Cognizant

August 2017

# Blockchain and DLT Introduction

Concept overview and technical introduction

## The need for a DLT based solution

Why do we need DLT?

# The Digital Age – Transfer of Information

*Information is today transferred seamlessly and instantaneously through the Internet*

## However when it comes to value transfer…

*There are various questions being asked..*

Is the Transaction **genuine**?

Is the Transactions **secure**?

Is the transaction final?

## A Trustless Environment – Not conducive for value transfer

*When it comes to value transfer, people don’t trust each other….*

## This leads to..

Need for an intermediator to provide trust

Need for reconciliation

Results in

Longer settlement cycle

Higher cost of transaction

**Multiple Isolated Ledgers:**
as the number of parties needing to reconcile increases, the number of reconciliations grows quadratically

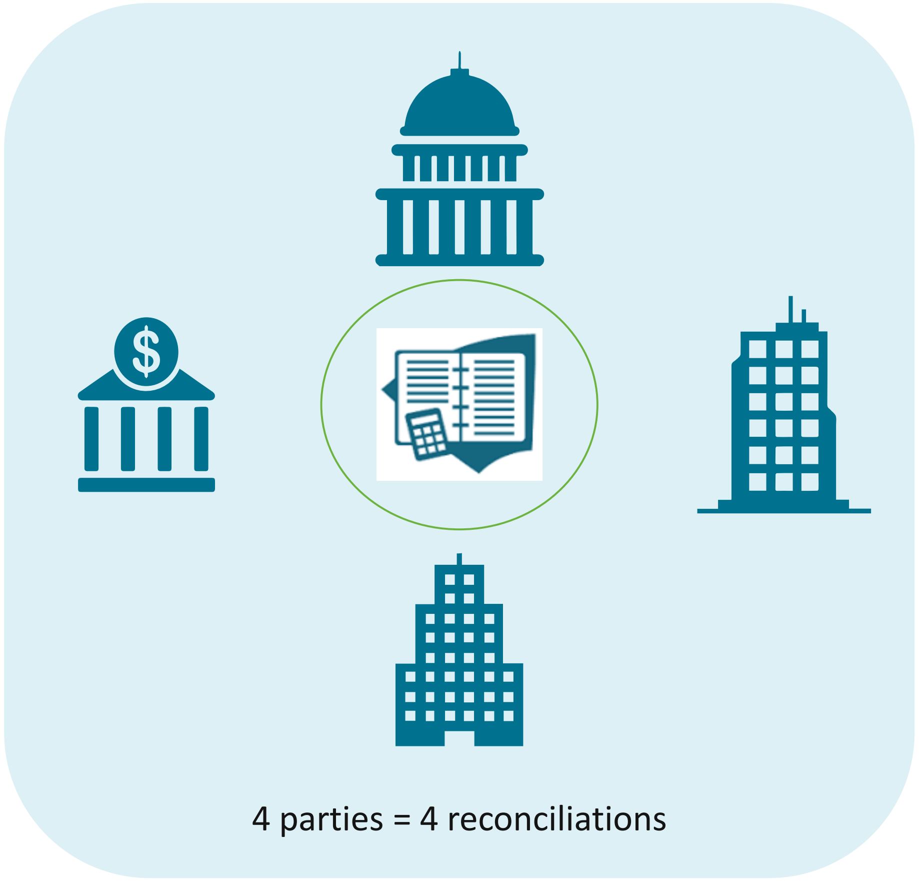
3 parties = 3 reconciliations

4 parties = 6 reconciliations

Cognizant

# Solutions for a streamlined process – Option 1

**Centralized Ledger:**
Reduces number of reconciliations required but creates central point of failure & intermediary costs



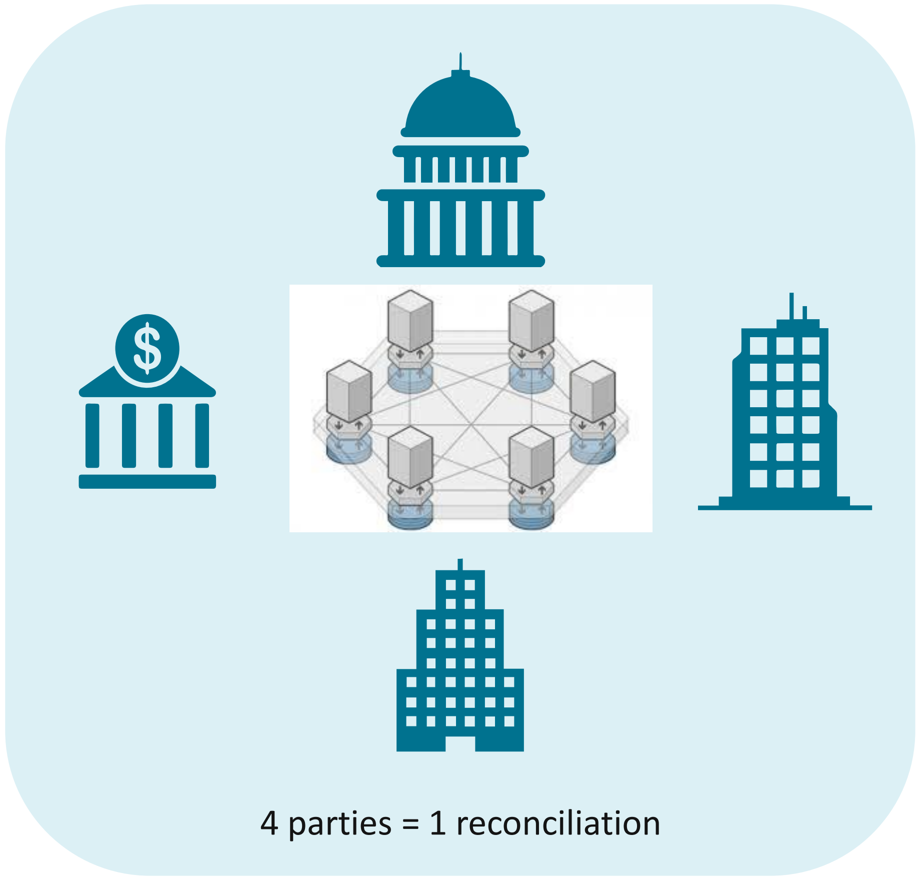
## Risks

- Single point of failure - Dependency on centralized party for data
- Risk of Data theft – Centralized solutions prone to hack and data theft risks
- Unclear data ownerships – Limited/no notion of who owns the data
- Limited visibility around the processing of the centralized solution

## Solutions for a streamlined process – Option 2

**Distributed Ledger:**

Eliminates need for reconciliation and eliminates single point of failure



**Benefits**

- Rules of ledger updates are coded on the platform which all participants follow
- Ownerships of assets are checked at the protocol level to ensure double spend cannot happen.
- Validation by peers ensures that no one can do something fishy.
- Only when a majority of stakeholders agree is when the transaction is finalized.
- Since transactions happen on a single source of truth, need of reconciliation is minimized.

## What is DLT?

Quick introduction of DLT technology and key constructs

# What is DLT?

**Database**

A Blockchain is a time stamped store of information

**Shared**

Single source of truth across all participating entities

**Distributed**

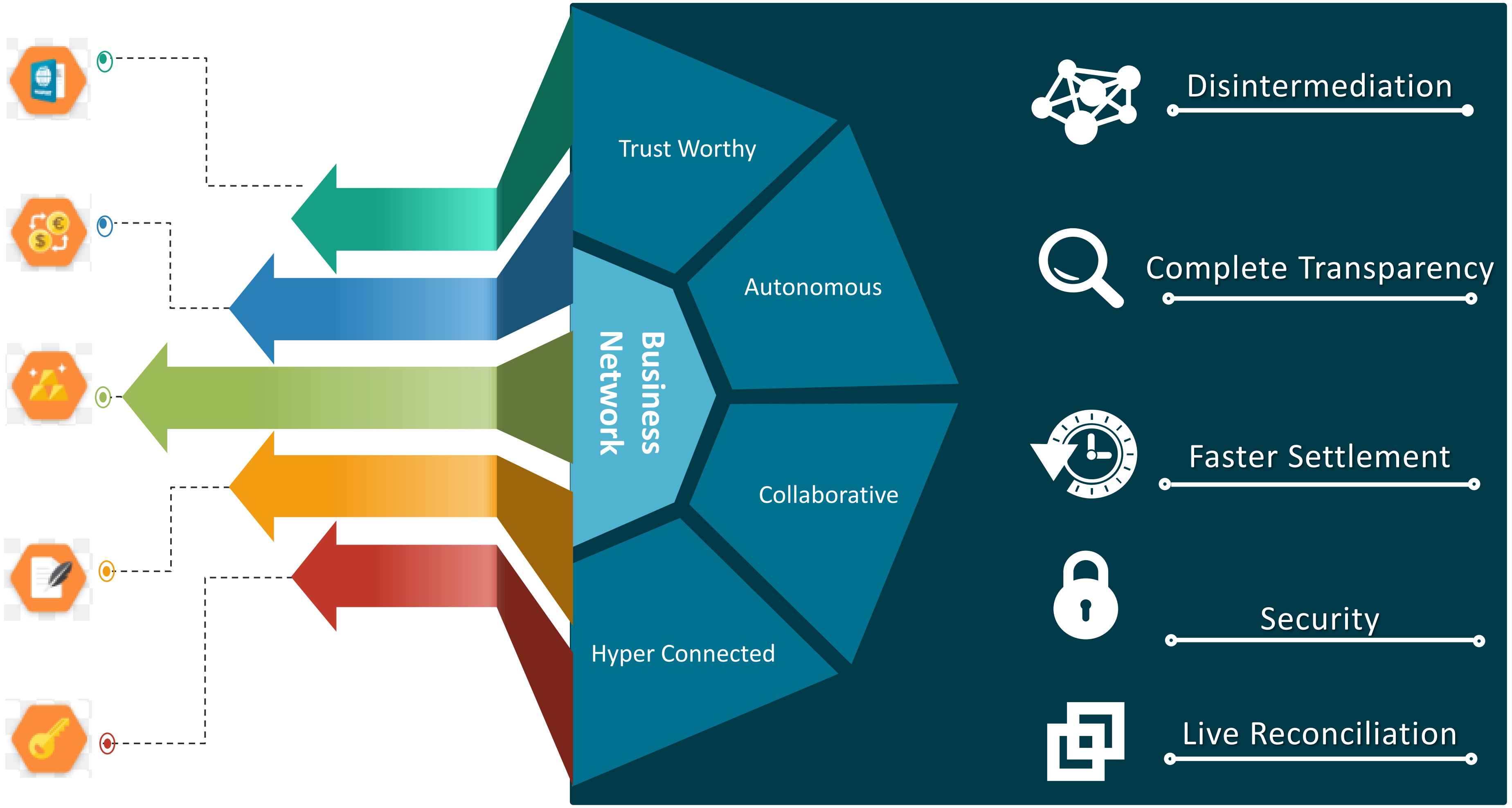
No central control or storage with equal participating entities

**Consensus**

Trusting & non-trusting entities transact through immutable agreements

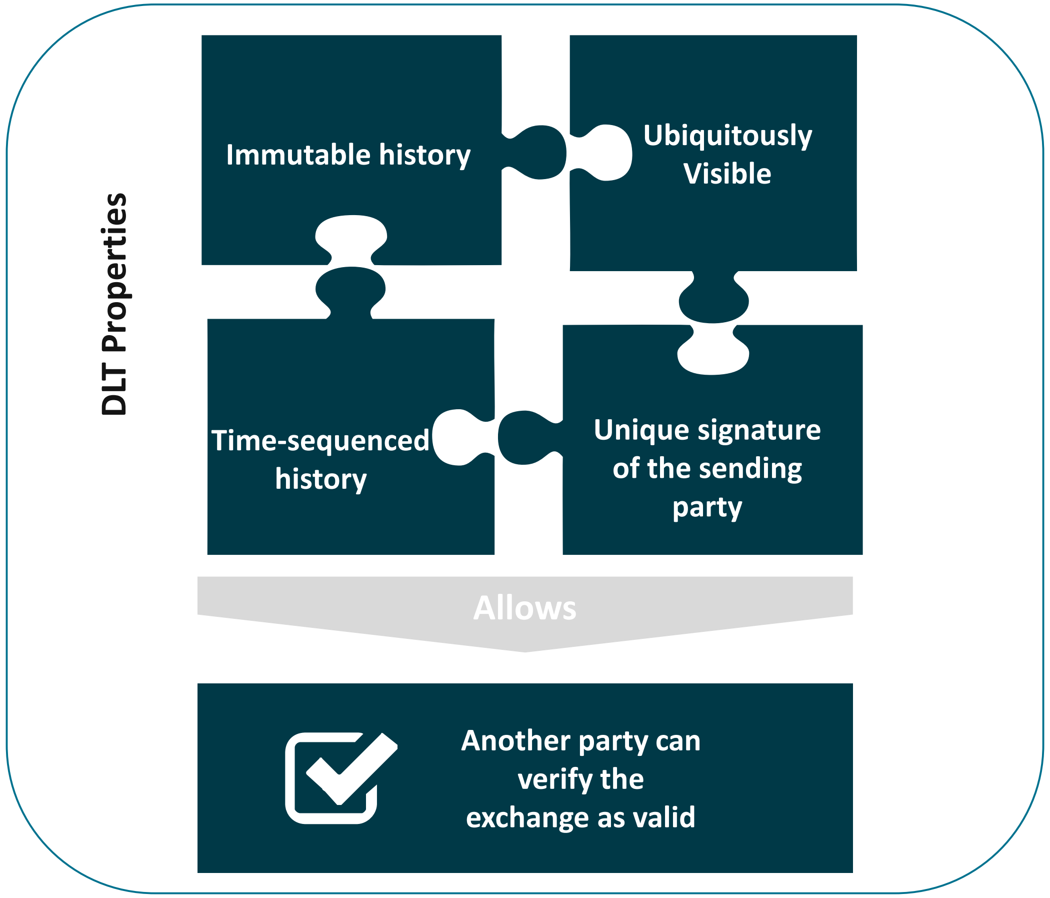
**Immutable**

Once written no one can change it

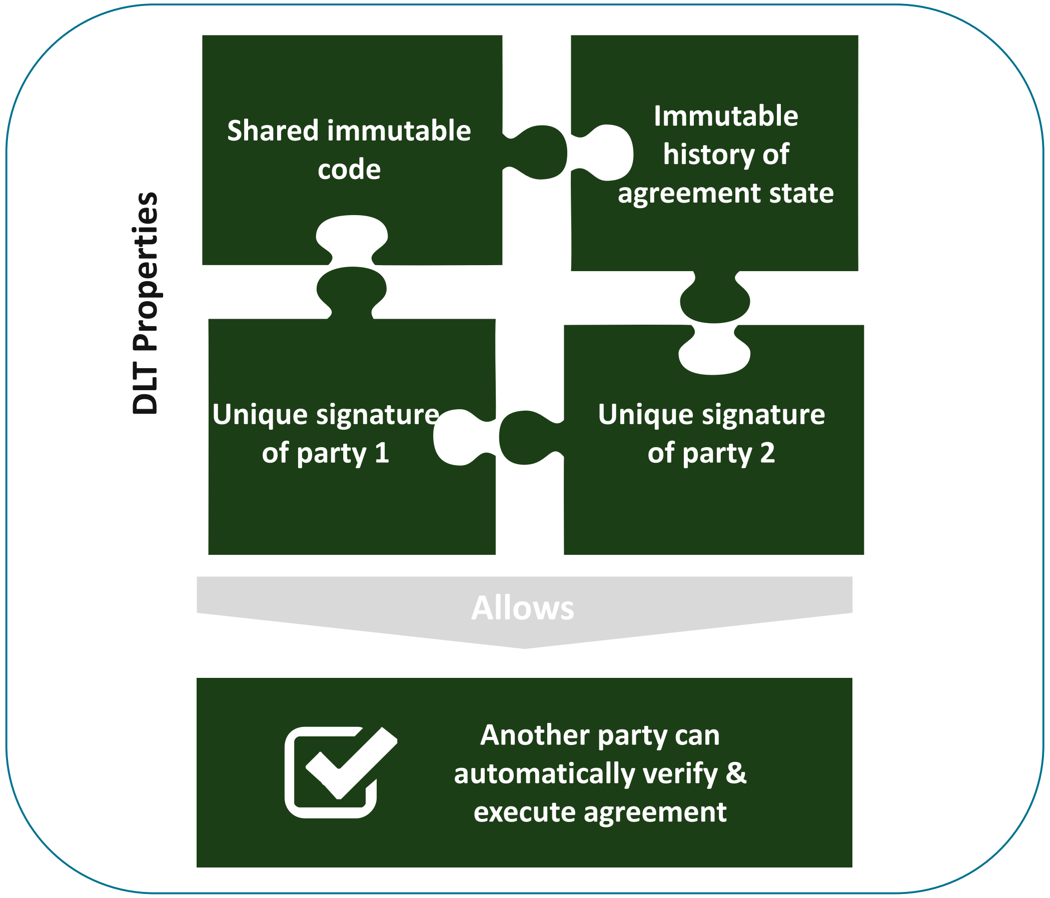


# Properties that allow untrusting parties to come to agreement

## Transfer of an Asset


DLT Properties
Immutable history
Ubiquitously Visible
Time-sequenced history
Unique signature of the sending party
Allows
Another party can verify the exchange as valid

## Satisfying an agreement

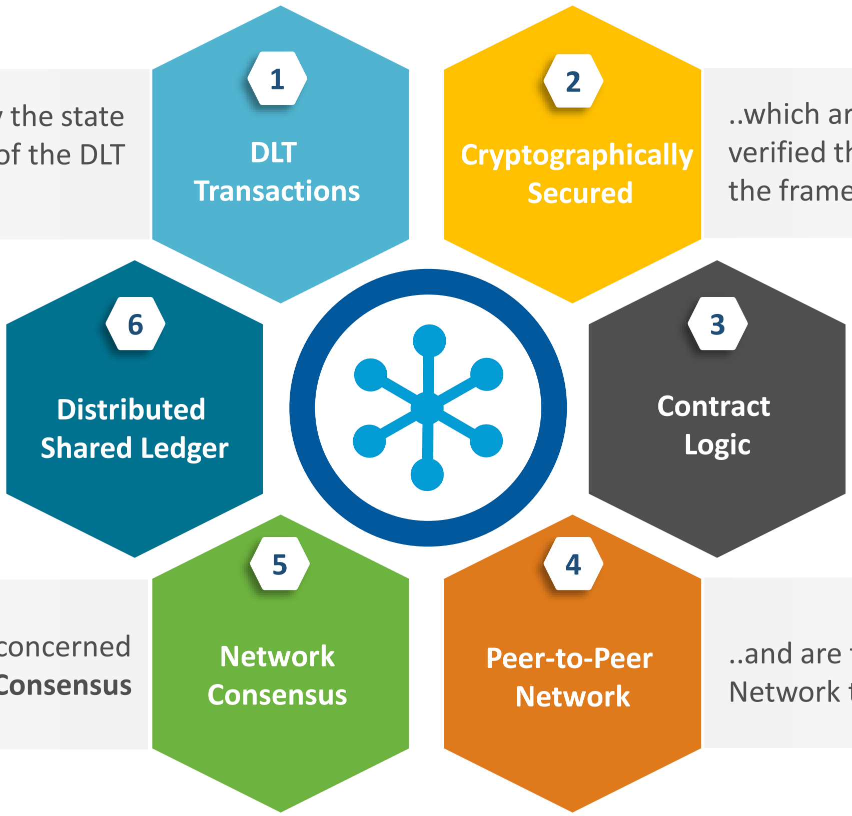

DLT Properties
Shared immutable code
Immutable history of agreement state
Unique signature of party 1
Unique signature of party 2
Allows
Another party can automatically verify & execute agreement

## Key Constructs of a DLT

What are the key constructs of a DLT solution…

Cognizant

# Key Constructs of a DLT Framework



A verifiable request, that wants to modify the state of the DLT

..which are secured, authenticated and verified through **Cryptography** provided by the framework

..leveraging distributed **Business rules** which are objectively verifiable and can be automated

..and are then sent through the **Peer-to-Peer** Network to another participant

..and are agreed upon by concerned participants through **Consensus**

..and are then added to the ledger in a chronological fashion as a **record of transaction** in a distributed ledger (book-of-records) shared across a business network

# The DLT Transaction

A request that is initiated with the purpose of changing the ledger state

## It Contains

**Sender's Signature**

A signature from the owner proving the ownership

**Transaction metadata**

Input/output or amount(dependent on the framework) that is part of the transaction.

**Transaction Id**

An Id which uniquely identifies the transaction on the network

**Participants' Information**

Details of Sender/Receiver or other participants that are part of the transaction

**Contract Logic**

Optional contract logic through which the metadata can be changed.

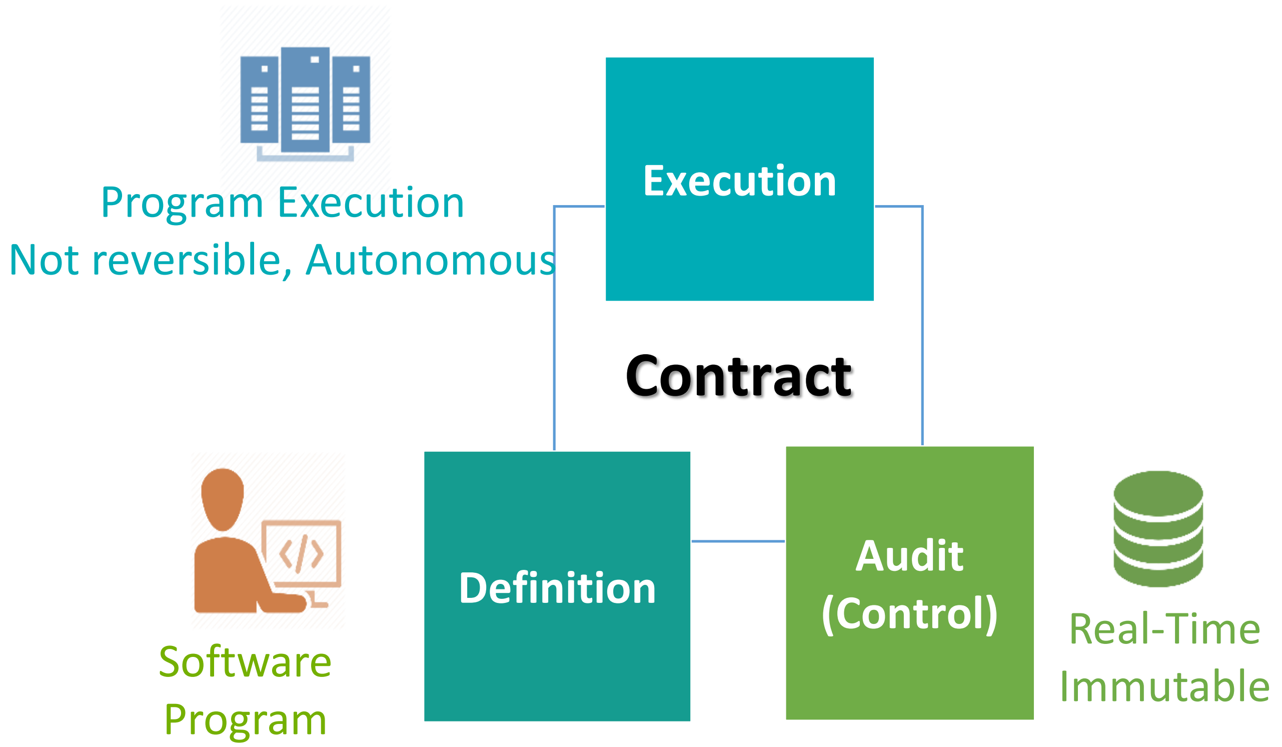
**Timestamp**

Specifying when the transaction is valid

# Which is secured through Cryptography…

| | |
|---|---|
| **Ownership Check** | • Sender has to provide cryptographic signature to prove ownership for the amount being spent.<br>• The verification is done at each validating node on the network. The result should match to arrive at agreement between nodes. |
| **Double Spend Check** | • Validating nodes have to validate that the amount being spent by the sender is not already spent earlier.<br>• The data present on the validating node  from previous transactions is used to verify this .<br>• The verification is done at each validating node on the network. The result should match to arrive at agreement between nodes |

## And additionally verified through Contracts..



- Decentralized business logic are rules for the contract **DEPLOYED** on the DLT, **AUTOMATICALLY** verifies the transaction and then **EXECUTES** the agreed terms.

- All DLTs have decentralized business logic built into the framework.
- Business Logic is stored on the DLT which all concerned parties have a copy of.
- It can execute agreed stored terms when triggered by an authorized event just like traditional systems to verify the transaction.
- All contract transactions are stored in chronological order on the DLT for future access along with the complete audit trail of events

## Which can be framework defined or user defined..

### Logic Optimized

**Provides a clear and efficient way to verifiably track business and governance process logic in a decentralized environment**

- A good fit to build arbitrary business processing logic

Frameworks have opened the platform for user defined contracts, where the user has flexibility to define their own contracts, deploy them on the DLT and execute them.

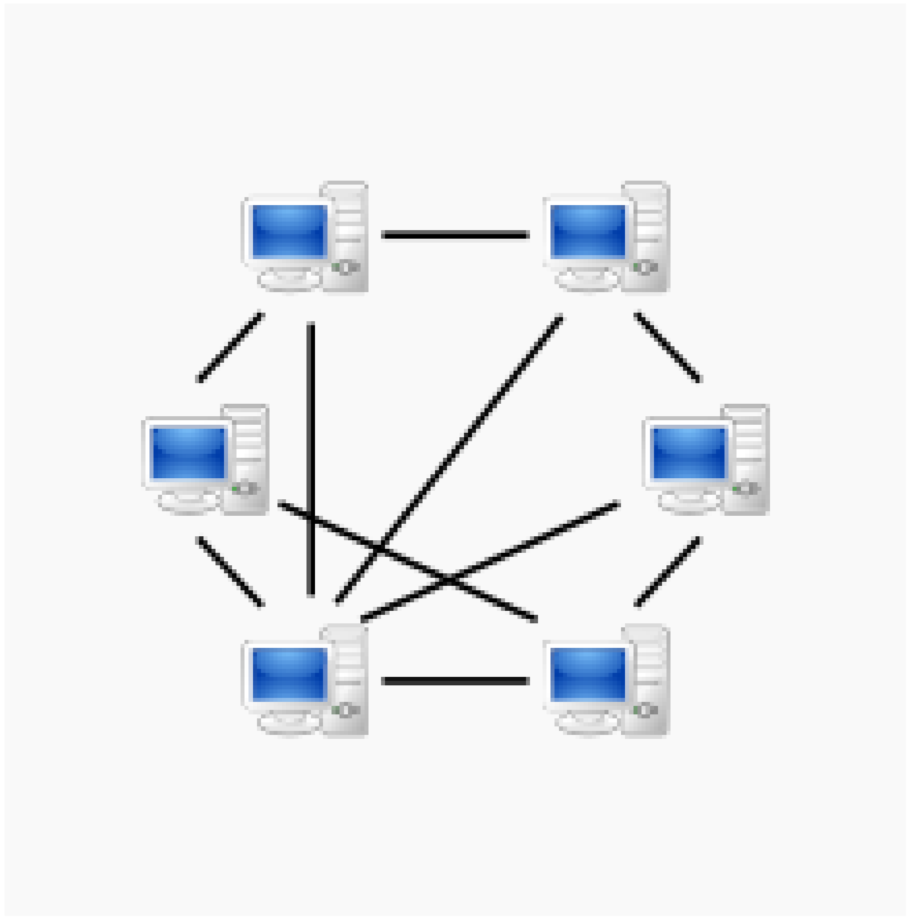
### Transaction Optimized

**Provides a clear and efficient way to verifiably track asset transfers in a decentralized environment**

- Typically provide limited logic capabilities (e.g. multi-signature)
- A good fit for application dealing with asset transfer, clearing & settlement and provenance application

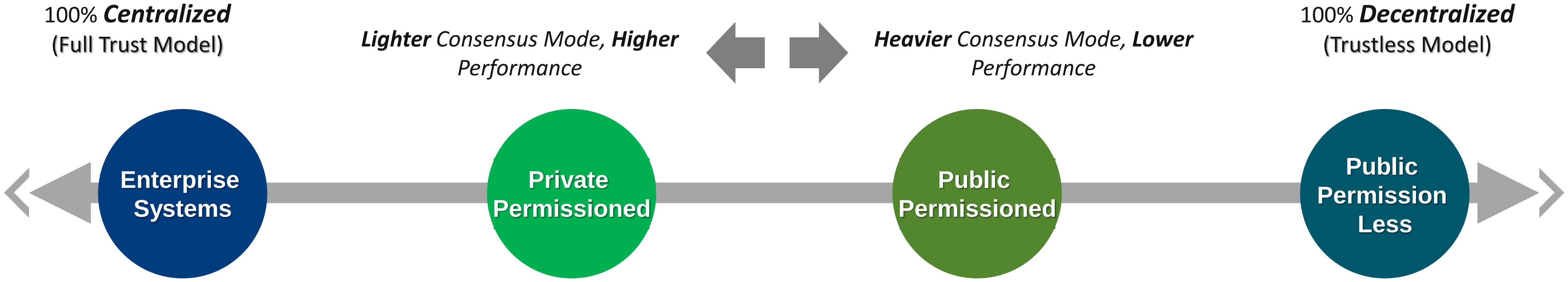
Frameworks that are specific to certain types of use cases (e.g. Asset transfer) have predesigned contracts that are tailored made for a particular purpose.

## And transferred through a network…



- A network comprises of nodes connected to each other in a **peer to peer** fashion.
- A **Node** is a Computer Running a **Protocol Software** that adheres to the guidelines of the network.
- Auto discovery of nodes is built into the framework, where nodes can connect to other nodes or connections can be managed through configuration flies. Some frameworks provide special service software called membership services which onboard nodes on the network.
- Nodes on the network are **together responsible** for managing the state of the ledger.

# Which can be of various types..



- **Centralized** Solutions under the control of an entity
- All Data sharing happens on **a full trust basis**

- **Decentralized** Solution for a group of **known actors**
- Permissions are granted for participation in transaction and verification process

- **Decentralized** Solution for a group of actors, **some of whom can be unknown**
- Permissions are granted for verification of transactions, transaction initiation and view does not require permission

- **De-Centralized** Solutions with all unknown actors working in the fully trustless model.
- All parties are capable of participating in transaction initiation and verification process

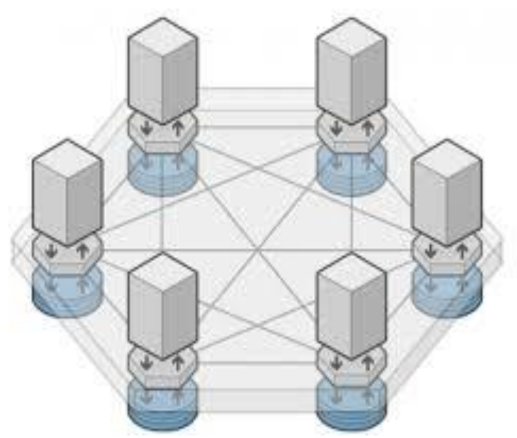
- **Public** : No restrictions on reading data (which may be encrypted) or submitting transactions for inclusion into the DLT
- **Private :** Direct access to Blockchain data and submitting transactions is limited to a predefined list of entities

- **Permission less :** No restrictions on identities of transaction processors (E.g.: Bitcoin, Ethereum)
- **Permissioned :** Transaction processing is performed by a predefined list of subjects with known identities

# On which participants agree through a consensus process…

## What is Consensus?

- A computer algorithm using which participants on a network agree on the validity of the state/transaction on the ledger.
- Algorithm used for Consensus varies from framework to framework.
- Type of network determines the complexity of the algorithm and the transaction throughput.



## Why is it needed?

- Consensus needs to solve for Byzantine Fault Tolerance allowing network coordination despite faulty nodes.
- Provides agreement on the last state of the ledger.
- Verification of future transactions uses the last confirmed ledger state.

## How does it happen?

- Each participants in the consensus process verify transactions based on business rules and the data present in the DLT.
- Since each participant has the same data and execute the same business rules, the outcome should be same.
- For consensus to be reached, the outcome should be same

In a decentralized architecture, participants have to agree to the rules and how to apply them, ensuring the **integrity**, **security**, and **consistency** of the DLT

**Transaction Validation** **Transaction Ordering**

## Who participates in it?

- Varies from framework to framework e.g. All participants on the network for public networks, to specific participants in a private network.
- Should allow for a successful agreement even when some participants are faulty.

## And then data is stored on the ledger…

**How is the data stored?**

- Data is time sequenced.
- Data structure used to store transaction varies from framework to framework.
- Some frameworks store it as a chain of blocks, others as individual states in a RDBMS

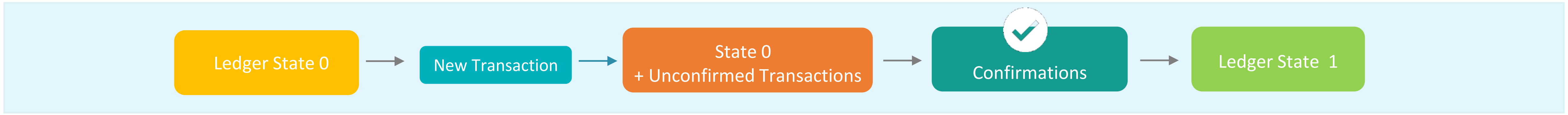
**Who stores the data?**

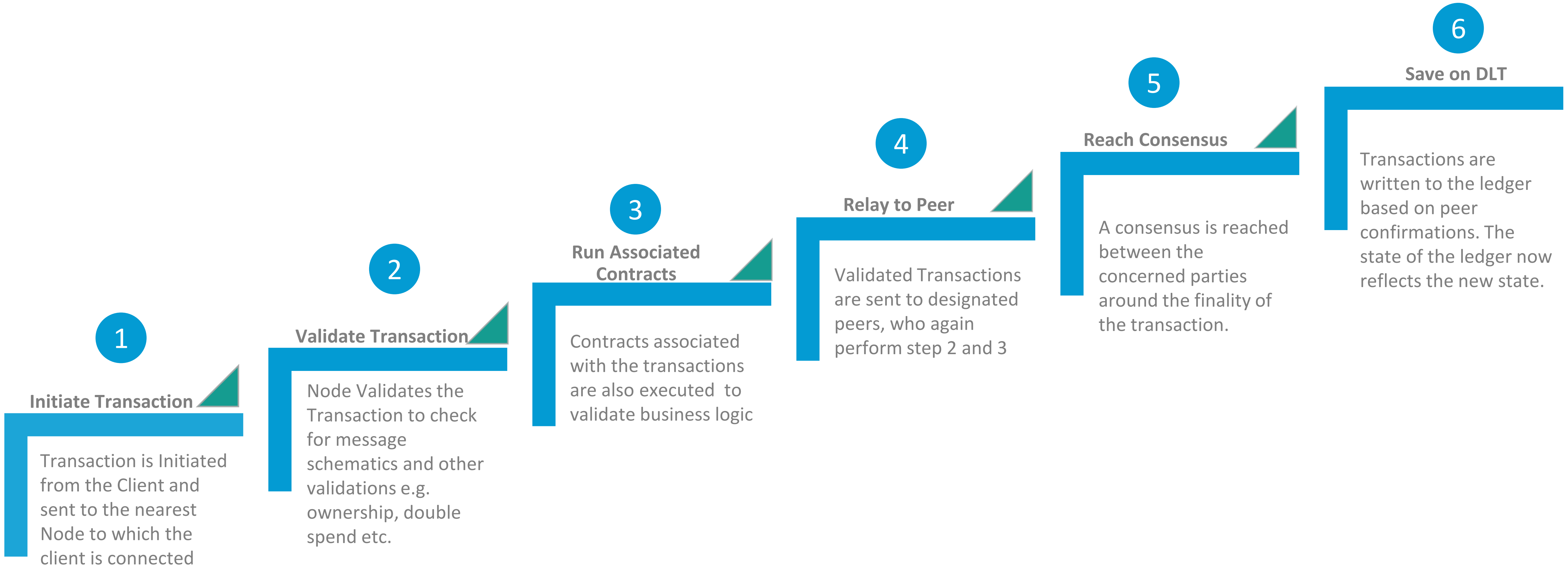
- Varies from framework to framework
- In Public ledgers all nodes store the data. In some private ledgers, only designated participants store the data.

**How is the data made tamperproof**

- Consensus process prevents a node from tampering with the data and creating new transactions off it.

# So in a nutshell, the transaction lifecycle on a DLT…





Transaction is Initiated from the Client and sent to the nearest Node to which the client is connected

Node Validates the Transaction to check for message schematics and other validations e.g. ownership, double spend etc.

Contracts associated with the transactions are also executed to validate business logic

Validated Transactions are sent to designated peers, who again perform step 2 and 3

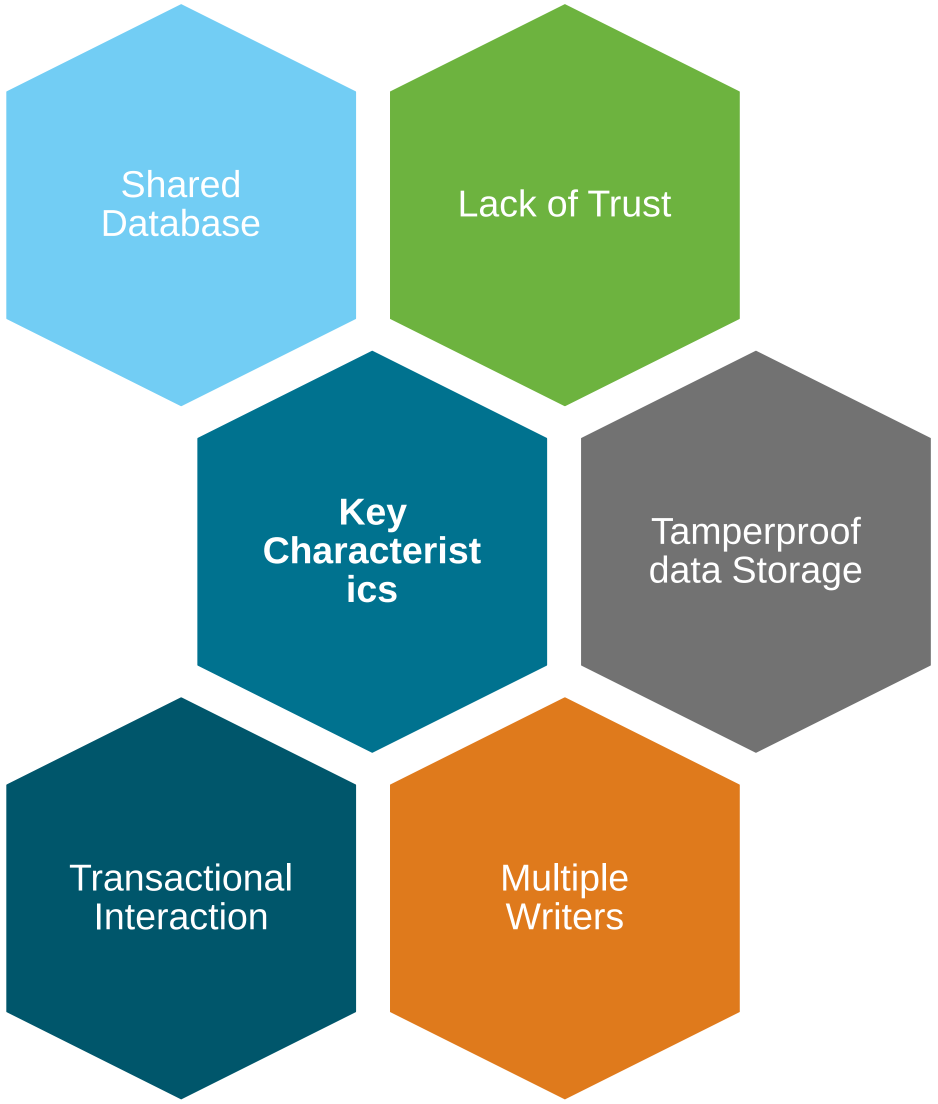
A consensus is reached between the concerned parties around the finality of the transaction.

Transactions are written to the ledger based on peer confirmations. The state of the ledger now reflects the new state.

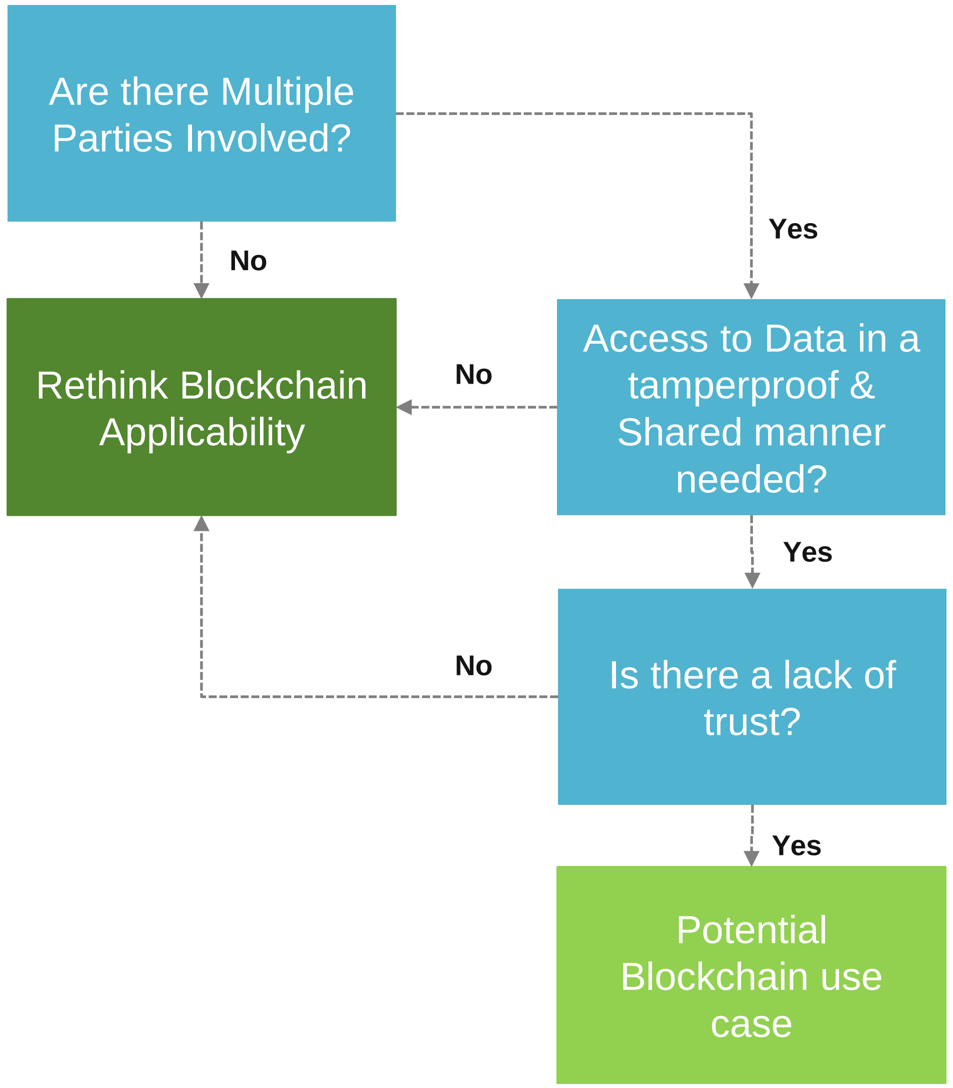
## Applicability of DLT

Key areas where DLT can be used

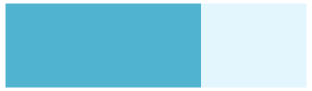
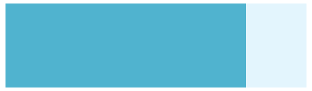
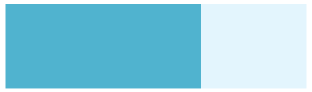
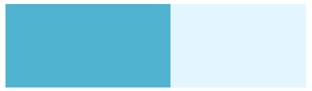
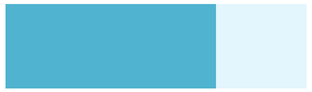
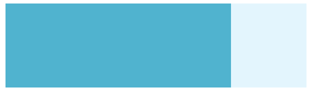
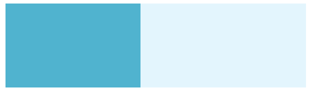
## Framework for selecting use cases



***Process Flow for Selecting a Blockchain Use case***


Are there Multiple Parties Involved?
Yes
No
Rethink Blockchain Applicability
No
Access to Data in a tamperproof & Shared manner needed?
Yes
Is there a lack of trust?
No
Yes
Potential Blockchain use case

# Focusing on developing solutions for Financial Services . . .

| | Size of Opportunity | Complexity | Use Case Time to Realization | | |
|---|---|---|---|---|---|
| | | | Near | Medium | Long |
| Payments |  | Low | Loyalty \| Cross-border payments \| Gift Card Issuance | Inter-bank Settlement \| Automated bill pay \|Micropayments | IoT \| Widespread bitcoin e-commerce |
| Capital Markets |  | High | Crowdsourcing \| Private equity issuance \| IPO registration | Repo Agreements \|Post-trade settlement\| Commodities\| OTC derivatives | T+0 settlement |
| Trade/Supply Chain Finance |  | High | Asset based financing \| LOC \| Anti-counterfeit \| Participation loan | Inventory tokenization \| Materials/ingredients tracking | AR/AP Smart Contracts |
| Investments & Loans |  | Medium | Credit score \| Escrow services \| Bond issuance | Syndicated loans \| Mortgage backed securities \| Auto lending | Decentralized P2P Loans |
| Compliance |  | Medium | KYC processing \| Bitcoin AML analytics | Transaction monitoring \| Digital, smart contracts | Blockchain government audits \| Taxes |
| Insurance |  | Medium | Underwriting \| Policy exchange/trading | Claims Processing \| Automated payout \| Policy holder authentication | Smart Contract policy execution \| P2P Insurance |
| Utilities |  | Low | Document management \| Asset digitization \| Identity management | Contract management \|Digital smart contracts | Shared databases |

# And other verticals…

| | | | | |
|---|---|---|---|---|
| RCG; Manlog | Inventory Tokenization | Counterfeit Prevention | Supply Chain Provenance | Inventory Tracking |
| Healthcare /Life Sciences | Medical Record Exchange | Drug Traceability | FDA Clinical Trials | Automated Claims Audit |
| Media/ Communications | Anti Piracy and Media rights | Royalty distribution | Micro payments | Decentralized Advertising |

| | | |
|---|---|---|
| Energy; Telecom | Peer to Peer Trading | Smart Grid Infrastructure |
| Education | Credential Issuance | Credential Verification |
| Travel & Hospitality | Travel Distribution and Settlement | |

## Evolution of DLT

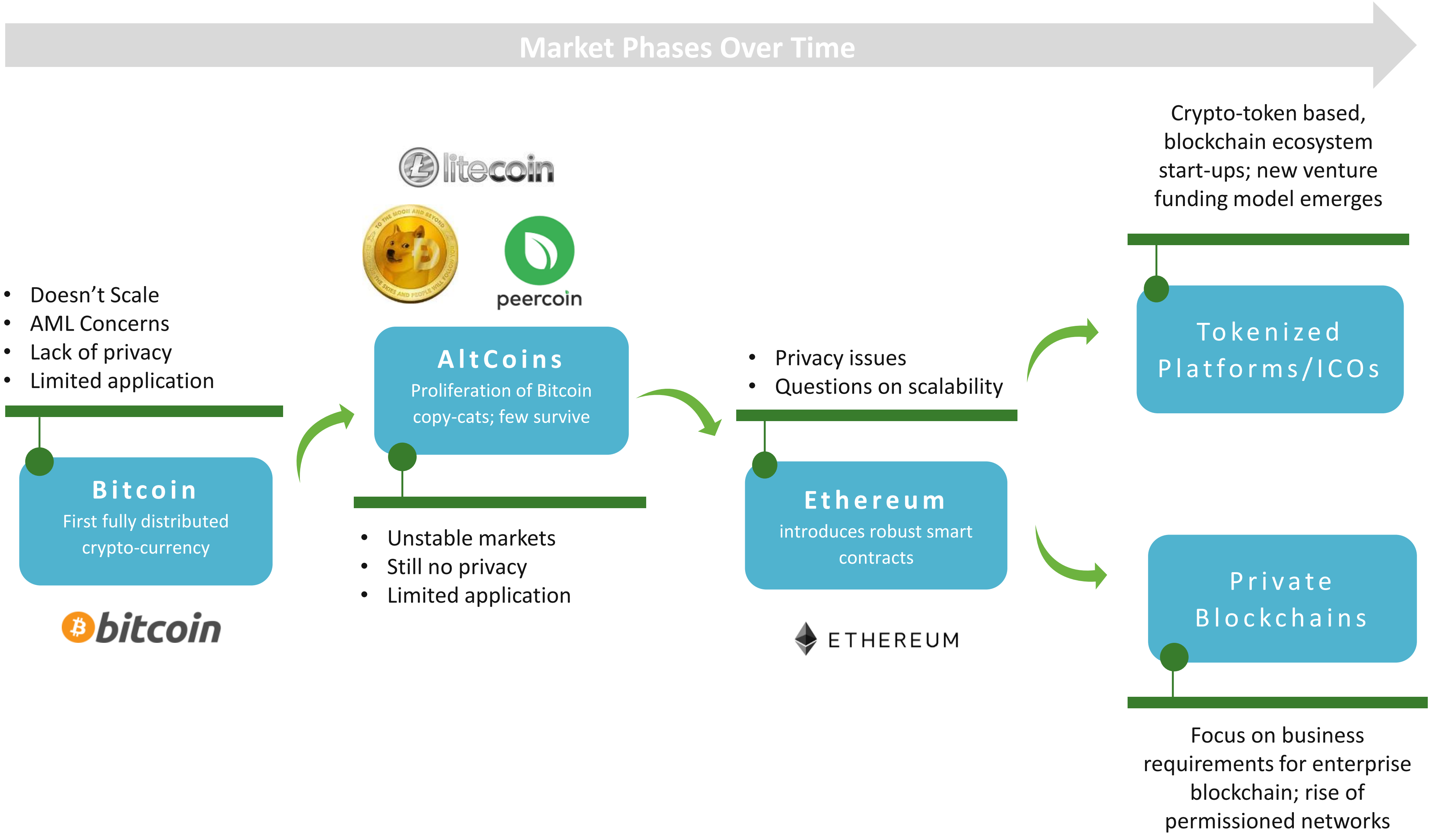
How has DLT evolved over the years

# Evolution of Blockchain frameworks..

**$2.3b**
Estimated blockchain market size by 2021

**61.5%**
CAGR from 2016 to 2021

**85%**
Of Fis surveyed indicated blockchain will have considerable or transformative impact on their industry



**Bitcoin**
First fully distributed crypto-currency

- Doesn't Scale
- AML Concerns
- Lack of privacy
- Limited application

**AltCoins**
Proliferation of Bitcoin copy-cats; few survive

- Unstable markets
- Still no privacy
- Limited application

**Ethereum**
introduces robust smart contracts

- Privacy issues
- Questions on scalability

**Tokenized Platforms/ICOs**

Crypto-token based, blockchain ecosystem start-ups; new venture funding model emerges

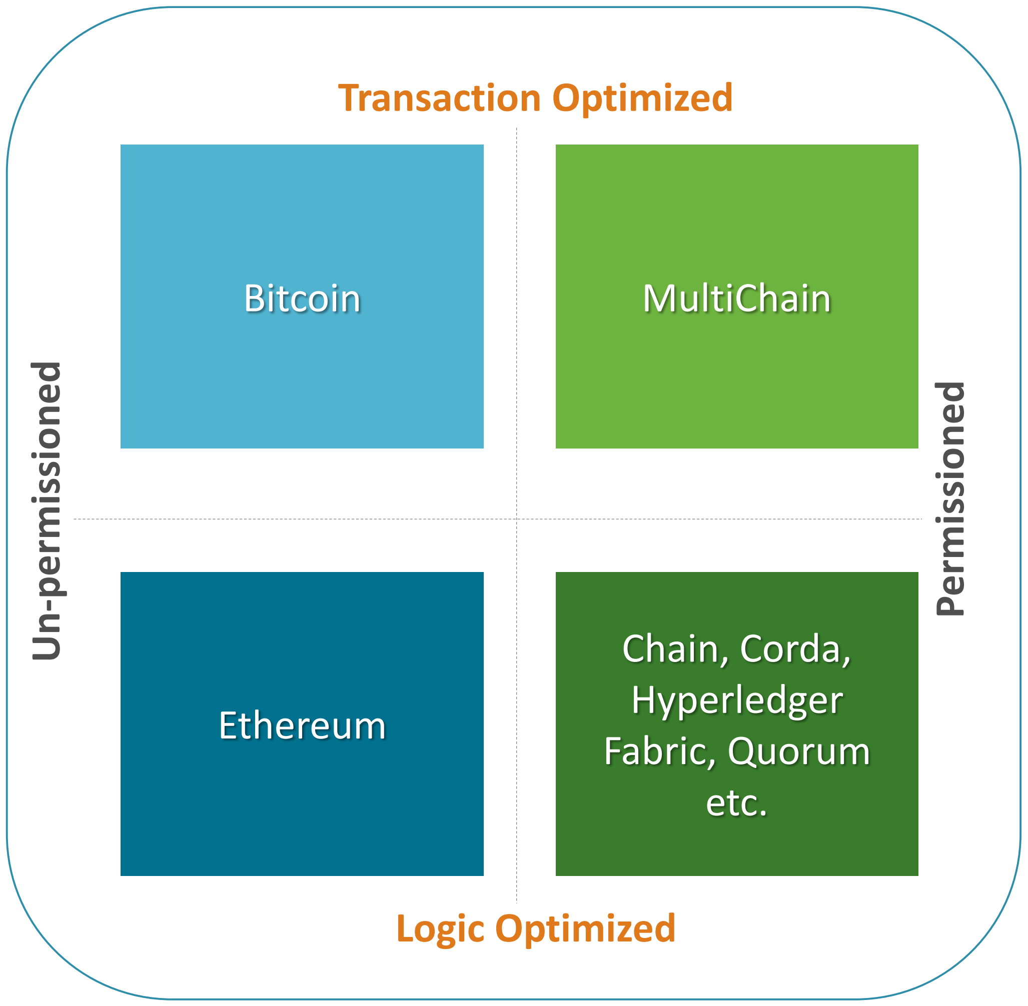
**Private Blockchains**

Focus on business requirements for enterprise blockchain; rise of permissioned networks

Cognizant

# Positioning of some key platforms.

## DLT Platform positioning



## Innovations by Private Permissioned Platforms

### Open to anyone?

**Innovation**: Permissioned blockchain networks

- Participants are authorized by the governance structure
- Akin to an intranet or shared utility

### Role of crypto-currencies?

**Innovation**: token-less consensus methods

- Tokens are used for incentivizing consensus in public networks
- PBFT & Round-robin are some examples of token-less methods

### Role of Miners?

**Innovation**: Network governance models

- No unknown 'miners' in permissioned blockchain
- Participants of network agree to governance model to achieve consensus without miners

### Privacy Concerns?

**Innovation**: Channels & Notaries

- New innovations in some platforms allow for agreements not to be broadcast to the entire network

Cognizant

## HyperLedger Fabric

Smart Contract Framework By Linux Foundation

# HyperLedger – History and Overview

**HYPERLEDGER**

- **Open-source, cross-industry** private, permissioned blockchain initiative
- Managed by the **Linux Foundation**
- **100+ members**, including Amex, DTCC, CME group, JPMorgan, IBM, R3, Wells Fargo, Bank of England, Daimler AG etc.

## Key Innovations

**Transaction Channels**

Channels provides transfer of data only to parties in the channel. Ledger is maintained at the Channel Level

**Ordering Services**

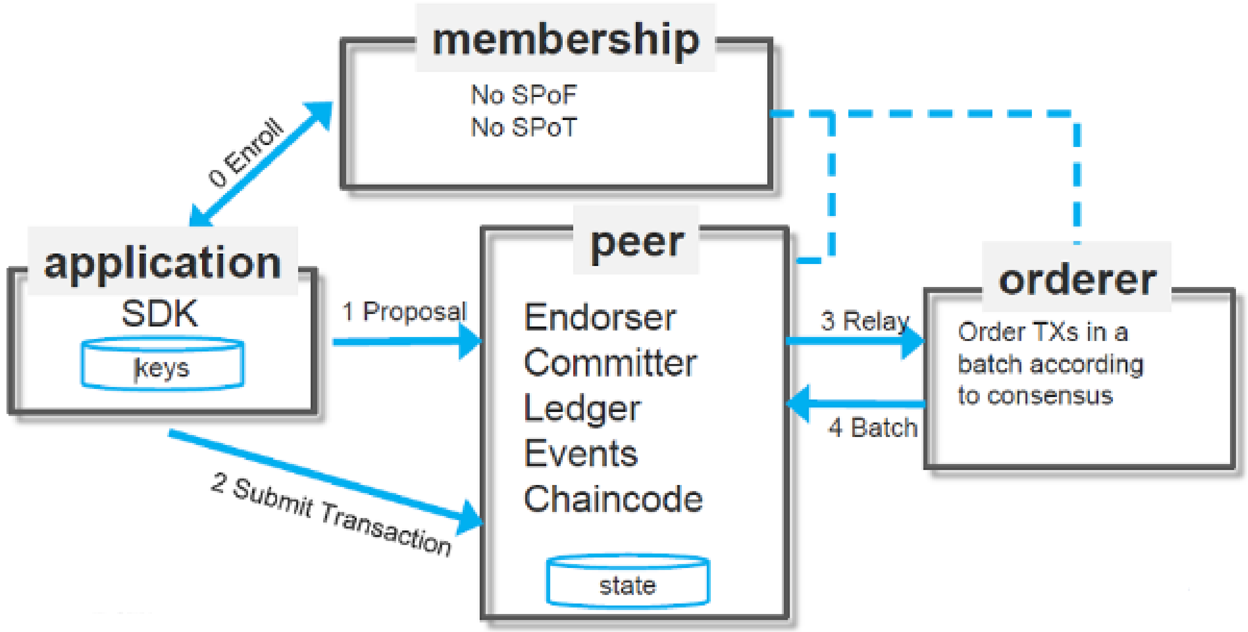
Introduction of designated Ordering Services ordering transactions

**Enterprise Search**

Pluggable database like CouchDB to enable searches

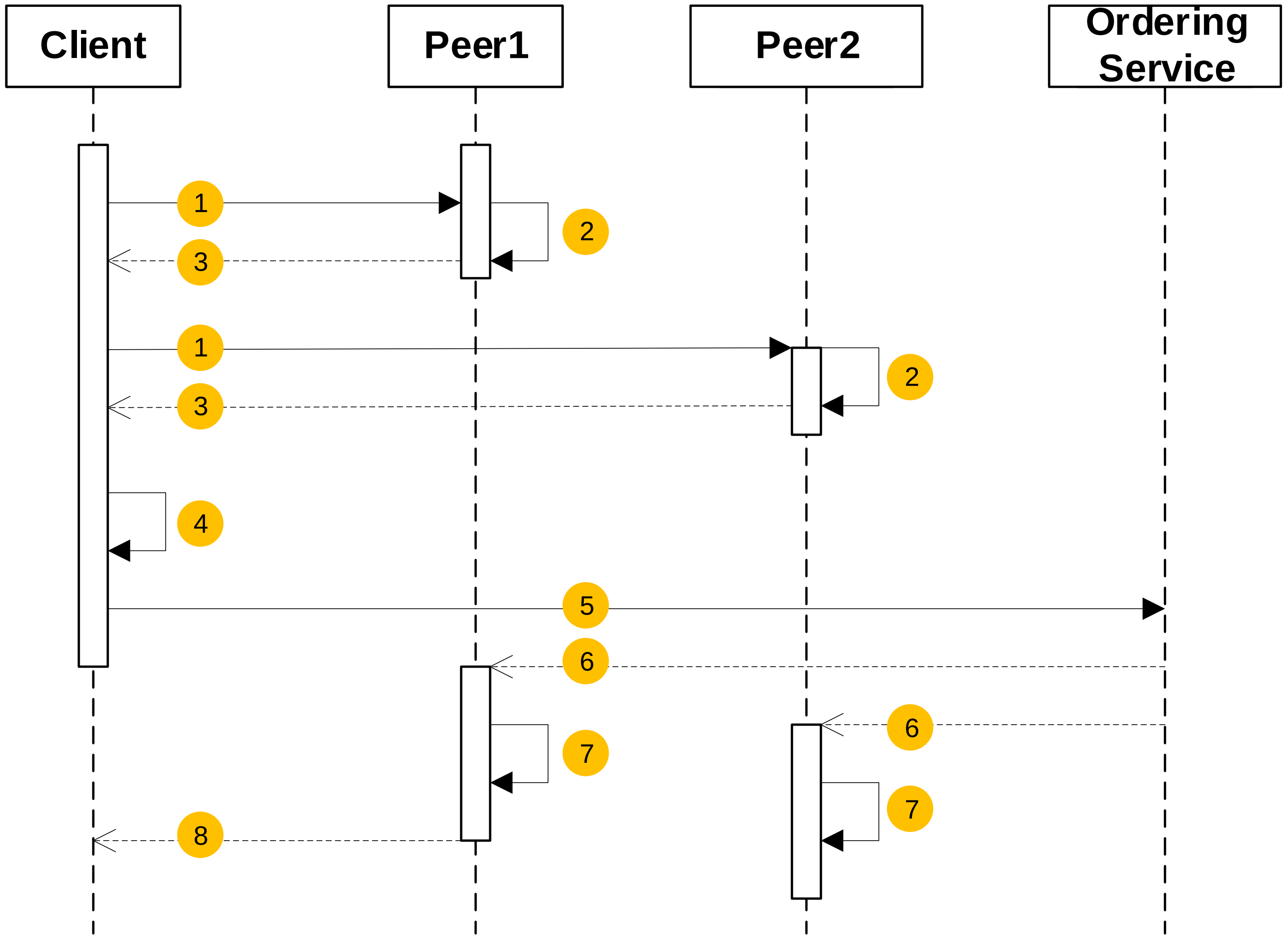
**Data Encryption**

Certificate based data encryption



| Criteria | Public or Private | Transaction or Logic Optimized | Built for Industry? | Smart Contract Language | Multi-Sig Transactions | Underlying Currency needed | UTXO or Account Based | Permissioned | Transaction Propagation Model | Data Security | Query Capabilities | Rollup Support | Consensus Mode |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| HyperLedger | Private | Logic | No | Java, Go | Yes | No | UTXO | Yes | Channel | Certificate Based | Yes | Yes | Pluggable |

# Transaction Lifecycle in HyperLedger



1. **The client initiates the transaction**- The client needs to send the transaction to both Peer1 and Peer2 as both of them have been marked as endorsers for the transaction.
2. **Endorsing peers verify signature and execute the transaction** – The endorsing peer does the following
3. **Endorsing peer returns the output** - The set of these values, along with the endorsing peer's signature is passed back as a "proposal response" to the SDK which parses the payload for the application to consume.
4. **Proposal responses are inspected -** The client application verifies the endorsing peer signatures, compares the proposal responses to determine if the proposal responses are the same. If the client application intends to submit the transaction to Ordering Service to update the ledger, the application determines if the specified endorsement policy has been fulfilled before submitting.
5. **The client assembles the endorsements into a transaction** - The client "broadcasts" the transaction proposal and response within a "transaction message" to the Ordering Service. The transaction will contain the read/write sets, the endorsing peer's signatures and the Channel ID.
6. **The transaction is validated and committed** – The ordering service delivers the block to all peers on the channel. The transactions within the block are validated to ensure endorsement policy is fulfilled. Transactions in the block are tagged as being valid or invalid.
7. **The ledger state is updated** - Each peer appends the block to the channel's chain, and for each valid transaction the write sets are committed to current state database.
8. **Notification event sent to the client** - An event is emitted, to notify the client application that the transaction (invocation) has been immutably appended to the chain, as well as notification of whether the transaction was validated or invalidated.

# Hyperledger Components for Design Considerations

## Nodes : 3 Types

- **Client** : Clients are the end-user facing nodes. Hyperledger Fabric provides multiple interfaces to the blockchain (SDKs and CLI). These are presented by the client nodes.
- Users (application software) send transaction requests: **Invoke** or **Query** to the Fabric network using these interfaces via a client node which in turn connects to Peer on blockchain
- Invoke calls potentially state-changing chaincode and query call read-only chaincode function
- **Peer** : Peers maintain the state of the ledger. Peers execute chaincode and participate in consensus formation. **Chaincode** is installed on peers
- A Peer does not create new blocks but it validates blocks and transactions. Transaction validation is performed by simulating transactions and applying the **endorsement policy**.
- **Ordering Service (OS) Node :** The OS creates new blocks by ordering the transactions. It provides a shared communication channel to clients and peers. It has two services: 1 Broadcast: to inject messages into the system. 2. Deliver: for receiving ordered batches
- **Solo** ordering service: Primarily for development and testing without consensus using a single central authority. Kafka for ordering without Byzantine Fault Tolerance

## Ledger & State Database

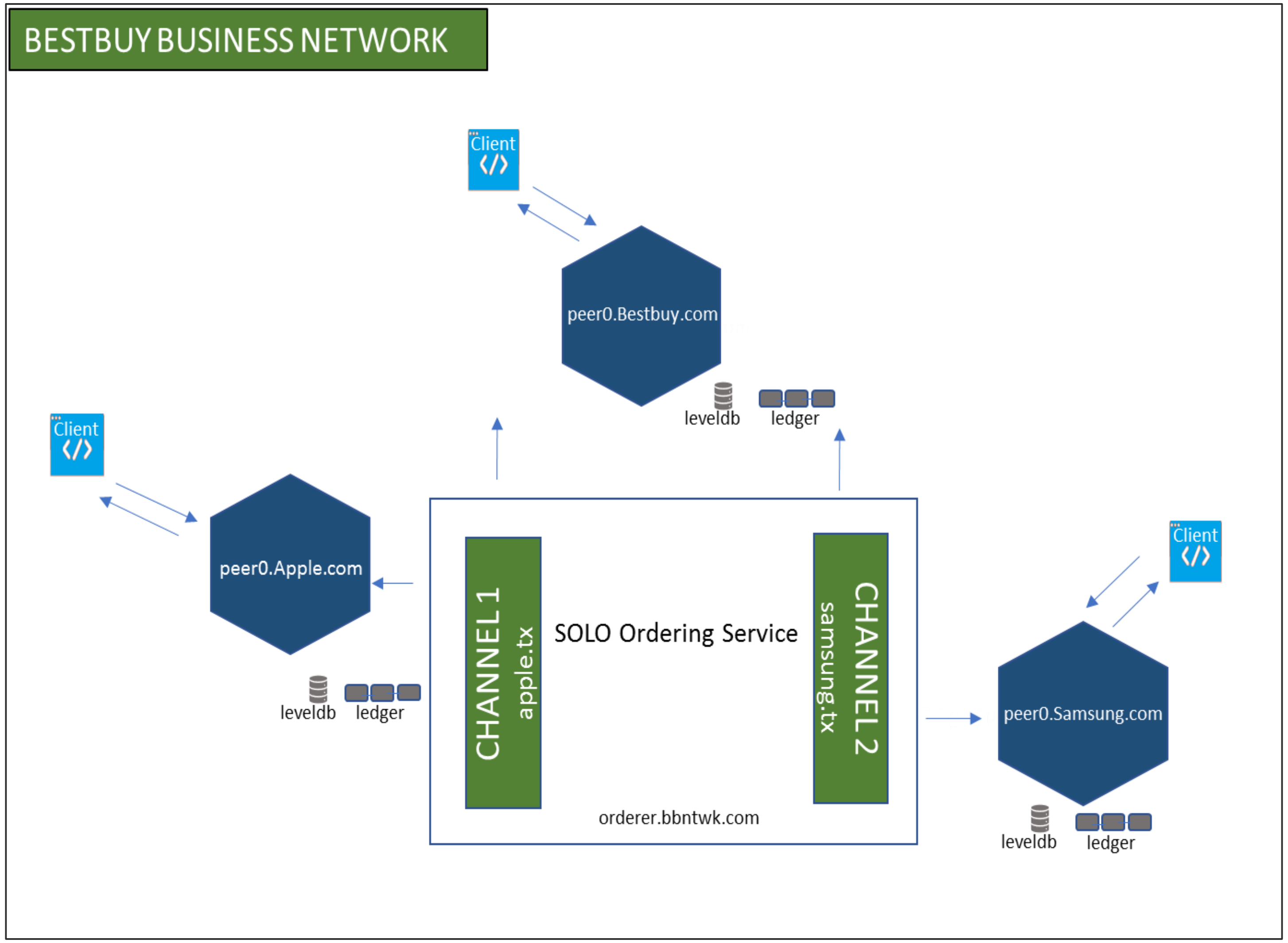
- The **ledger** is maintained and validated by the Peers. The ledger includes the history of all transactions. This can be used to reconstruct the world state (also called state database).

- **State databases** on each node contain the latest key values agreed upon by the nodes on each channel. Local state databases support efficient chaincode interactions. CouchDB and **LevelDB** are both supported for actual storage of the state database.

## Channels

- Channel is a kind of separate chain. It includes only the transactions made in the channel. Channels also play an important role in ensuring privacy by restricting read access to authorized signatories. Lastly, channels provide a means to control deployment of chaincode which is Hyperledger Fabric's implementation of Smart Contracts.

# Use case for Our Demo today



**Use Case Description**

1. A business network run by Bestbuy to procure cellphones
2. We have two cellphone manufacturers Apple and Samsung
3. The platform allows the manufactures to submit the new devices
4. New devices are then reviewed by Bestbuy
5. Best Buy will either purchase or return the device.

**Network Components**

1. A peer for each member i.e. 3 Peers in total
2. Pre-created certificates and one orderer, using SOLO
3. 2 Channels (1 each for apple & samsung).
4. We will call the network BBNTWK

*Think! Why do we need separate channels*

**Network Functions**

1. Each vendor should be able to:
   - **createCell**: a vendor writes into the ledger when a new cellphone is created. The key shall be the serialnumber of the cellphone.
2. BestBuy should be able to:
   - **buyCell**: BestBuy buys a cellphone from a vendor to sell it to the market.
   - **returnCell**: BestBuy wants to return a bought Cellphone.

Cognizant

# Working with HyperLedger Step 1 – Generating Certificates

## Why is needed?

- Every node on the network including orderers and peers need a certificate to sign transactions.
- In Production this can be done through CA authorities. In development to assign certificates to each node

## How is it generated?

- Using the **cryptogen** tool and the **crypto-config.yaml** to generate certificate to each participating node.
- crypto-config.yaml contains the network topology that is being set for the network.

## What is the Output?

- Under the **crypto-config** folder certificates are generated for each node.

## Important Aspects of the crypto-config.yaml configuration

| | |
|---|---|
| Orderer Orgs | Definition of organizations managing the Orderer Nodes |
| PeerOrgs | Definition of organizations managing peer nodes |
| Specs | Used in case explicit definition of hosts need to be done e.g. provide for a specific host name |
| Template | Can be used instead of a Specs to automatically generate host name e.g. peer%d where “%d is replaced by count(1,2,3) per organization |
| Users | Number of user accounts in addition to the Admin account |

## Working with HyperLedger Step 2 – Generating Transaction Config

### Why is needed?

- Create the genesis block for the orderer.
- Set up the channel configurations blocks
- Set up anchor peer transaction

### How is it generated?

- Using the **configtxgen** tool and the **configtx.yaml**.
- **configtx.yaml** contains specifications for the genesis block, channels, the mapping of the channels with the Organizations and the Orgs with the certificates generated

### What is the Output?

- Genesis block
- Channel transactions
- Anchor Peer transactions

OrdererGenesis

Definition of orderer organizations, the consortium that would use this genesis block and the consortium members

OrgChannel

Name of the Channel and the list of organizations from the consortium who would participate in it.

Organizations

The mapping of the Organizations to the Membership Service Provider(MSP) certificate generated.

OrdererDefaults

Default parameters for the orderer, e.g. orderer type of solo or using kafka, address of the orderer, batch size, list of orgs defined as participants on the orderer side.

ApplicationDefaults

Defaults for list of orgs defined as participants on the application side

# Working with HyperLedger Step 3 – Other Configurations

## docker-compose.yaml

- Set-up the docker-compose environment configurations for each of the peer entities to use during load.
- For example which Fabric image for orderer to use, the peer.yaml file to use to configure each peer, the port numbers on which docker images for each participants would run, the dpendency between each of the docker images(e.g. peer dependent upon docker)
- Also included is the CLI configurations, which allows each docker instance to be exposed through Command Line interface

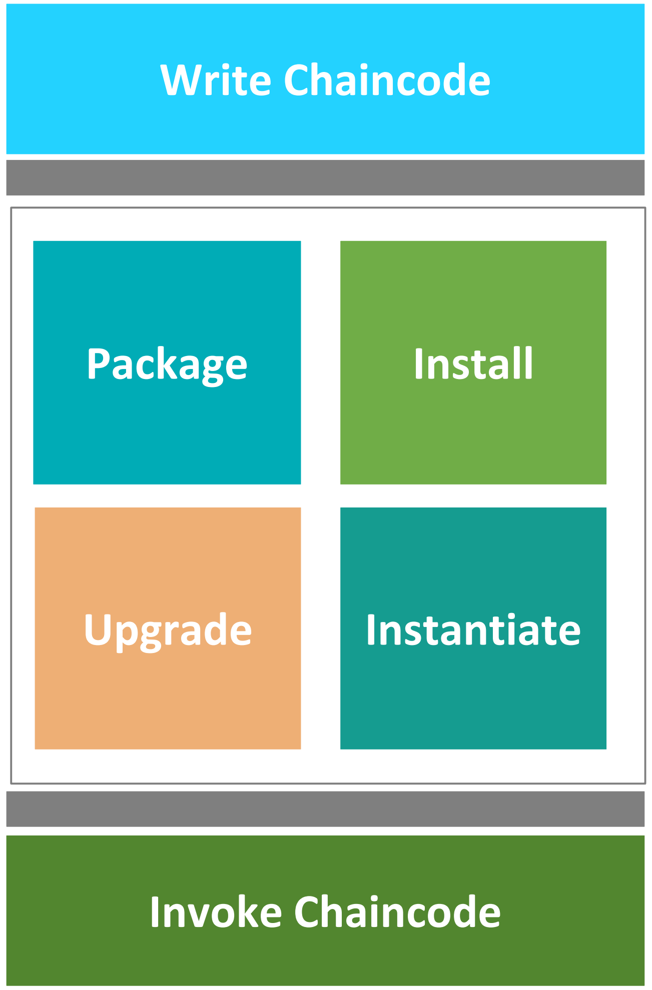
## cli.yaml

- Contains the details of the configurations that the are pertinent to cli interface.
- Can be part of the docker-compose itself, but provides modularity.

## peer.yaml

- Contains the details of the configurations that the peer would need, which Fabric image each peer would use.
- Can be part of the docker-compose itself, but provides modularity.

# Working with HyperLedger Step 4 – Writing the Chaincode



What is a Chaincode?
ChainCode
Chaincode Development
Chaincode Lifecycle

» A program, written in Go, that implements a prescribed interface. Other languages e.g. Java would be supported in future.

» They run in secured docker container, isolated from the endorsing peer processes handling business logic agreed by peer processes.

» They initialize and manage the ledger state through transactions submitted by applications. State is internal to the Chaincode and is accessible from outside through Chaincode apis.

---

» Every Chaincode must implement the Chaincode interface

» It contains the **init** method called when the Chaincode is instantiated or upgraded and an **invoke** method called in response to invoke transactions.

» Init method is use to initialize data, or to move data in case of an upgrade.

» Another interface is ChaincodeStubInterface, which is used to access and modify the ledger.

---

» Package – Contains the ChaincodeDeploymentSpec(CDS), optionally the endorsement policies(rules to check if the transaction is properly endorsed), set of signatures of entities who own the Chaincode, allowing for tamper detection as well.

» Install – The install transaction converts the Chaincode source code into a CDS and installs it on a peer node. You need to do this on each node.

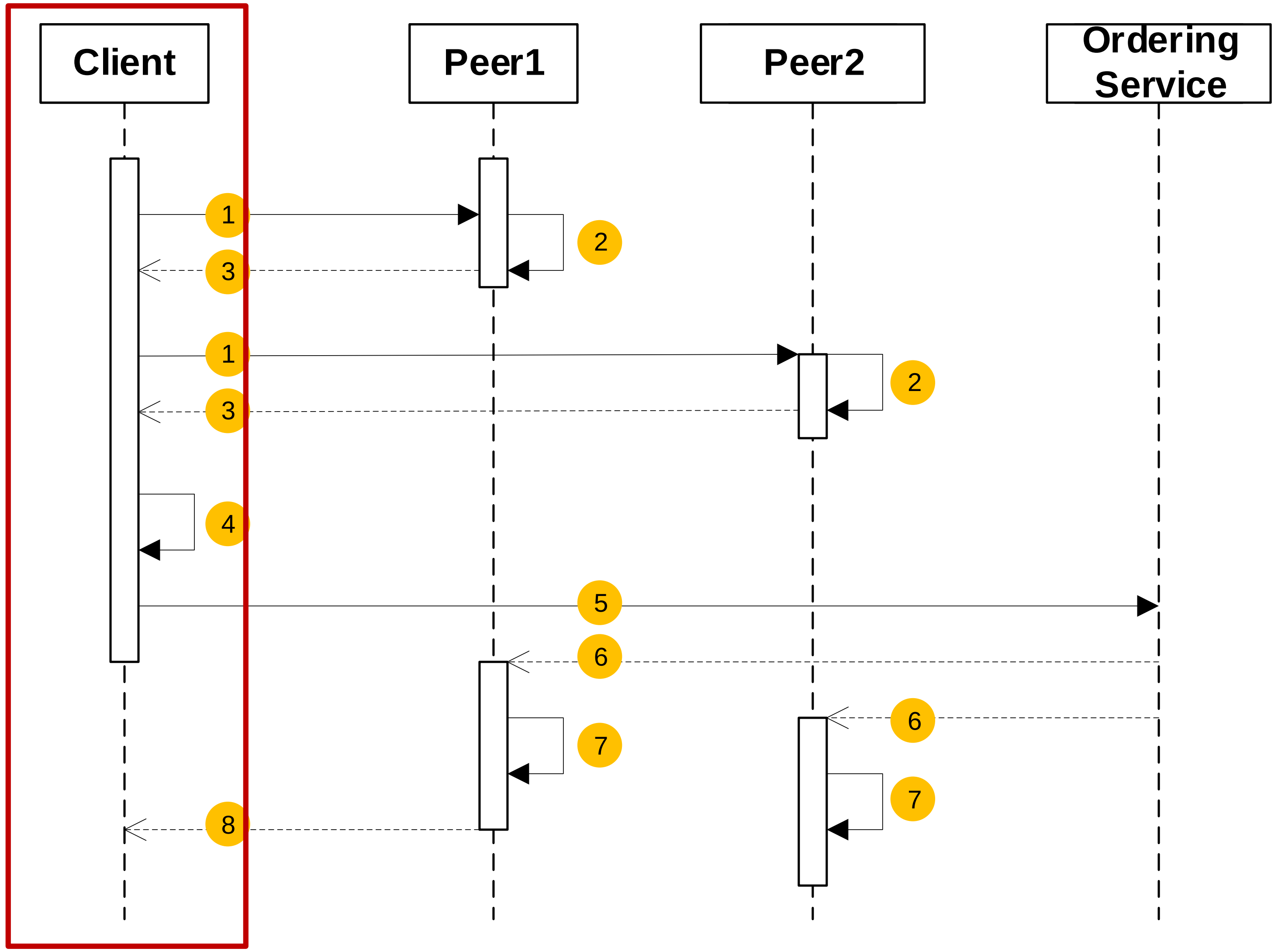
» Instantiate – Instantiates the Chaincode on a particular channel based on the instantiation policy of the Chaincode as well as the write privileges on that channel. It calls the init method.

» Upgrade – Chain code version can be changed, however the name of the Chaincode must be the same. New Version of the Chaincode must be installed before calling upgrade. It also calls the init method

# Working with HyperLedger Step 5 – Call Chaincode with Node.JS SDK

**Node.JS SDK supports**

- Creation of Channels
- Ask Peer to Join a Channel
- Install Chaincode on peers
- Instantiate Chaincode on channel
- Invoke Chaincode transactions
- Query the Ledger for transaction blocks



Cognizant

KEEP CHALLENGING™

## Thank You !

**Blockchain Hub**

BlockchainHub@cognizant.com