Ethereum - #07

Blockchain and Distributed Ledger technologies

SAPIENZA
UNIVERSITÀ DI ROMA

Massimo La Morgia
massimo.lamorgia@uniroma1.it

Ethereum

Ethereum is a blockchain founded by Vitalik Buterin and Gavin Wood.

It was conceived in 2013, while the inception block was mined on July 20, 2015.

At that time Vitalik was 17 years old

In 2017 Gavin Wood created Polkadot another blockchain.

Ethereum

The idea behind Ethereum is to build a blockchain with a computer embedded in it. It is the foundation for building apps and organizations in a decentralized, permissionless, censorship-resistant way.

Everyone who participates in the Ethereum network (every Ethereum node) keeps a copy of the state of this computer.

Everyone can broadcast a request to this computer to perform arbitrary computation.

This execution cause a state change in the **EVM** (Ethereum Virtual Machine), which is committed and propagated to the entire network.

Thus, Ethereum we have to think to Ethereum as a distributed stat machine rather than a distributed ledger.

Ether

The native coin of Ethereum is called **Ether** (ETH).

Ether can be subdivided into smaller units, down to the smallest unit possible, which is named **wei.**

A wei is $1{*}10^{-18}$ Ether.

Any participant who broadcasts a transaction request must also offer some amount of ETH to the network as a bounty (like in Bitcoin).

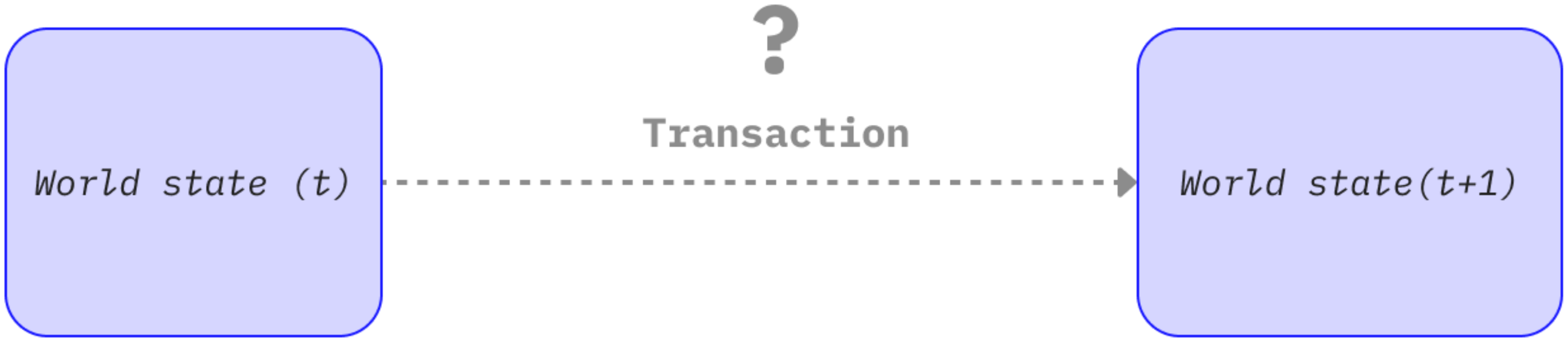
The amount of ETH paid corresponds to the resources required to do the computation (Differently from Bitcoin where the fee paid the amount of space used into a block).

Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM) is a decentralized virtual environment that executes code consistently and securely across all Ethereum nodes.

Ethereum's state is a large data structure which holds not only all accounts and balances, but a machine state, which can change from block to block according to a pre-defined set of rules, and which can execute arbitrary machine code. The specific rules of changing state from block to block are defined by the EVM.

The state is an enormous data structure called a **modified Merkle Patricia Trie**, which keeps all accounts linked by hashes and reducible to a single root hash stored on the blockchain.



Smart contract

They are computer programs stored on the blockchain that follow "if this then that" logic, and are guaranteed to execute according to the rules defined by its code, which cannot be changed once created.

Smart contracts can not initiate transactions.

Smart contract deterministically execute unambiguous code when certain conditions are met. There is no need to wait for a human to interpret or negotiate the result.

Smart contracts are useful for audits and tracking. Since Ethereum smart contracts are on a public blockchain, anyone can instantly track asset transfers and other related information. For example, you can check to see that someone sent money to your address.

They can perform computations, create currency, store data, mint NFTs, send communications.

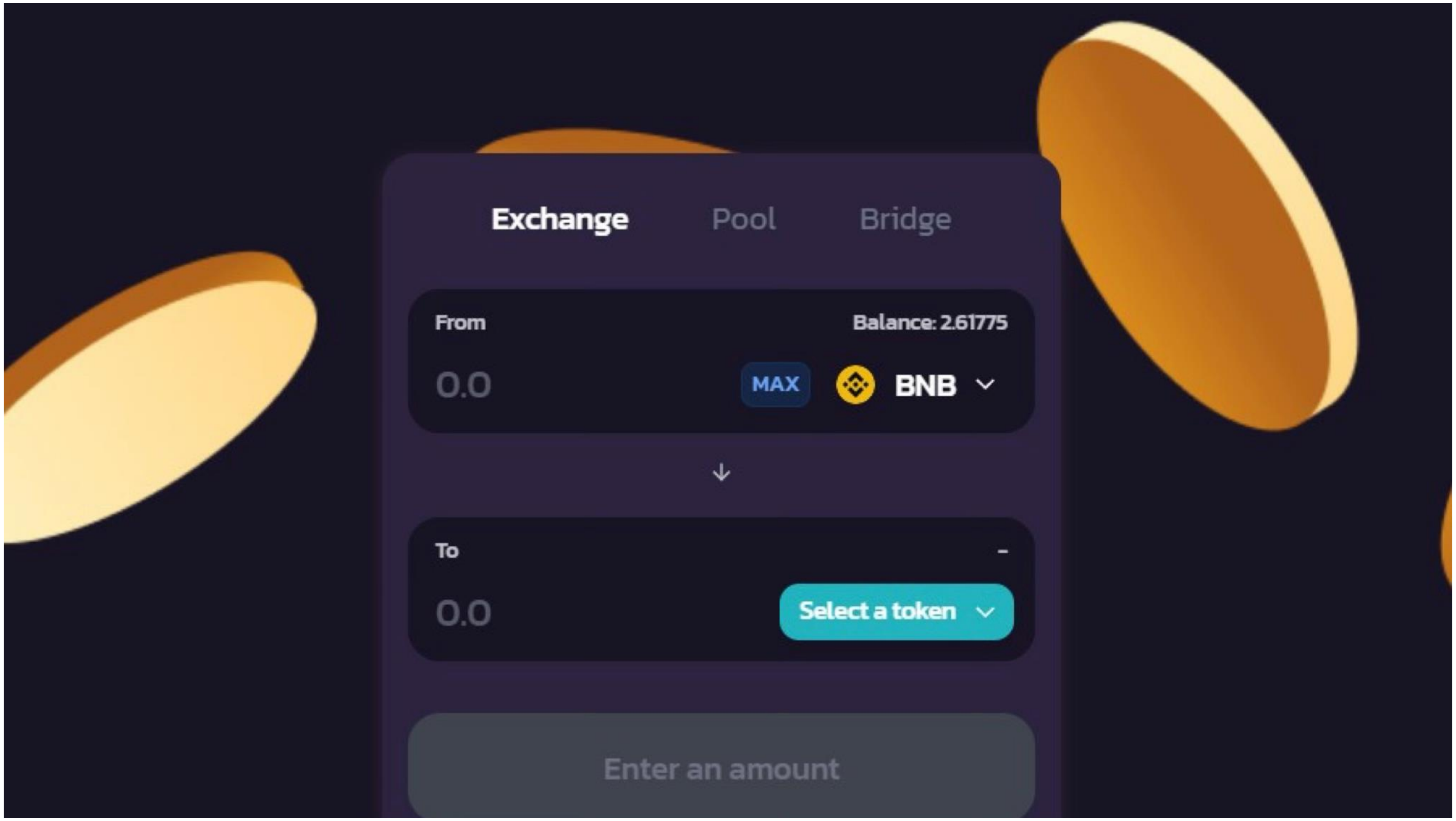
Smart contract are written in **Solidity** or **Vyper**.

Stable coin

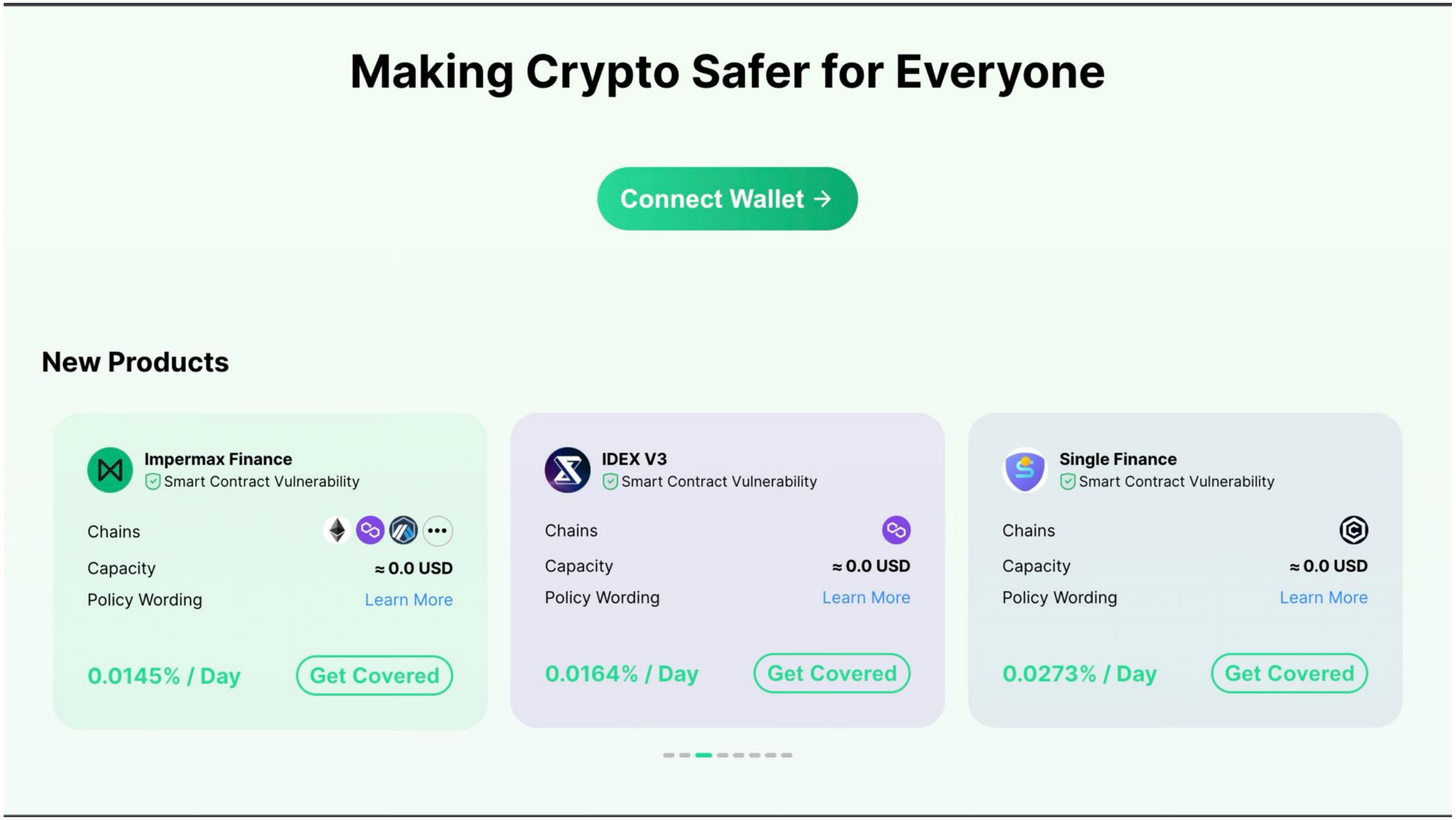
Stablecoins are cryptocurrencies without the volatility. Their value is steady, more like a traditional currency. Some time they are pegged to the value of a FIAT currency.

NFT - Not Fungible token

DeX


Exchange
Pool
Bridge
From
Balance: 2.61775
0.0
MAX
BNB
To
-
0.0
Select a token
Enter an amount

insurance



Making Crypto Safer for Everyone

New Products

	Impermax Finance	IDEX V3	Single Finance
	Smart Contract Vulnerability	Smart Contract Vulnerability	Smart Contract Vulnerability
Capacity	≈ 0.0 USD	≈ 0.0 USD	≈ 0.0 USD
Policy Wording	Learn More	Learn More	Learn More
	0.0145% / Day	0.0164% / Day	0.0273% / Day

Ethereum Account

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

Ethereum has two types of accounts:

Externally Owned Account (EOA)

Controlled by anyone with the private keys-

Public address of EAO is generated by taking the last 20 bytes of the Keccak-256 hash of the public key and adding **0x** to the beginning.

```
0x5e97870f263700f46aa00d967821199b9bc5a120
```

Contract account

An account controlled by a smart contract.
They are also made of 20 bytes and start with 0x
the address is computed from the Keccak-256 hash of the **RLP-encoded** sender address and nonce. The resulting contract address is the last 20 bytes (160 bits) of that hash.

Recursive Length Prefix (RLP) is the kind of encoding used in Ethereum to transfer of data between nodes in a space-efficient format.

EOA vs Contract

EOA	Contract
Creating an account costs nothing	Creating a contract has a cost because you're using network storage
Can initiate transactions	Can only send messages in response to receiving a transaction
Transactions between EOA accounts can only be ETH/token transfers	Transactions from an EOA to a contract account can trigger code which can execute many different actions, such as transferring tokens or even creating a new contract
Made up of a cryptographic pair of keys.	don't have private keys.

Both account types have the ability to:

- Receive, hold and send ETH and tokens
- Interact with deployed smart contracts

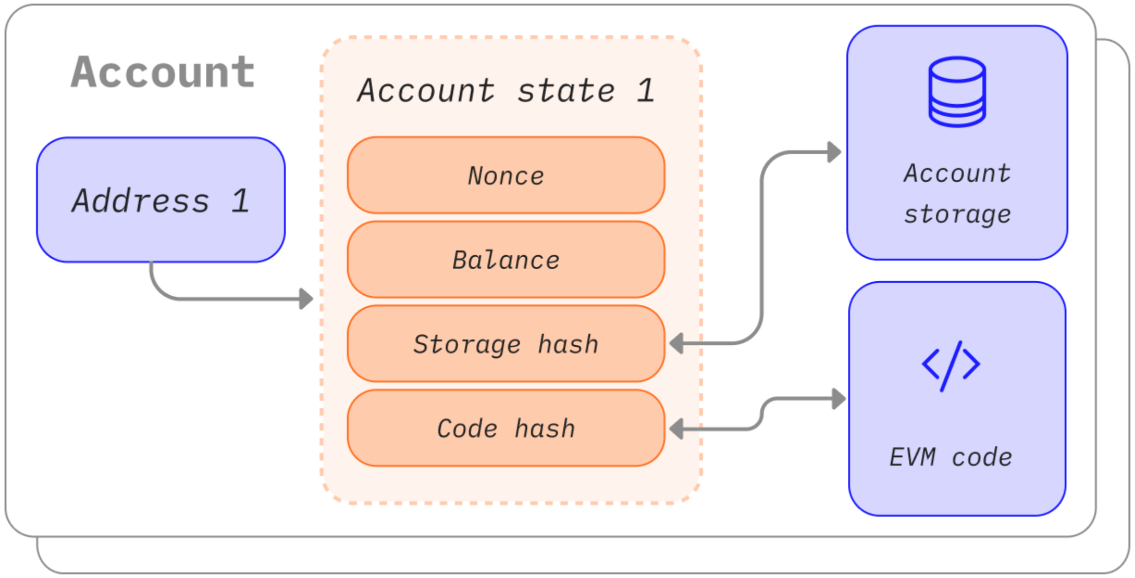
Ethereum Account

Nonce: A counter that indicates the number of transactions sent from an externally-owned account or the number of contracts created by a contract account. Only one transaction with a given nonce can be executed for each account.

Balance: The number of wei owned by this address

codeHash: It is an hash that refers to the code of an account on the Ethereum virtual machine (EVM). It is used as the key to retrieve the corresponding code from the database where it is stored.
In EOA it is the hash of an Empty String.

StorageRoot or Storage Hash: A 256-bit hash of the root node of a **Merkle Patricia Trie** that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash



Transactions

Transactions are signed messages originated by an externally owned account (EOA), transmitted by the Ethereum network, and recorded on the Ethereum blockchain.

They are the only things that can trigger a change of state, or cause a contract to execute in the EVM.

On Ethereum there are a few different types of transactions:

Regular transactions: a transaction from one account to another.

Contract deployment transactions: a transaction without a 'to' address, where the data field is used for the contract code.

Execution of a contract: a transaction that interacts with a deployed smart contract. In this case, 'to' address is the smart contract address.

Transactions

A submitted transaction includes the following information:

From: the address of the sender, that will be signing the transaction. This will be an externally-owned account as contract accounts cannot send transactions

To: the receiving address (if an externally-owned account, the transaction will transfer value. If a contract account, the transaction will execute the contract code)

Signature: the identifier of the sender. This is generated when the sender's private key signs the transaction and confirms the sender has authorized this transaction

Nonce: a sequentially incrementing counter which indicates the transaction number from the account

Value: amount of ETH to transfer from sender to recipient (denominated in WEI, where 1ETH equals 1e+18wei)

input data: optional field to include arbitrary data

gasLimit: the maximum amount of gas units that can be consumed by the transaction. The EVM specifies the units of gas required by each computational step

maxPriorityFeePerGas: the maximum price of the consumed gas to be included as a tip to the validator

maxFeePerGas: the maximum fee per unit of gas willing to be paid for the transaction (inclusive of baseFeePerGas and maxPriorityFeePerGas)

Transactions

Transaction example:

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```

The Noce

It is 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.

It ensures:

- Correct transaction order
- Protection against transaction replay/duplication

The Noce

Scenario

You need to send 6 ETH for something important.
Then you also send 8 ETH afterward.
Your account only has 10 ETH total.

Network Behavior Without Nonce

The network is decentralized; nodes may receive and relay the transactions in different orders.

Some nodes might see the 8 ETH transaction first and accept it, leaving the 6 ETH one to fail, that it is the opposite of what you intended. The result would be unpredictable and inconsistent across the network.

Network Behavior With Nonce

The 6 ETH transaction has a lower nonce (e.g., nonce = 3).
The 8 ETH transaction has nonce = 4.
The network must include the nonce 3 transaction before nonce 4.
Even if the 8 ETH transaction arrives first, it is placed on hold until nonce 3 is confirmed.

Outcome

The 6 ETH transaction succeeds consistently.
The 8 ETH transaction is rejected due to insufficient balance.

https://github.com/ethereumbook/ethereumbook/blob/develop/06transactions.asciidoc

The Noce

Scenario:

You send 2 ETH to buy something.
Your signed transaction is broadcast to the network for validation.

Risk Without Nonce:

Every transaction is visible to the network and can be copied.
If the transaction had no nonce, anyone (including the recipient or an attacker) could simply rebroadcast your signed transaction again and again.
Each replay would transfer another 2 ETH, draining your account without your consent.

Protection With Nonce:

Each transaction carries a unique nonce, and the network records which nonces have been used.
Once nonce 10 (for example) has been processed, it cannot be reused.
Any attempt to replay the transaction is automatically invalid and rejected.

Outcome:

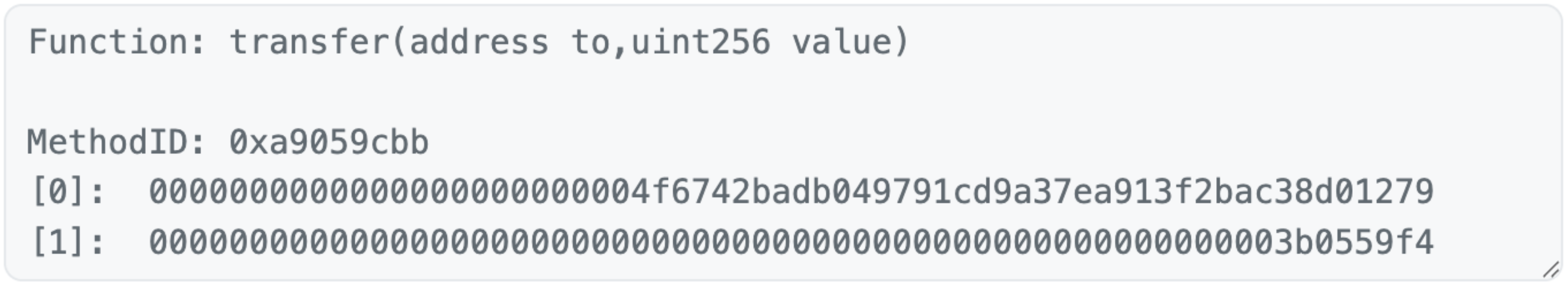
No one can duplicate your payments.
Each transaction can only happen once, guaranteeing balance safety.

The input field

Data can be added to the data field of a transaction whenever a user deploys or makes a call to a function in a smart contract. This data is a message to the smart contract that points to the function that will be executed.

The **input field** contains the data that is included in the transaction. If this field is empty, the transaction is a transfer between users and doesn't involve a smart contract.

The first **four bytes** specify which function to call, using the hash of the function's name and arguments. The rest of the call data is the arguments.



```
Function: transfer(address to,uint256 value)

MethodID: 0xa9059cbb
[0]:  0000000000000000000000004f6742badb049791cd9a37ea913f2bac38d01279
[1]:  000000000000000000000000000000000000000000000000000000003b0559f4
```

So we know that the to address is `4f6742badb049791cd9a37ea913f2bac38d01279`.
The value is `0x3b0559f4 = 990206452`

https://www.4byte.directory/signatures/?bytes4_signature

The Application Binary Interface

Code

```
contract Counter {
         uint256 public number;
         function setNumber(uint256 newNumber) public { number = newNumber; }
         function increment() public { number++; }
}
```

What we see in the blockchain

```
0x6080604052348015610010576000080fd5b5060f78061001f6000396000f3fe608060405230
48015600f57600080fd5b5060043610603c5760003560e01c80633fb5c1cb146041578063830
81f58a146053578063d09de08a14606d575b600080fd5b6051604c366004608356565b6000555
65b005b605b60005481565b60405190815260200160405180910390f35b6051600080549080
607c83609b565b919050555056565b6000602082840312156094576000080fd5b5035919050565
b600060018201606ba57634e487b7160e01b6000052601160045260246000fd5b506001019056
fea2646970667358221220d7dee18f1939c15cddbe7e354ba7008730b3b1f831d9e3192e0fc
d25106ce31064736f6c63430008160033
```

The Application Binary Interface

So, to interact with the smart contract, we should probably know:

What functions can we call:
- How to call the functions
- What parameters can we pass to the functions

The smart contract **ABI** tells us how to interact with this bunch of nonsense hex on the chain because, remember, the human-readable code is not stored on chain; the hex does!

The ABI (Application Binary Interface) in Ethereum is a standardized format that defines how to interact with smart contracts. It describes the contract's functions, inputs, outputs, and data types in a machine-readable way.

It is useful because:

- It allows applications (like wallets, DApps, and scripts) to correctly encode function calls and decode responses.

- Without the ABI, you could see the contract on-chain, but you wouldn't know how to call its functions or interpret the returned data.

The Application Binary Interface

```
{
            "type": "function",
            "name": "setNumber",
            "inputs": [
                {
                    "name": "newNumber",
                    "type": "uint256",
                    "internalType": "uint256"
                }
            ],
            "outputs": [],
            "stateMutability": "nonpayable"
        }
```

GAS

Gas refers to the unit that measures the amount of computational effort required to execute specific operations on the Ethereum network.

The **gas fee** is the amount of gas used to do some operation, multiplied by the cost per unit gas. The fee is paid regardless of whether a transaction succeeds or fails.

Gas fees have to be paid in Ether (ETH) and are usually quoted in **Gwei**.

A gwei is equal to one-billionth of an ETH (0.000000001 ETH or 10-9 ETH).

Total Gas = unit of gas (base fee + priority fee)

The ***base fee*** is set by the protocol and it is the minimum amount that you have to pay for your transaction to be considered valid.

The priority fee is a tip added to the base fee to make your transaction **attractive** by validators.

The 'correct' value of the priority fee is determined by the the network usage, in a kind of virtual auction.

Thus, a transaction without priority fee is **valid,** but it is unlikely that it will be included because it offers no incentive to validators.

GAS – Why this mechanism

(Recall) first-price auction system: everyone submits a bid, and then if they get included they pay exactly the bid that they submit.

Mismatch between volatility of transaction fee levels and social cost of transactions.
Under the first-price auction system, Ethereum's gas fees often changed much more than the actual cost of processing transactions. This volatility made the network inefficient and sometimes unfair, since users could end up paying very different amounts for the same type of transaction. In short, fees rose and fell sharply even though the real work done by the network stayed almost the same.

Needless delays for users
Fixed block sizes can cause delays in getting transactions accepted. This creates the need for a mechanism that can dynamically increase or decrease the block size based on network demand.

Inefficiencies of first price auctions
It is hard to estimate the fee and the there is no simple strategy for choosing the optimal bid price.

Example: if you value a tx getting included right now at $1, you would be willing to bid anything up to $1, but if everyone else is bidding $0.05, then you could keep more money by bidding $0.08 instead.

GAS – Why this mechanism

Instability of blockchains with no block reward

In the long run, blockchains where there is no issuance (like Bitcoin) at present intend to switch to rewarding miners entirely through transaction fees. However, there are known issues with this that likely leads to a lot of instability.

Counter inflation

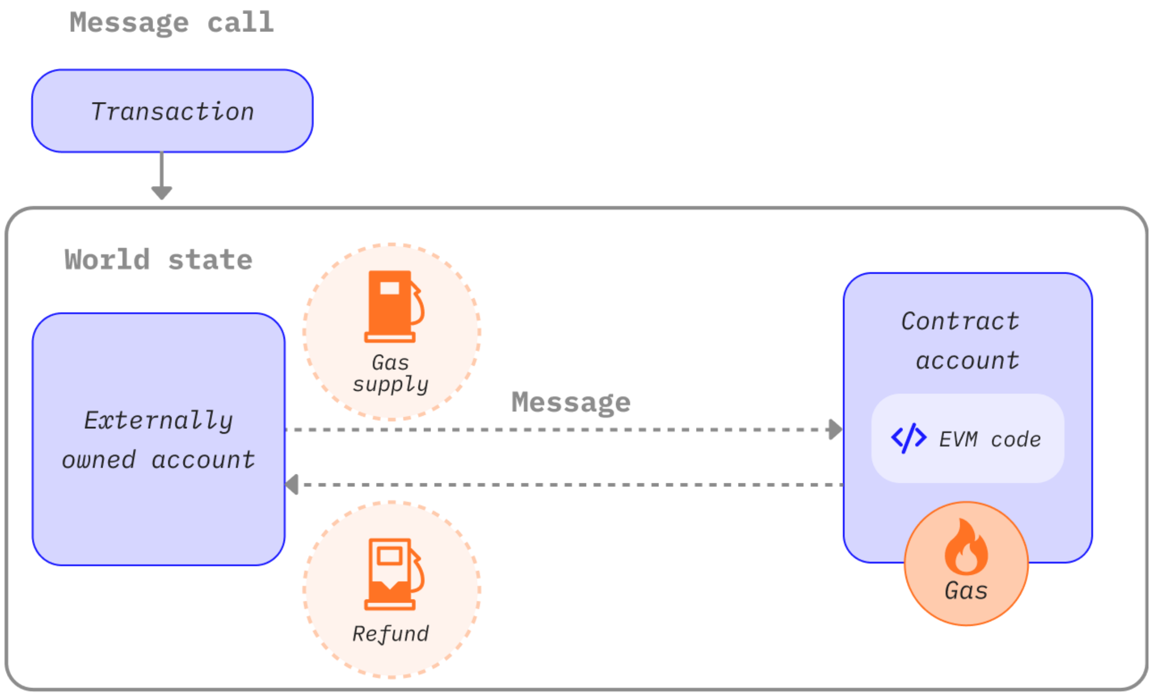
Ethereum has not a cap to the Ether maximum supply, the burning mechanism will help in reduce the inflation and in some case make Ether a deflationary coin.

https://www.cs.princeton.edu/~arvindn/publications/mining_CCS.pdf

GAS

Gas fees help keep the Ethereum network secure.
By requiring a fee for every computation executed on the network, we prevent bad actors from spamming the network. In order to avoid accidental or hostile infinite loops or other computational wastage in code.
Each transaction is required to set a limit to how many computational steps of code execution it can use. The fundamental unit of computation is "gas".

Although a transaction includes a limit, any gas not used in a transaction is returned to the user (e.g., max fee - (base fee + tip) is returned).



Gas Example

A standard Ethereum transaction requires 21000 unit of gas

The base fee: 190 gwei

Maxpiorityfee: 10 gwei

Amount of ETH to send = 1 ETH

`21000 * (190+10) = 4 200 000`

Burnt amount: `190*21000 = 3990000` gwei

Amount collected by the validator: `21000 * 10 = 210000` gwei

Transaction gas limit

The gas limit refers to the maximum amount of gas you are willing to consume on a transaction. More complicated transactions involving smart contracts require more computational work, so they require a higher gas limit than a simple payment.

- If the gas limit is set higher, unused gas is refunded.
- If the gas limit is set too low, the transaction fails before being included in a block and no gas is spent.
- If gas runs out during execution (e.g., within a smart contract), all changes are reverted but all provided gas is consumed.

Base fee

Every block has its own base fee.

The base fee of the block X is determined by the block before (X-1).

When a block is created the base fee is **burned.**

Block size: Each block has a target size **of half the current block gas limit**, but the size of blocks will increase or decrease in accordance with network demand, up until the block limit is reached (2x the target block size).The block gas limit can be adjusted upwards or downwards by a factor of 1/1024 from the previous block's gas limit.

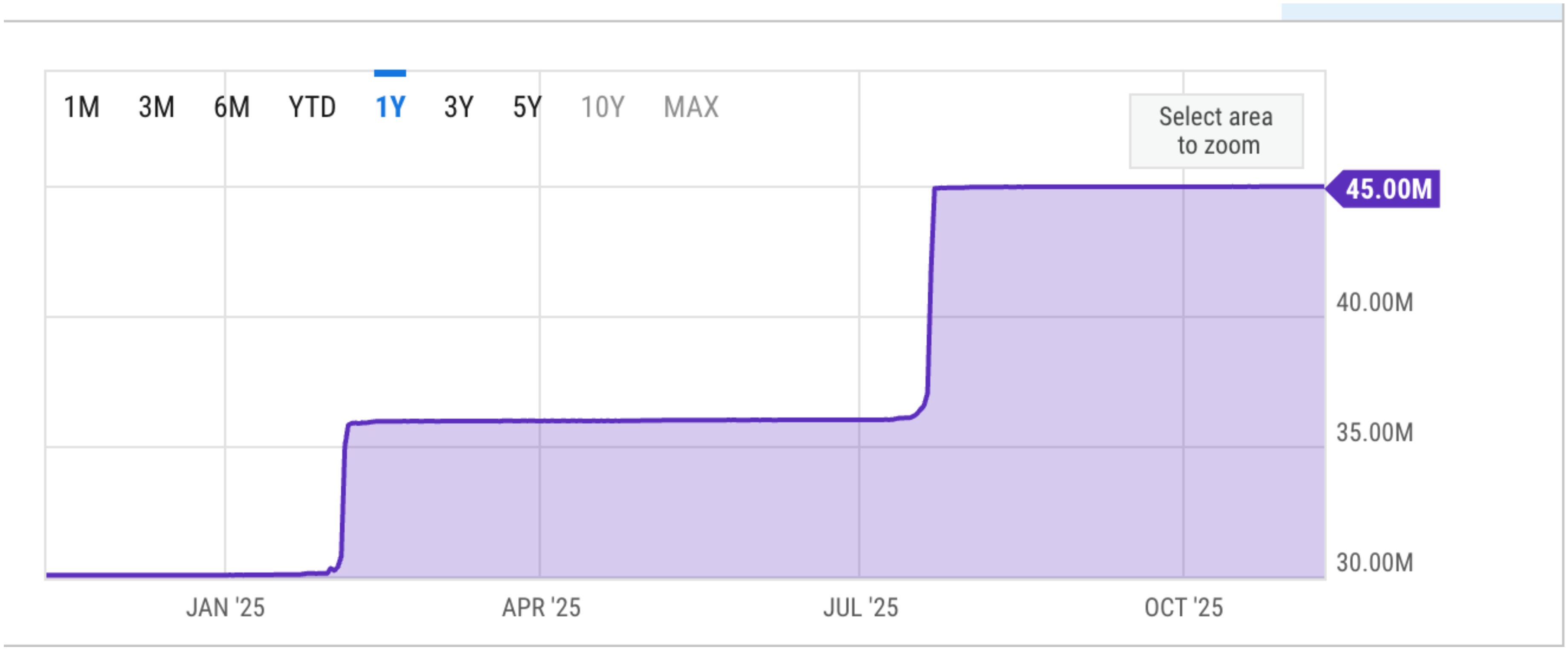
Block Gas limit: actual block gas limit: **45 million gas**.
The block gas limit can be adjusted upwards or downwards by a factor of 1/1024 from the previous block's gas limit.

Base fee adjustment: The amount by which the base fee is adjusted is proportional to how far the current block size is from the target. This is a linear calculation from -12.5% for an empty block, 0% at the target size, up to +12.5% for a block reaching the gas limit.

Base fee

Block Number	Included Gas	Fee Increase	Current Base Fee
1	22,5M	0%	100 gwei
2	45M	0%	100 gwei
3	45M	12.5%	112.5 gwei
4	45M	12.5%	126.6 gwei
5	45M	12.5%	142.4 gwei
6	45M	12.5%	160.2 gwei
7	45M	12.5%	180.2 gwei
8	45M	12.5%	202.7 gwei
30	45M	12.5%	2705.6 gwei
...	...	12.5%	...
50	45M	12.5%	28531.3 gwei
...	...	12.5%	...
100	45M	12.5%	10302608.6 gwei

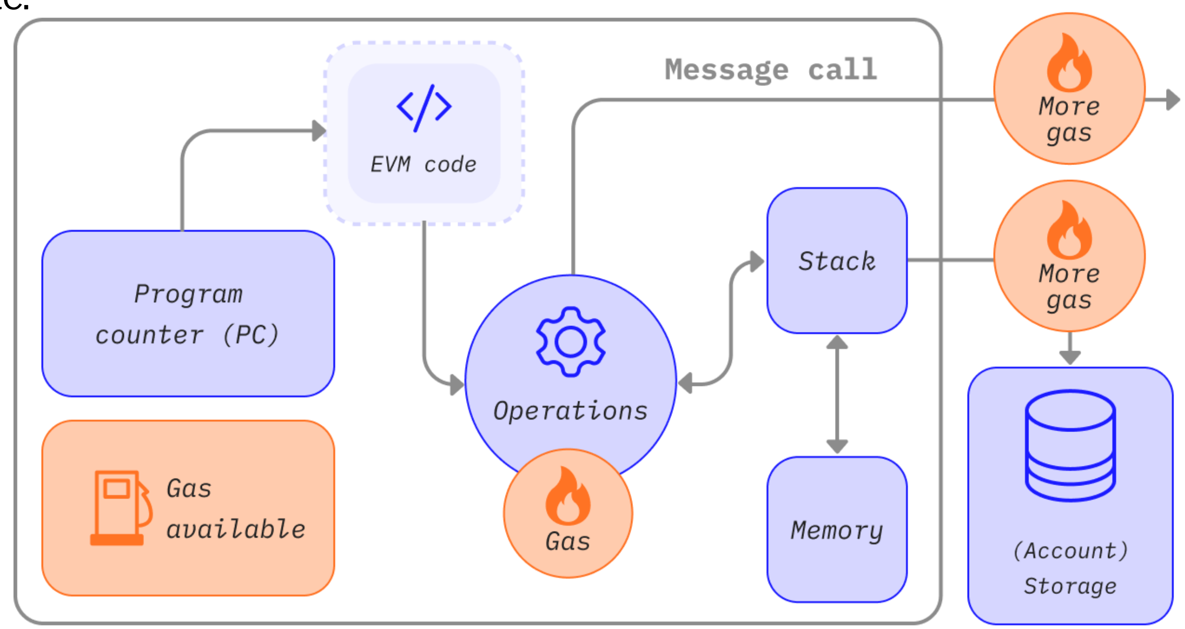
Average gas limit



Gas and smart contract

Gas unit represent the actual number of gas units used by the smart contract interaction triggered by the transaction.

At low level a smart contract mart contract bytecode executes as a number of EVM opcodes, which perform standard stack operations like XOR, AND, ADD, SUB, etc. The EVM also implements a number of blockchain-specific stack operations, such as ADDRESS, BALANCE, BLOCKHASH, etc.



Gas is required for any transaction that involves a smart contract.

However, function that do not alter the state of the smart contract will not require any gas. This function are known as `view` or `pure`.

Gas and smart contract

Stack	Name	Gas
00	STOP	0
01	ADD	3
02	MUL	5
03	SUB	3
04	DIV	5
05	SDIV	5
06	MOD	5
07	SMOD	5
08	ADDMOD	8
09	MULMOD	8

Sources

https://ethereum.org/developers

https://ycharts.com/indicators/ethereum_average_gas_limit

https://eips.ethereum.org/EIPS/eip-1559

https://ethresear.ch/t/first-and-second-price-auctions-and-improved-transaction-fee-markets/2410