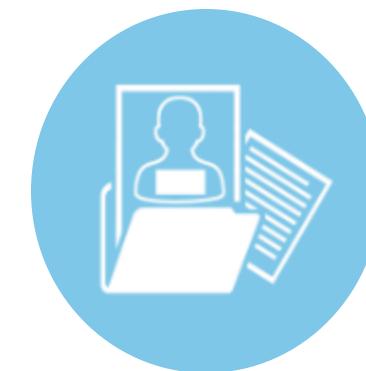


# Blockchain Certification Training

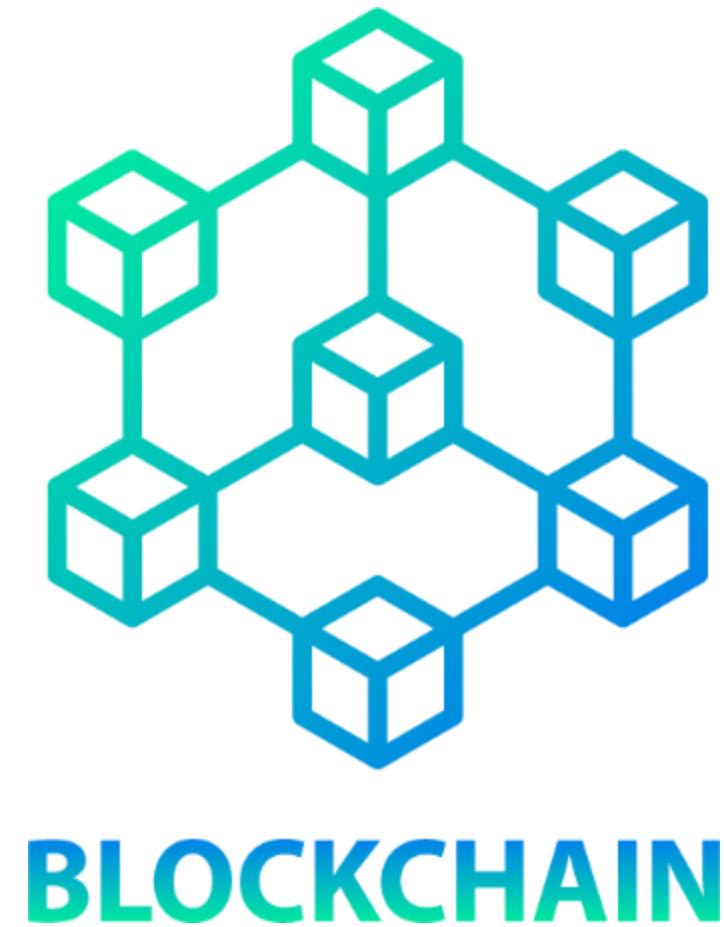
## Lesson 0: Course Introduction



# Course Objectives

By the end of this course, you will be able to acquire the following skills:

- Describe Blockchain technology and its key concepts
- Explain Bitcoin Transaction Process
- Work with Ethereum deployment tools
- Deploy a private Ethereum Blockchain
- Work with Hyperledger projects
- Deploy a business network using Hyperledger Composer
- Develop and deploy smart contracts on Ethereum test network
- Develop a private Blockchain using Multichain
- Describe the future prospects and real world use cases of Blockchain



# Course Outline

Overview of Blockchain

1

Bitcoin Blockchain

2

Ethereum

3

Deploying Smart Contracts  
on Ethereum Network

4

Hyperledger

5

Hyperledger Composer

6

Blockchain on Multichain

7

Blockchain Prospects

8



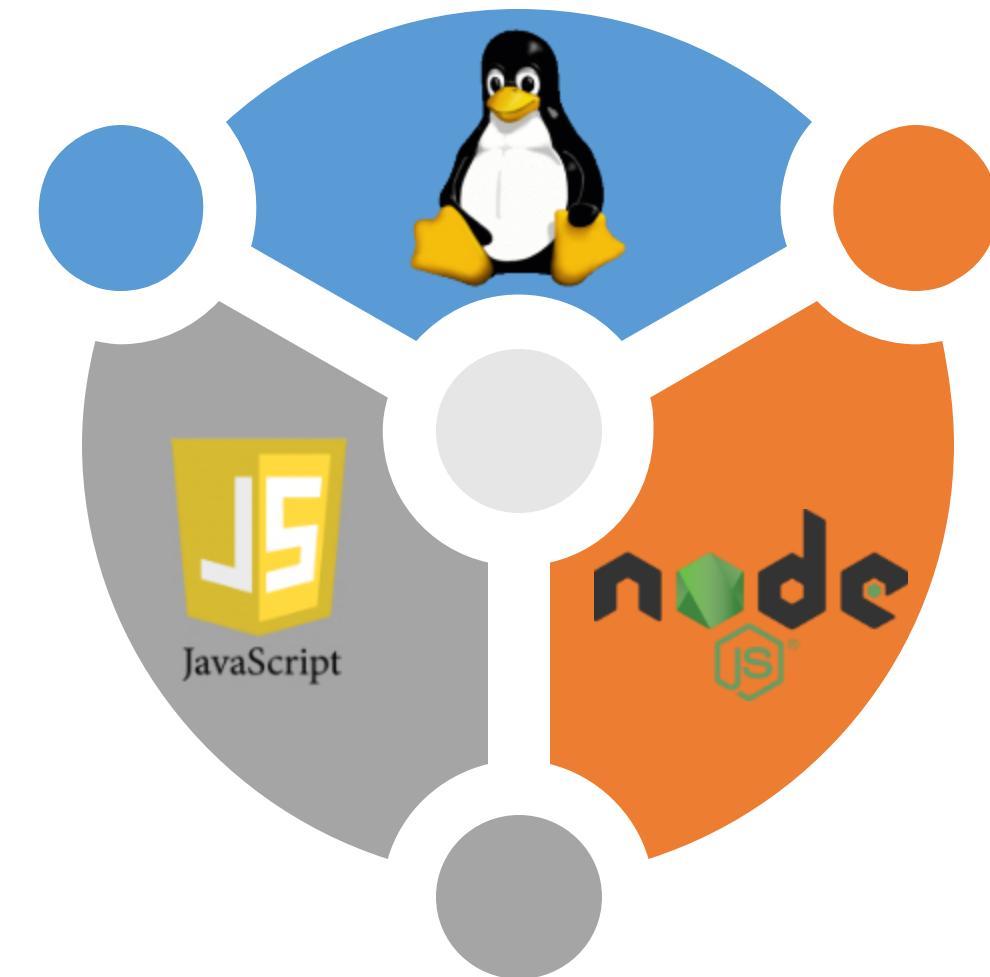
# Prerequisites for This Course

There are no prerequisites for this course. However, prior knowledge of the following technologies would be helpful:

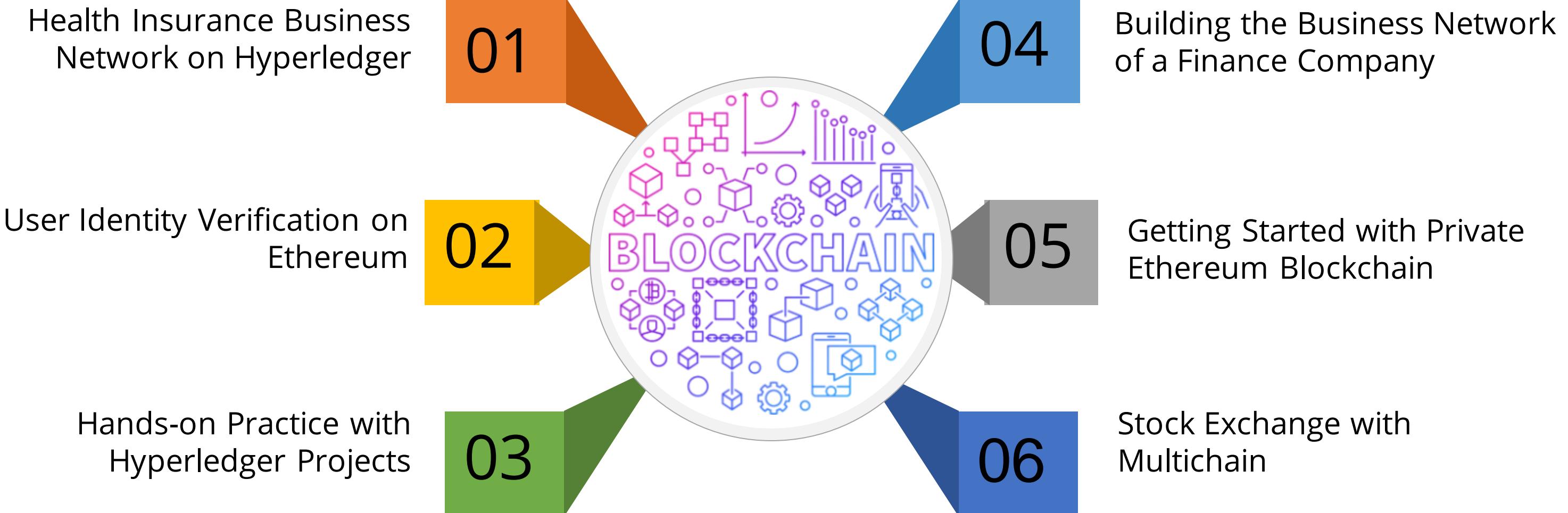
**Linux Fundamentals**

**Node.js Fundamentals**

**JavaScript**



# Highlights of This Course





**Thank You**

# Blockchain

## Lesson 1: Overview of Blockchain



# Learning Objectives

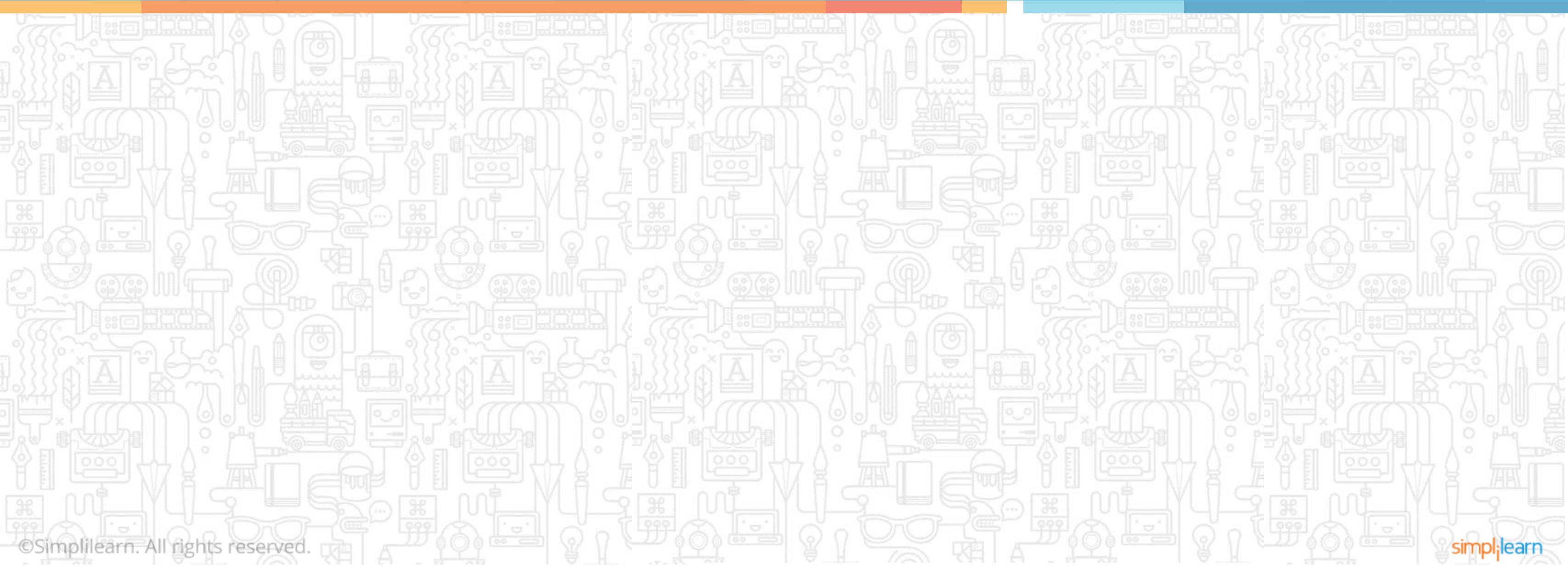


By the end of this lesson, you will be able to:

- ➊ Describe the Blockchain transaction process
- ➋ Generate a public key and a digital signature
- ➌ Generate a nonce, a hash code, and a Blockchain block
- ➍ Work with a distributed system and perform Blockchain transaction

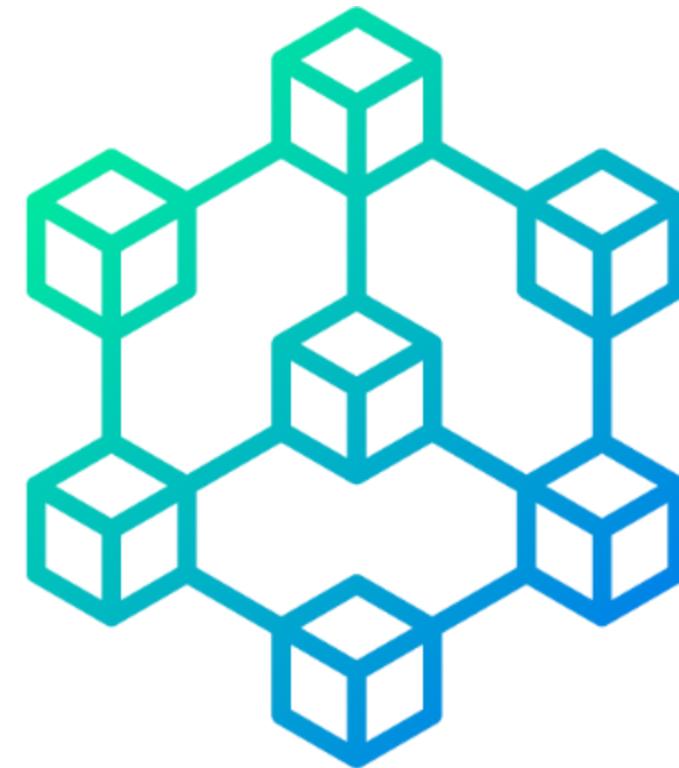
# Overview of Blockchain

## Blockchain and Its Importance



# Blockchain

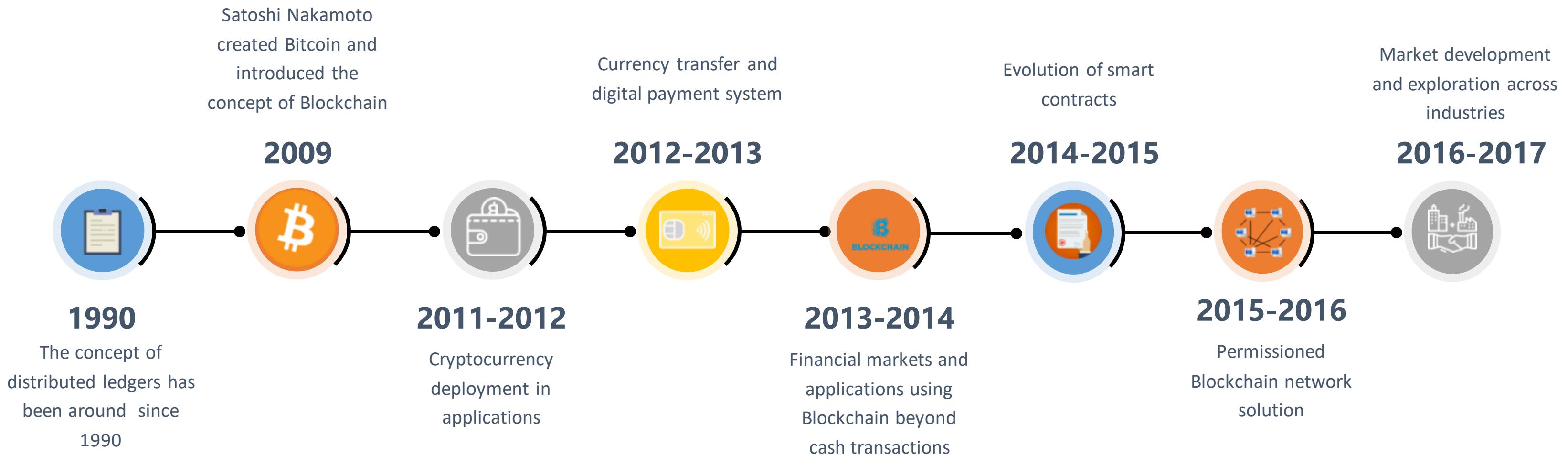
---



## BLOCKCHAIN

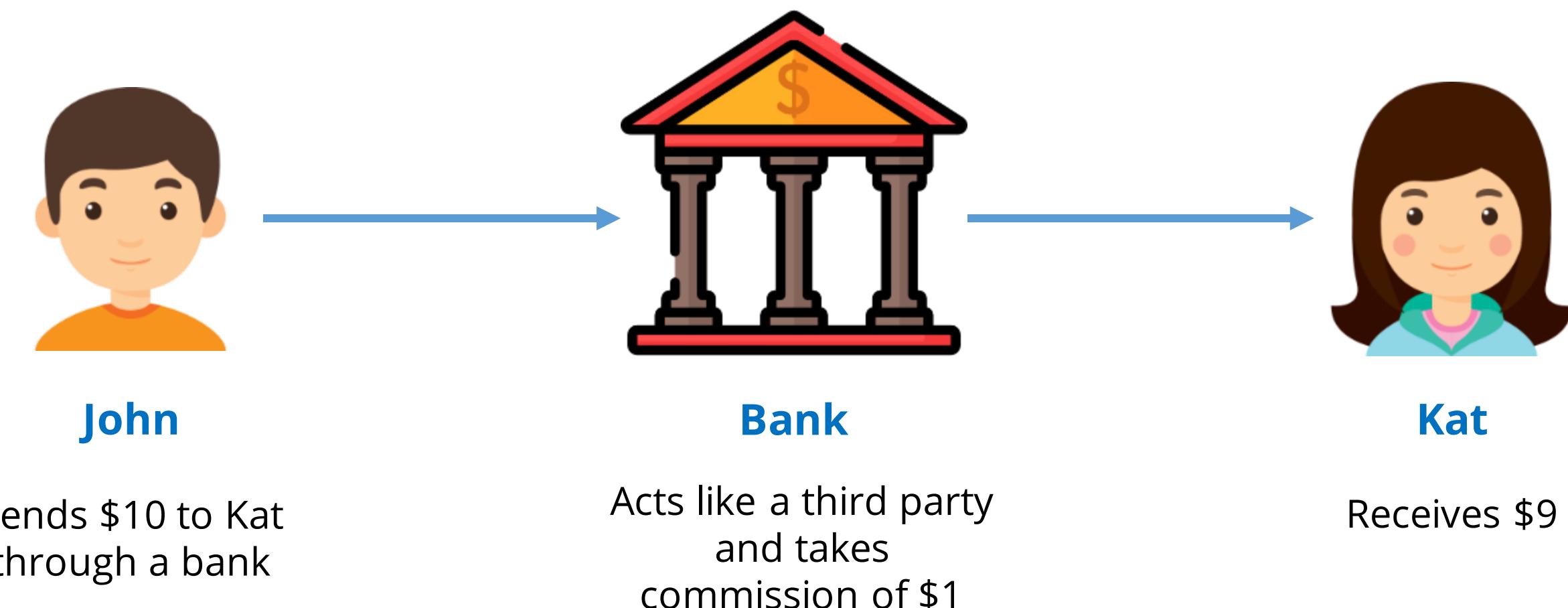
- Blockchain is a decentralized ledger of all transactions across peer-to-peer network.
- It is a technology that enables Bitcoin and is also applied to many business processes.
- It not only performs transactions but also ensures anonymity and security of the users.

# History of Blockchain



# Current Banking System

---



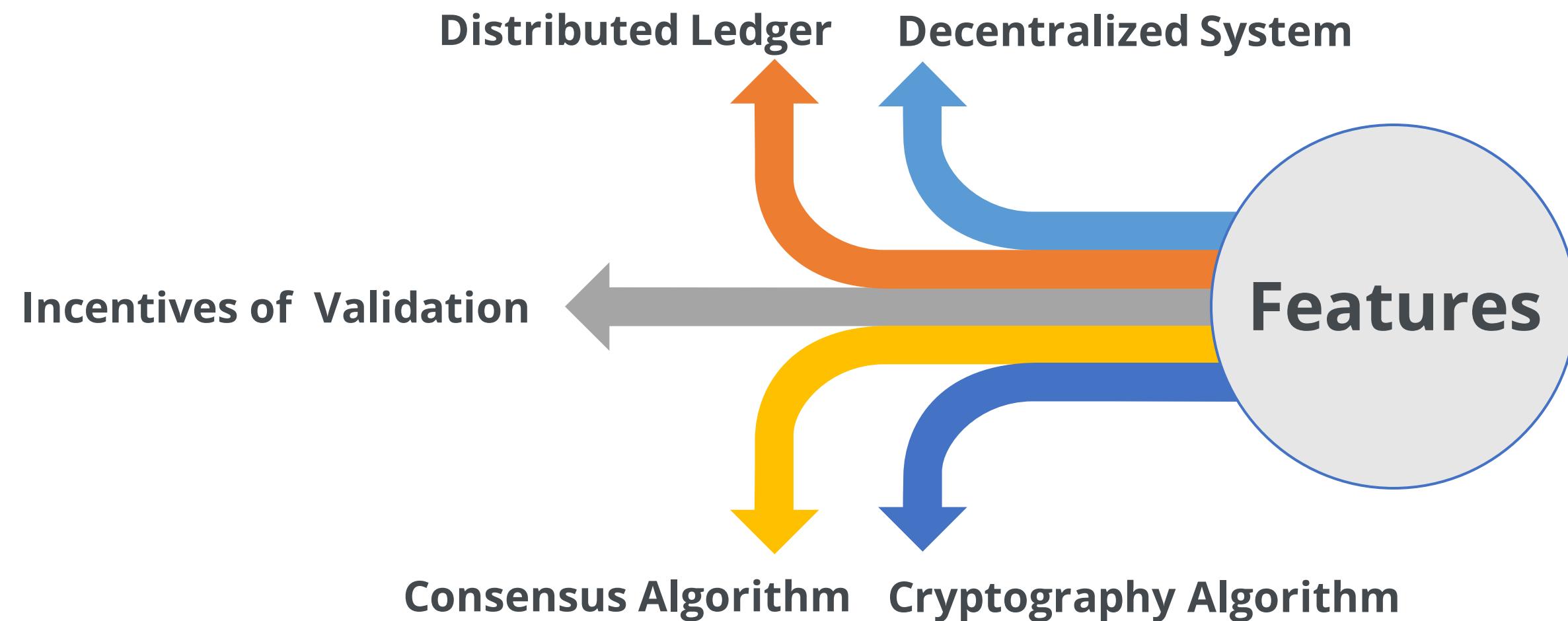
# Issues in Banking System

There were few issues in the previous banking system that lead to the rise of Blockchain technology.



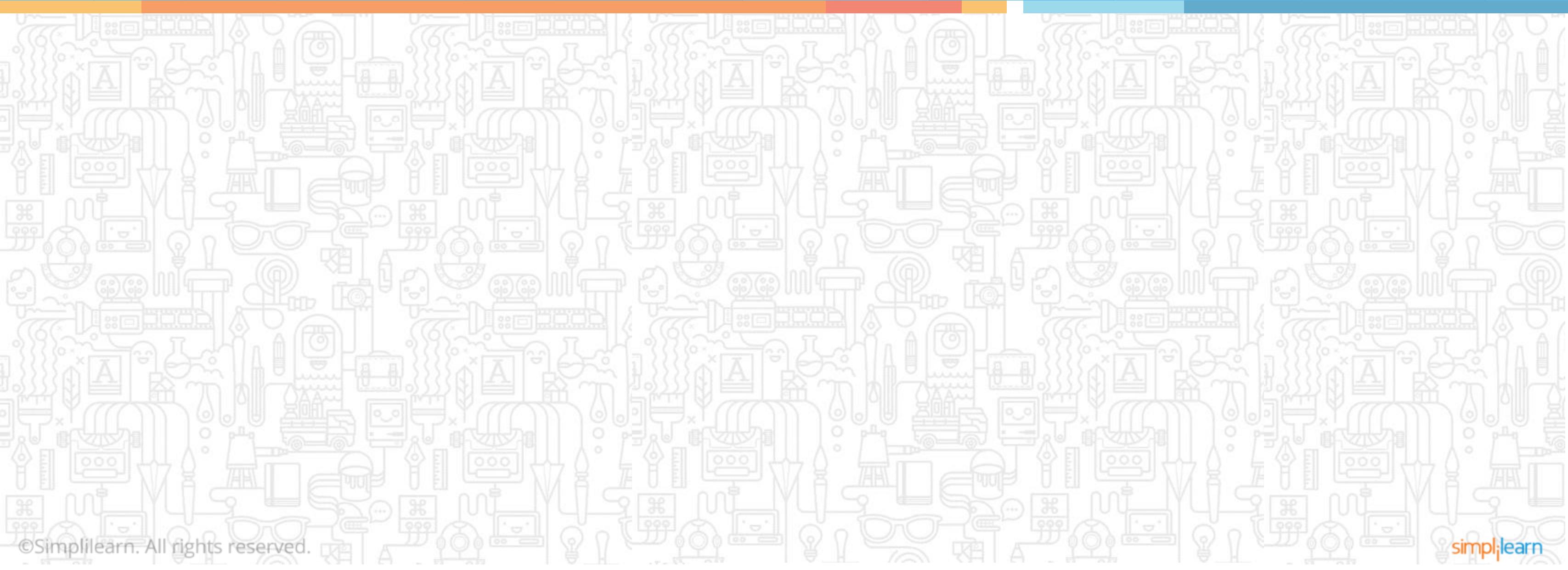
# Blockchain Solution to the Issues

Blockchain tackled the issues in the previous system with some of its features mentioned below:

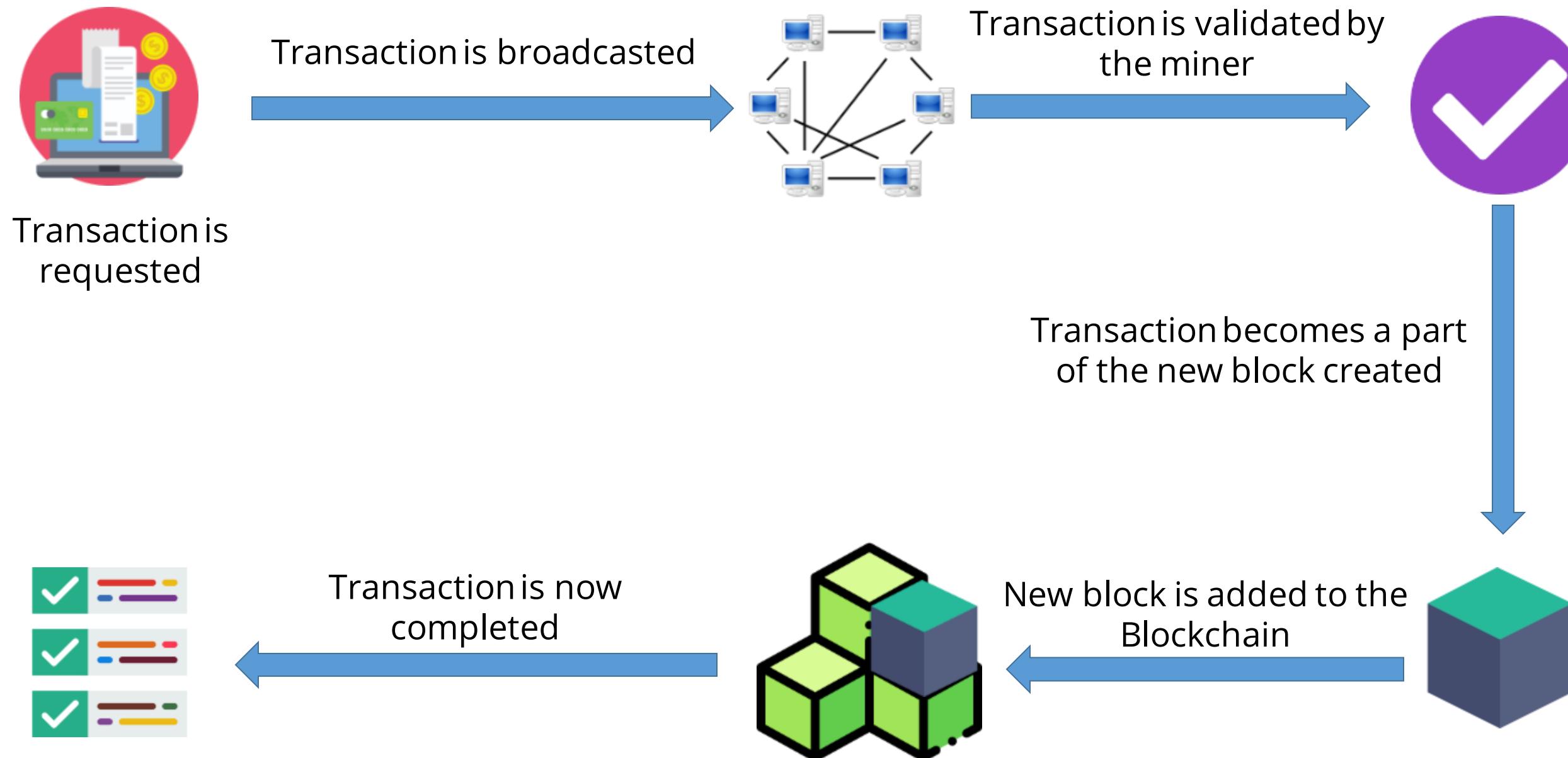


# Overview of Blockchain

## Blockchain Transaction Process



# Blockchain Transaction Process



# Steps of Blockchain Transaction



Blockchain transaction works implementing one of the following features in each step:

Cryptography  
Algorithm

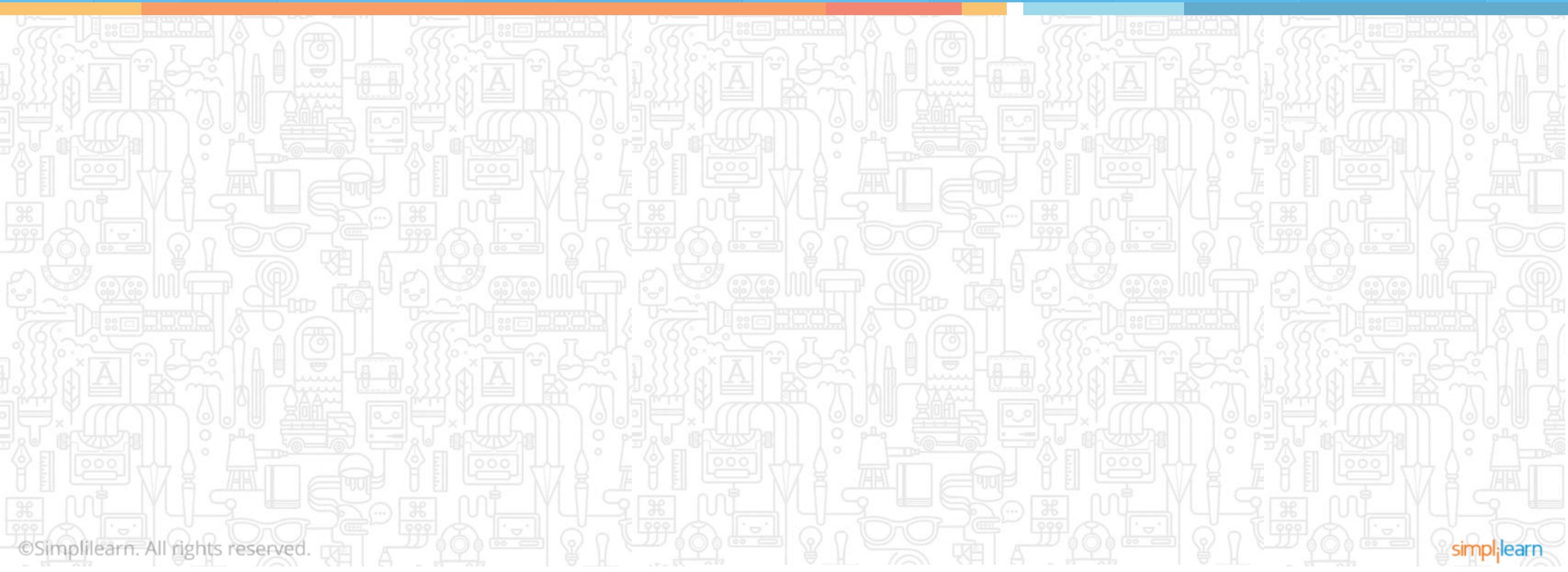
Decentralized  
Network

Consensus  
Algorithm

Distributed  
Ledger

# Overview of Blockchain

## Transaction Initiation



# Features of Blockchain



**Cryptography  
Algorithm**

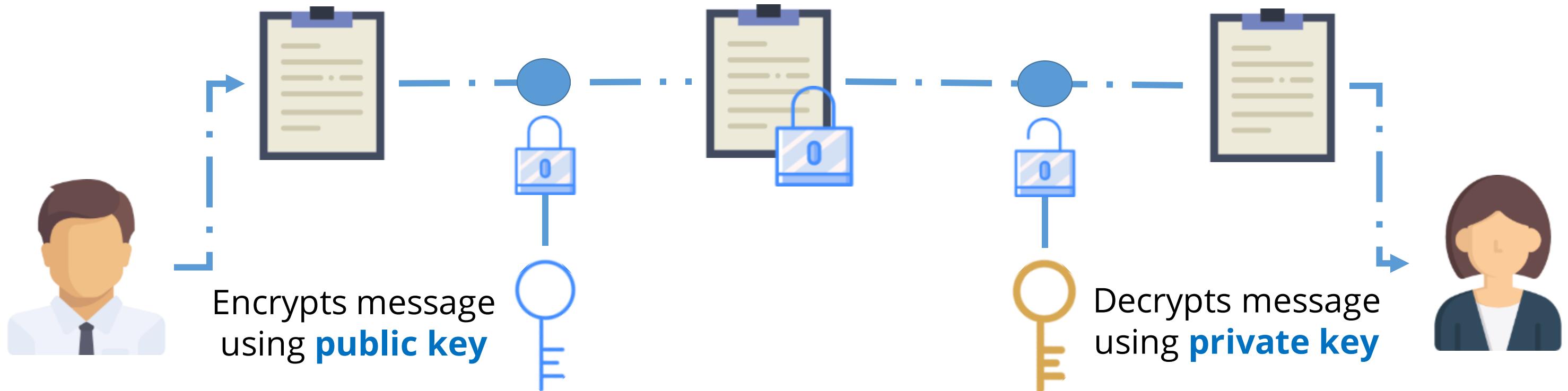
Decentralized  
Network

Consensus  
Algorithm

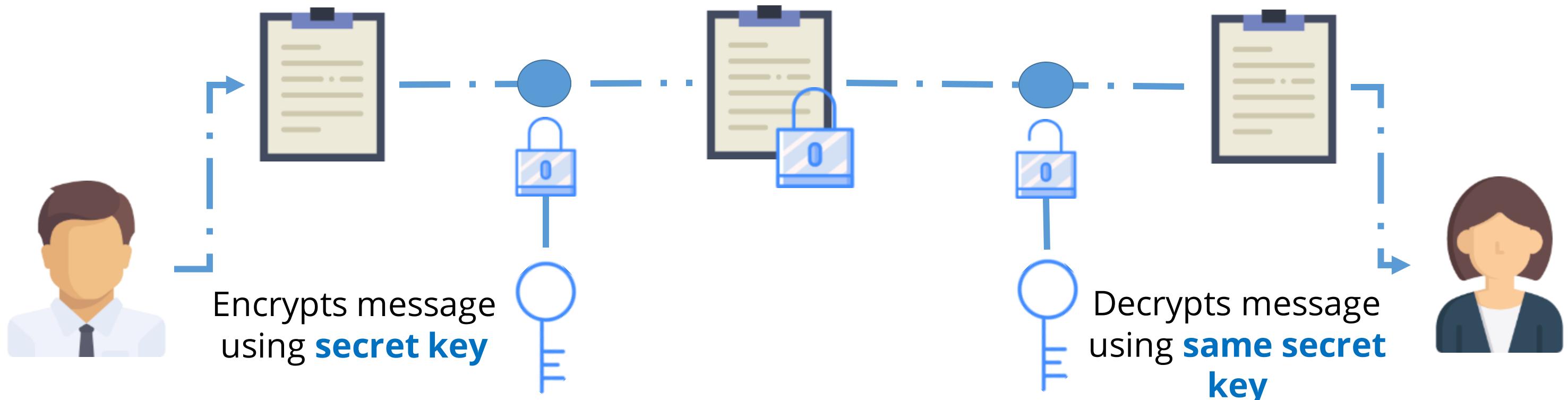
Distributed  
Ledger

It ensures authentication and integrity of the transaction.

# Asymmetric Key Cryptography



# Symmetric Key Cryptography



# Digital Signature



## Digital Signature

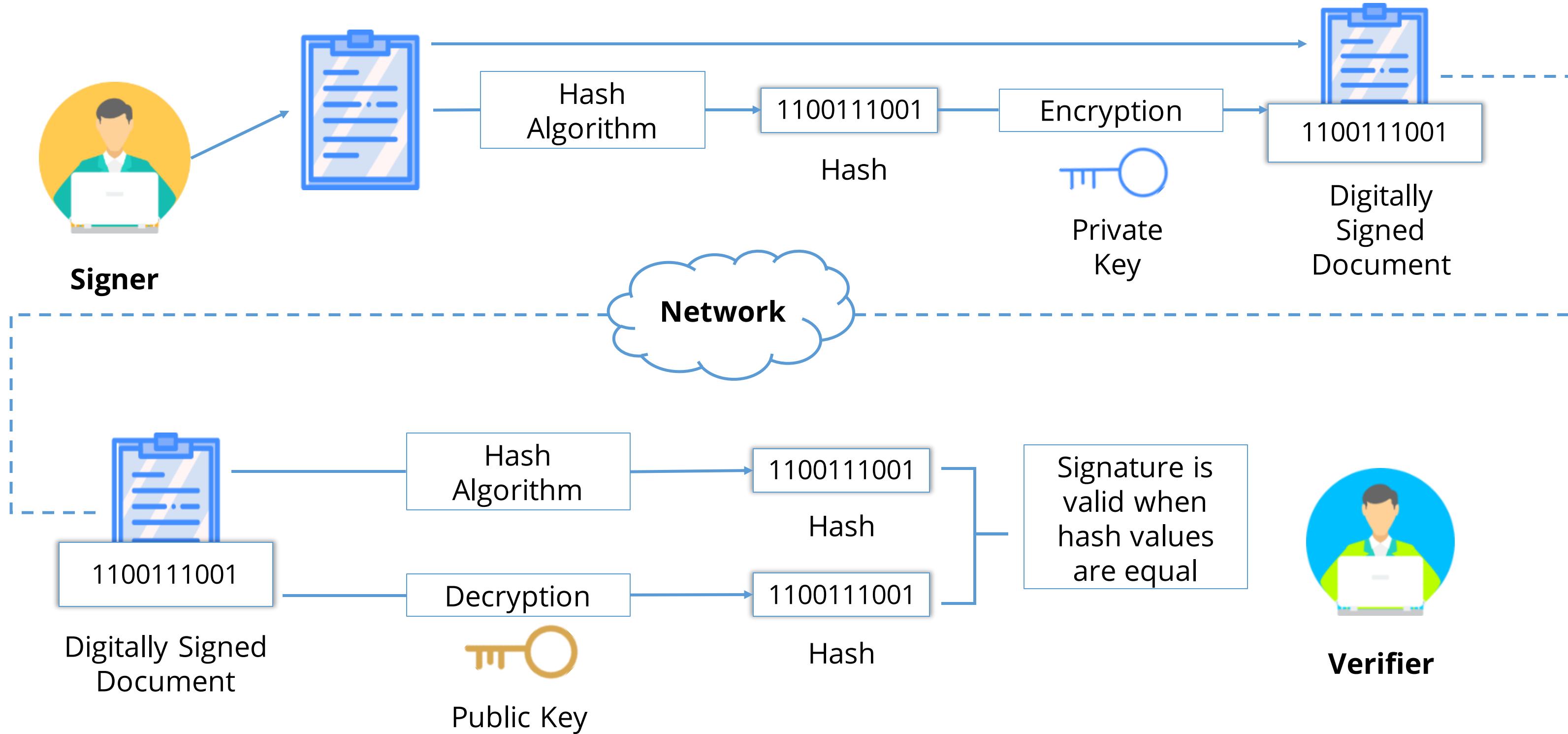
A digital signature provides authentication and validation as normal signatures do.

It ensures the security and integrity of data recorded on the Blockchain.

It uses asymmetric cryptography in which information can be shared using a public key.

Primary keys are linked to users providing digital signatures a quality of nonrepudiation.

# Digital Signature Creation



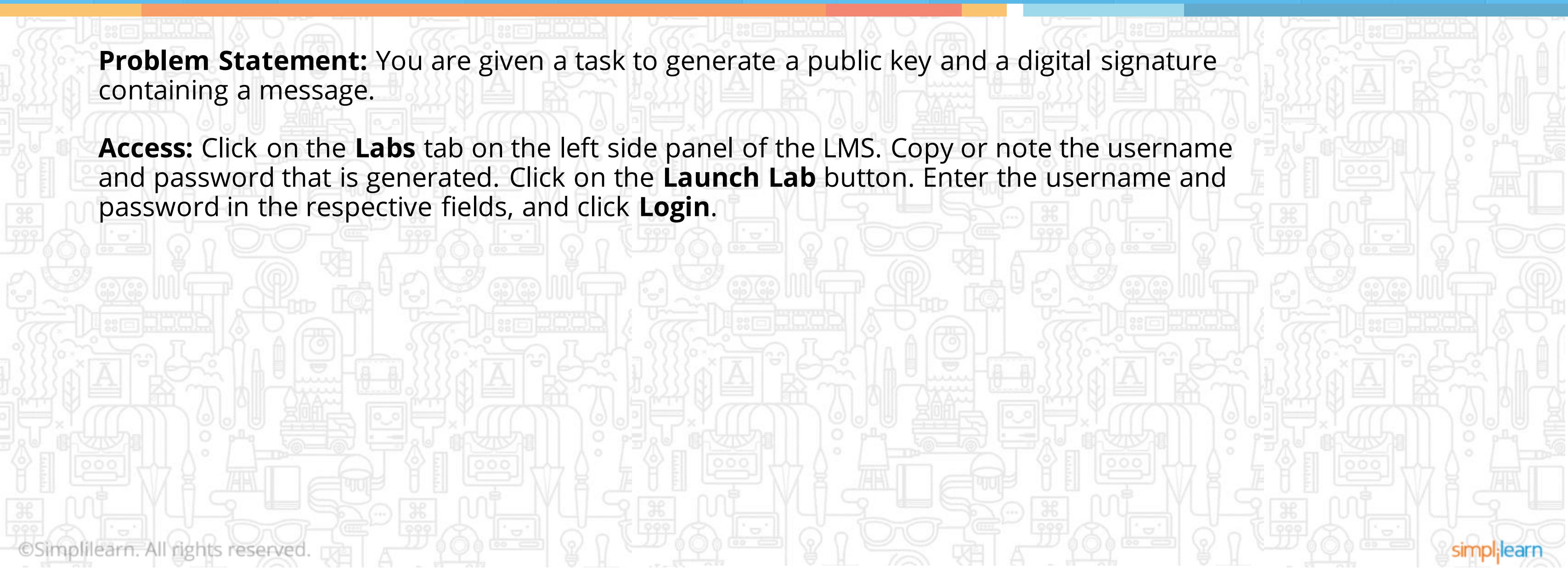
## Assisted Practice

Duration: 5 mins

### Generation of a Public/Private Key Pair and a Digital Signature

**Problem Statement:** You are given a task to generate a public key and a digital signature containing a message.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. Enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/public-private-keys/keys.html>

### Step 02

Click on the random button to generate public and private key

### Step 03

Visit <https://anders.com/blockchain/public-private-keys/signatures.html>

### Step 04

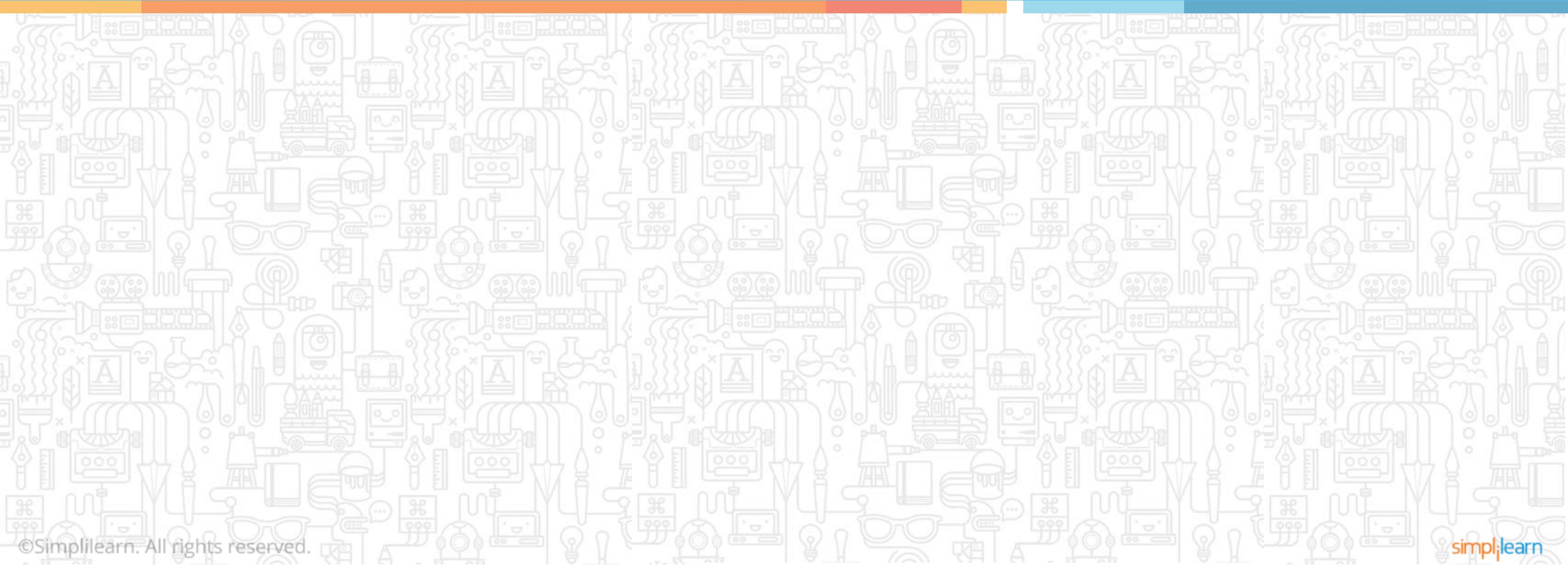
Enter a message, and click on sign button to generate a digital signature

### Step 05

Click on verify tab to verify the digital signature created

# Overview of Blockchain

## Transaction Broadcast



# Features of Blockchain



Cryptography  
Algorithm

**Decentralized  
Network**

Consensus  
Algorithm

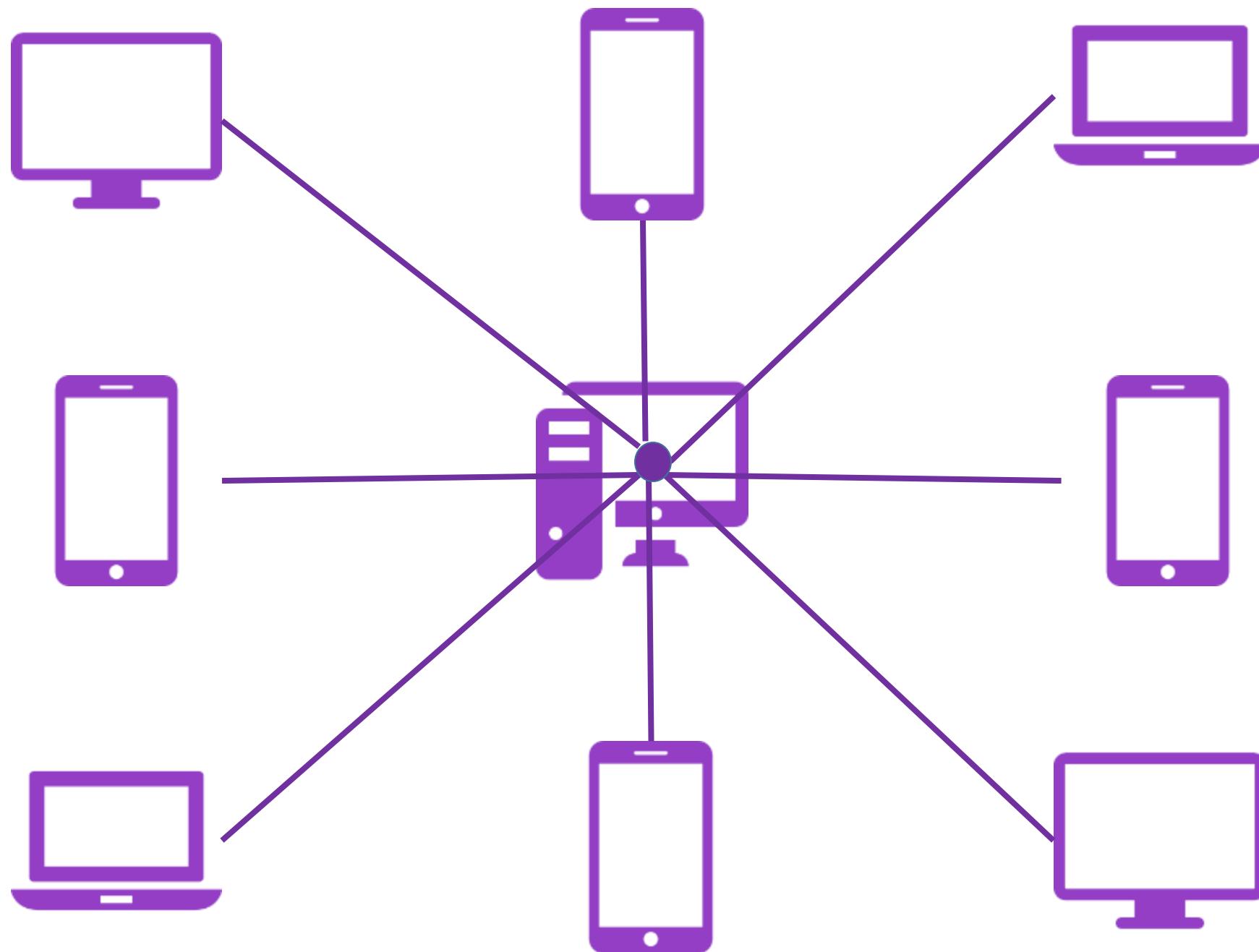
Distributed  
Ledger

It is a system where miners play an important role in the validation of transaction taking place.

# Types of Network

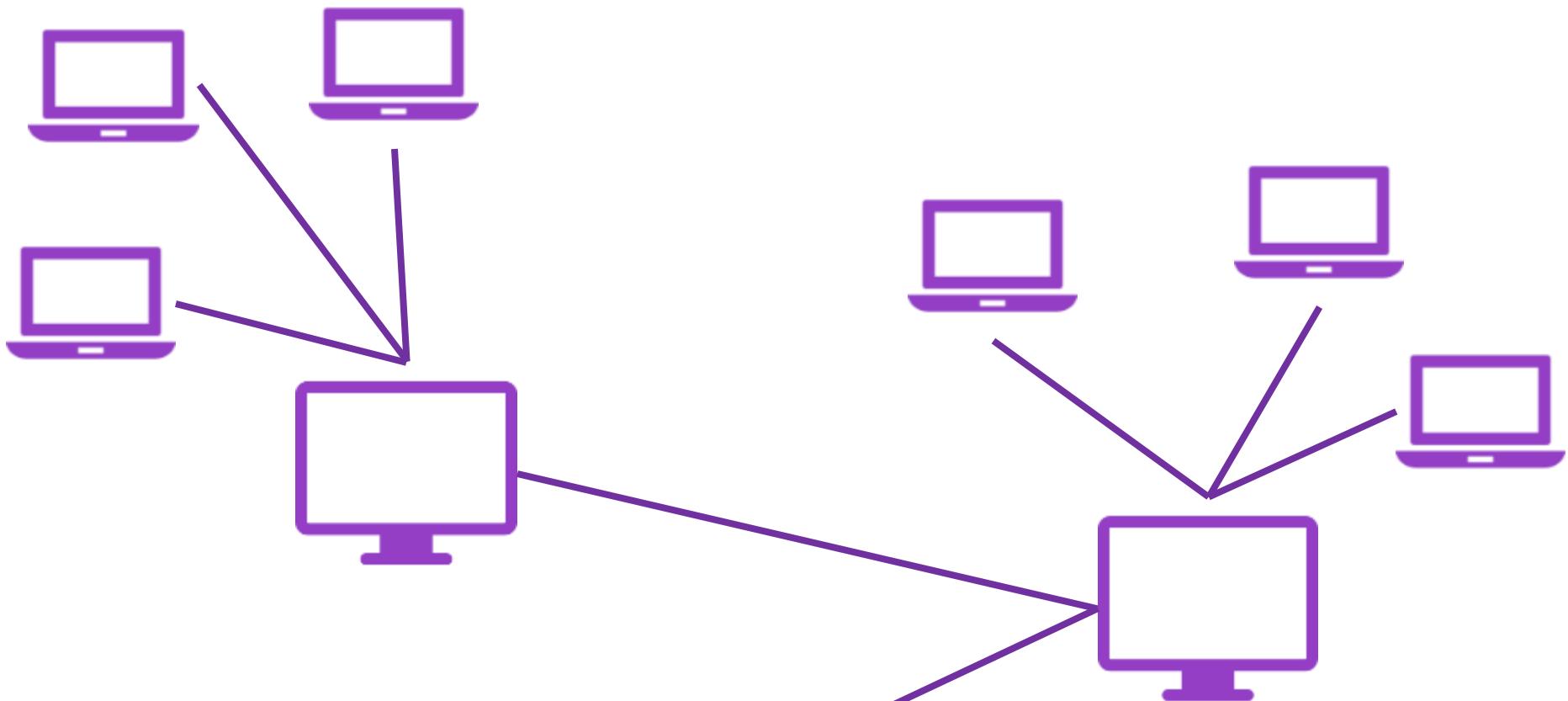
Centralized Network

Decentralized Network

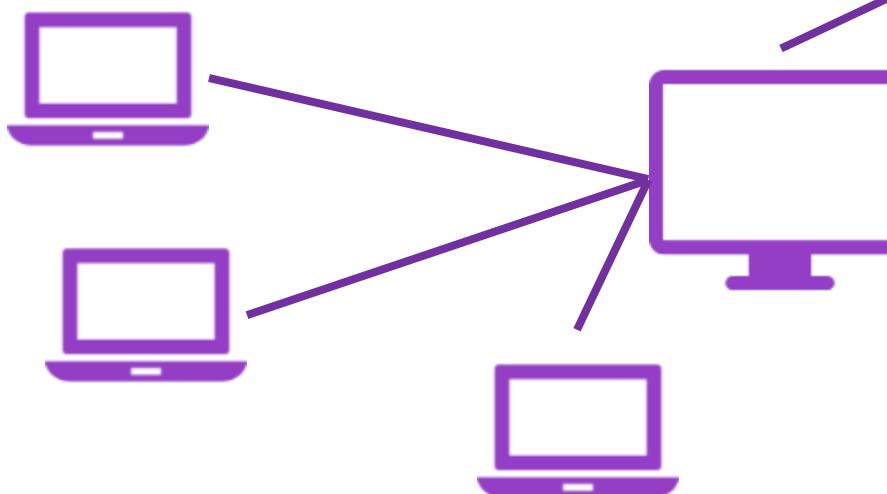


# Types of Network

Centralized Network

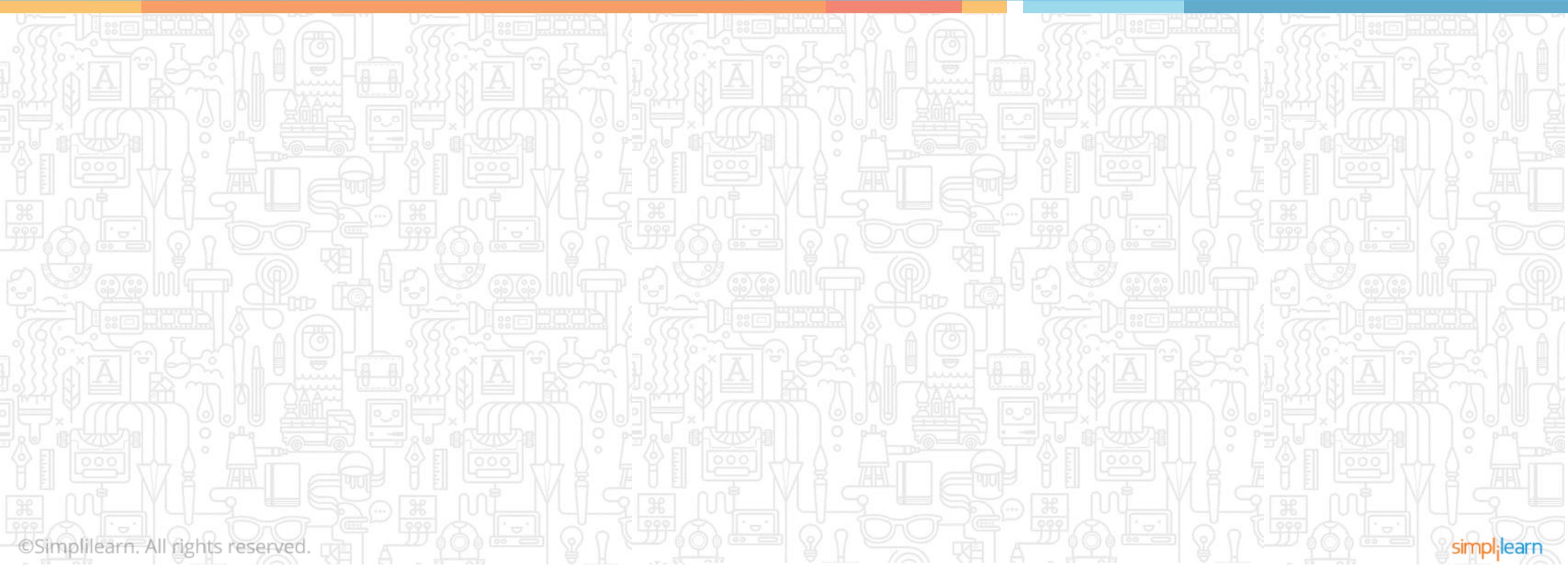


Decentralized Network



# Overview of Blockchain

## Transaction Validation



# Features of Blockchain



Cryptography  
Algorithm

Decentralized  
Network

**Consensus  
Algorithm**

Distributed  
Ledger

Miners will validate the transaction using mathematical puzzles.

# Consensus Protocol

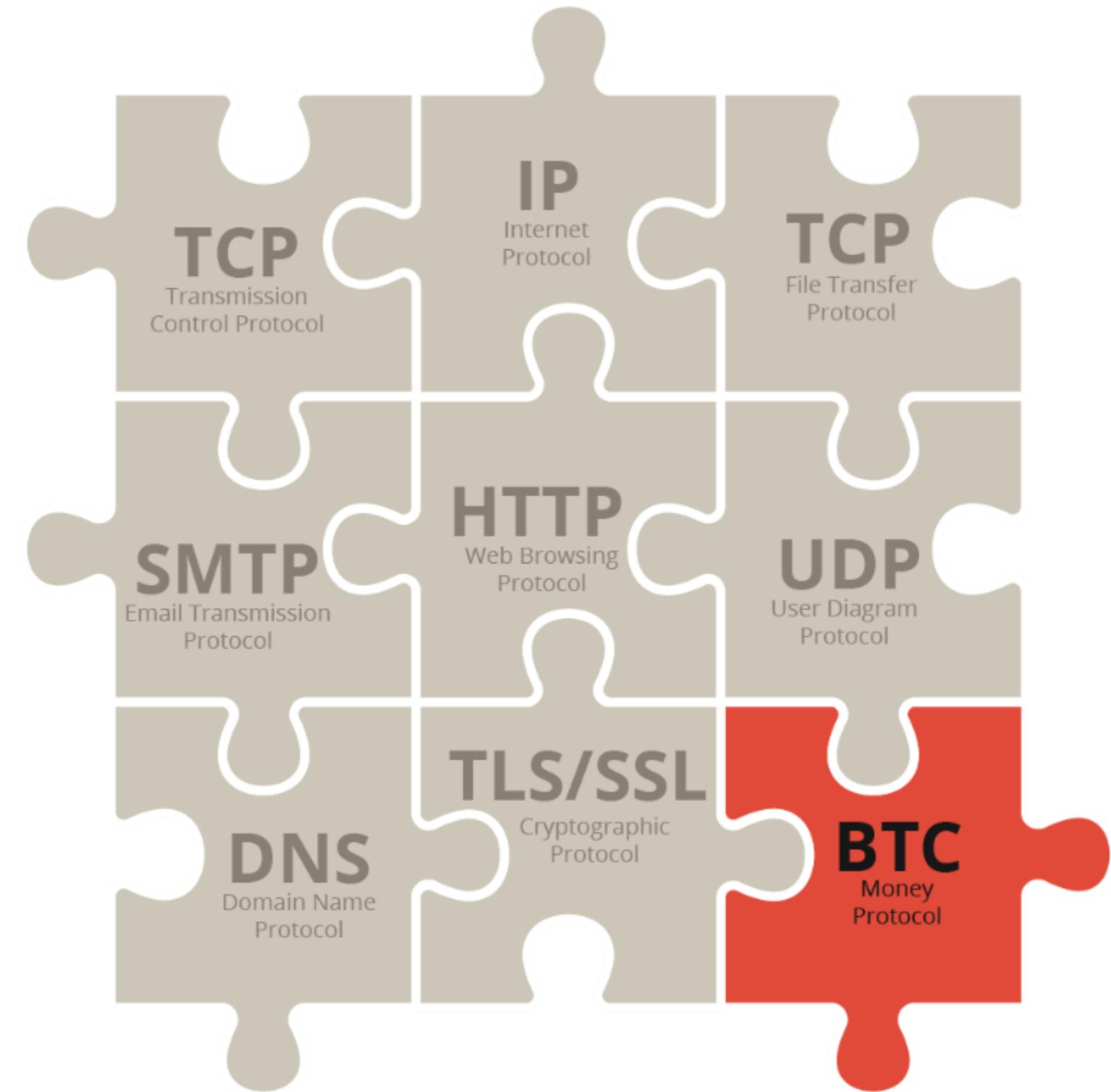


It is a fault-tolerant protocol that is used to achieve the necessary agreement on a single data value or a single state of network.

It is a set of rules that decides on the contribution of the various participants of the Blockchain.

It ensures that all transactions occurring on the network are genuine and all participants agree on the consensus of the ledger.

# Blockchain Protocols



# Features of Consensus Protocol

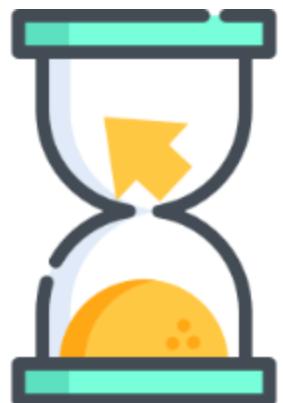
---



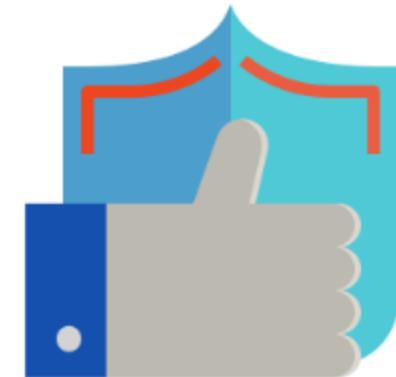
**Efficient**



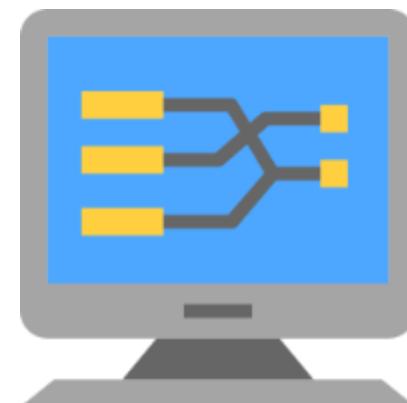
**Secure**



**Real-time**



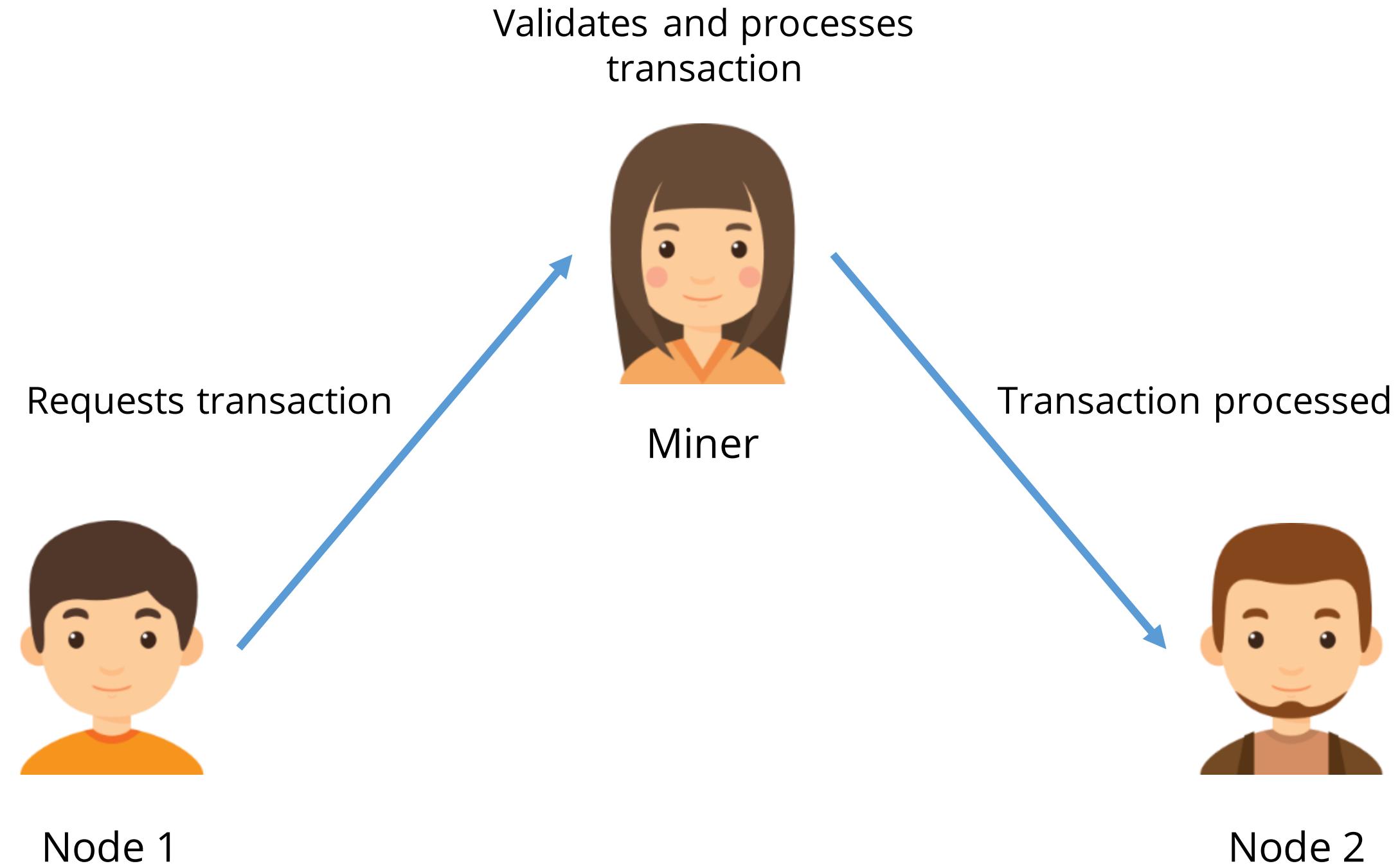
**Reliable**



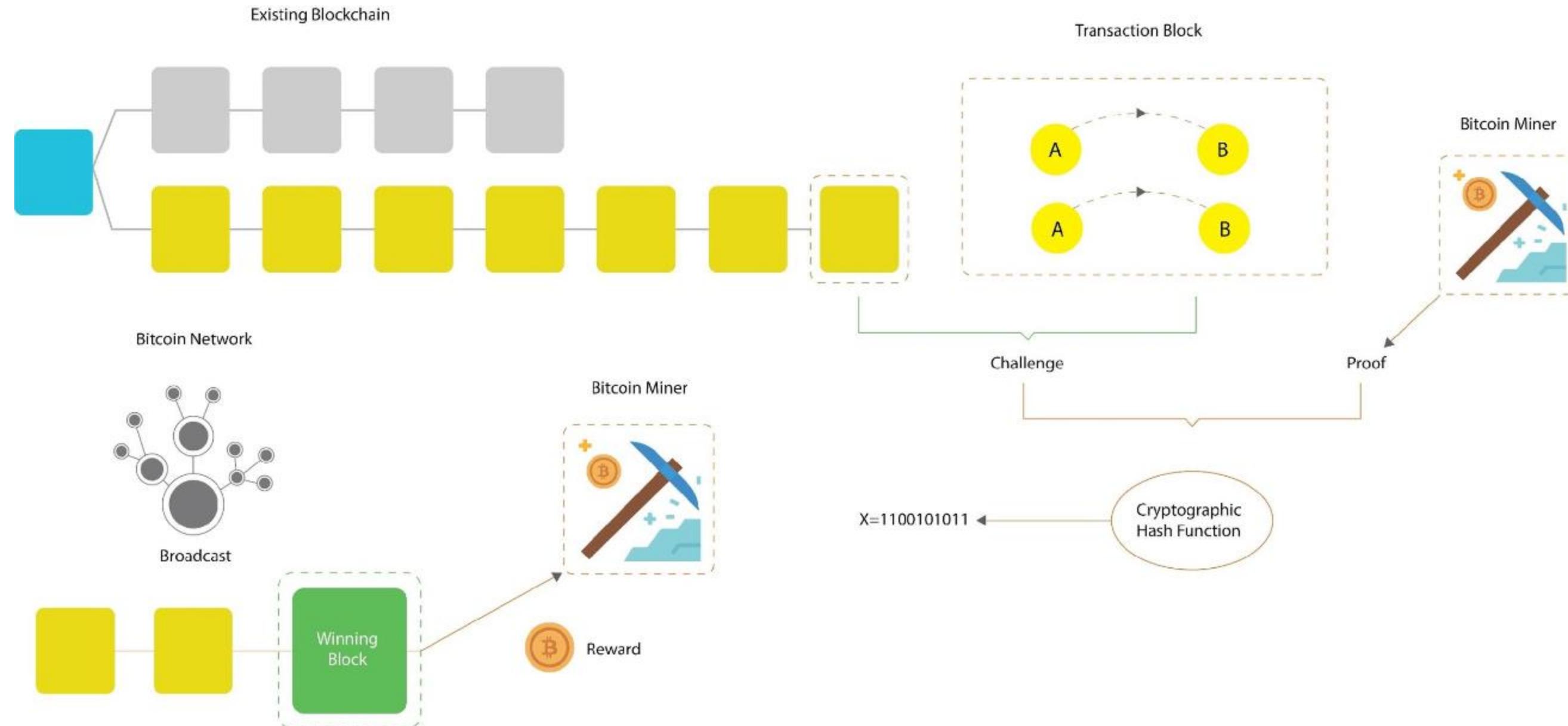
**Functional**

## Role of Miner

---

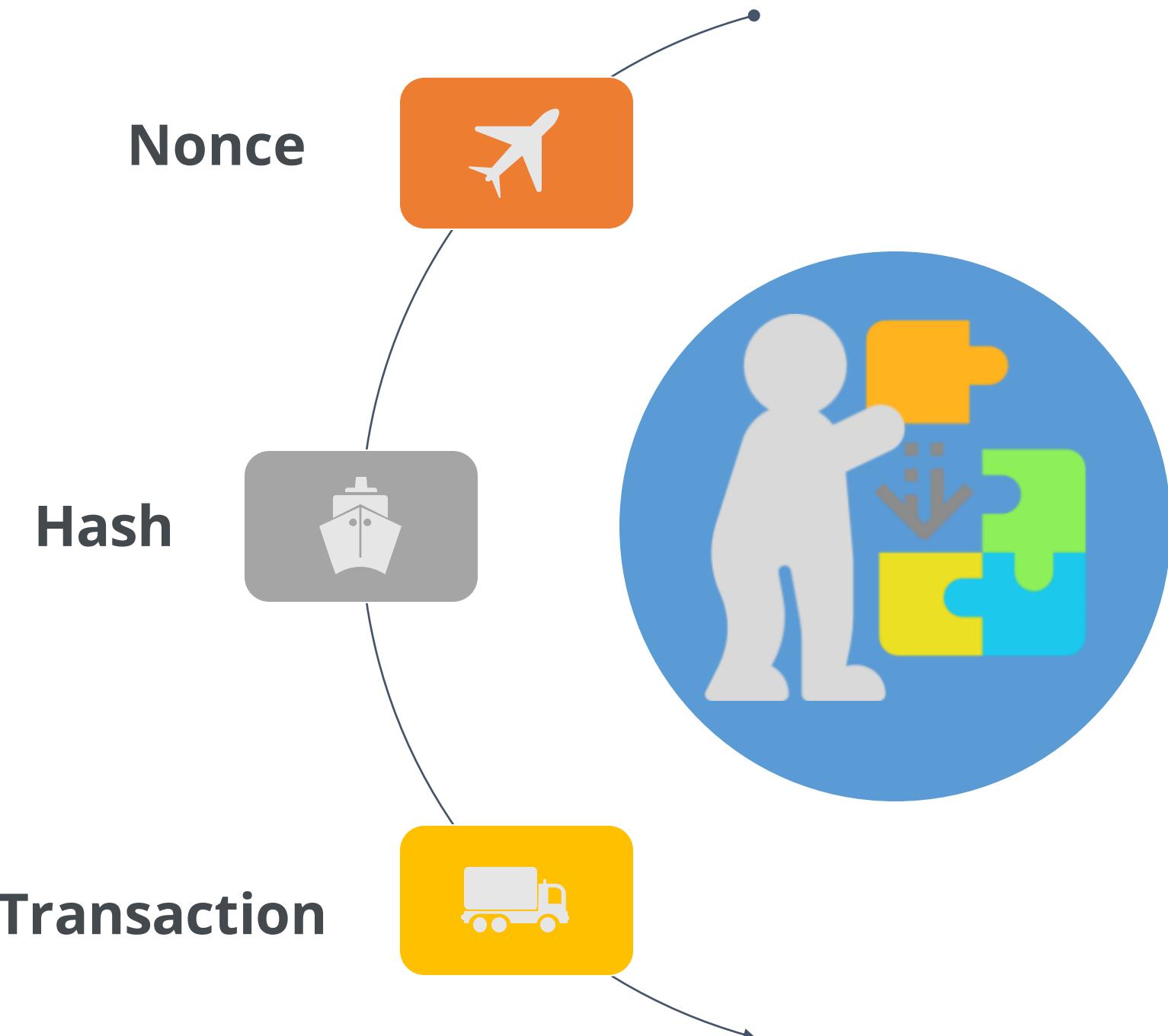


# Proof of Work



# Proof of Work

---



# Nonce



A random number whose value is set so that the hash of the block will contain a run of leading zeros.

Block #248  
Prev Block Hash:  
#65A...  
Transaction:  
Txn 673...  
Txn a63...  
Random number (guess) :  
2435681

New Block  
Prev Block Hash:  
#78E...  
Transaction:  
Txn 725...  
Txn 434...  
Random number (guess) :  
6873838

#78E...

---

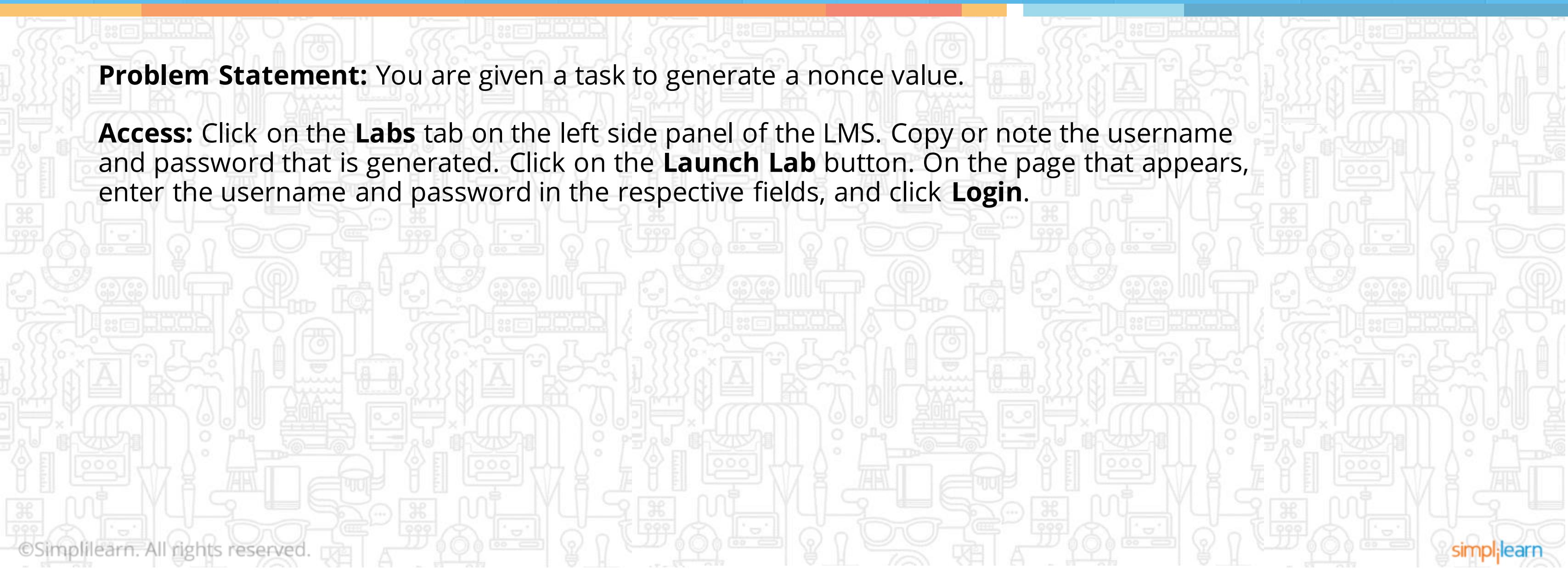
## Assisted Practice

### Generation of a Nonce Value

Duration: 5 mins

**Problem Statement:** You are given a task to generate a nonce value.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/block.html>

### Step 02

Enter some data in the data field, and click **Mine**

### Step 03

Observe the generation of new nonce value

## Hash Code

- The code generated by taking an input and converting it to cryptographic output using mathematical algorithm is hash code.
- $H(x)$  is the hash of  $x$ .



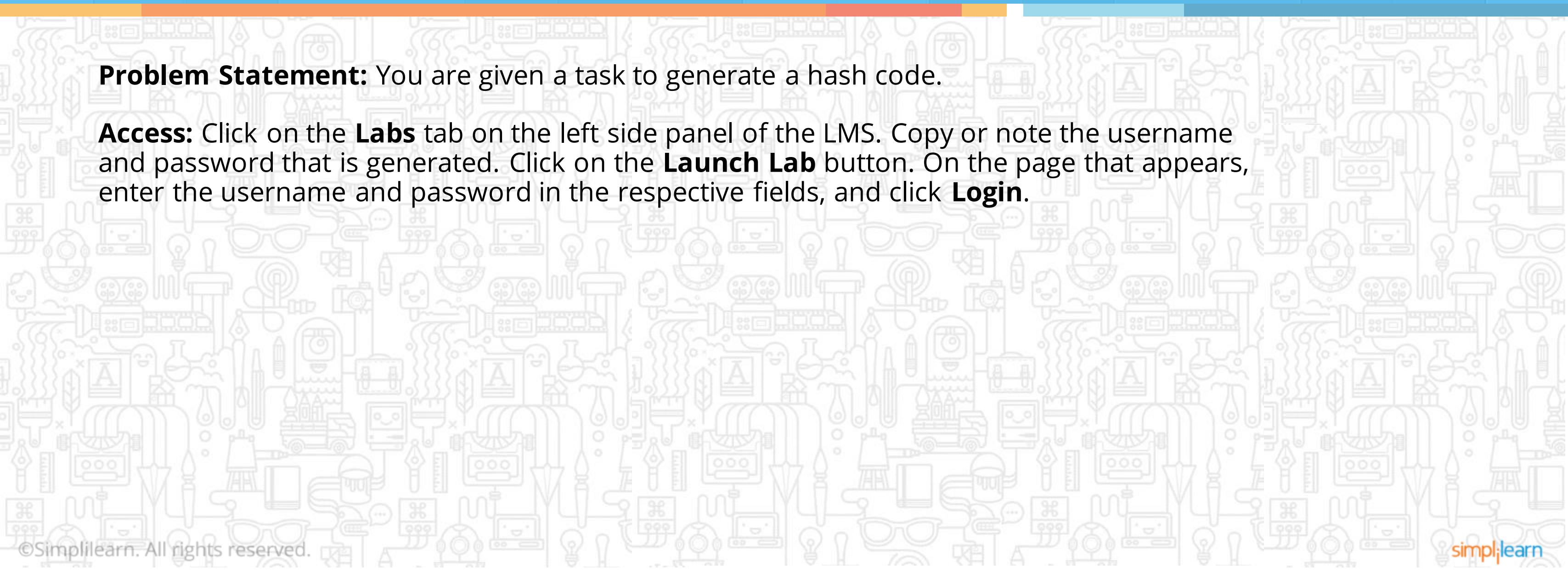
## Assisted Practice

### Generation of a Hash Code

Duration: 5 mins

**Problem Statement:** You are given a task to generate a hash code.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



**Step 01**

Visit <https://anders.com/blockchain/hash.html>

**Step 02**

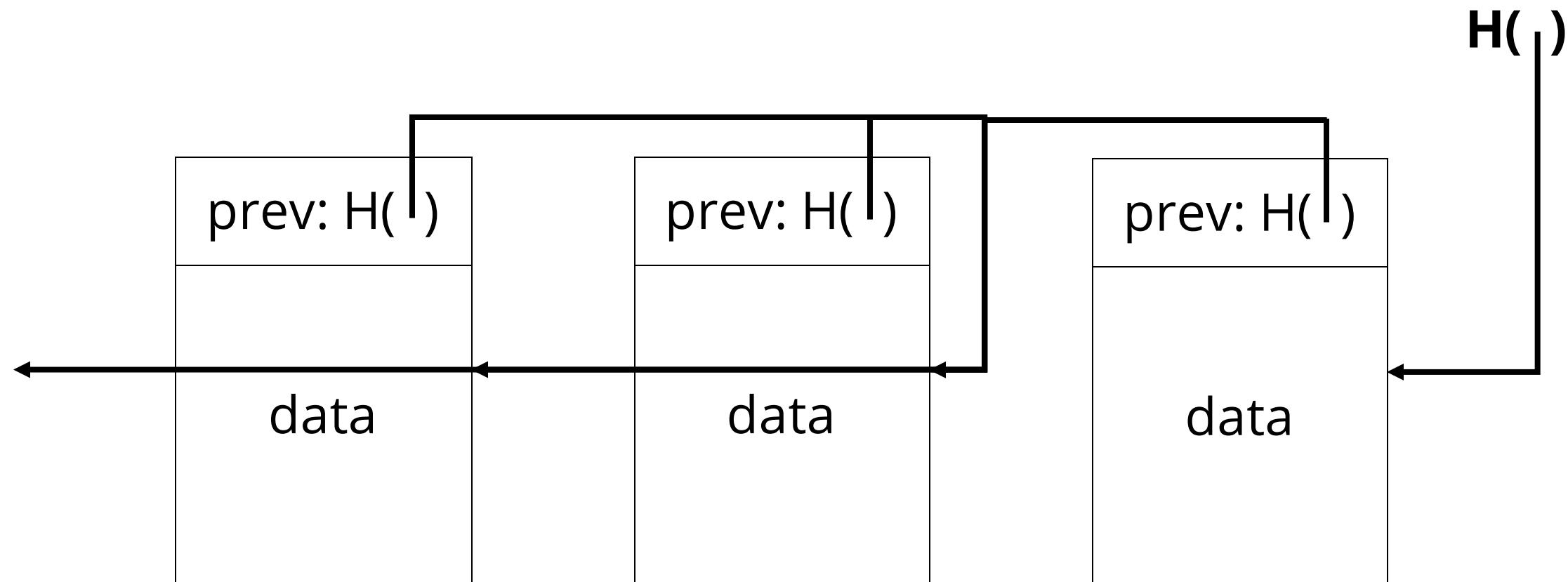
Enter some data in the data field

**Step 03**

Observe the generation of hash value for the date

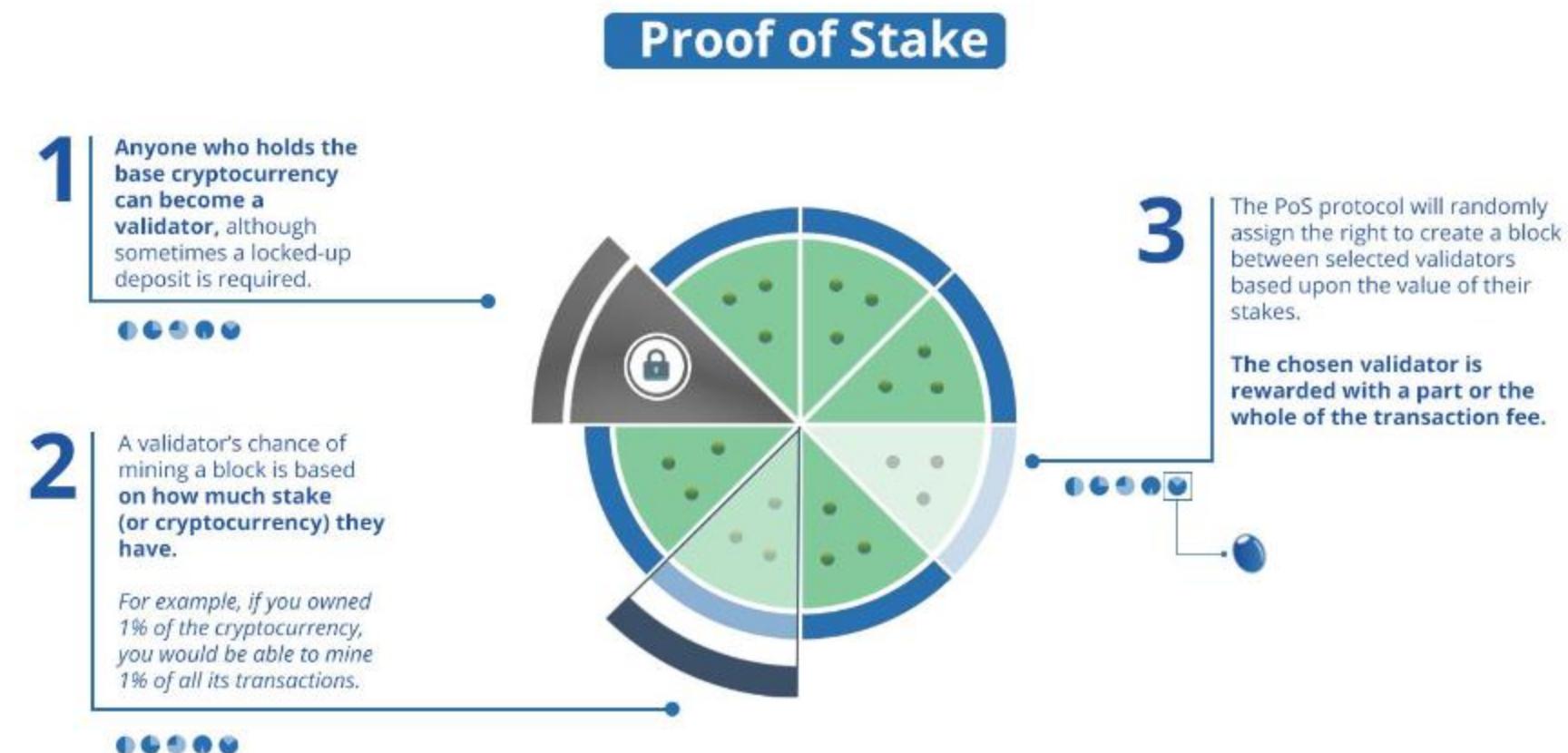
# Hash Pointer

- Hash pointer is a pointer to the location where information or hash of that information is stored
- If we retrieve information that the pointer points at, we can get hash of the information and confirm it to be unchanged
- It requires information of previous hash



# Proof of Stake

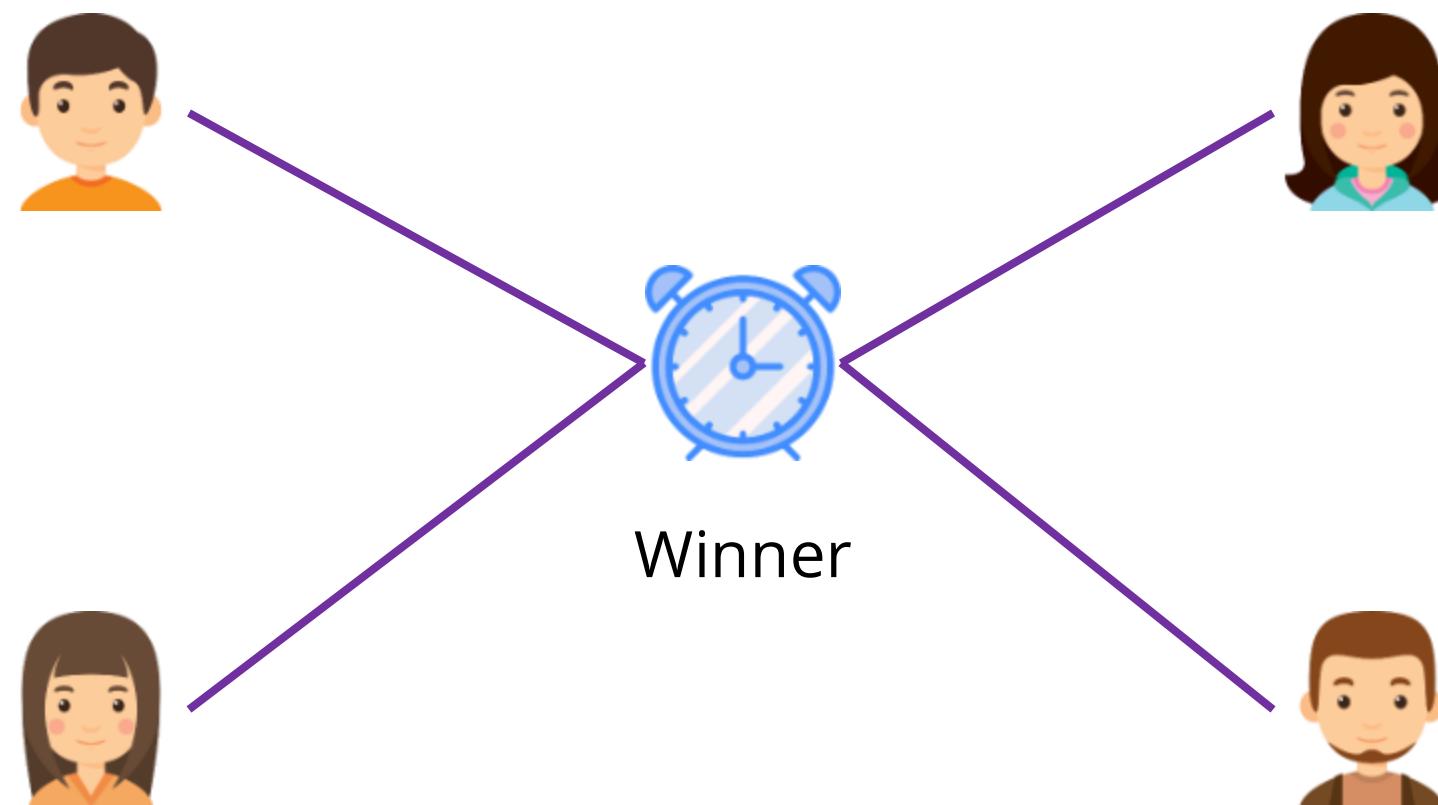
A low cost, low energy consuming algorithm which states that a person can mine and validate transaction based on how many coins he or she holds.



# Proof of Elapsed Time

Consensus algorithm prevents high energy consumption and resource utilization following a lottery system.

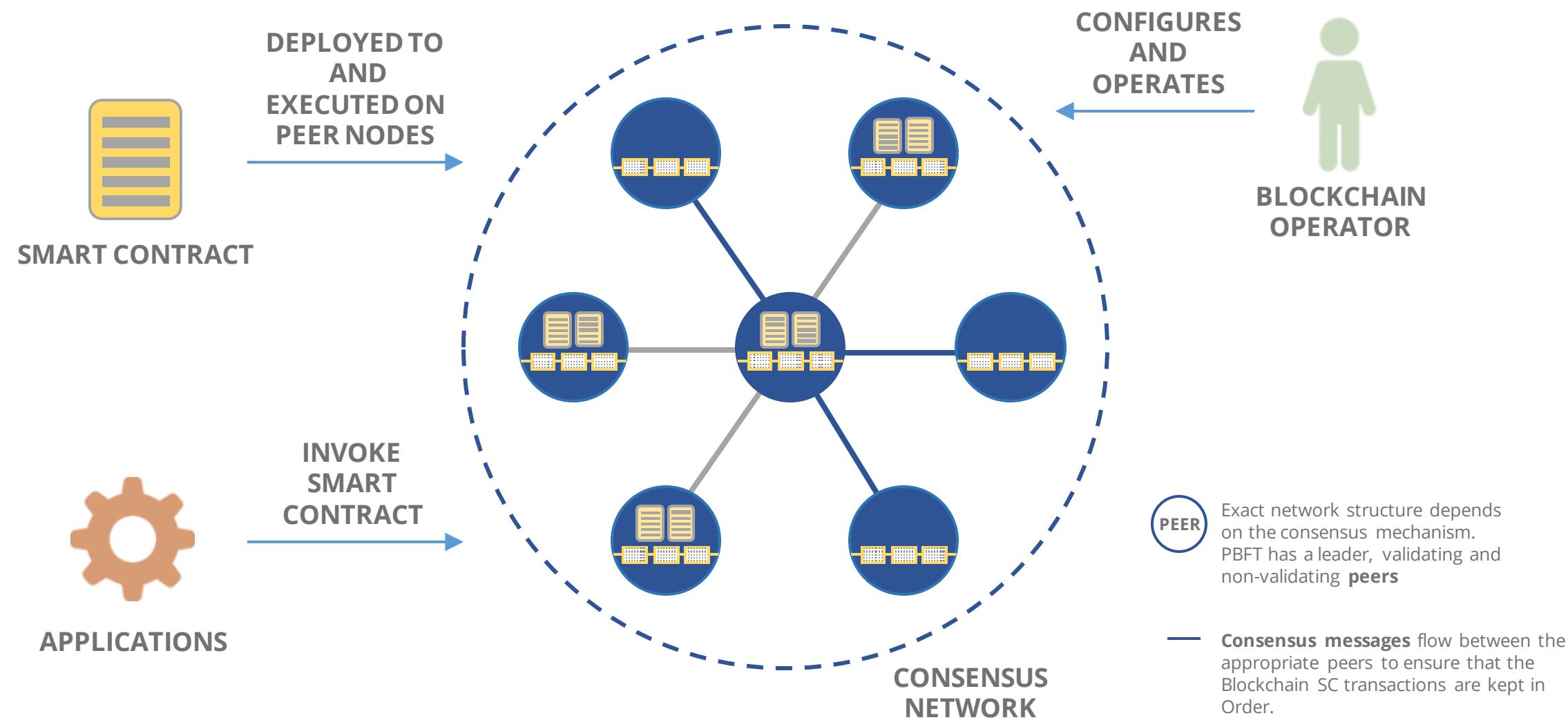
Each participant in the network is required to wait. One who completes the waiting time is the winner.



# PBFT

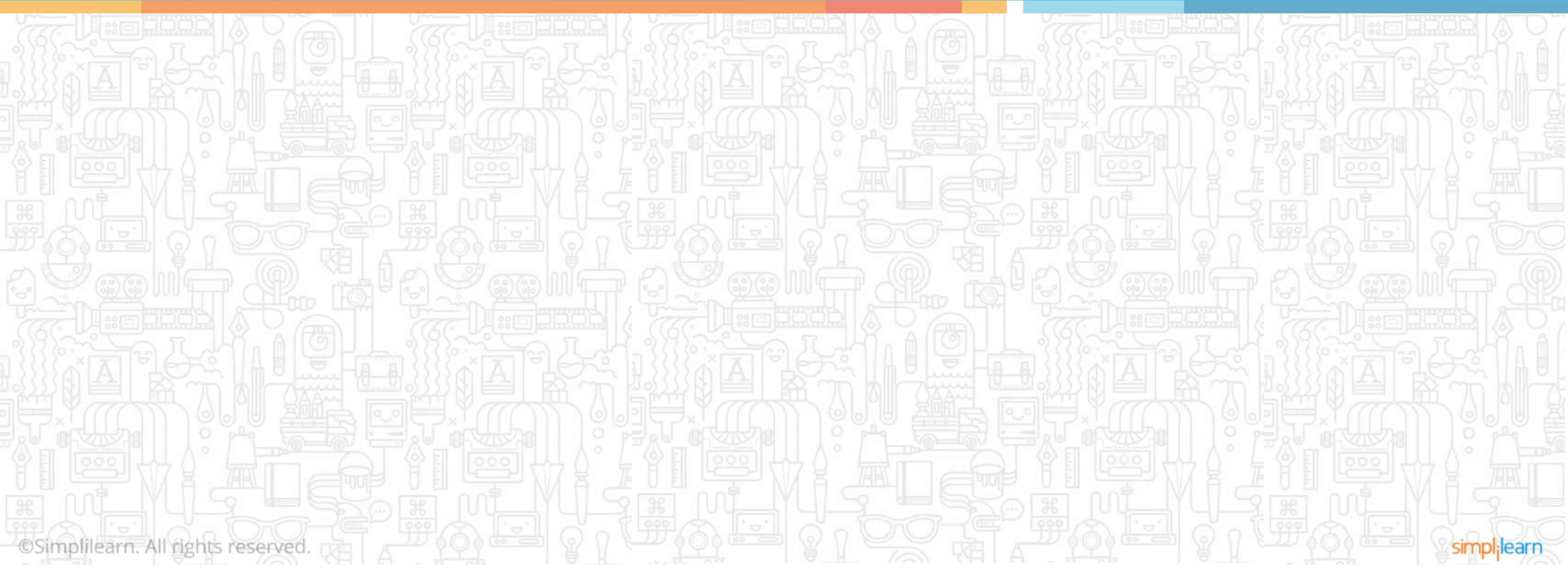
**Practical Byzantine Fault Tolerance (PBFT)** improves the robustness and performance of transaction by directing peer-to-peer messages with minimal latency.

## PRACTICAL BYZANTINE FAULT TOLERANCE

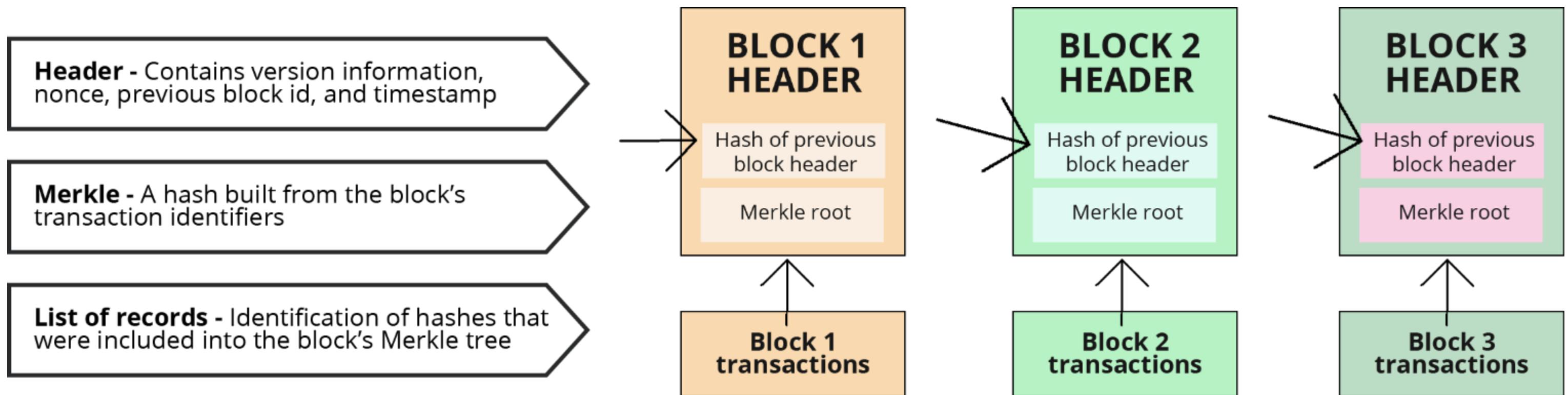


# Overview of Blockchain

## Block Creation



# Blockchain Block Structure



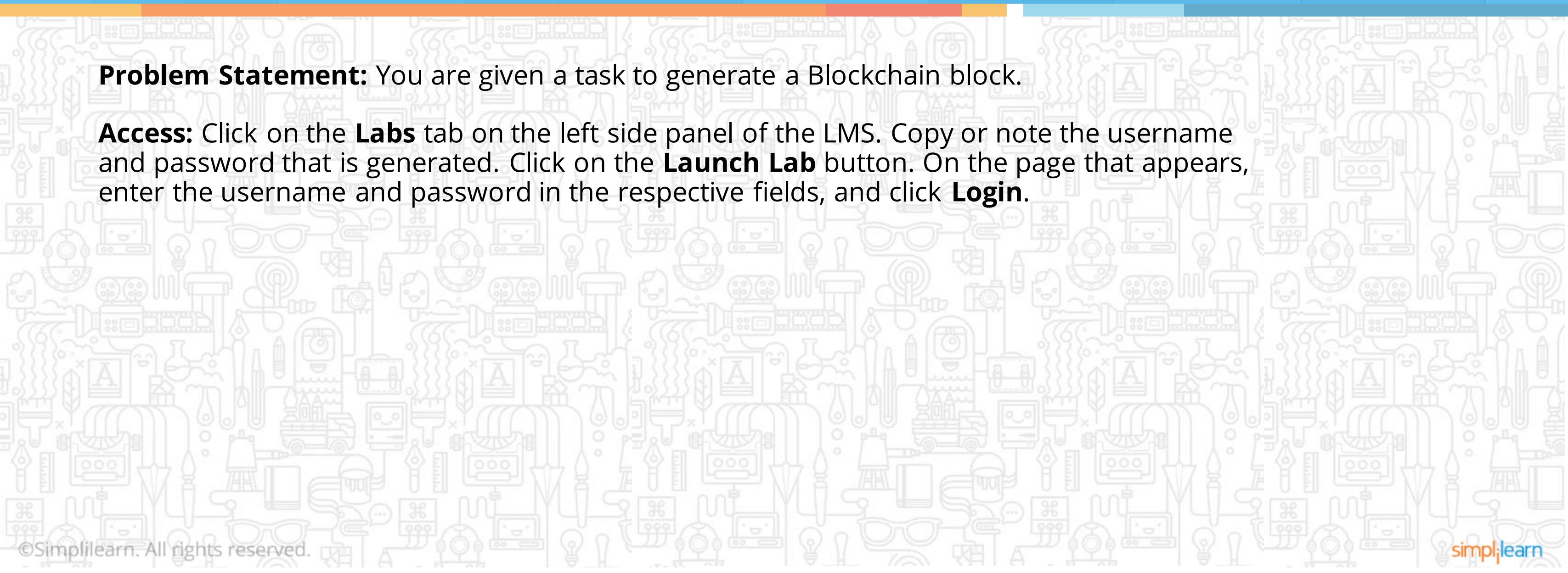
## Assisted Practice

Duration: 5 mins

### Generation of a Blockchain Block

**Problem Statement:** You are given a task to generate a Blockchain block.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/block.html>

### Step 02

Input any arbitrary data in the data field, and mine the block

### Step 03

Observe the structure of the generated block.

# Blockchain Header

---

Field	Description	Size
Magic number	Value always 0xD9B4BEF9	4 bytes
Block size	Number of bytes	4 bytes
Block header	Consists of six items	80 bytes
Transaction counter	Positive integer VI = VarInt	1-9 bytes
Transaction	The non-empty list of transactions	Depends on the number of transactions

# Blockchain Identifiers: Block Header

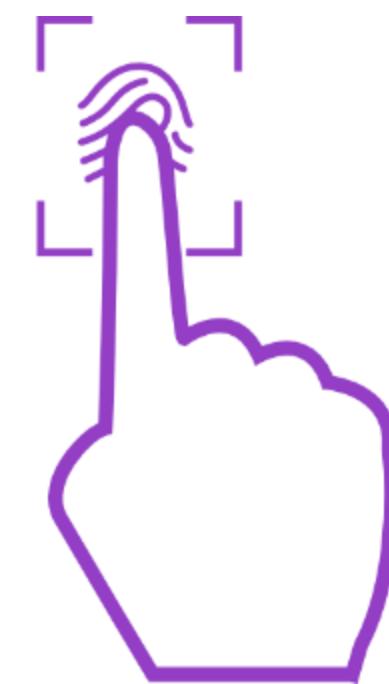
---

Primary identifier of a block

Digital fingerprint, twice the size of block header

Unique identification of a block

32-byte hash



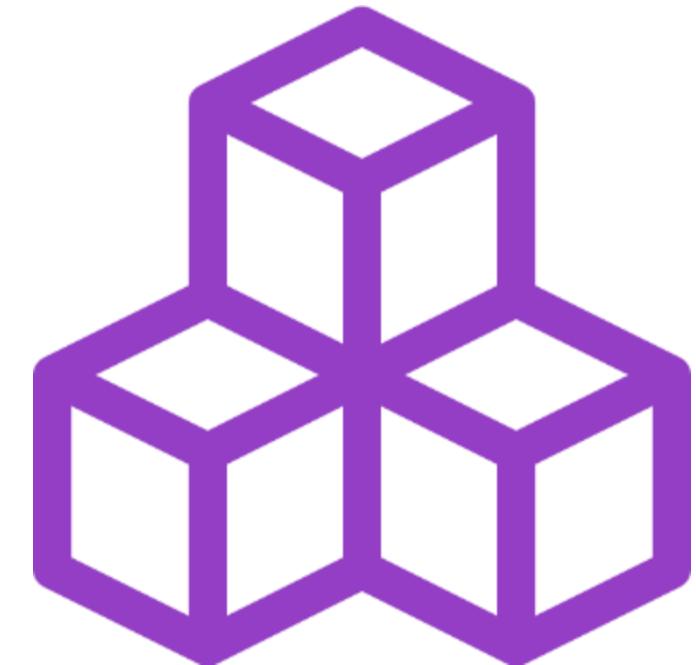
# Blockchain Identifiers: Block Height

---

Position of the block in Blockchain

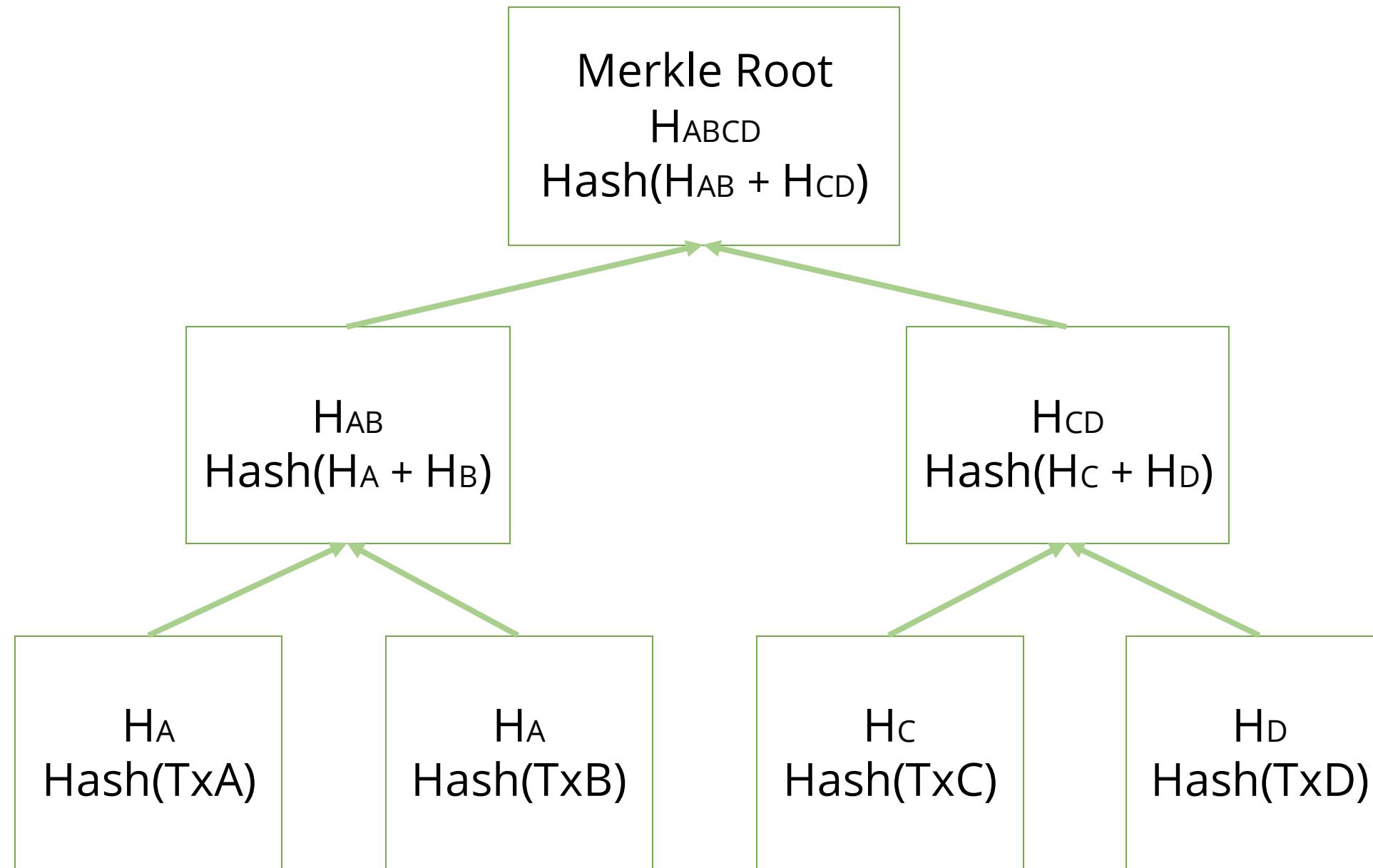
The first block is of height 0

Each node dynamically identifies a block

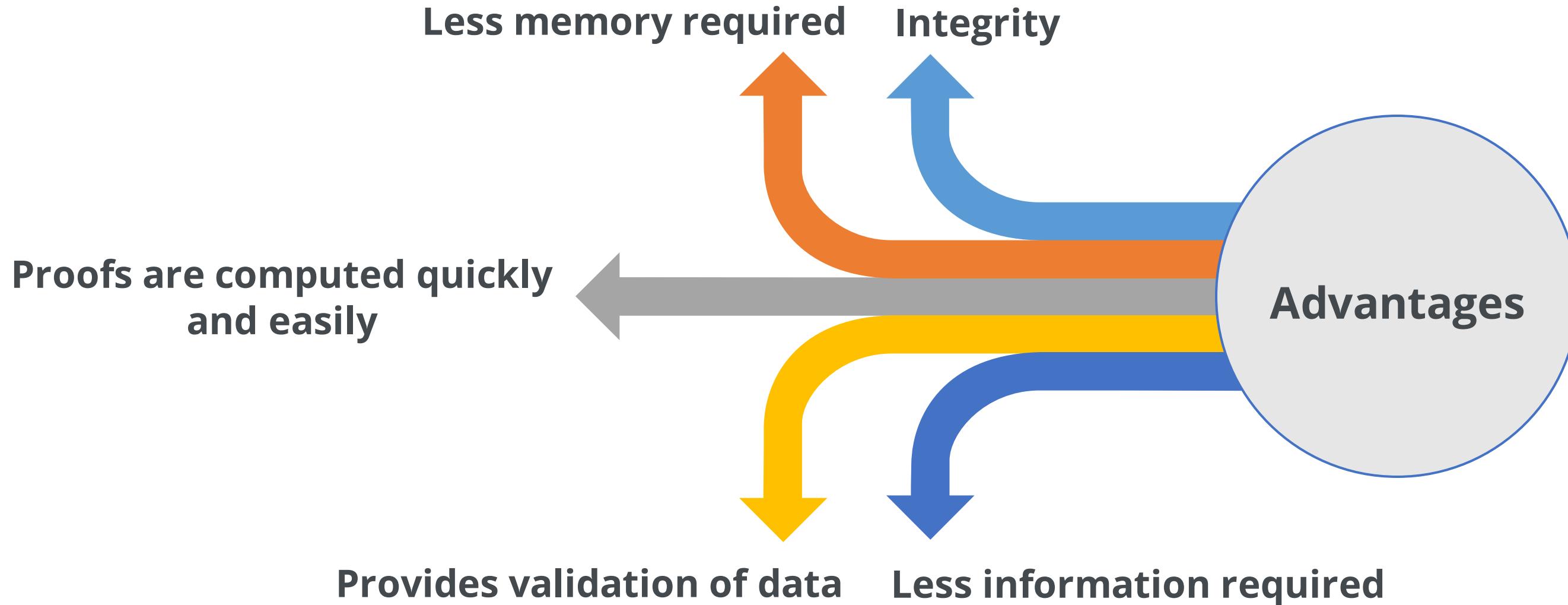


# Blockchain Merkle Tree

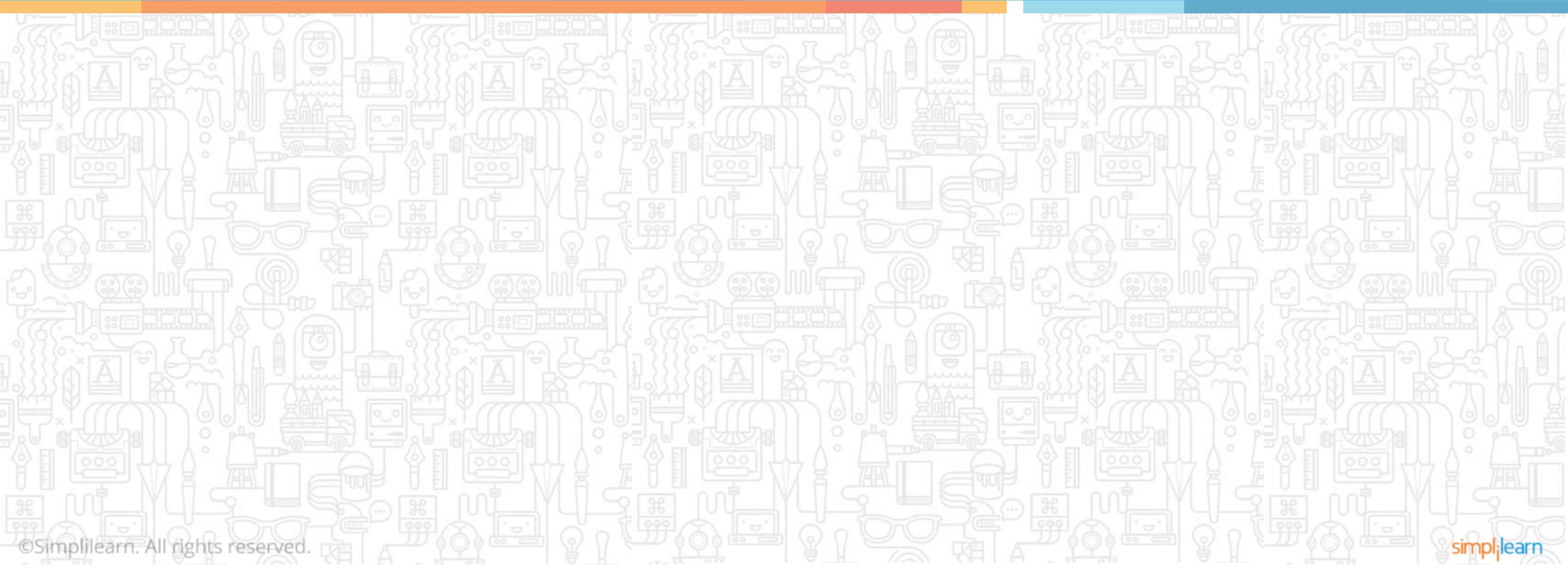
Data structure used for summarizing and verifying the integrity of large sets of data. It is also known as **Binary Hash Tree**.



# Advantages of Merkle Tree



# Overview of Blockchain Transaction Record



# Features of Blockchain



Cryptography  
Algorithm

Decentralized  
Network

Consensus  
Algorithm

**Distributed  
Ledger**

Blockchain stores records of each transaction in the distributed ledger.

# Evolution of Distributed Ledgers



clay tablets



papyrus



tally sticks

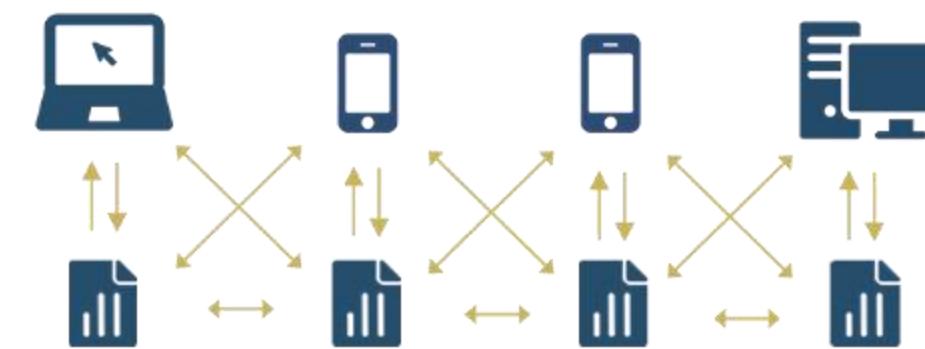


double entry book keeping

A screenshot of a Microsoft Excel spreadsheet titled "Untitled Spreadsheet". The table has columns labeled A through F. Column A contains row numbers from 1 to 30. Column B contains country names like "Ecuador", "Japan", "Russia", etc. Column C contains currency codes like "EUR", "JPY", "RUB", etc. Column D contains exchange rates like "3.2032 EUR = 1000". Column E contains dates like "06/05/2010". Column F contains values like "3.2028".

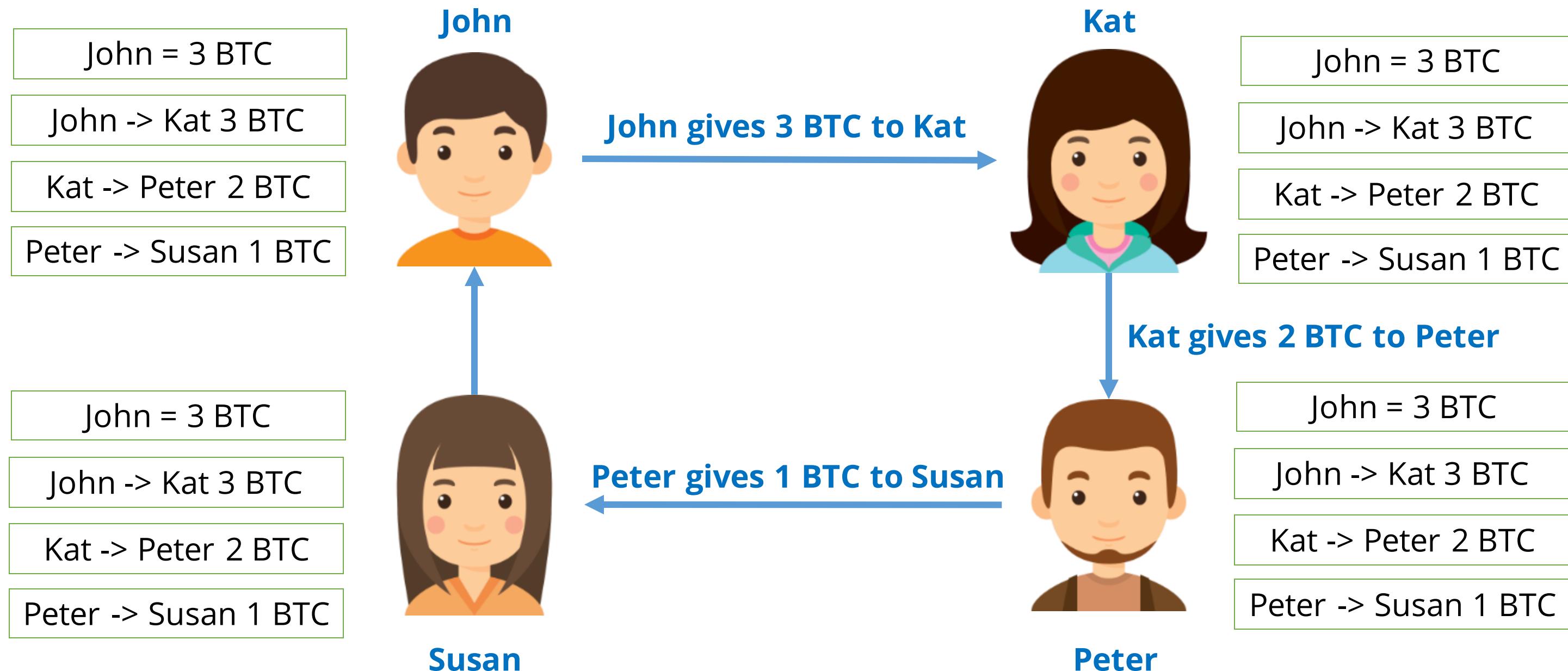
	COUNTRY	CURRENCY	LEVEL	VALUES	DATE	CHANGE
1						
2	Ecuador	EUR	3.2032 EUR = 1000	06/05/2010	3.2028	
3		JPY	32.032 JPY = 1000	06/05/2010	3.2021	
4		RUB	9.8904 RUB = 1000	06/05/2010	9	
5		USD	0.7740 USD = 1000	06/05/2010	0.7739	
6		CAD	1.3897 CAD = 1000	06/05/2010	1.3895	
7		AUD	1.3596 AUD = 1000	06/05/2010	1.3592	
8		NZD	1.5259 NZD = 1000	06/05/2010	1.5256	
9		SGD	0.7235 SGD = 1000	06/05/2010	0.7232	
10		NOK	8.2325 NOK = 1000	06/05/2010	8.2320	
11		DKK	1.3250 DKK = 1000	06/05/2010	1.3248	
12		BRL	3.6800 BRL = 1000	06/05/2010	3.6798	
13		CNY	0.3691 CNY = 1000	06/05/2010	0.3688	
14		RMB	0.3690 RMB = 1000	06/05/2010	0.3688	
15		INR	6.9496 INR = 1000	06/05/2010	6.9493	
16		IRR	2.8998 IRR = 1000	06/05/2010	2.8995	
17		MMK	0.0002 MMK = 1000	06/05/2010	0.0002	
18		PKR	10.8000 PKR = 1000	06/05/2010	10.7998	
19		BDT	2.0000 BDT = 1000	06/05/2010	1.9998	
20		THB	31.2000 THB = 1000	06/05/2010	31.1998	
21		VND	45.2000 VND = 1000	06/05/2010	45.1998	
22		CLP	600.0000 CLP = 1000	06/05/2010	600.0000	
23		ARS	0.0002 ARS = 1000	06/05/2010	0.0002	
24		UYU	1.0000 UYU = 1000	06/05/2010	1.0000	
25		PEN	0.0002 PEN = 1000	06/05/2010	0.0002	
26		BOB	0.0002 BOB = 1000	06/05/2010	0.0002	
27		GTQ	0.0002 GTQ = 1000	06/05/2010	0.0002	
28		HNK	0.0002 HNK = 1000	06/05/2010	0.0002	
29		PYG	0.0002 PYG = 1000	06/05/2010	0.0002	
30		VEF	0.0002 VEF = 1000	06/05/2010	0.0002	

spreadsheets



# Distributed Ledger

Ledgers store the copy of transactions that have happened. Every single person of the network has a copy of the ledger.



## Assisted Practice

Duration: 5 mins

### Working of a Distributed System

**Problem Statement:** You are given a task to demonstrate the working of a distributed system.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/distributed.html>

### Step 02

Note the hash value of Block 1-Peer A

### Step 03 value

Enter data in the data field of Peer A, click **Mine**, and note the hash

### Step 04

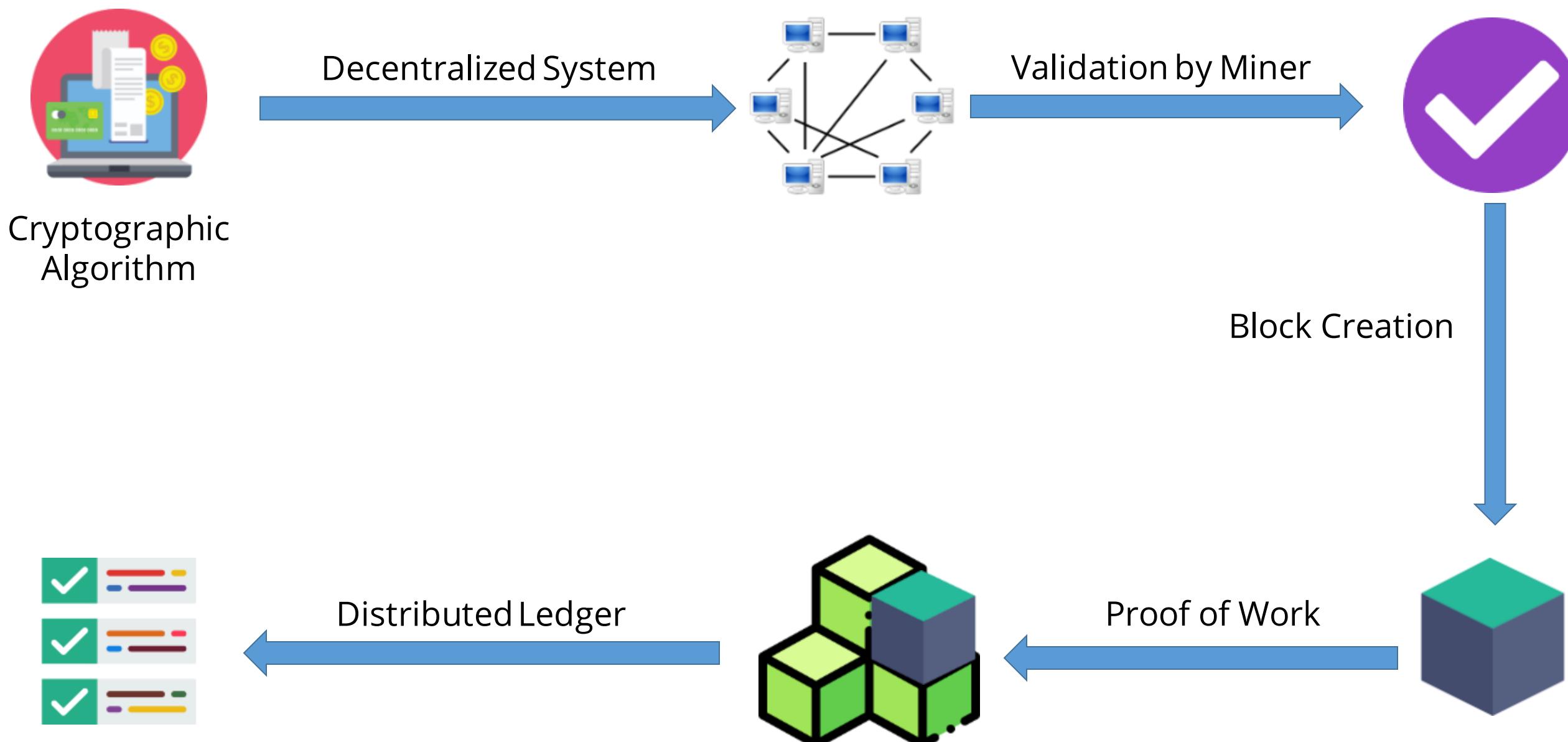
Enter data, and mine the other blocks of Peer A

### Step 05

Verify if the hash of previous block is same in the next

# Features of Blockchain in Transaction Process

The diagrams shows the usage of Blockchain features in every step of the Blockchain transaction process.



## Assisted Practice

Duration: 5 mins

### Working of a Blockchain Transaction

**Problem Statement:** You are given a task to demonstrate the working of a Blockchain transaction.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/blockchain.html>

### Step 02

Enter any data in the data field, and observe the change in hash

### Step 03 value

Change the information in any block, and observe the change in hash

### Step 04

Observe how the hash values change in the subsequent blocks

## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/tokens.html>

### Step 02

Change any value of the token in any of the blocks

### Step 03

Mine the block, and observe change in the hash value

## Assisted Practice: Steps



### Step 01

Visit <https://anders.com/blockchain/coinbase.html>

### Step 02

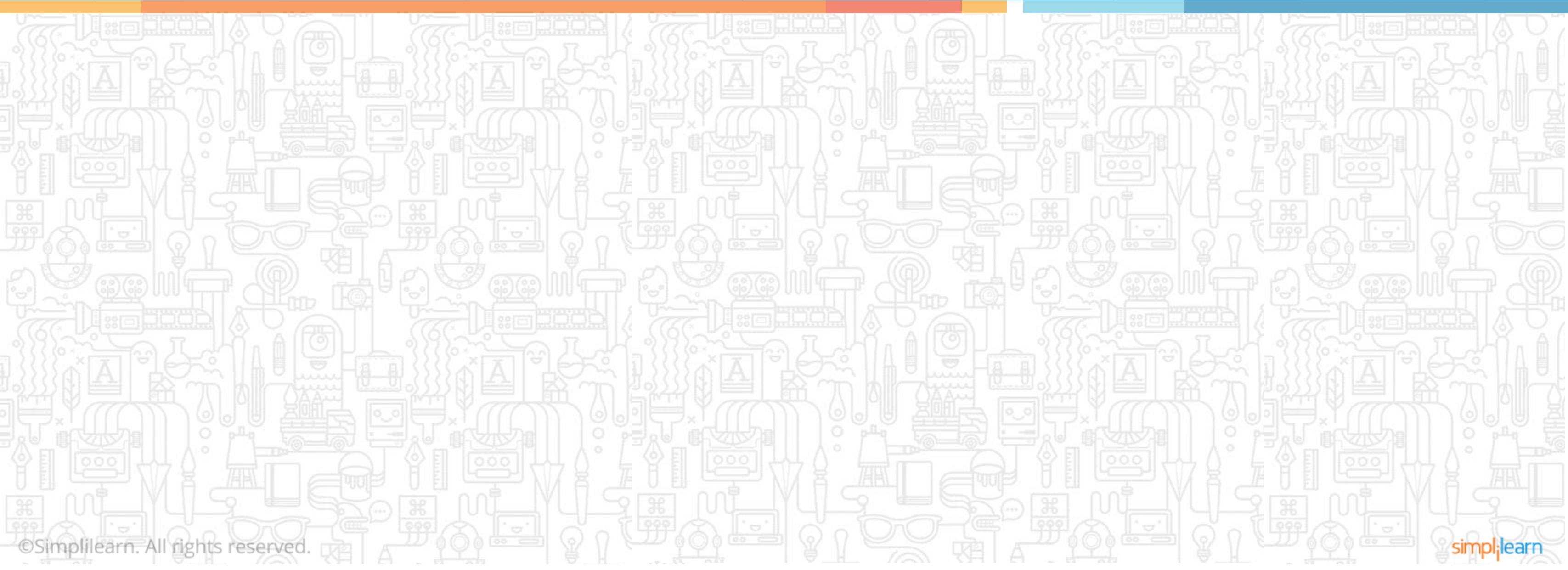
Transfer money from Ander's account to four different accounts

### Step 03 to transfer

Observe the Coinbase transaction, and check if the accounts have money

# Overview of Blockchain

## Types of Blockchain



# Types of Blockchain

Public



Bitcoin



Ethereum



Dash



Factom

Private

Consortium

Ledgers are publicly available to verify or add blocks to the Blockchain

# Types of Blockchain

Public

Private

Consortium



**MultiChain**



**BLOCKSTACK**

Only authorized users can add or verify the blocks, but anyone can view it

# Types of Blockchain

Public

Private

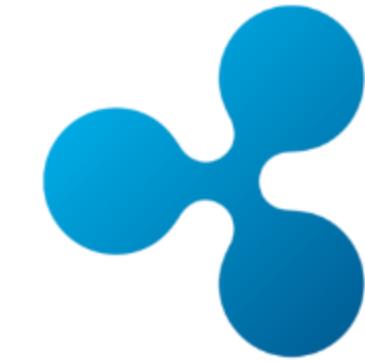
Consortium



Hyperledger 1.0



R3

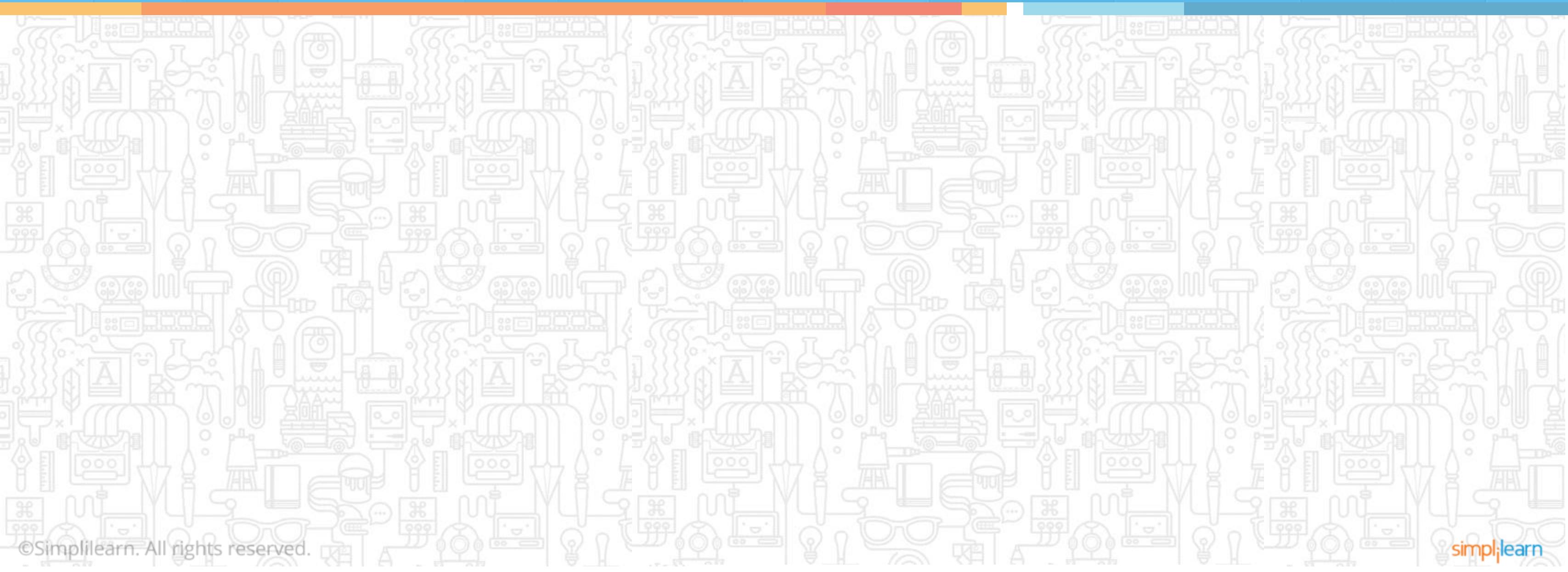


Ripple

Semi-centralized Blockchain in which only predefined set of nodes has permission to write the block

# Overview of Blockchain

## Blockchain Platforms, Application Components, and Templates



# Blockchain Platforms

---



ethereum

**Based on Proof of Work**



**Pluggable frameworks**



**Cross-border payments**



**Federated consensus  
mechanism**

# Blockchain Platforms

---



**HYPERLEDGER**

**More than 185 collaborating  
companies**



**Backs nano payments**



**High degree of authority**



**Majority voting consensus  
mechanism**

# Blockchain Application Components

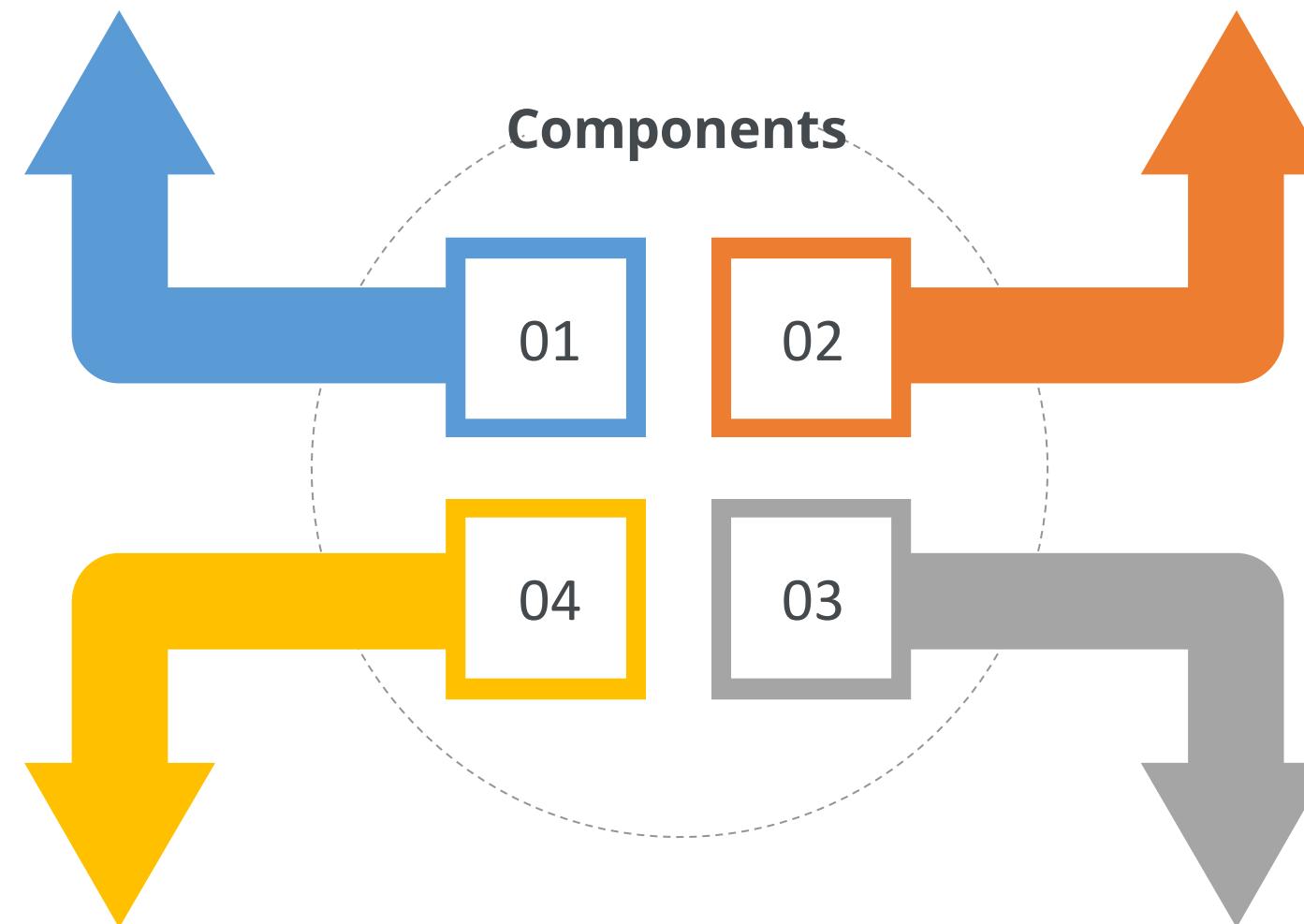
## **Node Application**

Each node has to install and run a computer application definitive to the ecosystem.

## **Virtual Machine**

An abstraction of instruction-operated machine implemented as a part of node application.

## **Components**



## **Shared Ledger**

A ledger is managed inside the node application whose contents can be viewed once the application is running.

## **Consensus Algorithm**

It provides the rule for how the ecosystem will arrive at a single view in a ledger, and how it will be implemented as a part of node application.

# Blockchain Application Templates

**Many-to-One**  
Used for applications where a user sets up the contracts for a specific purpose.



**One-to-One**  
Used for applications which involve financial contracts between two entities.

## Types of Templates

# Key Takeaways



You are now able to:

- ➊ Describe the Blockchain transaction process
- ➋ Generate a public key and a digital signature
- ➌ Generate a nonce, a hash code, and a Blockchain block
- ➍ Work with a distributed system and perform Blockchain transaction



## Knowledge Check



Knowledge  
Check

1

**In which algorithm is the peer-to-peer message directed with minimal latency?**

- a. Proof of Work
- b. Proof of Stake
- c. Practical Byzantine Fault Tolerance
- d. Proof of Elapsed Time



## In which algorithm is the peer-to-peer message directed with minimal latency?

- a. Proof of Work
- b. Proof of Stake
- c. Practical Byzantine Fault Tolerance
- d. Proof of Elapsed Time



The correct answer is C

**PBFT improves the robustness and performance by directing peer-to-peer messages with minimal latency.**

**What is the code generated by taking an input and converting it to cryptographic output using mathematical algorithm?**

- a. Pseudo code
- b. Hash code
- c. ASCII code
- d. Binary code



**What is the code generated by taking an input and converting it to cryptographic output using mathematical algorithm?**

- a. Pseudo code
- b. Hash code
- c. ASCII code
- d. Binary code



The correct answer is **b**

**Blockchain uses Hash code for mathematical algorithms such as Proof of Work.**

## What is a miner?

- a. An algorithm that predicts the next part of the chain
- b. A type of Blockchain
- c. An application that processes and validates Blockchain transactions
- d. A person doing calculations to verify a transaction



## What is a miner?

- a. An algorithm that predicts the next part of the chain
- b. A type of Blockchain
- c. An application that processes and validates Blockchain transactions
- d. A person doing calculations to verify a transaction



The correct answer is

C

**Miner is an application that performs the validation of the transaction using consensus algorithm.**

## Which algorithm prevents resource utilization following a lottery system?

- a. Proof of Work
- b. Proof of Stake
- c. Proof of Elapsed Time
- d. Practical Byzantine Fault Tolerance



## Which algorithm prevents resource utilization following a lottery system?

- a. Proof of Work
- b. Proof of Stake
- c. Proof of Elapsed Time
- d. Practical Byzantine Fault Tolerance



The correct answer is C

**Proof of Elapsed Time increases the efficiency of the process by preventing resource utilization.**

Knowledge  
Check

5

**What is the random number called whose value is set so that the hash of the block will contain a run of leading zeros?**

- a. Hash code
- b. Hash pointer
- c. Binary code
- d.Nonce



**What is the random number called whose value is set so that the hash of the block will contain a run of leading zeros?**

- a. Hash code
- b. Hash pointer
- c. Binary code
- d.Nonce



The correct answer is

**d**

**Nonce is used to hash the block so that it will contain a run of leading zeros.**

# Lesson-End Project

Duration: 10 mins

## Creating a Blockchain Network

### Problem Statement:

You are an employee of Simplilearn and have been asked to create a Blockchain network to demonstrate the working of Blockchain. You have to perform the following actions:

- Create blocks for the existing peer
- Create another peer and blocks for the same
- Connect the newly created peer with the existing peer

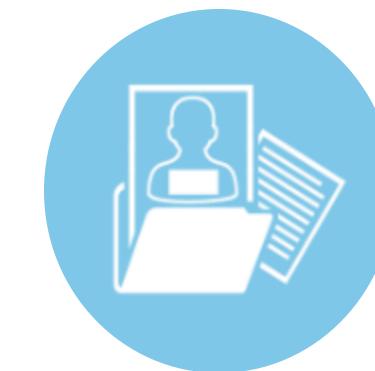
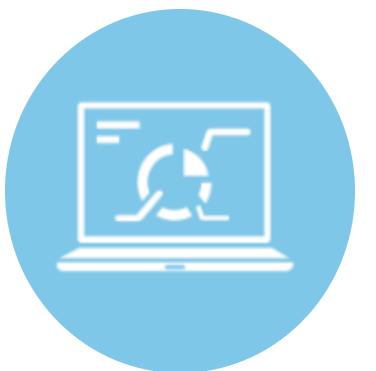
**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 2: Bitcoin Blockchain



# Learning Objectives

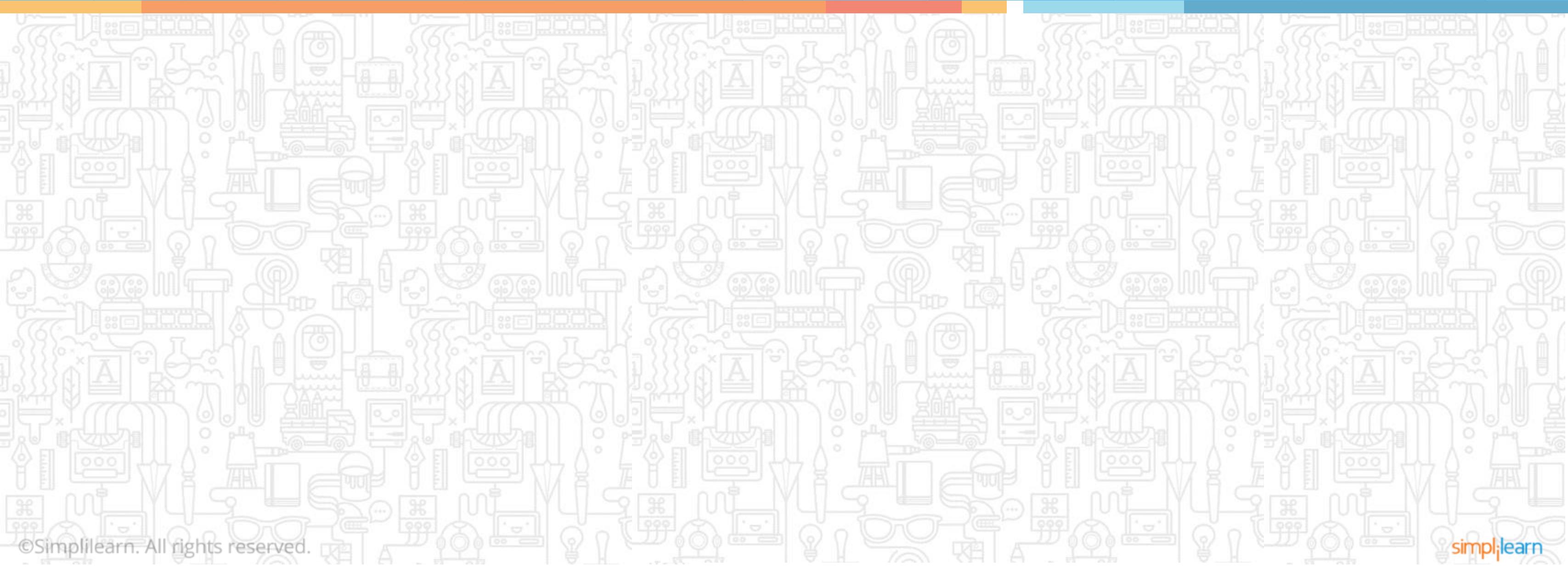
By the end of this lesson, you will be able to:

- ➊ Explain Bitcoin and ways to procure it
- ➋ Set up a Bitcoin wallet
- ➌ Identify where to spend Bitcoin
- ➍ Describe the transaction structure of Bitcoin Blockchain
- ➎ Interpret the role of Bitcoin Scripts in transactions
- ➏ Explain the network and nodes in Bitcoin
- ➐ Identify the steps involved in Bitcoin mining



# Bitcoin Blockchain

## Bitcoin and Ways to Acquire Them



# Introduction to Bitcoin

Bitcoin was the first digital currency that facilitated instant transfer of value across the world.

Transaction Management:  
transferring the coins from  
one address to another

1

Two major  
operations in  
Bitcoin system

2

Currency generation:  
regulating the monetary value

# Bitcoin Controlled Supply

Bitcoin supply is limited for the currency to have value

The rate of block creation is adjusted every 2016 blocks

The number of Bitcoins generated per block decreases by 50% every 210,000 blocks

Date reached	Block	Reward Era	BTC/block	Year (estimate)	Start BTC	BTC Added	End BTC	BTC Increase	End BTC % of Limit
2009-01-03	0	1	50.00	2009	0	2625000	2625000	infinite	12.500%
2010-04-22	52500	1	50.00	2010	2625000	2625000	5250000	100.00%	25.000%
2011-01-28	105000	1	50.00	2011*	5250000	2625000	7875000	50.00%	37.500%
2011-12-14	157500	1	50.00	2012	7875000	2625000	10500000	33.33%	50.000%
2012-11-28	210000	2	25.00	2013	10500000	1312500	11812500	12.50%	56.250%
2013-10-09	262500	2	25.00	2014	11812500	1312500	13125000	11.11%	62.500%
2014-08-11	315000	2	25.00	2015	13125000	1312500	14437500	10.00%	68.750%
2015-07-29	367500	2	25.00	2016	14437500	1312500	15750000	9.09%	75.000%
2016-07-09	420000	3	12.50	2016	15750000	656250	16406250	4.17%	78.125%
2017-06-23	472500	3	12.50	2018	16406250	656250	17062500	4.00%	81.250%
	525000	3	12.50	2019	17062500	656250	17718750	3.85%	84.375%
	577500	3	12.50	2020	17718750	656250	18375000	3.70%	87.500%
The number of Bitcoins generated per block decreases by 50% every 210,000 blocks	630000	4	6.25	2021	18375000	328125	18703125	1.79%	89.063%
	682500	4	6.25	2022	18703125	328125	19031250	1.75%	90.625%
	735000	4	6.25	2023	19031250	328125	19359375	1.72%	92.188%
	787500	4	6.25	2024	19359375	328125	19687500	1.69%	93.750%

# Ways to Get Bitcoins



## Buying online

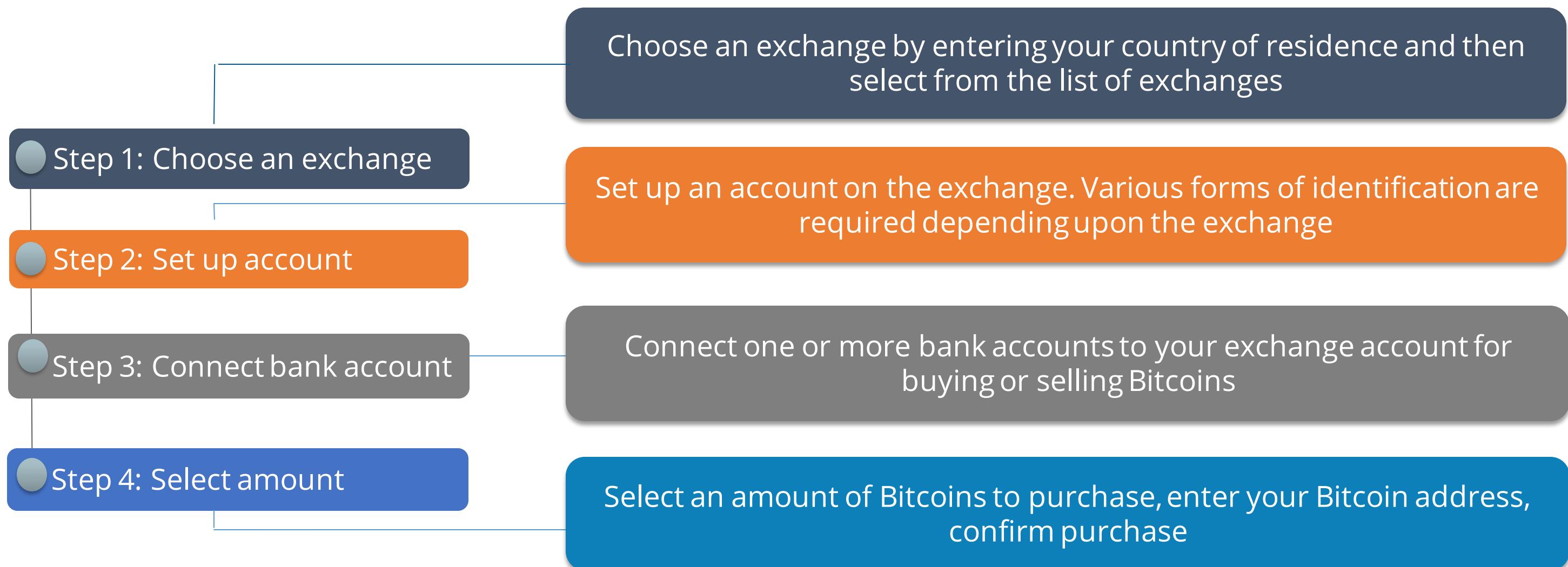
- Create an account on an online exchange
- Exchanges are used to buy and sell Bitcoins on your behalf

## Buying with cash

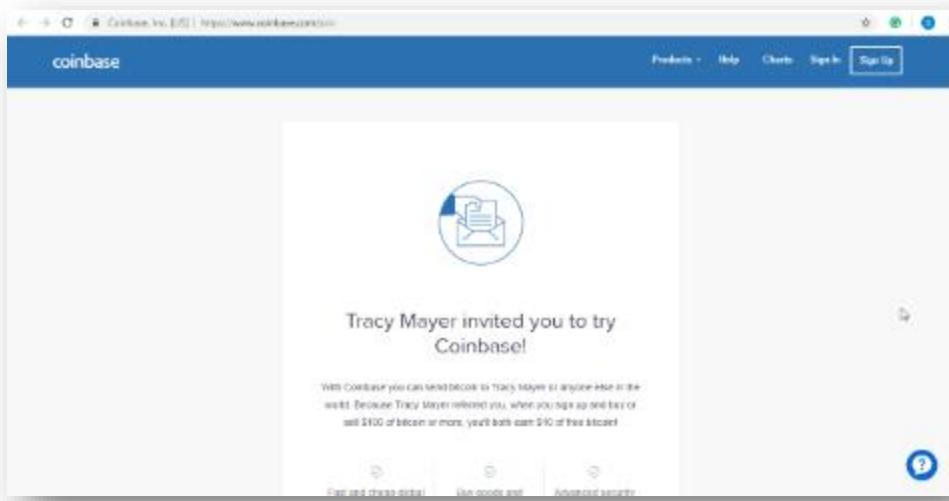
- Choose a purchase method
- Platforms like LocalBitcoins will help you find people near you who are willing to exchange Bitcoins for cash
- There are ATMs that send Bitcoins to your wallet in exchange for cash

# Buying Bitcoin from an Exchange

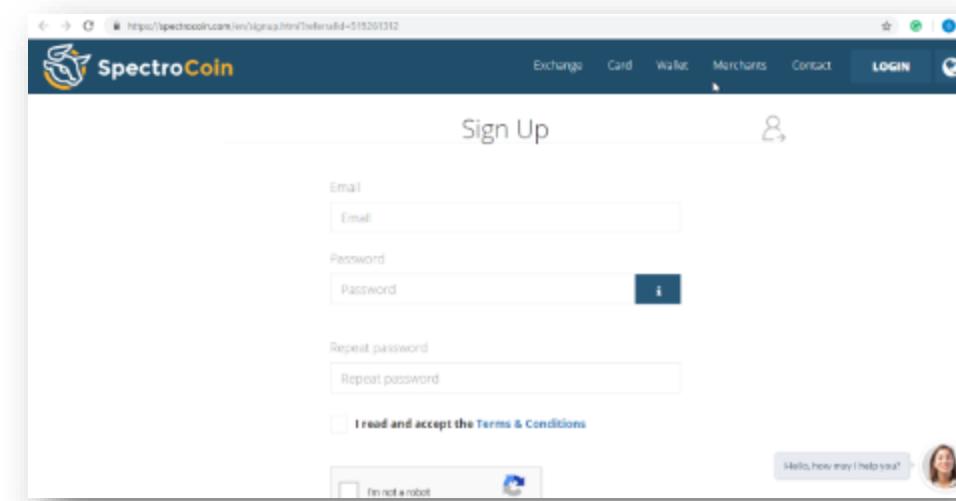
Follow this basic rundown to buy Bitcoin from an exchange:



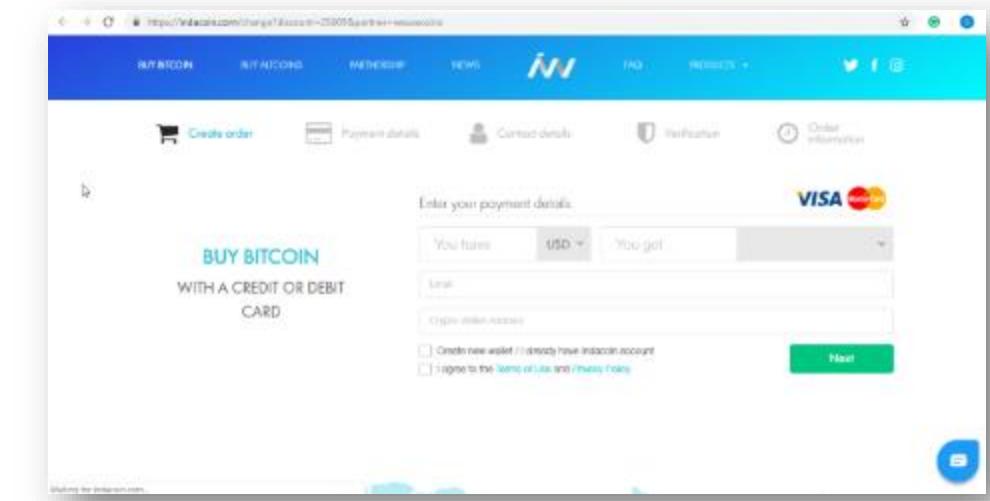
# Best Bitcoin Exchanges



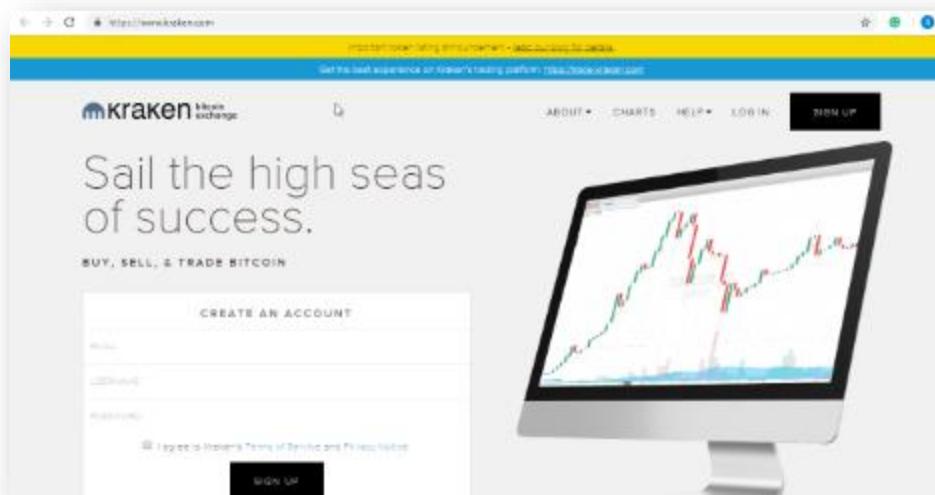
**Coinbase**  
One of the world's largest Bitcoin exchanges



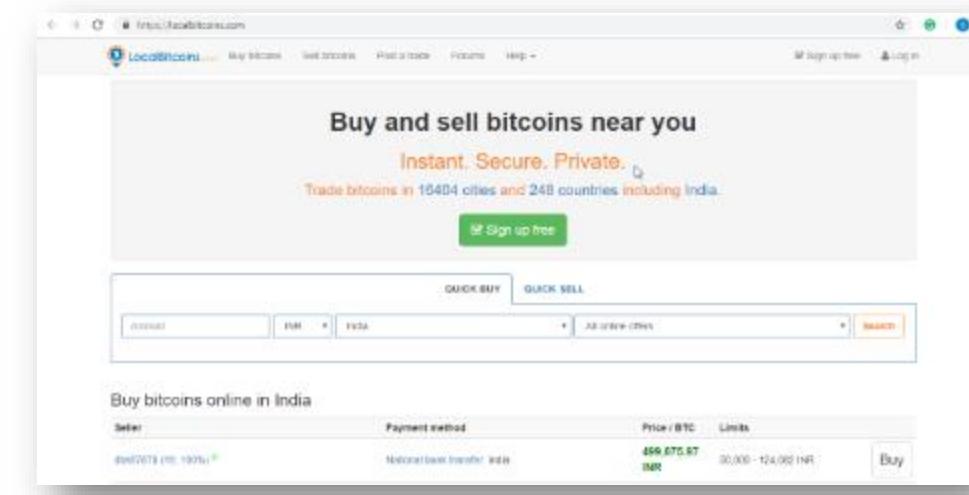
**SpectroCoin**  
Quick and easy exchange offering more than 20 payment options



**Indacoin**  
Let's you buy Bitcoins and 700 other coins with Visa & Mastercard



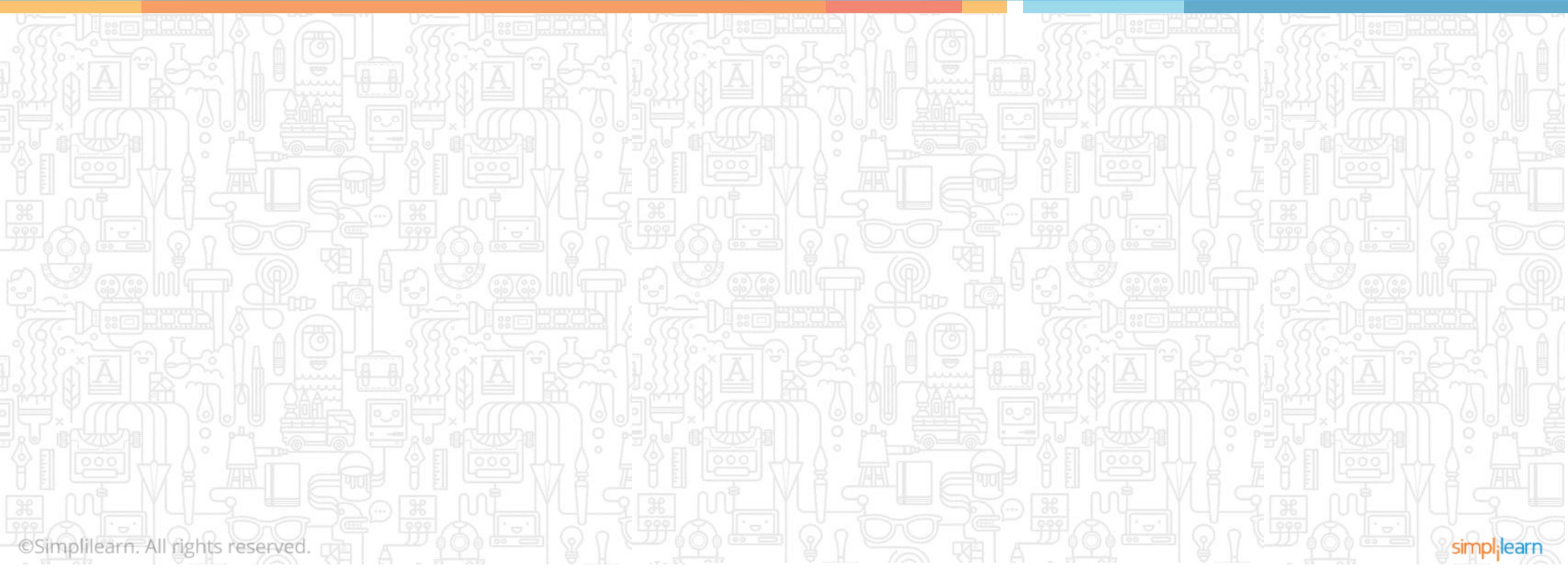
**Kraken**  
Best exchange for deposit and purchase of Bitcoins on same day



**LocalBitcoins**  
An escrow service that helps match Bitcoin buyers and sellers

# Bitcoin Blockchain

## Bitcoin Wallets and Ways to Set Them Up



## Wallets

---



Bitcoins wallets are digital wallets where private keys are used to access the Bitcoin address and signature for transactions

# Types of Bitcoin Wallets

## Software Wallets

Software wallets are installed on your computer to store your private keys

## Web Wallets

Online wallets are convenient, because you can access your funds from any device

## Hardware Wallets

Hardware wallets are used to store private keys on storage devices such as USB drives

## Brain Wallets

Computer creates a passphrase that user commits to memory

## Mobile Wallets

Mobile wallets run as apps storing your private keys to pay directly from your phone

# Best Bitcoin Wallets

---

## Software Wallets



**Armory**



**Bitcoin Core**



**MultiBit HD**

## Web Wallets

**coinbase**



## Hardware Wallets



**Ledger**



**TREZOR**



**OPENDIME**

## Mobile Wallets



**Breadwallet**



**Mycelium**



**Bitcoin Wallet**

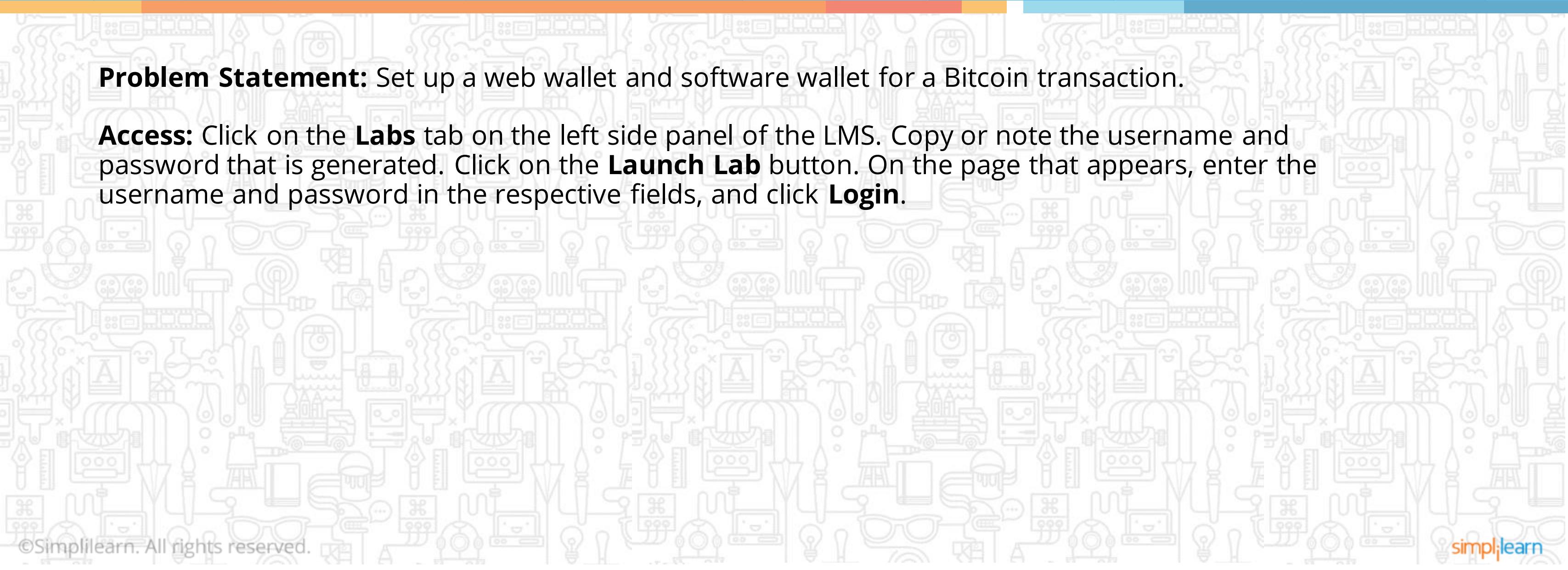
# Assisted Practice

## Bitcoin Wallet

Duration: 15 mins

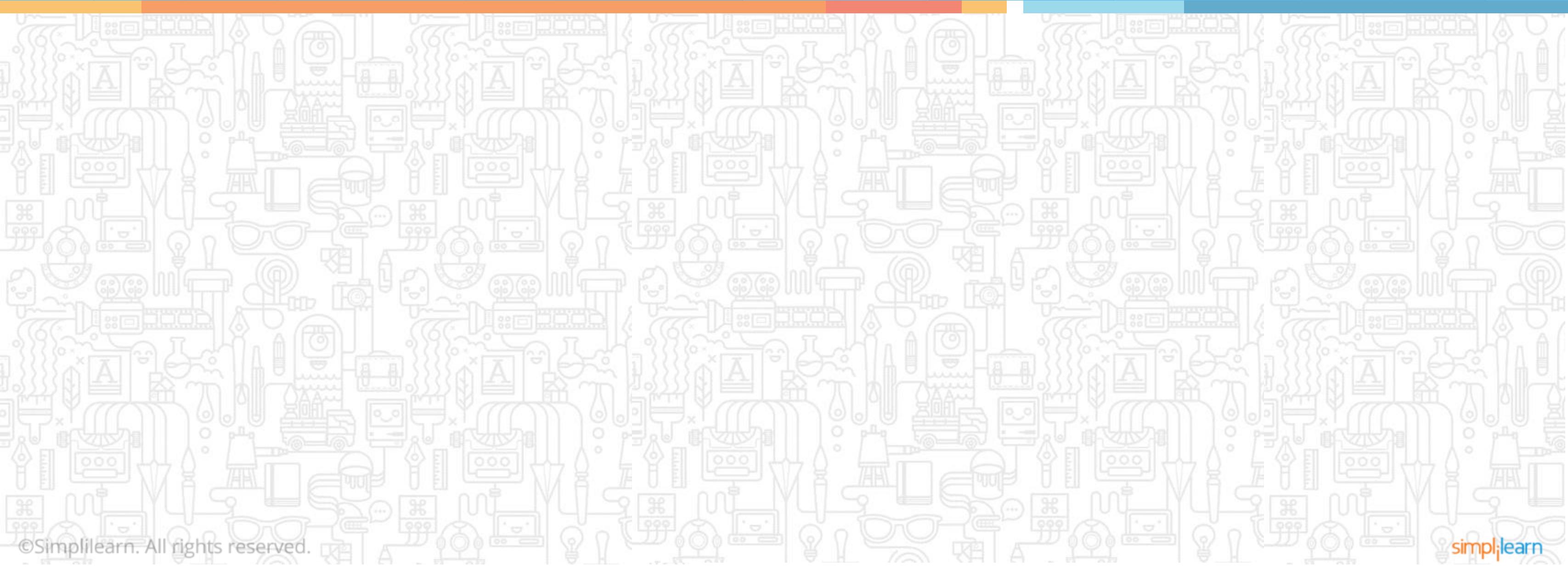
**Problem Statement:** Set up a web wallet and software wallet for a Bitcoin transaction.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



# Bitcoin Blockchain

## Examples of Where to Use Bitcoins



# Spending Bitcoins



## Spending Online

- Common household items: There are millions of products you can buy using Bitcoins
- Gift cards: You can buy gift cards online using Bitcoins
- Video games: Companies have started offering games and other apps in Bitcoins

## Spending Offline

- Food: You can buy food with Bitcoins now. There is a coffee shop in Prague that only accepts Bitcoins
- Travel: Pay for hotel bookings online using Bitcoins. Also, you can buy tickets of ships, cruises, flights

# Places to Spend Bitcoins

---

## Overstock.com

First big online retailer to accept Bitcoins

## Shopify stores

An e-commerce platform where you can pay in Bitcoins

## CheapAir

Choose your flights and pay in Bitcoins for flying with CheapAir

## Expedia

Expedia provides the option to pay for their hotel bookings with Bitcoins

## eGifter

A popular gift card site accepting Bitcoins as payments

## Reeds jewelry Inc.

American Jewelers where you can purchase jewelry with Bitcoins

## Newegg

An electronic retail giant that accepts Bitcoin

## Dish

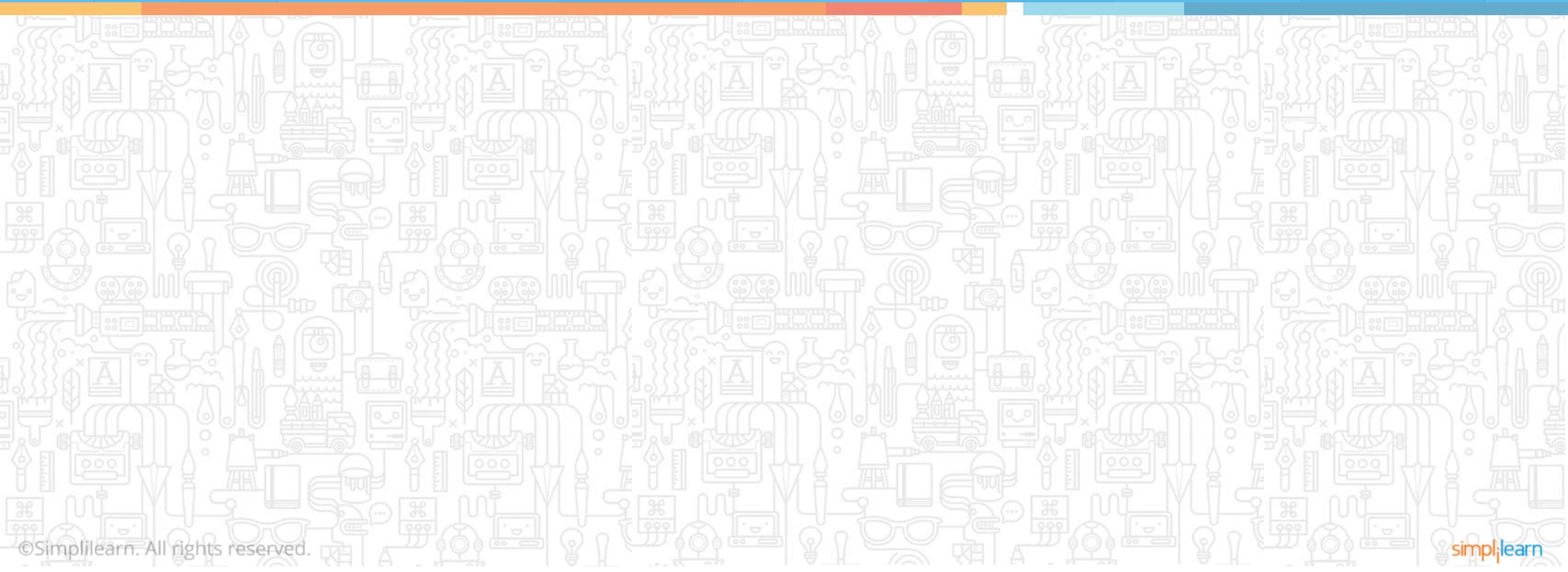
Satellite TV and internet provider that started accepting Bitcoin payment in 2014

## Microsoft

You can use Bitcoins to deposit funds into your Microsoft account

# Bitcoin Blockchain

## Transaction Structure



# Transaction in Bitcoin System

Scenario: Ross wants to send 5 BTC to Joey

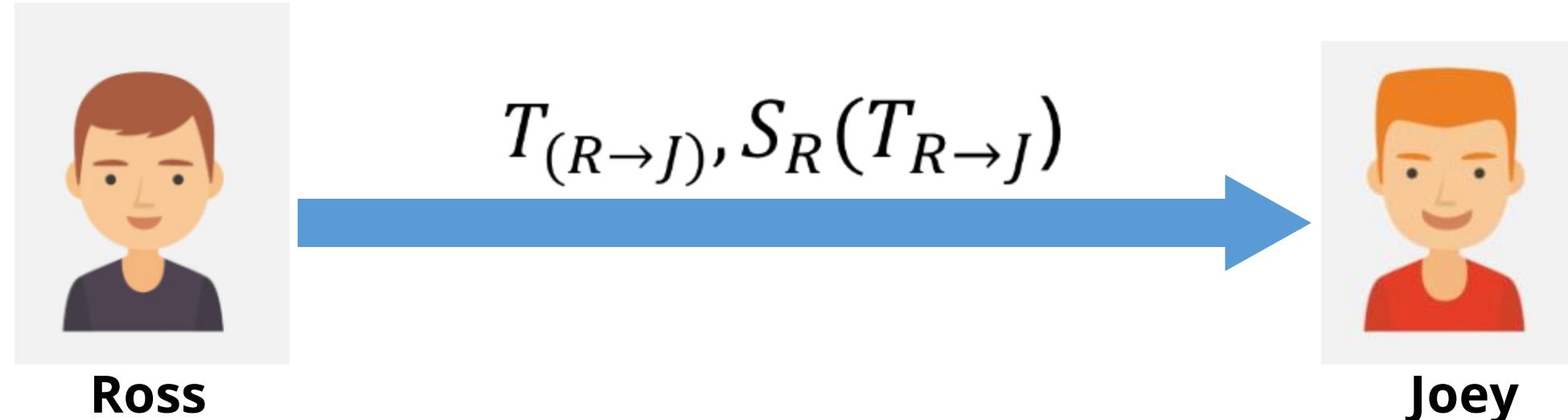
Here's how the transaction happens:

**Step 1:** Joey shares his address with Ross

**Step 2:** Ross creates a transaction message and adds Joey's address and 5 BTC to it

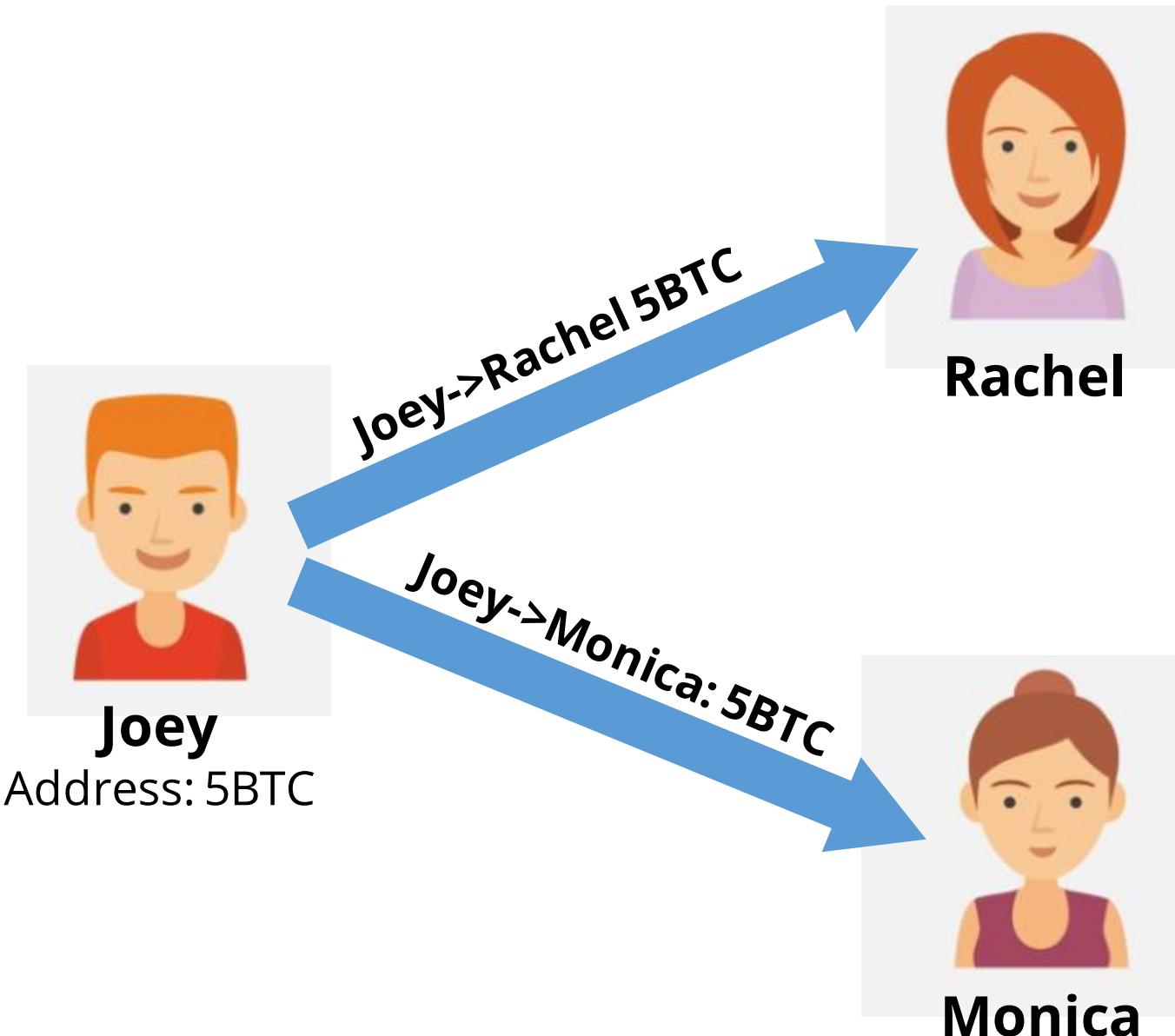
**Step 3:** Ross then signs the transaction with his private key and shares his public key to the network for signature verification

**Step 4:** Ross broadcasts the transaction on the Bitcoin network



# Double Spending Problem

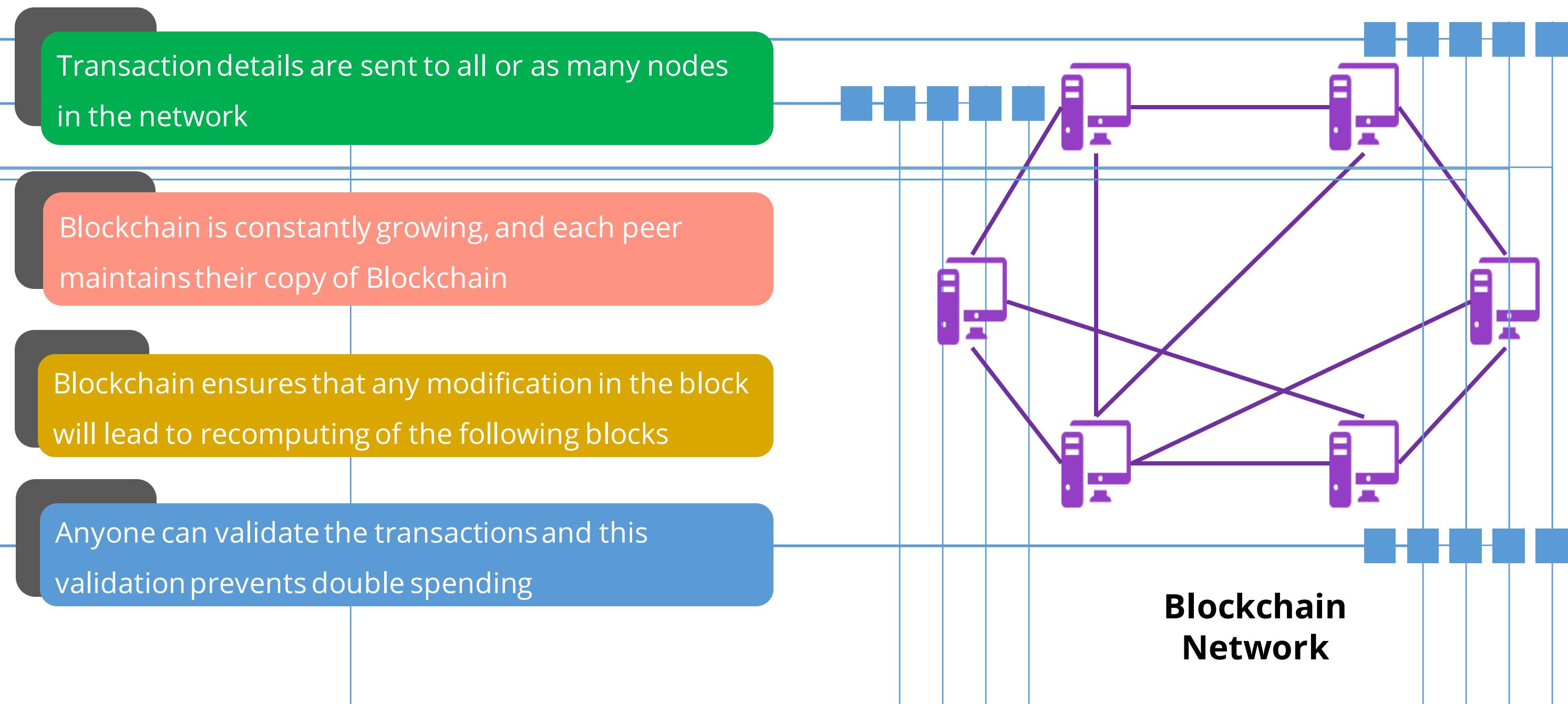
Scenario: If Joey uses his BTC in more than one transaction



Spending the same Bitcoins in more than one transaction is called double spending problem

# Encountering Double Spending in Blockchain

The security measures which prevent double spending in Blockchain:



# Pseudonymity in Bitcoin

Pseudonymity is the near-anonymous state in which users have disguised identities and do not disclose their true identities

- Bitcoin is a permissionless Blockchain where you don't need to set up an account
- The public and private keys are generated by the wallet
- The address acts as an identifier or pseudonym of a user's transaction

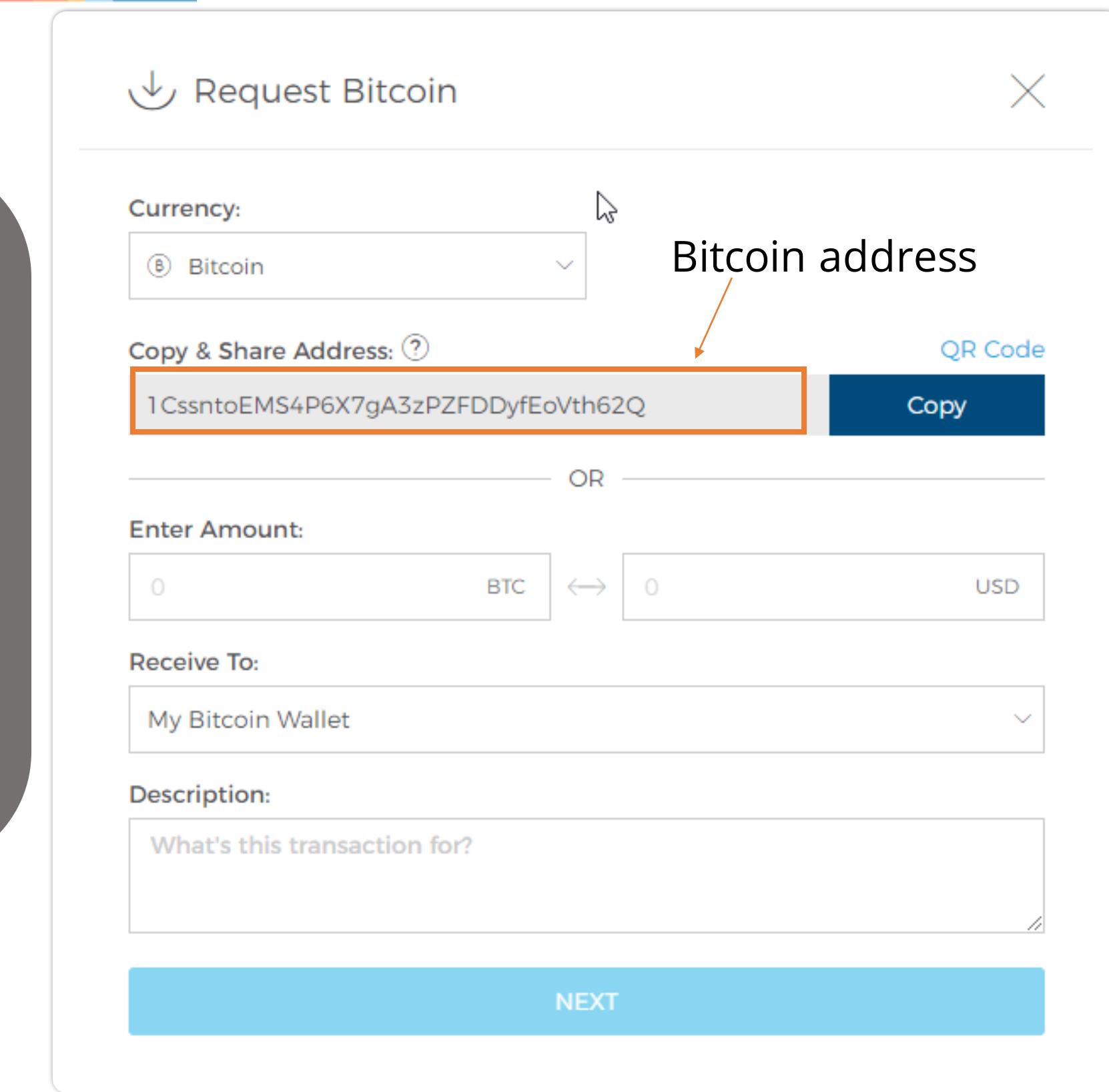


# Bitcoin Addresses

The Bitcoin address corresponds to a public key based on ECDSA(Elliptic Curve Digital Signature Algorithm) used in Bitcoin

Here is a sample Bitcoin address:  
1PHYrmdJ22MKbJevpb3MBNpVckjZHt89hz

A wallet can have many such addresses and can be used for transactions

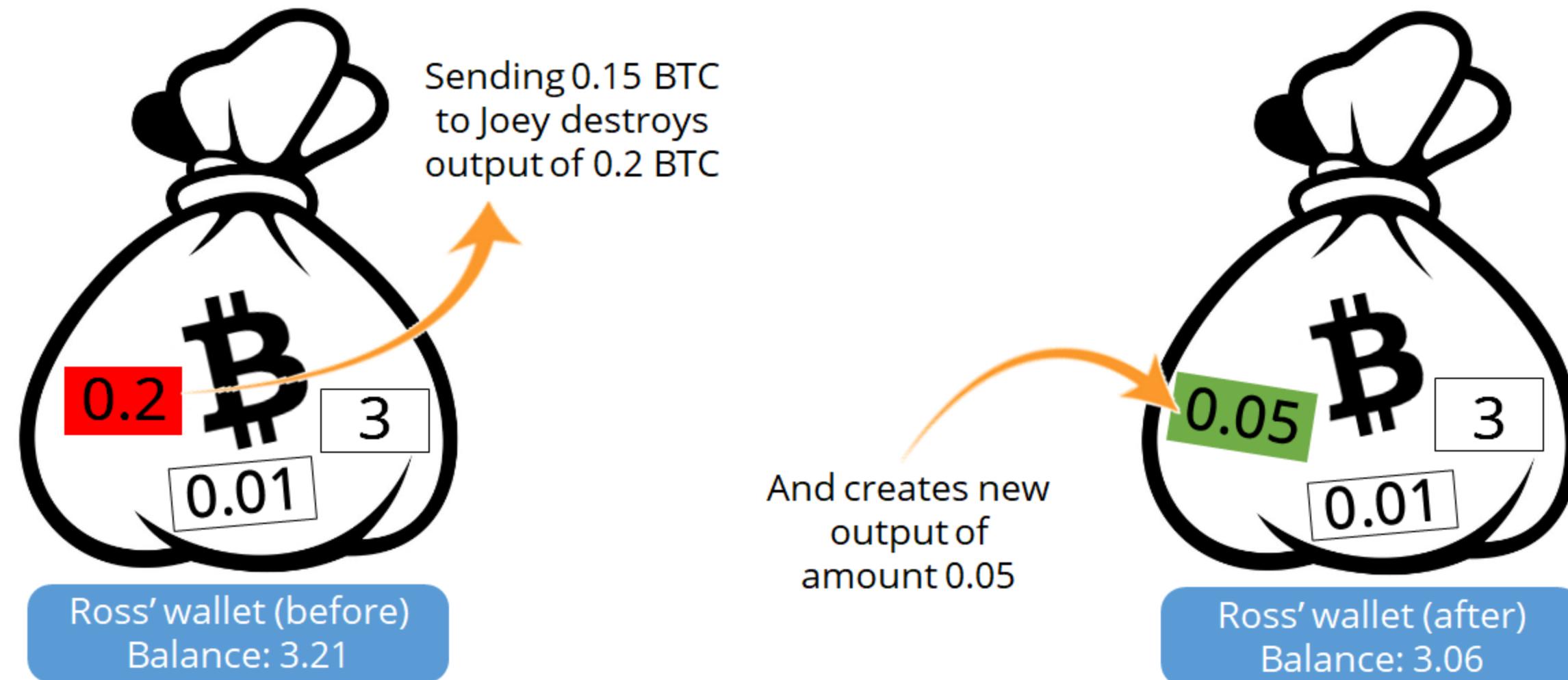


Example of a Blockchain wallet

# Bitcoin Transactions: UTXOs

UTXO or Unspent Transaction Output is the fundamental building block of Bitcoin.

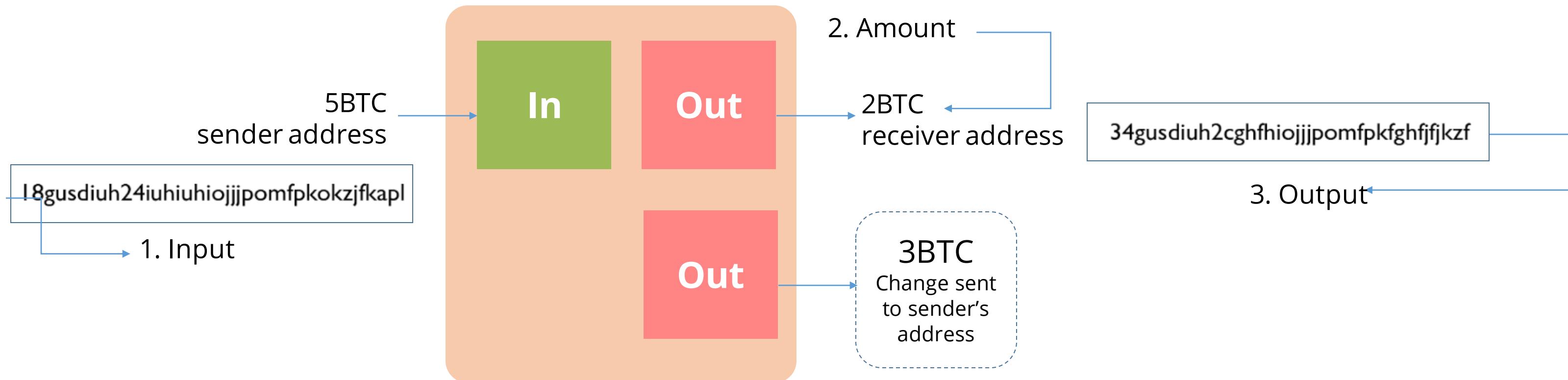
Ross wants to send 0.15 BTC to Joey



# Bitcoin Transaction Structure

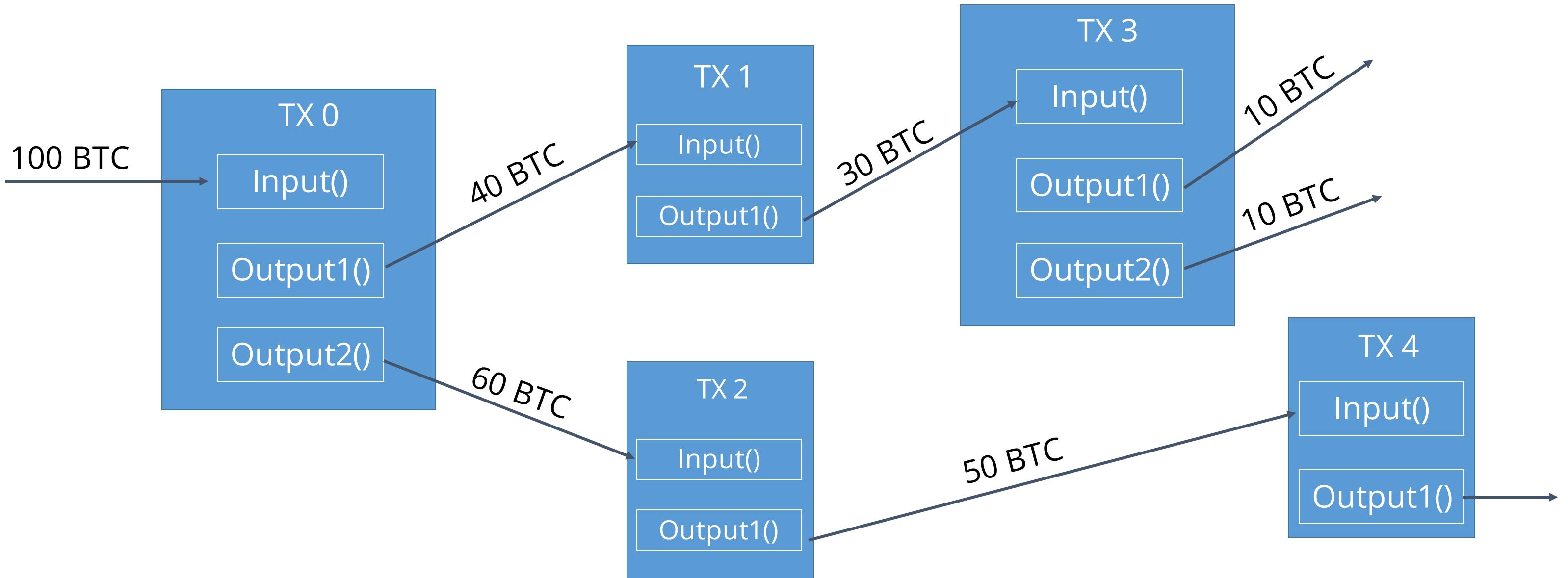
A Bitcoin transaction has three pieces of information:

- 1. Input:** This is a record of which Bitcoin address was used to send the Bitcoins to receiver in the first place
- 2. Amount:** This is the amount of Bitcoins that sender sends to the receiver
- 3. Output:** This is sent to sender's address as change



# Bitcoin Transactions and Input/Output

Transaction structure in Bitcoin reflects double-entry bookkeeping



# Example of a Bitcoin Transaction

**BLOCKCHAIN**   [WALLET](#)   [DATA](#)   [API](#)   [ABOUT](#)    [GET A FREE WALLET](#)

## Transaction View information about a bitcoin transaction

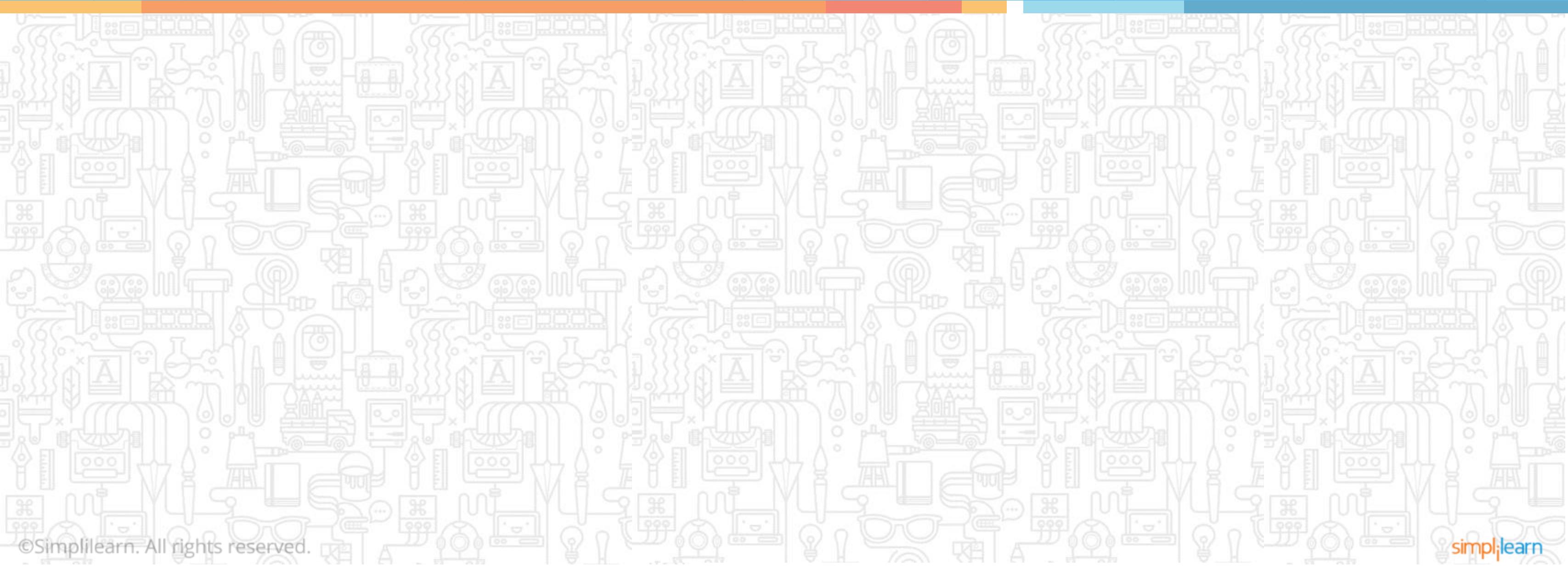
701b91591b36e8924349caec87993bd652e120426efceb0d204fb49db15eb4d

Input	Output	
<a href="#">1ADQ3q7po7THFSTsVJMzS1L4G69ZjsSMBQ (U)</a> (0.02446736 BTC - Output)	<a href="#">1HscL3Yuz3ahu7nfHkoA3CeA4GZZDU5Z5L</a> - (Unspent) Unable to decode output address - (Unspent)	0.02445736 BTC 0 BTC
		<a href="#">Unconfirmed Transaction!</a> <a href="#">0.02445736 BTC</a>

Summary		Inputs and Outputs	
Size	284 (bytes)	Total Input	0.02446736 BTC
Weight	1136	Total Output	0.02445736 BTC
Received Time	2018-10-30 06:34:12	Fees	0.00001 BTC
Visualize	<a href="#">View Tree Chart</a>	Fee per byte	3.521 sat/B
		Fee per weight unit	0.88 sat/WU
		Estimated BTC Transacted	0 BTC
		Scripts	<a href="#">Hide scripts &amp; coinbase</a>

# Bitcoin Blockchain

## Scripts in Bitcoins



## Bitcoin Script: Features

---

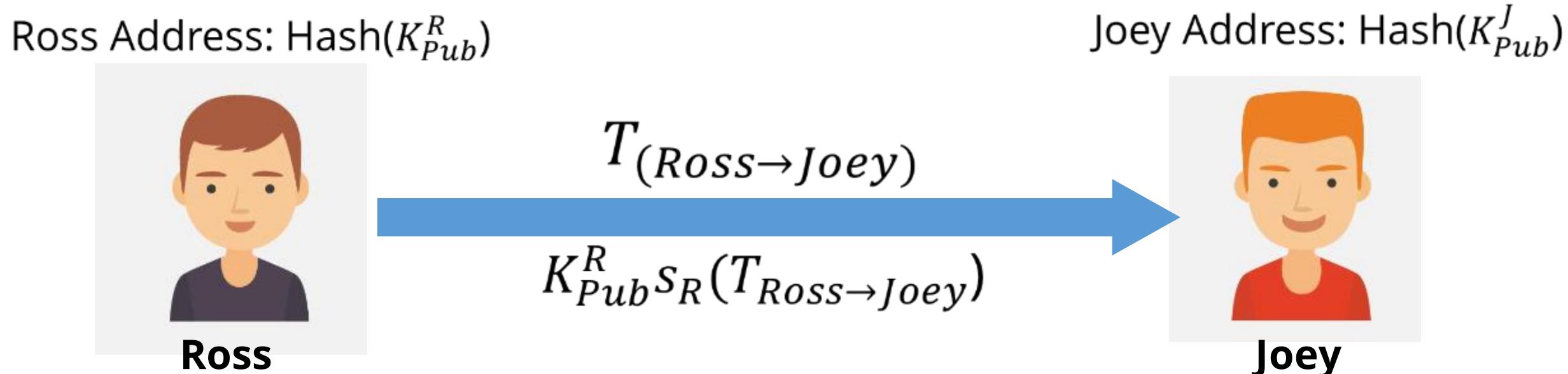
All Bitcoin transactions have scripts embedded into their inputs and outputs

- Bitcoin Script is a stack-based programming language like Forth
- A list of instructions are recorded with each transaction
- Operations in Bitcoin are composed of opcodes
- Bitcoin Script describes how the person can access the Bitcoins if they want to spend them
- Bitcoin Script is turing incomplete

# Bitcoin Script: An Example

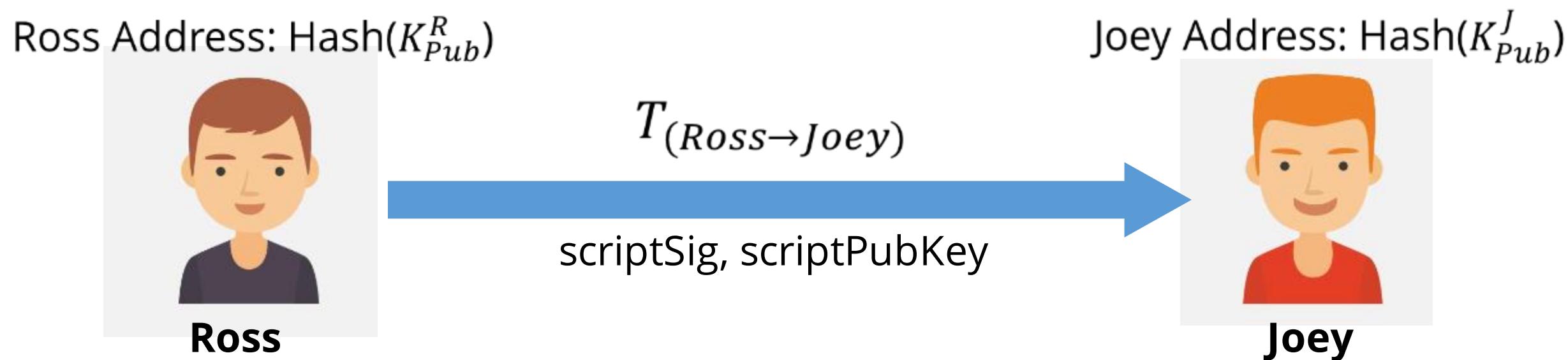
Scenario: Joey verifies the transaction is initiated by Ross

- 1 → Transaction from Ross is initiated ( $T_{(Ross \rightarrow Joey)}$ )
- 2 → Public Key of Ross  $K_{Pub}^R$  is sent along with the signature  $s_R(T_{Ross \rightarrow Joey})$  which Joey can verify



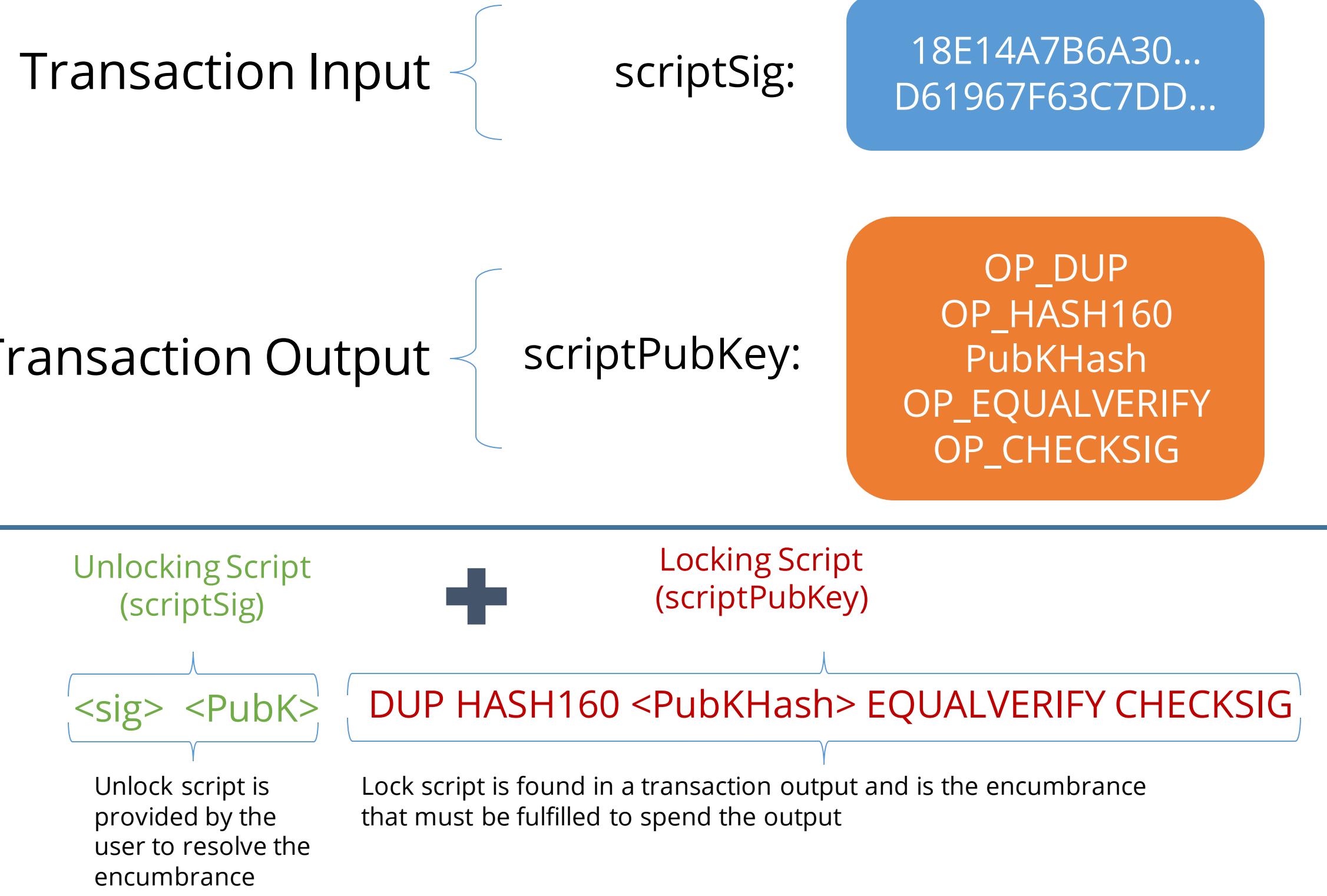
## Bitcoin Script: An Example (Contd..)

Bitcoin transfers scripts instead of signature and the public key



Joey can spend the Bitcoins only if both the scripts  
return true after execution

# Bitcoin Script Construction

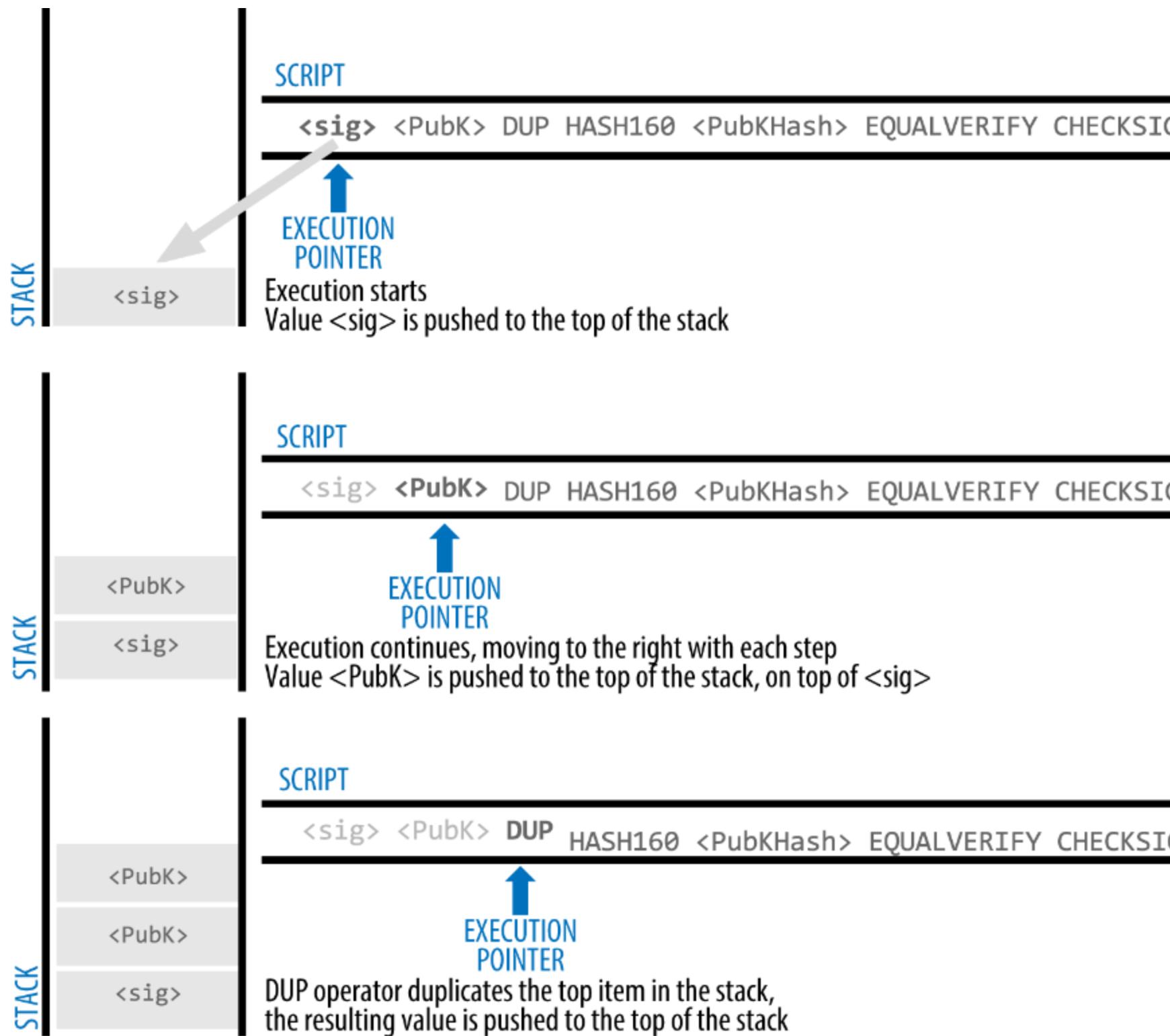


# Bitcoin Script Instructions

---

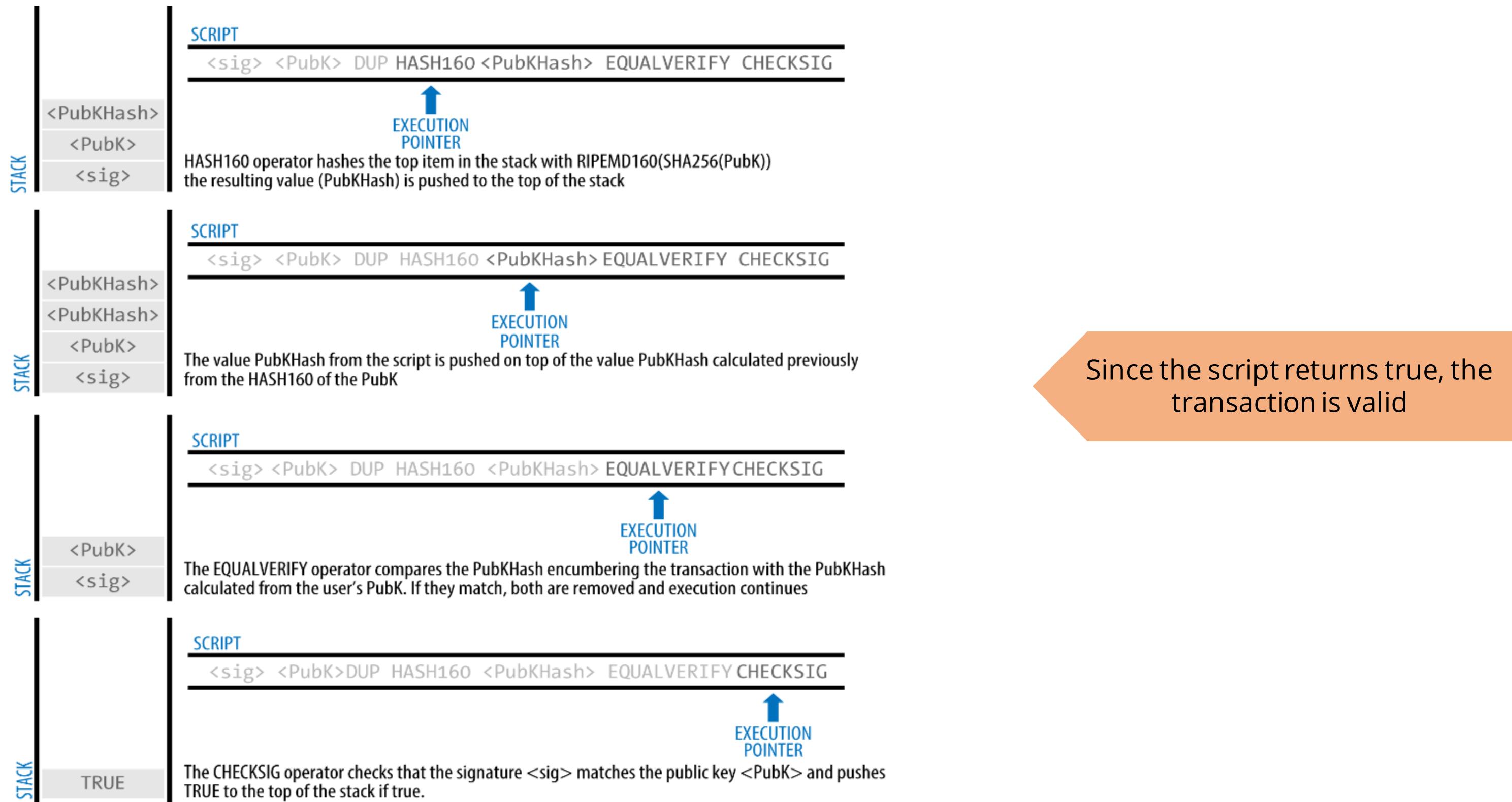
- Bitcoin scripting language is small
- It consists of 256 opcodes out of which 15 are disabled and 75 are reserved
- Each instruction is given one byte
- Opcodes have the following: constants, flow control, stack, bitwise logic, arithmetic, crypto, locktime, pseudo-words, reserved words

# Evaluating Bitcoin Transaction Script



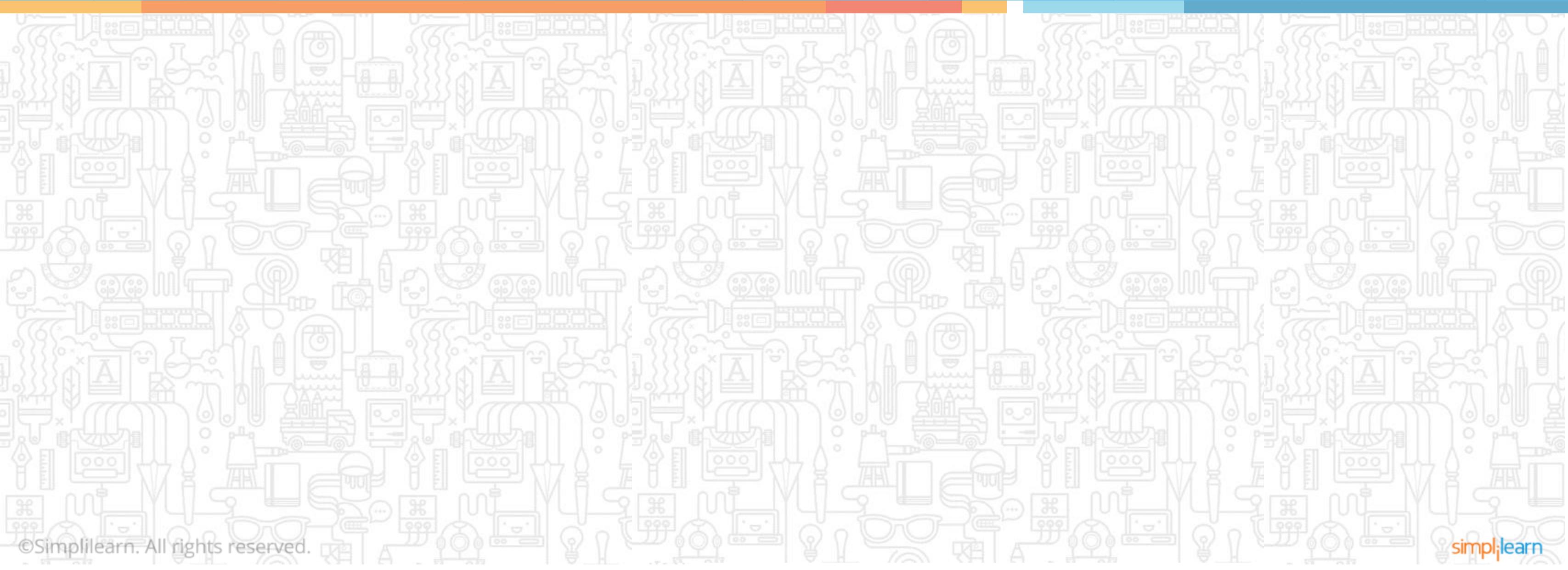
A step-by-step execution of the combined script which will prove this is a valid transaction

# Evaluating Bitcoin Transaction Script(Contd..)



# Bitcoin Blockchain

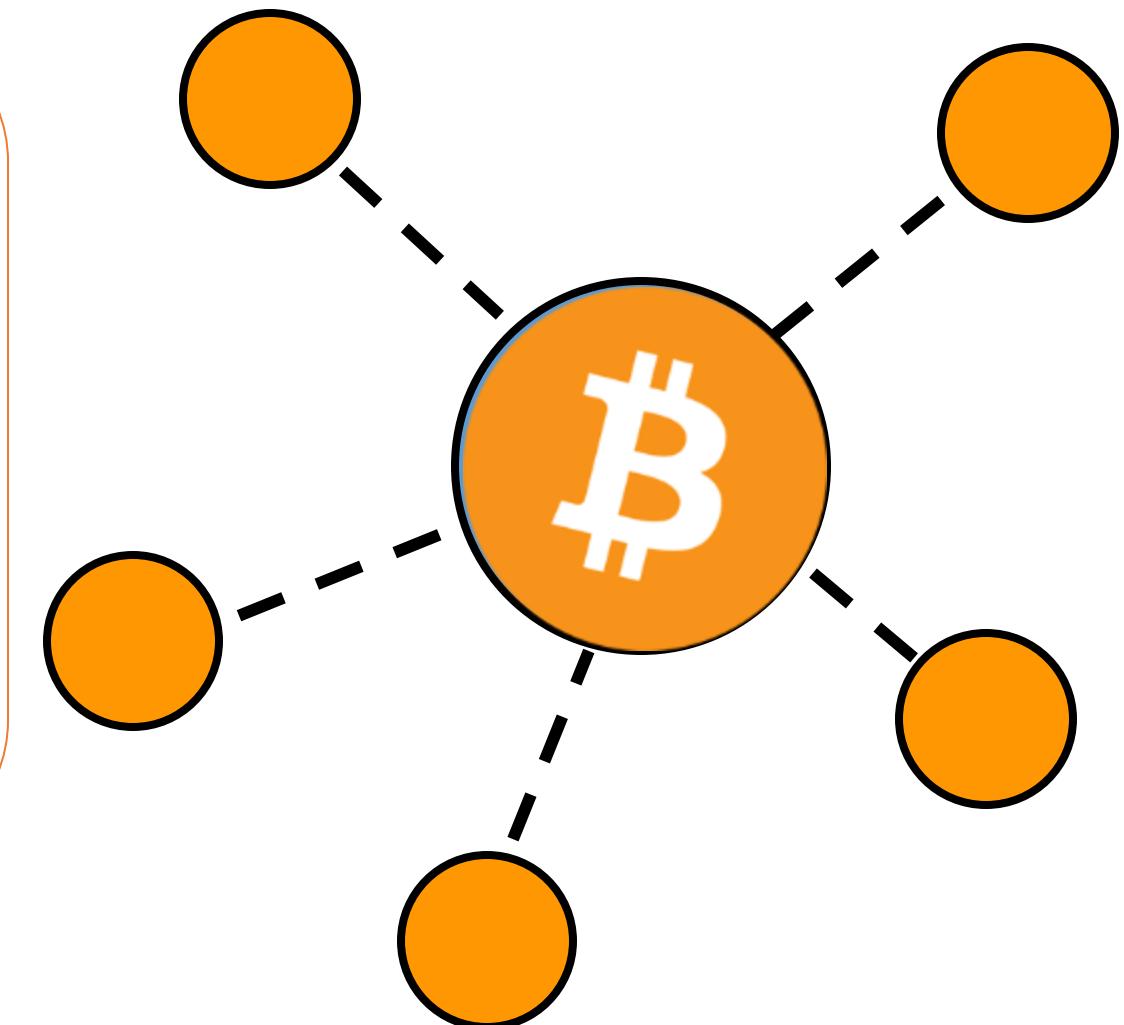
## Bitcoin Nodes and Network



# Bitcoin Network

---

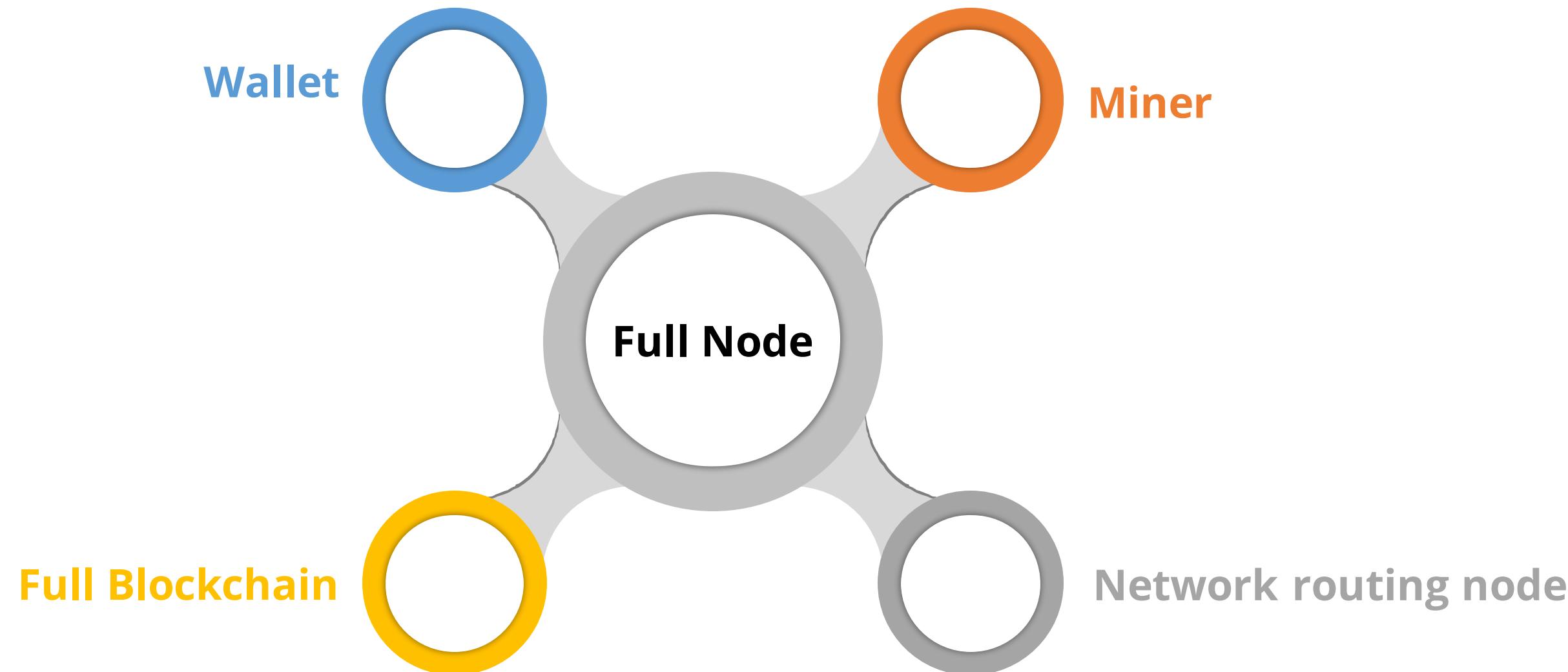
- Bitcoin is an ad-hoc network with random topology
- The protocol runs on TCP 8333
- All the nodes are treated equally in Bitcoin network
- New nodes can join the network anytime, and the nonresponsive nodes are removed after 3 hours



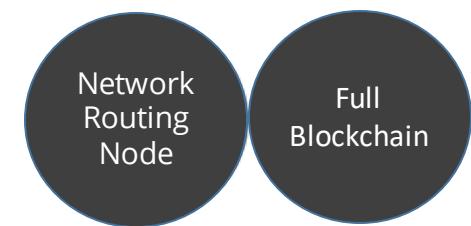
# Bitcoin Network Node

Nodes in the Bitcoin network may take on different roles depending on the functionality they support

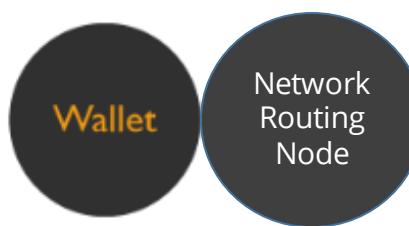
**A full Bitcoin node is a collection of four functions:**



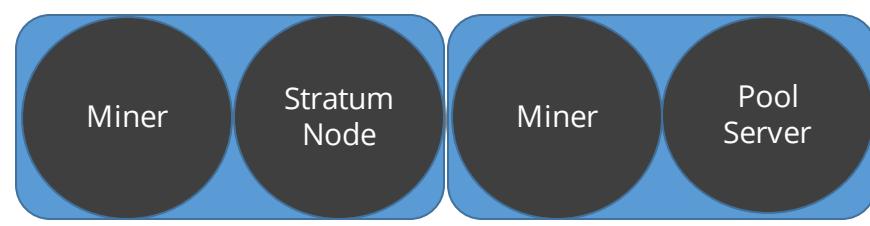
# Types of Bitcoin Nodes



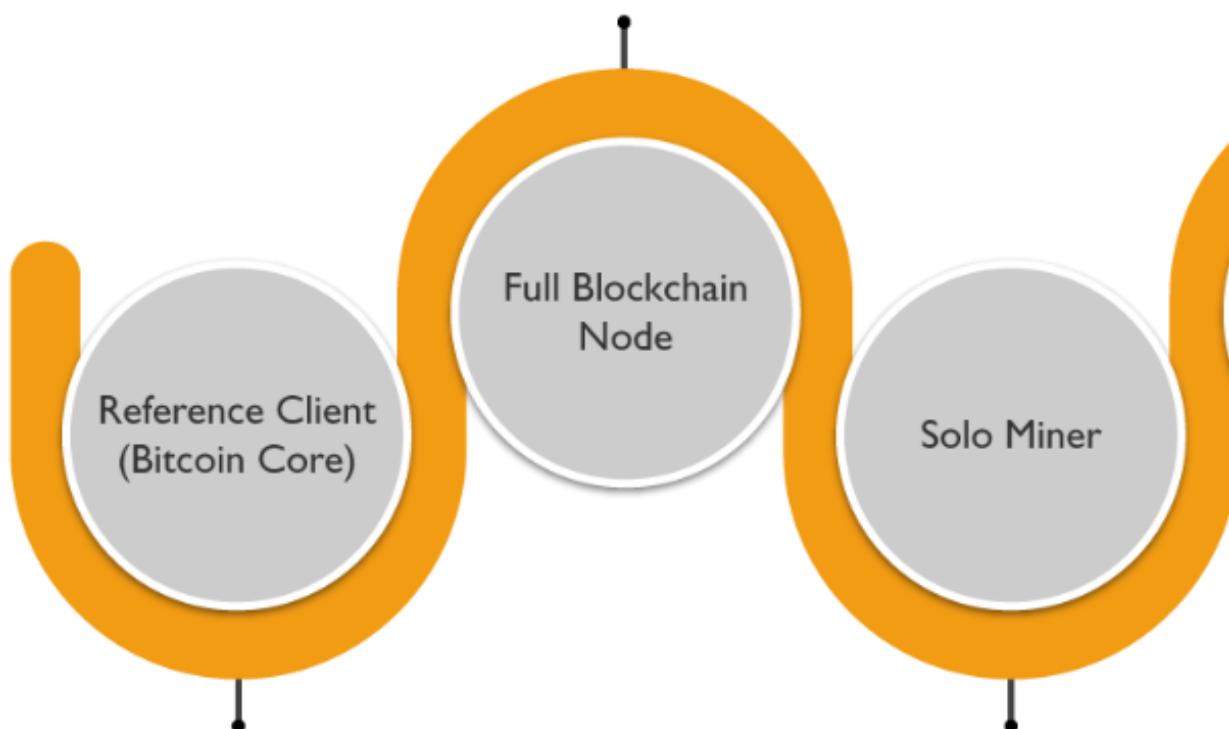
Contains a full Blockchain database and network routing node on the Bitcoin p2p network



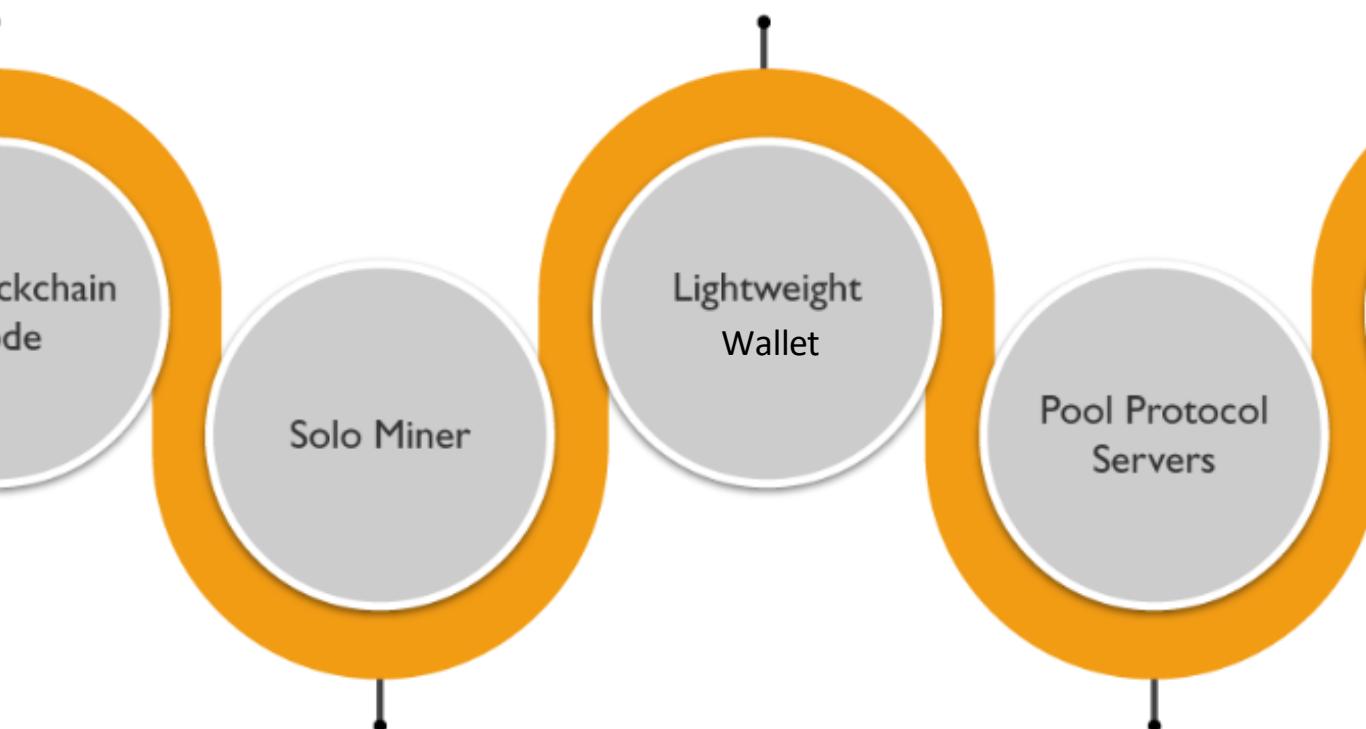
Contains a wallet and a network node on the Bitcoin p2p protocol without a Blockchain



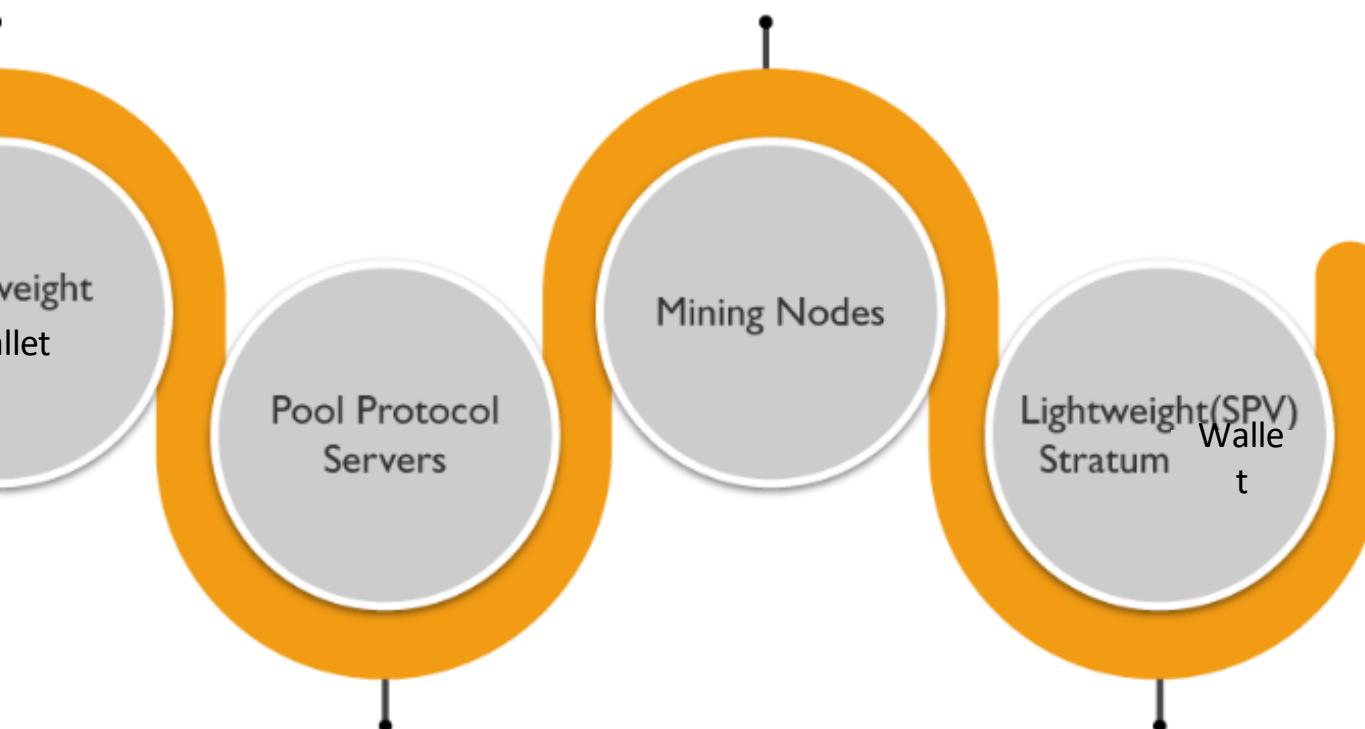
Contains a mining function without a Blockchain but with a Stratum protocol node or other pool mining protocol nodes



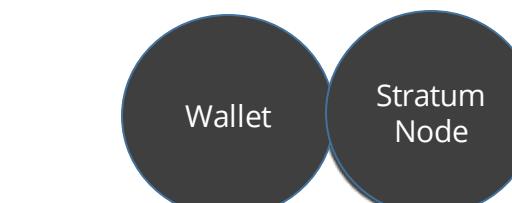
Contains a wallet, miner, full Blockchain database, and network routing node on the Bitcoin p2p network



Contains a mining function with a full copy of the Blockchain and a Bitcoin p2p network routing node



Gateway routers connecting the Bitcoin p2p network to nodes running other protocols such as pool mining or Stratum nodes

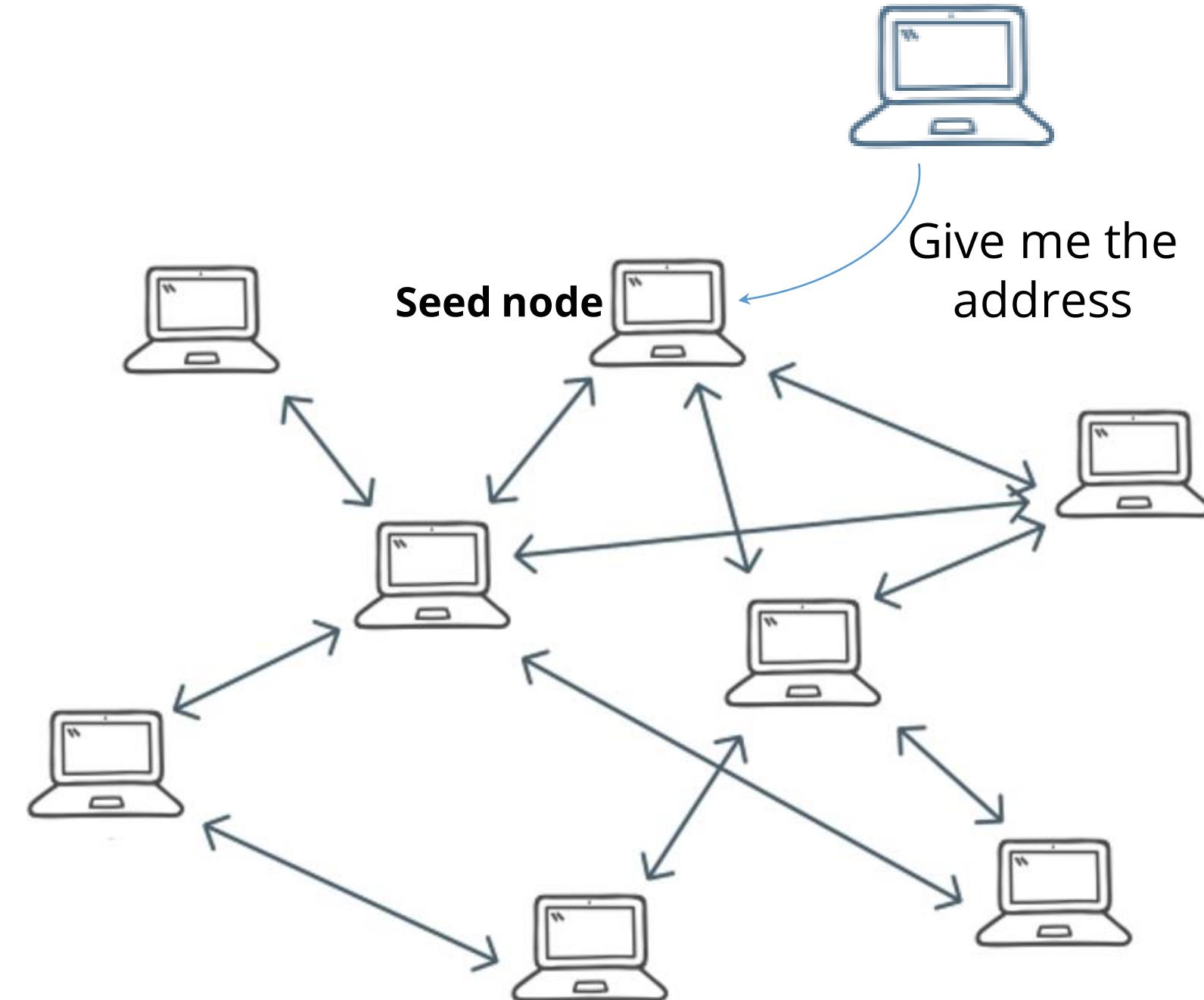


Contains a wallet and a network node on the Stratum protocol without a Blockchain

# Joining Bitcoin Network

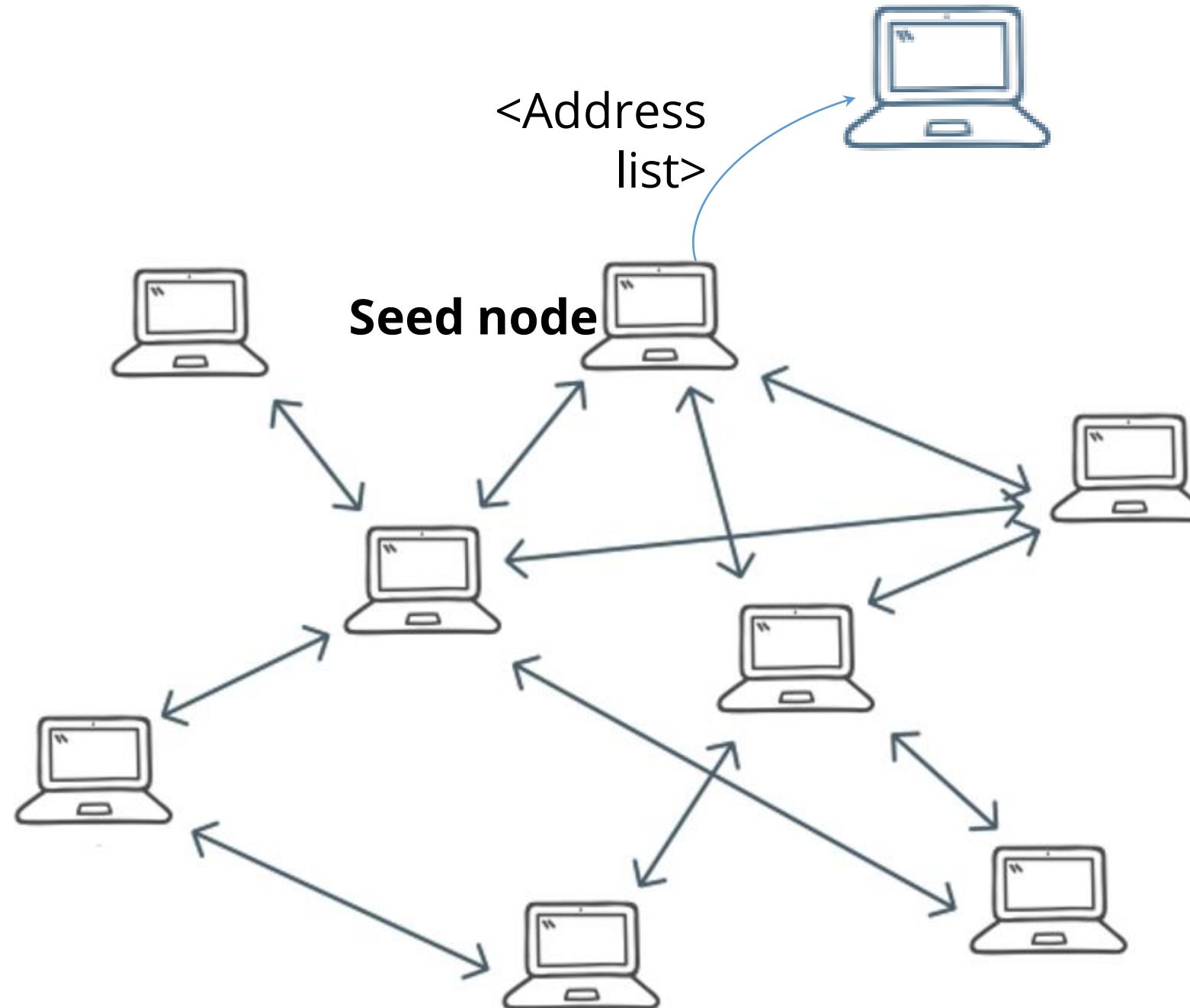
There are certain special nodes in the Bitcoin network called seed nodes which have the list of addresses

Joining nodes ask for the list of addresses from the seed node



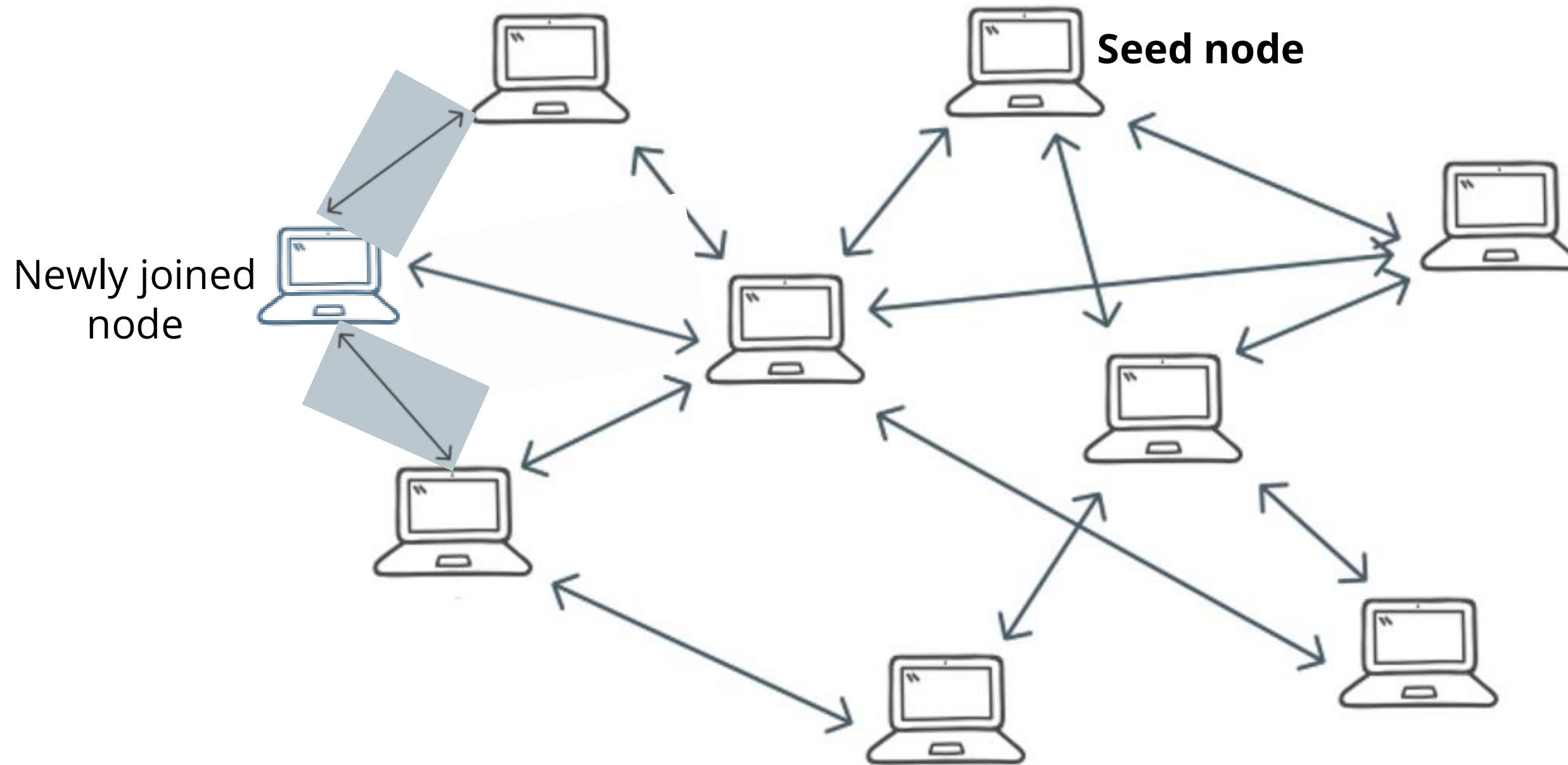
# Joining Bitcoin Network

In response to the node's request to join the network, the seed node sends them the address list



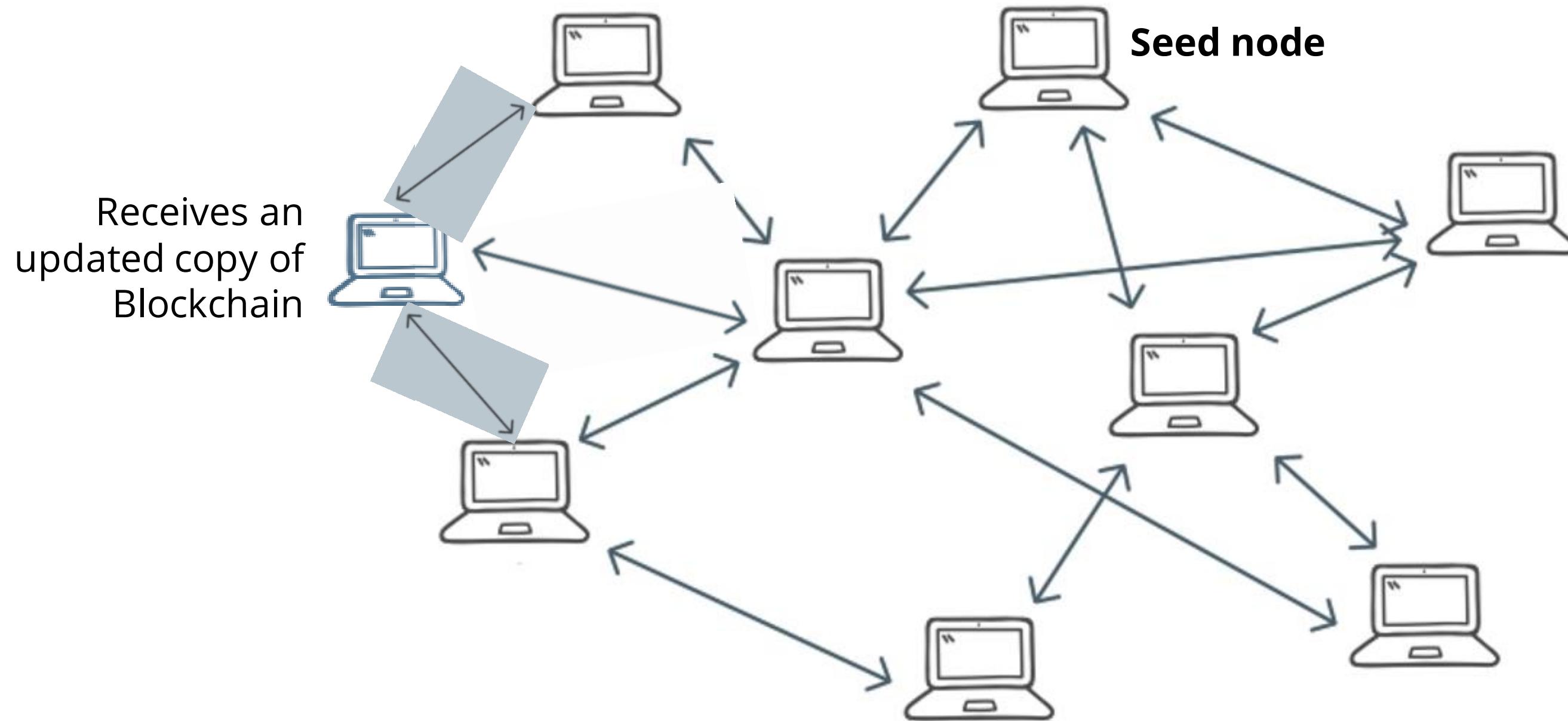
# Joining Bitcoin Network

The joining node picks up an address from the list and joins the network



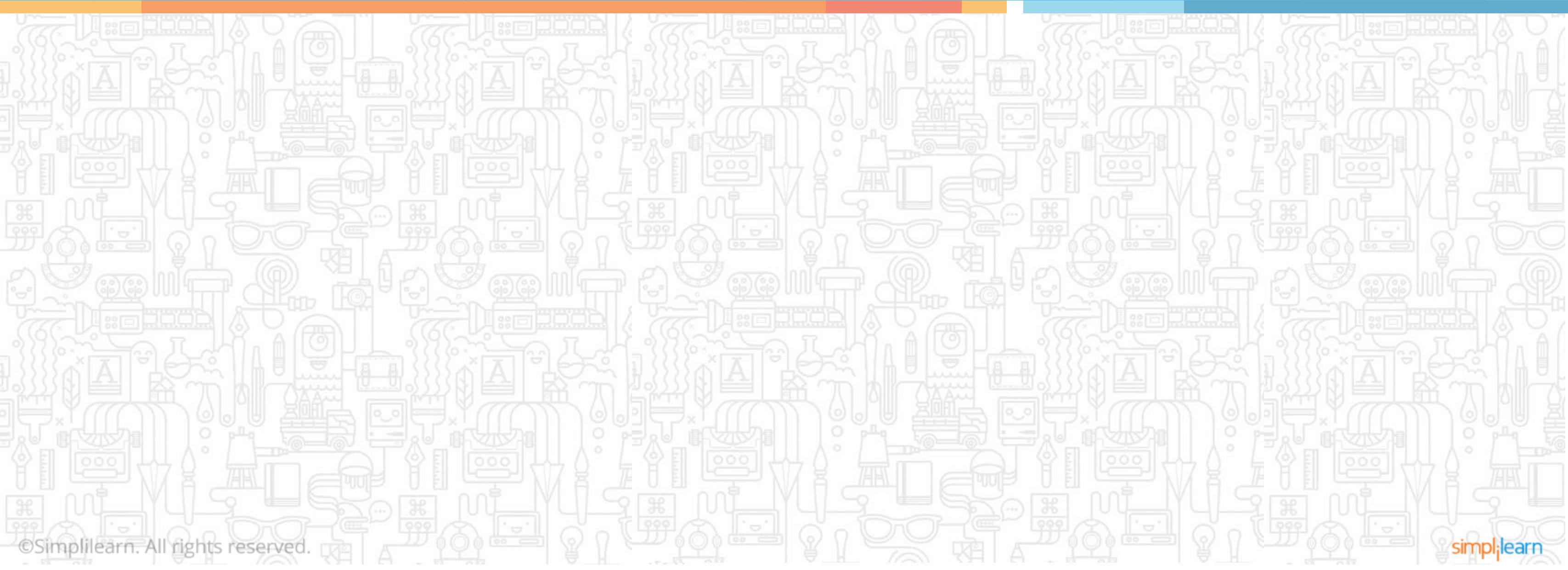
# Joining Bitcoin Network

The newly joined nodes then get the most recent Blockchain from its peers



# Bitcoin Blockchain

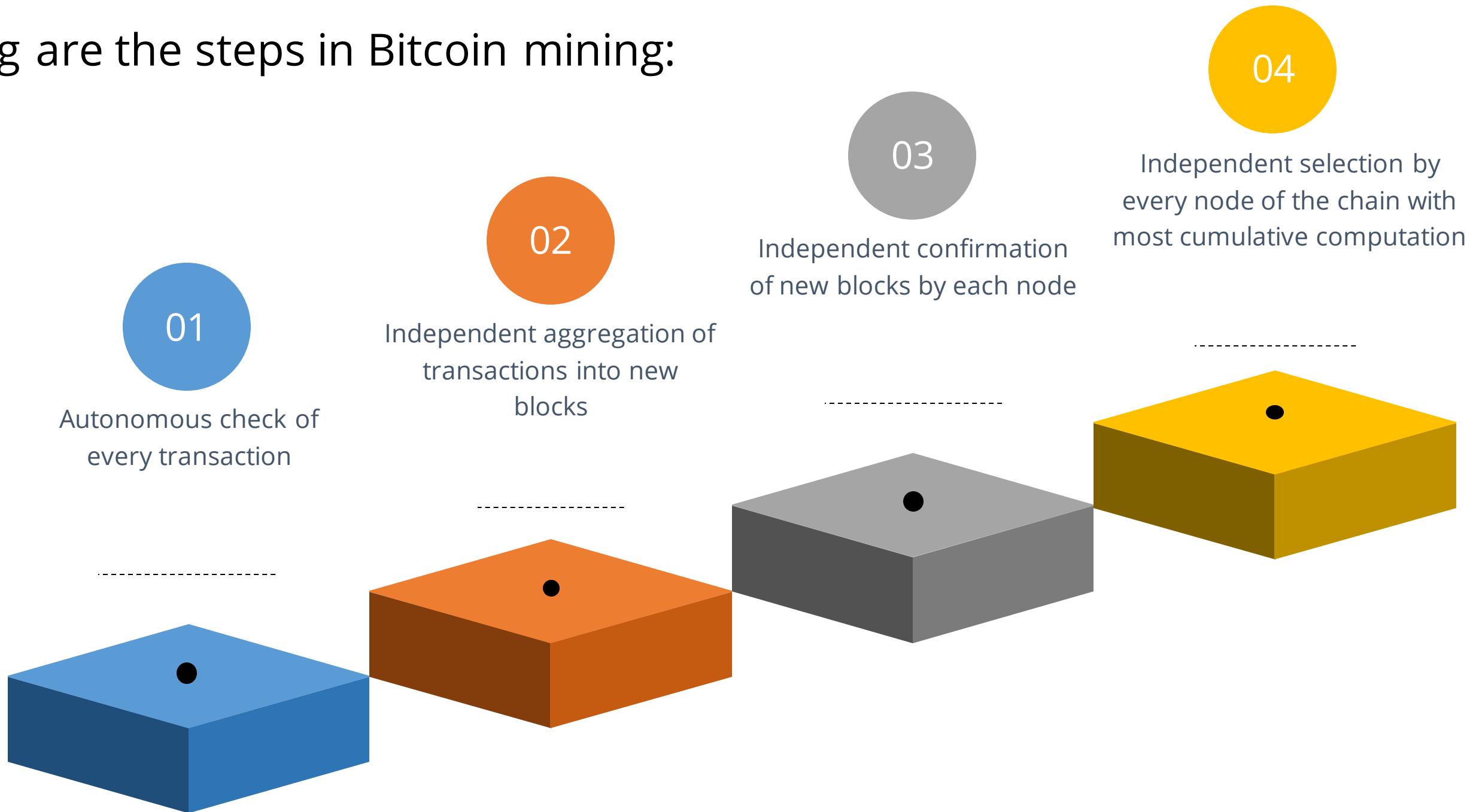
## Mining in Bitcoin Blockchain



# Bitcoin Mining Process

Bitcoins are created by the process of mining

Following are the steps in Bitcoin mining:



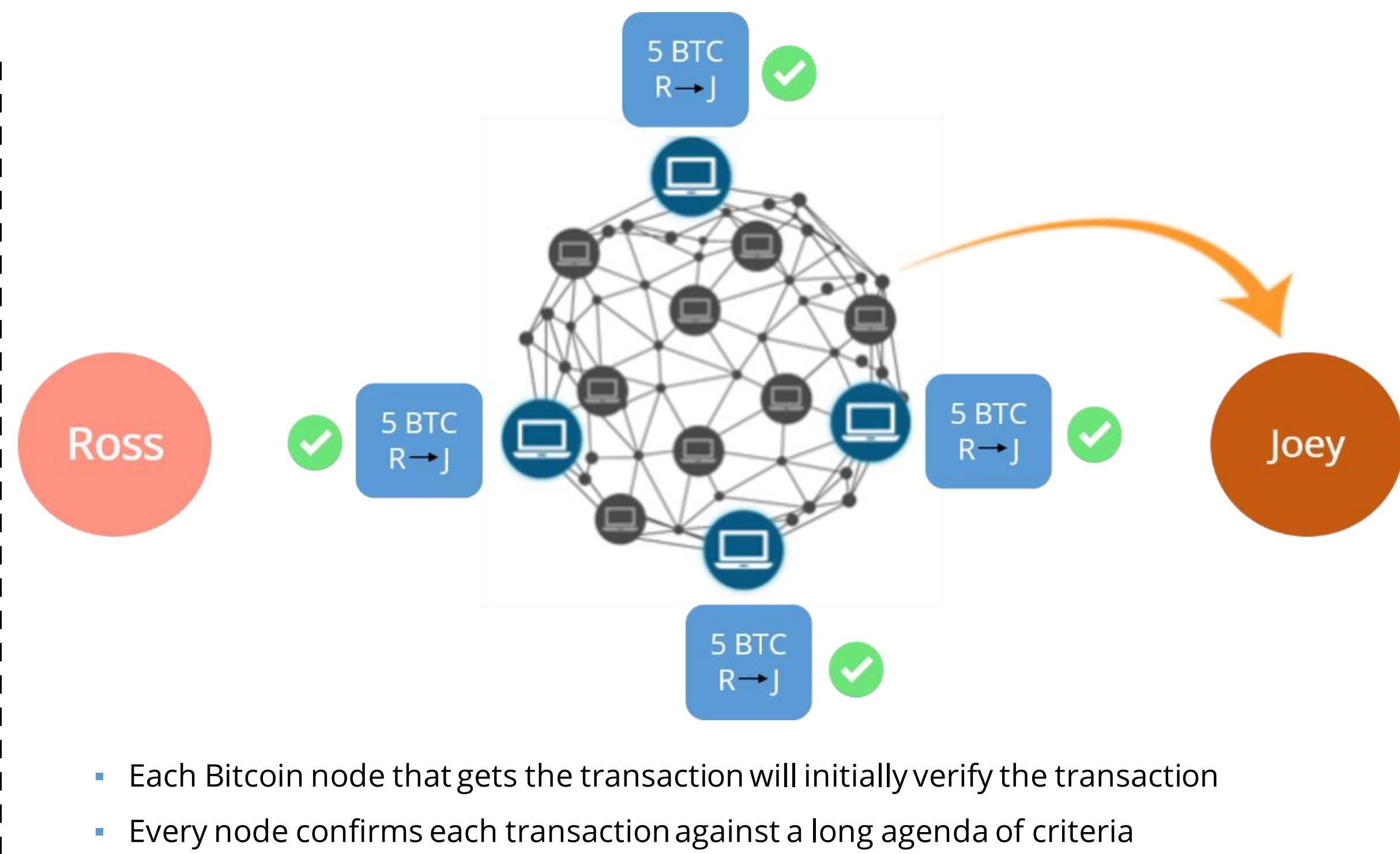
# Bitcoin Mining Process

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



# Bitcoin Mining Process

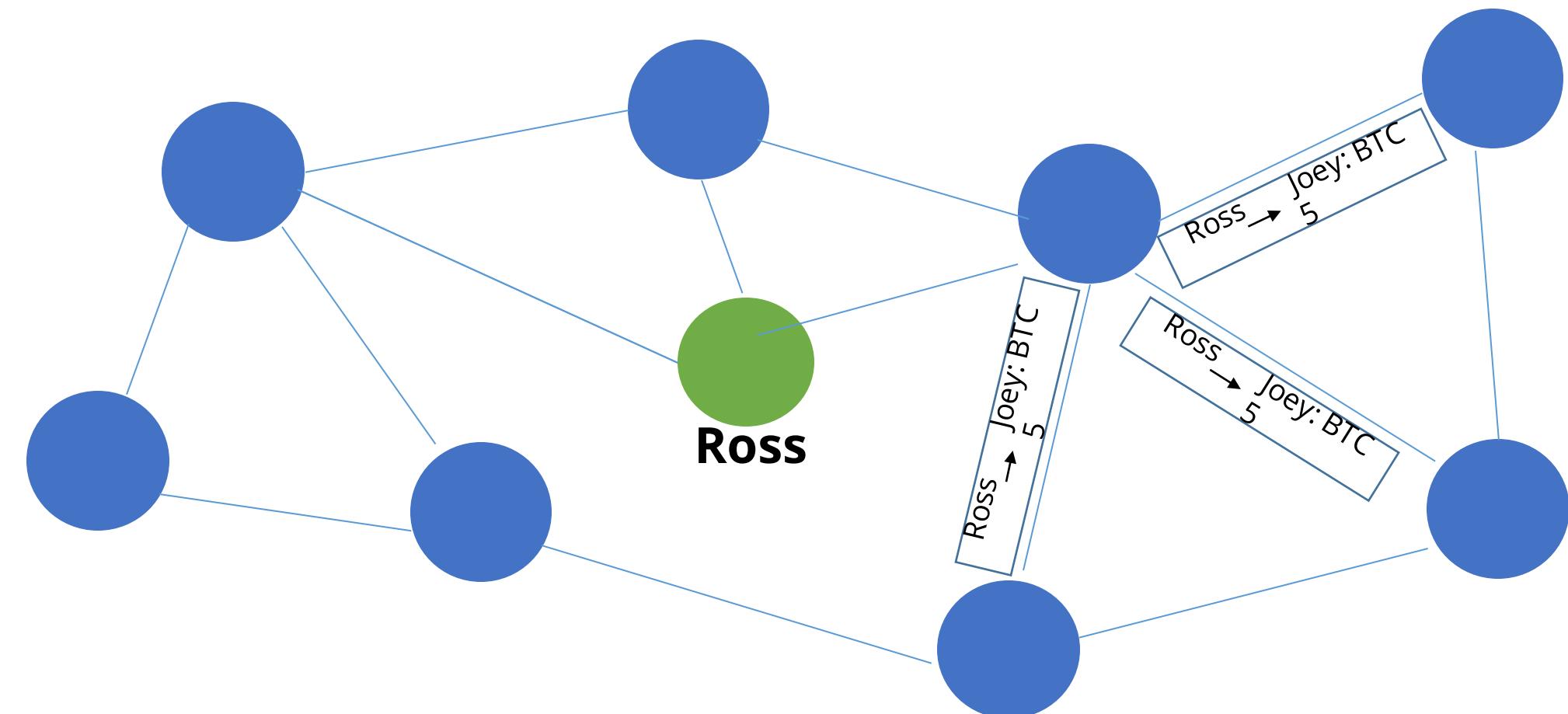
Transactions that can further be relayed:

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



# Bitcoin Mining Process

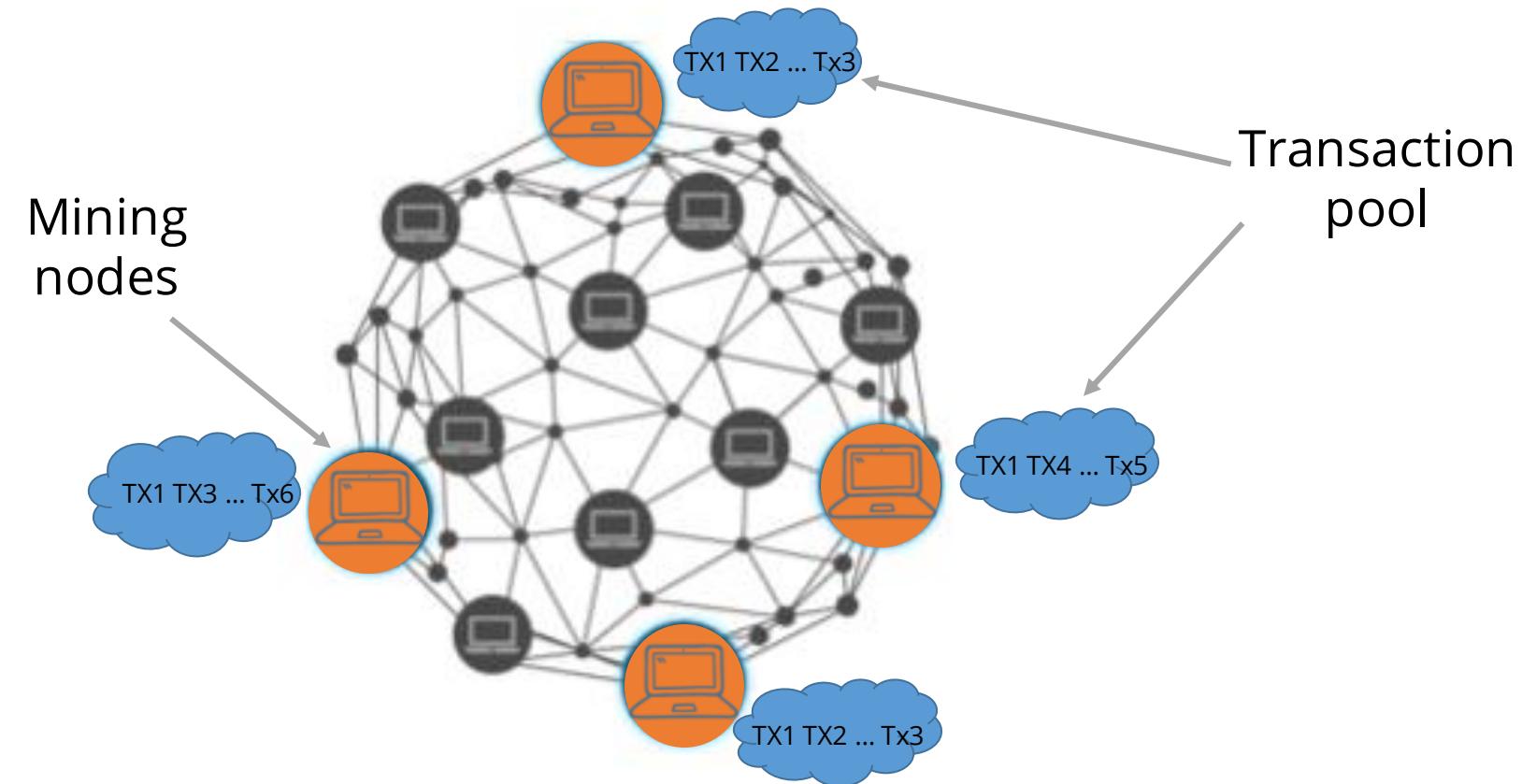
Miner nodes aggregate the transactions in a candidate block

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

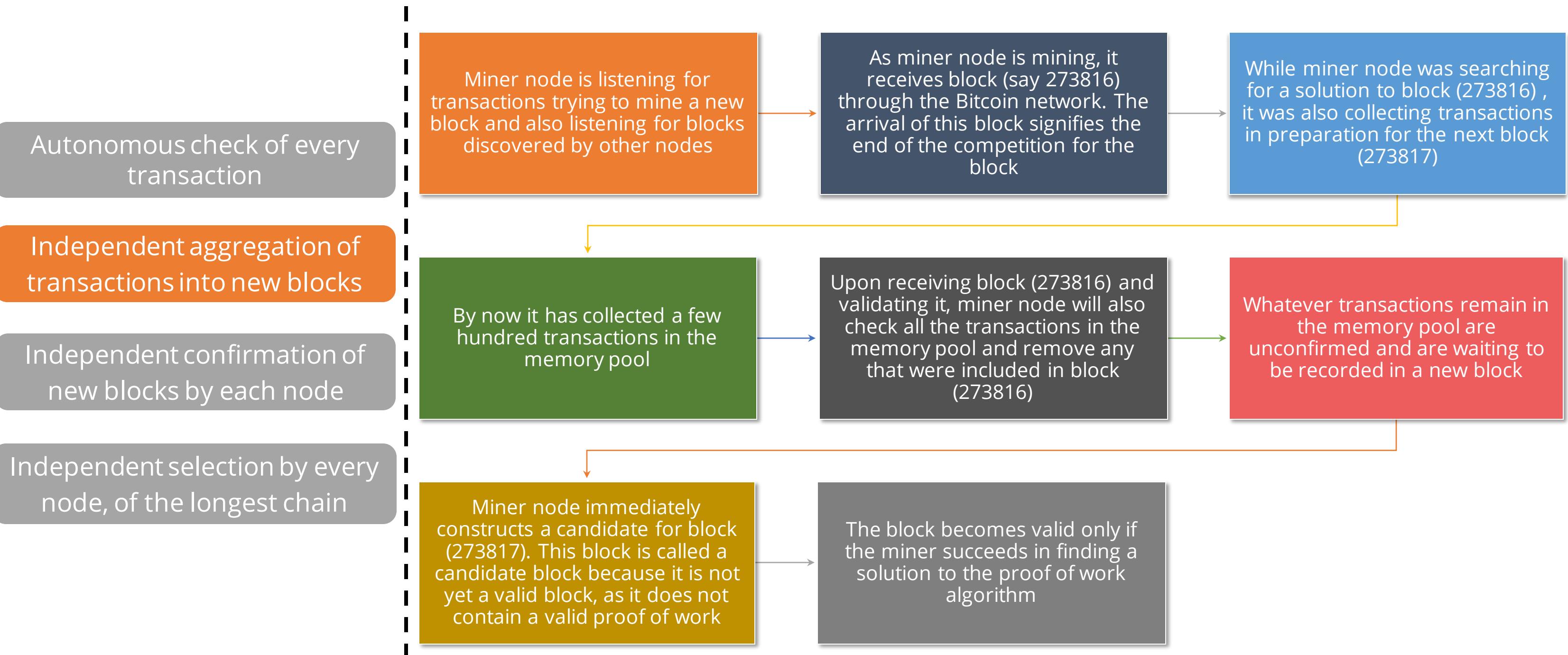
Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



- Each node fabricates a pool of valid, however, unconfirmed transactions known as the transaction pool, memory pool or mempool
- Unlike other nodes, miner node will then aggregate these transactions into a candidate block

# Bitcoin Mining Process



# Bitcoin Mining Process

Miner nodes start solving the mining puzzle:

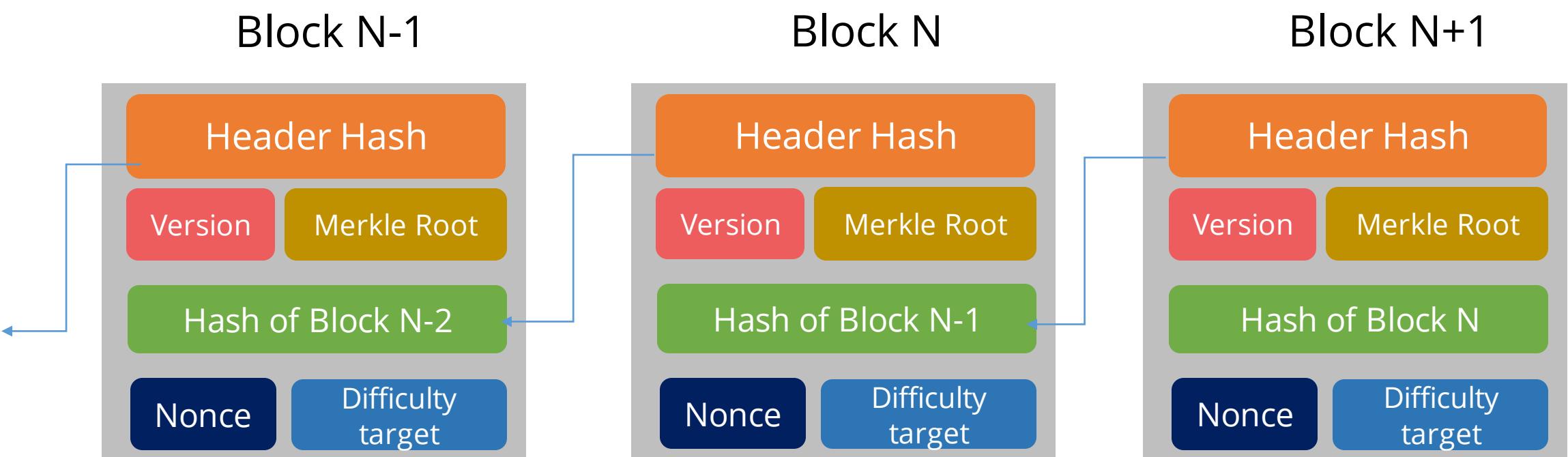
Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain

- Mining node constructs the block header using the following fields: version, Merkle root, previous block hash, difficulty target, nonce
- The goal is now to find a value for the nonce that results in a block header hash that is less than the difficulty target



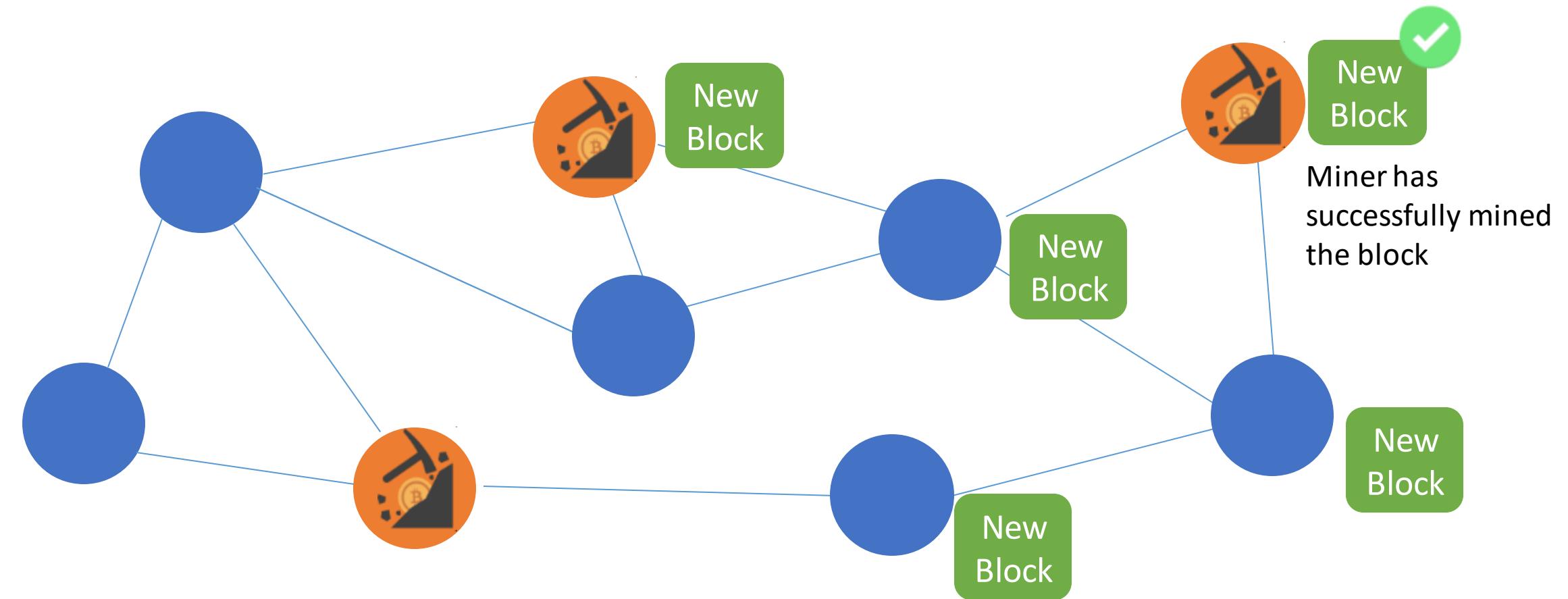
# Bitcoin Mining Process

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



# Bitcoin Mining Process

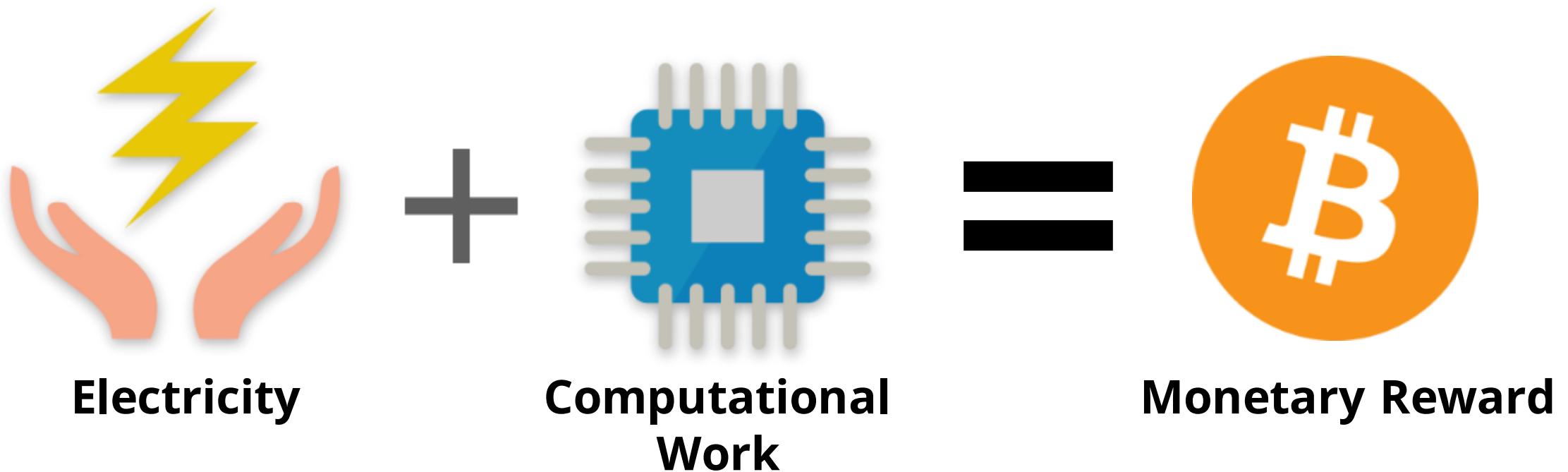
Since miners use their valuable resources to validate the block, they are given a **monetary award**

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



- Successful miner is rewarded with newly created Bitcoins
- The block reward is halved every 210,000 blocks or every four years

# Bitcoin Mining Process

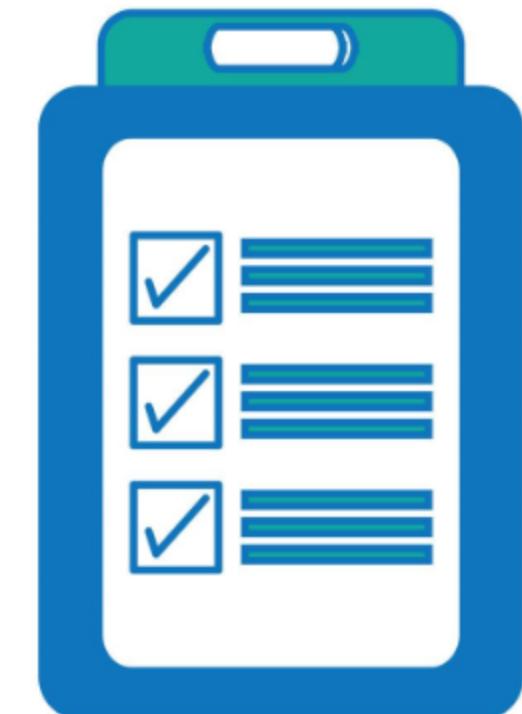
Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain

- In Bitcoin's consensus mechanism, each new block is validated independently by every node on the network which ensures that only valid blocks are propagated on the network
- Nodes validate the block by checking it against a long list of criteria that must be met



# Bitcoin Mining Process

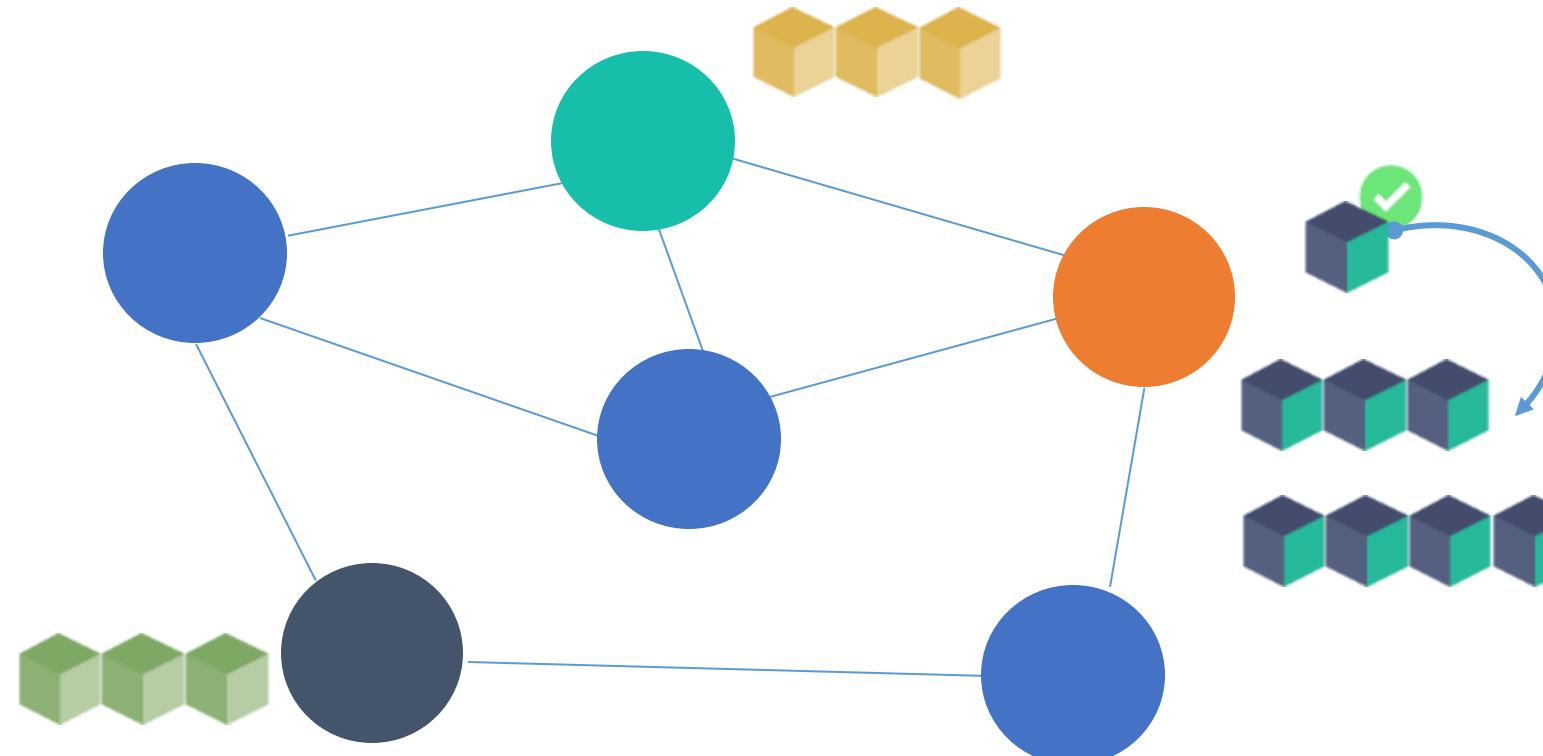
## Assembling and selecting chains of blocks:

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain



In the network shown above, once the node(in orange color) validates the block, it assembles the chain by connecting the block in the existing Blockchain

# Bitcoin Mining Process

Bitcoins are thus assembled in the longest chain

Autonomous check of every transaction

Independent aggregation of transactions into new blocks

Independent confirmation of new blocks by each node

Independent selection by every node, of the longest chain

- By choosing the greatest difficulty chain, all nodes in the long run accomplish a wide consensus
- Brief errors between chains are settled in the long run as more proof of work is included broadening one of the conceivable chains
- Mining nodes vote with their mining power by picking which chain to extend by mining the next block

# Key Takeaways



You are now able to:

- ➊ Explain the term Bitcoin and ways to procure them
- ➋ Set up a Bitcoin wallet
- ➌ Identify where to spend Bitcoin
- ➍ Describe the transaction structure of Bitcoin Blockchain
- ➎ Interpret the role of Bitcoin Scripts in transactions
- ➏ Explain the network and nodes in Bitcoin
- ➐ Identify the steps involved in Bitcoin mining



## Knowledge Check



Knowledge  
Check

1

## How many Bitcoins will ever be created in the network?

- a. Unlimited
- b. 77340109
- c. 21000000
- d. 210000000



Knowledge  
Check

1

## How many Bitcoins will ever be created in the network?

- a. Unlimited
- b. 77340109
- c. 21000000
- d. 210000000



The correct answer is C

**Bitcoin supply is controlled to have a value, hence the Bitcoin protocol is hardbound to have a maximum supply of 21000000 BTC.**

## What is a Bitcoin address?

- a. Hash of a public key
- b. Hash of private key
- c. Hash of public key and private key together
- d. A random hash code



## What is a Bitcoin address?

- a. Hash of a public key
- b. Hash of private key
- c. Hash of public key and private key together
- d. A random hash code



The correct answer is **a**

**A Bitcoin address is a 160-bit hash of the public portion of a public/private ECDSA key pair.**

## Which of these is not a part of block Metadata?

- a. Version
- b. Merkle Root
- c. Nonce
- d. Block Header



## Which of these is not a part of block Metadata?

- a. Version
- b. Merkle Root
- c. Nonce
- d. Block Header



The correct answer is **d**

**Block header is the resultant hash of the combination of version, merkle root, nonce, previous block hash.**

## How often does a block reward halve?

- a. After 100,000 blocks or eight months
- b. Every 210,000 blocks or every four years
- c. It never halves
- d. Every 25,000 blocks or six months



## How often does a block reward halve?

- a. After 100,000 blocks or eight months
- b. Every 210,000 blocks or every four years
- c. It never halves
- d. Every 25,000 blocks or six months



The correct answer is

**b**

**Reward schedule was set in the Bitcoin protocol at the time of genesis which says that the reward started at 50 BTC halves every 210,000 blocks.**

## Which port is used by Bitcoin clients to connect to other Bitcoin nodes?

- a. TCP 8080
- b. TCP 8200
- c. TCP 8333
- d. TCP 992



## Which port is used by Bitcoin clients to connect to other Bitcoin nodes?

- a. TCP 8080
- b. TCP 8200
- c. TCP 8333
- d. TCP 992



The correct answer is C

**By default, Bitcoin clients connect to other bitcoin nodes on TCP port 8333.**

# Lesson-End Project

Duration: 10 mins

## Generate a Bitcoin Wallet

### Problem Statement:

You are an employee of Simplilearn and have been asked to create a Bitcoin wallet for the company. You have to perform the following actions:

- Generate a single wallet
- Generate a paper wallet with a passphrase
- Check wallet details

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

# Lesson-End Project

Duration: 10 mins

## Work with BitPay Wallet

### Problem Statement:

You are an employee of Simplilearn and have been asked to create a BitPay wallet account for the company and use it to test Bitcoins. Download the BitPay wallet to get test Bitcoins in your wallet.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 3: Ethereum



# Learning Objectives

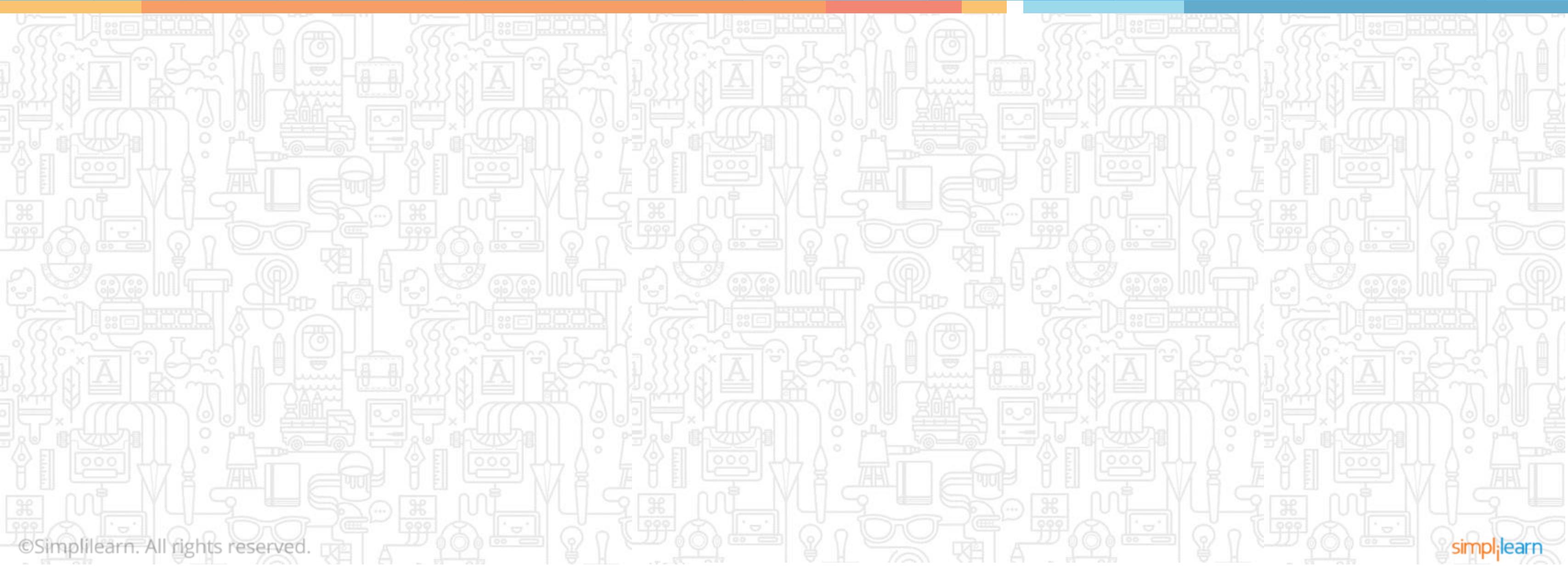


By the end of this lesson, you will be able to:

- ➊ Describe Ethereum and its concepts
- ➋ Install Geth and Ganache on your computer
- ➌ Use MetaMask and connect it with Ganache
- ➍ Install and use Mist Wallet

# Ethereum

## Ethereum and Its Concepts



# Ethereum

---



ethereum

Ethereum is a decentralized platform that runs the smart contracts without any downtime, fraud, and third-party interference and also has its own cryptocurrency called Ether.

# Features of Ethereum

---



**Secured**



**Community**



**Corporate Friendly**



**Assets**



**Fast**



**Uncensored**

## Gas in Ethereum

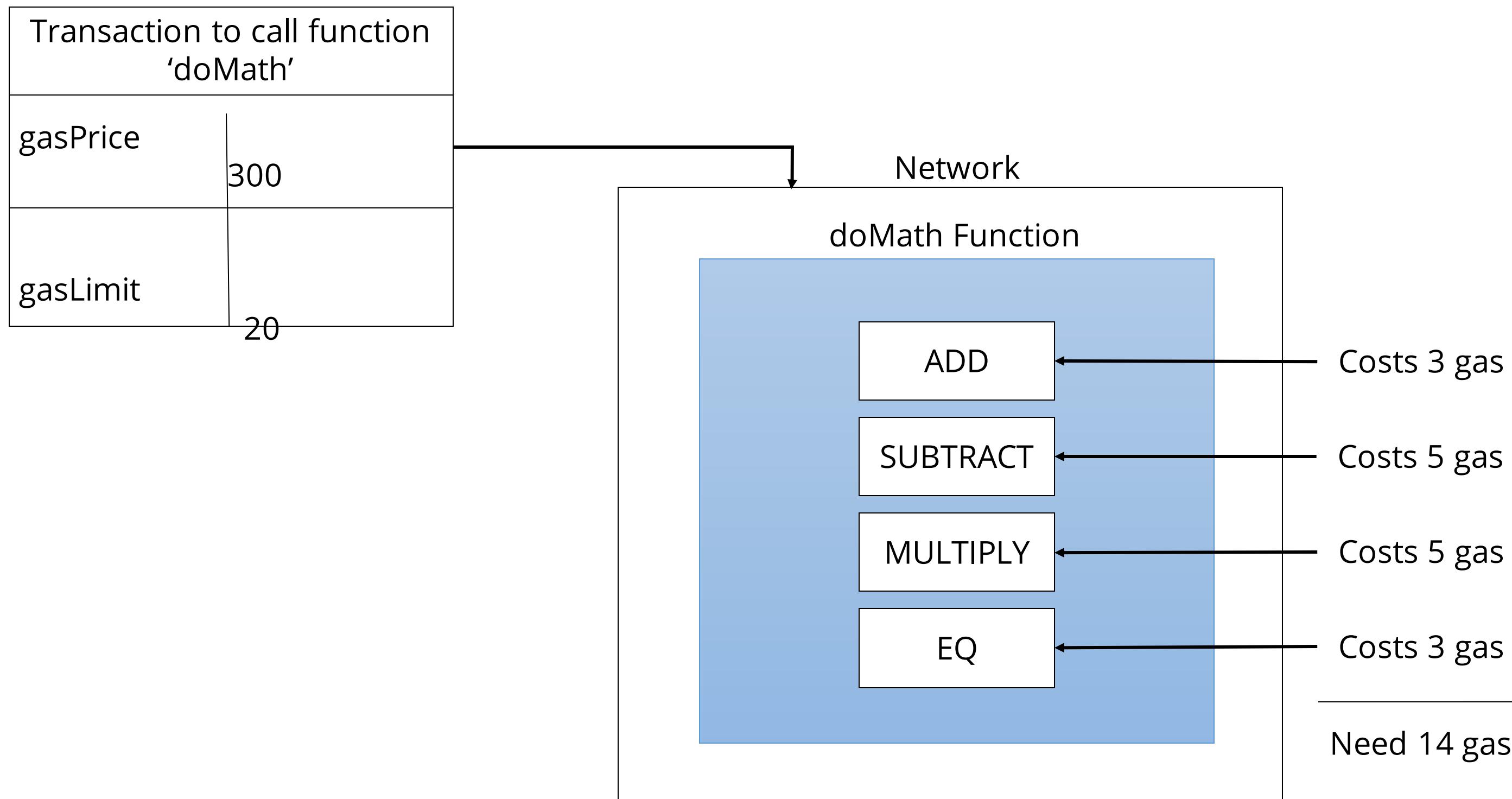
---



Gas is the fundamental block of Ethereum ecosystem that is paid for every operation performed on Ethereum Blockchain.

Its price is expressed in ether, and the miner decides whether to refuse the transaction process or not based on the expected gas price.

# Gas in Ethereum



# Ether

Ether is a necessary element needed for operating the Ethereum platform.

- An incentive that the client of the platform pays to execute requested operations
- It ensures that the developers write error-free codes, because unnecessary codes will increase the cost



**ETHER**

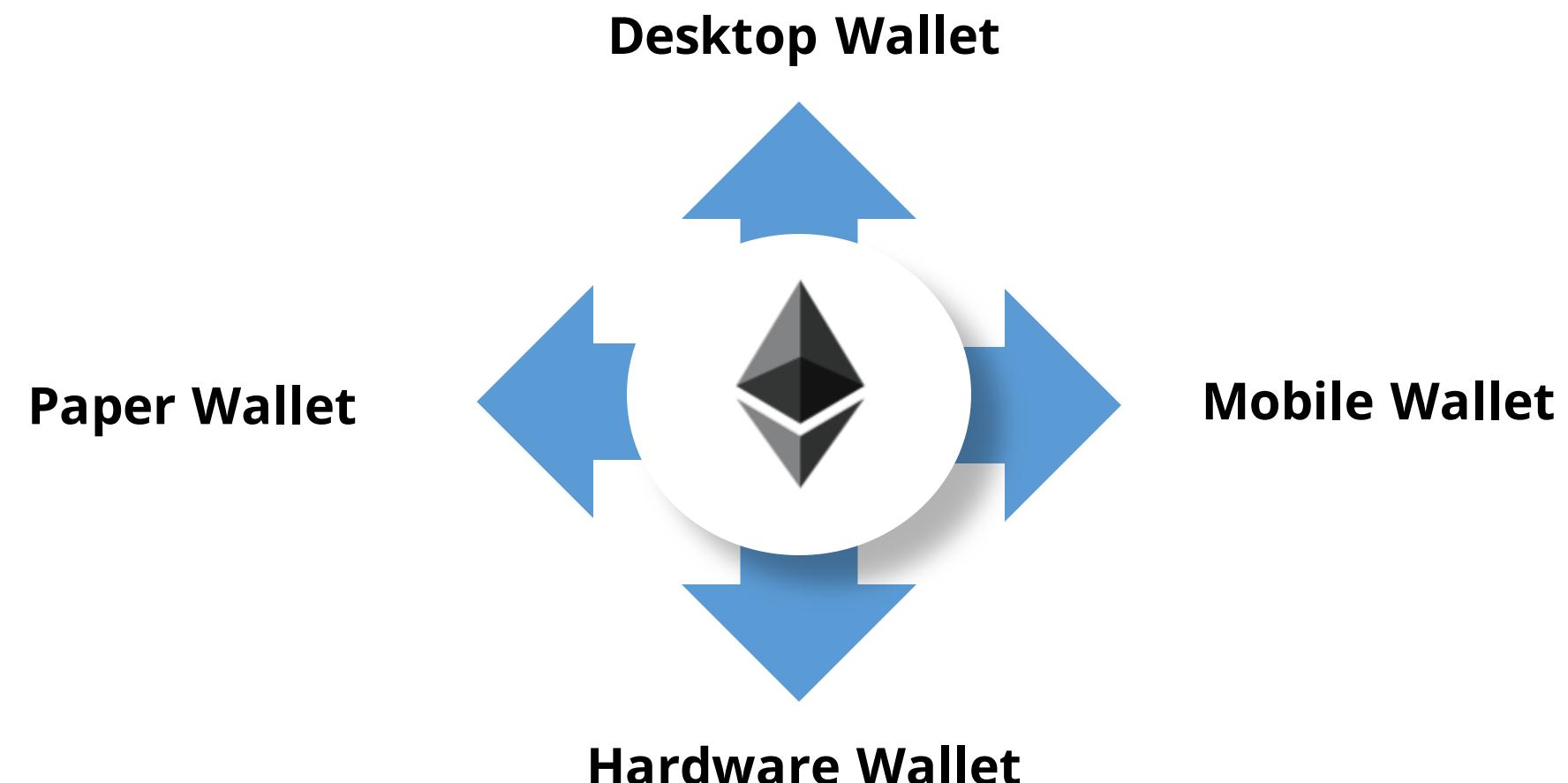
The crypto-fuel  
for the Ethereum  
network

# Bitcoin vs. Ether

Bitcoin	Ether
Uses secure hash algorithm (SHA-256)	Uses ethash algorithm
Used for purchasing goods and services	Used for making decentralized apps
\$8500 is the price at the moment	\$520 is the price at the moment
A currency created to compete against the gold and fiat currencies	A token created for facilitating smart contracts
Transaction speed is measured in minutes	Transaction speed is measured in seconds

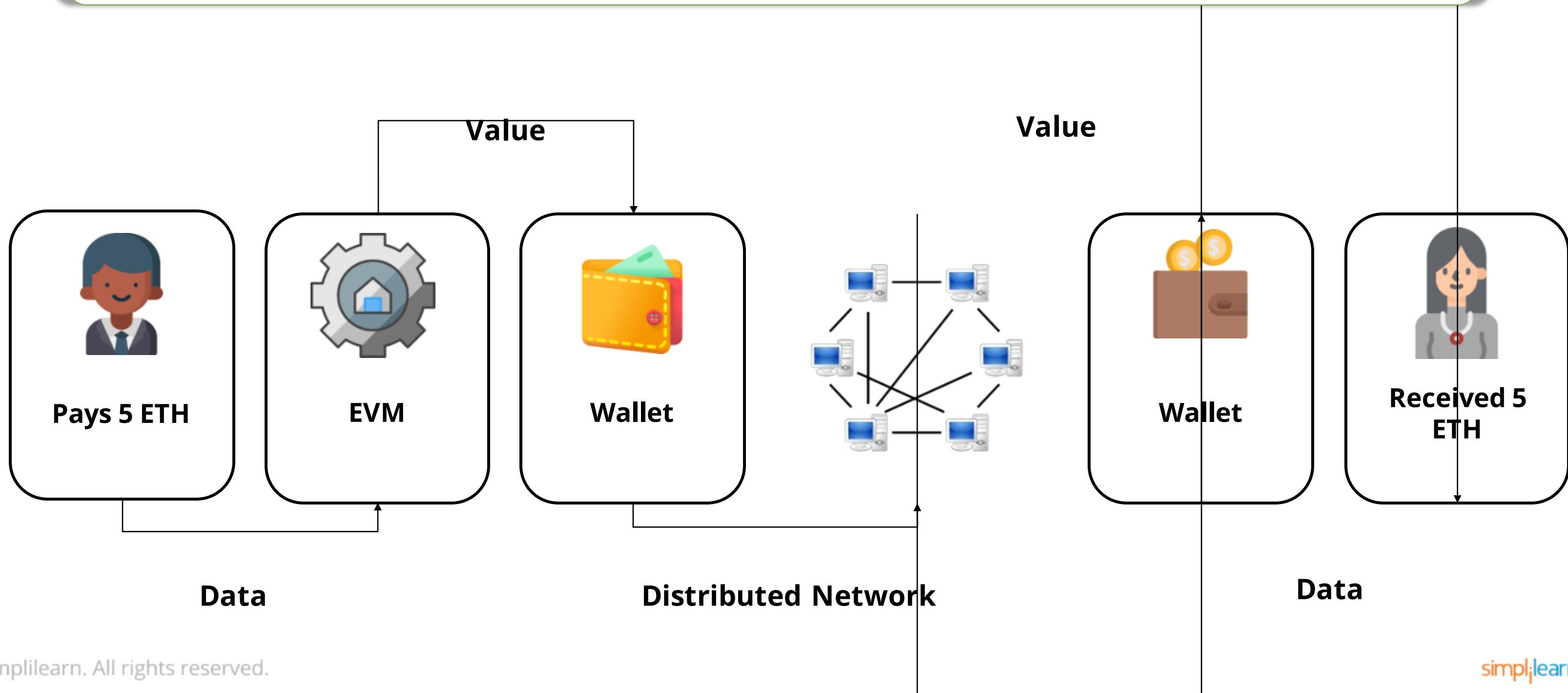
# Ethereum Wallets

The place to securely store the ether is called wallets.



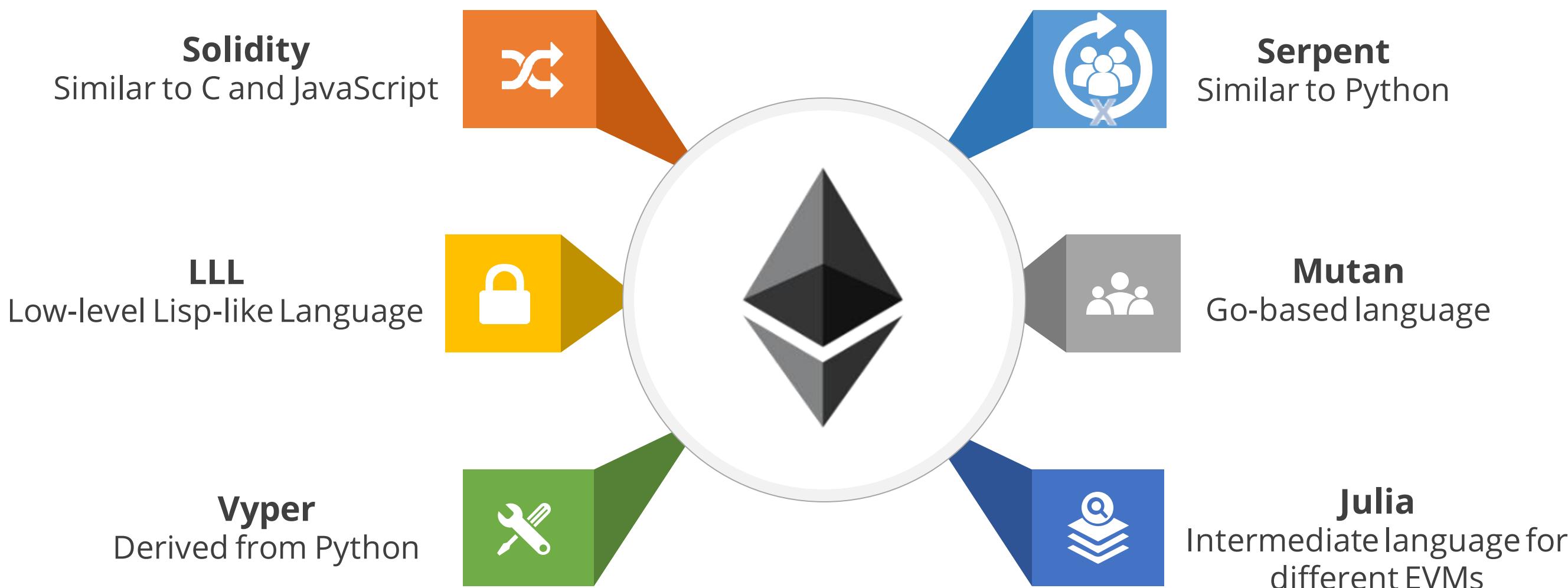
# Ethereum Virtual Machine

Ethereum Virtual Machine (EVM) is an engine which executes translation code. Smart contracts are compiled into bytecode which an EVM can read and execute.



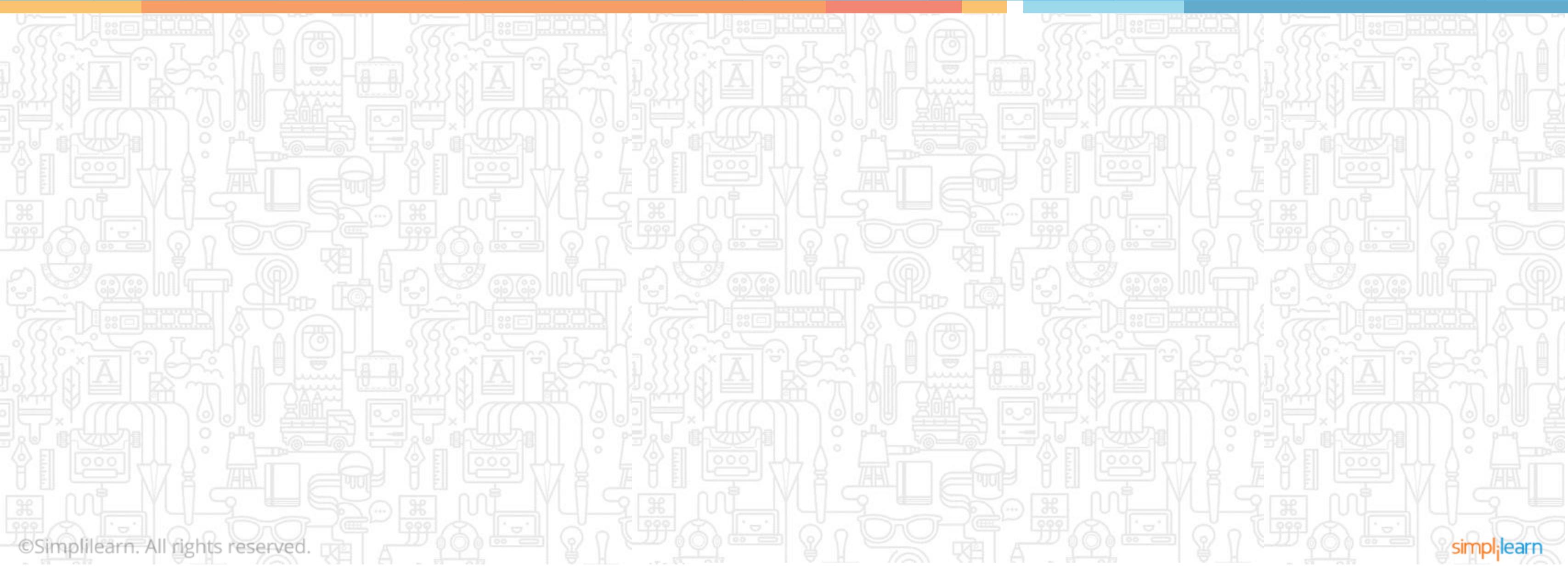
# Ethereum Languages

Contracts are compiled into EVM bytecode and deployed to Ethereum Blockchain for execution.



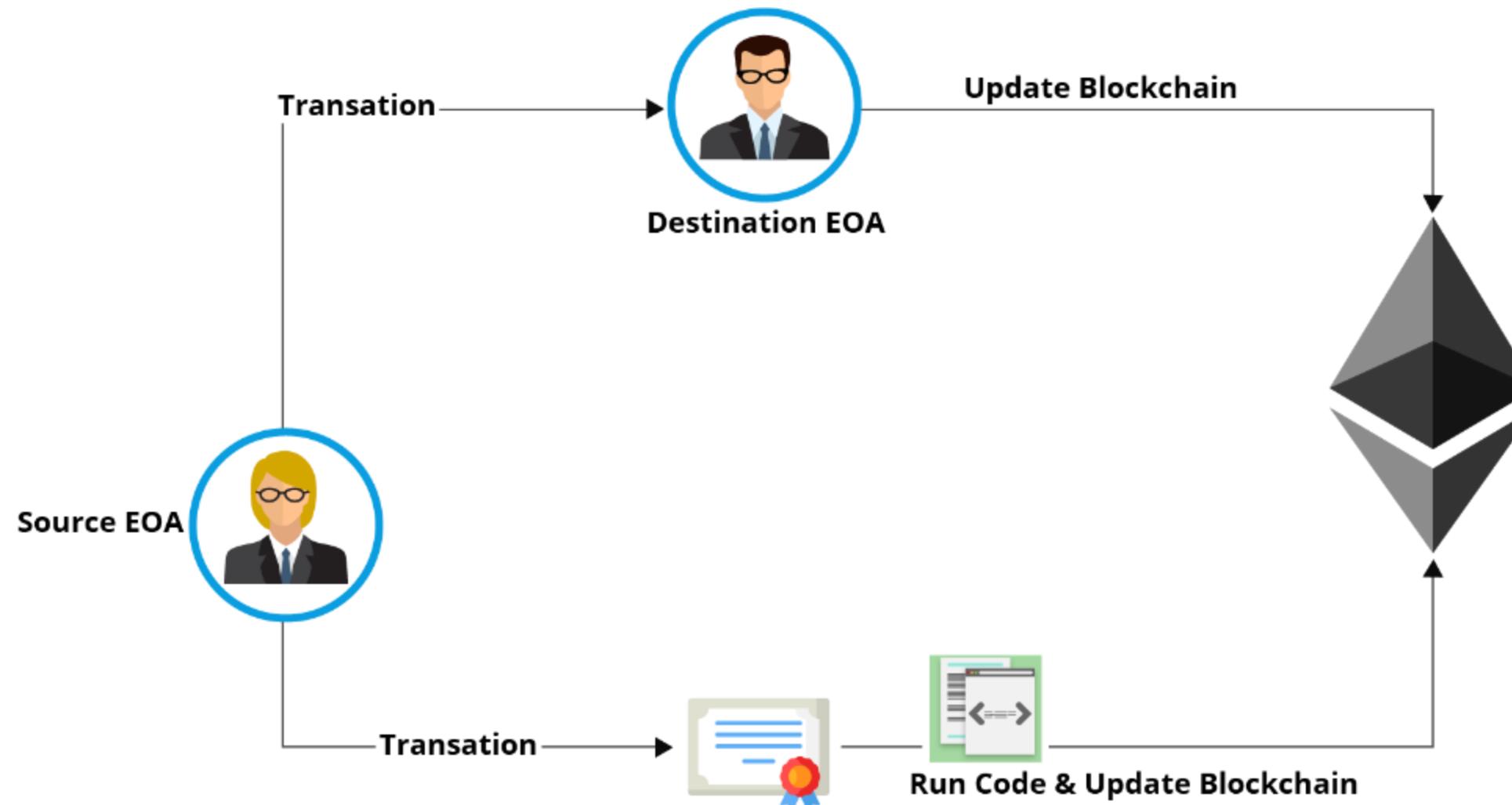
# Ethereum

## Types of Ethereum Accounts



# Externally Owned Accounts (EOA)

EOA is an account controlled by a private key that has the ability to send Ethers and messages from it.



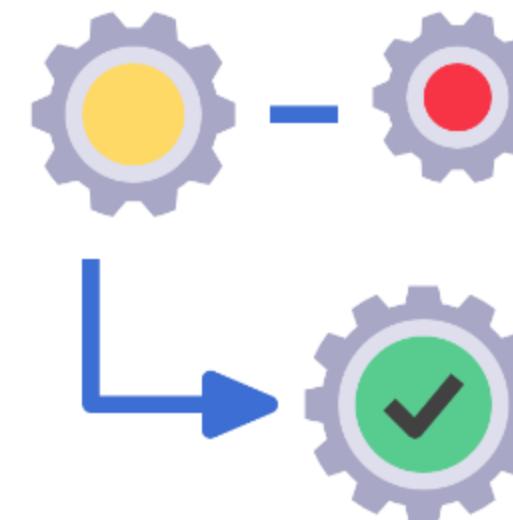
# Smart Contract

A code running on top of the Blockchain containing the set of rules for the nodes to agree upon so that they interact with each other.



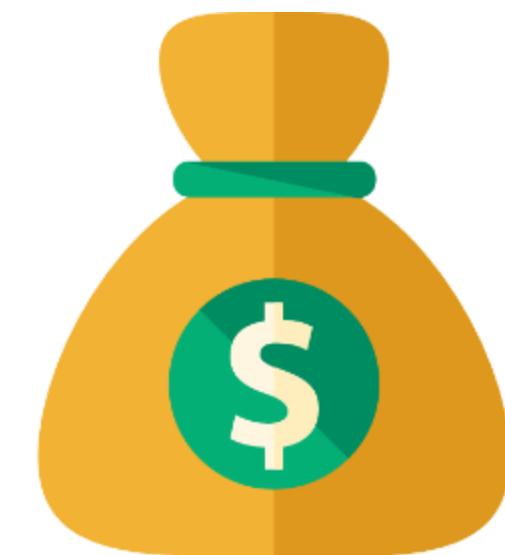
# Characteristics of Smart Contract

**Self-verifying**



**Self-executing**

**Cost Saving**



**Tamper Resistant**

# Process of Smart Contracts



An optional contract between the parties is written as a code into the Blockchain. The individuals involved are anonymous, but the contact is made through a public ledger



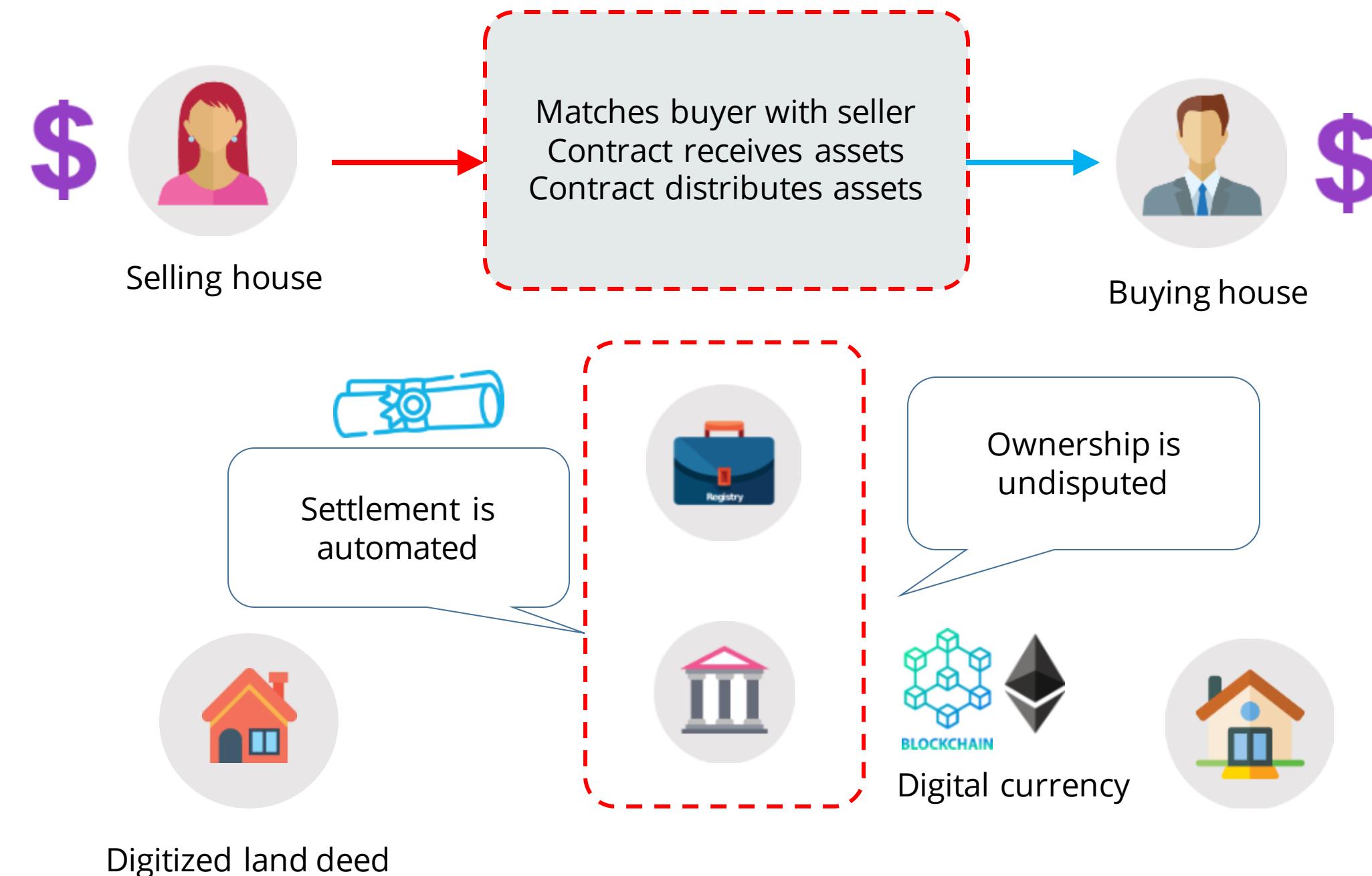
A trigger event like an expiration date or a strike price is hit, and the contract executes itself according to the coded terms



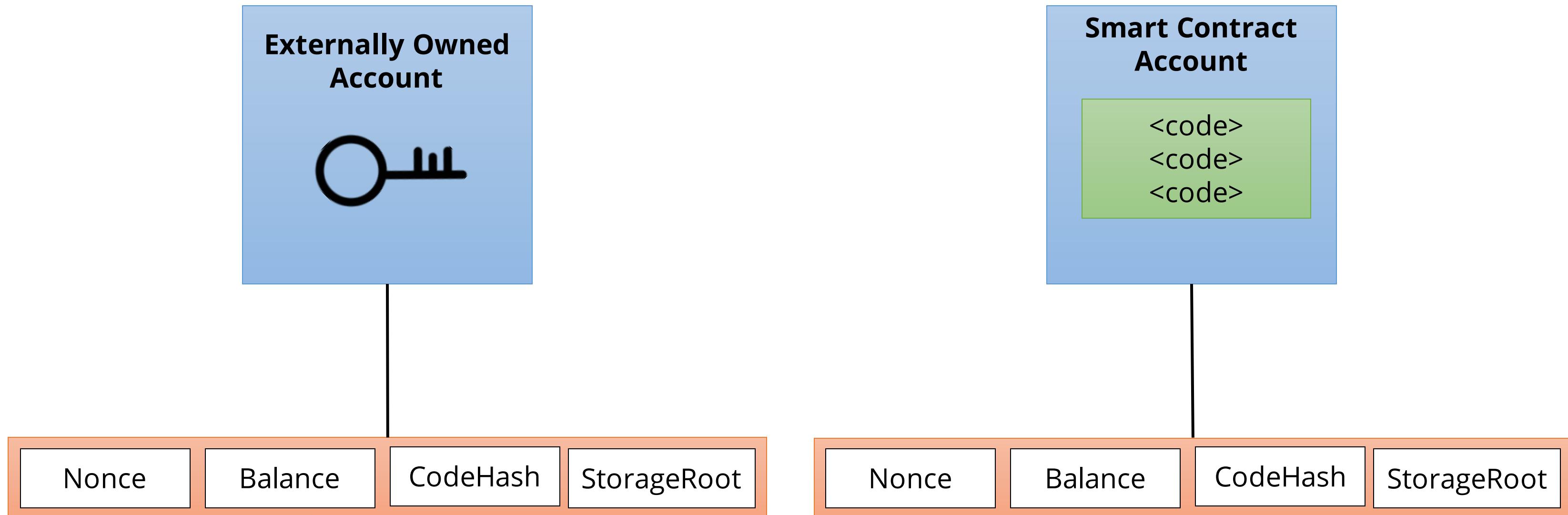
Regulators can use the Blockchain to understand the activities in the market while maintaining the privacy of an individual's position



# Smart Contract: Example

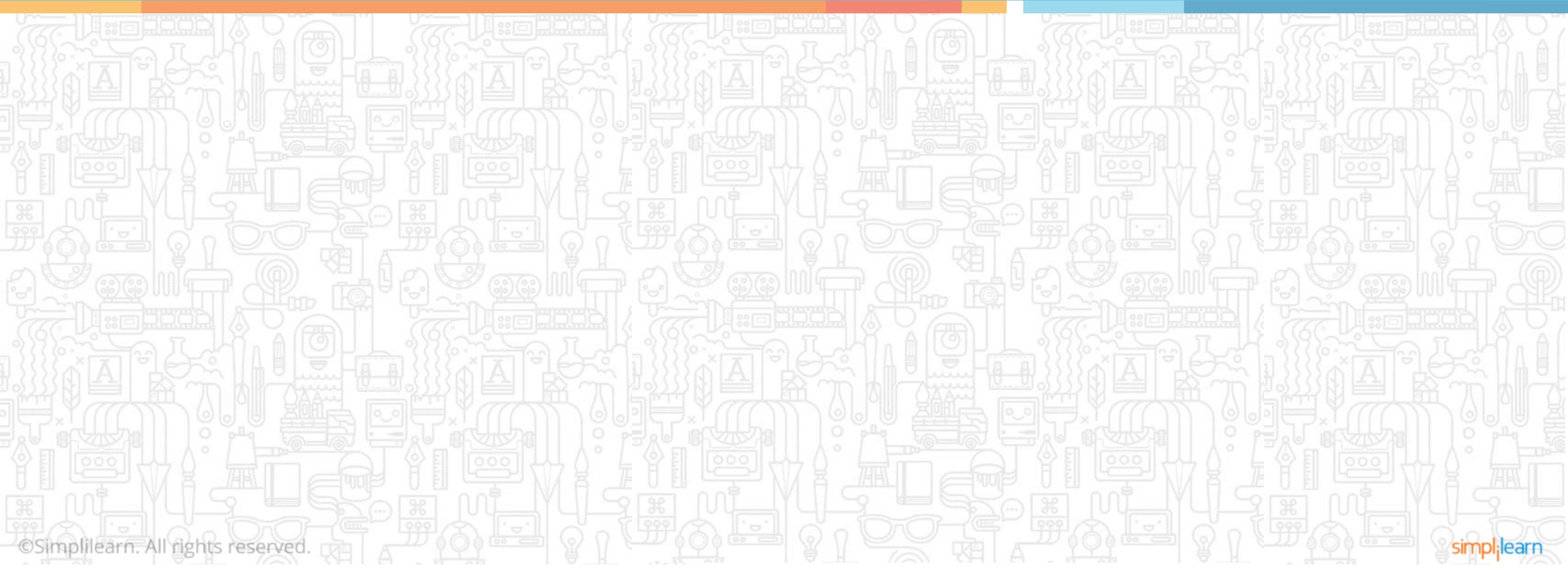


# EOA vs. Smart Contract



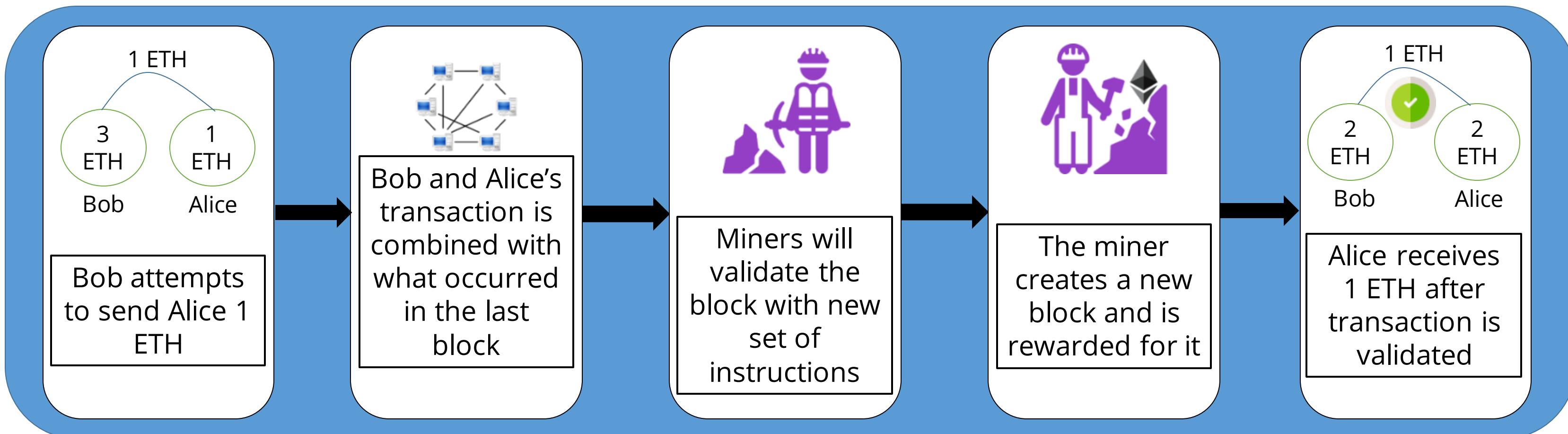
# Ethereum

## Ethereum Mining

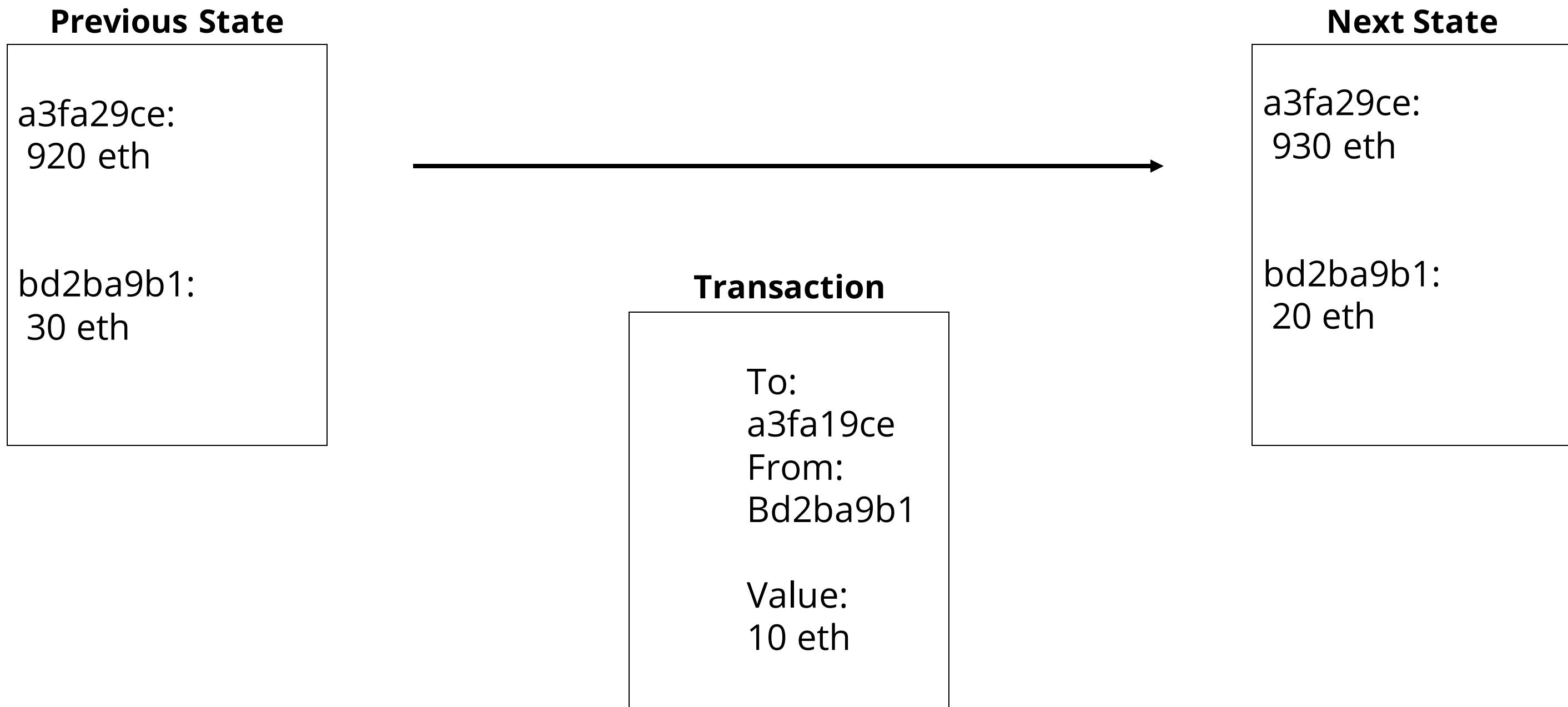


# Ethereum Mining

Ethereum makes use of Proof of Work mechanism to ensure security.  
The algorithm used is Ethash.



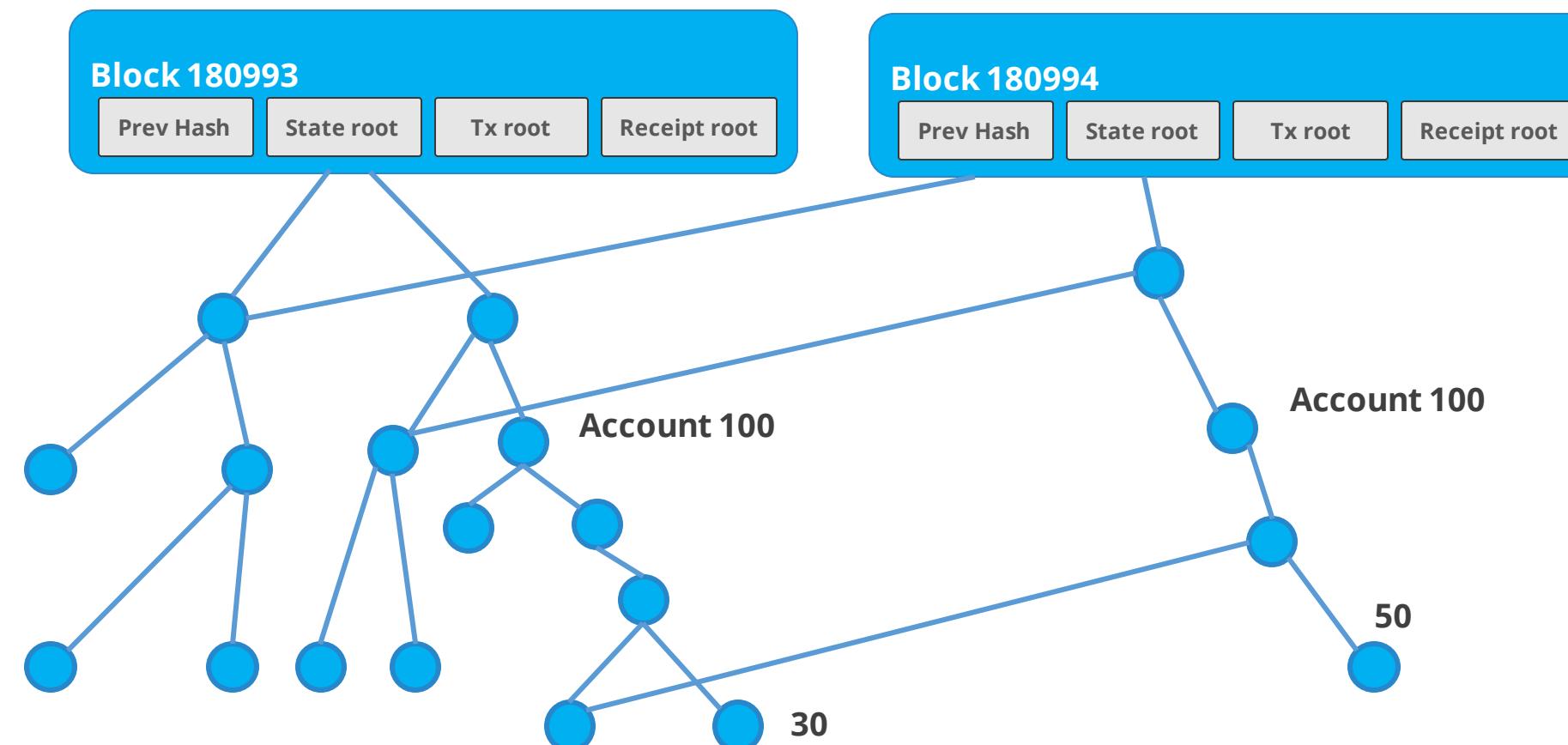
# Ethereum Mining



# State Storage

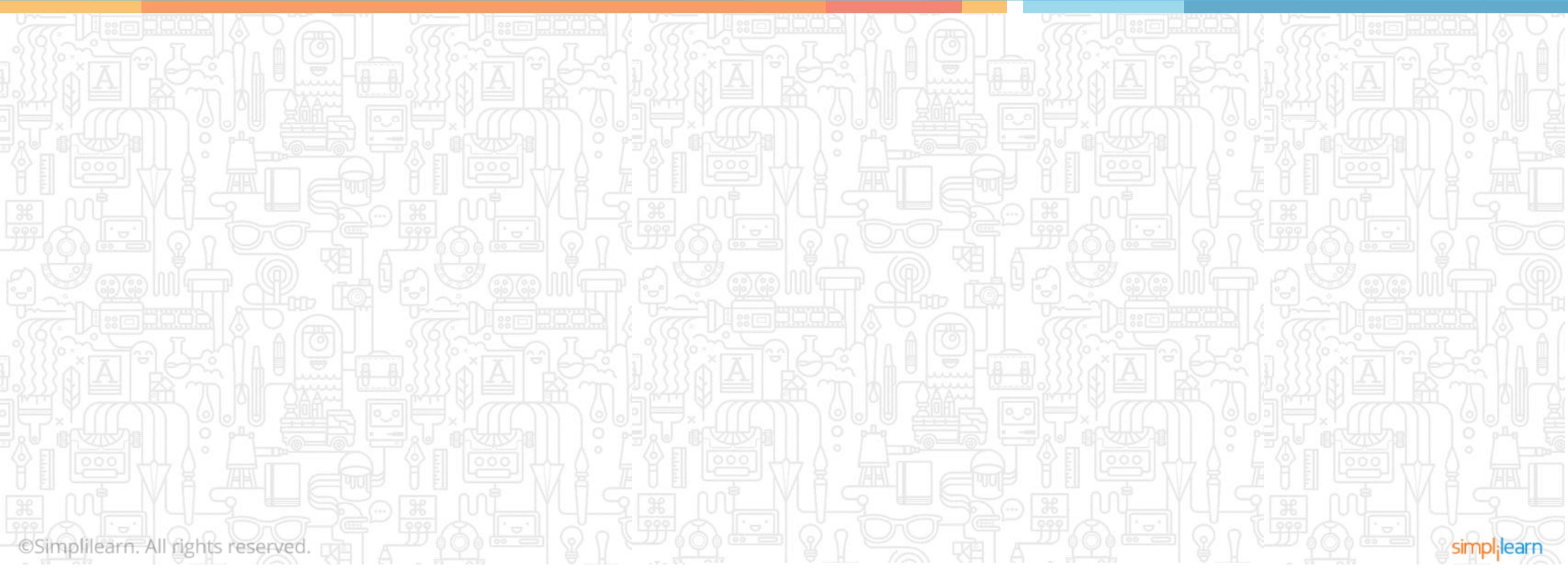
The states are assembled into a state tree that is linked to the account and the block.

The Ethereum includes state roots that store the root hash of the hash tree which represents the system state when the block was created.



# Ethereum

## Ethereum Ecosystem



# Types of Ethereum Tools

---

Geth

Mist Wallet

Ganache

Swarm

Parity

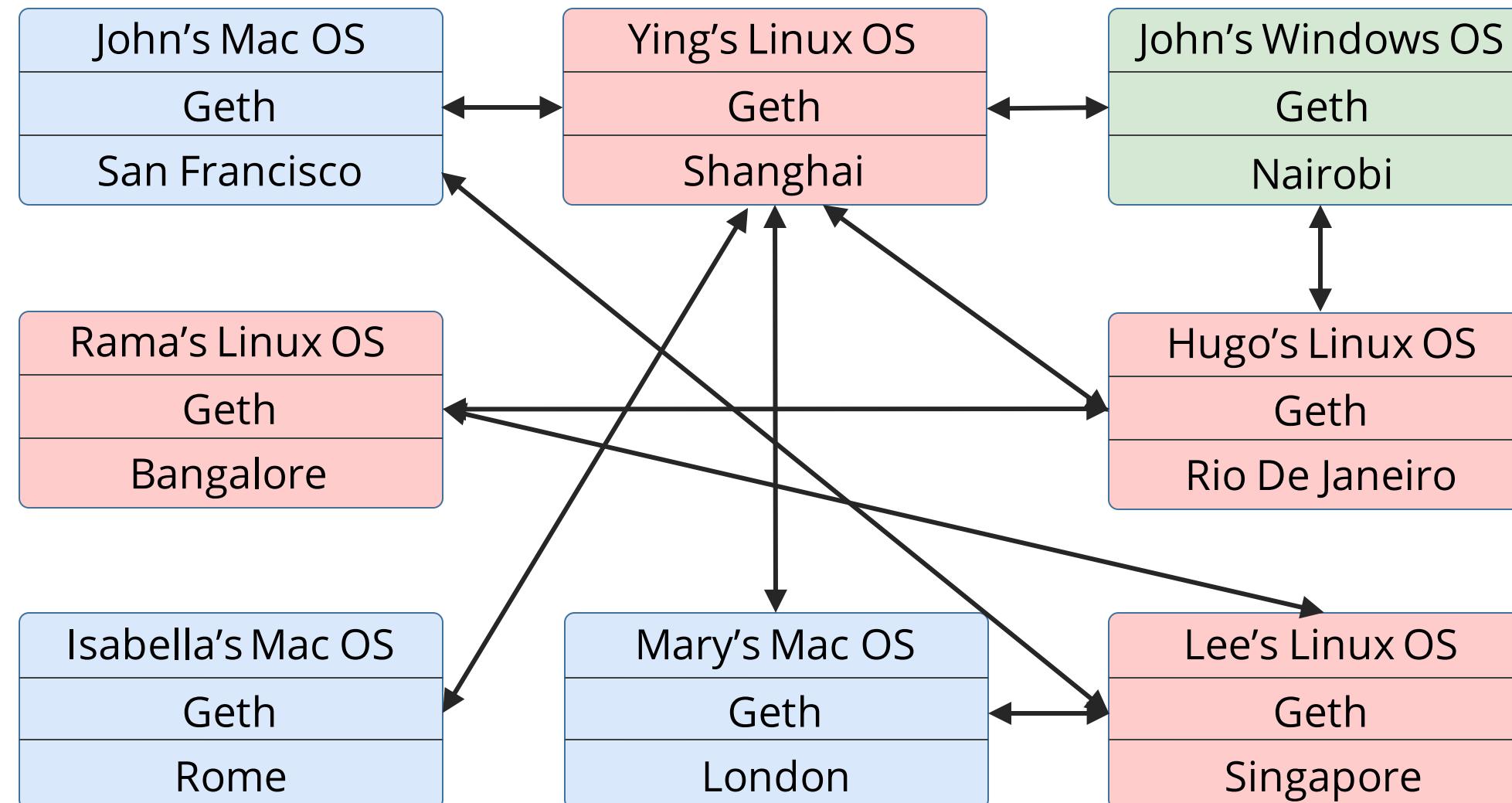
IPFS

MetaMask

Whisper

# Geth

Multipurpose command line tool that runs an Ethereum node implemented in Go



## Ethereum Blockchain Network

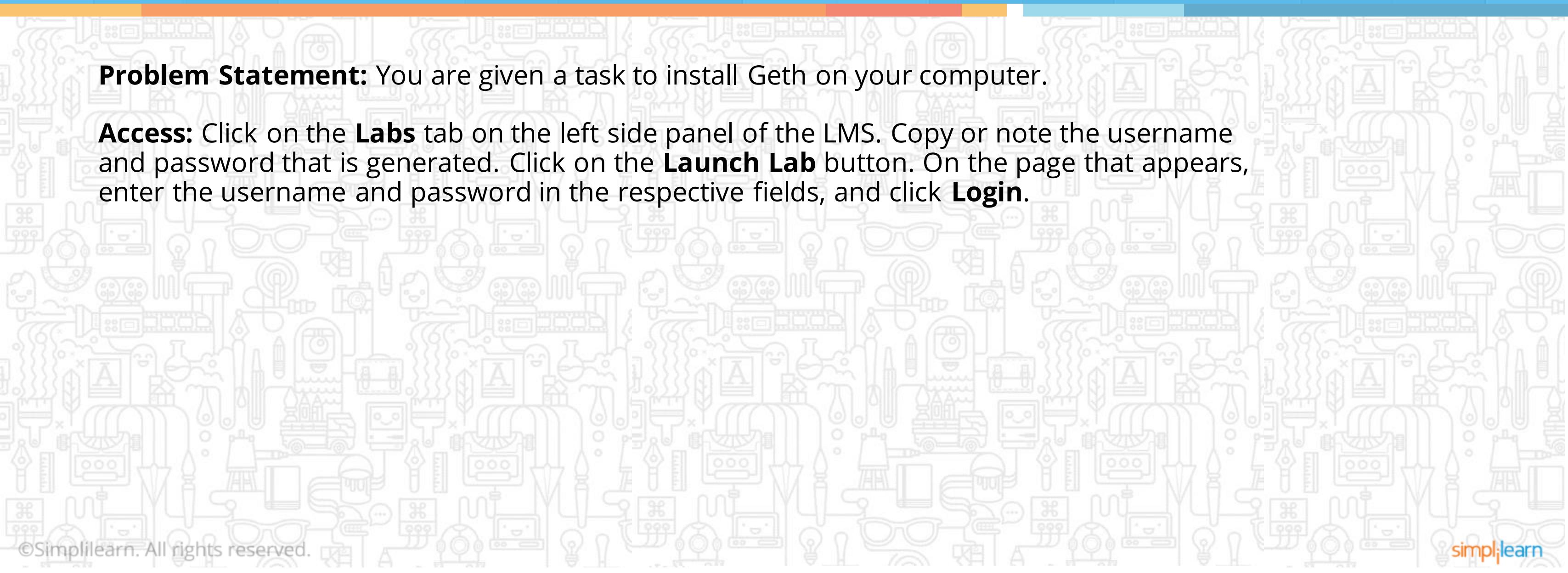
## Assisted Practice

### Installation of Geth

Duration: 10 mins

**Problem Statement:** You are given a task to install Geth on your computer.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



Perform the following commands to install Geth:

**Step 01**

`sudo apt-get install software-properties-common`

**Step 02**

`sudo add-apt-repository -y ppa:ethereum/ethereum`

**Step 03**

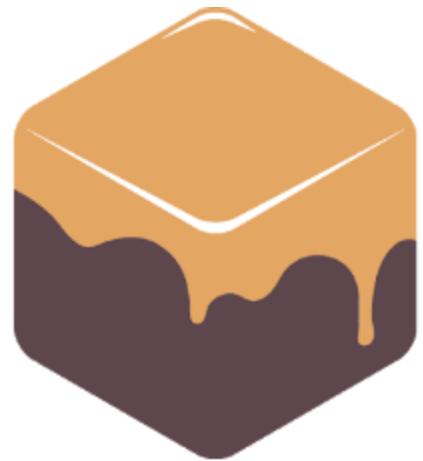
`sudo apt-get update`

**Step 04**

`sudo apt-get install ethereum`

# Ganache CLI

---



*Ganache*

Ganache CLI is a fast and customizable Blockchain emulator which allows to make calls to Blockchain without running the actual Ethereum node.

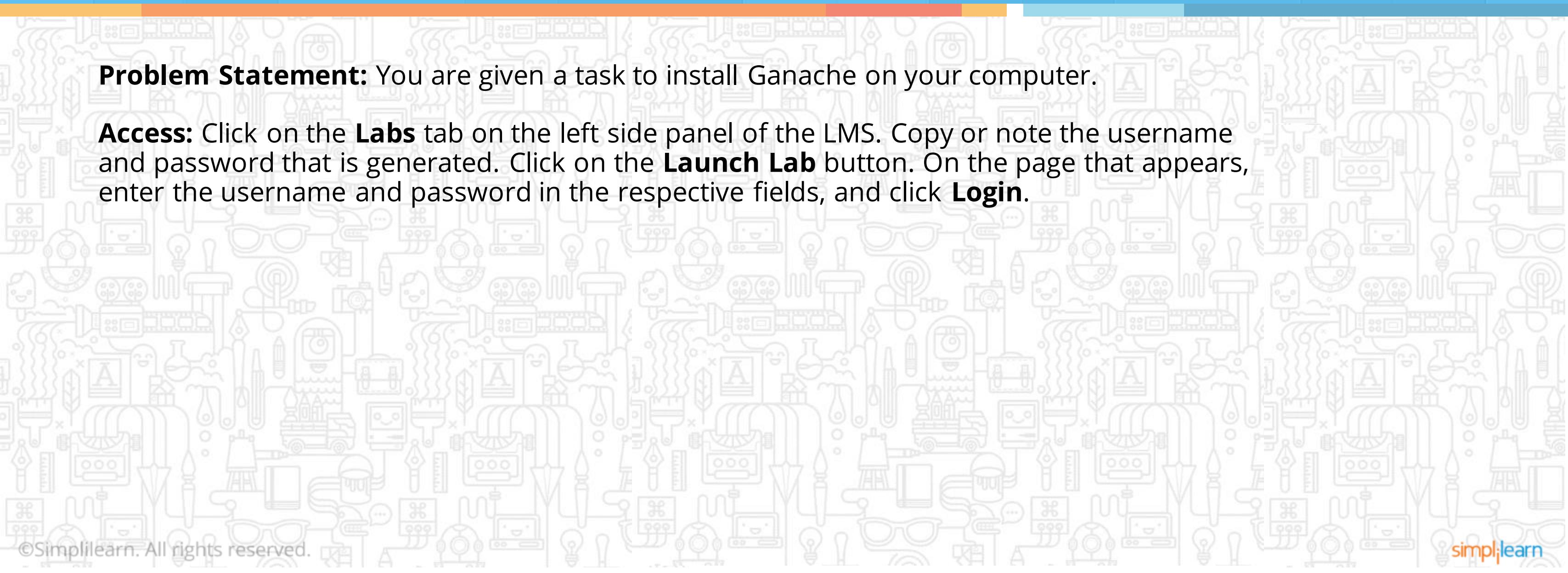
## Assisted Practice

### Installation of Ganache

Duration: 10 mins

**Problem Statement:** You are given a task to install Ganache on your computer.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



Perform the following commands to install Ganache:

### Step 01

```
git clone  
https://github.com/trufflesuite/ganache.git
```

### Step 02

```
cd ganache
```

### Step 03

```
npm install
```

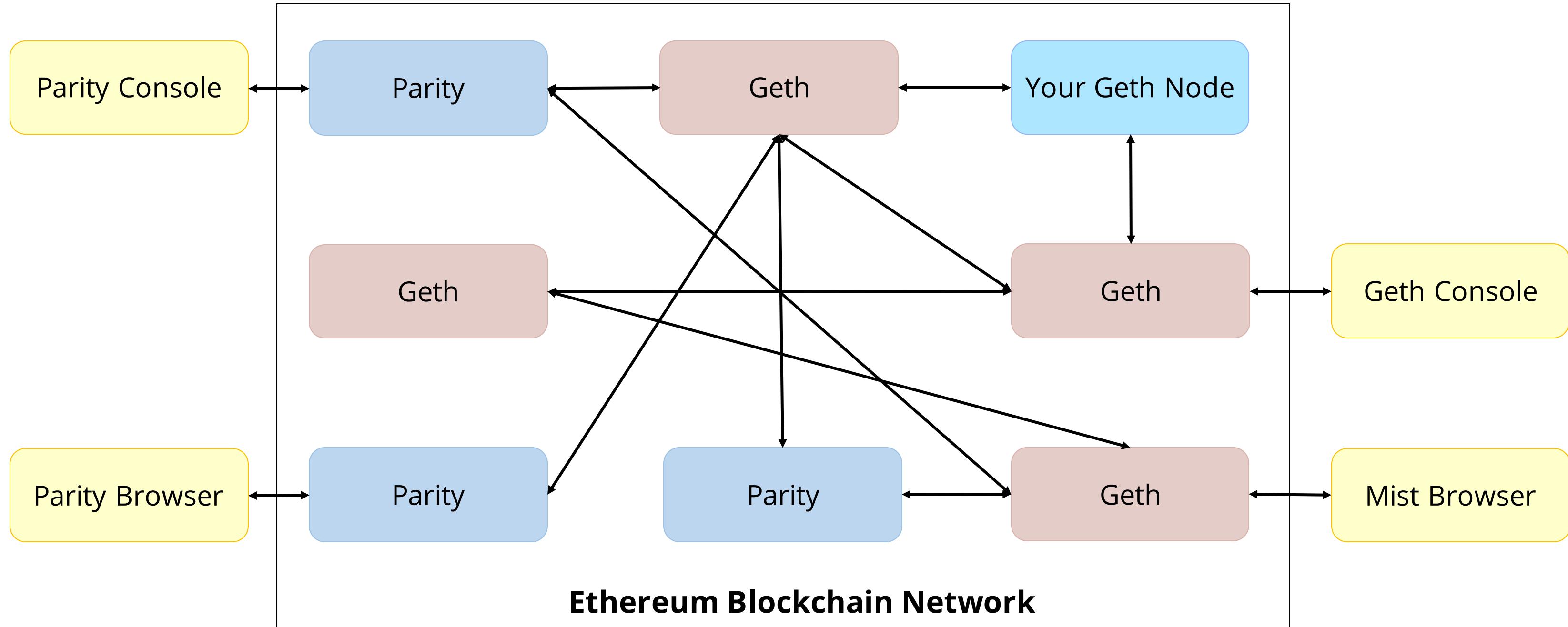
### Step 04

```
npm start
```

Parity

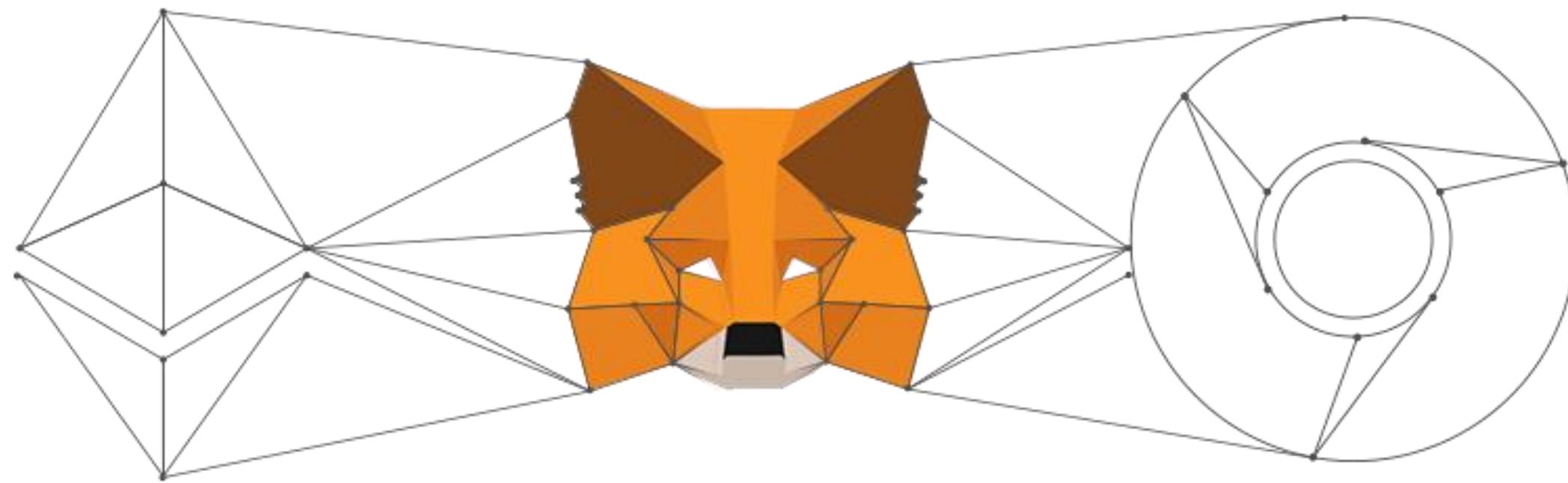


The fastest, lightest, and most secure Ethereum client that provides the core infrastructure essentials for quick and reliable services.



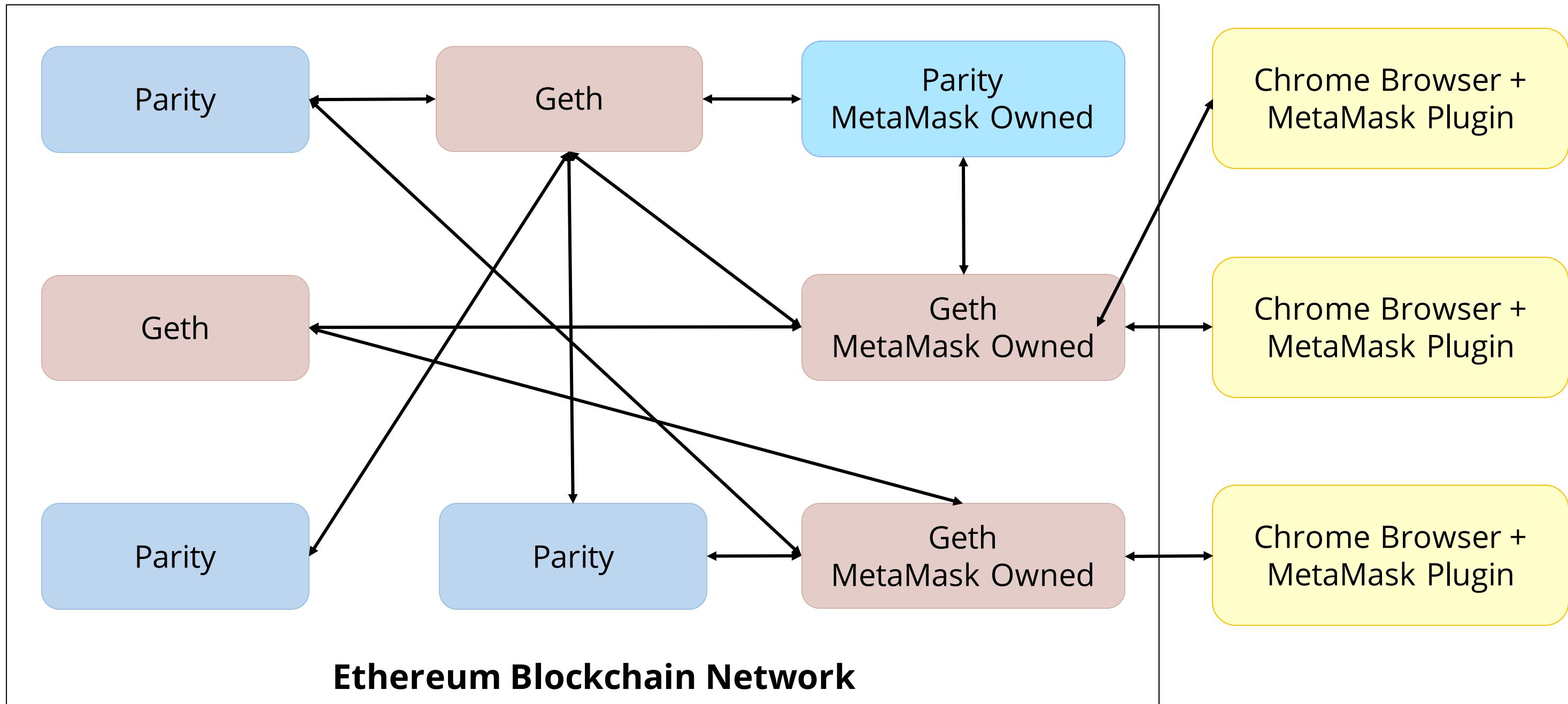
# MetaMask

---



MetaMask turns Google Chrome into a browser that allows the users to send and receive transactions and also to fetch data from the Blockchain.

# MetaMask Accounts



## Assisted Practice

Duration: 15 mins

### Transfer Ethers Using MetaMask

**Problem Statement:** You are given a task to connect MetaMask to Ganache and transfer Ethers from one account to another.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

## Assisted Practice: Steps



**Step 01**

Add MetaMask extension on Google Chrome

**Step 02**

Create two accounts on MetaMask

**Step 03**

Connect MetaMask to Ganache

**Step 04**

Send Ethers from one account to another

## Unassisted Practice

Duration: 10 mins

### MetaMask Ether Faucet

**Problem Statement:** You are given a task to add Ethers to your MetaMask account using Ether Faucet.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

**Note:** This practice is not graded. It is only intended for you to apply the knowledge you have gained to solve real-world problems.

## Unassisted Practice: Steps



### Step 01

Open MetaMask, and select Ropsten Test Network

### Step 02

Click **Deposit** to add Ethers

### Step 03

Under Test Faucet, click **Get Ether**

### Step 04

Click **request 1 ether from faucet**

### Step 05 account)

Wait for the transaction to complete (one Ether will be added to your

# Unassisted Practice: Output

## MetaMask Ether Faucet

faucet

address: 0x81b7e08f65bdf5648606c89998a9cc8164397647

balance: ...

[request 1 ether from faucet](#)

user

address: 0xc2ac9c24b7aef5aff9f521176082ce6b2eb0aadd

balance: ...

donate to faucet:

1 ether

10 ether

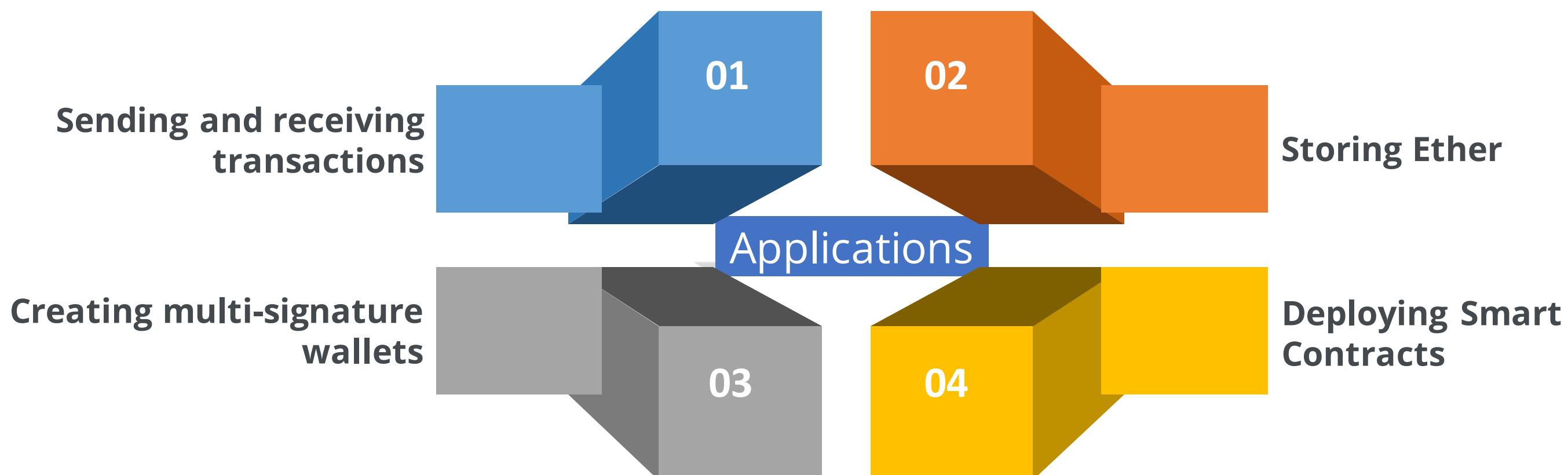
100 ether

transactions

[0xcc7370446134edd8595e230012f8631e798b7029f881d5d579e4fb0ec48b4f42](#)

# Mist Wallet

End-user interface for Ethereum that is developed for browsing and using Dapps



# Mist Wallet

The screenshot shows the Mist Ethereum Wallet interface. At the top, there's a dark header bar with the title "Ethereum Wallet". Below it is a light-colored navigation bar with icons for "WALLETS" (a folder icon), "SEND" (an upward arrow icon), "CONTRACTS" (a document icon), and "BALANCE" (a coin icon). The balance is shown as "0.00 ETHER". In the center of the screen, the title "Accounts Overview" is displayed above a list of accounts. The first account is named "ACCOUNT 1" with a green alien icon, showing a balance of "0.00 ether" and the address "0xA63DA94b5c6Ed296...". The second account is named "ACCOUNT 2" with a red alien icon, showing a balance of "0.00 ether" and the address "0xb22fa02799bBAe0F785FB09431A661c7a76CEd44". At the bottom left, there's a blue button with a plus sign and the text "ADD ACCOUNT". A small "Wallet Contracts" tab is visible at the bottom left corner.

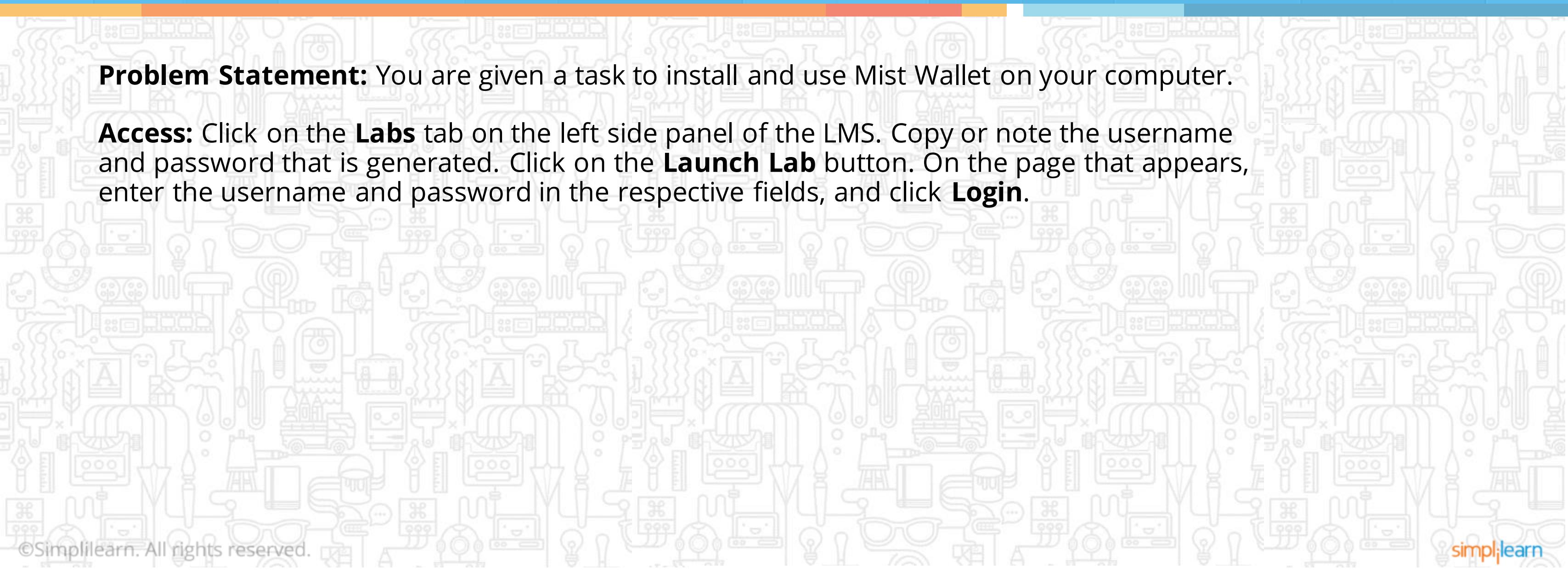
## Assisted Practice

Duration: 15 mins

### Install and Use Mist Wallet

**Problem Statement:** You are given a task to install and use Mist Wallet on your computer.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



**Step 01**

Visit <https://github.com/ethereum/mist/releases>

**Step 02**

Download Mist wallet, and run the exe file

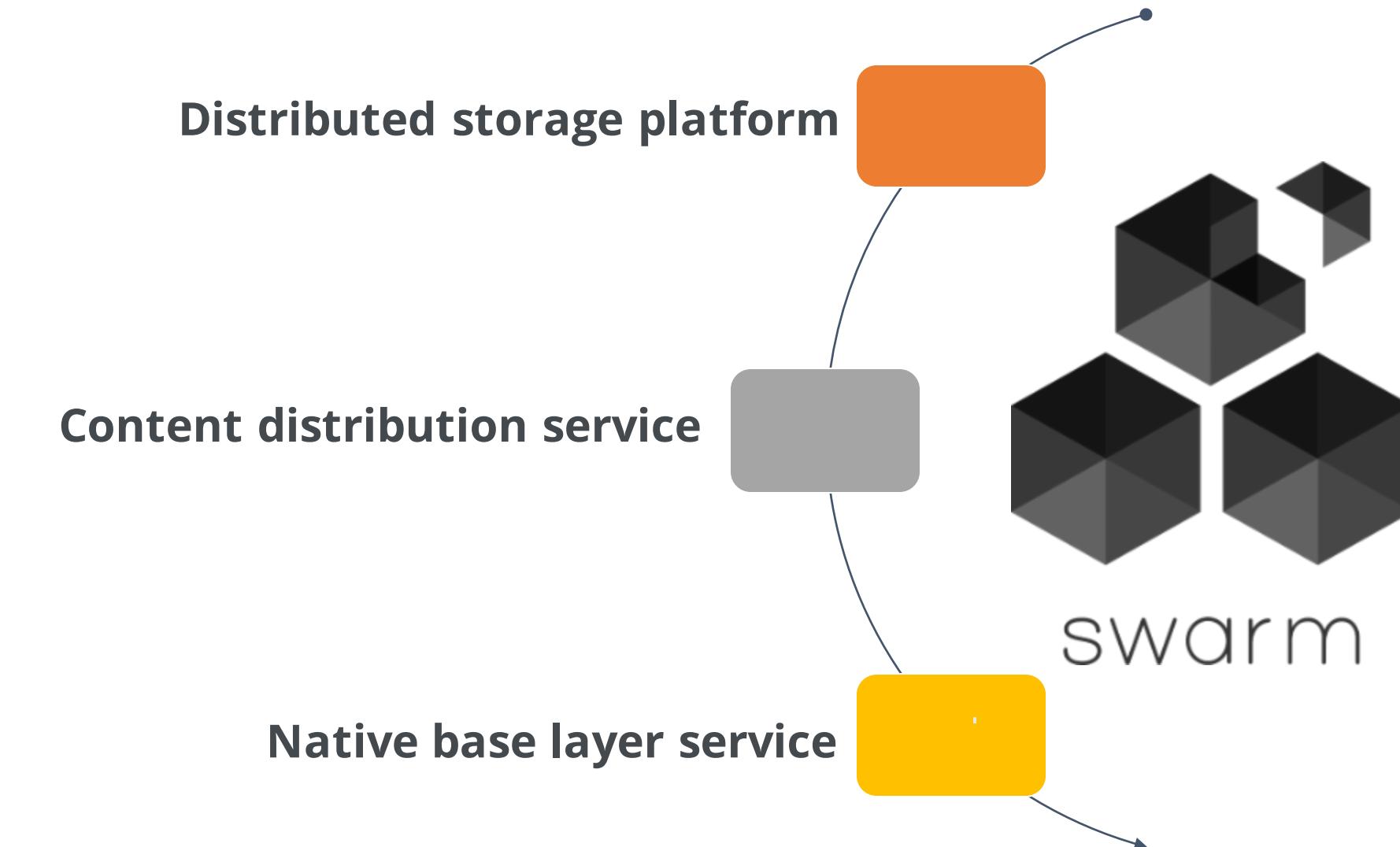
**Step 03**

Create two accounts

**Step 04**

Send Ethers from one account to another

# Swarm



Provides sufficient decentralization and redundant storage of Ethereum records to distribute Blockchain data as well as Dapp codes

# InterPlanetary File System (IPFS)

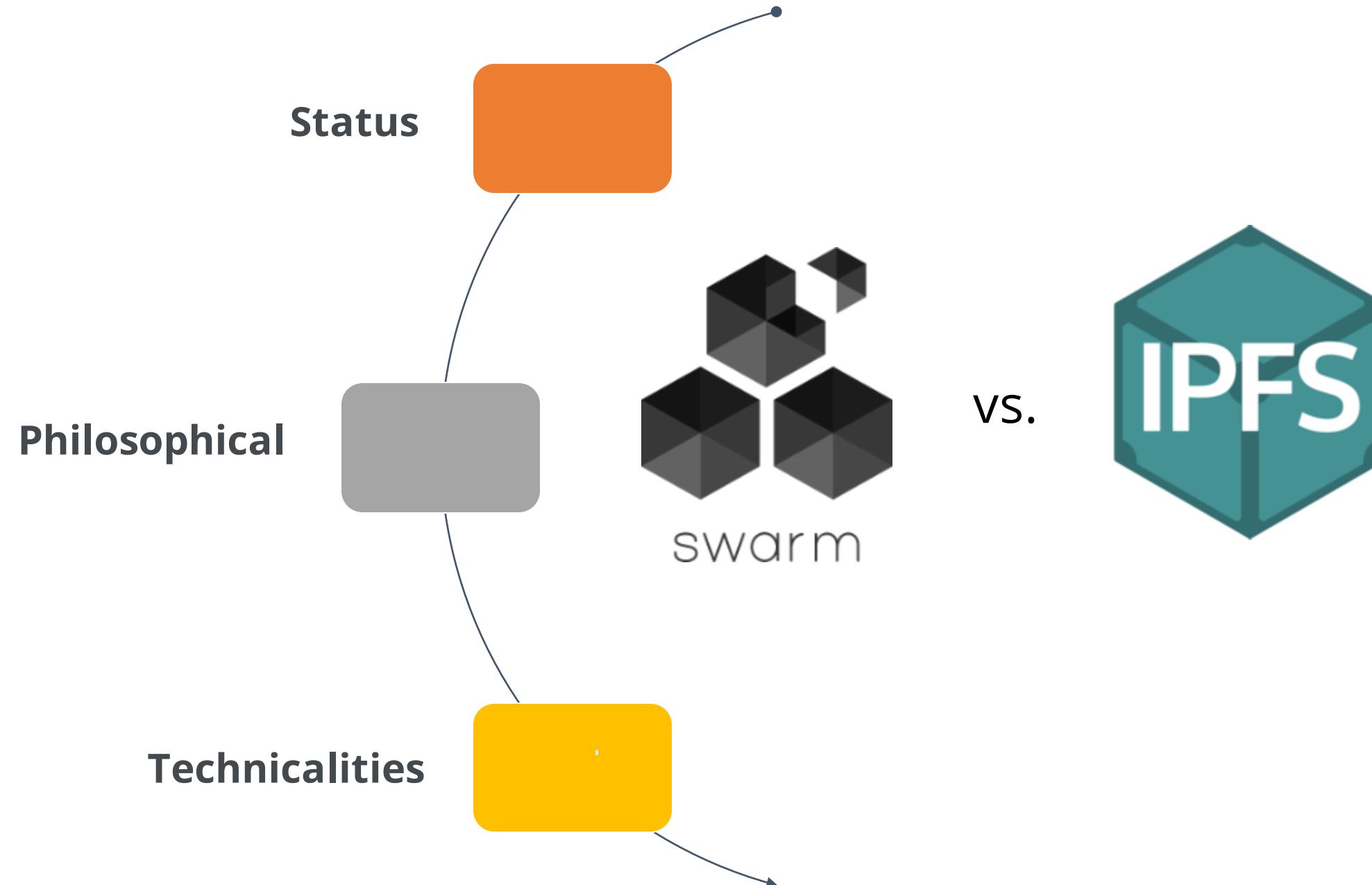
---



IPFS is a decentralized storage system that is not related directly to Ethereum but can be integrated with it.

# Swarm vs. IPFS

Swarm and IPFS differs on the given parameters:



An identity-based communication protocol for Dapps to interact with each other

## Drawbacks in Whisper



**Low-level API**



**Low bandwidth**



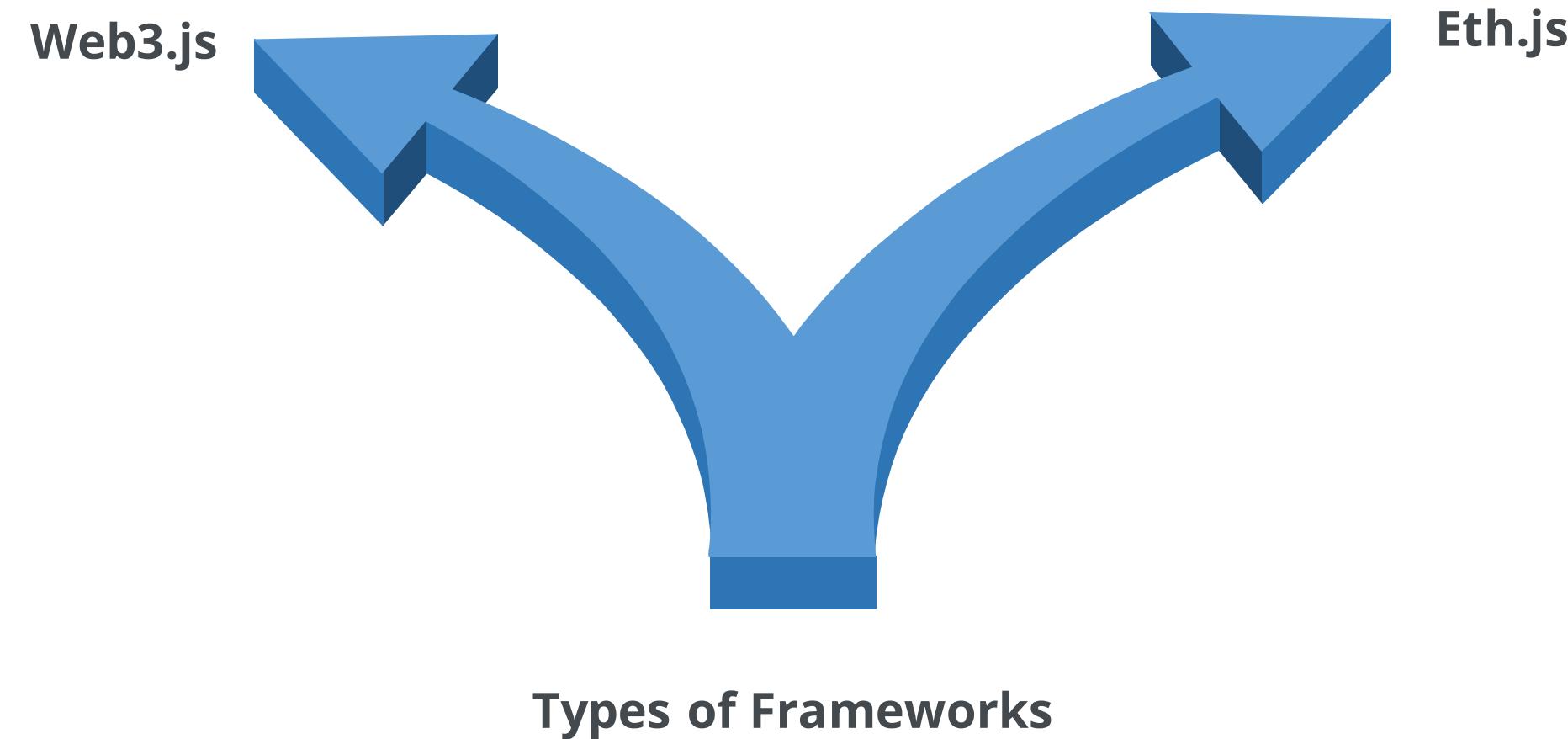
**Uncertain latency**



**Unreliable for tracing packets**

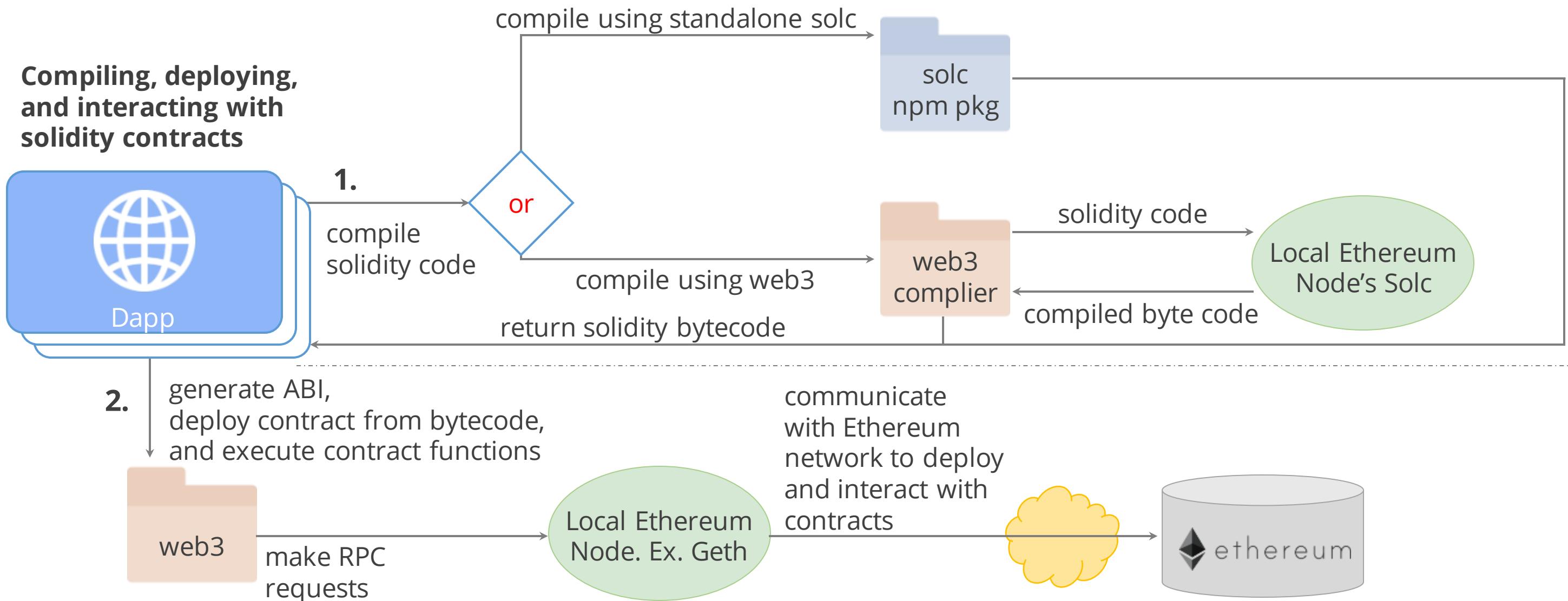
# Ethereum Frameworks

---



# Web3.js

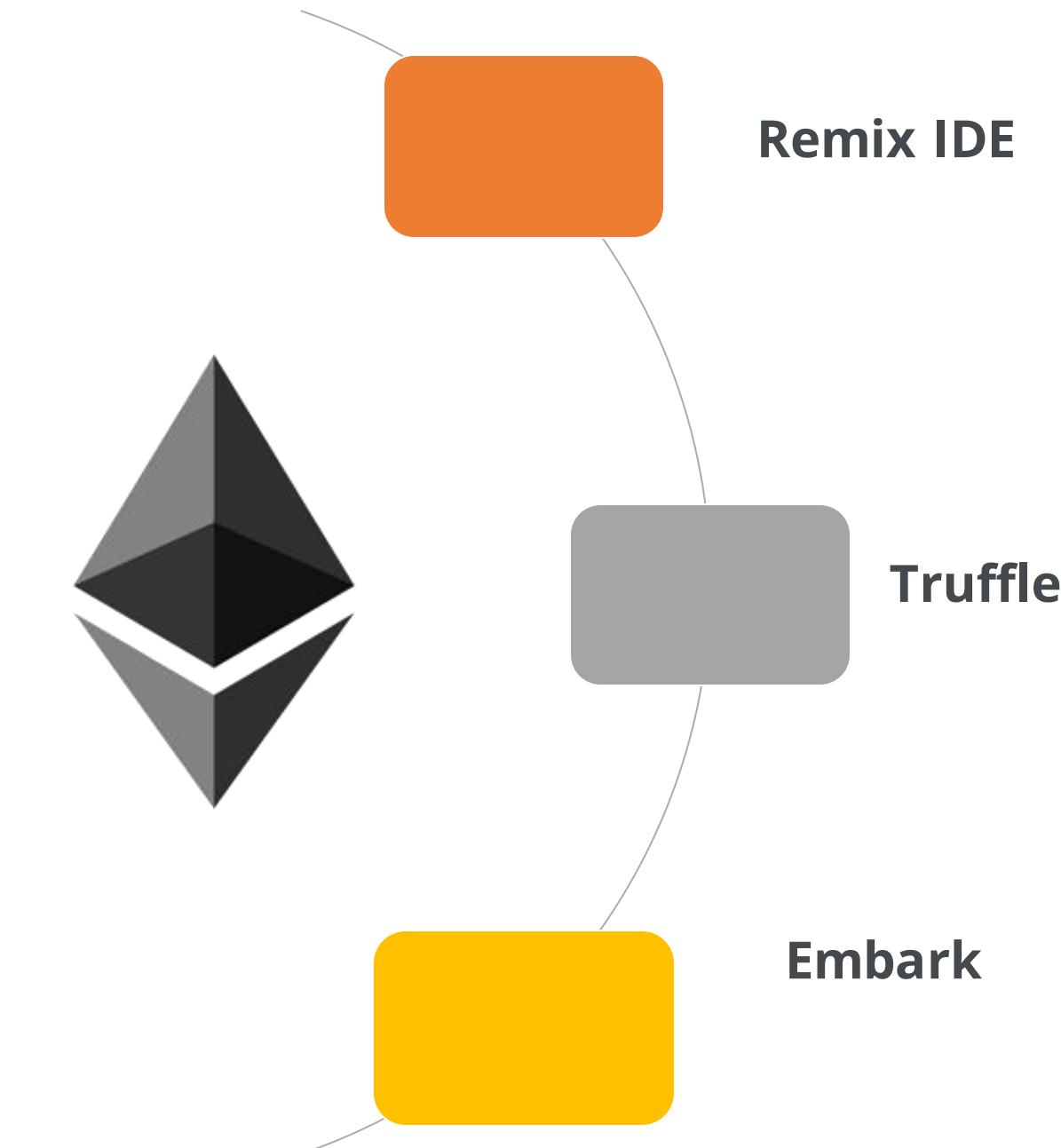
The key connection between Ethereum network and Dapp that allows to compile, deploy, and interact with smart contracts



A highly optimized, lightweight JS utility for Ethereum based on web.js

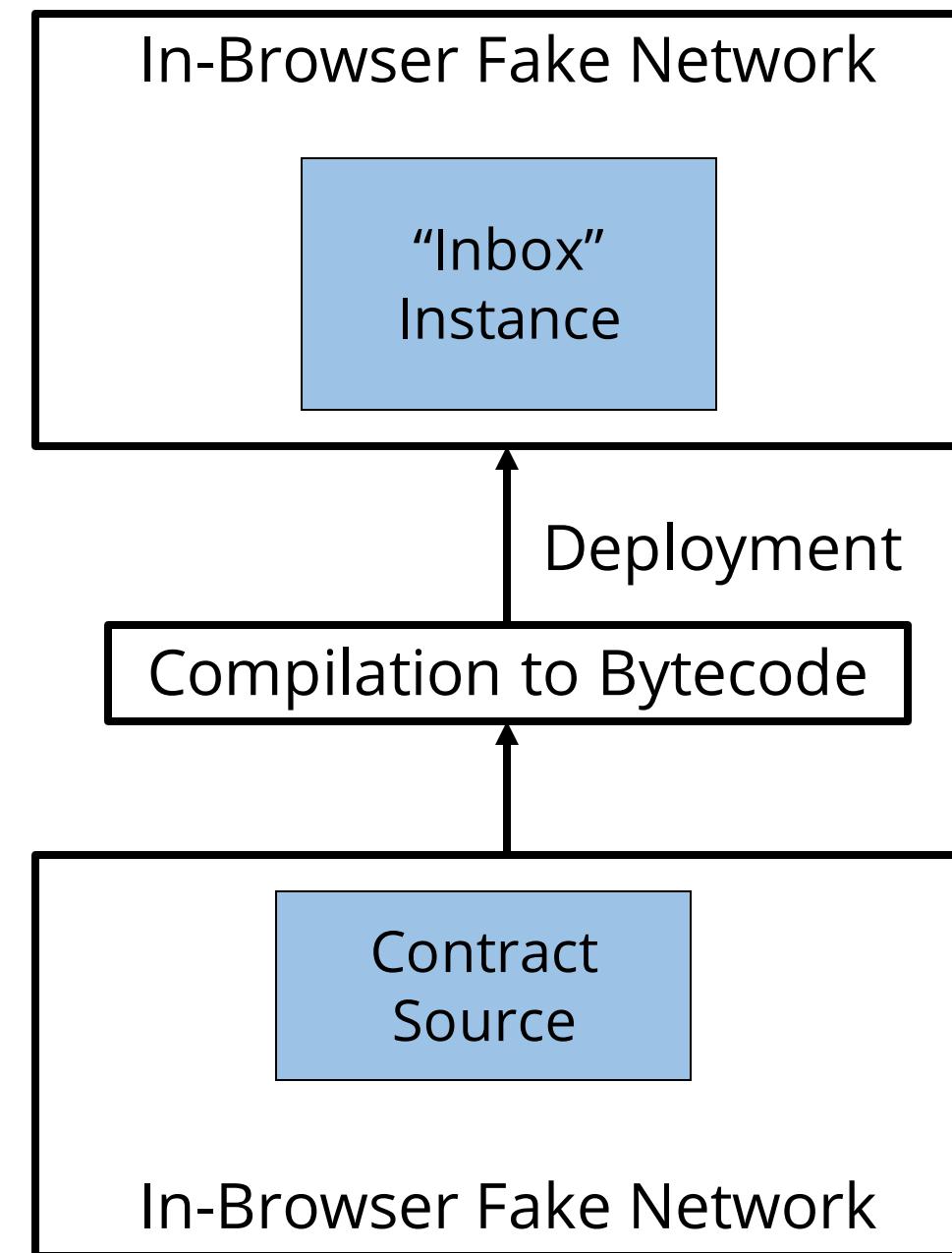
All the unnecessary characters are removed from the source code, and the code is reduced to 160 KB

# Ethereum Development Environment



# Remix IDE

A browser-based compiler and IDE that enables users to debug transactions and build Ethereum contracts with Solidity language.



# Remix IDE

The screenshot shows the Remix IDE interface. On the left, there's a file tree with 'browser' and 'config' expanded. A central code editor window titled 'browser/demo.sol' contains the following Solidity code:

```
pragma solidity ^0.4.18;
contract YourContract { }
```

At the top right of the code editor, it says 'ContractDefinition YourContract' and '0 reference(s)'. To the right of the code editor is a control panel with the following sections:

- Current**: version:0.4.18+commit.9cf6e910.Emscripten clang
- Select new compiler version** dropdown
- Auto compile  Enable Optimization
- Hide warnings
- Start to compile** button
- YourContract** dropdown with 'Details', 'ABI', and 'Bytecode' buttons
- A green box labeled 'YourContract' with a close button

At the bottom of the interface, there are transaction-related controls: a dropdown set to '[2] only remix transactions, script', a search bar for 'Search transactions', and some small icons.

A command line tool used to write, test, and deploy contracts on any Ethereum network

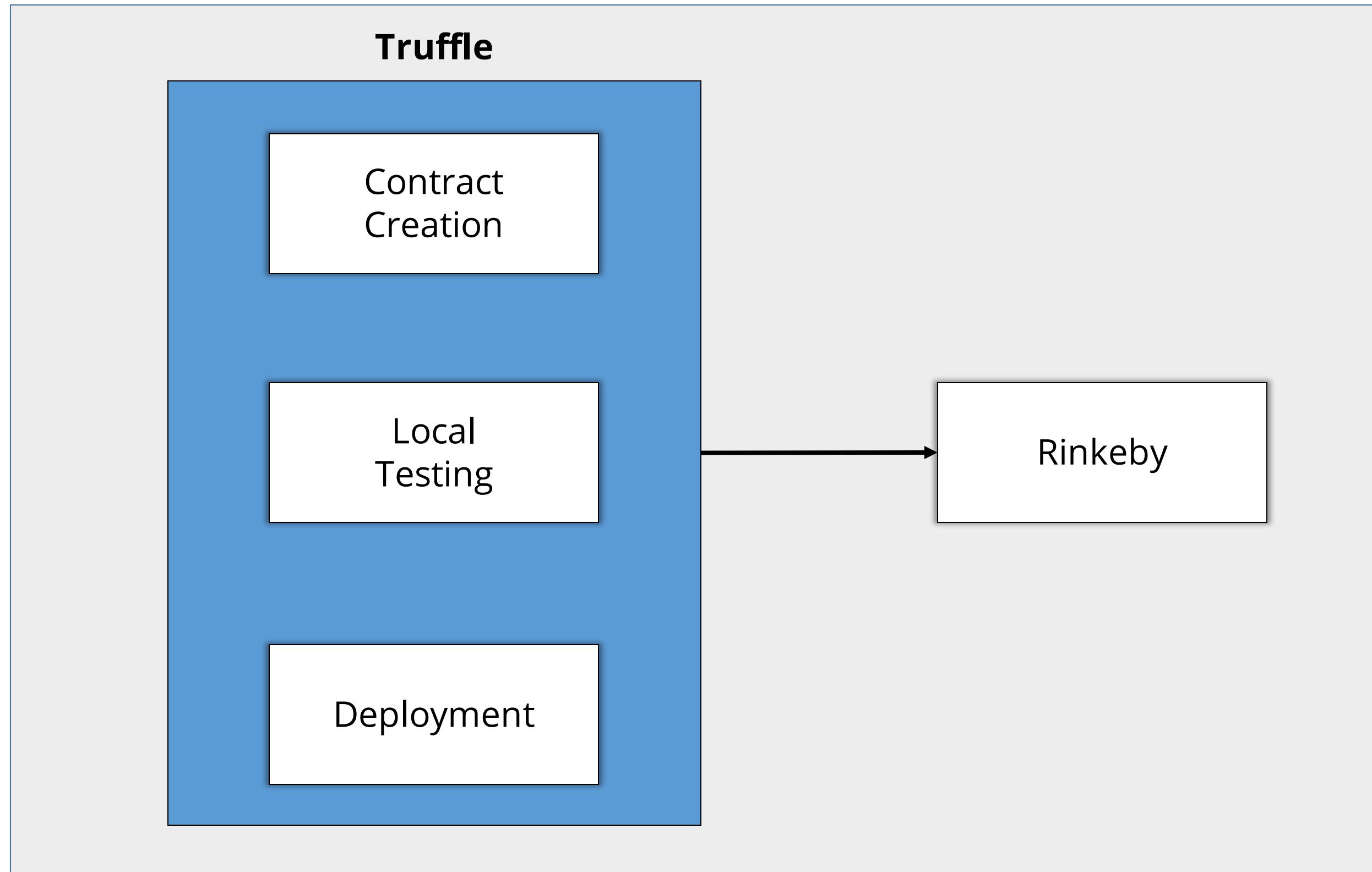
**Smart contract lifecycle management**



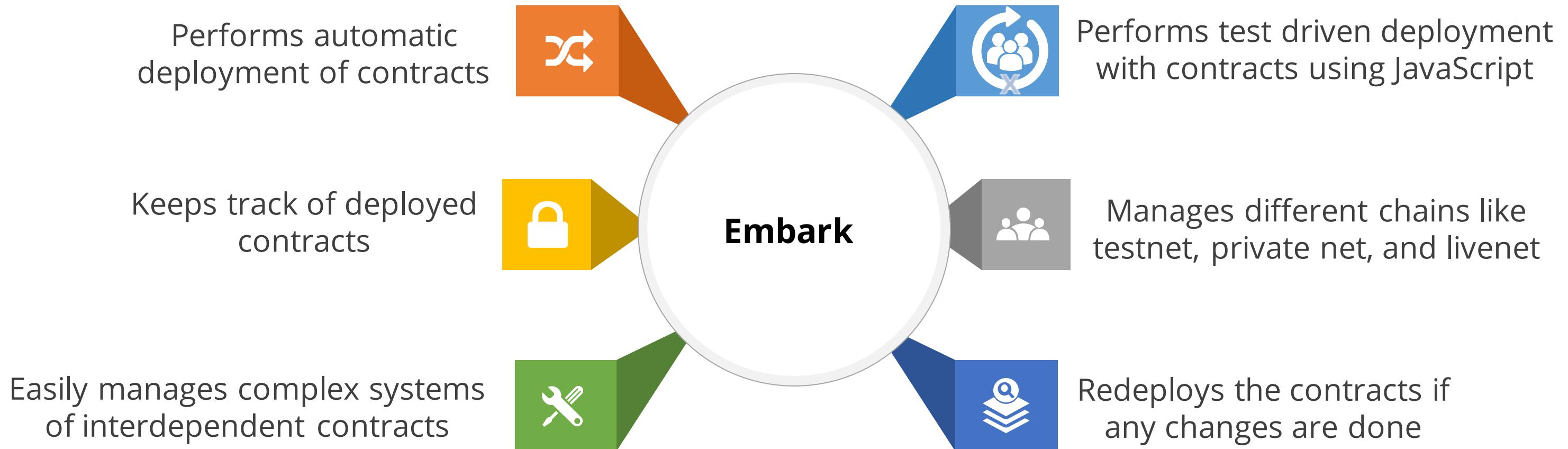
**Automated contract testing**

**Scriptable deployment and migrations**

**Powerful interactive console**



An environment that allows you to easily develop and deploy decentralized applications



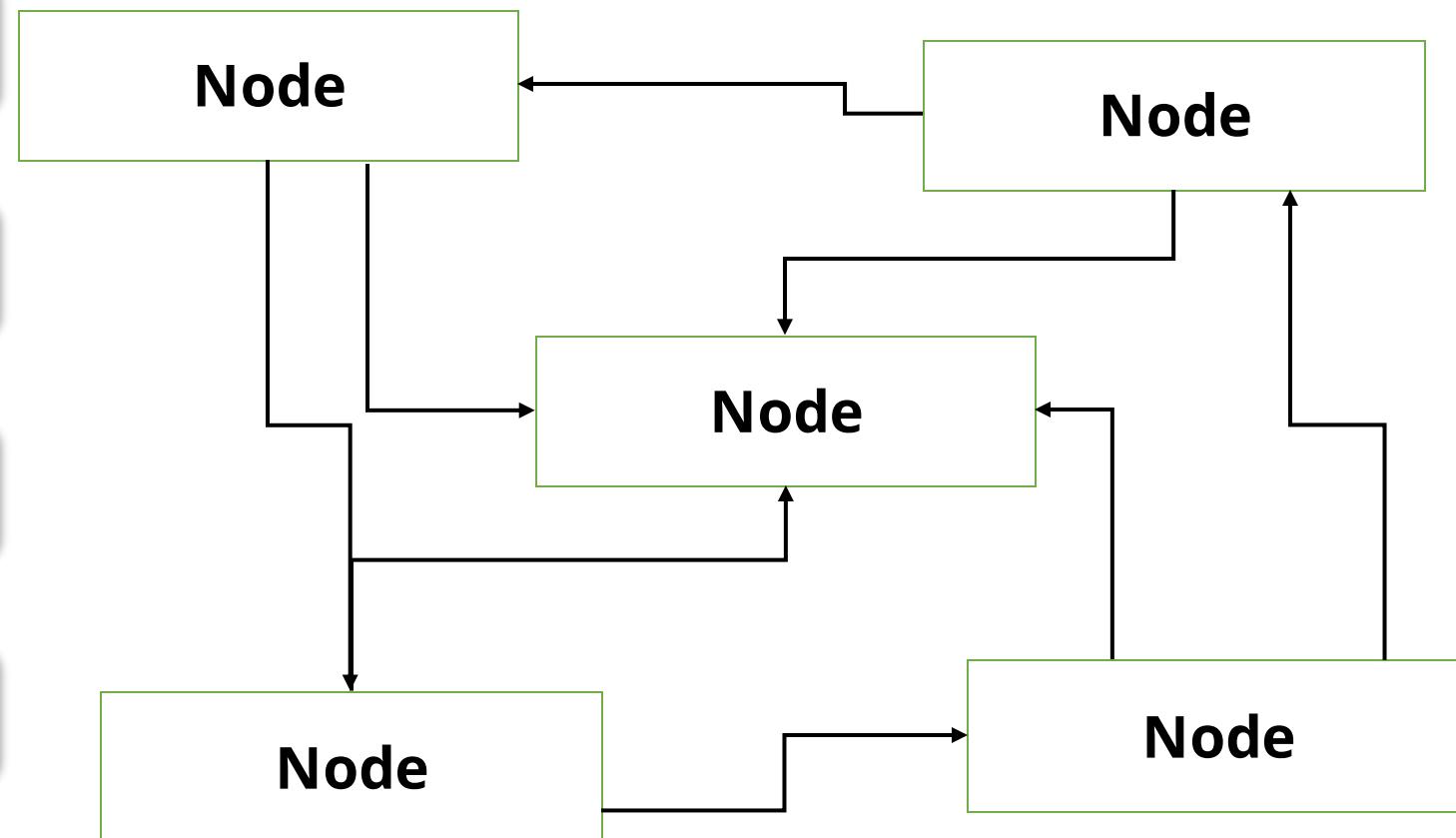
# Ethereum Networks

A node is a machine running as an Ethereum client

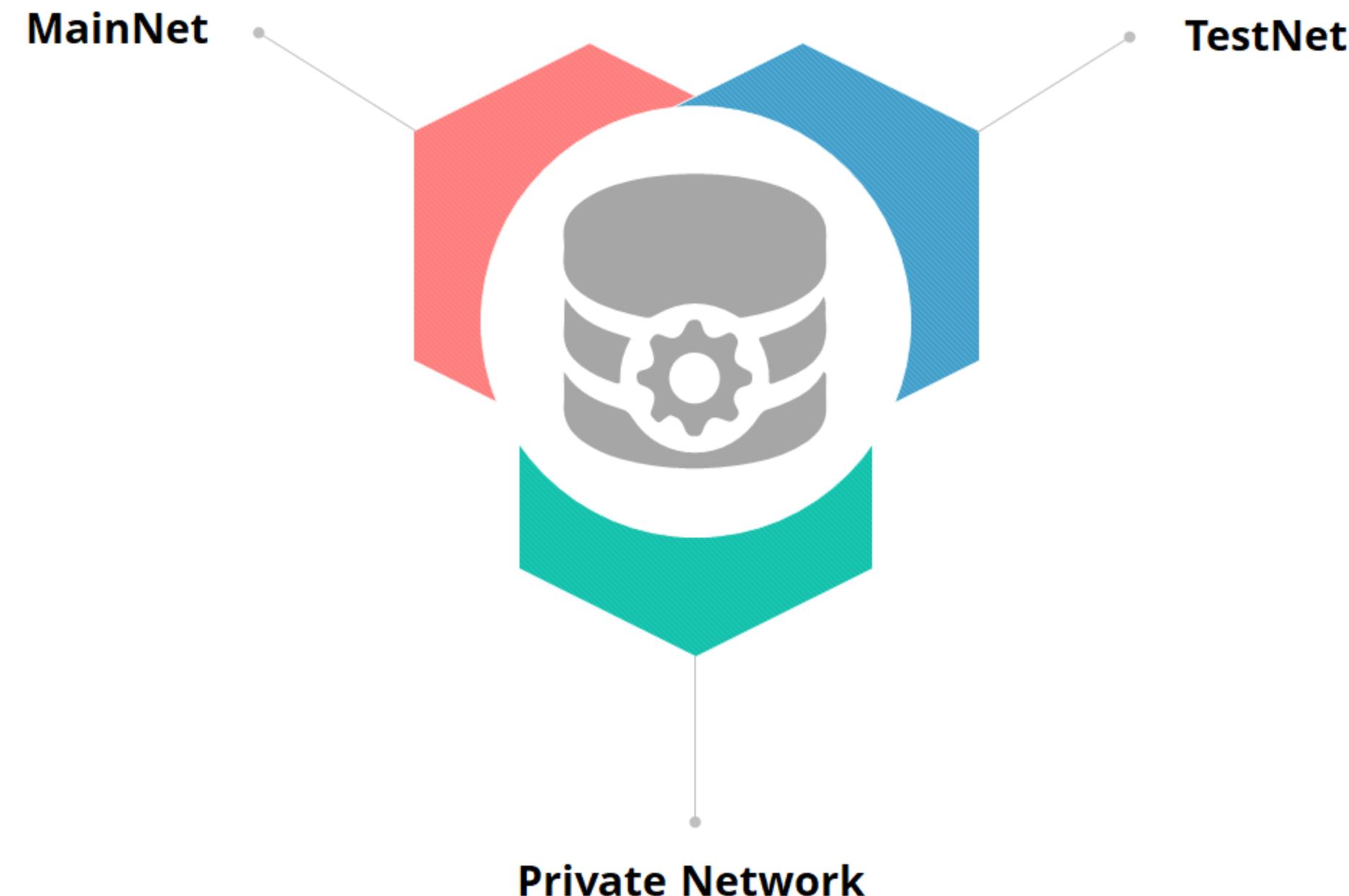
Used to store the data and transfer money

Formed by combining one or more nodes

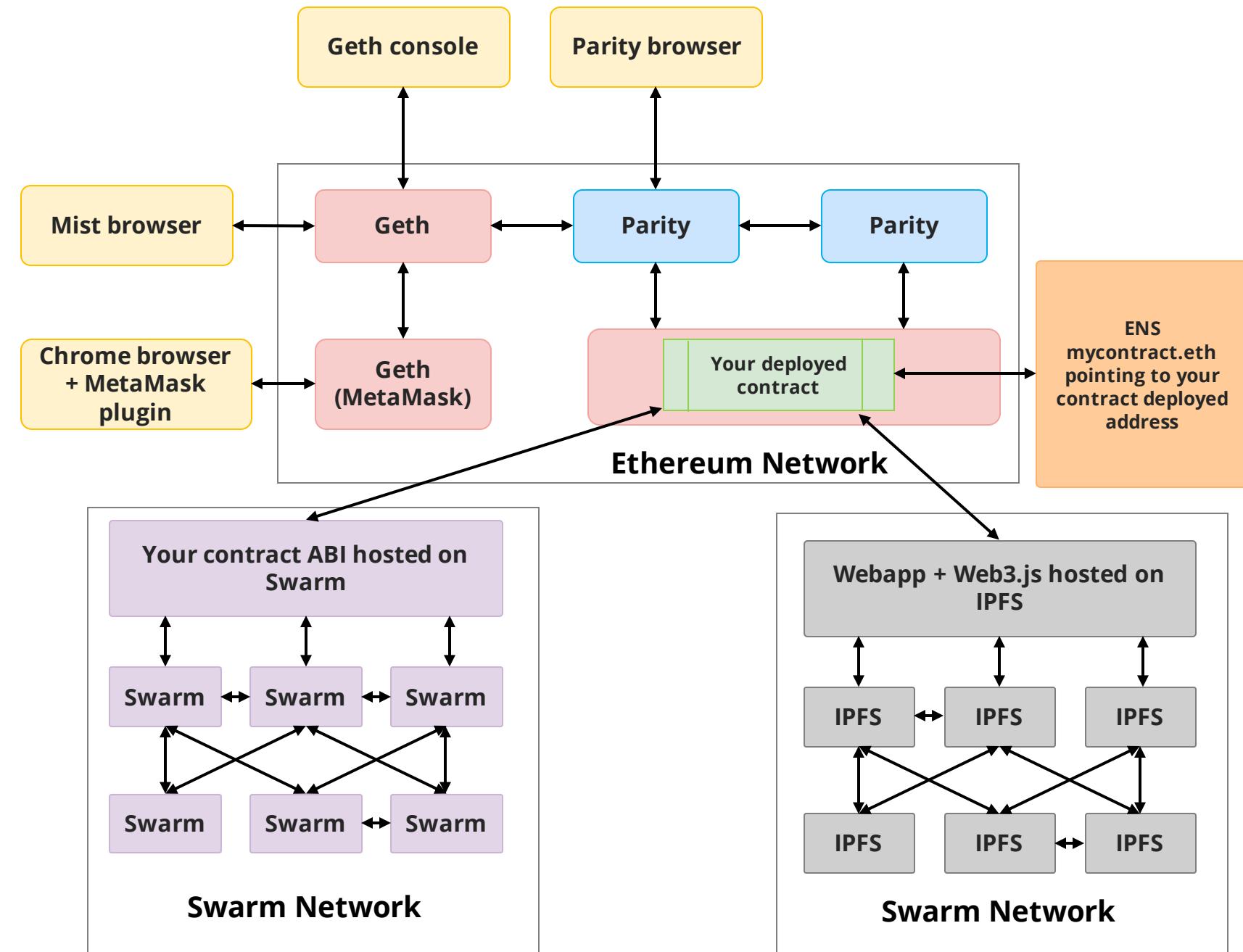
Each node contains a copy of the Blockchain



# Types of Network



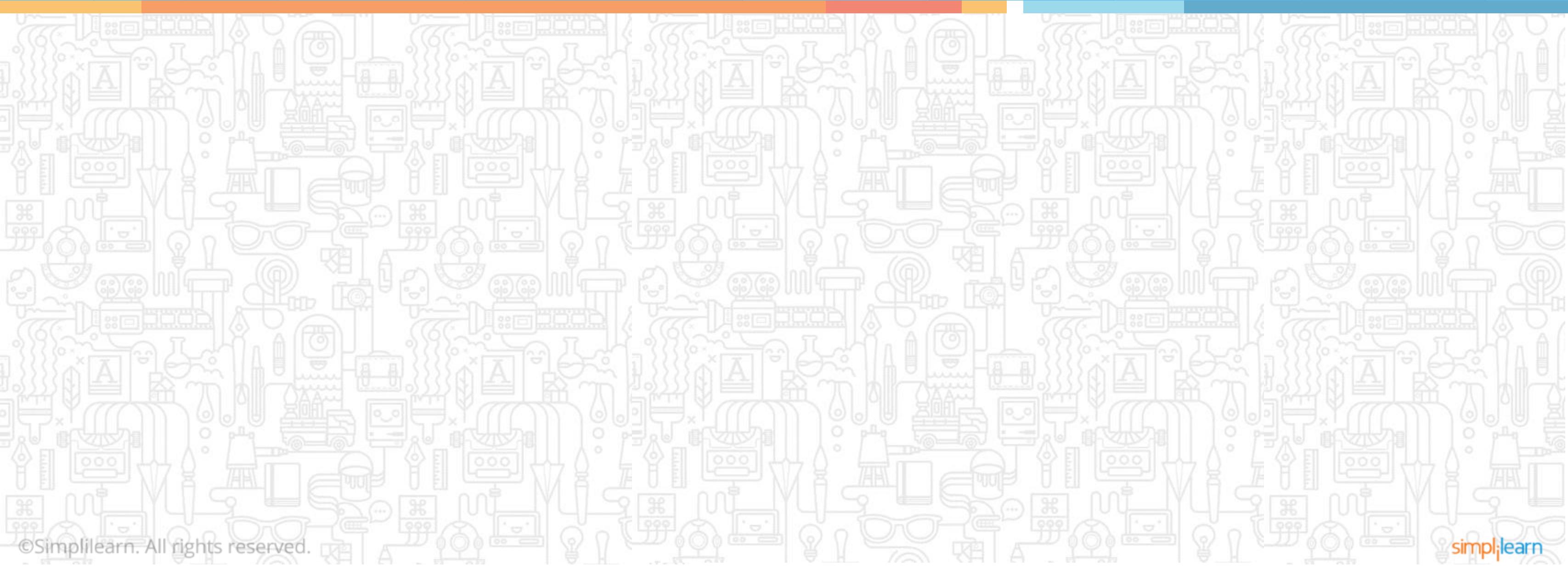
# Ethereum Ecosystem



# Ethereum Ecosystem

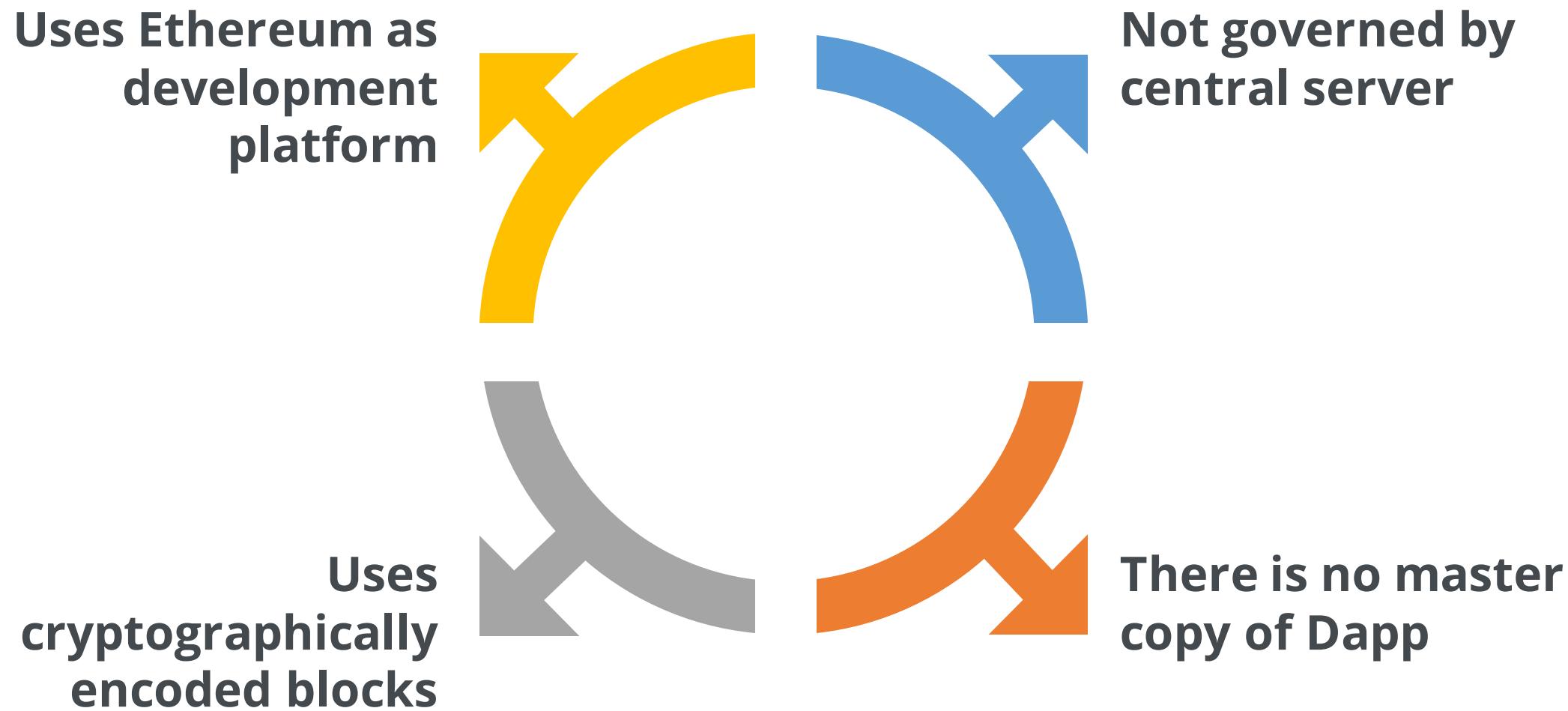
# Ethereum

## Ethereum DAPPs and DAOs



# Ethereum Decentralized Application(DAPP)

Decentralized application is an application that is used on networks with trustless protocols to avoid a single point of failure.

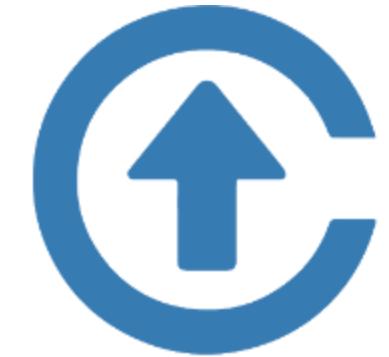


# Advantages of Ethereum DAPPs



## Examples of DAPPs

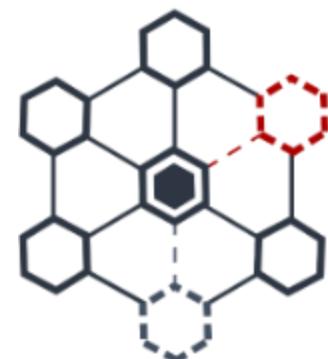
---



WeiFund



STORJ.IO



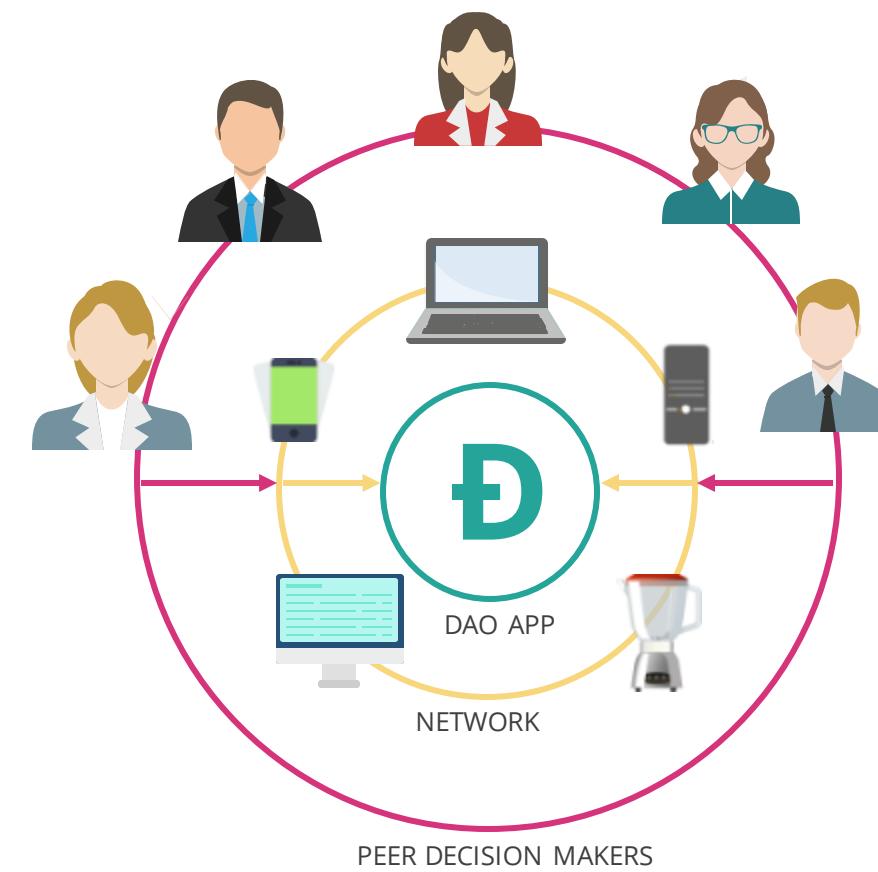
KYC-CHAIN



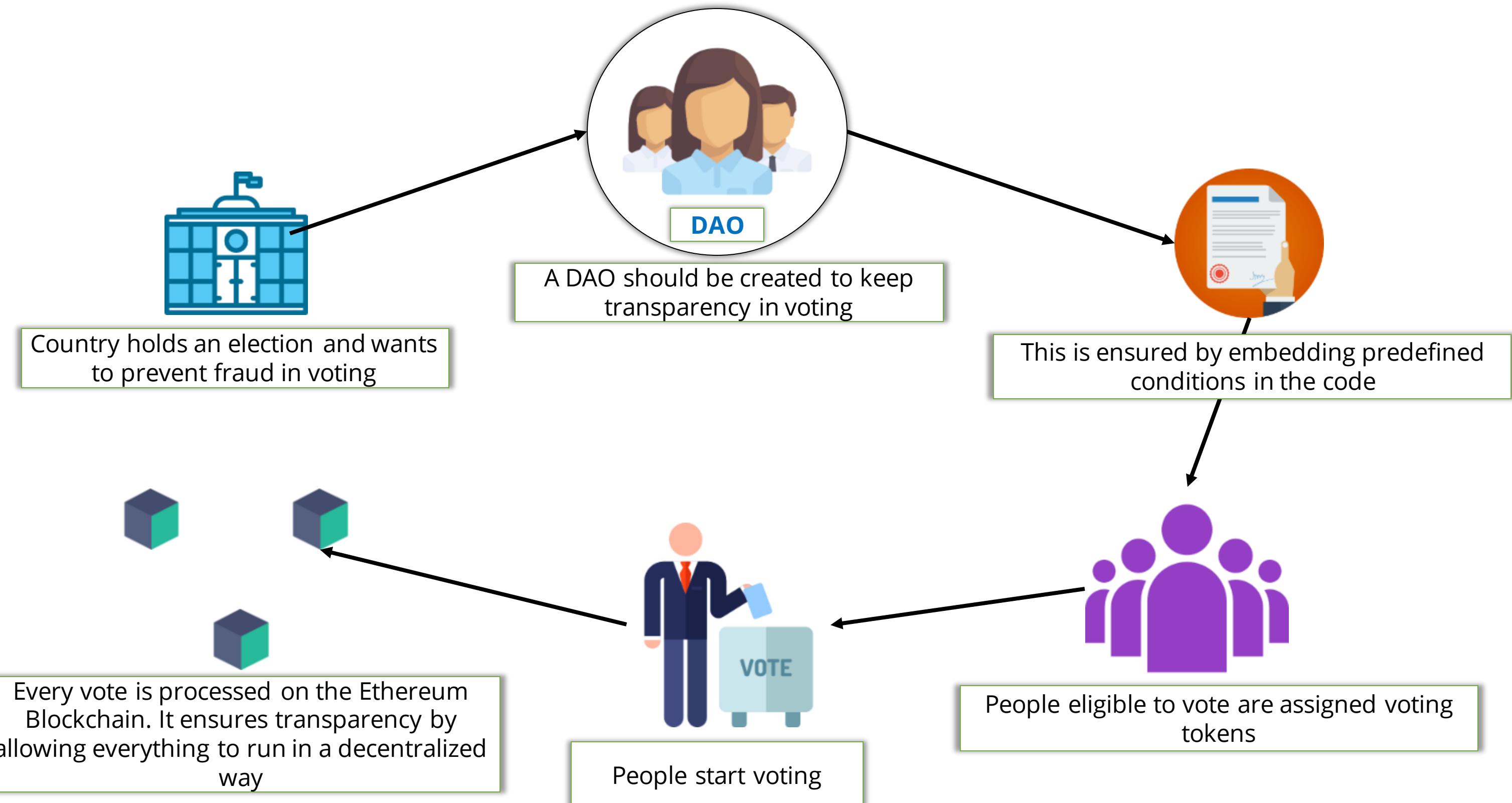
4G CAPITAL

# Decentralized Autonomous Organization(DAO)

DAOs are organizations designed to hold assets and use the voting system to distribute them. DAOs exist entirely on Blockchain and are governed by the consensus protocols.



# Process of DAO



# Key Takeaways



You are now able to:

- ➊ Describe Ethereum and its concepts
- ➋ Install of Geth and Ganache on your computer
- ➌ Use MetaMask and connect it with Ganache
- ➍ Install and use Mist Wallet



## Knowledge Check



Knowledge  
Check

1

## What is gas in Ethereum?

- a. The cost of generating a correct block hash
- b. A way of pricing transactions based on their computational complexity
- c. A measurement of how many nodes are attached to the network
- d. The amount of power the network requires to hash the transaction



## What is gas in Ethereum?

- a. The cost of generating a correct block hash
- b. A way of pricing transactions based on their computational complexity
- c. A measurement of how many nodes are attached to the network
- d. The amount of power the network requires to hash the transaction



The correct answer is

**b**

**Its price is expressed in ether, and the miner decides whether to refuse the transaction process or not based on the expected gas price.**

Knowledge  
Check  
2

**Which Ethereum account contains the set of rules for the nodes to agree upon so that they interact with each other?**

- a. Externally Owned Account
- b. Smart contracts



Knowledge  
Check  
2

**Which Ethereum account contains the set of rules for the nodes to agree upon so that they interact with each other?**

- a. Externally Owned Account
- b. Smart contracts



The correct answer is **b**

**Smart contract contains the set of rules and can turn legal obligations into automated processes.**

## Which development environment performs automatic deployment of contracts?

- a. Remix IDE
- b. Truffle
- c. Embark
- d. None of the above



## Which development environment performs automatic deployment of contracts?

- a. Remix IDE
- b. Truffle
- c. Embark
- d. None of the above



The correct answer is C

**Embark is a development environment that performs automatic deployment and also keeps track of deployed contracts.**

## Which Ethereum tool allows user to send and receive transactions using Google Chrome?

- a. Geth
- b. Ganache
- c. Swarm
- d. MetaMask



## Which Ethereum tool allows user to send and receive transactions using Google Chrome?

- a. Geth
- b. Ganache
- c. Swarm
- d. MetaMask



The correct answer is **d**

**MetaMask turns Google Chrome into a browser that allows the users to send and receive transactions and also to fetch the data from the Blockchain.**

# Lesson-End Project

Duration: 10 mins

## Transfer Bonus Using MetaMask

### Problem Statement:

You are an employee of Simplilearn and have been asked to transfer 100 Ethers from Simplilearn account to your personal account as a part of your performance bonus. You have to perform the following actions:

- Create two accounts, one Simplilearn account and the other one as your personal account. Connect both with Ganache
- Transfer 100 ETH from Simplilearn account to your account

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

Lesson 4: Deploying Smart Contracts on Private Ethereum Network



# Learning Objectives

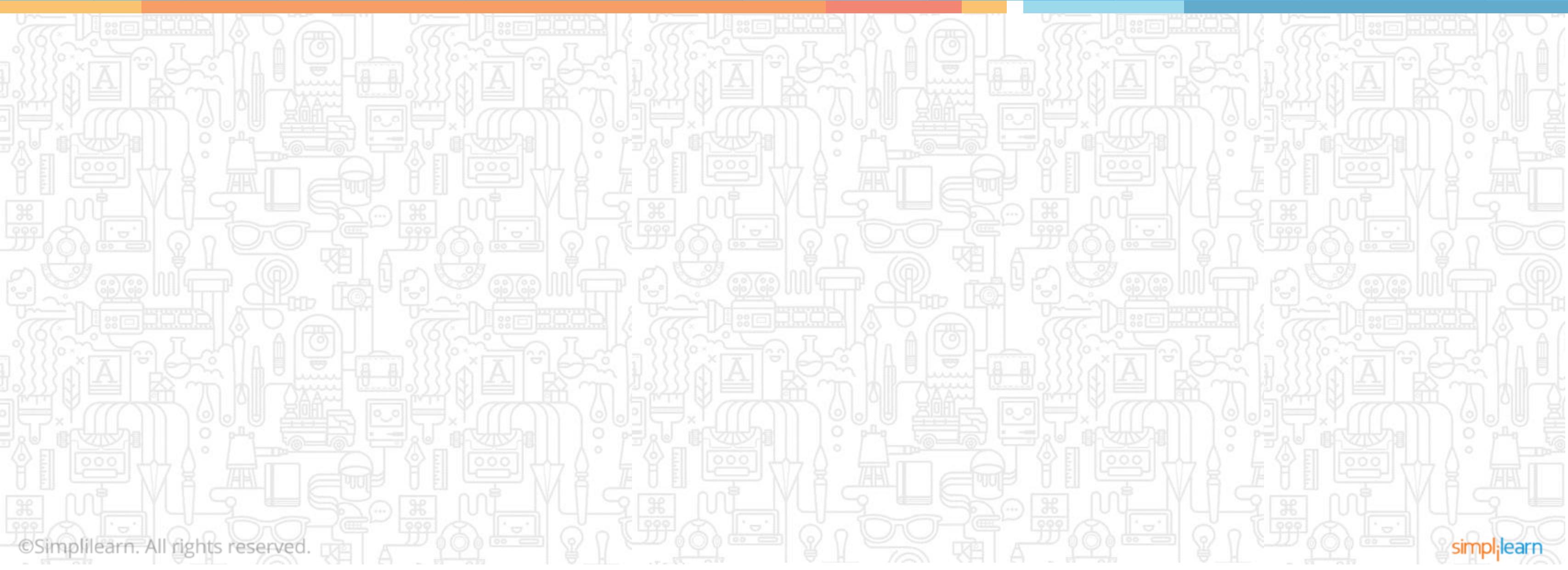


By the end of this lesson you should be able to:

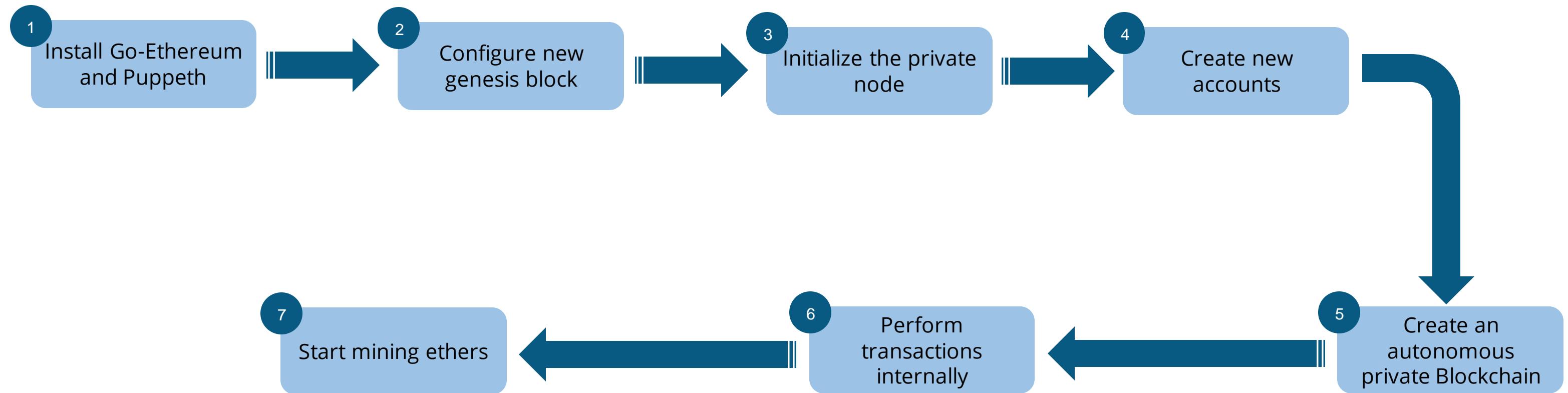
- ➊ Set up private Ethereum Blockchain
- ➋ Write Solidity programming codes
- ➌ Develop and deploy smart contracts on Ethereum test network
- ➍ Build an Ethereum based property transfer application
- ➎ Create a marketplace application on Ethereum network

# Deploying Smart Contracts on Private Ethereum Network

## Private Ethereum Blockchain



# Steps to Develop Private Blockchain



# Genesis Block

Genesis is the first block in the chain, and it is the only block without a predecessor.

```
{  
  "config": {  
    "chainId": 123,  
    "homesteadBlock": 0,  
    "eip155Block": 0,  
    "eip158Block": 0  
  },  
  "nonce": "0x3",  
  "timestamp": "0x0",  
  "parentHash":  
    "0x0000000000000000000000000000000000000000000000000000000000000000",  
  "extraData": "0x0",  
  "gasLimit": "0x4c4b40",  
  "difficulty": "0x400",  
  "mixhash":  
    "0x0000000000000000000000000000000000000000000000000000000000000000",  
  "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",  
  "alloc": {}  
}
```

Example of Genesis.json file

## Assisted Practice

### Private Ethereum Network

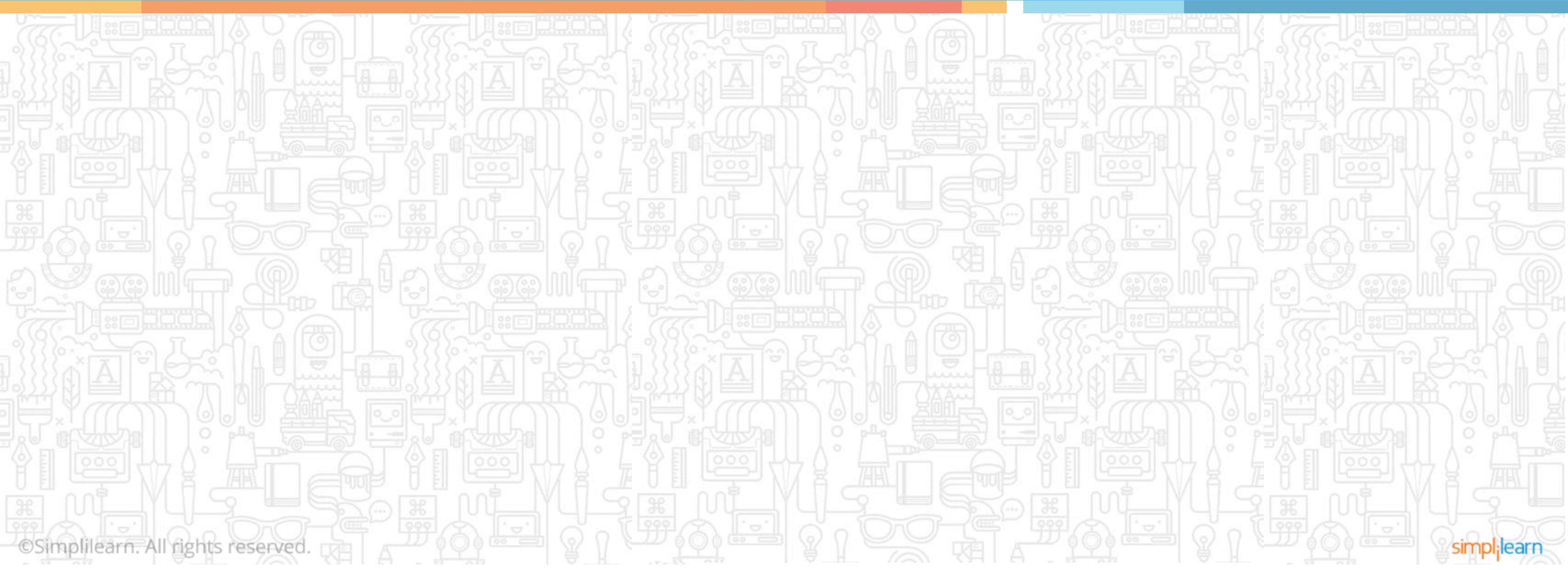
Duration: 15 mins

**Problem Statement:** Your organization has asked you to create a private Ethereum network developing and deploying decentralized business applications. Develop a private Ethereum Blockchain Network.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

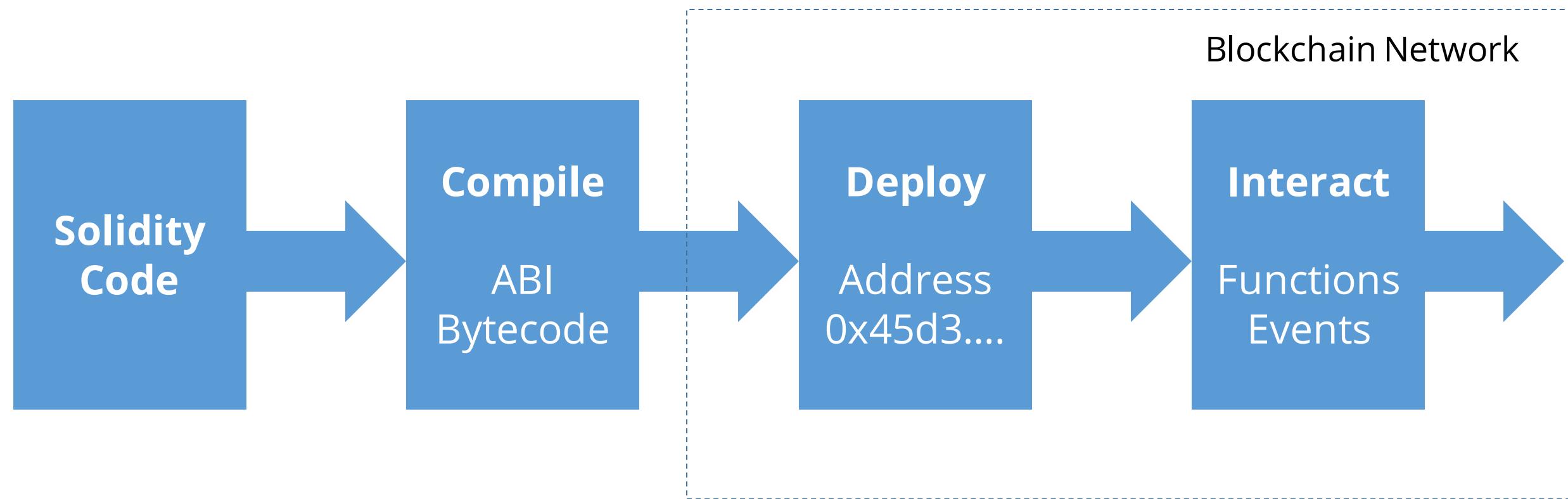
# Deploying Smart Contracts on Private Ethereum Network

## Ethereum Smart Contracts

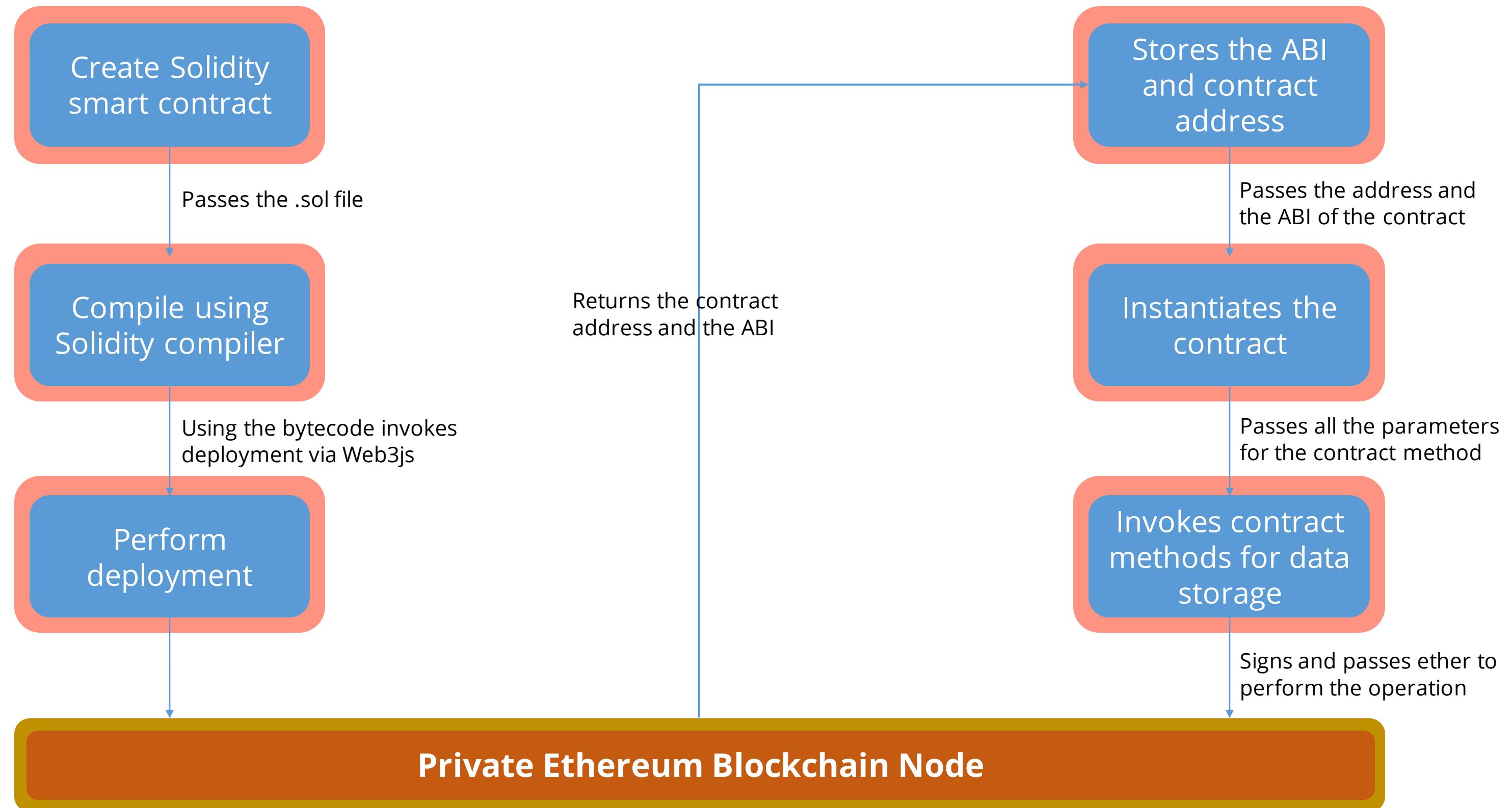


# Smart Contract Life Cycle

Solidity code is compiled to a bytecode and is deployed to the network of nodes. Deployed code can be interacted by calling functions using contract's address.



# Smart Contract Development



## Assisted Practice

### Smart Contract Development Environment

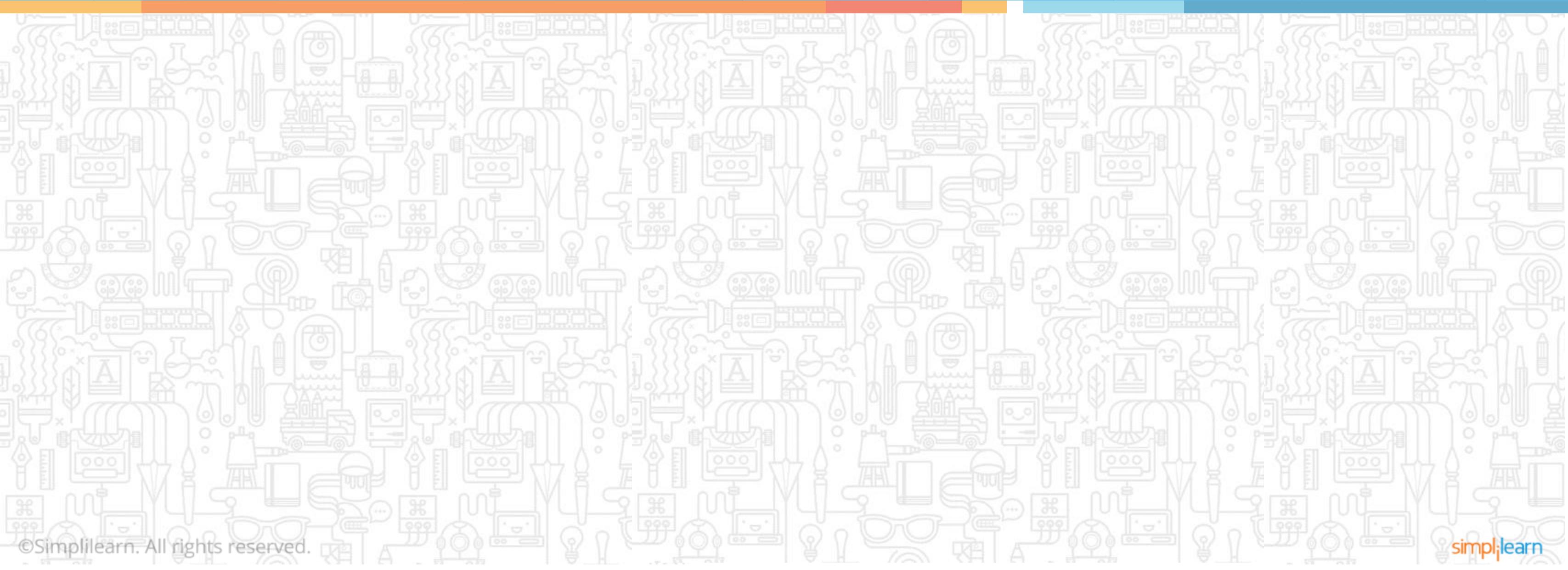
Duration: 10 mins

**Problem Statement:** Set up a development environment for developing and deploying Solidity smart contracts..

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

# Deploying Smart Contracts on Private Ethereum Network

## Solidity Programming



# Introduction to Solidity

---

- Solidity is an Ethereum smart contract language
- It is an object-oriented high-level language
- The syntax of Solidity is similar to JavaScript
- It is designed to enhance the Ethereum Virtual Machine (EVM)
- Solidity is a statically-typed scripting language where processes are verified at compile-time



# Comparison between Memory and Storage in Solidity

## Memory

- It is used to hold temporary values
- It is erased between external function calls and is cheaper to use

## Storage

- All the contract state variables reside in storage
- It is persistent between function calls and quite expensive to use

# Layout of a Solidity Smart Contract

---

Source files can contain contract definitions including directives and pragma directives.

```
pragma Solidity ^0.4.18 //version pragma declares the
version of the compiler to be used to avoid breaking
changes introduced on future versions
contract HelloWorld{

}
```

# General Value Types

General value types are always passed by values.

```
pragma Solidity ^0.4.18
contract ValueTypes{
    uint x; // uint is mostly used for currency value or amount or Unix timestamp
    int constant variable1 = 8; // int of 256 bit with constant value 8
    int256 constant variable2 = 8; // same effect as line above, here the 256 is
explicit
    uint constant VERSION_ID = 0x123A1; // A hexadecimal value stored in a constant
with           'constant', compiler replaces each occurrence with actual value
    uint8 b;
    int64 c;
    uint248 e;
    int x = int(b); // Type casting
    bool b = true; // or can also be declared by 'var b = true;' for inferred typing
    address public owner; // Addresses - holds 20 byte/160 bit Ethereum addresses
    byte a;
    bytes2 b; // Bytes available from 1 to 3. byte is same as bytes1
    bytes32 c;
    string n= "hello"; // strings are stored in UTF8 in double quotes
}
```

# Arrays



Arrays in Solidity can be of a fixed or dynamic size.

```
pragma Solidity ^0.4.0;

contract C {
    function f(uint len) {
        string[5] name; // array of fixed size 5 of type String
        uint[] dynamicArray // dynamic array of type uint
        uint[] memory a = new uint[](7); // allocating memory array
        bytes memory b = new bytes(len);
        // Here we have a.length == 7 and b.length == len
        a[6] = 8;
    }
}
```

## Arrays: Example Code

---

```
pragma Solidity ^0.4.0;

contract ArrayExample {
    bytes32[5] Names;
    bytes32[] examples;
    uint[] myArray;
    function test() constant returns (uint[]) {
        string[5] memory inlineArray = ["You", "can", "init", "like", "this"];
        uint[][] memory multidem; // multidimensional array
        myArray.push(123);
        examples.push("Matt");
        Names[0] = "Matt";
        return myArray;
    }
}
```

# Enums



Enums are one way to create a user-defined type in Solidity. They are explicitly convertible to and from all integer types, but implicit conversion is not allowed.

```
pragma Solidity ^0.4.0;

contract Purchase {
    enum State { Created, Locked, Inactive } // Enum
}
```

## Enums: Example Code

```
pragma Solidity ^0.4.0;
contract test {
    enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }
    ActionChoices choice;
    ActionChoices constant defaultChoice = ActionChoices.GoStraight;

    function setGoStraight() {
        choice = ActionChoices.GoStraight;
    }
    function getChoice() returns (ActionChoices) {
        return choice;
    }

    function getDefaultChoice() returns (uint) {
        return uint(defaultChoice);
    }
}
```

# Structs



Structs are custom defined types that can group several variables.

```
pragma Solidity ^0.4.18;
contract VotingContract {
    struct Voter { // Struct
        uint weight1, weight2, weight3;
        bool voted;
        address delegate1, delegate2, delegate3, delegate4;
        uint votel, vote2, vote3, vote4, vote5;
        uint height1, height2, height3 //structs can have upto 16
members. Exceeding the limit might result in error [stack too deep]
    } }
```

# Mapping

Mapping are declared as `Mapping( _Keytype => _ValueType )`

- `_KeyType` can be any type except for a dynamically sized array, a contract, an enum, and a struct
- `_ValueType` can be any type, including mappings

```
pragma Solidity ^0.4.18;
contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) {
        balances[msg.sender] = newBalance;
    }
}
contract MappingUser {
    function f() returns (uint) {
        MappingExample m = new MappingExample();
        m.update(100);
        return m.balances(this);
    }
}
```

## Mapping: Example Code

---

```
contract MyContract {  
    struct Data {  
        uint a;  
        uint b;  
    }  
    mapping (uint => Data) public items;  
    function MyContract() {  
        items[0] = Data(1,2);  
        items[1] = Data(3,4);  
        items[2] = Data(5,6);  
        delete items[1];  
    }  
}
```

# Function Declaration in Solidity

Examples of function declaration using pure and view keywords:

```
pragma Solidity ^0.4.16;
contract C { function f(uint a, uint b) public view returns (uint)
//a function declared as view cannot modify the state
{
    return a * (b + 42) + now;
}
}
```

```
pragma Solidity ^0.4.16;
contract C {
function f(uint a, uint b) public pure returns (uint)
{//a function declared as pure cannot read or modify the state
    return a * (b + 42);
}
}
```

# Function Calls and Return Types

```
pragma Solidity ^0.4.0;
contract FunctionCall {
// Constructor calls are also a function calls and are defined like this
function FunctionCall(uint param1) {
// Initialize state variables here
}
// you can create a contract object with a name & then use it inside the
function calls like this
Miner m;
function setMiner(address addr) {
m = Miner(addr); // type casted the addr to Miner type and stored in m
    }//function setMiner(Miner _m) { m = _m; } is also correct
// Now you can use the Miner's function which is info to sent
// some ether with optionally specifying the gas like this
function callMinerInfo() {
m.info.value(10).gas(800)();
}
```

## Function Calls and Return Types (Contd.)

```
function someFunction(uint key, uint value) {  
    // Do something }  
function demoFunction() {  
    // named arguments  
    someFunction({value: 2, key: 3}); }  
    //also note that variable names are optional in parameters & in returns  
function someFunction2(uint key, uint) returns (uint) {  
    return key; // Do something} }  
contract Miner{  
    //The modifier payable has to be used for info,  
    // because otherwise, we would not be able to  
    //send Ether to it in the call m.info.value(10).gas(800)()  
function info() payable returns (uint ret) {  
    return 42; } }
```

# Fallback Function

---

Fallback functions are unnamed functions. These are triggered when the function signature does not match with any of the available functions in Solidity contract.

```
pragma Solidity ^0.4.18;
contract FallbackExample {
    function() payable {
        a=5;
    }
}
```

# Function Modifiers

---

Modifiers enable you to control the behavior of your smart contract.

```
function sayHello() public view returns(string) {  
  
function multiply(uint a, uint b) private pure returns (uint) {  
    return a*b;  
}  
function increment(uint x, uint y) returns (uint x, uint y) {  
    x+=1;  
    y+=1;  
}  
//when a function returns multiple values we need to parallel assign  
(x1,y1) = increment(1,2);
```

# Function Modifiers: Example Code

```
pragma Solidity ^0.4.18;

contract FunctionModifiers{
    address public creator;
    function FunctionModifiers() {
        creator= msg.sender; }

Modifier onlyCreator() {
if(msg.sender!=creator) {
    throw;
}
_; // resumes the function execution wherever the access modifier is used
}
function killContract() onlyCreator{ //this function will not execute if
an exception occurs
    self-destruct(creator); } }
```

# Inheritance

---

Inheritance is similar to any other object-oriented programming language. The new contract inherits the functionality of the base class.

```
Contract A{
    uint a;
    function getA() public view returns(uint) {
        returns a;
    }
}

contract B is A {
    uint b;
    function getBsumA() public view returns(uint) {
        return b+a;
    }
}
```

## Inheritance: Example Code

---

```
pragma Solidity ^0.4.18;

contract fun {
    address person;
    function fInherit() {
        person = msg.sender;
    }
}
contract Solidity is fun{      // 'is' keyword is used for Inheritance
    function kill() {
        self-destruct(person); }
}

contract smartcontract is fun, Solidity //Multiple Inheritance
{
String public PersonName;
function smartcontract(String _name) {
UserName = _name;
}
}
```

# Abstract Contracts

---

Abstract contract cannot be compiled and can lack an implementation, but they can be used as a base contract.

```
pragma Solidity ^0.4.18;

contract Voter {
    function Ballot() returns (bytes32);
}

contract Person is Voter {
    function Ballot() returns (bytes32) {
        Return "yes";
    }
}
```

# Abstract Contracts: Example Code

---

```
pragma Solidity ^0.4.18;

contract abstractContract {
    function foo(uint i) returns(uint j);
}

contract newContract is abstractContract{
    function foo(uint i) returns(uint j) {
        j=3*i;
    }
}
```

# Events



Events allow the use of EVM logging for calling front-end callbacks inside the Dapps interface for the users to listen to the events.

```
contract Coin {  
    //Your smart contract properties...  
    // Sample event definition: use 'event' keyword and define the  
parameters  
    event Sent(address from, address to, uint amount);  
  
    function send(address receiver, uint amount) public {  
        //Some code for your intended logic...  
        //Call the event that will fire at browser (client-side)  
        Sent(msg.sender, receiver, amount);  
    }  
}
```

## Events: Example Code

---

```
pragma Solidity ^0.4.18;
event Voter(
    string name,
    uint age /* we have passed 2 types that represent name and age. When the
event is successfully returned in the UI, these values can be accessed */
);
function setVoter(string _vName, uint _age) public {
vName = _vName;
age = _age;
Voter(_vName, _age); //trigger event
}
```

# Creating Contracts Using New Operator

A contract can create a new contract using the new keyword.

```
Pragma Solidity 0.4.18
Contract SampleNew{

    uint x;
    function SampleNew(uint a) payable {
        x=a;
    }
}

contract New {
    SampleNew n= new SampleNew(4);

    function createdSample(uint arg) {
        SampleNew newD = new SampleNew(arg);
        SampleNew newD2 = (new).value(amount)(arg);
    }
}
```

# Globally Available Variables and Functions

Following are the variables and functions that always exist in the global namespace:

- `Block.blockhash(uint blockNumber)`: returns (bytes32): hash of the given block- only works for 256 most recent blocks excluding current
- `Block.coinbase(address)`: current block miner's address
- `Block.difficulty(uint)`: current block difficulty
- `Block.GasLimit(uint)`: current block GasLimit
- `Block.number(uint)`: current block number
- `Block.timestamp(uint)`: current block timestamp as seconds since unix epoch
- `msg.data(bytes)`: complete calldata
- `msg.sender(address)`: sender of the message call (current call)
- `msg.sig(bytes4)`: first four bytes of the message (current call)
- `msg.value(uint)`: number of wei sent with the message
- `now(uint)`: current block timestamp (alias for `block.timestamp`)
- `tx.gasprice(uint)`: gas price of the transaction
- `tx.origin(address)`: sender of the transaction (full call chain)

# ERC20 Token

ERC stands for Ethereum Requests for Comments, and 20 stands for a unique ID number to distinguish this standard from others

```
contract TokenContractFragment {  
mapping(address => uint256) balances;  
mapping(address => mapping (address =>  
uint256)) allowed;  
function balanceOf(address tokenOwner) public  
constant returns (uint balance) {  
    return balances[tokenOwner];  
}  
function transfer(address to, uint tokens)  
public returns (bool success) {  
    balances[msg.sender] =  
balances[msg.sender].sub(tokens);  
    balances[to] = balances[to].add(tokens);  
    Transfer(msg.sender, to, tokens);  
    return true;  
} //code continued in the second box
```

```
function transferFrom(address from, address to,  
uint tokens) public returns (bool success) {  
    balances[from] =  
balances[from].sub(tokens);  
    allowed[from][msg.sender] =  
allowed[from][msg.sender].sub(tokens);  
    balances[to] = balances[to].add(tokens);  
    Transfer(from, to, tokens);  
    return true;  
}  
function approve(address spender, uint tokens)  
public returns (bool success) {  
    allowed[msg.sender][spender] = tokens;  
    Approval(msg.sender, spender, tokens);  
    return true;  
}
```

# ERC20 Token: Standard Interface

```
contract ERC20Interface {  
    function totalSupply() public constant returns (uint);  
  
    function balanceOf(address tokenOwner) public constant returns (uint balance);  
  
    function allowance(address tokenOwner, address spender) public constant returns  
(uint remaining);  
  
    function transfer(address to, uint tokens) public returns (bool success);  
  
    function approve(address spender, uint tokens) public returns (bool success);  
  
    function transferFrom(address from, address to, uint tokens) public returns (bool  
success);  
  
    event Transfer(address indexed from, address indexed to, uint tokens);  
  
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);  
}
```

## Assisted Practice

Duration: 15 mins

### Create a Smart Contract to Issue Your Own Digital Token

**Problem Statement:** Working as a software professional in a financial organization, you are required to develop a tradable token with a fixed supply that can be utilized as a currency, share, or an asset. Develop a smart contract to design and issue your own digital token.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

# Smart Contract to Issue Your Own Cryptocurrency

```
pragma solidity ^0.4.16;
interface tokenRecipient { function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) external; }
contract TokenERC20 {
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint256 public totalSupply;
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
    event Burn(address indexed from, uint256 value);
    function TokenERC20(
        uint256 initialSupply,
        string tokenName,
        string tokenSymbol
    ) public {
        totalSupply = initialSupply * 10 ** uint256(decimals);
        balanceOf[msg.sender] = totalSupply;
        name = tokenName;
```

# Smart Contract to Issue Your Own Cryptocurrency(Contd.)

```
symbol = tokenSymbol;
}
function _transfer(address _from, address _to, uint _value) internal {
    require(_to != 0x0);
    require(balanceOf[_from] >= _value);
    require(balanceOf[_to] + _value >= balanceOf[_to]);
    uint previousBalances = balanceOf[_from] + balanceOf[_to];
    balanceOf[_from] -= _value;
    balanceOf[_to] += _value;
    emit Transfer(_from, _to, _value);
    assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
}
function transfer(address _to, uint256 _value) public returns (bool success) {
    _transfer(msg.sender, _to, _value);
    return true;
}
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success) {
    require(_value <= allowance[_from][msg.sender]);          // Check allowance
    allowance[_from][msg.sender] -= _value;
    _transfer(_from, _to, _value);
    return true;
}
```

# Smart Contract to Issue Your Own Cryptocurrency(Contd.)

```
function approve(address _spender, uint256 _value) public
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);
    return true;
}

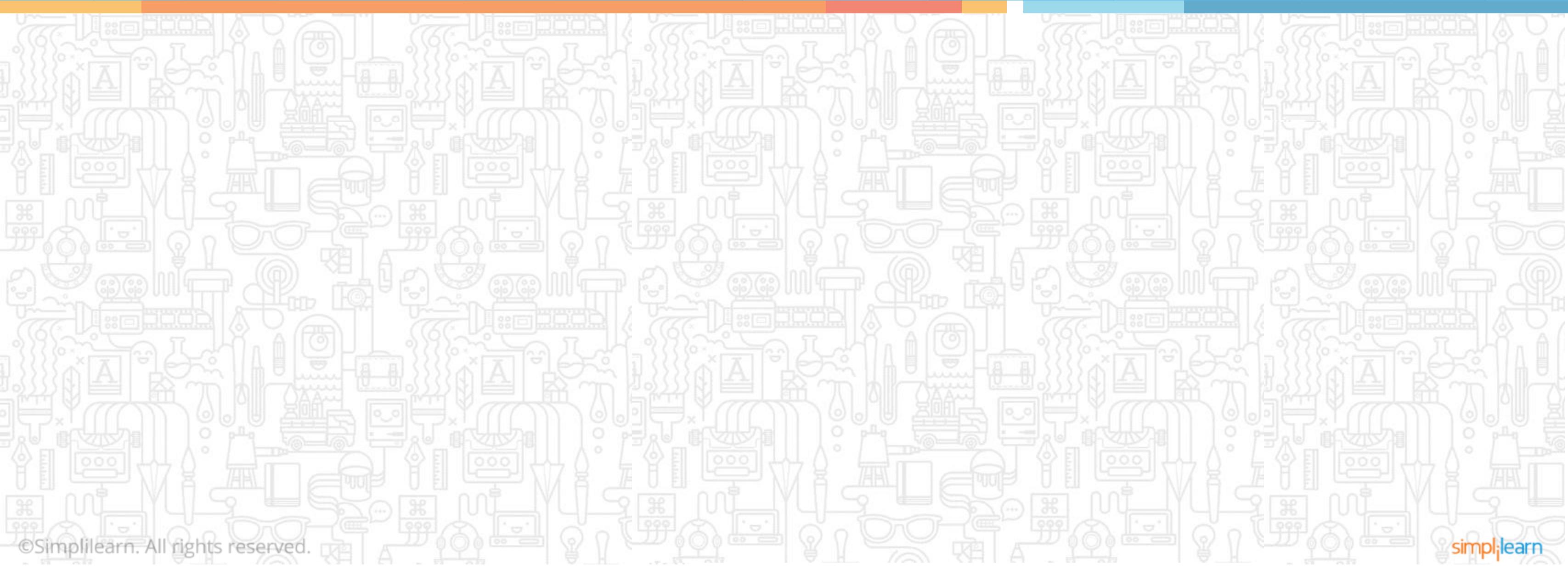
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
public
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
        return true;
    }
}
```

# Smart Contract to Issue Your Own Cryptocurrency(Contd.)

```
function burn(uint256 _value) public returns (bool success) {
    require(balanceOf[msg.sender] >= _value);
    balanceOf[msg.sender] -= _value;
    totalSupply -= _value;
    emit Burn(msg.sender, _value);
    return true;
}
function burnFrom(address _from, uint256 _value) public returns (bool success) {
    require(balanceOf[_from] >= _value);
    require(_value <= allowance[_from][msg.sender]);
    balanceOf[_from] -= _value;
    allowance[_from][msg.sender] -= _value;
    totalSupply -= _value;
    emit Burn(_from, _value);
    return true;
}
```

# Deploying Smart Contracts on Private Ethereum Network

## Solidity Smart Contract Design Patterns



# Important Smart Contract Design Notes

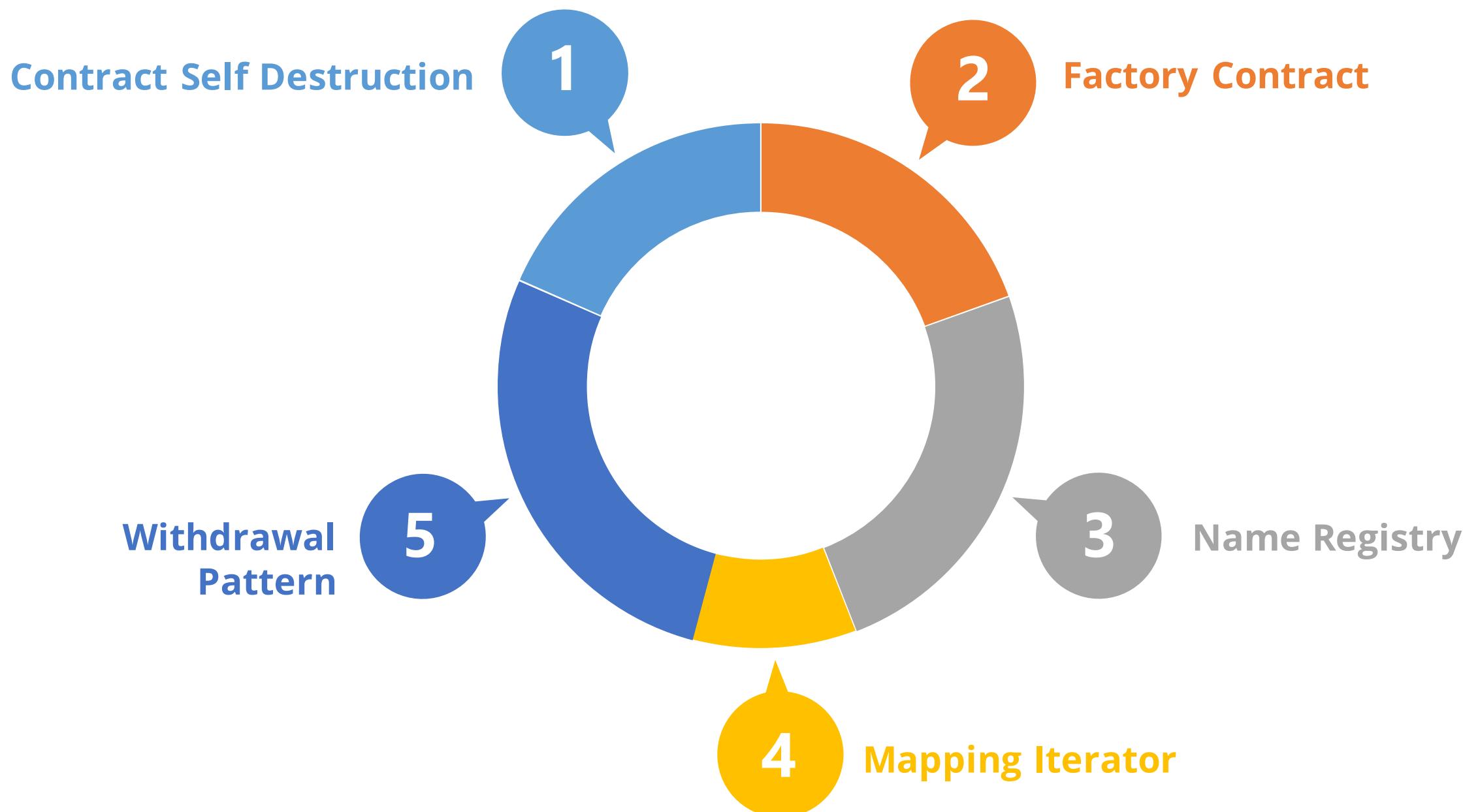
---

- **Obfuscation:** All variables are publicly viewable on Blockchain. So, anything that is private needs to be obfuscated
- **Storage optimization:** Smart contract should be carefully developed because writing to Blockchain is expensive as data is stored forever
- **Cron job:** Contracts must be manually called to handle time-based scheduling
- **Cost of Gas:** Almost every instruction in a smart contract costs gas, so always write a cost-effective smart contract

# Smart Contract Design Patterns

Design patterns are proven to be the go-to solution in addressing Ethereum smart contract design challenges.

Following are common design patterns:



# Smart Contract Design Patterns

Self destruction contract is used for terminating a contract, which means removing it forever from the Blockchain.

Self Destruction Contract

Factory Contract

Name Registry

Mapping Iterator

Withdrawal Pattern

```
contract SelfDesctructionContract {  
    public address owner;  
    public string someValue;  
    modifier ownerRestricted {  
        require(owner == msg.sender);  
        _;  
    }  
    // constructor  
    function SelfDesctructionContract() {  
        owner = msg.sender;  
    }  
    // a simple setter function  
    function setSomeValue(string value) {  
        someValue = value;  
    }  
    // you can call it anything you want  
    function destroyContract() ownerRestricted {  
        suicide(owner);  
    }  
}
```

# Smart Contract Design Patterns

Factory contracts are used to develop and deploy child contracts. The addresses of these child contracts are stored in the factory and can be extracted whenever necessary.

Self Destruction Contract

Factory Contract

Name Registry

Mapping Iterator

Withdrawal Pattern

```
contract CarShop {
    address[] carAssets;
    function createChildContract(string brand, string model) public payable {
        // insert check if the sent ether is enough to cover the car asset
        address newCarAsset = new CarAsset(brand, model, msg.sender);
        carAssets.push(newCarAsset);
    }
    function getDeployedChildContracts() public view returns (address[]) {
        return carAssets;
    }
}
contract CarAsset {string public brand;
    string public model;
    address public owner;
    function CarAsset(string _brand, string _model, address _owner) public {
        brand = _brand;
        model = _model;
        owner = _owner;
    } }
```

# Smart Contract Design Patterns

Name registry is useful when your contract is dependent on multiple contracts. By using name registry pattern, you can keep the address of one contract instead of all the addresses.

Self Destruction Contract

Factory Contract

Name Registry

Mapping Iterator

Withdrawal Pattern

```
contract NameRegistry {
    struct ContractDetails {
        address owner;
        address contractAddress;
        uint16 version;
    }
    mapping(string => ContractDetails) registry;
    function registerName(string name, address addr, uint16 ver) returns(bool) {
        // versions should start from 1
        require(ver >= 1);

        ContractDetails memory info = registry[name];
        require(info.owner == msg.sender);
        // create info if it doesn't exist in the registry
        if (info.contractAddress == address(0)) {
            info = ContractDetails({
                owner: msg.sender,
                contractAddress: addr,
                version: ver
            });
        }
    }
}
```

# Smart Contract Design Patterns

Self Destruction Contract

Factory Contract

Name Registry

Mapping Iterator

Withdrawal Pattern

```
else {
    info.version = ver;
    info.contractAddress = addr;
}
// update record in the registry
registry[name] = info;
return true;
}

function getContractDetails(string name) constant returns(address,
uint16) {
    return (registry[name].contractAddress, registry[name].version);
}
```

# Smart Contract Design Patterns

Mappings cannot be iterated, but it is possible to implement a data structure with mappings.

```
contract MappingIterator {
    mapping(string => address) elements;
    string[] keys;
    function put(string key, address addr) returns (bool) {
        bool exists = elements[key] != address(0)
        if (!exists) {
            keys.push(key);
        }
        elements[key] = addr;
        return true;
    }
    function getKeyCount() constant returns (uint) {
        return keys.length;
    }
    function getElementAtIndex(uint index) returns (address)
    {
        return elements[keys[index]];
    }
    function getElement(string name) returns (address) {
        return elements[name];
    }
}
```

Self Destruction Contract

Factory Contract

Name Registry

Mapping Iterator

Withdrawal Pattern

# Smart Contract Design Patterns

The goal of withdrawal pattern is to keep track of balances internally and force each user to withdraw their funds immediately.

Self Destruction Contract

Factory Contract

Name Registry

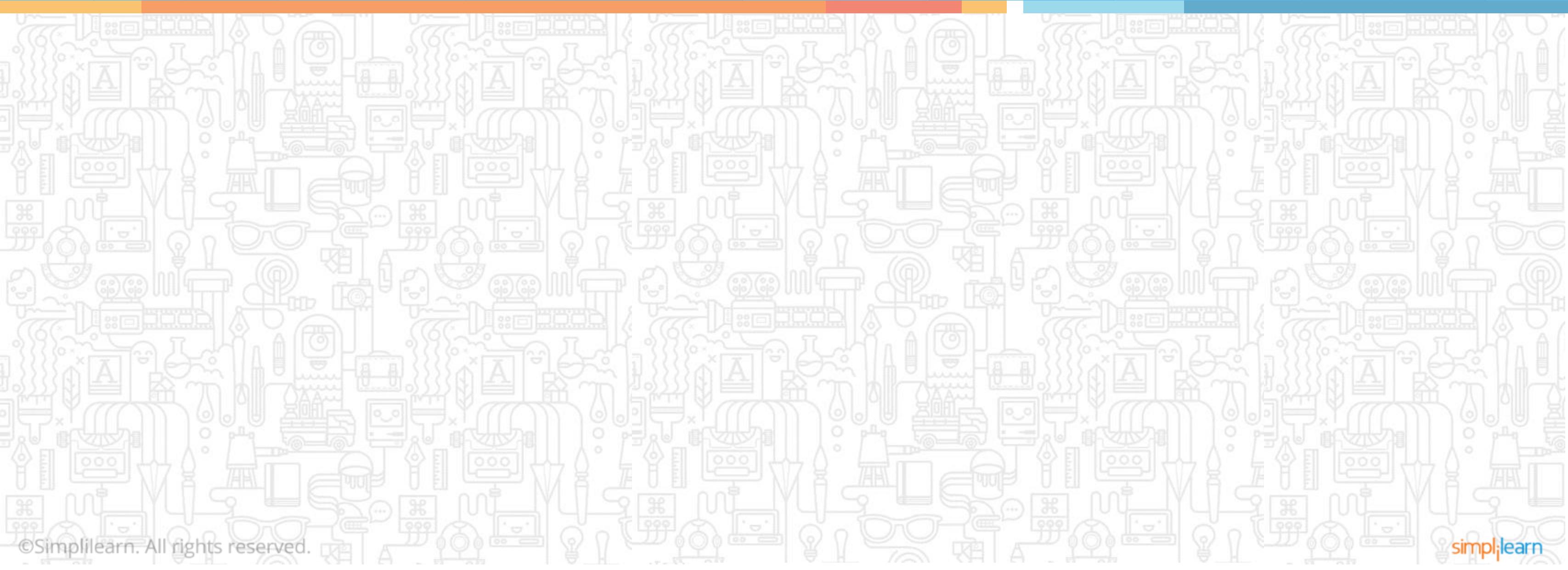
Mapping Iterator

Withdrawal Pattern

```
contract WithdrawalContract {  
    mapping(address => uint) buyers;  
    function buy() payable {  
        require(msg.value > 0);  
        buyers[msg.sender] = msg.value;  
    }  
    function withdraw() {  
        uint amount = buyers[msg.sender];  
        require(amount > 0);  
        buyers[msg.sender] = 0;  
        require(msg.sender.send(amount));  
    }  
}
```

# Deploying Smart Contracts on Private Ethereum Network

## Solidity Smart Contract Use Cases



## Assisted Practice

Duration: 20 mins

### Solidity Smart Contract to Develop Property Transfer System

**Problem Statement:** You are given a project to remove the third-party dependency for transferring the ownership of a property from one individual to another. Create a smart contract for decentralized property transfer system.

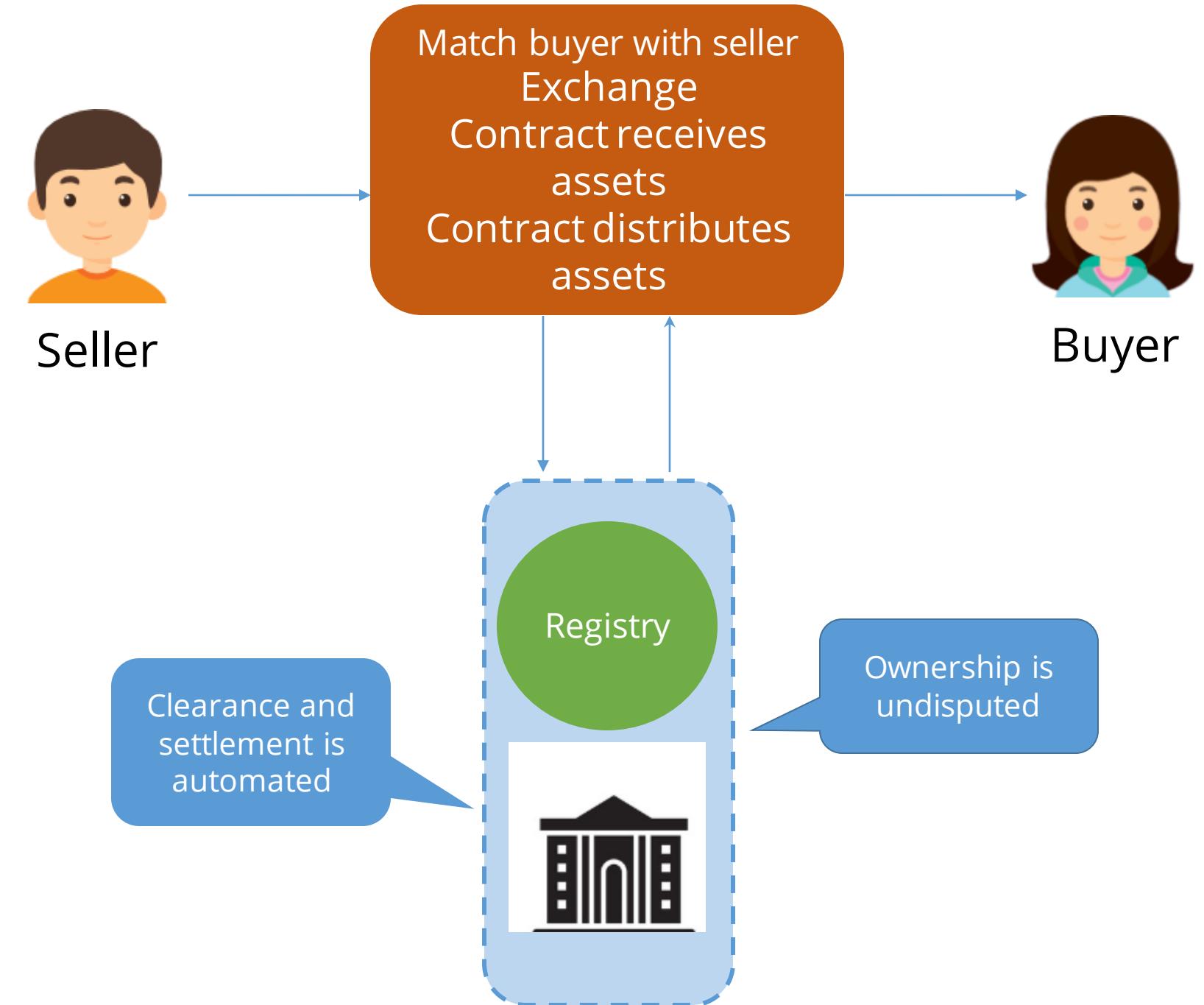
**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

# Assisted Practice: Property Transfer

**Possible Solution:** Blockchain can be used for accelerating the process of property transfer.

A digital record with permanent audit trail can be instituted, which allows the government to easily access the chain of data corresponding to that property.

This creates transparency in the system and reduces frauds.



# Assisted Practice: Guidelines For Property Transfer

---

You need to build a smart contract which is capable of handling property transfer with below cases:

1. Insert some dummy properties to replicate the real world. These properties shall have basic characteristics such as address, location, floors
2. Check the ownership of property before transferring it to the other owner
3. Include as many general parameters as possible to keep the smart contract almost replicating the real-world transaction
4. After the successful transfer of the property, you need to ensure that the old owner is not able to send the property again
5. Finally, the new owner must be able to send the freshly received property to any owner(address)

# Property Transfer: Smart Contract

```
pragma solidity ^0.4.18;
contract PropertyTransfer {
address public DA;
    uint256 public totalNoOfProperty
function PropertyTransfer() {
    DA = msg.sender;
}
modifier onlyOwner() {
    require(msg.sender == DA);
    _;
}
struct Property{
    string name;
    bool isSold;
}
mapping(address => mapping(uint256=>Property)) public propertiesOwner;
mapping(address => uint256) individualCountOfPropertyPerOwner;
event PropertyAlloted(address indexed _verifiedOwner, uint256 indexed _totalNoOfPropertyCurrently,
string _nameOfProperty, string _msg);
event PropertyTransferred(address indexed _from, address indexed _to, string _propertyName, string _msg);
function getPropertyCountOfAnyAddress(address _ownerAddress) constant returns (uint256) {
    uint count=0;
```

# Property Transfer: Smart Contract(Contd.)

```
for(uint i =0; i<individualCountOfPropertyPerOwner[_ownerAddress];i++) {
    if(propertiesOwner[_ownerAddress][i].isSold != true)
        count++;
}
return count;
}
function allotProperty(address _verifiedOwner, string _propertyName)
onlyOwner
{
    propertiesOwner[_verifiedOwner][individualCountOfPropertyPerOwner[_verifiedOwner]++].name =
    _propertyName;
    totalNoOfProperty++;
    PropertyAlloted(_verifiedOwner,individualCountOfPropertyPerOwner[_verifiedOwner], _propertyName,
    "property allotted successfully");
}
function isOwner(address _checkOwnerAddress, string _propertyName) constant returns (uint) {
    uint i ;
    bool flag ;
for(i=0 ; i<individualCountOfPropertyPerOwner[_checkOwnerAddress]; i++) {
if(propertiesOwner[_checkOwnerAddress][i].isSold == true) {
    break;
}
```

# Property Transfer: Smart Contract(Contd.)

```
flag = stringsEqual(propertiesOwner[_checkOwnerAddress][i].name,_propertyName);
if(flag == true) {
    break;
}
if(flag == true) {
    return i;
}
else {
    return 99999999;
} }
function stringsEqual (string a1, string a2) constant returns (bool) {
return sha3(a1) == sha3(a2)? true:false;
}
function transferProperty (address _to, string _propertyName)
returns (bool , uint )
{
uint256 checkOwner = isOwner(msg.sender, _propertyName);
bool flag;
```

# Property Transfer: Smart Contract(Contd.)

```
if(checkOwner != 99999999 && propertiesOwner[msg.sender][checkOwner].isSold == false){  
    propertiesOwner[msg.sender][checkOwner].isSold = true;  
    propertiesOwner[msg.sender][checkOwner].name = "Sold";  
    propertiesOwner[_to][individualCountOfPropertyPerOwner[_to]++].name = _propertyName;  
    flag = true;  
    PropertyTransferred(msg.sender , _to, _propertyName, "Owner has been changed." );  
}  
else {  
    flag = false;  
    PropertyTransferred(msg.sender , _to, _propertyName, "Owner doesn't own the property." );  
return (flag, checkOwner);  
} }
```

# Unassisted Practice

## Decentralized Marketplace Application

Duration: 20 mins

**Problem Statement:** Centralized marketplace has issues about trusting the transacting participants. People tend to trust sellers with reputation in the market, which makes it hard for the new sellers to do business. Develop a smart contract for decentralized marketplace application.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

**Note:** This practice is not graded. It is only intended for you to apply the knowledge you have gained to solve real-world problems.

# Unassisted Practice: Simple Marketplace Guidelines

The simple marketplace application expresses a workflow for a transaction between an owner and a buyer.

## Application Roles

**Owner:** A person who wants to sell in the marketplace.

**Buyer:** A person who wants to buy from the marketplace.

## States

**Item Available:** Indicates that an owner has made the item they want to sell available in the marketplace.

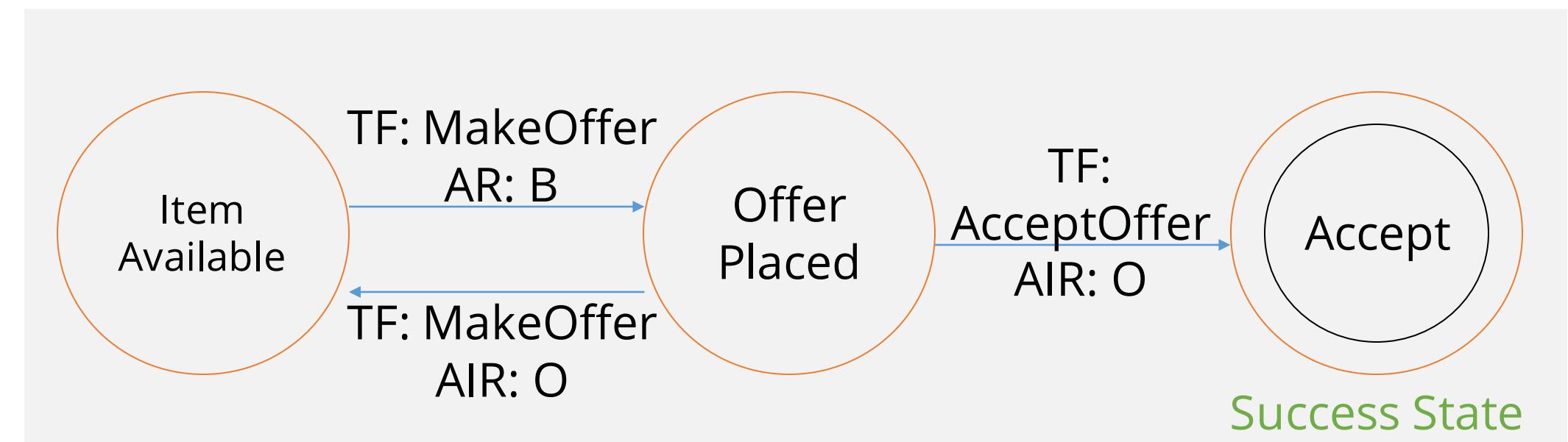
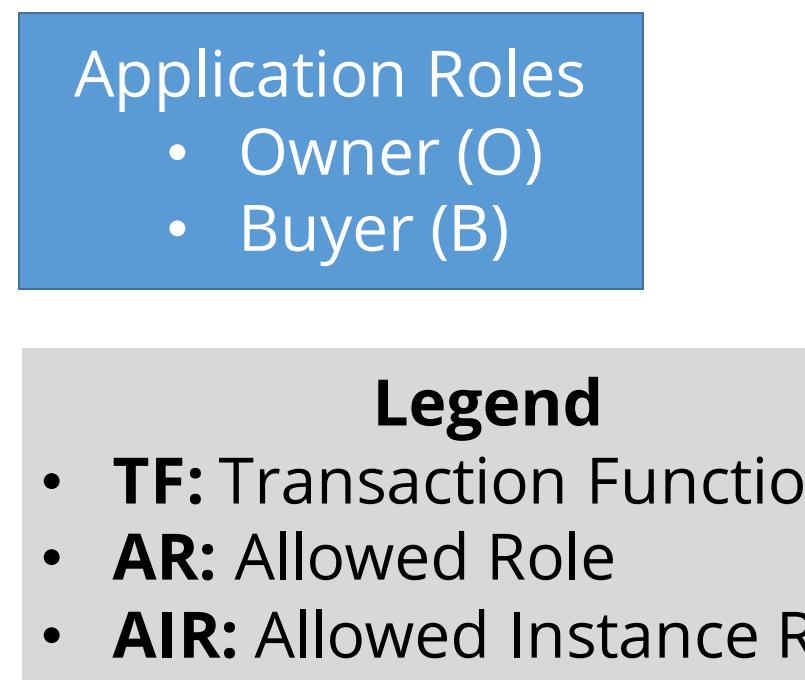
**Offer Places:** Indicates that a seller has made an offer to buy the item listed by an owner.

**Accepted:** Indicates that the owner has accepted the buyer's offer for the item.

# Unassisted Practice: Simple Marketplace Guidelines(Contd.)

## Workflow details

- Workflow starts at Item Available when an owner makes an item available for sale
- Buyer can then make an offer. This causes the state change Offer Placed
- If owner agrees to buyer's offer then owner accepts the offer and the workflow reaches successful state
- If the owner rejects the offer state changes to itemAvailable indicating that the item is still up for sale
- The transition between the ItemAvailable and the OfferPlaced states can continue until the owner is satisfied with the offer made



The state transition diagram below shows the interactions among the states in this workflow

# Steps for Creating a Simple Marketplace

---

**Step 1:** Open remix IDE using the browser

**Step 2:** Create a new file in the IDE and write the solidity smart contract

**Step 3:** Run the following command in a terminal to start ganache for testing and deploying your smart contract:

`ganache-cli`

**Step 4:** Now you need to start block explorer. To start block explorer, get into the directory where it is installed and run the following command: `npm start`

**Step 5:** Click on the run tab in the remix IDE to set the environment to web3 provider. A drop-down dialog box will appear, click ok.

**Step 6:** Click deploy button to deploy the contract on to your test Blockchain

**Step 7:** Check for the contract deployment in `ganache-cli` terminal

**Step 8:** Interact with the contract using the contract functions

# A Simple Marketplace: Smart Contract

## Code for step 2

```
pragma solidity ^0.4.20;
contract WorkbenchBase {
    event WorkbenchContractCreated(string applicationName, string workflowName, address originatingAddress);
    event WorkbenchContractUpdated(string applicationName, string workflowName, string action, address originatingAddress);
    string internal ApplicationName;
    string internal WorkflowName;

    function WorkbenchBase(string applicationName, string workflowName) internal {
        ApplicationName = applicationName;
        WorkflowName = workflowName;
    }
    function ContractCreated() internal {
        WorkbenchContractCreated(ApplicationName, WorkflowName, msg.sender);
    }
    function ContractUpdated(string action) internal {
        WorkbenchContractUpdated(ApplicationName, WorkflowName, action, msg.sender);
    }
}
```

# A Simple Marketplace: Smart Contract(Contd.)

## Code for step 2

```
contract SimpleMarketplace is WorkbenchBase('SimpleMarketplace', 'SimpleMarketplace')
{
    enum StateType {
        ItemAvailable,
        OfferPlaced,
        Accepted
    }
    address public InstanceOwner;
    string public Description;
    int public AskingPrice;
    StateType public State;
    address public InstanceBuyer;
    int public OfferPrice;
    function SimpleMarketplace(string description, int price) public
    {
        InstanceOwner = msg.sender;
        AskingPrice = price;
        Description = description;
        State = StateType.ItemAvailable;
        ContractCreated();
    }
}
```

# A Simple Marketplace: Smart Contract(Contd.)

## Code for step 2

```
function MakeOffer(int offerPrice) public
{
    if (offerPrice == 0)
    {
        revert();
    }

    if (State != StateType.ItemAvailable)
    {
        revert();
    }
    if (InstanceOwner == msg.sender)
    {
        revert();
    }
    InstanceBuyer = msg.sender;
    OfferPrice = offerPrice;
    State = StateType.OfferPlaced;
    ContractUpdated('MakeOffer');
}
```

# A Simple Marketplace: Smart Contract(Contd.)

## Code for step 2

```
function Reject() public
{
    if ( State != StateType.OfferPlaced )
    {
        revert();
    }
    if (InstanceOwner != msg.sender)
    {
        revert();
    }
    InstanceBuyer = 0x0;
    State = StateType.ItemAvailable;
    ContractUpdated('Reject');
}
function AcceptOffer() public
{
    if ( msg.sender != InstanceOwner )
    {
        revert();
    }
    State = StateType.Accepted;
    ContractUpdated('AcceptOffer');
}
```

# Key Takeaways



You are now able to:

- ✔ Set up private Ethereum Blockchain
- ✔ Write Solidity programming codes
- ✔ Develop and Deploy smart contracts on Ethereum test network
- ✔ Build an Ethereum based property transfer application
- ✔ Create a marketplace application on Ethereum network



## Knowledge Check



Knowledge  
Check

1

**Which of the following is not a part of genesis.json file?**

- a. Gas
- b. GasLimit
- c. Nonce
- d. Mixhash



Knowledge  
Check

1

**Which of the following is not a part of genesis.json file?**

- a. Gas
- b. GasLimit
- c. Nonce
- d. Mixhash



The correct answer is

**a**

**Gas is calculated at the time of transaction. It is not a part of genesis file.**

Knowledge  
Check  
2

**Smart contracts hold the potential not only to execute automated processes but also to restrict behavior.**

- a. True
- b. False



Knowledge  
Check  
2

**Smart contracts hold the potential not only to execute automated processes but also to restrict behavior.**

- a. True
- b. False



The correct answer is

**a**

**Smart contract can restrict behavior using function modifiers.**

Knowledge  
Check

3

**Which of these programming languages is used for writing Ethereum smart contracts?**

- a. Python
- b. Serpent
- c. JavaScript
- d. C++



## Which of these programming languages is used for writing Ethereum smart contracts?

- a. Python
- b. Serpent
- c. JavaScript
- d. C++



The correct answer is **b**

**Serpent is used to write Ethereum smart contracts.**

**Which of these enables you to hold and secure ether and other crypto-assets built on Ethereum as well as write, deploy, and use smart contracts?**

- a. Geth
- b. Web3.js
- c. Truffle
- d. Mist Wallet



**Which of these enables you to hold and secure ether and other crypto-assets built on Ethereum as well as write, deploy, and use smart contracts?**

- a. Geth
- b. Web3.js
- c. Truffle
- d. Mist Wallet



The correct answer is **d**

**Mist wallet is a browser used to write, deploy, and use smart contract. It is also used to store Ethereum-based crypto-assets.**

# Lesson-End Project

## Smart Contract for Banking Application

Duration: 60 mins

### Problem Statement:

Write a simple bank smart contract in solidity that allows users to do the following:

- Deposit money into their account
- Withdraw money from their account
- Check balance

After a contract is created, deploy the contract on Ropsten network.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 5: Hyperledger



# Learning Objectives

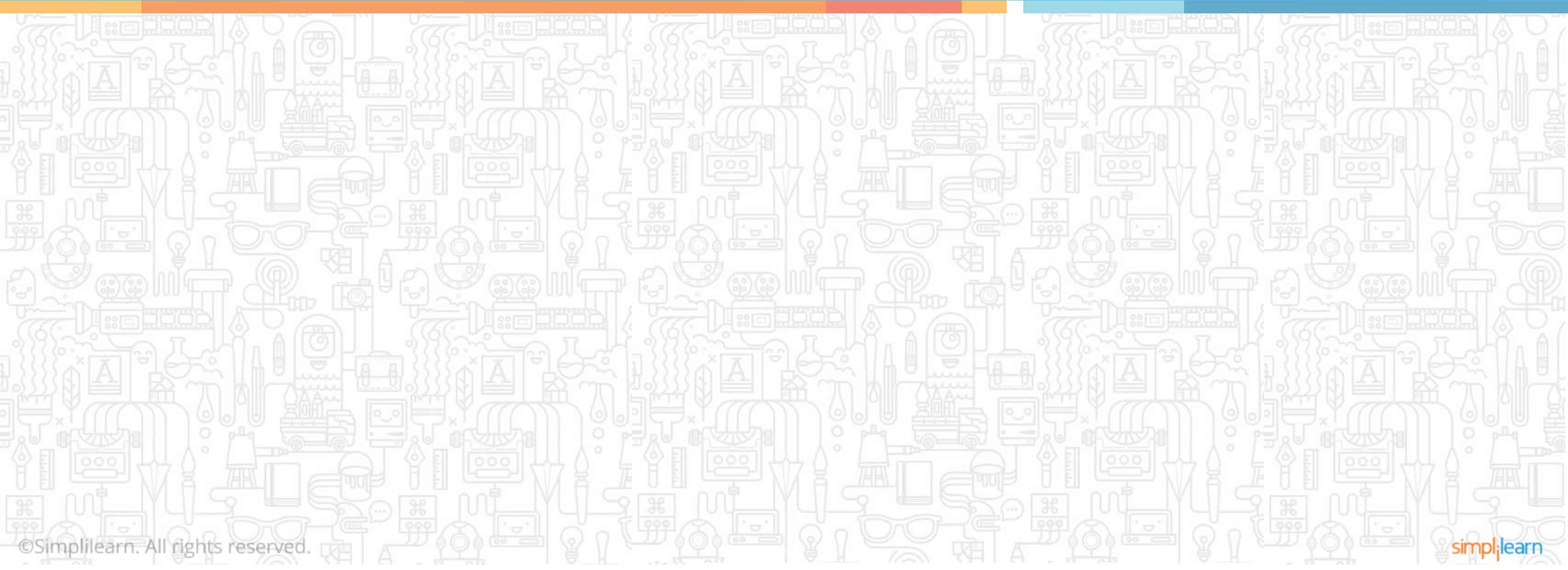


By the end of this lesson, you will be able to:

- ➊ Describe Hyperledger and its architecture
- ➋ Set up Hyperledger Sawtooth environment and perform transactions with it
- ➋ Set up Hyperledger Iroha network and perform transactions with it
- ➋ Interpret permissioned Blockchain and its consensus model

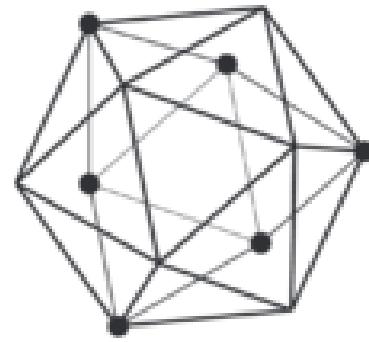
# Hyperledger

## Hyperledger and Its Importance



“Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by

The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing, and technology.”



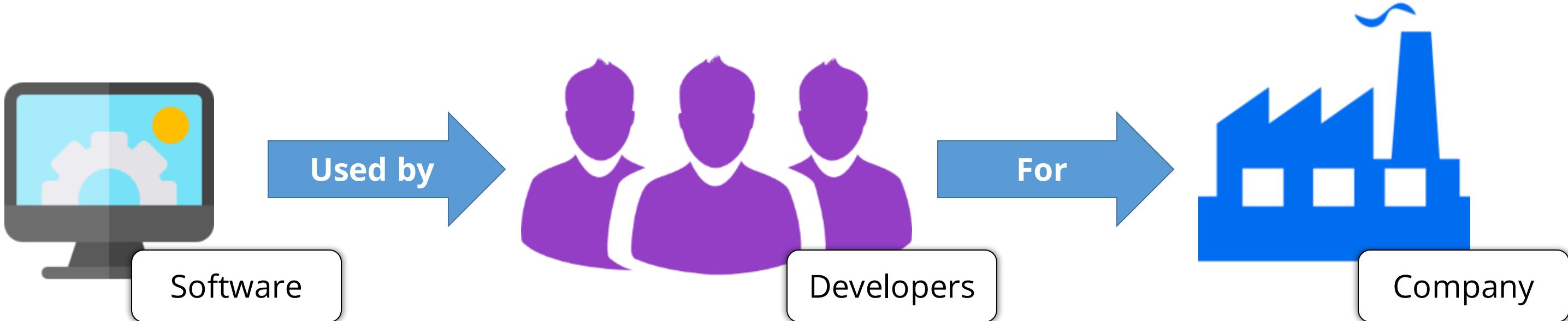
# HYPERLEDGER

Source: <https://www.hyperledger.org/>

# Concept of Hyperledger

Hyperledger is not a Blockchain, a company, or a cryptocurrency.

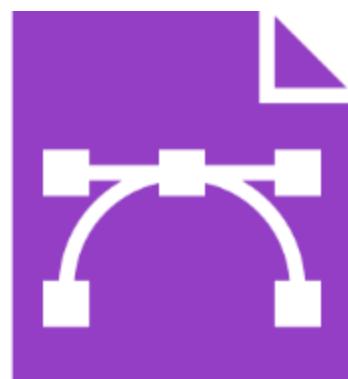
Hyperledger is a software used to create one's own personalized Blockchain service.



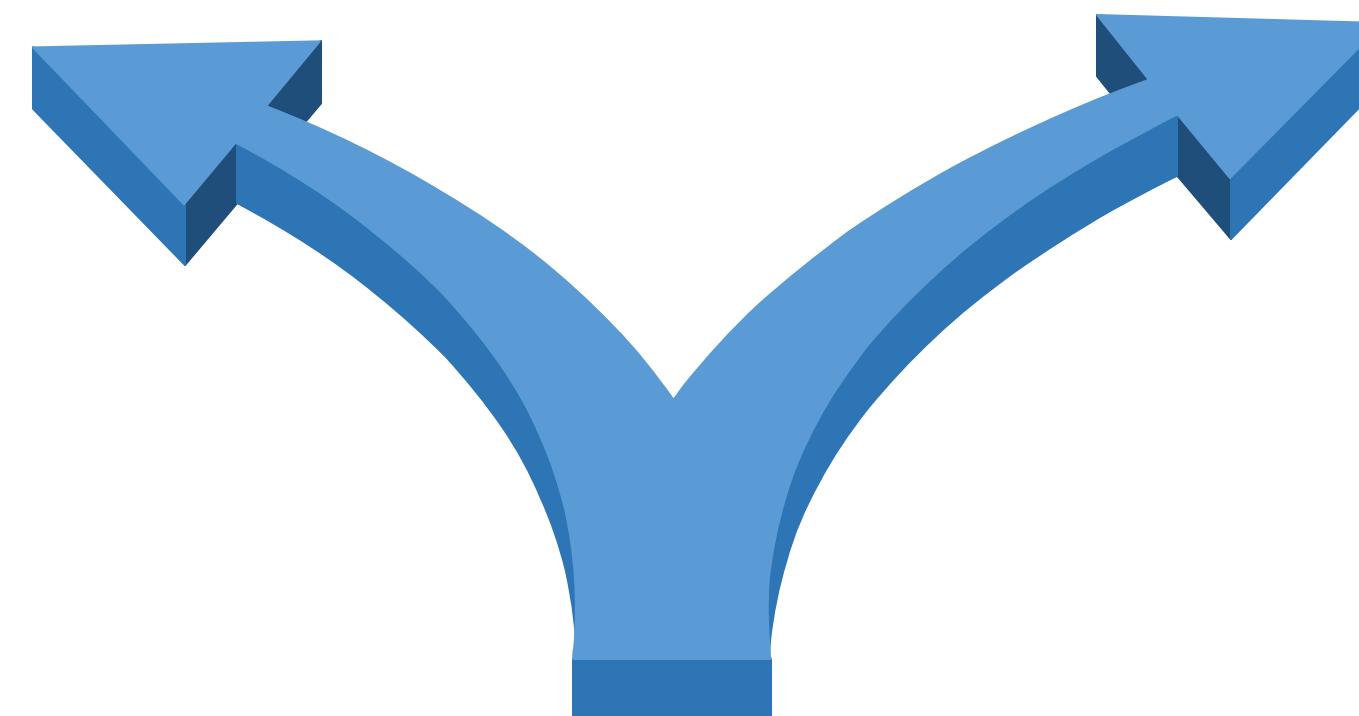
# Restrictions in Public Blockchain

In public Blockchain, each peer has to execute each and every transaction and run consensus algorithm to validate it.

**Not Scalable**



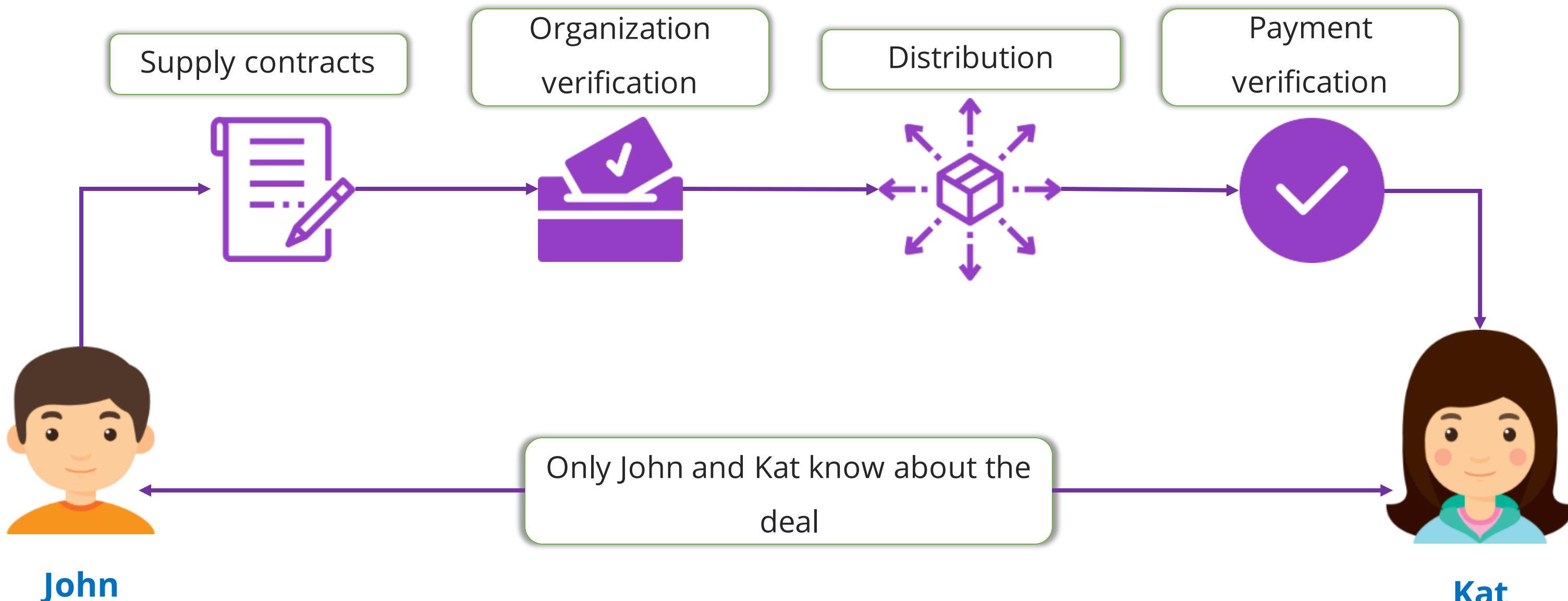
**Not Confidential**



**Public Blockchain**

# Solution To The Restrictions

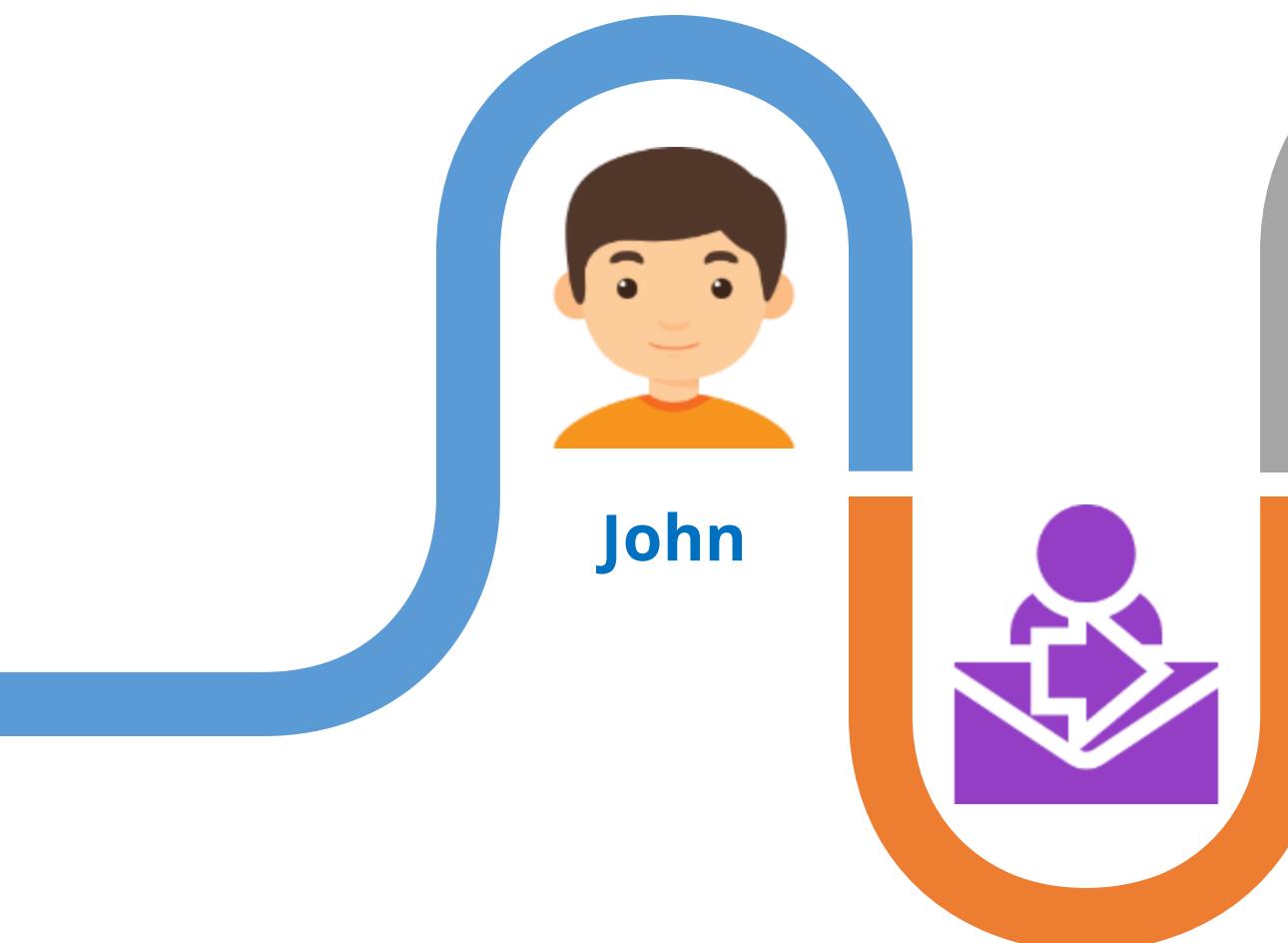
On Hyperledger network, only parties directly related to the transaction deal are updated on the ledger, thus maintaining privacy and confidentiality.



# Hyperledger Transaction

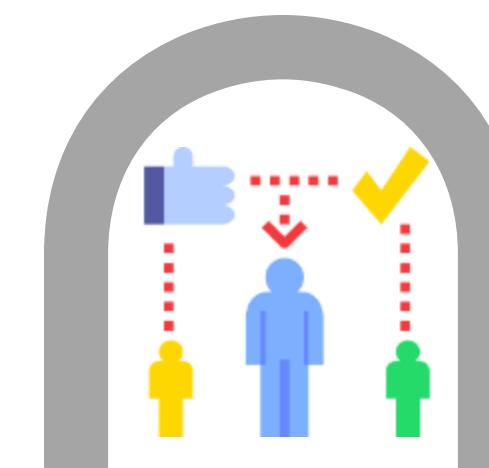
## Initiate Process

Plans to send message to Kat



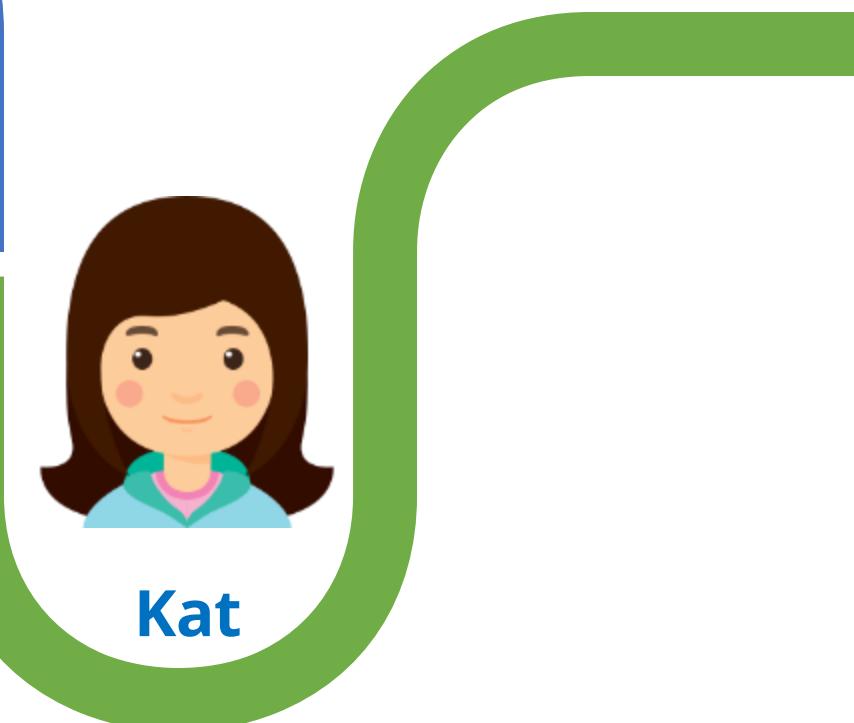
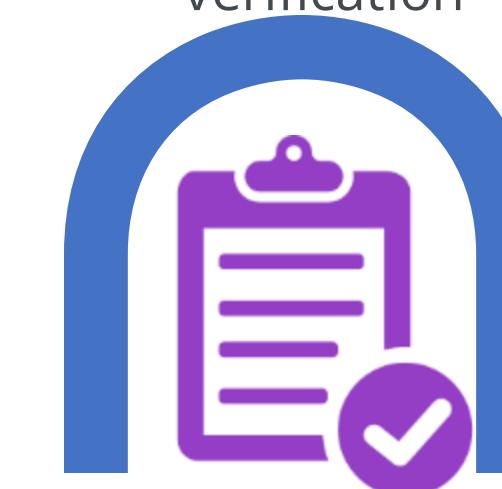
## Membership Services

Validates Kat's address



## Verification

Results generated by both the parties are sent to consensus cloud for verification



## App Query

Looks for Kat's address in the network

## Hyperledger

Hyperledger connects both parties directly related to the deal

## Process Ends

Once the transaction is verified from the consensus cloud, Kat receives the message

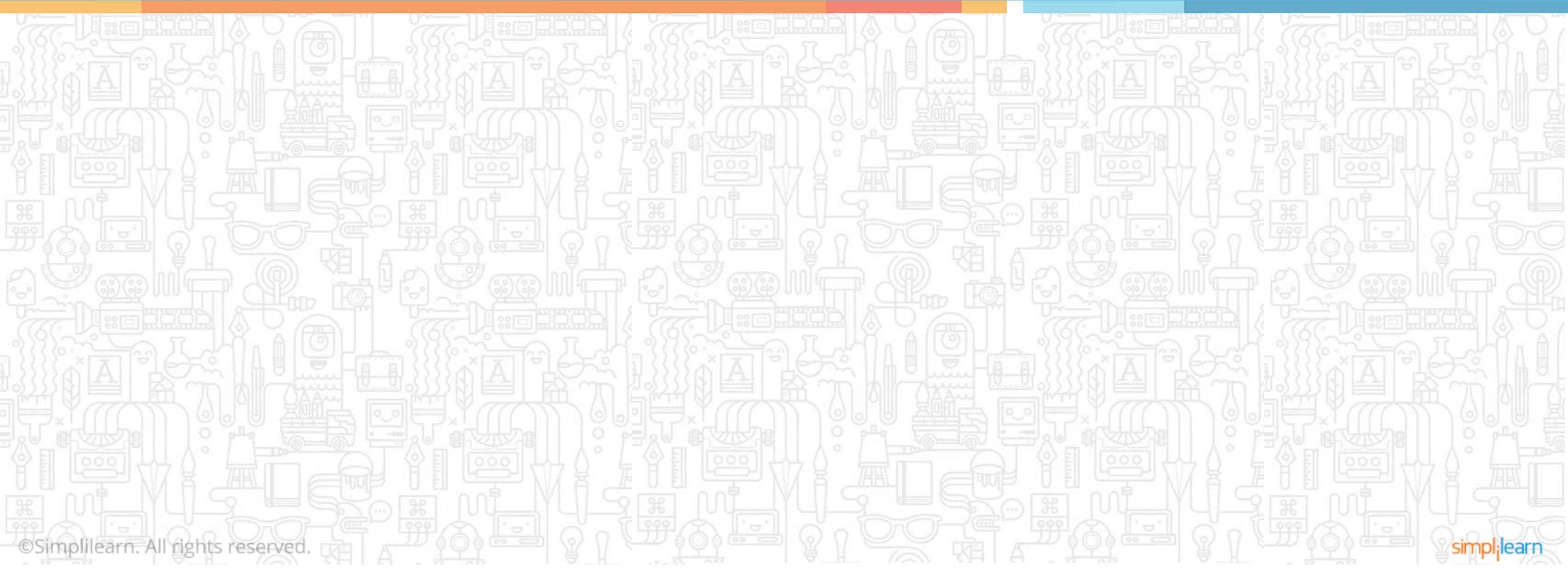
# Linux Family

Hyperledger is a part of the umbrella project under Linux Foundation where a community of developers work on open source projects.



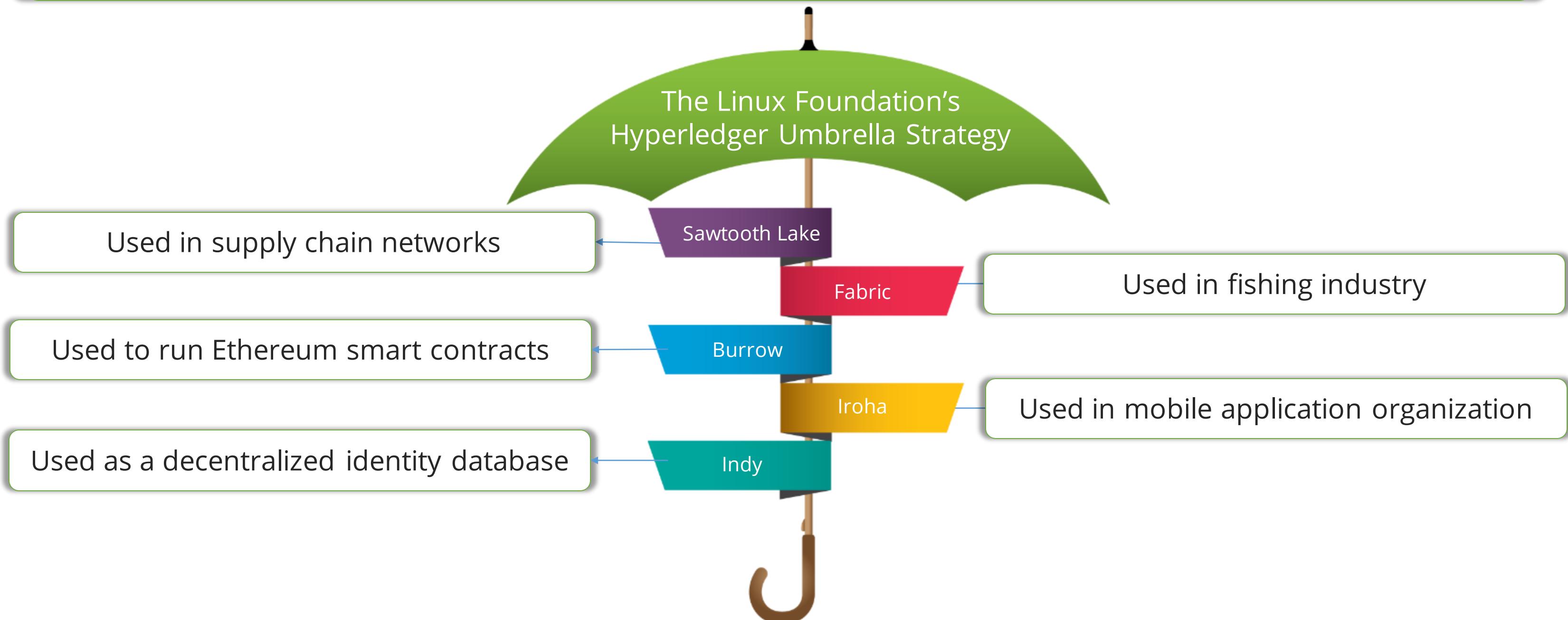
# Hyperledger

## Hyperledger Projects



# Hyperledger Umbrella Strategy

Hyperledger nurtures a lot of Blockchain frameworks and technologies under the umbrella strategy.



# Sawtooth

It's an enterprise Blockchain platform for building distributed ledger applications and networks. It keeps ledgers distributed and smart contracts safe.



## Assisted Practice

### Credits Exchange Using Sawtooth

Duration: 20 mins

**Problem Statement:** Simplilearn wants its participants to exchange credits. You are an employee of Simplilearn and have been asked to set up a Sawtooth environment for this.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

## Assisted Practice: Steps



### Step 01

Set up the Sawtooth environment

### Step 02

Create a participant, and sign in as existing participant

### Step 03

Create an asset and a holding

### Step 04

Create and accept an offer

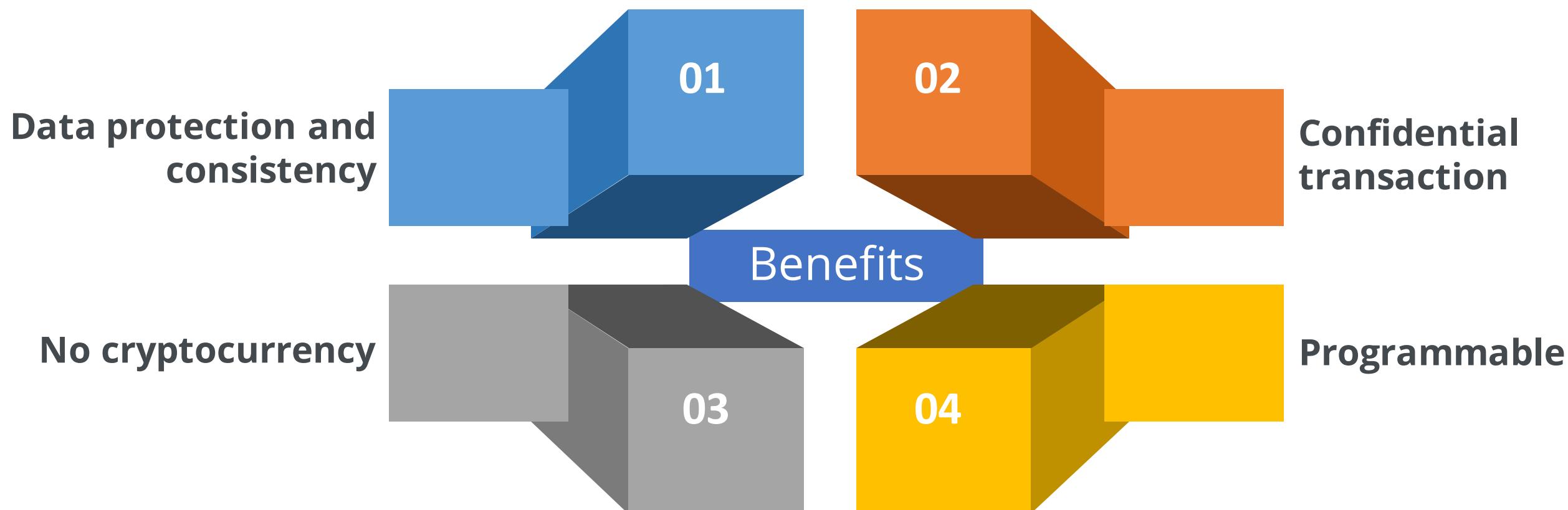
### Step 05

Chain the offers and transfer the credits directly

# Hyperledger Fabric



- Allows consensus and membership services to be plug-and-play
- Leverages container technologies to host smart contracts
- Is used extensively in supply chain networks



# Burrow

It's a permissioned Blockchain node built for a multi-chain universe that executes smart contracts following the Ethereum specification. It consists of the following components:



# Iroha



It's a distributed-ledger platform with open-source code written in C++, created with financial use-cases in mind.



**Simple**



**Trusted**



**Secure**



**Client-centric**



**HYPERLEDGER  
IROHA**



**Portable**

Source: <https://iroha.tech/>

## Assisted Practice

### Set Up Iroha Network

Duration: 20 mins

**Problem Statement:** You are given a task to set up an Iroha network on your computer and perform a transaction.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

## Assisted Practice: Steps



### Step 01

Install Docker, Ubuntu, and create a sample Docker network

### Step 02

Add PostgreSQL to the network, and create a blockstore

### Step 03 depth=1

Download Iroha code from <https://github.com/hyperledger/iroha -- depth=1>

### Step 04

Pull the Docker image, and run the Iroha Docker container

### Step 05

Run Iroha, and attach Docker container to the terminal

### Step 02

Launch the Iroha-CLI tool, and login as admin@test

### Step 03

Perform a transaction

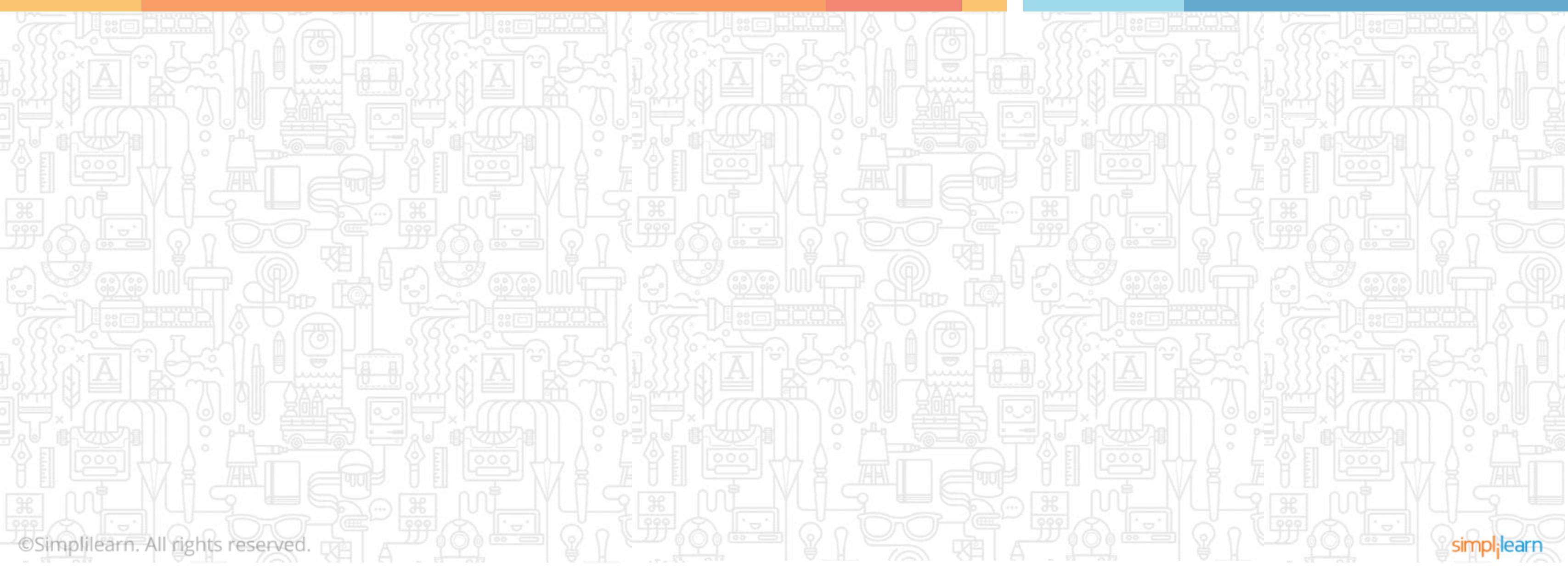
Indy



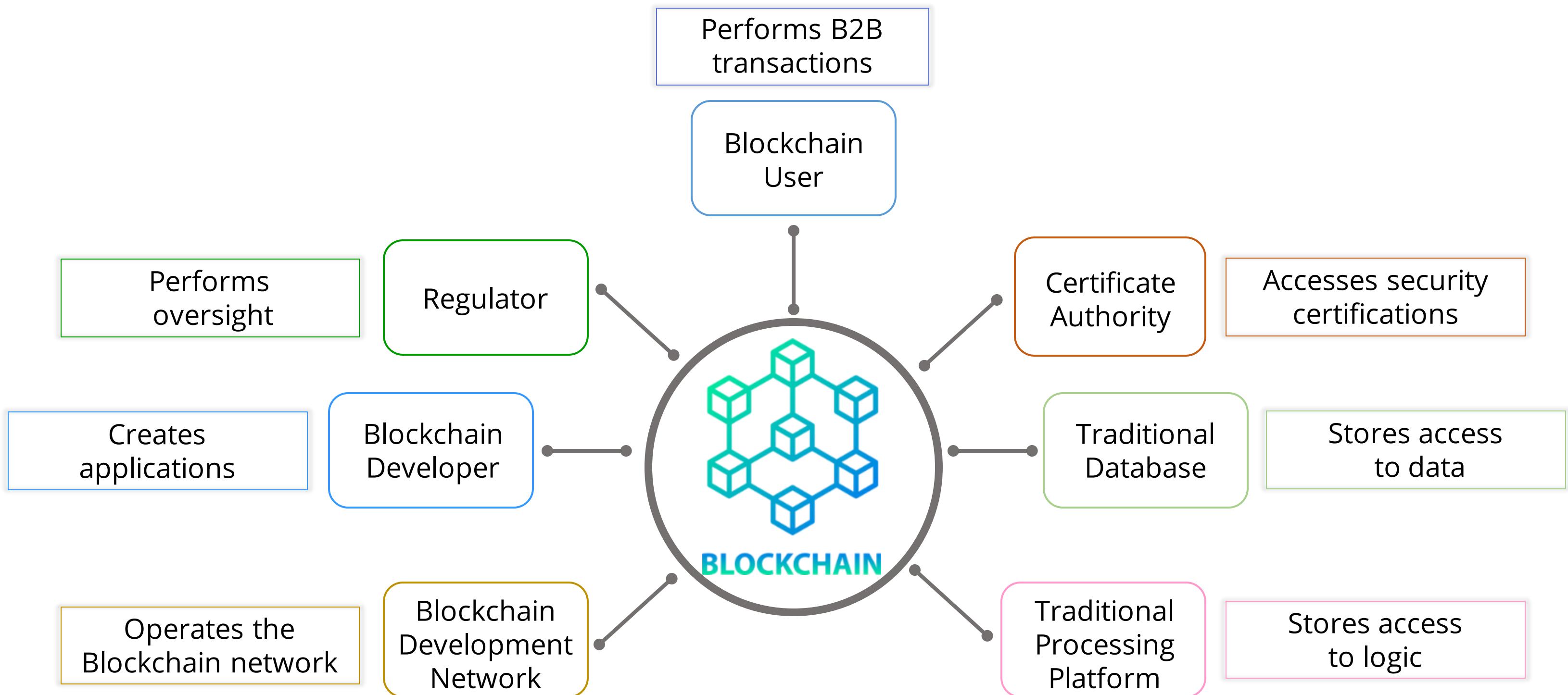
It's a distributed ledger built for decentralized identity that provides tools, libraries, and reusable components for creating digital identities rooted on Blockchain.

# Hyperledger

## Hyperledger Architecture



# Hyperledger Blockchain Network Participants



# Hyperledger Architecture Components



**Consensus layer**

**Smart contract layer**

**Communication layer**

**Data storage abstraction**

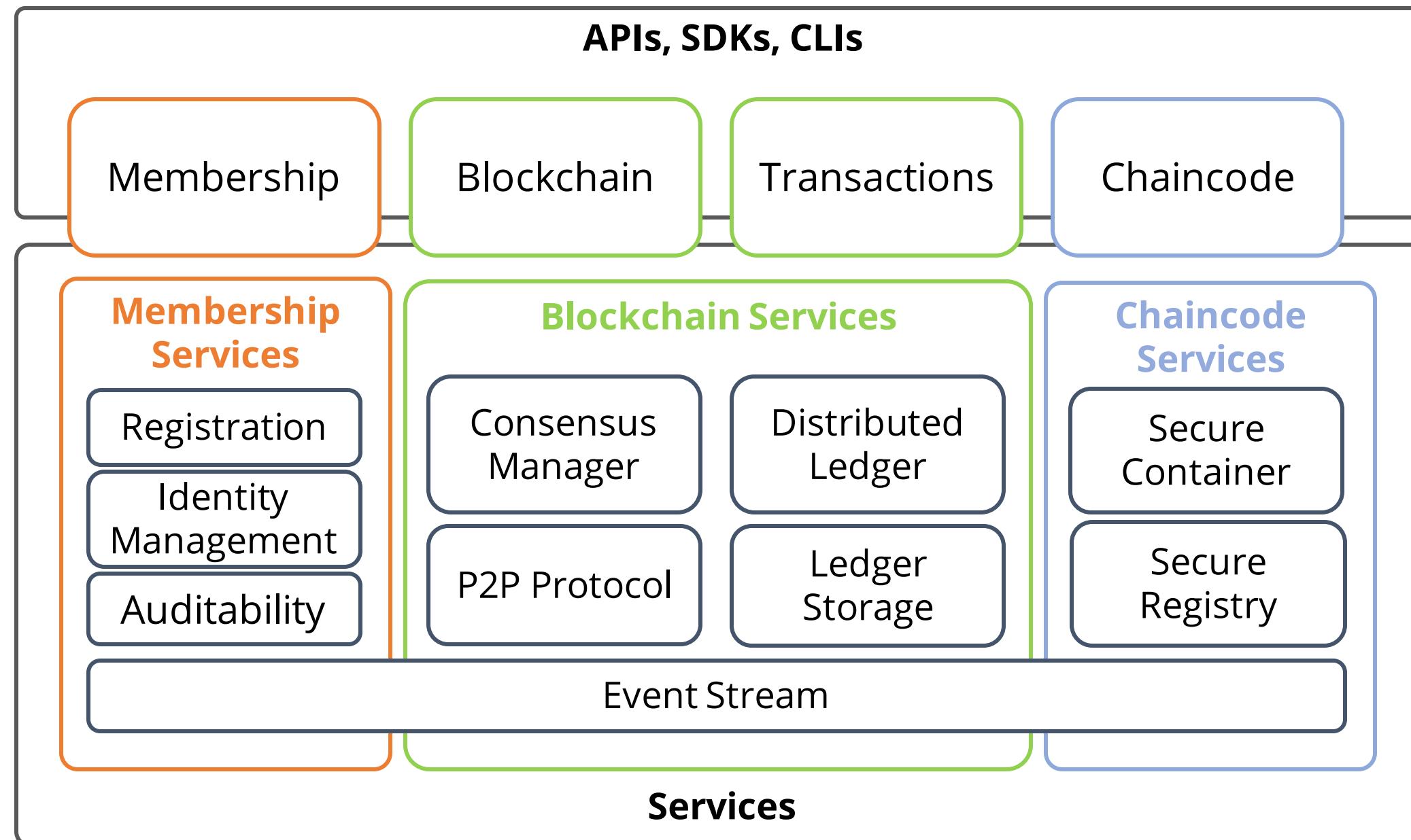
**Identity and policy services**

**Crypto abstraction**

**API**

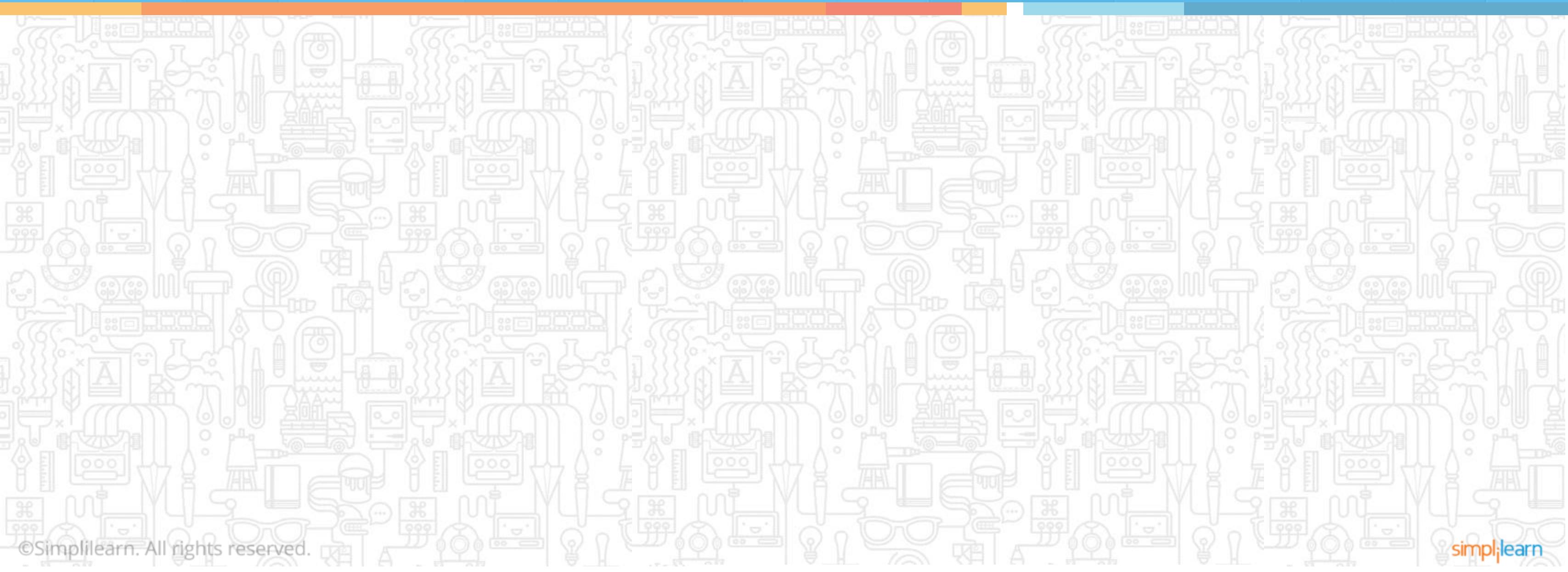
**Interoperation**

# Hyperledger Architecture



# Hyperledger

## Permissioned Blockchain and Its Consensus Model



# Permissioned Blockchain

Permissioned Blockchain maintains an access control layer and allows certain actions to be performed only by a few identifiable participants. It is also called Private Blockchain.

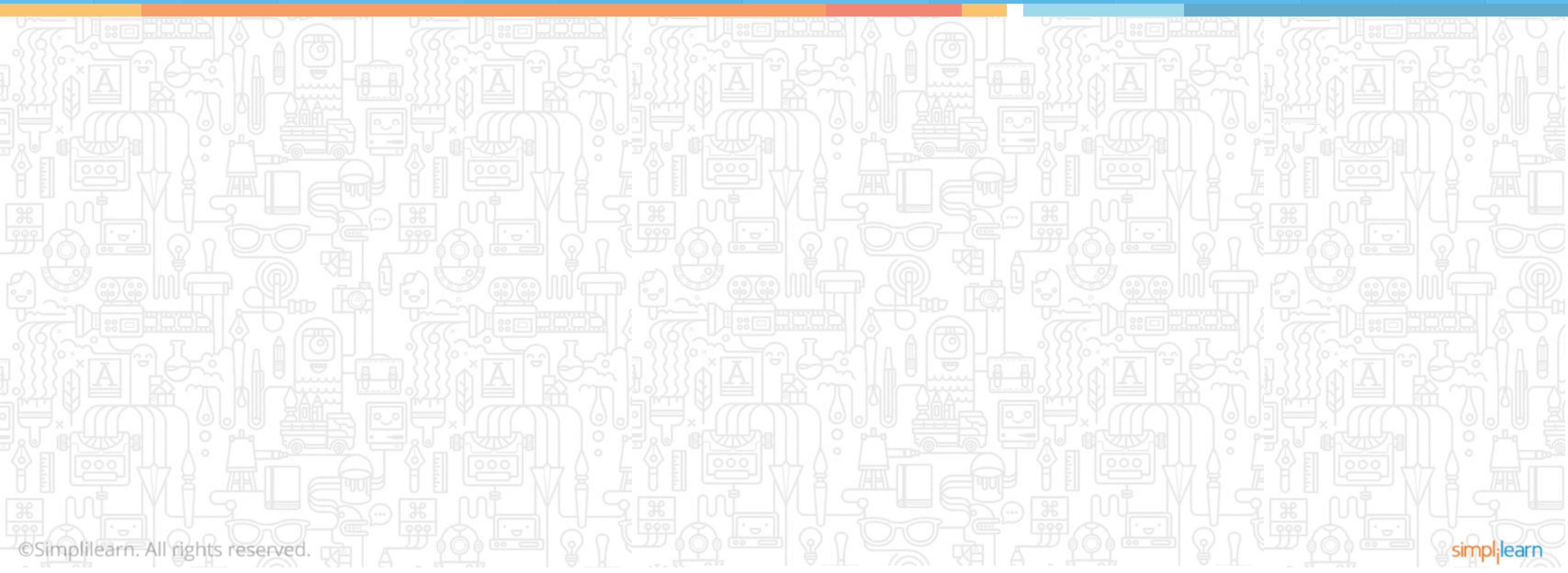


# Permissioned vs. Permissionless Blockchain

Characteristics	Permissioned	Permissionless
Access	Approved members only	Open and transparent access
Performance	Faster	Slower
Consensus	Proof-of-Stake	Proof-of-Work
Transaction cost	Low	High
Stability	Highly stable	Limited stability
Access control	Only members have full access	Same access level for all participants
Trust	Trusted environment	Trust-free environment

# Hyperledger

## Consensus and Its Interaction with Architectural Layers



# Consensus

---

Consensus is the process by which a network of nodes validates the block of transactions and provides a guaranteed ordering of transactions.

**Confirms accuracy of all transactions in a block**



**Agrees on the order and precision of execution**

**Depends on smart contract layer to verify the exact order of transaction**

Source: <https://www.hyperledger.org/>

# Consensus Properties

## Safety

Guarantees the same sequence of input and output of each node



## Liveness

Every submitted transaction is received by each non-faulty node

# Activities Performed to Reach Consensus



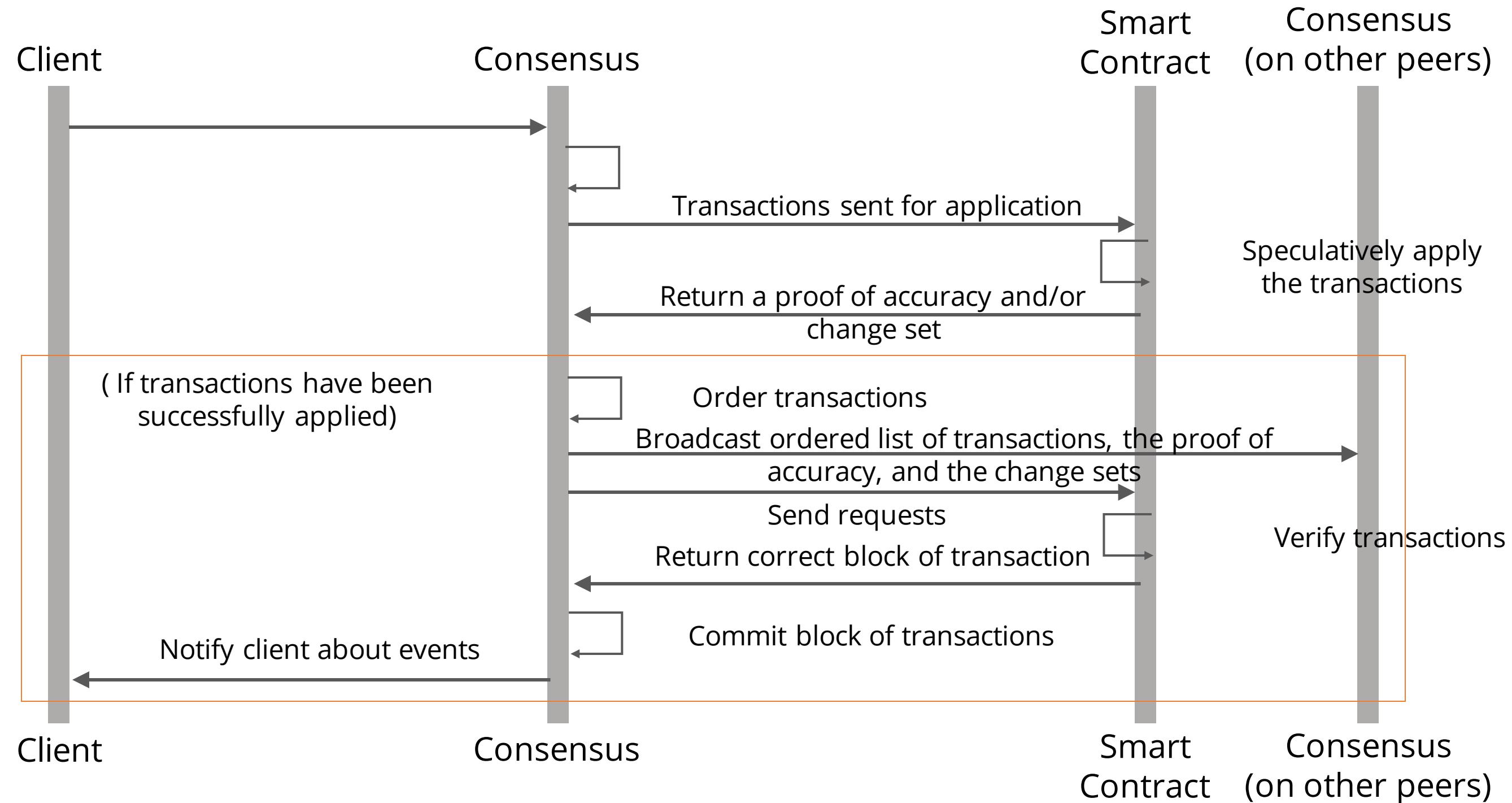
## Transaction Order

- Enables confidentiality of the transactions
- Increases the efficiency of transactions

## Transaction Validation

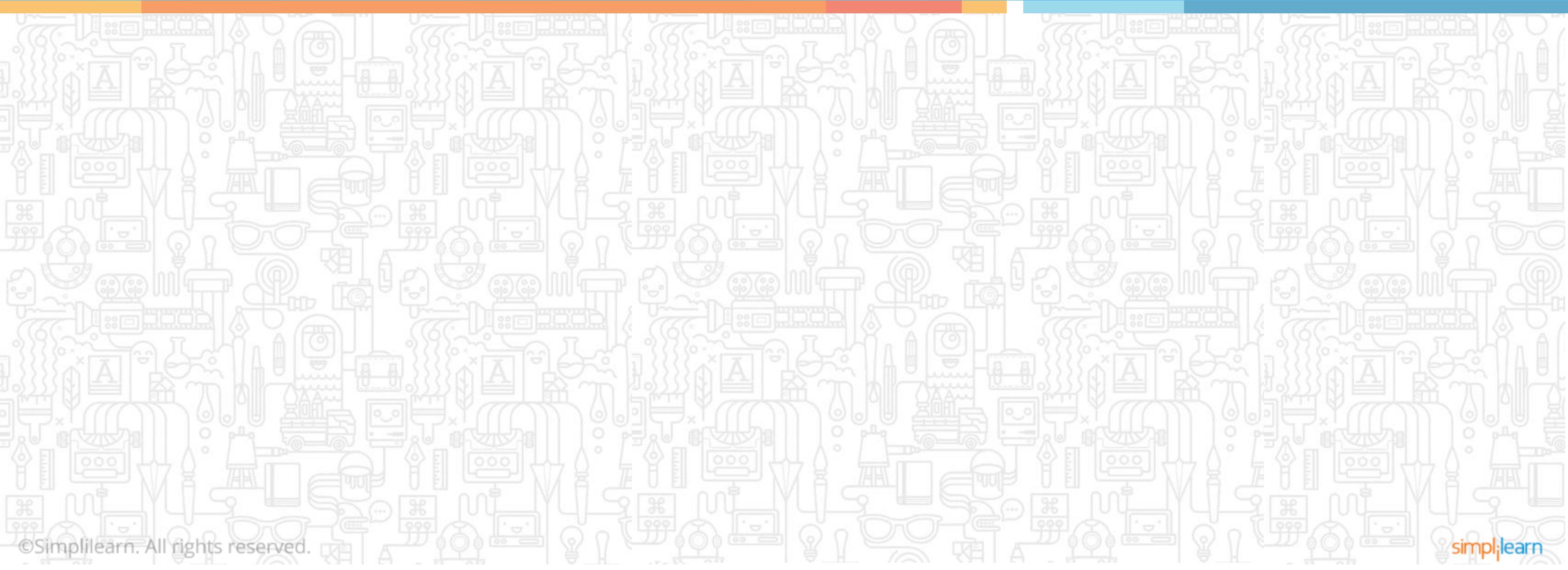
- Contains business logic to make the transaction valid
- Ensures the confirmation of policies and contracts specified for the transaction
- Checks for syntax and logic errors

# Consensus Model for Permissioned Blockchain



# Hyperledger

## API and Network Topology in Hyperledger



# API



It enables clients and applications to interface to the Blockchains.

## Admin and Client API

Used to perform business operations and administration tasks

## Common API

Used to obtain information about connected networks



## Runtime API

Used to perform all transaction operations

# Network Topology

Network topology is a technical infrastructure that provides smart contract services and ledger to applications.

Listed below are the components of network topology:

**Ledger**

**Smart contracts**

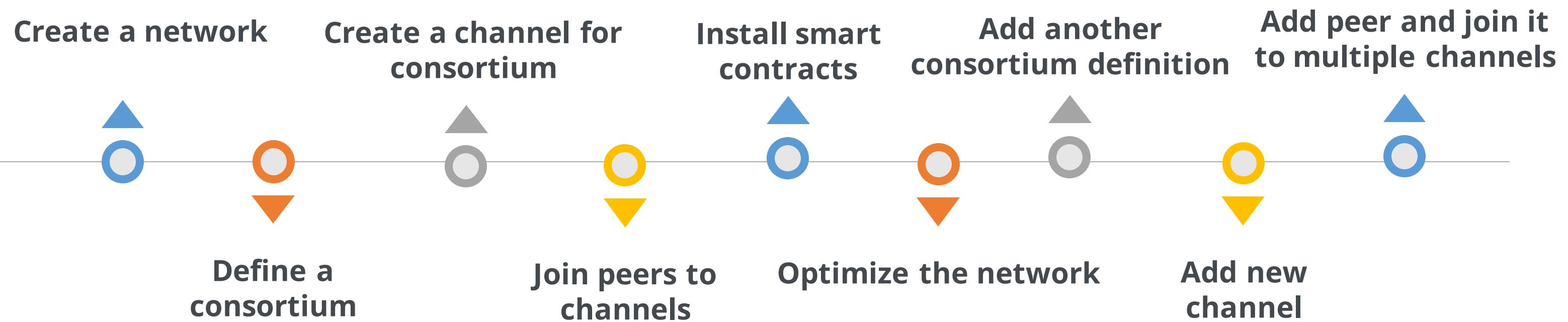
**Peer nodes**

**Ordering services**

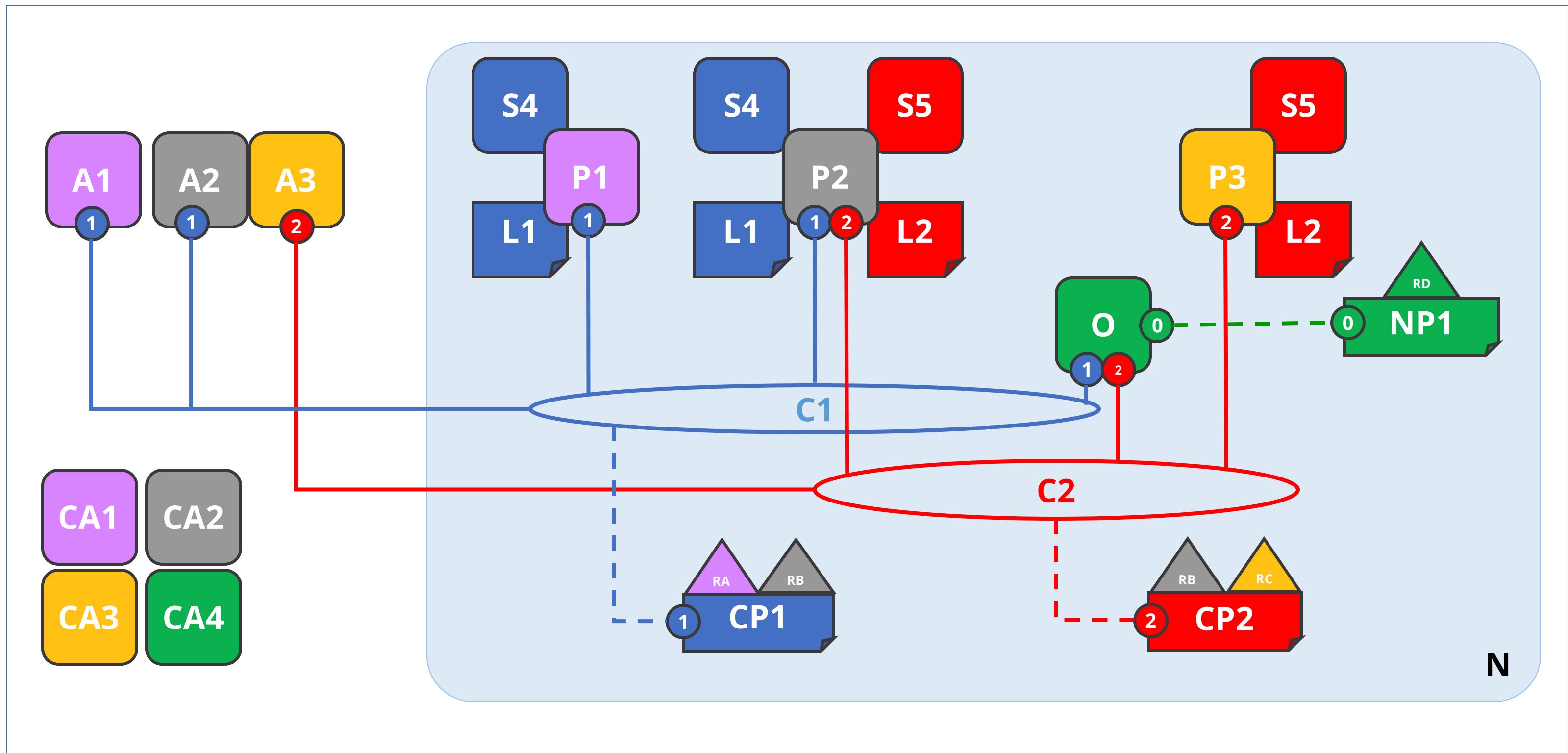
**Channels**

**Fabric Certificate Authorities**

# Steps to Create a Hyperledger Network



# Network Topology



# Key Takeaways

You are now able to:

- ➊ Describe Hyperledger and its architecture
- ➋ Set up Hyperledger Sawtooth environment and perform transactions with it
- ➌ Set up Hyperledger Iroha network and perform transactions with it
- ➍ Interpret permissioned Blockchain and its consensus model





## Knowledge Check



## What is Hyperledger?

- a. A cryptocurrency
- b. A Blockchain
- c. A company
- d. A software



## What is Hyperledger?

- a. A cryptocurrency
- b. A Blockchain
- c. A company
- d. A software



The correct answer is **d**

**Hyperledger is a software used to create one's own personalized Blockchain service.**

## Which consensus algorithm does permissioned Blockchain use?

- a. Proof of Work
- b. Proof of Stake
- c. Proof of Elapsed Time
- d. Practical Byzantine Fault Tolerance



## Which consensus algorithm does permissioned Blockchain use?

- a. Proof of Work
- b. Proof of Stake
- c. Proof of Elapsed Time
- d. Practical Byzantine Fault Tolerance



The correct answer is **b**

**Consensus algorithm makes use of Proof of Stake algorithm to perform the transaction.**

## Which API is used to perform administration tasks?

- a. Common API
- b. Runtime API
- c. Admin and Client API
- d. None of the above



## Which API is used to perform administration tasks?

- a. Common API
- b. Runtime API
- c. Admin and Client API
- d. None of the above



The correct answer is

C

**Admin and Client API is used to perform business operations and administration tasks.**

## Lesson-End Project

Duration: 30 mins

### Transform the Supply Chain

**Problem Statement:** The traditional seafood supply chain industry has illegal, unreported, and unregulated fishing practices. You are required to bring traceability and accountability to the supply chain through the power of Hyperledger Sawtooth technology.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 6: Hyperledger Composer



# Learning Objectives

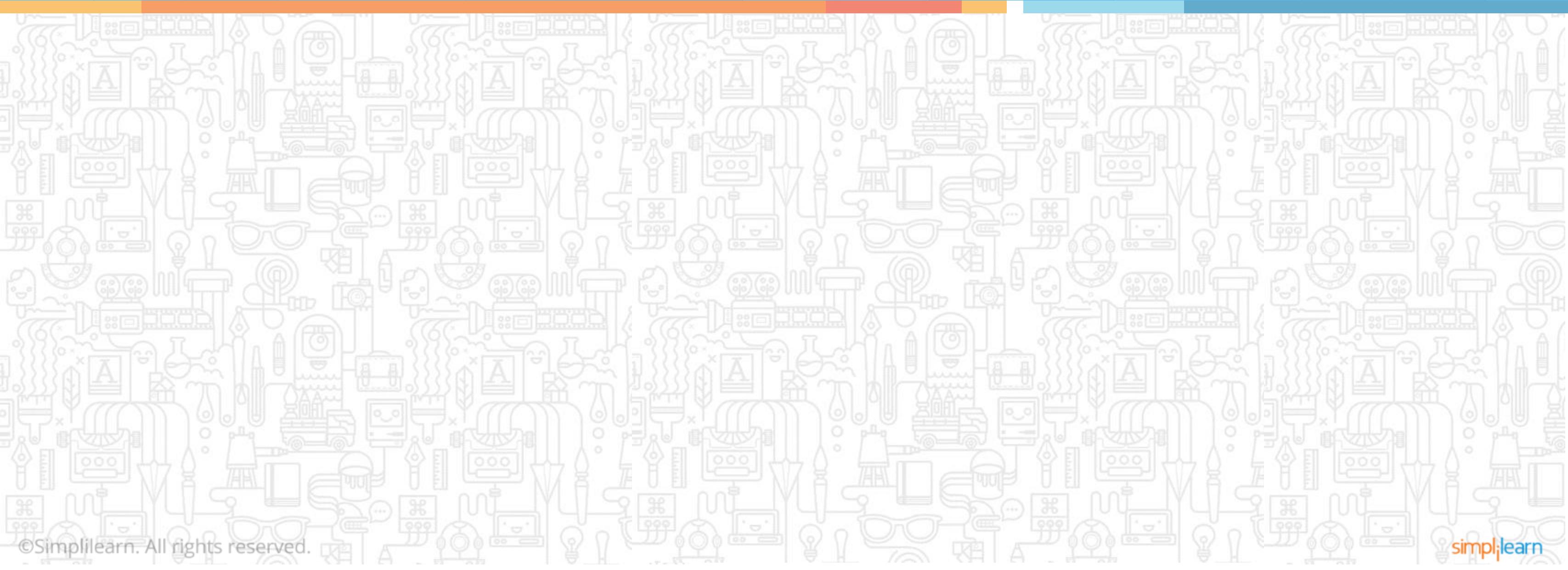


By the end of this lesson, you will be able to:

- ➊ Describe Hyperledger Fabric and Hyperledger Composer
- ➋ Set up a development environment using Hyperledger Composer
- ➌ Create, deploy, and test a business network

# Hyperledger Composer

## Hyperledger Fabric and Its Concepts



# Hyperledger Fabric

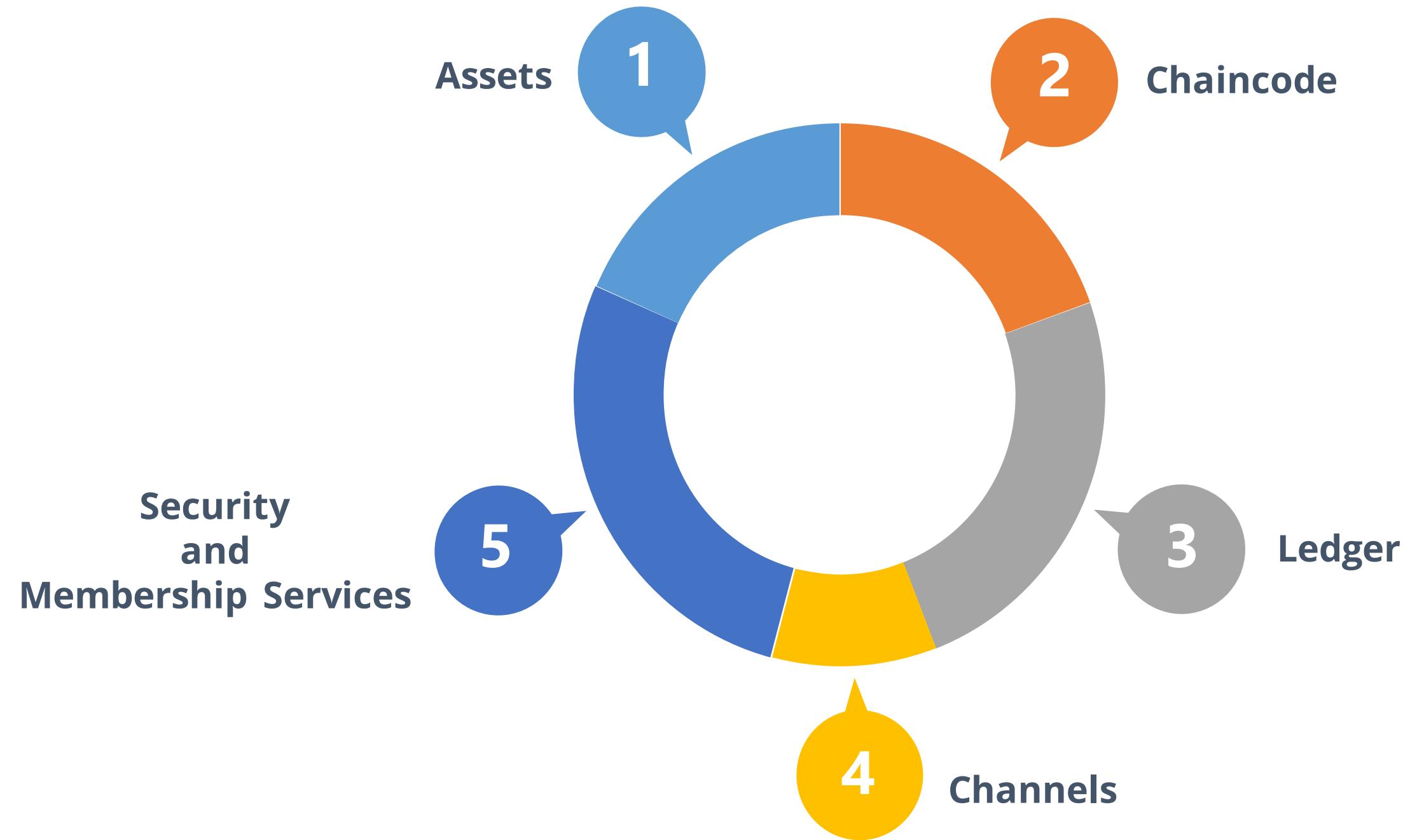


Hyperledger Fabric is a permissioned Blockchain system used for developing applications. It allows unknown identities to participate in the network enrolled through Member Service Provider (MSP).

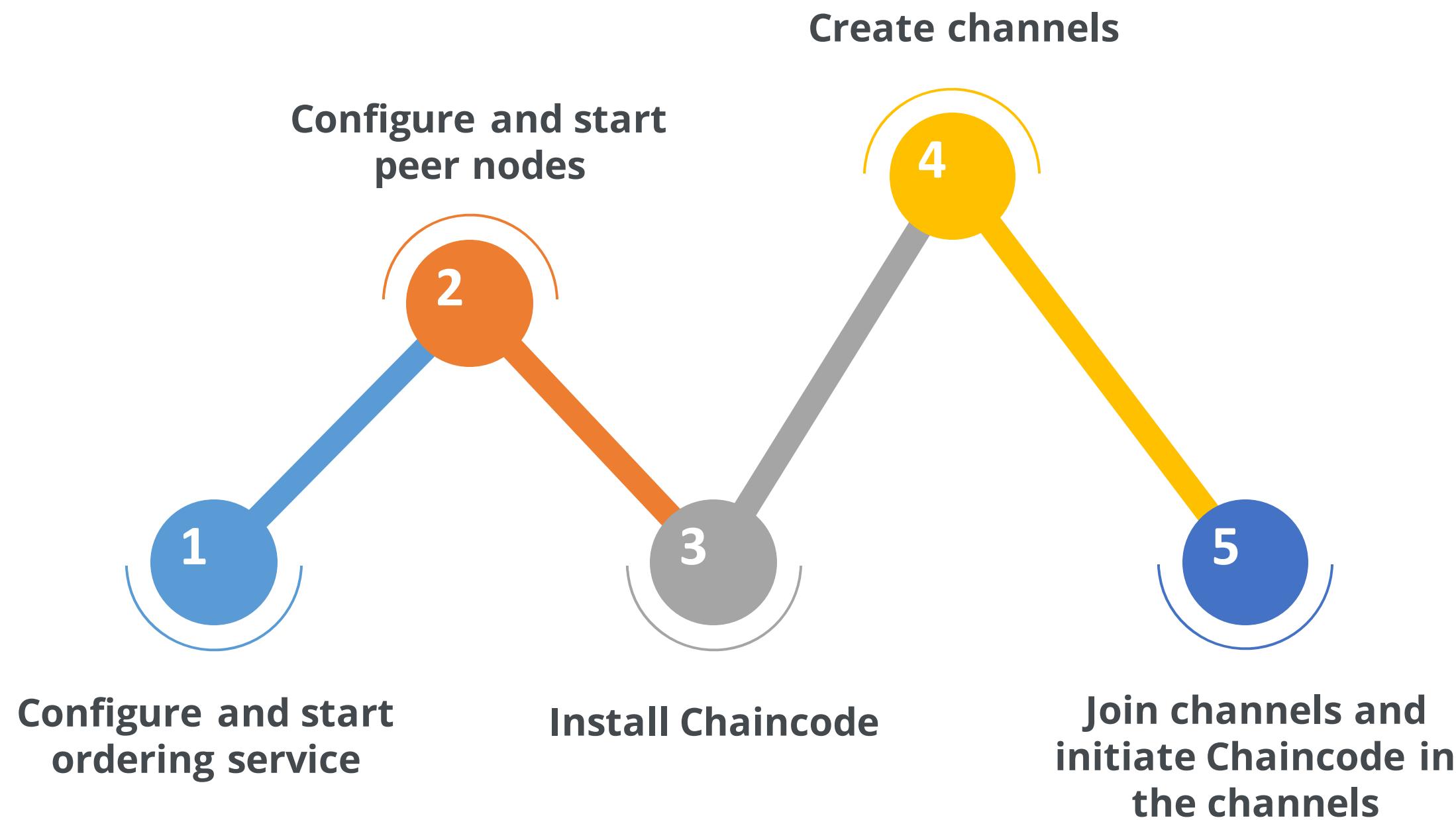
# Issues in Hyperledger Fabric



# Hyperledger Fabric Model



# Creating Hyperledger Fabric Network

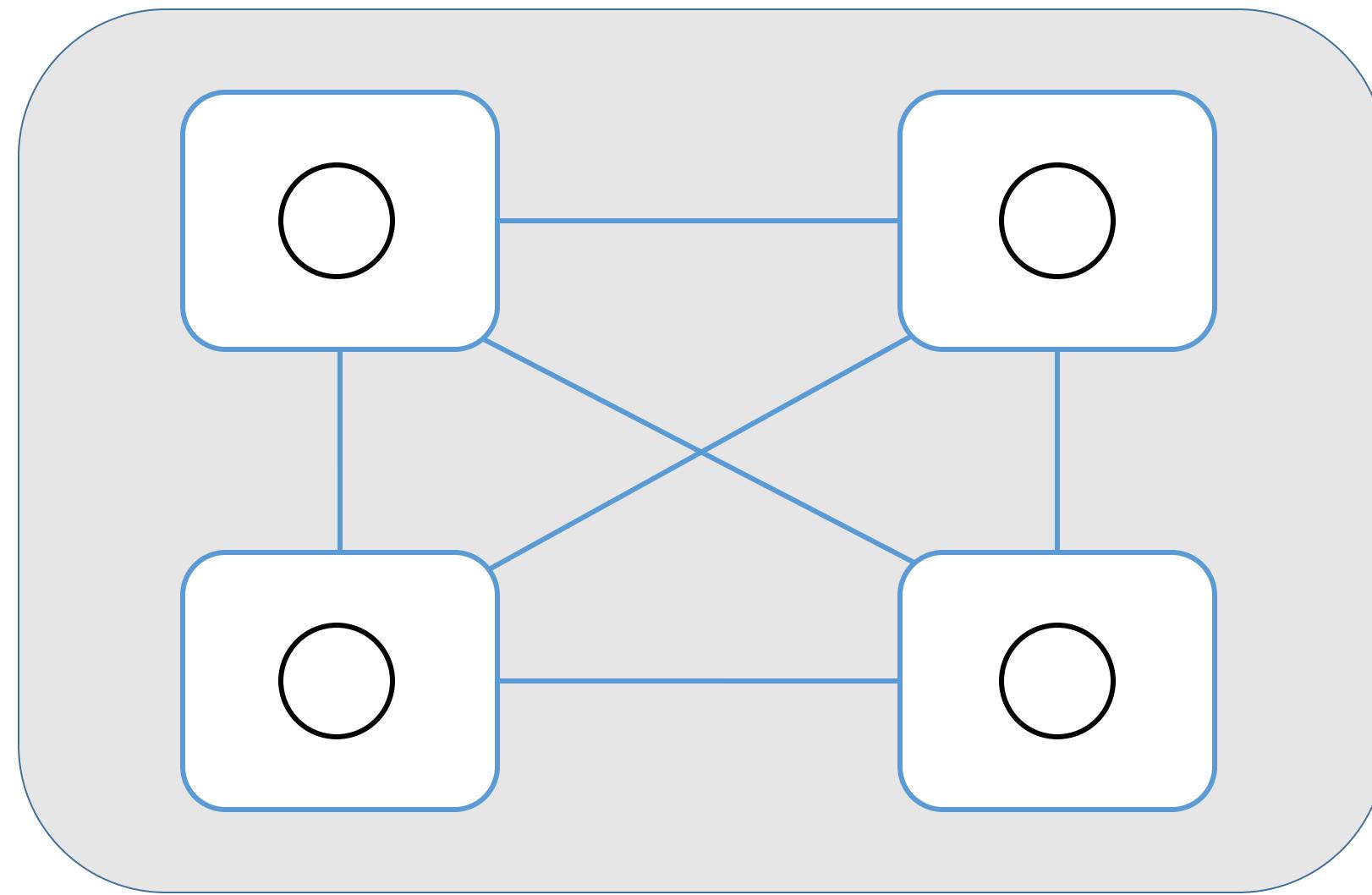




- Chaincode is a program that initiates and manages the ledger state through transaction
- It runs in a secured Docker container
- It handles the business logic agreed to by the members of the network

# Ordering Service

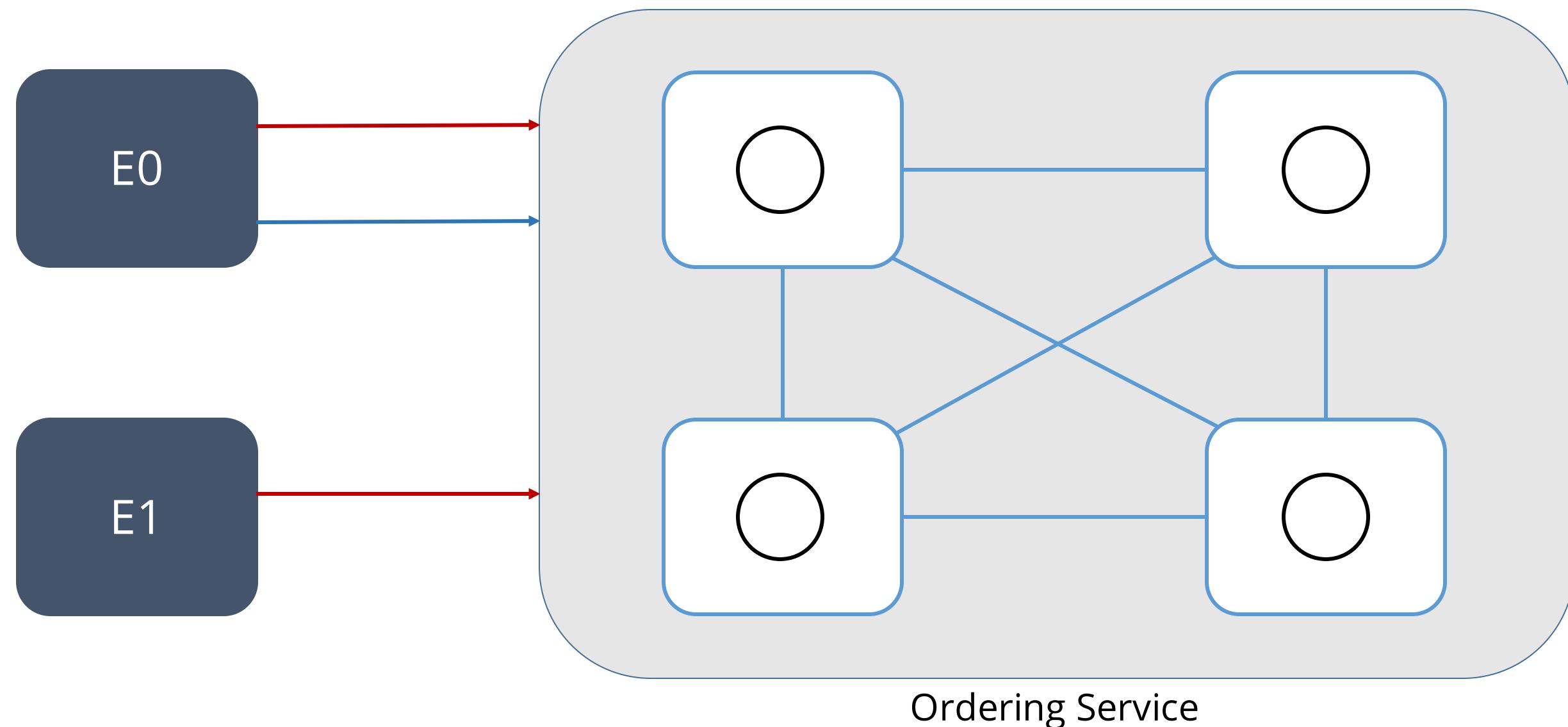
The ordering service packages transactions into the blocks that are delivered to the peers. The communication with the services is done via channels.



Ordering Service

# Channels

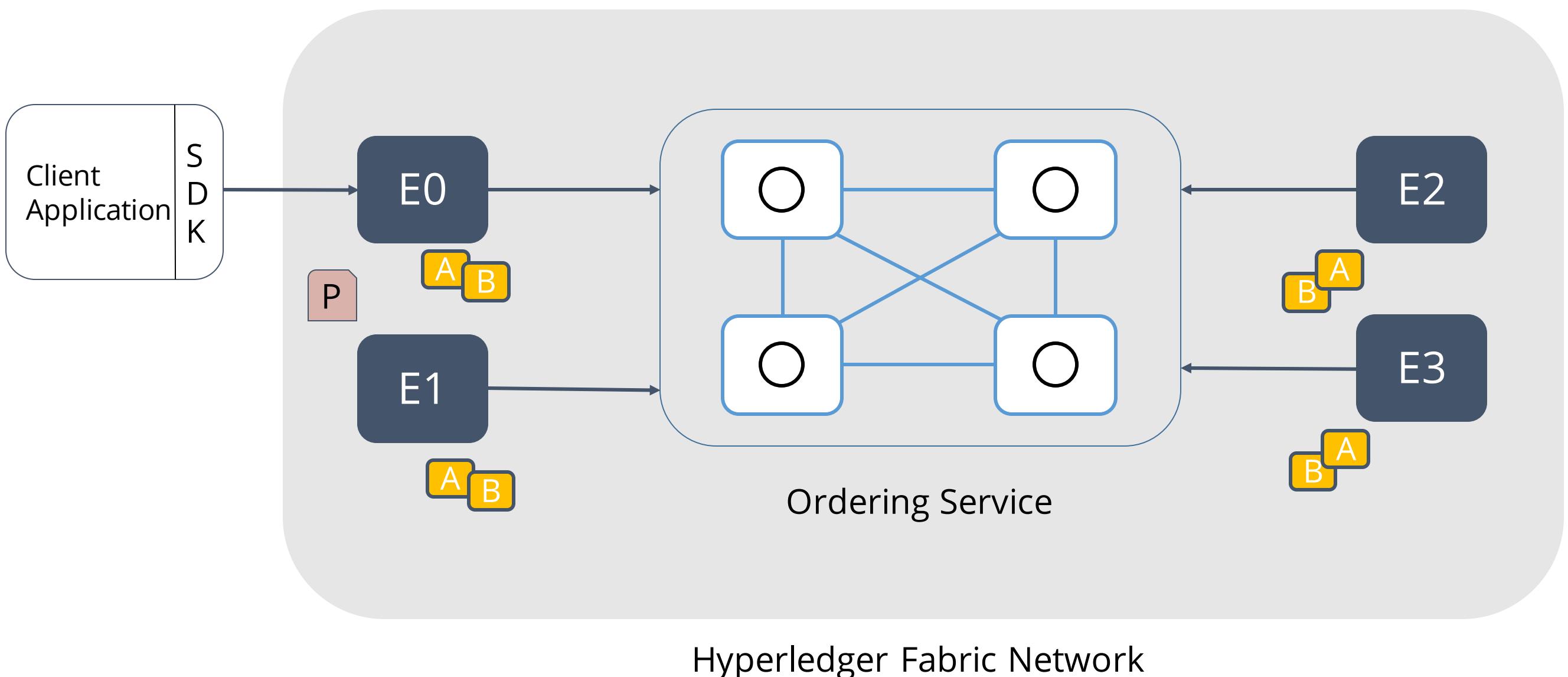
Channels are shared across an entire network and are permissioned for a specific set of participants.



Ordering Service

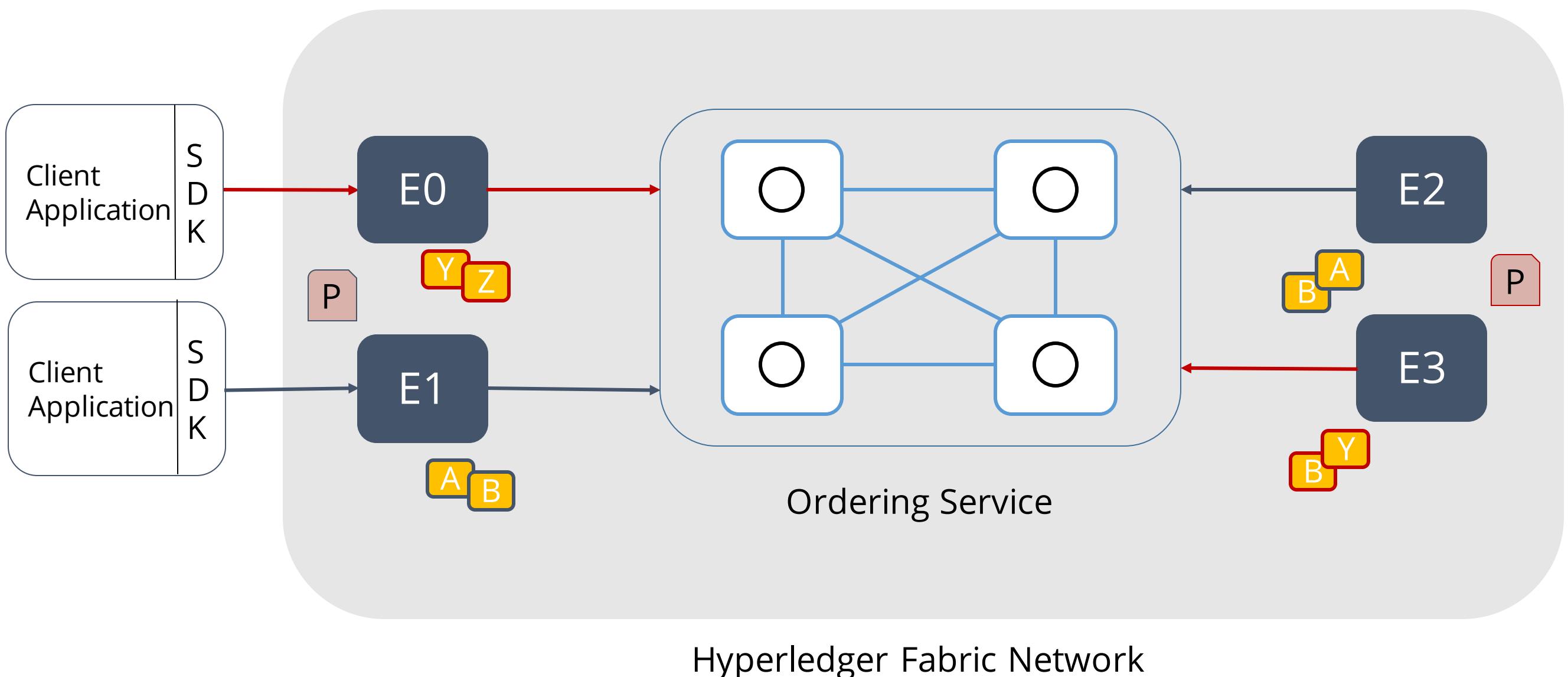
# Single-Channel Network

- All peers connect to the same system channel
- All peers maintain the same ledger and have the same Chaincode



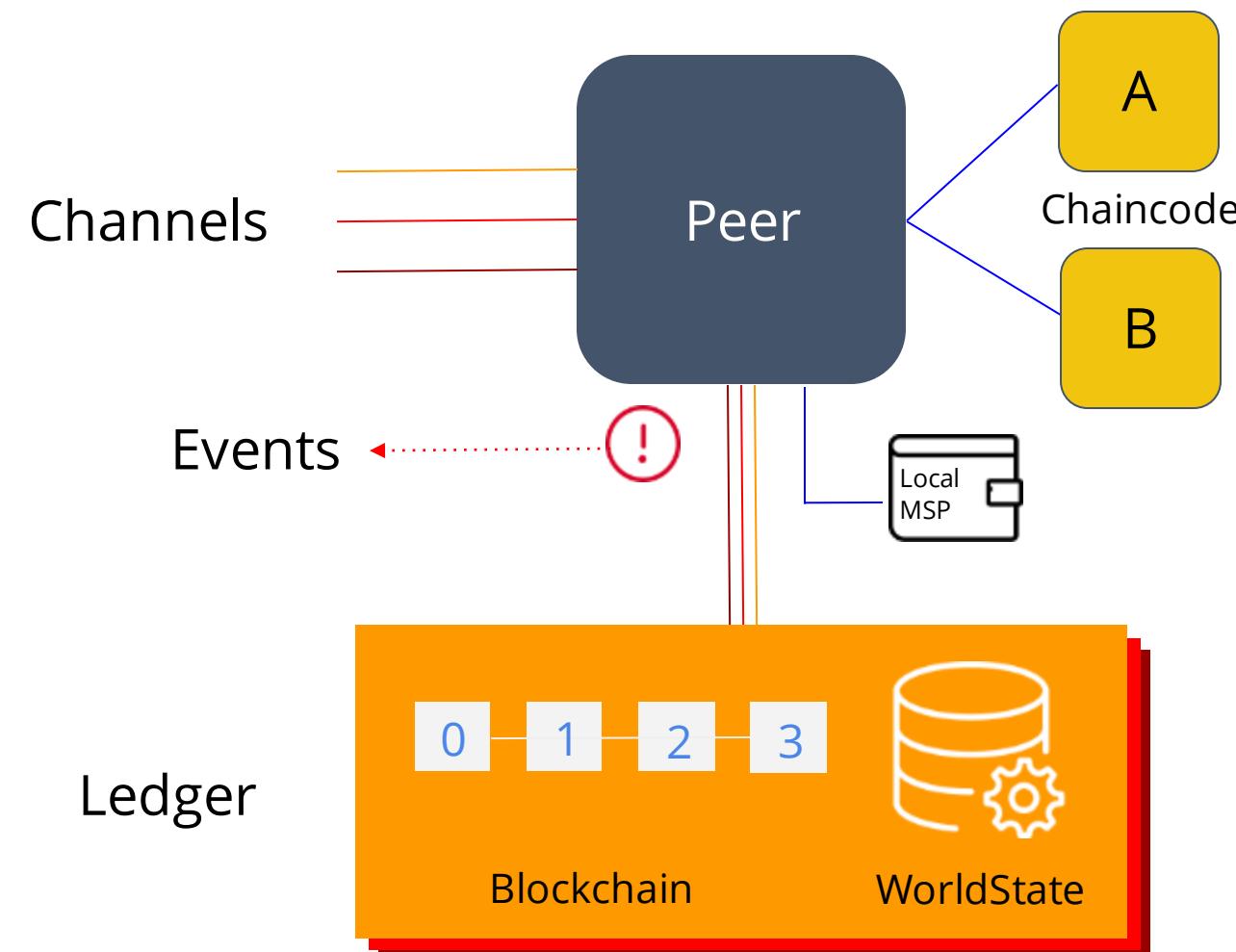
# Multi-Channel Network

- All peers are not connected to the same system channel
- All peers do not have the same Chaincode



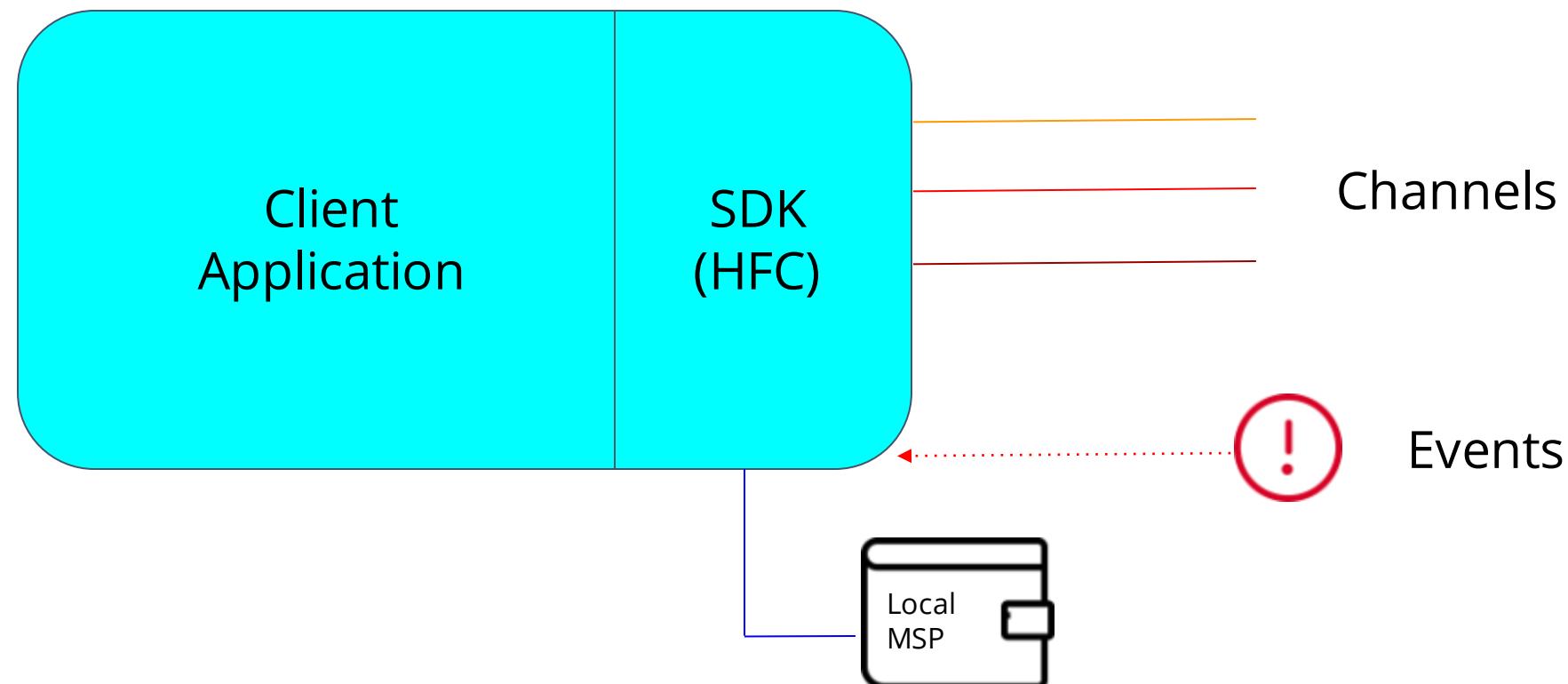
# Fabric Peer

- Maintains one or more ledger and connects to more than one channels
- Assigns events to client application



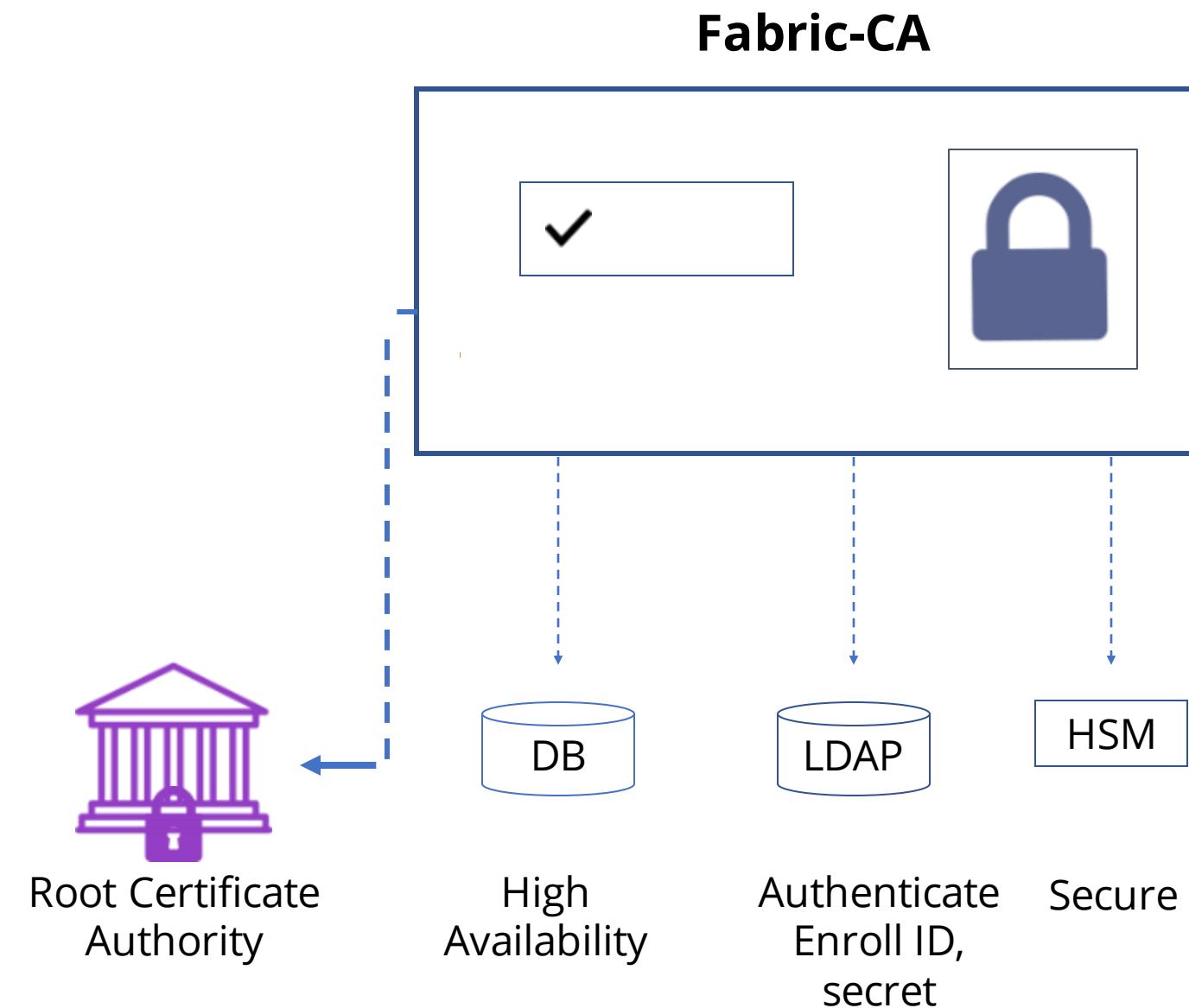
# Client Application

- Uses Fabric SDK to connect to peers over channels and receive events from them
- Can be written in different languages like Node.js, Go, Java, and Python



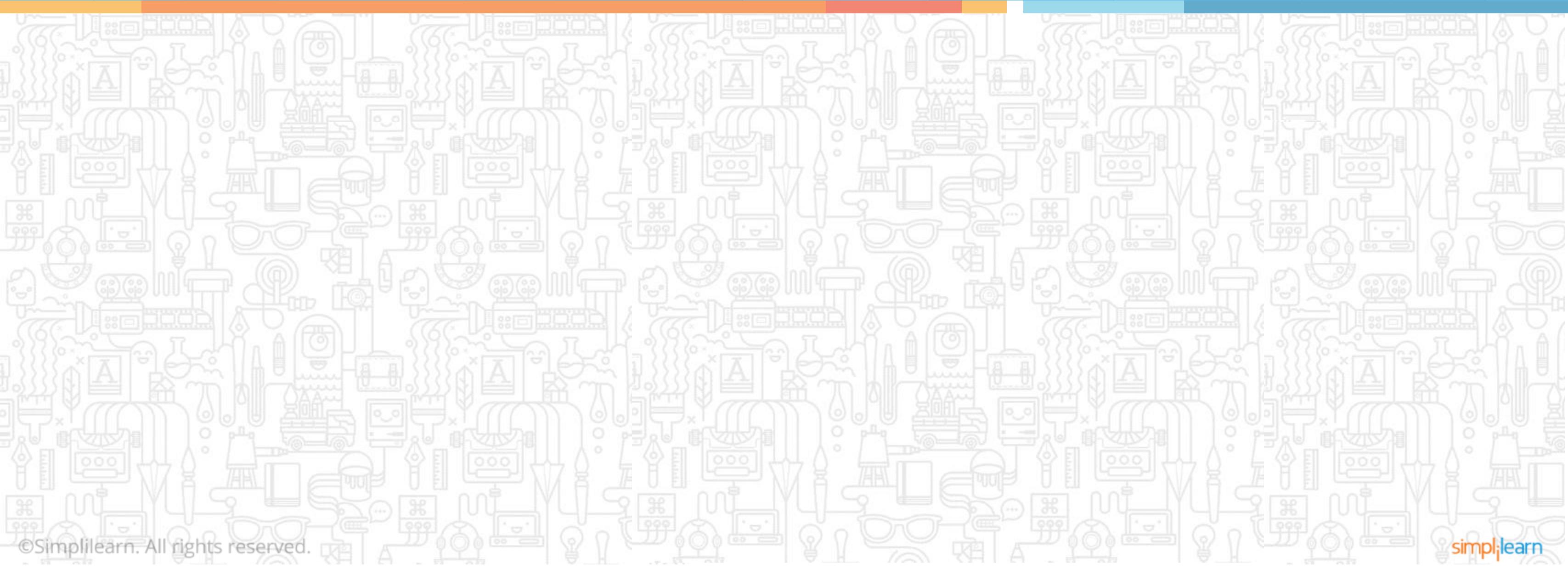
# Fabric Certificate Authority

- Used within Fabric network for issuing long-term identity
- Provides authentication and security



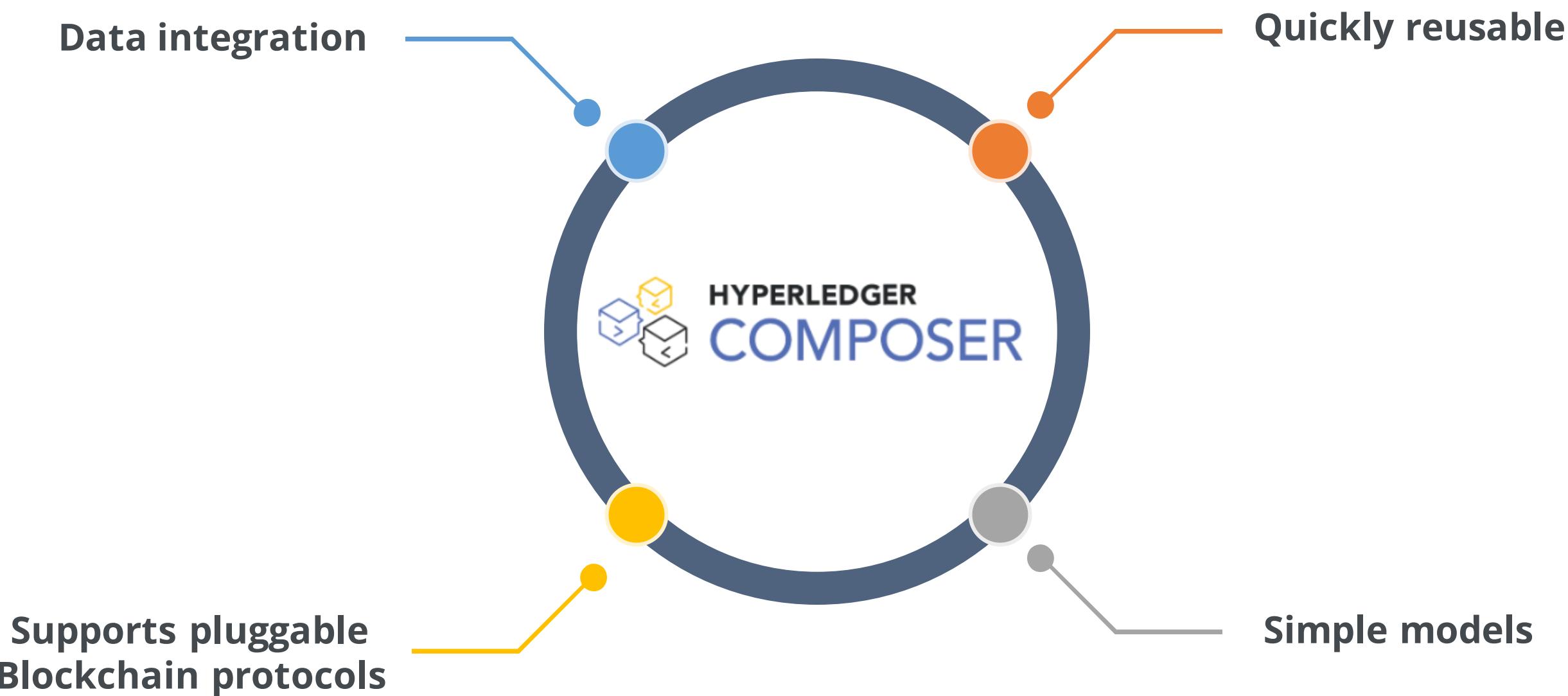
# Hyperledger Composer

## Hyperledger Composer and Its Features



# Hyperledger Composer

Hyperledger Composer is a development tool that is used to develop use cases and deploy Blockchain applications easily.



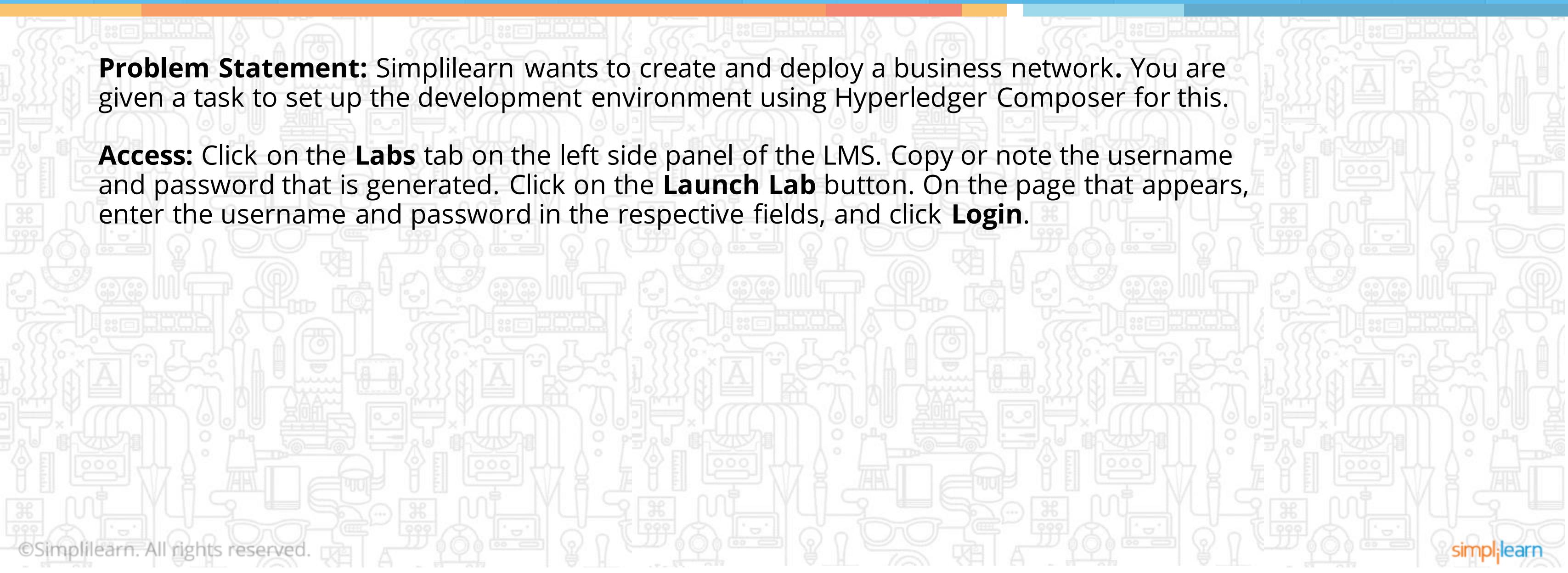
## Assisted Practice

Duration: 10 mins

### Set Up Development Environment Using Hyperledger Composer

**Problem Statement:** Simplilearn wants to create and deploy a business network. You are given a task to set up the development environment using Hyperledger Composer for this.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



**Step 01**

Install NVM

**Step 02**

Install the CLI tools

**Step 03**

Install Playground

**Step 04**

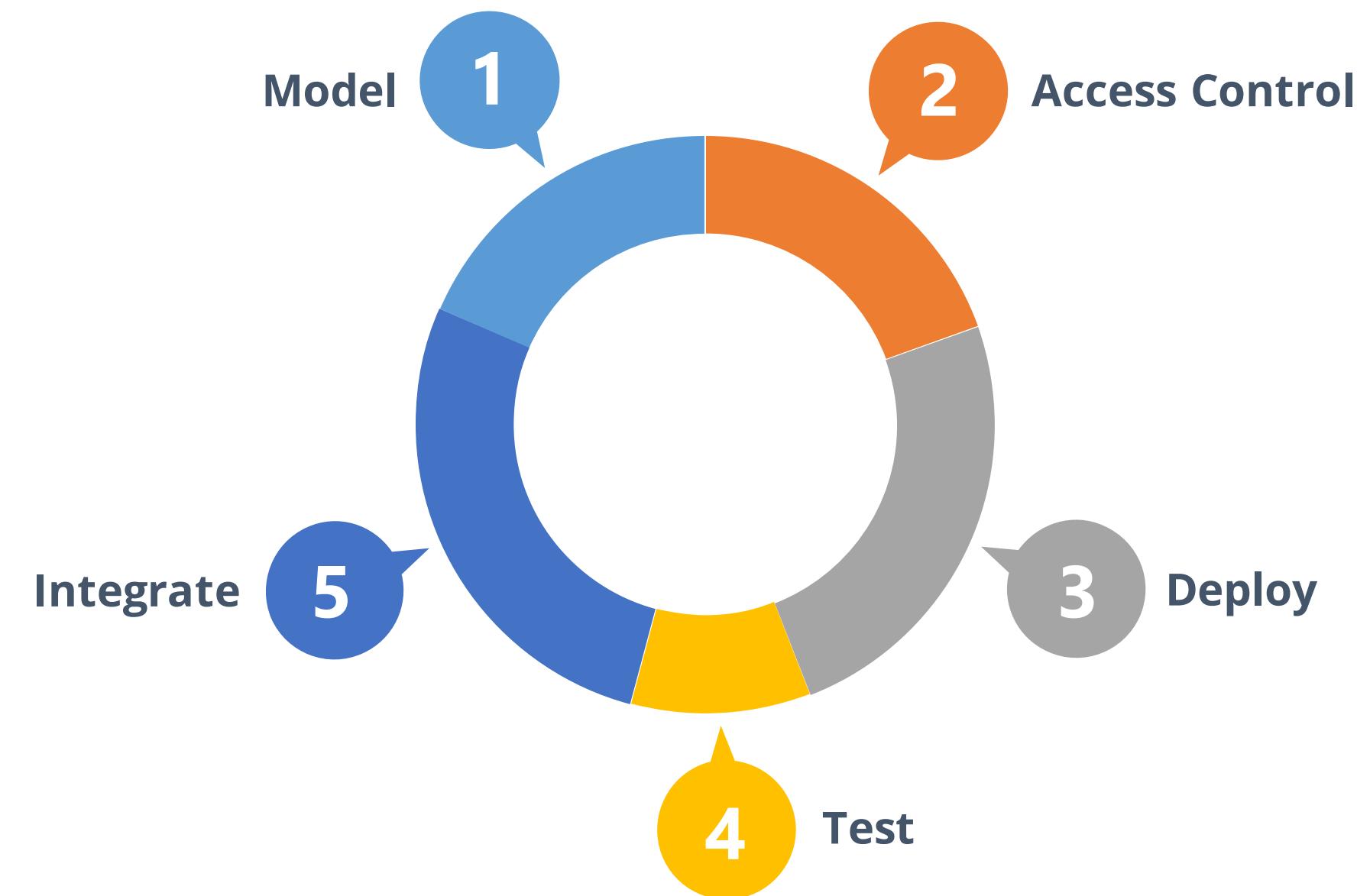
Set up your IDE

**Step 05**

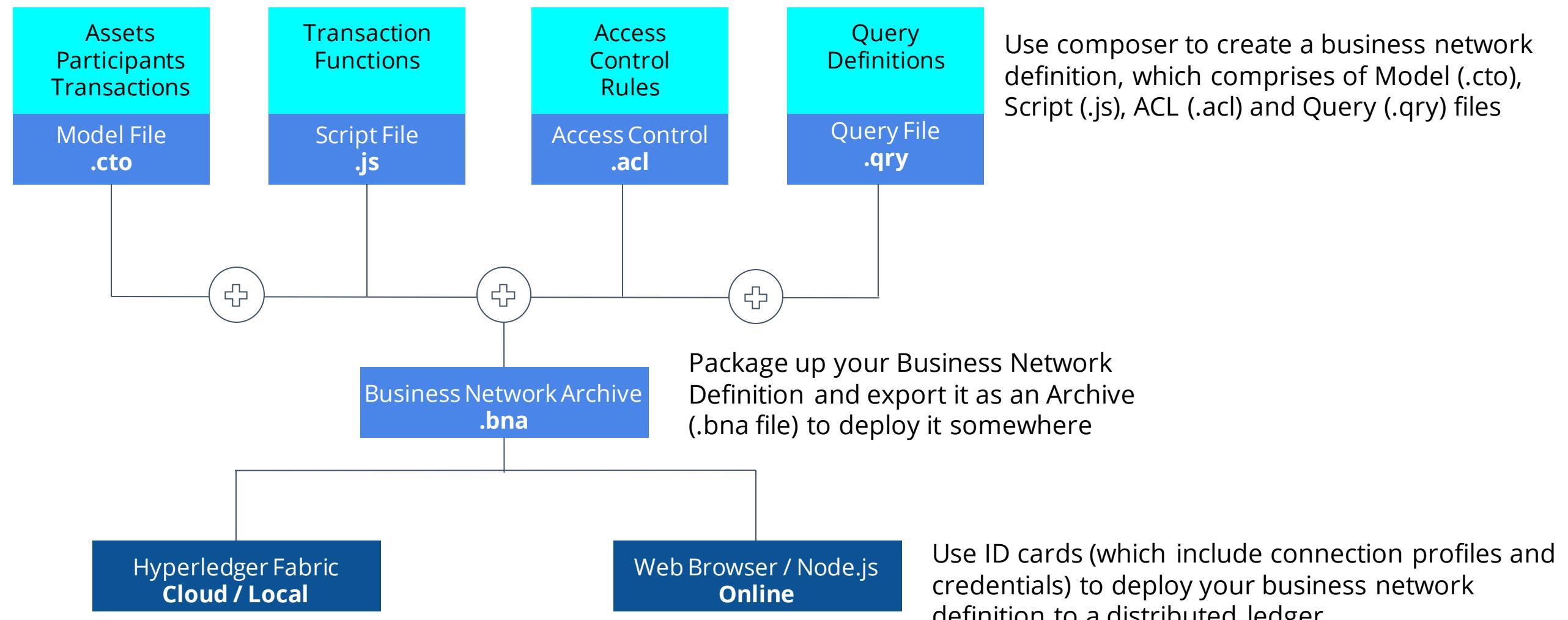
Install Hyperledger Fabric

# Blockchain Business Network

It is a decentralized network that is used to transfer business assets securely and efficiently with the help of distributed ledgers. The network is deployed in the following steps:



# Business Network Template



# Testing Business Network

There are four ways of testing a business network:



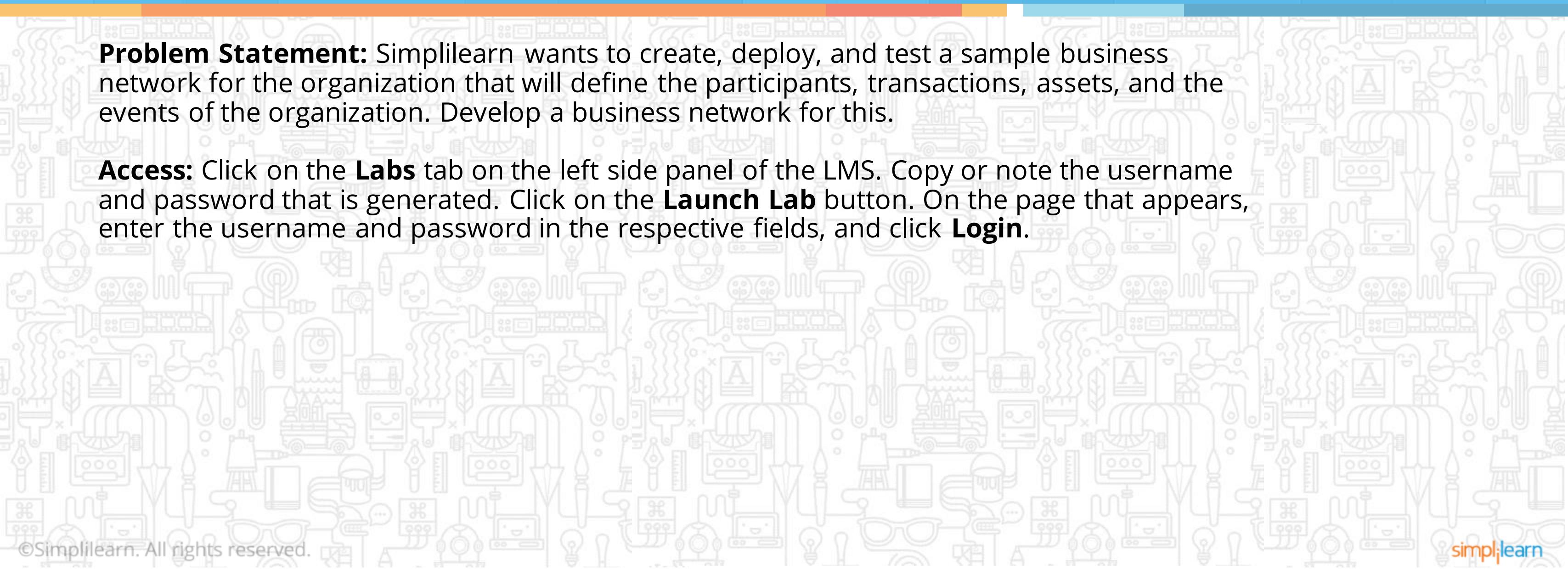
## Assisted Practice

Duration: 20 mins

### Create, Deploy, and Test a Business Network

**Problem Statement:** Simplilearn wants to create, deploy, and test a sample business network for the organization that will define the participants, transactions, assets, and the events of the organization. Develop a business network for this.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



## Assisted Practice: Steps



**Step 01**

Generate a business network definition

**Step 02**

Install the CLI tools

**Step 03**

Create the business network administrator

**Step 04**

Deploy the business network

**Step 02**

Perform Cucumber testing

# Unassisted Practice

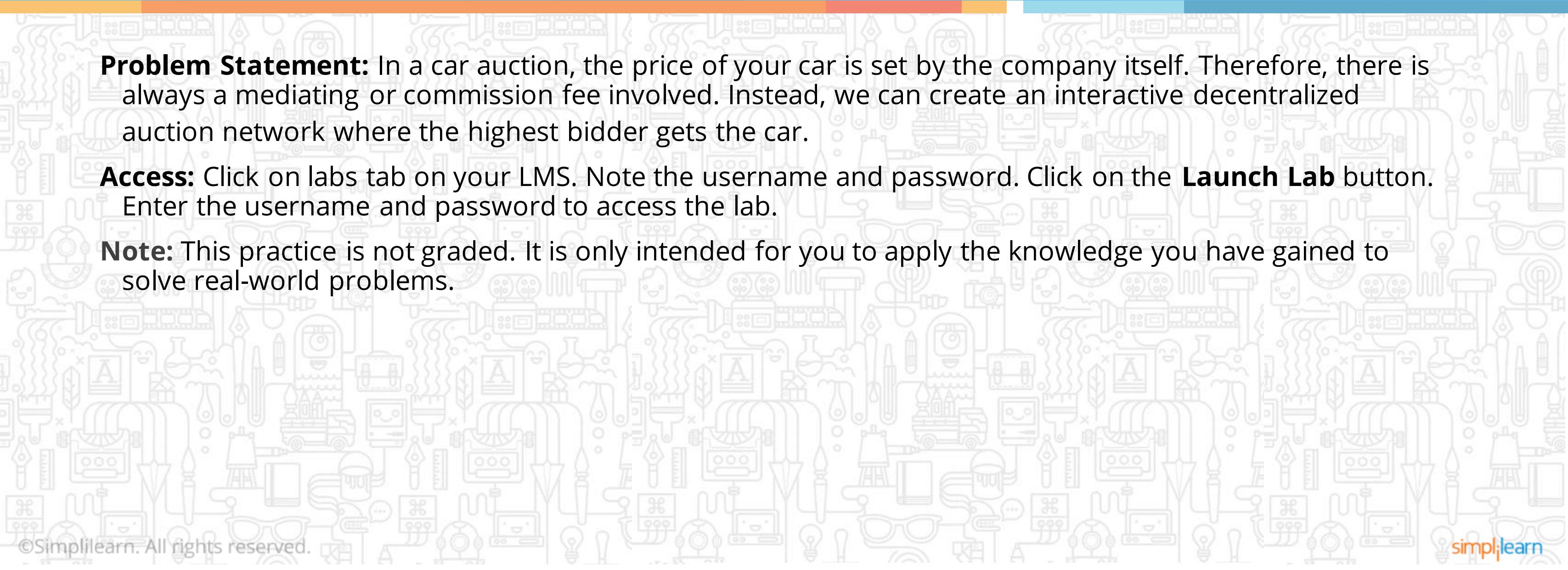
## Deploy a Car Auction Business Network

Duration: 20 mins

**Problem Statement:** In a car auction, the price of your car is set by the company itself. Therefore, there is always a mediating or commission fee involved. Instead, we can create an interactive decentralized auction network where the highest bidder gets the car.

**Access:** Click on labs tab on your LMS. Note the username and password. Click on the **Launch Lab** button. Enter the username and password to access the lab.

**Note:** This practice is not graded. It is only intended for you to apply the knowledge you have gained to solve real-world problems.



# Use Case Solution



Step 1

Create a new participant in the Auctioneer participant registry

Step 2

Step 3

Step 4

Step 5

Step 6

```
{  
    "$class": "org.acme.vehicle.auction.Auctioneer",  
    "email": "auction@acme.org",  
    "firstName": "Jenny",  
    "lastName": "Jones"  
}
```

# Use Case Solution



Step 1

Create two participants in the Member participant registry

Step 2

```
{  
    "$class": "org.acme.vehicle.auction.Member",  
    "balance": 5000,  
    "email": "memberA@acme.org",  
    "firstName": "Amy",  
    "lastName": "Williams"  
}
```

Step 3

```
{  
    "$class": "org.acme.vehicle.auction.Member",  
    "balance": 5000,  
    "email": "memberB@acme.org",  
    "firstName": "Billy",  
    "lastName": "Thompson"  
}
```

Step 4

Step 5

Step 6

# Use Case Solution



Step 1

Create a new asset of vehicle owned by memberA@acme.org in the Vehicle asset registry

Step 2

Step 3

Step 4

Step 5

Step 6

```
{  
  "$class": "org.acme.vehicle.auction.Vehicle",  
  "vin": "vin:1234",  
  "owner":  
    "resource:org.acme.vehicle.auction.Member#memberA@acme.org"  
}
```

# Use Case Solution



Step 1

Create a vehicle listing for car vin:1234 in the VehicleListing asset registry

Step 2

Step 3

Step 4

Step 5

Step 6

```
{  
    "$class": "org.acme.vehicle.auction.VehicleListing",  
    "listingId": "listingId:ABCD",  
    "reservePrice": 3500,  
    "description": "Arium Nova",  
    "state": "FOR_SALE",  
    "vehicle":  
        "resource:org.acme.vehicle.auction.Vehicle#vin:1234"  
}
```

# Use Case Solution

Step 1

Submit an Offer transaction

Step 2

```
{  
    "$class": "org.acme.vehicle.auction.Offer",  
    "bidPrice": 2000,  
    "listing":  
        "resource:org.acme.vehicle.auction.VehicleListing#listingId:ABCD",  
        "member":  
            "resource:org.acme.vehicle.auction.Member#memberA@acme.org"  
}
```

Step 3

```
{  
    "$class": "org.acme.vehicle.auction.Offer",  
    "bidPrice": 3500,  
    "listing":  
        "resource:org.acme.vehicle.auction.VehicleListing#listingId:ABCD",  
        "member":  
            "resource:org.acme.vehicle.auction.Member#memberB@acme.org"  
}
```

Step 4

Step 5

Step 6

# Use Case Solution



Step 1

Step 2

Step 3

Step 4

Step 5

Step 6

End the transaction

```
{  
    "$class": "org.acme.vehicle.auction.CloseBidding",  
    "listing":  
        "resource:org.acme.vehicle.auction.VehicleListing#listingId:ABCD"  
}
```

# Key Takeaways



You are now able to:

- ➊ Describe Hyperledger Fabric and Hyperledger Composer
- ➋ Set up a development environment using Hyperledger Composer
- ➌ Create, deploy, and test a business network



## Knowledge Check



Knowledge  
Check

1

**Which of the following is a program that initiates a transaction?**

- a. Ordering service
- b. Fabric peer
- c. Client application
- d. Chaincode



Knowledge  
Check

1

**Which of the following is a program that initiates a transaction?**

- a. Ordering service
- b. Fabric peer
- c. Client application
- d. Chaincode



The correct answer is **d**

**Chaincode is a program that initiates the transaction and manages the ledger state.**

Knowledge  
Check

2

**In which channel network do all peers maintain the same ledger?**

- a. Single-Channel network
- b. Multi-Channel network
- c. Both a and b
- d. None of the above



## In which channel network do all peers maintain the same ledger?

- a. Single-Channel network
- b. Multi-Channel network
- c. Both a and b
- d. None of the above



The correct answer is

**a**

**In single channel network, all peers connect to the same system channel. Hence, it maintains the same ledger.**

## Lesson-End Project

Duration: 30 mins

### Deploy an Animal Tracking Business Network

#### **Problem Statement:**

The government farming regulators are not able to track the locations of all animals and their movements between farms. You are supposed to create and deploy a business network to solve the problem. The business network should define the following:

- Participants: farmers and regulator
- Assets: animal, business, and field
- Transactions: AnimalMovementDeparture, AnimalMovementArival, and SetupDemo

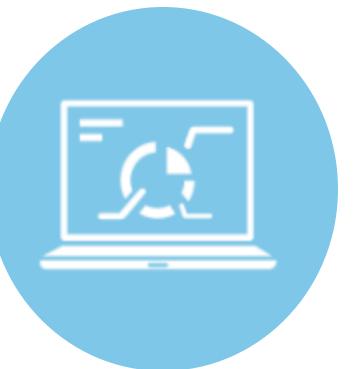
**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 7: Blockchain on Multichain



# Learning Objectives

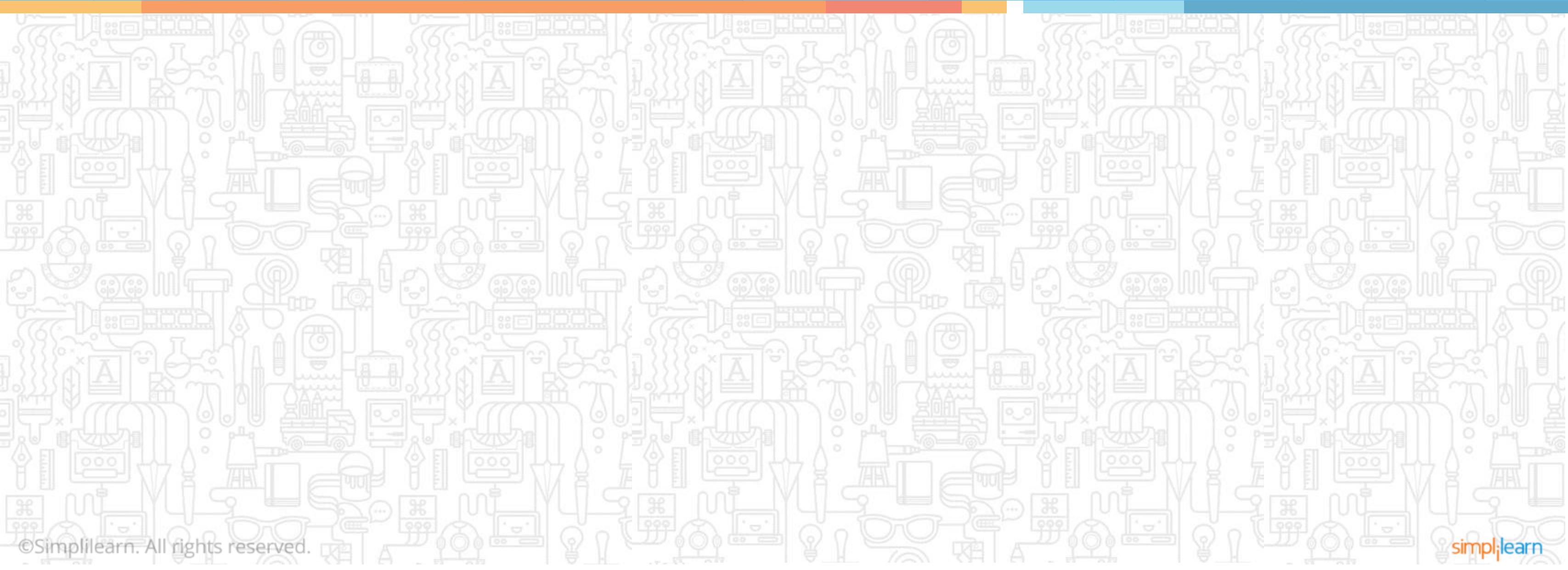


By the end of this lesson, you will be able to:

- Create and deploy your first private Blockchain using Multichain
- Create an asset in Multichain
- Publish data to the stream
- Perform mining in Multichain using round-robin consensus scheme
- Create private stock exchange on Multichain

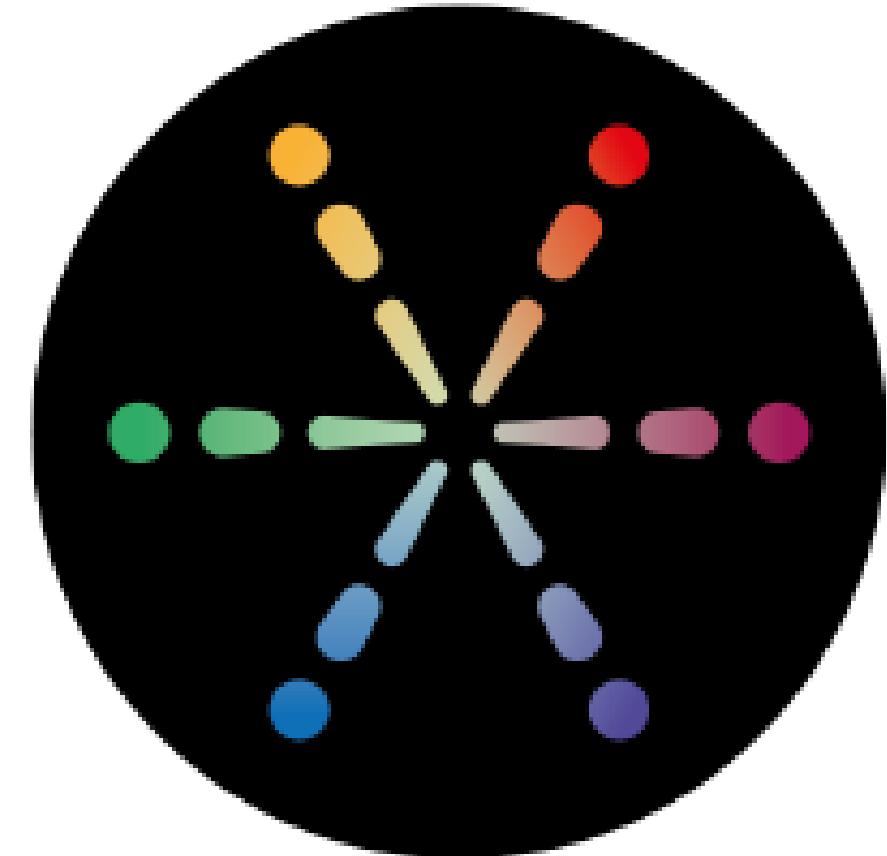
# Blockchain on Multichain

## Overview of Multichain



# Introduction to Multichain

Multichain technology is a simple yet powerful platform for creating and deploying private Blockchain for enterprise.



# Objectives of Multichain

To ensure the visibility of Blockchain's activity within the chosen participants



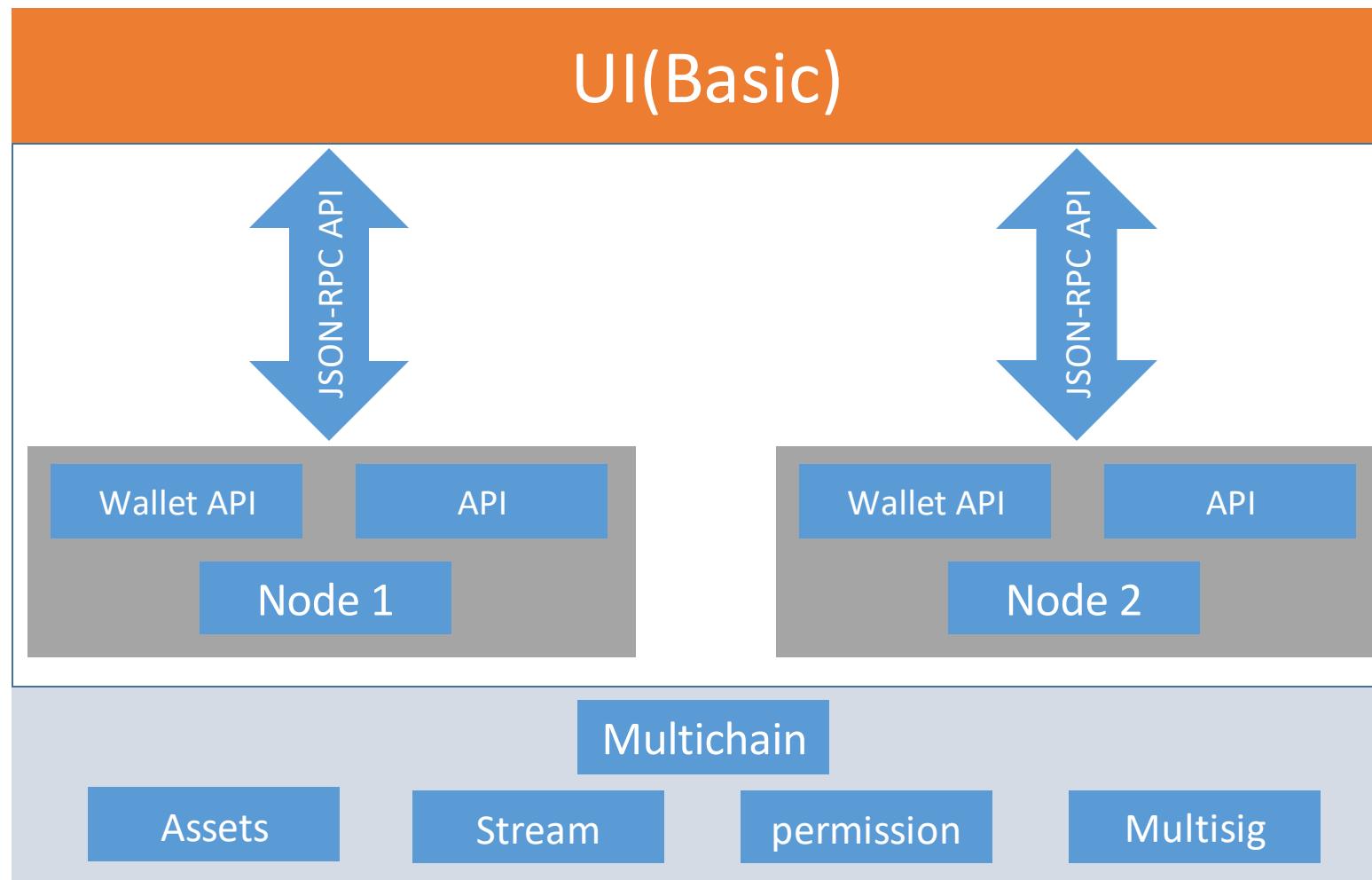
To enable secure mining without proof of work

To ensure stability and control over transactions

To store transactions related only to participants

# Multichain Architecture

In Multichain, every node has a separate application programming interface (API) to connect to each user interface(UI).



# Assisted Practice

## Create a Private Blockchain

Duration: 10 mins

**Problem Statement:** Develop and deploy a private Blockchain for a financial institution.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

# Steps to Create Multichain

---

**Step 01**

Create a new Blockchain with a name of your choice

**Step 02**

Check the Blockchain's default settings in params.dat file

**Step 03**

Initialize the Blockchain

**Step 04**

Connect to the Blockchain using the second server

**Step 05**

Add connection permission to the address of the first server

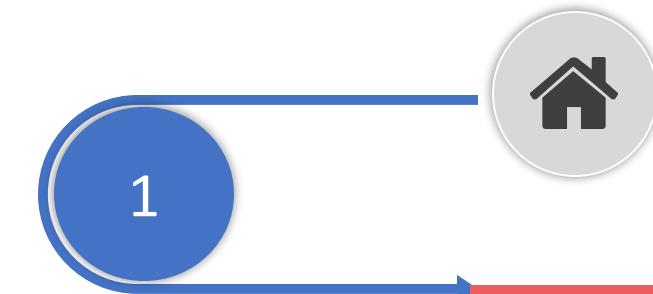
**Step 06**

Reconnect using the second server

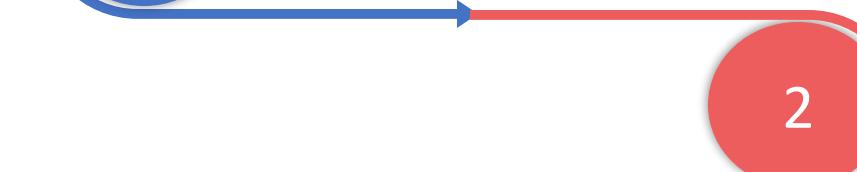
# The Hand-Shaking Process

Hand-shaking in Multichain occurs when two nodes or hubs connect.

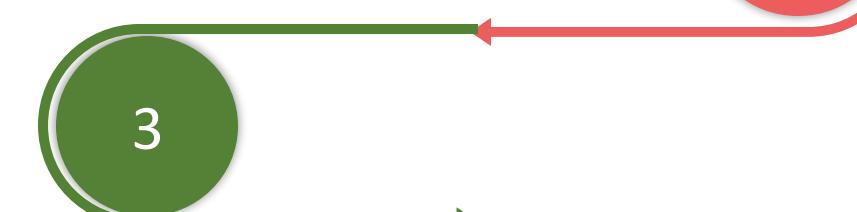
Each node presents its public address on the permitted list



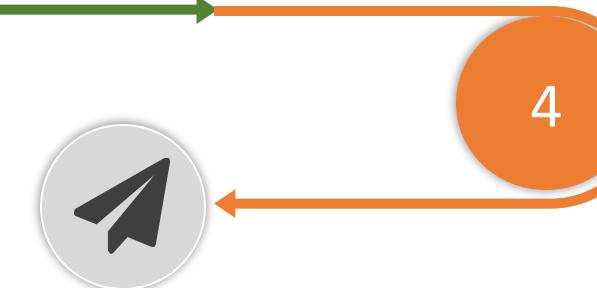
A challenge message is sent by each node to the other node



Each node checks if other's address exists on its version of permitted list



The other node has to sign the message with the private key of the presented address

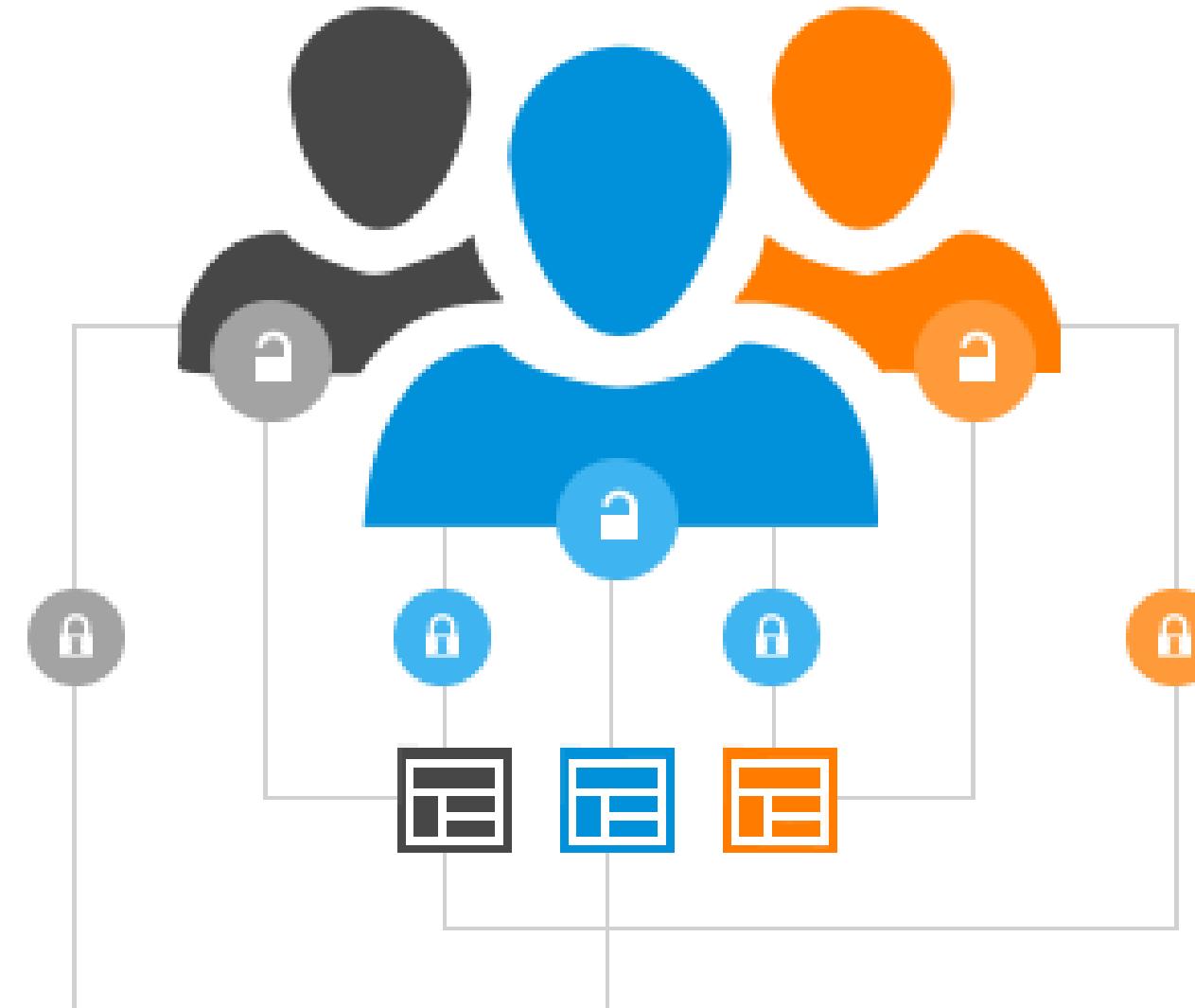


The peer-to-peer connection aborts if either of the nodes is not satisfied with the results.

# Permission in Multichain

Multichain provides the following permission:

- Connect to network
- Send and receive transactions
- Write to a stream
- Issue assets
- Create streams
- Add blocks to chain
- Change permission by consensus



# Features of Assets in Multichain

---



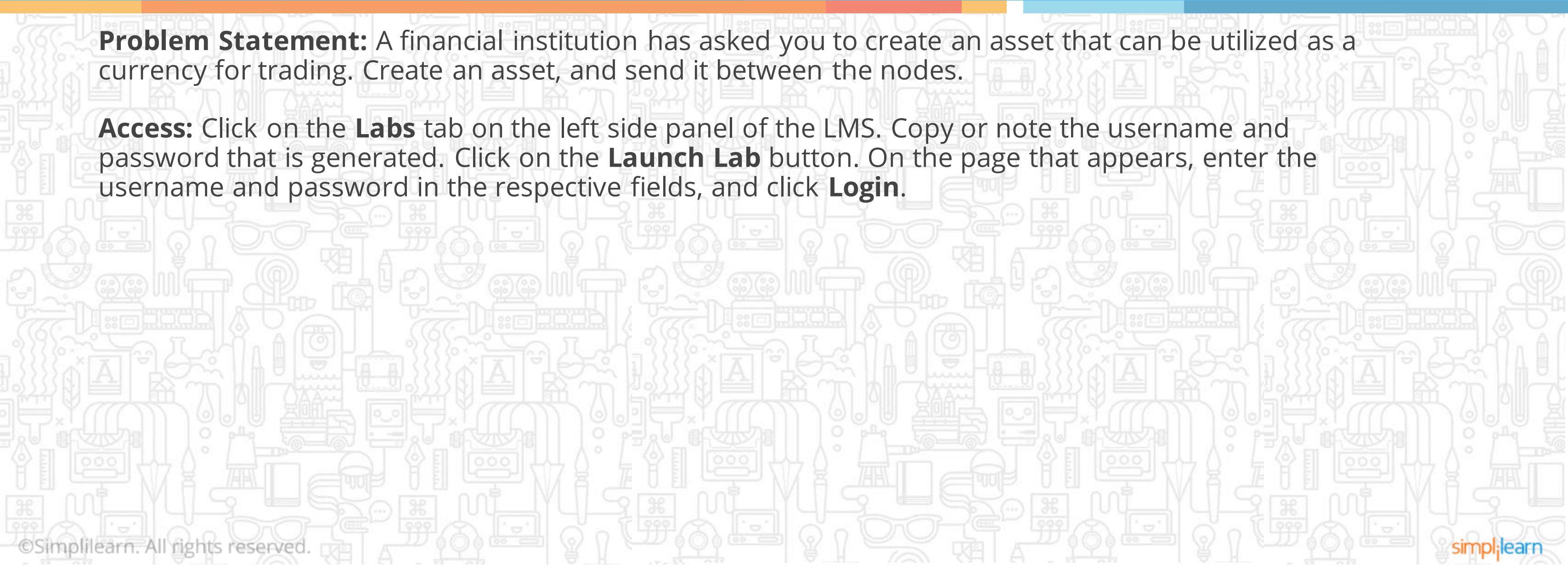
# Assisted Practice

## Create an Asset in Multichain

Duration: 7 mins

**Problem Statement:** A financial institution has asked you to create an asset that can be utilized as a currency for trading. Create an asset, and send it between the nodes.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



# Steps to Create Assets in Multichain

---

Step 1: Switch to interactive mode

Step 2: Check for permission assigned to the wallet's address

Step 3: Get the address that has permission to create an asset using the first server

Step 4: Create a new asset

Step 5: Verify that the asset is listed using both the servers

## Steps to Create Assets in Multichain(Contd.)

---

Step 6: Check asset balances using both the servers

Step 7: Send some units of the asset to the second server's wallet using the first server

Step 8: Grant permission to the address of second server's wallet to send and receive the asset

Step 9: Send asset to the address

Step 10: Check asset balances, and view transaction on each server

# Multichain Streams

---

- Multichain streams enable a Blockchain to be used as a general purpose append-only database
- They provide an abstraction for Blockchain use cases on general data retrieval, timestamping, and archiving

Streams can be used to implement three types of databases on a chain:

- 01 A key-value database or document store, in the style of NoSQL
- 02 A time series database, which focuses on the ordering of entries
- 03 An identity-driven database where entries are classified according to their author

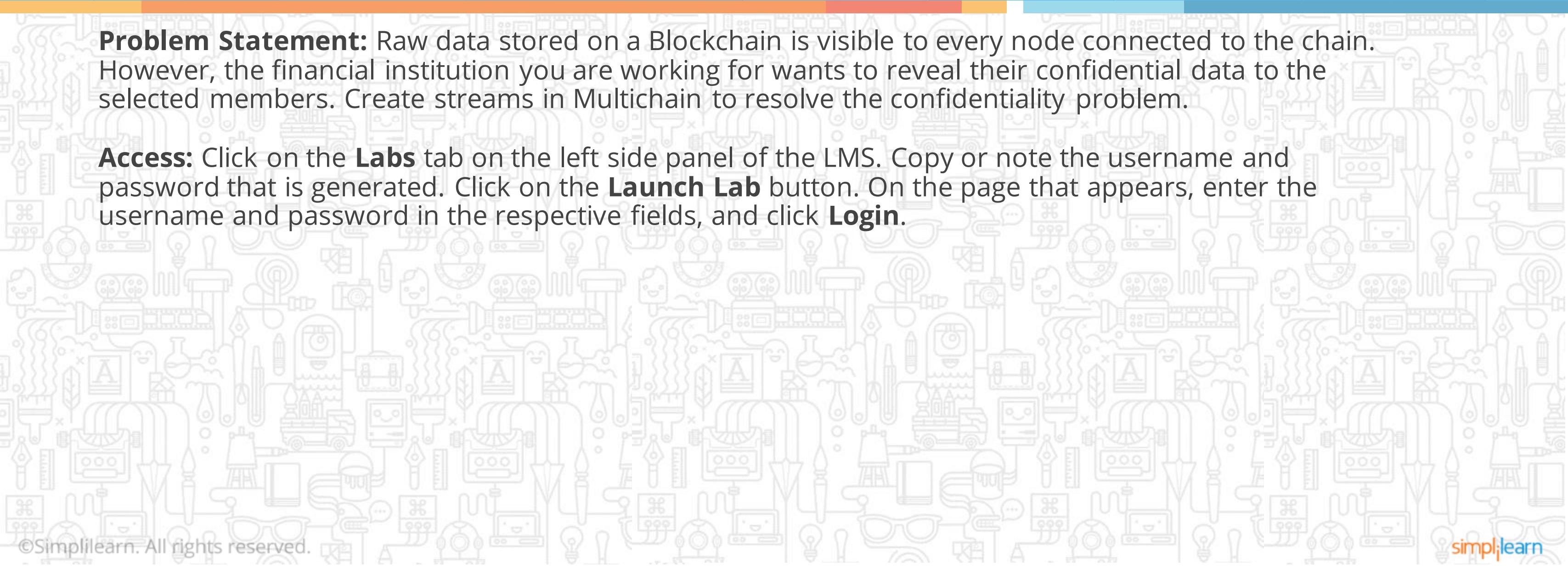
# Assisted Practice

Duration: 10 mins

## Create Streams in Multichain

**Problem Statement:** Raw data stored on a Blockchain is visible to every node connected to the chain. However, the financial institution you are working for wants to reveal their confidential data to the selected members. Create streams in Multichain to resolve the confidentiality problem.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



# Steps to Create Streams in Multichain

---

- 
- Step 1: Create a stream for data storage and retrieval
  - Step 2: Check permission of the stream
  - Step 3: Publish something, such as your name, to the stream using a key
  - Step 4: Check if the stream is visible on another server
  - Step 5: Allow the second server to subscribe to the stream and view its content
  - Step 6: Allow the second server to publish to the stream
  - Step 7: Check the stream contents

# Consensus in Multichain

Multichain has distributed consensus between identified block validators. There is one validator per block, working in a round-robin system.

The following are the features of consensus in Multichain:



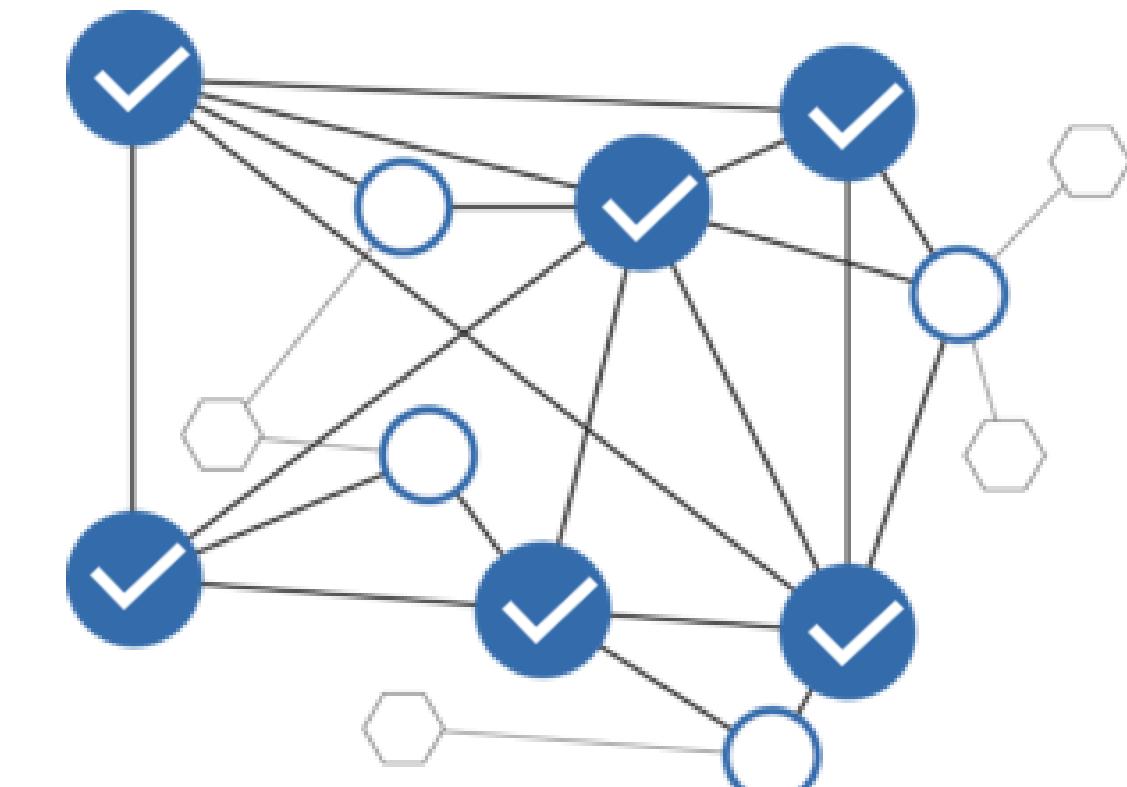
**Only permissioned nodes are allowed to mine**



**Distributed consensus has mining diversity**



**There is no proof of work or cryptocurrency**



# Mining in Multichain

- Multichain restricts the number of blocks that may be created by the same miner within a given time frame
- Multichain uses a parameter called mining diversity, which defines the strictness in the mining system
- Mining in Multichain uses a randomized round-robin system for block adders
- Mining diversity can be between 0-1



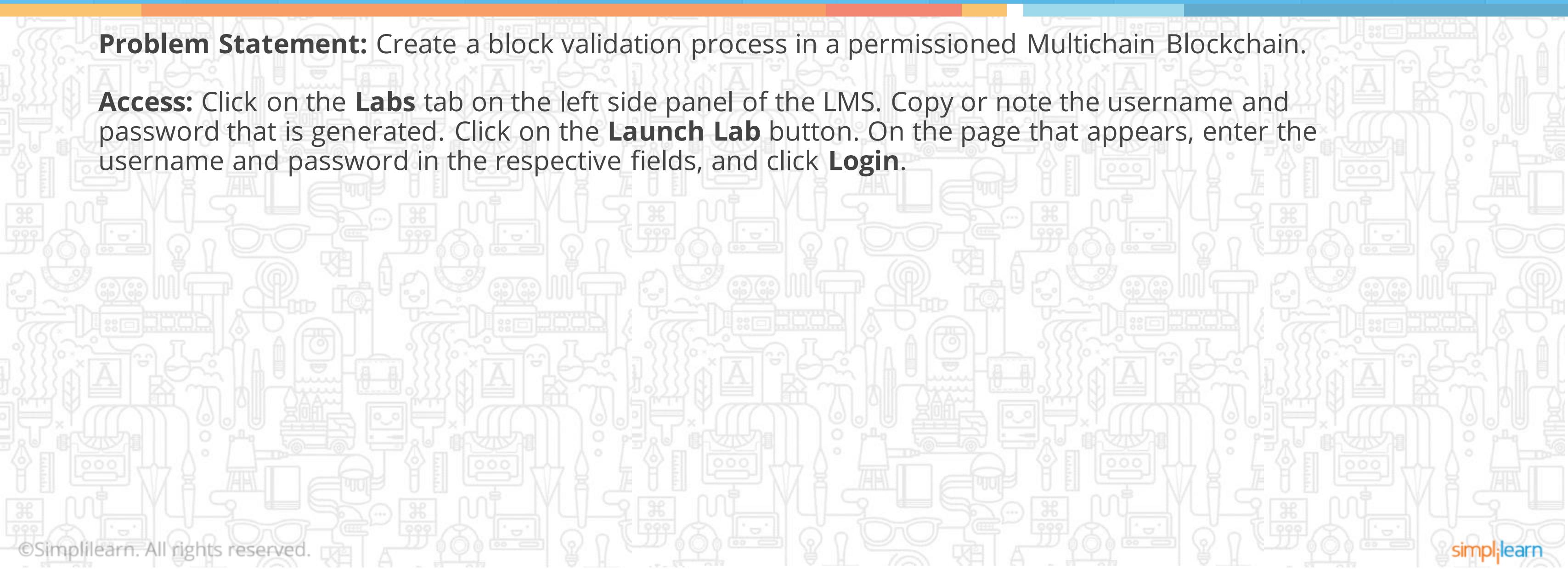
## Assisted Practice

Duration: 10 mins

### Perform Mining in Multichain

**Problem Statement:** Create a block validation process in a permissioned Multichain Blockchain.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



# Steps for Mining in Multichain

---

Step 1

Grant mining permission to both the nodes

Step 2

Check that the two permitted mining nodes are listed

Step 3

Set mining-turnover to maximum to allow all the nodes to create a block

Step 4

Set run-time parameters

Step 5

Check current block height and information of last few blocks

# Multichain Flexibility

---

Multichain provides a fair level of flexibility:

It has more than 45 Blockchain parameters

It has an option to change permission by consensus

It has unified JSON-RPC API for applications

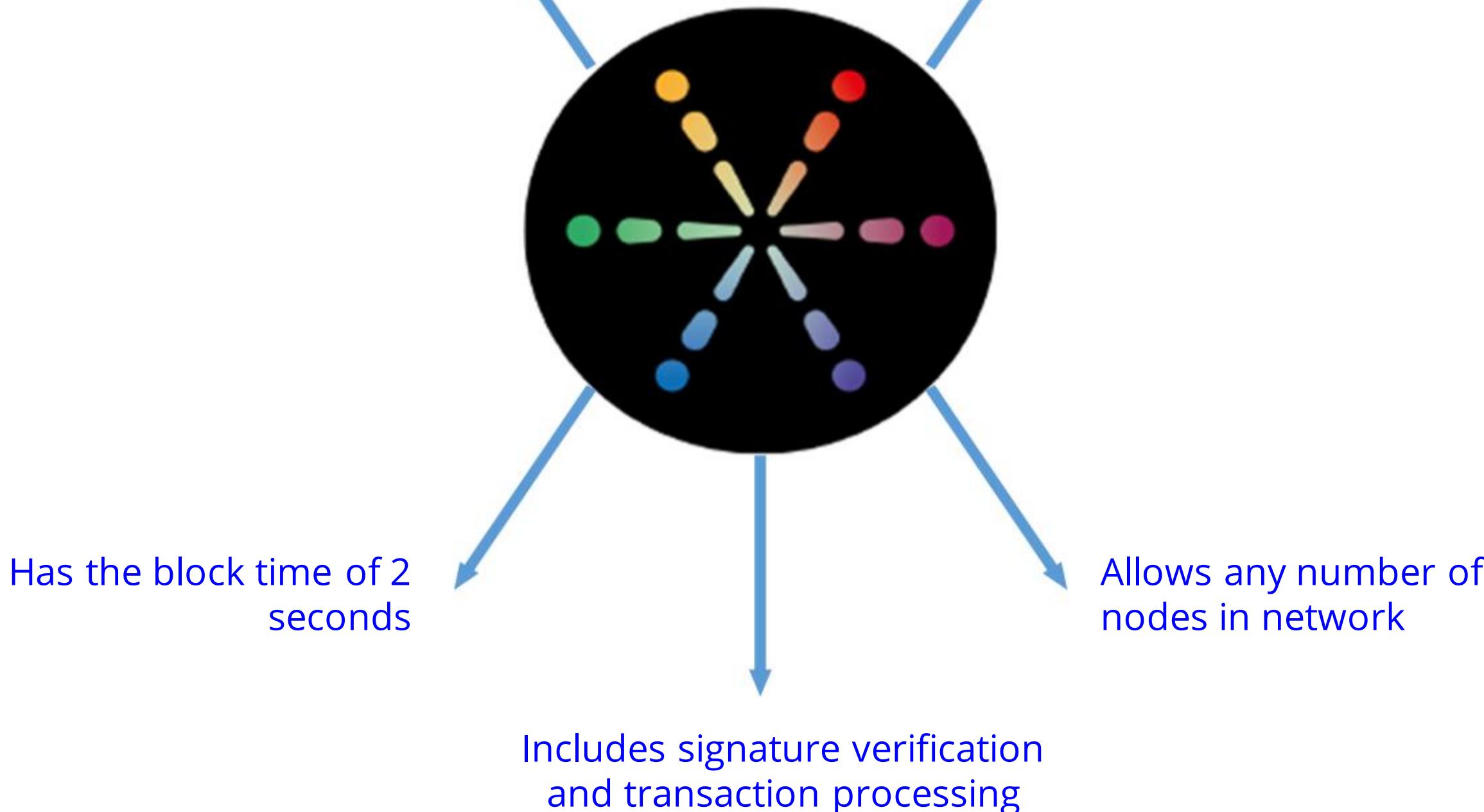
It allows nodes to be added or removed from the network

# Multichain: Speed and Scalability

---

Can have millions of addresses, assets, and streams

Can process unlimited transactions or stream items



## Unassisted Practice

### Stock Exchange on Multichain Platform

Duration: 20 mins

**Problem Statement:** The stock exchange model is centralized, slow, and expensive. Hence, optimization is required. Develop a Multichain based stock exchange market, which makes the system decentralized, simple, efficient, fast, transparent, and secure.

**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.

**Note:** This practice is not graded. It is only intended for you to apply the knowledge you have gained to solve real-world problems.

# Steps to Create Multichain based Stock Exchange

- Step 01** Create a private Blockchain
- Step 02** Connect to the Blockchain
- Step 03** Create assets
- Step 04** Transfer assets to the peer



# Steps to Create Stock Exchange on Multichain

---

Step 1

- Run the following command using the first server: `multichain-util create stockexchange`

Step 2

- Run the following command to initialize the Blockchain:

`multichainind stockexchange -daemon`

(The server will start, and a message will appear saying “genesis block found”)

Step 3

Step 4

# Steps to Create Stock Exchange on Multichain

Step 1

- Run the following command using the second server to connect to this Blockchain:  
`multichainind stockexchange@[ip-address]:[port]`

Step 2

- Run the following command using the first server to add permission for the new address:  
`multichain-cli stockexchange grant <walletaddress> connect`

Step 3

- Run the following command and try reconnecting using the second server:  
`multichainind stockexchange -daemon`

Step 4

# Steps to Create Stock Exchange on Multichain

Step 1

- Run the following command using the first server to get the address that has the permission to create an asset pool:  
`multichain-cli stockexchange listpermission issue`

Step 2

- Run the following command to create the company shares:  
`multichain-cli stockexchange issue <peer-address> SimpliLtd 1000 0.01`  
(1000 are the units, and they are divisible by 0.01)

Step 3

- Run the following command using either of the servers to confirm the listing of the asset:  
`multichain-cli stockexchange listassets`

Step 4

# Steps to Create Stock Exchange on Multichain

---

Step 1

- Run the following command to transfer some shares to the second server:

```
multichain-cli stockexchange sendassettoaddress  
<second_node_address> SimpliLtd 501
```

Step 2

- Run the following command to confirm the transfer of ownership on both the servers:

```
multichain-cli stockexchange gettotalbalances 0
```

Step 3

Step 4

# Key Takeaways



You are now able to:

- ✔ Create and deploy your first private Blockchain using Multichain
- ✔ Create an asset in Multichain
- ✔ Publish data to the stream
- ✔ Perform mining in Multichain using round-robin consensus scheme
- ✔ Create private stock exchange on Multichain



## Knowledge Check



Knowledge  
Check

1

**Multichain ensures that Blockchain activities are visible to all the participants.**

- a. True
- b. False



Knowledge  
Check

1

**Multichain ensures that Blockchain activities are visible to all the participants.**

- a. True
- b. False



The correct answer is **b**

**Multichain ensures that the visibility of Blockchain's activity is within the chosen participants.**

Knowledge  
Check

2

**Multichain has a general API, which is used by each node to connect to an application.**

- a. True
- b. False



**Multichain has a general API, which is used by each node to connect to an application.**

- a. True
- b. False



The correct answer is **b**

**Multichain doesn't have a general API. Each node has a separate API to connect to UI.**

## Which of the following is not applicable to Multichain?

- a. Multi-way atomic asset exchanges
- b. Permissioned follow-on issuance
- c. Need for smart contracts
- d. Flexible asset metadata



## Which of the following is not applicable to Multichain?

- a. Multi-way atomic asset exchanges
- b. Permissioned follow-on issuance
- c. Need for smart contracts
- d. Flexible asset metadata



The correct answer is

C

**There is no need for smart contracts in Multichain.**

## Which of the following is true about Multichain?

- a. Multichain cannot have millions of addresses, assets, streams
- b. Transaction verification doesn't include signature verification
- c. Multichain has block time as low as 10 seconds
- d. Multichain allows unlimited nodes in the network



## Which of the following is true about Multichain?

- a. Multichain cannot have millions of addresses, assets, streams
- b. Transaction verification doesn't include signature verification
- c. Multichain has block time as low as 10 seconds
- d. Multichain allows unlimited nodes in the network



The correct answer is

**d**

**Multichain can have any number of nodes in the network.**

# Lesson-End Project

Duration: 20 mins

## Create Private Multichain Blockchain

**Problem Statement:** Create a private Multichain with the following modifications in parameters in params.dat file:

1. **Basic chain parameters**

Provide chain description = “you can provide description to your blockchain”

2. **Global permissions**

Set “Anyone-can-connect= true”

3. **Native Blockchain currency**

Set “Initial-block-reward= 286730”(Set any value of your choice)

Set “Minimum relay fee= 130”

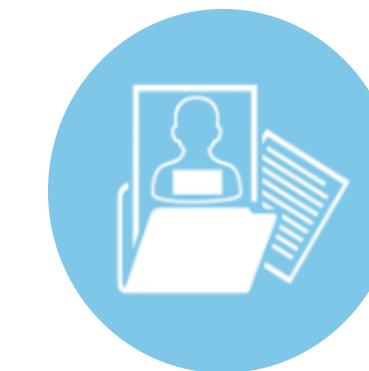
**Access:** Click on the **Labs** tab on the left side panel of the LMS. Copy or note the username and password that is generated. Click on the **Launch Lab** button. On the page that appears, enter the username and password in the respective fields, and click **Login**.



**Thank You**

# Blockchain

## Lesson 8: Blockchain Prospects



# Learning Objectives

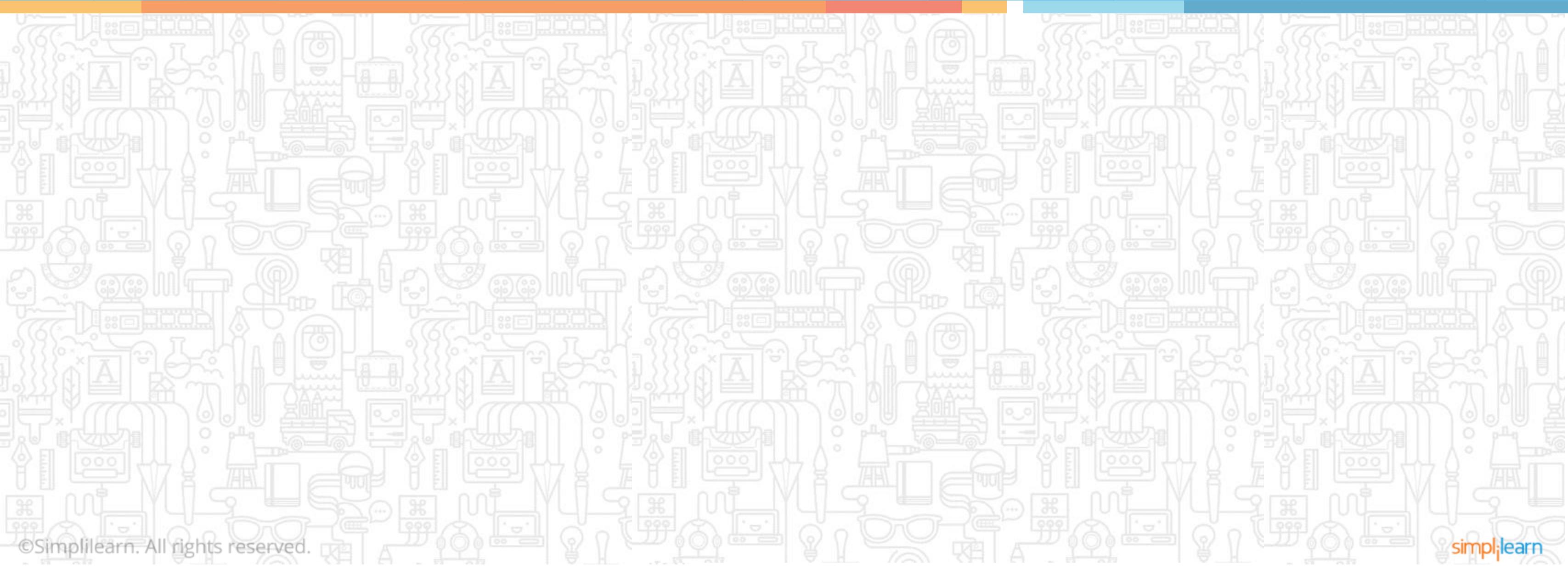


By the end of this lesson, you will be able to:

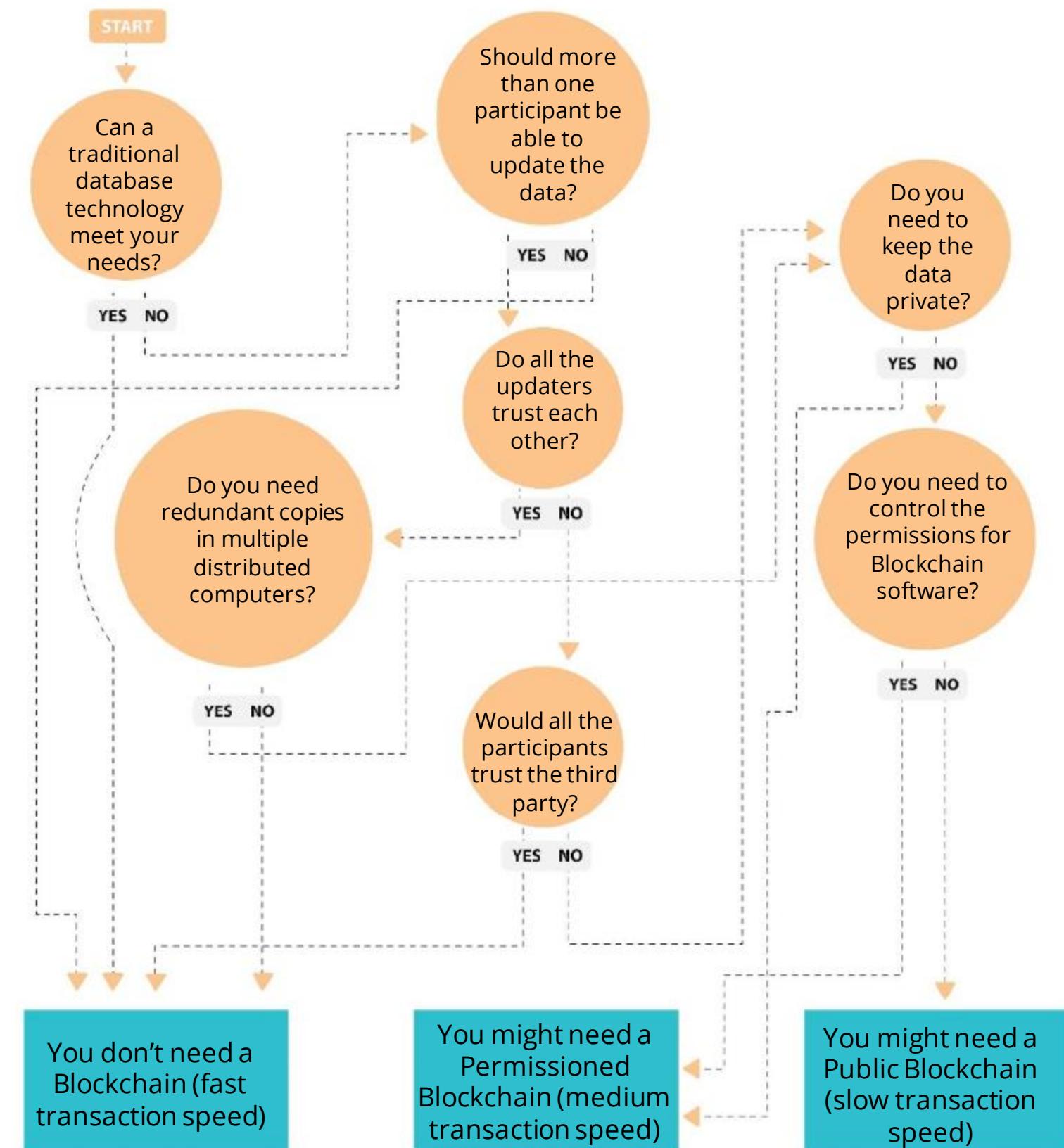
- ➊ Identify the use cases of Blockchain in healthcare industry
- ➋ Identify the use cases of Blockchain in government organization
- ➌ Identify the use cases of Blockchain in finance industry
- ➍ Identify the use cases of Blockchain in other industries worldwide

# Blockchain Prospects

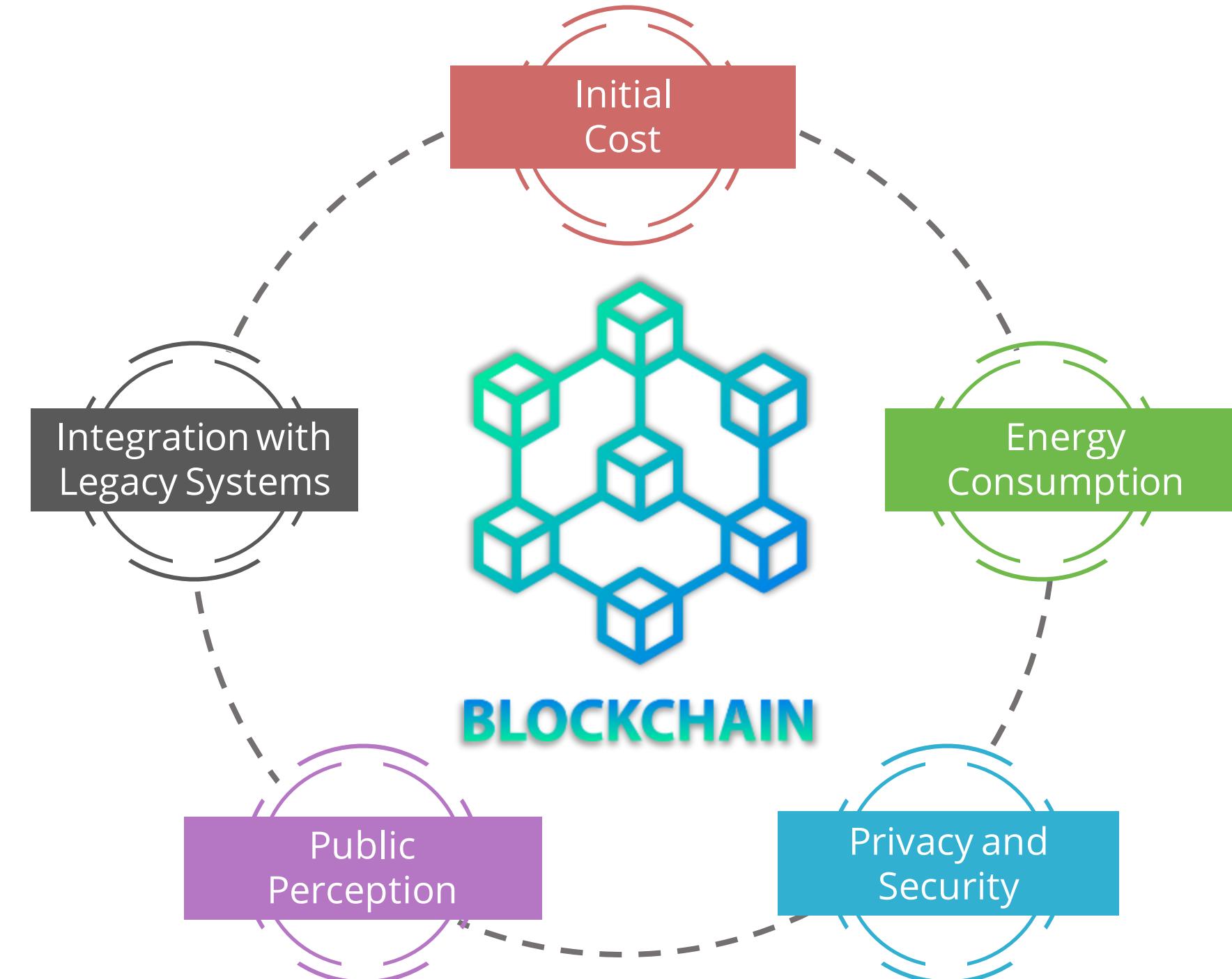
## Uses of Blockchain



# Do We Need Blockchain?

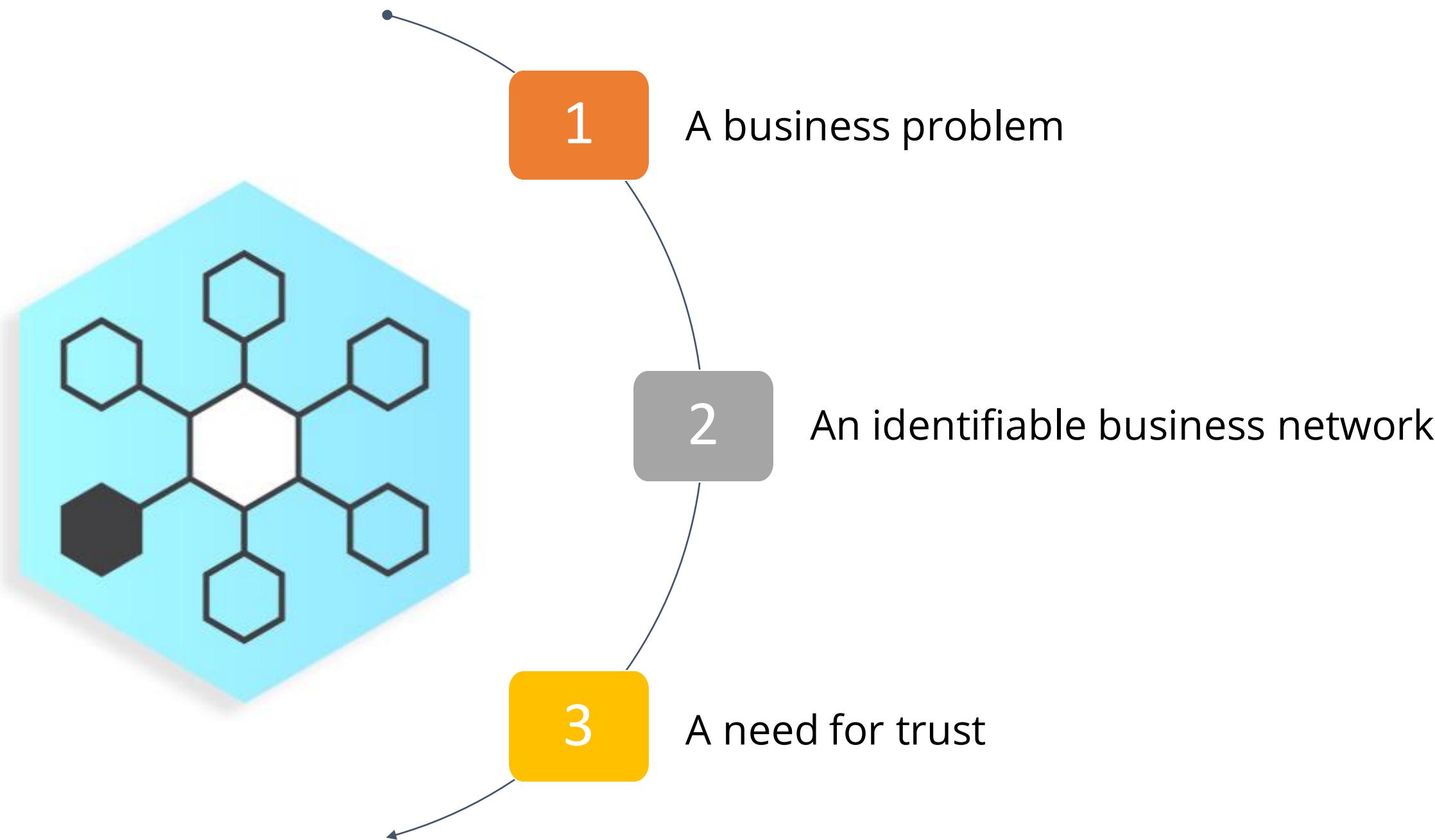


# Challenges in Blockchain



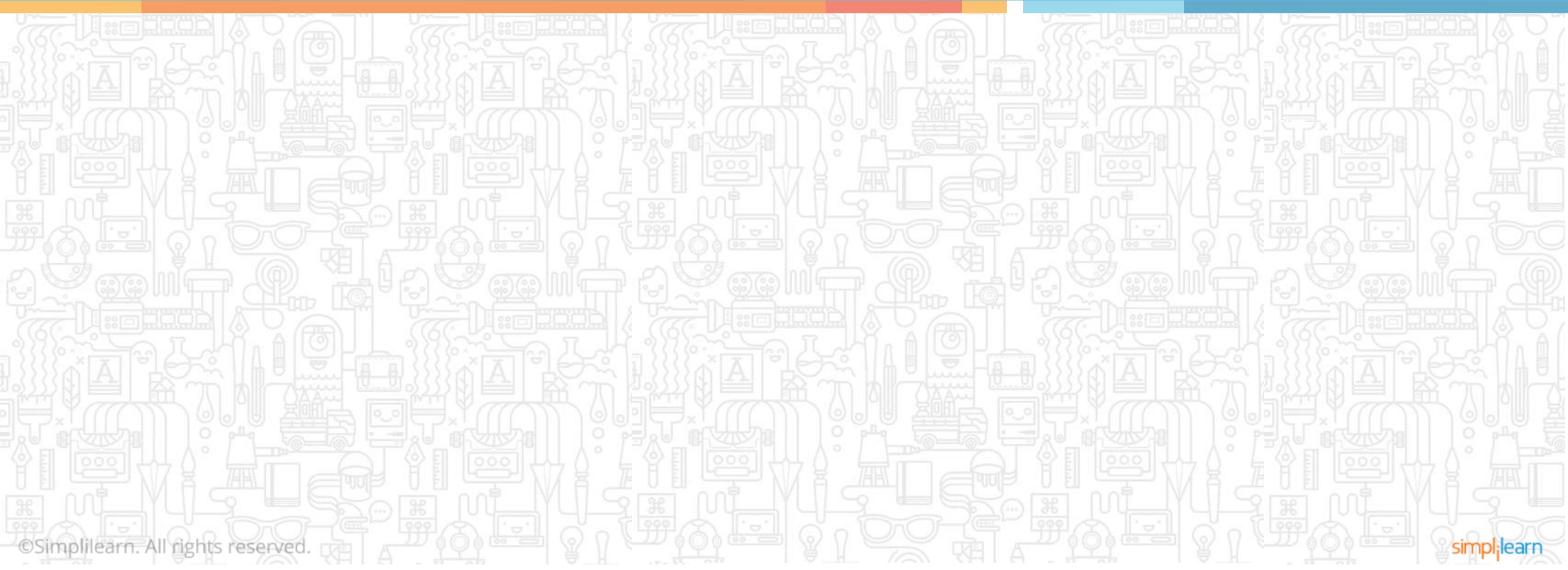
# Identifying a Blockchain Use Case

A good Blockchain use case should have the following:

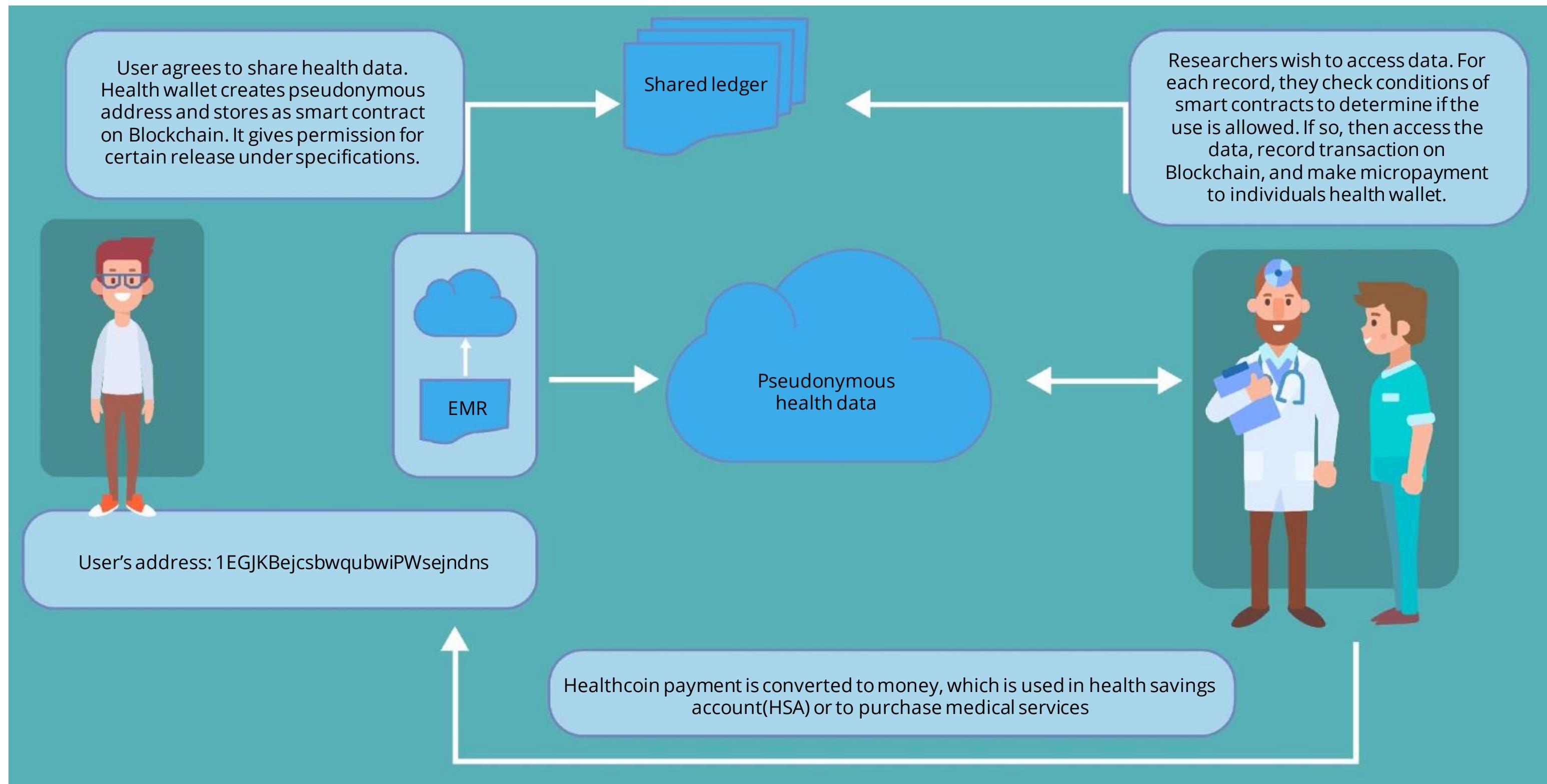


# Blockchain Prospects

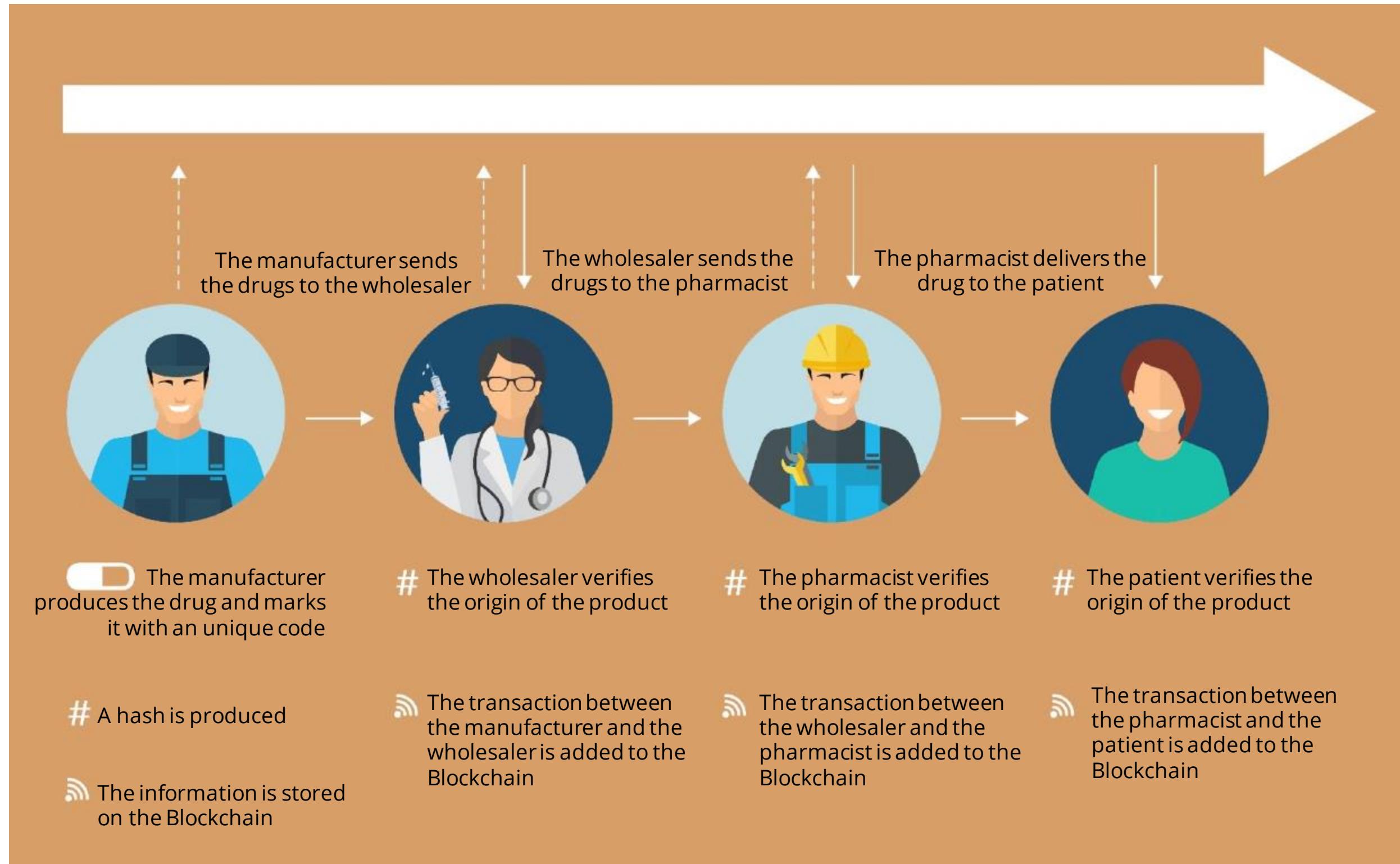
## HealthCare Use Cases



# Patient Data Management

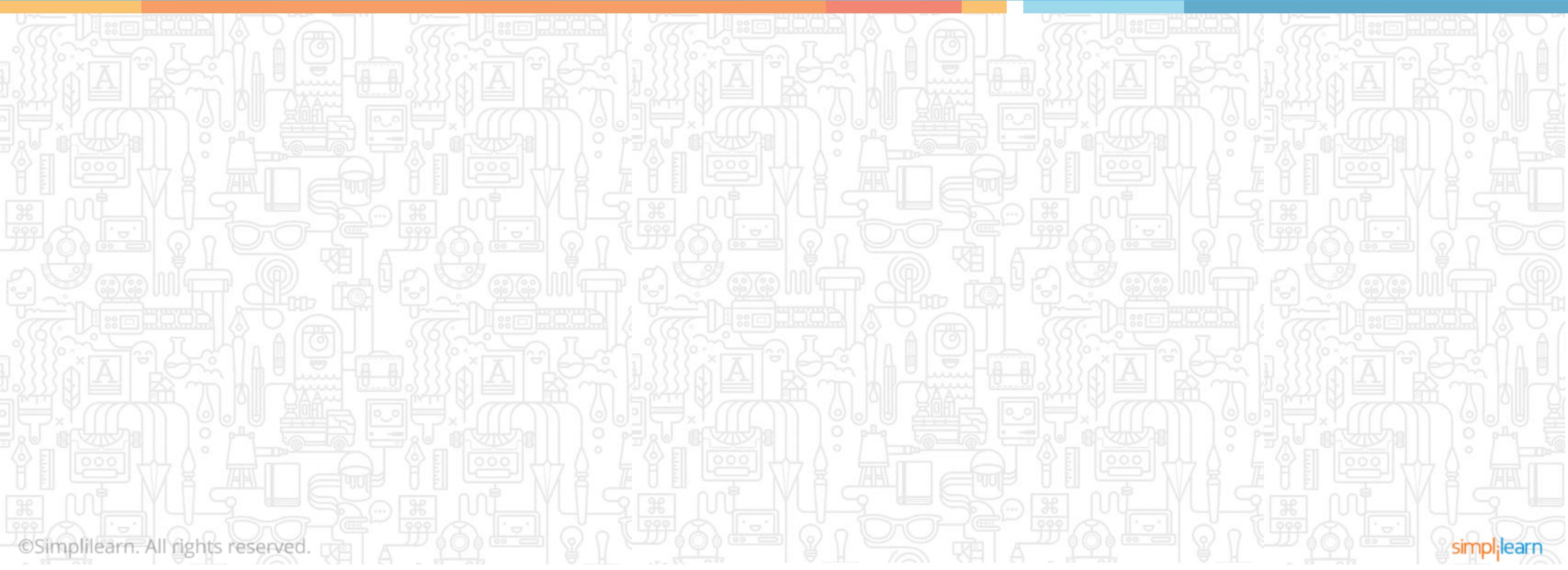


# Drug Traceability

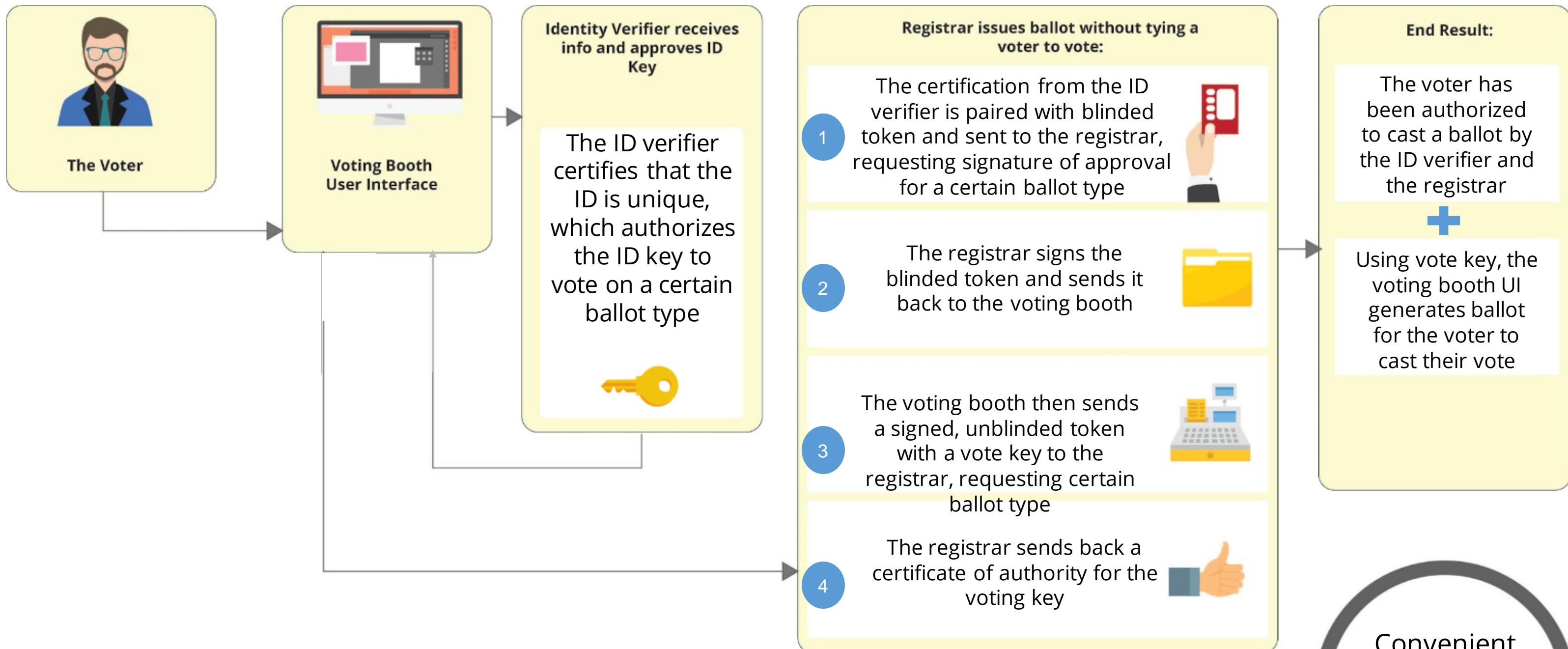


# Blockchain Prospects

## Government Use Cases



# Blockchain in Voting



Convenient  
notification is  
sent when  
your ballot is  
ready

# Blockchain Around the World

## Russia

Sberbank partnered with Russia's Federal Antimonopoly Service(FAS) to implement document transfer and storage using Blockchain

## South Korea

Dayli Financial Group(DFG) is developing a Blockchain based ecosystem called ICON. It will allow government, universities, hospitals, securities, and banks to interact without third-party networks

## India

IndiaChain is a pilot project by the government of India. It utilizes Blockchain technology for digitization and validation of educational degree certificates

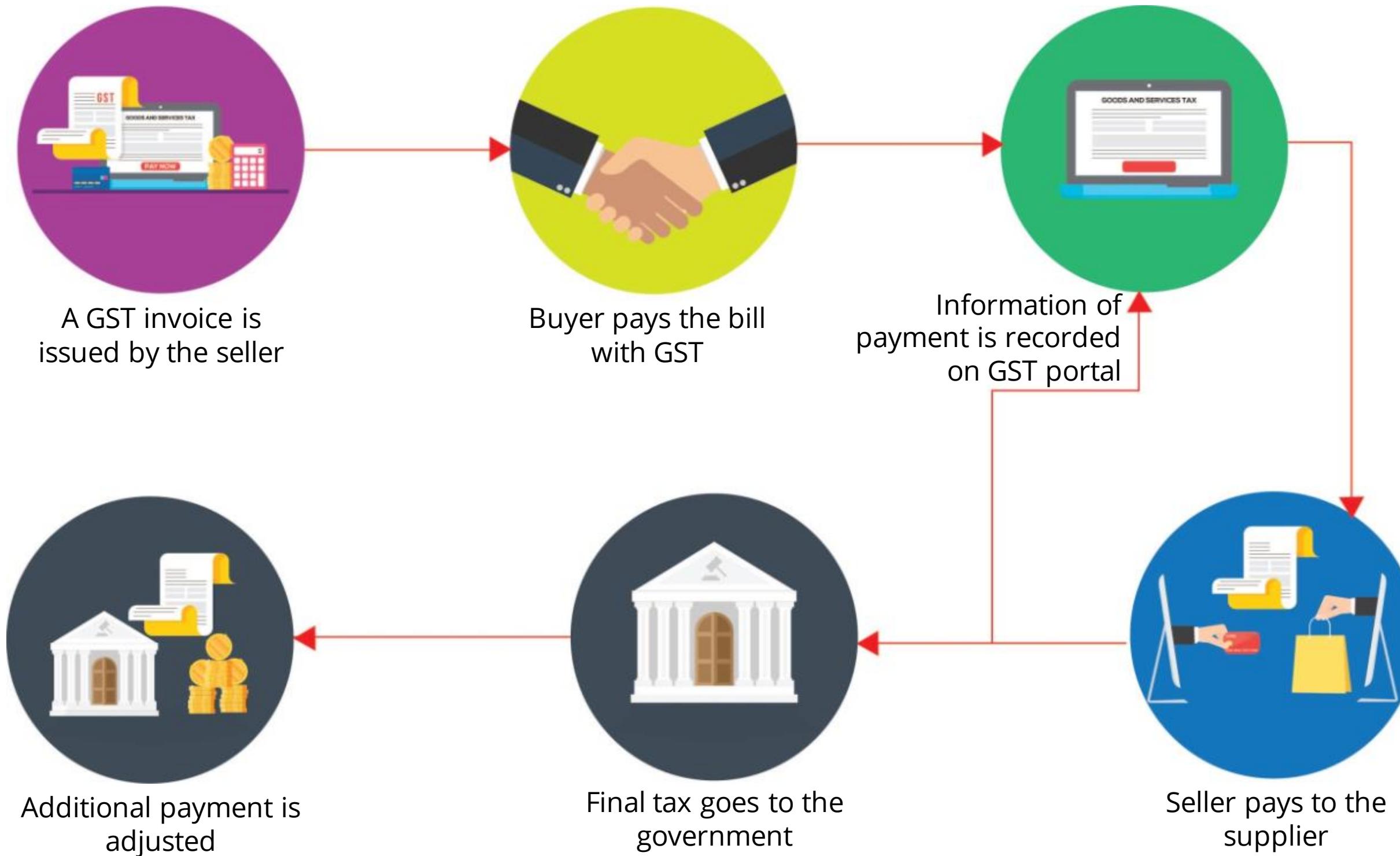
## Singapore

Singapore government has initiated project Ubin for clearing and settling payments and securities

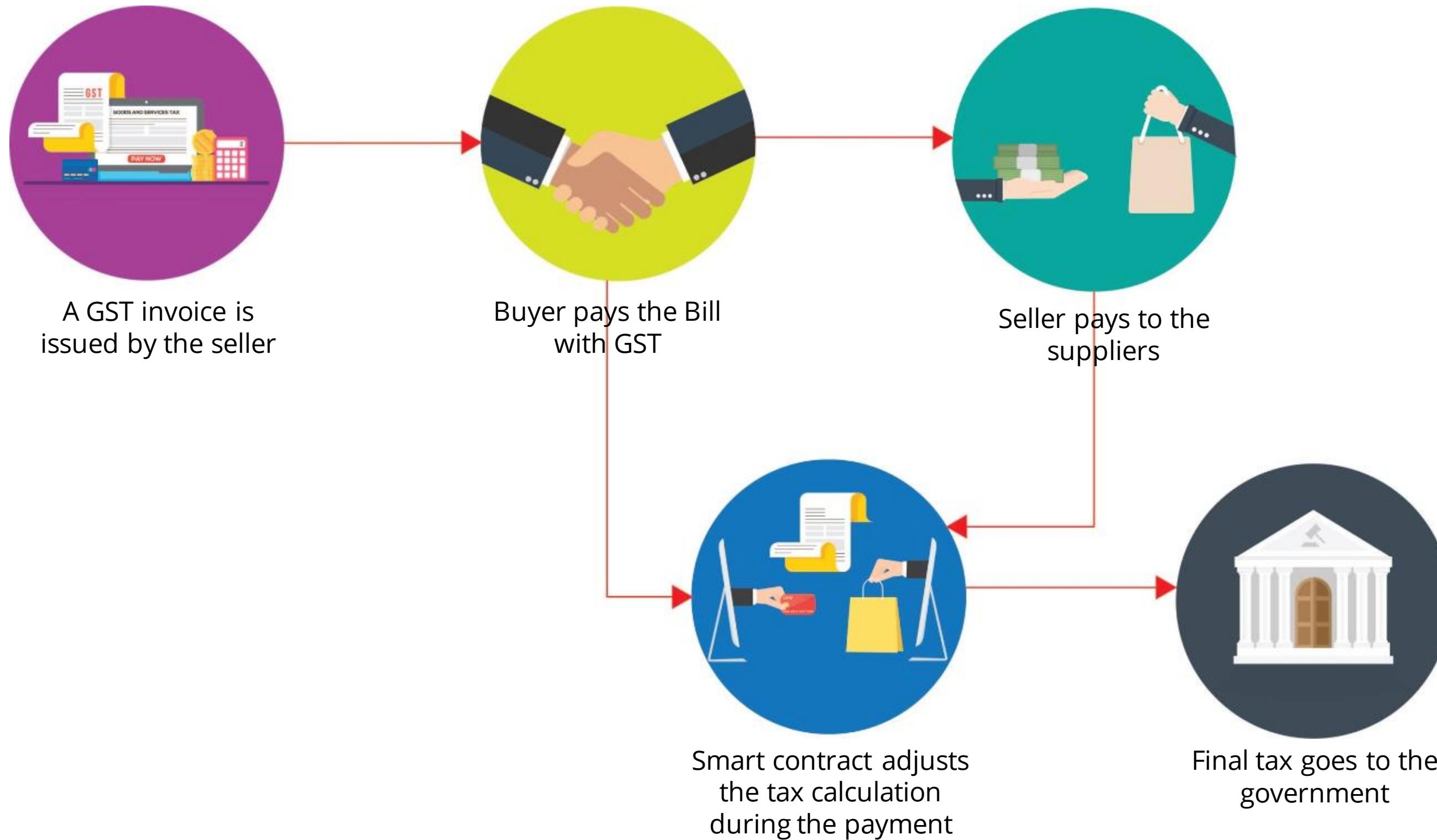
## USA

General Services Administration of USA is interested to use Blockchain for financial management, procurement, supply chain management, and many other industries.

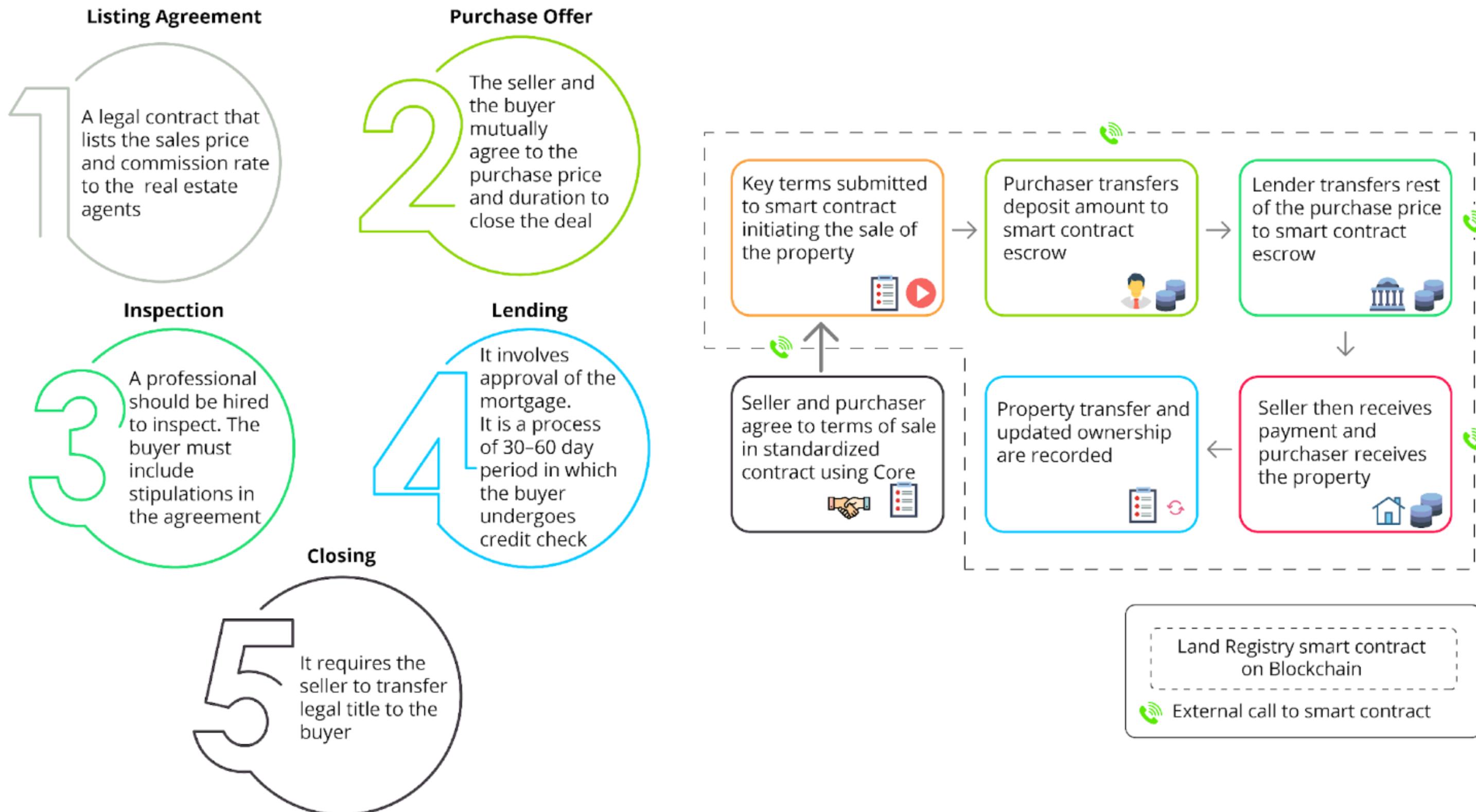
# Tax Processing: GST without Blockchain



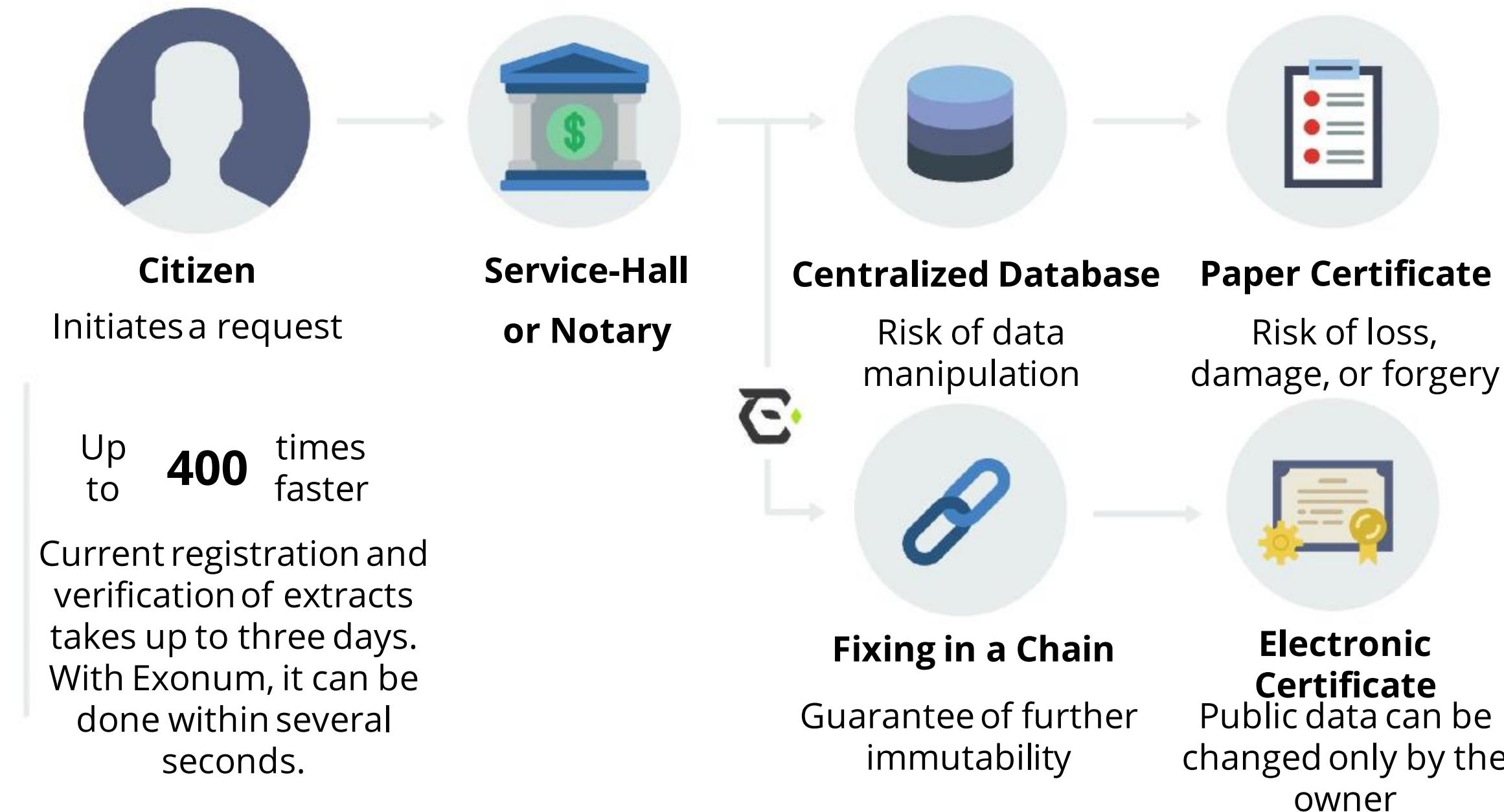
# Tax Processing : GST with Blockchain



# Blockchain in Real Estate: OpenLaw Core

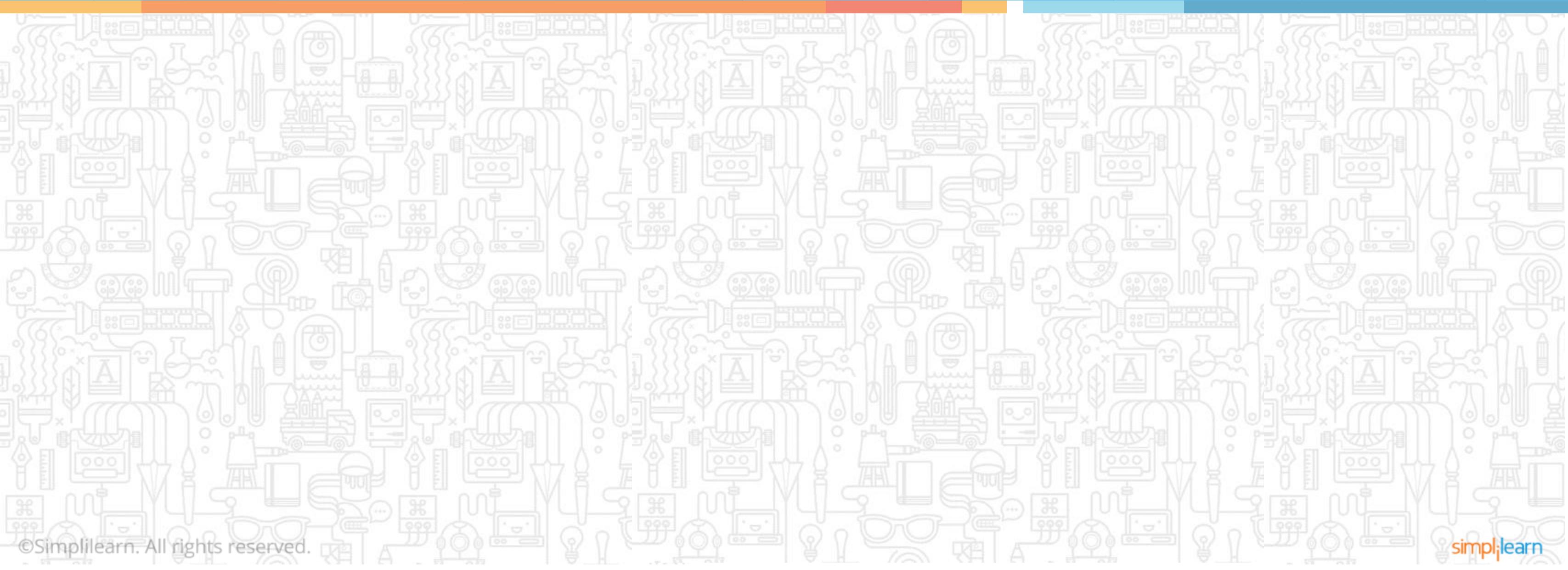


# Blockchain in Land Registry: Exonum



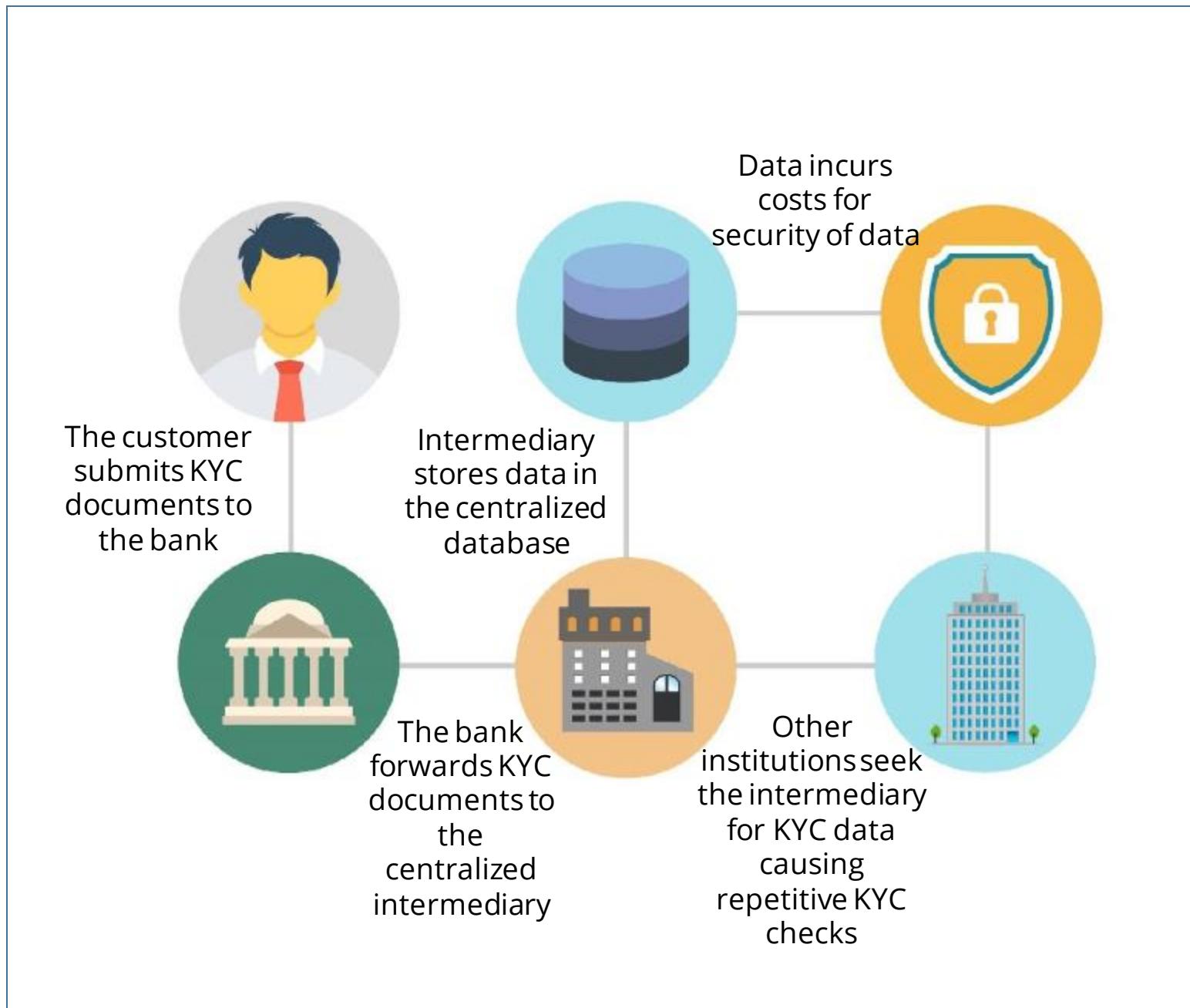
# Blockchain Prospects

## Finance Use Cases

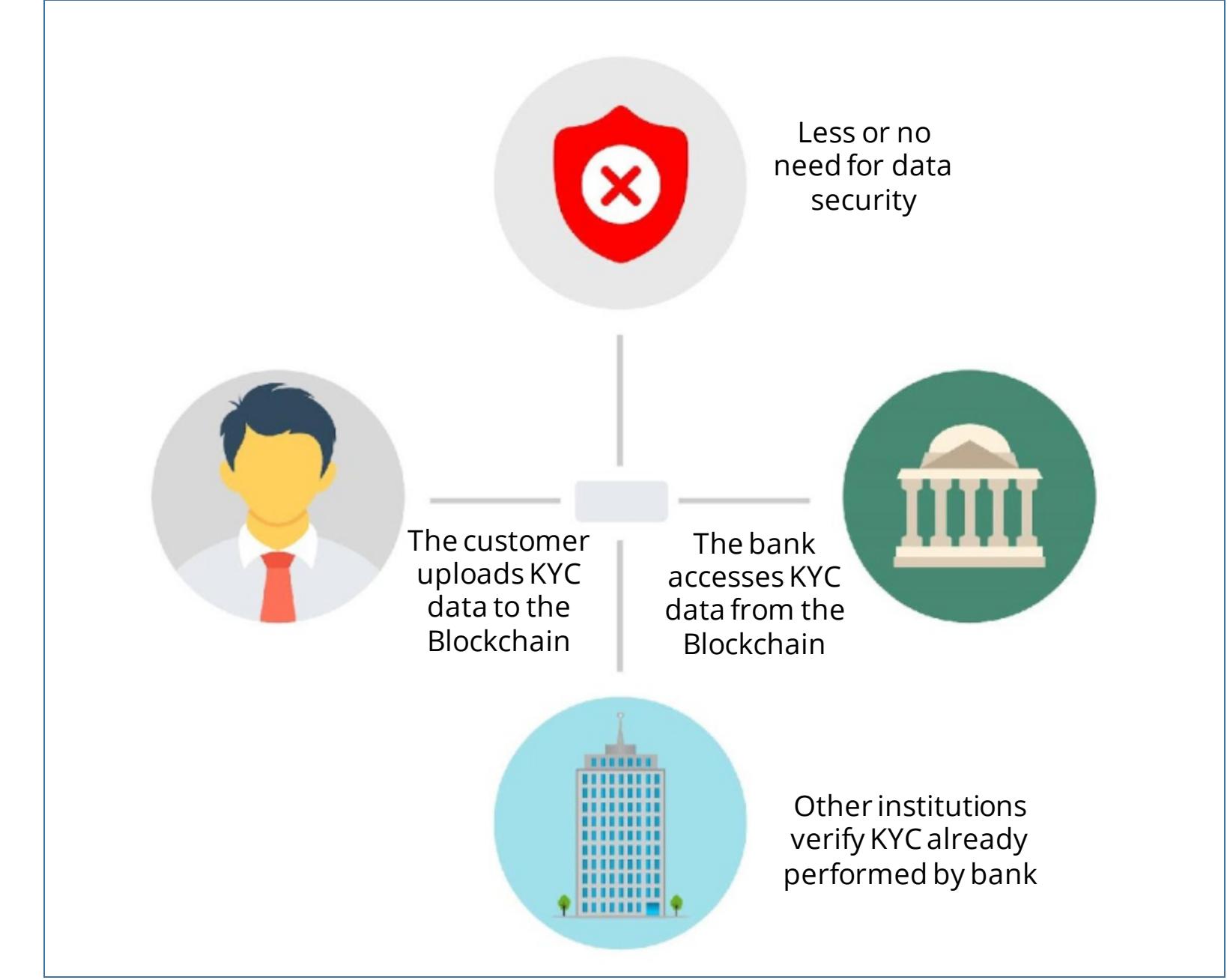


# Blockchain in KYC

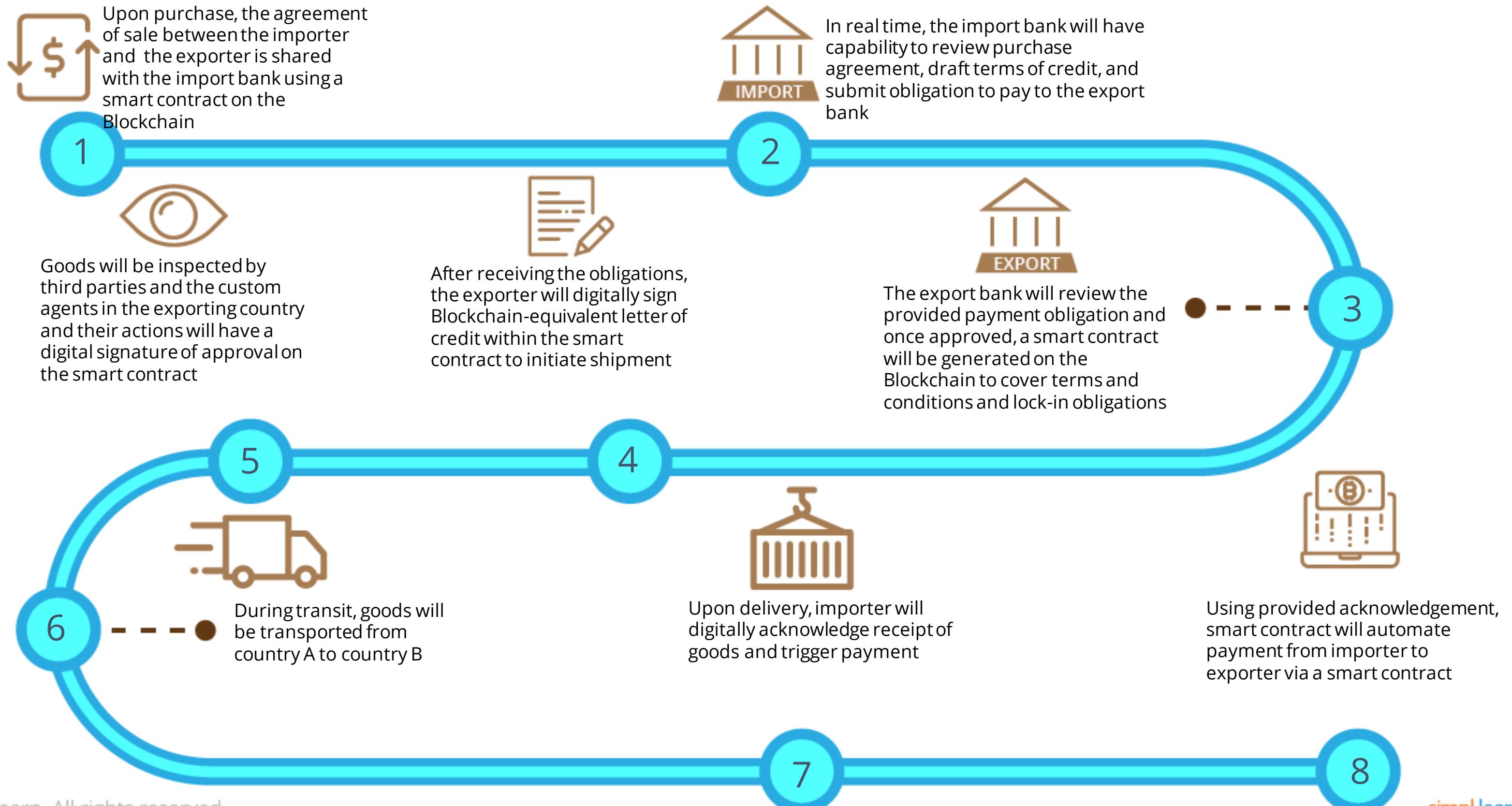
Centralized database requires heavy cost and effort:



Blockchain saves cost, time, and effort:

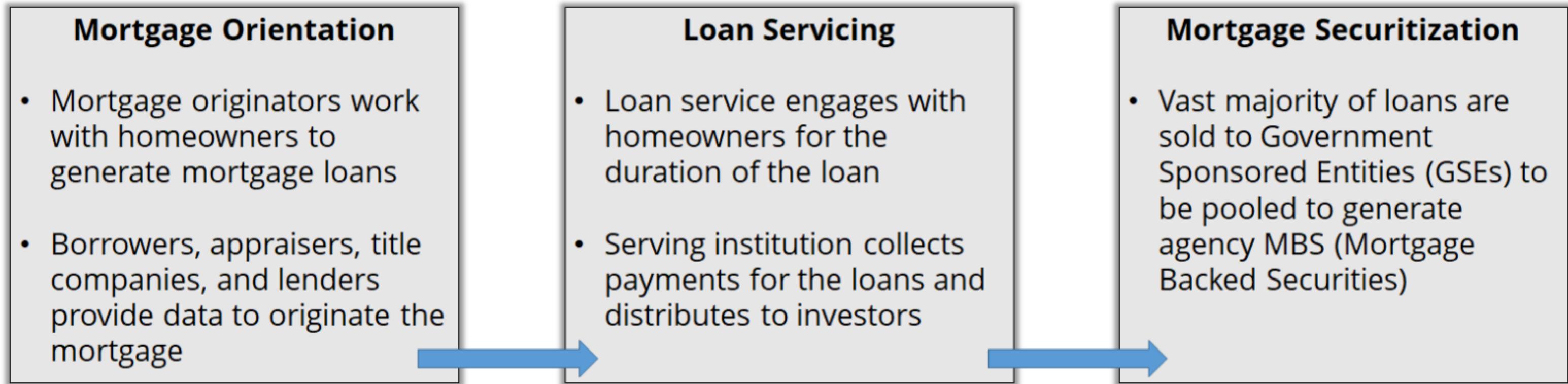


# Blockchain in Trade Finance



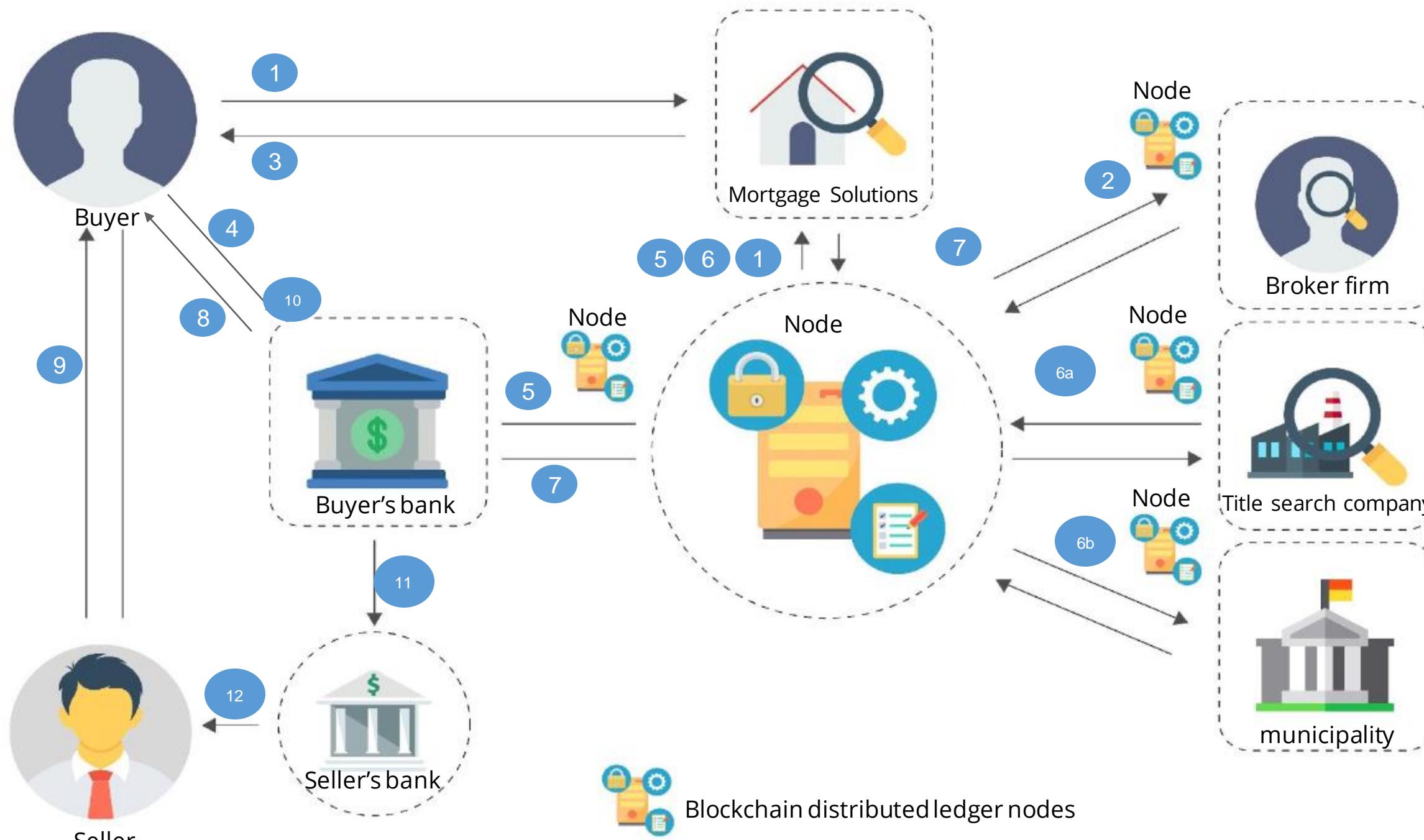
# Blockchain in the Mortgage Industry

## Current Mortgage Process:



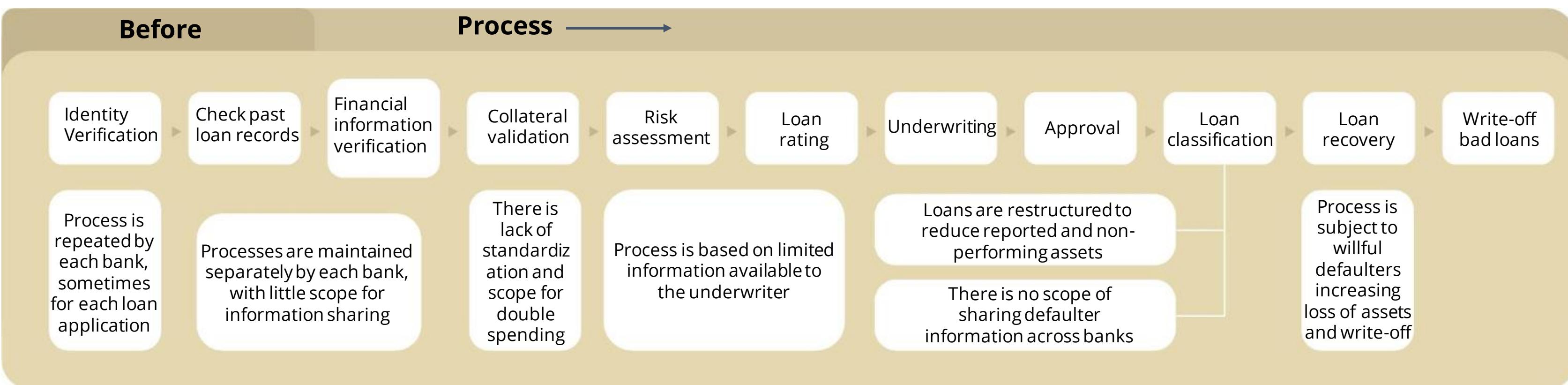
# Blockchain in the Mortgage Industry (Cont.)

## Mortgage process using Blockchain



# Blockchain in Loan Management

Loan management before implementing Blockchain



# Blockchain in Loan Management (Cont.)

## Loan management after implementing Blockchain

After

Blockchain application ————— Process Improvement ————— Process —————>

Smart Identity

Information sharing

Smart collaterals

Information sharing

Smart contracts

Smart property

Foster identity verification is done using smart identity

Shared information provides access to richer information for well-informed decision making

Share the collateral valuation information, and eliminate the scope of double spend

Shared information and consensus assure efficient governance and transparency in underwriting

Cryptographically signed immutable loans records. Restructuring entails approval from distributed network

Foster identification of risky customers is possible with high debt repayment capacity

Smart property used for effective collateral management across banks and faster transfer of collateral ownership in case of lost recovery

Identity verification

Check past loan records

Financial information verification

Collateral valuation

Risk assessment

Loan rating

Underwriting

Approval

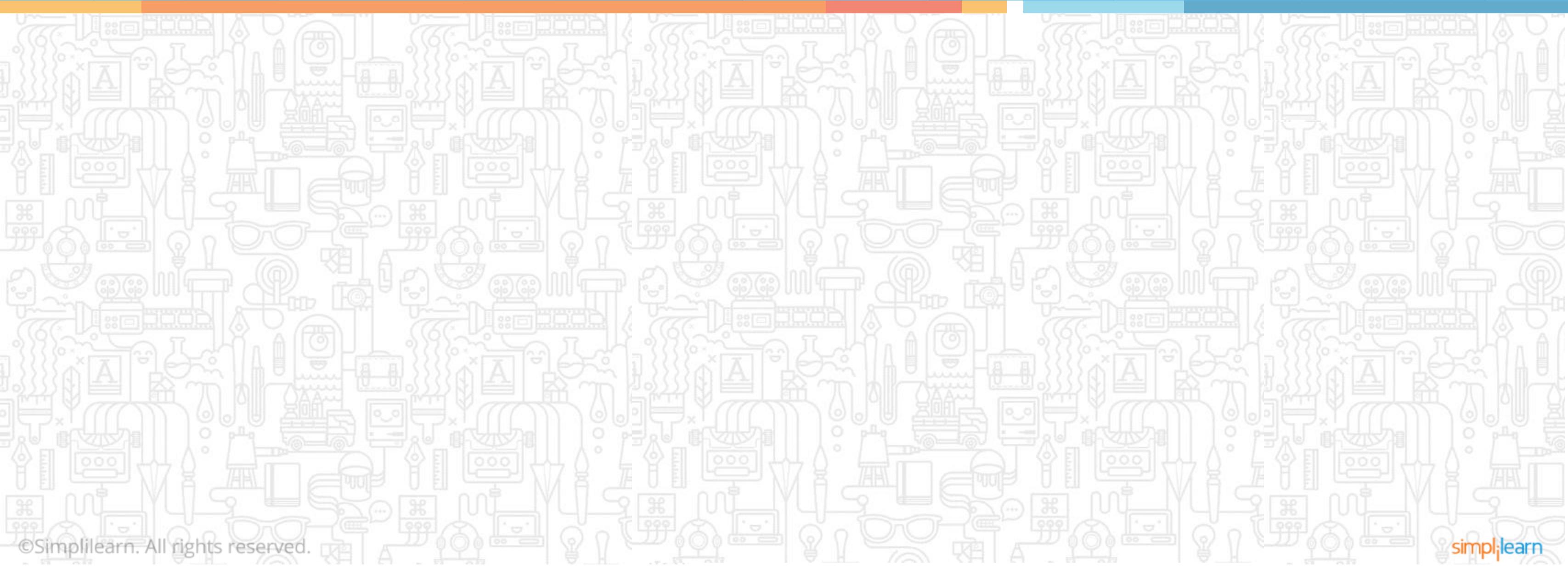
Loan classification

Loan recovery

Write-off bad loans

# Blockchain Prospects

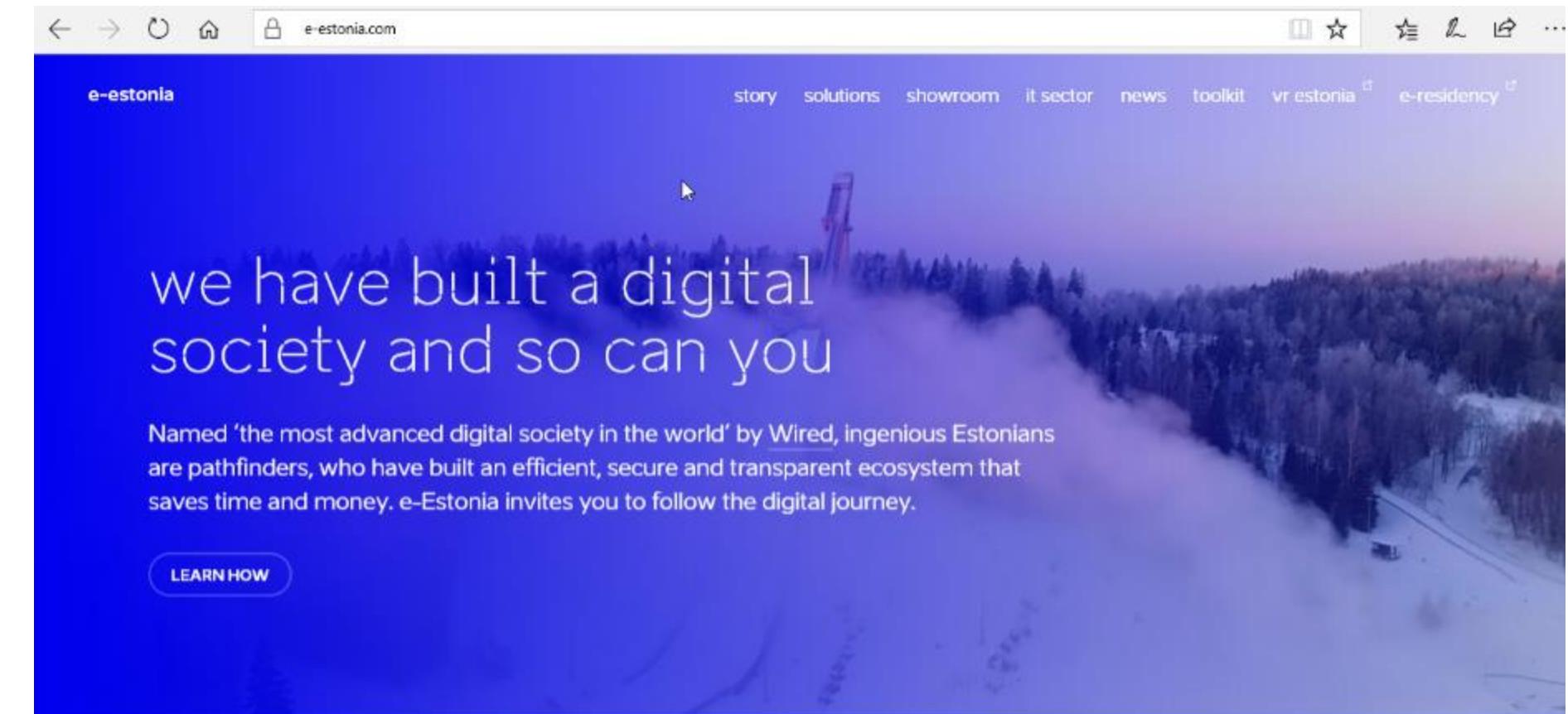
## Other Worldwide Use Cases



# Blockchain in e-Estonia

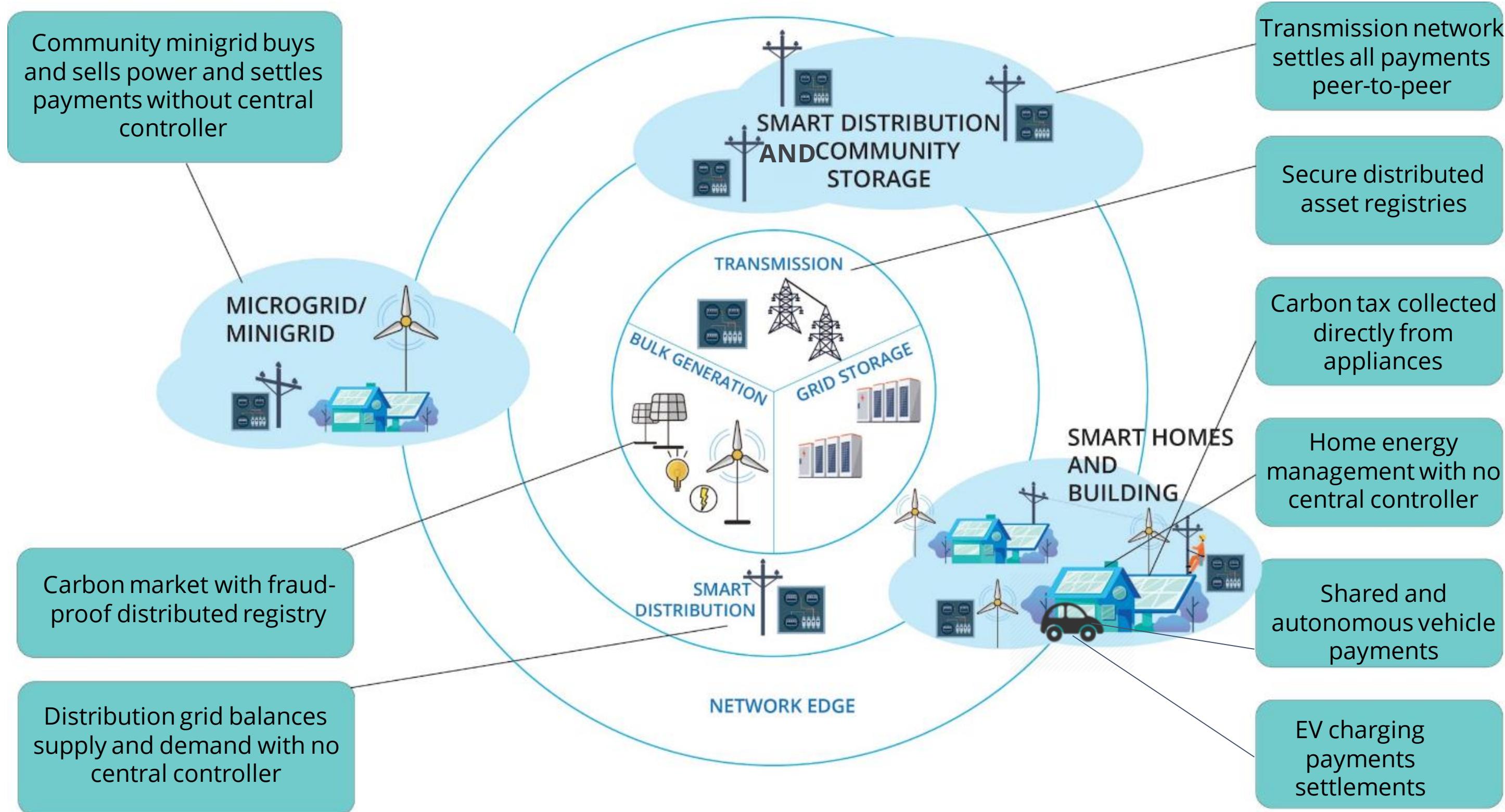
Blockchain in e-Estonia is used for the following:

- Securing health records by writing every update
- Maintaining all transactions of stock exchange
- Storing birth certificates, business contracts, and marriage registrations



e-estonia.com

# Blockchain in Energy Markets: Gridchain



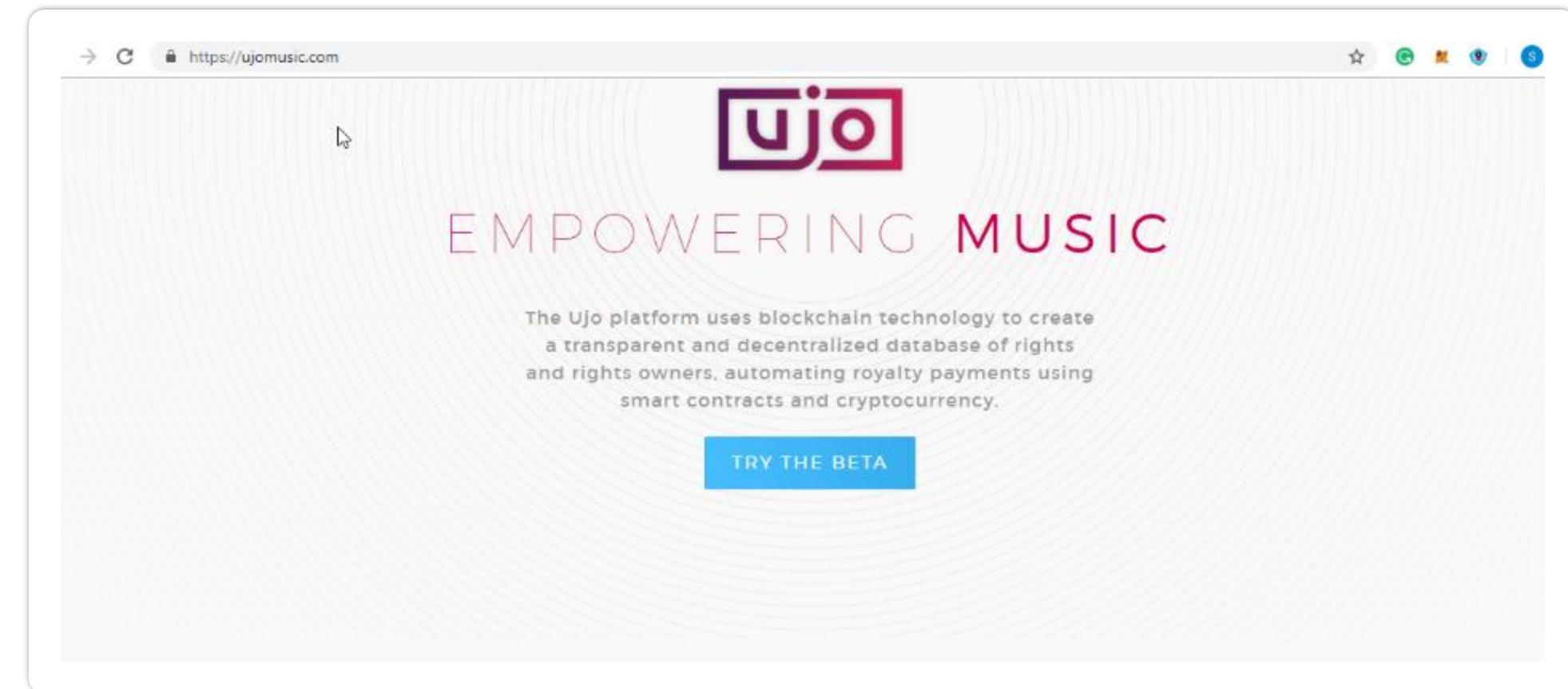
# Blockchain in Media: Ujo Music

Problems:

- Comprehensive database of music copyright doesn't exist
- Royalty amount takes longer than usual to reach the music makers

Solution:

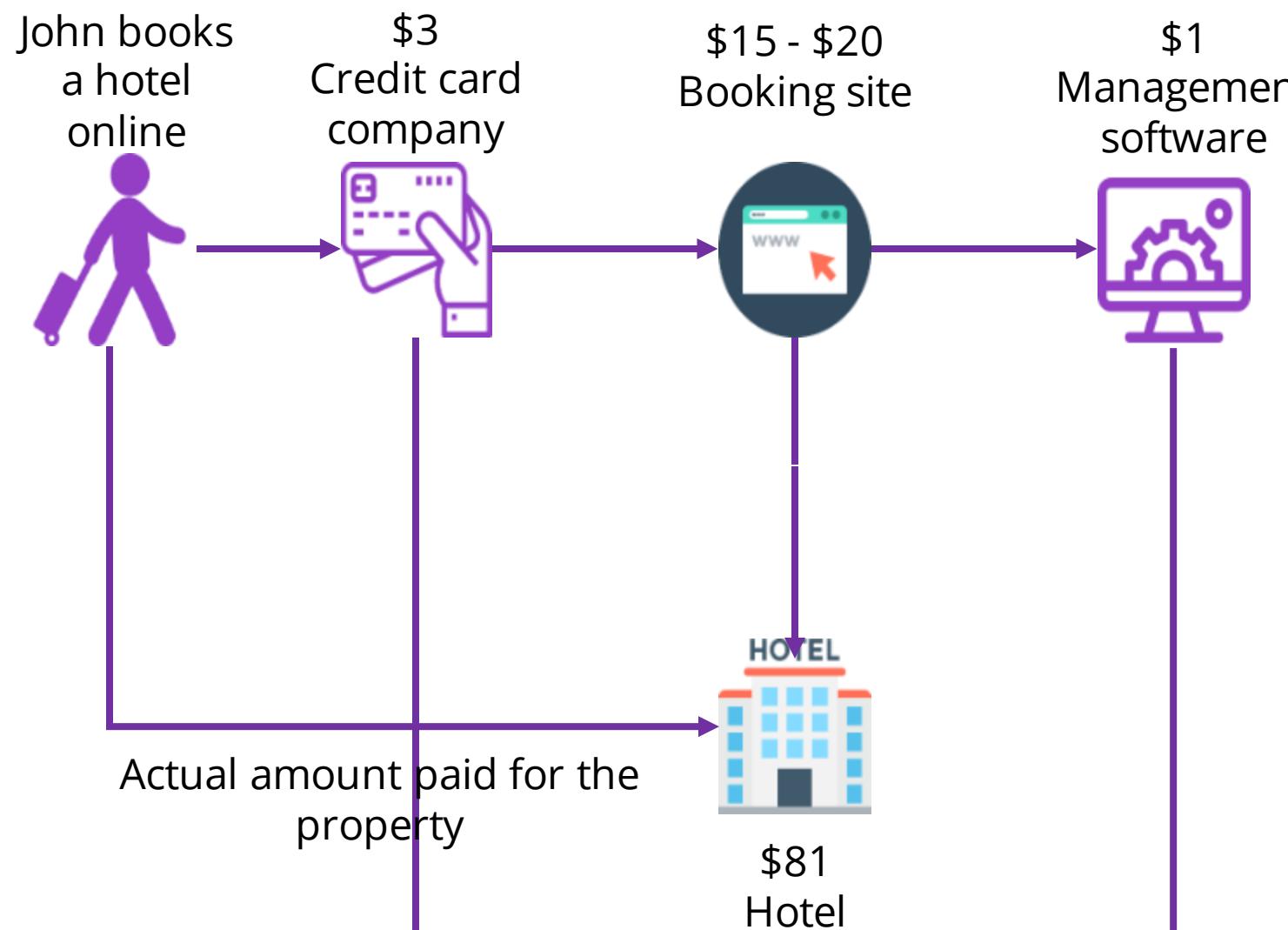
- Blockchain can make the royalties calculation fast, transparent, and secure
- Blockchain can provide a tamper-proof and easily auditable database



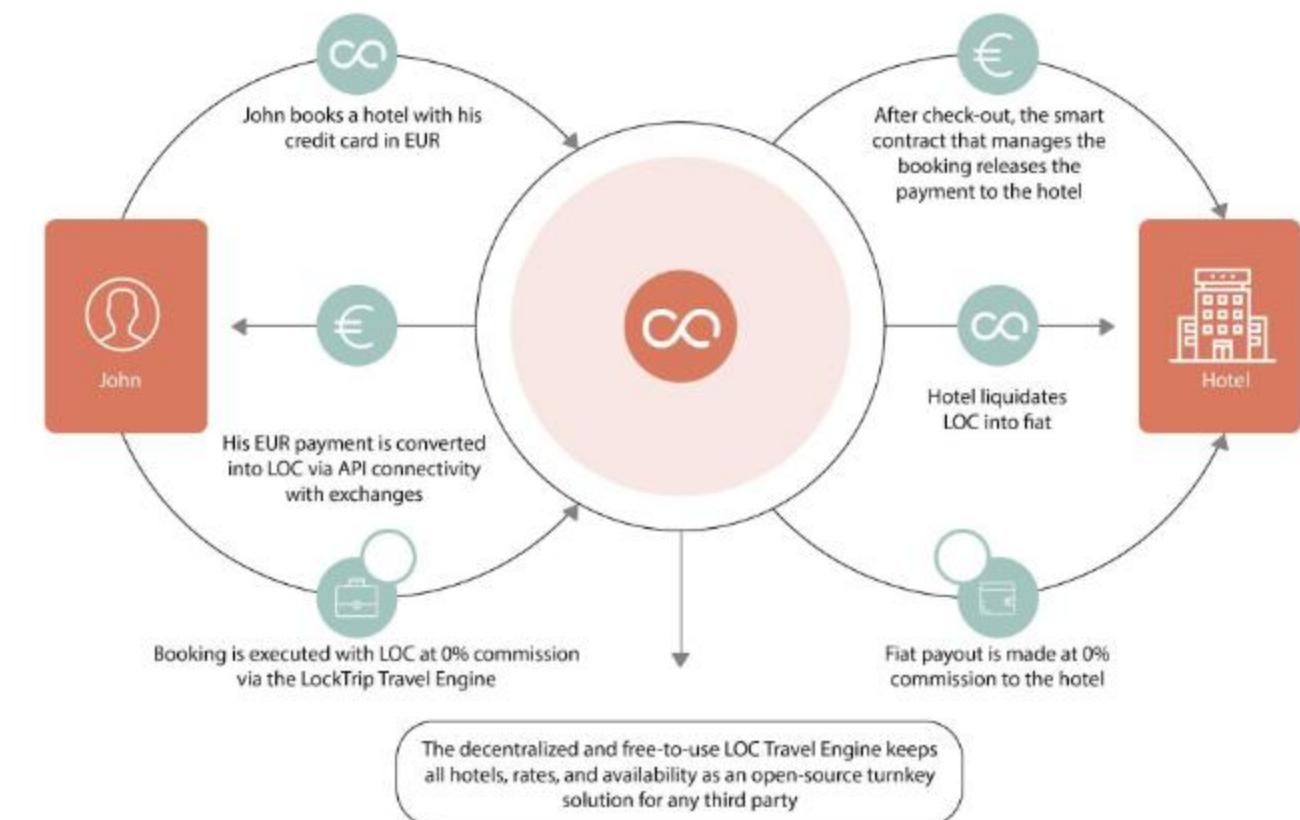
Ethereum based Decentralized music platform: ujomusic.com

# Blockchain in Travel: LockTrip

## Travel industry before implementing Blockchain

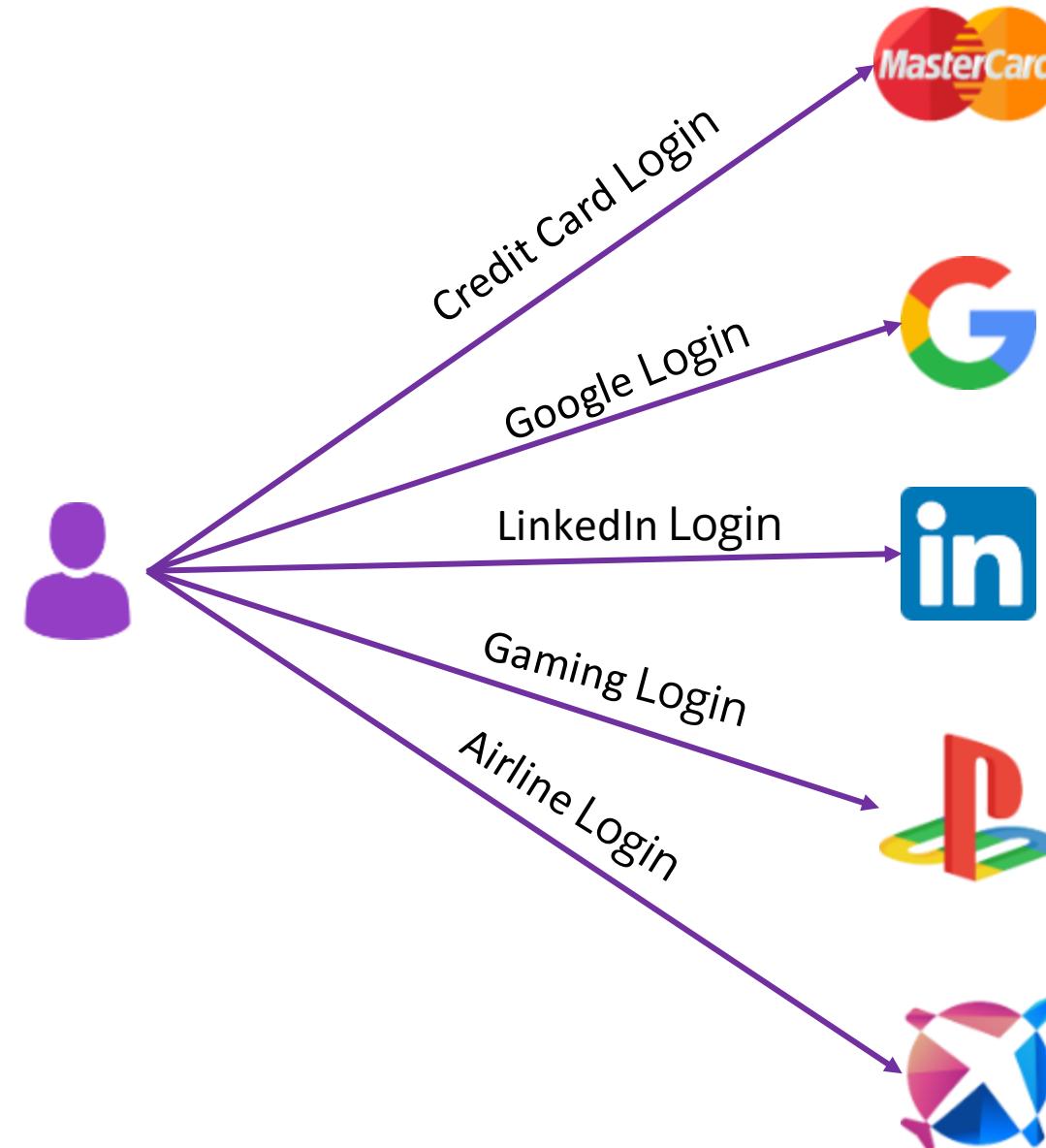


## Travel industry after implementing Blockchain

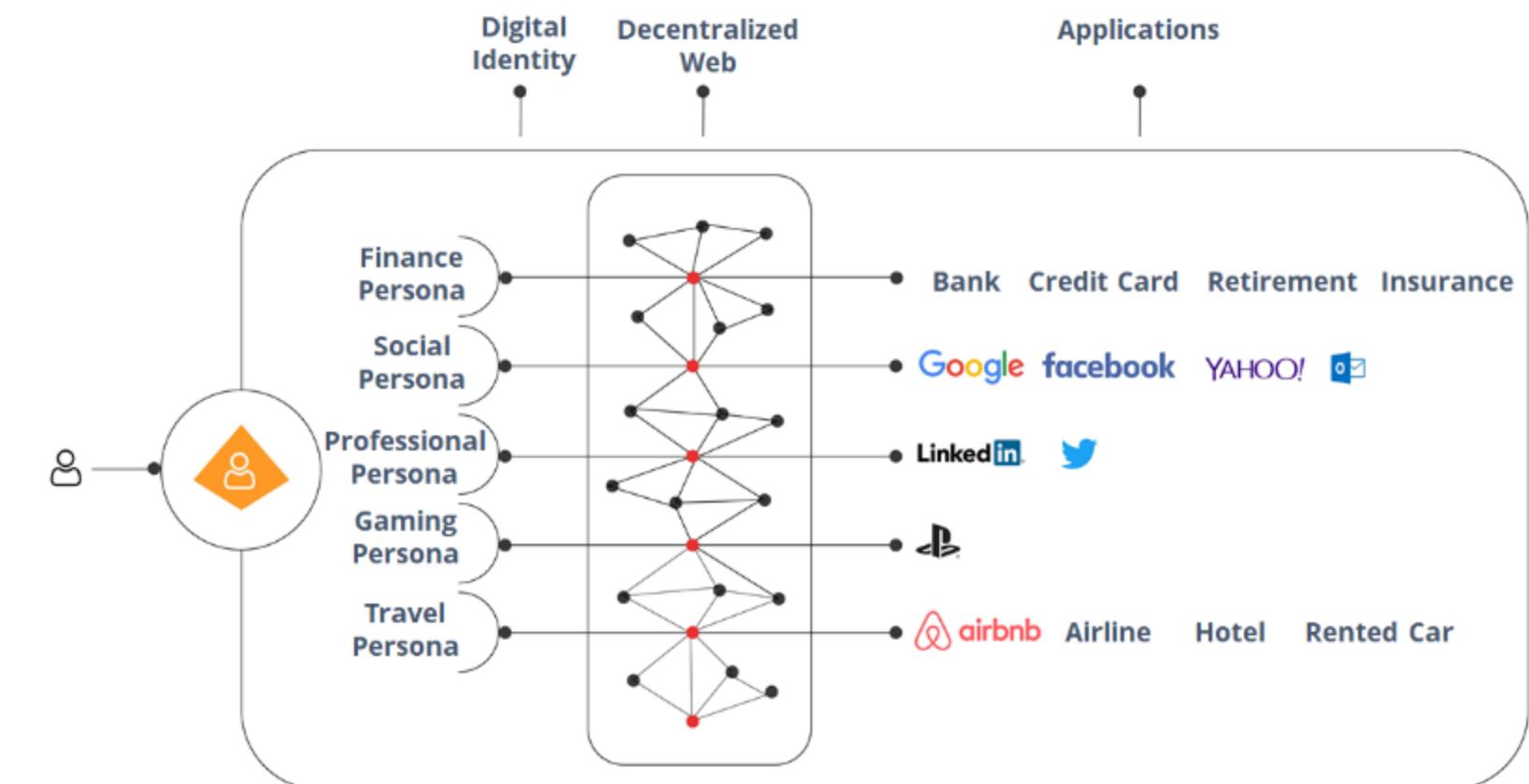


# Blockchain in Identification: uPort

Identification process before implementing Blockchain



Identification process after implementing Blockchain



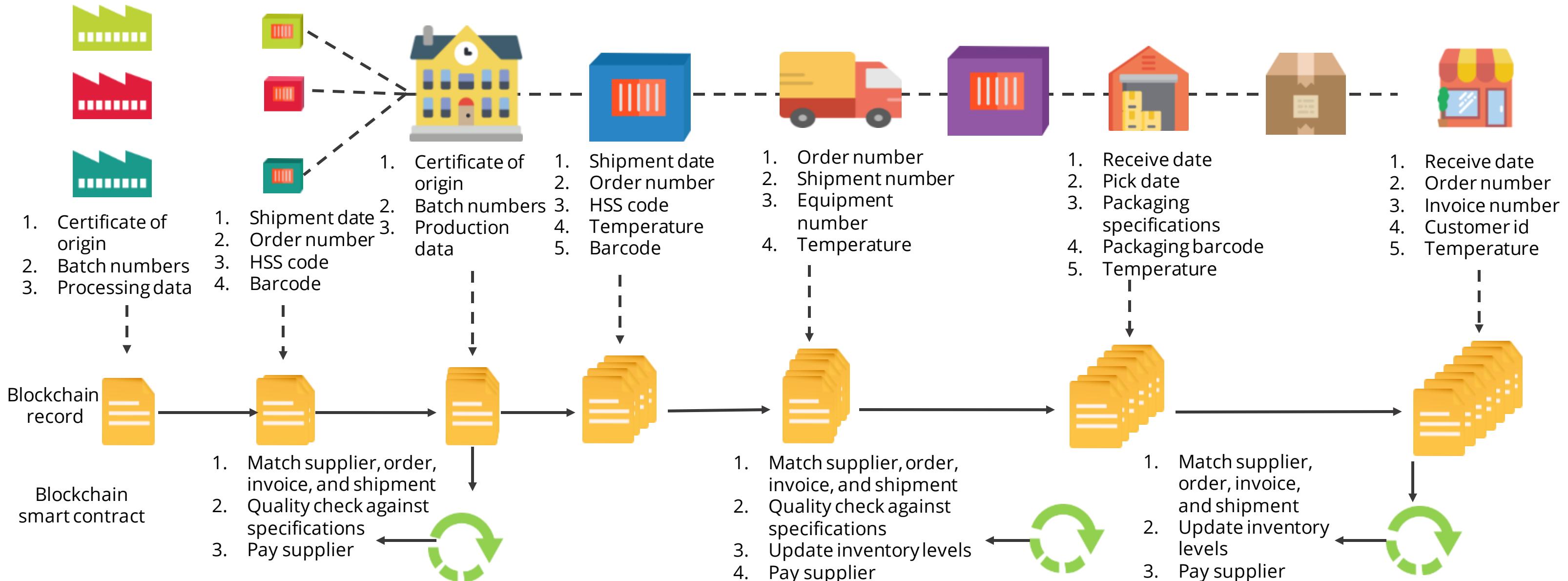
# Blockchain in Supply Chain

## Order Placement: Traditional approach

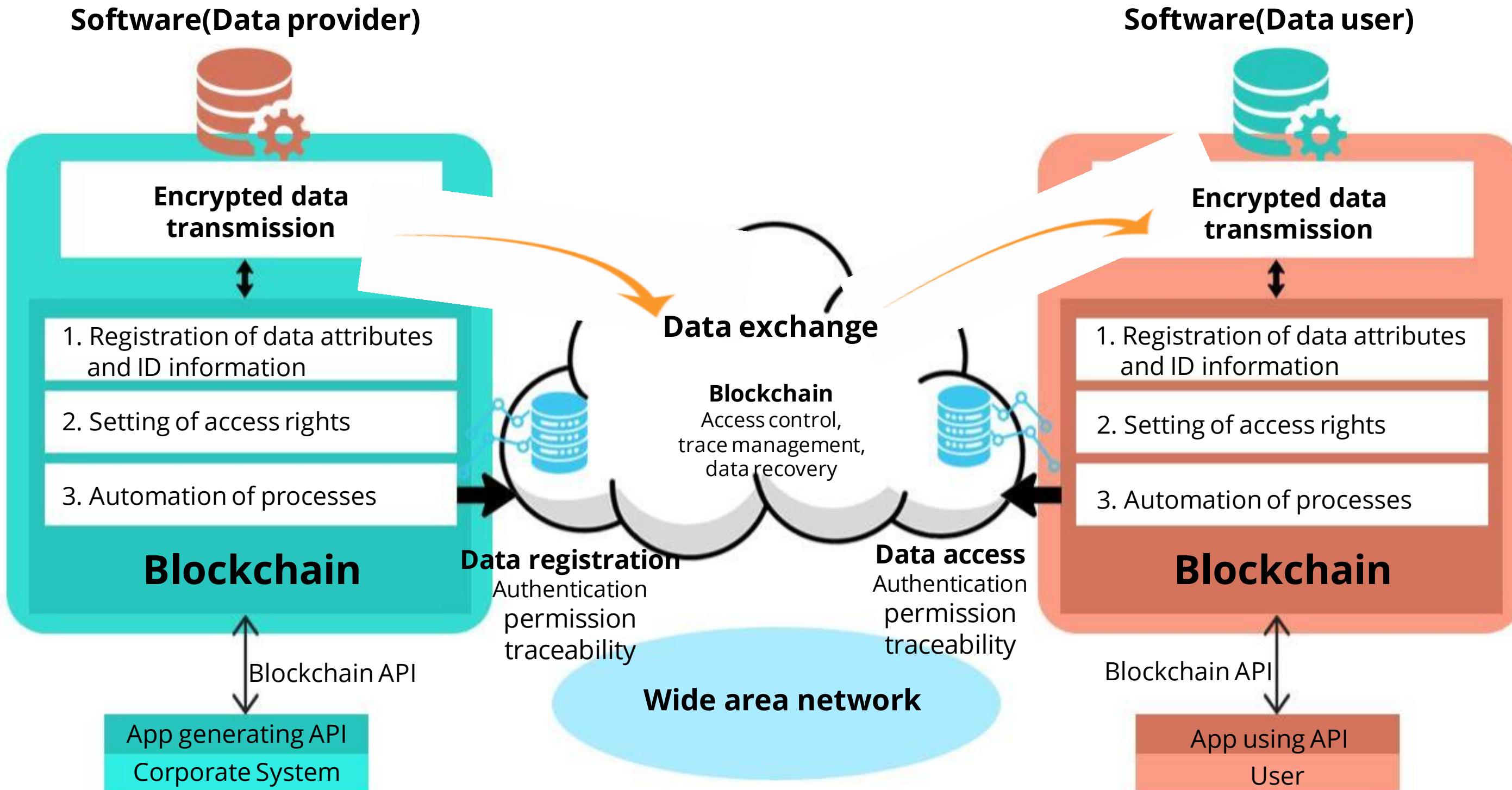


# Blockchain in Supply Chain

## Order placement using Blockchain smart contracts



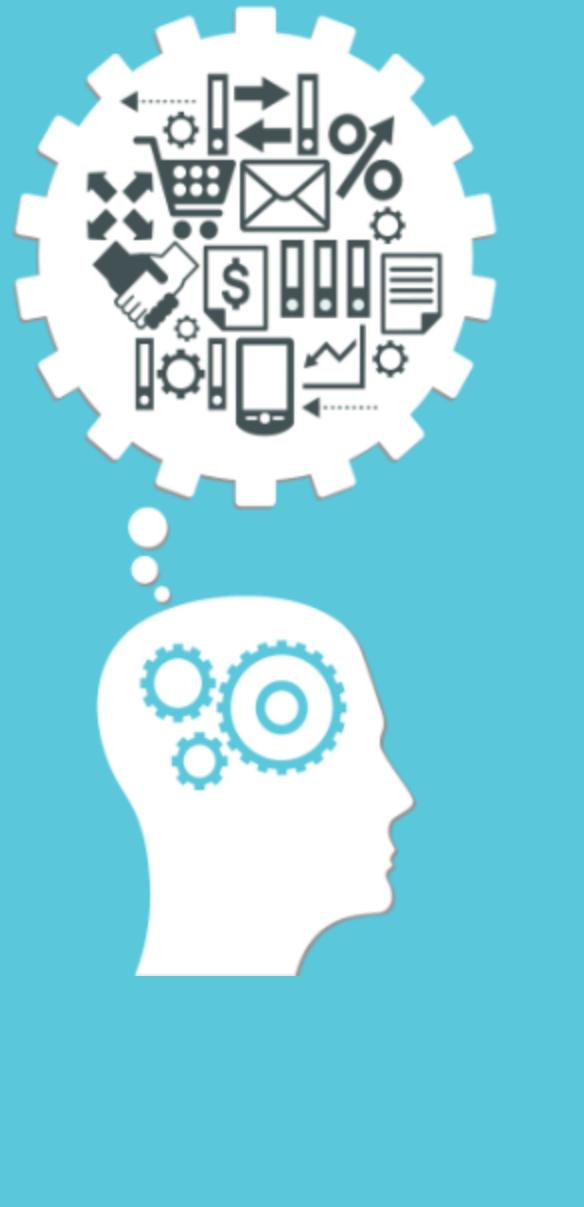
# Blockchain for Network Security: Korea Telecom



# Key Takeaways

You are now able to:

- Identify the use cases of Blockchain in healthcare industry
- Identify the use cases of Blockchain in government organization
- Identify the use cases of Blockchain in finance industry
- Identify the use cases of Blockchain in other industries worldwide





## Knowledge Check



Knowledge  
Check

1

**Which of the following advantages are provided by the blockchain based solution for trade logistics?**

- a. Tamper proof and digitally signed documents
- b. Process automation
- c. Real time visibility and analytics
- d. All of the above



## Which of the following advantages are provided by the blockchain based solution for trade logistics?

- a. Tamper proof and digitally signed documents
- b. Process automation
- c. Real time visibility and analytics
- d. All of the above



The correct answer is **d**

**Blockchain provides tamper proof, digitally signed documents, process automation, real time visibility, and analytics for trade logistics.**

## **The blockchain can address the errors in the current supply chain by:**

- a. Automatically tracking the information and verifying it using the smart contract
- b. Only verifying the information using the smart contract
- c. Only tracking the information
- d. Making the data available to auditors to manually verify and correct errors



## The blockchain can address the errors in the current supply chain by:

- a. Automatically tracking the information and verifying it using the smart contract
- b. Only verifying the information using the smart contract
- c. Only tracking the information
- d. Making the data available to auditors to manually verify and correct errors



The correct answer is

**a**

**Blockchain can address the errors by automatically tracking the information and verifying it using the smart contract.**

Knowledge  
Check

3

**Ujo music is based on which of the following blockchain platforms?**

- a. Hyperledger Fabric
- b. Hyperledger Sawtooth Lake
- c. Ethereum
- d. HydraChain



Knowledge  
Check

3

**Ujo music is based on which of the following blockchain platforms?**

- a. Hyperledger Fabric
- b. Hyperledger Sawtooth Lake
- c. Ethereum
- d. HydraChain



The correct answer is

**C**

**Ethereum is used by Ujo music to build flexible and modular licensing systems.**

## Which of the following is not a valid example of identity fraud?

- a. Someone opens a social account using your social history
- b. Someone accesses online shopping website using your personal information
- c. Academic institute accesses its students' current academic record
- d. A banker uses your banking details for his own purpose



## Which of the following is not a valid example of identity fraud?

- a. Someone opens a social account using your social history
- b. Someone accesses online shopping website using your personal information
- c. Academic institute accesses its students' current academic record
- d. A banker uses your banking details for his own purpose



The correct answer is

C

**Academic institute accessing their student's current academic record does not fall under the fraud identity example.**

**Which of the following features in Blockchain can allow proper management of patient consent?**

- a. Digitally signed messages
- b. Automation
- c. Secure transactions
- d. Tamper-proof data storage



## Which of the following features in Blockchain can allow proper management of patient consent?

- a. Digitally signed messages
- b. Automation
- c. Secure transactions
- d. Tamper-proof data storage



The correct answer is

**a**

**Digitally signed messages provide an approval from the patients to access and manage their personal data.**



**Thank You**