PROOF-OF-WORK(POW)

- Proof of work (PoW) is a decentralized consensus mechanism that requires network members to expend effort in solving an encryption puzzle.
- Proof of work allows for secure peer-to-peer transaction processing without needing a trusted third party.
- Proof of work is a concept used in some public blockchains to demonstrate that a program did the work required to propose a new block for the chain.
- Proof of work is provided by sending the information in a block through a hashing algorithm, then adjusting variable fields until a hexadecimal number is reached that has a lower value than the network's difficulty target.

- The Proof of Work consensus algorithm involves solving a computationally challenging puzzle in order to create new blocks in the Bitcoin blockchain. The process is known as ‘mining’, and the nodes in the network that engages in mining are known as ‘miners’.
- *The following equation sums up the PoW requirement in bitcoin:*

$$H (N \| P\_hash \| Tx \| Tx \| \ldots Tx) < Target$$

Where N is a nonce, P_hash is a hash of the previous block, Tx represents transactions in the block, and Target is the target network difficulty value. This means that the hash of the previously mentioned concatenated fields should be less than the target hash value.

- *The only way to find this nonce is the brute force method. Once a certain pattern of a certain number of zeroes is met by a miner, the block is immediately broadcasted and accepted by other miners.*

- First, the worker, which is called a miner, creates a temporary file (a block). If it wins the competition to solve for a winning hash, this file will be stored on the blockchain. The block has the four following fields:
- Block size
- Block header
- Transaction counter
- Transactions

- The block header contains the following fields:

a) Software version

b) Previous block's hash

c) Merkle root

d) Timestamp

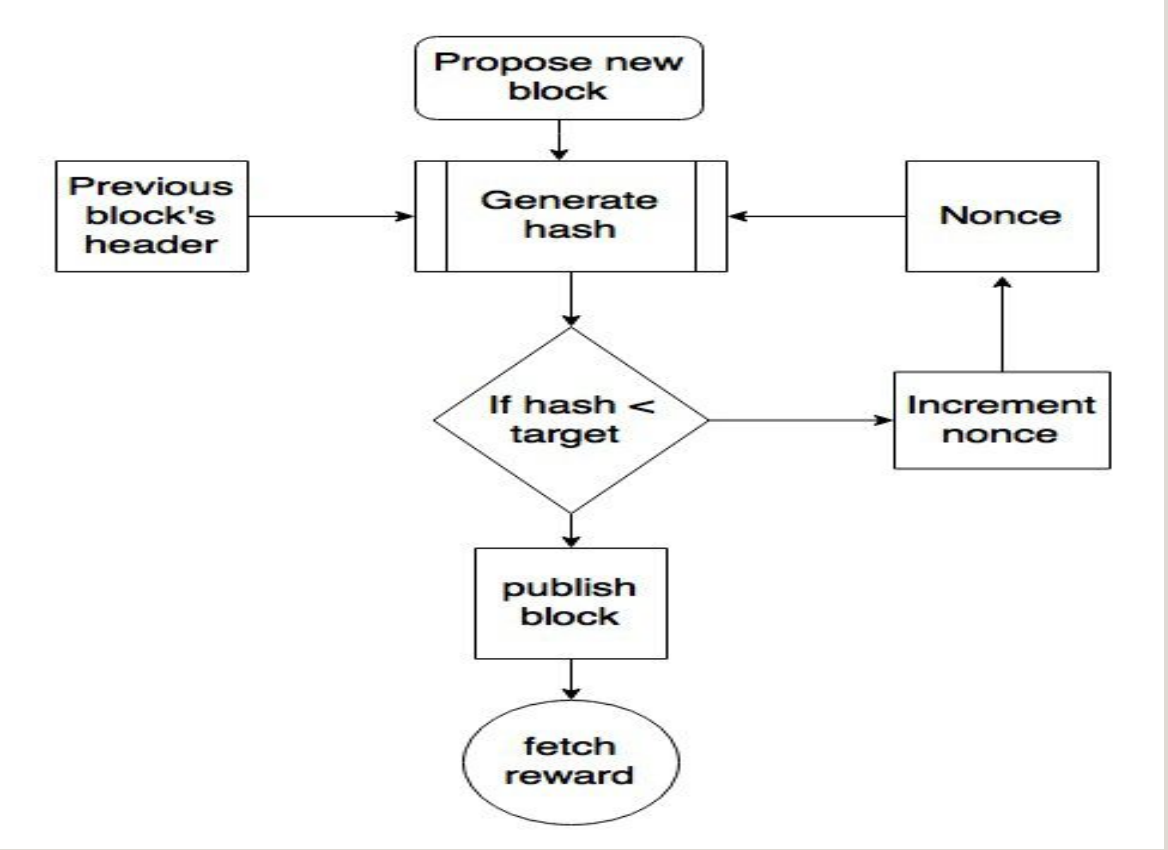
e) Difficulty target

f) Nonce

- The mining program assembles this block and places the transactions it has prioritized in the transaction field. It continuously adjusts the nonce and the extra nonce (which is part of the coinbase transaction in the Merkle tree) and sends the information in the block through a hashing algorithm.
- It repeats this process until it finds a solution, which is a value less than or equal to the difficulty target. The difficulty target is set so that a certain number of hashes per second must be attempted before a solution is found.

THE MINING ALGORITHM

- 1. The previous block's header is retrieved from the bitcoin network.
- 2. Assemble a set of transactions broadcasted on the network into a block to be proposed.
- 3. Compute the double hash of the previous block's header combined with a nonce and the newly proposed block using the SHA-256 algorithm.
- 4. Check if the resultant hash is lower than the current difficulty level (target) then PoW is solved. As a result of successful PoW the discovered block is broadcasted to the network and miners fetch the reward.
- 5. If the resultant hash is not less than the current difficulty level (target), then repeat the process after incrementing the nonce.


Propose new block
Previous block's header
Generate hash
Nonce
If hash < target
Increment nonce
publish block
fetch reward

- All successfully mined blocks must contain a hash that is less than this target number. This number is updated every 2 weeks or 2016 blocks to ensure that on average 10-minute block generation time is maintained.
- The hashing rate basically represents the rate of calculating hashes per second. In other words, this is the speed at which miners in the Bitcoin network are calculating hashes to find a block.

MINING SYSTEMS

- Over time, bitcoin miners have used various methods to mine bitcoins.
- Different types of mining methods,

  a) CPU Mining

  b) GPU

  c) FPGA

  d) ASIC

- ***CPU mining*** *was the first type of mining available in the original bitcoin client. Users could even use laptop or desktop computers to mine bitcoins.*
- *Due to the increased difficulty of the bitcoin network and the general tendency of finding faster methods to mine, miners started to use* ***GPUs or graphics cards*** *available in PCs to perform mining. GPUs support faster and parallelized calculations that are usually programmed using the OpenCL language.*
- Even GPU mining did not last long, and soon miners found another way to perform mining using FPGAs. **Field Programmable Gate Array** (FPGA) is basically an integrated circuit that can be programmed to perform specific operations. FPGAs are usually programmed in Hardware Description Languages (HDLs), such as Verilog and VHDL.

- ***Application Specific Integrated Circuit*** *(ASIC) was designed to perform the SHA-256 operation.*

CPU

GPU

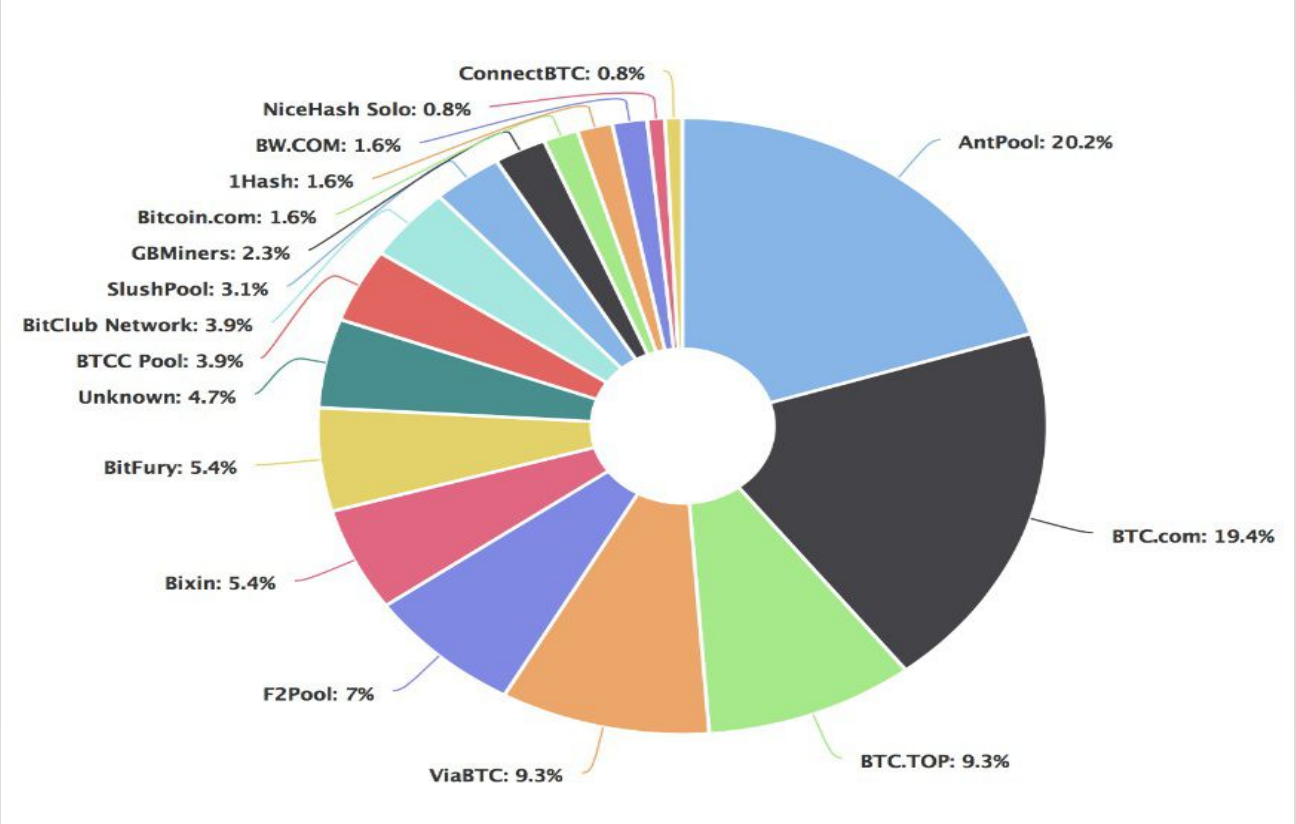
FPGA

ASIC

MINING POOLS

- *A mining pool forms when group of miners work together to mine a block. The pool manager receives the coinbase transaction if the block is successfully mined, which is then responsible for distributing the reward to the group of miners who invested resources to mine the block.*
- There are various models that a mining pool manager can use to pay to the miners, such as the Pay Per Share (PPS) model and the proportional model.
- *In the PPS model, the mining pool manager pays a flat fee to all miners who participated in the mining exercise, whereas in the proportional model, the share is calculated based on the amount of computing resources spent to solve the hash puzzle*

MINING POOLS



51% ATTACK

- A 51% attack is a potential threat to blockchain networks, where a group of miners may control more than 50% of the network's mining hash rate.
- This control may allow the attackers to prevent new transactions from gaining confirmations, halt payments, and even reverse transactions.
- Attackers with majority network control can interrupt the recording of new blocks by preventing other miners from completing blocks.

RISKS AND CONSEQUENCES OF A 51% ATTACK

- **Double-Spending**: This is the most feared consequence. The attacker could spend their money twice — first, they perform a regular transaction and then change the blockchain to show they never used the money at all.
- **Transaction Reversal**: The attacker can block payments between some or all users. This disrupts the normal operation of the network and can lead to significant delays in transaction confirmations, undermining confidence in the network's reliability.
- **Denial-of-Service (DoS) Attack**: The hacker takes over and blocks the addresses of other miners for a while. This stops the good guys – the honest miners – from getting back control of the network. As a result, the attacker's false chain of transactions can become permanent.

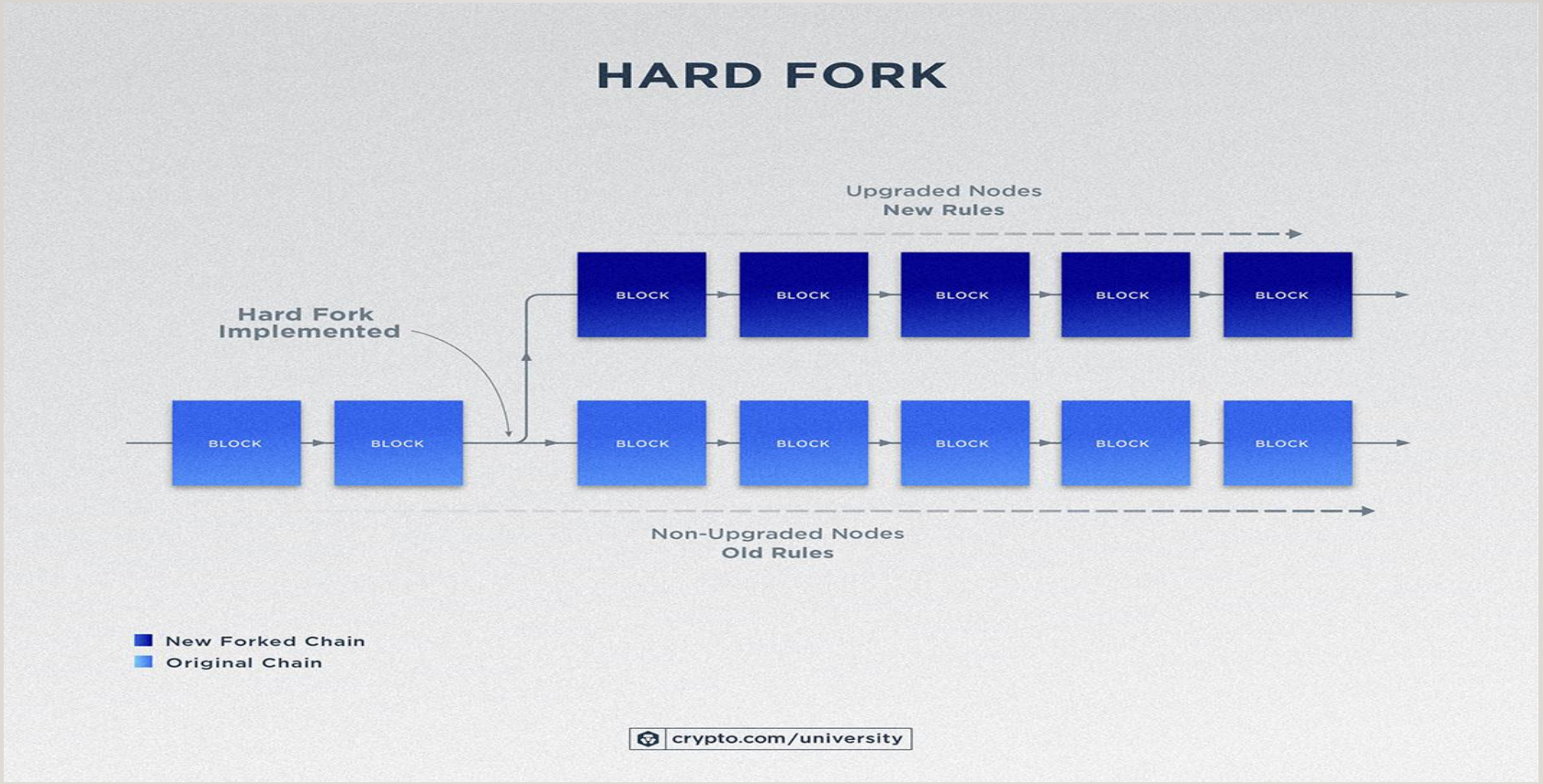
BLOCKCHAIN FORK

- A fork is a change to the blockchain's underlying protocol.
- A blockchain fork is an important upgrade to the network and can either represent a radical change or a minor one and can be initiated by developers or community members.
- A hard fork is a radical change in a cryptocurrency protocol that is incompatible with the previous blockchain versions.
- A soft fork is a change in a cryptocurrency protocol that keeps it backward compatible.

- Since cryptocurrencies are decentralised networks, all participants in a network — known as nodes — need to follow the same rules in order to properly work together. That set of rules is known as a ‘protocol’.
- Typical rules in a protocol include the size of a block on a blockchain, the rewards miners receive for mining a new block, and many more.
- The decentralised nature of blockchains means that nodes on the network must be able to come to an agreement (i.e., consensus) as to the shared state of the blockchain.
- a fork is more often used to implement a fundamental change or create a new asset with similar (but not equal) characteristics as the original.

HARD FORKS

- A hard fork is a radical change in a cryptocurrency protocol that is incompatible with the previous versions. This means that nodes with the older protocol (pre-fork) aren't able to process transactions or push new blocks to the post-fork (newer) blockchain.
- All nodes and miners have to upgrade to the latest version of the protocol if they want to be on the new forked chain.
-


HARD FORK
Upgraded Nodes
New Rules
Hard Fork
Implemented
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
Non-Upgraded Nodes
Old Rules
New Forked Chain
Original Chain
crypto.com/university

- A hard fork in blockchain is a permanent split in a blockchain that creates two separate blockchains, each operating independently. This happens when a blockchain's programming is changed in a way that's incompatible with the previous version.
- Backward incompatibility

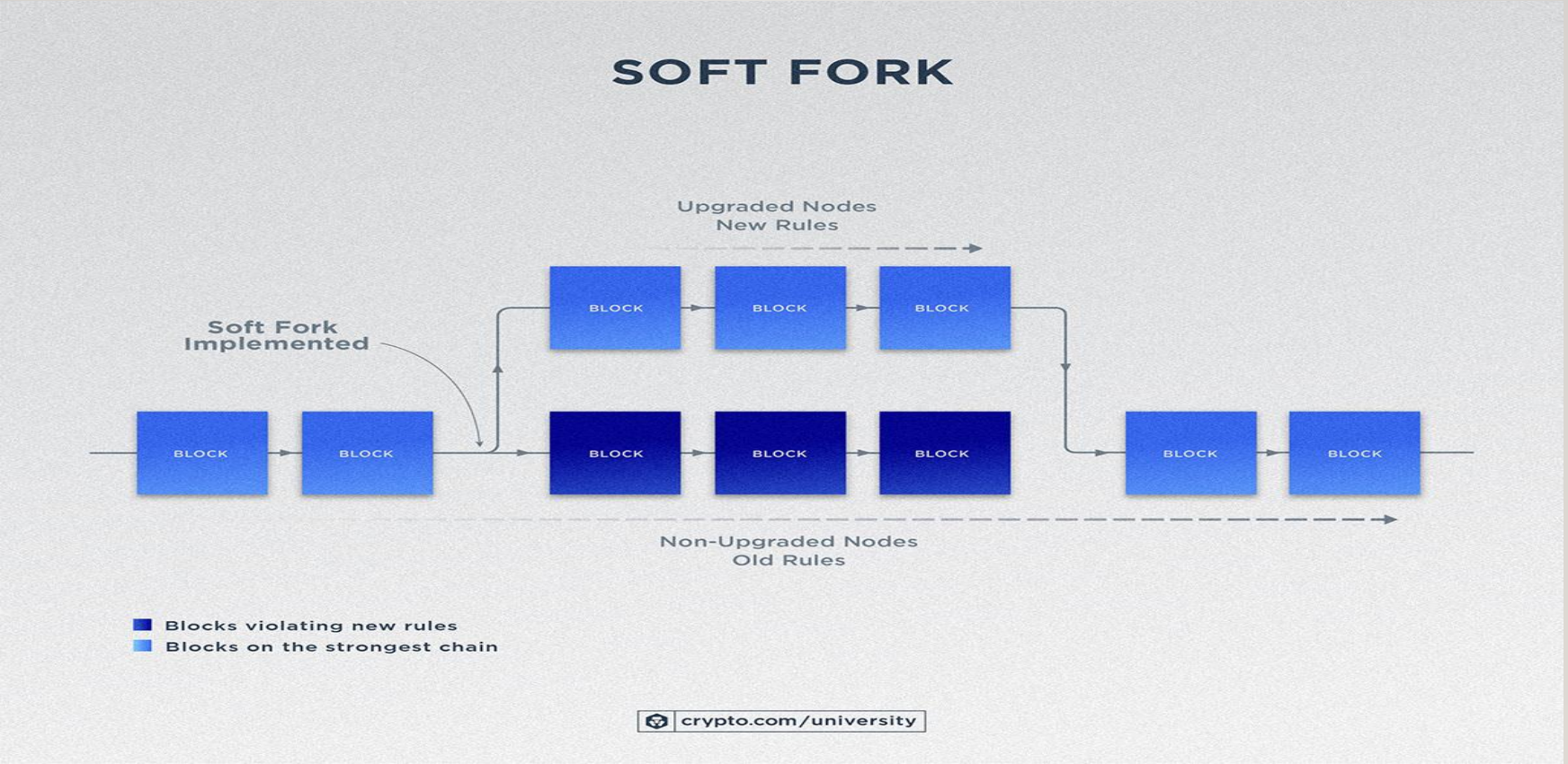
  Hard forks are not backward compatible, meaning that users need to upgrade their software to continue participating in the network.
- Splitting the community

  Hard forks often split the community into two factions
- Coexistence

  It's possible that the two blockchains can run parallel to each other indefinitely.

SOFT FORKS

- A soft fork is a change in a cryptocurrency protocol that keeps it backward compatible.
- Non-updated nodes are still able to process transactions and push new blocks to the blockchain, so long as they follow the new protocol rules.
- This kind of fork requires only a majority of the miners upgrading to adjust to the new rules, as opposed to a hard fork, which requires (almost) all nodes to upgrade and agree on the new version.


SOFT FORK
Upgraded Nodes
New Rules
Soft Fork
Implemented
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
BLOCK
Non-Upgraded Nodes
Old Rules
Blocks violating new rules
Blocks on the strongest chain
crypto.com/university

- A soft fork is a change to a blockchain's software protocol that's backward-compatible and doesn't split the network:
- Backward-compatible

  Soft forks maintain compatibility with the old rules, so nodes that don't update to the new protocol can still work with it.
- Majority of miners upgrade

  Only a majority of the network's miners need to upgrade to enforce the new rules.

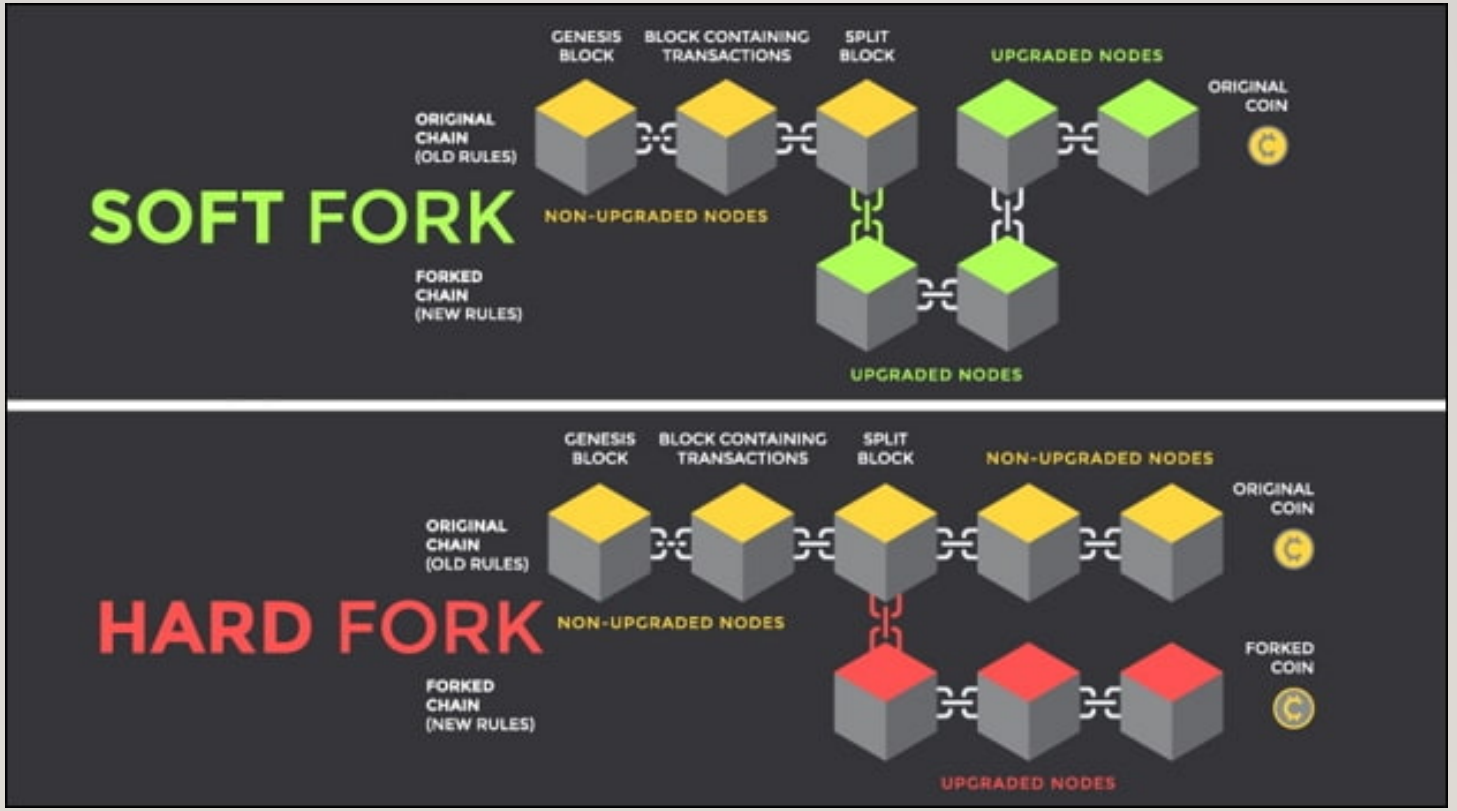
- No new blockchain

  Soft forks don't create a new blockchain, so the network can gradually transition to the new rules.

- Add new features

  Soft forks are often initiated by community members to add new features or functions.

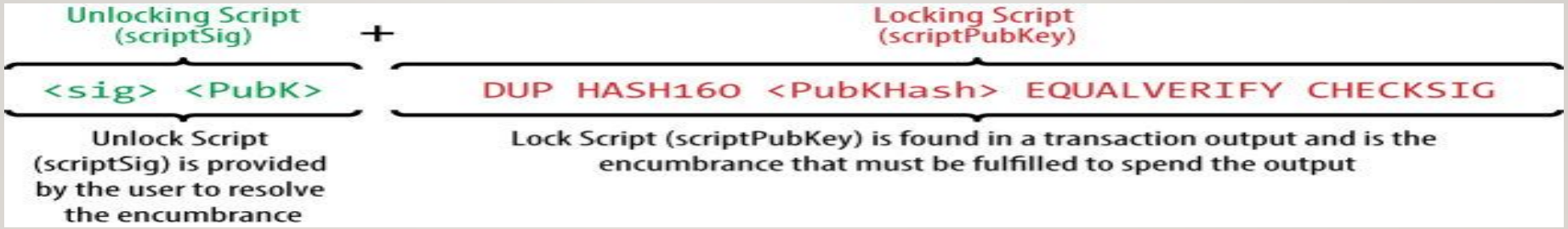
A well-known example of a soft fork is the Segregated Witness (SegWit) upgrade to Bitcoin in 2015. SegWit improved the network's capacity by allowing more transactions per block.


GENESIS BLOCK
BLOCK CONTAINING TRANSACTIONS
SPLIT BLOCK
UPGRADED NODES
ORIGINAL COIN
ORIGINAL CHAIN (OLD RULES)
SOFT FORK
NON-UPGRADED NODES
FORKED CHAIN (NEW RULES)
UPGRADED NODES
GENESIS BLOCK
BLOCK CONTAINING TRANSACTIONS
SPLIT BLOCK
NON-UPGRADED NODES
ORIGINAL COIN
ORIGINAL CHAIN (OLD RULES)
HARD FORK
NON-UPGRADED NODES
FORKED COIN
FORKED CHAIN (NEW RULES)
UPGRADED NODES

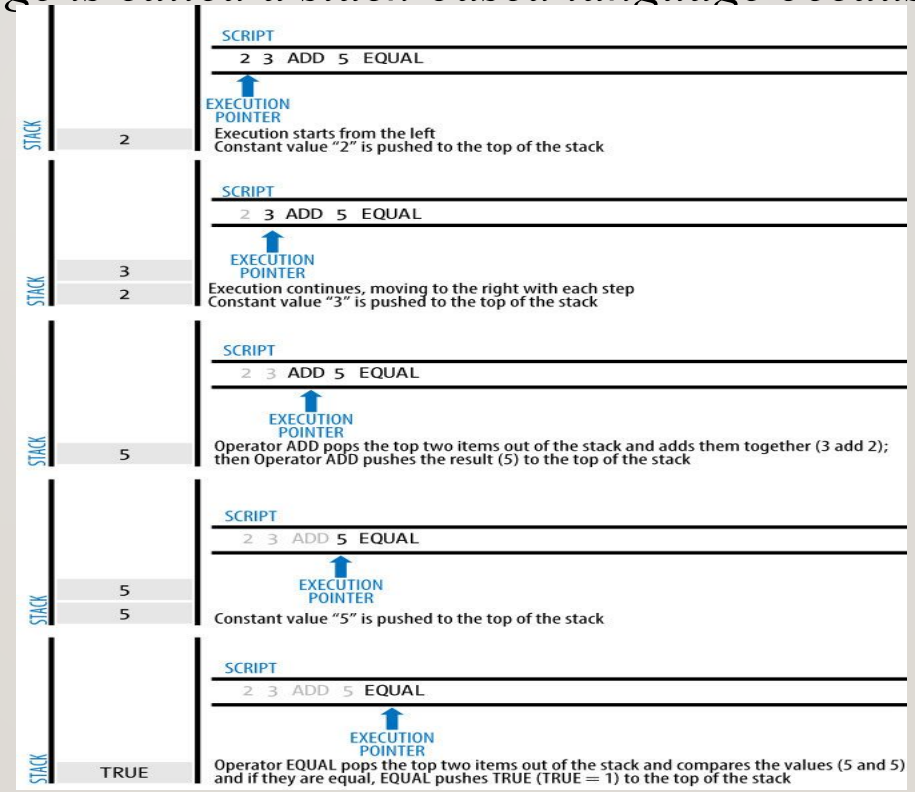
- Why do we need Forking in Blockchain?
  - Security updates
  - To prevent emerging cyberattacks on the current Blockchain.
  - To improve functionality by updating the software.

- *The bitcoin transaction script language, called Script, is a Forth-like reverse-polish notation stack-based execution language.*
- *Bitcoin's transaction validation engine relies on two types of scripts to validate transactions: a locking script and an unlocking script.*
- *A locking script is a spending condition placed on an output: it specifies the conditions that must be met to spend the output in the future.*
- *An unlocking script is a script that "solves," or satisfies, the conditions placed on an output by a locking script and allows the output to be spent.*

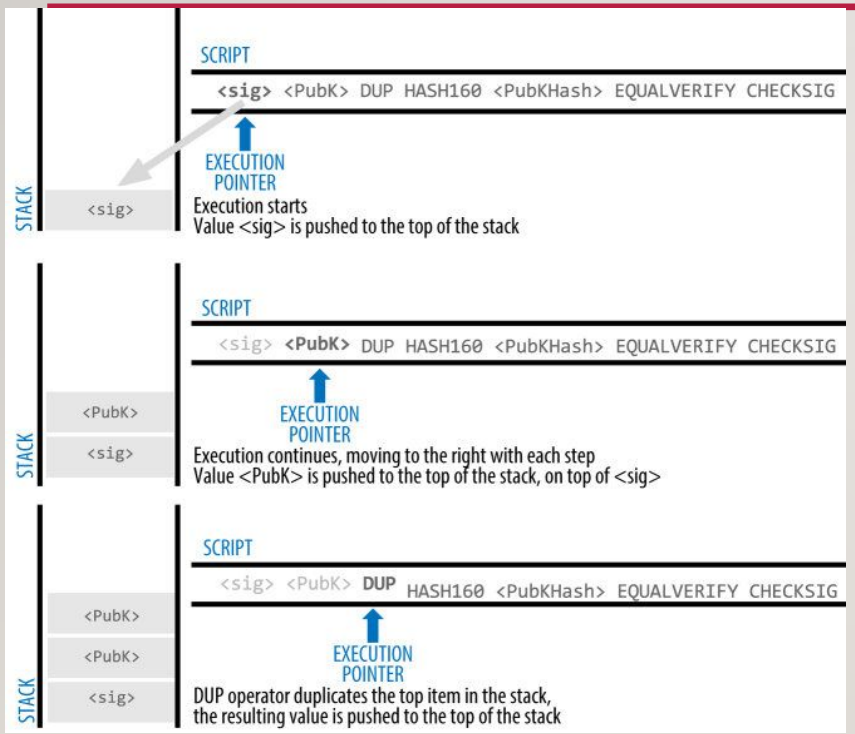
- *Every bitcoin validating node will validate transactions by executing the locking and unlocking scripts together.*

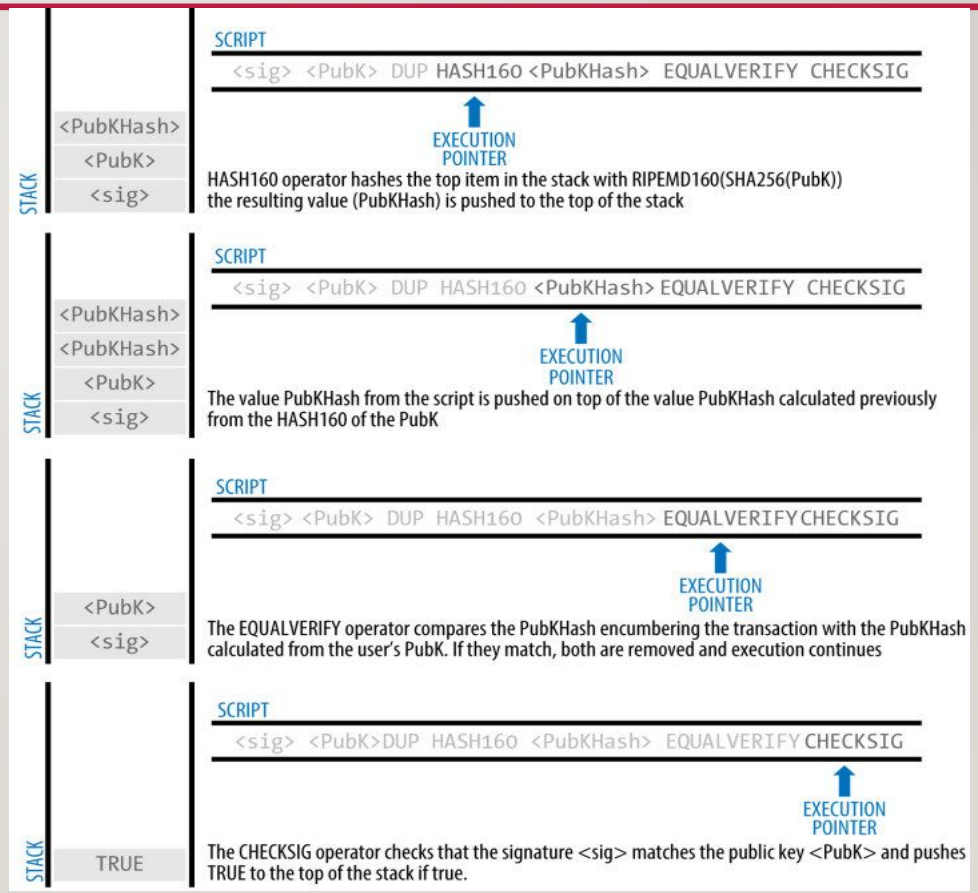


- *Bitcoin's scripting language is called a stack-based language because it uses a data structure called a stack.*



- *The vast majority of transactions processed on the bitcoin network spend outputs locked with a Pay-to-Public-Key-Hash or "P2PKH" script.*
- *These outputs contain a locking script that locks the output to a public key hash, more commonly known as a bitcoin address.*
- *An output locked by a P2PKH script can be unlocked (spent) by presenting a public key and a digital signature created by the corresponding private key.*


SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<sig>
Execution starts
Value <sig> is pushed to the top of the stack
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>
Execution continues, moving to the right with each step
Value <PubK> is pushed to the top of the stack, on top of <sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<PubK>
<sig>
DUP operator duplicates the top item in the stack,
the resulting value is pushed to the top of the stack


SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubKHash>
<PubK>
<sig>
HASH160 operator hashes the top item in the stack with RIPEMD160(SHA256(PubK))
the resulting value (PubKHash) is pushed to the top of the stack
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubKHash>
<PubKHash>
<PubK>
<sig>
The value PubKHash from the script is pushed on top of the value PubKHash calculated previously
from the HASH160 of the PubK
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>
The EQUALVERIFY operator compares the PubKHash encumbering the transaction with the PubKHash
calculated from the user's PubK. If they match, both are removed and execution continues
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
TRUE
The CHECKSIG operator checks that the signature <sig> matches the public key <PubK> and pushes
TRUE to the top of the stack if true.

- *A digital signature serves three purposes in bitcoin (see the following sidebar). First, the signature proves that the owner of the private key, who is by implication the owner of the funds, has authorized the spending of those funds.*
- *Secondly, the proof of authorization is undeniable (nonrepudiation).*
- *Thirdly, the signature proves that the transaction (or specific parts of the transaction) have not and cannot be modified by anyone after it has been signed*

- *Pay-to-Public-Key-Hash (P2PKH)*
- *These outputs contain a locking script that locks the output to a public key hash, more commonly known as a bitcoin address. An output locked by a P2PKH script can be unlocked (spent) by presenting a public key and a digital signature created by the corresponding private key.*
- *For example, let's look at Alice's payment to Bob's Cafe again. Alice made a payment of 0.015 bitcoin to the cafe's bitcoin address. That transaction output would have a locking script of the form:*
- `OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG`

- *The `Cafe Public Key Hash` is equivalent to the bitcoin address of the cafe, without the Base58Check encoding.*
- *The preceding locking script can be satisfied with an unlocking script of the form:*
- `<Cafe Signature> <Cafe Public Key>`
- *The two scripts together would form the following combined validation script:*
- `<Cafe Signature> <Cafe Public Key> OP_DUP OP_HASH160`
- `<Cafe Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG`

- *The digital signature algorithm used in bitcoin is the Elliptic Curve Digital Signature Algorithm, or ECDSA. ECDSA is the algorithm used for digital signatures based on elliptic curve private/public key pairs.*
- *ECDSA is used by the script functions* `OP_CHECKSIG`, `OP_CHECKSIGVERIFY`, `OP_CHECKMULTISIG`, *and* `OP_CHECKMULTISIGVERIFY`.

ETHEREUM

- Ethereum is a blockchain-based computing platform that enables developers to build and deploy decentralized applications.
- Ethereum is a decentralised blockchain platform that establishes a peer-to-peer network that securely executes and verifies application code, called smart contracts.
- Ethereum offers an extremely flexible platform on which to build decentralized applications using the native Solidity scripting language and Ethereum Virtual Machine.
- Ethereum applications, with wallets like MetaMask, Argent, Rainbow and more offering simple interfaces through which to interact with the Ethereum blockchain and smart contracts deployed there.

	Ethereum	Hyperledger Fabric
Public vs. Private	Public	Private
Permissions	Permissionless	Permissioned
Governance	Decentralized	Federated
Consenus Mechanism	Proof-of-Work	Pluggable BFT
Smart Contract Languages	Solidity, Vyper	Go, Java, Javascript (Node.js)
Private Transactions	No	Yes
Ideal Use Cases	Tokenization (stablecoins, NFTs), DeFi, public transaction settlement	B2B data exchange, transaction settlement, and non-repudiation

ETHEREUM FEATURES

- Ether: This is Ethereum's cryptocurrency.
- Smart contracts: Ethereum allows the development and deployment of these types of contracts.
- Ethereum Virtual Machine: Ethereum provides the underlying technology—the architecture and the software—that understands smart contracts and allows you to interact with it.
- Decentralized applications (Dapps): A decentralized application is called a Dapp for short. Ethereum allows you to create consolidated applications, called decentralized applications.
- Decentralized autonomous organizations (DAOs): Ethereum allows you to create these for democratic decision-making.

ETHER

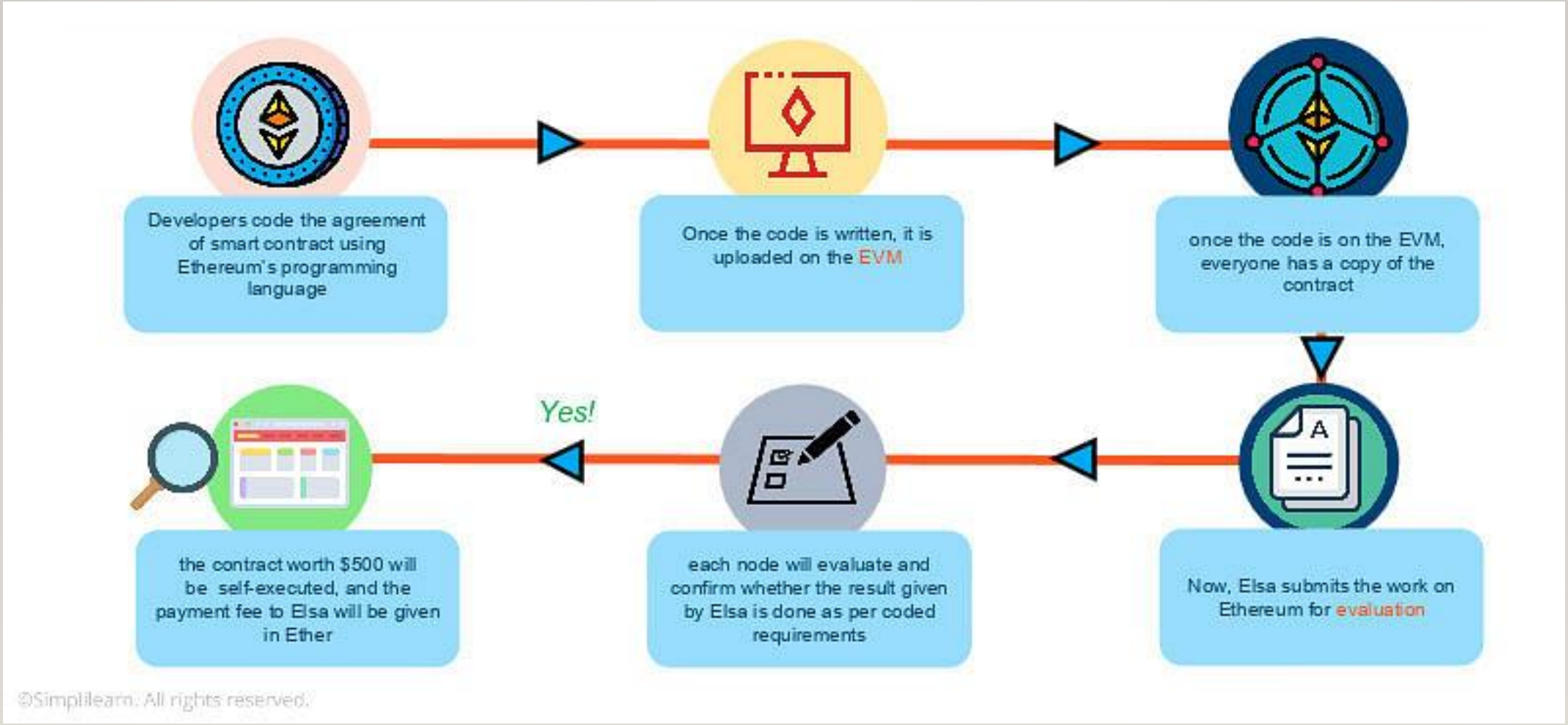
- It is the fuel that runs the network. It is used to pay for the computational resources and the transaction fees for any transaction executed on the Ethereum network.
- Ether can be utilized for building decentralized applications, building smart contracts, and making regular peer-to-peer payments.
- Gas is the execution fee paid by a user for running a transaction in Ethereum.

SMART CONTRACT

- A smart contract is a simple computer program that facilitates the exchange of any asset between two parties. It could be money, shares, property, or any other digital asset that you want to exchange.
- Anyone on the Ethereum network can create these contracts. The contract consists primarily of the terms and conditions mutually agreed on between the parties (peers).
- The smart contract's primary feature is that once it is executed, it cannot be altered, and any transaction done on top of a smart contract is registered permanently—it is immutable.

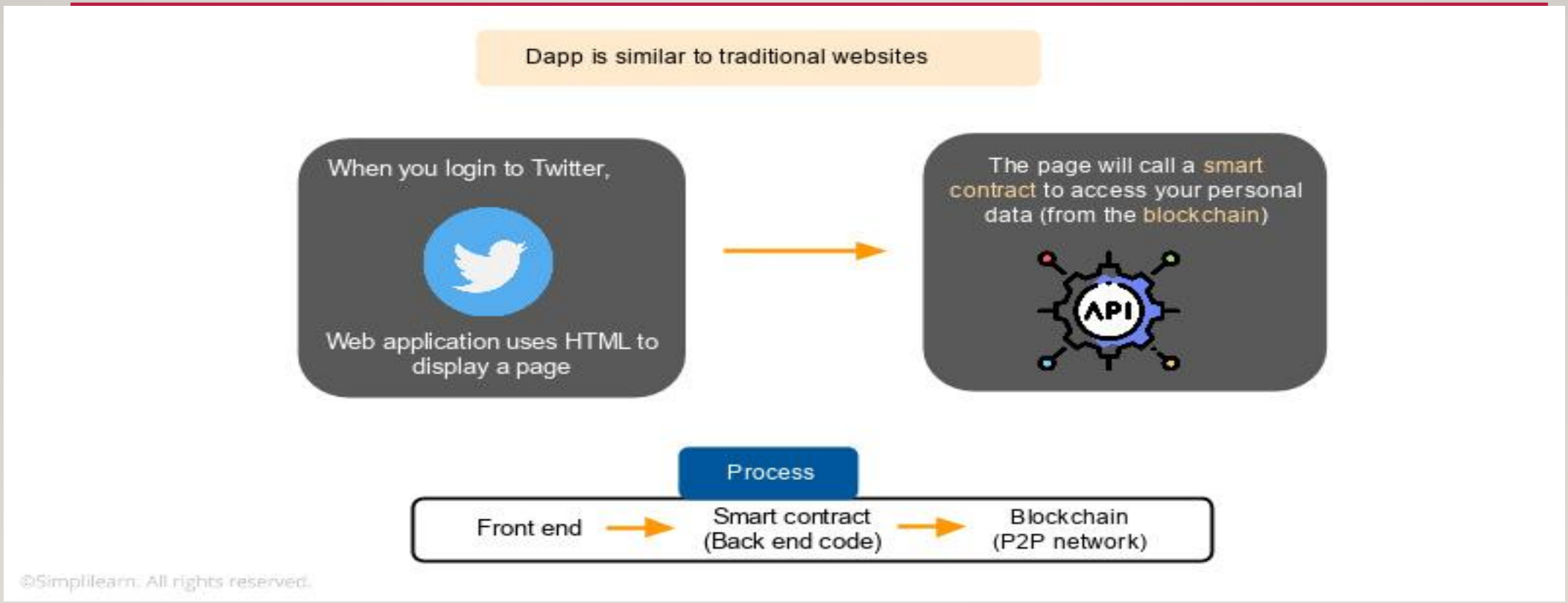
ETHEREUM VIRTUAL MACHINE

- EVM is designed to operate as a runtime environment for compiling and deploying Ethereum-based smart contracts.
- EVM is the engine that understands the language of smart contracts, which are written in the Solidity language for Ethereum.
- Zack has given a contract of $500 to someone named Elsa for developing his company's website. The developers code the agreement of the smart contract using Ethereum's programming language.



	Bitcoin	Ethereum
Hashing Algorithm	SHA-256	Ethash
Time is taken to mine a block	An average of 10 minutes	An average of 12-15 seconds
Reward	12.5 BTC	3 ETH
USD - 18/09/2024	1 Bitcoin = **60,582.60**	1 Ether = **2,357.76**

DECENTRALIZED APPLICATIONS (DAPPS)



DECENTRALIZED AUTONOMOUS ORGANIZATIONS (DAOS)

- A DAO is a digital organization that operates without hierarchical management; it works in a decentralized and democratic fashion.
- DAO is an organization in which the decision-making is not in the hands of a centralized authority but preferably in the hands of certain designated authorities or a group or designated people as a part of an authority.

ETHEREUM NODES

- To maintain the accuracy and independence of its blockchain and allow open access to all users, Ethereum employs an open network with three different participants, known as Nodes.
- A Node is a computer running athe software application (called client) that allows Ethereum to function, described as clients.
- All nodes are connected to each other in a network that coordinates voluntarily to verify transactions and maintain the true state of the shared blockchain.

- **Full Nodes**
  - Maintain and store a copy of the Ethereum blockchain
  - Periodically pruned to manage the size of the blockchain
  - Serve data on request
  - Can verify blocks and states
  - Can validate new blocks

- **Lightweight Nodes**

  Maintain and store a limited copy of the Ethereum blockchain with only the minimum amount of data required to transact

- Archive Nodes

  A historical archive of all state changes; aren't required to validate blocks

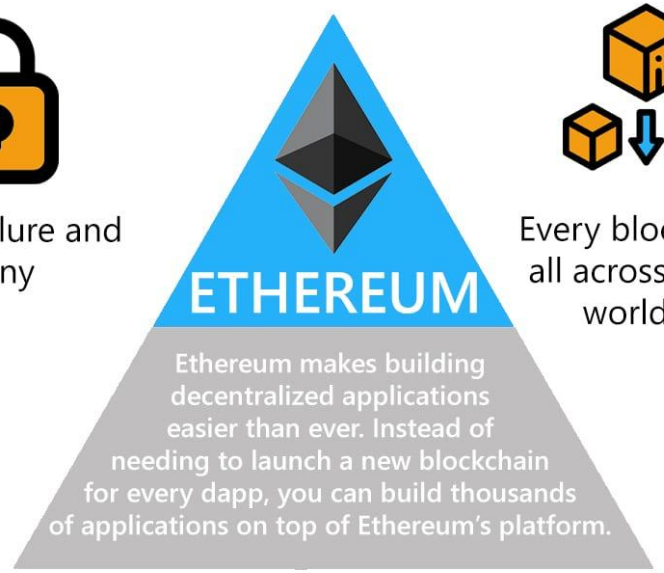
- Full Nodes run two pieces of client software: one to execute the smart contract instructions or new transactions in the EVM; the other is the consensus software to ensure that the changes are valid.
- Ethereum's process to agree on the validity of information recorded in new blocks is called Proof-of-Stake (PoS).
- In PoS, Full Nodes acting as validators are chosen to create new blocks and validate transactions based on the amount of ETH they stake.

- Validators must stake at least 32 ETH, but the more staked above that threshold, the more likely a validator will be chosen to propose a new block.
- The alternative is to contribute less than 32 ETH to a staking pool.
- Staking pools are run by a single validator with the power to propose a new block. The reward you earn will be proportionate to your contribution to the pool.
- In Ethereum's PoS consensus system, validators are randomly chosen to propose new blocks and validate transactions in slots every 12 seconds.

Decentralized Networks

- Immutable
- Tamper Proof
- Secure

With no central point of failure and security by cryptography, any applications are protected against fraud and attacks.



Blockchains

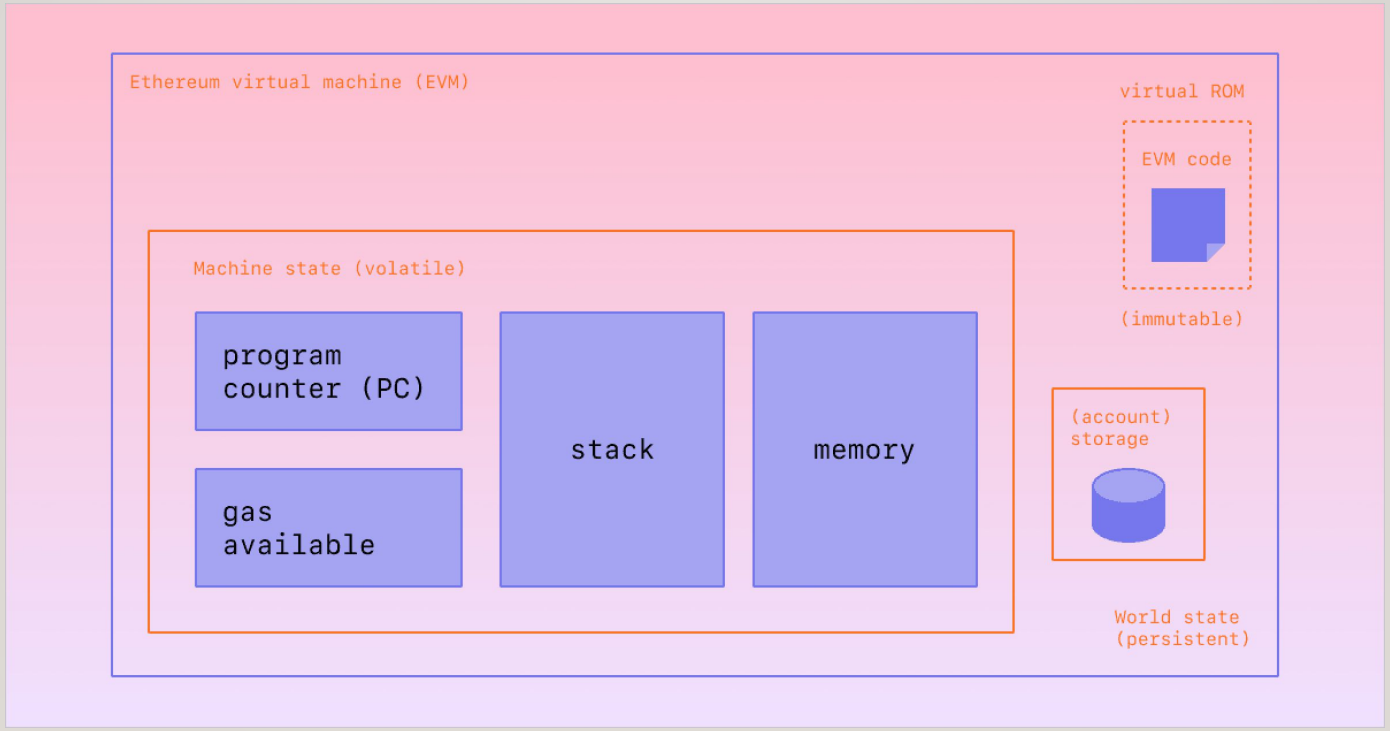
- Trustless
- Global
- Permanent

Every block of information is stored all across the network, leading to a world-wide environment where everyone is in the know.

Blockgeeks

SOLIDITY

EVM


Ethereum virtual machine (EVM)
virtual ROM
EVM code
(immutable)
Machine state (volatile)
program counter (PC)
gas available
stack
memory
(account) storage
World state (persistent)

INITIALIZATION OF STATE VARIABLE

```
// Solidity program to initialize
// state variable at the time of declaration
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.9.0;

contract example1 {

  string public name = ”arunkumar";
  uint public age = 35;
}
```

```
// Solidity program to initialize
// state variables using constructors
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.9.0;

contract example1 {
    string public name;
    uint public age;

     Constructor() public
    {
     name = ”arunkumar";
     age = 35;
    }
}
```

```
// SPDX-License-Identifier: MIT
pragma solidity >= 0.5.0 < 0.9.0;
contract example {
    string public name;
    uint public age;
      function setName() public
    {
      name = ”arunkumar";
    }
      function setAge() public
    {
      age = 35;
    }
}
```

SOLIDITY – VIEW AND PURE FUNCTIONS

- The **view functions** are read-only functions, which ensures that state variables cannot be modified after calling them.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;
contract Test {
    // Declaring state variables
    uint num1 = 2;
    uint num2 = 4;

   function getResult( ) public view returns( uint product, uint sum){
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```

THE PURE FUNCTIONS DO NOT READ OR MODIFY THE STATE VARIABLES, WHICH RETURNS THE VALUES ONLY USING THE PARAMETERS PASSED TO THE FUNCTION OR LOCAL VARIABLES PRESENT IN IT.

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.5.0;
contract Test {
   function getResult( ) public pure returns(uint product, uint sum){
      uint num1 = 2;
      uint num2 = 4;
      product = num1 * num2;
      sum = num1 + num2;
   }
}
```

MAPPINGS

- Mapping in Solidity acts like a hash table or dictionary in any other language.
- These are used to store the data in the form of key-value pairs, a key can be any of the built-in data types but reference types are not allowed while the value can be of any type.
- Mappings are mostly used to associate the unique Ethereum address with the associated value type.
- mapping(key => value) <access specifier> <name>;

SOLIDITY – ERROR HANDLING

- **1. `require`: Validate Inputs and Conditions**
  - The `require` function acts as a guard, validating inputs and conditions before executing specific parts of the code.
  - It ensures that the specified conditions are met, and if not, it throws an error and reverts all changes made to the contract's state.
  - This function is commonly used to validate inputs, check conditions before execution, and validate return values from other functions.
  - ``*solidity*
    require(_i > 10, "Input must be greater than 10");
    ```
    ```

- **2. `revert`: Complex Conditions and Custom Errors**
  - The `revert` function is similar to `require` in terms of reverting state changes and throwing an error.
  - However, `revert` is particularly useful when the conditions to check are more complex or when you want to provide more detailed error messages.
  - ``solidity
    if (_i <= 10) {
    revert(“Input must be greater than 10”);
    }

- **3. `assert`: Testing Invariants and Internal Errors**
  - The `assert` function is used to check for conditions that should never be false.
  - It is primarily used to test for internal errors and verify invariants within the contract. If an `assert` statement fails, it indicates the presence of a bug in the code.
  - ```*solidity*
    *assert(num == 0);*
    ```
    ```

- **Custom Error: Gas Optimization and Detailed Error Messages**
  - Solidity allows you to define custom errors using the `error` keyword. Custom errors help optimize gas costs by avoiding unnecessary storage operations and provide more informative error messages.
  - ```
    error InsufficientBalance(uint balance, uint withdrawAmount);
    function testCustomError(uint _withdrawAmount) public view {
    uint bal = address(this).balance;
    if (bal < _withdrawAmount) {
    revert InsufficientBalance({balance: bal, withdrawAmount:
    _withdrawAmount});
    }
    }
    ```
    ```

Require	Revert	Assert
Used at the beginning of a function Validates against illegal input Verifies state conditions prior to execution Refunds leftover gas	Identical to require Useful for more complex logic flow gates (i.e., complicated if-then blocks) Refunds leftover gas	Used at the end of a function Validates something that is impossible Critical for static code analysis tools Does not refund leftover gas

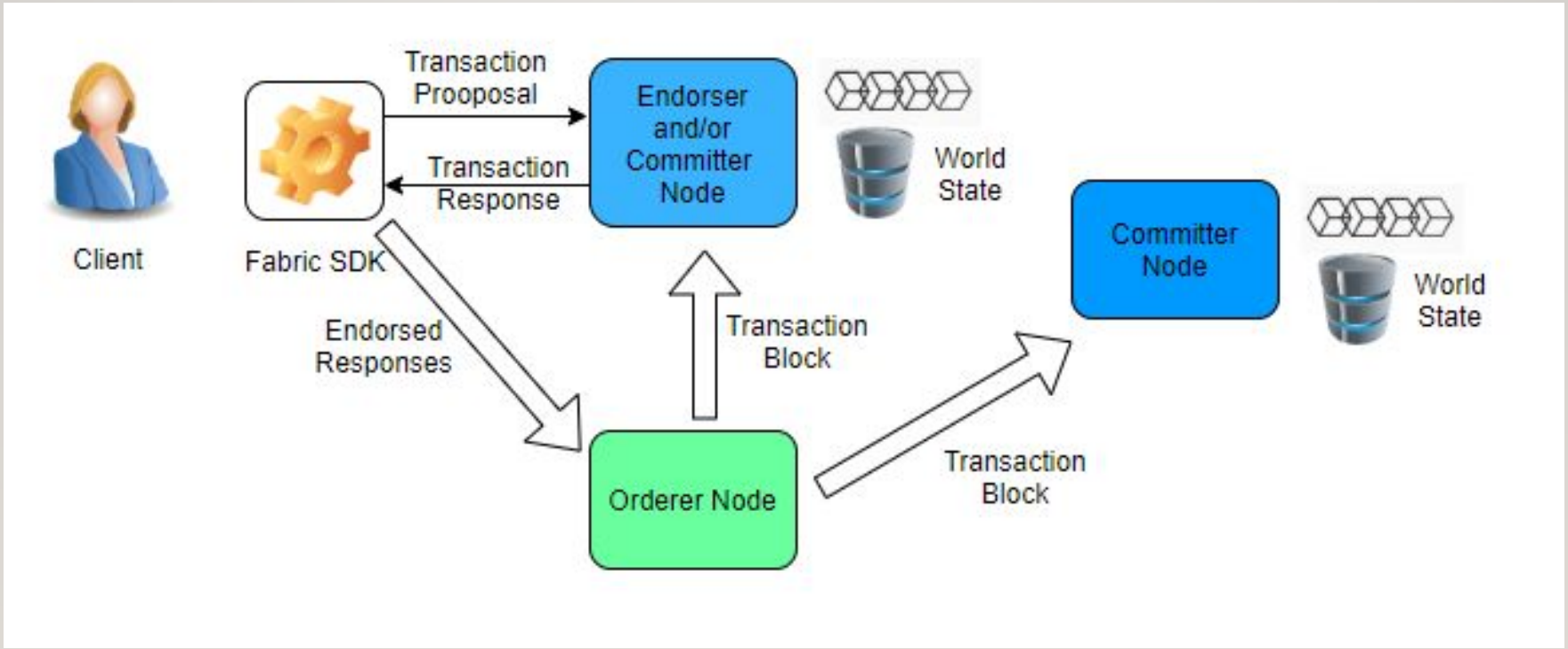
```
function transfer(address recipient, uint amount) public
{
    require(amount > 0, "Amount must be greater than 0");
    balances[msg.sender] -= amount;
    balances[recipient] += amount;
}
```

```
function withdraw(uint amount) public {
    assert(balances[msg.sender] >= amount);
    balances[msg.sender] -= amount;
    payable(msg.sender).transfer(amount);
}
```

```
function withdraw(uint amount) public {
    require(balances[msg.sender] >= amount, "Insufficient balance");
    balances[msg.sender] -= amount;
    if (!msg.sender.send(amount)) {
        revert("Failed to send funds");
    }
}
```

HYPERLEDGER FABRIC

- Hyperledger Fabric is a permissioned blockchain framework, with a modular architecture (plug-and-play).
- It leverages container technology to host smart contract (Chaincode) which contains the application logic.
  - **Modular Architecture**
  - **Permissioned Network**
  - **Channels**
  - **Smart Contracts (Chaincode)**


Client
Fabric SDK
Transaction Prooposal
Transaction Response
Endorser and/or Committer Node
World State
Endorsed Responses
Transaction Block
Orderer Node
Transaction Block
Committer Node
World State

- Fabric consist of below major components
  - Membership Service Provider(MSP)
  - Client
  - Peer
  - Orderer

MEMBERSHIP SERVICE PROVIDER (MSP)

- The membership service provider (MSP), is a component that defines the rules in which, identities are validated, authenticated, and allowed access to a network.
- The MSP manages user IDs and authenticates clients who want to join the network. This includes providing credentials for these clients to propose transactions. The MSP makes use of a *Certificate Authority*, which is a pluggable interface that verifies and revokes user certificates upon confirmed identity.
  - Fabric-CA
  - External-CA

CLIENTS

- Clients are applications that act on behalf of a person to propose transactions on the network.
- The client uses a Fabric SDK in order to communicate with the network. The client communicates with the SDK in order to Read or Write the data in a Fabric blockchain and an in-state DB.
- Even the client is issued with a certificate from the CA authority in order to make sure that a valid client has initiated the transaction over the network.

NODES

- A “Node” is only a logical function in the sense that multiple nodes of different types can run on the same physical server.
  - **Client**: A client that submits an actual transaction-invocation to the endorsers, and broadcasts transaction-proposals to ordering service. In short, clients communicate with both peers and the ordering service
  - **Peer**: A node that commits transactions and maintains the state and a copy of the ledger. A peer receives ordered state updates in the form of *blocks* from the ordering service and maintains the state and the ledger.
  - **Ordering-service-node or Orderer**: a node running the communication service that implements a delivery guarantee, such as atomic or total order broadcast.

- **Endorsing Peer**: Endorsing peers is a special type of committing peers who have an additional role to endorse a transaction.
- They endorse the transaction request which comes from the client. Each endorsing peer possesses the copy of smart contract installed and a ledger.
- The main function of Endorser is to simulate the transaction. It is executed based on the smart contract on the personal copy of the ledger, and generates the Read/Write sets which are sent to Client.

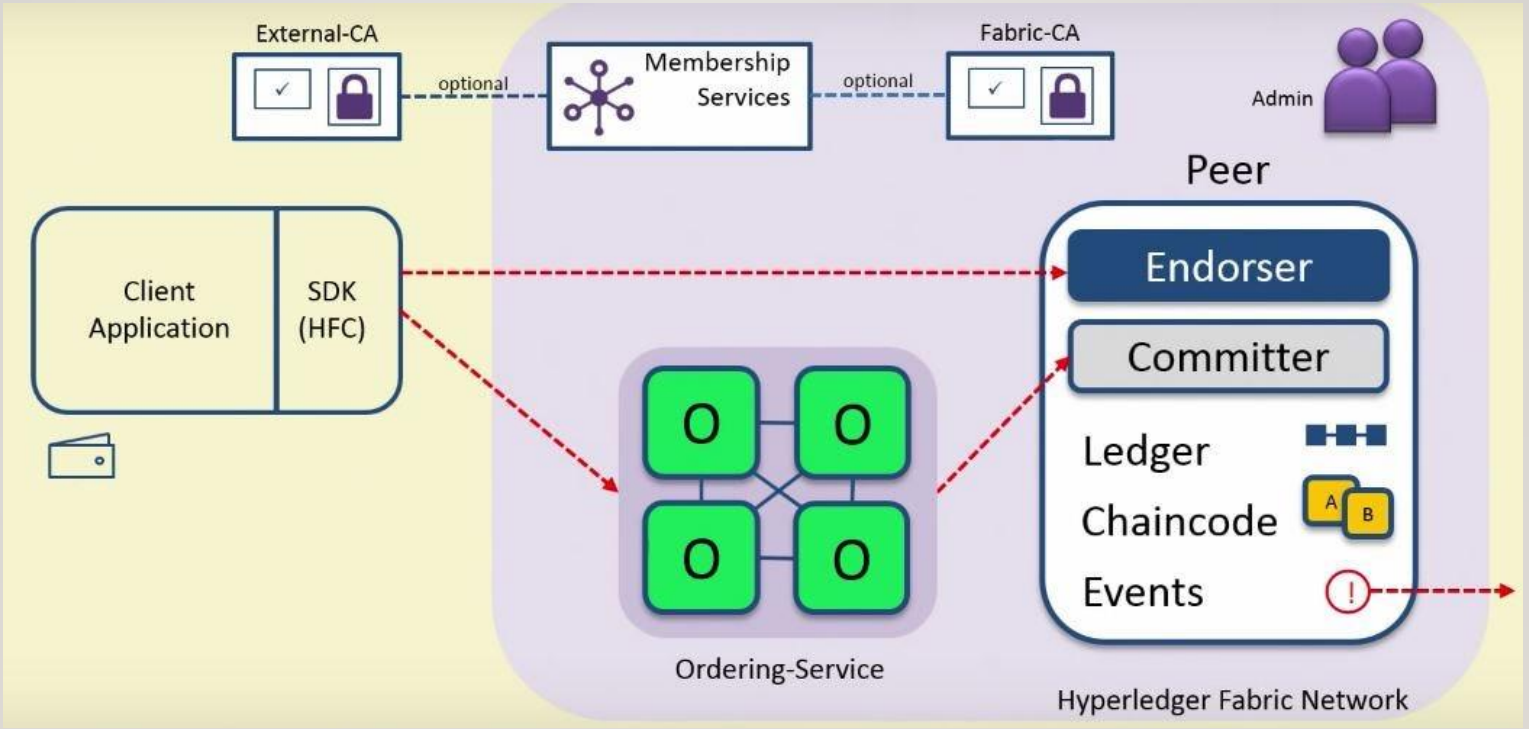
- **Committing Peer:** Peers who commit the block which is received from the Ordering service, in their own copy of the blockchain.
- This block contains the list of transactions where committing peer to validate each transaction and mark it as either valid or invalid and commits to the block.
- All transaction either valid or invalid are all committed to blockchain for future audit purpose.

ORDERER

- In a Blockchain network, transactions have to be written to the shared ledger in a consistent order.
- The order of transactions has to be established to ensure that the updates to the world state are valid when they are committed to the network.
- Unlike the Bitcoin blockchain, where ordering occurs through the solving of a cryptographic puzzle, or *mining.*
- Hyperledger Fabric provides three ordering mechanisms: SOLO, Kafka, and Simplified Byzantine Fault Tolerance (SBFT).

CHANNELS

- A fabric network can have multiple channels. Channels allow organizations to utilize the same network while maintaining separation between multiple blockchains.
- Only members(peers) of the channels are allowed to see the transaction created by any member in a channel.
- channels partition the network in order to allow transaction visibility for stakeholders only.
- Only the members of the channel are involved in consensus, while other members of the network do not see the transactions on the channel.


External-CA
optional
Membership Services
optional
Fabric-CA
Admin
Peer
Client Application
SDK (HFC)
Endorser
Committer
Ledger
Chaincode
A
B
Events
Ordering-Service
Hyperledger Fabric Network

BENEFITS OF HYPERLEDGER FABRIC

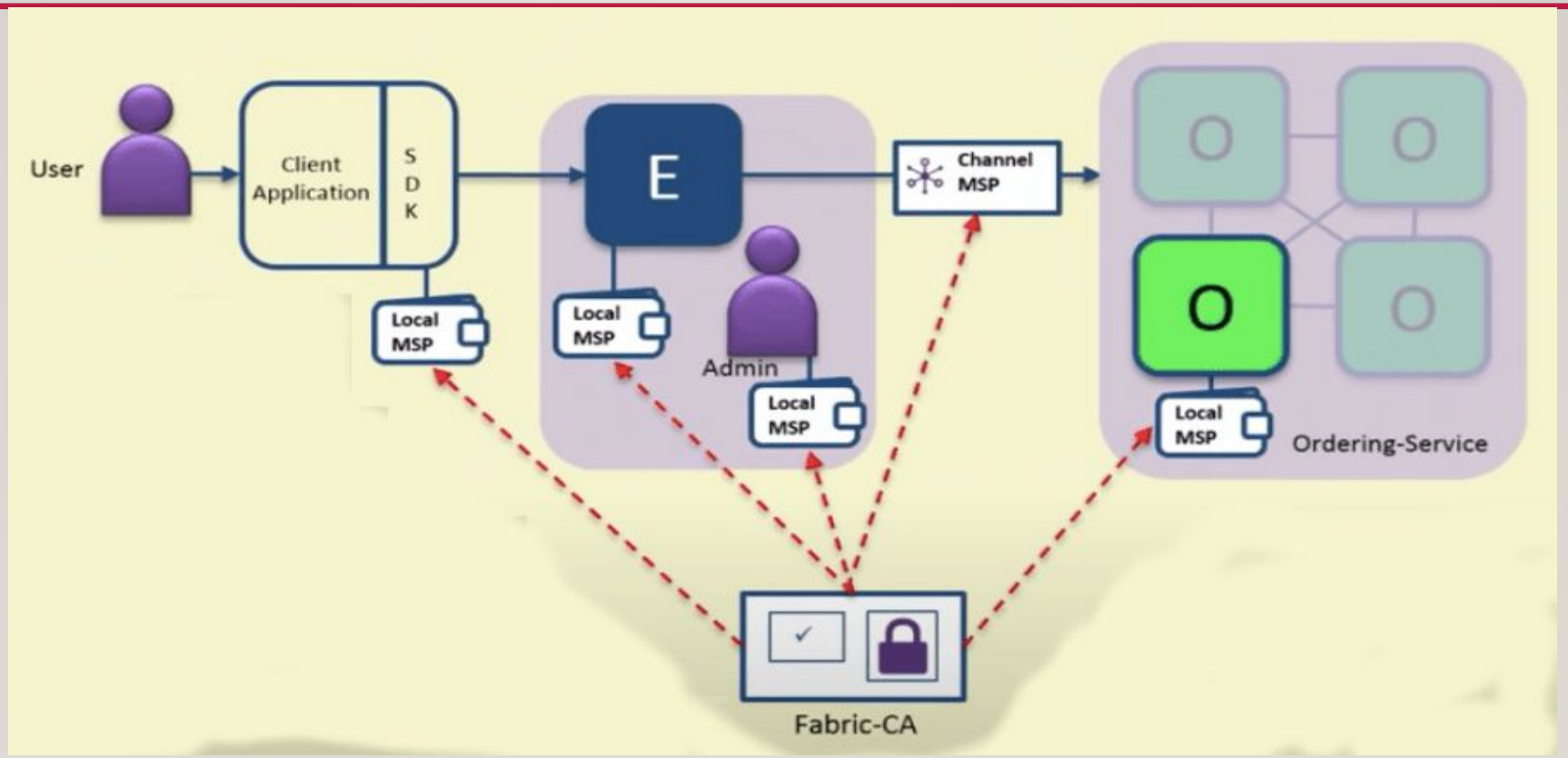
- Scalability: Hyperledger Fabric's modular architecture allows for multiple ledgers to exist within the same network, which makes it more scalable than other blockchain platforms.
- Flexibility: Hyperledger Fabric's pluggable consensus mechanism allows developers to choose from a variety of consensus algorithms depending on their specific use-case.
- Security: Hyperledger Fabric uses a permissioned blockchain model that allows only authorized users to participate in the network. This makes it more secure than public blockchain networks.

- Privacy: Hyperledger Fabric allows developers to define access control rules for their blockchain network, which makes it more private than other blockchain platforms.
- Interoperability: Hyperledger Fabric is designed to be interoperable with other blockchain platforms, which makes it easier to integrate with existing enterprise systems.

MSP

- Membership Service Provider (MSP) is a component in Hyperledger Fabric that manages the identities of nodes in the system, and provides roles and permissions to network members.
- There are a set of entities in the distributed fabric network, and each entity must be associated with a unique identity.
- These identities can be issued by Fabric Certificate Authority (Fabric-CA) or an external CA, and the Membership Service Provider (MSP) manages these identities across the entire network.
- The identities are issued by the CA, along with the CA signature on it, is called a certificate, and one of the predominated certificate formats is x.509.

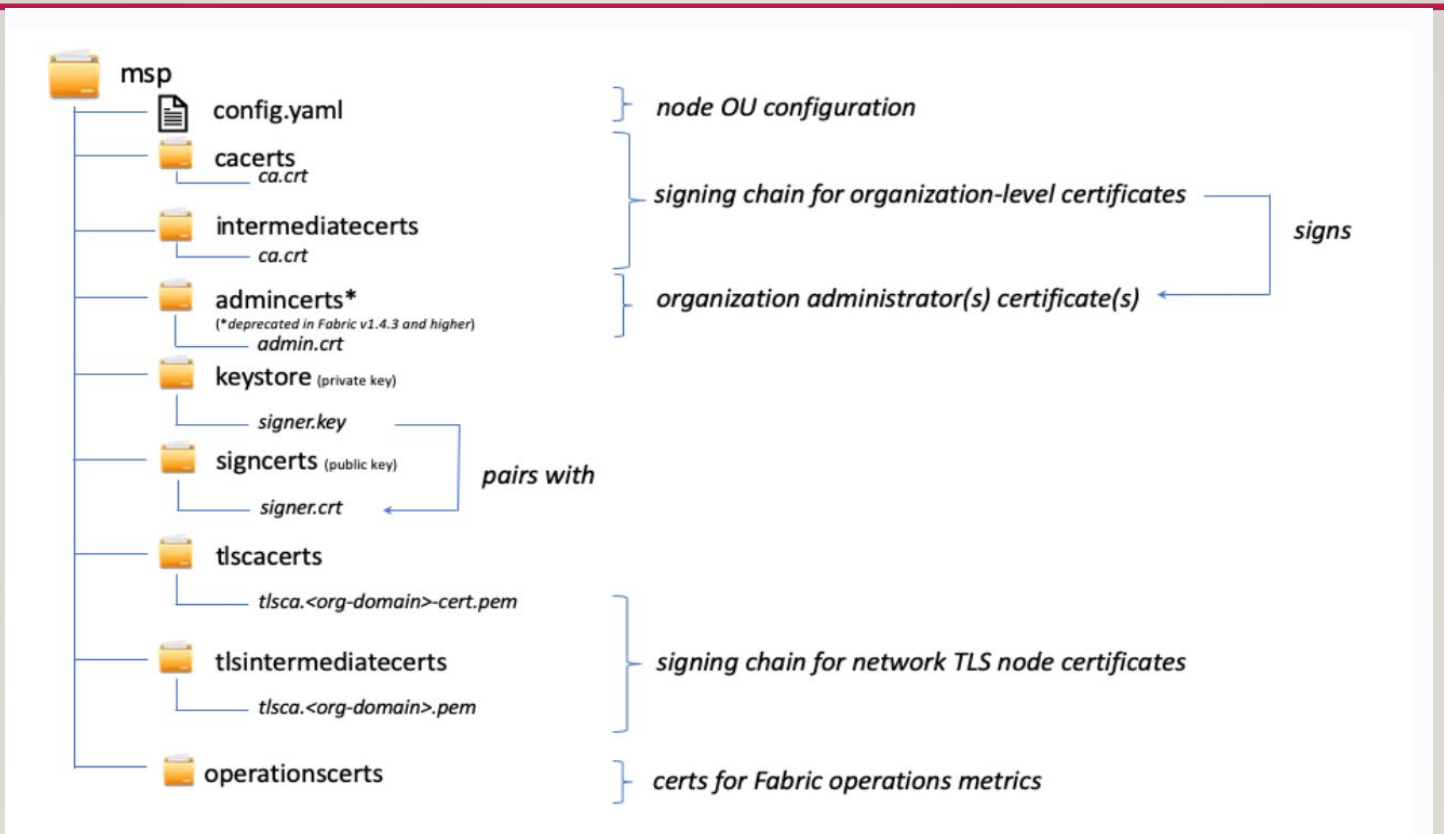
- These are various entities such as peers, orderers, client applications, and administrators in the blockchain network, and only the authorized entities can perform the operations.
- The user will sign a transaction send it to the peer, and the peer will execute the transaction, and when it endorses, it is actually signed the endorsement. In the same way, the ordering service when it creates a block and sends it to the peers. Each block is going to be signed by the ordering service.


User
Client Application
S
D
K
E
Local MSP
Local MSP
Admin
Local MSP
Channel MSP
O
O
O
O
Local MSP
Ordering-Service
Fabric-CA

- The MSP can support different crypto standards, and it is a pluggable interface.
- A network can have multiple MSPs; however, it is recommended to have one MSP per organization. So based on the number of organizations, the entire network have that many MSPs.
- Each entity must have a local MSP, and it is used to store credentials, such as private keys, securely using hardware modules provided by hardware platforms.

- Local MSPs: These are defined for clients and nodes, determining permissions at a node level. Each node must have a local MSP, defining administrative rights and participatory roles.
- Channel MSPs: Channel MSPs define administrative and participatory rights at the channel level. They include MSPs of all organizations participating in a channel, ensuring authentication and policy enforcement.

MSP STRUCTURE



- **cacerts:** Contains self-signed X.509 certificates of Root CAs trusted by the organization.
- **intermediatecerts:** Holds X.509 certificates of Intermediate CAs, establishing a chain of trust.
- **admincerts:** Deprecated from Fabric v1.4.3 onwards, contains certificates of administrators.
- **keystore:** Contains the node's private key for signing transactions.
- **signcert:** Holds the node's certificate issued by the CA, representing its identity.
- **tlscacerts:** Contains Root CA certificates for secure communication using TLS.
- **tlsintermediatecacerts:** Holds Intermediate CA certificates for TLS communication.
- **operationscerts:** Contains certificates for communicating with the Fabric Operations Service API.
- **revoked Certificates:** Stores identifiers of revoked identities, ensuring network integrity.

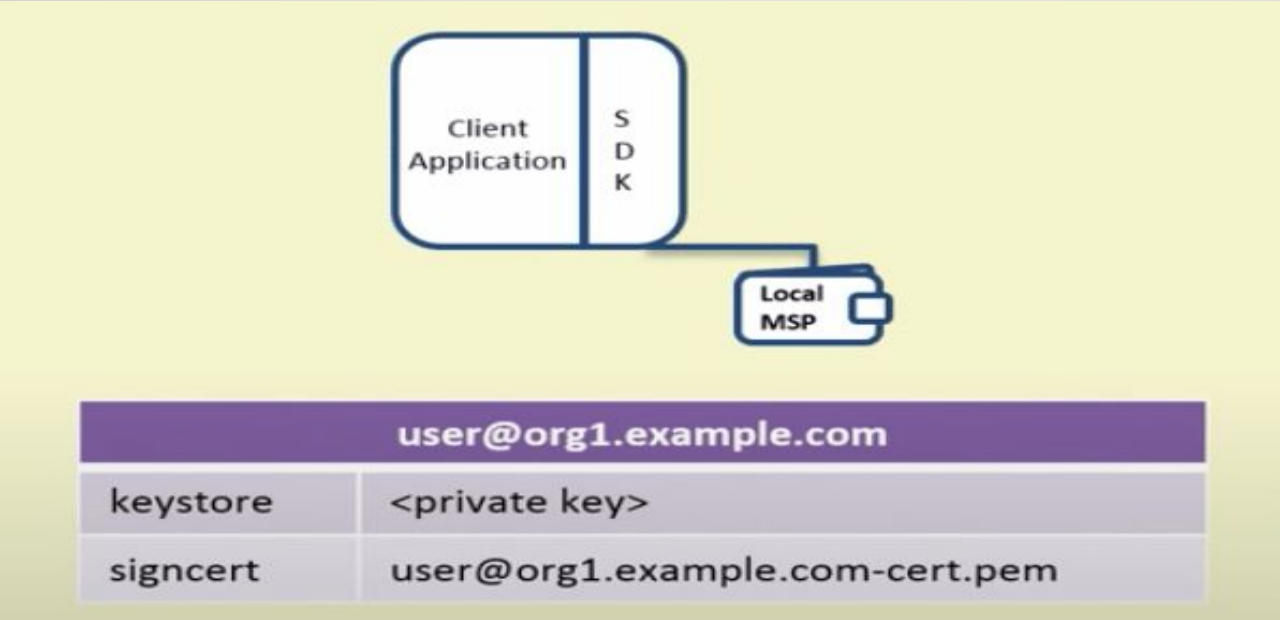
THE ROLE OF MSP

1. Defining Trust Domains: MSPs identify trusted Certificate Authorities (CAs) and list the identities of their members, establishing trust within a network.
2. Role Assignment: MSPs turn identities into roles by specifying privileges within the network. For instance, an identity registered with the “peer” role is designated for a peer node.
3. Revocation Management: MSPs facilitate the identification of revoked identities, ensuring the integrity of the network. This prevents unauthorized access and maintains security.

- Issuing and maintaining identities: MSPs issue and maintain identities for all nodes in the Hyperledger Fabric system using digital signatures.
- Registering participants: MSPs register participants on the network.
- Storing participant information: MSPs store information about participants.
- Generating and issuing certificates: MSPs generate and issue certificates for participants.
- Providing access control: MSPs provide access control over permissions.
- Turning identities into roles: MSPs turn identities into roles by identifying specific privileges an actor has on a node or channel.
- Identifying revoked identities: MSPs can identify a list of identities that have been revoked.

USER IDENTITIES

- Every user in the network received an identity and an enrollment certificate. The enrollment certificate has two parts, one is a private key, and the private key is private to a particular user and used for digitally signing the transactions that will submit onto the network. The private key is stored securely using a hardware security module provided by the computer hardware platform.
- The second part is a signcert, which is a public x.509 certificate in the fabric implementation. The public key includes a public key and other attributes provided by the certificate authority (CA). However, other crypto standards can also be used.



A signcert includes various attributes, including organization details, certain roles, authorization, and other information.

PEER AND ORDERER IDENTITIES

- Each peer and ordered are having a private, public key, and sign certificate from CA. A local MSP is attached to the peer who holds these identities. In addition, the peer and MSPs can identify authorized administrators. So it can have one or more administrators. Each administrator has the following artifacts:
  - admincert: A list of administrator certificates.
  - cacerts: The CA public cert for verification. As peers and orderers deal with transactions and transactions are generated and signed by different users from different organizations. So it includes multiple CA public certificates for verification of requests.
  - crls: List of revoked certificates.

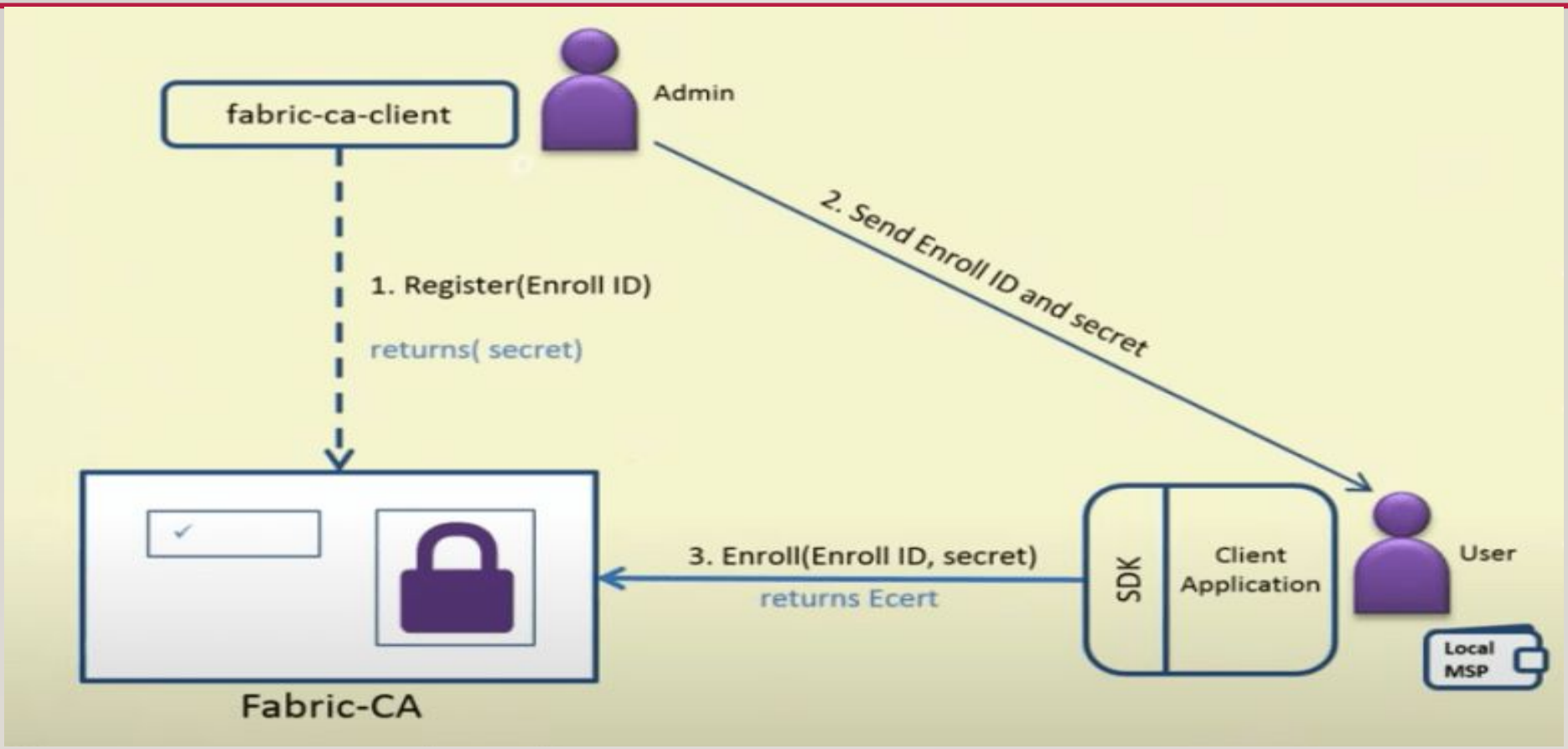
HYPERLEDGER FABRIC – PEER AND ORDERER IDENTITY

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>

CHANNELS MSP

- Channels include additional organizational MSP information, such as which peers and orderers are joining or leaving the channel, and these are dynamic in nature.
- The channel MSP includes:
  - admincerts: Any public certificates for administrators.
  - cacerts: The CA public certificate for this MSP.
  - crls: List of revoked certificates.

NEW USER REGISTRATION AND ENROLLMENT



TRANSACTION MANAGEMENT