Lec 14 - handling node failure in replication protocols.

When do you need consensus? 8:10

→ You have a bunch of processes, and...

(i) need to make sure they deliver the same messages in the same order.
(you'll need them all to keep track of which processes exist. You would need them all to keep a list of which processes exist. And whenever anyone dies or leaves for whatever reason, everyone's list has to get updated)

(ii) you need them all to know which processes exists and which are running.

(iii) You want one process to play a distinguish role (and you want other processes to know about it).

(iv) You want them to take turns getting mutually exclusive access to some resource.

(v) They are all participating in a transaction and you need them to agree on whether it's committed/aborted.

It turns out that these are all classic distributed system problem.

and they have names

P T C

(I) → Totally Ordered BroadCast problem
(also called atomic broadcast)

(II) → Group membership problem
(or failure detector ;))

(III) → Leader ~~detection~~ election problem.

(IV) → Distributed mutual exclusive problem

(V) → Distributed Transaction commit problem.

So, what does all these processes have in common?

→ They all involve this bunch of processes trying to co-ordinate with one another and reach agreement about Something.

The thing they are trying to agree on can vary

But all of these are going to boil down to the consensus problem,

(which is getting a group of processes to co-ordinate.

and agree on something.

Note:- The thing that makes consensus hard in DS is faults (crash faults & omission faults which we are going to consider)

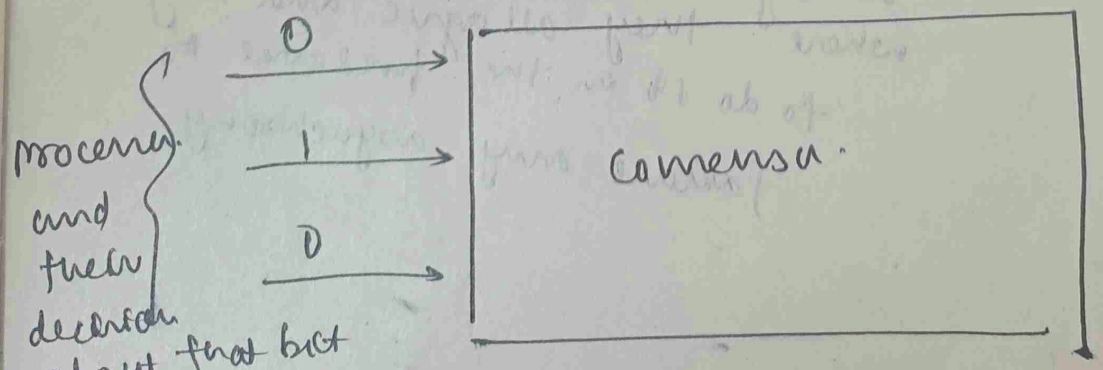
Consensus

→ So, we have some processes and we want to get them agree.

And to make things simple, let's say that the thing that they are trying to agree on is just one bit

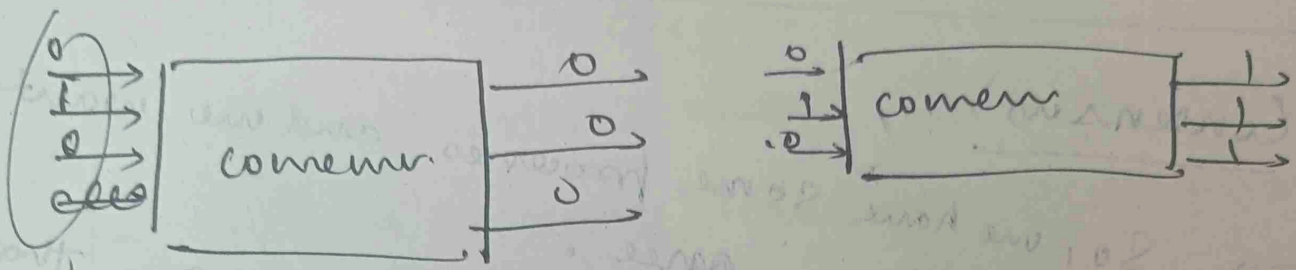
(zero or one.

So, every processes gets to have a say, and every processes gets to contribute one bit → zero or one



The role of the consensus is to make all process agree on a single ~~bit~~ thing

either or



and indeed two notion of majority is ~~going~~ is important

Note:- You have to keep in mind is there isn't some external process that looks at these numbers and makes the choice.

Rather, it's {there} three processes coming in without any knowledge of what number the other ones picked that have to coordinate among themselves and reach a point where they all agree and they have to do it in the presence of failure any asynchrony.

Properties that algorithms try to satisfy :-

(i) Termination :-

Each correct process eventually decides a value.

(ii) Agreement :-

all correct processes agree on the same value.

(iii) validity or Integrity or non-trivality

→ the agreed upon values must be one of the proposed values.

We said that consensus algorithm try to satisfy these properties.

But it turns out that no consensus algorithm actually does satisfy them all, especially not in asynchronous network model.

Paxos

— It is a consensus algorithm invented by "Leslie Lamport"

→ Paxos is a family of algorithm.

— for now, we are just going to talk about "Vanilla Paxos"

→ There are three roles that a Paxos can play.

- proposer :- propose values
- acceptor :- contribute to a decision from among the proposed values.
- learner :- learn the agreed upon value.

Note :- In practice, a single process might can take multiple roles or even all of them.

but when you are learning Paxos it is good to assume that one process can take one role at a time.

- one thing that every node is to know that what is the "majority" of acceptor is.

  (eg if 3 acceptors then majority is 2.
  if 5 " " " 3.

So, all nodes need to know this in advance.

For now, the protocol we are looking at is for deciding on a single value.

↳ But, if you want to decide on a sequence of values then you have to run the protocol again.

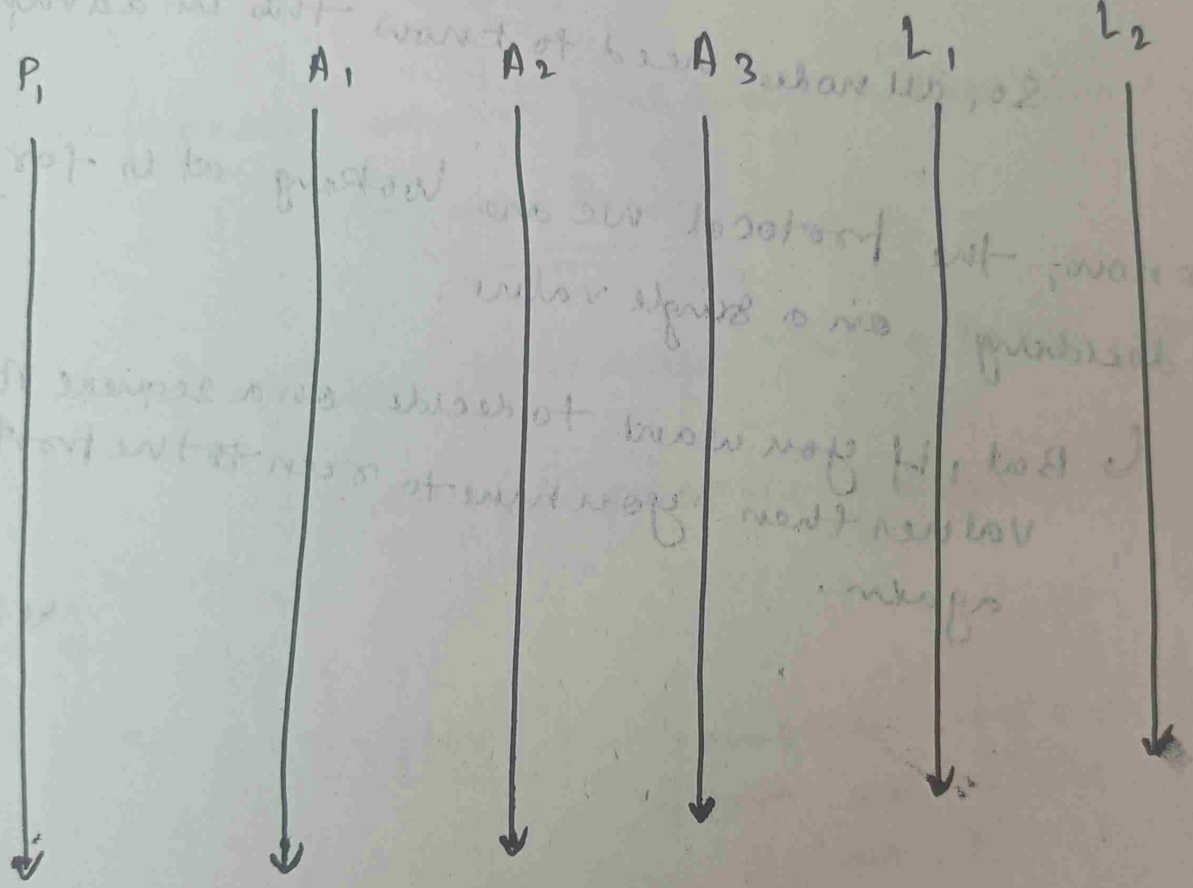
40:00

Let's look at an actual run of the algo:

→ Let's say there's one proposer → P_1

→ 3 acceptors ✓ majority → 2

→ 2 learners.



Let's say our proposer P_1 wants to propose a value.

So, the first thing it had to do is to send what's called an "Prepare message" to atleast a majority of acceptors.

So, P_1 is going to send a msg "prepare", and that msg is going to come along with "proposal number"

Note: The "proposal number" need to be unique

(And if you had multiple proposers, you might establish rules in advance,

like: one proposer can only use odd numbered proposal nos.

and other proposer can only use even numbered proposal number)

That has to be established in advance

→ And the proposal number has to be higher than any of the proposal number that proposer has used before.

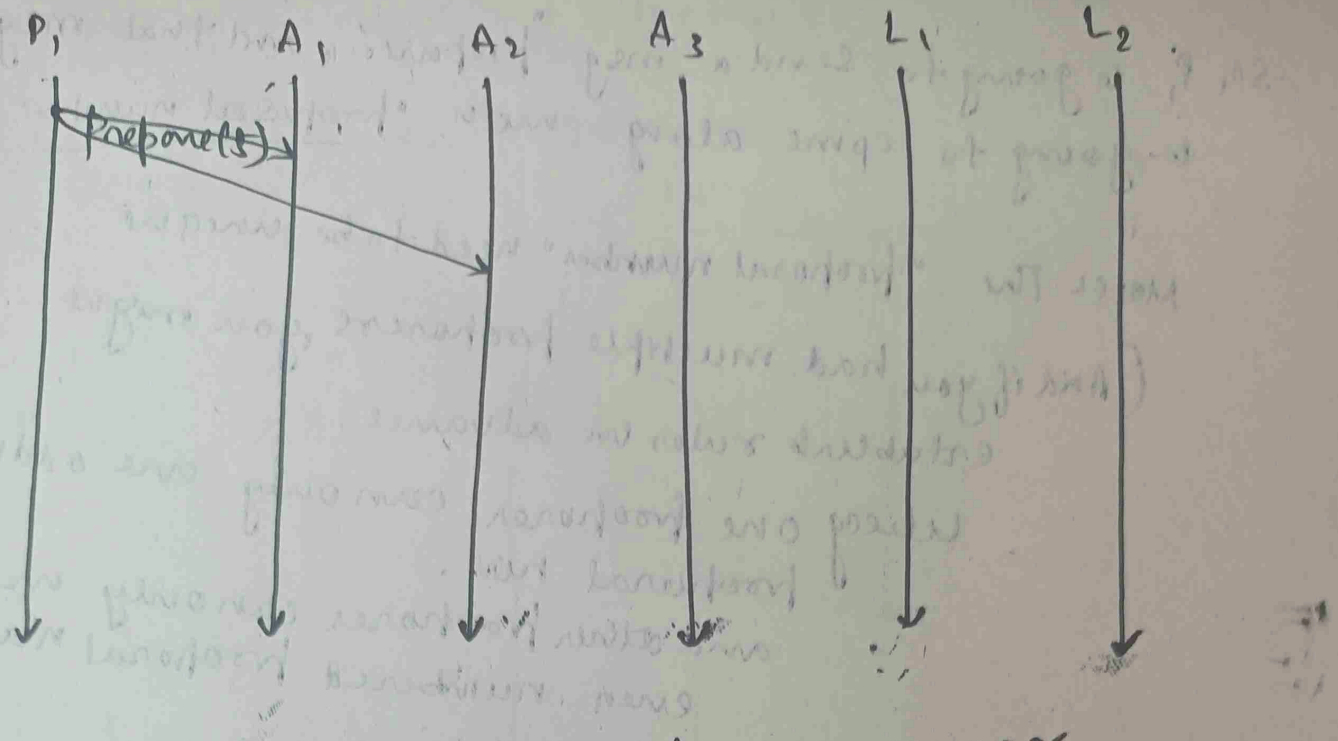
(So, it has to be able to store the proposal numbers it's used and make sure not to reuse them)

So, proposer 1 ($P1$) sends a prepare message with a proposal number.

(Let's say the proposal number is 1)

and P_1 is going to send it to majority of acceptors.

(Let's say it goes to A_1 & A_2)



→ when an acceptor gets a prepare message, it looks at that proposal number, and it does this check! -

"Did I previously promise to ignore requests with this proposal number?"

If it did → then it ignore it

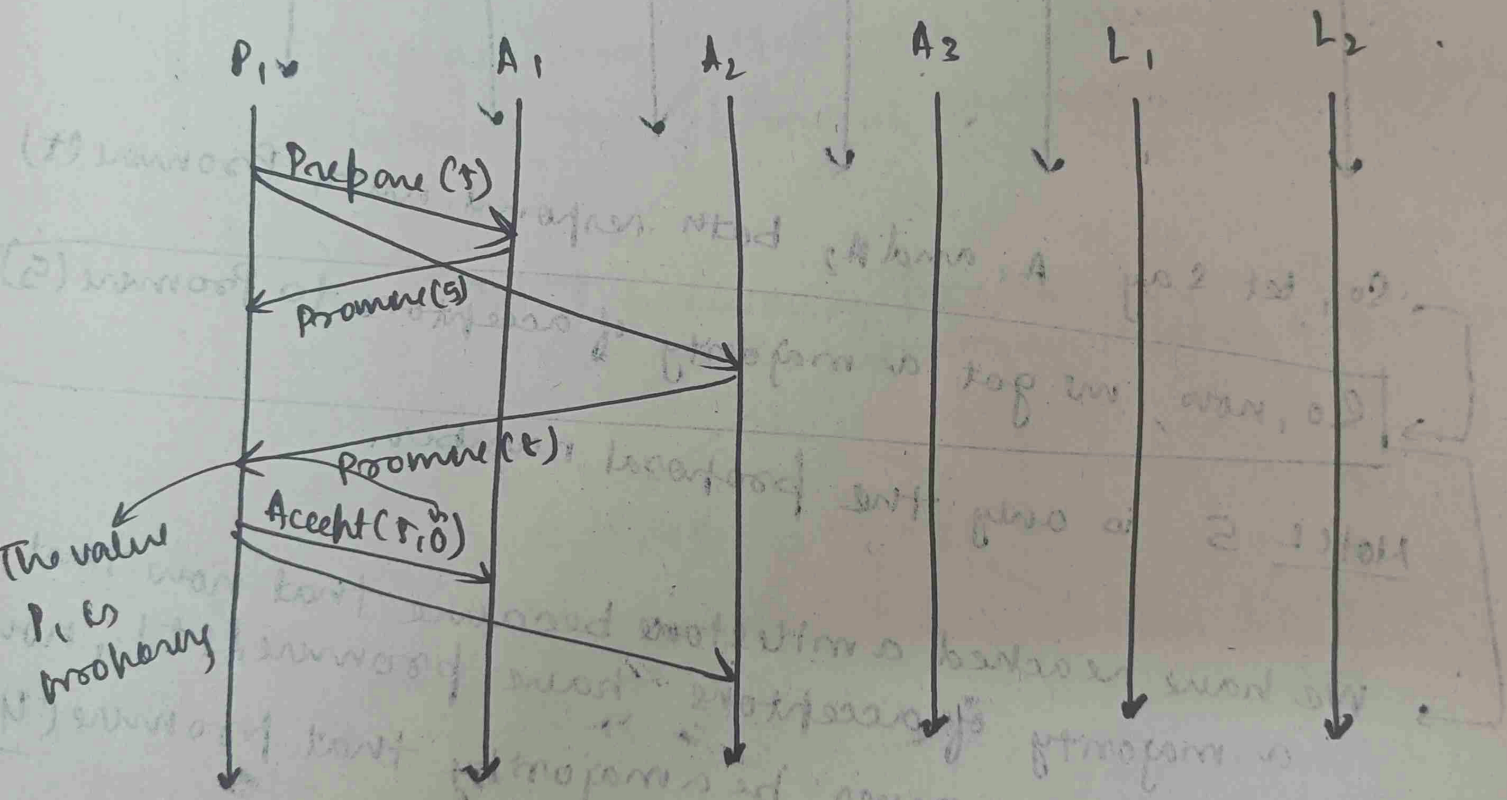
If no → It now promises to ignore any request that has a proposal number lower than that.

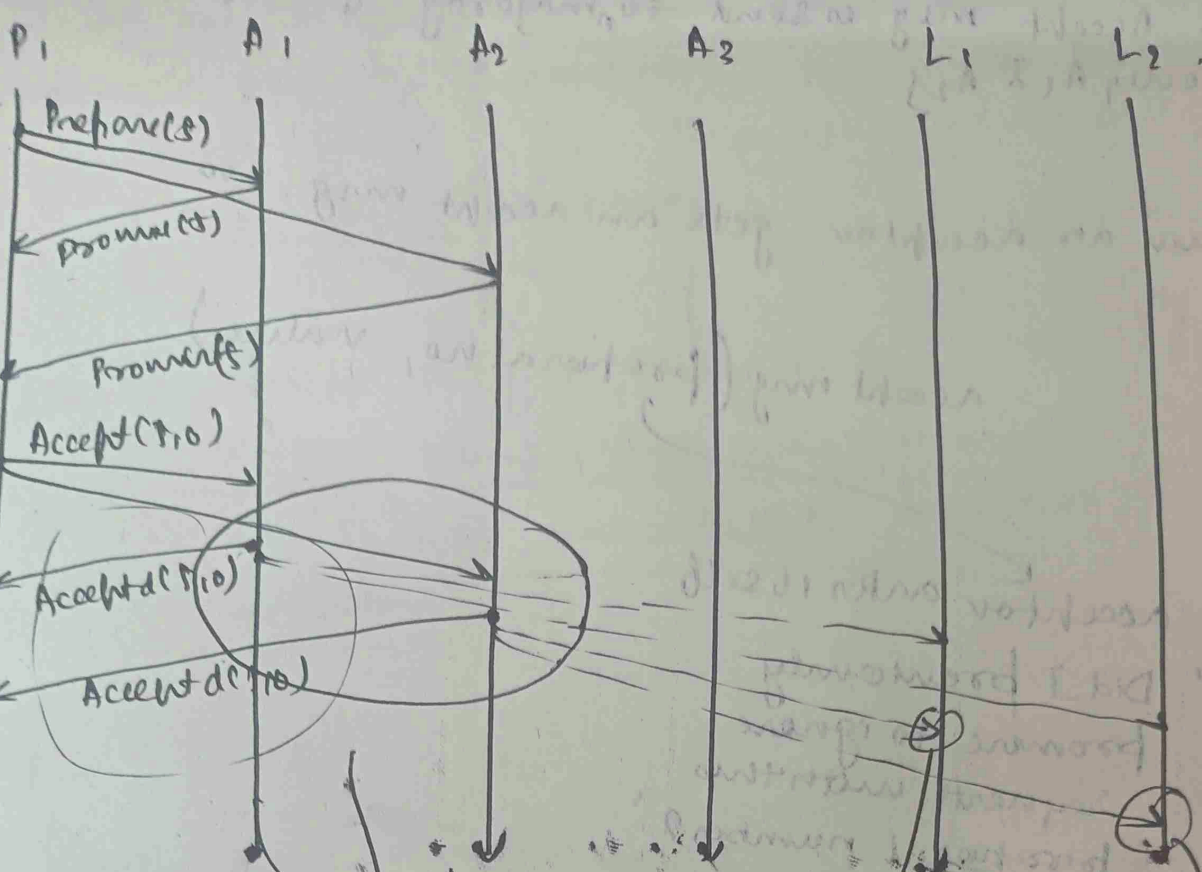
So, the Acceptors replies Proposer with a "Promise" msg

Promise(5)

Now that the proposer has received promise messages from a majority of acceptors, it can now send what is called an "accept msg" to a majority of acceptors.

Note: The Accept msg is going to have the 'proposal number that was promised' and the 'actual value that P_1 wants to propose'.





So, the point when a majority of acceptors send their {Accepted} msg for a particular proposal no. & particular value is the point when consensus is reached.

Consensus is reached

Here P_1 knows that consensus is reached (when majority reply accepted)

L_2 knows here.

L_1 knows here

"So there is a difference btw the moment at which consensus is reached and the moment at which everybody finds out that consensus is reached"