* States
* Contracts
* Transactions
* Flows
* Corda Nodes
* Corda Networks

**Networks**

1. peer-to-peer network 🡪  **no global broadcasts** ( TLS-encrypted messages sent over AMQP/1.0)
2. each node is a JVM run-time environment hosting Corda services and executing applications known as CorDapps
3. Each node has got an identity . Identity can also be anonymous

**States**

* States are stored in vault
* States are always associated with a contract
* States are immutable. Moment it is transferred , it means the state is spent and the ownership is with some other party ( This is state sequence )
* Reference State 🡪 Received a Loan amount of 10000 $ . This amount is not going to change , this will be a reference state for calculating EMI

**Transactions**

* Transaction is a proposal
  + A transaction can contain any number of inputs, outputs and references of any type
  + Issuance of Bond 🡪 Zero inputs but an output exists
  + Exit of Bond 🡪 Input consumed but zero output
  + Reference State 🡪 Loan amount reference etc.

**Types of Transactions**

1. **General Transactions**
2. **Transaction which updates Notary State**

While every state can have a different notary, all of the states consumed by a particular transaction must be assigned to the same one. This avoids issues relating to deadlocks and synchronization, which should be familiar for those with distributed database experience. Let’s say Alice and Bob agree to exchange Alice’s $10 for Bob’s £7. The transaction for this exchange must be signed by the notaries of both states, but which one goes first? If Alice’s notary signs but Bob’s fails for some reason, then Alice will be left with an incomplete transaction and can never use her $10 again. If Bob’s signs first then he is similarly exposed. While we might like notaries to simply work together, in practice this requires mutual trust and the use of a consensus protocol, complications which Corda’s designers chose to avoid.

If states with different notaries are required as inputs to a single transaction, their owners first execute special “notary change” transactions, which move a state from one notary to another, changing nothing else.

**Transaction Chain**

**Example : Bond is in Issued State – 100**

**STEP 1 (Initial Transaction) : No inputs but output exists**

**LTI BONDS**

Input Output

NIL 100

Transaction Hash 🡪 0safdasdfsasd + Index 0 ( which has 100 )

Notary knows

Party A

100

**STEP 2 (Transaction Proposal) : 10 Bonds Transfer to Party B**

**INPUT**

**Reference the previous Transaction**

0safdasdfsasd + Index 0 🡺 creates a new output 10 for Party B

* Notary verifies whether 10 input exists by Party A

Once verified, Notary knows

Party A

90

Party B

10

**How a transaction is validated by Party B ?**

* Signatures from Party A
* Contract validity – deterministic execution/result
* Walking the chain – Chain of transactions got to be valid

Notary has to approve the state in not consumed yet

Transaction can also contain

* Reference state 🡪 Referring the state for the purpose of computation. Notary will confirm if the give state is the current one
* Command 🡪 ISSUE, TRANSFER, EXIT
* Attachment 🡪 ZIP/JAR containing Legal document, exchange codes
* Time Window 🡪 Choosing a particular time to execute a transaction
* Notary 🡪 Choose a notary

**Contracts**

* Deterministic JVM – Result is either Boolean
* Signatures validity is not sufficient. Contractual validity is required
* State is always associated with Contract

**Flows**

* Flows will be stored as part of Cordapp
* It can be serialized and stored in disk . Can be initiated anytime (even after restart)
* There are many predefined library flows. Custom flows can also be written

**API Documentation**

**Demerits with Corda**

**Scalability Problem**

Having said that, there is one situation in which Corda performs far worse than a blockchain. This occurs when a node receives a new transaction which depends on many other transactions it has not seen before. Imagine a highly liquid asset that was issued 10 years ago, and changes hands about every five minutes. The path from any new transaction back to this asset’s issuance will be over a million transactions long. When a node receives this asset for the first time, it must retrieve these million transactions from the sender and verify each one in turn. At a (fairly optimistic) rate of 1000 transactions per second, there would be a 17 minute delay before the recipient could send the asset on – clearly too long for something so liquid.

**Confidentiality Problem**

 if Alice pays Bob $10, then Bob sends that $10 on to Charlie, Charlie’s node has to be shown the transaction between Alice and Bob, even though it doesn’t involve him. At the time that Alice paid Bob, she had no way of knowing who might see this transaction in future, and anyone might be sent it at any time.

Corda’s only unique advantage in terms of confidentiality remains its reduced transaction visibility – an incomplete solution at best.

Proposed Solution : using SGX enclaves