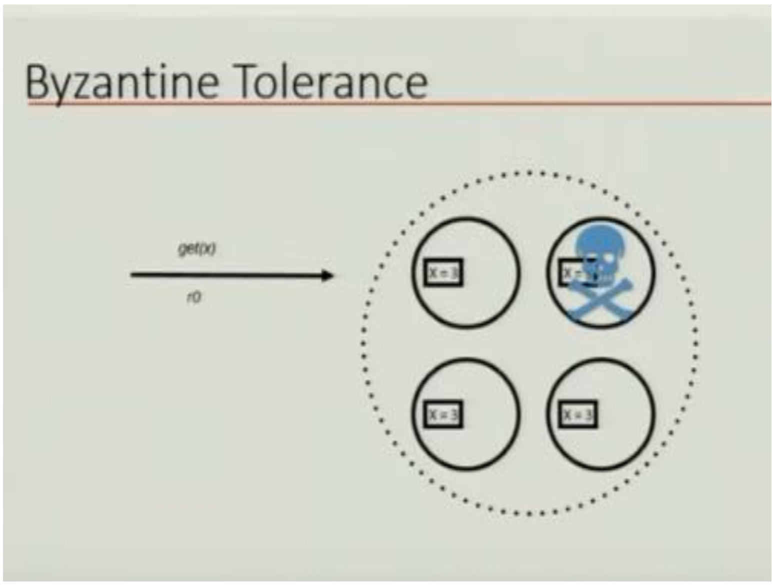
Introduction to Blockchain Technology & Applications

Prof. Sandeep Shukla
Department of Computer Science and Engineering
Indian Institute of Technology-Kanpur

Lecture - 24

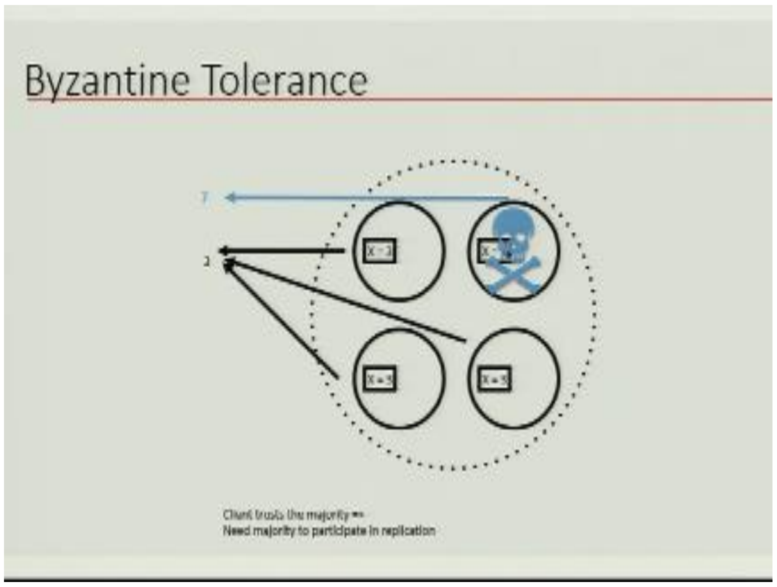
Welcome back. So we were talking about replicated state machine. And we talked about how to order transactions on them. And now we also talked about fail, fail-stop failure. And there we saw that t for, if you assume t of them will fail, then t + 1 servers are needed to tolerate that. Now let us talk about Byzantine tolerance.

(Refer Slide Time: 00:40)



So again, we have four copies of the same state machine. And now let us say one of them decides to behave arbitrarily. So it is a Byzantine node. So now a transaction comes.

(Refer Slide Time: 00:56)

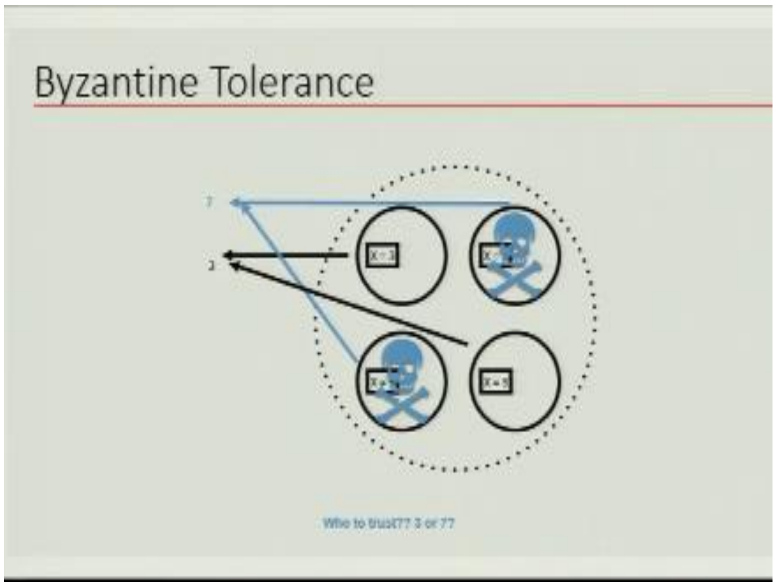


Byzantine Tolerance

Client trusts the majority ⇒
Need majority to participate in replication

Now all the three nodes will give you the correct value that are not affected by Byzantine, but the one that is Byzantine may not answer you or may answer you wrong. So how do you decide? So client will have to do a majority based decision. So client will say, whoever is whatever is the majority saying I will go with that. So in this case, the correct value 3 will be decided by the client as the correct value correct answer to its query.

(Refer Slide Time: 01:29)

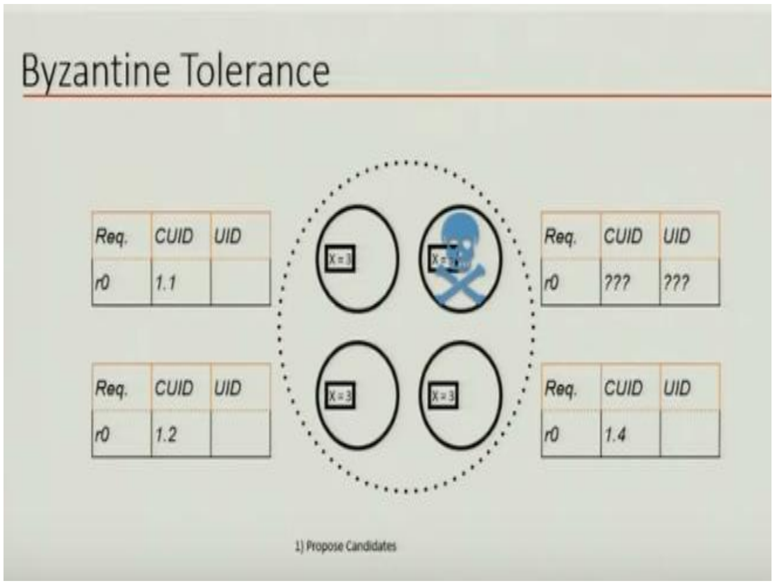


Byzantine Tolerance

Who to trust?? 3 or 7?

Now if two of them does fail, as Byzantine, and two of this Byzantine guys are under undergoing the effect of the same malware, so they decide to actually fool the client with the same value. Now we have a problem because majority voting will not work anymore, because there are two 7 answers and two 3 answers and the client will not be able to make a decision.

(Refer Slide Time: 01:59)



Byzantine Tolerance

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	???	???

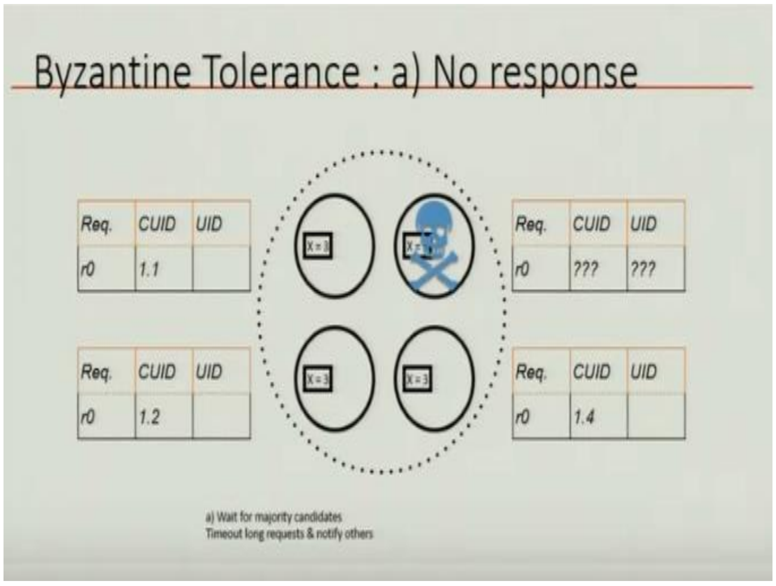
Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	1.4	

1) Propose Candidates

So now again so let us say we want to try this algorithm as before. Now I have r 0, and r 0 has been given this local IDs. But the Byzantine node may or may not give it any ID, or give wrong ID. So in this case, let us assume that the Byzantine node decides to keep quiet and not give any response.

(Refer Slide Time: 02:17)



Byzantine Tolerance : a) No response

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	???	???

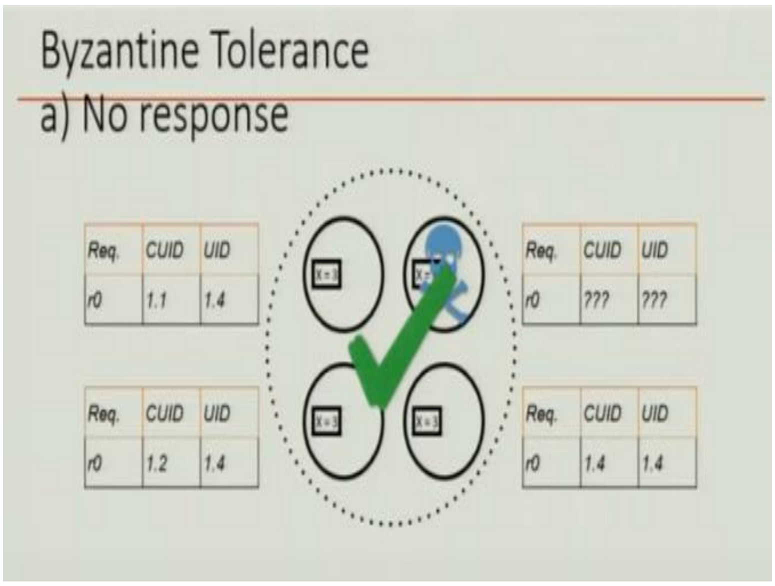
Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	1.4	

a) Wait for majority candidates
Timeout long requests & notify others

So then you basically get the majority of the candidates, and then you use timeout to decide that the fourth answer is not coming. And then you decide the majority is this. So I will based on that I will give it a UID.

(Refer Slide Time: 02:45)



Byzantine Tolerance
a) No response

Req.	CUID	UID
r0	1.1	1.4

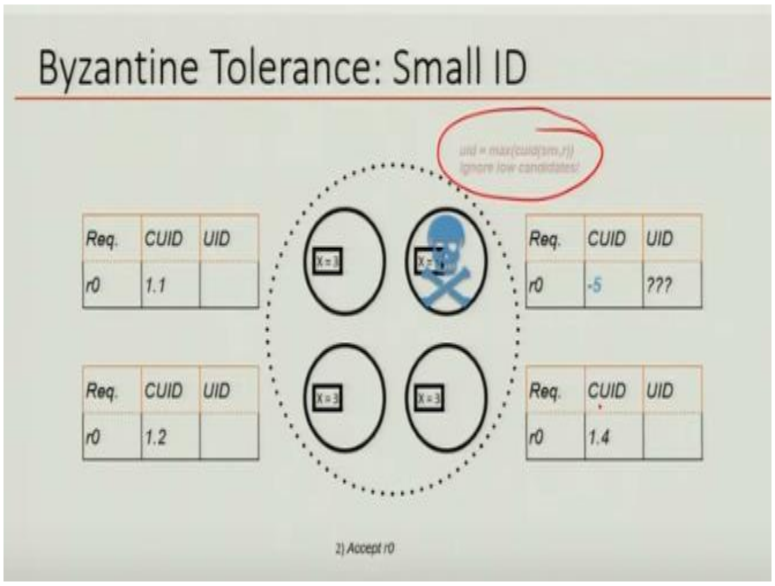
Req.	CUID	UID
r0	???	???

Req.	CUID	UID
r0	1.2	1.4

Req.	CUID	UID
r0	1.4	1.4

So in this case, 1.4 was the highest so you give 1.4, it worked. So one node being Byzantine and not giving any response did not affect your ability to give ID to a agreed upon ID to the transaction.

(Refer Slide Time: 03:07)



Byzantine Tolerance: Small ID

uid = max(cuid(sm,r))
ignore low candidates!

Req.	CUID	UID
r0	1.1	

Req.	CUID	UID
r0	-5	???

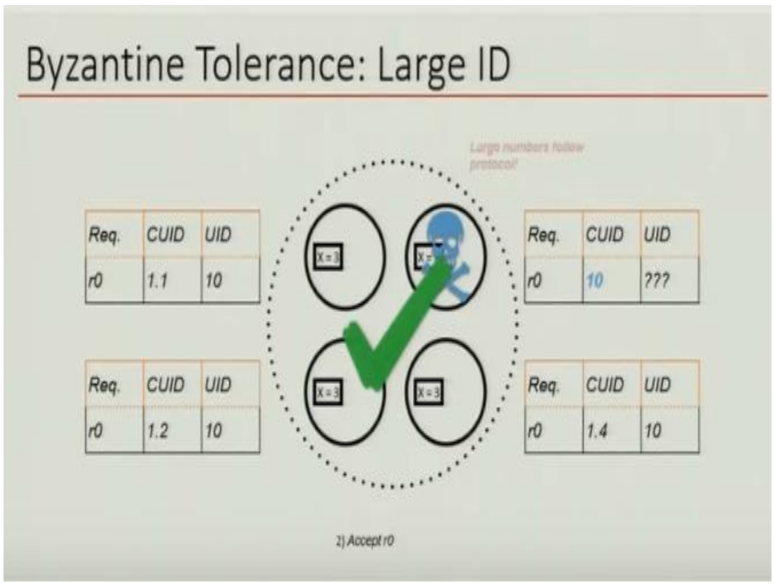
Req.	CUID	UID
r0	1.2	

Req.	CUID	UID
r0	1.4	

2) Accept r0

Now suppose the this other guy, the Byzantine guy wants to behave more maliciously. So he wants to confuse you. So he gives you an ID but he chooses a small ID because he does not know what ID others are choosing. So he gives you an ID let us say -5. Same issue that the algorithm says that you have to take the UID as the maximum among the all the local IDs. So even if it says -5 no problem 1.4 is still the highest.

(Refer Slide Time: 03:40)


Byzantine Tolerance: Large ID

Large numbers follow protocol!

Req.	CUID	UID
r0	1.1	10

Req.	CUID	UID
r0	10	???

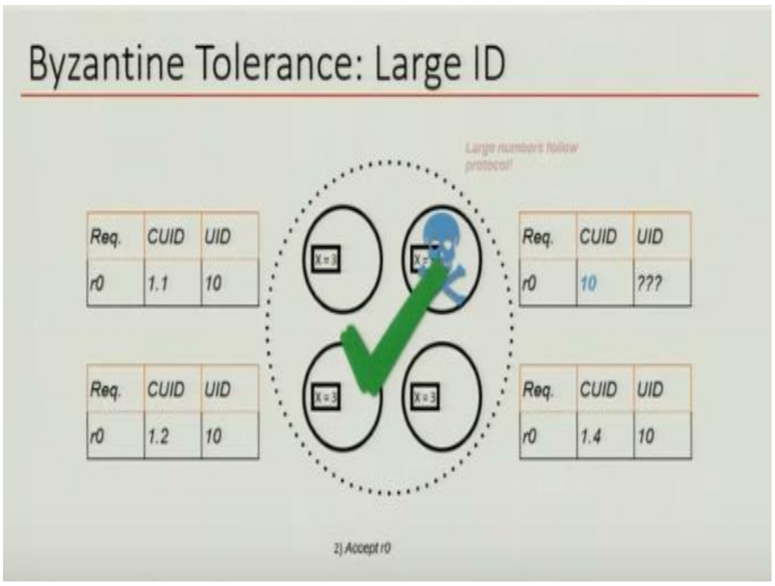
Req.	CUID	UID
r0	1.2	10

Req.	CUID	UID
r0	1.4	10

2) Accept r0

So I am going to choose 1.4 no problem, right. So this also worked.

(Refer Slide Time: 03:44)


Byzantine Tolerance: Large ID

Large numbers follow protocol!

Req.	CUID	UID
r0	1.1	10

Req.	CUID	UID
r0	10	???

Req.	CUID	UID
r0	1.2	10

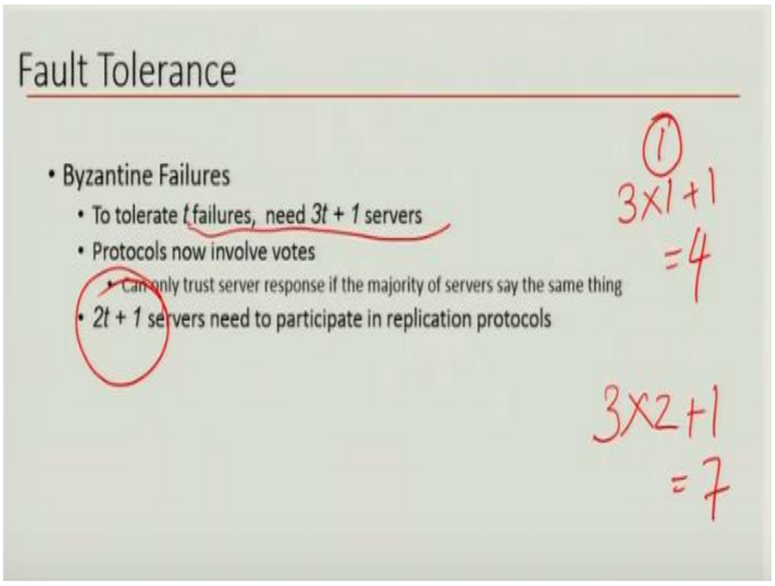
Req.	CUID	UID
r0	1.4	10

2) Accept r0

What if this guy gives you large ID, right? So he gives 10. Now based on the algorithm, now you have to decide on the ID of 10 for everybody else. Now this was fine as long as there is only one transaction. Otherwise, if this happens again for r 1, again the this guy can actually give you a large ID and make r 1 have a different ID. But that does not make any difference because all the other guys will decide the same ID for each of the transactions.

Now that ID might be dictated by the Byzantine node because he is choosing very large ID, or it may be one of the IDs that these three guys give. So that will not be a problem.

(Refer Slide Time: 04:36)



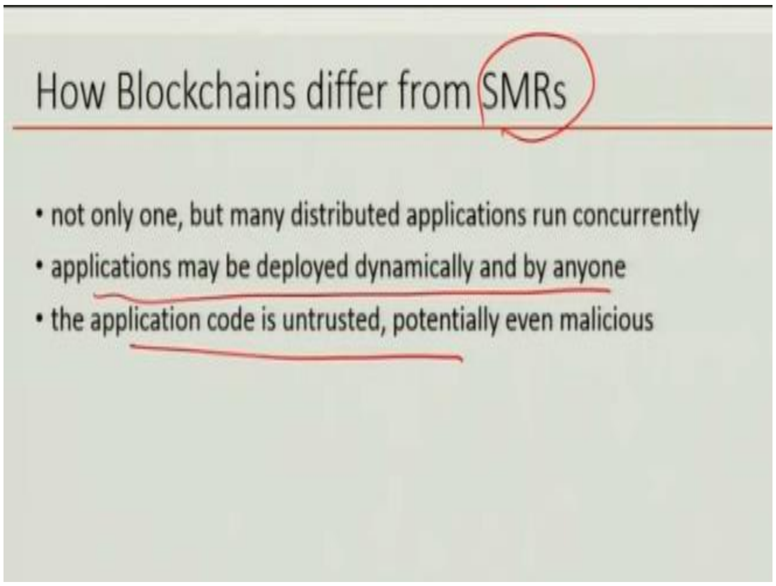
Fault Tolerance

- Byzantine Failures
  - To tolerate t failures, need 3t + 1 servers
  - Protocols now involve votes
    - Can only trust server response if the majority of servers say the same thing
  - 2t + 1 servers need to participate in replication protocols

1
3x1+1
=4
3x2+1
=7

So to tolerate t failures in this case, you will need 3t + 1. So we assume that one node goes Byzantine. So we need three times 1 plus 1 that is 4 servers. So that is what we saw there. But when you saw two failures, we saw that 3 times 2 plus 1 is 7, but we had only 4. That is why we were not able to proceed. And so 2t + 1 servers need to participate in the replication protocol.

(Refer Slide Time: 05:11)



How Blockchains differ from SMRs

- not only one, but many distributed applications run concurrently
- applications may be deployed dynamically and by anyone
- the application code is untrusted, potentially even malicious

So now you understand a little bit about the state machine replication and the models for different kinds of faults. So in case of Hyperledger, this applies only to the ordering nodes. The ordering nodes are involved in making ordering decisions and they have to tolerate Byzantine failure provided the designer of the blockchain decides that there is a possibility that this ordering nodes are not necessarily extremely highly secured.

There may be patches that are missing and therefore they may be infected by malware or it may be taken over by some rogue entity and therefore they are not going to behave properly. Another possibility is that this ordering nodes belong to different organization. For example, if this blockchain is spanning three banks, then each bank may actually provide one of the nodes in the ordering service.

Ordering services is a replicated servers, set of servers, but each of them are coming from different organization. And you are not trusting the other organization. Maybe they will not do anything because of their business ethics or whatever. But you may not you may agree among each other at the business level, that let us not trust each other, let us have Byzantine tolerance.

But if I am going to assume that two of you will behave rogue, then I will have maybe 7 servers to do ordering service. If you if I assume one of them may be going rogue, then I will have 4 etc. They can make that decision. Now blockchains is more than just you know simple replicated state machines. Because replicated machines are just one service like a database service or something running in each of them.

It is the servers associated with the database, let us say. But in blockchain there may be many different services or applications running. As I was saying that same ordering service may be used for one banking, interbanking transaction, log keeping. Another one for supply chain and etc. All of them in the same infrastructure. So the ordering service may be associated with all of them.

Also applications may come and go as applications may be deployed dynamically. And then the application code is untrusted and potentially even malicious, which in this case in Hyperledger it does not matter because ordering service does not run any of the application code. But that is the reason why ordering services separate.

(Refer Slide Time: 07:54)

How Fabric's Execute-order-validate works

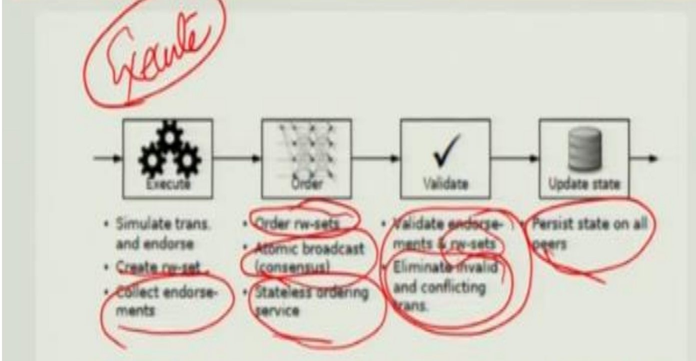

Execute
Execute
Order
Validate
Update state
• Simulate trans. and endorse
• Create rw-set
• Collect endorsements
• Order rw-sets
• Atomic broadcast (consensus)
• Stateless ordering service
• Validate endorsements & rw-sets
• Eliminate invalid and conflicting trans.
• Persist state on all peers

Figure 2: Execute-order-validate architecture of Fabric (*rw-set* means a readset and writeset as explained in Sec. 3.2).

So coming back to a picture similar to what you saw before. In Fabric, Hyperledger Fabric, we have execute. That is where the endorsers execute or simulate the execution of the transactions and create the read/write sets. So they create the read/write set and send the endorsements.

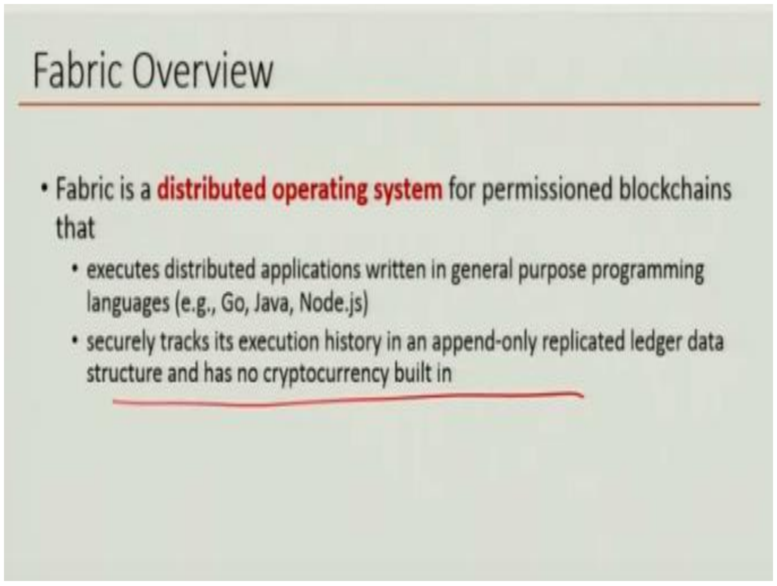
The ordering service then only order based on the read/write sets and do an atomic broadcast among each other and do a you know decide and then this ordering service is stateless in the sense that it does not keep the state at all. It is totally agnostic of what application is running, what data is being kept, etc. It only does ordering.

And then the nodes that are validators, they will validate that a transaction has all the necessary endorsements, and then the read/write sets have the right versions of each of the piece of each piece of data. And then it eliminates invalid and conflicting transactions. And then after that, they send it to every peer.

And then every peer will persist the data by adding to the blockchain the block that was given by the ordering service. And with a bit vector that has one's and zeros. One means that the first if it is 100, that means first transaction is valid second, and third transaction is not valid, etc. So that information goes to every node.

And every node keeps the block with the bit vector so that later on when you retrieve the blocks and want to see which transactions were valid and actually was persisted, and which transaction was tried, but it did not persist all that information there. And this is very important for auditing, especially security auditing. So that is the basic idea of the execute-order-validate framework.

(Refer Slide Time: 10:00)



Fabric Overview

- Fabric is a **distributed operating system** for permissioned blockchains that
  - executes distributed applications written in general purpose programming languages (e.g., Go, Java, Node.js)
  - securely tracks its execution history in an append-only replicated ledger data structure and has no cryptocurrency built in

So Fabric, the Hyperledger authors, they actually consider Fabric more like an operating system which runs applications. So why they are calling it an operating system like why is it not Ethereum or bitcoin not considering it as an operating system? In operating system you have many applications running. Each application may be consisting of one process or multiple processes.

And these applications as they run the processes run their own address space. So there is full confidentiality between two processes. So what is happening inside this process, what is in its memory etc., you know unless there is a cyber-attack based exfiltration of some kind, buffer overflow, etc., these processes are silos in which the application runs.

Or if an application is multiple processes, they may exchange data with each other through shared memory or through, you know RPC etc. But all the other applications do not know what is happening there. In Ethereum, this ability to silo multiple different applications that are running on the Ethereum is not done. So every smart contract for every application are on the blockchain, every node sees them.

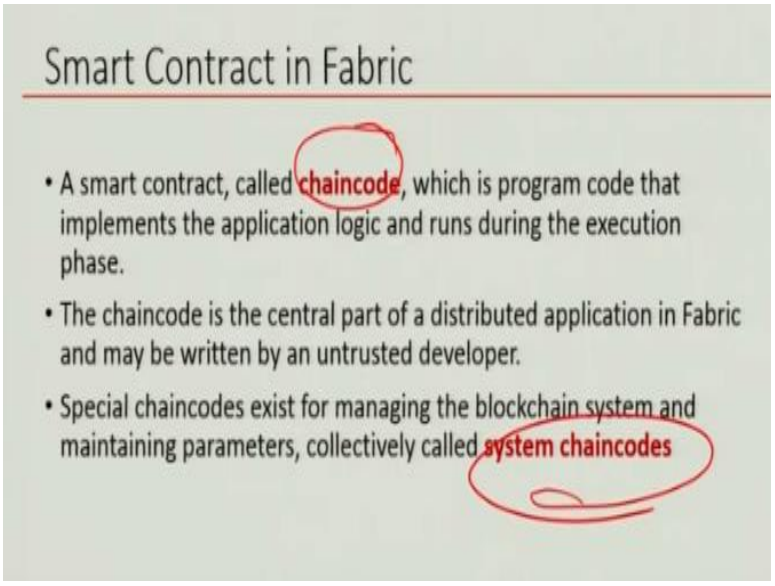
So therefore, we cannot compare Ethereum as sort of like a operating system where if I create a Hyperledger infrastructure, I can have multiple channels, running multiple different applications and the chain code associated with each application will be confidential to each other. So therefore, it is more like an operating system which is enabling multiple applications with full silos.

And they are reusing the infrastructure such as the identity service or the service that gives digital certificates. The ordering service which may be common between all the applications and the actual infrastructure may be all common the network and the actual servers. But each server is running this applications, the chain code in a in a way that you know the chain code runs in Dockers.

So they are siloed out from each other. And also there is a common code that is called the system chain codes that basically does for example, things like validation, etc. These are you know put in as libraries. So it is very much like an operating system. Also in another sense, it is an operating system. Operating system can run applications written in any language. And here also the same thing that they have chosen by design.

And then you can have the additional thing that execution history is kept in replicated ledger which is tamper resistant and so on. And it has no cryptocurrency built in.

(Refer Slide Time: 13:09)

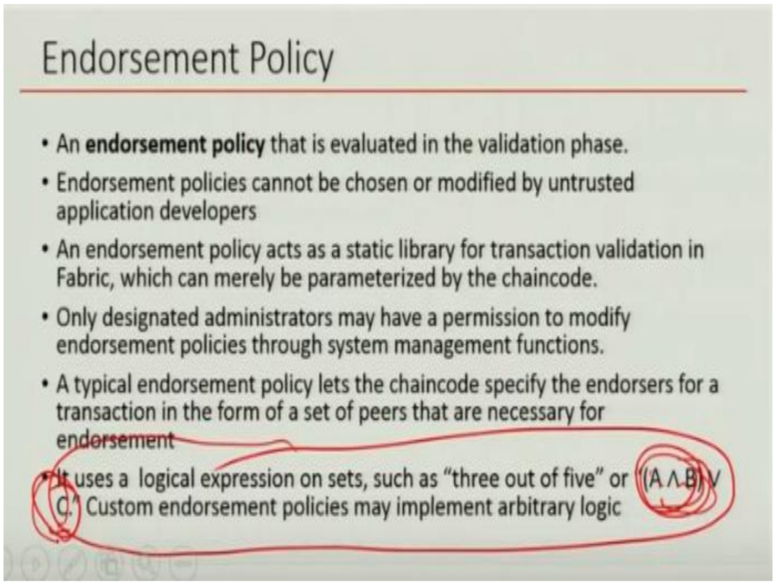


Smart Contract in Fabric

- A smart contract, called chaincode, which is program code that implements the application logic and runs during the execution phase.
- The chaincode is the central part of a distributed application in Fabric and may be written by an untrusted developer.
- Special chaincodes exist for managing the blockchain system and maintaining parameters, collectively called system chaincodes

So again the smart contract is called chain code which basically implements the application logic and runs during execution phase. The chain code is the main thing and fabric and there are also system chain codes as I said that basically are using managing the blockchain system, maintaining parameters etc., and they are called the system chain code.

(Refer Slide Time: 13:36)



Endorsement Policy

- An **endorsement policy** that is evaluated in the validation phase.
- Endorsement policies cannot be chosen or modified by untrusted application developers
- An endorsement policy acts as a static library for transaction validation in Fabric, which can merely be parameterized by the chaincode.
- Only designated administrators may have a permission to modify endorsement policies through system management functions.
- A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement
- It uses a logical expression on sets, such as "three out of five" or "$(A \wedge B) \vee C$." Custom endorsement policies may implement arbitrary logic

So endorsement policy is something that we talked a lot about. But we never explained how this endorsement policy is set, who sets the endorsement policy, what exactly is the effect of an endorsement policy etc. So endorsement policy is evaluated at the validation phase which is a pre commit phase. And the endorsement policy cannot be decided by chain code developers.

So let us say you are a bank and you are running a banking application on Hyperledger. So and the code that is running in the application are designed by vendors, right? So you have you hired some coders, they write the chain code in Go or C or C++ whatever. And they will put them in Dockers, etc. But every time you want, let us say a transaction to be to be endorsed, who will endorse the transaction?

Whether it is just one node or whether it is three nodes and which specific three nodes etc., is part of the endorsement policy. And that is a that is based on the business logic and business policy of the application and the business that is running that application. So they might say that for any transaction below, you know which has an associated value of say 10,000, it has to be endorsed by two nodes and to this kind of specific nodes.

If it is above 10,000 to 100,000 it has to be endorsed by at least three nodes. It is like check endorsement in a business. And that decision is encoded in the application. So endorsement policy becomes part of the application, but it is not part of the chain code that does endorsement. So when a client, let us say wants to do a monetary transaction, let us say like Internet Banking.

He says that from my account send 1 lakh to this other account. Then maybe the policy is that it has to be endorsed by a manager node and a sub manager node and maybe a vice president node or something. So any nodes that are that is running a chain code on behalf of the manager, sub manager etc., will actually be getting that, so client would the client the program that is running on behalf of the client will know that for this kind of transaction I need these three endorsements.

So it will send this transaction in for request or proposal to these three nodes. These three nodes the chain code in there will pre execute the transaction to see whether the balance is available on the user's account and whether the and what would be the value after the transfer and what would be the value at the other place after the transfer and then it will create the read/write sets and then it will digitally sign it.

And then the client gets endorsement back. This endorsement will come from all the three nodes that are supposed to be part of this endorsement policy. Only then the client will know that, well by client I do not mean the human sitting at the terminal, you know logged into this application, let us say through a web front end.

But the application that is running on behalf of the client would know that now I have gotten my endorsement policy has been satisfied. However, then the client will send it to the orderers and orderers will order etc. But as it goes to the validator nodes, the validators will also check whether the endorsement policy has been satisfied for example for this kind of transaction.

So the validators know the endorsement policy for each type of transaction. So they will check whether that has been met. Then they will check whether the digital signatures associated with each of the endorsements check out. And then they will check the read/write sets have the right versions etc. So that is what endorsement is all about.

And normally this endorsement policy is expressed by what is called monotone logic. So like three out of five, or it has to be either these two nodes, or this node may be a very high privilege node. So if this guy endorses, then we do not need these two nodes to do anything etc. So this may be two sub managers, and this may be the manager. See, if two sub manager nodes do it, then it is fine.

But if it is one sub manager, that is not enough. But if the manager himself does it or the program running on behalf of the manager does it then that is fine.

(Refer Slide Time: 18:53)

Fabric Transaction flow

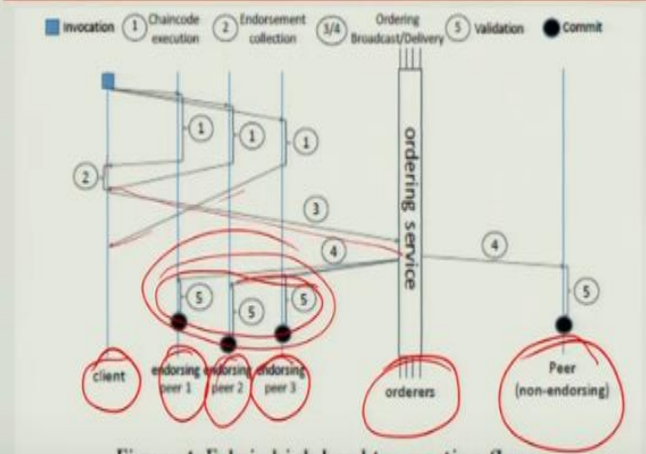


Figure 4: Fabric high level transaction flow.

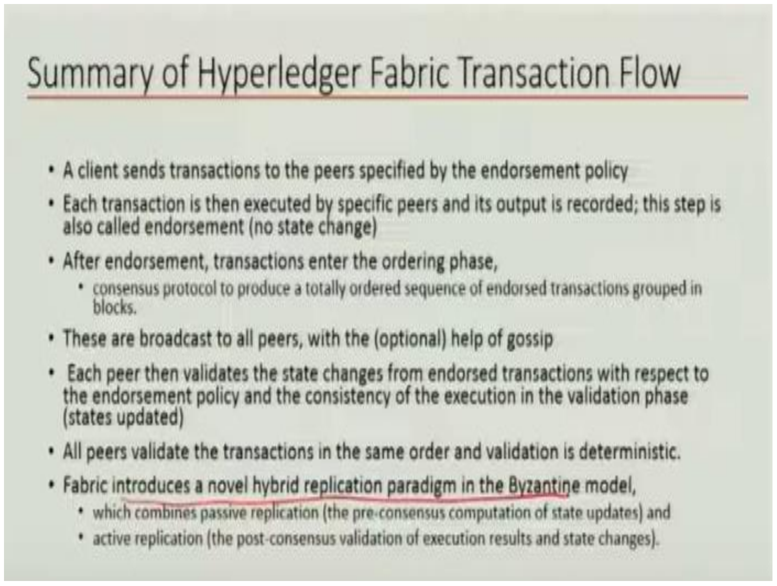
So the transaction flow here looks like this, that you have the client. So this is the client. It invokes the transaction. And then it sends the transaction to the endorsing peer 1, endorsing peer 2, endorsing peer 3. Endorsing peers will, as I said pre execute etc. and then send back endorsement. All this endorsement sent to sent back to the client.

At that point the client will all the three endorsements who come or maybe the endorsement policy is just two. So as soon as it gets two of them then it will send it to the ordering service. The ordering service is maybe made up of multiple nodes because of all that fault tolerant issues that we talked about. And then ordering service will order them and then it will send it. So it will then send it to the validator nodes.

So validator nodes are could be the same nodes as endorsers, but different chain codes So therefore, you know they are running in different Dockers. So same servers are being used for multiple purposes, but they are in silos so they do not talk to each other, see each other. So therefore that is fine, or it may be different servers, it does not matter. But in this case, let us say this is the same nodes as validators.

So then validators will send this to commit and then the and this commit will go to all the other nodes, right. So and then all the stuff will be done.

(Refer Slide Time: 20:37)



Summary of Hyperledger Fabric Transaction Flow

- A client sends transactions to the peers specified by the endorsement policy
- Each transaction is then executed by specific peers and its output is recorded; this step is also called endorsement (no state change)
- After endorsement, transactions enter the ordering phase,
  - consensus protocol to produce a totally ordered sequence of endorsed transactions grouped in blocks.
- These are broadcast to all peers, with the (optional) help of gossip
- Each peer then validates the state changes from endorsed transactions with respect to the endorsement policy and the consistency of the execution in the validation phase (states updated)
- All peers validate the transactions in the same order and validation is deterministic.
- Fabric introduces a novel hybrid replication paradigm in the Byzantine model,
  - which combines passive replication (the pre-consensus computation of state updates) and
  - active replication (the post-consensus validation of execution results and state changes).

So here, I want to summarize the Hyperledger Fabric transaction flow. So a client sends transaction to peers specified by the endorsement policy. Each transaction is executed by specific peers, and its output is recorded. But nobody changes anything in the database or in the blockchain. So we call it a simulation of the execution.

And this step is called the endorsement and there will be no change of state in the terms of the state of the blockchain or any of the data in the database. After the endorsement, the transactions will enter the ordering phase and the consensus protocol will produce a totally ordered sequence of endorsed transactions grouped into blocks. Now these are broadcast to all the peers with the help of a gossip protocol.

So gossip is a protocol by which the node who wants to spread the news to everybody, he will tell its neighbors. And the neighbors will then tell its neighbors and neighbor will not tell back to the neighbor that told it, right. So this is how the gossip protocol works. So they broadcast to all the peers. Each peers then validates the state change from endorsed transaction with respect to the endorsement policy.

And then the consistency of the execution of the transactions by looking at the version numbers of the states of the data that has been that is going to be is being read or written due to this transaction. The ones that are not correct, they will be marked as invalid transaction. And then all state changes for the valid transactions will then be made. And that is what we call the states are being updated.

And the validation process is deterministic, because, you know they will they know the endorsement policy. So they will check whether the endorsement is being made properly, according to the policy and they will check the read/write sets for the validity of the transactions. And then now a Fabric introduces a novel hybrid replication paradigm in the Byzantine model.

So you can have passive replication or and you can also have active replication. So passive replication is because pre consensus computation of state updates. So there is no real replication happening here. And active replication is happening when a post consensus, when everybody replicates the state change and everybody replicates the new block added to the blockchain.

And as you can see that in this scenario, there is no possibility of the blockchain diverging. Because after the ordering service creates a block the only that block is available to be added to the blockchain. And the so there is never will happen that there will be another block added to the last block and then there will be a race between two branches and then eventually one branch will win, all that stuff is gone in this scenario.

And then other thing is that in blockchain in the bitcoin and Ethereum, we saw that all transactions in a block must be valid, right. Otherwise the block will never be accepted. So when each node gets a block that has solved the proof of work puzzle, what it does is that it then checks first that the proof of puzzle solution given in the block is correct. And then it will check whether the transactions are valid by executing the transactions.

And that means that if it finds one of the transaction is invalid, then it will basically tell you that this block is invalid. So then anybody else who has solved the proof of work puzzle, that block will eventually come and then that will be accepted. Now since proof of work is so expensive, nobody actually in their right mind will create a block with invalid transaction because they are also executing the transaction to check validating before they cut the block.

So if I have put so much expense, computational expense to solve the proof of work puzzle, I better not put an invalid transaction because I know that peers will again execute the transactions and they will find that it is an invalid transaction, right. So that is the incentivization mechanism that works in case of bitcoin and Ethereum.

Because if you cut a block with invalid transaction, you are going to lose the money that you would have earned by you know your block being rejected. Since here there is no native currency there is no way to incentivize. Therefore the ordering service or consensus has to be done using standard distributed algorithm concepts and algorithms and also then you have to consider Byzantine failure and what happens, how to solve the Byzantine failure problem.

So that basically brings us to the end of the Hyperledger discussion. In the next class what we are going to do is that we are going to show you two more different types of blockchain technology. One is called IOTA. IOTA is a blockchain that was particularly designed for IoT devices, because IoT devices do a lot of transaction among themselves because smart home and smart transport and smart factory etc., are supposed to have these IoT devices and they cooperate with each other.

So they have to do transactions among each other. So IOTA is a infrastructure where they use a particular type of data structure called Tango. And they have a very different idea of a blockchain. It is in fact, it is not really a blockchain in the sense of blocks. It is actually a directed acyclic graph. So we will look at that, you know not in as much details as we have seen this blockchains.

But to give you a flavor of what the other kind of blockchains that are being developed for very different purposes, we can discuss that. Second thing that I want to discuss is another blockchain called Corda, which is again a business blockchain. So in some way, it is more closer to Hyperledger. But in some ways, it is very different in purpose and design than Hyperledger.

But we will discuss Corda also to show you what the financial technology financial world, the banks etc., they are thinking in terms of a blockchain based smart contract system, where the contracts are actually the part of the design and language and there we are talking about financial business contracts. Not the contract in the sense that, you know I, if you give me this, I will give you that kind of thing.

I mean, it is kind of that kind of thing, but, but it is more formalized. And the blockchain is designed with a much more formalization of business contract recording as well as business contract execution. So these two blockchain examples will be given lot more briefly than we have been doing with this three these three major block chains. Because in this course, we wanted to give you a very good conceptual understanding of what blockchain technology is about.

What where it is going? What are the different types of blockchain, it is not all size fits all, same size fits all. So there are different types of solution for different purposes and they have very different design, different intent. And then, so these two will give you some more exposure to this kind of blockchain. Once we have exposed you to that those two, of course there are many more, right.

So there are at least now about 100 probably different types of blockchains. Now the question is, once you have gotten the this very basic ideas, you will be able to choose based on your application needs and your business needs, etc., what is the right type of blockchain that you might use. Finally, we will towards the end of the course, we will talk about some applications.

We talked about applications in a very abstract manner like you know we talked about, you know supply chain, banking, finance, monetary system etc. But now we will talk about a few concrete examples of the blockchain technology being applied to non cryptocurrency applications. And hopefully with all this, you will be in a position to actually have a very conceptual knowledge of blockchain technology and applications.

And the ability to articulate what your problem requires if you want to have a blockchain solution. Also you know you should be able to articulate when your application does not need a blockchain. So we will discuss all that in the last week of classes. So we will see you next time.