Lesson 3

Plan for this week

Mon : Design Patterns / EVM Review
Tues : Solidity Types, Functions / Assembly Introduction
Weds : Solidity Best Practices / Tips & Tricks
Thurs : MEV / ETH 2.0

Design Patterns

Class question - how can we judge a good pattern ?

Possible criteria

- Efficiency
- Simplicity / Readability
- Security
- Decentralisation

Patterns

Contract Registry

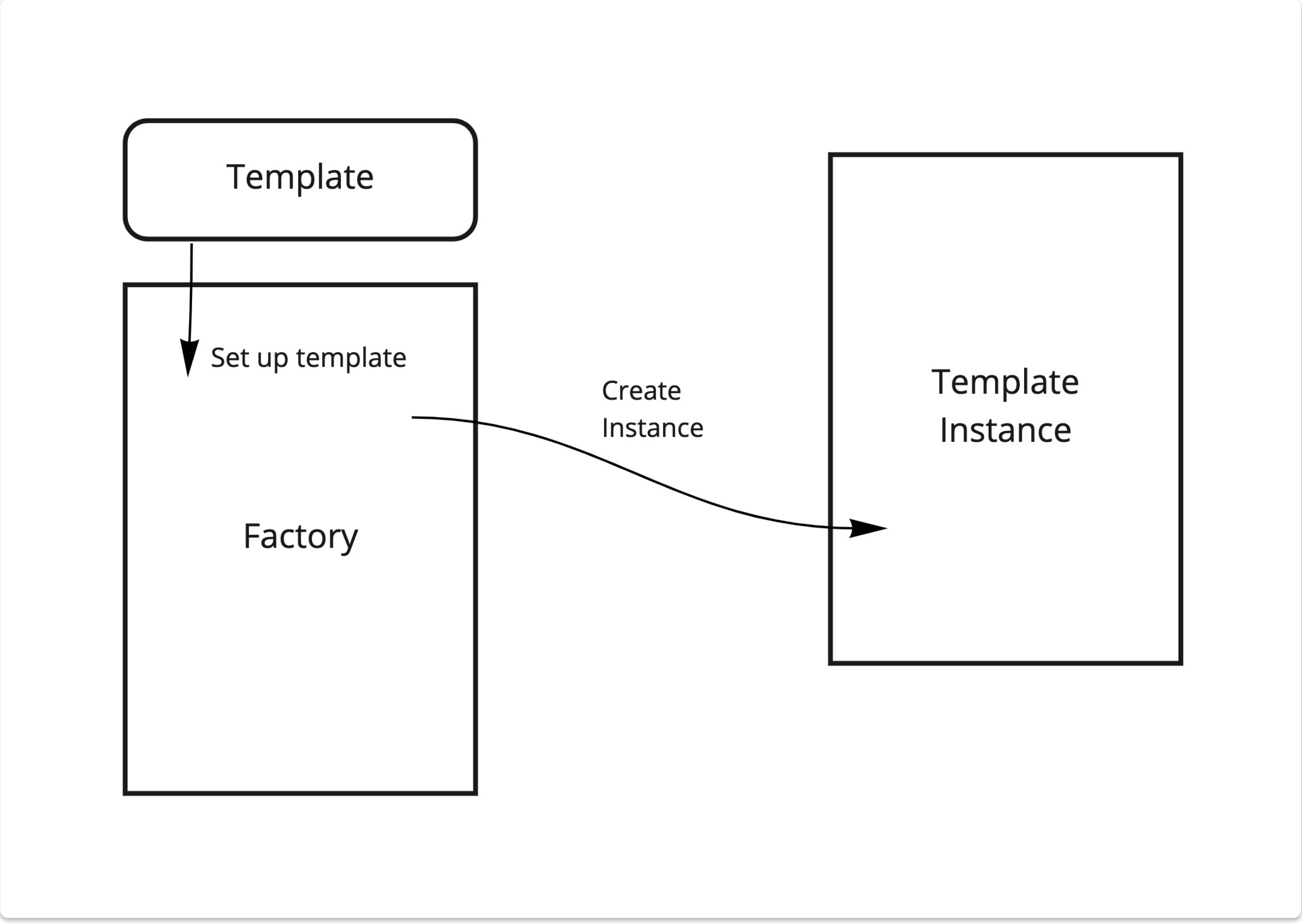
This can be seen as an anti pattern, if it is being used for upgradability, there are other approaches

Data Contract

We will cover this area in the upgradability lesson

Factory Contract

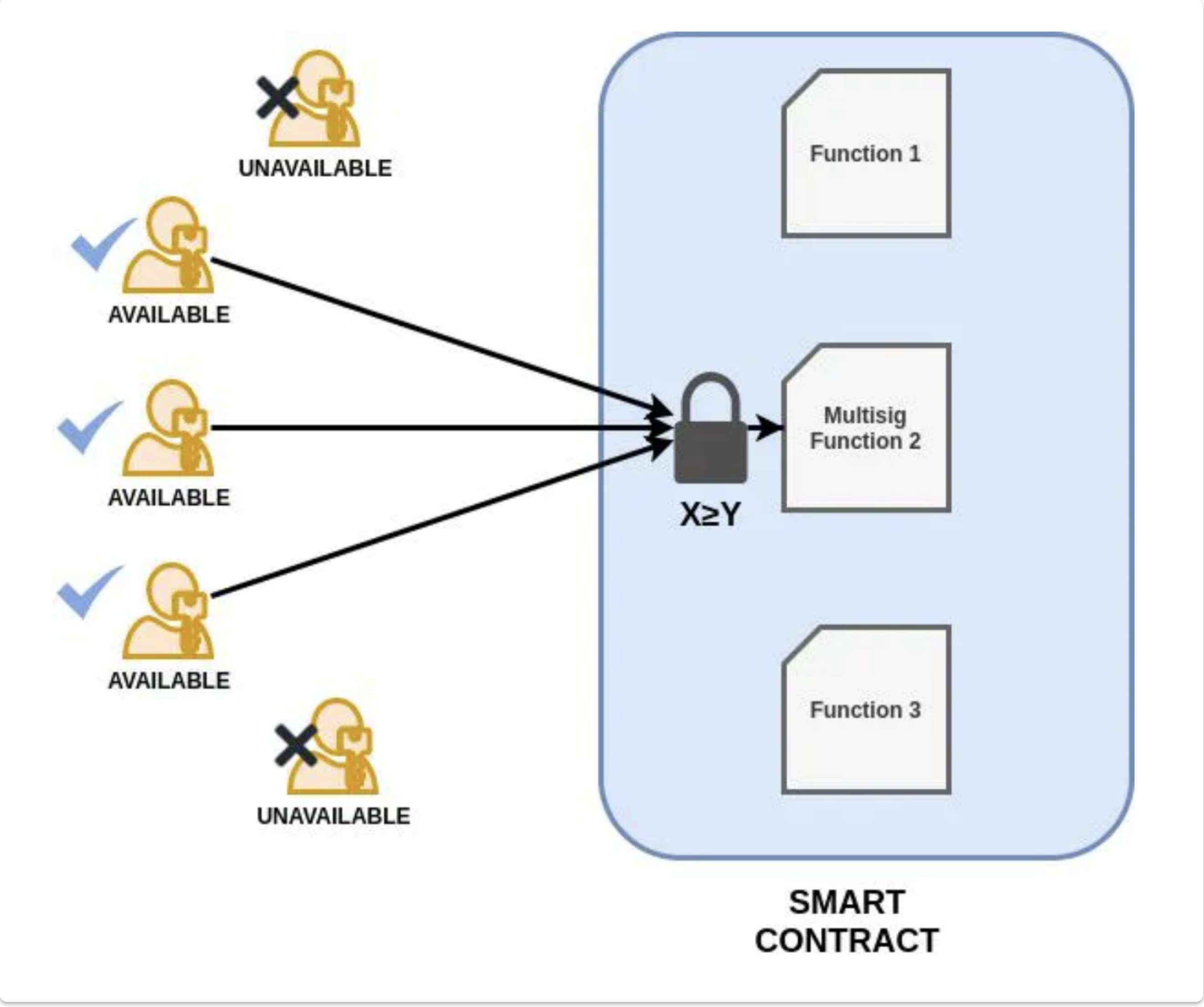
A factory contract is used to produce template contracts at runtime.



incentive execution

Offer other users an incentive for calling for functions
An example is the ethereum alarm clock
For an alternative approach, see Chainlink keepers

Multisig Authorisation



See Gnosis safe for an implementation

State machine

A well known pattern in cs
https://fravoll.github.io/solidity-patterns/state_machine.html

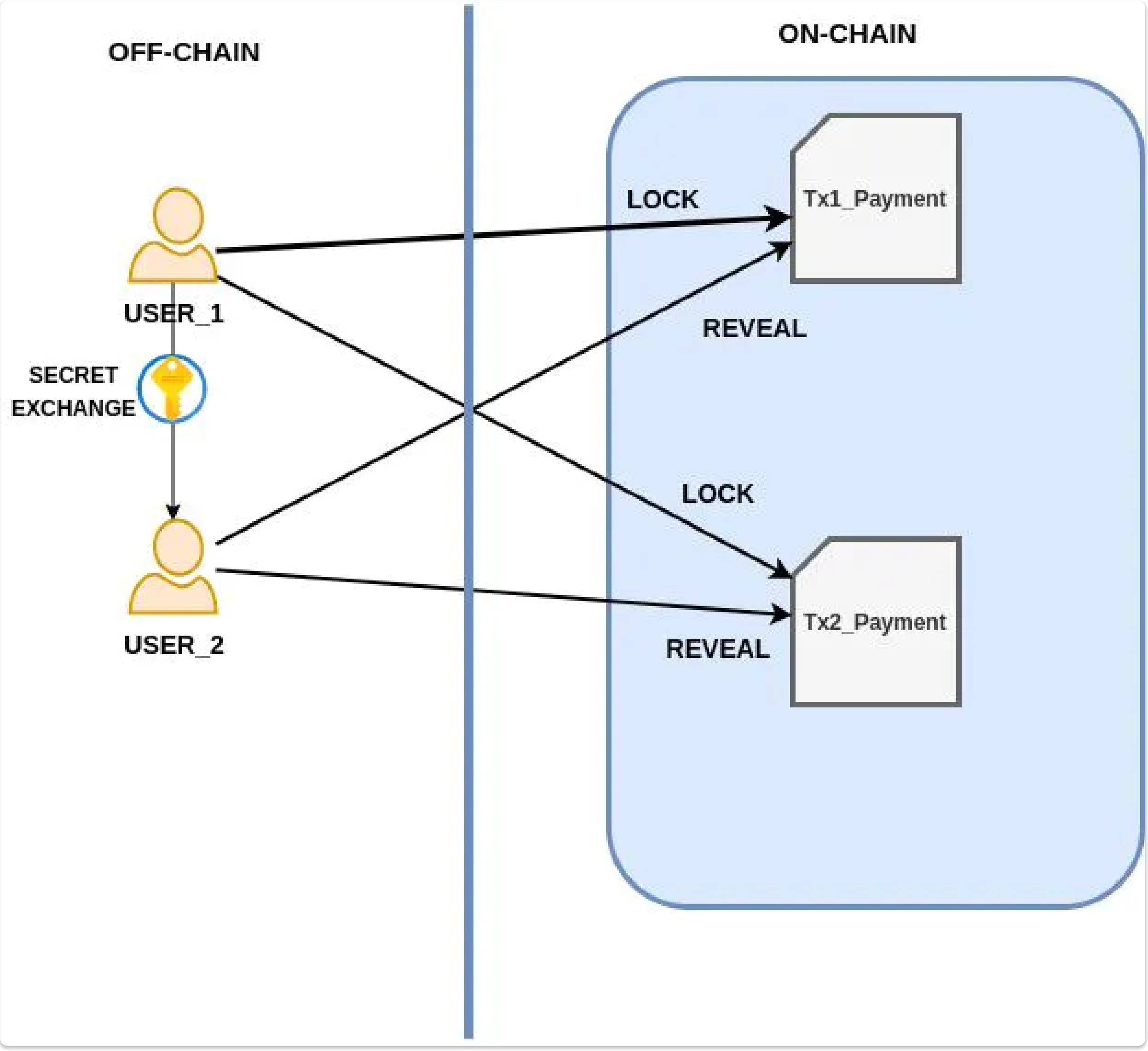
To implement a state machine we need to define

- The states allowed
- The transitions between those states
- Function logic that will vary depending on which state we are currently in, or access to functions may depend on our current state.

Role based access control

An example is Access Control from Open Zeppelin

Off chain authorisation



Circuit Breaker / Escape Hatch

See Open Zeppelin pausable

Checks-Effects-Interactions pattern

See Solidity Docs

First check that the transaction should proceed (is there sufficient allowance ?)
Next change the state in this contract (reduce the allowance)
Finally interact with other contracts (send ether to an address / contract)

Pull payments

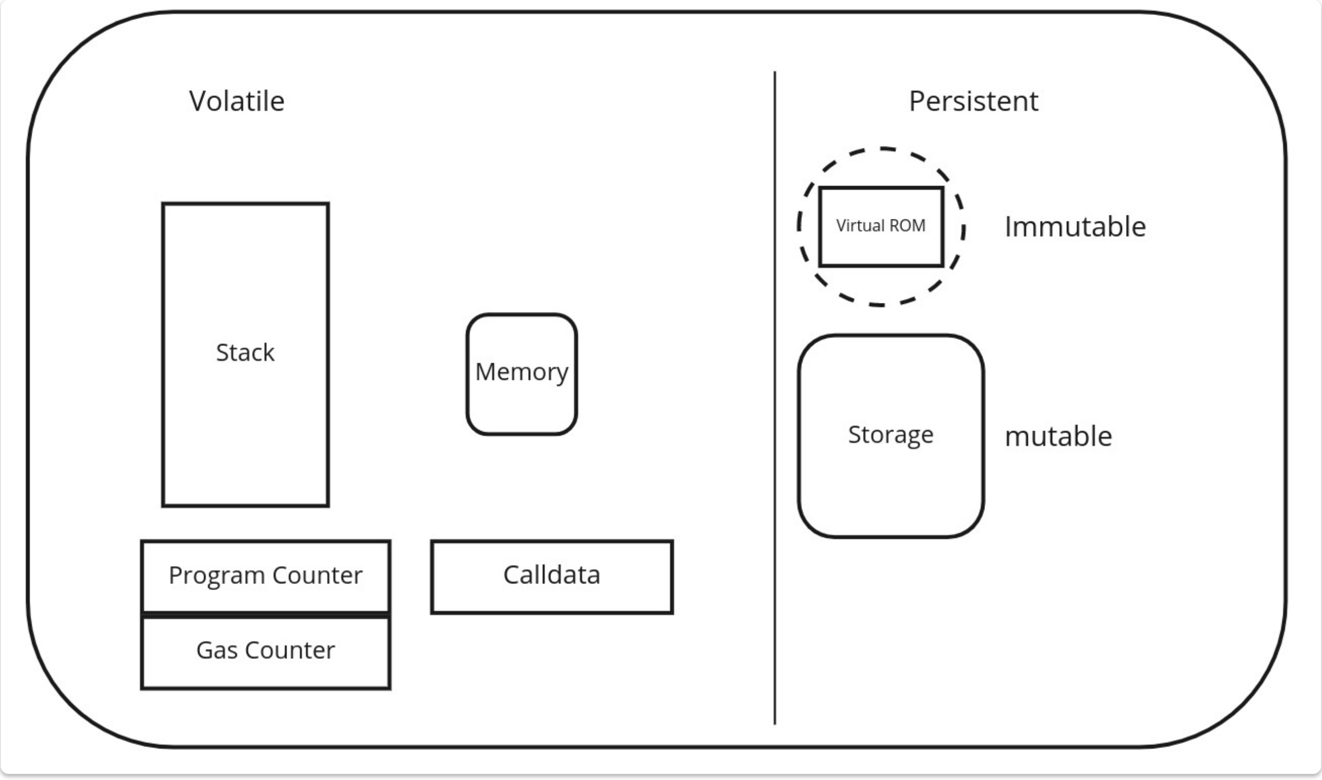
```
js function withdrawFund(address recipient, uint amount) external {
require(recipient != address(0)); require(amount > 0); (bool sent, bytes memory
data) = recipient.call{value: amount}(""); require(sent, "Failed to send ");
emit PaymentMade(recipient, amount); }
```

Oracle Patterns

See https://dev.to/ahmedmansoor012/ethereum-oracle-design-patterns-5api

- publish-subscribe
  broadcast service for frequently changing data
  when data is changed a flag is set / event
  interested (subscribed) parties poll the flag (or listen for events)
- immediate-read
  single lookup of (fairly fixed) data, probably stored in a contract
- request-response
  This is a comprehensive approach used by chainlink involving on and off chain components

EVM Review

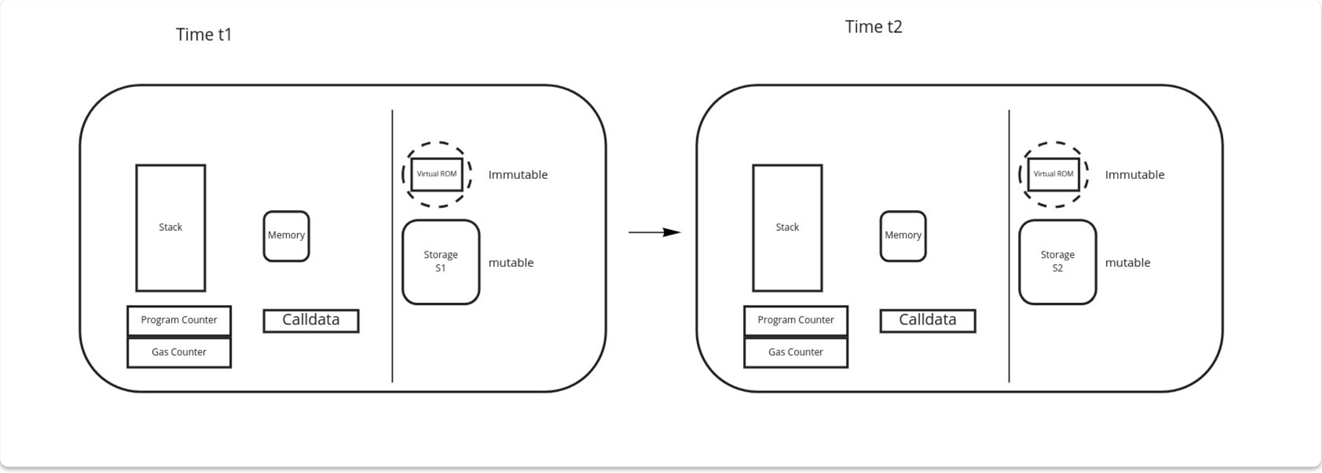


Data areas

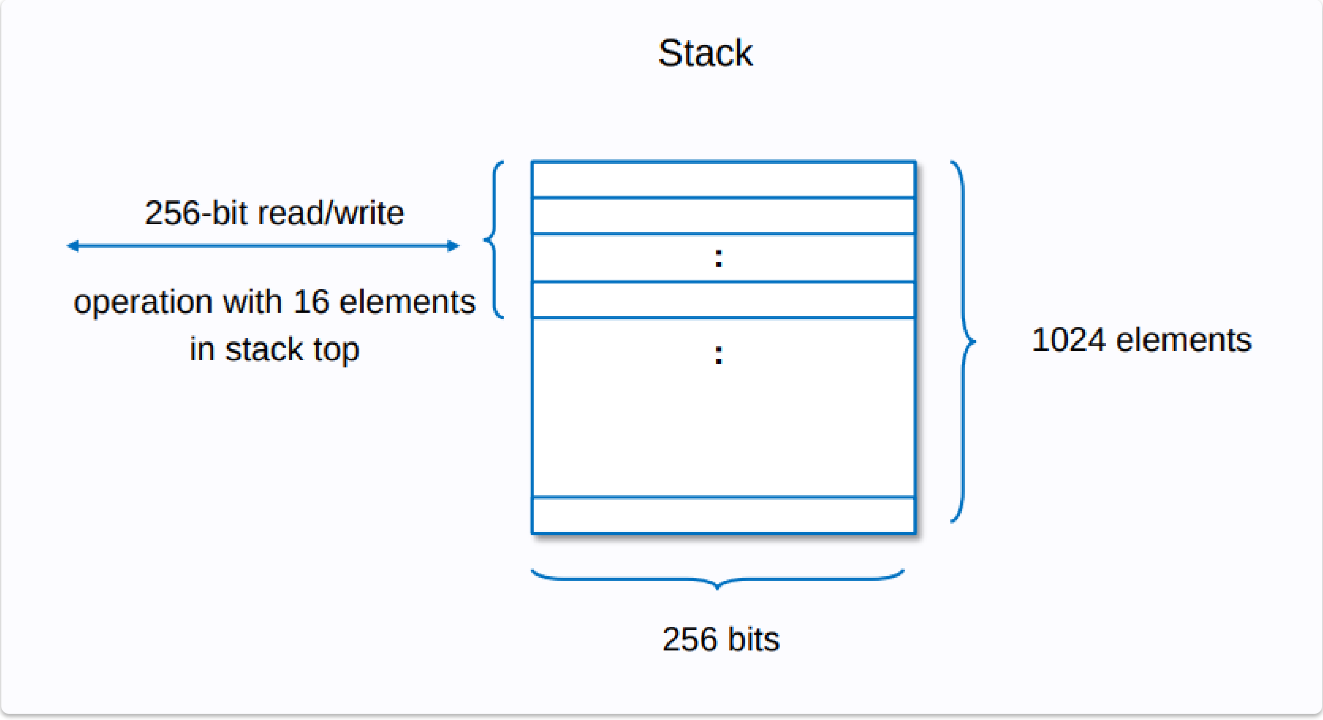
Data can be stored in

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

EVM State transition

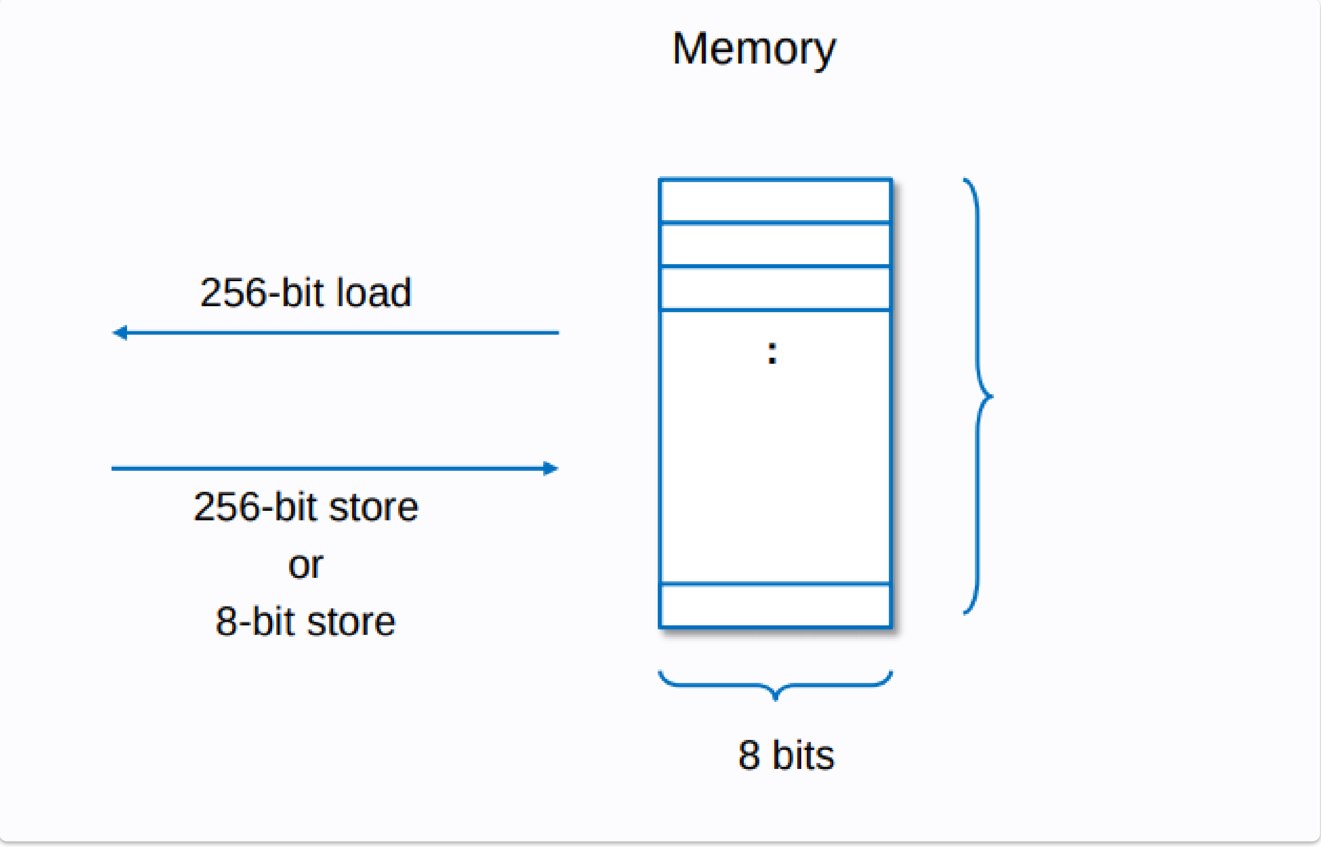


The Stack



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

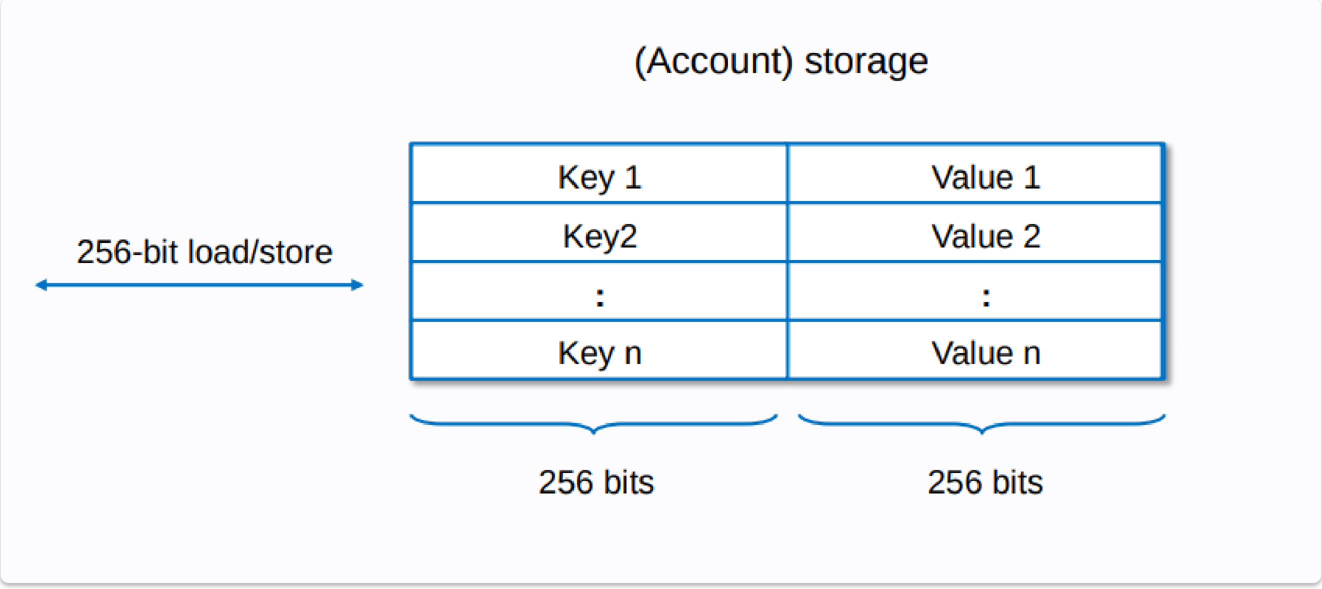
Memory



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.

- MLOAD loads a word from memory into the stack.
- MSTORE saves a word to memory.
- MSTORE8 saves a byte to memory.

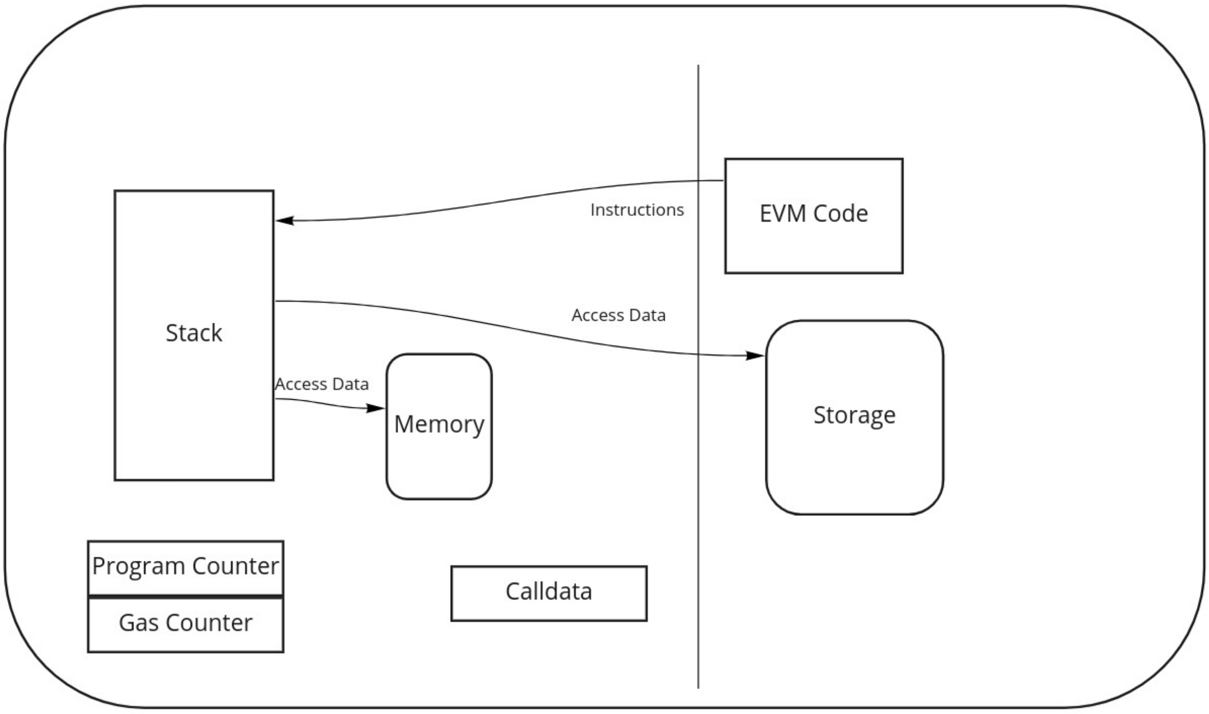
Storage



Key 1	Value 1
Key2	Value 2
:	:
Key n	Value n

See Documentation

Code Execution



OpCodes

- Stack-manipulating opcodes (POP, PUSH, DUP, SWAP)
- Arithmetic/comparison/bitwise opcodes (ADD, SUB, GT, LT, AND, OR)
- Environmental opcodes (CALLER, CALLVALUE, NUMBER)
- Memory-manipulating opcodes (MLOAD, MSTORE, MSTORE8, MSIZE)
- Storage-manipulating opcodes (SLOAD, SSTORE)
- Program counter related opcodes (JUMP, JUMPI, PC, JUMPDEST)
- Halting opcodes (STOP, RETURN, REVERT, INVALID, SELFDESTRUCT)

https://www.ethervm.io/