

**Secure Computation: Part II**  
**Prof. Ashish Choudhury**  
**Department of Computer Science and Engineering**  
**Indian Institute of Science, Bengaluru**

**Lecture - 12**  
**Randomized Protocol for Byzantine Agreement: Part I**

(Refer Slide Time: 00:24)

**Lecture Outline**

- Byzantine agreement with a constant expected number of rounds
- ❖ The framework of Ben-Or and Rabin

*t+1 or more number of rounds*  
*- Not constant*

The slide features a video inset of Prof. Ashish Choudhury in the bottom right corner. The text 't+1 or more number of rounds' and '- Not constant' are handwritten in red ink in the top right corner.

Hello everyone, welcome to this lecture. Till now we have seen Byzantine Agreement protocols, broadcast protocols, which requires at least  $t + 1$  number of rounds ok, which was not constant; that means, it depends upon the value of  $t$ . A very fundamental question is can we have a byzantine agreement protocol or a broadcast protocol, which requires only a constant number of rounds; that means, it does not matter whether  $t$  is equal to 1 or  $t$  is equal to 2 or  $t$  is equal to 1000, the protocol requires the same number of rounds. Why it is critical to have a protocol with a constant number of rounds? Because remember when I say communication rounds, that means, when you implement the protocol the parties have to exchange messages and every new round means the party has to open a fresh connection with the other party and send messages to the other party.


So, we will prefer to have a protocol where we have less number of interaction because every time interacting with every other party might be very time consuming, right. So, the protocols that we have discussed till now do not provide us with constant number of rounds. What we

will now discuss is a framework, which allows us to design a protocol with constant number of rounds, but in expectation.

That means it could be possible that the actual number of rounds required in the protocol is not a constant and in the worst case the protocol may never terminate as well, but the probability of that will be very very less. And it will be guaranteed that in expected constant number of rounds fixed number of rounds the protocol produces an output and this is a very powerful framework. The framework was independently formulated by Ben-Or and Rabin ok.

(Refer Slide Time: 02:41)

### The Round Complexity of BA



- ❑ Round complexity of BA is very important
  - ❖ More rounds  $\Rightarrow$  more interaction
- ❑ All the BA protocols discussed till now require at least  $t + 1$  rounds
  - ❖ EIG
  - ❖ Phase-king
  - ❖ Dolev-Strong

} Deterministic protocols
- ❑ **Deterministic** BA protocols with a constant number of rounds ?
  - ❖ **Impossible** --- works of Fischer, Lynch, Dolev and Strong
- ❑ **Randomized** BA protocols with a constant expected number of rounds ?

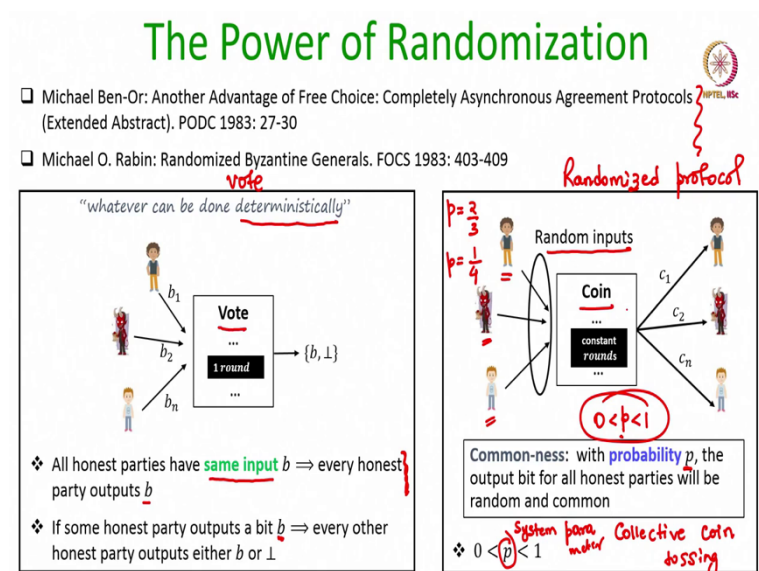
So, as I said the round complexity of BA is very important, more rounds means more interaction and we will prefer to have protocols with less number of rounds and all the BA protocols that we have discussed till now require at least  $t + 1$  rounds, namely the EIG protocol, the phase king protocol, the Dolev -Strong protocol.

Now, all these protocols are deterministic protocols namely the set of actions the set of steps which are executed during the protocol execution they remain the same if you fix your input ok. So, a natural question is can we have a deterministic BA protocol with a constant number of rounds irrespective of what is the value of  $n$ , what is the value of  $t$  and in a very interesting work due to Fischer-Lynch-Dolev and strong, which is a very seminal result, which says that it is impossible to do that.

It is impossible to design a deterministic BA or RB protocol which requires a constant number of rounds irrespective of what is the value of  $t$ , what is the number of corruptions you want to tolerate. That automatically means that if at all we want to get rid of this impossibility we have to embrace randomization, we have to go for randomized BA protocols, randomized broadcast protocols and such protocols will give us the guarantee that they require constant number of rounds in expectation ok.

So, I am now assuming some background regarding probability expectations and so on. Of course, we will do a detailed derivation regarding the expected number of rounds required in the protocol.

(Refer Slide Time: 04:27)



So, as I said earlier this framework for randomized byzantine agreement, randomized broadcast was independently developed by two pioneers of our area, Michael Ben or and Michael Rabin independently in the same year and their idea is basically to combine two primitives.

One of the primitives will be a deterministic primitive, which we call as the vote protocol, vote primitive and this is a deterministic protocol; that means, inside the protocol there will be no random steps which will be executed there will be no random coin tosses based on which you decide what actions have to be performed. So, this will be a one round protocol just one round of communication will be involved, where parties will start with some input and they produce an output which is either going to be a bit or the value null.

And the output property of this protocol will be that if all the honest parties have the same input during this vote protocol, then the output will be the same bit  $b$  for every honest party ok. And the second property will be that if some honest party outputs a bit  $b$  then every other honest party will output either the same bit  $b$  or  $\perp$ , it would not be the bit  $\bar{b}$  ok.

So, the first property says that ok if the parties are already in agreement to begin with; that means, they start the vote protocol with a state where all of all the honest parties have the same input, they are on the same page, then they stick to that output at the end of the vote protocol. But it could be possible that the inputs of all the honest parties were not the same at the beginning of the vote protocol, but it could be still possible that some honest party output some honest party output a bit  $b$  ok. It is not necessary that if the inputs of all the honest parties were different then everyone will output  $\perp$ , no that is not necessary, it could be possible that some honest party ends up outputting  $b$ . So, the vote protocols requirement is that if this happens; that means, if some honest party outputs a bit  $b$ , then every other honest party has to either output  $b$  or the value  $\perp$ , it cannot be the value  $\bar{b}$ .

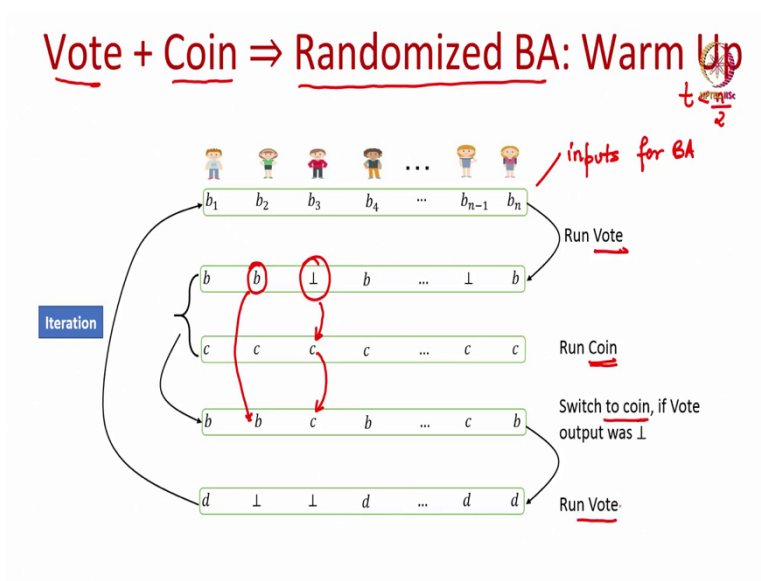
That is the vote primitive, we will see the exact details of the vote protocol in the next lecture or in the subsequent lectures to be more precise. And the second gadget which is used in the framework is a randomized protocol and this is called the coin primitive, it requires constant number of rounds and here the parties do not have any external inputs specific and as part of the protocol they will toss random coins, each party will toss some random coins. And after interacting for constant number of rounds each party will obtain an output a bit ok. Now, what are the security properties? The security property is the commonness property, which basically guarantees that there is some probability  $p$  where  $p$  is strictly greater than 0 and strictly less than 1 such that with probability  $p$  the output bit of all honest parties will be random and common. So, you can imagine this coin primitive as some kind of collective coin tossing.

Because it is a distributed protocol every party will be participating with some random inputs and they expect at the end of the protocol after interacting for constant number of rounds each party has a coin value and with probability  $p$ , it is a random coin and same for everyone ok. So, this is like a distributed version of the regular coin tossing. In the regular coin tossing we know that if we have an unbiased coin and if I toss it, with probability half it could be 0 and with probability half it could be 1.

But now each party has its own coin which is tossing deciding some random values and then they are interacting for constant number of rounds and they expect that at the end of the protocol after interacting for constant number of round the coin values of all the honest parties is the same and it is a random value. And this probability  $p$  will be a system parameter which you can set as part of the protocol ok.

So, notice that this probability  $p$  is strictly less than 1; that means, it will not be the case that definitely all the honest parties will we have the same coin, no that will not be the case ok. In fact, later on we will see instantiations of the coin protocol where probability  $p$  will be  $2/3$  and we will see instantiation of the coin protocol where the probability  $p$  will be  $1/4$  and so on. Notice that during both the vote protocol and the coin protocol at most  $t$  parties could be corrupt.

(Refer Slide Time: 10:16)



Now, let us see that how this vote and coin gadgets can be combined to get a randomized byzantine agreement protocol ok. So, I am explaining the framework in the context of byzantine agreement, but remember that we have always this conversion from byzantine agreement to broadcast.

So, if we have a randomized byzantine agreement protocol using that conversion, which works as long as  $t < n/2$  you can get a randomized broadcast protocol as well; that means, you do not need to run Dolev-Strong protocol, which requires  $t + 1$  rounds, if this randomized BA

protocol is a constant expected round protocol. So, for the moment assume that you have a vote protocol with whatever properties I explained just now and you have a coin protocol.

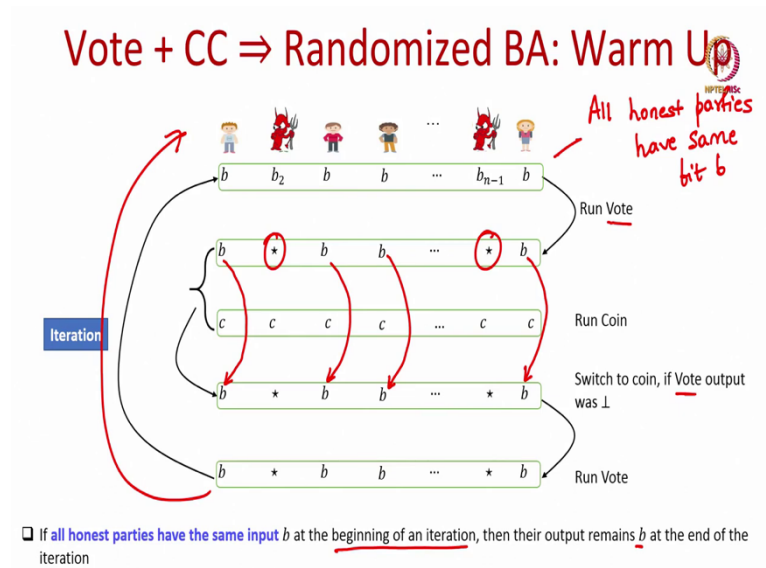
Later on we will see the exact instantiation of the vote protocol, exact instantiation of the coin protocol ok. So, to begin with every party will have their respective bits, their respective inputs for BA ok. The first thing that they will do is they will run an instance of vote which is a deterministic protocol, remember vote is a deterministic protocol, and they try to find out whether all the honest parties have the same bit or not that is the first thing basically they try to do. At the end of the vote protocol some parties may output a bit some parties may output a  $\perp$ , ok.

Of course if all the honest parties have the same input everyone would have output  $b$ , but the honest parties do not know who are the honest parties in the system, what are their inputs they are running the protocol with, ok. Now, at the end of the vote protocol the parties run the coin protocol independent of what output they obtained from the instance of vote they run a coin protocol. And the hope is that a coin protocol will guarantee that with probability  $p$  the output for every individual party is same, say  $c$ , and it is a random bit.

Of course with probability  $1 - p$  that may not happen, but for the moment assume that with probability  $p$  all the honest parties output a common bit at the end of the coin protocol and it is a random bit. Now, what each party does is the following. Party  $P_i$  checks whether its outcome from the vote protocol was bot or a bit. If it was bot then change the bit to  $c$  otherwise stick to that bit if your outcome from the vote was  $b$  then stick to  $b$  only, but if your output from the vote was bought then changed to the output of coin.

And then again run an instance of vote protocol ok. So, we run vote we run coin decide our output and then we decide the input for the next invocation of vote by comparing the output of the vote and the output of the coin ok. And then we do the same process again. So, this whole process of running one instance of vote followed by running one instance of coin and then switching to the coin value if vote output was bot and then again running an instance of vote protocol is considered as an iteration ok.

(Refer Slide Time: 13:56)



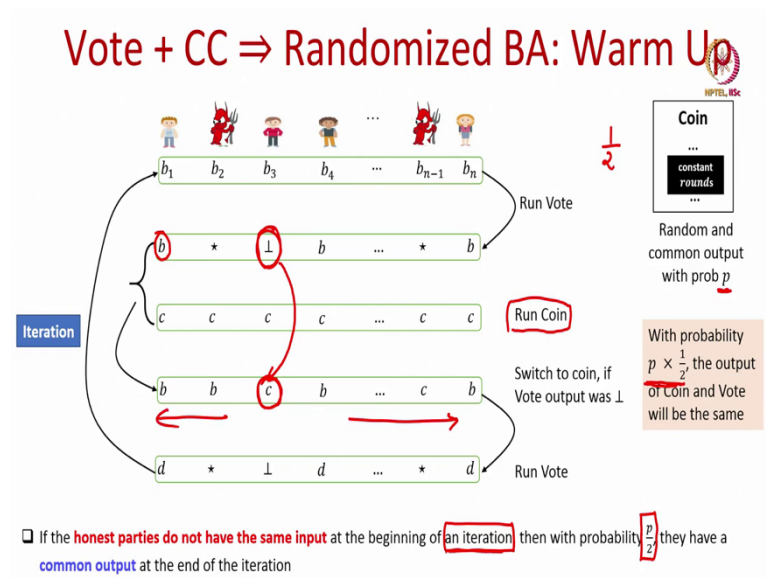
Now, let us try to analyze what is happening ok. Forget about the number of times, the number of iterations we run this. If all the honest parties have the same bit  $b$ , all honest parties have same bit  $b$  at the beginning of the iteration it could be any iteration, it could be the first iteration, it could be the second iteration, it could be the third iteration what I am claiming here is that if all the honest parties have the same input  $b$  at the beginning of any iteration, then at the end of the iteration their output remains  $b$  and it cannot be anything else. Why so?

So, imagine there are some corrupt parties who participate with any bit and all the honest parties have the same bit  $b$ . Now, when they run the vote protocol the outputs of all the honest parties will be  $b$ , I do not care what is the output of the vote for the corrupt parties. Now they run the coin protocol, I do not care what is the output. And now no honest party will switch their output to the output of the coin value, they will simply ignore the output that they have obtained from the coin protocol because the output from the previous instance of vote was not 1.

And now again they are running an instance of vote where all the honest parties have the same input and remember the vote protocol has the property that if all the honest parties have the same input, then everyone obtains the same output and this keeps on happening in every iteration.

So, this shows basically that if we have if we reach an iteration where at the beginning of that iteration all the honest parties have the same input, then agreement has been achieved. Then that agreement is never disturbed irrespective of the number of times we run this framework ok. That is the first observation.

(Refer Slide Time: 16:10)



The second observation is that if the honest parties do not have the same input at the beginning of an iteration, then what is the probability that they have the same output at the end of that iteration? The claim is that the probability with which they have the same output at the end of the iteration is at least  $p/2$ , provided the commonness probability of the coin primitive is  $p$  ok. Why so? So, the first instance of the vote is run over a mixed bag of inputs where the honest parties have different inputs. So, as a result of that the vote protocol might produce a common bit  $b$  for one subset of honest parties and the output  $\perp$  for another subset of honest parties. Now, they run the coin primitive and with probability  $p$  the output is a random bit and common for everyone.

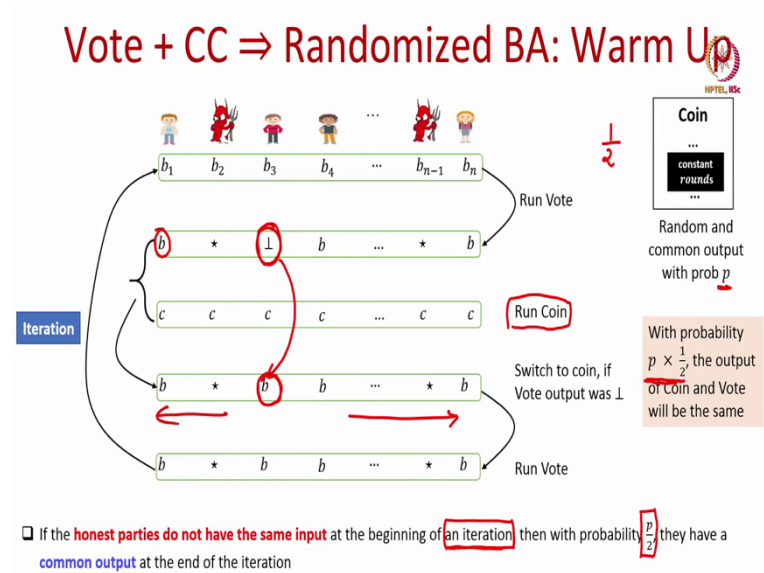
And now everyone every honest party who has obtained the value  $\perp$  during the first invocation of vote protocol will take the output of the coin, will switch to their respective local coins ,whatever they have obtained from the coin primitive. Now, what is the probability that the coin output  $c$  is same as the output  $b$ , which the honest parties would have obtained at the end of the first instance of vote, protocol its  $p/2$ .



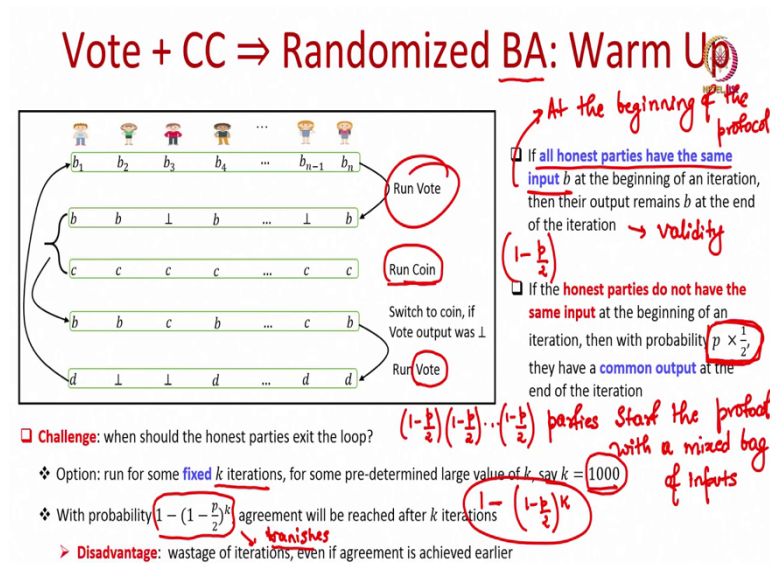
Because with probability  $p$ , the output  $c$  will be common and random for everyone and given that it is common and random for everyone, what is the probability that  $c$  is equal to  $b$ , its  $1/2$ . So, that is why with probability  $p \times 1/2$ , the output  $c$  equal to  $b$ . Now the condition  $c = b$  will hold for all honest parties and due to that all the honest parties who have obtained  $\perp$  at the end of the first instance of vote protocol would have switched to the outputs  $b$  only during the beginning of the second instance of the vote protocol.

That means, with probability  $p/2$  all the honest parties during the second invocation of the vote protocol in an iteration will be having the same input. And as a result of that all the honest parties will obtain the same output at the end of the second instance of the vote protocol with probability  $p/2$  ok.

(Refer Slide Time: 18:54)



(Refer Slide Time: 19:03)



So, these are the two observations which lie at the heart of this framework, ok. So, let us see why these two properties intuitively help us to get a byzantine agreement. If all the honest parties have the same input  $b$  at the beginning of the protocol ok, that is one of the cases.

Then basically vote protocol ensures that that agreement is not disturbed and they retain that input bit as their output irrespective of the number of iterations that are executed, that intuitively ensures the validity property. However, another case could be that parties start the protocol with a mixed bag of inputs; that means, one set of honest parties have the input 0 and another set of honest parties have the input 1. In that case vote does not help them.

So, the idea is then you run a coin after a vote to ensure that second time when you run the vote protocol all of you are all the honest parties are on the same page and the probability that all the honest parties are on the same page before the beginning of the second instance of vote will be  $p/2$ . And then with probability  $p/2$ , the second instance of vote will help them to reach agreement.

Of course there is a probability that even the instance of coin does not guarantee that all the honest parties have the same input at the beginning of the second instance of vote, that can happen with probability  $1 - p/2$ , that is also possible. So, that is why we have to run this process for several iterations. Now, the big challenge is when should the honest parties exit the loop? That means, how many iterations they have to run.

Because they do not know at which iteration the second instance of vote have ensured that all the parties have the same output, they do not know that. So, there are two options to deal with this challenge; the first option is that we run this protocol and when I say protocol; that means, the number of iterations, each iteration consisting of vote, coin, vote for some fixed number of iterations say  $k$  iterations, for some predetermined large value of  $k$ , say 1000.

Then it will be guaranteed that with this much probability agreement has been achieved after  $k$  iterations. How come we get this probability? So, as I said if all the honest parties reach an iteration where they have the same input at the beginning of that iteration, then from that iteration onwards agreement will be maintained.

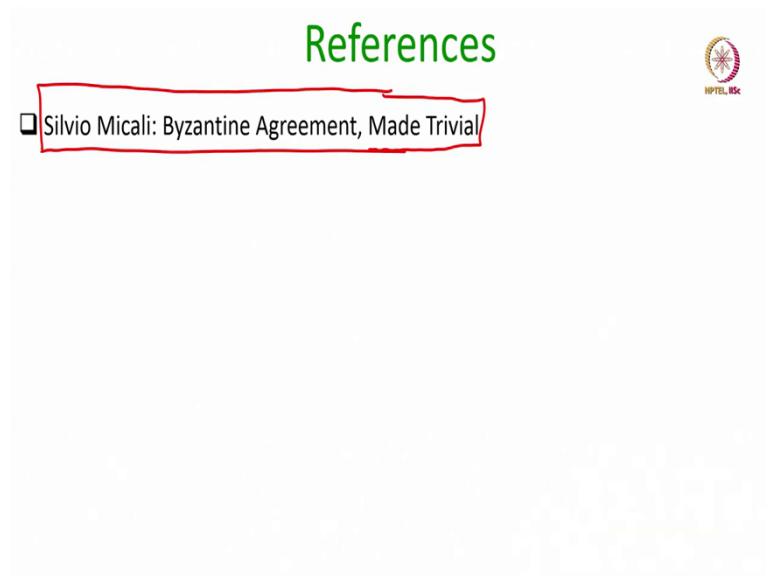
So, everything boils down to the analysis that what is the probability that none of the  $k$  iterations help the parties to ensure that they have an iteration where all the honest parties have the same input. Now, the probability that all the honest parties have the same output at the end of a single iteration is  $p$  over 2. So, the probability that they do not have the same output at the end of an iteration is one minus  $p$  over 2; that means, the probability that the first iteration does not help them to reach agreement is  $1$  minus  $p$  over 2 and the probability that the first two iteration does not help them to reach agreement is  $1$  minus  $p$  over 2 into  $1$  minus  $p$  over 2. And like that the probability that none of the  $k$  iterations help them to reach agreement is  $1$  minus  $p$  over 2 times  $1$  minus  $p$  over 2 whole raise to power  $k$ . So, the probability that at least one of the iterations among this  $k$  iterations guarantee that all the honest parties have the same input at the beginning of the iteration will be this quantity.

Now, you can see if I substitute  $k$  to be a sufficiently large value, then this quantity vanishes, it becomes very very small and that is why the guarantees of this byzantine agreement protocol is probabilistic, it gives you probabilistic guarantees that agreement has been achieved. So, even though this quantity  $1$  minus  $p$  over 2 whole raise to the power  $k$  approaches 0, as you keep on increasing the value of  $k$ .

But the problem is we do not know what is the right value of  $k$  because we cannot simply set  $k$  to be an astronomically large quantity ok because the number of iterations has to be reasonably small. The other disadvantage of this approach of setting the value of  $k$  to a sufficiently large quantity irrespective of whether the agreement has been achieved or not earlier is that it leads to a wastage of enormous number of iterations because it could be possible that agreement has been achieved even immediately after the two iterations.

And then unnecessarily you are keep on running the protocol for 10000 or 20000 or 1000 number of iterations. So, that is a challenge for this framework, but intuitively we now know that how we can run these two gadgets vote followed by the coin gadget for many number of iterations to get a BA protocol with probabilistic guarantees.

(Refer Slide Time: 25:00)



So, there are many references where you can find the description of the framework that we have discussed in today's lecture. For my presentation I have taken the description from this very nice paper which is a very beautifully written paper it is very simple you can very easily understand it. So, I strongly recommend to read this paper.

Thank you.