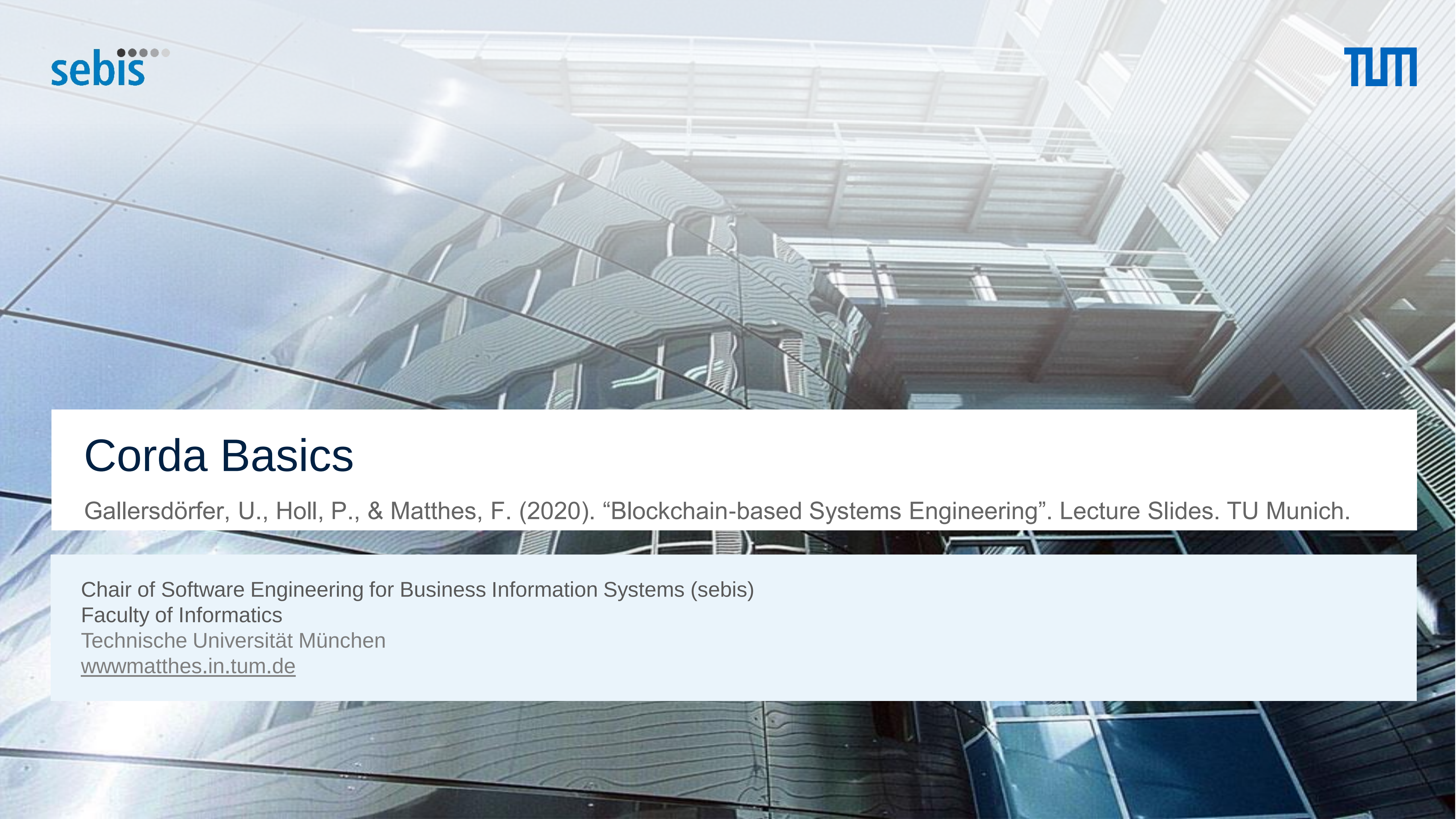


Corda Basics

Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

Chair of Software Engineering for Business Information Systems (sebis)
Faculty of Informatics
Technische Universität München
wwwmatthes.in.tum.de

Outline

1. Introduction to Corda
    1. History
    2. Definition and mission
    3. Vision and principles
    4. The Corda Foundation
    5. Novel features
2. System Architecture
    1. Network
    2. Ledger
    3. Vaults
    4. States
    5. Transactions
    6. Contracts
    7. Flows / Flow Framework
    8. Consensus
    9. Notaries
    10. The Global Corda Network
3. CorDapps
    1. Concept: Concepts and components
    2. Cordapp example: Token minting
    3. Cordapps: Real world CorDapps
    4. Conclusion

Introduction to Corda: History

Richard G. Brown, CTO

David E. Rutter, Founder

- Corda was developed by **R3**, an enterprise blockchain software firm founded in 2014, in collaboration with over **200 technology and industry partners**, e.g. AWS, Intel, Microsoft, Deutsche Bank, etc. Together they form the R3 Alliance.
- The **first GitHub commit** was in November **2015**.
- Corda was **first introduced** in April **2016** and in November, Corda goes open source, and the **white paper** was released.
- **Between 2017 and 2019**, Corda has gone through **4 big releases** with 1800+ commits.
- Initially, Corda was developed to service the financial industry, however, it has evolved to **support business DLT applications in general** at the scale level of markets and industries**.**

History of Corda (https://www.corda.net/history/)
Industry partners (https://marketplace.r3.com/directory)
Mike Hearn, and Richard G. Brown, Corda: A distributed ledger (https://docs.corda.net/_static/corda-technical-whitepaper.pdf)

Introduction to Corda: Definition and mission

c•rda

Corda is an **open source project** with the aim of developing an **enterprise-grade distributed ledger platform** for business across a variety of industries. Similar to Fabric, Corda is used to set up the infrastructure for a **permissioned P2P network**. This network holds a set of **parameters**, which are encoded in a data structure, signed by the **network operator,** and distributed via the network infrastructure**.** The **network nodes** use these common parameters for communication and other main activities.

Corda aims to build a platform for businesses to **transact freely** with any counter-party while retaining **strict privacy** (need-to-know basis). While it **does not support a cryptocurrency** like Bitcoin, it also allows **issuance of digital assets and tokens** in the broader Corda or particular business networks. Digital assets allow for **incentive mechanisms** for adoption and participation, and **payment** of services.

Corda is developed in **Kotlin** and specifically designed for **enterprise use cases**. Corda provides an implementation of this vision in a code base which others are free to build on, contribute to or innovate around. The system supports "**smart contracts"** that need to be written in Java or any other **language that compiles to JVM bytecode**.

Mike Hearn, and Richard G. Brown, Corda: A distributed ledger (https://docs.corda.net/_static/corda-technical-whitepaper.pdf)

Introduction to Corda: Vision and principles

The Corda vision is one where multiple applications, products and services are deployed to an **openly governed** common shared network, where assets gained in one context from one trading partner for one service can be immediately redeployed without friction or transfer costs for another purpose to pay a different trading partner utilising a different application.

The interaction between entities in the network will be done through **Corda Applications** (CorDapps).

Information and assets can be used in different CorDapps, in opposition to other isolated enterprise-focus blockchain deployments.

Business Principles	Definition
Assured Identity	The identity of participants is verified.
Inclusion	Participants may discover and transact with each other freely.
Privacy	Access control on transaction details.
Shared Logic	The code underlaying the contracts is shared to ensure consistency.
Legal Footing	The code underlaying the contracts is shared to ensure consistency.
Authoritative	Facts recorded are authoritative.
Immutability	Records are immutable.
Open	Open source, development, participation, governance, and standards.

Architectural Principles	Definition
Scalability	Billions of transactions daily.
Longevity	Different version interoperability.
Security	Operation under assumption of an adversarial security environment.
Stability	Operation under assumption of an adversarial security environment
Interoperability	Multiple applications can interoperate thanks to standard interfaces, and to the use of industry-standard tools.

Mike Hearn, and Richard G. Brown, Corda: A distributed ledger (https://docs.corda.net/_static/corda-technical-whitepaper.pdf)

The Corda Foundation

A commercial entity shall not control Corda Network. On the contrary, the governing entity shall be a **not-for-profit** entity. The **Cora Foundation** takes this role.

The Foundation entities shall include:

1. A **Governing Board** of 11 directors with privileges and responsibilities.
2. A **Technical Advisory Committee**, comprised of representatives of Participant organisations.
3. A **Governance Advisory Committee**, comprised of representatives of Participant organisations.
4. A **Network Operator**, charging the Foundation reasonable costs for providing network and administration services, paid by the Foundation through membership funds, and accountable directly to the Board.
5. These entities operate on behalf of the **Participant Community. Participants** have a general membership, which is open to any organisation subject to meeting normal Corda Network access conditions.

The Foundation shall meet costs by levying an annual **participation fee** (operational costs of the Network Operator, and the Foundation) and a **transaction notary fee** (per-use basis) for all Participants.

Corda Foundation Governance Guidelines (https://corda.network/governance/governance-guidelines/)

Introduction to Corda: Novel Features

Corda is a distributed ledger platform but it is technically **not a blockchain-based system**. Corda is designed to be used in an isolated environment and therefore comes with several features that significantly differ from public Blockchain systems like Ethereum or Bitcoin:

- Entities behind nodes are **legally identifiable**, unlike the pseudonyms used with public keys in other DLT platforms. Each node undergoes a know-your-customer (KYC) process to obtain an identity certificate from the **corda global network members**.
- Nodes are arranged in a **peer to peer permissioned network**.
- All **communication is direct**; there is no gossip protocol.
- **Transactions may execute in parallel** (allows horizontal scaling), on different nodes, without either node being aware of the other's transactions (need-to-know basis).
- There is **no global broadcast or central ledger** that is shared by all participants. This results in **private transactions**.

Gendal Brown, R..: The Corda Platform: An Introduction (https://docs.corda.net/_static/corda-introductory-whitepaper.pdf)

Introduction to Corda: Novel Features

Here are some other unique features:

- There is **no blockchain**. Instead, Corda uses clusters of distrusting nodes called *notaries,* which use pluggable consensus algorithms among them to prevent double-spending, e.g. BFT.
- There is **no global consensus algorithm or mining** as defined in Ethereum or Bitcoin, the notaries maintain different ledgers independently.
- The ledger is defined as a set of immutable state objects instead of blocks. **States are an atomic** unit of data and are consumed and created following an **UTXO model**.
- **Each transaction must be signed by a notary** cluster to be included in the ledger. However notaries do not necessarily see the content of the transactions, they may only see the index of the state being consumed and the unique hash of the transaction updating the ledger.
- No “real” smart contracts. In Corda the term **smart contract** is used for a **pure function that verifies a transaction.** These contracts can be seen as a list of rules the transaction needs to fulfil to interact with the application.

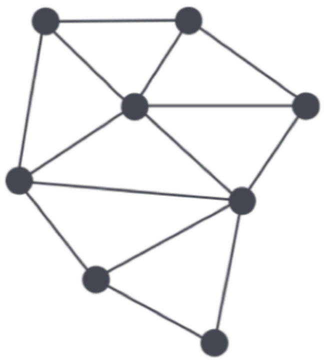
Gendal Brown, R..: The Corda Platform: An Introduction (https://docs.corda.net/_static/corda-introductory-whitepaper.pdf)

Outline

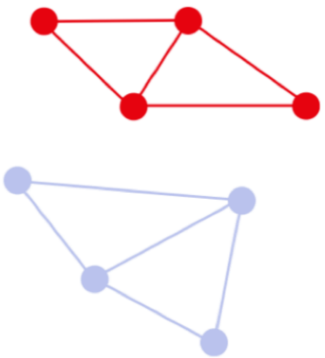
1. Introduction to Corda
    1. History
    2. Definition and mission
    3. Vision and principles
    4. The Corda Foundation
    5. Novel features
2. System Architecture
    1. Network
    2. Ledger
    3. Vaults
    4. States
    5. Transactions
    6. Contracts
    7. Flows / Flow Framework
    8. Consensus
    9. Notaries
    10. The Global Corda Network
3. CorDapps
    1. Concept: Concepts and components
    2. Cordapp example: Token minting
    3. Cordapps: Real world CorDapps
    4. Conclusion

Network – Vision: Seamless Interoperability

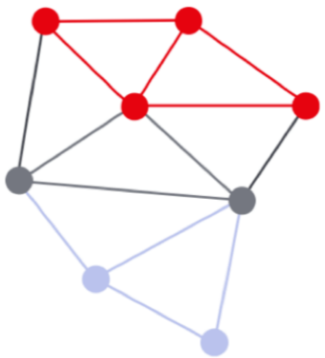
Corda aims a seamless interoperability through its “Global Corda network”: Assets gained from a particular service should be redeployed immediately without friction to pay a different trading partner utilizing a different application.



Public Blockchains



1st gen. Permissioned Blockchains



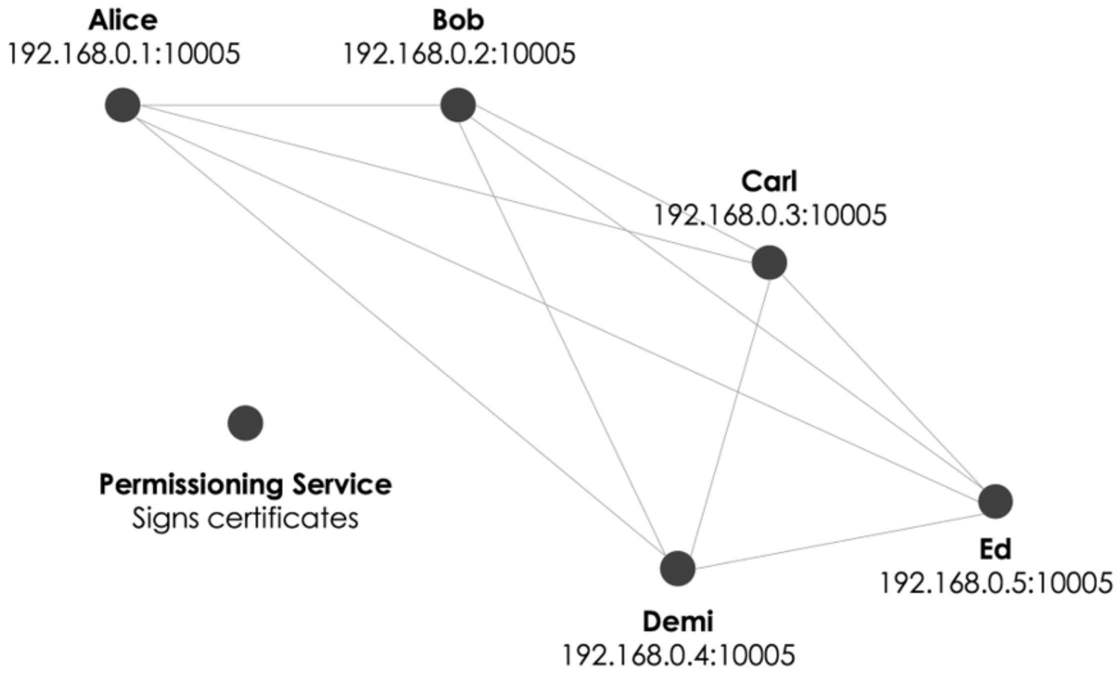
Corda Platform

1st gen. permissioned Blockchains: Interoperability between different Business Networks not possible.

Corda Platform:
Standardized set of interfaces for contracts are included to maximize interoperability from a diverse range of providers.

Source: The Corda Platform - https://www.corda.net/content/corda-platform-whitepaper.pdf

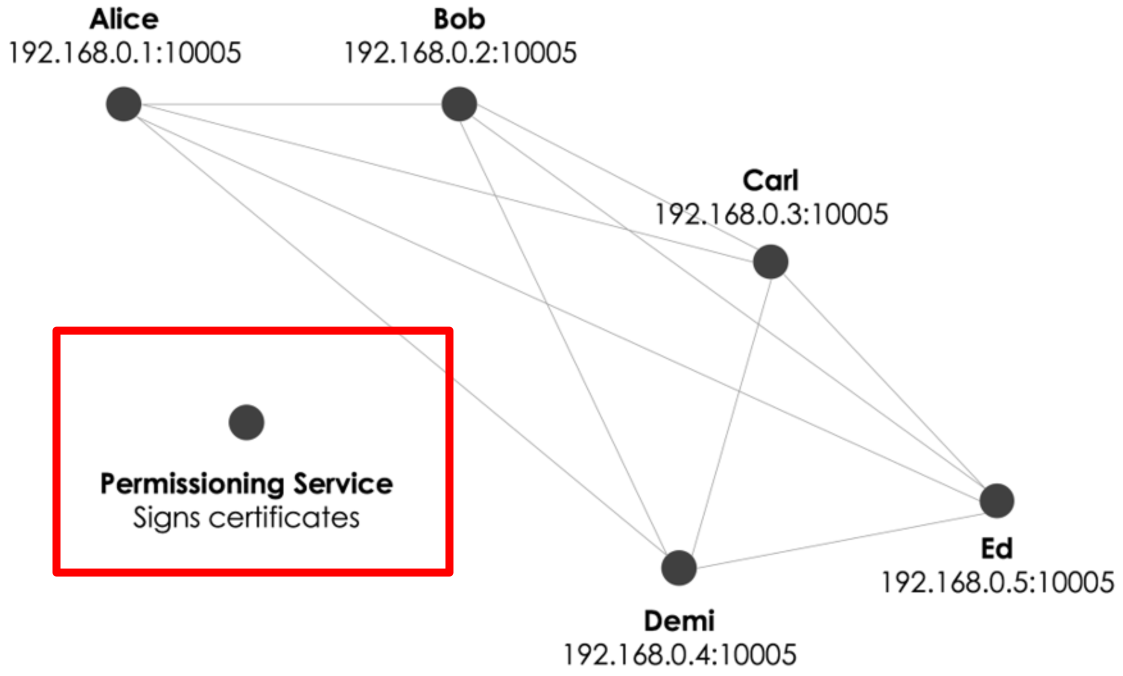
Network: Structure, Identities and Discovery



- **Peer-to-peer** network of **nodes**
- Each node runs an **instance of Corda** and one or more **CorDapps** (Corda applications)
- Communication between nodes is **point-to-point** and encrypted using **TLS**
- **No global broadcasts,** lazy propagation (need-to-know principle)
- **Persistent queues** allow connections to be non-persistent
- Each node has exactly one **well-known identity**
- **Network map service** maps the identity to an IP address
- **Confidential identities** for **individual transactions**
- **Node discovery** via network map service (comparable to phone book)

Source: Corda Documentation (https://docs.corda.net/index.html)

Network: Admission to the Network

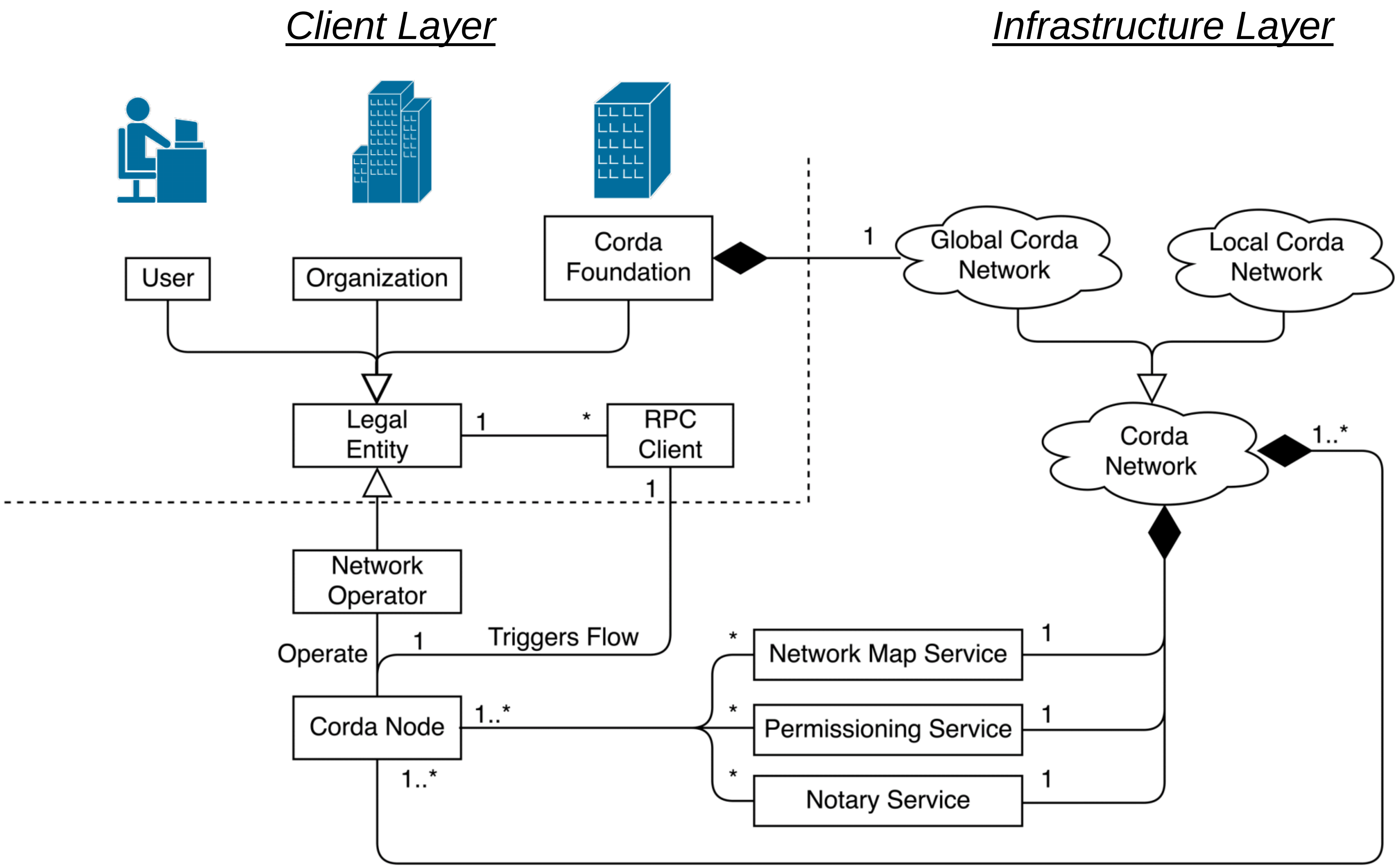


- A Corda network is a **semi-private network.**
- **Network operator** issues **certificate** for a node to join the network
- Well-known node identity provides **information** (**know-your-customer processes**) to receive the certificate
- Certificate maps node identity to a **real-world legal identity** and a **public key**

Source: Corda Documentation (https://docs.corda.net/index.html)

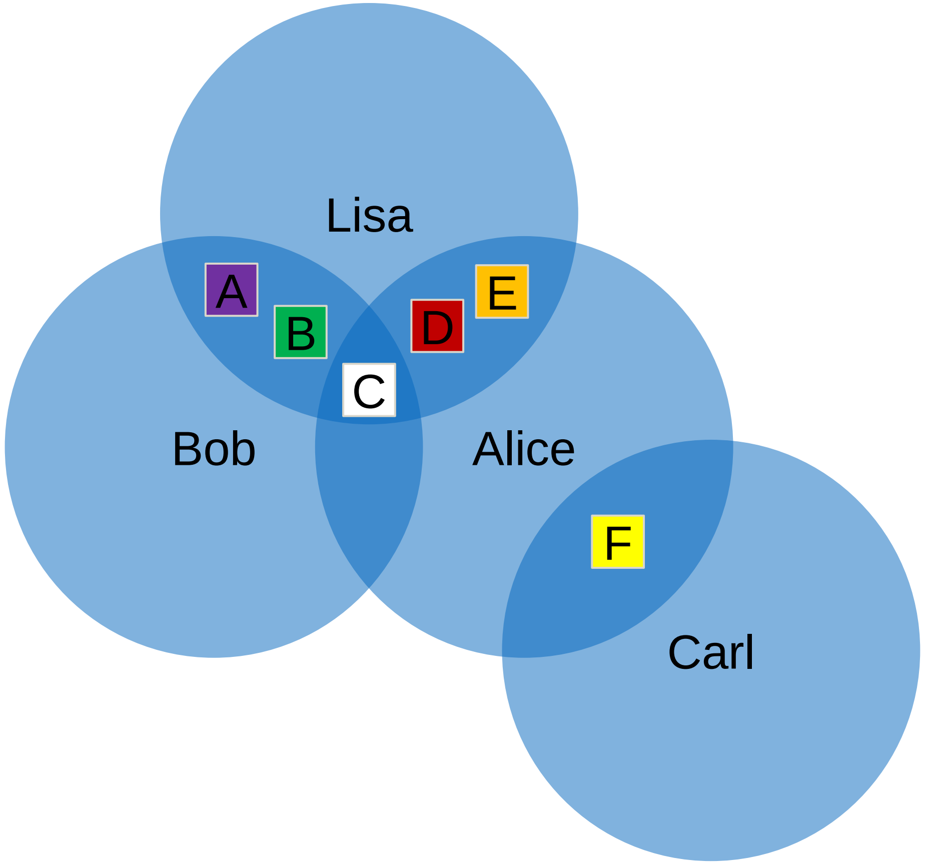
13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). "Blockchain-based Systems Engineering". Lecture Slides. TU Munich.

Entities of a Corda Network



Ledger

Corda uses individual **“vaults”** between peers instead of one central ledger like in usual blockchain-based systems like Ethereum or Bitcoin. A vault is an **accessible subset of information** in the network.



Alice, Bob, Carl and Lisa represent **nodes** in a Corda network.

All information in the network = A, B, C, D, E, F:

- Lisa has access to information: A, B, C, D, E

A,B,C is shared with Bob and C,D,E is shared with Alice

- Bob has access to information: A, B, C

A,B,C is shared with Lisa and C is shared with Alice

- Alice has access to information: C, D, E

C is shared with Bob and C, D, E is shared with Bob

- Carl has access to information: F

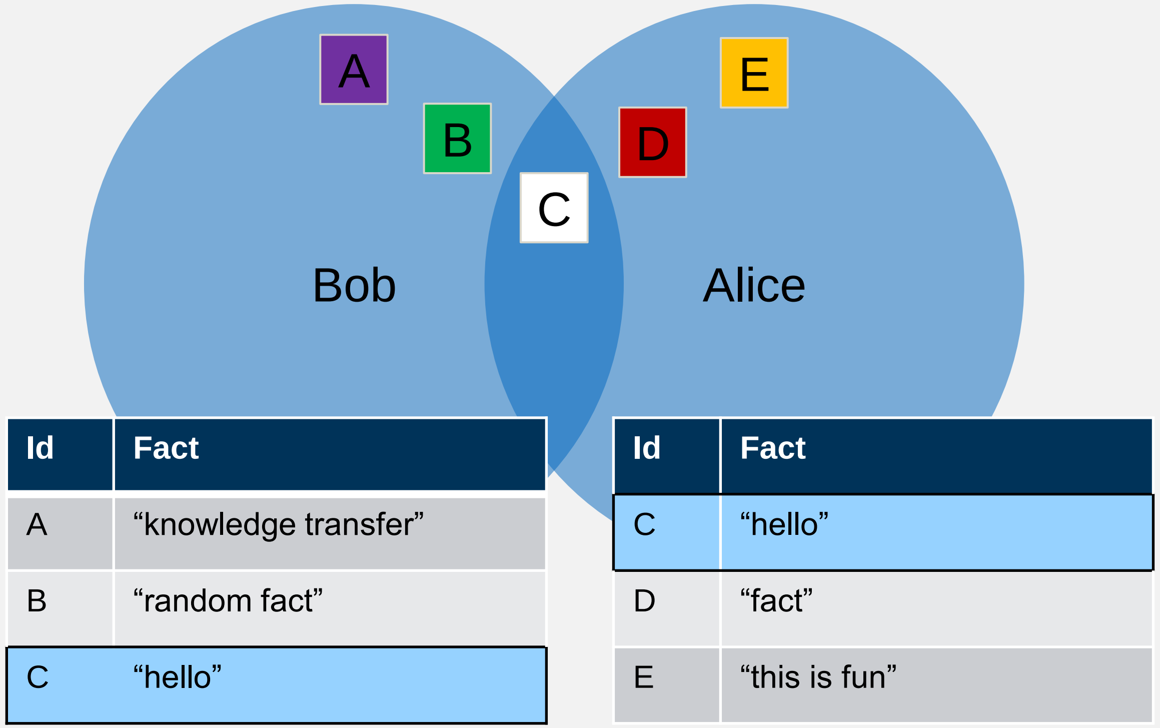
F is shared with Alice

No node is required to hold the ledger in its entirety!

Source: Corda Documentation (https://docs.corda.net/index.html)

Vaults

- The Corda ledger is **subjective** to each node
- There is **no central database** that contains all facts. The complete ledger, i.e. containing all facts, is the union of all vaults.
- Each node on the network maintains its **own vault** (set of facts)
- Technically, the vault is a **SQL database** including a table of facts
- Facts that are shared by several nodes evolve lockstep in each database, which guarantees **identical versions**



Id	Fact
A	“knowledge transfer”
B	“random fact”
C	“hello”

Id	Fact
C	“hello”
D	“fact”
E	“this is fun”

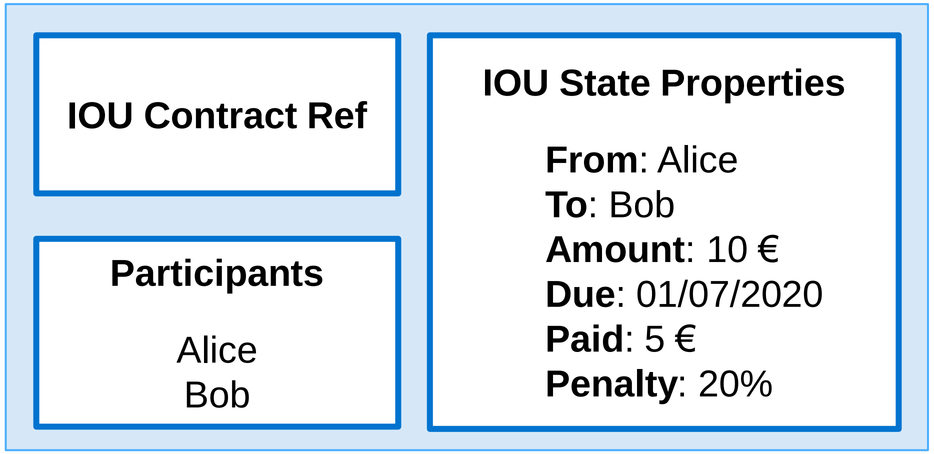
If the IDs are **equal**, the information must be **the same on both tables**

Source: Corda Documentation (https://docs.corda.net/index.html)

States

- A state is an **immutable object** representing a **fact** over which agreement is reached
- Contains **arbitrary data** to represent any kind of fact

- The **following state** represents an IOU of 10 € from Alice to Bob:



IOU Contract Ref

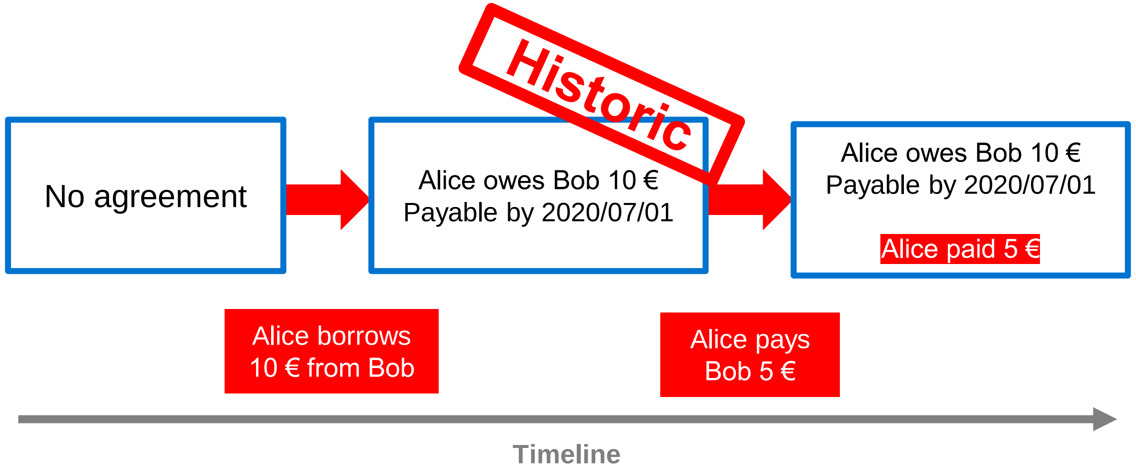
Participants

Alice
Bob

IOU State Properties

From: Alice
To: Bob
Amount: 10 €
Due: 01/07/2020
Paid: 5 €
Penalty: 20%

- A **state sequence** represents the lifecycle of a state: existing state + historic versions



No agreement

Alice owes Bob 10 €
Payable by 2020/07/01

Alice owes Bob 10 €
Payable by 2020/07/01
Alice paid 5 €

- The **vault** contains all **current and historic states** that it is aware of

Source: Corda Documentation (https://docs.corda.net/index.html)

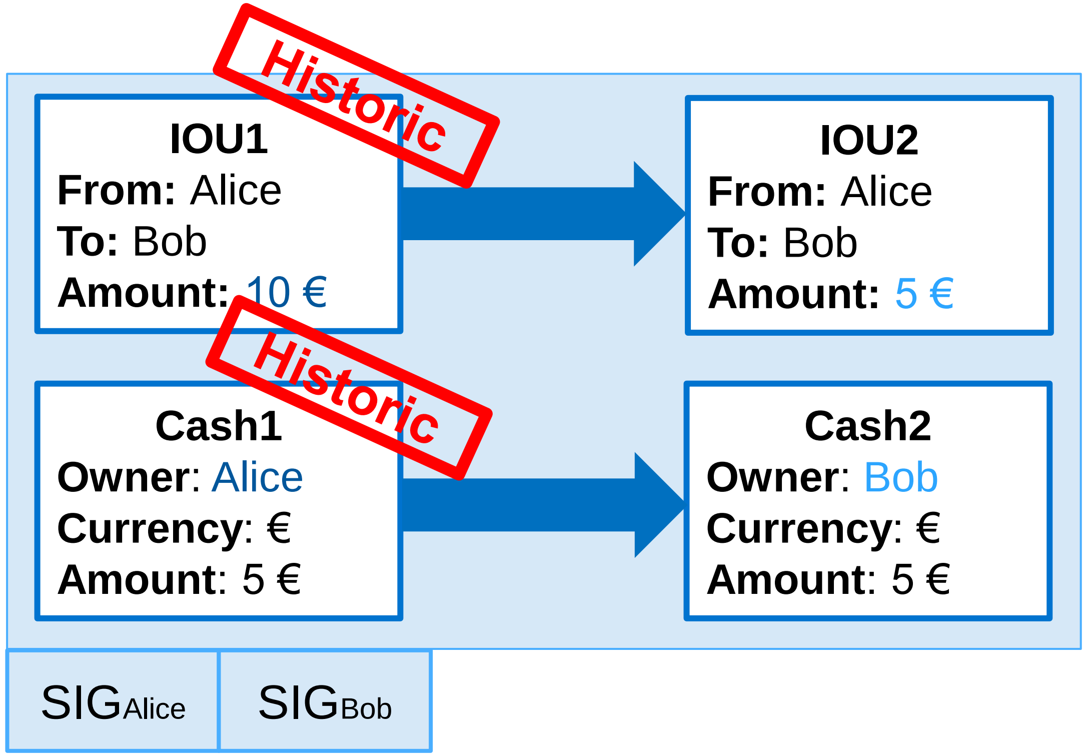
13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). "Blockchain-based Systems Engineering". Lecture Slides. TU Munich.

Transactions

- Corda uses the **UTXO model** and therefore, transactions are similar to Bitcoin transactions
- A **transaction** can contain various **inputs, outputs and references**
- **Atomicity**: Either all changes proposed in one transaction are accepted, or none of them
- Two different **types** of transactions: **Notary-change transactions** (for changes of a state's notary) and **general transactions** (for everything else)

1. Proposed transaction
2. Alice and Bob sign the proposed transaction
3. If all required signatures are gathered, the transaction gets committed

- Additionally a transaction can contain:
  - Commands
  - Notary
  - Time-window

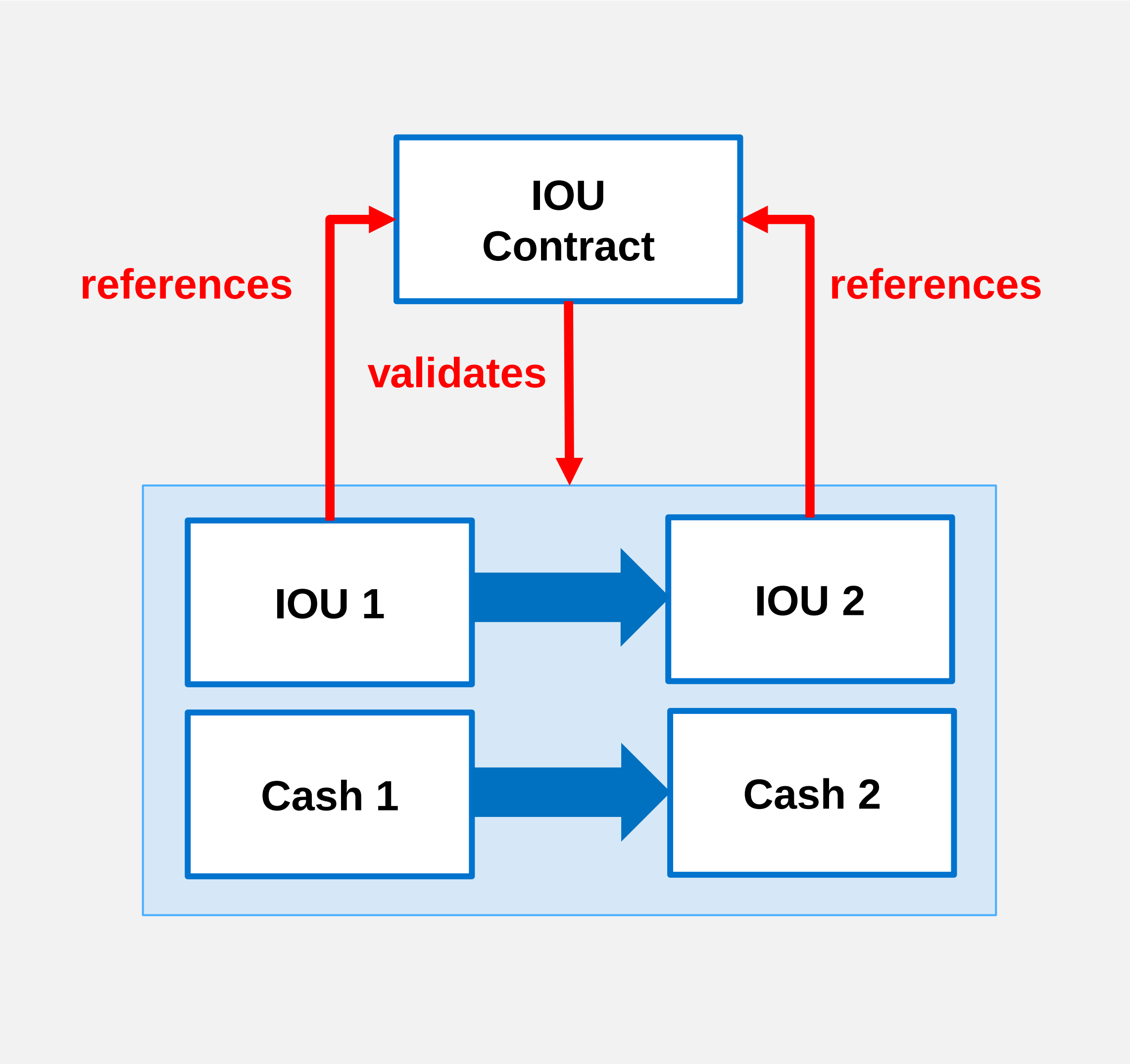


Source: Corda Documentation (https://docs.corda.net/index.html)

Contracts

- **Set of functions,** which **specify constraints** that ensure state transitions are valid according to pre-agreed **rules**
- Contracts are **deterministic**

- A **transaction is valid** if
  - all required signers **digitally signed** the transaction and
  - it is **contractually valid**
- **Contract validity**:
  - Each transaction state specifies a **contract type**
  - 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



Source: Corda Documentation (https://docs.corda.net/index.html)

Flows

- **Reminder**: Corda networks use peer-to-peer messaging instead of a global broadcast.
- **Question**: How **to coordinate a ledger update**?

- Automating the process of a ledger update using **flows**
- A flow is a **sequence of steps** that tells a node **how to achieve a specific ledger update**

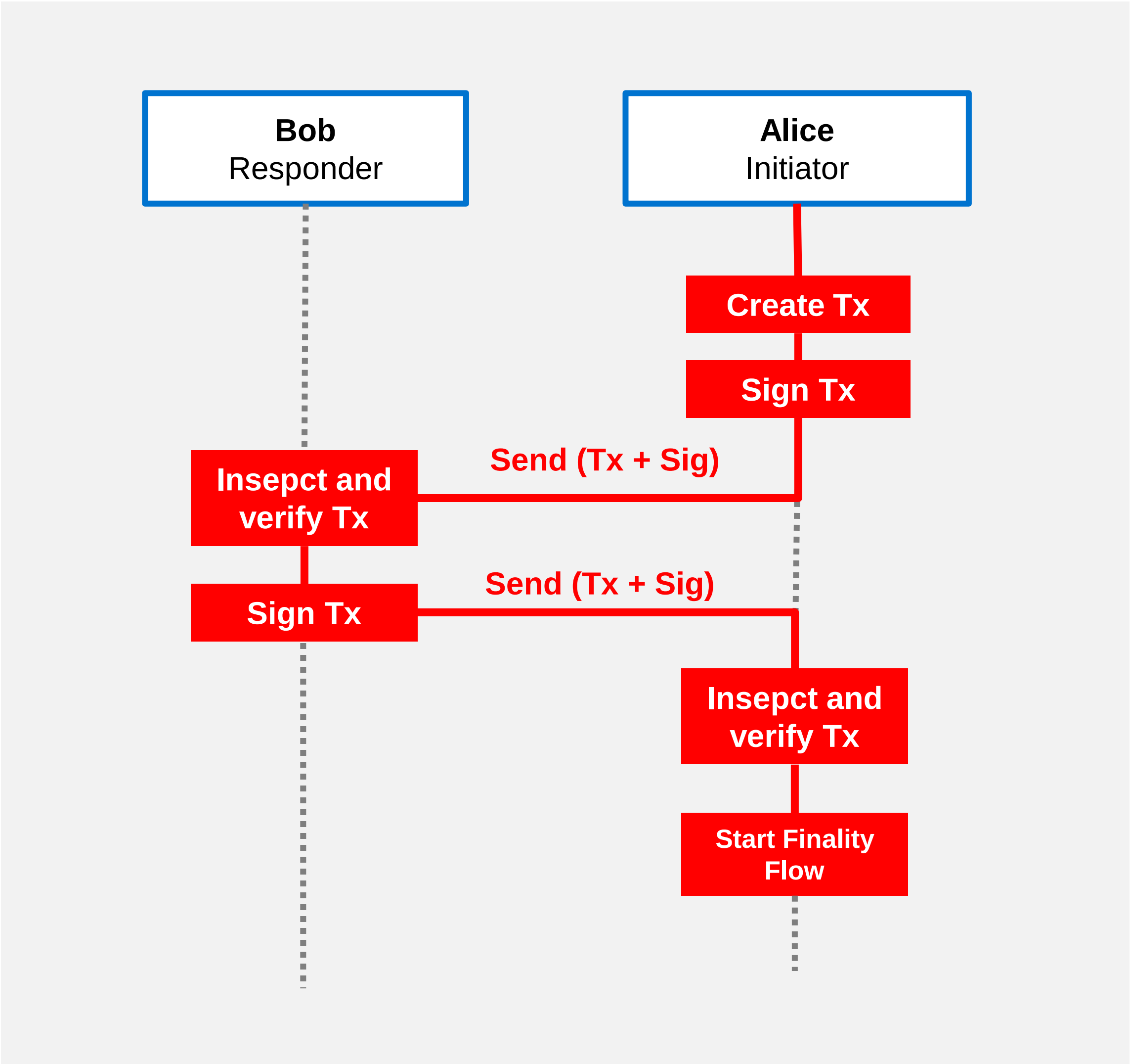
- Enables parties to **coordinate actions without a central controller**

- **Subflows**: A flow started as a subprocess in the context of another flow (**composition of flows**)
- **The flow library**: Corda provides a library of flows to handle common tasks
- **Concurrency**: The flow framework allows nodes to have many flows active at once

Source: Corda Documentation (https://docs.corda.net/index.html)

Flows

Example: Agreeing on a ledger update (Alice and Bob)



Bob
Responder

Alice
Initiator

Create Tx

Sign Tx

Send (Tx + Sig)

Insepct and verify Tx

Sign Tx

Send (Tx + Sig)

Insepct and verify Tx

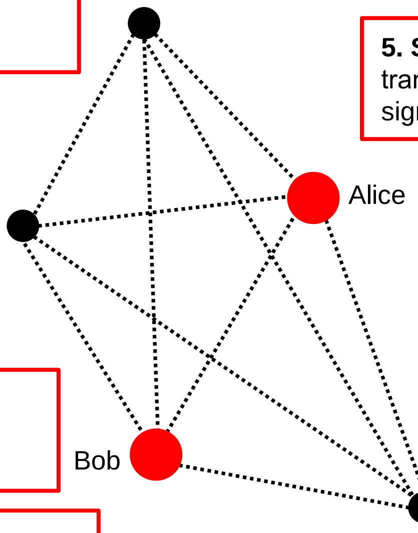
Start Finality Flow

1. Step: Alice creates a new transaction proposal and signs it

2. Step: Alice sends the proposal and her signature to Bob

Uninvolved peers do not receive any of Alice’s or Bob’s transactions.

5. Step: Alice verifies the transaction and checks Bob’s signature



3. Step: Bob inspects the proposal, verifies and signs it

4. Step: Bob sends back the transaction and his signature to Alice

Source: Corda Documentation (https://docs.corda.net/index.html)

Consensus

To determine if a transaction is valid, **two types** of consensus must be reached:

Validity consensus

- Checks that the following conditions hold for the proposed transaction and for every transaction in the transaction chain:
  - The transaction is **accepted by the contracts** of every input and output state
  - The transaction has all the **required signatures**

Uniqueness consensus

- Checks that **none of the inputs** to a proposed transaction have already been **consumed in another transaction** to avoid double-spending
- Provided by **notaries**

Source: Corda Documentation (https://docs.corda.net/index.html)

Notary Cluster

- Network **service** that provides **uniqueness consensus**
- **Notarises** a **transaction**, either through:
  - **Signing** the transaction; if the notary has not already signed other transactions consuming any of the proposed transaction’s input states
  - **Rejecting** the transaction and flag that a double-spend attempt has occurred otherwise

Notaries **maintain a map** indicating the ID of the transaction (and requesting peer) which used a **state** as an **input** (marked as historic):

Key: (Transaction ID, Output Index)

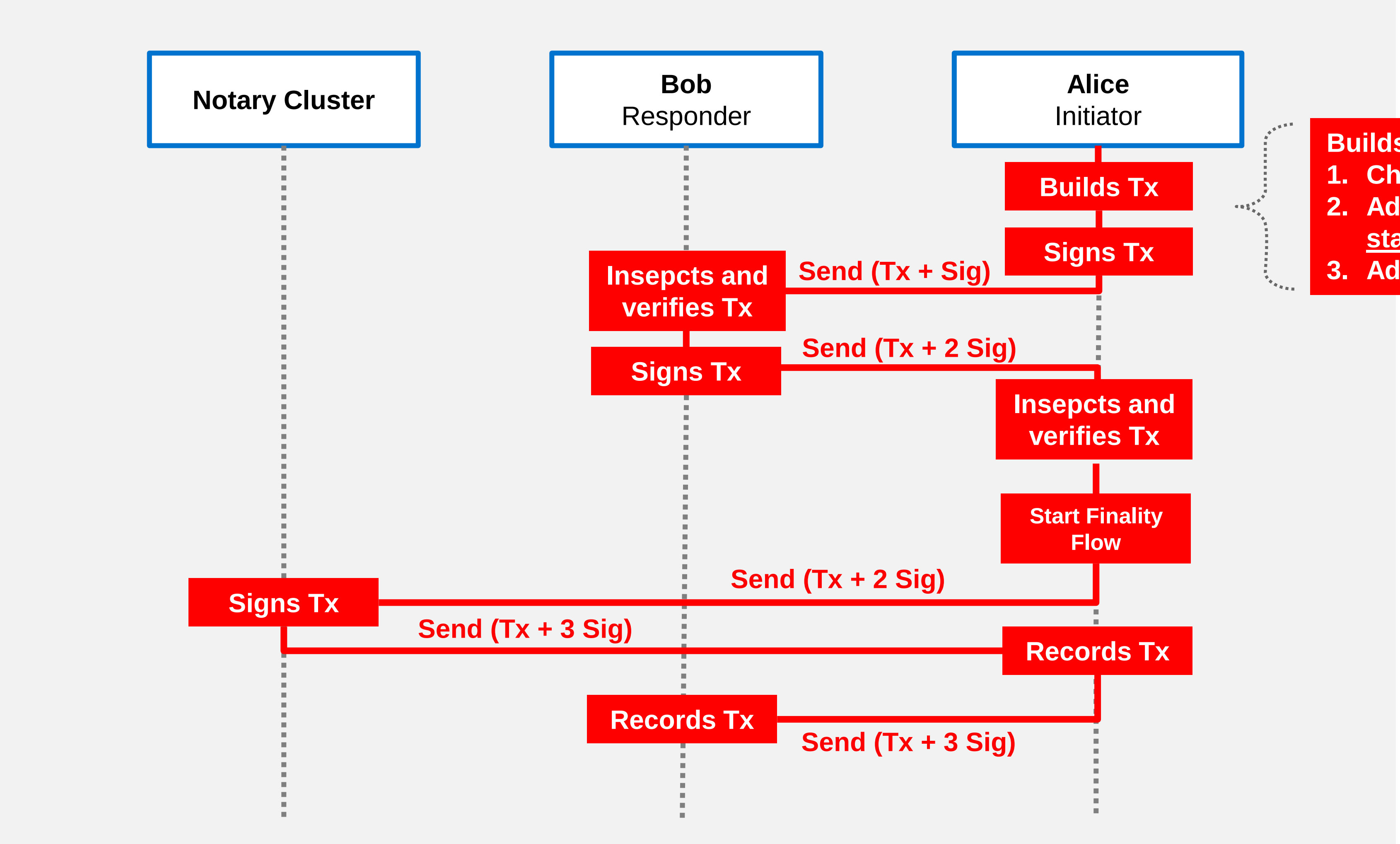
Value: (Transaction ID, Input Index, Requesting Peer)

- **Every state** has an **appointed** notary cluster
- Corda has “**pluggable**” consensus, allowing notary clusters to choose a consensus algorithm based on their requirements in terms of privacy, scalability, legal-system compatibility and algorithmic agility
- Notary clusters may differ in terms of structure, consensus algorithm and validity consensus

Source: Corda Documentation (https://docs.corda.net/index.html)

Notary Cluster

***Example*: Agreeing on a ledger update (Alice and Bob):**



Builds Tx:
1. **Choosing Notary**
2. **Adding input and output states**
3. **Adding commands**

Source: Corda Documentation (https://docs.corda.net/index.html)

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

The Global Corda Network

6thJanuary 2019 (London) – R3 launched Corda Network, the underlying, open shared DLT network linking participants using Corda. Corda Network is operated and managed by the **Corda Network Foundation**. However, the **Corda Network Operator** is in charge of operational activities, including technical activities such as hosting services, marketing activities, community management and promotion.

The network forms an **open ecosystem** with **transaction legality, finality and privacy**, where services can be offered to the global network or to specific groups within.

For a node to enter the network, it must obtain an **identity certificate**. It is issued by the Corda Network Foundation after the entity behind the node undergoes a KYC process. Once the node joins the network, the node publishes the certificate including the legal name, their IP address, public key, etc., to the **network map service**. The network map service can be seen as an address book maintained by the Corda Network Foundation.

In practicality, the **global Corda network** consists of the set of nodes in the world that are configured with common parameters such that they can locate each other, with assured identity, to transact directly. These parameters are set, maintained, and updated by the Corda Network Operator to avoid disputes between users.

Gendal Brown, R..: The Corda Platform: An Introduction (https://docs.corda.net/_static/corda-introductory-whitepaper.pdf)

The Global Corda Network: Basic components

The **basic components** of the global Corda Network are

i. the network parameters (define the consensus guidelines nodes must follow for global compatibility),
ii. an identity framework (thus firms can enter into real-world contracts with confidence),
iii. recommendations for notary pools (provide unique consensus services),
iv. Oracles (provide information services),
v. facilitation of digital tokens (for seamless transfer of value),
vi. reliance on open governance (represents each stakeholder of the Corda ecosystem),
vii. network map service (publishes information about each node in the network),
viii. and notary service.

Gendal Brown, R..: The Corda Platform: An Introduction (https://docs.corda.net/_static/corda-introductory-whitepaper.pdf)

Outline

1. Introduction to Corda
   1. History
   2. Definition and mission
   3. Vision and principles
   4. The Corda Foundation
   5. Novel features
2. System Architecture
   1. Network
   2. Ledger
   3. Vaults
   4. States
   5. Transactions
   6. Contracts
   7. Flows / Flow Framework
   8. Consensus
   9. Notaries
   10. The Global Corda Network
3. CorDapps
   1. Concept: Concepts and components
   2. Cordapp example: Token minting
   3. Cordapps: Real world CorDapps
   4. Conclusion

CorDapps: Concept and components

CorDapps (Corda Distributed Applications) are distributed applications that run on the Corda platform. The goal of a CorDapp is to allow nodes to reach agreement on updates to the ledger. They achieve this goal by defining flows that Corda node owners can invoke over **RPC**:

At a high level, a CorDapp is a combination of:

(1) State objects (Data over which agreements are reached)

(2) Contracts (Define what is a valid ledger update)

(3) Flows (Business logic routine for the node to update the ledger)

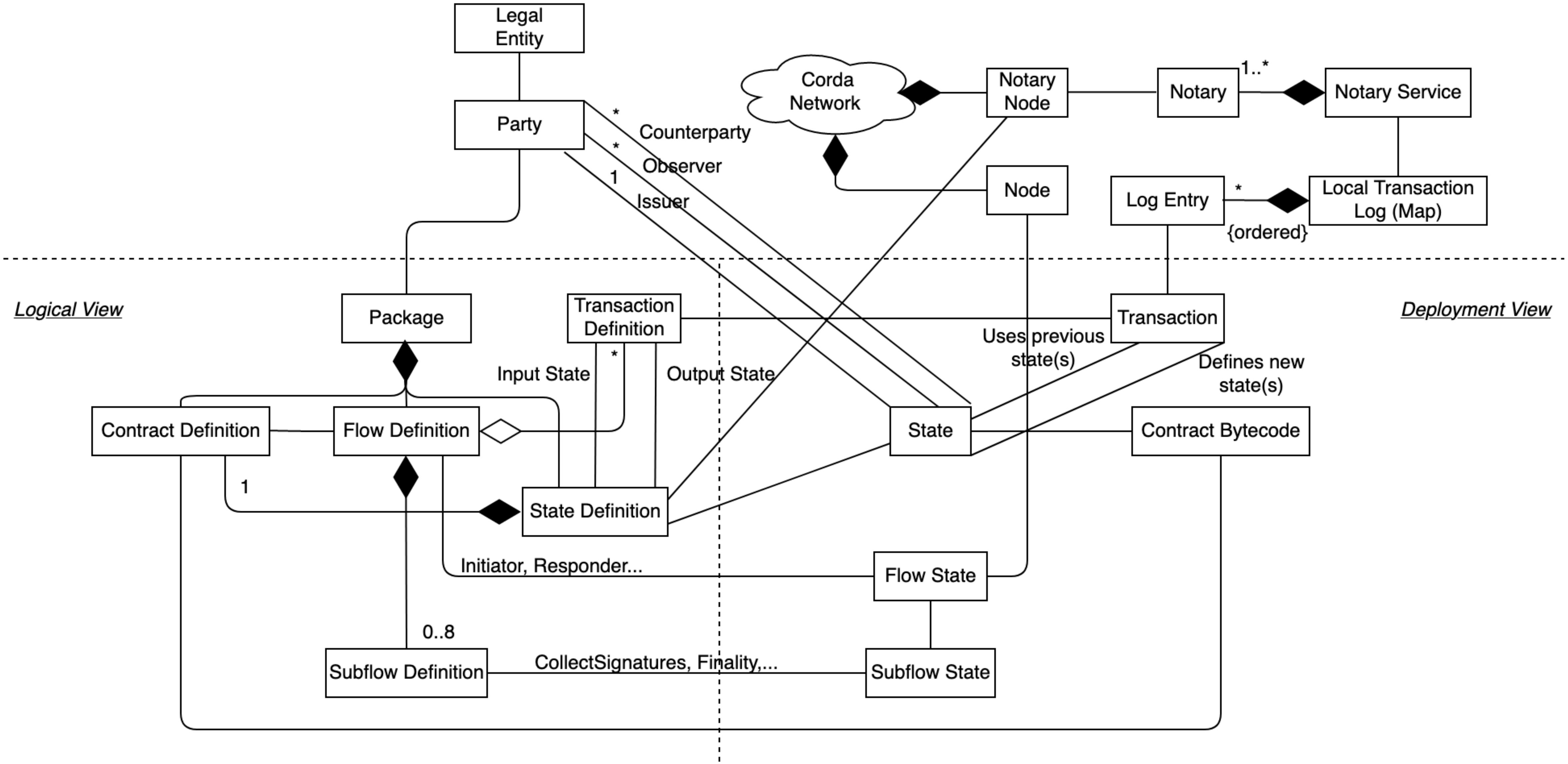
(4) Transactions (Update the ledger states)

(5) APIs and a UI (Served Corda's built in web-server)

In Ethereum, each full node contains a copy of all the smart contracts available. However, in Corda, for a node owner to use a CorDapp, she needs to install the CorDapp in her node.

Corda Official docs (https://docs.corda.net/cordapp-overview.html)

CoreDapps: Logica and deployment view



13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

CorDapp example: Token minting

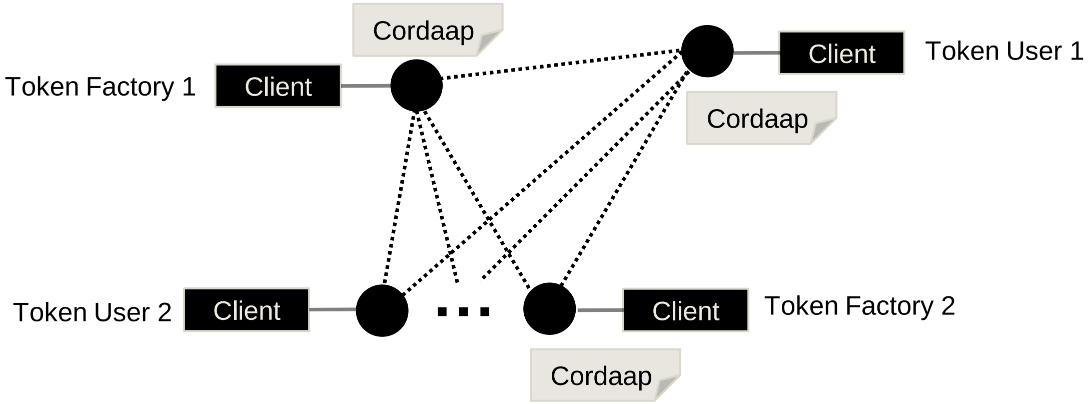
This Cordapp allows an issuer to mint an amount of tokens and assign them to a new owner.

1. There are a number of legal entities involved in this application, some of them are Token Factories, and others are Token Users.
2. Each legal entity holds at least one Corda node, forming together a local Corda Network.
3. The Corda nodes of each legal entity are managed by a Corda client.
4. The token factories program collaboratively a Cordapp to mint tokens.
5. Each token factory installs such Cordapp. Token users do not need to install this Cordapp in order to receive tokens. To use them however, given that there are more functionalities in the Cordapp e.g. sending tokens, users must have the Cordapp installed.
6. Any token factory can mint an amount of tokens and assign them to any token user, who agrees to accept it.

1. Step: Token Factory 1 creates a transaction to create X tokens to be assigned to Token User 2.

2. Step: A notary cluster of the network's choosing verifies the transaction.

3. Step: Everyone declared as a participant is notified of the transaction and Token User 2 has new X extra tokens.

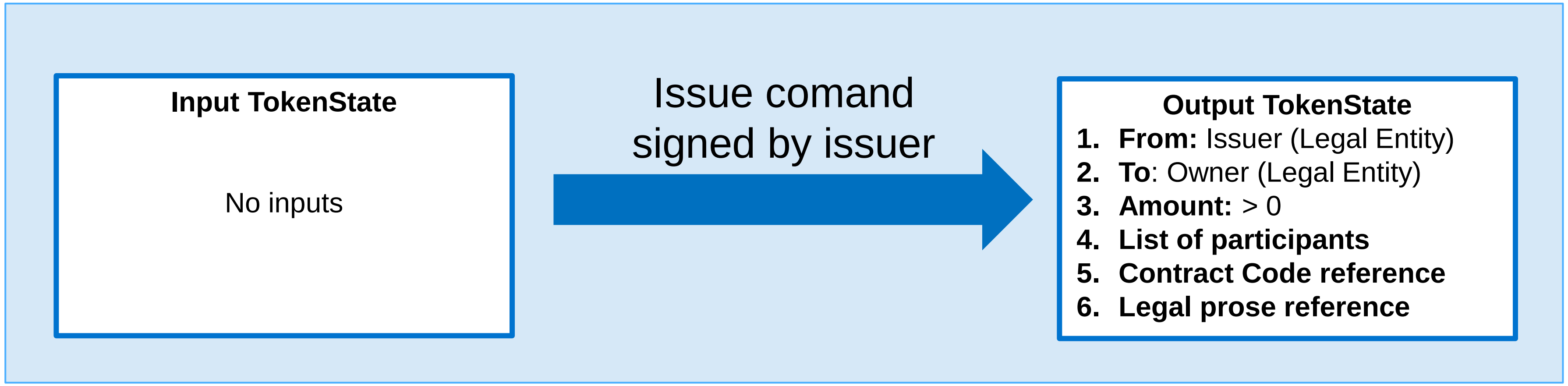


GitHub repositories of CorDapps (https://github.com/corda/bootcamp-cordapp)

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). "Blockchain-based Systems Engineering". Lecture Slides. TU Munich.

CorDapp example: Token minting

There is no need to consume an input state, as we are minting a token.



Input TokenState

No inputs

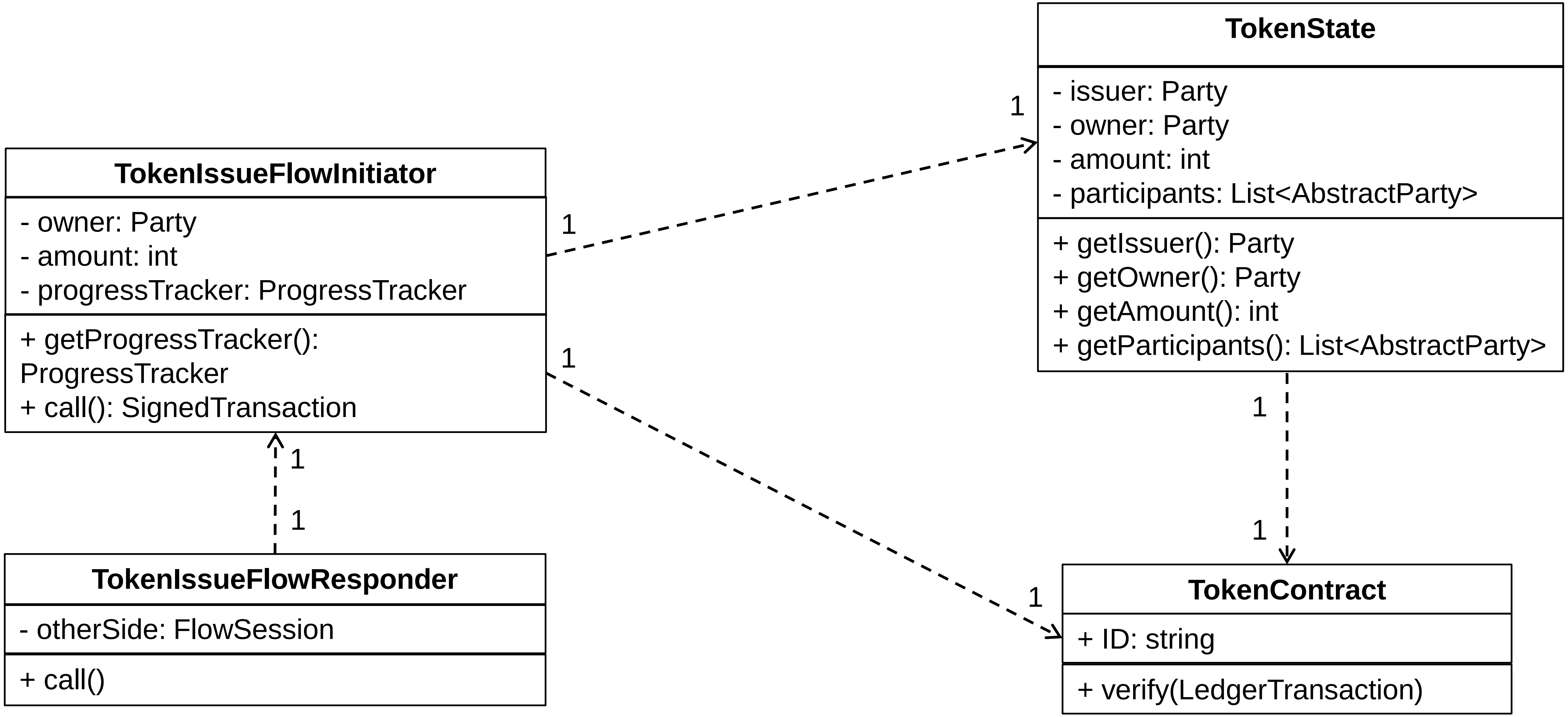
Issue comand
signed by issuer

Output TokenState

1. **From:** Issuer (Legal Entity)
2. **To**: Owner (Legal Entity)
3. **Amount:** > 0
4. **List of participants**
5. **Contract Code reference**
6. **Legal prose reference**

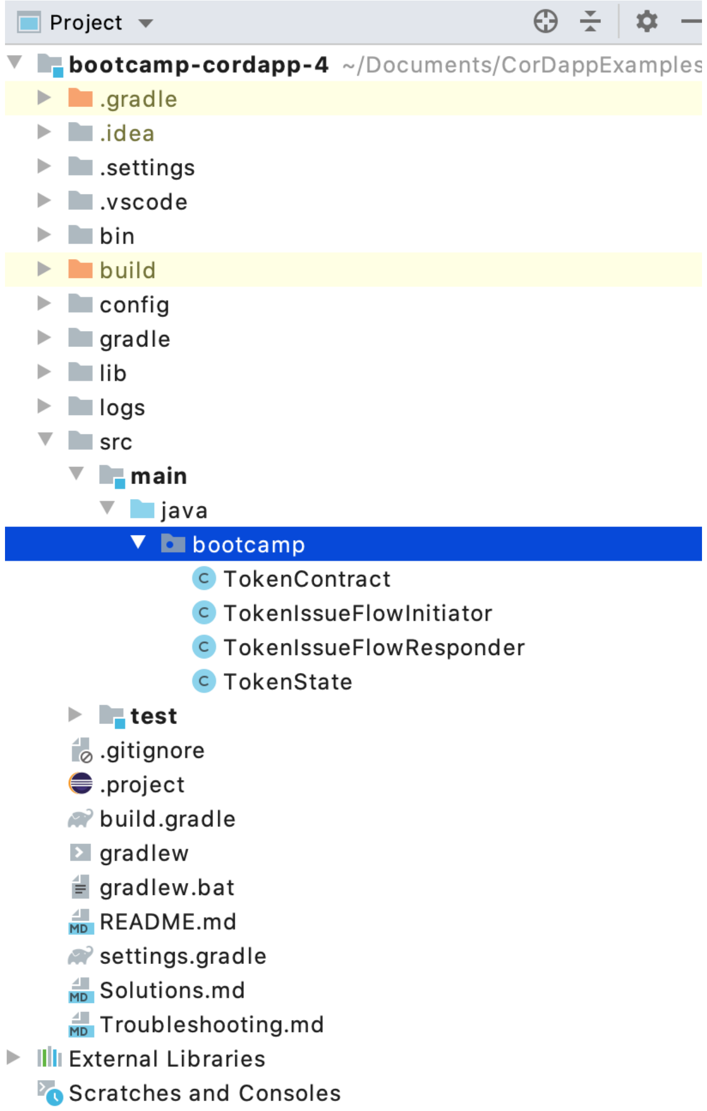
GitHub repositories of CorDapps (https://github.com/corda/bootcamp-cordapp)

CorDapp example: Class UML of token minting



13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

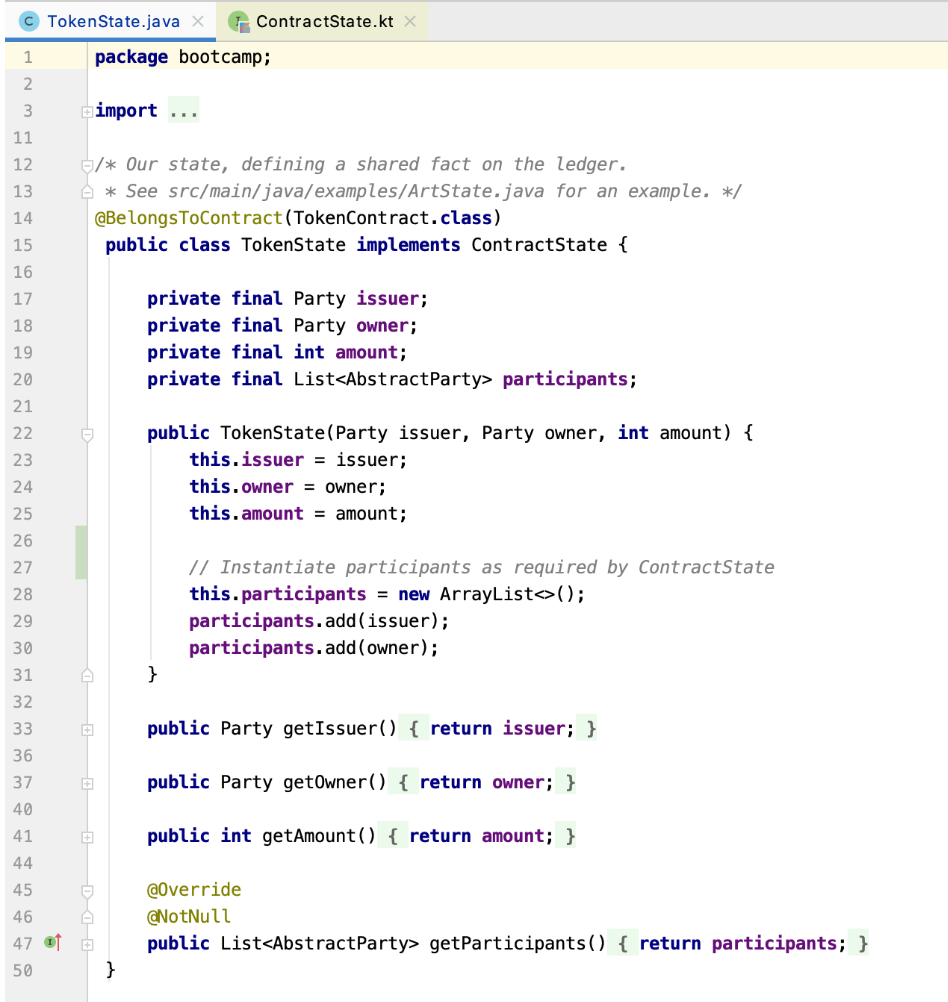
CorDapp Example: Overview



```
Project
▼ bootcamp-cordapp-4  ~/Documents/CorDappExamples
    ▶ .gradle
    ▶ .idea
    ▶ .settings
    ▶ .vscode
    ▶ bin
    ▶ build
    ▶ config
    ▶ gradle
    ▶ lib
    ▶ logs
    ▼ src
        ▼ main
            ▼ java
                ▼ bootcamp
                    TokenContract
                    TokenIssueFlowInitiator
                    TokenIssueFlowResponder
                    TokenState
        ▶ test
    .gitignore
    .project
    build.gradle
    gradlew
    gradlew.bat
    README.md
    settings.gradle
    Solutions.md
    Troubleshooting.md
▶ External Libraries
  Scratches and Consoles
```

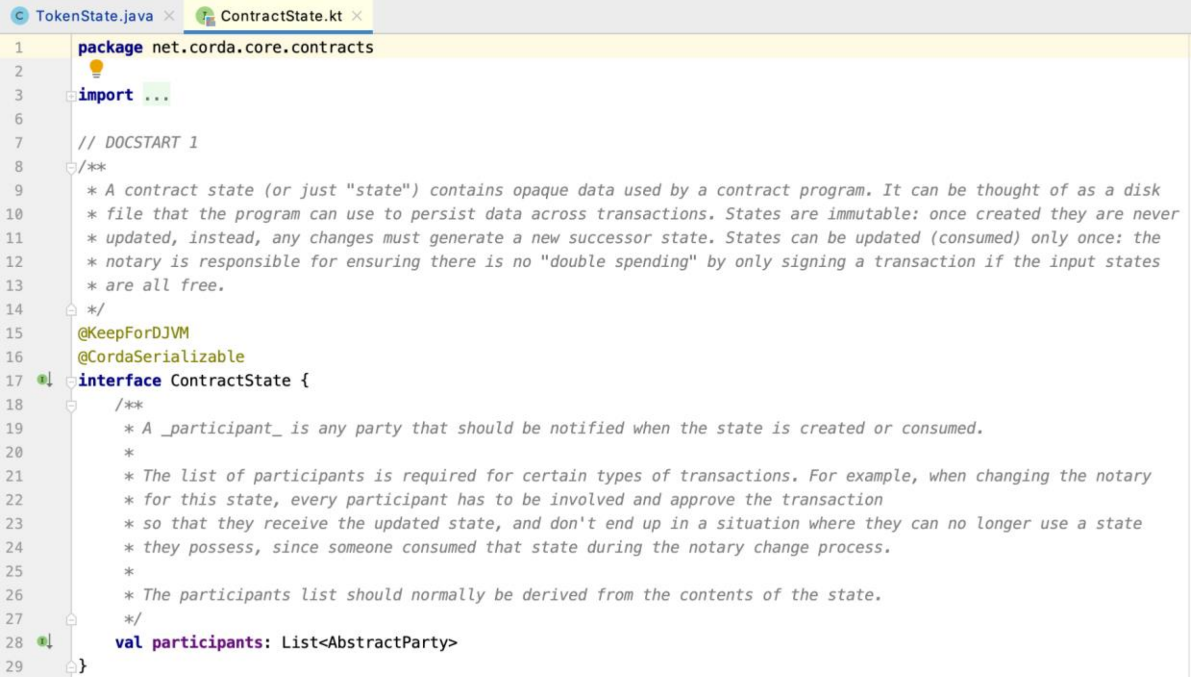
CorDapp Example: TokenState

TokenState



```java
package bootcamp;

import ...

/* Our state, defining a shared fact on the ledger.
 * See src/main/java/examples/ArtState.java for an example. */
@BelongsToContract(TokenContract.class)
public class TokenState implements ContractState {

    private final Party issuer;
    private final Party owner;
    private final int amount;
    private final List<AbstractParty> participants;

    public TokenState(Party issuer, Party owner, int amount) {
        this.issuer = issuer;
        this.owner = owner;
        this.amount = amount;

        // Instantiate participants as required by ContractState
        this.participants = new ArrayList<>();
        participants.add(issuer);
        participants.add(owner);
    }

    public Party getIssuer() { return issuer; }

    public Party getOwner() { return owner; }

    public int getAmount() { return amount; }

    @Override
    @NotNull
    public List<AbstractParty> getParticipants() { return participants; }
}
```

ContractState

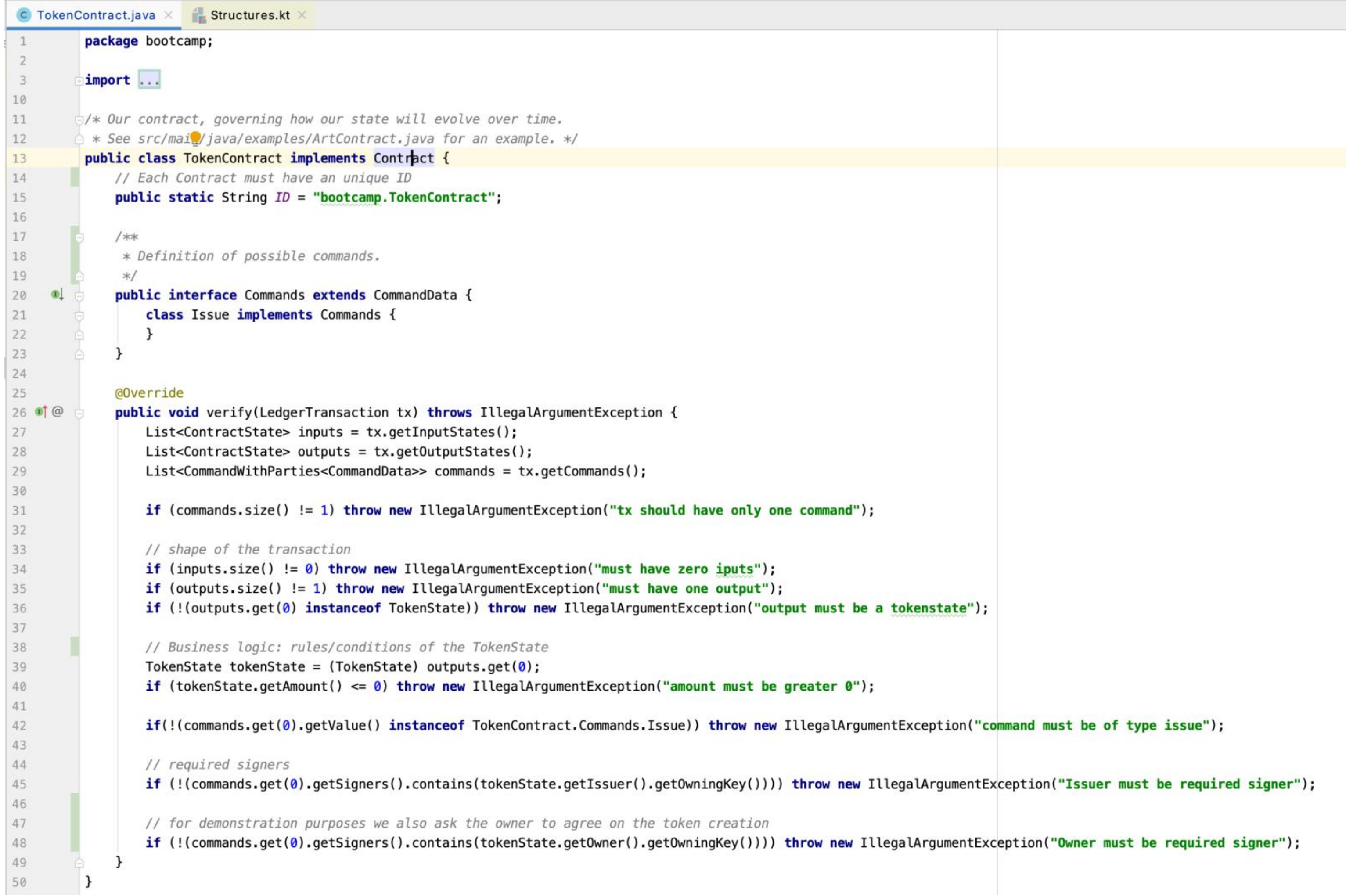


```kotlin
package net.corda.core.contracts

import ...

// DOCSTART 1
/**
 * A contract state (or just "state") contains opaque data used by a contract program. It can be thought of as a disk
 * file that the program can use to persist data across transactions. States are immutable: once created they are never
 * updated, instead, any changes must generate a new successor state. States can be updated (consumed) only once: the
 * notary is responsible for ensuring there is no "double spending" by only signing a transaction if the input states
 * are all free.
 */
@KeepForDJVM
@CordaSerializable
interface ContractState {
    /**
     * A _participant_ is any party that should be notified when the state is created or consumed.
     *
     * The list of participants is required for certain types of transactions. For example, when changing the notary
     * for this state, every participant has to be involved and approve the transaction
     * so that they receive the updated state, and don't end up in a situation where they can no longer use a state
     * they possess, since someone consumed that state during the notary change process.
     *
     * The participants list should normally be derived from the contents of the state.
     */
    val participants: List<AbstractParty>
}
```

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

CorDapp Example: TokenContract

TokenContract

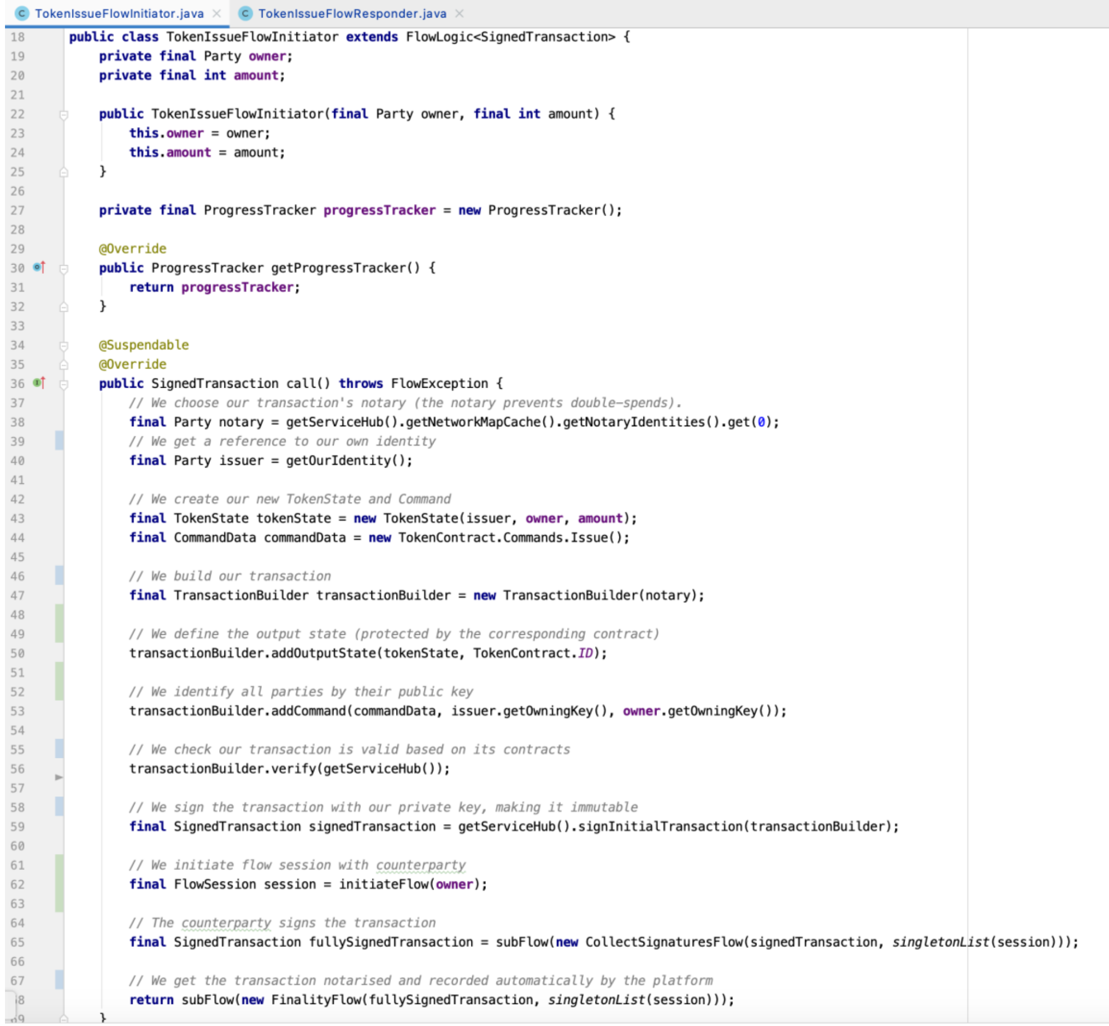


```java
package bootcamp;

import ...

/* Our contract, governing how our state will evolve over time.
 * See src/main/java/examples/ArtContract.java for an example. */
public class TokenContract implements Contract {
    // Each Contract must have an unique ID
    public static String ID = "bootcamp.TokenContract";

    /**
     * Definition of possible commands.
     */
    public interface Commands extends CommandData {
        class Issue implements Commands {
        }
    }

    @Override
    public void verify(LedgerTransaction tx) throws IllegalArgumentException {
        List<ContractState> inputs = tx.getInputStates();
        List<ContractState> outputs = tx.getOutputStates();
        List<CommandWithParties<CommandData>> commands = tx.getCommands();

        if (commands.size() != 1) throw new IllegalArgumentException("tx should have only one command");

        // shape of the transaction
        if (inputs.size() != 0) throw new IllegalArgumentException("must have zero iputs");
        if (outputs.size() != 1) throw new IllegalArgumentException("must have one output");
        if (!(outputs.get(0) instanceof TokenState)) throw new IllegalArgumentException("output must be a tokenstate");

        // Business logic: rules/conditions of the TokenState
        TokenState tokenState = (TokenState) outputs.get(0);
        if (tokenState.getAmount() <= 0) throw new IllegalArgumentException("amount must be greater 0");

        if(!(commands.get(0).getValue() instanceof TokenContract.Commands.Issue)) throw new IllegalArgumentException("command must be of type issue");

        // required signers
        if (!(commands.get(0).getSigners().contains(tokenState.getIssuer().getOwningKey()))) throw new IllegalArgumentException("Issuer must be required signer");

        // for demonstration purposes we also ask the owner to agree on the token creation
        if (!(commands.get(0).getSigners().contains(tokenState.getOwner().getOwningKey()))) throw new IllegalArgumentException("Owner must be required signer");
    }
}
```

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

CorDapp Example: TokenIssueFlowInitiator and TokenIssueFlowResponder

TokenIssueFlowInitiator

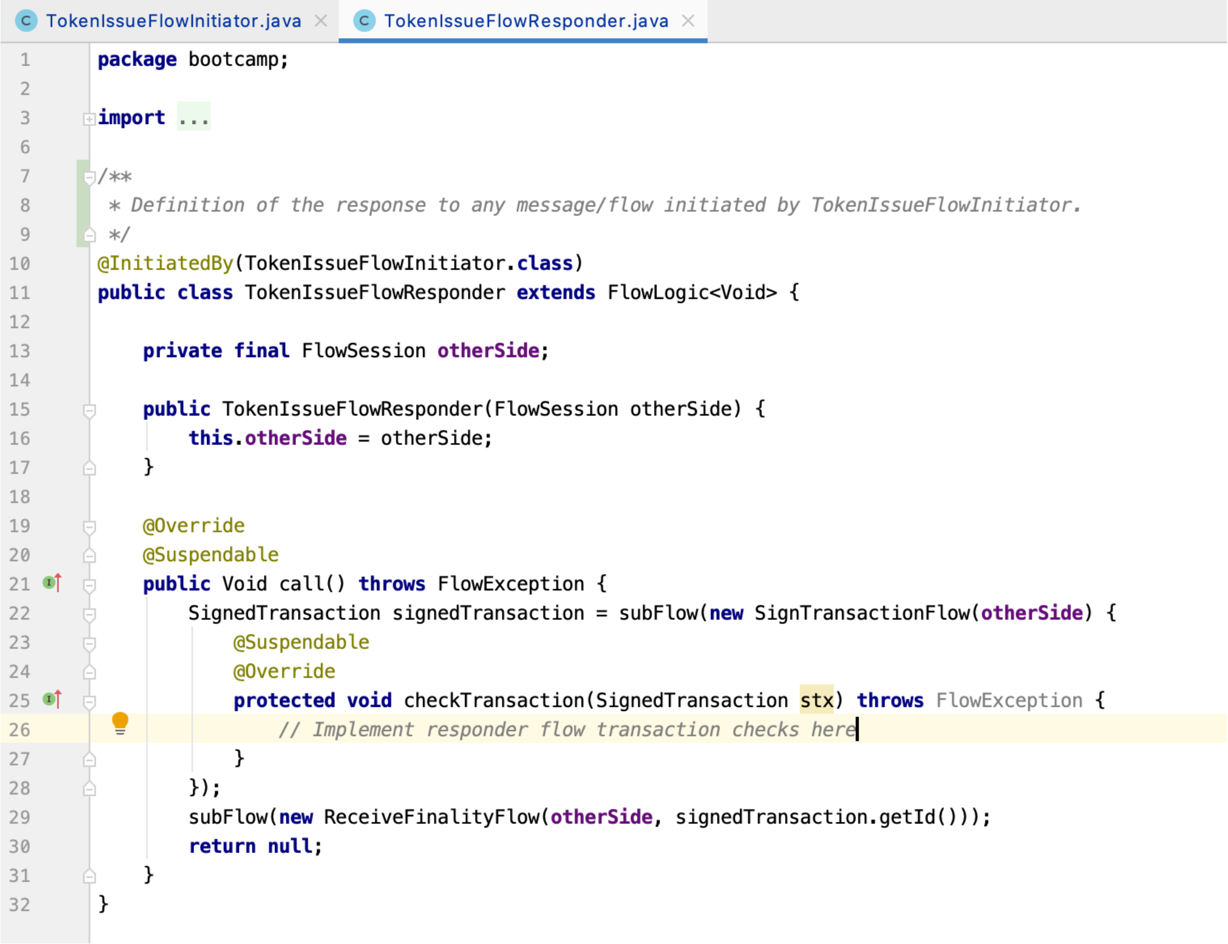


```java
public class TokenIssueFlowInitiator extends FlowLogic<SignedTransaction> {
    private final Party owner;
    private final int amount;

    public TokenIssueFlowInitiator(final Party owner, final int amount) {
        this.owner = owner;
        this.amount = amount;
    }

    private final ProgressTracker progressTracker = new ProgressTracker();

    @Override
    public ProgressTracker getProgressTracker() {
        return progressTracker;
    }

    @Suspendable
    @Override
    public SignedTransaction call() throws FlowException {
        // We choose our transaction's notary (the notary prevents double-spends).
        final Party notary = getServiceHub().getNetworkMapCache().getNotaryIdentities().get(0);
        // We get a reference to our own identity
        final Party issuer = getOurIdentity();

        // We create our new TokenState and Command
        final TokenState tokenState = new TokenState(issuer, owner, amount);
        final CommandData commandData = new TokenContract.Commands.Issue();

        // We build our transaction
        final TransactionBuilder transactionBuilder = new TransactionBuilder(notary);

        // We define the output state (protected by the corresponding contract)
        transactionBuilder.addOutputState(tokenState, TokenContract.ID);

        // We identify all parties by their public key
        transactionBuilder.addCommand(commandData, issuer.getOwningKey(), owner.getOwningKey());

        // We check our transaction is valid based on its contracts
        transactionBuilder.verify(getServiceHub());

        // We sign the transaction with our private key, making it immutable
        final SignedTransaction signedTransaction = getServiceHub().signInitialTransaction(transactionBuilder);

        // We initiate flow session with counterparty
        final FlowSession session = initiateFlow(owner);

        // The counterparty signs the transaction
        final SignedTransaction fullySignedTransaction = subFlow(new CollectSignaturesFlow(signedTransaction, singletonList(session)));

        // We get the transaction notarised and recorded automatically by the platform
        return subFlow(new FinalityFlow(fullySignedTransaction, singletonList(session)));
    }
```

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

CorDapp Example: TokenIssueFlowInitiator and TokenIssueFlowResponder

TokenIssueFlowResponder



```java
package bootcamp;

import ...

/**
 * Definition of the response to any message/flow initiated by TokenIssueFlowInitiator.
 */
@InitiatedBy(TokenIssueFlowInitiator.class)
public class TokenIssueFlowResponder extends FlowLogic<Void> {

    private final FlowSession otherSide;

    public TokenIssueFlowResponder(FlowSession otherSide) {
        this.otherSide = otherSide;
    }

    @Override
    @Suspendable
    public Void call() throws FlowException {
        SignedTransaction signedTransaction = subFlow(new SignTransactionFlow(otherSide) {
            @Suspendable
            @Override
            protected void checkTransaction(SignedTransaction stx) throws FlowException {
                // Implement responder flow transaction checks here
            }
        });
        subFlow(new ReceiveFinalityFlow(otherSide, signedTransaction.getId()));
        return null;
    }
}
```

13 Corda - Gallersdörfer, U., Holl, P., & Matthes, F. (2020). “Blockchain-based Systems Engineering”. Lecture Slides. TU Munich.

CorDapps: Real world CorDapps

Corda may be applied to areas other than insurance and banking. Here are the key utilized features per use case, and the corresponding CorDapps.

- **Healthcare:** Focused on medical claim management (ZKP3, RevBlox).
- **Energy**: emphasis on payments and contract settlements of energy trades (EBX).
- **Government:** Property can take a digital form (Instant Property Network, UBIN).
- **Telecommunication:** Providing a ledger for corporations to share a unique identifier between customers (DMNP).
- **Insurance:** Focused on digitalization and automation of administrative tasks (B3i, Guardtime).
- **Trade finance:** Building a trade platform and secure storage of trading documents (TradeCloud, x-DeFraud).
- **Digital assets:** Trading and balance sheet management available for digital assets (IVNO, VaultChain).
- **Digital identity:** Trust provisioning of one's identity (Tradle, Luxoft).
- **Capital markets:** Transparency for markets (Finastra, HQLAx).

These uses cases share common features, such as a design for highly regulated environments, payments, digital asset trading, secure data storage and ease of access, digitalization of administrative tasks, immutability, identity verification, and provenance. However, the key distinguishing factor for using Corda instead of other DLTs lies on the ability of users to execute private exchange information.

Gendal Brown, R..: The Corda Platform: An Introduction (https://docs.corda.net/_static/corda-introductory-whitepaper.pdf)

Conclusion

Why one would choose Corda instead of other DLTs?

1. **Privacy by design:** Unless it is specified in the transaction, no other entities will have access to the information that a set of individuals exchange between each other (need-to-know basis). This provides more flexibility than e.g. the channels offered in Hyperledger fabric.

2. **Transactions may execute in parallel:** This allows horizontal scaling, in contrast to other DLTs such as Ethereum.

3. **Strong developer ecosystem:** Large developer community, industry standard libraries, known and approved technologies in use.

4. **Use of technologies already adopted in the industry:** Corda uses the JVM as the runtime environment, messaging based on AMPQ, and industry standards like SQL databases.

5. **Niche industry:** Corda was tailored to improve the flow of transactions in complex processes of the finance and insurance industry.

In summary, the perfect environment for Corda is in the financial industry, and more so if privacy and scalability are paramount in the use case.