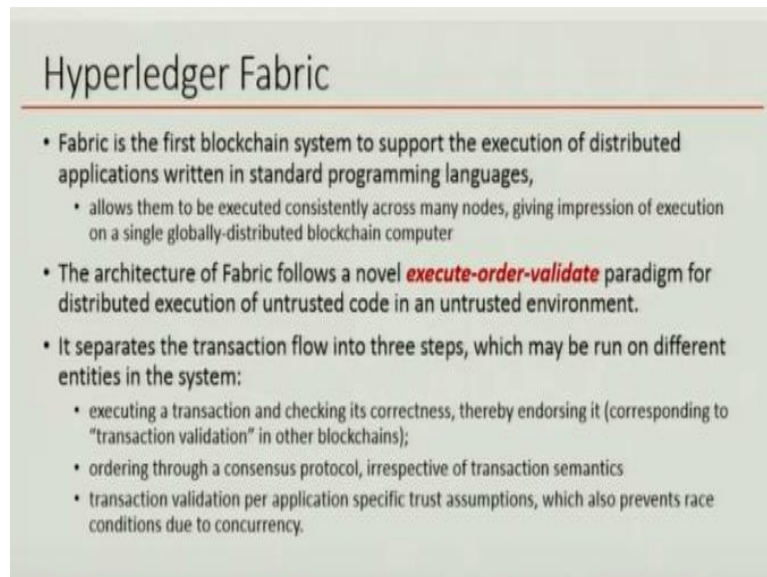


Introduction to Blockchain Technology & Applications
Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 22

Welcome to another session of Blockchain Technology and Application on NPTEL. So last time in the last session, I was talking about Hyperledger Fabric.

(Refer Slide Time: 00:25)



Hyperledger Fabric

- Fabric is the first blockchain system to support the execution of distributed applications written in standard programming languages,
 - allows them to be executed consistently across many nodes, giving impression of execution on a single globally-distributed blockchain computer
- The architecture of Fabric follows a novel **execute-order-validate** paradigm for distributed execution of untrusted code in an untrusted environment.
- It separates the transaction flow into three steps, which may be run on different entities in the system:
 - executing a transaction and checking its correctness, thereby endorsing it (corresponding to "transaction validation" in other blockchains);
 - ordering through a consensus protocol, irrespective of transaction semantics
 - transaction validation per application specific trust assumptions, which also prevents race conditions due to concurrency.

And I was talking about this execute-order-validate paradigm as compared to order-execute paradigm in the previous blockchains that you have seen. So going back a little bit. So Fabric actually distinguishes itself from the previous blockchains that you have studied, like bitcoin blockchain or Ethereum blockchain is that it follows the execute-order-validate paradigm and the transaction flow has three steps.

First executing the transaction and checking its correctness. And this execution does not change anything in the blockchain or in the state of the blockchain. It just in a sense simulate the execution of the transaction. And then the transaction results are endorsed by some of the nodes. They are called the endorsers, and then endorsers send the result of the, of executing the transaction as read/write sets.

And these read/write sets are then sent back to the client which originally requested the or proposed the transaction. And the endorsement is also given by multiple different nodes based on an endorsement policy. We will see what endorsement

policy means. And each endorser also attaches the digital signature of that node so that others can check whether the right endorsers have endorsed the transaction.

Then the transaction when the enough endorsements are collected by the original client, then it sends it or broadcast it to the nodes that are responsible for consensus. And here the consensus basically what it does is that it orders the transactions. And the ordering of the transaction is based on this read/write sets and ordering does not care what these transactions are about what kind of domain it is being done on.

Like for example, it could be financial transactions, it could be land record transactions, it could be other kinds of any other kind of transaction about supply chain, it does not care. So transaction semantics is not looked into by the ordering nodes. The ordering nodes basically run a consensus algorithm and then they do decide on an order.

And then once they have ordered the transactions, then they will bunch them together from various clients multiple transactions will come. If these transactions are working on the same data then the ordering is more important. If the transactions are on completely different sets of data, then their ordering is not that important, but in any case the orderer will audit the transactions. And then this ordered transactions are put into a block.

And we call it cutting a block and then the block hash is computed. A sequence number is given to the block and then the block is broadcast by the consensus nodes to the rest of the nodes. The rest of the nodes, not necessarily all the nodes depending on what the administrator of the blockchain decided who should be the nodes that are called validator nodes.

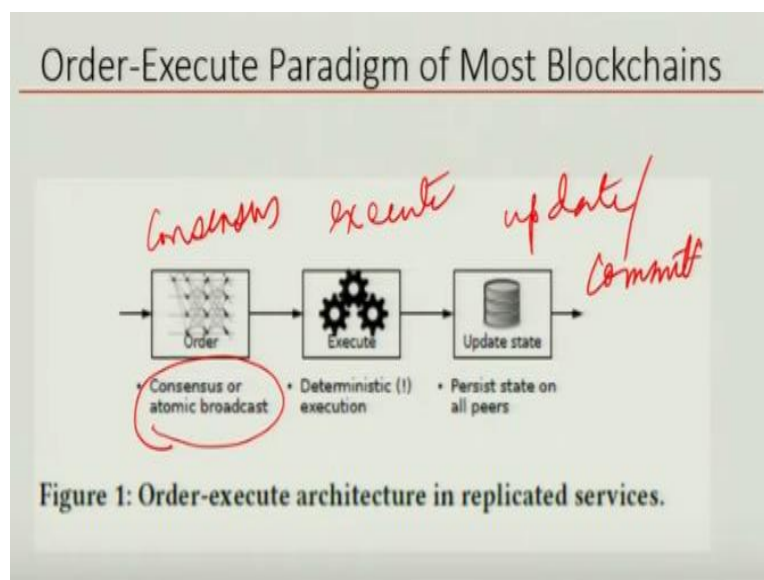
The validator nodes will look into every transaction and then look at the version information in the read/write set and decide which of the transactions are valid and which transactions are not valid. For example, if the version number of a particular data being read in a transaction does not match with the version number of the same piece of data in the state database, then that is not a valid transaction.

Because that means it is reading a different version value of the value of that particular piece of data. And therefore it will if you execute that transaction, it will be bad. So those invalid transactions are then marked. And this marking is done by creating a bid vector, such that valid transactions are set to be 1 and the invalid transactions are set to be 0. Since the transactions are ordered we can do a bid vector to mark it.

And then everybody, every node that is maintaining the blockchain will add that node to the end of the current blockchain and therefore, and then update the state for the valid transactions. The valid transactions will write may write on certain nodes, a certain date piece of data. So that will be written in the state database. So that is how the execution ordering transaction execution ordering validation happens.

And that is the end of a particular block activity. In the meantime, the clients might have submitted more transactions for endorsement. Endorsement might have come and then they were sent to the orderer. So these things are happening sort of in a pipeline. So that is what is the execute order validate paradigm means.

(Refer Slide Time: 05:59)



So going back pictorially if you want to see what this order execute paradigm is, which is first a consensus happens. So everybody gets broadcasted all the transaction. I am talking about bitcoin, Ethereum, etc. So everybody gets a copy of any transaction that happens, then they will bunch some of these transactions together and they will try to cut a block, but in order to cut a block, they have to solve a puzzle.

And once they solve the puzzle, then they have won that round of consensus. Then once the consensus is reached, the winning block is broadcast to everybody. And then everybody has to then execute the transactions at their own nodes and make sure that these transactions are valid, they have the right signatures, etc. And then they will update the state and add that block to the end of the blockchain.

So this is the consensus is ordering. This is the execute and then update or permit. You permit the those set of transactions. So that is what we saw before.

(Refer Slide Time: 07:23)

The slide is titled "Order-Execute in PoW blockchain". It contains a list of bullet points describing the process:

- PoW-based permission-less blockchain such as Ethereum combines consensus and execution of transactions as follows:
 - (1) every peer (i.e., a node that participates in consensus) assembles a block containing valid transactions
 - (to establish validity, this peer already pre-executes those transactions)
 - (2) the peer tries to solve a PoW puzzle ✓
 - (3) if the peer is lucky and solves the puzzle, it disseminates the block to the network via a gossip protocol
 - (4) every peer receiving the block validates the solution to the puzzle and all transactions in the block
- Effectively, every peer repeats the execution of the lucky peer from its first step.
- All peers execute the transactions sequentially (within one block and across blocks)

So now we are we explained to you that we do it differently in Hyperledger. Now let us see, let us discuss a little bit on order execute in proof of work block chains like bitcoin or Ethereum. So we combine the consensus and execution of the transaction as follows. So every peer assembles a block containing valid transaction. To establish the validity, you already pre execute the transactions.

You execute the transaction and you only execute the transactions if the signature matches and the output the you know coin is going to the going to a valid public key etc. The peer then tries to solve the proof of work puzzle. And if the peer is lucky, and win the puzzle, it disseminates a block to the network via gossip protocol. And every peer receiving the block validates the solution to the puzzle and all the transactions in the block.

And doing so they have to also execute the transaction. Every peer repeats the execution of the lucky peer from its first step. And that means that we are doing this execution again and again and again in all the nodes in the system to although it is happening concurrently, but within each node, the execution of transactions are happening in a sequential manner.

But if you think back about what I said about Hyperledger, I have a transaction I am getting endorsed. You have a transaction you are getting endorsed, maybe by the same set of endorsers, maybe a different set of endorsers. So I can have the execution of multiple transactions in parallel. Because if my endorsers do not overlap, then those transactions can altogether be endorsed and sent back to the respective clients.

And all the clients can then submit those transactions to the ordering service. So therefore, and then afterwards, we do not execute the transactions, we only use the read/write set to update the state if the transaction is valid. So therefore execution happens before ordering and before validation. And these executions can also happen in parallel in concurrently on behalf of multiple different clients provided the endorsers are not all common.

So that is the difference that throughput for blockchain that are execute order-execute based their execution of all the transactions even though those transactions came from different clients into the system, they are all executed once a block has won, that block's transactions are all executed sequentially. Whereas here in Hyperledger, I can have parallel execution of transactions.

And then I do ordering and then I do the validation. And then after validation, everybody updates their state. So that is one advantage that Hyperledger has in increasing the throughput of the transactions.

(Refer Slide Time: 10:48)

Limitations of Order-Execute Paradigm

- **Sequential execution** Executing the transactions sequentially on all peers limits the effective throughput
 - In contrast to traditional SMR, the blockchain forms a universal computing engine and its payload applications might be deployed by an adversary. A denial-of-service (DoS) attack:
 - could simply introduce smart contracts that take a very long time to execute.
 - a smart contract that executes an infinite loop
 - To cope with this problem, public programmable blockchains with a cryptocurrency account for the execution cost
 - Ethereum Gas
 - However, blockchain with no native currency does not have this facility

So as I already hinted, that this can be viewed as a limitation of the older order execute paradigm, the transaction's sequential execution by every peer after the block the POW problem has been solved is bad for the effective throughput. Now POW itself takes time as we know and then on top of that everybody has to execute this transaction sequentially.

So therefore, the other problem that can happen is among these transactions remember these transactions are being set as in case of bitcoin through bitcoin scripts. And remember the transaction execution would mean that I put the scriptPubKey and ScriptSig of the previous execution, previous transaction associated with this transaction's inputs together and execute that.

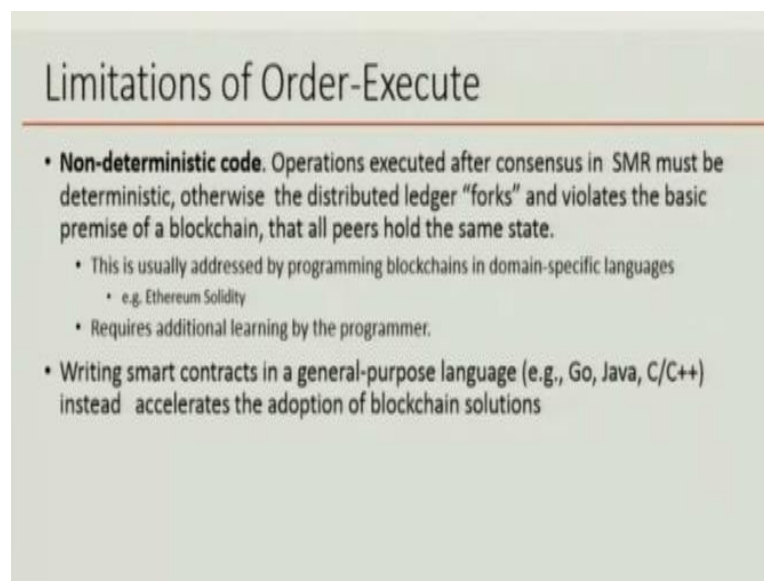
So each transaction execution is basically executing a small program. In case of Ethereum each transaction would be executing a function or multiple functions in the smart contract and therefore, if this smart contract is written in bitcoin, this is not easy to do or not even possible to do because you cannot have loops in a bitcoin script.

But in case of Ethereum, what you will you could do is that if you are a bad player, you could put a smart contract and convince everybody to execute a function in that smart contract in and the function itself may have an infinite loop or whatever. And therefore, you can launch a denial of service attack. Of course, Ethereum solves it by limiting the gas etc.

But if you do not have a native cryptocurrency, in your blockchain, then how do you price the gas, right. So therefore, it is a free for all. You can actually make the system execute very long functions and therefore slowing down the throughput very much. So Hyperledger does not have a native cryptocurrency. And Hyperledger is even allowing writing smart contracts that are not even in a stylized language like Solidity.

It is it could be C, C++ whatever, where you know very long loops can be possible and there is no notion of gas or gas limits. So therefore, executing a transaction sequentially by the by all nodes would not be a good idea, because very easily one can do a DOS or denial of service attack. So that is the idea, one of the limitations of order-execute paradigm.

(Refer Slide Time: 13:46)



Another issue, we are already hinted about that in the last session is that you cannot have non-deterministic programs as smart contracts in the Ethereum or non-determinism in the bitcoin because if the transaction execution, non-determinism means that the same code gives you different result at different times, right. For example, the function if you call random function twice in two different nodes, it will give you different results.

So therefore, if you the blockchain not to get forked because the value the result of a transaction is different in one block and they are different in another block then you must make sure that your smart contracts or scripts are deterministic. So to do that, to

ensure that what you do is you create special languages in which there is no way to create non-determinism right.

So now you might say that if I limit my language will it still be Turing-complete and turns out yes, it will be still be Turing-complete and but you do a domain specific language and then force people who will write smart contracts to program in that language. Now when you want to bring a new now there are a lot of solidity programmers because Ethereum got popular.

But if you start a new blockchain like Hyperledger and say that you have to program it in a very specific language, then there may not be any takers and therefore, the blockchain may not become popular and it will die. So therefore, you have to allow anybody who can program in any language to be able to write contracts. And therefore, and then if you do that, you cannot stop non-determinism because those languages will have non-determinism as a built-in property.

So therefore, what Hyperledger decided is that they will not worry about non-determinism because their final result does not depend on determinism and non-determinism because if the transaction is non-deterministic, then during the execution itself right, by the endorsers, when the endorsements come back, you will see different the client will see different results coming from different endorsers the read/write sets and this transaction will be discarded.

So therefore, we do not have to limit the language and still have determinism in the blockchain state.

(Refer Slide Time: 16:37)

Limitations of Order-execute

- **Confidentiality of execution:** Usually they run all smart contracts on all peers.
- Many intended use cases for permissioned blockchains require confidentiality,
 - i.e., that access to smart contract logic, transaction data, or ledger state can be restricted.
- cryptographic techniques, ranging from data encryption to advanced zero-knowledge proofs and verifiable computation can help to achieve confidentiality but
 - comes with a considerable overhead
- it suffices to propagate the same state to all peers instead of running the same code everywhere.
 - Execution of a smart contract can be restricted to a subset of the peers trusted for this task, that vouch for the results of the execution.

Another issue is confidentiality of your IP. So suppose you are using Ethereum and you are creating distributed application based on multiple smart contracts which are meant for supply chain management. Now Ethereum has, you know other nodes which are not interested in your application but they will all get all your smart contracts because all smart contracts are part of the blockchain.

And so they can see how you have what your business logic is. If they you know take the byte code, do reverse engineering, they can see all that. They will also see the results of all your transactions. In fact, they will execute all the smart contracts. What that gives you is that, so in such a system in a public blockchain, you cannot keep confidentiality of your IP. The IP means intellectual property the way you run your application.

So that basically means that if you want confidentiality, then you have to have in such that your smart contracts are not part of every node or it is not part of the blockchain. So who has the smart contracts in them, the endorsing nodes. They execute the transaction, so they have to execute the smart contracts. So if you trust the endorsers, because they are part of your business environment or business entity, and those who run those nodes are also having real world identity associated with their identity.

So therefore, you can have the endorsing node know your chain code, which is the smart contract in Hyperledger, but no other nodes the orderers do not need to know

your smart contracts. The other nodes that are keeping the blockchain do not need to know your smart contract.

And Hyperledger was specifically thought about as a way for multiple businesses to work together on workflows and data sharing that is meant for inter business activities. Supply chain management is an inter business activity. Banking between multiple banks is an inter business activity. So you may trust all the nodes that are associated with your bank, but you may not trust the nodes that are associated with another bank.

So these kind of environments will be supported, if you have this, cannot be supported on say Ethereum but you can support this on Hyperledger. Now you might say that well in I can do lot of, you know very complex use of cryptography, zero knowledge proof etc., to ensure that kind of thing the confidentiality and probably yes. But it will have a lot of overhead of running cryptography, running zero knowledge proof which basically has heavyweight cryptography.

So therefore, that is not a very good solution. So what Hyperledger decided that all you need to know for all you all the every entity needs to know who is the purveyor of your business blockchain is the state, what state be after every transaction, the system goes into. And what transactions were done, but the transaction ID, etc., who did the transaction, all that stuff.

You do not need to really have the transaction details or the code for the smart contract and every node. So that is the basic idea.

(Refer Slide Time: 20:29)

Limitations of Order-Execute

- **Fixed trust model:** Most permissioned blockchains rely on BFT replication protocols to establish consensus
 - Such protocols typically rely on a security assumption that among $n > 3f$ peers, up to f are tolerated to misbehave and exhibit Byzantine faults
- The same peers often execute the applications as well, under the same security assumption
 - even though one could actually restrict BFT execution to fewer peers
- such a quantitative trust assumption, irrespective of peers' roles in the system, may not match the trust required for smart contract execution
 - trust at the application level should not be fixed to trust at the protocol level.
- A general purpose blockchain should decouple these two assumptions and permit flexible trust models for applications.

The other thing that again, we discussed, you know superficially last time, the trust model. So blockchain in bitcoin for example, you do not trust any of your peers. All the nodes are untrusted and therefore, you have the consensus algorithm, which is very complex, very expensive, with proof of work etc. In Ethereum again the same thing. Now in a private version of an Ethereum, which is not public, you can simplify the consensus process if you trust most of your nodes.

And then you can use something like a proof of authority proof of you know other kinds of consensus techniques. But in a public blockchain, your trust is with no, you do not trust anybody. So that is the trust about the consensus, like how the consensus making nodes are trusted among each other. That is one type of trust. The other type of trust is the trust of the business application, right?

So do you trust everybody to faithfully execute your transaction, everybody to not breach the privacy of your code, not breach the privacy of your activities, etc. So what Hyperledger decided is that to split this notion of trust of the business application and notion of trust in the infrastructure. The infrastructure, which is responsible for consensus, there the nodes may all trust each other.

Or the nodes may have some assumptions about trust, like we only we trust everybody, but some of the nodes may be compromised. So we only trust maybe one-third of the nodes, two-thirds of the nodes, etc. And accordingly, you can design your

consensus algorithm. But the trust mechanism at the business level would be enforced by a notion of what they call channels.

So what the channels do is that they separate the on the same infrastructure you can run multiple blockchains. So one entity might be running a supply chain application, and one the same entity can run the supply chain application and some entity can run the banking application or monetary transaction applications between the same set of businesses.

And they do not need to see each other let us say or they do not trust each other because these are two different entities. One may be audited another one may be for internal purposes etc. So you have this notion of channels. So the channels basically maintain separate block chains. Every channel has a separate blockchain, same set of nodes are running.

The contracts, smart contracts associated with each channel are also different but they are running on the same node, but on separate dockers. So they are completely separated. But the ordering service which is transaction agnostic, semantics of agnostic, they do not care what business it is and so on. They just look at the read/write set the version numbers etc., and do the ordering.

So they can be common between the two because you are never sending your code or you know you do a semantics of your data to them, you are only asking them to do the ordering. Now that set of nodes which are doing ordering has a business semantics agnostic infrastructure activity, which is to order transactions. So they can order transactions for this application, they can order transactions for that application, it does not matter.

So that split of trust is possible because your execution is never happening by the same nodes who do consensus. So that is the other idea that is not possible in the order execute paradigm.

(Refer Slide Time: 24:45)

Limits of Order-execute

- **Hard-coded consensus:** all blockchain systems, permissioned or not, came with a hard-coded consensus protocol
- one may want to replace BFT consensus with a protocol based on an alternative trust such as Paxos/Raft and ZooKeeper (Kafka)

And the final thing is that the in bitcoin and blockchain the this thing is hard-coded, the order-execute blockchains they have the consensus algorithm hardcoded. Which means that bitcoin has proof of work, Ethereum has proof of work, but if it wants to go to proof of stake, it has to do a hard fork.

So whereas here since you your ordering service or consensus service is agnostic of the business application, therefore you can decide the system configuration person or the blockchain configurer may decide that I will use simpler consensus mechanism if I trust all the nodes that are involved in consensus. If I feel that there I need crash tolerance, I will use a different type of consensus mechanism.

If I need Byzantine tolerance, then I will use a different type of consensus. If I trust if I do not worry about crash and Byzantine behavior then I can even use a single node which will do all the ordering, which is called a solo. So that also is this consensus is pluggable.

(Refer Slide Time: 26:09)

SMR (State Machine Replication Model)

- Can represent **deterministic** distributed system as *Replicated State Machine*
- Each replica reaches the same conclusion about the system **independently**

So now let us talk about this state machine replication model. Because blockchain as I have mentioned the most of the technology that is used in blockchain is reincarnation of many things that the cryptography community, digital cash community and distributed algorithm and distributed computing community has invented over last 40 or more years.

So replicated state machine is one such model for distributed computing. And then the idea is that if you have a database server, and that database server is actually in the database server, you have you have a dedicated service that accepts your transactions and then execute them on the database and then sends you back the results or success or failure etc. So that service, if it is a single service, then it may crash.

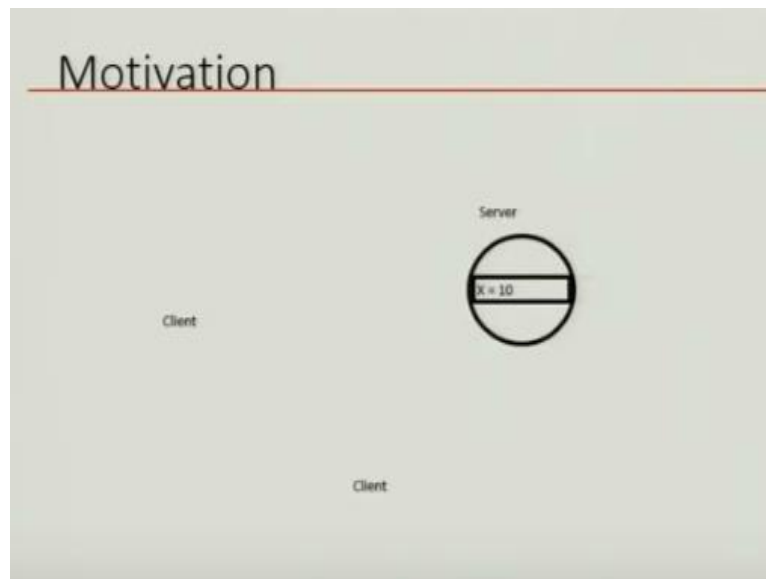
So therefore, people have started thinking about primary backup etc. So primary backup would be that there will be a primary for the service and there will be a replica for the service and the replica will be having the same set state transitions as state changes happen in the original primary service, because, if suddenly it has to take over from the primary because the primary crashed, then it has to be in the right state to continue as if nothing has happened.

So therefore, the question is how do you make sure that one or more replica are in the same state and every time they decide to do the same activity all the time. And this is where the original problem of consensus came about.

So each replica and you cannot have a leader follower thing all the time, because you have to then also elect a leader right, which basically is another way another form of consensus mechanism, that who would be the leader and who would be follower to follow the commands. So therefore, it is best that there should be a way that everybody independently come to the same conclusion about something.

It may be that about choosing the leader and then follow the leader all through, or it may be that in every step, you want to build a consensus about what to do next, or what state to go to next, and then go there everybody independently goes there. But there has to be a consensus.

(Refer Slide Time: 29:03)



So let us look at some simple animations to understand this. So what but before that, we will take a break, and then when I come back, we will start with the animations. So we will see you next time.