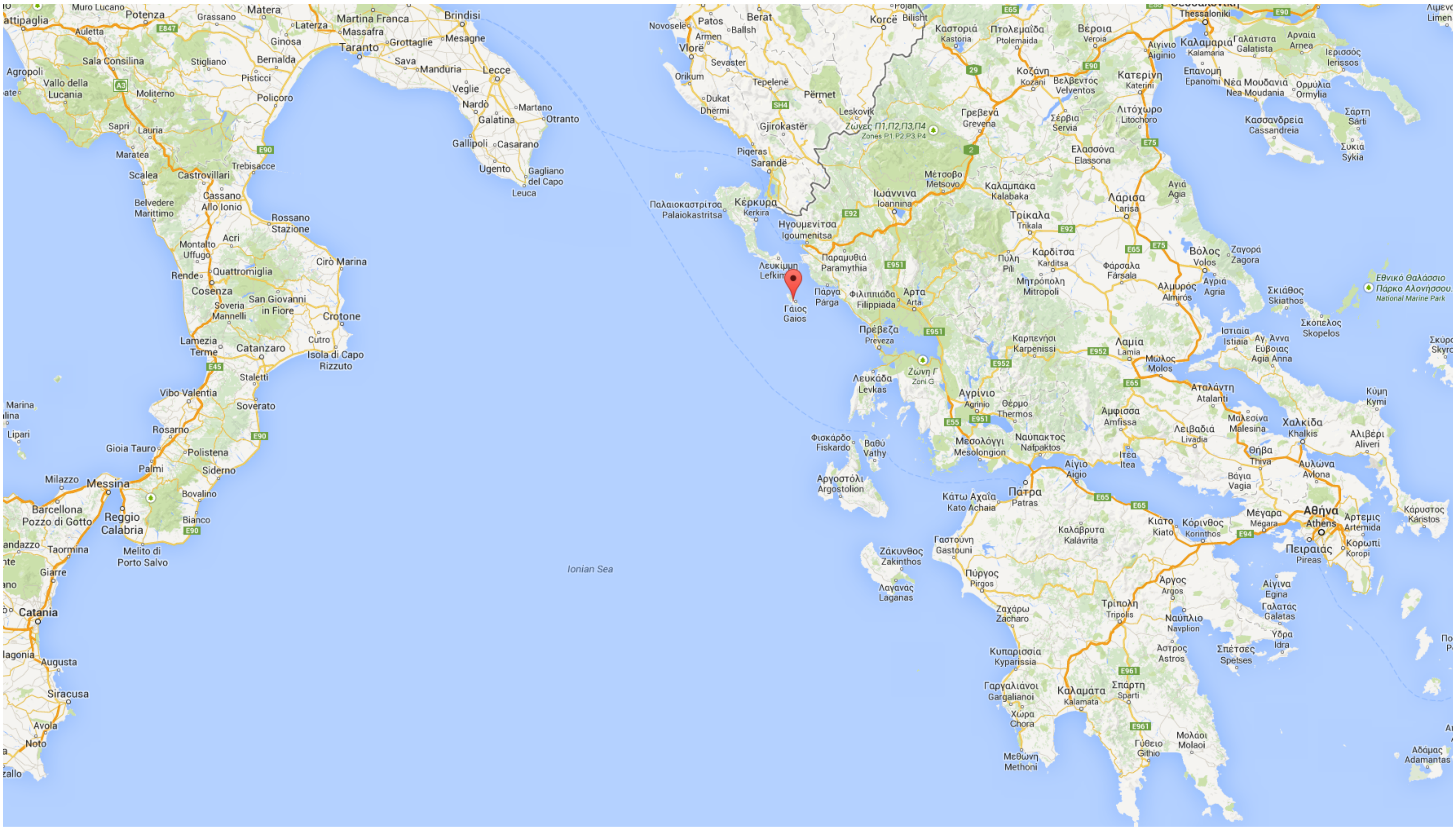
PAXOS


Γάιος
Gaios
Ionian Sea
Κέρκυρα
Kerkira
Λευκίμμη
Lefkimi
Πάργα
Parga
Πρέβεζα
Preveza
Ηγουμενίτσα
Igoumenitsa
Ιωάννινα
Ioannina
Λευκάδα
Levkas
Αργοστόλι
Argostolion
Ζάκυνθος
Zakinthos
Πάτρα
Patras
Αθήνα
Athens
Λάρισα
Larisa
Βόλος
Volos
Καλαμάτα
Kalamata
Τρίπολη
Tripolis
Brindisi
Lecce
Taranto
Otranto
Sarandë
Vlorë
Gjirokastër
Messina
Reggio Calabria
Catanzaro
Cosenza
Crotone
Catania
Siracusa

PAXOS

- Leslie Lamport. **The part-time parliament**. ACM Transactions on Computer Systems, 16(2):133–169, May 1998.
- Leslie Lamport described in 1990 the algorithm as the solution to a problem of the parliament on a fictitious Greek island called Paxos (not Italy)
- Many readers were so distracted by the description of the activities of the legislators, they did not understand the meaning and purpose of the algorithm. The paper was rejected.
- Leslie Lamport refused to rewrite the paper. He later wrote that he "was quite annoyed at how humorless everyone working in the field seemed to be"
- After a few years, some people started to understand the importance of the algorithm
- After eight years, Leslie Lamport submitted the paper again, basically unaltered. It got accepted!

Leslie Lamport
ACM Turing Award 2014

Consensus (again)

- We have a collection of processes
- Each process can propose a value
- A single value among the proposed values is chosen
- If no value is proposed, no value should be chosen
- If a value has been chosen, processes should be able to learn the chosen value

System Model

- <u>Asynchronous</u> communications
- <u>Non-byzantine</u> failures:
  - Nodes crash
  - Messages can be lost, duplicated arbitrarily late
  - No corrupted messages
- <u>Fail-recover</u>: crashed nodes may recover later

Roles

- Each nodes has one or more of 3 roles
- Proposer: a node that can propose a certain value for acceptance
- Acceptor: a node that receives proposals from proposers and that can either accept or reject a proposal
- Learner: a node not involved in the decision process that wants to know the final result of the decision process
- We will assume all nodes act both as proposer and as acceptor

Acceptors

- A single acceptor (a.k.a. coordinator) can fail and block the whole procedure

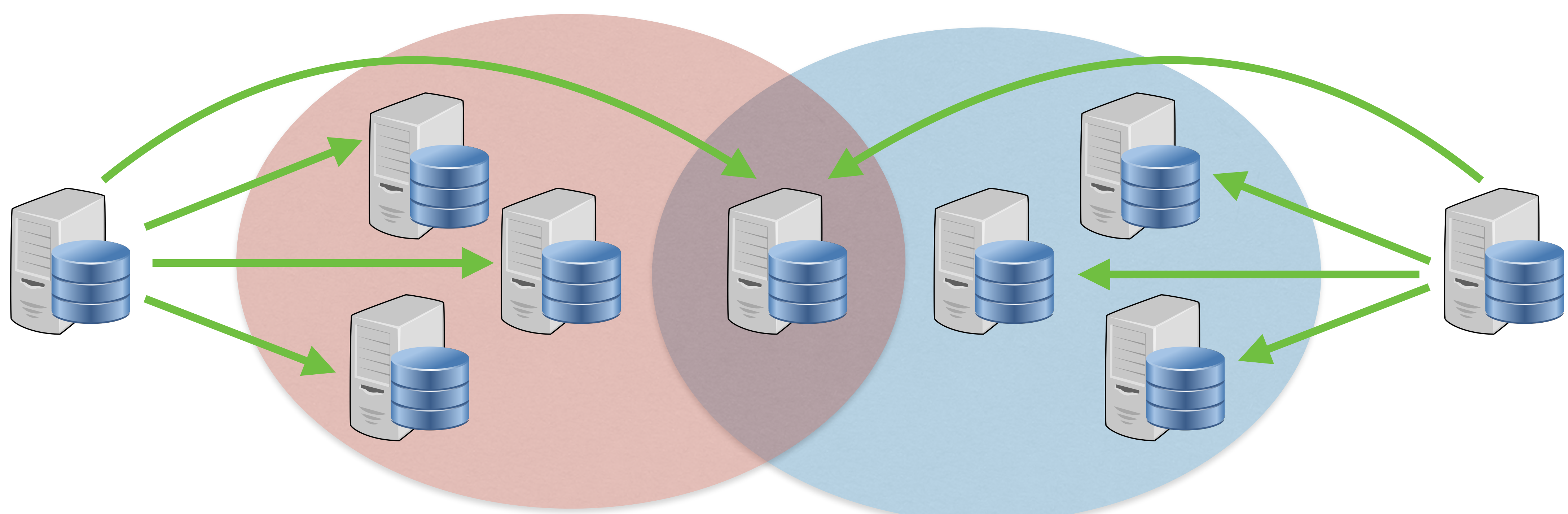
- There is not a coordinator, but multiple acceptors.

- An acceptor may **accept** a single value (e.g., express a single vote)

- How many acceptors do we need?

Majority

- To ensure that a single value is chosen, any majority (50% +1) of acceptors is enough
- The intersection of two majorities is not empty
- An acceptor may **accept** a single value
- A value is **chosen** when a majority of acceptors has accepted it
- If a majority choses a value, no other majority can chose a different value
- If an acceptor crashes, chosen value still available



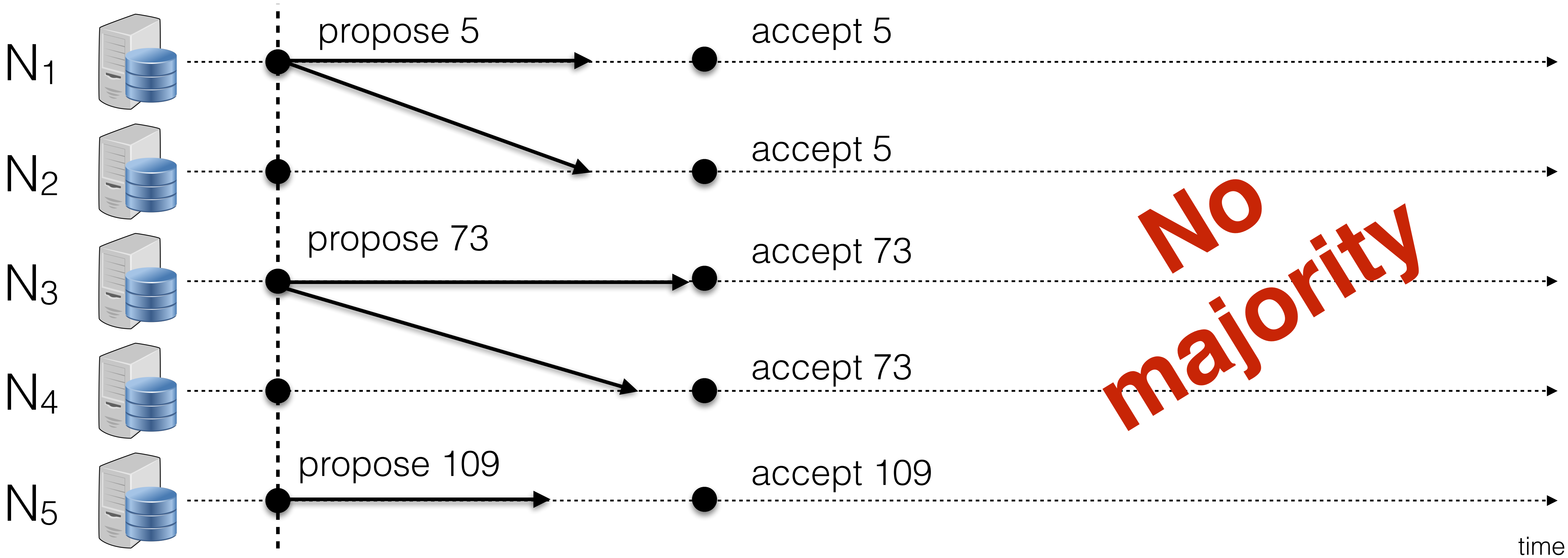
Acceptors

Since there can be a single proposer, the algorithm must guarantee that

P1: An acceptor must accept the first proposed value it receives

Since there can be a single proposer, the algorithm must guarantee that a single value is chosen only when it is accepted by a majority of acceptors

Split Votes



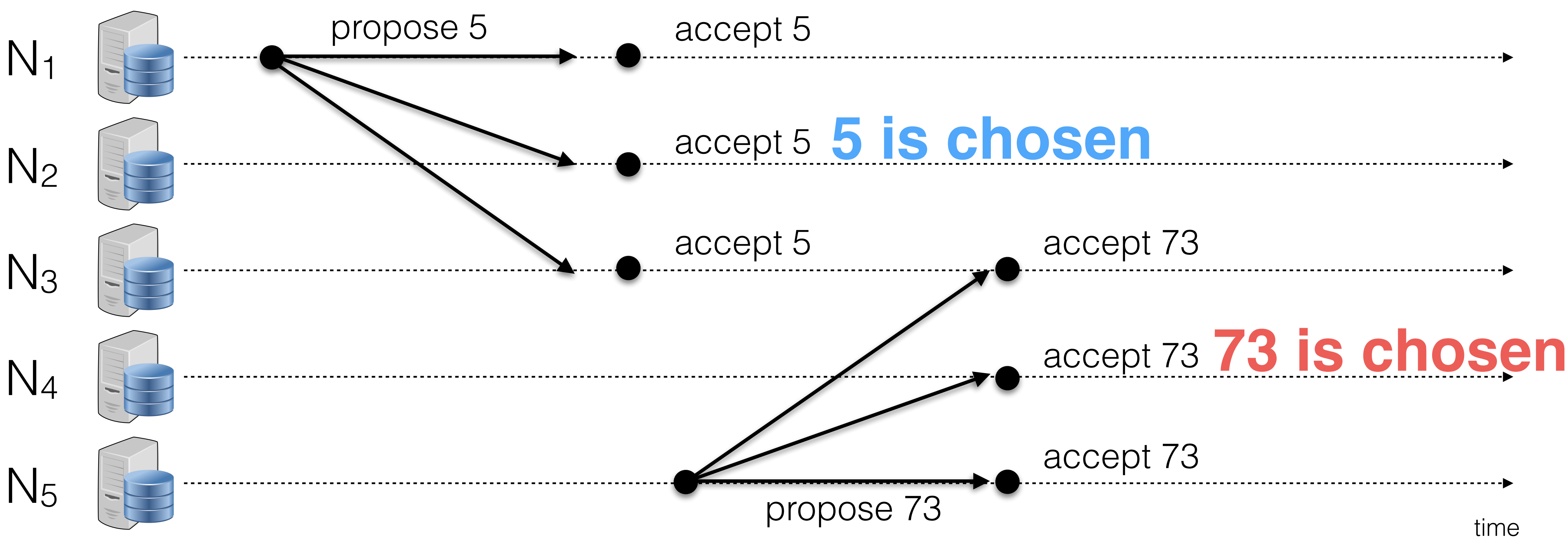
An acceptor must accept the first proposed value it receives

(and can accept more than one proposed value)

An acceptor can **accept** any number of proposed values, but an accepted proposed value may not necessarily be **chosen**.

Conflicting choices

Should acceptors accept every proposed value? No

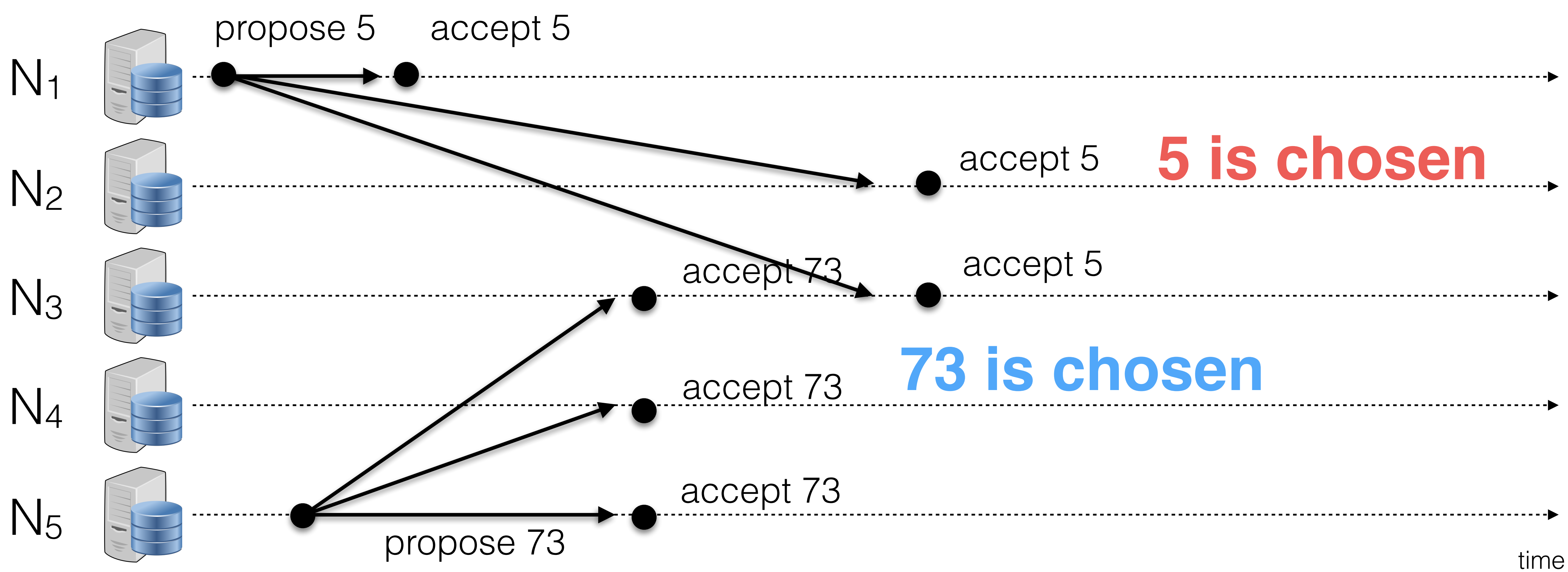


Once a value has been **chosen**, proposers must **propose** that same value

We need a two-phase protocol

Slides based on https://ramcloud.stanford.edu/~ongaro/userstudy/

Conflicting choices



Once a value has been **chosen**, older proposed values must be ignored

We need to order proposed values and reject old ones

Proposals

- A **proposal** *(v, n)* consists in the **proposed value** *v* and a **proposal number** *n*
- Whenever a proposer issues a new proposal, it chooses a strictly-increasing proposal number
- For a given proposer, a new proposal number must be greater than anything it has seen/used before
- Simple implementation

ROUNDNUMBER	NODEID

- Each server stores MAXROUND, i.e., the largest round number it has seen so far
- To generate a new proposal number: increment MAXROUND and concatenate with NODEID
- Proposers must persist MAXROUND on disk: must not reuse proposal numbers after crash/restart

Slides based on https://ramcloud.stanford.edu/~ongaro/userstudy/

Phase 1: PREPARE

- A proposer broadcasts a *prepare proposal (v, **n**)* with its own proposal value *v* and proposal number ***n***
- An acceptor receives a *prepare proposal (v, **n**)*:
    - if it has never received a *prepare proposal*:
        - it *promises* to never accept *proposal numbers* lesser than ***n*** (n_{min} = ***n***)
        - it returns (∅, *0*);
    - otherwise it checks its *promise*:
        - if ***n*** > n_{min} it sends its last *accepted proposal* (v_{last}, n_{last}); note that n_{last} < ***n***.
        - otherwise it does nothing

Phase 2: PROPOSE

- When a proposer receives a majority of responses, it broadcast to them a *propose proposal ($\boldsymbol{v'}$, n)*
    - If all acceptors returned ($\varnothing$, 0), $\boldsymbol{v'}$ is its own proposal value v
    - Otherwise $\boldsymbol{v'}$ is the v_{last} proposal value in the returned proposals with the greatest proposal number n_{last}
- If a majority of acceptors replies with **ACK**, the proposal is chosen
- Note that after a timeout, the proposer gives up and may send a new proposal

PAXOS Algorithm

Proposers	Acceptors

0) $n_p = 0$ highest prepare number seen
$n_a = 0$ highest accepted proposal number
$v_a = \varnothing$ highest accepted proposal value

These value must stably persist on disk

1) Choose new proposal number $\mathbf{n} > n_p$

2) Broadcast PREPARE(v, $\mathbf{n}$) to all nodes → 3) Handle PREPARE(v,$\mathbf{n}$):
   If ($\mathbf{n} > n_p$) then
      $n_p = \mathbf{n}$
      REPLY(v_a, n_a)

4) If REPLY(v_a, n_a) from majority: ←
   $\mathbf{v'} = v_a$ with greatest n_a
   If $\mathbf{v'} = \varnothing$ then
      $\mathbf{v'} = v$

5) Broadcast PROPOSE($\mathbf{v'}$, n) to all nodes → 6) Handle PROPOSE($\mathbf{v'}$, n):
   If ($n \geq n_p$) then
      $n_p = n$
      $(v_a, n_a) = (\mathbf{v'}, n)$
      ACCEPT($\mathbf{v'}$, n)

7) If ACCEPT($\mathbf{v'}$, n) from majority: ←
   Broadcast DECIDED($\mathbf{v'}$) to all nodes

Learning a decision

- After a proposal is chosen, only the proposer knows about it!
- How do the other nodes get informed?
  1. The proposer could inform all nodes directly
     - If the proposer fails, the others are not informed (directly)...
  2. The acceptors could broadcast every time they accept a proposal
     - Much more fault-tolerant
     - Many accepted proposals may not be chosen...
     - Moreover, choosing a value costs $O(n^2)$ messages without failures!
  3. The proposer could inform some nodes directly
     - They will broadcast the decision to other nodes

PAXOS is safe!

If a proposal *(v, n)* is **chosen**, then for every proposed proposal *(u, m)* for which $m > n$ it holds that $v = u$

- Assume that there is a proposed proposal *(u, m)* for which $m > n$ and $u \neq v$. Consider such a proposal with the smallest *m*.

- Consider the non-empty intersection *S* of the two majority sets of nodes that are acceptors for *(v, n)* and *(u, m)* proposals.

- Since proposal *(v, n)* has been accepted and $m > n$, nodes in *S* must have received PREPARE(*u, m*) after (*v, n*) has been accepted, thus returning REPLY(*v, n'*), with $n \leq n' < m$.

- As a consequence, the proposer of *(u, m)* should propose *(v, m)*, hence $u = v$, that is a contradiction.

PAXOS is correct!

If a value is chosen, all acceptors choose this value

- Once a proposal *(v, n)* is chosen, each following proposal *(u, m)* has the same proposal value, i.e., $u = v$, according to the previous theorem.

- Since every following proposal has the same value *v*, every proposal that is accepted after *(v, n)* is chosen will have the same proposal value *v*.

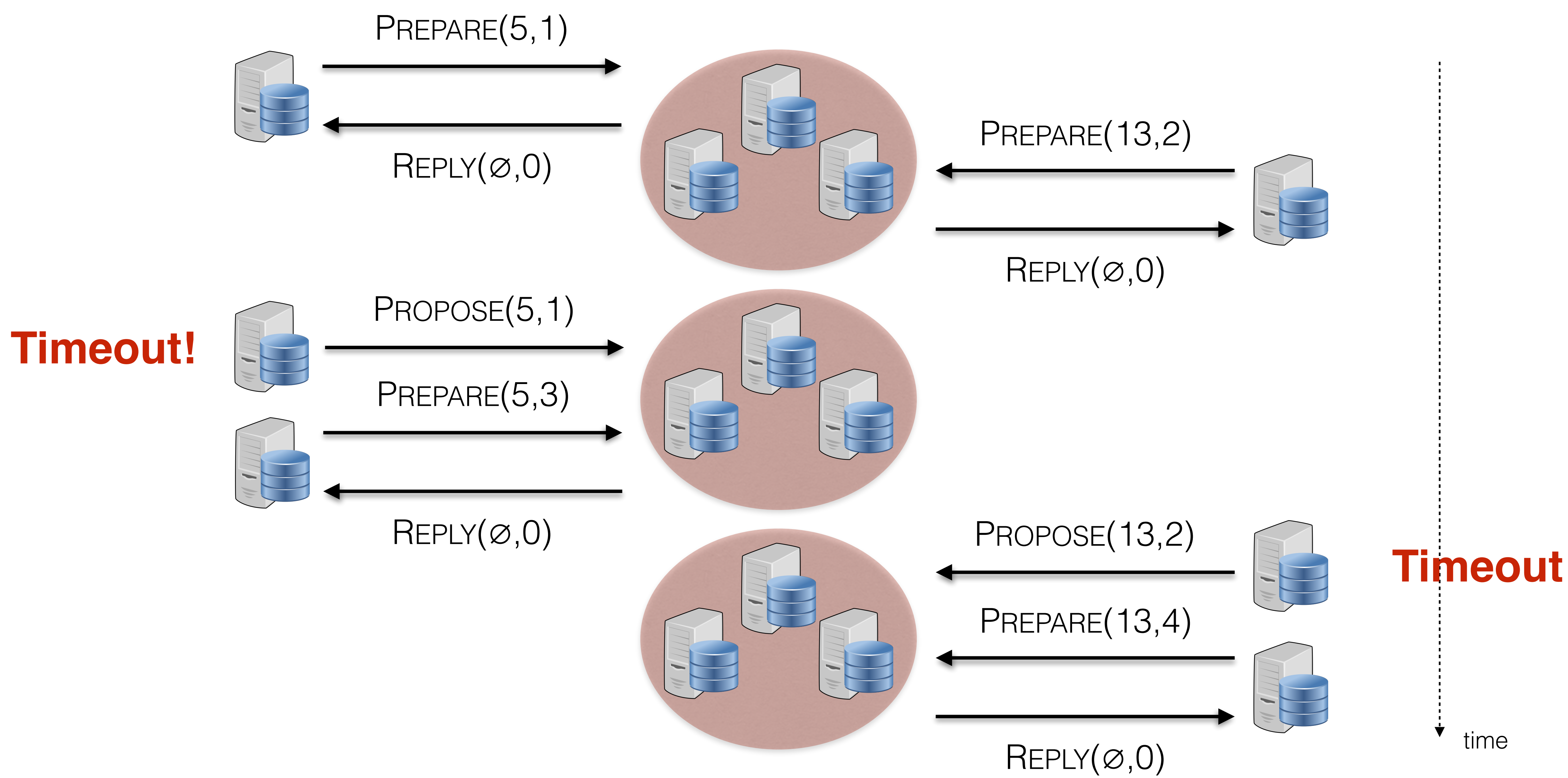
- Since no other value than *v* can be accepted, no other values can be chosen.

PAXOS is great?

- PAXOS is a deterministic algorithm working for asynchronous systems and tolerating ***f < n/2*** failures.
- Many optimizations exists (Multi-PAXOS, Disk-PAXOS, RAFT)
- **FLP Theorem** [1985]. No totally correct consensus algorithm exists (for the given system model).
- PAXOS disproves the FLP Theorem?

No Liveness Guarantees

- PAXOS only guarantees that if a value is chosen, the other nodes can only choose the same value
- PAXOS does not guarantee that a value is chosen!



Correctness vs. Termination

- In asynchronous systems, we cannot guarantee termination and correctness at the same time
- PAXOS is correct, so termination is not guaranteed
- PAXOS cannot guarantee that a consensus is reached in a finite number of steps
- In practice, PAXOS can be optimized to reduce probability of no termination
- For example, the acceptors could send NAK if they do not accept a prepare message or a proposal (this optimization increases the message complexity)
- PAXOS is used in Apache’s Zookeeper and Google’s Chubby