Corda is a Distributed Ledger Technology to be used by businesses, such as financial institutions, to keep a shared ledger of transactions and thus removing the need for the involved parties to constantly check that each of their books are in line after interacting with each other. This is the primary problem that Corda is trying to solve.

It also removes the need for all parties on the Corda network to know about each and every transaction.

Software requirements

* Oracle JDK 8 JVM - minimum supported version 8u171
* IntelliJ IDEA - supported versions 2017.x and 2018.x (with Kotlin plugin version 1.2.51)
* Git

NOTE -:

* Corda runs in a JVM. JVM implementations other than Oracle JDK 8 are not actively supported.
* Applications on Corda (CorDapps) can be written in any language targeting the JVM. However, Corda itself and most of the samples are written in Kotlin. Kotlin is an [official Android language](https://developer.android.com/kotlin/index.html).

Key concepts

*The network*

* A Corda network is made up of nodes running Corda and CorDapps
* The network is permissioned, with access controlled by a doorman
* Communication between nodes is point-to-point, instead of relying on global broadcasts.
* All communication between nodes is direct, with TLS-encrypted messages sent over AMQP/1.0.
* Each network has a **network map service** that publishes the IP addresses through which every node on the network can be reached, along with the identity certificates of those nodes and the services they provide.
* Each network has a doorman service that enforces rules regarding the information that nodes must provide and the know-your-customer processes that they must complete before being admitted to the network.
* To join the network, a node must contact the doorman and provide the required information. If the doorman is satisfied, the node will receive a root-authority-signed TLS certificate from the network’s permissioning service.

*Network services*

Nodes can provide several types of services:

1. One or more Notary services –

* May be run on a single node, or across a cluster of nodes.
* It guarantees the uniqueness and the validity of ledger updates.

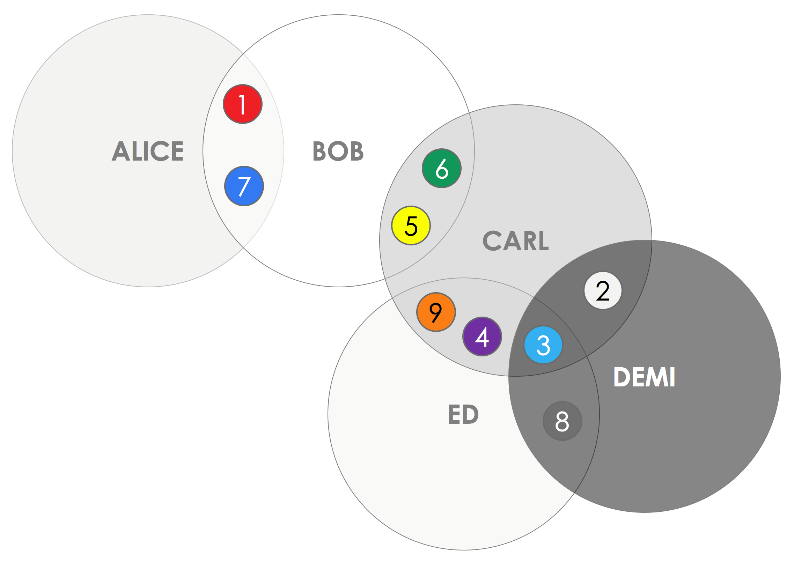
1. Zero or more **oracle services –**

* It signs transactions

*The ledger*

Each node maintains a separate database of known facts. As a result, each peer only sees a subset of facts on the ledger, and no peer is aware of the ledger in its entirety.

Imagine a network with five nodes, where each colored circle represents a shared fact:



*Identity*

Identities are attested to by X.509 certificate signed by the Doorman or a well-known identity.

Identities in Corda can represent:

* Legal identity of an organization - are used for parties in a transaction, such as the owner of a cash state.
* Service identity of a network service -  are used for those providing transaction-related services such as notary or oracle.

Identity Types -:

1. Well known identity - This certificate is published in the network map service for anyone to access.
2. Confidential identities - are only published to those who are involved in transactions with the identity.

Confidential identities are used to ensure that even if a third party gets access to an unencrypted transaction, they cannot identify the participants without additional information.

Are transactions encrypted in Corda?

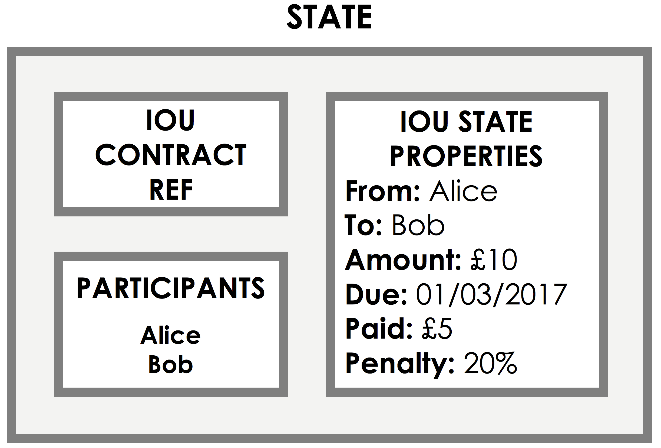
*Certificates*

Not clear

<https://docs.corda.net/key-concepts-identity.html#certificates>

*States*

* States represent on-ledger (Shared) facts
* Each node has a vault where it stores any relevant states to itself
* States are immutable and cannot change
* For shared facts to evolve we must create a new state by copying the current state and make updates as required.
* Previous state must be marked as historic which will remain accessible and provide audit trial.
* States are never deleted.
* States are statically typed – a bond state is always a bond state.
* The vault tracks the head of each state sequence.

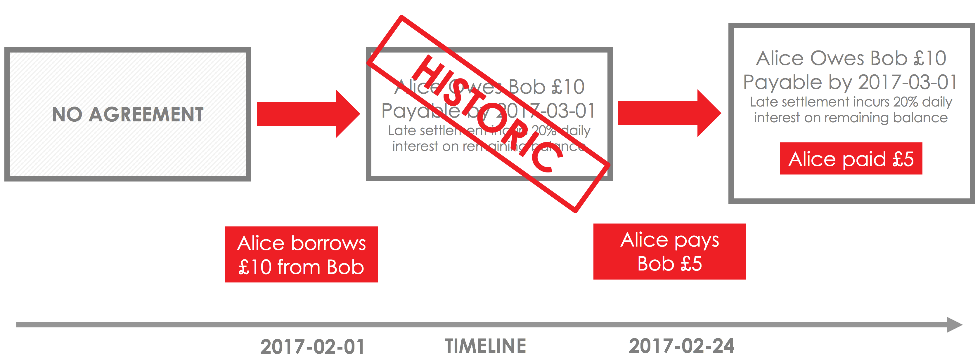


State object contains

1. Properties – reflect the state of an agreement or contract at a specific point in time.
2. Participants – lists the peer who can consume this state in a transaction.
3. Contract ref – points to a contract which defines the verification function.

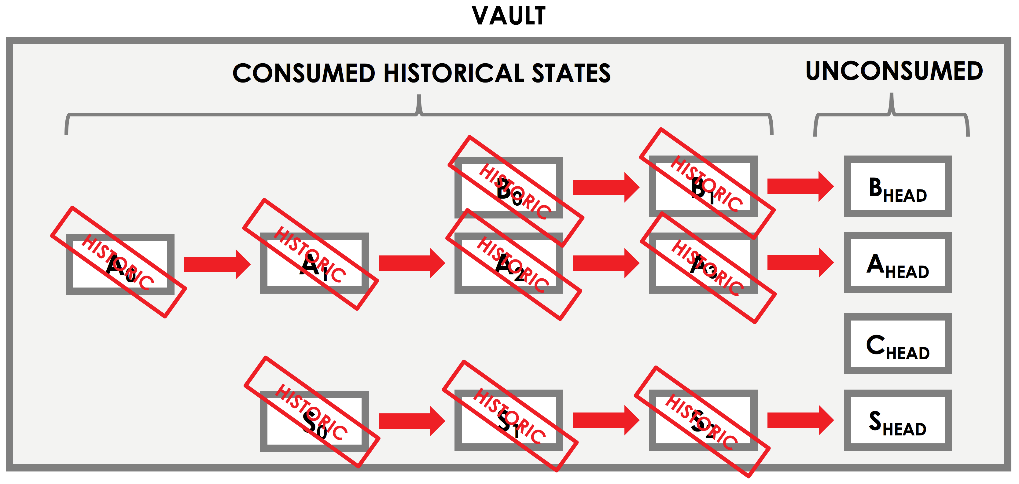
State Sequences

The lifecycle of a shared fact over time is represented by a **state sequence**.



The vault

Each node on the network maintains a *vault* - a database where it tracks all the current and historic states that it is aware of.



Contracts

* Contracts are written in a JVM programming language (e.g. Java or Kotlin)
* The verification function is defined in the contract code.
* The function takes a transaction as a parameter and either throws an exception if the transaction fails verification or return nothing if the transaction verifies.
* To verify a transaction, the contract code must be executed by all required peers on a need to know basis.
* Corda must guarantee that the contract code produces the same output for all peers each time it is executed.

Transaction verification

A transaction is only valid if it is digitally signed by all required signers. However, even if a transaction gathers all the required signatures, it is only valid if it is also **contractually valid**.

**Contract validity** is defined as follows:

* Each state points to a contract
* A contract takes a transaction as input, and states whether the transaction is considered valid based on the contract’s rules
* A transaction is only valid if the contract of every input state and every output state considers it to be valid
* A transaction that is not contractually valid is not a valid proposal to update the ledger, and thus can never be committed to the ledger.
* As transactions may contain multiple state types, multiple contracts can be referenced in a transaction.

The contract sandbox

* The contract code cannot contain any non-determinism e.g. random number generators, uncontrolled referencing of data external to ledger which may change.
* Corda executes the contract code within a “sandboxed” environment that guarantees determinism.
* The “sandbox” is a modified version of JVM.

Contract limitations

* Since a contract has no access to information from the outside world, it can only check the transaction for internal validity.
* It cannot check, for example, that the transaction is in accordance with what was originally agreed with the counterparties.
* Peers should therefore check the contents of a transaction before signing it, *even if the transaction is contractually valid*, to see whether they agree with the proposed ledger update.
* A peer is under no obligation to sign a transaction just because it is contractually valid. For example, they may be unwilling to take on a loan that is too large, or may disagree on the amount of cash offered for an asset.

Oracles

Sometimes, transaction validity will depend on some external piece of information, such as an exchange rate. In these cases, an oracle is required.

* An oracle is a service that will only sign the transaction if the included fact is true
* It creates digitally signed data structures which assert facts.
* An oracle is a source of data or calculations which has been accepted by multiple peers for an agreed set of values or range of calculations.

Oracle implementation

Using commands -:

* The fact is incorporated into a command inside the transaction itself.
* The oracles public key is added to the commands signers list – the oracle becomes a co-signer to the transaction.

