

# CPSC 416 – Distributed Systems

## Lecture 13: From Paxos to Multi-Paxos

UBC Vancouver

2026-02-24

# Welcome Back

It's been 12 days. Let's see what stuck.

# Explain Paxos

Turn to your neighbor. You have 3 minutes.

**Explain Paxos:** what are the phases, and *why* does each step exist?

No slides. No notes. Just talk.

*(Pairs: 3 minutes)*

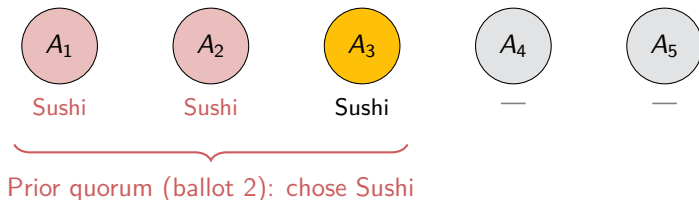
# Checkpoint 1

What did you remember? What did you have to reconstruct?

**The parts you reconstructed are the parts you understand.  
The parts you recited are the parts you memorized.**

*(Plenary: 3 minutes)*

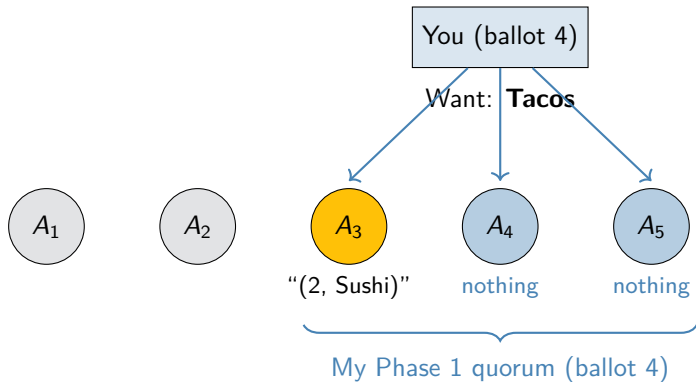
## Agreeing on Dinner: The Setup



Someone already proposed dinner at ballot 2. A quorum accepted **Sushi**.

$A_4$  and  $A_5$  weren't in that quorum. They don't know yet.

## Agreeing on Dinner: The Intersection Decides



You wanted Tacos.  $A_3$  says: "I already accepted **Sushi** at ballot 2."

Too bad. `ACCEPT(4, Sushi)`

Your preference is irrelevant. The quorum intersection decided.

*Keep this in mind. We're about to break it.*

## TLA+ Activity: Breaking Paxos

**Recall L09:** TLA+ lets us model-check distributed protocols.

**Setup:** I'll give you a correct Paxos TLA+ spec.

**Task:** Find the ballot comparison in the acceptor's Prepare handler. **Remove it.** Run the model checker. Find the safety violation.

**Goal:** Understand *why* that line exists by seeing what breaks without it.

*(Groups of 3: 15 minutes)*

# The Line You're Removing

## Original (correct):

```
HandlePrepare(a, b) ==  
  IF b > maxBallot[a]  
  THEN /\ maxBallot' = [maxBallot EXCEPT ![a] = b]  
        /\ Reply(a, PrepareResponse(...))  
  ELSE /\ Reply(a, Reject)
```

## Broken:

```
HandlePrepare(a, b) ==  
  /\ maxBallot' = [maxBallot EXCEPT ![a] = b]  
  /\ Reply(a, PrepareResponse(...))
```

## What happens?

# What the Model Checker Finds

## The counterexample:

- 1 Proposer A completes Phase 1 with ballot 1
- 2 Proposer A gets value  $v$  accepted by a quorum (chosen!)
- 3 Proposer B runs Phase 1 with ballot 2
- 4 **WITHOUT THE CHECK:** Acceptors in A's quorum "forget" ballot 1
- 5 Proposer B's Phase 1 doesn't see  $v$  (witnesses lost their memory)
- 6 Proposer B proposes  $w \neq v$  and gets it accepted
- 7 **Two different values chosen. Safety violated.**

The ballot check ensures witnesses remember.

## Checkpoint 2

You just discovered *why* the ballot check exists.

**In your own words: what invariant does it preserve?**

*(Plenary: 3 minutes)*

## Recall: The Liveness Problem (L12)

Before the break, we established:

- **Dueling proposers** prevent progress (infinite preemption)
- **FLP**: No way around this without timing assumptions
- **Escape**: Eventually, a *single proposer* must run long enough

**Question:** How do we get a single proposer?

Answer: Look at what Phase 1 already does.

# A Different Way to See Phase 1

Phase 1 doesn't just gather information.

**Phase 1 establishes who gets to propose.**

An acceptor's promise means:

*"I won't listen to anyone with a lower ballot."*

That's leader election.

# Single-Decree vs Multi-Decree

## Single-decree Paxos:

- Phase 1: “Who gets to propose?” (leader election)
- Phase 2: “What value?” (the decision)

## Multi-Paxos:

- Phase 1: “Who gets to propose?” (do this **once**)
- Phase 2, Phase 2, Phase 2...: slot 1, slot 2, slot 3...

Same machine. The leader doesn't change between decisions.

*This is why “Paxos” in practice looks different from textbook Paxos.*

# The Stable Leader Optimization

In practice, most Paxos deployments use a **stable leader**:

- One proposer is designated leader
- Leader runs Phase 1 **once** for a range of slots
- Subsequent proposals skip Phase 1 (leader already holds promises)
- If leader fails, new leader runs Phase 1 to take over

Phase 1 = leader election. Phase 2 = normal operation.

## Checkpoint 3

If Phase 1 is leader election, **what triggers a new Phase 1?**

*(Pairs: 2 minutes)*

## Checkpoint 4

Multi-Paxos amortizes leader election across many decisions.

**When would you *not* use Multi-Paxos?**

What kind of system needs single-decree Paxos?

*(Pairs: 2 minutes)*

## Example: Corfu

**Corfu** (Microsoft Research, 2012): a distributed shared log.

- Each log position is a **write-once** slot on a flash unit
- A *sequencer* hands out position numbers — but it's a **hint**, not a leader
- If the sequencer dies, clients probe for empty positions directly
- No stable leader on the data path at all

Each position is an independent single-decree decision.

Multi-Paxos assumes decisions are sequential. Corfu's aren't.

# State Machine Replication

**Goal:** All replicas apply the same commands in the same order.

The *log* is the fundamental abstraction.

Each slot in the log is one Paxos instance.

Agreeing on a sequence = agreeing on each entry independently.

# Who Is the Learner?

## What does “chosen” mean?

A value is chosen when accepted by a *quorum of acceptors*.

**Who are the acceptors?** The replicas. The database nodes.  
They already know what they accepted. *They are the learners.*

**The client** doesn't hear from the quorum. It asks one replica.

That replica must ensure consistency before answering:

- 1 Check for in-flight proposals that could change the value
- 2 Serialize the read against pending writes
- 3 Only then return the result

*The learner is the database, not the client.*

# The Log

The log as the fundamental abstraction:

- **Writes:** Propose a value for the next slot (via Phase 2)
- **Reads:** Find the current value, ensuring no in-flight conflicts
- **Snapshots:** Capture state at a point, truncate log before it

The log grows. Snapshots let you forget.

*(The idea of compaction will come back later in the course.)*

## Checkpoint 5

You're building a replicated key-value store using Multi-Paxos.

A client reads key  $K$ . There's a Phase 2 in progress for a write to  $K$ .

**What does the learner do?**

*(Pairs: 2 minutes)*

# What We Learned

- Phase 1 is leader election
- **Multi-Paxos** = leader election once + repeated Phase 2
- The **learner is the database**, not the client
- Consistent reads require serialization against in-flight proposals
- The **log** is the fundamental abstraction

# Paxos Made Live: The Gap

*The paper you read (Paxos Made Live, Google 2007) describes the gap between the algorithm and production.*

Key gaps:

- Disk corruption
- Leader leases
- Group membership changes
- Snapshots

Understanding single-decree Paxos is necessary but not sufficient.

L14 returns to first principles: *the majority was never the point.*

## Evidence Question

You're operating a Multi-Paxos system. A client reports **stale reads**.

Where in the system would you look? What would you check?

*(Hint: think about the learner's serialization.)*

- **Next lecture:** Flexible Paxos – quorum flexibility
- **Reading for L14:** Flexible Paxos (Howard, Malkhi, Spiegelman, 2016)
- **Capstone Report 6:** Due Thursday (Feb 26) at 11:30
- **DP1: Reflection Report:** Due Tuesday (Mar 3) at 11:30
- **DP2: Peer Reviews:** Due Tuesday (Mar 3) at 11:30