Unit 4 Notes

Smart Contract

- Self executing code
- Third party to witness the transaction
- Removing the usage of the third party, so that the system works in a trusted way
- System takes care of the validity
- Similar to legal agreement

Definition

A smart contract is similar to a **contract** in the physical world, but it's **digital** and is represented by a tiny **computer program stored inside a blockchain.**

A smart contract is a piece of **software** that **stores rules for negotiating** the **terms of an agreement**, automatically **verifies fulfillment**, and then **executes the agreed terms**.

What's the main idea of a smart contract?

- Smart contract are created by developers and enforced with Boolean logic, mathematics and encryption.
- Smart contract **removes reliance on a third party** when establishing business relations, the parties making an agreement can **transact directly with each other.**
- According to **Nick Szabo**, an American computer scientist who devised a virtual currency called "Bit Gold" in 1998, **Smart contracts are computerized transaction protocols that execute contract conditions.**
- Using it makes the transactions traceable, transparent, and irreversible.

Example

Crowdfunding platform- Kickstarter

Why Trust a Smart Contract?

- Immutable
- Distributed

Benefits of Smart Contracts

- **Accuracy, Speed, Efficiency**
  - The contract is immediately executed when a condition is met.

    - No paperwork, no time was spent correcting errors that can occur when filling out documentation by hand.
- **Trust and Transparency**
    - No third party tampering
    - Encrypted transaction logs are exchanged among participants.
- **Security**
    - Blockchain transaction records are encrypted, they are extremely difficult to hack.
    - Entry on a distributed ledger is linked to the entries before and after
- **Savings**
    - No fee to register in smart contract, or third party payment

How a Smart Contract Works

- A Logically Behaved Algorithm
- The Logic of the One-Size-Fits-All Smart Contract

Example:
Following terms and events can be set out in a smart contract between Alice and Bob:

- Independent storage is created, where both Alice and Bob can put value but can't easily take out.
- **Bob puts money** for rent **in storage**.
- **Alice puts the address** and the code to her apartment **in storage.**
- **Alice gets payment confirmation** and **Bob receives the address** and apartment code.
- **If Bob comes to LA** and the **address and code** provided by Alice are right, **Alice gets the payment.**
- If it appears that the **address or code** supplied by Alice are **wrong, Bob gets his money back.**
- If Bob **doesn't come to LA**, **Alice gets her liquidated damages payment** and Bob gets the rest of what he paid.
- At the end of the agreement, the smart contract is considered fulfilled and remains stored in the blockchain network.

How a Smart Contract Works

- **Step 1:** Business teams **collaborate with developers** to **define** their **criteria** for the **smart contract's desired behavior** in response to certain events or circumstances.

- **Step 2:** Conditions such as **payment authorization, shipment receipt, or a utility meter reading threshold** are examples of **simple events.**

- **Step 3:** More **complex operations**, such as **determining the value of a derivative financial instrument, or automatically releasing an insurance payment**, might be encoded using more sophisticated logic.

- **Step 4:** The developers then use a **smart contract writing platform** to **create and test the logic**. After the application is written, it is sent to a separate team for security testing.

- **Step 5:** An **internal expert or a company** that **specializes in vetting** smart contract **security** could be used.

- **Step 6:** The contract is then **deployed on an existing blockchain** or other distributed ledger infrastructure once it has been authorized.

- **Step 7:** The smart contract is **configured to listen for event updates from an "oracle,"** which is effectively a cryptographically secure streaming data source, once it has been deployed.

- **Step 8:** Once it obtains the **necessary combination of events** from one or more oracles, the **smart contract executes.**

Use cases of Smart Contracts

Technical Use-Cases

- Self-verifying
- Self-executing
- Tamper resistance

Legal Use-Cases

- It can map legal obligations into an automated process.
- If implemented correctly, they can provide a greater degree of contractual security

Economic Use-Cases

- Higher transparency
- Fewer intermediaries
- Lower transaction costs

An Example

Digital Vending machine:
To get a snack from a vending machine:

money + snack selection = snack dispensed
This logic is programmed into the vending machine.
A smart contract, like a vending machine, has logic programmed into it.

\

```
pragma solidity 0.8.7;

contract VendingMachine {
   // Declare state variables of the contract
   address public owner;
   mapping (address => uint) public
cupcakeBalances;

   // When 'VendingMachine' contract is deployed:
   // 1. set the deploying address as the owner of
the contract
   // 2. set the deployed smart contract's cupcake
balance to 100
   constructor() {
      owner = msg.sender;
      cupcakeBalances[address(this)] = 100;
   }
```

```
// Allow the owner to increase the smart contract's cupcake
balance
   function refill(uint amount) public {
      require(msg.sender == owner, "Only the owner can
refill.");
      cupcakeBalances[address(this)] += amount;
   }

   // Allow anyone to purchase cupcakes
   function purchase(uint amount) public payable {
      require(msg.value >= amount * 1 ether, "You must pay
at least 1 ETH per cupcake");
      require(cupcakeBalances[address(this)] >= amount,
"Not enough cupcakes in stock to complete this purchase");
      cupcakeBalances[address(this)] -= amount;
      cupcakeBalances[msg.sender] += amount;
   }}
```

Usage of smart contracts

Permissionless

- **Anyone can write a smart contract** and deploy it to the network.
- Have **enough ETH to deploy** your contract.
- Deploying a smart contract is technically a **transaction**

- Pay your Gas in the same way that you need to pay gas for a simple ETH transfer.
- Ethereum has developer-friendly languages for writing smart contracts:
  - Solidity
  - Vyper
- However, they **must be compiled before** they can be deployed so that Ethereum's virtual machine can interpret and store the contract.

Composability

- Smart contracts are **public on Ethereum** and can be thought of as **open APIs.**
- You **can call other smart contracts** in your own smart contract to greatly extend what's possible.
- Contracts can even deploy other contracts.

Limitation of Smart Contracts

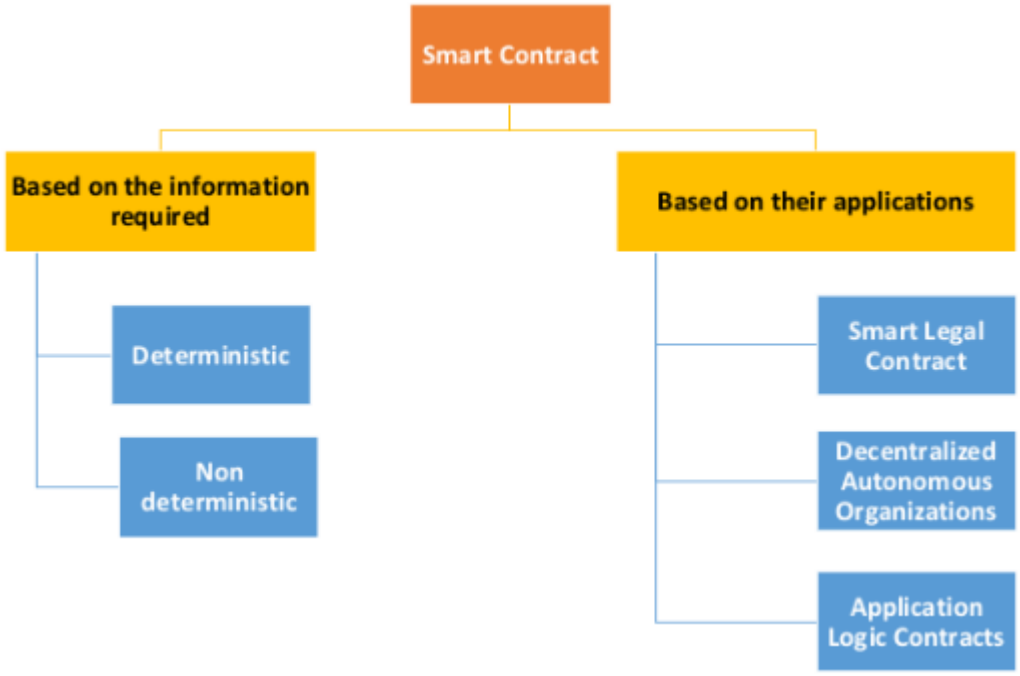
- Because smart contracts **can't send HTTP queries**, they can't acquire information about "real-world" events. This is by design.
- Using external data could jeopardize consensus, which is critical for **security and decentralization.**

SMART CONTRACT: CREATING AND DEPLOYING

Goals:

- **Removal of manual intervention** and oversight
  eg. from legal counsel
- **Reduction** in associated legal **costs**, fees and **process** (time) speed of contract creation and execution
- Automated **transfer of funds** via computer recognizable/definable events
- **Flexibility in contract**\

Types of smart contracts



Types of Smart Contracts

Based on the information required

- **Deterministic**- does not depend on outside information, on the network which they are live(blockchain info) has all the info
- **Non deterministic**- oracle is used for data outside (Eg. driver's license validated by government needed)

Based on their application

- Smart legal contract
- Decentralized autonomous organization(DAO) - communities that exist on a blockchain
- Application logic contracts

CREATE AND DEPLOY THE SMART CONTRACT

1. Understand the use-case
2. Create a basic architecture- flowchart how functions will interact with each other.
3. Development using any IDE with documentation- Truffle, remix
4. testing smart contracts on test-net or private blockchain(manual testing).
5. Analyze results of all transactions with actual use case or business logic of smart contract.
6. Unit testing & Integration Testing: Example : Truffle framework on ganache.
7. 3rd party Audit of smart contract.
8. Bug bounty programs- efficient to secure smart contracts.
   Note: bug bounty programs
   - get a reward if you find any bugs

- you have opened it to public, so someone can steal your code(Use copyrights that prevents that)

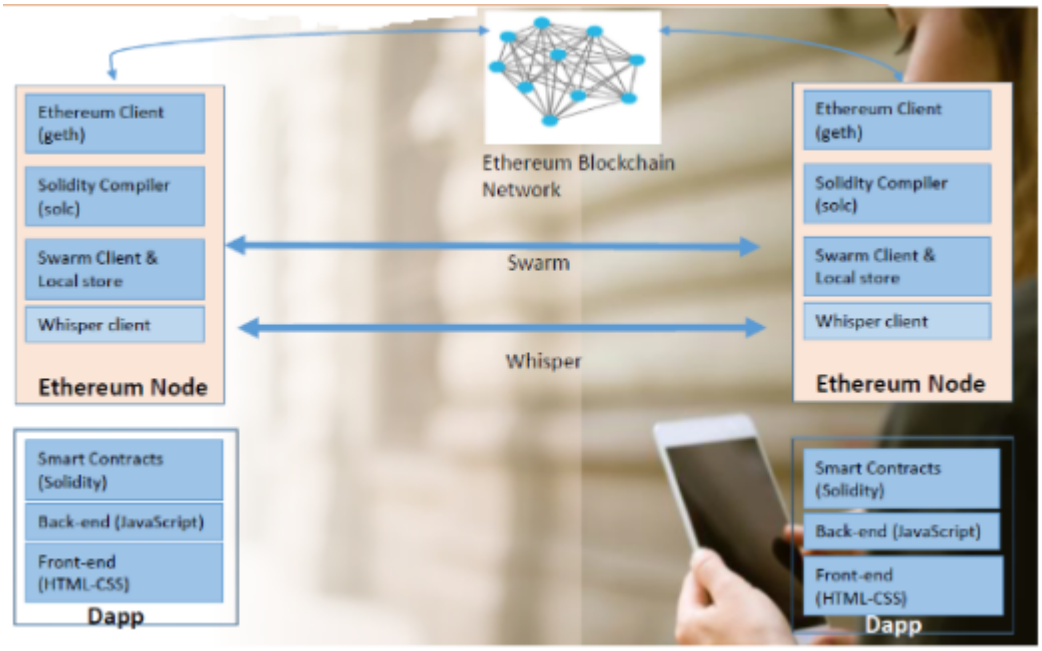
Blockchain Networks Using Smart Contracts

- **Bitcoin** is a smart contract→ running bitcoin itself is a smart contract(limited functionality)
- **Ethereum** uses smart contract for different applications

ADVANTAGES OF SMART CONTRACTS

- Direct dealing with customers
- Resistance to failure
- Immutability
- Fraud reduction
- Cost efficiency
- Records keeping

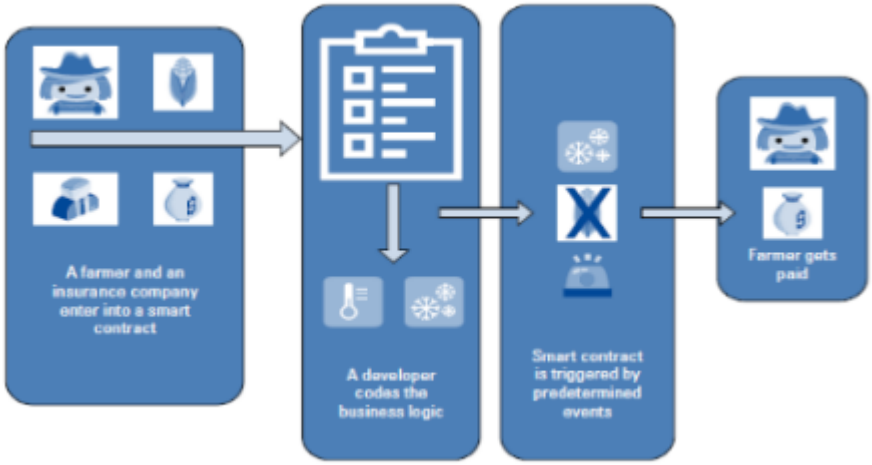
ETHEREUM BLOCKCHAIN COMPONENTS



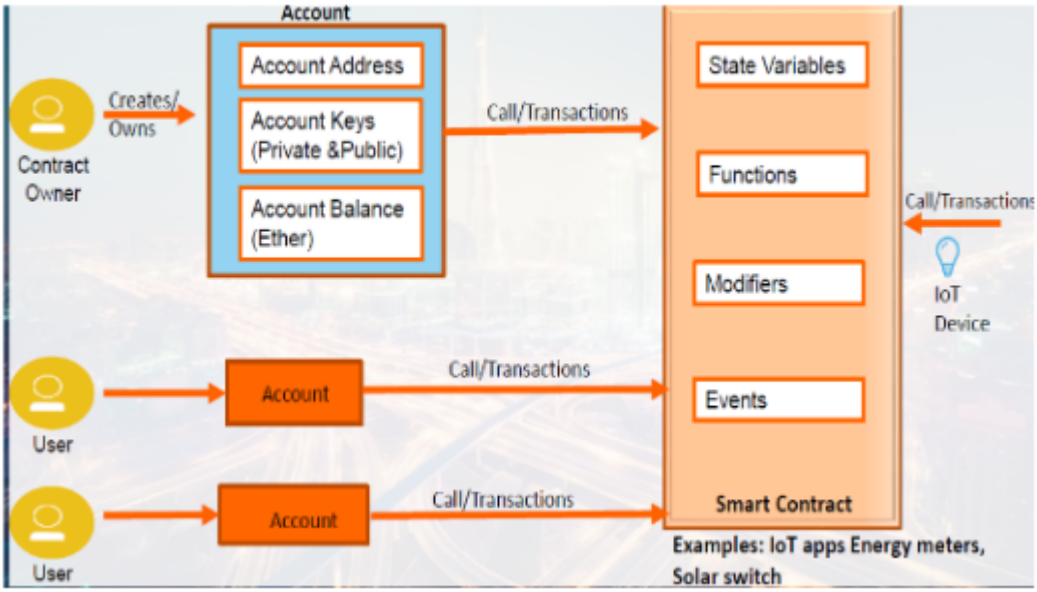
SMART CONTRACTS CAN AUTOMATE EXECUTION OF TRANSACTIONS

- The parties agree on a contract
- The contract defines a set of rules
- The rules are coded in a program
- This program is stored in the nodes of the Blockchain
- The nodes of the Block chain will execute the program of the smart contract

EXAMPLE- FARMING SMART CONTRACT



USE CASE: SMART SWITCH CONTRACT IOT



Open questions

1. Is smart contract reversible?

No. there is set procedure to do so

- However, some smart contracts may include conditions that allow for reversibility under certain circumstances, such as if certain conditions are not met, or if a particular event occurs. For example, a smart contract may include a "timeout" condition, where if a transaction is not confirmed within a certain period of time, the contract can be reversed.

2. What are blockchain oracles?

Entities that connect blockchains to external systems, allowing smart contracts to execute depending on real-world inputs and output

Second Generation Tokens

Tokens:

- It is a **self authenticating data packets** that represent a rare **digital bit of information.**
- It gain ability to represent themselves and prove authenticity automatically because they generated on blockchain.

First Method:

- Tokens created by sending a special transaction on bitcoin that had a small message.
- This message says that some new assets had been created and were credited to a Bitcoin address.

Second Method:

- Tokens created by the complex transactions developed by blockchains.
- Example: Ethereum foundation improved token technology by building a framework for the programming of tokens and allowing them to be used for **more complex tasks.**

TOKENS STANDARDS

- ERC 20
- ERC 721
- ERC 7411

SECOND GENERATION TOKENS

Tokens saw a surge in popularity in 2017 and 2018

- Stable Coins
- Security Tokens

DECENTRALIZED APPLICATIONS(DApps)

- A decentralized application (dApp) is a type of **distributed open source software application** that runs on a **peer-to-peer** (P2P) blockchain network rather than on a single computer.

- The decentralized nature of dApps means that once a developer has released a dApp's codebase, others can build on top of it.

- Utilizes smart contracts w.r.t different applications
    - Bitcoin cannot do this exactly
    - Ethereum- A game changer
- They are more **flexible, transparent, distributed, resilient**, and have a better incentivized structure than current software models.
- It will work with cryptographically stored ledger.
- It is a scarce-asset model, and peer-to-peer technology.

Requirements of Dapps

1. Open source:

- Codebase should be freely available for all.
- Any changes in the structure or working of the app should only be taken by agreement of the majority

2. Decentralized:

- All the information and operations should be stored on a public and decentralized Blockchain
- Ensure security and transparency

3.Censorship resistant:

- Not bounded to any legal agencies
- Good/bad?

4. Incentivize:

- dApps should offer some sort of incentives to their users in the form of cryptographic tokens.
- These are a sort of liquid assets and they provide incentives for users to support the Blockchain dApp ecosystem

5.Never go offline:

- At least one system which is awake is sufficient for DAPPs to work correctly
- Speed is slow however

6.Protocol:

- dApps should have a particular protocol to demonstrate proof of value.

- This means showing the value of a particular process in a way that can be easily verified by others.

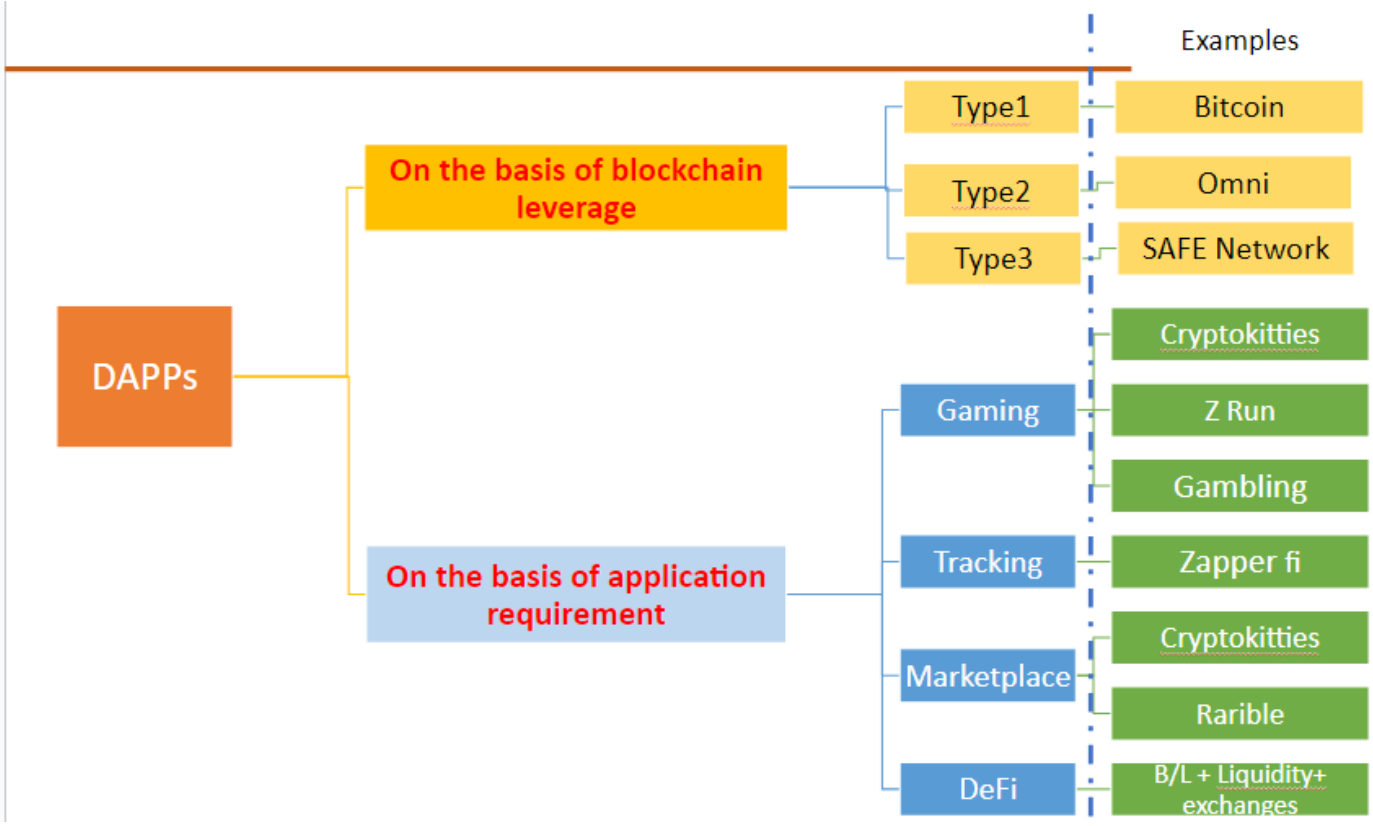
Benefits of Dapps

- **Zero downtime**
  - able to serve clients
  - cannot launch DOS attacks
- **Privacy**
  - You don’t need to provide real-world identity to interact with a dapp.
- **Resistance to censorship**
  - No single entity has control to block users
- **Complete data integrity**
  - Data is immutable ,indisputable(due to cryptographic primitives.)
- **Trustless computation/verifiable behavior**

Drawbacks of Dapps

- **Maintenance**
  - Code and data published to the blockchain are harder to modify.
- **Performance overhead**
  - Scaling is really hard.
  - To achieve the level of security, integrity, transparency, and reliability - **every node runs and stores every transaction.**
  - Proof-of-work takes time as well.
- **Network congestion**
  - When dapp uses too many computational resources, the entire network gets backed up.
- **User experience**

Types of Dapps



On basis of blockchain leverage:

- **Type 1:** Bitcoin
- **Type 2:** Omni
- **Type 3:** SAFE Network

On basis of application requirement

- **Gaming**
  - Cryptokitties
  - Z Run
  - Gambling
- **Tracking**
  - Zapper fi
- **Marketplace**
  - Cryptokitties
  - Rarible
- **DeFi**

- B/L + liquidity+ exchanges

How dapps work

- Dapps have their **backend code (smart contracts) running on a decentralized network** and not a centralized server.
- Use **Ethereum blockchain for data storage** and **smart contracts for their app logic.**
- Once dapps are deployed on the Ethereum network, you can't change .

Illustration of Dapp

- The **frontend** of a decentralized application represents what you see.
- The **backend** represents the entire business logic.
- This **business logic** is represented by one or several smart contracts interacting with the underlying blockchain.
- The frontend, as well as files like a photo, a video, or audio, could be hosted on decentralized storage networks such as **Swarm or IPFS.**

Traditional Web Applications

- Traditional websites: **Front End → API → Database.**

Decentralized Applications

- It contains a “wallet” that communicates with the blockchain.
- The wallet manages cryptographic keys and the blockchain address.
- **Public-key infrastructure** is used for user identification and authentication.
- Instead of an API connecting to a database, a **wallet software triggers activities of a smart contract**, which interacts with a blockchain: Web3 compatible website:
- Decentralized Apps: **Front End (including wallet) → Smart Contract → Blockchain**.

The Rise of WEB3 over WEB2

Web1: you only get the information
Web2: you can do exchange from both the sides
Web3: You want to own the data

Differences:

- Web3 applications need a connection to the blockchain- “wallet.”
- It keeps a record of the **private keys** and **blockchain address,** which represents the **unique 30 identities** and point of reference.
- In the backend, the Web3 adds a whole new infrastructure layer for decentralized applications to interact with – the decentralized protocol stack.

- Decentralized apps need to have a **component that manages a user's private keys,** with which one can sign transactions on the state layer, the blockchain.

HOW DAPPS FIT IN THE REAL WORLD?

- Money Management & Transfer
- Business Process Management
- DAO (Decentralized Autonomous Organization)

WHY CHOOSE DAPP DEVELOPMENT OVER CONVENTIONAL APP DEVELOPMENT?

- **Faster & payment processing** without needing to integrate payment gateway to accept funds.
- **High levels of data security** due to smart contracts governed by private keys.
- **Greater anonymity** without needing the users to follow the lengthy signup process.
- **Reliable data records** as users can access the public blockchain to verify transaction information.

KEY FEATURES

A distributed app, abbreviated as "DApp", is a piece of software that is run on a **distributed or cloud network** rather than on a single dedicated server.

- More resistant to attack as there is no single point of failure that can be undermined.

DECENTRALIZED AUTONOMOUS ORGANIZATIONS (DAOs)

Dao is a subset of DApp

- Decentralized application- requires human to give some input

What is the difference between DAPPs and DAOs?

- If you remove a human usage, from a smart contract→ it is called DAO
- Dapp has interaction with humans through smart contract

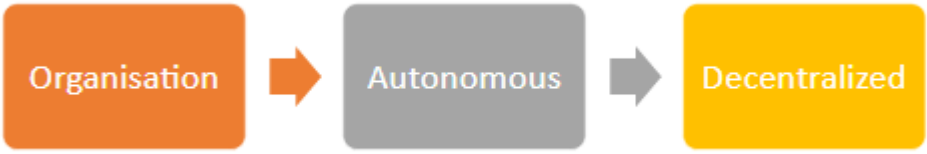
DAO is an organisation with no human intervention machine smart contracts takes care of failures

- uses the term stakeholders
- DAO is used when we talk about

two types: **profitable and non profitable**

- For example, they could be used for voting for a new government, or by shareholders in a public or private corporation irrespective of the size.
- The **Bitcoin network is considered to be a DAO.**

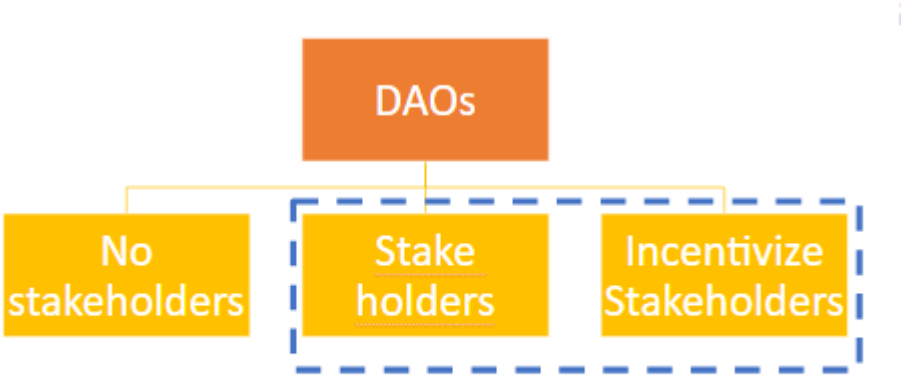
- It allows any person to join in the creation and security of the system and cooperation is completely voluntary.
- The rules for the network are held within its consensus protocol, and the **miners vote by choosing what blocks to build on**.



Organization → is any task you want to do with a set of people, but with a registered name and goal

Autonomous → can make decisions on their own, without any permissions(treasury smart contract)

Decentralized → There is no one person in charge



DAOs → DB entry + treasury smart contract

Treasury smart contract → where you have to give money/assets
smart contracts makes the decisions of where to spend the money

Stake holder→

Three types of DAO:

- **No stakeholders** : the code and smart contract is running , no one is going to stop it. The code takes care of all faulty scenarios
- **Stakeholders**: there are actual people who have, stakes/tokens where they can participate in the DAO process
- **Incentivized Stakeholders:**

Can a stakeholder be byzantine?
yes.

Need for DAOs over traditional organizations

Principal agent dilemma

This dilemma is a **conflict in priorities between a person or group** (the principal) and those **making decisions and acting on their behalf** (the agent).

Problems can occur in some situations :-

- The **agent (the CEO)** may work in a way that's not in line with the priorities and goals determined by the principal (the stakeholders) and instead **act in their own self-interest.**
- The agent takes excessive risk because the **principal bears the burden.**

For example:- a trader can use extreme leverage to chase a performance bonus, knowing the **organization will cover any downside.**

DAOs **solve** the principal-agent dilemma **through community governance.**

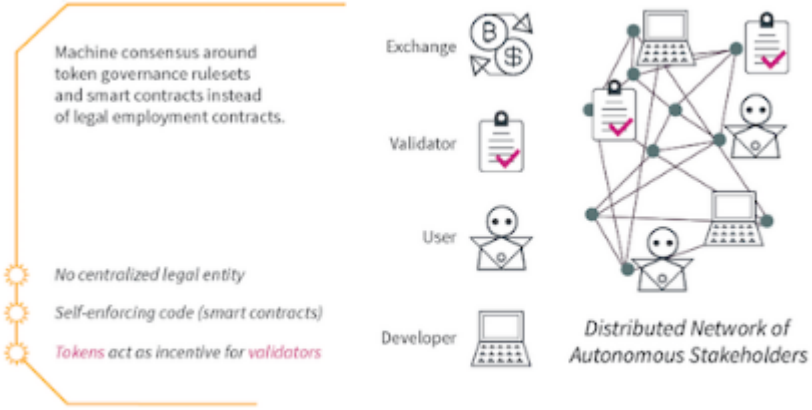
- Stakeholders aren't forced to join a DAO
- They don't need to trust any agent acting on their behalf

Advantages of DAO

- **Lack of trust needed between two parties** - only the code needs to be trusted.
- Code publicly available and **extensively tested** before launch.
- **Every action** a DAO takes after being launched has to be approved by the community and is completely **transparent and verifiable.**
- organization has **no hierarchical structure** - controlled by stakeholders via its **native token.**
- **Any stakeholder** can put forward an **innovative idea** that the entire group will consider and improve upon.
- **Internal disputes** solved through the **voting system**(through the pre-written rules in the smart contract)

TRADITIONAL ORGANIZATIONS

DECENTRALIZED AUTONOMOUS ORGANIZATIONS



HOW DAOs WORK?

- DAOs run through **rules encoded within their smart contracts**.
- They **live completely online** but can govern assets that are offline, like real estate or natural resources.
- DAOs allow **two parties** that have never met **to cooperate** and make decisions
- DAOs enable these individuals to do things like **hiring other individuals** to perform tasks that can't be automated.
- Many DAOs have hired individuals to develop software, for example.
  DAOs **rely on tokens to gain cooperation.**

There are different models for DAO membership.

Membership can determine how voting works and other key parts of the DAO.

- Token-based membership
- Share-based membership

LEGALITY OF DAOs

- If you are thinking about creating a DAO, seek legal counsel.

- Many countries have begun to create a legal framework that allows for the unique nature of DAOs(security, checking the idea ,proof of value)
- For example, Malta created a legal framework for DAOs that classifies them as a new type of legal entity, referring to them as **"technology arrangements".**
- **Malta** has created a new regulatory body called the **Digital Innovation Authority (MDIA).**

THE DAO STORY

- first DAO was created by a company called Slock.it(German-based)
- Built on top of Ethereum blockchain
- In early 2016, Slock.it wanted to find a way to raise money, according to an interview with the company's founder Christoph Jentzsch from 2018, and in April created a DAO, which was akin to a Kickstarter or GoFundMe.
- The distinguishing factor: the **DAO gave all of the investors/members a vote in the decisions** the company makes when spending the raised capital.
- Raised over **$150 million worth of Ether**
- But by June of that year, the **DAO was hacked and lost about $50 million**
- DAO never regained its original status.

Pros of DAO

- Decentralization
- DAOs Provide "Skin in the game" for Participants
- Community Driven

Cons of DAO

- A disputed point is trust.
- People develop smart contracts. Therefore, it can have errors.
- Thus, the whole trust lies in the code.

DAOs goes well with Dapps as code visibility is there.

Difference b/w DA, DO, DAO, DAC, AA, DIF, DAECO?

1. **DA (Decentralized Autonomous):** A decentralized autonomous organization is a blockchain-based organization that operates through a set of smart contracts, with no central control or ownership. It typically uses tokens to represent ownership and has a democratic governance structure.

2. **DO (Distributed Organization):** A distributed organization is similar to a decentralized autonomous organization but does not necessarily rely on blockchain technology. It can be a network of nodes or participants that work together to achieve a common goal.

3. **DAO (Decentralized Autonomous Organization):** A DAO is a specific type of decentralized organization that operates through smart contracts and blockchain technology. It typically has a set of rules encoded in its smart contracts that govern its operations, including voting and decision-making.

4. **DAC (Decentralized Autonomous Corporation):** A DAC is a type of DAO that focuses on creating value for shareholders. It operates like a traditional corporation, with profit-sharing and voting rights for shareholders, but without centralized control or ownership.

5. **AA (Auction and Announcement):** An AA is a decentralized market mechanism used to distribute tokens. It involves an initial auction of tokens, followed by a continuous distribution through a series of smaller auctions.

6. **DIF (Decentralized Impact Fund):** A DIF is a type of decentralized organization that focuses on creating social or environmental impact. It uses blockchain technology to ensure transparency and accountability in its operations.

7. **DAECO (Decentralized Autonomous Economic Organization):** A DAECO is a type of DAO that focuses on creating economic value through decentralized decision-making. It typically uses smart contracts to automate economic processes, such as pricing and revenue sharing.

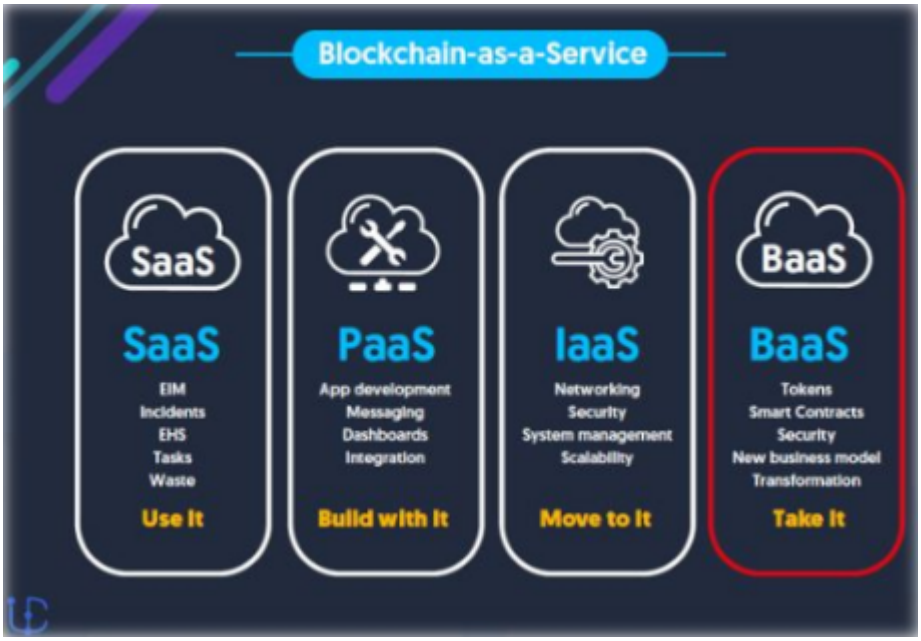
BLOCKCHAIN-AS-A-SERVICE (BAAS)

- Blockchain-as-a-service (BaaS) refers to **third-party cloud-based infrastructure and management for companies** building and operating blockchain apps.

- BaaS functions like a sort of **web host, running the back-end operation** for a block-chain based app or platform.

- The business of blockchain technology has moved well beyond its best-known use in cryptocurrency transactions and has broadened to address secure transactions of all kinds.

As a result, there is a demand for hosting services.



Major Players in the BaaS

Major players in the BaaS space include:

- **Microsoft partnered with ConsenSys-**>Ethereum blockchain-as-a-service on Microsoft Azure
- **Amazon**→ Amazon Managed Blockchain
- **R3**→distributed financial ledger called Corda
- **PayStand**→ which specializes in sending and receiving payments between companies.

How to Handle Security Concerns Related to BaaS?

- Choose a reputable BaaS provider
- Implement strong access controls
- Secure communications → such as SSL/TLS, to encrypt data in transit
- Regularly monitor and audit activity
- Implement disaster recovery and business continuity plans
- Regularly review and update security measures

Criteria for Selecting a Blockchain as a Service Partner

- Prior experience in setting up Blockchain infrastructure/Market Credential
- Commitment to Quality
- Security Assurance
- Choice of Operating Systems
- Ease of Use
- Pricing and Support

Factors that can help to choose the right BaaS Platform

- Smart Contracts Integration
- IAM(Identity Access Management) Platforms
- Different Runtimes and Frameworks
- Identity-based Consensus Mechanisms

ADVANTAGES of BaaS

- Flexible and Efficient
- Open and Easy
- Privacy Protection and Security
- Cost-Effective

DISADVANTAGES of BaaS

- Limited In-depth Knowledge
- Lack of Visibility and Control
- Data Compliance Challenges
- Business Continuity

HYPERLEDGER FABRIC

- **The Linux Foundation** founded the Hyperledger project in **2015**
- Hyperledger Fabric is one of the blockchain projects within Hyperledger.
- Like other blockchain technologies, it has a ledger, uses smart contracts, and is a system by which participants manage their transactions.
- The members of a Hyperledger Fabric network **enroll** through a **trusted Membership Service Provider (MSP)**.
- Hyperledger is an **enterprise-grade, open-source distributed ledger framework**
- Fabric is a highly-modular, **decentralized ledger technology (DLT)** platform that was **designed by IBM**.
- Hyperledger Fabric is **private and requires permission to access**, businesses can segregate information (like prices), plus transactions can be **speed up** because the **number of nodes on the network is reduced.**
- **Fabric 2.0 was released in January 2020**. The main features of this version are **faster transactions, updated smart contract technology, and streamlined data sharing.**

Key design features of Hyperledger Fabric Model

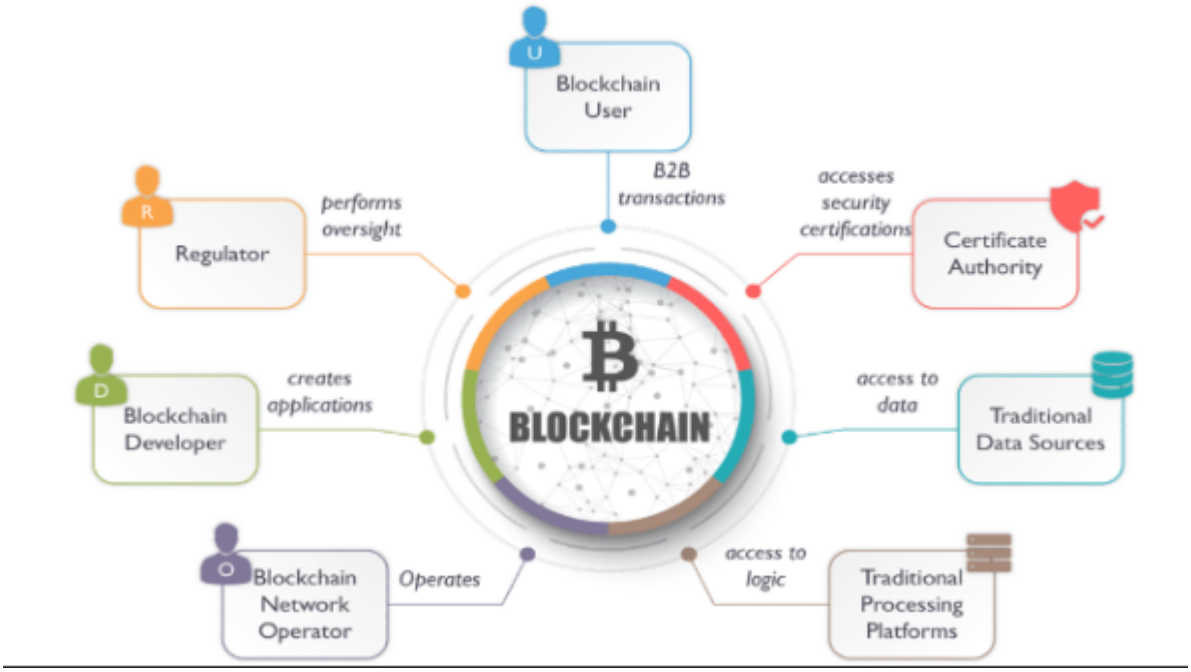
- Assets
- Chaincode
- Ledger Features
- Privacy
- Security & Membership Services

- Consensus

Key Benefits of Hyperledger Fabric

1. **Data Protection & Consistency**
   - Use permissions to ensure accountability of membership & access rights
2. **Confidential transactions**
   - Give businesses the flexibility & security to make transactions visible to select parties with the correct encryption keys
3. **No Cryptocurrency**
   - No mining & expensive computations
4. **Programmable**
   - Leverage the embedded logic in smart contracts to automate business process across your network

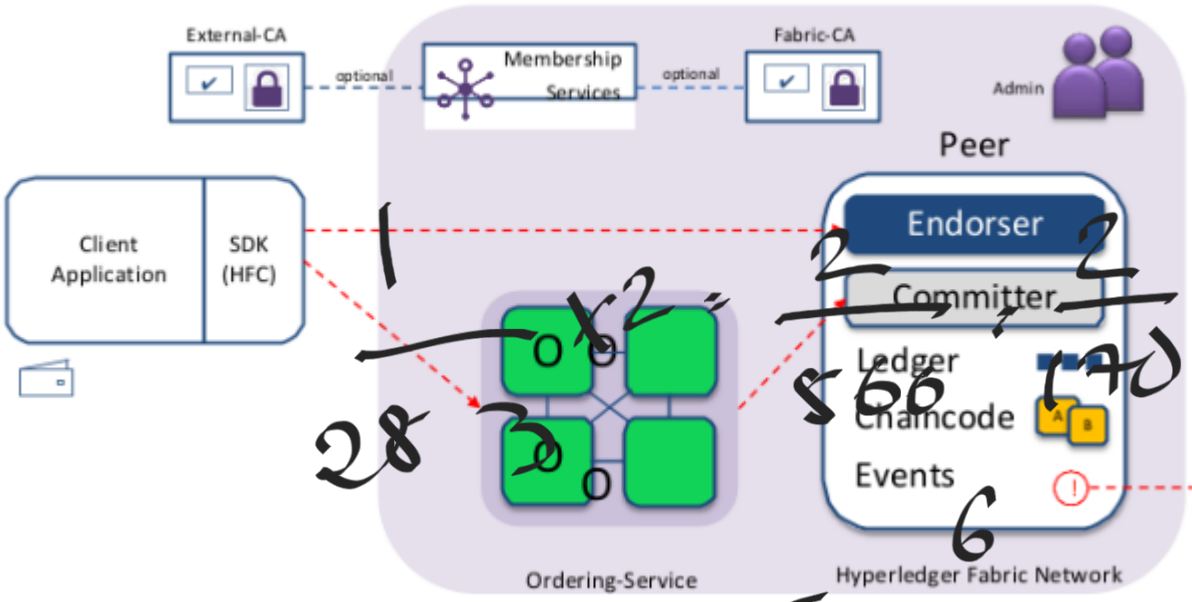
Participants of Hyperledger Fabric



1. **Regulator**- performs oversight
2. **Blockchain developer** - creates apps
3. **Blockchain network operator**- operates
4. **Traditional processing platforms** -access to logic
5. **Traditional data sources**- access to data
6. **Certificate authority** - accesses security certifications
7. Blockchain user B2B blockchain user

Hyperledger Fabric V1 Architecture

22x18 = 396



Nodes and Roles

Committing peer:

- Maintains ledger and state
- **Commits transactions** valid- maintains blockchain
- May hold smart contracts(not compulsory)
(Whatever is reaching to them is post the execution of the smart contract)
- keeps copy of ledger

Endorsing peer

Specialized committing peer that receives a transaction proposal for endorsement responds **granting or denying endorsement**. Must hold smart contract

- Has to hold smart contract and execution of the smart contract
- Producing output and transferring the output to another node
- Keeps copy of ledger

Ordering node

Approves the inclusion of transaction blocks into the ledger and communicates with committing and endorsing peer nodes.

- Does not hold ledger
- Only **orders the transaction**, so consistency is maintained

Transaction flow

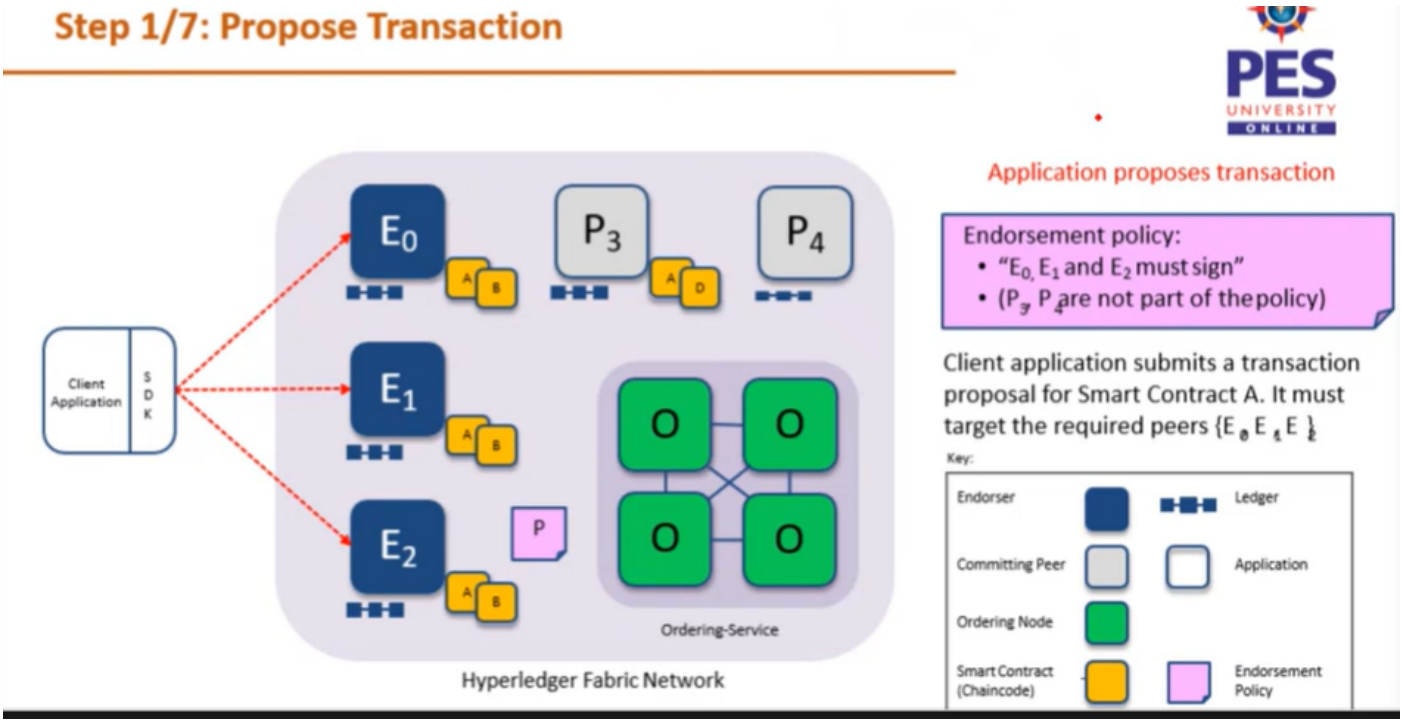
endorse→ order→ validate

How it works

- Client asks for a query
- Query is checked if its a valid query or not done in endorsing part
- The endorses peers signs output and gives it back
- Depending on the responses the transactions are collected in the orderer state
- Validation stages two things are checked
  - Double spending
  - There is variable x has 100, to add 500 or 5000 rs transactions. Managing variables conditions

Transaction flow

1. Propose transactions



- Has to **check whether the transaction is valid** and give responses back to the client- **Endorsement policy**
- Can client make a payment of 100 with a balance of 150? yes
- **Doesn't check for double spending**

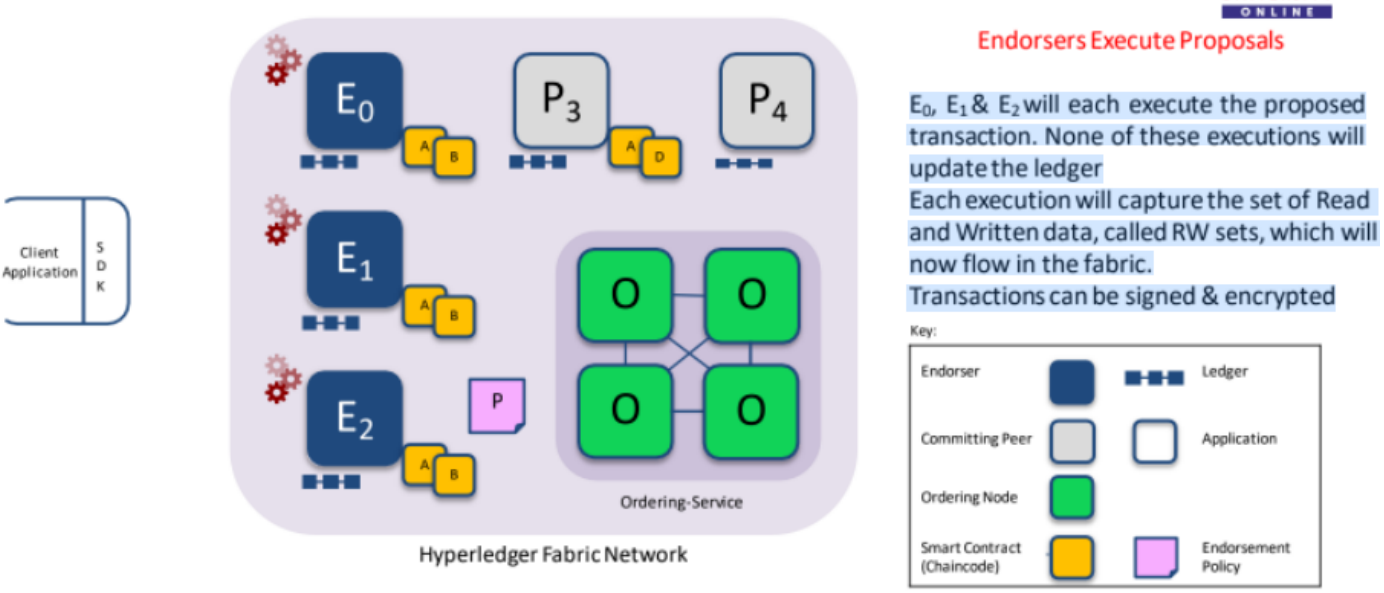
When will the endorses peers give the data back? On completion of endorsement policy

- **E0, E1, E2 have to sign**(but this must be stated in endorsement policy)
- p3, p4 not part of the policy

Client application submits a transaction proposal for smart contract A. It must target the required peers(E0, E1, E2)

2. Execute Proposed Transaction

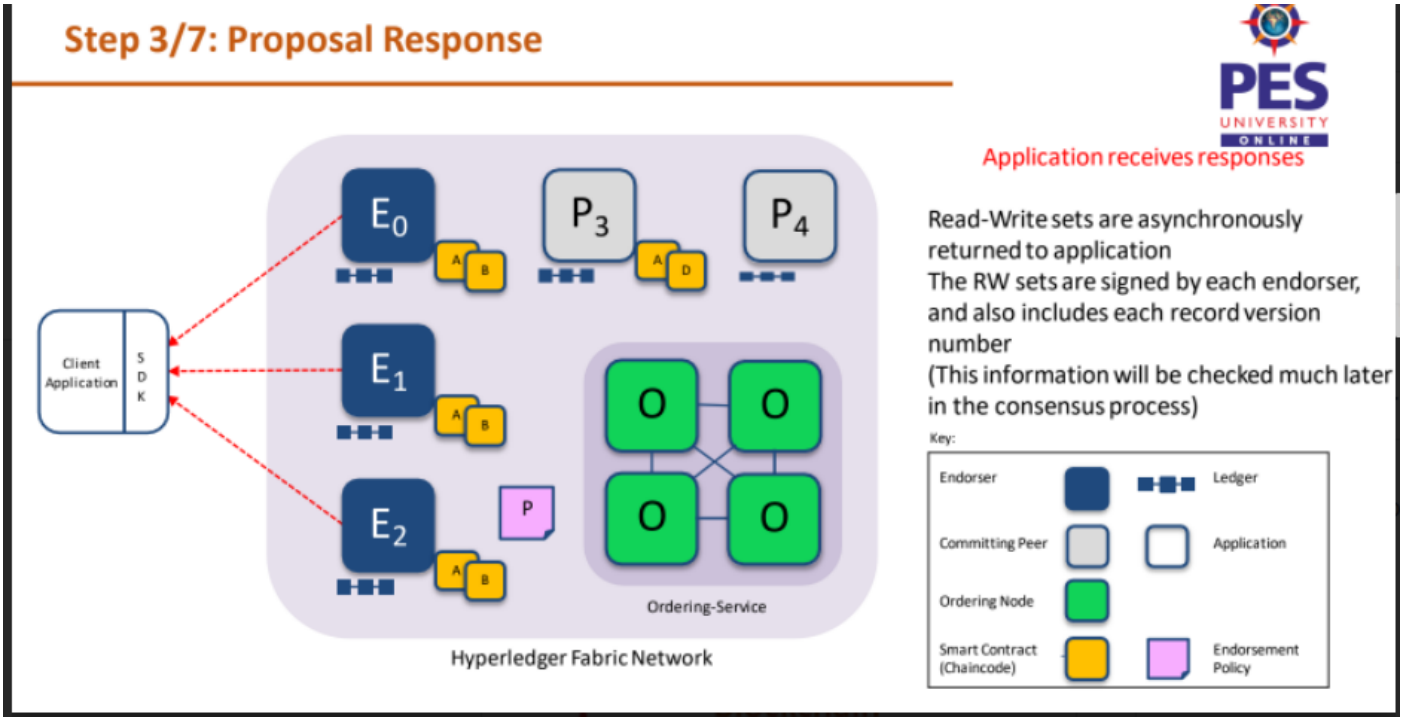
Step 2/7: Execute Proposed Transaction



E0, E1& E2 will each execute the proposed transaction.

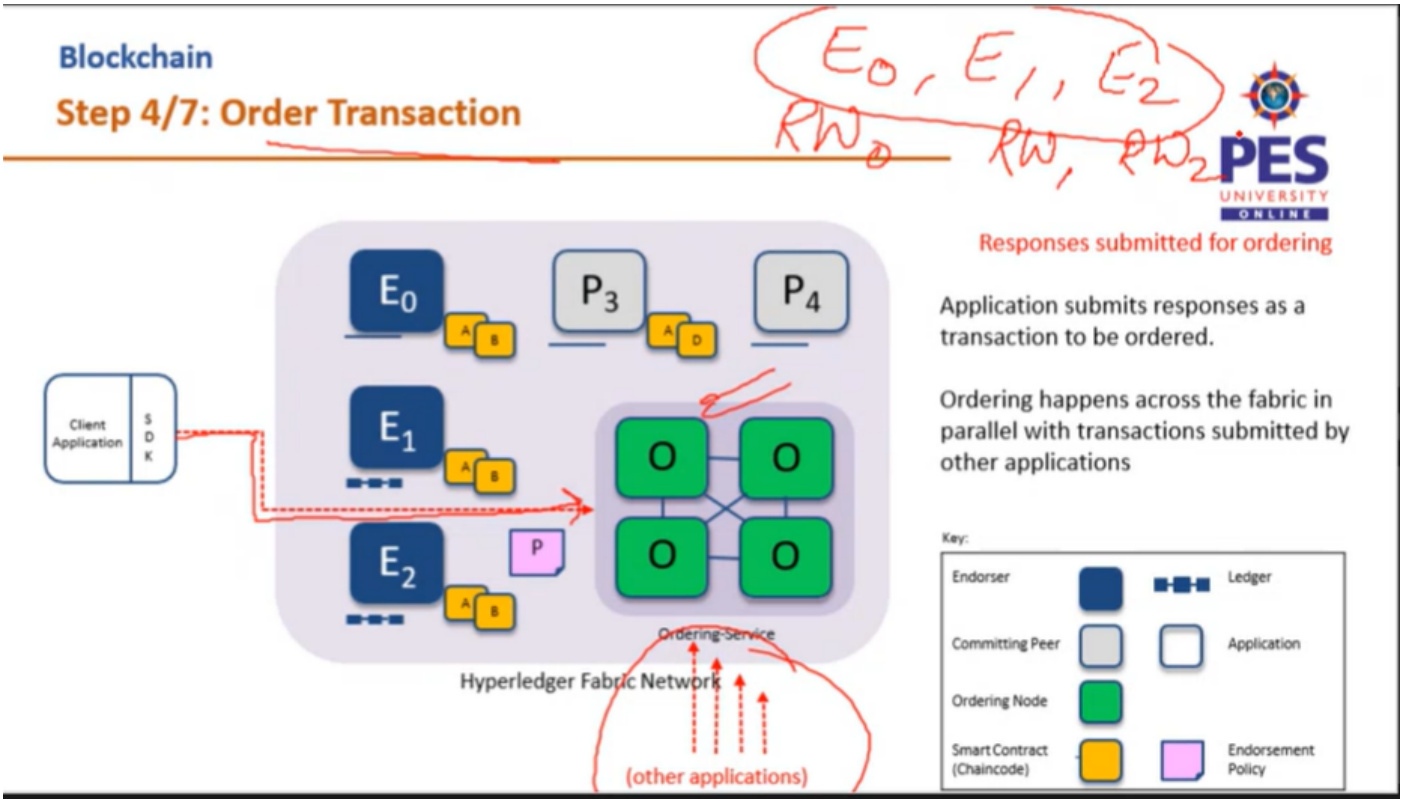
- None of the executions will update the ledger **(not committed)**
- Reply sent to back to the client **(RW sets sent back)**
- Future analysis , what will be state of machine at that time
- The RW set is signed with their **digital signatures+ encryption** so that the info isn't changed .
- Valid endorser that has updated the transaction
- Each execution will capture the set of **Read and Written data, called RW sets,** which will now flow in the fabric.
- Transactions can be signed & encrypted

3. Proposal Response



- Read-Write sets are **asynchronously returned to application**
- The RW sets are **signed by each endorser**, and also includes each **record version number** (This information will be checked much later in the consensus process)- checks only whether the given is valid node

4. Order Transaction

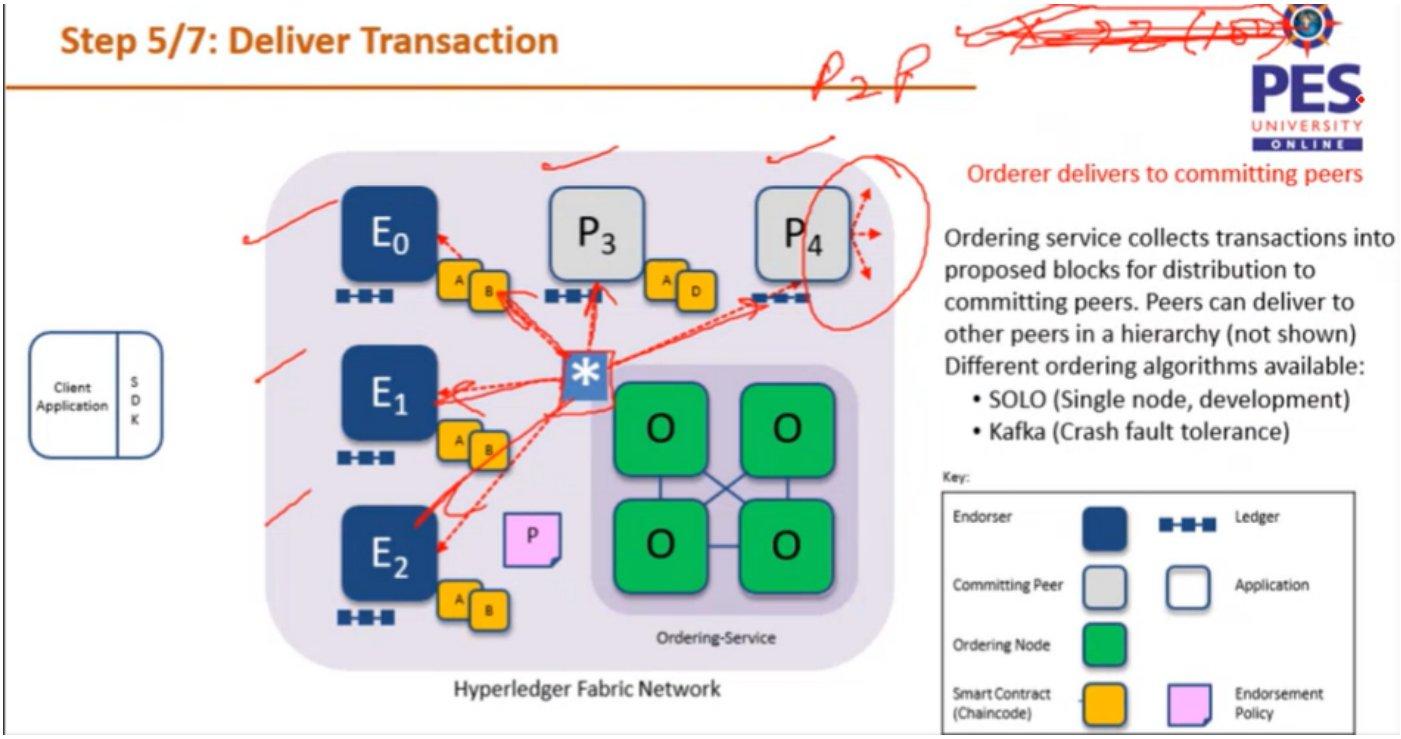


- **If the responses** from the endorsers **are the same** then it forward the transactions and responds to orderer to order the transactions

- Application submits responses as a transaction to be ordered.
- Ordering happens across the fabric in parallel with transactions submitted by other applications
  You can have different ordering services or shared ordering services shared by all applications

Client doesn't care about how many responses it got, only whether they are the same

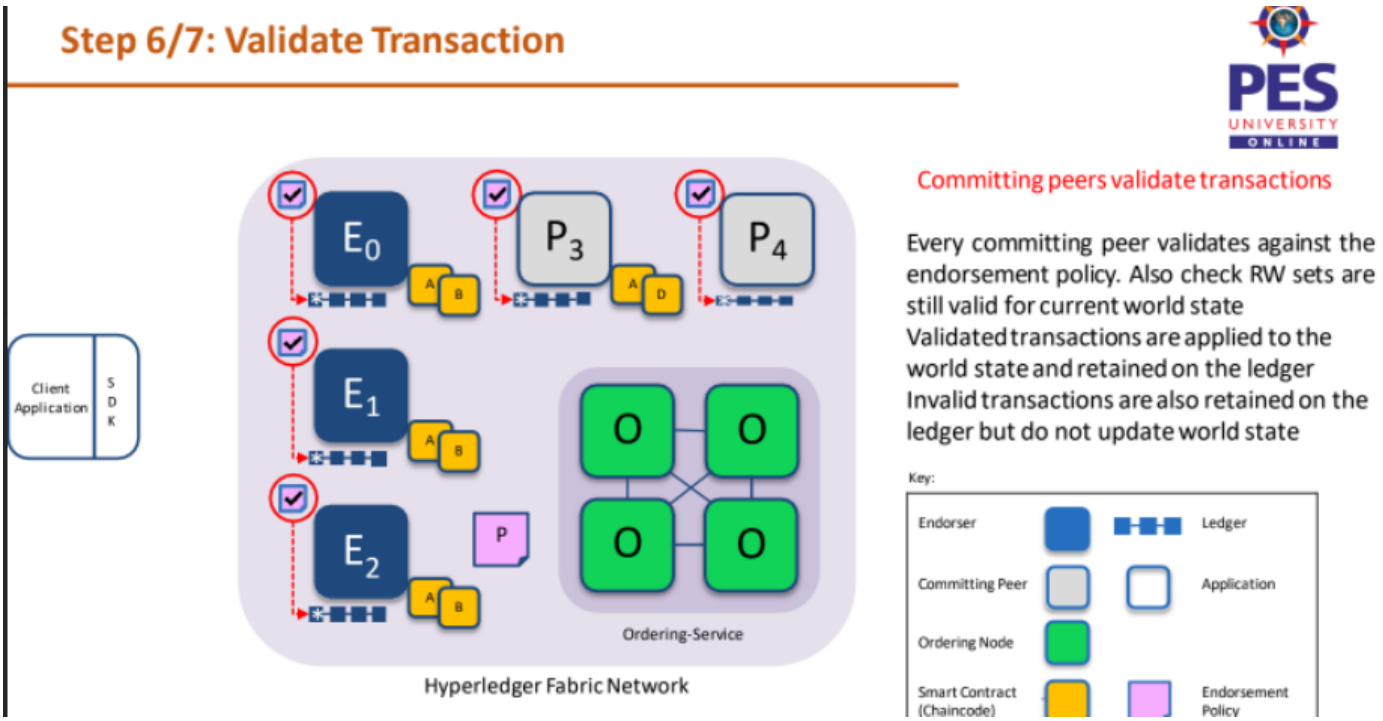
5. Deliver Transaction



- **Orderer delivers to committing peers**
- Ordering service **collects transactions into proposed blocks for distribution to committing peers**. Peers can deliver to other peers in a hierarchy.
- Different ordering algorithms available:
  - **SOLO** (Single node, development) fifo
  - **Kafka** (Crash fault tolerance)

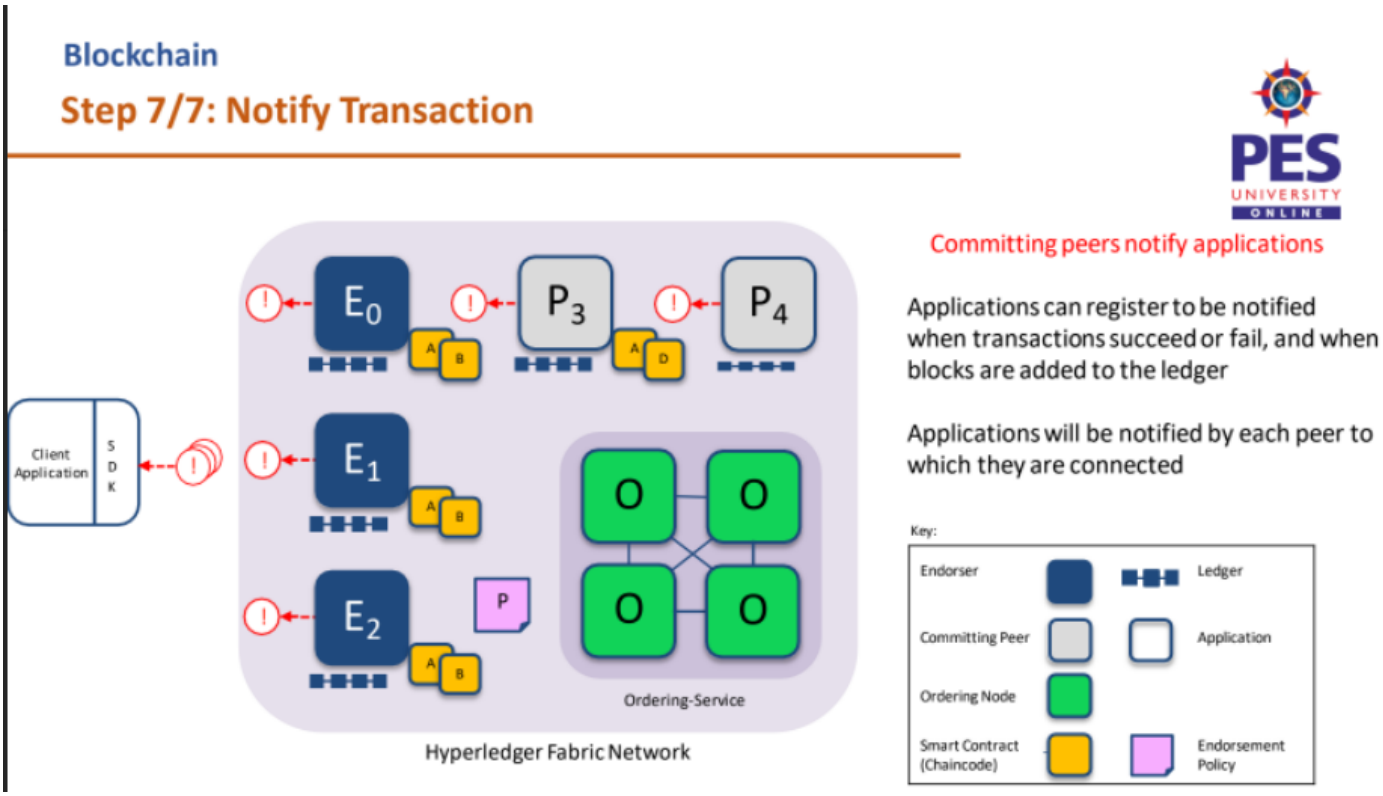
SOLO- first come first serve(2 transactions at the same time, can vary)
KAFKA- needs minimum number of nodes - 3

6. Validate Transaction



- **Committing peers validate transactions**
- Every committing peer **validates against the endorsement policy.**
- Also **check RW sets** are still valid for **current world state**
- Validated transactions are applied to the world state and retained on the ledger
- Invalid transactions are also retained on the ledger but do not update world state

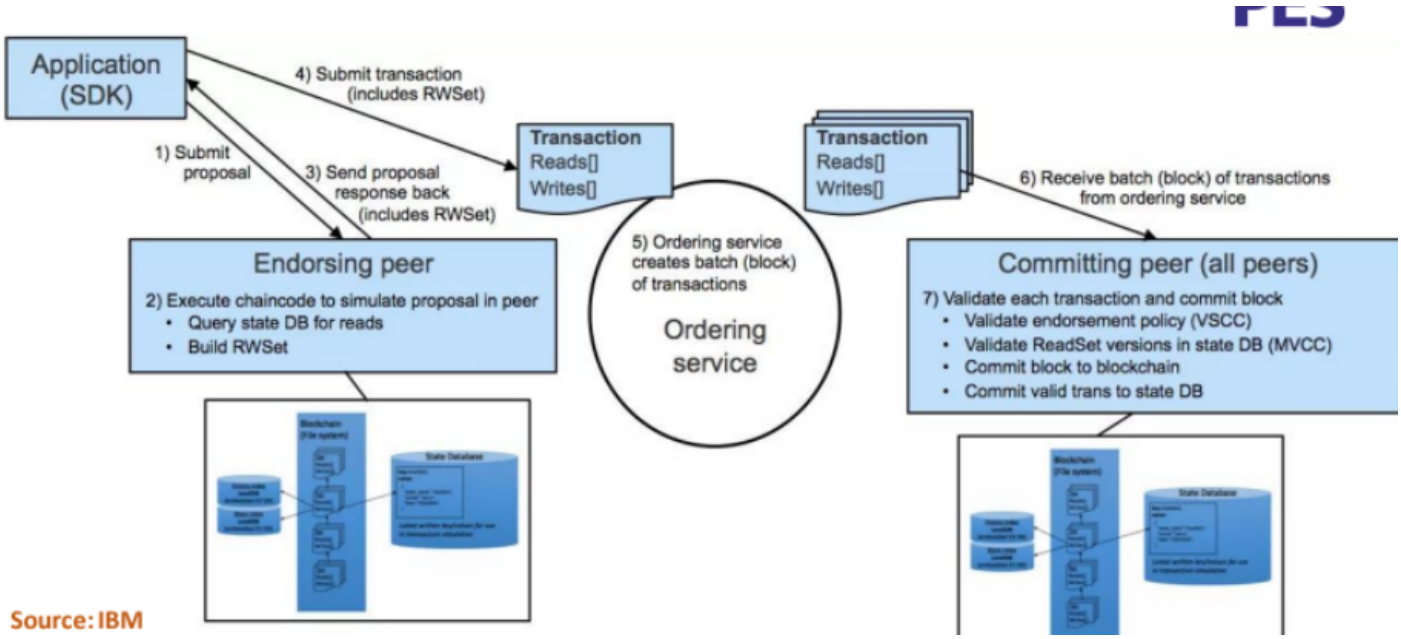
7. Notify Transactions



- **Committing peers notify applications**
- Applications can register to be notified when transactions succeed or fail, and when blocks are added to the ledger
- Applications will be notified by each peer to which they are connected
- You can code without smart contracts and still update the ledger

Key Benefits of the Transaction Flow

- Better reflect business processes by specifying **who endorses transactions**
  - *Eliminate non-deterministic* transactions
  - Scale the number of participants and transaction throughput



HYPERLEDGER FABRIC CORE COMPONENTS

Ordering Service

- The ordering service **packages transactions into blocks** to be delivered to peers. Communication with the service is via channels.
- Different configuration options for the ordering service include:
  SOLO:
  - Single node for development
    Kafka : Crash fault tolerant consensus
  - 3 nodes minimum
  - Odd number of nodes recommended

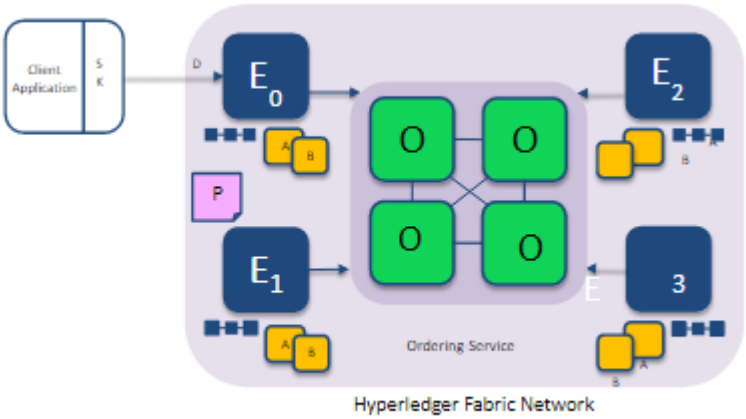
Channel

Channels provide **privacy between different ledgers**

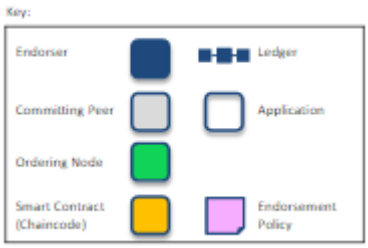
- Ledgers exist in the scope of a channel
  - Channels can be **shared across an entire network** of peers
  - Channels can be **permissioned for a specific set** of participants
- Chaincode is **installed** on peers to access the world state
- Chaincode is **instantiated** on specific
- Peers can participate in multiple channels
- Concurrent execution for performance and scalability

Single Channel Network

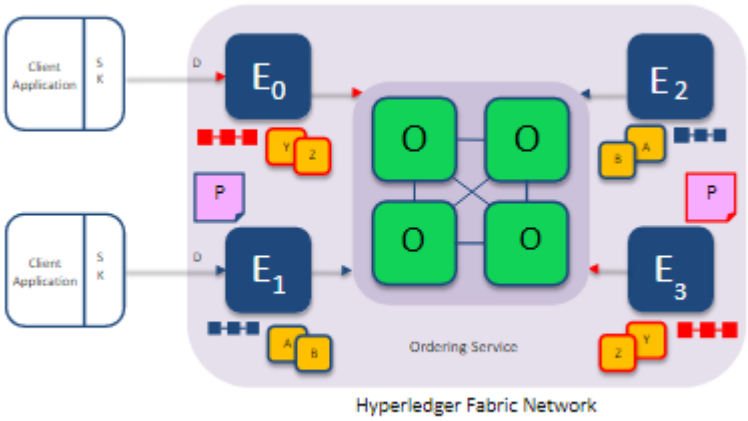
- All peers connect to the same system channel (blue).
- All peers have the same chaincode and maintain the same ledger
- Endorsement by peers E0, E1,E2 and E3



Hyperledger Fabric Network



Multi-Channel Network



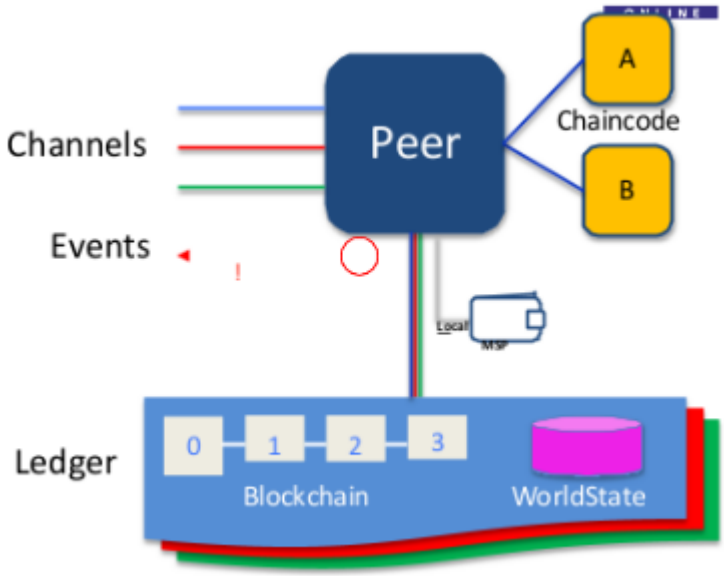
Hyperledger Fabric Network

- Peers E0 and E3 connect to the red channel for chaincodes Y and Z
- Peers E1 and E2 connect to the blue channel for chaincodes A and B

Fabric Peer

Each peer:

- Connects to one or more channels
- Maintains one or more ledgers for each channel
- Chaincodes are instantiated in **separate docker containers**
- Chaincodes are shared across channels (no state is stored in chaincode container)
- **Local MSP (Membership Services Provider)** provides crypto material
- Emits events to the client application

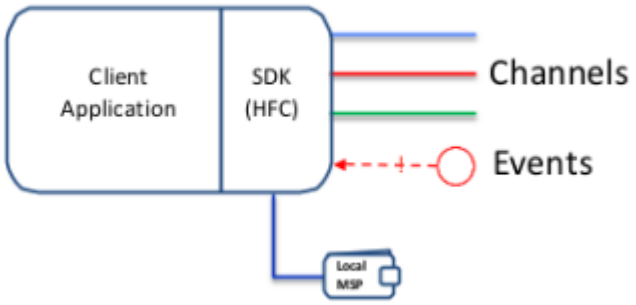


Client Application

Each client application uses Fabric SDK to:

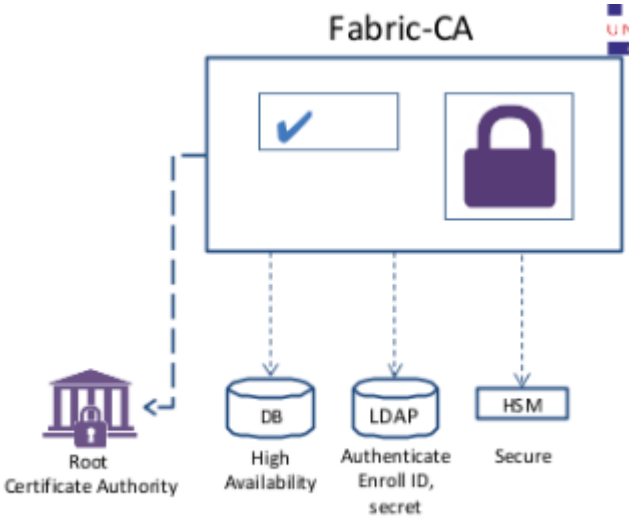
- Connects over channels to one or more peers

- Connects over channels to one or more orderer nodes
- **Receives events from peers**
- Local MSP provides client crypto material
- Client can be written in different languages (Node.js, Go, Java, Python)



Fabric Certificate Authority

- Default (optional) **Certificate Authority within Fabric network** for **issuing Ecerts** (long-term identity)
- Supports clustering for HA characteristics
- Supports **LDAP for user authentication**
- Supports **HSM for security**
- Can be configured as an intermediate CA



HYPERLEDGER FABRIC ARCHITECTURE

Organizations

Organizations define boundaries within a Fabric Blockchain Network
Each organization defines:

- Membership Services Provider(MSP) identities
- Administrators
- Users
- Peers
- Orderers(optional)

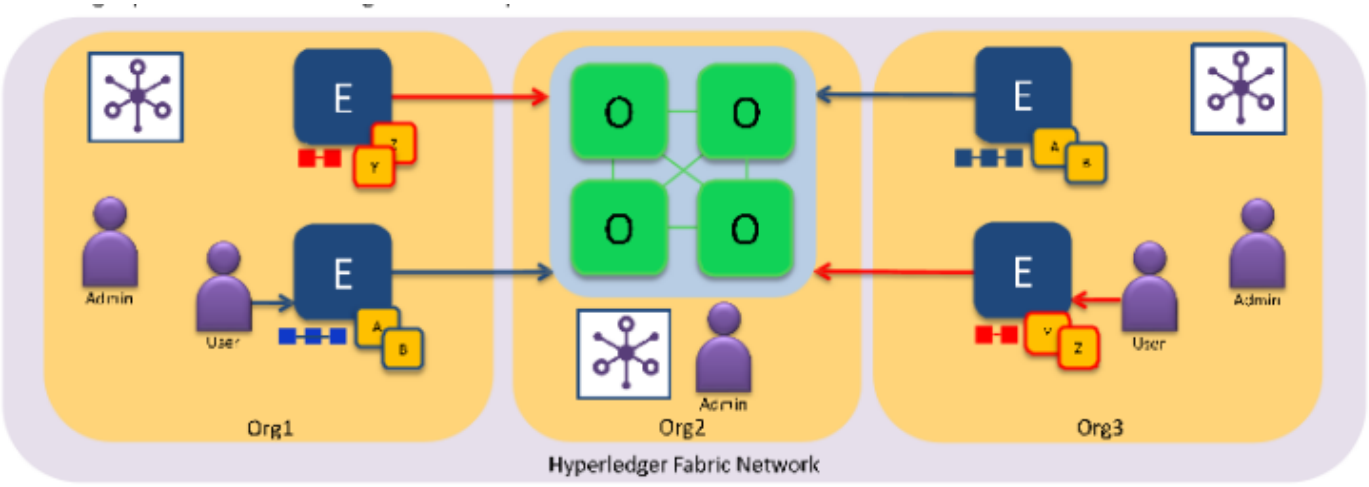
A network can include many organizations representing a consortium

- Each organization has an ID

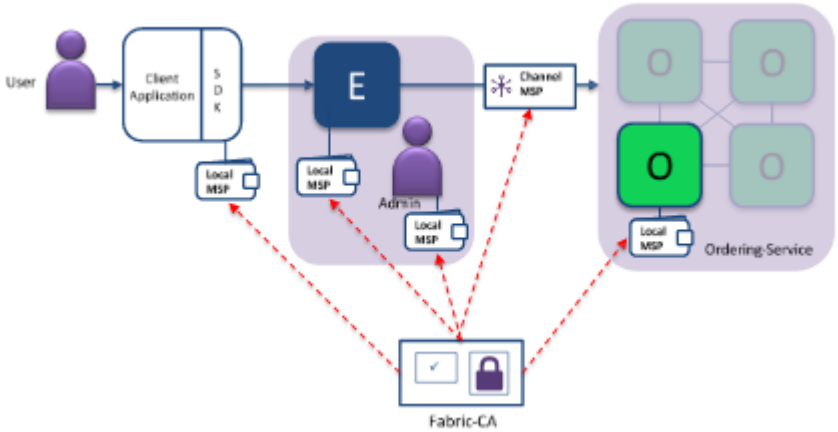
Consortium network

An example consortium network of 3 organizations

- Orgs 1 and 3 run peers
- Org 2 provides the ordering service only



Membership Service Provider



A MSP manages a set of identities within a distributed Fabric network

- **Provides identity for:**

    - Peers and Orderers
    - Client Applications
    - Administrators
- **Identities can be issued by:**
    - Fabric CA
    - An external CA
- **Provides:** Authentication, Validation , Signing and Issuance
- Supports different crypto standards with a **pluggable interface**
    - A network can include multiple MSPs (typically 1 per org)
    - Includes TLS crypto material for encrypted communications

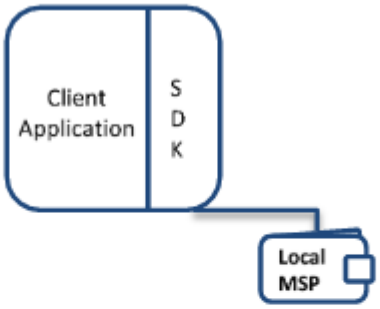
Transport Layer Security

- **Cryptographic protocol**s that **provide communications security** over a computer network
- Provides privacy and data integrity
- **Symmetric cryptography** is used to **encrypt the data** transmitted(privacy)
- **Public-key cryptography** is used to **authenticate the identities** of the communicating parties
- Include **message integrity check** to prevent loss or alteration of the data
- All component communication in Fabric secured using **TLS**(client-peer, peer-peer, peer-orderer, orderer-orderer)

User Identities

Each client application has a local MSP to store user identities

- Each local MSP includes:
    - **Keystore**- Private key for signing transactions
    - **Signcert** - Public x.509 certificate
- May also include **TLS credentials**
- Can be backed by a **Hardware Security Module(HSM)**



	user@org1.example.com
keystore	<private key>
signcert	user@org1.example.com-cert.pem

Admin identities

Each **Administrator** has a **local MSP to store their identity**

- Each local MSP includes:
  - **Keystore**- Private key for signing transactions
  - **Signcert** - Public x.509 certificate
- May also include **TLS** credentials
- Can be backed by a **Hardware Security Module(HSM)**

Peer and Orderer Identities

Each **peer and orderer has a local MSP**

- Each local MSP includes:
  - **keystore**- Private key for signing transactions
  - **signcert** - Public x.509 certificate
- In addition Peer/Orderer MSPs identify authorized administrators:
  - **admincerts**- List of administrator certificates
  - **cacerts** - The CA public cert for verification
  - **crls** - List of revoked certificates
- Peers and Orderers also receive channel MSP info

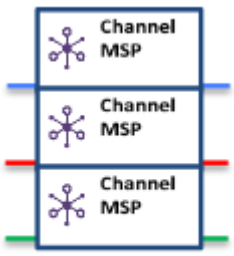
- Can be backed by a Hardware Security Module(HSM)

peer@org1.example.com	
admincerts	admin@org1.example.com-cert.pem
cacerts	ca.org1.example.com-cert.pem
keystore	<private key>
signcert	peer@org1.example.com-cert.pem
crls	<list of revoked admin certificates>

Channel MSP information

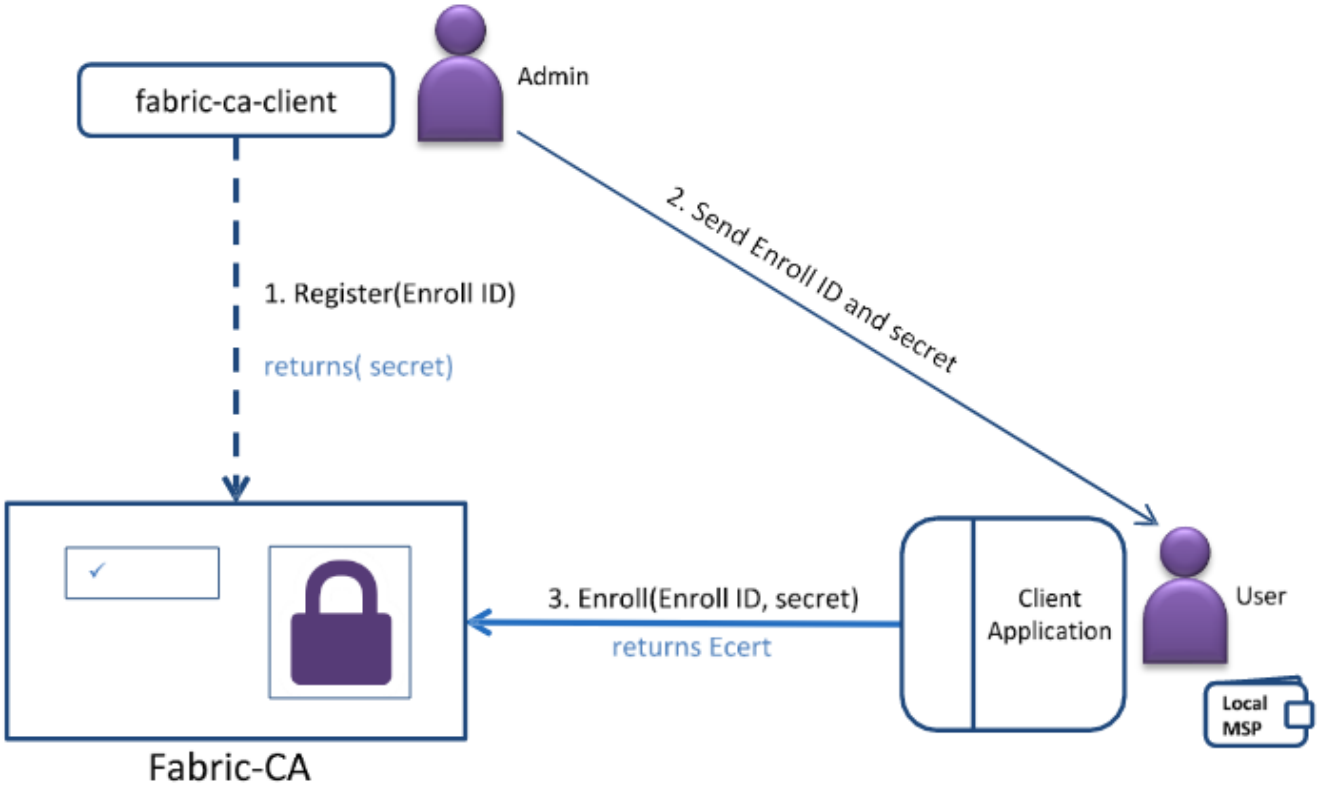
Channels include additional organizational MSP information

- Determines **which orderers or peers can join the channel**
- Determines **client application read or write access** to the channel
- **Stored in configuration blocks** in the ledger
- Each channel MSP includes:
  - **admincerts** - Any public certificates for administrators
  - **cacerts**- The CA public certificate for this MSP
  - **crls**- List of revoked certificates
- Does not include any private keys for identity



Channels

New User Registration and Enrolment

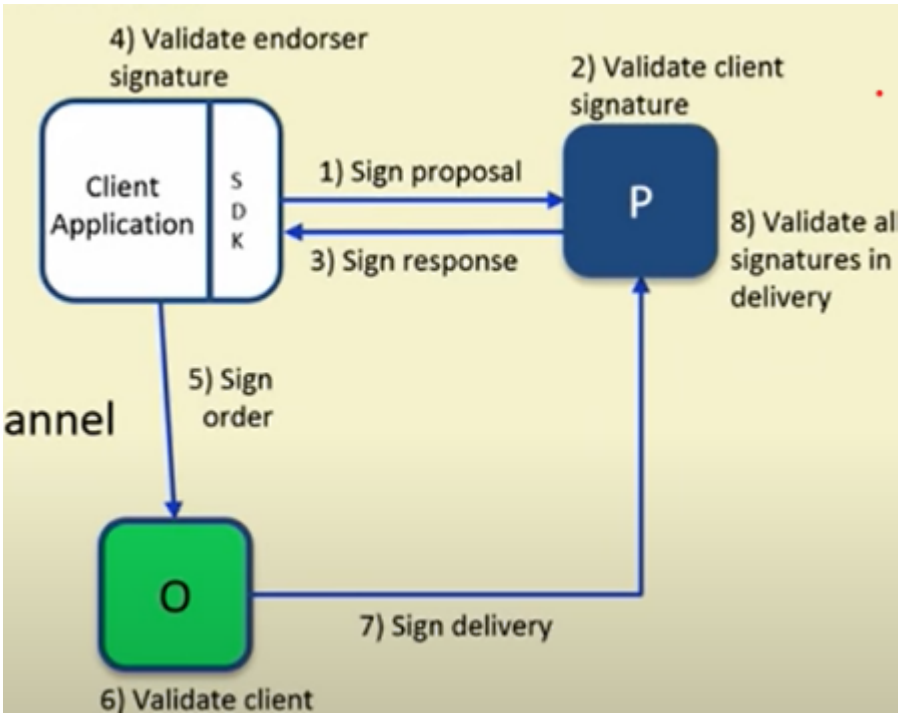


1. Admin registers new user with Enroll ID
2. Fabric CA generates a secret and returns it back to the Admin
3. The Admin sends the credential ie. Enroll ID and secret to the User
4. The User then sends these credentials to the Fabric CA for enrollment - Enroll(Enroll ID, secret)
5. The Fabric CA returns Ecert

Transaction Signing Flow

All transactions within a Hyperledger Fabric network are signed by permissioned actors, and those signatures are validated

- Actors sign transactions with their **enrolment private key**
  - Stored in their local MSP
- Components validate transactions and certificates
  - **Root CA certificates and CLRs stored in local MSP**
  - **Root CA certificates and CRLs stored in Org MSP in channel**



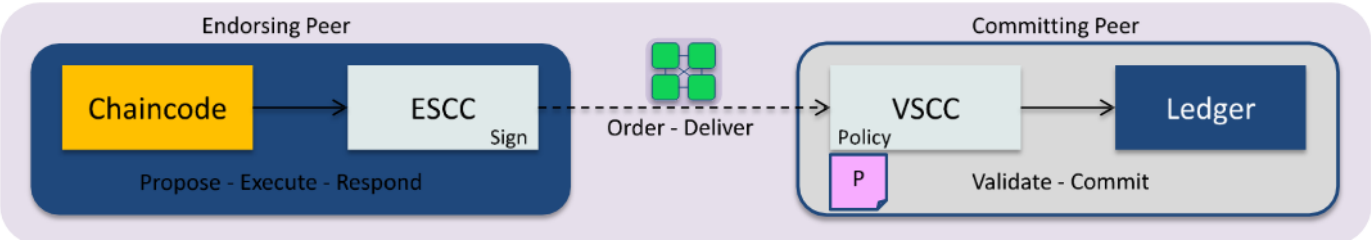
1. Client signs proposal - sends to endorser
2. Endorser Validates client signature
3. Endorser signs response and sends it back to client
4. Client validates endorsers signature
5. Client signs the order and sends to orderer
6. Orderer validates the client
7. Orderer signs for delivery
8. Endorses then validates all signatures in delivery

Endorsement Policies

- An endorsement policy describes the conditions by which a transaction can be endorsed.
- A transaction can only be considered valid if it has been endorsed according to its policy.

Each chaincode is deployed with an Endorsement Policy

- **ESCC** (Endorsement System Chaincode) signs the proposal response on the endorsing peer
- **VSCC** (Validation System ChainCode) validates the endorsements



Endorsement Policy Syntax

- Instantiate the chaincode **mycc** on channel **mychannel** with the policy **AND('Org1MSP.member')**

```
$ peer chaincode instantiate
-C mychannel
-n mycc
-v 1.0
-p chaincode_example02
-c '{"Args":["init","a", "100", "b","200"]}'
-P "AND('Org1MSP.member')"
```

Policy Syntax: **EXPR(E[, E...])**

- Where EXPR is either **AND** or **OR** and E is either a principal or nested EXPR
  Principal Syntax: **MSP.ROLE**
- Supported roles are: **member** and **admin**
- Where MSP is the MSP ID, and ROLE is either "member" or "admin"

N-out-of-K policy specification also possible
(e.g 3 out of 5 peers in the channel must endorse)

Endorsement policy examples

Examples of policies:

- Request 1 signature from all three principals
  - **AND('Org1.member', 'Org2. member', 'Org3.member')**
- Request 1 signature from either one of the two principals
  - **OR('Org1.member', 'Org2.member')**
- Request either one signature from a member of the Org1 MSP or (1 signature from a member of the Org2 MSP and 1 signature from a member of the Org3 MSP)
  - **OR('Org1.member', AND('Org2. member', 'Org3.member'))**

Bitcoin v/s Ethereum v/s Hyperledger

Blockchain characteristics comparison			
Characteristics	Bitcoin	Ethereum	Hyperledger
Permission restrictions	Permissionless	Permissionless	Permissioned
Restricted public access to data	Public	Public or private	Private
Consensus	Proof-of-Work	Proof-of-Work	PBFT
Scalability	High node-scalability, Low performance-scalability	High node-scalability, Low performance-scalability	Low node-scalability, High performance-scalability
Centralized regulation (governance*)	Low, decentralized decision making by community/miners	Medium, core developer group, but EIP process	Low, open-governance model based on Linux model
Anonymity	Pseudonymity, no encryption of transaction data	Pseudonymity, no encryption of transaction data	Pseudonymity, encryption of transaction data
Native currency	Yes, bitcoin, high value	Yes, ether	No
Scripting	Limited possibility, stack-based scripting	High possibility, Turing-complete virtual machine, high-level language support (Solidity)	High possibility, Turing-complete scripting of chaincode, high-level Go-language

Features	Hyperledger	Ethereum
Purpose	Preferred platform for B2B businesses	Platform for B2C businesses and generalized applications
Confidentiality	Confidential transactions	Transparent
Mode of Peer Participation	Private and Permissioned Network	Public/Private and Permissionless Network
Consensus Mechanism	Pluggable Consensus Algorithm: No mining required	PoW Algorithm: Consensus is reached by mining
Programming Language	Chaincode written in Golang	Smart Contracts written in Solidity

USE CASES

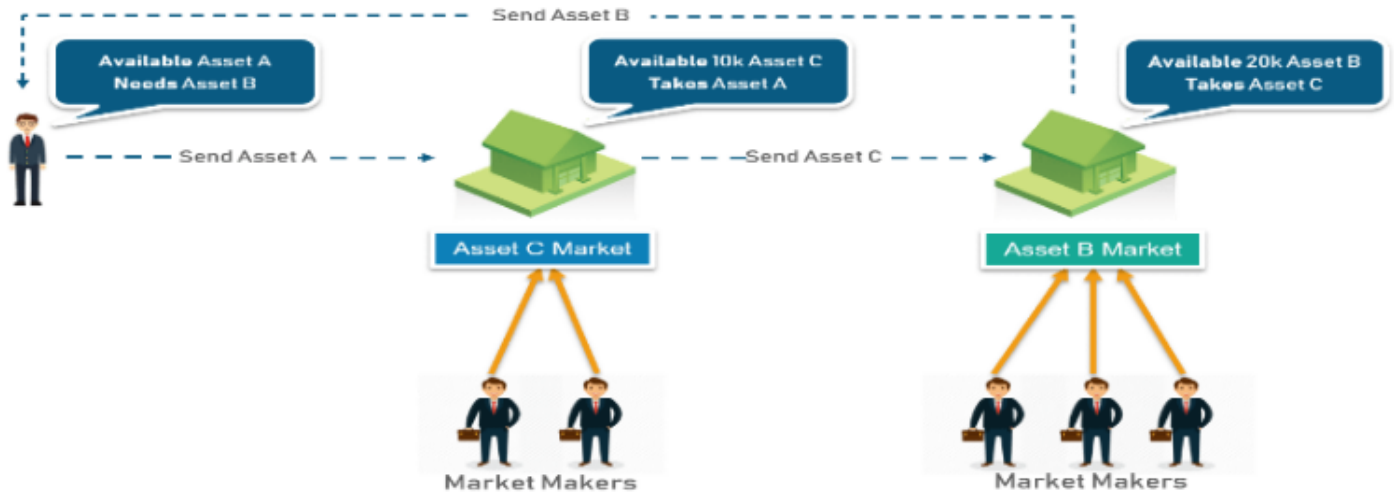
Use Case : Interoperability of Assets

Description: Interoperability of assets means the exchange of assets among a group of people.

Problem statement: If an organization requires 20,000 units of asset B but instead owns 10,000 units of asset A, it needs a way to exchange asset A for asset B. Though the current market might not offer enough liquidity to fulfill this trade quickly, there might be plenty of liquidity available between asset A and asset C, and also between asset C and asset B.

Now there are market limits on direct trading between A & B, so what can be the probable solution?

Solution: In this case, a chain network connects buyers with “buried” sellers,finds the best match (which could be buried under several layers of assets),and executes the transaction. So basically a business network of a group of individuals can be set up on the Hyperledger Fabric and the assets can be exchanged among the buyer and the sellers.



Financial Services	Public Sector	Retail	Insurance	Supply Chain & Logistics
• Trade Finance • Cross currency payments • Mortgages • KYC • Cross border tax	• Asset Registration • Citizen Identity • Medical records • Medicine supply chain	• Supply chain • Loyalty programs • Information sharing (supplier – retailer)	• Claims processing • Risk provenance • Asset usage history • Claims file	• Supply chain finance • Maintenance tracking • Provenance • Supply chain compliance

What makes a good blockchain use case?

1. A **business problem** to be solved
    - That cannot be more efficiently solved with other technologies
2. An identifiable **business network**
    - With Participants, Assets and Transactions
3. A need for **trust**
    - Consensus, Immutability, Finality or Provenance