Total-Ordering

vanilladb.org

Why Paxos?

- Flooding consensus algorithm spends too much time waiting for the last message in every round
  - On WAN, this largely increases the response time
- Paxos: why not skip the late messages and make them insignificant to decision?
  - Idea: consensus can be reached by a ***majority*** of nodes

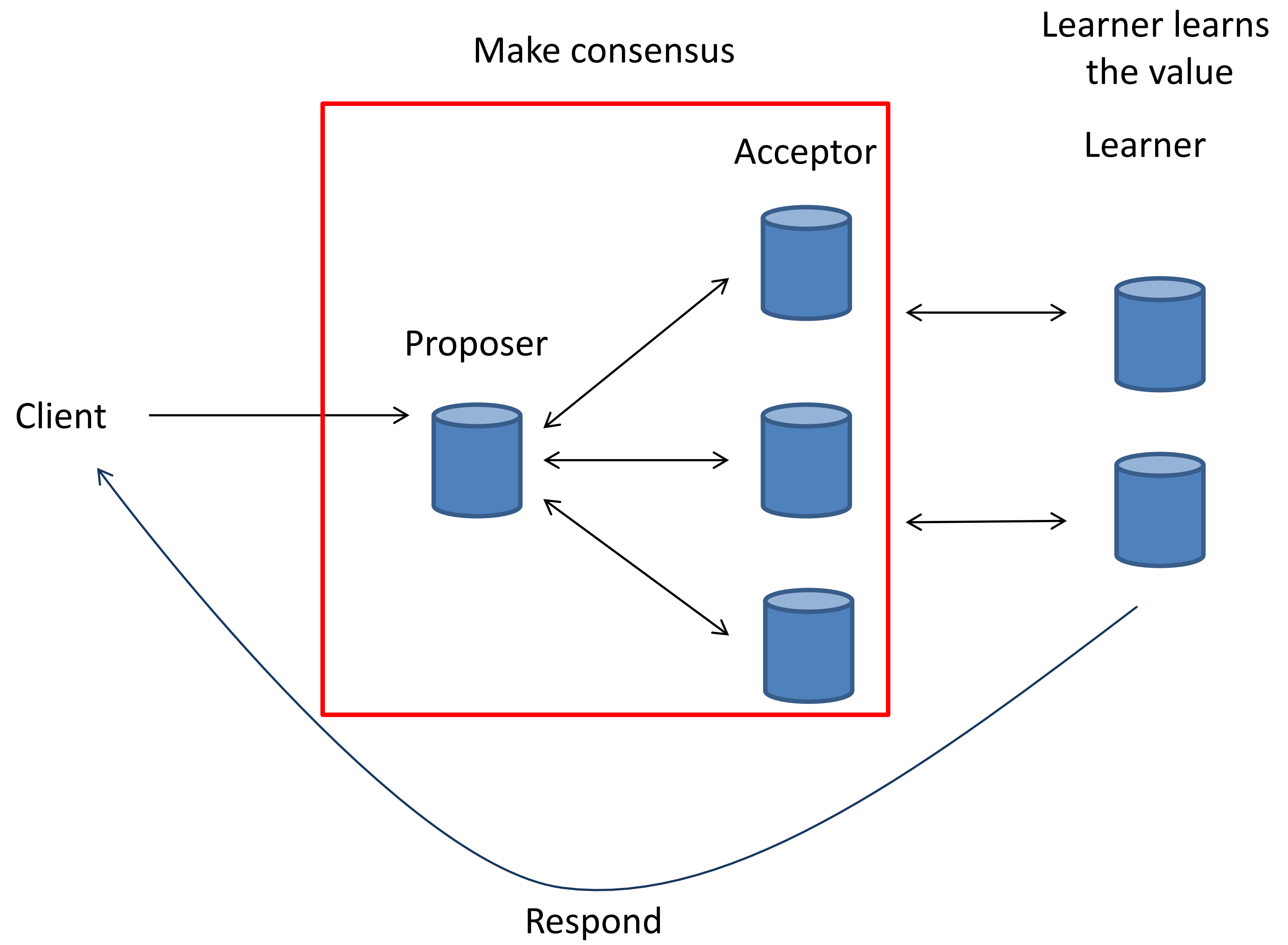
The Goal of Paxos

- In a Paxos run, the protocol should
  - Ensure a proposed value is eventually chosen, and correct nodes can eventually learn the value
- More precisely, the protocol should meet the following safety requirements
  - If a node decides a value *v*, then *v* was proposed by some nodes.
  - Only a single value is eventually chosen
  - A node never learns that a value has been chosen unless it actually has been

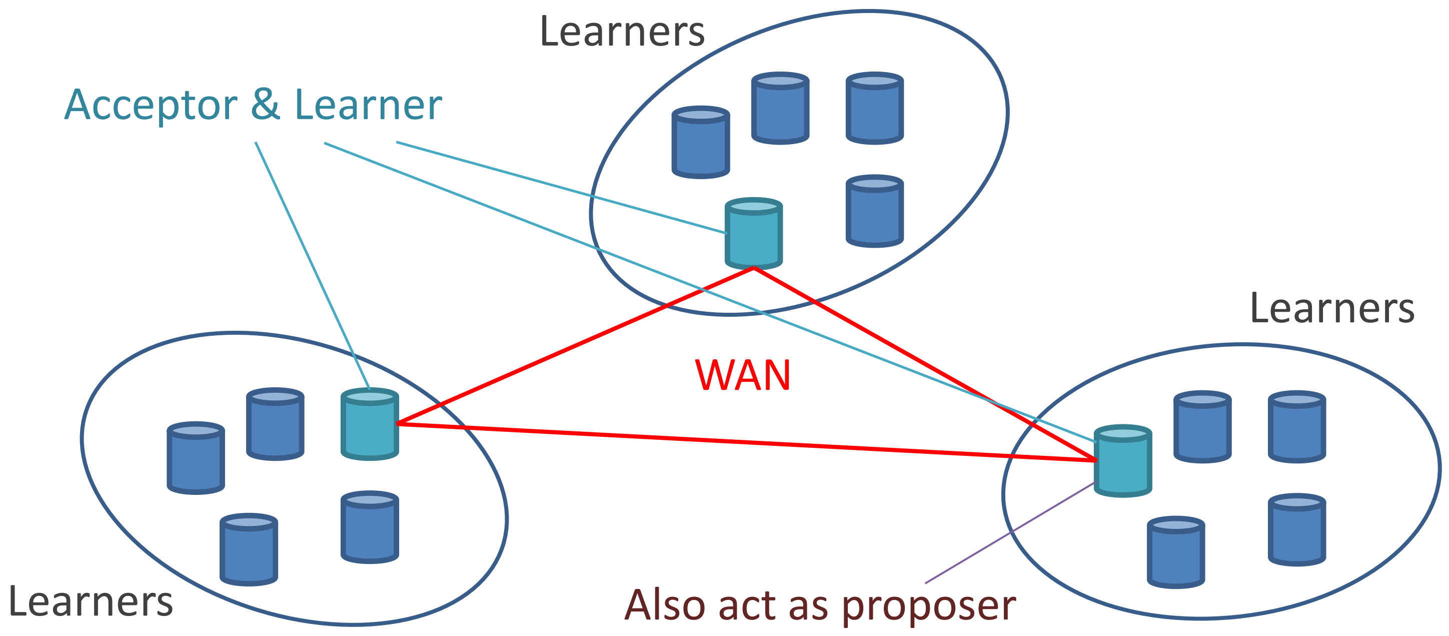
Roles in Paxos

- Client
  - The user that send the request to the server nodes
- Server, may play multiple roles:
  - Proposer
    - Clients send requests to the proposer.
    - Proposer attempts to convince the Acceptors to agree on some value, and acting as a coordinator to move the protocol forward when conflicts occur.
  - Acceptor
    - The proposer sends proposals to the Acceptors.
    - The Acceptors vote to accept the proposals or not.
  - Learner
    - Act as the replication factor for the protocol.
    - Once a client request is agreed by the acceptors, the learner executes the request and responses the result to the client.

System Architecture

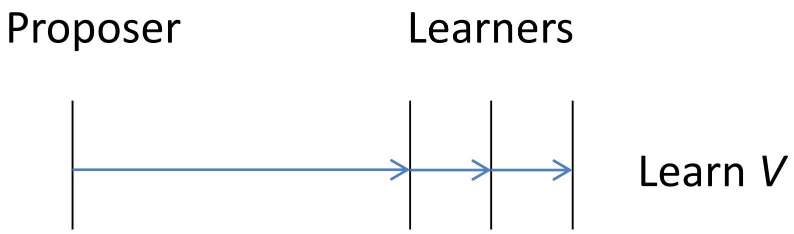


Real World System Architecture



Reach Consensus on Learners

- The goal:
  - Reach consensus on learners
  - All learners should ***learn*** the same value
- How can we achieve this?
  - Have the proposer send the value to learners directly, and the learners learn the value when they receive any value?



Reach Consensus on Learners

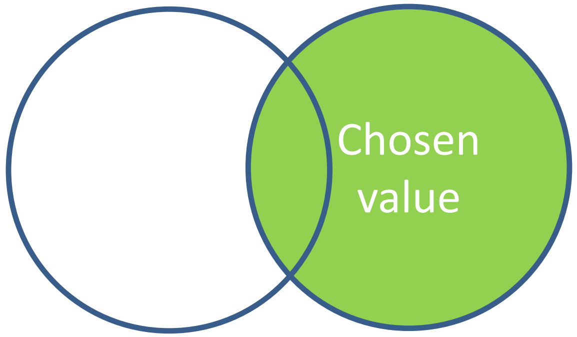
- No
  - The proposer may propose multiple values
  - Or, there may be multiple proposers
  - The messages could be out of order
- Learners could learn different values from different proposers!
- To reach consensus on learners, proposers should communicate with acceptors and ***reach consensus on acceptors*** first
  - Reaching consensus on acceptors implies consensus on learners

Reach Consensus on Acceptors

- If an acceptor receives a proposal, it can ***accept*** (which means voting “yes”) the proposal.
- If a proposal with a value *v* is accepted by a majority of acceptors, the consensus on acceptors is reached, we say that the value *v* is ***chosen***

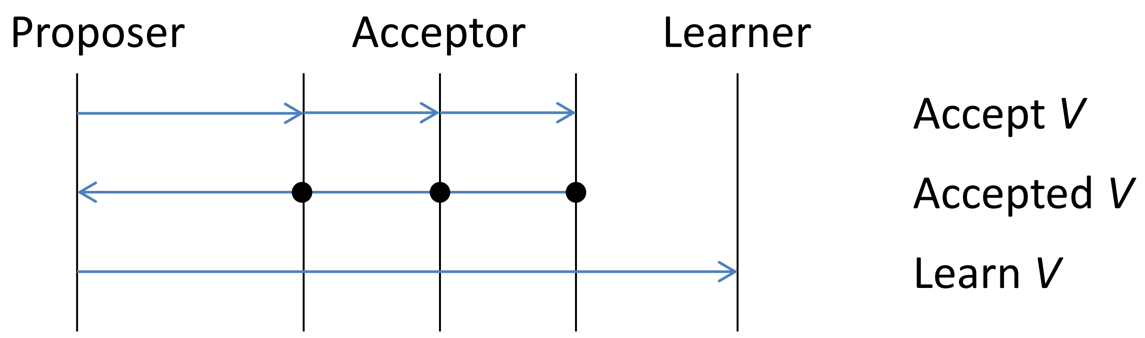
Why majority ?

- There must be at least one common acceptor in two majority sets
- The common acceptors can ensure that at most one value can be accepted by majority of acceptors



Accept Phase

- We first consider the case with only one proposer. A proposer proposes a value, and acceptors accept the proposal
- If the proposer knows its proposal is chosen (accepted by a majority of acceptors), it can notify all the learners what value is chosen
- Note that acceptors do not know whether the value is chosen unless the proposer tells them
- However, the problem caused by multiple proposers still exists

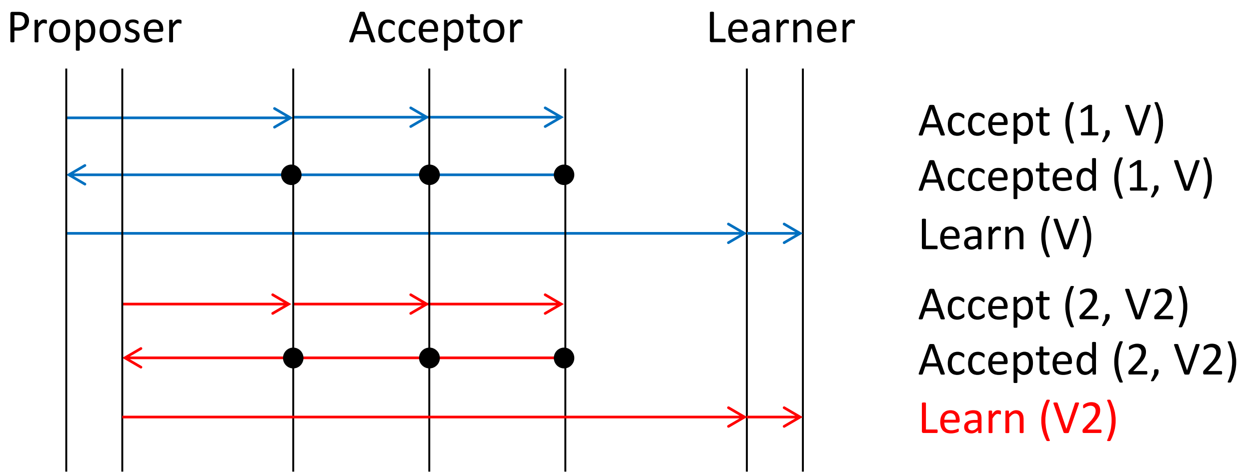


Multiple Proposers

- There may be multiple proposers. If more than one proposer propose at the same time, which one should be accepted by acceptors ?
- Can every acceptor only accept one proposal ?
  - No, if there are three or more proposers, no proposals can be accepted by a majority of acceptors
  - So the acceptors should accept more than one proposal
- Then how should an acceptor choose the proposal ?
  - We assume that all proposals have their distinct number. How ?
    - Each proposer's own counter and its node id.
  - Acceptors accept the highest-numbered proposal it has ever seen
- Then we get:
  - P1. An acceptor must accept the first proposal that it receives

Multiple Chosen Proposals

- Since acceptors can accept more than one proposal, multiple proposals may be chosen, but only one value should be chosen. How to solve this ?
- We can allow multiple proposals to be chosen, but we must guarantee that all the chosen proposals have the same value. By induction on the proposal number, it suffices to guarantee:
  - P2. If a proposal with value *v* is chosen, then every higher-numbered proposal that is chosen has value *v*



How to guarantee P2 ?

- We now have P2, since a chosen value must be accepted by acceptors, we can guarantee P2 by guaranteeing P2a:
  - P2a. If a proposal with value *v* is chosen, then every higher-numbered proposal accepted by any acceptor has value *v*

How to guarantee P2a ?

- Since the proposal is proposed by proposers, we can guarantee P2a by guaranteeing P2b:
  - P2b. If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

How to guarantee P2b ?

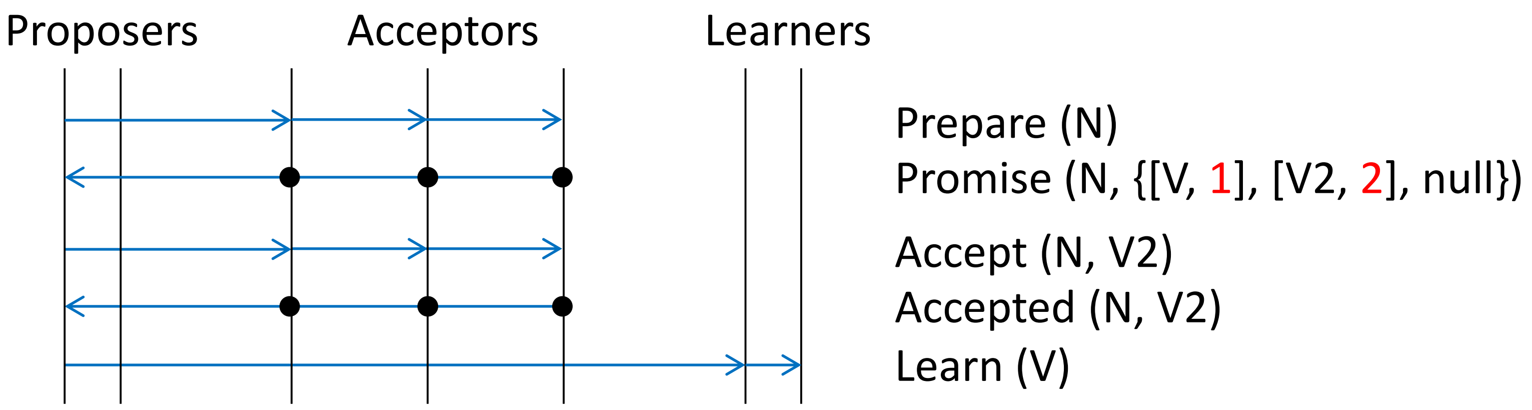
- If a value *v* is chosen, it must have been accepted by some set *C* consisting of a majority of acceptors
- Since any majority set *S* contains at least one member of *C*, we can conclude that a proposal numbered *n* has the chosen value *v* by ensuring P2c:
  - P2c. For any *v* and *n*, if a proposal with value *v* and number *n* is issued, then there is a set *S* consisting of a majority of acceptors such that either
    - (a) no acceptor in *S* has accepted any proposal numbered less than *n*,
    - (b) *v* is the value of the highest-numbered proposal among all proposals numbered less than *n* accepted by the acceptors in *S*
- If we can guarantee P2c, by induction, every higher-numbered proposals have value *v*. Then P2b is guaranteed, P2b implies P2a, and P2a implies P2

How To Achieve P2c ?

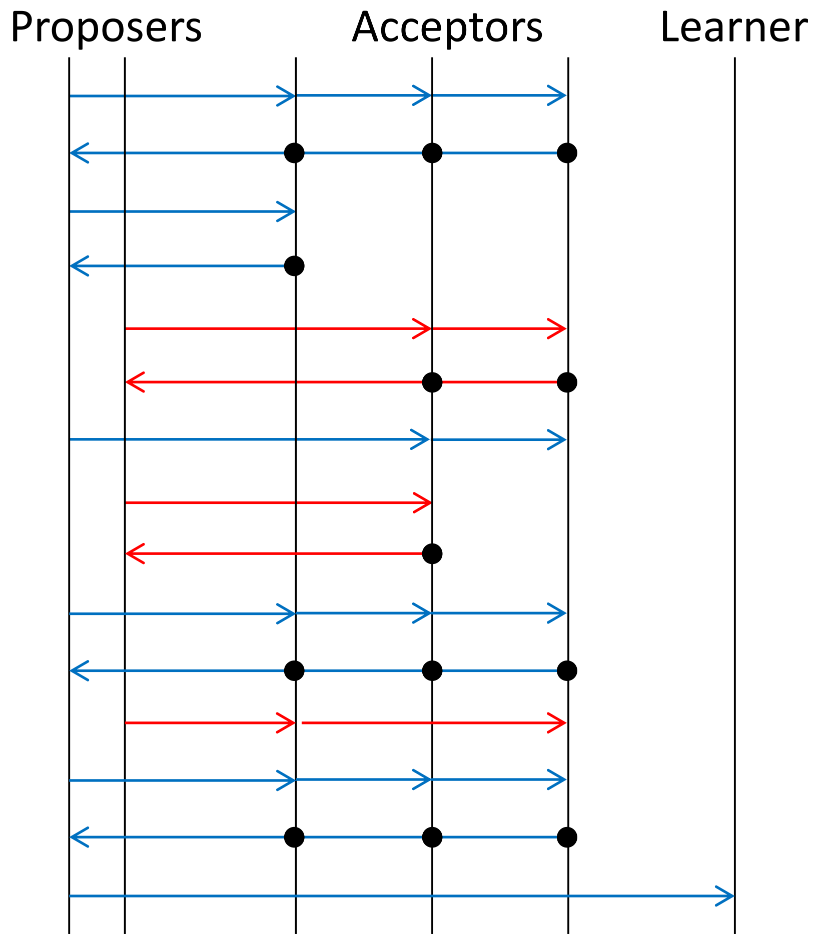
- How to modify the behavior of proposer and acceptor?
  - Before sending the accept message, proposers send a ***prepare*** message to a majority of acceptors to ask if there are already some proposals accepted by acceptors. If there's any, propose the value of the highest-numbered proposal
- Can the acceptor accept any lower-numbered proposals after responding the proposer ?
  - No, the new accepted proposal can't be known by the proposer. So the acceptor should ***promise*** not to accept any lower-numbered proposals again
- Then we should modify P1 to P1a:
  - P1a. An acceptor can accept a proposal numbered *n* iff it has not responded to a prepare request having a number greater than *n*

The Example

- We use the notation
  - *Promise(N, {R1, R2, …. RM})* where *N* is the proposal number, and *{R1, R2, …. RM}* is the set of responses from M acceptors.
    - *Ri* = [Accepted value, Proposal number]
    - *Ri* = null if there is no accepted value.



Example of Prepare Phase



Prepare (1)
Promise (1, {null, null, null})
Accept (1, V)
Accepted (1, V)
Prepare (2)
Promise (2, {null, null})
Accept (1, V) // must ignore
Accept (2, V2)
Accepted (2, V2)
Prepare (3)
Promise (3, {[V, 1], [V2, 2], null})
Accept (2, V2) // ignore
Accept (3, V2)
Accepted (3, V2)
Learn (V2)

Basic Paxos

Algorithm 5.5 RW Abortable Consensus: Read Phase

```
Implements:
    Abortable Consensus (ac).

Uses:
    BestEffortBroadcast (beb);
    PerfectPointToPointLinks (pp2p).

upon event ⟨ Init ⟩ do
    tempValue := val := $\perp$;
    wAcks := rts := wts := 0;
    tstamp := rank(self);
    readSet := $\emptyset$;

upon event ⟨ acPropose | $v$ ⟩ do
    tstamp := tstamp+$N$;
    tempValue := v;
    trigger ⟨ bebBroadcast | [READ, tstamp] ⟩;

upon event ⟨ bebDeliver | $p_j$,[READ, ts] ⟩ do
    if rts $\geq$ ts or wts $\geq$ ts then
        trigger ⟨ pp2pSend | $p_j$, [NACK] ⟩;
    else
        rts := ts;
        trigger ⟨ pp2pSend | $p_j$, [READACK, wts, val] ⟩;

upon event ⟨ pp2pDeliver | $p_j$, [NACK] ⟩ do
    trigger ⟨ acReturn | $\perp$ ⟩;

upon event ⟨ p2pDeliver | $p_j$, [READACK, ts, v] ⟩ do
    readSet := readSet $\cup \{(ts, v)\}$

upon (|readSet| > $N/2$) do
    (ts, v) := highest(readSet);
    if v $\neq \perp$ then tempValue := v;
    trigger ⟨ bebBroadcast | [WRITE, tstamp, tempValue] ⟩;
```

Algorithm 5.6 RW Abortable Consensus: Write Phase

```
Implements:
    Abortable Consensus (ac).

upon event ⟨ bebDeliver | $p_j$, [WRITE, $ts, v$] ⟩ do
    if rts > ts or wts > ts then
        trigger ⟨ pp2pSend | $p_j$,[NACK] ⟩;
    else
        val := v;
        wts := ts;
        trigger ⟨ pp2pSend | $p_j$, [WRITEACK] ⟩;

upon event ⟨ pp2pDeliver | $p_j$, [NACK] ⟩ do
    trigger ⟨ acReturn | $\perp$ ⟩;

upon event ⟨ pp2pDeliver | $p_j$, [WRITEACK] ⟩ do
    wAcks := wAcks+1;

upon (wAcks > $N/2$) do
    readSet := $\emptyset$;
    wAcks := 0;
    trigger ⟨ acReturn | tempValue ⟩;
```

Details of P2c (1/2)

- Why is sending prepare message to ***a majority set*** of acceptors enough to know the chosen value?
  - If a value *v* is chosen, it was accepted by a majority set *C*. By sending prepare message to any majority set of acceptors *S*, since *S* must contain at least one acceptor in C, so at least one acceptor knows *v* and it can tell the proposer.
- Why choosing the highest-numbered proposal ?
  - If a proposal with number less than *n* is chosen, then proposal *n* has the value *v*. By induction, the highest-numbered proposal must have the chosen value.

Details of P2c (2/2)

- Why must the proposer propose the value responded by acceptors ?
  - If there’s any value responded by one or some acceptors, the value is possible to be chosen or isn’t chosen, and we can’t be sure with only majority of responses.
  - For example, if there are three acceptors and proposer gets responses { v, null }, and the third acceptor’s response is unknown.
    - If the last acceptor accepted v, then v is chosen ({v, null, v}). The proposer can only propose the value v.
    - If the last doesn’t accept v, no value is chosen yet ({v, null, ?}). The proposer can propose v to reach consensus.
  - Then the safety requirement “only one value is chosen” is reached.

Three Phases of Paxos

- Prepare phase
  - The proposer sends a ***prepare*** message with number n to acceptors to ask for ***promise*** that
    - Never again to accept a proposal numbered less than n
    - Response the highest-numbered proposal that it accepted
- Accept phase
  - If the proposer gets a majority of acceptors' promise,
    - It can decide the value v, where v is the value of highest numbered proposal among the responses, or is any value selected by the proposer if there are no reported proposals
    - It sends the ***accept*** message with the value
  - Else it can choose a higher proposal number and restart prepare phase.
- Learn phase
  - If the proposal is ***accepted*** by a majority of acceptors, the proposer can send the value to the learners.

Algorithm Of Each Role (1/2)

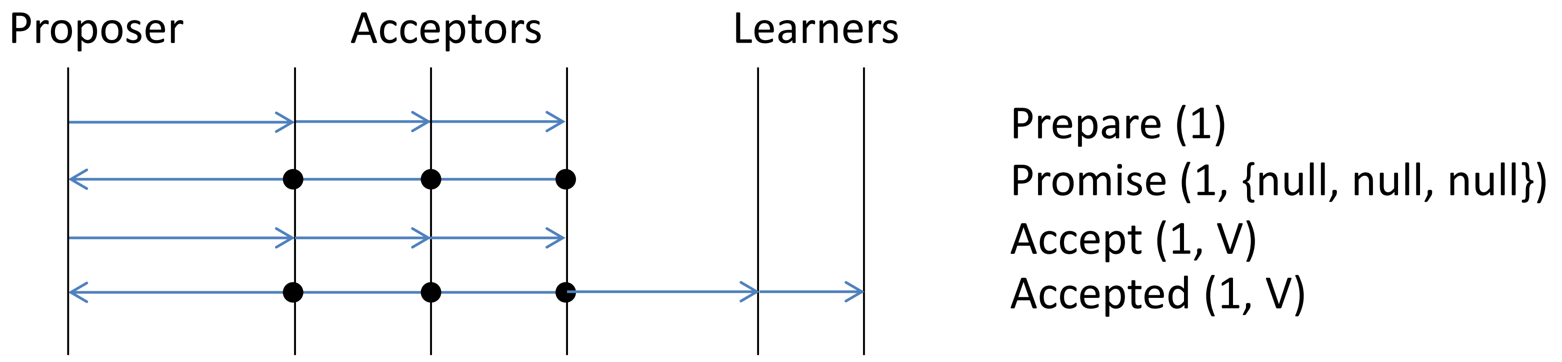
- Proposer
  - Phase 1(a)
    - A proposer selects a proposal number n and sends a ***prepare*** request with number n to a majority of acceptors.
  - Phase 2(a)
    - If the proposer gets a majority of acceptors' ***promise***, it can decide the value. If there are some values responded by acceptors in 1(b), choose the highest numbered one, else choose any value it want. Send the accept request to acceptors.
  - Phase 3
    - If a majority of acceptors ***accepted*** the proposal, send it to learners.

Algorithm Of Each Role (2/2)

- Acceptor
  - Phase 1(b)
    - If it receives a prepare request with a number higher than it has promised, it responds to the request with a ***promise*** not to accept any more proposals numbered less than n and with the highest-numbered proposal (if any) that it has accepted.
  - Phase 2(b)
    - If it receives an ***accept*** request with a number not less than it has promised, it accepts the proposal.
- Learner
  - Learn any value sent by any proposer.

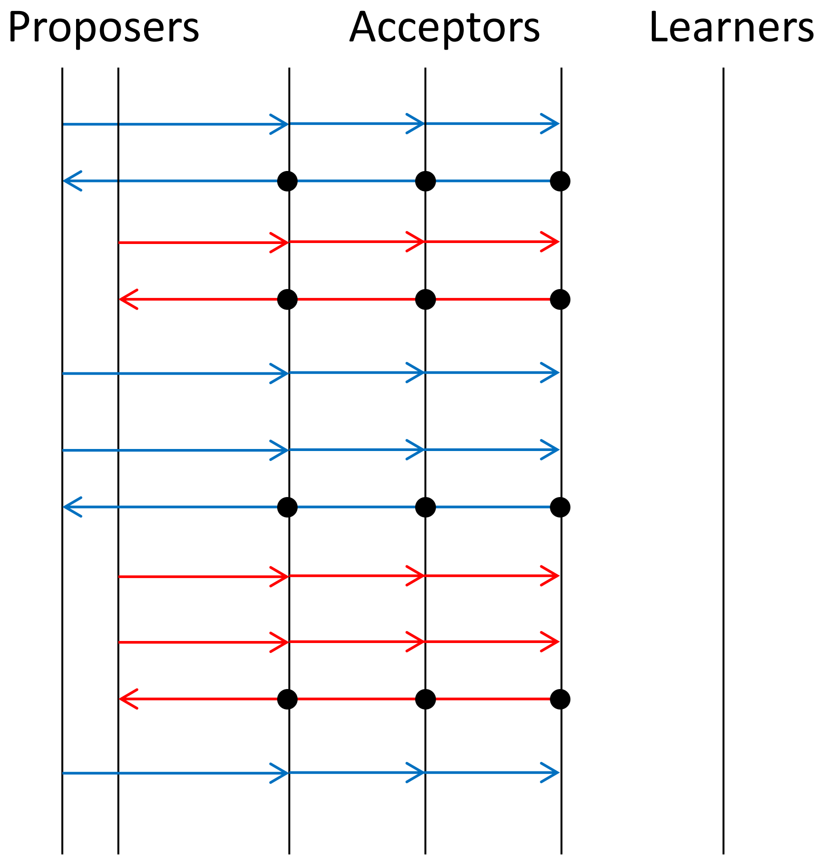
Another Way for Learn Phase

- If the acceptors accept any proposal, then they send the proposals to all the learners. Since the accepted proposal isn’t considered chosen only if a majority of acceptors accept it. The learner can only learn the proposal if it receives accepted proposals from a majority of acceptors.
- This way decreases one communication round, but increases (amount of acceptors * amount of learners) messages.



Progress

- It's possible that no proposers can make acceptors accept value.
- How to solve it ?



Prepare (1)
Promise (1, {null, null, null})
Prepare (2)
Promise (2, {null, null, null})
Accept (1, V) // ignored
Prepare (3)
Promise (3, {null, null, null})
Accept (2, V2) // ignored
Prepare (4)
Promise (4, {null, null, null})
Accept (3, V3) // ignored
…

Total Order via Paxos

- Now we know how Paxos works: each Paxos instance reaches consensus on a single value.
- How to use Paxos to achieve total order?
  - One Paxos run is used to decide the next total order message
  - After the nodes have a consensus on the ith message, the nodes can use a new Paxos run to decide what the (i+1)th message is

Paxos V.S. Two-Phase Commit

- 3 phases in Paxos:
  - Prepare, accept and learn
- 2 phases in 2PC:
  - Prepare and commit
- Which two phases in paxos are similar to the two phases in two phase commit ?
  - Accept phase and learn phase in Paxos are similar to prepare phase and commit phase in 2PC
- Why does Paxos need the first phase ?
  - To prevent there is another proposer
  - In 2PC, there is only one coordinator for one transaction

Paxos V.S. Two Phase Commit

- Why can't two phase commit use majority to make decision?
  - In 2PC, if one participant says "no", then it must abort.
- In Paxos, the consensus value is unknown when a proposer sends prepare messages. But in 2PC, the value is known at the beginning (which is "commit").

Leader

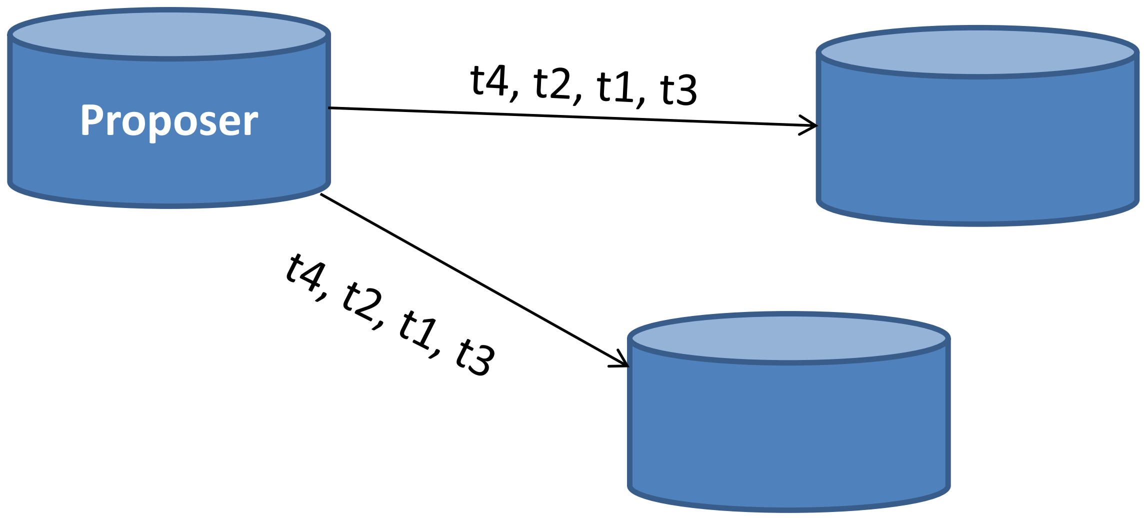
- We can find that Paxos is easier to have progress when there are less proposers
- Why not letting the successful proposer become a ***leader***?
  - The only proposer who can propose in the next Paxos run
  - When Acceptors accept a request, they also acknowledge the leadership of the proposer
  - Clients send request to the leader
- If the old leader fails, a new leader will be elected
- If old leader resumes, there will be two leaders
  - Paxos by nature allows multiple leaders
  - But guarantees progress if one of them is eventually chosen (e.g., by another election)

Zab

- If there is always on ***one*** leader, the first phase is not needed!
- How?
  - The failed leader, after recovery, triggers a re-election first to determine the final leader before sending any proposal

Zab

- In addition, Zab uses TCP connections, which guarantees casualty
  - Zab could act as a total order broadcast, rather than just a consensus protocol
  - The learn phase is similar to sequencer broadcast



View-Change in Zab

- How to know a leader fail ?
  - A Zab leader send heart-beat messages periodically
  - If there is one node that didn’t receive messages, it would start a reelection process
- Zab doesn’t restrict what re-election algorithm must be used
- New leader must ensure
  - All messages that are in its transaction log have been proposed to and committed by a quorum of followers
  - If older leaders proposed a new message, other node would simply ignore it by checking its epoch number