



# coding-bootcamps.com

Introduction to Corda Blockchain Development



# Coding Bootcamps

By Jim Sullivan  
from [Coding Bootcamps](https://coding-bootcamps.com)

# Build blockchain applications with R3 Corda

Lecture Session

# Agenda

1. What is Corda?
2. Getting set up
3. Writing our CorDapp
  - a. States
  - b. Contracts
  - c. Flows
4. Running our CorDapp

# Requirements

If you are unfamiliar with Java and Kotlin, taking the following self-paced courses is highly recommended:

1. [Introduction to Java Programming](#)
2. [Learn Kotlin Programming by Examples](#)

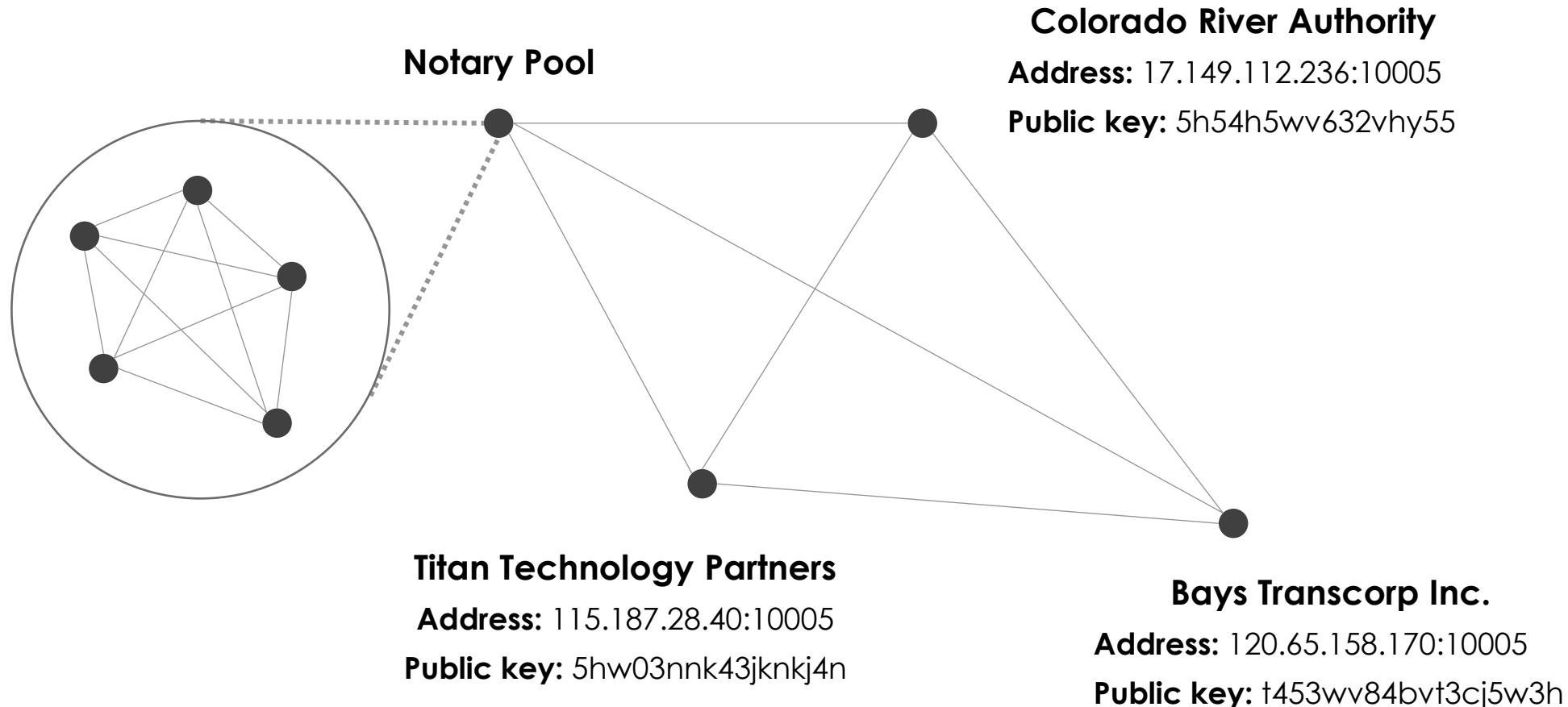
Also, if you are unfamiliar with the blockchain technology, taking the below course is recommended:

[Intro to Blockchain Technology](#)

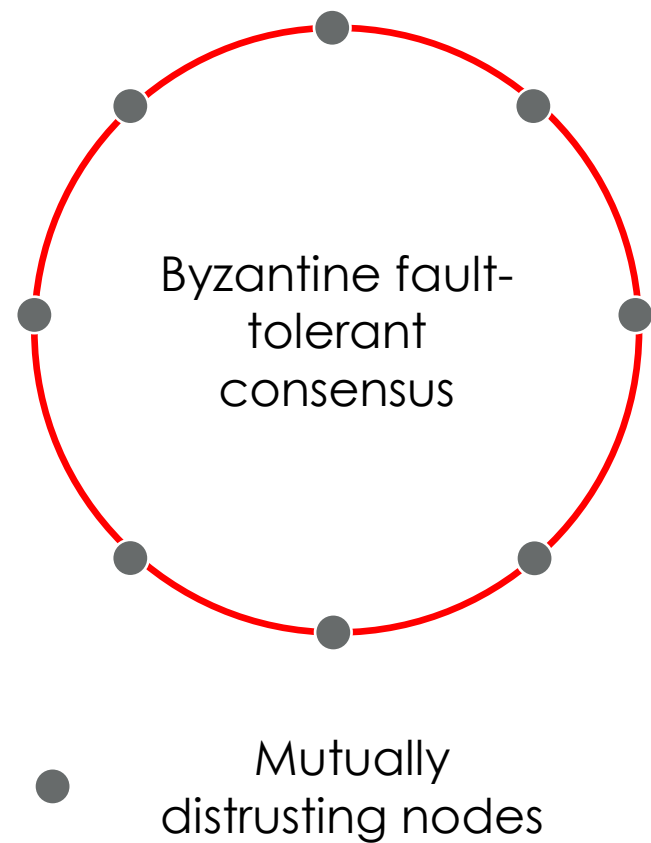


# What is Corda?

# Corda is for **permissioned nodes** to communicate on a **need-to-know basis** about updating shared facts

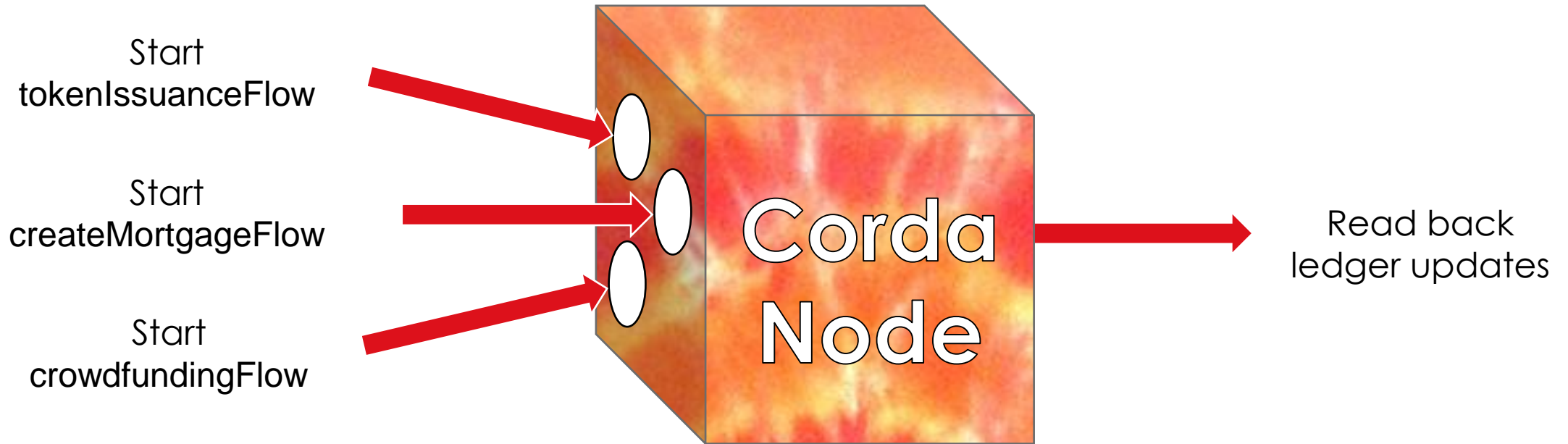


# Consensus is pluggable using notary pools



Transaction Hash	Output Index
622B2C606A23FEAE2798170	0
CC739CA364D7B396FE960B	2
628B4CE58B4A5764948AC4E	1
18265993CBC000D12DCCEE	4
F0F01F2ED2B442452499567E	2
...	...

# Corda nodes abstract away the complexity of updating the ledger



**Abstracts away:**

Messaging

Storage

Peer discovery

Data distribution

Concurrency

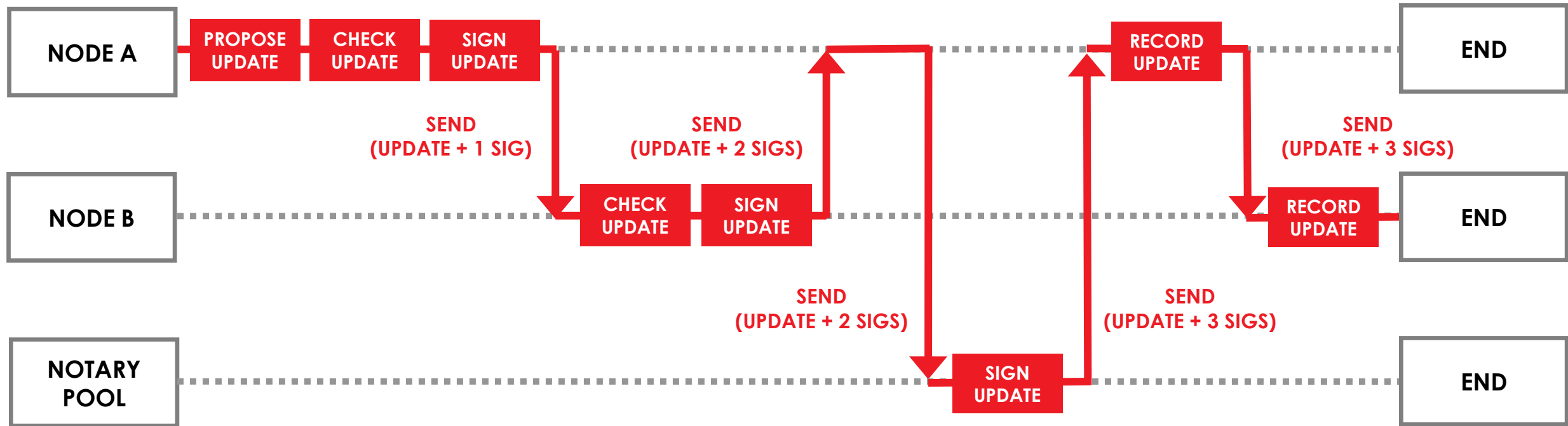
Disaster recovery

Key mgmt.

*and more!*



# The complexity is abstracted away using flows



A dark blue background with a network diagram. Red dots representing nodes are connected by thin white lines, forming a complex web. Some nodes are highlighted with thicker red lines. The overall aesthetic is technical and digital.

# Summary

Corda is a unique blockchain platform that:

- Allows private transactions
- Between legally-identifiable counterparties
- In an easy-to-use way

While maintaining the benefits of traditional blockchains



# Getting set up

1. Download and install:
  - Oracle JDK 8 JVM (minimum supported version **8u131**)
  - IntelliJ Community Edition (supported versions **2017.x** and **2018.x**)
2. Download the bootcamp-cordapp repository:  
**`git clone https://github.com/corda/bootcamp-cordapp`**
3. Open IntelliJ. From the splash screen, click **Import Project**, select the **bootcamp—cordapp** folder and click **Open**
4. Select **Import project from external model > Gradle > Next > Finish**
5. Click **File > Project Structure...** and select the **Project SDK** (Oracle JDK 8, 8u131+)
  - Add a new SDK if required by clicking **New...** and selecting the JDK's folder
6. If required, wait for Gradle and the dependencies to download
7. Open the **Project** view by clicking **View > Tool Windows > Project**
8. Run the test in **src/test/java/java\_bootcamp/ProjectImportedOKTest.java** by clicking the little green arrow. It should pass!

Check the troubleshooting guide if you get into trouble: <https://github.com/corda/bootcamp-cordapp/wiki/Troubleshooting>

The background of the slide features a dark, textured surface with a faint grid of small dots. Overlaid on this are several interconnected clusters of red circular nodes. These nodes are linked by thin, light-colored lines, forming a complex network structure that resembles a molecular model or a data visualization. The overall aesthetic is technical and modern.

# Writing our CorDapp

# The Token CorDapp



A **TokenState** class to represent tokens on the ledger  
(like Bitcoin inputs/outputs)

A **TokenContract** class to govern the evolution of tokens  
(like the Bitcoin rules – input values equal output values, owner must sign...)

A **TokenIssueFlow** class to issue tokens onto the ledger  
(building the transaction, gathering signatures, reaching consensus, updating the ledger...)

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111

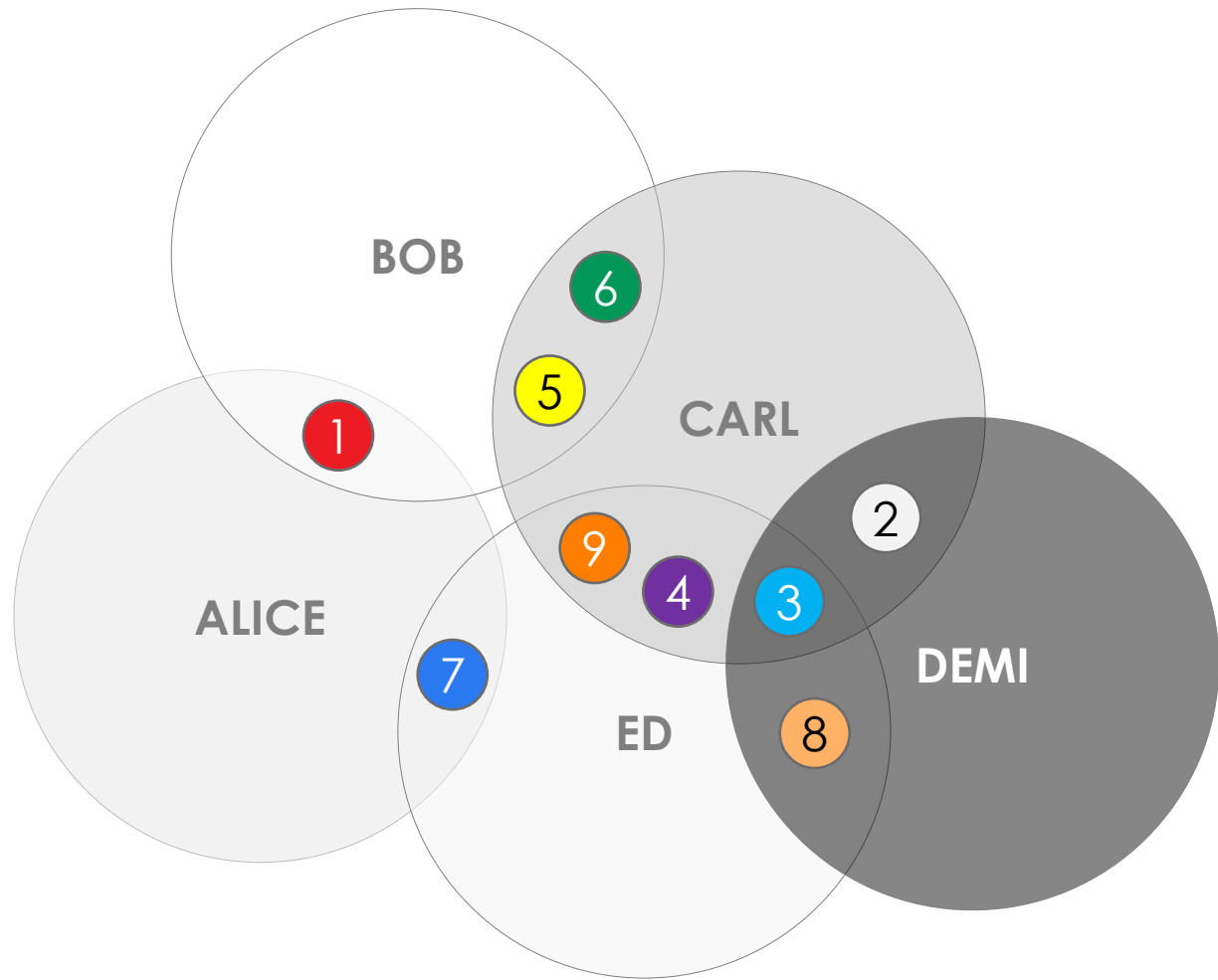


A network graph visualization on a dark background with a faint grid pattern. The graph consists of numerous red circular nodes connected by thin, light-colored lines. The nodes are arranged in several distinct clusters or communities. Some clusters are more densely connected than others. The overall structure suggests a complex network, possibly representing social connections, a web of information, or a system of interacting components. The word "States" is written in large white text on the right side of the image.

# States

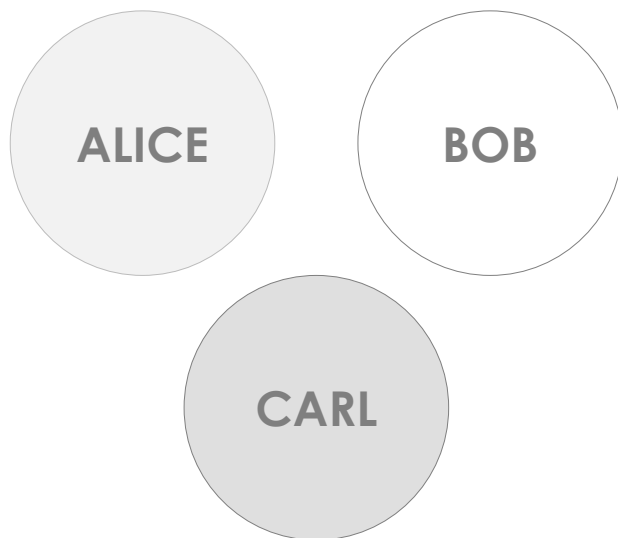


# Each node only sees a subset of states on the ledger

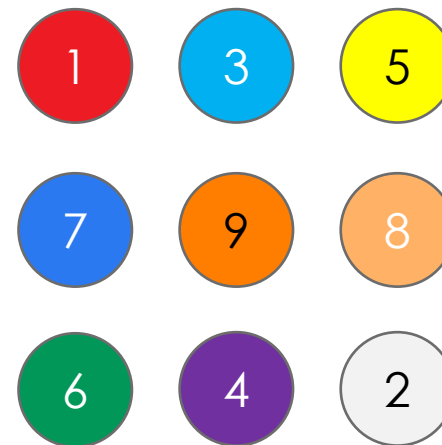


	1	2	3	4	5	6	7	8	9
Alice	x						x		
Bob	x				x	x			
Carl		x	x	x	x	x			x
Demi		x	x					x	
Ed			x	x			x	x	x

# Corda State API



Party

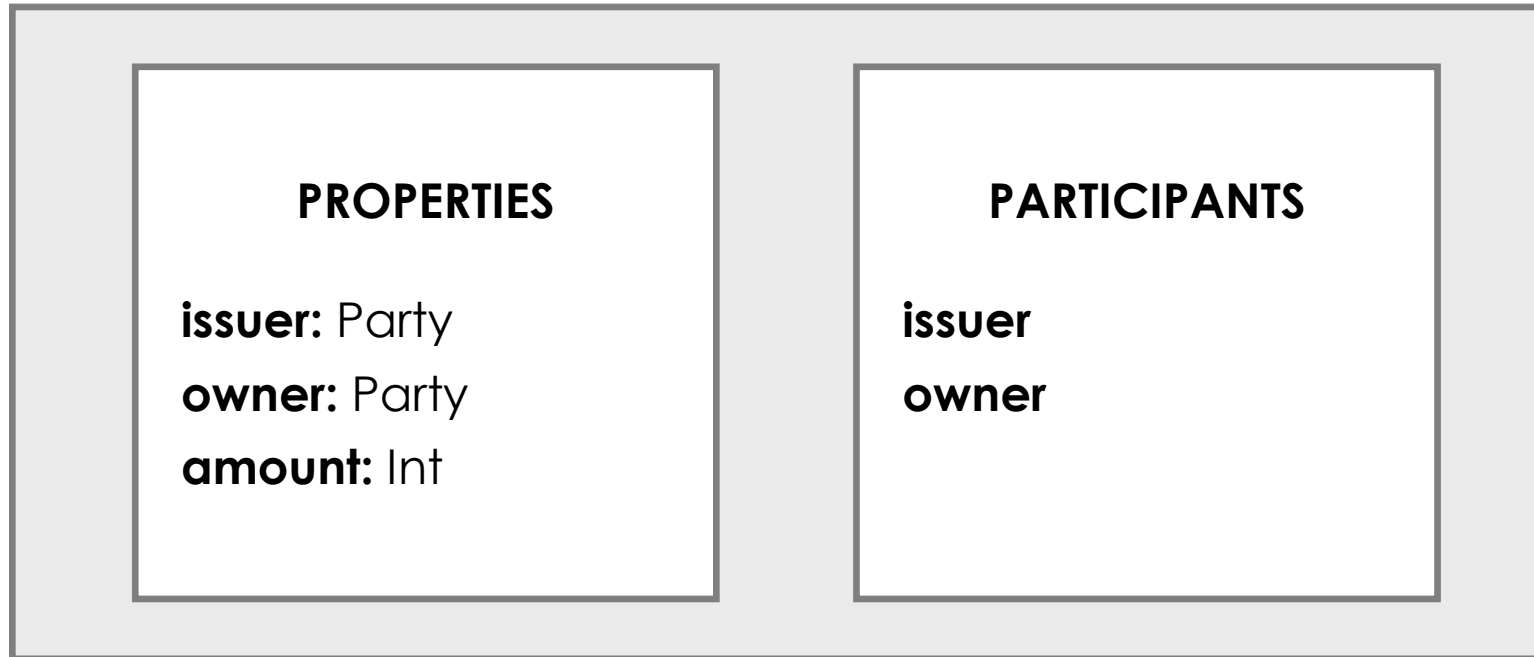


ContractState

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111

# Our state: the TokenState

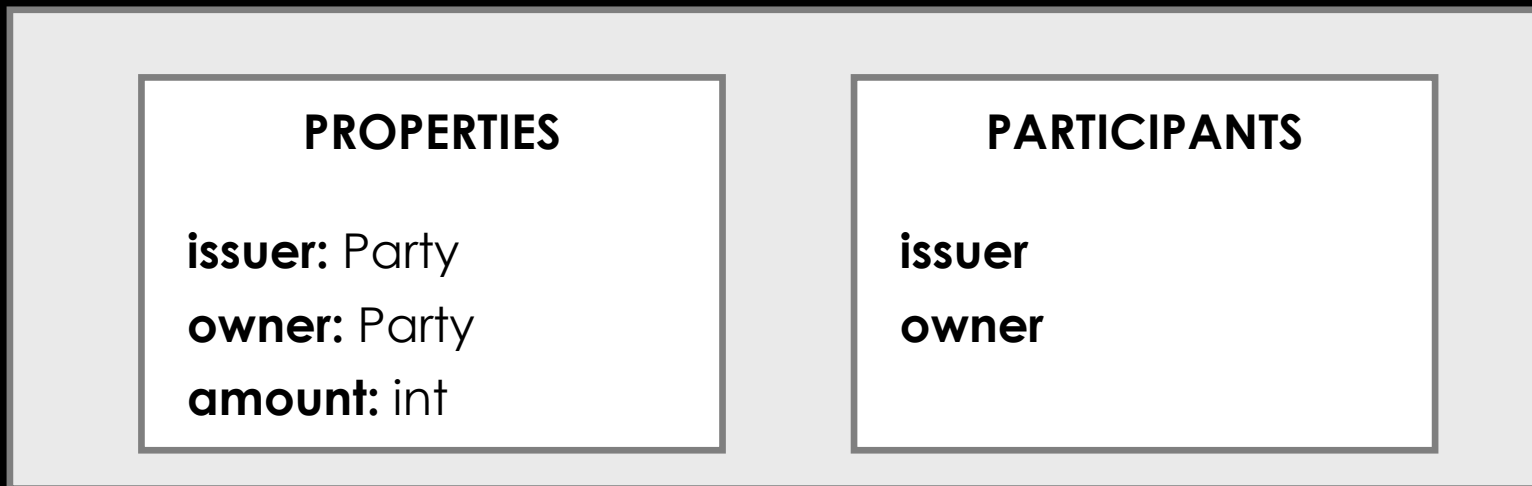
**TokenState**  
implements ContractState



1. Uncomment the tests in `src/test/java/java_bootcamp/StateTests.java` one-by-one
  2. Modify `TokenState` (defined in `src/main/java/java_bootcamp/TokenState.java`) to make the tests pass
- Refer to the wiki for resources, troubleshooting tips and solutions: <https://github.com/corda/bootcamp-cordapp/wiki>
  - Key IntelliJ shortcuts:
    - **Double-tap shift**: search everywhere
    - **Ctrl/cmd + click on function/class name**: navigate to declaration
    - **Ctrl/cmd + /**: uncomment highlighted block
    - **Alt + enter**: import class

# TokenState

implements ContractState



11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111



# Summary

- Shared facts are distributed on a need-to-know basis
- Shared facts are represented using states - instances of classes implementing the `ContractState` interface
- `ContractState` classes should have:
  - participants, who will be notified of updates
  - Additional fields to capture the shared-fact information



# Contracts



# The ledger is updated using transactions

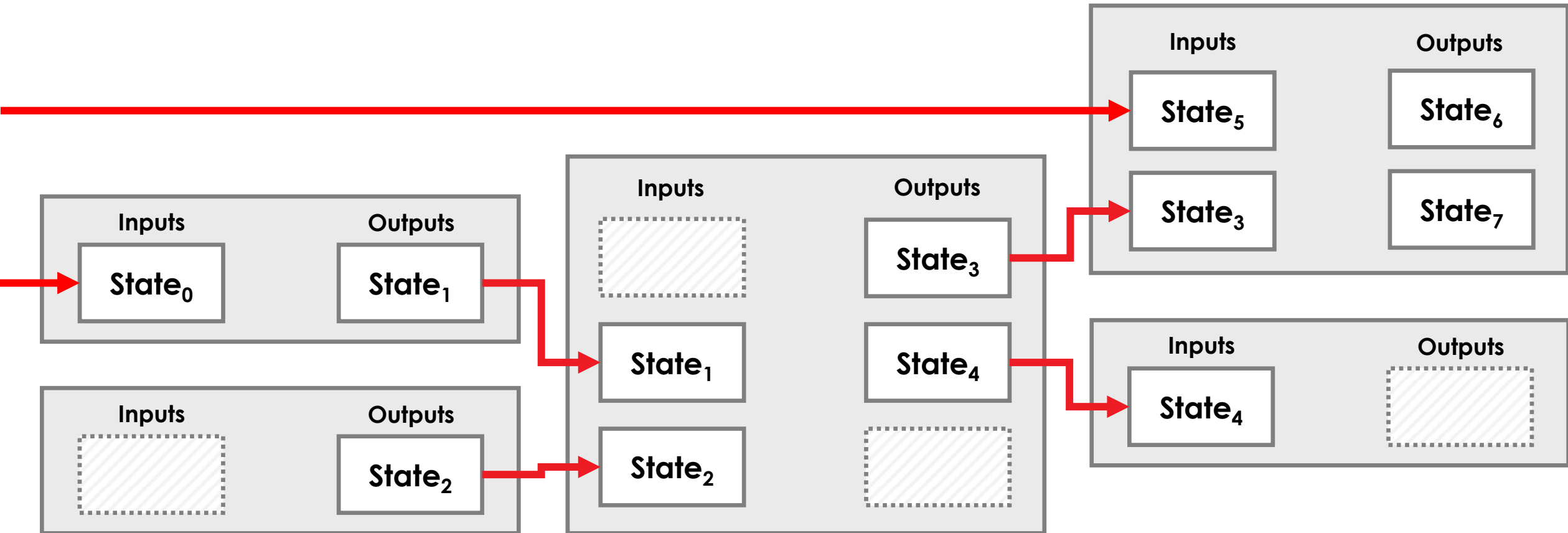
**Before**



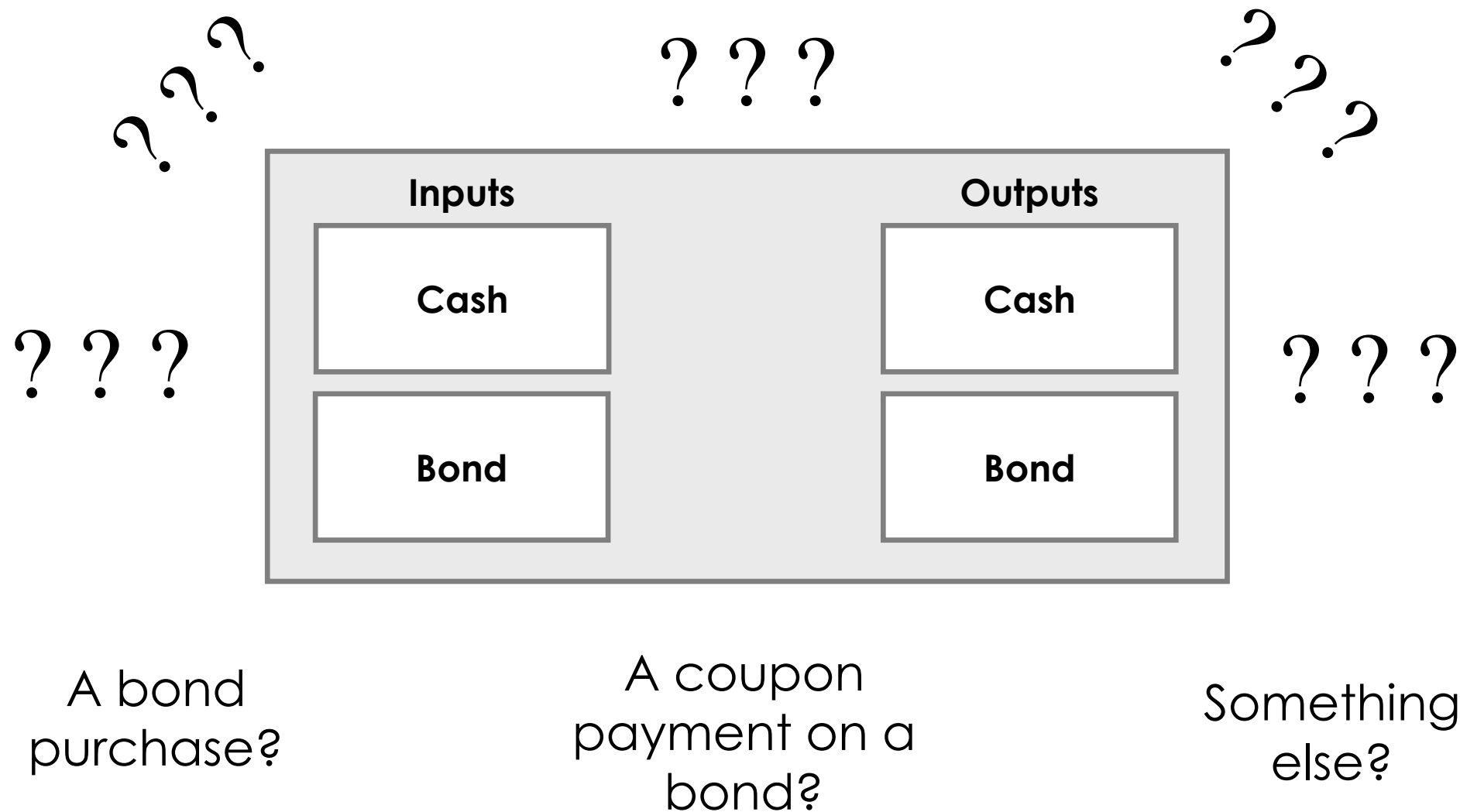
**After**



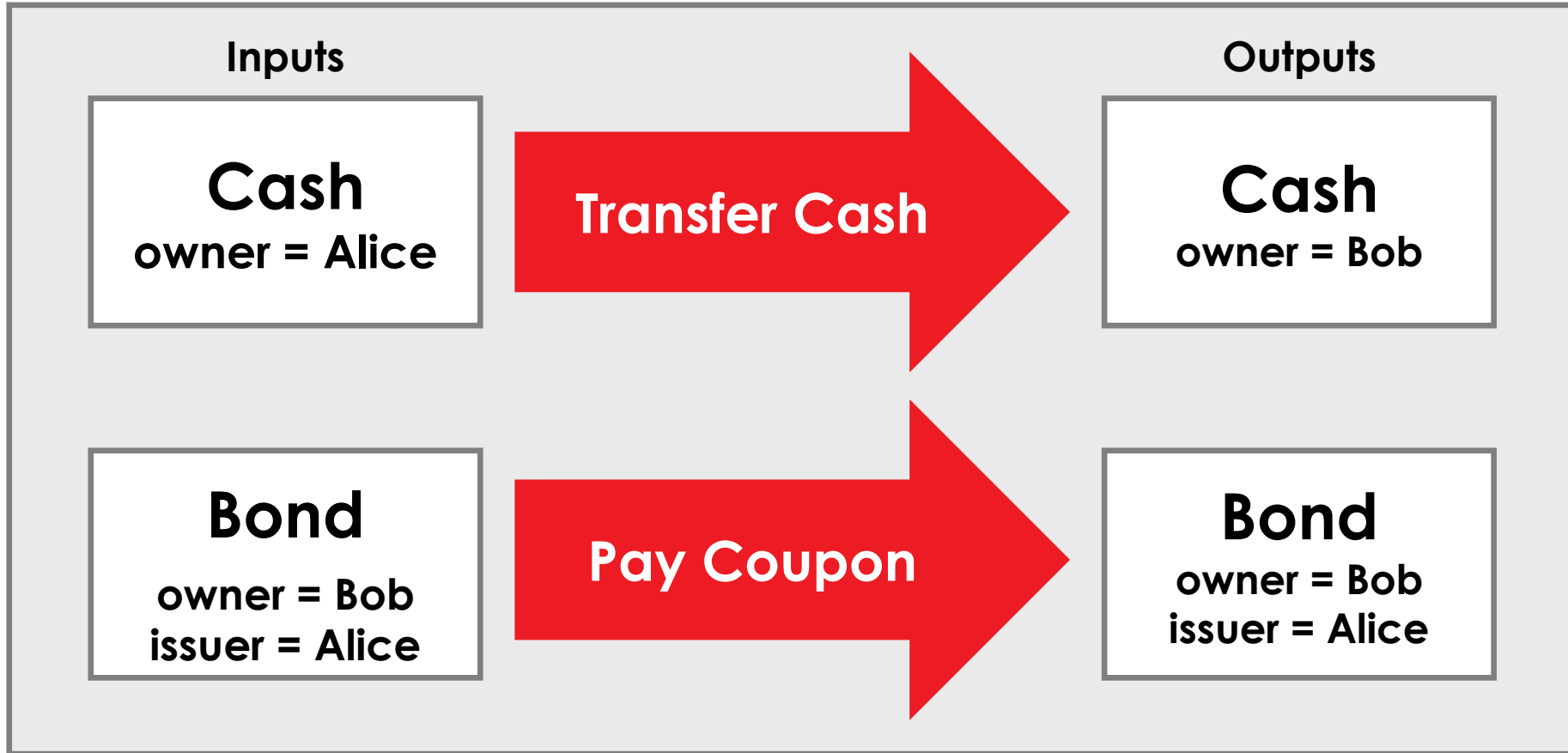
# Transaction outputs are inputs to future transactions



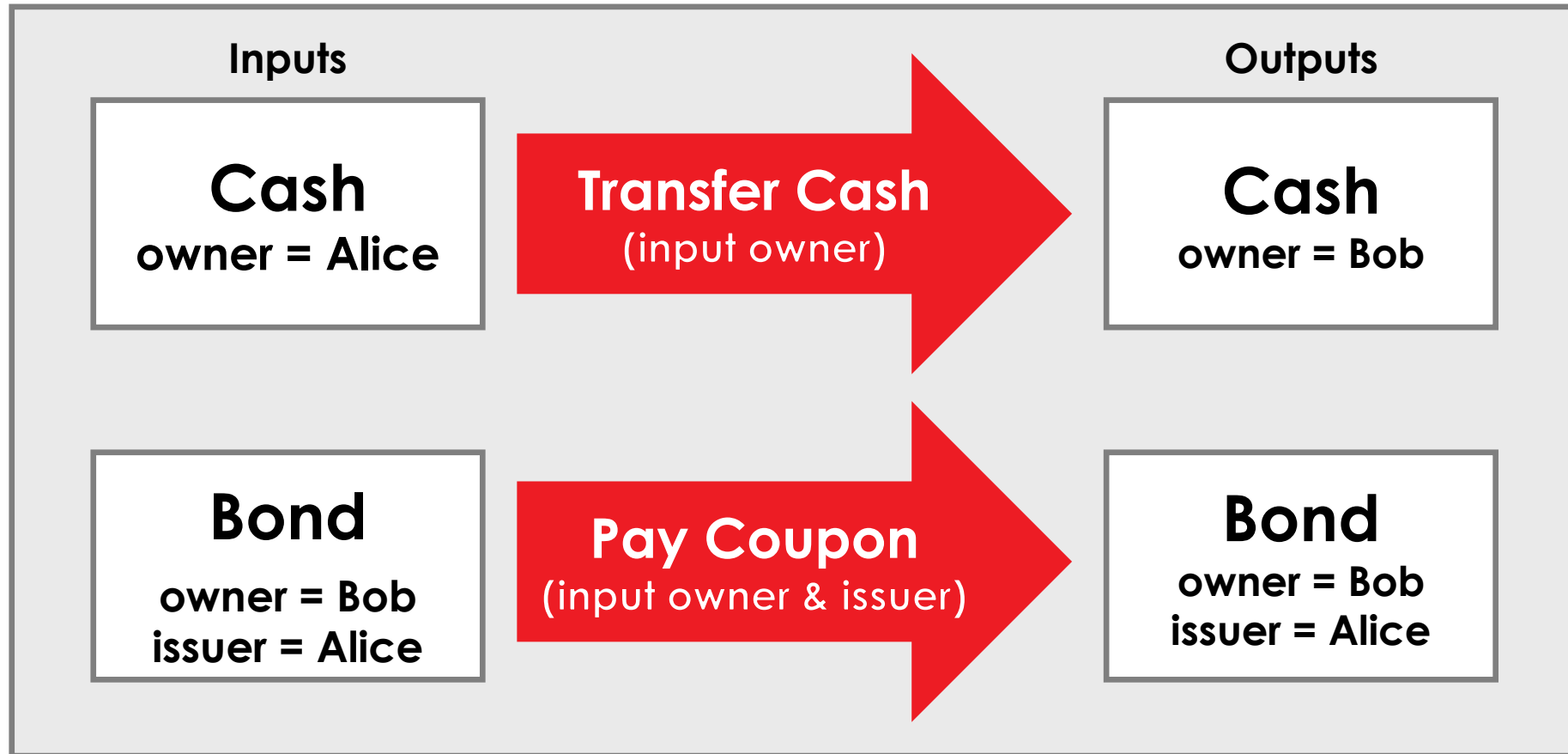
# Transactions can be ambiguous



**So transactions include commands to indicate intent**



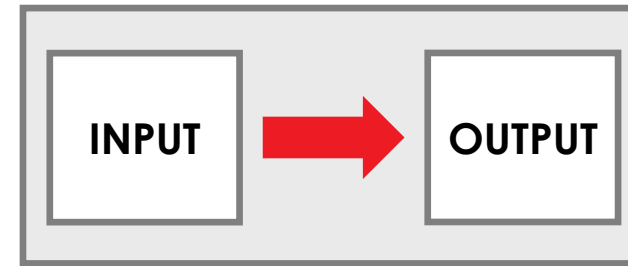
# Each command has associated signatures



# Corda Transaction API



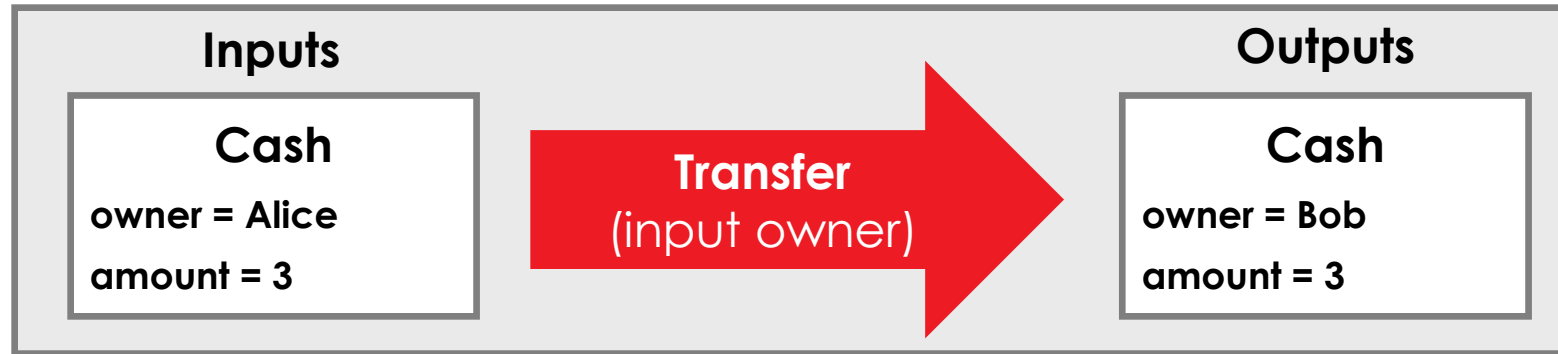
CommandWithParties



Ledger  
Transaction

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111

# Contracts decide if transaction proposals are valid



**Rule 1:** Has one cash input and one cash output



**Rule 2:** Has a transfer command



**Rule 3:** Cash in == cash out



**Rule 4:** Input cash's owner is a required signer



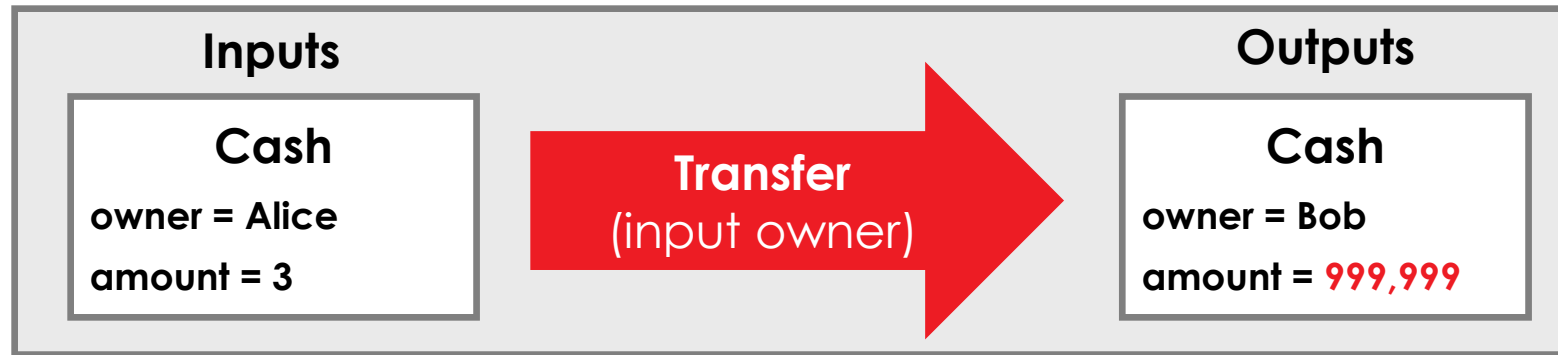
---

**Is transaction valid?**





# Contracts decide if transaction proposals are valid



**Rule 1:** Has one cash input and one cash output



**Rule 2:** Has a cash transfer command



**Rule 3:** Cash in == cash out



Contract  
Verification  
Exception

**Rule 4:** Input cash's owner is a required signer

---

**Is transaction valid?**



# Corda Contract API

Transaction has a cash input and a cash output	✓
Transaction has a cash transfer command	✓
The input and output cash have the same value	✗
The cash transfer command lists the input cash's owner as a required signer	✓
Is transaction valid?	✗

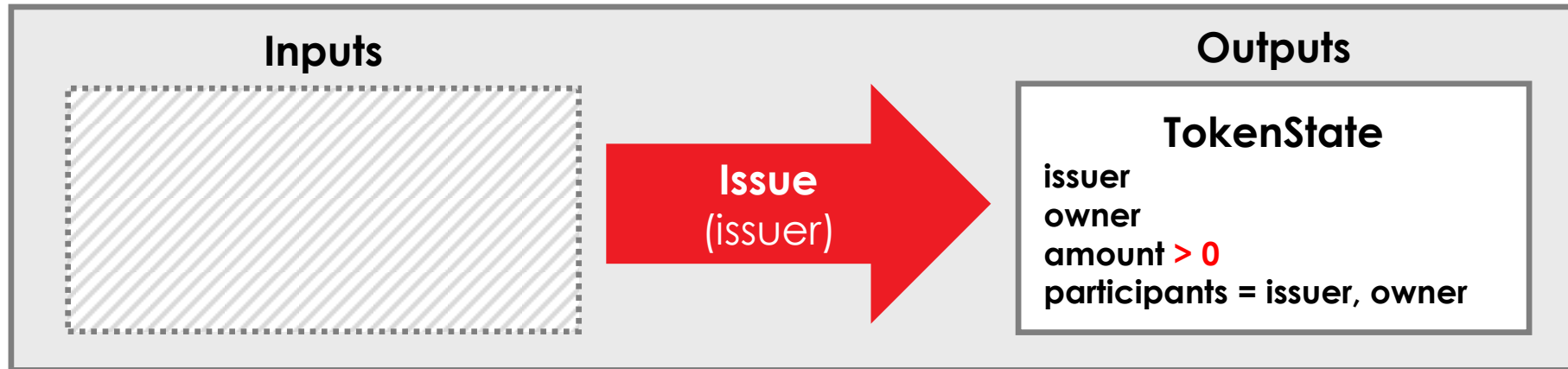


Contract

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111

# Our transaction: Token issuance

## Token issuance transaction



No inputs

One issue command,  
issuer is required signer

One output,  
amount is positive

# Our contract: the TokenContract

## TokenContract implements Contract

1. No inputs
2. One output
3. One command

4. Output is TokenState
5. Output's amount field is positive
6. Command is Issue

7. Issuer is required signer

1. Uncomment the tests in [src/test/java/java\\_bootcamp/ContractTests.java](#) one-by-one
  2. Modify **TokenContract** (defined in [src/main/java/java\\_bootcamp/TokenContract.java](#)) to make the tests pass
  3. You'll probably get a **java.io.NotSerializableException** the first time you run the tests. Check <https://github.com/corda/bootcamp-cordapp/wiki/Troubleshooting> to address it
- Refer to the wiki for resources, troubleshooting tips and solutions: <https://github.com/corda/bootcamp-cordapp/wiki>

# TokenContract

implements Contract

1. No inputs
2. One output
3. One command

4. Output is TokenState
5. Output's amount field is positive
6. Command is Issue

7. Issuer is required signer

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111



# Summary

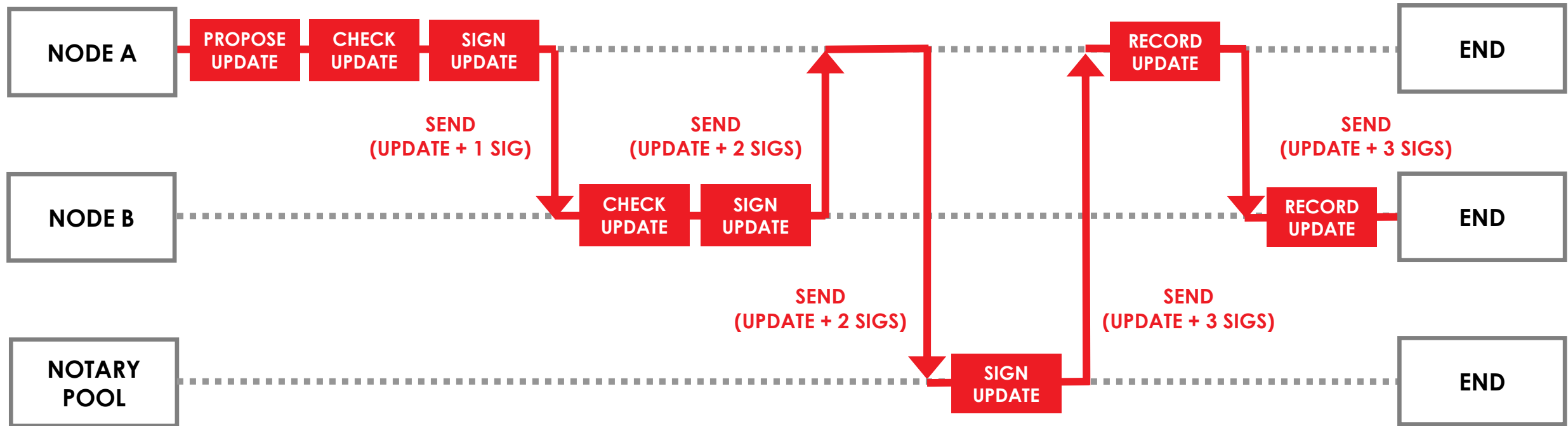
- Transactions consume old states and create new states
- Commands indicate their intent and required signers
- Contracts are classes implementing the Contract interface
- Contract classes should override `verify`, which should throw an exception if the transaction is invalid



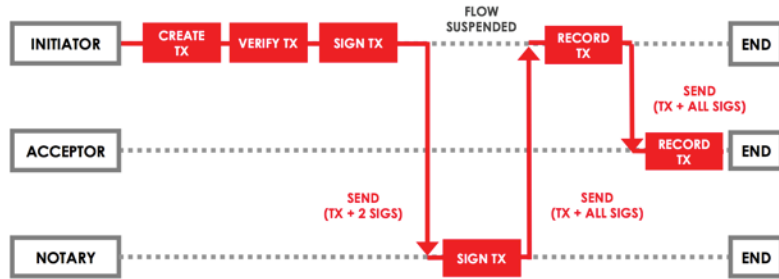


# Flows

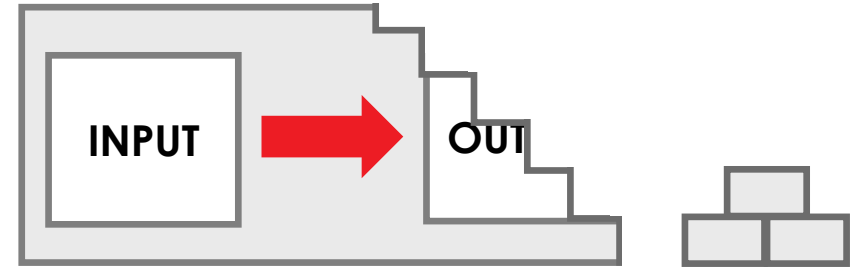
# Nodes run flows (mostly) to update the ledger



# Corda Flow API



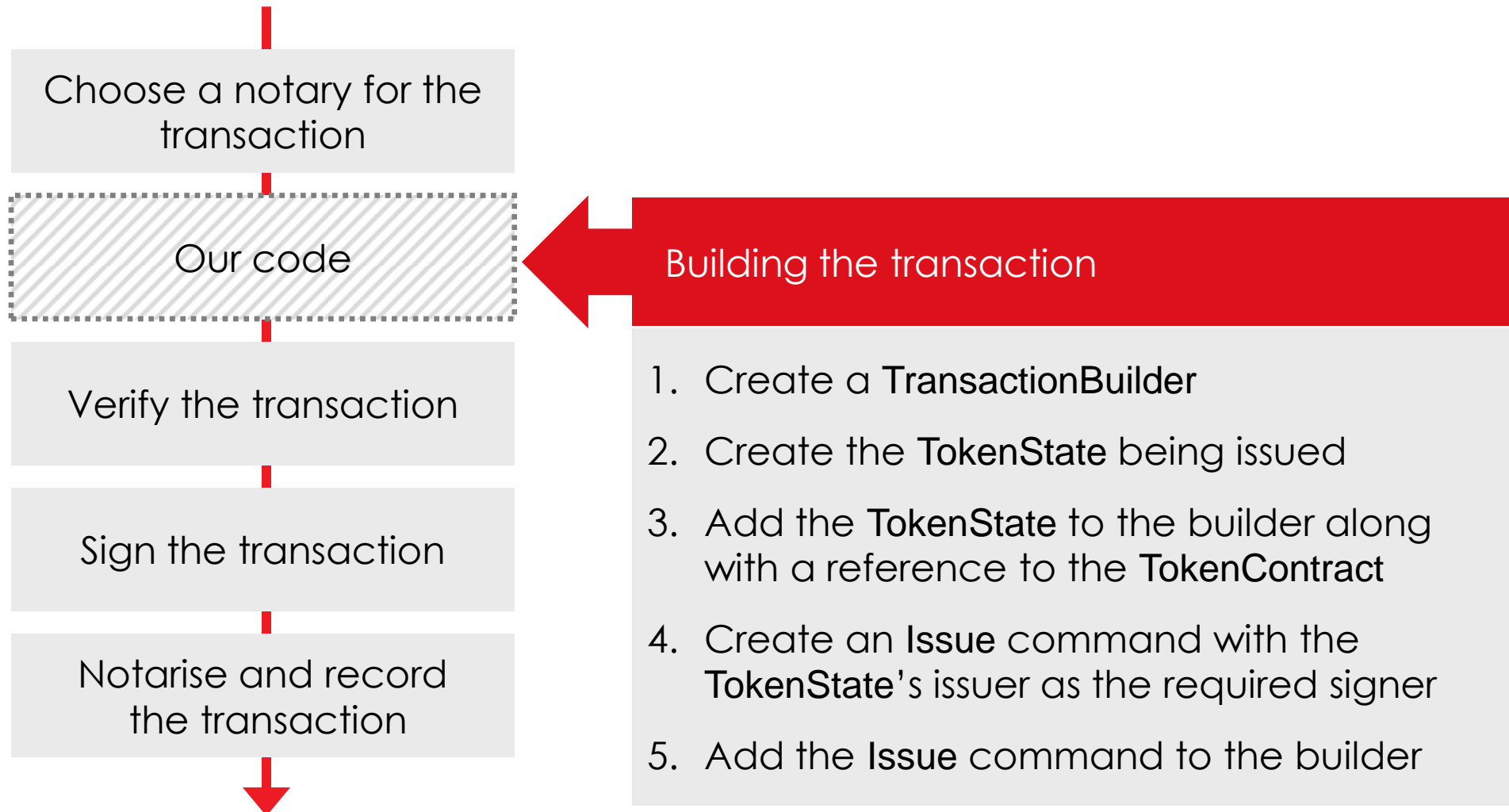
FlowLogic



Transaction  
Builder

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111

# Our flow: the TokenIssueFlow for issuing new tokens



1. Uncomment the tests in `src/test/java/java_bootcamp/FlowTests.java` one-by-one
  2. Modify `TokenIssueFlow` (defined in `src/main/java/java_bootcamp/TokenIssueFlow.java`) to make the tests pass
  3. You'll probably get a `java.io.IllegalStateException` the first time you run the tests. Check <https://github.com/corda/bootcamp-cordapp/wiki/Troubleshooting> to address it
- Refer to the wiki for resources, troubleshooting tips and solutions: <https://github.com/corda/bootcamp-cordapp/wiki>

### Building the transaction

1. Create a `TransactionBuilder`
2. Create the `TokenState` being issued
3. Add the `TokenState` to the builder along with a reference to the `TokenContract`
4. Create an `Issue` command with the `TokenState`'s issuer as the required signer
5. Add the `Issue` command to the builder

11100011 01111101 10000101 00001100 10111011 01111001 11101101 00100111 01101000 00011110  
01110010 11110000 10101111 11110001 00000100 00001100 00111001 01110000 10001010 11011101  
00011011 01001111 10001110 11000101 10000010 11110001 10100001 10001100 11001011 11110010  
00101000 01011011 11000000 00111100 11001011 10011000 00111000 11110110 11010010 10101101  
00001111 11001100 10000010 10110011 00101000 00101100 11001110 00000001 00111001 10111110  
10110111 11001000 00000110 01111000 01000010 11010100 01111101 00010100 01010010 01110011  
11000010 01110000 00011010 00000011 11010010 00000101 10001111 11000010 01000010 01110011  
11111100 00100110 01101101 10100101 00000000 01111000 10101011 01111100 11100010 00011000  
00101100 01101001 11011010 10101010 01101100 00000011 00010100 11110100 10111011 00100010  
00100000 00101011 11111101 11010011 00010000 11011100 10111100 10111001 01111011 11110111  
00010000 10010010 10000110 11000101 10110001 00010011 11001100 00000111 11000011 00001010  
00110111 10111110 11011101 10101101 11101111 11101000 10011110 01000110 01001000 11110001  
00000000 10001011 10110000 11110101 00101001 11001010 00111100 10110100 11101011 01000111  
10011101 11110001 10101110 11010110 01010110 00110010 00110010 00110101 11111110 10101110  
01011110 00110101 00111011 00111001 10100111 11100101 01110111 11100011 01010110 00001101  
01001011 01101101 00000100 01110010 10111011 11111010 10010100 10001000 00110101 11010010  
11011100 01001101 01000011 01111000 11010001 10001011 00110010 00101100 11111111 11110000  
00001000 01011101 10010011 11110000 11110010 10100110 01010101 10111110 00010001 01100000  
00110111 01110100 10001111 10000000 01000101 11111000 00110110 11010111 11000111 11100000  
11010011 01000010 10100111 10100100 01111000 11010100 01010111 10010011 11011101 00011010  
11101111 11110010 00011001 10000011 01100100 00101101 10000111



# Summary

- Flows automate the process of updating the ledger
- Flows are classes implementing the FlowLogic interface
- FlowLogic classes should override `call`, which defines the flow's steps
- TransactionBuilder is used to construct transactions



The background of the slide features a dark, textured surface with a faint grid of small dots. Overlaid on this are several interconnected clusters of nodes. Each node is represented by a small red circle, and they are connected by thin, light-colored lines. Some clusters are more densely connected than others, forming a complex, web-like structure that spans the width of the slide.

# Running our CorDapp

```

task deployNodesJava(type: net.corda.plugins.Cordform, dependsOn: ['jar']) {
    directory "./build/nodes"
    node {
        name "O=Notary,L=London,C=GB"
        notary = [validating : false]
        p2pPort 10002
        cordapps = [
            "$project.group:cordapp:$project.version"
        ]
    }

    node {
        name "O=PartyA,L=London,C=GB"
        p2pPort 10005
        rpcSettings {
            address("localhost:10004")
            adminAddress("localhost:10005")
        }
        cordapps = [
            "$project.group:cordapp:$project.version"
        ]
        rpcUsers = [[ user: "user1", password: "test", permissions: ["ALL"]]]
    }

    ...
}

```

1. Build a test network of nodes by opening a terminal window at the root of your project and running the following command:
  - Windows: `gradlew.bat deployNodesJava`
  - macOS: `./gradlew deployNodesJava`
2. Start the nodes by running the following command:
  - Windows: `build\nodes\runnodes.bat`
  - macOS: `build/nodes/runnodes`
3. Wait for the nodes to start, then go to the terminal of Party A (not the notary!) and run the following command to issue 99 tokens to Party B:  
  
`flow start TokenIssueFlow owner: PartyB, amount: 99`
4. You can now see the tokens in the vaults of nodes A and B (but not of node C!) by running the following command in the nodes' terminals:

`run vaultQuery contractStateType: java_bootcamp.TokenState`

A dark blue background with a network diagram. Red dots representing nodes are connected by thin white lines, forming a complex web. The nodes are distributed across the frame, with some clusters and some isolated nodes. The overall effect is a technical, digital aesthetic.

# Summary

Corda is a unique blockchain platform offering:

- Legally-identifiable counterparties
- Need-to-know data distribution
- Ease-of-use

While maintaining the benefits of traditional blockchains

# **Corda Private Coaching Sessions**

Private tutoring sessions for blockchain design and development -  
Weekly and monthly plans

Blockchain cybersecurity- Private tutoring sessions

Blockchain development with Corda R3- Private tutoring sessions

# More Blockchain Self-Paced Courses

- [Hyperledger Fabric for System Admins](#)
- [Learn Solidity Programming by Examples](#)
- [Introduction to Ethereum Blockchain Development](#)
- [Intro to Blockchain Cybersecurity](#)
- [Intro to Hyperledger Sawtooth for System Admins](#)

# Live Blockchain Courses

- [Live and self-paced blockchain development with Ethereum](#)
- [Live and self-paced blockchain development with Hyperledger Fabric](#)
- [Live and self-paced blockchain development with Corda](#)
- [Immersive Blockchain Bootcamp with live and self-paced courses](#)

# Resources

- The docs  
([docs.corda.net](https://docs.corda.net))
- Stack Overflow, corda tag  
([stackoverflow.com/questions/tagged/corda](https://stackoverflow.com/questions/tagged/corda))
- The cordaledger Slack  
([slack.corda.net](https://slack.corda.net))





coding-bootcamps.com

Thank you



**Coding**  
Bootcamps