



Ethereum

The World Computer

Ethereum Compared to Bitcoin

Ethereum shares many common elements with other open blockchains:

1. a peer-to-peer network connecting participants
2. a Byzantine fault-tolerant consensus algorithm for synchronization of state updates (a proof-of-work blockchain)
3. the use of cryptographic primitives such as digital signatures and hashes
4. and a digital currency (ether)

Ethereum Compared to Bitcoin

1. Ethereum's purpose is not primarily to be a digital currency payment network.
2. While the digital currency ether is both integral to and necessary for the operation of Ethereum
3. ether is intended as a utility currency to pay for use of the Ethereum platform as the world computer.

Ethereum Compared to Bitcoin

1. Bitcoin has a very limited scripting language
2. Bitcoin's Script language is, intentionally, constrained to simple true/false evaluation of spending conditions
3. Ethereum is designed to be a general-purpose programmable blockchain that runs a virtual machine capable of executing code of arbitrary and unbounded complexity.
4. Ethereum's language is Turing complete, meaning that Ethereum can straightforwardly function as a general-purpose computer.

Ethereum

Components of a Blockchain

Components of a Blockchain

1. A peer-to-peer (P2P) network connecting participants and propagating transactions and blocks of verified transactions, based on a standardized "gossip" protocol
2. Messages, in the form of transactions, representing state transitions
3. A set of consensus rules, governing what constitutes a transaction and what makes for a valid state transition
4. A state machine that processes transactions according to the consensus rules

Components of a Blockchain

5. A chain of cryptographically secured blocks that acts as a journal of all the verified and accepted state transitions
6. A consensus algorithm that decentralizes control over the blockchain, by forcing participants to cooperate in the enforcement of the consensus rules
7. A game-theoretically sound incentivization scheme (e.g., proof-of-work costs plus block rewards) to economically secure the state machine in an open environment
8. One or more open source software implementations of the above ("clients")

Components of a Blockchain

1. In Bitcoin, the reference implementation is developed by the *Bitcoin Core* open source project and implemented as the *bitcoind* client.
2. In Ethereum, rather than a reference implementation there is a *reference specification*, a mathematical description of the system in the Yellow Paper.
3. There are a number of clients, which are built according to the reference specification.

Ethereum

Four Stages of Development

Ethereum's Four Stages of Development

1. The four main development stages are codenamed
 - a. Frontier
 - b. Homestead
 - c. Metropolis
 - d. Serenity
2. The intermediate hard forks are codenamed
 - a. Ice Age
 - b. DAO
 - c. Tangerine Whistle
 - d. Spurious Dragon
 - e. Byzantium
 - f. Constantinople

Ethereum

General-Purpose Blockchain

A General-Purpose Blockchain

1. Ethereum is distributed state machine.
2. Ethereum tracks state of currency ownership, and state transitions of a general-purpose data store, i.e., a store that can hold any data expressible as a *key-value tuple*
3. Ethereum has memory that stores both code and data, and it uses the Ethereum blockchain to track how this memory changes over time.
4. Ethereum can load code into its state machine and *run* that code, storing the resulting state changes in its blockchain
5. Two of the critical differences from most general-purpose computers are that Ethereum state changes are governed by the rules of consensus and the state is distributed globally

Ethereum

Turing Completeness

Turing Completeness

1. Ethereum can compute any algorithm that can be computed by any Turing machine, given the limitations of finite memory.
2. Groundbreaking innovation is to combine the general-purpose computing architecture of a stored-program computer with a decentralized blockchain, thereby creating a distributed single-state (singleton) world computer.
3. Ethereum programs run "everywhere," yet produce a common state that is secured by the rules of consensus.
4. The fact that Ethereum is Turing complete means that any program of any complexity can be computed by Ethereum

Turing Completeness

5. Flexibility of computing any complexity brings some security and resource management problems.
6. Turing proved that you cannot predict whether a program will terminate by simulating it on a computer
7. Turing-complete systems can run in "infinite loops," a term used (in oversimplification) to describe a program that does not terminate.

Ethereum

GAS

Turing Completeness and GAS

1. To answer this challenge, Ethereum introduces a metering mechanism called *gas*.
2. As the EVM executes a smart contract, it carefully accounts for every instruction (computation, data access, etc.).
3. Each instruction has a predetermined cost in units of gas.
4. Transaction that triggers the execution of a smart contract must include an amount of gas that sets the upper limit of what can be consumed
5. EVM will terminate execution if the amount of gas consumed by computation exceeds the gas available in the transaction.

Ethereum

How does one get GAS?

How does one get gas

1. You won't find gas on any exchanges.
2. Only be purchased as part of a transaction, and can only be bought with ether.
3. Ether needs to be sent along with a transaction and it needs to be explicitly earmarked for the purchase of gas, along with an acceptable gas price.
4. Just like at the pump, the price of gas is not fixed.
5. Gas is purchased for the transaction, the computation is executed, and any unused gas is refunded back to the sender of the transaction.

Ethereum

DApps

Decentralized Applications (DApps)

DApp is a web application that is built on top of open, decentralized, peer-to-peer infrastructure services.

A DApp is composed of at least:

- Smart contracts on a blockchain
- A web frontend user interface

In addition, many DApps include other decentralized components, such as:

- A decentralized (P2P) storage protocol and platform
- A decentralized (P2P) messaging protocol and platform

Ethereum

Currency Units

Ethereum Currency Units

1. Ethereum's currency unit is called ether
2. It identified as "ETH" or with the symbolsΞ (from the Greek letter "Xi" or, less often, ♦)
3. For example: 1 ether, or 1 ETH, orΞ1, or♦1



Ether Smallest Unit = WEI



ethereum

1. Ether is subdivided into smaller units, wei.
2. One ether is 1 quintillion wei (10^{18} or 1,000,000,000,000,000,000).
3. Ethereum is the system, ether is the currency.
4. When you transact 1 ether, the transaction encodes 1,000,000,000,000,000 wei as the value.

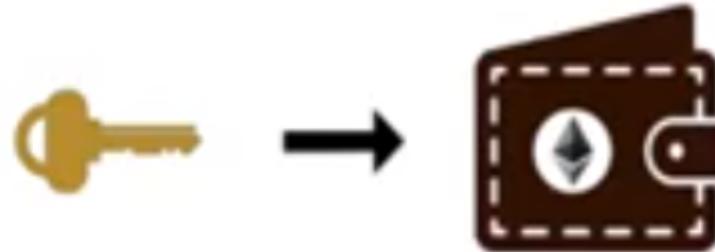
Value (in wei)	Exponent	Common name	SI name
1	1	wei	Wei
1,000	10^3	Babbage	Kilowei or femtoether
1,000,000	10^6	Lovelace	Megawei or picoether
1,000,000,000	10^9	Shannon	Gigawei or nanoether
1,000,000,000,000	10^{12}	Szabo	Microether or micro
1,000,000,000,000,000	10^{15}	Finney	Milliether or milli
1,000,000,000,000,000,000	10^{18}	Ether	Ether
1,000,000,000,000,000,000,000	10^{21}	Grand	Kiloether
1,000,000,000,000,000,000,000,000	10^{24}		Megaether
1,000'000'000'000'000'000'000'000	10^{27}		Myriaether
1,000'000'000'000'000'000'000'000	10^{30}	Gigaether	Kiloether

Ethereum

Ethereum Wallets

Types of Wallet

1. Mobile wallet,
2. Desktop wallet, and
3. Web-based wallet.



Generally the more popular a wallet application is, the more trustworthy it is likely to be.

It is good practice to avoid "putting all your eggs in one basket" and have your Ethereum accounts spread across a couple of wallets.

Some good starter wallets

MetaMask

1. Browser extension wallet that runs in your browser.
2. It is easy to use and convenient for testing,
3. as it is able to connect to a variety of Ethereum nodes and test blockchains.
4. MetaMask is a web-based wallet.



Some good starter wallets

Jaxx



1. Multiplatform and multicurrency wallet that runs on a variety of operating systems
2. It is often a good choice for new users as it is designed for simplicity and ease of use.
3. Jaxx is either a mobile or a desktop wallet

Some good starter wallets

MyEtherWallet (MEW)

1. Web-based wallet that runs in any browser.



Emerald Wallet

1. It's an open source desktop application
2. Can run a full node/ connect to public remote node, working in light mode.
3. It also has a companion tool to do all operations from the command line.



Ethereum

Control and Responsibility

Control and Responsibility

1. Open Blockchain & operate as Decentralize System
2. Each user can control their own private keys
3. That mean: control access to funds and smart contracts.
4. We sometimes call the combination of access to funds and smart contracts an "account" or "wallet."

Tips to manage this responsibility

1. Do not improvise security. Use tried-and-tested standard approaches.
2. The more important the account the higher security measures should be.
3. The highest security is gained from an air-gapped device, but this level is not required for every account.
4. Never store your private key in plain form, especially digitally.
5. Private keys can be stored in an encrypted form, as a digital "keystore" file. Being encrypted, they need a password to unlock. When you are prompted to choose a password, make it strong, back it up, and don't share it.

Tips to manage this responsibility

6. If you don't have a password manager, write it down and store it in a safe and secret place. To access your account, you need both the keystore file and the password.
7. Do not store any passwords in digital documents, digital photos, screenshots, online drives, encrypted PDFs, etc. Again, do not improvise security. Use a password manager or pen and paper.
8. When you are prompted to back up a key as a mnemonic word sequence, use pen and paper to make a physical backup. Do not leave that task "for later"; you will forget.

Tips to manage this responsibility

9. However, they can also be used by attackers to get your private keys, so never store them digitally, and keep the physical copy stored securely in a locked drawer or safe.
10. Before transferring any large amounts (especially to new addresses), first do a small test transaction (e.g., less than \$1 value) and wait for confirmation of receipt.

Ethereum Clients

Clients Introduction

Ethereum Client

- An Ethereum client is a software application that implements the Ethereum specification.
- Ethereum is defined by a formal specification called the "Yellow Paper"
- Different Ethereum clients *interoperate* if they comply with the reference specification and the standardized communications protocols.

Ethereum Improvement Proposals - EIP

This yellow paper, in addition to various Ethereum Improvement Proposals - EIP, defines the standard behavior of an Ethereum client.

The Yellow Paper is periodically updated as major changes are made to Ethereum.

Benefit of Specification vs Implementation

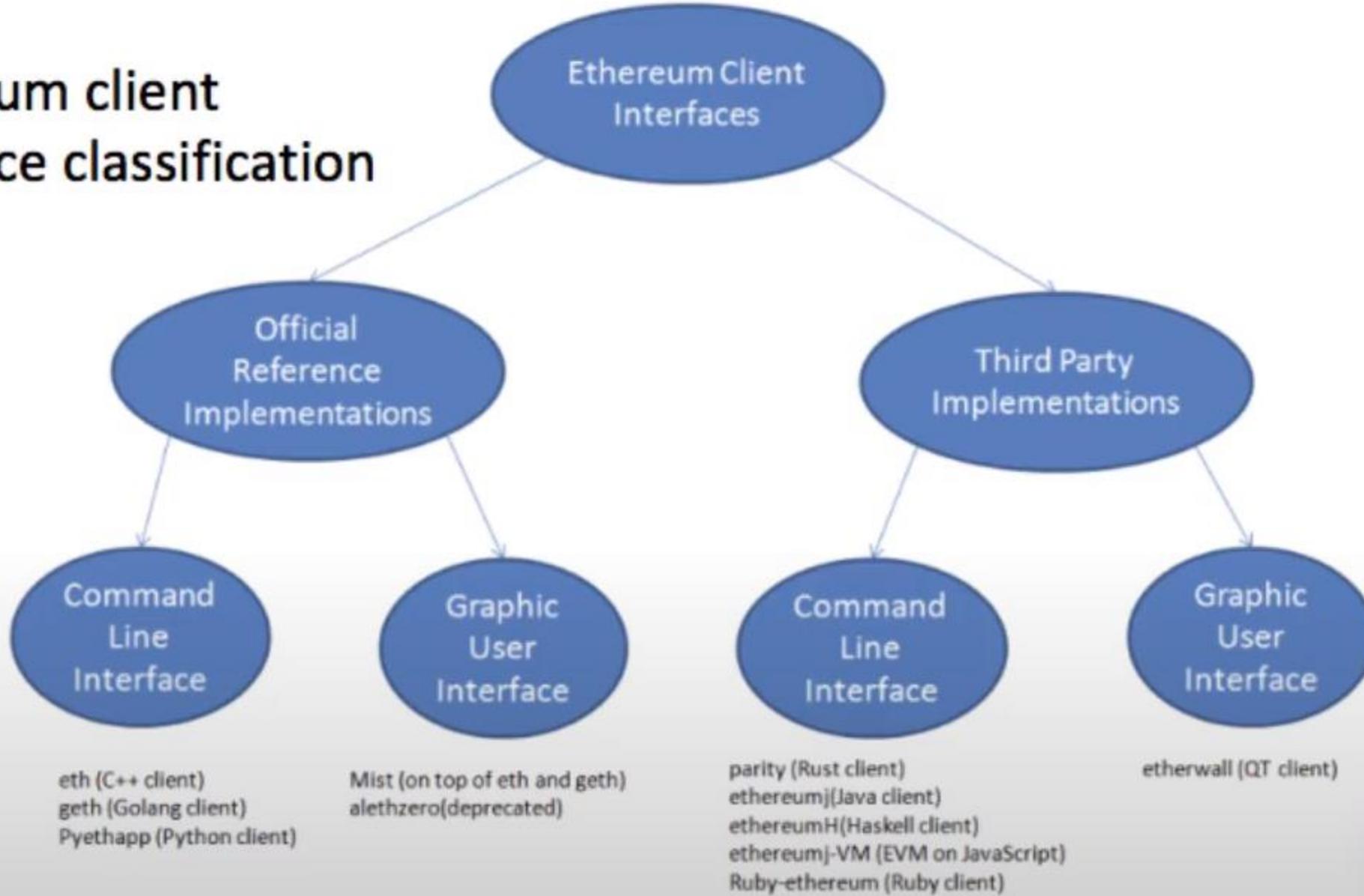
Built on specification proven itself to be an excellent way of defending against attacks on the network, because exploitation of a particular client's implementation strategy simply hassles the developers while they patch the exploit, while other clients keep the network running almost unaffected.

Ethereum Network

Currently, there are six main implementations of the Ethereum protocol, written in six different languages:

- **Parity**, written in **Rust**
- **Geth**, written in **Go**
- **cpp-ethereum**, written in **C++**
- **pyethereum**, written in **Python**
- **Mantis**, written in **Scala**
- **Harmony**, written in **Java**

Ethereum client interface classification



Should I run full Node?

For Ethereum development a full node running on a live *mainnet* network is not necessary.

You can do almost everything you need to do with

- A *testnet* node,
- Local private blockchain like **Ganache**,
- **Cloud-based Ethereum client** offered by a service provider like Infura.

Ethereum Clients

Remote Clients
Test Networks

Remote Client

Remote Client does not store a local copy of the blockchain or validate blocks and transactions.

They are wallet and can create and broadcast transactions.

They can be used to connect to existing networks, such as;

- Own Full Node,
- Public Blockchain (Mainnet)
- Testnet or
- Private (Local) blockchain

ETH Remote client vs BTC Light node

Do not confuse the concept of a remote wallet in Ethereum with that of a *light client* (which is analogous to a Simplified Payment Verification client in Bitcoin)

Light clients validate block headers and use Merkle proofs to validate the inclusion of transactions in the blockchain and determine their effects, giving them a similar level of security to a full node.

Ethereum remote clients do not validate block headers or transactions. They entirely trust a full client to give them access to the blockchain, and hence lose significant security and anonymity guarantees.

Ethereum Full Node

Advantage

- Supports the resilience and censorship resistance of Ethereum-based networks
- Authoritatively validates all transactions
- Can interact with any contract on the public blockchain without an intermediary
- Can directly deploy contracts into the public blockchain without an intermediary
- Can query (read-only) the blockchain status (accounts, contracts, etc.) offline
- Can query the blockchain without letting a third party know the information you're reading

Disadvantage

- Requires significant and growing hardware and bandwidth resources
- May require several days to fully sync when first started
- Must be maintained, upgraded, and kept online to remain synced

Running Ethereum Client

Minimum Requirements:

- CPU with 2+ cores
- At least 80 GB free storage space
- 4 GB RAM minimum with an SSD, 8 GB+ if you have an HDD
- 8 MBit/sec download internet service

Recommended Requirements:

- Fast CPU with 4+ cores
- 16 GB+ RAM
- Fast SSD with at least 500 GB free space
- 25+ MBit/sec download internet service

The disk size requirements listed here assume you will be running a node with default settings, where the blockchain is "**pruned**" of old state data. If you instead run a full "**archival**" node, where all state is kept on disk, it will likely require more than 1 TB of disk space.

Software Requirements

Goto <https://geth.ethereum.org/downloads/> and download geth

[Go Ethereum](#) [Install](#) [Downloads](#) [Documentation](#)

Download Geth – Punisher (v1.8.27) – [Release Notes](#)

You can download the latest 64-bit stable release of Geth for our primary platforms below. Packages for all supported platforms, as well as develop builds, can be found further down the page. If you're looking to install Geth and/or associated tools via your favorite package manager, please check our [installation](#) guide.

 [Geth 1.8.27 for Linux](#)

 [Geth 1.8.27 for macOS](#)

 [Geth 1.8.27 for Windows](#)

 [Geth 1.8.27 sources](#)

After installation run command

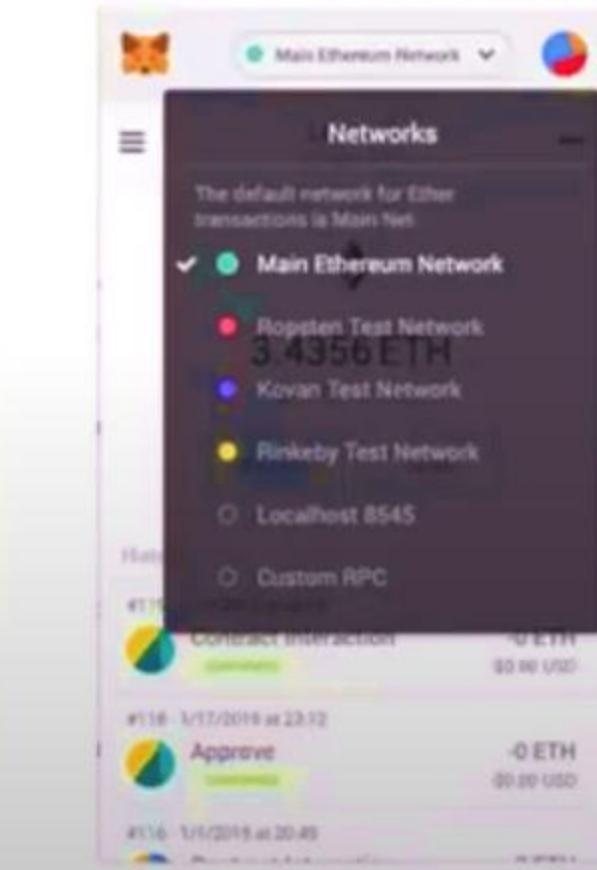
`geth --testnet --networkid 3 --datadir <testnetdatapath>`

Testnets

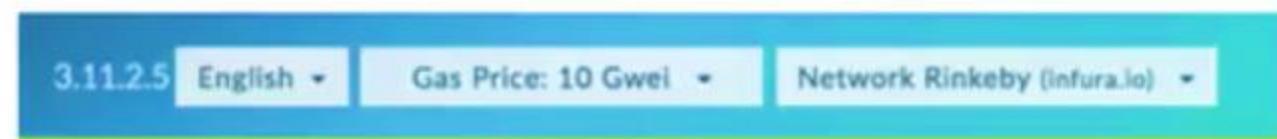
- Ropsten: A **proof-of-work** blockchain that most closely resembles Ethereum; you can easily mine faux-Ether.
- Kovan: A **proof-of-authority** blockchain, started by the Parity team. Ether can't be mined; it has to be requested.
- Rinkeby: A **proof-of-authority** blockchain, started by the Geth team. Ether can't be mined; it has to be requested.

Connecting to a testnet

MetaMask Network Selection



MyEtherWallet



MyCrypto wallet



Ethereum Clients

Test Network - Faucet

→ C https://ropsten.etherscan.io/tx/0x6603101fec4694cea473c6b1520dd32aedb7ca74f5dd4c... 🔍 ⚡

BLOCK: 3000300 2 BLOCK Confirmations

⑦ Timestamp: 1 min ago (Jun-25-2019 04:54:39 AM +UTC)

⑦ From: 0x687422eea2cb73b5d3e242ba5456b782919afc85

⑦ To: 0x340de497c3c073e4d01246dec7e94c75c8e2b199

⑦ Value: 1 Ether (\$0.00)

⑦ Transaction Fee: 0.000021 Ether (\$0.000000)

Click to see More ↓

Powered by Ethereum

Ropsten Test Network

Account 1 0x340d...b199

2.7082 ETH

Deposit Send

History

#75 - 6/23/2019 at 16:57 Withdraw -0 ETH CONFIRMED

#74 - 6/23/2019 at 16:52 Withdraw -0 ETH FAILED

#73 - 6/23/2019 at 16:46

Ethereum Wallets

Addresses - Keys - Wallets

Addresses

One of Ethereum's foundational technologies is *cryptography*

Two types of Accounts

1. Externally owned Accounts (EoA)
2. Contract Account

Externally owned account



Contract account

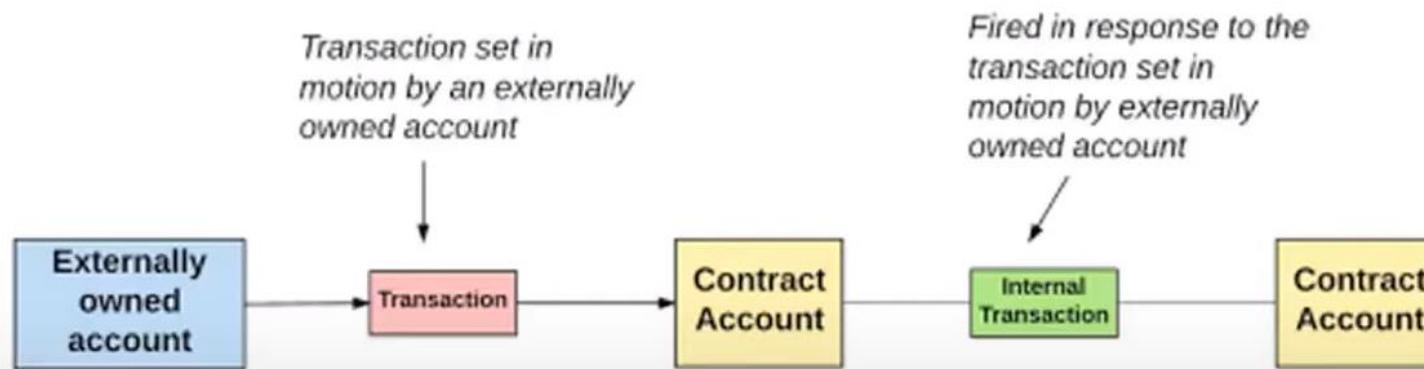
```
<code>  
<code>  
<code>
```

Ownership of ether by EOAs is established through

1. Digital private keys,
2. Ethereum addresses, and
3. Digital signatures

Addresses

- An externally owned account can send messages to other externally owned accounts OR to other contract accounts by creating and signing a transaction using its private key.



- Unlike externally owned accounts, contract accounts can't initiate new transactions on their own

Keys

Private keys are not used directly in the Ethereum system.

- ◆ Account addresses are derived directly from private keys.
- ◆ Private key uniquely determines a single Ethereum address, also known as an account.

Access and control of funds is achieved with digital signature

- ◆ Ethereum transactions require a valid digital signature to be included in the blockchain.
- ◆ Anyone with a copy of a private key has control of the corresponding account and any ether it holds.
- ◆ Contracts account are not backed by public-private key pairs

Public Key Cryptography and Cryptocurrency

In Ethereum, we use public key cryptography (also known as asymmetric cryptography) to create the public–private key pair.

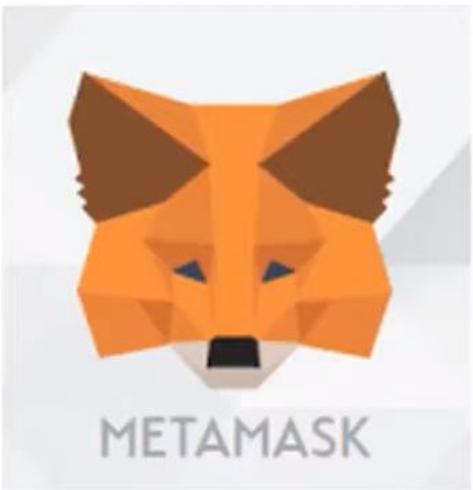
Together, they represent an Ethereum account by providing, respectively, a publicly accessible account handle (the address) and private control over access to any ether in the account and over any authentication the account needs when using smart contracts.

The private key controls access by being the unique piece of information needed to create *digital signatures*, which are required to sign transactions to spend any funds in the account. Digital signatures are also used to authenticate owners or users of contracts

Ethereum transaction is basically a request to access a particular account with a particular Ethereum address

Wallets

- A wallet is a software application that serves as the primary user interface to Ethereum.
- The wallet controls access to a user's money, managing keys and addresses, tracking the balance, and creating and signing transactions.
- Ethereum wallets can also interact with contracts, such as ERC20 tokens.
- Interfaces to Ethereum-based decentralized applications, or *DApps*



ERC20



Wallet technology overview

Ethereum wallet is a keychain containing pairs of private and public keys. Ethereum wallet doesn't contain ether or tokens. From these keys they control the ether or tokens.

Users sign transactions with the private keys, thereby proving they own the ether. The ether is stored on the blockchain.

Wallet Type

Non-deterministic wallet (JBOK wallet)

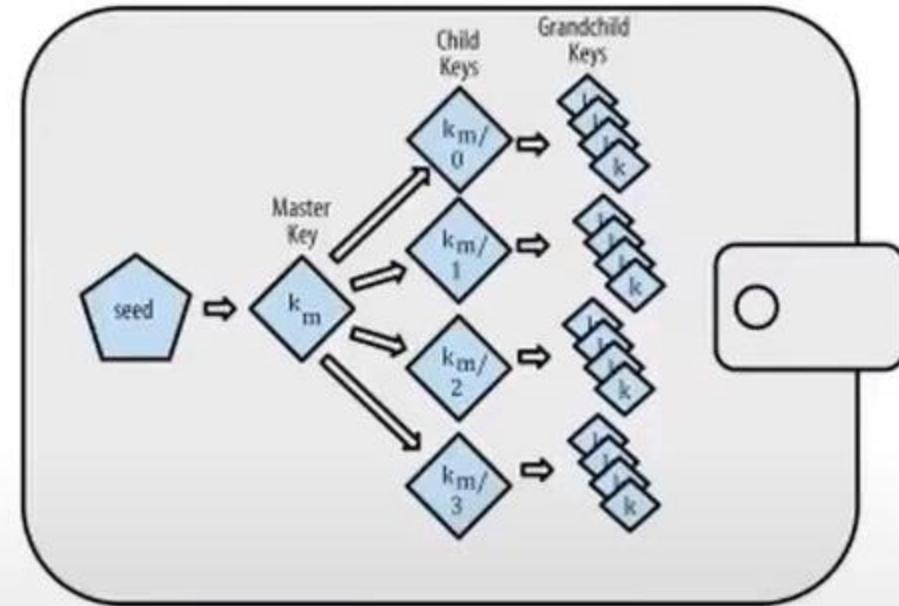
- Each key is independently generated from a different random number
- The keys are not related to each other
- Also known as JBOK (Just a Bunch of Keys) wallet

Deterministic wallet (HD Wallet)

- All the keys are derived from a single master key. Known as **Seed**.
- Keys are related to each other.
- Can be generated again from original seed.
- The most common method is tree-like structure. Also known as HD (Hierarchical Deterministic) Wallet

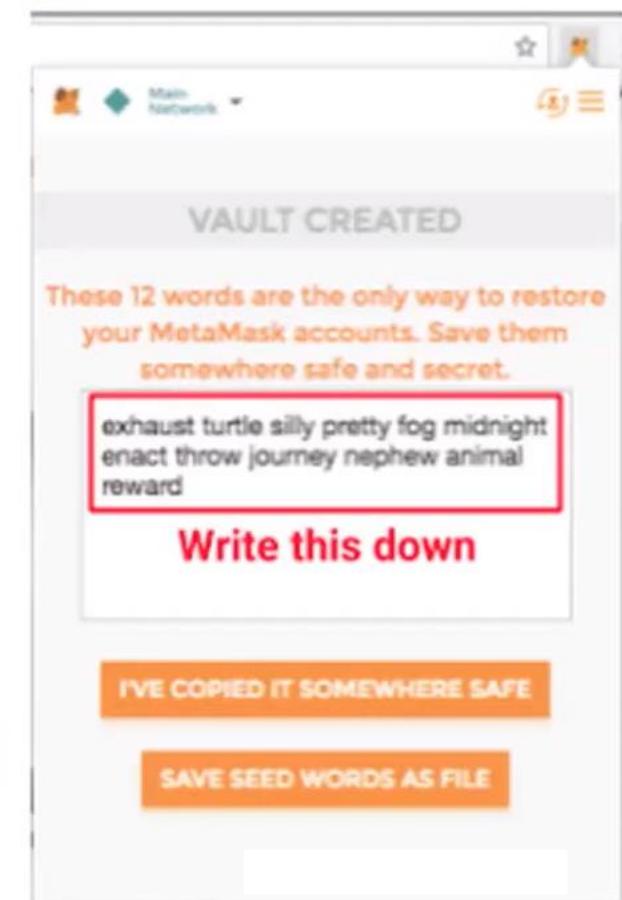
Hierarchical Deterministic (HD) Wallets

- The most advanced form of deterministic wallet is the *hierarchical deterministic* (HD) wallet defined by Bitcoin's [BIP-32 standard](#).
- HD wallets contain keys derived in a tree structure, such that a parent key can derive a sequence of child keys, each of which can derive a sequence of grandchild keys, and so on.
- HD wallets offer a few key advantages over simpler deterministic wallets.
 - The tree structure can be used to express additional organizational meaning.
 - Users can create a sequence of public keys without having access to the corresponding private keys. *This allow your wallet to execute receive-only mode.*



Seeds and Mnemonic Codes (BIP-39)

- Using a sequence of words that, when taken together in the correct order, can uniquely recreate the private key. Known as mnemonic or seed phrase.
- BIP-39 standardized this approach.
- The seed is needed to recover a wallet in the case of data loss whether through accident or theft.
- The seed must be kept extremely private, so digital backups should be carefully avoided; hence the earlier advice to back up with pen and paper.



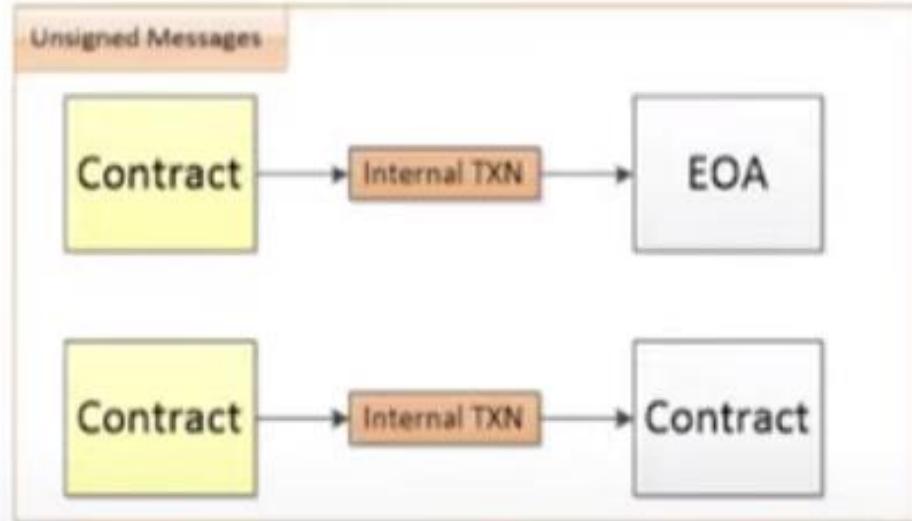
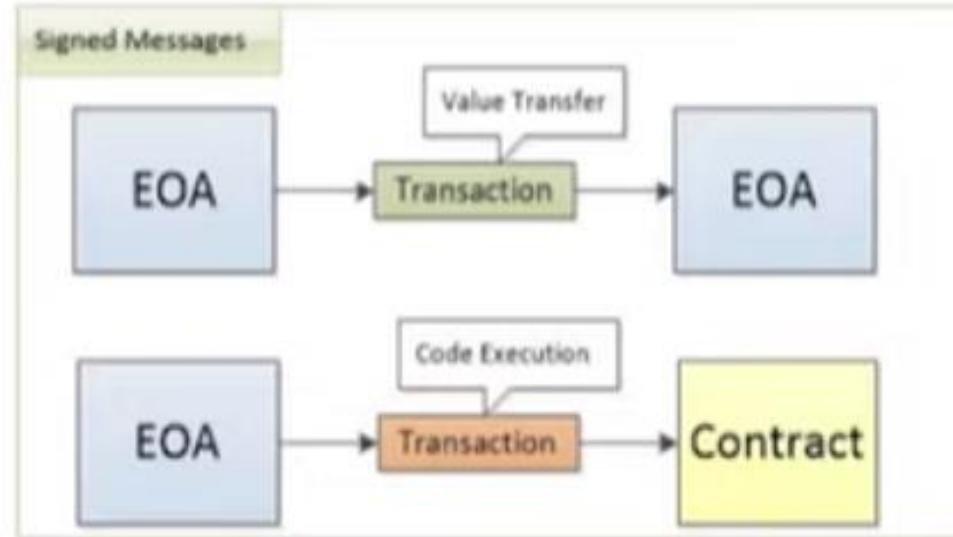
Ethereum

Transactions

Transactions

- Transactions are **signed messages originated by an externally owned account (EOA), transmitted by the Ethereum network, and recorded on the Ethereum blockchain.**
- Transactions are the only things that can **trigger a change of state, or cause a contract to execute** in the EVM.
- Ethereum is a global singleton state machine, and transactions are what make that state machine "tick," changing its state.
- **Contracts don't run on their own.** Ethereum doesn't run autonomously. Everything starts with a transaction.

Type of Transactions



Transaction Structure

Nonce	A sequence number, issued by the originating EOA, used to prevent message replay. This number represents the number of transactions sent from the account's address
Gas Price	The price of gas (in wei) the originator is willing to pay
Gas Limit	The maximum amount of gas the originator is willing to buy for this transaction
Recipient	The destination Ethereum address
Value	The amount of ether to send to the destination
Data	The variable-length binary data payload
v,r,s	The three components of an ECDSA digital signature of the originating EOA

Transaction Structure (Cont...)

- The transaction message structure is serialized using the **Recursive Length Prefix (RLP) encoding** scheme.
- All numbers in Ethereum are **encoded as big-endian integers**, of lengths that are **multiples of 8 bits**.
- The data are **separated by fixed length**. No label is included for any field in transaction serialization.
- **There is no “from” data** in the address identifying the originator EOA. That is because the **EOA’s public key can be derived from the v,r,s components** of the ECDSA signature. The address can be **derived from the public key**.

The Transaction Nonce

nonce: A scalar value equal to the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.

For EOA

This number represents the number of confirm transactions sent from the account's address

For Contract Address

The nonce is the number of contracts created by the account.

It is important to note that the use of the nonce is actually vital for an *account-based* protocol, in contrast to the "Unspent Transaction Output" (UTXO) mechanism of the Bitcoin protocol.

Transaction Nonce (Cont..)

Why does it matter?

This value prevents **double-spending**, as the nonce will always specify the order of transactions. If a double-spend does occur, it's typically due to the following process:

- A transaction is sent to one party.
- They wait for it to register.
- Something is collected from this first transaction.
- Another transaction is quickly sent with a high gas price.
- The second transaction is mined first, therefore invalidating the first transaction.

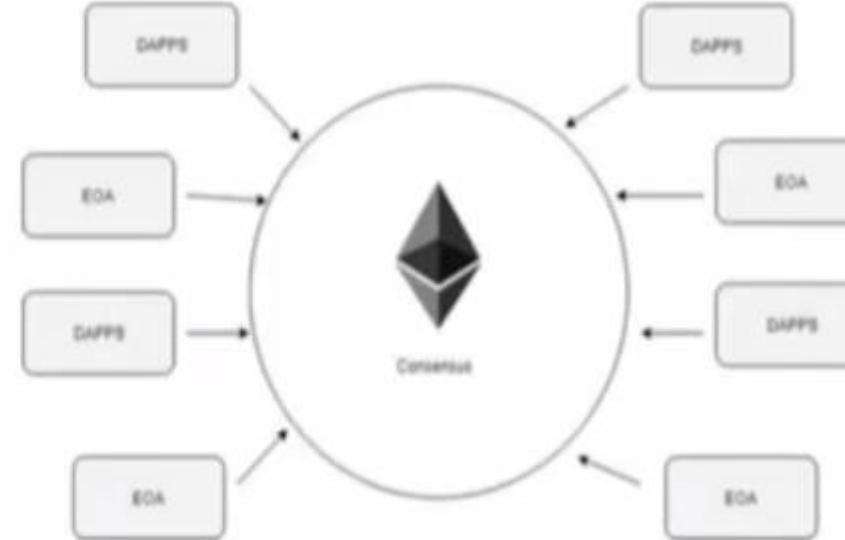
In Ethereum, this method of “double-spending” is not possible because each transaction has a nonce included with it. Even if you attempt to do the above, it will not work as the second transaction (nonce of 3) cannot be mined before the first transaction (nonce of 2).

Ethereum

Concurrency

Concurrency

Ethereum, by definition, is a system that **allows concurrency of operations** (nodes, clients, DApps) but **enforces a singleton state through consensus**.



Concurrency Problem

Now, imagine that you have multiple independent wallet applications that are generating transactions from the same address or addresses.

Example Scenario:

An exchange processing withdrawals from the exchange's hot wallet (a wallet whose keys are stored online, in contrast to a cold wallet where the keys are never online).

There are more than one computer processing withdrawals, so that it doesn't become a bottleneck or single point of failure. However, this quickly becomes problematic, as having more than one computer producing withdrawals will result in some thorny concurrency problems, not least of which is the selection of nonces.

How do multiple computers generating, signing, and broadcasting transactions from the same hot wallet account coordinate?

Potential Solutions:

1. Separate Computer to assign nonce:

You could use a **single computer to assign nonces**, on a first-come first-served basis, **to computers signing transactions**.

Pro:

Avoided gap and duplication of nonce

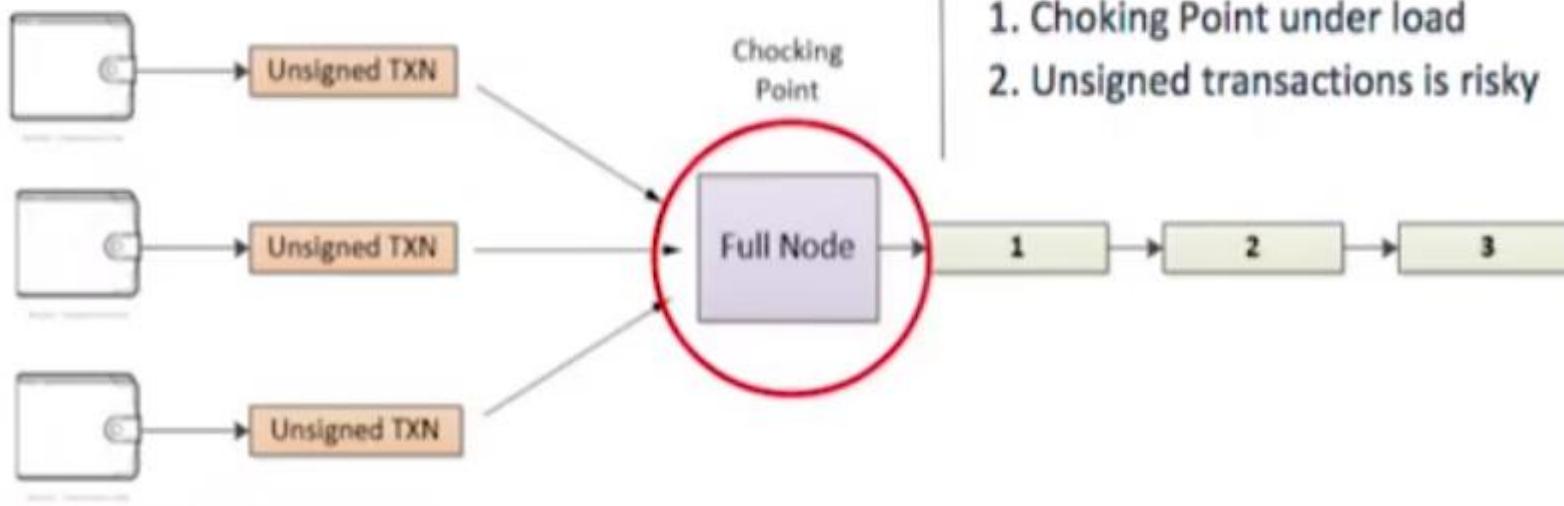
Con:

1. Single Point of failure
2. Transaction can be stuck on non-confirmation

Potential Solutions: (cont..)

2. Full Node will assign the nonce

Send unsigned transaction and queue on **single node to sign, assign nonces and process transactions**



Pro:

Avoided gap and duplication of nonce

Con:

1. Choking Point under load
2. Unsigned transactions is risky

Avoid Concurrent Transactions



These concurrency problems, on top of the difficulty of tracking account balances and transaction confirmations in independent processes, force most implementations toward avoiding concurrency and creating bottlenecks such as a single process handling all withdrawal transactions in an exchange, or setting up multiple hot wallets that can work completely independently for withdrawals and only need to be intermittently rebalanced.

Ethereum

Transaction Gas

Transaction GAS



- Gas is the **fuel of Ethereum**.
- Gas is not ether—it's a **separate virtual currency**
- Gas has **its own exchange rate** against ether.
- Ethereum uses gas to **control resources usage** of transaction.
- The open-ended (Turing-complete) computation model requires some form of **metering** in order to avoid **denial-of-service attacks** or **inadvertently resource-devouring transactions**.

Gas Price

- Wallets can adjust the gasPrice in transactions they originate to achieve faster confirmation of transactions
- The higher the gasPrice, the faster the transaction is likely to be confirmed
- The minimum value that gasPrice can be set to is zero
- GasLimit gives the maximum number of units of gas the transaction originator is willing to buy in order to complete the transaction.



The highest paid Gas Price

ETH 2100

Equivalent of

US\$ 628,446.40

Gas Limit

- GasLimit is the maximum gas the originator is willing to pay for the transaction.
- For Contract, the amount of gas cannot be determined with accuracy. Since contract execution is not predictable.
- This value insures that in case of an issue executing your transaction (like infinite loop), you account is not drained of all the funds.
- You are only billed for gas actually consumed by your transaction. Remaining gas is sent back to your account

Ethereum

Transaction Value & Data

Transaction Value and Data

- The main "payload" of a transaction is contained in two fields: value and data.
- Transactions can have 4 valid combinations;
 - a. only value - **Payment**
 - b. only data - **Invocation**
 - c. both value and data - **Payment and Invocation**
 - d. neither value nor data.- **Empty Transaction**
-

Transaction Recipient

The recipient of a transaction is specified in the `to` field. This contains a 20-byte Ethereum address. The address can be an EOA or a contract address.

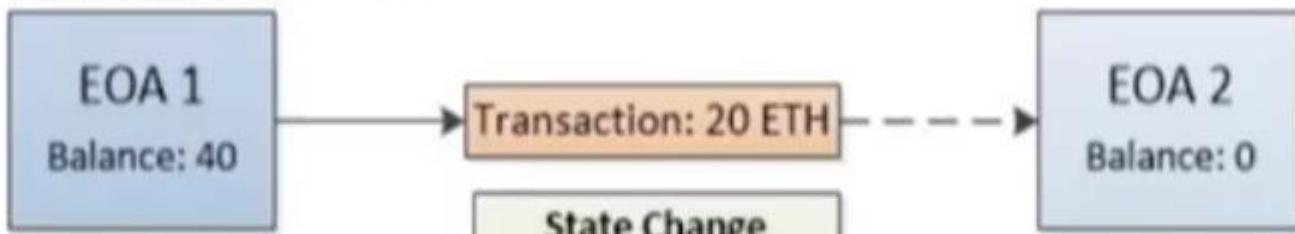
Any 20-byte value is considered valid. If the 20-byte value corresponds to an address without a corresponding private key, or without a corresponding contract, the transaction is still valid.

Sending a transaction to the wrong address will probably *burn* the ether sent.

Validation should be done at the user interface level.

Transmitting Value to EOA

Pending Transaction

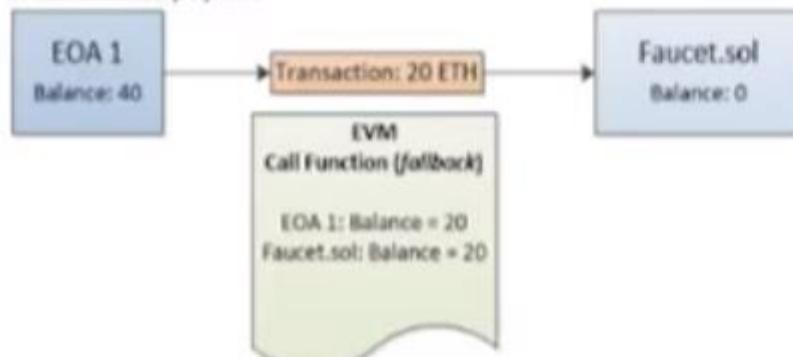


Confirm Transaction

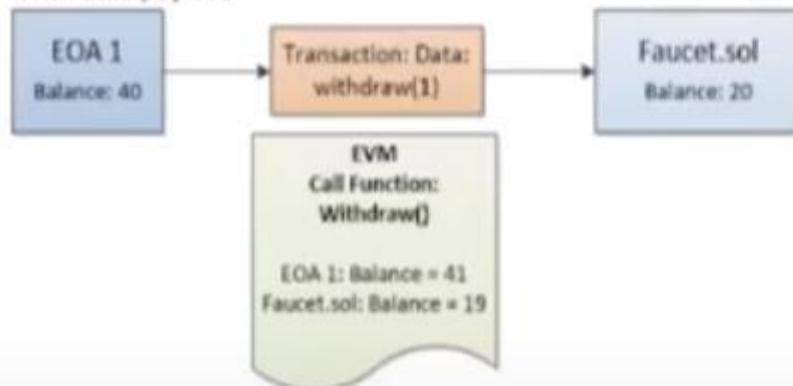


Transmitting Value to Contract

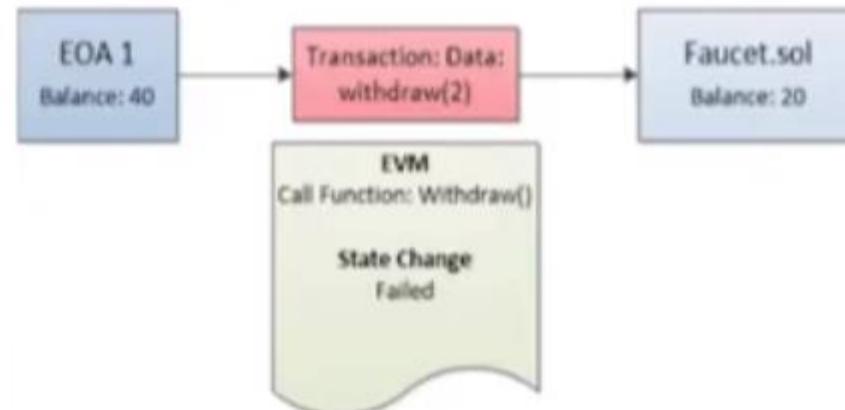
Without data payload



With data payload



Failed Transactions



Transmitting Data to EOA

You can send a data payload to an EOA—that is completely valid in the Ethereum protocol.

Most wallets also ignore any data received in a transaction to an EOA they control.

Transmitting data payload to an EOA is not subject to Ethereum's consensus rules, unlike a contract execution.

Transmitting Data to Contract

The data will be interpreted by the EVM as a *contract invocation*. Most contracts use this data more specifically as a *function invocation*, calling the named function and passing any encoded arguments to the function.

The data payload is a hex-serialized encoding of:

A function selector

The first 4 bytes of the Keccak-256 hash of the function's prototype. This allows the contract to unambiguously identify **which function you wish to invoke**.

The function arguments

The function's arguments, **encoded according to the rules for the various elementary types** defined in the ABI specification.

Special Transaction: Contract Creation

Contract creation transactions are sent to a special destination address called the *zero address*; **0x0**

It can never spend ether or initiate a transaction. It is only used as a destination, with the special meaning "create this contract."

Burn Address

0x00000000000000000000000000000000dEaD

Any ether sent to the designated burn address will become unspendable and be lost forever.

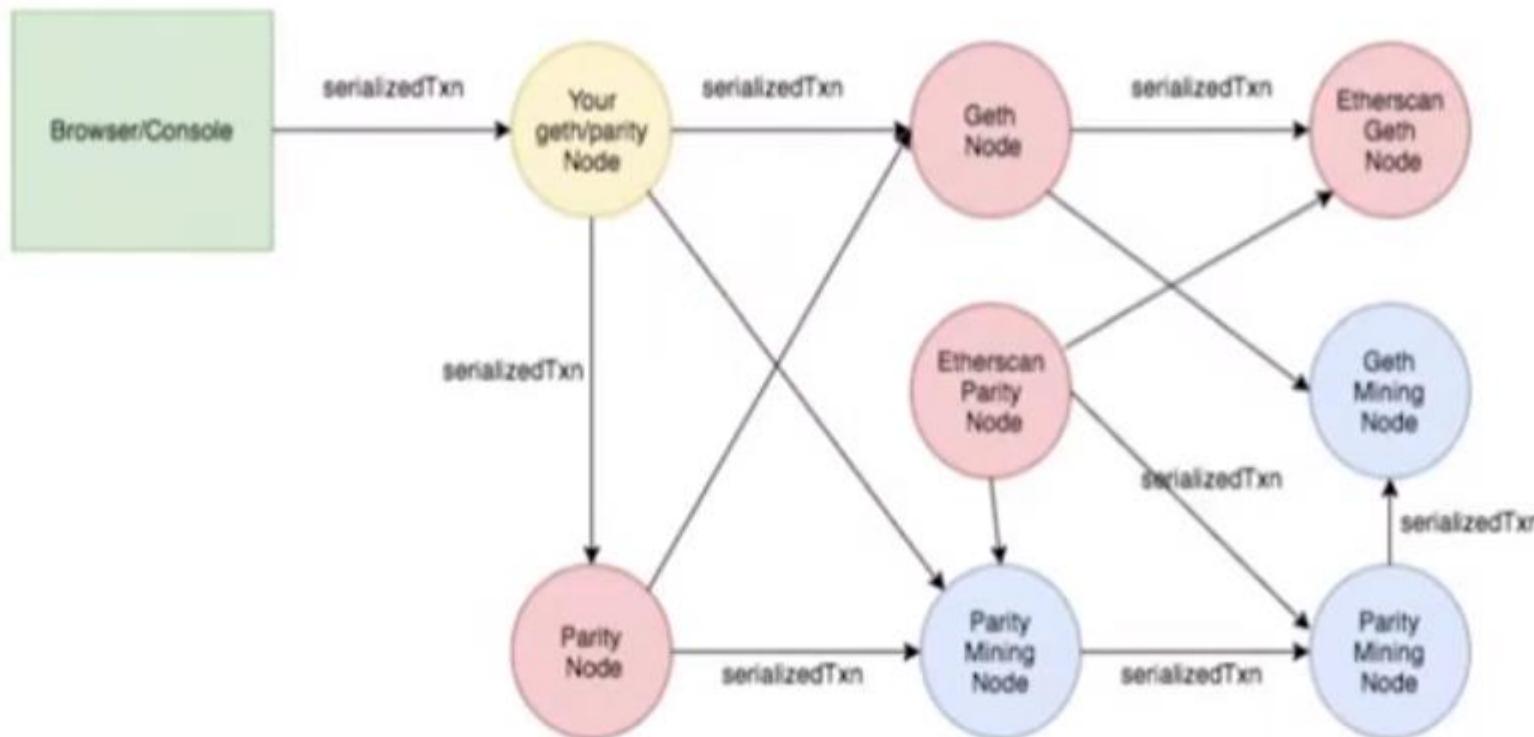
Ethereum

Transaction Lifecycle and Propagation

Transaction Lifecycle

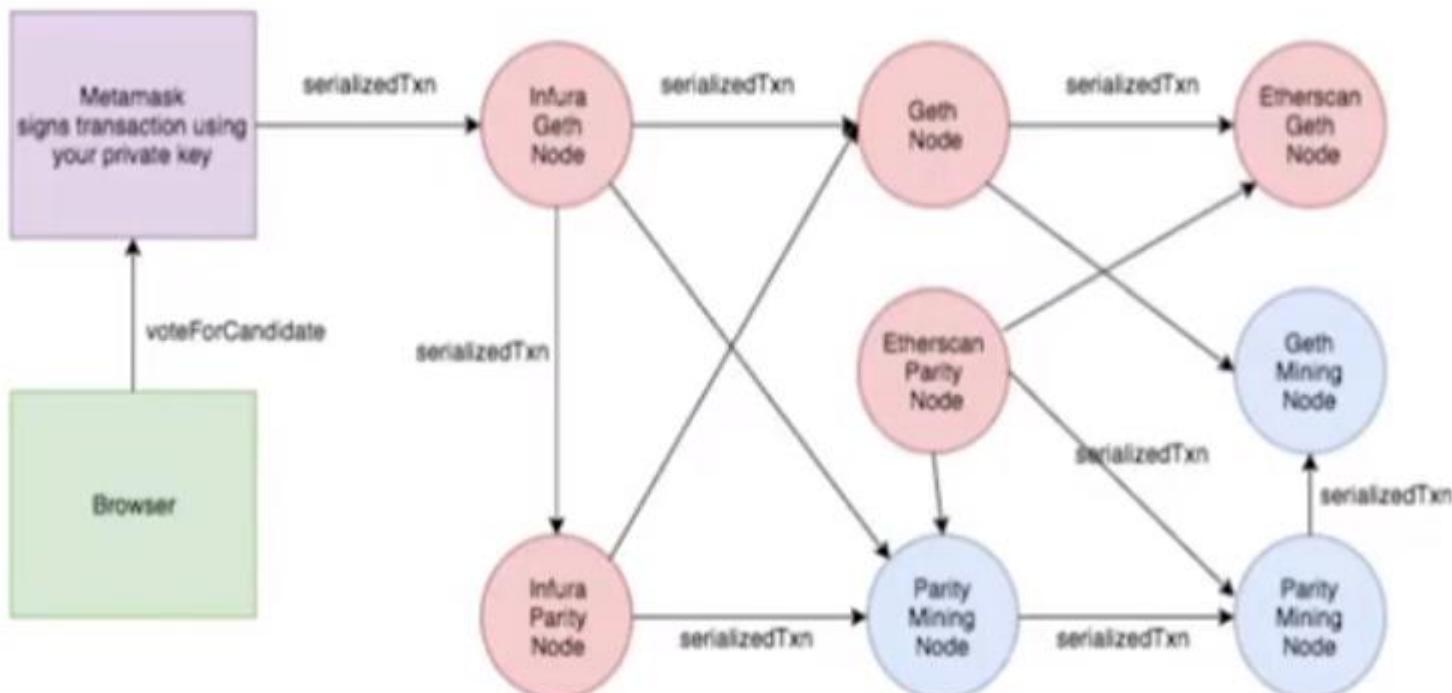
- Step 1. Construct Raw Transaction**
- Step 2. Sign the Transaction**
- Step 3. Transaction is validated locally**
- Step 4. Broadcast to network**
- Step 5. Miner Node accepts**
- Step 6. Miner Node finds a valid block and broadcast**
- Step 7. Local Node received/Sync the new block**

Transaction Lifecycle: Local Node



Signed Transaction Propogates to the network

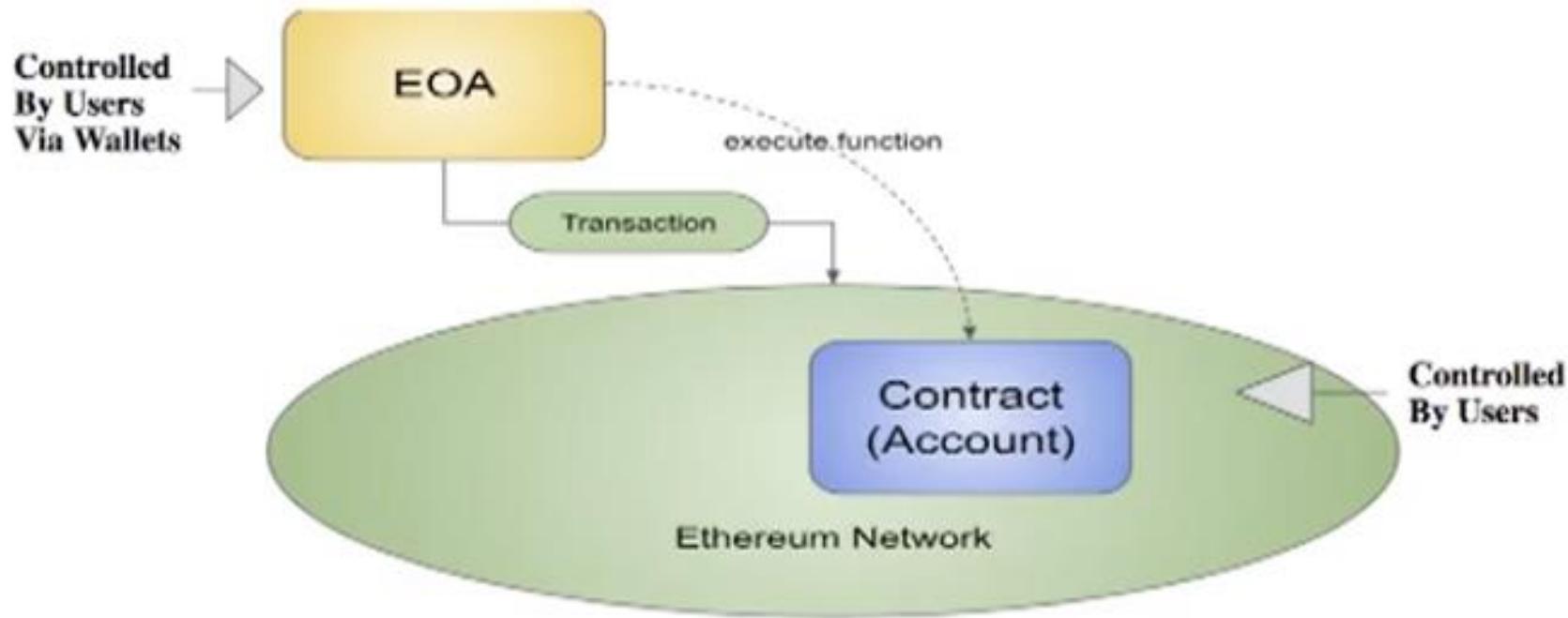
Transaction Lifecycle: From Remote Client



Ethereum

EOA and Contract Account

Smart Contracts & Solidity



Ethereum

Smart Contract

Smart Contract

- Computer Program
- Immutable
- Deterministic
- EVM Context
- Decentralized World Computer



In the 1990s, cryptographer Nick Szabo coined the term and defined it as "a set of promises, specified in digital form, including protocols within which the parties perform on the other promises."

Ethereum

Life Cycle of a Smart Contract

Life Cycle of a Smart Contract

- Written in a **high-level language**, such as **Solidity**
- Compiled to the low-level **bytecode** that runs in the **EVM**.
- **Deployed** using a special *contract transaction*
- **Contract Address** can be used in a transaction as the **recipient** for
Sending funds to the contract or **calling** one of the **contract's**
functions
- **Contracts only run if they are called by a transaction**
- Contracts never run “on their own” or “in the background.”
- Smart contracts are not executed “in parallel” in any sense

Ethereum

Life Cycle of a Smart Contract

Smart Contract execution

- **Can call another contract** that can call another contract, and so on,
- **First contract** in a chain of execution will always have been **called by a transaction from an EOA**
- **Recorded only if all execution terminates successfully**, other wise Roll Back
- **Failed transaction is still recorded** as having been attempted,
- **Ether spent on gas for the execution is deducted from originating account**,
- Otherwise has **no other effects on contract or account state**

Ethereum

Deleting a Smart Contract

Deleting a Smart Contract



- Contract's code cannot be changed
- By Removing the code and its Internal state (storage) from its address,
 - Leaving a blank account.
- Executing an EVM opcode called SELFDESTRUCT
- Costs “negative gas,” a gas refund (Incentive for releasing network resources)
- Does not remove the transaction history (past) of the contract. - IMMUTABILITY
- SELFDESTRUCT capability will only be available if the contract to have that functionality.

Ethereum

High-Level Languages

Introduction - Ethereum High-Level Languages

EVM is a **virtual machine** that runs a special form of code called **EVM bytecode**

- Any high-level language could be adapted to write smart contracts
- a number of special-purpose languages have emerged for programming smart contracts.

Declarative (Functional) programming languages (Haskell and SQL)

Imperative (Procedural) programming languages (C++ and Java)

Hybrids include Lisp, JavaScript, and Python

Ethereum

Tokens

What are Tokens

Commonly used to refer to **privately issued special-purpose coin**

- like items of **insignificant intrinsic value**,
- i. e. Transportation, food courts, and game tokens.

Nowadays, "tokens" administered on blockchains are **redefining the word** to mean **blockchain-based abstractions** that can be owned and that represent

- Assets,
- Currency, or
- Access rights.



Ethereum

Traditional Vs Blockchain

Tokens

Traditional Vs Blockchain Tokens

Traditional Tokens

Restricted to specific businesses, organizations, or locations.

Physical tokens are **not easily exchangeable** and **typically have only one function**.



Blockchain Tokens

these restrictions are lifted

Many blockchain tokens serve multiple purposes globally &

Can be traded for each other / for other currencies

Ethereum

How are Token Used

How are Token Used

Most obvious use of tokens is as digital private currencies

This is only one possible use, **Currency is just the first application:**

Tokens can be programmed to serve many different functions

What can be those difference Functions?



How are Token Used

Currency - Serve as a form of currency, with a value determined through private trade

Resource - *Earned or Produced* in a sharing economy or resource-sharing environment; i.e., a storage or CPU token representing resources that can be shared over a network.

Asset - Ownership of an intrinsic or extrinsic, tangible or intangible asset; i.e., gold, real estate, a car, oil, energy, MMOG items, etc.

Access - Access rights and grant access to a digital or physical property, i.e. discussion forum, an exclusive website, a hotel room, or a rental car.

Equity - Shareholder equity in a digital organization (e.g., a DAO) or legal entity (e.g., a corporation).

How are Token Used

Voting - Voting rights in a digital or legal system.

Collectible - Digital / physical collectible

Identity - Digital identity (e.g., avatar) or legal identity (e.g., national ID).

Attestation - Certification or attestation of fact by some authority or by a decentralized reputation system (e.g., marriage record, birth certificate, college degree).

Utility - Access or pay for a service.

Ethereum

Fungible and Non-fungible Tokens

Tokens & Fungibility

"In economics, fungibility is the property of a good or a commodity whose individual units are essentially interchangeable."

Tokens are fungible when we can substitute any single unit of the token for another without any difference in its value or function.



Tokens & Fungibility

Fungibility is the property of a good or a commodity whose individual units are essentially interchangeable.

Tokens are **fungible** when we can substitute any single unit of the token for another without any difference in its value or function.

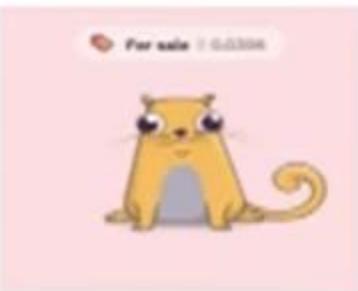
Non-fungible tokens are tokens that each represent a unique tangible or intangible item and therefore are not interchangeable.

Non-fungible

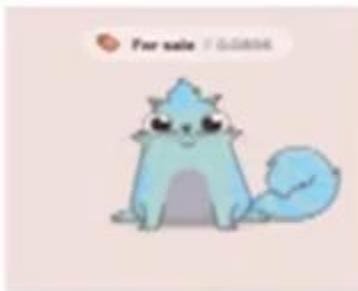


Non-fungible tokens are unique and hold information instead of value.

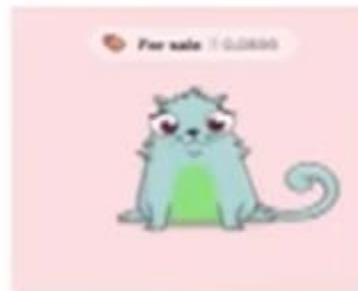
What is Crypto Kitties??



Kitty 240616 - Gen 8
Snappy



Kitty 240611 - Gen 6
Swift



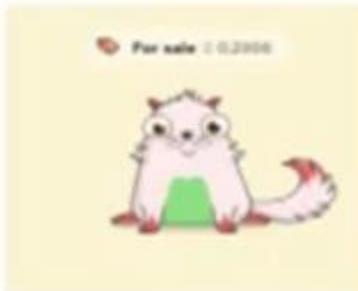
Kitty 240609 - Gen 7
Snappy



Kitty 240603 - Gen 7
Snappy



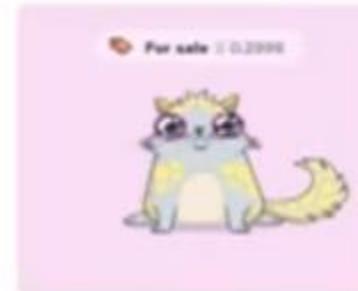
Kitty 240601 - Gen 9
Snappy



Kitty 240570 - Gen 7
Snappy



Kitty 240568 - Gen 15
Pudding



Kitty 240560 - Gen 9
Snappy

Ethereum

Fungible/Non-fungible and
Tangible/Intangible

NON-FUNGIBLE

FUNGIBLE



Intangible

Tangible

Ethereum

Counterparty Risk

Counterparty Risk

Risk that the *other* party in a transaction will fail to meet their obligations

Some transactions suffer **additional counterparty risk** if there are more than two parties involved.

when an asset is traded indirectly through the exchange of a token of ownership

- **Custodian of the asset**
- ie. such as a certificate, deed, title, or **digital token**

In the world of digital tokens representing assets, as in the non digital world,

It is important to understand **who holds the asset** that is represented by the token



Ethereum

Intrinsic and Extrinsic Value

Token & Intrinsicity

Tokens that represent **intrinsic assets** do not carry additional counterparty risk

If you hold the keys for a **CryptoKitty**, you own it directly

- No other party holding that CryptoKitty for you

Conversely, many tokens are used to represent **extrinsic** things.

- Real estate, shares, trademarks, and gold bars
- Which are not "within" the blockchain, is governed by law, custom, and policy
- Token issuers and owners may still depend on real-world non-smart contracts,
- Extrinsic assets carry additional counterparty risk as they are held by custodians,
- i.e. Recorded in external registries, or controlled by laws outside the blockchain.

Ethereum

Tokens on Ethereum

Tokens on Ethereum

Blockchain tokens existed before Ethereum.

In some ways, the first blockchain currency, Bitcoin, is a token itself.

Many token platforms were also developed on Bitcoin and other cryptocurrencies before Ethereum.

Introduction of first token standard on Ethereum led to an explosion of tokens

Ethereum

Token Standards - ERC20

Token Standards - ERC20

First standard was introduced in November 2015 by [Fabian Vogelsteller](#) as an [Ethereum Request for Comments \(ERC\)](#). It was automatically assigned [GitHub issue number 20](#), giving rise to the name "ERC20 token." The vast majority of tokens are currently based on the ERC20 standard.

ERC20 is a [standard for fungible tokens](#), meaning that [different units of an ERC20 token are interchangeable](#) and have [no unique properties](#).

The ERC20 standard defines a common interface for contracts implementing a token, such that any compatible token can be accessed and used in the same way.

Ethereum

ERC20 -- Required Functions and Events

ERC20 required functions and events

An ERC20-compliant token contract must provide at least the following functions and events:

totalSupply - Returns the total units of this token that currently exist. ERC20 tokens can have a fixed or a variable supply.

balanceOf - Given an address, returns the token balance of that address.

transfer - Given an address and amount, transfers that amount of tokens to that address, from the balance of the address that executed the transfer.

transferFrom - Given a sender, recipient, and amount, transfers tokens from one account to another. Used in combination with approve.

ERC20 required functions and events

approve - Given a recipient address and amount, authorizes that address to execute several transfers up to that amount, from the account that issued the approval.

allowance - Given an owner address and a spender address, returns the remaining amount that the spender is approved to withdraw from the owner.

Transfer - Event triggered upon a successful transfer (call to transfer or transferFrom) (even for zero-value transfers).

Approval - Event logged upon a successful call to approve.

Ethereum

ERC20 -- Optional Functions

ERC20 optional functions

name - Returns the human-readable name (e.g., "US Dollars") of the token.

symbol - Returns a human-readable symbol (e.g., "USD") for the token.

decimals - Returns the number of decimals used to divide token amounts. For example, if decimals is 2, then the token amount is divided by 100 to get its user representation.

Ethereum

Smart Contract

Misconceptions

Smart Contract

When smart people hear the term “smart contracts”, their imaginations tend to run wild

A smart contract is just a **fancy name** for code that runs on a blockchain, and interacts with that blockchain’s state.

A smart contract is a **piece of code** that is stored on an blockchain, triggered by blockchain transactions and which reads and writes data in that blockchain’s database.

Each Language has its strengths and weaknesses – you’d be crazy to build a website in C or compress HD video in Ruby.

But in principle at least, you could if you wanted to. You’d just pay a heavy price in terms of convenience, performance, and quite probably, your hair.

Ethereum

Contacting External Service

Use Case 1: Contacting External Service

Problem:

Because the source is outside of the blockchain, there is no guarantee that every node will receive the same answer. Since the smart contracts are executed independently by every node.

Perhaps the source will change its response in the time between requests from different nodes, or perhaps it will become temporarily unavailable. Either way, consensus is broken and the entire blockchain dies.

Use Case 1: Contacting External Service

Workaround:

Oracle pushes the data onto the blockchain rather than a smart contract pulling it in.

Instead of a smart contract initiating the retrieval of external data, one or more trusted parties (“oracles”) creates a transaction which embeds that data in the chain. Every node will have an identical copy of this data, so it can be safely used in a smart contract state computation.

Use Case 2: Enforcing on-chain payments

Many like the idea of a smart contract which calls a bank's API in order to transfer money.

But if every node is independently executing the code in the chain to take out money from account, The bank account got empty instantly and who is responsible for calling this API?

Solution:

A bank could proactively watch a blockchain and perform money transfers which mirror the on-chain transactions. This presents no risk to the blockchain's consensus because the chain plays an entirely passive role.

Ethereum

Enforcing on-chain payments

2. Enforcing on-chain payments

Smart Bond:

The idea is for the **smart contract code to automatically initiate the payments** at the appropriate times, avoiding manual processes and guaranteeing that **the issuer cannot default**.

Of course, in order for this to work, the funds used to make the payments **must live inside the blockchain as well, otherwise a smart contract could not possibly guarantee their payment.**

The above makes a smart bond either **pointless for the issuer, or pointless for the investor.**

2. Enforcing on-chain payments

From an **investor's perspective**, the whole point of a bond is **its attractive rate of return, at the cost of some risk of default**.

And for the issuer, a bond's purpose is to **raise funds for a productive but somewhat risky activity**, such as building a new factory.

There is **no way** for the **bond issuer to make use of the funds raised**, while simultaneously guaranteeing that the investor will be repaid.

It should not come as a surprise that the **connection between risk and return is not a problem that blockchains can solve**.

Ethereum

Hiding Confidential Data

3. Hiding Confidential Data

The biggest challenge in deploying blockchains is the radical transparency which they provide.

10 banks set up a blockchain together, and two conduct a bilateral transaction, this will be immediately visible to the other eight.

Wrong Assumption:

Every Smart contract is owner of its data and it has full control over it. It's impossible for other Smart Contract to read data directly.