

Advanced Distributed Systems
Professor. Smruti R Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture No. 12
Paxos made simple

(Refer Slide Time: 0:17)

Problem of Consensus
Paxos

Paxos
Distributed Consensus

Smruti R. Sarangi

Department of Computer Science
Indian Institute of Technology
New Delhi, India

NPTEL

Smruti R. Sarangi Assorted Algorithms 1/22

In this lecture, we will discuss the paxos algorithm. Paxos is a distributed consensus algorithm that was proposed more than 20 years ago and henceforth many avatars of paxos have been proposed. So, we will be looking at something called paxos made simple which is a simplified explanation of paxos which was published later roughly to in 2001.

(Refer Slide Time: 0:44)

Problem of Consensus
Paxos

Outline

- 1 Problem of Consensus
- 2 Paxos
 - Motivation
 - Algorithm
 - Analysis

NPTEL

Sruti R. Sarangi Assorted Algorithms 2/22

So, we will first discuss the problem of consensus which is a very very important problem in distributed systems and then we will discuss the paxos algorithm and finally prove it.

(Refer Slide Time: 0:59)

Problem of Consensus
Paxos

Consensus

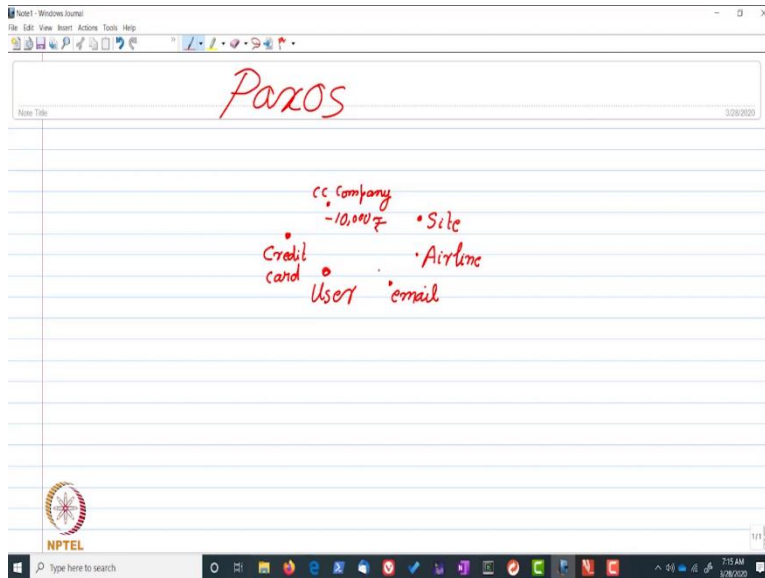
- Each process might propose a value. *3, 4, 5, 6, 8*
- Processes co-ordinate to agree upon one value, V .
- V needs to be proposed by at least one process.
- All processes need to agree upon V .

Uses

- 1 Solve the distributed commit problem
 - 1 Each process can propose two values – **commit** or **abort**
 - 2 All the processes agree to either commit or abort.
- 2 Run multiple copies of a computation, and use voting to decide the result.

NPTEL

Sruti R. Sarangi Assorted Algorithms 3/22



So, the problem of consensus is very simple. So, it is like this that we have a multitude of processes and so we have different processes right over here. Each process may propose a value. So, the processes coordinate among themselves to agree upon one of the proposed values. So, let us say that this process over here proposes 3, this proposes 4, 5, 6, and 8. So, then these five processes coordinate among themselves to agree upon one of the values that has been proposed.

So, kindly note that the value that all the processors agree upon has to be one among the proposed values. So, it is not the case that all of them can agree upon value 0 because value 0 it is true that all of them would agree on the same value but the value of 0 has not been proposed. So, this criteria has to hold that the value of v has to be proposed by at least one process.

So, why is the consensus algorithm so important? Well, so the reason it is that important is basically because in a distributed system the main aim is that we want the entire system to look like a single system to an outsider which means that between the different nodes there has to be a notion of an agreement that they agree upon something, such that that agreed value can be showed to an outsider.

So, let me give one example. So, one example in this case would be that of somebody trying to buy an airline ticket with a credit card machine. So, we have different processors. So, one process of course is the user, one process is the credit card machine, one more process is the credit card company. So, let me call it the cc company then if I am using an online booking site like MakeMyTrip, Expedia that would be one more entity. So, let me call this the site then of course

we will have the airline once the ticket is booked I get an email with my e-ticket. So, then of course my email is one more process.

So, how many processes do we have? We have 1, 2, 3, 4, 5, 6. So, if I book a ticket all six of us have to be in agreement about two things either the ticket is booked. So, if the ticket is booked well then the machine validates my credit card the money is debited from my credit card, it goes to the site and then the airline books the ticket and I get the ticket via email. If let us say the ticket is not booked for some reason, then also there is no problem I am duly informed that my ticket could not be booked.

The problem arises when let us say that I book a ticket for 10,000 Indian rupees. So, then 10,000 rupees get deducted from my account and the ticket is not booked. So, this has happened to all of us it has definitely happened to me a lot that my ticket my money was debited but my ticket was never booked and, so then I had to keep on calling the site of the travel agent the online site.

And they would they would say that all is well whereas all is not well and because I do not have my ticket and the main problem is that there was no consensus among these 6 sites, have there been a consensus among these 6 processes, then this particular problem would not have happened.

So, either all of them would have said that my ticket is booked which means I would have had my ticket with me or all 6 would have said that for a certain reason they were not able to book my ticket, which is also fine by me but I do not want an intermediate situation. So, this is the example that we gave is essentially a commit, abort problem, where I say that commit means that my ticket is booked and abort means that my ticket is not booked. So, all of us have to agree on (())(6:02).

And well, there can be many many other examples of a consensus. So, that can also be where I am doing distributed computation, and so then I need to ensure that all the processes get the same values and also others.

(Refer Slide Time: 6:24)

Problem of Consensus
Paxos

Consensus with Reliable Processes

Simple Algorithm

- Elect a leader.
- Use the leader to collect proposals, decide on a consensus value, and broadcast the all processes.
- Issues:
 - Fault tolerance
 - Centralized processing

NPTEL

Smruti R. Sarangi Assorted Algorithms 4/22

Problem of Consensus
Paxos

Consensus with Reliable Processes

Simple Algorithm

- Elect a leader.
- Use the leader to collect proposals, decide on a consensus value, and broadcast the all processes.
- Issues:
 - Fault tolerance
 - Centralized processing

The consensus problem is interesting in the presence of faults

NPTEL

Smruti R. Sarangi Assorted Algorithms 4/22

So, we can say that what is the big deal with consensus, it is very simple. We can elect a leader we already have many algorithms to elect a leader, we use the leader to collect proposals decide on a consensus value and broadcast it to all processes. This seems to be a very fair idea where we have a set of processes. They elect one leader among them all the processes send their proposals to the leader, the leader chooses one and distributes it.

Well, this sounds to be a good idea, the only problem is that if the leader fails then there is an issue but then of course we can say that we can elect one more leader. Sadly, there is also an issue with that, and so we will see that on the next slide and also the philosophical issue with this algorithm

is that in a certain sense we are centralizing the distributed algorithm, which in a certain sense also violates the spirit of distributed computing for distributing, in distributed computing the main idea is to increase throughput by allowing multiple servers, multiple processes, multiple processes running on multiple servers to do the job.

So, why is this consensus problem that interesting, well because we are going to consider faults and that is because faults always happen in the real world and also see the reason that I was not able to book my ticket in many instances was basically because there was a failure at some point either there was a failure in the network or there was a failure in the credit card authentication process. So, these failures are what give a real world feel to the problem.

(Refer Slide Time: 8:21)

Problem of Consensus
Paxos

Basic Results

FLP

Result by Fischer, Lynch, Patterson

It is impossible to achieve consensus with even one faulty process in an asynchronous system.

Reasons:

- We cannot distinguish between a failed and a slow process.
- Assume $2n + 1$ processes.
 - Let n processes propose 1, and let n processes propose 0.
 - The faulty/slow process holds the key.
 - The algorithm can thus get stuck forever.

NPTEL

Sruti R. Sarangi Assorted Algorithms 5/22

So, now let me come back to what I said on the previous slide, slide number 4 that we can elect a leader, the leader can choose one of the proposals. So, there is a very very famous result in distributed systems called FLP result which we have already discussed in the class. So, this says that it is impossible to achieve consensus where even we have even if we have one faulty process in an asynchronous system it is not possible to achieve consensus.

So, what again is an asynchronous system? It is a system where the processors do not have a shared time base and they are not obliged to give a reply within a certain period of time. So, that is an asynchronous system. So, here if we have a faulty process, so of course we have no way of

distinguishing between a faulty or a failed process in a slow process a consensus is not possible. So, of course we had a long and elaborate proof.

So, if I were to just take the gist out of that proof the gist would be that let me assume that I have $2n + 1$ processes. Let n processes propose 1 and that n processes propose 0. So, one of the $2n + 1$ th process essentially holds the key and this is where the algorithm can get stuck forever because we will never know what it decided and this is essentially why the problem of consensus becomes that interesting even when we consider faulty processes.

So, now the choices in front of us are reasonably stuck, we know that because of the FLP result, we cannot devise a consensus algorithm where even one process is faulty, but well faults are a part of real worlds. So, should we then abandon our efforts? Well the answer is, No.

(Refer Slide Time: 10:37)

Problem of Consensus
Paxos

Way Ahead

- **Safety Property** : Something wrong does not happen.
 - If the traffic light on one road is green, then the traffic light on the perpendicular road is red.
- **Liveness Property** : Something good always happens.
 - The red light will ultimately turn green.
- Let us propose protocols that never violate the safety constraint.
- Liveness is a secondary criteria.

6/22

Instead, we look at what exactly we want. So, we need to prioritize. So, when we know that we cannot get all of it at least we should be happy when if we get some of it. So, what is that? So, we can look at the safety property and the liveness property. So, the safety property (sales) says that something wrong does not happen.

So, consider a traffic light, if one of the roads is green, if the lights on one of the roads is green, then we do not want the light on the other road to also be green otherwise there will be a collision. So, we want the lights on the perpendicular road to be red. So, this is a safety property. The other

is a liveness property which says that something good always happens which means the red light over here will ultimately turn green.

So, whenever we are looking at a protocol we would ideally like to have both but now we know that it is not possible to have both. So, the FLP result clearly says we cannot have, so, we cannot have both let us have protocols that never violate the safety property which in the case of consensus would mean that the key assumptions of consensus which is that choose one among the proposed values and everybody agrees on the same value this should not be violated.

But liveness in this case would be that the algorithm always terminates and always is successful for all processes. So, let us assume that we are not concerned about liveness which means that there will be cases in which the algorithm will not terminate. So, let us just live with safety alone and propose the pack source algorithm that satisfies the safety property.

(Refer Slide Time: 12:45)

Problem of Consensus Paxos Motivation Algorithm Analysis

Types of Agents

- **Proposer** Proposes a value
- **Acceptor** A set of nodes that accept proposed values
- **Learner** Nodes that join the consensus protocol and learn the accepted value.
- Note that a node can be a proposer, and acceptor at the same time.

process/node propose → accept → choosing temporary

NPTEL Smruti R. Sarangi Assorted Algorithms 7/22

So, coming to the paxos algorithm, we have three kinds of nodes. So, so a process and a node we will use synonymously. So, we have a proposer, a proposer is one that proposes a value, then we have an acceptor. So, acceptor is a set of nodes. So, this that accept proposed values, so here accept is a temporary acceptance it is not a permanent acceptance it is just a temporary acceptance which means that it records the value that is being proposed and then, so we have three levels essentially of a proposal we first propose a proposal, then we send it to a set of acceptors where they can temporarily accept it and finally when the consensus is done, we call this choosing.

So, choosing means that a value has been chosen and consensus has been achieved. So, accept is a path in the middle where a value is just temporarily accepted or buffered. Then we have a learner, so learners are nodes that join the consensus protocol late and they want to learn the value that has been chosen, they want to learn the value that has been kind of accepted by everybody. So, note that a node can be a proposer or an acceptor at the same time and so it is not that we have designated nodes that are proposers and designated nodes that are acceptors.

(Refer Slide Time: 14:52)

So, the main motivation of the algorithm actually comes from several of these conditions that we will name C1, C2, C2A, C2b etc. So, let us outline these conditions, the first condition is called,

the first accept condition, condition C1. See if I am an acceptor I do not know how many proposals are there in the system. So, any node in a distributed system always has a very local view. So, it does not know how many proposals are actually floating around in the system.

Hence, the only option or the only choice that an acceptor actually has is to accept the first proposal that it gets. So, it has no other choice. So, this is pretty much the best that the acceptor can do is that accept the first proposal that it gets subsequently it can become more choosy but at least the first proposal it needs to accept. So, now several values can be proposed by different proposers and different acceptors could accept different proposals.

So, then the problem is we will have a situation in which our accepted nodes are sort of sitting there with accepting different proposals p, p' . It is further more possible that because of this but finally something has to be chosen. So, we will find that it will become necessary for an acceptor to actually accept multiple proposals and then ultimately do something such that one of them is ultimately chosen as the consensus value.

And so, we will see that each proposer after several rounds of messages will ultimately need to choose a consensus proposal and we will see how that will happen. But the important point in C1 that I would like to make again is that an acceptor has a very local view an extremely local view. So, the first proposal that it gets it needs to accept and it also needs to accept many more proposals after that primarily because you will have other acceptors that have accepted many other proposals.

So, the model that I am talking about is that you have different proposals that keep proposing and you have a set of acceptors that would be like buffering a subset of the proposals that it gets. And ultimately, from this subset one will be chosen, we will see how.

(Refer Slide Time: 17:46)

Problem of Consensus Paxos Motivation Algorithm Analysis

Proposal Numbering

$\langle pid, ctr \rangle$

- 1 Proposal \rightarrow (number, value)
- 2 All the proposal numbers are unique

Condition – C2(Consensus Condition)

If a proposal (n, v) is chosen, then every proposal with a number greater than n that is chosen, has value v .

- By induction, we can prove that only one value is chosen.

Let us now see, how to satisfy condition C2 to achieve consensus.

NPTEL Snruli R. Sarangi Assorted Algorithms 10/22

Say every proposal is a tuple is a two tuple number and value. So, all the proposal numbers we assume are unique and it is not required but it would be good to assume that they are also monotonically increasing at least from in every node issues monotonically increasing proposals. So, mind you this is not required but this will make our job easy.

How do we ensure that the proposal numbers are unique? Well, this is also easy we can assume that it is a tuple of a pid and a local counter and so we have already seen this in the case of local LAN port blocks that with a combination of the process ID and a local counter we can generate unique numbers. Furthermore, each proposal proposes a value and one among these proposed values needs to be chosen. So, we will assume both of these are integers.

So, the consensus condition is that if a proposal (n, v) is chosen then every proposal with a number greater than n that is chosen has to have value v . So, this is the consensus condition which we will prove that the moment we have chosen a proposal with a certain number then every subsequent proposal that is actually chosen by the system, so we will see that we can choose multiple proposals but all of them have to choose the same value v which means that this value v becomes the consensus value.

So, this is sufficient to satisfy. So, condition C2 is sufficient to achieve consensus. So, which is reasonably clear that the moment we have chosen one value any higher numbered proposal issued by any process does not matter has to again end up choosing the same value.

(Refer Slide Time: 20:04)

Problem of Consensus
Paxos

Motivation
Algorithm
Analysis

Condition - C2a

Condition - C2a
If a proposal with value v is chosen, then every higher numbered proposal accepted by any acceptor has value v .

- Assume that a process wakes up, and gets a proposal. By condition C1, it needs to accept it.
- This is not desirable.
- We need to strengthen C2a.

Srnul R. Sarangi Assorted Algorithms 11/22

So, we can now further kind of specialize condition C2 to condition C2a. So, we will propose several specializations. The first specialization is that if a proposal with value v is chosen then every higher number proposal that is accepted by any accepted has value v . So, what this essentially says is that, if a proposal with value v is chosen after that of course still nodes can keep proposing still nodes can keep accepting, but the acceptors are now bound by value v . So, they are not going to for any higher number proposal except any value that essentially does not have value v .

So, here the explanation is that assume that a process wakes up and gets a proposal by condition C1 it needs to accept it this is clearly not desirable hence to ensure condition C2a we should, we have to strengthen C2a such that it will simply not accept any proposal that does not have value v .

(Refer Slide Time: 21:21)

Problem of Consensus Paxos Motivation Algorithm Analysis

Condition – C2b

Condition – C2b

If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v .

- C2b is a fairly strong condition.
- It ensures that after a proposal is chosen, all higher numbered proposals propose the chosen value.
- Any acceptor, which joins the protocol late, is proposed the consensus value, which it can readily accept.
- The main challenge is to ensure condition – C2b.

NPTEL Smruti R. Sarangi Assorted Algorithms 12/22

Problem of Consensus Paxos Motivation Algorithm Analysis


Condition – C2a

$C2b \Rightarrow C2a$

Condition – C2a

If a proposal with value v is chosen, then every higher numbered proposal accepted by any acceptor has value v .

- Assume that a process wakes up, and gets a proposal. By condition C1, it needs to accept it.
- This is not desirable.
- We need to strengthen C2a.



NPTEL Smruti R. Sarangi Assorted Algorithms 11/22

Problem of Consensus Paxos Motivation Algorithm Analysis

Proposal Numbering

$\langle pid, ctr \rangle$

- Proposal \rightarrow (number, value)
- All the proposal numbers are unique

Condition – C2 (Consensus Condition)

If a proposal (n, v) is chosen, then every proposal with a number greater than n that is chosen, has value v .

- By induction, we can prove that only one value is chosen.

Let us now see, how to satisfy condition C2 to achieve consensus.

$C2a \Rightarrow$ $C2$ $\leftarrow C2a$ $\leftarrow C2b$

Consensus

Smruti R. Sarangi Assorted Algorithms 10/22

Problem of Consensus Paxos Motivation Algorithm Analysis

Outline

- Problem of Consensus
- Paxos
 - Motivation
 - Algorithm
 - Analysis

$C1 \rightarrow$ first proposal

$C2$

\rightarrow choose
 $-$ accept
 $-$ propose

$C2a$ \leftarrow $C2b$

Smruti R. Sarangi Assorted Algorithms 13/22

So, the way that we want to ensure condition C2a is by proposing a condition C2b which will essentially ensure condition C2a. So, C2b this is what it says that if a proposal with value v is chosen. So, assume that a certain proposal with value b is chosen then every higher number proposal that is issued by any proposer has to have value v .

So, so we are further tying up the system we are saying that look once something has been chosen then subsequently if any proposer is issuing any proposal then that also has to have a value v . So, this is a reasonably strong condition but let us look at the implications of this. So, the implication of C2b is that it automatically implies C2a.

Let us look since a proposal that value v is chosen. Every higher number proposal accepted by an acceptor also has to have value v because the only v is being proposed nothing else is being proposed. So, if you look back at C2b just take a look at this line. So, what I am trying to say over here is the moment that a proposal is chosen any other proposal proposed by any other proposer subject to the fact that that proposal is higher number, it has to propose the chosen value.

We will see how to ensure that but assuming C2b is ensured, it is very easy to write this expression that C2b would imply C2a which is that any acceptor has to accept only value v well that is clear because only value v is being proposed and then essentially C2a would further imply C2 that since only value v is being accepted ultimately value v will also be chosen by a higher number proposal.

So, what did we do? We wanted to ensure C2 to ensure C2 we tried to ensure C2a ensuring this will ensure this, to ensure this we are trying to we created a C2b and so if we can ensure C2b it will ensure C2a which will ensure C2 and now, and then we have said that if we can ensure C2 then we can ensure consensus, so that is what our claim is.

So, this, these conditions C1, so what did we do, we had a condition C1 and then we had C2 which of course we had these sub conditions. So, this is reasonably complicated. So, I would advise the listener of this video to go through this several times and to also read this in the paper because appreciating this part is reasonably tricky.

And this is probably why many people have had a lot of problem with the pack source algorithm, but once we go through the algorithm and the proof these assumptions the need for these assumptions and these conditions will become clear. So, where we again, so condition C1 was let us go back and see that an acceptor does not know how many proposals are there in the system. Hence, the acceptor must accept the first proposal that comes by its way.

So, this is that you accept the first proposal you get and furthermore you have to accept a few more to actually complete the algorithm. Condition C2 was that a value v has been chosen by so condition C2 over here was that if a certain proposal n v is chosen and in n v value v is chosen. Then any higher number proposal whose number is more than n that essentially has to choose value v that essentially has to accept value v that essentially has to propose value. If we can ensure these things we can ensure consensus and how to do that is what we will see in the algorithm.

(Refer Slide Time: 26:15)

Problem of Consensus
Paxos

Motivation
Algorithm
Analysis

Paxos Algorithm

```

Algorithm 1: Paxos: Phase 1
1 Propose: Proposer sends prepare( $n$ ) to a majority of acceptors.
2 Receive prepare( $n$ ):
   if  $n > \text{maxPrep}$  then
3      $\text{maxPrep} \leftarrow n$ 
     return  $\text{maxAccept}$ 
4   end
   else
     ignore

```

Handwritten notes:
 - "value C2b" next to "prepare(n)"
 - "A" with arrows pointing to "maxPrep" and "maxAccept"
 - "else" with an arrow pointing to "ignore"

NPTEL
Smruti R. Sarangi Assorted Algorithms 14/22

So, the paxos algorithm is divided into two phases. So, they are known as phase 1 and phase 2. So, the algorithm might seem simple but it is in the category of deceptively simple algorithms where even a very simple algorithm is reasonably hard to prove. So, the first we start by sending a prepare message to a majority of acceptors. So, it is assumed that the size of the network is known.

So, to a majority of acceptors we send a prepare message. The other very important point to note over here is that in the prepare message we only send the number of the message, the message number which is assumed to be unique. So, no two messages will have the same number. So, we are not sending the value this is important and we will see that this will be useful in proving condition C2b which essentially means that the proposer is not at liberty to propose a value.

So, the proposer is basically dependent on other factors to propose a value and this liberty of proposing any value that the proposer wants to propose is at least not there with the proposer in phase 1. In phase 1, if we look at line number 2, the prepare message is sent with just the number of the proposal that is all. So, every acceptor maintains two state variables one is maxPrep and the other is maxAccept.

So, maxPrep is the largest number that the acceptor has received as a part of the prepare message. So, the every prepare message sends the number of the proposal and the largest such proposal number that has been received by an acceptor is called maxPrep. So, this is accepted the prepare message is accepted by an acceptor only if $n > \text{maxPrep}$. So, mind you n cannot be equal to maxPrep because message numbers are assumed to be unique.

So, only if $n > \text{maxPrep}$ is the proposal accepted otherwise we can add an else over here, we just do not do anything we just ignore and nothing goes back to the proposer or we can send a NACK a negative acknowledgement saying that this proposal has been terminated this proposal has failed.

So, assuming it is accepted we set $\text{maxPrep} \leftarrow n$, so this always monotonically increases and we return the value of maxAccept . So, maxAccept is essentially a value which currently the acceptor has accepted. So, the why the max comes and maxAccept will be clear in later slides but let us say that at the moment all that we can say is that maxAccept is the value that the acceptor has accepted.

So, in fact if it is given a choice of values it has taken the maximum that is where max comes from, but for let us assume that for the time being if the prepare message is accepted then the currently accepted value is sent back.

(Refer Slide Time: 30:16)

Algorithm 2: Paxos: Phase 2

- 1 **Receive (maxAccept_i) from a majority of acceptors:**
 $v \leftarrow \max(\text{maxAccept}_i)$ *if ($v = \phi$) $v = \text{proposed-value}()$*
 send $\text{accept}(n, v)$ to all the acceptors in the quorum
- 2 **Received $\text{accept}(n, v)$ request:**
 if $n \geq \text{maxPrep}$ then
 - 3 $\text{maxAccept} \leftarrow v$
 - accept the proposal (n, v)
 - send response to proposer
- 4 end
- 5 **Received all responses to accept messages:**
 Choose the value (v)

Diagram: A proposer (P) is shown at the top, connected by arrows to three acceptors (A1, A2, A3) below. A handwritten note 'majority' is written near the acceptors. A red arrow points from the proposer to the acceptors, and another red arrow points from the acceptors back to the proposer.

So, now let us look at, so once it is sent back we look at phase 2. So, let us say phase 2 starts from here. So, the end of phase 1 what happens is that we have the proposer here and we have multiple acceptors. So, they first got a prepare message, so then they are supposed to if they, if it passes phase 1 they are supposed to send a message back and the i th acceptor will send back the value of maxAccept_i . So, on similar lines the rest of the acceptors. So, you are assuming the rest of the majority acceptors will send back a maxAccept message.

So, what the proposer will do is that after it gets back all of these maxAccept messages it will compute the maximum of these messages. So, it is essentially it will find the maximum value out

of these and assign it to a variable v . So, it is important to consider what happens at the time of initialization. So, I have not mentioned this but this is a side note. So, the variable maxAccept and maxPrep both are initialized to the null value.

You can assume that if all the values and proposal numbers are positive this can be (-1) or if is any other value that indicates that it is null. So, when we are starting phase 1 which means at the time of initialization all the maxAccept values that will be sent to the proposer all of them will be null. So, then we can add an extra line over here which is basically over here just after this that after this if all of them are null then the max of all the nulls will still be a null.

So, if $v = \phi$ then essentially we set $v = \text{proposed value}$. So, what this essentially says is that if $v = \phi$ which means that if all the acceptors send a null value then the proposer is at liberty. This is the only time when the proposer is at liberty to actually propose a value on its own, which would be the value that it would have, it wanted to propose but it is important that it does not have this liberty, it gets the freedom only if only when all the acceptors return a null value and then the proposer proposes a value and sends it to v otherwise, it needs to take the maximum of the values that it has gotten.

So, after that we send an accept message. So, this time the message has the number and the value quite unlike the prepare message that did not have the value. So, we send an accept $n v$ message to all the acceptors in the quorum and that begins phase 2. So, the phase 1 messages where the prepare message and the response maxAccept message and we begin phase 2 with the accept message.

(Refer Slide Time: 34:19)

The slide displays the following text:

```
Algorithm 2: Paxos: Phase 2
1 Receive ( $maxAccept_i$ ) from a majority of acceptors:
 $v \leftarrow max(maxAccept_i)$ 
send  $accept(n, v)$  to all the acceptors in the quorum

2 Received  $accept(n, v)$  request:
if  $n \geq maxPrep$  then
3  $maxAccept \leftarrow v$ 
accept the proposal ( $n, v$ )
send response to proposer
4 end

5 Received all responses to accept messages:
Choose the value ( $v$ )
```

The diagram shows a proposer (P) at the top with a value v and a sequence number n . It sends 'prop' messages to a majority of acceptors (A). One acceptor responds with 'accept'. Handwritten notes include 'insight' and 'maxPrep = n'.

So, we begin phase 2 over here with the accept message that is sent from the proposing node to the accepting node. So, phase 2 also works on a similar manner. So, in this case if $n \geq maxPrep$. So, note the equal to well the equal to comes because of the following reason. So, the first we send a prepare message then so this is phase 1 then we send an accept message assuming it passes phase 1.

So, in the middle if there are no higher numbered prepare messages then in this case we set $maxPrep = n$ and $maxPrep$ maintains its value till this point till the accept point it maintains its value. So, in phase 2 we might very well find that $maxPrep = n$ it might be larger also. So, if it is larger then what does it mean it means that another prepare message came with a higher numbered proposal. So, in this case of course phase 2 fails otherwise phase 2 passes.

So, if $n \geq maxPrep$ which basically means that no higher number proposal has come in the middle then phase 2 passes and we set the value of $maxAccept$ to the value that the proposer is sending which is v . So, essentially the $maxAccept$ field captures the value that has been sent by the proposer the last time phase 2 passed. So, subsequently the proposal $n v$ is accepted, and v becomes the value of the acceptor, and this is what it uses later on, and it sends a response to the proposer that the proposal has been successfully accepted.

So, it is important to at this stage understand and also at this stage realize that this phase 2 will only be successful if no higher number proposal has come in the middle. Because if a higher

number proposal comes in the middle, it will set $\text{maxPrep} = n$ where n is the number of that higher numbered proposal.

And thus, in phase 2, the current proposal will actually fail. If it does not fail it essentially means that during this time period no other proposal with a higher number has come and this is a very very crucial and important insight which we will be using in the proves. So, we set maxAccept to v in this case and which essentially means that this is a candidate for the consensus.

So, it is a two-phase algorithm which essentially means that this is a candidate for the consensus. So, then we send the response to the proposer at the end of phase 2. So, the end of phase 2 what happens is, that all the acceptors that accept the proposal send a response back to the proposer saying that they have accepted the proposal.

If we receive all the responses to the accept messages. So, recall that we select a majority of acceptors and we run the protocol with them if we receive all our responses to all the accept messages that have been sent then we are sure that the value v that the proposer had proposed either on its own or because of this line over here the max of maxAccept line, we can choose the value v and value v is the consensus value.

So, this terminates the algorithm it is a very simple algorithm divided into two phases the first phase and the second phase and if the second phase completes successfully for all the acceptors who are part of the algorithm the phases then we declare victory and we say that value v has been chosen.

So, then what the proposing node can then do is that it can send a message to all the other nodes that look consensus has been achieved and v is the consensus value. Now here is the fun part, the fun part is that even if the proposer just keeps quiet and uses the value v all the later proposers are still going to observe value v to be the consensus value and they are simply not going to observe any other value. So, we can sort of say, this, so this is a kind of a very intriguing result.

(Refer Slide Time: 40:08)

The slide displays the following text:

Algorithm 2: Paxos: Phase 2

- 1 **Receive ($maxAccept_i$) from a majority of acceptors:**
 $v \leftarrow \max(maxAccept_i)$ ✓
send $accept(n, v)$ to all the **acceptors** in the quorum
- 2 **Received $accept(n, v)$ request:**
if $n \geq maxPrep$ then
 - 3 $maxAccept \leftarrow v$
accept the proposal (n, v)
send response to proposer
- 4 **end**
- 5 **Received all responses to accept messages:**
Choose the value (v)

The diagram shows a state transition box labeled 'S' with an input 'P' and an output 'v'. Below it, a 'Learner' is shown with a 'max Prep' value and a response 'v'.

So, this is very non-intuitive, in the sense what, this is in a sense trying to say is that well we if I consider my entire system I ran an algorithm sent a few messages and after that the state of the system has been set to a certain value such that for every subsequent proposer if it proposes a value it is bound to actually choose v . So, the state of the system is not going to change it has been set and then every subsequent proposer will find v to be the consensus value. So, it does not really have to be broadcast.

So, after the protocol if somebody wants to just come and learn the value. Well it can ask a few nodes if they are aware of the consensus value, if they are aware they can send a response. We can always designate distinguished nodes for this purpose or what the learner can do is that it can simply act as a proposal and try to propose a new value.

So, if consensus has already been achieved, then the line over here will essentially not allow it to propose a new value instead it will have to propose something which is already there in the system and if the request goes through which means both the phases pass it will choose a consensus value which again will be v .

So, as I said this is a pretty difficult thing to fathom a pretty difficult thing to visualize but this is actually true that after both the phases regardless of how many proposals come all of them are guaranteed and bound to choose v as their consensus value and the crux of this algorithm was a phase 1.

So, the phase 1 sets a temporary state in each acceptor which is maxPrep. So, maxPrep essentially sets a maxPrep = their proposal number. In phase 2, all that they do is that they come and check if maxPrep has maintained the same value or not. If it has not it means that there is an intruder in the middle and then the abort but for any for any sequence of messages where there is no intruder with a higher proposal number in the middle consensus will be achieved and the value that will be chosen will be the consensus value.

Once consensus is reached, it is reached for all time and all values chosen will be the same which is our claim and we will proceed to prove this in the next few slides.

(Refer Slide Time: 43:04)

The slide is titled "Outline" and is part of a presentation on the "Problem of Consensus" and "Paxos". The navigation menu on the left includes icons for home, back, forward, search, and other controls. The main content area shows the following structure:

- 1 Problem of Consensus
- 2 Paxos
 - Motivation
 - Algorithm
 - Analysis

The footer of the slide indicates the presenter is Smruti R. Sarangi, the course is Assorted Algorithms, and the slide number is 16/22.

The slide is titled "Analysis of Paxos" and is part of a presentation on the "Problem of Consensus" and "Paxos". The navigation menu on the left is the same as the previous slide. The main content area contains two sections, both titled "Definitions":

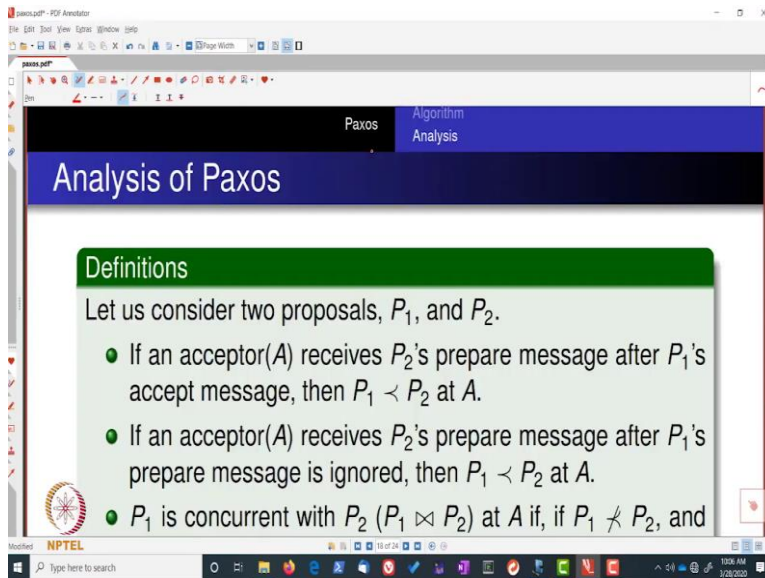
Let us consider two proposals, P_1 , and P_2 .

- If an acceptor(A) receives P_2 's prepare message after P_1 's accept message, then $P_1 < P_2$ at A.
- If an acceptor(A) receives P_2 's prepare message after P_1 's prepare message is ignored, then $P_1 < P_2$ at A.
- P_1 is concurrent with P_2 ($P_1 \bowtie P_2$) at A if, if $P_1 \not< P_2$, and $P_2 \not< P_1$.

The second "Definitions" section contains:

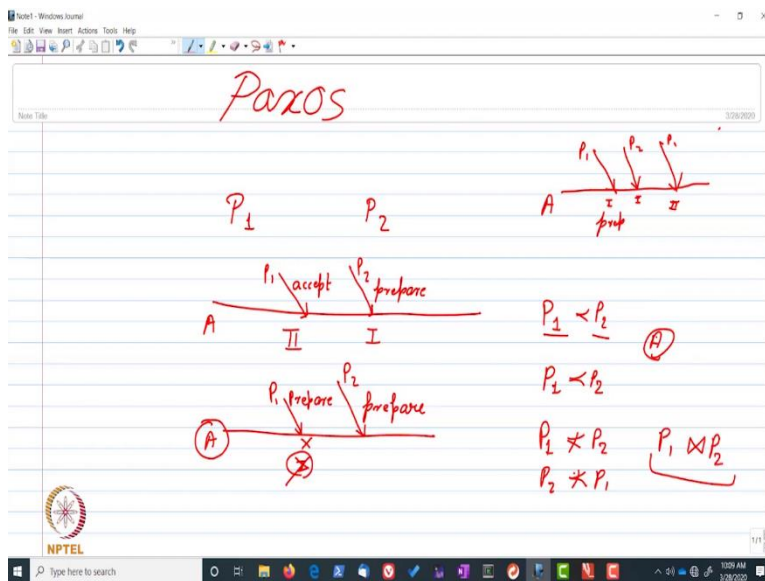
- $P_1 \bowtie P_2$, if $P_1 \bowtie P_2$ at any acceptor.
- Otherwise, either $P_1 < P_2$, or $P_2 < P_1$.

The footer of the slide indicates the presenter is Smruti R. Sarangi, the course is Assorted Algorithms, and the slide number is 17/22.



So, now the proof follows. So, before we explain the proof. Let us go through a few definitions. So, consider two proposals P_1 and P_2 . So, if an acceptor a receives P_2 's prepare message after P_1 's accept message which essentially means, so let us maybe go out and look at this in some detail.

(Refer Slide Time: 43:44)



So, consider proposals P_1 and P_2 . So, let us assume that there is an acceptor A and let this be the timeline. So, assume that P_1 's accept message comes at this point and then comes P_2 's prepare message, then what is clear is that P_1 has finished its phase 2 and then P_2 's phase 1 begins that part is clear. So, we can say that $P_1 < P_2$.

We can have another case at the common acceptor where P1 sends its prepare message then what happens is that the prepare message gets cancelled because that n greater than maxPrep check fails. So, this essentially the phase 1 of this fails and then P2 sends a prepare message, so if P2 sends a prepare message then clearly there is a strict ordering between P1 and P2. So, then also we say that $P1 \prec P2$.

Similarly, we can have other cases where some $P3 \prec P4$. So, this is a standard precedence relationship at a certain acceptor. So, if at all acceptors, so it is, if at all acceptors $P1 \prec P2$ we then say that $P1$ globally $\prec P2$ but we did not go that far. What we want to now show is that it is possible for P1 and P2 to actually be concurrent. See $P1 \nprec P2$ and $P2 \nprec P1$, then essentially we say that P1 and P2 are concurrent and concurrency we are depicting by this horizontal hourglass that P1 is concurrent with P2.

So, what would be a practical example for that well at an acceptor what we would have is that we will have P1's phase 1 and then we will have P2's phase 1 and then P1 will try to initiate its phase 2. So, in this case clearly there is no precedence relationship between P1 and P2. So, let us say this is a phase 1 is a prepare message for them and phase 2 is they accept. So, clearly in this case there is no precedence relationship we cannot say that $P1 \prec P2$ or $P2 \prec P1$. So, P1 and P2 both are concurrent.

So, what we further say is that for two proposals if they are concurrent at any acceptor they are just said to be concurrent. So, now let us go back to our slide presentation. So, we say P1 is concurrent with P2 at A, if P1 and P2 do not have a precedence relationship between each other and at any acceptor if P1 and P2 are concurrent then we say that P1 and P2 are globally concurrent otherwise at all acceptors the precedence relationship holds.

(Refer Slide Time: 47:44)

Problem of Consensus
Paxos
Motivation
Algorithm
Analysis

Analysis - II

n

Theorem
If $P_1 \bowtie P_2$, and $P_1.num < P_2.num$, P_1 will not pass phase II at the common acceptor.

Handwritten notes: P_1 prepare, P_2 prepare, P_2 accept, $maxProp = P_2.num$

Proof.

- There must be some acceptor that gets messages for both proposals.
- Assume it first gets a prepare message from P_1 .
✓ P_1 will not succeed in phase II.
- Assume it first gets a prepare message from P_2 .
✓ P_1 will not succeed in phase I.

Handwritten notes: P_1 , P_1 , $P_2.num$, $maxProp = P_2.num$

Smruti R. Sarangi Assorted Algorithms 18/22

Problem of Consensus
Paxos
Motivation
Algorithm
Analysis

Paxos Algorithm - II

Algorithm 2: Paxos: Phase 2

- Receive ($maxAccept_i$) from a majority of acceptors:**
 $v \leftarrow \max(maxAccept_i)$
 send `accept(n, v)` to all the `acceptors` in the quorum
- Received `accept(n, v)` request:**
 if $n \geq maxPrep$ then ✓
 $maxAccept \leftarrow v$
 accept the proposal (n, v)
 send response to proposer
- end**
- Received all responses to accept messages:**
 Choose the value (v)

Handwritten diagram: A box labeled 'S' (Server) receives a 'P' (Proposer) message and sends a 'P' (Proposer) message to a circle labeled 'Learner'. The Learner has a box with 'I' and 'II' and a 'V' below it. A red 'X' is next to 'I', and a red checkmark is next to 'II'. A red arrow labeled 'maxProp' points to 'II'.

Smruti R. Sarangi Assorted Algorithms 15/22

So, now we will prove a sequence of three theorems which will essentially prove that paxos successfully achieves consensus. So, let us prove the first theorem which says that if P_1 and P_2 are concurrent and P_1 's number. So, recall that we were using the n field in the algorithm but we are using num now, they mean the same thing but this is easier to explain and visualize.

So, if P_1 and P_2 are concurrent and P_1 's number is less than P_2 's number. So, P_1 , so we claim that for two concurrent proposals of this nature the one with the lower number which is P_1 that will not pass phase 2 at the common acceptor. So, it will not complete its algorithm basically it

will not complete both the phases so that, this is easy to prove. So, there must be some acceptor that gets messages for both proposals.

So, let us assume that that acceptor is A. So, we will consider two sub cases. So, the two sub cases that we will consider sorry one second, so the two sub cases that we will consider are like this that assume it first gets a prepare message from P1. So, which means that first from P1 it gets a prepare message. So, since they are concurrent, so it will get a prepare message for P2 after that.

So, since $P2.num > P1.num$ the value of $maxPrep$ at this point will be set equal to $P2.num$. Subsequently, P1 will send its proposal P1's except message will come. So, the first thing that the accept message actually checks is the value of $maxPrep$ and at this point the except the phase 2 for P1 will fail because the value of $maxPrep$ has already been set to $P2.num$ and $P2.num > P1.num$, so, the $maxPrep > P1.num$ and this check which is essentially shown over here will fail.

Because this will fail P1 will not succeed in phase 2. let us consider the other case, the other case is we first get a prepare message from P2. Well so this is even simpler if this is this basically says that at this point $maxPrep$ is set to be $P2.num$. Now, P1 sends $P1.num$ as a part of its prepare message. So, this will clearly fail phase 1 because in phase 1 we check if $P1's\ number > maxPrep$ or not which in this case it is not.

So, we see that in both the cases for both this first case and second case P1 is not able to pass both the phases which means that P1 is not successful process of proposal P1 is not successful. So, what, so whatever does proven we have proven that whenever two requests are concurrent two proposals are concurrent the one with the lower number always fails. We cannot say about P2 it might be concurrent with something else but at least P1 will fail that much we are sure. So, we considered concurrent proposals in the last theorem.

(Refer Slide Time: 51:57)

Problem of Consensus
Paxos
Motivation
Algorithm
Analysis

Analysis - III

$P_1 < P_2$ ✓

Theorem
If $P_1 < P_2$, and $P_1.num > P_2.num$, P_2 will not pass phase I.

Proof.

- There must be some acceptor that gets messages for both proposals. (A)
- $P_2.num < maxPrep$ at that acceptor.
- Hence, P_2 will not pass phase I.

P_1 prepare P_2
 $maxPrep = P_1.num$ □

Smruti R. Sarangi Assorted Algorithms 19/22

Now let us look at two proposals where one precedes the other, so if $P_1 < P_2$ what happens, see if $P_1 < P_2$ and P_1 's $< P_2$'s number, then P_2 has to fail, P_2 will not pass phase 1 P_2 has to fail. So, this is very easy to prove we will consider a common acceptor and there has to be a common acceptor because they send messages to a majority of acceptors. So, there has to be one acceptor in common. So, that acceptor what will happen, so since $P_1 < P_2$, P_1 's prepare message comes first $maxPrep = P_1.num$.

Subsequently, when P_2 's prepare message comes, we do the same check with $maxPrep$ and this will fail because P_2 's number $> P_1$ number. So, the check in phase 1 will essentially fail and P_2 will not pass phase 1 because we check with $maxPrep$ but P_2 's number $> P_1$'s number. So, take a look at this.

So, what have we just proven? We have proven that if one proposal precedes the other. So, let us say $P_1 < P_2$ for P_2 to actually pass and be successful it needs to have a higher number see if I take the sum total of the last two theorems they say something very important. What they essentially say is, what they essentially say over here is that if for two concurrent proposals the one with the lower number is going to fail and if proposal $P_1 < P_2$, then again if P_2 has a lower number it is going to fail.

So, in a certain sense it establishes a certain monotonicity or a strictly increasing order of past proposals. So, for proposals to pass both phases they need to have strictly increasing numbers. If I

were to draw a graph it needs to be strictly increasing and further more if two proposals are concurrent then the one with the lower number will fail for sure. So, essentially if I were to if I am an external observer and I keep seeing the proposals that are passing both the phases I will see an increasing number monotonically increase.

(Refer Slide Time: 55:03)

Problem of Consensus Paxos Motivation Algorithm Analysis

Analysis - IV

Theorem
 If $P_1 < P_2$, and both of them succeed (pass phase II at all acceptors), then they choose the same value.

Proof.

- There must be some acceptor (A) that gets messages for both proposals.
- For P_2 to succeed $P_2.num > P_1.num$.
- Induction hypothesis: All the successful proposals with numbers between $P_1.num$ and $P_2.num$, choose the value chosen by P_1 .
- A must have forwarded its value v at the end of phase 1 to the proposer of P_2 .
- If v is the maximum value of the response, we are done.

Smruti R. Sarangi Assorted Algorithms 20/22

Problem of Consensus Paxos Motivation Algorithm Analysis

Analysis - V

Proof.

- If it is not the case, then assume $v' > v$ is the maximum value.
- Let P_3 propose v' . Now, by the induction hypothesis $P_3.num < P_1.num$ or $P_3.num > P_2.num$.
- ~~If $P_3.num < P_1.num$, then P_1 must have seen P_3 's value (cannot be concurrent). **Not Possible**~~
- ~~If $P_3.num > P_2.num$, then P_2 will not pass phase II.~~

Smruti R. Sarangi Assorted Algorithms 21/22

I will combine the last two results to groove the main result which is theorem 3. So, this says that if $P_1 < P_2$ and both of them succeed, both of them phase 2. Then they have to choose the same value this is the operative part. So, which essentially means we have already established a

monotonicity of the numbers of the chosen proposals of the proposals that pass both the phases, now we want to say that for all of them they choose the same value.

So, again we can do the same between P1 and P2 find a common acceptor. So, for P2 to succeed P1, well $P2.number > P1.num$ again follows from the monotonicity. So, now let us set up a mathematical induction problem. So, the induction hypothesis is that all the successful proposals with numbers between P1.num and P2.num choose the value chosen by P1. So, if P1 chooses v all the proposals intermediate proposals which pass both the phases also choose value v .

Now what do we have to prove for the induction to be correct. What we need to prove is essentially that P2 also chooses value v and this has to follow from the induction hypothesis. So, let us see. So, the common node A, what did it do? So, what did it do is that, so this will come into picture at this point at the end of phase 1 when those maxAccept values are being sent, the common node A must have forwarded its value at the end of phase one to the proposer of P2. So, that it has to do because for P2 to succeed a value has to be forwarded.

So, from the induction hypothesis this value has to be v at the end of phase one because that is the chosen value and, so this has to be the case. Why is this the case well you need to understand that A is the common acceptor between P1 and P2 and since P1 chose v . So, it must have sent v at the end of its phase 1 to A such that A would have recorded it and then A would have sent a response and this successful response would have been recorded by P1.

So, the value v that P1 chose must have been there with A and A must have forwarded this value at the end of phase 1 to the proposer of P2. So, this is an important argument to be made here. So, now for P2 what would it do at the end of phase 1? Well it will take a look at all the values that it gets one of them is v and if v is the maximum. Then there is no problem P2 will choose v and we are done and we have successfully proven.

So, what we need to prove is that it will never be the case that other than v some other value is chosen. So, let us just assume for the sake of proving that some value v' is chosen which is actually not v . So, since the value of v' is chosen and we and if we see this since v is proposed we cannot choose any value that is less than v . Whatever we choose since it is a max operation has to be v' and $v' > v$.

Assuming we are choosing something which is not v . So, in this case, let us assume that let the proposal P_3 propose v' . So, by the induction hypothesis where we are assuming that all the proposals between P_1 and P_2 not including P_2 of course all of them choose v . So, what would be actually happening all of them propose and choose v . So, what would be happening is that there are two cases either P_3 appears before P_1 or P_3 's number appears after P_2 .

So, if P_3 let us say appears before P_1 then P_1 must have seen P_3 's value and P_1 would not have chosen v because $v' > v$. So, it is clearly not possible. So, this case is not possible. So, the other case that we are looking at if I were to draw the same diagram would be P_1 a set of chosen proposals then P_2 and then P_3 .

So, now let us look at two sub cases is P_2 , do P_2 and P_3 have a precedence relationship. Well if you look at this it is not really possible because P_2 is choosing the value proposed by P_3 . So, they cannot have a precedence relationship. So, the relationship that the only relationship that they are allowed to have is essentially that both of them are concurrent. Now if both of them are concurrent and P_3 's number $>$ P_2 's number by theorem 2 by the second theorem.

We observe that, so what do we observe, we observe that P_2 and P_3 are concurrent and P_2 's number $<$ P_3 's number. So, by theorem 2 it is patently clear that P_2 cannot succeed but now we know that P_2 has succeeded since P_2 has succeeded this case also cannot happen. So, given that these two cases cannot happen we have proven by contradiction that we cannot use a value of v' which is actually greater than v which proves that the value that we actually choose for P_2 is v and this proves our induction hypothesis.

So, just outline of the proof again and the outline of the proof is that if P_1 chooses v and if you assume that all subsequent higher numbered proposals that are totally successful also choose v , we need to prove that P_2 also chooses v . So, we set up an induction and then we show that the common acceptor must have proposed v . Say any value that P_2 chose has to be greater than equal to v if it is v no problem, if it is greater than v then whoever was the original proposer of the value greater than v which is v dash either had a number less than P_1 's number or a number greater than P_2 's number both the cases are not possible hence P_2 chose v .

So, this establishes our induction hypothesis that once proposal P_1 is fully successful and the consensus value is v our previous theorems tell us that no proposal with a number less than P_1 .num

will ever pass both the phases that is clear. Any subsequent proposal whose number is greater than P_1 will of course pass both the phases but the only value it is allowed to choose is v .

So, that is the only value it is allowed to choose and furthermore if I were to consider the condition C2b when I am computing the max operation what we have just been able to prove that one of them will at least propose v and there is no value in the system which is higher than v which exceeds v .

So, even the values that will be proposed will all be less than equal to v and one of them will be equal to v . So, ultimately the values that will be also be proposed will also actually be v and also the value that will be accepted will be v . So, this has established and we have been able to prove the safety property which essentially says that consensus is going to hold.

(Refer Slide Time: 64:17)

The slide is titled "Progress" and is part of a presentation on the Paxos algorithm. It features a diagram at the top showing a timeline with proposals P_1 , P_2 , and P_3 . Each proposal has a "prepare" phase and an "accept" phase. P_1 and P_2 are concurrent, with P_2 starting before P_1 finishes its prepare phase. P_3 starts after P_1 finishes its prepare phase. The diagram shows that P_1 and P_2 fail, and P_3 is the only one that succeeds. Below the diagram, the slide contains the following text:

- Note that we have not said anything about progress.
- If $P_1 \bowtie P_2$, one of them will not succeed. \times
- We can possibly have a chain of failures, and theoretically never achieve consensus! \Rightarrow FLP
- Solution** Eliminate concurrent proposals.
 - Use a distinguished proposer (leader), who can propose.
 - If the leader fails, choose a new leader.
 - Paxos provides safety, just in case the old leader wakes up.

The slide also includes a navigation sidebar on the left and a footer with the NPTEL logo and the text "Sriniv R. Sarangi Assorted Algorithms 22/22".

So, we have not said anything about liveness which is the progress property. So, let us assume P_1 and P_2 are concurrent and we will show that there is a scenario where nobody will succeed. So, consider P_1 , P_1 sends its prepare message then P_2 sends its prepare message I assume $P_2.num > P_1.num$. So, then P_1 comes with its accept message the accept message will not pass phase 2 because of intervening P_2 so it gets failed.

Then the proposer of P_1 sends P_3 , P_3 is prepare message has a greater number than P_2 's prepare message. So, then P_2 sends its accept the accept again fails and then, so on and so forth. Then P_2 sends a P_4 and P_3 fails, and P_4 fails, and P_5 fails, so it is possible we might have an infinite

sequence of such kind of failures. So, at the end none of no proposal will succeed in both the phases and because of this infinite chain of failures will theoretically never achieve consensus.

So, this is clearly in line with the FLP result which says that we will always find some sequence of actions where consensus is not achievable but again liveness was not our aim because we knew that both safety and liveness are not simultaneously enforceable. So, one way is to of course eliminate concurrent proposals, we can use a leader who is only allowed to propose similar ideas have been taken up by later consensus proposals to simplify things.

Of course, we still cannot find a way around the FLP result, so no point in trying and so but at least paxos provides safety. Just in case any, anything happens say the paxos still provides a certain degree of safety net and furthermore the idea now is that paxos is complex. So, why is paxos complex? Well because this was the protocol for accepting a single value but let us say that we want to send a sequence of values like a sequence of transactions to different servers, then essentially all the servers have to agree on the sequence of values.

So, we will have to run multiple rounds of paxos. So, this makes things very complicated and that is where other consensus protocols will come in which are simpler and simpler than paxos and provide more elegant solutions but nevertheless paxo still remains one of the most popular and one of the most widely used and widely discussed consensus protocols may be the most popular. And anything else any other later consensus protocol use paxos as a gold standard for comparison.

(Refer Slide Time: 68:02)

Problem of Consensus
Paxos

Motivation
Algorithm
Analysis

The part-time parliament, Leslie Lamport. ACM TOCS, 1998.

Paxos made Simple, Leslie Lamport. ACM Sigact News 32.4 (2001): 18-25.

NPTEL

Smruti R. Sarangi | Assorted Algorithms | 22/22

So, the original paxos paper was published in 1998, which was reasonably long and reasonably hard to understand. So, what we have presented in this lecture is paxos made simple by the same author Leslie lamport in 2, which was published in 2001. So, in subsequent lectures we will discuss other kinds of consensus protocols which are simpler and which are specialized to certain application domains