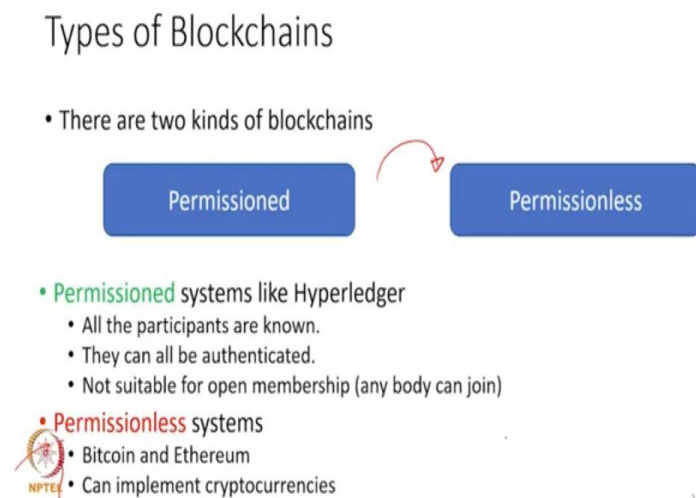


**Advanced Distributed Systems**  
**Professor Smruti Sarangi**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Delhi**  
**Lecture – 28**  
**Stellar Consensus Protocol**

Welcome to the lecture on the Stellar Consensus Protocol. So, in this lecture we will discuss a very different kind of consensus protocol. So, we have seen a lot of consensus protocols up till now, we have seen Paxos, raft, we have seen Bitcoin, Ethereum. So, stellar is different, it is more scalable, it is also new, so, stellar is a 2016-18 protocol. So, we will see it is much newer, much faster, much more complicated as well.

(Refer Slide Time: 00:50)



So, as we have discussed, there are two kinds of blockchains that are permissioned blockchains, which are easy systems such as hyperledger, for instance. Then, we have permission less blockchain systems, such as Bitcoin and Ethereum, where you do not need permission. But, because of probabilistic or rather cryptographic guarantees, it is possible to ensure that there is only a single version of the chain that is accepted. In the case of a permissioned system, such as hyperledger, all the participants are known they have login IDs, so all of them can be authenticated. And if they turn malicious, some action can be taken.

So, it is clearly not suitable for open membership as we have seen, so, it is not that anybody can join at any time. So, this is more like a corporate intranet kind of system. So, what makes

blockchains really interesting is permissionless systems of which Bitcoin and Ethereum are prime examples. So, we can use them to implement cryptocurrencies, which is by far their biggest selling point.

(Refer Slide Time: 02:04)

## Notion of Quorums

- Fault model
- Byzantine faults
  - Nodes can **behave** arbitrarily and maliciously
  - Malicious nodes can also **collaborate** with each other
- Classic fault tolerance result with Byzantine **faults**



With  $3f+1$  nodes, we can tolerate at the most  $f$  Byzantine failures.



In this case, at least  $2f+1$  nodes make a **quorum** (all need to be correct)

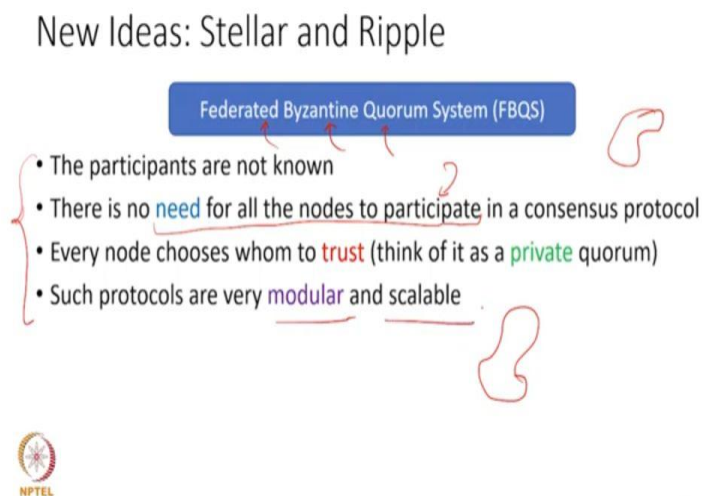
So, the key point that all of them use, including any protocol which is tolerant to Byzantine faults, and that includes in this case, Ethereum and Bitcoin as well. Is that in the case of Byzantine faults, nodes can behave arbitrarily and maliciously. And furthermore, malicious nodes can kind of cooperate with each other, collaborate with each other, and send messages to confuse the rest. So, if you look at our classical fault tolerance result, which is there earlier in this lecture series, you will find that Byzantine faults if we have  $3f + 1$  nodes, we can tolerate at the most  $f$  Byzantine failures.

And then also it takes a lot of time, it kind of takes  $n$  factorial time to still come at an agreement. So, still come at a consensus it takes that much of time, but nonetheless, with  $3f + 1$ , the maximum number of Byzantine failures we can tolerate is  $f$ . Not more than that, because more than that the system is not guaranteed to complete. So, in this case,  $2f + 1$  nodes make a quorum. So, what does that mean? All of them need to be correct. And all of them need to agree on the same value, agree on the same consensus value, regardless of the rest. So, we can think of them as having a quorum.

Even a classic centralized system, where if the centralized server is guaranteed to be honest, then all that you need to do is you need to send a message to the centralized system, get a reply. And

insofar as all the nodes are concerned, the centralized system is the quorum. So, what is a quorum in this case? A quorum is a set of nodes who have to accept your right and to provide you the value of a read. So of course, they may collaborate among each other to provide you the value of a read. But, at least if they are correct, you know that the entire system is correct. In the case of classical Byzantine fault tolerance  $2f + 1$  node, is the minimum that you should have for a quorum size, to ensure you are getting the correct answer back.

(Refer Slide Time: 04:24)



So, some new ideas have been proposed, such as stellar and ripple. So, they use a very interesting new concept known as federated Byzantine quorum system. So, Byzantine we know, quorum we know, what is federated? Well, federated basically means, like a federal in a federation of states. So, in a sense, it is like it is a hierarchical network or hierarchical setup that different groups kind of elect their value, and then ultimately, one of them is chosen. So, you can think of the fact that the entire network is broken into different subgroups, and each subgroup in a sense, elects in a certain sense its value.

But, what are the key advantages? So, why have stellar and why not have others? Why are we discussing it? The biggest advantage of stellar is that the participants are not known. So, unlike many of the other protocols, which includes Bitcoin and Ethereum, fair if at least the participants are known, or the network is known or some multicast ID is known, where you can send it to all

the participants. In this case the participants are genuinely not known, they can join and they can leave at will. There is no need for all the nodes to participate in the consensus protocol, this is key.

Because what was happening in Bitcoin, for instance, is that all the nodes had to participate, in fact, 51 percent had to agree, so, we needed a lot of participation. And only then was a block getting ratified, but in this case, that is not the requirement at all. This makes a heaven and earth difference. Furthermore, every node chooses whom to trust.

So, think of it as a private quorum, in the sense that every node only knows a few other nodes, and it chooses that I will trust only those nodes and not the rest. So of course, there are some restrictions, this protocol is not as general purpose as ethereum, for instance. But again, there are advantages in the sense that this relies on local knowledge, it does not rely on global knowledge. You do not have to know all the nodes, you will know a few and you trust them. So, this is modular, scalable and ultra-fast.

(Refer Slide Time: 06:47)

## Background and Assumptions

- The universe of nodes:  $V$
- A correct node **executes** as per specifications
- If a node is either **correct** or just fails (without sending any malicious messages), then it is dubbed as **honest**
- The rest of the nodes are **faulty**
- Partially Synchronous Network:
  - After a global stabilization time (GST), all **messages** have a bounded **delay**
  - The clock **skew** is **bounded**.



So, let us first discuss some background and assumptions. So, let us see the universe of nodes is  $V$ , the set of all the nodes. A node is correct if it executes as per specifications, so, it is correct meaning it is correct. So, it is the dictionary meaning of correct, which means as per laid out specifications the node executes. If a node is either correct, or it just fails without sending any malicious messages, and node just fails. That is okay that is fail Stop, it is not Byzantine failure, it

is a failed stop failure, then the node is dubbed honest. So, so where are we? We are at the point where we say that a node is correct, which means it executes as per specifications.

Or, we are saying that it can have a failed stop failure, but will not send any malicious message. So, then we will say that the node is dubbed to be honest, the rest of the nodes we are we say are faulty. So, stellar protocol can work in an asynchronous setting, it is just we cannot make certain guarantees. But, if it is a partially synchronous network, which is often the case that the clocks are loosely synchronized, we can make more guarantees as we will see. So, what happens is we assume periodic clock synchronization is known as a global stabilization event. And when the clocks are synchronized, it is a global stabilization time.

All the messages have a bounded amount of delay and the clock skew is bounded, so, that is important. The messages themselves have a bounded amount of delay, a bounded degree of delay, and the clock skew itself between two clocks is bounded. So, this is a partially synchronous network and most practical systems are partially synchronous. So, there is nothing to be surprised about this, it is not a wild assumption at all.

(Refer Slide Time: 08:53)

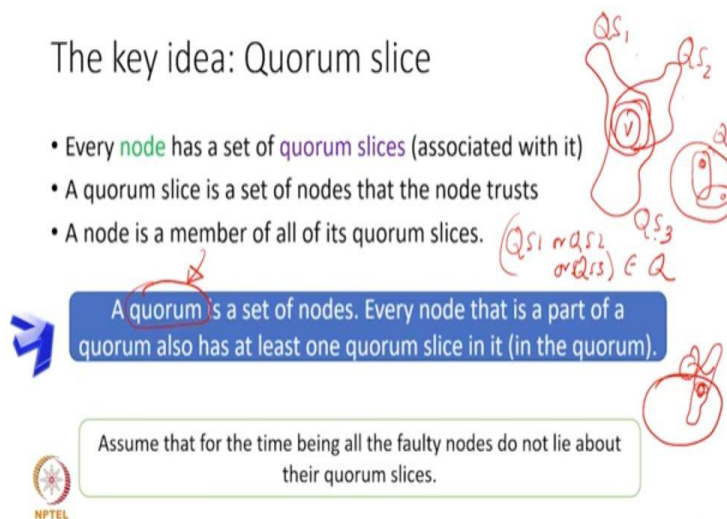
The key idea: Quorum slice

- Every node has a set of quorum slices (associated with it)
- A quorum slice is a set of nodes that the node trusts
- A node is a member of all of its quorum slices.

$(Q_{S1} \cap Q_{S2} \cap Q_{S3} \cap Q_{S4}) \in Q$

A quorum is a set of nodes. Every node that is a part of a quorum also has at least one quorum slice in it (in the quorum).

Assume that for the time being all the faulty nodes do not lie about their quorum slices.



So, what is the key idea? So, the key idea is that every nodes trusts a set of nodes. If this is one node, you create a quorum slice. So, this could be a quorum slice, this could be a quorum slice, so, of course, the node has to be a part of the quorum slices. So, you can say that in this case that there are three quorum slices. And the node is a part of all three, of course because every node has to

trust itself, and the node can be a part of multiple quorum slices, as you can see over here. And a quorum slice informally is a set of nodes that the node trusts, and node, a node is a member of all of its quorum slices.

So, this is very easy to do in the sense if you have a large network, you can say that a node will trust the its system administrator. And if there could be multiple system administrators, it could trust them. That could be one quorum slice or a node could trust the machines of some colleagues, or a node could trust the machines of some students.

So then, as you can see, we have three quorum slices over here, we are not restricted to one. A quorum is defined as a set of nodes, so, a quorum is a set of nodes, a very important concept. Every node that is a part of a quorum will also have at least one of its quorum slices, so, let us say V is a part of a quorum.

It has three quorum slices. Either, QS1 or QS2 or QS3 has to be a part of the quorum, so, either QS1 or QS2 or QS3, anyone of them has to be a part of the quorum. So, so what is the idea? The idea is that a quorum is a set of nodes, you take any node in the quorum. You will have at least one of its quorum slices, that is fully contained within the quorum.

So, let us say, if this is a node and this could have the node could have multiple quorum slices, we are not concerned about that. But, there is at least one quorum slice which is fully contained within the quorum and this is the case for all the nodes, so this is a quorum. So, what does this mean? So intuitively, what does this mean? So, quorum slice basically is the set of nodes that a given node trusts, alright, so that is a quorum slice.

Now, the moment we have a quorum and we consider every node in the quorum, so you consider this node. If one of its quorum slices is within the quorum, then at least what can you say? You can say that if the entire quorum takes a decision, if all the nodes within the quorum take a decision, then you know that at least one of its quorum slices also has taken the decision. And given the fact that the quorum slice is trusted, the node can use that fact to make a decision.

So, this node can use this fact some other node, this node can also use this fact, because it will have its quorum slice at least one which is fully contained within the quorum. So, this basically means that if the entire quorum takes a decision, whatever be the decision in the case of consensus, for instance, then all the quorums, then all the nodes within the quorum can have some degree of

relief. The relief is that some set of trusted nodes which is at least one of their quorum slices has been a part of it. So, this gives a degree of sanctity to the decision which we will see to be very important.

So clearly, in this case, the notion of the quorum is the most important concept. So, now I assume that for the time being, all the faulty nodes are not lying about their quorum slices, because faulty nodes are Byzantine faulty. So, they in principle could lie about their quorum sizes. Let us assume this is not happening, but even if it does happen, it is not an issue.

(Refer Slide Time: 12:57)

### Example of Quorums and Quorum Slices

Node	Quorum Slices
v1	{v1, v2}
v2	{v1, v2}, {v2, v3}
v3	v3
v4	v4



We can have many quorums  
 $\{v1, v2\}, \{v3\}, \{v4\}, \{v1, v2, v3, v4\}, \dots$

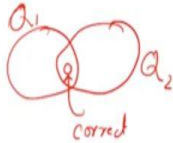


So, let us give an example of quorums and quorum slices. So, let us say v1, v1 has one quorum, slice v1, v2. V2 has two quorum slices as you can see, it is a part of both v3 and v4. So, there is some trivial quorum. So, let us say v3 is a quorum in itself, because it is the only node, and v3 is its quorum slice, which is fully contained fair enough. v1, v2 is a quorum, why?

Because v1 quorums slice v1, v2 is fully contained here. v2 has two quorum slices, one is not fully contained, but v1, v2 is fully contained. And then of course, the universe of all the nodes is also a quorum, because all the quorum slices will definitely be in it. So, as you can see for a system with different quorum slice definitions, multiple quorums are possible.

(Refer Slide Time: 13:51)

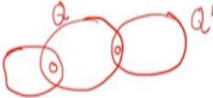
### Some Basic Requirements



- Every two quorums must intersect at a **correct** node.


This **correct** node will ensure that there is **common** agreement (consensus) across all quora.

*quorums*



**Few more definitions:**

Two nodes,  $v_1$  and  $v_2$ , are said to be **intertwined**, if they are both **correct** and every **quorum** containing  $v_1$  intersects every quorum containing  $v_2$  in at least one **correct** node.



### Example of Quorums and Quorum Slices

Node	Quorum Slices
$v_1$	$\{v_1, v_2\}$
$v_2$	$\{v_1, v_2\}, \{v_2, v_3\}$
$v_3$	$v_3$
$v_4$	$v_4$



We can have many quorums  
 $\{v_1, v_2\}, \{v_3\}, \{v_4\}, \{v_1, v_2, v_3, v_4\}, \dots$



So, now we need to come at some basic requirements that we defined what is a quorum slice? What is a quorum? But, now let us see when is it that you can guarantee consensus in this FBQS, Federated Byzantine Quorum System argument. When is it that consensus can be guaranteed? This is by and large, the most important requirement.

So, as I have said over here, you can have a lot of quorums. So, it is saying you take any two quorums  $Q_1$  and  $Q_2$ . All pairs of quorums have to have a non-empty intersection. Furthermore, one of the nodes within that intersection has to be correct, so what is the idea? The idea is you take all the quorums in the system, intersect them, the intersection has to be non-null.



And furthermore, in the intersection, you need to have a correct node, which means they need to intersect at the correct node. So, we will discuss the insights later. But, the most important point here is that when you are designing a system, in fact, the most, the trickiest aspect of designing a stellar network is to basically choose these quorums, is to basically choose these quorums such that this condition holds. And furthermore, let us say I create quorums and then there is an intersection. How do I know whether the three nodes that are intersecting? The all out of those three at least one is correct, because I may not have control over the runtime system, well, that part is correct.

If I do not have control, of course, I will have to go to other algorithms the way that we have seen in Ethereum, Bitcoin and others, which do not have this restriction, when many a time, you would have some nodes which are more fault tolerant than others, for example, the machine or the system administrator and so on. And if we have maybe four nodes in the intersection, we have to look at probabilities. So, the probability of all four of them developing a Byzantine fault might be very very low. And this is something that we want to hedge against while designing our quorums.

So, what will the correct node to secure is a fun part right. So, the correct node will essentially ensure that there is a common agreement or all the, or consensus across all. I use the term quora here for, the plural of quorums, but I should use quorums because that is what we have been using. So, the key point is that if there is one correct node in the intersection and if let us see, one quorum has decided something. So, what is the advantage of a quorum? Quorum decides a consensus value. Now, if every other quorum in the system has an intersection with it at a correct node.

Later on, let us say another quorum tries to decide something, or tries to decide something which is the opposite of what Q decided. If Q' tries to decide something, which is which goes against what Q has decided, then, the correct node at the intersection is actually going to stop it. You can never rely on faulty nodes, but the correct node is going to stop it, and it is going to say that look, a quorum has already decided something and you are going in the opposite direction. So, do not go that way and the same holds for all the other quorum. So, that is why quorum intersection is important.

So, in other words, in this case, all that you have to do, all you need to do is to convince a quorum to accept a value, then you are done. The quorum is small, you do not have to broadcast the message to all the nodes, all the nodes will ultimately get it. Few more definitions. Do nodes v1

and  $v_2$  are set to be intertwined, if they are both correct, so that is important. Whenever we are talking of nodes with some decision-making capability, both of them have to be correct that is necessary. Furthermore, every quorum that contains  $v_1$ , intersects every quorum that contains  $v_2$ , in at least a correct node.

So, this is slightly providing, this is slightly specializing the definition to a pair of nodes that look two nodes are intertwined. Number one if they are correct, and number two, if let us say every quorum that contains  $v_1$  will intersect every quorum that contains  $v_2$  at a correct node, which is a specialization of the definition. But, we will see why it is being done like this.

(Refer Slide Time: 18:44)

**Intact Set**

- Projection of the FBQS  $S$  to set  $I$

$S|I \Rightarrow S|I(v) = \{q \cap I \mid q \in S(v)\}$

Projects all the slices to a given set  $S$

A set  $I$  is an intact set if

1.  $I$  is a quorum in  $S$
2. All pairs in  $I$  are intertwined in  $S|I$ .

## Some Basic Requirements



- Every two quorums must intersect at a **correct** node.

This **correct** node will ensure that there is **common** agreement (consensus) across all quora.

### Few more definitions:

Two nodes,  $v_1$  and  $v_2$ , are said to be **intertwined**, if they are both **correct** and every **quorum** containing  $v_1$  intersects every quorum containing  $v_2$  in at least one **correct** node.



That does not define an intact set. So, the definition of an intact set is quite crucial, for the rest of the paper as well as for the proofs. So, what we do is let us start with a Federated Byzantine Quorum System FBQS  $S$ . And let us project it to set  $I$ , so let us see what that means. So, informally, what it means is, we consider only those elements in  $S|I \Rightarrow S|I(v) = \{q \cap I/q \in S(v)\}$ . so for every vertex,  $I$  take a look at all the quorum slices, so  $SV$  is basically all the quorum slices. So, you consider each one at a time.

See, if  $q$  is an element of this,  $I$  only consider those elements of each quorum slice that are also an element of  $I$ . Or basically,  $I$  compute an intersection between  $q$  and  $I$ . So, basically what  $I$  do is that if let us say this is my full set  $S$ , and  $I$  have this  $I$  as the set on which  $I$  am projecting. So, what  $I$  do is  $I$  take a look at every vertex  $v$ , and for this vertex  $v$ ,  $I$  take a look at all of its quorum slices. Let us say the vertex  $v$  is over here and these are its quorum slices.  $I$  only consider that part of the slices which lie within  $I$ , so this is the projection. So, a set  $I$  is an intact set in a such kind of a set  $I$  is an intact set, if it follows two properties. The first is  $I$  itself is a quorum in  $S$ .

So, what we do is that we take the Federated Byzantine Quorum System  $S$ , and in that let, us say  $I$  choose one quorum, and that quorum is  $I$ . So,  $I$  call that  $I$  as an intact set if all pairs of vertices within  $I$ , let say this vertex and this vertex are intertwined, are intertwined. So, go back to the previous slide to understand what intertwining means, which means both the nodes are correct. And they are intertwined in  $S$  projected to  $I$ , which basically means that there is some quorum slice

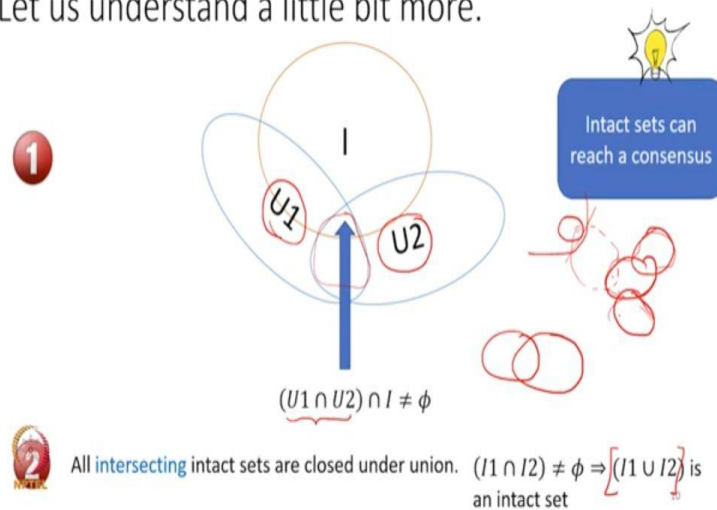
of this which completely lies within I. And there is some quorum slice of this which completely lies within I.

And the fact that they are intertwined basically means that they intersect at the correct node. So of course, the way that I have drawn, they are not intersecting but you get the idea. So, the idea is that if I consider if my new universe, everything is limited to the set I, and for every quorum slice, if I only consider those elements that are a part of set I, then I will consider I to be an intact set, if I was a quorum in S, so that is the first condition. The second is if I project S to I, in the sense I only look at all the nodes and quorum slices of S that belong that have a nonzero intersection in I.

Then, any two pairs of nodes in I are intertwined in this projection. So, this basically means that this is kind of a self-contained set, that is the reason it is being called an intact set. It is it is a self-contained set, where if I take any two nodes and I basically take their quorum slices, then there is an intersection at a correct node. And of course, these completely belong within I, so, it is like a self-contained universe, so this is an intact set.

(Refer Slide Time: 22:33)

Let us understand a little bit more.



So, let us understand a little bit more and look at a few theorems with regards to intact sets. So, why do we look at intact sets in the first place? The reason we look at intact sets in the first place is basically because intact sets can reach a consensus, using the stellar algorithm. Independently, intact sets can reach a consensus. And of course, if the entire network is an intact set, then the

entire network can reach a consensus. So, the first theorem is like this the, let us consider a quorum  $U_1$ , and a quorum  $U_2$ , and let us say both of them intersect with the intact set with an intact set  $I$ .

If that is the case, then  $(U_1 \cap U_2) \cap I \neq \phi$ . So, this is an important result that is used in the proofs of the stellar protocol. So, I would encourage all of you to take a look at the papers and understand the proof of the protocol.

But, the basic idea is that if let us say there are two quorums, which intersect the intact set, then the intersection of those quorums also intersects the intact set, so the intersection is not disjoint. Furthermore, if I take two intact sets, for instance, this is one intact set, this is one more intact set, and let us say that they are intersecting.

So, they are not separate, these are intersecting, then, they are closed under union which basically means that if there is a common node between them, then the union of intact sets is also an intact set. So, this basically means that in any kind of a network unless a node is totally disconnected, if you have intact sets of this type, then essentially all of this is an intact set and all of this can reach a consensus. So, basically the only intact sets that will not reach a consensus, along with the rest of the intact sets or if something is totally cut off. So, what does it mean to be totally cut off in the sense? It does not have any intersection with the rest of the intact sets.

So, that is when this is totally cut off. Because, what we also get to see is that by the first theorem over here, let us assume that there is some other intact set like this, such that  $(U_1 \cap U_2) \neq \phi \Rightarrow (U_1 \cup U_2)$ . So, you know that part is clear that these two have to intersect, and if they intersect, the intact set will further grow.

So, that is the reason either you have a separate kind of disconnected component over here, which does not intersect any intact set. So, so then, of course, it is separate, otherwise, if there is any intersection, you can consider this to be a larger intact set. And that can independently reach consensus as per the stellar algorithm which is something that we will see.

(Refer Slide Time: 25:59)

## Non-Blocking Byzantine Consensus for Intact Sets

- Given a maximal intact set /



1. **Integrity** → No correct node decides twice
2. **Agreement** → No two nodes decide differently
3. **Weak validity** → If all nodes are **honest**, then the value that is decided is one of the proposed values
4. **Non-blocking** → If all **malicious** nodes **stop**, then all the nodes ultimately come to a consensus.



By the FLP protocol, we cannot guarantee termination if malicious/faulty nodes are active.

11

## Non-Blocking Byzantine Consensus for Intact Sets

- Given a maximal intact set /

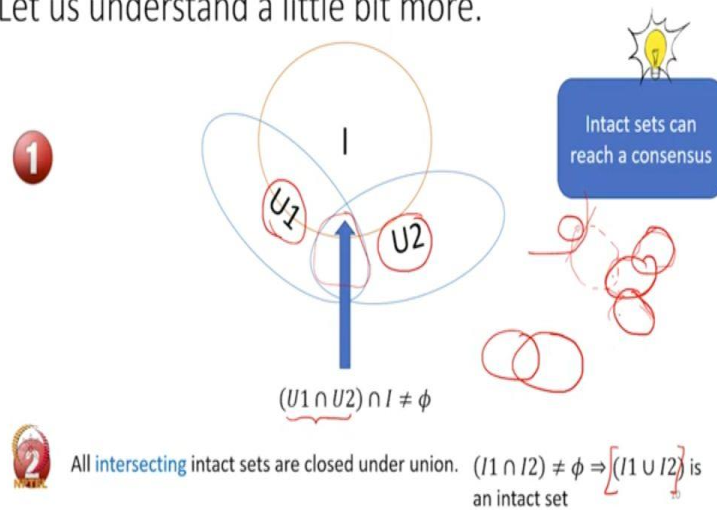
1. **Integrity** → No correct node decides twice
2. **Agreement** → No two nodes decide differently
3. **Weak validity** → If all nodes are **honest**, then the value that is decided is one of the proposed values
4. **Non-blocking** → If all **malicious** nodes **stop**, then all the nodes ultimately come to a consensus.



By the FLP protocol, we cannot guarantee termination if malicious/faulty nodes are active.

11

Let us understand a little bit more.



So, let us now look at the idea of non-blocking consensus. So, I will discuss what is non-blocking over here, but basically in the presence of nodes that could have Byzantine failures. So, let us look at this for an intact set, and given the fact that we have discussed this union and intersection business over here.

Let us consider a maximal intact set I, where clearly, we cannot go it further, in the sense, there is no other intact set with an intersection such that we can grow it. So, the scope of growing it is not there, it is a maximal intact set. So, what we say is that these four properties will be satisfied by Stellar, and in fact should be satisfied by any consensus algorithm that deals with such kind of intact sets constructed in this manner.

So, what are the properties? The first is integrity. So, integrity basically means that no correct node decides twice. So, this follows from the definition of consensus that if you are a correct node, once you have said that this is the consensus value that remains, you do not change your mind. Agreement again follows from the definition of consensus, it basically means that no two nodes decide differently. Of course, weak validity, which basically means that if all the nodes are honest, then the value that is decided is one of the proposed values of course. So, in this case, the idea basically is that if we are looking at honest nodes, then of course one of the value has to be one of the proposed values.

So, this again follows from consensus. What does not follow? But, what technically makes sense is that let us say there are no malicious nodes, or all the malicious nodes just stop. Then, the

protocol is guaranteed to terminate because in this case, this will not be against the FLP result. Because, what does the FLP in a result or protocol say? We cannot guarantee termination if any malicious or faulty nodes are active. But, the point is that in this case, malicious and faulty node if they are not active, you should be in a position to guarantee termination of the consensus algorithm.

So, the first three kind of follow from consensus and the fourth one follows from the fact that you would like to ensure our minimum amount of liveness in the system. So, we have these four properties, which are you the consensus guarantees, you would like to have at least for maximal intact sets.

(Refer Slide Time: 28:49)

### Key part of the algorithm: Federated Voting

• Properties

Vote (a) → Deliver (a')

For correct nodes

*{ 2-phase Paxos*

1. No duplication → Every correct node delivers at most one voted value (s)
2. Totality → If a node in I delivers a voted value, every node in I delivers a voted value (L)
3. Consistency → If two intertwined nodes deliver a and a' resp., a = a' (same value)
4. Validity → If all nodes vote for a, they eventually deliver a

Reliable Byzantine voting

12

### Key part of the algorithm: Federated Voting

• Properties

Vote (a) → Deliver (a')

For correct nodes

*(FV) → Consensus*

1. No duplication → Every correct node delivers at most one voted value (s)
2. Totality → If a node in I delivers a voted value, every node in I delivers a voted value (L)
3. Consistency → If two intertwined nodes deliver a and a' resp., a = a' (s)
4. Validity → If all nodes vote for a, they eventually deliver a (s)

Reliable Byzantine voting

12



## Non-Blocking Byzantine Consensus for Intact Sets

- Given a maximal intact set  $I$

1. **Integrity** → No correct node decides twice
2. **Agreement** → No two nodes decide differently
3. **Weak validity** → If all nodes are **honest**, then the value that is decided is one of the proposed values
4. **Non-blocking** → If all **malicious** nodes **stop**, then all the nodes ultimately come to a consensus.



By the FLP result, we cannot guarantee termination if malicious/faulty nodes are active.

11

So, the key part of the algorithm is federated voting, which draws inspiration from two phase commit. So in this case, we have two phases, so it is called vote and delivered, but later on, we will see it is also called prepare and commit, it does not matter. There are two phases. So, I would like to request the viewers to look at two phase commit, to also look at the Paxos algorithm, because that also has two phases. And it is quite similar to this part, where you vote and you deliver. So, for the correct nodes, federated voting would like to ensure these four properties, so this is also known as reliable Byzantine voting.

So, the reliable voting basically means what? So, in this case, mind your voting is not consensus. Voting is a process, which provides some safety guarantees very little liveness guarantees, but let us nonetheless see what it is. So, the idea is that nodes vote for a value. And let say if a quorum successfully votes for it, then it is delivered, so, there is no duplication. In the sense, every correct node would deliver at most one voted value. So basically, as I said, delivery is the second phase, voting is the first phase. So, in the second phase, you will deliver at most one voted value, not more than that.

Then, you have totality, if a node in  $I$ , where  $I$  is an intact set delivers a voted value, every node in  $I$  delivers a voted value. So, basically what does that mean? It basically means that you have, so this is a liveness guarantee, which basically says that look if one of the nodes. In one of the correct nodes in an intact set terminates and delivers a voted value, then all the other nodes also will terminate, so, this is a liveness guarantee. The first one is of course, a safety guarantee. So,

consistency basically means if two intertwined nodes which is like any two nodes in an intact set, because all pairs are intertwined.

If they deliver a and a dash respectively, then a is equal to a dash, in the sense it delivers the same value. So, this is just a sophisticated way of saying that all the nodes in an intact set deliver the same value. So, the way that it is written because I have kind of paraphrased what is there in the paper, that if two intertwined nodes delivery a and a dash, then a is equal to a dash. So, which basically means that once a node delivers, everybody delivers, and everybody delivers the same value, that would be the sum total of 2 and 3. So, this is like what you would like your federated voting algorithm to ultimately satisfied, and then of course, we have validity.

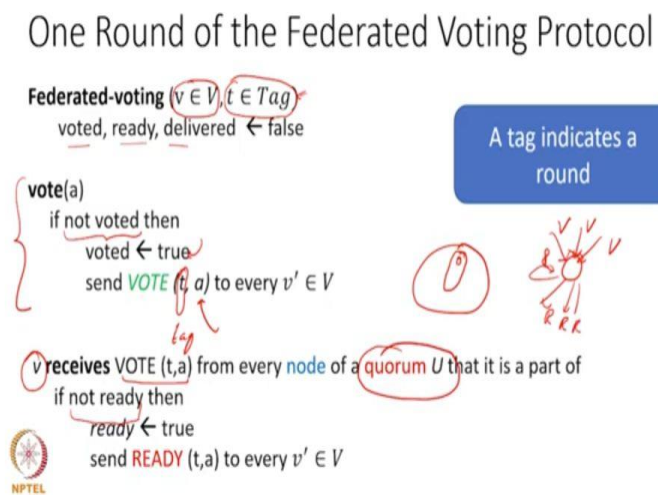
So, validity basically means that, so this again is this was again, you know a safety condition. So, this is one more safety condition the last one, which essentially says that look, you deliver only what you are voted for. So, you do not deliver something else, you deliver only what you are voted for. So, if you look at it, so no duplication was what that on only one of the values, once node delivers, it cannot kind of vote for any other value, or it cannot also deliver any other value. So then, this was a safety condition, the last one as we have just seen as safety condition.

And the third one is also a safety condition basically, because it says that all the intertwined nodes in an intact set which is all the nodes, deliver the same value. So, the liveness condition is basically like this that if one node delivers, everybody delivers is only liveness condition. So, what we see over here is that federated voting is again taking some aspects of consensus, it is again, of course, it has a different liveness condition as compared to this slide. But, the idea basically is that the first slide was consensus for intact sets. And this is a property of federated voting, which will ultimately take us towards consensus for intact sets of federating federated voting, as such is a step. So, Fv is a state step, and Fv will take us towards consensus.

So then, what do we have now? So, what we have now is let us look at the federated voting protocols. As I have said, it is a two-phase protocol. There first you vote and then you deliver. But again, you are guaranteed to deliver only a single value, and the entire intact set either delivers or does not deliver. So, you have this liveness guarantee, and if it delivers, it delivers the same value, otherwise, it does not deliver at all.

And of course, if you in the rare case where everybody votes for the same value, it is only that value that gets delivered. So, the way that you need to think about it is that it is a two-phase algorithm of vote and deliver, where the voting process may actually not successfully complete. If that is the case, nobody delivers. But if it successfully completes, then everybody is ultimately guaranteed to deliver the same value.

(Refer Slide Time: 34:36)



So, this is essentially the crux of federated voting, which the algorithm on this page will convince you that it works. So, what we do is that we do a  $(v \in V, t \in Tag)$  essentially indicates around, because the way that the master protocol works is that we have many rounds of voting. We do not have just one round of voting, but we have many many rounds of voting, and the tag indicates the round of voting. Then, we have this state variable voted, ready, delivered all start with false, so let us start with the vote. So of course, if any node wants to vote for a value, it can initiate, or we will see that there are other conditions also.

But, at least let us say that a node decides to initiate that it needs to vote for a certain value. Say for that specific round, for that specific tag actually, if it is not voted, then it will vote only one. So basically, the idea is that for a given tag, so I will use tags and round interchangeably. So, for a given tag or round, you can vote only once. The moment you vote, the variable voted becomes true. Then, you send the vote message to every vertex, including yourself in the vertex set. And

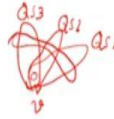
what would that include? That would include the tag number and that would include the value that you voted for.

And mind you, you are allowed to do this only once, not multiple times only once, because of this variable over here. So, the moment you receive a vote  $t$ , a message, it is a moment node  $v$ . So, here node  $v$  is always a node that is processing either sending or receiving. The moment it receives a vote  $t$ , a message from every node have a quorum, that it is a part of right. So, here the idea of a quorum is coming, so, you can go several slides back. But, the basic idea of a quorum was that every node in the quorum of course belongs to the quorum, and one of its quorum slices definitely belongs to the quorum.

So, if a quorum votes for  $v$ , and of course we have a quorum intersection and so on. If it is not ready, so again, we have a ready variable, if I am not ready, which means ready is false. I set ready to true, and then they I send a ready message to every other node including myself. So, go back to two phase commit. In two phase commit and Paxos, we were doing something quite similar. So, the first round you send vote messages, once you get vote messages from a quorum, so once you get these vote messages from a quorum. Then, what you do is that you begin the second phase and then you send a ready message to everybody else, including yourself.

(Refer Slide Time: 37:40)

## FV Protocol (Part II)



A **v-blocking set** sends a **READY (t,a)** message

if **not ready**

$ready \leftarrow true$

send **READY (t,a)** to every  $v' \in V$

A set is v-blocking if it overlaps with every quorum slice of v

Can send a **READY** message for value that it has not voted for

Received **READY (t, a)** from a **quorum U** that v is a part of

if **not delivered**

$delivered \leftarrow true$

**deliver (a)**

Similar to 2-phase commit



## One Round of the Federated Voting Protocol

**Federated-voting** ( $v \in V, t \in Tag$ )

$voted, ready, delivered \leftarrow false$

A tag indicates a round

**vote(a)**

if not voted then

$voted \leftarrow true$

send **VOTE (t, a)** to every  $v' \in V$



**v receives VOTE (t,a)** from every **node** of a **quorum U** that it is a part of

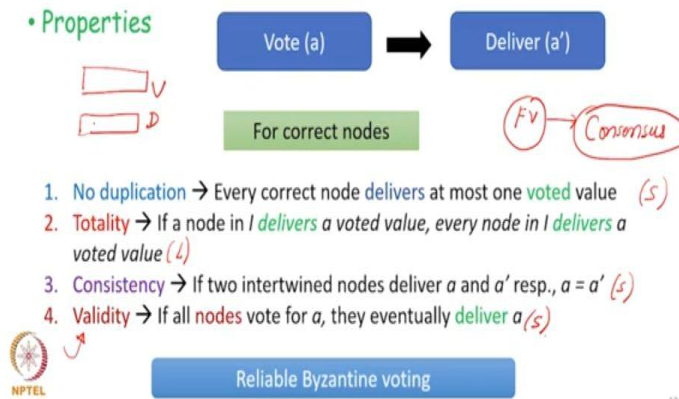
if not ready then

$ready \leftarrow true$

send **READY (t,a)** to every  $v' \in V$



## Key part of the algorithm: Federated Voting



In the second set, so so here is also a fun part that the ready message is also sent. If a v-blocking sets, I will describe what is a v-blocking set. So, let us say this is vertex v, and these are its quorum slices. A v-blocking set is a set which overlaps with every quorum slice of v, so it is like this, this is a v-blocking set. So, if I am a node v and a v-blocking set, which as I have said these are my quorum slices.

So, v-blocking set pretty much overlaps with every quorum slice in a sense, it has a nonzero intersection. So, if it sends a ready message to me, so if it sends a ready message back to me and I have vertex v, then what do I know? What I know is that at least, for all my quorum slices, one of the nodes has changed his status from not ready to ready, and it will not change its status ever again.

So, from not ready, it has become ready which means it is done. It has made up its mind, it has voted as well as, as well as it has changed its status. So, then whatever ready message it sends, remind you this may be different from what you originally voted for. It may be this value of a may be different, but that is okay. It means that whatever I originally voted for, did not find adequate support. But, now given a v-blocking set, which means pretty much one entry from every quorum slice of mine, if that is committed, it has changed its state from not ready to ready. So, that will remain it is kind of final.

So, what I will do is if I am not ready again, because I changed my state only once. For a given round, I changed my state from not ready to ready only once. So, the moment I do that, if I am not

ready, I become ready, and then again I broadcast the ready message to every vertex. So, what you see over here is that a ready message can be propagated, of course, the first node to send a ready message has to receive votes from a quorum, so that is given. But after that, a ready message can be propagated if a  $v$ -blocking set has sent the same ready message to a node  $v$ . And in this case, of course, it can be different from his voted value, but it will still propagate it.

And then what happens is that for node  $v$  if it receives ready messages from a quorum, which means a full quorum is sending a ready message again similar to two phase commit. If it has not delivered, then it will say delivered to true and deliver the message. So, this is basically what delivery in an intact set means, which means it has passed both the rounds. So, the first round is not guaranteed to terminate mainly because it is possible that an entire quorum may not agree, you have a 50-50 situation or even worse, when multiple values are being proposed. So, that is why I said the first set, the first round is not really guaranteed to terminate.

And we will see timeout mechanisms for that, but assuming it does, then of course, you can guarantee beautiful things, and what you can guarantee is that the same value is delivered, number one. You can guarantee everything that federated voting guarantees, no duplication, you can clearly see that you deliver only one voted value, because only once your state changes from not delivered to delivered. If a node in  $I$  delivers a voted value, it is then it is guaranteed that it has support of a quorum. So, no other value can be delivered because of quorum intersection.

So, you will you take any two quorums, and you will have one correct node that is in common and that cannot lie. So, given that it has already committed to a given value, it will keep that commitment, so nobody else can deliver anything else. So, that proves property three as well. And furthermore, if all the nodes vote for a single value, then that has to be delivered, because if you see our algorithm, there is no way to introduce any other spurious value, extraneous value in the middle. So, this is like a two-phase commit for as I said the first phase need not finished, but if it does, then you have this beautiful safety and as well as liveness properties of federated voting.

(Refer Slide Time: 42:30)

## Insights

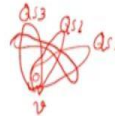


- The first READY message is **received**, only after the same VOTE message is received from the entire **quorum**
- Then it is **propagated** (a v-blocking set is enough)
- Termination is not **guaranteed**:
  - Otherwise, it will **violate** the FLP result
- However, once one message is **delivered**
  - The rest get delivered in finite time (for correct nodes), if we assume **bounded** clock skew



15

## FV Protocol (Part II)



A **v-blocking set** sends a **READY (t, a)** message  
if **not ready**  
ready  $\leftarrow$  true  
send **READY (t, a)** to every  $v' \in V$

A set is v-blocking if it overlaps with every quorum slice of v

Can send a READY message for value that it has not voted for

**Received** **READY (t, a)** from a **quorum U** that v is a part of

if **not delivered**  
delivered  $\leftarrow$  true  
**deliver (a)**

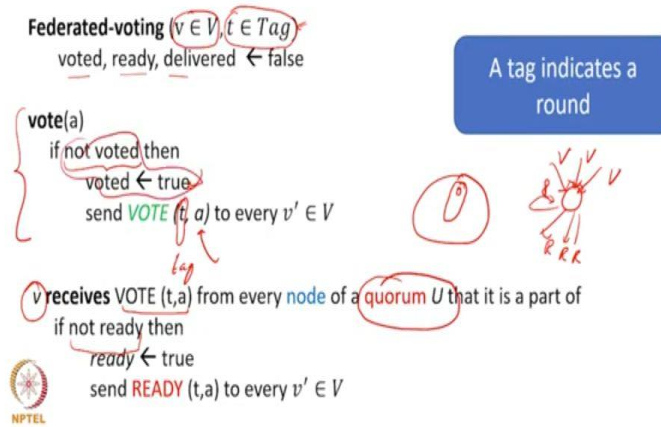
Similar to 2-phase commit



16



## One Round of the Federated Voting Protocol



Now, let us look at some of the insights. So, the insights anyway, we have discussed, but still I am repeating them. The first ready message is received, only after the same VOTE message or the vote message for the same value. It received from an entire quorum, this is received from a full quorum, so only when a full quorum sends a ready message to a single vertex  $v$ .

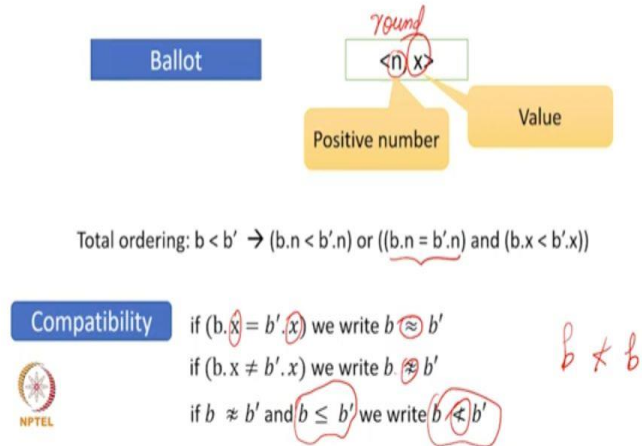
And then, so then only. So, when a full quorum sends a VOTE message for the same value and a vertex  $V$  receives all of that, then, it initiates the ready message. And then of course, it is propagated by other  $v$ -blocking sets. So, via this line over here, it is propagated. Termination is not guaranteed, because if it would be that would violate the FLP result.

And in particular, the first phase is the one that causes the problems primarily. However, once a message is delivered, what does that mean? What that means is that a full quorum has voted for the message, it is not going to vote for any other message. So, that is being ensured with this line with this. So, if it is not going to vote for any other message and no other message can be delivered. So, we are at least assured that it is only this message which is going to be delivered.

And for all the correct nodes, they will get the delivery in finite time, if we assume bounded clock skew. So, we assume that we have a partially synchronous system and all the nodes are active in the sense executing this algorithm. Then within a finite amount of time, which can also be made bounded also. If one message is delivered all the messages will be delivered to the intact set.

(Refer Slide Time: 44:35)

## Some More Terminology



So, let us now take this to the level of a ballot. So, let us now create a consensus protocol or rather a blockchain out of this. So, what we want is that we want to use federated voting as the primitive, and then we want to build on top of it. So, let us introduce the notion of a ballot, where ballot is a tuple of a positive number, which keeps monotonically increasing. So, this is the round number. The tag that we discussed and  $x$  is the value, value that is being voted upon. So, we see that  $b < b'$ , if either the number  $b.n < b'.n$ , or if the numbers are the same, but the value is less than that.

So, this the value is used as a tiebreaker only when the numbers are the same, otherwise we go by the number. So, two ballots are set to be compatible if their values are the same, so that makes sense. This is the symbol that we have for compatibility and two values are not compatible. Again, this symbol if the values are not the same. So, the term that we will use assume that  $b \leq b'$ , in the sense the numbers are less than equal to that. So, say it will follow the same notion of equality and less than over here. So, if we have this, and the ballots are incompatible, so it is less than equal to this and the ballots are incompatible, we write  $b \notin b'$ .

So, hopefully this is clear that if the ballots are not compatible with each other, then we basically write that, and let us say is less than equal to the other, then we write it in this fashion. So, actually, we can write less than as well because if the ballots are incompatible, we will never have strict equality. But, we could have equality in terms of numbers, but the values will not be the same. So, that is being captured with the less than or equal to over here. But nonetheless, the term is clear to

the moment we have something like this, it is essentially indicating an incompatibility. And along with that, it is also indicating that  $1 \leq$  the other, in terms of these numbers that we have defined over here.

(Refer Slide Time: 47:17)

### Abstract Consensus Algorithm for node $v$

```

ballots  $\leftarrow$  array of FV processes (specific to each ballot) ( $\infty$ )
candidate, prepared  $\leftarrow$  (0,  $\phi$ )
round  $\leftarrow$  0

propose(x)
  candidate  $\leftarrow$  <1,x>
  for all  $b' \prec$  candidate do ballots[b'].vote(false)

when for every  $b' \prec b$  ballots[b].delivered(false) and prepared < b
  prepared  $\leftarrow$  b
  if candidate  $\leq$  prepared
    candidate  $\leftarrow$  prepared
    ballots[candidate].vote(true)
  
```

NPTEL

Ensure all incompatible ballots with a lower number are voted out

Ensure that the candidate ballot is accepted by all

### Abstract Consensus Algorithm for node $v$

```

ballots  $\leftarrow$  array of FV processes (specific to each ballot)
candidate, prepared  $\leftarrow$  (0,  $\phi$ )
round  $\leftarrow$  0

propose(x)
  candidate  $\leftarrow$  <1,x>
  for all  $b' \prec$  candidate do ballots[b'].vote(false)

when for every  $b' \prec b$  ballots[b'].delivered(false) and prepared < b
  prepared  $\leftarrow$  b
  if candidate  $\leq$  prepared
    candidate  $\leftarrow$  prepared
    ballots[candidate].vote(true)
  
```

NPTEL

Ensure all incompatible ballots with a lower number are voted out

Ensure that the candidate ballot is accepted by all

*Handwritten notes:* A diagram shows a vertical list of ballot numbers  $b_1, \dots, b_n$ . An arrow labeled "Candidate" points to the first element. A bracket on the right side groups the elements from  $b_1$  to  $b_n$  and is labeled "prepared".

So, now let us discuss an abstract consensus algorithm. The reason I am using the term abstract is basically because it assumes infinite resources, which is not the case. So, we will discuss a version of this algorithm that uses finite resources. But, at the moment, let us proceed with the version that uses infinite resources. So, let us say that specific to each ballot, we have a federated voting process. So, given the fact that we do not place any bounds on this array, in principle this could be

infinite, so that is why I say it is an abstract algorithm, or a theoretical algorithm. But, very soon we will place bounds on it.

So, an array of federating voted voting processes are in ballots. They maintain two state variables candidate and prepare initialize them to 0 and null. And of course, the first round is round 0 and this keeps on increasing monotonically. So here again, we will have two phases, so we have taken a lot of inspiration from the two-phase voting protocols.

So, here also we will have two phases in the ballot algorithm, so this is the consensus algorithm, so we start with a candidate. So, the candidate is basically a tuple of round one, and the value  $x$  that is being proposed. As you can see, it is the same value that is being proposed, so, here is what we need to do.

So, here is the fun part of where the federated voting aspect is being used as a function. So, for all the ballots for all the values of  $b'$ , which are less than an incompatible with candidate, we have to ensure that everybody agrees that you vote for false. Say, vote for false basically means that all of those ballots and their values are voted to be false, in the sense those ballots everybody agrees that we will not accept it, or this will not be our consensus value.

So, you are essentially voting false on them. So, this basically means that whatever ballots are not compatible with your candidate, and which are which have a lower number, you first invalidate all of them, and then only we try to validate this candidate, otherwise we do not, so this also makes sense.

So essentially, you are cutting out the possibility of any candidate which is which has a lower number than you, which is less than an incompatible to you. Because, it is compatible, you do not care is the same value. But if it is incompatible, you are cutting out the chances of it ever being chosen as the consensus candidate, because we are waiting for all of them to vote false.

Let us now look at the second part. So now, what have we done? We have said we are given an order that all the ballots which are less than and not compatible with the current candidate have to be invalidated first. So, when all of them are invalidated, so of course here I am changing the connotations.

So, when  $b$  is always our current candidate and  $b'$  is always the other one. So, in that sense, it still remains the same, but of course, it should be clear that these are separate algorithms. So, when every  $b'$  which is lower and incompatible with  $b$ , its ballot delivers a false, which means that it is successfully invalidated.

So again, as I said, this process is not guaranteed. Because, if this process was guaranteed, we would have used this as our consensus algorithm. But, once all the lower and incompatible ballots are successfully invalidated, and we have not prepared a ballot which is as high as the current one.

So, that is also important that it is not as high as the current one, because  $b$  is always the current ballot. So then, what we do is that we set that to prepare, so this is the same as if not voted, then vote, it is the same logic. And what we see is that if let us see, so this is prepared. So,  $b$  in this case, is basically a ballot less than which we have received all the false messages. So, the point is we set that to prepare, because that is the largest value of the ballots that we have prepared. So, what is preparing in this case mean? Preparing in this case means that all the ballots which are lower and incompatible, all the  $b'$ 's have been invalidated.

So, once we have such a  $b$  decided to prepare if that is the largest such value, and then we look at the relationship between candidate and prepared. So, as long as we have prepared a bunch of ballots, which are at least equal to the candidate or greater than it, then we are fine. Then, it is a valid candidate, which means the candidate is good to go.

Because, if this is the level of the candidate the round and if this is the level of prepared, and if less than prepared, everything which is incompatible has been invalidated. Then, it also means that anything less than a candidate also has been invalidated, and basically the candidate is good to go.

So, we said candidate as prepared, we set it as prepared, because it is of course, you are assuming that the same value is being proposed, so candidate and prepared are compatible. And the reason that I say that is basically because the same node will only propose a single value in this case. And if let us say less than prepared, everything is invalidated, less than candidate also holds. So, we set candidate as as prepared, say the sense we kind of boost its rank up and set prepared, take the value of prepared and set that as candidate.

And then I set ballots candidate dot vote true. So, what does this mean? So, what this basically means is that once I am sure that all lower and incompatible ballots are all invalidated, then I try to convince everybody that look, here is a candidate and you vote true for it. So, this kind of begins the second phase.

(Refer Slide Time: 54:01)

### Consensus Algorithm – II

when ballots[b].delivered (true)  
 decide (true) *consensus*

Once, one true message is delivered, we know that the rest will also be delivered (property of intact sets). Declare victory

---

There exists a quorum  $U$  ( $v$  is a part of it) s.t. for all  $u \in U$   
 There is a ballot  $b_u$  (associated with  $u$ ) s.t.  
 1. round  $< b_u, \pi$   
 2. Either, {VOTE,READY} ( $b_u, \text{true}$ ) has been received from  $u$  OR received {VOTE, READY} ( $b', \text{false}$ ) from  $u$  for every  $b' \in [z_u, b_u), z_u < b_u$  *left behind*

A quorum is pretty much deciding for another round.

round  $\leftarrow \min \{b_u, \pi \mid u \in U\}$  *start\_timer*  
 Upgrade the round to be in sync with the quorum

### Abstract Consensus Algorithm for node $v$

ballots  $\leftarrow$  array of FV processes (specific to each ballot)  
 candidate, prepared  $\leftarrow (0, \phi)$   
 round  $\leftarrow 0$

propose(x)  
 candidate  $\leftarrow \langle 1, x \rangle$   
 for all  $b' \prec$  candidate do ballots[b'].vote(false) *FV*

when for every  $b' \prec b$  ballots[b'].delivered(false) and prepared  $< b$   
 prepared  $\leftarrow b$   
 if candidate  $\leq$  prepared  
 candidate  $\leftarrow$  prepared  
 ballots[candidate].vote(true)

Ensure all incompatible ballots with a lower number are voted out

Ensure that the candidate ballot is accepted by all

So, when all the ballots have delivered a true in the sense all of them agree on the candidate, we decide that it is true and this finishes the consensus, so, this means consensus has been achieved. So, once one true message is delivered from earlier theorems, we know that the rest will also be delivered because of the property of intact sets.

So, we do not really have to wait for all the delivered messages to reach their destinations, the moment we get one we declare victory, and we see consensus has been achieved. So, as you can see, we use federated voting, and even the top-level algorithm is also federated voting.

So essentially, it is basically a two-level hierarchical federated voting, where our aim is to kind of cancel out all the ballots with a lower number which are incompatible. Once they are all cancelled out, then what we do is we say that look here is the candidate. So first, we vote false with this is canceling, this process is canceling, and then we vote true which basically means now all of you agree on this candidate. So, as I said, the process of federated voting is not guaranteed to complete. So, it is not guaranteed to complete which basically means that there could be the first phase could get stuck.

So, that is why we need a timeout mechanism. So, what we, so here is what we do? Assume a node  $v$  and a quorum  $U$  where  $v$  is a part of the quorum. If let say for all nodes in the quorum, so all nodes in a quorum that  $v$  is a part of, there exists a ballot  $b_u$ , such that. So,  $b_u$  is a ballot which is associated with  $u$ . And what is  $U$ ?  $U$  is a node in the quorum, so that we do this.

This is the quorum, this is a  $u$  with it, and associated with it is a ballot  $b_u$ . So, there has to exist a ballot  $b_u$ , such that the current round is less than the round of the ballot, so,  $b_u$ . n. so, we have either received a vote or ready message for this ballot is a true, or we have received a vote or ready message. But, for some ballot  $b'$  which is false, where  $b$  dash is defined as follows.

So, which basically means that there is an open interval that ends at  $b_u$ , in a sense this is  $BU$ , and these are these are all the ballots which are less than it. So, there is an open interval in the sense it does not contain  $b_u$ , but it contains all of these. So, as long as for each of these, so for each of these, so each of these iterated is  $b'$ . So, as long as for every  $b'$  which is a part of this open interval, we have received a false, in the sense that we this has been invalidated. See if that is the case, then what we do is we realize that we are in trouble. So, why do you realize you are in trouble?

When you realize you are in trouble because a ballot exists, whose current round is high, which is higher than the round of the current node which is  $v$  and  $v$  is a part of  $b_u$ . So, it has clearly seen other ballots whose current round is quite high, that is the first point. And the other is it has also received messages from them, to either confirm the correct ballot or basically a prepared message,

so there is basically for all the ballots which are lower than it. It has received, prepared, you know, these prepare messages, so vote ready are part of that repairs. I will show you where once again.

So, they are part of this, so vote and ready are part of this federated voting process. So, as a part of this federated voting process, it has received a lower number ballots from an open interval which is less than  $bu$ , saying that look, all of you are false. So, either it has received a vote of true or it received a vote of false for these, which basically means that there is some ballot which is active at a higher round. And all the other ballots in the quorum are higher. They have also been sending messages and we have gotten those messages, which means that I am somehow left behind.

So, what is the sum total of this argument? The sum total of this argument is that I have been left behind. So, I am behind other nodes, I am left behind. So, what I do now is that I increment my round. So, I take all the nodes in the quorum and I take the minimum  $n$  value, which is a minimum value in their ballots that they are sending.

And I set that to my current round and I also started a timer, so this is the timeout mechanism that I was talking about. But, what you do is you realize that given that all the nodes are not making progress at the same rate. a node basically realizes that it has fallen behind, mainly because the round numbers that all the nodes in the quorum including it are using, is much higher than the value of the round state variable. It is round state variable, so, that is why it increments its round, and starts the timer.



(Refer Slide Time: 59:45)

## Consensus Algorithm – III

Quite similar to propose(x)

At a timeout:

if prepared is uninitialized, candidate  $\leftarrow$  <round + 1, candidate.x>  
else candidate  $\leftarrow$  <round+1, prepared.x>

for all  $b' \prec$  candidate do ballots[b'].vote(false)

- lower numbered (incompatible) ballots X
- agree on the candidate ✓



19

## Abstract Consensus Algorithm for node v

ballots  $\leftarrow$  array of FV processes (specific to each ballot)  
candidate, prepared  $\leftarrow$  (0,  $\phi$ )  
round  $\leftarrow$  0

propose(x)

candidate  $\leftarrow$  <1,x>  
for all  $b' \prec$  candidate do ballots[b'].vote(false)

Ensure all incompatible ballots with a lower number are voted out

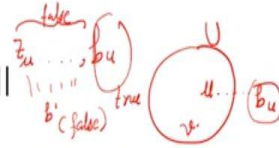
when for every  $b' \prec b$  ballots[b'].delivered(false) and prepared  $\prec$  b

prepared  $\leftarrow$  b  
if candidate  $\leq$  prepared  
candidate  $\leftarrow$  prepared  
ballots[candidate].vote(true)

Ensure that the candidate ballot is accepted by all



## Consensus Algorithm – II



when ballots[b].delivered (true)

decide (true)

*consensus*

Once, one true message is delivered, we know that the rest will also be delivered (property of intact sets). Declare victory

There exists a quorum  $U$  ( $v$  is a part of it) s.t. for all  $u \in U$

There is a ballot  $b_u$  (associated with  $u$ ) s.t.

1. round <  $b_u.n$

2. Either, {VOTE,READY} ( $b_u$ , true) has been received from  $u$  OR

received {VOTE, READY} ( $b$ , false) from  $u$  for every  $b' \in [z_u, b_u), z_u < b_u$

*left behind*



round  $\leftarrow \min \{b_u.n \mid u \in U\}$   
start\_timer  $\rightarrow$

Upgrade the round to be in sync with the quorum

18

So, timeout what happens? So basically, you can see the paper, so in detail they have explained what it means. What is the connotation of the timer? So here of course, so then, what you do is that if prepared is uninitialized, then you start is this similar to the proposed function, so, I will show you the proposed function. Here is what you do. You vote on a candidate is you basically prepare. So, you increment the round, round + 1, and with candidate's value, you start. Otherwise, if you are already prepared, then so as I said, it is never the case that you propose different values.

So, whatever value you prepared with prepared.  $x$ , you just increment the round number and set that as the current candidate. Then as you can see, the second line is exactly the same as this line over here, which is you go forward and invalidate all the lower numbered ballots. So, if I were to summarize here is the key idea, you take all the lower numbered and incompatible ballots, you invalidate them.

And then once that is done, you try to make everybody agree on the candidate, that is the idea. And if let say for some reason, the protocol gets stuck which it can, because anyway, federated voting liveness is an issue, nothing you just set a timer. So, the timer will basically ensure that the system stabilizes in the sense that all the messages we have sent are received. And after that, you just increment the round and start afresh.

(Refer Slide Time: 62:01)

## Some Basic Properties



- If some node **decided** on a ballot, some node must have **prepared** it
- If a node has **prepared** a ballot, some node must have **proposed** the value
- All nodes in the intact set **decide** the same value (if the protocol terminates)
- If quorum **intersection** holds, this is obvious.

(FV)



We need **infinite** state because the state of all the ballots and tags (rounds) has to be maintained.



20

## Abstract Consensus Algorithm for node $v$

$ballots \leftarrow$  array of FV processes (specific to each ballot)  
 $candidate, prepared \leftarrow (0, \phi)$   
 $round \leftarrow 0$

$propose(x)$   
 $candidate \leftarrow \langle 1, x \rangle$   
~~for all  $b' \prec candidate$  do  $ballots[b'] \cdot vote(false)$~~

~~when for every  $b' \prec b$   $ballots[b'] \cdot delivered(false)$  and  $prepared \prec b$~~   
 $prepared \leftarrow b$   
 if  $candidate \leq prepared$   
 $candidate \leftarrow prepared$   
 $ballots[candidate] \cdot vote(true)$

Ensure all incompatible ballots with a lower number are voted out

Ensure that the candidate ballot is accepted by all

*Handwritten notes:*  
 -  $candidate$  is linked to  $b$   
 -  $ballots[b'] \cdot vote(false)$  is linked to  $b, x$   
 -  $ballots[b'] \cdot delivered(false)$  is linked to  $b$   
 -  $prepared \prec b$  is linked to  $b$

So, here are some basic properties of this algorithm. So, the basic properties would be like this, if some node decided on a ballot, it means some other node must have prepared it, there has to be the case, because after all, there has been a flow of information between the nodes. So, you see the algorithm, you cannot decide a value out of thin air.

So, if a ballot has been decided, it must have been prepared. Now, if a node has prepared a ballot, some node must have proposed the value. And, again, this can be the same argument that we have a continuous information flows. And someone must have proposed the value. All the nodes in intact set decide the same values that this we have been discussing for quite some time now.


That it is not possible for the nodes in intact set to decide some other value, and this is a direct result of the federated voting protocol, that because of quorums and quorum intersections and so on. Given, the fact that all quorums intersect at least the correct node and the correct node can only decide once, and it for the same round.

I mean, it actually cannot prepare different values, so that is not possible. If you take a look at this algorithm over here, so basically the key point here is actually quite clear that for any node, the information flow is maintained. And so, I am not discussing the timeout et-cetera in fact, but given the fact that quorum intersection holds.

It is quite obvious that a single correct node that is there, that will not give confusing messages to the both the quorums that it is a part of. So sadly, even though this algorithm works, we need an infinite amount of state, because the state of all the ballots and all the tags or rounds has to be maintained, which is not in the best interest of the algorithm designer. Because, it will lead to an unbounded amount of state possibly infinite.

(Refer Slide Time: 64:25)

### Finite Version of the Protocol





- The **main** idea is that we cannot **maintain** an unbounded amount of state: all **tags** (rounds) and all **ballots**
- We need to do some **garbage** collection (**dynamic** removal basically)
- We need to maintain a **subset** of all messages (and throw out of messages that are beyond the **range**)

New Terminology

instead of **vote** and **deliver**

**prepare**      **commit**      2-phase




21

So, we have to work on a finite version of the protocol, for the idea is that we do not maintain an unbounded amount of state, instead, we have minima and maxima and so on. So, we remove entries, there is some garbage collection. And also, we do not maintain all the messages and all the state, that we maintain a subset of the full state or a subset of the full set of messages.

And anything that we feel is beyond the range in the sense should not be maintained, or can be inferred, that is removed. So again, we will have some new terminologies as I said, terminology keeps changing, but thankfully, it still remains a two-phase algorithm, fair. So, last time we had prepared and decide in this case, we will have prepare and commit, same thing. And so basically, we have used many terms would deliver, and now it will be prepared commit. But again, in a prepare commit is to kind of refer to the finite version of the algorithm, same concept.

(Refer Slide Time: 65:35)

New FV Algorithm (finite)



prepare (b)  
 if ( $\text{max-voted-prep} < b$ ) then  
    $\text{max-voted-prep} \leftarrow b$   
   send VOTE (PREP  $\text{max-voted-prep}$ ) to all the nodes

If not voted for the current ballot (or later ones), then vote

---

if there exists a maximum ballot (b) ( $> \text{max-voted-prep}$ ) and if every node,  $u$ , in the quorum (containing  $v$ ) sends VOTE (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$   
    $\text{max-readied-prep} \leftarrow b$   
   send READY (PREP  $\text{max-readied-prep}$ ) to every node

1 Always consider the maximum ballot

2 If it receives a  $b_u$  where lower and incompatible ballots are alive, it waits

22

So, the new FV algorithm, the new federated voting algorithm which is finite, works like this. So, what happens is that if you look at when you are preparing a ballot, you see what is the largest ballot that you have voted for. If it is less than  $b$ , and if this is clearly higher number ballot, because you want this to be monotonically increasing. See, if I not voted for the current ballot or for later ones once, then you said  $\text{max-voted-prep}$  to  $b$ , which is also what we were doing previously. So, in this case, what you do is that you say that look, I will keep on voting, I do not have an issue.

But the issue, but the thing is that I will only vote for ballots with monotonically increasing numbers, then I will send the vote. So, the message that I will send is prepare prep, and  $\text{max-voted-prep}$  is just what I voted for, which is basically what I proposed the ballot that I proposed to all the nodes. So, this part is the same, it is like sending your basic vote messages.

So, then what will happen is if there exists a maximum ballot  $b$ , so again, this is the next part. So, there is a ballot  $b$  ( $> \text{max-voted-prep}$ ), and if every node  $u$  in the quorum that contains  $v$  has sent

vote (PREP  $b_u$ ), where  $b_u$  has the following property. If there is any  $b' \prec b$ , it is also lower and  $b' \prec b_u$ .

So, which basically means that if this is  $b$ ,  $b_u$  is somewhere over here, where everything is lower and incompatible is also lower and incompatible to this. So, which basically means that everybody has voted either for the value in this ballot or for a ballot with a higher number, so you are fine.

So, at least you are sure that this ballot has been prepared correctly, in the sense that anything which is lower and incompatible for this is also lower and incompatible with something else, but that is something else has been sent. So, in this case, you go for the ready version. So, max-readied-prep, you set it to  $b$ , and you send READY PREP. So again, the PREP message is the same, but this is the ready message max-readied-prep to every other node.

So, what is the idea? So, here instead of maintaining a Boolean state as we were doing earlier, we are only increasing the numbers are max-voted-prep and max-readied-prep. So, this is like the same thing. It is like saying for a given round, I will vote on the only once. In other words, that is tantamount to saying that I will monotonically increase the round number.

Which means, if let us say my current round number is 10, I will not consider any other round number which is between 1 and 10, but only something which is higher. Furthermore, if I receive a value of  $b_u$ , where lower and incompatible ballots are alive, then it can clearly cannot be prepared. So, I will wait, so I will not proceed with this statement over here, but I will wait.

(Refer Slide Time: 69:16)

## Finite Version of the FV Algorithm – II

if there exists a maximum ballot  $b$  ( $> \text{max-readied-prep}$ ) and if every node  $u$  in a  $v$ -blocking set sends READY (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-readied-prep} \leftarrow b$   
send READY (PREP  $\text{max-readied-prep}$ ) to every node

*(propagate)*

Just propagate READY messages

if there exists a maximum ballot  $b$  ( $> \text{max-delivered-prep}$ ) and if every node  $u$ , in the quorum (containing  $v$ ) sends READY (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-delivered-prep} \leftarrow b$   
prepared ( $\text{max-delivered-prep}$ )

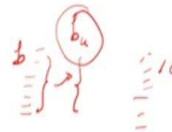
*deliver the PREP message*

Finally deliver the prepare message

NPTEL

23

## New FV Algorithm (*finite*)



prepare  $(b)$

if ( $\text{max-voted-prep} < b$ ) then  
 $\text{max-voted-prep} \leftarrow b$   
send VOTE (PREP  $\text{max-voted-prep}$ ) to all the nodes

If not voted for the current ballot (or later ones), then vote

if there exists a maximum ballot  $b$  ( $> \text{max-voted-prep}$ ) and if every node  $u$ , in the quorum (containing  $v$ ) sends VOTE (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-readied-prep} \leftarrow b$   
send READY (PREP  $\text{max-readied-prep}$ ) to every node

Always consider the maximum ballot

2 If it receives a  $b_u$  where lower and incompatible ballots are alive, it waits

NPTEL

22

So now again, in a two-sub algorithm, so let us look at this. So, what had we seen in the earlier version of the infinite algorithm that once we get a vote from a quorum, which is over here, we send a ready message. So, here also we are doing the same and then we propagate the readied message when we get it from a  $v$ -blocking set.

So, here exactly we are doing the same, is just that the format has changed a little bit. So, idea is the same, you need a ballot  $b$  ( $> \text{max-readied-prep}$ ). And if every node  $u$  in a  $v$ -blocking set has sent PREP  $b_u$ , where again this condition holds, we have seen this before, which basically means anything which is lower and incompatible with your maximum ballot with your stored.

If that is also lower and incompatible than  $b_u$  which you have gotten from another node, then you are fine. So, this is the same as kind of getting ready message from a max from a v-blocking set, for either this ballot or something which is newer. So, you can happily set max-readied-prep to b and just propagate the readied message ready message.

So, this is just propagate function, where we are propagating the message PREP max-readied-prep to every other node, which is propagating the ready messages. So, after propagating, we again look at delivery. So, again delivery we have the same format that there is a max ballot b (>max-delivered-prep).

So, this is basically saying again you deliver once per round. And if, every node u in the quorum sends READY PREP  $b_u$ , where again this condition holds same ballot or a newer. Then, we do the same, we set max-delivered-prep to b and we say that this has been fully prepared. So, we are not using the delivered term here, but we are saying that this has been fully prepared, or we are delivering the prepared message, so we can say deliver the message. And in this case, it was a prepare message, so, deliver the PREP message. So, the key idea is the same, we are not changing anything. In the earlier case, we are a Boolean variable.

The Boolean variable was just saying for this tag if you have not prepared, then prepare if you have not delivered, then deliver. In this case, we are saying that the variable max-readied-prep and max-delivered-prep. And in this case, max-voted-prep will all increase monotonically, which basically means if something has happened for one round, it cannot happen again is the same thing. And so, this is again a two-phase process with exactly the same restrictions and limitations. It is just we maintain finite state.



(Refer Slide Time: 72:24)

## The Commit Function

commit (b):

if  $b \notin \text{ballots\_voted\_cmt}$  and  $\text{max-voted-prep} = b$  then  
ballots\_voted\_cmt  $\leftarrow$  ballots\_voted\_cmt  $\cup$  b  
send VOTE (CMT b) to every node

A prepared ballot is committed (done only once)

when received VOTE (CMT b) from a quorum and  $b \notin \text{ballots\_readied\_cmt}$   
ballots\_readied\_cmt  $\leftarrow$  ballots\_readied\_cmt  $\cup$  b  
send READY (CMT b) to every node

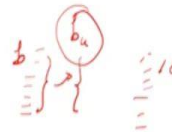
Send READY messages after a quorum votes



VOTE  
READY

24

## New FV Algorithm (finite)



prepare (b)

if  $\text{max-voted-prep} < b$  then  
max-voted-prep  $\leftarrow$  b  
send VOTE (PREP max-voted-prep) to all the nodes

If not voted for the current ballot (or later ones), then vote

if there exists a maximum ballot  $b$  ( $> \text{max-voted-prep}$ ) and if every node,  $u$ , in the quorum (containing  $v$ ) sends VOTE (PREP  $b_u$ ) where  $b' \triangleleft b \Rightarrow b' \triangleleft b_u$

max-readied-prep  $\leftarrow$  b  
send READY (PREP max-readied-prep) to every node



Always consider the maximum ballot

2

If it receives a  $b_u$  where lower and incompatible ballots are alive, it waits

22

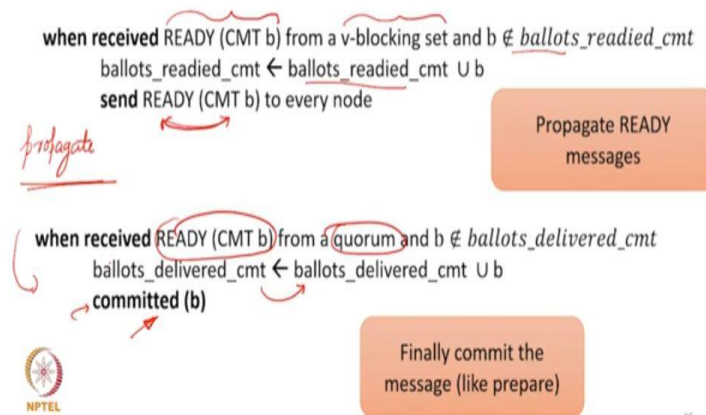
So, as we said we will have two functions prepare and commit. So, prepare is again federated voting on the on the prep message, and commit is basically the same. So, it will have a very similar format, it is slightly simple, simpler. So, here the idea is if the ballot  $b$  is not something that I voted commit for and  $\text{max-voted-prep}$  is equal to  $b$ .

So, where is  $\text{max-voted-prep}$  set, I will show you, it is set over here in the prepare function. So, look, if this is what I have prepared, then I decide to commit it, I decide to go to the next step. When I have received VOTE commit  $b$  from a quorum and I have not readied the commit, so, in this case, I have not voted, and in this case I have not readied it.

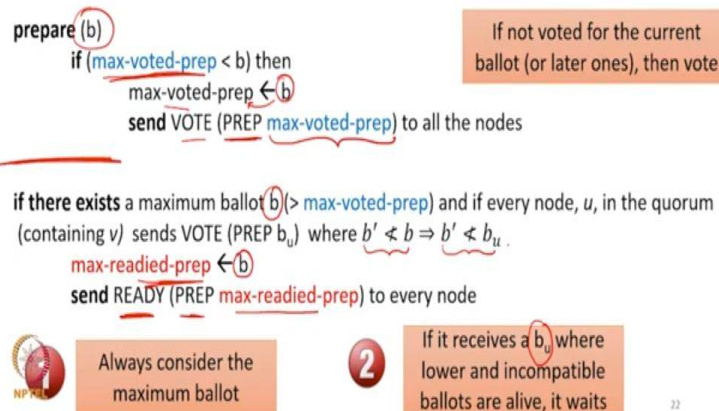
Same idea, instead of Boolean and maintaining a set, because the point is that I could be committing and reading many ballots corresponding to different rounds. So again, in this case, if it is not a part of it, I make it a part of it. So, that is why I have the union operation over here. And once I have gotten the vote, I send a READY message, this was again the same. As you can see, the same pattern of sending VOTE messages and once you get VOTE messages from a quorum, you send a READY message. You send READY messages after a quorum votes.

(Refer Slide Time: 73:58)

## Commit Function – II



## New FV Algorithm (*finite*)

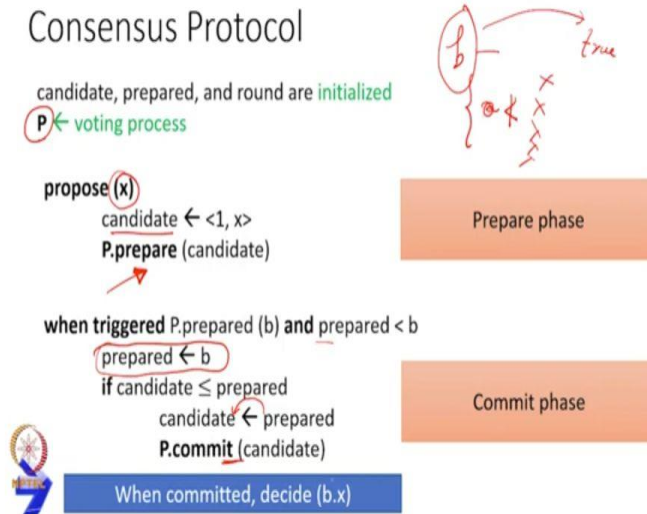


So, the commit function over here is that when you receive READY (CMT  $b$ ) from a v-blocking set, so this is again the propagate. So, we have always had a function to propagate the ready

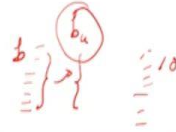
messages. So, when we receive READY commit from a v-blocking set, same idea, and we have already not readied it, then we put it into the ballots readied committed, READY commit set. And we send a READY (CMT b) message to every node, so to every node, we send a message saying that just propagating the READY message. And this part is again, the same we have seen it.

So, in this case, we deliver the commit message when we get the READY message from a quorum, and b is not a part of ballots delivered commit. So, in this case, we add the ballot to ballots delivered commit and we commit the message, and what we do is that we finally deliver it. So, it is the end of the two-phase protocol. So, we commit the message, the same way we did like prepare.

(Refer Slide Time: 75:20)



## New FV Algorithm (finite)



prepare  $(b)$   
 if  $(\text{max-voted-prep} < b)$  then  
      $\text{max-voted-prep} \leftarrow b$   
     send VOTE (PREP  $\text{max-voted-prep}$ ) to all the nodes

If not voted for the current ballot (or later ones), then vote

---

if there exists a maximum ballot  $b (> \text{max-voted-prep})$  and if every node,  $u$ , in the quorum (containing  $v$ ) sends VOTE (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$ .  
      $\text{max-readied-prep} \leftarrow b$   
     send READY (PREP  $\text{max-readied-prep}$ ) to every node

Always consider the maximum ballot

2 If it receives a  $b_u$  where lower and incompatible ballots are alive, it waits

So, now our job is much simpler. So, the basic consensus protocol as you will see, and again this is the finite version, actually becomes far far much much simpler. So, we initialize candidate, prepared and round the same way we were doing. We let P be the voting process. Propose x exactly the same thing where we have a candidate and we prepare the candidate. So, preparing the candidate would basically mean what it would mean going back to the federated voting of the prepared candidate, where basically we do it. But, how do we ensure that everything less than the it is incompatible?

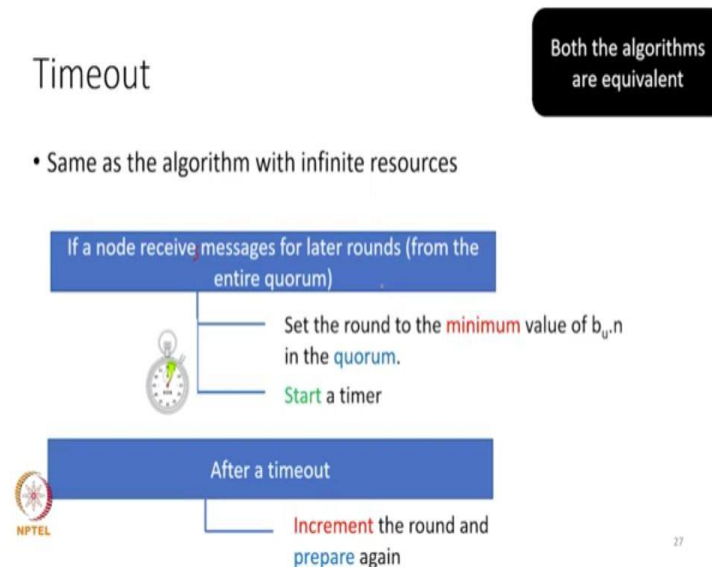
Well, using the check over here that anything that will lower and incompatible is also lower and incompatible with that. So, so that is how we ensure that the moment something is prepared, it is nothing lower and incompatible with it is actually alive. So, once the ballot has been prepared, and if let say prepare is less than the current ballots.

So again, this is a standard check that we have had to ensure that we prepared a ballot only once. So, then what we do is if the candidate is less than equal to prepared, we set candidate to prepared. Again, we have seen this earlier, and we commit the candidate, and when it is committed, we decide.

So, as you can see, this is the same pretty much as the algorithm with infinite resources where you prepare first, preparing means what? Take the ballot anything lower and incompatible. you cancel all of them essentially make everything vote false. And in this case, once that is done, once you have prepared something it becomes our current candidate. And then you commit it, commit it

basically means you make everybody agree on these values, make it vote true. If let us say this message is delivered in our two-phase protocol, then you decide, otherwise, you go for a timeout mechanism.

(Refer Slide Time: 77:36)



And in the timeout mechanism, both the algorithms are equivalent. It is same as the algorithm with infinite resources. See, if a node, receives messages for later rounds from the entire quorum, any set round to the minimum value of  $b_u \cdot n$  in the quorum, and you start a time. After the timeout you ensure that all the messages from this node has been received.

And then you increment the round and then you again prepare for the later round. So, in the worst case, what can happen is because liveness is not guaranteed, the rounds will keep on increasing till infinity, which is fine. Because, the FLP result in any case says that if we have faulty processes, and because of that our algorithm is not converging. That is fine, it may go on forever.

(Refer Slide Time: 78:28)

## Lying about Quorum Slices

- We do not make any **assumptions** about **faulty** nodes
- As long as we have an intact set that guarantees a non-empty quorum intersection comprising **correct** nodes, there is no **problem**
- All these protocols are **obstruction** free
  - This means that if the **faulty** nodes stop
  - **Consensus** is achieved (subject to bounded clock skew)



28

So, now let us come to the issue of lying about quorum slices, so because we are considering Byzantine failures, nodes may lie about their quorum slices. So, let us not make any assumptions about faulty nodes not required. As long as we have an intact set that guarantees a non-empty quorum intersection comprising correct nodes, there is no problem. Even if they lie, you will see a proof in the paper, there is no issue. All of these protocols are obstruction free. This means that if the faulty nodes stop, consensus can still be achieved, subject to bounded clock skew, but consensus can be still achieved.

(Refer Slide Time: 79:12)

## Key Lemmas in the Proof Sketch

1. If two **nodes** in an intact set send **READY** ( $t, a$ ) and **READY** ( $t, a'$ ) messages, then  $(a=a')$ .
2. If a node  $v_1$  **commits** a ballot  $b_1$ , then the **largest** ballot  $b_2$  **prepared** by any other node in the **same** intact set (before the commit) is such that  $b_1 \approx b_2$
3. All correct nodes in an intact set ultimately **decide** the same value (if the algorithm **terminates**)
4. It is a **proposed** value.



29

## New FV Algorithm

**prepare** ( $b$ )

if ( $\text{max-voted-prep} < b$ ) then  
 $\text{max-voted-prep} \leftarrow b$   
 send VOTE (PREP  $\text{max-voted-prep}$ ) to all the nodes

If not voted for the current ballot (or later ones), then vote

if there exists a maximum ballot  $b$  ( $> \text{max-voted-prep}$ ) and if every node,  $u$ , in the quorum (containing  $v$ ) sends VOTE (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-readied-prep} \leftarrow b$   
 send READY (PREP  $\text{max-readied-prep}$ ) to every node

Always consider the maximum ballot

2 If it receives a  $b_u$  where lower and incompatible ballots are alive, it waits

NPTEL

22

## Finite Version of the FV Algorithm – II

if there exists a maximum ballot  $b$  ( $> \text{max-readied-prep}$ ) and if every node,  $u$ , in a  $v$ -blocking set sends READY (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-readied-prep} \leftarrow b$   
 send READY (PREP  $\text{max-readied-prep}$ ) to every node

Just propagate READY messages

if there exists a maximum ballot  $b$  ( $> \text{max-delivered-prep}$ ) and if every node,  $u$ , in the quorum (containing  $v$ ) sends READY (PREP  $b_u$ ) where  $b' \prec b \Rightarrow b' \prec b_u$

$\text{max-delivered-prep} \leftarrow b$   
 prepared ( $\text{max-delivered-prep}$ )

Finally deliver the prepare message

NPTEL

23

Key Lemmas in the proof sketch. So, how do you prove that this is correct? So, I will maybe walk you through some few key Lemmas and the proofs are all there in the papers. So, if two nodes in an intact set send READY ( $t, a$ ) and READY ( $t, a'$ ) messages. And  $a = a'$ , so we have already seen this, and because of quorum intersection, this will happen. Because, the single correct node will not lie into the quorum is that it is a part of it. If  $v_1$  commits a ballot  $b_1$ , say if let say node  $v_1$  commits a ballot  $b_1$ , then the largest ballot  $b_2$  prepared by any other node in the same intact set before the commit.

So, before you commit is such that  $b_1 \approx b_2$ , this can be easily seen in the proof. So, this I can show you, it is being insured by this line over here. So, let me maybe clear off the view 12013 the slide.




So, it is this line over here which is saying that look everybody has to have prepared for this, or basically a compatible ballot which is larger. And given the fact that the other node will have to pass through this stage, and this check is being done as you can see everywhere, so again I will.


So, as you can see, this check is being done everywhere. So, it is not possible to bypass this check at any step. So, that is why it is not possible that something else will get prepared and move through all the stages, once some node in intact set has gone as far as commit. So, the next thing is that all the correct sets in an intact set ultimately decide the same value.

Of course, if the algorithm terminates, so we have already seen different 12106 of this. So, I will not describe this further and needless to say it is a proposed value. So, this again follows all the properties or let us say guarantees all the properties that an ambiguous algorithm is supposed to provide you.

(Refer Slide Time: 81:23)

Conclusion 

- The notion of quorums and quorum slices is the most important. They need to be constructed wisely.
- Practical implementations are very fast.
- Some guarantees regarding the reliability of intersecting nodes and partial synchrony need to be made.



30

So, what is the conclusion? The conclusion is that we were able to kind of give you a very different algorithm. But again, the algorithm is basically based on quorum some quorum slices, where of course, in the intersection you need a correct node which is not going to live. So, as long as this can be ensured, we can ensure that our protocol is much faster, because all the nodes do not have to participate. It is only a limited number of nodes into participate until one quorum decides, until at least one intact set I should be more precise, until at least one intact set decides then we are done.



Some guarantees regarding the reliability of nodes and partial synchrony need to be made. But, those are not impractical guarantees, they are quite practical guarantees. And that is why the stellar protocol is called an internet level protocol, because you can really achieve consensus on a very large and wide scale.

(Refer Slide Time: 82:26)

## References

- [Main reference] García-Pérez, Álvaro, and Maria A. Schett. "Deconstructing stellar consensus (extended version)." *arXiv preprint arXiv:1911.05145* (2019).
- [Original paper] Mazieres, David. "The stellar consensus protocol: A federated model for internet-level consensus." *Stellar Development Foundation* 32 (2015): 1-45.



31

So, these are the two papers. I have primarily reference the commentary of the original paper, because I felt that the readability was much more. But, the original paper is over here. You can also go to the website or stellar and download the code, and take a look at it. So, that would be quite interesting. So, this was a reasonably long lecture, and but also stellar was a quite complex protocol. And it is also very new, and it is very fast and scalable.

So, it is solving many of the existing issue that Bitcoin and Ethereum had, which was of a very low throughput. Of course, at the cost of additional assumptions, which at least to my mind are not very impractical. But, I would like to encourage all of you to go to the, go and take a look at the source code of stellar.