

# Learning objectives

- Understand that Corda has been designed to solve a specific business problem for regulated financial institutions and this is reflected in its architecture
- 2. Understand how to think "the Corda way"
- 3. Understand the key concepts which underpin how Corda works

## **Agenda**

- 1. The genesis of Corda
- 2. The Corda way of thinking
- 3. Corda key concepts

### Recommended reading:

- Corda Non-Technical Whitepaper
- Corda Technical Whitepaper
- The Corda Way of Thinking blogpost
- https://docs.corda.net/key-concepts.html



## The genesis of Corda

- 1. Why build Corda?
- 2. Design rationale



## Mission objective

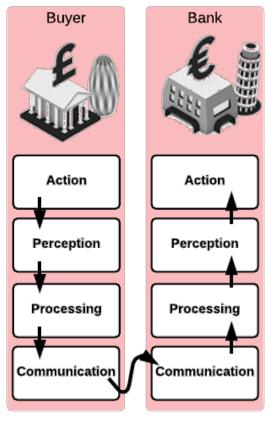
Requirements driven design

To establish the architecture for an **open**, **enterprise-grade**, **shared** platform for the **immutable** recording of financial events and execution of logic.

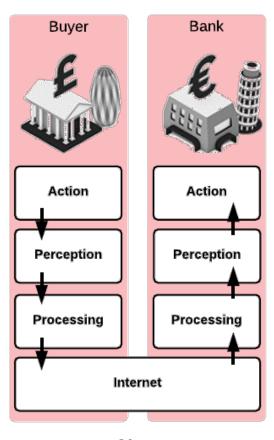
**r**3

## **Back to basics**

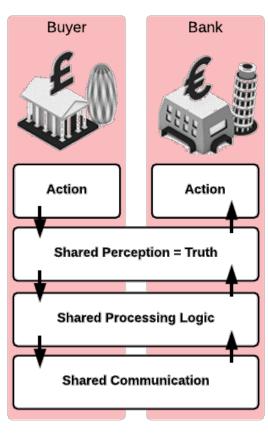
What problem are we trying to solve with distributed ledger technology?



Before the Internet



Now



After Shared Ledgers

**r**3

# Defining characteristics of distributed ledgers

Distributed ledgers are systems that enable parties who don't fully trust each other to form and maintain consensus about the existence, status and evolution of a set of shared facts.

**r**3.

9

## Introducing Corda

Corda is a **distributed ledger platform** designed and built from the ground up to record, manage and synchronise agreements (legal contracts), designed for use by regulated financial institutions





# Managing trade-offs

Designing the architecture of a distributed ledger platform is rooted in **managing a set of trade-offs**, primarily:

- Privacy and confidentiality
- Scalability
- Security
- Complexity
- Among others...

## You cannot have your cake and eat it!

- Corda maintains the optimal balance of trade-offs for
  - the domain it is intended to operate within, and;
  - the business problems it is designed to solve
- Meanwhile, Corda also provides flexibility for developers to tweak some of these trade-offs to suit their requirements
- Example: pluggable consensus
- There are no silver bullets!

## **Permissioned / Permissionless**

#### Permissioned network

Permissionless networks are inappropriate for most scenarios involving regulated financial institutions

## No blockchain / Blockchain

#### No blockchain

When considering permissioned networks, the use of a blockchain is not required to facilitate consensus

## Point-to-point/Broadcast

### Point-to-point

On the grounds of confidentiality, we reject the notion that data should be broadcast to all participants or cumbersome pre-defined groups

## Message queues / RPC

### Message queues

Messaging queues tend to be more robust at the cost of complexity and extra infrastructure

## Reliance on existing judicial systems / "Code is law"

### **Legal Prose**

A key requirement for financial institutions was the ability to rely on courts of law in the case of conflicts



## **UTXO / Account model**

#### **UTXO Model**

To maintain true immutability of the ledger, and make it possible to apply transactions in parallel

## Re-use/Build

#### Re-use

Wherever possible, we use industry standard bank friendly libraries and technology instead of "reinventing the wheel".

## But Corda also offers flexibility

- Pluggable consensus
- Flexible transaction model
- States can contain arbitrary information
- Databases and message brokers



## **Summary**

- Corda is designed to solve a range of problems for regulated financial institutions
- Corda exists because other available platforms did not satisfy the stated mission objective
- Building a DLT platform requires managing trade-offs
- Corda is opinionated but also facilitates developer flexibility



# **Key concepts**

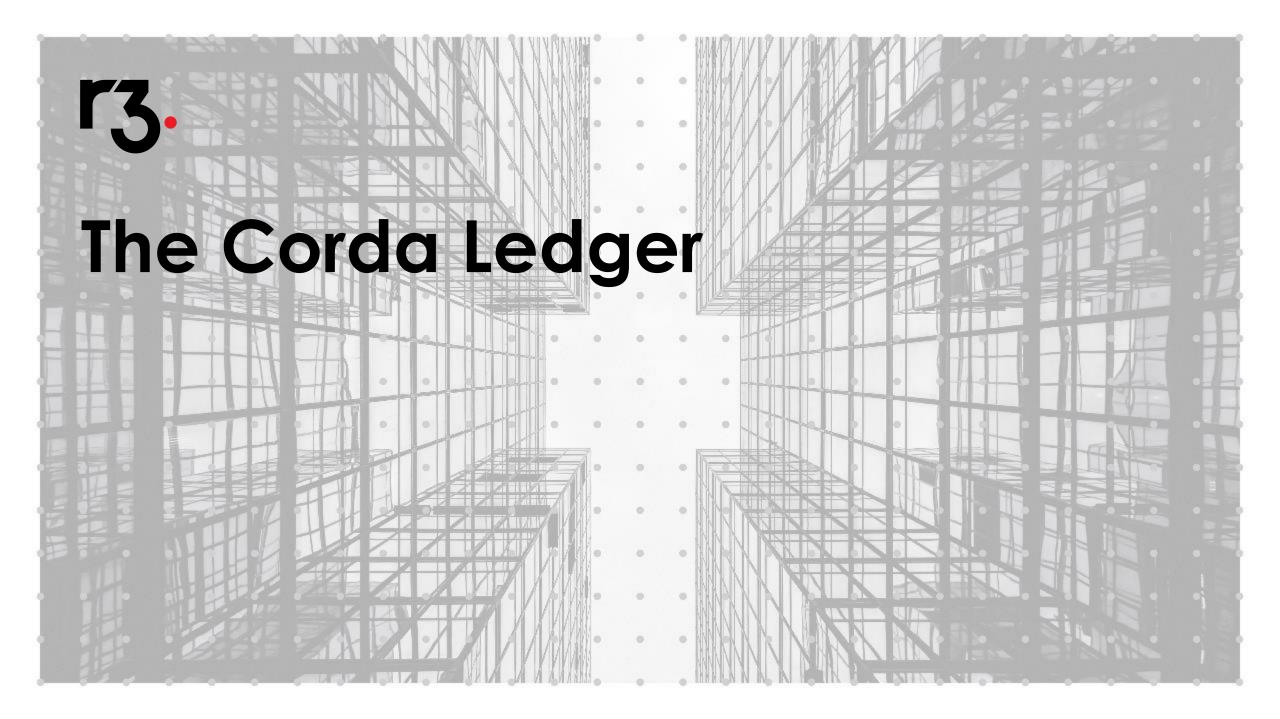
#### **Corda Key Concepts**

- 1. The Corda ledger
- 2. States
- 3. Transactions
- 4. Contracts
- 5. Legal prose
- 6. Commands
- 7. Timestamps

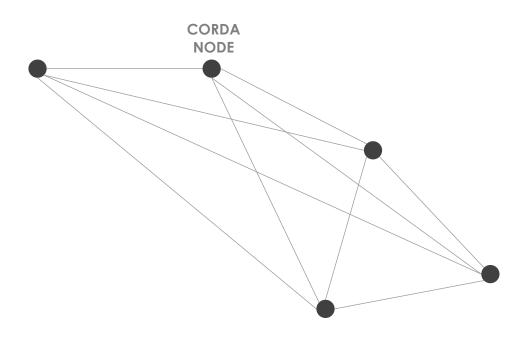
- 8. Attachments
- 9. Flows
- 10. Consensus
- 11. Notary services
- 12. Oracles
- 13. The Corda node and CorDapps
- 14. A Corda network

## Before we start

- These slides introduce Corda key concepts one at a time
- New concepts build upon previously introduced concepts
- If you have a question, it will likely be answered later on in the session!
- Example: Uniqueness consensus which prevents "double spending" is introduced near the end of the session but is a clear concern from the beginning

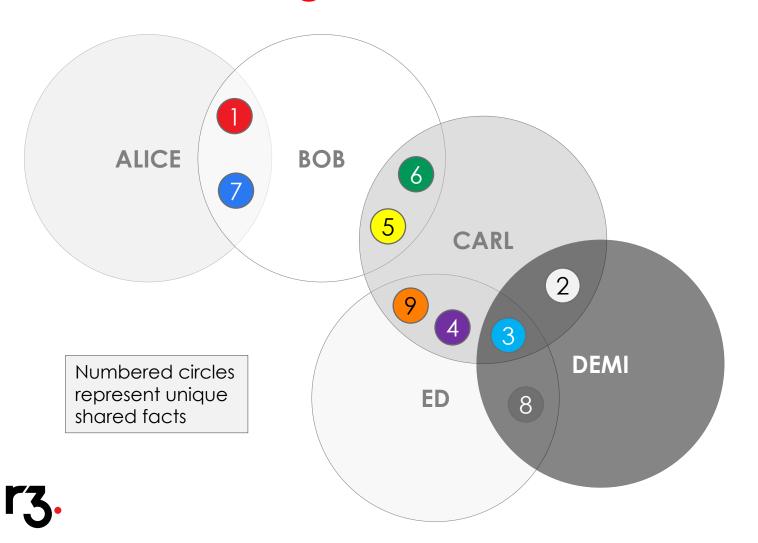


### A Corda network



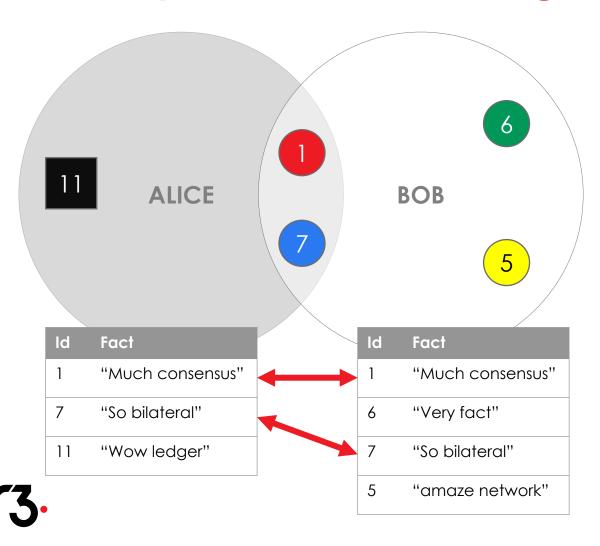
- A Corda network is a fully connected graph
- No global broadcast or gossip network
- Communication occurs on a point-to point basis only
- Peers communicate using AMQP/1.0 over TLS
- Network map service publishes list of peers
- Graph edges represent the potential to communicate, not persistent connections
- Think Email and SMTP

## The Corda ledger



The ledger from

## Anatomy of a bilateral ledger



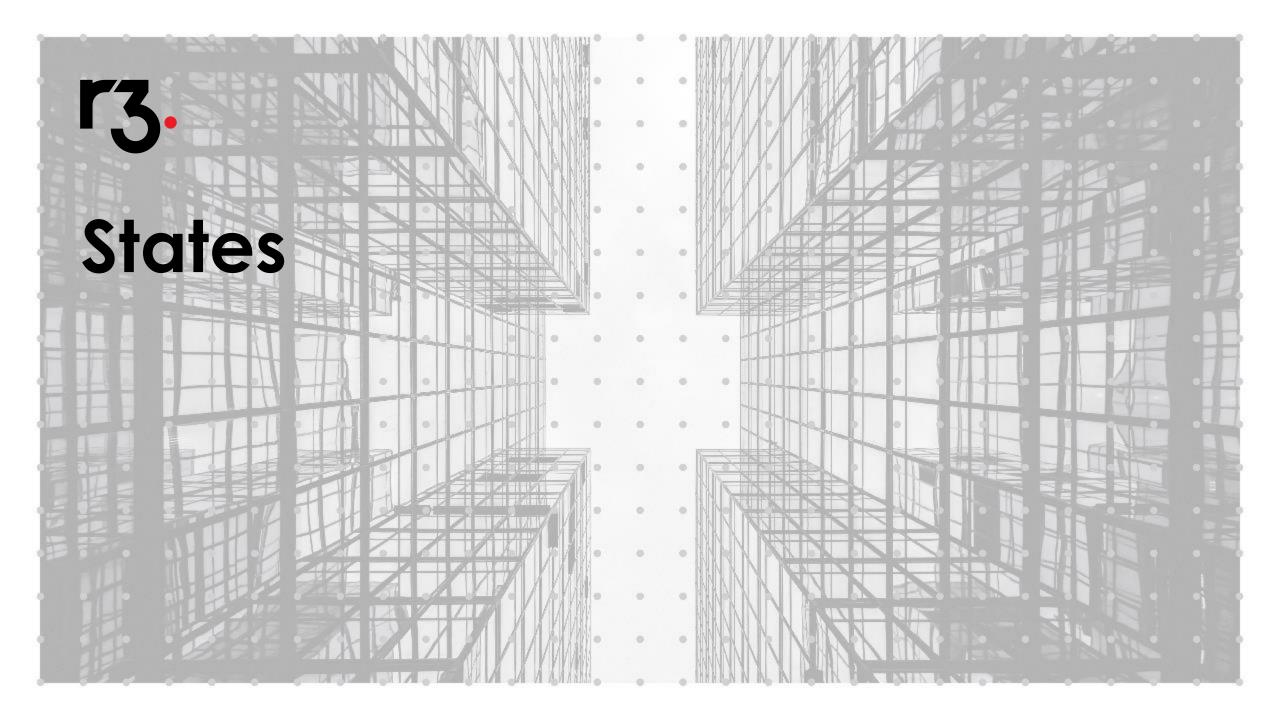
- There is no "central ledger"
- Each network peer maintains a separate
   vault of facts
- Facts are like rows in a table
- All peers to a shared fact store identical copies
- Not all on-ledger facts have to be shared with other peers
- The black square "11" is an example of a on-ledger fact not shared with any peers

p29.

The Corda ledger is

subjective from each

peer's perspective



# States represent (shared) facts

### ALICE (IOUs)

ld	From	То	Amt	Ву	Penalty	Paid
1	Alice	Bob	£10	2017-03-31	20% daily	£O

#### BOB (IOUs)

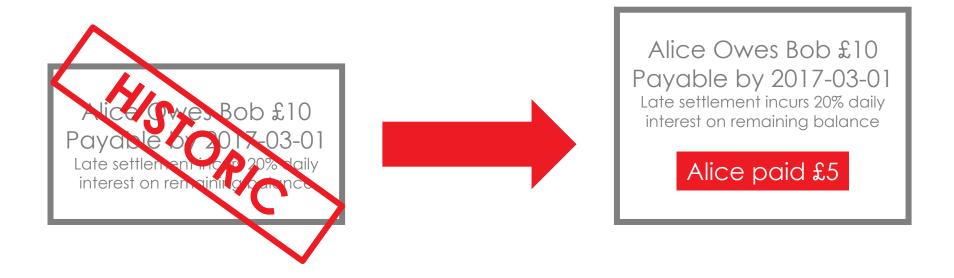
ld	From	То	Amt	Ву	Penalty	Paid
1	Alice	Bob	£10	2017-03-31	20% daily	£O

Alice Owes Bob £10
Payable by 01/03/17
Late settlement incurs 20% daily interest on remaining balance



## Alice pays Bob £5

Alice settles £5 of a £10 IOU with Bob so she creates a new updated state to reflect this and marks the old one as historic





# States can contain anything

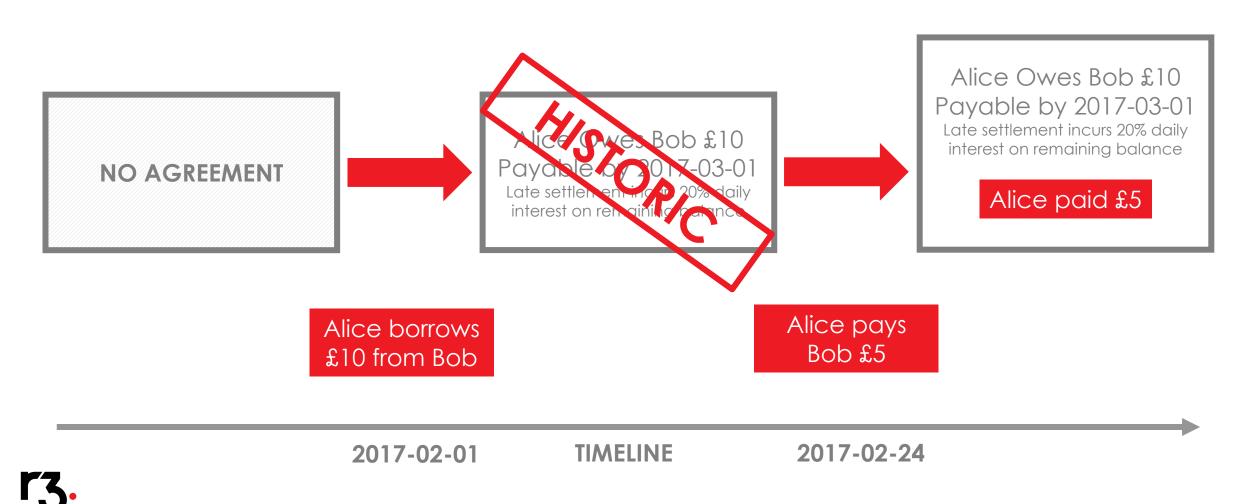
- In Corda, states don't just represent digital cash
- The state model can be used to represent literally anything
- States are statically typed an IOU state is always an IOU state

Bonds, Syndicated Loans, CDOs, CLOs, Reference Data, Invoices, Letter of Credit, Purate Swaps, Accounting Entries, Contract For Difference, Commercial Bank Credit, Contract For Difference, Commercial Bank Credit, Contractions, Collateral, KYC Data, Credit Default Swaps, Bids/Offers, Personal Information

**r**3.

States are immutable objects that represent (shared) facts such as an agreement or contract at a specific point in time

## State sequences



p36.

### State sequences

Alice Owes Bob £10
Pavaca hy 2017-03-01
Late sattlement inche 20% daily interest an remail in a balance

Alice paid £5

Alice Owes Bob £10
Payable by 2017-03-01
Pate Settlement incurs 20% daily interest on remaining balance

Alice Incurs

interest of £1

AGREEMENT EXPIRED

interest is charged!

Alice pays
Bob £6 (£5
principal and
£1 interest)

2017-03-02

**TIMELINE** 

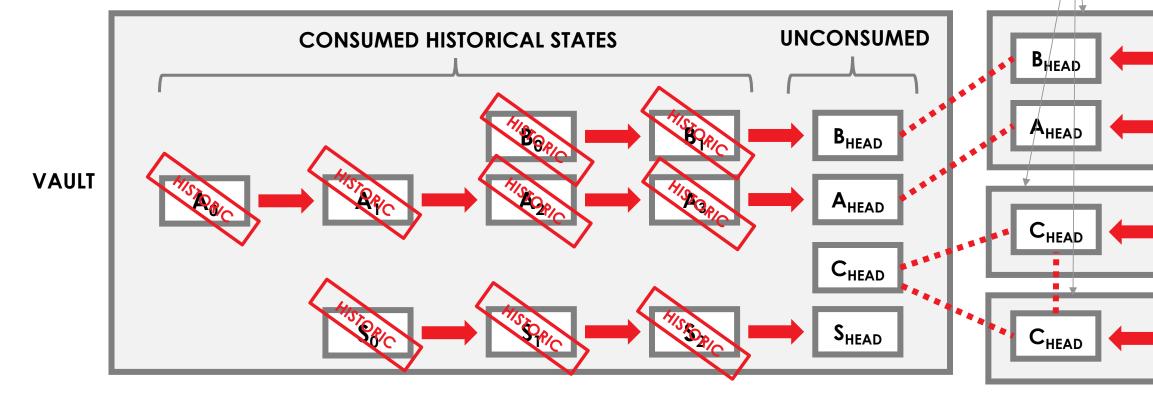
2017-03-02

**r**3

The life-cycle of a shared fact or agreement over time is represented by a state sequence

### The vault tracks state sequence heads

- The vault tracks the heads of each state sequence
- For states representing shared facts, each peer retains an identical copy

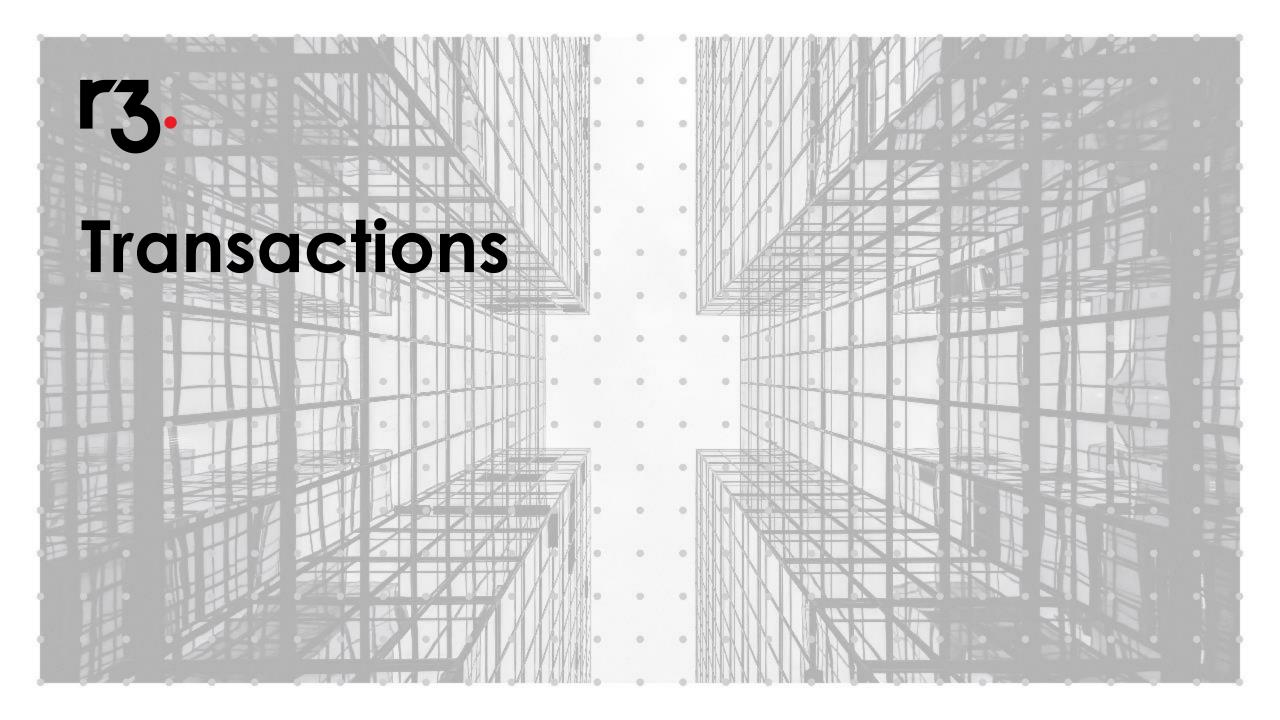


**r**3

**Counterparty** 

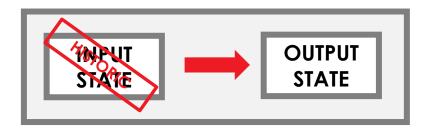
vaults

The ledger from each peer's point of view consists of all the state sequence heads (or non-historic states) tracked in the vault



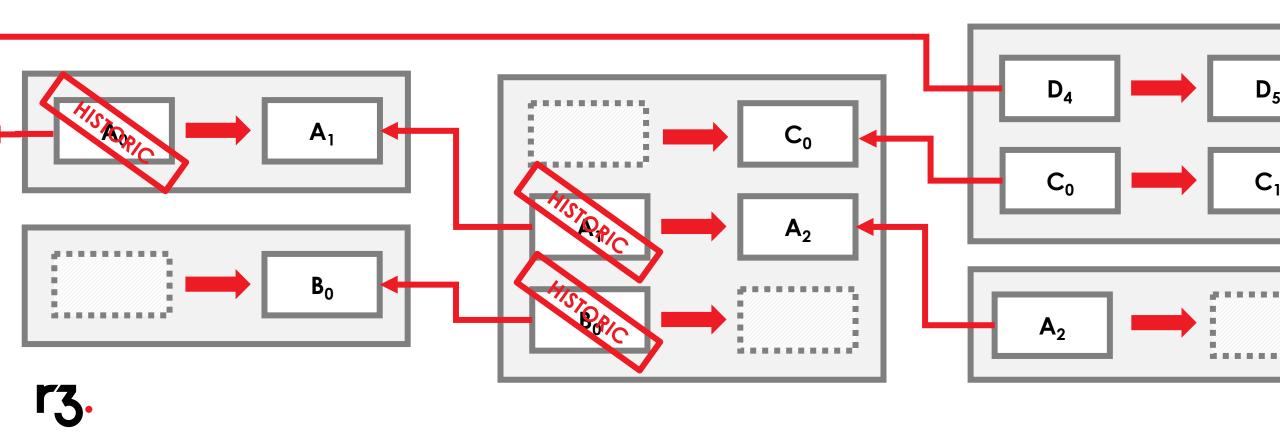
#### **Transactions**

- We introduce transactions as atomic units of change to update the ledger
- Transactions reference zero or more input states and create zero or more output states
- The newly created output states replace the input states which are marked as historic



## Transactions can be arbitrarily complex

Transactions may combine issuances, updates and exits.



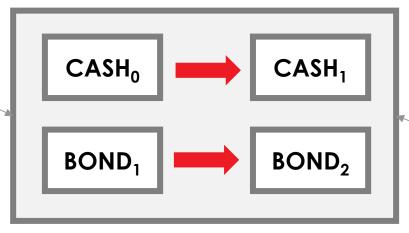
### Transactions can be arbitrarily complex

Transactions can contain states of many types.

Or this transaction could represent a delivery versus payment of some cash for a bond

- 1) Cash is paid from the buyer to the seller
- 2) The bond is moved from the seller to the buyer

#### **TRANSACTION**



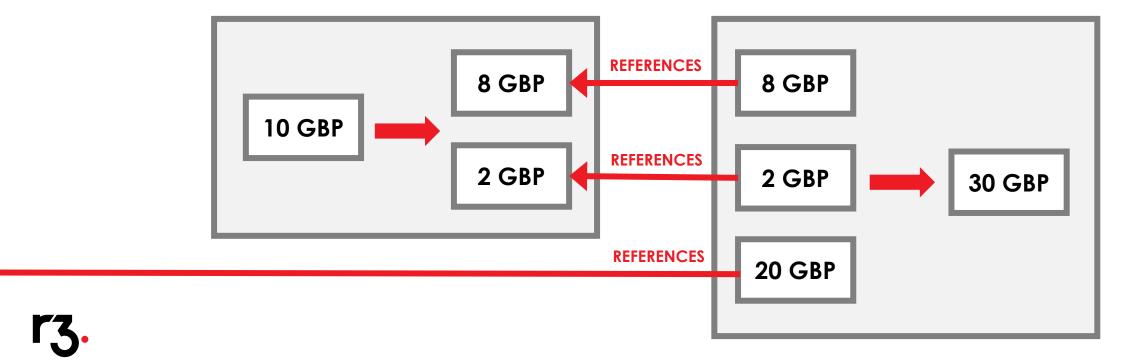
This transaction could represent a coupon payment on a bond:

- 1) Cash is paid from the bond issuer to the bond owner.
- 2) The bond shared fact is updated to reflect the coupon payment



### Transactions can split and merge states

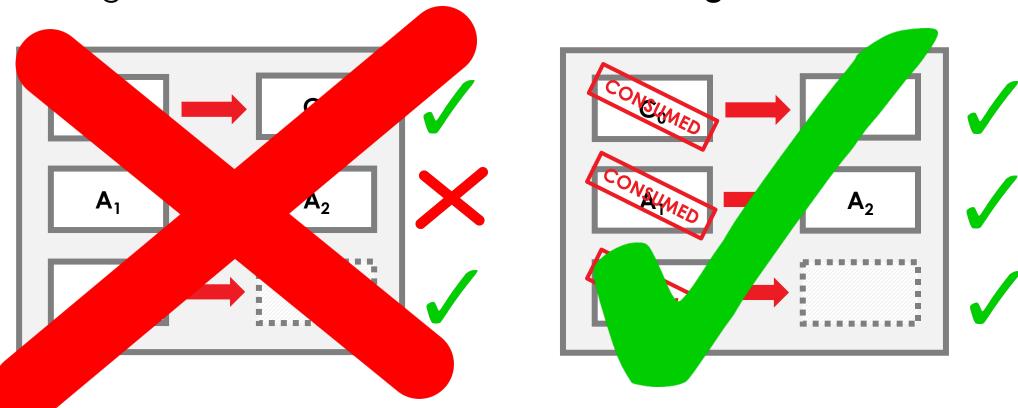
Transactions can also split and merge states representing fungible assets.



p45.

#### Transactions are atomic

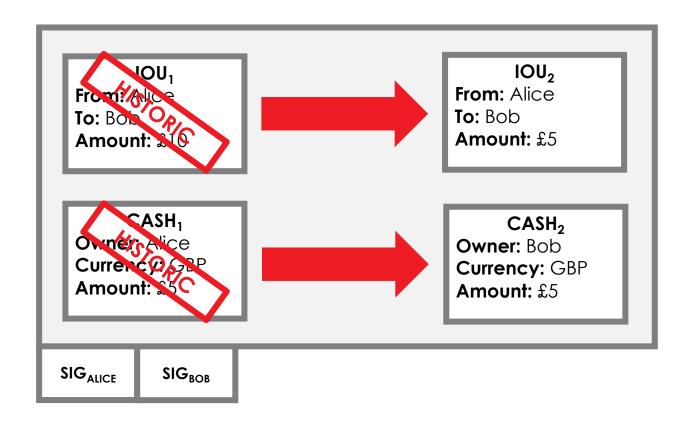
A Transaction in Corda is an **indivisible** and **irreducible** set of changes such that either **all** occur, or **nothing** occurs.



Transactions are an atomic set of changes to update the ledger

### **Committing transactions**

**Example:** Alice partially settles an IOU with Bob.



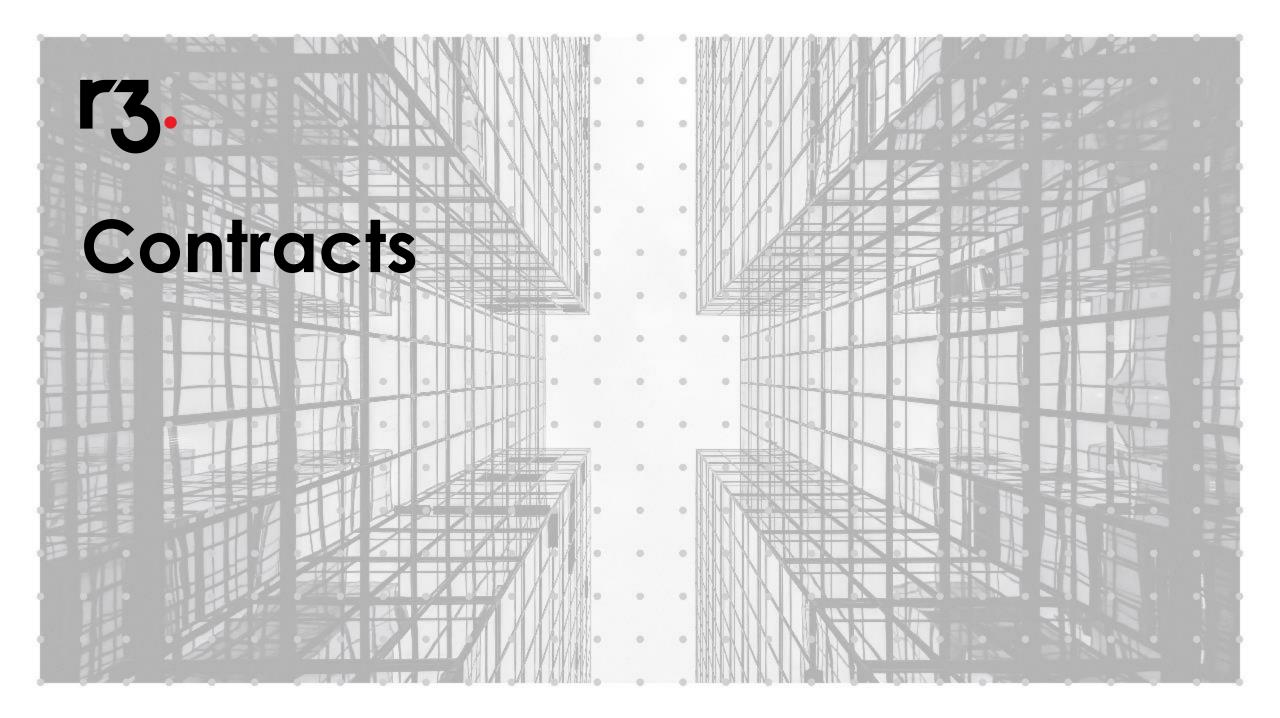
And signs as action is The transaction is as a sign of the transaction is a sign of t

**r**3.

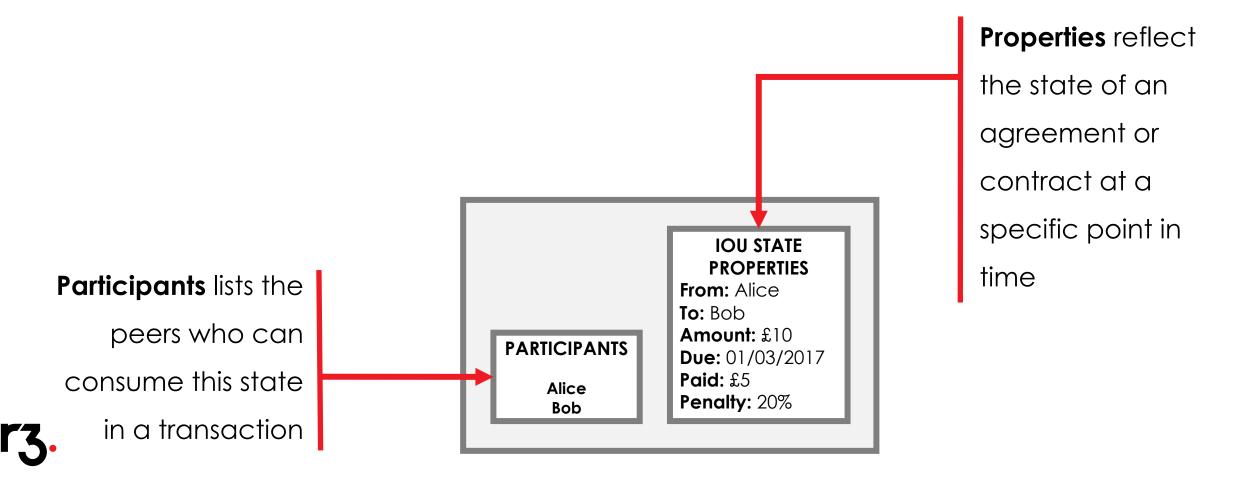
**Transactions** are committed and update the ledger only when signed by all required peers

Uncommitted transactions are proposals to update the ledger

**Committed transactions** are immutable



## A closer look at state objects...

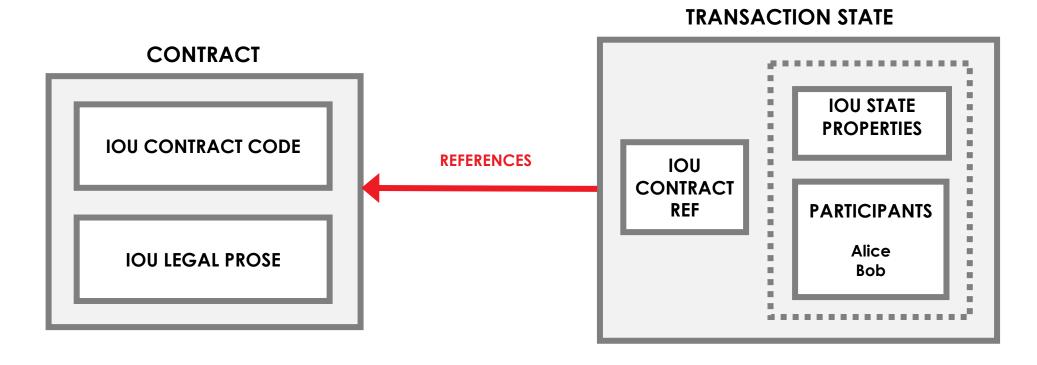


### **Transaction states**

Contract ref point to a contract which defines the verification function IOU **CONTRACT REF STATE** 

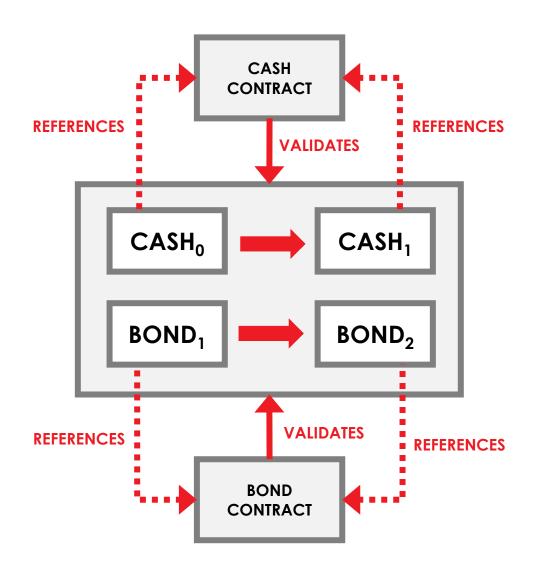
#### **Contracts**

Corda mandates that each state inside a transaction must reference a contract



#### **Contracts**

- As transactions may contain multiple state types, multiple contracts can be referenced in a transaction
- The Corda platform will use all referenced contract code to verify a (proposed) transaction



**r**3.

#### Contract code

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 returns nothing if the transaction verifies.

fun verify(tx: Transaction): Unit

In Kotlin, **Unit** is a type with only one value: the **Unit** object. This type corresponds to the **void** type in Java/C.

**r**3

#### Contract code

```
// Transaction must have zero inputs.
tx.inputs.size() == 0
// Transaction must have one output.
tx.outputs.size() == 1
// The lender cannot be the borrower.
tx.outputs[0].lender != tx.outputs[0].borrower
// The output state must contain an amount > 0.
tx.outputs[0].amount > 0
```

p56.

The **contract code** is a "pure" function executed in a deterministic environment, on a need-to-know basis which verifies transactions

#### Transactions are not instructions

- Transactions are **not** instructions which require action
- Instead, transaction creators calculate an updated ledger which is reflected by the output states
- The output states are the updated ledger!
- In other words: Corda transactions state what the updates are as opposed to how to calculate the updates

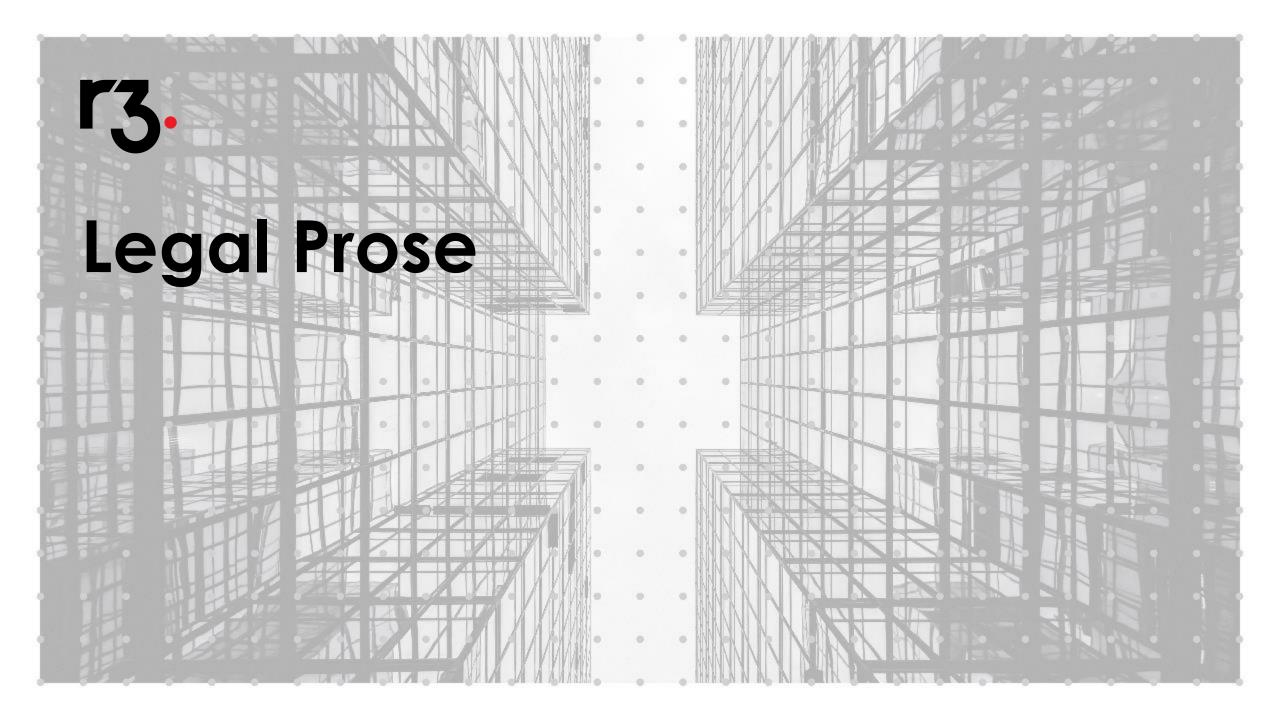
Transaction proposals require

verification which is performed

separately to transaction creation

### Transactions in summary

- 1. Any peer may create a transaction proposal
- 2. Transaction proposals are uncommitted by default
- 3. Before a transaction proposal is committed it must first be digitally signed and then verified by all required peers on a need-to-know basis
- 4. Once a transaction is committed it marks the input state references as historic and creates new output states reflecting an updated ledger



# Legal prose

HASH

### IOU LEGAL PROSE TEMPLATE

This document binds
BORROWER to pay LENDER
the sum of AMOUNT by
DATE.

If the full amount is not paid by **DATE** then **BORROWER** will incur **RATE** interest on a **PERIOD** basis until full payment is received.

IOU LEGAL PROSE PARAMETERS

**BORROWER** ALICE **LENDER** BOB

AMOUNT £10

**DATE** 2017-03-01

RATE 20%
PERIOD DAILY

IOU CONTRACT CODE

IOU LEGAL PROSE

92e723du6d986bc...

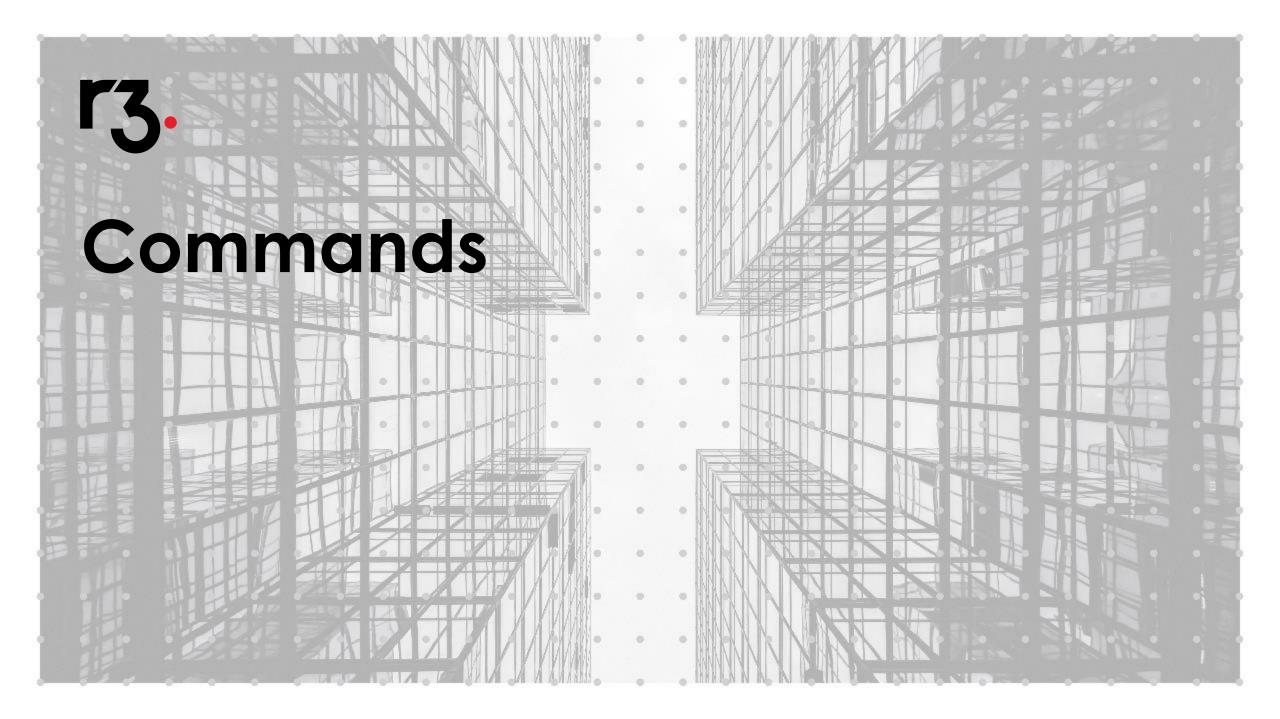
**CONTRACT** 

**r**3.

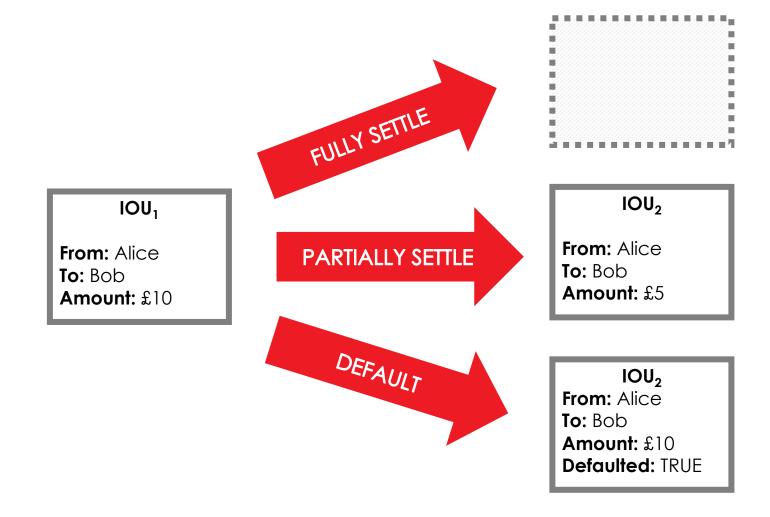
p62.

Legal prose is referenced within

Contracts and can be relied upon
when the contract code alone is
insufficient



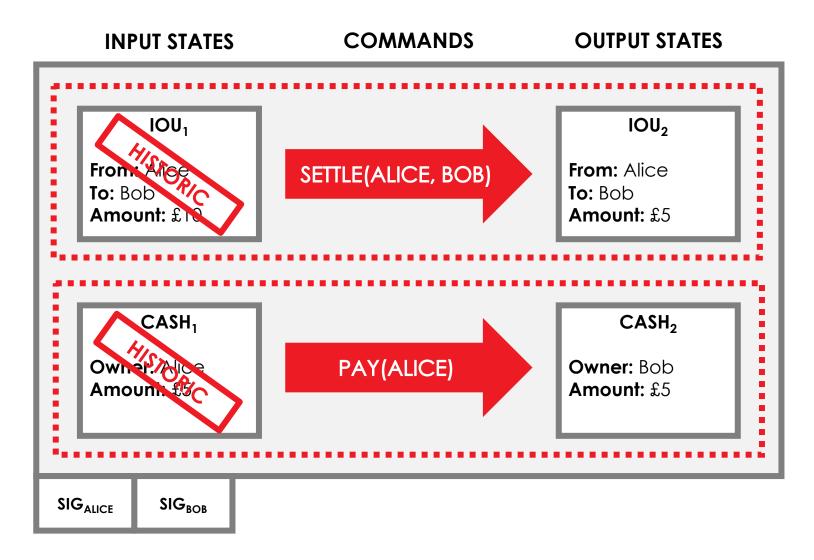
### States can evolve in multiple ways



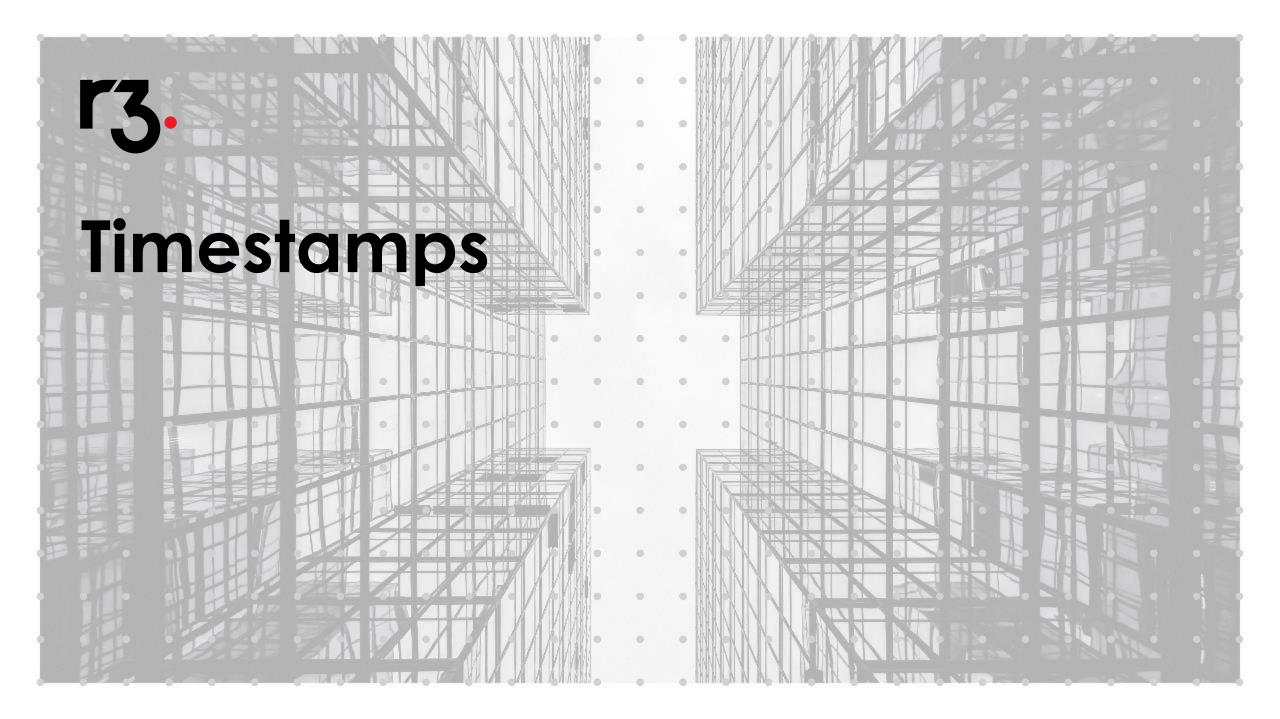
**r**3.

#### Commands

Alice settles £5 of a £10 debt with Bob, so creates a transaction proposal



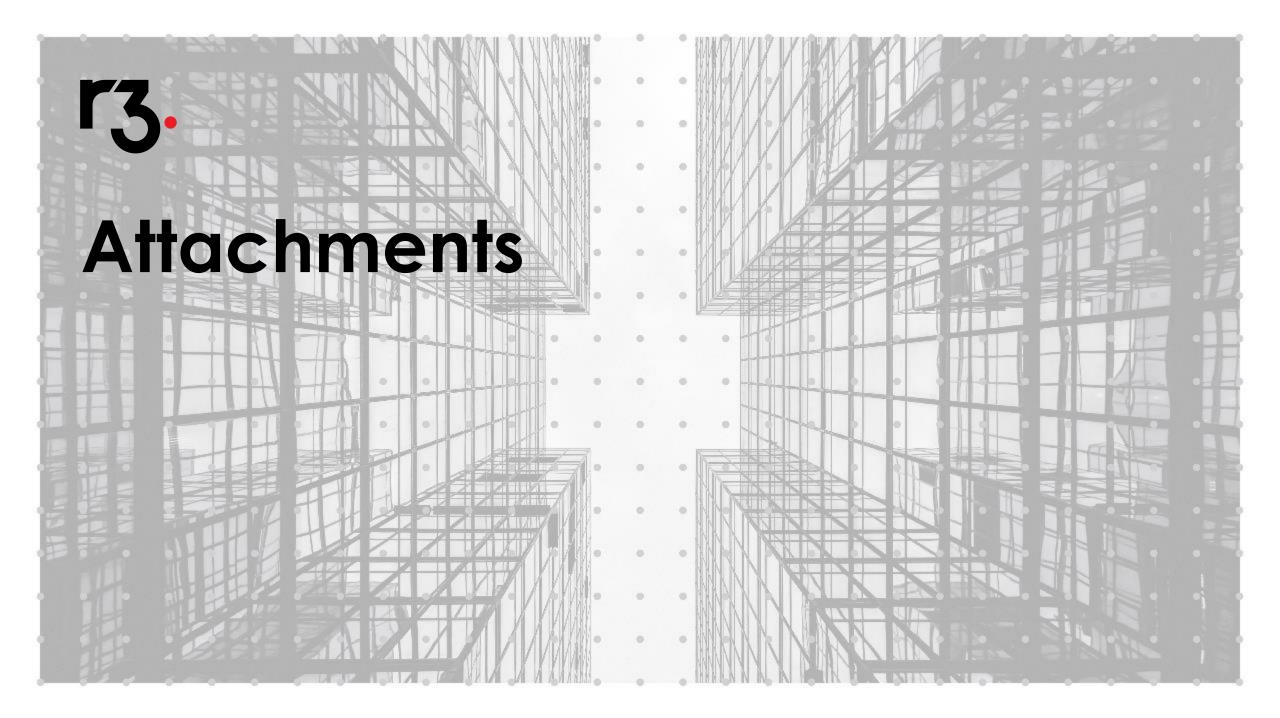
Commands parameterise
transactions hinting to their intent
and specify the required signers via
a list of public keys



### **Timestamps**

Timestamp windows may be open ended in order to Timestamps can communicate that the transaction sastianted curred before a certain aintination of the power nuch before or after is unimportant. specified time **TIMELINE** 

Timestamps assert that a transaction happened within a specified time window

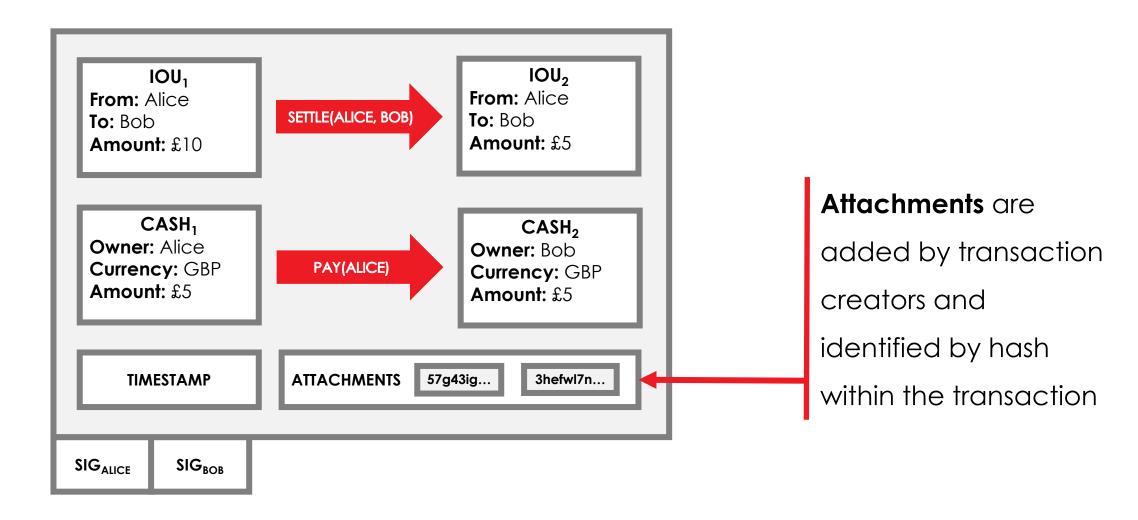


#### **Attachments**

- Transactions can also contain a number of attachments
- Attachments are zip files and identified by hash
- Attachments are referenced within a transaction, but not included in the transaction itself
- Attachments are intended for data on the ledger that multiple peers may wish to reuse over and over again
- A transaction creator chooses which files to attach

**r**3

#### **Attachments**



**r**3.

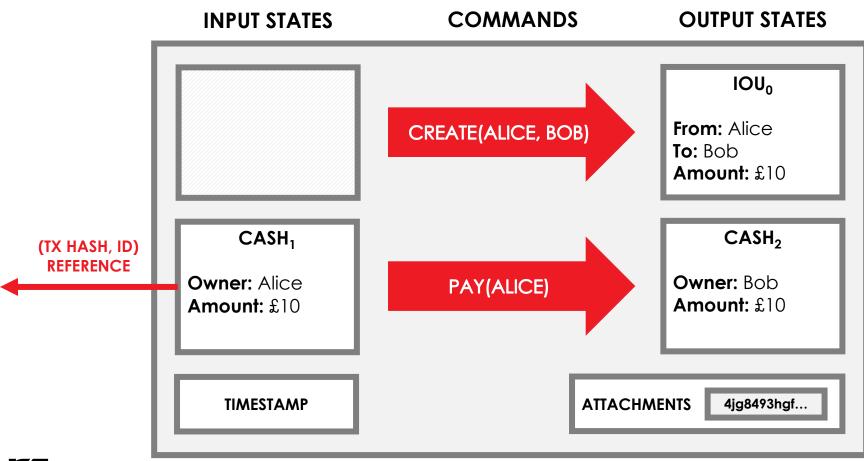
#### **Attachments**

Attachments may contain:

- Contract code and associated state definitions (.class files)
- Legal prose template and parameters
- Data files which support the contract code e.g. currency definitions, public holiday calendars or financial data

Attachments are zip files
referenced in a transaction by
hash but not included in the
transaction itself

#### Creating a transaction proposal



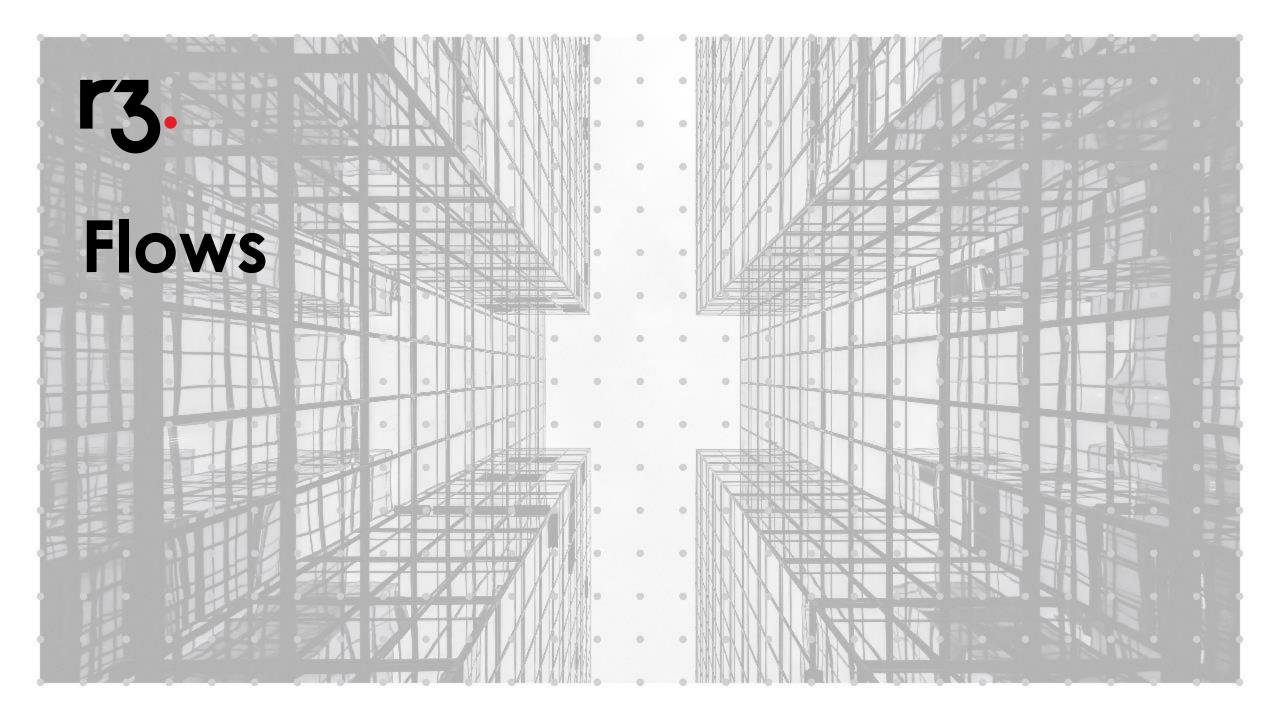
E. Mathinitish should be dependent of the should be described by the should be described to sign the transaction

**r**3

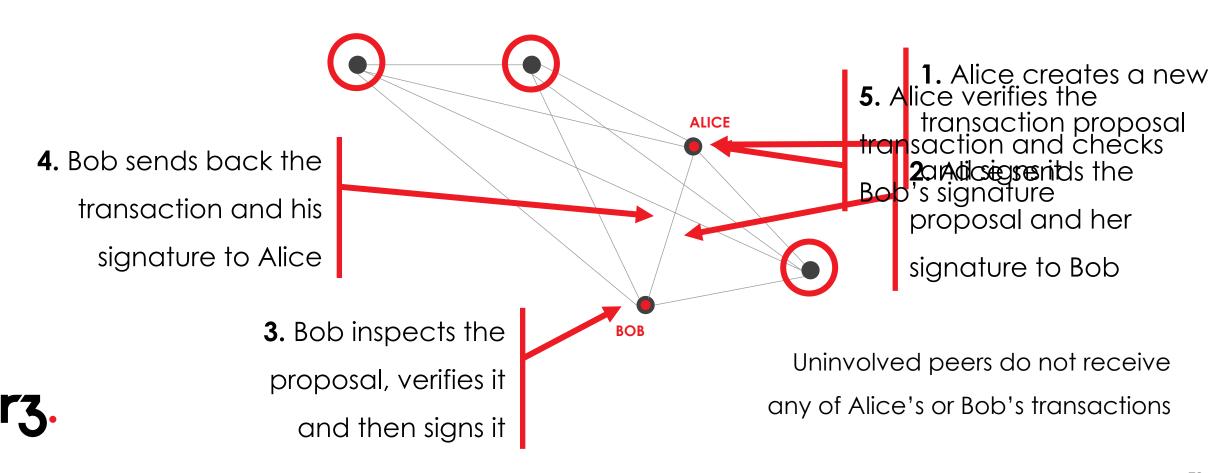
### Transactions summary

- Transactions are an atomic set of changes
- Transactions contain zero or more input and output states
- States are grouped by type
- Transactions contain one or more commands
- Transactions can contain a timestamp
- Transactions can contain zero or more attachments
- Transactions are proposed and then subsequently signed and verified by the required peers

**r**3.



### Alice and Bob agree upon an IOU



### You can write blocking code that never blocks!

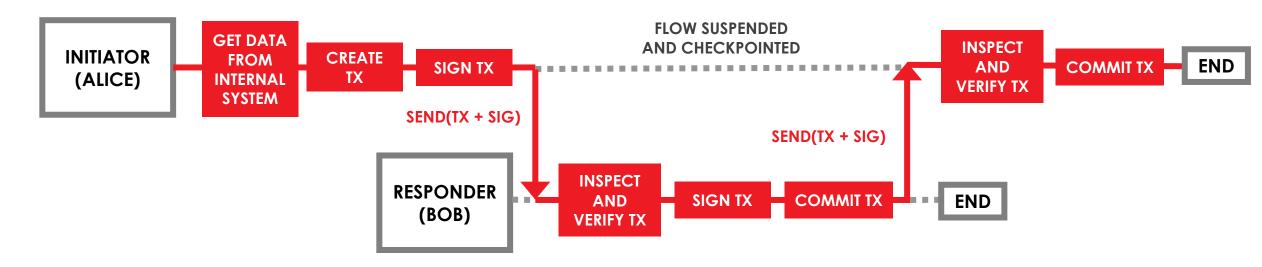
```
check-pointed
// Alice
                                                       BOB
                     When calling the network
                                                                               here
                     Alice's fibre is suspended and
                     serialised to disk (or check-
                                                   peer = "Alice"
tx = new Tx()
                     pointed). If Alice's node fails
                     or restarts, she can deserialise
peer = "Bob"
                                                   Res = receive(peer)
                     the fibre and continue the
                     flow when her node reboots
sig = sign(tx)
                                                   check(res.sigA)
payload = (tx, sig)
                                                   verify(res.tx)
res = sendAndReceive(payload, peer)
                                                   sigB = sign(res.tx)
                          Check-pointed
check(res.sigB)
                                                   payload = (res.tx, sigB)
verify(res.tx)
                                                   send(payload, peer) 
commit(res.tx)
                                                   commit(res.tx)
```

**r**3

Bob's flow is

### The two party deal flow

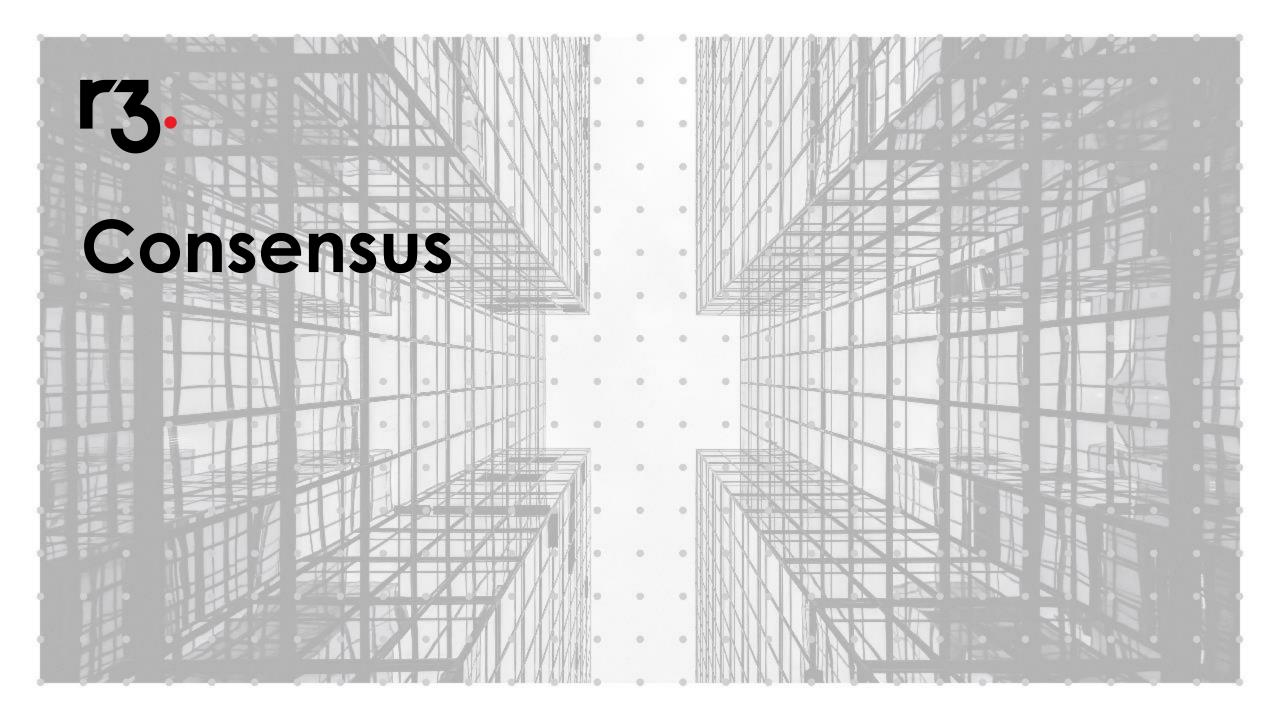
This is the flow Alice and Bob use to agree upon an IOU, in Corda it can be called as a sub-flow.





p81.

Flows are light-weight processes used to coordinate the complex multi-step, multi-peer interactions required for peers to reach consensus about shared facts



### Two types of consensus

Peers reach consensus over transactions in two ways:

Verification consensus

Uniqueness consensus



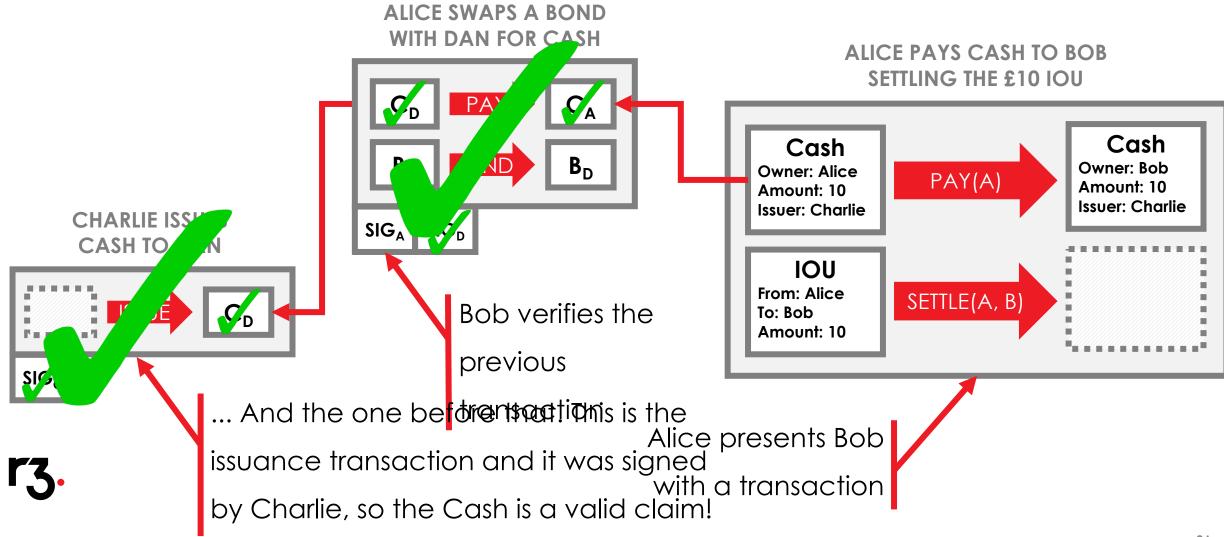
#### **Verification consensus**

In Corda, verification consensus involves peers reaching certainty that a transaction:

- 1. is signed by all required peers listed in the commands
- 2. satisfies the constraints defined by the contracts pointed to by the input and output states

However there is an additional step required...

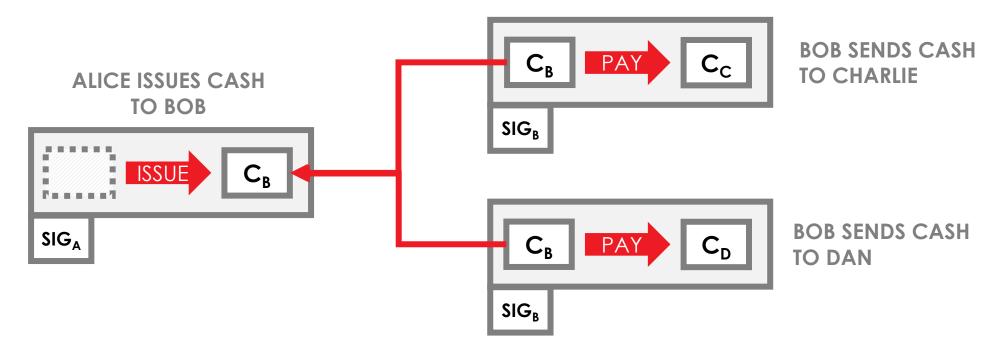
#### **Verification consensus**



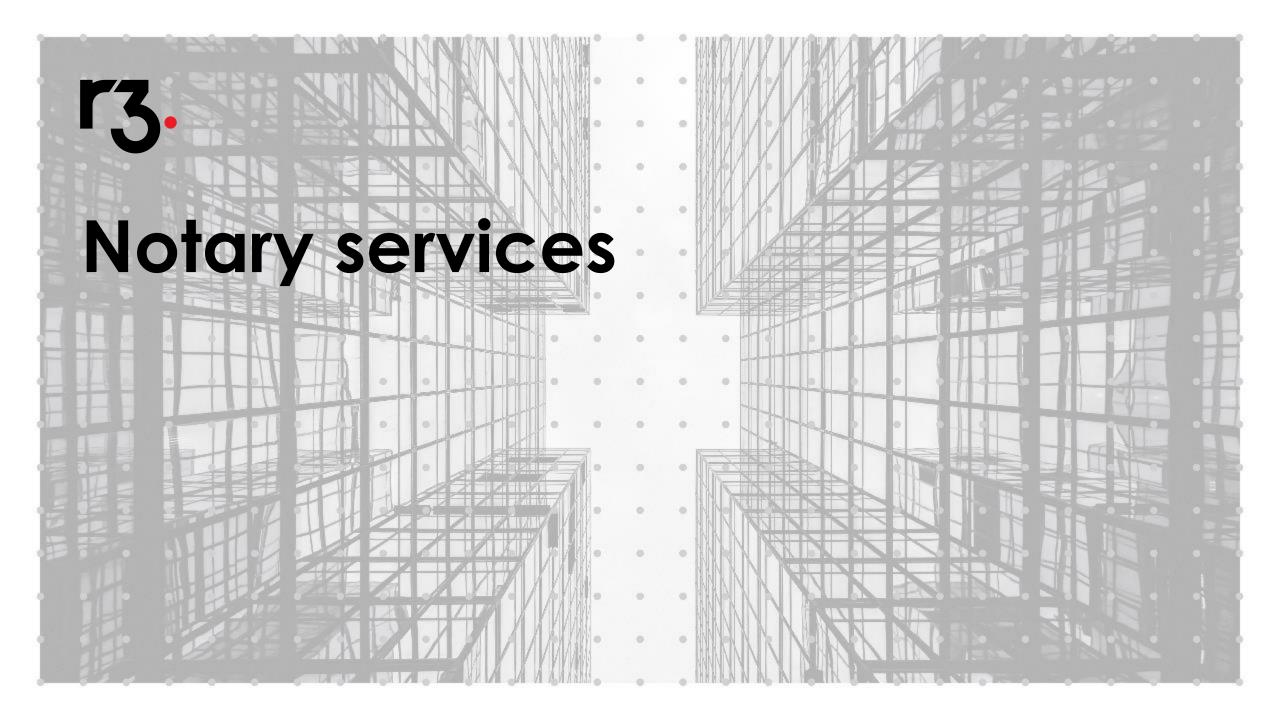
Verification consensus involves
reaching certainty that a transaction
(and all its dependencies) is signed by
all required peers and satisfies the
constraints in the contract code

#### Uniqueness consensus

By now you have probably realised that without uniqueness consensus a nefarious actor can use the same cash input state reference in multiple transactions. How do we stop this?



Uniqueness consensus involves peers reaching certainty that the output states created in a transaction are the unique successors to the input states referenced by that transaction



### Notary services track used states

In simple terms, notaries **maintain a map** keyed with input state references:

Key: (Transaction ID, Output Index)

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

The map values indicate the **ID** of the transaction which **used** the state as an input and marked it as historic, as well as the identity of the requesting peer.

### **Notary services workflow**

When a proposed transaction is sent to a notary service the notary checks if any of the input state references are already in the map and one of two things may happen:

- If any are in the map, the notary throws an Exception and notes that there is a conflict
- 2. If none are in the map then the notary adds each input state to the map and signs the proposed transaction

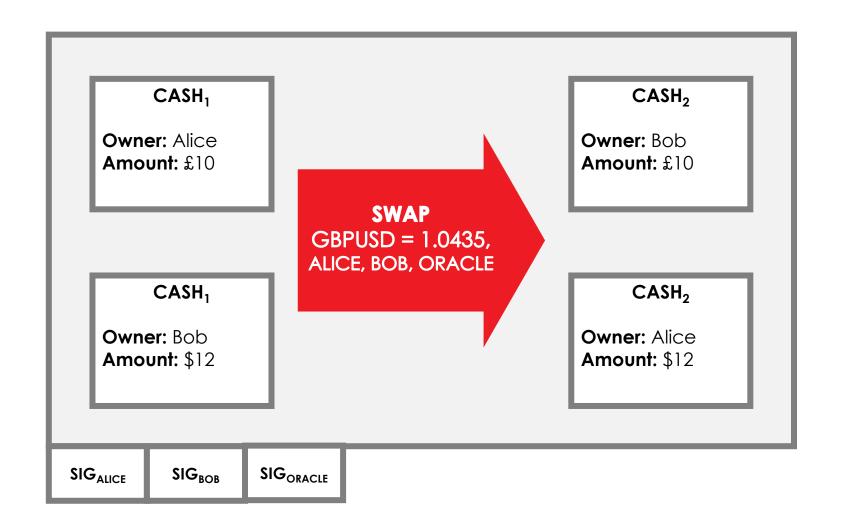
Notaries provide uniqueness consensus



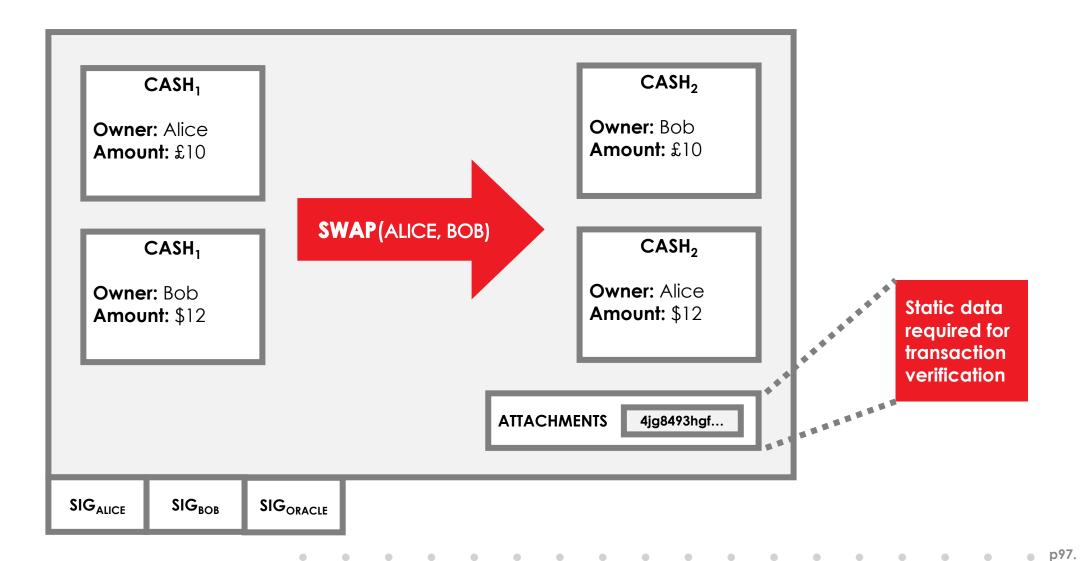
#### **Oracles**

- An oracle is a source of data or calculations, which has been accepted by multiple peers as authoritative,
   binding and definitive for an agreed set of values or range of calculations
- The oracle may source its data from external observations or calculate its results based on inputs received from onledger states or attachments

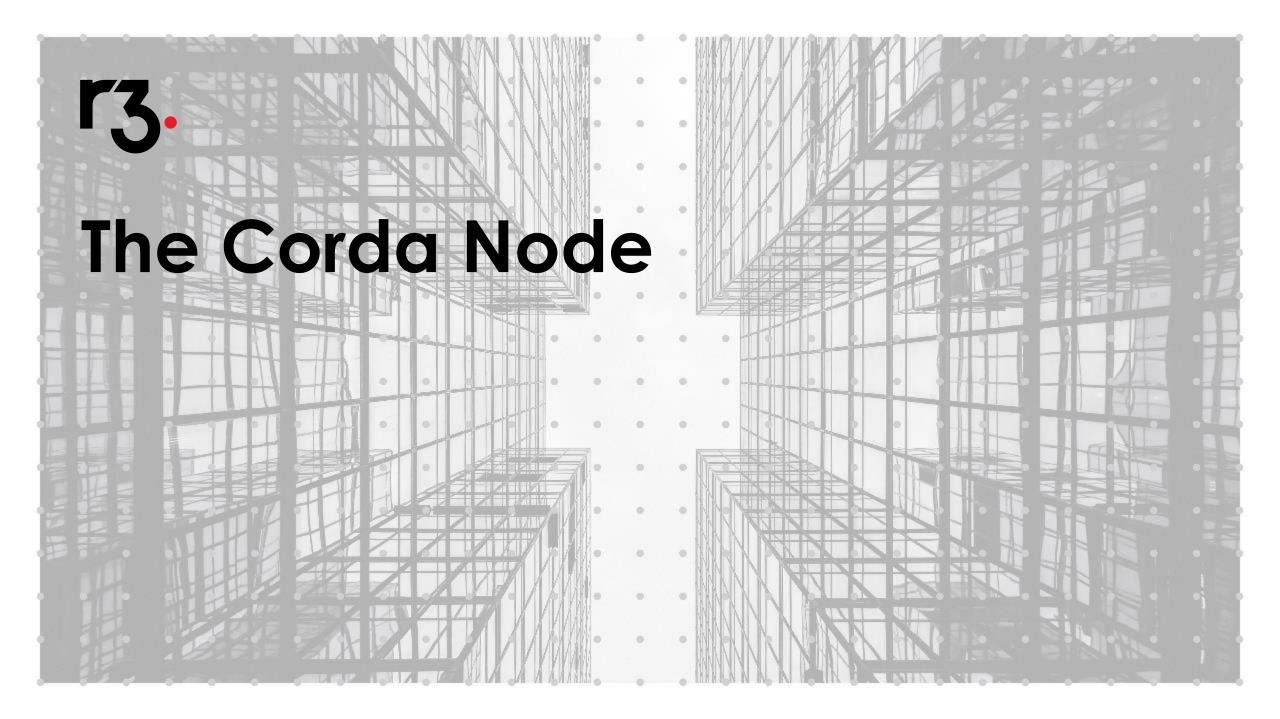
#### Embedding external data in commands



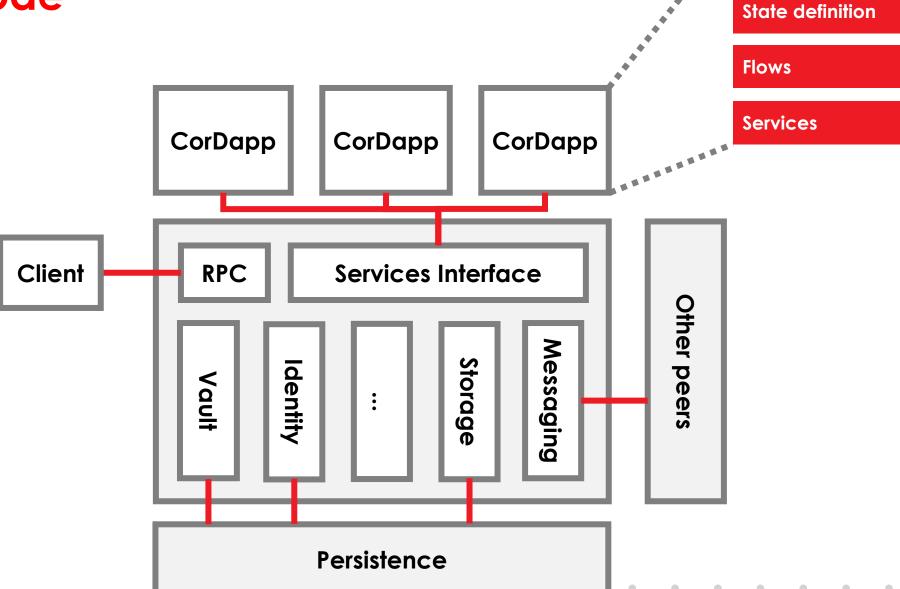
# **Embedding data in attachments**



Oracles are an authority that attests to (and may also provide) off-ledger facts needed to verify transaction proposals

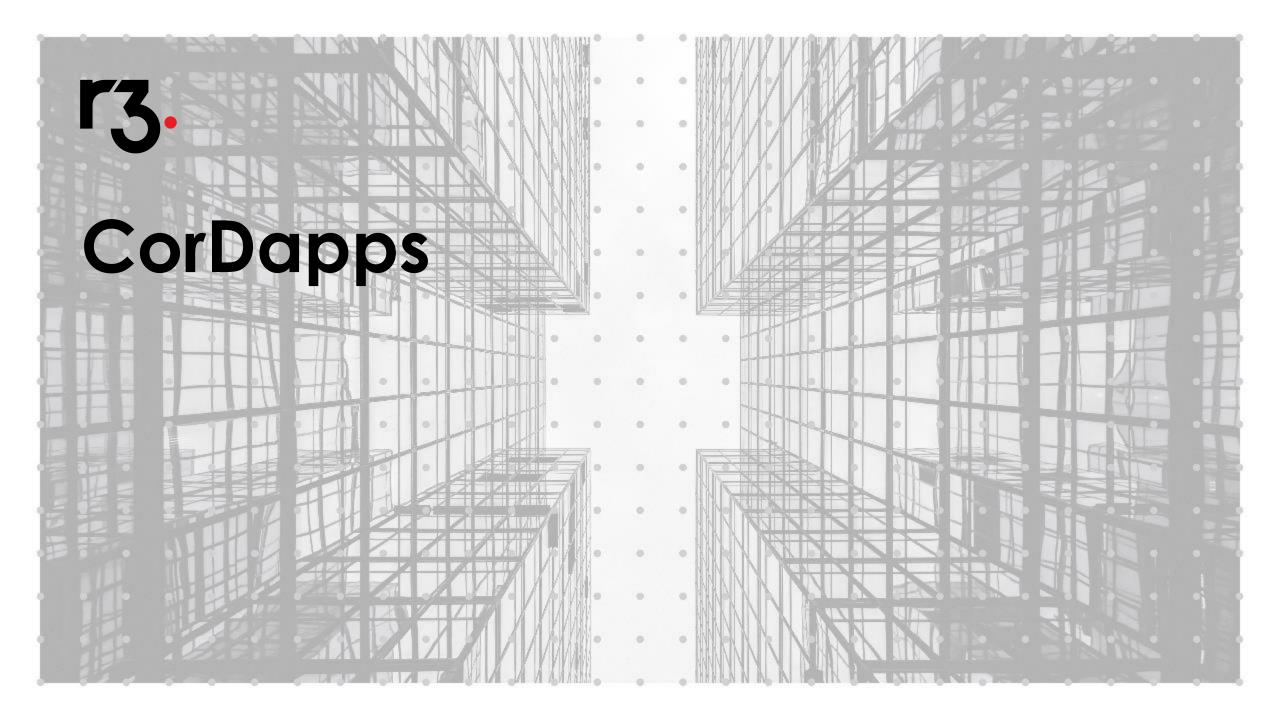


#### A Corda node

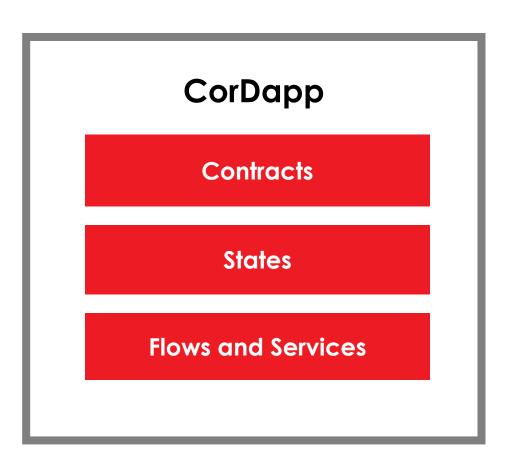


Contracts

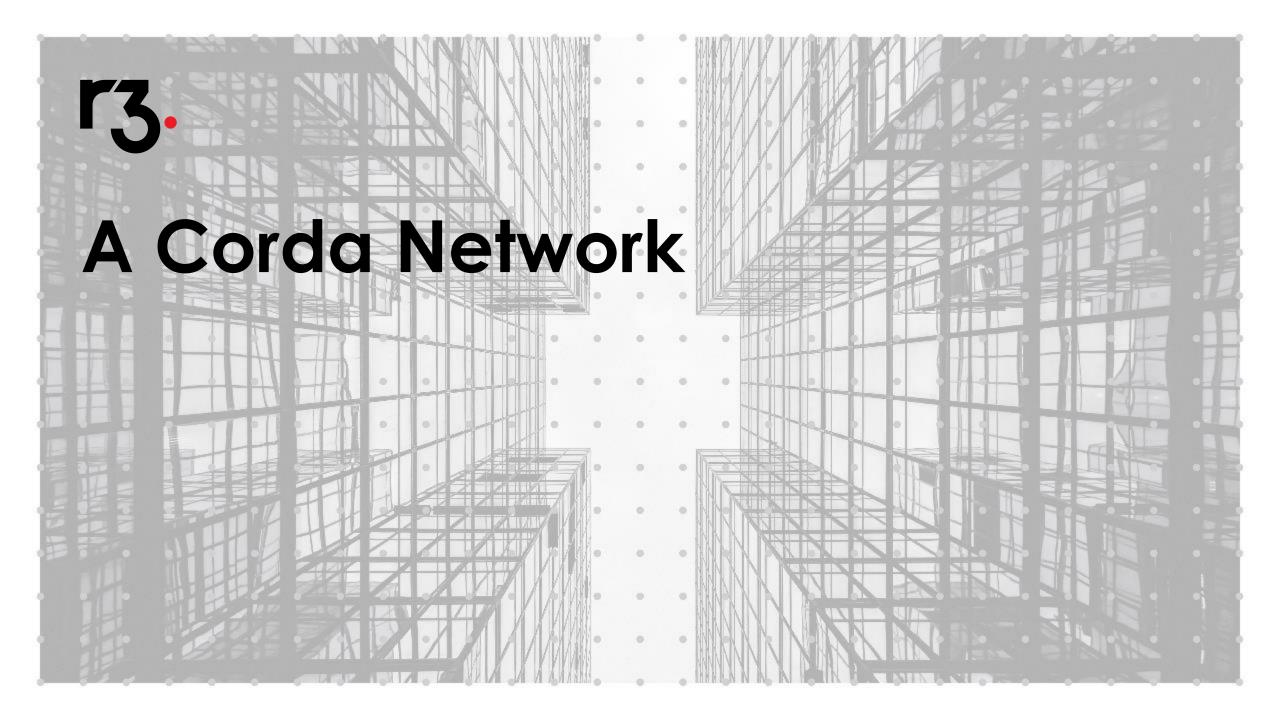
A Corda node implements a collection of services required to participate in a Corda network



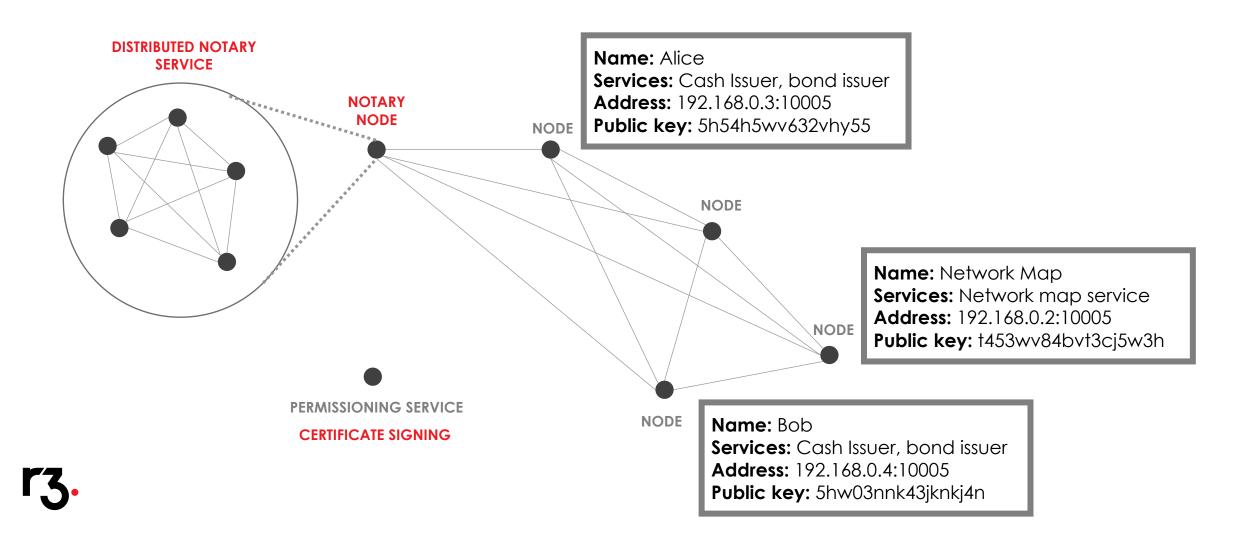
# CorDapps



CorDapps are Corda node extentions which comprise the states, contracts and flows required to implement some specific business logic



#### A Corda network



#### A Corda network comprises:

- A doorman
- Two or more Corda nodes
- Zero or more oracles
- A network map service
- One or more notary services

### **Summary**

#### **Corda Key Concepts**

- 1. The Corda ledger
- 2. States
- 3. Transactions
- 4. Contracts
- 5. Legal prose
- 6. Commands
- 7. Timestamps

- 8. Attachments
- 9. Flows
- 10. Consensus
- 11. Notary services
- 12. Oracles
- 13. The Corda node and CorDapps
- 14. A Corda network

## Learning objectives

- Understand that Corda has been designed to solve a specific business problem for regulated financial institutions and this is reflected by its architecture
- 2. Understand how to think "the Corda way"
- 3. Understand the key concepts which underpin how Corda works