Paxos Made Simple
&
Multi-Paxos

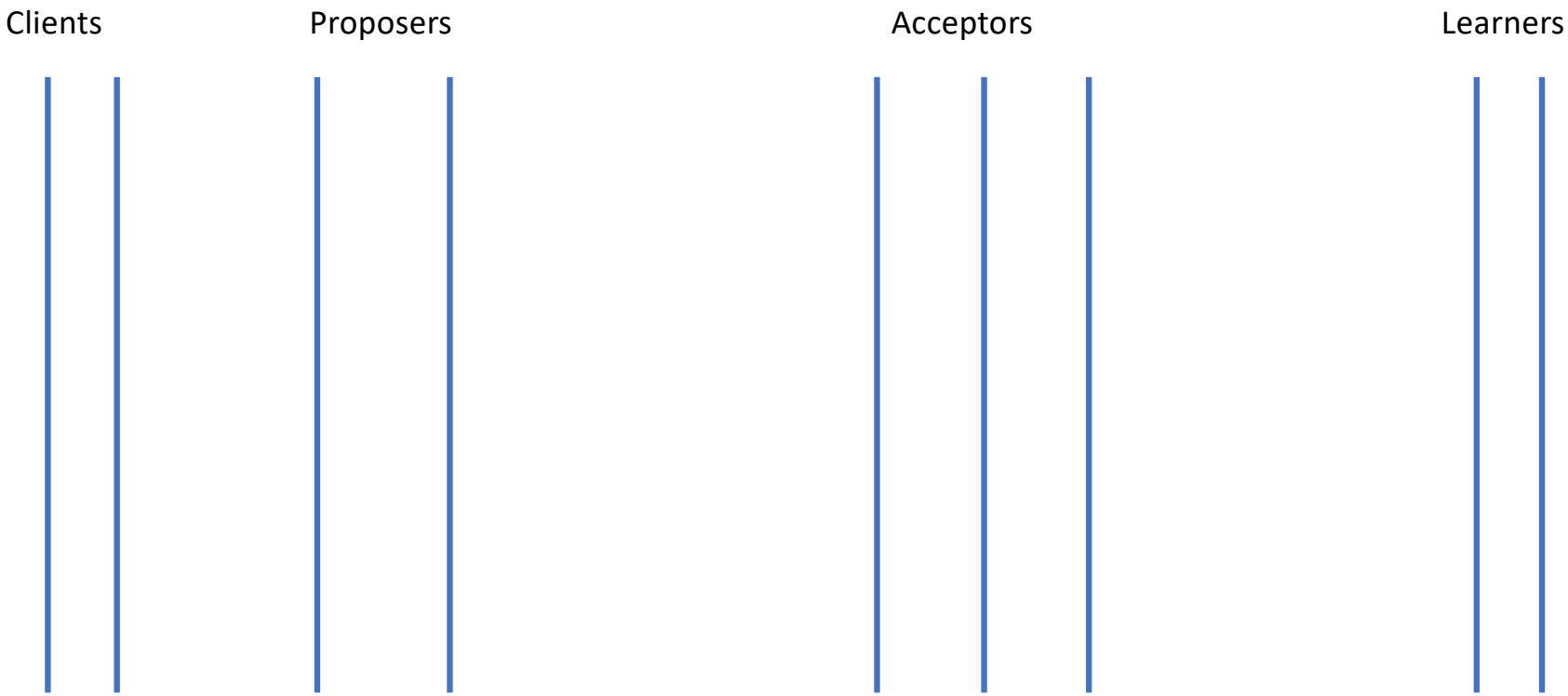
DIDA: Class 04

Paxos Made Simple

Leslie Lamport

01 Nov 2001

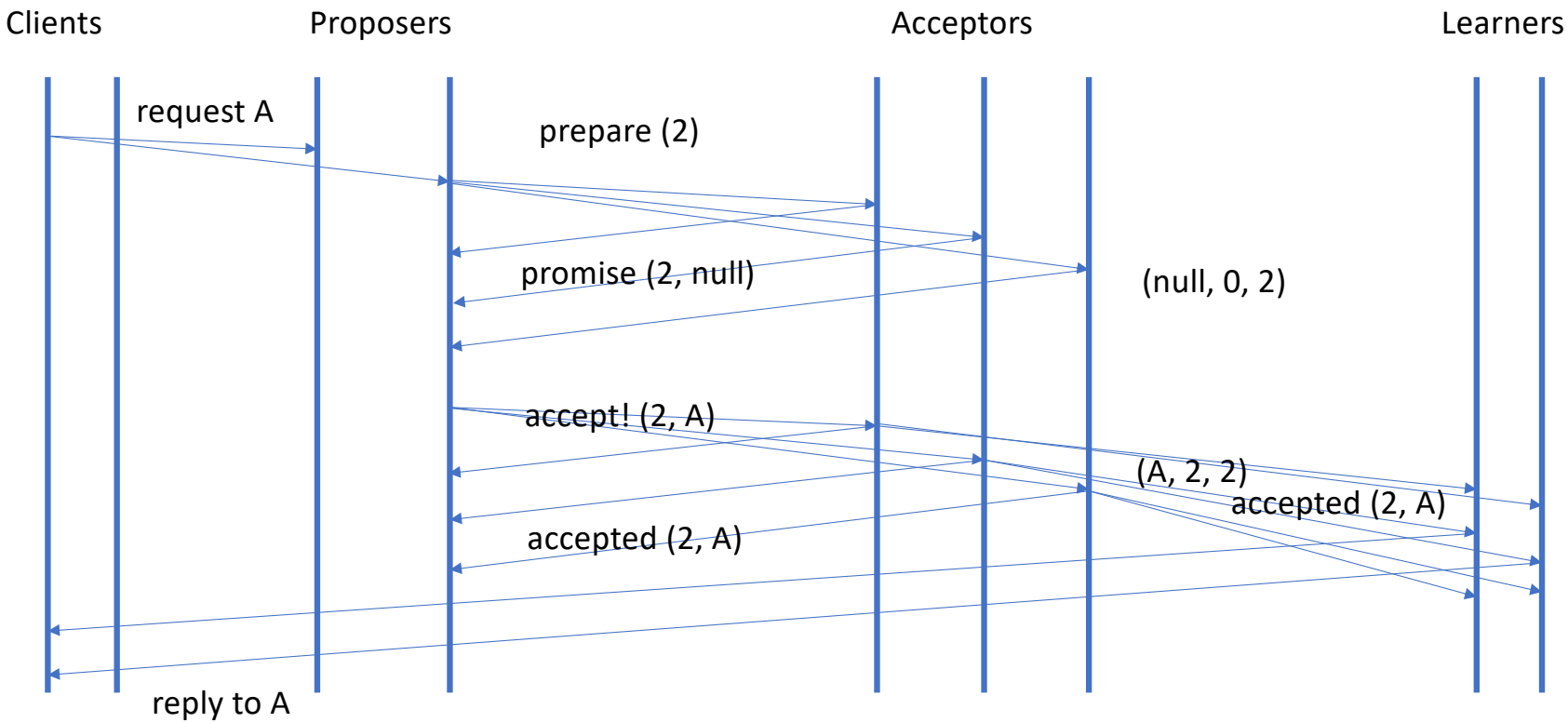
Paxos made simple



Roles

- Clients propose input values to consensus
- Proposers are the leaders: execute the read+write phases of a Paxos leader to commit a value
- Acceptors keep a record of the last value read/written by the most recent proposer (I.e., the tuple <*value, write_ts, read_ts*>).
- Learners obtain the output of consensus

Paxos made simple



From Paxos to SMR

- In State Machine Replication we need to order many commands
- Thus we need to run multiple instances of consensus, one after the other
- We could use a different set of proposers and acceptors for each consensus but that would be very inefficient:
  - A client would send the request to the proposers of instance 1, but if the request would not be accepted, the client would need to send it again to the proposers of instance 2, and so on.

From Paxos to SMR

- We want to keep the same proposers and acceptors for multiple instances
- Proposers pick one request from a client and propose it to instance 1 of consensus
- If this request is not selected as the output of consensus (another proposer has proposed a different request, and that request was accepted), the proposer can re-submit it to the next instance without further interaction with the clients
- Proposers should be learners, to avoid proposers from proposing values that have already been accepted in other instances

Optimizations

- A proposer can start a new instance of consensus while the other is still executing
  - But this may create some weird executions, specially is channels are not FIFO.
  - For instance, decision for command 10 may become known while decision for command 9 is still being agreed

Leader change

Assume that proposer 1 fails and the state of the learners is has follows.

Now assume that proposer 2, that is also learner 2, becomes the leader. It needs to find out what the previous leader has proposed for instances 2, 3, 5, 6, …

Consensus instance	Learner 1	Learner 2	Learner 3
1	F	F	F
2	D		D
3			
4	A	A	A
5	H		
6			

Leader change

Assume that we have 3 acceptors and their state is the following.

Consensus instance	Acceptor 1	Acceptor 2	Acceptor 3
1	<F, 1 ,1>	<F, 1, 1>	<F, 1, 1>
2	<D, 1, 1>	<null, 0, 0>	<D, 1, 1>
3	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>
4	<A, 1, 1>	<A, 1, 1>	<A,1, 1>
5	<H, 1, 1>	<null, 0, 0>	<null, 0, 0>
6	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>

Leader change

Leader 2 will send a “prepare (2)” message for instances 2, 3, 5, 6

Consensus instance	Acceptor 1	Acceptor 2	Acceptor 3
1	<F, 1 ,1>	<F, 1, 1>	<F, 1, 1>
2	<D, 1, 1>	<null, 0, 0>	<D, 1, 1>
3	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>
4	<A, 1, 1>	<A, 1, 1>	<A,1, 1>
5	<H, 1, 1>	<null, 0, 0>	<null, 0, 0>
6	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>

Leader change

- In instance 2, it will learn and adopt the value D
- In instance 3, it will learn that no value has been accepted
- In instance 5, it may lean H or learn learn that no value has been accepted (depending on the replies it receives)
- In instance 6, it will learn that no value has been accepted

Consensus instance	Acceptor 1	Acceptor 2	Acceptor 3
1	<F, 1 ,1>	<F, 1, 1>	<F, 1, 1>
2	<D, 1, 1>	<null, 0, 0>	<D, 1, 1>
3	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>
4	<A, 1, 1>	<A, 1, 1>	<A,1, 1>
5	<H, 1, 1>	<null, 0, 0>	<null, 0, 0>
6	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>

Leader change

- For the instance 3, the new leader is free to propose any value it may want. It may also propose a special “no-op” value and propose new values just in instance 6 and above.

Consensus instance	Acceptor 1	Acceptor 2	Acceptor 3
1	<F, 1 ,1>	<F, 1, 1>	<F, 1, 1>
2	<D, 1, 1>	<null, 0, 0>	<D, 1, 1>
3	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>
4	<A, 1, 1>	<A, 1, 1>	<A,1, 1>
5	<H, 1, 1>	<null, 0, 0>	<null, 0, 0>
6	<null, 0, 0>	<null, 0, 0>	<null, 0, 0>

Optimizations

- After “clean-up”, leader 2 will propose values for instance 6 and above
- Now, every instance of consensus must run step 1 and step 2
  - Remember that only leader 1 was exempted from running step. 1
- How can we prevent this slowdown?
  - Leader can execute step 1 for multiple instances in parallel!
  - In fact, it can execute step 1, in a single message for instances [6, infinity]
  - Then, it just needs to execute step 2, when new commands are received from clients