Blockchains Architecture, Design and Use Cases

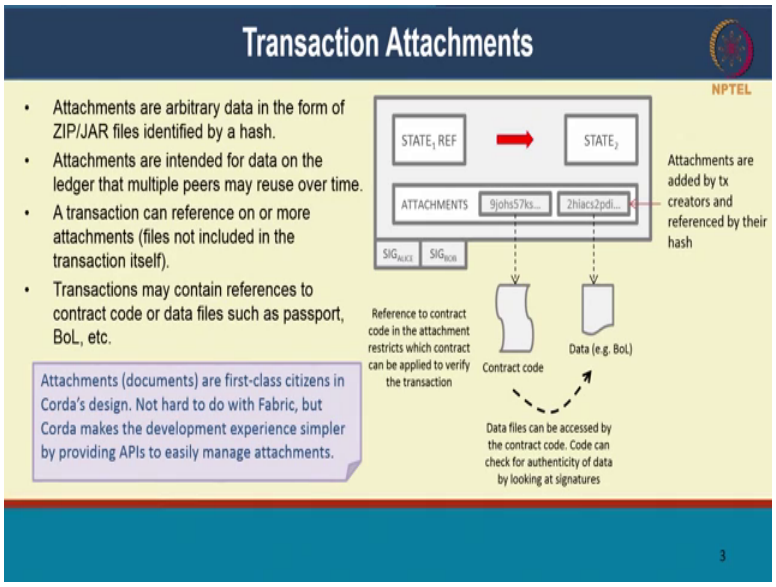
Prof. Sandip Chakraborty
Prof. Praveen Jayachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
IBM Research, India

Lecture - 59
Comparing Ecosystems - Corda Part 2

Hello everyone. Welcome back to our Blockchains course. This is going to be the second part on Corda. So, we had a brief overview of Corda, we looked at what Corda transactions look like some of the state information high level overview.

Now, we going to and talk about how transactions itself work how its consensus reached, notion of notaries and how oracles are used. So, we will talk about some of those concepts in this part. So, before we get that one of the key for design principles is the notion of attachments. So, I mentioned about having contract code associated with legal agreements or legal proofs.

(Refer Slide Time: 00:57)



Transaction Attachments

- Attachments are arbitrary data in the form of ZIP/JAR files identified by a hash.
- Attachments are intended for data on the ledger that multiple peers may reuse over time.
- A transaction can reference on or more attachments (files not included in the transaction itself).
- Transactions may contain references to contract code or data files such as passport, BoL, etc.

Attachments (documents) are first-class citizens in Corda's design. Not hard to do with Fabric, but Corda makes the development experience simpler by providing APIs to easily manage attachments.

NPTEL
$STATE_1$ REF
$STATE_2$
ATTACHMENTS
9johs57ks...
2hiacs2pdi...
SIG_{ALICE}
SIG_{BOB}
Attachments are added by tx creators and referenced by their hash
Reference to contract code in the attachment restricts which contract can be applied to verify the transaction
Contract code
Data (e.g. BoL)
Data files can be accessed by the contract code. Code can check for authenticity of data by looking at signatures
3

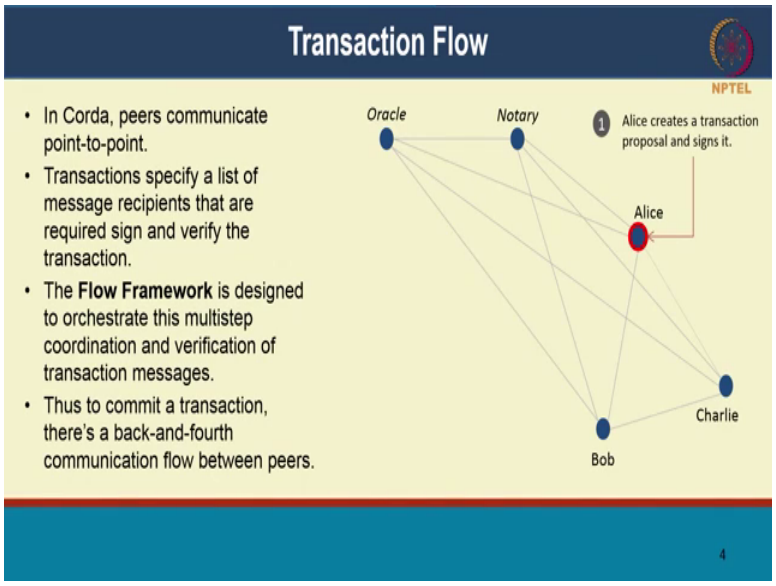
So, they can be provided as attachments to a transaction. So, if you look at a particular transaction it not only specifies input states and output states, it can optionally provide a set of attachments. So, it can refer back to the attachment refer back to the contract code, it can refer back to external documents; maybe this might be physical documents or scanned documents for instance.

All of these can be added as attachment to support a transaction. So, these attachments can be arbitrary data in the form of ZIP or JAR files identified by a hash. And the JAR files are typically for contract code reference itself. And attachments are intended for a data on the ledger that multiple peers may be used over time. So, this is supporting documentation in some sense. A transaction can reference one or more attachments and it may contain references to data files also.

So, for instance, it can refer to an identity document, it can refer to a bill of lading which is a trade document, for if you are doing trade finance for instance. And attachments are of first class citizens for Corda right. In any other blockchain platform hyper ledger fabric, an ethereum you can of course store attachments but it does not come as a first class citizen, as part of your transaction structure itself and that I believe has value right.

It does make things a little easier for the for an application developer, especially, in I think financial services where, typically there are financial documents that need to be attached with a particular transaction. So, now, coming to the notion of transaction flows, so these transaction flows are the ones that are going to validate the sequence of steps that are needed to make a transaction legitimate and be committed on the ledgers of different peers right, that are different transacting.

(Refer Slide Time: 02:50)



Transaction Flow

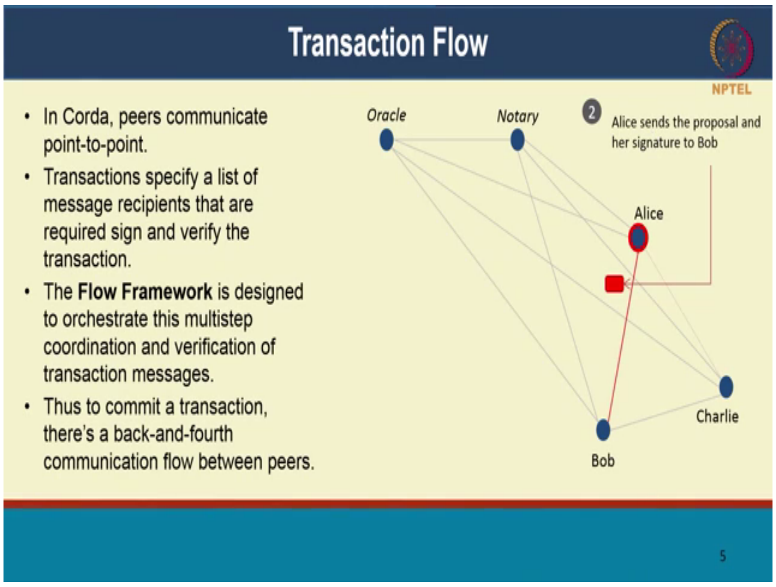
- In Corda, peers communicate point-to-point.
- Transactions specify a list of message recipients that are required sign and verify the transaction.
- The **Flow Framework** is designed to orchestrate this multistep coordination and verification of transaction messages.
- Thus to commit a transaction, there's a back-and-fourth communication flow between peers.

So, let us say, we have an example where, we have Alice, Bob and Charlie in a network and there is a 1 notary and 1 oracle. This notary think of it as a service, so it can actually be multiple nodes there, but in logically it is a single entity. So, as I mentioned Corda all communication is point to point and the transactions will along with the contract logic itself, the transactions specify a list of message recipients that are required to sign and verify the transaction.

So, the flow framework is then designed to orchestrate this whole thing. So, I will say that first Alice needs to sign, then Bob needs to sign and then it can be finalized right. And the logic that Alice and Bob used to sign this transaction can also be part of the flow or it can also be external information. So, let us go through this flow rate. So, first let us say Alice creates a transaction proposal and signs it. So, it is going to take some input back transform it into some output, input and output can be null as well or one of them can be null for instance.

And this transaction state along with attachments is signed by Alice. Saying here is a proposal for a transaction.

(Refer Slide Time: 04:04)

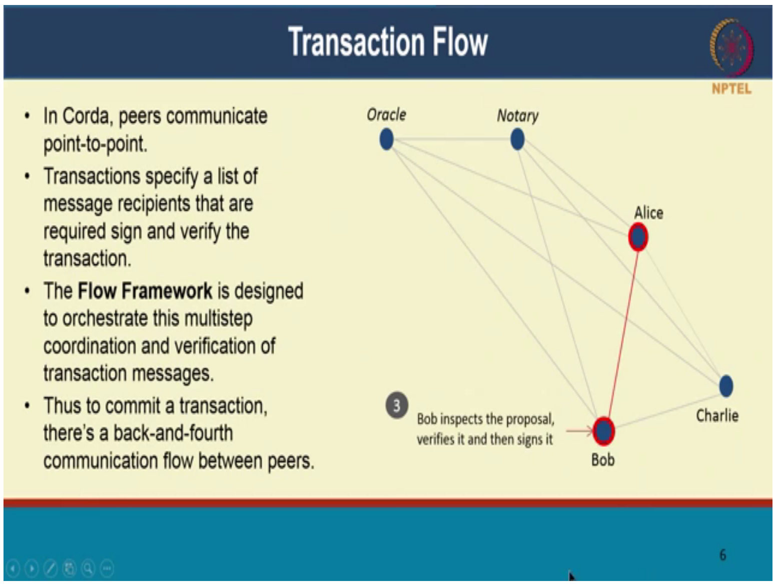


Transaction Flow

- In Corda, peers communicate point-to-point.
- Transactions specify a list of message recipients that are required sign and verify the transaction.
- The **Flow Framework** is designed to orchestrate this multistep coordination and verification of transaction messages.
- Thus to commit a transaction, there's a back-and-fourth communication flow between peers.

Alice then sends this proposal using a point to point message to Bob.

(Refer Slide Time: 04:11)



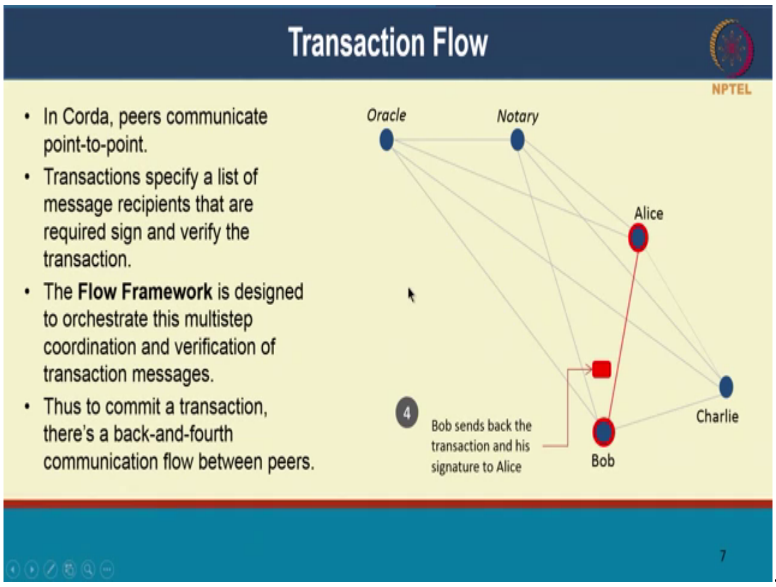
Transaction Flow

- In Corda, peers communicate point-to-point.
- Transactions specify a list of message recipients that are required sign and verify the transaction.
- The **Flow Framework** is designed to orchestrate this multistep coordination and verification of transaction messages.
- Thus to commit a transaction, there's a back-and-fourth communication flow between peers.

Bob on receiving this proposal can inspect it, can use any logic of their choice, this is off contract logics, this is not the contract logic itself.

But it can look at the contract executions see if the state derived by Alice is correct and it can also use external information to see whether, Alice is making if this is a payment for instance whether, Alice is making a payment in time for instance. You can use all sorts of logic that is private to just Bob and use that information to determine whether it wants to sign it or not.

(Refer Slide Time: 04:49)



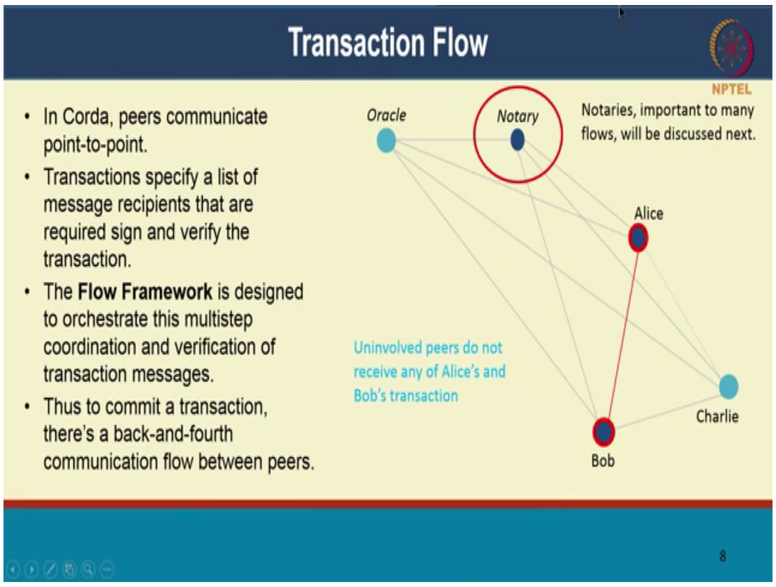
Transaction Flow

- In Corda, peers communicate point-to-point.
- Transactions specify a list of message recipients that are required sign and verify the transaction.
- The **Flow Framework** is designed to orchestrate this multistep coordination and verification of transaction messages.
- Thus to commit a transaction, there's a back-and-fourth communication flow between peers.

.

Once it agrees with this proposal from Alice, Bob attaches its signature also and sends it back to Alice right. So, now, both have this transaction, that is signed by both parties. And let us say that was all that was needed for this transaction to complete.

(Refer Slide Time: 05:06)



Transaction Flow

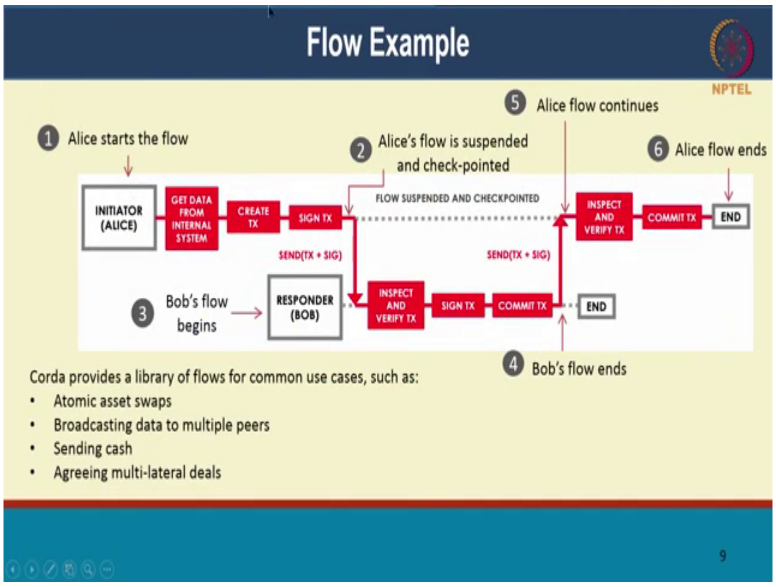
- In Corda, peers communicate point-to-point.
- Transactions specify a list of message recipients that are required sign and verify the transaction.
- The **Flow Framework** is designed to orchestrate this multistep coordination and verification of transaction messages.
- Thus to commit a transaction, there's a back-and-fourth communication flow between peers.

Notaries, important to many flows, will be discussed next.

Uninvolved peers do not receive any of Alice's and Bob's transaction

This transaction could potentially be sent to a notary. And the notary can attest to the fact that there is no double spend that is happened for instance in the network. It provides certain functionality. So, the rotaries can also be part of the flow itself and that is also captured before transaction finality is reached.

(Refer Slide Time: 05:26)



Flow Example

Corda provides a library of flows for common use cases, such as:

- Atomic asset swaps
- Broadcasting data to multiple peers
- Sending cash
- Agreeing multi-lateral deals

So, let us just take a flow example. So, we start with Alice creating a flow, starting a flow. It creates a proposal, it creates a transaction, it signs the transaction, sends it to Bob and that is where Bobs flow begins. So, at this point Alice s flow is suspended, it is check pointed and Alice is going to wait right.

Even if Alice goes down and comes back up at a later point, its flow can resume. Bob once at received the transaction it is again going to a inspect and verify if then signs the transaction and then now it has signatures from both Alice and Bob right, so now, it can commit that transaction. It commits that transactions and you can also send that transaction, the signed transaction by both parties it sends it back to Alice.

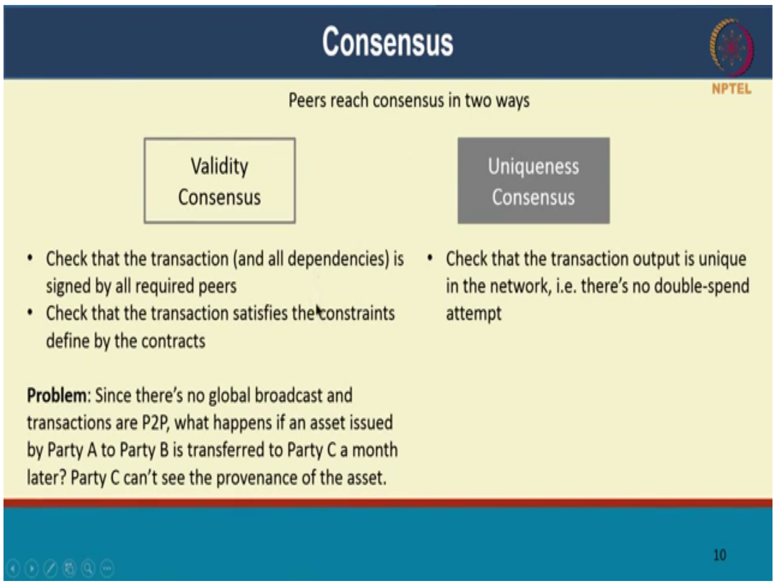
Alice again inspects and commits. So, that is the full flow. So, if you look at Bobs flow, this what Bob did, Alice in between waited for Bob’s signature and then it also committed; so, at this end of this both nodes of committed. In case Alice did not receive this message, it can timeout and it can ask Bob whether did you agree to it or not, it can always fetch that back from Bob.

And because all of these are recorded immediately, once both of them commit no one can go back on that transactions. Because some of these flows can be complicated, it can involve multiple parties, here is it is just a two party flow.

These flows can involve multiple parties and multiple signatures. Corda provides a library of flows for common use cases, such as atomic assets swaps, basically, assets are exchanged between parties, broadcasting to multiple peers, sending cash information. This broadcast is basically achieved through multiple uni cast right, so agreeing multilateral deals.

So, for some of these common flows that are seen in financial use cases Corda provides a library of these flows. So, these flows together with the contract logic itself comprise the whole transaction lifecycle and how transactions from proposals to commit how they have.

(Refer Slide Time: 07:38)



Consensus

Peers reach consensus in two ways

Validity Consensus

- Check that the transaction (and all dependencies) is signed by all required peers
- Check that the transaction satisfies the constraints define by the contracts

Problem: Since there's no global broadcast and transactions are P2P, what happens if an asset issued by Party A to Party B is transferred to Party C a month later? Party C can't see the provenance of the asset.

Uniqueness Consensus

- Check that the transaction output is unique in the network, i.e. there's no double-spend attempt

10

So, now we have only talked about the transaction execution, we need to talk about how consensus is reached between peers right, at least peers, who are transacting with each other.

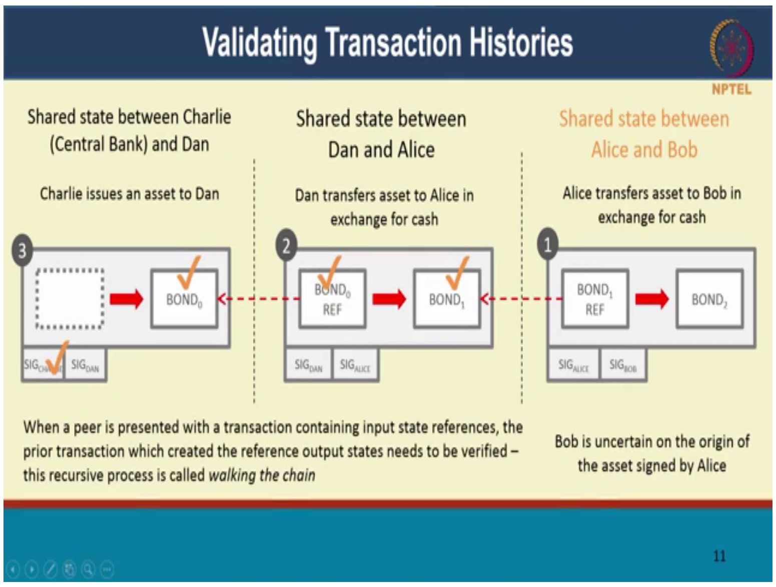
So, Corda distinguishes two notions of consensus. One of them is validity consensus, which is to say that this transaction is actually subjected as is meeting all criteria, it is all the dependencies are met, all the required parties have signed. That makes the transaction valid right, all the constraints are satisfied, so that is the validity consensus. The uniqueness consensus is checks whether the state, I mentioned the UTXO's state information right it checks whether the UTXO has not been spent right.

So, the transaction output is really an unspent transaction. Sorry, the transaction output is unique right. So, no one is trying to double spend in this network. So, that is a uniqueness consensus. So, the validity is separated away from uniqueness. We will see how both of these are achieved. Now, one of the problems that you could see is since, there is no global broadcast, and transactions are all peer to peer, now what are the problems that can happen is, if an asset is issued by one party and it is transferred to another party B and then to C, for C to verify a let us say a month later, they will have to go through the provenance of the entire asset to see if all the transactions have been (Refer Time: 09:10). In some ways that is similar to how are financial transactions and other transactions also happen in the real world. For instance, let us say you are buying a new piece of land.

Before you buy the land you will probably get the services of a lawyer or you will probably do it yourself. You will verify whether this piece of land this legal document has changed hands in a legitimate manner, maybe you will look at a 100 year history, it might have changed hands maybe dozens of times.

You will go through all that history of transactions that whole provenance, that provenance is not recorded in blockchain, it will be great if it is. But what Corda is providing is that provenance is recorded on blockchain, but you will still have the overhead of having to go through and verify every one of those transactions because, all that is new to you right. You are a new buyer for this piece of land. Before you buy it, you will have to verify all of those previous transactions on exchanges, make sure they are legitimate before you can accept it. So, that is an overhead Corda will incur alright with especially assets that are changing hands.

(Refer Slide Time: 10:17)



Validating Transaction Histories

Shared state between Charlie (Central Bank) and Dan

Charlie issues an asset to Dan

Shared state between Dan and Alice

Dan transfers asset to Alice in exchange for cash

Shared state between Alice and Bob

Alice transfers asset to Bob in exchange for cash

When a peer is presented with a transaction containing input state references, the prior transaction which created the reference output states needs to be verified – this recursive process is called *walking the chain*

Bob is uncertain on the origin of the asset signed by Alice

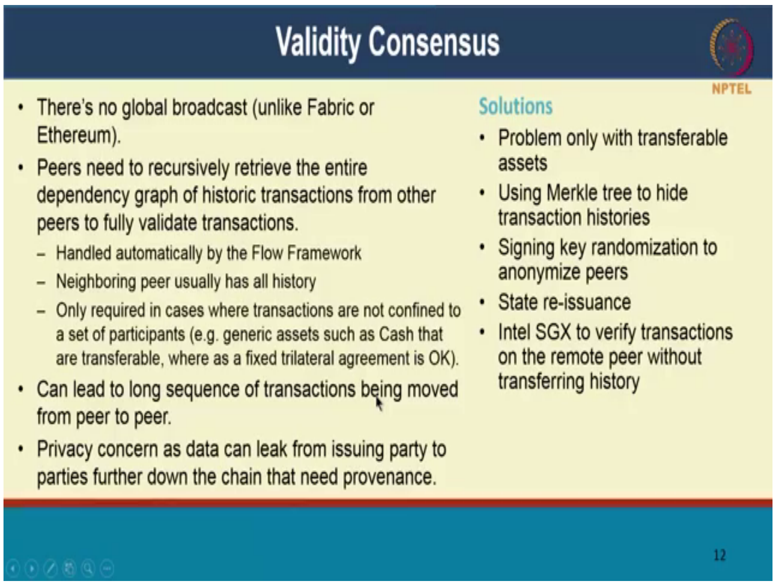
So, now how, does validation of transaction histories work right? So, let us say there is a Central Bank, that issues a particular asset to Dan.So, this is a bond that is issued. So, Charlie is the central bank here. So, it is signed by Charlie and once Dan accepts it, Dan also signs it with Dan’s with his signature. So, that is a transaction that is recorded on blockchain. Now, let us say Dan shared that same asset with Alice. So, the reference to the previous bond is made and a new state is created. So, each of these this it is not a the same state that is modified, it is actually a new state that is created, that is how its remains unspent.

So, this is how the unspent transaction, this becomes a spent transaction, sorry a spent state. Now, this transaction is signed by both Dan and Alice.

And now let us say Alice shares this with Bob. Now, Bob needs to verify the whole chain; the fact that Charlie issued to Dan, Dan then issued to Alice. All that has to be verified by Bob that this is actually a legitimate piece of state that has come from the Central Bank and has gone through a certain set of legitimate transactions, so, I can believe this, so, then I will be willing to purchase that bond.

So, then, the final transaction is also signed by Alice and Bob. Now, whenever a peer is presented with the transaction containing input references that it is not seen before then, it needs to fetch all of this and verify this. So, this is called walking the chain right. This is a recursive process and it can could be time consuming.

(Refer Slide Time: 12:09)



Validity Consensus

- There's no global broadcast (unlike Fabric or Ethereum).
- Peers need to recursively retrieve the entire dependency graph of historic transactions from other peers to fully validate transactions.
  - Handled automatically by the Flow Framework
  - Neighboring peer usually has all history
  - Only required in cases where transactions are not confined to a set of participants (e.g. generic assets such as Cash that are transferable, where as a fixed trilateral agreement is OK).
- Can lead to long sequence of transactions being moved from peer to peer.
- Privacy concern as data can leak from issuing party to parties further down the chain that need provenance.

Solutions
- Problem only with transferable assets
- Using Merkle tree to hide transaction histories
- Signing key randomization to anonymize peers
- State re-issuance
- Intel SGX to verify transactions on the remote peer without transferring history

12

That could be could act as a negative for Corda because of the fact that they do not have a global blockchain, people are not validating transactions on a continuous basis but only on a need to know basis, when they are accessing that asset right. So, that is a inherent trade off that probably Corda is ok with that is why they have designed it this way.

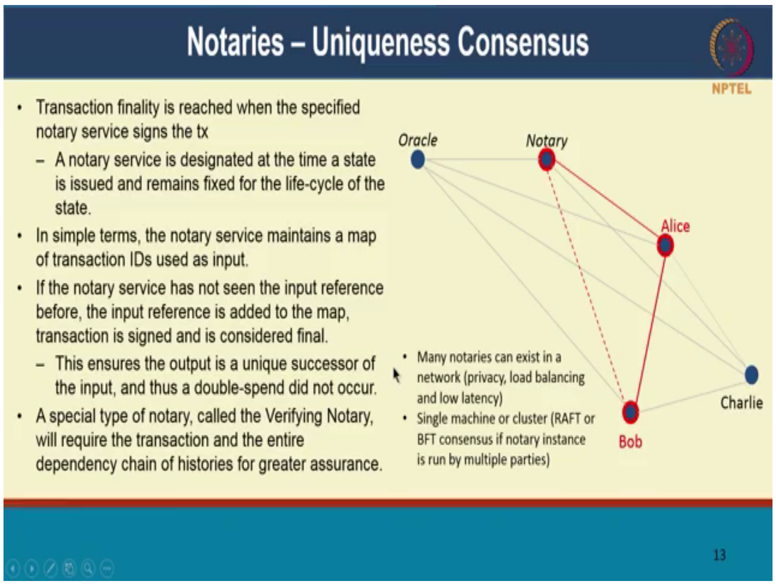
But, one thing with the Corda one thing Corda does is this walking the chain and verifying these past transactions is done automatically by the flow framework. So, the flow framework can fetch all of these transactions, verify it and that is all automated for you but it still incurs a computational overhead.

So, the in only required in cases where, so, this I think this negative that I am talking about here at least by the way treated as my own perspective here. This only this is only needed for assets that are being transferred right. For assets that are not transferred across users, let us say there are just two parties that are trading these assets right, so then, there is no verification that is needed, there is no walking the chain that is needed right.

So, it is only when parties are seeing assets that they did not see before will this additional validation, verification and walking the chain will be needed. Now, there is also a privacy concern as data can be leaked from the issue parties to parties further down the chain, but a Corda is working through multiple alternatives to address that privacy issue. So, there are they are looking at signing key randomization, anonymizing peers so that the peer identity is not revealed.

They are also looking to use a trusted platform such as, Intel SGX to verify transactions on the remote peer without transferring history. So, there are many alternatives that are being considered. So, the other alternative is also state reissuance. So, it is as if, there could be a say maybe a central authority that comes back in and says yes this is a valid state, I will issue an a new state for you right, so all that is possible.

(Refer Slide Time: 14:17)



Notaries – Uniqueness Consensus

- Transaction finality is reached when the specified notary service signs the tx
  - A notary service is designated at the time a state is issued and remains fixed for the life-cycle of the state.
- In simple terms, the notary service maintains a map of transaction IDs used as input.
- If the notary service has not seen the input reference before, the input reference is added to the map, transaction is signed and is considered final.
  - This ensures the output is a unique successor of the input, and thus a double-spend did not occur.
- A special type of notary, called the Verifying Notary, will require the transaction and the entire dependency chain of histories for greater assurance.

- Many notaries can exist in a network (privacy, load balancing and low latency)
- Single machine or cluster (RAFT or BFT consensus if notary instance is run by multiple parties)

So, that was the validity consensus and how, peers validate whether all the history of transactions are valid. Now, the uniqueness consensus is provided by a special entity called notaries right. In some sense notaries are similar to fabric, but I will highlight of the similar to order the ordering service in fabric but I will highlight the difference as we talk through this right. So, once let us say the Bob and Alice both sign this transaction, they can actually send it to a notary and say hey notary tell us whether this is this transaction is really unspent right.

So, the notary service is designated at a time, the state is issued right. So, then for each state at the time of the issuance of the state, you can specify a notary service that will determine the determine the uniqueness of that state right. Let us say it is a bond a state representing bonds, so whenever that bond is created, that state you also specify this node is going to be the notary for this bond and that notary will take care of ensuring uniqueness. So, what the notary will do is fairly simple, if it receives the transaction and it has inputs that has it has not seen before then it will be added and it will be the notary will sign it and say yes this transaction can be admitted because I have not seen the inputs of this transaction form, this is still is definitely unspent and notary will sign it and send it back to the send it back to Alice and Bob right.

A special type of notary called a verifying notary will require the entire dependency chain of histories for greater assurance. So, this is a verifying notary will not only do uniqueness consensus, but will also do a validation consensus that we talked about right. It will validate the entire history of transactions and on top of that will also ensure that this transactions input has been unspent. So, of course, the notary will only permit one of if there are conflicting transactions that are trying to use the same input states, the notary is going to be the tiebreak, so it is going to say ok.

This transaction will go through first, the second transaction will be invalidated. Note that, many notaries can exist in a network. The notaries are created at the level of states. So, based on the states, different notaries can be present in the network for different states. This can provide not only privacy, but it also provides load balancing and ensures lower latency right. If everyone all the state information maintained in the network went you just one node of course, that node is going to become a bottleneck, it is also a privacy concern. So it is possible to separate out notaries for different states.

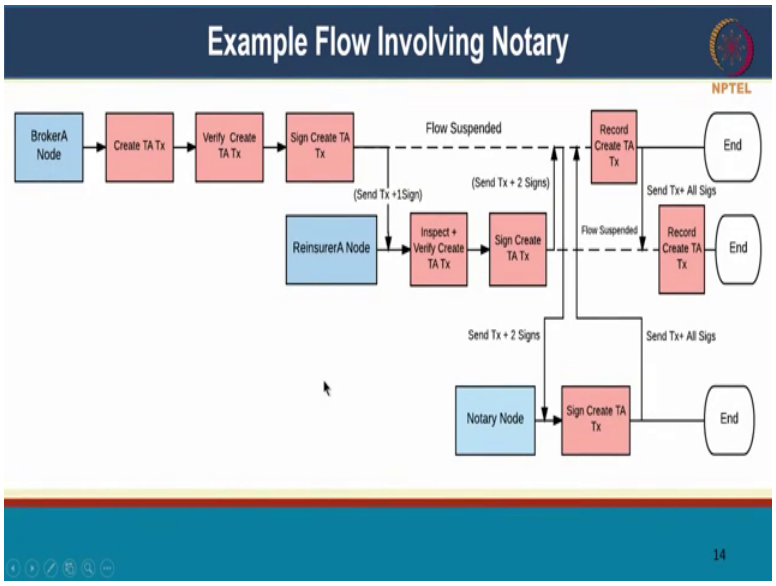
Now I mentioned ordering service in fabric. Similar to that the ordering service in fabric also provides some notion of uniqueness but not really so right. The ordering service in fabric just orders transactions, it does not do anything else, it does not determine whether two transactions are conflicting with each other, whether they are unique, it does not look at that.

It simply looks at the transaction almost as a black box and says here are 100 transactions I am going to order them 1 to 100 that is all the ordering service in fabric does. However, the notary in Corda actually needs to look into the input states and output states of the transaction itself. So, it has to be private to the transaction details and only then can it verify whether this is this transaction is unique.

So, in addition to just ordering and avoiding duplication, avoiding double spend, notaries also look at the have to look at the data to an in order to be able to say that, double spent has not happened. I mentioned notaries, again notaries is a logical concept. It can be a single node, it can be a cluster, it can implement any arbitrary logic within itself within that service to determine this uniqueness right; it can be draft based consensus it can be byzantine fault organ consensus, so all that is possible within a notary service.

So, again these kinds of consensus algorithms are similar to the ordering service. Ordering service in fabric uses it will be for ordering here it does a little more, in actually determining whether double spent has is happening or not ok.

(Refer Slide Time: 19:08)


Example Flow Involving Notary

So, here is an example transaction flow involving a notary. So, let Us say there is a Alice broker known for Alice, that Is going to create a transaction, it Is going to verify this transaction is well formed it is going to sign it.

It then sends it to another node, so this could be a reinsurer so, this would be Bob. The Bob then verifies this transaction, Bob also signs it, sends it back to Alice. Now, both parties have signed this transaction. Alice can then send this to the notary. The notary will then determine whether, this is indeed unspent right. Once it determines that it is unspent, it might be receiving many such transactions for multiple parties this the notary will then order all of them through either a single machine is ordering them either or multiple nodes are employing consensus to order them.

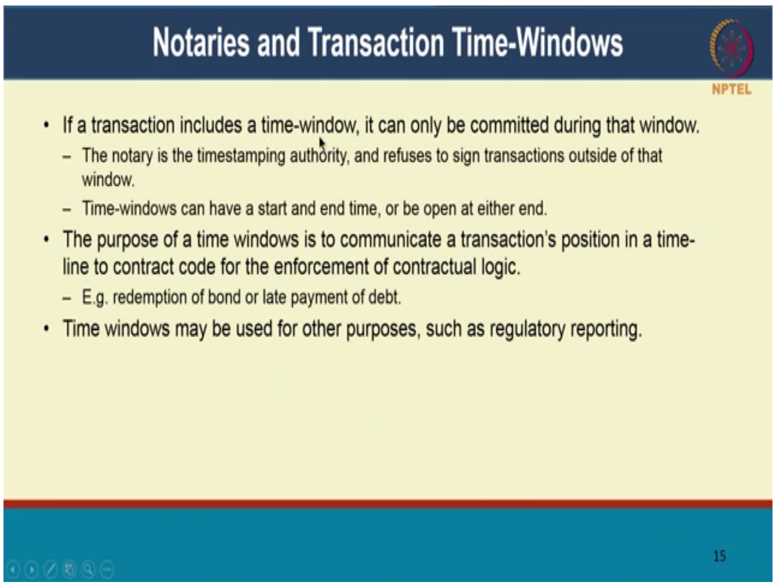
It will then detect whether, there are duplicates and only if it is a it is not double spent if it is a unique transaction output. That I will sign that transaction, so, the notary adds its signature and sends it back. Now, this is the point where notary signs it is when the transaction is considered final.

And then, Alice and Bob and then commit those transactions into their respective ledgers. Likewise, Alice and Charlie are if a different node might have their own a flow working on different states but they can employ the same notary service for the same what we service to order.

So, then the notary you service will check whether, Alice is trying to send the same state to both Bob and Charlie and it will prevent others from double spending. So, unless can then leverage the notary service to prove to Bob and Charlie that it has not doubled spent this transaction right.

So, that is why it is the notaries are specified at the level of state that way, this notary can verify that this state is unspent. And different states can employ different notary services because; there is no double spending problem across them.

(Refer Slide Time: 21:14)



Notaries and Transaction Time-Windows

- If a transaction includes a time-window, it can only be committed during that window.
  - The notary is the timestamping authority, and refuses to sign transactions outside of that window.
  - Time-windows can have a start and end time, or be open at either end.
- The purpose of a time windows is to communicate a transaction's position in a time-line to contract code for the enforcement of contractual logic.
  - E.g. redemption of bond or late payment of debt.
- Time windows may be used for other purposes, such as regulatory reporting.

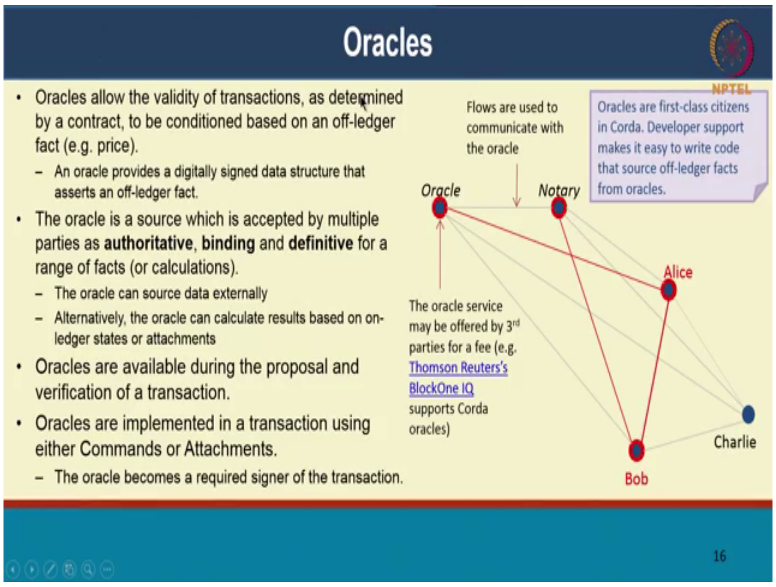
15

Notaries in addition can also be a time stamping authority and transactions can include a time window for to be committed. So, if the if the transaction comes to the notary too late, the notary can refuse to sign it saying, it is outside the time window and does not meet the criteria mentioned in the in the transaction.

Time windows can have a start and end time and or it can also be open ended. So, the purpose of time windows is to communicate transactions position in time line, corresponding to contract code. So, like I mentioned previously this is the almost the dag of states maintained by each node but if I do want to put a time stamp on it, then the notary service is the one that is going to put a time stamp on it.

And if you do want to have a time stamped ordering across different states that are present in the network, again the notary service can help with that. Again the notary service is also helpful in regulatory reporting. To say that, I perform this transaction on this date again that sort of a reporting needed for compliance is can also be provided by the notary service.

(Refer Slide Time: 22:27)



Oracles

- Oracles allow the validity of transactions, as determined by a contract, to be conditioned based on an off-ledger fact (e.g. price).
  - An oracle provides a digitally signed data structure that asserts an off-ledger fact.
- The oracle is a source which is accepted by multiple parties as **authoritative**, **binding** and **definitive** for a range of facts (or calculations).
  - The oracle can source data externally
  - Alternatively, the oracle can calculate results based on on-ledger states or attachments
- Oracles are available during the proposal and verification of a transaction.
- Oracles are implemented in a transaction using either Commands or Attachments.
  - The oracle becomes a required signer of the transaction.

Flows are used to communicate with the oracle

Oracles are first-class citizens in Corda. Developer support makes it easy to write code that source off-ledger facts from oracles.

The oracle service may be offered by 3rd parties for a fee (e.g. Thomson Reuters's BlockOne IQ supports Corda oracles)

Finally coming to oracles, which is also a first class principle in Corda. Oracles are serve as a authority to binding and definitive authority on facts that are external to the network. For instance, if a smart contract depends on the share price of a particular share right, so then, the there could be oracle responsible for providing the share price and all the peers trust the oracle to provide the same price information to all of them, regardless of who is requesting right, so the oracle will say ok.

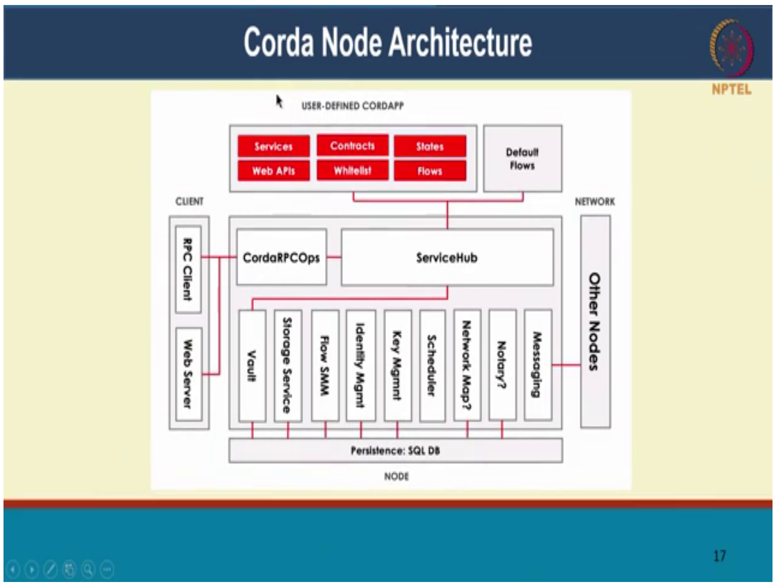
Anyone whose requesting the price in this time window, I will I will give them the same price value. So, that will ensure determinism of the contract and the contract itself can be access controlled to be within that time window.

Like I mentioned, you can specify that the contract has to be executed in this time window and the oracle can also be configured to provide the same share price to all peers within that time window. So, that way contract execution can be deterministic and contracts can also leverage facts from the outside world. So, the so oracles are of course trusted entities the peers are agreeing to trust the oracle for this service.

So, in financial services again the need for oracles, so it could be the need for oracles is seen a lot and I guess that is why Corda has incorporated oracles as a first class principle. There are oracles that have been created for instance in ethereum as a separate entity, not really part of the platform itself. But here in Corda oracles are a first class citizen and oracles can be included in the flow. For instance, in the flow itself Alice can contact the oracle get some information do some validation checks and check whether it wants to sign this transaction.

Oracles are implemented in a transaction using either commands or attachments. So, it can be an attachment to a transaction and there can be validation checks on that and in this case, when it when an oracle is present, signature from the oracle is needed as part of the transaction proposal.

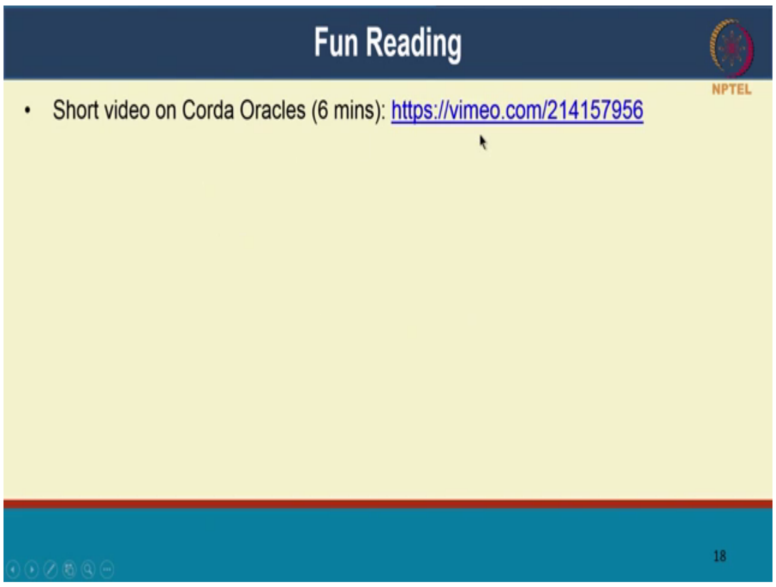
(Refer Slide Time: 24:46)


Corda Node Architecture

So, this is the overall reference architecture for Corda. It includes all the components of Corda. We talked about how the vault stores, the state information that is that each peer is aware of the storage itself, the transaction flows management, identity management, key management, the network map that tells you which peers are running on which IP addresses, how do I send messages to a particular bank; for instance, how should I address them, if they can refer to notaries, it can be part of a messaging it will have a messaging service that connects to other nodes and of course, all the communication with the Corda node happens through an RPC client and can also you also have a web server along with it.

The user defined Corda application includes all of this includes the services, includes the contract, the state information, the transaction flows, referring to these sates all of that is captured as part of the Corda application. And the Corda application can also refer to existing flows refer and utilize existing flows that are already defined as libraries by Corda. So, that is roughly the high level old view architecture picture of Corda.

(Refer Slide Time: 26:09)



Fun Reading

- Short video on Corda Oracles (6 mins): https://vimeo.com/214157956

18

So, with that we come to the conclusion of this lecture. Here is a short video on Corda Oracles right and why the why cord oracles are needed, how they function, so very short video that you can perhaps take a look at to get a sense of how oracles work in Corda. So, I hope that you have got a good overview of the Corda architecture.

We have looked at a bunch of these platforms now I hope you are appreciative of the fact that this is lot of innovation going on in different platforms. There are different design principles they are all starting with a different constructs that they are creating to a address the enterprise requirements of many of these blockchain applications.

So, some of these networks utilizing these platforms are just starting to come into production and it is really a very exciting time for enterprise blockchain applications to really make a big impact in all of our lives.

So, with that, thanks a lot. It was great having you with us in this course and I hope you enjoyed it as well, talk to you soon bye.