FTEC 5520 – Week 6

Agenda – Ethereum and Smart Contracts – Week 6

1. Fundamental Background behind blockchain
2. Another Blockchain - Ethereum
3. Ethereum 2.0
4. Ethereum Use Case
5. Smart Contracts
6. Algorithmic Decision Making

Fundamental background behind blockchain

Distributed Systems

A set of independent nodes or computers working together

Appear as a single computer or process to end-user

These computers shared state and operate concurrently

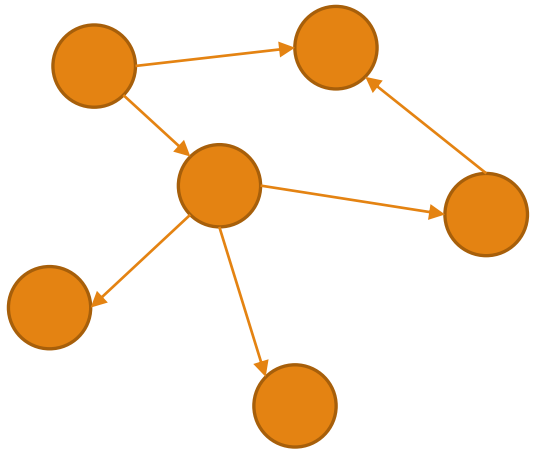
Can fail independently without affecting the whole system's uptime

Accomplish a common goal

No global clock

Lamport stated: Category of Property

- Safety
  - This will **NOT** happen
- Liveness
  - This **must** happen



What is P2P file sharing?

File sharing

File sharing refers to the **providing and receiving** of **digital files** over a network, usually following the peer-to-peer (P2P) model, where the files are **stored on and served by personal computers** of the users. (Wikipedia)

Peer to Peer (P2P)

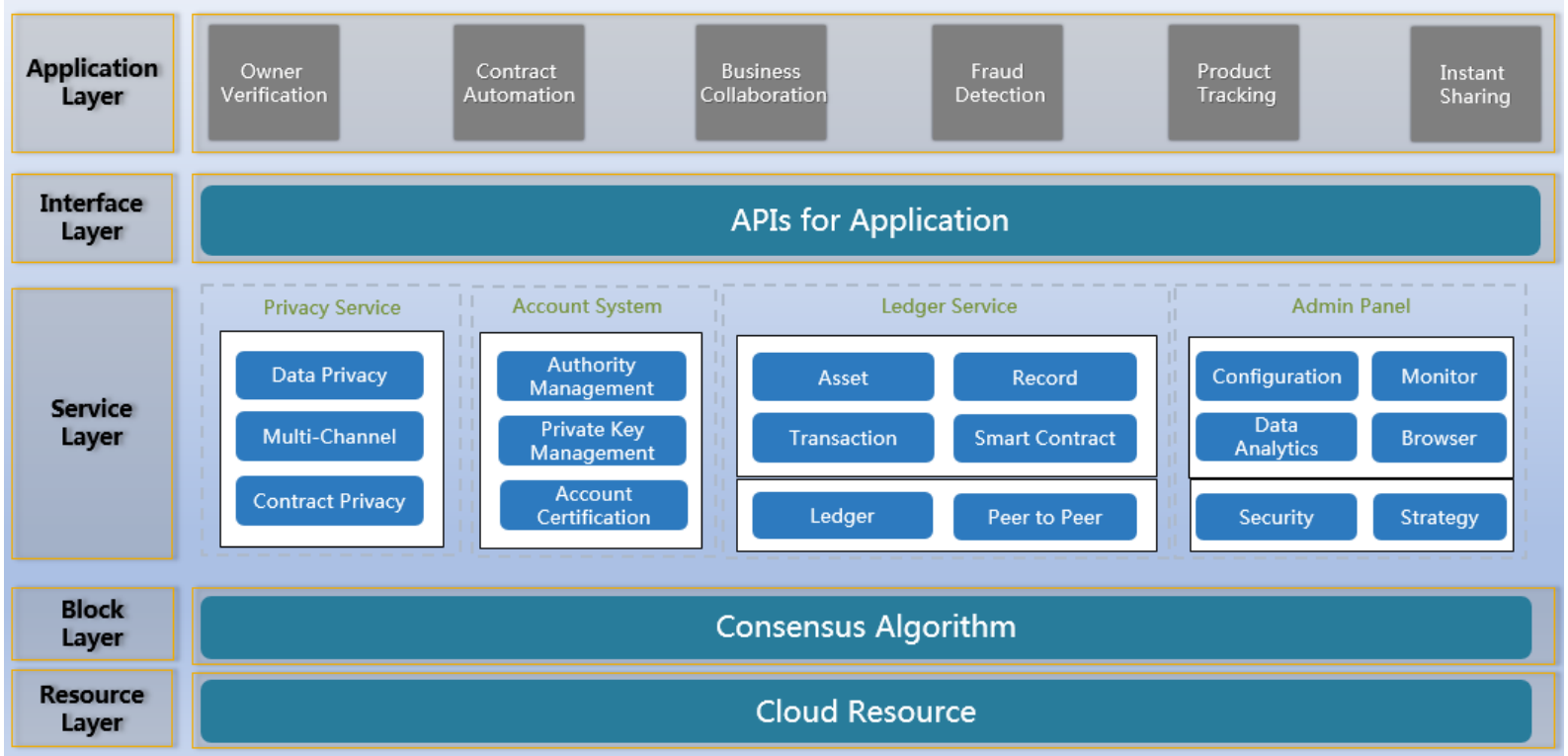
A Peer to Peer (or P2P) computer network uses diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application.

Types of P2P client

File sharing type P2P	• Napster, WinMX, BitTorrent, eMule, Foxy, KaZaA
Streaming media	• P2PTV, PPLive, PeerCast, PPStream
VoIP	• Skype
Instant Messaging	• Google Chat, AIM, ICQ, Tencent QQ, Windows Live Messenger (MSN), Yahoo Messenger, Lotus Sametime

- Discovery Protocol
- Data Transfer Protocol

System Architecture Overview (a Typical DLT system)



Ethereum

> *From a computer science perspective, Ethereum is a deterministic but practically unbounded state machine, consisting of a globally accessible singleton state and a virtual machine that applies changes to that state. (In 2013)*

Ethereum

Ethereum was designed with a broader vision, as a decentralized or world computer that attempts to marry the power of the blockchain, as a trust machine, with a Turing-complete contract engine

The Ethereum main net has processed over 500 million transactions — surpassing the 500 million mark on July 17th 2019

In November 2013, a Russian-Canadian guy named Vitaly Dmitriyevich "Vitalik" Buterin publishes the white Ethereum paper.

Over 70 million unique addresses exist on Ethereum, 16 million of which (16%) have been created since the beginning of the year.

Well known to be Public blockchain but can be both Public or Private blockchain

Considered to possess these features

- General Purpose Blockchain
- Smart Contracts
- Decentralized Applications (DApps)

Ethereum

Some blockchains, such as Ethereum, can host applications, sort of acting like Apple's App Store or the Google Play Store of the decentralized world.

Ethereum is Turing complete, which means that the network can process complete programming languages

Ethereum can compute any algorithm that can be computed by any Turing machine.

Ethereum is like the operating system for which decentralized applications can run off of.

Ethereum's groundbreaking innovation is to combine the general-purpose computing architecture of a stored-program computer with a decentralized blockchain,

What is Ethereum?
Leverage the Blockchain / Distributed Shared Ledger for general use

Focus on Smart Contracts that can be executed in decentralized way without downtime, censorship, fraud or interference.

To create markets of assets exchange like debts, promises or funds to reduce middleman

Ethereum foundation is presales in August 2014 and is an open source project.

	bitcoin	ethereum
concept	digital money	smart contracts
transaction	send from alice to bob	send from alice to bob if.. • date = jan 1, 2018 • bob's balance < 10 eth
market cap (as of feb 2017)	~$18 billion	~$1 billion
founder	satoshi nakamoto (unknown)	vitalik buterin and team
release date	jan 2009	july 2015
release method	early mining	presale raised $18M in bitcoin

General Purpose Blockchains

Ethereum started as a way to make a general-purpose blockchain that could be programmed for a variety of uses.

Ethereum is also a distributed state machine.

But instead of tracking only the state of currency ownership, Ethereum tracks the state transitions of a general-purpose data store, i.e., a store that can hold any data expressible as a key–value tuple.

Ethereum has memory that stores both code and data, and it uses the Ethereum blockchain to track how this memory changes over time.

Like a general-purpose stored-program computer, Ethereum can load code into its state machine and run that code, storing the resulting state changes in its blockchain.

Ethereum is a platform or rather, a super-computer (made up of a set of computers or nodes connected to each other) capable of running programs through its own virtual machine called EVM (Ethereum Virtual Machine). These programs are called Smart Contract.

Ethereum: Programmable Blockchain

Bitcoin	Ethereum
Address	Address + data storage + code
Balance with “native” code	Balance with Ether + application-specific tokens with specific behavior
Transaction	Transaction + creation of contract code + calling contract code
Blocks: Transactions bundle	Blocks: “Transactions” + new state of entire system
Settle in 10 minutes	Settle in 12 Seconds
Currency: bitcoin	Currency: ether

What is Turing Completeness

Alan Turing, in 1936, created a mathematical model of a computer consisting of a state machine that manipulates symbols by reading and writing them on sequential memory (resembling an infinite-length paper tape).

With this construct, Turing went on to provide a mathematical foundation to answer (in the negative) questions about universal computability, meaning whether all problems are solvable.

He proved that there are classes of problems that are uncomputable.

Alan Turing further defined a system to be *Turing complete* if it can be used to simulate any Turing machine. Such a system is called a *Universal Turing machine* (UTM).

Turing complete which means that it's code is more rigid, and developers cannot launch apps to be hosted with it

Currency in Ethereum

Ether is the currency of Ethereum.

Every activity on Ethereum that modifies its state costs Ether as fee and miners who are successful in generating and writing a block in chain are also rewards Ether.

Ether can easily be converted to dollars or other traditional currencies through Crypto-exchanges.

The smallest denomination aka base unit of ether is called Wei.

https://github.com/ethereum/web3.js/blob/0.15.0/lib/utils/utils.js#L40

Currency in Ethereum

Ether is traded on public exchanges and its price fluctuate daily.

Gas is the internal currency of Ethereum.

The execution and resource utilization cost is predetermined in Ethereum in terms of Gas units.

This is also known as Gas Cost.

There is also Gas price that can be adjusted to lower price when price of Ether increases and higher price when price of Ether decreases

Why need Gas

It means you pay to compute, and the fees to compute are calculated by the instructions you execute.

This is not new, mainframes charged users to compute, and calculated fees based on CPU time.

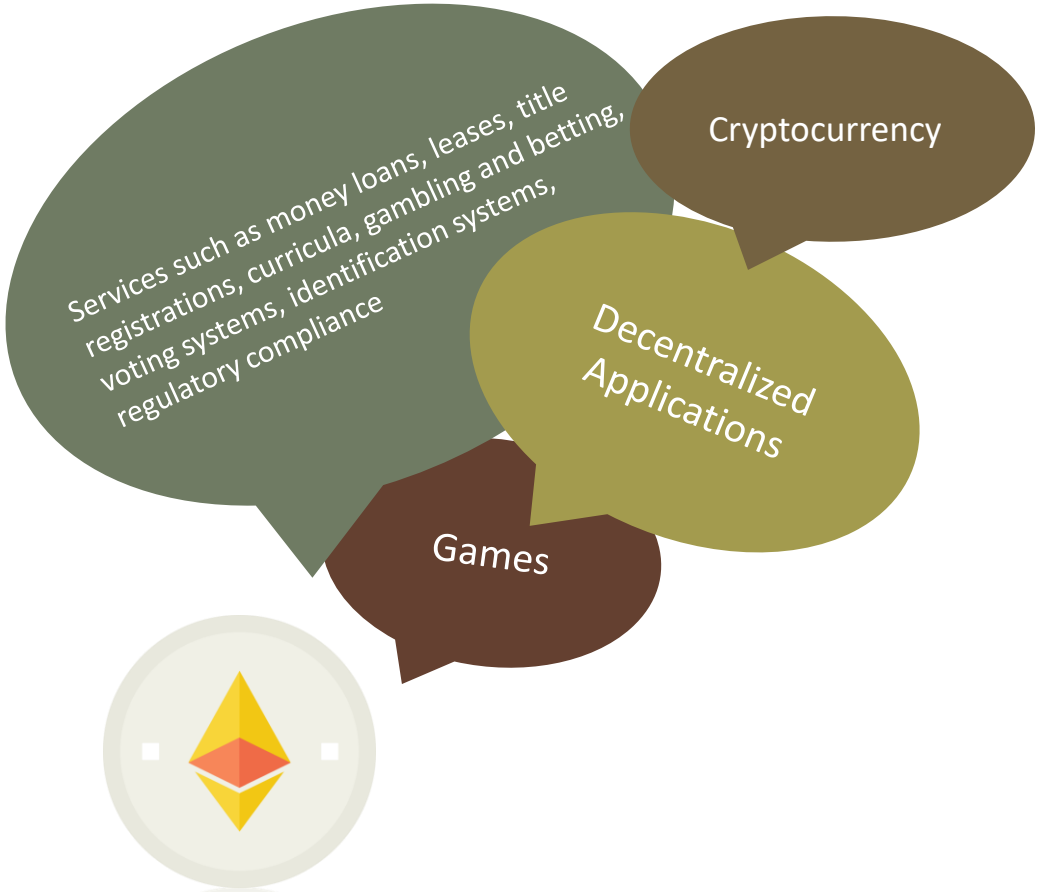
The problem with the Ethereum model is stability and ability to budget, because the

Gas is actually the cryptocurrency Ether represented by ETH on cryptocurrency exchanges.

The volatility for Ether and all cryptocurrencies is problematic for trying to budget and launch applications on the Ethereum platform.

Gas is measured in what Ethereum calls Gwei and is equal to 0.000000001 ETH.

What's Ethereum for?



Ethereum Discovery

Every node has a specific ID that is unique (Ethereum uses the SHA3 hash of your public key, node-IDs also consist of global IP and port so a connection can be established).

Each node stores a table of known active nodes each has K nodes in it. The node's main goal is to maintain exactly K known nodes in each row.

Row i represents all the known nodes that have the same first i bits in their address as the current node (row numbering starts at 0).

Ethereum clients are hard-coded with main peers that are maintained by the Ethereum Foundation.

Upon joining the network for the first time, the new peer will ask one of these bootstrap peers for a list of active nodes. Take note, these three bootstrap peers are a form of centralization

Ethereum Discovery

Hard coded in https://github.com/ethereum/go-ethereum/blob/master/params/bootnodes.go

// ETH/DEV Go Bootnodes (examples)

discover.MustParseNode("enode://a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c284339968eef29b69ad0dce72a4d8db5ebb4968de0e3bec910127f134779fbcb0cb6d3331163c@52.16.188.185:30303"), // IE

discover.MustParseNode("enode://de471bccee3d042261d52e9bff31458daecc406142b401d4cd848f677479f73104b9fdeb090af9583d3391b7f10cb2ba9e26865dd5fca4fcdc0fb1e3b723c786@54.94.239.50:30303"), // BR

discover.MustParseNode("enode://1118980bf48b0a3640bdba04e0fe78b1add18e1cd99bf22d53daac1fd9972ad650df52176e7c7d89d1114cfef2bc23a2959aa54998a46afcf7d91809f0855082@52.74.57.123:30303"), // SG

Ethereum RLPx and Data Transfer

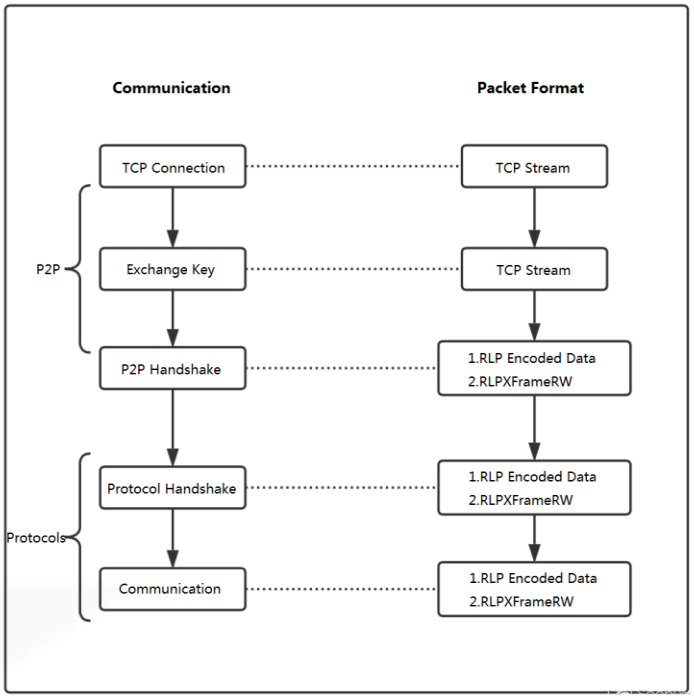
After node discovery, nodes are connected.

Ethereum then uses a protocol named RLPx.

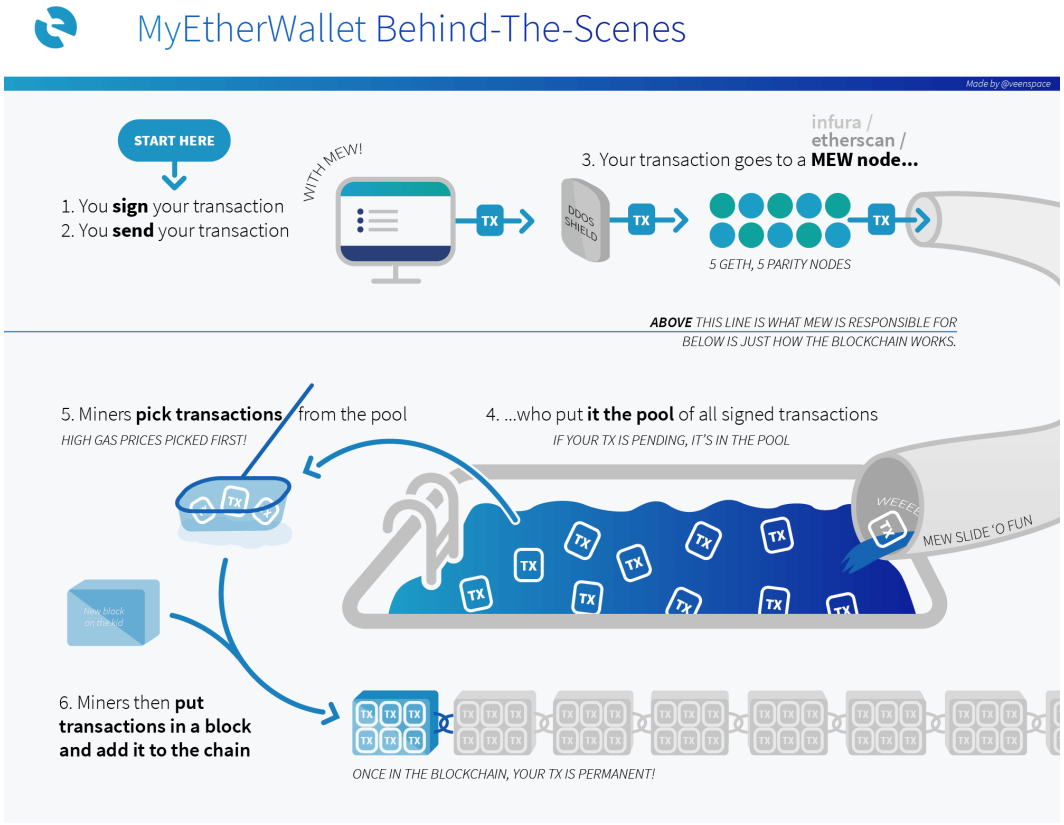
RLPx enables nodes to transfer encrypted, serialized data

When two nodes want to communicate,

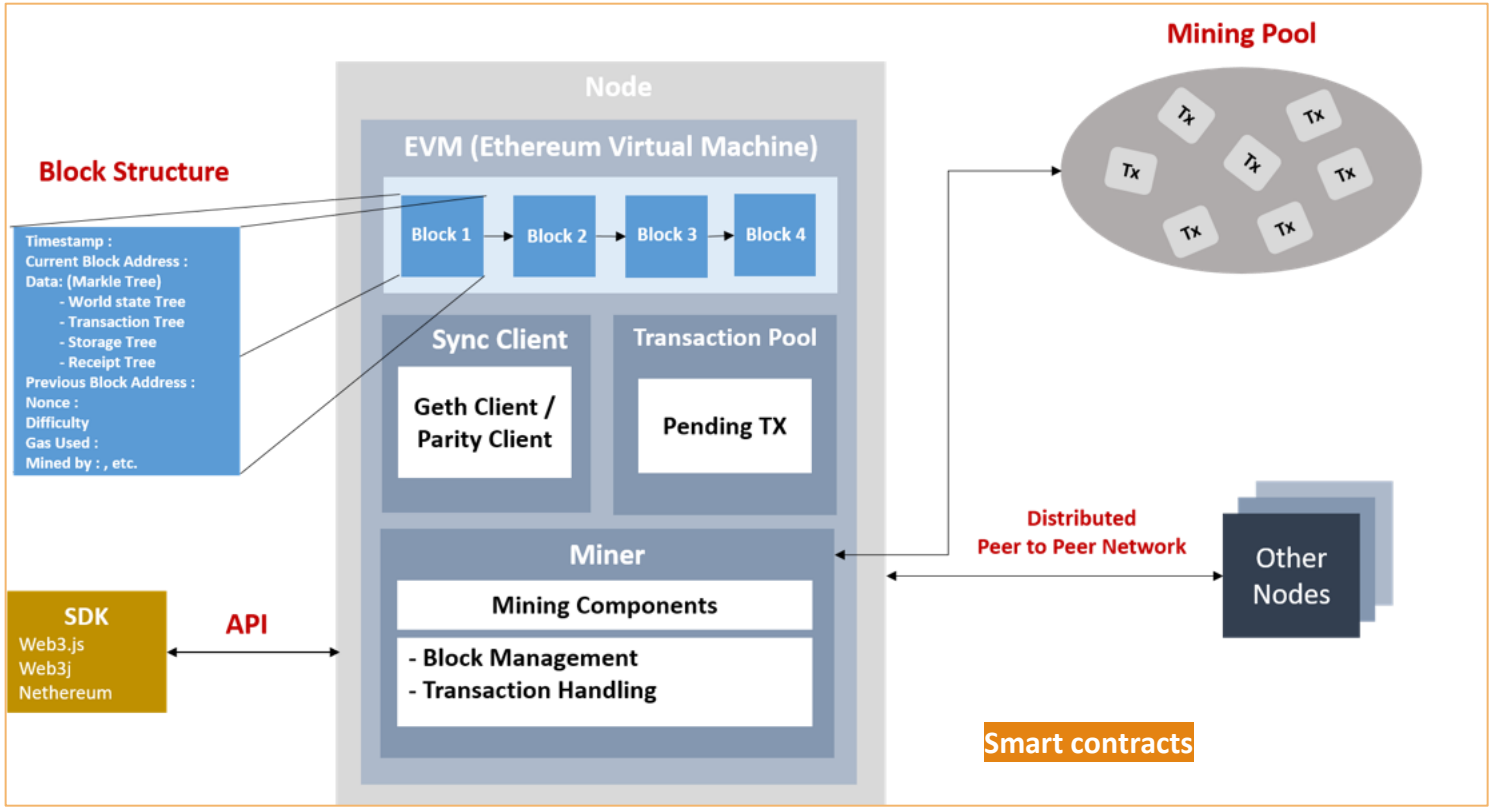
- they send each other some cryptographic data (public keys and such) to make sure all of the subsequent data transfer is encrypted (using ECDH, ECDHE, ECIES and more elliptic curve cryptography)
- cryptographically signed (you can learn more about elliptic curve cryptography here).



Ethereum Transaction Process

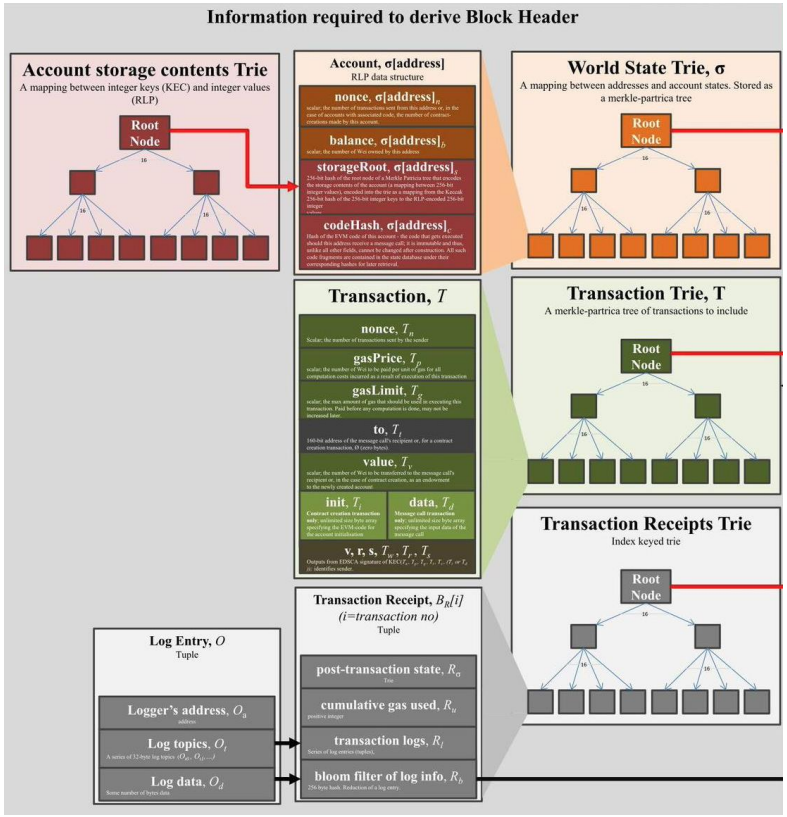


Ethereum Components



Consensus algorithm **Accounts** **Ether and Gas**

Ethereum Components (Data Merkle Tree)



World State:

- Key, value pairs for each account in the Ethereum network
- Values are the account's nonce, balance, storage root, and hash of its code (if a smart contract account)

Account Storage content:

- The data storage for each account's smart contracts

Transaction Trees

- A Merkle Tree containing the transactions of a given block

Transaction Receipts Trees

- Contains the gas usage, logs, Bloom filter, and status code of each transaction in a given block

Ethereum Block structure

```
// Header represents a block header in the Ethereum blockchain.
type Header struct {
    ParentHash  common.Hash    `json:"parentHash"       gencodec:"required"`
    UncleHash   common.Hash    `json:"sha3Uncles"       gencodec:"required"`
    Coinbase    common.Address `json:"miner"            gencodec:"required"`
    Root        common.Hash    `json:"stateRoot"        gencodec:"required"`
    TxHash      common.Hash    `json:"transactionsRoot" gencodec:"required"`
    ReceiptHash common.Hash    `json:"receiptsRoot"     gencodec:"required"`
    Bloom       Bloom           `json:"logsBloom"        gencodec:"required"`
    Difficulty  *big.Int       `json:"difficulty"       gencodec:"required"`
    Number      *big.Int       `json:"number"           gencodec:"required"`
    GasLimit    *big.Int       `json:"gasLimit"         gencodec:"required"`
    GasUsed     *big.Int       `json:"gasUsed"          gencodec:"required"`
    Time        *big.Int       `json:"timestamp"        gencodec:"required"`
    Extra       []byte         `json:"extraData"        gencodec:"required"`
    MixDigest   common.Hash    `json:"mixHash"          gencodec:"required"`
    Nonce       BlockNonce     `json:"nonce"            gencodec:"required"`
}
```

Header structure

Ethereum Header

Every block header in Ethereum contains not just one Merkle tree, but **three trees** for three kinds of objects:

Transaction

- Keep the status whether transaction has been stored in particular block

Receipts

- essentially, pieces of data showing the effect of each transaction

State

- Status of account
- Status of current balance of account
- Also used for pretend the output of the transaction of the contract

Ethereum Typical Block

```
{ difficulty: BigNumber { s: 1, e: 5, c: [ 135070 ] },
  extraData: '0xd783010702846765746885676f312e398777696e646f7773',
  gasLimit: 4011042861,
  gasUsed: 43406,
  hash: '0xba93a91df520c7565e80347346e47b83a41d473a33352d1cf7e689c30b305ba5',
  logsBloom: '0x000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000',
  miner: '0xa57de277ede9c1521f51f6989ed2497a5b9c1926',
  mixHash: '0x4e80de770c329aebbc2e9e861190784f57b7f9910bbb0495348280522 84f32e4',
  nonce: '0x655dee191333922c',
  number: 70,
  parentHash: '0x27d3dbc34614f88583f29ea1b7546e83563e55638b1d0a258de04f4912f42aaf',
  receiptsRoot: '0x5dff465dd85c4ad02c71ec4099284ecaf91ed9bb600f7903978386059c16fb8d',
  sha3Uncles: '0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347',
  size: 742,
  stateRoot: '0xb15363a8958d218eff295b5e877517ee4243f1b681d987793c5ec30a04bc4592',
  timestamp: 1511421241,
  totalDifficulty: BigNumber { s: 1, e: 6, c: [ 9302609 ] },
  transactions:
   [ '0x6b65b86462e6aa89d5f9469ce03b9ea21e8bf72f8c11aa72de2978f9b7a5b9fd' ],
  transactionsRoot: '0x5aceca068d1a7ac8d8ecd8f19469ec7c687cb6ceed54e0db39b58dfdf6481de9',
  uncles: [] }
```

Ethereum Nodes

Types of Nodes	Features
Full Nodes	• Stores the full blockchain data available on disk and can serve the network with any data on request. • Receives new transactions and blocks while participating in block validation. • Verifies all blocks and states. • Stores recent state only for more efficient initial sync. • All state can be derived from a full node. • Once fully synced, stores all state moving forward similar to archive nodes (more below).
Light nodes (for low capacity devices)	• Stores the header chain and requests everything else on demand. • Can verify the validity of the data against the state roots in the block headers.
Archive nodes	• Stores everything kept in the full node. • Also builds an archive of historical states.
Mining Nodes	• A miner is responsible for writing transactions to the Ethereum chain. A miner job is very similar to that of an accountant.

Account and Account Address

Each account has a state associated with it and a 20-byte address.

An address in Ethereum is a 160-bit identifier that is used to identify any account.

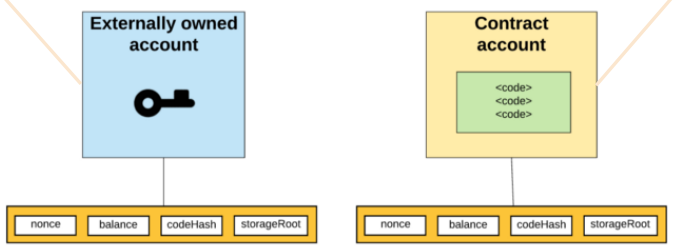
Two types of accounts

- Externally owned accounts, which are controlled by private keys and have no code associated with them.
- Contract accounts, which are controlled by their contract code and have code associated with them.

Account state

- **Nonce**
- **Balance**
- **StorageRoot**
- **CodeHash**

- has an ether balance,
- can send transactions (ether transfer or trigger contract code),
- is controlled by private keys,
- has no associated code.



- has an ether balance,
- has associated code,
- code execution is triggered by transactions or messages (calls) received from other contracts.
- when executed – perform operations of arbitrary complexity (Turing completeness) – manipulate its own persistent storage, i.e., can have its own permanent state – can call other contracts

Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM) is where smart contracts run in Ethereum.

Every single operation that is executed inside the EVM is actually simultaneously executed by every node in the network.

EVM is a 256bit machine. It is most natural to manipulate data in chunks of 32 bytes.

Persistent storage is quite expensive.

The Solidity compiler makes interesting choices in order to minimize gas usage.

An EVM compiler doesn't exactly optimize for bytecode size or speed or memory efficiency.

In the context of blockchains, an oracle is a system that can answer questions that are external to Ethereum.

Ethereum Components

Ethereum Client: Geth

Ethereum Network: Use Geth to connect to the network (example)

- $ geth --networkid 3 --datadir "./ropsten-db" --keystore "./ropsten-keys" --syncmode "fast" --rpc --rpcport "8546" --rpcapi "web3,eth,miner,admin" --rpccorsdomain "*" --port 30301 console

For Private Ethereum Network

- $ geth account new --datadir datadir
- Create a genesis.json file with content added:
  - “{
  - "config": {
  - "chainId": 1100,
  - "homesteadBlock": 0,
  - "eip155Block": 0,
  - "eip158Block": 0,
  - "byzantiumBlock": 0
  - },
  - "difficulty": "400",
  - "gasLimit": "2000000",
  - "alloc": {
  - "87db8fceb028cd4ded9d03f49b89124a1589cab0": {
  - "balance": "1000000000000000000000000"
  - }
  - }
  - }”

Layer 2 on Ethereum

The Ethereum mainnet, also called ‘layer 1,’ regularly processes well over 1 million transactions a day, but demand is much greater than capacity.

As the average cost of transacting on layer 1 increases, more and more people are priced out of using decentralized apps like decentralized exchanges or NFT marketplaces.

Layer 2 is a solution that build on top of layer 1 to improve the scalability of the Ethereum network.

A layer 2 blockchain regularly communicates with Ethereum (by submitting bundles of transactions) in order to ensure it has similar security and decentralization guarantees. All this requires no changes to the layer 1 protocol (Ethereum).

Generalized layer 2s behave just like Ethereum — but cheaper. Anything that you can do on Ethereum layer 1, you can also do on layer 2. Many dapps have already begun to migrate to these networks or have skipped Mainnet altogether to deploy straight on a layer 2.

Ethereum's Cryptographic Hash Function (SHA3)

Ethereum uses the Keccak-256 cryptographic hash function in many places. Keccak-256 was designed as a candidate for the SHA-3 Cryptographic Hash Function Competition held in 2007 by the National Institute of Science and Technology.

Keccak was the winning algorithm, which became standardized as Federal Information Processing Standard (FIPS) 202 in 2015.

Keccak256("") = c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470

SHA3("") = a7ffc6f8bf1ed76651c14756a061d662f580ff4de43b49fa82d80a4b80f8434a

Category of Blockchain Networks

Public Blockchains

Private Blockchains

Consortium Blockchains

Ethereum Networks

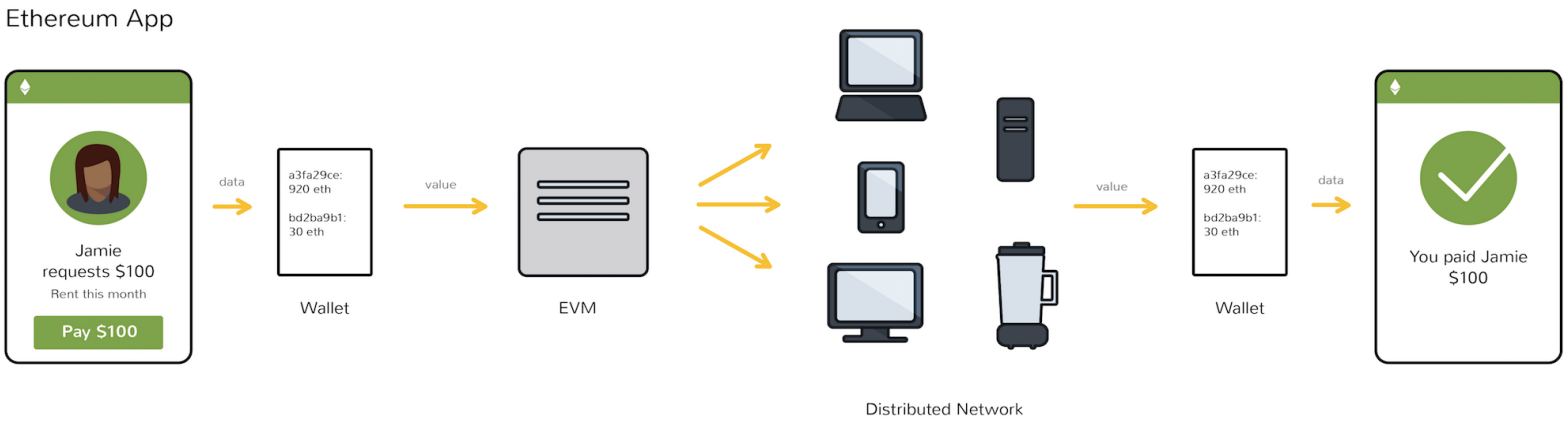
MainNet: Main Ethereum Network

"**TestNet**:" the Test Ethereum Network. The Ether on the TestNet does not carry any real value and is only for a collaborative testing on the network.

The **private nodes** can be quickly launched with minimal storage and memory requirements and without much effort.

https://ethernodes.org/network/1

Ethereum Transaction



Ethereum Transaction

A transaction is an agreement between a buyer and seller, a supplier and a consumer or a provider and a consumer that there would be exchange of assets, products or services in lieu of currency, cryptocurrency or some other asset either in present or in future

There are three types of transaction that can be executed in Ethereum.

1. Transfer of Ether from one account to another. The accounts can be externally owned accounts or contract account. Following are the possible cases

a. An externally owned account sending ether to another externally owned account in a transaction.

b. An externally owned account sending ether to a contract account in a transaction.

c. A contract account sending ether to another contract account in a transaction.

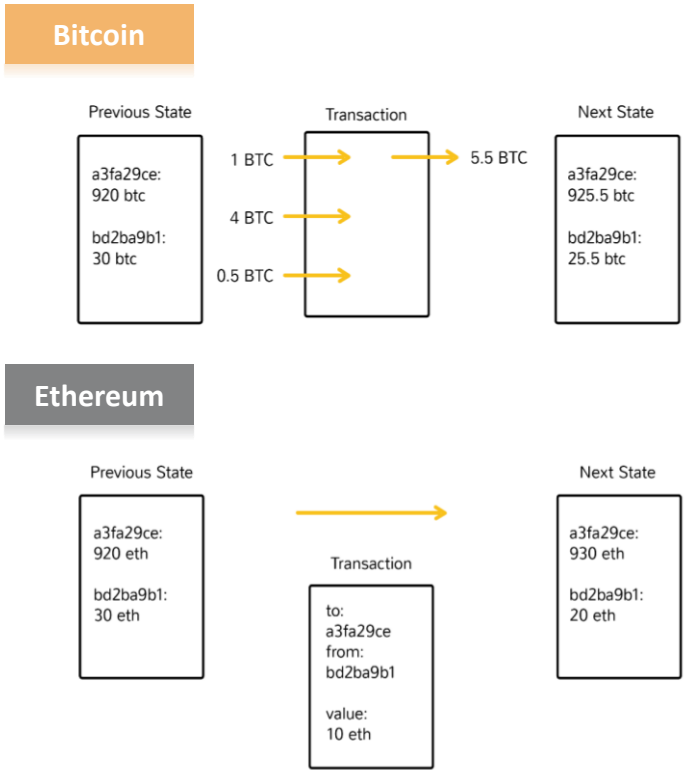
d. A contract account sending ether to an externally owned account in a transaction

Ethereum – account based

Bitcoin uses unspent transaction outputs to track who has how much bitcoin.

Ethereum, uses accounts.
Like bank account funds, ether tokens appear in a wallet, and can be ported (so to speak) to another account.

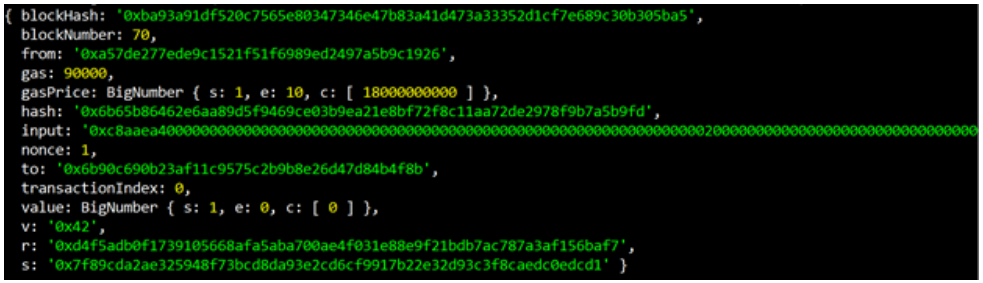
The big difference with ethereum is that its nodes store the most recent state of each smart contract, in addition to all of the ether transactions.

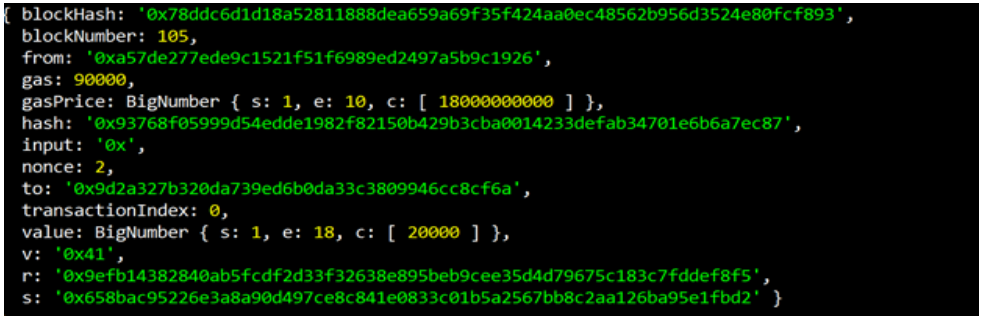


Ethereum Transaction

2. Deployment of Smart contract — An externally owned account can deploy a contract using a transaction in Ethereum virtual machine.

3. Using or invoking a function within a contract — Executing a function in a contract that changes state are considered as transactions in Ethereum. If executing a function does not change state, it does not require a transaction





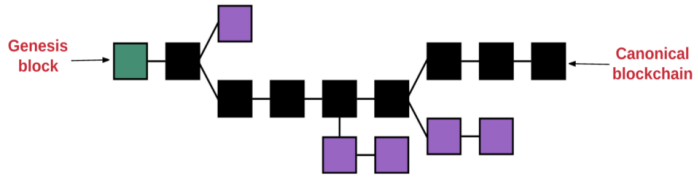
Ethereum Transaction

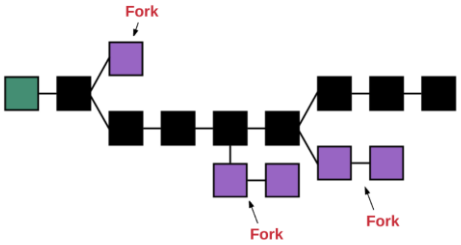
For a transaction to be considered valid, it must go through a validation process known as mining. Mining is when a group of nodes (i.e. computers) expend their compute resources to create a block of valid transactions.

Whenever multiple paths are generated, a “fork” occurs. We typically want to avoid forks, because they disrupt the system and force people to choose which chain they “believe” in.

Ethereum uses a mechanism called the **“GHOST protocol.”**

“GHOST” = “Greedy Heaviest Observed Subtree”

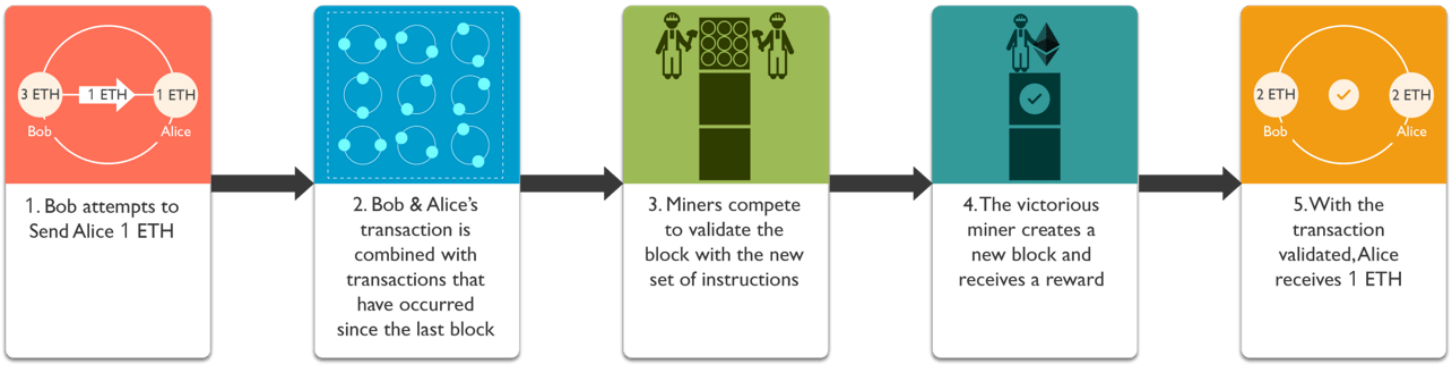




Ethereum Transaction

Ethereum is a transaction-based state machine. In other words, transactions occurring between different accounts are what move the global state of Ethereum from one state to the next.

In the most basic sense, a transaction is a cryptographically signed piece of instruction that is generated by an externally owned account, serialized, and then submitted to the blockchain.



Why do we need mining?

Mining is needed because of the consensus requirement across distributed nodes within the network

Determine the scheme of restriction

- Generator of Block
- Validator of Block

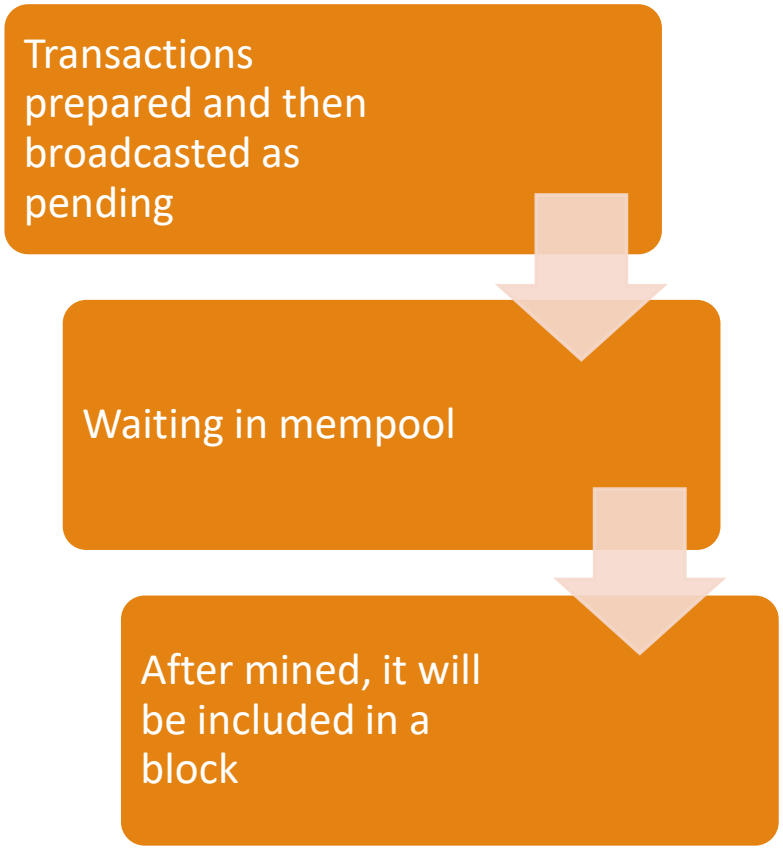
How to establish the trust

- Whitelist
- Blacklist
- Vote or compete to trust
- Automatically trust

How to restrict new chain

So do we need consensus across federated and/or private blockchain? How?

Ethereum Mining



Transactions in Ethereum may not always be successful

- Running out of gas during execution
- Failed precondition check in the smart contract code

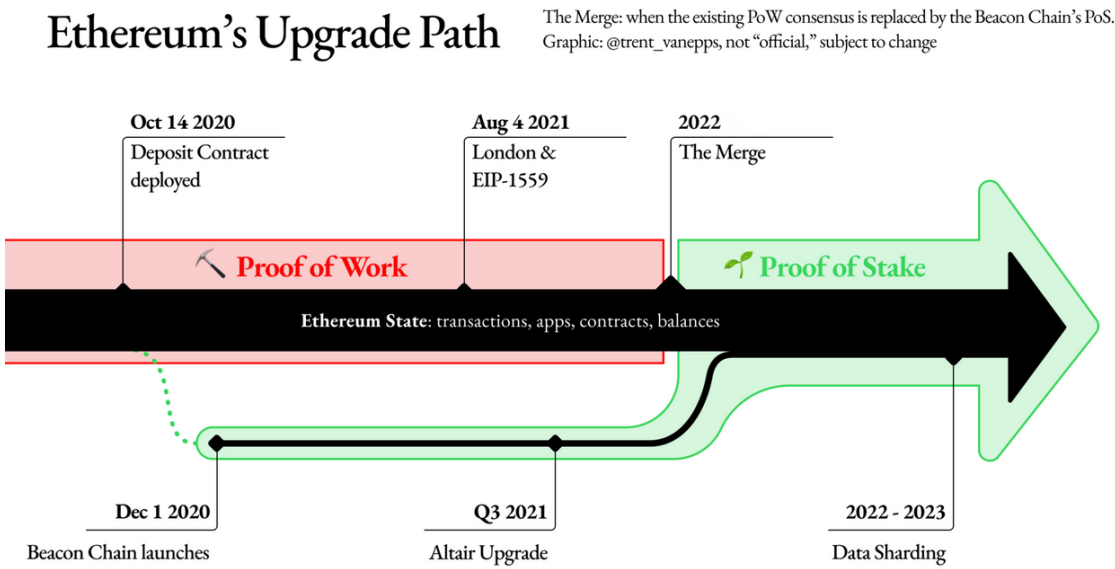
Failed transactions are either Aborted or Reverted

Ethereum Mining

Mining is the process of creating a block of transactions to be added to the Ethereum blockchain.

In Ethereum,

- Execution Layer (Eth1)
- Consensus Layer (Eth2)



About Ethereum Testnets

Ethereum has several testnets (Ropsten, Rinkeby, Kovan, and Goerli). The consensus algorithm were developed using various smart contracts

- Ropsten (id 3) is the only proof-of-work-based testnet, which makes it the most similar one to mainnet, but is also very unreliable.
- Rinkeby (id 4) is a proof-of-authority-based testnet, which means that there are a set of trusted nodes which have the authority to add new blocks to the blockchain.
- Kovan (id 42) is similar to Rinkeby, in that it is a proof-of-authority- based testnet, but its consensus algorithm is compatible with Parity clients instead of Geth.
- Goerli (id 6) is the most recent testnet set up. It uses proof-of- authority as well, with the advantage that it is compatible with both Geth and Parity clients.

Ethereum Classic (ETC)

Ethereum Classic came to be after members of the Ethereum community implemented a time-sensitive hard fork (codenamed “DAO”).

On July 20, 2016, at a block height of 1.92 million, Ethereum introduced an irregular state change via a hard fork in an effort to return approximately 3.6 million ether that had been taken from a smart contract known as The DAO.

Almost everyone agreed that the ether taken had been stolen and that leaving it all in the hands of the thief would be of significant detriment to the development of the Ethereum ecosystem as well as the platform itself.

A number of people in the ecosystem disagreed with this change, believing immutability should be a fundamental principle of the Ethereum blockchain without exception

They elected to continue the original chain under the moniker of Ethereum Classic.

Ethereum ERC-20

The Ethereum Improvement Proposal repository is located at https://github.com/ether eum/EIPs/.

A token can generally represent any asset that has a value attached to it. In this case, we are talking about tokens which represent a smart contract and use of the Ethereum blockchain.

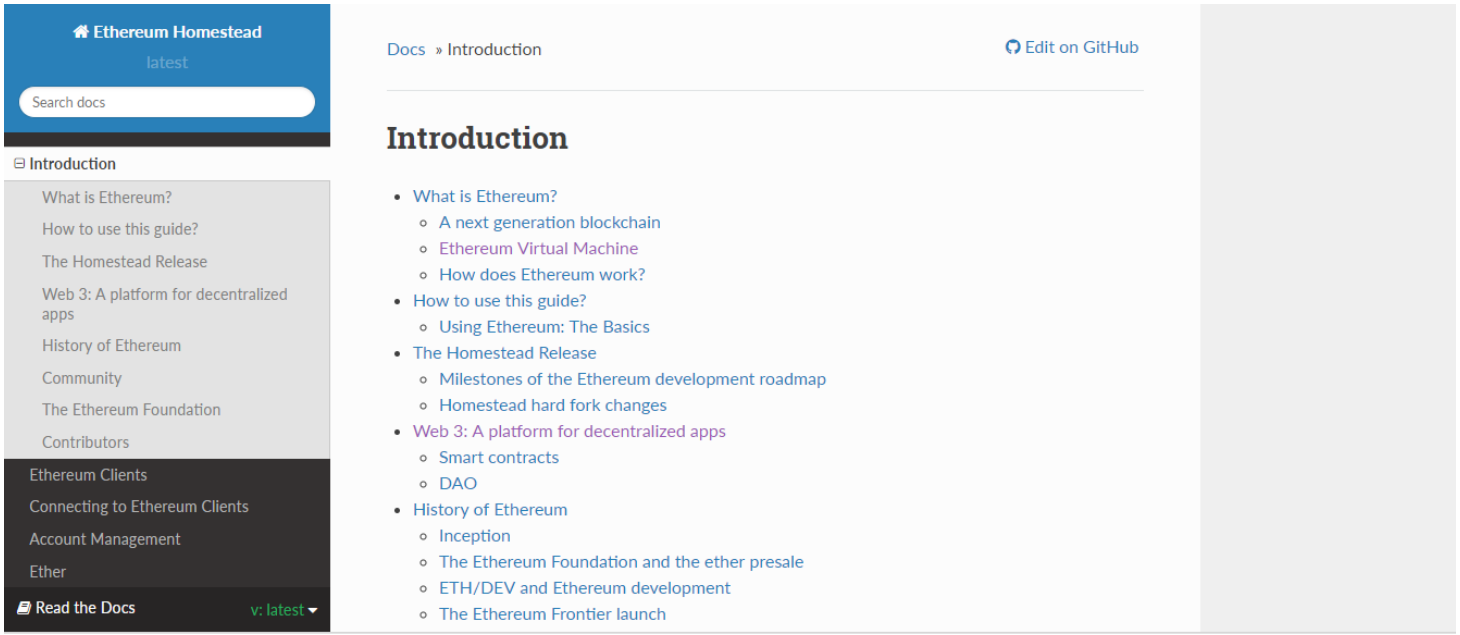
The most common ETH-based cryptocurrencies are built on the ERC-20 token standard.

These tokens can be traded, bought or sold. ERC-20 is a token protocol for all tokens implemented by the Ethereum blockchain (Token protocol means rules or standards that the token must follow).

All tokens which implement the protocol become a compliant token of ERC-20.

The technical specification of this protocol comprises six functions that ensure that all tokens based on the Ethereum system work anywhere on the platform

More readings



http://www.ethdocs.org/en/latest/introduction/index.html

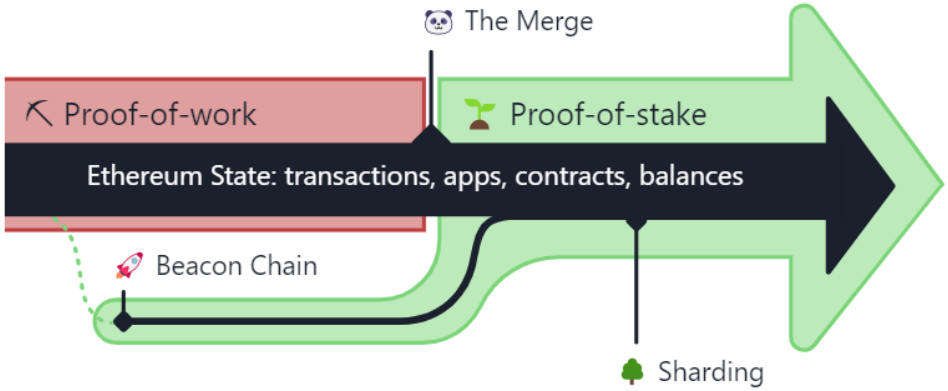
Ethereum 2.0

Both Ethereum 2 and layer 2 are developed to solve scalability problems in Ethereum mainnet and improve energy efficiency

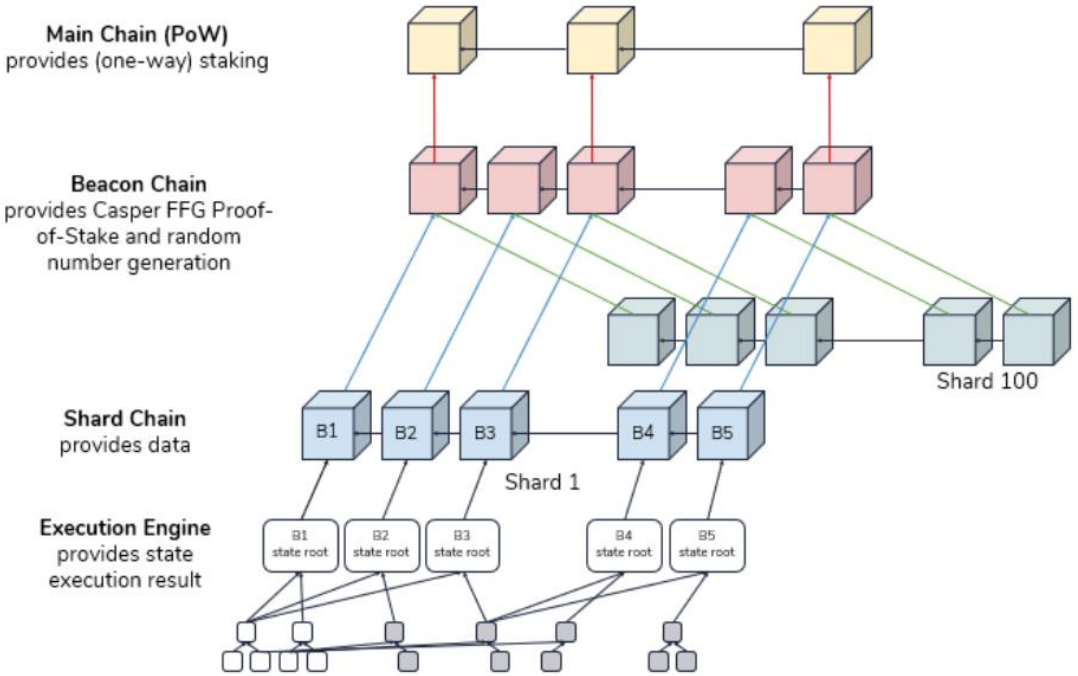
While the previous Ethereum network only assists 15-30 transactions per second. Ethereum's new Proof-of-stake(PoS) can assure 100,000 transactions per second

Changes

- 1) Transfer from Proof of Work (PoW) to Proof of Stake (PoS)
- 2) Beacon chain upgrade – as essential PoS blockchain that connect stakers and shards
- 3) Use Sharding that allow hierarchical set of Blockchains to process transactions independently



Ethereum 2.0 setup and architecture



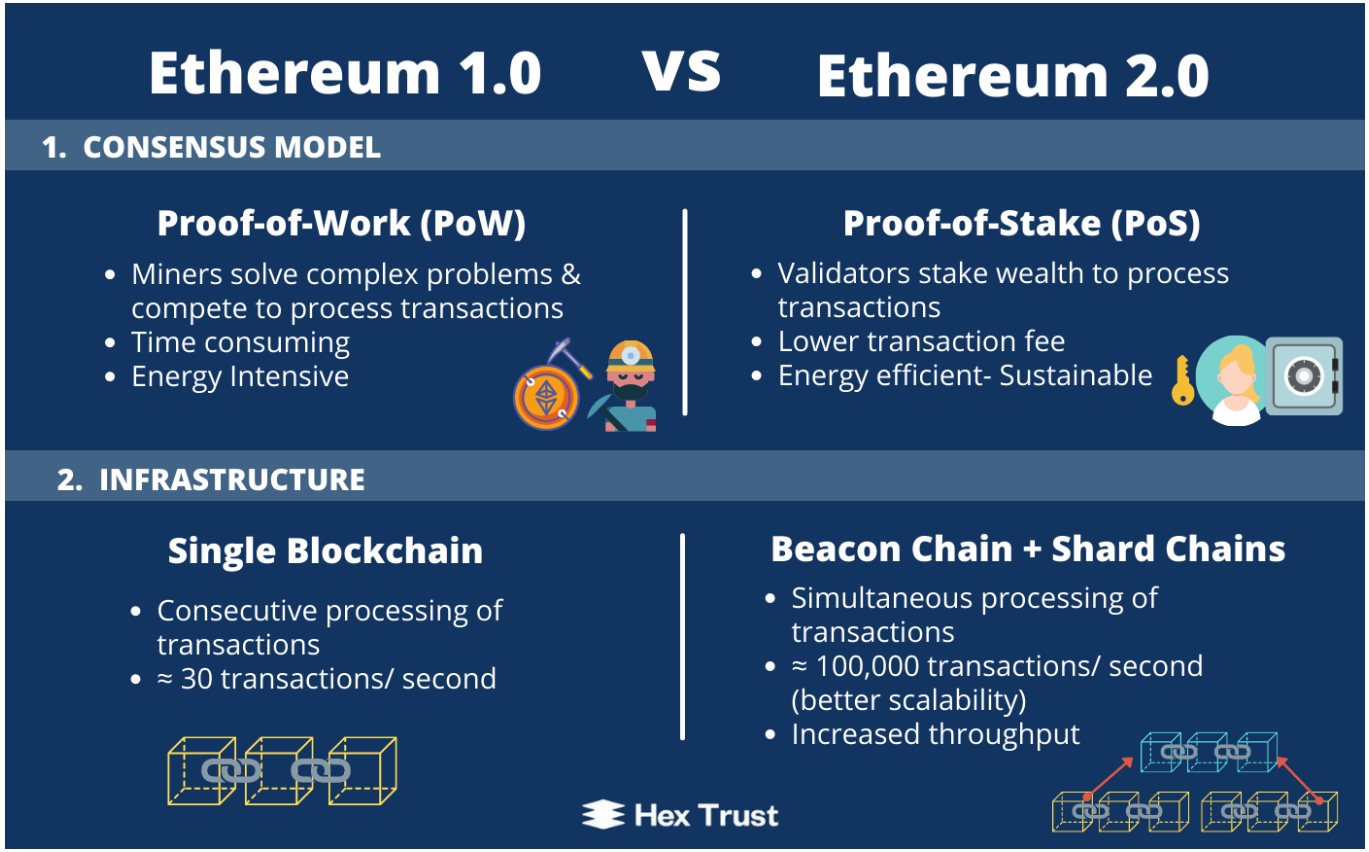
With 3 blockchains in Ethereum 2

- Legacy Ethereum mainnet
  - Provides the staking assets for PoS
- Beacon chain
  - For performing PoS consensus
- Sharding blockchains
  - Store Ethereum 2 data and carry out EVM computations and process transactions

4 types of client nodes

- Eth1 client nodes (e.g. geth client)
- Beacon nodes for propose block
- Validation node for beacon chain
- Sharding node for blocks of the sharding blockchains

Main difference between Ethereum 1.0 and 2.0



Ethereum Mining with PoS

When Ethereum 2.0 replaces Proof-of-Work with Proof-of-Stake, there will be the added complexity of shard chains. These are separate blockchains that will need validators to process transactions and create new blocks.

As a result, extra coordination is necessary and will be done by the beacon chain.

The beacon chain receives state information from shards and makes it available for other shards, allowing the network to stay in sync. The Beacon Chain was launched in Phase 0 of the network upgrade and marked the first step in the transition from PoW to PoS.

Validators are algorithmically chosen by the beacon chain to propose new blocks. A deposit of 32ETH is required to set up a validator.

Why Not immediately move to PoS?

Major roadblock that's stopping PoS from being completely viable – Nothing at Stake problem.

If we were using PoW, you couldn't move to a new chain without having to spend lots of processing power and time generating new blocks for as long as people decide to accept it as a new blockchain

But with PoS, you can simply diversify your portfolio and bet on all the blocks that look appealing without consequences (no risk), since you won't be losing your stake as long as you're validating good transactions.

This kind of possibility could end up creating hundreds of different blockchains, since people could be mining on all of them simultaneously.

Switching to PoS (Mar 2022)

From Ethereum.org

The Kintsugi merge testnet, launched late December, has been a valuable testing ground for The Merge.

Through various test suites, multi-client devnets, shadow forks of Goerli, application deployments, and the community's help #TestingTheMerge, we've arrived at a set of stable and robust protocol specifications.

Now that clients have implemented these latest specs, a successor to Kintsugi, Kiln , is being launched!

Like the Ethereum mainnet, Kiln's execution layer was launched under proof-of-work in parallel to a Beacon Chain running proof-of-stake.

The Merge happened on Kiln on March 15, 2022. The network is now running entirely under proof-of-stake!

以太坊核心開發者 Tim Beiko 在推特發文表示，以太坊合併公共測試網 Kiln 已經上線，該測試網上週僅以「工作量證明」（PoW）的模式啟動，本週內將全面過渡到「權益證明」（ PoS）共識機制，為接下來以太坊主網與信標鏈的合併進行一場「模擬彩排」。

Tim Beiko | timbeiko.eth
@TimBeiko

Kiln, the next iteration of Ethereum merge testnets, is now live

Highly recommended that node operators, application developers, stakers, tooling/infra providers test their setups on the network. Blog post has all the info to get started

Announcing the Kiln Merge Testnet

Posted by Protocol Support Team on March 14, 2022

Research & Development

The Kintsugi merge testnet, launched late December, has been a valuable testing ground for The Merge. Through various test suites, multi-client devnets, shadow forks of Goerli, application deployments, and the community's help #TestingTheMerge, we've arrived at a set of stable and robust protocol specifications. Now that clients have implemented these latest specs, a successor to Kintsugi, Kiln , is being launched!

Like the Ethereum mainnet, Kiln's execution layer was launched under proof-of-work in parallel to a Beacon Chain running proof-of-stake. The Merge happened on Kiln on March 15, 2022. The network is now running entirely under proof-of-stake!

https://blog.ethereum.org/2022/03/14/kiln-merge-testnet/

Is Ethereum already supporting PoS?

Ethereum 2.0 is not entirely a new blockchain. It's just a new update on the existing mechanism. The governance mechanism of the network is changed from proof-of-work to proof-of-stake.

Eth 1.0 is the "execution layer" and Eth 2 is the "consensus layer."

As the term "Merge" indicates, it is not planned that the current mainnet gets offline once the Beacon chain takes over.

Instead, the old PoW mainnet and new Beacon chain will be merged into one more robust, scalable, and sustainable system.

The PoW chain will ensure that smart contracts will run on the PoS chain while keeping the full history and transaction data available.

Proof of Stake in Ethereum

In a Proof-of-Stake consensus mechanism, participants who have staked crypto assets (staking means "locking" them up by sending them to a specific smart contract) are selected at random to become validators who propose new blocks - and are rewarded for doing so.

Validators thus take over the role held by miners in a Proof-of-Work system.

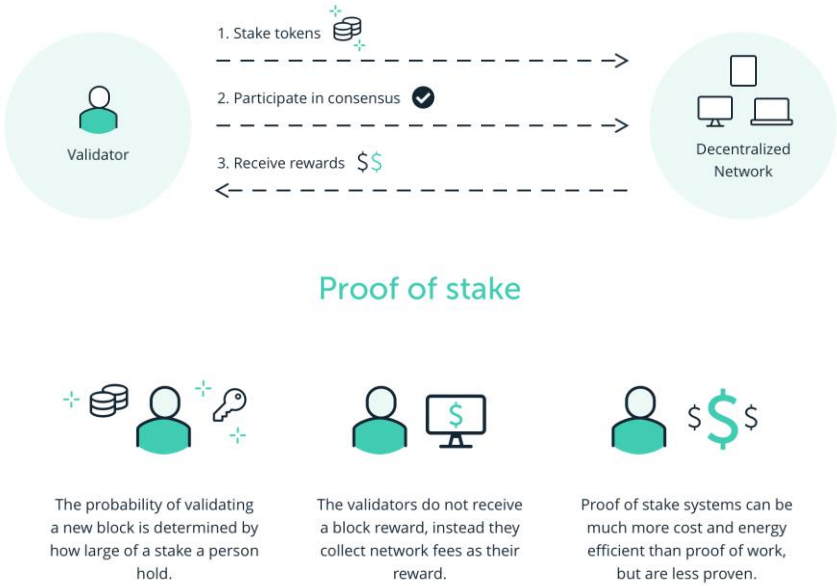
To participate as a validator, a user must deposit 32 ETH into the deposit contract and run three separate pieces of software: an execution client, a consensus client, and a validator.

One validator is randomly selected to be a block proposer in every slot.

This validator is responsible for creating a new block and sending it out to other nodes on the network.

Participants who violate the rules of the protocol are subject to forfeiture, in part or in full, of their staked assets.

They are not paid in block rewards but in transaction fees



The Fork and Merge

In this phase, the Proof-of-Work Ethereum mainnet will merge with the Proof-of-Stake Beacon Chain.

At the completion of the merge, Proof of Work on Ethereum will be discontinued.

The Berlin and London Hard Fork aren't the regular non-contentious upgrades that we see every year.

They bring together a series of changes in the network to continue the natural progression of Ethereum towards Ethereum 2.0, also known as the Serenity

The London Hard Fork is the most anticipated upgrade among the Ethereum community that aims to solve the decades-old problem of high transaction fees on the Ethereum network.

In this fork, layer-2 solutions will be added.

The Merge is likely to take place in the second half of 2021 or in the first half of 2022.

Ethereum's last major hard fork was The Merge, a significant change to the rules of the blockchain network which made the network change from proof-of-work (PoW) to proof-of-stake (PoS).

According to a recent Ethereum Core development meeting on Dec. 8, developers disclosed that the next Ethereum hard fork, called Shanghai, could be implemented by March 2023.

Shanghai hard fork will be able to manage the network's staked ethereum withdrawals.

Shanghai also may address the network's staked ethereum withdrawals as the community has established importance on the matter

The Fork

BERLIN FORK

Apr 15, 2021

EIP-2565, which reduces gas cost for a specific transaction type that uses modular exponentiation.

EIP-2718, makes all transaction types “backwards compatible” using so-called “envelope transactions,” which allows the addition of new transaction logic into Ethereum.

EIP-2929, increases gas costs for “op code” transactions, a pain point for denial of service attacks on Ethereum in the past.

EIP-2930, a new transaction type (made possible by EIP-2718’s envelope transactions) which allows its users to create templates for future, complex transactions in a bid to lower gas costs.

LONDON FORK

Aug 05, 2021

EIP-1559 aims to prevent major fluctuations in gas prices by setting a “base fee,” which is then algorithmically adjusted in accordance with demand for block space on the network.

EIP-3198 serves as a companion to EIP-1559 that adds the BASEFEE value, which allows smart contracts on the Ethereum Virtual Machine (EVM) to interact with the new mechanism on the network

EIP-3541 doesn’t do much on its own but lays the groundwork for broader EVM improvements in the future by reserving the “0xEF” byte.

EIP-3554 once again delays Ethereum’s difficulty bomb, designed to usher in Ethereum’s “Ice Age” by freezing mining operations as the network migrates to a proof-of-stake system with Ethereum 2.0, to December 1st, 2021.

The Fork

PARIS (THE MERGE)

Sep 15, 2022

the most anticipated Ethereum update

September is a test period to find out how sustainable and secure the upgrades were

The testnet is already live as well as a decentralized exchange (DEX, PulseX) to trace PRC-20 tokens.

What makes Pulsechain different in practice is the ability to import every single asset and contract from Ethereum (ERC-20).

Merging with Mainnet and will use PoS.

SHANGHAI FORK

March, 2023

EIP-4844 deals with concepts like proto-danksharding and shard blob transactions.

Both of these concepts leverage technology that's meant to implement a "new kind of transaction type to Ethereum which accepts 'blobs' of data to be persisted in the beacon node for a short period of time."

After the Shanghai upgrade, 'staked' ETH (stETH) users will be able to withdraw their funds as well as applicable staking rewards to validate transactions on the network. (Group stETH to Beacon chain)

Post-merge situation of Ethereum

Execution layer issuance

- Proof-of-work is no longer a valid means of block production under the upgraded rules of consensus.

Consensus layer issuance

- Proof-of-work is no longer a valid means of block production under the upgraded rules of consensus.
- separate Ethereum accounts to the accounts we're used to on Mainnet

Impact of Hard Fork to Miners

The burning mechanism has been met with controversy from the mining community, as it effectively slashes the income made from gas fees by 30%.

Combined with the looming threat of obsolescence from the Ethereum 2.0 migration, miners are now facing increased short-term pressure of revenue streams, which is sure to put a squeeze on low-margin operations.

ZK-STARKs

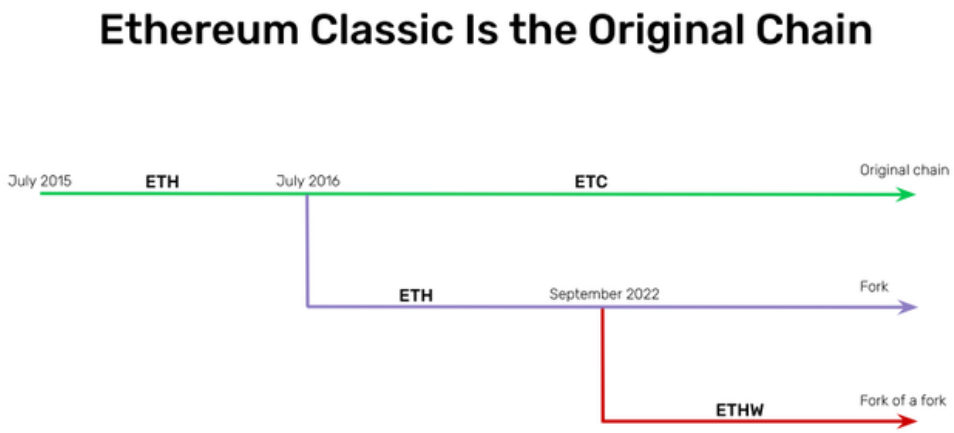
Zero Knowledge Scalable Transparent Arguments of Knowledge (ZK-STARKs).

- It works by proving that one party is in possession of specific data without actually revealing the data to the network.
- Zk-STARKs are a crucial element of Zero-Knowledge Rollups (zk-Rollups), a popular Layer-2 scaling solution.
- Zk-STARKs are a new development based on zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKS), and are distinguished by being more trustless than their predecessor.

Merge or Fork in the future?

According to Messari, the Merge will make the estimated $19 billion Ethereum mining industry and mining rigs obsolete.

Naturally, Ethereum miners are interested in putting their equipment to work elsewhere.



Usage of Ethereum

Use cases of Ethereum

Decentralized Finance (DeFi)

ICOs, Crowdfunding

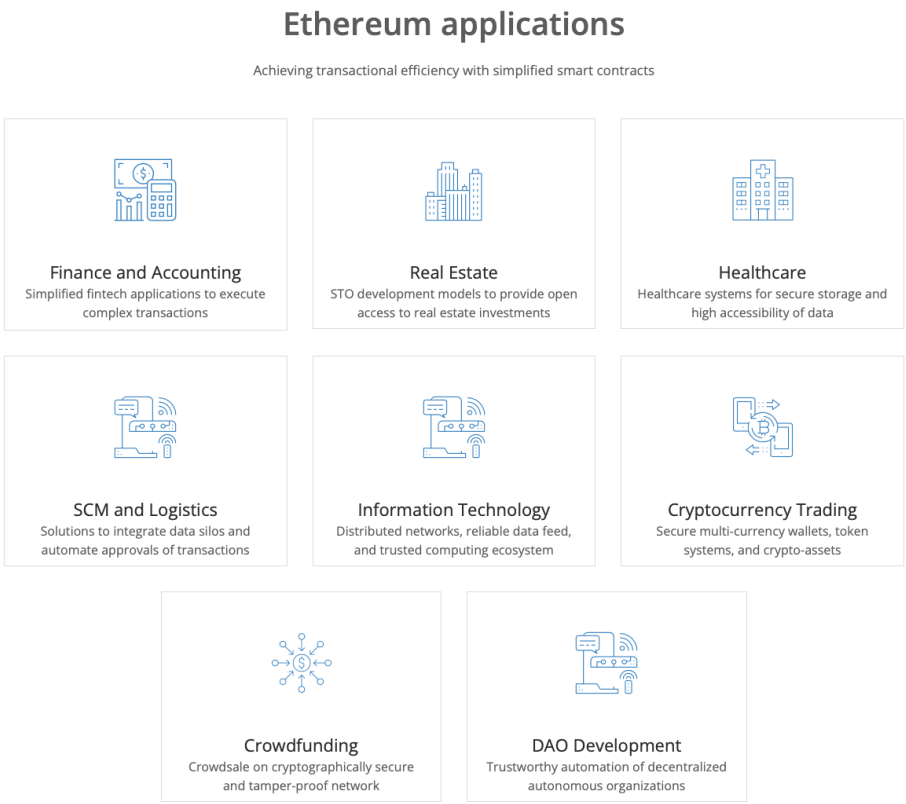
Health Applications

Payment

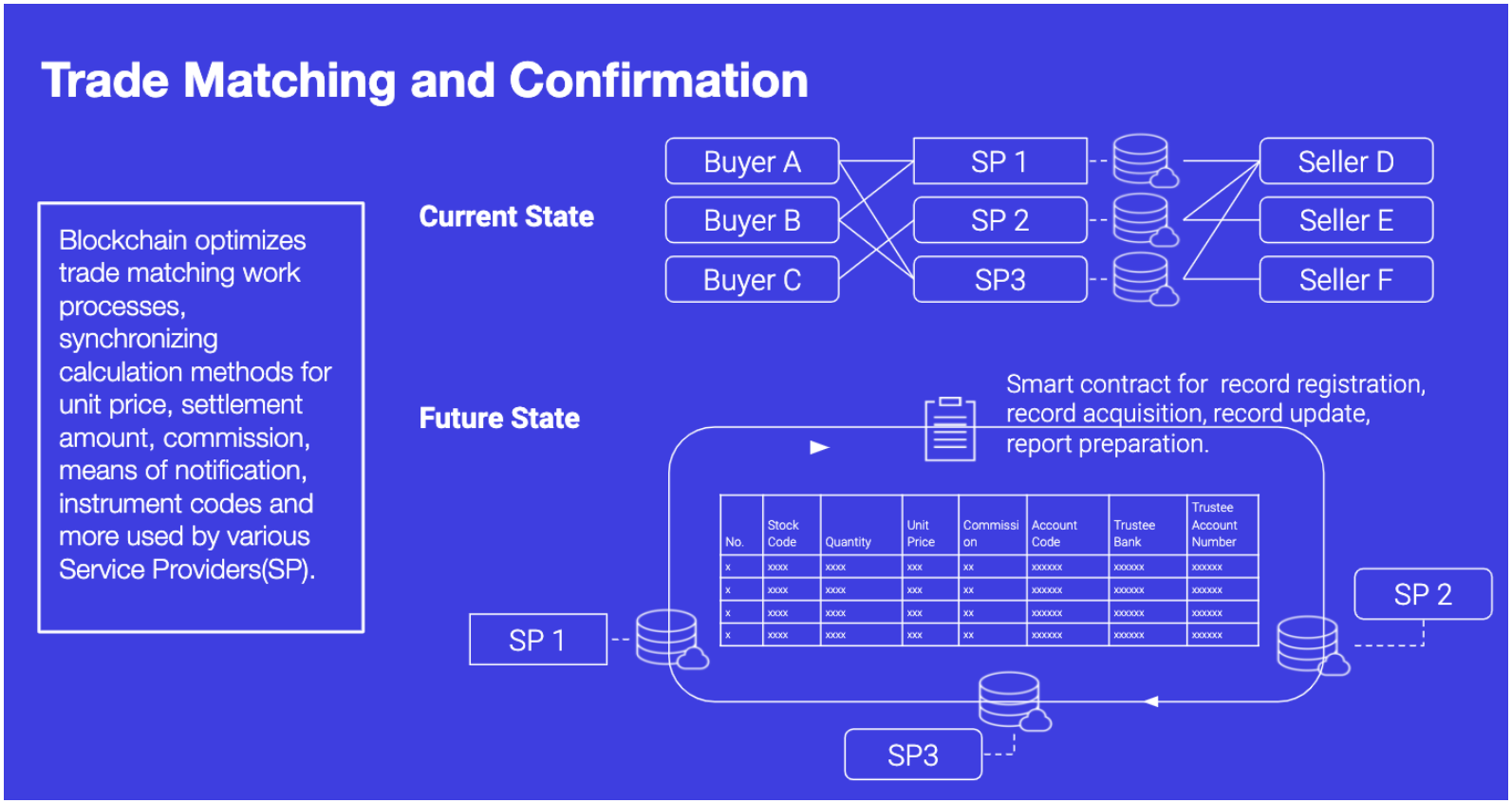
Replacing Escrow

Digital Identity Management

…



Ethereum for Trading Market



Quorum – Enterprise Ethereum

Quorum is a joint product of JP Morgan Chase along with Ethereum Enterprise Alliance

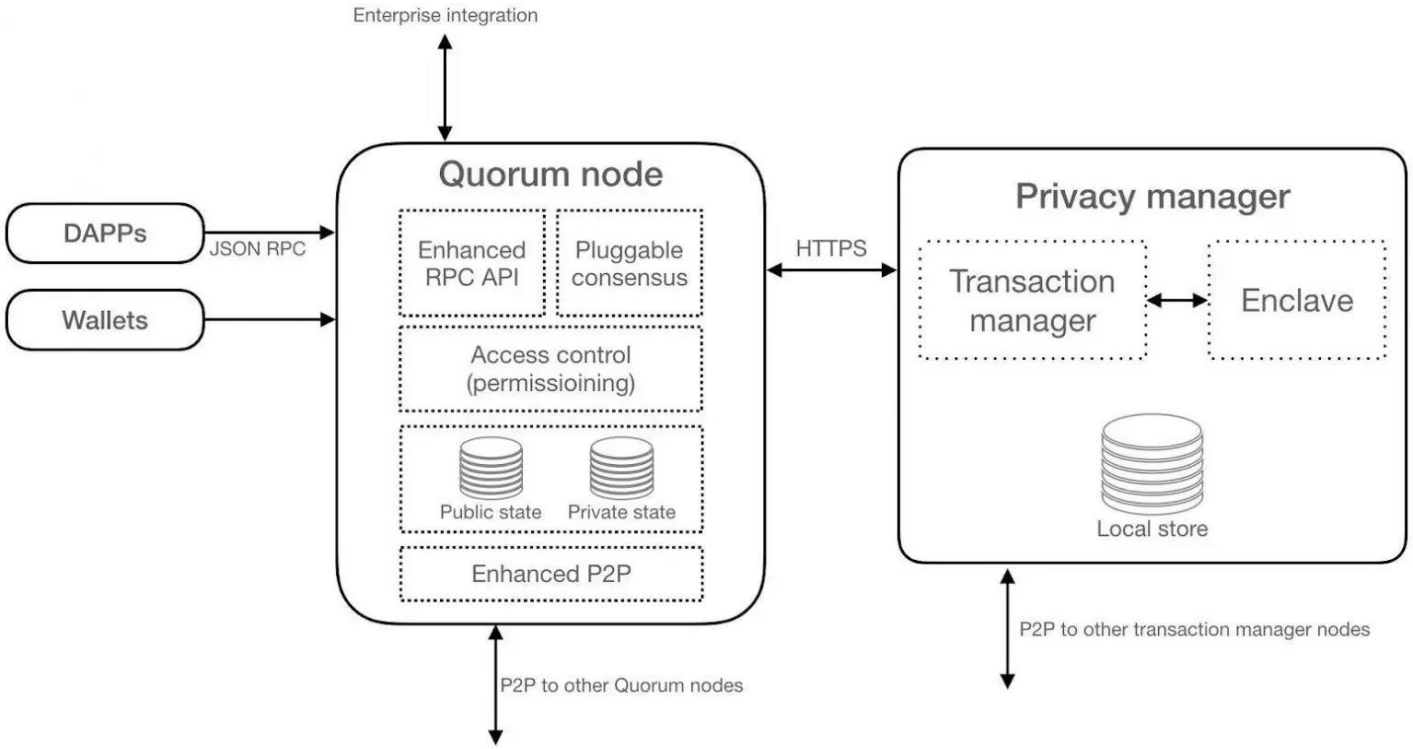
Quorum is an Ethereum-based distributed ledger protocol that has been developed to provide industries such as finance, supply chain, retail, real estate, etc. with a permissioned implementation of Ethereum that supports transaction and contract privacy.

The primary features of Quorum, and therefore extensions over public Ethereum, are:

- Enhanced Transaction and contract privacy
- Multiple voting-based consensus mechanisms
- Network/Peer permissions management
- Higher performance

Quorum – Enterprise Ethereum

High Level Architecture



Quorum Node

The Quorum node is a lightweight fork of geth.

As such, Quorum is regularly updated inline with geth releases to keep up with the latest improvements.

- The Peer to Peer (P2P) layer has been modified to only allow connections to/from permissioned nodes.
- The block generation logic has been modified to replace the 'global state root' check with a new 'global public state root'.
- The block validation logic has been modified to replace the 'global state root' in the block header with the 'global public state root'
- Transaction creation has been modified to allow for Transaction data to be replaced by hashes of the encrypted payloads in order to preserve private data where required
- The pricing of Gas has been removed, although the Gas itself remains.
- The State Patricia trie has been split into two: a public state trie and a private state trie.

Privacy Manager

The privacy manager component (private transaction manager) is responsible for providing transaction privacy on the Quorum network.

It consists of two sub elements, namely the transaction manager and enclave.

Transaction manager

- Quorum's Transaction Manager is responsible for Transaction privacy
- It stores and allows access to encrypted transaction data, exchanges encrypted payloads with other participant's Transaction Managers but does not have access to any sensitive private keys
- a restful and stateless service which is primarily responsible for the following operations.
  - Automatic discovery of the other transaction manager nodes on the network
  - Exchanges encrypted payloads with other nodes' transaction managers
  - Stores and allows access to encrypted transaction data
- Constellation Transaction manager is the original Privacy Manager developed n Haskell
- Tessera Transaction manager is an Enterprise transaction manager developed in Java which blend of a distributed key server, PGP encryption and Mail Transfer Agents (MTAs)

Transactions

There are two transaction signing mechanisms in Quorum.

For public transactions, ethereum EIP-155 based transaction signing mechanism is used, and for private transactions, ethereum Homestead based transaction signing mechanism is used.

Enclave

Distributed Ledger protocols typically leverage cryptographic techniques for transaction authenticity, participant authentication, and historical data preservation (i.e. through a chain of cryptographically hashed data.)

To achieve the "separation of concerns" most of the cryptographic operations, including symmetric key generation and data encryption/decryption is delegated to the Enclave

Handles

- Public/Private key access
- Public keys of extra recipients
- Default identity of attached nodes

Characteristics of Quorum

Enhanced Transaction and contract privacy

- Open transactions are similar to Ethereum but when it comes to the private transaction then it is confidential, and the data is not exposed to the public
- Public transactions work normally as in public ethereum where private transactions are enabled through a separate component called private transaction manager (privacy manager).
- It secures the messages by enslaving it

Network/Peer permissions management

- Permissioned system
- Only validated and authorised people can he a part of the network
- Exchange takes place between participants who are pre-approved by a designated authority.

Characteristics of Quorum

Multiple voting-based consensus mechanisms

- Quorum is based on voting consensus mechanism which is also known as QuorumChain.
- The functioning of this consensus mechanism is very simple; it delegates voting rights to others.
- The consensus is achieved with RAFT, PoA or Istanbul BFT consensus algorithms instead of using Proof-of-Work.
- To assign voting rights, QuorumChain makes use of the smart contract and also tracks the status of all the voting nodes.
- Quorum transactions include:
  - Global Transaction Hash
  - Public State root hash
  - Block maker's signature

Higher performance

- As per the development team, the system can easily suffice more than 100 transactions per second which are higher than Bitcoin and Ethereum

Smart Contracts

What is Smart Contract?

Smart Contracts by Nick Szabo in 1994

More recent papers and essays on smart contracts, commercial controls and security.

Smart Contracts

Glossary

A smart contract is a computerized transaction protocol that executes the terms of a contract.

Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs[1].

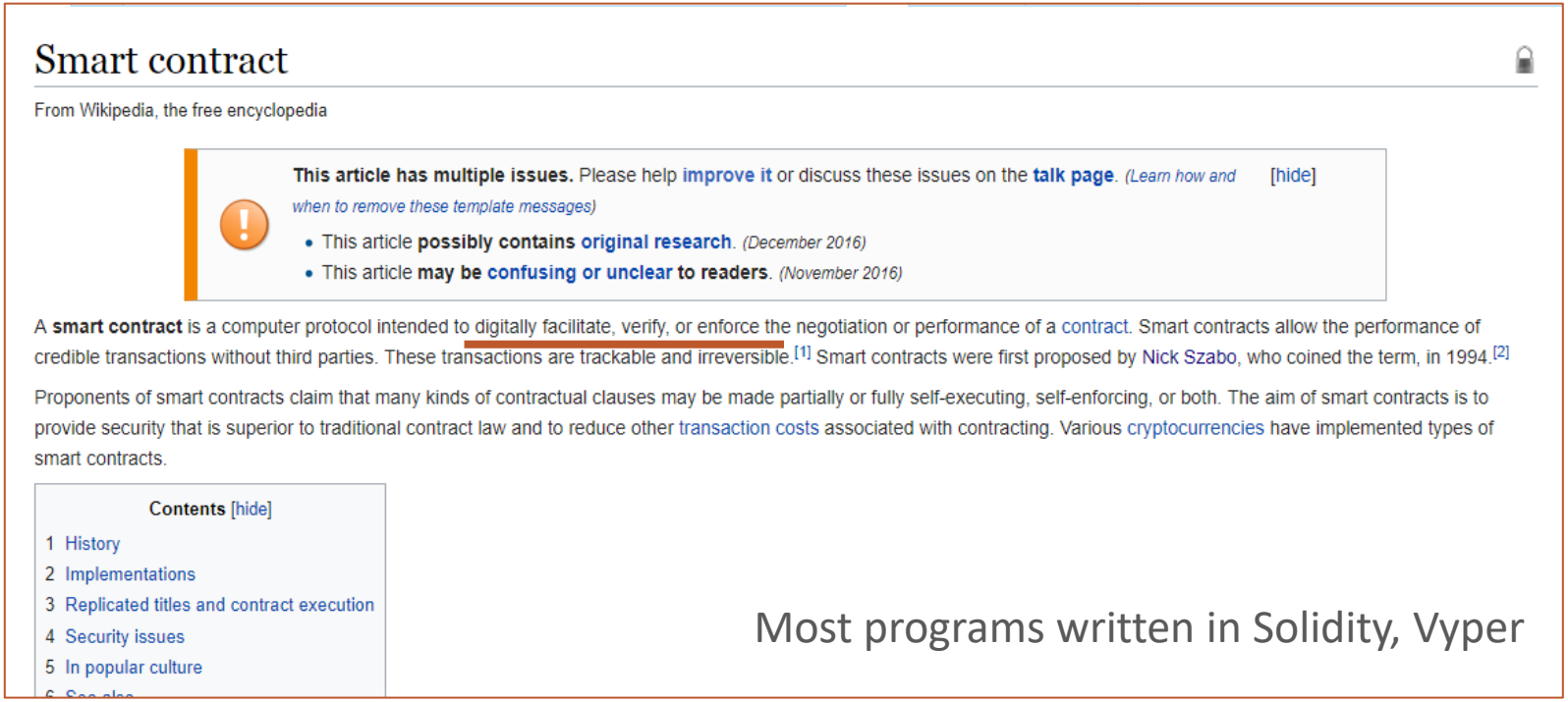
Some technologies that exist today can be considered as crude smart contracts, for example POS terminals and cards, EDI, and agoric allocation of public network bandwidth.

Digital cash protocols[2,3] are fine examples of smart contracts. They enable online payment while honoring the characteristics desired of paper cash: unforgeability, confidentiality, and divisibility. When we take a second glance at digital cash protocols, considering them in the wider context of smart contract design, we see that these protocols can be used to implement a wide variety of electronic bearer securities, not just cash. We also see that to implement a full customer-vendor transaction, we need more than just the digital cash protocol; we need a protocol that guarantees that product will be delivered if payment is made, and vice versa. Current commercial systems use a wide variety of techniques to accomplish this, such as certified mail, face to face exchange, reliance on credit history and collection agencies to extend credit, etc. Smart contracts have the potential to greatly reduce the fraud and enforcement costs of many commercial transactions. Digital cash protocols use several of the rich new building blocks coming out of the fields of cryptography and computer science. Most of these components have not yet been widely exploited to facilitate contractual arrangements, but the potential is vast. These subprotocols include Byzantine agreement, symmetric and asymmetric encryption, digital signatures, blind signatures, cut & choose, bit commitment, multiparty secure computations, secret sharing, oblivious transfer, and multiparty secure computation. All of these except the first are described in [2,3].

The consequences of smart contract design on contract law and economics, and on strategic contract drafting, (and vice versa), have been little explored. As well, I suspect the possibilities for greatly reducing the transaction costs of executing some kinds of contracts, and the opportunities for creating new kinds of businesses and social institutions based on

http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

Smart Contract is a computer program?



Smart contract

From Wikipedia, the free encyclopedia

This article has multiple issues. Please help **improve it** or discuss these issues on the **talk page**. *(Learn how and when to remove these template messages)* [hide]

- This article **possibly contains original research**. *(December 2016)*
- This article **may be confusing or unclear to readers**. *(November 2016)*

A **smart contract** is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract. Smart contracts allow the performance of credible transactions without third parties. These transactions are trackable and irreversible.[1] Smart contracts were first proposed by Nick Szabo, who coined the term, in 1994.[2]

Proponents of smart contracts claim that many kinds of contractual clauses may be made partially or fully self-executing, self-enforcing, or both. The aim of smart contracts is to provide security that is superior to traditional contract law and to reduce other transaction costs associated with contracting. Various cryptocurrencies have implemented types of smart contracts.

Contents [hide]

1 History
2 Implementations
3 Replicated titles and contract execution
4 Security issues
5 In popular culture

Most programs written in Solidity, Vyper

https://en.wikipedia.org/wiki/Smart_contract

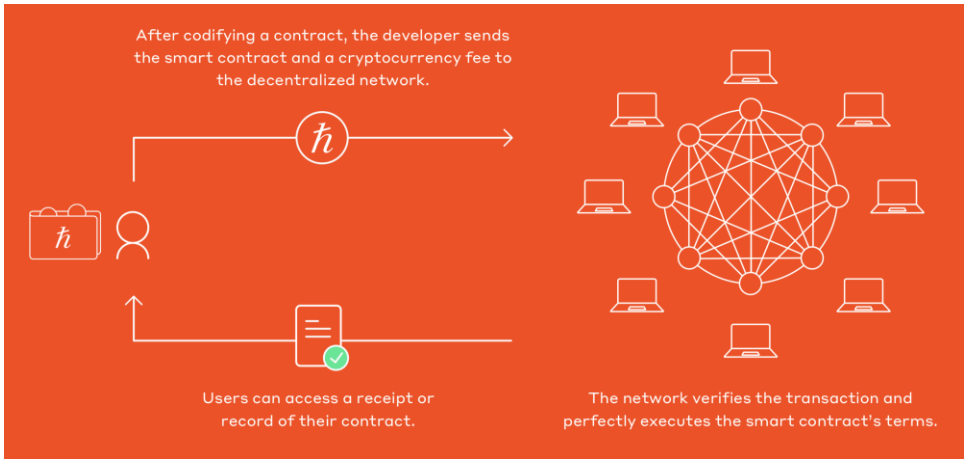
Smart contract = World Computer

What is Smart Contract?

Standard contracts rely on lawyers or government agencies to enforce and execute the described exchange

“Smart” contracts are self-enforcing and self-executing through computer programs

Like vending machines, smart contracts fulfill an agreement without middlemen. Transacting parties simply agree on their terms, put a cryptocurrency coin into the program, and then the smart contract executes the desired digital exchange.



What is a Smart Contract?

If a blockchain is the database, then the smart contract is the application layer that makes much of the promise of block chain technology a reality.

Most conventional contracts have no direct relationship with the computer code that executes them.

This is quite effective when signing up to use a service (e.g. video on demand), but highly challenging when delivering multiple complex services to one user (e.g. updating an address in multiple government department databases).

A smart contract is a special protocol intended to contribute, verify or implement the negotiation or performance of the contract.

Specifically, smart contracts are decentralized, trackable, irreversible, and secure transactions without the need for third parties.

No patching after release

The Goal of Smart Contract

Four important objectives of designing a contract as delineated by Szabo

- Observability
  - The people (or things) involved in the contract have to be able to see that the other party is performing correctly to the terms of the contract, and be able to prove that they themselves are performing correctly to the other party
- Verifiability
  - All parties to a contract need the ability to prove to the chosen arbitrator of the contract that it has either been performed correctly or that one party (or parties) have breached their obligations in the contract.
- Privity
  - The contract should be structured as privately as possible
- Enforceability
  - There needs to be some mechanism of ensuring that things execute correctly, even in the case of one or more parties violating their obligations under the terms of the contract, and as well, the contract should be structured to make the likelihood that enforcement will be needed very unlikely.

How do smart contracts work?

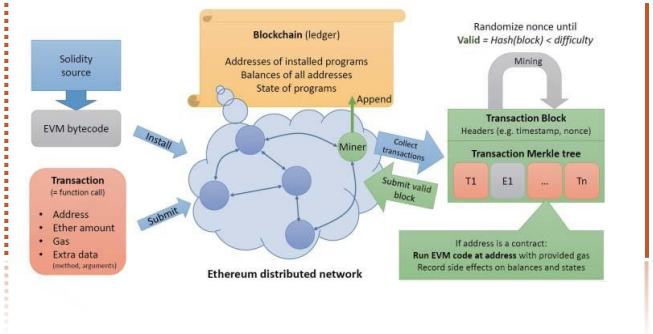
Distributed ledgers are platforms which enable users to securely transact without the need for trusted intermediaries

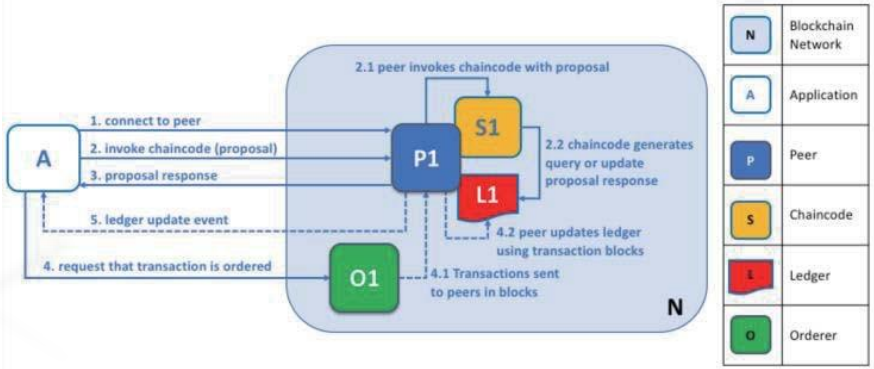
Smart contracts represent the software that executes a vending machine transaction,

A distributed ledger represents the hardware of the vending machine itself.

More specifically, the distributed ledger represents the mechanisms which take in currency, transport the product, and securely (but transparently) carry out the exchange behind tamper-proof locks and bullet-proof glass

The smart contract vending machine analogy goes as such: you input Bitcoin or whatever you are transferring into the vending machine. Once your input satisfies the code within the smart contract, it automatically executes the obligations agreed upon by both parties.





Bitcoin with Smart Contract

Highlights

- A smart contract is a digital agreement which is automatically executed and based on predefined criteria.
- Bitcoin's scripting language enables a variety of smart contracts.
- Layers such as the Lightning Network and sidechains can open more possibilities for smart contracting on Bitcoin.
- The Taproot upgrade will expand the flexibility, efficiency, and utility of smart contracting features for Bitcoin.

A smart contract is a digital agreement which is automatically executed and based on predefined criteria

Script allows users to establish criteria for their bitcoin to be spent, and Bitcoin transactions lock specific amounts of bitcoin to these scripts

Bitcoin with Smart Contract

Script has proven useful for powering the Bitcoin network for over a decade, but it is not Turing Complete, meaning it does not allow for logical loops

- Pay-to-Public-Key-Hash (P2PKH) scripts allow bitcoin to be sent to a Bitcoin address, such that only the owner of the corresponding private key can spend the bitcoin.
- Multi-Signature (multisig) scripts can require any number of signatures, optionally belonging to any number of users.
- Bitcoin transactions can be time locked scripts, meaning they are only valid after a certain time.
- Arbitrarily complex scripts have been made possible by the Pay-to-Script-Hash (P2SH) standard, which was extended to include P2WSH as part of the SegWit upgrade.
- Bitcoin's Taproot upgrade will introduce a new script type called Pay-to-Taproot (P2TR), which will unite the functionality of P2PKH and P2SH scripts, allowing bitcoin to be sent to both a public key and arbitrary scripts.

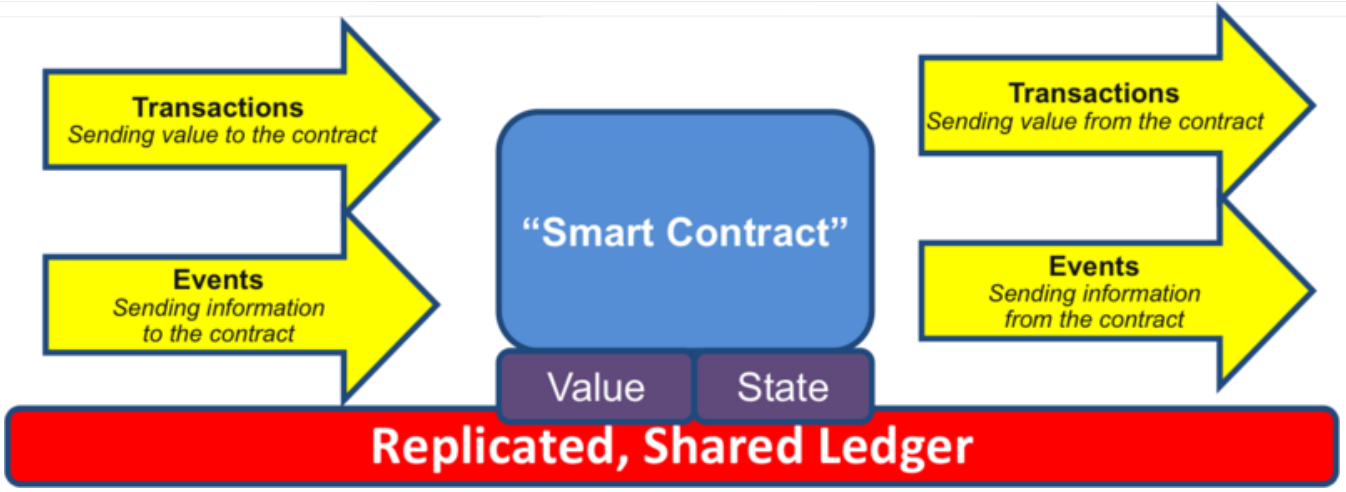
Smart Contract = Code

A smart contract is an agreement between two people in the form of computer code. They run on the blockchain, so they are stored on a public database and cannot be changed.

The transactions that happen in a smart contract processed by the blockchain, which means they can be sent automatically without a third party. This means there is no one to rely on!

The transactions only happen when the conditions in the agreement are met — there is no third party, so there are no issues with trust.

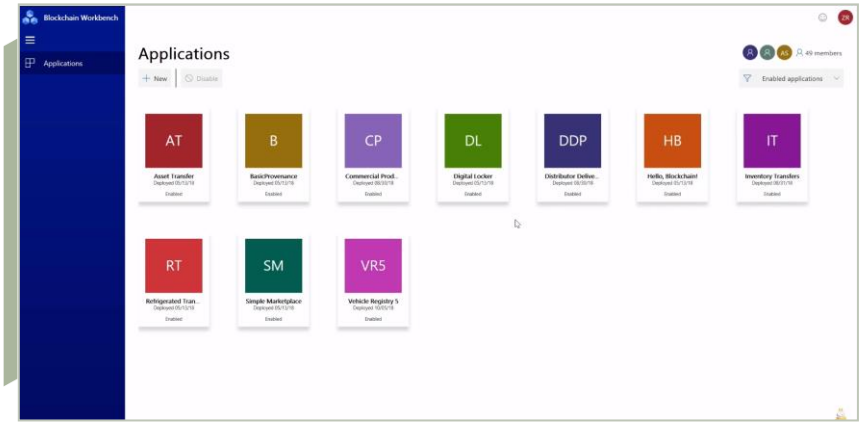
Smart Contract

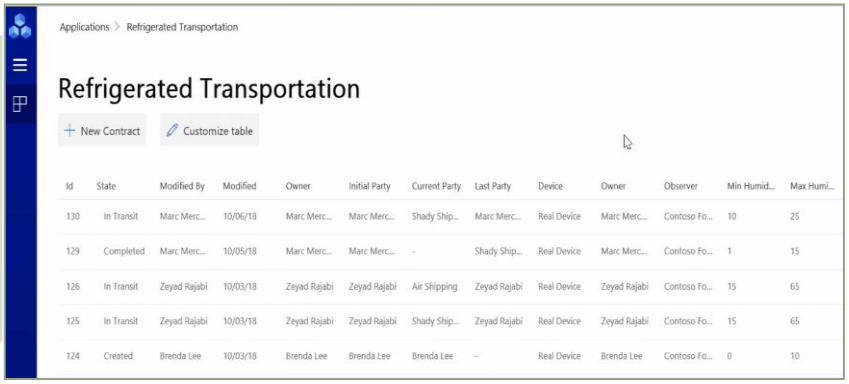


https://en.bitcoinwiki.org/wiki/Contract

- Transaction: each transaction typically executes one public function.
- Global variables: kept in persistent storage across different transactions.
- Local variables (not in the example): temporarily used within the current transaction.
- Payment statement: contract external calls.

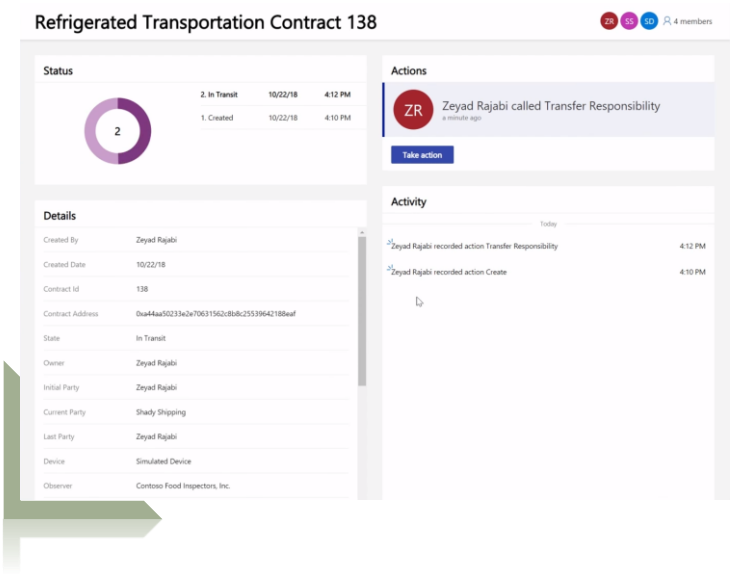
Example of Smart Contract

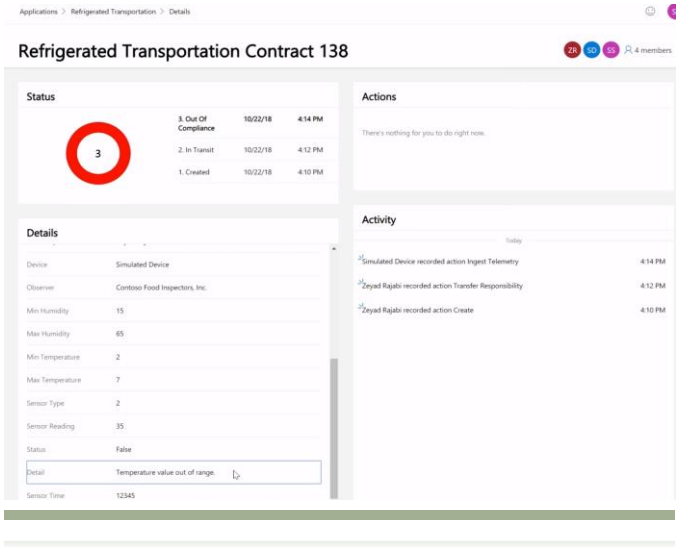


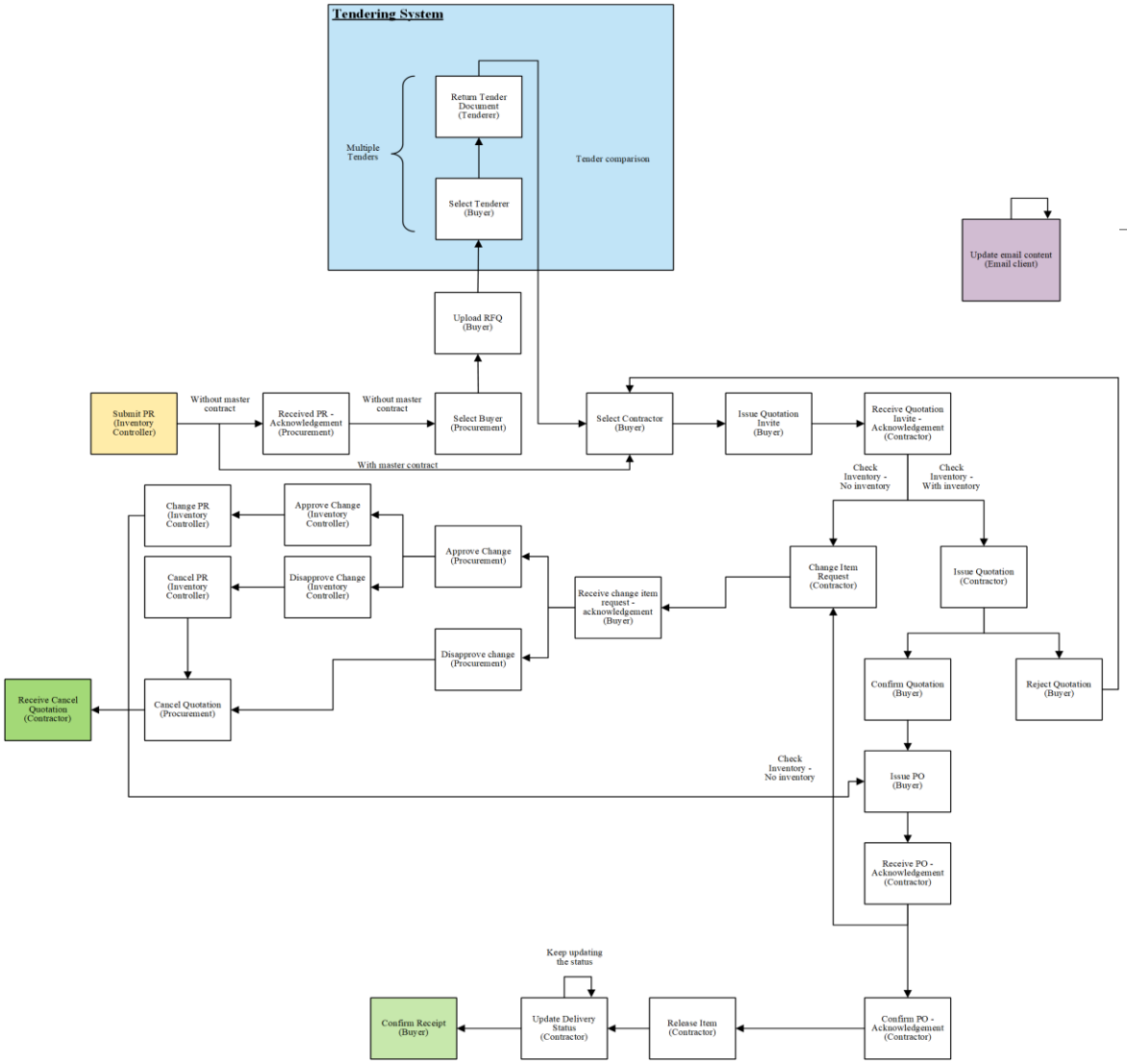


Tool: Microsoft Azure Blockchain Workbench

Example of Smart Contract







Real Case of Smart Contract

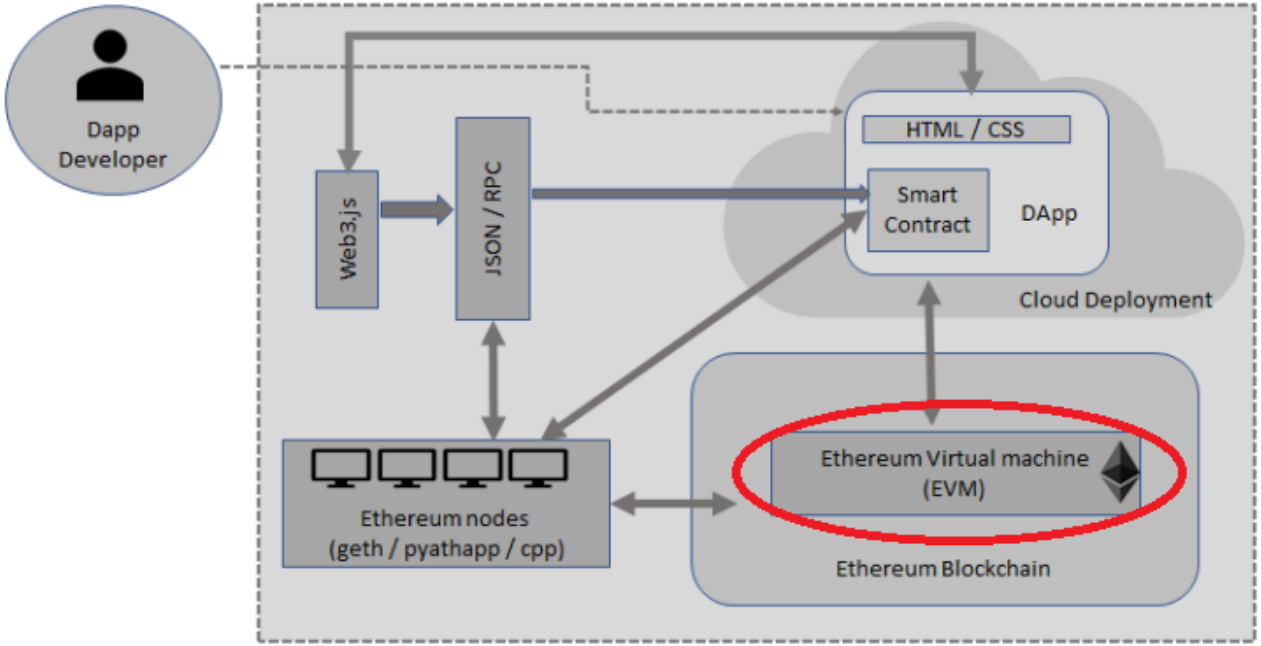
COMPONENT-BUY-AND-TEST PROCESS

DIFFERENT ROLE OF PEOPLE, EG:

INVENTORY CONTROLLER, PROCUREMENT,

BUYER, TENDERER, FORWARDER

Ethereum Virtual Machine



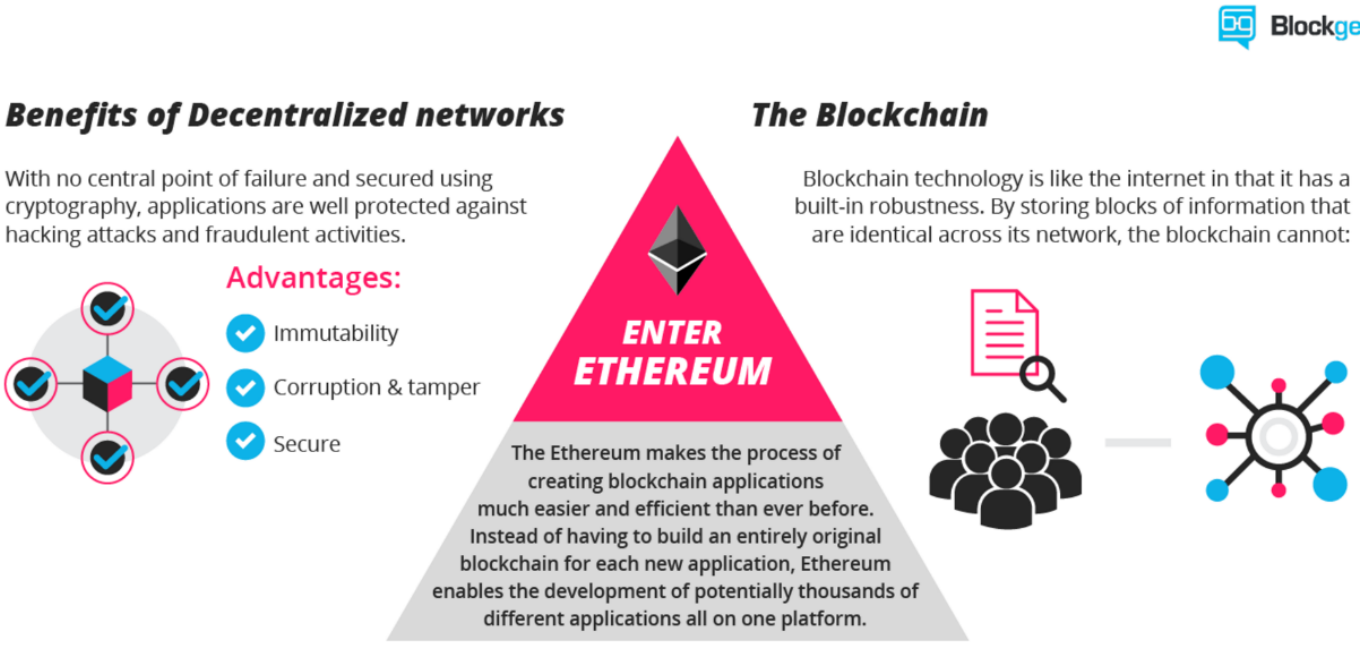
Ethereum Virtual Machine (EVM) is a Turing complete software that runs on the

Smart contracts are created with a programming language called Solidity, similar to Javascript

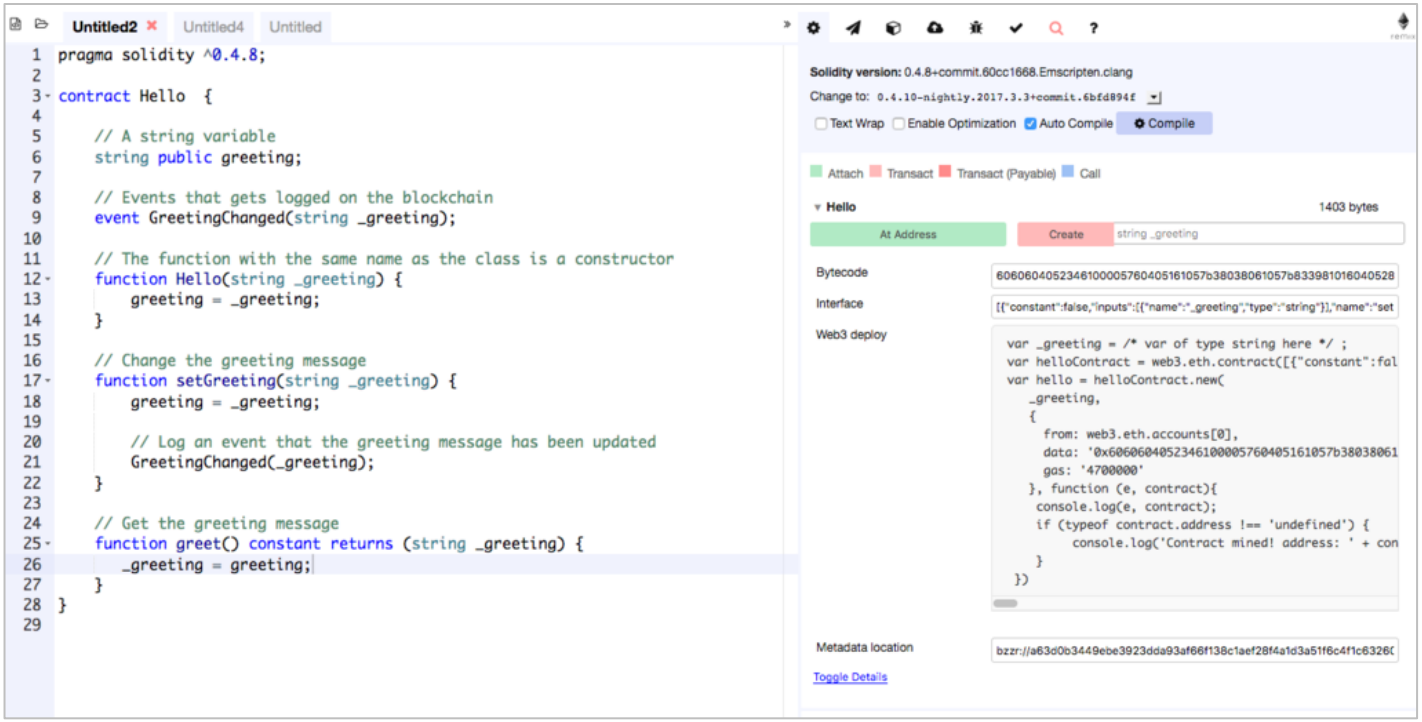
An Ethereum Virtual Machine is a computer that all full nodes in the Ethereum network agree to run, the reason why the EVM is referred to as the heart of the Ethereum Blockchain.

A consensus is required for agreeing how every code or data functions on the Ethereum network. Every node in the Ethereum network runs on the EVM in order to maintain a single consensus across the blockchain.

Ethereum Recap



Development



Development platform on Remix is available at http://remix.ethereum.org

Smart Contracts Language

Solidity Programming Language

- Solidity is a high-level language whose syntax is similar to that of JavaScript, and it is designed to compile the code for the Ethereum Virtual Machine.
- Solidity is a statically typed, contract programming language that has similarities to Javascript and C
- Object-oriented programming, each contract contains state variables, functions, and standard data types.

Solidity Program code

The procedure for creating a Contract is mentioned below.

- contract Contract1 {
- }

- Here, the contract name is FirstContract.
  - Every content between the curly braces is the body of the contract. Nothing is in the body of the contract (what it's going to do or what code it will run)
  - Can have string variable, Unsigned integer data type

Functions

- function Function1(string Param1, string Param2) {
- }

- Excerpt From: Sarah Swammy. "Crypto Uncovered". Apple Books.

Control Flow and Function Calls

The basic if, else, else if, for, and while control structures are available in Solidity with the exact same syntax as in C

Functions are declared with the keyword function, a name, a list of arguments, an optional list of modifiers, and an optional return type, in that order. Of these, all are standard in other languages except the modifiers.

Excerpt From: Kedar Iyer. “Building Games with Ethereum Smart Contracts”. Apple Books.

Simple Smart Contract

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Subcurrency Example

```
pragma solidity ^0.4.21;

contract Coin {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    function Coin() public {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Our version of Voting.sol

```
pragma solidity >=0.4.0 <0.6.0;
contract Voting {
 mapping (bytes32 => uint256) public votesReceived;

 bytes32[] public candidateList;

 constructor(bytes32[] memory candidateNames) public {
  candidateList = candidateNames;
 }

// This function returns the total votes a candidate has received so far
 function totalVotesFor(bytes32 candidate) view public returns (uint256) {
  require(validCandidate(candidate));
  return votesReceived[candidate];
 }

// This function increments the vote count for the specified candidate. This
 // is equivalent to casting a vote
 function voteForCandidate(bytes32 candidate) public {
  require(validCandidate(candidate));
  votesReceived[candidate] += 1;
 }
 function validCandidate(bytes32 candidate) view public returns (bool) {
  for(uint i = 0; i < candidateList.length; i++) {
   if (candidateList[i] == candidate) {
    return true;
   }
  }
  return false;
 }
}
```

Simple Smart Contract

```
pragma solidity >=0.4.0 <0.6.0;;

import "./ECRecovery.sol";


contract Voting {
  using ECRecovery for bytes32;
  mapping (bytes32 => uint8) public votesReceived;
  mapping(bytes32 => bytes32) public candidateHash;
  mapping(address => bool) public voterStatus;
  mapping(bytes32 => bool) public validCandidates;
  function Voting(bytes32[] _candidateNames, bytes32[]
_candidateHashes) public {
    for(uint i = 0; i < _candidateNames.length; i++) {
      validCandidates[_candidateNames[i]] = true;
      candidateHash[_candidateNames[i]] = _candidateHashes[i];
    }
  function totalVotesFor(bytes32 _candidate) view public
returns (uint8) {
    require(validCandidates[_candidate]);
    return votesReceived[_candidate];
  }
  function voteForCandidate(bytes32 _candidate, address
_voter, bytes _signedMessage) public {
    require(!voterStatus[_voter]);
    bytes32 voteHash = candidateHash[_candidate];
    address recoveredAddress =
voteHash.recover(_signedMessage);
    require(recoveredAddress == _voter);
    require(validCandidates[_candidate]);
    votesReceived[_candidate] += 1;
    voterStatus[_voter] = true;
  }
}
```

Restrict Smart Contract to run in Ethereum 2.0 chain

Currently Eth2 (the Beacon Chain) and Eth1 (the Ethereum Mainnet) they are different chain.

'Eth1' is now the 'execution layer', which handles transactions and execution.

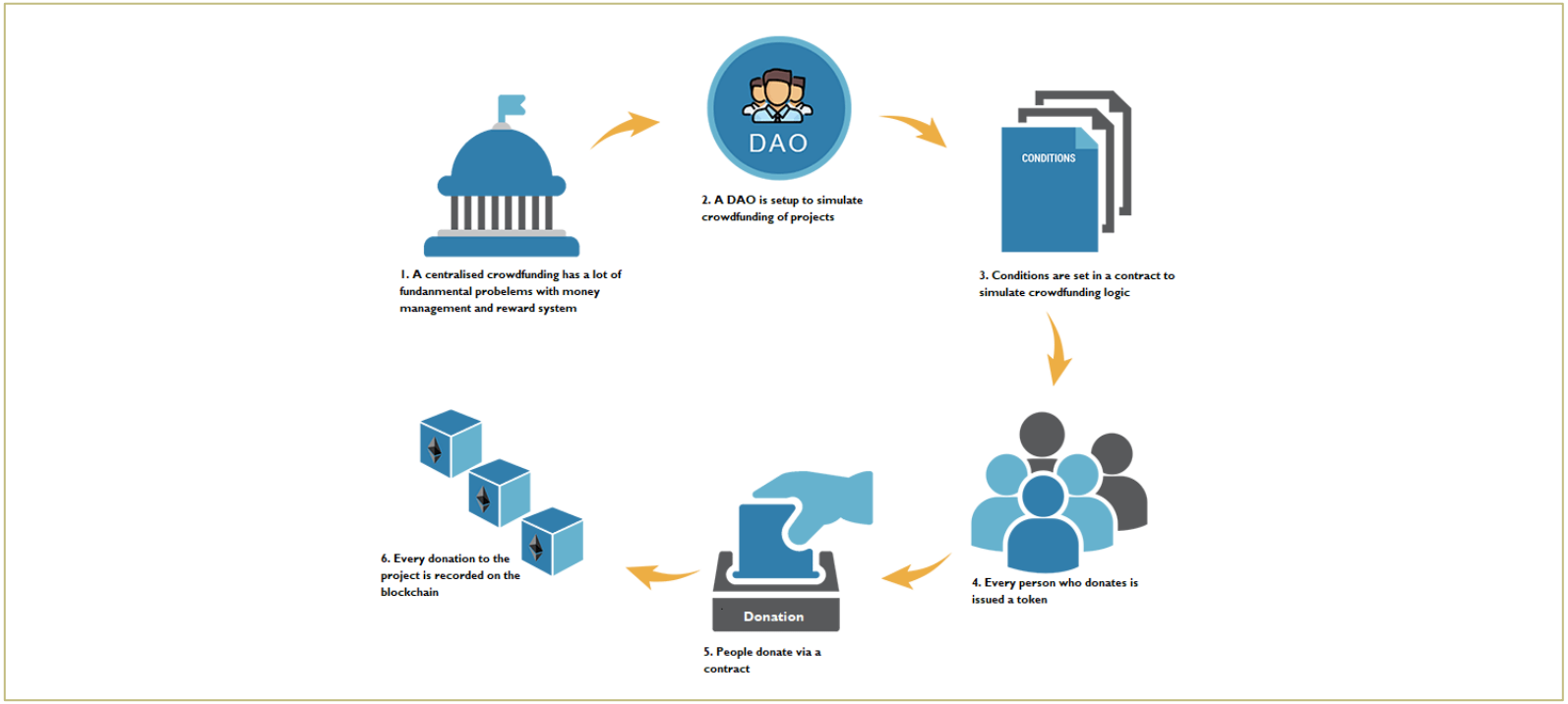
'Eth2' is now the 'consensus layer', which handles proof-of-stake consensus.

To restrict the DApp execute on Eth2, they just need to deploy it to Eth2 chain instead of the mainnet.

The premise is the DApp is design for Eth2.

Decentralized Crowd Funding Use Case (Example)

"Kickstarter" organization provide projects with the public exposure needed for donations towards their project to get it up an running.



https://www.edureka.co/blog/ethereum-tutorial-with-smart-contracts/

Ethereum Standards EIP

This is a standard interface for the token contract that details the contract properties, functions, function arguments, and function return types.

Ethereum Request for Comment (ERC)

- This technical standard dictates a number of rules and actions that an Ethereum token or smart contract must follow and steps to be able to implement it.

Fungible means “replaceable” or “interchangeable.”

A Non-Fungible Token means that it’s a unique token that has no other token like it.

Backend ERC721

```
pragma solidity ^0.4.17;
import "./ERC721/ERC721Token.sol";

/** * @title Repository of ERC721 Deeds * This contract contains the list of deeds registered by users. * This is a demo to show how tokens (deeds) can be minted and added * to the repository. */

contract DeedRepository is ERC721Token {
  ◦ function registerDeed(uint256 _tokenId, string _uri) public {

_mint(msg.sender, _tokenId)
  ◦ function addDeedMetadata(uint256 _tokenId, string _uri) public returns(bool){

_setTokenURI(_tokenId, _uri);
  ◦ event DeedRegistered(address _by, uint256 _tokenId);

}
```

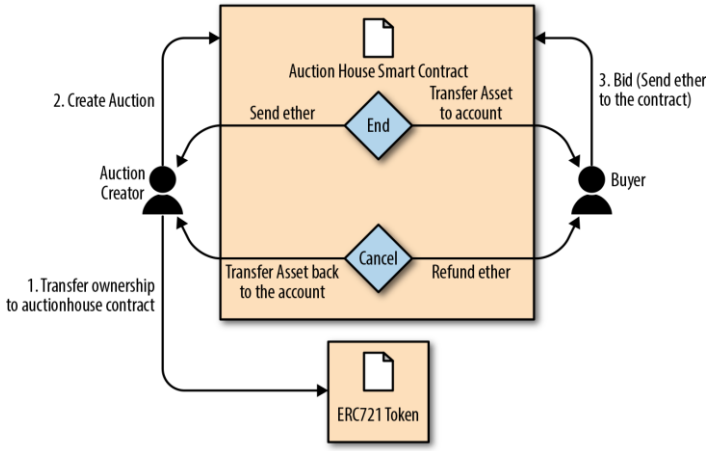
The DeedRepository contract is a straightforward implementation of an ERC721-compatible token.

https://github.com/ethereumbook/ethereumbook/blob/develop/code/auction_dapp/backend/contracts/DeedRepository.sol

Auction DApp Example

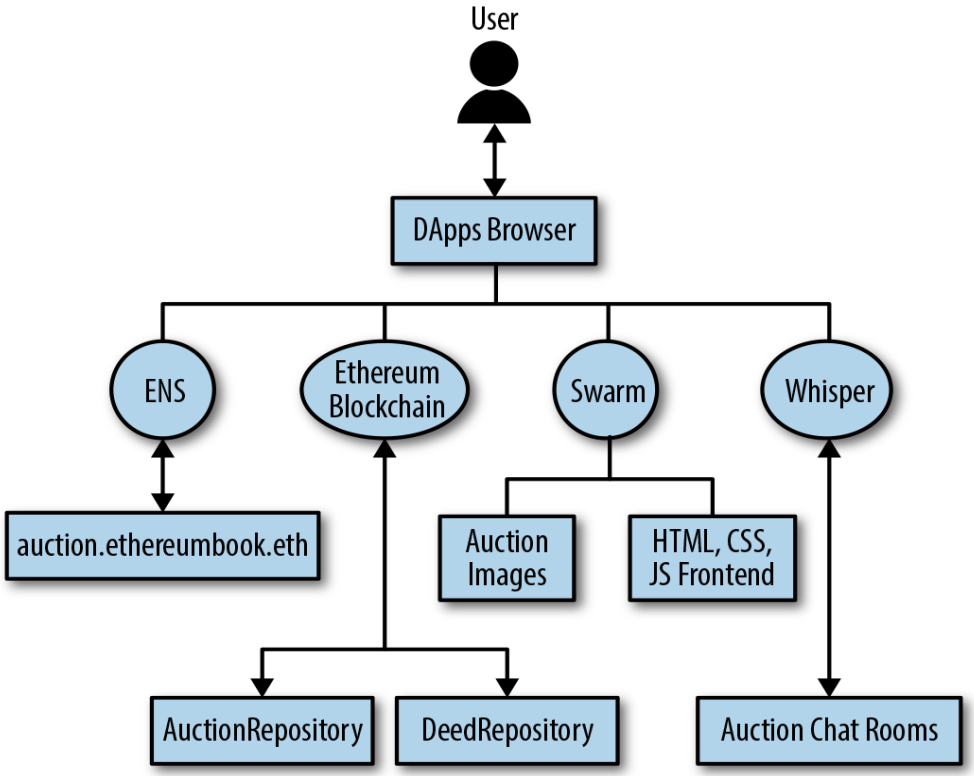
The main components of our Auction DApp are:

- A smart contract implementing ERC721 non-fungible "deed" tokens (DeedRepository)
- A smart contract implementing an auction (AuctionRepository) to sell the deeds
- A web frontend using the Vue/Vuetify JavaScript framework
- The *web3.js* library to connect to Ethereum chains (via MetaMask or other clients)
- A Swarm client, to store resources such as images
- A Whisper client, to create per-auction chat rooms for all participants



https://github.com/ethereumbook/ethereumbook/tree/develop/code/auction_dapp

Auction DApp Architecture



Backend AuctionRepository

contract AuctionRepository {

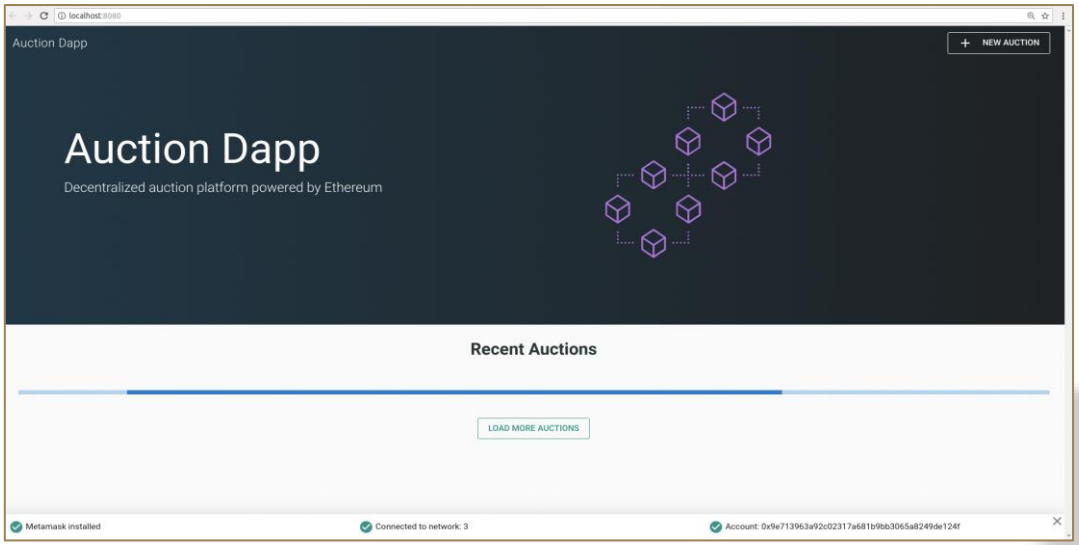
- *// Array with all auctions*
- Auction[] **public** auctions;
- *// Mapping from auction index to user bids*
- **mapping**(**uint256** => Bid[]) **public** auctionBids;
- *// Mapping from owner to a list of owned auctions*
- **mapping**(**address** => **uint**[]) **public** auctionOwner;
- *// Bid struct to hold bidder and amount*
- **struct** Bid {
  - **address from**;
  - **uint256** amount;
- }
- *// Auction struct which holds all the required info*
- **struct** Auction {
  - **string** name;
  - **uint256** blockDeadline;
  - **uint256** startPrice;
  - **string** metadata;
  - **uint256** deedId;
  - **address** deedRepositoryAddress;
  - **address** owner;
  - **bool** active;
  - **bool** finalized;
- }

Functions

- getCount()
- getBidsCount(**uint** _auctionId)
- getAuctionsOf(**address** _owner)
- getCurrentBid(**uint** _auctionId)
- getAuctionsCountOfOwner(**address** _owner)
- getAuctionById(**uint** _auctionId)
- createAuction(**address** _deedRepositoryAddress, **uint256** _deedId, **string** _auctionTitle, **string** _metadata, **uint256** _startPrice, **uint** _blockDeadline)
- approveAndTransfer(**address** _from, **address** _to, **address** _deedRepositoryAddress, **uint256** _deedId)
- cancelAuction(**uint** _auctionId)
- finalizeAuction(**uint** _auctionId)
- bidOnAuction(**uint** _auctionId)

https://github.com/ethereumbook/ethereumbook/blob/develop/code/auction_dapp/backend/contracts/AuctionRepository.sol

Frontend Auction DApp




```
frontend/
|-- build
|   |-- build.js
|   |-- check-versions.js
|   |-- logo.png
|   |-- utils.js
|   |-- vue-loader.conf.js
|   |-- webpack.base.conf.js
|   |-- webpack.dev.conf.js
|   `-- webpack.prod.conf.js
|-- config
|   |-- dev.env.js
|   |-- index.js
|   `-- prod.env.js
|-- index.html
|-- package.json
|-- package-lock.json
|-- README.md
|-- src
|   |-- App.vue
|   |-- components
|   |   |-- Auction.vue
|   |   `-- Home.vue
|   |-- config.js
|   |-- contracts
|   |   |-- AuctionRepository.json
|   |   `-- DeedRepository.json
|   |-- main.js
|   |-- models
|   |   |-- AuctionRepository.js
|   |   |-- ChatRoom.js
|   |   `-- DeedRepository.js
|   `-- router
|       `-- index.js
```

https://github.com/ethereumbook/ethereumbook/tree/develop/code/auction_dapp/frontend

Smart contract interface

```
pragma solidity ^0.4.16;


interface token {
    function transfer(address receiver, uint amount);
}

contract Crowdsale {
    address public beneficiary;
    uint public fundingGoal;
    uint public amountRaised;
    uint public deadline;
    uint public price;
    token public tokenReward;
    mapping(address => uint256) public balanceOf;
    bool fundingGoalReached = false;
    bool crowdsaleClosed = false;
    event GoalReached(address recipient, uint totalAmountRaised);
    event FundTransfer(address backer, uint amount, bool isContribution);
```

Smart contract Constructor

```
/**
* Constrctor function
* Setup the owner
*/
function Crowdsale(
  ◦ address ifSuccessfulSendTo; // the address of the owner when funding is successful
  ◦ uint fundingGoalInEthers; // target amount to raise
  ◦ uint durationInMinutes; //given time
  ◦ uint etherCostOfEachToken; //cost of equity in ether
  ◦ address addressOfTokenUsedAsReward; //token address
) {
  ◦ beneficiary = ifSuccessfulSendTo;
  ◦ fundingGoal = fundingGoalInEthers * 1 ether;
  ◦ deadline = now + durationInMinutes * 1 minutes;
  ◦ price = etherCostOfEachToken * 1 ether;
  ◦ tokenReward = token(addressOfTokenUsedAsReward);
}
```

Smart contract Fallback function

```
/**
* Fallback function
*
* The function without name is the default function that is called whenever anyone sends funds to a contract
*/
function () payable {
    require(!crowdsaleClosed);
    uint amount = msg.value;
    balanceOf[msg.sender] += amount;
    amountRaised += amount;
    tokenReward.transfer(msg.sender, amount / price);
    FundTransfer(msg.sender, amount, true);
}
```

Smart contract checkGoalReached function

```
modifier afterDeadline() { if (now <= deadline) _; }


/**
* Check if goal was reached
*
* Checks if the goal or time limit has been reached and ends the campaign
*/
function checkGoalReached() afterDeadline {
    if (amountRaised >= fundingGoal){
    fundingGoalReached = true;
    GoalReached(beneficiary, amountRaised);
}
crowdsaleClosed = true;
}
```

Smart contract safeWithdrawal function

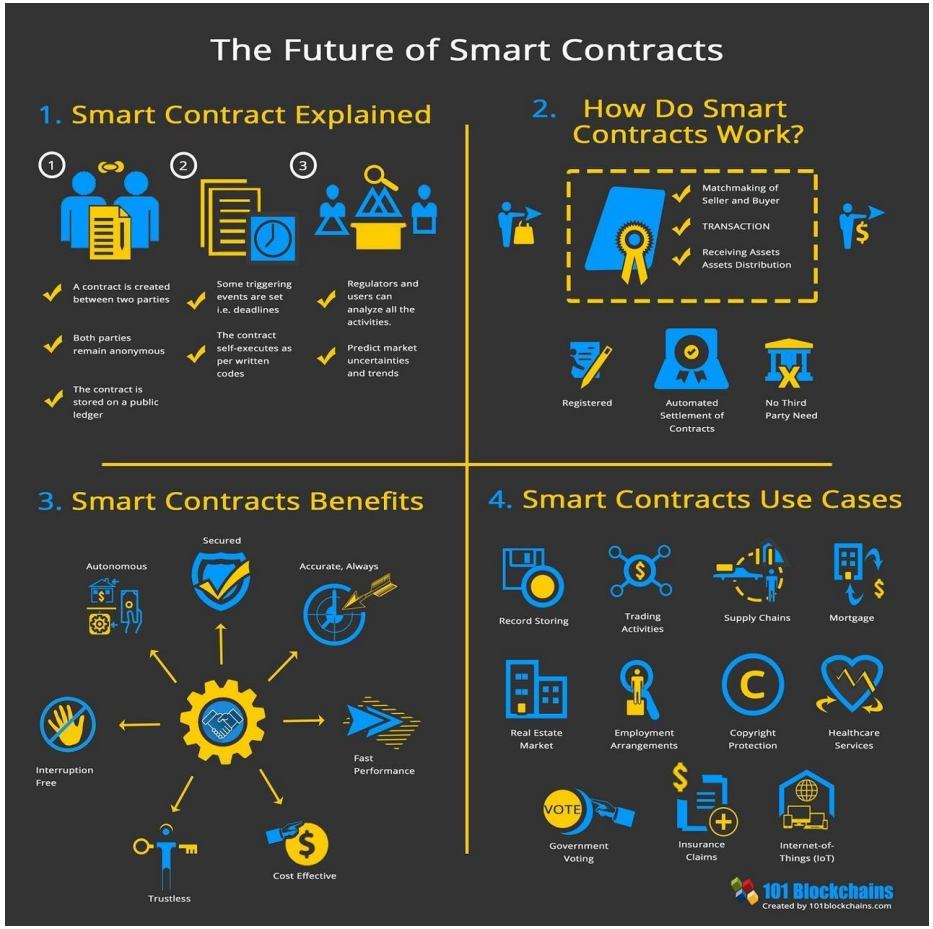
```
/**
* Withdraw the funds
* Checks to see if goal or time limit has been reached, and if so, and the funding goal was reached,
* sends the entire amount to the beneficiary. If goal was not reached, each contributor can withdraw
* the amount they contributed.
*/
function safeWithdrawal() afterDeadline {
    if (!fundingGoalReached) {
        uint amount = balanceOf[msg.sender];
        balanceOf[msg.sender] = 0;
        if (amount > 0) {
            if (msg.sender.send(amount)) {
                FundTransfer(msg.sender, amount, false);
            } else {
                balanceOf[msg.sender] = amount;
            }
        }
}

if (fundingGoalReached && beneficiary == msg.sender) {
    if (beneficiary.send(amountRaised)) {
        FundTransfer(beneficiary, amountRaised, false);
    } else {
        //If we fail to send the funds to beneficiary, unlock funders balance
        fundingGoalReached = false;
        }
    }
}
```

Smart Contract and Languages

	Supports Smart Contracts?	Turing Complete?	Language
Bitcoin	No	n/a	n/a
Corda	Yes	Yes	Kotlin, Java
Ethereum	Yes	Yes	Solidity
Fabric	Yes	Yes	Go, JavaScript
Multichain	No	n/a	n/a
Neo	Yes	Yes	C#, Java
NXT	No	n/a	n/a
Quorum	Yes	Yes	Solidity
Sawtooth	Yes	Yes	Solidity, Go JavaScript, Python

Smart Contracts



Algorithmic Decision Making

Many decisions that could be made by human beings – from interpreting medical images to recommending books or movies – can now be made by computer algorithms with advanced analytic capabilities and access to huge stores of data

Decentralized Autonomous Organization (DAO) – Computer program runs on top of a blockchain and embedded within it. “The most famous DAO project is The DAO (https://daohub.org)”

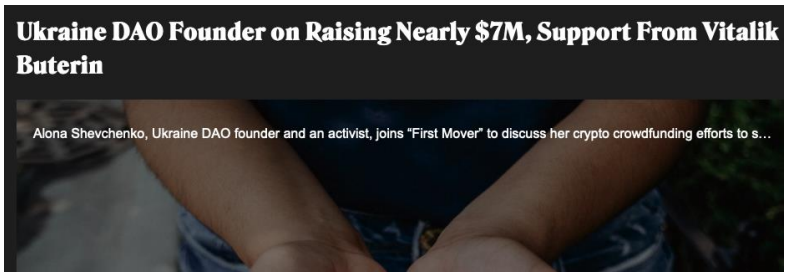
Then algorithm can be built?

Future of DAO

Decentralized Autonomous Organization

- Focus on individual personal
- Record the return of effort into the blockchain
- Record the capital

DAOs are blockchain-based organizations that operate without central authorities.



Regarding this , Founder of Ethereum Vitalik This phenomenon , Put forward DAO1CO The concep t of , **Put his ideal DAO （ Decentralized autonomous organizations ） and 1CO （ First pass i ssue ） Combined with** . Use a smart contract to lock 1CO The raised pass , According to the ex pected project schedule , At each different project progress point , Unlock certain Token, Remuner ation as a project developer .

This way you can **Effectively prevent the core team from fraud ——** Because if the team doesn' t do anything , Or the things made have not been recognized by the investor （ Voting method writ ten in advance through smart contract ） , The team won't get any reward , It's time , All investors' funds will be returned to investors .

Just like 20 Years ago **It is predicted that " In the future, all companies will be Internet compa nies " equally** ,20 Today, years later , It can also be predicted **" In the future, all companies will be DAO organization ."** As a creative attempt ,DAO Essentially, **A new type of production relatio nship that subverts the modern company system —— Decentralized partnerships .** Due to th e emergence of blockchain Technology , It brings the possibility of new production relations .