

CPSC 416 – Distributed Systems

Lecture 20: Distributed Transactions

UBC Vancouver

2026-03-19

From Last Time

Turn to your neighbor. 2 minutes.

Spanner gives you globally consistent reads via TrueTime.

How does it handle *writes* that touch multiple shards?

(Pairs: 2 minutes)

Recall from L19

Spanner combines two mechanisms:

- ① **Within each shard:** Paxos replication (Multi-Paxos log)
- ② **Across shards:** ??? (we named it but didn't examine it)

Today: we examine that cross-shard mechanism, understand why it blocks, and see alternatives.

This is the lecture where everything connects.

Failure-first question

Your data is sharded across 5 machines. A transaction touches 3 shards.

How do you make it atomic?

“Atomic” means: either all 3 shards commit, or none do.

No shard should commit if another will abort.

No shard should abort if another has already committed.

Why Shard?

Single-machine databases are simple. One log, one lock manager, one commit.

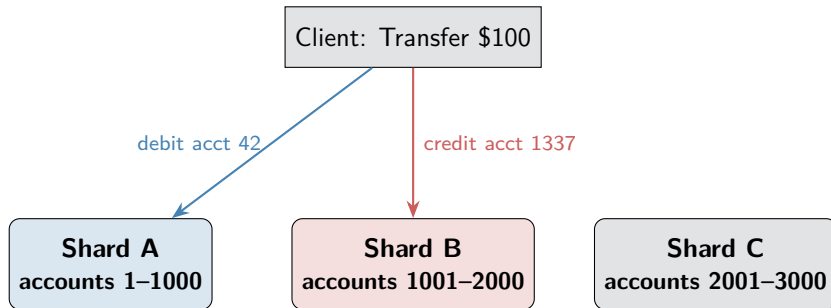
But:

- Data grows beyond one machine's capacity
- Read throughput needs to scale horizontally
- Failure domains: one machine crash shouldn't lose everything

Sharding = partitioning data across machines by key range or hash.

Sharding gives you scale. It takes away atomicity.

The Cross-Shard Problem



Two shards must agree: **both commit** or **both abort**.
Shard A can't decide alone. Shard B can't decide alone.

What if Shard A commits but
Shard B crashes before committing?

Checkpoint 1

Articulate:

What makes cross-shard transactions harder than single-shard transactions?

What specific failure would violate atomicity?

(Pairs: 2 minutes)

Two-Phase Commit (2PC)

Idea: Separate the *decision* from the *execution*.

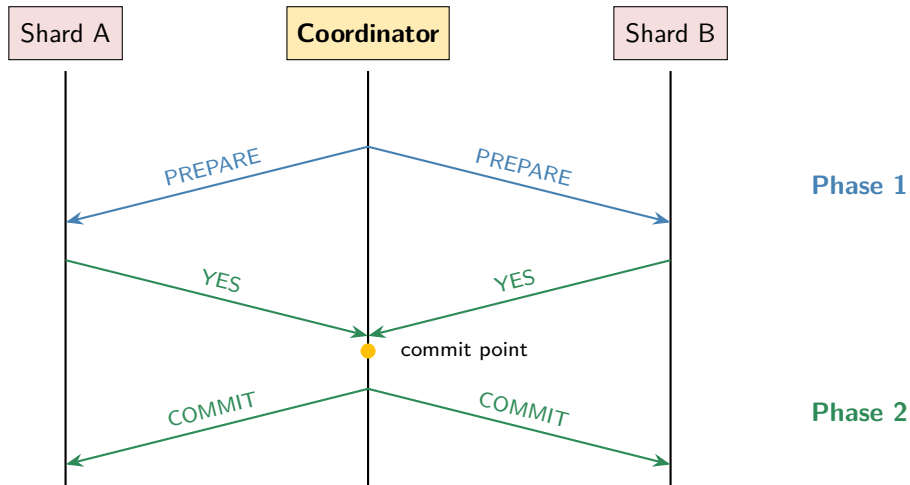
Phase 1 (Prepare):

- Coordinator asks each participant: “Can you commit?”
- Each participant acquires locks, writes to durable log
- Participant votes YES or NO

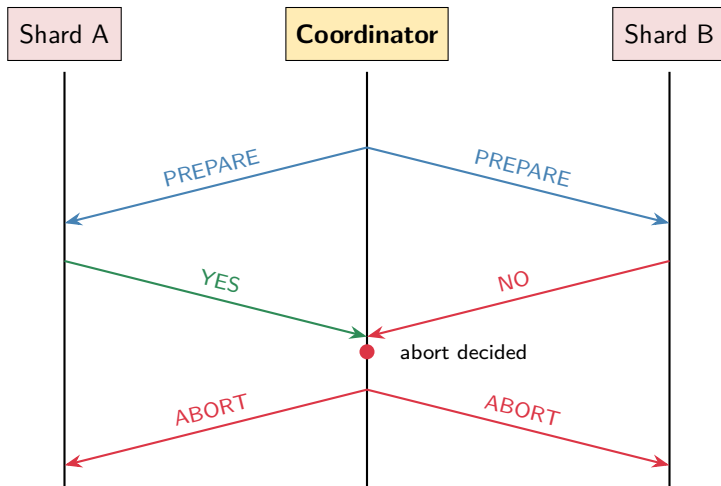
Phase 2 (Commit/Abort):

- If **all** vote YES: coordinator sends COMMIT
- If **any** votes NO (or times out): coordinator sends ABORT
- Participants execute the decision

2PC: The Happy Path



2PC: The Abort Path



Any NO vote (or timeout) → abort. **2PC requires unanimous consent.**

Recognize This?

2PC requires *unanimous consent*.

That's where L10 started.

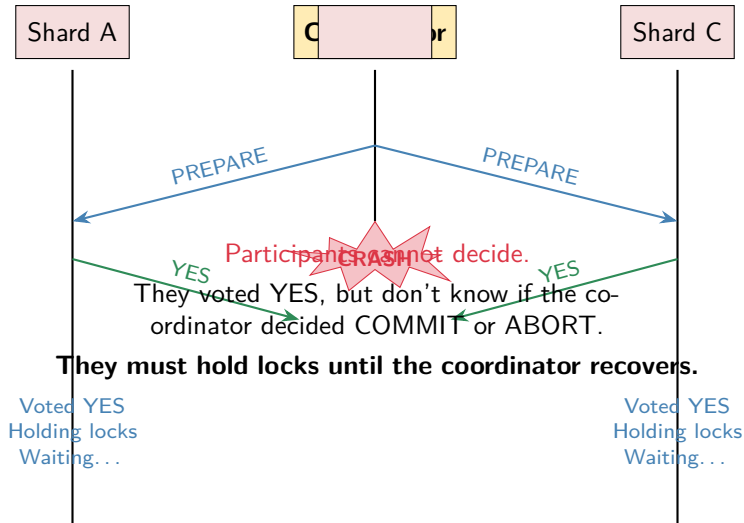
- Coordinator proposes, participants vote
- Any “no” or timeout → abort
- Coordinator fails after prepare → participants blocked

We escaped unanimity for *consensus* via quorum intersection.

For transactions, we're back to unanimity. Why?

Because each shard holds *different data*. You can't skip a shard.

The Blocking Problem



Why Participants Are Stuck

After voting YES, a participant is in the **prepared** state:

- **Can't commit:** Maybe the coordinator will abort (another participant voted NO)
- **Can't abort:** Maybe the coordinator already sent COMMIT to others
- **Can't release locks:** Must preserve isolation until decided

The coordinator is the single point of failure.

This is not a bug. It is a structural consequence of unanimous consent.

2PC is safe. It blocks because safety + unanimity + coordinator failure = waiting.

2PC is “Paxos with One Acceptor”

Think about it:

Paxos	2PC
Proposer	Coordinator
Acceptors (quorum)	Coordinator (alone)
Chosen = accepted by quorum	Decided = coordinator says so
Proposer crash → new proposer	Coordinator crash → block
Safety via quorum intersection	Safety via unanimous vote

In 2PC, the coordinator is the *only* node that knows the decision.

There is no quorum. There is no intersection. There is no alternative witness.

One acceptor = no fault tolerance for the decision.

Checkpoint 2

Articulate:

In Paxos, a proposer crash doesn't block progress. Why not?

In 2PC, a coordinator crash blocks everything. Why?

What's structurally different?

(Pairs: 2 minutes)

In-Class Activity: Breaking 2PC

Setup: 3 shards (A, B, C). Coordinator sends PREPARE to all three.

Scenario:

- 1 Coordinator sends PREPARE to A, B, C
- 2 A and B reply YES
- 3 C has not yet replied
- 4 **Coordinator crashes**

Draw the trace. For each participant, answer:

- What state is it in?
- What can it safely do?
- What happens if it contacts the other participants?

Then: **compare to Paxos.** What happens when a proposer crashes mid-Phase-2?

(Groups of 3: 8 minutes)

Activity Debrief: The 2PC Trace

Shard A: Voted YES. Holding locks. Cannot decide alone.

Shard B: Voted YES. Holding locks. Cannot decide alone.

Shard C: Never received PREPARE (or received it, hasn't voted). Can abort unilaterally.

Can A and B ask each other?

Yes — but they only learn that both voted YES. They don't know about C.

Can they ask C?

If C is reachable: C hasn't voted, so the transaction's fate is unknown.

If C is also down: A and B are stuck indefinitely.

The protocol has no mechanism for participants to decide without the coordinator.

Activity Debrief: The Paxos Comparison

In Paxos, when a proposer crashes mid-Phase-2:

- A new proposer runs Phase 1 with a higher ballot
- Phase 1 quorum intersects Phase 2 quorum — discovers accepted values
- New proposer adopts the value and completes Phase 2
- Progress is made. No blocking.

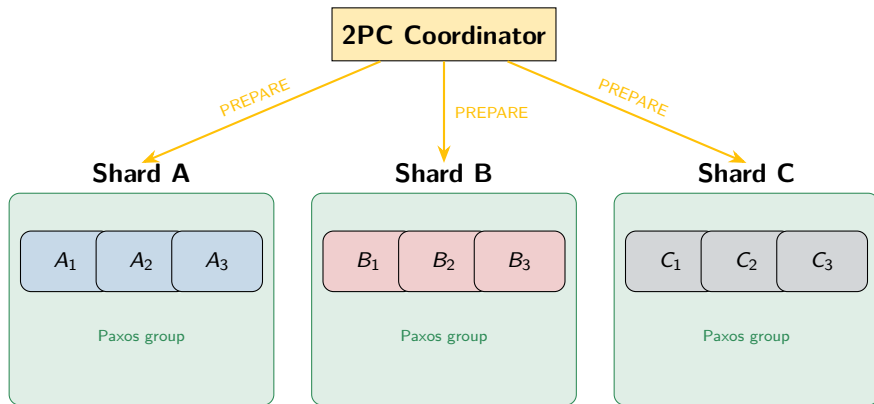
Why doesn't 2PC have this escape?

Because there is no “higher ballot.” There is no quorum of coordinators.

The decision lives in exactly one place: the coordinator.

Paxos replicates the decision. 2PC doesn't.

The Spanner Architecture



Paxos within each shard (replication, fault tolerance)
2PC across shards (atomicity)

Spanner's Key Insight

Problem: 2PC coordinator is a single point of failure.

Solution: *Replicate the coordinator's state with Paxos.*

- The coordinator is the leader of a Paxos group
- The PREPARE/COMMIT decisions are logged via Paxos
- If the coordinator crashes, a new leader takes over *with the decision intact*

Each participant is also a Paxos group:

- The “prepared” state is replicated
- Participant crash doesn't lose the vote

2PC provides atomicity. Paxos provides fault tolerance. Together: atomic + available.

The Two Levels

Level	Mechanism
Within a shard	Multi-Paxos (L13)
Across shards	2PC
Coordinator survival	Paxos (replicate coordinator)
Participant survival	Paxos (replicate “prepared” state)

This is not two protocols bolted together.

It's a **layered composition**: 2PC for the transaction structure, Paxos for durability of every piece.

The blocking problem disappears because the coordinator is replicated.

Checkpoint 3

Articulate:

2PC blocks because the coordinator is a single point of failure.
Spanner replicates the coordinator with Paxos.

Does Spanner's 2PC still block? Under what conditions?

(Pairs: 3 minutes)

Calvin: A Different Path

Spanner's approach: Let transactions execute, then coordinate the commit.

Calvin's approach (Thomson et al., 2012): Agree on the *order* first.

- A global sequencer assigns a total order to all transactions
- Each shard receives the ordered log of transactions
- Each shard executes transactions in that order, *deterministically*
- No 2PC needed: if everyone executes the same log, they agree

If the input is agreed, the output is agreed. Determinism replaces coordination.

Calvin vs Spanner: The Trade-off

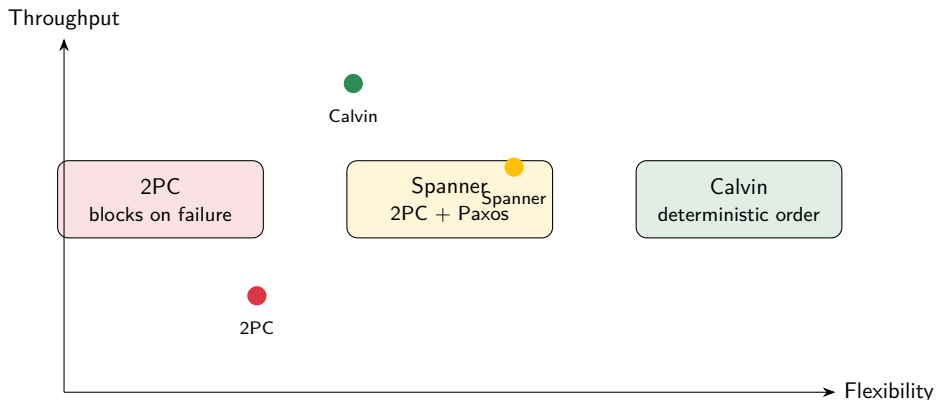
	Spanner	Calvin
Cross-shard mechanism	2PC (replicated)	Deterministic ordering
Coordinator	Paxos-replicated	Sequencer (replicated)
Latency per txn	2PC round-trips	Sequencing + execution
Aborts	Natural (vote NO)	Must pre-declare read/write sets
Interactive txns	Supported	Not supported
Blocking	Only if Paxos group fails	Only if sequencer fails

Calvin's constraint: You must know the read/write set *before* execution.

No “read X, then decide whether to write Y.” The plan is fixed.

Calvin trades flexibility for throughput.

The Design Space



- **2PC:** Simple, blocks, maximum flexibility
- **Spanner:** Eliminates blocking via replication, keeps flexibility
- **Calvin:** Eliminates 2PC entirely, restricts transaction model

Side-by-Side: 2PC Blocking vs Paxos Proposer Crash

2PC: Coordinator Crash

1. Coordinator: PREPARE
2. Participants: YES
3. **Coordinator crashes**
4. Participants hold locks
5. **Wait forever**

Decision stored in
one place (coordinator).
No alternative path.

Paxos: Proposer Crash

1. Proposer: ACCEPT(b, v)
2. Some acceptors: accepted
3. **Proposer crashes**
4. New proposer: PREPARE($b+1$)
5. **Discovers v , completes**

Decision replicated across
quorum of acceptors.
New proposer finds witness.

The structural difference: Paxos
stores the decision in a quorum.
2PC stores it in a single coordinator. **That's**
why Spanner replicates the coordinator.

Evidence Question

You're running Spanner. A cross-shard transaction is stuck.

What would you look for in the logs?

How do you distinguish:

- A slow participant from a crashed participant?
- A 2PC timeout from a Paxos leader election?
- A genuine conflict from a false positive?

(This is a real operational challenge at Google.)

What We Learned

- **Sharding** creates the cross-shard atomicity problem
- **2PC** solves atomicity via unanimous consent — but blocks on coordinator failure
- 2PC is structurally “Paxos with one acceptor” — no quorum, no fault tolerance for the decision
- **Spanner**: Paxos within shards + 2PC across shards + replicated coordinator
- **Calvin**: Deterministic ordering eliminates 2PC entirely (at the cost of flexibility)

The lattice connection:

L10 started from 2PC's unanimity problem and escaped via quorum intersection. L13 built Multi-Paxos logs. L14 showed quorums are more flexible than majority. L20 combines everything: the log, the quorum, the transaction, the shard.

Every tool we built serves this composition.

- **Reading:** “Paxos Made Live” (Chandra, Griesemer, Redstone — Google, 2007)
- **Optional:** “Calvin: Fast Distributed Transactions for Partitioned Database Systems” (Thomson et al., 2012)
- **Next lecture (L21):** Adversarial Protocol Design — includes a 2PC breaking exercise
- **Self-assessment:** Opens 13:00, due next class 11:30 (Pacific)
- Upcoming capstone milestones on Canvas