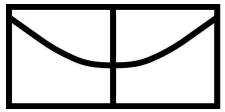


MIT Digital Currency Initiative and the University of Brasilia presents

Cryptocurrency Design and Engineering

Lectures 14 and 15: Programmability in Ethereum
Taught by: James Prestwich
Date: 10/28/2025
MAS.S62

What is the Etheruem Virtual Machine?

1. Goals
    a. Generalize blockchain architecture
    b. Simulate arbitrary state-transition functions
    c. Brief historical screed about application-specific chains

What is the Etheruem Virtual Machine?

1. Goals
    a. Generalize blockchain architecture
    b. Simulate arbitrary state-transition functions
    c. [brief historical screed about application-specific chains]
2. Instantiation
    a. Stack-based
    b. 8-bit opcodes
    c. 32-byte words
    d. The State.

EVM Theory

This is a broad overview. It is intended to communicate the core structures and concepts. It is not intended to be exhaustive.

These slides were made in October 2025, and were reasonably correct at the time. EVM particulars change regularly.

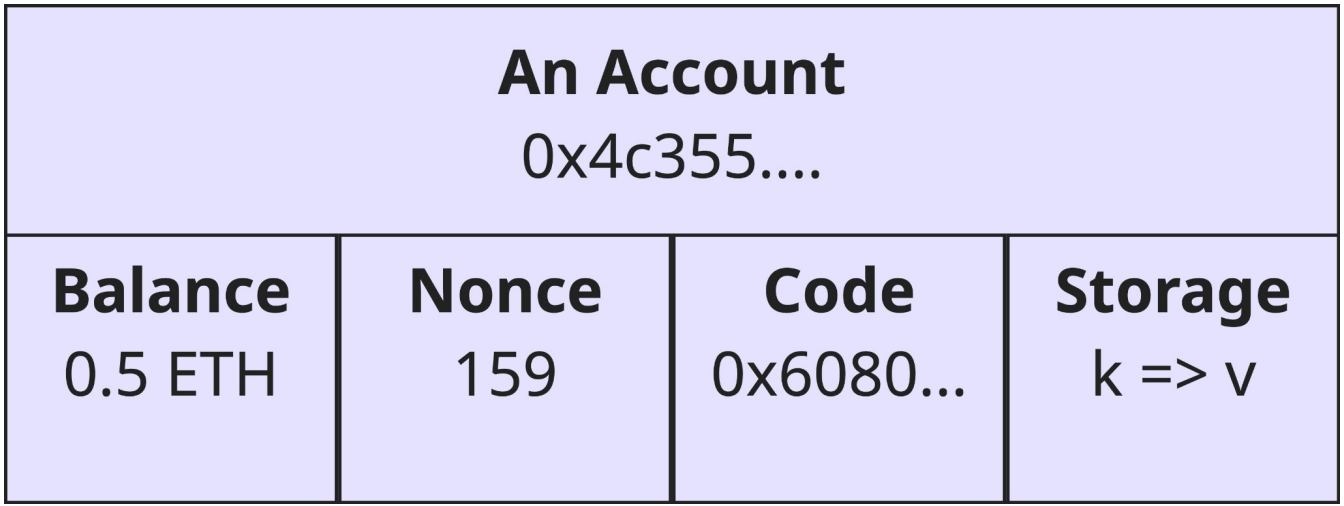
Unless otherwise credited, I made the images and diagrams myself.

EVM Theory

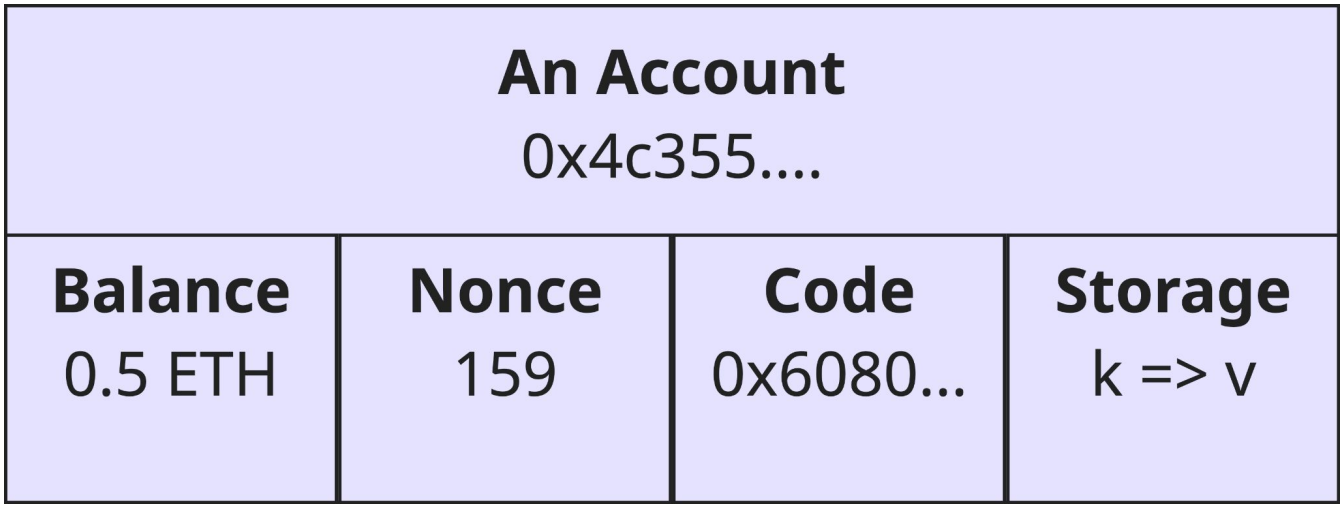
- The State
- Execution Basics
- Some Historical Changes

The State

The State


An Account
0x4c355....
Balance
0.5 ETH
Nonce
159
Code
0x6080...
Storage
k => v

The State


An Account
0x4c355....
Balance
0.5 ETH
Nonce
159
Code
0x6080...
Storage
k => v

The State: Account Types

An "Externally-Owned" Account 0x4c355....			
Balance 0.5 ETH	**Nonce** 159	**Code**	**Storage** k => v

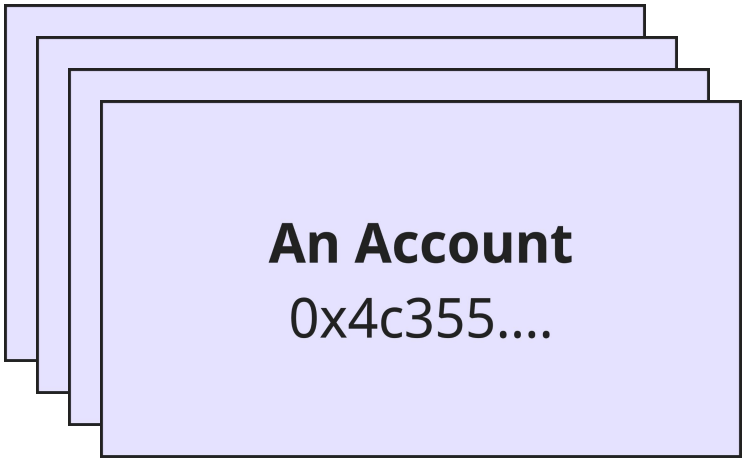
The State: Account Types

A "Smart Contract" Account 0x4c355....			
Balance 0.5 ETH	**Nonce** 159	**Code** 0x6080...	**Storage** k => v

The State: Account Types

A "Delegated" Account 0x4c355....			
Balance 0.5 ETH	**Nonce** 159	**Code** 0xef0100..	**Storage** k => v

The State



x 350,000,000(ish)

EVM Theory

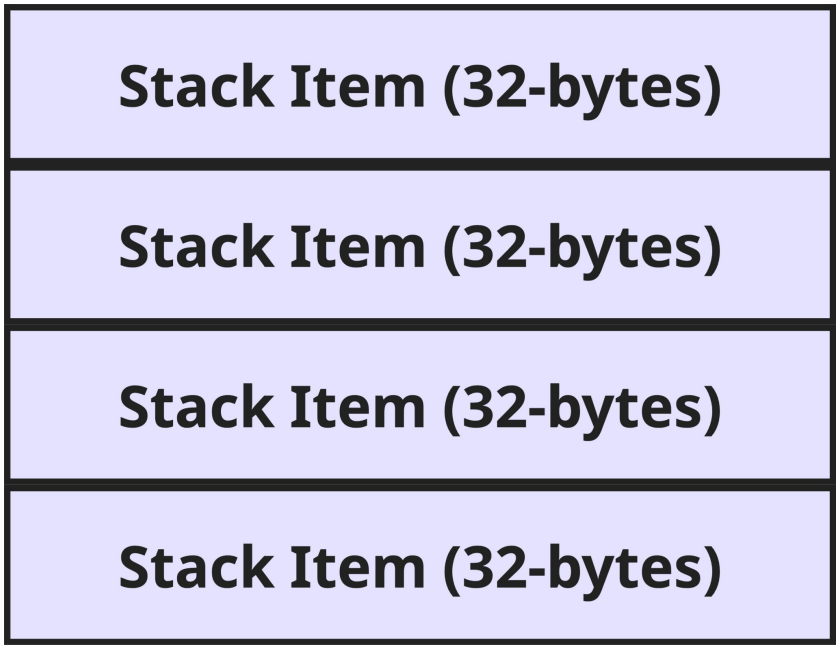
- ~~The State~~

- Execution Basics

- Some Historical Changes

Execution basics

- The Ethereum Virtual Machine
- CallFrames
- Messages
- The CallStack



Execution basics: The EVM

Check out http://evm.codes :)

Execution basics: The EVM

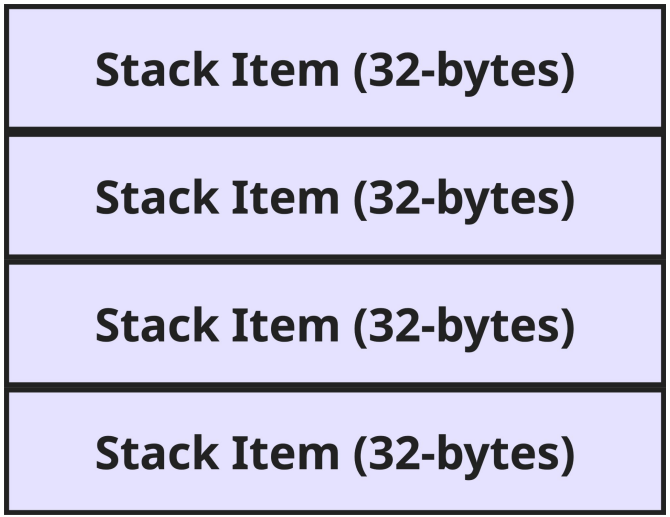
So what's gas?

- Metered execution. Ensure execution halts.
- C.f. SigOps in bitcoin

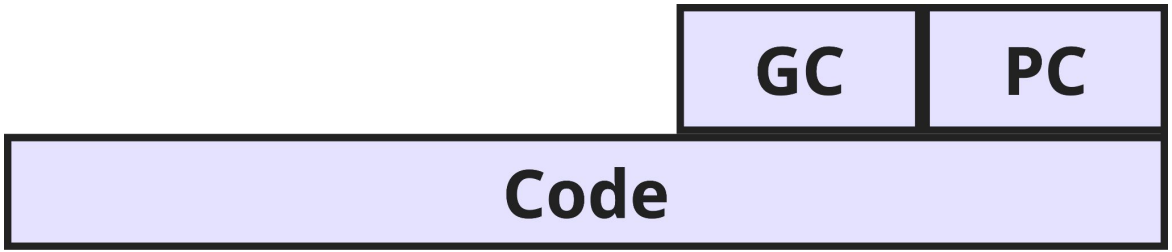
- Opcodes are priced on observed behavior / vibes

Check out http://evm.codes :)

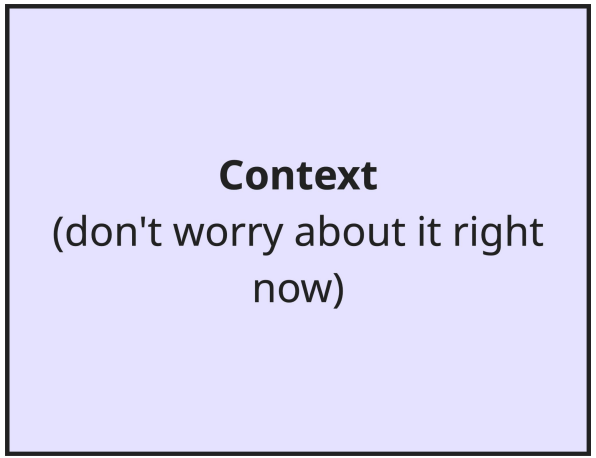
Execution basics: The EVM



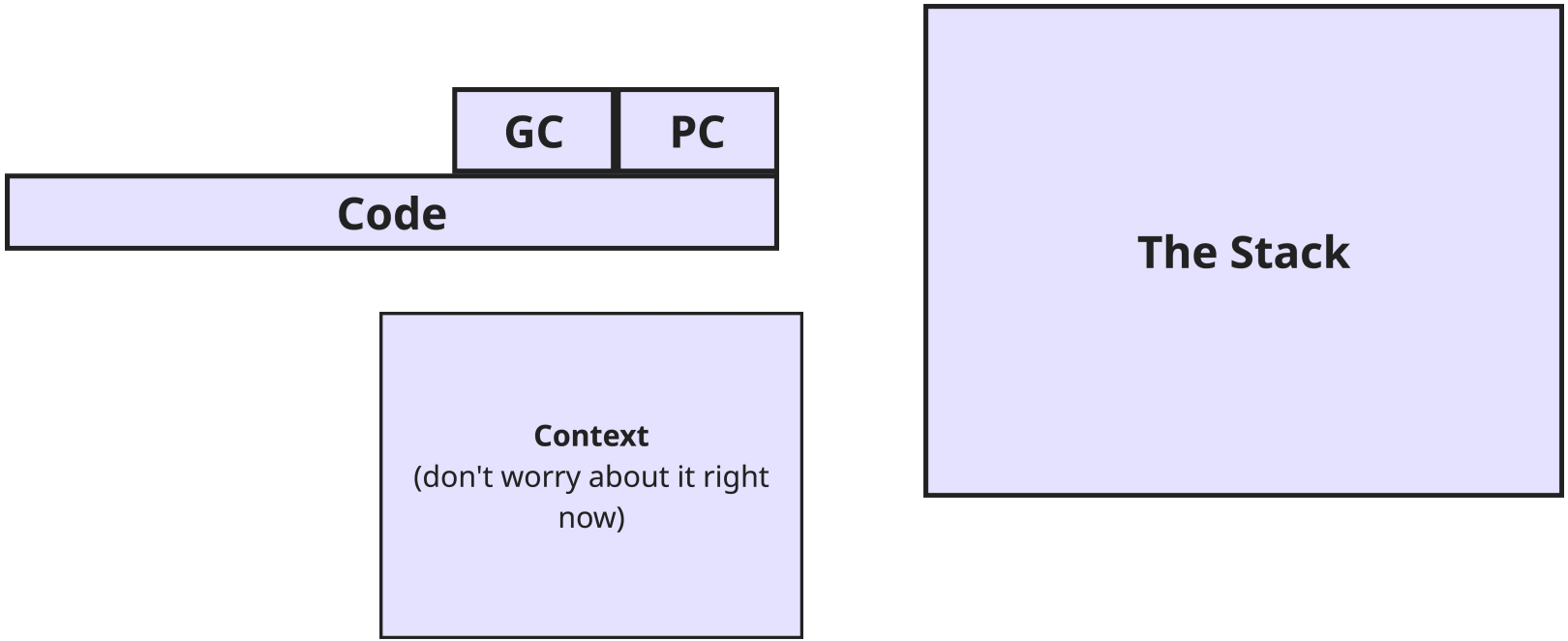
Execution basics: The EVM



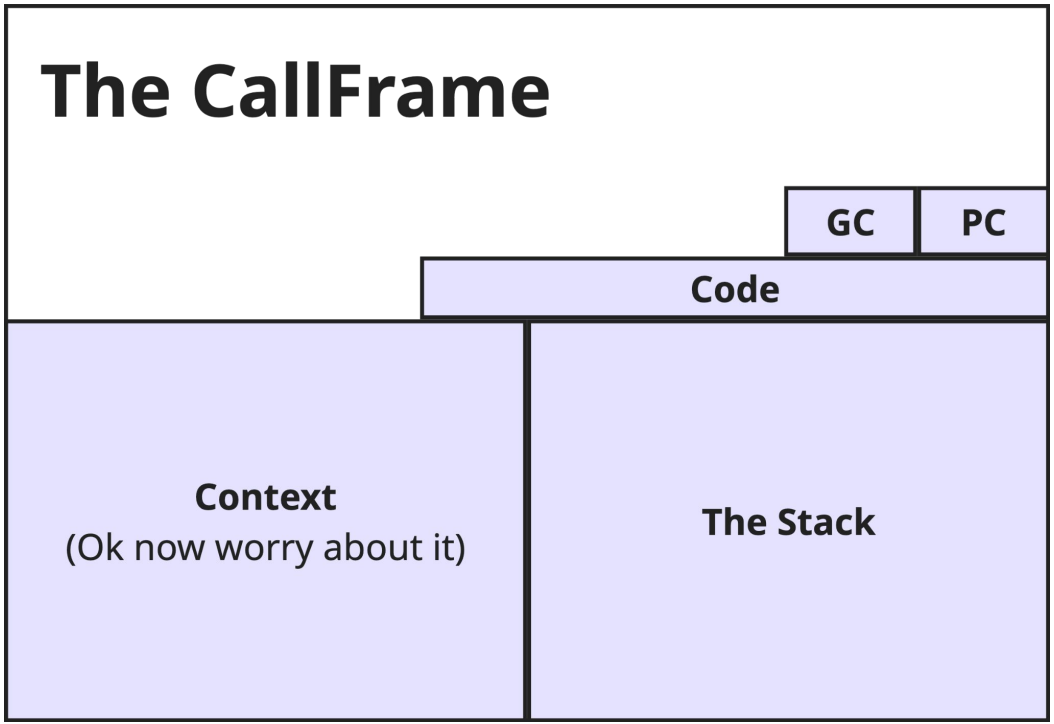
Execution basics: The EVM



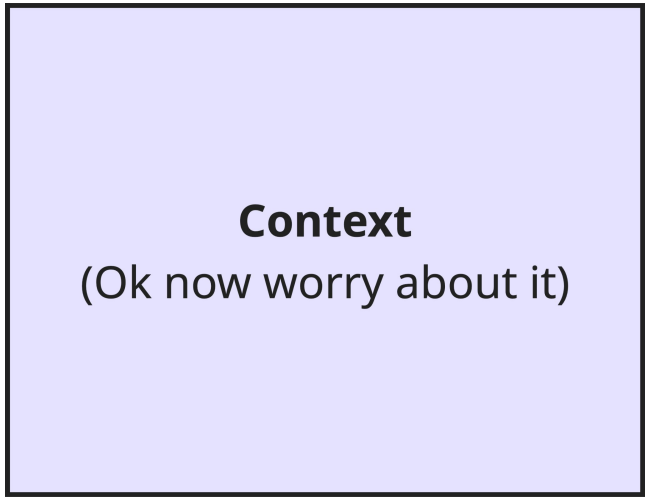
Execution basics: The EVM



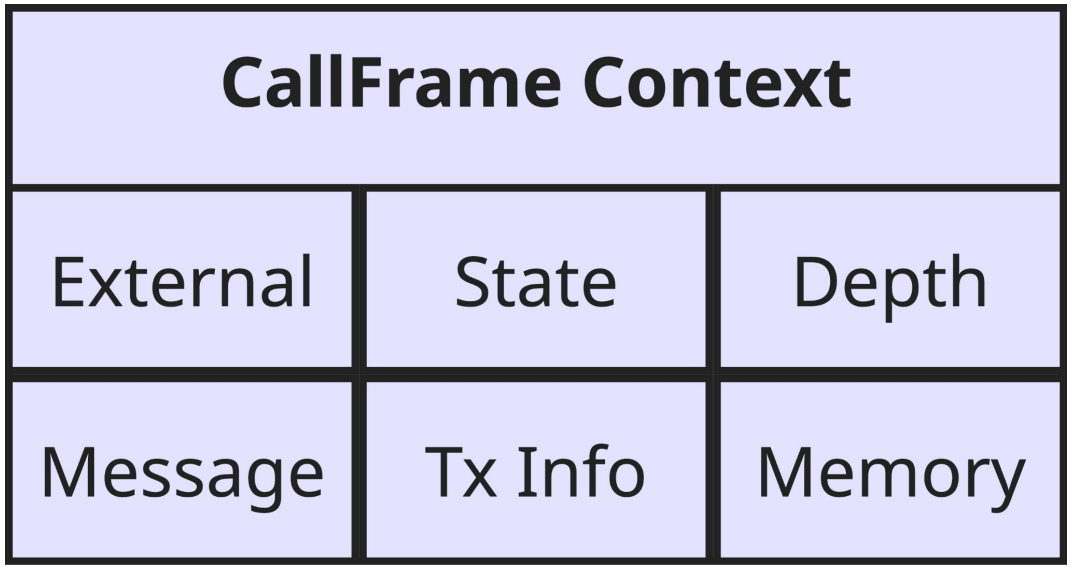
Execution basics: CallFrames



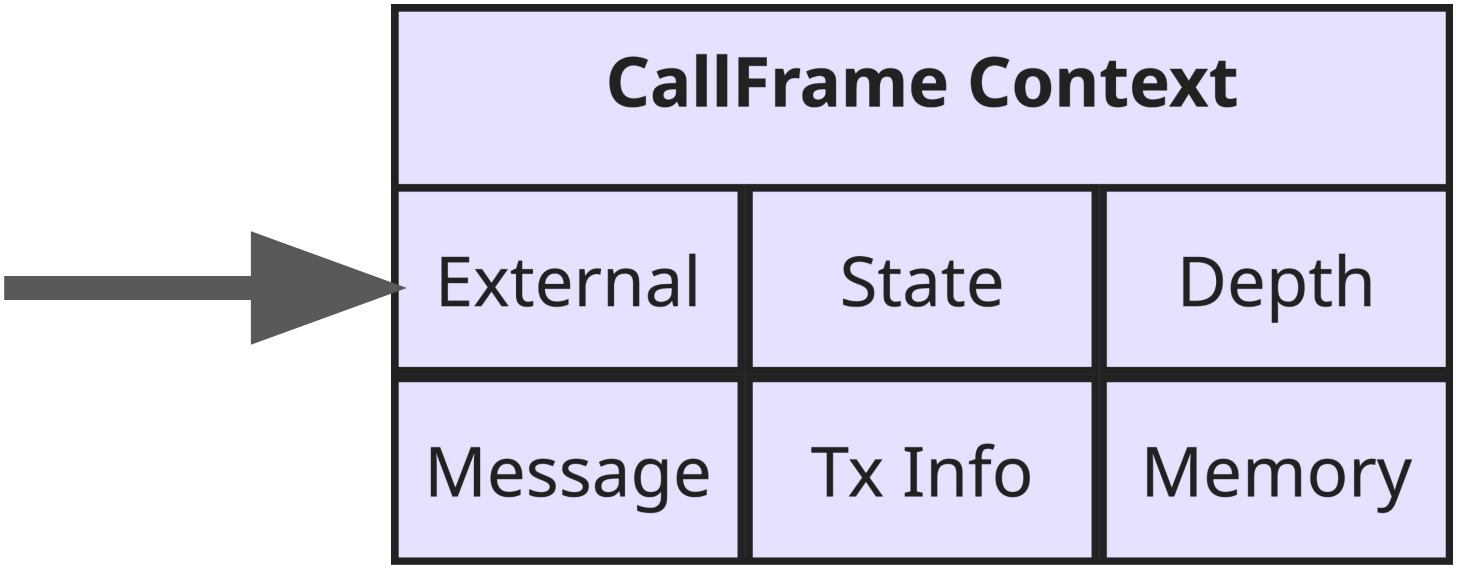
Execution basics: CallFrames



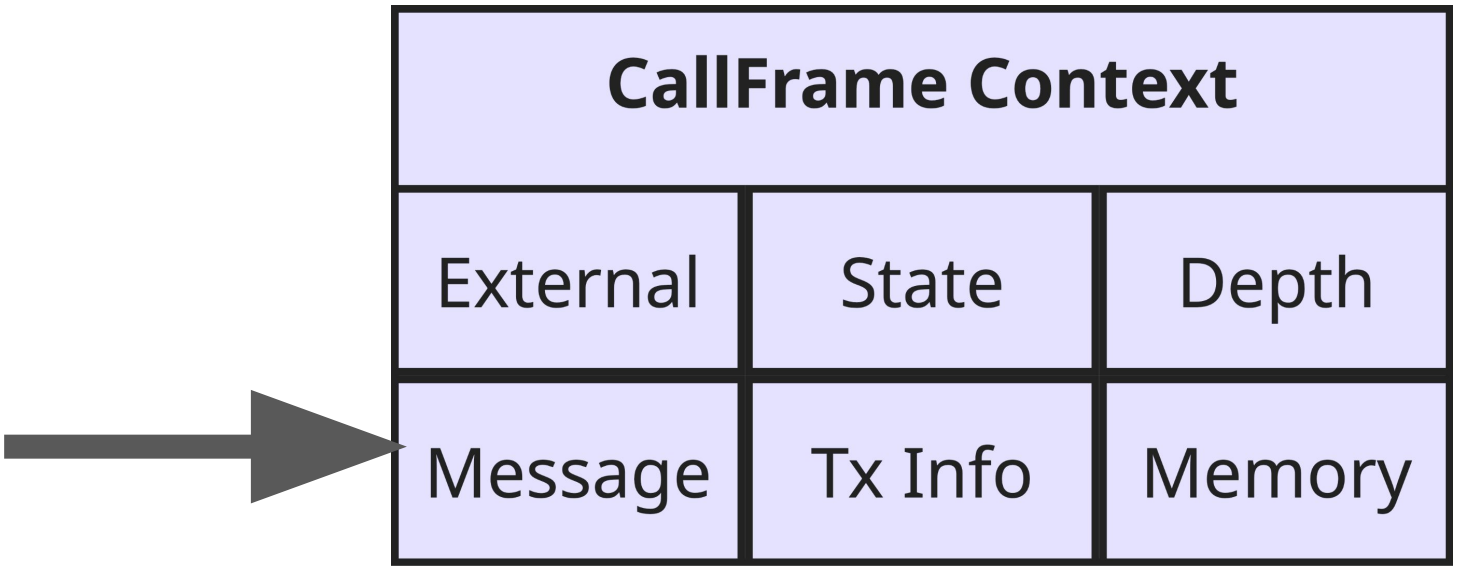
Execution basics: CallFrames



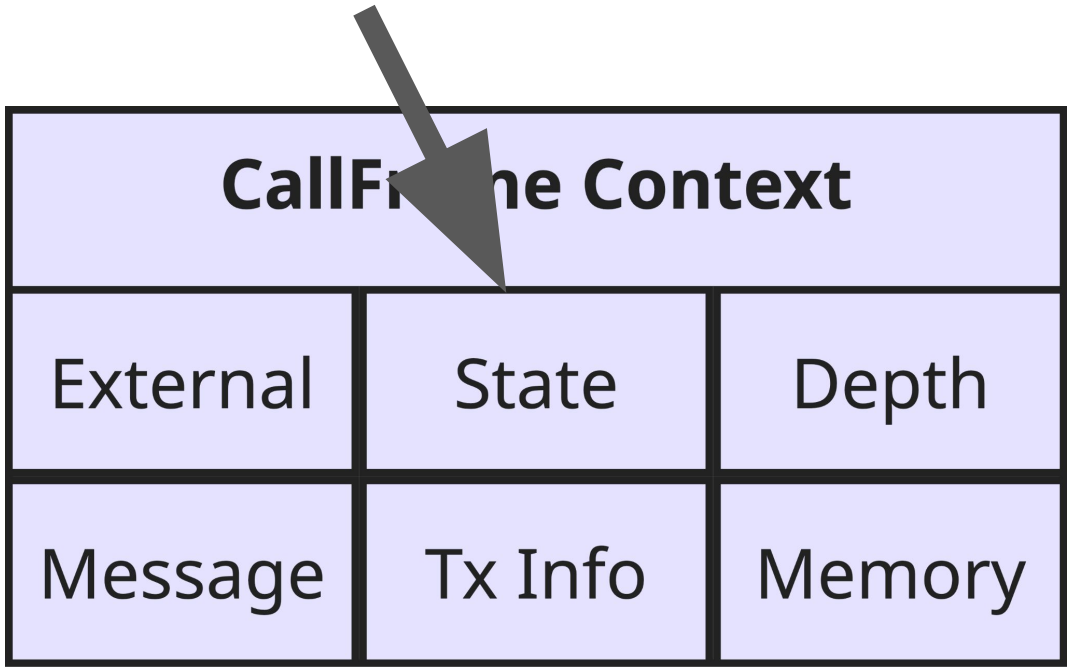
Execution basics: CallFrames



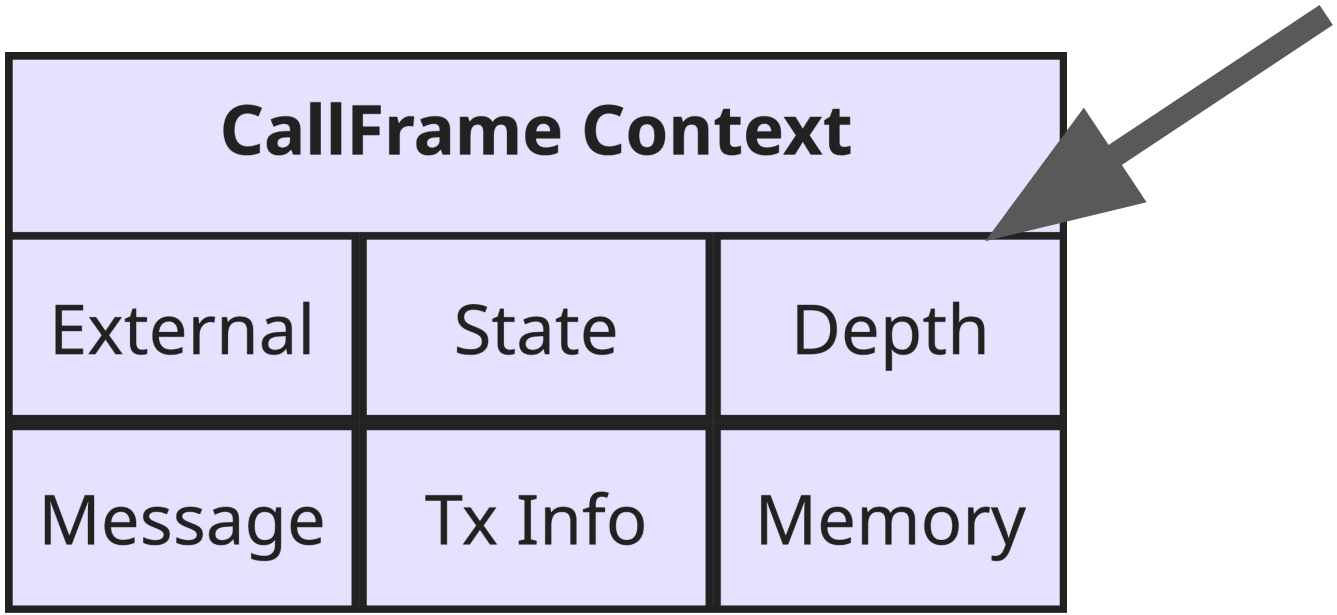
Execution basics: CallFrames



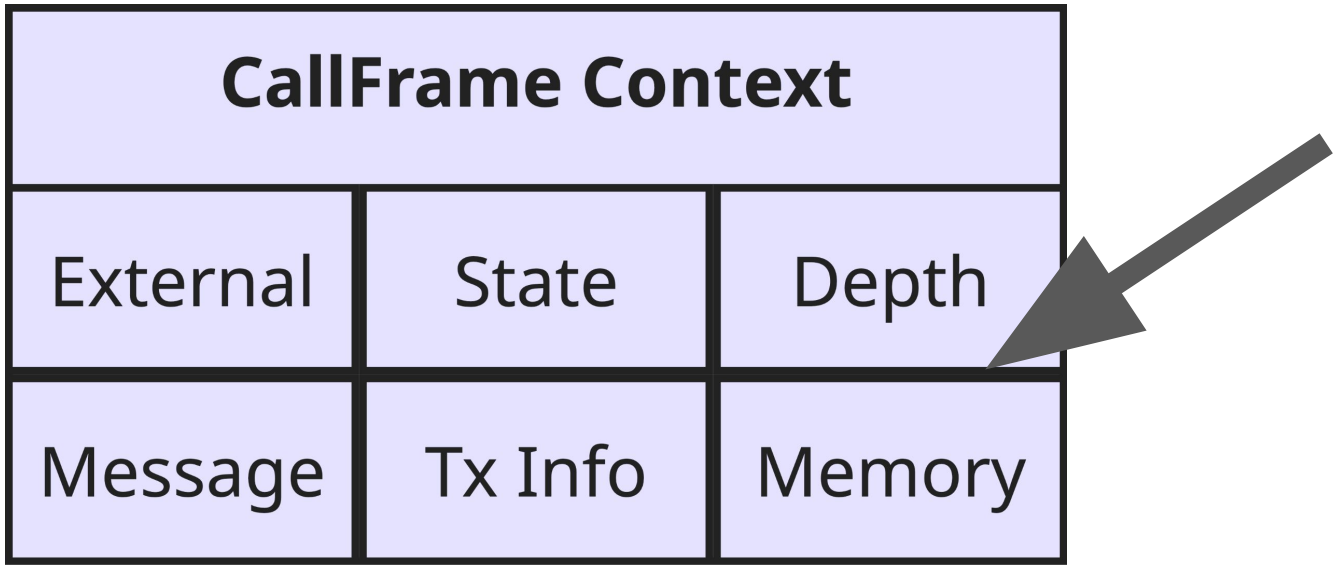
Execution basics: CallFrames



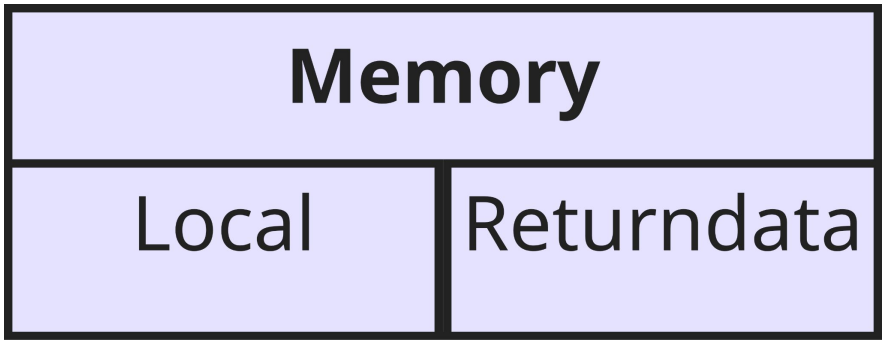
Execution basics: CallFrames



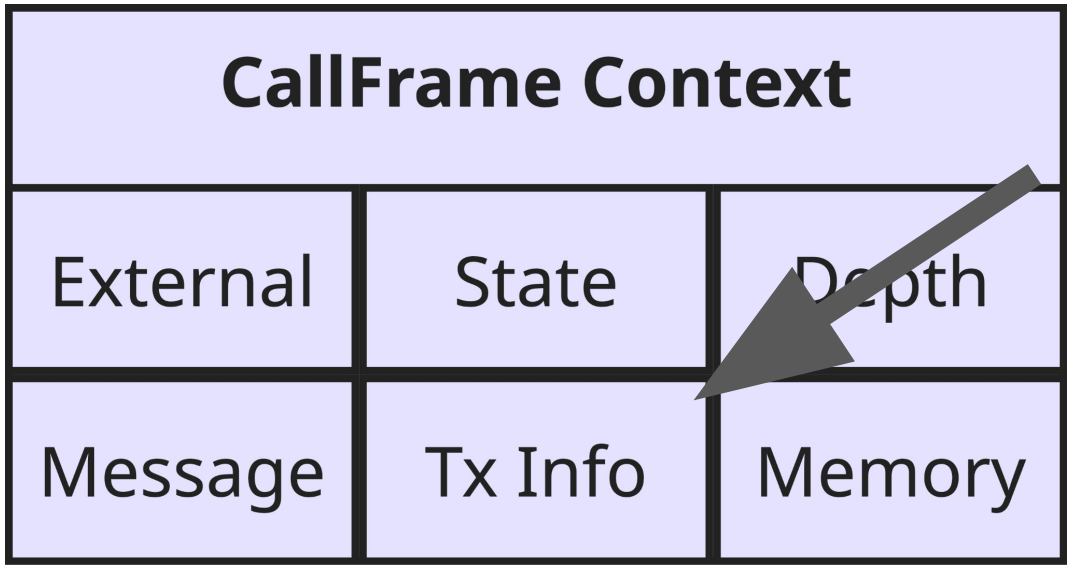
Execution basics: CallFrames



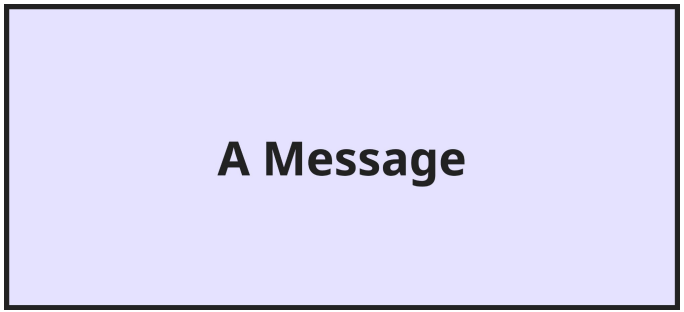
Execution basics: CallFrames



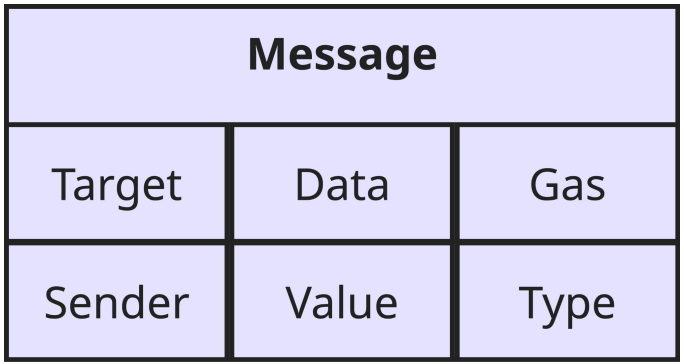
Execution basics: CallFrames



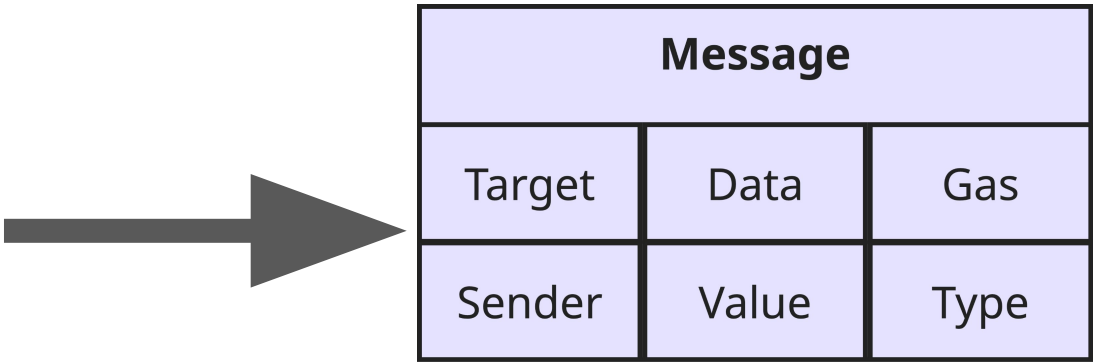
Execution basics: Messages



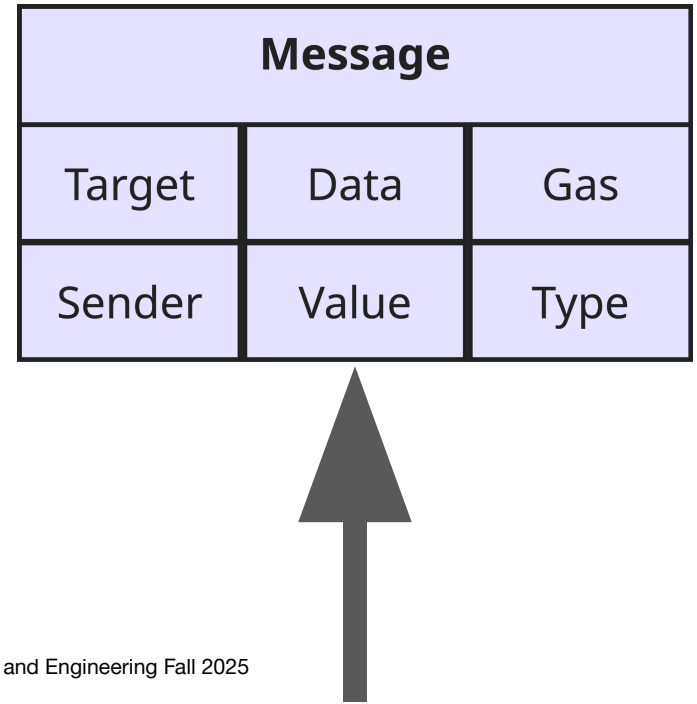
Execution basics: Messages



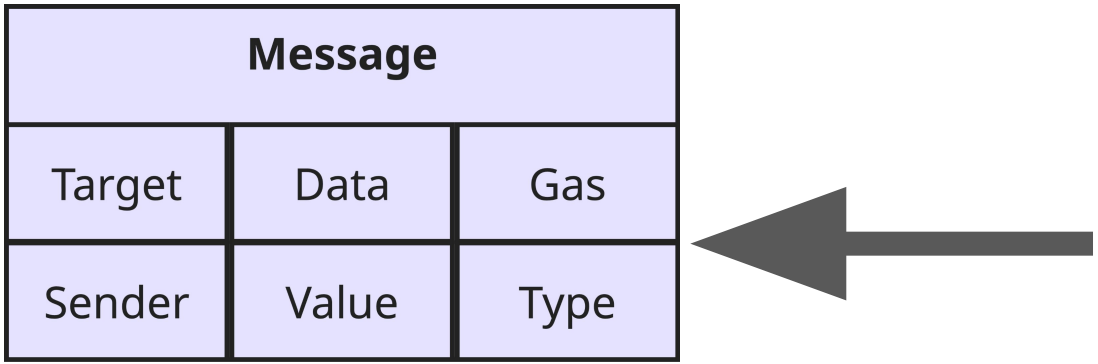
Execution basics: Messages



Execution basics: Messages

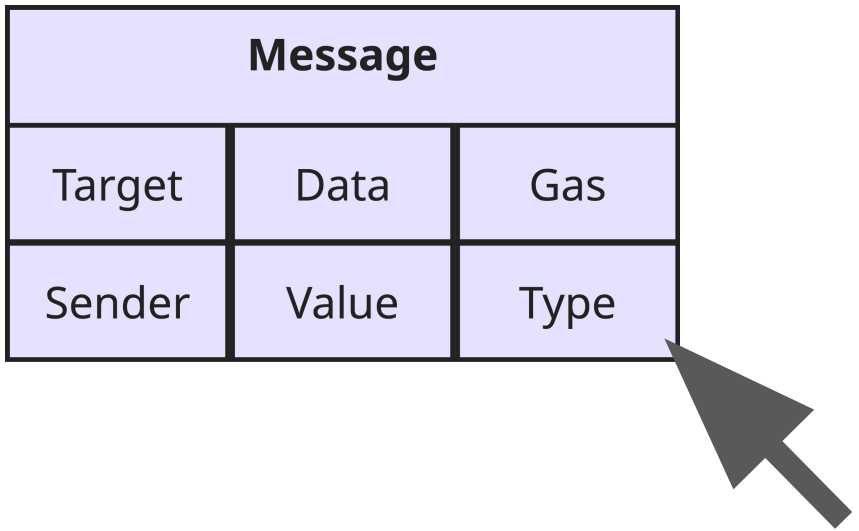


Execution basics: Messages



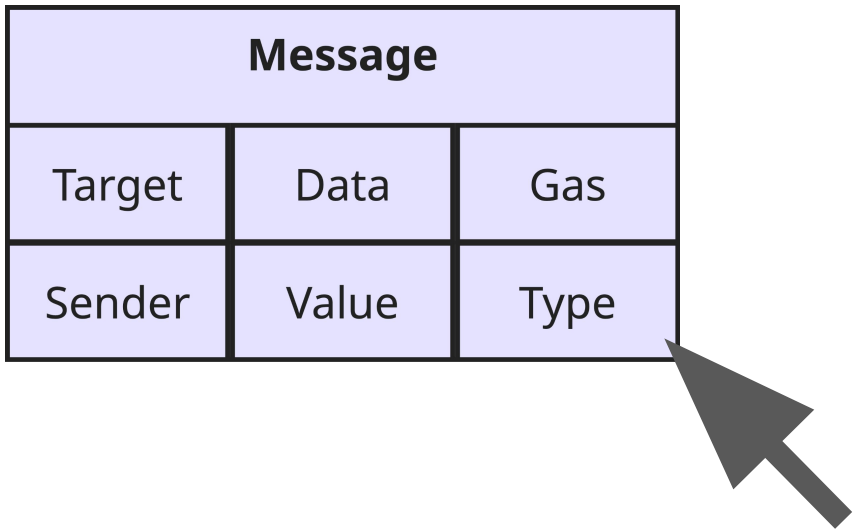
Execution basics: Message Types

1. **Call**
2. Staticcall
3. Delegatecall + Callcode
4. Create + Create2



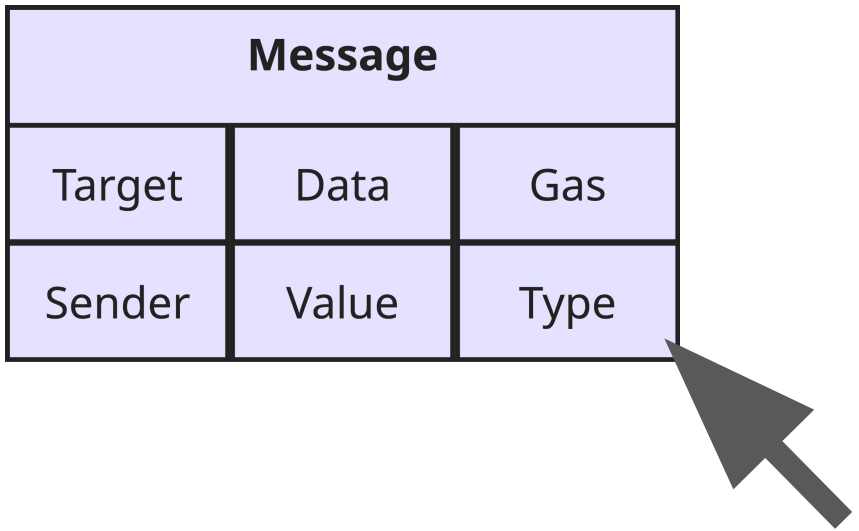
Execution basics: Message Types

1. Call
2. **Staticcall**
3. Delegatecall + Callcode
4. Create + Create2



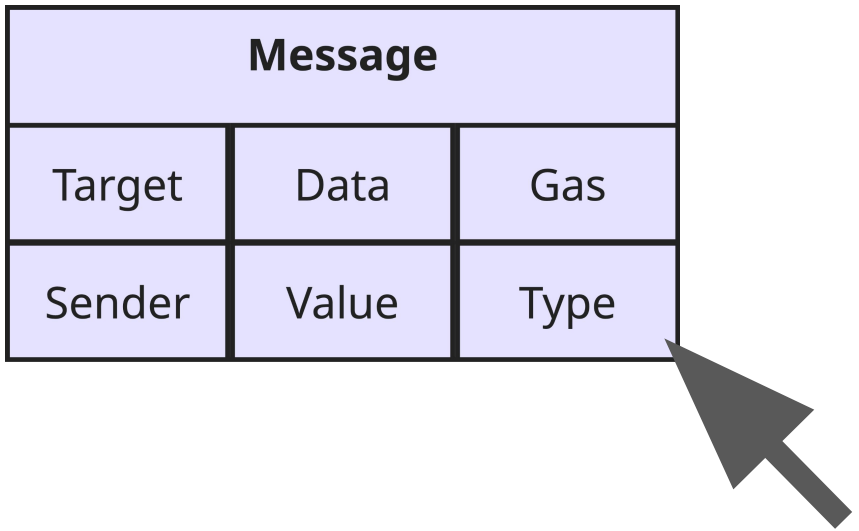
Execution basics: Message Types

1. Call
2. Staticcall
3. **Delegatecall + Callcode**
4. Create + Create2

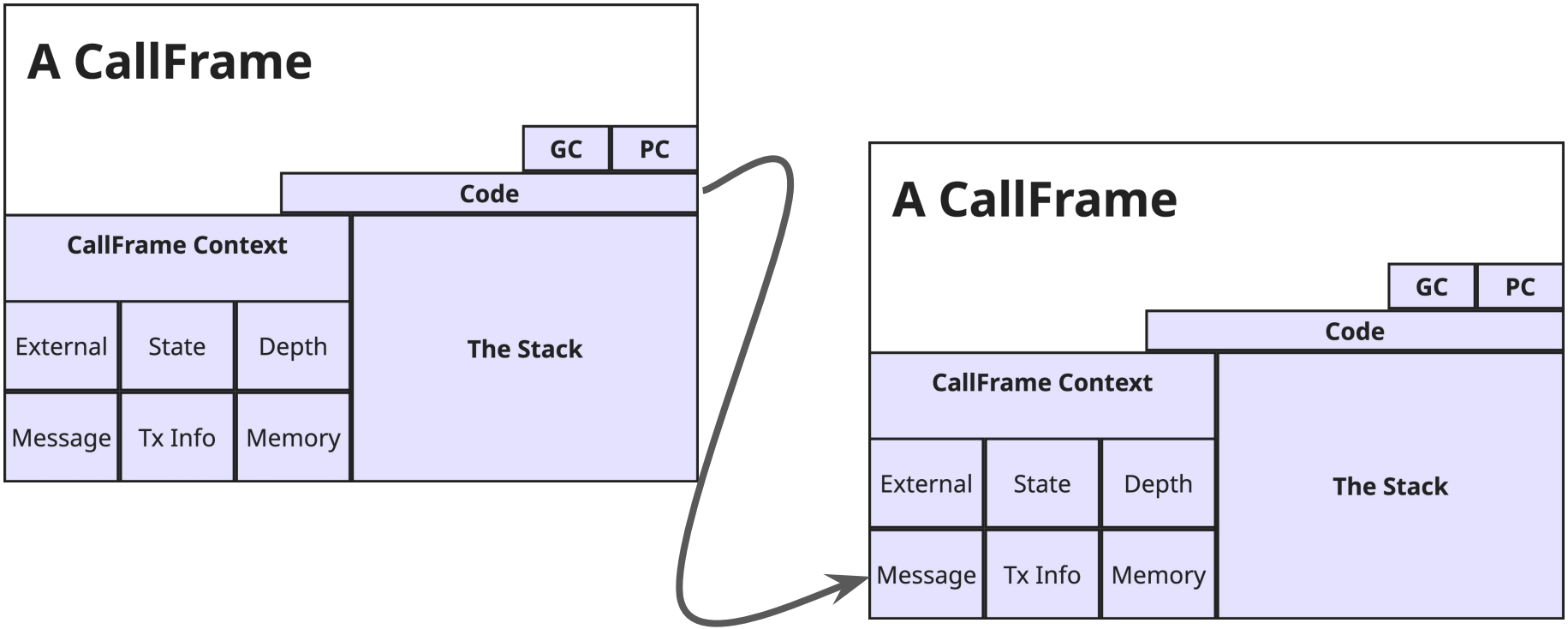


Execution basics: Message Types

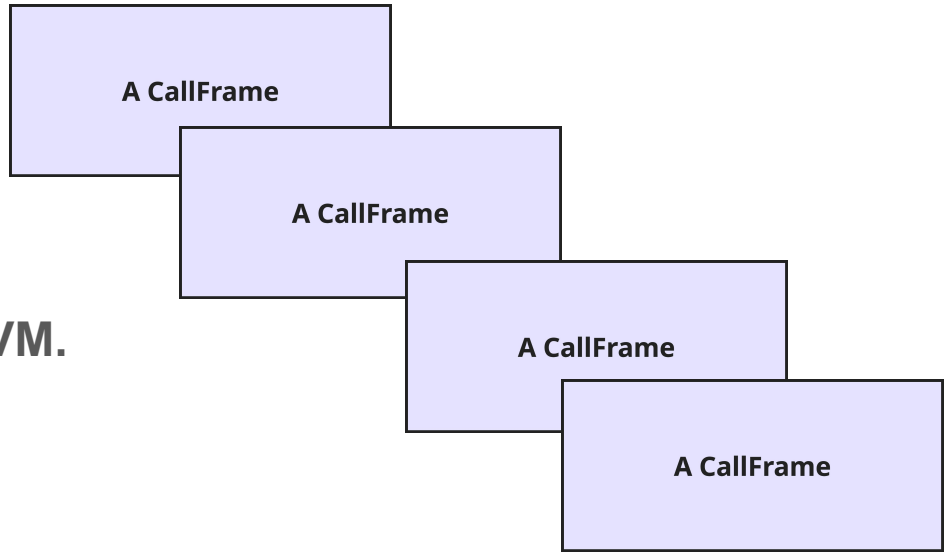
1. Call
2. Staticcall
3. Delegatecall + Callcode
4. **Create + Create2**



Execution basics: The CallStack

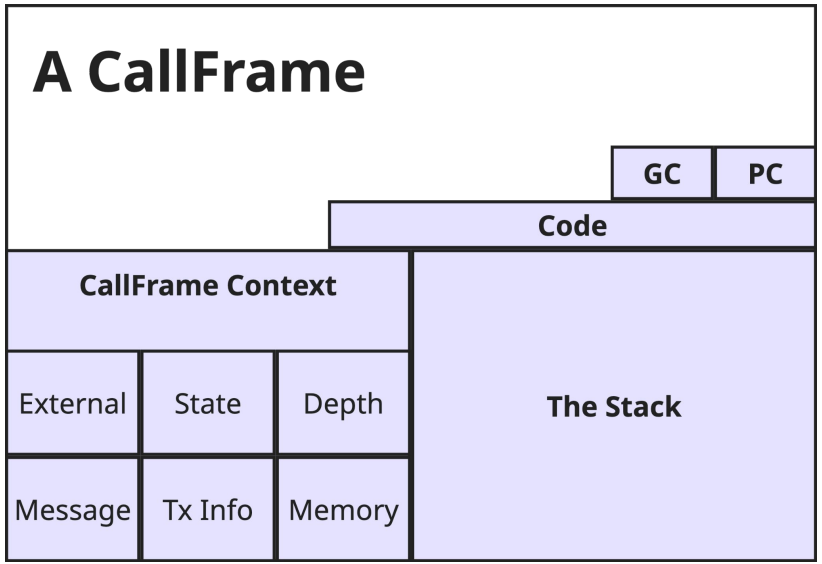


Execution basics: The CallStack



Each of these are an independent EVM.

Execution basics: Callframe Resolution

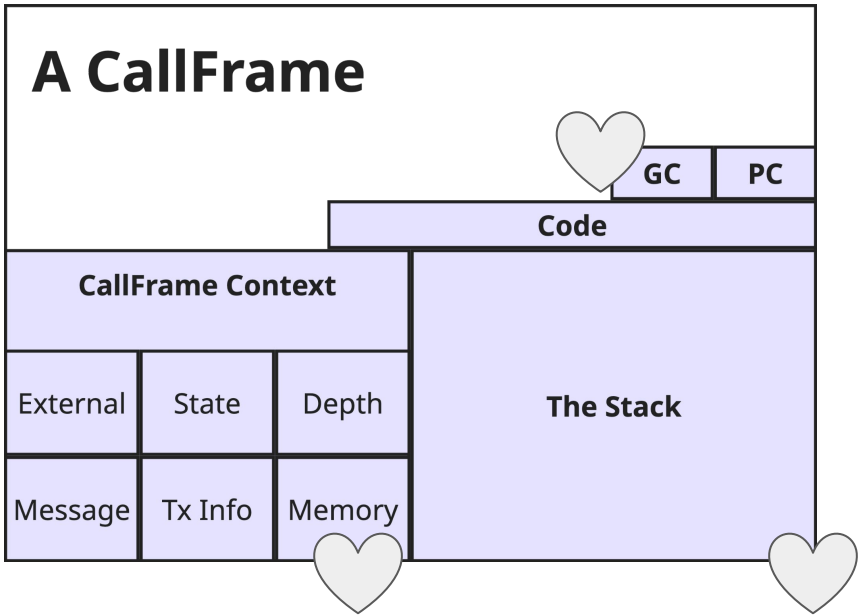


Execution basics: Callframe Resolution

null code (SUCCESS)

If the target has no code:

- Push true to parent stack
- Set parent returndata to empty
- Increase parent GC by child GC
- Pretty Trivial.

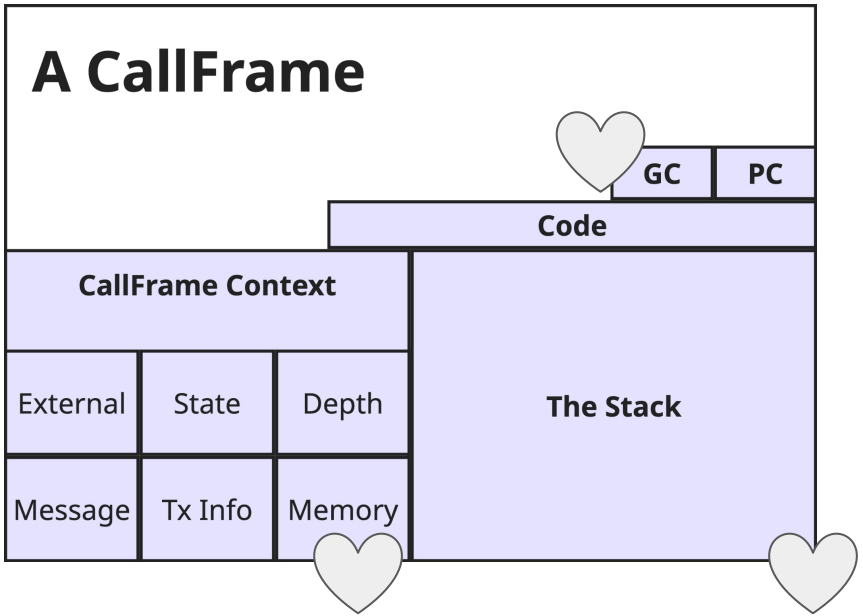


Execution basics: Callframe Resolution

Halt (FAILURE)

There are many halt conditions :(

- **Discard local context and state**
- Push false to parent stack
- Set parent returndata to empty
- Increase parent GC by child **gas**
  - **NOT the child gas counter**

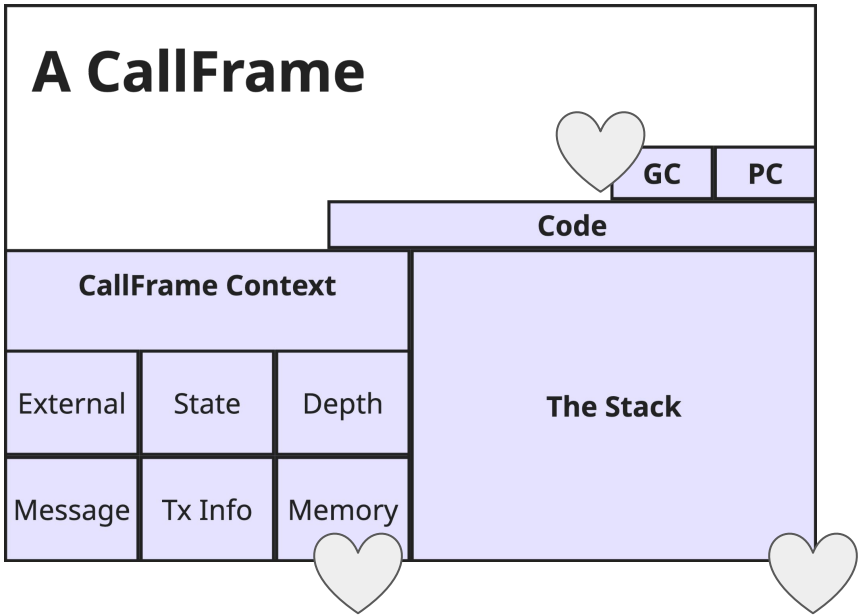


Execution basics: Callframe Resolution

Revert (FAILURE)

If target executes the REVERT opcode:

- Read offset and size from stack
- **Discard local context and state**
- Push false to parent stack
- Set parent returndata to
  - localMemory[offset..offset+size]
- Increase parent GC by child GC

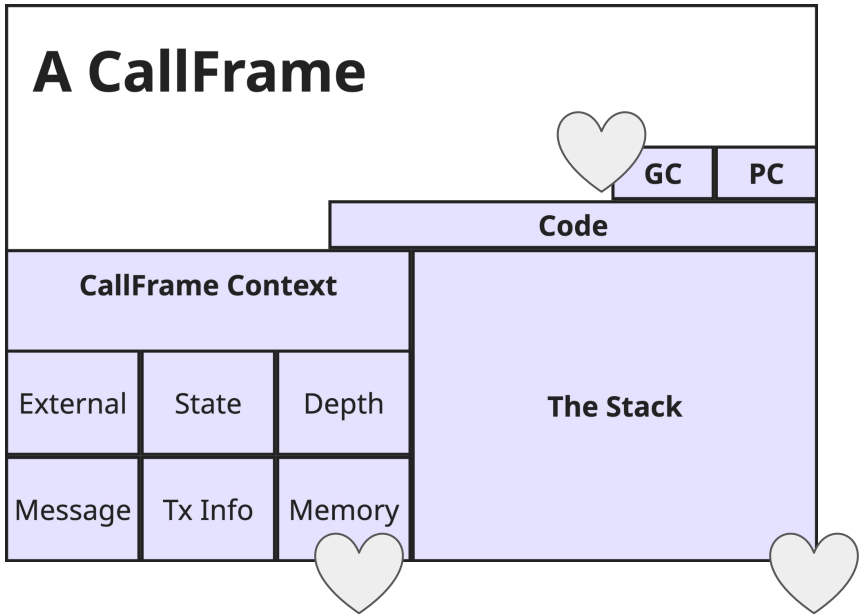


Execution basics: Callframe Resolution

Return (SUCCESS)

If target executes the RETURN opcode:

- Read offset and size from stack
- Push true to parent stack
- Set parent returndata to
  - localMemory[offset..offset+size]
- Increase parent GC by child GC
- **Fold local context and state into parent callframe.**

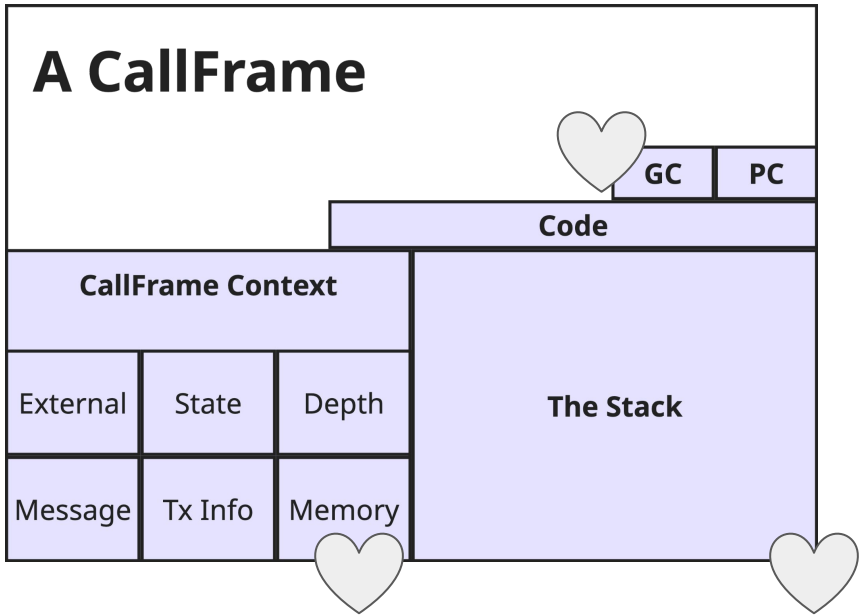


Execution basics: Callframe Resolution

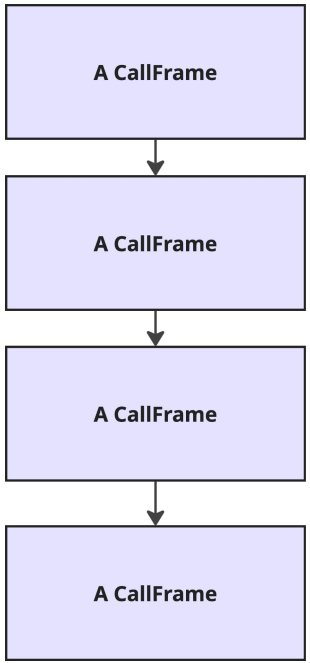
Return – but from a CREATE(2)

If target executes the RETURN opcode:

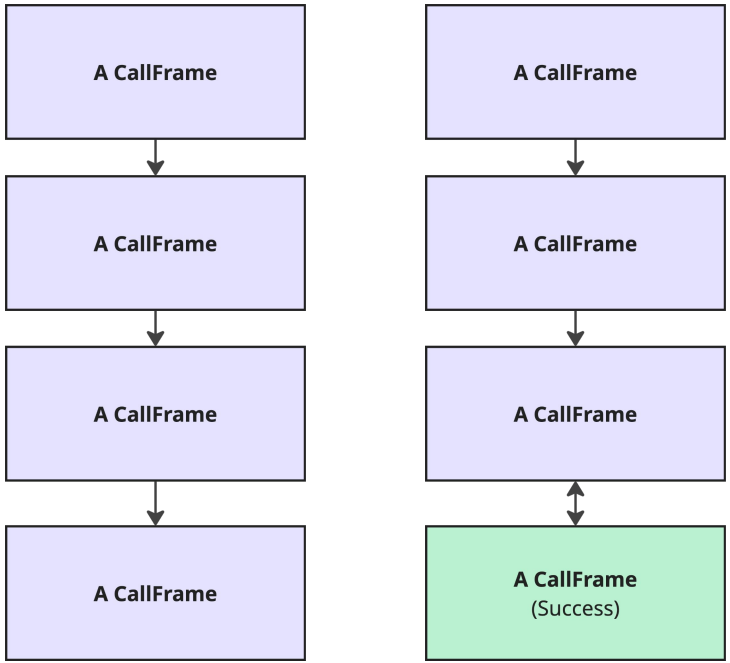
- Read offset and size from stack
- Push this address to parent stack
- Set the **code** of this account to:
  - localMemory[offset..offset+size]
- Increase parent GC by child GC
- **Fold local context and state into parent callframe.**



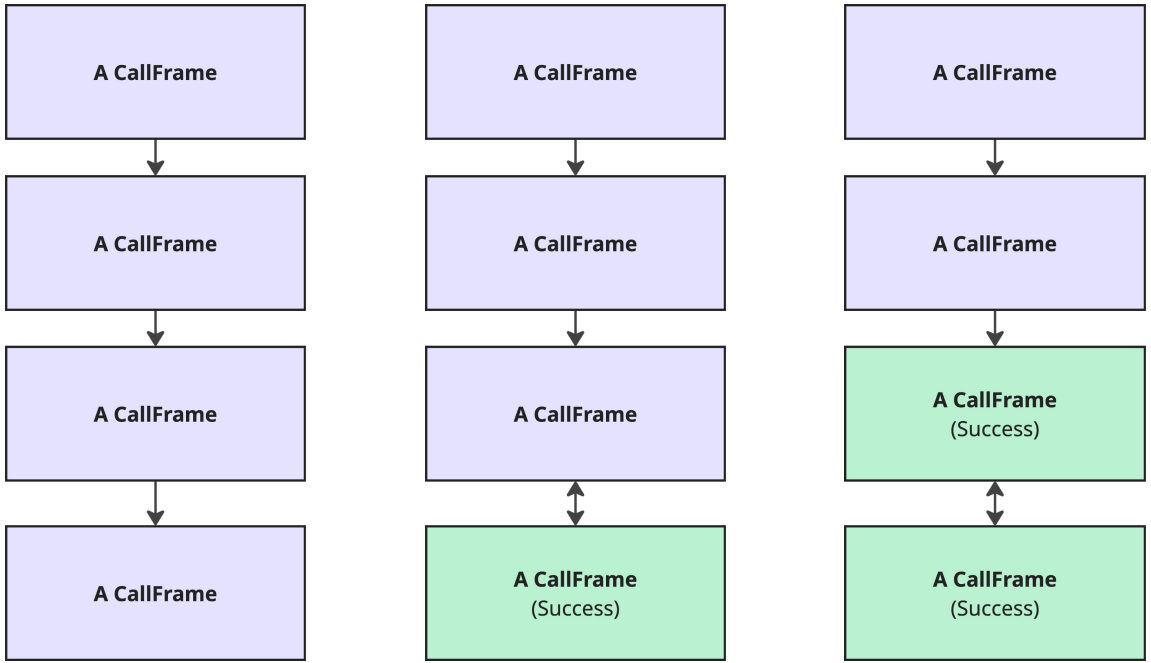
Execution basics: Reverting in the CallStack



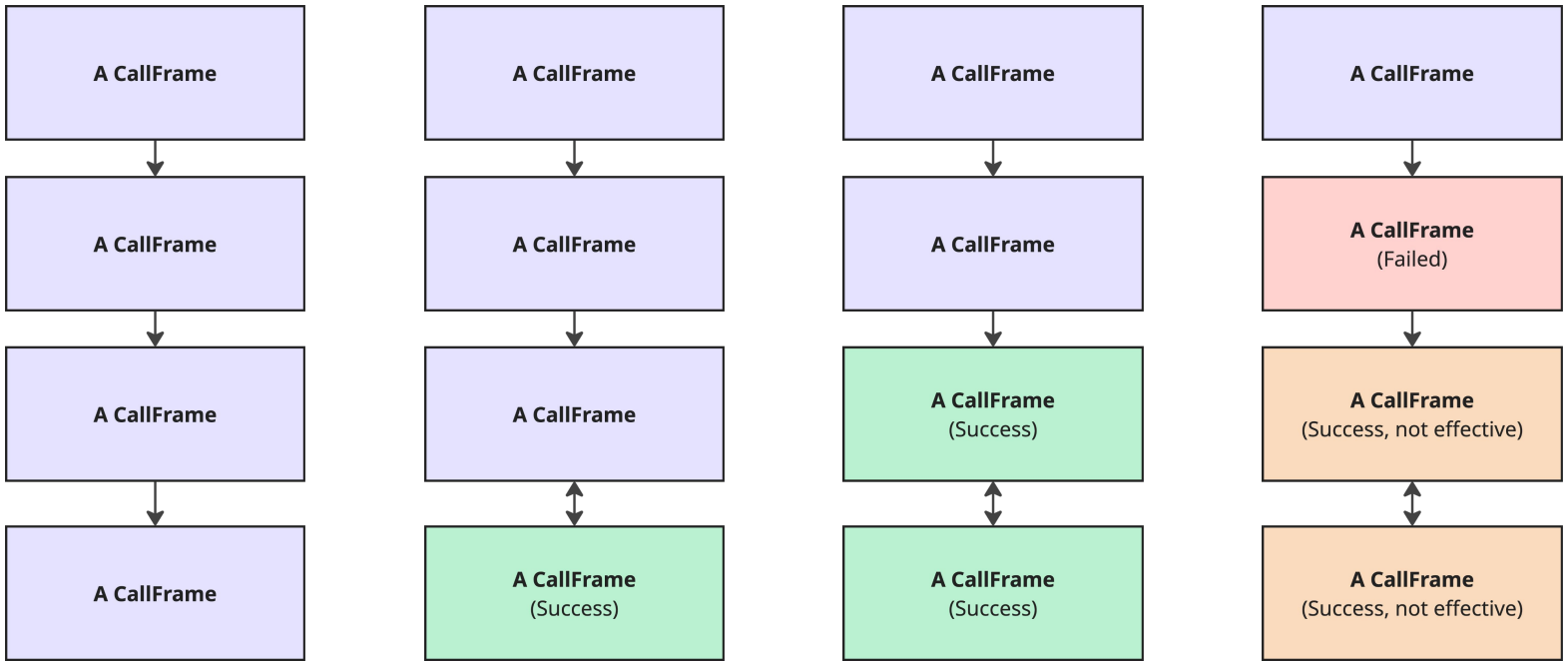
Execution basics: Reverting in the CallStack



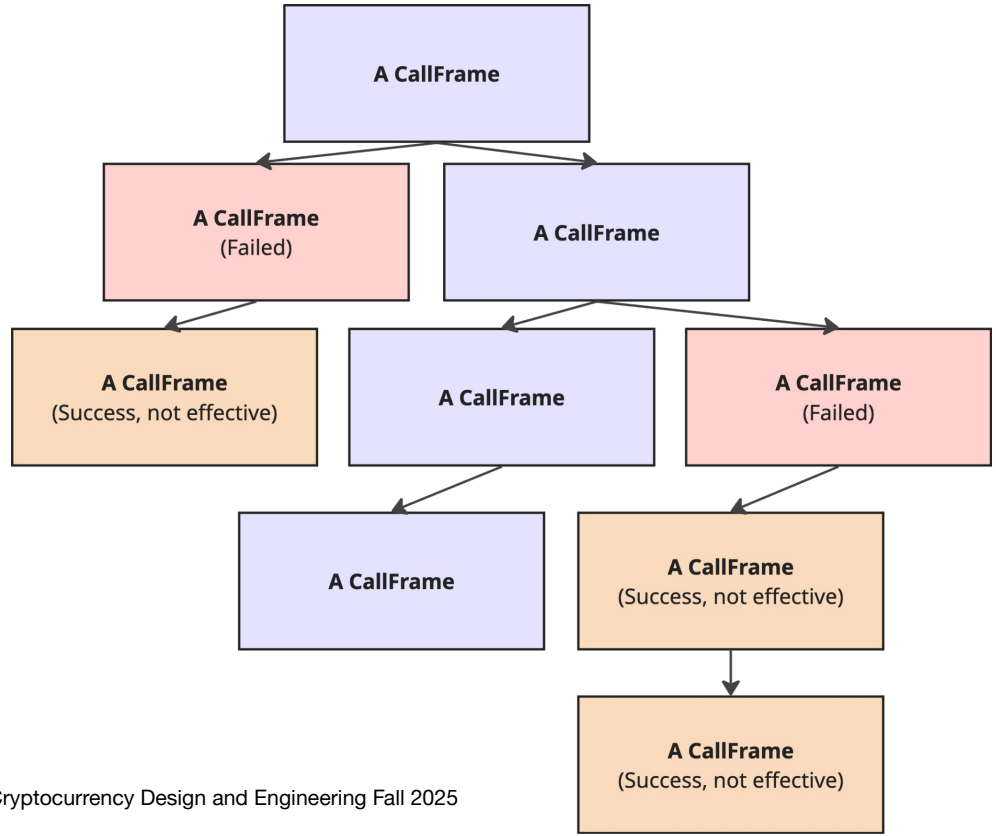
Execution basics: Reverting in the CallStack



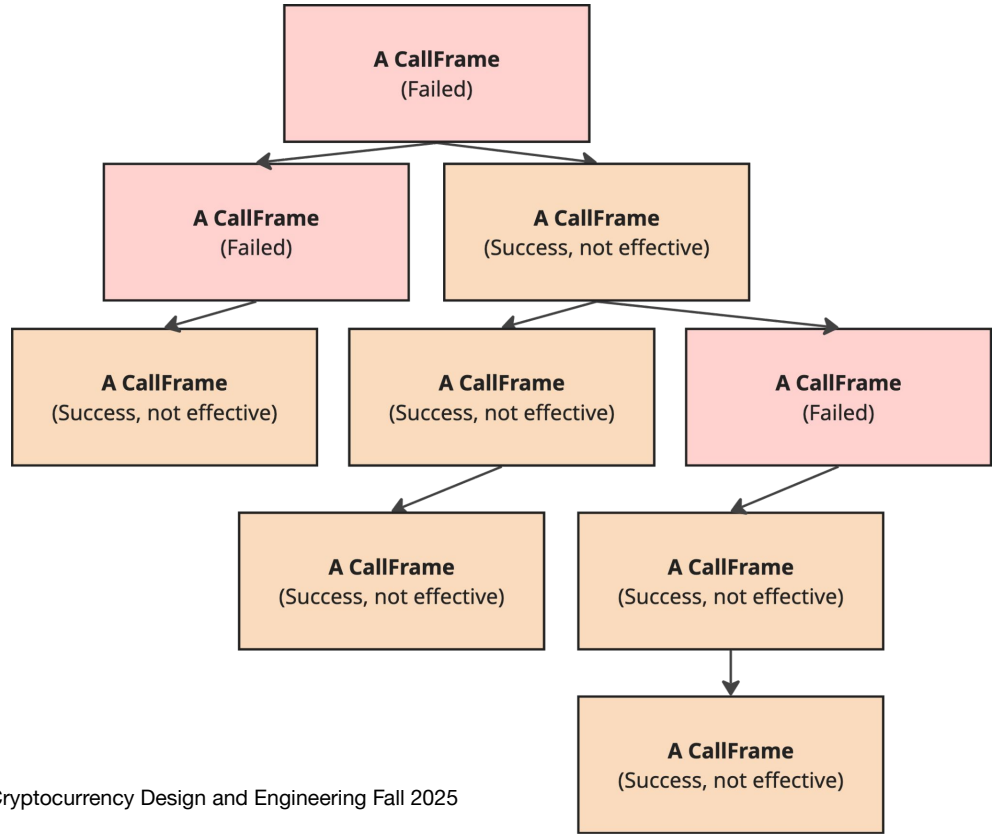
Execution basics: Reverting in the CallStack



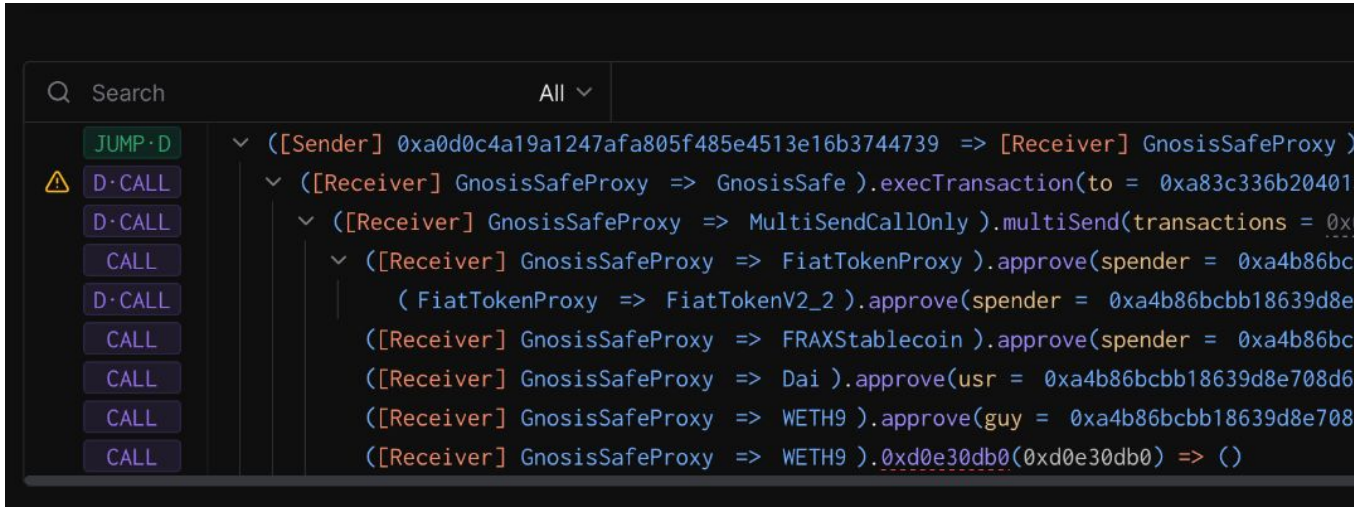
Execution basics: Reverting in the CallStack



Execution basics: Reverting in the CallStack



Execution basics: Reverting in the CallStack



I screenshot this from http://tender.ly

EVM Theory

- ~~The State~~
- ~~Execution Basics~~
- Some Historical Changes

Some historical changes: EIPs we already know

- **EIP-7**: DELEGATECALL opcode (2015)
- **EIP-140**: REVERT opcode (2017)
- **EIP-211**: returndata in callframe context (2017)
- **EIP-214**: STATICCALL opcode (2017)
- **EIP-1014**: CREATE2 opcode (2018)

https://eips.ethereum.org/core

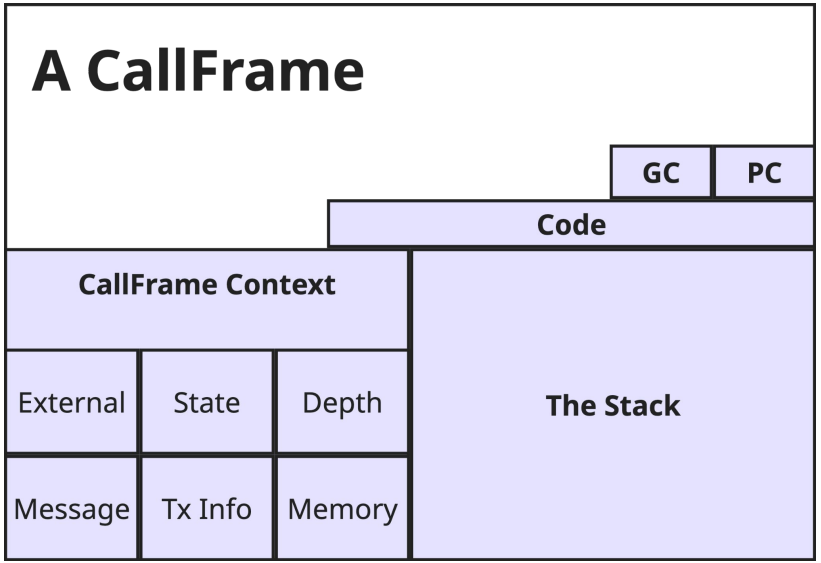
Some historical changes: Challenging EIPs

- **EIPs-1283, 2200, 2929, 2930 and 8037**: State access gas costs (every year)
- **EIP-6780**: SELFDESTRUCT changes (2023)
- **EIP-7251**: Increase MEB (2023-24)
- **EIP-7702**: EOA Delegations (2024-25)

https://eips.ethereum.org/core

Some historical changes

All of this is mutable



EVM Theory

- ~~The State~~
- ~~Execution Basics~~
- ~~Some Historical Changes~~

Questions and Break

EVM Engineering

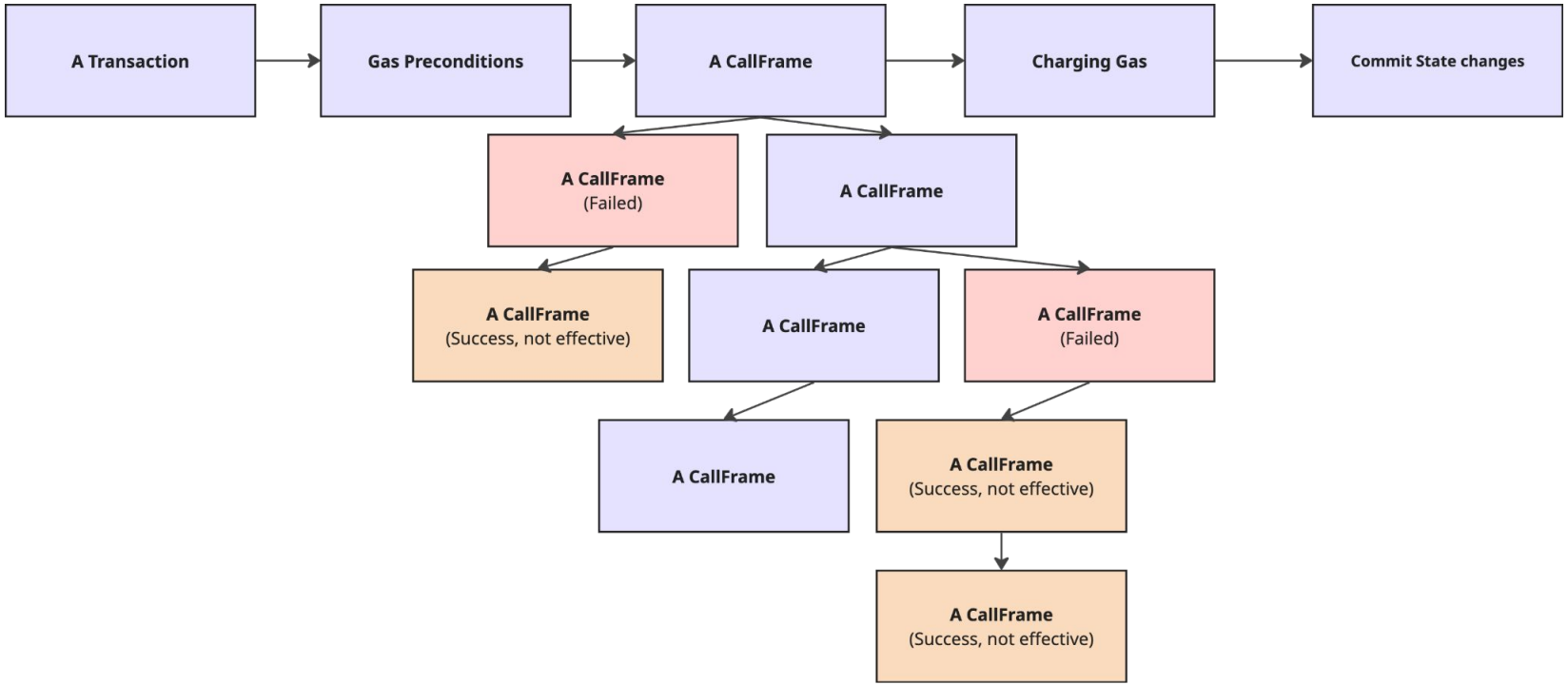
- Transactions

- Solidity & its standards

Transactions

Screenshot from https://etherscan.io

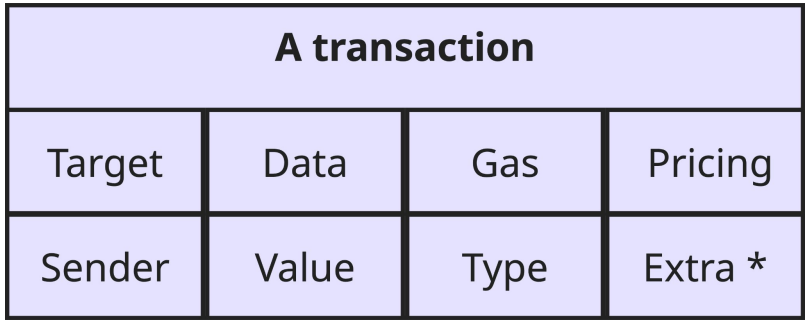
Transactions



Transactions: Properties

All transactions have the following properties:

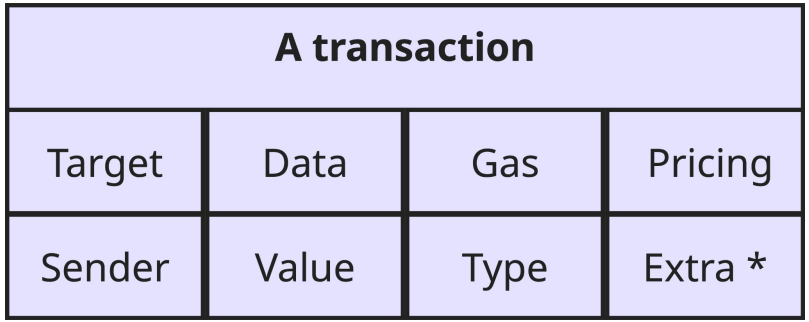
- A valid ECDSA signature
- An amount of gas
- A target (or null for contract creation)
- Value
- A nonce



Transactions: Properties

All transactions have the following properties:

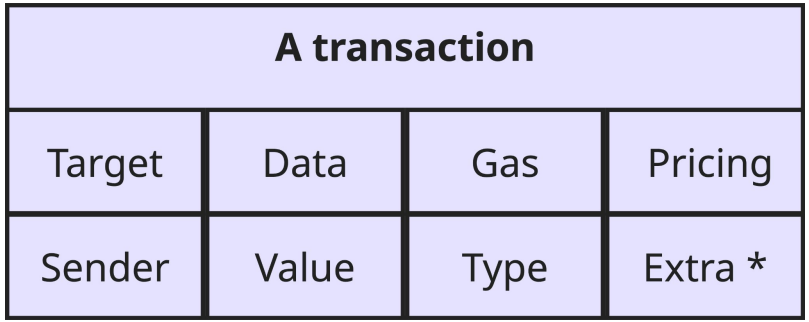
- **A valid ECDSA signature**
- An amount of gas
- A target (or null for contract creation)
- Value
- A nonce



Transactions: Properties

All transactions have the following properties:

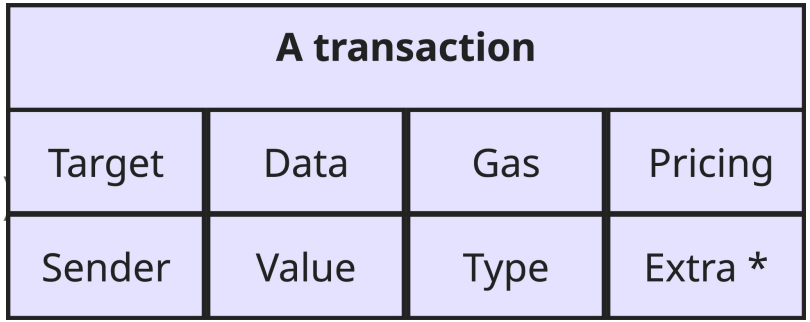
- A valid ECDSA signature
- **An amount of gas**
- A target (or null for contract creation)
- Value
- A nonce



Transactions: Properties

All transactions have the following properties:

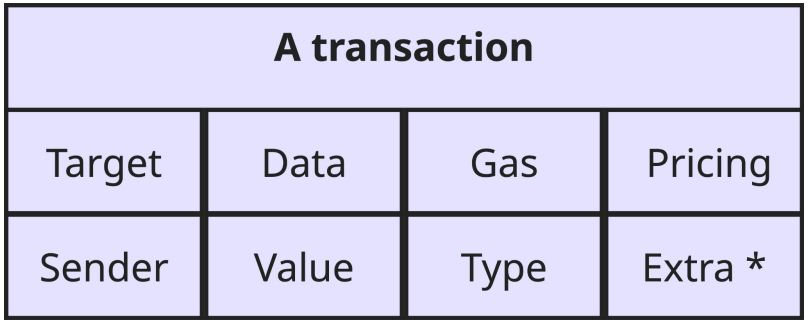
- A valid ECDSA signature
- An amount of gas
- **A target (or null for contract creation)**
- Value
- A nonce



Transactions: Properties

All transactions have the following properties:

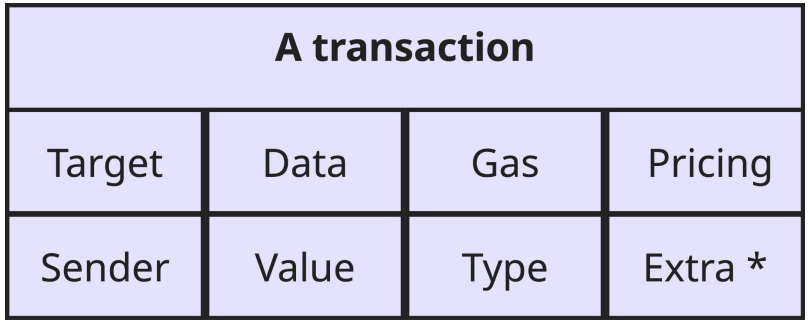
- A valid ECDSA signature
- An amount of gas
- A target (or null for contract creation)
- **Value**
- A nonce



Transactions: Properties

All transactions have the following properties:

- A valid ECDSA signature
- An amount of gas
- A target (or null for contract creation)
- Value
- **A nonce**

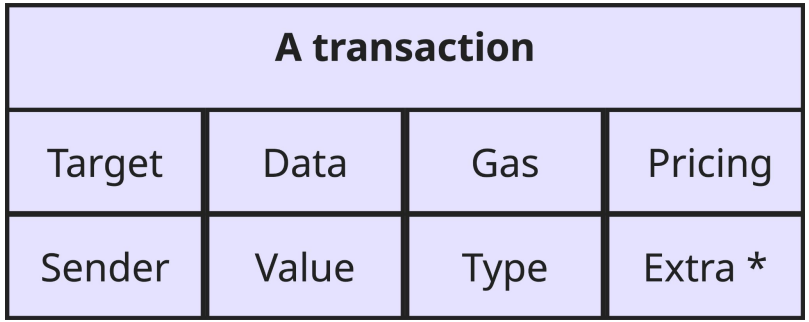


Transactions: Properties

All transactions have the following properties:

- A valid ECDSA signature
- An amount of gas
- A target (or null for contract creation)
- Value
- A nonce

Most of these these look familiar.

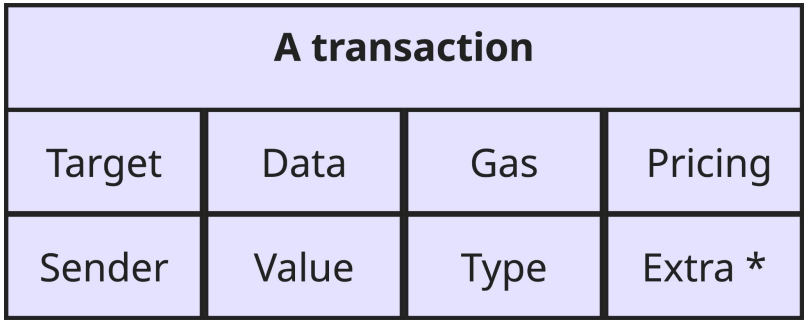


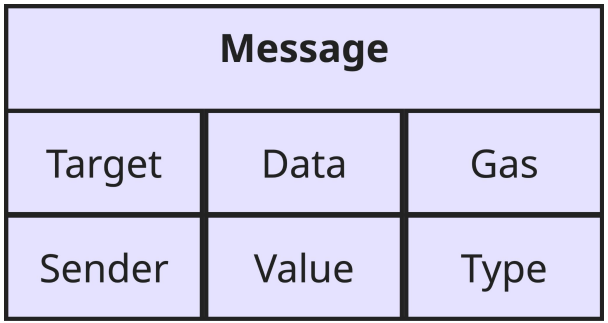
Transactions: Properties

All transactions have the following properties:

- A valid ECDSA signature
- An amount of gas
- A target (or null for contract creation)
- Value
- A nonce

Most of these these look familiar.





Transactions: Types

Let’s talk about the extra field. There are 5 tx types

- Legacy
- EIP-2930 - Access Lists
- EIP-1559 - New gas semantics
- EIP-4844 - Blobs
- EIP-7702 - EOA Delegations

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Types

Let’s talk about the extra field. There are 5 tx types, with unique extra data

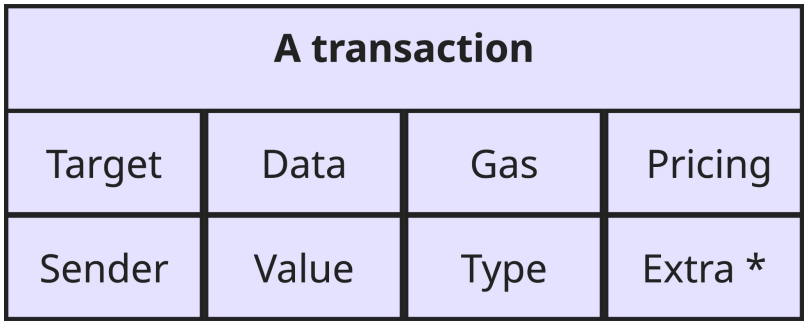
- **Legacy**
- EIP-2930
- EIP-1559
- EIP-4844
- EIP-7702

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Types

Let’s talk about the extra field. There are 5 tx types, with unique extra data

- Legacy
- **EIP-2930 - Access Lists**
- EIP-1559
- EIP-4844
- EIP-7702



Transactions: Types

Let’s talk about the extra field. There are 5 tx types, with unique extra data

- Legacy
- EIP-2930 - Access Lists
- **EIP-1559 - New gas semantics**
- EIP-4844
- EIP-7702

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Types

Let’s talk about the extra field. There are 5 tx types, with unique extra data

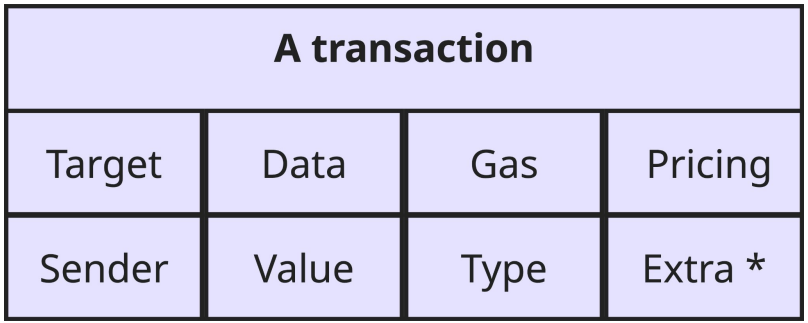
- Legacy
- EIP-2930 - Access Lists
- EIP-1559 - New gas semantics
- **EIP-4844 - Blobs**
- EIP-7702

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Types

Let’s talk about the extra field. There are 5 tx types, with unique extra data

- Legacy
- EIP-2930 - Access Lists
- EIP-1559 - New gas semantics
- EIP-4844 - Blobs
- **EIP-7702 - EOA Delegations**



Transactions: Pricing

There are 3 ways that a transaction can specify its pricing

- EIP-1559
- Legacy
- EIP-4844

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Legacy

Legacy and **EIP-2930 (access list)** transactions specify a gas price.

- gasPrice

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: EIP-1559

EIP-1559, **EIP-7702 (delegation)** transactions specify a max fee, and a max “tip”

- maxFeePerGas
- maxPriorityFeePerGas

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Strongly recommended reading: https://eips.ethereum.org/EIPS/eip-1559

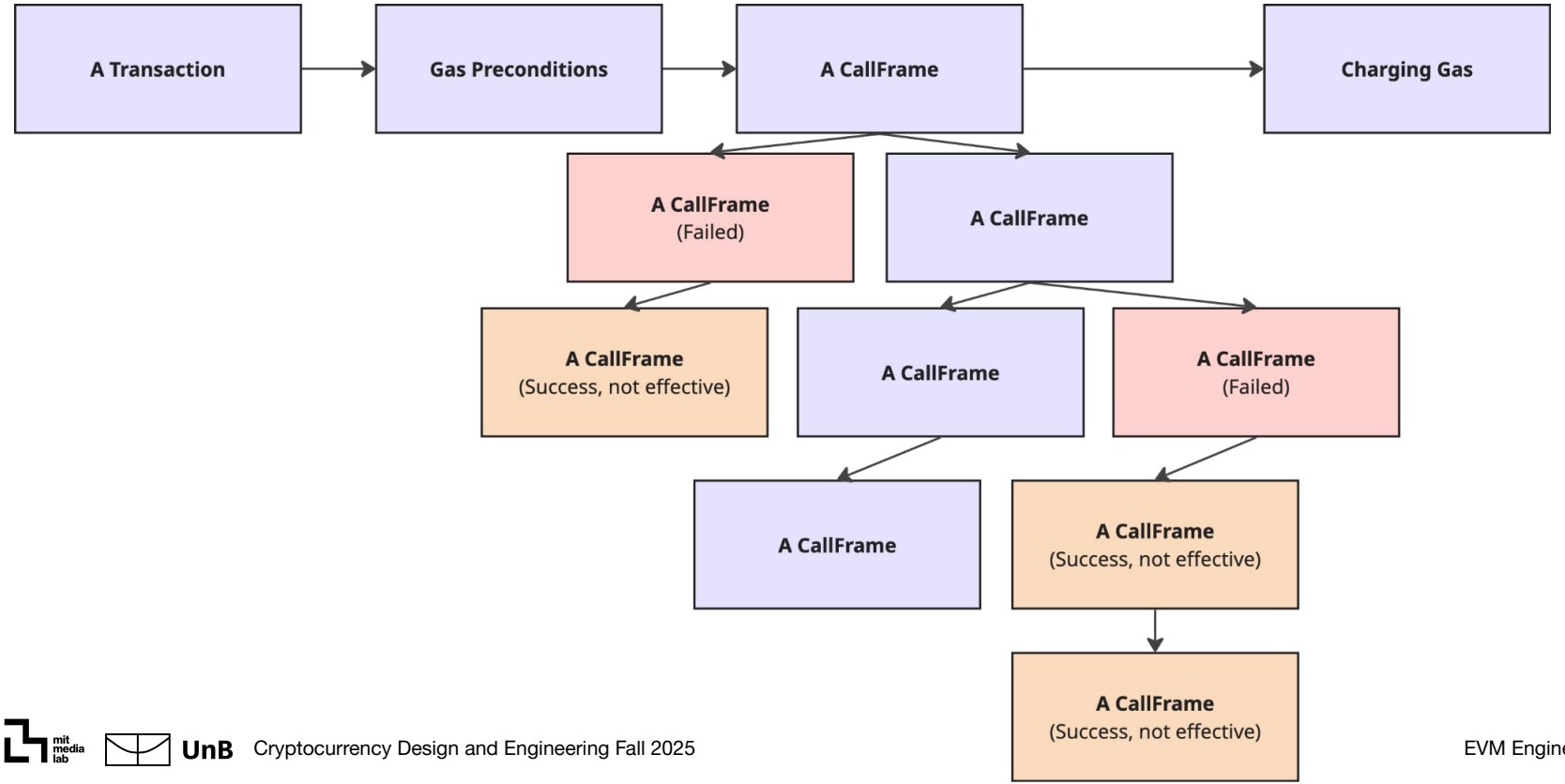
Transactions: EIP-4844

EIP-4844 transactions are as 1559, but include a fee for blob data.

- maxFeePerGas
- maxPriorityFeePerGas
- maxFeePerBlobGas

A transaction			
Target	Data	Gas	Pricing
Sender	Value	Type	Extra *

Transactions: Gas



Transactions: Gas Preconditions

A transaction is only valid if the following conditions hold:

$$\text{basefee} <= \text{maxFeePerGas}^{1}$$

$$\text{blobBaseFee} <= \text{maxFeePerBlobGas}^{2}$$

$$\text{senderBalance} >= (\text{gas} * \text{maxFeePerGas})^{1} + (\text{blobGas} + \text{maxFeePerBlobGas}^{2})^{2}$$

1. for legacy, maxFeePerGas = gasPrice
2. ignored unless transaction is of type 4844

Transactions: Charging Gas

In most cases, gas is charged according to the following formulae:

priorityFee = min(maxFeePerGas - basefee, maxPriorityFeePerGas)[1]
gasFee = (basefee + priorityFee) * gasUsed[2]
tip = priorityFee * gasUsed
blobFee = blobBaseFee * blobGasUsed[3]

fee = gasFee + blobFee

1. for legacy, maxPriorityFeePerGas = gasPrice
2. notionally the gas counter of the root callframe when it resolves, however…..
3. ignored unless transaction is of type 4844. Calculated as a flat fee per blob.

Transactions: Fee Payment

After the root callframe resolves the following happens:

1. The sender account balance is reduced by the **fee**
2. The block's coinbase account balance is increased by the **tip**

Important:

Fees payments **DO NOT** cause EVM execution.

EVM Engineering

- ~~Transactions~~

- Solidity & its standards

- Upgradability

Solidity

- This is pretty small, sorry
- We will break it down.

```
// A contract is an abstraction for an EVM program.
// It is compiled to deploycode. The deploycode instantiates the contract.
UnitTest stub | dependencies | uml | funcSigs | draw.io
contract BasicStorage {
    // Storage variable
    address public storedAddress;

    // Transaction-scoped storage.
    bool transient alreadyChanged;

    // A structured log
    event AddressStored(address indexed _addr, string _data);

    // Private functions
    ftrace | funcSig
    function addressIsEven(address _addr) private pure returns (bool) {
        return uint160(_addr) % 2 == 0;
    }

    // Public functions are included in the solidity preamble jumptable
    // Implicitly unpacks the calldata bytes into the arguments.
    ftrace | funcSig
    function storeAddress(address _addr, string calldata _data) public {
        // Reverts if already modified in this tx
        require(!alreadyChanged, "Reentrant call");
        // Reverts if address is not even.
        require(addressIsEven(_addr), "Address is not even");
        // modifies storage.
        storedAddress = _addr;
        // locks further modifications in this tx.
        alreadyChanged = true;
        // implicitly copies _data from calldata to memory
        emit AddressStored(_addr, _data);
    }
}
```

Solidity

Solidity is a

- strongly typed
- opinionated
- high level language
- that compiles to EVM bytecode

Strongly recommended reading: https://soliditylang.org

Solidity: Bits and Pieces

```
1   // A contract is an abstraction for an EVM program.
2   // It is compiled to deploycode. The deploycode instantiates the contract.
    UnitTest stub | dependencies | uml | funcSigs | draw.io
3   contract BasicStorage {
4       // Storage variable
5       address public storedAddress;
6
7       // Transaction-scoped storage.
8       bool transient alreadyChanged;
9
10      // A structured log
11      event AddressStored(address indexed _addr, string _data);
12
```

Solidity: Bits and Pieces

```
1   // A contract is an abstraction for an EVM program.
2   // It is compiled to deploycode. The deploycode instantiates the contract.
    UnitTest stub | dependencies | uml | funcSigs | draw.io
3   contract BasicStorage {
4       // Storage variable
5       address public storedAddress;
6
7       // Transaction-scoped storage.
8       bool transient alreadyChanged;
9
10      // A structured log
11      event AddressStored(address indexed _addr, string _data);
12
```

Solidity: Bits and Pieces

```
1   // A contract is an abstraction for an EVM program.
2   // It is compiled to deploycode. The deploycode instantiates the contract.
    UnitTest stub | dependencies | uml | funcSigs | draw.io
3   contract BasicStorage {
4       // Storage variable
5       address public storedAddress;
6
7       // Transaction-scoped storage.
8       bool transient alreadyChanged;
9
10      // A structured log
11      event AddressStored(address indexed _addr, string _data);
12
```

Solidity: Bits and Pieces

```
1   // A contract is an abstraction for an EVM program.
2   // It is compiled to deploycode. The deploycode instantiates the contract.
    UnitTest stub | dependencies | uml | funcSigs | draw.io
3   contract BasicStorage {
4       // Storage variable
5       address public storedAddress;
6
7       // Transaction-scoped storage.
8       bool transient alreadyChanged;
9
10      // A structured log
11      event AddressStored(address indexed _addr, string _data);
12
```

Solidity: Bits and Pieces

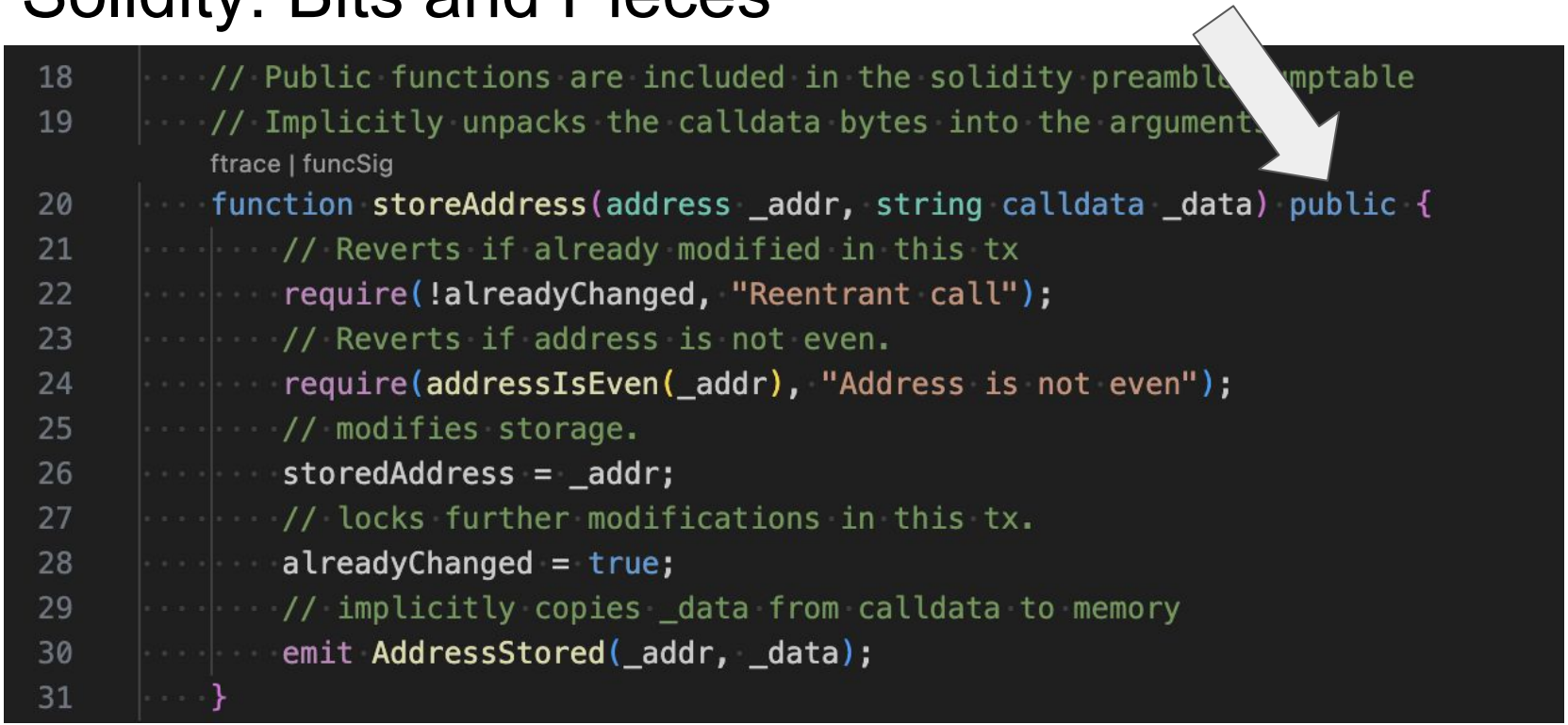
```
1   // A contract is an abstraction for an EVM program.
2   // It is compiled to deploycode. The deploycode instantiates the contract.
    UnitTest stub | dependencies | uml | funcSigs | draw.io
3   contract BasicStorage {
4       // Storage variable
5       address public storedAddress;
6
7       // Transaction-scoped storage.
8       bool transient alreadyChanged;
9
10      // A structured log
11      event AddressStored(address indexed _addr, string _data);
12
```

Solidity: Bits and Pieces

Solidity: Bits and Pieces

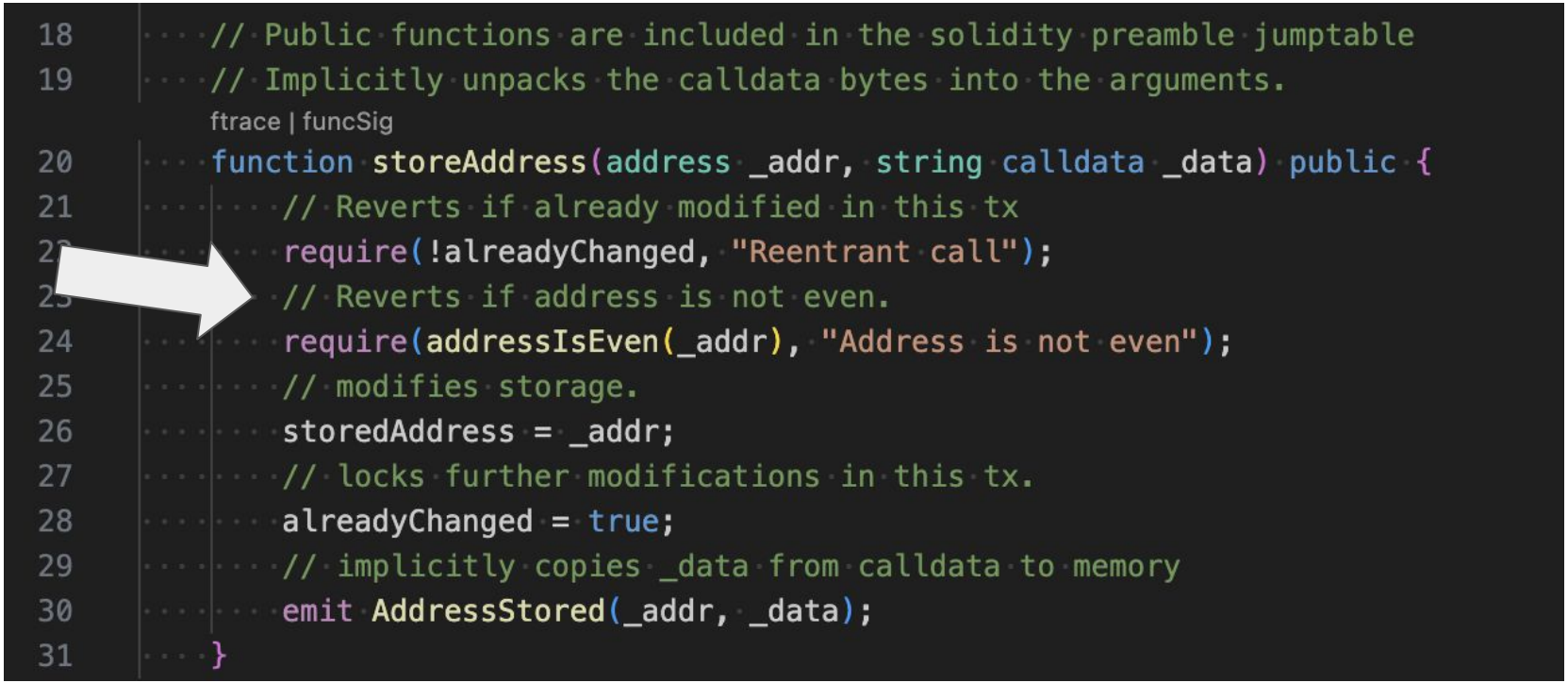
```
    // Function definition
    ftrace | funcSig
    function addressIsEven(address _addr) private pure returns (bool) {
        return uint160(_addr) % 2 == 0;
    }
```

Solidity: Bits and Pieces



```
18      // Public functions are included in the solidity preambl mptable
19      // Implicitly unpacks the calldata bytes into the argument
        ftrace | funcSig
20      function storeAddress(address _addr, string calldata _data) public {
21          // Reverts if already modified in this tx
22          require(!alreadyChanged, "Reentrant call");
23          // Reverts if address is not even.
24          require(addressIsEven(_addr), "Address is not even");
25          // modifies storage.
26          storedAddress = _addr;
27          // locks further modifications in this tx.
28          alreadyChanged = true;
29          // implicitly copies _data from calldata to memory
30          emit AddressStored(_addr, _data);
31      }
```

Solidity: Bits and Pieces



```
18      // Public functions are included in the solidity preamble jumptable
19      // Implicitly unpacks the calldata bytes into the arguments.
        ftrace | funcSig
20      function storeAddress(address _addr, string calldata _data) public {
21          // Reverts if already modified in this tx
22          require(!alreadyChanged, "Reentrant call");
23          // Reverts if address is not even.
24          require(addressIsEven(_addr), "Address is not even");
25          // modifies storage.
26          storedAddress = _addr;
27          // locks further modifications in this tx.
28          alreadyChanged = true;
29          // implicitly copies _data from calldata to memory
30          emit AddressStored(_addr, _data);
31      }
```

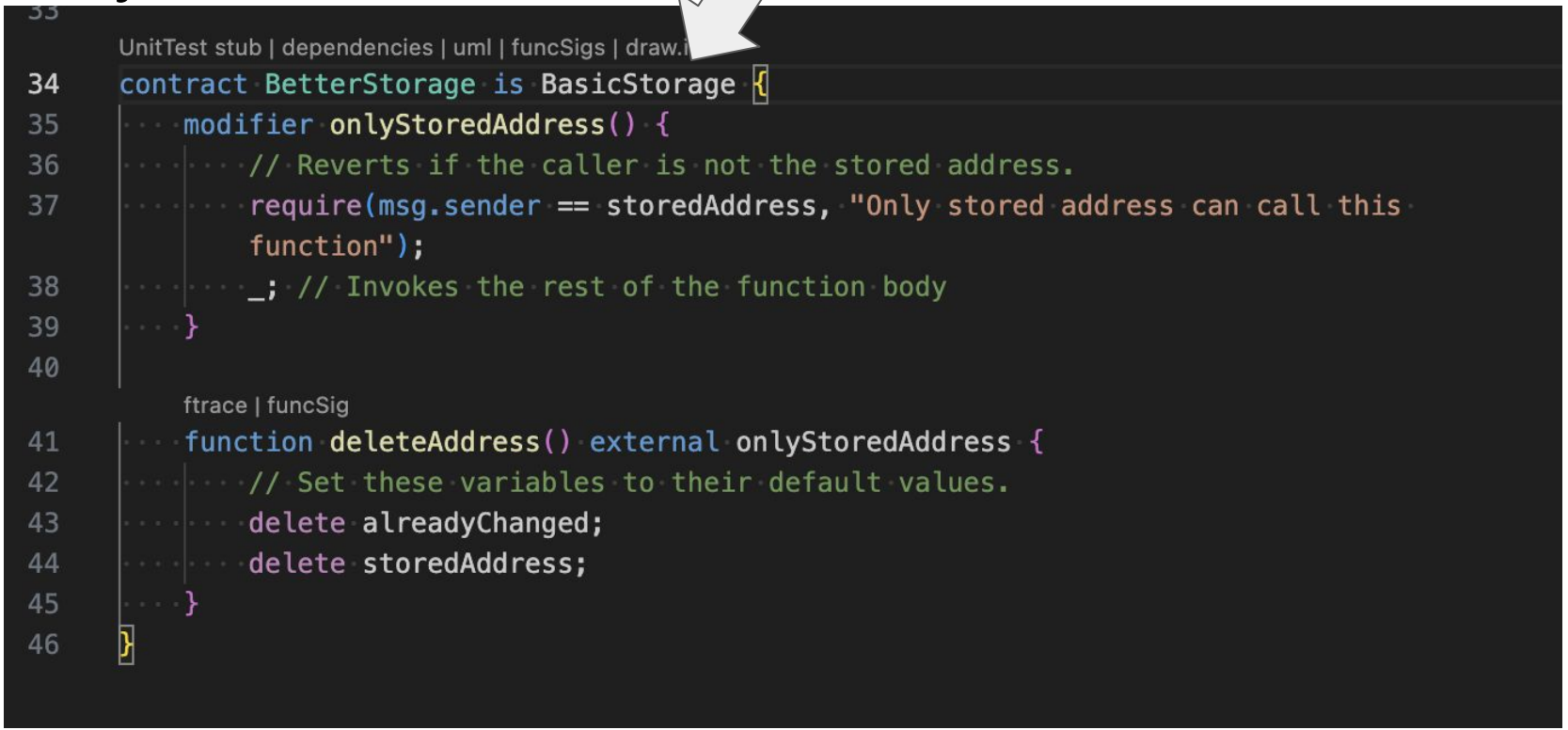
Solidity: Bits and Pieces

```
18      // Public functions are included in the solidity preamble jumptable
19      // Implicitly unpacks the calldata bytes into the arguments.
        ftrace | funcSig
20      function storeAddress(address _addr, string calldata _data) public {
21          // Reverts if already modified in this tx
22          require(!alreadyChanged, "Reentrant call");
23          // Reverts if address is not even.
24          require(addressIsEven(_addr), "Address is not even");
25          // modifies storage.
26          storedAddress = _addr;
27          // locks further modifications in this tx.
28          alreadyChanged = true;
29          // implicitly copies _data from calldata to memory
30          emit AddressStored(_addr, _data);
31      }
```

Solidity: Bits and Pieces

```
18      // Public functions are included in the solidity preamble jumptable
19      // Implicitly unpacks the calldata bytes into the arguments.
        ftrace | funcSig
20      function storeAddress(address _addr, string calldata _data) public {
21          // Reverts if already modified in this tx
22          require(!alreadyChanged, "Reentrant call");
23          // Reverts if address is not even.
24          require(addressIsEven(_addr), "Address is not even");
25          // modifies storage.
26          storedAddress = _addr;
27          // locks further modifications in this tx.
28          alreadyChanged = true;
29          // implicitly copies _data from calldata to memory
30          emit AddressStored(_addr, _data);
31      }
```

Solidity: Bits and Pieces



```
UnitTest stub | dependencies | uml | funcSigs | draw.i
34  contract BetterStorage is BasicStorage {
35      modifier onlyStoredAddress() {
36          // Reverts if the caller is not the stored address.
37          require(msg.sender == storedAddress, "Only stored address can call this
            function");
38          _; // Invokes the rest of the function body
39      }
40
        ftrace | funcSig
41      function deleteAddress() external onlyStoredAddress {
42          // Set these variables to their default values.
43          delete alreadyChanged;
44          delete storedAddress;
45      }
46  }
```

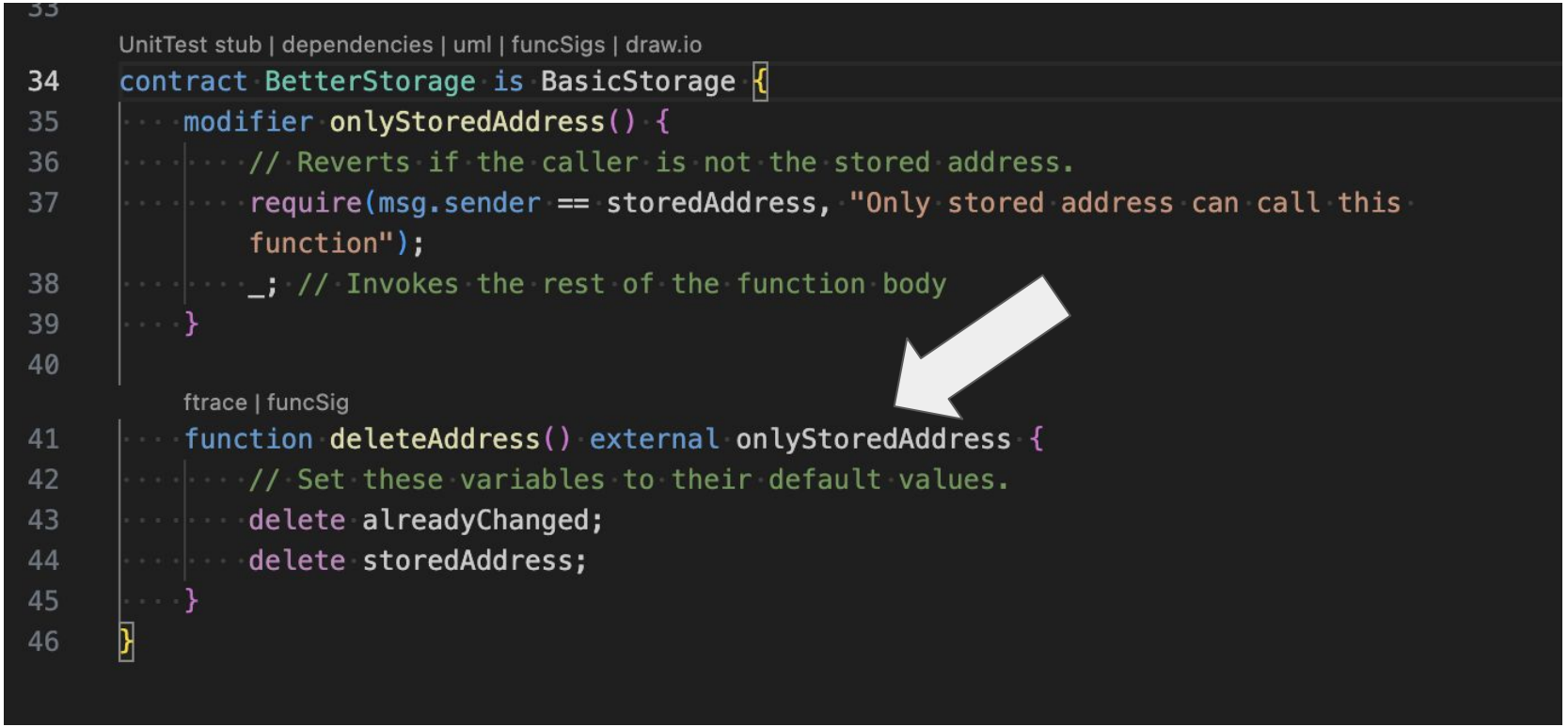
Solidity: Bits and Pieces

```
UnitTest stub | dependencies | uml | funcSigs | draw.io
contract BetterStorage is BasicStorage {
    modifier onlyStoredAddress() {
        // Reverts if the caller is not the stored address.
        require(msg.sender == storedAddress, "Only stored address can call this
        function");
        _; // Invokes the rest of the function body
    }

    ftrace | funcSig
    function deleteAddress() external onlyStoredAddress {
        // Set these variables to their default values.
        delete alreadyChanged;
        delete storedAddress;
    }
}
```

Solidity: Bits and Pieces

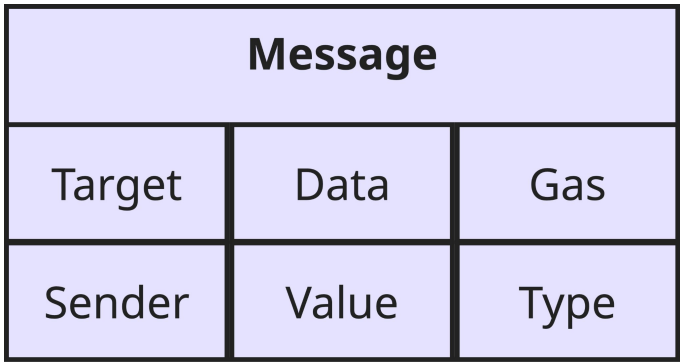
Solidity: Bits and Pieces



Solidity: Bits and Pieces

Solidity: Bits and Pieces

Solidity: Calling Conventions ☠️



Remember this guy?

Solidity: Calling Conventions ☠️

How do we turn this

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

into this

Message		
Target	Data	Gas
Sender	Value	Type

Solidity: Calling Conventions ☠️

1. target - present in state. So we load it.

Solidity: Calling Conventions ☠️

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

1. target - present in state. So we load it.
2. sender - the current contract

Solidity: Calling Conventions ☠️

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

1. target - present in state. So we load it.
2. sender - the current contract
3. value - set to 0 unless specified as follows:

```
betterStorage.storeAddress{value: 1 ether}(_addr⬆, _data⬆);
```

Solidity: Calling Conventions ☠️

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

1. target - present in state. So we load it.
2. sender - the current contract
3. value - set to 0
4. gas - set to 63/64 of the current remaining gas, unless specified as follows:

```
betterStorage.storeAddress{gas: 35_000}(_addr⬆, _data⬆);
```

Solidity: Calling Conventions ☠️

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

1. target - present in state. So we load it.
2. sender - the current contract
3. value - set to 0
4. gas - set to 63/64 of the current remaining gas
5. type - *staticcall* if the target function has the *view* modifier, *call* otherwise

Solidity: Calling Conventions ☠️

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

1. target - present in state. So we load it.
2. sender - the current contract
3. value - set to 0
4. gas - set to 63/64 of the current remaining gas
5. type - staticcall if the target function has the *view* modifier, *call* otherwise
6. data - we're going to talk about ABI later ☠️

Solidity: Calling Conventions

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

Before the call occurs, solidity does some a couple things:

- First, check that the contract in fact does have code.
- Second encode the memory arguments and set up the stack

At this point, the CALL or STATICCALL opcode is invoked, based on the target's interface

Solidity: Calling Conventions

```
betterStorage.storeAddress(_addr⬆, _data⬆);
```

Solidity also does post-call checks.

- Check the status code (pushed to its stack during callframe resolution)
  - If false, copy *returndata* to memory, then revert locally with identical returndata
  - If true, continue
- If the callee interface specifies return types ABI decode the return types, and push them to the stack
  - If decoding fails, revert with empty returndata[1]

1. There's a compiler argument to insert useful messages for these halts. But production contracts don't use it. So this can be hard to debug in the wild.

Solidity: Invocation, Selectors, and the Preamble

```
betterStorage.storeAddress(_addr, _data);
```

How do we communicate to the callee what function we want to invoke?

```
// Public functions are included in the solidity preamble jumptable
// Implicitly unpacks the calldata bytes into the arguments.
ftrace | funcSig
function storeAddress(address _addr, string calldata _data) public {
```

Solidity: Invocation, Selectors, and the Preamble

```
// Public functions are included in the solidity preamble jumptable
// Implicitly unpacks the calldata bytes into the arguments.
ftrace | funcSig
function storeAddress(address _addr, string calldata _data) public {
```

```
functionSignature = “storeAddress(address,string)”
   selector = keccak256(functionSignature)[0:4]
   callData = selector || abiEncode(arguments)
```

Solidity: Invocation, Selectors, and the Preamble

Every deployed Solidity contract begins with a preamble.

The preamble contains a jump table

function	selector	modifiers	Jump location
storeAddress	0x9b622a11	nonpayable	0x03C9
deleteAddress	0x643f8605	nonpayable onlySender	0x0442
storedAddress	0x8f63640e	nonpayable	0x031D

Solidity: Let’s decompile! (with ethervm.io)

```
1
2   // This is the free memory pointer. Solidity uses the memory at 0x40 to store
3   // the next unused location in memory. Think of it as a simple bump allocator.
4   memory[0x40:0x60] = 0x80;
5
6   // Because the contract has no payable functions, we revert if any ether is
7   // sent.
8   var msgValue = msg.value;
9   if (msgValue) { revert(memory[0x00:0x00]); }
10
```

Solidity: Let's decompile a preamble!

```
11    // We have no fallback method, so we revert if there is no data
12    // because there's no function selector.
13    if (msg.data.length < 0x04) { revert(memory[0x00:0x00]); }
```

```
label_000F:
        // Incoming jump from 0x000B, if !msg.value
        // Inputs[1] { @0013  msg.data.length }
        000F    5B  JUMPDEST
        0010    50  POP
        0011    60  PUSH1 0x04
        0013    36  CALLDATASIZE
        0014    10  LT
        0015    61  PUSH2 0x003f
        0018    57  *JUMPI
        // Stack delta = -1
        // Block ends with conditional jump to 0x003f, if msg.data.length < 0x04
```

Solidity: Let’s decompile a preamble!

```
15    // Extract the function selector from the calldata
16    var selector = msg.data[0x00:0x20] >> 0xe0;
```

```
label_0019:
        // Incoming jump from 0x0018, if not msg.data.length < 0x04
        // Inputs[1] { @001A  msg.data[0x00:0x20] }
        0019    5F  PUSH0
        001A    35  CALLDATALOAD
        001B    60  PUSH1 0xe0
        001D    1C  SHR
```

Solidity: Let’s decompile a preamble!

```
if (selector == 0x643f8605) {
    // Dispatch table entry for deleteAddress()
    // Ommitted
```

```
001E     80  DUP1
001F     63  PUSH4 0x643f8605
0024     14  EQ
0025     61  PUSH2 0x0043
0028     57  *JUMPI
```

Solidity: Let’s decompile a preamble!

```
} else if (selector == 0x8f63640e) {
        // Dispatch table entry for storedAddress()
```

```
0029     80   DUP1
002A     63   PUSH4 0x8f63640e
002F     14   EQ
0030     61   PUSH2 0x004d
0033     57   *JUMPI
```

Solidity: Let's decompile a preamble!

```
} else if (selector == 0x8f63640e) {
        // Dispatch table entry for storedAddress()
        storedAddress = storage[0x00] &
        0xffffffffffffffffffffffffffffffffffffffff;
        alloc = memory[0x40:0x60];
        var2 = encodeToMemory(storedAddress, alloc);
        var freeMem = memory[0x40:0x60];
        return memory[freeMem:freeMem + var2 - freeMem];
} else if (selector == 0x9b622a11) {
        // Dispatch table entry for storeAddress(address,string)
        // Omitted
} else { revert(memory[0x00:0x00]); }
```

Solidity: Invocation, Selectors, and the Preamble

```
functionSignature = “storeAddress(address,string)”
   selector = keccak256(functionSignature)[0:4]
   callData = selector || abiEncode(arguments)
```

1. Solidity uses function signatures to handle incoming messages[1].
2. The preamble jump table invokes specific functions
3. The compiler handles encoding arguments and decoding returns for you.
4. The high-level structures all have direct, obvious low-level instantiations in bytecode.

[1]As a consequence, function signatures must be unique.

Solidity: Application Binary Interface (ABI) Encoding

The slides I never wanted to make

```
function transfer(address to, uint256 amount) public returns (bool) {
```

In solidity, this line of code specifies two coding routines.

1. The argument coder.
2. The return coder.

These coders follow a scheme called ABI.

Solidity: Application Binary Interface (ABI) Encoding

```
function transfer(address to, uint256 amount) public returns (bool) {
```

The argument coder's job is to handle conversion of `(address,uint256)` to and from a string of bytes.

The return coder does the same for `bool`

Solidity: Application Binary Interface (ABI) Encoding

This is where everything gets fidgety.

Solidity ABI is optimized for the EVM. This means:

1. It operates on 32-byte words. All types are encoded to a minimum of 1 word
    a. Yes, even booleans.
2. Encoding is recursive
    a. Contract bytecode sizes are tightly limited.
3. Encoding operates on Solidity types
    a. (or reasonable representations of them)

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

ABI sorts types into fixed-length (“static”) and variable-length (“dynamic”)

Static types	uint256	bool	**static**[N]	(**static**, **static**)
Dynamic types	string	bytes	**T**[ ] or **dynamic**[N]	(**dynamic**, ..)

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

For each type, we define an encoding, that outputs “head” and “tail” elements. Static types NEVER output tails. Dynamic types ALWAYS output tails and MAY output heads.

```
enc(T) -> (heads, tails)
```

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

The encoding of a series, like a tuple is:

1. All its component heads, concatenated
2. All its component tails, concatenated

$$\text{enc}((T_1, .., T_k)) \rightarrow ((\text{heads}_1 \,||\, .. \,||\, \text{heads}_k), \text{tails}_1 \,||\, .. \,||\, \text{tails}_k))$$

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

Clear as mud, right?

$$\text{enc}((T_1, .., T_k)) \rightarrow ((\text{heads}_1 \,||\, .. \,||\, \text{heads}_k), \text{tails}_1 \,||\, .. \,||\, \text{tails}_k))$$

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

This is an ABI encoded blob of the form `(string,)`:

```
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000024
556e69737617056323a20494e535546464934945 4e545f494e5055545f414d
4f554e540000000000000000000000000000000000000000000000000000000
```

It's string payload is "`UniswapV2: INSUFFICIENT_INPUT_AMOUNT`"

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000024
556e69737617056323a20494e5355464649349454e545f494e5055545f414d
4f554e540000000000000000000000000000000000000000000000000000000
```

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000024
556e69737617056323a20494e53554646494349454e545f494e5055545f414d
4f554e540000000000000000000000000000000000000000000000000000000
```

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000024
556e6973776170563a20494e5355464649349454e545f494e5055545f414d
4f554e540000000000000000000000000000000000000000000000000000000
```

Solidity: Application Binary Interface (ABI) Encoding

Read this instead of these slides:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000024
556e69737617056323a20494e53554646494349454e545f494e5055545f414d
4f554e54000000000000000000000000000000000000000000000000000000000
```

Solidity: Application Binary Interface (ABI) Encoding

Read this:
https://docs.soliditylang.org/en/latest/abi-spec.html

So now that we've talked about this, promise me you'll never implement it.

Solidity: Application Binary Interface (ABI) Encoding

Read this:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
function transfer(address to, uint256 amount) public returns (bool) {
```

So now that we've covered this, promise me you'll never implement it.

Soldity does this for you.

Solidity: Application Binary Interface (ABI) Encoding

Read this:
https://docs.soliditylang.org/en/latest/abi-spec.html

```
function transfer(address to, uint256 amount) public returns (bool) {
```

So now that we've covered this, promise me you'll never implement it.

Soldity does this for you.

In Rust, you can use my work :)

Solidity: Review!

Solidity is

1. A set of calling conventions
    a. Encoding!
    b. Selectors!
2. A compiled high-level language
    a. Functions!
    b. Constructors!
    c. Contracts!
    d. Storage!

EVM Engineering

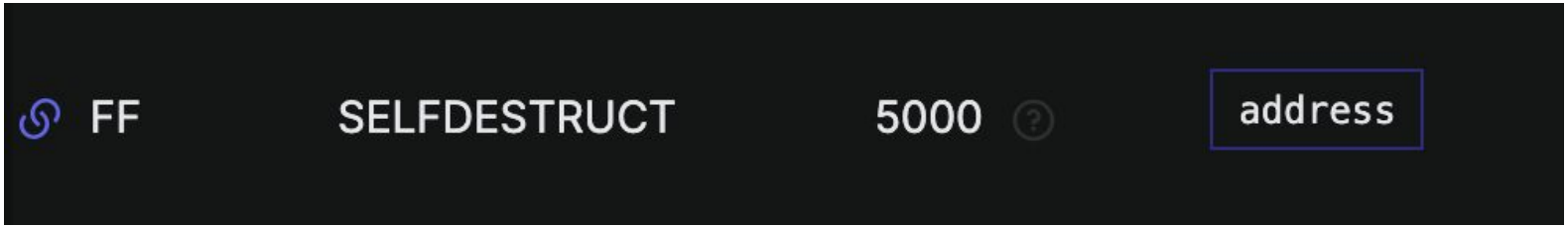
- ~~Transactions~~

- ~~Solidity & its standards~~

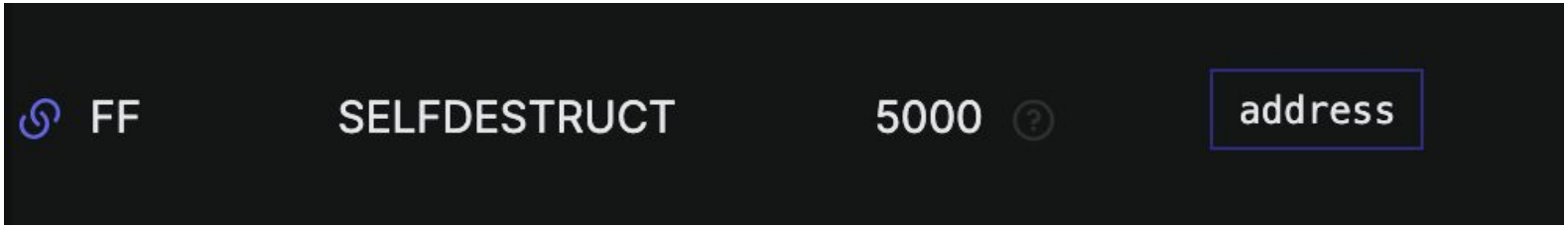
- Bonus round: Code mutability.

EVM Engineering: Mutation

This is a roundabout way of covering EIP-7702 as well



EVM Engineering: Mutation



Halt execution and register account for later deletion or send all Ether to address (post-Cancun)

EVM Engineering: Mutation

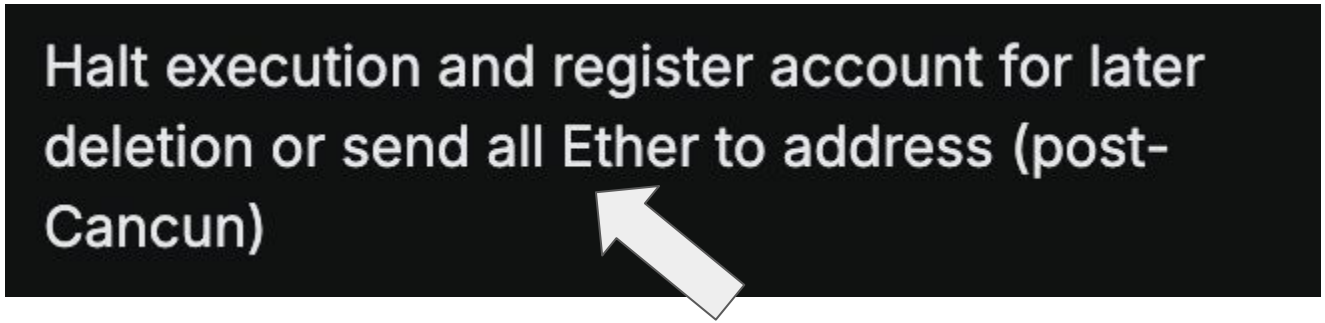
Self destruct **used to** cause code to be deleted.

> Halt execution and register account for later deletion or send all Ether to address (post-Cancun)

This would allow you to deploy new code at the same address.

EVM Engineering: Mutation

Self destruct **used to** cause code to be deleted.



Now, it just moves ether. There is currently **no way to delete or modify code[1].**

1. Except via 7702

EVM Engineering: Mutation

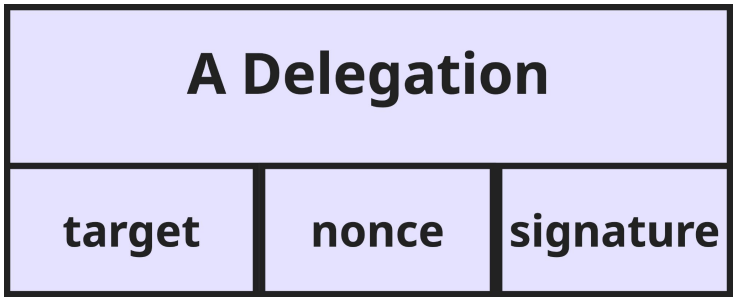
When you deploy a contract it is forever.

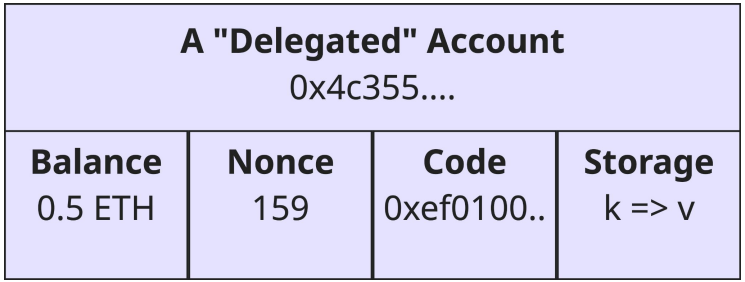
EVM Engineering: Mutation via 7702

When we make a 7702 transaction, any valid delegations it contains **MODIFY** the code of their signer.

They insert 0xef0100….. into the code, overriding anything already there.

This is weird.



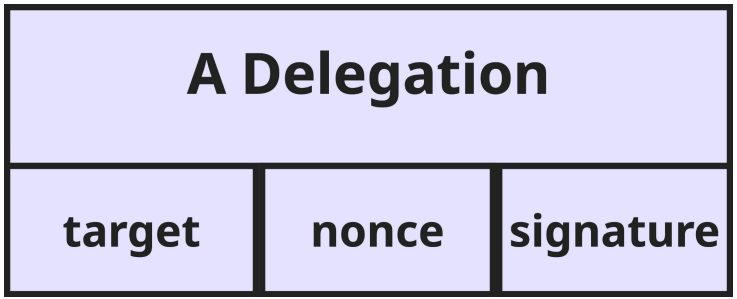


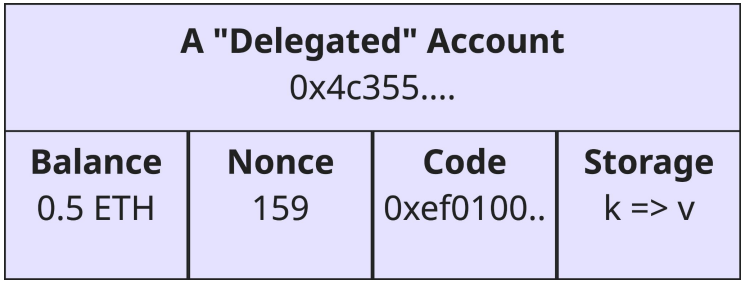
EVM Engineering: Mutation via 7702

When we make a 7702 transaction, any valid delegations it contains **MODIFY** the code of their signer.

They insert 0xef0100….. into the code, overriding anything already there.

This is weird. **And it gets weirder!**

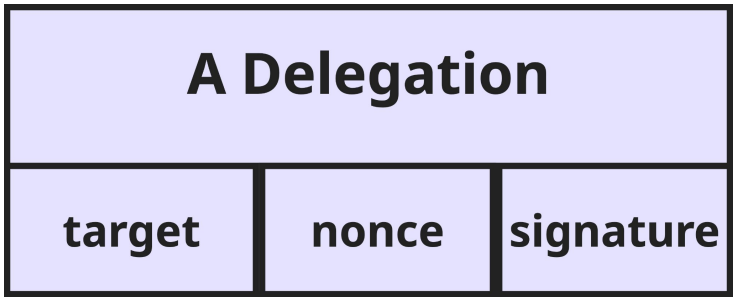


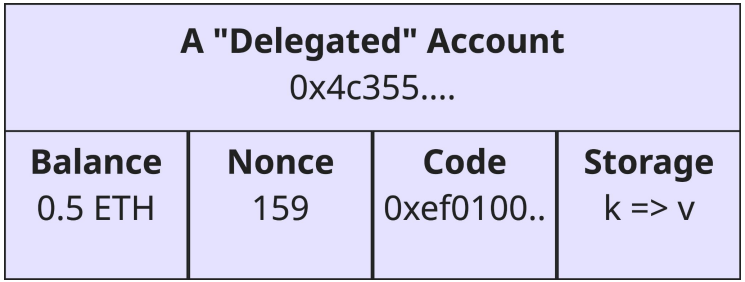


EVM Engineering: Mutation via 7702

The **target** address is preprended with the magic bytes `0xef0100` and then included in the code of the signer's account.

Because a delegation requires a signature, only EOAs can delegate

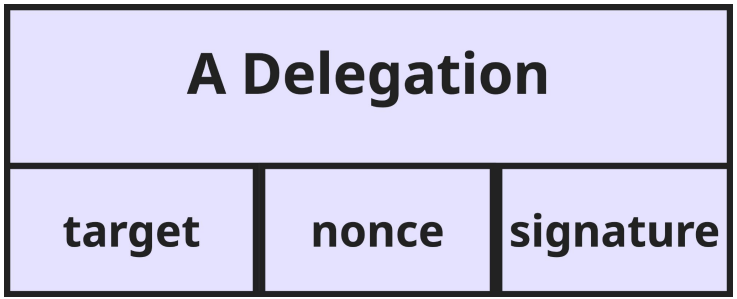




EVM Engineering: Mutation via 7702

From now on, when the EOA receives a message, the code at **target** is loaded and executed.

In this way, the code and behavior of the EOA can change over time.

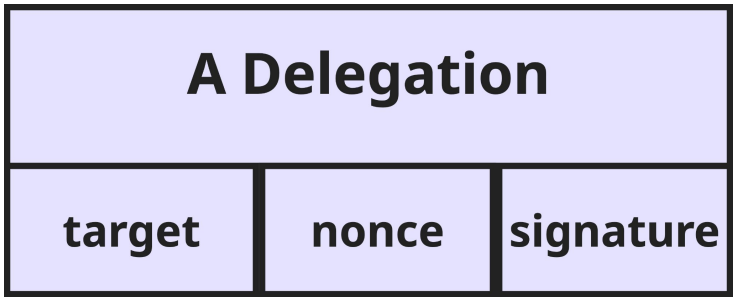


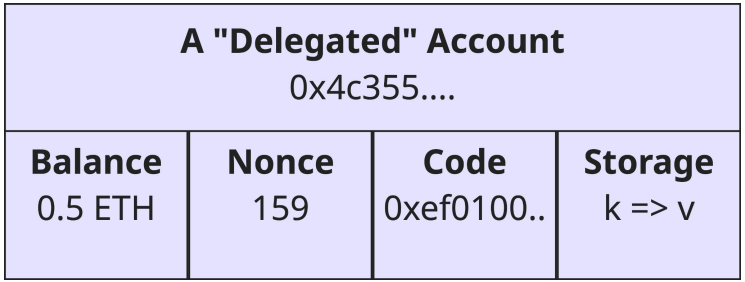
A "Delegated" Account 0x4c355....			
Balance 0.5 ETH	**Nonce** 159	**Code** 0xef0100..	**Storage** k => v

EVM Engineering: Mutation teaser :)

Contract code is immutable.

But its behavior can change anyway.





Questions and Break

Upgradeability: Necessary Evils

- We’ve talked about immutability.
  - Bytecode, once it’s deployed to the EVM, never changes.
- Soooo… How do you build production software?
- User needs change, product and business needs change.

Upgradeability

- Who integrates with contracts? → Other contracts, often.
  - Which are also immutable 🙃
  - And which you do not control the migration/upgrade strategies of.
- If your contract changes underneath them, you can break external partners' contracts.
- So again – how do you ship new features, improve your product offering, respond to changing market needs…?

Upgradeability

- Well. You can do it, but – It's hard.
- This kinda sucks
- In **traditional software engineering** you can launch a product, learn from it, and iterate based on those learnings
  - Breaking things is cheap. Fixing things is easy.
- With **smart contract development**, you have to think pretty hard *upfront* in the *initial* development stage, before you've had real market feedback.
  - Your initial design often needs to consider not only how your product is being used today, but how it *might* be used in the future.
  - Breaking things is expensive, and fixing things may not be possible

Upgradeability

- Alright, so it’s hard, but how do you actually do it?
- Let’s look at some options →

Option 1: Migration

- What:
  - write an **entirely new contract** and **deploy it separately** from the old one
- Downsides –
  - New product won’t have your existing TVL/users
  - Need to encourage your user base to migrate over
  - Need to start growth flywheels from scratch
  - The old contracts don’t go away.
  - You have to continue to monitor, support, and protect them from vulnerabilities, respond to incidents, etc.
- Upsides
  - Vibes-wise, it’s safer because it’s more “True” to the append-only reality of the environment

Option 2: Upgradeability

- What: use some **special tricks** to **change functionality in-place** 🎃
- Huh? You just said bytecode doesn't change
- How
  - Basic concept: proxy + implementation
  - **Proxy** is the "front" for the Implementation – the **public-facing address** that users interact with directly
  - **Implementation** contains the actual business logic.
  - Proxy uses *delegatecall* to execute the bytecode of the Implementation
  - Implementation address is **configurable** – change it & change the logic of the Proxy, essentially Upgrading it

Option 2: Upgradeability

- Upsides
  - Keep the same address
  - Keep all your TVL and users automatically
  - Add new functionality OR Update/improve existing functionality behind the scenes
  - Launch fast, then build out missing portions of the protocol later (Common approach)

Option 2: Upgradeability

- Downsides
  - You gotta think **hard** about how modifying behavior in-place will affect integrators
  - Existing External interfaces MUST NOT change, or contracts that integrate begin to revert automatically
    - (remember function selectors?).
    - If new functionality needs new/different params, you need to add new external interfaces.
  - Upgradeability contains allllllllll kinds of footguns to introducing vulnerabilities into *your own* protocol

Option 2: Upgradeability – Footguns

- There are so many footguns they needed their own slide.
- These have led to literally billions of dollars in hacks
- Borking storage
  - Messing up storage by re-ordering variable declarations (especially tricky with contract inheritance)
  - Unsafely re-writing the rules of how certain storage slots are used
- *Selfdestruct*-ing the implementation or the proxy and blowing the whole thing up <3
  - Google “I accidentally killed it”
- Not initializing proxies OR not initializing implementations, allowing takeover of either contract by anyone and everyone

Option 2: Upgradeability – Patterns

- Proxy-controlled Upgrades (standard):
  - Upgrade logic is written into the proxy.
  - Problem: storage collision. A careless implementation can accidentally overwrite the upgrade-related storage in the proxy. This bricks the contract. 🧱
- Implementation-controlled Upgrades (also called UUPS):
  - Upgrade logic is written into the implementation.
  - Storage collision is fixed bc the implementation contains *both* upgrade and core logic
  - Problem: A careless implementation can mess up the upgrade logic. This bricks the upgradeability, and maybe the entire contract. 🧱

Option 2: Upgradeability – Patterns

- Upgrade Beacon:
  - Upgrade logic is written into a secret third thing, an external contract called the Upgrade Beacon
  - Proxy calls Beacon to get Implementation address, then delegates to it
  - This is really, really safe from storage collision
  - The upgrade logic is all immutable, so bad Implementations can’t really brick it
  - ALSO, this allows one weird trick™ – you can have hundreds of proxies point to the SAME single UpgradeBeacon, and thus perform upgrades to hundreds of contracts in one fell sweep
  - Example: smart wallets all use the same logic, but need one per user

Contract Standards: the ERC process

Ethereum Request for Comment (ERC) documents are community-driven standards for Ethereum software.

Some cool/broadly-used smart contract ERCs:

- ERC-20 - Assets
- ERC-712 - SignedData
- ERC-721 (yes this is confusing) - NFTs
- ERC-2612 - Permit extension for ERC-20

Check out the full list here: https://eips.ethereum.org/erc

Contract Standards: ERC-20

```solidity
/// @notice Returns the amount of tokens in existence.
function totalSupply() external view returns (uint256);

/// @notice Returns the amount of tokens owned by `account`.
function balanceOf(address account) external view returns (uint256);

/// @notice Moves `amount` tokens from the caller's account to `to`.
function transfer(address to, uint256 amount) external returns (bool);

/// @notice Returns the remaining number of tokens that `spender` is allowed
/// to spend on behalf of `owner`
function allowance(address owner, address spender) external view returns (uint256);

/// @notice Sets `amount` as the allowance of `spender` over the caller's tokens.
/// @dev Be aware of front-running risks: https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
function approve(address spender, uint256 amount) external returns (bool);

/// @notice Moves `amount` tokens from `from` to `to` using the allowance mechanism.
/// `amount` is then deducted from the caller's allowance.
function transferFrom(address from, address to, uint256 amount) external returns (bool);
```

Contract Standards: ERC-20

```
/// @notice Moves `amount` tokens from the caller's account to `to`.
function transfer(address to, uint256 amount) external returns (bool);
```

This is the standard

```
function transfer(address _to, uint _value) public whenNotPaused;
```

This is Tether (USDT, about $101,000,000,000).

Spot the problem.

Contract Standards: A broken ERC-20!

```
/// @notice Moves `amount` tokens from the caller's account to `to`.
function transfer(address to, uint256 amount) external returns (bool);
```

This is the standard

```
function transfer(address _to, uint _value) public whenNotPaused;
```

This is Tether (USDT)

The missing bool return is treated as false by solidity callers. This is often interpreted as a failure. Yikes.

Contract Standards: ERC-20

Basically every token on Ethereum is an ERC-20.

Contract Standards: ERC-721

```
/// @notice Count all NFTs assigned to an owner
function balanceOf(address _owner) external view returns (uint256);

/// @notice Find the owner of an NFT
function ownerOf(uint256 _tokenId) external view returns (address);

/// @notice Transfer ownership of an NFT
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;

/// @notice Change or reaffirm the approved address for an NFT
function approve(address _approved, uint256 _tokenId) external payable;

/// @notice Enable or disable approval for a third party ("operator") to manage
/// all of `msg.sender`'s assets
function setApprovalForAll(address _operator, bool _approved) external;

/// @notice Get the approved address for a single NFT
function getApproved(uint256 _tokenId) external view returns (address);

/// @notice Query if an address is an authorized operator for another address
function isApprovedForAll(address _owner, address _operator) external view returns (bool);
```

Contract Standards: ERC-721

Please do not ask me questions about ERC-721 assets

Contract Standards: ERC-712

A standard for signing structured data, intended for use in Solidity.

1. Embed actions from Alice in a transaction from Bob.
2. Used by wallets to report to users what they're signing

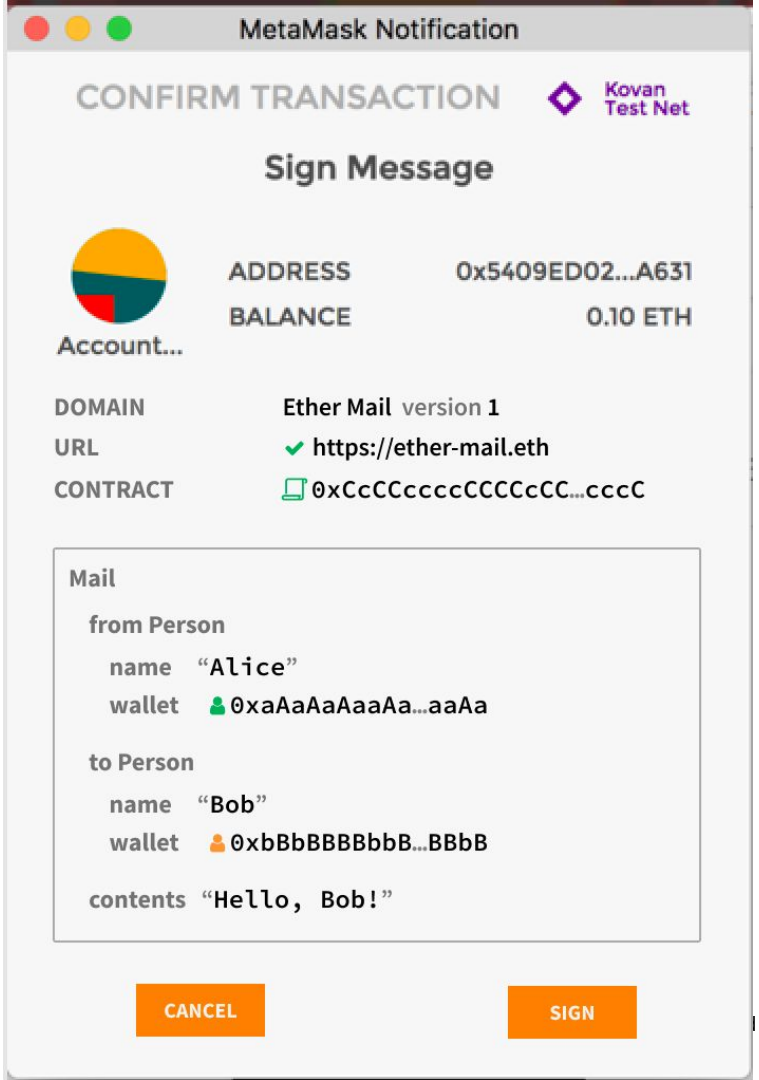


Image:
https://eips.ethereum.org/EIPS/eip-712
(CC0 1.0)

Contract Standards: ERC-2612

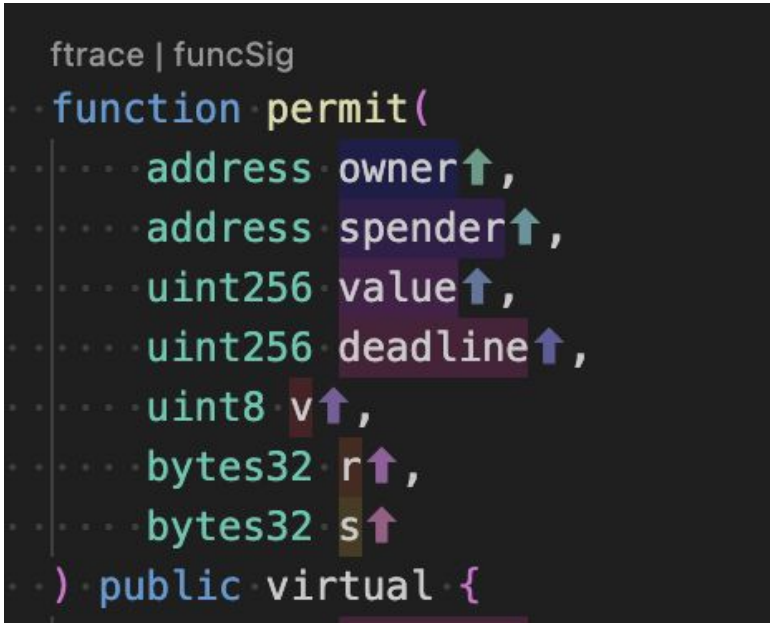
An application of ERC-712 to ERC-20:

The ERC-20 approve function requires a transaction from Alice.

The ERC-2612 extension does NOT require Alice to make a transaction.

Instead, she can sign a permit instruction off-chain using EIP-712.

Neat :)



Contract Standards

There are many more contract standards.

Check out https://eips.ethereum.org

Contracts in the Wild

- ~~Contract Standards~~

- Decentralized Finance (DeFi)

- Interfaces with Real Life

DeFi

- Automated Market Makers
- Staking and Liquid Staking
- Lending

DeFi: AMMs

AMMs are smart contracts that hold assets, and trade automatically.

The simplest is the constant product (or xyk) AMM. Which sets a constant k, as the product of its balances:

$$k = x * y$$

When you send it asset x, it attempts to keep k constant by sending you asset y:

$$k = (x + \Delta_x) * (y - \Delta_y)$$

Feel free to solve for Δ_y if you want to :)

Defi: Staking + Liquid Staking

Staked assets are illiquid in PoS. But it is possible to tokenize the rights.

This creates an asset with a cashflow.

But who manages the stakers? And how?

Defi: Lending

Lending markets track the Loan-to-value ration “LTV” of a position.

$$LTV = (y * price_y) / (x * price_x)$$

The borrower must keep LTV above some safe value, e.g. 1.5x, otherwise a forced sale of collateral called “liquidation” occurs. This is bad.

Defi: Lending

You may notice we now need accurate price information.

Where does that come from?

Contracts in the Wild

- ~~Contract Standards~~
- ~~Decentralized Finance (DeFi)~~
- Interfaces with Real Life

Real Life

- Oracles
- Stablecoins and Real-world Assets (RWAs)
- Maker and DAI
- Governance

Real Life: Oracles

Remember how we needed price info for borrowing earlier?

$$LTV = (y * price_y) / (x * price_x)$$

An “oracle” is a source of information that cannot be verified by the chain.

- Price feeds
- Prediction market outcomes
- Any real-world data at all.

Real Life: Stablecoins and Real-world Assets (RWAs)

Can’t we just make an Oracle for my bank balance? Then my dollars could be tokens.

Yes we can. This is called a stablecoin. The non-currency version is called an RWA. Stablecoins predate Ethereum, and were launched on Bitcoin first via Mastercoin

Real Life: Maker and DAI

Maker is a combination stablecoin and lending market.

When you open a maker vault, you add collateral, and borrow “DAI”

$$LTV = (y * \$1) / (x * price_x)$$

$DAI is a token that’s worth $1[1]

Like lending markets, your position can be liquidated if the LTV falls too far

1. There are so many caveats here. Go read about flip flap & flop, and the cat vat. This footnote is not a joke.

Real Life: Governance and DAOs

How does one administer a smart contract? Who makes the decisions?

Real Life: Governance and DAOs

1. An administrator

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”
    a. Token Voting

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”
    a. Token Voting
    b. A council

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”
    a. Token Voting
    b. A council
    c. Multipartite

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”
    a. Token Voting
    b. A council
    c. Multipartite
    d. Hybrid

Real Life: Governance and DAOs

1. An administrator

2. A “DAO”
    a. Token Voting
    b. A council
    c. Multipartite
    d. Hybrid
    e. Give control to Vitalik?

Things I wanted to get to but can’t cover in depth

- Rollups, L2s, Bridges, etc
- Tradeoffs of programmability
- Closing Thoughts

Rollups, L2s, Bridges, etc

- A Layer2 is an off-chain system that purchases some security guarantee from a chain. E.g. payment channels purchase settlement guarantees.

Rollups, L2s, Bridges, etc

- A Layer2 is an off-chain system that purchases some security guarantee from a chain. E.g. payment channels purchase settlement guarantees.

- A Rollup is a long-lived L2 that purchases data availability from the host chain. It is effectively an extension of the host consensus.

Rollups, L2s, Bridges, etc

- A Layer 2 is an off-chain system that purchases some security guarantee from a chain. E.g. state channels purchase settlement guarantees.

- A Rollup is a long-lived L2 that purchases data availability from the host chain. It is effectively an extension of the host consensus. Rollups are usually chains in their own right

- A Bridge is a system that inspects a remote chain’s state. E.g. a fault- or zk-proof bridge inspects a rollup’s state.

Things I wanted to get to but can’t cover in depth

- ~~Rollups, L2s, Bridges, etc~~

- Tradeoffs of programmability

- Closing Thoughts

Some Tradeoffs of Programmability

1. **Predictability**: You do not know what a smart contract will do until you run it. Even if you simulate it, the state it reads may change before your transaction.

Some Tradeoffs of Programmability

1. **Predictability**: You do not know what a smart contract will do until you run it. Even if you simulate it, the state it reads may change before your transaction.

2. **Complexity**: The EVM is far far far more complex than Bitcoin Script. Many security issues arise due to imperfect human understanding of the system.

Some Tradeoffs of Programmability

1. **Predictability**: You do not know what a smart contract will do until you run it. Even if you simulate it, the state it reads may change before your transaction.

2. **Complexity**: The EVM is far far far more complex than Bitcoin Script. Many security issues arise due to imperfect human understanding of the system.

3. **Administration**: Complex systems require maintenance. Gas schedules, new opcodes, EIPs, etc. Programmable chains are updated more often.

Things I wanted to get to but can’t cover in depth

- ~~Rollups, L2s, Bridges, etc~~
- ~~Tradeoffs of programmability~~
- Closing Thoughts

Closing Thoughts

1. If you come out of this thinking you understand the EVM, I’ve done a bad job.

Closing Thoughts

1. If you come out of this thinking you understand the EVM, I’ve done a bad job.

2. Billions of dollars run through these systems every day.

Closing Thoughts

1. If you come out of this thinking you understand the EVM, I've done a bad job.

2. Billions of dollars run through these systems every day.

3. You are capable of this, and there's plenty of work to go around.