Lesson 5 - EVM / Test Networks / Wallets

This week

Mon : EVM / Test Networks / Wallets
Tues : Team game
Weds : Security Introduction
Thurs: Development tools

EVM

Topics

- Ethereum state
- Transaction and Block details
- The EVM
  Storage
- EVM Languages
- Test Networks
- Wallets

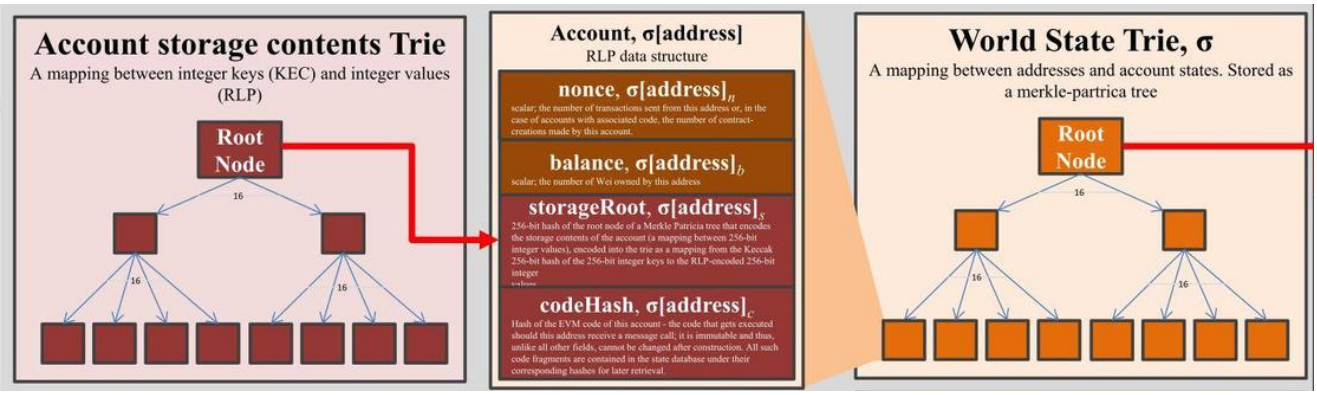
Ethereum State

There are 3 Tries

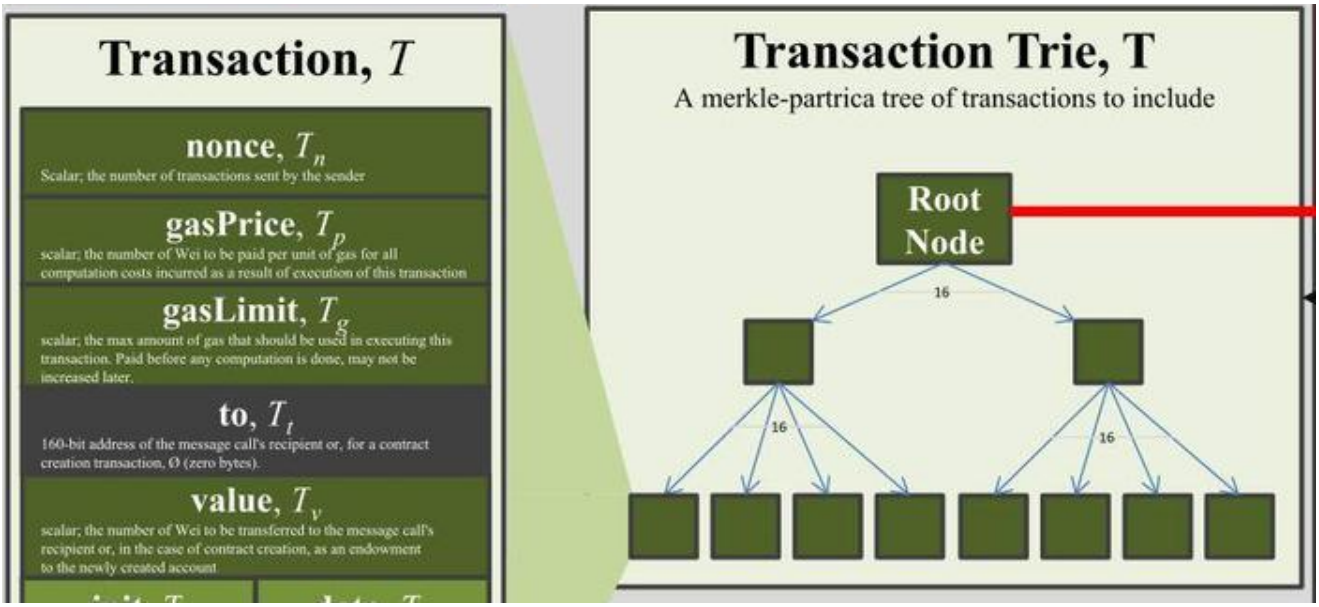
- World State
- Transaction
- Transaction Receipt

See ：Ethereum State Trie Architecture Explained**

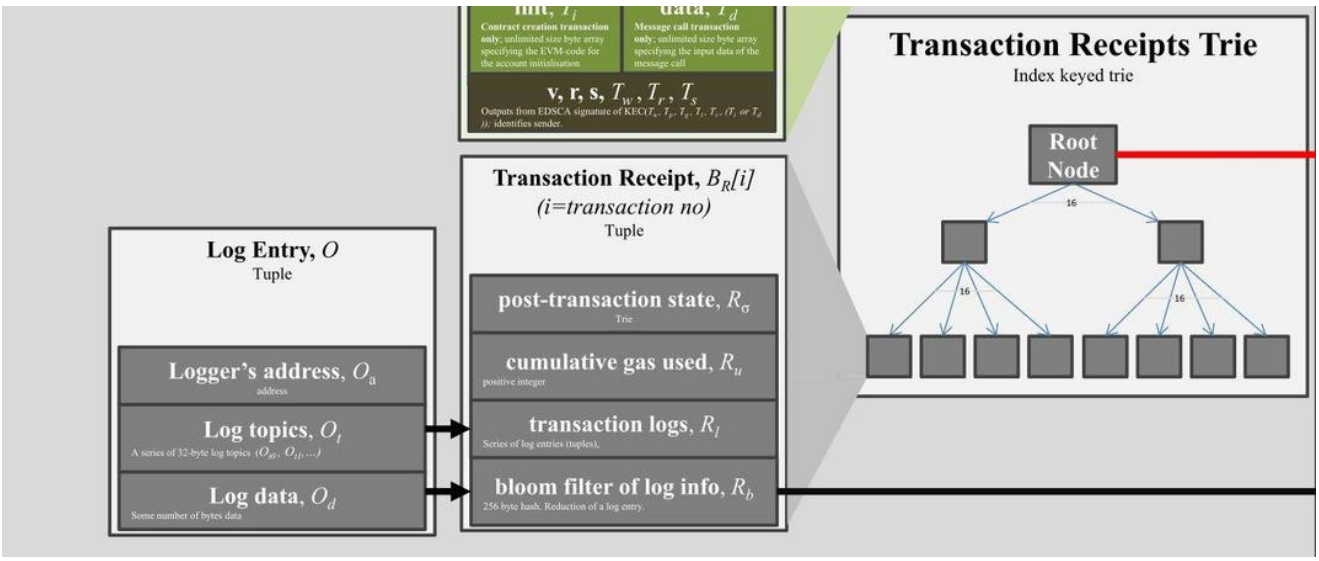
World and Account State



Transactions



Transaction Receipts and Logs



Transaction and Transaction Receipt Tries

Purpose:

- Transaction Tries: records transaction request
- Transaction Receipt Tries: records the transaction outcome

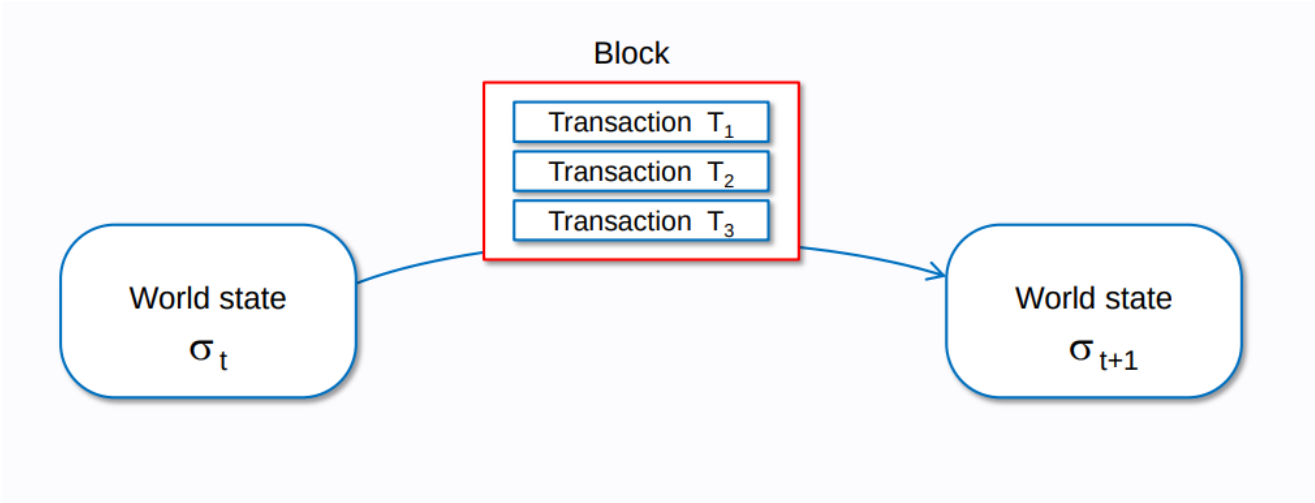
Parameters used in composing a Transaction Trie details in section 4.3 of the [yellow paper]

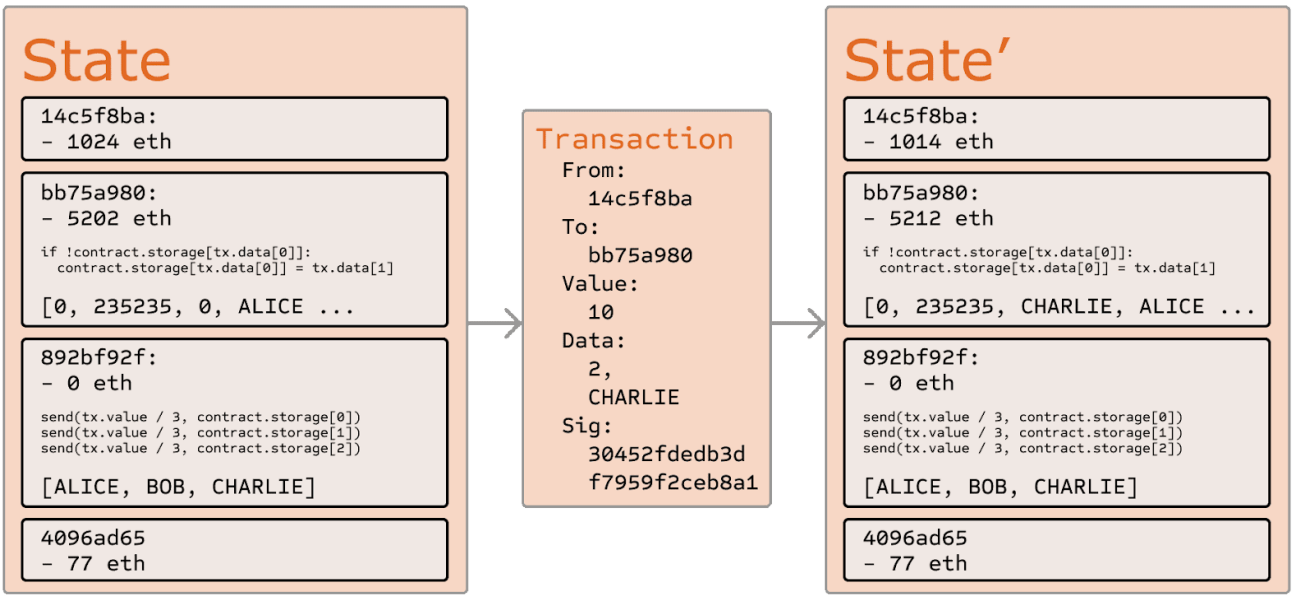
- nonce,
- gas price,
- gas limit,
- recipient,
- transfer value,
- transaction signature values, and
- account initialization (if transaction is of contract creation type), or transaction data (if transaction is a message call)

Parameters used in composing a Transaction Receipt Trie details in section 4.4.1 of the [yellow paper]:

- post-transaction state,
- the cumulative gas used,
- the set of logs created through execution of the transaction, and
- the Bloom filter composed from information in those logs

Ethereum Transactions





Some practical points about transaction selection

- Miners choose which transactions to include in a block
- Miners can add their own transactions to a block
- Miners choose the order of transactions in a block
- Your transaction is in competition with other transactions for inclusion in the block

We will talk about the consequences of these points in our MEV lesson.

Transaction Processing

Before the transaction executes it needs to pass some validity tests

- The transaction follows the rules for the encoding scheme (now Simple Serialize SSZ, previous RLP)
- The signature on the transaction is valid.
- The nonce on the transaction is valid, i.e. it is equivalent to the sender account's current nonce.
- The gas_limit is greater than or equal to the intrinsic_gas used by the transaction.
- The sender's account balance contains the cost required in up-front payment.

(For details of SSZ see docs)

View Functions and modifying state

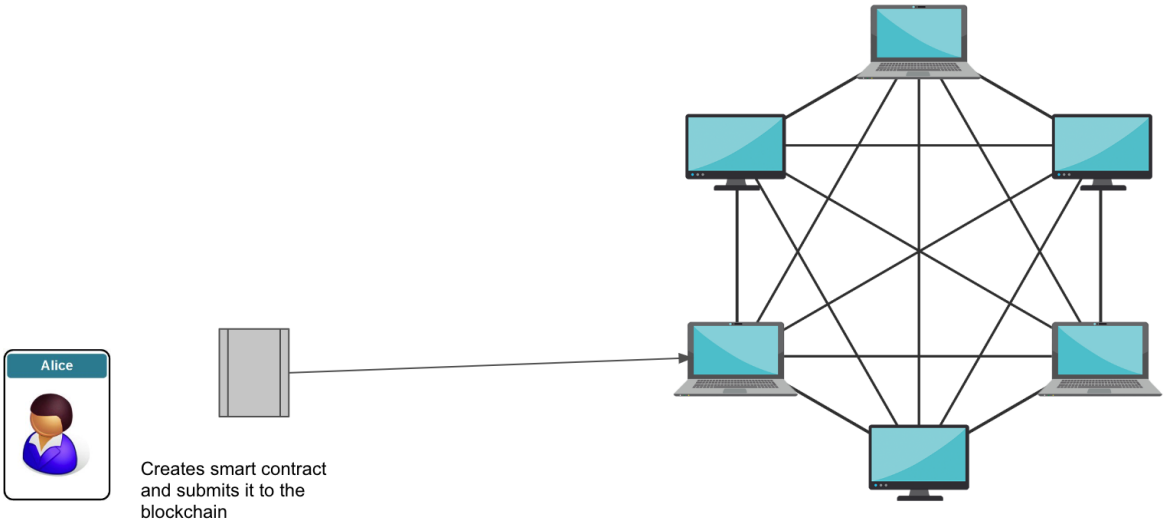
From documentation

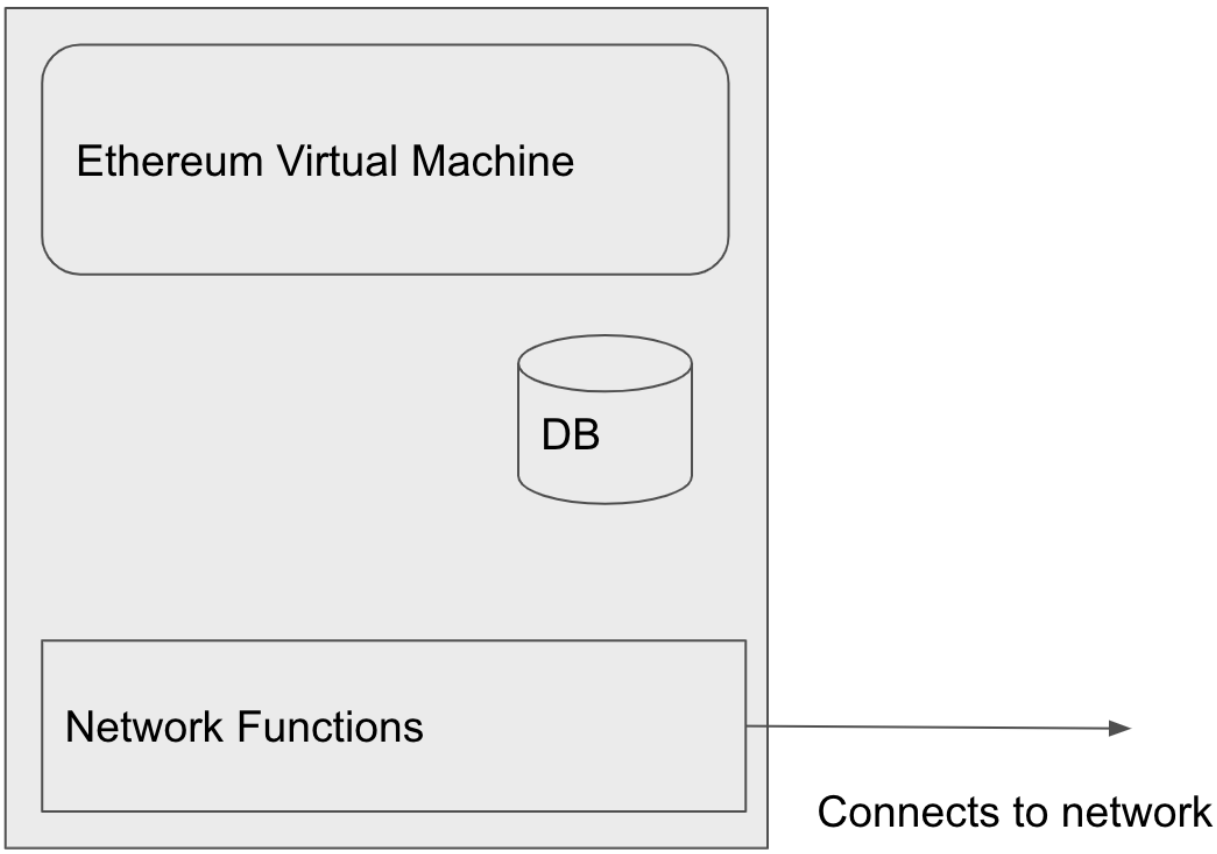
The following statements are considered modifying the state:

1. Writing to state variables.
2. Emitting events
3. Creating other contracts
4. Using `selfdestruct`.
5. Sending Ether via calls.
6. Calling any function not marked `view` or `pure`.
7. Using low-level calls.
8. Using inline assembly that contains certain opcodes.

Note : Getter methods are automatically marked `view`.

The EVM





Ethereum contracts run within the Ethereum Virtual Machine.
This is a virtual machine running on the node software.
It is a self contained and restricted environment.

Contracts are much more dependent on this environment than say a python program, which is fairly independent of the environment on which it runs.

Ethereum contracts and dApp design is very much shaped by the constraints of the blockchain and the EVM.

The EVM is a 'stack machine', the stack has a maximum size of 1024 slots.

Stack items have a size of 256 bits; in fact, the EVM is a 256-bit word machine (this facilitates Keccak256 hash scheme and elliptic-curve computations).

Data areas

Data can be stored in

- Stack
- Calldata
- Memory
- Storage
- Code
- Logs

Storage, memory and calldata

See documentation

Storage

Storage data is permanent, forms part of the smart contract's state and can be accessed across all functions. Storage data location is expensive and should be used only if necessary. The storage keyword is used to define a variable that can be found in storage location.

Memory

Memory data is stored in a temporary location and is only accessible within a function. Memory data is normally used to store data temporarily whilst executing logic within a function. When the execution is completed, the data is discarded. The memory keyword is used to define a variable that is stored in memory location.

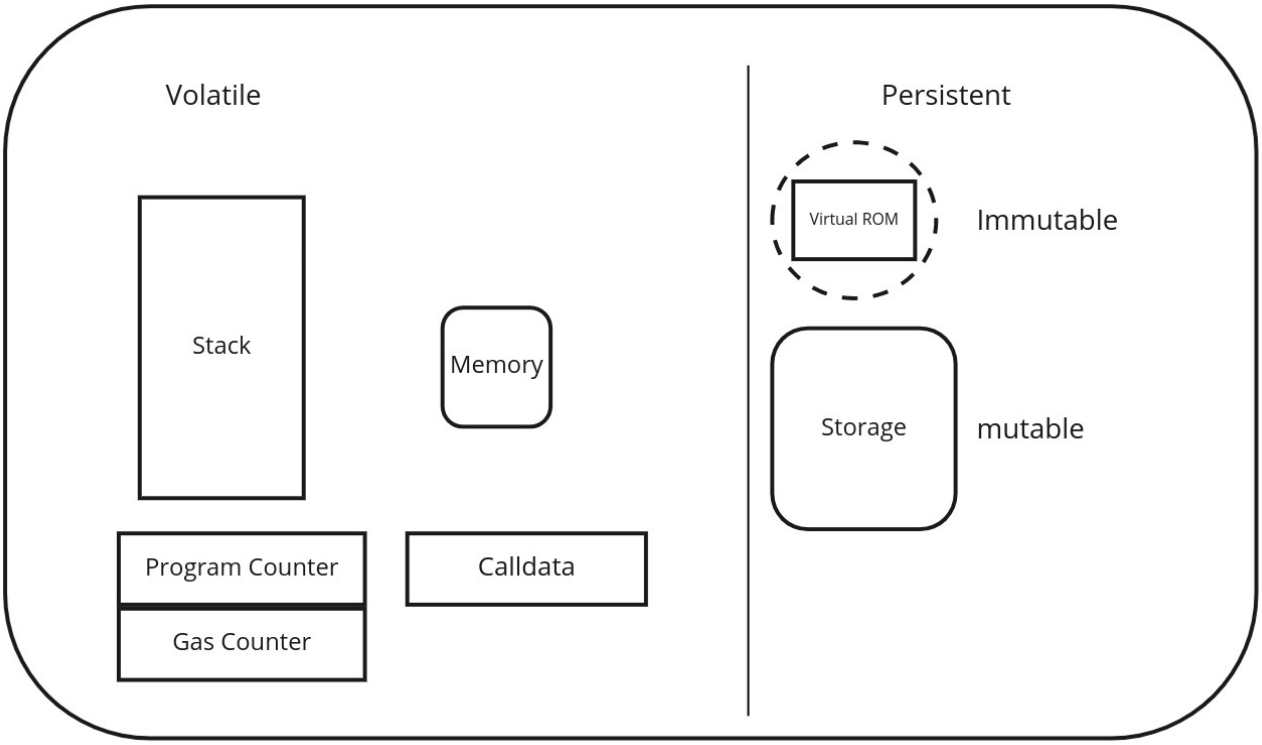
Calldata

Calldata is the location where external values from outside a function into a function are stored. It is a non-modifiable and non-persistent data location. The calldata keyword is required to define a variable stored in the calldata location.

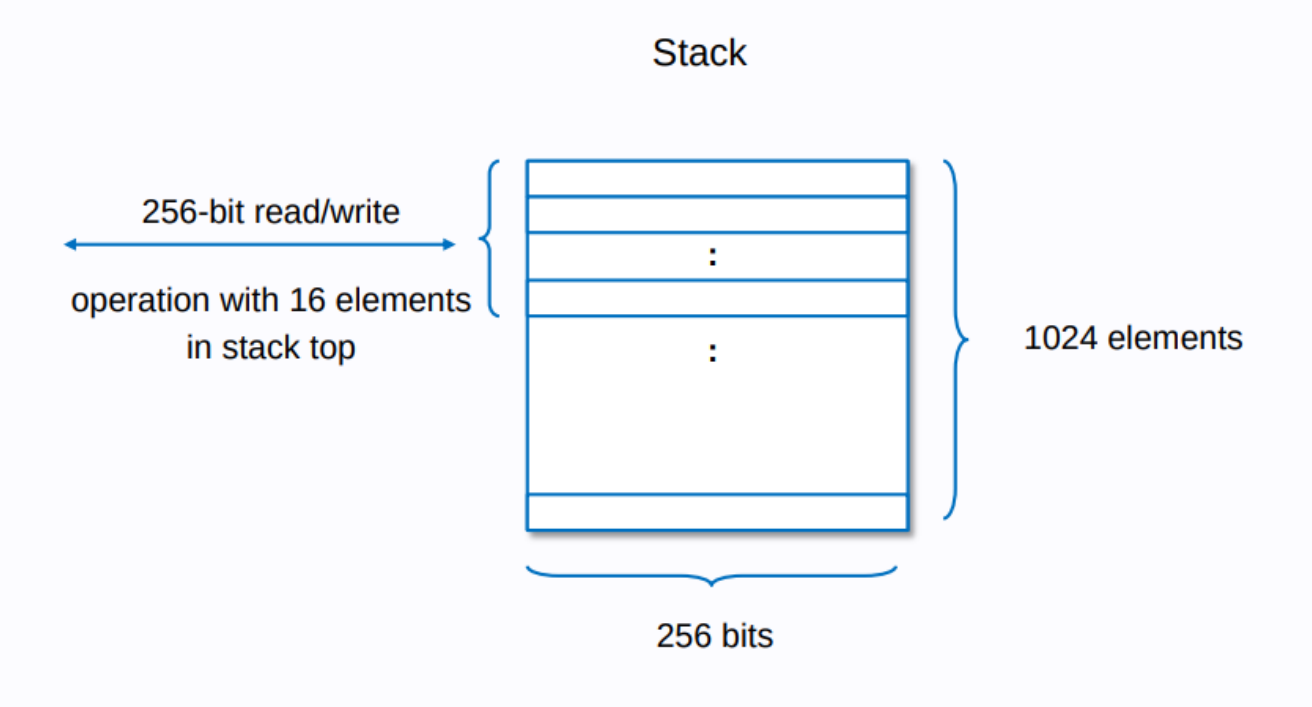
The difference between calldata and memory is subtle, calldata variables cannot be changed.
For example :

```
pragma solidity ^0.8.0;

contract Test {

function memoryTest(string memory _exampleString)
public pure
returns (string memory) {
    _exampleString = "example";  // You can modify memory
    string memory newString = _exampleString;
    // You can use memory within a function's logic
    return newString;  // You can return memory
}

function calldataTest(string calldata _exampleString) external
pure returns (string calldata) {
    // cannot modify _exampleString
    // but can return it
    return _exampleString;
}
}
```

EVM Components

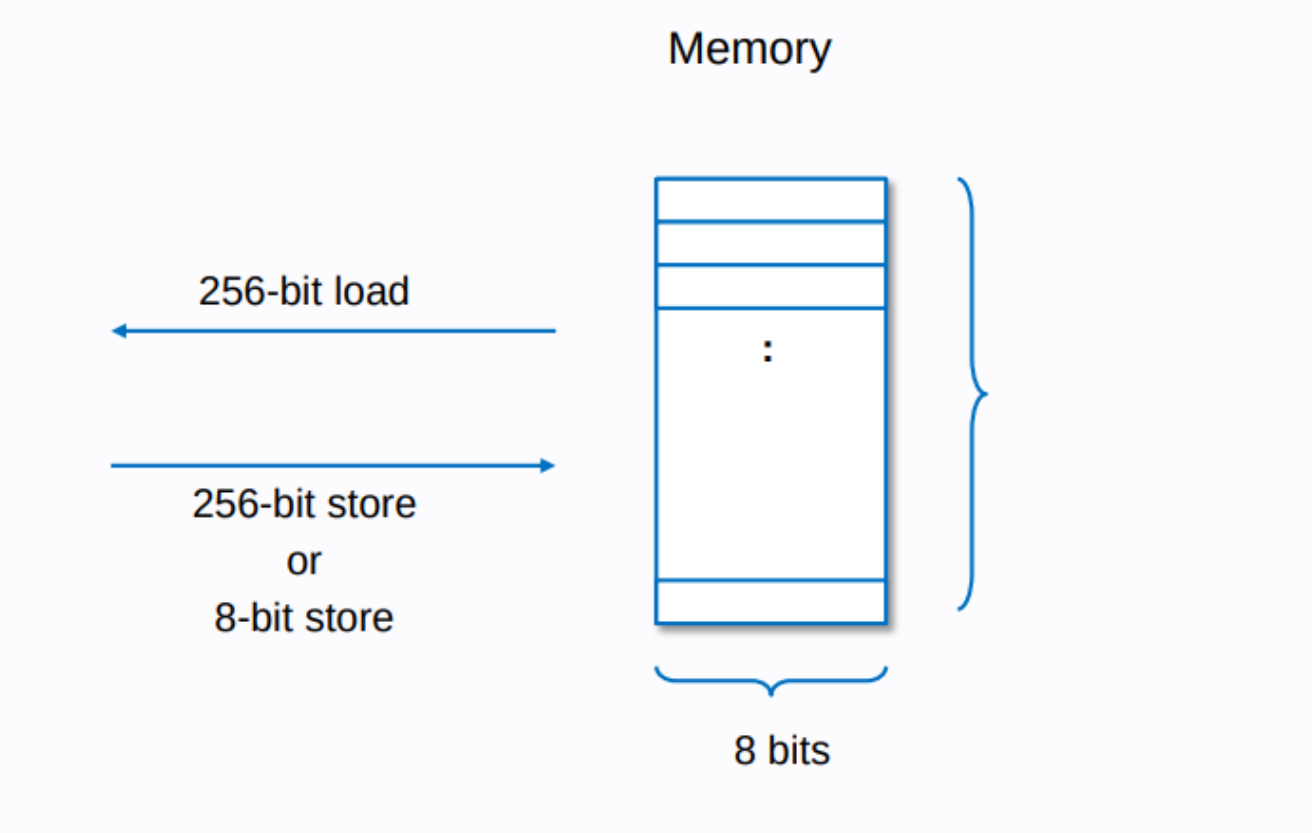


The Stack



The top 16 items can be manipulated or accessed at once (or stack too deep error)

Memory



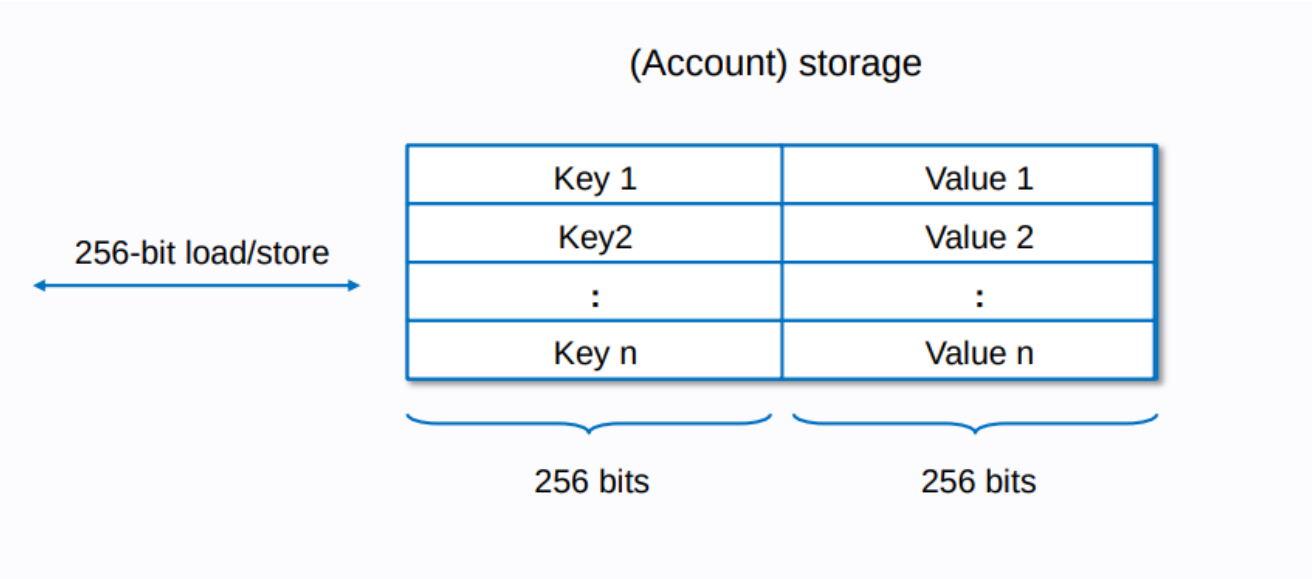
Memory is a byte-array. Memory starts off zero-size, but can be expanded in 32-byte chunks by simply accessing or storing memory at indices greater than its current size. Since memory is contiguous, it does save gas to keep it packed and shrink its size, instead of having large patches of zeros.

Memory Expansion

From Explanation

When your contract writes to memory, you have to pay for the number of bytes written. If you are writing to an area of memory that hasn't been written to before there is an additional memory expansion cost for using it for the first time.
Memory is expanded in 32 bytes (256-bit) increments when writing to previously untouched memory space.

Storage

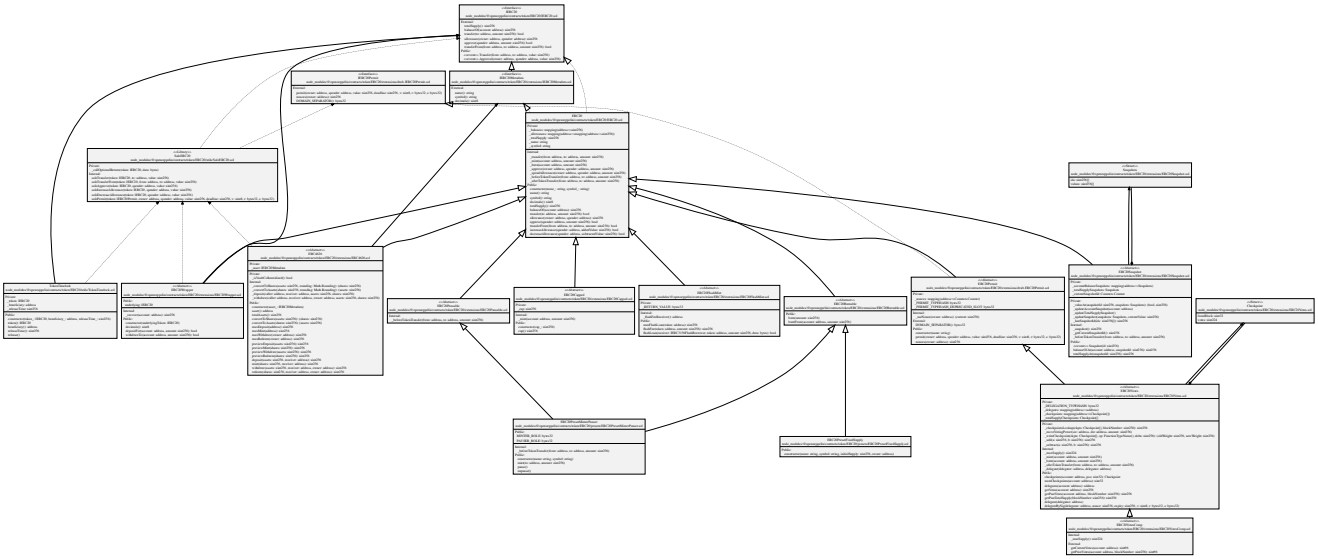


See Documentation

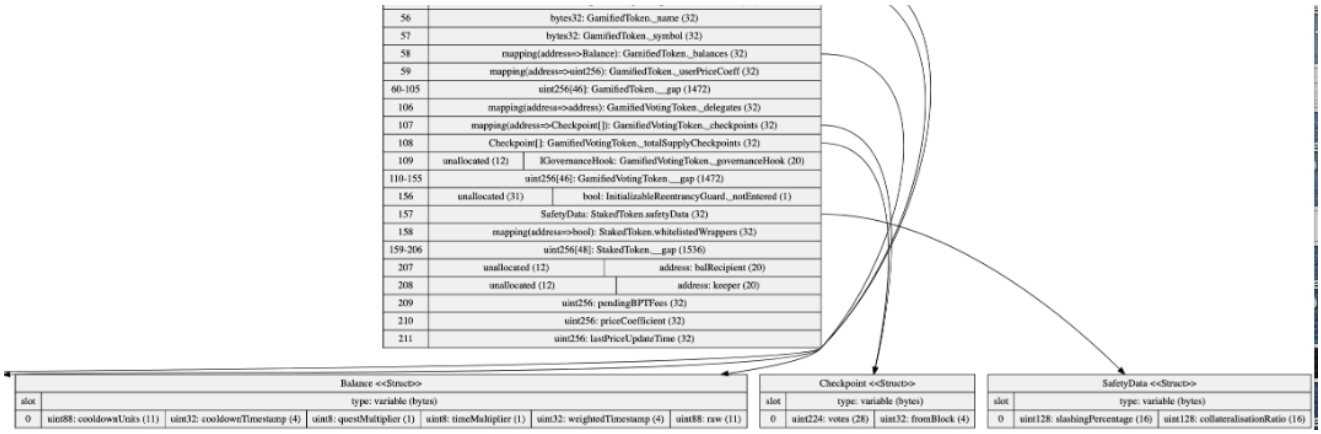
Sol2UML Visualisation Tool

See repo

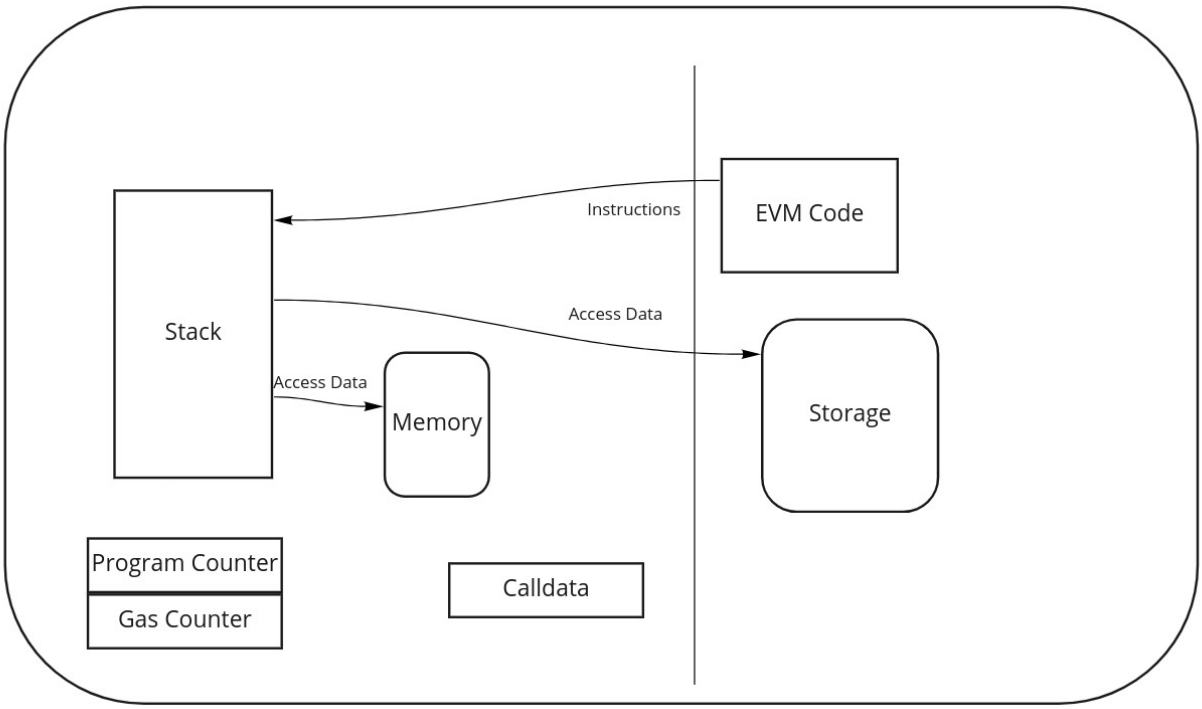
It provides visualisation of storage, plus UML diagrams for the contract.



FiatTokenV2_1 <<Contract>> 0xa2327a938febf5fec13bacfb16ae10ecbc4cbdcf			
slot	type: <inherited contract>.variable (bytes)		
0	unallocated (12)	address: Ownable._owner (20)	
1	unallocated (11)	bool: Pausable.paused (1)	address: Pausable.pauser (20)
2	unallocated (12)	address: Blacklistable.blacklister (20)	
3	mapping(address=>bool): Blacklistable.blacklisted (32)		
4	string: FiatTokenV1.name (32)		
5	string: FiatTokenV1.symbol (32)		
6	unallocated (31)	uint8: FiatTokenV1.decimals (1)	
7	string: FiatTokenV1.currency (32)		
8	unallocated (11)	bool: FiatTokenV1.initialized (1)	address: FiatTokenV1.masterMinter (20)
9	mapping(address=>uint256): FiatTokenV1.balances (32)		
10	mapping(address=>mapping(address=>uint256)): FiatTokenV1.allowed (32)		
11	uint256: FiatTokenV1.totalSupply_ (32)		
12	mapping(address=>bool): FiatTokenV1.minters (32)		
13	mapping(address=>uint256): FiatTokenV1.minterAllowed (32)		
14	unallocated (12)	address: Rescuable._rescuer (20)	
15	bytes32: EIP712Domain.DOMAIN_SEPARATOR (32)		
16	mapping(address=>mapping(bytes32=>bool)): EIP3009._authorizationStates (32)		
17	mapping(address=>uint256): EIP2612._permitNonces (32)		
18	unallocated (31)	uint8: FiatTokenV2._initializedVersion (1)	


56 bytes32: GamifiedToken._name (32)
57 bytes32: GamifiedToken._symbol (32)
58 mapping(address=>Balance): GamifiedToken._balances (32)
59 mapping(address=>uint256): GamifiedToken._userPriceCoeff (32)
60-105 uint256[46]: GamifiedToken.__gap (1472)
106 mapping(address=>address): GamifiedVotingToken._delegates (32)
107 mapping(address=>Checkpoint[]): GamifiedVotingToken._checkpoints (32)
108 Checkpoint[]: GamifiedVotingToken._totalSupplyCheckpoints (32)
109 unallocated (12) IGovernanceHook: GamifiedVotingToken._governanceHook (20)
110-155 uint256[46]: GamifiedVotingToken.__gap (1472)
156 unallocated (31) bool: InitializableReentrancyGuard._notEntered (1)
157 SafetyData: StakedToken.safetyData (32)
158 mapping(address=>bool): StakedToken.whitelistedWrappers (32)
159-206 uint256[48]: StakedToken.__gap (1536)
207 unallocated (12) address: balRecipient (20)
208 unallocated (12) address: keeper (20)
209 uint256: pendingBPTFees (32)
210 uint256: priceCoefficient (32)
211 uint256: lastPriceUpdateTime (32)
Balance <<Struct>>
slot type: variable (bytes)
0 uint88: cooldownUnits (11) uint32: cooldownTimestamp (4) uint8: questMultiplier (1) uint8: timeMultiplier (1) uint32: weightedTimestamp (4) uint88: raw (11)
Checkpoint <<Struct>>
slot type: variable (bytes)
0 uint224: votes (28) uint32: fromBlock (4)
SafetyData <<Struct>>
slot type: variable (bytes)
0 uint128: slashingPercentage (16) uint128: collateralisationRatio (16)

Code Execution



EVM Languages

- Solidity
  - The most popular programming language for Ethereum contracts
- LLL
  - Low-level Lisp-like Language
- Vyper
  - A language with overflow-checking, numeric units but without unlimited loops
- Yul / Yul+
  - An intermediate language that can be compiled to bytecode for different backends. Support for EVM 1.0, EVM 1.5 and Ewasm is planned, and it is designed to be a usable common denominator of all three platforms.
- FE
  - Statically typed language Inspired by Rust and Python
- Huff see article
  - Low level language
- Pyramid Scheme (experimental)
  - A Scheme compiler into EVM that follows the SICP compilation approach
- Flint
  - A language with several security features: e.g. asset types with a restricted set of atomic operations
- LLLL
  - An LLL-like compiler being implemented in Isabelle/HOL
- HAseembly-evm
  - An EVM assembly implemented as a Haskell DSL
- Bamboo (experimental)
  - A language without loops

Vyper

- Pythonic programming language
- Strong typing
- Small and understandable compiler code
- Deliberately has less features than Solidity with the aim of making contracts more secure and easier to audit. Vyper does not support
  - Modifiers
  - Inheritance
  - Inline assembly
  - Function overloading
  - Operator overloading
  - Recursive calling
  - Infinite-length loops

FE

See :Repo

- Statically typed language for the Ethereum Virtual Machine (EVM).
- Inspired by Python and Rust.
- Aims to be easy to learn -- even for developers who are new to the Ethereum ecosystem.
- Fe development is still in its early stages, the language had its alpha release in January 2021.

Features

- Bounds and overflow checking
- Decidability by limitation of dynamic program behavior
- More precise gas estimation (as a consequence of decidability)
- Static typing
- Pure function support
- Restrictions on reentrancy
- Static looping
- Module imports
- Standard library
- Usage of YUL IR to target both EVM and eWASM
- WASM compiler binaries for enhanced portability and in-browser compilation of Fe contracts
- Implementation in a powerful, systems-oriented language (Rust) with strong safety guarantees to reduce risk of compiler bugs

Gas Refunds

Since EIP-3529 gas refunds are not given for self destructing contracts, and the amount of refund for storage has been reduced.

Test Networks

See Docs

Goerli is a proof-of-stake testnet. It is expected to be maintained long-term as a stable testnet for application developers.

- Website
- GitHub
- Etherscan

Goerli faucets

- Goerli faucet
- Chainlink faucet
- Alchemy Goerli Faucet

Sepolia

Sepolia is a proof-of-stake testnet. Although Sepolia is still running, it is not currently planned to be maintained long-term. Before undergoing The Merge in June 2022, Sepolia was a proof-of-work testnet.

- Website
- GitHub
- Otterscan
- Etherscan

Sepolia faucets

- Sepolia faucet
- FaucETH

Deprecated Networks

- Ropsten
- Rinkeby
- Kovan

There are layer 2 test networks, we will cover these in week 4

Wallets

See Docs

Unstoppable wallet
iOS | Android

Zerion Wallet
iOS | Android | macOS

MetaMask
iOS | Android | Chromium | Firefox

Bitcoin.com Wallet
iOS | Android | macOS

Tally Ho!
Chromium | Firefox

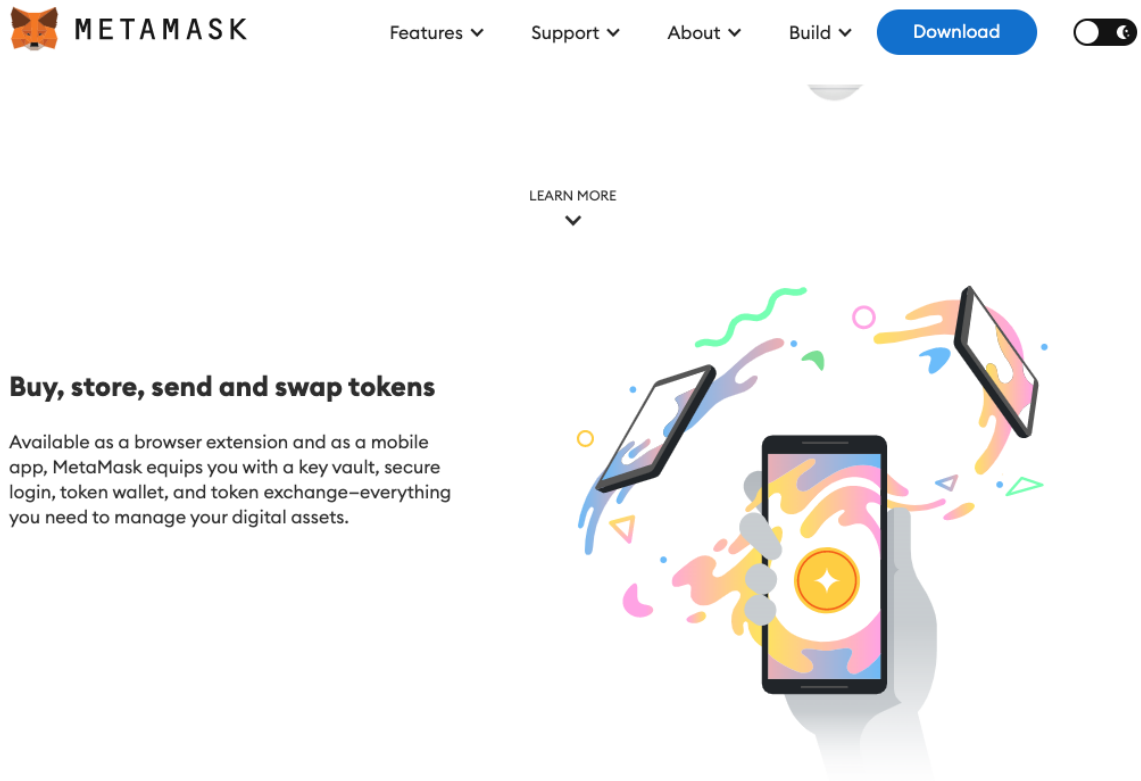
1inch Wallet
iOS | Android

Enkrypt
Chromium

and many more

Metamask

See Docs



References

DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Mastering Ethereum by Andreas Antonopoulos
White paper
Beige Paper
Yellow Paper
EVM languages
Noxx Articles about the EVM