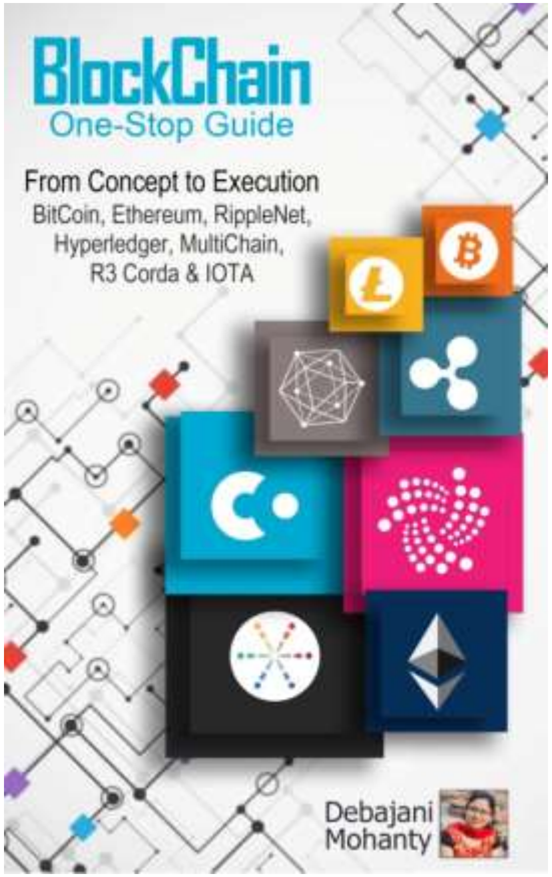
Triple Entry Accounting
A BlockChain Use Case for Banks
With R3 Corda

Debajani Mohanty
Senior Technical Architect
NIIT Technologies Ltd

Who Am I?

- Foodie, Traveler, Writer
- Sleep-deprived mom of 2 little gals
- Author of “BlockChain One-Stop Guide: From Concept to Execution” on Amazon
- Senior Architect with NIIT Technologies Ltd

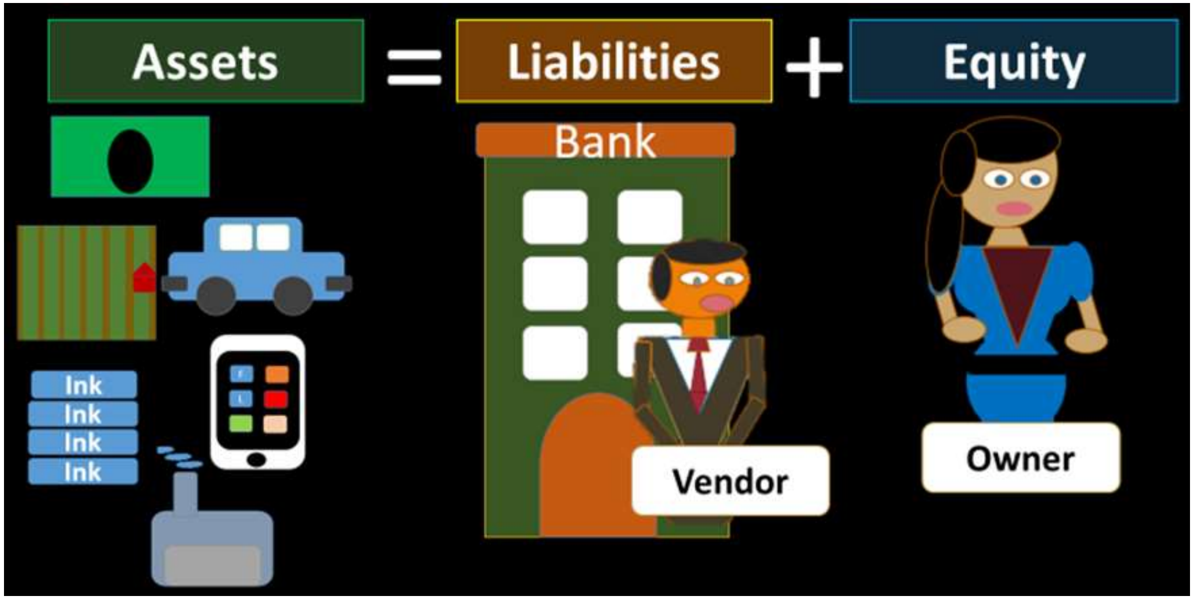


Single Entry Accounting

- Single entry accounting is there since the beginning of civilization.
- Someone buys and someone sells and one keeps own accounting details.
- Any mistake in accounts, nobody would blame.

Double Entry Accounting

- Double-entry accounting introduced some 600 years back today is the worldwide standard for business accounting.
- A and B are doing some transaction with each other and both are keeping a transaction of the exchange i.e. every transaction records in at least two accounts—in one as a debit and in the other as a credit.
- Debits increase expense and asset accounts but decrease income and liability accounts. Conversely, credits increase income and liability accounts and decrease expense and asset accounts.
- So the transaction at each level should look equal and opposite total assets remaining the same.



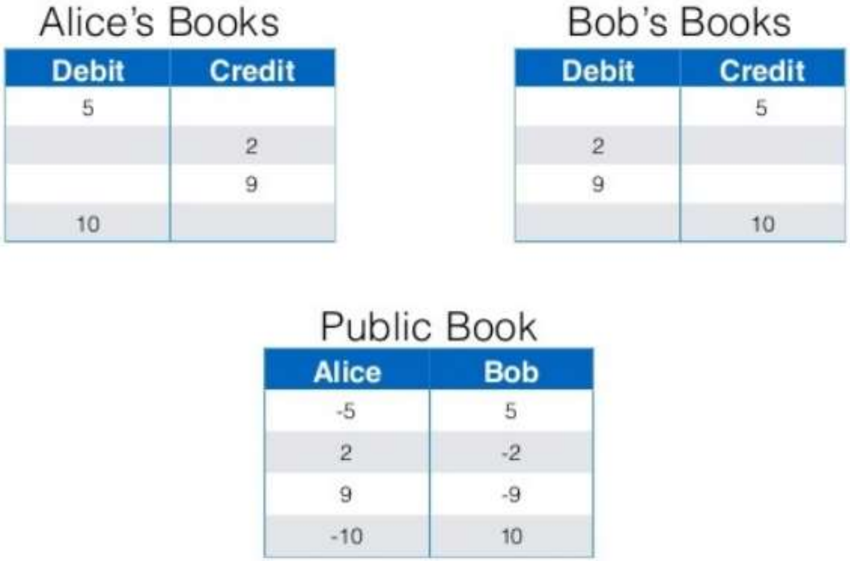
Issues with Double Entry Accounting

- Time consuming, manual and error prone.
- Fraud: People may manipulate over here to get opportunity as no third party is involved.
- Auditing difficulty: Double entry accounting records are difficult to audit and costly.

Triple Entry Accounting

Triple entry accounting was a process introduced by financial cryptography expert **Ian Grigg** in Dec 2005

http://iang.org/papers/triple_entry.html



Alice's Books

Debit	Credit
5	
	2
	9
10	

Bob's Books

Debit	Credit
	5
2	
9	
	10

Public Book

Alice	Bob
-5	5
2	-2
9	-9
-10	10

Transactions go through a contract in real-time maintained by a 3rd party agent or entry which both parties connect to and agree.

Why to Use BlockChain For Accounting?

- Cryptographically secure
- Transactions can only be added, not altered. Hence no manipulation possible.
- Fraud proof. Forces honesty.
- Data is shared. No data duplication.
- Transaction itself is receipt
- IOT & report integration

Solution Frameworks

Bitcoin and Ethereum were not architected to meet business needs of Finance domain. The Enterprise Blockchain frameworks that have the potential to address these issues (but not limited to) are

- Hyperledger
- Eris
- Ripple
- MultiChain
- R3 Corda

Why R3 Corda?

- **By Architecture it is a triple entry system.**
- R3 Corda's BlockChain inspired DLT product is crafted for Finance industry and automate payment processes.
- R3 was first adopted by Barcley bank and now supported by 100+ world leader banks and financial agencies so far.
- R3 Corda is open source and soon to be integrated to Hyperledger umbrella.
- **Ian Grigg, the inventor of triple entry accounting is now part of R3 CEV**

R3 CEV's Consortium: 42 Banks with Combined $600bn+ Market Cap; 60% Are Global SIFIs

SEB
BBVA
HSBC
MUFG
WELLS FARGO
CIBC
Goldman Sachs
RBS
UBS
Nordea
MACQUARIE
ING
BMO
citi
usbancorp
Morgan Stanley
TD Bank
Credit Suisse
Bank of America
BNY MELLON
Deutsche Bank
CommonwealthBank
State Street
BARCLAYS
RBC Royal Bank
SOCIETE GENERALE
BNP PARIBAS
MIZUHO
UniCredit
Santander
J.P.Morgan
SMBC
COMMERZBANK
Danske Bank
INTESA SANPAOLO
Scotiabank
NATIXIS
NOMURA
Northern Trust
Westpac
National Australia Bank

Source: CoinDesk

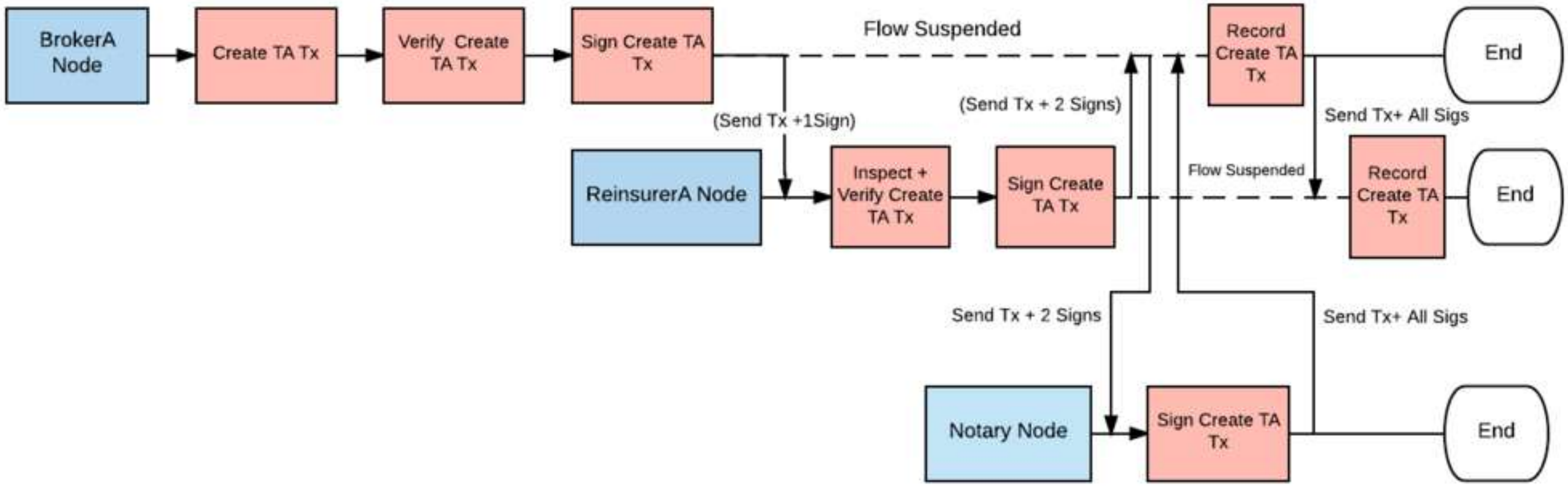
State of Bitcoin and Blockchain 2016 CoinDesk 99

Corda Architecture

- **Alice** issues the instrument, she signs it.
- **Bob** receives the instrument, agrees with it and he signs it.
- **Bob** passes the record to the **Notary**, who signs and then stores the instrument.
- **Bob**, **Alice** can now query the **Notary** to ascertain whether the transaction happened. **Bob** and **Alice** can also keep a copy of the notarised instrument themselves.
- Alice and Bob now always have the same data
- As the **Notary** is a SRL or Shared Ledger, it’s practically impossible to delete or update the instrument without counter-party approval.

Transaction Flow

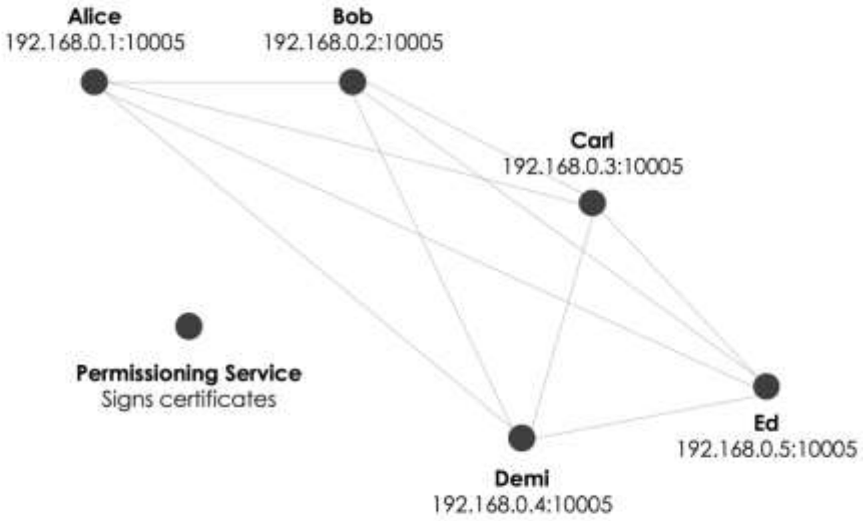
Flow 1: CORDA EBOT Create Tech Account (TA) Flow



Corda Is..

- Easier to install, easier to use
- Uses existing technologies: Java/Kotlin JVM languages, H2 SQL database, message queue
- Has excellent documentation & easier to climb the learning curve
- It guarantees privacy and scalability
- It’s open source

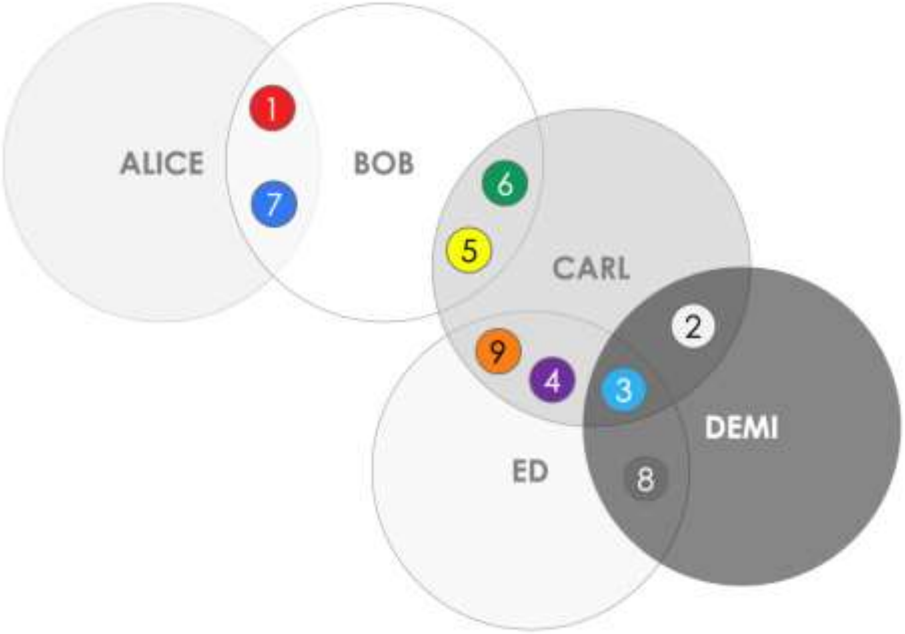
Network

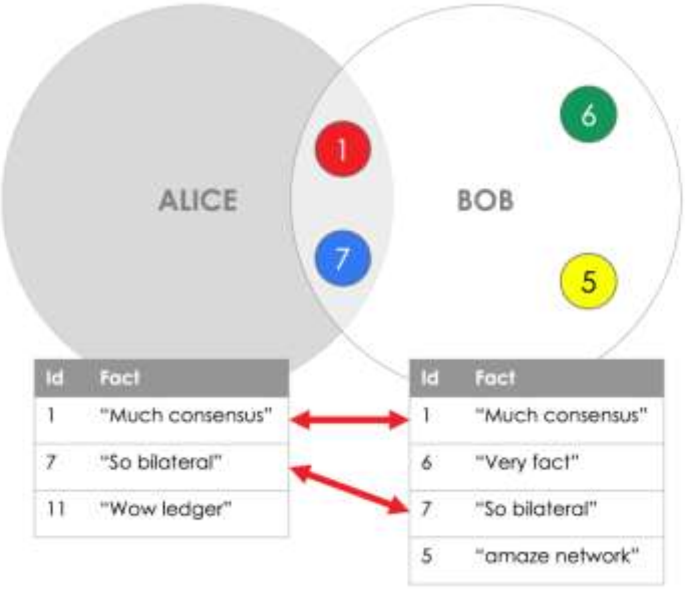


- *A Corda network is made up of nodes running Corda and CorDapps*
- *The network is permissioned, with access controlled by a doorman*
- *Communication between nodes is point-to-point, instead of relying on global broadcasts*

Ledger with No Central Data

- *The ledger is subjective from each peer's perspective*
- *Two peers are always guaranteed to see the exact same version of any on-ledger facts they share*



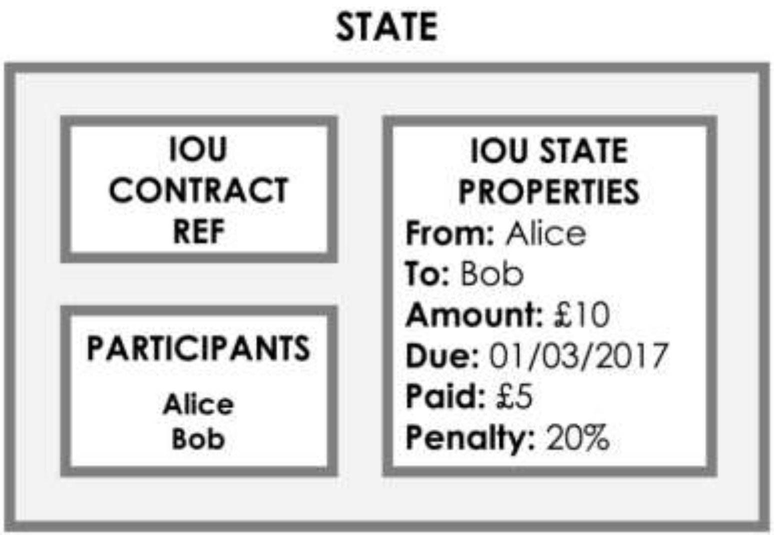


Id	Fact
1	"Much consensus"
7	"So bilateral"
11	"Wow ledger"

Id	Fact
1	"Much consensus"
6	"Very fact"
7	"So bilateral"
5	"amaze network"

States

- *States represent on-ledger facts*
- *Each node has a vault where it stores any relevant states to itself. Data is stored through ORM mapping to a H2 Java SQL database which is super-fast and open-source*
- *It tracks all the current and historic states that it is aware of, and which it considers to be relevant to itself*



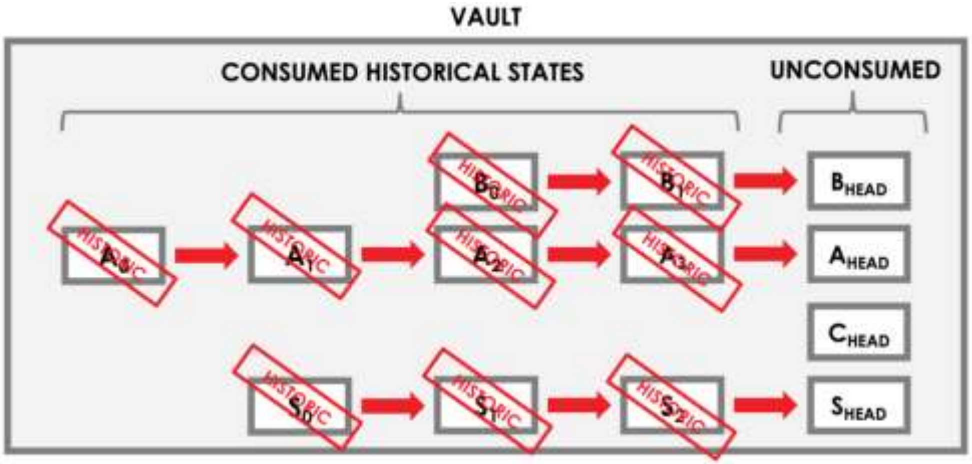
STATE

IOU CONTRACT REF

PARTICIPANTS

Alice
Bob

IOU STATE PROPERTIES
From: Alice
To: Bob
Amount: £10
Due: 01/03/2017
Paid: £5
Penalty: 20%



VAULT

CONSUMED HISTORICAL STATES

UNCONSUMED

A_0 HISTORIC → A_1 HISTORIC → A_2 HISTORIC → A_3 HISTORIC → A_{HEAD}

B_0 HISTORIC → B_1 HISTORIC → B_{HEAD}

C_{HEAD}

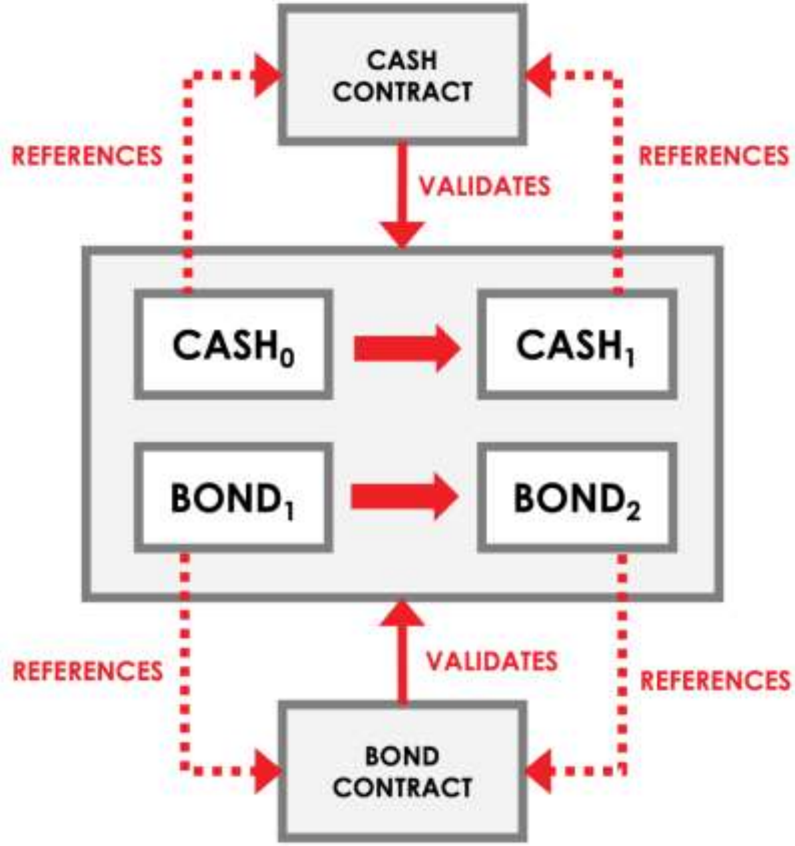
S_0 HISTORIC → S_1 HISTORIC → S_2 HISTORIC → S_{HEAD}

Contracts

- *A valid transaction must be accepted by the contract of each of its input and output states*
- *Contracts are written in a JVM programming language (e.g. Java or Kotlin)*
- *Contract execution is deterministic and its acceptance of a transaction is based on the transaction’s contents alone*

Contract Validity

- Each state points to a *contract*
- A *contract* takes a transaction as input, and states whether the transaction is considered valid based on the contract's rules
- A transaction is only valid if the contract of **every input state** and **every output state** considers it to be valid



Transactions

- *Transactions are proposals to update the ledger*
- *A transaction proposal will only be committed if:*
  - *It doesn’t contain double-spends*
  - *It is contractually valid*
  - *It is signed by the required parties*

Notaries

- *Notaries prevent “double-spends”*
- *Notaries may optionally also validate transactions*
- *A network can have several notaries, each running a different consensus algorithm*

Time Windows

- *If a transaction includes a time-window, it can only be committed during that window*
- *The notary is the time stamping authority, refusing to commit transactions outside of that window*
- *Time-windows can have a start and end time, or be open at either end*

DEMO

- **Obligations** - IOU CorDapp implemented in Java. Handles the transfer and settlement of obligations
- Retains participant anonymity using confidential identities (i.e. anonymous public keys)

Let’s run Corda on 4 nodes

i.e. One Controller (hosts the network map service and a validating notary service) &

Party A (http://localhost:10007/web/example/)

Party B (http://localhost:10010/web/example/)

Party C (http://localhost:10013/web/example/)

Thank You