# Blockchain
# **Basics and Beyond**

# Objectives

- Newbie to Intermediate
- Introduction to the World of Blockchain
- Learn the inner workings of Blockchain with Bitcoin
- Deep Dive on Ethereum and EVM based Blockchains
- Understand how to interact with Blockchain Applications.
- Build Smart Contracts in Solidity
- Create Decentralized Applications.

# Prerequisites

- Beginner Friendly Course
- Fundamentals of Programming and Application Development
- Basics of Modern Javascript would be handy

# Contents of the Course

- Blockchain and Bitcoin
  - Introduction to Digital Currencies
  - Cryptography
  - Blockchain Structure and Features
  - Consensus in Distributed Networks
- Ethereum and Dapps
  - Intro to Ethereum
  - Ethereum Virtual Machine
  - Programming in Solidity
  - Build Smart Contracts
  - Build Dapps

# Contents of the Course

- Blockchain and Bitcoin
  - Introduction to Digital Currencies
  - Cryptography
  - Blockchain Structure and Features
  - Consensus in Distributed Networks
- Ethereum and Dapps
  - Intro to Ethereum
  - Ethereum Virtual Machine
  - Programming in Solidity
  - Build Smart Contracts
  - Build Dapps

# Key Takeaways

- Clear working Knowledge of Blockchains
- Bitcoin and Ethereum Blockchains functionalities and differences.
- Explore Blockchain Application Source Codes.
- Create Decentralized Application using Smart Contracts built using Solidity.

# Sub sections

- Crypto Wallets

# Srikanth Alva

Full Stack Developer, Blockchain Mentor

linkedin.com/in/srikanthalva

subtlecrypto.substack.com

Module 1

# Blockchain and Bitcoin

# Intro to Digital Currencies

# Currency

## 1 Medium of Exchange

Durable, Transportable, Divisible, Fungible and Non-counterfeit.

The ease with which a currency can be exchanged for products and services.
Ex. Dollar

## 2 Unit of Account

Stability

Ability to Price products in terms of a stable measure. Indicates **measurement of value** of products.
Ex. Dollar, Euros

## 3 Store of Value

Ability to preserve value over a long period with some level of predictability.
Ex: Gold, Silver, Real Estate, Stocks

Note: No currency plays all these roles perfectly. There are always trade-offs, between efficiency and stability, inflation risks and deflation risks.

# Phases of Currency Evolution

| Before ~700 BCE | ~700 BCE | ~900 AD | ~1200 AD | 2008 Onwards |
|---|---|---|---|---|
| **Barter System** | **Primitive Money** | **Coinage** | **Fiat Currency** | **Digital Currency** |
| trading goods directly, without any intermediary ‘medium of exchange.’ | cowry shells, cattle, tobacco and others emerged as a form of “medium of exchange” | precious metals and stone replace the primitive money they where much durable and transportable. | Paper Money started as receipt of deposit of coins. Later, trusted paper notes itself was considered as currency, non redeemable in coins or metals | With the invention of Bitcoin, first ever digital currency without any centralized issuer was created. |

# Fiat Currency vs Digital Currency

## Fiat Currency

- Controlled By the Government Institution..
- You can pay your taxes.
- Has Physical Form.
- Government controls the supply of the currency.
- Always Centralized.

## Digital Currency

- Independent of Government Institutions Control.
- Cannot pay your taxes (most cases)
- Exists only in Digital Form.
- Algorithm controls the supply of the currency.
- Mostly Decentralized.

# When Currencies goes Wrong

**1923**

**Hyperinflation in Germany** - Unable to meet its obligations of very high war reparation obligations after **WW1**, it tried to print more Marks to buy foreign currency, triggering a further drop in Marks, necessitating more printing and creating a vicious spiral.

**1930**

**The Great Depression in US** - stock market collapse of 1929 triggered a recession. Constrained by the gold standard, the Federal Reserve constricted money supply, causing severe deflation and brought the end of the domestic gold standard in the US.

**1945**

**Hungary** is believed to have had the worst hyperinflation in history due to an high war reparation costs. Its inflation peak was at 1.3 × 10^16 percent per month (prices doubled every 15 hours)

**2016-20s**

**Venezuela and Zimbabwe** Poor governance, corruption and misguided policies, extensive printing of currency have fueled runaway inflation, shuttered businesses.

# Bitcoin as Currency?

### As Medium of Exchange

- Highly Durable
- Highly Portable
- Highly Fungible
- Highly Divisible
- Resistant to Counterfeit

+

### As Unit of Account

**0.0036 BTC = 100 USD**

**0.0039 BTC = 100 EURO**

**High Price Fluctuation.**
Regulation, Onramp/offramp services and futures markets would contribute to price stabilization.

+

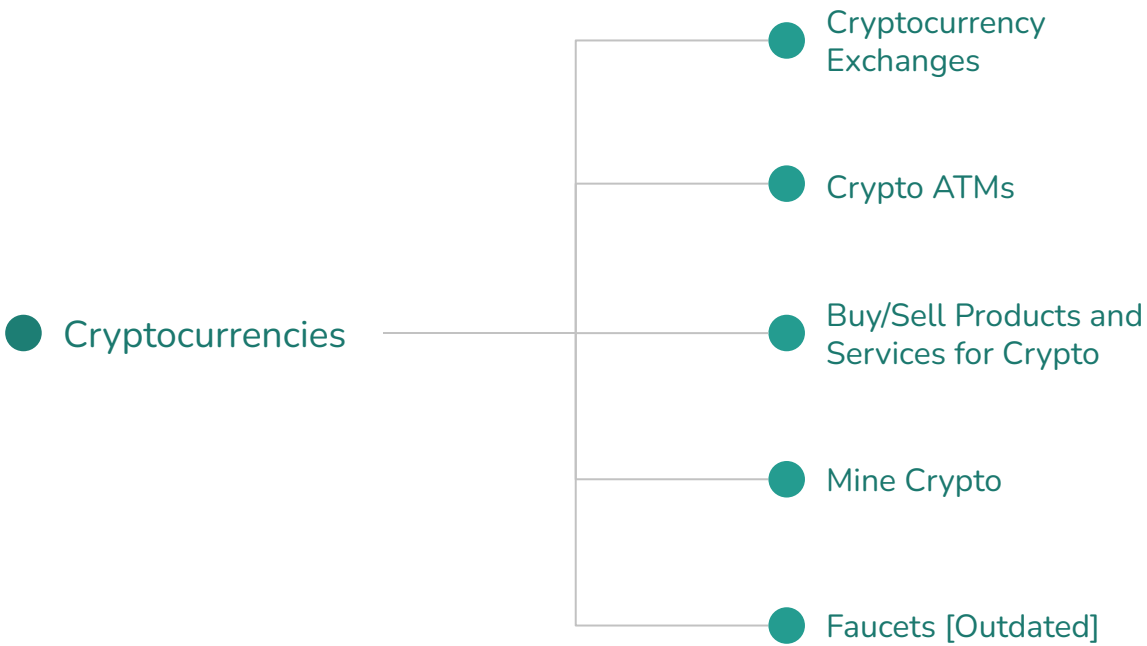
### As Store of Value

- Supply is Predictable
- Issuance is predetermined
- Issuance decrease in half every ~ 4 Yrs
- **High Price Volatility**

## Bitcoin is not just a Currency

Bitcoin is a financial instrument, a currency, a method of investment, a medium of exchange, a unit of account a store of value, a type of settlement system and much more

# Five Ways to get Cryptocurrencies



# Evolution of Web

# Web 1.0

Web 1.0 refers to the first stage of the World Wide Web evolution.

- Few content creators
- Majority were Content Consumers.
- Static pages.
- Content is served from the server’s file system.
- Few Opportunities for interactive engagement.
- Top Applications: Email and Real-time news retrieval.

# Web 2.0

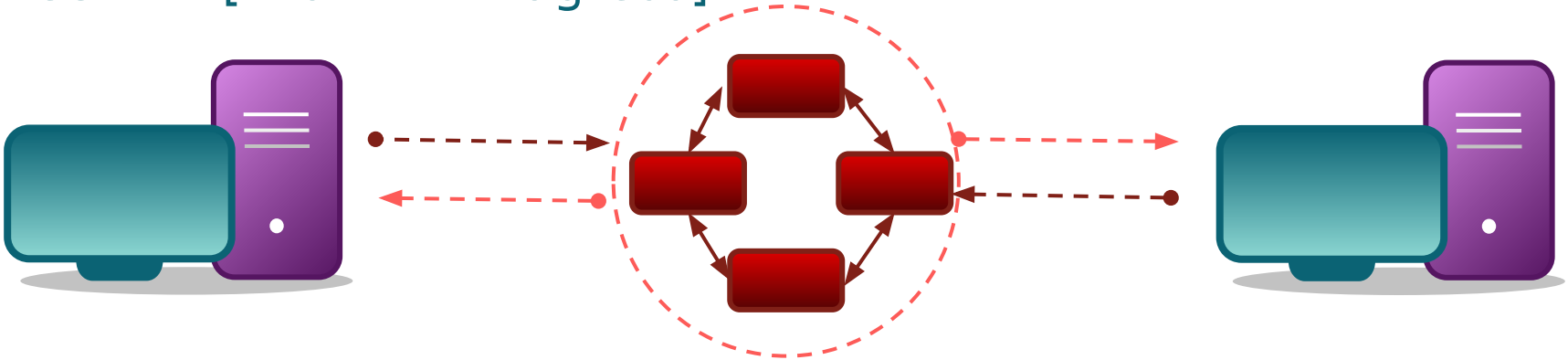
Web 2.0 refers to applications that highlight user-generated content, usability, and interoperability for end users.

- Web 2.0 is also called the participative social web.
- Search engines allowed users to sort, classify  and retrieve information.
- Innovation in mobile internet made internet accessible to billions across the globe.
- Dynamic content that is responsive to user input and preferences.
- APIs allowed integration of third parties with ease
- Top Applications: Social Media Platforms and Streaming Services.

# Problems with Web 2.0

| Issue | Description |
| --- | --- |
| Control and Ownership of Data | Extracted private information is used to spam you with unwanted advertisements, or even selling your data to a third party |
| Public Access and Censorship | Governing bodies can limit access to information and limit expression, essentially at their whims when they operate under a centralized system. |
| Mismanagement of User Data | Companies can store, use or sell user's data without their knowledge or consent |
| Trusting the Good-Faith Operation | Sometimes, organizations may misuse data or there may be a security lapse in the system. |
| Single Point of Failure | Mission-critical services suffer from uncertainty and vulnerability when they are used because of this limitation of centralized systems. |

# Web 3.0 [Work in Progress]

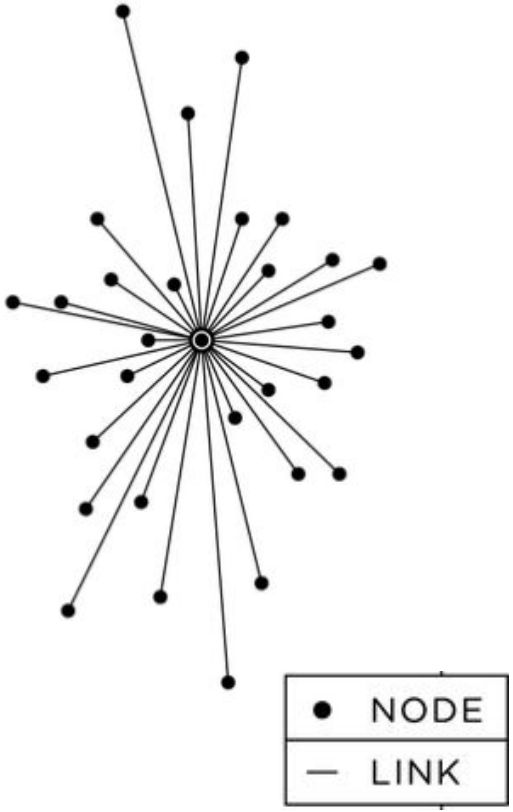


The third phase of the evolution of internet is characterized by decentralization, permissionless, trustlessness and artificial intelligence with a vision to open web with greater utility for its users.

- Defining features: decentralization, artificial intelligence and machine learning;
- Decentralization and permissionless systems, will also give users much greater control over their personal data.
- limit the practice of data extraction
- curb the network effects that have enabled the technology giants to become near-monopolies

# Network Types

# Centralised Networks




In a centralized system, there is a central network owner (or server) which is the only store of information that users can query, access information from, and store information on.

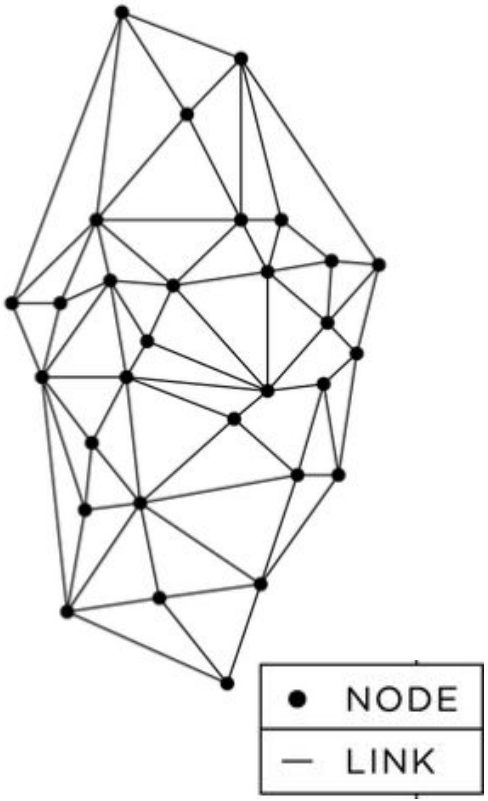
Centralized systems are quite common and they are the primary system used in the development of the internet.

Advantages

- Simple deployment – only needs one server
- Can be developed quickly
- Affordable to maintain
- Practical when data needs to be controlled centrally

Example: small business using a single domain controller

# Distributed Networks




In a distributed system, the computation and resources are shared among the nodes of a network, hence provides better security and user experience.

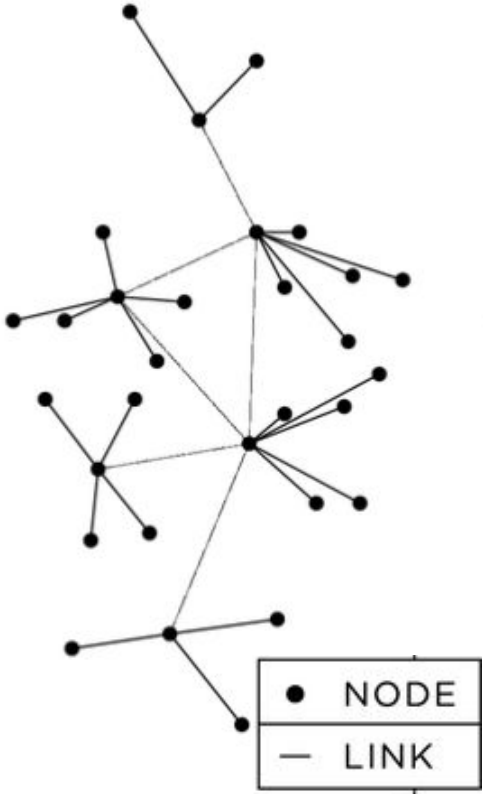
A distributed system is similar to a centralized one but it has high fault tolerance, while still managed under a central authority.

Advantages

- Fault-tolerant
- Secure
- Promotes resource sharing
- Extremely scalable

Example: Google search system.

# Decentralised Networks




These systems are distributed systems but they don't have a single owner, but ownership is spread across the users in the network itself.

fault tolerance is high with this system, maintenance costs are relatively high, and scalability is achievable relative to a centralized network

Advantages

- Less likely to fail than a centralized system
- Better performance
- Allows for a more diverse and more flexible system
- Decentralization of Control
- Difficult to censor

Example: Torrents and Bitcoin

# Comparison: Centralized & Decentralized Networks

| Characteristic | Centralized | Decentralized |
|---|---|---|
| Third-Party Involvement | Yes | No |
| Ownership and Use of Data Administered by | A central authority | The user |
| Vulnerability to Hacking | More prone to hacks and data leaks | Less prone to hacks and data leaks. |
| Single Point of failure | Yes | No |
| Ease of use | Intuitive and easy to use | Can be difficult, especially at first |
| Anonymity | Users can act pseudo-anonymously | Can offer full anonymity |

# Cryptography

# Hash Functions

Algorithms that **transform** an **input of arbitrary length** to an **output of fixed length string.** It is used to create **unique representation** of **digital data**. 1

Input

Output

Arbitrary Length Data

Fixed Length Representation of Data

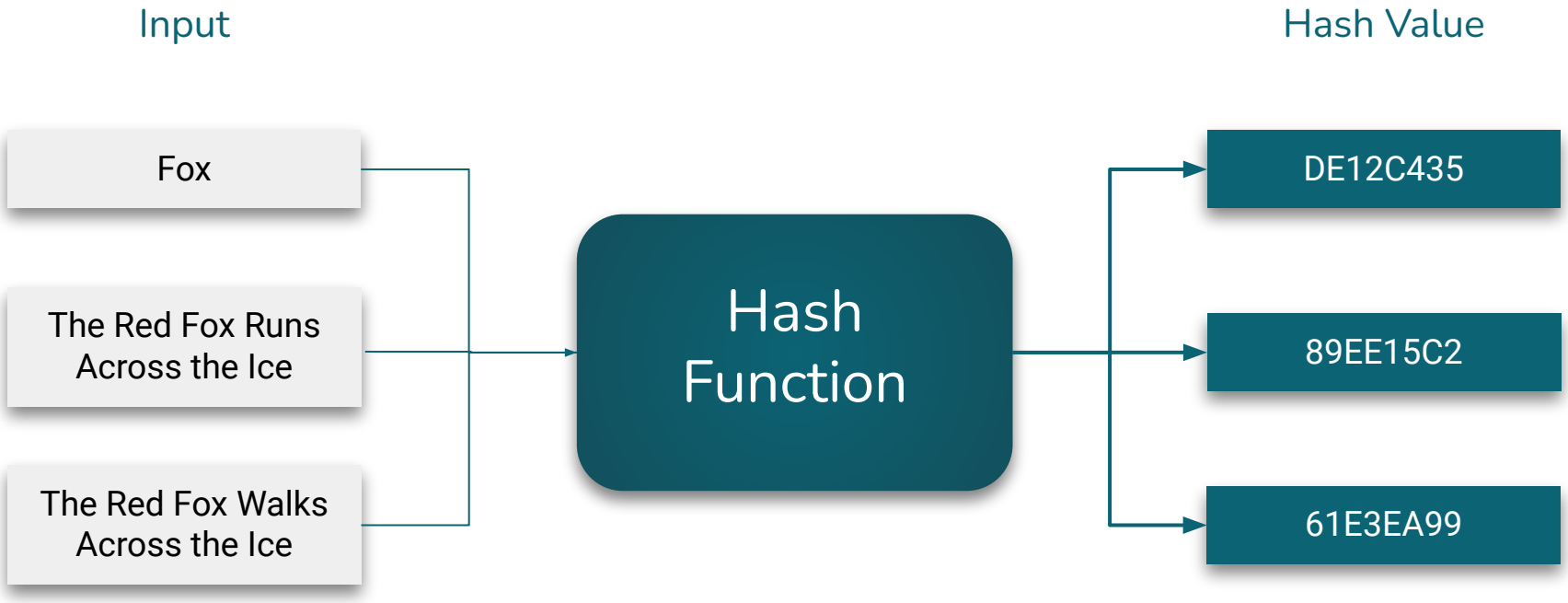
Sample Text Data

# Criterias for viable Hash Functions

1. **Fast to Compute** - calculation of outputs is extremely fast.
2. **Deterministic** - A given input always needs to produce the same output irrespective of time.
3. **One Way Function** - it is extremely difficult to guess the input using the output.
4. **Collision Resistant** - different inputs should **NOT** produce same output.
5. **Pseudorandom -** one input-output combination does not reveal anything about any other input-output combo..

| Fast to Compute | Deterministic | Collision Resistance | One Way | Pseudo-Random |
|---|---|---|---|---|

# Hash Functions




Fast to Compute | Deterministic | One Way | Collision Resistance | Pseudo-Random

# Use-cases of Hash Functions

General Use-case:

- Password Storage

Blockchain Use-cases:

- Creating an address
- Proving ownership
- Mining
- Transaction Verification
- Merkle Trees

## Hashing Algorithms

- SHA
  - SHA-256 - produces 256 bit Output
  - SHA-512
- RIPEMD
  - RIPEMD160

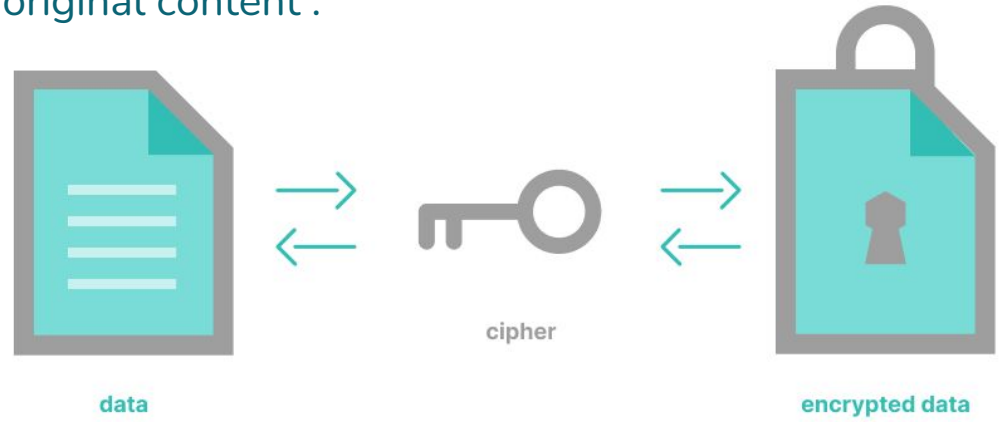
# Encryption

# Encryption

**Encryption** is a process that encodes/scrambles content (e.g., messages, files) to protect its confidentiality and integrity.

- Algorithms are used to scramble data to create **cipher text**.
- Encrypted data is rendered virtually unusable without the key.

**Decryption** is the process of using a **unique key** to unscramble the encrypted data(Cipher text) to retrieve the original content .



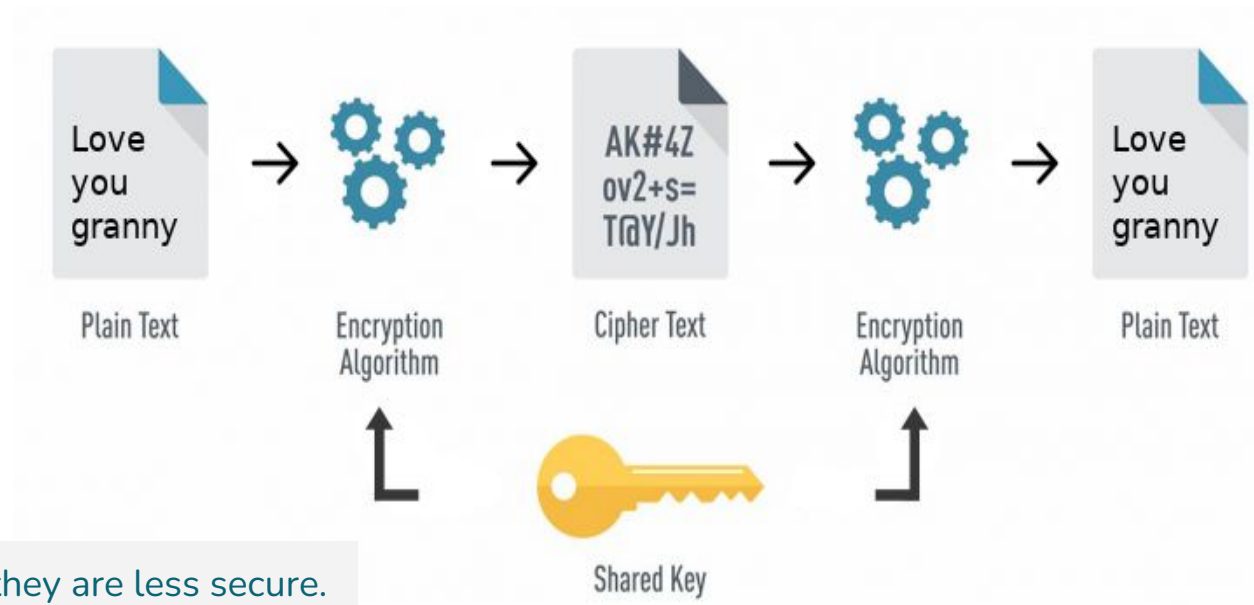
# Symmetric Encryption

- One Key is shared between multiple entities.
- Algorithms are not complex and fast.
- Less Secure.
- Prone to brute force attacks.

## Common Algorithms

- AES
- DES

Lock and Key Analogy

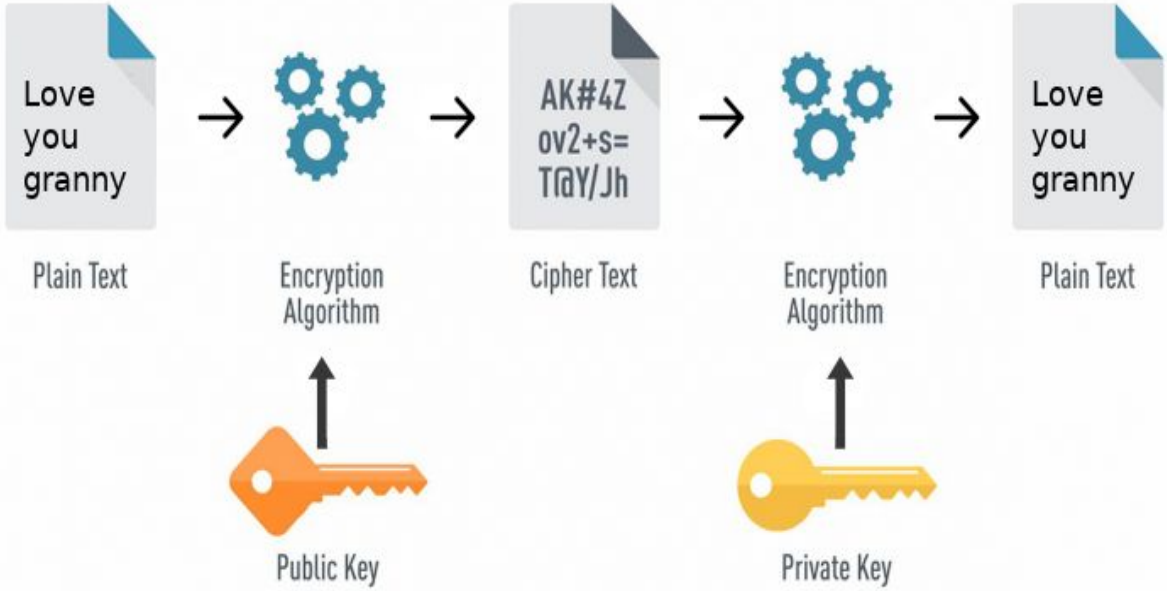



Note: Not used in Blockchains as they are less secure.

# Asymmetric [Public Key] Encryption

- One Entity has Public Key and the other has the Private Key.
- Uses Complex Algorithms and is considerably slow.
- Secure & Less prone to brute force attacks.
- Public Key is derived from Private Key.
- RSA & Diffie-Hellman

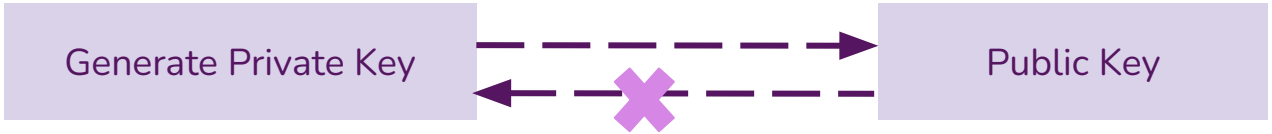
Mail-Box Analogy




Use Cases

- HTTPS
- End-to-End Encryption
- Key Generation
- Digital Signatures

# Two Types of Usage




1

**Public Key** for Encryption
**Private key** for Decryption

→ Secure Message Transmission

---

2

**Private Key** for Encryption
**Public Key** for Decryption

→ Digital Signatures

# Comparison: Encryption and Hashing

## Encryption

- Encryption is the practice of scrambling information in a way that only someone with a corresponding key can unscramble and read it.
- Encryption is a **two-way** process.
- Used in Securing Transactions and Digital Signatures

## Hashing

- One-way transformation of data to an unreadable piece of data using an algorithm to map data of any size to a fixed length.
- Hashing is a **one-way** function.
- Used in Key Generation and Digital Signatures and Verify Integrity in Blockchains

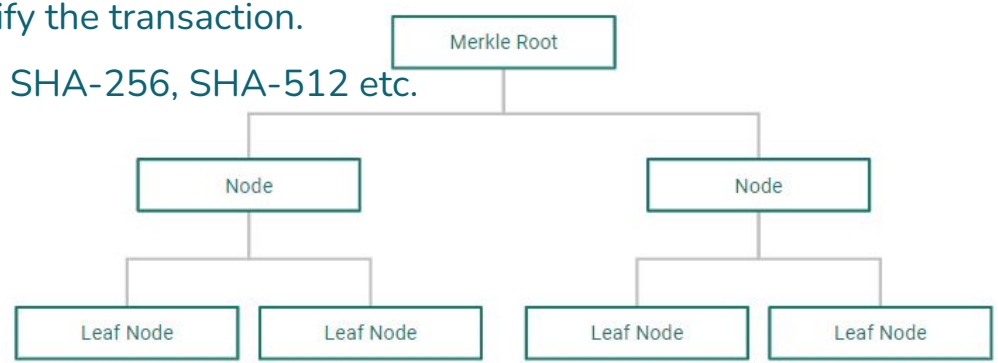

Cryptography

# Merkle Tree & Merkle Root

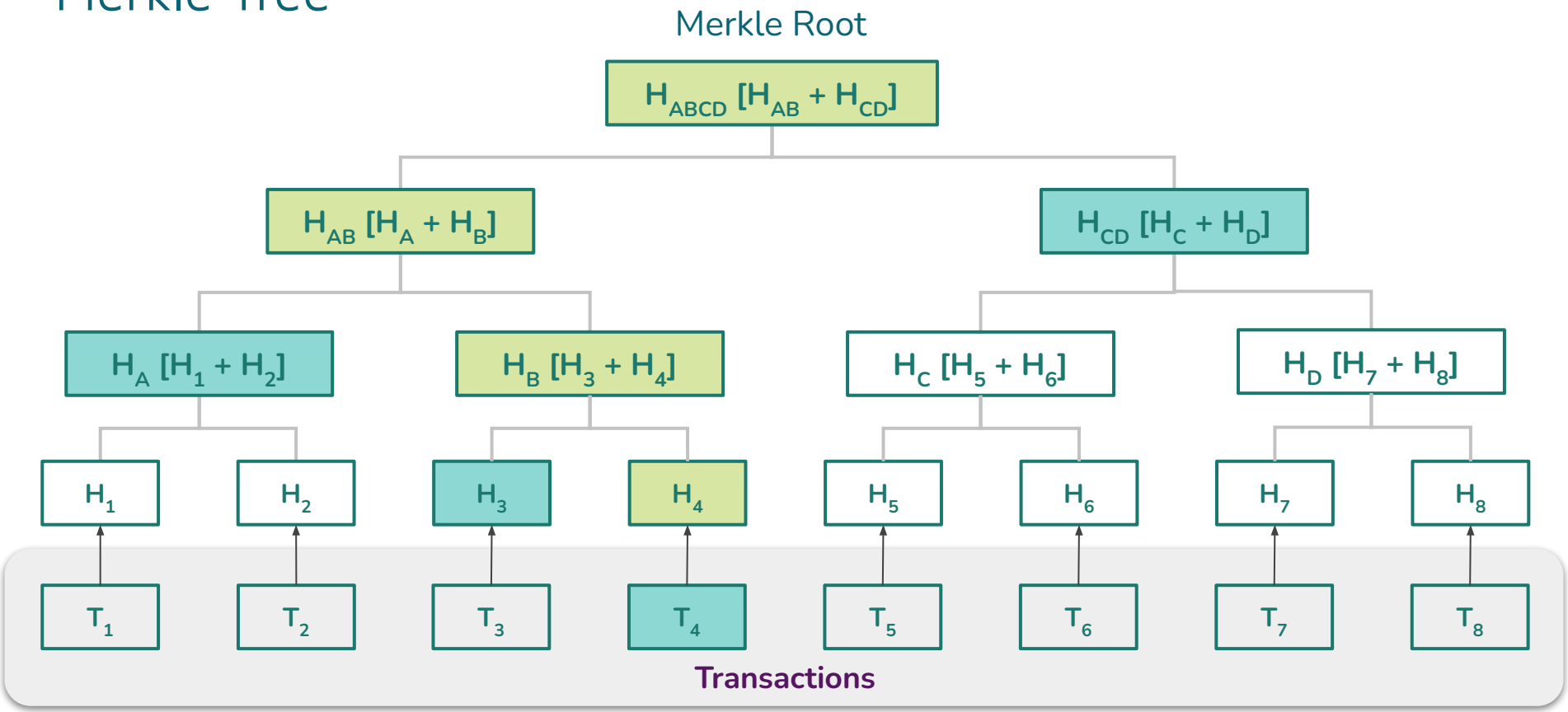
# Merkle Tree

- Merkle Tree is a binary tree.
- Merkle tree is constructed by hashing pairs of nodes recursively until there is one remaining. It is called merkle root.
- Every block in a blockchain, has a placeholder, which contains the summary [digital fingerprint] of all the transactions recorded in a block. This is done using a merkle tree.
- The summary is used to verify whether a particular transaction is included in a block or not.
- Lightweight Nodes in most cases download only block headers instead of the complete block. These nodes use Merkle Tree details to verify the transaction.
- Usually SHA-2 family Hashes are used like SHA-256, SHA-512 etc.
- SPV - Simplified Payment Verification

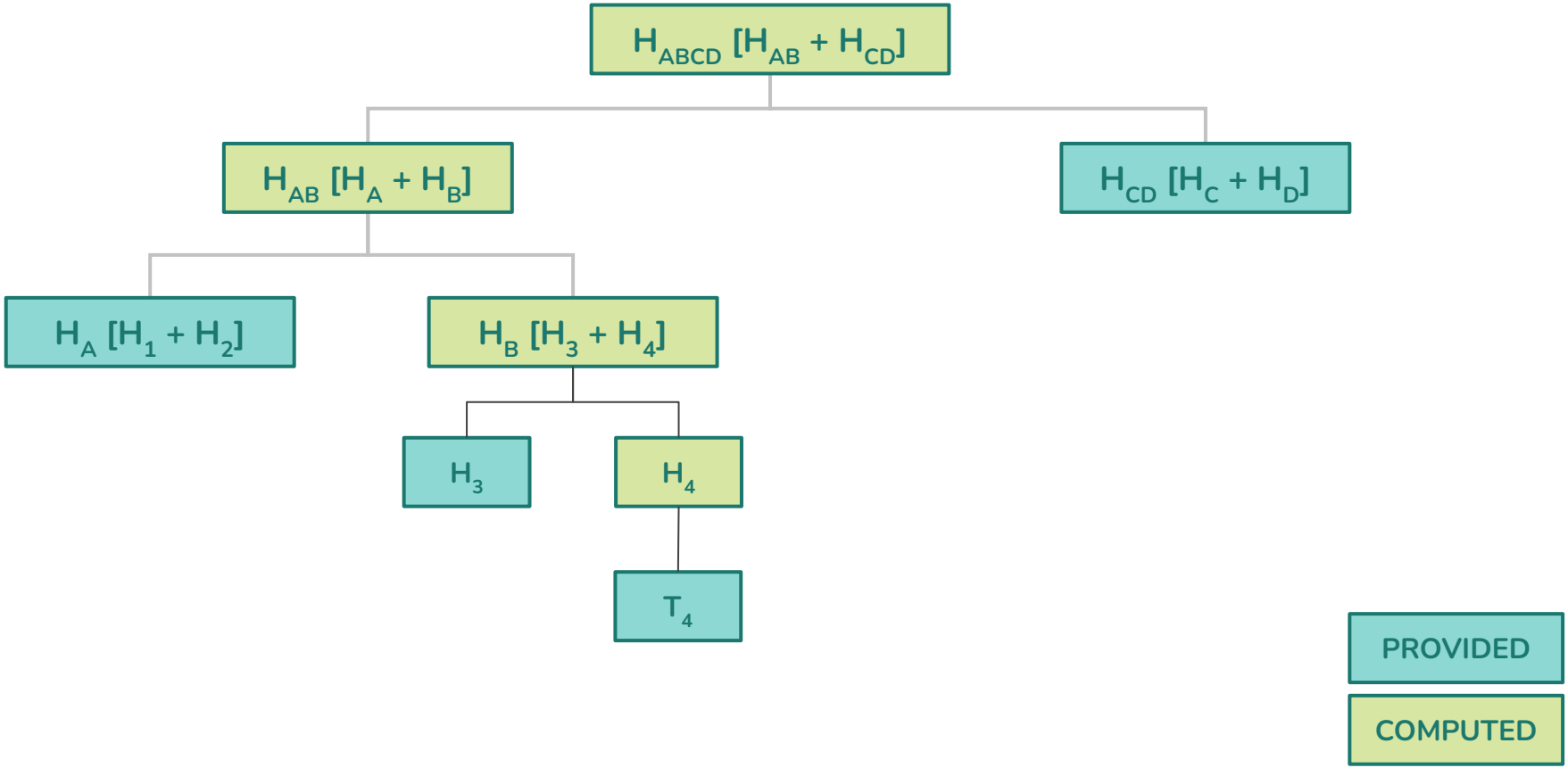
Merkle Tree →



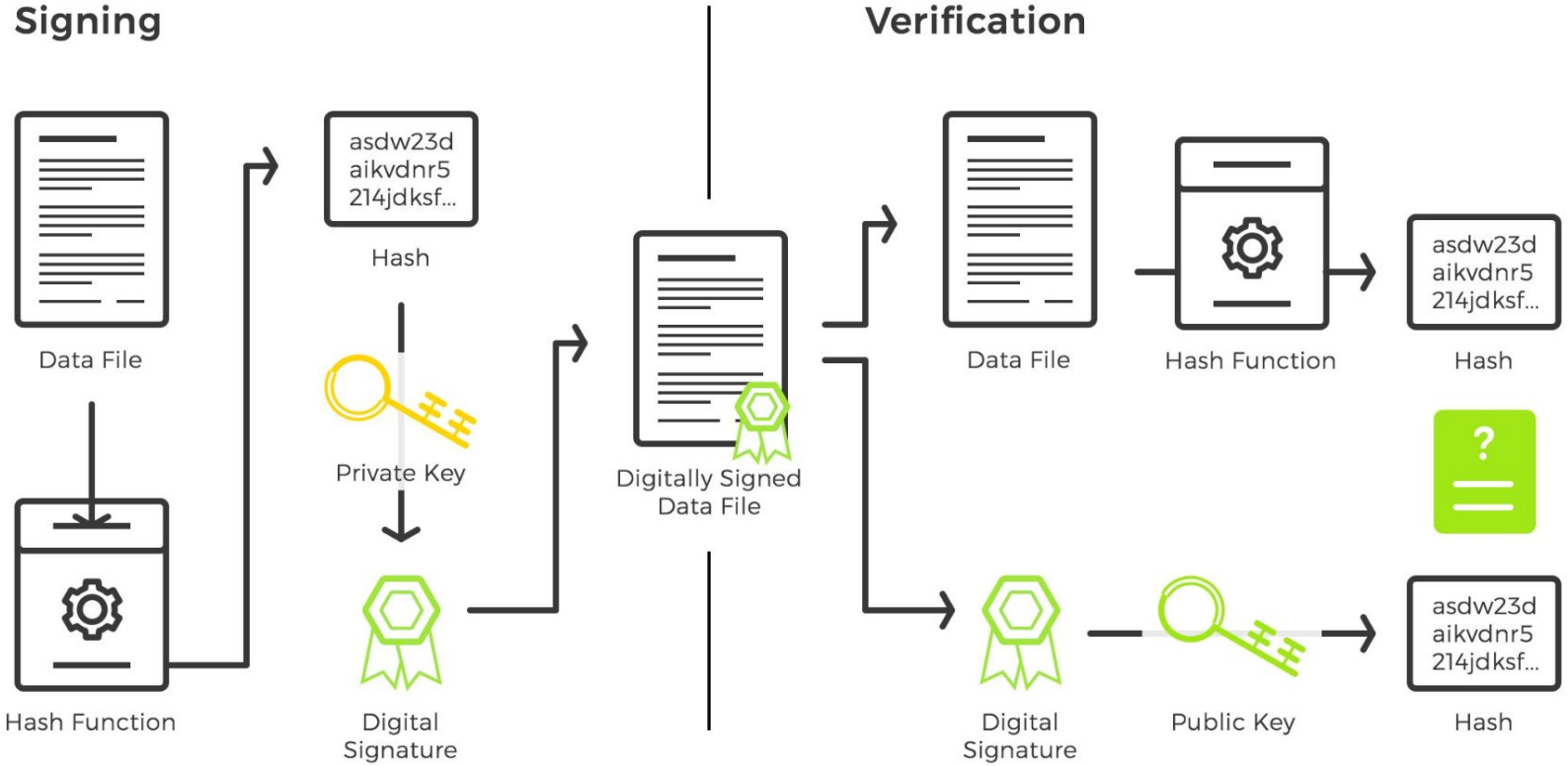
# Merkle Tree


Merkle Root
$H_{ABCD}$ [$H_{AB}$ + $H_{CD}$]
$H_{AB}$ [$H_A$ + $H_B$]
$H_{CD}$ [$H_C$ + $H_D$]
$H_A$ [$H_1$ + $H_2$]
$H_B$ [$H_3$ + $H_4$]
$H_C$ [$H_5$ + $H_6$]
$H_D$ [$H_7$ + $H_8$]
$H_1$
$H_2$
$H_3$
$H_4$
$H_5$
$H_6$
$H_7$
$H_8$
$T_1$
$T_2$
$T_3$
$T_4$
$T_5$
$T_6$
$T_7$
$T_8$
Transactions

# Merkle Tree


$H_{ABCD}$ $[H_{AB} + H_{CD}]$
$H_{AB}$ $[H_A + H_B]$
$H_{CD}$ $[H_C + H_D]$
$H_A$ $[H_1 + H_2]$
$H_B$ $[H_3 + H_4]$
$H_3$
$H_4$
$T_4$
PROVIDED
COMPUTED

# Digital Signature with Public Key Encryption



# Bitcoin

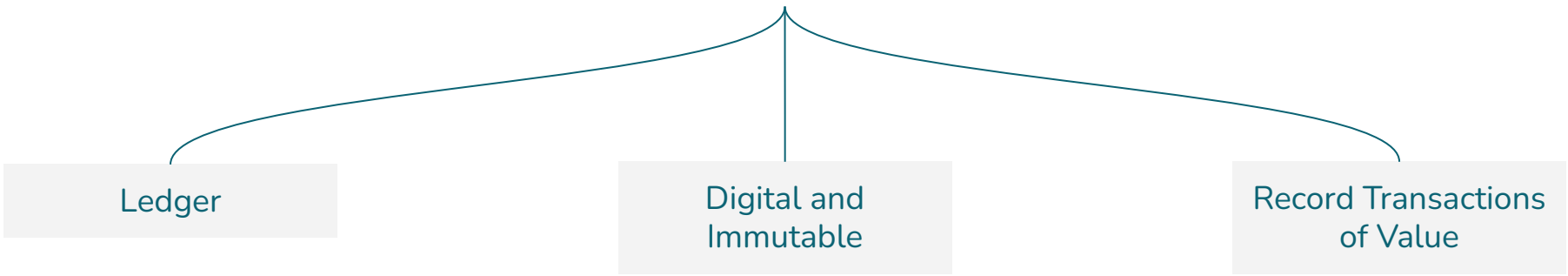
## The First Blockchain

# Blockchain Fundamentals

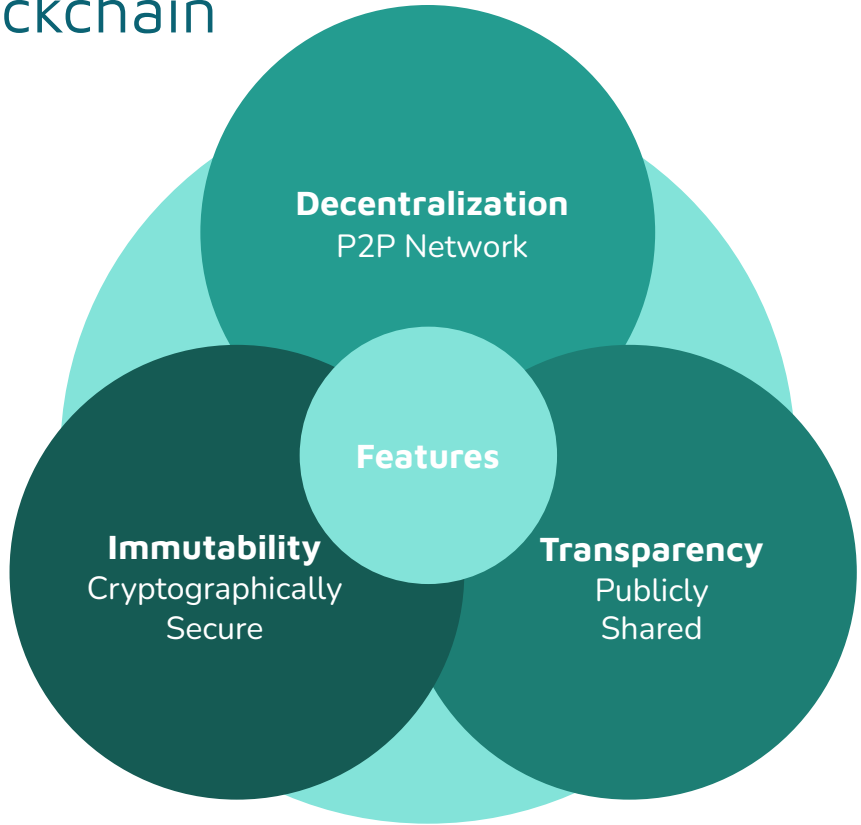
## With Bitcoin

# Blockchain

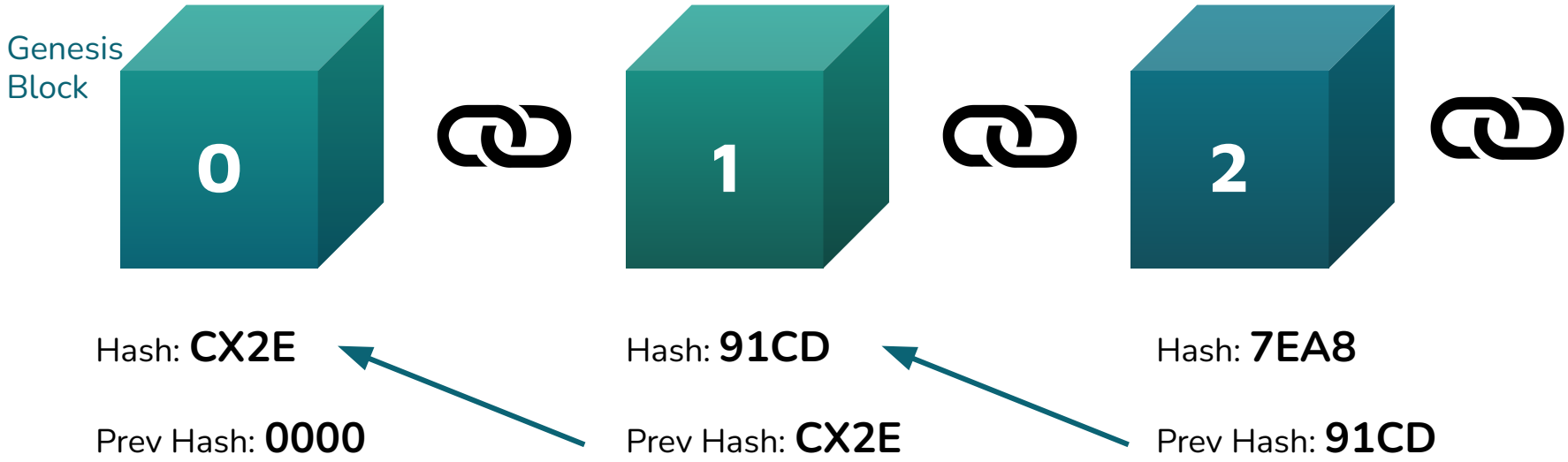
is an immutable digital ledger that facilitates the process of recording transactions of value.



# Features of a Blockchain

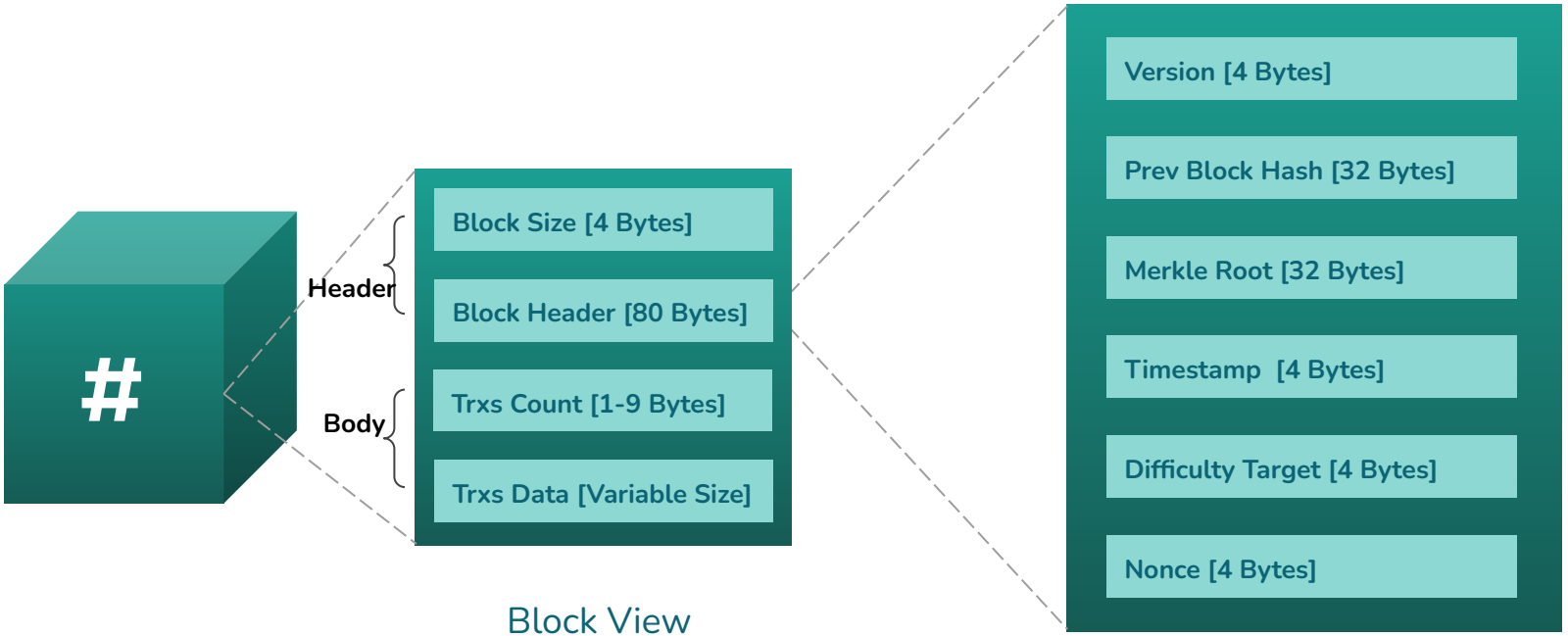


# Structure of Blockchain




- Each block consists of valid transactions and metadata related to the block.
- Each block references the hash of the previous, or “parent,” block
- The sequence of hashes linking each block to its parent creates a chain
- First Block in the chain is called as Genesis Block.

# Contents of a Block




Block View

Max Bitcoin Block Size: 1MB

Block Header

Block Header Size: 80 Bytes

# Contents of a Block

**Block Header**

1 Merkle root, timestamp, previous blockhash, version, bits and nonce

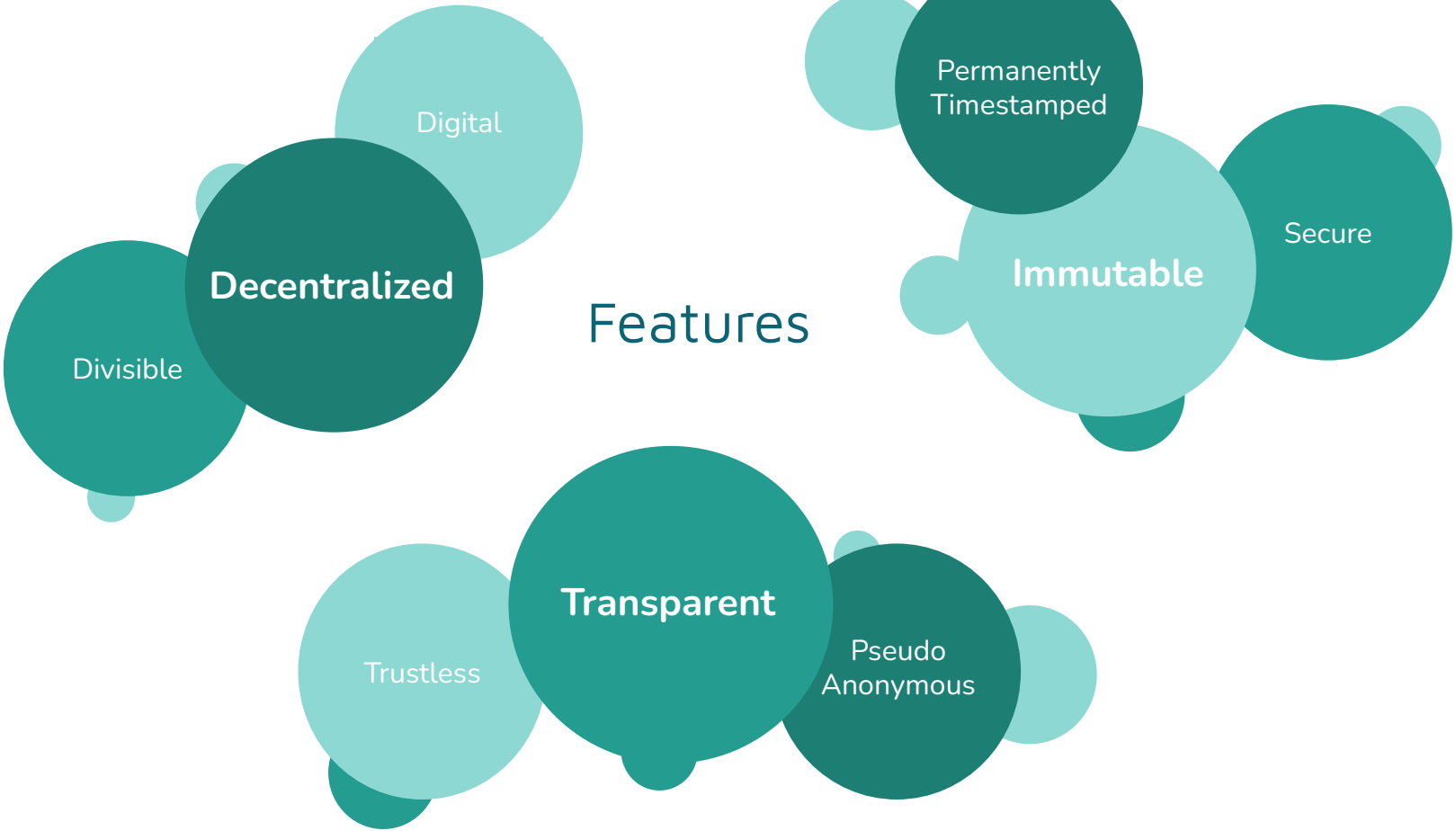
**Transaction Data**

{from: Alice, to: Bob amount: 20BTC} 2

**Block Identifiers**

3 Block Hash, Block Height etc.

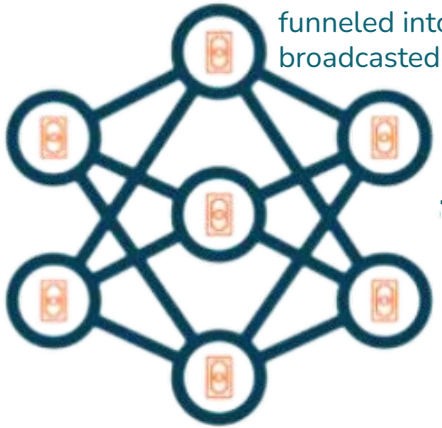
# Features of a Blockchain



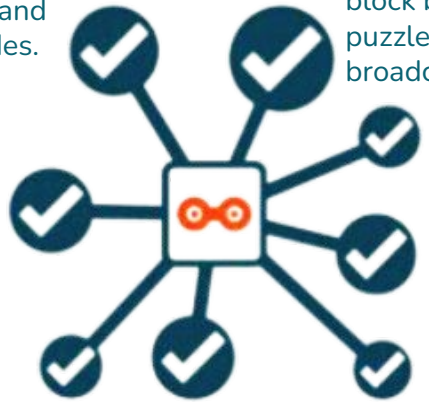
# How does Blockchain Work?

One party initiates a transaction

Initiated transactions are validated & funneled into a P2P network and broadcasted to individual nodes.



Miner nodes include the transactions into a block. They compete to create valid block by solving a crypto puzzle. Once solved, miners broadcast the valid block.



Individual nodes verify puzzle solution and the block is added to a public ledger

Once a block is added to an existing chain, transactions are complete and permanent.

Image Credits: G2 Crowd

# Blockchain is a Book

{ from: Alice to: Bob value: 10BTC }

Single Line on a Page == Transaction

Block Header for #10

{ from: Alice to: Bob value: 10BTC }

{ from: Bob to: Alice value: 12BTC }

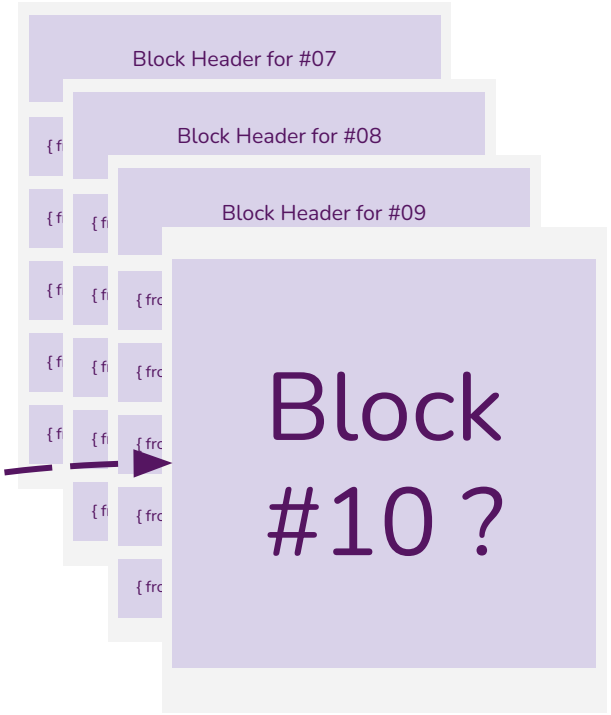
{ from: Carol to: Dan value: 18 BTC}

{ from: Dan to: Bob value: 16BTC }

{ from: Alice to: Bob value: 15BTC }

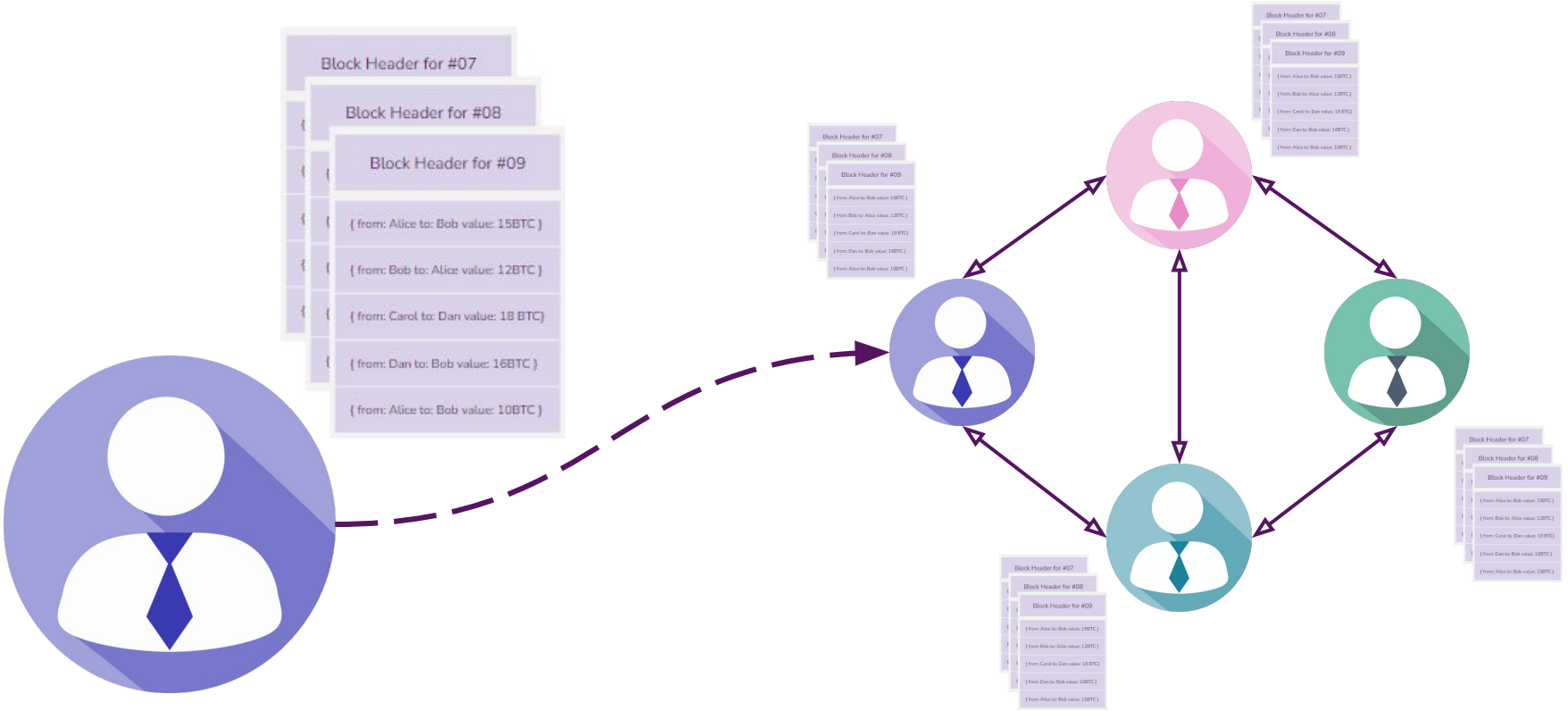
Approved

A Page in a Book == Block




Blockchain == Book

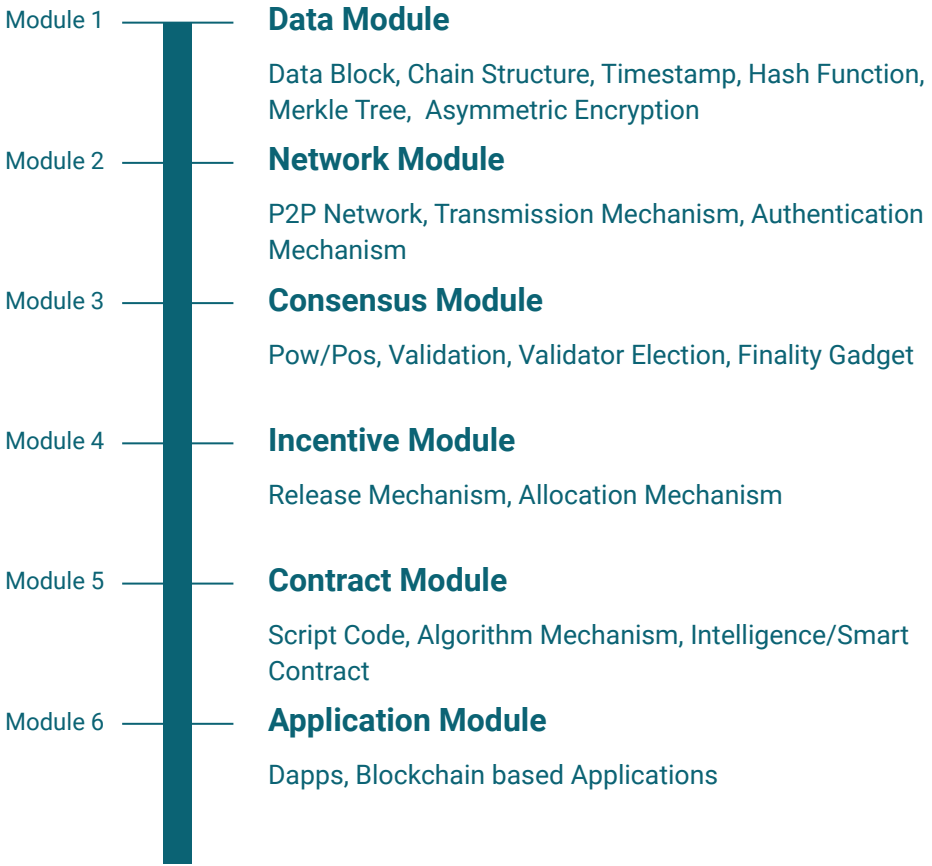
# Blockchain is a Book




Student == Node

Class Room == Blockchain Network

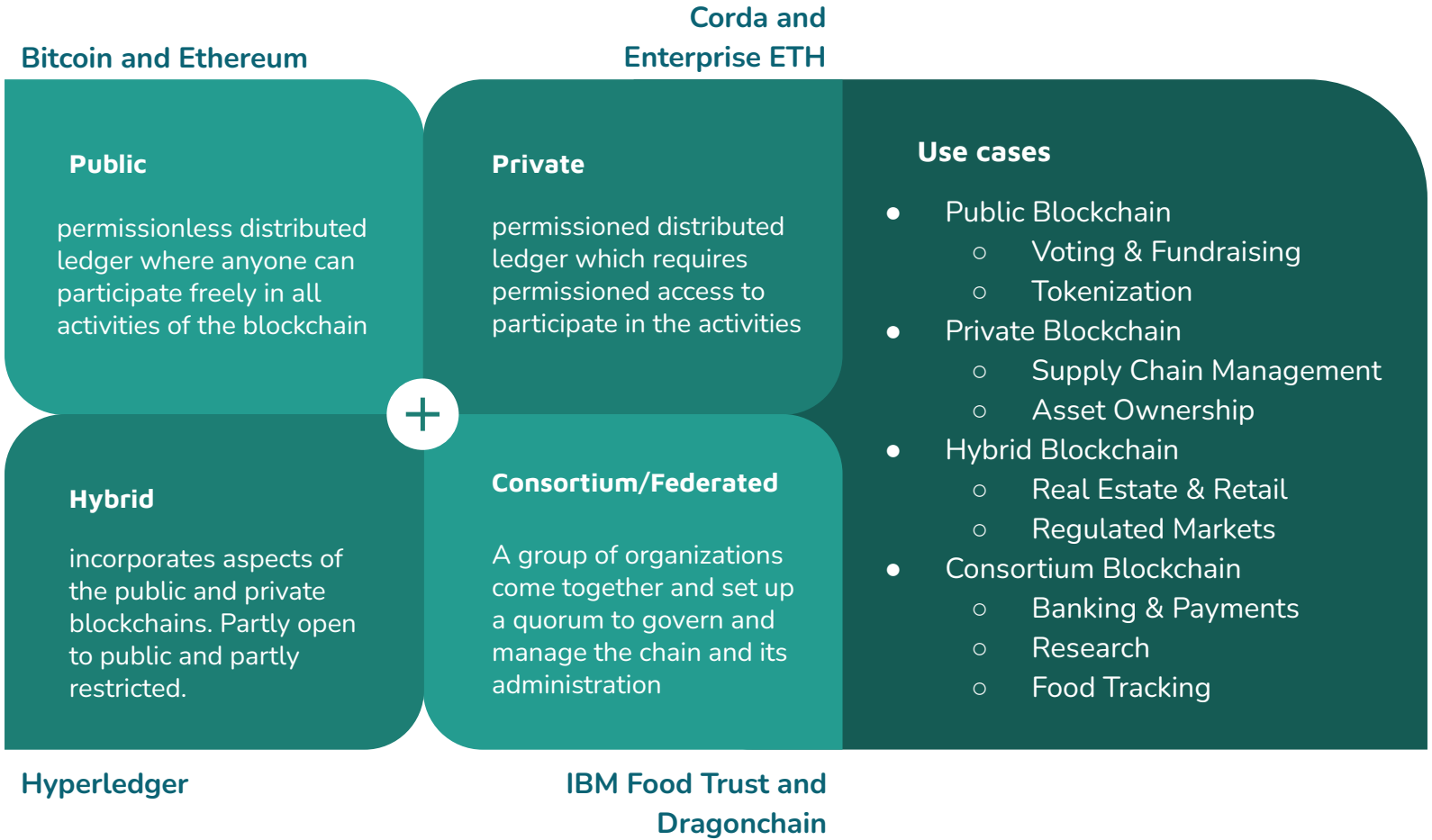
# Breakdown of Blockchain Architecture



# Actors in a Blockchain

| | | |
|---|---|---|
| 01 | Validator | Provides computation, security and storage for the network in return for rewards from the protocol |
| 02 | Developer | Builds profitable applications, powered by the underlying infrastructure of the protocol |
| 03 | User | Users of application and platform derive value by their interactions. |
| 04 | Token Holder | Holders of native token of the Blockchain |
| 05 | Protocol Governance Body | Responsible for governance and development of the platform |

# Types of Blockchains



# Types of Blockchains

Bitcoin and Ethereum

**Public**

permissionless distributed ledger where anyone can participate freely in all activities of the blockchain

Corda and Enterprise ETH

**Private**

permissioned distributed ledger which requires permissioned access to participate in the activities

**Hybrid**

incorporates aspects of the public and private blockchains. Partly open to public and partly restricted.

Hyperledger

**Consortium/Federated**

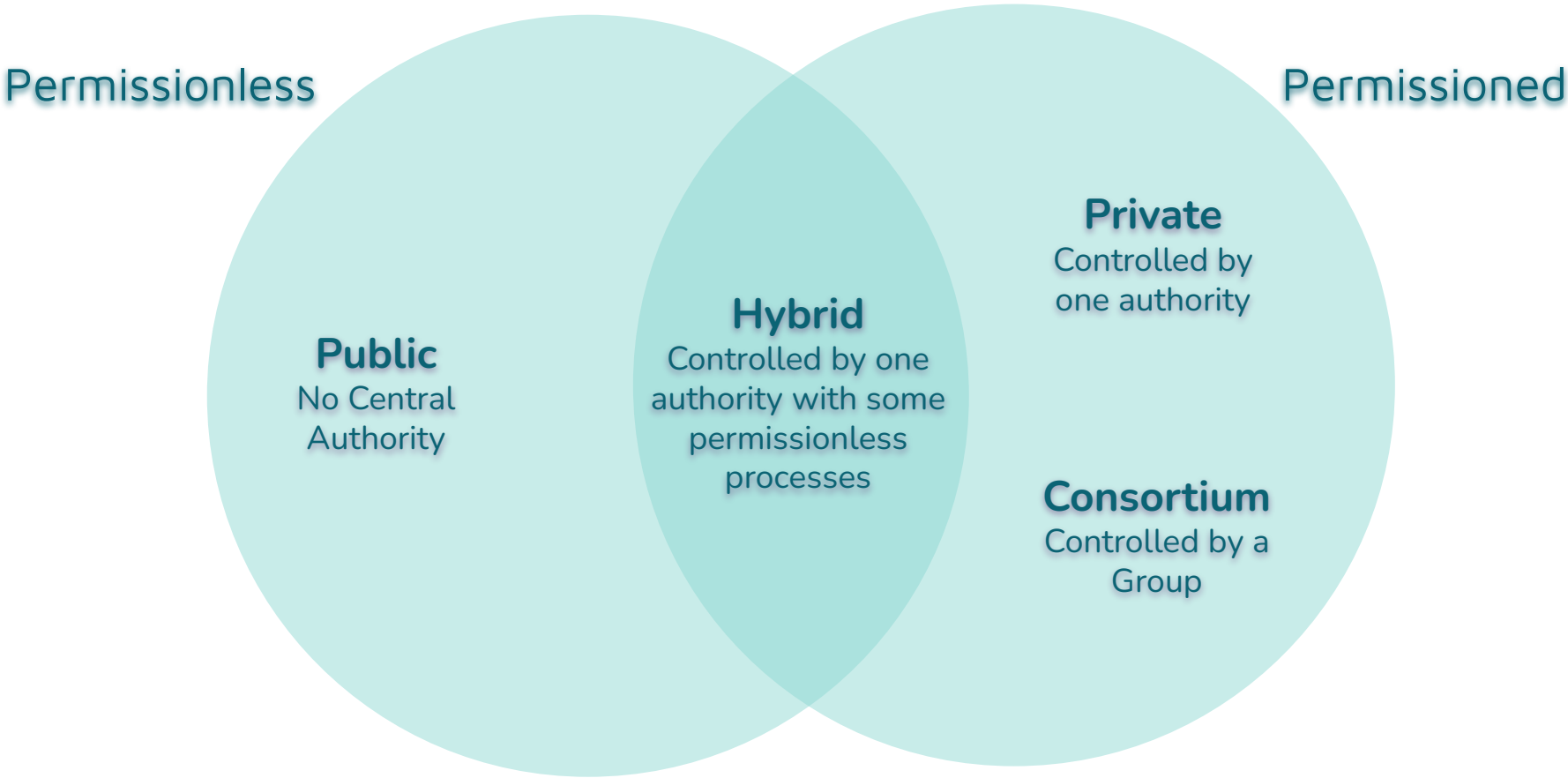
A group of organizations come together and set up a quorum to govern and manage the chain and its administration

IBM Food Trust and Dragonchain

**Use cases**

- Public Blockchain
  - Voting & Fundraising
  - Tokenization
- Private Blockchain
  - Supply Chain Management
  - Asset Ownership
- Hybrid Blockchain
  - Real Estate & Retail
  - Regulated Markets
- Consortium Blockchain
  - Banking & Payments
  - Research
  - Food Tracking

# Types of Blockchain



# Types of Blockchains

**Public**

permissionless distributed ledger on which anyone can participate freely in all activities of the blockchain

**Bitcoin**

**Ethereum**

| | Public | Private | Hybrid | Consortium |
|---|---|---|---|---|
| Advantages | Trustless, Transparent & Secure | Control based on limited access, speed | Control based on limited access, closed system, speed & scalability | Control based on limited access, speed, scalability, security |
| Disadvantages | Low TPS, Low Scalability | Manufactured Trust, High Centrality | Low Transparency & upgradability | Transparency & upgradability |

# Attacks on Blockchains

Blockchain isn't as secure as we tend to think. Though security is an integrated part of blockchains, even the strongest blockchains come under attack by modern cybercriminals.

## P2P Network Attacks

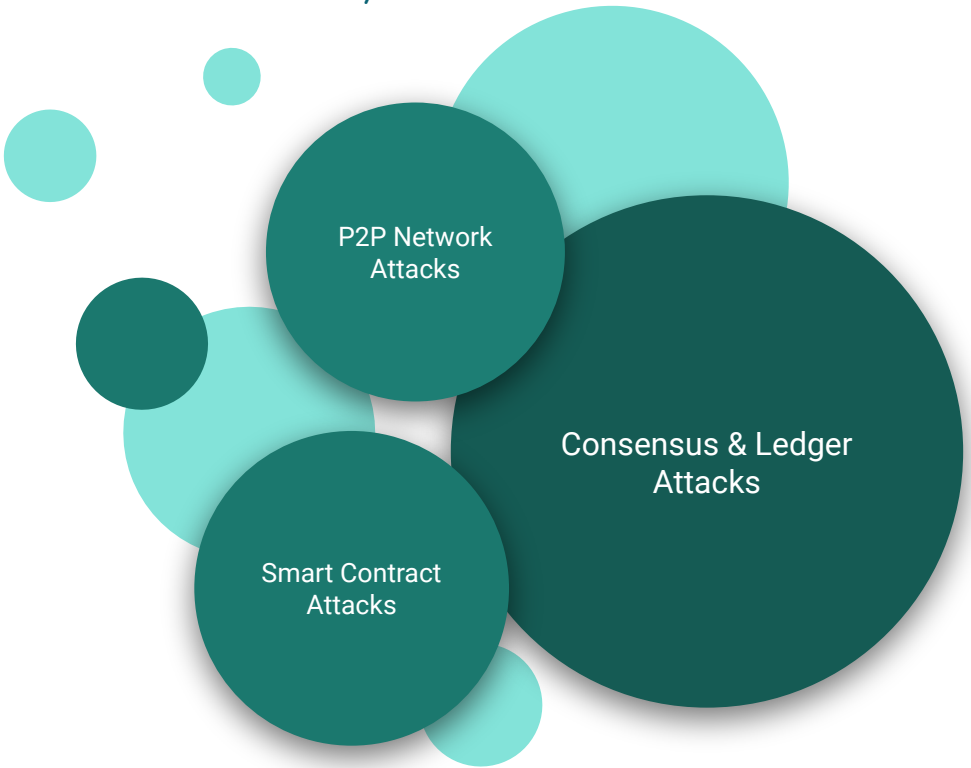
- **Eclipse Attack**
- Sybil Attack
- DDOS Attack

## Smart Contract Attacks

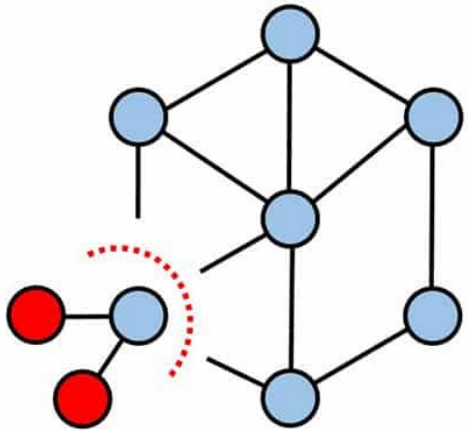
- **Parity Multi-Sig Wallet Hack**
- **DAO Hack**

## Consensus & Ledger Attacks

- **51% Attack**
- Race Attack
- Time Jacking Attack
- Mining Malware



# Eclipse Attack



An eclipse attack is an attack that a malicious actor may deploy to interfere with nodes on a network. The attack aims to obscure a **single node**'s view of the peer-to-peer network, in order to cause general disruption or to prepare for more sophisticated attacks.

This is achieved by taking over all the peer connections to the node. The unsuspecting victim has no view of the wider network and they can be fed incorrect data by the attacker.

**Consequences**

- 0-confirmation double spends
- N-confirmation double spends
- Weakening competing miners

**Countermeasures**

- Increase Outgoing Connections
- Deterministic Random Eviction and others

P2P Network Based Attacks

# Parity Multi-Sig Wallet Hack

In 2017, a vulnerability was found on the Parity Multisig Wallet version 1.5+. Multi-Sig wallets are similar to joint accounts where multiple user control the funds in the wallet.

Hacker exploited the vulnerability by

- Gaining Ownership of the Wallets Created
- Transferring out all the funds from the wallets.

**Consequences**

- Over 150,000 ETH was stolen by the attacker.
- The second largest hack, in terms of ETH stolen
- A total of 550+ Wallets were affected in this hack.

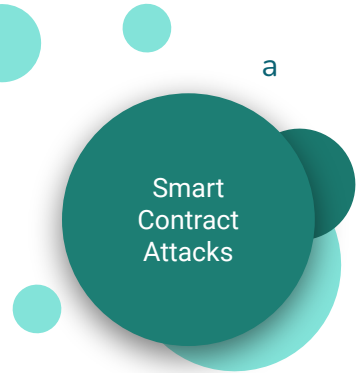
# The DAO Hack [2016]

## The DAO

- An early decentralized autonomous organization intended to act as an investor-directed venture capital firm.
- It was one of the earliest crowdfunding efforts and high-profile projects built on the Ethereum.
- In 3 month of its launch, raised $150 million USD worth of ether (ETH) from over 11,000 investors

## The Hack

- The Attackers found a re-entrancy vulnerability in the contract and exploited it to siphon all the funds from the contract.
- The DAO team and Ethereum Community had 28 days to act, before the attacker could transfer all the funds to his personal wallet.
- After failed Soft Forks Attempt, Ethereum Foundation with the help of miners implemented a hard fork and created new Ethereum blockchain.
- Displeased community continued to mine on the older chain, which later became ETH Classic.
- Though the funds stolen from The DAO were restored to its investors, the attacker did not lose out entirely.



# 51% Attack

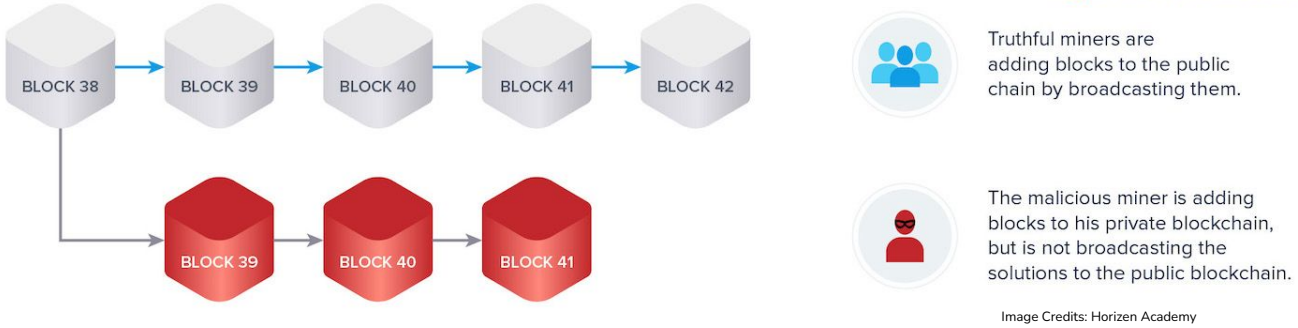


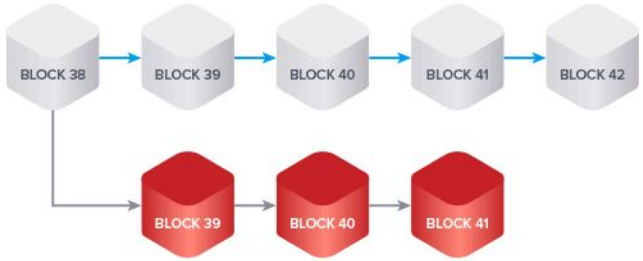

Image Credits: Horizen Academy

A 51% attack (also referred to as a majority attack or a double-spend attack) is a potential attack on blockchains that are created through the process of mining.

- A Malicious actor with the majority of the mining power could enforce their version of the blockchain and reverse completed transactions in order to spend the same coins twice.
- 51% attacks is a greater threat for newer blockchain networks that are still small in size
- The costs of launching the attack would far outweigh any profits made.

Consensus & Ledger Based Attacks

# 51% Attack



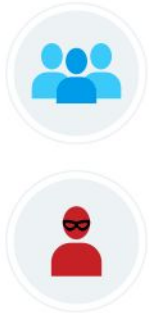



Image Credits: Horizen Academy

## Steps to perform 51% Attack

1. Purchases loads of mining hardware.
2. Joins the Network as a Miner.
3. Spends his tokens on the chain built by valid miners.
4. Omits these spending transactions and mine blocks selfishly.
5. Attacker does not broadcast the blocks created yet.
6. Once the selfish mined blockchain has more blocks than the valid miner chain, attacker broadcasts the selfish mined chain.
7. Due to longest chain rule, everyone will accept the blocks created by the attacker.
8. Blocks built by valid miners will be orphaned.

## Summary

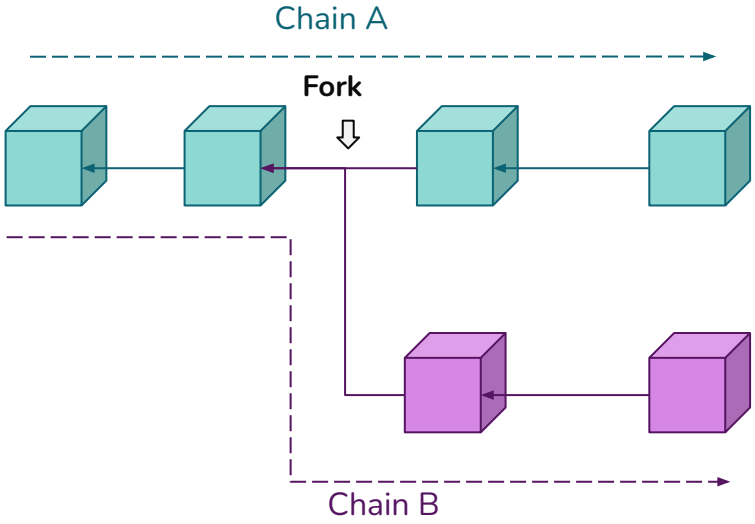
- Gains for Attack: Double Spending his tokens.
- Cost of Attack:  Billions of Dollar for a blockchain like Bitcoin. Doesn't gain access to other users coins.

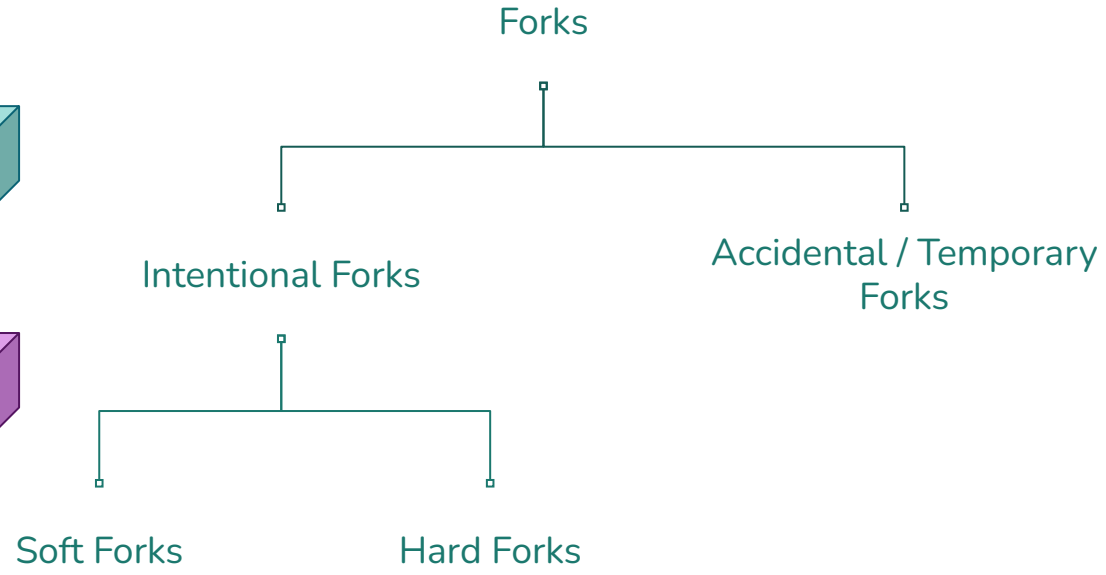
Consensus & Ledger Based Attacks

# Forks

A fork is a technical phenomenon that occurs when a blockchain splits into separate branches. These branches share their transaction history up until the point of the split.

Fork is a blockchain split that produces two competing branches.



# Soft Forks

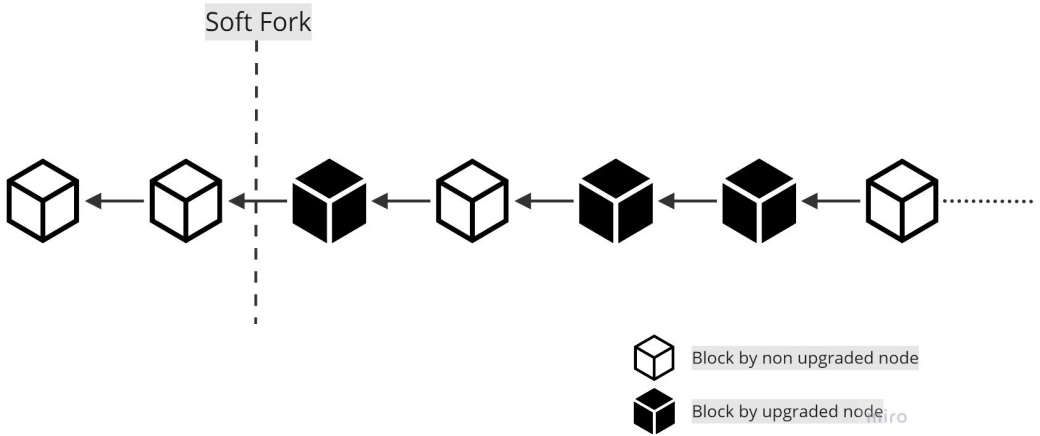
A soft fork is a software upgrade that is meant for the whole blockchain network but is not mandatory for all the nodes to upgrade to this latest software.

- The blocks created by the upgraded nodes and the non-upgraded nodes are compatible with each other. There is no conflict of ideology here.
- End result is a single blockchain, the changes made here are backward compatible.
- Used to add new features to existing blockchains.

## Taproot

Taproot is an agreed soft fork in the transaction format of Bitcoin.

The fork adds support for Schnorr signatures, and improves functionality of smart contracts and the Lightning Network. This upgrade also adds privacy features to Bitcoin Blockchain.



# Hard Forks

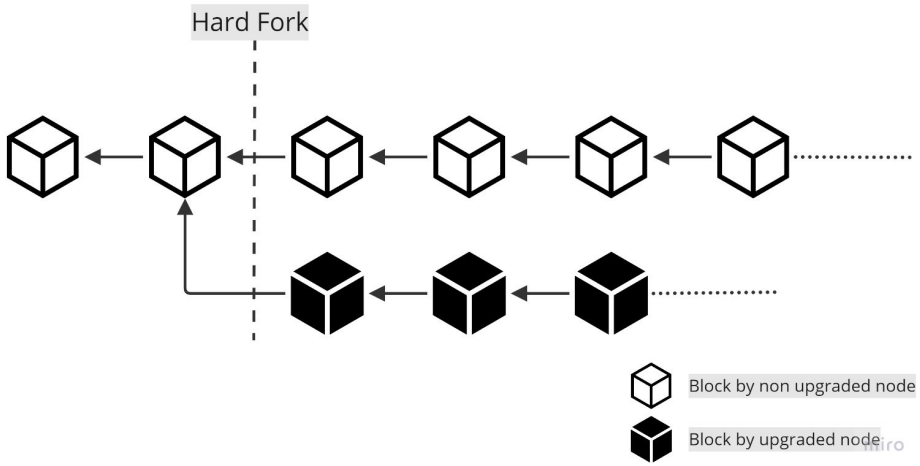
A hard fork is a software upgrade that is meant for the whole blockchain network and **must** for all the nodes to upgrade to this latest software because the blocks created by the upgraded nodes would be incompatible with the block created by the non-upgraded nodes.

- This results in a fork-like structure. If continued it results in the formation of a new chain altogether.
- Since there is a conflict in the acceptance of the new upgrade the creation of the new blockchain is evident.

## ETH Proof of Work [ETHW]

In Sep, 2022, the Paris upgrade of Ethereum was triggered to switch from PoW mining algorithm and associated consensus logic to PoS instead.

A small portion of dissatisfied miners continued without upgrading, creating the Ethereum Proof Of Work Chain [ETHW].



# Accidental or Temporary Forks

**Scenario**
A node receives two valid blocks from two different miners for the same block height.

How does the node know which one to use and which one to throw away?
How does the network come to consensus about which block to use?

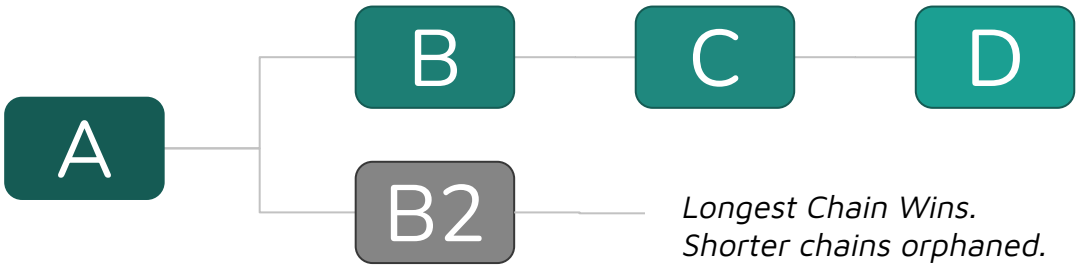


- Nodes can pick either block (usually the first seen) and would store the other block for reference.
- Similarly, other nodes will also make their decisions.
- Eventually, one of the blocks will have another block mined on it.
- This leads to a temporary fork of the blockchain existing in the same network.

How to resolve this temporary fork and maintain single chain in the network?

# Longest Chain Rule

The chain with highest number of blocks should be considered for chain of record, others should be discarded (considered orphan).

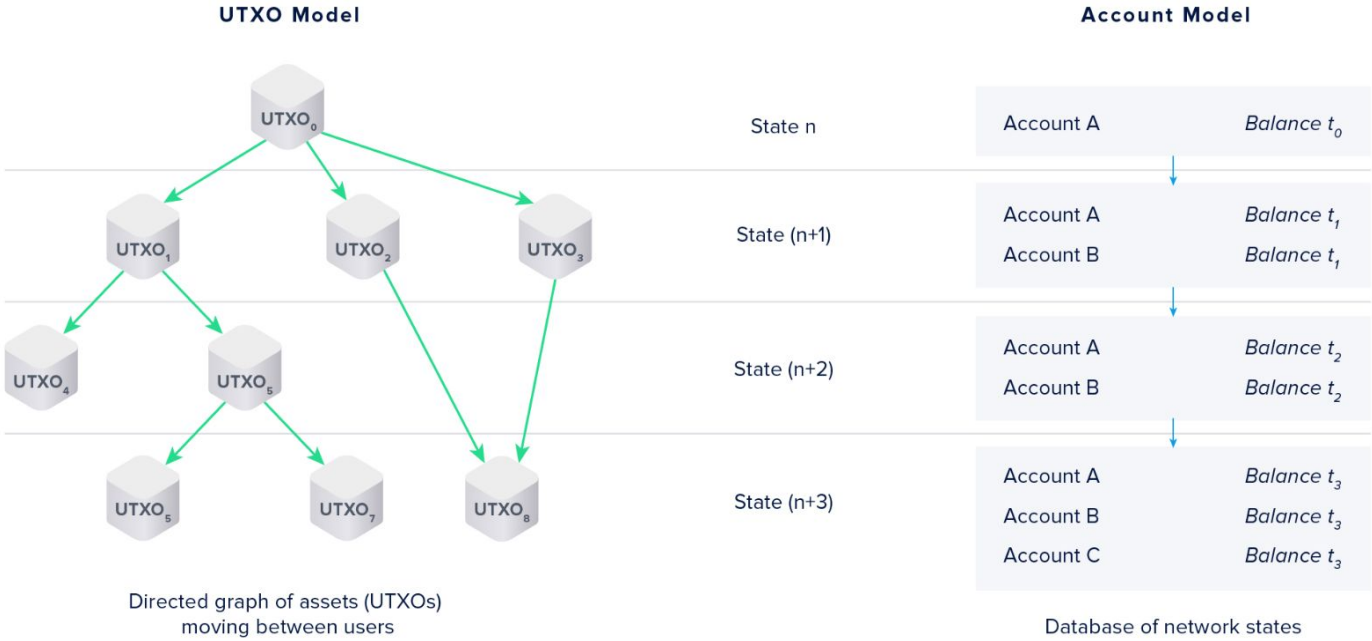



What happens to the transactions in the orphaned block?

- They are considered as if they have never been part of a valid block & therefore are ‘unconfirmed’.
- They will just be included in later blocks along with other unconfirmed transactions.

# UTXO Model vs Account Model

## Maintaining Record of Balances in the State of the Blockchain




Cash Transactions are similar to UTXOs

Banking Transactions follow Account based Model

# Finality Gadget

Finality is the assurance that a cryptocurrency transaction cannot be altered, reversed, or canceled after it is marked as completed.

- Used to measure the amount of time one has to wait for the transactions to be stored permanently
- Latency of the network and the consensus mechanism are the key contributors
- PoW chains finality takes longer.
- The finality in a bitcoin transaction is 6 confirmations.

| Blockchain | Time to Finality |
|---|---|
| Bitcoin | ~60 Minutes |
| Ethereum | ~13-15 Minutes |
| Polkadot | ~6 Seconds |
| Solana | 1-1.15 Seconds |
| Aptos | 0.9 Seconds |

# Difficulty Adjustment

Every time a block is mined, New tokens are rewarded to the miner.



Every time a block is mined, New tokens are rewarded to the miner.



Miners increase processing capacity to find nonce faster, thus mining more blocks in less time.



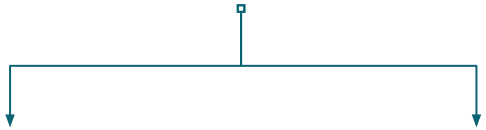
This could lead to difficulty in achieving consensus in the networks due to multiple forks being created.



This also leads to increase in supply of new tokens quickly, diminishing the value of each token.

**How to maintain a steady supply of new tokens in this scenario?**

Avg Block Time of last 2016 Blocks [~ 2 Weeks]



Greater than 10 Mins



Increase Difficulty

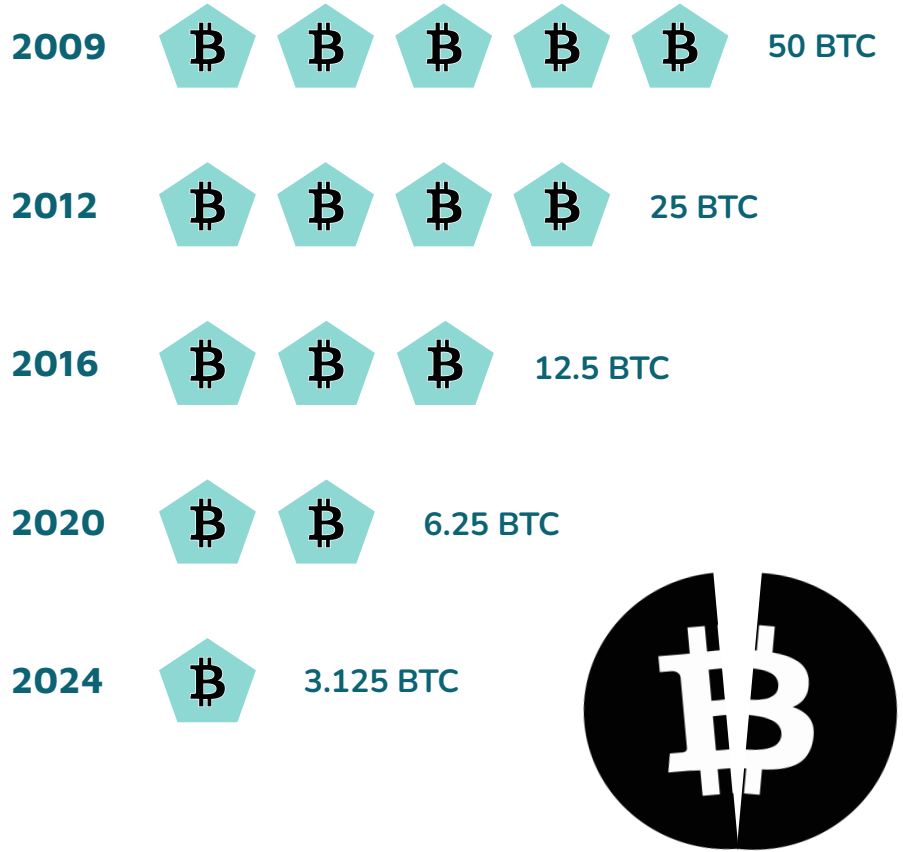
Lesser Than 10 Mins



Decrease Difficult

# Bitcoin Halving

- Every 210,000 blocks, the bitcoin issuance is cut in half.
- The halving decreases the amount of new bitcoins generated per block.
- Halvings are at the core of the cryptocurrency economic models
- Ensures coins will be issued at a steady pace, following a predictable decaying rate.
- Controlled rate of monetary inflation
- Deflation in bitcoin is not caused by a collapse in demand, but by a predictably constrained supply.
- Inflation causes a slow but inevitable debasement of currency, resulting in a form of hidden taxation.



# Consensus in Blockchain

# How are Blockchains Secured?

- ➔ In **Permissioned and Private blockchains** the controlling entity will usually organize a vote.
- ➔ **Public blockchains** have found a method to stay secure in a more decentralized way: by using a consensus mechanism.

- Blockchains store their information distributed across the whole network in nodes.
- Nodes must verify each new transaction before it can be added to the blockchain.
- meticulous process to add new transaction to the chain, since they all need to stay in sync.Otherwise they'd end up with conflicting records.
- Once the block is added, every node in a blockchain must arrive at the same state.
- Each transaction added to the chain could represent practically anything, but generally represents a transfer of value.

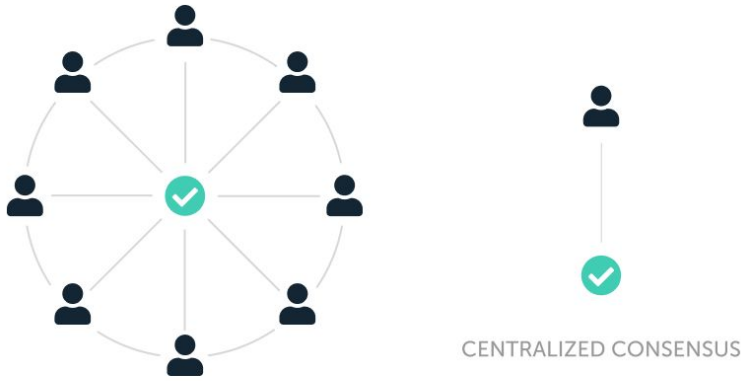



Image Credits: Ledger Academy

# Consensus

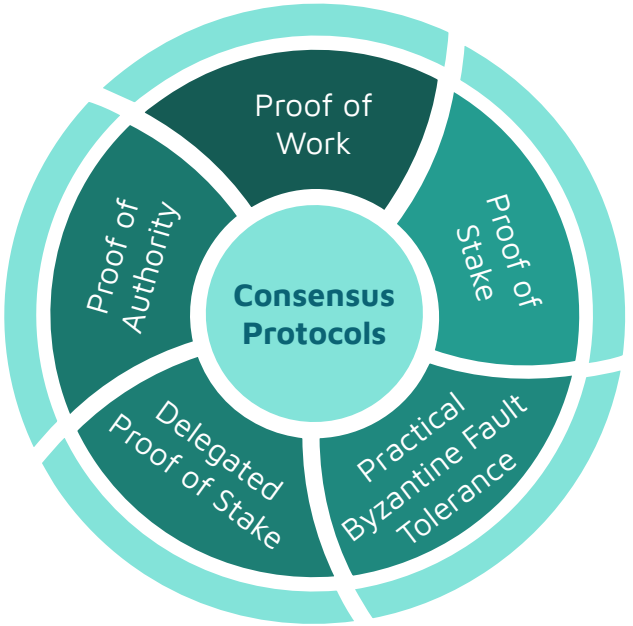
How participants of a Network come to agree on the public record.

# Consensus Protocols and the need for them

Set of rules that govern how two or more nodes interact, exchange information, and perform associated processes to reach a common acceptance about the real-time state of the distributed ledger.

## Need for Consensus Protocols

- Unified agreement
- Prevent double-spending
- Align Economic Incentive
- Fair and equitable
- Reliable



# Proof of Work [PoW]

**Cryptographic Nonce** is an arbitrary integer that can only be used once.

Step 1: Compile some data to input to a calculation

Hash of last block in existing blockchain + Block of New Transactions + 0 2 3 7 5 Nonce

Input: Data from Step 1

Step 2: Perform Cryptographic Calculation (SHA-256)

Output: Hash Value

Step 3: Hash < Target Hash ?

No

Try Another Random Number. Repeat Step 1-3.

Yes

Valid Block Created. Puzzle Solved!

**Target Hash** is a threshold determined by mining difficulty that the hash must be less than for a proposed block to be added to a chain.

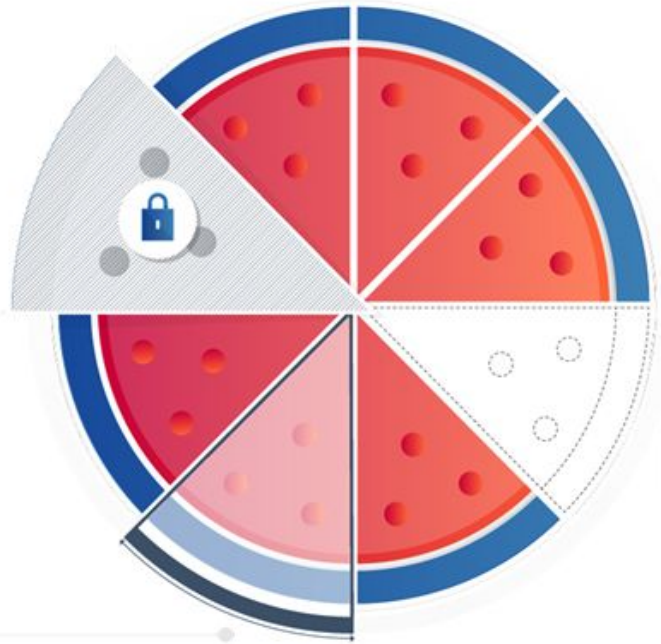
New **Bitcoins** are distributed by the network to miners for each solved block.

# Proof of Stake [PoS]

**1** **Anyone who holds the base cryptocurrency can become a validator,** although sometimes a locked up deposit is required.

**2** A validators chance of mining a block is based **on how much of a stake (or cryptocurrency) they have.**

*For example, if you owned 1% of the cryptocurrency, you would be able to mine 1% of all its transactions.*



**3** The PoS protocol will randomly assign the right to create a block in between selected validators, based upon the value of their stakes.

**The chosen validator is rewarded by a part or the whole of the transaction fee.**

Image Credits: Lisk Academy

# Proof of Work [Pow] vs Proof of Stake [PoS]

| Proof of Work | Proof of Stake |
| --- | --- |
| The first miner who solves the crypto puzzle is selected. Competition between miners to solve the puzzle. | Using deterministic selection process. Competition between miners to be selected. |
| Specialized equipment to optimize processing power. | Standard server grade unit is usually(more than) enough |
| Initial investment to buy the hardware. | Initial investment to buy the stake and build the reputation. |
| High Energy Consumption | Standard energy consumption |

# Crypto Wallets

Crypto wallets store users' public and private keys, while providing an easy-to-use interface to manage crypto balances.

- Support cryptocurrency transfers
- Some wallets even allow buying and selling or interacting with decentralised applications (dapps).
- Every transaction is signed with user's private key and broadcasted
- Network then includes the transaction to reflect the updated balance in both the sender's and recipient's address.

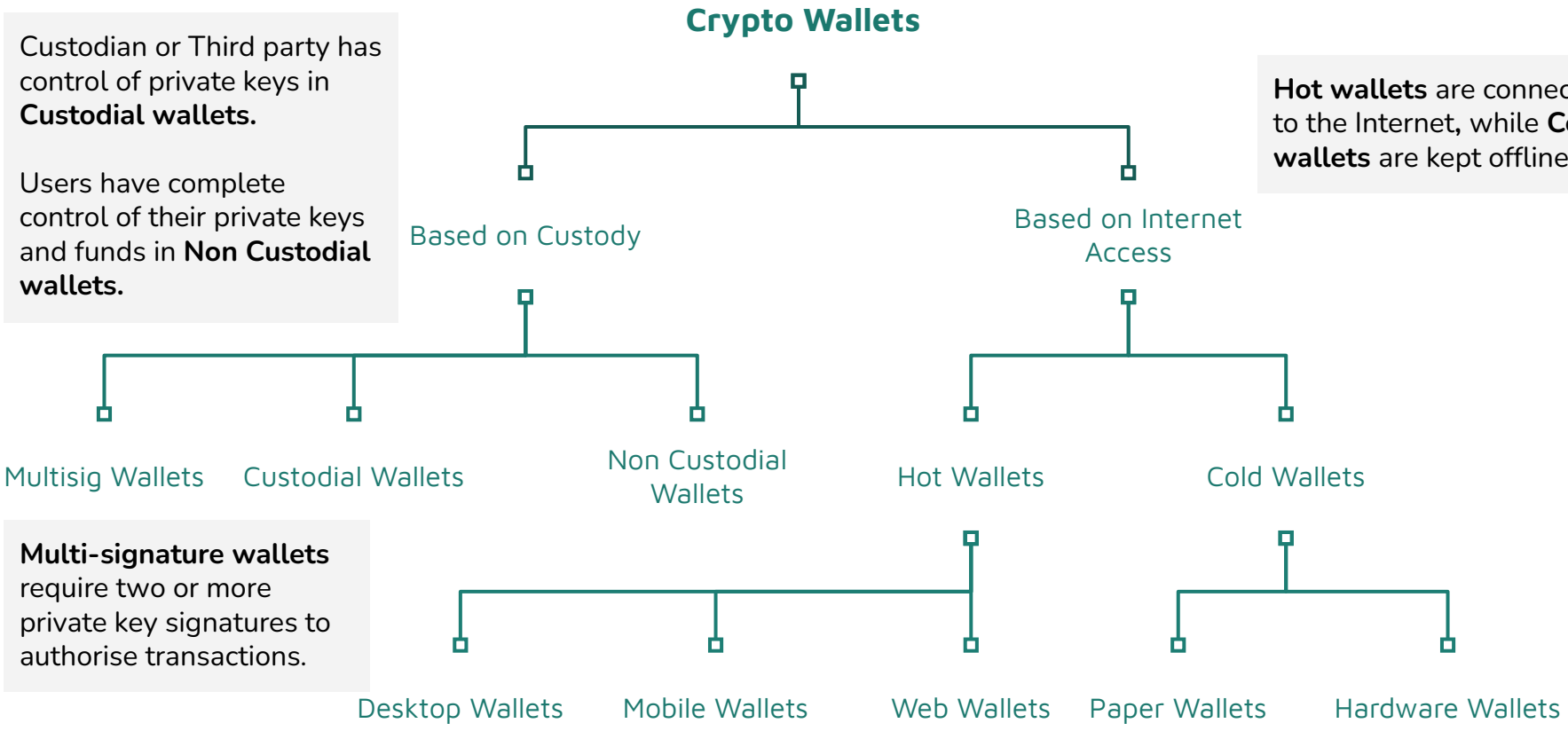
**Note:** The term 'wallet' is somewhat of a misnomer, as crypto wallets don't actually store cryptocurrency in the same way physical wallets hold cash.

Instead, they read the public ledger to show the balances in a user's addresses, as well as hold the private keys that enable the user to make transactions.

# Types of Crypto Wallets

Custodian or Third party has control of private keys in **Custodial wallets.**

Users have complete control of their private keys and funds in **Non Custodial wallets.**




**Hot wallets** are connected to the Internet, while **Cold wallets** are kept offline.

**Multi-signature wallets** require two or more private key signatures to authorise transactions.

## Desktop Wallets

keep a user's private keys securely stored on their computer hard drive.

- Excellent for securely conducting small crypto transactions using a computer
- Free and easy to use
- No third-party holds your private keys
- Some can be used offline for cold storage

Examples: Electrum, Exodus, Bitpay

## Web Wallets

are provided by a third party, which offer seamless access to a user's holdings using a web browser.

- Easy to use;
- generally favored by newcomers
- Support a variety of transactions
- Account security outsourced to trusted third-party

Examples: Metamask, Coinbase, Binance

## Mobile Wallets

Mobile wallets allow users to quickly and securely spend or receive cryptocurrency anywhere they have their phone and an active internet connection.

Easily send or receive crypto payments on the go
Highly convenient and easy to use

Examples: Trust, Edge, Bitpay

## Paper Wallets

A Paper wallet is an offline wallet solution where private keys are written down or printed and securely stored.

- Being completely offline makes it impossible to hack
- No third-party has control of your private keys
- Optional inclusion of QR code allows for easier access

Examples:

WalletGenerator.net
BitcoinPaperWallet.com

## Hardware Wallets

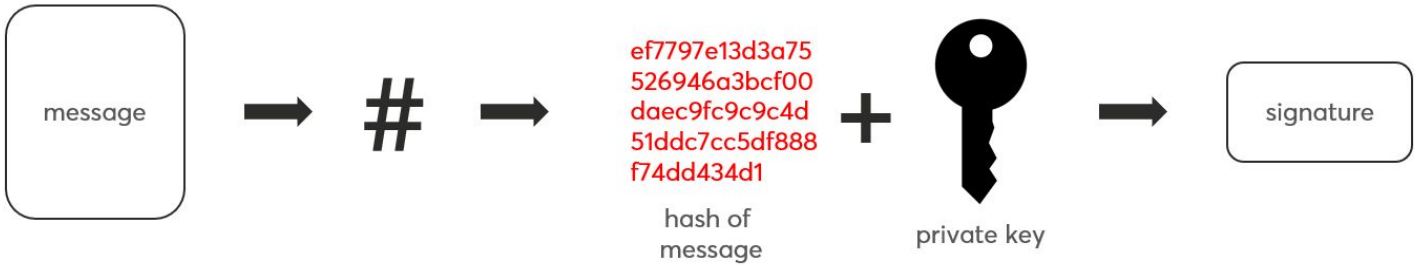
A Hardware wallet offers secure private key storage in a number of formats. These physical devices, often resembling a USB thumbdrive, are offline unless plugged into a computer or mobile device.

- One of the most secure methods of crypto storage
- Transactions are signed using private key offline, and only online to upload the transaction to the blockchain
- Available at most major electronics retailers

Examples: Ledger Nano X, Trezor Model One

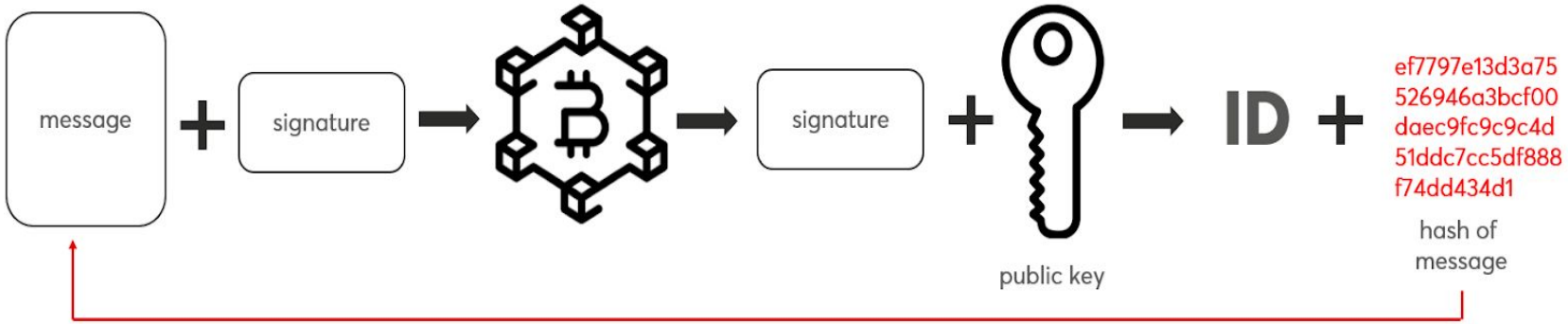
# Transaction Validation

- Wallets are necessary to manage your Transactions.
- Wallets contain
  - Private Key [Lose your private key, lose your money]
  - Public Key [Can be shared publicly]
- Initiating the transaction is similar to sending an email.
- Three Components of a Transaction
  - A message with the details of the transactions (amount and the to-address)
  - Your public Key
  - Your Signature
- Signature is generated as follows



# Transaction Validation

- Signature is decrypted with the public key.
- Reveals the hash of the message[#1] and the sender.
- The message (not the hashed one) is hashed[#2].
- This hash[#2] is compared with the hash that came out of the decryption of the signature[#1].
- If they are the same you know that the message hasn't been altered.
- If the transaction is valid, your funds are checked and when you have enough funds the transaction is added to a pool of transactions.
- Transactions out of the pool will be added to the next block in the blockchain.



# Blockchain Nodes

Nodes are computers that run blockchain software and enable the blockchain to validate transactions and keep the network secure. They communicate with one another and keep the network decentralized.

- When a user sends a transaction, a node receives it and broadcasts it.
- The transaction is checked by nodes
  - ensure that the sender has the funds available
  - is authorized to send them.
- Once transaction is validated,
  - nodes add it to the newest block,
  - later added to the chain.

- Different types of nodes exist in a Blockchain network.
- Nodes help
  - maintain security and data integrity
  - reliability and credibility of the network.
- Developers use these nodes to create Dapps

# Function and Purpose

Blockchain nodes keep a complete copy of the distributed ledger and are responsible for the data's reliability.

Assigning a unique identifier to each node in the network makes it easier to distinguish one node from another.

Basic functions of a node

- Managing transactions and ensuring their validity
- Transaction acceptance or rejection
- Serving as a conduit of communication
- Keeping track of the cryptographically linked blocks

Nodes serve three main purposes

- Maintenance - keep copies of the ledger in sync
- Validation - use consensus mechanisms to ensure all nodes remain in sync
- Accessibility - serve as storage containers of the blockchain

# Types of Nodes

01 **Full Node**

Store copy of the blockchain and maintain consensus among nodes. Ensure correctness & verification of data

**Miner Node** 02

Specifically designed to carry out the mining process, solves cryptographic problem using hardware components

03 **Pruned Full Node**

Downloads entire chain and then deletes older blocks till it holds only the most recent trxs up to size limit.

**Archival Full Node** 04

Full nodes with an archive. useful for querying historical blockchain data; have no set storage capacity

05 **Light Node**

These nodes only downloading block headers to saves download time & storage space.

# Types of Nodes [Contd]

Complete nodes that cannot add new blocks to the chain but only validate and record transactions.

**Master Node** 06

07 **Lightning Node**

Reduce the load on the network by enabling off-chain transactions, enable faster and cheaper transactions.

They are used for signing transactions offline and storing private keys away from the network.

**Staking Node** 08

09 **Authority Node**

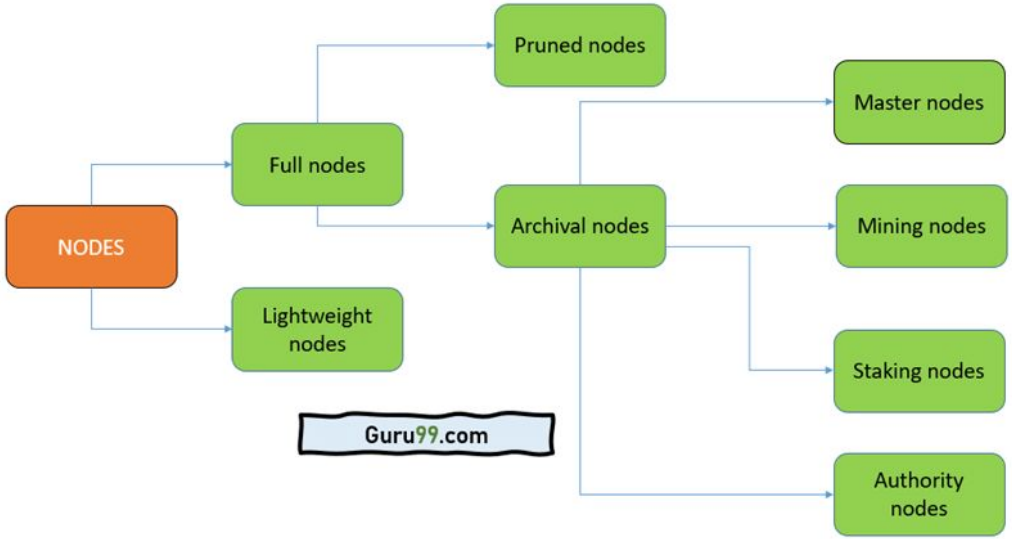
chosen node by the organization that can authorize new nodes to join and also manage other nodes' access permissions.

Perform specialized tasks critical to the operation & maintenance. Ex. enforce network regulations or implement upgrades.

**Super Node** 10

Blockchain Nodes [My Categorization]

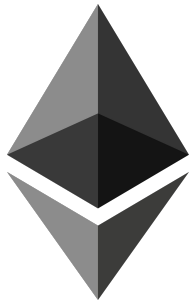
- Lightweight Nodes
  - Client Nodes
  - Cold Nodes
  - Lightning Nodes
- Full Blockchain Nodes
  - Full Node
  - Archival Nodes
    - Staking Nodes
    - Mining Nodes
    - Authority Nodes
  - Pruned Full Nodes
  - Master Nodes
  - Super Nodes(Listening Nodes)



# Ethereum

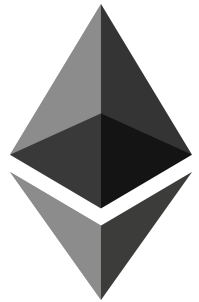
# Ethereum

- Ethereum is a blockchain-based computing platform that enables developers to build and deploy decentralized applications.
- Laid the foundation for many emerging technological advances based on blockchain.
- Creation of decentralized applications using smart contracts.
- Smart Contracts enabled users to securely executes and verifies application code.
- Smart contracts allow participants to transact with each other without a trusted central authority.
- Extremely flexible platform on which to build decentralized applications using the native Solidity scripting language and Ethereum Virtual Machine.
- Rich ecosystem of developer tooling
- Established best practices and inspired creation of smart contract enabled blockchains.



# Features of Ethereum

- Ether
- Smart contracts
- Ethereum Virtual Machine
- Token Creation
- Decentralized applications (Dapps)
- Decentralized autonomous organizations (DAOs)



# Applications of Ethereum

**Voting**

- Blockchain based voting using Smart Contracts allows the results of polls are available publicly, ensuring a transparent fair system.

**Agreements**

- With Ethereum smart contracts, agreements and contracts can be maintained and executed without any alteration.

**Banking systems**

- De-centrality of Ethereum makes it challenging for hackers to gain unauthorized access to the network and make payments secure.

**Shipping**

- Ethereum provides a tracking framework that helps with the tracking of cargo and prevents goods from being misplaced.

**Crowdfunding**

- Applying smart contracts to blockchain-based crowdfunding platforms helps to increase trust and information symmetry.

# Bitcoin vs Ethereum

| Parameters | Bitcoin [BTC] | Ethereum [ETH] |
|---|---|---|
| Purpose | Exchange of decentralized currency | Smart contract development platform |
| Data Blocks | Only Transaction Information | Can hold transaction info and contracts |
| Mining | Proof of Work | Proof of Stake [Since Sep 2022] |
| Block Time | 10 Mins | 12-15 Seconds |
| Transaction Finality | Around 1 hours | Less the 3 minutes |
| Scripting Language | Bitcoin Script (limited functionality) | Solidity (Turing Complete) |
| Hash Function | SHA-256 | Keccak-256 |
| Smart Contracts | Although bitcoin supports it, they are not flexible or complete. | Smart contracts have all the functionality that a programming language would give. |

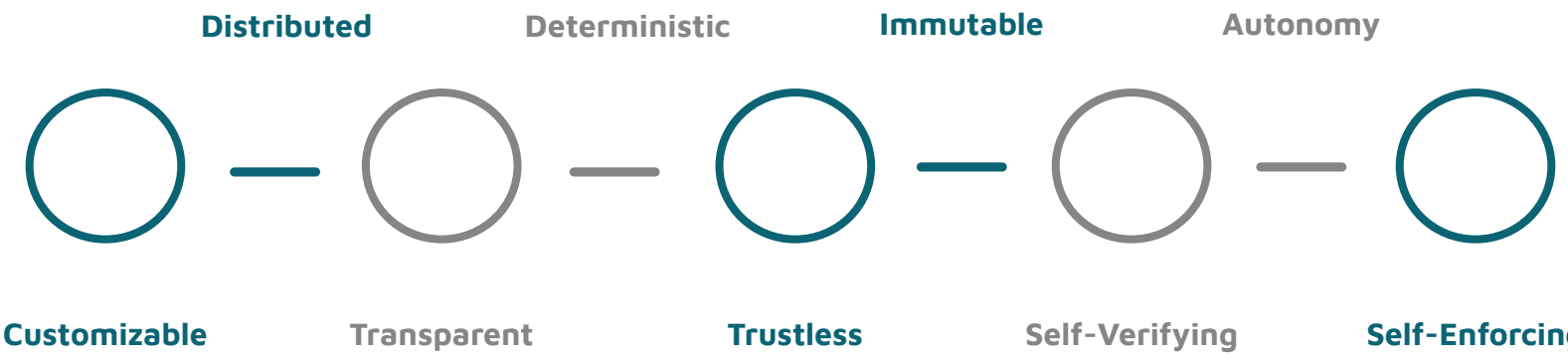
# Smart Contracts

Smart contracts are programs that reside at a specific address on the blockchain and execute exactly as they are programmed by their creators.

Applications can call the smart contract functions, change their state, and initiate transactions.

- Traditional contracts is enforceable by law
- Smart contracts are enforceable by code.

## Features of Smart Contracts



# How does a
# Smart Contract Work?

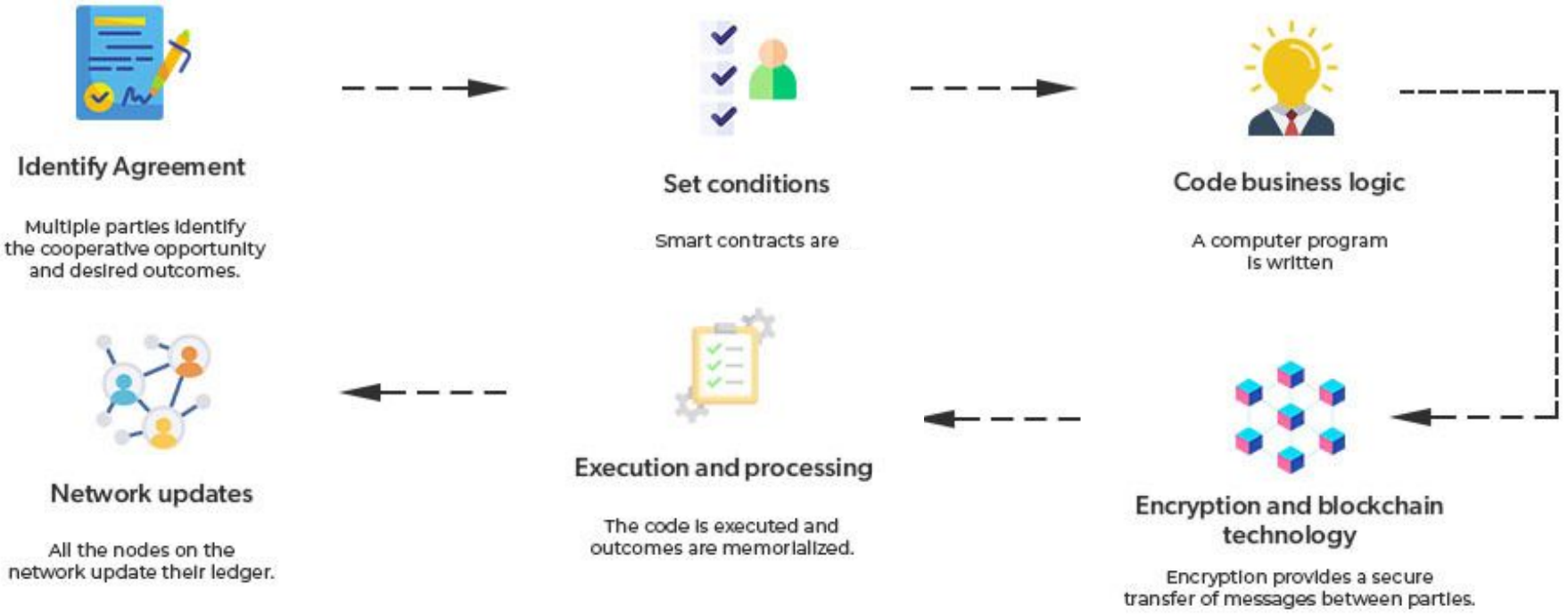
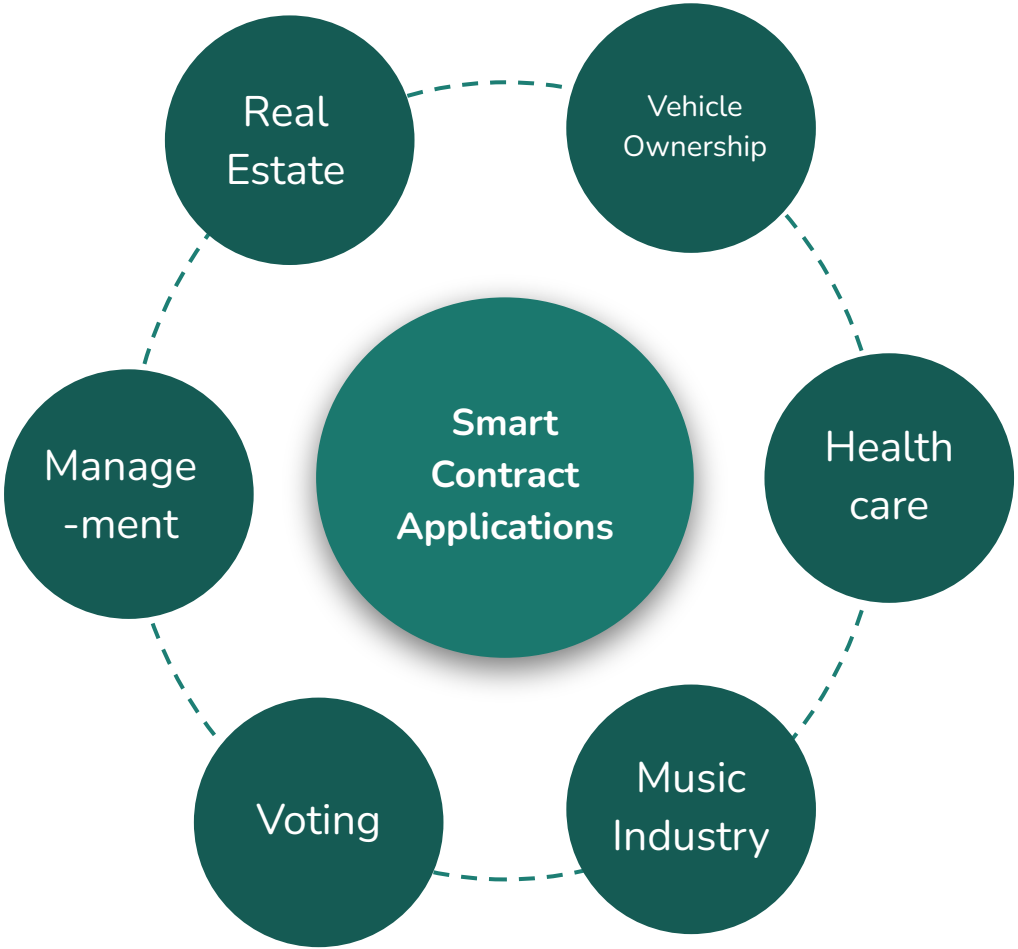


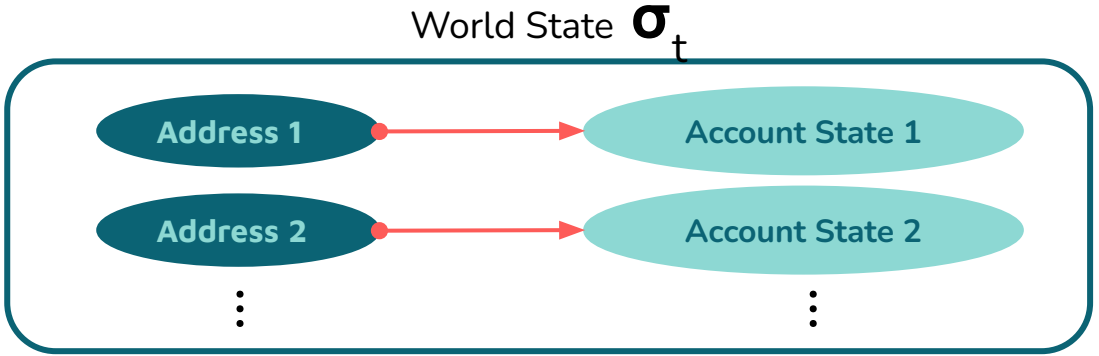

Image Credits: Geek for Geeks

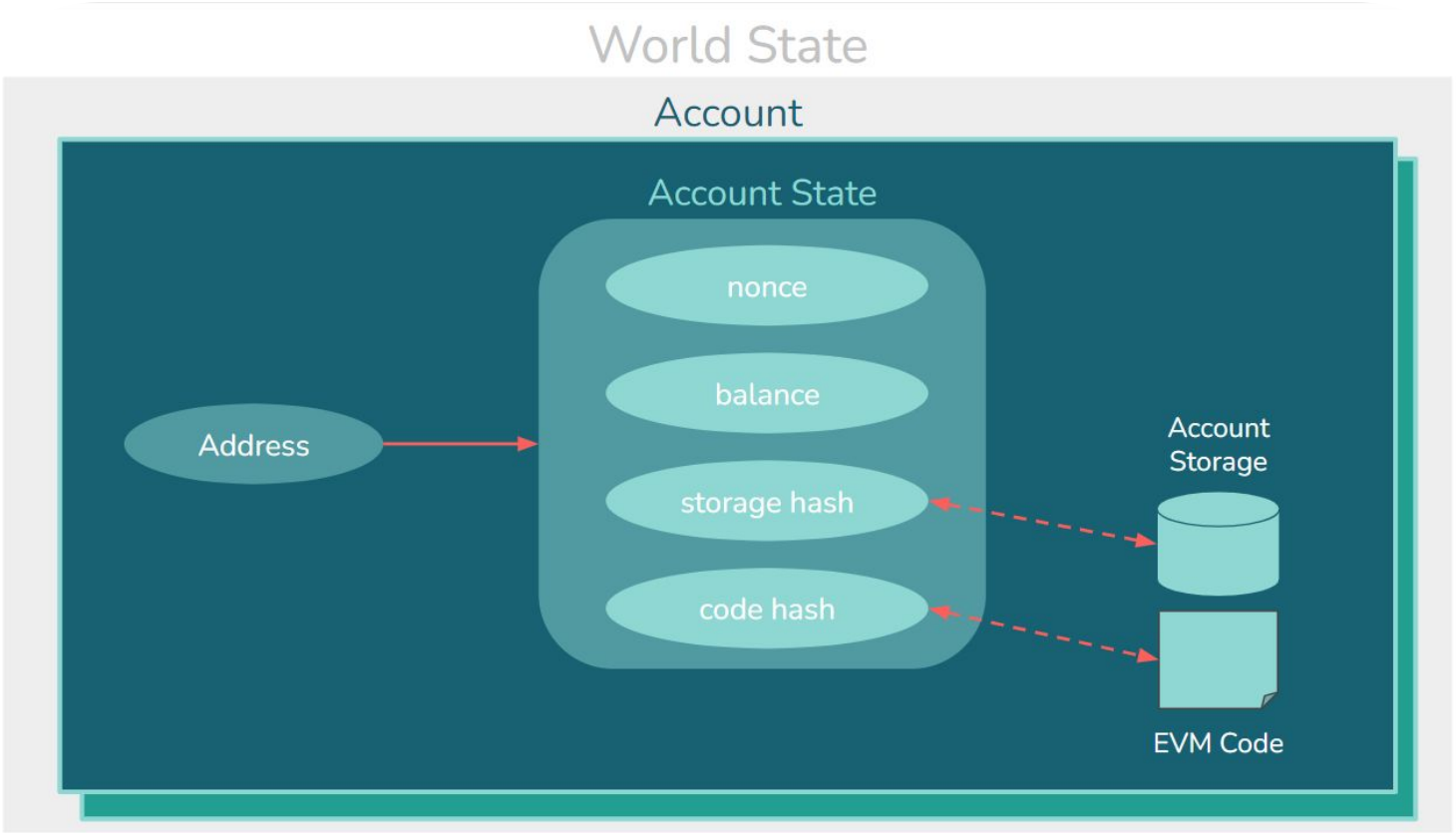

Real Estate
Vehicle Ownership
Manage -ment
Smart Contract Applications
Health care
Voting
Music Industry

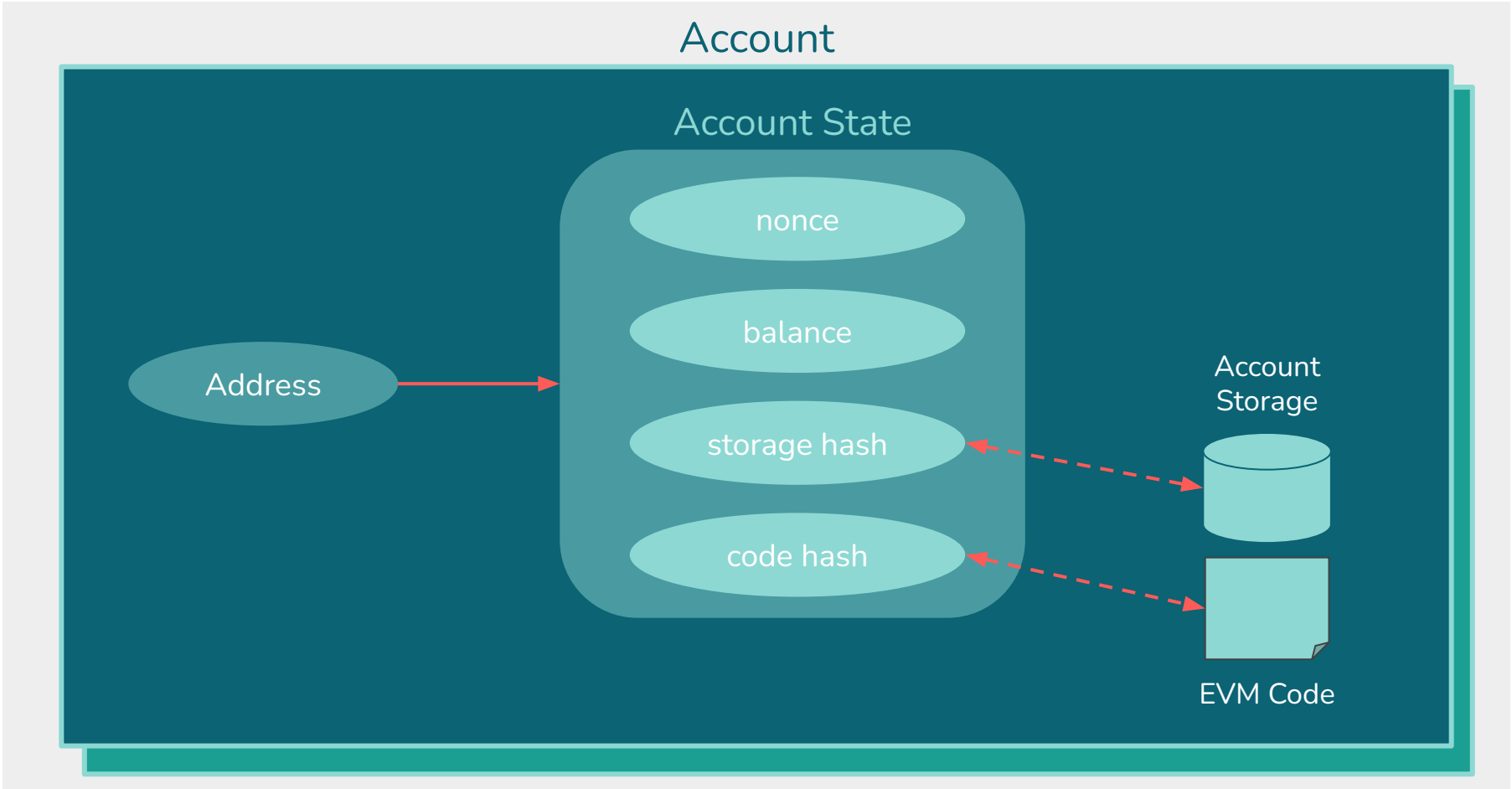
# World State

- The world state is a mapping between addresses (accounts) and account states.
- The world state can be seen as the global state that is constantly updated by transaction executions.
- All the information about Ethereum accounts live in the world state and is stored in the world state trie.
- To know the balance of an account you query the world state trie to retrieve the account state of that account.

**Note:** The world state is not stored on the blockchain but store this data in a state trie.




World State
Account
Account State
nonce
balance
storage hash
code hash
Address
Account
Storage
EVM Code

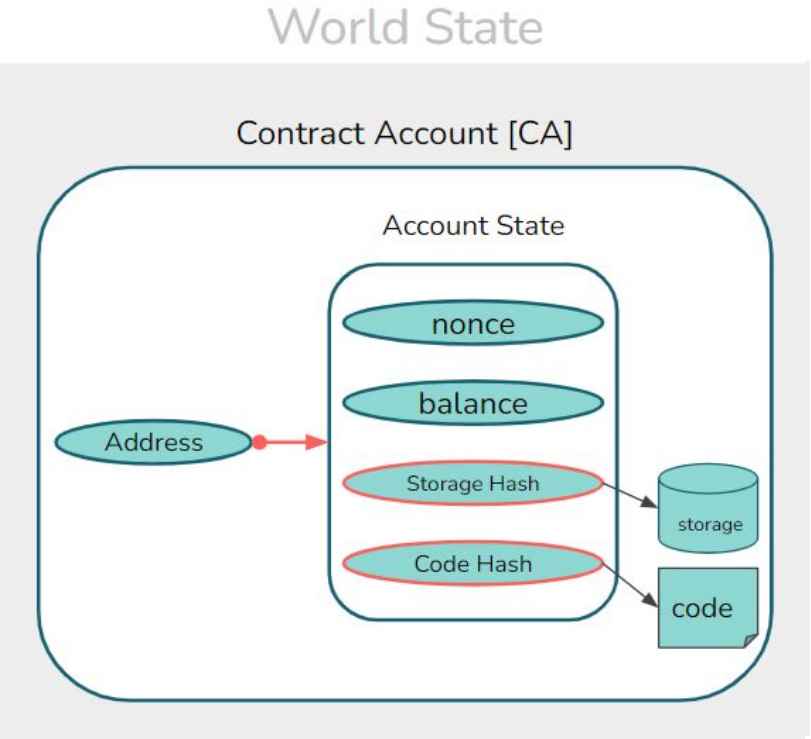

World State
Account
Account State
nonce
balance
storage hash
code hash
Address
Account
Storage
EVM Code

# Ethereum Accounts

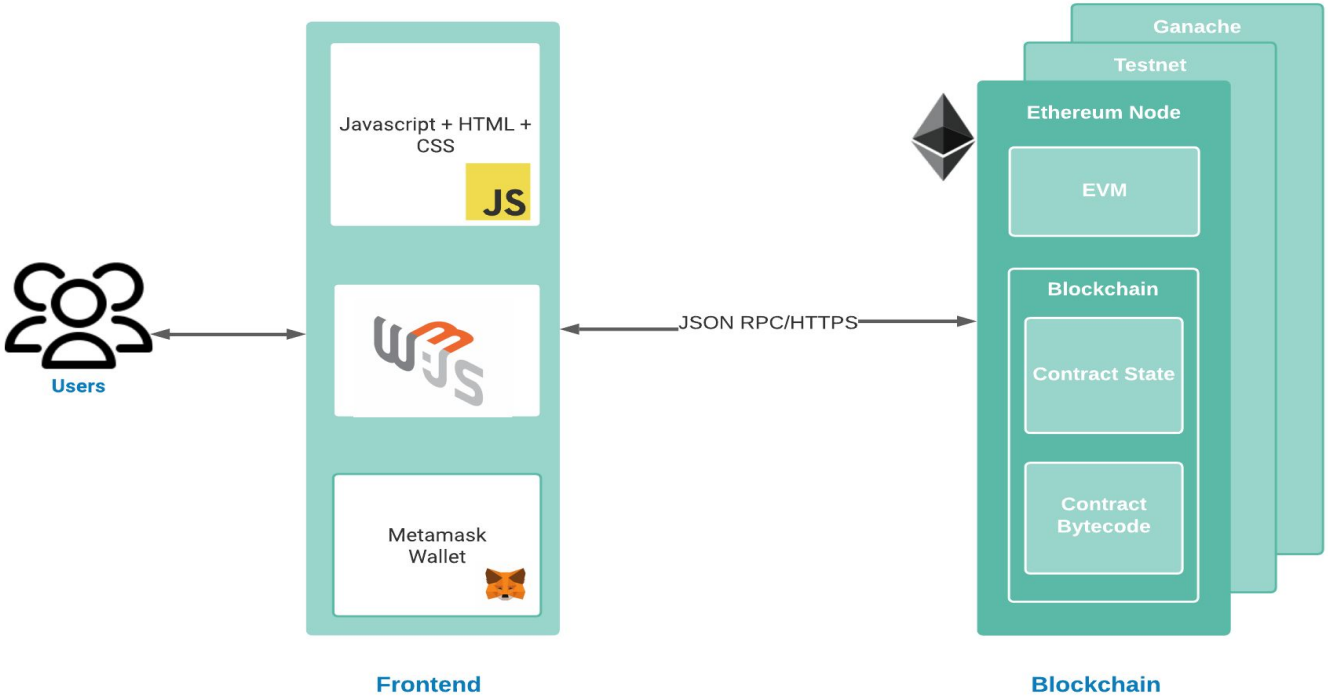
An Ethereum account is an entity with an ether (ETH) balance that can send transactions on Ethereum. Accounts can be user-controlled or deployed as smart contracts.

### Contract Owned Address [COA]

- Smart contract deployed to the network, controlled by code
- Receive, hold and send ETH and tokens
- Interact with deployed smart contracts
- Creating a contract has a cost due to network storage
- Can only send transactions in response to receiving a transaction
- Transactions from an external account to a contract account can trigger code which can execute many different actions, such as transferring tokens or even creating a new contract.



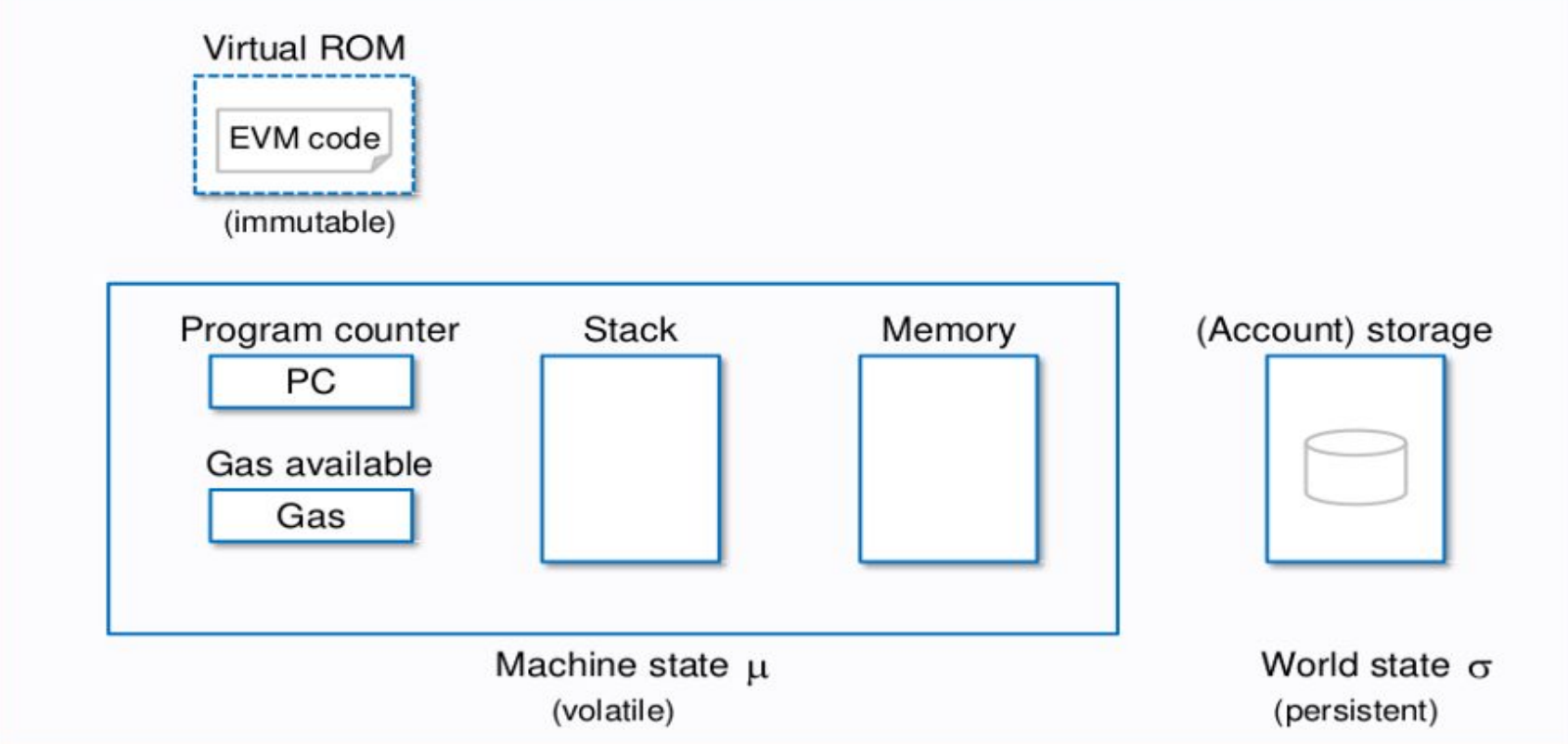
# Decentralized Application

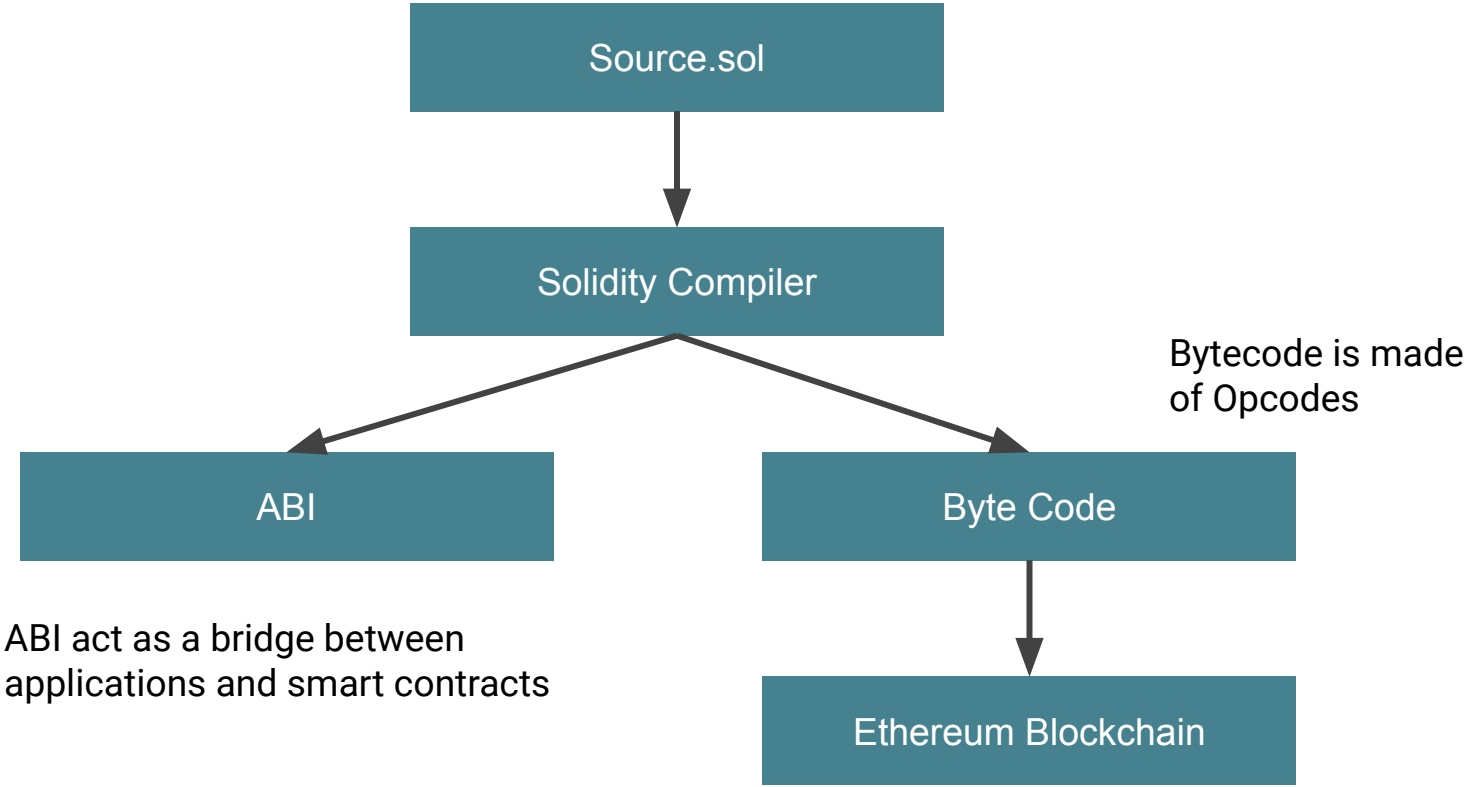


# Ethereum Virtual Machine

- The computation engine for Ethereum that manages the state of the blockchain and enables smart contract functionality.
- EVM is contained within the client softwares like Geth,Parity, Nethermind and others
- participates in block creation
    - sets standards for managing the state from block to block
- participates in transaction execution
    - the EVM executes tasks by interpreting the instructions in Opcodes
- Operates off of a stack-based memory structure
    - It is a stack machine in which each item on the stack is 256-bits or 32 bytes.
- Contains memory components such as Memory, Storage, and Stack
    - Storage is used to store data permanently while memory is used to store data during function calls
    - Storage is used to store data permanently
    - Memory is used to store data during function calls
- Smart contracts are written in higher-level languages, such as Solidity, Vyper, or Yul, and subsequently broken down into EVM bytecode via a compiler.

# Ethereum Virtual Machine




Source.sol
Solidity Compiler
Bytecode is made of Opcodes
ABI
Byte Code
ABI act as a bridge between applications and smart contracts
Ethereum Blockchain

# Opcode in EVM

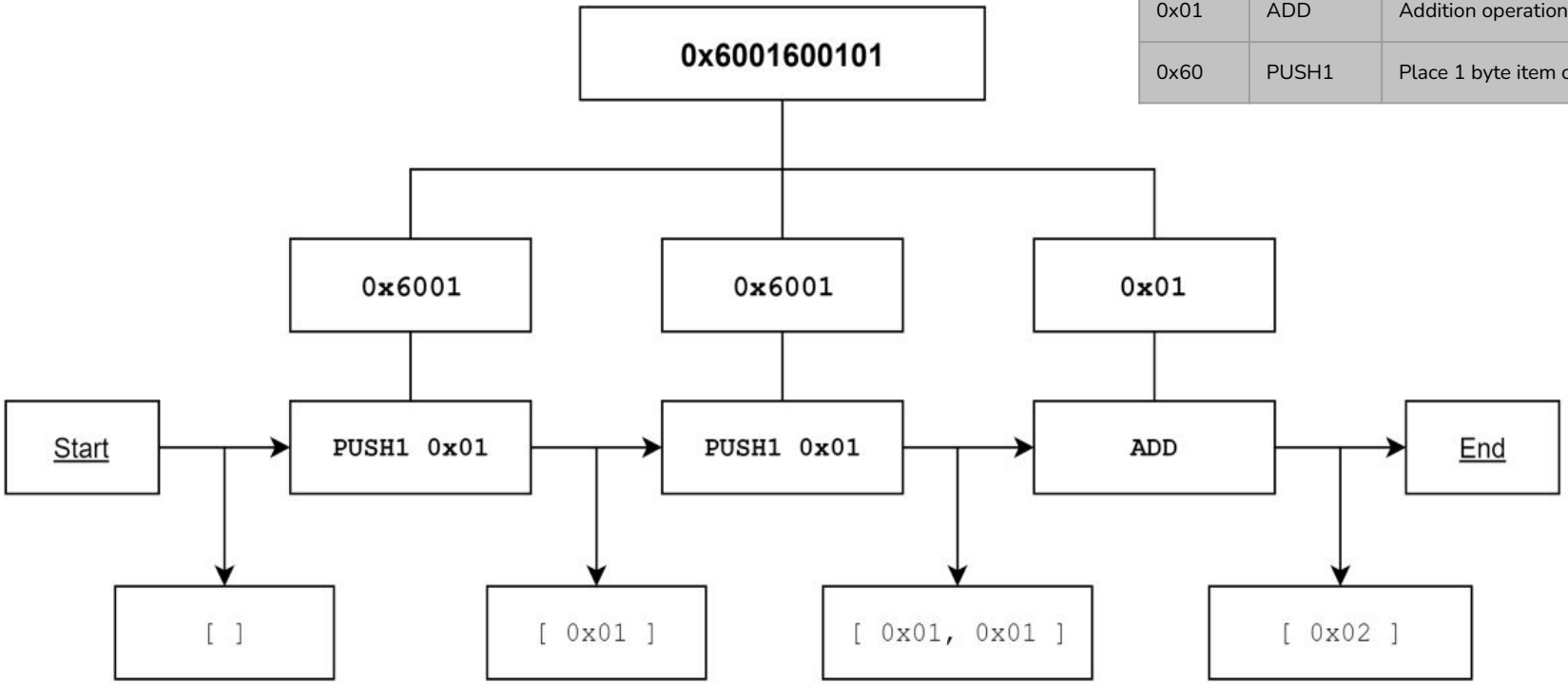
Explore
https://evm.codes

## Instructions SHANGHAI

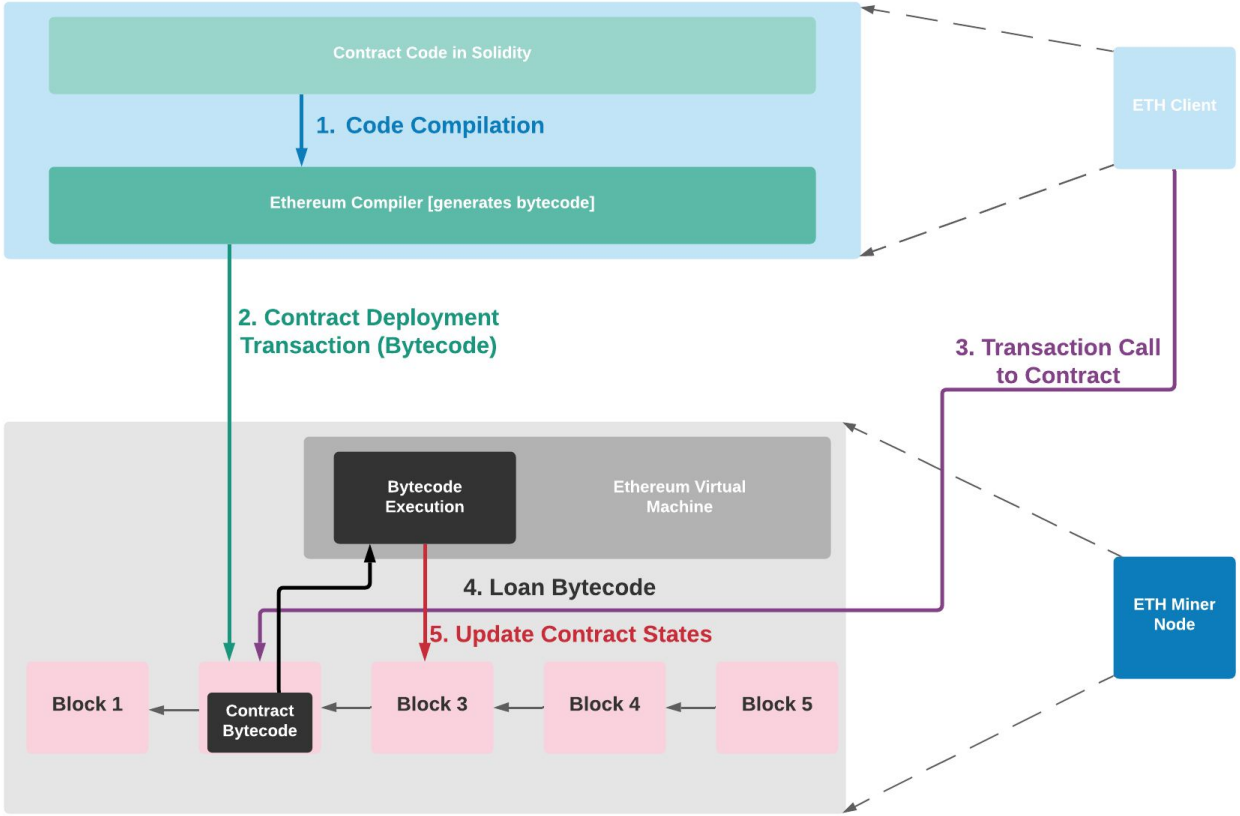
| OPCODE | NAME | MINIMUM GAS | STACK INPUT | STACK OUTPUT |
|---|---|---|---|---|
| 00 | STOP | 0 | | |
| 01 | ADD | 3 | a b | a + b |
| 02 | MUL | 5 | a b | a * b |
| 03 | SUB | 3 | a b | a - b |
| 04 | DIV | 5 | a b | a // b |
| 5F | PUSH0 | 2 | | 0 |
| 60 | PUSH1 | 3 | | value |
| 61 | PUSH2 | 3 | | value |

# Opcode Execution in EVM

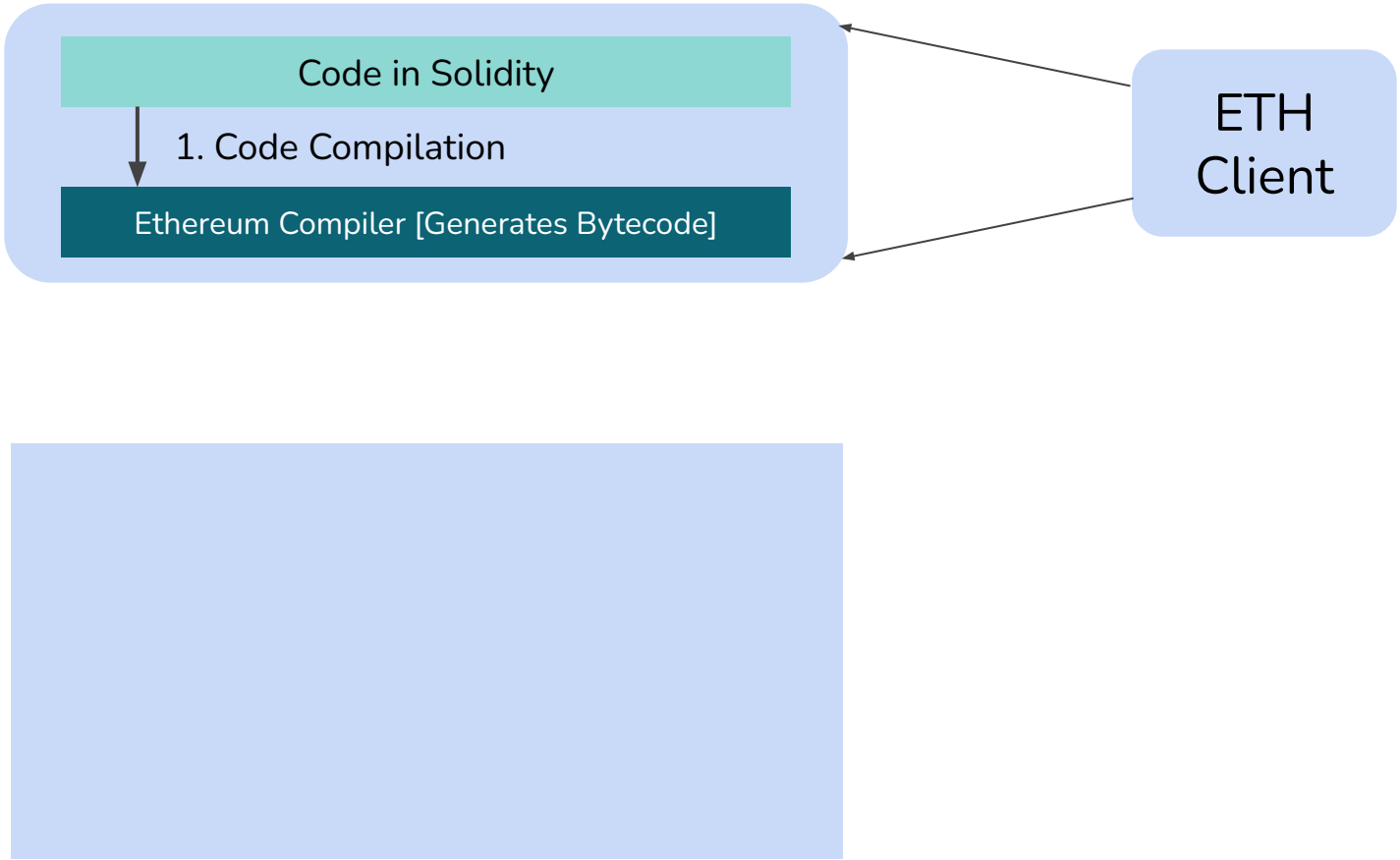
| Opcode | Operation | Description |
|---|---|---|
| 0x01 | ADD | Addition operation |
| 0x60 | PUSH1 | Place 1 byte item on stack |



# EVM Transaction Execution



# EVM Transaction Execution



# Transaction Architecture

The transaction format for Ethereum was originally one standard but has evolved over time to allow other transaction formats. The set of instructions in a transaction object looks like this:

- **from** - the sending address.
- **to** – the receiving address
- **value** – the amount of ETH
- **data** – can contain code or a message to the recipient.
- **gasLimit** – the maximum amount of gas units.
- **nonce** - a number used to track ordering of transactions
- **maxPriorityFeePerGas** - the maximum amount of gas
- **maxFeePerGas** - the maximum amount of gas willing to be paid for the transaction
- **signature** – derived from the sending account's private key

## Types of transactions

1. **Regular transactions:** a transaction from one account to another to transfer native tokens.
2. **Contract deployment transactions:** a transaction without a 'to' address, where the data field is used for the contract code.
3. **Execution of a contract:** a transaction that interacts with a deployed smart contract. In this case, 'to' address is the smart contract address.

# What is Gas in Ethereum?

- “gas” is a unit describing the amount of computational power needed to execute specific operations on the network.
- Every Ethereum transaction consumes computational resources, transactions come with a cost.
- Prevents users from spamming the network with endless transactions.
- These fees are typically paid to miners who validate transactions,
- These fees incentivizes users to mine crypto.
- Gas fee to applicable to run applications
- Sending ETH between wallets also requires fees.

**Note:**

Ethereum fees are typically paid in ETH.

ETH gas prices are denominated in a unit known as “**gwei**.”

**1 ETH = 10*9 gwei**

| Unit | In Wei | Wei | Ether Value |
|---|---|---|---|
| wei | 1 wei | 1 | $10^{-18}$ ETH |
| kwei | $10^{3}$ wei | 1,000 | $10^{-15}$ ETH |
| mwei | $10^{6}$ wei | 1,000,000 | $10^{-12}$ ETH |
| gwei | $10^{9}$ wei | 1,000,000,000 | $10^{-9}$ ETH |
| microether | $10^{12}$ wei | 1,000,000,000,000 | $10^{-6}$ ETH |
| milliether | $10^{15}$ wei | 1,000,000,000,000,000 | $10^{-3}$ ETH |
| ether | $10^{18}$ wei | 1,000,000,000,000,000,000 | 1 ETH |

# How Ethereum Gas Works

## Pre London Upgrade

- Assume Alice wants to pay Bob 1 ETH. The gas limit is 21,000 units, while the gas price is 200 gwei.
- The total fee is calculated as: (gas units (limit) x gas price per unit).
- In this example, that would equal: 21,000 x 200 = 4,200,000 gwei, or 0.0042 ETH.
- When Alice sends the ETH, 1.0042 ETH comes from her Ethereum wallet. Bob receives 1.0000 ETH. An Ethereum miner receives 0.0042 ETH.

## Post London Upgrade

- Assume Alice wants to pay Bob 1 ETH. The gas limit is 21,000 units, the base fee is 100 gwei
- Alice includes a tip of 10 gwei.
- The new formula to calculate total fee is: gas units (limit) x (base fee + tip)
- This can be calculated as 21,000 x (100 + 10) = 2,310,000 gwei, or 0.00231 ETH.
- When Alice sends the ETH, 1.00231 ETH will be subtracted from her wallet. Bob will receive 1.0000 ETH. A miner will receive the tip of 0.00021 ETH. And 0.0021 ETH will be burned.

# What is Ethereum Gas Limit?

- The maximum amount of gas a user can consume to conduct a transaction.
- The standard limit is **21,000** units.
- Smart contracts transactions are complex, and require more computational power.
- So these transactions need a higher gas limit than simpler transactions.
- If the gas limit is set too high the EVM will refund what doesn't get used.
- Low gas limit could result in a user losing gas Fee and transaction being declined.

**Note:**
If a user were to place an Ether gas limit of 50,000 for an ETH transfer, for example, the EVM would consume 21,000 and refund the remaining 29,000. But if someone were to set a gas limit of 20,000, and the transaction were to require 21,000 units, the EVM could consume 20,000 gas units as it tries to fulfill the transaction, but the transaction won't complete.

In this case, the user would hold on to the ETH they tried to send, but their 20,000 gas units would be lost because the EVM consumed it trying to complete the failed transaction.

## Benefits of Gas Fee

- Post London Upgrade, users can better anticipate what their total transaction cost will be.
- Users can also send higher tips to miners to prioritize their transactions.
- Adequate ETH gas fee is that it ensures a transaction will be accepted by the network.

# Token Standards

- set of rules, conditions, and functions that guide the design, development, behavior, and operation of tokens
- Are designed to allow tokens on the Ethereum Blockchain to interact seamlessly
- For token standards to be useful, they must be widely adopted.

### The Need

- offer guidelines for the creation, issuance, deployment, transfer, destruction of tokens
- standards help add to the diversity of implementations
- Composability - we can design contracts to interact with each other because standardization enables us to know what functions, methods, data types, and behaviors a complying smart contract will expose.

### Ethereum Request for Comments (ERC)

- is essentially a set of technical documents containing guidelines on developing a smart contract
- These documents define a specific set of functions for each token type and facilitate the interaction between applications and smart contracts.
- After working through the EIP process, some of these documents are accepted by the Ethereum community, finalized, and then developers implement them. These documents are then called ERC Standards.

# Token Standards

Explore
https://eips.ethereum.org/erc

| Standards | Use Cases | Examples |
|---|---|---|
| ERC-20<br>Fungible Token<br>Standard | • Allows Token Transfer and Allowance<br>• Token are Highly Divisible<br>• Each ERC-20 Token is strictly equivalent to the same value regardless of its feature and structure.<br>• Stablecoins, governance tokens, and ICOs | 1. Chainlink [LINK]<br>2. Dai [DAI]<br>3. Tether [USDT] |
| ERC-721<br>Non Fungible Token<br>Standard | • Tokens are unique and non-divisible<br>• These tokens cannot be exchanged for anything of equal value since they are one-of-a-kind.<br>• Each ERC-721 represents the value of the respective NFT, which may differ<br>• For Digital Collectibles like art, tickets, domain names | 1. Bored Ape Yacht Club<br>2. CryptoPunks<br>3. Azuki |
| ERC-1155<br>Multi-token<br>Standard | • ERC-1155 is a smart contract interface representing fungible, semi-fungible, and non-fungible tokens.<br>• Can perform the function of ERC-20 and ERC-721 and even both simultaneously.<br>• ERC-1155 is applicable for creating NFTs, redeemable shopping vouchers, ICOs | 1. Online Gaming<br>2. Shopping Vouchers |

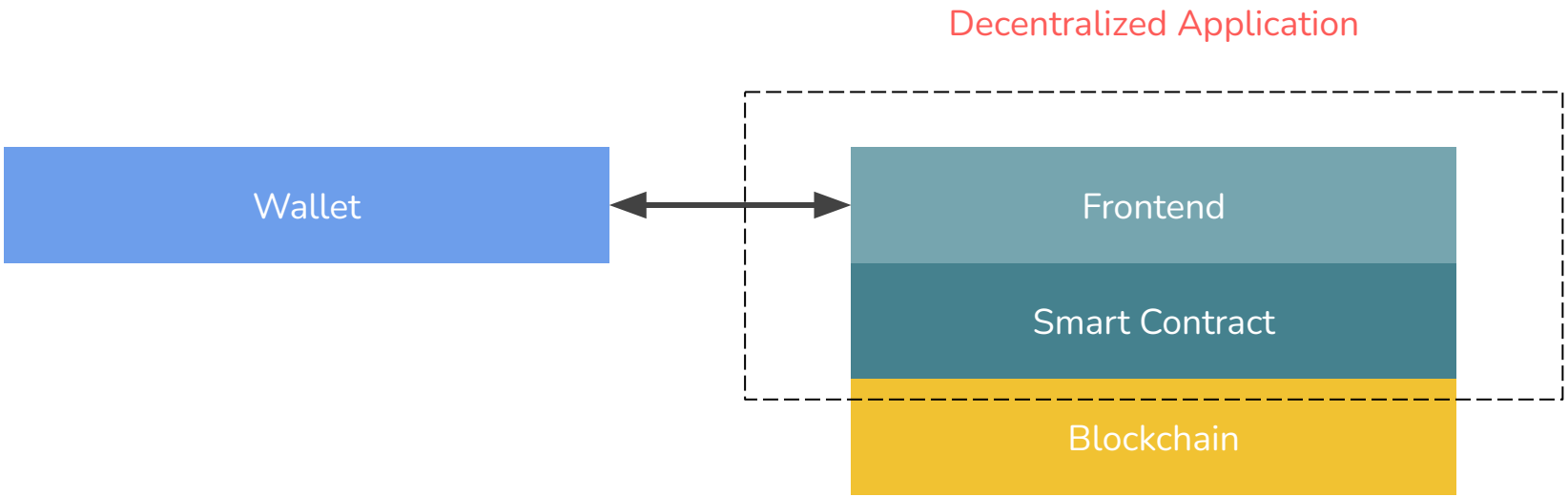
# Coins vs Tokens

- Digital coins are generally used in the same way as a real-life coin is – as money.
- Bitcoin
- Ethereum
- Dash

- Tokens are created on existing blockchains.
- Do not have to worry about building and maintaining a Blockchain.
- EOS (EOS)
- VeChain (VET)
- ICON (ICX)

# Dapps
# [Decentralized Applications]

# Dapp Architecture



METAMASK

# REMIX IDE

SOLIDITY

# Industry Applications

Insurance

Digital Governance

Personal Identity

Voting Mechanism

Healthcare

Certificates

Automobile

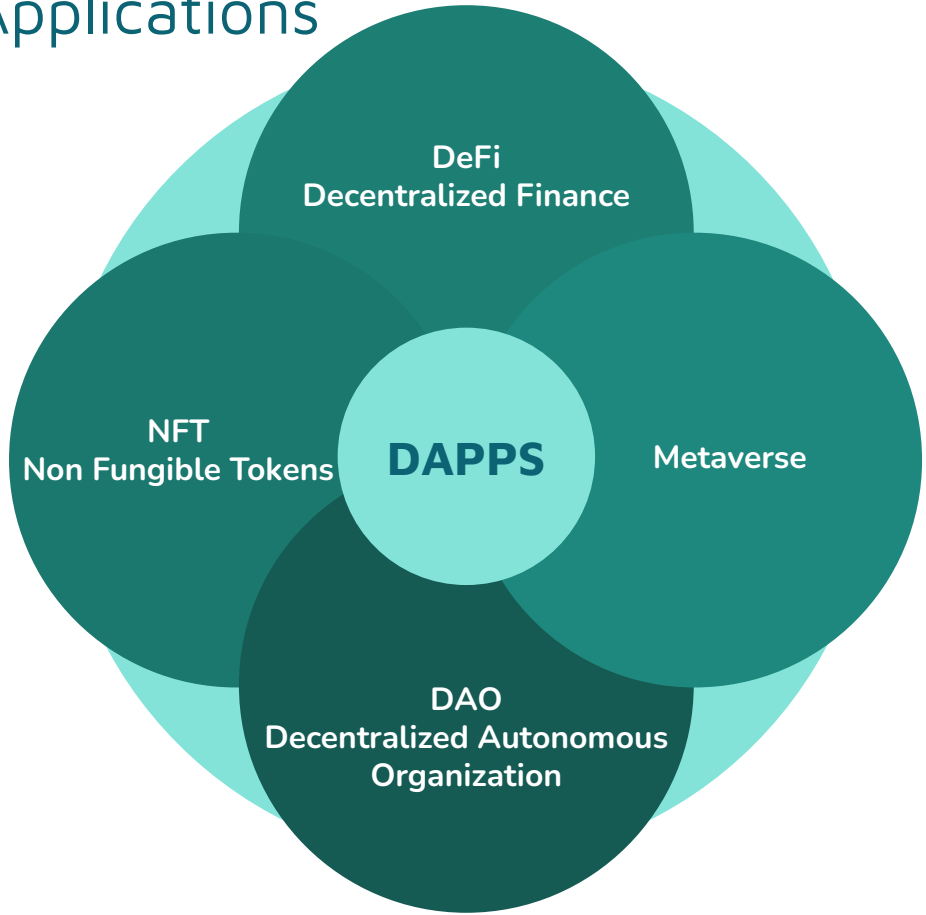
Supply Chain Management

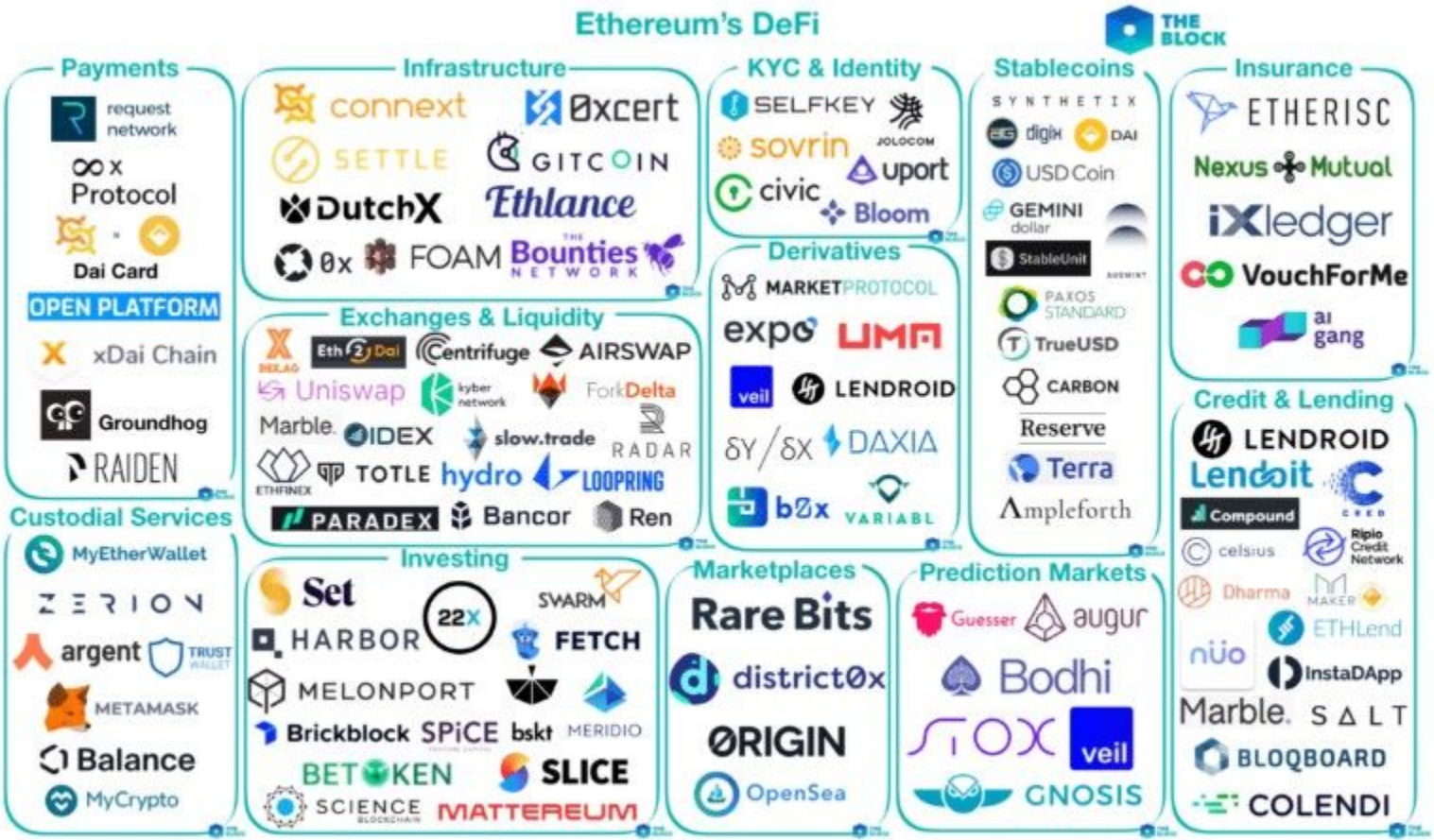
Cross Border Payments

Asset Management

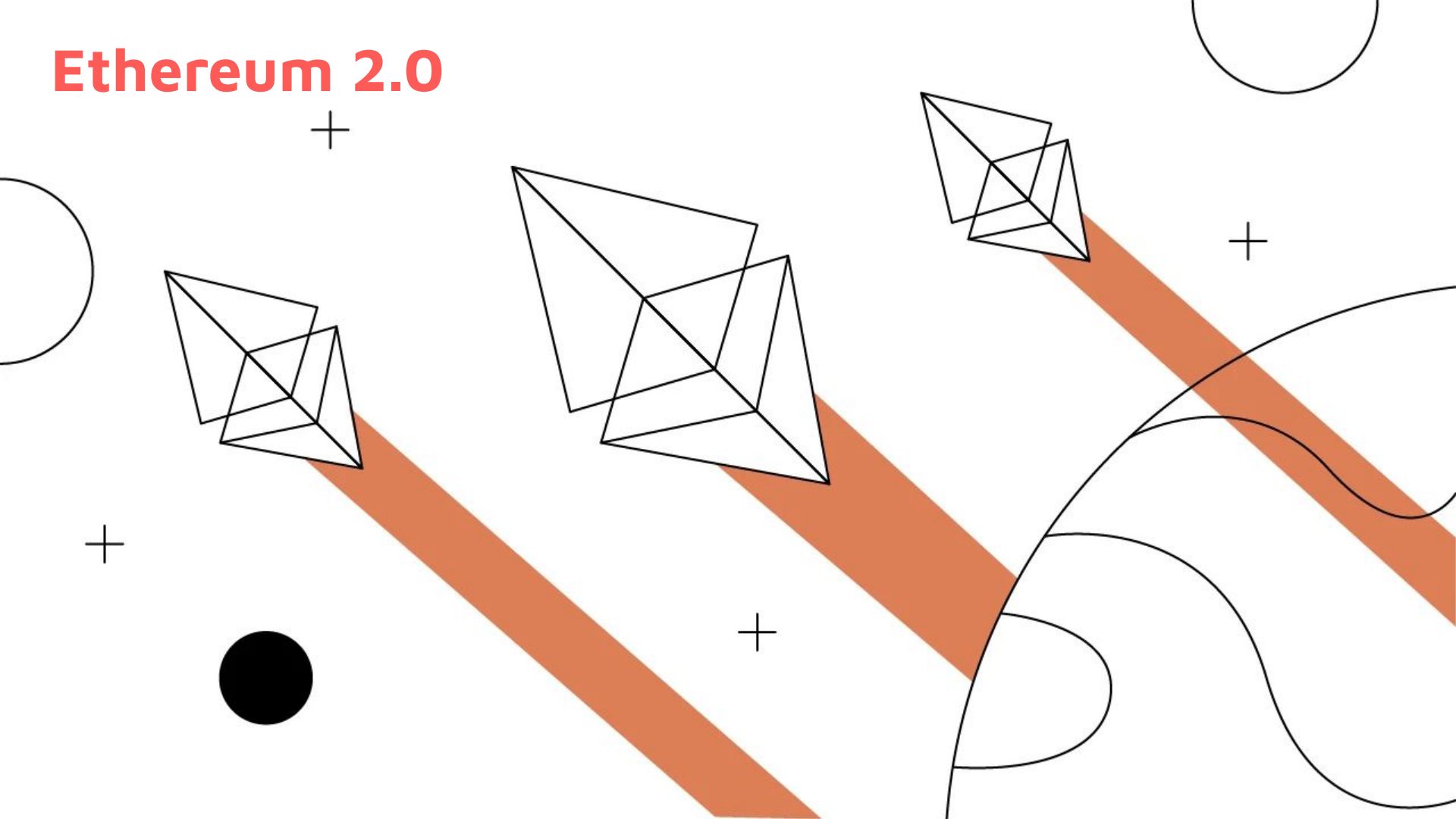
Agriculture

# Decentralised Applications

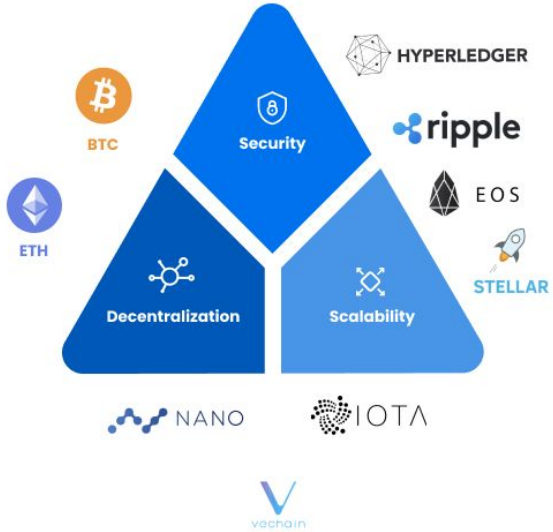


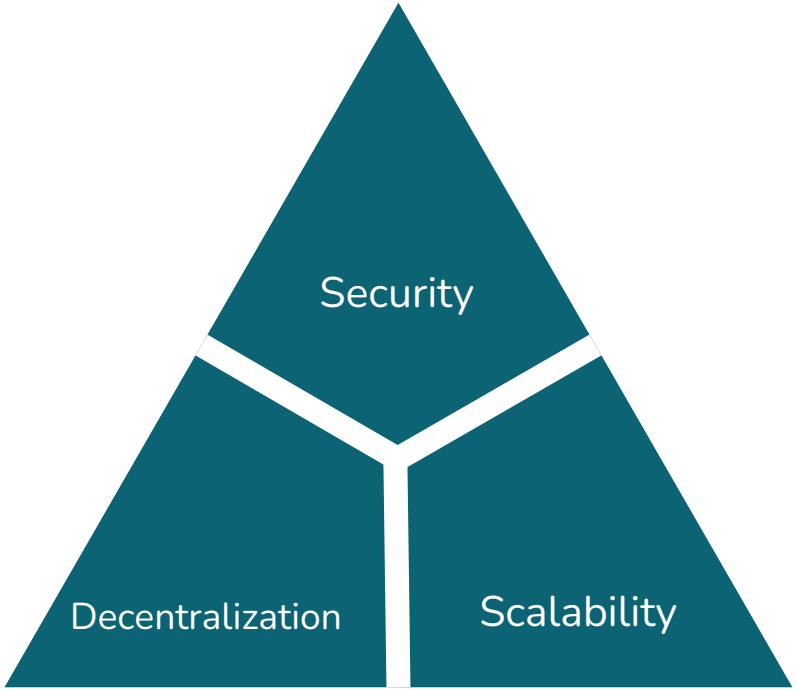

Ethereum's DeFi
THE BLOCK
Payments
request network
0x Protocol
Dai Card
OPEN PLATFORM
xDai Chain
Groundhog
RAIDEN
Infrastructure
connext
0xcert
SETTLE
GITCOIN
DutchX
Ethlance
0x
FOAM
Bounties NETWORK
Exchanges & Liquidity
Eth2Dai
Centrifuge
AIRSWAP
Uniswap
kyber network
ForkDelta
Marble
IDEX
slow.trade
RADAR
ETHFINEX
TOTLE
hydro
LOOPRING
PARADEX
Bancor
Ren
KYC & Identity
SELFKEY
sovrin
JOLOCOM
uport
civic
Bloom
Derivatives
MARKETPROTOCOL
expo
UMA
veil
LENDROID
δY/δX
DAXIA
bZx
VARIABL
Stablecoins
SYNTHETIX
digix
DAI
USD Coin
GEMINI dollar
StableUnit
PAXOS STANDARD
TrueUSD
CARBON
Reserve
Terra
Ampleforth
Insurance
ETHERISC
Nexus Mutual
iXledger
VouchForMe
ai gang
Credit & Lending
LENDROID
Lendoit
Compound
celsius
Ripio Credit Network
Dharma
MAKER
nüo
ETHLend
InstaDApp
Marble
SALT
BLOQBOARD
COLENDI
Custodial Services
MyEtherWallet
ZERION
argent
TRUST WALLET
METAMASK
Balance
MyCrypto
Investing
Set
SWARM
HARBOR
22X
FETCH
MELONPORT
Brickblock
SPiCE
bskt
MERIDIO
BETOKEN
SLICE
SCIENCE
MATTEREUM
Marketplaces
Rare Bits
district0x
ORIGIN
OpenSea
Prediction Markets
Guesser
augur
Bodhi
veil
GNOSIS

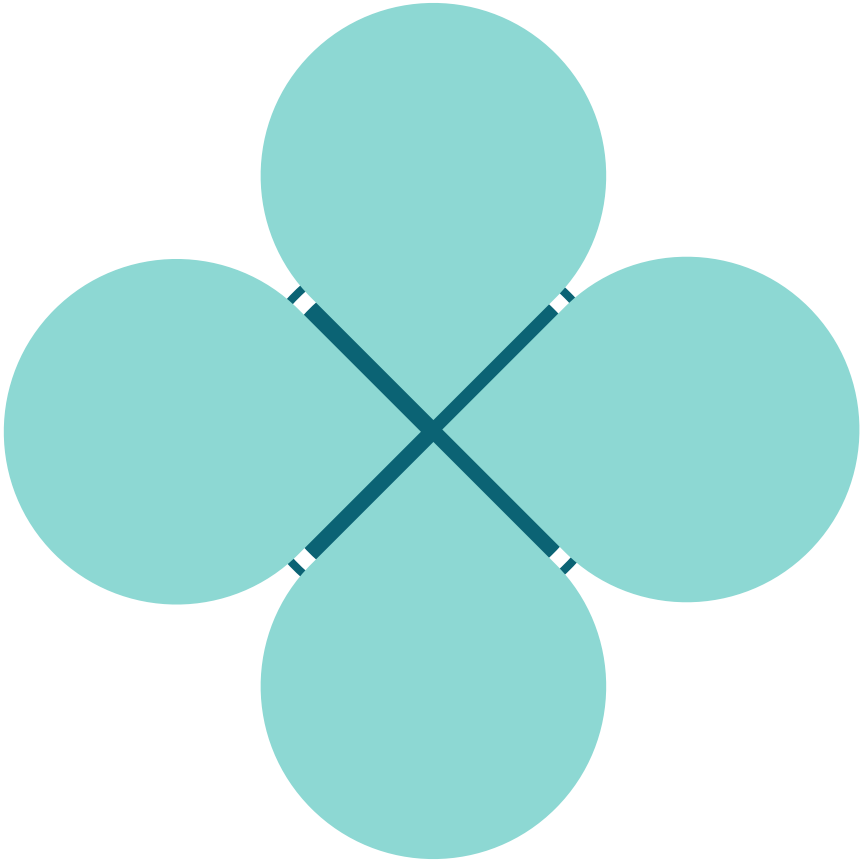
# Ethereum 2.0



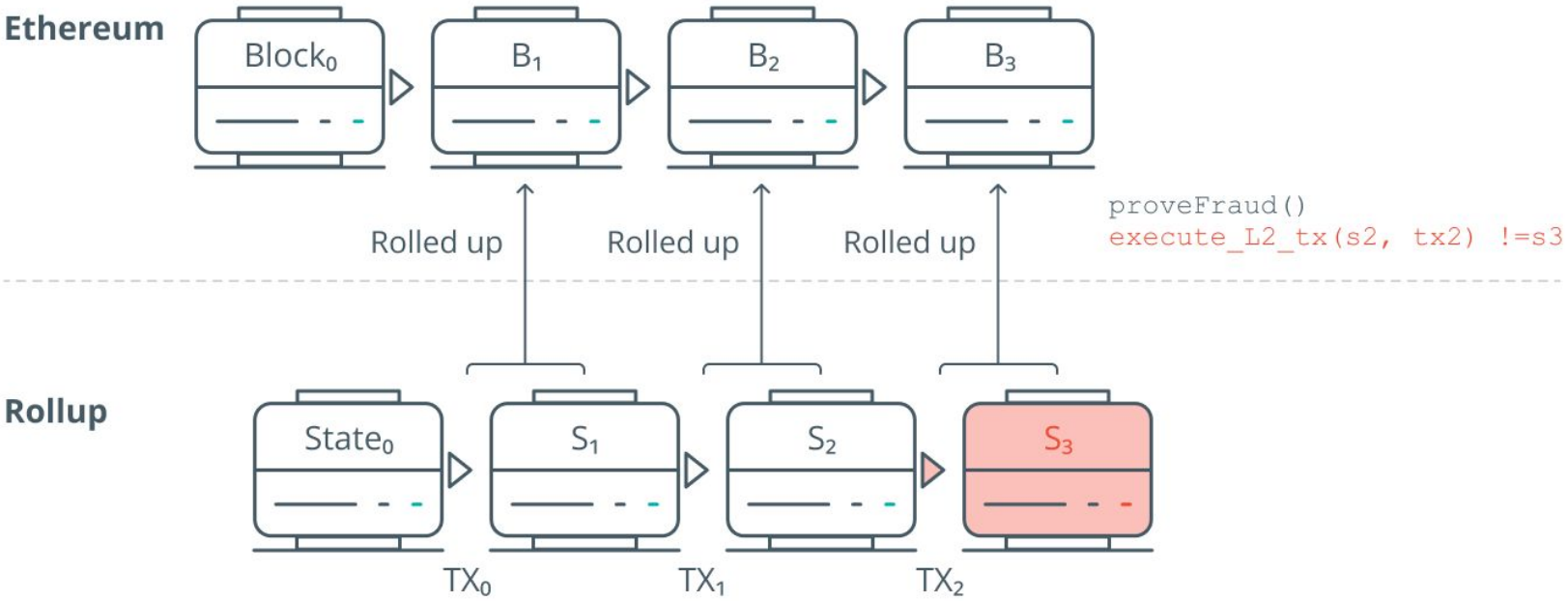
# Blockchain Trilemma



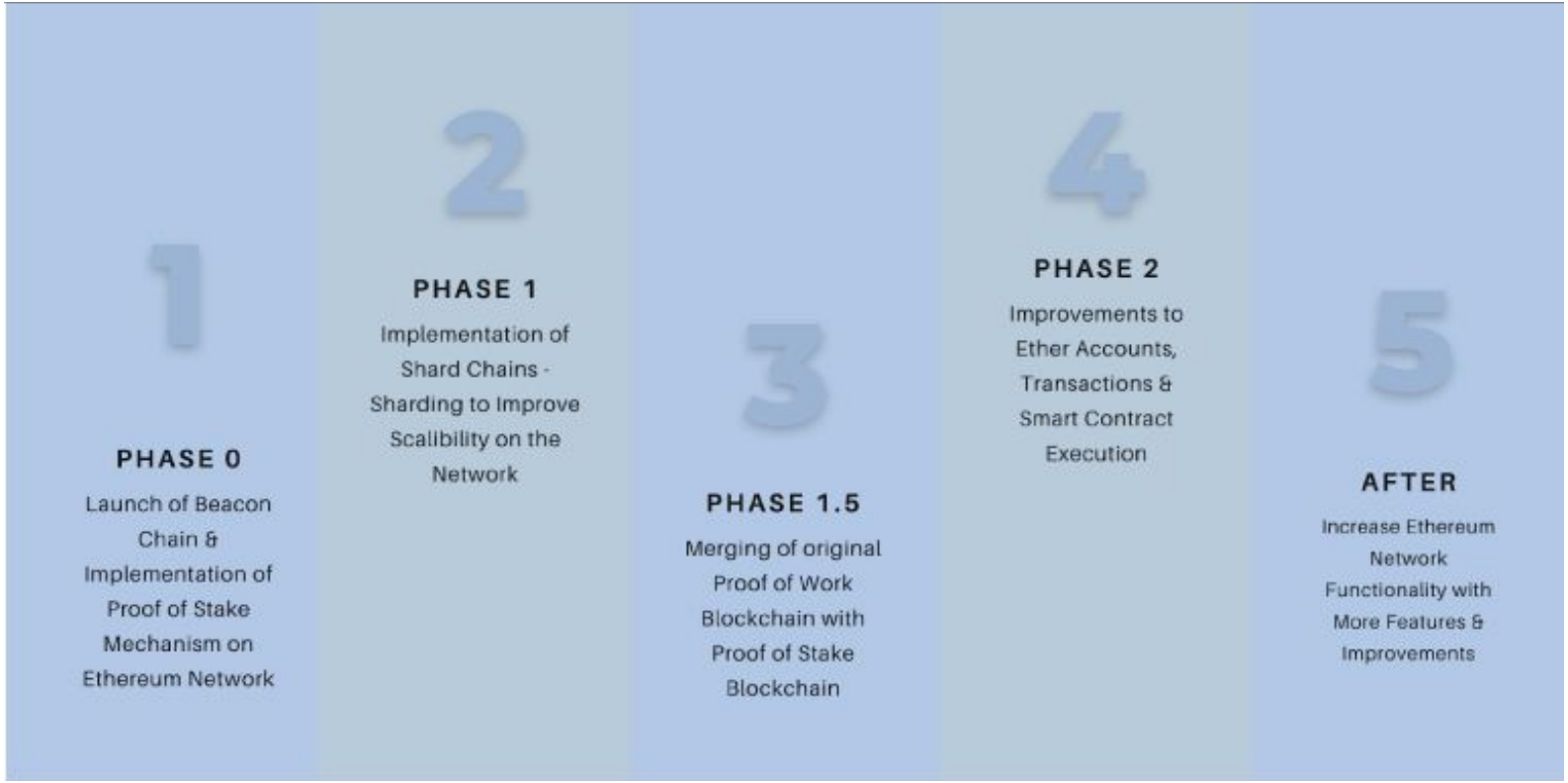



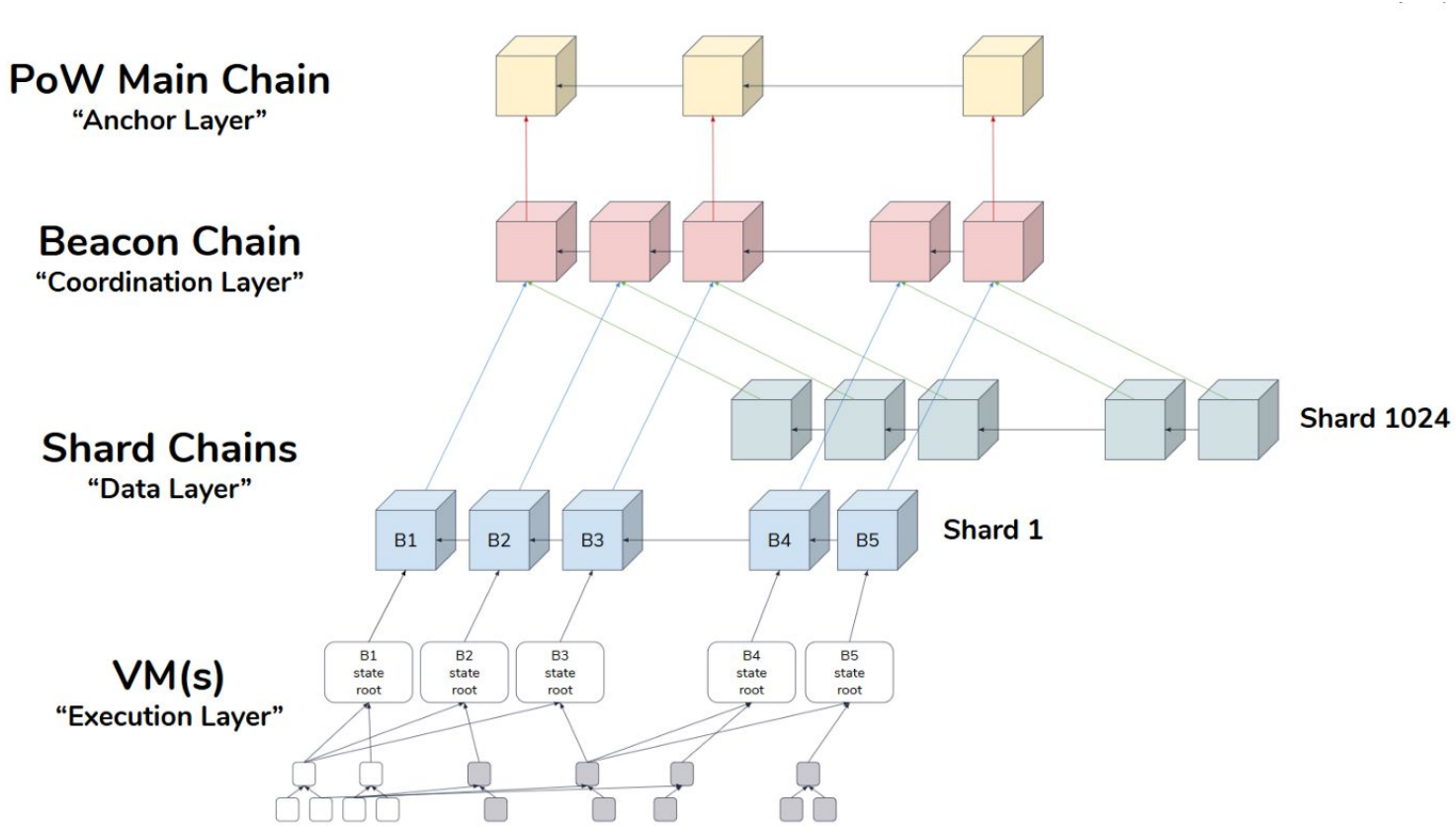


# How Roll Ups work?



# ETH 2.0 Upgrade Phases

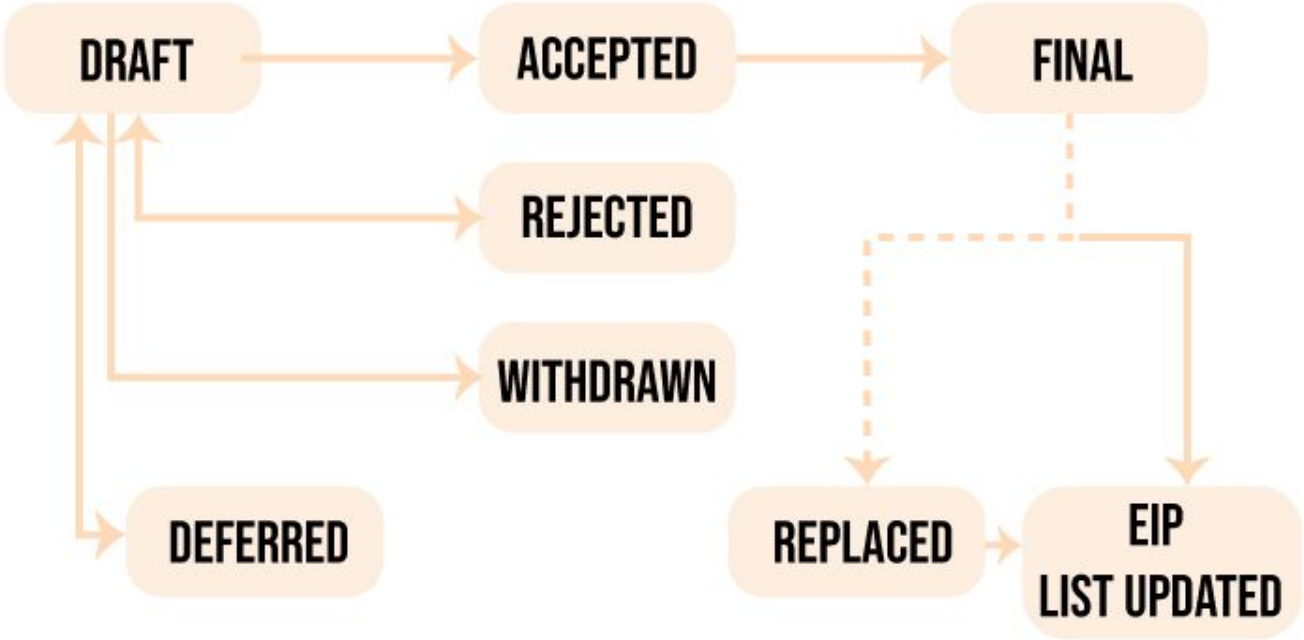



PoW Main Chain
“Anchor Layer”
Beacon Chain
“Coordination Layer”
Shard Chains
“Data Layer”
VM(s)
“Execution Layer”
Shard 1024
Shard 1
B1
B2
B3
B4
B5
B1 state root
B2 state root
B3 state root
B4 state root
B5 state root


SOLIDITY

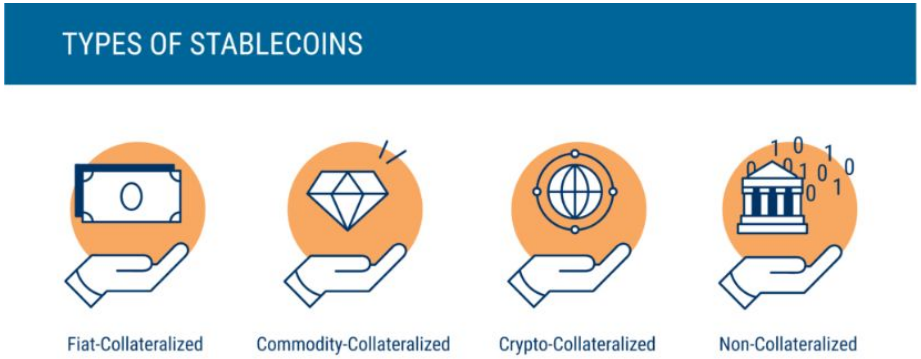
The **tx.origin** global variable refers to the original external account that started the transaction while **msg.sender** refers to the immediate account (it could be external or another contract account) that invokes the function.

# EIP PROCESS


DRAFT
ACCEPTED
FINAL
REJECTED
WITHDRAWN
DEFERRED
REPLACED
EIP
LIST UPDATED

# What are Stable Coins?

- A day-to-day currency
- Streamlining recurring and P2P payments
- Fast and affordable remittances for migrant workers
- Protection from local currency crashes
- Improved cryptocurrency exchanges




Stable Coins

- USD - USDC, USDT, BUSD
- GP - LXGPeg,
- Gold Collateral - DGX
- Crypto Collateral - DAI
- Non Collateral - Basis

# Solidity
# Project Assignments

# 1. Rock Paper Scissors

- Create Rock Paper Scissors Contract
- Rules of the Game
    - Rock Beats Scissors; Scissors Beat Paper; Paper beats Rock.
    - If one of the players selection is invalid, other player wins by default.
    - Both player choose same option or both players enter invalid option, Game will result in a Draw.
    - Use Commit Reveal Method to Implement the Game
- Players' Choice should be hidden until both players choices has been recorded.
- After the game is completed, users must be able to view the game result.
- Multiple Games can run simultaneously.

NFRs

- Document the Functionality
- Provide Execution Steps
- Deployed to Goerli Testnet
- Contract must be verified on Etherscan.
- Add Modifiers and Interfaces and Type Declarations at suitable places.
- Clear Error Messages Needed.
- Add Events at crucial stages.
- Optimize code for Gas
- Add your own features. - Optional
- Test Cases - Optional

**Team**
**Karthik, Rishabh, Sonil and Sahil**

# 2. Kickstarter Campaign

- User creates a campaign.
- Campaign Details
  - Creator of campaign & Amount of ETH to raise
  - Total amount pledged
  - Timestamp of start & end of campaign
  - Status of Campaign - True if goal reached
- Users can pledge, transferring their token to a campaign.
- After the campaign ends, campaign creator can claim the funds if total amount pledged is more than the campaign goal.
- Otherwise, campaign did not reach it's goal, users can withdraw their pledge.
- Multiple Campaigns can run simultaneously.

NFRs

- Document the Functionality
- Provide Execution Steps
- Deployed to Goerli Testnet
- Contract must be verified on Etherscan.
- Add Modifiers and Interfaces and Type Declarations at suitable places.
- Clear Error Messages Needed.
- Add Events at crucial stages.
- Optimize code for Gas
- Add your own features. - Optional
- Test Cases - Optional

**Team**

**Vipin, Jitender and Sumit**

# 3. Lottery on Chain

- Owner creates a Lottery Smart Contract
- Lottery Details
  - Name of the Lottery & Creator of Lottery
  - Start Time & End Time
  - Ticket Value - 0.1 ETH
- User can buy Multiple Tickets to increase his chances of winning
- Owner cannot buy Tickets; Max of 20 Tickets can be sold.
- Winner has to be chosen in Random. - Chainlink VRF
- After the end of Lottery time, Owner can trigger the findWinner to decide the winner
- Winner can withdraw his lottery amount 80%
- Owner can withdraw 20% to his address.

NFRs

- Document the Functionality
- Provide Execution Steps
- Deployed to Goerli Testnet
- Contract must be verified on Etherscan.
- Add Modifiers and Interfaces and Type Declarations at suitable places.
- Clear Error Messages Needed.
- Add Events at crucial stages.
- Optimize code for Gas
- Add your own features. - Optional
- Test Cases - Optional

## Team

- Invoking methods of Smart Contract with Hardhat
-

# 4. Auction Contract

Auction

- Seller of NFT deploys this contract.
- Auction lasts for 12 hours.
- Participants can bid by depositing ETH greater than the current highest bidder.
- All bidders can withdraw their bid if it is not the current highest bid.

After the auction

- Highest bidder becomes the new owner of the NFT.
- The seller receives the highest bid of ETH.

## Team

NFRs

- Document the Functionality
- Provide Execution Steps
- Deployed to Goerli Testnet
- Contract must be verified on Etherscan.
- Add Modifiers and Interfaces and Type Declarations at suitable places.
- Clear Error Messages Needed.
- Add Events at crucial stages.
- Optimize code for Gas
- Add your own features. - Optional
- Test Cases - Optional

# 5. Voting on Chain

- Owner creates a Voting Contract
- Voting Details
    - Name of the Election
    - Start Time & End Time
- Before the Start time, Owner or Candidates can add their Candidature
- After the Start Time, other Addresses and Candidates can Vote in the Election
- At the end of the Vote Duration Winner is Chosen based on the no of votes.
- Invalid Votes should not be counted
- Share a Report No of Votes, Valid Votes and Invalid Votes etc.

NFRs

- Document the Functionality
- Provide Execution Steps
- Deployed to Goerli Testnet
- Contract must be verified on Etherscan.
- Add Modifiers and Interfaces and Type Declarations at suitable places.
- Clear Error Messages Needed.
- Add Events at crucial stages.
- Optimize code for Gas
- Add your own features. - Optional
- Test Cases - Optional

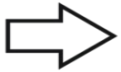
**Team**

**Kapil, Vishwanath**

**Himanshu Rajput, Prince Shakya & Abhinav Sharma**

```
pragma solidity 0.5.3;

contract ExampleContract {
    uint256 number = 1;
}
```



```
0x6080604052600160005534801560145760080fd5b5060358060226000396000f3fe6080604052600080fdfea165627a7a723058204e048d6cab20eb0d9f95671510277b55a61a582250e04db7f6587a1bebc134d20029
```

- When deploying a smart contract, a regular transaction is created without a ***to*** address.
- Additionally, some bytecode is added as input data.
- This bytecode acts as a constructor, which is needed to write initial variables to storage before copying the runtime bytecode to the contract's code.
- During deployment, creation bytecode will only run once, while runtime bytecode will run on every contract call.

EVM can access and store information in six places

- Stack
- Memory
- CallData
- Storage
- Bytecode
- Logs

Solidity Tricks

## Variable Locations

- Stack
  - Cheapest
  - use it like cache
- Memory
  - Function Parameters are stored here
- CallData
  - External Function Parameters are stored in the transaction call itself
  - these cannot be edited
- Storage
  - Most Expensive
  - For State Variables
- Bytecode
  - Constant Variables are stored in the contract bytecode

## Variable Visibility

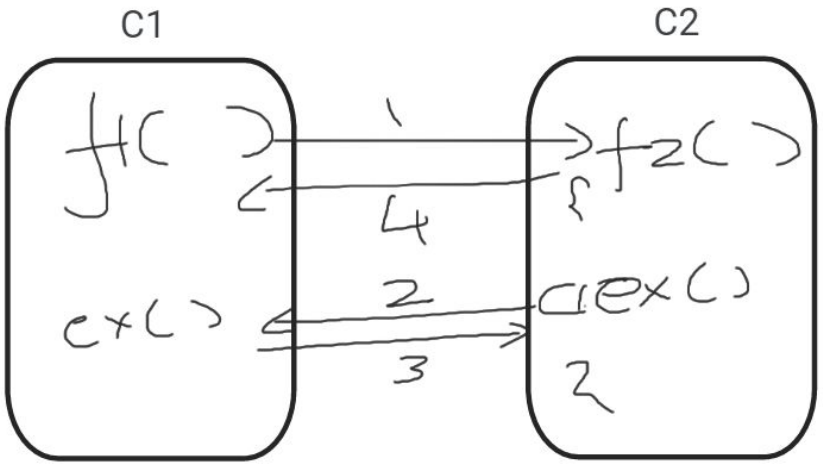
- Types
  - Internal - Default
  - Public
  - Private
- Private Variables cannot be modified by child contracts
- Private Variables are not truly private
  - any variable can be accessed externally with web3.eth.getStorageAt(address, slot)
- Public variable have getter functions by the same name.
- Internal variables are cheaper

Note: Max Storage Space per Contract on EVM is 24KB

Solidity Tricks - 2

Function Access Types

- Types
  - Public
  - Private
  - Internal
  - External
- External Functions can only be called by EOAs or other Contracts
- This can be bypassed
  - Receive response from function of other contract which calls the external function.
  - Using “this” - this.externalFunction()
- Internal functions are the cheapest
  - As it doesn’t use calldata
  - As it doesn’t copy data to memory
  - And it uses references to variables



Solidity Tricks - 3

Packing Variables

- Contract Stroage is divided into slots of 32 bytes
- Smaller variables can be packed together to save space
- Variable in storage can be packed
- Variables in memory cannot be packed
- Different datatypes can be packed together like
    - Address (20 bytes)
    - Uint64 (8 bytes) etc
- As long as datatypes are less than 32 bytes they can be packed together.
- Booleans use 8 bits while we only need 1 bit

EVM works with 256 bits

- All operations done on EVM are on 256 bits data
- All small datatypes are internally converted to full size datatypes before operation
- Therefore uint8 operations are more expensive than uint256 operation.
    - As storage is significantly more expensive than type conversion overall trxs will be cheaper

Note: If packing of datatypes is possible use smaller datatypes others refrain from using them.

Solidity Tricks - 4

## Miscellaneous

- Mappings are cheaper than arrays if the data cannot be packed
- In Solidity, data is always stored in key value pair unlike other programming languages.
- Internally even arrays are stored as key value pairs.
- Fixed size datatypes like bytes32 is cheaper than dynamic sized datatypes like bytes.
    - As first byte will be used for storing the size of the datatype in dynamic types.
- Short Circuits are efficient “&&” and “||” etc
- Function modifiers can be inefficient
    - Function code is copied into the modifier at “_;”
    - If possible use internal functions over modifiers
- Use Solidity Compiler Optimization
- Single line swaps
    - (hello, world) = (world, hello)
- Use native solc than solcjs as native solc is faster

# Assembly in Solidity - Yul

```
contract StoringData {

  function setData(uint256 newValue)
    public {
    assembly {
      sstore(0, newValue)
    }
  }

  function getData() public view
    returns(uint256) {

    assembly {
      let v := sload(0)
      mstore(0x80, v)
      return(0x80, 32)
    }
  }
}
```

Yul Documentation:
https://docs.soliditylang.org/en/latest/yul.html

# Assembly in Solidity - 2

```
contract SendETH {

address[2] owners =
      [0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2,
       0xdD870fA1b7C4700F2BD7f44238821C26f7392148];

function withdrawETH(address _to, uint256 _amount) external payable {

      bool success;
      assembly {
        for { let i := 0 } lt(i, 2) { i := add(i, 1) } {
         let owner := sload(i)
         if eq(_to, owner) {
           success := call(gas(), _to, _amount, 0, 0, 0, 0)
         }
        }
      }

    require(success, "Failed to send ETH");
  }

}
```

# Assembly in Solidity - 3

```
contract UselessEncryption {

  function encrypt(string memory _input, bool _decrypt) external pure returns (string memory) {
    bytes32 output;

    assembly {
      function stringToBytes(a) -> b {
        b := mload(add(a, 32))
      }
      function addToBytes(bs,decrypt) -> r {
        if eq(decrypt, false) {
          mstore(0x0, add(bs,0x0101010101010101010101010101010101010101010101010101010101010101))
        }
        if eq(decrypt, true) {
          mstore(0x0, sub(bs,0x0101010101010101010101010101010101010101010101010101010101010101))
        }
        r := mload(0x0)
      }
      let byteString := stringToBytes( input)
      output := addToBytes(byteString,_decrypt)
    }
    bytes memory bytesArray = new bytes(32);
    for (uint256 i; i < 32; i++) bytesArray[i] = output[i];
    return string(bytesArray);
  }
}
```

| Instruction | Description |
| --- | --- |
| add(x,y) | Adds the top two values in the stack together and replaces them with result |
| sub(x,y) | Same for subtraction |
| mul(x,y) | Same for multiplication |
| div(x,y) | Same for division |
| mod(x,y) | Same for modulus |
| lt(x,y) | Less than. Returns 1 if x is less than y. |
| gt(x,y) | Greater than |
| eq(x,y) | Equal to. Returns 1 if x == y |
| not(x) | Bitwise NOT (1010 > 0101) |
| and(x,y) | Bitwise AND (1000 AND 1100 > 1000) |
| or(x,y) | Bitwise OR (1000 AND 1100 > 1100) |
| xor(x,y) | Bitwise XOR (1000 XOR 1100 > 0100) |

Full List: https://docs.soliditylang.org/en/latest/yul.html

| Instruction | Description |
| --- | --- |
| byte(n,x) | Nth byte of X |
| keccak256(pos,n) | Native hashing algorithm |
| pop(x) | Pop x off the stack |
| mload(pos) | Load from memory at position pos |
| mstore(pos,value) | Store value in memory at pos |
| sload(pos) | Load from storage at position pos |
| sstore(pos,value) | Store value in storage at pos |
| balance(address) | Eth balance of an address in wei |
| call(gas,address,value,in,insize,out,outsize) | Call an external contract, also used to send funds, returns 1 on success |
| delegatecall(gas,address,value,in,insize,out,outsize) | As above but call from user instead of contract |
| revert(p,s) | Revert transaction and any state changes |
| return(p,s) | End execution returning data |

**What is selfdestruct?**

***Selfdestruct* is a lowlevel function used to terminate a contract, remove the bytecode from the Ethereum blockchain, and send any contract funds to a specified address.**

*Selfdestruct* was created to serve as an exit door in case of security threats. Although it was first called the “suicide function”, it was renamed to *selfdestruct* from **Solidity v0.5.0** onwards.

**Why is *selfdestruct* considered a sensible security practice?**

**Including a *selfdestruct* function in your contract is a sensible security practice as it safeguards contract funds against malicious attacks.**

**Why do developers use the *selfdestruct* function?**

**Developers use the *selfdestruct* function primarily to improve smart contract code security, clean up unused contracts and transfer Ethereum assets quickly.**

*Selfdestruct* is helpful when developers need to upgrade smart contracts. they use the *selfdestruct* function to extract funds from the current contract and build a new contract with the required functionalities.

Developers also use the *selfdestruct* function to safeguard against potential security threats.

**What are the disadvantages of using *selfdestruct*?**

**The disadvantages of the *selfdestruct* function are that all transferable funds should be in ETH and any funds sent to the contract after it is destroyed are lost.**

Typically, *selfdestruct* functions are called by developers who fail to communicate about it on time. Even when a team communicates the update, it does not circulate fast enough. As a result, some people might end up sending funds to the destroyed contract thinking it’s still active. In such a case, the funds are lost.

The second disadvantage of using *selfdestruct* is that it can only transfer Ether (ETH) and not other ERC-20 tokens such as altcoins or NFTs which follow the ERC-721 token standard. Once *selfdestruct* is called, these assets can never be recovered.

**Why is *selfdestruct* considered a double-edged sword function?**
**The *selfdestruct* function is a double-edged sword because it gives away the power to the core team to modify immutable contracts even after they have been deployed.**
First, adding a *selfdestruct* function gives an easier avenue for a project's core team to rug their users or community. Since the developers have access to the contract, they can call the *selfdestruct* function and direct the funds to their personal wallets.
Secondly, the main intent behind using the *selfdestruct* keyword in a contract is to provide an escape route during hacks or to replace it with a better contract. However, malicious actors can use this as an exploit.

**How does *selfdestruct* work?**
***Selfdestruct* works by erasing the contract bytecode from the chain, sending any liquidity to a specified address, and then, refunding a portion of the gas fees to developers.**
A contract consists of two components: state and functions. These components define contract behavior and **callable functions**. Removing the bytecode means that the contract has no components that define it, rendering the contract uncallable.
When *selfdestruct* is called, the function caller has to input an alternative address for the funds. This functionality can become a potential security flaw if the contract is exploited to trigger the *selfdestruct* function.
Once the function call is performed, Ethereum reverses a portion of the transaction fees called negative gas.

**What is negative gas and why is it relevant?**
**Negative gas refers to a part of the transaction fee that the Ethereum chain refunds whenever the developers use the *selfdestruct* function.**
Since cleaning up the contract code from a blockchain is beneficial to the chain's health, the Ethereum community uses negative gas to incentivize using the *selfdestruct* function.
When a contract calls the *selfdestruct* function, half of the total gas used for the transaction is refunded to the calling function.

**Does *selfdestruct* remove the contract's history on the Ethereum blockchain?**
**No, the *selfdestruct* function does not remove the history of a contract from the Ethereum chain — it only removes the contract's bytecode.**
Ethereum is a blockchain, a public ledger, where a network of interconnected nodes always keep a copy of the blockchain's state. Therefore, all the data and transactions done before calling the *selfdestruct* function are permanently recorded on-chain and cannot be altered.

**Are funds permanently lost after calling the *selfdestruct* function?**
**No, existing funds in the contract are not permanently lost when the *selfdestruct* function is called — they are sent to another address specified by the function caller.**
However, if anyone sends funds to a selfdestructed contract (i.e. terminated contract) those funds cannot be recovered because the contract code has been deleted from the chain.

> *How to check if a contract is selfdestruct?*
> *You can call getCode (web3.eth.getCode(address)). The call will return 0x if it selfdestruct'd. Note that this isn't a positive way to know it selfdestruct'd, it'll also return 0x if it's an EOA or the contract was created with no code.*

# Ethernaut Fallout

That was silly wasn't it? Real world contracts must be much more secure than this and so must it be much harder to hack them right?

Well... Not quite.

The story of Rubixi is a very well known case in the Ethereum ecosystem. The company changed its name from 'Dynamic Pyramid' to 'Rubixi' but somehow they didn't rename the constructor method of its contract:

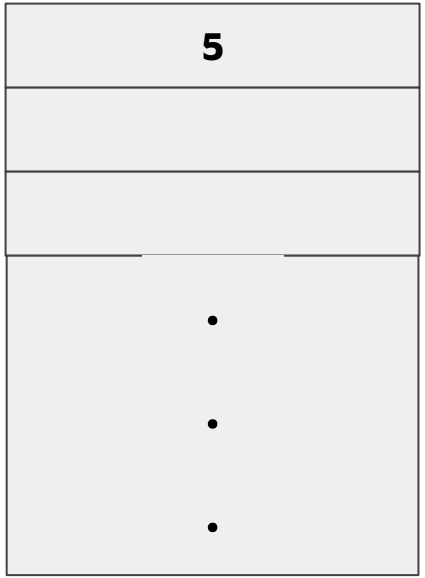
```
contract Rubixi {
  address private owner;
  function DynamicPyramid() { owner = msg.sender; }
  function collectAllFees() { owner.transfer(this.balance) }
  ...
```

This allowed the attacker to call the old constructor and claim ownership of the contract, and steal some funds. Yep. Big mistakes can be made in smartcontractland.

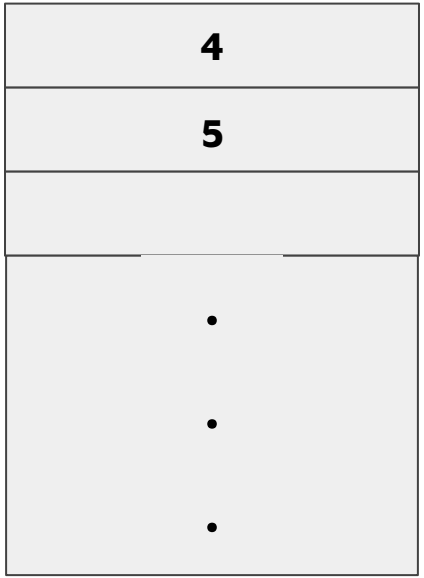
# Bytecode Execution by EVM

- Bytecode Snippet
  - **6005600401**
- Opcode Translation
  - **PUSH1 0x05**
  - **PUSH1 0x04**
  - **ADD**

# Stack

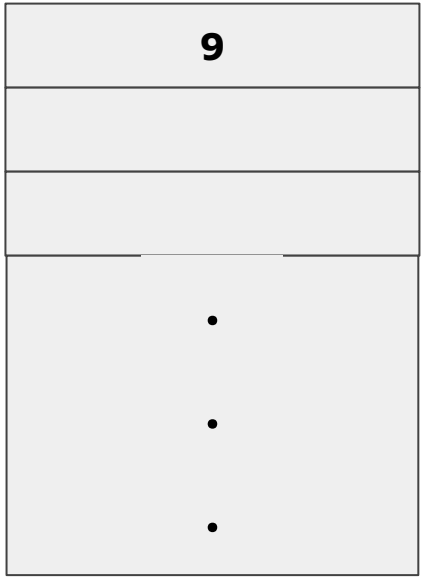



PUSH1 0x05




PUSH1 0x05

PUSH1 0x04




PUSH1 0x05

PUSH1 0x04

ADD

## Storage

[0] 0x00…19

[1] 0x00…01

[2]

[3]

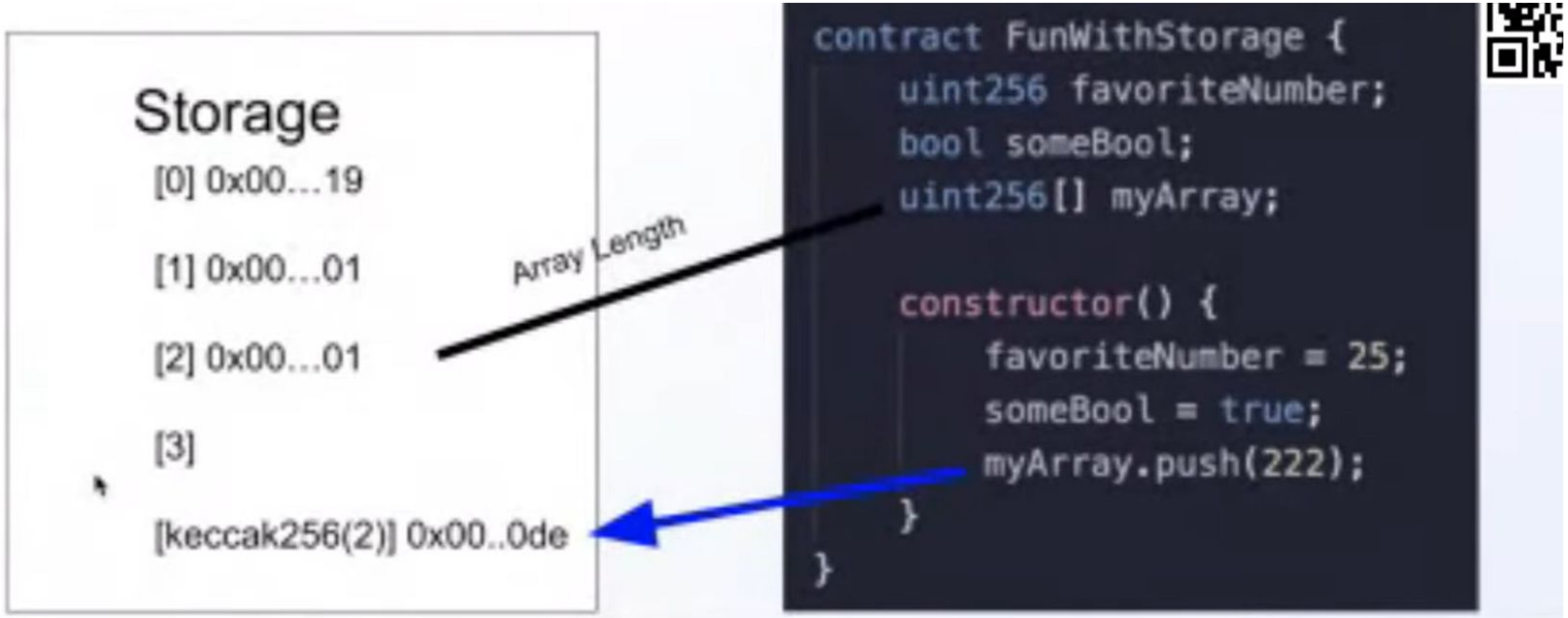
[4]

```
contract FunWithStorage {
    uint256 favoriteNumber;
    bool someBool;

    constructor() {
        favoriteNumber = 25;
        someBool = true;
    }
}
```

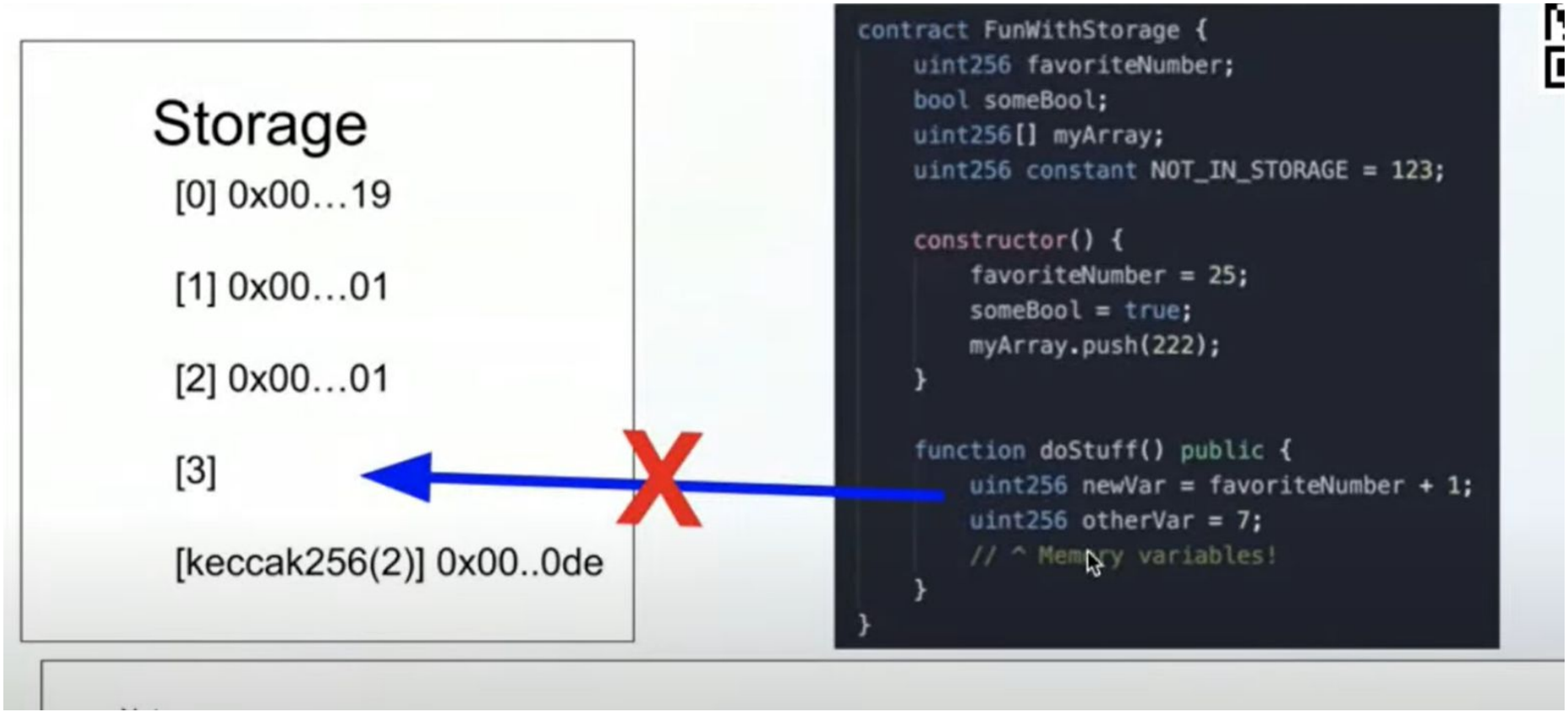
Notes:

- Each slot is 32 bytes long, and represents the bytes version of the object
  - For example, the uint256 25 is 0x000…0019, since that's the hex representation
  - For a "true" boolean, it would be 0x000…001, since that's it's hex




Notes:

- Each slot is 32 bytes long, and represents the bytes version of the object
  - For example, the uint256 25 is 0x000...0019, since that's the hex representation
  - For a "true" boolean, it would be 0x000...001, since that's it's hex
- For dynamic values like mappings and dynamic arrays, the elements are stored using a hashing function. You can see those functions in the documentation.
  - For arrays, a sequential storage spot is taken up for the length of the array.
  - For mappings, a sequential storage spot is taken up, but left blank

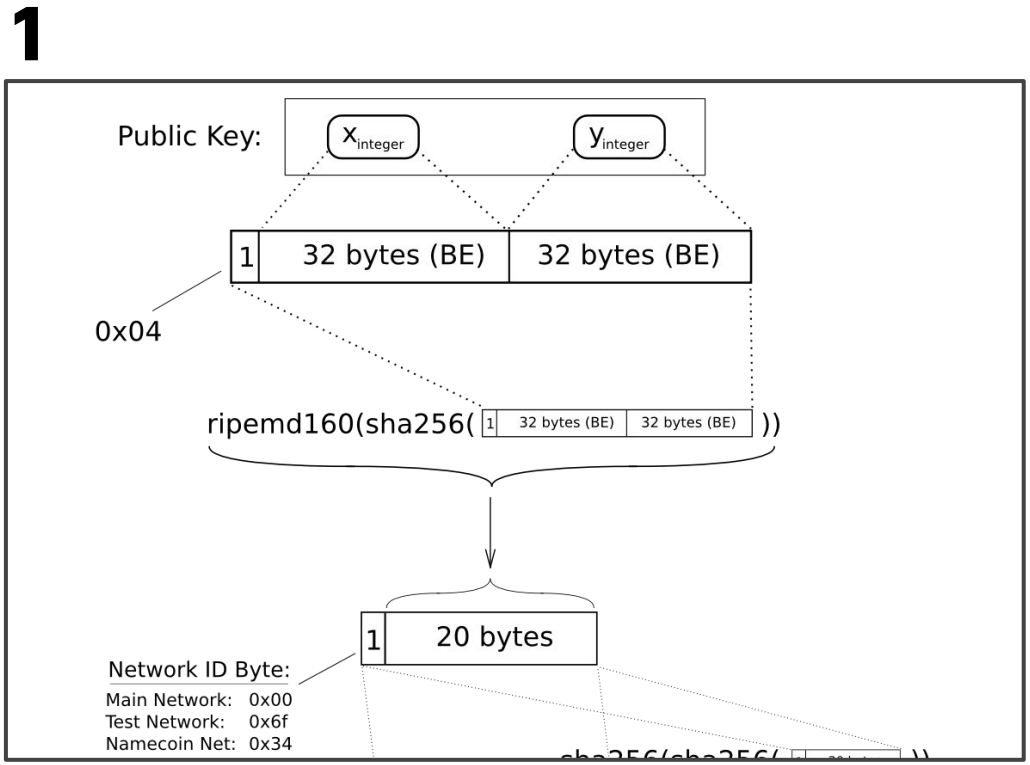

Storage
[0] 0x00…19
[1] 0x00…01
[2] 0x00…01
[3]
[keccak256(2)] 0x00..0de
contract FunWithStorage {
uint256 favoriteNumber;
bool someBool;
uint256[] myArray;
uint256 constant NOT_IN_STORAGE = 123;
constructor() {
favoriteNumber = 25;
someBool = true;
myArray.push(222);
}
function doStuff() public {
uint256 newVar = favoriteNumber + 1;
uint256 otherVar = 7;
// ^ Memory variables!
}
}

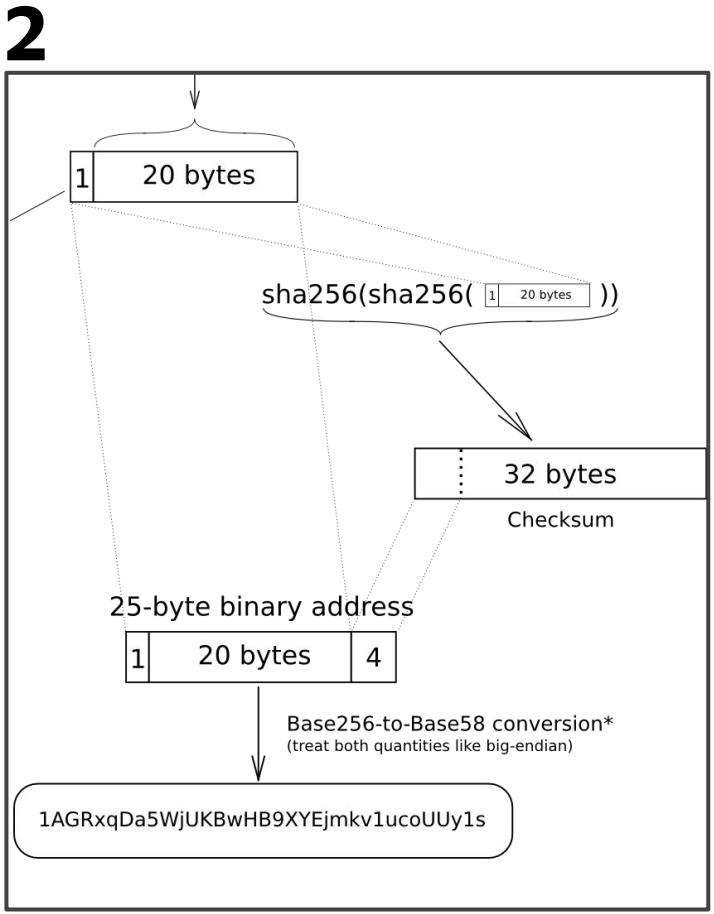
## Public Key to Address

Bitcoin and Ethereum both have a transformation process to take a public key and turn it into an address. For Bitcoin it includes a checksum and Base58 encoding. Ethereum's address transformation is quite a bit simpler, its address is the last 20 bytes of the hash of the public key.

The important thing to recognize here is that the address is differentiated from the public key, but you can always derive the address if you have the public key.

# Elliptic-Curve Public Key to BTC Address conversion



# Crypto Exchanges

| | DEX | CEX |
|---|---|---|
| Trading | Peer-to-peer basis | Centralized intermediary |
| KYC | Often not needed | Indispensable |
| Security | Backed by blockchain | More fragile to attacks |
| Convinience | Problematic for newcomers | User-friendly |
| Private keys | Kept by users | Owned by exchange |
| Tokens | Far more options | Limited selection |

# Automated Market Makers [AMM]

Traditional exchanges work on order-book model. Buyers and sellers place their orders.

If you need to buy something at a certain price, there should be a seller willing to sell at that price, and vice versa.

eg: Robinhood, Nasdaq, Bombay Stock Exchange are based on order-book model.

Crypto decentralised exchanges (DEXes) like Uniswap, Sushiswap or Pancakeswap are based on Automated Market Maker or AMM model.

An AMM model has 'Liquidity Pools' of different pairs of assets. eg: ETH-USDC pool. People can buy and sell assets from/to this pool.

Who creates these pools? Liquidity Providers. They get certain percentage of trading fee for doing so. Anyone can contribute to a pool and become a liquidity provider.

To create an ETH-USDC liquidity pool, we need to add the same value of both assets. eg: If 1 ETH = 1k USDC, then we can create create a pool of 10 ETH + 10k USDC.

In reality, this pool is much larger having much higher amount of each asset, but let's take 10 ETH + 10k USDC to understand.

Thus, we see that the value of each asset is same in the pool i.e. 10,000.

To buy ETH with USDC, you can add USDC to the pool, and in return, you'll get some ETH.

But now, since the amount of ETH gets lesser as compared to USDC, its value increases in the pool (>1k). This is because of something called 'Constant Product Formula'.

The increased ETH price creates 'arbitrage' opportunity for traders to buy ETH for cheap from somewhere else and sell it at a higher price in the pool. Thus, when they sell ETH to the pool, they add ETH and withdraw USDC, hence restoring the balance.

# Automated Market Makers [AMM]

But what is constant product formula?

It says that the product of amount of both assets in the pool should always remain same.

i.e. 10 ETH * 10,000 USDC = 100,000

Hence, the product of ETH and USDC amount should always be 100,000

So initially, if you want to buy 1 ETH, you will reduce it from 10 to 9. Hence, the new USDC amount in pool should be 100,000/9 = 11,111.

i.e. you'll need to pay 11,111 - 10,000 = 1,111 USDC for 1 ETH.

If you want to buy an ETH again, ETH amount becomes 8. Hence, USDC amount in pool should be 100,000/8 = 12,500.

i.e. you'll pay 12,500 - 11,111 = 1,389 USDC for 1 ETH.

Thus, we see that more the supply of an asset decreases in the pool, more its price increases due to the constant product formula.

In reality, the pools are much larger, hence the price impact of buying and selling 1 ETH would be much lesser on the pool.

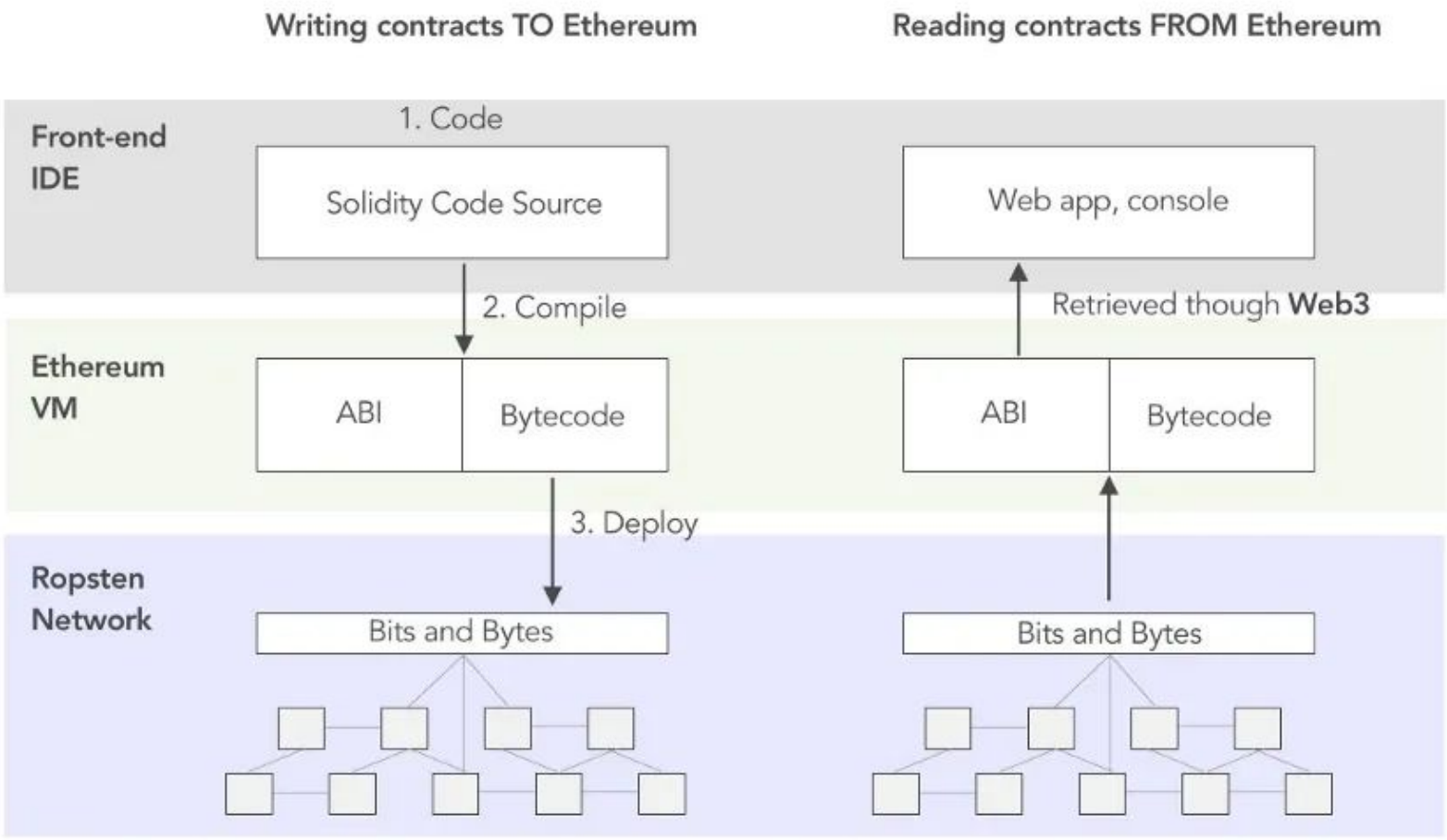
Also, arbitrage opportunities make sure that the prices are corrected to its fair value.

# Coins vs Tokens

- Digital coins are generally used in the same way as a real-life coin is – as money.
- Rely on own blockchain for security and governance.
- Bitcoin BTC
- Ethereum ETH
- THETA, TFUEL

- Tokens are created on existing blockchains.
- Do not have to worry about building and maintaining a Blockchain.
- Chainlink [LINK]
- Tether [USDT]
- Basic Attention Token [BAT]

# Types of Crypto Offerings


Writing contracts TO Ethereum
Reading contracts FROM Ethereum
Front-end
IDE
1. Code
Solidity Code Source
Web app, console
2. Compile
Retrieved though Web3
Ethereum
VM
ABI
Bytecode
ABI
Bytecode
3. Deploy
Ropsten
Network
Bits and Bytes
Bits and Bytes

# Whitepaper vs Yellowpaper

## Whitepaper

- Vision Document
- To persuade potential investors and customers to learn about the service or technology.
- Contains a Problem Statement
- Provide an overview of the solution the stated problem.

## Yellowpaper

- More of Technical Document
- Referred by the developers of the project.
- Details out the execution plan for problem statement
- Contains mathematical formula and code sample if necessary.

# Ethereum Whitepaper Breakdown

## Background

*“I thought [those in the Bitcoin community] weren’t approaching the problem in the right way. I thought they were going after individual applications; they were trying to kind of explicitly support each [use case] in a sort of Swiss Army knife protocol.“*

- *Vitalik Buterin*

*“Rather than being a closed-ended, single purpose protocol intended for a specific array of applications in data storage, gambling or finance, Ethereum is open ended by design, and we believe that it is extremely well-suited to serving as a foundational layer for a very large number of both financial and non-financial protocols in the years to come.”*

- *Vitalik Buterin*

# Ethereum Whitepaper Breakdown

## Background

Vitalik noted that developers were using three limited approaches to building applications. They were either

- building a new blockchain, or
- using scripting on top of Bitcoin, or
- building a meta-protocol on top of Bitcoin.

These approaches all came with limitations.

# Ethereum Whitepaper Breakdown

## Solution

Building a general purpose blockchain which can address to range of problem over creating specific solutions for each problem.

*“Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats and state transition functions.”*

### Decentralized Application

- A decentralized application or Dapp for short, in this context, refers to an application that is built on top of blockchain technology.
- They run on a blockchain and benefit from all of its properties like immutability, security, tamper resistance and zero downtime.
- For Instance, Bitcoin is a decentralized cryptocurrency application for payments.

# Ethereum Whitepaper Breakdown

## Solution - 5 Key Elements

- **Smart Contracts**
  - Bundle of code that manages the exchange of anything of value from property and shares to information and money between parties
- **The Ethereum Virtual Machine**
  - The EVM turns Ethereum into a programmable blockchain, keeping all the smart contracts running on time and coordinating them with the rest of the network
- **Solidity**
  - enables developers to write programs (smart contracts) on Ethereum and is designed to enhance the Ethereum Virtual Machine (EVM)
- **Ether**
  - A fuel that provides an incentive to ensure developers write quality applications and the network runs smoothly
- **Proof of Work**
  - Like the Bitcoin network, Ethereum relies on Proof of Work protocol to reach consensus about transaction in the network.

# Ethereum Whitepaper Breakdown

## What can Ethereum be used for?

### Financial applications
This includes sub-currencies, financial derivatives, hedging contracts, savings wallets, wills, and ultimately even some classes of full-scale employment contracts.

### Semi-financial applications
Where money is involved but there is also a heavy non-monetary side to what is being done; a perfect example is self-enforcing bounties for solutions to computational problems.

### Non financial applications
Applications such as online voting and decentralized governance that are not financial at all.

**Token Systems**

**Financial Derivatives**

**Identity and Reputation Systems**

**Decentralized File Storage**

**Decentralized Autonomous Organizations**

**And more...**

# Ethereum Foundation


Ethereum Foundation is a non-profit organization dedicated to supporting Ethereum and related technologies.

Their role is not to control or lead Ethereum, nor they are the only organization that funds critical development of Ethereum-related technologies.

The Ethereum Foundation's mission is to promote and support Ethereum platform and base layer research, development and education to bring decentralized protocols and tools to the world that empower developers to produce next generation decentralized applications (dapps), and together build a more globally accessible, more free and more trustworthy Internet.

# Ethereum Foundation Features and Services

**Smart money, smart wallet.**
The Ethereum Wallet is a gateway to decentralized applications on the Ethereum blockchain. It allows you to hold and secure ether and other crypto-assets built on Ethereum, as well as write, deploy and use smart contracts.

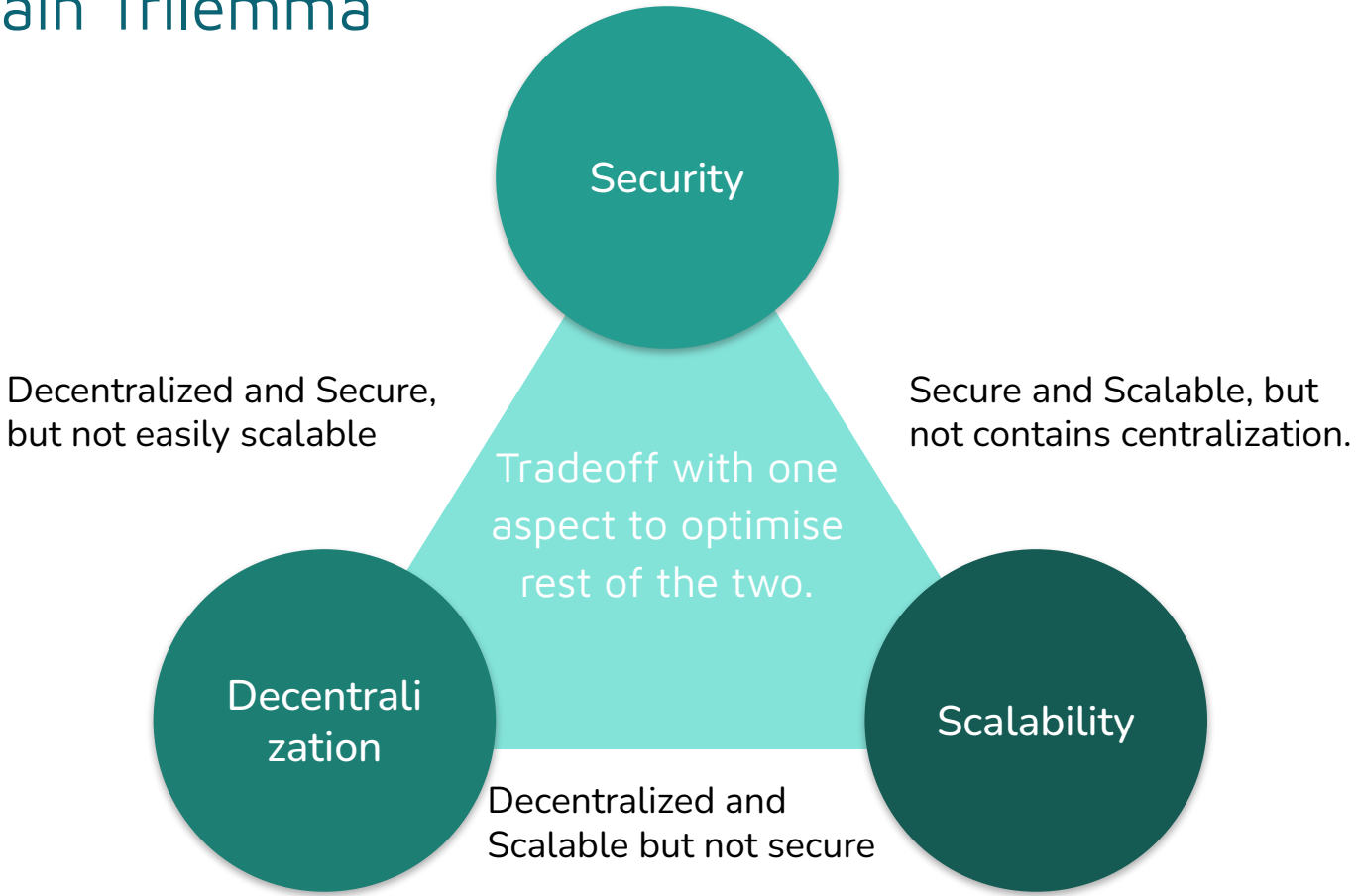
**Build Unstoppable Applications.**
Ethereum is a decentralized platform that runs smart contracts: applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference.

**Learn Solidity, A New Language for Smart Contracts.**
Create a tradeable digital token that can be used as a currency, a representation of an asset, a virtual share, a proof of membership or anything at all.

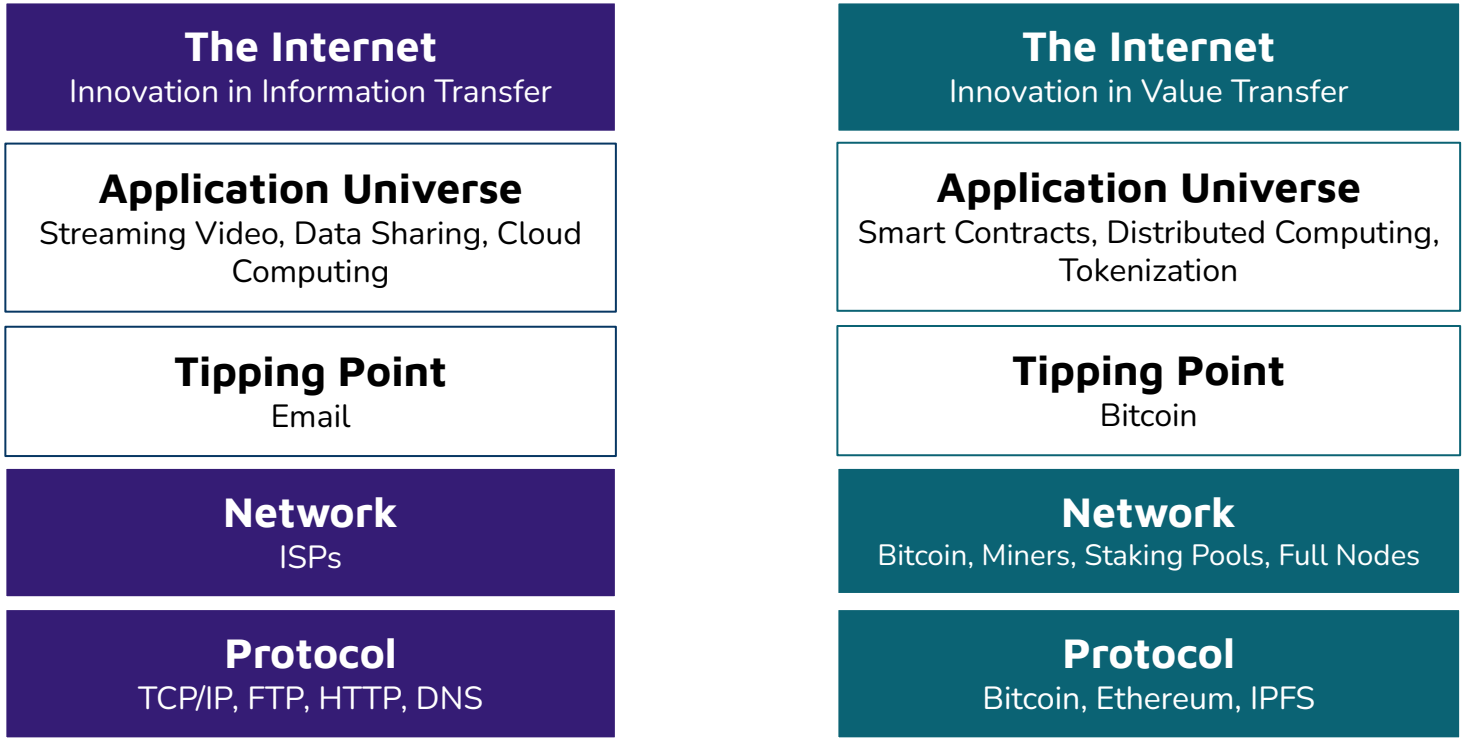
**Kickstart a Project with a Trustless Crowdsale.**
You can build a crowdfund to pre-sell a product, a crowdsale to sell virtual shares in a blockchain organization, and an auction of a limited number of items.

**Create a Democratic Autonomous Organization.**
You can build a virtual organization where members vote on issues, a transparent association based on shareholder voting, your own country with an unchangeable constitution, and a better delegative democracy.

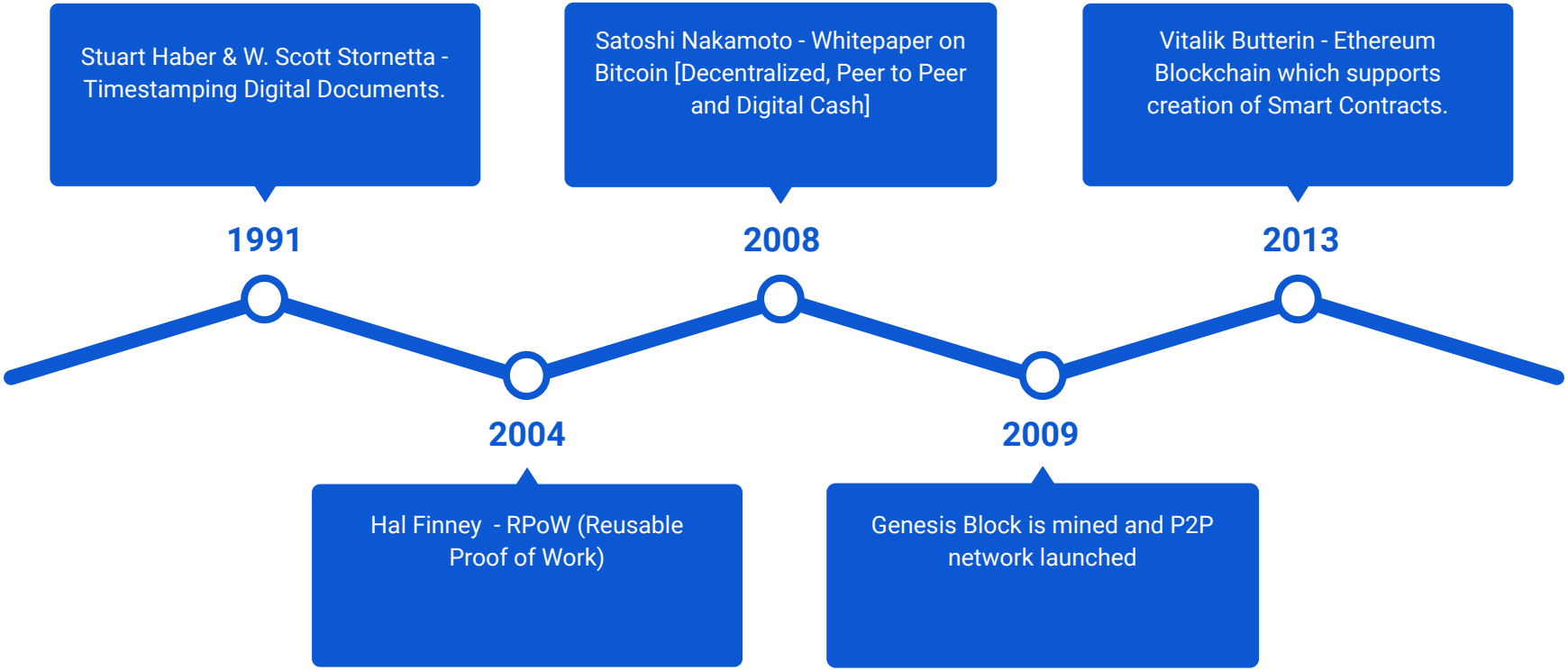
# Blockchain Trilemma



# Blockchain Infrastructure Stack



# **History** of Blockchain



# Transaction Validation

1. Intent: “I want to swap X for Y”
2. Compose: “Here is how I plan to achieve my intent.”
3. Price: “To do this, how much do I expect to pay in transaction fees?”
4. Preview: “What would happen if I submitted this transaction right now?”
5. Transmit: “I have signed and submitted my transaction to the network.”
6. Monitor: “What is happening with my transaction? Is it going to be confirmed soon?”
7. Finality: “Did my transaction succeed? What were the outcomes?”

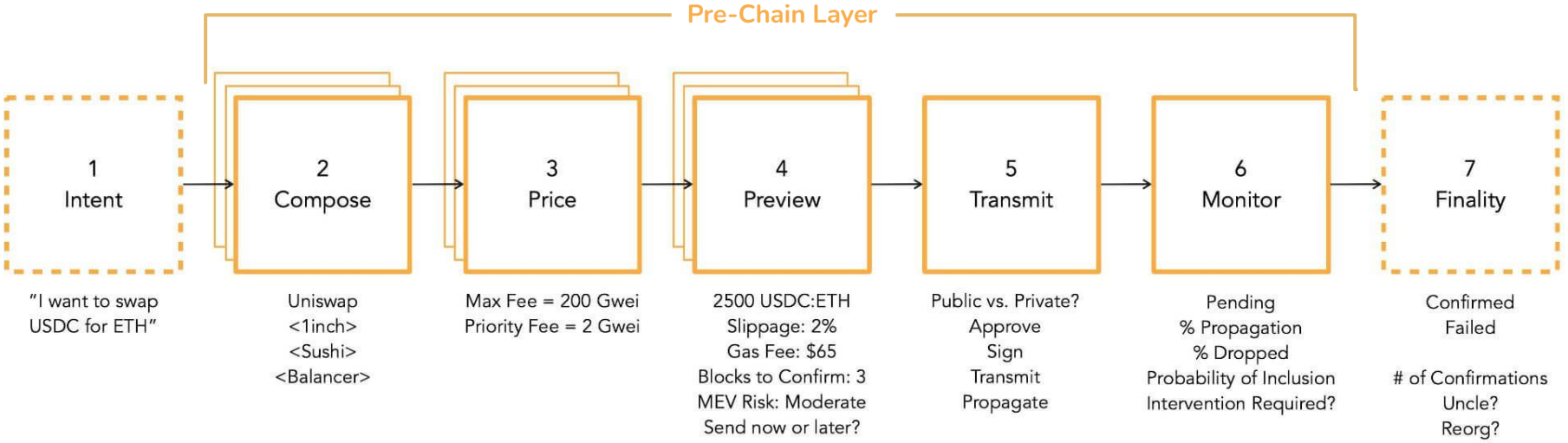
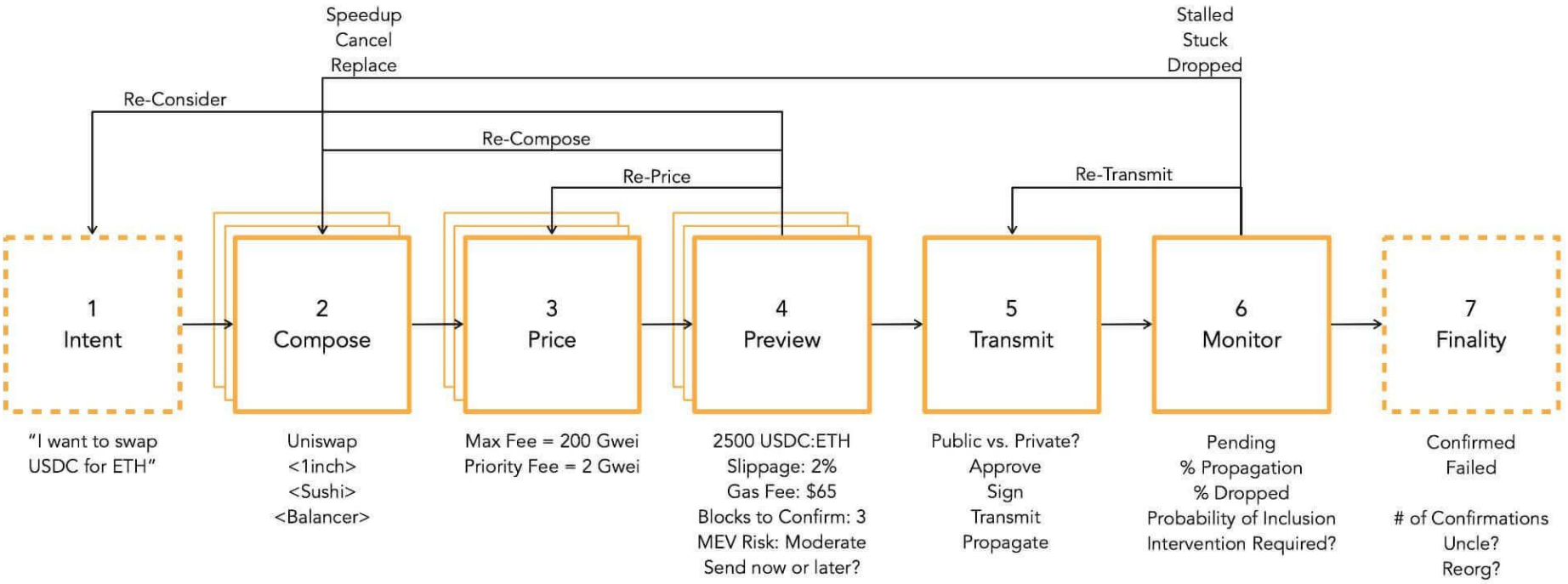
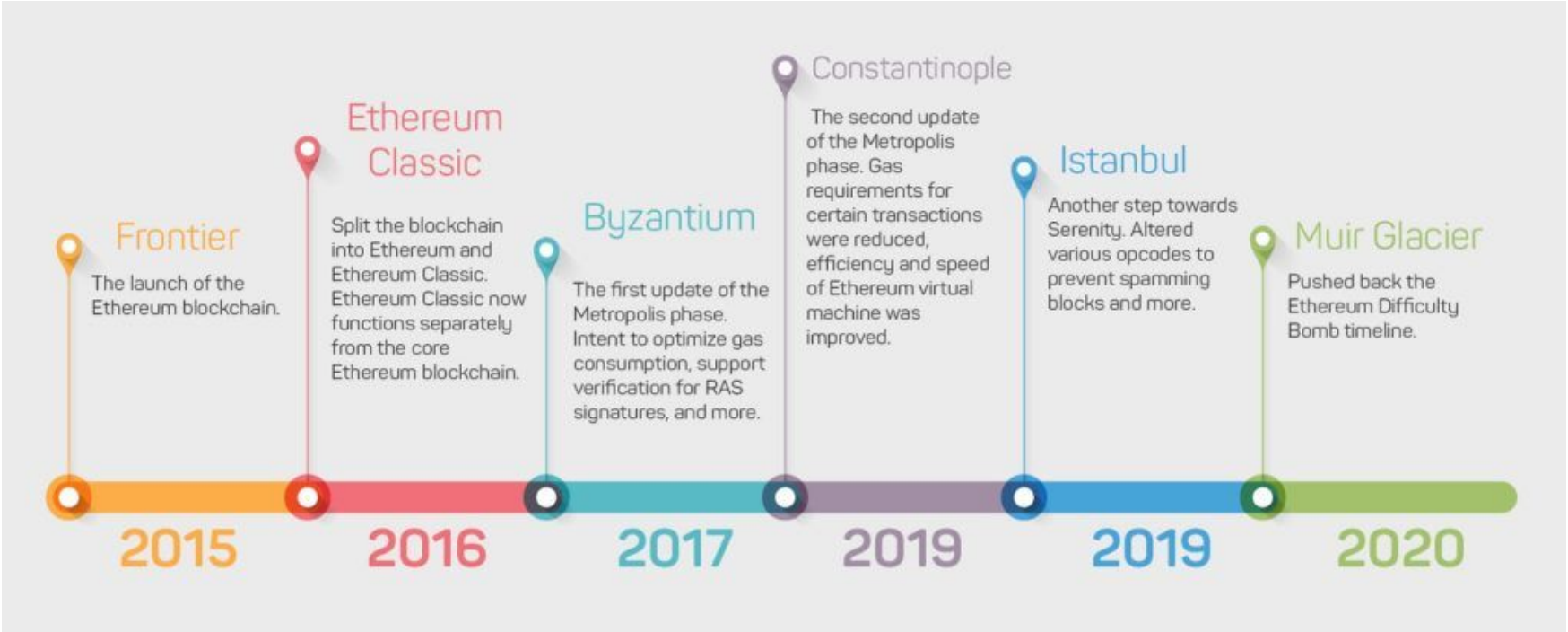



Image Credits: Block Native

# Transaction Validation

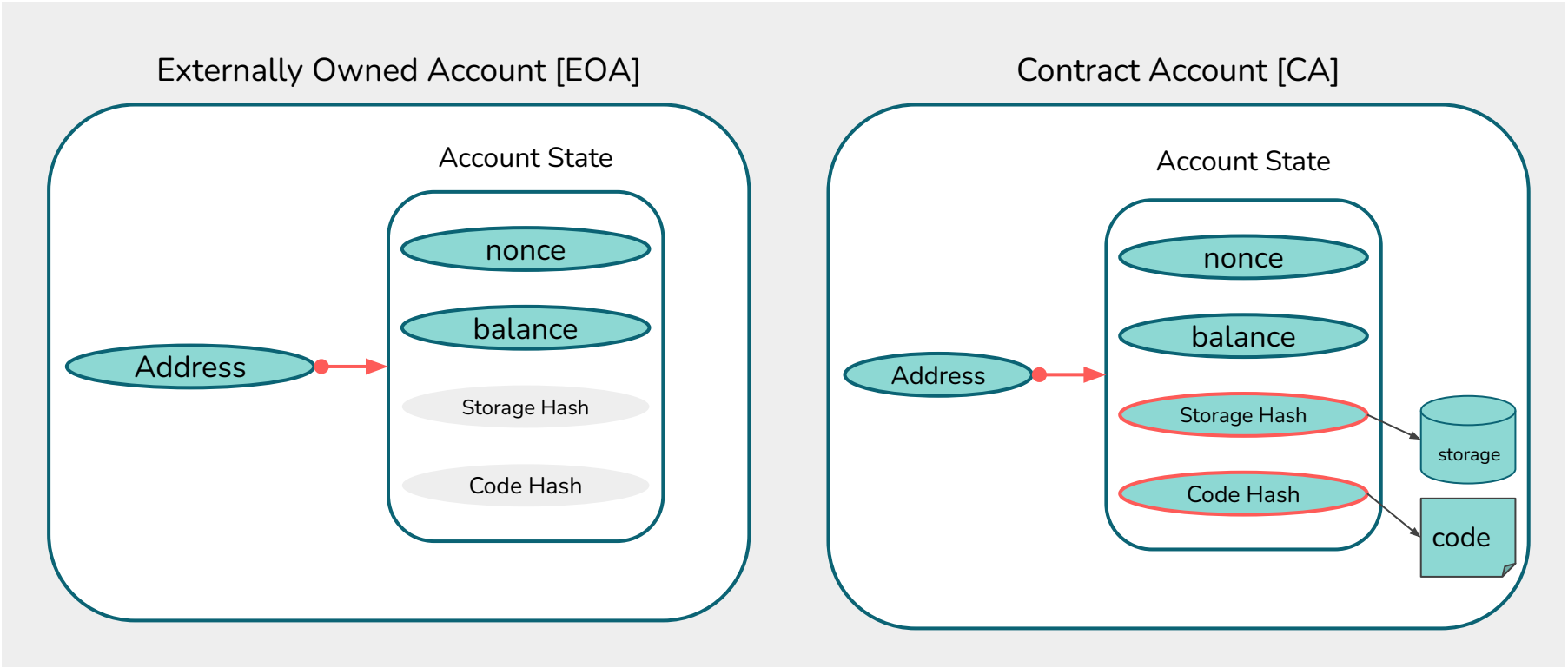


# History of Ethereum

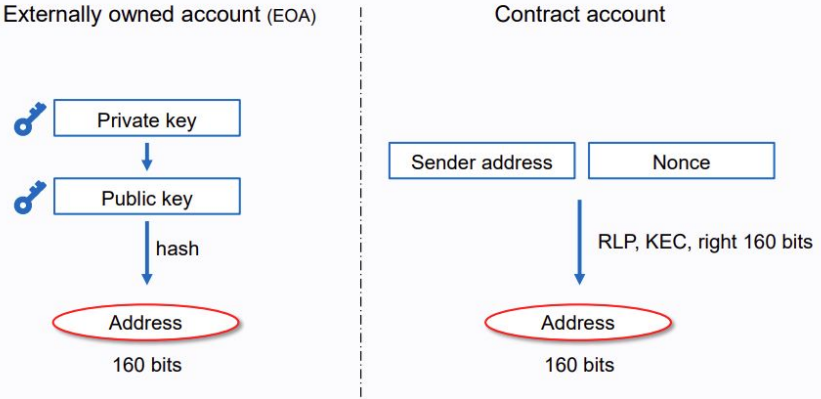


# Ethereum Accounts

World State

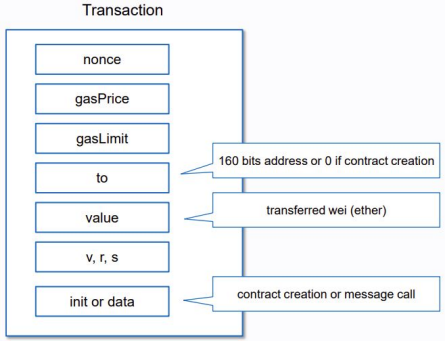


# Address of account




A 160-bit code used for identifying accounts.

## Field of a transaction

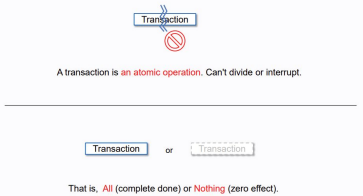



## Message

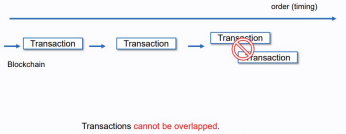

Message is passed between two Accounts.

Message is Data (as a set of bytes) and Value (specified as Ether) .

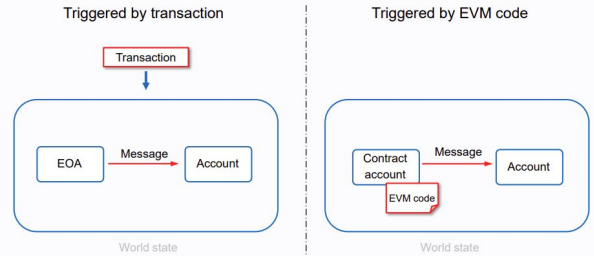
## Atomicity of transaction




## Order of transactions



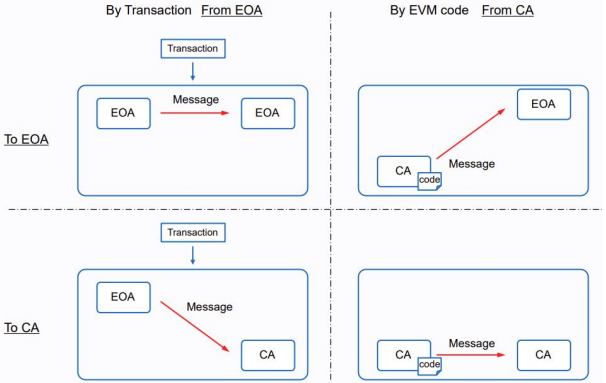

## Message



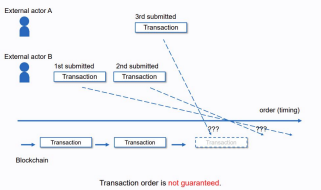

Transaction triggers an associated message.

EVM can also send a message.

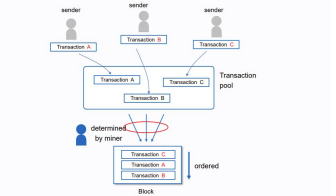
## Four cases of message



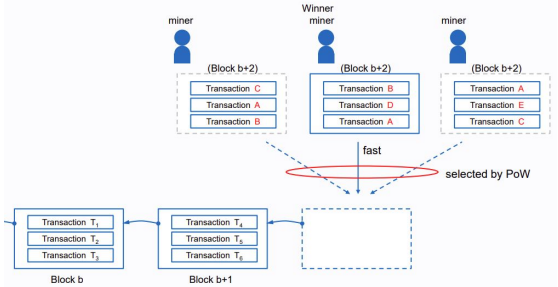

## Order of transactions




## Ordering inner block

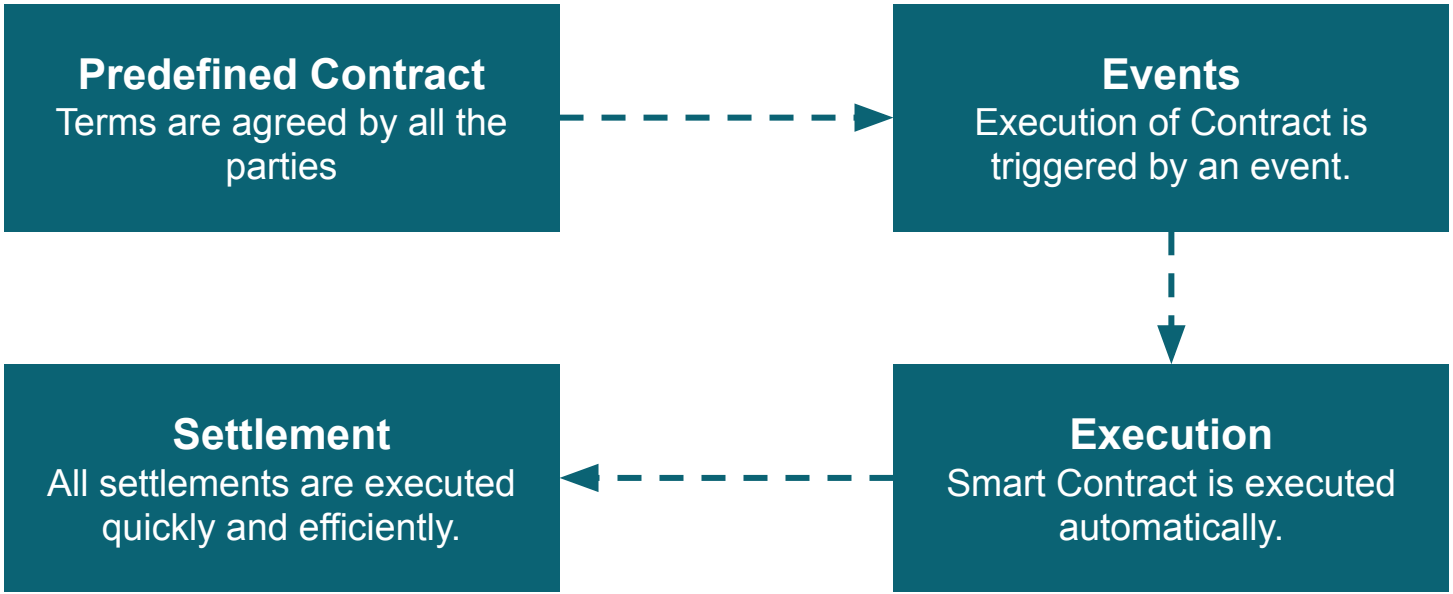


# Ordering inter blocks




The order between blocks is determined by a consensus algorithm such as PoW.

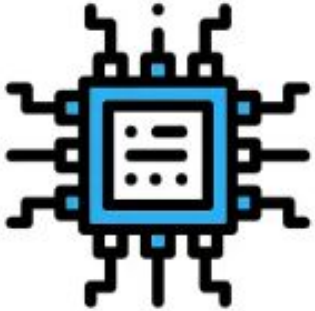
# How does Smart Contracts work?



1

Smart Contracts are **written as code** and committed to the blockchain. The code and conditions in the contract are **publicly available** on the ledger.

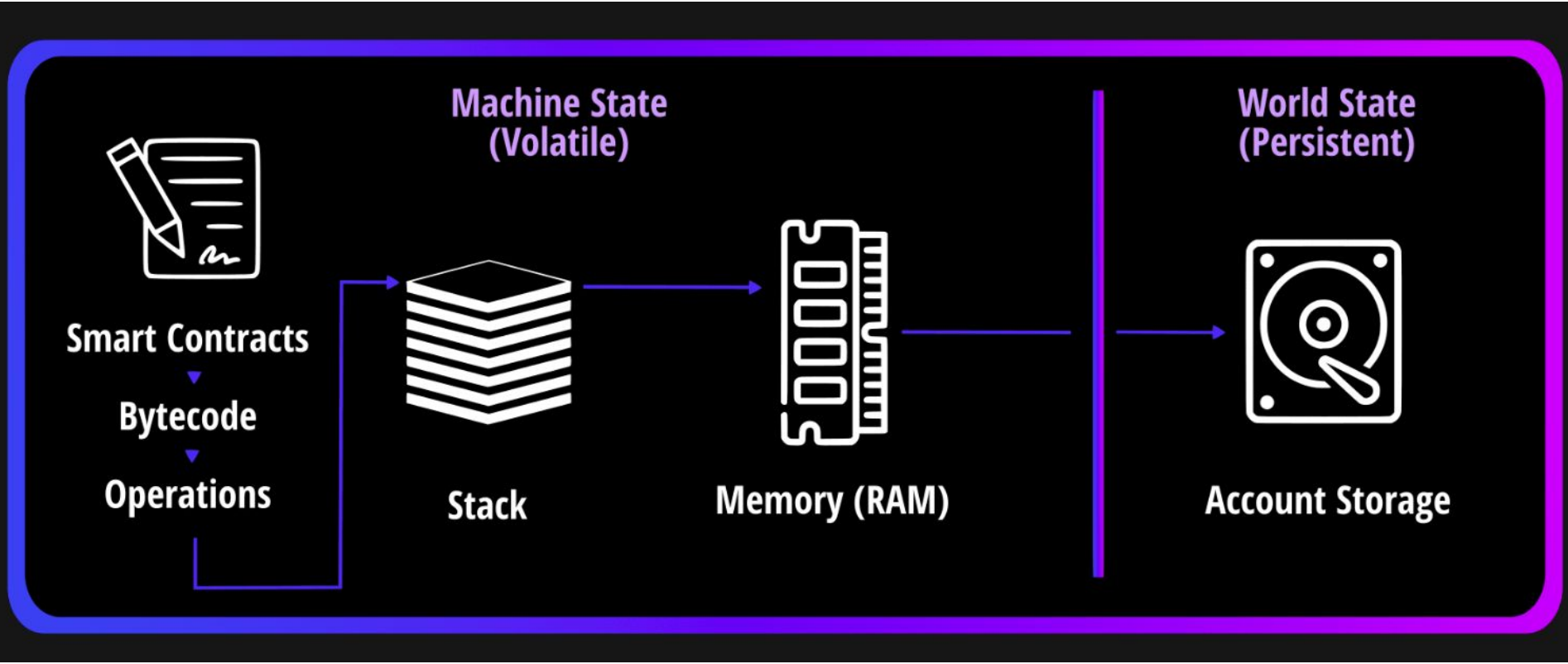
2



When an event outlined in the contract is triggered, like an expiration date or an asset's target price is reached-- the **code executes**.

3

Regulators can watch contract activity on the blockchain to **understand the market** while still **maintaining the privacy** of individual actors.


Machine State
(Volatile)
World State
(Persistent)
Smart Contracts
Bytecode
Operations
Stack
Memory (RAM)
Account Storage

# Byte Code and Contract State

## ByteCode

- For effective storage of opcodes, they are encoded to Bytecode.
- Every opcode is allocated a byte. (Ex. STOP 0x00)

## Contract State

- Most high level languages allow users to directly pass arguments to functions[foo(arg1, arg2)], low level languages use stack to pass values to functions.
- EVM uses 256-bit register stack from which the most recent 16 items can be accessed or manipulated at once. In total stack can only hold 1024 items.
- Because of these limitations, opcodes instead use contract **memory** to retrieve or pass data. **Memory** is not persistent. When contract execution finishes, the **memory** contents will not be saved.
- To store data indefinitely, make it accessible for future contract executions, one can use **Storage**. Contract **storage** acts as a public database, from which values can be read externally without having to send a transaction to the contract(no fees). However, writing to **storage** is very expensive compared to writing to memory.

# Cost of Interacting with Smart Contracts

- All contract executions are run by everyone running an Ethereum Node, an attacker could try creating contracts including lots of computationally expensive operations to slow down the network.
- To prevent such attacks from happening, every opcode has its own base gas cost.
- Some complex opcodes have dynamic gas costs.
- When executing instructions which reduce state size, gas can also be refunded. Setting a storage value to zero from non-zero refunds 15000 gas, while completely removing a contract (using the SELFDESTRUCT opcode) refunds 24000 gas.

# Deploying a Smart Contract

- When deploying a smart contract, a regular transaction is created without a ***to*** address.
- Additionally, some bytecode is added as input data.
- This bytecode acts as a constructor, which is needed to write initial variables to storage before copying the runtime bytecode to the contract's code.
- During deployment, creation bytecode will only run once, while runtime bytecode will run on every contract call.

```
pragma solidity 0.5.3;

contract ExampleContract {
    uint256 number = 1;
}
```

```
0x6080604052600160005534801560145760600080fd5b506035806022600039600f3fe6080604052600080fdfea165627a7a723058204e048d6cab20eb0d9f95671510277b55a61a582250e04db7f6587a1bebc134d20029
```

# Deploying a Smart Contract [Contd]

The Bytecode is split into 3 parts.

**Constructor**

6080604052600160005534801561457600080fd5b506035806022600039600f3fe

```
PUSH1 0x80
PUSH1 0x40
MSTORE
PUSH1 0x1
PUSH1 0x0
SSTORE
CALLVALUE
DUP1
ISZERO
PUSH1 0x14
JUMPI
PUSH1 0x0
DUP1
REVERT
JUMPDEST
POP
PUSH1 0x35
DUP1
PUSH1 0x22
PUSH1 0x0
CODECOPY
PUSH1 0x0
RETURN
INVALID
```

```
MSTORE(0x40,0x80)              /* Reserve 64 bytes of memory at location 32                  */

SSTORE(0x0,0x1)                /* Write 1 to storage at slot 0 (this is our number variable) */

JUMPI(0x14,ISZERO(CALLVALUE))  /* Jump to location 20 if the included amount of ether is zero */

REVERT(0x0,0x0)                /* If the jump didn't occur, halt execution (revert creation) */

JUMPDEST                       /* Marks valid jump location for the JUMPI                    */

CODECOPY(0x0,0x22,0x35)        /* Copy 53 bytes of code at position 34 to memory at location 0 */

RETURN(0x0,0x0)                /* Return without any values                                  */

INVALID                        /* Make sure runtime bytecode is never reachable by the EVM   */
```

# Deploying a Smart Contract [Contd]

**Runtime**
6080604052600080fdfe

```
PUSH1 0x80
PUSH1 0x40
MSTORE
PUSH1 0x0
DUP1
REVERT
```

```
MSTORE(0x40,0x80)    /* Reserve 64 bytes of memory at location 32    */

REVERT(0x0,0x0)      /* Halt execution and revert transaction        */
```

**Metadata**
a165627a7a723058204**e048d6cab20eb0d9f95671510277b55a61a582250e04db7f6587a1bebc134d2**0029

- At the end of this bytecode, a Swarm [decentralized file storage] hash of a metadata file created by Solidity gets appended. Although the Swarm hash will also be included in the runtime bytecode, it will never be interpreted as opcodes by the EVM.
- Solidity utilizes the format: 0xa1 0x65 'b' 'z' 'z' 'r' '0' 0x58 0x20 [32 bytes swarm hash] 0x00 0x29
- Swarm Hash here: **4e048d6cab20eb0d9f95671510277b55a61a582250e04db7f6587a1bebc134d2**

# Decompiling Bytecode

- Contract calls usually require an “ABI” (Application Binary Interface), which is a piece of data documenting all functions and events, including their needed input and output.
- When calling a function on a contract, the function signature is determined by hashing the name of the function including its inputs (using keccak256), and truncating everything but the first 4 bytes.

Solidity

```
pragma solidity 0.5.3;

contract ExampleContract {
    function HelloWorld() public returns(string memory) {
        return "Hello World!";
    }
}
```

```
> keccak256("HelloWorld()");
"7fffb7bdf7d16635144da549e9a4eedff43ed43d64e49e18d7e365f9e5521232"
```

Function signature

ABI

```
[{
    "constant": false,
    "inputs": [],
    "name": "HelloWorld",
    "outputs": [{
        "name": "",
        "type": "string"
    }],
    "payable": false,
    "stateMutability": "nonpayable",
    "type": "function"
}]
```

# Decompiling Bytecode

- Function HelloWorld() resolves to the signature hash 0x7fffb7bd.
- If we would like to call this function, our transaction data needs to start with 0x7fffb7bd.
- Arguments which need to be passed to a function (none in this case) can be added in 32-byte pieces called words after the signature hash in a transaction's input data.
- If an argument contains over 32 bytes (256 bits) of data, like an array or string, the argument is split into multiple words which are added to the input data after all other arguments have been included.
- The total size of all words gets included as another word, before all array words.
- At the location where the argument would have been included, the start position of the array words (including the size word) is added instead.

# Network Types

# Network Types

# Project

# Launch NFT Collection on Opensea

# Launching NFT Collections

1. Create Artwork
2. Store the Artwork on IPFS
3. Write and Deploy Smart Contract
4. Mint NFT
5. Trade NFTs