Module 4

Designing for Corda

Learning outcomes

- Learn how to map a real-world process to a CorDapp
- Design our own CorDapp, which we'll then implement

Designing for Corda

General process for creating CorDapps

1. Objective
2. Scenario and parties (and any required assumptions)
3. The data being agreed upon (**states**)
4. How the data evolves over time (**contracts**, **transactions**)
   - Define what transactions are required
   - Define the contracts such that they only allow these transactions
5. How the parties reach consensus on proposed evolutions (**flows**)
   - What data needs to be shared and when?

Objective

- What are you **aiming to achieve** with your CorDapp?
- **Example:** Let's design a CorDapp that is intended to:

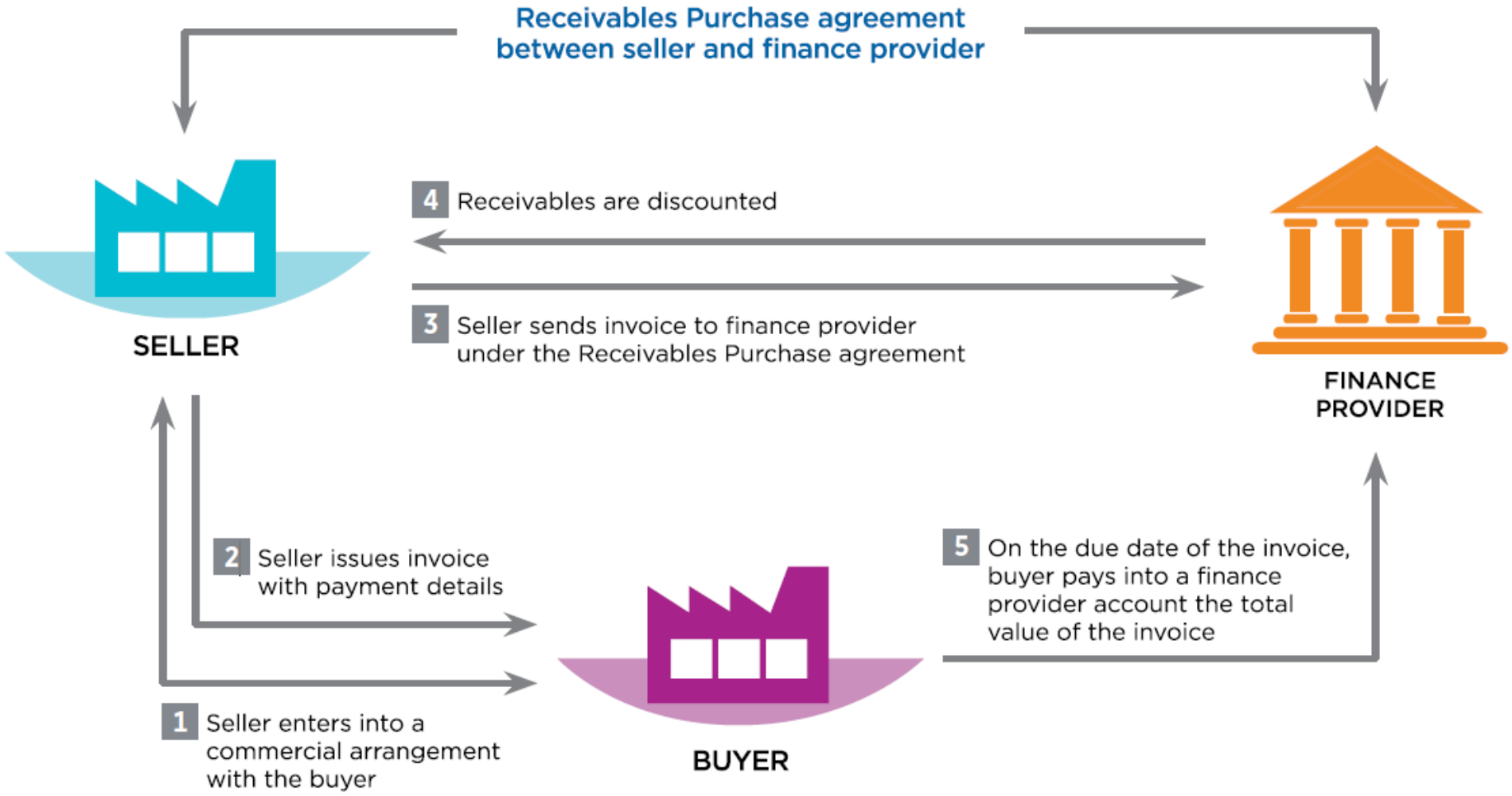
Model the end-to-end process for an account receivable (AR) purchase

Scenario and parties

Which parties are involved?

- Start by asking yourself the following questions:
  - What is being done?
  - Who is it being done by?
    - Buyer, seller, broker, merchant, bank, central bank, etc.
    - Consider participants at the entity-wide level
- Examples:
  - Purchasing cycle
  - Mortgage applications
  - OTC derivatives

Accounts receivable financing



Source: Global SCF Forum

Scenario and parties

The AR scenario involves three parties:

- **Seller** – The party who sold the goods to a buyer, and is now looking to sell the resulting accounts receivable
- **Buyer** – The buyer of goods from the seller (also known as the **account debtor**)
- **Bank** – The buyer of the AR, which entitles them to receive payment from the buyer of the goods

Assumptions

For the sake of simplicity, we will make some assumptions:

1. There is already an invoice discounting agreement in place between the seller and the bank
2. The scenario will begin from the point of the seller invoicing the buyer (i.e. no purchase orders)
3. We will ignore the bank's fees
4. The account receivable is assignable (i.e. can be sold on)
5. The buyer and seller already have an business relationship

States

Which data items need to be stored on ledger?

- Next, we need to establish which states are required
- Ask yourself:
  - What data do we need to track?
    - Split these into agreements and assets
  - What fields/properties do these data items have?
- Examples of states:
  - **Option** (instrument, strike price, expiry date, seller, buyer)
  - **Invoice** (items and prices, delivery date, payment date, terms)
  - **Bond** (coupon, redemption date, issuer, current owner)

States

For our accounts-receivable example:

- **Agreements required:**
  - **Invoice** (buyer, seller, items, settlement date, terms, etc.),
    which is an agreement between the buyer and the seller
  - **Account Receivable** (amount, seller, buyer, currency, etc.),
    which is a credit agreement between the seller and bank
- **Assets required:**
  - Cash (issuer, currency, amount, owner),
    which is a financial asset that has an issuer and current owner

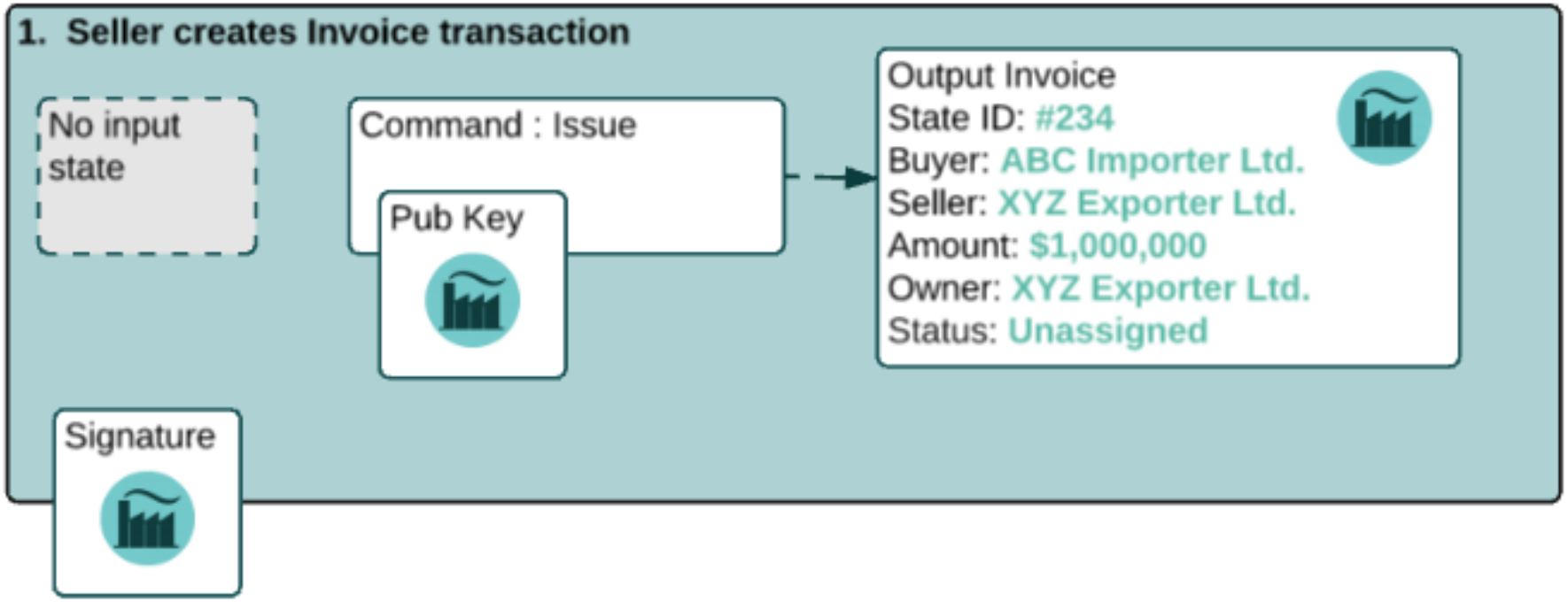
Transactions

Given the contract rules, which evolutions of the data can occur?

- Next, we need to work out which data evolutions are required
- Ask yourself:
  - How many stages are there in your scenario?
    - Typically, each stage corresponds to a transaction
  - Which parties are involved in each stage?
  - What documentation, contracts or financial instruments are involved at each stage?

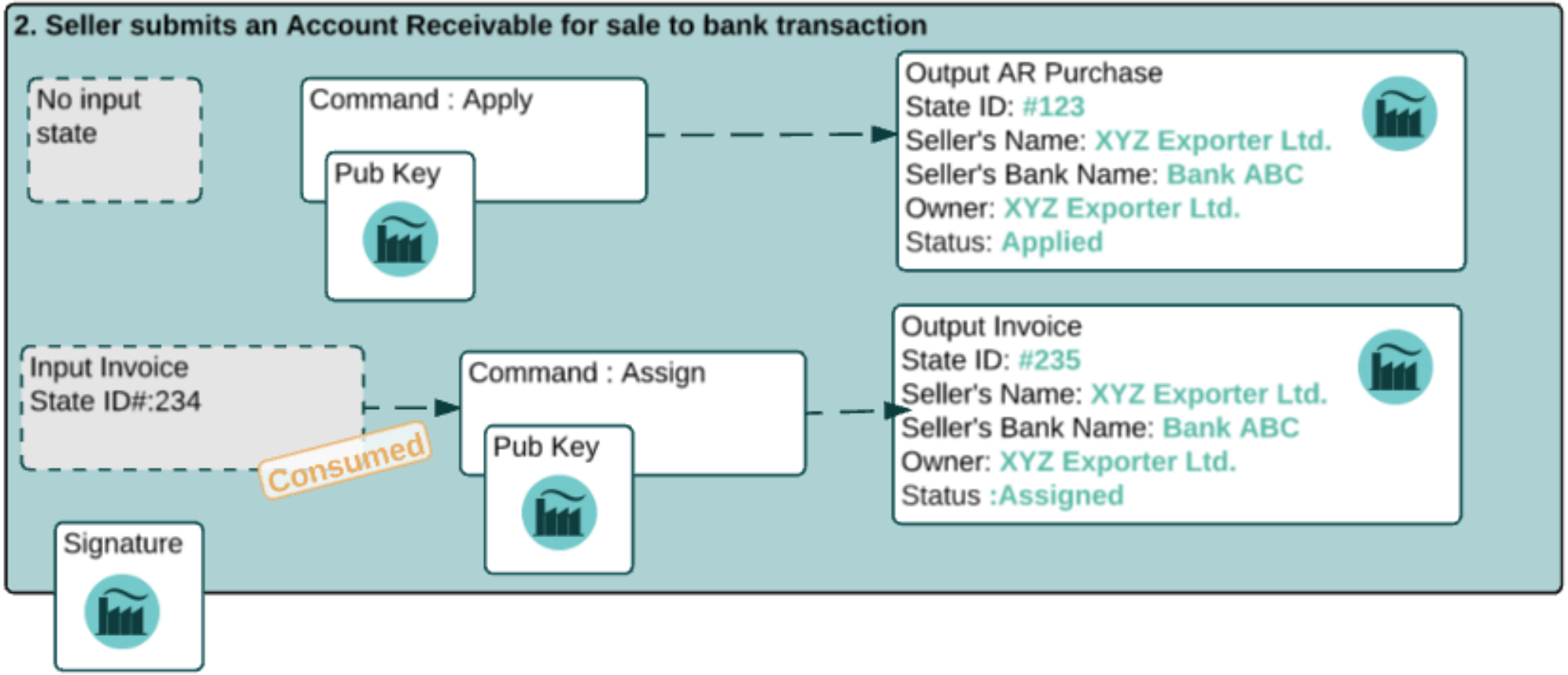
How do the states evolve?

The seller issues an invoice to the buyer.



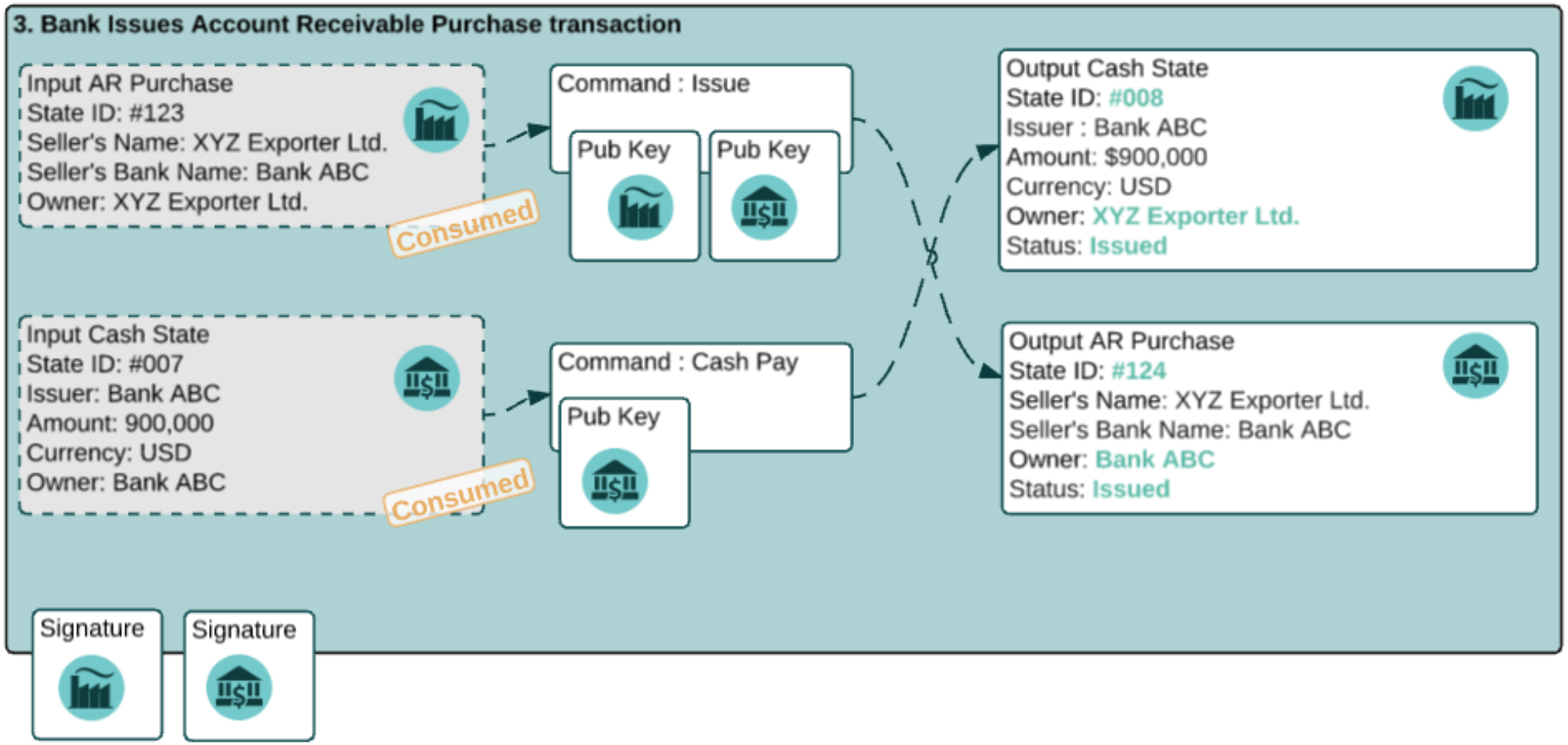
How do the states evolve?

The seller applies to the bank to sell the account receivable.



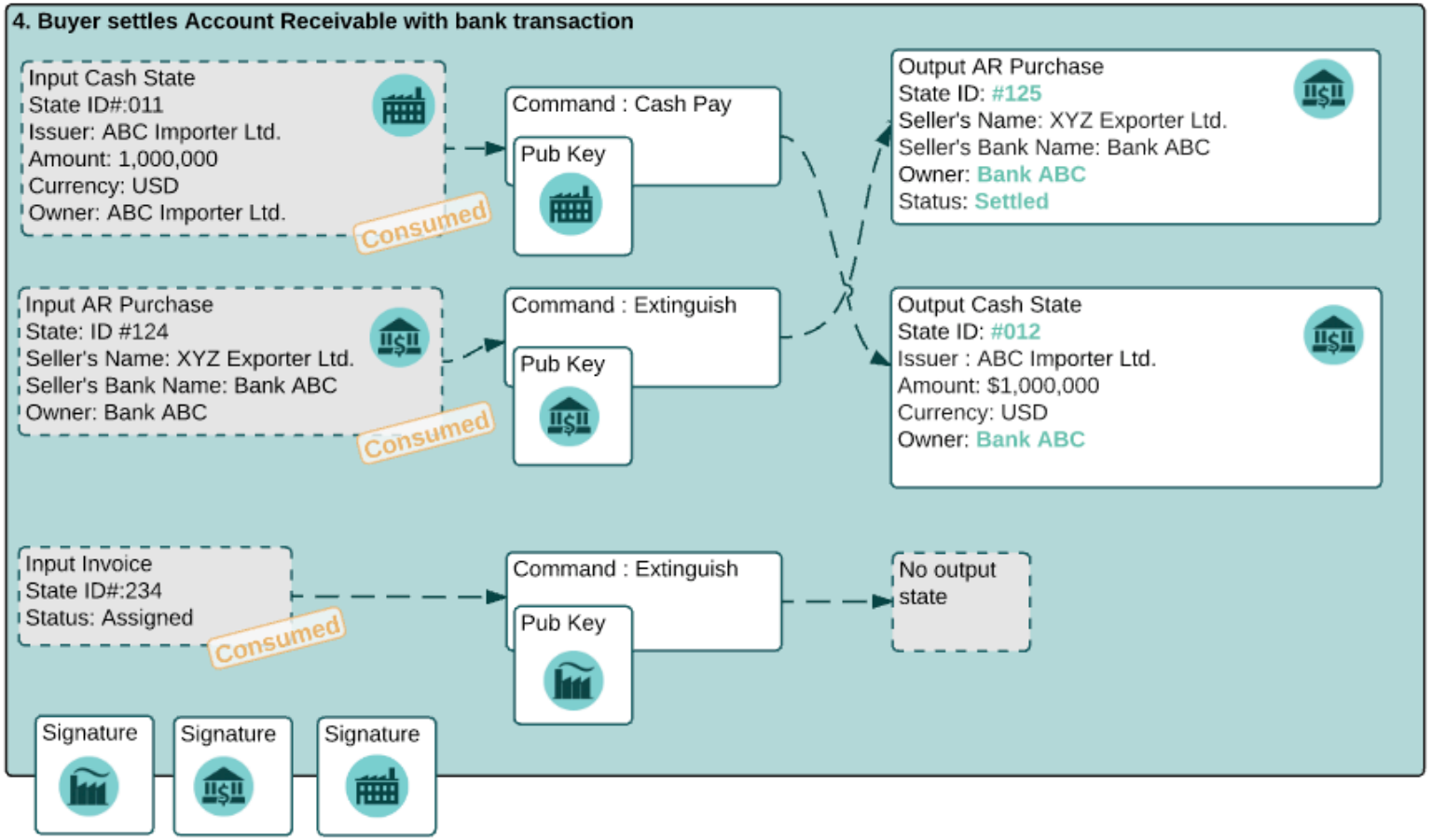
How do the states evolve?

The bank purchases the AR and provides the seller some cash in return.



How do the states evolve?

The buyer pays the bank. The invoice and the AR are settled.

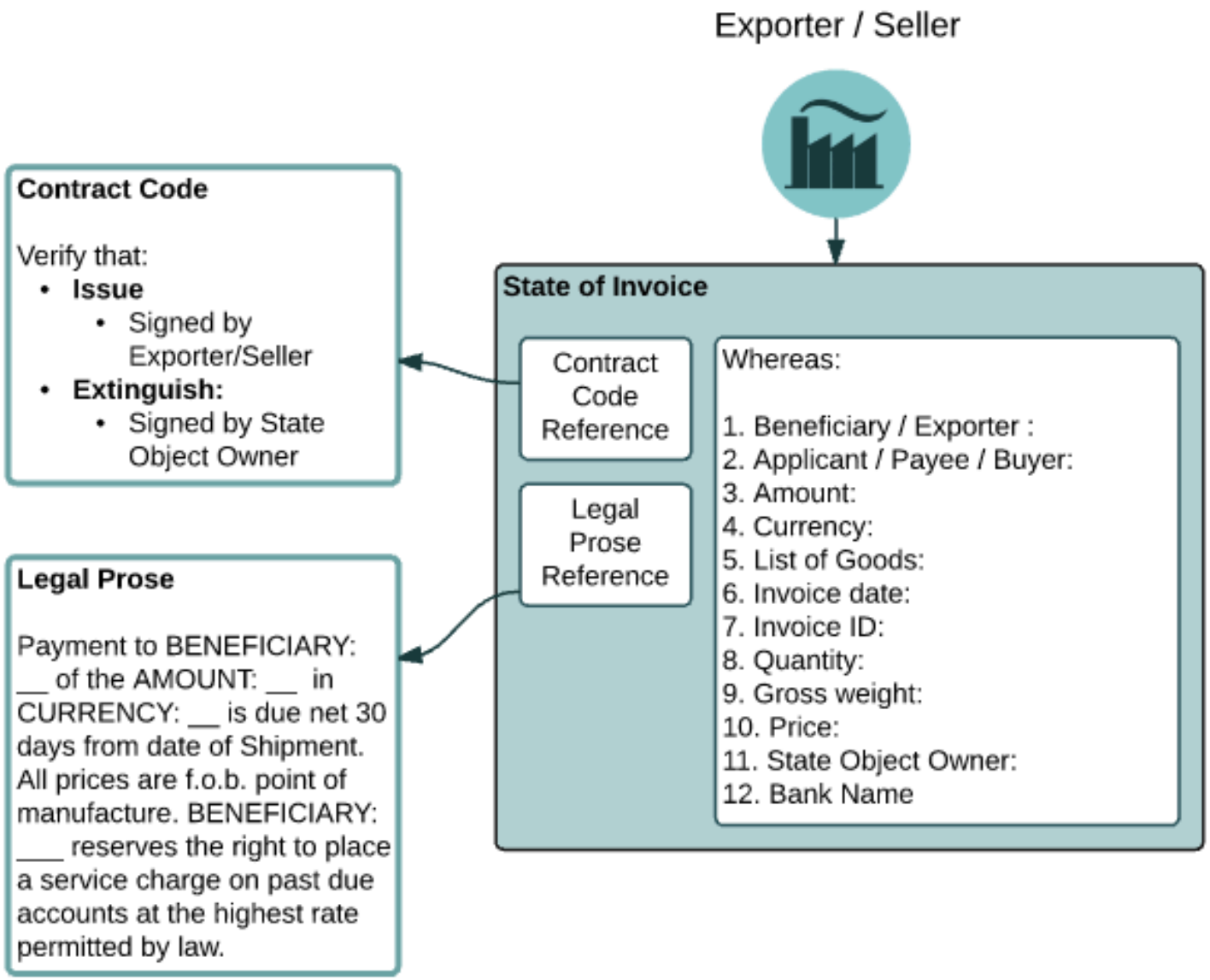


Contracts

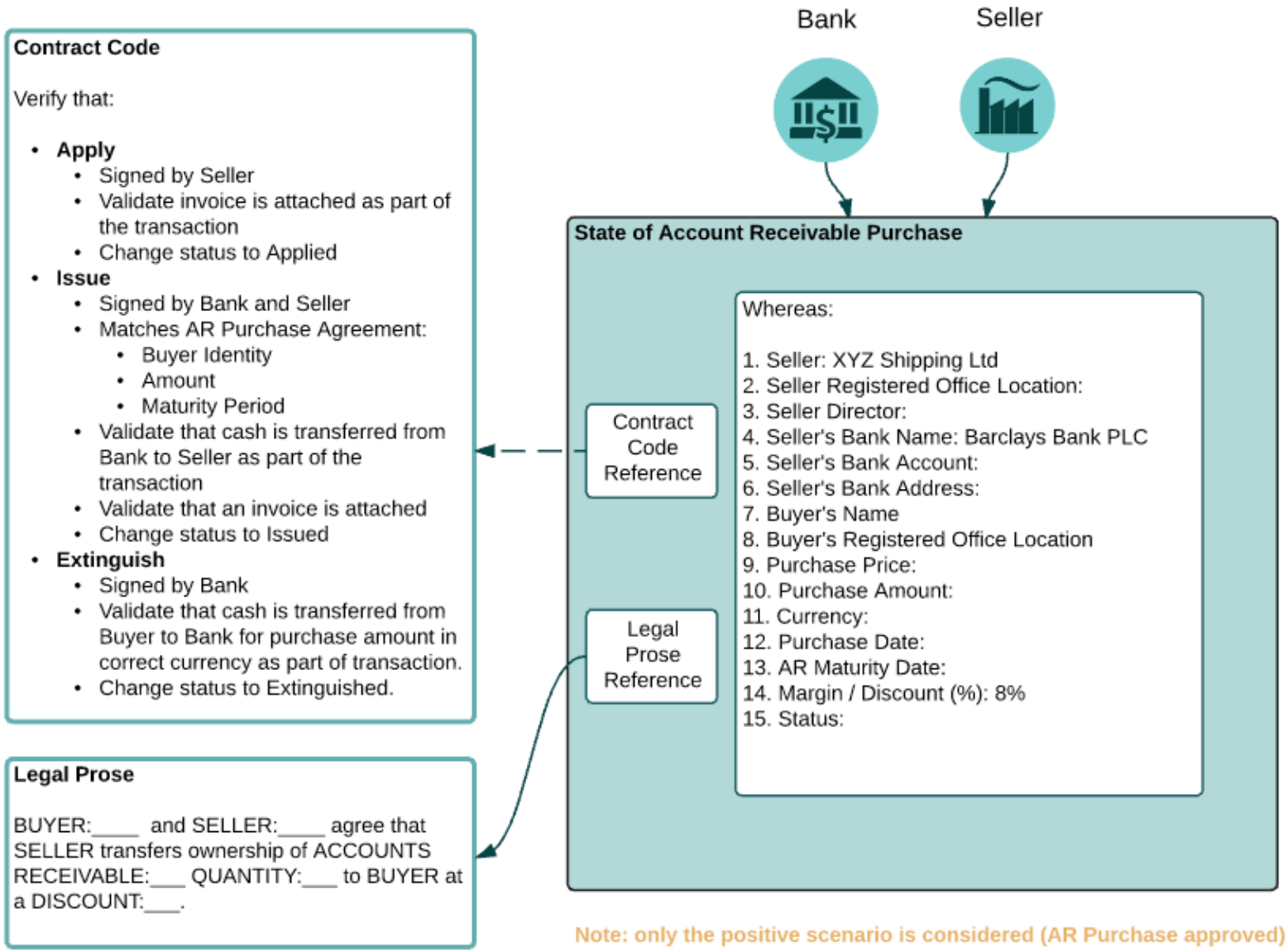
Which evolutions of the data are acceptable?

- Next, we need to create contract rules that guarantee that these transactions, and only these transactions, are possible
- Ask yourself:
  - What actions can be performed on our "data"/states at each point in their life-cycle?
  - Who needs to approve each action?
  - How should the data evolve for each action?

Invoice

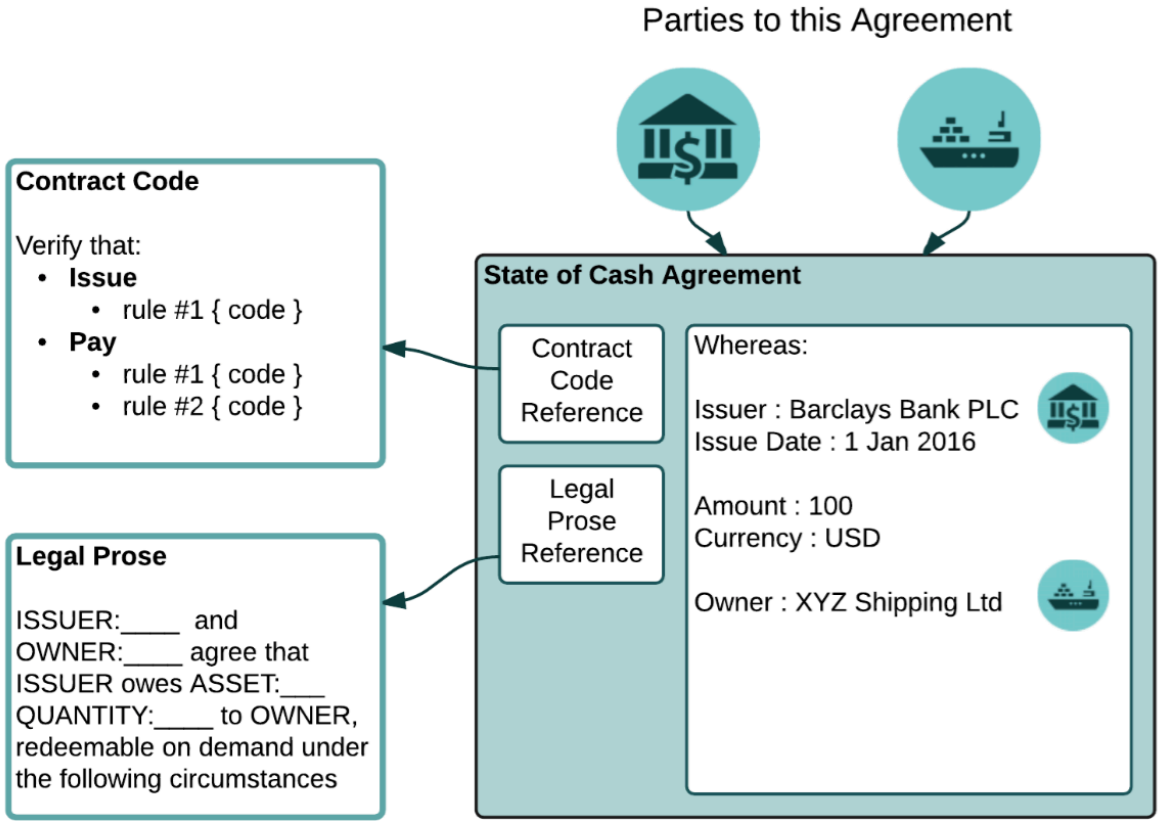

Exporter / Seller
Contract Code
Verify that:
Issue
Signed by Exporter/Seller
Extinguish:
Signed by State Object Owner
Legal Prose
Payment to BENEFICIARY: __ of the AMOUNT: __ in CURRENCY: __ is due net 30 days from date of Shipment. All prices are f.o.b. point of manufacture. BENEFICIARY: ___ reserves the right to place a service charge on past due accounts at the highest rate permitted by law.
State of Invoice
Contract Code Reference
Legal Prose Reference
Whereas:
1. Beneficiary / Exporter :
2. Applicant / Payee / Buyer:
3. Amount:
4. Currency:
5. List of Goods:
6. Invoice date:
7. Invoice ID:
8. Quantity:
9. Gross weight:
10. Price:
11. State Object Owner:
12. Bank Name

Accounts receivable



Note: only the positive scenario is considered (AR Purchase approved)

Cash


Parties to this Agreement
Contract Code
Verify that:
Issue
rule #1 { code }
Pay
rule #1 { code }
rule #2 { code }
Legal Prose
ISSUER:____ and
OWNER:____ agree that
ISSUER owes ASSET:___
QUANTITY:____ to OWNER,
redeemable on demand under
the following circumstances
State of Cash Agreement
Contract Code Reference
Legal Prose Reference
Whereas:
Issuer : Barclays Bank PLC
Issue Date : 1 Jan 2016
Amount : 100
Currency : USD
Owner : XYZ Shipping Ltd

Corda-specific considerations

- Notary service
  - Designate one node as the notary
- Network map service
  - Designate one node as the network map service
- Communicating with your nodes
  - Covered in the practical sessions
- Testing
  - Covered in the practical sessions

Summary

- Define your scenario - who's involved and their interactions
- Define your states – what data objects need to be evolved
- Define the transactions which need to take place to evolve these states
- Define contracts that limit the evolution of data to only these transactions
- Build the flows that implement the interactions required to reach consensus

Designing Our CorDapp

Building a CorDapp

- The remainder of this course will focus on the development of a CorDapp from (almost) scratch
- Once completed, you will be in a good position to build your own CorDapps

Key for practical sections

- Brainstorming

- Instructions/Coding Exercises

- Solutions

Domain

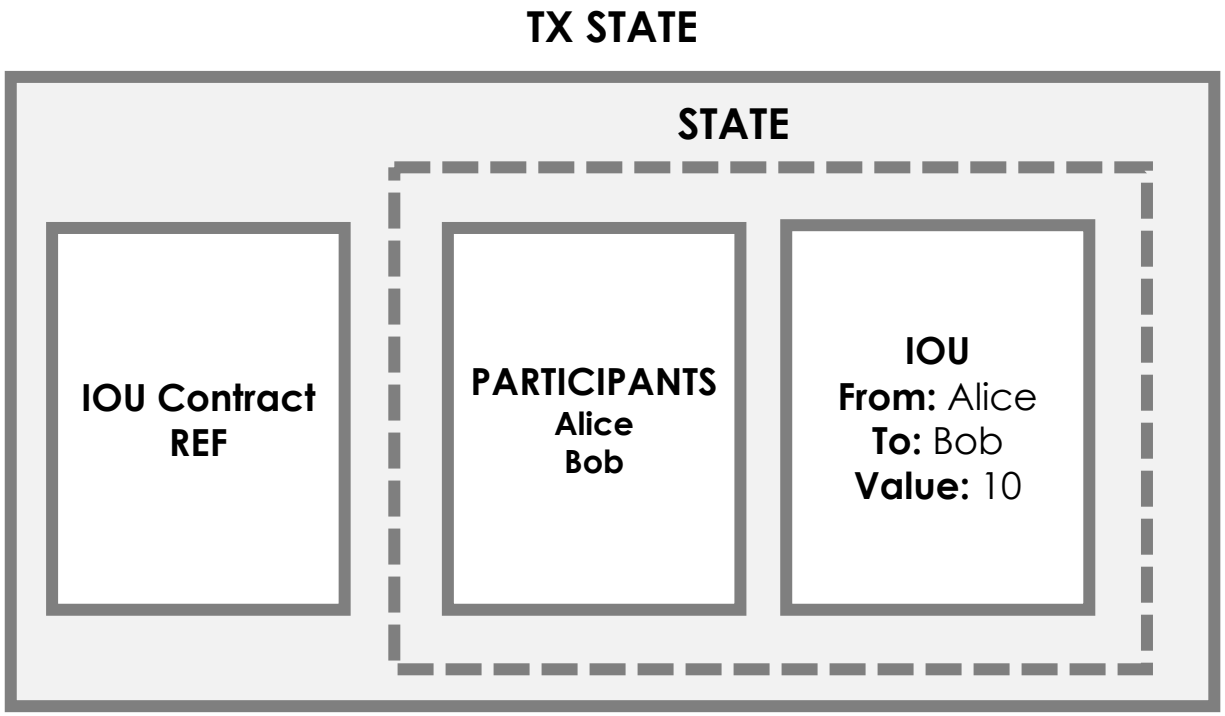
- The CorDapp we develop will allow parties to create and send IOUs (**I O**we yo**U**s)
- An IOU records that Alice owes Bob an amount (e.g. £5

Design brainstorm

- **How would we model an IOU on Corda?**
- Think in terms of the basic Corda elements:
  - States
  - Contracts
  - Transactions
- We'll design the flow later

IOU State diagram

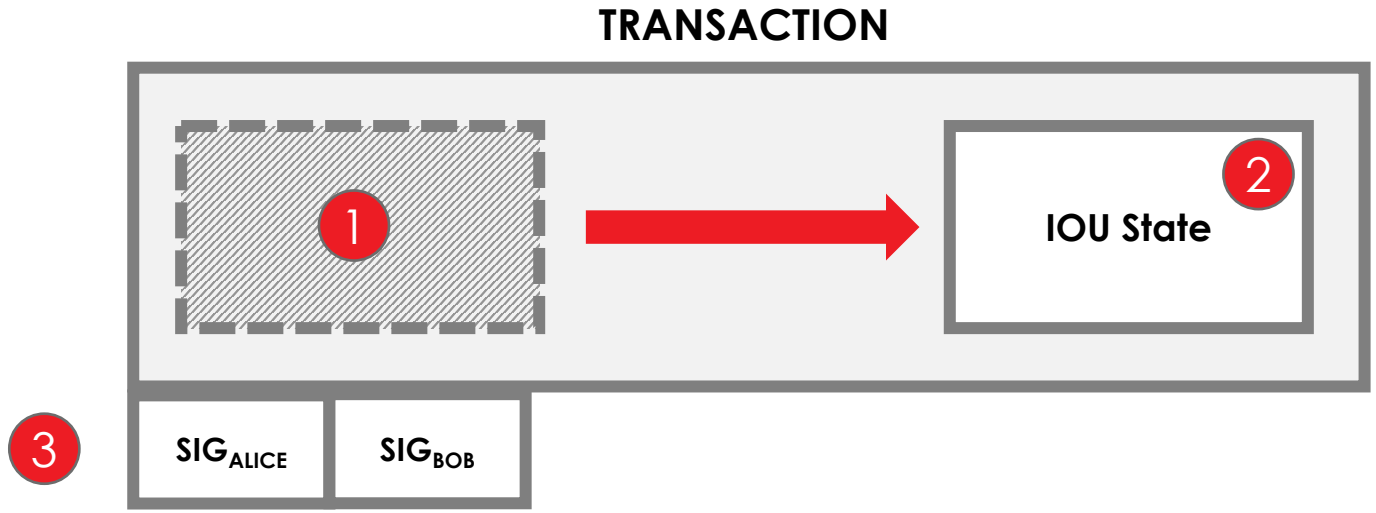
- The `IOUState` has a sender, a recipient, and a value
- In an `IOUState`, the `participants` – the parties able to consume the state - are the sender and the recipient



IOU Creation Transaction diagram

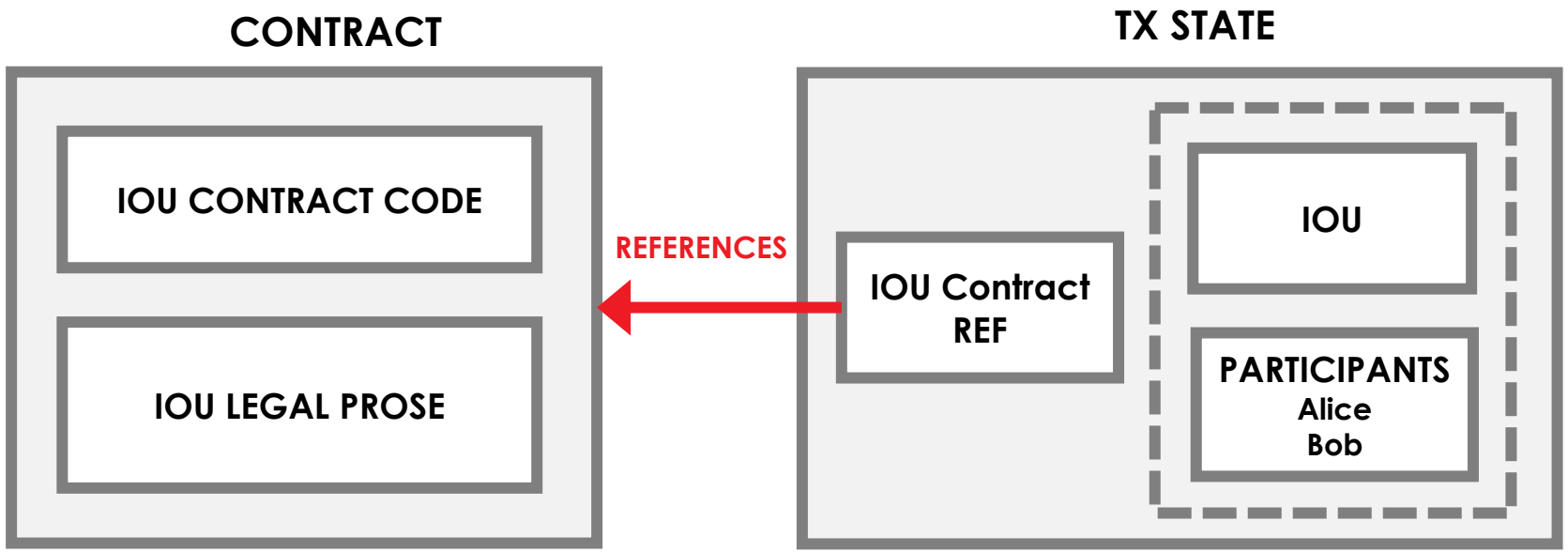
- A valid IOU transaction should have:

1. No inputs
2. One output
3. Signatures from both the sender and the recipient



IOU Contract diagram

- `IOUState`, when added to a transaction, points to an `IOUContract`, which imposes constraints



Approach

- We will build our CorDapp in four steps:
  1. The state, modelling the IOU and its possible evolutions
  2. The contract, governing the IOUs evolutions over time
  3. The flow, choreographing the agreement of the IOU between nodes
  4. The API, enabling the user to communicate with their node

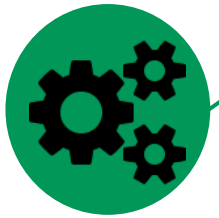
- We will use a Test Driven Development approach:
  - Test, Fail, Write, Pass
  - Corda provides built-in DSLs (domain-specific languages) for testing
  - We can test our CorDapp without every deploying it

Training repos

- We'll be building our CorDapp from a simple template:
  https://github.com/roger3cev/corda-training-template

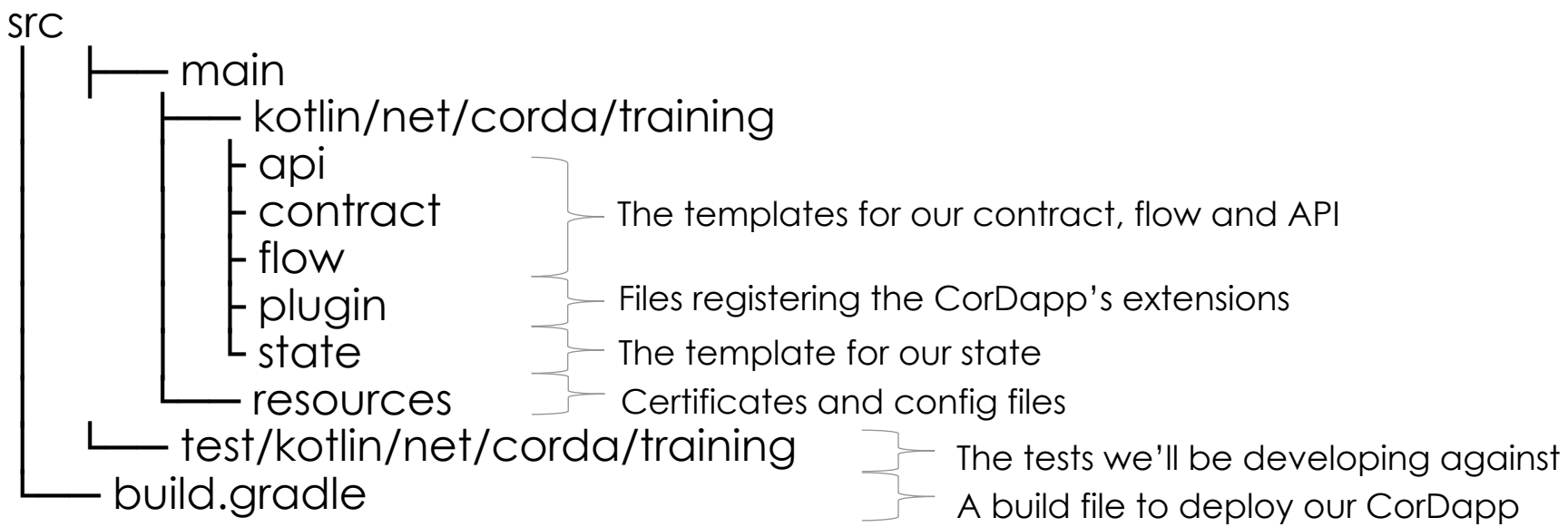
- Solutions for each step are in a work-along repo:
  https://github.com/roger3cev/corda-training-solutions

Training repos - Instructions

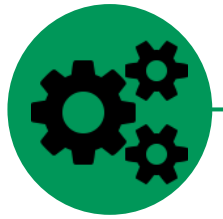


Goal	Clone the template and solutions repos for this section
Where?	N/A
Steps	1. Clone the template and the solutions repositories to your local machine: 2. If you have git installed, you can do this from the command line, using `git clone [repoName]`
Key Docs	N/A

Repo layout



Opening the project in IDEA - Instructions



Goal	Open the template repository in IDEA
Where?	N/A
Steps	1. Open IDEA 2. On the splash screen, click "Open" and point the dialog your local cordapp-training-template repository • Do NOT click "Import Project", which will delete our Run Configurations 3. Once you've opened the project, a tool window should appear in the bottom-right, asking you to "Import Gradle Project" – click this link to build the project
Key Docs	https://docs.corda.net/getting-set-up.html

Checkpoint – Progress So Far

Our progress so far

- We now have a basic design for our IOU CorDapp
- Going forward, modules will be split into:
  - **Theory** sections - covering Corda's key APIs
  - **Practical** sections - using the APIs to build the IOU CorDapp