PIAIC

Introduction to Blockchain Technology

Blockchain and Crypto currencies are shaking the system

CNBC MARKETS BUSINESS INVESTING TECH POLITICS CNBC TV

TECH

Cryptocurrencies are ‘clearly shaking the system,’ IMF’s Lagarde says

PUBLISHED WED, APR 10 2019 • 9:15 PM EDT

President of the European Central Bank

Incumbent

Assumed office
1 November 2019

Vice President	Luis de Guindos
Preceded by	Mario Draghi

Managing Director of the International Monetary Fund

In office
5 July 2011 – 12 September 2019

Deputy	John Lipsky David Lipton
Preceded by	Dominique Strauss-Kahn
Succeeded by	Kristalina Georgieva

Blockchain and Crypto currencies are shaking the system

FINANCIAL TIMES

HOME WORLD US COMPANIES TECH MARKETS CLIMATE OPINION WORK & CAREERS LIFE & ARTS HOW TO SPEND IT

Opinion **Swift**

How blockchain is shaking Swift and the global payments system

A crucial linchpin of the world's financial plumbing is ripe for disruption

GILLIAN TETT + Add to myFT

Today Swift remains a non-profit co-operative, with 11,000 members and facilitating payments worth an eye-popping $1.5tn a day. It does this not by actually moving money, but by enabling banks to dispatch messages that credit or debit their accounts as payments occur.

Gillian Tett JULY 22 2021

Blockchain and Crypto currencies are shaking the system

Elon Musk: 'Paper money is going away'

Published Wed, Feb 20 2019 • 11:52 AM EST • Updated Mon, Apr 8 2019 • 10:21 AM EDT

Catherine Clifford
@CATCLIFFORD

Share

Blockchain and Crypto currencies are shaking the system

BANKING FEBRUARY 23, 2018 16:31

Bank of America Admits Cryptocurrencies Are a Threat to Its Business Model

Blockchain and Crypto currencies are shaking the system

News › Business

SEC Approves Fourth Bitcoin Futures ETF—But This One Is Different

The SEC has approved another Bitcoin futures ETF to start trading. Here's why that might be good news for a Bitcoin spot ETF.

By Stacy Elliott

Apr 8, 2022

3 min read

Investors can buy shares of an ETF to gain exposure to those securities without owning them directly.

In the case of Bitcoin ETFs, there have been two main types:

Bitcoin futures are derivative contracts speculating on the price of the cryptocurrency.

Bitcoin spot price is its current price.

Teucrium is the fourth Bitcoin futures fund to be approved, following Proshares (BITO), Valkyrie (BTF) and VanEck (XBTF) funds that all began trading late last year.

Blockchain and Crypto currencies are shaking the system

Bitcoin: El Salvador hosts 44 countries to encourage crypto adoption

President Nayib Bukele recently 'bought the dip' with the purchase of another 500 BTC

Anthony Cuthbertson • Monday 16 May 2022 13:15 • Comments

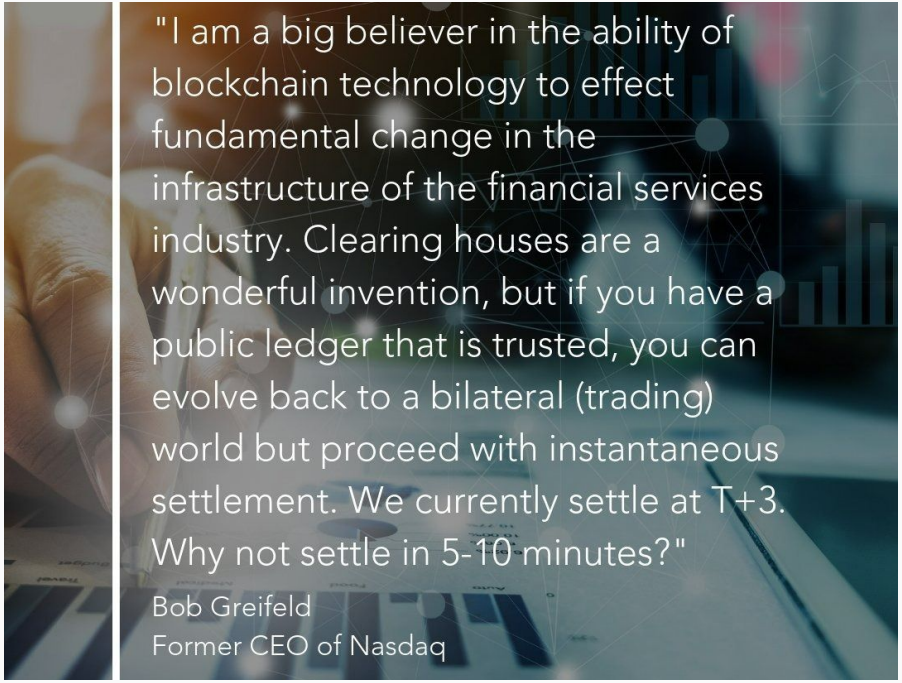
Eight months after becoming the first country in the world to adopt bitcoin as an official currency, El Salvador has invited 44 countries to discuss the "rollout and benefits" of the cryptocurrency.

32 central banks and 12 financial authorities (44 countries) met in El Salvador to discuss financial inclusion, digital economy, banking the unbanked, the bitcoin rollout and its benefits in our country,"

El Salvador became the first country in the world to adopt bitcoin as an official currency in Sep 2021

https://www.pwc.com/gx/en/financial-services/pdf/el-salvadors-law-a-meaningful-test-for-bitcoin.pdf

Blockchain and Crypto currencies are shaking the system



Blockchain and Crypto currencies are shaking the system

"Trust is the new currency, blockchain will enable that trust."
- Tim Vasko, Founder BlockCerts Int'l

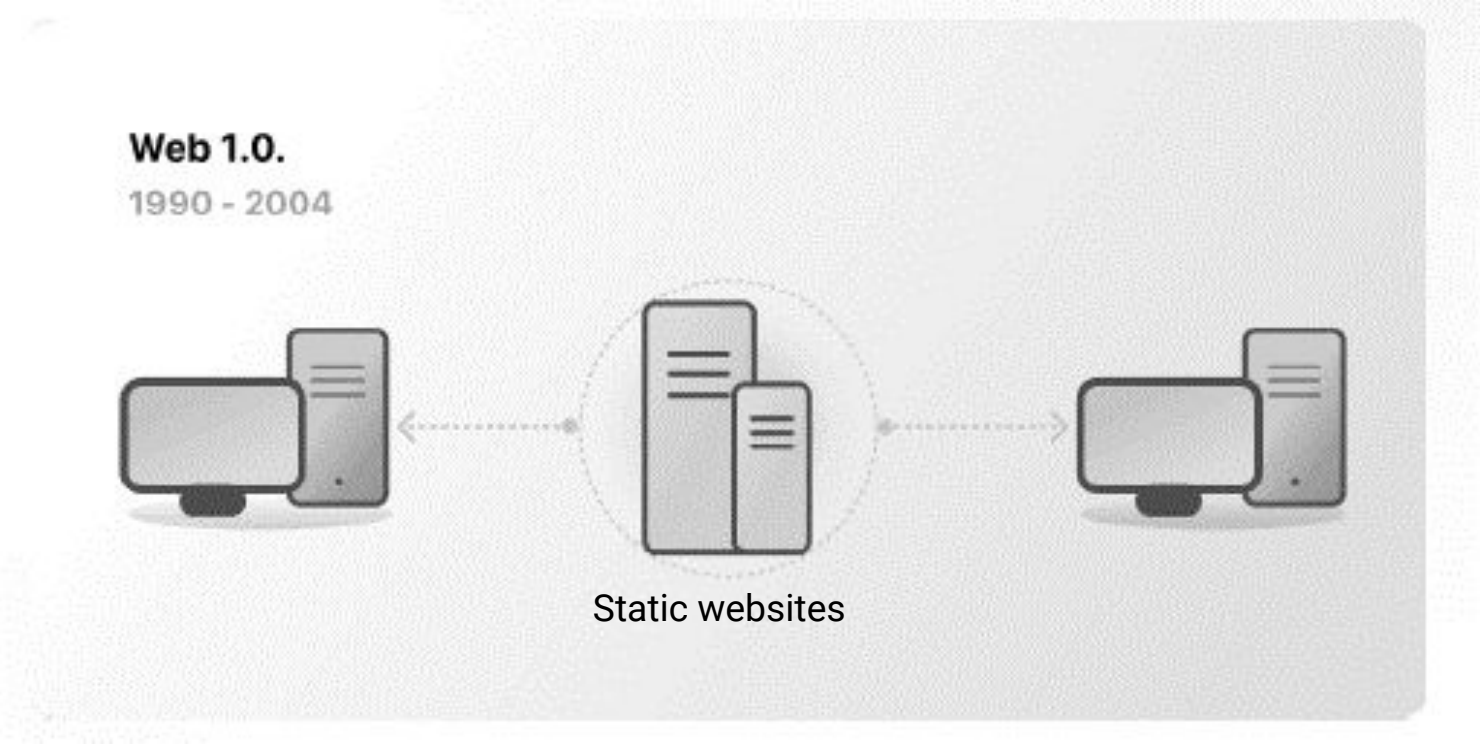
"Blockchain is a technological tour de force."
- Bill Gates

"The biggest opportunity set we can think of over the next decade."
- Bob Grifeld, CEO NASDAQ

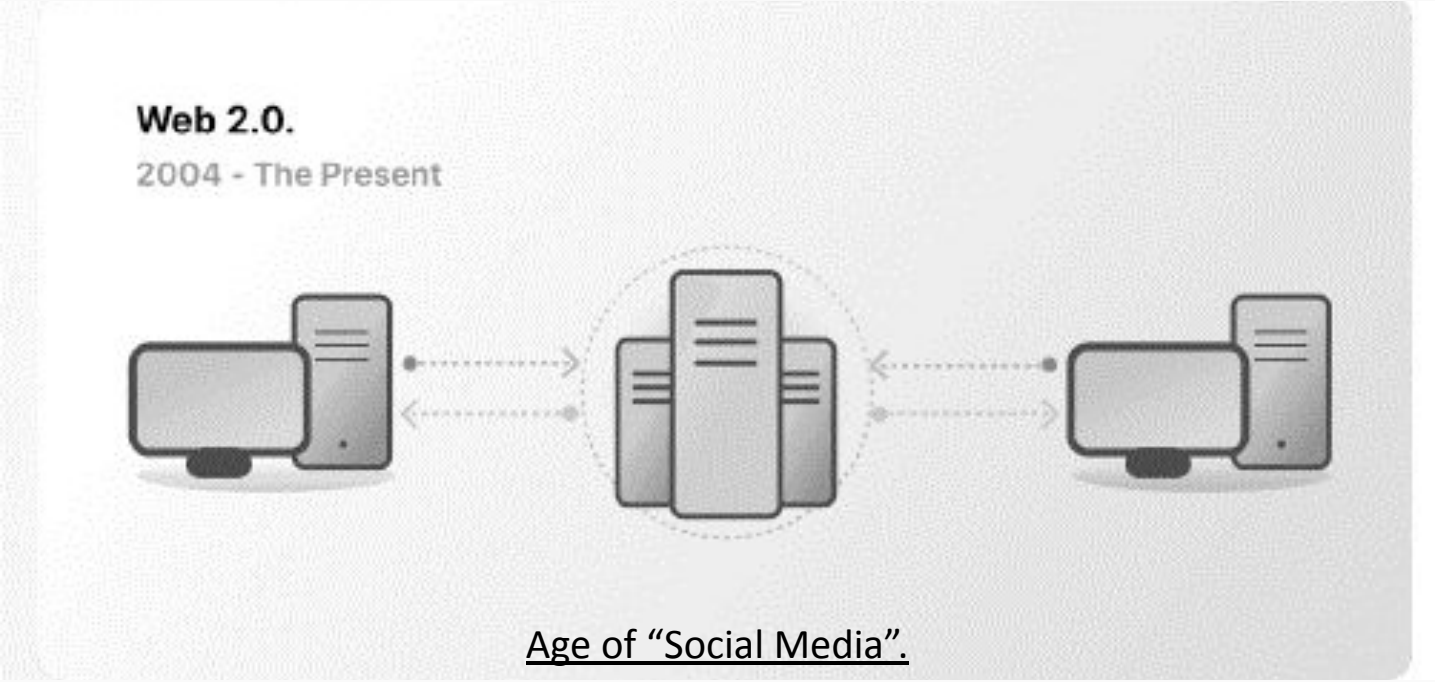
Is Blockchain Technology

The New Internet?

Web 1.0 : Read-Only (1990-2004)



Web 2.0: Read-Write (2004-Now)



Age of “Social Media”.

YouTube, Wikipedia, Flickr & Facebook gave voice to the voiceless and a means for like-minded communities to thrive

Information is the New Oil

Data is the most valuable commodity in the world.

As large companies realized the value of personal information they stockpiling the data in centralized server and start selling browser habits, searches and shopping information to advertisers.

All power is accumulating with big companies.
(Facebook, Google, Apple & Amazon)

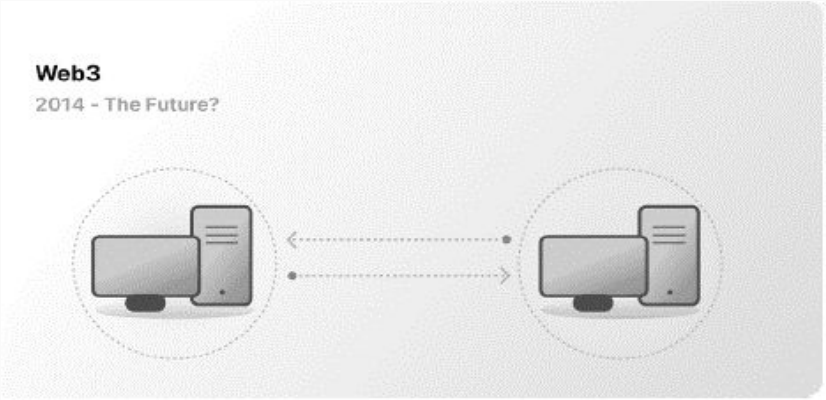
Problems in Centralization

- The privacy/data of Internet user is at the stake.
- Your data might be or may be sold…
- Single point of failure – Hacking

Web 3.0 is a new paradign!

Most of the internet (and the data on it) that people know and use today relies on trusting a handful of private companies to act in the public's best interests. They own the internet.

Web 3.0 uses **blockchains**, **cryptocurrencies**, and **NFTs** to give power back to the users in the form of ownership! It allows users to own the internet and their own data.

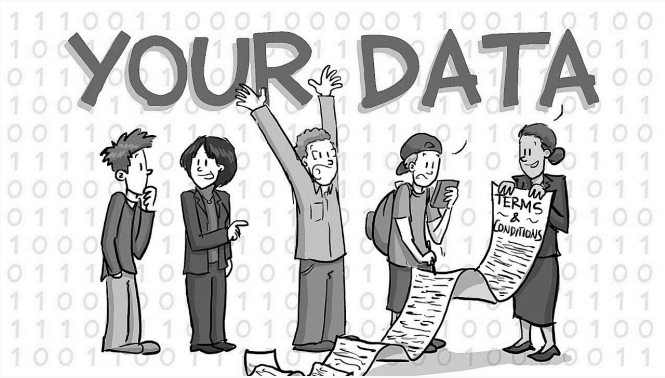


Web 3.0: Read – Write - Own

Rather than concentrating the power (and data) in the hands of huge corporates/behemoths with the questionable motives, it would be returned the rightful owners.

Decentralization was the idea, blockchain was the means

Blockchain can make you, owner of your data



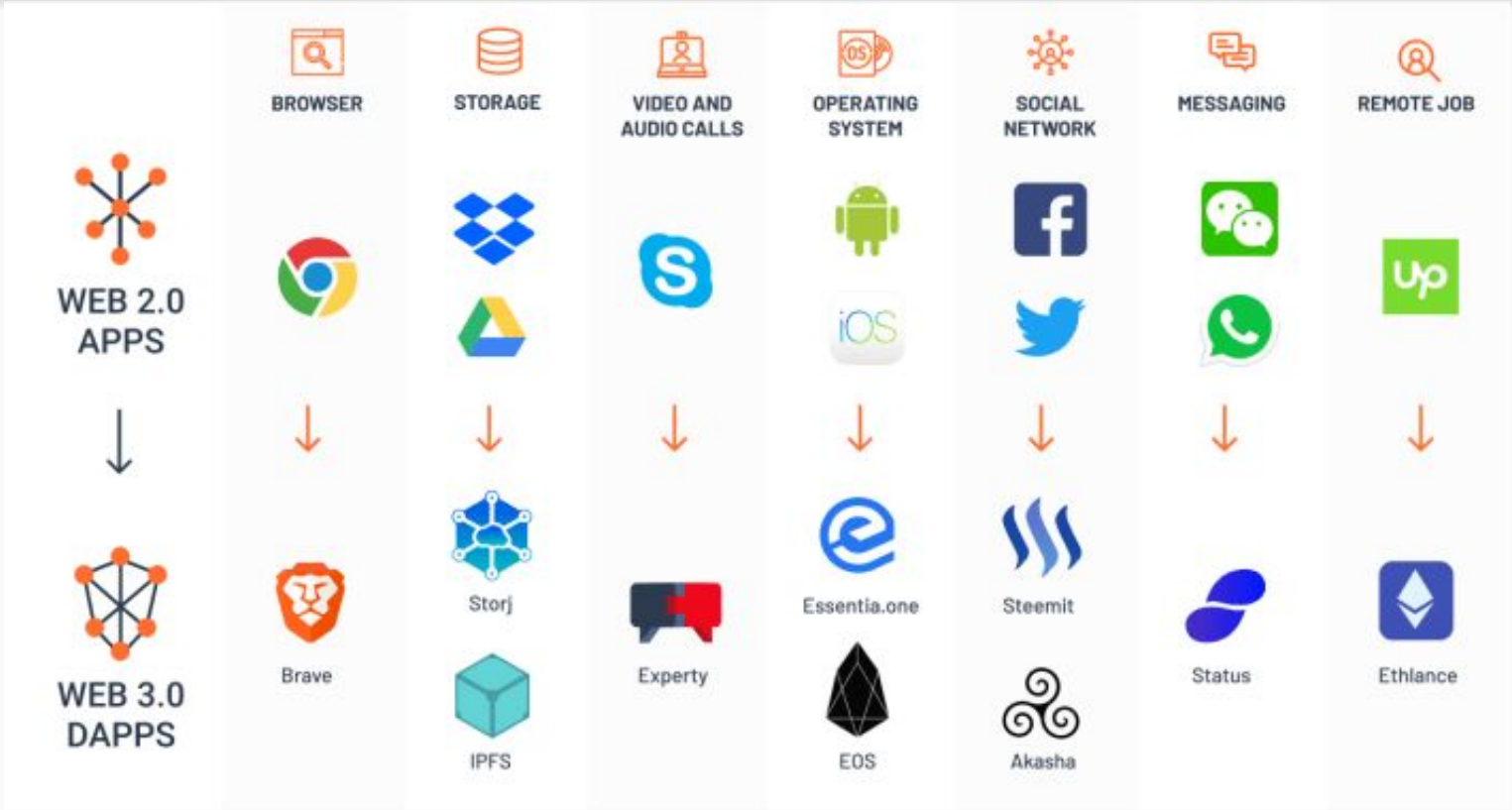
Web 3.0: Read – Write - Own

Web 3.0 is permissionless: everyone has equal access to participate in Web3, and no one gets excluded.

Web 3.0 has native payments: it uses cryptocurrency for spending and sending money online instead of relying on the outdated infrastructure of banks and payment processors.

Web 3.0 is trustless: it operates using incentives and economic mechanisms instead of relying on trusted third parties.

Web 3.0 Dapps


BROWSER
STORAGE
VIDEO AND AUDIO CALLS
OPERATING SYSTEM
SOCIAL NETWORK
MESSAGING
REMOTE JOB
WEB 2.0 APPS
WEB 3.0 DAPPS
Brave
Storj
IPFS
Experty
Essentia.one
EOS
Steemit
Akasha
Status
Ethlance

Industries than can be Disrupted by the Blockchain

- Banking & Payments
- Supply chain Management
- IOT
- Insurance
- Private Transport and Ride Sharing
- Online Data Storage
- Charity
- Governance
- Healthcare
- Online Music
- Retail
- Crowdfunding
- ID Management
- Voting

Distributed Network

- Distributed networks a higher level of security,
- To carry out malicious attacks would have to attack a large number of nodes at the same time.
- As the information is distributed among the nodes of the network: if some illegitimate change is made, the rest of the nodes will be able to detect it and will not validate this information.
- Consensus between nodes protects the network from deliberate attacks or accidental changes of information.

It all started with Idea: A Digital Currency

- David Chaum first proposed the concept of E-Cash in 1982
- David Chaum then founded a company called Digi Cash
- It uses cryptography security and anonymity
- Idea has some problem as with traditional currency,
  it requires central clearing house or single point of trust
- DigiCash declared bankruptcy in 1998
- Many other tried faced the same fate

https://www.vice.com/en/article/j5nzx4/what-was-the-first-blockchain

Story of Digital Cash

- DigiCash (David Chaum) – 1989
- Mondex (National Westminster Bank) - 1993
- CyberCash (Lynch, Melton, Crocker & Wilson) – 1994
- E-gold (Gold & Silver Reserve) – 1996
- **Hashcash (Adam Back) – 1997 (Hashing and Cryptography usage)**
- **Bit Gold (Nick Szabo) – 1998 (Smart Contract)**
- **B-Money (Wei Dai) - 1998** (Earlier version of Bitcoin)
- Lucre (Ben Laurie) – 1999

Why is there so much hype around blockchain technology?

There have been many unsuccessful attempts to create digital money in past

The prevailing issue is trust. If someone creates a new currency called X dollar, how can we trust that they won't give themselves million X dollars, or steal your X dollars for themselves?

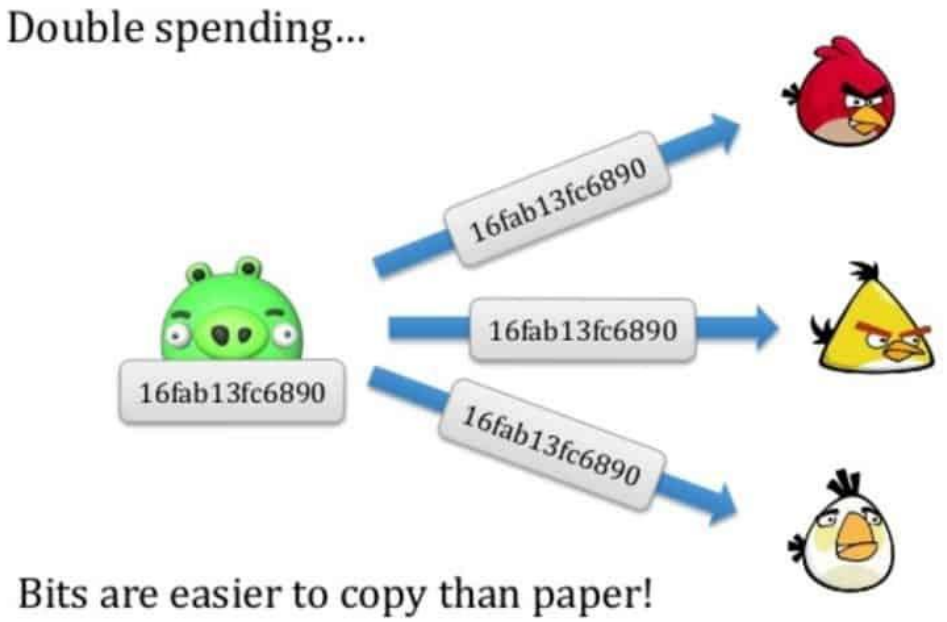
Bitcoin was designed to solve this problem using specific type of database called blockchain.

Most normal databases, i.e. SQL database, have someone in charge who can change the entries (e.g. giving themselves a million X dollars).

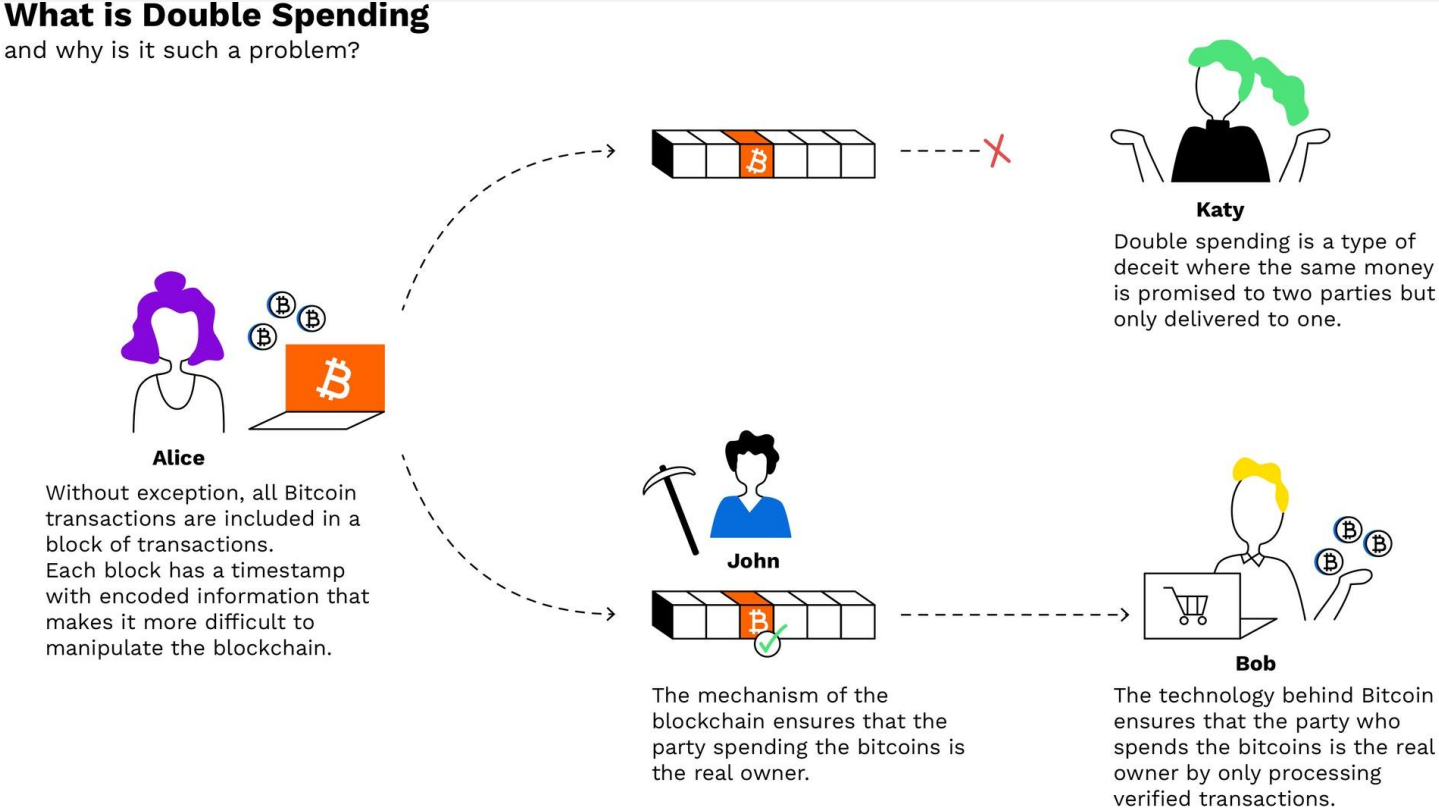
Blockchain is different because nobody is in charge; it's run by the people who use it.
What's more, bitcoins can't be faked, hacked or double spent – so people that own this money can trust that it has some value.

Double Spend Problem

The Double Spend Problem describes **the difficulty of ensuring digital money is not easily duplicated**.



Double Spend Problem

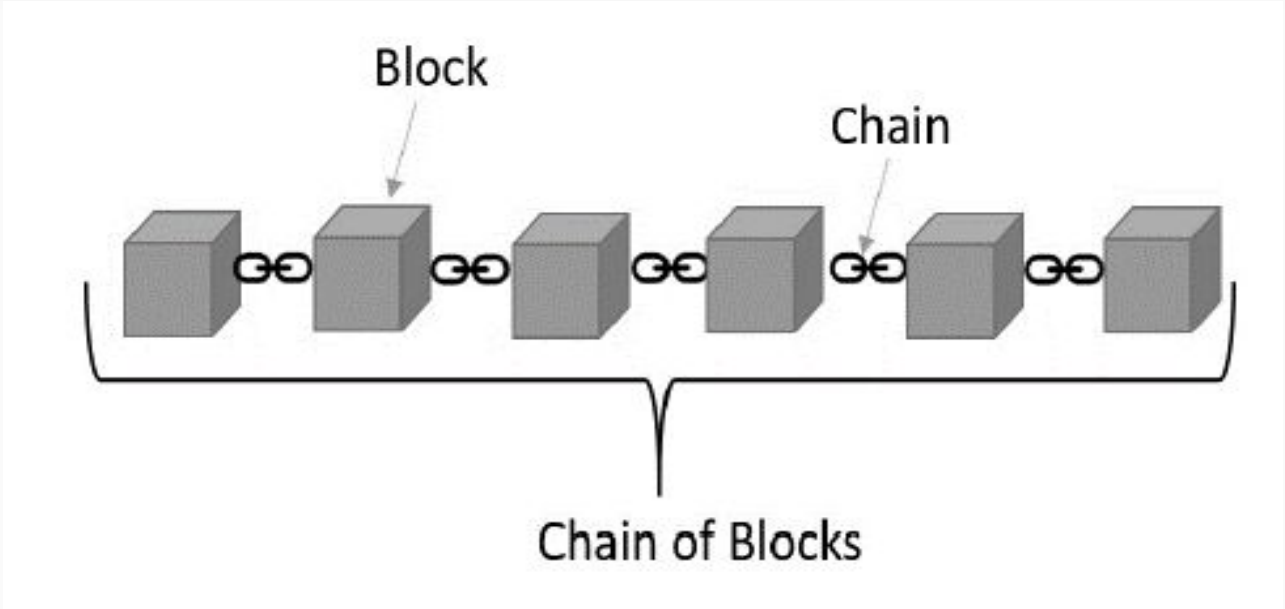


What is Blockchain

"The blockchain is an incorruptible digital ledger of economic transactions that can be programmed to record not just financial transactions but virtually every thing of value."

- Don & Alex Tapscott, authors Blockchain Revolution (2016)

What is a blockchain?



Blockchain is a time-stamped series of immutable record of data is managed by cluster of computers not owned by any single entity. Each of these blocks of data are secured and bound to each other using cryptographic principles

What is a blockchain?

- Stable, Robust & Durable,
- No single point of failure.
- No single node/server is the data owner
- Everyone participates as a stakeholder and get financial rewards.
- No one can change / modify past transactions
- Highly trustworthy, transparent, and incorruptible.

Ethereum also allows extending its functionality with the help of smart contracts.

What is Blockchain – Detail

Blockchain is a system of recording information in a way that makes it difficult or impossible to change, hack, or cheat the system.

A blockchain is essentially a distributed ledger of transactions that is duplicated and distributed across the entire network of computer systems on the blockchain.

Each block in the chain contains a number of transactions, and every time a new transaction occurs on the blockchain, a record of that transaction is added to every participant's ledger.

The decentralised database managed by multiple participants is known as Distributed Ledger Technology (DLT).

What is Blockchain – Detail Cont...

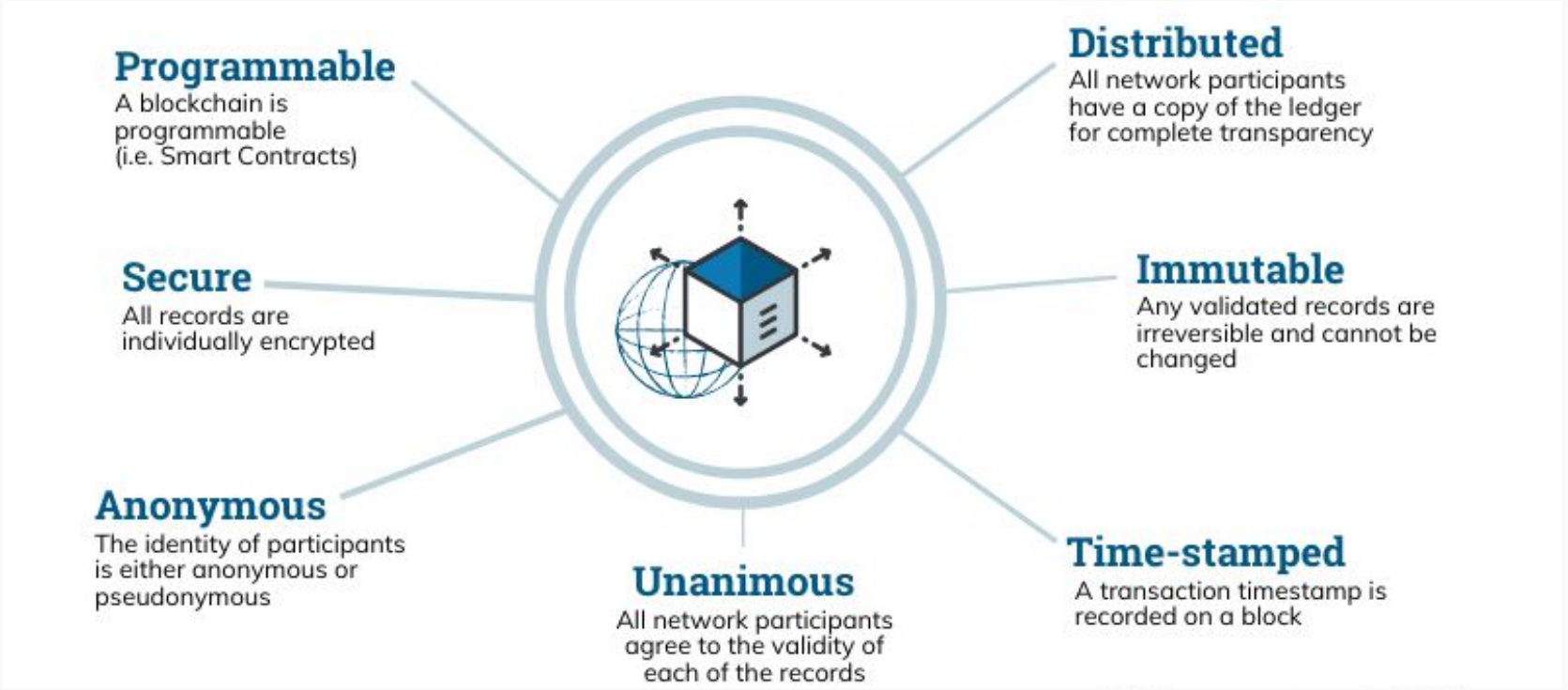
Blockchain is a type of DLT in which transactions are recorded with an immutable cryptographic signature called a hash.

This means if one block in one chain was changed, it would be immediately apparent it had been tampered with.

If hackers wanted to corrupt a blockchain system, they would have to change every block in the chain, across all of the distributed versions of the chain.

Blockchains such as Bitcoin and Ethereum are constantly and continually growing as blocks are being added to the chain, which significantly adds to the security of the ledger

Distributed Ledger Technology



Bitcoin

- In 2008 a white paper was published “Bitcoin: A Peer-to-Peer Electronic Cash System.” by Satoshi Nakamoto
- In 2009 first ever block of bitcoin, known as Genesis Block was mined
- Bitcoin uses:
  - Secure digital signatures
  - Not requiring the use of a third party
  - Proof of work
  - Hashing the transactions together to form a chain
- Satoshi Nakamoto is unknown person or group of people, wrote the Bitcoin paper
- Satoshi Disappears in December 2010

Bitcoin Properties

- Decentralized – Peer to peer ledger of balances
- Immutable – can never be changed, transactions are permanent
- Fungible - each BTC is equal, maintain it value
- Permission less and without borders – anyone can participate by downloading software
- Divisible – down to 8 decimal places
- Scarcity – 21 million coins ever
- Transferrable – can send any amount in seconds

What is a Bitcoin?

- A collection of concepts and technologies.
- It behaves like conventional currencies
- Can be purchased, sold & exchanged for other currencies at specialized exchanges.
- They are completely virtual with no physical existence.
- Fast, Secure & Borderless

Pizza for Bitcoin

May 22, 2010, 07:17:26 PM

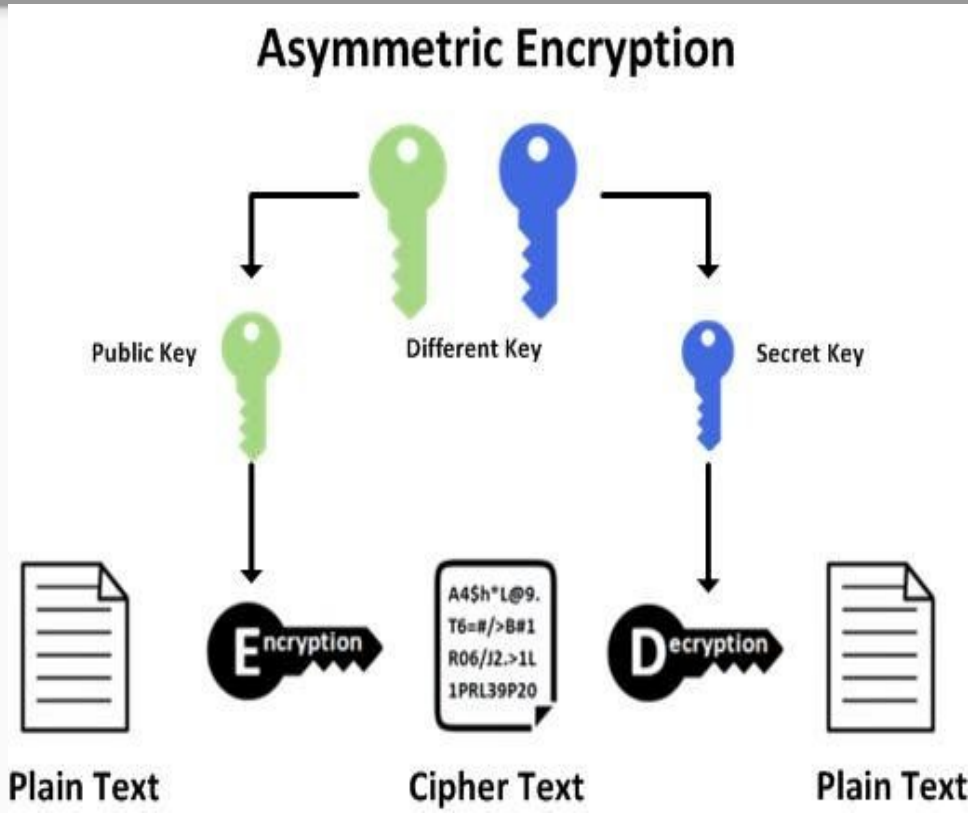
“I just want to report that I successfully traded **10,000** bitcoins for pizza”

Value:

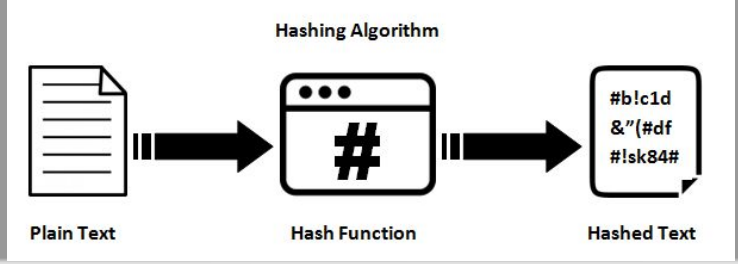
- May 22, 2010 - $41
- $20.50 per pizza

Asymmetric Cryptography

- Using two keys for encryption and decryption.
- Any key can be used for encryption and decryption.
- Message encryption with a public key can be decrypted using a private key and messages encrypted by a private key can be decrypted using a public key.



Hashing



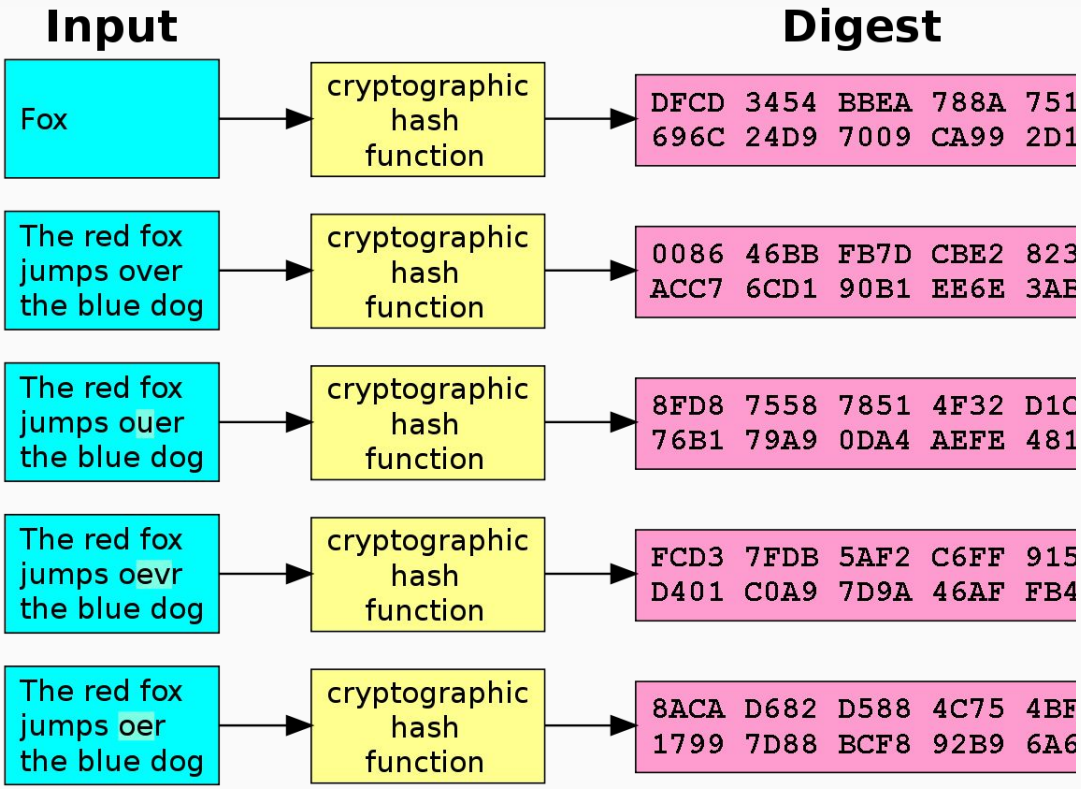
Is the process of transforming any input data into fixed length random character data, and it is not possible to regenerate or identify the original data from the resultant string data.

Hashes are also known as fingerprint of input data. It is next to impossible to derive input data based on its hash value.

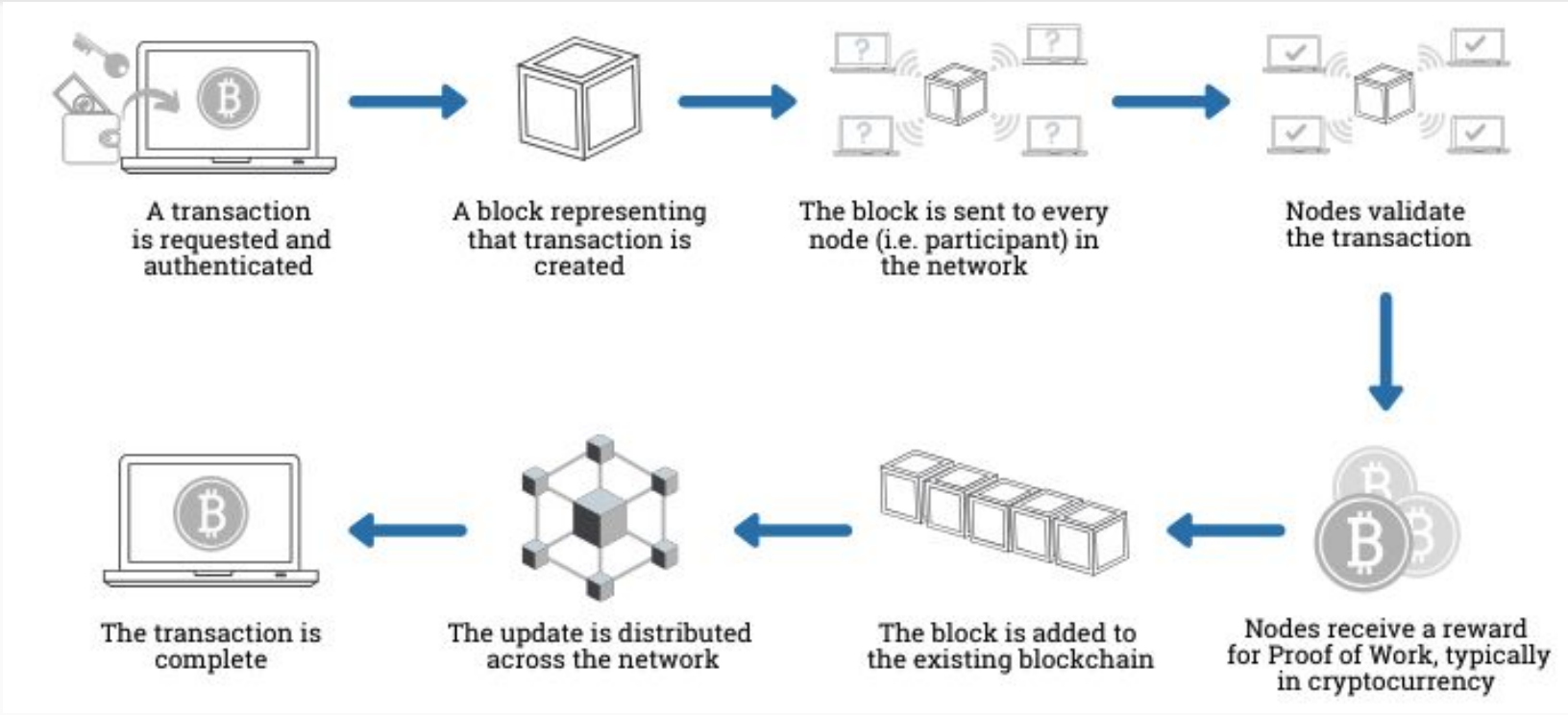
Hashing ensures that even a slight change in input data will completely change the output data, and no one can ascertain the change in the original data.

Another important property of hashing is that no matter the size of input string data, the length of its output is always fixed.This can especially become useful when large amounts of data can be stored as 256 bit output data

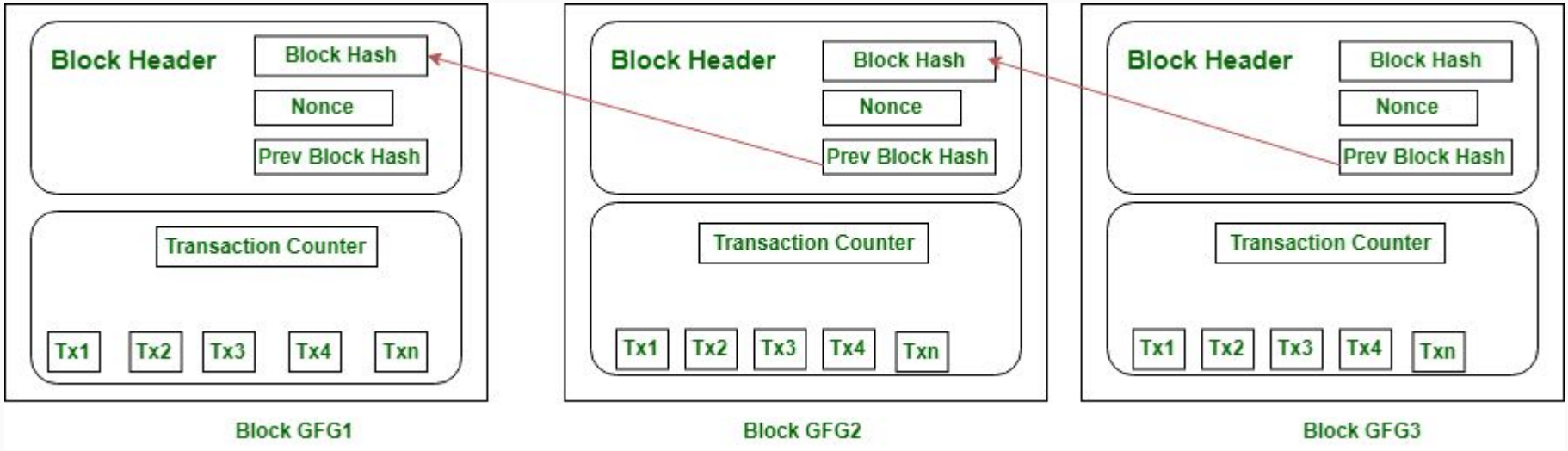
Hashing – Example



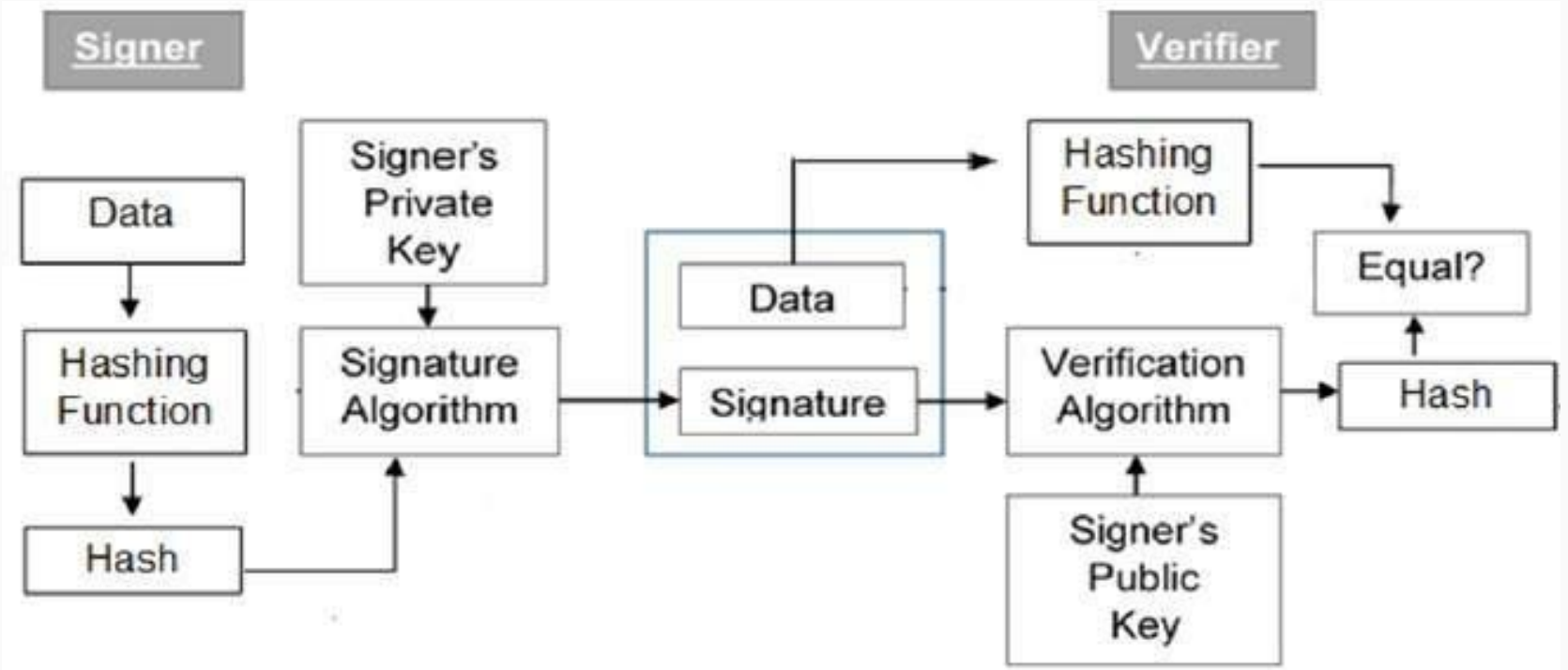
How does a transaction get into blockchain



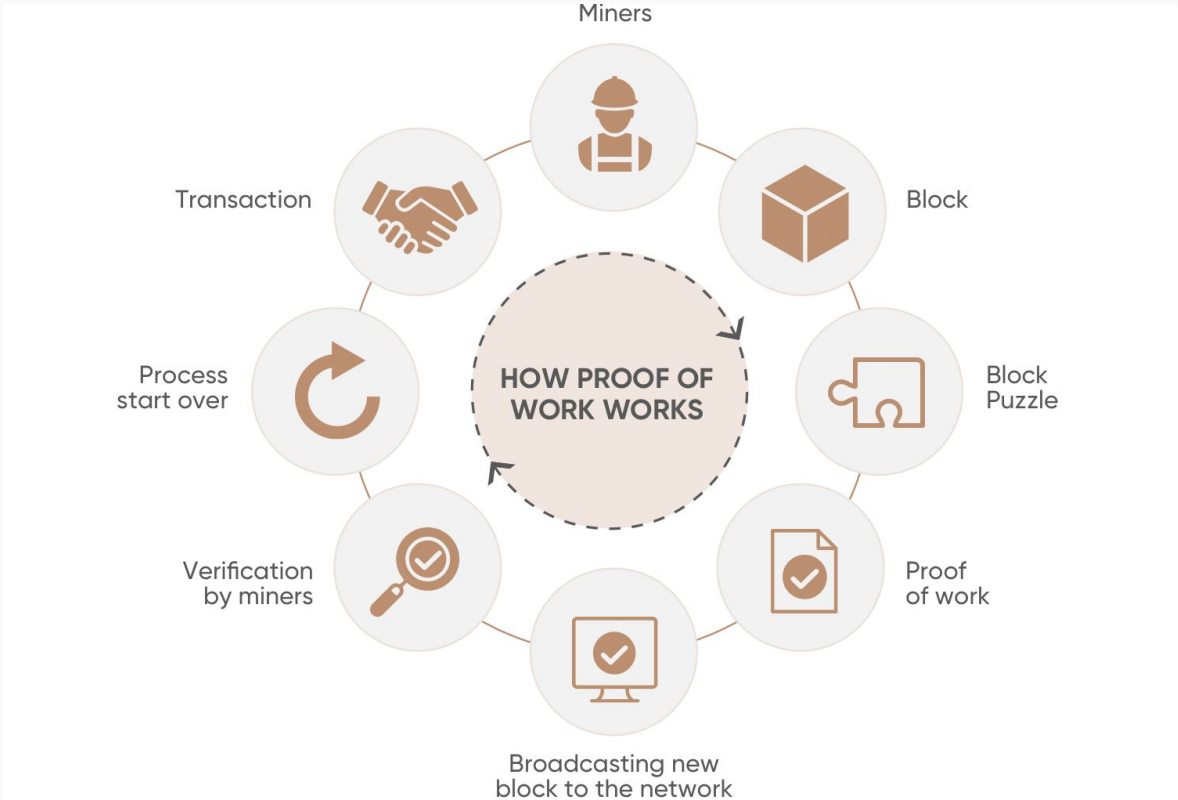
Block in Blockchain



Public Vs Private Key, Hash, Signer & Verifier



Proof of Work



How proof of work enables trustless consensus

- The main innovation that Satoshi Nakamoto introduced is using so-called *proof of work (POW)* to create distributed trustless consensus and solve the double-spend problem.
- POW is not a new idea, but the way Satoshi combined this and other existing concepts — cryptographic signatures, merkle chains, and P2P networks — into a viable distributed consensus system.
- POW is a requirement that expensive computations, also called *mining*, *to* be performed in order to facilitate transactions on the blockchain.
- To understand the link between computational difficulty and trustless consensus within a network implementing a distributed cryptocurrency system is a serious mental feat. With this writing I hope to help those who are attempting it.

Mining Nodes

A miner is responsible for writing transactions to the chain.

A miner's job is very similar to that of an accountant.

As an accountant is responsible for writing and maintaining the ledger;

Solely responsible for writing a transaction to an Ethereum ledger.

A miner is interested in writing transactions to a ledger because of the reward associated with it.

Miners get two types of reward—a reward for writing a block to the chain and cumulative gas fees from all transactions in the block

How Mining Works

Miner constructs a new block & adds all transactions to it.

Before adding these transactions, it will check if any of the transactions are not already written in a block that it might receive from other miners.

If so, it will discard those transactions

How Block Header is created

1. The miner takes hashes of two transactions at a time to generate a new hash till he gets a single hash from all transactions. The hash is referred to as a Merkle root transaction hash. This hash is added to the block header.

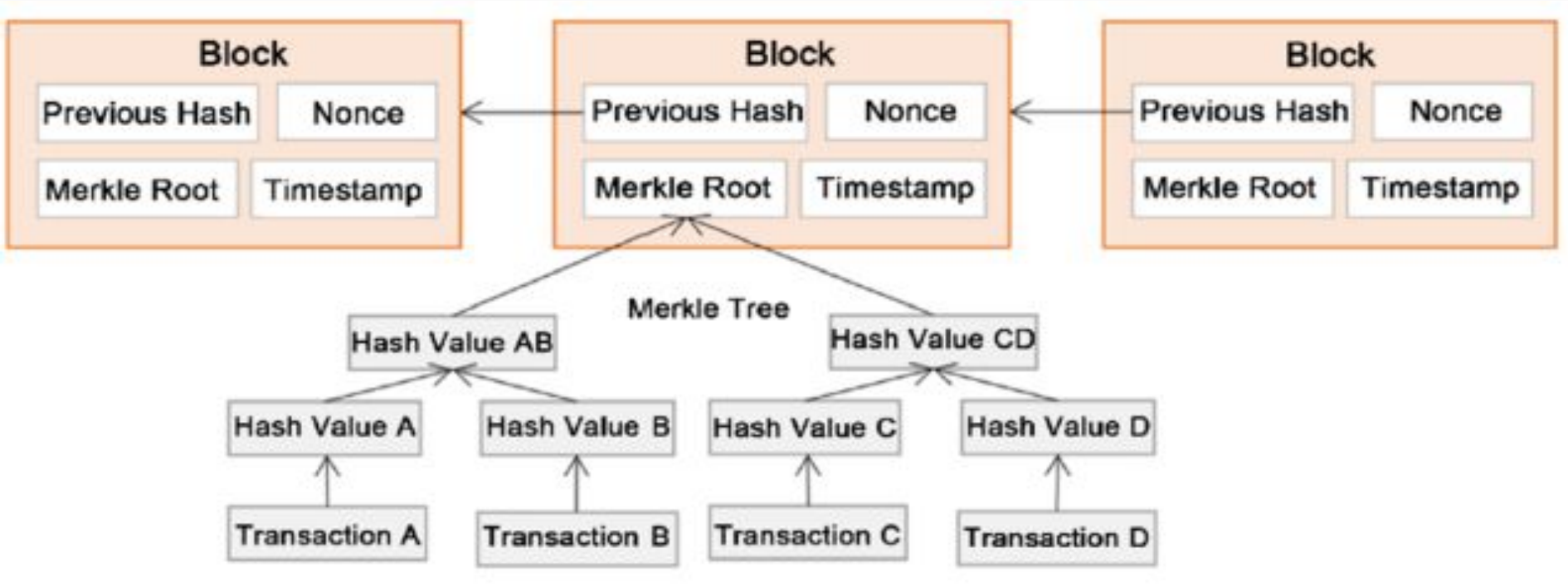
2. Miner also identifies the hash of the previous block. Previous block will become parent to the current block and its hash will also be added to the block header.

3. Miner calculates the state and receipts of transaction root hashes & adds them to block header

4. A nonce and timestamp is also added to the block header.

5. Eventually, one of the miners will be able to solve the puzzle and advertise the same to other miners in the network. The other miners will verify the answer and, if found correct, will further verify every transaction, accept the block, and append the same to their ledger instance.

Block Structure



Ether

Ether is the currency of Ethereum.

Every activity on Ethereum that modifies its state costs Ether as a fee

Successful miners are also rewarded Ether.

Converted to dollars or other traditional currencies through cryptoexchanges.

Ethereum has a metric system of denominations used as units of Ether. The smallest denomination of Ether is called wei

Ethereum Accounts

Are the main building blocks for the Ethereum ecosystem.

It is an interaction between accounts that Ethereum wants to store as transactions in its ledger.

There are two types of accounts available in Ethereum

- —**Externally owned accounts** and
- —**Contract accounts.**

Each account, by default, has a property named balance that helps in querying the current balance of Ether.

Transaction

Is an agreement between a buyer and a seller that there will be an exchange of assets, products, or services for currency, cryptocurrency, or some other asset

Following are the three types of transactions that can be executed in Ethereum:

1. **Transfer of Ether from one account to another:**

   Following are the possible cases:

- An EOA sending Ether to another EOA in a transaction
- An EOA sending Ether to a contract account in a transaction
- A contract account sending Ether to another contract account in a transaction
- A contract account sending Ether to an EOA in a transaction

Transaction

2. Deployment of a smart contract:

- An EOA can deploy a contract using a transaction in EVM.

3. Using or invoking a function within a contract:

- Executing a function in a contract that changes state is considered a transaction in Ethereum.
- If executing a function does not change a state, it does not require a transaction.

Transaction - Account Properties

“from” denotes the account that is originating the transaction and represents an account that is ready to send some gas or Ether. The from account can be externally owned or a contract account.

“to” refers to an account that is receiving Ether or benefits in lieu of an exchange. For transactions related to deployment of contract, the to field is empty. It can be externally owned or a contract account.

“value”refers to the amount of Ether that is transferred from one account to another.

“input” refers to the compiled contract bytecode and is used during contract deployment in EVM. It is also used for storing data related to smart contract function calls along with its parameters.

Transaction – Account Properties

“**blockHash**” - hash of block to which this transaction belongs.

“**blockNumber**” - is the block in which this transaction belongs.

hash“**gas**” - amount of gas supplied by the sender who is executing this transaction.

“**gasPrice**” - price per gas the sender was willing to pay in wei. (Total gas is computed at gas units * gas price).

“**hash**” - hash of the transaction.

Transaction – Account Properties

“**nonce**” - number of transactions made by the sender prior to the current transaction.

“**transactionIndex**” - serial number of current transactions in the block.

“**value**” - amount of Ether transferred in wei.

“**v, r, and s**” - relate to digital signatures and the signing of the transaction.

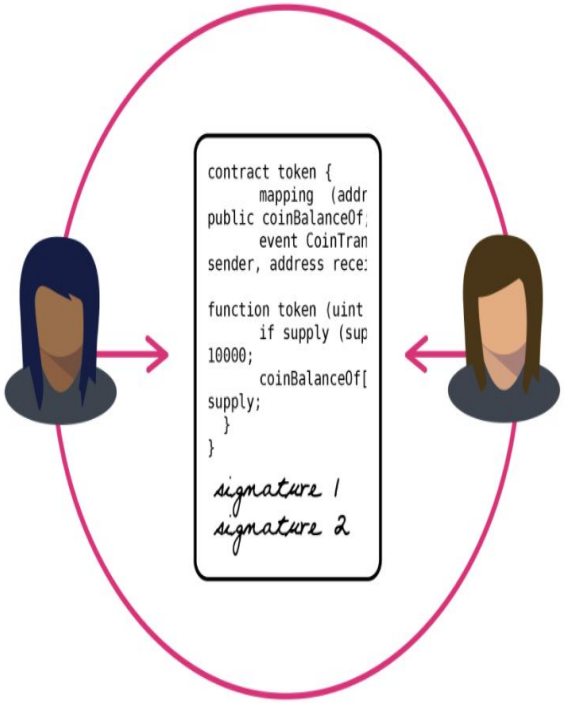
What is Contract

- A contract is a legal document that binds two or more parties who agree to execute a transaction immediately or in the future

- Since contracts are legal documents, they are enforced and implemented by law.

What is Smart Contract

- Custom logic and code deployed and executed within an Ethereum virtual environment.
- Digitized and codified rules of transaction between accounts.
- Help in transferring digital assets between accounts as an atomic transaction.
- Smart contracts can store data.



How to write smart contracts?

- Easiest and fastest way to develop smart contracts is to use a browser-based tool known as **Remix**. http://remix.ethereum.org.

- Remix provides a rich integrated development environment in a browser for authoring, developing, deploying, and troubleshooting contracts written using the Solidity language.

- All contract management related activities such as authoring, deploying, and troubleshooting can be performed from the same environment without moving to other windows or tabs.

How to write smart contracts?

- Easiest and fastest way to develop smart contracts is to use a browser-based tool known as **Remix**. http://remix.ethereum.org.

- Remix provides a rich integrated development environment in a browser for authoring, developing, deploying, and troubleshooting contracts written using the Solidity language.

- All contract management related activities such as authoring, deploying, and troubleshooting can be performed from the same environment without moving to other windows or tabs.

Remix - IDE

Compile tab compiles the contract into bytecode

Run tab is the place where you will spend the most time, apart from writing the contract. The Run tab allows you to deploy the contract to this runtime using the JavaScript VM environment in the Environment option

Injected Web3 environment is used along with tools such as Mist and MetaMask, which will be covered in the next chapter, and Web3 Provider can be used when using Remix in a local environment connecting to private network. In our case for this chapter, the default, JavaScript VM is sufficient.

Chapter # 2

Installing Ethereum and Solidity

Ethereum Networks

An **Open Source Platform** for **Creating and Deploying Distributed Applications**.

Backed up by a **large number of computers** (Nodes)

All Nodes are **interconnected** and **storing data** in a distributed ledger.(Each Node)

Choose an appropriate network based on their **requirements** and **use cases**.

Different networks help in deploying solutions and smart contracts on networks that do **not actually cost any Ether** or money.

Ethereum Networks

Main Ethereum Network is a global public network that anybody can use.

- Main network incurs costs in terms of Gas
- Main network is known as Homestead
- Anybody can connect to it and access both Data and Transactions stored in it

Test Network help to facilitate and increase adoption of the Ethereum blockchain.

- Exact Replica of the main network
- Does not cost anything for deployment and usage of contracts
- Test Ethers can be generated using faucets and used on these networks
- Multiple test networks available at the time of writing, such as Ropsten, Kovan, and Rinkeby

Ethereum Networks

Private Network is created and hosted on a private infrastructure.

- Are controlled by a single organization and they have full control over it.
- There are solutions, contracts, and use cases that an organization might not want to put on a public network even for test purposes.
- They want to use private chains for development, testing, and production environments.
- Organizations should create and host a private network and they will have full control over it

Ethereum Networks

Consortium network is also a private network, however, with a difference.

Consortium network comprises nodes, each managed by a different organization.

In effect, no organization has a control over the data and chain.

However, it is shared within the organization and everyone from these organizations can view and modify the current state.

These might be accessible through the internet or completely private networks using VPN

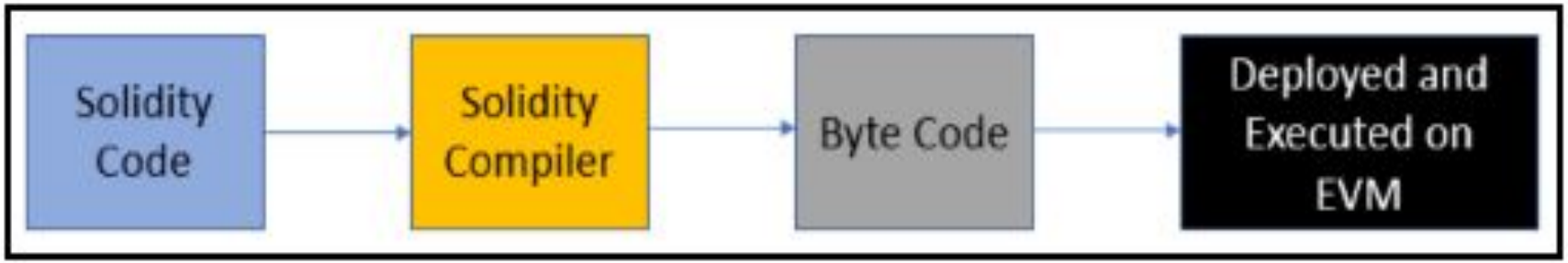
Chapter # 3

Introducing Solidity

Solidity Compiler & EVM

- **EVM** executes code that is part of **smart contracts**.
- **Smart contracts** are written in **Solidity**;
- However, **EVM does not understand Solidity.**

- **EVM** understands **bytecode**.
- Solidity comes with a **SOLC** which translate **Solidity into Bytecode**



Solidity & Solidity files

Solidity is a programming language that is very close to JavaScript

Solidity is a **statically-typed**, **case-sensitive**, and **object-oriented**

programming (OOP) language.

A Solidity file is composed of the following four high-level

- **constructs:**
- **Pragma**
- **Comments**
- **Import**
- **Contracts/library/interface**

Pragma and Compiler version

- Pragma is generally the first line of code within any Solidity file.
- ***pragma*** is a directive that specifies the compiler version to be used for
- current Solidity file.
- Although it is **not mandatory**, it is a good practice to declare the **pragma directive as the first statement in a Solidity file**.
- The syntax for the pragma directive is as follows:

```
pragma Solidity <<version number>> ;
```

- The version number comprises of two numbers
    - a **major build** and a **minor build** number.

0.5.0

Pragma and Compiler version

The ^ character, also known as caret, is optional in version numbers but plays a **significant role in deciding the version number based** on the following rules:

1. The ^ character refers to the latest version within a major version. So, ^0.5.0 refers to the latest version within build number 4, which currently would be 0.5.11.
2. The ^ character will not target any other major build apart from the one that is provided.
3. The Solidity file will compile only with a compiler with 4 as the major build. It will not compile with any other major build.

As a good practice, it is better to compile Solidity code with an exact compiler version rather than using ^

Comments

- Single-line comments

  // This is a single-line comment in Solidity

- Multiline comments

  /* This is a multiline comment In Solidity. Use this when multiple consecutive lines Should be commented as a whole */

- **Ethereum Natural Specification (Natspec)**

  Natspec is used for documentation purposes and it has its own specification.

Comments - NetSpecs

For more information
https://solidity.readthedocs.io/en/v0.5.10/natspec-format.html

```
pragma solidity ^0.5.6;

/// @title A simulator for trees
/// @author Larry A. Gardner
/// @notice You can use this contract for only the most basic simulation
/// @dev All function calls are currently implemented without side effects
contract Tree {
    /// @author Mary A. Botanist
    /// @notice Calculate tree age in years, rounded up, for live trees
    /// @dev The Alexandr N. Tetearing algorithm could increase precision
    /// @param rings The number of rings from dendrochronological sample
    /// @return age in years, rounded up for partial years
    function age(uint256 rings) external pure returns (uint256) {
        return rings + 1;
    }
}
```

The import statement

The **import** keyword helps **import other Solidity files** and we can **access its code within the current Solidity file** and code.

This helps us write modular Solidity code.

```
import <<filename>> ;
```

File names can be fully **explicit or implicit paths**

Contracts

Apart from pragma, import, and comments, we can define **contracts, libraries, and interfaces** at the global or top level.

The **multiple contracts, libraries, and interfaces** can be declared within the **same Solidity file**.

```
//contracts.sol

pragma solidity 0.4.19;

// This is a single line comment in Solidity

/* This is a multi-line comment
    In solidity. Use this when multiple consecutive lines
Should be commented as a whole */

contract firstContract {

}

contract secondContract {

}

library stringLibrary {

}

library mathLibrary {

}

interface IBank{

}

interface IAccount {

}
```

Structure of a contract

A contract consists of the following multiple constructs:

1. **State variables**
2. **Structure definitions**
3. **Modifier definitions**
4. **Event declarations**
5. **Enumeration definitions**
6. **Function definitions**

1. State variables

- Variables in programming refer to storage location that can contain values.
- These values can be changed during runtime.
- The variable can be used at multiple places within code and they will all refer to the value stored within it.
- Solidity provides two types of variable
  — State variable and
  — Memory variables.
    - State variables are permanently stored in a blockchain ledger by miners.
    - Variables declared in a contract that are not within any function are called state variables.
    - State variables store the current values of the contract.
    - The allocated memory for a state variable is statically assigned and it cannot change during the lifetime of the contract.
    - Each state variable has a type that must be defined statically.

1. State variables

State variables also have additional qualifiers associated with them.

internal: By default, (if nothing is specified).
Can only be used within current contract functions and any contract that inherits from them. These variables cannot be accessed from outside for modification; however, they can be viewed.
int internal StateVariable ;

private: like internal with additional constraints.
Can only be used in contracts declaring them. They cannot be used even within derived contracts.
int private privateStateVariable ;

1. State variables

public: This qualifier makes state variables access directly.

The Solidity compiler generates a getter function for each public state variable. An example of a public state variable is as follows:

```
int public stateIntVariable ;
```

constant: This qualifier makes state variables immutable.

Value must be assigned at declaration time itself. In fact, the compiler will replace references of this variable everywhere in code with the assigned value.

```
bool constant hasIncome = true;
```

Data Types

Solidity provides the following multiple out-of-box data types:

- **bool**
- **uint/int**
- **bytes**
- **address**
- **mapping**
- **enum**
- **struct**
- **bytes/String**

Using enum and struct, it is possible to declare custom user-defined data types as well.

Literals

Solidity provides usage of **literal for assignments to variables**. **Literals do not have names; they are the values themselves.** Variables can change their values during a program execution, but a literal remains the same value throughout

Examples:

- **Integer literal** are 1, 10, 1,000, -1, and -100.
- **String literals** are "Ritesh" and 'Modi'.
- **Address literals** are 0xca35b7d915458ef540ade6068dfe2f44e8fa733c
- **Hexadecimal literals** is hex"1A2B3F".
- **Decimal literals** 4.5 and 0.2.

Integers

Integers help in storing numbers in contracts

Signed integers: Signed integers can hold both negative and positive values.

Unsigned integers: Unsigned integers can hold only positive values along with zero. They can also hold negative values apart from positive and zero values

There are **multiple flavors of integers** in Solidity. Depending on requirements, **an appropriately sized integer should be chosen**

Integers - Flavours

Solidity provides **uint8 type to represent 8 bit unsigned integer** and thereon in **multiples of 8 till it reaches 256.**

There could be **32 different declarations of uint with different multiples of of 8**, such as **uint8, uint16, unit24,** as far as **uint256 bit**.

While storing values **between 0 and 255 uint8 is appropriate**, and while storing values between **-128 to 127 int8 is more suitable.**

The **default value** for **both signed and unsigned integers is zero**

Integers are **value types**; however, when **used as an array they are** referred as **reference types.**

Integer

Mathematical operations such as addition, subtraction, multiplication, division, exponential, negation, post-increment, and pre-increment can be performed on integers.

```
pragma solidity 0.4.19;

contract AllAboutInts {


    uint  stateUInt = 20 ; //state variable
    uint  stateInt = 20 ; //state variable

    function getUInt(uint incomingValue)
    {
        uint memoryuint = 256 ;
        uint256 memoryuint256 = 256 ;
        uint8 memoryuint8 = 8 ; //can store value from 0 upto 255

        //addition of two uint8
        uint256 result = memoryuint8 + memoryuint ;

        // assignAfterIncrement = 9 and memoryuint8 = 9
       uint8 assignAfterIncrement = ++memoryuint8 ;

        // assignAfterIncrement = 9 and memoryuint8 = 10
      uint8 assignBeforeIncrement = memoryuint8++;


    }

    function getInt(int incomingValue)
    {
        int memoryInt = 256 ;
        int256 memoryInt256 = 256 ;
        int8 memoryInt8 = 8 ; //can store value from -128 to 127
    }

}
```

Boolean

```
bool isPaid = true;
```

The **bool** data type can be used to represent scenarios **that have binary results,** such as **true or false.**

It is to be noted that **bools** in Solidity **cannot be converted to integers, like javascript.**

It's a **value type** and any **assignment to other boolean variables creates a new copy.** The **default value** for bool in Solidity is **false.**

Bytes

Byte refers to 8 bit signed integers. Everything in memory is stored in bits consisting of binary values—0 and 1.

The byte data type also used to store information in **binary format**.

It provides data types in the range from **bytes1** to **bytes32**

These are called fixed sized byte arrays and are implemented as value types

bytes1 = 1 byte **bytes2 = 2 bytes.**

The default value for byte is 0x00

Bytes: Assignments

Hexadecimal: `bytes1 aa = 0x65;` 😊

Integer: `bytes1 aa = 8;` 😊

Negative Integer: `bytes1 aa = -8;` 😊

Character: `bytes1 aa = 'a';` 😊

We can also perform **bitwise operations** such as and, **or, xor, not, and left and right shift operations** on the byte data

Arrays

Arrays refer to groups of values of the same type.

Arrays help in storing these values together and ease the process of iterating, sorting, and searching for individuals or subsets of elements within this group

Fixed Arrays

Fixed arrays refer to arrays that have a **pre-determined size mentioned at declaration**.

```
int[5] age ; // array of int with 5 fixed storage space allocated
byte[4] flags ; // array of byte with 4 fixed storage space allocated
```

```
int[5] age ;
age = [int(10),2,3,4,5];
```

Dynamic Array

Arrays that **do not have a pre-determined size at the time of declaration;** however, **their size is determined at runtime.**

```
int[] age ; // array of int with no fixed storage space allocated. Storage is
allocated during assignment
byte[] flags ; // array of byte with no fixed storage space allocated
```

Dynamic Array

Dynamic arrays can be initialized inline or using the **new** operator.

```
int[] age = [int(10), 20,30,40,50] ;
int[] age = new int[](5) ;
```

Array properties

Index:

This property used for **reading individual array elements is supported by all types of arrays, except for the string type.**

Writing individual element is supported for **dynamic arrays, fixed arrays, and the bytes type only.**

Writing is not supported for string and fixed sized byte arrays.

Array properties

push:

This property is supported by dynamic arrays only.

length:

This property is supported by all arrays from read perspective, except for the **string** type.

Only **dynamic arrays** and **bytes** **support modifying the length property.**

The Bytes Array

The bytes array is a dynamic array that can hold any number of bytes.

It is not the same as **byte[]**. The byte[] array takes 32 bytes for each element whereas **bytes tightly holds all the bytes together.**

Bytes Array

Bytes can be declared

```
bytes localBytes = new bytes(0) ;
```

Bytes can be assigned

```
localBytes = "Ritesh Modi";
```

Values can be pushed

```
localBytes.push(byte(10));
```

Provides read/write length property

```
localBytes.length = 4;
```

```
return localBytes.length;
```

The String

Strings are dynamic data types that are based on bytes arrays. With some additional restrictions

- Strings cannot be indexed
- Cannot be Pushed or increased
- Do not have the length property

To perform any of these actions on string variables, they should first be **converted into bytes**

The String: Assignment

Strings can be declared and assigned values directly, as follows:

```
String name = 'Ritesh Modi" ;
```

They can be also converted to bytes, as follows:

```
Bytes byteName = bytes(name) ;
```

Struct / Structure

Structures or structs helps implement custom user-defined data types.

A structure is a composite data type, consisting of multiple variables of different data types.

They are very similar to contracts; however, they do not contain any code within them. They consist of only variables.

For Example. you want to store related data together. information about an employee: name, age, marriage status & bank account numbers.

To represent this, these individual variables related to single employee, a structure in Solidity can be declared using the **struct** keyword.

Struct / Structure

```
//structure definition
struct myStruct {
    string name; //variable fo type string
    uint myAge; // variable of unsigned integer type
    bool isMarried; // variable of boolean type
    uint[] bankAccountsNumbers; // variable - dynamic array of unsigned integer
}
```

To create an instance of a structure, the following syntax is used. There is no need to explicitly use the new keyword. The new keyword can only be used to create an instance of contracts or arrays

```
human = myStruct("Ritesh",10,true,new uint[](3)); //using struct myStruct
```

Multiple instance of struct can be created in functions. Structs can contain array and the mapping variables, while mappings and arrays can store values of type struct

Functions

- Functions are the heart of Ethereum and Solidity.
- Ethereum maintains the current state of state variables and executes transaction to change values in state variables.
- When a function in a contract is called or invoked, it results in the creation of a transaction.
- Functions are the mechanism to read and write values from/to state variables.
- Functions are a unit of code that can be executed on-demand by calling it.
- Functions can accept parameters, execute its logic, and optionally return values to the caller.
- Solidity provides named functions with the possibility of only one unnamed function in a contract called the **fallback function**.

Functions

- Functions are the heart of Ethereum and Solidity.
- Ethereum maintains the current state of state variables and executes transaction to change values in state variables.
- When a function in a contract is called or invoked, it results in the creation of a transaction.
- Functions are the mechanism to read and write values from/to state variables.
- Functions are a unit of code that can be executed on-demand by calling it.
- Functions can accept parameters, execute its logic, and optionally return values to the caller.
- Solidity provides named functions with the possibility of only one unnamed function in a contract called the **fallback function**.

Modifier

- Modifier is always associated with a function.
- It has the power to change the behavior of functions that it is associated with.
- It is like a function that will be executed before execution of the target function.

Example: you want to invoke the getAge function but, before executing it, you would like to execute another function that could check the current state of the contract, values in incoming parameters, the current value in state variables, and so on and accordingly decide whether the target function getAge should be executed.

- Helps in writing cleaner functions without cluttering them with validation and verification rules.
- Moreover, the modifier can be associated with multiple functions.
- This ensures cleaner, more readable, and more maintainable code.

Modifier

```
//modifier declaration
modifier onlyBy(){
    if (msg.sender == personIdentifier) {
        _;
    }
}
```

The modifier is associated with a `getAge` function as shown in the following screenshot:

```
//function definition
function getAge (address _personIdentifier) onlyBy() payable external returns (uint) {

   human =  myStruct("Ritesh",10,true,new uint[](3)); //using struct myStruct

   gender _gender = gender.male; //using enum
}
```

Events

- Events are fired from contracts such that anybody interested in them can trap/catch them and execute code in response.
- Used primarily for informing the calling application about the current state of the contract by means of the logging facility of EVM.
- Used to notify applications about changes in contracts and applications can use them to execute their dependent logic.
- Instead of applications they keep polling the contract for certain state changes; the contract can inform them by means of events.
- Declared within the contract at the global level and invoked within its functions.
- Declared using the event keyword, followed by an identifier and parameter list
- Parameters can be used to log information or execute conditional logic.
- Event information and its values are stored as part of transactions within blocks.

Events

There is no need to explicitly provide parameter variables—only data types are sufficient as shown in the following screenshot:

```
// event declaration
event ageRead(address, int );
```

An event can be invoked from any function by its name and by passing the required parameters, as shown in the following screenshot:

```
//function definition
function getAge (address _personIdentifier) onlyBy() payable external returns (uint) {

   human =  myStruct("Ritesh",10,true,new uint[](3)); //using struct myStruct

   gender _gender = gender.male; //using enum

   ageRead(personIdentifier, stateIntVariable);
}
```

Enumeration

- **enum** keyword is used to declare enumerations.
- Help in declaring a custom user-defined data type.
- enum consists of an enumeration list, a predetermined set of named constants.
- Constant values within an enum can be explicitly converted into integers.
- Each constant value gets an integer value, with the first one having a value of 0 and the value of each successive item is increased by 1.
- An enum declaration uses the **enum** keyword followed identifier and a list of enumeration values within the {} brackets.
- **Enum declaration** does not have a semicolon as its terminator and that there should be at least one member declared in the list.

Enumeration

- An example of enum is as follows:

  enum gender {male, female}

- A variable of enumeration can be declared and assigned a value as shown in the following code:

  gender _gender = gender.male ;

- It is not mandatory to define enum in a Solidity contract.
- **enum** should be defined if there is a constant list of items that do not change

Data Type

Solidity data types can broadly be classified in the following two types:

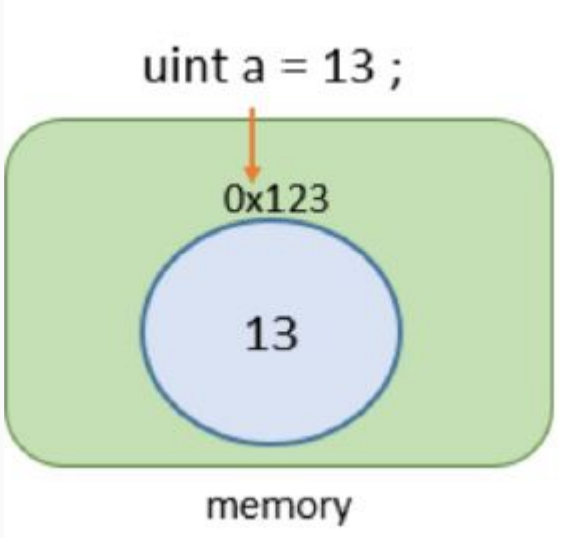
Value types

Reference types

Data Type - Value Type

A type is referred as **value type** if **it holds the data (value) directly within the memory owned by it.**

Value types are types that **do not take more than 32 bytes of memory in size.**



Data Type - Value Type

Solidity provides the following value types:

bool: The boolean value that can hold true or false as its value

uint: These are unsigned integers that can hold 0 and positive values only

int: These are signed integers that can hold both negative and positive values

address: This represents an address of an account on Ethereum environment

byte: This represents fixed sized byte array (byte1 to bytes32)

enum: Enumerations that can hold predefined constant values

Passing by Value

When a value type variable is **assigned to another variable** or when a value type variable is **sent as an argument to a function**, **EVM creates a new variable instance and copies the value of original value type into target variable.**

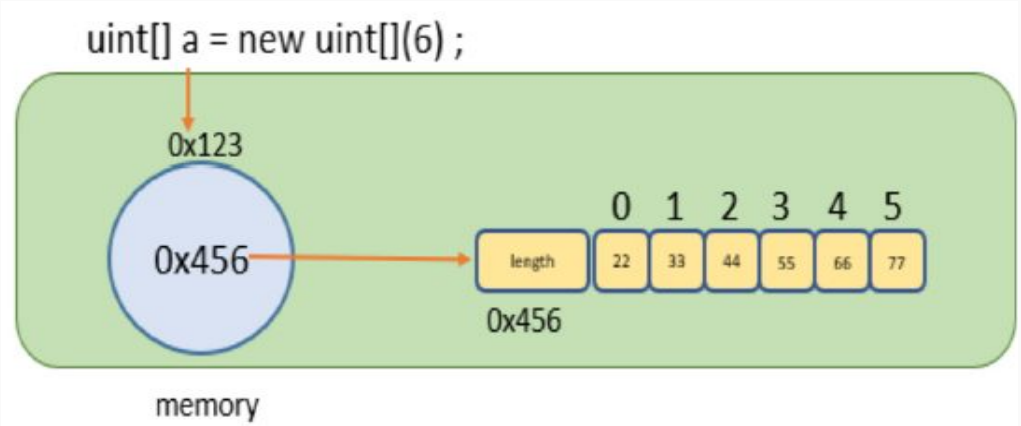
Changing values in original or target variables will not affect the value in another variable.

Both the variables will maintain their independent, isolated values and they can change without the other knowing about it.

Data Type - Reference types

Reference types, unlike value types, do not store their values directly within the variables themselves.

Instead of the value, they store the address of the location where the value is stored.



Data Type - Reference types

Solidity provides the following reference types:

Arrays: These are fixed as well as dynamic arrays.

Structs: These are custom, user-defined structures.

String: This is sequence of characters. In Solidity, strings are eventually stored as bytes.

Mappings: This is similar to a hash table or dictionary in other languages storing key-value pairs.

Passing by reference

When a **reference type variable is assigned to another variable** or when a reference type variable is **sent as an argument to a function**, **EVM creates a new variable instance and copies the pointer from the original variable** into the target variable

Both the variables are pointing to the same address location.

Both the variables will share the same values and change committed by one is reflected in the other variable.

Storage and memory data locations

The amount of gas you will use during a transaction depends on the data location you use in your smart contract. Hence, the best practice is to write an optimized code that uses a minimum amount of gas.

Storage

1. Global
2. Permanent
3. State Vars
4. Expensive

Memory

1. Within Function
2. Temporary
3. Function Parameter
4. Cheaper

Memory Location: Rules

1. The storage and memory keywords are used to reference data in storage and memory respectively.
2. Contract storage is pre-allocated during contract construction and cannot be created in function call. After all, it makes little sense to create new variable in storage in a function if it is to be persisted.
3. Memory cannot be allocated during contract construction but rather created in function execution. Contract **state variable** is always declared in storage. It makes little sense to have state variable that cannot persist.
4. When assigning a memory referenced data to a storage referenced variable, we are copying data from memory to storage. **No new storage is created.**
5. When assigning a storage reference data to a memory referenced variable, we are copying data from storage to memory. **New memory is allocated.**
6. When a storage variable is created in function locally by look up, it simply reference data already allocated on Storage. **No new storage is created.**

Integers

Integers help in storing numbers in contracts. S olidity provides the following two types:

Signed integers: Signed integers can hold both negative and positive values.

Unsigned integers: Unsigned integers can hold only positive values along with zero. They can also hold negative values apart from positive and zero values.

There could be 32 different declarations of uint with different multiples of of 8, such as uint8, uint16, unit24, as far as uint256 bit.

Similarly, there are equivalent data types for integers such as int8, int16 till int256.

Integers

bool data type can be used to represent scenarios that have binary results, such as true or false, 1 or 0, and so on.

The valid values for this data type are true and false.

It is to be noted that bools in Solidity cannot be converted to integers,.

It's a value type and any assignment to other boolean variables creates a new copy.

The default value for bool in Solidity is false.

Byte

The byte data type Byte refers to 8 bit signed integers.

Solidity has multiple flavors of the byte type. It provides data types in the range from bytes1 to bytes32 inclusive, to represent varying byte lengths, as required.

These are called fixed sized byte arrays and are implemented as value types.

bytes1 data type represents 1 byte and **bytes2** represents 2 bytes.

We can also perform bitwise operations such as and, or, xor, not, and left and right shift operations on the byte data type

Byte

The **default value** for byte is 0x00 and it gets initialized with this value.

A byte can be assigned byte values in hexadecimal format, as: **bytes1 aa = 0x65;**

A byte can be assigned integer values in decimal format, as: **bytes1 bb = 10;**

A byte can be assigned negative integer values in decimal format, as: **bytes1 ee = -100;**

A byte can be assigned character values as : **bytes1 dd**

Array

Arrays refer to groups of values of the same type.

Arrays help in storing these values together and ease the process of iterating, sorting, and searching for individuals or subsets of elements within this group.

Solidity provides rich array constructs that cater to different needs.

An example of an array in Solidity is as: **uint[5] intArray ;**

Arrays in Solidity can be fixed or dynamic.

Fixed Arrays

Fixed arrays refer to arrays that have a pre-determined size mentioned at declaration.

Examples of fixed arrays are as follows:

int[5] age ; // array of int with 5 fixed storage space allocated

byte[4] flags ; // array of byte with 4 fixed storage space allocated

Fixed arrays cannot be initialized using the new keyword.

They can only be initialized inline, as: **int[5] age = [int(10), 20,30,40,50] ;**

They can also be initialized inline within a function later, as: **int[5] age ; age = [int(10),2,3,4,5];**

Dynamic Arrays

Arrays that do not have a pre-determined size at the time of declaration; their size is determined at runtime.

int[] age ; // Storage is allocated during assignment
byte[] flags ; // array of byte with no fixed storage space allocated

The initialization can happen at the time of declaration as follows:
int[] age = [int(10), 20,30,40,50] ;
int[] age = new int[](5) ;

The initialization can also happen within a function later in the following two different steps:
int[] age ;
age = new int[](5) ;

Arrays Properties

Due to the multiple types of array, not every type supports all of these properties.

index: This property used for reading individual array elements is supported by all types of arrays, except for the string type. The index property for writing to individual array element is supported for dynamic arrays, fixed arrays, and the bytes type only. Writing is not supported for string and fixed sized byte arrays.

push: This property is supported by dynamic arrays only.

length: This property is supported by all arrays from read perspective, except for the string type. Only dynamic arrays and bytes support modifying the length property.

Enumerations

Enums are value types comprising a pre-defined list of constant values.

They are passed by values and each copy maintains its own value.

Enums cannot be declared within functions and are declared within the global contract.

Predefined constants are assigned consecutively, increasing integer values starting from zero.

It is to be noted that **web3** and **Decentralized Applications (DApp)** do not understand an enum declared within a contract. They will get an integer value corresponding to the enum constant.

Address

“address” is a 20 bytes data type, specifically designed to hold account addresses.

It can hold contract account addresses as well as externally owned account addresses.

Address is a value type and it creates a new copy while being assigned to another variable.

Address has a balance property that returns the amount of Ether available in account and has a few functions for transferring Ether to accounts and invoking contract functions.

It provides the two functions to transfer Ether: **transfer** & **send**

Address

Transfer function is a better alternative for transferring Ether to an account than the send function.

Send function returns a boolean value depending on successful execution of the Ether transfer while the transfer function raises an exception and returns the Ether to the caller.

It also provides the following three functions for invoking the contract function:

Call

DelegateCall

Callcode

Mapping

Mappings are one of the most used complex data types in Solidity.

Mappings are similar to hash tables or dictionaries in other languages.

They help in storing **key-value pairs** and enable retrieving values based on the supplied key.

Mappings are declared using the mapping keyword followed by data types for both key

and value separated by the => notation.

Example of mapping is as: **Mapping (uint => address) Names ;**

*uint data type is used for storing the **keys** and the address data type is used for storing the **values**. Names is used as an identifier for the mapping*