Introduction to EVM

Martinet Lee

/Outline

- EVM
  - State, Transactions, Block
  - Contract Deployment
  - Execution Model
- Code Immutability

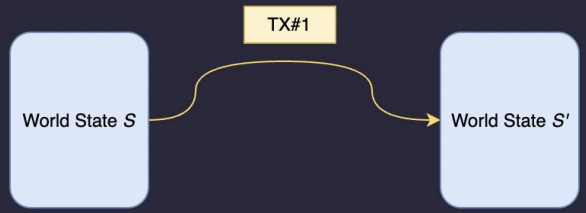
<EVM>

Lots of Illustrations from
https://github.com/takenobu-hs/ethereum-evm-illustrated

/What is EVM

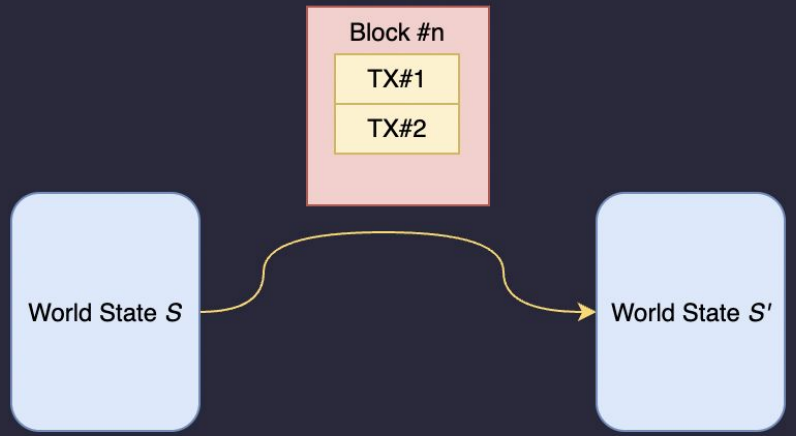
EVM, Ethereum Virtual Machine, is a transaction-based state machine. It defines a state transition function. Given an old state S, and a set of ordered transactions TXS, it will deterministically output a new state S'.

As such, we'll first need to understand the state, the transaction, and how the machine executes the transaction.



/PoV of Blocks

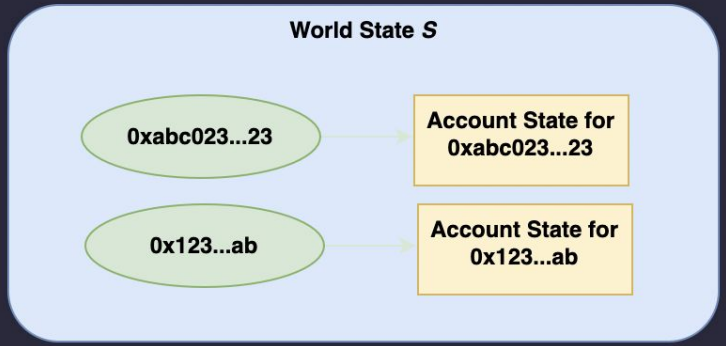
Miners / Proposers pack the transactions together as blocks, deciding the order of transactions, and transition the whole state.



/EVM World State

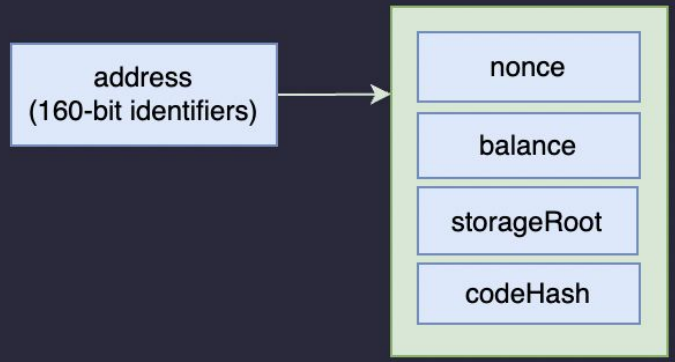
The world state is a mapping of Ethereum addresses to account states. Byte arrays to byte arrays.

The mapping is encoded in a Merkle Patricia Tree.

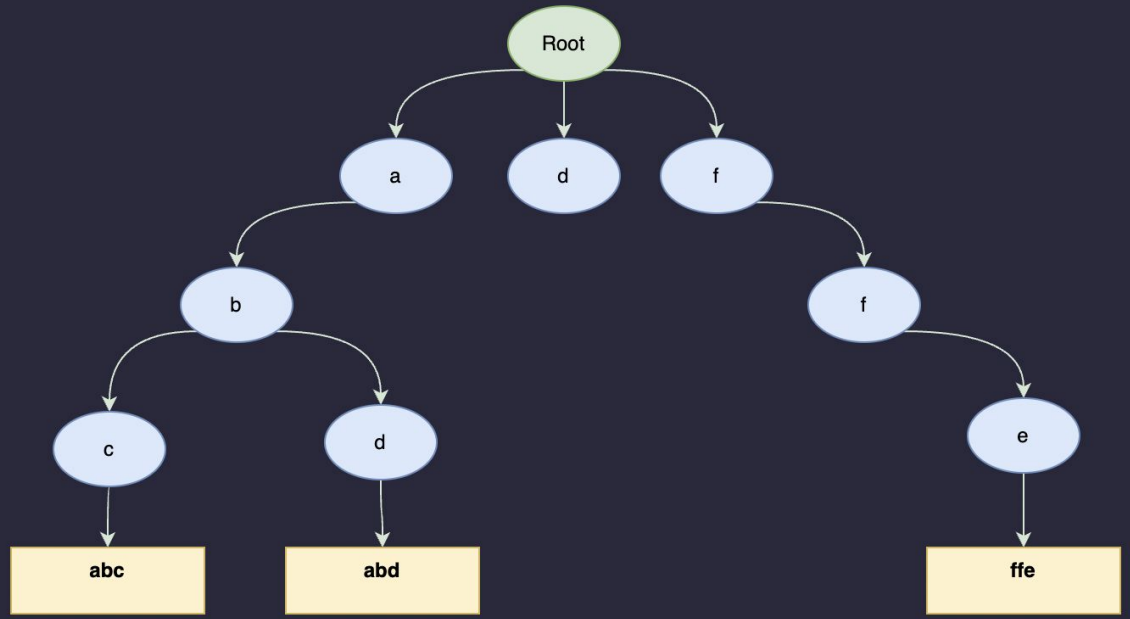


/Account State

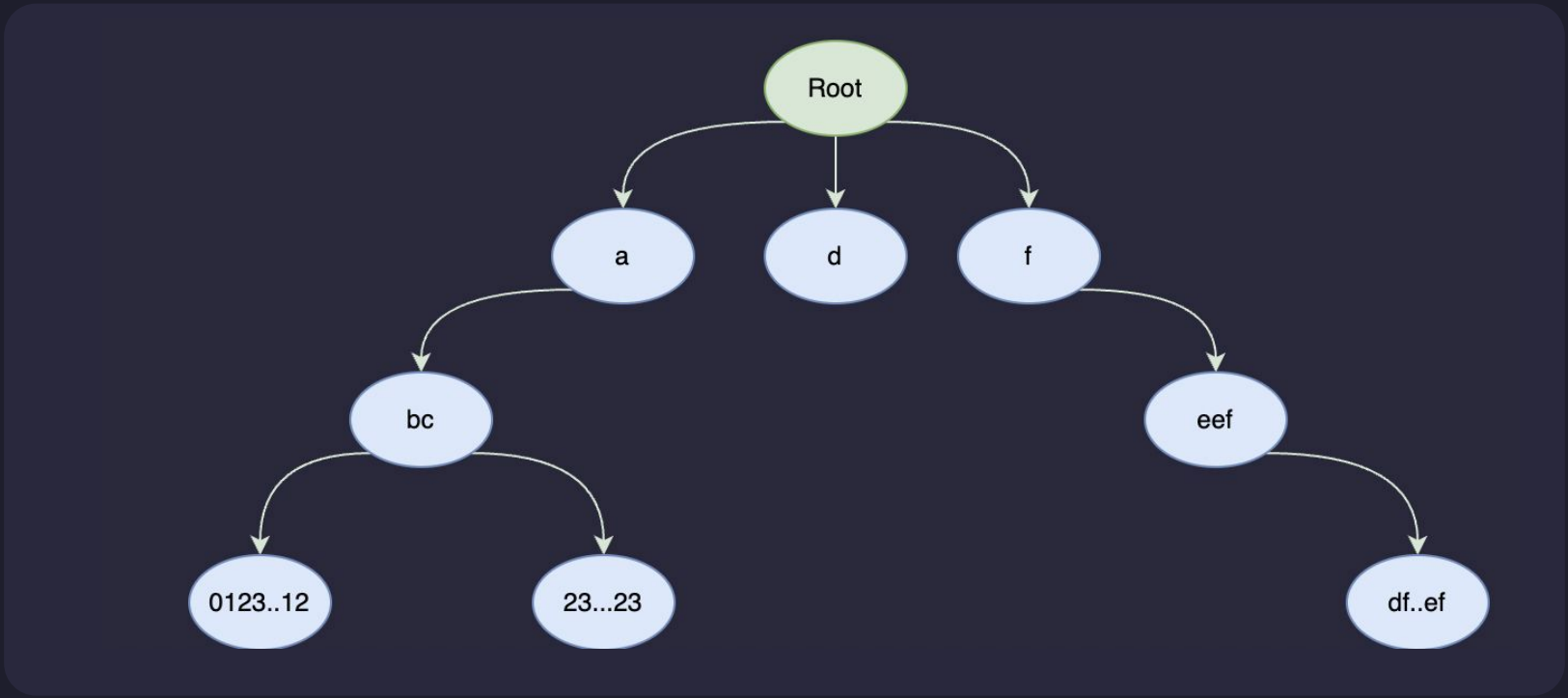
- **nonce**
- **balance**: Ether balance of the account
- **storageRoot**: hash of the root node of a Merkle Patricia Tree that encodes the storage of the account.
- **codeHash**: hash of the EVM code if this address is a smart contract.



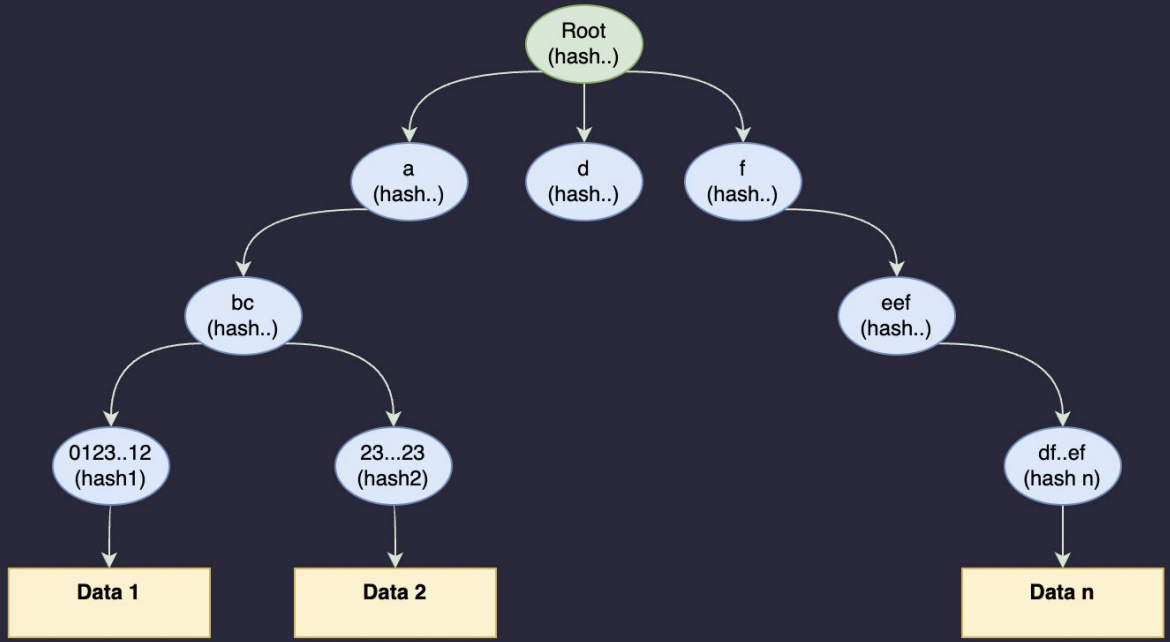
/Trie



/Radix Tree

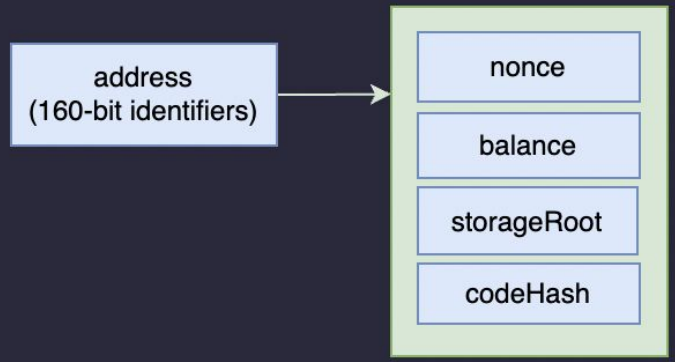


/Merkle Patricia Tree

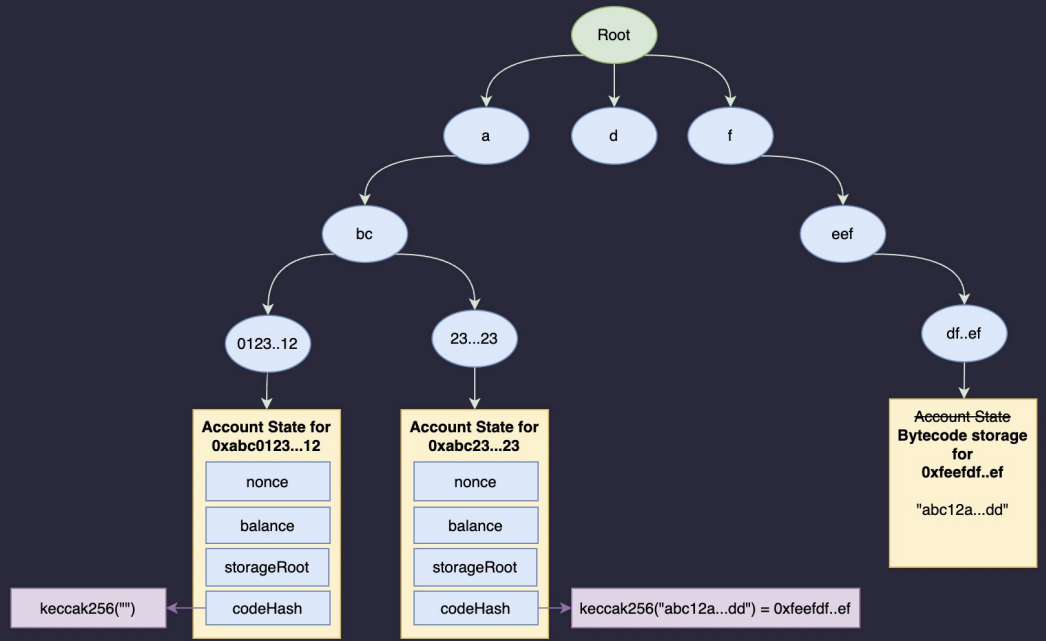


/Account State

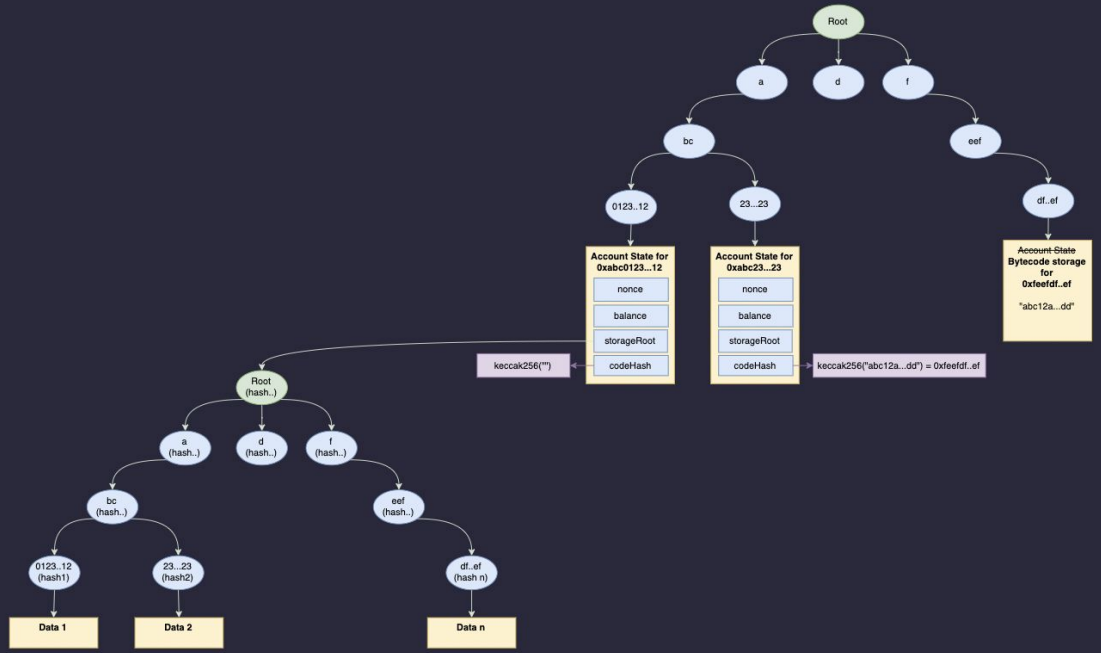
- **codeHash**: hash of the EVM code if this address is a smart contract. Where is the code stored?



/World State


Root
a
d
f
bc
eef
0123..12
23...23
df..ef
Account State for 0xabc0123...12
nonce
balance
storageRoot
codeHash
Account State for 0xabc23...23
nonce
balance
storageRoot
codeHash
~~Account State~~
Bytecode storage for 0xfeefdf..ef
"abc12a...dd"
keccak256("")
keccak256("abc12a...dd") = 0xfeefdf..ef

/World State


Root
a
d
f
bc
eef
0123..12
23...23
df..ef
Account State for 0xabc0123...12
nonce
balance
storageRoot
codeHash
Account State for 0xabc23...23
nonce
balance
storageRoot
codeHash
~~Account State~~ Bytecode storage for 0xfeefdf..ef
"abc12a...dd"
keccak256("")
keccak256("abc12a...dd") = 0xfeefdf..ef
Root (hash..)
a (hash..)
d (hash..)
f (hash..)
bc (hash..)
eef (hash..)
0123..12 (hash1)
23...23 (hash2)
df..ef (hash n)
Data 1
Data 2
Data n

/Discussion

- The benefits of Merkle Patricia Tree in World state?
- Bytecode storage .. clashing?

/Transactions in EVM

- A transaction represents the intention of the external party:
  - is it interacting with other accounts or creating a contract?
  - how much operation cost does the external party willing to pay for the transaction.
- A transaction has to be signed by the external party.

/Transactions, more formally

Common fields

- type: legacy or eip-2930
- nonce
- gasPrice
- gasLimit
- to
- value
- r,s: signatures

Legacy fields

- w: (chainId and yParity) (EIP-155)

EIP-2930 fields

- accessList
- chainId
- yParity

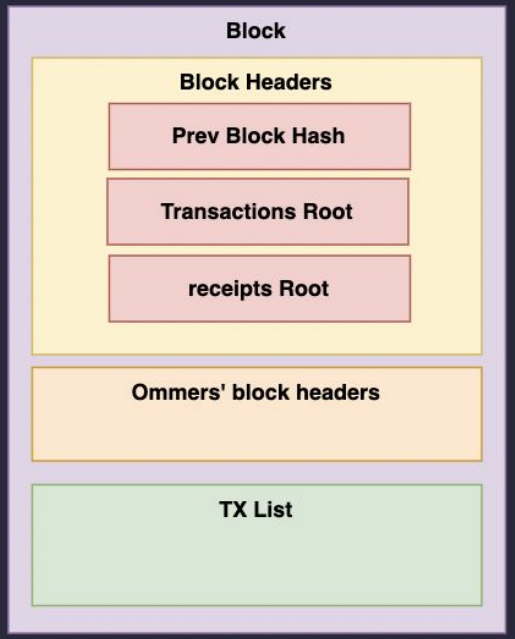
Fields for Contract Creation

- init

Fields for Message Call

- data

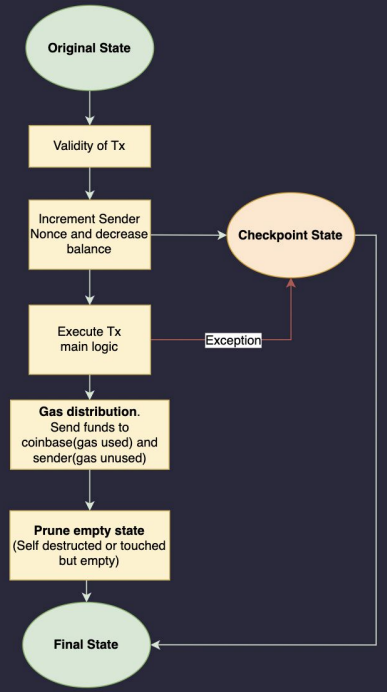
/Block



Block Headers

- incorporates information needed by the consensus.
- incorporates commitment of the world state and the transactions included in the block.

/Transaction Execution Stages



- Note that any exception during the execution goes back to the checkpoint state ⇒ Either the transaction is executed, or not at all.

/Validity of Transaction

The transaction itself is valid

1. The transaction is properly formatted
2. The transaction is properly signed
3. Gas Limit has to be greater than intrinsic gas.

The transaction is valid against the current state

1. The transaction nonce matches the sender account’s nonce
2. The sender account has no contract deployed (EIP-3607)
3. The sender account balance contains at least the cost that is required as an up-front payment.

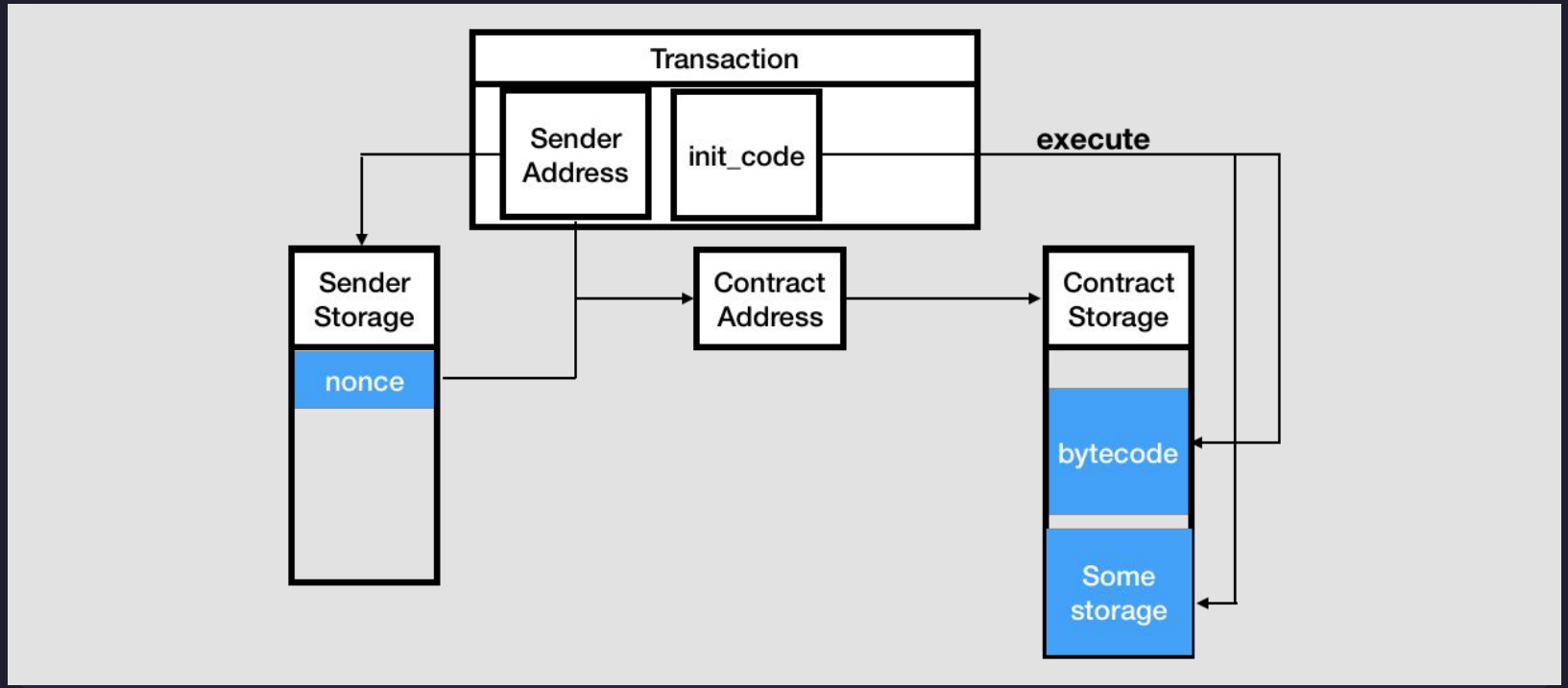
/Intrinsic Gas of a transaction

- Upfront cost of a transaction.
- Tx Basic Gas + Contract Creation Basic Gas // Basic Tax
  + Tx Data Gas // per Data Tax
  + Tx Warming Address + Tx Warming Storage

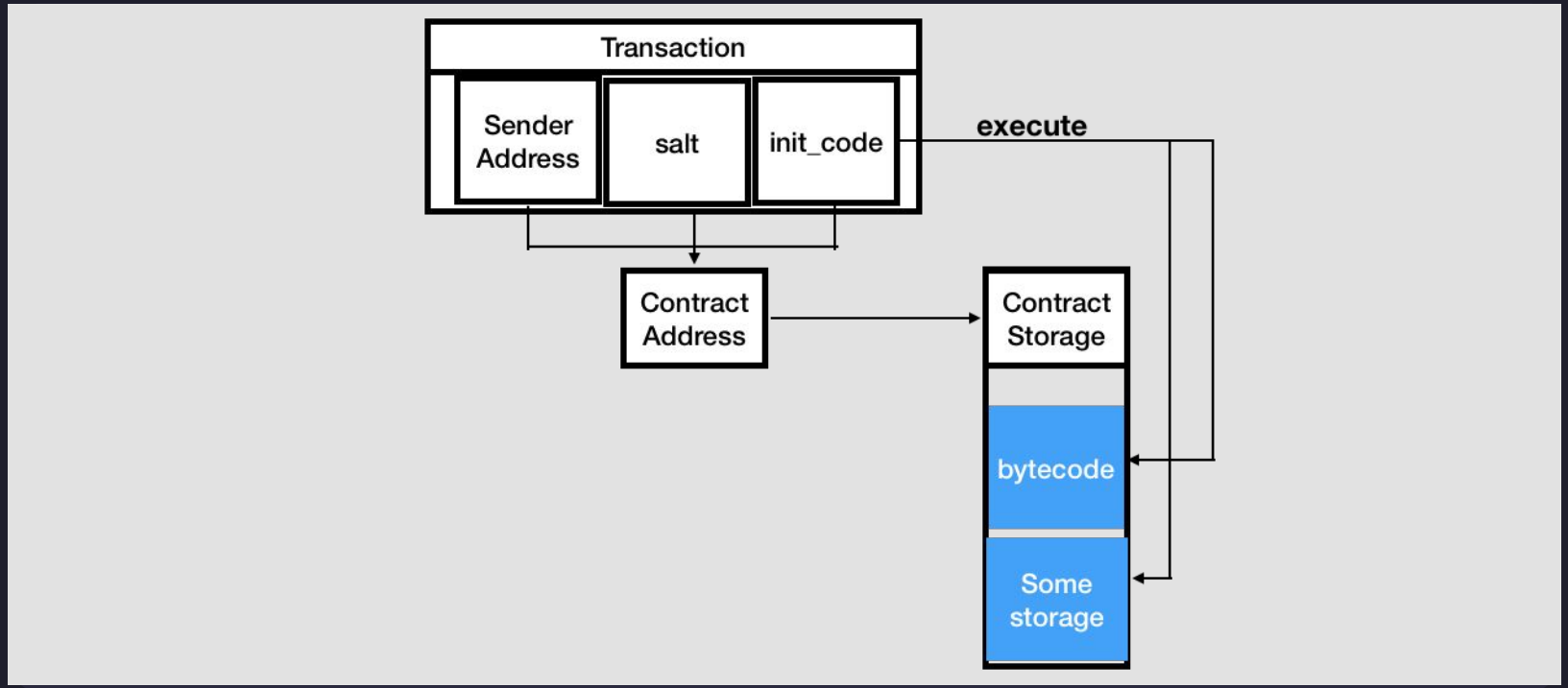
/Processing Contract Creation

- Calculate Account Address
  - Create1 (nonce, sender account)
  - Create2 (salt, initCode, sender account)
- EVM executes the initialization code, initialization code returns the deployed code.
- A code deposit cost is being paid according to the size of the code.

/Create1


Transaction
Sender Address
init_code
execute
Sender Storage
nonce
Contract Address
Contract Storage
bytecode
Some storage

/Create2


Transaction
Sender Address
salt
init_code
execute
Contract Address
Contract Storage
bytecode
Some storage

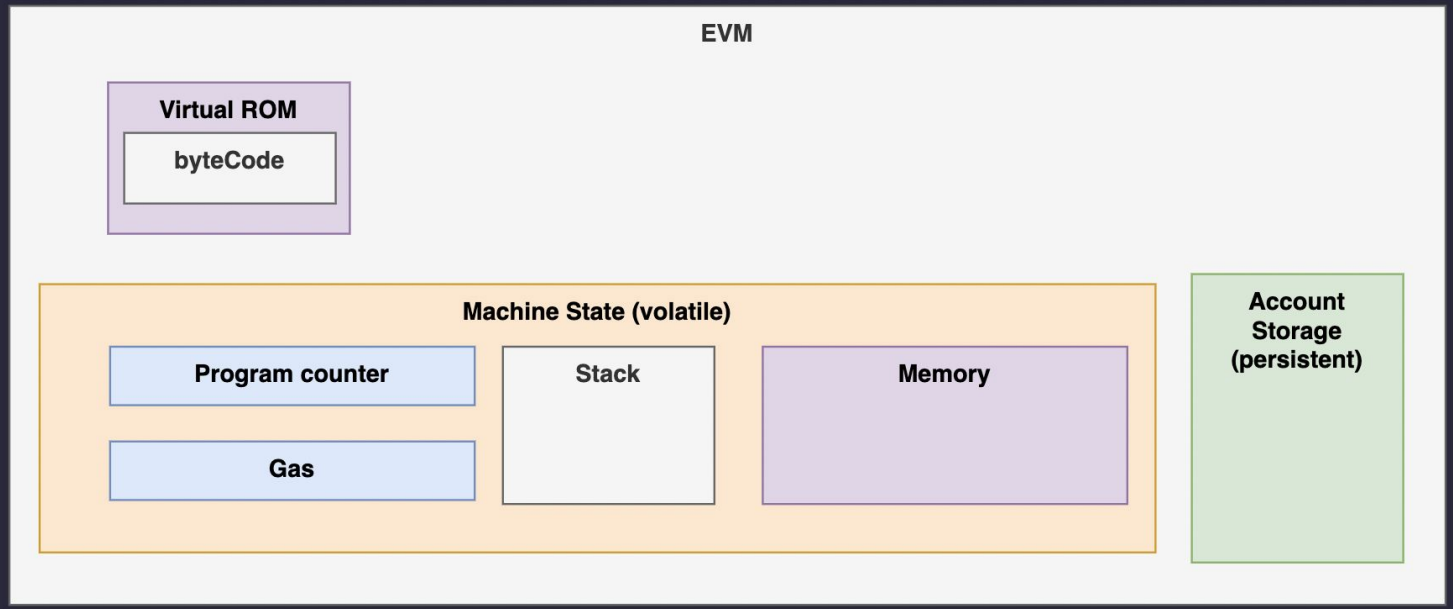
/Important Properties for Create2

- Contract address can be predetermined before deployment
  - To get the same address, `init_code` cannot change
- If a contract is self-destructed, it can be re-deployed again at the same address
  - However, the same `init_code` can produce a different runtime bytecode!
- A contract can be redeployed **at the same address** but end up having a **different runtime bytecode!**

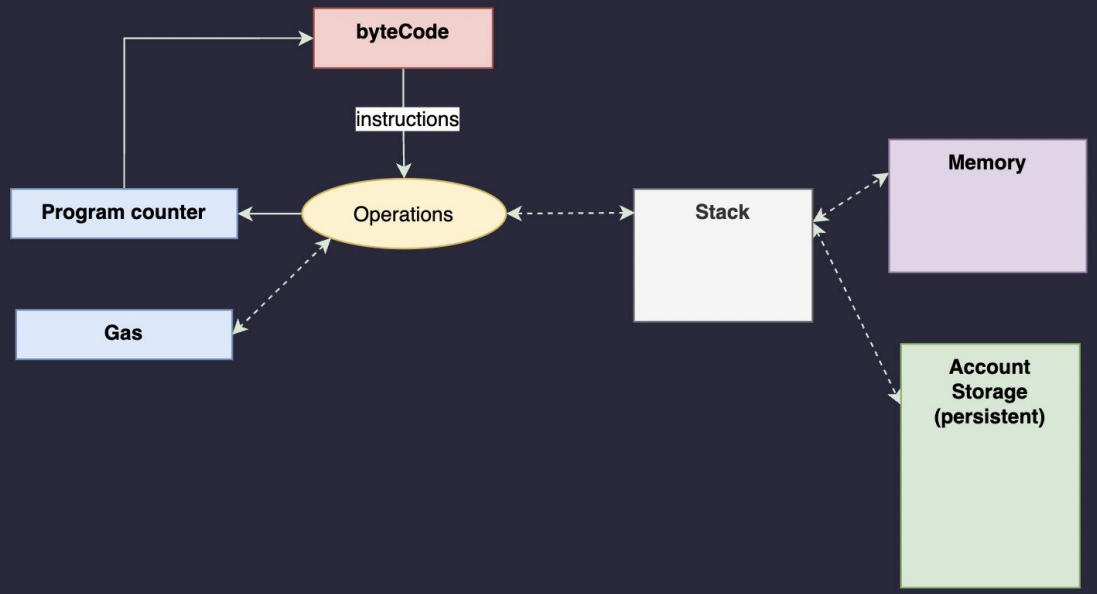
/Processing Message Calls

- 4-byte signatures are defined as the first four bytes of the Keccak hash (SHA3) of the canonical representation of the function signature.

/EVM Architecture



/EVM Architecture



/EVM Bytecode

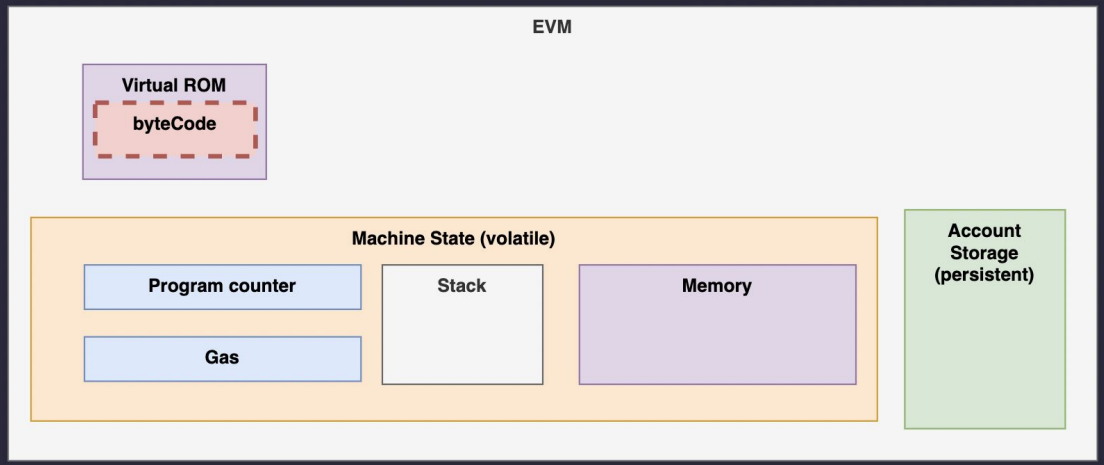
```
6042600052602060 00F3
```

```
PUSH1 0×42
PUSH1 0
MSTORE
PUSH1 32
PUSH1 0
RETURN
```

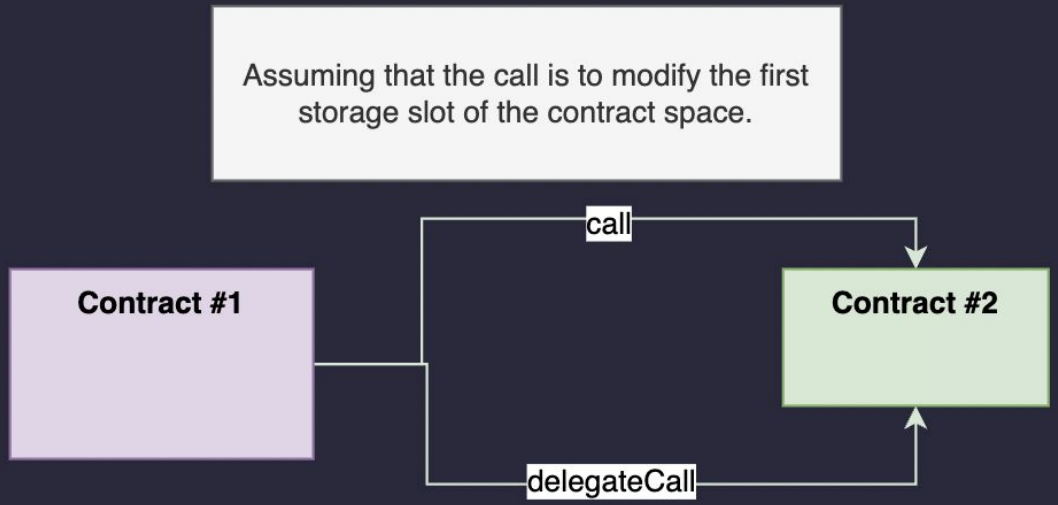
https://www.evm.codes/playground?fork=merge

/DelegateCall

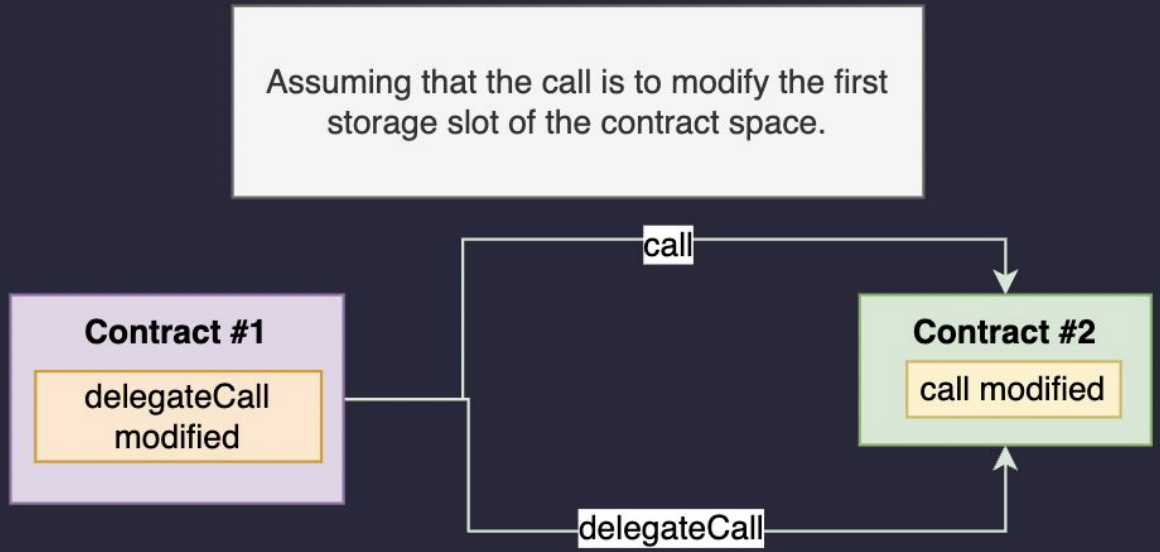
Message-call into this account with an alternative account's code, but persisting the current values for sender and value.



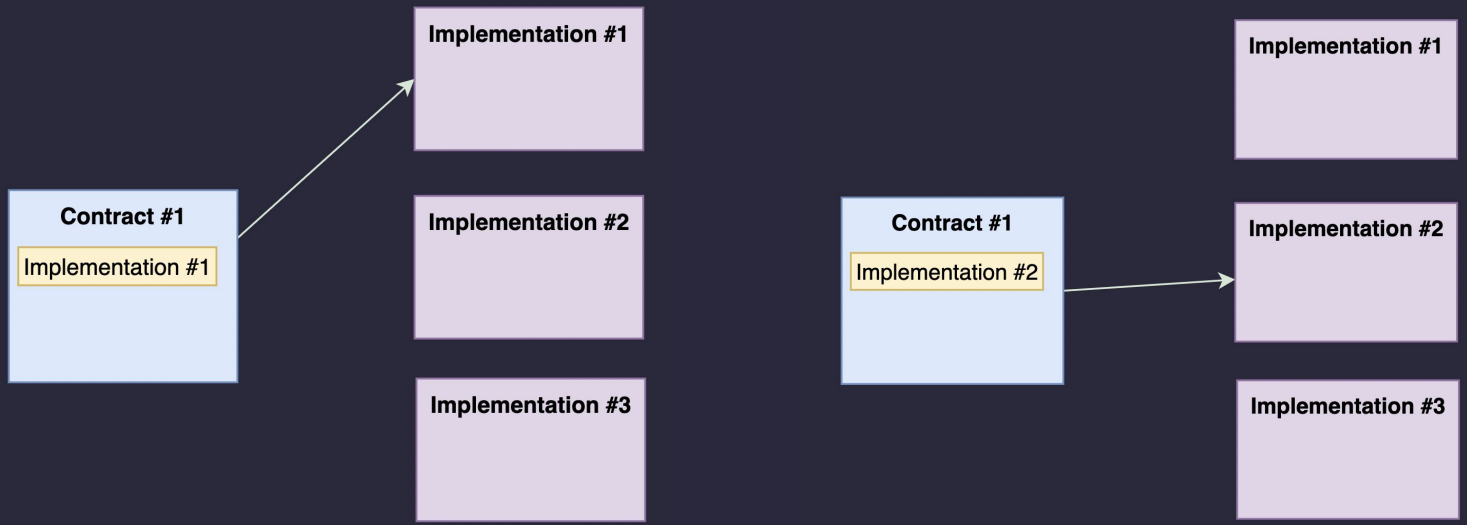
/DelegateCall in practice



/DelegateCall in practice



/Proxy Pattern



/Proxy

```
// This code has not been professionally audited, therefore I cannot make any promises about
// safety or correctness. Use at own risk.
contract Proxy {

    address delegate;
    address owner = msg.sender;

    function upgradeDelegate(address newDelegateAddress) public {
        require(msg.sender == owner);
        delegate = newDelegateAddress;
    }

    function() external payable {
        assembly {
            let _target := sload(0)
            calldatacopy(0x0, 0x0, calldatasize)
            let result := delegatecall(gas, _target, 0x0, calldatasize, 0x0, 0)
            returndatacopy(0x0, 0x0, returndatasize)
            switch result case 0 {revert(0, 0)} default {return (0, returndatasize)}
        }
    }
}
```

/Security Issue around Proxy pattern

- Clash of function signatures
- Clash of storage

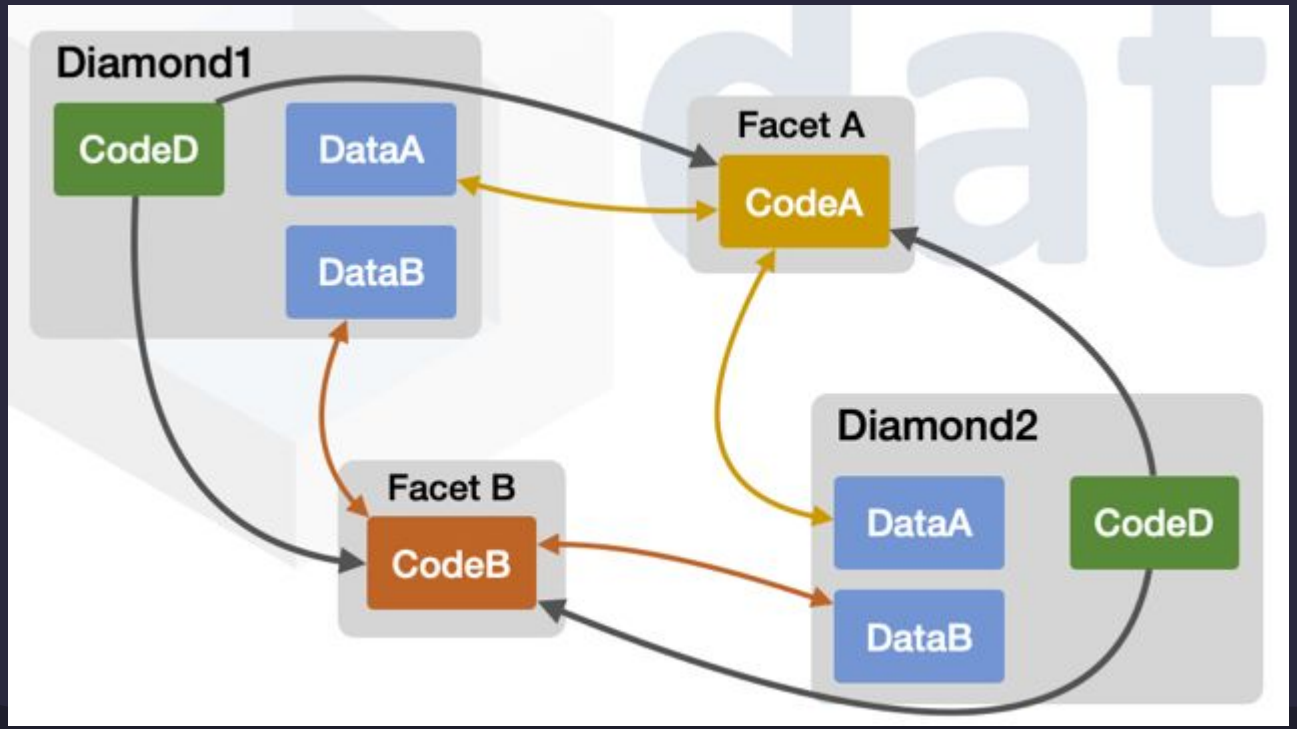
/EIP-170 Contract Code Size Limit

24kB

when a contract is called, even though the call takes a constant amount of gas, the call can trigger O(n) cost in terms of reading the code from disk, preprocessing the code for VM execution, and also adding O(n) data to the Merkle proof for the block’s proof-of-validity

https://eips.ethereum.org/EIPS/eip-170

/Diamond Standard



/THAT'S A WRAP!  (mic drop)