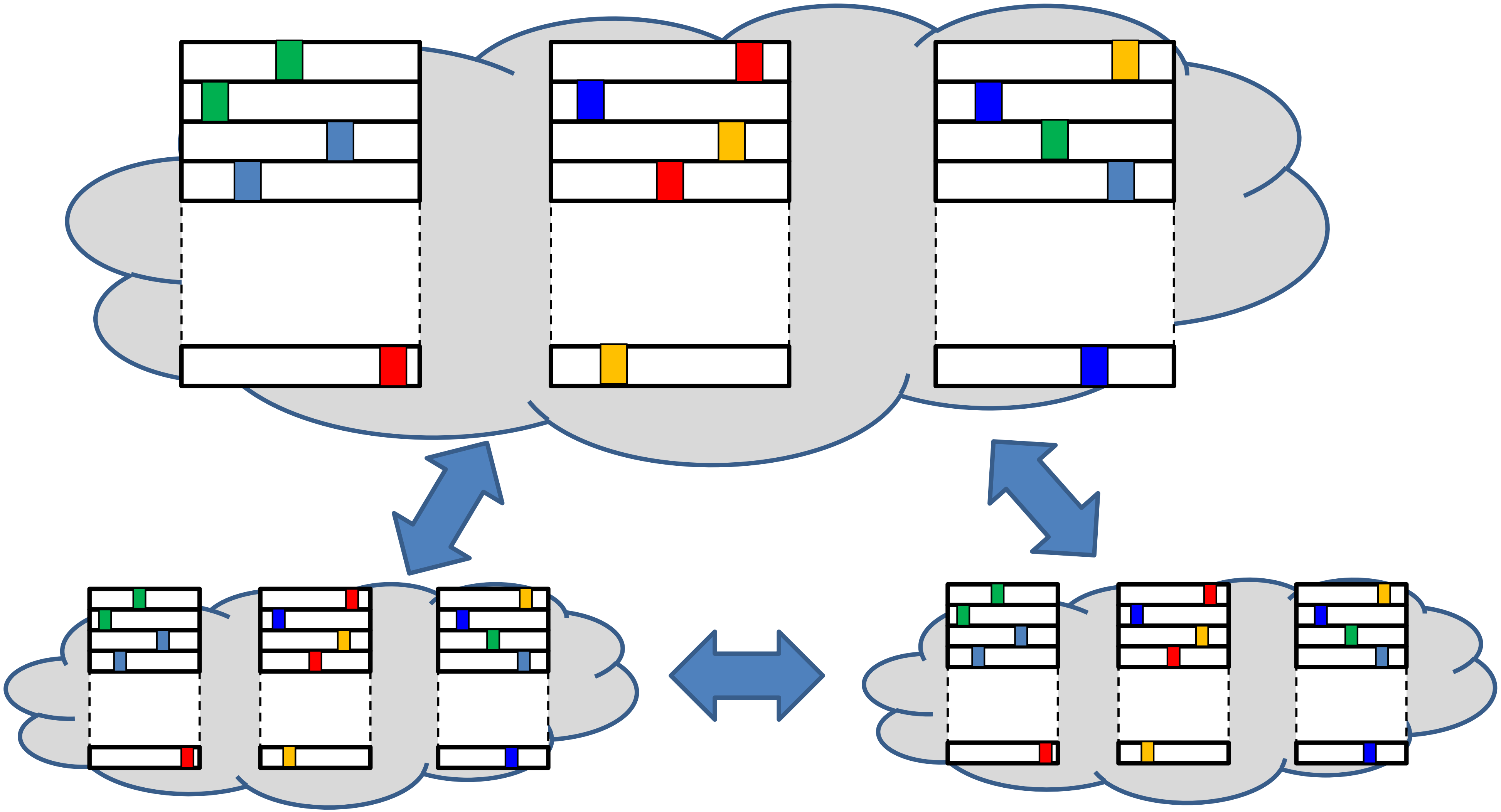
Horizontal Scaling

- Data sharding and replication

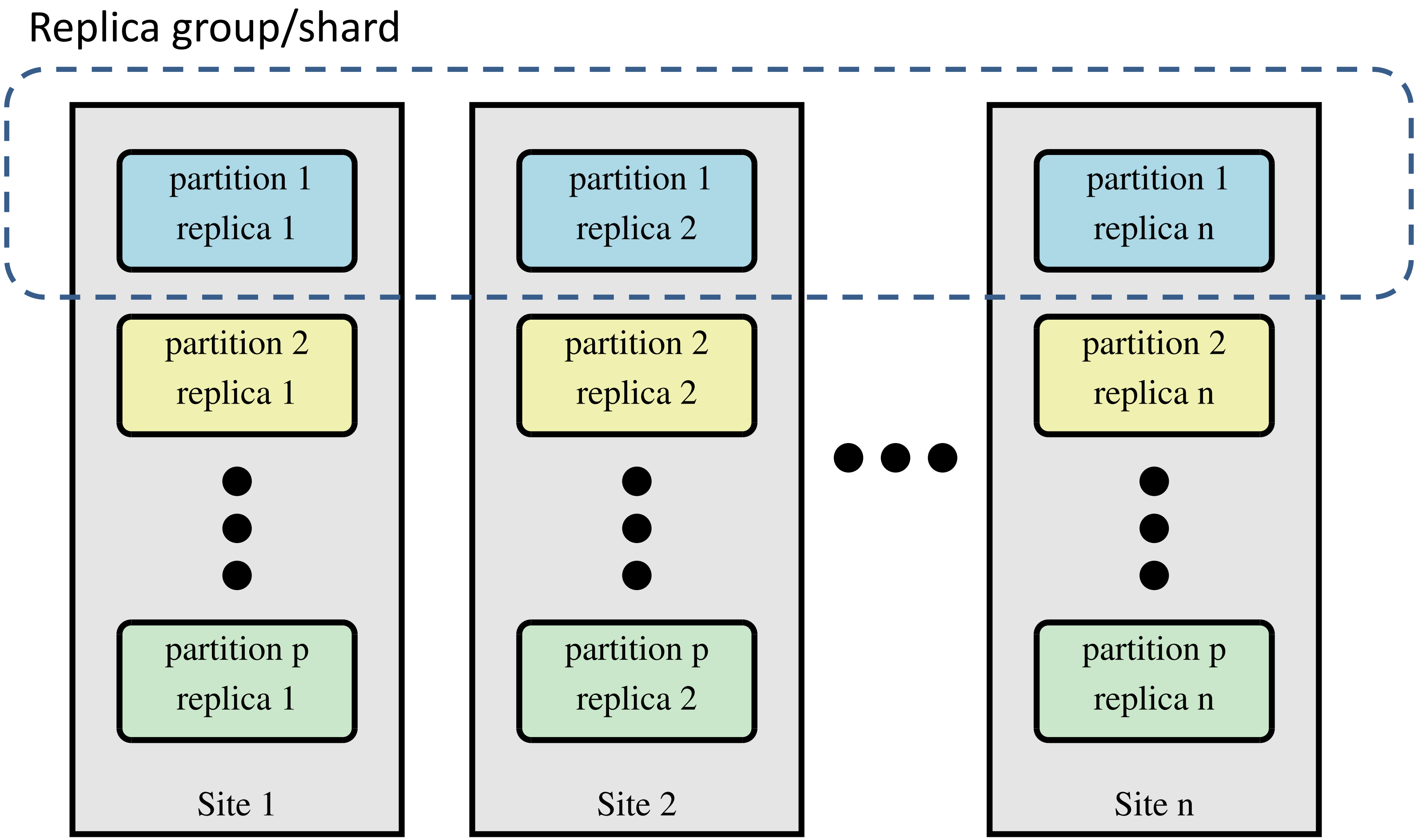




Replicated Sharded Data Store



Abstract View



Agrawal, El Abbadi, Salem, A Taxonomy of Partitioned Replicated Cloud-Based Database Systems

Sharded Transactional Store

- Full ACID semantics for transactions
- Transactions can span multiple shards
- Servers should only process transactions involving the shards they store
- Unlimited concurrency
- Up to f servers can fail in every shard

How to commit a transaction?

- In a sharded (partitioned) datastore
- Touching objects in multiple shards
- Assuming reliable shards (for now)
- ACID guarantees

How to commit a transaction?

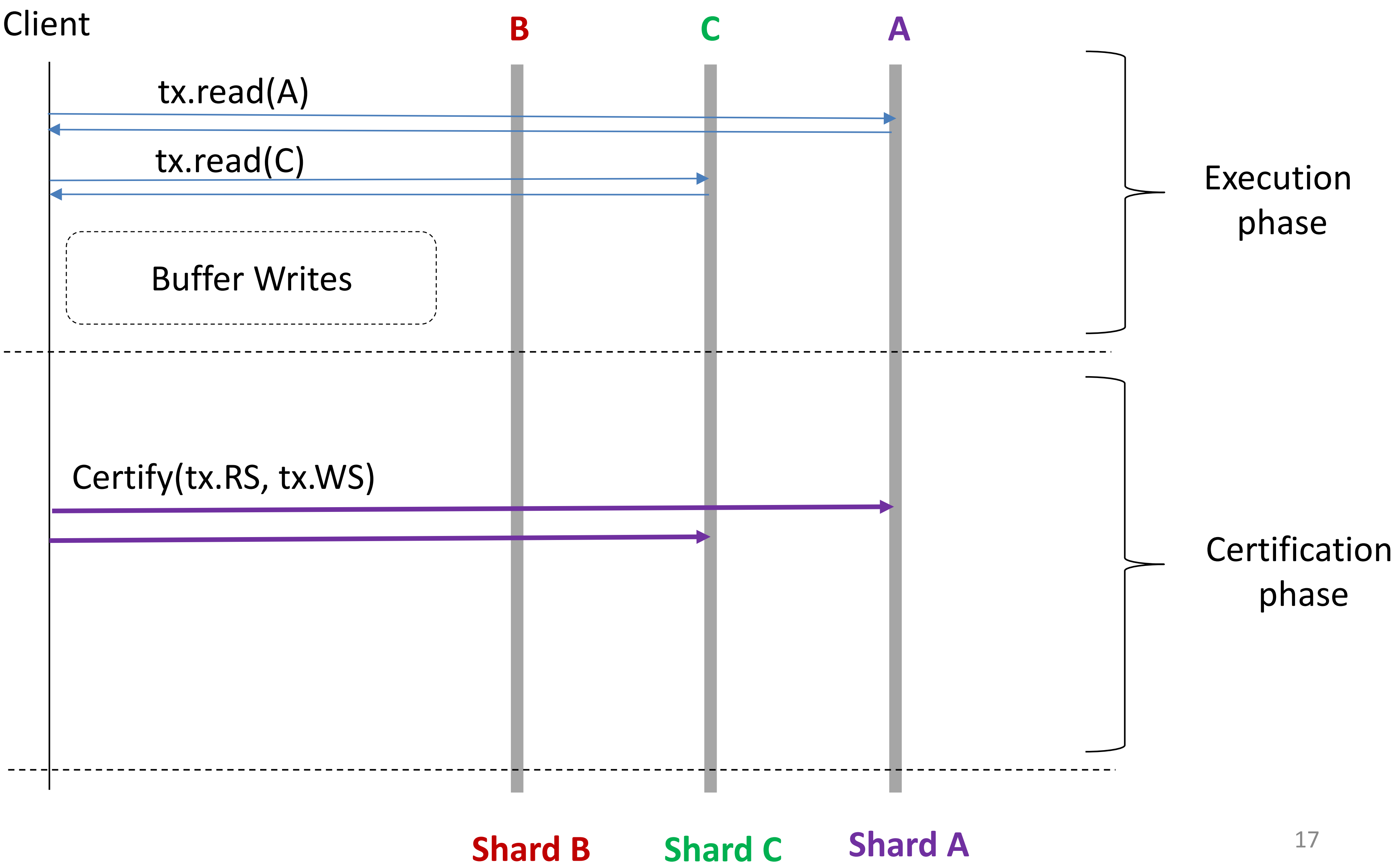
- Two-Phase Commit (2PC)
  - Coordinate distributed decision
- Concurrency control (CC)
  - Ensure a suitable isolation level

How to commit a transaction?

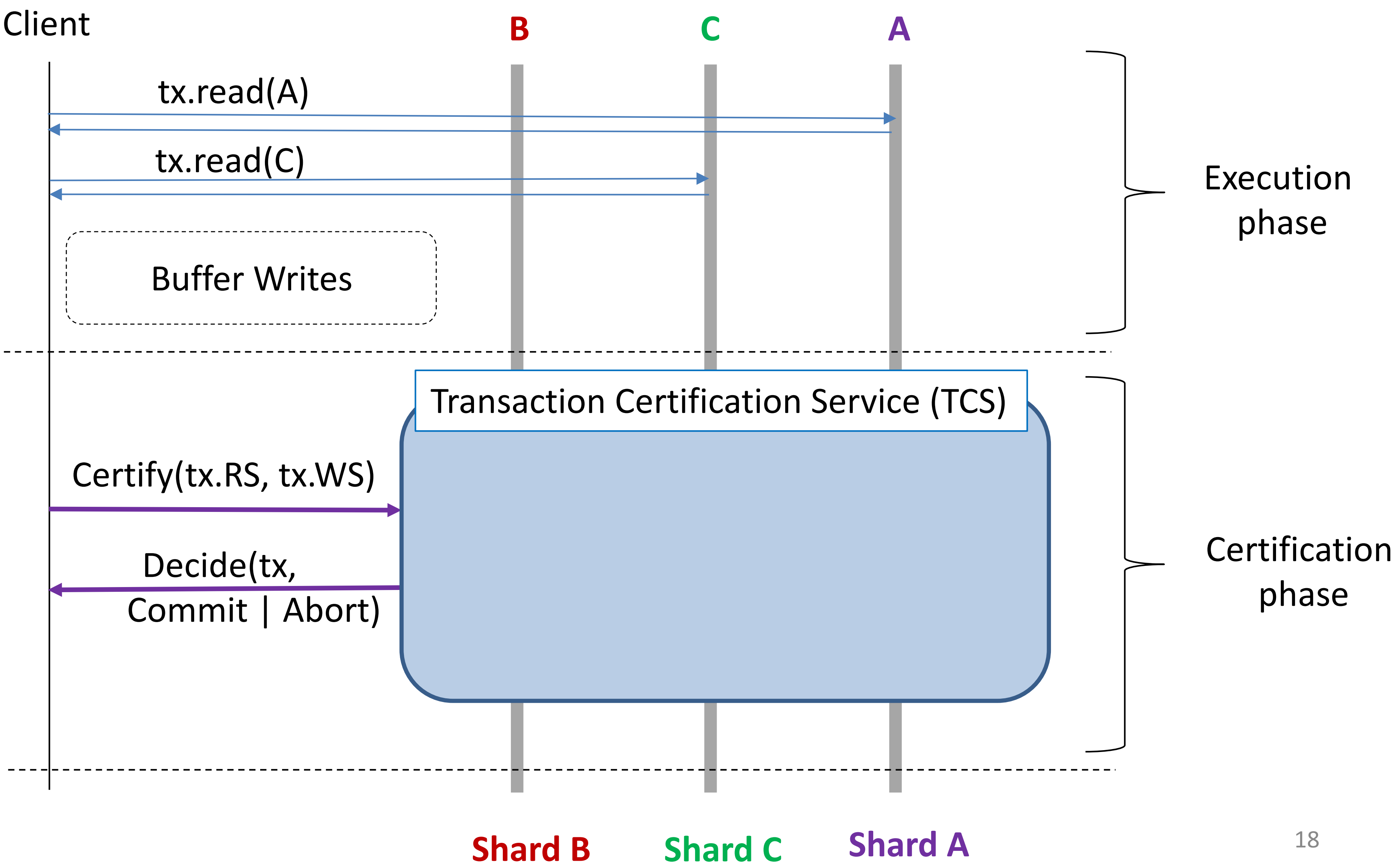
- Traditionally studied in isolation
- In reality, tightly intertwined

- We introduce a unified framework that captures both

Reliable Shards



Reliable Shards



Optimistic Concurrency Control (OCC)

- Reads can be served in any order by any replica
- Writes are buffered ("deferred")
- Certification requests carry read-set and write-set

Certification Functions

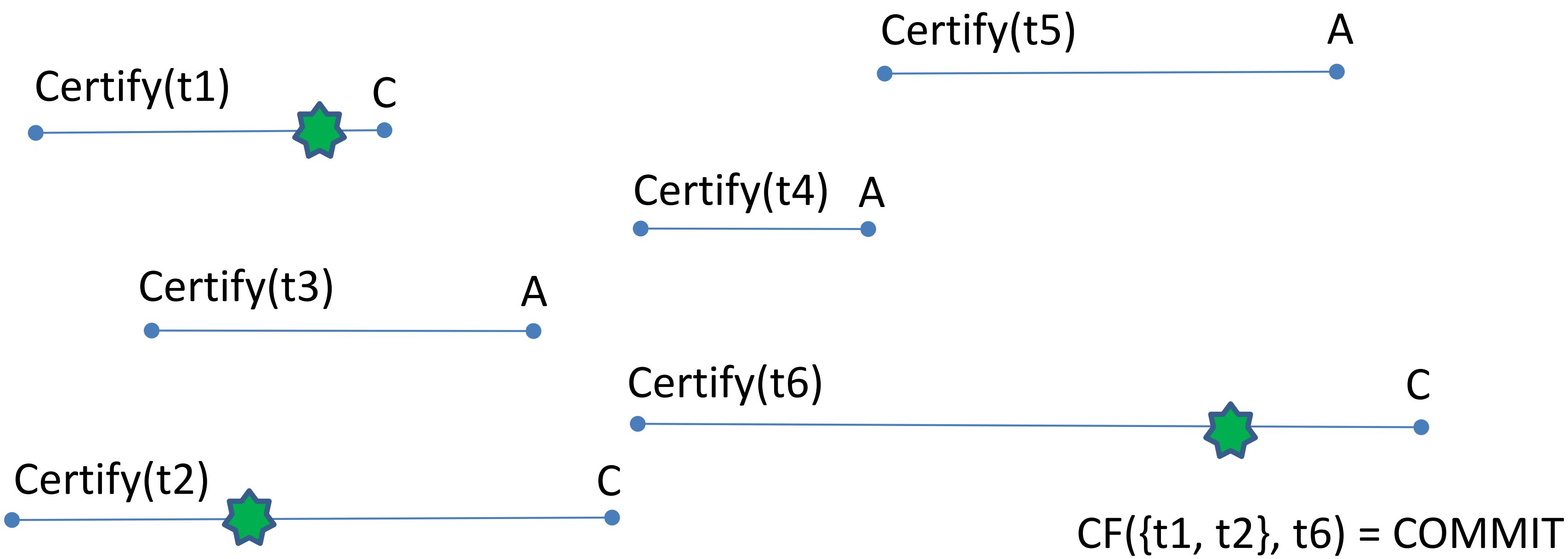
- Models conflict checks
- Determines if a transaction should COMMIT or ABORT given its context (prior committed transactions)

Serializability: $f(T, t) = \text{COMMIT}$ *iff*

$$\forall x, v.\, (x, v) \in R(t) \implies (\forall t' \in T.\, (x, \_) \in W(t') \implies V_c(t') \leq v).$$

Multi-Shot Transaction Commit

- Certify(tx), Decide(tx, COMMIT | ABORT)
- Correctness: CF-consistent linearization of committed Certify requests



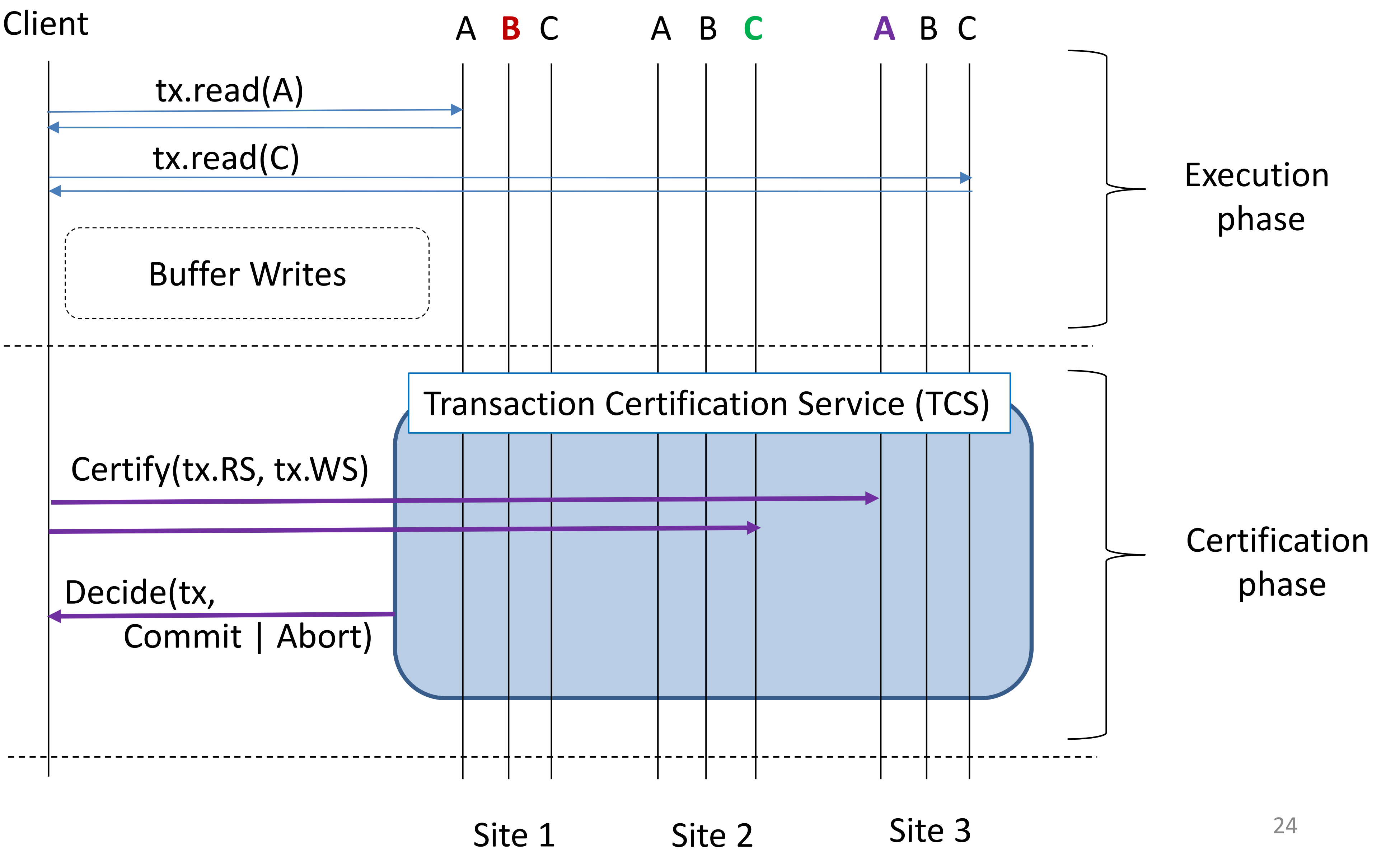
Problem Statement

- Implement a resilient TCS with a given CF
  - ≤ f servers can fail in every shard
  - Transactions can span multiple shards
  - Servers must only process transactions involving the shards they store
  - Unlimited concurrency
  - Eventual synchrony (or eventual leader)

Our Contribution

- Family of resilient cross-shard TCS protocols
- Crash fault-tolerant (CFT): 2f+1 replicas/group
  - Optimal latency, unbounded pipelining
- Byzantine fault-tolerant (BFT): 3f+1 replica/group
  - Latency matching PBFT, unbounded pipelining
  - First of a kind?
- Formal framework for multi-shot transaction commit

Setting



Client
A B C
A B C
A B C
tx.read(A)
tx.read(C)
Buffer Writes
Execution phase
Transaction Certification Service (TCS)
Certify(tx.RS, tx.WS)
Decide(tx,
Commit | Abort)
Certification phase
Site 1
Site 2
Site 3

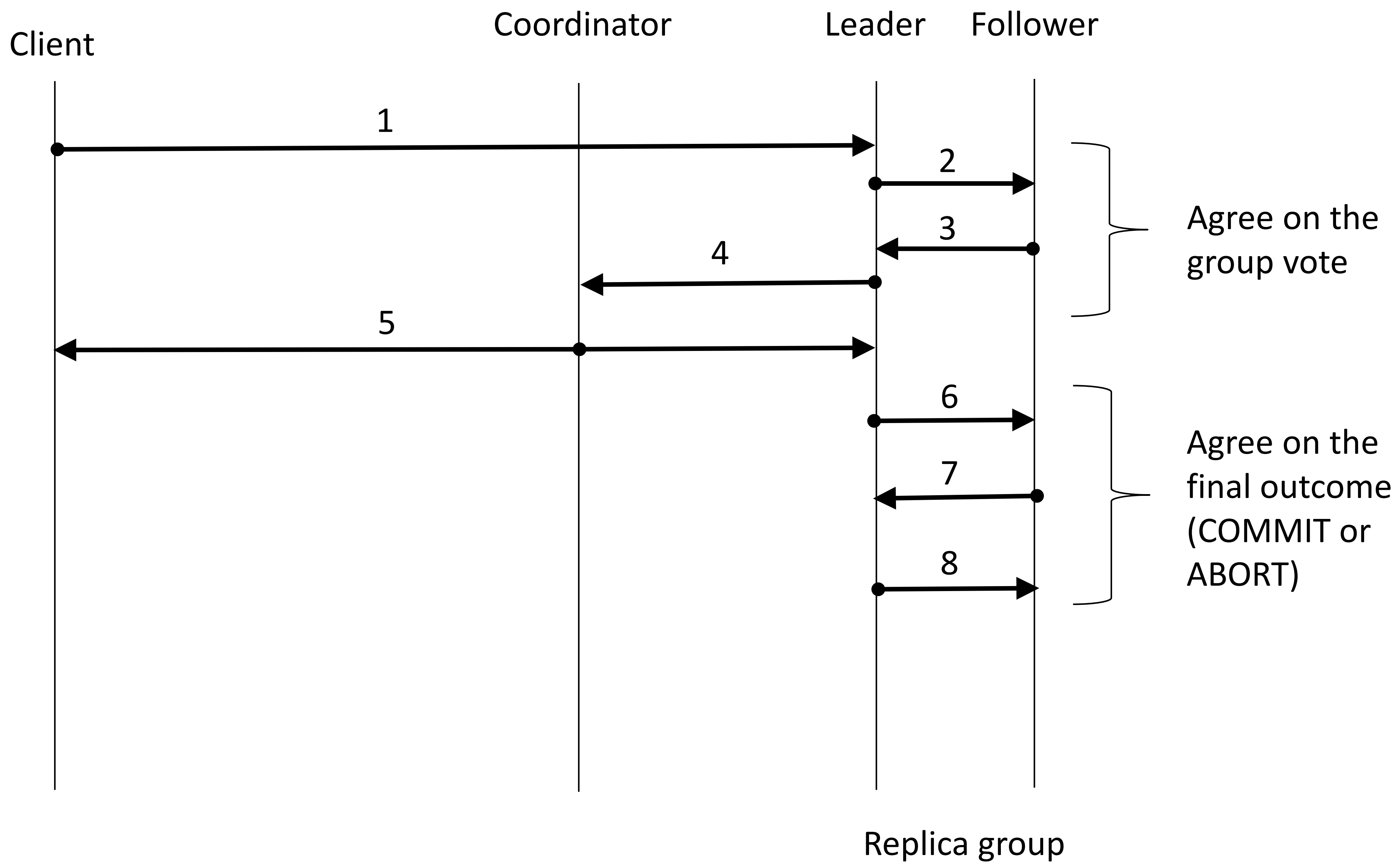
"Black-box" Solution

- Two-Phase Commit (2PC) to coordinate cross-shard transaction commit

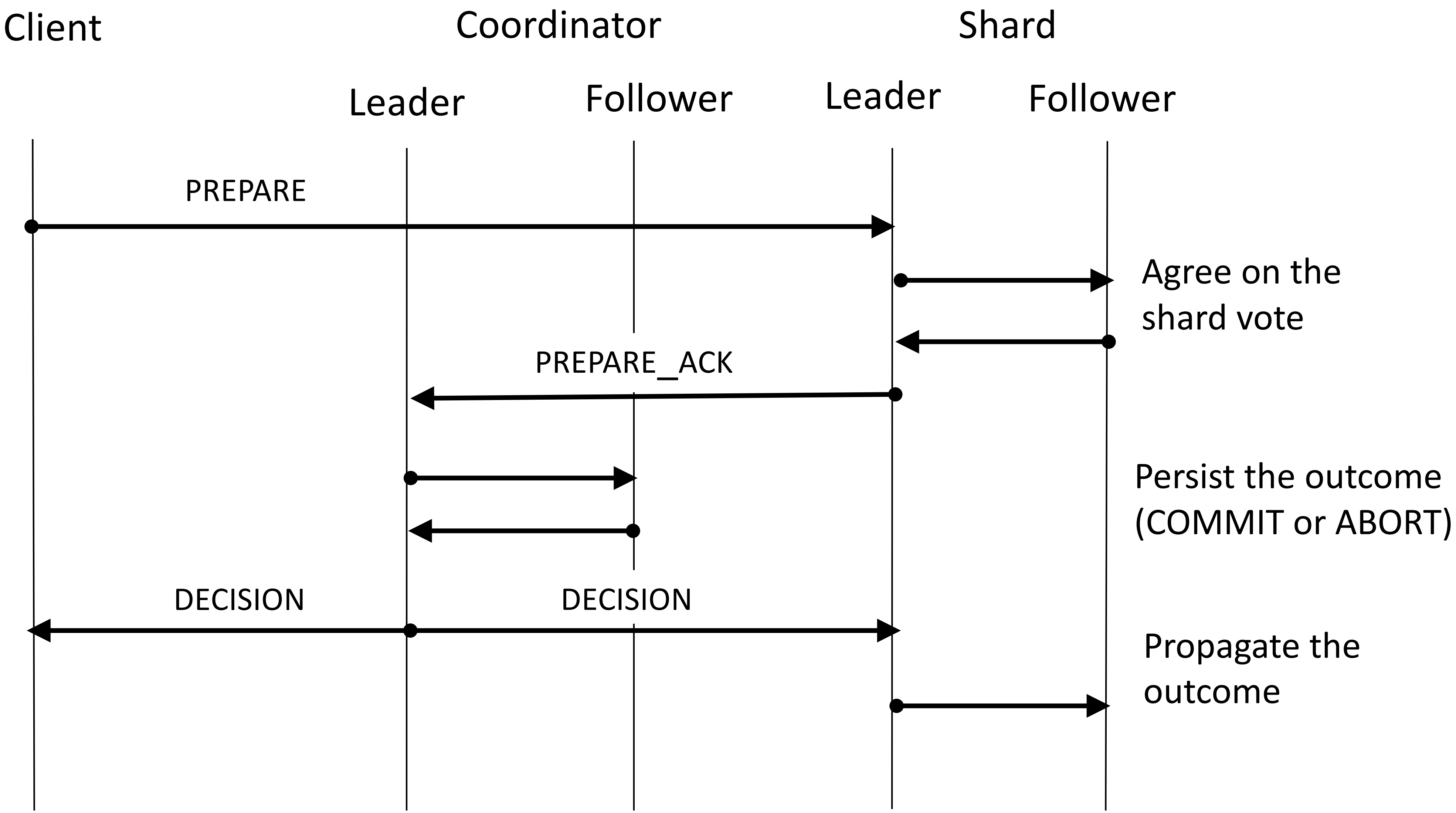
- Paxos in every replica group
  - Ordered transaction log for concurrency control
  - 2PC state for each transaction (prepared/committed/aborted)

- Examples: Spanner, Scatter, Granola, etc.

“Black-Box”: Failure-Free Case



“Black-Box”: Failure-Free Case



Goals

- Eliminate black-box abstractions
- Design single coherent protocol
- Identify optimization opportunities
  - For failure-free case

Recipe

- Abstract protocol to solve multi-shot transaction commit with reliable shards
- Refine to obtain a resilient solution for a desired failure model
  - CFT or BFT

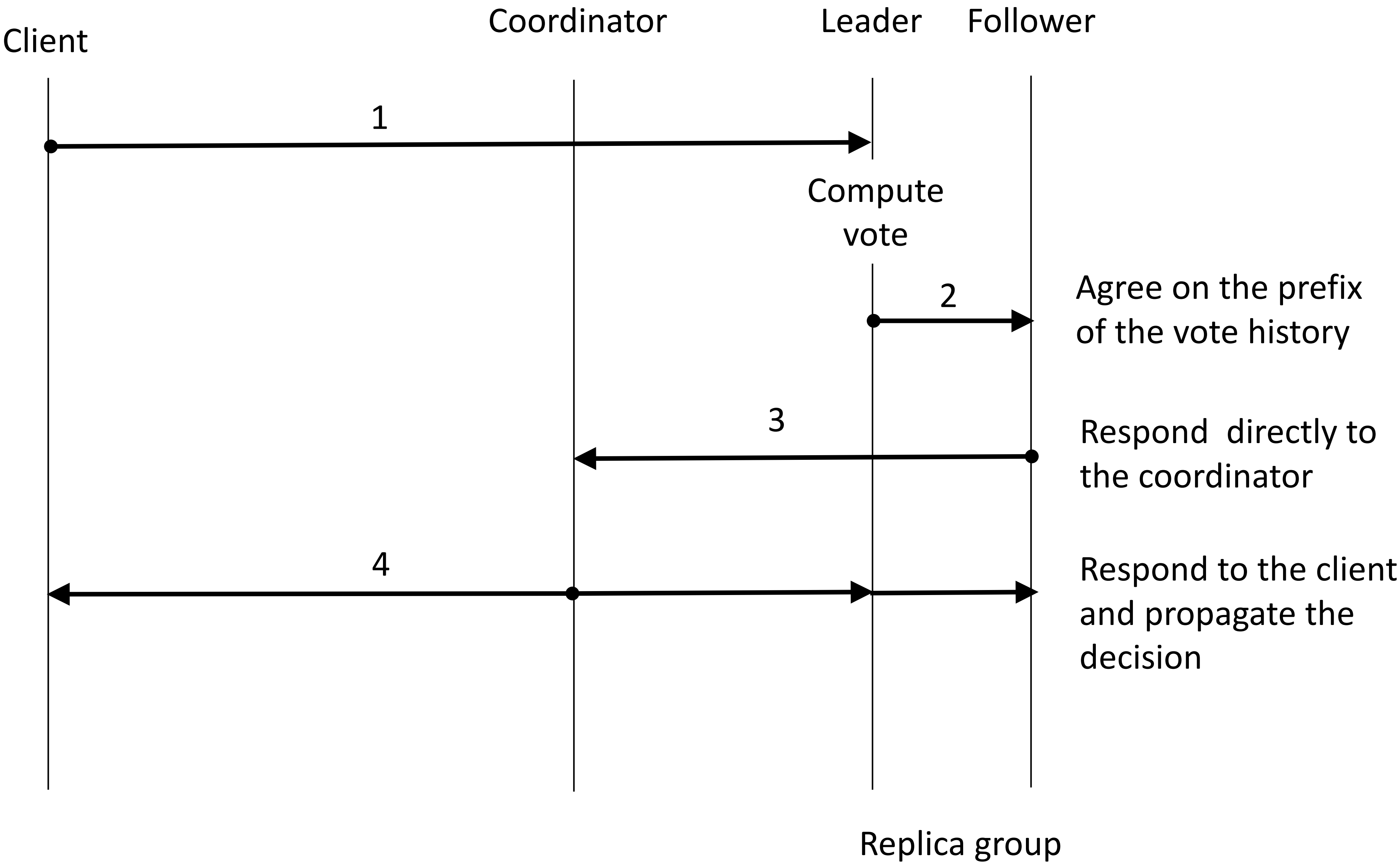
Abstract Multi-Shot 2PC

```
function certify(t)
    send PREPARE(t) to proc(shards(t));

when received PREPARE(t)
    next ← next + 1;
    txn[next] ← t;
    vote[next] ← f_{s_0}({txn[k] | k < next ∧ phase[k] = DECIDED ∧ dec[k] = COMMIT}, t) ⊓
                 g_{s_0}({txn[k] | k < next ∧ phase[k] = PREPARED ∧ vote[k] = COMMIT}, t);
    phase[next] ← PREPARED;
    send PREPARE_ACK(s_0, next, t, vote[next]) to coord(t);

when received PREPARE_ACK(s, pos_s, t, d_s) for every s ∈ shards(t)
    send DECISION(t, ⨅_{s∈shards(t)} d_s) to client(t);
    forall s ∈ shards(t) do send DECISION(pos_s, ⨅_{s∈shards(t)} d_s) to proc(s);

when received DECISION(k, d)
    dec[k] ← d;
    phase[k] ← DECIDED;
```

CFT Protocol Overview

- Maintain a contiguous prefix of the vote history (certification order) in every replica group
- Leaders can compute votes locally and persist at a majority through a consensus round
  - No need in 2nd consensus to persist the decision!
- Coordinator is stateless
  - Can learn of the votes directly from replicas
  - Can also drive a new decision without risking conflicts

CFT Protocol: Failure-Free Case



BFT Protocol Overview

- Reduce to a single consensus via certification order agreement in every replica group
- Extra message delay to verify the leader's vote
  - Vote computation verification to prevent spurious aborts
- Many-to-many exchange pattern for independent verification
- Failure-free latency is the same as PBFT

Failure-Free Case Latency

	CFT	BFT
"Black-box" TCS protocols	5 (client) 8 (decision)	N/A
Our TCS protocols	4 (client) 4 (decision)	5 (client and decision)

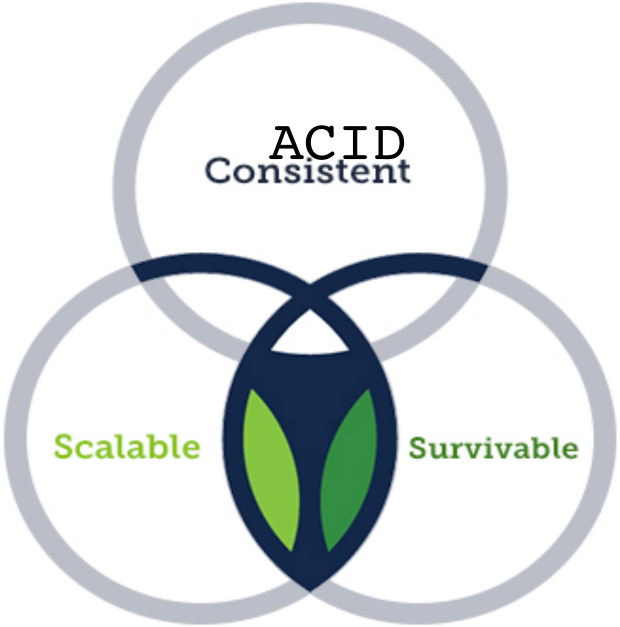
- Match latency of consensus
- Known to be optimal for CFT
  - Collocated coordinator and client
  - Use many-to-many exchange instead of leader-based protocol

Future Work

- Study complexity for different sharding strategies

- Support pessimistic concurrency control

- TCS BFT protocol for blockchain scaling
  - E.g., cross-channel transactions in Hyperledger



Thank You!