

**PIAIC**

# **Introduction to Blockchain Technology**

# Blockchain and Crypto currencies are shaking the system



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

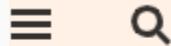
Vice President Luis de Guindos  
Preceded by Mario Draghi

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  
<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

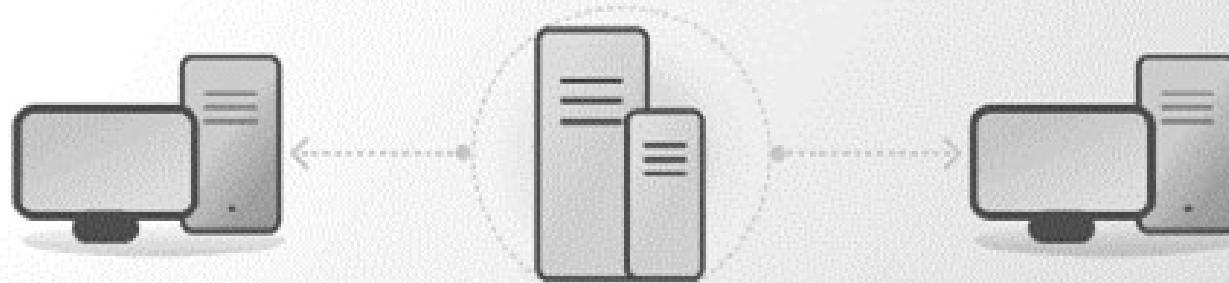
# Is Blockchain Technology

**The New Internet?**

# Web 1.0 : Read-Only (1990-2004)

**Web 1.0.**

1990 - 2004



Static websites

# Web 2.0: Read-Write (2004-Now)

## Web 2.0.

2004 - The Present



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)



**“DATA IS THE NEW GOLD”**

# 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 paradigm!

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.

Web3  
2014 - The Future?



# 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



# Industries than can be Disrupted by 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.



*Distributed*

# 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 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.

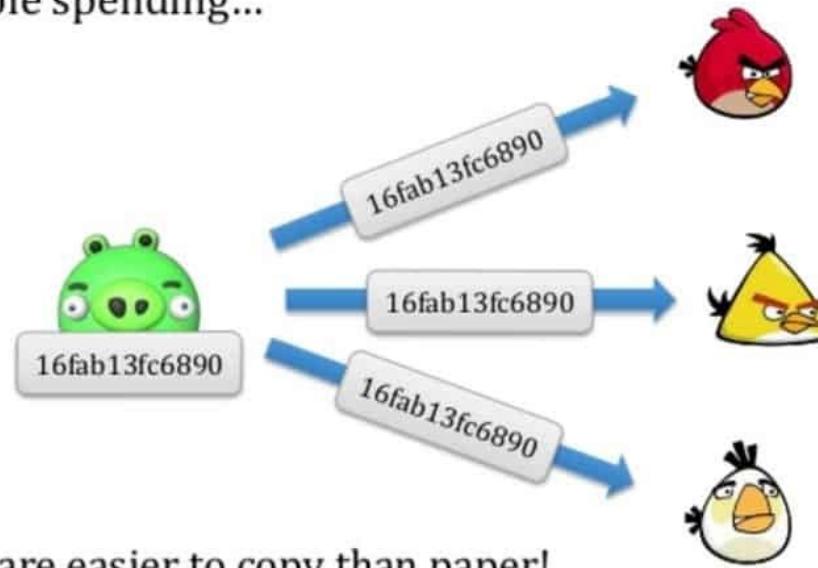
Most normal databases, i.e. SQL database, have someone in charge who can change the entries (e.g. giving themselves a million X dollars).

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.**

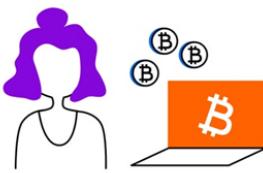
Double spending...



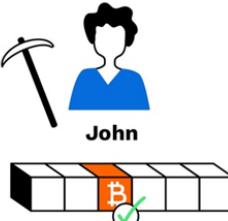
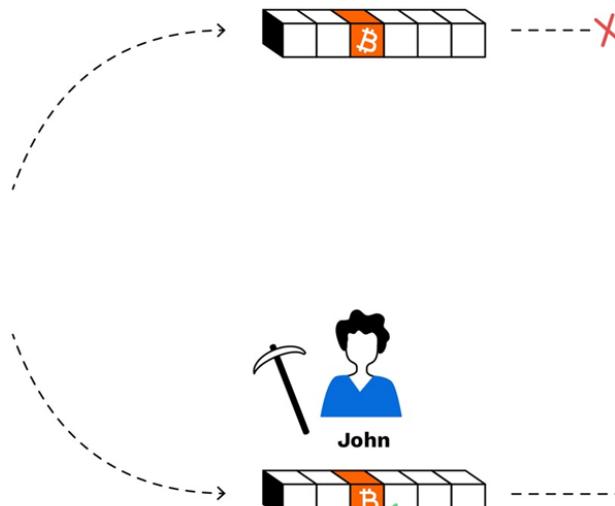
# Double Spend Problem

## What is Double Spending

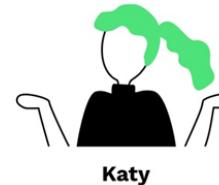
and why is it such a problem?



Without exception, all Bitcoin transactions are included in a block of transactions. Each block has a timestamp with encoded information that makes it more difficult to manipulate the blockchain.



The mechanism of the blockchain ensures that the party spending the bitcoins is the real owner.



Double spending is a type of deceit where the same money is promised to two parties but only delivered to one.



The technology behind Bitcoin ensures that the party who spends the bitcoins is the real owner by only processing verified transactions.

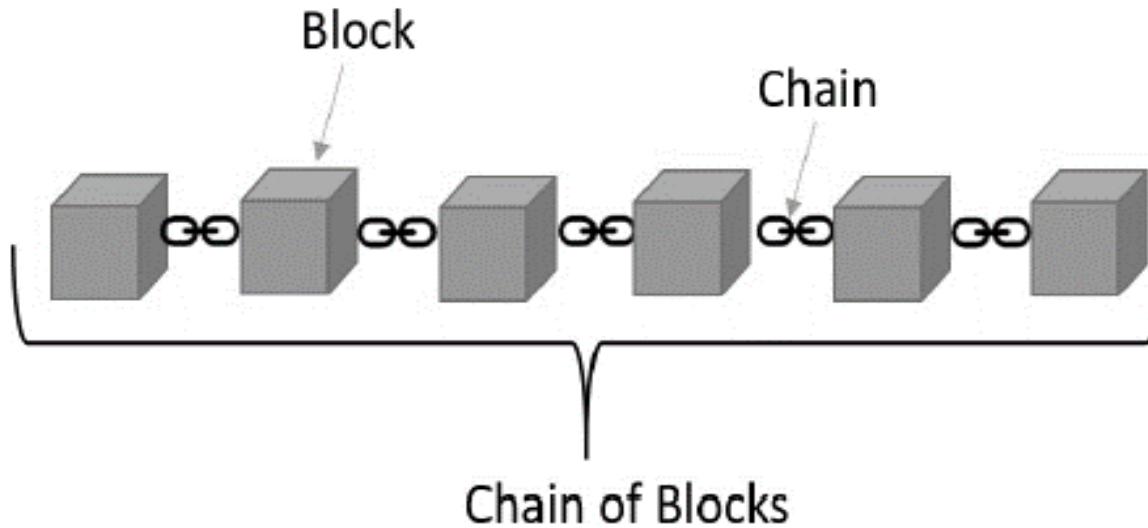
# What is Blockchain

“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 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 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...

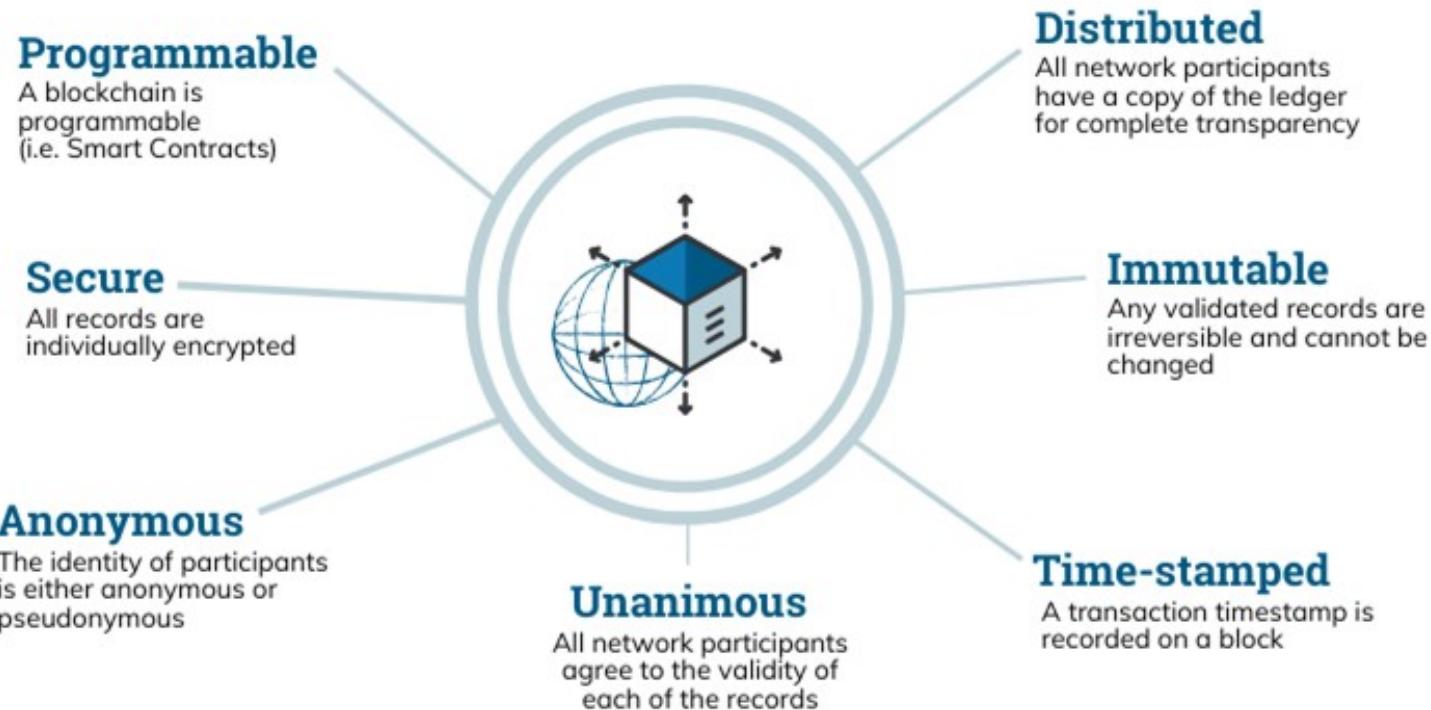
Blockchain is a type of DLT in which transactions are recorded with an immutable cryptographic signature called a [hash](#).

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 its 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”

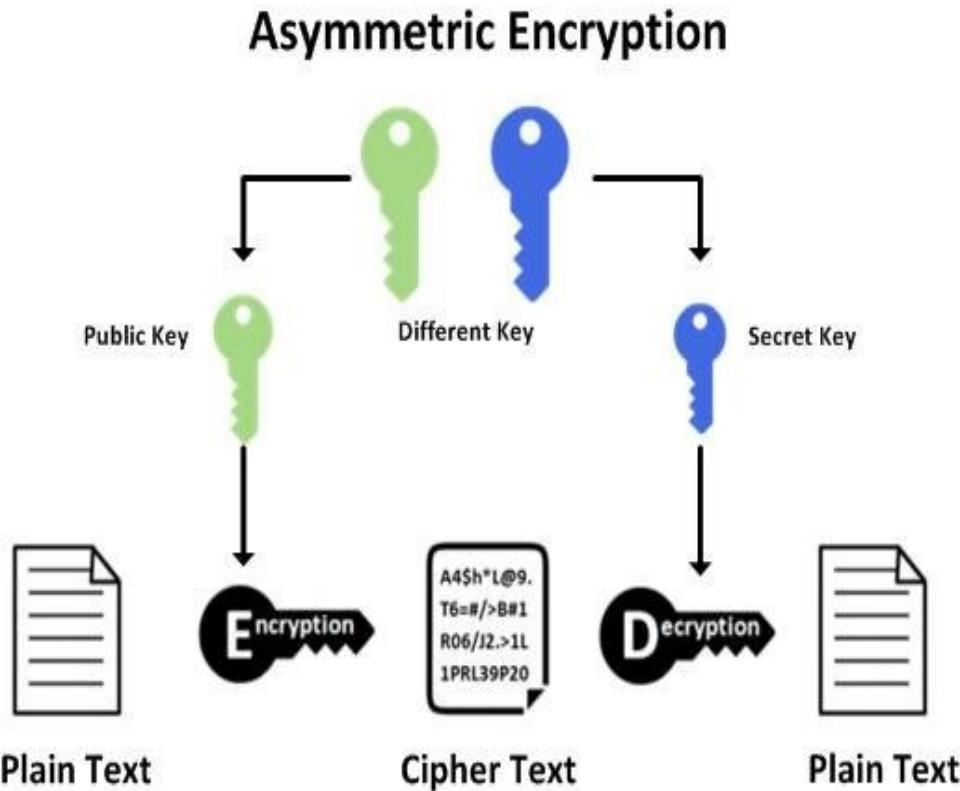
## Value:

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



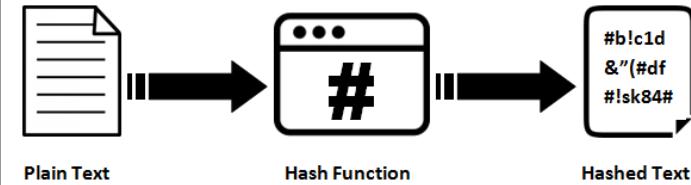
# Asymmetric Cryptography

- 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

Hashing Algorithm



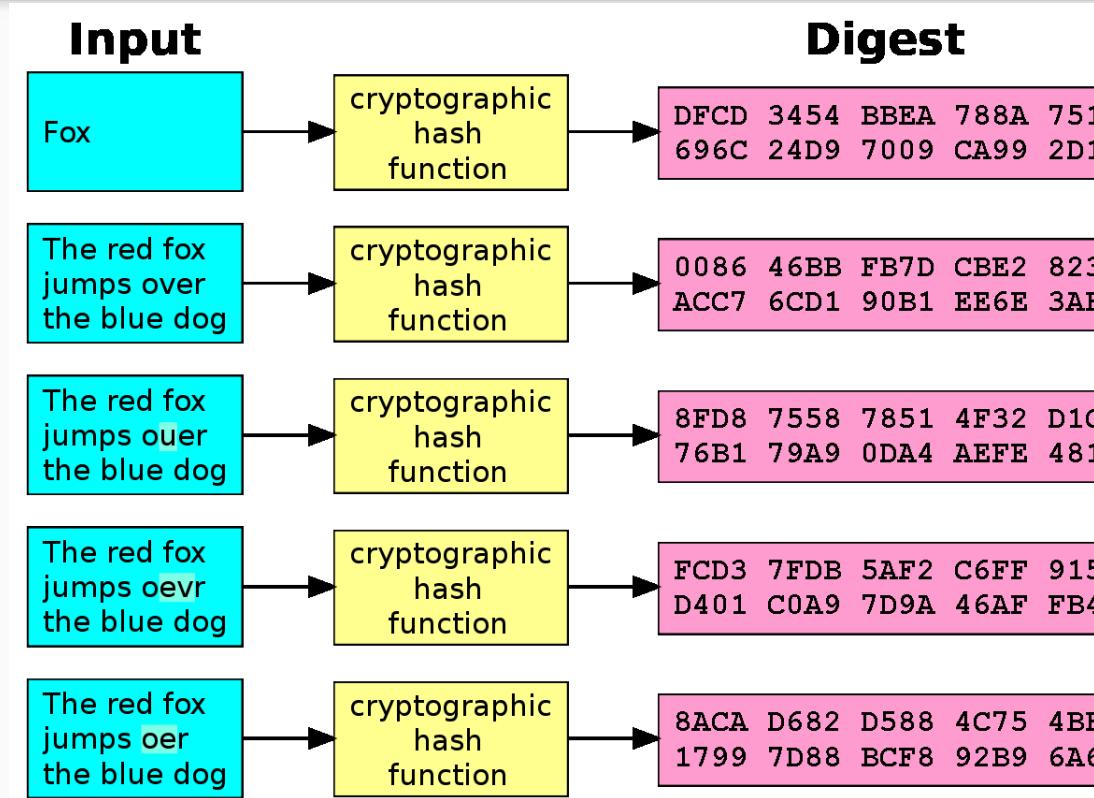
Is the process of transforming any input data into fixed length random character data, and it is not possible to regenerate or identify the original data from the resultant string data.

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

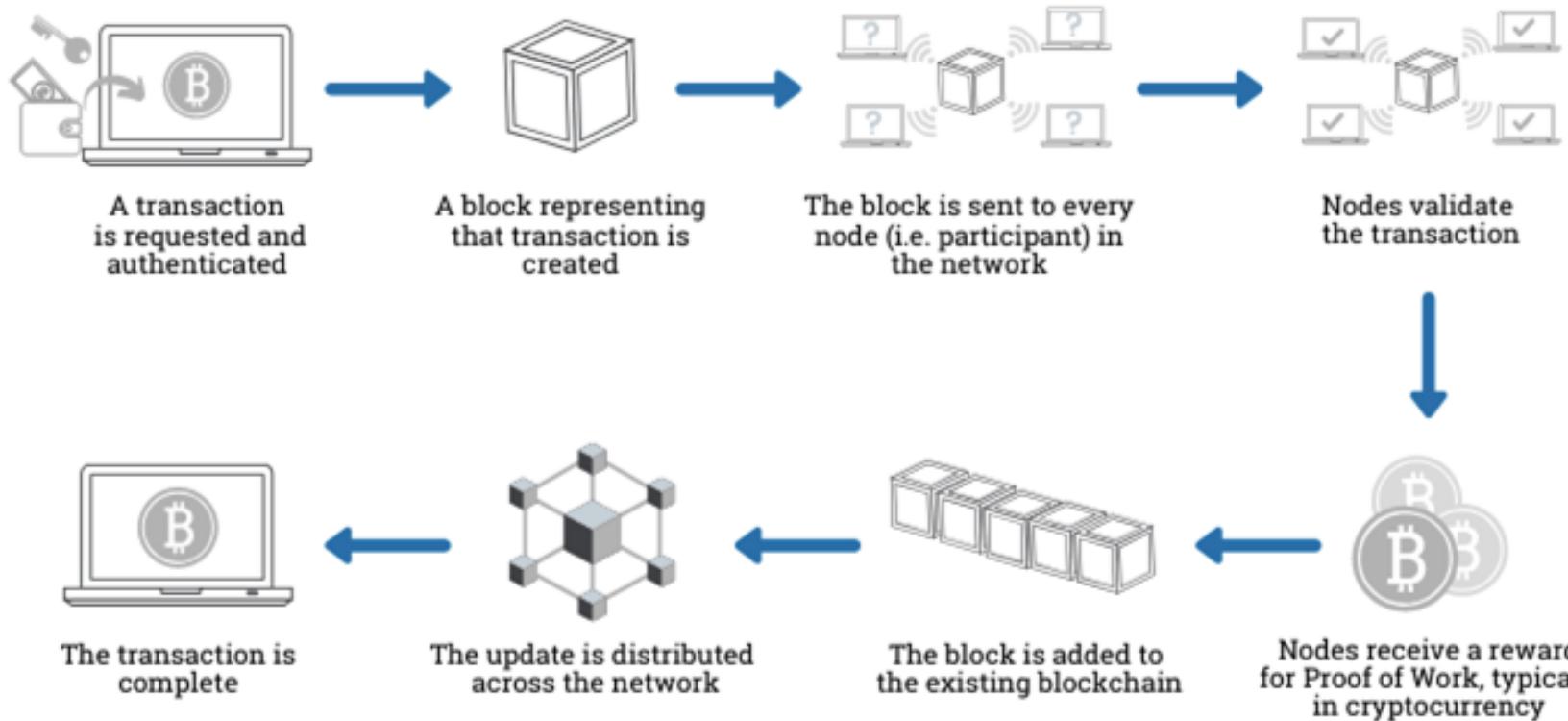
Hashing ensures that even a slight change in input data will completely change the output data, and no one can ascertain the change in the original data.

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

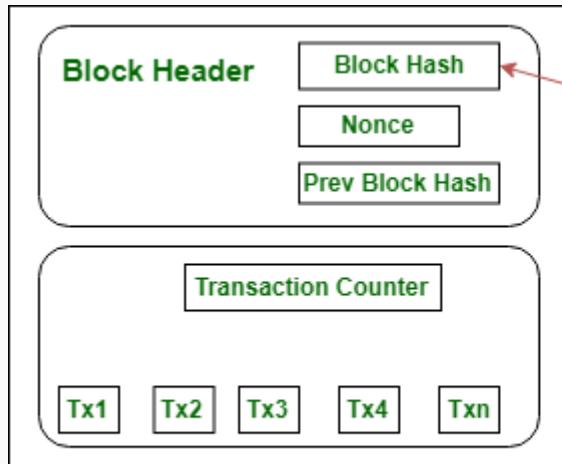
# Hashing - Example



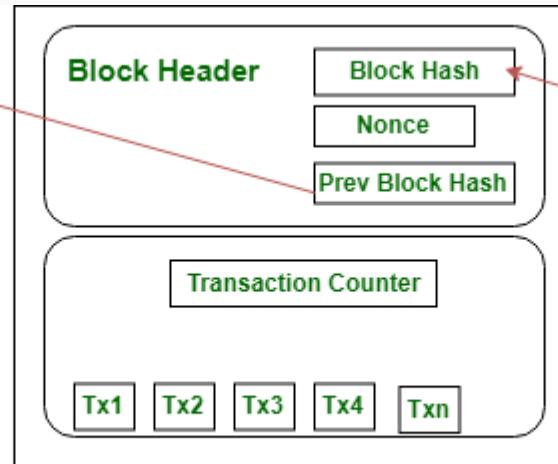
# How does a transaction get into blockchain



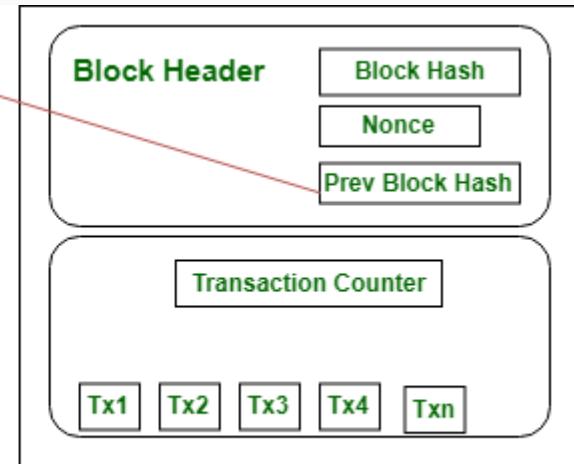
# Block in Blockchain



Block GFG1

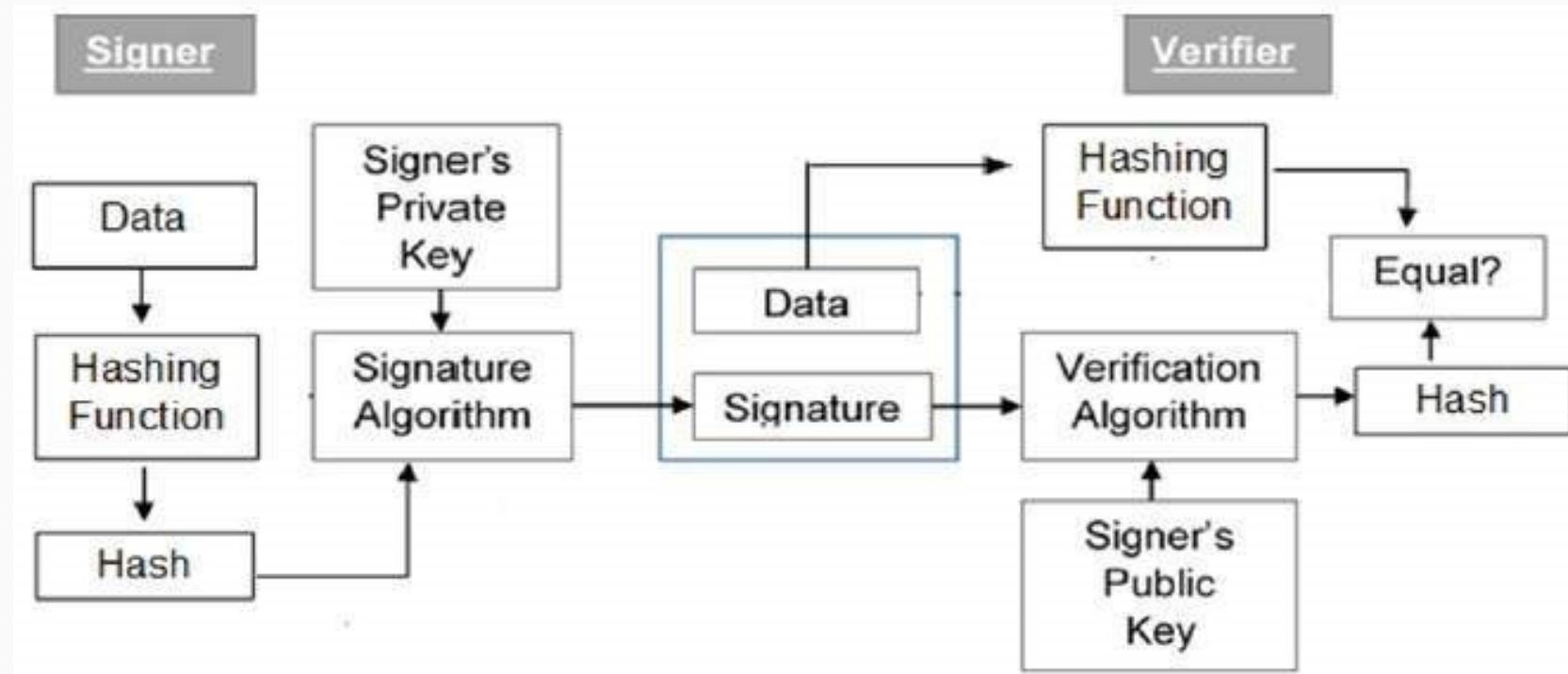


Block GFG2



Block GFG3

# Public Vs Private Key, Hash, Signer & Verifier



# New Book

Mastering Blockchain

Third Edition

By : Imran Bashir

EXPERT INSIGHT

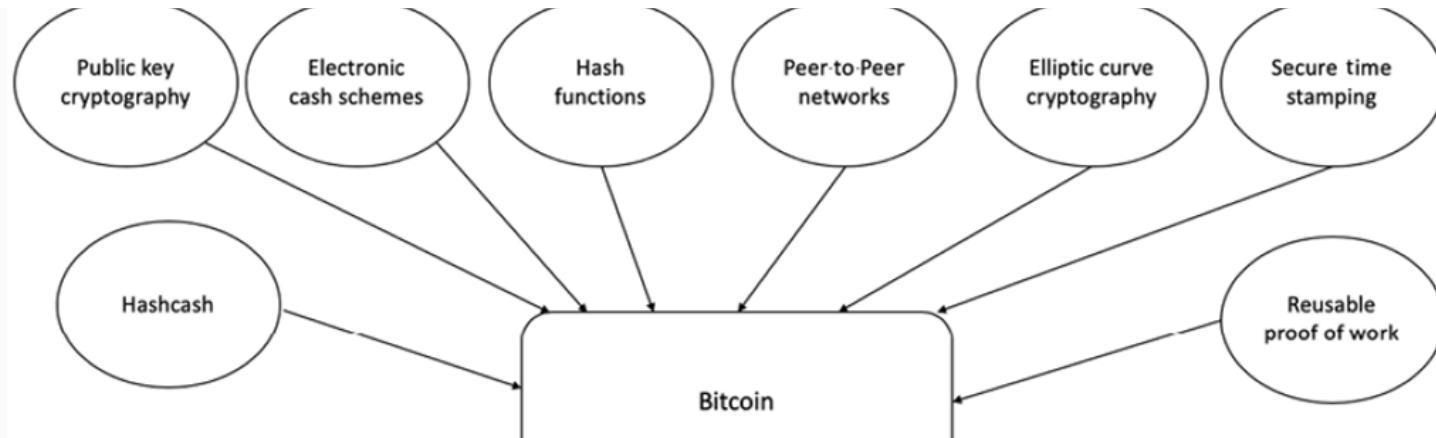
# Mastering Blockchain

A deep dive into distributed ledgers, consensus protocols, smart contracts, DApps, cryptocurrencies, Ethereum, and more



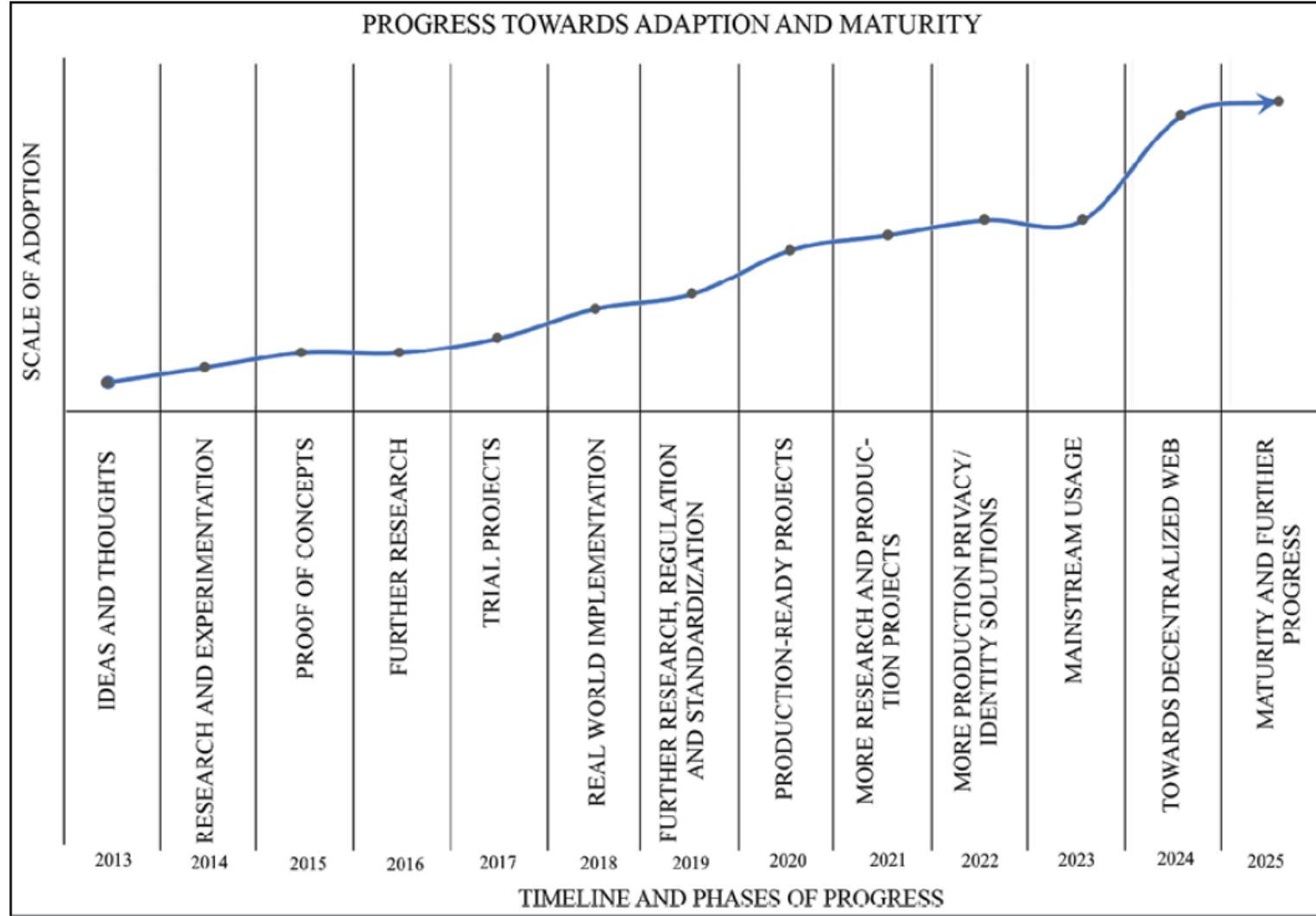
# How did blockchain technology develop?

- 1950s – Hash functions
- 1970s – Merkle trees - hashes in a tree structure
- 1970s continued – Research in distributed systems, consensus, state machine replication
- 1980s – Hash chains for secure logins
- 1990s – e-Cash for e-payments
- 1991 – Secure timestamping of digital documents.
- 1992 – Hashcash idea to combat junk emails
- 1994 – S/KEY application for Unix login.
- 1997/2002 – Hashcash
- 2008/2009 – Bitcoin (the first blockchain)



# Limitations of blockchain

- Scalability
- Adaptability
- Regulation
- Relatively immature technology
- Privacy



# Distributed systems

Blockchain originally intended to be a decentralized platform

It can be thought of as a system that has properties of the both decentralized and distributed paradigms.

It is a decentralized-distributed system.

# Node

A node can be defined as an individual player in a distributed system.

All nodes are capable of sending and receiving messages to & from each other.

Nodes can be honest, faulty, or malicious, they have memory and a processor.

A node that exhibits irrational behavior is also known as a Byzantine node after the Byzantine Generals problem

# The Byzantine Generals problem

Retreat?



Attack?



Attack?



Attack?

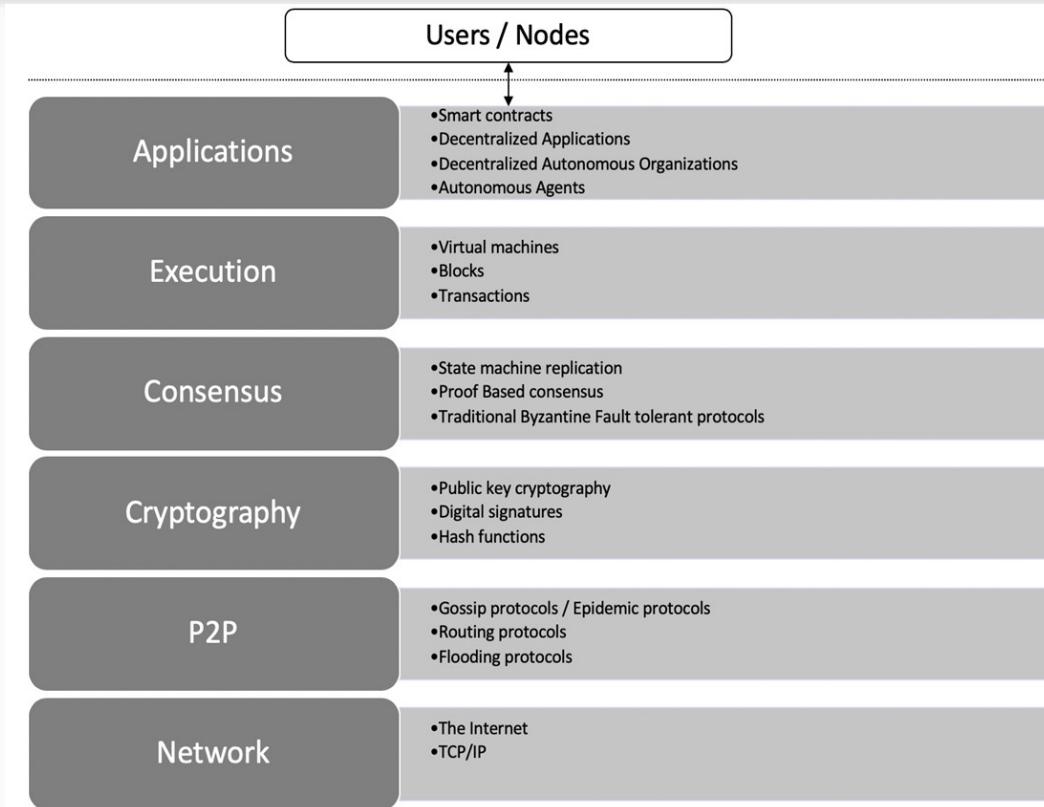


Retreat?



Attack or retreat?  
Consensus required to win

# Architectural view of Blockchain



# Blockchain Definition

**Layman's definition:** Blockchain is an ever-growing, secure, shared recordkeeping system in which each user of the data holds a copy of the records, which can only be updated if all parties involved in a transaction agree to update.

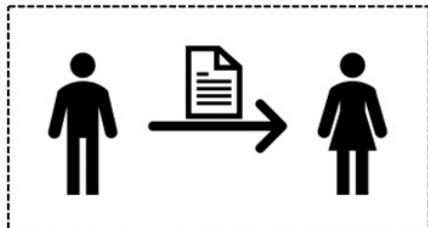
**Technical definition:** Blockchain is a peer-to-peer distributed ledger that is cryptographically-secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers.

# Blockchain Definition

- Peer-to-peer
- Distributed ledger
- Cryptographically secure
- Append only
- Updateable via consensus (consensus-driven)

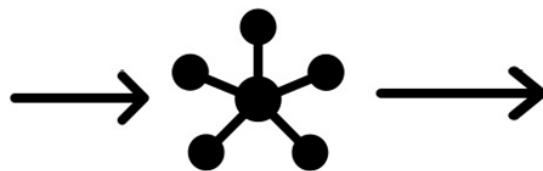
# How blockchain works

1- Transaction initiated



Smart contract or transfer of value  
e.g. User A transacts with User B

2- Transaction broadcast



3- Find new block (mining)



4- New block found (mined)



5- Add new block to the blockchain

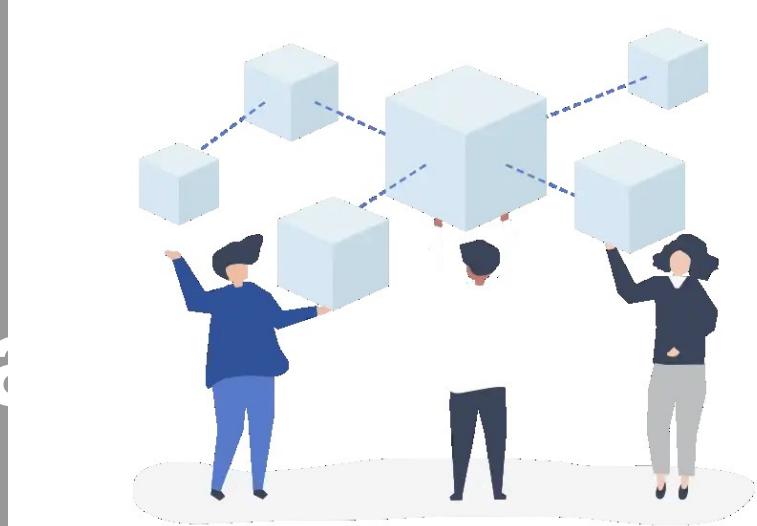
# Generic Element of Blockchain

- Addresses
- Accounts
- Transactions
- Blocks
- Peer-to-peer network
- Scripting or programming language
- Virtual machine
- State machine
- Nodes
- Smart contracts

# Features of Blockchain - Summary

- Distributed consensus
- Transaction verification
- Platform for smart contracts
- Transferring value between peers
- Generation of cryptocurrency
- Provider of security
- Immutability
- Uniqueness

# Type of Blockchain



# Public Blockchain

Are not owned by anyone.

They are open to the public & anyone can participate as a node (decisionmaking)

Users may or may not be rewarded for their participation.

All users of these "permissionless" ledgers maintain a copy of the ledger

use a distributed consensus mechanism to decide the eventual ledger state.

Bitcoin and Ethereum are both considered public blockchains.

# Private Blockchain

As the name implies, private blockchains are just that—private.

They are open only to a consortium or group of individuals or organizations who have decided to share the ledger among themselves.

There are various blockchains now available in this category, such as Quorum.

# Semi Private

**Part of the blockchain is private and part of it is public.**

**Note that this is still just a concept today, and no real-world proofs of concept have yet been developed.**

**With a semi-private blockchain, the private part is controlled by a group of individuals, while the public part is open for participation by anyone.**

**This type of blockchain can also be called a "semi-decentralized" model, where it is controlled by a single entity but still allows for multiple users to join the network by following appropriate procedures.**

# Permissioned Blockchain

Where participants of the network are already known and trusted.

Permissioned ledgers do not need to use a distributed consensus mechanism;

An agreement protocol is used to maintain a shared version of the truth about the state of the records on the blockchain.

All verifiers are already preselected by a central authority

No need for a mining mechanism.

# Consensus



# Consensus

Consensus is the backbone of a blockchain, as it provides the decentralization of control through an optional process known as mining.

The choice of the consensus algorithm to utilize is governed by the type of blockchain in use; that is, not all consensus mechanisms are suitable for all types of blockchains.

For example, in public permissionless blockchains, it would make sense to use PoW instead of mechanisms that are more suitable for permissioned blockchains, such as Proof of Authority (PoA)

# Consensus

Consensus is a process of achieving agreement between distrusting nodes on the final state of data.

It is easy to reach an agreement between two nodes i.e. client-server systems, but when multiple nodes are participating in a distributed system and they need to agree on a single value, it becomes quite a challenge to achieve consensus.

This process of attaining agreement on a common state or value among multiple nodes despite the failure of some nodes is known as distributed consensus.

# Consensus mechanism

A consensus mechanism is a set of steps that are taken by most or all nodes in a blockchain to agree on a proposed state or value.

The following describes these requirements:

Agreement: All honest nodes decide on the same value.

Integrity: no node can make decision more than once in a single consensus cycle.

Validity: The value agreed upon by all honest nodes must be the same

Fault tolerant: able to run correctly in presence of faulty/malicious/Byzantine nodes

Termination: All honest nodes terminate the execution of the consensus process and eventually reach a decision.

# Types of consensus mechanism

Proof-based consensus mechanisms: requires nodes to compete in a leader-election lottery, and the node that wins proposes the final value.

The algorithm works on the principle of providing proof of some work and the possession of some authority or tokens to win the right of proposing the next block.

I.e , PoW mechanism used in Bitcoin falls into this category, where a miner who solves the computational puzzle as proof of computational effort expended wins the right to add the next block to the blockchain.

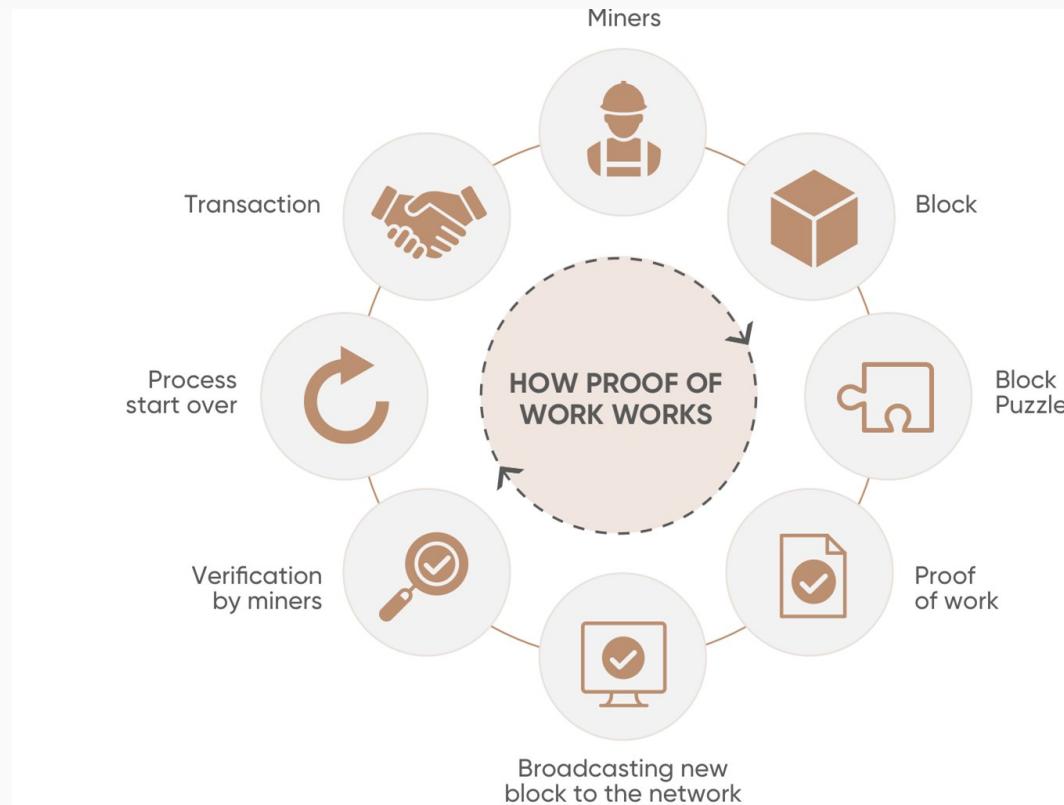
# Types of consensus mechanism

Traditional fault tolerance-based: With no compute-intensive operations

this type of consensus mechanism relies on a simple scheme of nodes that publish and verify signed messages in a number of phases.

Eventually, when a certain number of messages are received over a period of rounds (phases), then an agreement is reached.

# 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

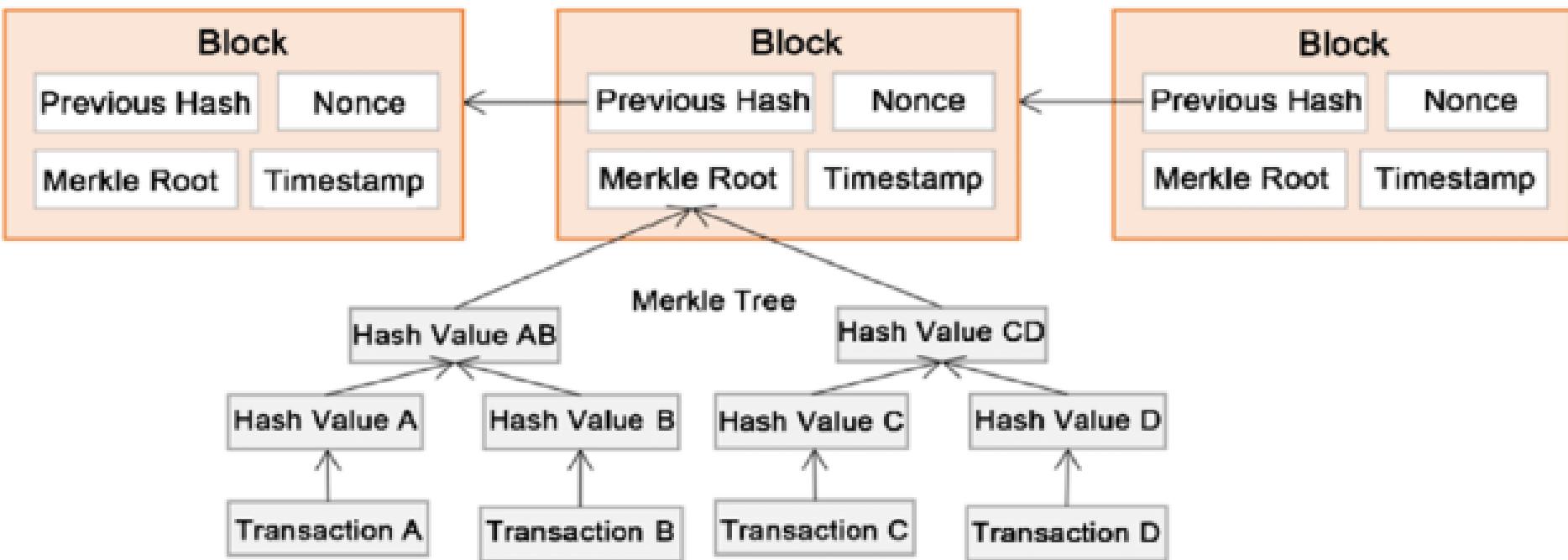
# How Block Header is created



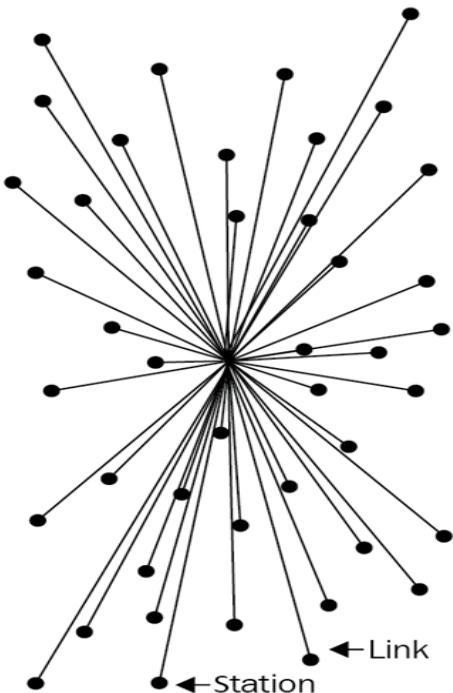
Proof of Work

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.

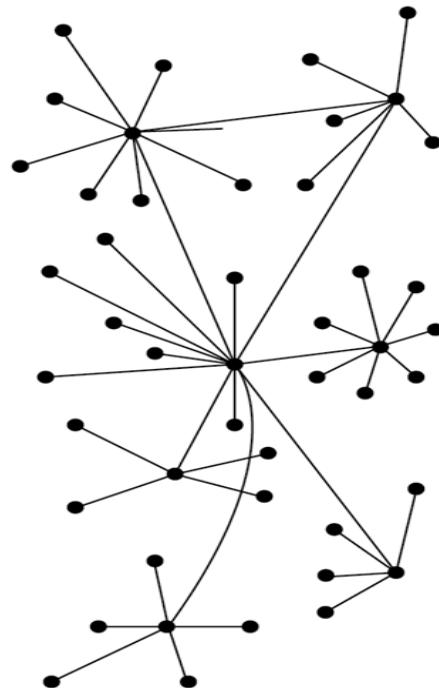
# Block Structure



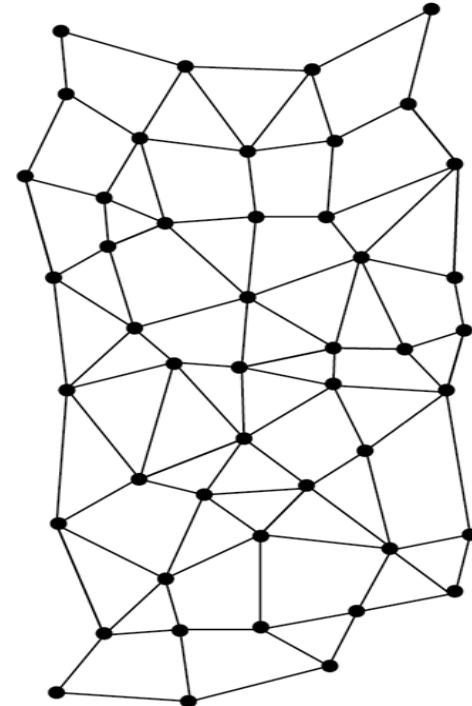
# Decentralization Vs Distributed



CENTRALIZED



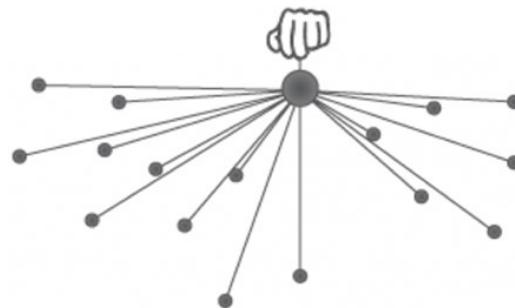
DECENTRALIZED



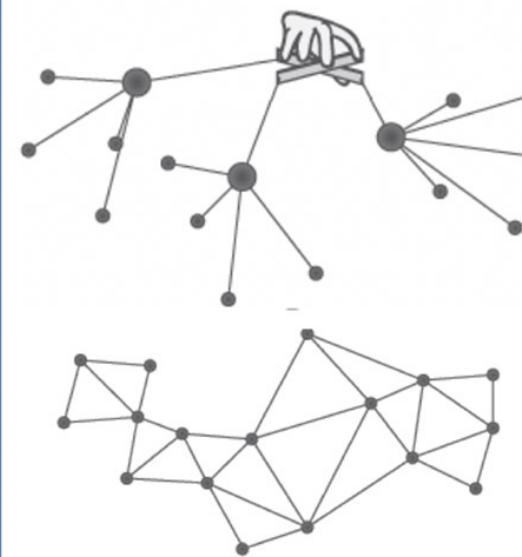
DISTRIBUTED

# Decentralization Vs Distributed

Centralized



Distributed



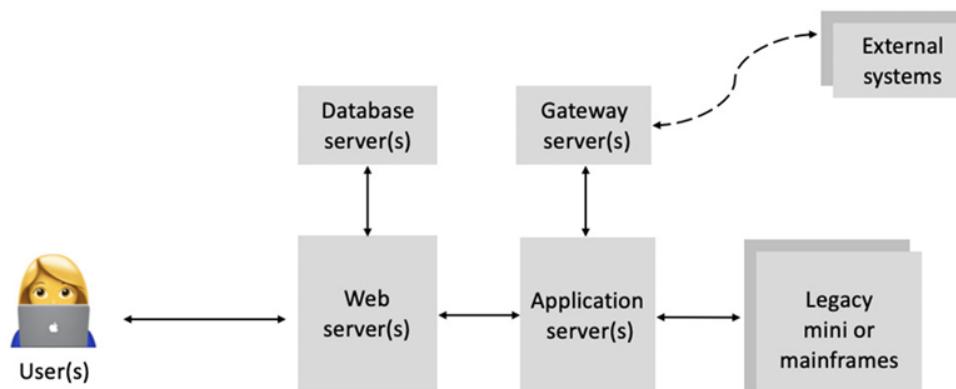
Decentralized



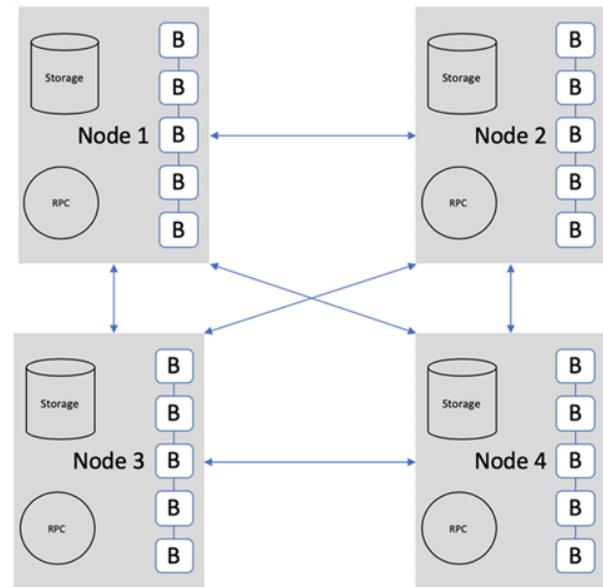
Notice no hand  
means no central controller / authority

# Decentralization Vs Distributed

## Distributed



## Decentralized



# Types of Decentralization

## Fully Centralized

Where central intermediaries are in control, for example the current financial system



## Semi- decentralized

Where intermediaries compete with each other, for example when multiple service providers compete to win a contract



## Fully Decentralized

Where no intermediaries are required, for example in Bitcoin

# Centralized Vs Decentralized

<b>Feature</b>	<b>Centralized</b>	<b>Decentralized</b>
Ownership	Service provider	All users
Architecture	Client/server	Distributed, different topologies
Security	Basic	More secure
High availability	No	Yes
Fault tolerance	Basic, single point of failure	Highly tolerant, as service is replicated

# Centralized Vs Decentralized

Collusion resistance	Basic, because it's under the control of a group or even single individual	Highly resistant, as consensus algorithms ensure defense against adversaries
Application architecture	Single application	Application replicated across all nodes on the network
Trust	Consumers have to trust the service provider	No mutual trust required
Cost for consumer	Higher	Lower

# DO Vs DAO Vs DAC

Decentralized Organizations (DO) are software programs that run on a blockchain and are based on the idea of actual organizations with people and protocols.

Once a DO is added to the blockchain in the form of a smart contract or a set of smart contracts, it becomes decentralized and parties interact with each other based on the code defined within the DO software.

# DO Vs DAO Vs DAC

Decentralized autonomous organizations (DAO) is also a computer program that runs on top of a blockchain, and embedded within it are governance and business logic rules.

The main difference, is that DAOs are autonomous, which means that they are fully automated and contain artificially intelligent logic. DOs, on the other hand rely on human input to execute business logic.

Ethereum blockchain led the way with the introduction of DAOs. In a DAO, the code is considered the governing entity rather than people or paper contracts.

# DO Vs DAO Vs DAC

Decentralized autonomous corporations (DACs) general distinction is that DAOs are usually considered to be nonprofit,

whereas DACs can earn a profit via shares offered to the participants and to whom they can pay dividends.

DACs can run a business automatically without human intervention based on the logic programmed into them.

# Dapps - Decentralized Applications

DAOs, DACs, and DOs are DApps that run on top of a blockchain in a peer-to-peer network

Type 1: Run on their own dedicated blockchain,

for example, standard smart contract based DApps running on Ethereum. If required, they make use of a native token, for example, ETH on Ethereum blockchain.

Etlance is a DApp that makes use of ETH to provide a job market.

# Dapps - Decentralized Applications

Type 2: Use an existing established blockchain. that is, make use of Type 1 blockchain and bear custom protocols and tokens, for example, smart contract based tokenization DApps running Ethereum blockchain.

An example is DAI, which is built on top of Ethereum blockchain, but contains its own stable coins and mechanism of distribution and control.

Type 3: Use the protocols of Type 2 DApps;

for example, the SAFENetwork uses the OMNI network protocol.

# Requirements of Dapps

Application to be considered decentralized, it must meet the following criteria.

1. DApp should be fully open source and autonomous, and no single entity should be in control of a majority of its tokens. All changes to the application must be consensus-driven based on the feedback given by the community.
2. Data and records of the application must be cryptographically secured and stored on a public, decentralized blockchain to avoid any central points of failure.

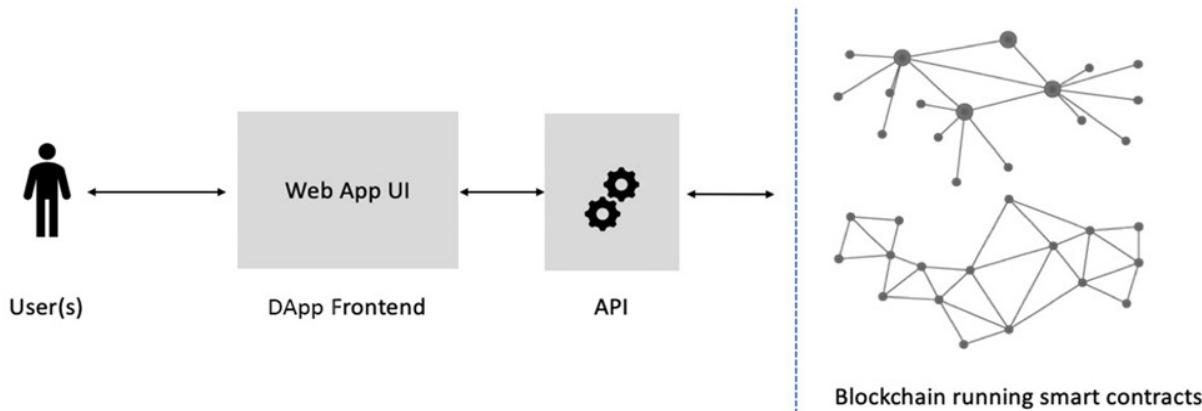
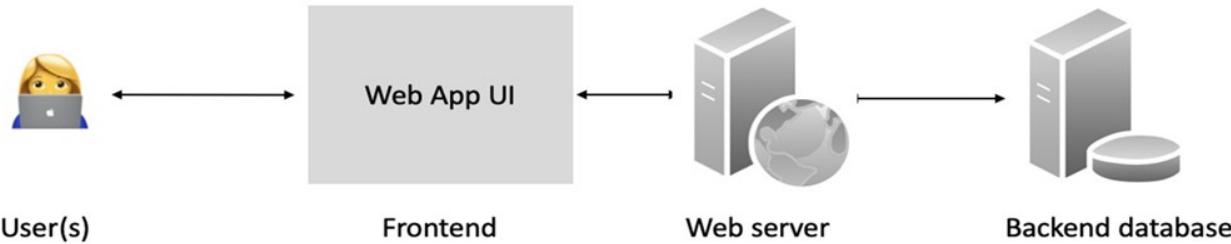
# Requirements of Dapps

Application to be considered decentralized, it must meet the following criteria.

3. A cryptographic token must be used by the application to provide access for and incentivize those who contribute value to the applications.
4. The tokens (if applicable) must be generated by the Dapp using consensus and an applicable cryptographic algorithm. This generation of tokens acts as a proof of the value to contributors (for example, miners).

DApps now provide all sorts of different services, including but not limited to financial applications, gaming, social media, and health.

# Decentralized Application (Dapps)



# Platforms for Decentralization

- Ethereum
- MaidSafe
- Lisk
- EOS

# Decentralized Finance (DeFi)

Finance is a business that is almost impossible to do without the involvement of a trusted third party. Some main disadvantages are listed as follows:

**Underdeveloped ecosystem:**

**Too technical:**

**Lack of regulation:**

**Human error:**

# Advantages of DeFi

DeFi comes with a number of advantages, such as inclusion, easy access, and cheaper services; however, it has its own challenges.

**Access barrier:**

**High cost:**

**Transparency issues:**

**Interoperability issues**

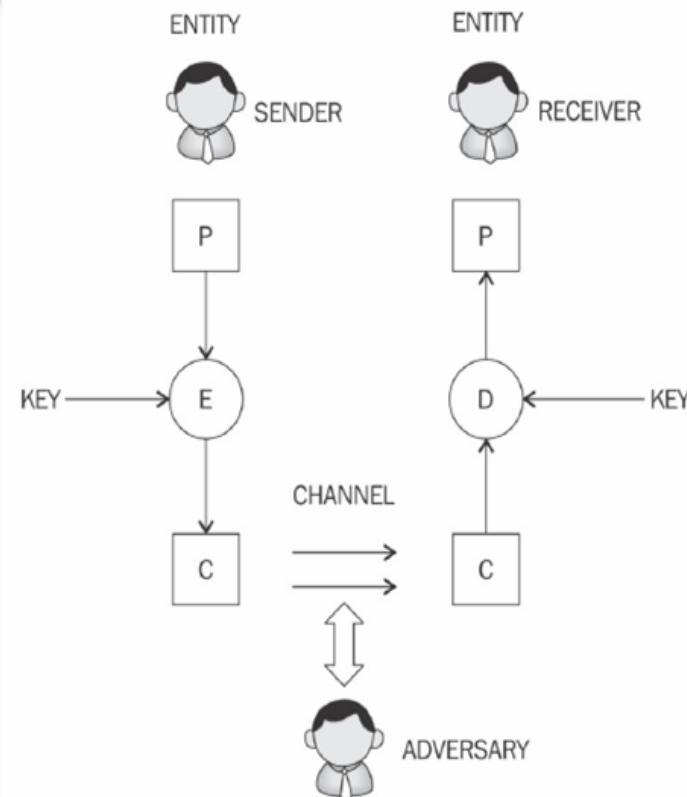
# Symmetric Cryptography

Chapter # 3

# Cryptography

Cryptography is the science of making information secure in the presence of adversaries.

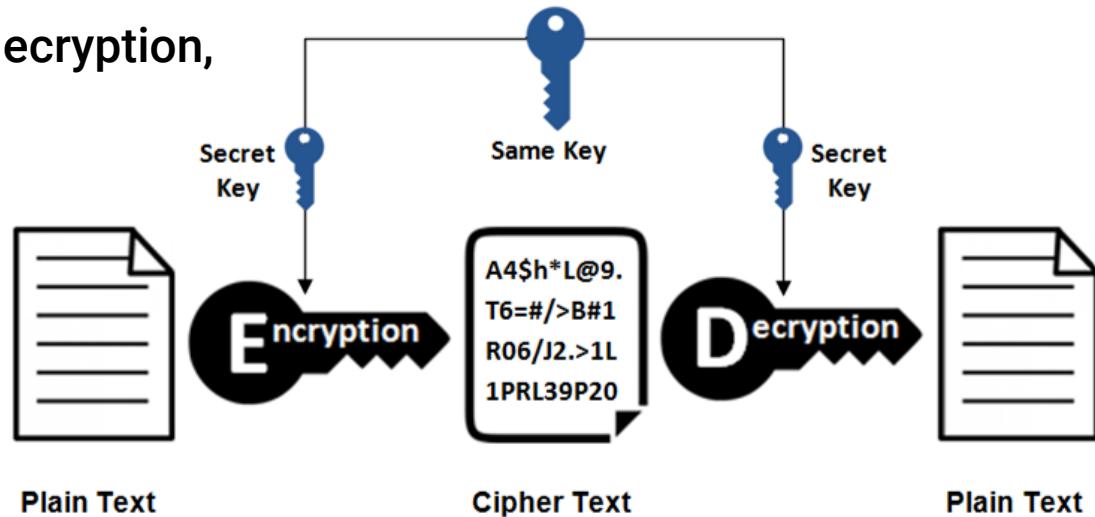
Ciphers are algorithms used to encrypt or decrypt data so that if intercepted by an adversary, the data is meaningless to them without decryption, which requires a secret key.



# Symmetric Cryptography

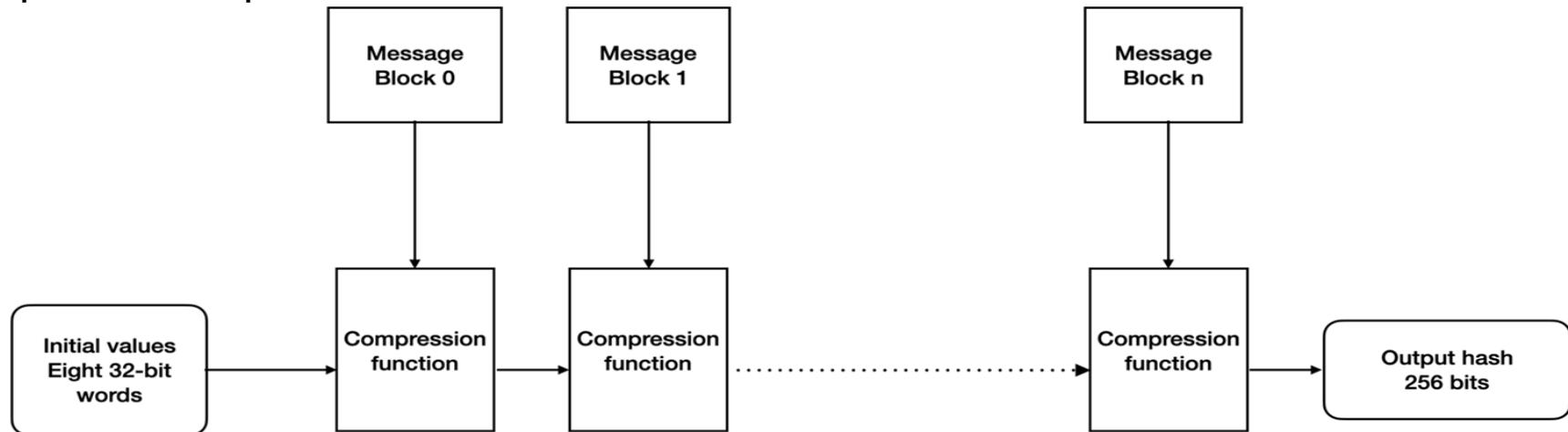
Is a type of cryptography where key that is used to encrypt the data is the same one that is used for decrypting the data. also known as shared key cryptography.

Only one key is required, which is shared between the sender and the receiver. It is used for both encryption and decryption,



# SHA - 256

SHA-256 takes the input message and divides it into equal blocks of 512 bits. Initial values of the initialization vector are composed of eight 32 bit words (256 bits) that are fed into the compression function with the first message. Subsequent blocks are fed into the compression function until all blocks are processed and finally, the output hash is produced.



# Cryptography provides various services

**Confidentiality** - assurance that information is only available to authorized entities

**Integrity** - assurance that information is modifiable only by authorized entities.

**Authentication** - assurance about the identity of an entity / validity of a message.

**Non-repudiation** - assurance that an entity cannot deny a previous commitment

**Accountability** - assurance that actions affecting security can be traced back to the responsible party.

# Cryptographic Primitives

Cryptographic primitives are the basic building blocks of a security protocol or system. In the following section, you are introduced to cryptographic algorithms that are essential for building secure protocols and systems.

Security protocol is a set of steps taken to achieve the required security goals by utilizing appropriate security mechanisms.

# Public Key Cryptography

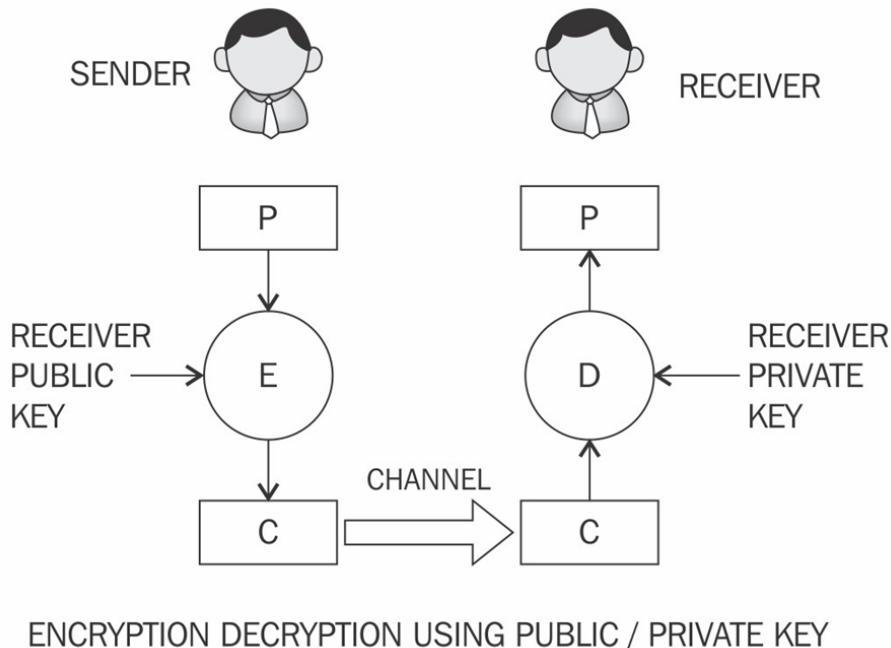
Chapter # 4

# Asymmetric Cryptography

## Encryption and decryption:

**Sender** uses the receiver's public key to encrypt, and the

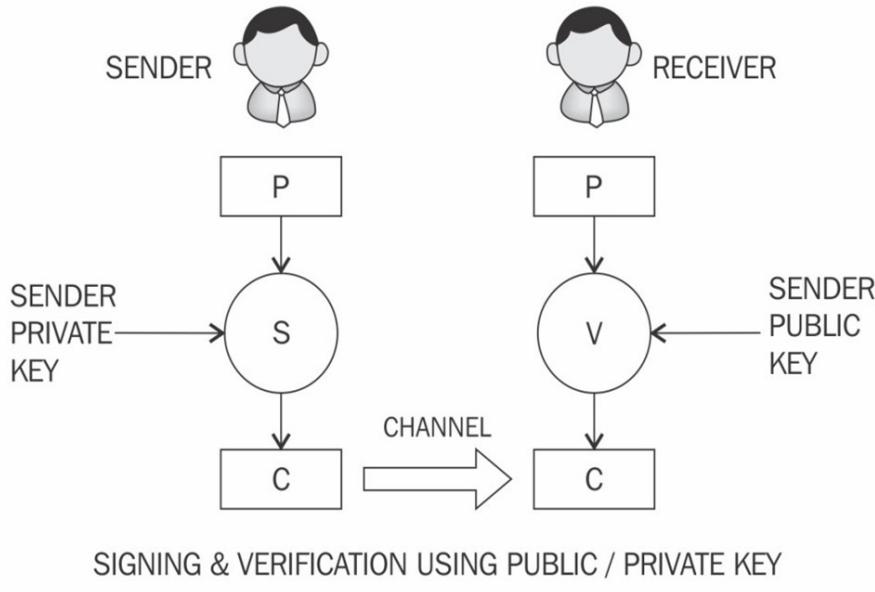
**Receiver** uses their private key to decrypt the message.



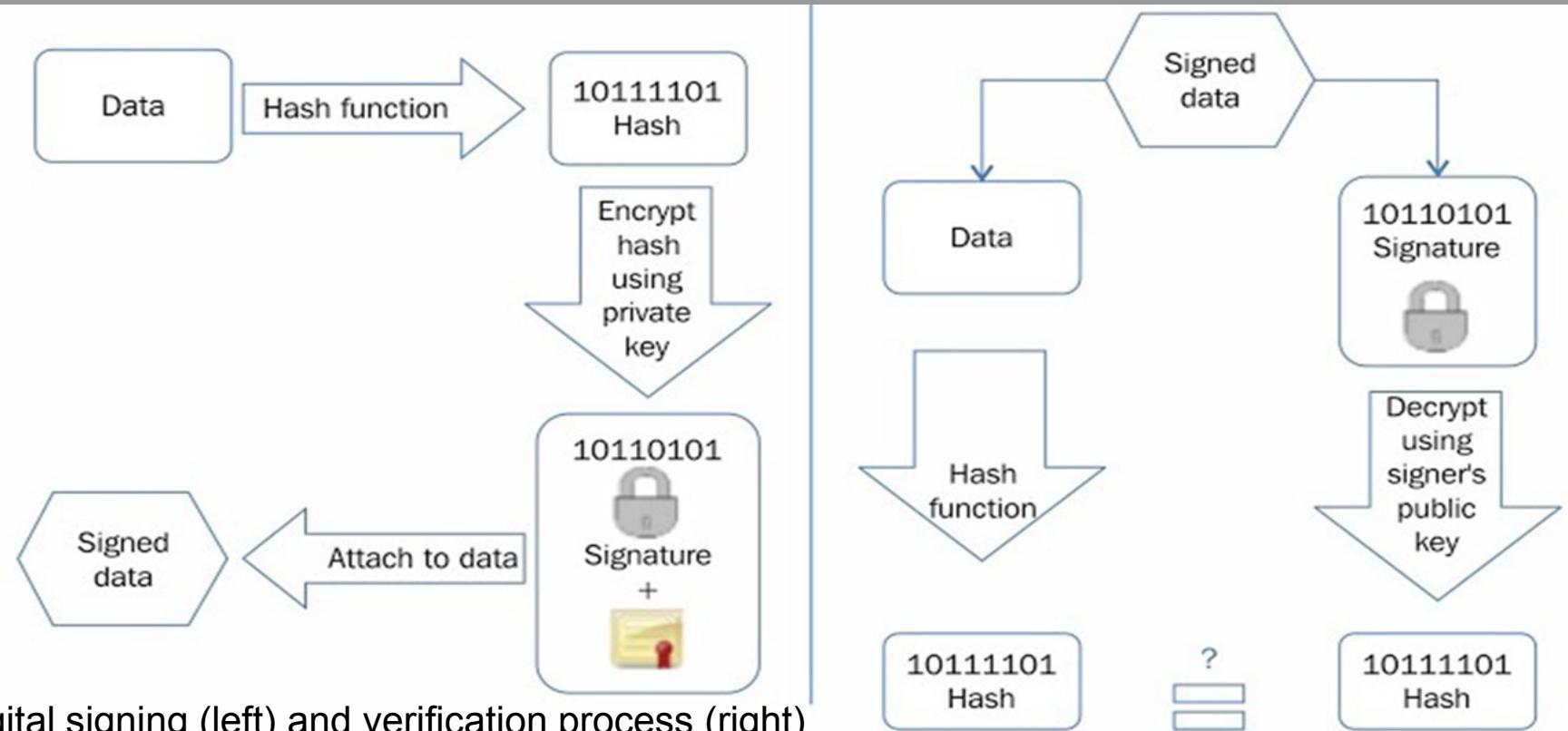
# Signing and Verification

**Sender** uses their private key to sign, and

**Receiver** uses the sender's public key to verify that the message has indeed been sent from the sender.



# Digital Signatures



# RSA digital signature algorithms

Digital signature are calculated using the two steps listed here.

- 1. Calculate the hash value of the data packet.** This will provide the data integrity guarantee, as the hash can be computed at the receiver's end again and matched with the original hash to check whether the data has been modified in transit. Technically, message signing can work without hashing the data first, but that is not considered secure.
- 2. Sign the hash value with the signer's private key.** As only the signer has the private key, the authenticity of the signature and the signed data is ensured.

# Digital Signatures - Properties

Authenticity means that the digital signatures are verifiable by a receiving party.

Unforgeability only the sender of the message can use the signing functionality using the private key. Digital signatures also provide protection against forgery.

Forgery means an adversary fabricating a valid signature for a message without any access to the legitimate signer's private key. which means that no one else can produce the signed message produced by a legitimate sender.

Non-reusability means that the digital signature cannot be separated from a message and used again for another message. In other words,

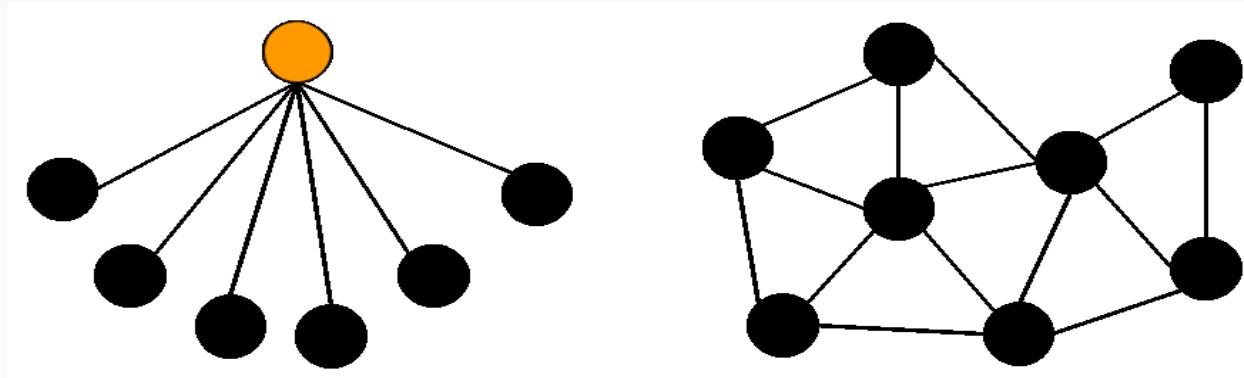
# Consensus Algorithm

## Chapter # 5



# Distributed Network

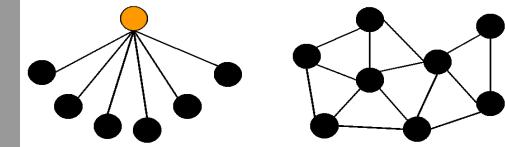
Participants in the distributed network work collectively in verifying and validating a huge amount of transactions in real-time.



# Categories of Distributed Systems

- Distributed systems are classified into two main categories  
(1) message passing and (2) shared memory.
- In blockchain we are concerned with “message passing” category.

# Challenges in Distributed System



- How are decisions made in a distributed network?
- How do all participants agree to the current status of the database if there is no central authority?
- How can we confirm that there is only one version of the truth in everyone's database?
- How can we be sure that transactions in the network are real and authentic?
- How do we ensure that everyone works together to secure the network and no one will act adversely?

# Consensus Algorithm

The advent of blockchain technology provided a solution to solve these critical questions, through the use of consensus mechanisms.

In distributed systems it is vital to have consensus (agreement) of nodes on the copy of the data with them.

To achieve this consensus (agreement) there are various consensus mechanisms (algorithm).

# What is Consensus?

- Consensus is a process of achieving agreement between distrusting nodes on the final state of data.
- It is easy to reach an agreement between two nodes i.e. client-server systems, but when multiple nodes are participating in a distributed system and they need to agree on a single value, it becomes quite a challenge to achieve consensus.
- This process of attaining agreement on a common state or value among multiple nodes despite the failure of some nodes is known as distributed consensus.

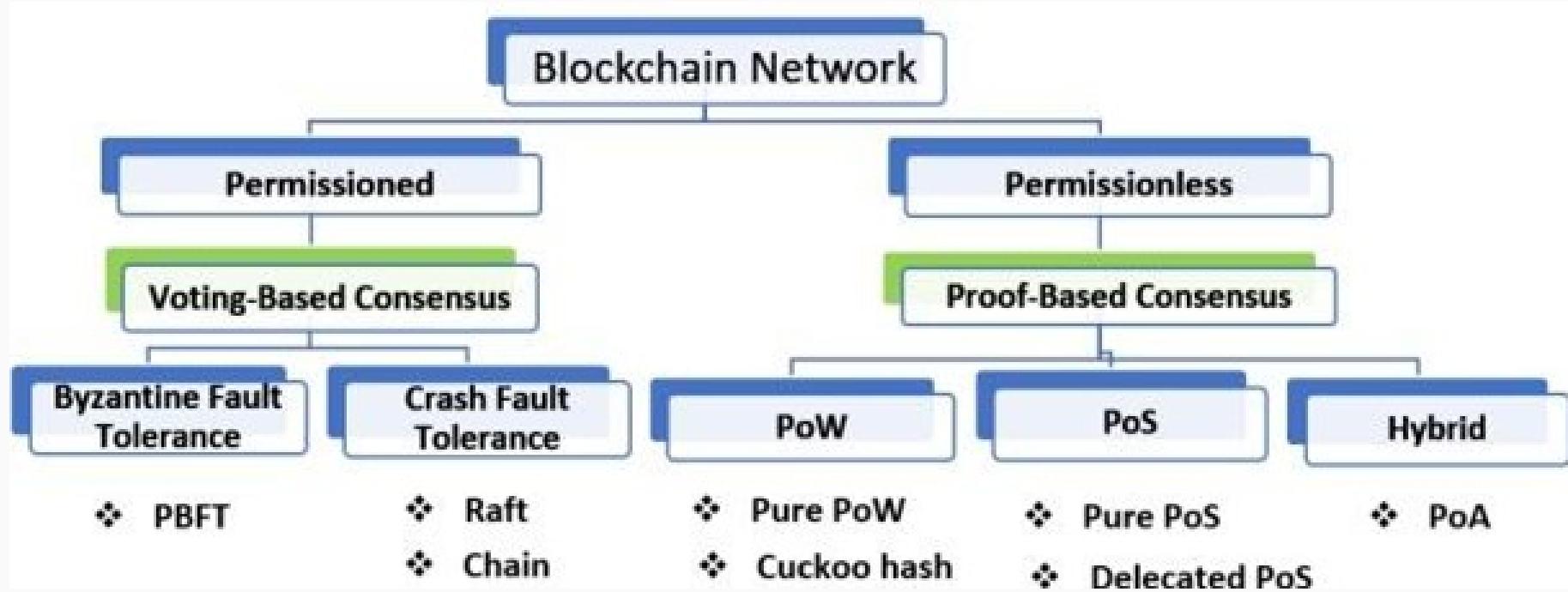
# Consensus Mechanisms

A consensus mechanism is a set of steps that are taken by most or all nodes in a blockchain to agree on a proposed state or value.

The following describes these requirements:

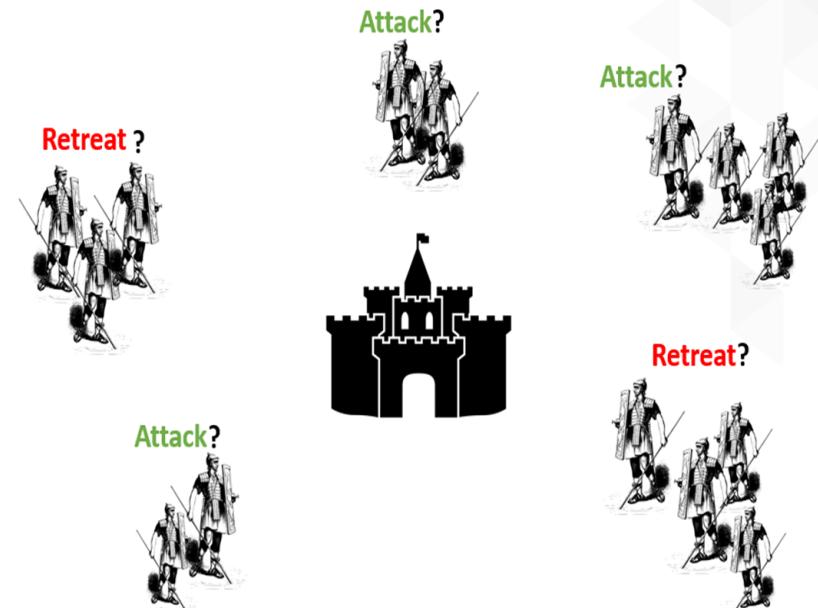
1. Agreement: All honest nodes decide on the same value.
2. Integrity: no node can make decision more than once in a single consensus cycle.
3. Validity: The value agreed upon by all honest nodes must be the same
4. Fault tolerant: able to run correctly in presence of faulty/malicious/Byzantine nodes
5. Termination: All honest nodes terminate the execution of the consensus process and eventually reach a decision.

# Types of Consensus Mechanism



# The Byzantine Generals Problem

- It is a hypothetical situation where an Army is divided into various units headed by a General.
- The General can communicate through a messenger.
- There is a probability of any General becoming treacherous



Attack or retreat?  
Consensus required to win

# Fault Tolerance

- A fundamental requirement in a consensus mechanism is that it is **fault tolerance**.
- It must be able to **tolerate a number of failures** in a network and should continue to work even in the presence of faults.
- This naturally means that there has to be some limit to the number of faults a network can handle, since no network can operate correctly if a large majority of its nodes are failing.

# Fault Tolerance

## Types of fault tolerance:

- Crash Fault Tolerance (CFT)
- Byzantine Fault Tolerance (BFT)

## How to achieve fault tolerance:

- Replication
  - Active
  - Passive
- State machine replication

# Consensus Algorithms at a glance

## **Algorithm 1:** PoW of a transaction

1. Group the transactions that are required to be verified and create a block.
2. The server generates a Mathematical puzzle.
3. Miner nodes compete with each other to solve that mathematical puzzle.
4. The node which solves the puzzle first shares its block with the proof of work with all other nodes.
5. All the nodes verify the solution
  - 5.1. If the Solution is correct then go to **Step 6.**
  - 5.2. Else go to **step 8.**
6. The block is added to the existing Blockchain.
7. After successfully appending a block, the miner node gets rewarded.
8. Exit.

# Consensus Algorithms at a glance

## **Algorithm 2:** PoS of a transaction

1. Transactions are grouped in a block.
2. Validators will randomly select a block and bet on it.
3. Validators create a new block.
4. If Validators verify fraudulent transactions, then they lose a portion of their stake and go to step 6 otherwise go to step 5.
5. Validated transactions are stored in the public Blockchain.
6. Exit.

# Consensus Algorithms at a glance

## **Algorithm 3:** PBFT of a transaction

1. A transaction request of a user comes to the primary node.
2. The primary node creates a pre-prepared message assigning a unique sequence number to the requested transaction and broadcasts it to the network.
3. If nodes accept the pre-prepared message,
  - 3.1 the pre-prepared message enters into the prepared phase.
  - 3.2 Message is broadcasted.
4. Else go to step 7.
5. The primary node and other nodes verify the prepared message.
6. If at the end of primary and all secondary nodes the prepared message matches with the corresponding pre-prepared message then the transaction message is committed otherwise rejected.
7. Exit.

# Consensus Algorithms at a glance

## **Algorithm 4:** Raft

1. Leader node receives various requested transactions from clients and adds those to the log entry.
2. Transaction messages containing the transaction “r” and previous log entry “pi” are broadcasted among all followers.
3. Follower receives the transaction message and matches the “pi” with the index number of the last transaction.
4. If (the received “pi” == the index number of the last transaction)
  - 4.1. The follower adds the received “r” to its log entry
5. Else go to step6.
6. Exit.

# Consensus Algorithms at a glance

## **Algorithm 5:** Chain

1. The Block generator receives a transaction request from the client.
2. If the request is valid then store it in a temporary list.
3. After each time interval, it takes the transactions sequentially from the list and puts them in the block.
4. Broadcast the block to all “block signers”.
5. If transactions present in the block are valid then go to step6 otherwise **Exit**.
6. “Block signers” sign into them and send it back to “Block generator”.
7. If the block is signed by the majority “block signer” nodes then go to step8 otherwise Exit.
8. “Block generator” appends the block with the existing chain and proposes the newly appended block to other nodes.

# Type of Fault Tolerance

## 1. Byzantine fault tolerance (BFT)

- BFT deals with the type of faults that are arbitrary and can even be malicious.
- Byzantine Fault Tolerance (BFT) is a trait of
  - Decentralized,
  - Permissionless systems

which are capable of successfully identifying and rejecting dishonest or faulty information.

Byzantine fault tolerant systems have successfully solved the Byzantine Generals Problem and are robust against sybil attacks.

## 2. Crash fault-tolerance (CFT)

CFT covers only crash faults or, in other words, benign faults.

Sybil attack targets a network of peer-to-peer nodes by flooding the network with nodes which are all controlled by the same entity.

# How to achieve Fault Tolerance

## 1) Replication:

- Replication is a standard approach to make a system fault-tolerant.
- Replication results in a synchronized copy of data across all nodes in a network. This technique improves the fault tolerance and availability of the network.
- This means that even if some of the nodes become faulty, the overall system/network remains available due to the data being available on multiple nodes.

# How to achieve Fault Tolerance

## 1) Replication:

There are two main types of replication techniques:

- **Active replication**, which is a type where each replica becomes a copy of the original state machine replica.
- **Passive replication**, which is a type where there is only a single copy of the state machine in the system kept by the primary node, and the rest of the nodes/replicas only maintain the state.

# How to achieve Fault Tolerance

## 2) State Machine Replication (SMR):

- SMR is a de facto technique that is used to provide deterministic replication services in order to achieve fault tolerance in a distributed system.
- State machine replication was first proposed by Lamport in 1978.
- Later, in 1990, Schneider formalized the state machine replication approach.

# How to achieve Fault Tolerance

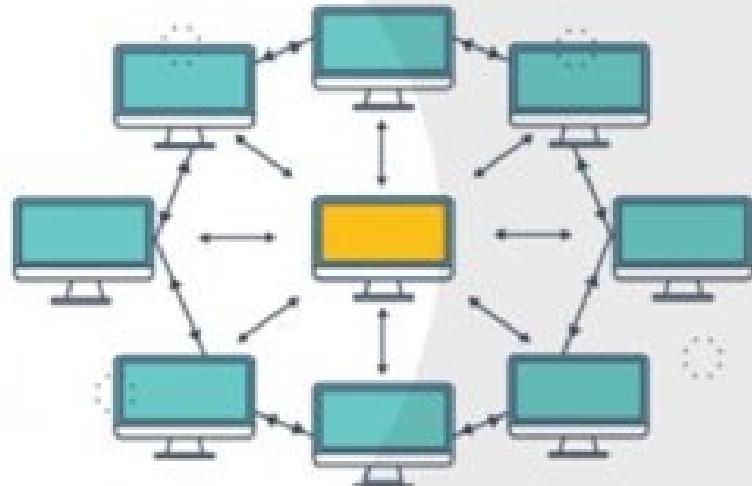
## 2) State Machine Replication (SMR):

The fundamental idea behind SMR can be summarized as follows:

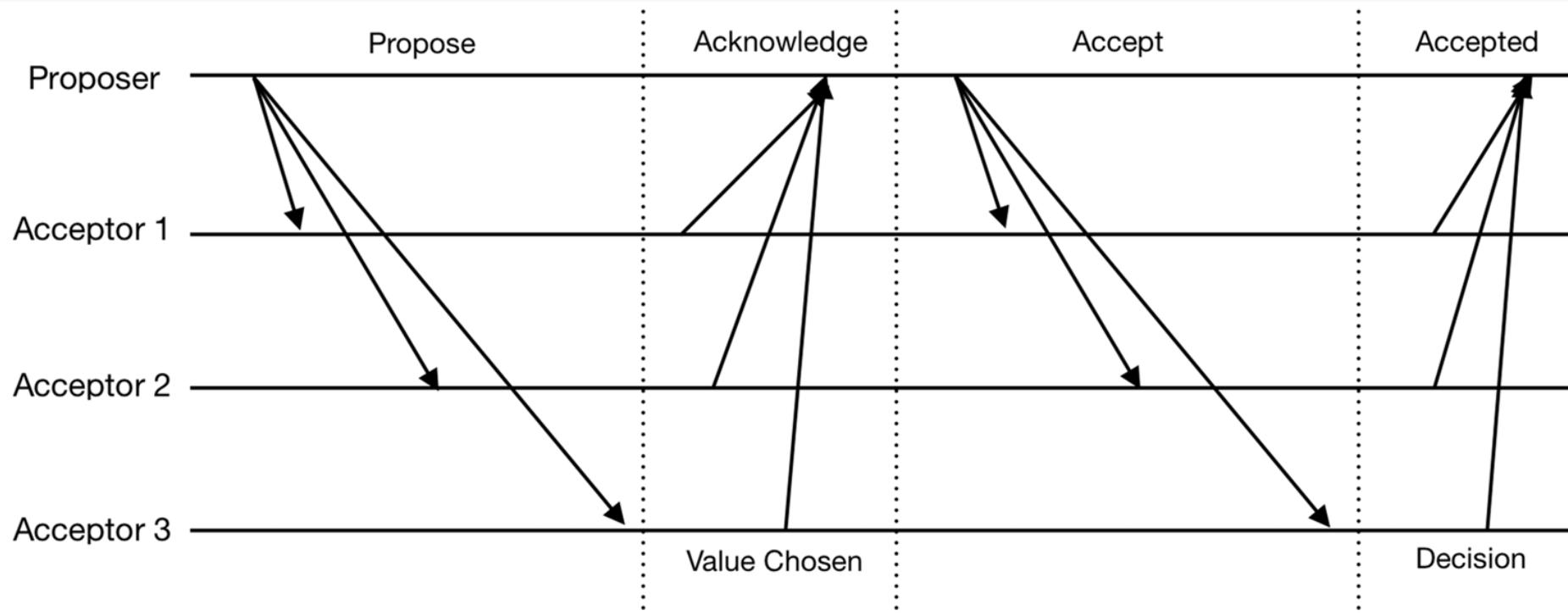
- 1. All servers always start with the same initial state.**
- 2. All servers receive requests in a totally ordered fashion (sequenced as generated from clients).**
- 3. All servers produce the same deterministic output for the same input.**

# Consensus Algorithms

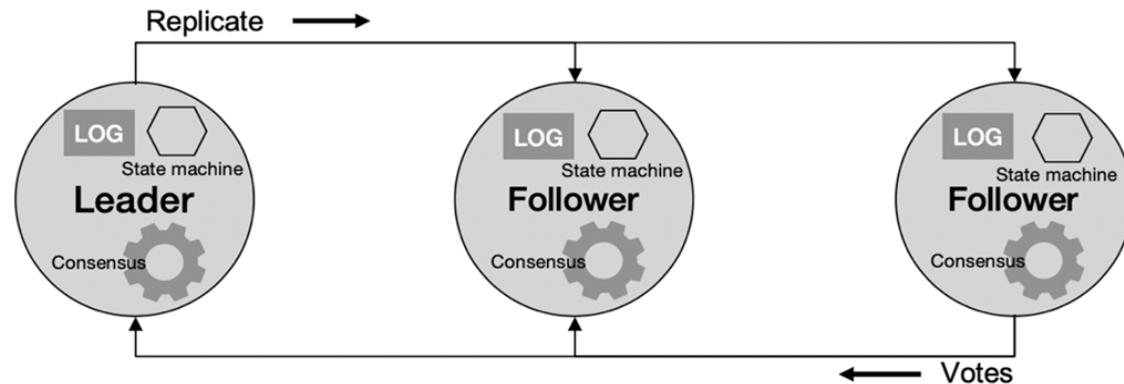
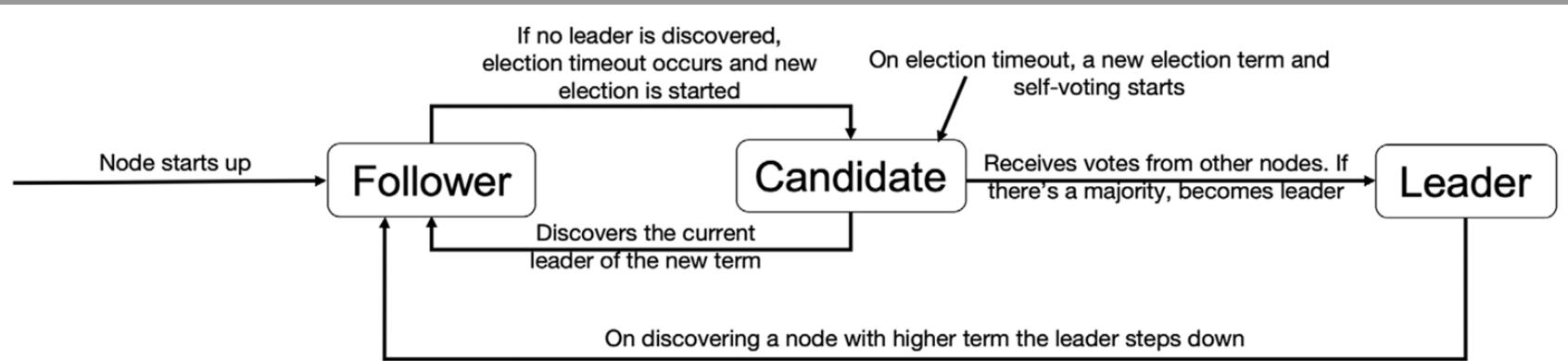
- **CFT (Crash Fault Tolerance)**
  - Paxos
  - Raft
- **BFT (Byzantine Fault Tolerance)**
  - PBFT
  - IBFT
  - Tendermint
  - HotStuff
- **Nakamoto and post-Nakamoto**
  - PoW
  - Proof of Stake (PoS)



# Paxos



# Raft - Replicated And Fault Tolerance



# Byzantine General Problem

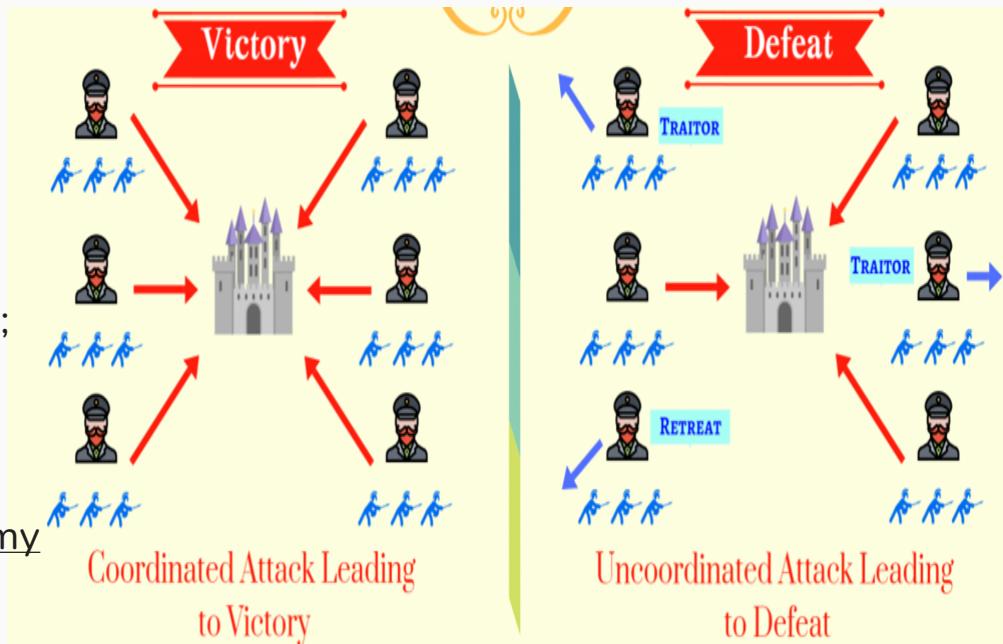
In order to achieve victory, All generals must attack the enemy castle in unison.

Consensus must be achieved by all generals in order for the Byzantine army to attack the enemy castle and win.

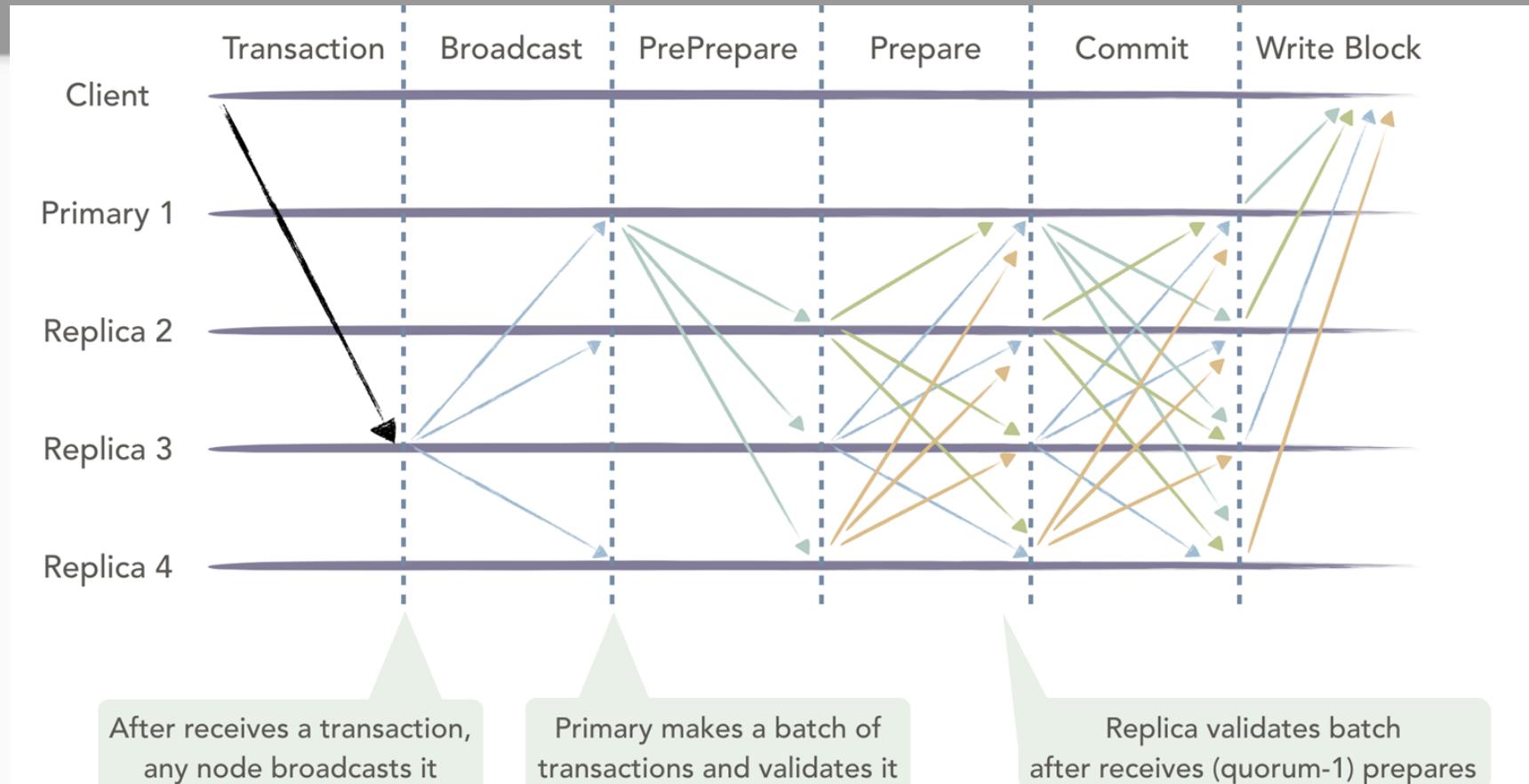
There are a few risks that could prevent victory;

- Not all generals are trustworthy;
  - Some may be traitors
- Not all messengers are trustworthy;
  - Some may be traitors
- A messenger could be caught by the enemy
  - Replaced by fake messenger

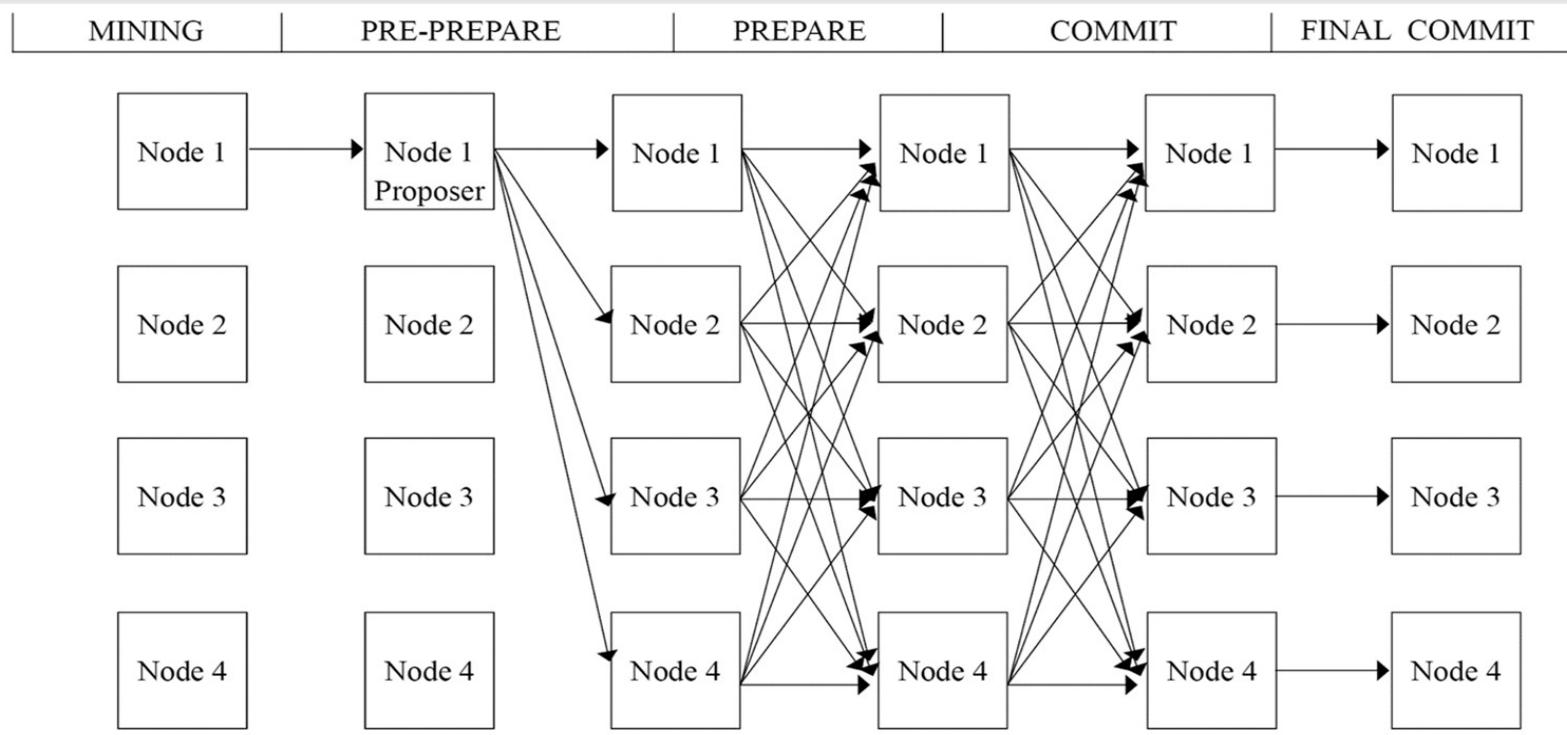
Classic problem of a distributed computing system



# PBFT - Practical Byzantine Fault Tolerance



# IBFT (Istanbul Byzantine Fault Tolerance)



Validator starts up and reads  
from the transaction pool

Creates a candidate block and  
broadcasts that as a pre-prepare message

Prepared when > 2Fprepare  
messages received

Committed if > 2Fcommit  
messages received

Insert into local blockchain  
and broadcast commit

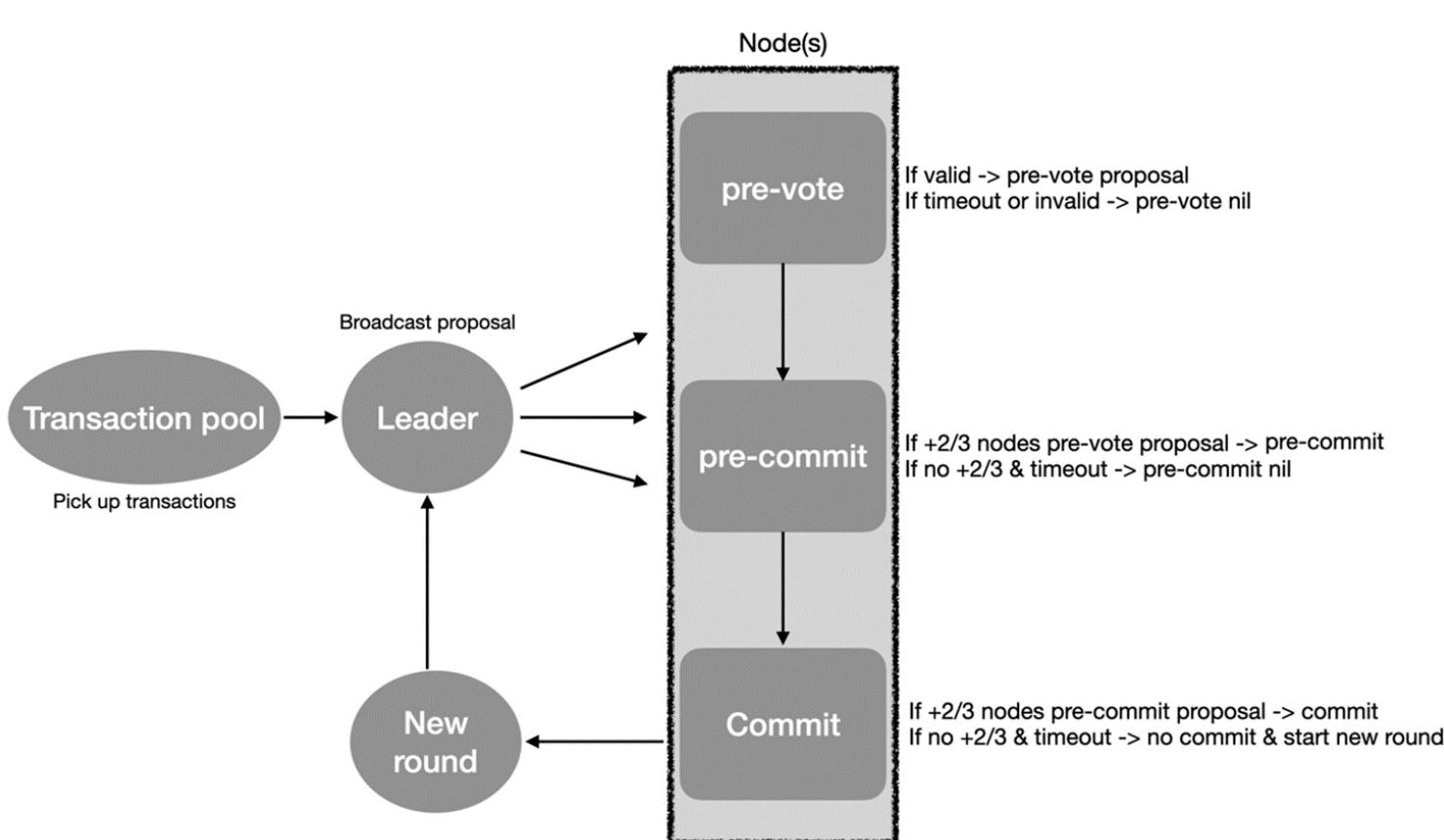
# Tindermint

New block is determined in a round.

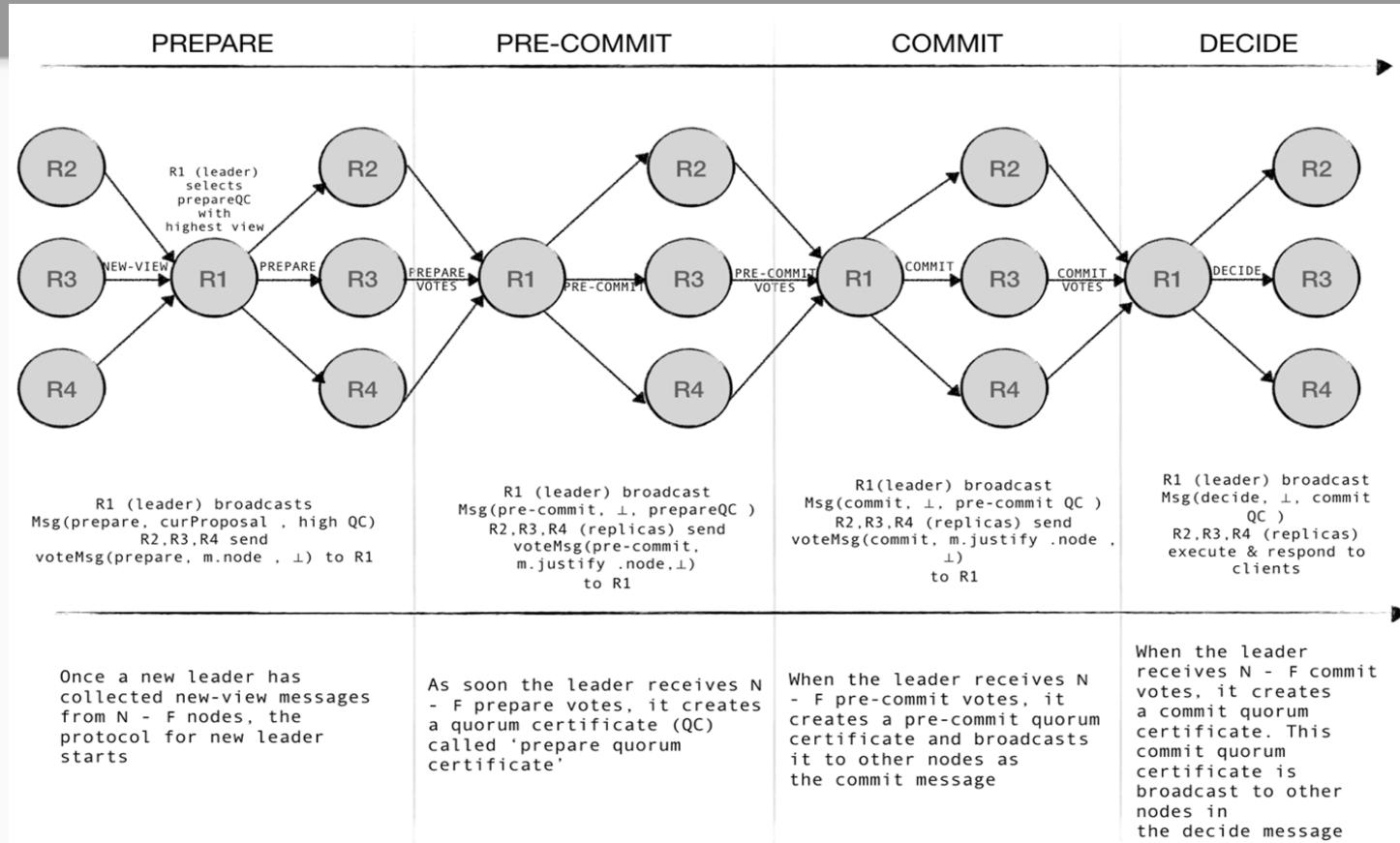
A proposer would be selected to broadcast an unconfirmed block in this round.

- 1) Prevote step.** Validators choose whether to broadcast a prevote for the proposed block.
- 2) Precommit step.** If the node has received more than  $2/3$  of prevotes on the proposed block, it broadcasts a pre commit for that block. If the node has received over  $2/3$  of precommits,it enters the commit step.
- 3) Commit step.** The node validates the block and broadcasts a commit for that block. if the node has received  $2/3$  of the commits, it accepts the block. Contrast to PBFT, nodes have to lock their coins to become validators. Once a validator is found to be dishonest, it would be punished.

# Tendermint



# Hotstuff



# POW - Consensus Mechanism

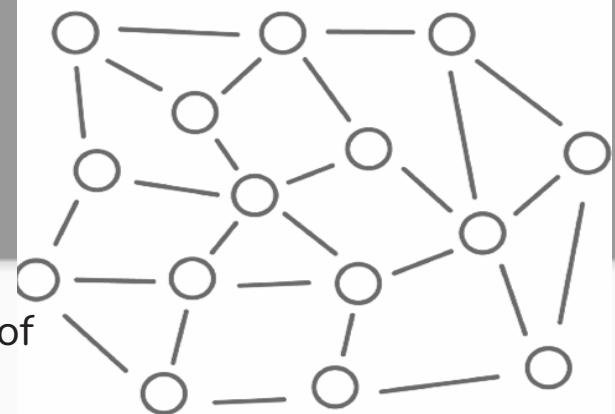
The algorithm works on the principle of providing proof of some work and the possession of some authority or tokens to win the right of proposing the next block.

PoW mechanism used in Bitcoin falls into this category, where a miner who solves the computational puzzle as proof of computational effort expended wins the right to add the next block to the blockchain.



# Proof Of Stake (POS)

- Miners in PoS have to prove the ownership of the amount of currency.
- It is believed that people with more currencies would be less likely to attack the network.
- The selection based on account balance is quite unfair because the single richest person is bound to be dominant in the network.
- Compared to PoW, PoS saves more energy and is more effective.
- Unfortunately, as the mining cost is nearly zero, attacks might come as a consequence.



## Decentralized

Users remain in control  
of their tokens



Hardware tools  
no necessary



Energy  
low consumption  
Sustainable concept

# POW VS POS

## Proof of Work

vs.

## Proof of Stake



To add each block to the chain, miners must compete to solve a difficult puzzle using their computers processing power.



In order to add a malicious block, you'd have to have a computer more powerful than 51% of the network.



The first miner to solve the puzzle is given a reward for their work.



There is no competition as the block creator is chosen by an algorithm based on the user's stake.



In order to add a malicious block, you'd have to own 51% of all the cryptocurrency on the network.



There is no reward for making a block, so the block creator takes a transaction fee.

# Classification

The consensus algorithms can be classified into two broad categories:

- **Traditional—voting-based consensus**
  - This category includes Paxos, BFT algorithms
- **Lottery-based—Nakamoto and post-Nakamoto consensus**
  - This category includes modern Proof of Work (PoW) algorithms

# Performance of Consensus Algorithms

Name→ Criteria	PoW	PoS	Hybrid	PBFT	Raft	
Energy Saving	No	Partial	No	Yes	Yes	
Miner Selection	By solving a mathematical puzzle	The node that has more stake	Based on the amount of transaction money of special transaction	By solving mathematical operations	Elected by a voting process	
Scalability	—	—	—	Weak	Weak	
51% attack	Possible	Not possible	—	Not Possible	—	
Double Spending attack	Possible	Not possible	Possible	Not Possible	—	
Speed of creating blocks	Low	Fast	Low	Low (Than the speed of creating in Raft)	Fast (Than PBFT)	
Transaction fees	Yes	Yes	Yes	No	No	
Who can participate in the consensus process	Everyone	Everyone	Everyone	The specific set of nodes	The specific set of nodes	
Block reward	Yes	Yes	Yes	No	No	
Blockchain Platform	Bitcoin	Nextcoin	Peercoin	Hyperledger Fabric	—	

# Factors in Choosing a Consensus Algorithm

It is not only use case-dependent, but some trade-offs may also have to be made to create a system that meets all the requirements without compromising the core safety and liveness properties of the system

Choosing a consensus algorithm depends on following factors.

- **Finality**
- **Speed**
- **Performance**
- **Scalability**

# Factors in Choosing a Consensus Algorithm

**Finality** refers to a concept where once a transaction has been completed, it cannot be reverted.

In other words, if a transaction has been committed to the blockchain, it won't be revoked or rolled back

# Speed - Performance - Scalability

- Performance is a significant factor that impacts the choice of consensus algorithms.
- PoW chains are slower than BFT-based chains
- If performance is a crucial requirement, then it is advisable to use voting-based algorithms for permissioned blockchains such as PBFT

# Consensus Algorithm Comparison

Consensus protocols	Advantage	Disadvantages
Pow	<ul style="list-style-type: none"><li>1.Safe and stable, high degree of freedom of nodes</li><li>2.High degree of decentralization, open node system</li></ul>	<ul style="list-style-type: none"><li>1.Weak scalability and low performance</li><li>2.Causing hardware equipment waste</li></ul>
Pos	<ul style="list-style-type: none"><li>1.Less energy</li><li>2.High degree of decentralization, open node system</li></ul>	<ul style="list-style-type: none"><li>1.Complex implementation process</li><li>2.Security breach</li></ul>
Dpos	<ul style="list-style-type: none"><li>1.Less energy</li><li>2.High performance</li><li>3.Finality</li></ul>	<ul style="list-style-type: none"><li>1. Weak degree of decentralization, closed node system</li></ul>
Pbft	<ul style="list-style-type: none"><li>1.Higher performance</li><li>2.Finality</li><li>3.High security</li></ul>	<ul style="list-style-type: none"><li>1.Weak degree of decentralization, closed node system</li><li>2.Low fault tolerance</li></ul>

# Comparison

TABLE I. COMPARISON OF THE FIVE CONSENSUS ALGORITHMS

characteristics	consensus algorithms				
	<i>PoW</i>	<i>PoS</i>	<i>DPoS</i>	<i>PBFT</i>	<i>RAFT</i>
Byzantine fault tolerance	50%	50%	50%	33%	N/A
crash fault tolerance	50%	50%	50%	33%	50%
verification speed	>100s	<100s	<100s	<10s	<10s
throughput( TPS)	<100	<1000	<1000	<2000	>10k
scalability	strong	strong	strong	weak	weak

# Introducing Bitcoin

CHAPTER # 6

# Bitcoin – definition

- Bitcoin can be defined in various ways;
  - a protocol,
  - a digital currency,
  - and a platform.
- It is a combination of a
  - peer-to-peer network,
  - protocols, and
  - software that facilitates the creation and usage of the digital currency.
- Nodes in this peer-to-peer network talk to each other using the Bitcoin protocol.



# Bitcoin an overview

- Bitcoin was introduced in 2008.
- Bitcoin has gained attention worldwide and is still the most valuable crypto currency.
- Bitcoin is built on decades of research in the field of
  - cryptography,
  - digital cash, and
  - distributed computing.
- Decentralization of currency was made possible for the first time with the invention of Bitcoin.



# Early proposals



## Early proposals to create digital cash

- In 1982, David Chaum, proposed a scheme that used blind signatures to build an untraceable digital currency. Centralized.
- In 1988, David Chaum I. proposed a refined version named eCash that not only used blind signatures, but also some private identification data to craft a message that was then sent to the bank. Identified but could not prevent double spending.
- Adam Back, current CEO of Blockstream, who is involved in Blockchain development, introduced hashcash in 1997. focus was to prevent email spam.
- In 1998, Wei Dai, a computer engineer who used to work for Microsoft, proposed b-money, which introduced the idea of using Proof of Work (PoW) to create money. A significant weakness in the system was that an adversary with higher computational power could generate unsolicited money without allowing the network to adjust to an appropriate difficulty level.



# Early proposals

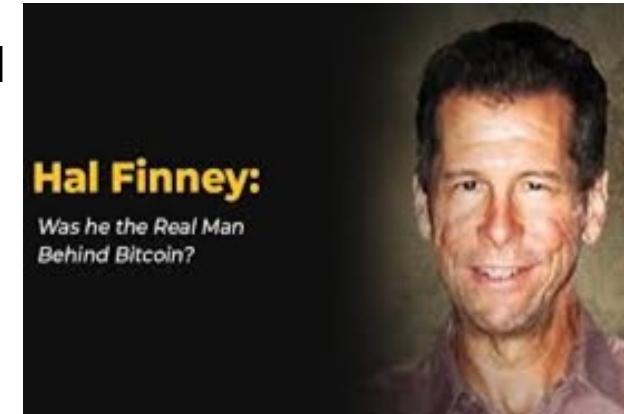
- At the same time, Nick Szabo, introduced the concept of **BitGold**, based on the **PoW mechanism** but had the same problems as b-money, with the exception that the **network difficulty level**
- Berkley, introduced an e-cash scheme in a research paper named *Auditable, Anonymous Electronic Cash* in 1999. This scheme, for the first time, used **Merkle trees** to represent coins and **zero-knowledge proofs** to prove the possession of coins. was adjustable.



Nick Szabo

# Early proposals

- RPOW (Reusable Proof of Work) was introduced in 2004 by Hal Finney, a computer scientist, developer, and the z.
- It used the hashcash scheme by Adam Back as a proof of computational resources spent to create the money.
- This was also a central system that kept a central database to keep track of all used PoW tokens.
- This was an online system that used remote attestation, made possible by a trusted computing platform (TPM hardware).



# Beginning of Bitcoin

## Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto  
satoshi@gmx.com  
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network.

- In 2008, Bitcoin was introduced in a paper called *Bitcoin: A Peer-to-Peer Electronic Cash System* by Satoshi Nakamoto.

- The first key idea introduced in the paper was of a purely peer-to-peer electronic cash that does not need an intermediary bank to transfer payments between peers.
- Bitcoin is built on decades of research. Various ideas and techniques from cryptography and distributed computing such as Merkle trees, hash functions, and digital signatures were used to design Bitcoin.

# Beginning of Bitcoin

Bitcoin solves a number of historically difficult problems related to electronic cash and distributed systems, including:

- The Byzantine generals problem
- The double-spending problem
- Sybil attacks

Bitcoin is an elegant solution to the **Byzantine generals problem** and the **double-spending problem**.

# Growth of Bitcoin

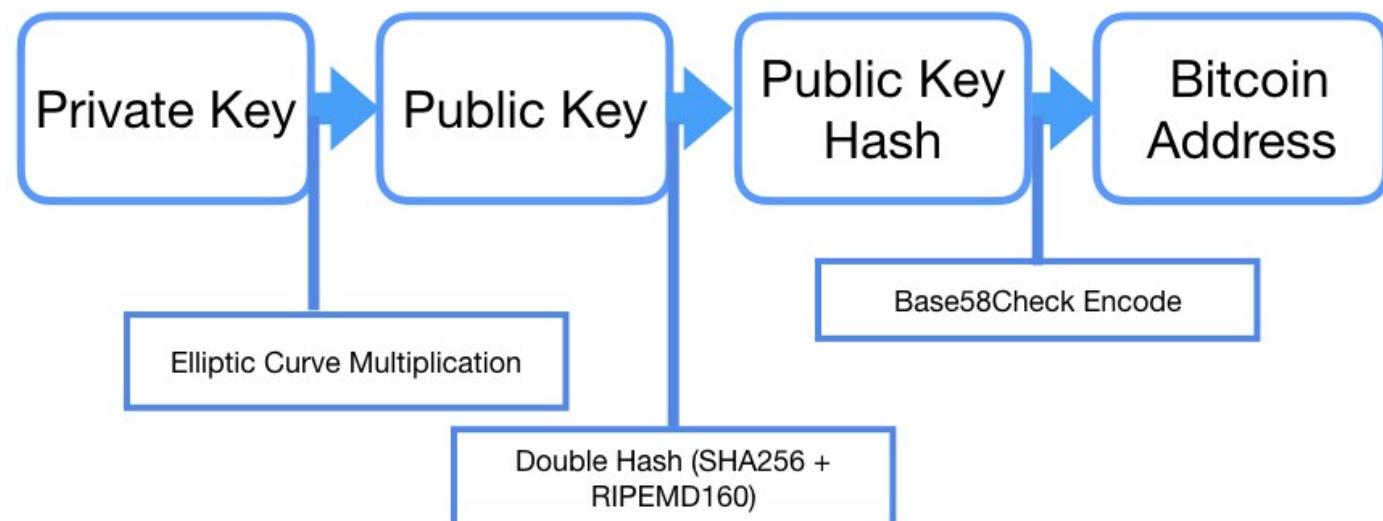
- The growth of Bitcoin is also due to the so-called **network effect**. Also called **demand-side economies of scale**, it is a concept that means that the more users use the network, the more valuable it becomes.
- Over time, an exponential increase has been seen in Bitcoin network growth. This increase in the number of users is mostly driven by financial incentives.
- Also, the scarcity of Bitcoin and its built-in inflation control mechanism gives it value, as there are only 21 million Bitcoins that can ever be mined.



# Cryptographic Keys

On the Bitcoin network, possession of Bitcoins and the transfer of value via transactions are reliant upon:

- Private keys
- Public keys
- Addresses



# Private keys in blockchain

**1. Private keys are required to be kept safe and normally reside only on the owner's side. Private keys are used to digitally sign the transactions, proving ownership of the bitcoins.**

**2. Private keys are fundamentally 256-bit numbers randomly chosen in the range specified by the SECP256K1 ECDSA curve recommendation.**

**3. Any randomly chosen 256-bit number from 0x1 to 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140 is a valid private key.**

**Hex:** DD5113FEDED638E5500E65779613BDD3BDBEB8EB5D86CDD3370E629B02E92CD

**Base64:** 3VET/t7WOOVQDmV3lhO9073b64612GzdM3DmKbAuks0=

**WIF:** 5KVkpWGfDQGJAUEEDUFbrFywNPjmXy5kBBmRzzBDf4JkgFXqXTa

**Binary:**

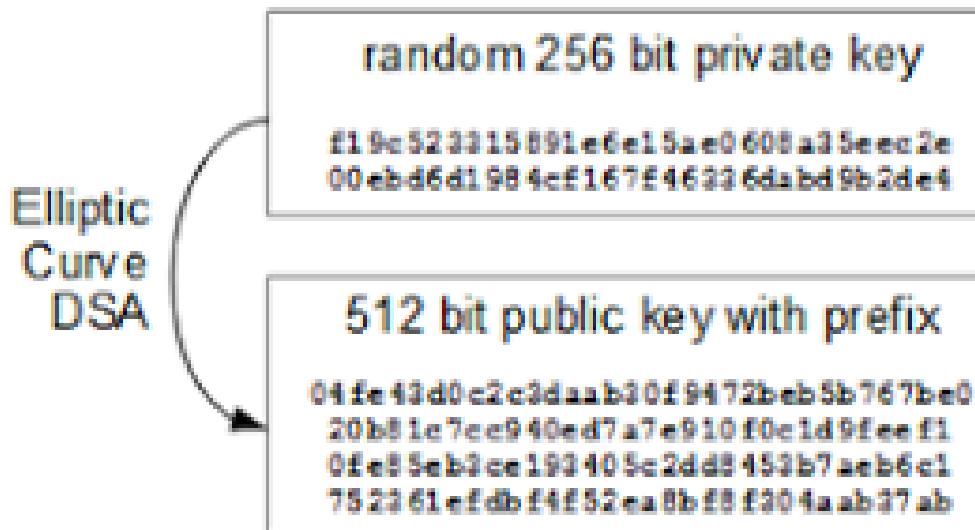
1101110101010001000100111111101101110110101100011100011100101010100  
0000001110011001010111011100101100001001110111011101001110111011101  
101111010111000111010110101110110000110110011011101001100110111000011  
1001100010100110110000001011101001001011001101

Bitcoin private key in various formats, including WIF.

**1. Private keys are usually encoded using Wallet Import Format (WIF) in order to make them easier to copy and use. It is a way to represent the fullsize private key in a different format.**

# Public keys in blockchain

- Public keys exist on the blockchain and all network participants can see them.
- Bitcoin uses ECC based on the SECP256K1 standard.
- More specifically, it makes use of an **Elliptic Curve Digital Signature Algorithm (ECDSA)** to ensure that funds remain secure and can only be spent by the legitimate owner.
- Once a transaction signed with the private key is broadcast on the Bitcoin network, public keys are used by the nodes to verify that the transaction has indeed been signed with the corresponding private key.
- This process of verification proves the ownership of the bitcoin.

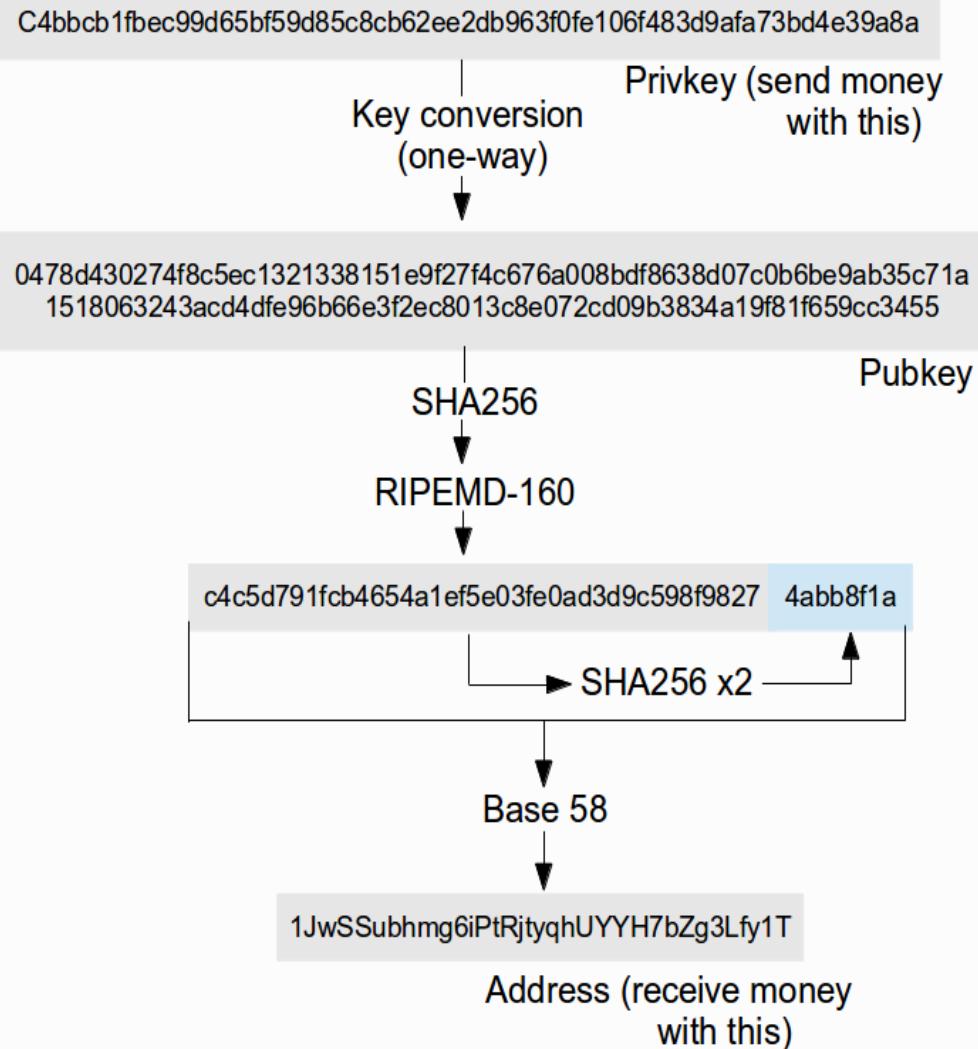


# Address in bitcoin

1. A Bitcoin address is created by taking the corresponding public key of a private key and hashing it twice, first with the SHA256 algorithm and then with RIPEMD160.

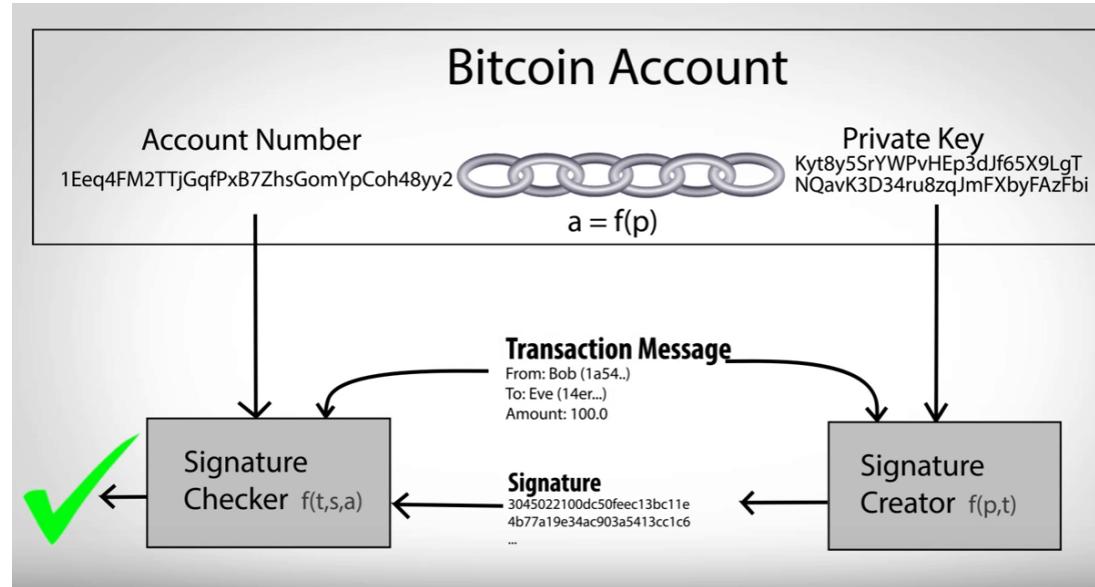
2. The resultant 160-bit hash is then prefixed with a version number and finally encoded with a Base58Check encoding scheme.

3. The Bitcoin addresses are 26-35 characters long and begin with digits 1 or 3.



# Transactions

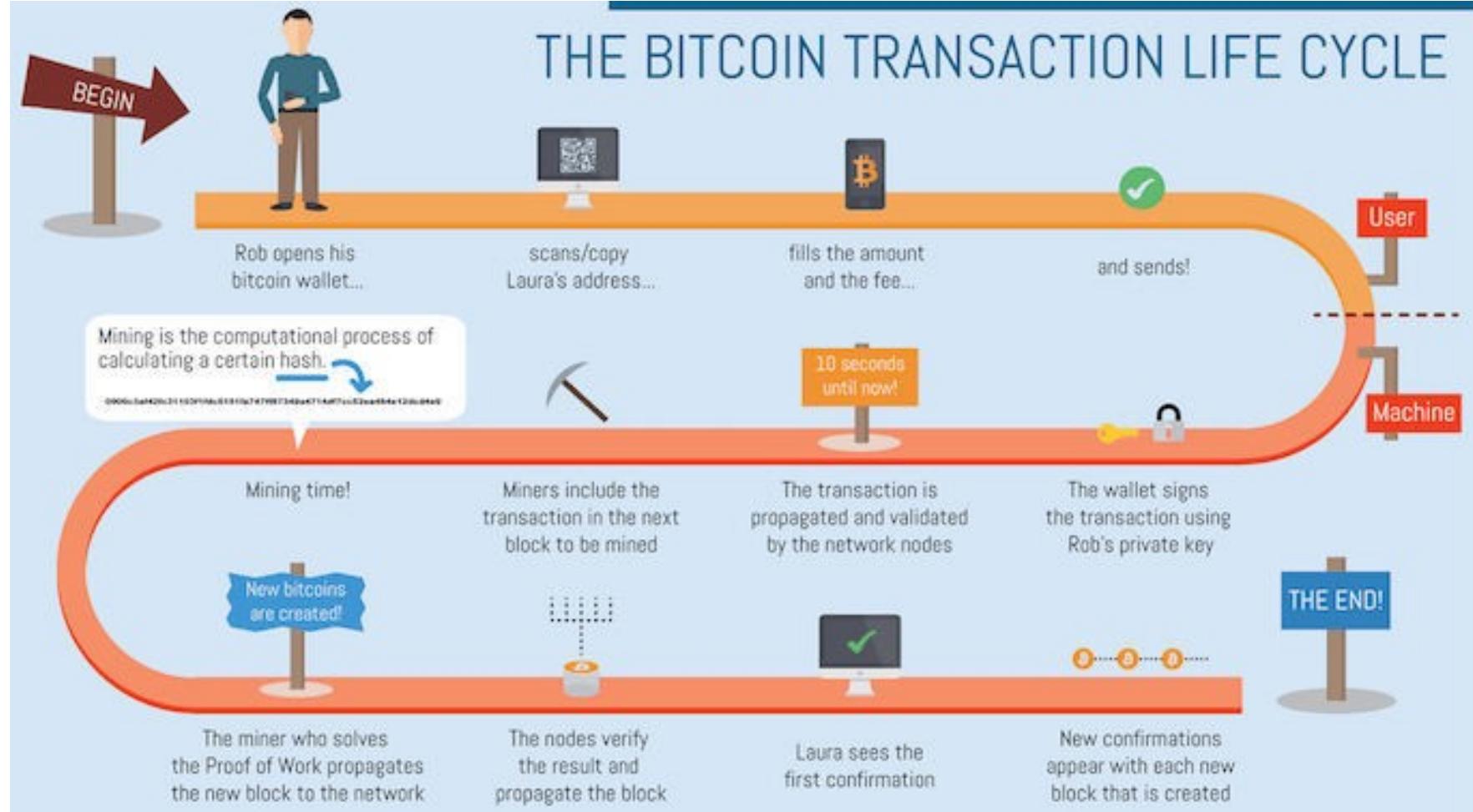
- Transactions are at the core of the Bitcoin ecosystem.
- Transactions can be as simple as just sending some bitcoins to a Bitcoin address, or can be quite complex, depending on the requirements.
- Each transaction is composed of at least one input and output. Inputs can be thought of as coins being spent that have been created in a previous transaction, and outputs as coins being created.
- If a transaction is minting new coins, then there is no input, and therefore no signature is needed.



# Transactions

- If a transaction should send coins to some other user (a Bitcoin address), then it needs to be signed by the sender with their private key. In this case, a reference is also required to the previous transaction to show the origin of the coins.
- Coins are unspent transaction outputs represented in Satoshis.
- Transactions are not encrypted and are publicly visible on the blockchain.
- Blocks are made up of transactions, and these can be viewed using any online blockchain explorer.

# THE BITCOIN TRANSACTION LIFE CYCLE



# Bitcoin Transaction Life Cycle

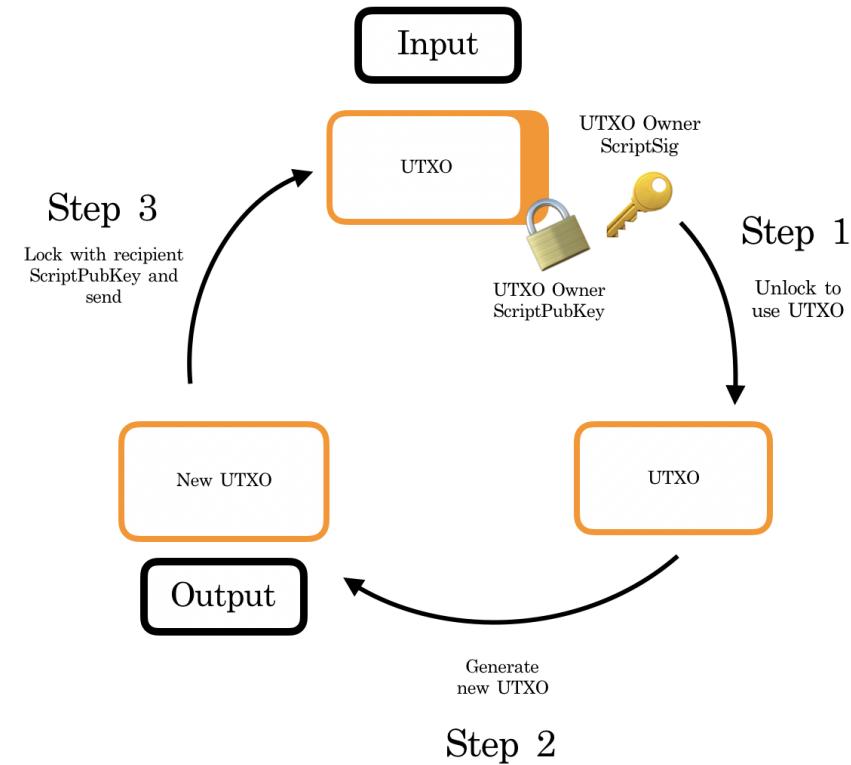
1. A user/sender sends a transaction using wallet software or some other interface.
1. The wallet software signs the transaction using the sender's private key.
1. The transaction is broadcasted to the Bitcoin network using a flooding algorithm.
1. Mining nodes (miners) who are listening for the transactions verify and include this transaction in the next block to be mined. Just before the transactions are placed in the block, they are placed in a special memory buffer called the transaction pool.
1. Next, the mining starts, which is the process through which the blockchain is secured and new coins are generated as a reward for the miners who spend appropriate computational resources. Once a miner solves the PoW problem, it broadcasts the newly mined block to the network. The nodes verify the block and propagate the block further, and confirmations start to generate.
1. Finally, the confirmations start to appear in the receiver's wallet and after approximately three confirmations, the transaction is considered finalized and confirmed. However, three to six is just a recommended number; the transaction can be considered final even after the first confirmation. The key idea behind waiting for six confirmations is that the probability of double spending is virtually eliminated after six confirmations.

# Transactions

Step 1: Owner unlocks UTXO with their own private keys

Step 2: Generate new UTXOs of desired value for use in transactions

Step 3: Lock these new UTXOs and send them to



# Example

Initial Balance =  $3 + 1 + 1.5 + 4.5 = 10$  BTC  
Final Balance =  $3 + 1.5 + 0.5 = 5$  BTC

Cake

Initial Balance = 1.5 BTC  
Final Balance =  $1.5 + 5 = 6.5$  BTC

Cake Buyer



Cake Seller



Price: 5 BTC

UTXOs

UTXO\_2  
3 BTC

UTXO\_3  
1 BTC

UTXO\_4  
1.5 BTC

UTXO\_5  
4.5 BTC

UTXO\_7  
0.5 BTC

Input

UTXO\_3  
1 BTC

UTXO\_5  
4.5 BTC

Output

UTXO\_6  
5 BTC

UTXO\_7  
0.5 BTC

Transfer

Spent

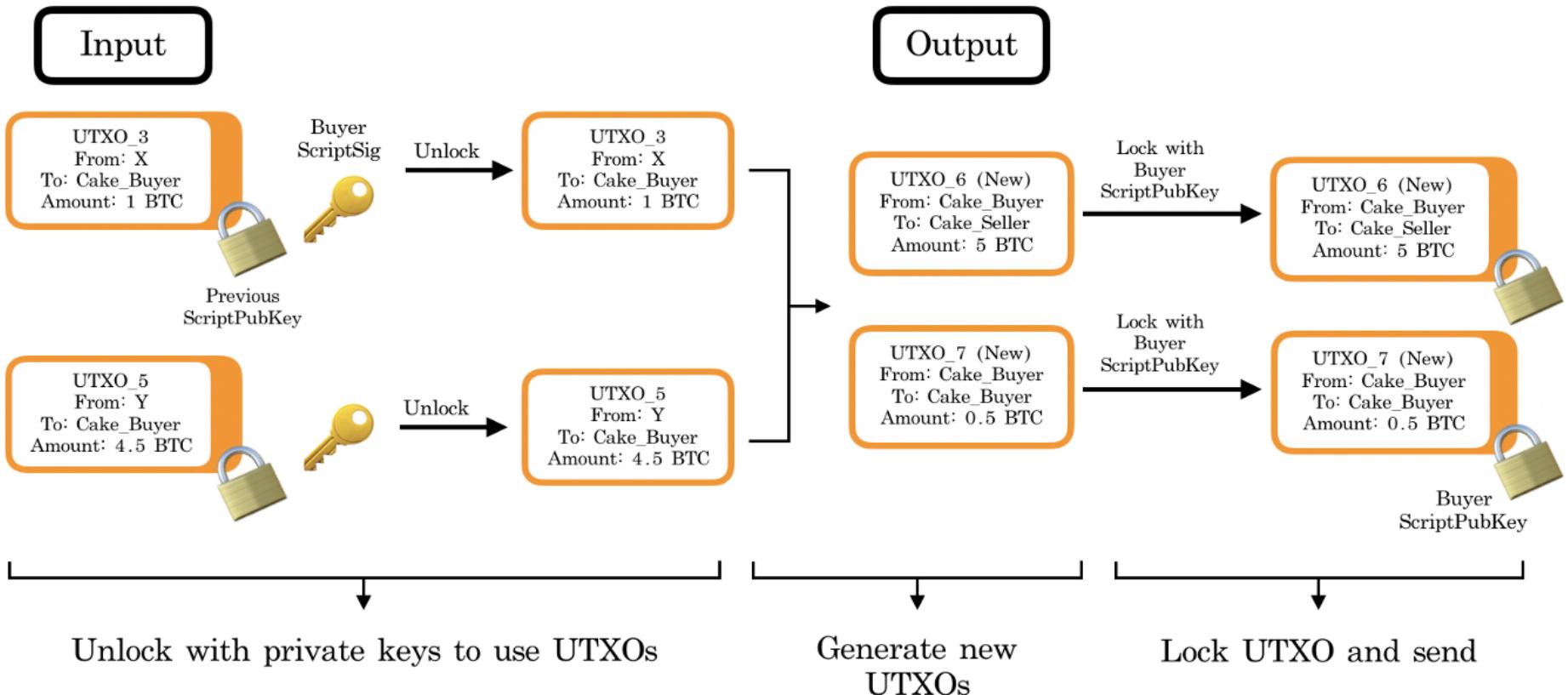
Spent

New UTXO

Payment for cake

New UTXO

Remaining amount  
sent back to herself  
in separate UTXO



# Transaction pool

- Also known as memory pools, these pools are basically created in local memory (computer RAM) by nodes (Bitcoin clients) in order to maintain a temporary list of transactions that have not yet been added to a block.
- Miners pick up transactions from these memory pools to create candidate blocks.
- Miners select transactions from the pool after they pass the verification and validity checks.
- The selection of which transactions to choose is based on the fee and their place in the order of transactions in the pool.
- Miners prefer to pick up transactions with higher fees.

# Transaction fees

- Transaction fees are charged by the miners.
- The fee charged is dependent upon the size and weight of the transaction.
- Transaction fees are calculated by subtracting the sum of the inputs from the sum of the outputs.

# Transaction Data Structure

Field	Size	Description
Version number	4 bytes	Specifies the rules to be used by the miners and nodes for transaction processing. There are two versions of transactions, that is, 1 and 2.
Input counter	1-9 bytes	The number (a positive integer) of inputs included in the transaction.
List of inputs	Variable	Each input is composed of several fields. These include: The previous transaction hash The index of the previous transaction Transaction script length Transaction script Sequence number The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs. In summary, this field describes which Bitcoins are going to be spent.
Output counter	1-9 bytes	A positive integer representing the number of outputs.
List of outputs	Variable	Outputs included in transaction. This field depicts the target recipient(s) of Bitcoins.



Data

<b>Filename:</b>	Tadzik.jpg
<b>Author:</b>	Piotr Kononow
<b>Date:</b>	August 15, 2016 6:40:10PM
	5,312 × 2,988 JPEG
<b>File:</b>	15.9 megapixels
	3,393,448 bytes
	(3.2 megabytes)
<b>Camera:</b>	Samsung SM-G920F
	4.3 mm
<b>Lens:</b>	Max aperture f/1.9 (shot wide open)
	Auto exposure
	Program AE
<b>Exposure:</b>	1/402 sec
	f/1.9
	ISO 40
<b>Flash:</b>	none



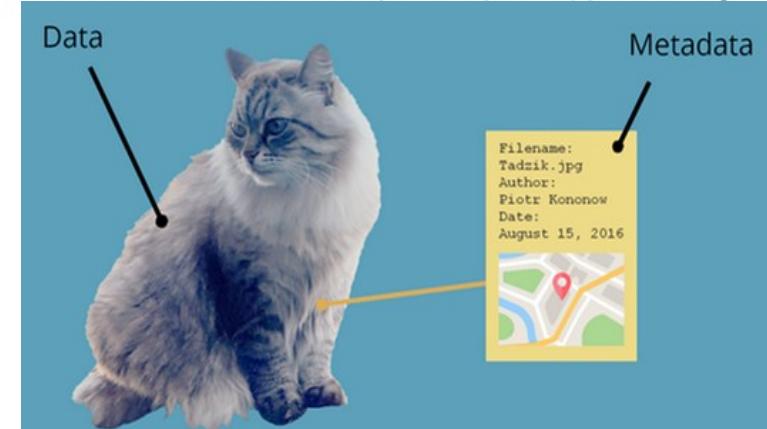
Metadata

# Metadata

- This part of the transaction contains values such as the size of the transaction, the number of inputs and outputs, the hash of the transaction, and a lock\_time field.
- Every transaction has a prefix specifying the version number.
- These fields are shown in the preceding example as lock\_time , size , and version .

metadata

```
{  
  "hash": "5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",  
  "ver": 1,  
  "vin_sz": 2,  
  "vout_sz": 1,  
  "lock_time": 0,  
  "size": 404,  
  "in": [
```



# Inputs

- Generally, each input spends a previous output.
- Each output is considered an **Unspent Transaction Output (UTXO)** until an input consumes it.
- A UTXO is an unspent transaction output that can be spent as an input to a new transaction.
- The transaction input data structure is explained in the following table:

Each input spends a previous output

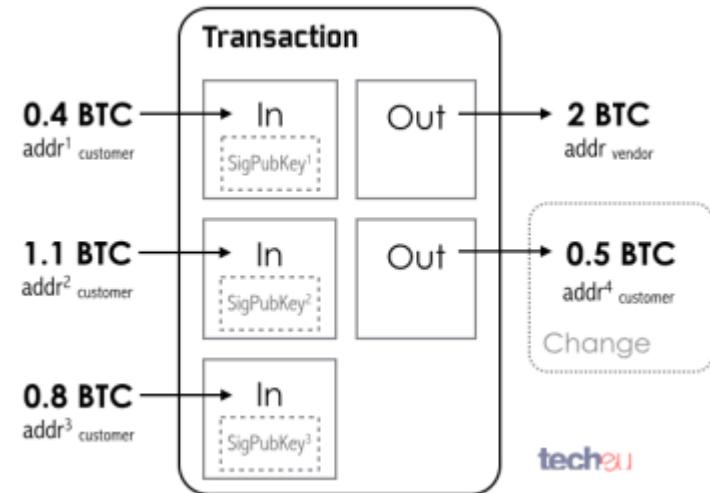
The Main Parts Of  
Transaction 0

Version	Inputs	Outputs	Locktime
---------	--------	---------	----------

The Main Parts Of  
Transaction 1

Version	Inputs	Outputs	Locktime
---------	--------	---------	----------

Each output waits as an Unspent TX Output (UTXO) until a later input spends it



# Input Data Structure

input(s)

```
{  
  "prev_out":{  
    "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",  
    "n":0  
  },  
  "scriptSig":"30440..."  
},  
{  
  "prev_out":{  
    "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",  
    "n":0  

```

Field	Size	Description
Transaction hash	32	The hash of the previous transaction with UTXO
Output index	4	This is the previous transaction's output index, such as UTXO to be spent
Script length	1-9	Size of the unlocking script
Unlocking script	vari	Input script ( ScriptSig ), which satisfies the requirements of locking script
Sequence number	4	Usually disabled or contains lock time – disabled is represented by 0xFFFFFFFF

# Coinbase Transactions

- A coinbase transaction or generation transaction is always created by a miner and is the first transaction in a block. It is used to create new coins.
- It includes a special field, also called the coinbase, which acts as an input to the coinbase transaction.
- This transaction also allows up to 100 bytes of arbitrary data storage.
- The structure of coinbase transaction is shown in [Next Slide](#)



# Coinbase Transaction - Structure

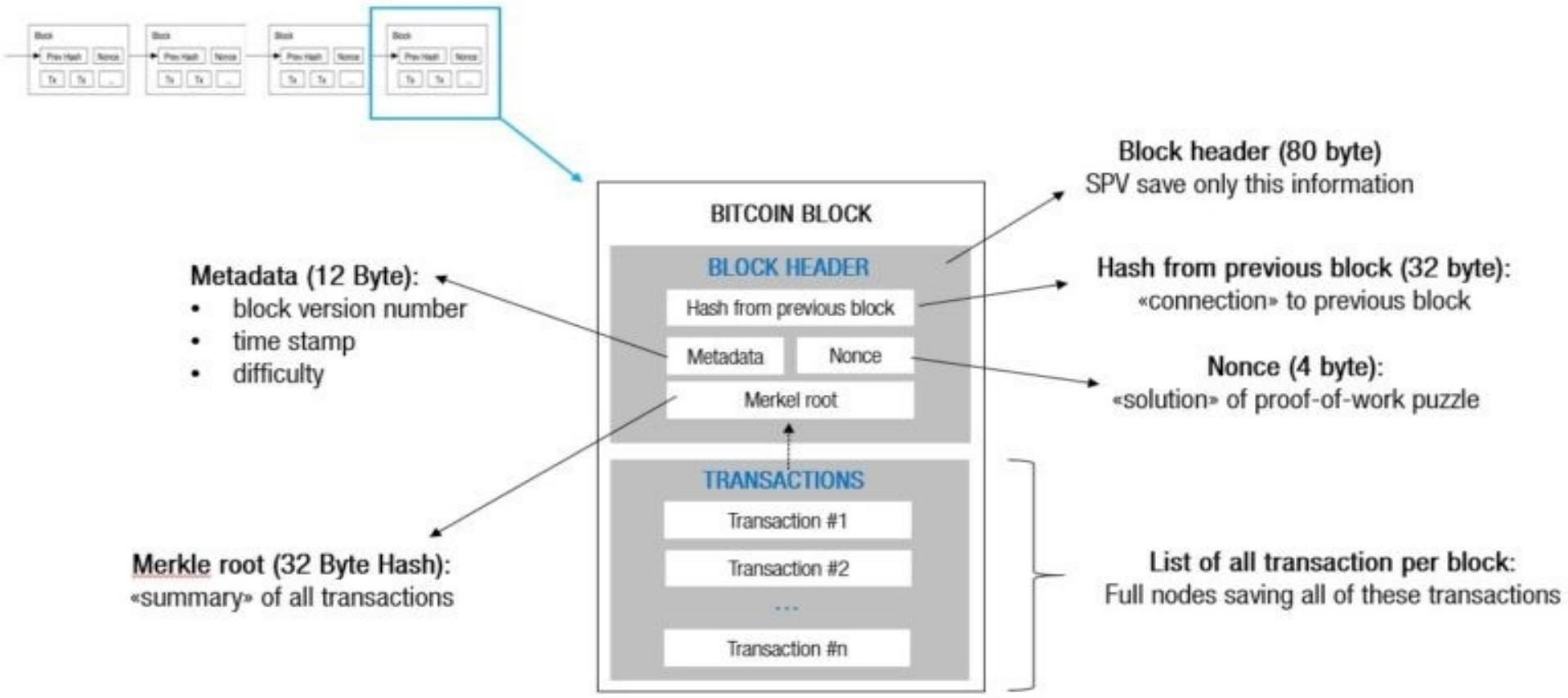
Field	Size	Description
Transaction hash	32 bytes	Set to all zeroes as no hash reference is used
Output index	4 bytes	Set to 0xFFFFFFFF
Coinbase data length	1-9 bytes	2-100 bytes
Data	Variable	Any data
Sequence number	4 bytes	Set to 0xFFFFFFFF

# Transaction Validation

This verification process is performed by Bitcoin nodes. There are three main things that nodes check when verifying a transaction:

1. That transaction inputs are previously unspent. This validation step prevents double spending by verifying that the transaction inputs have not already been spent by someone else.
2. That the sum of the transaction outputs is not more than the total sum of the transaction inputs. However, both input and output sums can be the same, or the sum of the input (total value) could be more than the total value of the outputs. This check ensures that no new bitcoins are created out of thin air.
3. That the digital signatures are valid, which ensures that the script is valid.

# Bitcoin Block



# Data Structure of Block

Field	Size	Description
Block size	4 bytes	The size of the block.
Block header	80 bytes	This includes fields from the block header described in the next section.
Transaction counter	Variable	Field contains the total number of transactions in the block, including coinbase transaction. Size ranges from 1-9 bytes.
Transactions	Variable	All transactions in the block.
Version	4 bytes	The block version number that dictates the block validation rules to follow.

# Data structure of Block-2

Field	Size	Description
Previous block's Header Hash	32 bytes	This is a double SHA256 hash of the previous block's header
Merkle root hash	32 bytes	This is a double SHA256 hash of the Merkle tree of all transactions included in the block.
Timestamp	4 bytes	This field contains the approximate creation time of the block in Unix epoch time format. More precisely, this is the time when the miner started hashing the header (the time from the miner's location).
Difficulty target	4 Bytes	This is the current difficulty target of the network/block.
Nonce	4 bytes	This is an arbitrary number that miners change repeatedly to produce a hash that is lower than the difficulty target.

# Network Difficulty

- Network difficulty refers to a measure of how difficult it is to find a new block, or in other words, how difficult it is to find a hash below the given target.
- New blocks are added to the blockchain approximately every 10 minutes, and the network difficulty is adjusted dynamically every 2,016 blocks in order to maintain a steady addition of new blocks to the network.
- Network difficulty is calculated using the following equation:

New Difficulty = Old Difficulty \* (Actual Time Of Last 2016 Blocks / 20160 Minutes)

# Mining

- Mining is a process by which new blocks are added to the Blockchain.
- Blocks contain transactions that are validated via the mining process by mining nodes.
- Blocks, once mined and verified, are added to the blockchain.
- This process is resource intensive due to the requirements of PoW, where miners compete to find a number less than the difficulty target of the network.

It goes like this

1. Verify if transactions are valid



2. Bundle transactions in a block
3. Select the header of the most recent block and insert it into the new block as a hash



4. Solve the Proof of Work problem
5. When the solution is found, the new block is added to the local blockchain and propagated to the network



# Tasks Of The Miners

## 1. Synching up with the network:

- Once a new node joins the Bitcoin network, it downloads the blockchain by requesting historical blocks from other nodes.

## 2. Transaction validation:

- Transactions broadcast on the network are validated by full nodes by verifying and validating signatures and outputs.

## 3. Block validation:

- Miners and full nodes can start validating blocks received by them by evaluating them against certain rules.
- This includes the verification of each transaction in the block along with verification of the nonce value.

# Tasks Of The Miners

## 4. Create a new block:

- Miners propose a new block by combining transactions broadcast on the network after validating them.

## 5. Perform PoW:

- This task is the core of the mining process and this is where miners find a valid block by solving a computational puzzle.
- Block header contains 32-bit nonce field and miners are required to repeatedly vary the nonce until resultant hash is less than a predetermined target.

## 6. Fetch reward:

- Once a node solves the hash puzzle (PoW), it immediately broadcasts the results, and other nodes verify it and accept the block.
- There is a slight chance that the newly minted block will not be accepted by other miners on the network due to a clash with another block found at roughly the same time, but once accepted, the miner is rewarded 6.25 bitcoins and transaction fees.

# Mining Rewards

- Miners are rewarded with new coins if and when they discover new blocks by solving the PoW.
- Miners are paid transaction fees in return, for the transactions in their proposed blocks.
- New blocks are created at an approximate fixed rate of every 10 minutes.
- The rate of creation of new bitcoins decreases by 50% every 210,000 blocks, which is roughly every 4 years. When Bitcoin started in 2009, the mining reward used to be 50 bitcoins. After every 210,000 blocks, the block reward halves. In November 2012 it halved down to 25 bitcoins, since May 2020, it is 6.25 bitcoins per block.
- This mechanism is hardcoded in Bitcoin to regulate & control inflation and limit supply of bitcoins.



# The Mining Algorithm

The mining algorithm consists of the following steps:

1. The previous block's header is retrieved from the Bitcoin network.
2. Assemble a set of transactions broadcast on the network into a block to be proposed.
3. Compute the double hash of the previous block's header, combined with a nonce and the newly proposed block, using the SHA256 algorithm.
4. Check if the resulting hash is lower than the current difficulty level (the target), then PoW is solved. As a result of successful PoW, the discovered block is broadcasted to the network and miners fetch the reward.
5. If the resultant hash is not less than the current difficulty level (target), then repeat the process after incrementing the nonce.

## How to win for a given block

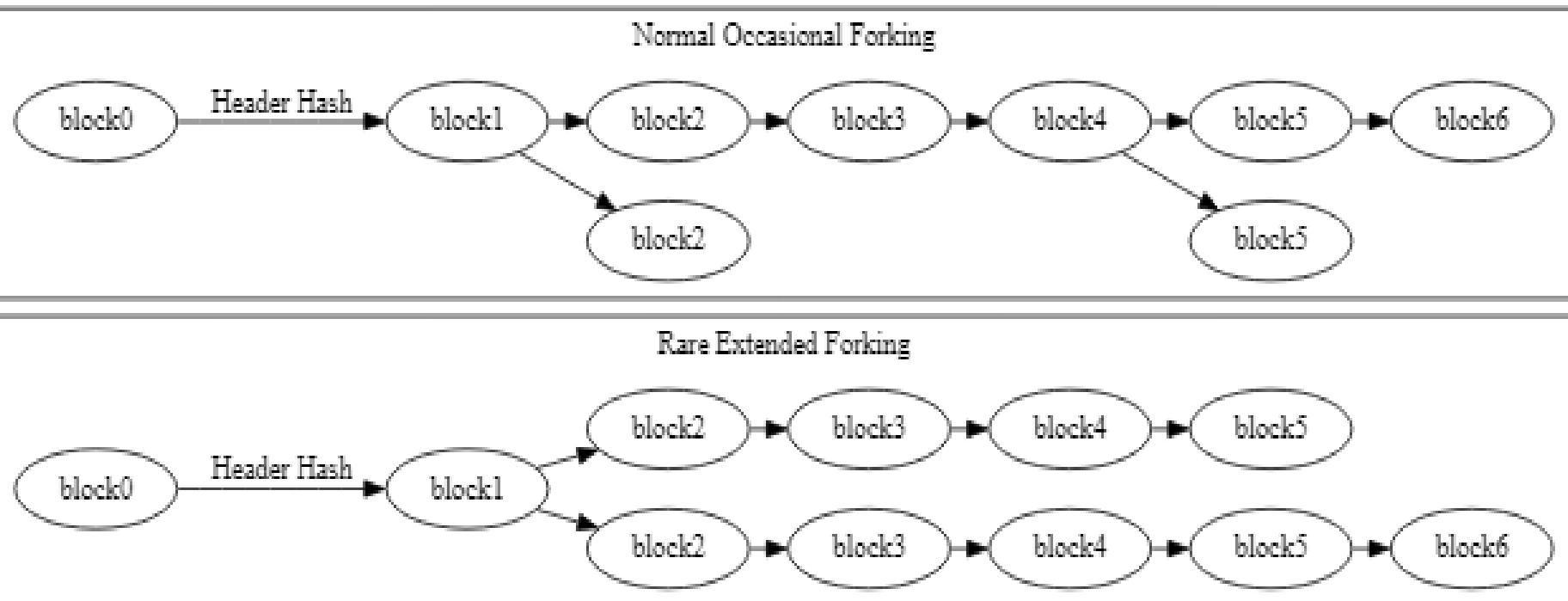
Target	Disqualified	Disqualified	Viable
00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000
<u>0</u> 57FCC70	<u>3</u> 57FCC70	0 <u>D</u> FCC70	0 <u>4</u> 7FCC70
8CF0130D	8CF0130D	8CF0130D	8CF0130D
95E27C58	95E27C58	95E27C58	95E27C58
19203E9F	19203E9F	19203E9F	19203E9F
967AC56E	967AC56E	967AC56E	967AC56E
4DF598EE	4DF598EE	4DF598EE	4DF598EE

Has only 16 zeros.  
(the target has 17).  
So all right answers  
need to have at least  
17 zeros.

18<sup>th</sup> digit it's a "d,"  
which in hexadecimal  
is 13. This is larger  
than the 18<sup>th</sup> digit of  
the target — "5."

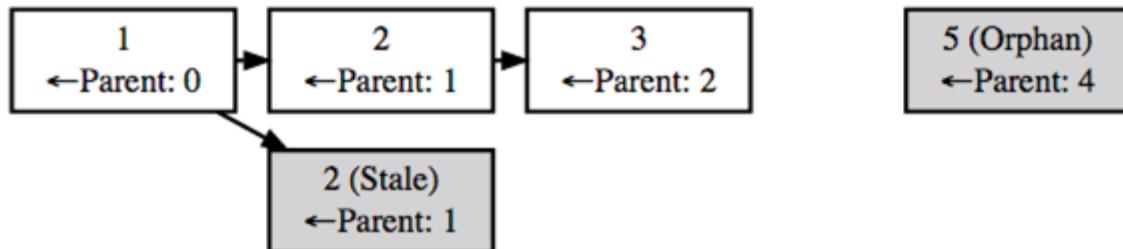
Smaller than the  
target hash.  
Get there before any  
other miner and get  
paid 12.5 BTC.

# Block Height and Forking



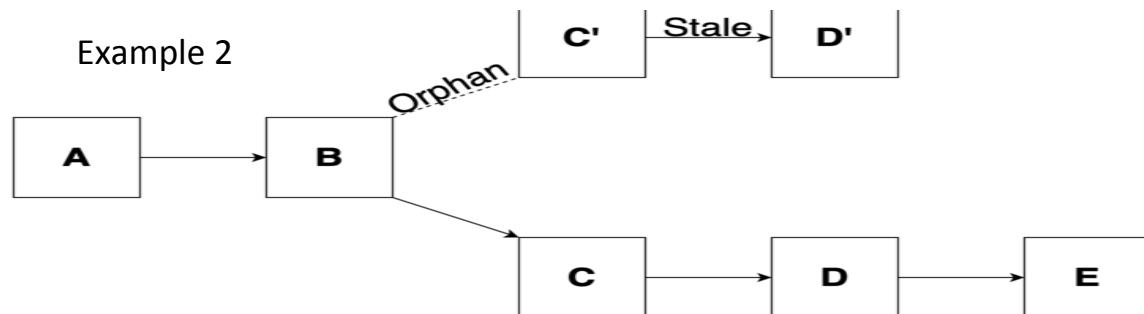
# Stale & Orphan Blocks

Orphan blocks have no known parent, so they can't be validated



Stale blocks are valid but not part of the best block chain

Example 2



Orphan blocks Their parent blocks are unknown, they cannot be validated. This problem occurs when two or more miners discover a block at almost the same time. These are valid blocks but now they are no longer part of the main chain.

# The Hash Rate

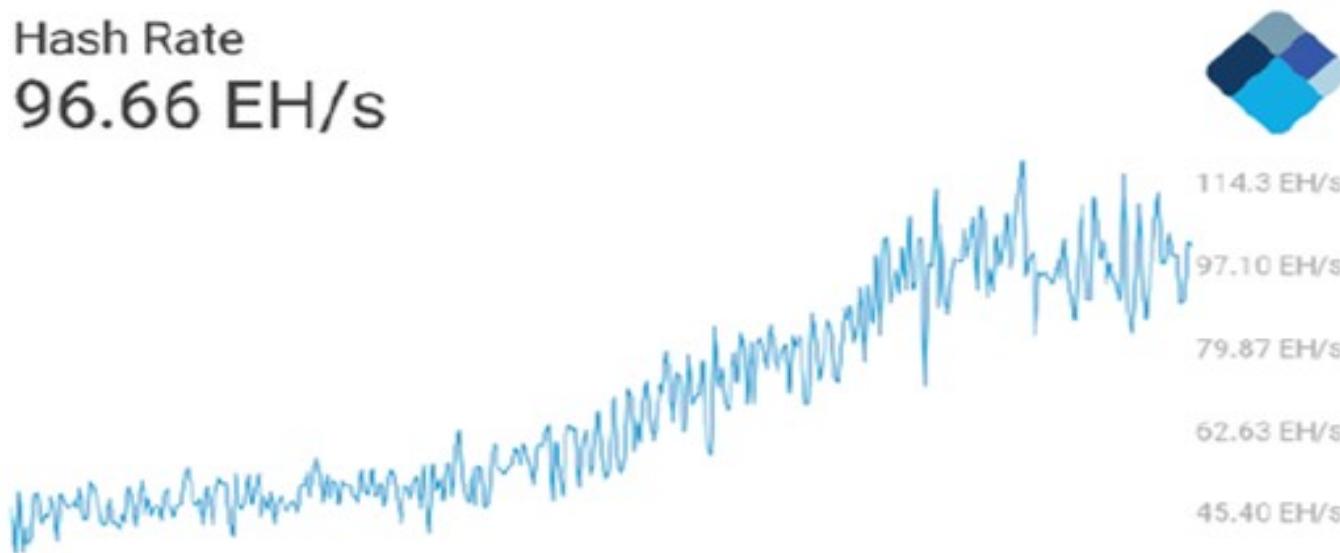
- Rate of hash calculation per second.
- This is the speed at which miners in the Bitcoin network are calculating hashes to find a block.
- In the early days of Bitcoin, it used to be quite small, as CPUs were used, which are relatively weak in mining terms.
- However, with dedicated mining pools and **Application Specific Integrated Circuits (ASICs)** now, this has gone up exponentially in the last few years.
- This has resulted in increased difficulty in the Bitcoin network.

# Increase in Hash Rate

Following hash rate graph shows the hash rate increases over time and is currently measured in exa-hashes. This means that in 1 second, Bitcoin network miners are computing more than 24,000,000,000,000,000 hashes per

Hash Rate

**96.66 EH/s**



# Mining Systems

- Over time, Bitcoin miners have used various methods to mine bitcoins.
- As the core principle behind mining is based on the double SHA256 algorithm, over time, experts have developed sophisticated systems to calculate the hash faster and faster.
- The following is a review of the different types of mining methods used in Bitcoin

# Mining Systems

**CPU mining** was the first type of mining available in the original Bitcoin client. Users could even use laptop or desktop computers to mine bitcoins. CPU mining is no longer profitable and now more advanced mining methods such as ASIC-based mining are used.

**GPU mining** - Due to the increased difficulty of the Bitcoin network and the general tendency of finding faster methods to mine, miners started to use the GPUs or graphics cards available in PCs to perform mining. GPUs support faster and parallelized calculations that are usually programmed using the OpenCL language.

# Mining Systems

**FPGA mining** - Even GPU mining did not last long, and soon miners found another way to perform mining using **Field Programmable Gate Arrays (FPGAs)**. An FPGA is basically an integrated circuit that can be programmed to perform specific operations. FPGAs are usually programmed in **hardware description languages (HDLs)**, such as Verilog and VHDL.

**ASIC mining** - ASICs were designed to perform SHA-256 operations. These special chips were sold by various manufacturers and offered a very high hashing rate. This worked for some time, but due to the quickly increasing mining difficulty level, single-unit ASICs are no longer profitable.

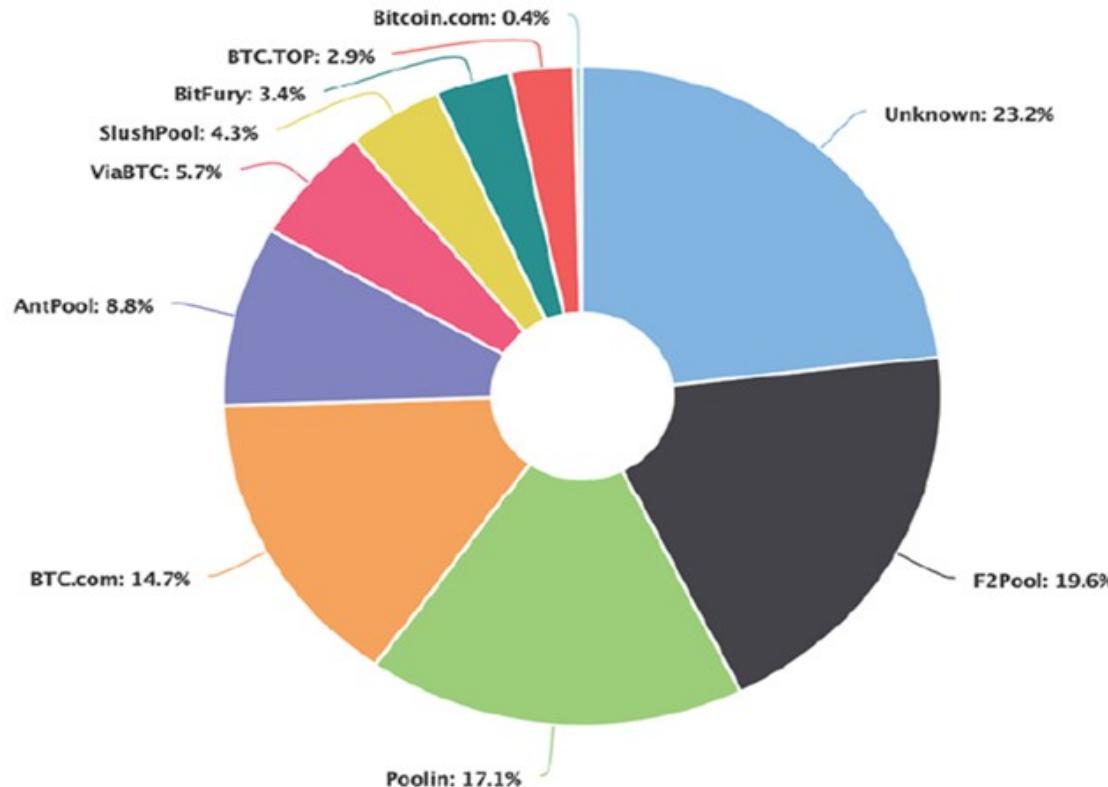
# Mining systems

A Mining Pool forms when a group of miners work together to mine a block.

The Pool Manager receives the coinbase transaction if the block is successfully mined, and is then responsible for distributing the reward to the group of miners who invested resources to mine the block.

This is more profitable than solo mining, where only one sole miner is trying to solve the partial hash inversion function (hash puzzle) because, in mining pools, the reward is paid to each member of the pool regardless of whether they solved the puzzle or not.

# Hashing Power Comparison of Mining Pools



# The Bitcoin Network & Payments

## Chapter 7

# Identification of Bitcoin Network

- A Bitcoin network is identified by its magic value.
- Magic values are used to indicate the message's origin network.
- A list of these values is shown in the following table:

Network	Magic value (in hex)
main	0xD9B4BEF9
testnet	0xDAB5BFFA
testnet3	0x0709110B

# Full Client and SPV client

- Bitcoin network nodes can fundamentally operate in two modes:

**Full clients** are thick clients or full nodes that download the entire blockchain; this is the most secure method of validating the blockchain as a client.

**SPV clients** are used to verify payments without requiring the download of a full blockchain. SPV nodes only keep a copy of block headers of the current longest valid blockchain. Verification is performed by looking at the Merkle branch, which links the transactions to the original block the transaction was accepted in.

# Wallets

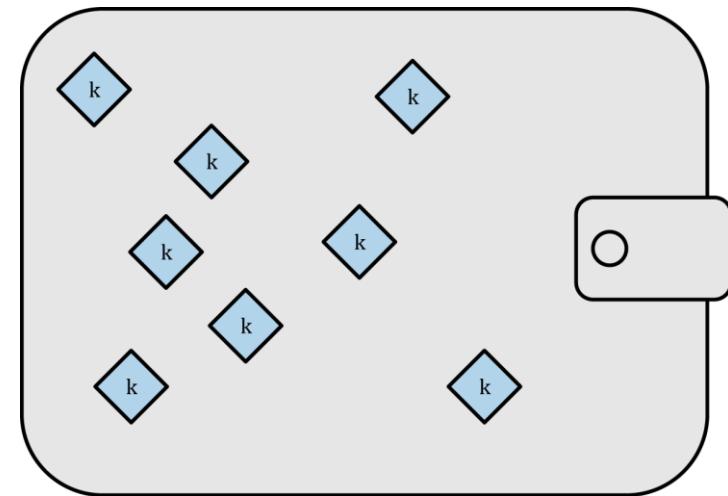
The wallet software is used to

- Generate and Store cryptographic keys.
- Receiving and Sending Bitcoin,
- Backing up keys, and
- Keeping track of the balance available.



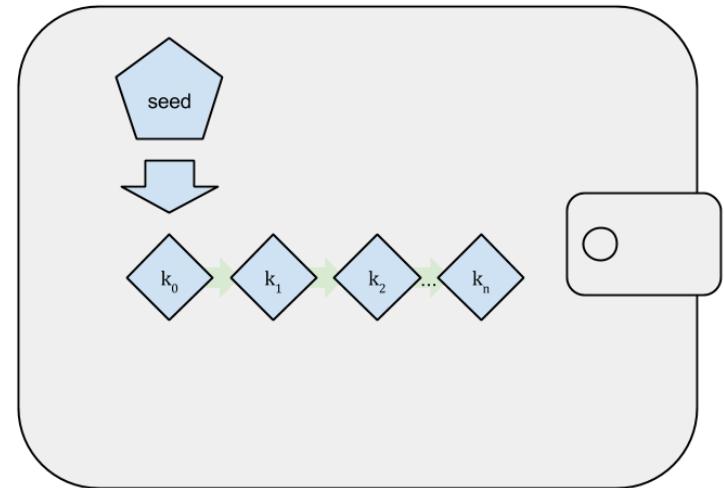
# Non-Deterministic Wallets

- These wallets contain randomly generated private keys and are also called **Just a Bunch of Key** wallets.
- The Bitcoin Core client generates some keys when first started and also generates keys as and when required.
- Managing a large number of keys is very difficult and an error-prone process that can lead to the theft and loss of coins. Moreover, there is a need to create regular backups of the keys and protect them appropriately, for example, by encrypting them in order to prevent theft or loss.



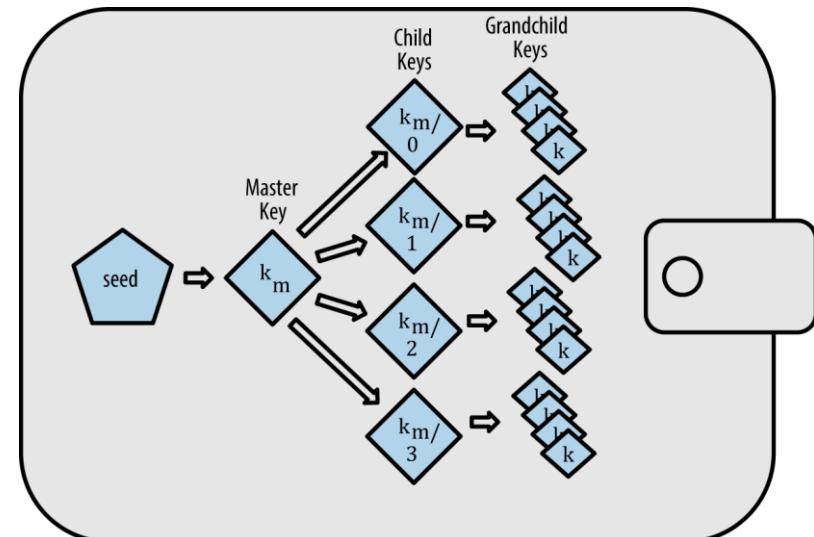
# Deterministic Wallets

- In this type of wallet, keys are derived from a seed value via hash functions.
- This seed number is generated randomly and is commonly represented by human-readable **mnemonic code** words. Mnemonic code words are defined in BIP39, a Bitcoin improvement proposal for Mnemonic code for generating deterministic keys.
- This BIP is available at <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>. These phrases can be used to recover all keys and make private key management comparatively easier.



# Hierarchical deterministic wallets

- Defined in BIP32 and BIP44, HD wallets store keys in a tree structure derived from a seed. The seed generates the parent key (master key), which is used to generate child keys and, subsequently, grandchild keys.
- Key generation in HD wallets does not generate keys directly; instead, it produces some information (private key generation information) that can be used to generate a sequence of private keys.
- The complete hierarchy of private keys in an HD wallet is easily recoverable if the master private key is known. It is because of this property that HD wallets are very easy to maintain and are highly portable.
- There are many free and commercially available HD wallets available, for example, Trezor (<https://trezor.io>), Jaxx (<https://jaxx.io/>), and Electrum (<https://electrum.org/>).



# Paper wallets

- As the name implies, this is a paper-based wallet with the required key material printed on it. It requires physical security to be stored.



Paper wallets can be generated online from various service providers, such as <https://bitcoinpaperwallet.com/> or <https://www.bitaddress.org/>.

# Hardware wallets

- Another method is to use a tamper-resistant device to store keys.
- This tamper-resistant device can be custom-built.
- With the advent of NFC enabled phones, this can also be a **secure element (SE)** in NFC phones.
- Trezor and Ledger wallets (various types) are the most commonly used Bitcoin hardware wallets:

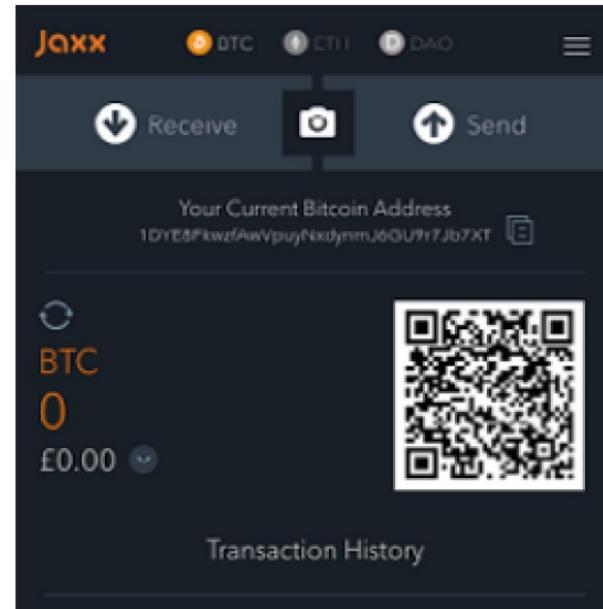


# Online wallets

- Online wallets, as the name implies, are stored entirely online and are provided as a service usually via the cloud.
- They provide a web interface to the users to manage their wallets and perform various functions, such as making and receiving payments.
- They are easy to use but imply that the user trusts the online wallet service provider.
- An example of an online wallet is GreenAddress, which is available at <https://greenaddress.it/en/>.

# Mobile Wallets

- Mobile wallets, as the name suggests, are installed on mobile devices.
- They can provide us with various methods to make payments, most notably the ability to use smartphone cameras to scan QR codes quickly and make payments.
- Mobile wallets are available for Android and iOS and include Blockchain Wallet, Breadwallet, Copay, and Jaxx:



# Choice of Wallet

- The choice of Bitcoin wallet depends on several factors such as security, ease of use, and available features. Out of all these attributes, security, of course, comes first, and when deciding about which wallet to use.
- Hardware wallets tend to be more secure compared to web wallets because of their tamper-resistant design.
- Web wallets, by their very nature, are hosted on websites, which may not be as secure as a tamper-resistant hardware device.
- Generally, mobile wallets for smart phone devices are quite popular due to a balanced combination of features and security.
- However, quite difficult to suggest which type of wallet should be used as it also depends on personal preferences and the features available in the wallet.
- Security should be kept in mind while making a decision on which wallet to choose.

# Bitcoin Payments



- Bitcoin can be accepted as payment using various techniques. Bitcoin is not recognized as a legal currency in many jurisdictions, but it is increasingly being accepted as a payment method by many online merchants and ecommerce websites.
- There are a number of ways in which buyers can pay a business that accepts Bitcoin. For example, on an online shop, Bitcoin merchant solutions can be used, whereas in traditional, physical shops, **Point of Sale (POS)** terminals and other specialized hardware can be used.
- Customers can simply scan the QR barcode with the seller's payment URI in it and pay using their mobile devices. Bitcoin URIs allow users to make payments by simply clicking on links or scanning QR codes. A **Uniform Resource Identifier (URI)** is basically a string that represents the transaction information. It is defined in BIP21. The QR code can be displayed near the point of the sale terminal. Nearly all Bitcoin wallets support this feature.
- Businesses can use the displayed image to advertise that they can accept Bitcoin as payment from customers:

# Bitcoin Payment Solutions

Various payment solutions, such as XBT terminal and the 34 Bytes Bitcoin POS terminal, are available commercially.

Generally, these solutions work by following these steps:

1. The salesperson enters the amount of money to be charged in fiat currency; for example, US dollars.
2. Once the value is entered in the system, the terminal prints a receipt with a QR code on it and other relevant information, such as the amount to be paid.
3. The customer can then scan this QR code using their mobile Bitcoin wallet to send the payment to the Bitcoin address of the seller embedded within the QR code.
4. Once the payment is received at the designated Bitcoin address, a receipt is printed out as physical evidence of the sale.

34 Bytes POS solution



# Bitcoin Payment Standards

- Various BIPs have been proposed and finalized in order to introduce and standardize Bitcoin payments. Most notably, BIP70 (secure payment protocol) describes the protocol for secure communication between a merchant and customers.
- This protocol uses X.509 certificates for authentication and runs over HTTP and HTTPS. There are three messages in this protocol: PaymentRequest, Payment, and PaymentACK.
- The key features of this proposal are defense against man-in-the-middle attacks and secure proof of payment.
- Man-in-the-middle attacks can result in a scenario where the attacker is sitting between the merchant and the buyer, and it would seem to the buyer that they are talking to the merchant, but in fact, the man in the middle is interacting with the buyer instead of the merchant. (Result in manipulation of the merchant's Bitcoin address to defraud the buyer)
- Several other BIPs, such as BIP71 (Payment Protocol MIME types) and BIP72 (URI extensions for Payment Protocol), have also been implemented to standardize payment scheme to support BIP70 (Payment Protocol).

# Innovation in Bitcoin

- Bitcoin has undergone many changes and is still evolving into a more and more robust and better system by addressing various weaknesses.
- As such, various proposals have been made in the last few years to improve Bitcoin performance, resulting in greater transaction speed, increased security, payment standardization, and overall performance improvement at the protocol level.
- These improvement proposals are usually made in the form of **Bitcoin Improvement Proposals (BIPs)** or fundamentally new versions of Bitcoin protocols, resulting in new networks altogether.
- Some of the changes proposed can be implemented via a soft fork, but a few need a hard fork and, as a result, give birth to a new currency.

# Bitcoin Improvement Proposals

BIPs, are used to propose improvements / inform the Bitcoin community about the improvement suggested, design issues/some aspects of Bitcoin ecosystem.

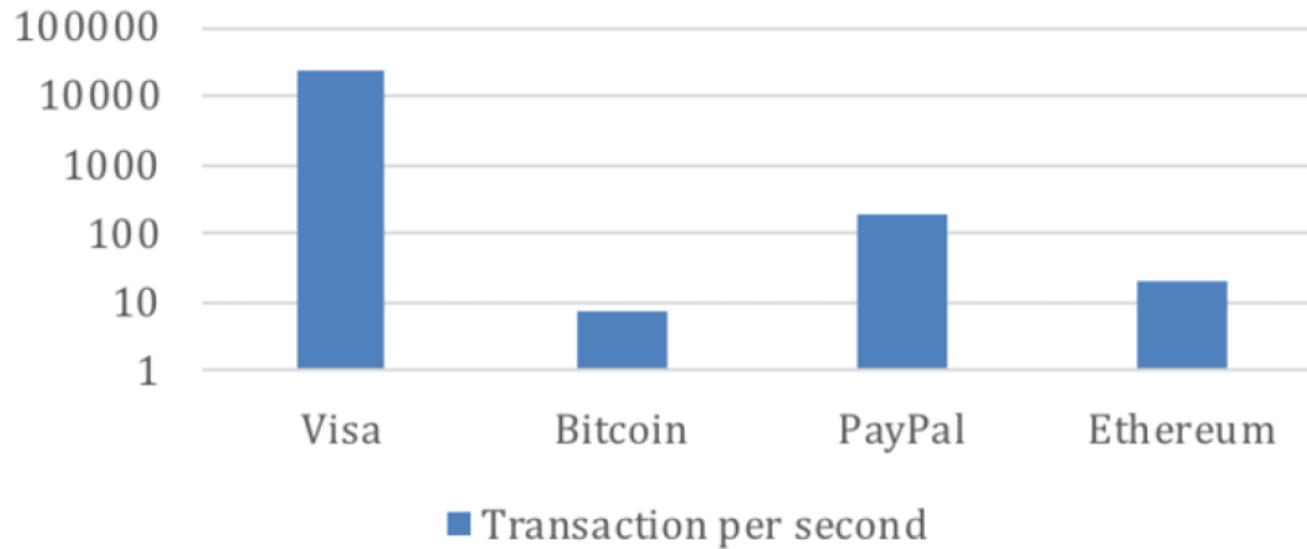
There are three types of BIPs:

- **Standard BIP:** Used to describe the major changes that have a major impact on the Bitcoin system; for example, block size changes, network protocol changes, or transaction verification changes.
- **Process BIP:** A major difference between standard and process BIPs is that standard BIPs cover protocol changes, whereas process BIPs usually deal with proposing a change in a process that is outside the core Bitcoin protocol. These are implemented only after a consensus among Bitcoin users.
- **Informational BIP:** These are usually used to just advise or record some information about the Bitcoin ecosystem, such as design issues.

# Advanced Protocols

- Various advanced protocols that have been suggested or implemented for improving the Bitcoin protocol.
- For example, transaction throughput is one of the critical issues that need a solution. The Bitcoin network can only process approximately three to seven TPS, which is a tiny number compared to other financial networks.
- For example, the Visa network can process approximately, on average, 24,000 TPS (transactions per second). PayPal can process approximately 200 transactions per second, whereas Ethereum can process up to, on average, 20.
- As the Bitcoin network has grown exponentially over the last few years, these issues have started to grow even further.
- The difference in processing speed is also shown in the graph on Next Slide.

## Transaction per second



# Segregated Witness

The SegWit or Segregated Witness is a soft fork-based upgrade of the Bitcoin protocol that addresses weaknesses such as throughput and security in the Bitcoin protocol.

- To spend an **unspent transaction output (UTXO)** in Bitcoin, a valid signature needs to be provided. In the pre-SegWit scenario, this signature is provided within the locking script, whereas in SegWit this signature is not part of the transaction and is provided separately.
- The stated purpose of Segregated Witness is to prevent non-intentional Bitcoin transaction malleability and allow for more transactions to be stored within a block.
- SegWit was also intended to solve a blockchain size limitation problem that reduced Bitcoin transaction speed.

# Innovation in Bitcoin

- Some other innovative ideas in the Bitcoin space. Not only has the original Bitcoin evolved quite significantly since its introduction, but there are also new blockchains that are either forks of Bitcoin or novel implementations of the Bitcoin protocol with advanced features
- Bitcoin cash
- Bitcoin unlimited
- Bitcoin gold



# Bitcoin Cash

- Bitcoin Cash (BCH) increases the block limit to 8 MB.
- This change immediately increases the number of transactions that can be processed in one block to a much larger number compared to the 1 MB limit in the original Bitcoin protocol.
- It uses Proof of Work (PoW) as a consensus algorithm, and mining hardware is still ASIC-based.
- The block interval is changed from 10 minutes to 10 seconds and up to 2 hours.
- It also provides replay protection and wipe-out protection, which means that because BCH uses a different hashing algorithm, it prevents it being replayed on the Bitcoin blockchain.
- It also has a different type of signature compared to Bitcoin to differentiate between two blockchains.

# Bitcoin Unlimited

- Bitcoin Unlimited increases the size of the block without setting a hard limit. Instead, miners come to a consensus on the block size cap over a period of time. Other concepts such as extremely thin blocks and parallel validation have also been proposed
- Extremely thin blocks allow for faster block propagation between Bitcoin nodes. In this scheme, the node requesting blocks sends a getdata request, along with a bloom filter, to another node. The purpose of this bloom filter is to filter out the transactions that already exist in the **memory pool (mempool)** of the requesting node. The node then sends back a **thin block** only containing the missing transactions. This fixes an inefficiency in Bitcoin whereby transactions are regularly received twice – once at the time of broadcast by the sender and then again when a mined block is broadcasted with the confirmed transaction.
- Parallel validation allows nodes to validate more than one block, along with new incoming transactions, in parallel. This mechanism is in contrast to Bitcoin, where a node, during its validation period after receiving a new block, cannot relay new transactions or validate any blocks until it has accepted or rejected the block.

# Bitcoin Gold

- This proposal has been implemented as a hard fork since block 491407 of the original Bitcoin blockchain. Being a hard fork, it resulted in a new blockchain, named Bitcoin Gold.
- The core idea behind this concept is to address the issue of mining centralization, which has hurt the original Bitcoin idea of decentralized digital cash, whereby more hash power has resulted in a power shift toward miners with more hashing power.
- It uses the Equihash algorithm as its mining algorithm instead of PoW; hence, it is inherently ASIC resistant and uses GPUs for mining.

# Other proposals

There are other proposals like

- Bitcoin Next Generation,
- Solidus,
- Spectre, and
- Segwit2x

# Bitcoin investment and buying and selling

- There are many online exchanges where users can buy and sell Bitcoin.
- This is a big business on the internet now and it offers Bitcoin trading, CFDs, spread betting, margin trading, and various other choices.
- Traders can buy Bitcoin or trade by opening long or short positions to make a profit when the price of Bitcoin goes up or down.
- Several other features, such as exchanging Bitcoin for other virtual currencies, are also possible, and many online Bitcoin exchanges provide this function.
- Advanced market data, trading strategies, charts, and relevant data to support traders is also available. An example is shown from CEX (<https://cex.io>)

# Exchange

CEX.IO

BUY / SELL

MARKET DATA

MARGIN TRADING

SIGN IN

REGISTER



BTC/USD  
641.6617

BTC/EUR  
581.9899

BTC/RUB  
40000.69

ETH/BTC  
0.01870728

ETH/USD  
12.08170000

ETH/EUR  
10.82830000

LTC/USD  
3.87650000

LTC/EUR  
3.4900

LTC/BTC  
0.00602549

GHS/BTC  
0.00010000

BTC/USD

Last price:  
**\$ 641.6617**

Daily change:  
**\$ -2.4695**

Today's open:  
**\$ 644.1312**

24h volume:  
**B 393.23521389**

Chart

1m 3m 5m 15m 30m 1h 2h 4h 6h 12h 1d 3d 1w

646.4

24

644.8

18

643.2

12

641.6

6

640.0

0



# Order Book

- Order book displays sell and buy orders. Sell orders are also called ask orders, while buy orders are also called bid orders.
- Ask price is what the seller is willing to sell, whereas the bid price is what the buyer is willing to pay.
- If the bid and ask prices match, then a trade can occur.
- The most common order types are market orders and limit orders.
- Market Orders mean that as soon as the prices match, the order will be fulfilled immediately.
- Limit Orders allow for buying and selling a set Quantity at a specified price or better.
- Also, a period of time can be set, during which the order can be left open. If it's not executed, then it will be canceled.

# Order Book

## Sell Orders

⌚ Total BTC available: 656.41831367

Price per BTC	BTC Amount	Total: (USD)
642.4085	฿0.20450000	\$ 131.38
642.4915	฿0.20910000	\$ 134.35
643.4470	฿0.05000000	\$ 32.18
643.4900	฿0.11944972	\$ 76.87
643.5000	฿1.85748652	\$ 1195.30
643.6500	฿3.00000000	\$ 1930.95
643.6999	฿0.13844181	\$ 89.12
643.7000	฿45.80000000	\$ 29481.46
643.7487	฿1.22995538	\$ 791.79

## Buy Orders

⌚ Total USD available: 380739.41

Price per BTC	BTC Amount	Total: (USD)
641.6210	฿0.01390000	\$ 8.92
641.6201	฿0.23162780	\$ 148.62
641.6200	฿0.12050000	\$ 77.32
641.6117	฿1.83477084	\$ 1177.22
641.5584	฿0.30000000	\$ 192.47
641.5217	฿0.18180000	\$ 116.63
641.0217	฿0.10000000	\$ 64.11
640.5300	฿0.67323160	\$ 431.23
640.5000	฿0.40815400	\$ 261.43

# Summary

- Introduction to the Bitcoin network
- Discussion on Bitcoin node discovery and block synchronization protocols
- Different types of network messages
- Bitcoin wallets and the various attributes and features of each type
- Bitcoin payments and payment processors
- BIPs and advanced Bitcoin protocols
- Basic introduction to Bitcoin buying and selling

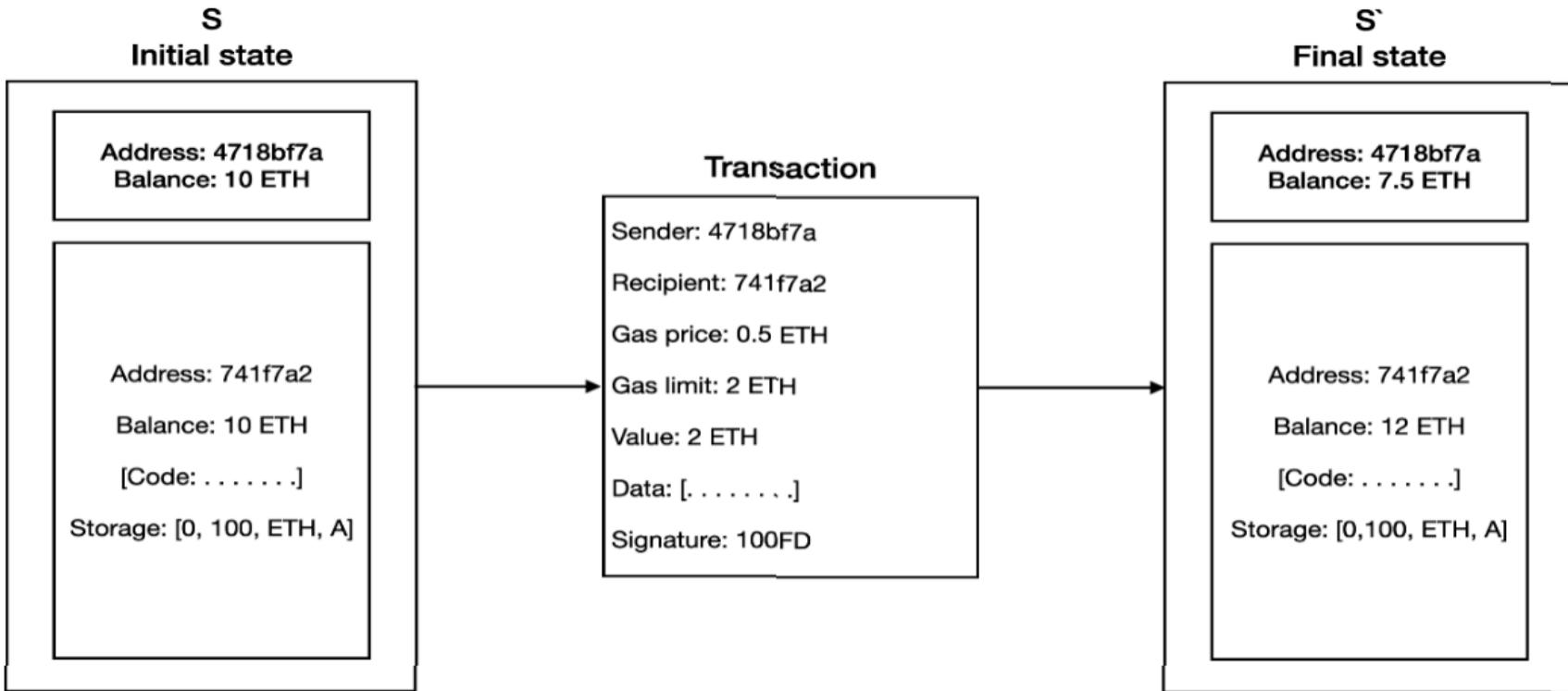
# Ethereum 101

CHAPTER # 11

# Ethereum Network

- Vitalik Buterin conceptualized Ethereum in November, 2013.
- The first version of Ethereum, called Olympic, was released in May, 2015.
- The core idea proposed was the development of a Turing-complete language that allows the development of arbitrary programs (smart contracts) for blockchain and Decentralized Applications (DApps).
- This concept is in contrast to Bitcoin, where the scripting language is limited and only allows necessary operations.

# The Ethereum Blockchain

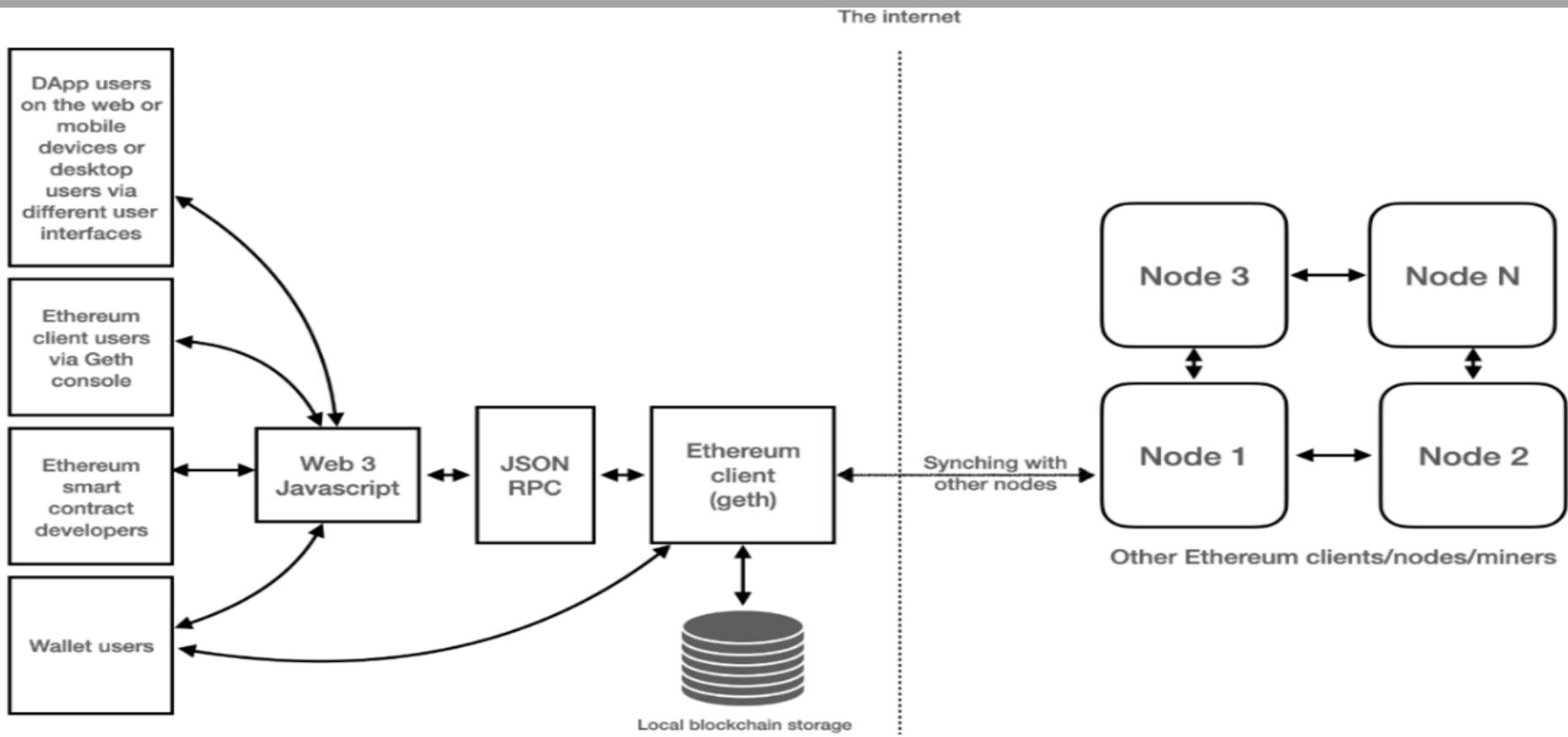


# Components of the Ethereum ecosystem

The Ethereum blockchain stack consists of various components.

- At the core, there is the Ethereum blockchain running on the peer-to-peer Ethereum network.
- Secondly, there's an Ethereum client (usually Geth) that runs on the nodes and connects to the peer-to-peer Ethereum network from where blockchain is downloaded and stored locally.
- It provides various functions, such as mining and account management.
- The local copy of the blockchain is synchronized regularly with the network.
- Another component is the web3.js library that allows interaction with the geth client via the Remote Procedure Call (RPC) interface.

# Ethereum's High Level Architecture



# Keys & Addresses

- Keys and addresses are used in the Ethereum blockchain to represent ownership and transfer ether.
- The keys used are made up of pairs of private and public parts.
- The private key is generated randomly and is kept secret, whereas a public key is derived from the private key.
- Addresses are derived from public keys and are 20-byte codes used to identify accounts

# Process of Key Generation & Address Derivation

- Private Key is randomly chosen (a 256-bit positive integer) under the rules defined by the elliptic curve secp256k1 specification.
- Public Key is then derived from this private key using the Elliptic Curve Digital Signature Algorithm (ECDSA) recovery function.
- Address is derived from the public key, specifically, from the rightmost 160 bits of the Keccak hash of the public key.

# Process of Key Generation & Address Derivation

## - Examples

- **Private Key:**

B51928c22782e97cca95c490eb958b06fab7a70b9512c38c36974f47b954ffc  
4

- **Public key:**

3aa5b8eef12bdc2d26f1ae348e5f383480877bda6f9e1a47f6a4afb35cf998ab  
8471e3948b1173622dafc6b4ac198c97b18fe1d79f90c9093ab2ff9ad99260

- **Address:**

0x77b4b5699827c5c49f73bd16fd5ce3d828c36f32

# Accounts

- Accounts are used by users to interact with the blockchain via transactions.
- Accounts are one of the main building blocks of the Ethereum blockchain, they are defined by pairs of private and public keys
- A transaction is digitally signed by an account before submitting it to the network via a node.
- All accounts have a state that, when combined together, represents the state of the Ethereum network.
- the state is created or updated as a result of the interaction between accounts and transaction executions, operations performed between and on the accounts represent state transitions.

# Accounts

The state transition is achieved using what's called the Ethereum state transition function, which works as follows:

- Confirm the transaction validity by checking the syntax, signature validity, and nonce.
- The transaction fee is calculated, and the sending address is resolved using the signature. Furthermore, the sender's account balance is checked and subtracted accordingly, and the nonce is incremented. An error is returned if the account balance is insufficient.

# Accounts

- Provide enough ETH (the gas price) to cover the cost of the transaction. This also depends on the amount of gas available.
- If enough gas is available, then the contract code will be executed fully; otherwise, it will run up to the point where it runs out of gas
- In cases of transaction failure due to insufficient account balance or gas, all state changes are rolled back except for the fee payment, which is paid to the miners.
- Finally, the remainder (if any) of the fee is sent back to the sender as change and the fee is paid to the miners accordingly.

# Types of Accounts

## 1. External Owned Account (EOA)

- They are associated with a human user, hence are also called user accounts.
- EOAs have an ether balance.
- They are capable of sending transactions.
- They have no associated code.
- They are controlled by private keys.
- EOAs cannot initiate a call message.
- Accounts contain a key-value store.
- EOAs can initiate transaction messages.

# Types of Accounts

## 2. Contract Account (CA)

- Created as a result of deployment of a smart contract
- They have a state, represented by an address
- They are not intrinsically associated with any user or actor on the blockchain.
- They have associated code that is kept in memory/storage on the blockchain.
- They have access to storage, they contain a key-value store.
- They can get triggered and execute code in response to a transaction or a message from other contracts.
- CAs can maintain their permanent states and can call other contracts.
- CAs' addresses are generated when they are deployed, This address of the contract is used to identify its location on the blockchain.

# Transactions & Messages

A transaction in Ethereum is a digitally signed data packet using a private key that contains the instructions that, when completed, either result in a message call or contract creation.

Transactions can be divided into two types based on the output they produce:

- Message call transactions: This transaction simply produces a message call that is used to pass messages from one CA to another.
- Contract creation transactions: As the name suggests, these transactions result in the creation of a new CA. This means that when this transaction is executed successfully, it creates an account with the associated code.

# Transactions and messages

A transaction in Ethereum consists of a number of fields, as shown here, along with transaction trie.

**Nonce:** The nonce is a number that is incremented by one every time a transaction is sent by the sender.

**Gas price:** The gas price field represents the amount of Wei required to execute the transaction.

**Gas limit:** The gas limit field contains the value that represents the maximum amount

of gas that can be consumed to execute the transaction.

**To:** As the name suggests, the To field is a value that represents the address of the recipient of the transaction. This is a 20 byte value, for transactions related to deployment of contract, the to field is empty.

# Transactions and messages

**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.

**Value:** refers to the amount of Ether that is transferred from one account to another.

**Init / 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.

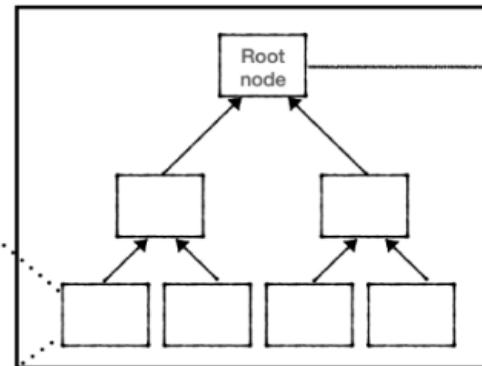
**Data:** If the transaction is a message call, then the Data field is used instead of init, and represents the input data of the message call

# Transactions and messages

Transaction

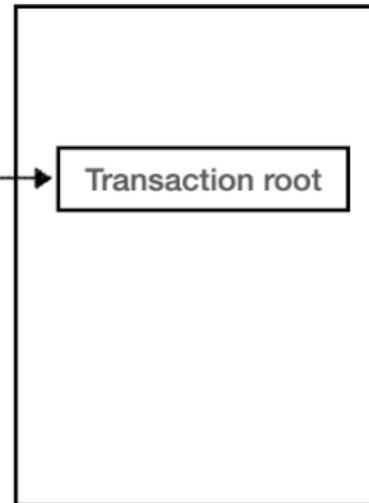
Nonce
Gas price
Gas limit
To (recipient)
Value
V, R, S (Sender)
Init or Data

Transaction Trie



Keccak 256-bit hash  
of the root node

Block header



# Recursive Length Prefix (RLP)

RLP is an encoding scheme developed by Ethereum developers.

- It is a specially developed encoding scheme that is used in Ethereum to serialize binary data for storage or transmission over the network and also to save the state in a Patricia tree on storage media.
- It is a deterministic and consistent binary encoding scheme used to serialize objects on the Ethereum blockchain, such as account state, transactions, messages, and blocks.
- It operates on strings and lists to produce raw bytes that are suitable for storage and transmission.
- RLP is a minimalistic and simple-to-implement serialization format that does not define any data types and simply stores structures as nested arrays. In other words, RLP does not encode specific data types; instead, its primary purpose is to encode structures.

# Contract creation transactions

- A contract creation transaction is used to create smart contracts on the blockchain
- Addresses generated as a result of a contract creation transaction are 160 bits in length.
- The new account is initialized when the EVM code is executed.
- In the case of any exception during code execution(out of gas) , such as not having enough gas the state does not change.
- If the execution is successful, then the account is created after the payment of appropriate gas costs.

# Contract creation transactions

There are a few essential parameters required for a contract creation transaction.

These are listed as follows:

- The sender
- The transaction originator
- Available gas
- Gas price
- Endowment, which is the amount of ether allocated
- A byte array of an arbitrary length
- Initialization EVM code
- The current depth of the message call/contract-creation stack (current depth means the number of items that are already present in the stack)

# Message call transactions

- Message call is the act of passing a message from one account to another. If the destination account has an associated EVM code, then the EVM will start upon the receipt of the message to perform the required operations.
- If the message sender is an autonomous object (external actor), then the call passes back any data returned from the EVM operation.
- The state is altered by transactions, These transactions are created by external factors (users) and are signed and then broadcasted to the Ethereum network.
- message call transaction is a write operation and is used for invoking functions in a CA (Contract Account, / smart contract), which does cost gas and results in a state change.

# Message call transactions

A message call requires several parameters for execution which are listed as follows:

- The sender
- The transaction originator
- The recipient
- The account whose code is to be executed (usually the same as the recipient)
- Available gas
- The value
- The gas price

(continued - next page )

# Message call transactions

- An arbitrary-length byte array
- The input data of the call
- The current depth of the message call/contract creation stack

# Messages

- Messages are the data and values that are passed between two accounts.
- Messages are generated when the CALL or DELEGATECALL opcodes are executed by the contract running in the EVM.
- Contracts have the ability to send “messages” to other contracts.
- Messages only exist in the execution environment and are never stored.
- It can either be sent via a smart contract (autonomous object) or from an external actor (an Externally Owned Account) in the form of a transaction that has been digitally signed by the sender.
- Messages are similar to transactions. When called by EOA it is called Transaction, when produced by contract it's called Message

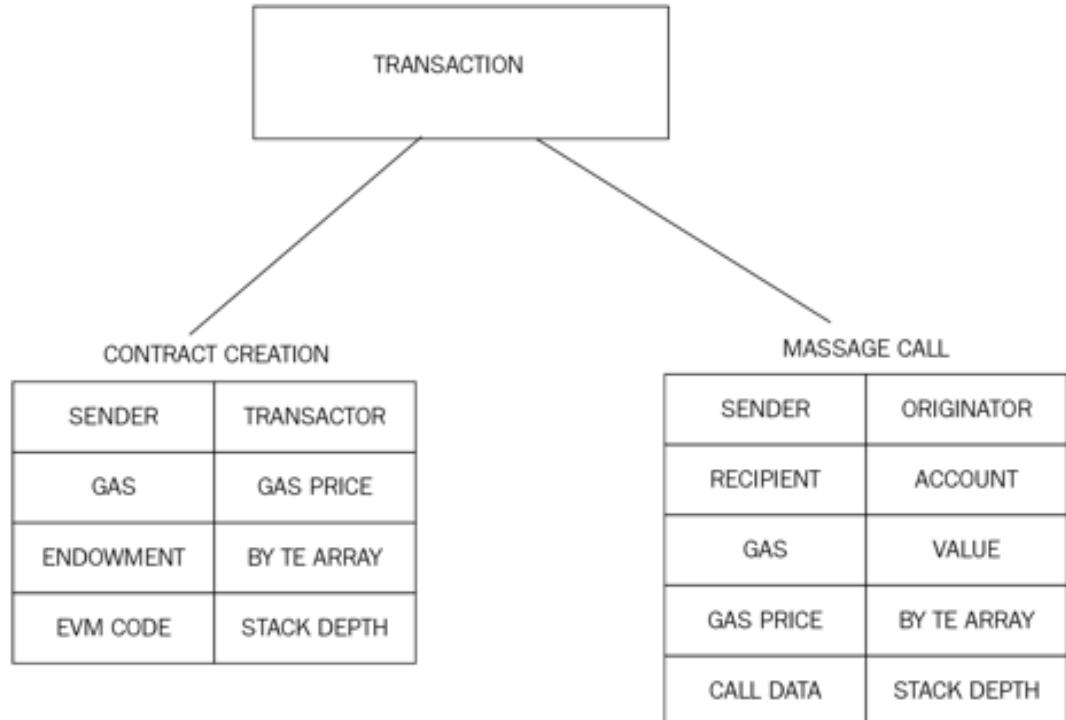
# Messages

A message consists of the following components:

- The sender of the message
- The recipient of the message
- Amount of Wei to transfer and the message to be sent to the contract address
- An optional data field (input data for the contract)
- The maximum amount of gas ( startgas ) that can be consumed

# Messages

The segregation between the two types  
Of transactions  
(contract creation and message calls)



# Calls

- A call does not broadcast anything to the blockchain, instead, it is a local call and executes locally on the Ethereum node.
- It does not consume any gas as it is a read-only operation.
- Calls also do not allow ether transfer to CAs.
- Calls are executed locally on a node EVM and do not result in any state change because they are never mined.
- Calls are processed synchronously and they usually return the result immediately

# Transaction Validation and Execution

Transactions are executed after their validity has been verified. The initial checks are listed as follows:

- A transaction must be well formed and RLP-encoded without any additional trailing bytes
- The digital signature used to sign the transaction must be valid
- The transaction nonce must be equal to the sender's account's current nonce
- The gas limit must not be less than the gas used by the transaction
- The sender's account must contain sufficient balance to cover the execution cost

# The Transaction Substate

A transaction substate is created during the execution of the transaction and is processed immediately after the execution completes.

This transaction substate is a tuple that is composed of four items.

- Suicide set or self-destruct set: This element contains the list of accounts (if any) that are disposed of after the transaction executes.
- Refund balance: This is the total price of gas for the transaction that initiated the execution. Refunds are not immediately executed; instead, they are used to offset the total execution cost partially.

# The Transaction Substate

- Log series: This is an indexed series of checkpoints that allows the monitoring and notification of contract calls to the entities external to the Ethereum environment, such as application frontends. It works like a trigger mechanism that is executed every time a specific function is invoked, or a specific event occurs. Logs are created in response to events occurring in the smart contract. It can also be used as a cheaper form of storage.

# The transaction substate

- Touched accounts: Touched accounts can be defined as those accounts which are involved any potential state changing operation. Empty accounts from this set are deleted at the end of the transaction. The final state is reached after deleting accounts in the self-destruct set and emptying accounts from the touched accounts set.

# State storage in the Ethereum blockchain

At a fundamental level, the Ethereum blockchain is a transaction- and consensus-driven state machine. the state needs to be stored permanently in the blockchain. for this purpose, the world state, transactions, and transaction receipts are stored on the blockchain in blocks.

- The world state : This is a mapping between Ethereum addresses and account states. the addresses are 20 bytes (160 bits) long. this mapping is a data structure that is serialized using RLP.
- The account state : The account state consists of four fields: nonce, balance, storage root, and code hash,

# State storage in the Ethereum blockchain

## The account state

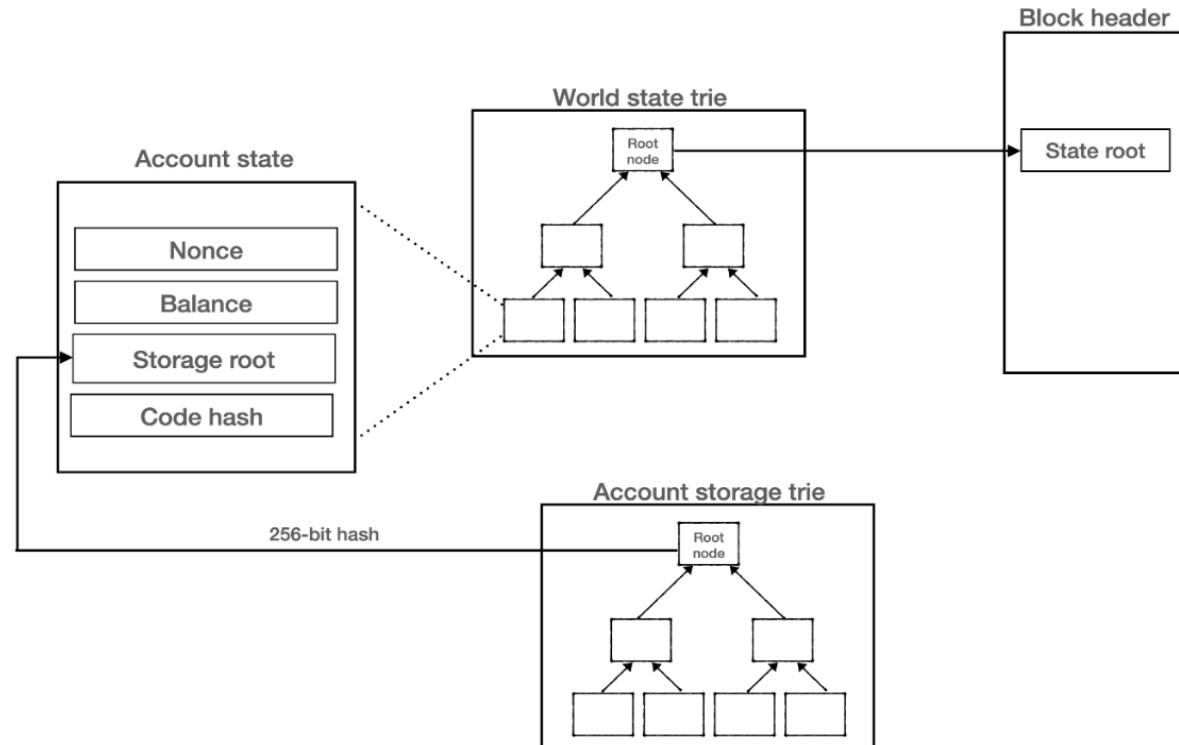
- **Nonce:** This is a value that is incremented every time a transaction is sent from the address. In the case of CAs, it represents the number of contracts created by the account.
- **Balance:** This value represents the number of weis, which is the smallest unit of the currency (ether) in Ethereum, held by the given address.
- **Storage root:** This field represents the root node of an MPT that encodes the storage contents of the account.
- **Code hash:** This is an immutable field that contains the hash of the smart contract code that is associated with the account. In the case of normal accounts, this field contains the Keccak 256-bit hash of an empty string. This code is invoked via a message call.

# State storage in the Ethereum blockchain

The diagram shows the fields contained within the account state, and how the various elements are contained Within the world state trie:

World state trie  
State root

Account state  
Account storage trie



# Transaction receipts

Transaction receipts are used as a mechanism to store the state after a transaction has been executed. In other words, these structures are used to record the outcome of the transaction execution. It is produced after the execution of each transaction. All receipts are stored in an index-keyed trie. The hash (a 256-bit Keccak hash) of the root of this trie is placed in the block header as the receipt's root.

It is composed of four elements - (Next Slide)

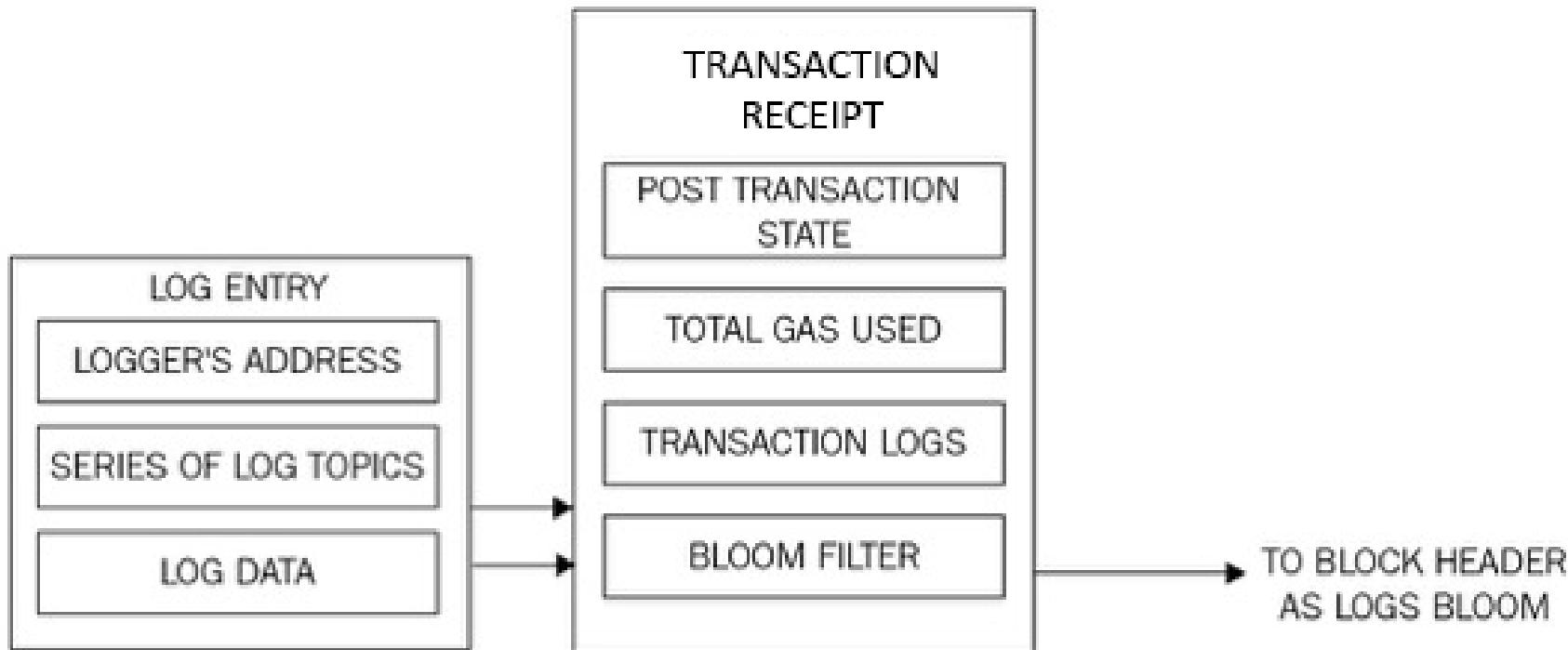
# Transaction receipts

- The post-transaction state: This item is a trie structure that holds the state after the transaction has been executed. It is encoded as a byte array.
- Gas used: This item represents the total amount of gas used in the block that contains the transaction receipt. The value is taken immediately after the transaction execution is completed. The total gas used is expected to be a non-negative integer.
- Set of logs: This field shows the set of log entries created as a result of the transaction execution. Log entries contain the logger's address, a series of log topics, and the log data.

# Transaction Receipts

**Bloom filter:** is created from the information contained in the set of logs discussed earlier. A log entry is reduced to a hash of 256 bytes, which is then embedded in the header of the block as the logs bloom. A log entry is composed of the logger's address, log topics, and log data. Log topics are encoded as a series of 32- byte data structures. The log data is made up of a few bytes of data.

# Transaction Receipts



# Ether Cryptocurrency (ETC and ETH)

- As an incentive to the miners, Ethereum rewards its own native currency called ether (abbreviated as ETH).
- After the Decentralized Autonomous Organization (DAO) hack a hard fork was proposed in order to mitigate the issue; therefore, there are now two Ethereum blockchains:
- one is called Ethereum Classic and its currency is represented by ETC.
- whereas the hard-forked version is ETH, which continues to grow and on which active development is being carried out

# Ether (ETH)

- Ether is minted by miners as a currency reward for the computational effort they spend to secure the network by verifying transactions and blocks.
- Ether is used within the Ethereum blockchain to pay for the execution of contracts on the EVM
- Ether is used to purchase gas as crypto fuel, which is required to perform computation on the Ethereum blockchain.
- Fees are charged for each computation performed by the EVM on the blockchain

# ETH Denomination

<b>Unit</b>	<b>Alternative name</b>	<b>Wei value</b>	<b>Number of weis</b>
Wei	Wei	1 Wei	1
KWei	Babbage	1e3 Wei	1,000
MWei	Lovelace	1e6 Wei	1,000,000
GWei	Shannon	1e9 Wei	1,000,000,000
microether	Szabo	1e12 Wei	1,000,000,000,000
milliether	Finney	1e15 Wei	1,000,000,000,000,000
ether	ether	1e18 Wei	1,000,000,000,000,000,000

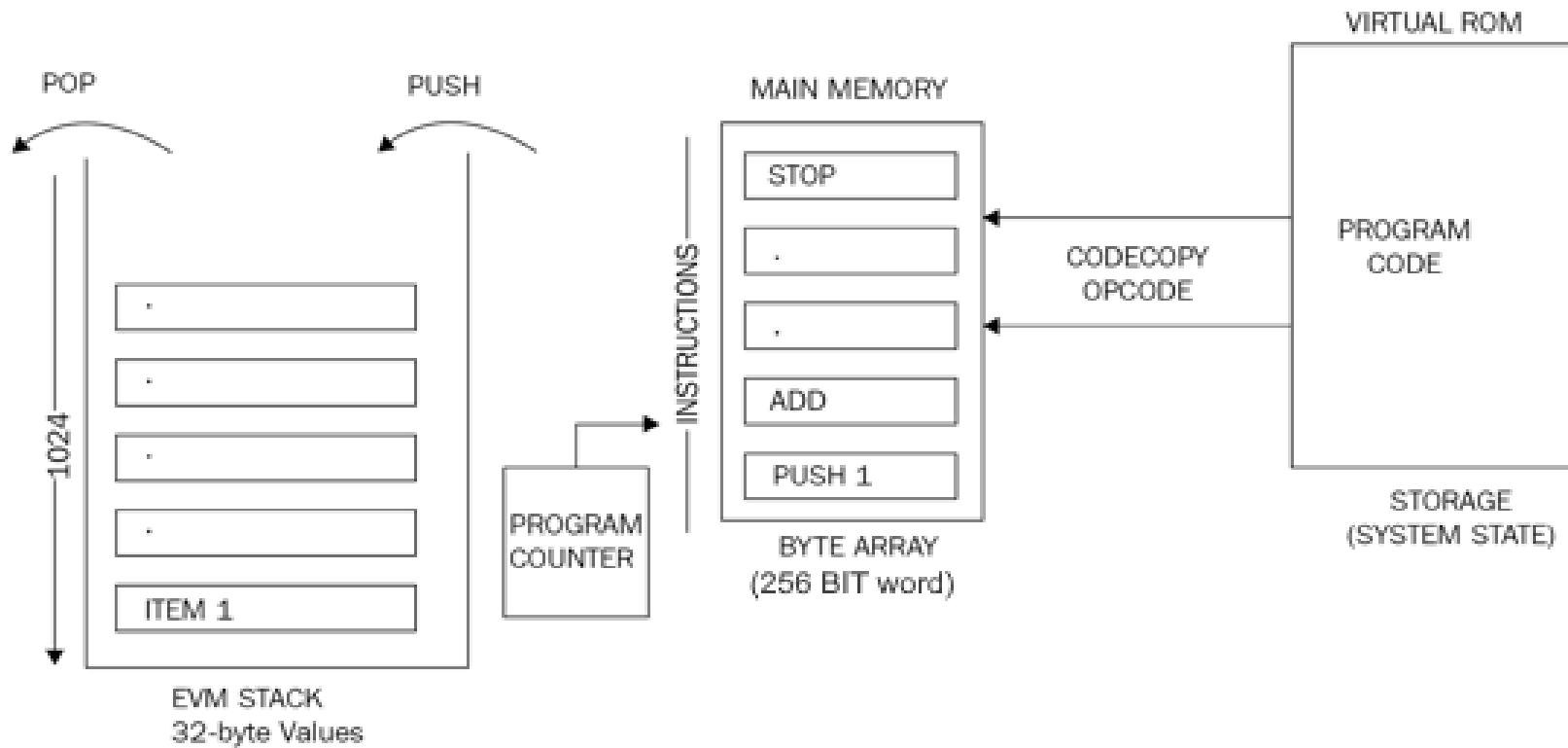
# Ethereum Virtual Machine (EVM)

- EVM is a simple stack-based execution machine that runs bytecode instructions to transform the system state from one state to another.
- Word size of the EVM is set to 256-bit.
- Stack size is limited to 1,024 elements and is based on the Last In, First Out (LIFO) queue.
- EVM is an entirely isolated and sandboxed runtime environment. The code that runs on the EVM does not have access to any external resources such as a network or filesystem. This results in increased security, deterministic execution, and allows untrusted code (code that can be run by anyone) to be executed on Ethereum blockchain.

# Storage Available on EVM

- **Memory:** Addressed byte array. When a contract finishes its code execution, the memory is cleared.
- **Storage:** is a key-value store and is permanently persisted on the blockchain. Keys and values are each 256 bits wide. It is allocated to all accounts on the blockchain. As a security measure, storage is only accessible by its own respective CAs. It can be thought of as hard disk storage.
- **Stack:** EVM is a stack-based machine, and performs all computations in a data area called the stack. All in-memory values are also stored in the stack. It has maximum depth of 1024 elements & supports word size of 256 bits.

# EVM operation design



# Execution Environment

There are some key elements that are required by the execution environment to execute the code  
As shown here:

Address of code owner
Sender address
Gas price
Input data
Initiator address
Value
Bytecode
Block header
Message call depth
Permission

# Machine State

Machine state is maintained internally & updated after each EVM execution cycle

An iterator function runs in the EVM, which outputs the results of a single cycle of the state machine.

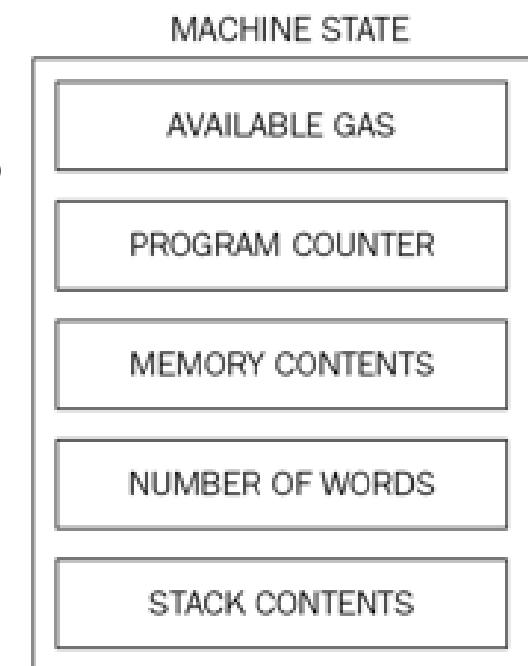
EVM is designed to handle exceptions and will halt (stop execution) if any of the following exceptions should occur:

- Not having enough gas required for execution
- Invalid instructions
- Insufficient stack items
- Invalid destination of jump opcodes
- Invalid stack size (greater than 1,024)

# Machine State

The machine state consists of the following elements:

- Available gas
- Program counter, which is a +ve integer of up to 256
- The contents of the memory (a series of zeroes of size  $2^{256}$ )
- The active number of words in memory (counting continuously from position 0)
- The contents of the stack



# The Iterator Function

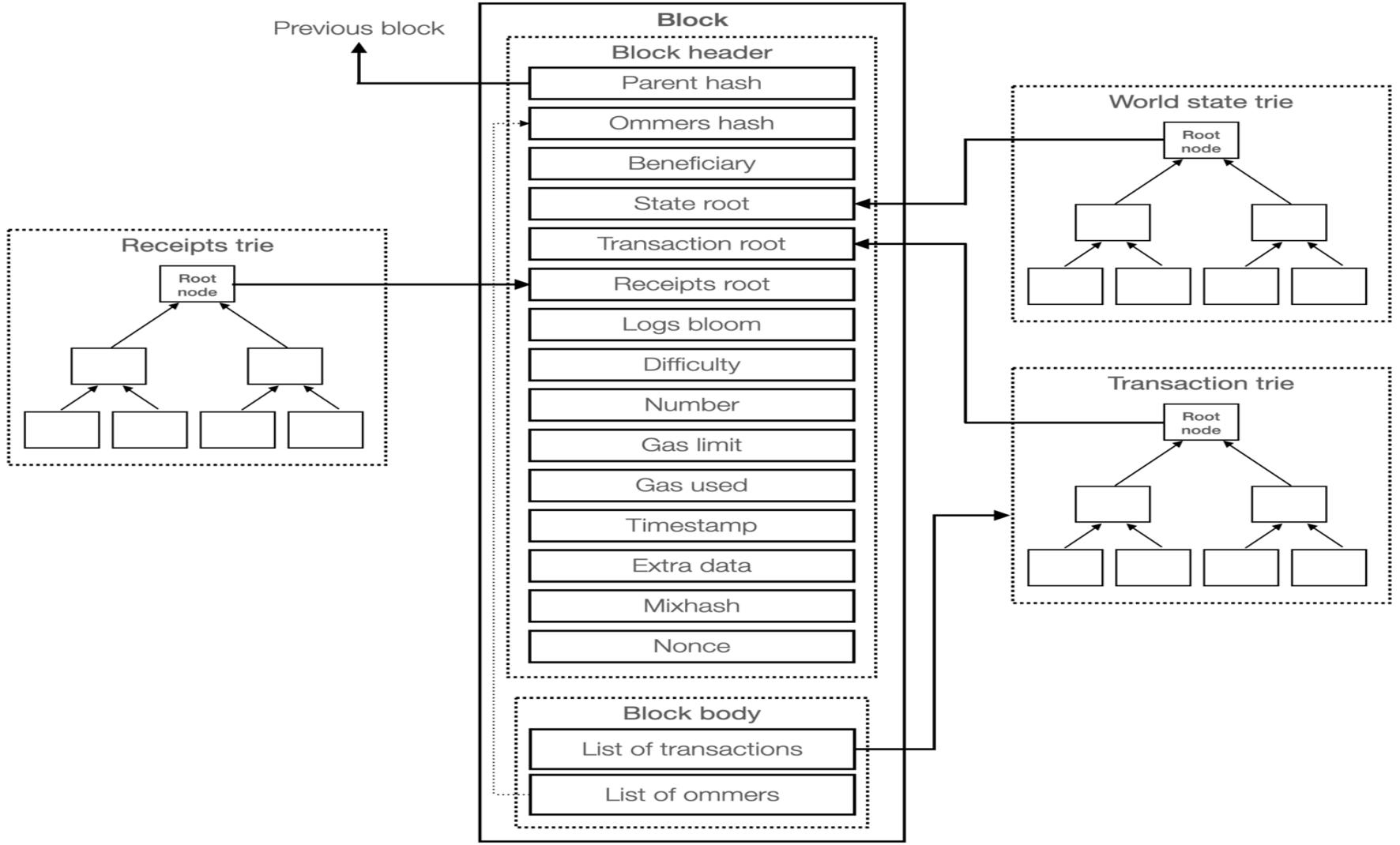
Performs various vital functions that are used to set the next state of the machine and eventually the world state

- It fetches the next instruction from a byte array where the machine code is stored in the execution environment.
- It adds/removes ( PUSH/POP ) items from the stack accordingly.
- Gas is reduced according to the gas cost of the instructions/opcodes.  
It increments the Program Counter (PC).

# Further Ethereum

CHAPTER # 12





# Blocks & Blockchain

Blocks are the main building structure of a blockchain. Blocks consist of :

- Block header
- Transactions list
- List of headers of ommers or uncles

The transaction list is simply a list of all transactions included in the block.  
Also, the list of headers of uncles is also included in the block.

# Blocks & Blockchain

## Block Header :

are the most critical and detailed components of an Ethereum block.

The header contains various elements

- **Parent hash:** This is Keccak 256-bit hash of the parent block's header.
- **Ommers hash:** This is the Keccak 256-bit hash of the list of ommers (or uncles) blocks included in the block.
- **The beneficiary:** This field contains the 160-bit address of the recipient that will receive the mining reward once the block is successfully mined.

# Blocks & Blockchain

## Block Header :

- **State Root:** This field contains Keccak 256-bit hash of the root node of the state. It is calculated once all transactions have been processed and finalized.
- **Transactions Root:** The transaction root is the Keccak 256-bit hash of the root node of the transaction trie. The transaction trie represents the list of transactions included in the block.
- **Difficulty:** The difficulty level of the current block.
- **Number:** Total number of all previous blocks; the genesis block is block zero.

# Blocks & Blockchain

## Block Header :

- **Receipts Root:** is the Keccak 256-bit hash of the root node of the transaction receipt trie. This trie is composed of receipts of all transactions included in the block. Transaction receipts are generated after each transaction is processed and contain useful post- transaction information.
- **Logs Bloom:** is a bloom filter that is composed of the logger address and log topics from the log entry of each transaction receipt of the included transaction list in the block.

# Blocks & Blockchain

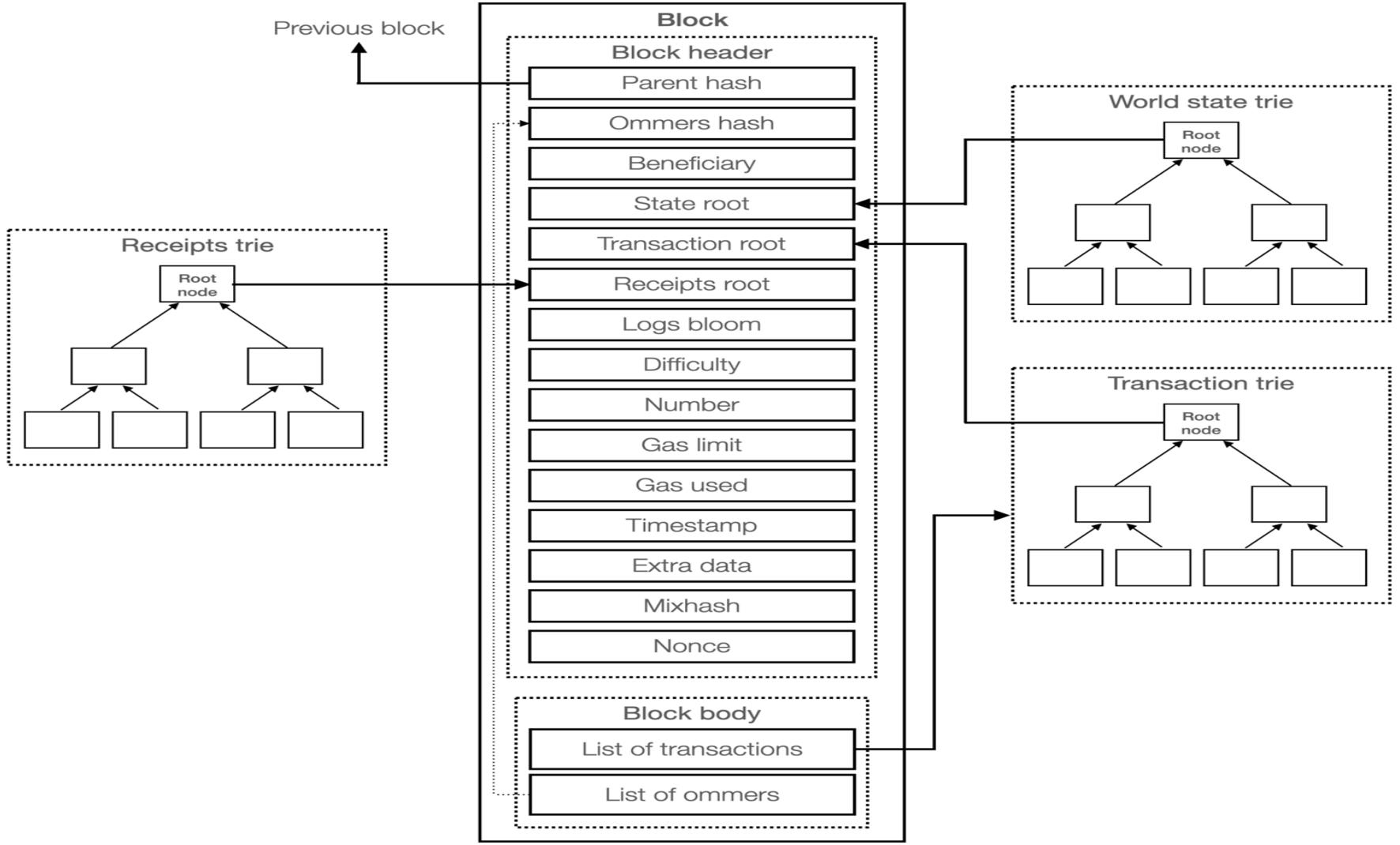
## Block Header :

- **Gas limit:** Value represents the limit set on the gas consumption per block.
- **Gas used:** Total gas consumed by the transactions included in the block.
- **Timestamp:** Epoch Unix time of the time of block initialization.
- **Extra data:** is used to store arbitrary data related to the block. Only up to 32 bytes are allowed in this field.

# Blocks & Blockchain

## Block Header :

- **Mixhash:** contains a 256-bit hash that, once combined with the nonce, is used to prove that adequate computational effort (PoW) has been spent in order to create this **block**.
- **Nonce:** is a 64-bit hash (a number) that is used to prove, in combination with the mixhash field, that adequate computational effort (PoW) has been spent in order to create this **block**.



# Genesis Block

# Element Description

**Timestamp : Jul-30-2015 03:26:13 PM +UTC**

**Transactions** : 8,893 transactions and 0 contract internal transactions in this BlockHash :

0xd4e56740f876aef8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3

## **Parent hash :**

# Genesis Block

**SHA-3 uncles :**

0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347

**Mined by :**

0x00 in 15 seconds

**Difficulty :** 17,179,869,184

**Total difficulty :** 17,179,869,184

**Size :** 540 bytes

**Gas used :** 0

# Genesis Block

**Nonce :** 0x00000000000000042

**Block reward :** 5 ETH

**Uncles reward :** 0

**Extra data :** In

hex(0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb  
7a38e1e50b1b82fa)

**Gas limit** 5,000

# Block Validation Mechanism

Ethereum block validation mechanism checks the following conditions:

- **If it is consistent with uncles and transactions.** This means that all uncles satisfy the property that they are indeed uncles, and also if the Proof of Work (PoW) for uncles is valid.
- **If the previous block (parent) exists and is valid.**
- **If the timestamp of the block is valid.** This means that the current block's timestamp must be higher than the parent block's timestamp. Also, it should be less than 15 minutes into the future.
- **If any of these checks fails, the block will be rejected.**

# Block Validation Mechanism

A list of errors for which the block can be rejected:

- The timestamp is older than the parent
- There are too many, or duplicate uncles
- The uncle is an ancestor, or the uncle's parent is not an ancestor
- There is non-positive difficulty
- There is an invalid mix digest or PoW

# Block Finalization

is a process that is run by miners to validate the contents of the block and apply rewards. It results in four steps being executed:

- **Ommers (uncles) validation.** In the case of mining, determine ommers. The validation process of the headers of stale blocks checks whether the header is valid and whether the relationship between the uncle and the current block satisfies the maximum depth of six blocks. A block can contain a maximum of two uncles.
- **Transaction validation.** In the case of mining, determine transactions. This process involves checking whether the total gas used in the block is equal to the final gas consumption after the final transaction, in other words, the cumulative gas used by the transactions included in the block.

# Block Finalization

- **Reward Application.** Apply rewards, which means updating the beneficiary's account with a reward balance. In Ethereum, a reward is also given to miners for stale blocks, which is 1/32 of the block reward. Uncles that are included in the blocks also receive 7/8 of the total block reward. The current block reward is 2 ether. A block can have a maximum of two uncles.
- **State and nonce validation.** Verify the state and block nonce. In the case of mining, compute a valid state and block nonce.

# Block Difficulty Mechanism

The block difficulty mechanism is represented by the formula below, which ensures that blocks are produced at a constant rate:

```
block_diff = parent_diff + parent_diff // 2048 *  
max(1 - (block_timestamp - parent_timestamp) // 10, -99) +  
int(2**((block.number // 100000) - 2))
```

The gas cost of a transaction can be calculated using this formula:

Total cost = gasUsed \* gasPrice

# Gas

- Gas is required to be paid for every operation performed on the Ethereum blockchain. This is a mechanism that ensures that infinite loops cannot cause the whole blockchain to stall due to the Turing-complete nature of the EVM.
- A transaction fee is charged as an amount of Ether and is taken from the account balance of the transaction originator.
- If the execution runs out of gas, everything is immediately rolled back; otherwise, if the execution is successful and some gas remains, then it is returned to the transaction originator.

# Gas

Each operation costs some gas;  
a high-level fee schedule of a  
few operations is shown as an example here:

Operation name	Gas cost
Stop	0
SHA3	30
SLOAD	800
Transaction	21000
Contract creation	32000

# Fee Schedule

Gas is charged in three scenarios as a prerequisite to the execution of an Operation:

- Computation of an operation
- For contract creation or message calls
- An increase in the use of memory

# Wallets & Client Software

**Clients** are full implementations of the Ethereum protocol, which support mining, account management, and wallet functions.

**Wallets** only store the public and private keys, provide essential account management, and interact with the blockchain for usually only payment (transfer of funds) purposes.

# Wallets & Client Software

## Geth

This is the official Go implementation of the Ethereum client.

The latest version is available at the following link:

<https://geth.ethereum.org/downloads/>.

## Eth

This is the C++ implementation of the Ethereum client.

Eth is available at the GitHub repository:

<https://github.com/ethereum/aleth>.

# Wallets & Client Software

## Parity

This implementation is built using Rust and developed by Parity technologies.

Parity can be downloaded from the following link: <https://www.parity.io/>

Note that Parity is now OpenEthereum. Henceforth.

## Trinity

Trinity is the implementation of the Ethereum protocol. It is written in Python.

Trinity can be downloaded from <https://github.com/ethereum/trinity>

The official website of Trinity client is <https://trinity.ethereum.org>

# Wallets & Client Software

There are three client synchronization types:

- **Full:** In this synchronization mode, the Geth client downloads the complete blockchain to its local node. This means that it gets all the block headers and block bodies and validates all transaction and blocks since the genesis block.

As of early 2020, the Ethereum blockchain size is roughly 210 GB, and downloading and maintaining that could be a problem. Usually, SSDs are recommended for full node, so that disk latency cannot cause processing delays.

# Wallets & Client Software

- **Fast:** The client downloads the full blockchain, but it retrieves and verifies only the previous 64 blocks from the current block. After this, it verifies the new blocks in full. It does not replay and verify all historic transactions since the genesis block; instead it only does the state downloads. This also reduces the on-disc size of the blockchain database quite significantly. This is the default mode of Geth client synchronization.
- **Light:** This is the quickest mode and only downloads and stores the current state trie. In this mode, the client does not download any historic blocks and only processes newer blocks

# Wallets & Client Software

## MetaMask

runs as a plugin or add-on in the web browser. It is available for the Chrome, Firefox, Opera, and Brave browsers. The key idea behind the development of MetaMask is to provide an interface with the Ethereum blockchain.

It allows efficient account management and connectivity to the Ethereum blockchain without running the Ethereum node software locally.

MetaMask allows connectivity to the Ethereum blockchain through the infrastructure available at Infura (<https://infura.io>). This allows users to interact with the blockchain without having to host any node locally.

# Wallets & Client Software

## Installation

MetaMask is available for download at <https://metamask.io>

Browse to <https://metamask.io/> where links to the relevant source are available to download the extension for your browser:

Home > Extensions > MetaMask



MetaMask

<https://metamask.io>

Featured

Add to Chrome

2,672

[Productivity](#)

10,000,000+ users

Overview

Privacy practices

Reviews

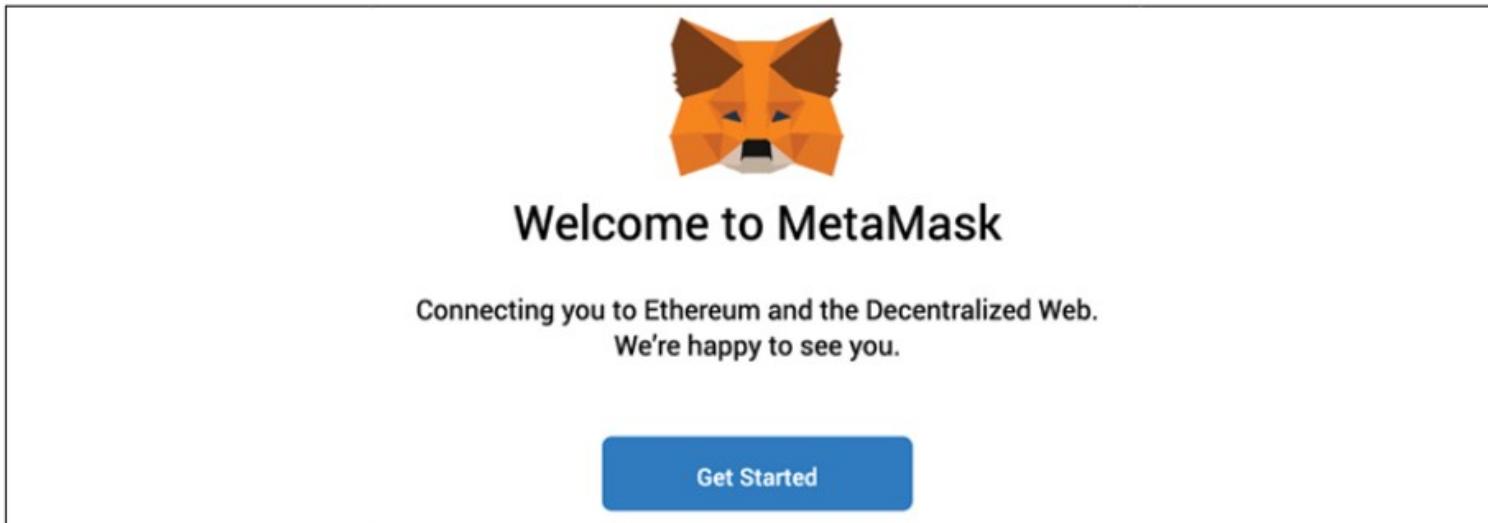
Support

Related

# Wallets & Client Software

Click the Add to Chrome button and it will install the extension for the browser.

It will install quickly, and if all goes well, you will see the following window:



# Wallets & Client Software

Click on Get Started, and it will show two options, either to import an already existing wallet using the seed, or to create a new wallet  
will select Create a Wallet here

New to MetaMask?



No, I already have a Secret Recovery Phrase

Import your existing wallet using a Secret Recovery Phrase

Import wallet



Yes, let's get set up!

This will create a new wallet and Secret Recovery Phrase

Create a Wallet

# Wallets & Client Software

## Create Password

New Password (min 8 chars)

\*\*\*\*\*

Confirm Password

\*\*\*\*\*



I have read and agree to the [Terms of Use](#)

Create

# Wallets & Client Software

## Secret Backup Phrase

Your secret backup phrase makes it easy to back up and restore your account.

**WARNING:** Never disclose your backup phrase. Anyone with this phrase can take your Ether forever.

buddy away super duty nephew  
gym still tube peace year pupil  
improve

### Tips:

Store this phrase in a password manager like 1Password.

Write this phrase on a piece of paper and store in a secure location. If you want even more security, write it down on multiple pieces of paper and store each in 2 - 3 different locations.

Memorize this phrase.

[Download this Secret Backup Phrase](#) and keep it stored safely on an external encrypted hard drive or storage medium.

[Remind me later](#)

[Next](#)

# Wallets & Client Software

Once everything has been completed you will see the following message:



## Congratulations

You passed the test - keep your seedphrase safe, it's your responsibility!

### Tips on storing it safely

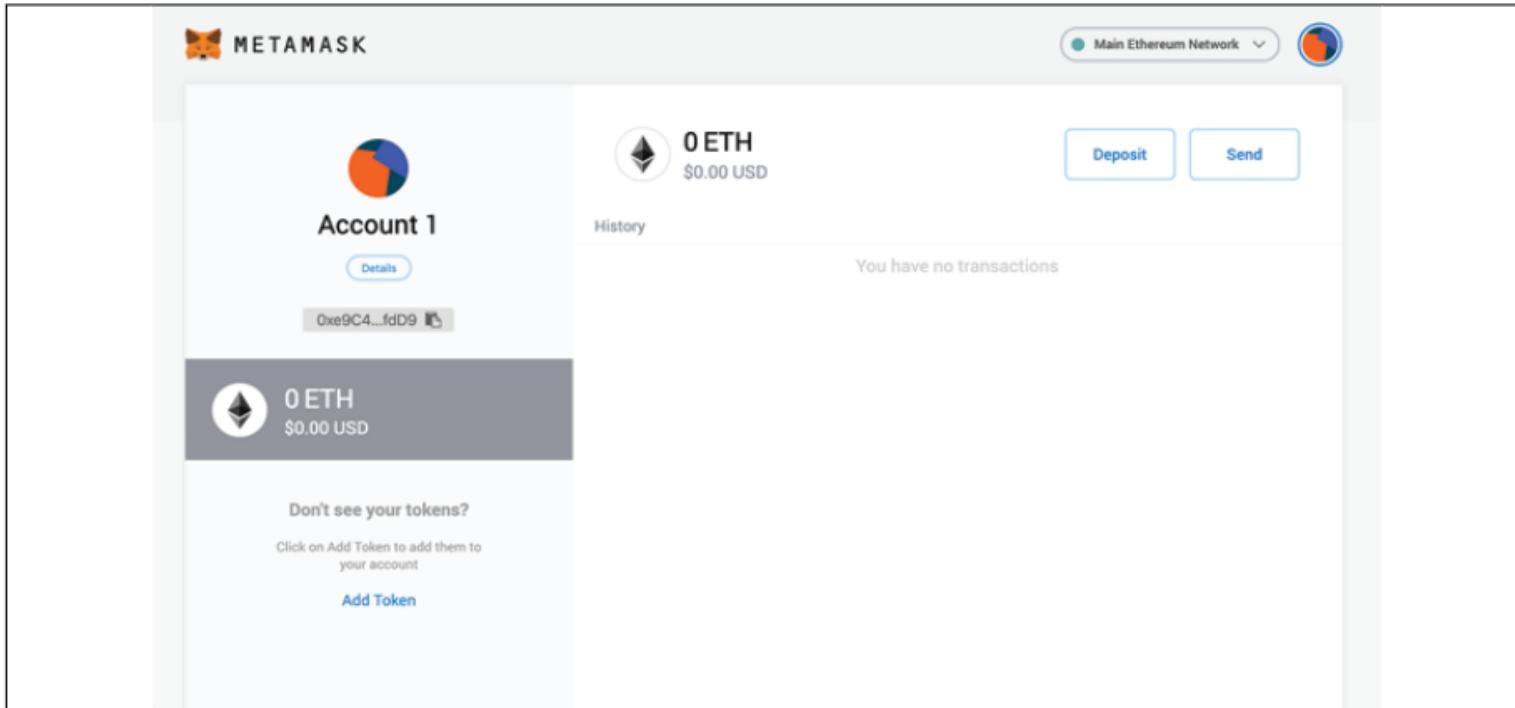
- Save a backup in multiple places.
- Never share the phrase with anyone.
- Be careful of phishing! MetaMask will never spontaneously ask for your seed phrase.
- If you need to back up your seed phrase again, you can find it in Settings -> Security.
- If you ever have questions or see something fishy, email support@metamask.io.

\*MetaMask cannot recover your seedphrase. [Learn more.](#)

All Done

# Wallets & Client Software

After clicking on the All Done button, the main MetaMask main view will open:



# Nodes & Miners

Mining is the process by which new blocks are selected via a consensus mechanism and added to the blockchain.

The process takes the following steps:

- It listens for the transactions broadcasted on the Ethereum network and determines the transactions to be processed.
- It determines stale ommer blocks and includes them in the blockchain.
- It updates the account balance with the reward earned from successfully mining the block.
- Finally, a valid state is computed and the block is finalized, which defines the result of all state transitions.

# Ethash

Ethash is the name of the PoW algorithm used in Ethereum.

Originally, this was proposed as the Dagger-Hashimoto algorithm, but much has changed since the first implementation, and the PoW algorithm has now evolved into what's known as Ethash.

Similar to Bitcoin, the core idea behind mining is to find a nonce (a random arbitrary number), which, once concatenated with the block header and hashed, results in a number that is lower than the current network difficulty level.

# Ethash

## Directed Acyclic Graph (DAG)

DAG is a large, pseudo-randomly generated dataset. This graph is represented as a matrix in the DAG file created during the Ethereum mining process. The Ethash algorithm expects the DAG as a two-dimensional array of 32-bit unsigned integers.

Mining can only start when DAG is completely generated the first time a mining node starts. This DAG is used as a seed by the algorithm called Ethash.

The Ethash algorithm requires a DAG file to work. A DAG file is generated every epoch, which is 30,000 blocks or roughly 6 days.

# Ethash

Ethash is the name of the Proof of Work algorithm used in Ethereum.

- First, the header from the previous block and a 32-bit random nonce is combined using Keccak-256.
- This produces a 128-bit structure called mix.
- mix determines which data is to be picked up from the DAG.
- Once the data is fetched from the DAG, it is "mixed" with the mix to produce the next mix, which is then again used to fetch data from the DAG and subsequently mixed. This process is repeated 64 times.

# Ethash

- Eventually, the 64th mix is run through a digest function to produce a 32-byte sequence.
- This sequence is compared with the difficulty target. If it is less than the difficulty target, the nonce is valid, and the PoW is solved.
- As a result, the block is mined. If not, then the algorithm repeats with a new nonce.

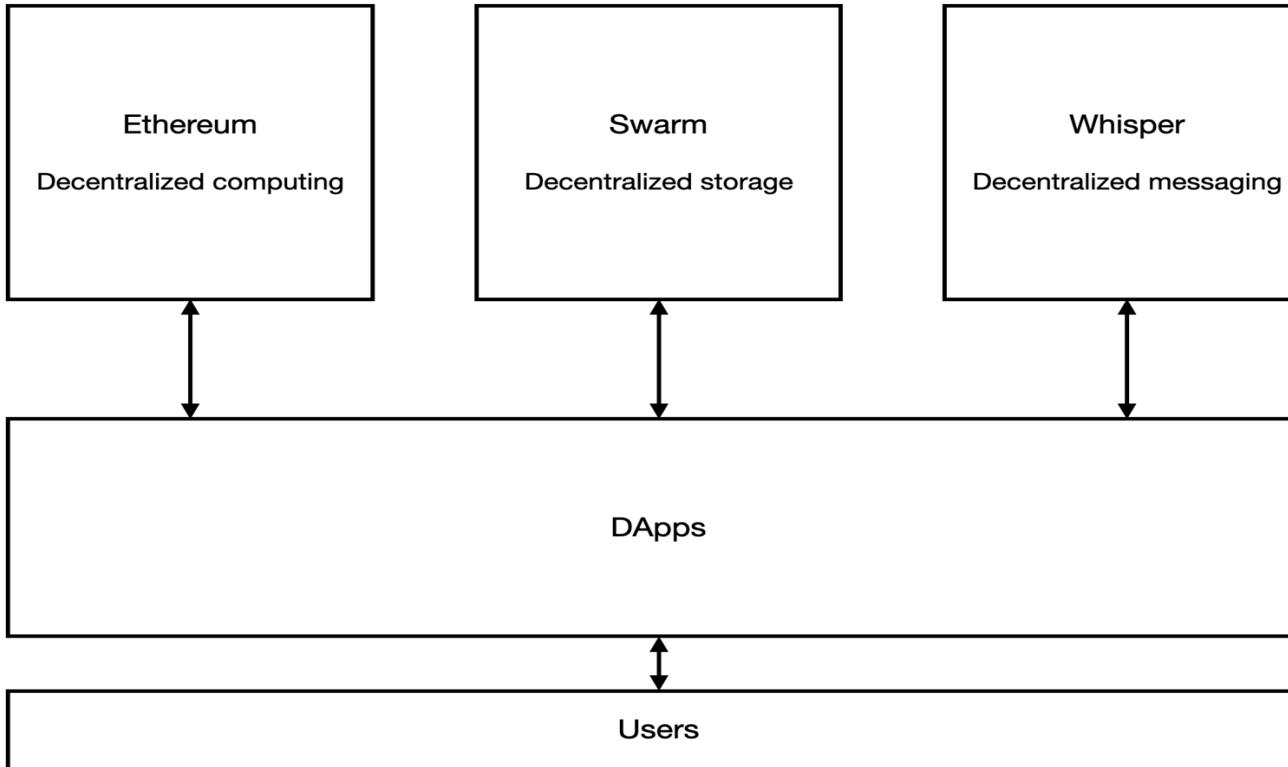
# Mining

- **CPU**  
Mining using a computer's built in CPU
- **GPU**  
Mining using graphical processing units, which provide better performance as compared to CPUs
- **ASICs**  
Specialized hardware—Application-Specific Integrated Circuit designed solely to run mining algorithms. These are currently the most efficient mining tools
- **Mining Pools**  
In mining pools, resources are shared between miners and rewards are split accordingly

# Supporting Protocols

- Various supporting protocols are available to assist the complete decentralized ecosystem.
- In addition to the contracts layer, which is the core blockchain layer, there are additional layers that need to be decentralized in order to achieve a complete decentralized ecosystem. This includes decentralized storage and decentralized messaging.
- Two main supporting protocols, Swarm and Whisper, are used to provide decentralized storage and messaging, in order to create a complete decentralized ecosystem.

# Supporting Protocols



# Programming Languages

- **Solidity** : is one of the high-level languages that has been developed for Ethereum. It uses JavaScript-like syntax to write code for smart contracts. Once the code is written, it is compiled into bytecode that's understandable by the EVM using the Solidity compiler called solc.
- **LLL** : is another language that is used to write smart contract code.
- **Serpent** : is a Python-like, high-level language that can be used to write smart contracts for Ethereum.
- **Vyper** : is a newer language that has been developed from scratch to achieve a secure, simple, and auditable language.

# Ethereum Development Environment

CHAPTER 13

# Ethereum Development Environment

There are various ways to perform development for Ethereum blockchain. We are going to learn the topics:

- Test networks
- Private network components
- Starting up the private network
- Mining on the private network
- Remix IDE
- MetaMask
- Using MetaMask and Remix IDE to deploy a smart contract

# Ethereum Development Environment

There are multiple ways to develop smart contracts on Ethereum:

- Develop and test Ethereum smart contracts either in a local private net or a simulated environment like Ganache.
- Then it can be deployed on a public testnet.
- After all the relevant tests are successful on a public testnet, the contracts can then be deployed to the public mainnet.

# Ethereum Development Environment

- There are new tools and frameworks available, like **Truffle** and **Ganache**, which make development and testing for Ethereum easier.
- Frameworks and tools make development easier, but hide most of the details that are useful for beginners to understand to build a strong foundation of knowledge.
- **Geth** is the leading client for Ethereum and the standard tool of choice
- Therefore, first we will demonstrate development using the **native tools** available in Ethereum,

# Test Networks

**testnets can play a vital role in testing the smart contracts before deploying on the mainnet.**

Ethereum has several testnets. A common testnet is called **Ropsten** and is used by developers or users as a test platform to test smart contracts and other blockchain-related proposals.

The following command will connect to the Ropsten network via geth client, which is a preconfigured proof of work (PoW) test network.

```
$ geth --testnet
```

# Test Networks

The screenshot shows the type of the network chosen and various other pieces of information regarding the blockchain download:

```
* - geth --testnet
INFO [06-30|19:43:31.395] Maximum peer count
INFO [06-30|19:43:31.423] Starting peer-to-peer node
INFO [06-30|19:43:31.424] Allocated trie memory caches
INFO [06-30|19:43:31.424] Allocated cache and file handles
INFO [06-30|19:43:34.583] Opened ancient database
INFO [06-30|19:43:34.521] Persisted trie from memory database
INFO [06-30|19:43:34.524] Initialised chain configuration
170000 Constantinople: 4230000 Petersburg: 4939394 Istanbul: 6485846, Muir Glacier: 7117117, Engine: ethash
INFO [06-30|19:43:34.524] Disk storage enabled for ethash caches
INFO [06-30|19:43:34.524] Disk storage enabled for ethash DAGs
INFO [06-30|19:43:34.525] Initialising Ethereum protocol
INFO [06-30|19:43:34.529] Loaded most recent local header
INFO [06-30|19:43:34.529] Loaded most recent local full block
INFO [06-30|19:43:34.529] Loaded most recent local fast block
INFO [06-30|19:43:34.532] Loaded local transaction journal
INFO [06-30|19:43:34.533] Regenerated local transaction journal
WARN [06-30|19:43:34.533] Switch sync mode from fast sync to full sync
INFO [06-30|19:43:34.610] New local node record
INFO [06-30|19:43:34.610] Started P2P networking
f2293b123812ce4984d425bf1f60ae23f6ea05@127.0.0.1:36303
INFO [06-30|19:43:34.619] IPC endpoint opened
INFO [06-30|19:43:37.081] New local node record
INFO [06-30|19:43:37.773] Mapped network port
INFO [06-30|19:43:38.573] Mapped network port
INFO [06-30|19:45:14.617] Block synchronisation started
INFO [06-30|19:45:20.080] Imported new chain segment
dirty=142.14KiB
ETH=50 lss=0 total=50
instance=Geth/v1.9.18-stable/darwin-and64/g01.13.6
clean=256.00MiB dirty=256.00MiB
database=/Users/drequinox/Library/Ethereum/testnet/geth/chaindata cache=612.00MiB handles=6120
database=/Users/drequinox/Library/Ethereum/testnet/geth/chaindata/ancient
nodes=355 size=50.67KiB time=1.280303ms gcnodes=0 gcsizes=0.00B gctime=<0s livenodes=1 livesize=0.00B
config={ChainID: 3 Homestead: 0 DAO: <nil> DAOSupport: true EIP150: 0 EIP155: 10 EIP158: 10 Byzantium:
dir=/Users/drequinox/Library/Ethereum/testnet/geth/ethash count=3
dir=/Users/drequinox/Library/Ethereum/testnet/geth/ethash count=2
versions="[64 63]" network=3 dbversion=7
number=80816 hash=1921f5...lbd680 td=8567527775346 age=3y7mo2w
number=80816 hash=1921f5...lbd680 td=8567527775346 age=3y7mo2w
number=80816 hash=1921f5...lbd680 td=8567527775346 age=3y7mo2w
transactions=0 dropped=0
transactions=0 accounts=0
seq=9 id=36466bba1d703662 ip=127.0.0.1 udp=30303 tcp=30303
self=enode://2aeadad8e005cc0db9cbad34fb3d03a1dea87d8fe3345dfb279196273812cc384617dd21ff79e1e9bd2671d32a
url=/Users/drequinox/Library/Ethereum/testnet/geth.ipc
seq=10 id=36466bba1d703662 ip=82.2.27.41 udp=30303 tcp=30303
proto=tcp extport=30303 intport=30303 interface="UPNP IGDbv2-IP1"
proto=udp extport=30303 intport=30303 interface="UPNP IGDbv2-IP1"
blocks=2 txs=38 mgas=3.957 elapsed=466.697ms mgasps=8.590 number=80818 hash=d0317a..f163f9 age=3y7mo2w d
```

# Test Networks

There are other ethereum test networks:

- Rinkeby and
- Görli (also referred to as Goerli).

Geth can be issued with a command-line flag to connect to the desired network:

```
--testnet Ropsten network: pre-configured proof-of-work test net  
--rinkeby Rinkeby network: pre-configured proof-of-authority tes  
--goerli Görli network: pre-configured proof-of-authority test r
```

# Private Blockchain Network

- A private blockchain network requires an invitation and must be validated by either the network starter or by a set of rules put in place by the network starter.
- Businesses who set up a private blockchain, will generally set up a permissioned network.
- This places restrictions on who is allowed to participate in the network, and only in certain transactions.
-

# Private Blockchain Network

## Components of a private network

- **Network ID** - The network ID can be any positive number except any number that is already in use by another Ethereum network. For example, 1 and 3 are in use by Ethereum mainnet and testnet (Ropsten), respectively. Network ID 786 normally chooses for private network
- **The genesis file** - The genesis file contains the necessary fields required for a custom genesis block. This is the first block in the network and does not point to any previous block.
- **Data directory** - This is the directory where the blockchain data for the private Ethereum network will be saved.

# Private Blockchain Network

The Ethereum network consists of four elements or layers:

- **Discovery** - is responsible for discovering other nodes on the network by using node discovery mechanism based on Kademlia protocol routing algorithm.
- **RLPx** - is a TCP-based transport protocol responsible for enabling secure communication between Ethereum nodes.
- **DevP2P** - DEVP2P (also called the wire protocol) is responsible for negotiating an application session between two nodes that have been discovered and have established a secure channel using RLPx.

# Private Blockchain Network

- **Application level sub-protocols** - Finally, the fourth element of the Ethereum network stack is where different Ethereum sub-protocols exist.

After discovering and establishing a secure transport channel and negotiating an application session, the nodes exchange messages using "capability protocols" or application sub-protocols. These capability protocols are responsible for different application-level communications.

# Private Blockchain Network

## Static nodes I

In the case of private networks, usually the connectivity is limited to a specific set of peers. In order to configure this list, the node IDs of these nodes are added to a configuration file called ***static-nodes.json***. This file is usually placed in the data directory of the Geth (Ethereum client) executable.

# Starting up Private Network

- 1- The first step is to create a directory named etherprivate under the home directory of the user.

```
$ mkdir ~/etherprivate
```

- 2- Place the *privategenesis.json* file shown earlier in The genesis file section. The initial command to start the private network is shown as

```
$ geth init ~/etherprivate/privategenesis.json --datadir ~/ether
```

```
Maximum peer count          ETH=50 LES=0 total=50
Allocated cache and file handles
Persisted trie from memory database
Successfully wrote genesis state
Allocated cache and file handles
Persisted trie from memory database
Successfully wrote genesis state
database=/Users/drequinox/etherprivate/geth/chaindata cache=16.00MiB handles=16
nodes=0 size=0.00B time=478.681µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
database=chaindata hash=6650a0..b5c158
database=/Users/drequinox/etherprivate/geth/lightchaindata cache=16.00MiB handles=16
nodes=0 size=0.00B time=14.268µs gcnodes=0 gcsize=0.00B gctime=0s livenodes=1 livesize=0.00B
database=lightchaindata hash=6650a0..b5c158
```

This output indicates that a genesis block has been created successfully.

# Starting up Private Network

In order for geth to start, the following command can be issued:

```
$ ./geth --datadir ~/etherprivate/ --networkid 786 --rpc --rpccap
```

This will produce the following output:

```
INFO [07-09|19:28:16.641] Maximum peer count          ETH=0 LES=0 total=60
INFO [07-09|19:28:16.670] Starting peer-to-peer node    instance=Geth/v1.9.10-stable/darwin-amd64/go1.13.6
INFO [07-09|19:28:16.670] Allocated trie memory caches   clean=256.00MiB dirty=256.00MiB
INFO [07-09|19:28:16.670] Allocated cache and file handles database=/Users/drequinox/etherprivate/geth/chaindata cache=512.00MiB
INFO [07-09|19:28:16.728] Opened ancient database        database=/Users/drequinox/etherprivate/geth/chaindata/ancient
INFO [07-09|19:28:16.733] Initialised chain configuration config={ChainID: 786 Homestead: 0 DAO: <nil> DAOSupport: false EIP-158: 0 Byzantium: <nil> Constantinople: <nil> Petersburg: <nil> Istanbul: <nil>, Muir Glacier: <nil>, Engine: unknown}
INFO [07-09|19:28:16.734] Disk storage enabled for ethash caches dir=/Users/drequinox/etherprivate/geth/ethash count=3
INFO [07-09|19:28:16.734] Disk storage enabled for ethash DAGs dir=/Users/drequinox/Library/Ethash count=2
INFO [07-09|19:28:16.735] Initialising Ethereum protocol  versions="(64 63)" network=786 dbversion=7
INFO [07-09|19:28:16.741] Loaded most recent local header number=5906 hash=b43696...004e81 td=2388807985 age=1mo2w3d
INFO [07-09|19:28:16.741] Loaded most recent local full block number=5906 hash=b43696...004e81 td=2388807985 age=1mo2w3d
INFO [07-09|19:28:16.741] Loaded most recent local fast block number=5906 hash=b43696...004e81 td=2388807985 age=1mo2w3d
INFO [07-09|19:28:16.742] Loaded local transaction journal transactions=0 dropped=0
INFO [07-09|19:28:16.743] Regenerated local transaction journal transactions=0 accounts=0
WARN [07-09|19:28:16.743] Switch sync mode from fast sync to full sync
INFO [07-09|19:28:16.865] New local node record          seq=53 id=d45806a5e5ac77c ip=127.0.0.1 udp=30303 tcp=30303
INFO [07-09|19:28:16.866] Started P2P networking        self=enode://f96578daac05df98e5895d83a86558ae7503abcc4b0d85075ae218a5874edd4ac90a0ff8371ed2e99ff9f0ee98da3b17d6adf4b5c6eed7fbba8@127.0.0.1:30303
INFO [07-09|19:28:16.869] IPC endpoint opened          url=/Users/drequinox/etherprivate/geth.ipc
INFO [07-09|19:28:16.869] HTTP endpoint opened        url=http://127.0.0.1:8545 cors=* vhosts=localhost
INFO [07-09|19:28:18.470] New local node record          seq=54 id=d45806a5e5ac77c ip=82.2.27.41 udp=30303 tcp=30303
INFO [07-09|19:28:19.391] Mapped network port           proto=tcp extport=30303 intport=30303 interface="UPNP IGDrv1-IP1"
INFO [07-09|19:28:19.593] Mapped network port           proto=udp extport=30303 intport=30303 interface="UPNP IGDrv1-IP1"
```

# Starting up Private Network

An important part of the output to note is the following log lines:

```
INFO [07-09|19:49:01.504] IPC endpoint opened  
url=/Users/drequinox/etherprivate/geth.ipc  
INFO [07-09|19:49:01.504] HTTP endpoint opened  
url=http://127.0.0.1:8545  
INFO [07-09|19:49:05.734] Etherbase automatically configured  
address=0xc9Bf76271b9E42E4bF7E1888e0F52351bDb65811
```

These lines show the information about the Inter-Process Communications (IPC) endpoint, HTTP endpoint, and Etherbase (coinbase) account information.

# Starting up Private Network

Geth can be attached via IPC to the running Geth client on a private network using the following command:

```
$ geth attach ~/etherprivate/geth.ipc
```

this will open the interactive JavaScript console for the running private net session:

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.9.10-stable/darwin-amd64/go1.13.6
coinbase: 0xc9bf76271b9e42e4bf7e1888e0f52351bdb65811
at block: 5906 (Sat, 23 May 2020 13:49:16 BST)
datadir: /Users/drequinox/etherprivate
modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

# Starting up Private Network

The following command creates a new account. In this context, the account will be created on the private network ID 786 :

```
> personal.newAccount("Password123")
"0xc9bf76271b9e42e4bf7e1888e0f52351bdb65811"
```

set this account as an etherbase/coinbase account so that the mining reward goes to this account. This can be achieved using the following command:

```
> miner.setEtherbase(personal.listAccounts[0])
true
```

# Starting up Private Network

**the etherbase account has no balance initially:**

```
> eth.getBalance(eth.coinbase).toNumber();  
0
```

# Mining on the Private Network

mining can start by simply issuing the following command:

```
> miner.start()  
true
```

the DAG generation process starts

```
INFO [01-25|13:39:53.143] Commit new mining work  
INFO [01-25|13:39:55.288] Generating DAG in progress  
INFO [01-25|13:39:55.712] Generating DAG in progress  
INFO [01-25|13:39:56.811] Generating DAG in progress  
INFO [01-25|13:39:57.927] Generating DAG in progress  
INFO [01-25|13:39:58.043] Generating DAG in progress  
INFO [01-25|13:39:58.213] Generating DAG in progress  
INFO [01-25|13:39:58.222] Generating DAG in progress  
INFO [01-25|13:39:59.772] Generating DAG in progress  
INFO [01-25|13:39:59.772] Generating DAG in progress  
INFO [01-25|13:40:12.861] Generating DAG in progress  
INFO [01-25|13:40:13.519] Generating DAG in progress  
INFO [01-25|13:40:15.899] Generating DAG in progress  
INFO [01-25|13:40:16.552] Generating DAG in progress  
INFO [01-25|13:40:17.958] Generating DAG in progress  
INFO [01-25|13:40:19.491] Generating DAG in progress  
INFO [01-25|13:40:20.962] Generating DAG in progress  
INFO [01-25|13:40:22.668] Generating DAG in progress  
INFO [01-25|13:40:24.638] Generating DAG in progress  
INFO [01-25|13:40:26.263] Generating DAG in progress  
INFO [01-25|13:40:27.737] Generating DAG in progress  
INFO [01-25|13:40:29.253] Generating DAG in progress  
INFO [01-25|13:40:30.765] Generating DAG in progress  
INFO [01-25|13:40:32.439] Generating DAG in progress  
INFO [01-25|13:40:33.949] Generating DAG in progress  
INFO [01-25|13:40:35.444] Generating DAG in progress  
INFO [01-25|13:40:36.561] Generating DAG in progress  
INFO [01-25|13:40:37.681] Generating DAG in progress  
INFO [01-25|13:40:38.798] Generating DAG in progress  
INFO [01-25|13:40:39.915] Generating DAG in progress  
INFO [01-25|13:40:41.032] Generating DAG in progress  
INFO [01-25|13:40:42.149] Generating DAG in progress  
INFO [01-25|13:40:43.266] Generating DAG in progress  
INFO [01-25|13:40:44.383] Generating DAG in progress  
INFO [01-25|13:40:45.500] Generating DAG in progress  
INFO [01-25|13:40:46.617] Generating DAG in progress  
INFO [01-25|13:40:47.734] Generating DAG in progress  
INFO [01-25|13:40:48.851] Generating DAG in progress  
INFO [01-25|13:40:49.968] Generating DAG in progress  
INFO [01-25|13:40:51.085] Generating DAG in progress  
INFO [01-25|13:40:52.202] Generating DAG in progress  
INFO [01-25|13:40:53.319] Generating DAG in progress  
INFO [01-25|13:40:54.436] Generating DAG in progress  
INFO [01-25|13:40:55.553] Generating DAG in progress  
INFO [01-25|13:40:56.670] Generating DAG in progress  
INFO [01-25|13:40:57.787] Generating DAG in progress  
INFO [01-25|13:40:58.904] Generating DAG in progress  
INFO [01-25|13:40:59.921] Generating DAG in progress  
INFO [01-25|13:40:59.921] sealhash=4be1cd_a80d43 uncles=0 txs=0 gas=0 fees=0 elapsed=11.009ms  
number=1 sealhash=4be1cd_a80d43 uncles=0 txs=0 gas=0 fees=0 elapsed=11.009ms  
epoch=0 percentage=0 elapsed=1.388s  
epoch=0 percentage=1 elapsed=3.893s  
epoch=0 percentage=2 elapsed=6.429s  
epoch=0 percentage=3 elapsed=9.047s  
epoch=0 percentage=4 elapsed=11.693s  
epoch=0 percentage=5 elapsed=13.483s  
epoch=0 percentage=6 elapsed=15.953s  
epoch=0 percentage=7 elapsed=18.242s  
epoch=0 percentage=8 elapsed=19.499s  
epoch=0 percentage=9 elapsed=21.279s  
epoch=0 percentage=10 elapsed=22.732s  
epoch=0 percentage=11 elapsed=24.138s  
epoch=0 percentage=12 elapsed=25.671s  
epoch=0 percentage=13 elapsed=27.142s  
epoch=0 percentage=14 elapsed=28.848s  
epoch=0 percentage=15 elapsed=30.810s  
epoch=0 percentage=16 elapsed=32.443s  
epoch=0 percentage=17 elapsed=33.917s  
epoch=0 percentage=18 elapsed=35.433s  
epoch=0 percentage=19 elapsed=36.945s  
epoch=0 percentage=20 elapsed=38.619s  
epoch=0 percentage=21 elapsed=40.129s  
epoch=0 percentage=22 elapsed=41.624s  
epoch=0 percentage=23 elapsed=42.996s
```

In the context of Ethereum Ethash PoW algorithm, DAG refers to Dagger, which is a memory-hard PoW algorithm

# Mining on the Private Network

It can be seen that blocks are being mined successfully with the mined potential block message:

```
INFO [01-25|13:58:07.405] Successfully sealed new block
number=96 sealhash=02334c...f691fe hash=75302b...a016d5 elapsed=1.33
INFO [01-25|13:58:07.405] ↘ mined potential block
number=96 hash=75302b...a016d5
INFO [01-25|13:58:07.405] Commit new mining work
number=97 sealhash=4d4e6d...2906a8 uncles=0 txs=0 gas=0 fees=0 elapsed=0.00
INFO [01-25|13:58:18.527] Successfully sealed new block
number=97 sealhash=4d4e6d...2906a8 hash=817c8f...8012f6 elapsed=11.1
INFO [01-25|13:58:18.529] Commit new mining work
number=98 sealhash=9e8370...008b58 uncles=0 txs=0 gas=0 fees=0 elapsed=0.00
INFO [01-25|13:58:18.529] ↘ mined potential block
number=97 hash=817c8f...8012f6
INFO [01-25|13:58:18.634] Successfully sealed new block
number=98 sealhash=9e8370...008b58 hash=caba0f...cc206e elapsed=105.
```

# Mining on the Private Network

Mining can be stopped using the following command:

```
> miner.stop()  
null
```

# Commands to Query Private Network

There are a few other commands that can be used to query the private Network.

query the current balance of Ether in wei

```
> eth.getBalance(eth.coinbase)  
55000000000000000000000000000000
```

Balance in ether

```
> web3.fromWei(eth.getBalance(eth.coinbase), "ether")  
550
```

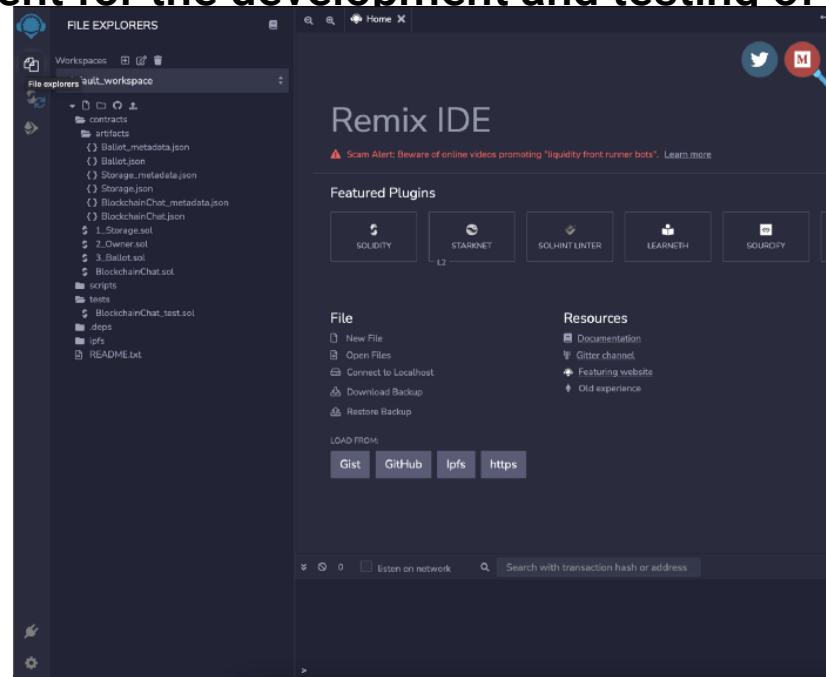
Create a new account. Note that Password123 is the password chosen as an example

```
> personal.newAccount("Password123")  
"0xd6e364a137e8f528ddbad2bb2356d124c9a08206"
```



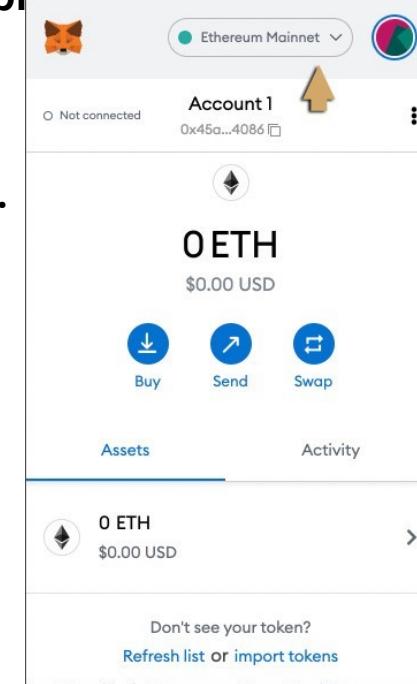
# Remix IDE

Remix is the web-based environment for the development and testing of contracts using Solidity.



# MetaMask

- MetaMask allows interaction with Ethereum blockchain via the Firefox and Chrome browsers.
- MetaMask also allows account management.



# Using MetaMask and Remix IDE to deploy a smart contract

**MetaMask** is an interface between the Ethereum blockchain and the web browser. It enables easy access to the blockchain and is quite useful for development activities.

**Remix IDE** provides an interface where contracts can be written in solidity and then deployed on to the blockchain.

# Using MetaMask and Remix IDE to deploy a smart contract

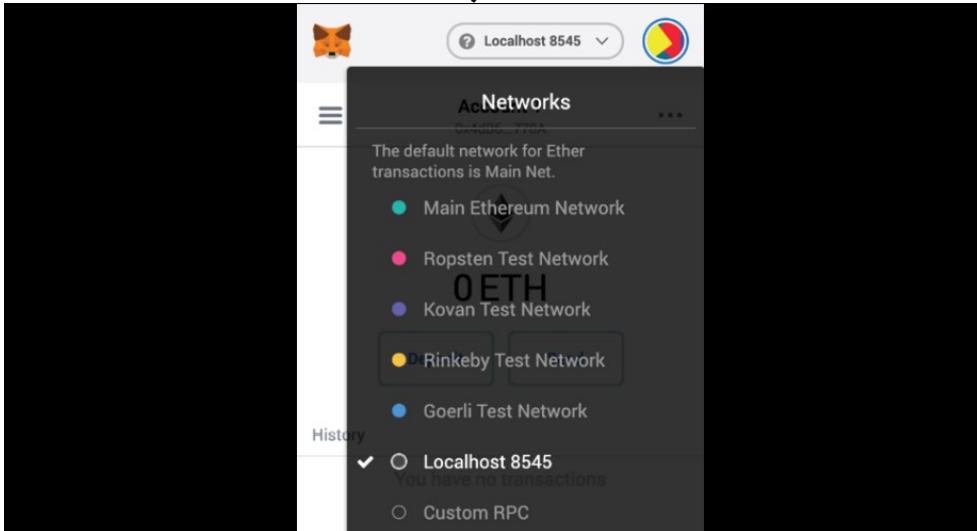
**MetaMask** is an interface between the Ethereum blockchain and the web browser. It enables easy access to the blockchain and is quite useful for development activities.

**Remix IDE** provides an interface where contracts can be written in solidity and then deployed on to the blockchain.

# Using MetaMask and Remix IDE to deploy a smart contract

Adding our local private network in MetaMask and then interact with it using Remix IDE.

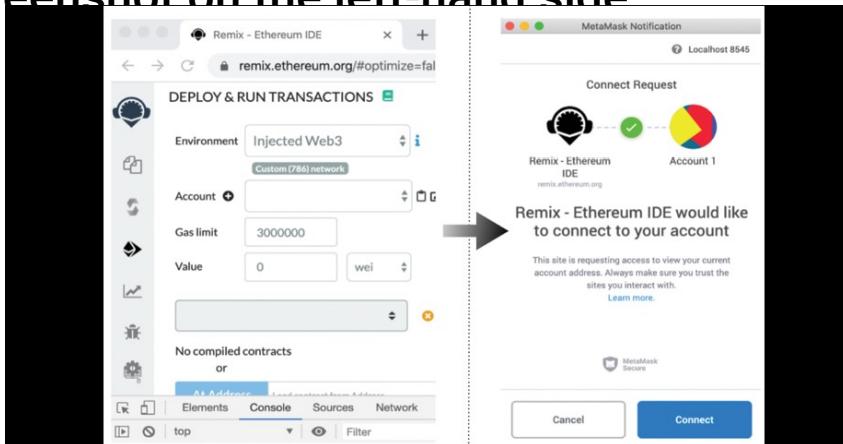
Open the Google Chrome web browser, where MetaMask is installed. Select Localhost 8545.



# Using MetaMask and Remix IDE to deploy a smart contract

Navigate to Remix IDE for Ethereum smart contract development on your browser at <https://remix.ethereum.org>.

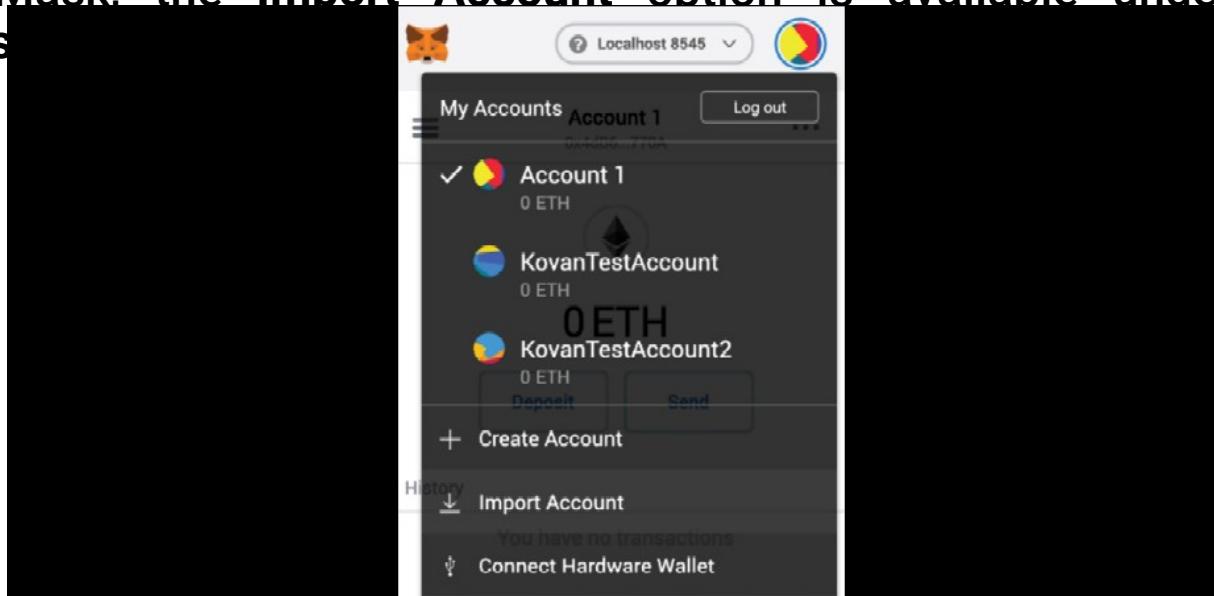
Notice the **DEPLOY & RUN TRANSACTIONS** option in the left-hand column. Choose **Injected Web3** as the Environment. This is shown in the following screenshot on the left-hand side.



# Using MetaMask and Remix IDE to deploy a smart contract

## Importing accounts into MetaMask using keystore files

In MetaMask, the **Import Account** option is available under the **My Accounts**

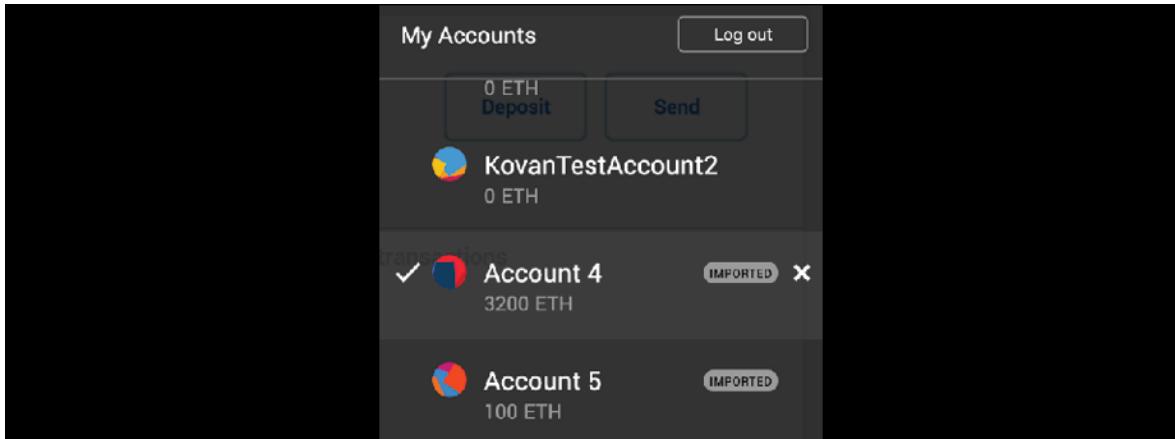


# Using MetaMask and Remix IDE to deploy a smart contract

## Importing accounts into MetaMask using keystore files

We can import these accounts into MetaMask using their associated keystore JSON files.

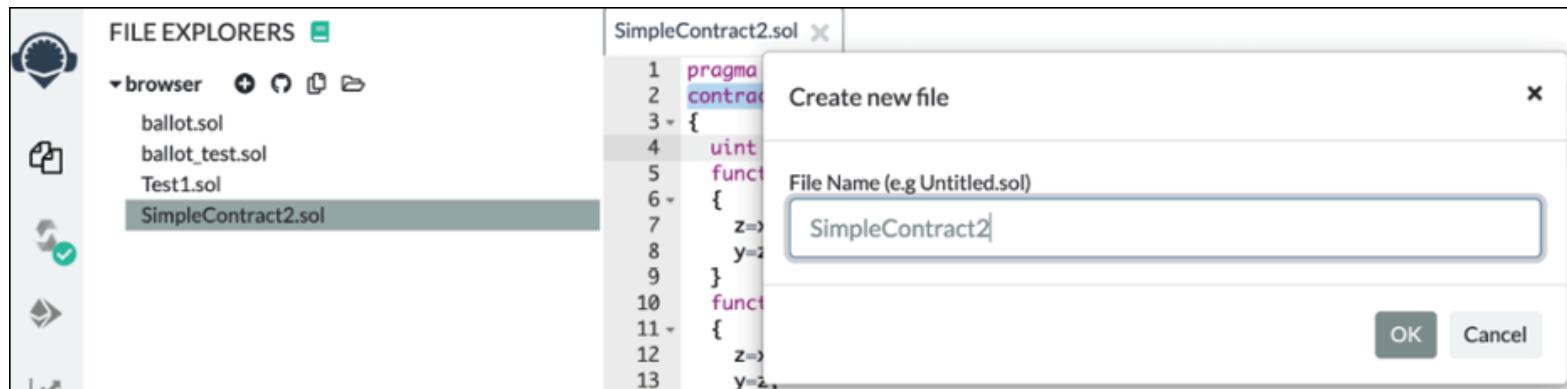
First, choose the JSON file from the **privatenet** keystore directory,



# Using MetaMask and Remix IDE to deploy a smart contract

## Deploying a contract with MetaMask

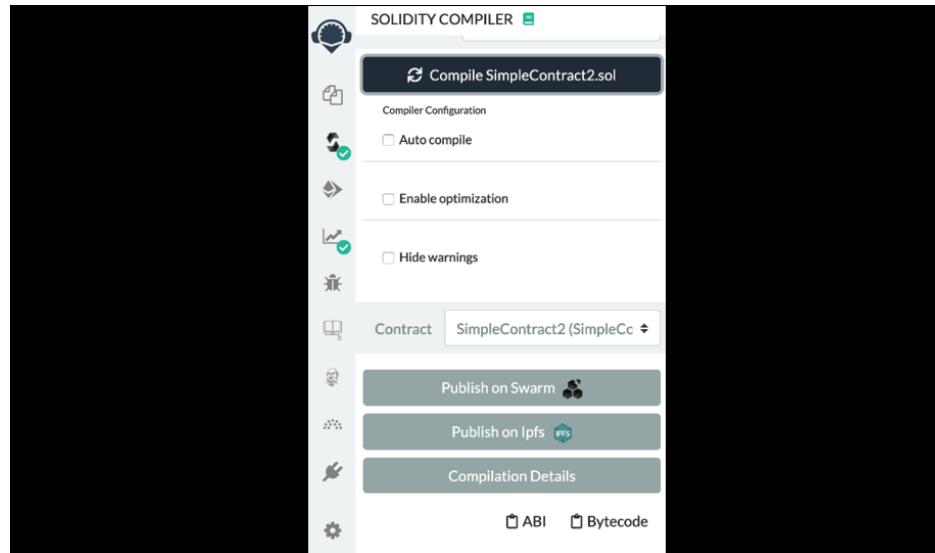
in Remix, we create a new file:



# Using MetaMask and Remix IDE to deploy a smart contract

## Deploying a contract with MetaMask

We then compile the smart contract by clicking on the **Compile** button



# Using MetaMask and Remix IDE to deploy a smart contract

## Deploying a contract with MetaMask

Once compiled successfully, we deploy the smart contract in the private net:

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons for interacting with the blockchain. The main area is divided into two sections: 'DEPLOY & RUN TRANSACTIONS' on the left and the Solidity code editor on the right.

In the 'DEPLOY & RUN TRANSACTIONS' section, the 'Environment' dropdown is set to 'Injected Web3'. The account selected is '0xc9b...65811(3199.99)', with a gas limit of '3000000' and a value of '0 wei'. Below these fields are buttons for 'Deploy' or 'At Address' and a field to 'Load contract from Address'. A dropdown for 'Transactions recorded:' shows '0'. At the bottom, it says 'Deployed Contracts' and 'Currently you have no contract instances to interact with.'

The code editor on the right contains the Solidity source code for 'SimpleContract2.sol':

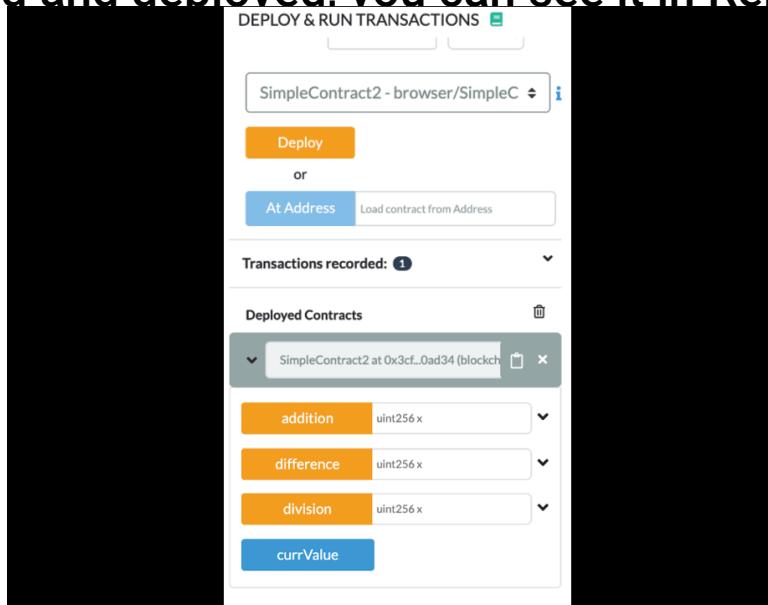
```
1 pragma solidity ^0.5.0;
2 contract SimpleContract2
3 {
4     uint z;
5     function addition(uint x) public returns (uint y)
6     {
7         z=x+5;
8         y=z;
9     }
10    function difference(uint x) public returns (uint y)
11    {
12        z=x-5;
13        y=z;
14    }
15    function division(uint x) public returns (uint y)
16    {
17        z=x/5;
18        y=z;
19    }
20
21    function currValue() public view returns (uint)
22    {
23        return z;
24    }
25
26 }
```

At the bottom of the interface, there are buttons for 'listen on network' and a search bar.

# Using MetaMask and Remix IDE to deploy a smart contract

## Deploying a contract with MetaMask

Once the contract is mined and deployed, you can see it in Remix, under the Deployed Contracts view:



# Using MetaMask and Remix IDE to deploy a smart contract

## Deploying a contract with MetaMask

Once compiled successfully, we deploy the smart contract in the private net:

The screenshot shows the Remix IDE interface. On the left, there's a sidebar with various icons for interacting with the blockchain. The main area is divided into two sections: 'DEPLOY & RUN TRANSACTIONS' on the left and the Solidity code editor on the right.

**DEPLOY & RUN TRANSACTIONS (Left):**

- Environment:** Injected Web3
- Account:** 0xc9b...65811 (3199.99)
- Gas limit:** 3000000
- Value:** 0 wei
- Contract Selection:** SimpleContract2 - browser/SimpleC
- Deployment Buttons:** Deploy or At Address
- Transactions Recorded:** 0
- Deployed Contracts:** Currently you have no contract instances to interact with.

**SimpleContract2.sol (Right):**

```
1 pragma solidity ^0.5.0;
2 contract SimpleContract2
3 {
4     uint z;
5     function addition(uint x) public returns (uint y)
6     {
7         z=x+5;
8         y=z;
9     }
10    function difference(uint x) public returns (uint y)
11    {
12        z=x-5;
13        y=z;
14    }
15    function division(uint x) public returns (uint y)
16    {
17        z=x/5;
18        y=z;
19    }
20
21    function currValue() public view returns (uint)
22    {
23        return z;
24    }
25
26 }
```

# Development Tools and Frameworks

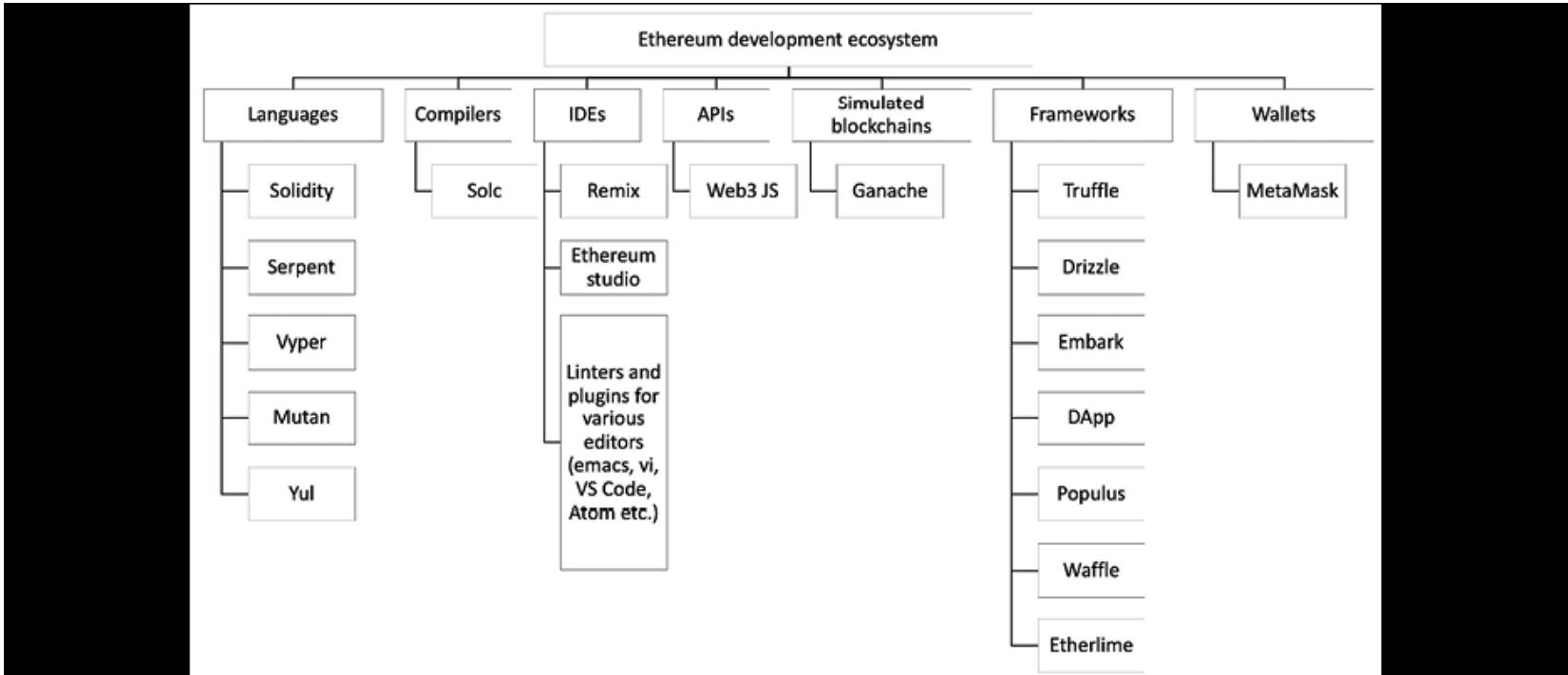
CHAPTER 14

# Development Tools & Frameworks

There are different methods of developing smart contracts for the Ethereum blockchain.

- Languages
- Compilers
- Tools and libraries
- Frameworks
- Contract development and deployment
- The layout of a Solidity source code file
- The Solidity language

# Taxonomy of Ethereum development ecosystem components



# Languages

Smart contracts can be programmed in a variety of languages for the Ethereum blockchain.

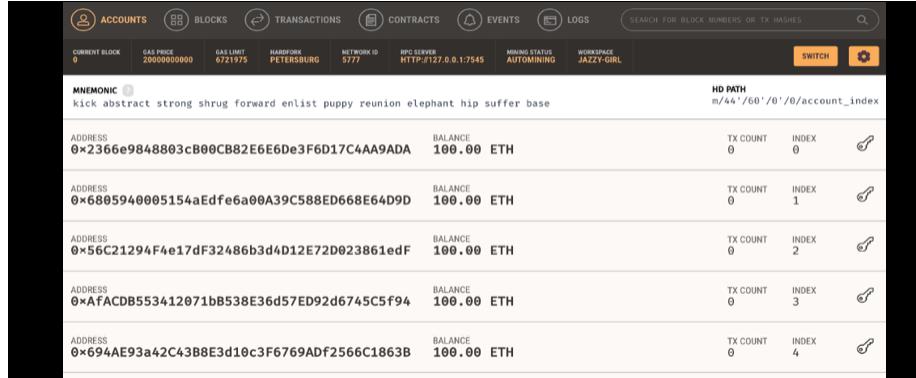
- Solidity
- Vyper
- Yul
- Mutan

# Tools & Libraries

**Node.js** - As Node.js is required for most of the tools and libraries.

**Ganache CLI** - Ganache CLI (formerly EthereumJS) comes in handy when quick testing is required and no testnet is available. It simulates the Ethereum geth client behavior and allows faster development testing.

**Ganache** - This is a simulated personal blockchain with a user-friendly GUI to view transactions, blocks, and relevant details.



# Frameworks

**Truffle** - Truffle provides contract compilation and linking along with an automated testing framework.

It also deploys the contracts to any privatenet, public, or testnet Ethereum blockchain.

**Drizzle** - Drizzle is a collection of frontend libraries that allows the easy development of web UIs for decentralized applications.

**Embark** - It is used for smart contract development, configuration, testing, and deployment.

**Brownie** - Brownie is a Python-based framework for Ethereum smart contract development and testing. It has the full support of Solidity and Vyper with relevant testing and debugging tools.

# Frameworks

**Waffle** - Waffle is a framework for smart contract development and testing. It is claimed to be faster than Truffle.

**Etherlime** - This framework allows DApp development, debugging, testing, and testing in Solidity and Vyper. It is based on Ether.Js

**OpenZeppelin** - This toolkit has a rich set of tools that allow easy smart contract development. It supports compiling, deploying, upgrading, and interacting with smart contracts.

# Contract Development & Deployment

**Writing smart contracts-** The writing step is concerned with writing the contract source code in Solidity. This can be done in any text editor.

**Testing smart contracts-** Testing is usually performed by automated means. However, manual functional testing can be performed as well by using the Remix IDE

**Deploying smart contracts** - Once the contract is verified, it can be deployed to a public testnet such as Ropsten and eventually to the live blockchain (mainnet).

# The Layout of a Solidity Source Code File

**Version pragma** - *pragma* can be used to specify the version of the compatible compiler as in the following example:

```
pragma solidity ^0.5.0
```

**Import** - Import in Solidity allows the importing of symbols from the existing Solidity files into the current global scope. This is similar to import statements available in JavaScript:

```
import "module-name";
```

**Comments** - Multiple-line comments are enclosed in /\* and \*/ , whereas single-line comments start with // .

.

# The Solidity language

- Solidity is a domain-specific language of choice for programming contracts in Ethereum.
- It is a statically typed language, which means that variable type checking in Solidity is carried out at compile time. This is beneficial in the sense that any validation and checking is completed at compile time and certain types of bugs
- Solidity is also called a contract-oriented language. In Solidity, contracts are equivalent to the concept of classes in other object-oriented programming languages.

# The Solidity language

- **Variables** - memory locations that hold values in a program:
  - Local variables - Variables whose values are permanently stored in a contract storage.
  - Global variables - Variables whose values are present till function is executing.
  - State variables - Special variables exists in the global namespace used to get information about the blockchain.

# The Solidity language

- **Data types**- we use various data types like character, wide character, integer, floating point, double floating point, boolean, etc. Solidity has two categories of data types:
  - value types - Value types are variables that are always passed by a value.
  - reference types - Reference types store the address of the memory location where the value is stored.

# The Solidity language

- **Value types-** :
  - Boolean
  - Integers
  - Address
    - Members are Balance, Send, Call functions, Array value types (fixed-size and dynamically sized byte arrays)
  - Literals (represent a fixed value)
    - Integer literals
    - String literals
    - Hexadecimal literals
    - Enums:

# The Solidity language

- **Reference types -**
  - Arrays - Arrays represent a contiguous set of elements of the same size and type laid out at a memory location.
  - Structs - These constructs can be used to group a set of dissimilar data types under a logical group. These can be used to define new custom types
  - Mappings - Mappings are used for a key-to-value mapping. This is a way to associate a value with a key.

# The Solidity language

**Control structures** - The control structures available in the Solidity language are

- **if...else ,**
- **do ,**
- **while ,**
- **for ,**
- **break ,**
- **continue ,**
- **and return .**

# The Solidity language

## Events -

- Events in Solidity can be used to log certain events in EVM logs.
- These logs are stored on the blockchain in transaction logs.

## Inheritance -

- Inheritance is supported in Solidity. The `is` keyword is used to derive a contract from another contract.

## Libraries -

- The key idea behind libraries is code reusability. They are similar to contracts and act as base contracts to the calling contracts.

# The Solidity language

**Functions-** A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again.

- **Internal functions** can be used only within the context of the current contract.
- **External functions** can be called via external function calls.

# The Solidity language

**Error handling-** Solidity provides various functions for error handling. By default, in Solidity, whenever an error occurs, the state does not change and reverts back to the original state.

Some constructs and convenience functions that are available for error handling

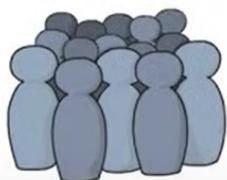
- Assert
- Require
- Revert
- Try/Catch
- Throw

# ETHEREUM 2.0

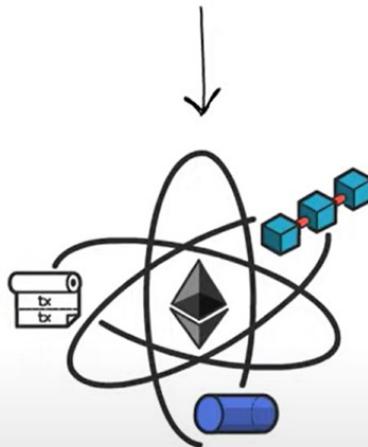
Set of integrated updates

# Ethereum 2.0

(ETH2)



More  
Scalable



More  
Secure



More  
Sustainable

# 1. Scalability



**15 TX/s**

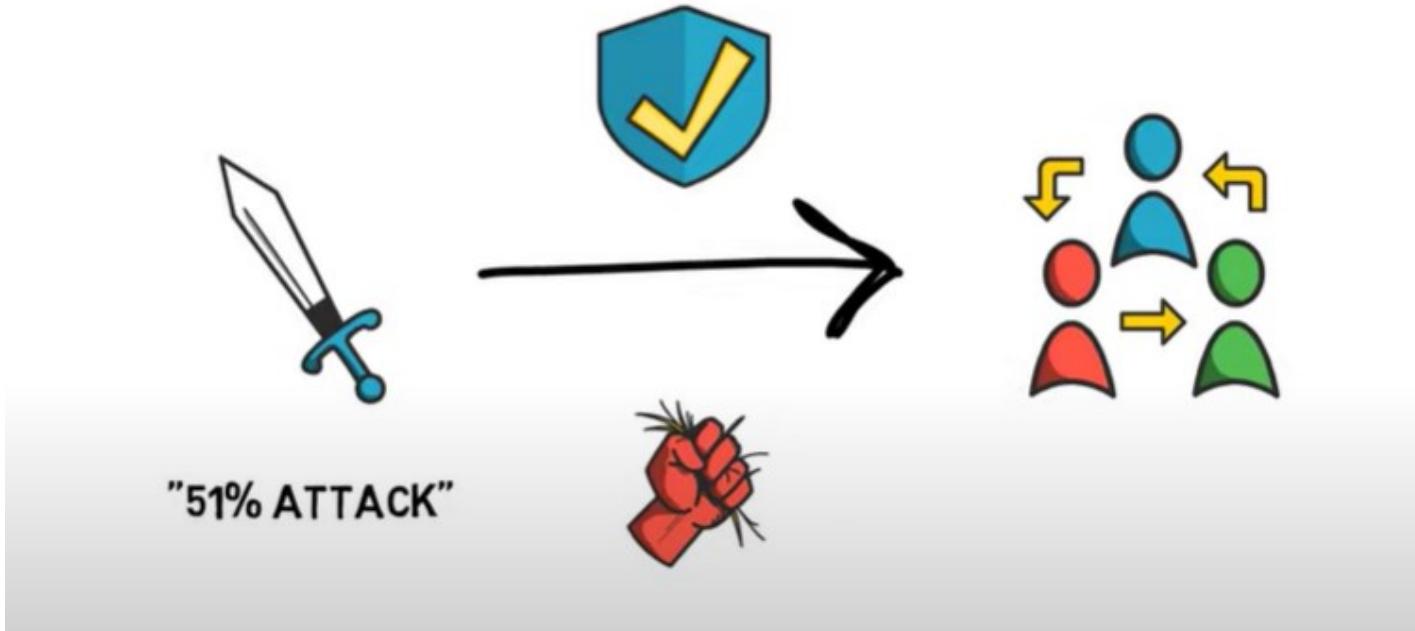


**1000s TX/s** ↑



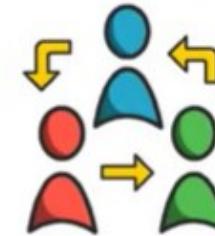
But not at the cost of increasing the size of the node

## 2. Security



Eth 2.0 aims to increase the security of the network

### 3. Sustainability



**PROOF OF WORK**



**PROOF OF STAKE**

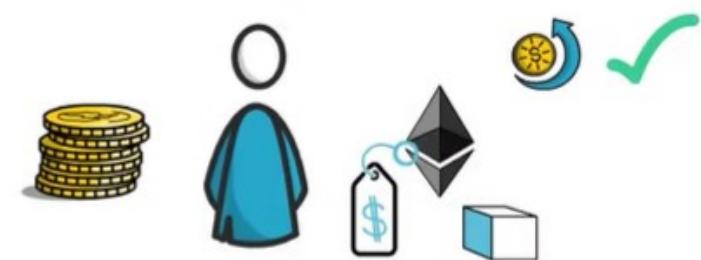


# POW Vs POS

Miners Invest Resources



VALIDATOR



Stakers get incentives of validating Transaction

## PROOF OF WORK



## PROOF OF STAKE



Well Known



Battle Tested



No Miners

# Goals of ETH 2.0

## SUSTAINABILITY



POS

## SECURITY



Stalking  
Penalty

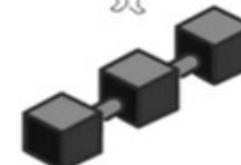
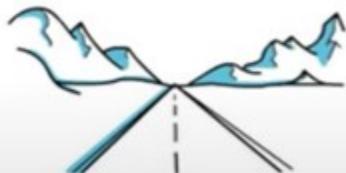
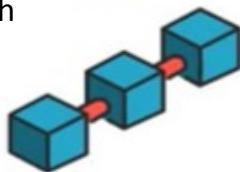
## SCALABILITY



Sharding

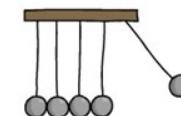
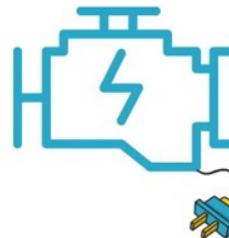
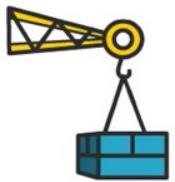
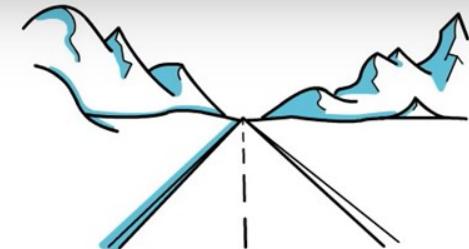
# The Beacon Chain

Runs a new POS  
network parallel  
to current Eth  
chain

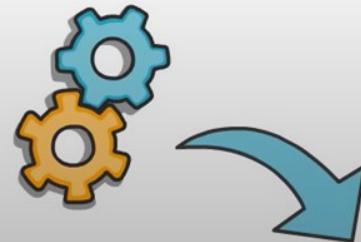


# The Beacon Chain



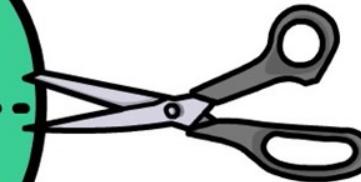


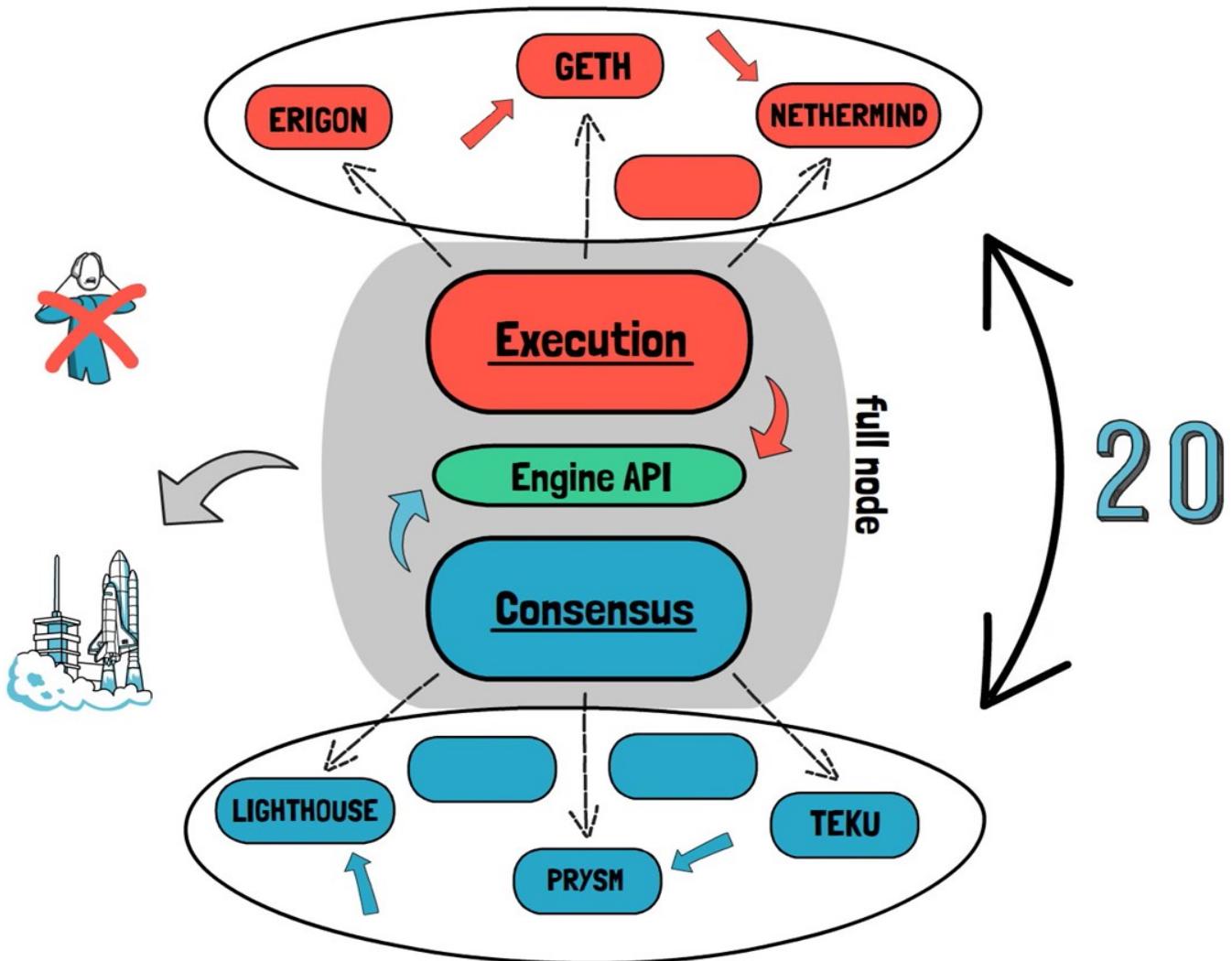
1.5 years



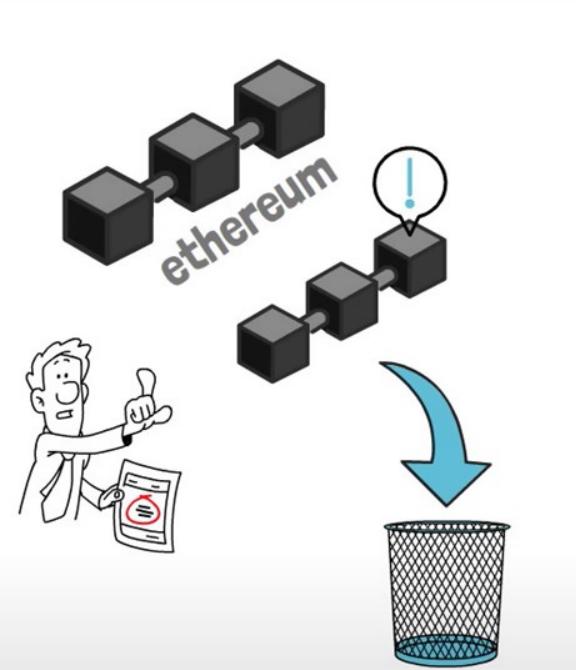


execution (EVM)  
-----  
consensus (PoW)

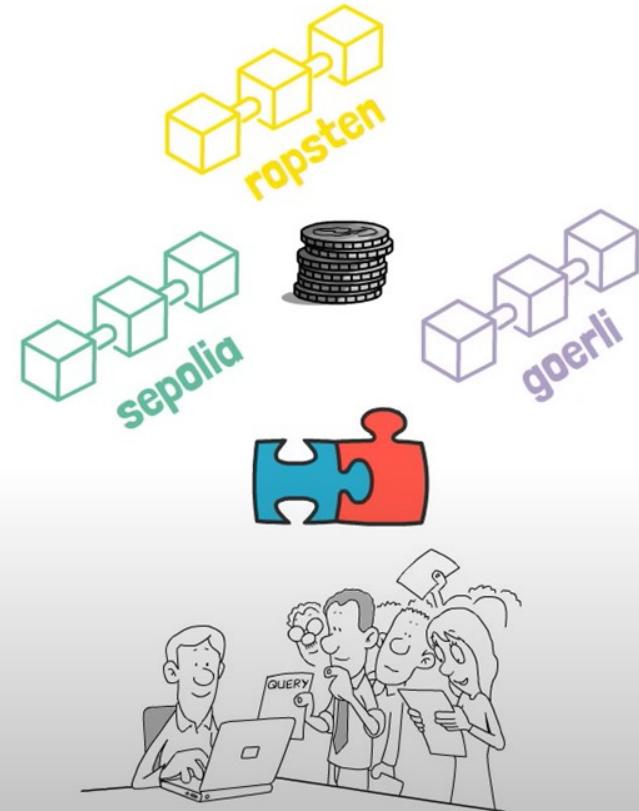
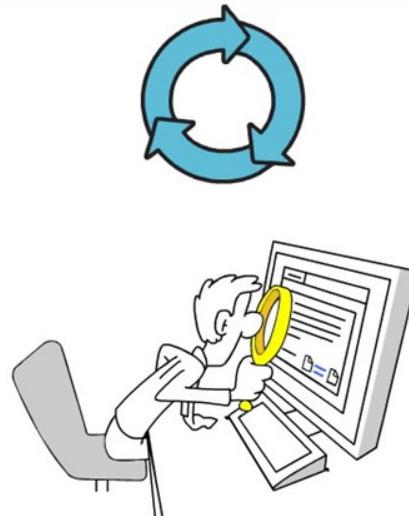




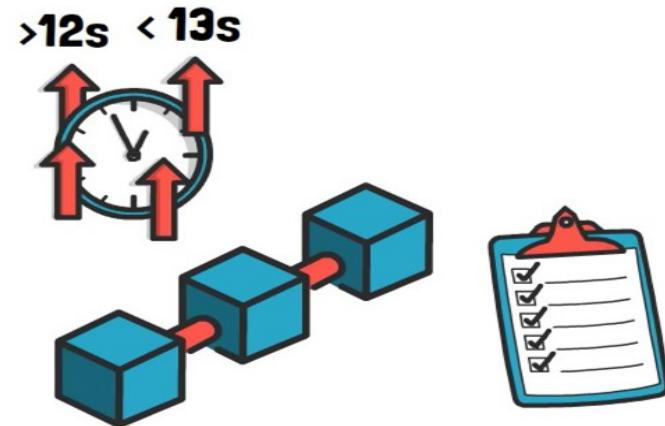
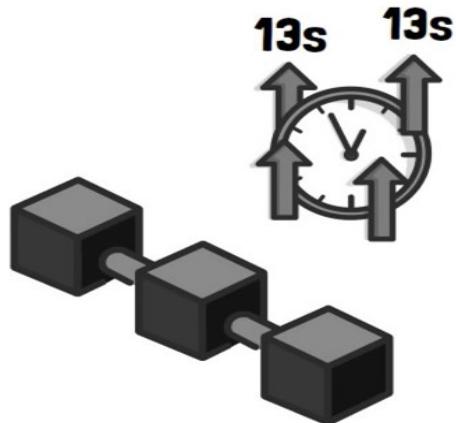
## SHADOW FORKS



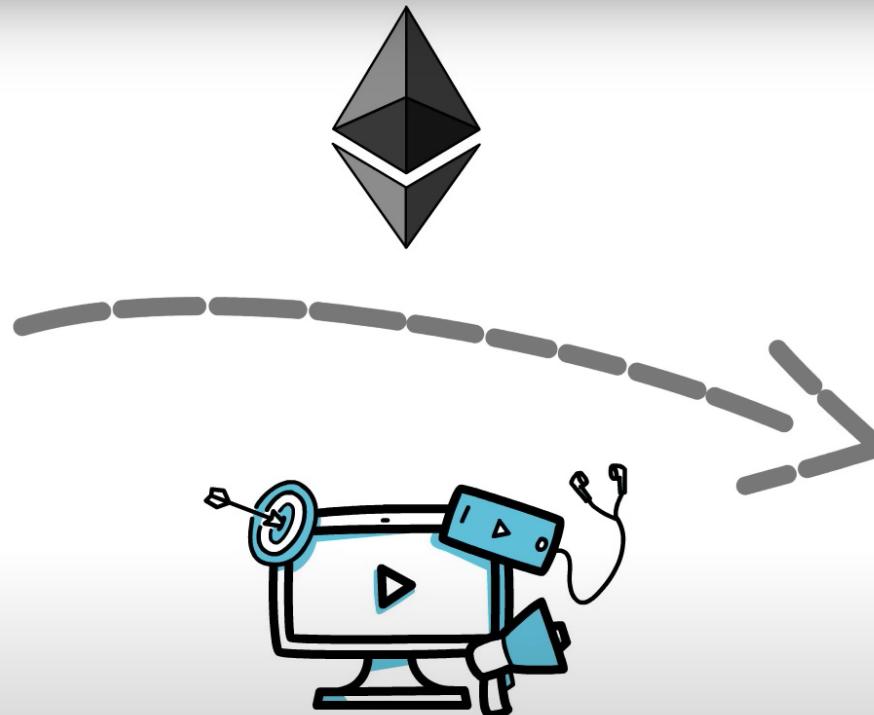
## TESTNETS



# Slightly faster transaction speed & Faster confirmation

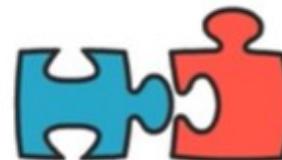
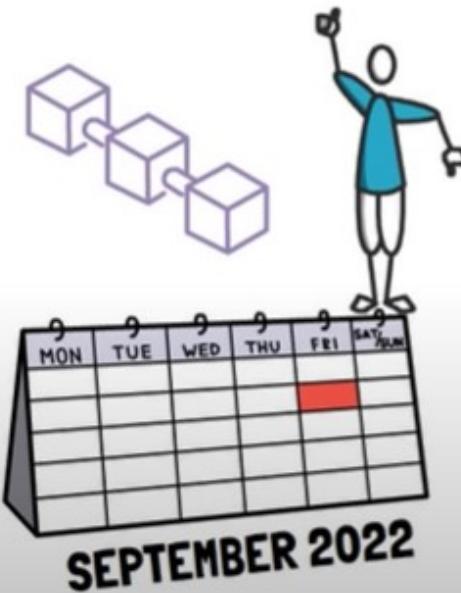


# Eth will be deflationary

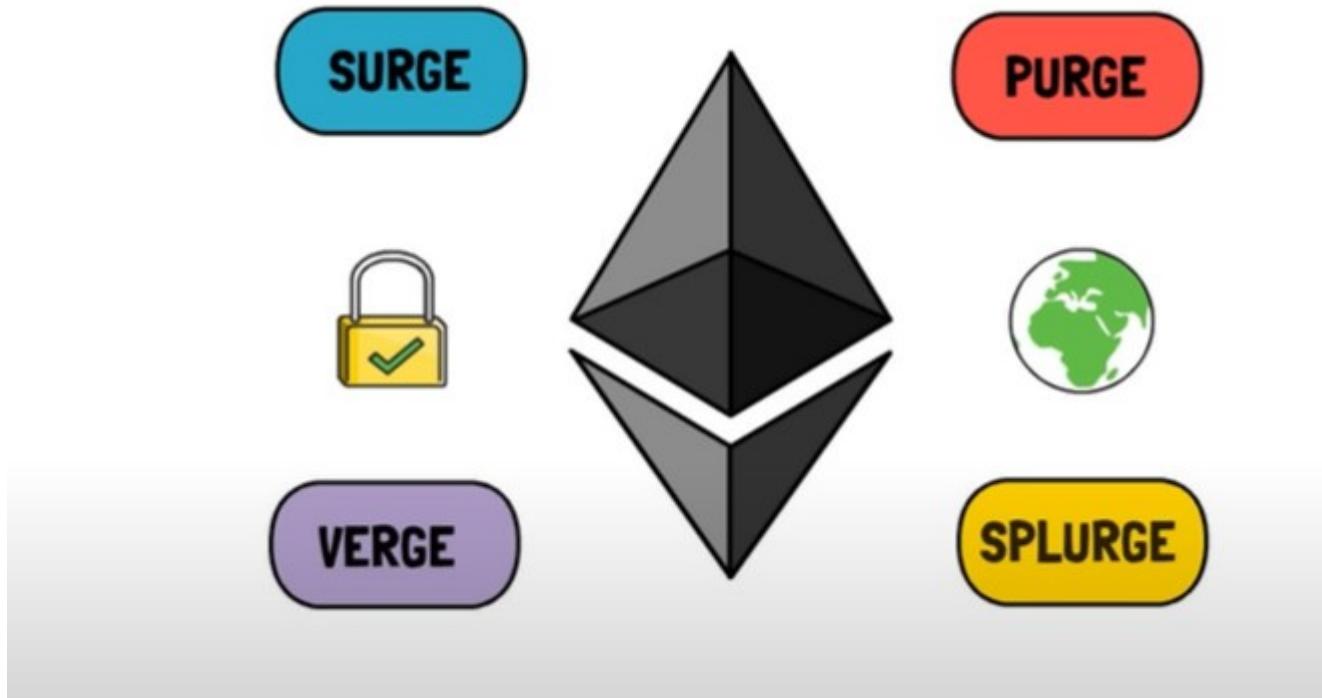


**"deflationary asset"**

# The Merge



# Future upgrades



# Learning TypeScript

Type-safe JavaScript

# From JavaScript to TypeScript

Chapter 1

# Vanilla JavaScript Pitfalls

- **Costly Freedom**

As the number of files grows in the project of JavaScript, you can only have vague ideas on how to call the functions.

```
function paintPainting(painter, painting) {  
    return painter  
        .prepare()  
        .paint(painting, painter.ownMaterials)  
        .finish();  
}
```

You might even make a lucky guess that painting is a string.

# Vanilla JavaScript Pitfalls

- **Loose Documentation**
  - There exists nothing in the JavaScript language specification to formalize description about code purpose.
  - Developers use JSDoc but it has key issues that often make it unpleasant to use in a large codebase
  - Maintaining JSDoc comments across a dozen files doesn't take up too much time, but across hundreds or even thousands of constantly updating files can be a real chore.

# Vanilla JavaScript Pitfalls

- **Weaker Developer Tooling**
  - Because JavaScript doesn't provide built-in ways to identify types.
  - It can be difficult to automate large changes to or gain insights about a codebase.

# TypeScript

- TypeScript was created internally at Microsoft in the early 2010s then released and open sourced in 2012.
- TypeScript is often described as a “superset of JavaScript” or “JavaScript with types.”

# TypeScript

## What is TypeScript

- Programming language - that includes all the existing JavaScript syntax, plus new TypeScript-specific syntax for defining and using types
- Type checker - It lets you know if it thinks anything is set up incorrectly
- Compiler - A program that runs the type checker, reports any issues, then outputs the equivalent JavaScript code
- Language service - A program that uses the type checker to tell editors such as VS Code how to provide helpful utilities to developers

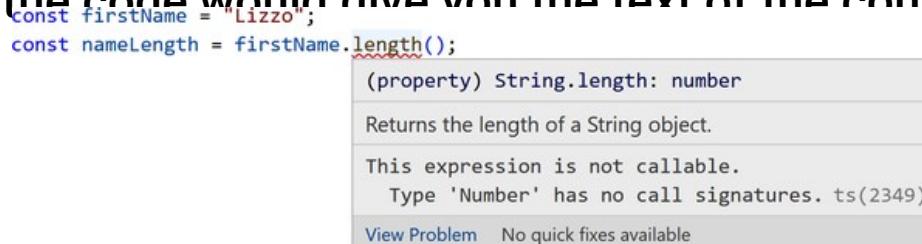
# Getting Started in the TypeScript Playground

The code is written in normal JavaScript syntax. If you tried to run that code in JavaScript, it would ~~crash!~~

```
const firstName = "Georgia";
const nameLength = firstName.length();
//                                     ~~~~~
// This expression is not callable.
```

If you were to run the TypeScript type checker on this code, it would use its knowledge that the length property of a string is a number—not a function

Hovering over the code would give you the text of the complaint



# Getting Started in the TypeScript Playground

## Freedom Through Restriction

- TypeScript allows us to specify what types of values may be provided for parameters and variables.
- If you change the number of required parameters for a function, TypeScript will let you know if you forget to update a place that calls the function.

# Getting Started in the TypeScript Playground

## Freedom Through Restriction

- **sayMyName** was changed from taking in two parameters to taking one parameter, but the call to it with two strings wasn't updated and so is triggering a TypeScript complaint:
- That code would run without crashing in JavaScript, but its output would be different from expected (it wouldn't include "Knowles"):

```
// Previously: sayMyName(firstName, lastName) { ...  
function sayMyName(fullName) {  
    console.log(`You acting kind of shady, ain't callin' me ${fullName}`);  
}  
  
sayMyName("Beyoncé", "Knowles");  
// ~~~~~  
// Expected 1 argument, but got 2.
```

# Getting Started in the TypeScript Playground

## Precise Documentation

a TypeScript version of the `paintPainting` function from earlier.

```
interface Painter {  
    finish(): boolean;  
    ownMaterials: Material[];  
    paint(painting: string, materials: Material[]): boolean;  
}  
  
function paintPainting(painter: Painter, painting: string): boolean { /* ... */ }
```

A TypeScript developer reading this code for the first time could understand that `painter` has at least three properties.

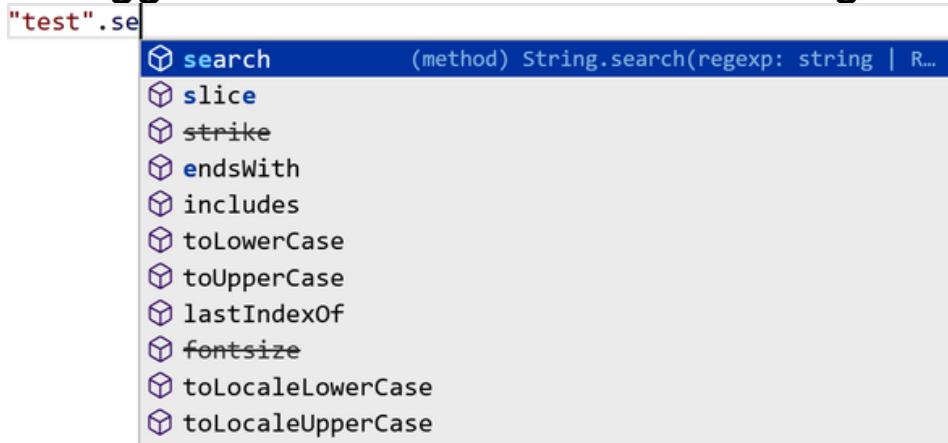
TypeScript provides an excellent, enforced system for describing how objects look.

# Getting Started in the TypeScript Playground

## Stronger Developer Tooling

TypeScript allows editors such as **VS Code** to gain much deeper insights into your code.

TypeScript can suggest all the members of the strings



# Getting Started in the TypeScript Playground

## Stronger Developer Tooling

When you add TypeScript's type checker for understanding code, it can give you these useful suggestions even for code you've written.

```
interface Painter {
    finish(): boolean;
    ownMaterials: Material[];
    paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean
    painter.|
```



The screenshot shows a code editor with the following TypeScript code:

```
interface Painter {
    finish(): boolean;
    ownMaterials: Material[];
    paint(painting: string, materials: Material[]): boolean;
}

function paintPainting(painter: Painter, painting: string): boolean
    painter.|
```

A tooltip is displayed over the variable `painter`, listing the available methods:

- `finish` (method) `Painter.finish(): boolean`
- `ownMaterials`
- `paint`

# Getting Started in the TypeScript Playground

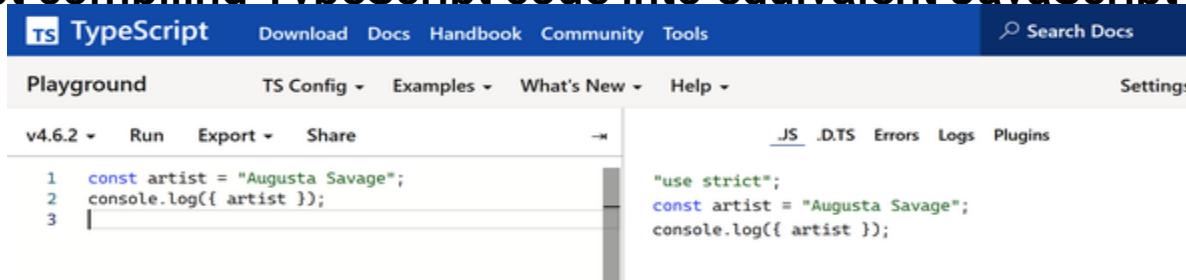
## Compiling Syntax

TypeScript's compiler allows us to input TypeScript syntax, have it type checked, and get the equivalent JavaScript emitted.

### TypeScript Code

```
const artist = "Augusta Savage";
console.log({ artist });
```

TypeScript compiling TypeScript code into equivalent JavaScript



The screenshot shows the TypeScript Playground interface. At the top, there's a navigation bar with the TypeScript logo, 'TypeScript' text, and links for 'Download', 'Docs', 'Handbook', 'Community', and 'Tools'. To the right is a search bar with the placeholder 'Search Docs'. Below the navigation is a secondary menu with 'Playground' selected, followed by 'TS Config', 'Examples', 'What's New', 'Help', and 'Settings'. Under the 'Playground' tab, there are dropdowns for 'v4.6.2', 'Run', 'Export', and 'Share', along with a 'JS' button and tabs for '.DTS', 'Errors', 'Logs', and 'Plugins'. The main area has two panes: an input pane on the left containing the TypeScript code, and an output pane on the right containing the generated JavaScript code. The input pane has lines 1, 2, and 3, with line 3 being active. The output pane shows the equivalent JavaScript code: 'use strict';, 'const artist = "Augusta Savage";', and 'console.log({ artist });'.

```
1 const artist = "Augusta Savage";
2 console.log({ artist });
3 |
```

```
"use strict";
const artist = "Augusta Savage";
console.log({ artist });
```

# Getting Started Locally

install the latest version of TypeScript globally

```
npm i -g typescript
```

run TypeScript on the command line with the tsc (TypeScript Compiler) command. Try it with the --version flag to make sure it's set up properly:

```
tsc --version
```

Output

```
C:\>tsc --version
Version 4.8.2
```

# Getting Started Locally

## Running Locally

- Create a folder somewhere on your computer and run this command to create a new `tsconfig.json` configuration file:

```
tsc --init
```

- A `tsconfig.json` file declares the settings that TypeScript uses when analyzing your code.
- Create a file named `index.ts` with the following contents:

```
console.log("Hello World");
```

- run `tsc` and provide it the name of that `index.ts` file:

```
tsc index.ts
```

# What TypeScript Is Not

Let's discuss the limitations of TypeScript !

## A Remedy for Bad Code

- TypeScript helps you structure your JavaScript, but other than enforcing type safety, it doesn't enforce any opinions on what that structure should look like.

# What TypeScript Is Not

## Extensions to JavaScript (Mostly)

- TypeScript does not try to change how JavaScript works at all.
- TypeScript's design goals explicitly state that it should:
  - Align with current and future ECMAScript proposals
  - Preserve runtime behavior of all JavaScript code

# What TypeScript Is Not

## Slower Than JavaScript

- TypeScript is slower than JavaScript. That claim is generally inaccurate and misleading.
- The only changes TypeScript makes to code are if you ask it to compile your code down to earlier versions of JavaScript to support older runtime environments such as Internet Explorer 11.
- Browsers and Node.js, will run it.

# What TypeScript Is Not

## Finished Evolving

- The TypeScript language is constantly receiving bug fixes and feature additions to match the ever-shifting needs of the web community.
- The current version of the TypeScript is `C:\>tsc --version`  
`Version 4.8.2`

# The Type System

Chapter 2

# What's in a Type?

- A “type” is a description of what a JavaScript value **shape** might be.
- “**shape**” means which properties and methods exist on a value.
- TypeScript understands the type of the value to be
  - one of the seven basic primitives:
    1. `null; // null`
    2. `undefined; // undefined`
    3. `true; // boolean`
    4. `"Louise"; // string`
    5. `1337; // number`
    6. `1337n; // bigint`
    7. `Symbol("Franklin"); // symbol`

# What's in a Type?

- If you hover your mouse over the variable's name. The resultant popover will include the name of the primitive,

```
2
3           let singer: string
4
5   let singer = "Ella Fitzgerald";
6
```

- TypeScript knows that the ternary expression always results in a string, so the `bestSong` variable is a string:

```
let bestSong: string
let bestSong = Math.random() > 0.5
  ? "Chain of Fools"
  : "Respect";
```

# What's in a Type?

## Type Systems

A type system is the set of rules for how a programming language understands what types the constructs in a program may have

```
let firstName = "Whitney";
firstName.length();
// ~~~~~
// This expression is not callable.
// Type 'Number' has no call signatures
```

TypeScript came to that complaint by, in order:

1. Reading in the code and understanding there to be a variable named `firstName`
2. Concluding that `firstName` is of type `string` because its initial value is a string, "Whitney"
3. Seeing that the code is trying to access a `.length` member of `firstName` and call it like a function
4. Complaining that the `.length` member of a `string` is a `number`, not a function (it can't be called like a function)

# What's in a Type?

## Kinds of Errors

While writing TypeScript, the two kinds of “errors” you’ll come across most frequently are:

### Syntax

Blocking TypeScript from being converted to JavaScript

```
let let wat;  
//      ~~~  
// Error: ',' expected.
```

### Type

Type errors occur when your syntax is valid but the TypeScript type checker has detected an error with the program’s types

```
console.blub("Nothing is worth more than laughter.");  
//      ~~~~~  
// Error: Property 'blub' does not exist on type 'Console'.  
.
```

# Assignability

TypeScript is fine with later assigning a different value of the same type to a variable.

If a variable is, say, initially assigned a string value, later assigning it another string would be fine:

```
let firstName = "Carole";
firstName = "Joan";
```

If TypeScript sees an assignment of a different type, it will give us a type error.

```
let lastName = "King";
lastName = true;
// Error: Type 'boolean' is not assignable to type 'string'.
```

# Assignability

## Understanding Assignability Errors

when we wrote

`lastName = true` in the previous snippet,

we were trying to assign the value of `true`—type `boolean`—to the recipient variable `lastName`—type `string`.

# Type Annotations

- Sometimes a variable doesn't have an initial value for TypeScript to read.
- It'll consider the variable by default to be implicitly the **any type**: indicating that it could be anything in the world.

```
let rocker; // Type: any

rocker = "Joan Jett"; // Type: string
rocker.toUpperCase(); // Ok

rocker = 19.58; // Type: number
rocker.toPrecision(1); // Ok

rocker.toUpperCase();
// ~~~~~
// Error: 'toUpperCase' does not exist on type 'number'.
```

# Type Annotations

- TypeScript provides a syntax for declaring the type of a variable without having to assign it an initial value, called a *type annotation*.
- A type annotation is placed after the name of a variable and includes a colon followed by the name of a type.

```
let rocker: string;  
rocker = "Joan Jett";
```

- These type annotations exist only for TypeScript—they don't affect the runtime code and are not valid JavaScript syntax.

# Type Annotations

## Unnecessary Type Annotations

The following : string type annotation is redundant because TypeScript could already infer that firstName be of type string:

```
let firstName: string = "Tina";
//           ~~~~~ Does not change the type system...
```

Many developers generally prefer not to add type annotations on variables where the type annotations wouldn't change anything.

# Type Annotations

## Type Shapes

- TypeScript also knows what member properties should exist on objects.
- If you attempt to access a property of a variable, TypeScript will make sure that property is known to exist on that variable's type.

Suppose we declare a rapper variable of type string. Later on, when we use that rapper variable, operations that only work on strings are allowed:

```
let rapper = "Queen Latifah";
rapper.length; // ok
```

# Type Annotations

## Modules

The JavaScript programming language did not include a specification for how files can share code between each other until relatively recently in its history.

### Module

A file with a top-level export or import

### Script

Any file that is not a module

# Type Annotations

## Modules

- Anything declared in a module file will be available only in that file unless an explicit export statement in that file exports it.
- A variable declared in one module with the same name as a variable declared in another file won't be considered a naming conflict (unless one file imports the other file's variable).

```
// a.ts
export const shared = "Cher";

// b.ts
export const shared = "Cher";
```

# Type Annotations

## Modules

- **c.ts** file causes a type error because it has a naming conflict between an imported shared and its own value:

```
// c.ts
import { shared } from "./a";
//           ~~~~~
// Error: Import declaration conflicts with local declaration of 'shared'.


export const shared = "Cher";
//           ~~~~~
// Error: Individual declarations in merged declaration
// 'shared' must be all exported or all local.
```

# Type Annotations

## Modules

- If a file is a script, all scripts have access to its contents.
- That means variables declared in a script file cannot have the same name as variables declared in other script files

```
// a.ts
const shared = "Cher";
// ~~~~~
// Cannot redeclare block-scoped variable 'shared'.
```

```
// b.ts
const shared = "Cher";
// ~~~~~
// Cannot redeclare block-scoped variable 'shared'.
```

The `a.ts` and `b.ts` files are considered scripts because they do not have module-style `export` or `import` statements.

That means their variables of the same name conflict with each other as if they were declared in the same file:

# Type Annotations

## Modules

if you need a file to be a module without an `export` or `import` statement, you can add an `export {};` somewhere in the file to force it to be a module:

```
// a.ts and b.ts
const shared = "Cher"; // Ok

export {};
```

# Unions and Literals

Chapter 3

# Union Types

- Take this mathematician variable:

```
let mathematician = Math.random() > 0.5  
? undefined  
: "Mark Goldberg";
```

What type is mathematician?

mathematician can be either undefined or string. This kind of “either or” type is called a union.

- handle code cases where we don't know exactly which type a value is, but do know it's one of two or more options.
- TypeScript represents union types using the | (pipe) operator between the possible values or constituents

```
let mathematician: string | undefined  
  
let mathematician = Math.random() > 0.5  
? undefined  
: "Mark Goldberg";
```

# Union Types

## Declaring Union Types

- Union types are an example of a situation when it might be useful to give an explicit type annotation for a variable even though it has an initial value.

```
let thinker: string | null = null;

if (Math.random() > 0.5) {
  thinker = "Susanne Langer"; // Ok
}
```

- `thinker` starts off `null` but is known to potentially contain a `string` instead.

Giving it an explicit `string | null` type annotation means TypeScript will allow it to be assigned values of type `string`:

# Union Types

## Union Properties

- TypeScript will only allow you to access member properties that exist on all possible types in the union.
- It will give you a type-checking error if you try to access a type that doesn't exist on all possible types.

# Union Types

## Union Properties

### Example

```
let physicist = Math.random() > 0.5
  ? "Marie Curie"
  : 84;

physicist.toString(); // Ok

physicist.toUpperCase();
// ~~~~~
// Error: Property 'toUpperCase' does not exist on type 'string | number'.
// Property 'toUpperCase' does not exist on type 'number'.

physicist.toFixed();
// ~~~
// Error: Property 'toFixed' does not exist on type 'string | number'.
// Property 'toFixed' does not exist on type 'string'.
```

physicist is of type `number | string`. While `.toString()` exists in both types and is allowed to be used, (common properties)

`.toUpperCase()` and `.toFixed()` are not because `.toUpperCase()` is missing on the `number` type and `.toFixed()` is missing on the `string` type:

# Narrowing

- Narrowing is when TypeScript infers from your code that a value is of a more specific type than what it was defined, declared, or previously inferred as.
- A logical check that can be used to narrow types is called a **type guard**.

# Narrowing

## Assignment Narrowing

If you directly assign a value to a variable, TypeScript will narrow the variable's type to that value's type.

```
let admiral: number | string;  
  
admiral = "Grace Hopper";  
  
admiral.toUpperCase(); // Ok: string  
  
admiral.toFixed();  
// ~~~~~  
// Error: Property 'toFixed' does not exist on type 'string'.
```

admiral variable is declared initially as a number | string, but after being assigned the value "Grace Hopper", TypeScript knows it must be a string:

# Narrowing

## Conditional Checks

**if statement** checking the variable for being equal to a known value.

```
// Type of scientist: number | string
let scientist = Math.random() > 0.5
  ? "Rosalind Franklin"
  : 51;

if (scientist === "Rosalind Franklin") {
  // Type of scientist: string
  scientist.toUpperCase(); // Ok
}

// Type of scientist: number | string
scientist.toUpperCase();
// ~~~~~
// Error: Property 'toUpperCase' does not exist on type 'string / number'.
// Property 'toUpperCase' does not exist on type 'number'.
```

TypeScript is smart enough to understand that inside the body of that if statement, the variable must be the same type as the known value:

# Narrowing

## Typeof Checks

TypeScript also recognizes the `typeof` operator in narrowing down variable types.

```
let researcher = Math.random() > 0.5
  ? "Rosalind Franklin"
  : 51;

if (typeof researcher === "string") {
  researcher.toUpperCase(); // Ok: string
}
```

checking if `typeof researcher` is "string" indicates to TypeScript that the type of researcher must be `string`:

# Literal Types

- When you declare a variable via `var` or `let`, you are telling the compiler that there is the chance that this variable will change its contents.
- In contrast, using `const` to declare a variable will inform TypeScript that this object will never change.