**R3 Corda**

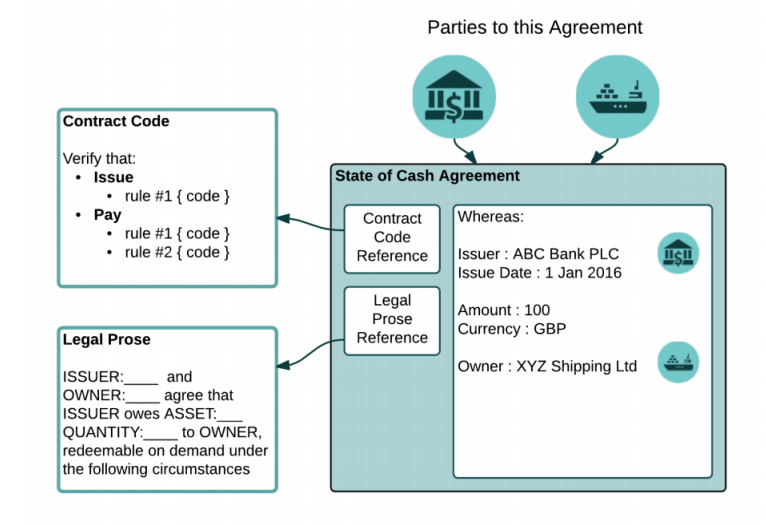
***Problem with financial institutions***

* Duplications, reconciliations, failed matches and breaks will be things of the past

***Core Concepts***

* No Blockchain ( No Blocks – It is all transactions – Distributed ledger )
* All the transactions are not distributed globally. Instead the transactions are sent only to the involved parties
* ***State Object [ Digital Document ]***
  + Content of the Data
  + Current State
  + Recording the existence

**Sample State Object**



* 2 Parties are involved in this contract 🡪 Commercial Bank and Shipping Company
* Contract – is smart contract 🡪 Hash of the reference only will exist in state object when shared with others
* Legal Prose 🡪 Hash of the reference only will exist in state object when shared with others

***Consensus***

**Transactions**

Transaction

State Object

New State Object

1. How ***Transactions are validated*** ?

* Smart contract is already pre compiled and deployed. If this runs successfully then it has a valid output state

Contract B

Contract A

State B

State A

If a transaction involves multiple contracts then the associates output states are also validated

1. ***Verification of Signatures*** ?

All the required signatures mandated by smart contracts are validated

1. ***Transaction uniqueness***

Let us says a Transaction happens between Party A and Party B.

The basic understanding is both Party A and Party B will execute the smart contract code and the logic will be validated. However, they don’t know if they both operated with the same state. ( May be Party B’s state is out of sync with Party A’s state) . In this scenario, it will lead to problem

Thus an Observer/Validator/Notary plays an important rule. *It checks whether the states consumed by a given transaction have not been previously been consumed*

Consensus is not at Ledger level ( Data is kept only on need to know basis [ON LEDGER] )

**OFF LEDGER**

Data held by only one actor is “off-ledger”.

**Corda has “pluggable” uniqueness services ( Validator/Notary nodes)**

This is not mandatory and is pluggable in nature.

**Example where it is not required**

In case , there are 4 parties and each state change requires signature from all the participants. In this case, notary doesn’t make any sense and it can be omitted

**Notaries info on Stack overflow**

<https://stackoverflow.com/questions/45816778/consensus-and-notaries/45817684#45817684>

Only the participants involved in a given transaction, plus the transaction's notary, verify a given transaction. This is core to Corda's privacy-first approach, where transaction information is only distributed on a need-to-know basis.

* Yes, each transaction is assigned to a specific notary service (there may be several on the network), and the notary ensures that there are no double-spends.
* That's correct. The non-verifying notary simply checks that the transaction's inputs haven't already been spent. The verifying notary also "walks the chain" to ensure that the transaction is valid. You have a choice between using a non-verifying and a verifying notary for each transaction, depending on how much information you wish to disclose.
* Both types of notaries store the information about which states have been spent locally. Only the notary itself can see the details of this map, by checking the NODE\_NOTARY\_COMMIT\_LOG in its H2 database.

**Points to remember**

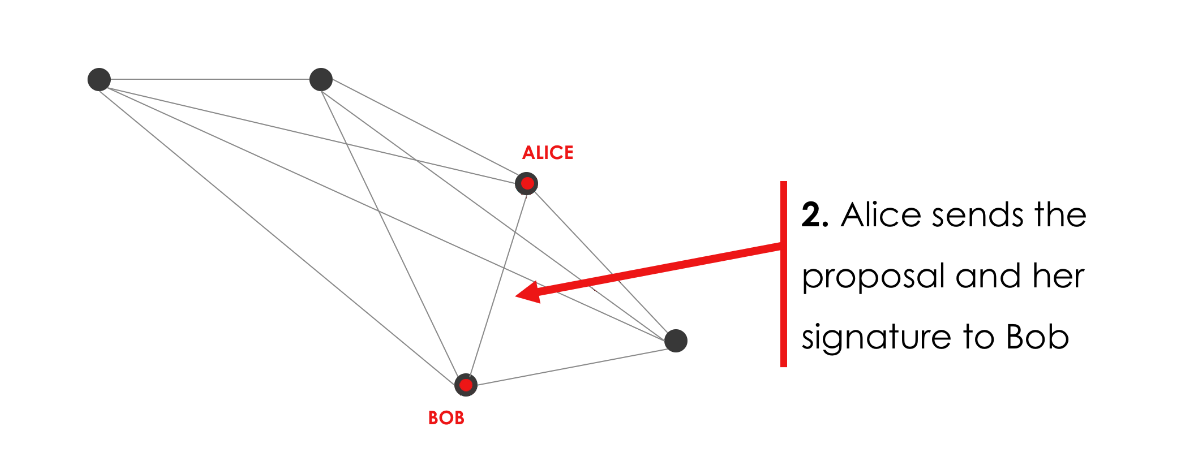
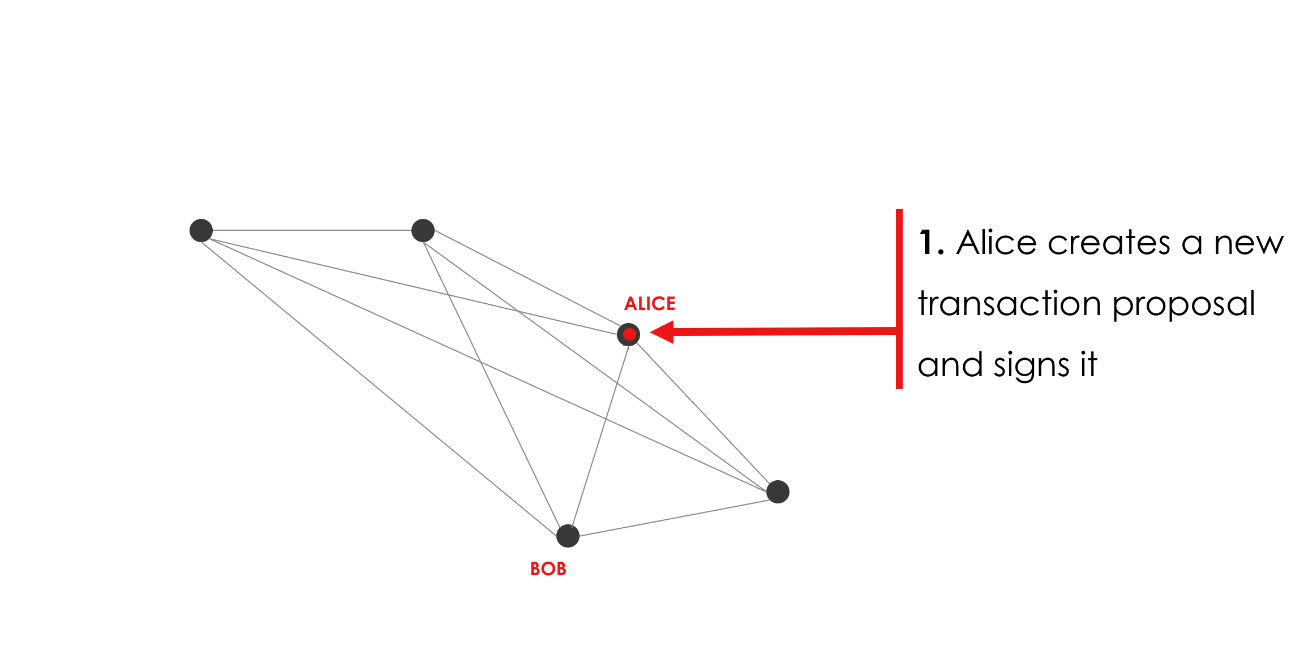
* UTXO Model
* Contracts do not have storage or the ability to interact with anything. Given the same transaction, a contract’s “verify” function always yields exactly the same result.
* Unlike Ethereum , smart contract code is not deployed across all participants. It is only shared with the needed parties. Also *smart contract , here is stateless 🡺 Just it gives a verifiable output*
* *Contract is executed inside a sandbox environment*

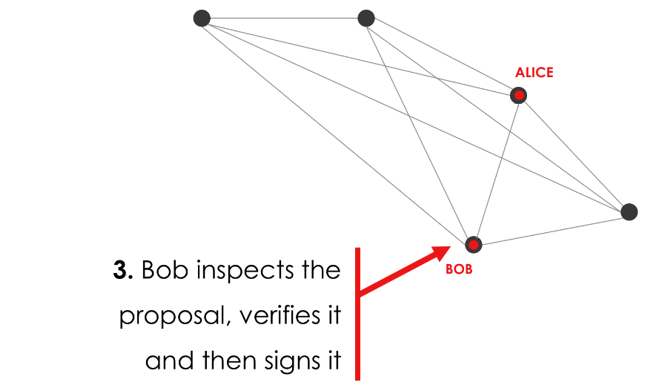
**Technical White Paper**

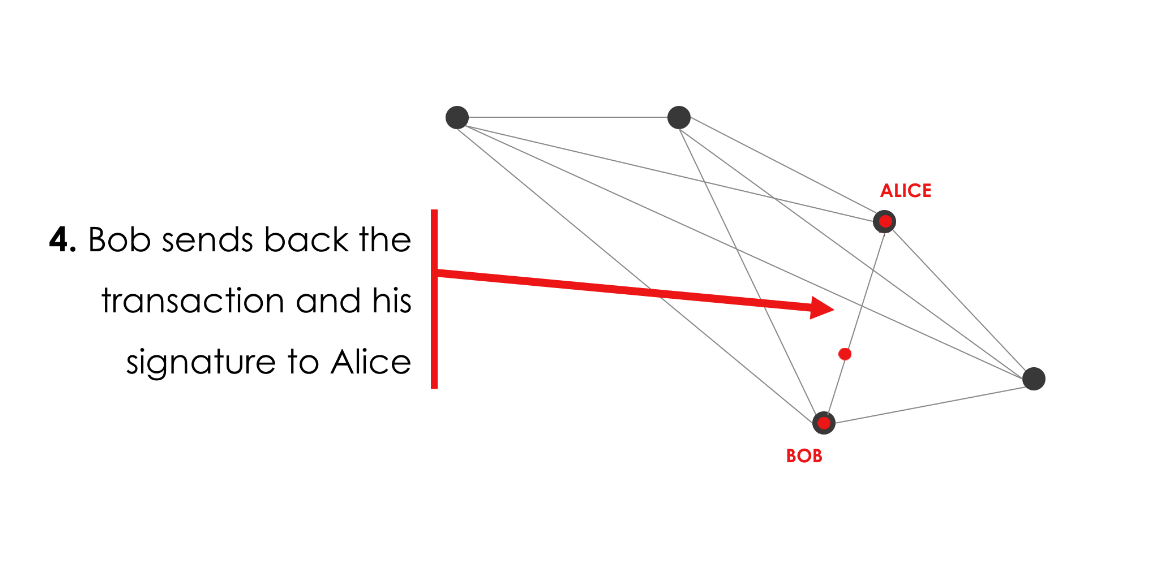
**Transaction Flows**

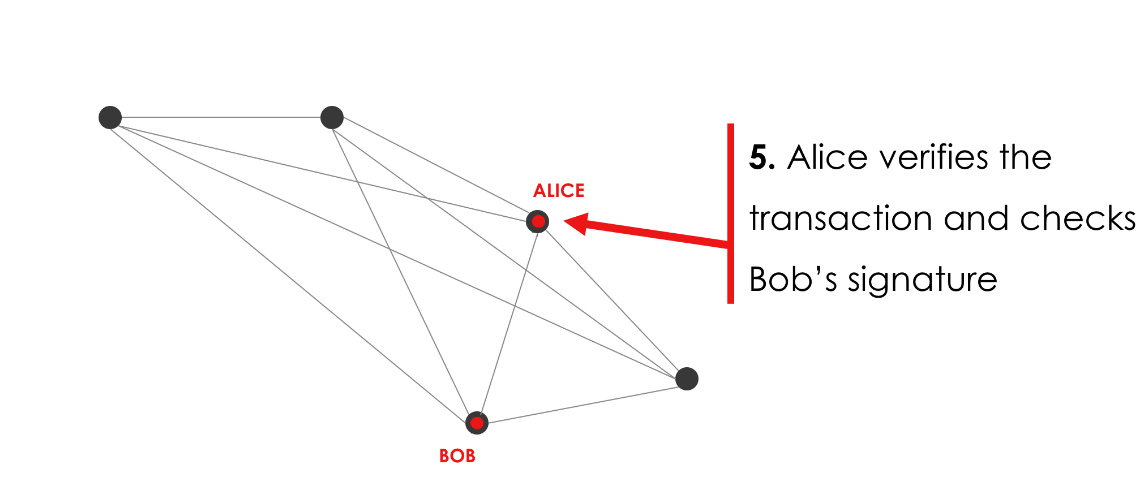
**Example :** Alice sends a transaction to Bob

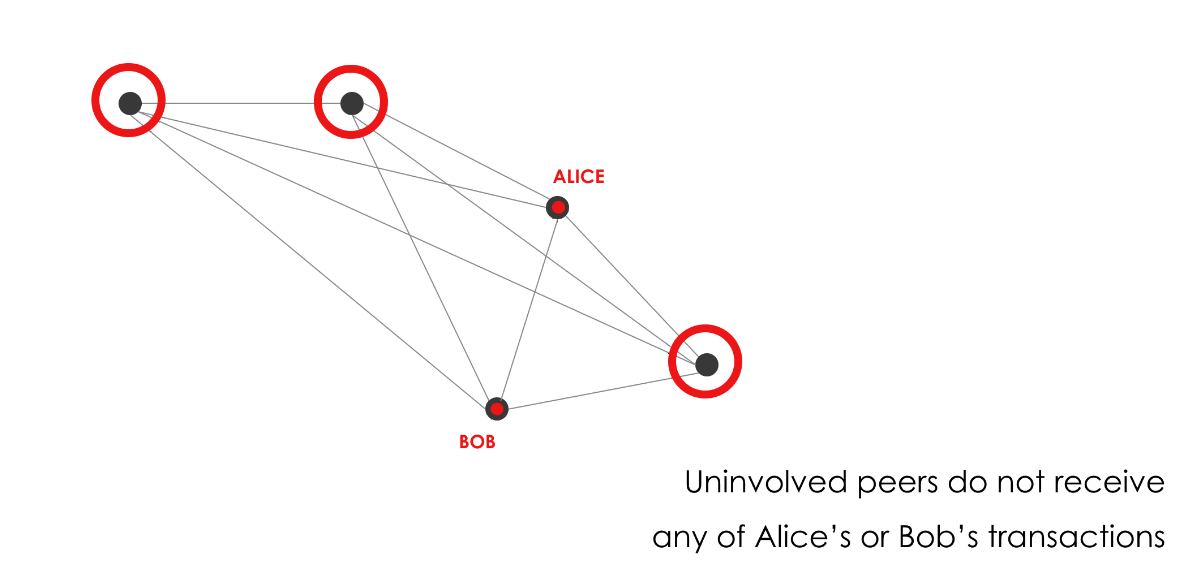
Let us look at it to see how this transaction flow is set up











**Corda’s rich features**

* Smart contract code is run in special JVM called [ Deterministic JVM Sandbox environment ]
* Data is shared on a need-to-know basis
* A single Corda network may contain multiple notaries that provide their guarantees using a variety of different algorithms
* Multi-step transaction building protocols called flows to be modelled as blocking code
* Nodes are backed by a relational database and data placed in the ledger can be queried using SQL as well as joined with private tables
* Support for Data types - dates, currencies, legal entities and financial entities such as cash, issuance, deals and so on
* Events on the ledger are exposed via an embedded JMS compatible message broker
* States can declare scheduled events. For example a bond state may declare an automatic transition to an “in default” state if it is not repaid in time

**Corda – No ordering of transactions /No Proof of work**

UTXO Model

* Notaries in the network ensures the transaction is not double spend ( i.e the outputs are not consumed)

**Peer to peer network**

Oracle service

Notaries ….

Peer 2

Peer1

Message Queue services to communicate

**Corda Identity Management**

* Global names like Email can be used (or) specific Public keys (PKI infrastructure) can also be used

**Flow framework**

This can be retrieved when a transaction is received or when node restarts

State objects are serialized and written in DB

State Machine

Creates a state machine

Quasar

Flow code is written in Java (Blocking calls ONLY)

* Flows are created based on identities . Internally by looking at the identity it will route to the right peer
* Flows can interact with other sub flows
* Flow progress can be tracked
* Flow when stopped – Can be recovered from Flow Hospitals

**Transaction resolving**

When a transaction is sent, the necessary input ( breath first transversal) is fetched from the tree and send as part of the transaction flow. ( This is required to prove the transaction is valid and you have the necessary inputs)

Problems with this processing

* When the tree is deep, processing is slow

**Overcome using**

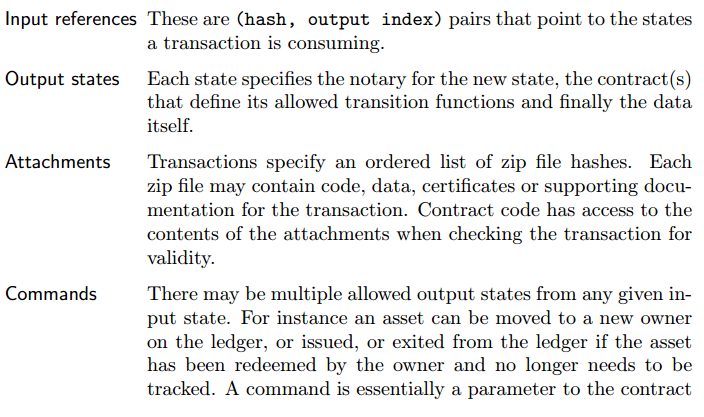
* Small-subgraph transactions
* Transaction privacy techniques ( Future plans – hardware based/ZK Snarks)
* State re-issuance

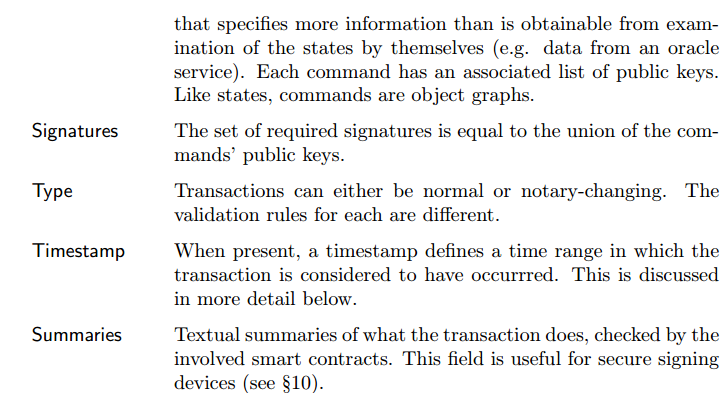
**Transaction Data model**

**UTXO Model – Same like Bitcoin but Corda has extra fields**

States are the atomic unit of information in Corda. They are never altered: they are either current (‘unspent’) or consumed (‘spent’) and hence no longer valid. Transactions consume zero or more states (inputs) and create zero or more new states (outputs). Because states cannot exist outside of the transactions that created them, any state whether consumed or not can be identified by the identifier of the creating transaction and the index of the state in the outputs list.

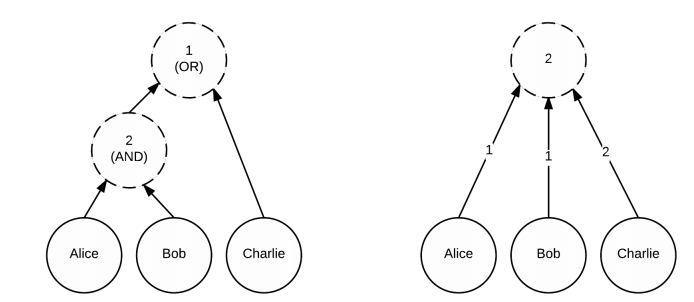
**Transaction Parameters**





**Composite Keys**

Public keys can be used AND/OR condition 🡪 Similar to HF Endorsement conditions



**Attachments**

The required documents, files etc are hashed and placed under attachment sections . The attachment section will not be validated until and unless the logic is built around it

**Example:**

* Expecting a file which is of certain hash or expecting a file with a particular signed user etc

**Attachments and contract bytecodes**

Transaction data is sent separately and attachments are sent separately

**Dispute Resolution**

As Corda contracts are simply zip files, it is ***easy to include a PDF or other documents describing what a contract is meant to actually do(Legal Prose)***. There is no requirement to use this mechanism, and there is no requirement that these documents have any legal weight. However in financial use cases it’s expected that they would be legal contracts that take precedence over the software implementations in case of disagreement.

**Roll backing Transaction ( In case of faulty transaction – no hard forks in Corda )**

* **Challenge** – All the information is not across all nodes. It is difficult to understand which nodes have got bad information

**Tool Which can investigate the request ( bad transaction)**

Corda defines a flow to assist with this, which can be used by anyone. A tool is provided that generates an “investigation request” and sends it to a seed node.

**Process**

The flow signals to the node administrator that a decision is required, and sufficient information is transmitted to the node to try and convince the administrator to take part (e.g. a signed court order). If the administrator accepts the request through the node explorer interface, the next hops in the transaction chain are returned. In this way the tool can semi-automatically crawl the network to find all parties that would be affected by a proposed rollback.

**Minimal Support**

The platform does not take a position on what types of transaction rollback are justified and provides only minimal support for implementing rollbacks beyond locating the parties that would have to agree.

**Problem fixing between parties**

**Case 1** - Once involved parties are identified there are at least two strategies for editing the ledger.

**Solution**

One is to extend the transaction chain with new transactions that simply correct the database to match the intended reality. For this to be possible the smart contract must have been written to allow arbitrary changes outside its normal business logic when a sufficient threshold of signatures is present. This strategy is simple and makes the most sense when the number of parties involved in a state is small and parties have no incentive to leave bad information in the ledger

**Case 2** - Result of theft or fraud ( Party involved would be eager to correct before it reaches all)

**Solution**

In this case a more complex approach can be used in which the involved parties minus the uncooperative party agree to mark the relevant states as no longer consumed/spent. This is essentially a limited form of database rollback.

**Identity Look Ups**

**Not able to understand**

. When a state is deserialised from a transaction in its raw form, the identity field of the Party object is null and only the public (composite) key is present. If a transaction is deserialised in conjunction with X.509 certificate chains linking the transient public keys to long term identity keys the identity field is set. In this way a single data representation can be used for both the anonymised case, such as when validating dependencies of a transaction, and the identified case, such as when trading directly with a counterparty. Trading flows incorporate sub-flows to transmit certificates for the keys used, which are then stored in the local database. However the transaction resolution flow does not transmit such data, keeping the transactions in the chain of custody pseudonymous.

**ORACLES AND TEAR-OFFS**

Oracle service

Notaries ….

Peer 2

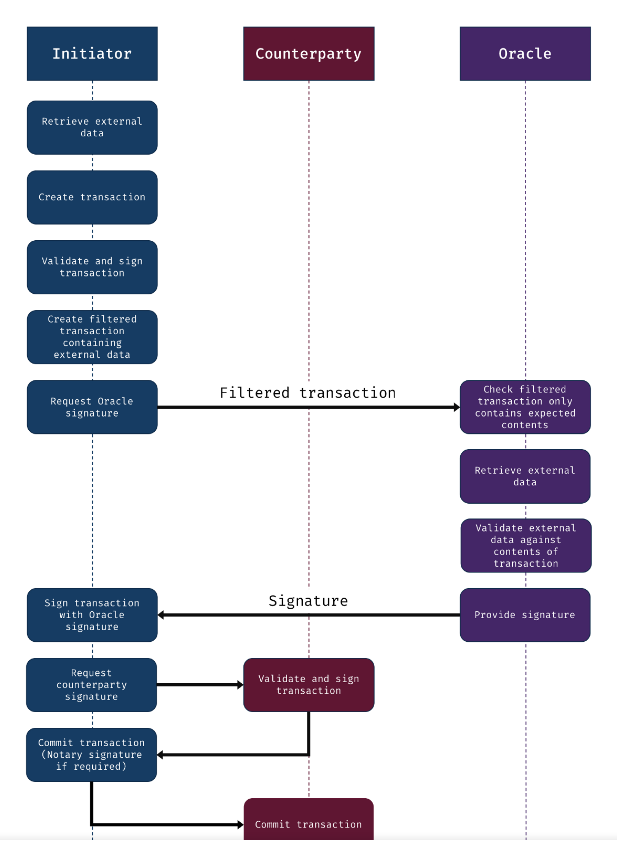
Peer1

Message Queue services to communicate

Good article on Oracles

<https://medium.com/corda/validating-external-data-with-an-oracle-d8a6a53d148e>

**Sequence Diagram of Oracle Flows**



**Contract Constraints**

State is tied to up contract code. This can be done via ,multiple ways

1. Contract code Hash is tied to the state
2. Contract code is added to ZIP archive JAR and it signature is validated
3. Using composite key model and validating the signatures

**Event Scheduling**

* Transaction flow can be scheduled

Ex: Options contract has to expire on certain date etc.

***API to use***

SchedulableState()

nextScheduledActivity()

**UTXO design in Corda**

* Notary will check if the already spent output is again sent ( double spending check)
* Also the reason why Spent XO is maintained in Notary is purely for privacy purpose. They should not know the unspent output because this is going to happen in future and also notary may be common for multiple peers and revealing its unspent output will be a breach

Maintains Peer1 UTXO Maintains Peer2 UTXO Notary maintains only Spent XO

Notary

Peer2

Peer1

**How Common Financial Constructs are addressed in Corda**

**Assets & Obligations**

* Asset modelling classes are pre built – like Owner, Quantity etc can be mentioned
* Obligation modelling classes also exists ( typically used wherever hard cash has to be substituted with – like IOU)

**Stock Buy and Sell using Corda as a Platform**

Consider a stock exchange. A buy order can be submitted along with a partially signed transaction that signs a cash input state and a output state representing some quantity of the stock owned by the buyer. By itself this transaction is invalid, as the cash does not appear in the outputs list and there is no input for the stock. A sell order can be combined with a mirror-image partially signed transaction that has a stock state as the input and a cash state as the output. When the two orders cross on the order book, the exchange itself can take the two partially signed transactions and merge them together, creating a valid transaction that it then notarises and distributes to both buyer and seller. In this way trading and settlement become atomic, with the ownership of assets on the ledger being synchronised with the view of market participants. Note that in this design the distributed ledger itself is not a marketplace, and does not handle distribution or matching of orders. Rather, it focuses on management of the pre- and post- trade lifecycles

**Consensus Algorithms**

Raft/Smart-BFT/SGX(future) or implement your own consensus algorithm

**Notaries**

**Validating notaries Non Validating Notaries**

* Only checks the input states are already consumed
* It can only see the input states and time-windows
* Entire Transaction is validated

**Core Developer Documentation**

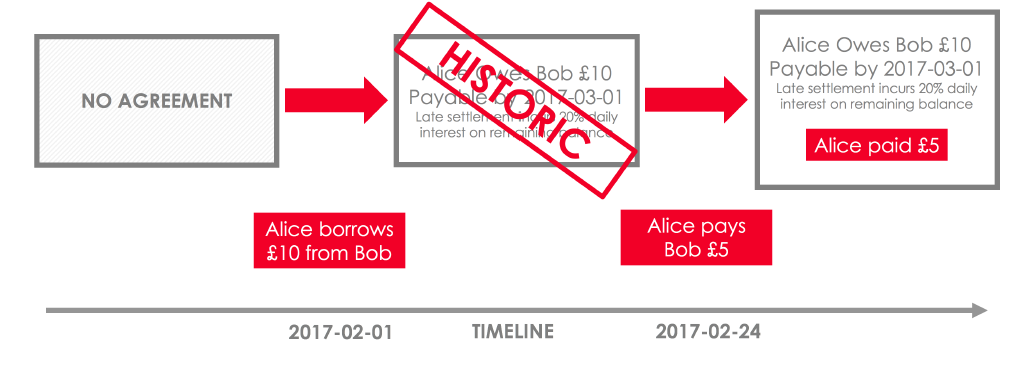
**How one can Participate ?**

* TO be a participating node, party has to obtain a certificate from the network operator ( This certificate can tie to the real world identity or a public key)

**State**

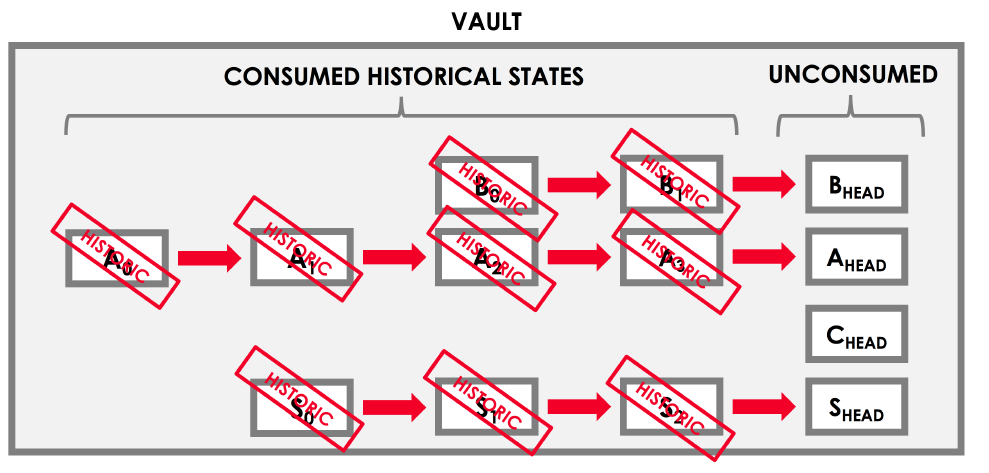
* State is immutable . Once the state id updated it becomes historic and the current state is referred

**Example :** Once Alice Pays 5 pounds, the latest state changes and the previous state becomes historic



**State updates by Nodes ( Each state is maintained in a database)**

NODE1



* A0 , A1 etc are historic states – Latest state is Ahead ( Similarly it applicable to other states as well)

**Reference States**

* This is similar to normal state . However in reference state, it has reference to the data where each node need not update/consume

**Example :** A Cash state may refer to the Premium or Deposit state – Cash state will decide the balance based on deposit state . But only the cash state will be consumed and not the deposit state

**Note**

• The specified notary for the transaction does check whether the reference states are current. However, reference states are not consumed when the transaction containing them is committed to the ledger.

• The contracts for reference states are not executed for the transaction containing them.

**Transactions**

* These are the ones which changes one state to another
* UTXO Model ( Inputs, Ouputs, Reference type)
  + Issue transaction – will have zero input
  + Exit transaction – will have zero output
* There can be n number of states like CASH, BOND etc.

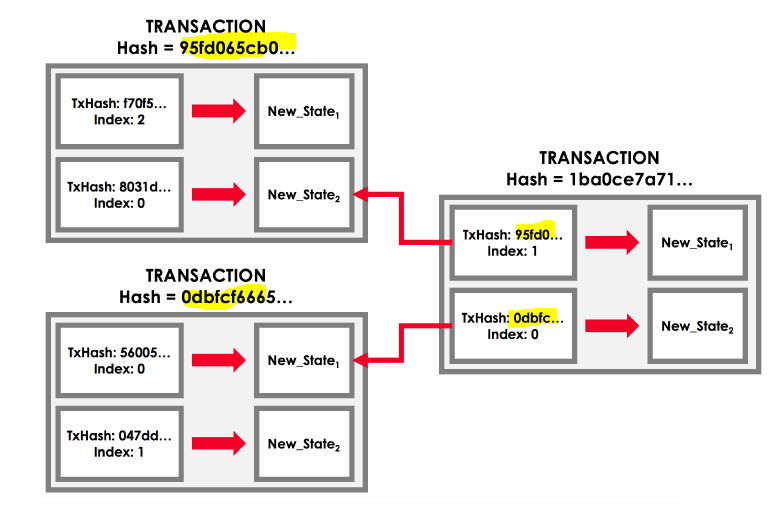
**Two types of transaction**

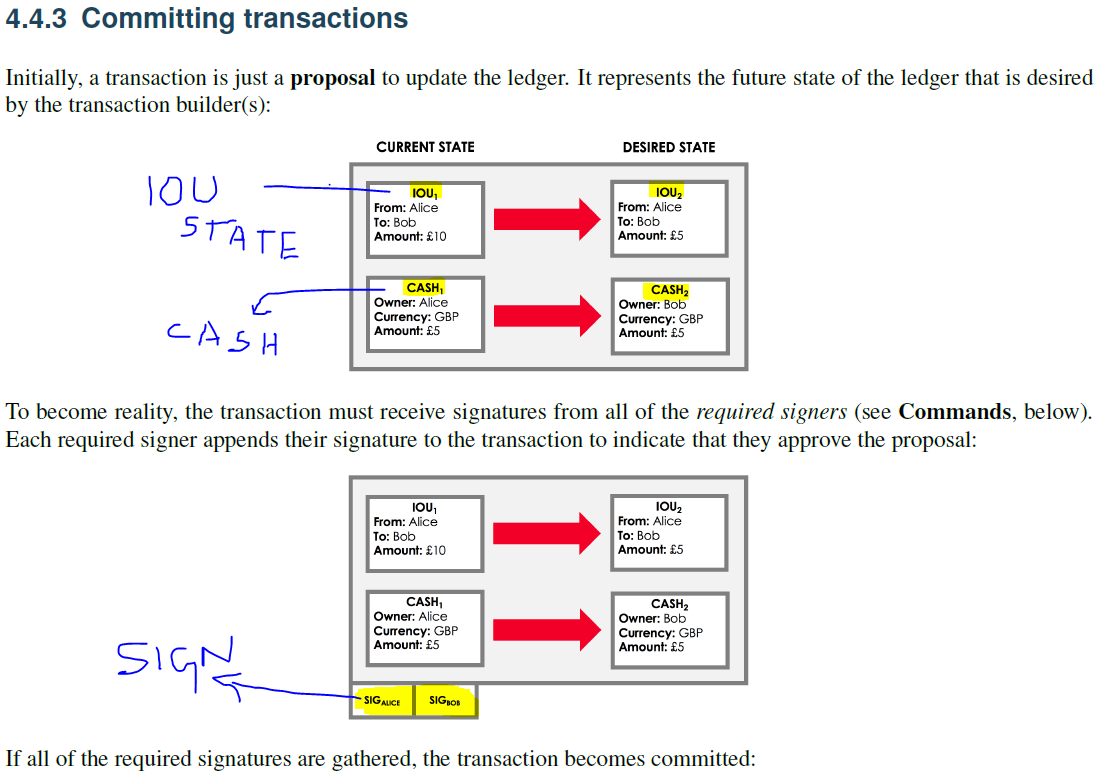
* Notary based transaction
* General transaction

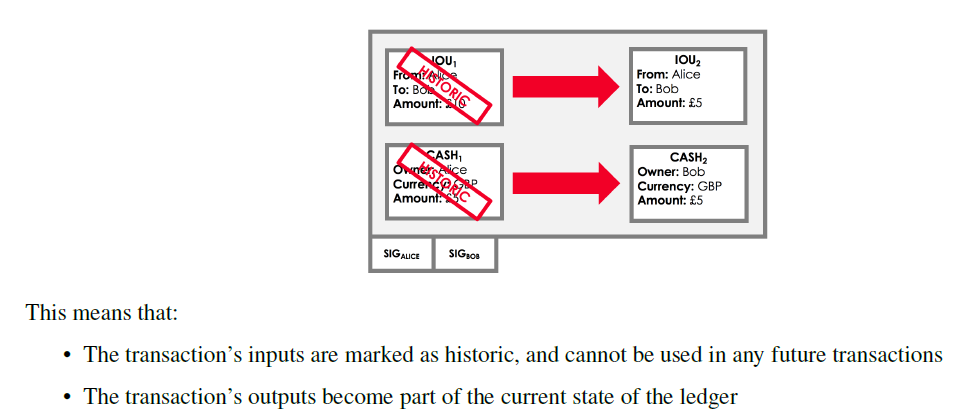
< Difference to be found out >

**Transaction Chain**

While changing the state , transaction always refers to the previous inputs ( UTXO Model) . This is similar to Bitcoin model







**How a transaction is validated**

1. Should have valid signatures
2. Smart Contract validity check
3. Double spend check ( Inputs should have consumed in previous transactions

* 1 and 2 🡪 Called Transaction Validity
* 3 – Called Transaction Uniqueness

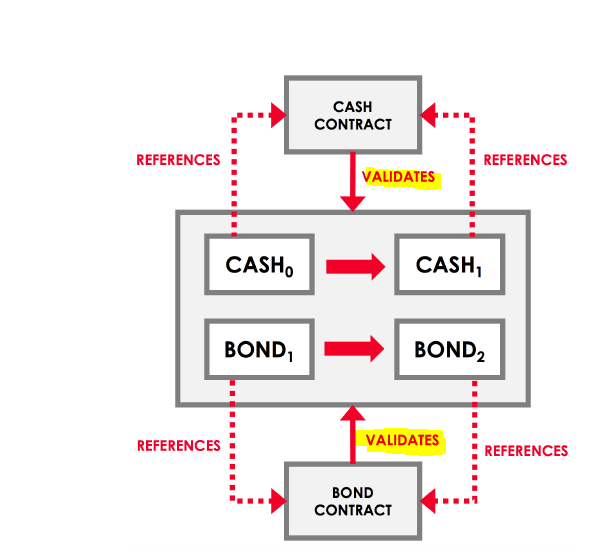
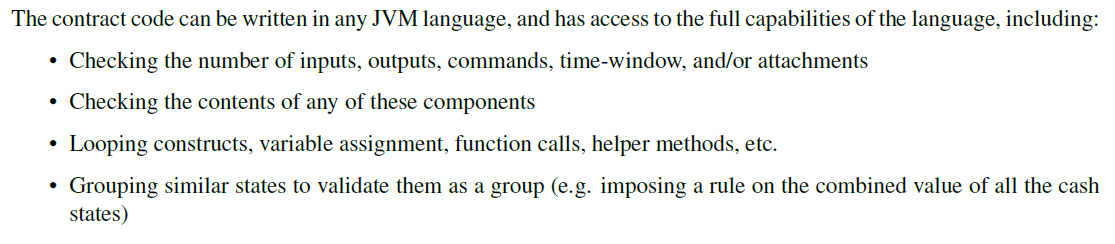
**Other Transaction Components**

* Commands – PAYMENT, SETTLE, CASH etc. – Many types of commands can be used depending on the use case
* Attachments – All the legal prose, public holidays, contract code are zipped and hash of these documents goes as part of transaction
* Time window – Certain transactions can be time bound. Ex: Execute bond transaction only between Monday through Friday ( 9 – 3 PM )
* Notaries – One who validates the consumed state (double spend check) . When there are no inputs, typical notaries are not required. 🡪 FINALITY is done by notary ( one who commits to the ledger)

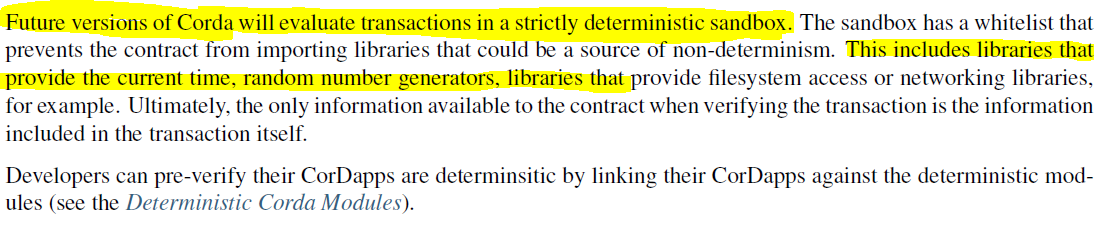
**Smart Contracts**

Refer “How a transaction is validated? Step 2 is smart contract check”

* A valid transaction must be accepted by the contract of each of its input and output states
* Contracts are written in a JVM programming language (e.g. Java or Kotlin)
* Smart contract code is deterministic in its evaluation. (i.e.) it either returns true or false

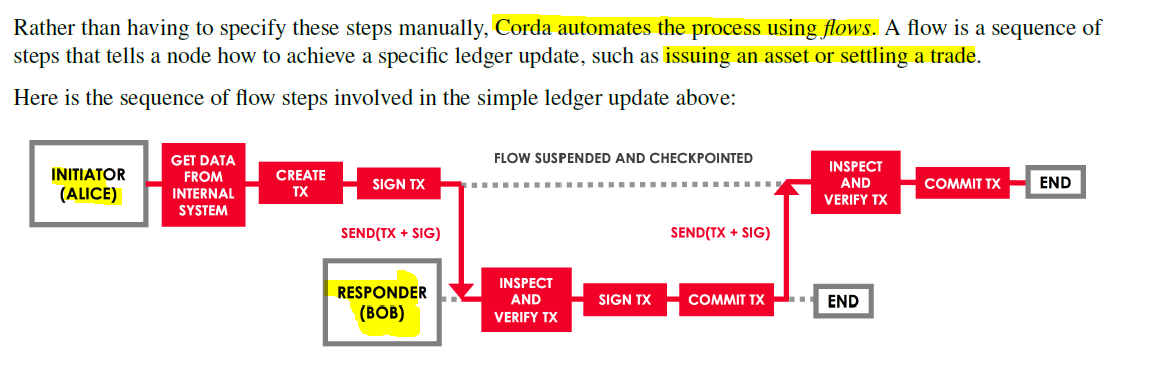
 

**Corda Sandbox**

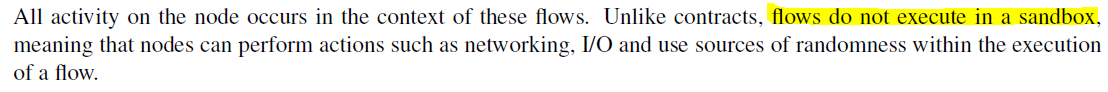


**Flows**

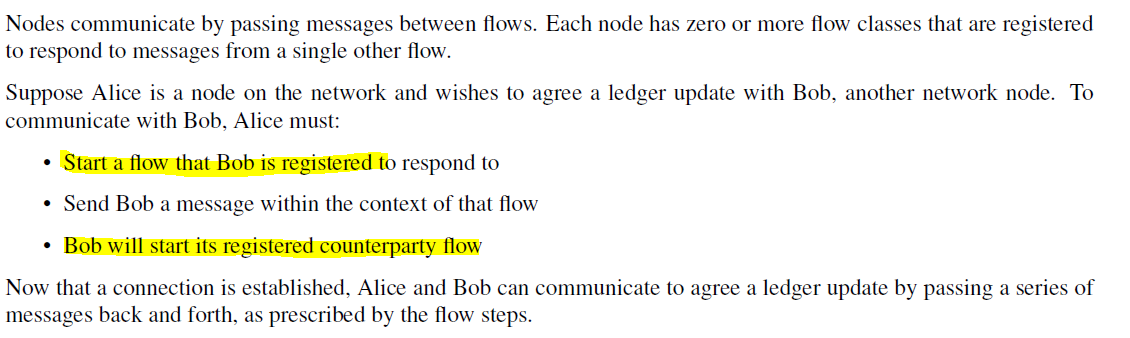
Flows are the crux of Corda. These flows are invoked by RPC calls. It will know



**Note:**



**How to different parties agree to a flow ? how is it set up**

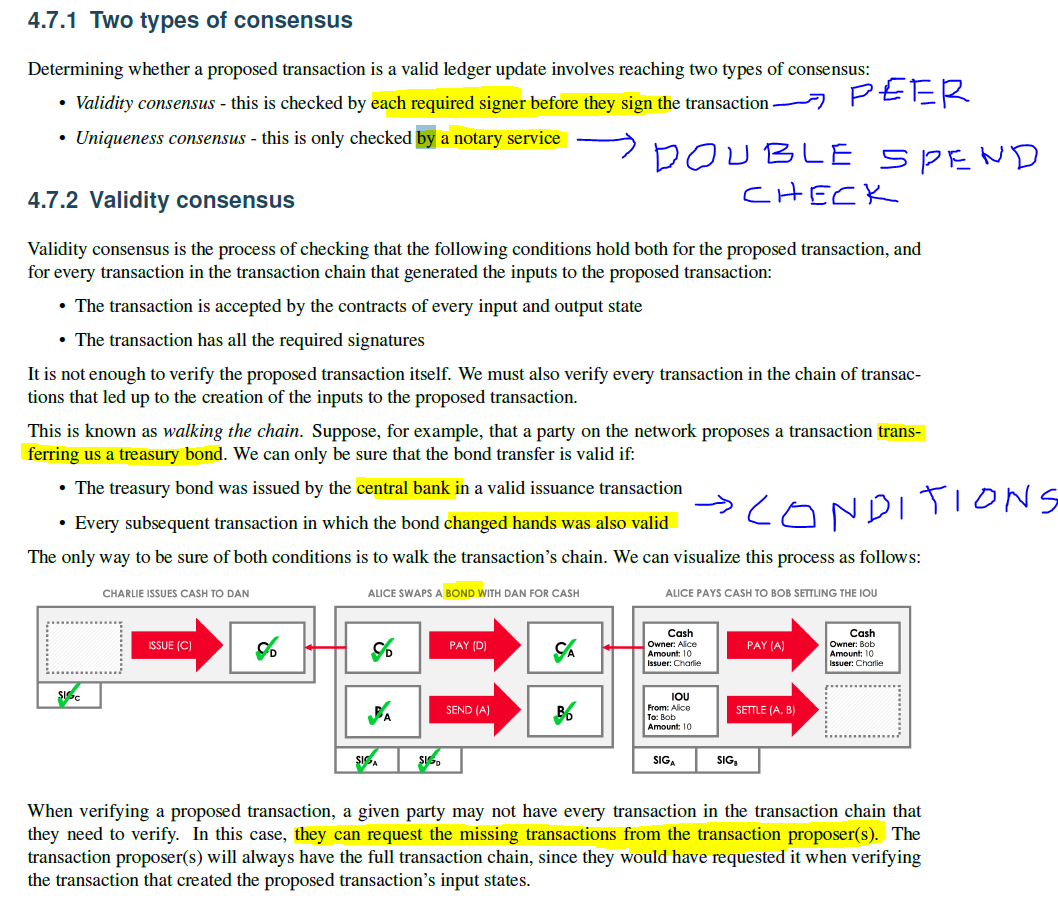


**Sub Flows**

One can create many sub flows ( Ex: Like how in OOZIE one job have to wait until the other one completes)

* Multiple in built flows are already defined ( Common ones) . Depending on the use case, one can create his own flow
* Flows can run parallel executed, can be stored and executed later point of time. It is checkpoint – available even after node is restarted

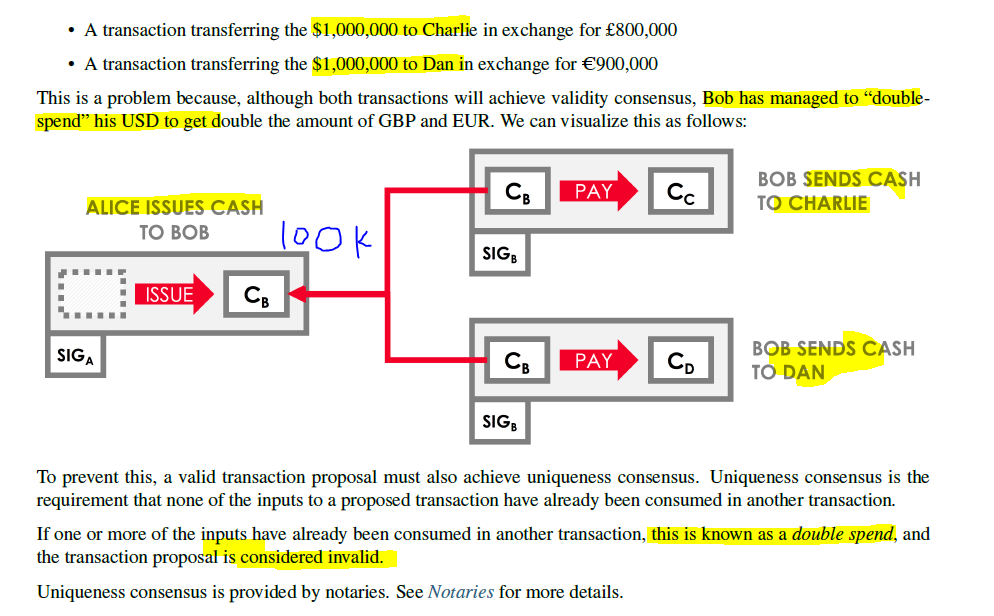
**Consensus**



*The above is checked in the node end . After this step, it will be passed to notary*

*Notary Side – Uniqueness Consensus*

**Example** : Bob has cash state issued by Central Bank which is worth of 100K



*Validating Notaries*

* Notaries normally check for double spends
* Validating notaries, in addition to double spends also checks for transaction validity ( which is typically done the participating nodes)

*One Problem with Validating Notary*

* *Every node has to share its input state and dependent transaction information*

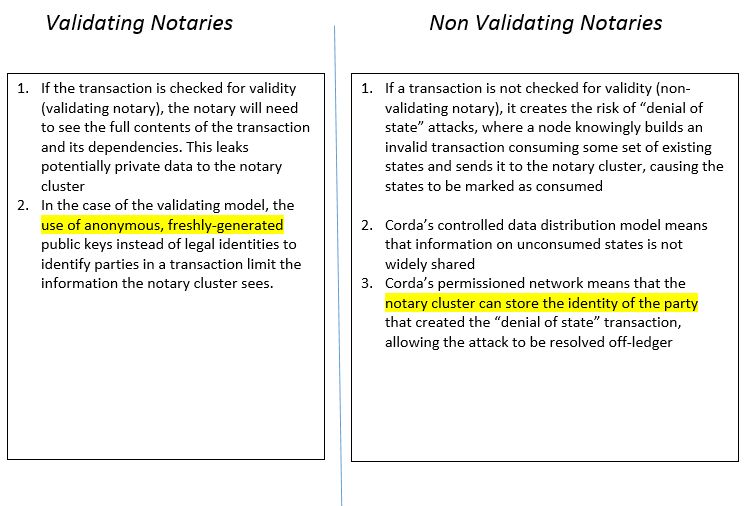
*Important*

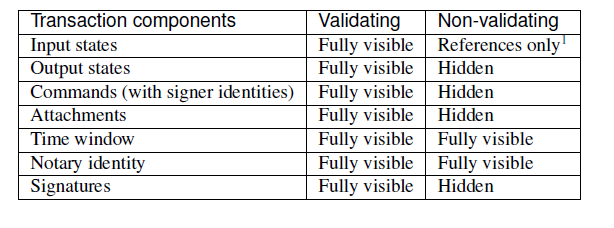
*Typically Notary Service will be a trusted Node. In case of non-trusted scenario, there can be Notary Clusters. The clusters can run an algorithm like RAFT, BFT*

*After the notary cluster’s signature is obtained, we can be sure that the proposed transaction’s input states have not already been consumed by a prior transaction. Hence, notarization is the* ***point of finality in the system***

*Notary Cluster – Requirement*

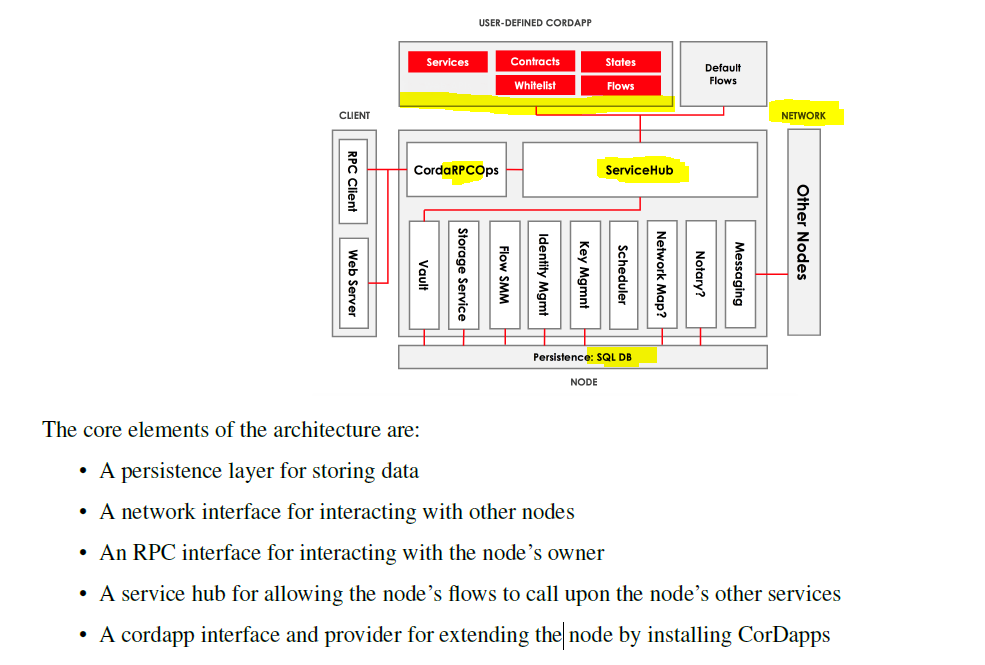
* All the nodes should shares its input states so that the Cluster can validate the consumed states





* *Notary cluster can be changed*

*Nodes in Corda*

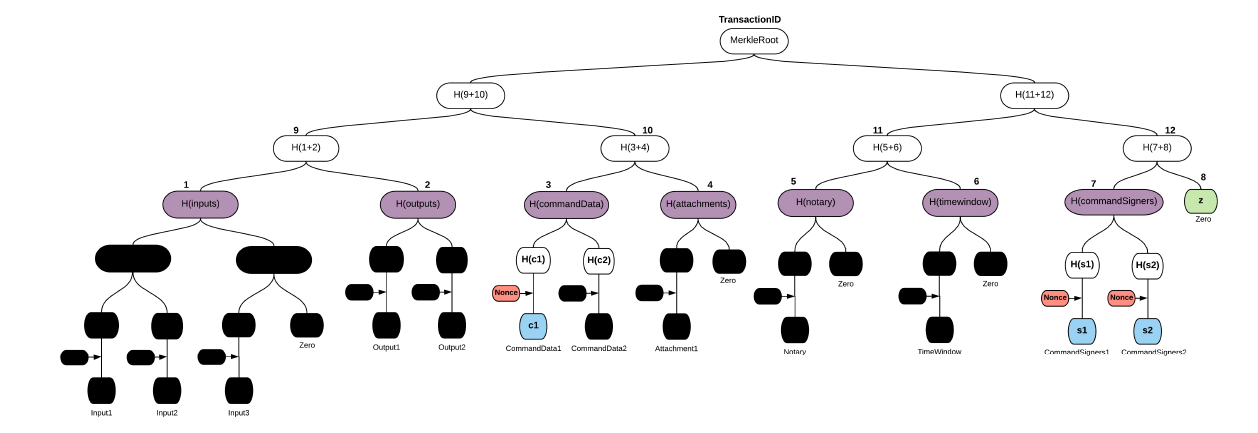


*Transaction Tear Off*

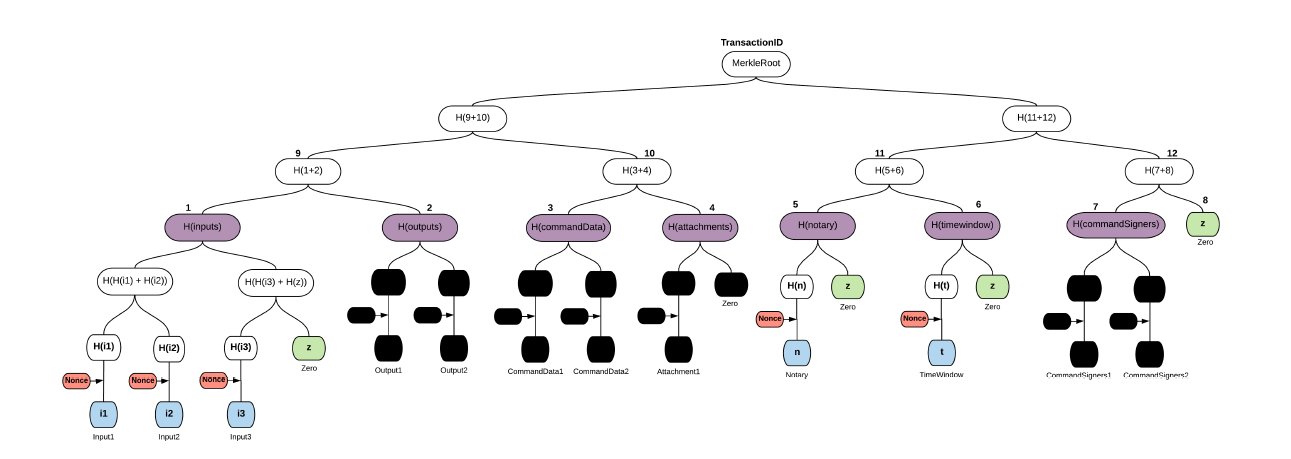
* Hiding the data when the transaction is sent across

**Oracles Scenario**

* Only command1 and the respective signatures are visible to Oracle nodes . ( Remaining data is hashed)



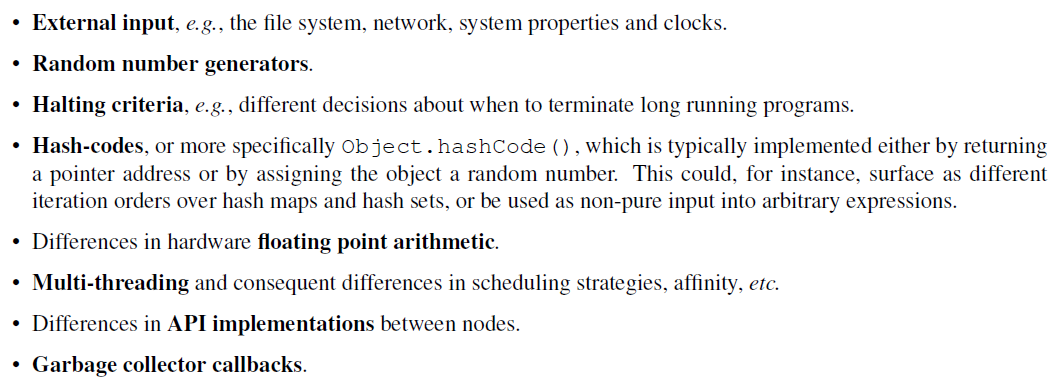
**Validating Notaries – Only Inputs are shown rest are hidden**



**Deterministic JVM – (For the defined inputs, the outputs are always same)**

Currently , non JVM is used but developers are asked to get used to Deterministic JVM

The following are some of the underlying problems where JVM is non deterministic



**Deterministic JVM –** The idea is to run the smart contract code in specific sandbox