

CPSC 416 – Distributed Systems

Lecture 11: Walking the Paxos Path

UBC Vancouver

2026-02-10

Failure-first question

We've derived the shape of consensus: quorum intersection + ballot ordering.

What does the actual protocol look like, and why those specific steps?

Recall from L10

We established:

- ① **Quorum systems:** Any collection where $\forall Q_1, Q_2 : Q_1 \cap Q_2 \neq \emptyset$
- ② **Ordering via ballots:** (round, proposer_id) from the Bakery
- ③ **The structure:** Claim a ballot $\rightarrow$ query quorum $\rightarrow$ propose to quorum

Today: we walk through Paxos and see *why each line exists*.

Paxos Roles

Proposers: Entities that want to propose values

- Pick ballots, drive the protocol

Acceptors: The quorum members

- Store accepted values, make promises
- The “nodes” in our quorum diagrams

Learners: Entities that need to know the chosen value

- Often the same as proposers in practice

These roles can overlap. A node can be all three.

Checkpoint 1

Articulate:

Why do we separate “proposer” from “acceptor”?

What would go wrong if every node just proposed its own value directly?

(Pairs: 2 minutes)

Phase 1: Prepare

Goal: Learn what (if anything) has already been accepted.

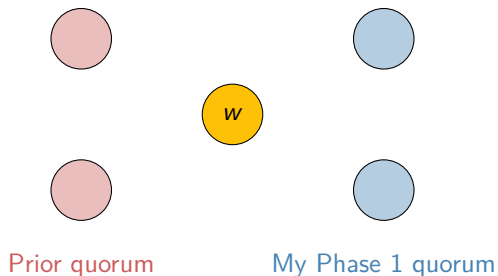
Proposer sends: PREPARE(ballot)

Acceptor responds:

- If $\text{ballot} > \text{highest_promised}$:
 Promise not to accept lower ballots.
 Return any previously accepted (ballot, value).
- Otherwise: Reject (or ignore).

Proposer waits for responses from a **quorum**.

Why Phase 1 Exists



The intersection property guarantees:
If anything was previously accepted by a quorum,
my Phase 1 quorum contains at least one witness.

The Promise

When an acceptor promises ballot b :

“I will not accept any proposal with ballot $< b$.”

This is the ordering mechanism.

Higher ballots “win” not by force, but because acceptors voluntarily reject lower ones.

(Recall: Bakery algorithm : customers with higher numbers wait.)

Checkpoint 2

Articulate:

What does the proposer *do* with the responses from Phase 1?

If multiple acceptors report different previously-accepted values, which one matters?

(Pairs: 2 minutes)

Phase 2: Accept

Proposer chooses a value:

- If any acceptor reported a previously accepted value:
Use the value with the **highest ballot**.
- Otherwise: Proposer can use its own value.

Proposer sends: `ACCEPT(ballot, value)`

Acceptor responds:

- If `ballot >= highest_promised`: Accept it.
- Otherwise: Reject.

Value is **chosen** when accepted by a quorum.

Why “Highest Ballot Wins”

Scenario:

- Ballot 5 chose value v (accepted by a quorum)
- New proposer runs Phase 1 with ballot 7

The intersection property:

The new proposer's Phase 1 quorum intersects the Phase 2 quorum that chose v .

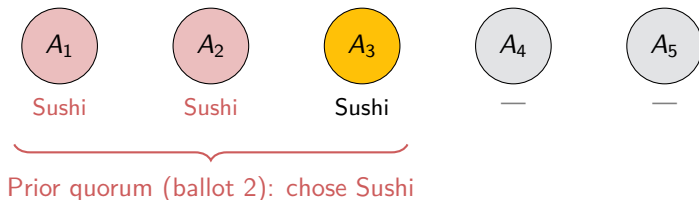
Therefore:

At least one acceptor reports $(5, v)$.

The proposer *must* use v in its Phase 2.

Chosen values are never overwritten. They propagate.

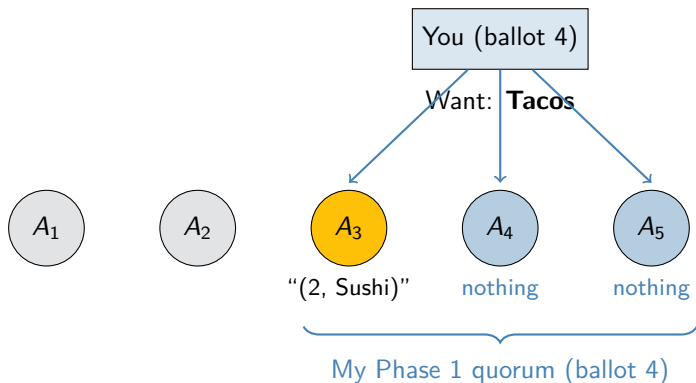
Agreeing on Dinner: The Setup



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: Your Turn



You send $\text{PREPARE}(4)$. You want Tacos.

A_3 replies: "I already accepted **Sushi** at ballot 2."

A_4, A_5 reply: "Nothing yet."

Agreeing on Dinner: The Intersection Decides

You wanted **Tacos**.

Too bad.

A_3 is in the intersection of the prior quorum and yours.

A_3 remembers Sushi. The protocol says: **adopt the highest-ballot value**.

ACCEPT(4, Sushi)

Your preference is irrelevant. The quorum intersection decided.

Chosen values propagate. They are never overwritten.

The Key Invariant

If value v is chosen, then any higher-ballot proposal will propose v .

Equivalently:

$$\text{chosen}(v_1) \wedge \text{chosen}(v_2) \implies v_1 = v_2$$

This is the safety property. Everything else serves this.

- **DP1 Implementation:** submitted
- **Due Thursday 23:59:** Capstone Team Declaration
- **Reading for Thursday:** Continue with “Paxos Made Simple”
- **Self-assessment:** Opens 13:00, due Thursday 11:30
- **Next lecture:** Breaking Paxos : failure modes and liveness