Lec-16 : Paxos wrap-up: nonterminaton, multi-paxos, fault tolerance

Q) what do you use a physical clock in distributed systems and what shouldn't you use a physical clock for in distributed systems?

physical clocks what are/aren't they good for in DS?

→ We talked about two physical clocks.

(i) Time of Day

(ii) monotonic

	time of day	mono tonic
working points in time	OK type, not good	Not good
duration/ interval	not good.	Good

monotonic are used for implementing timeouts

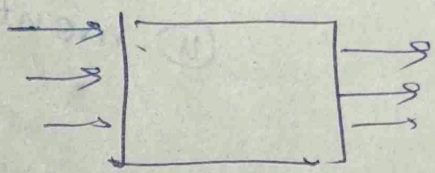
Three property that consensus algo satisfy:

→ Integrity / Validity. → The agreed upon value has to be one of the proposed value

→ Termination → The algo terminates.

- Agreement → They all agree

"Every cone company out of consensus box has a value or it"

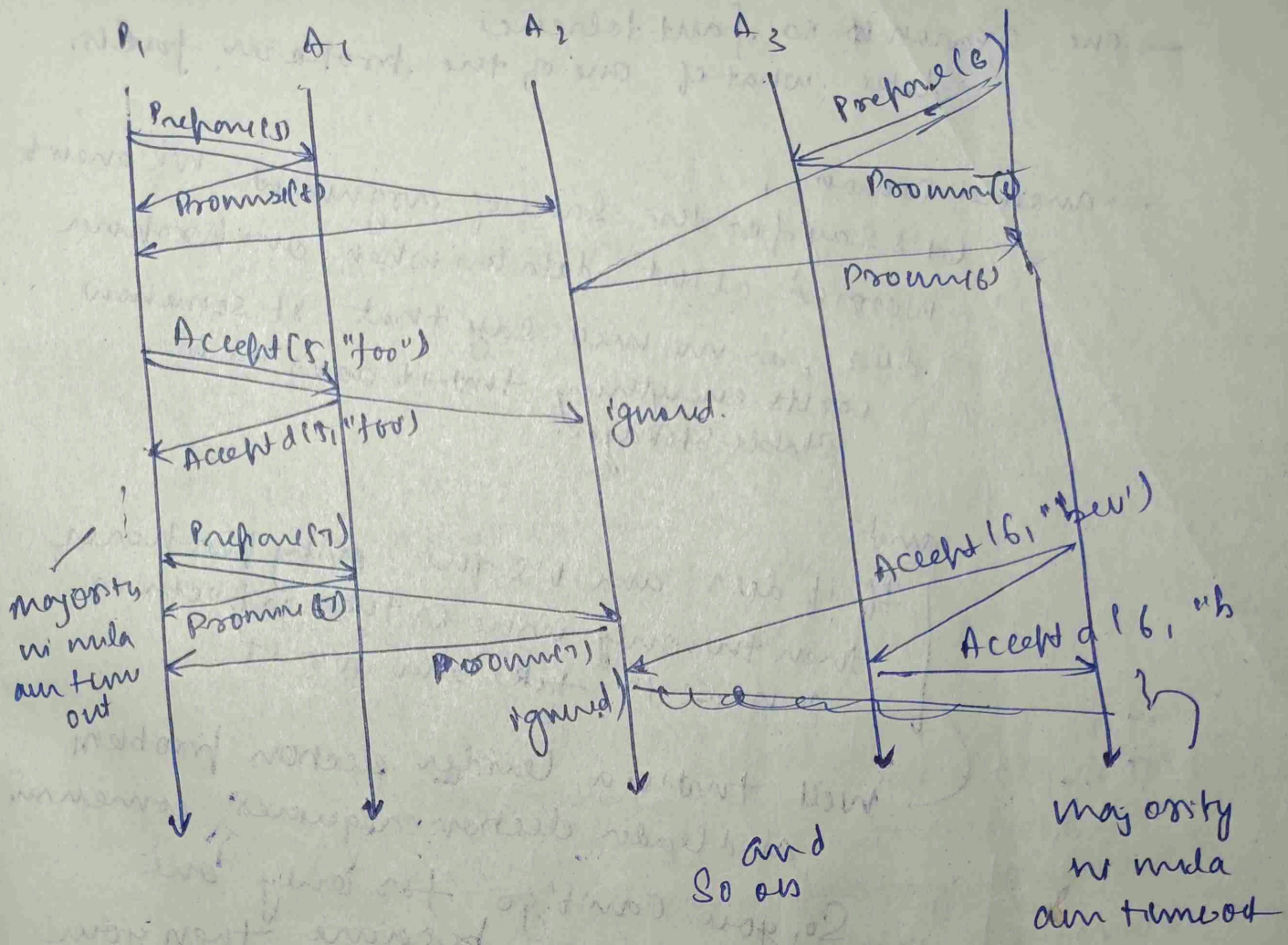


but one of them is not satisfied.

↳ Termination
→ a run of Paxos might not terminate.

→ what kind of a situation will lead to paxos not terminating?

→ It is a situation when different proposers are contending with each other forever.



So, consensus has never reached here; because there's no point where a majority of acceptors send accepted msg for a particular value and proposal no.

Q) All of this non-terminating badness, seems to come up as a result of multiple proposers. So, why not have just one proposer!?

18:10

→ one reason is fault tolerance
→ like what if one of the proposer fails,

→ another reason
let's say for the sake of argument we aren't worried about data loss when one proposer dies, as we will say that it somehow writes everything that it does to a stable storage.

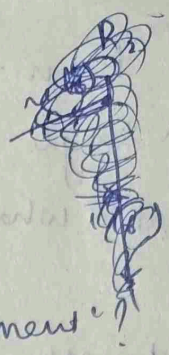
and
If it does and it's the only proposer then the only issue is then is picking a node to take over for it

well that's a leader election problem and leader election requires consensus

So, you can't go for only one proposer because then your consensus algorithm will depend on a consensus algo, and it might not terminate

If consensus protocols have to pick two out of three properties of "termination, agreement and validity" and we have been choosing "agreement" and "validity",

But would it ever make sense to pick a different combination?



Like
Choosing
"Validity" and "Termination"
I throw away "agreement"?

↖ three lines coming out of consensus box has same value on it

→ Yes, it would make sense for "some" consensus protocol to choose this one.

(b)

eg:- "leader election"

— So the thing you can actually do is, use a leader election protocol that doesn't satisfy agreement, but does satisfy termination and validity and use it to choose who should be the proposer for round of Paxos.

and in case if it happens to pick multiple nodes sometimes, then that would be OK, because paxo still works when there are multiple proposers

We have been talking about using paxos to decide on a single value

(that's what paxos does, it decides on one value)

and if you want to decide on sequence of values you have to run paxos again

And it turns out that deciding on a sequence of values is a really common thing you want to do.

Recall the problem of "Totally ordered broadcast"

The problem of making sure that a bunch of processes all deliver the same messages in the same order.

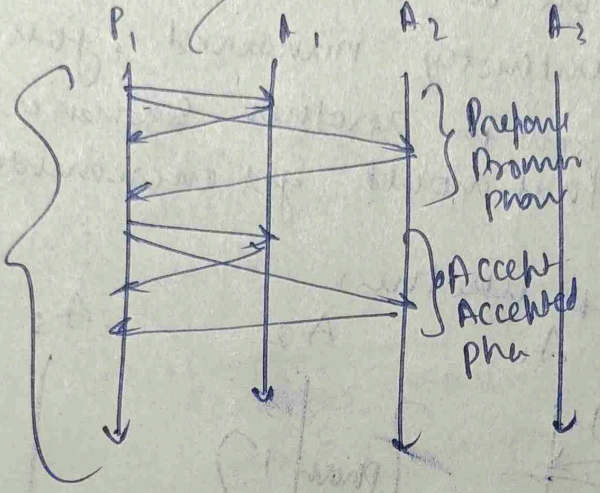
That is essentially the problem of getting consensus on what message should be first then getting consensus on what message should be second then consensus on third and so on.

So, totaly ordered broadcast ~~bcast~~ boils down to repeated consensus on single values.

~~TTC~~

This method can be really slow

cause in the best case Paxos take a lot of message

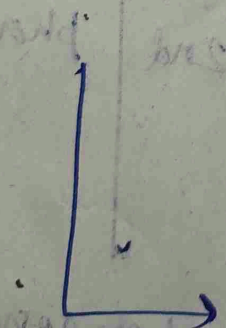


Full run of Paxos

That's the best case, where no process crashes, no proposers are competing with each other, but it's still two round trips with the acceptors.

So that's slow

Is there anything that can be done to make it faster so that if we have to get consensus on many values, we don't have to do two round trips every time?

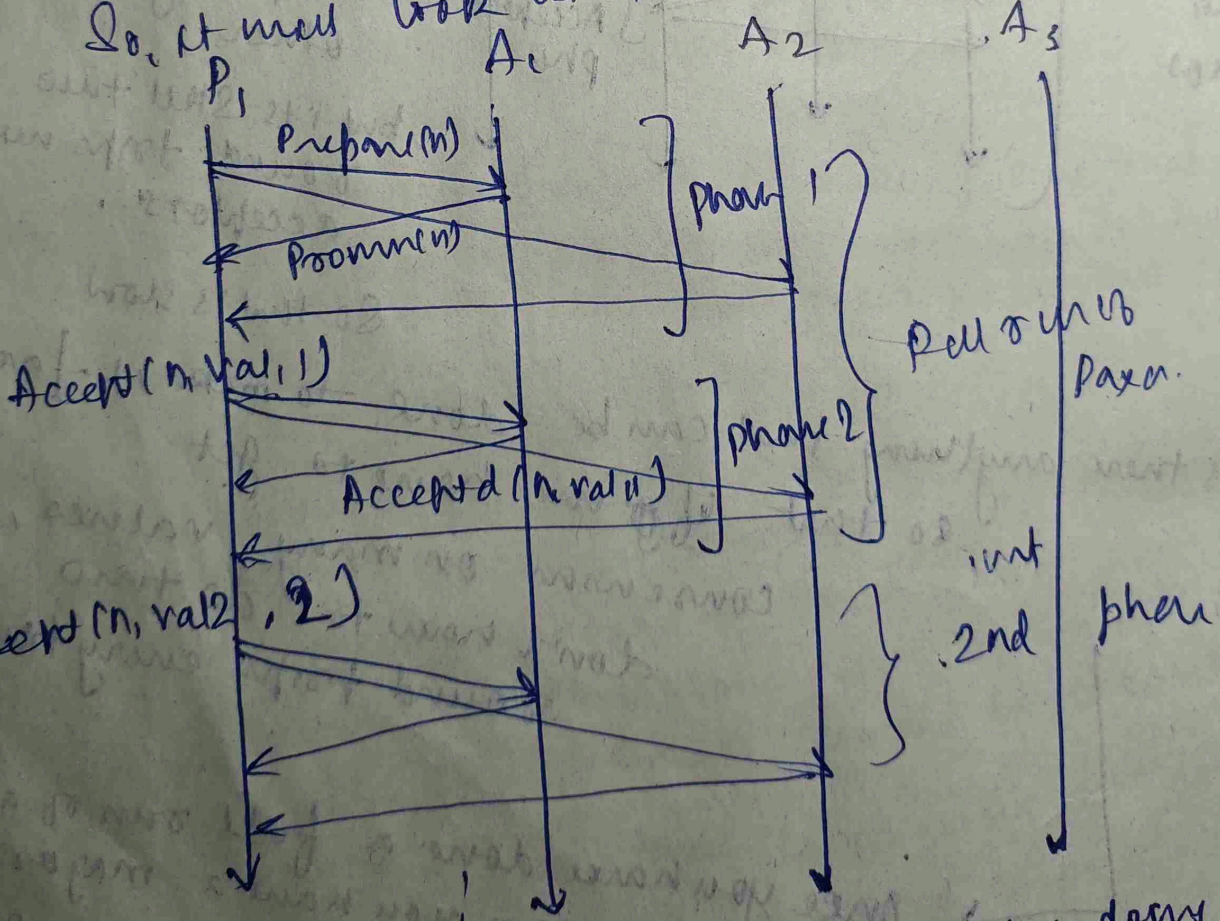


Once you have done a full run of a Paxos, now you have a majority of acceptors on board with you and you can just keep on doing

"accept and accepted" phases with that single proposer, as long as the proposer doesn't crash.

So, you can keep on doing accept messages with the same proposal number and if you want the accept messages to be distinctly numbered, you can throw in another sequence no. that would get incremented

So, it will look like this:



and so on, keep on doing 2nd phase.

This is what known as multi-paxos

— It turns out this is safe to do

How many acceptors can crash?

eg: If I have 3, and I still want my run of paxos to work, will it still work in a situation, where an acceptor crashes, would it still work?

— If we have three acceptors, and two is a majority but one, but then 2 is still a majority,

So, if you have three you can tolerate one acceptor crashing and still have the algo work, it might not be great as, then you would have no room for acceptors crashing

If I have 5 acceptors, then 2 can crash.

In general, you can handle minority of acceptors crashing.

How many proposers can ~~crash~~? Crash?

→ All except '1'.

In Summary,

If 'f' is the number of acceptor failure you want to tolerate, you need 2f+1 acceptor

and

If 'k' is the no. of proposer failure you want to tolerate, you need f+1 proposers.

What about omission faults?

→ How does Paxos do under omission faults?

If I look a msg here and there during a run of paxos what could go wrong?

→ It would be fine

(Let's say 3 acceptors mai ek acceptor ko msg gaya nhi lost h, to koi ni as long as majority se aa jayes respn.)

→ If all msg to all acceptor from a proposer is lost

→ To bhi koi ni

Let's talk about other consensus protocal (not in detail)

(khud se detail mai dekh le)

So, here are few consensus protocal that you might want to be aware of other than Paxos.

→ PBFT

- ViewStamp Replication (VSR)
  - by Brandt & Barbara Liskov (1988)
- Zab (ZooKeeper Atomic Broadcast)
  - by Yahoo Research, late 2007
  - opensource.
- Raft
  - by Diego Ongaro & John Ousterhout, 2014

They all are achieving consensus on a sequence of values, rather than one.

(So, they are more like 'MultiPaxos'

(and as far as the teacher understood, they all bake in leader election as a fundamental part of protocol.

(Recall, that teacher mentioned, that one way to do consensus protocol, would be to first have leader election, to decide who the proposer should be, and let that proposer do all the proposing and then avoid the problem of dueling proposers as you would only have one.

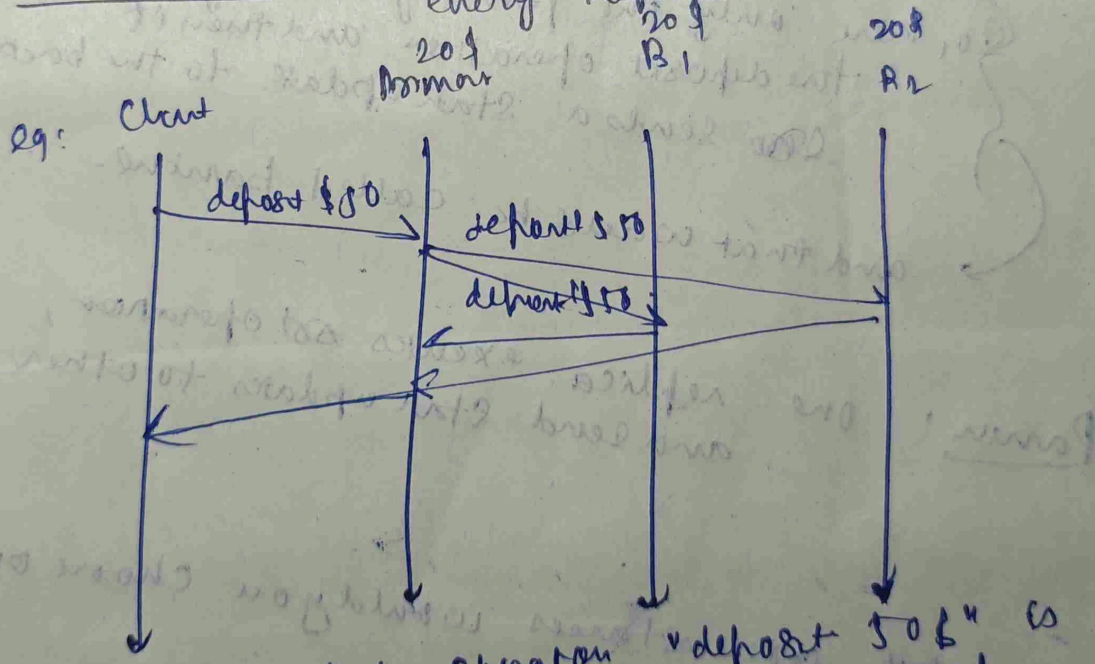
If ek proposer
se ye ni hua ki
termination ka ensure
khatam ho gaya
(kyuki 'Leader election'
khud mai ek alag
consensus protocol h jo
ye guarantee ni karta
terminate

Consensus can be it didn't fail to terminate ie:
If you compromise on agreement rather than on termination, then you can end up with two leaders, then you would have dueling proposers anyway)

— Another paper which compares many consensus algs.
"Vive La Difference" by Van Renesse, Schiper and Schneider (2014)

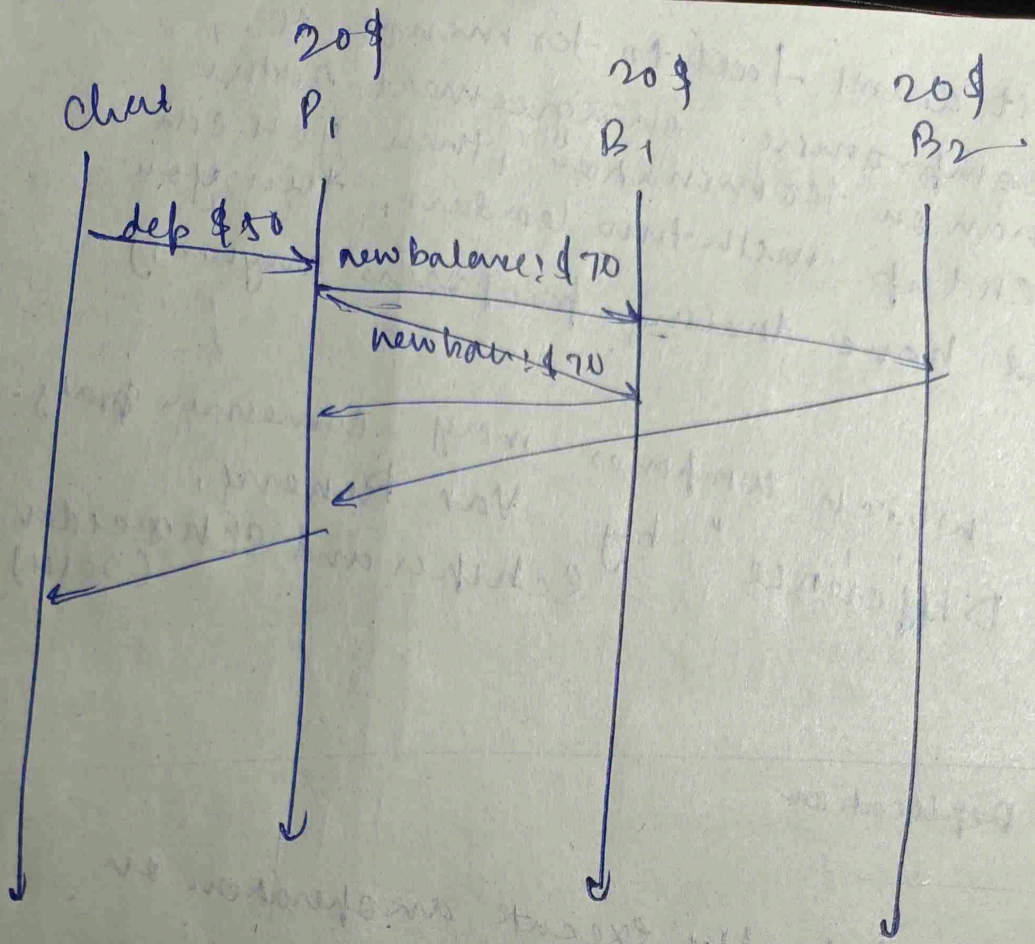
Active and Passive Replication

- Active Replication: Actually execute an operation on every replica.



You can have the operation "deposit 50$" is sended to all the replicas and executed on them.

alternatively you could do



So, here only the primary actually executes the deposit operation and then it sends a state update to the backup

and that would be called passive.

Passive: one replica executes operation, and send state updates to other.

Under what circumstances would you choose one over others?

If the update state is large than go for active replication.

and if computation is really expensive to do determine

Active replication is synonym of "State machine replication"

The idea of state machine replication is that you have bunch of replicas that all need to have same operations executed on them in the same order and therefore they go through the same sequence of states.