

Distributed Consensus: Paxos

Rong Chen

Institute of Parallel and Distributed Systems

Shanghai Jiao Tong University

http://ipads.se.sjtu.edu.cn/rong_chen

Concurrency so far

Atomicity and Concurrency Control

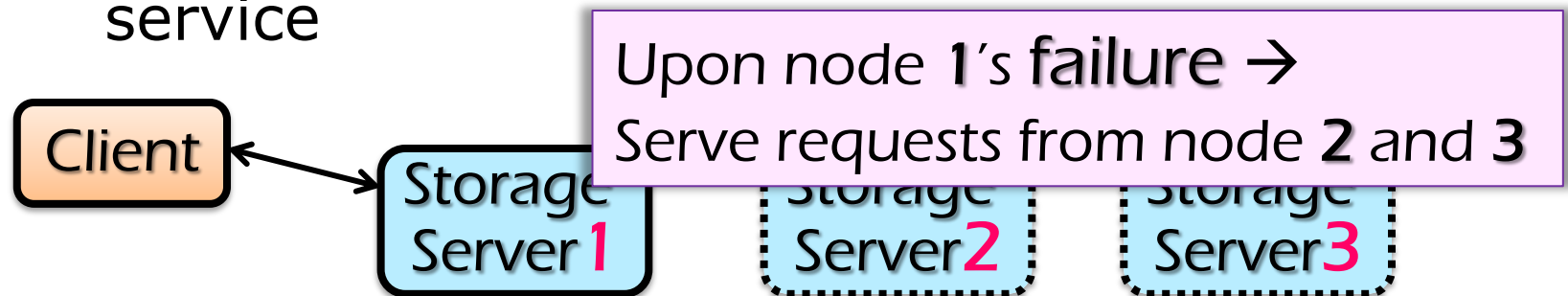
- All-or-nothing atomicity in transactions
 - **DO-REDO-UNDO** protocol
- Serializability of multiple transactions
 - **Two-phase Locking** (2PL)
 - **Snapshot Isolation** (SI)
- Multiple transactions in distributed systems
 - **Two-phase Commit** (2PC)

How about **fault tolerance**
in distributed systems?

Fault Tolerance: Replication

How to **recover** a single node from **failure**?

- Wait for **reboot**
 - Data is durable
 - Service is **unavailable** temporarily
- Use **multiple** nodes to provide **equivalent** service



Challenge: ensure **sequential** consistency across such reconfiguration

RSM: Replicated State Machine

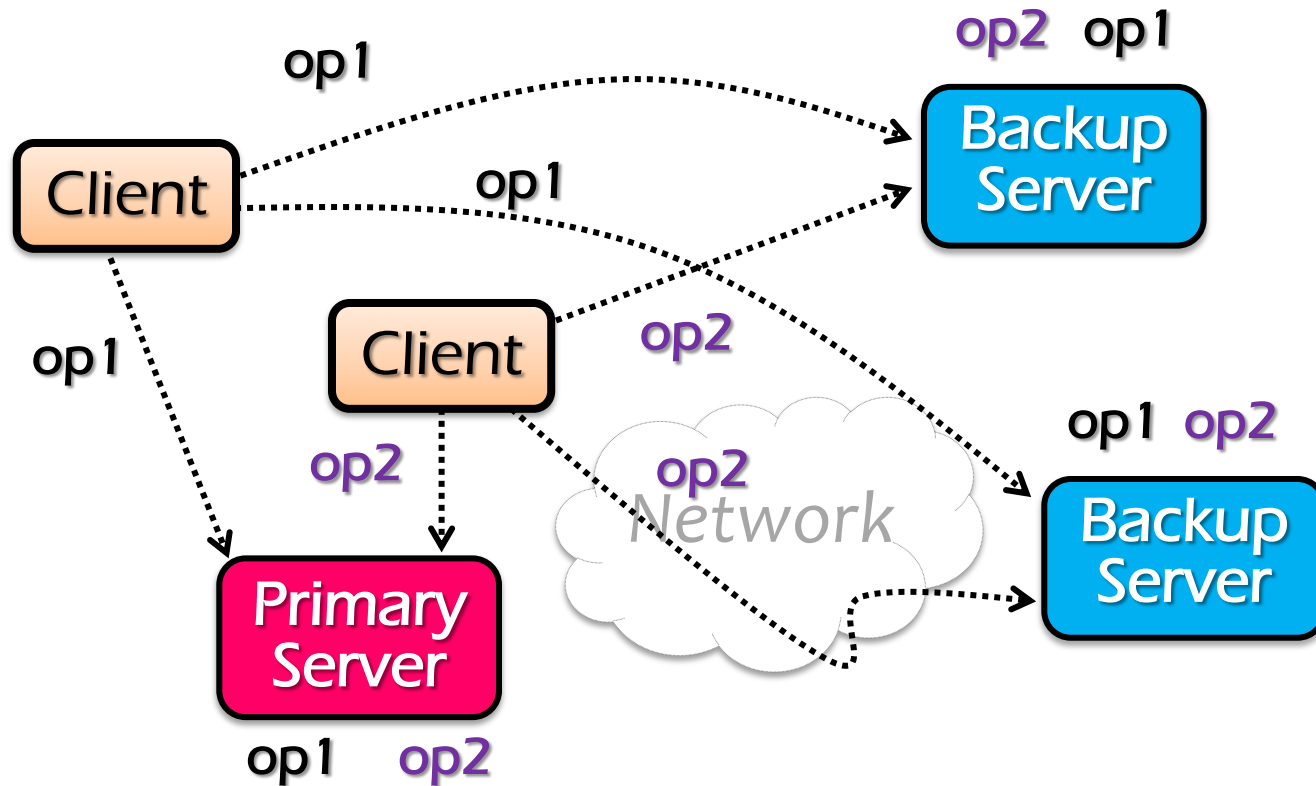
RSM is a general replication method

- e.g., apply RSM to lock service

RSM rules

- All replicas **start** in the **same** initial state
- Each replica apply ops in the **same** order
- All ops must be **deterministic**
- All replicas **end** up in the **same** state

Strawman RSM

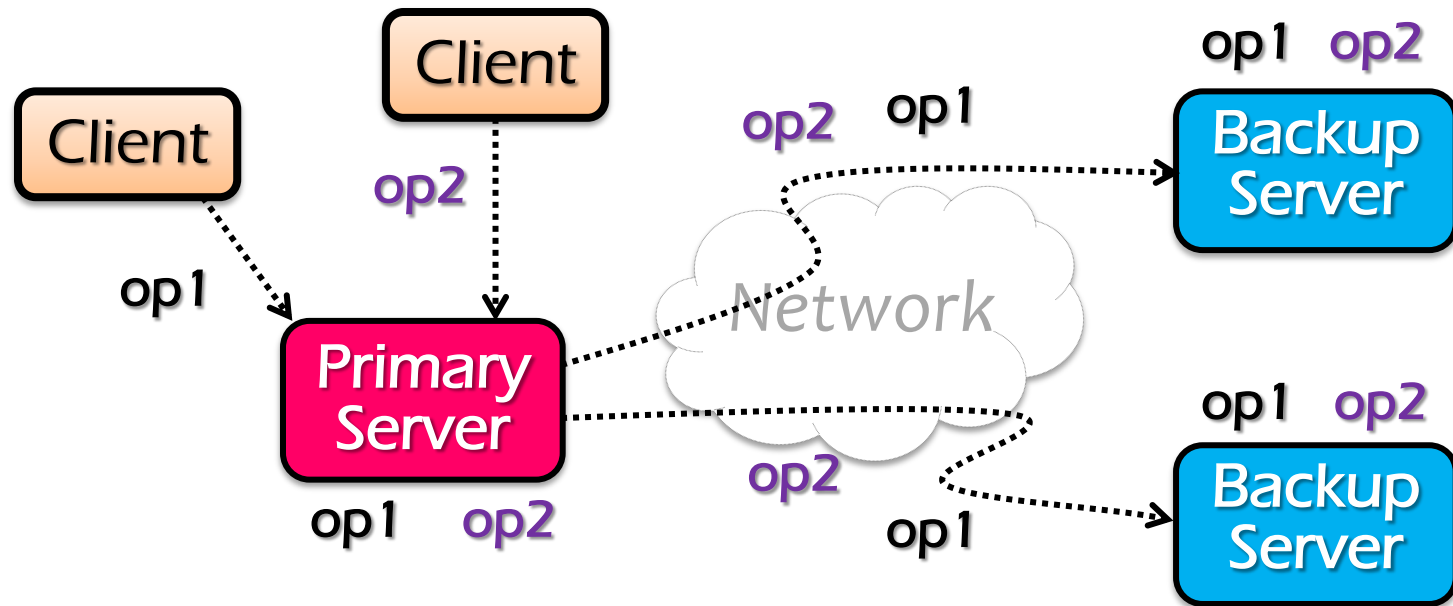


Does it ensure **sequential** consistency?

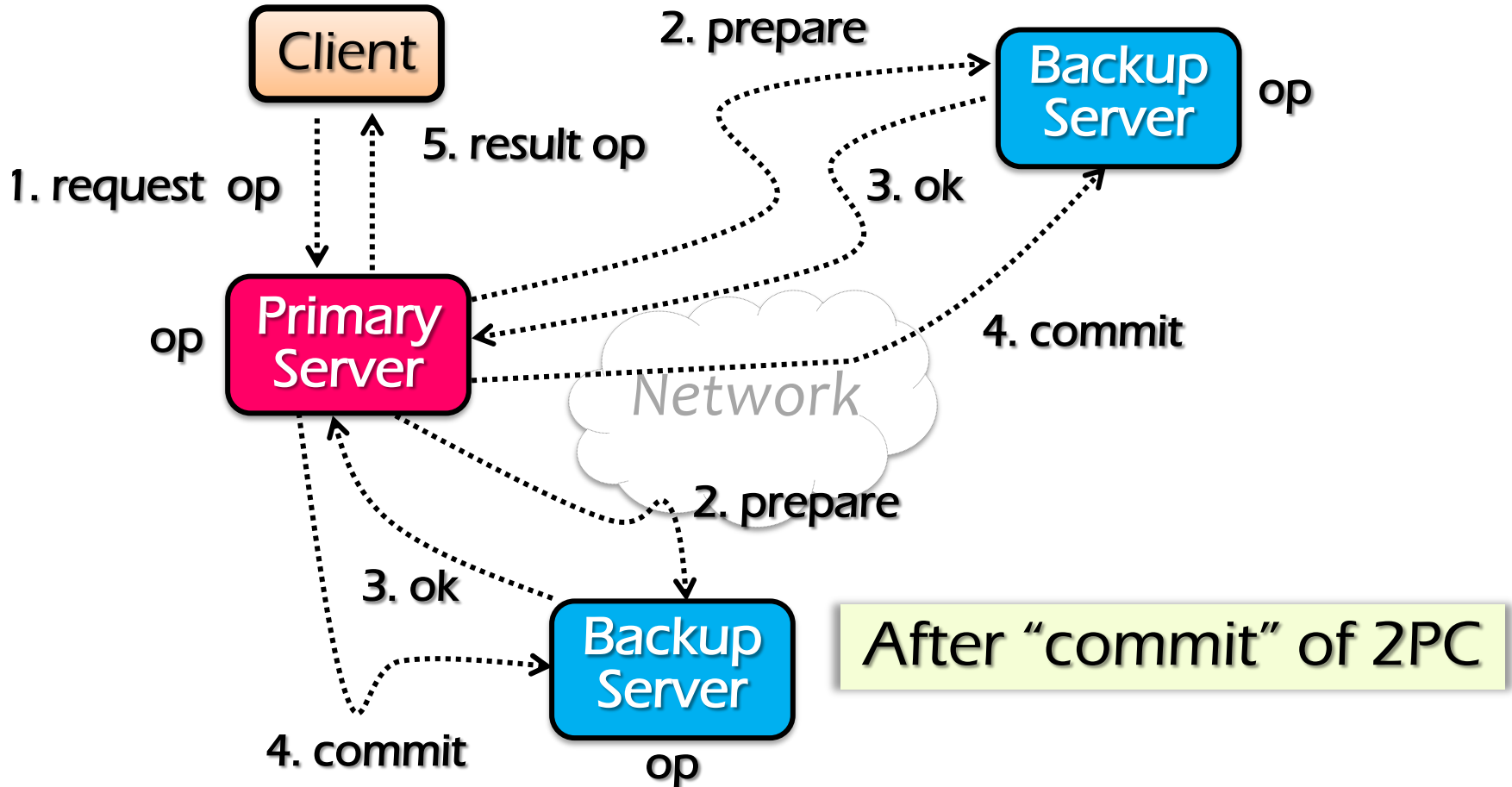
RSM based on Primary/Backup

Primary/Backup ensure a single order of ops

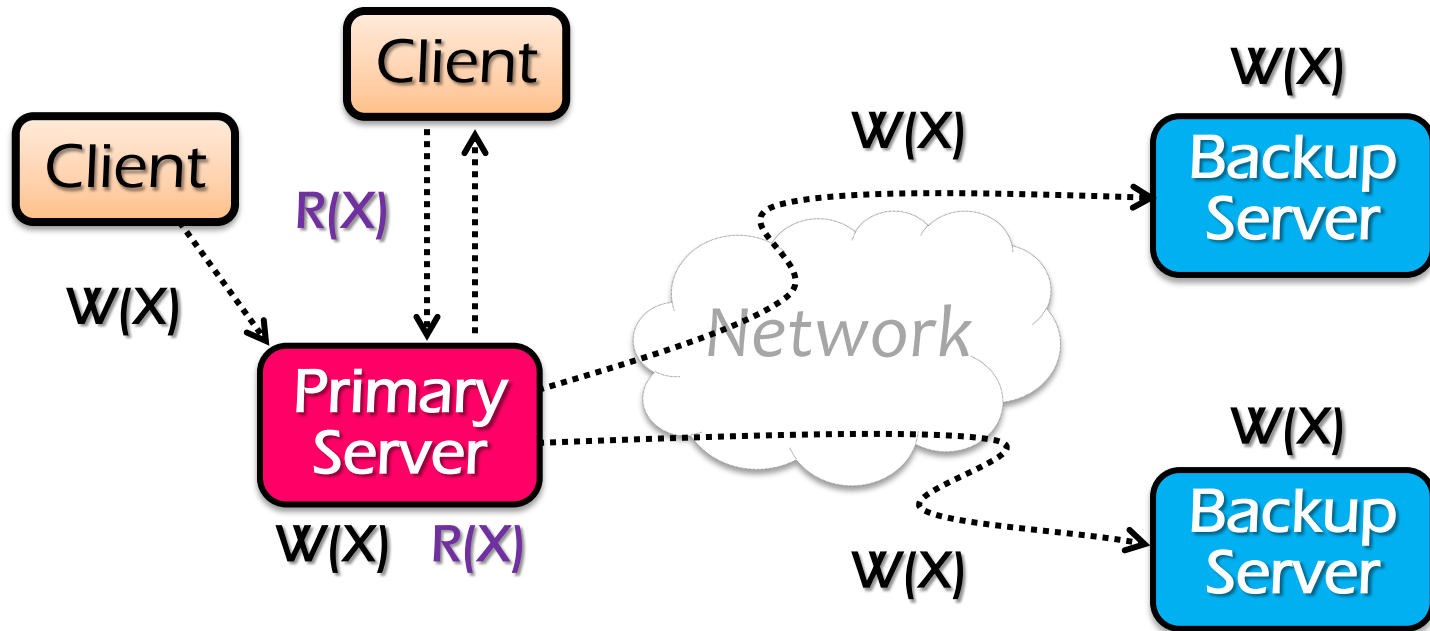
- Primary orders operations
- Backup execute operations in order



When does Primary Respond?

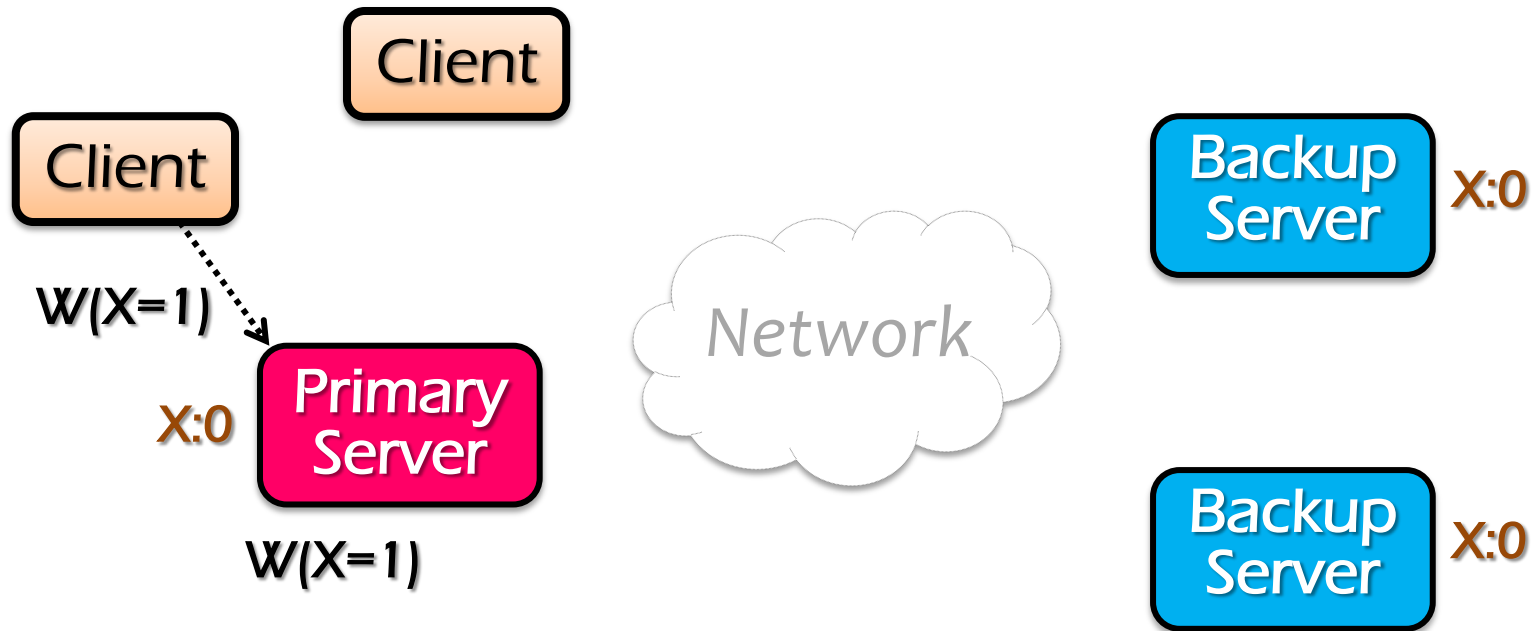


RSM: Read-only Operations



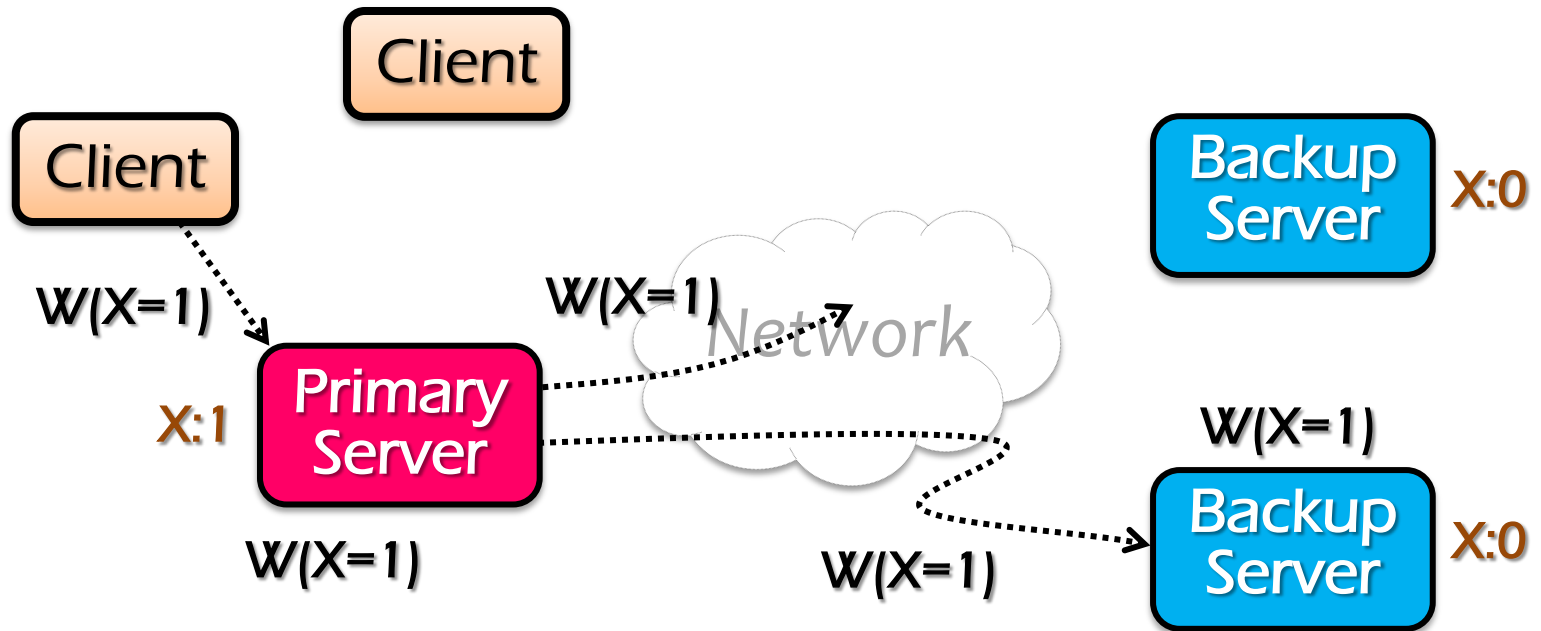
Read-only ops need not be replicated

RSM: Read-only Operations



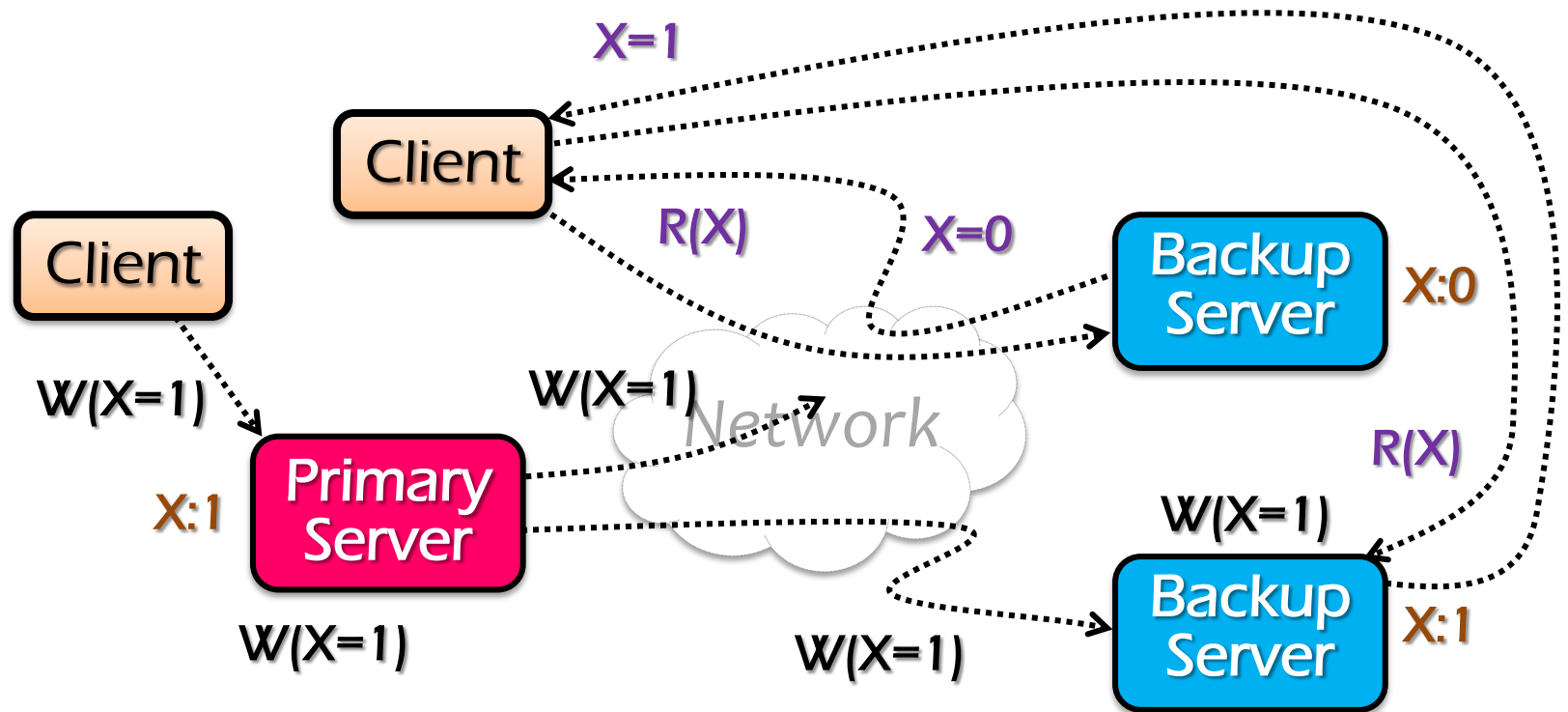
Can client send **read-only** ops to **any** server?

RSM: Read-only Operations



Can client send **read-only** ops to **any** server?

RSM: Read-only Operations



Can client send **read-only** ops to **any** server?

RSM: Failure Handling

If primary fails, one backup acts as the **new primary**

Challenges

1. How to reliably **detect** primary **failure**?
2. How to ensure no **two** backups simultaneously become new **primary**?
3. How to preserve sequential **consistency** across primary **changes**?

A fault tolerant distributed **consensus** protocol

Consensus

Allow a group of nodes to **agree** on a result

- All nodes must agree on the **same** value
- The value must be one that was **proposed** by **at least** one node (can't just make up a value)

Achieving **consensus** is **easy**!

- A system-wide **coordinator** determines outcome

BUT ... this **ASSUMES** there are **no failures**
... or we are willing to wait **indefinitely** for recovery

e.g., two-phase commit protocol → indefinite wait

Consensus Goal

Create an **algorithm** that does not **block** if a **majority** of processes are working

Goal: agree on one result among a group of participants

- Nodes may fail (may need stable storage)
- Messages: lost, out of order, or duplicated
- If delivered, messages are not corrupted

The **algorithm** needs **($2P+1$)** nodes survive the simultaneous failures of **P** nodes

Paxos

Paxos: fault-tolerant distributed consensus algorithm (the **only known**)

- All nodes agree on the **same** value despite node failure, network failure and delays

Extremely **useful**

- *e.g.*, Nodes agree that X is the primary
- *e.g.*, Nodes agree that OP1 should be the most recent operation executed

Requirements of Consensus

Correctness (safety)

- All nodes agree on the **same** value
- The **agreed** value X has been **proposed** by some node

Fault-tolerance

- If **minority** of nodes **fail**
→ the **rest** should still reach **agreement**

Termination

Impossibility of Consensus

There is **no way** to check whether a process **failed** or is **alive** but not communicating

It is **impossible** for a set of processors in an **asynchronous** system to agree on a binary value, even if only a **single** processor is subject to an unannounced **failure**.

– Fischer-Lynch-Paterson (FLP'85)

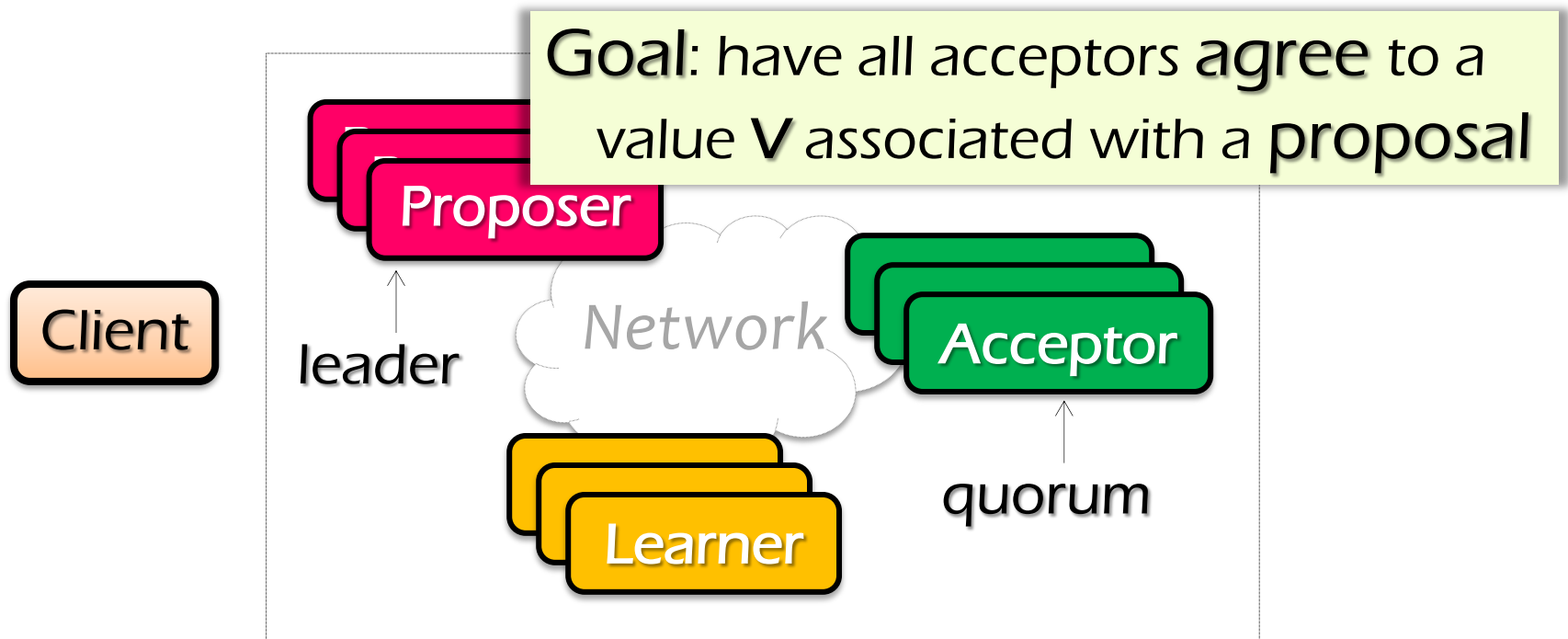
→ **Synchronous vs. Asynchronous** system

- Sync = responds to a message in a **bounded** time
- e.g., serial port transmission

Paxos

Paxos' properties: correct + fault-tolerance

- *No guaranteed termination*



Paxos: one machine may serve several roles

Paxos Players

Client

makes a request

Proposer

Get a request and run the protocol
Leader = elected **Coordinator**

Acceptor

Remember the state of the protocol
Quorum = any majority of **Acceptors**

Learner

When agreement has been reached,
a **Learner** executes the request and/or
sends a response back to the **Client**

General Approach

One proposer decides to be the leader

Leader proposes a value and solicits acceptance from **acceptors** (majority)

Leader announces result or try again

What if **>1** proposers become leaders simultaneously?

What if there is a **network partition**?

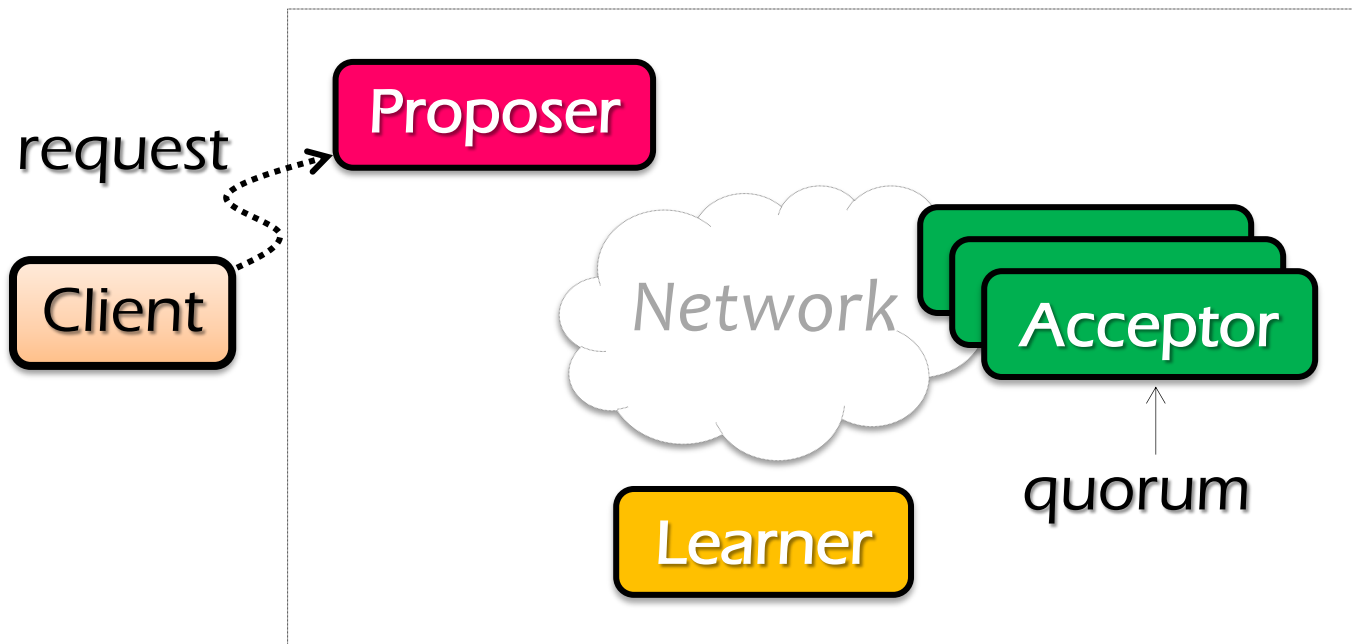
What if a leader **crashes** in the middle of solicitation?

What if a leader **crashes** after **deciding**
but before **announcing** results?

...

Paxos in Action: Phase 0

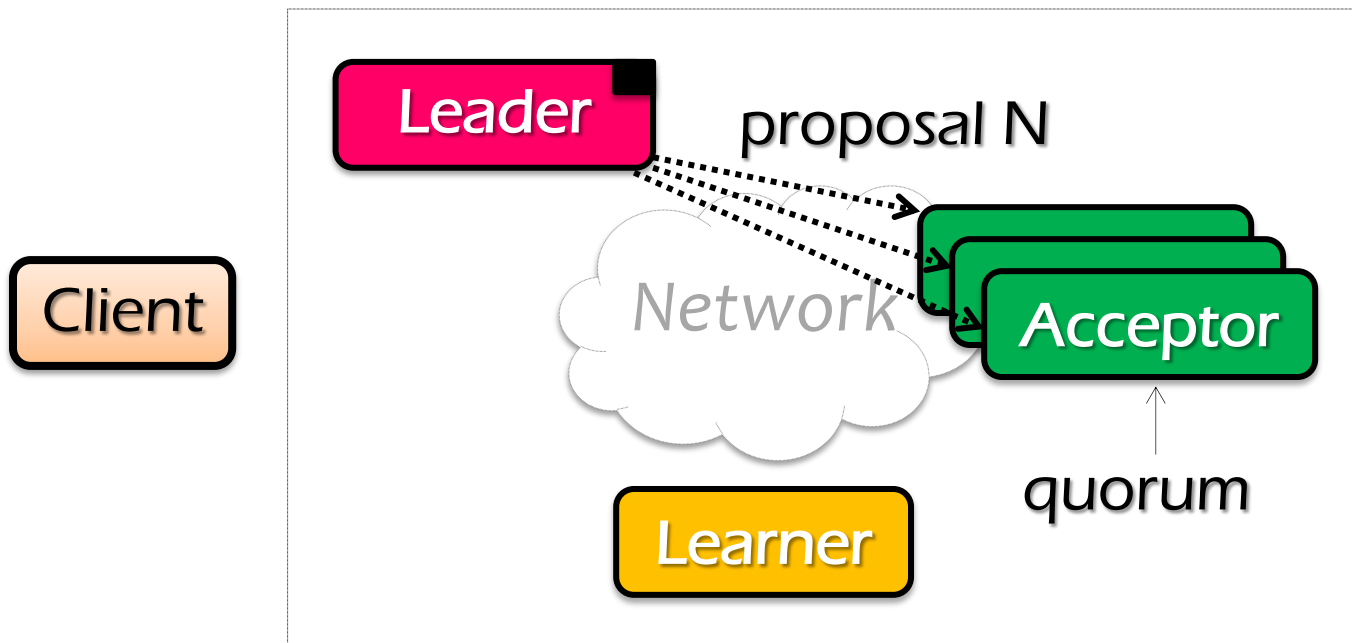
Client sends a request to a proposer



Paxos in Action: Phase 1a

Leader creates a proposal **N** and send to quorum

N is greater than **any** previous proposal number used by this proposer

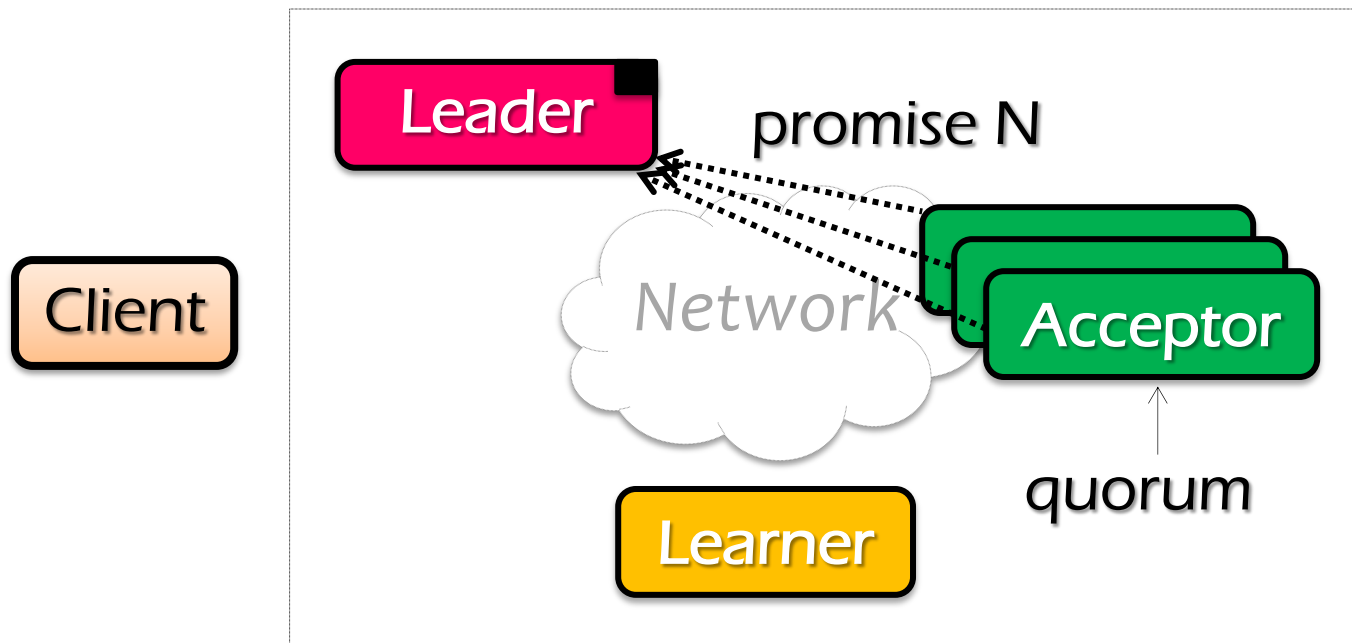


Paxos in Action: Phase 1b

Acceptor: **if** proposal **N** > **any** previous proposal

1. reply with **highest** past proposal # and value
2. promise to ignore all **IDs** < **N**

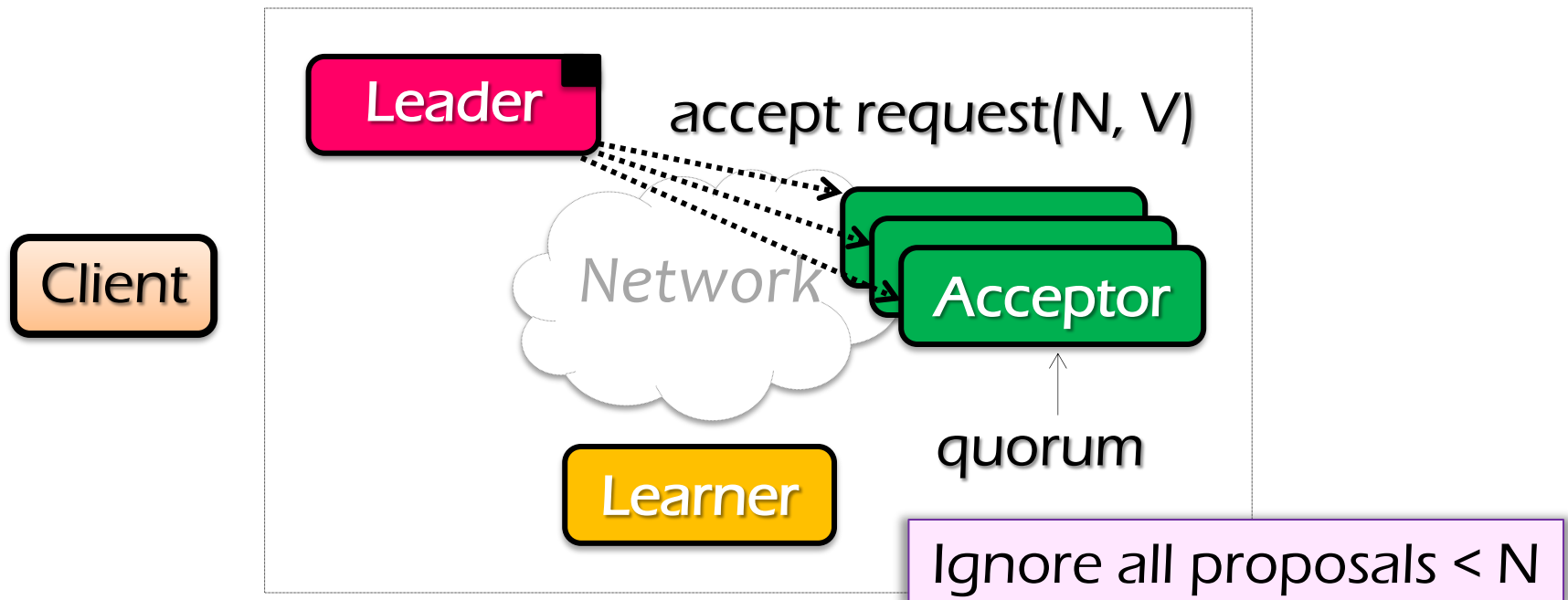
else ignore (proposal is **rejected**)



Paxos in Action: Phase 2a

Leader: **if** receive **enough** promise

1. set a value **V** to the proposal.
V can be **decided value** or **new value**
2. send **accept request** to quorum with the chosen value **V**



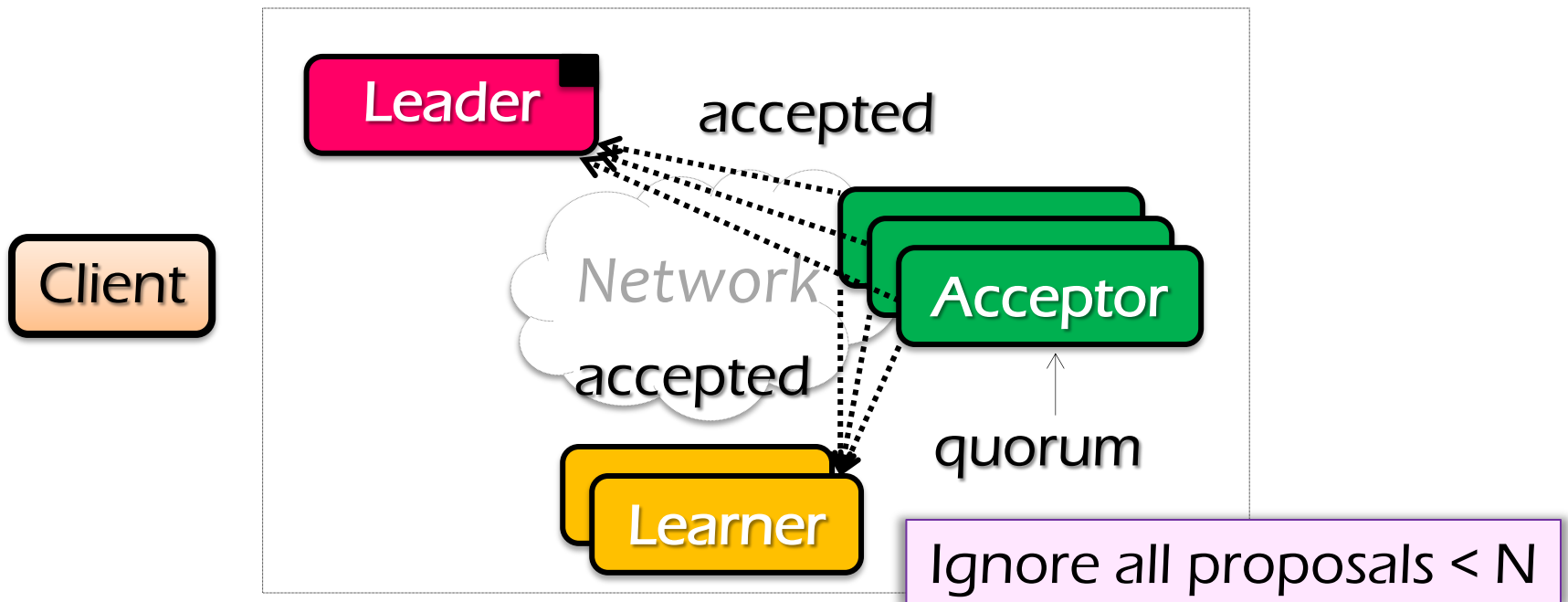
Paxos in Action: Phase 2b

Acceptor: **if** the promise still **holds**

1. register the value **V**

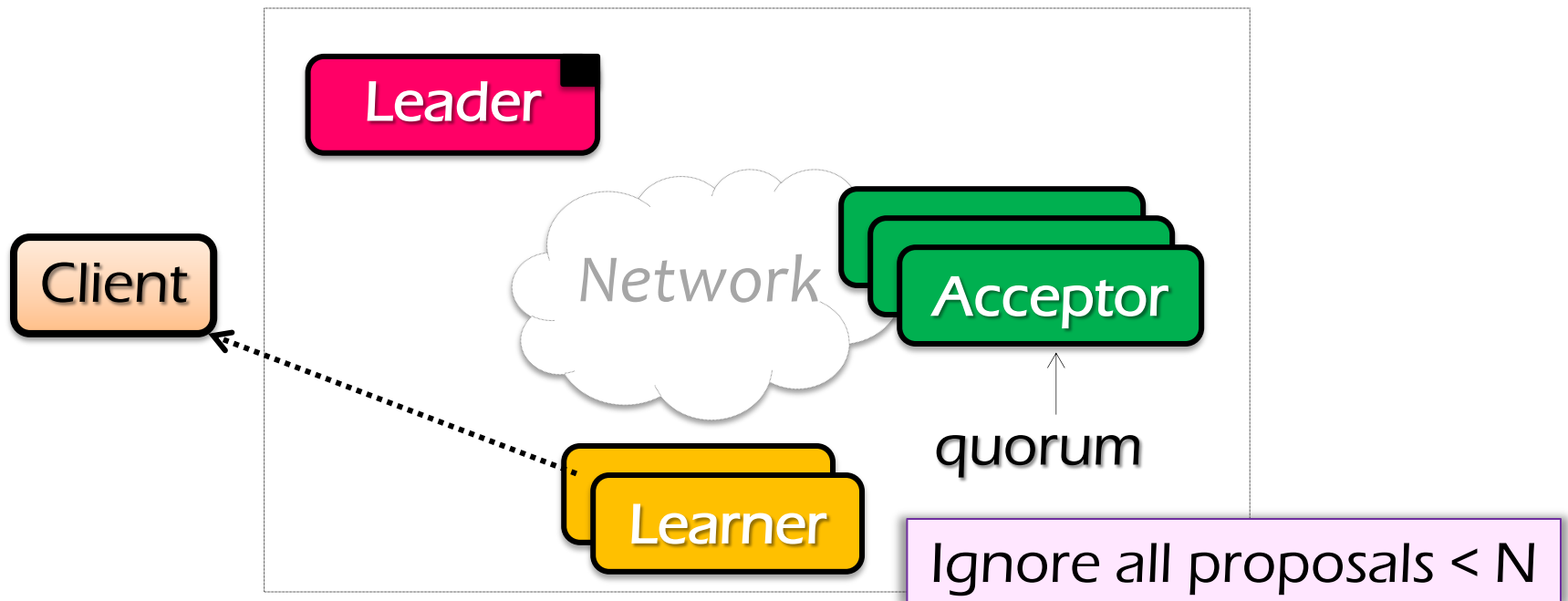
2. send **accepted message** to Proposer/Learners

else ignore the message



Paxos in Action: Phase 3

Learner: responds to Client and/or take action on the request



Paxos Setup



N_a : highest proposal # accepted
 V_a : accepted value of N_a
 N_n : highest proposal # seen
 M_n : my proposal #

each round of Paxos, each Node

Paxos Pseudo-code

A node decides to be Leader

Leader choose $M_n > N_h$

Leader sends $\langle \text{proposal}, M_n \rangle$ to all nodes

Acceptor receives $\langle \text{proposal}, N \rangle$

if $N < N_h$

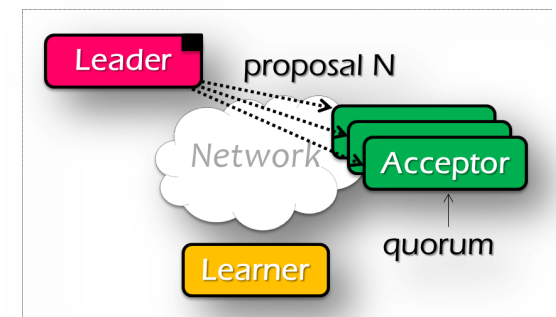
reply $\langle \text{promise-reject} \rangle$

else

$N_h = N$

reply $\langle \text{promise-ok}, N_a, V_a \rangle$

Ignore all proposals $< N_h$



Paxos Pseudo-code

If Leader gets promise-ok from a majority

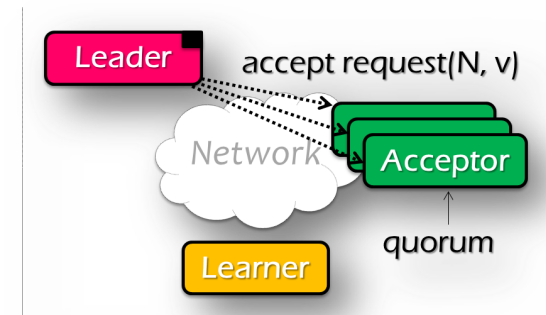
V = non-empty value (highest N_a received)
if $V = \text{null}$, then Leader can pick any V
send $\langle \text{accept}, M_n, V \rangle$ to all nodes

If Leader fails to get majority promise-ok

delay and restart Paxos

Upon receiving $\langle \text{accept}, N, V \rangle$

if $N < N_h$
 reply $\langle \text{accept-reject} \rangle$
else
 $N_a = N$; $V_a = V$; $N_h = N$;
 reply $\langle \text{accept-ok} \rangle$



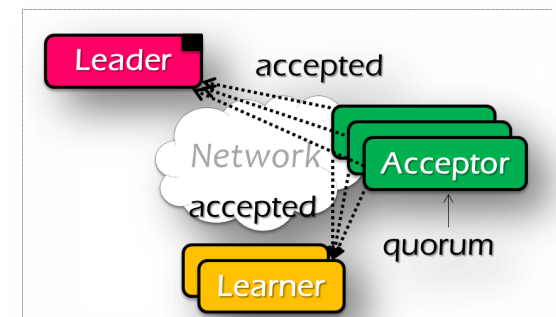
Paxos Pseudo-code

If Leader gets accept-ok from a majority

send $\langle \text{decide}, V_a \rangle$ to all nodes

If Leader fails to get majority accept-ok

delay and restart Paxos



Paxos Pseudo-code

$N_a = V_a = \text{null}$
 $N_h = N0:0$

$N_a = V_a = \text{null}$
 $N_h = N1:0$

$N_a = V_a = \text{null}$
 $N_h = N2:0$

$M_n = N1:1$

<proposal, N1:1>

<proposal, N1:1>

$N_h = \text{N1:1}$
 $N_a = \text{null}$
 $V_h = \text{null}$

<promise, null, null>

<promise, null, null>

$N_h = \text{N1:1}$
 $N_a = \text{null}$
 $V_h = \text{null}$

<accept, N1:1, V1>

<accept, N1:1, V1>

$N_h = \text{N1:1}$
 $N_a = \text{N1:1}$
 $V_h = \text{V1}$

<accept-ok>

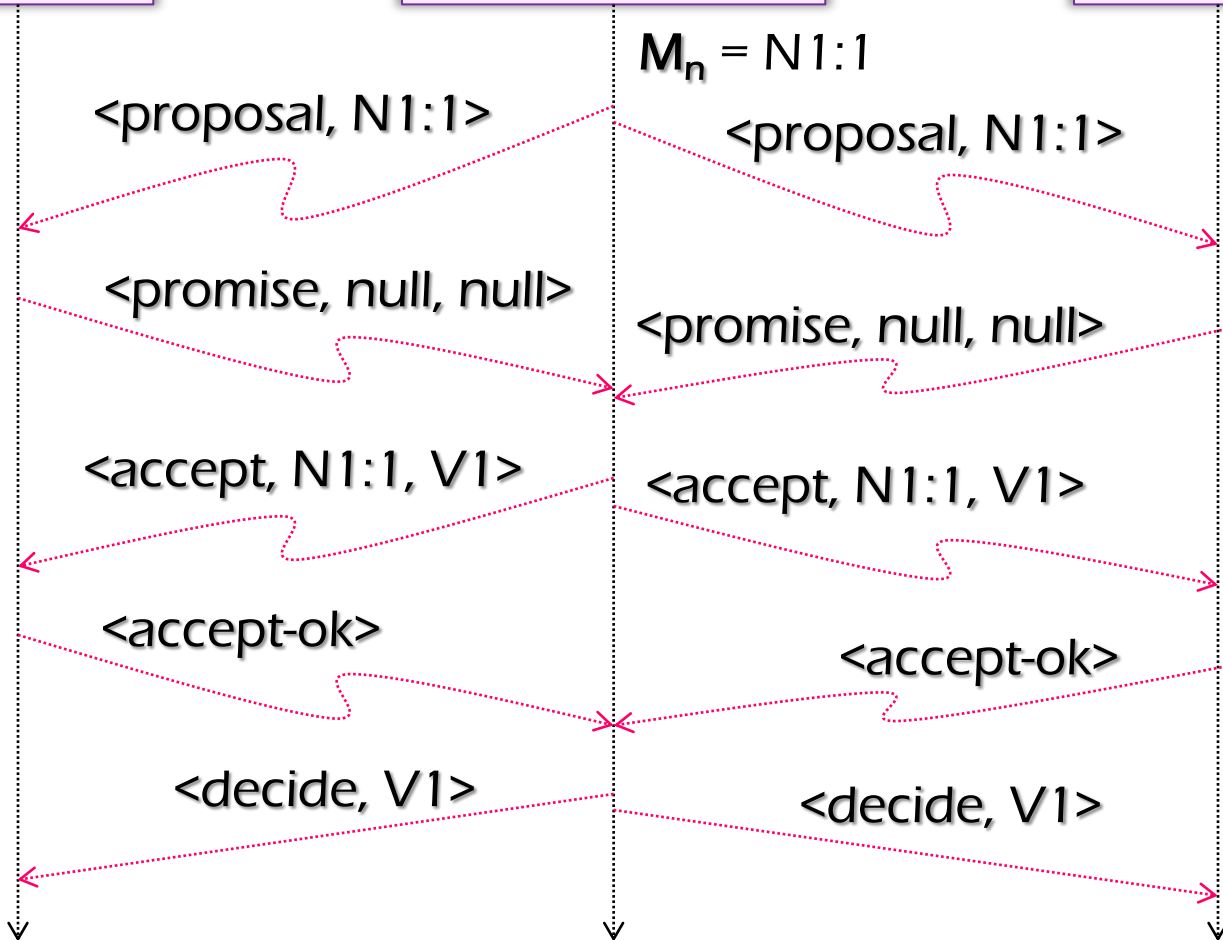
<accept-ok>

$N_h = \text{N1:1}$
 $N_a = \text{N1:1}$
 $V_h = \text{V1}$

<decide, V1>

<decide, V1>

Time



Inside of Paxos

Why setups multiple **acceptors**?

- Failure of the single acceptor halts decision

Why not accepts the **first** proposal and rejects the rest?

- Multiple leaders result in no majority accepting
- Leader dies

What if more than one leader is active?

- Can both leaders see a majority of promises?
- Then ... ?

Inside of Paxos

When is the value V chosen?

- Leader receives a majority $\langle \text{promise}, \dots \rangle$
- A majority acceptors receive $\langle \text{accept}, N, V \rangle$
- Leader receives a majority $\langle \text{accepted}, \dots \rangle$

What if acceptor **fails** after sending promise?

- Must remember N_h

What if acceptor **fails** after receiving accept?

- Must remember N_h and $N_a V_a$

What if leader **fails** while sending accept?

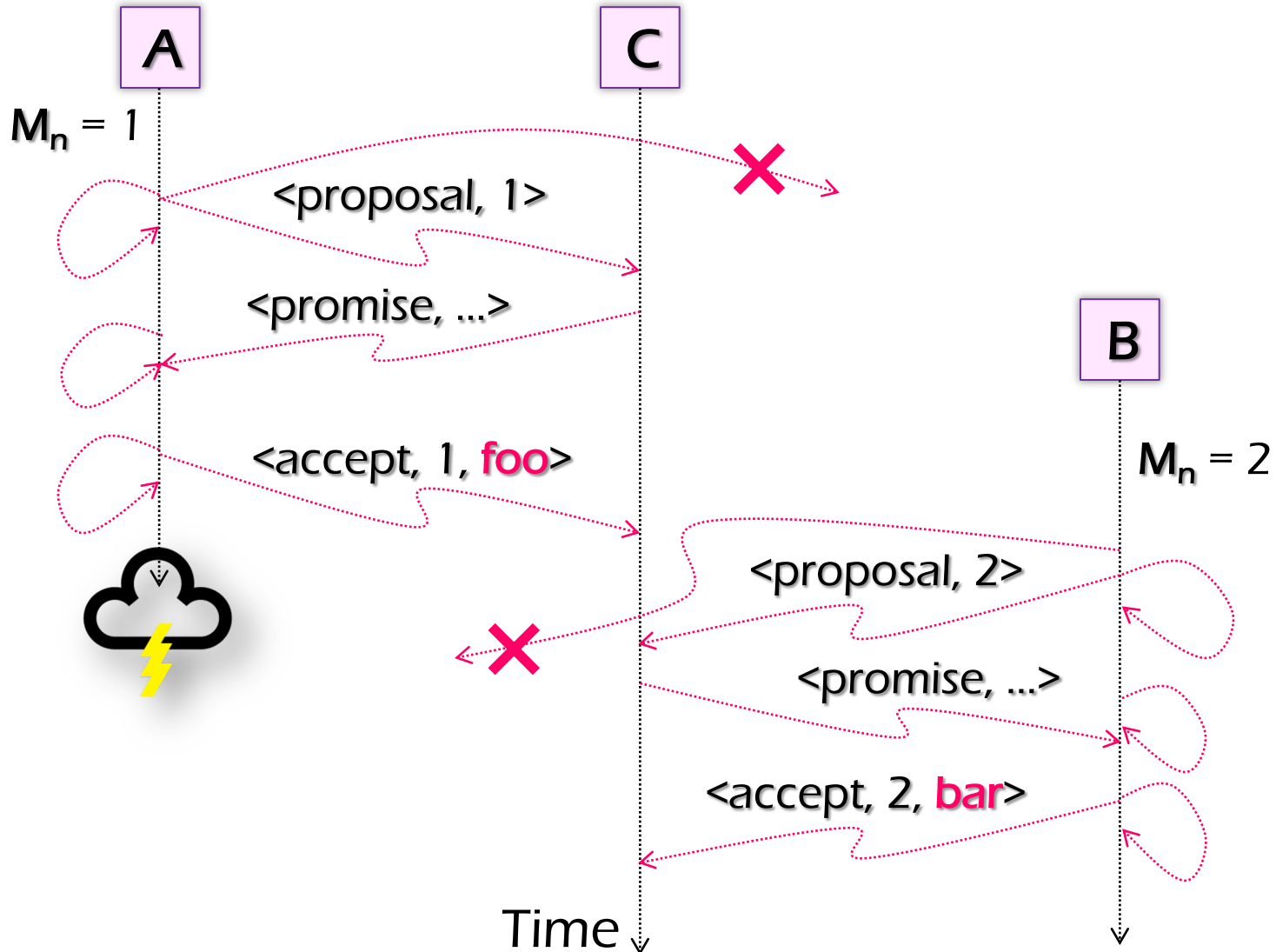
- Propose M_n again

Question

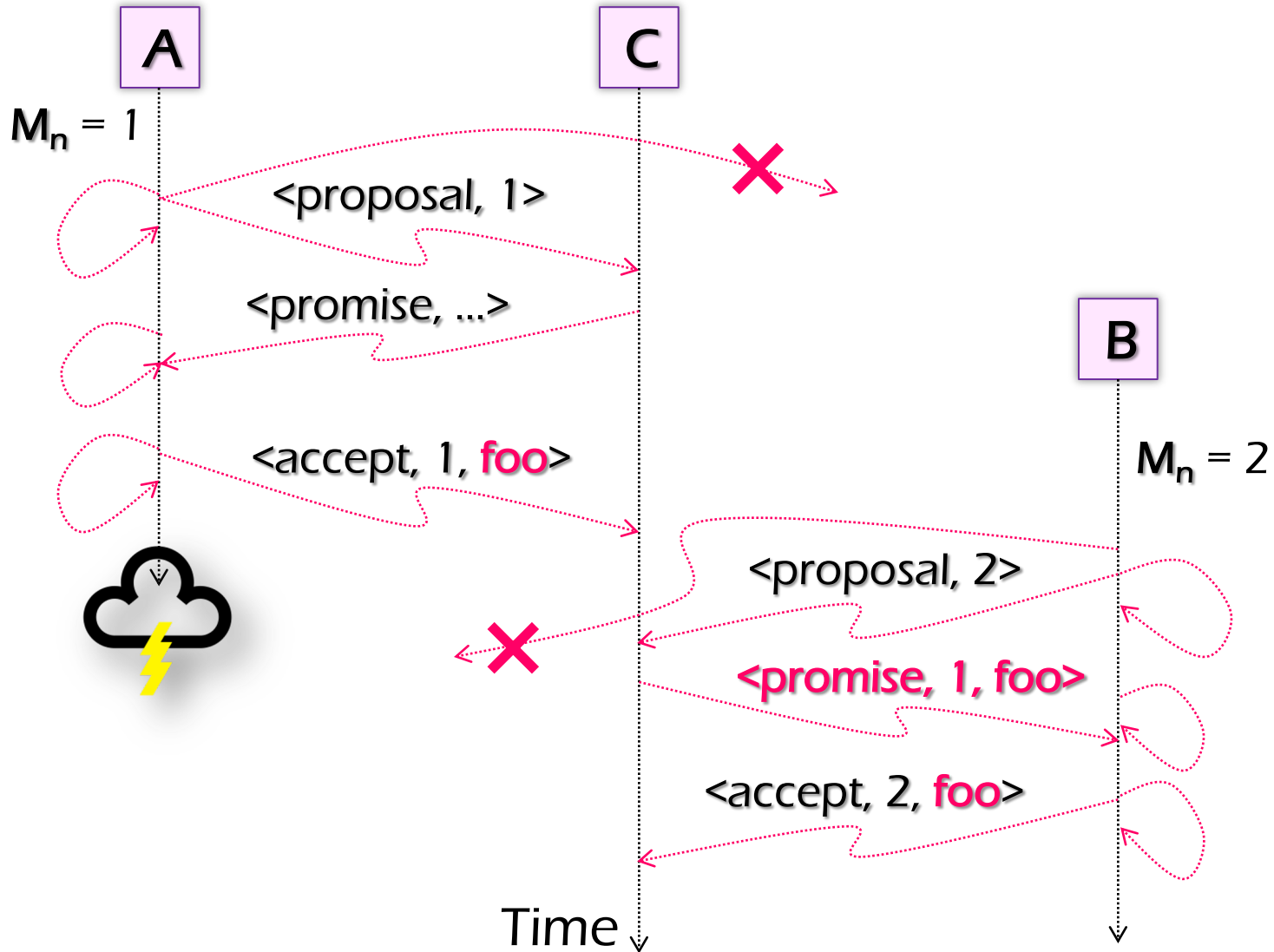
Suppose that the acceptors are **A**, **B**, and **C**. **A** and **B** are also proposers. How does Paxos ensure that the following sequence of events can't happen? What actually happens, and which value is ultimately chosen?

- A** sends **<prepare, 1>** requests with proposal number **1**, and gets responses from **A**, **B**, and **C**
- A** sends **<accept, 1, "foo">** to **A** and **C** and gets responses from both. Because a majority accepted, **A** thinks that **"foo"** has been chosen. However, **A** crashes before sending an **<accept, 1, "foo">** to **B**
- B** sends **<prepare, 2>** messages with proposal number **2**, and gets responses from **B** and **C**
- B** sends **<accept, 2, "bar">** messages to **B** and **C** and gets responses from both, so **B** thinks that **"bar"** has been chosen

Question



Question



Paxos Summary

Paxos allows us to ensure consistent (total) ordering over a set of **events** in a group of nodes

- Events = commands / actions / state updates

Each machine will have the **latest** state or a **previous** version of the state

Paxos Summary

To make a **change** to the system

1. Tell the **proposer(leader)** the **event**
(NOTE: these requests may occur concurrently)
2. The **leader** picks its next highest ID and asks proposal to all the **acceptors** with that ID
3. When the majority of **acceptors** accept the proposal, accepted **event** are sent to **learners**
4. The **learners** do event (e.g., update system state)

Paxos for RSM

Fault-tolerant RSM requires consistent replica view

- View: <primary, backups> (e.g., <node1, node2>)

All active nodes must agree on the sequence of view changes

- <vid-1, primary, backups>, ...

Use Paxos to agree on the <primary, backups>

- Each Paxos instance agree to a single view (e.g., <2, Node1, Node2>)

Next Lecture

Topic: Distributed File Systems

Paper

Sanjay Ghemawat et al, "***Google File System***".
International Symposium on Operating Systems
Principles (SOSP), **2003**

THANKS