Decentralized Finance

Introduction to Smart Contracts

Instructor: Dan Boneh, Arthur Gervais, **Andrew Miller,** Christine Parlour, Dawn Song

Stanford University

Imperial College London

University of Illinois Urbana-Champaign

Berkeley University of California

Outline

- ***What are smart contracts?***

  They're neither smart nor contracts!

  Developer's perspective: Program objects on the blockchain

- ***Basics of Solidity programming in Ethereum***

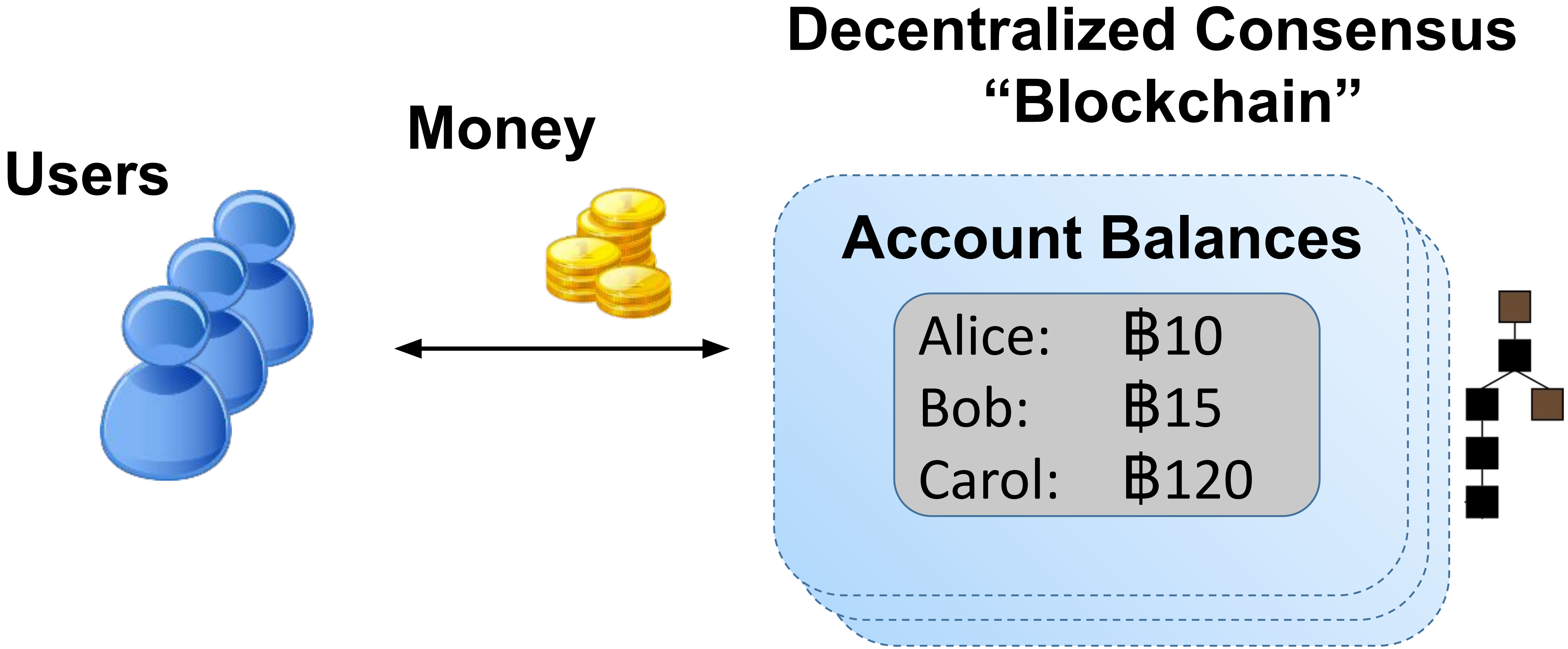
  Just enough to follow the DeFi examples later

- ***Case Study: The Dutch Auction from CryptoKitties***

- ***Comparing Legal Contracts and Smart Contracts***

Part 1: Smart Contracts from Programmer Perspective

Digital currencies: just one blockchain application



Users

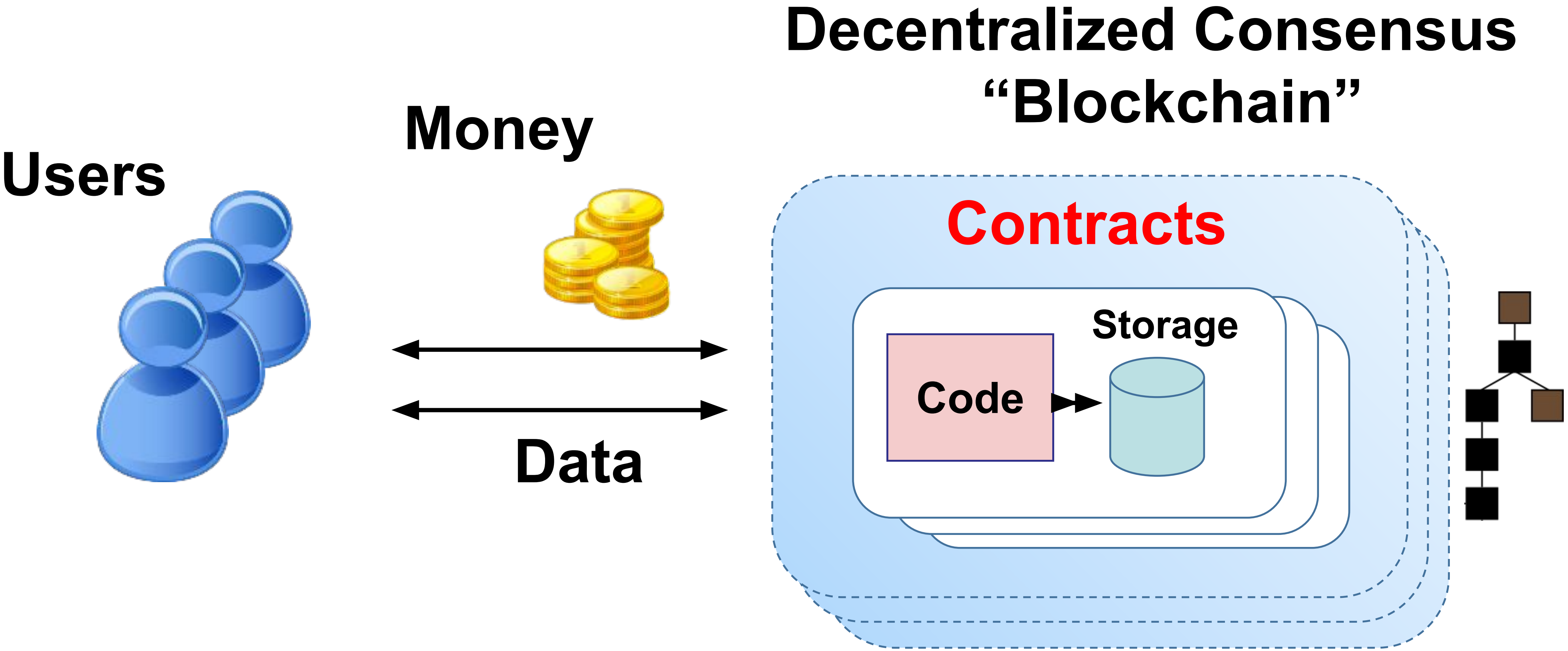
Money

Decentralized Consensus
“Blockchain”

Account Balances

Alice:	฿10
Bob:	฿15
Carol:	฿120

Smart Contracts: user-defined programs running on top of a blockchain



Decentralized Consensus
“Blockchain”

Users

Money

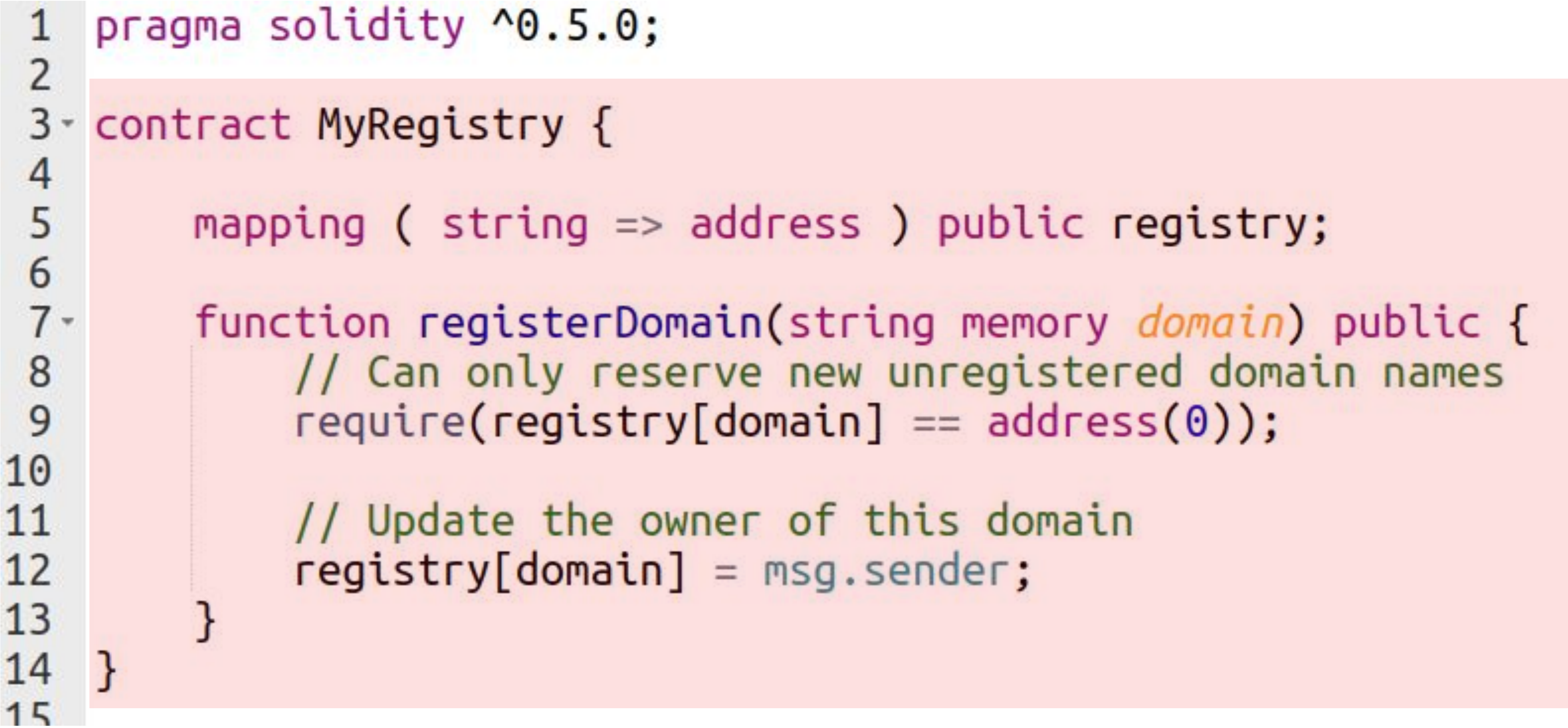
Data

Contracts

Storage

Code

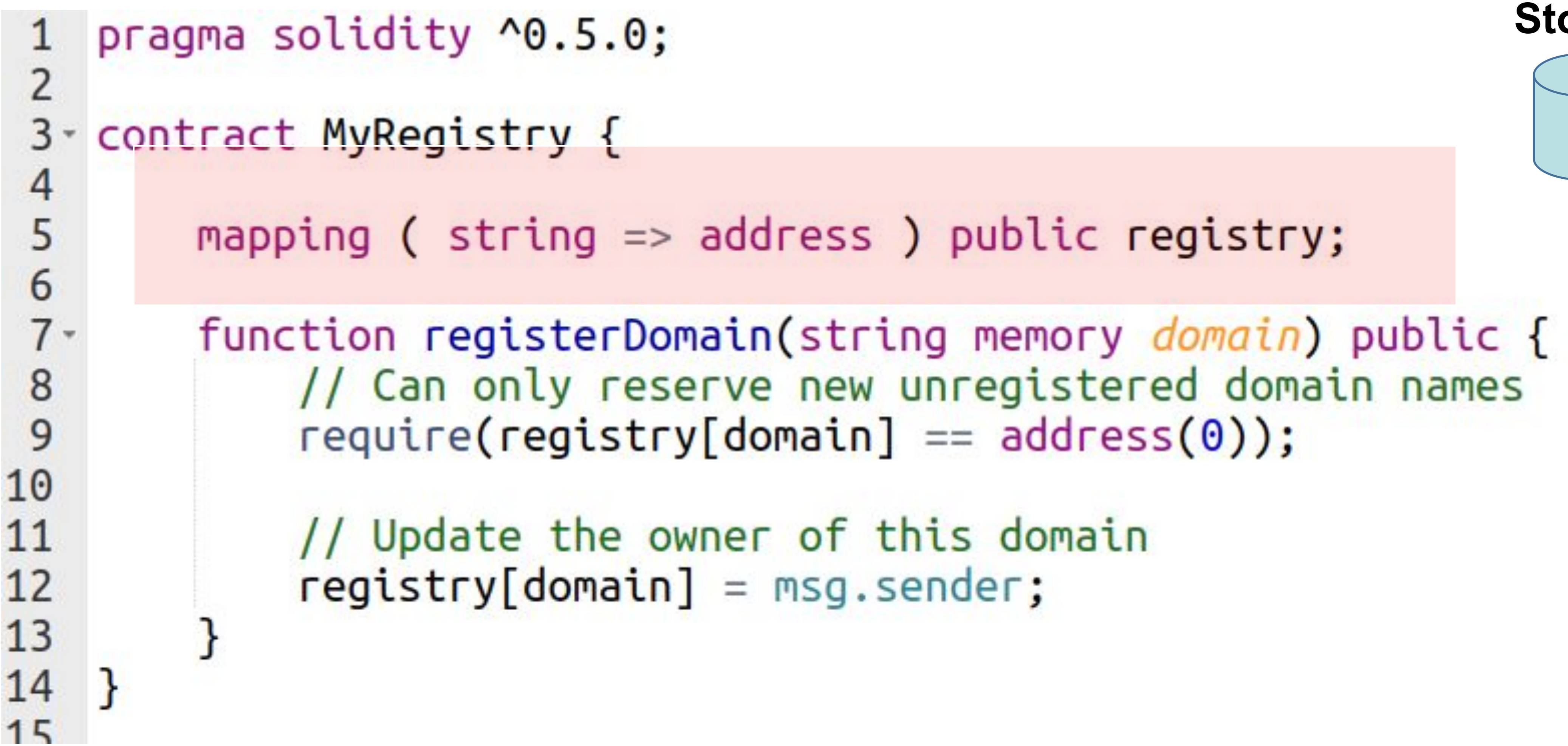
Example: Domain Name Registry in Ethereum



```solidity
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

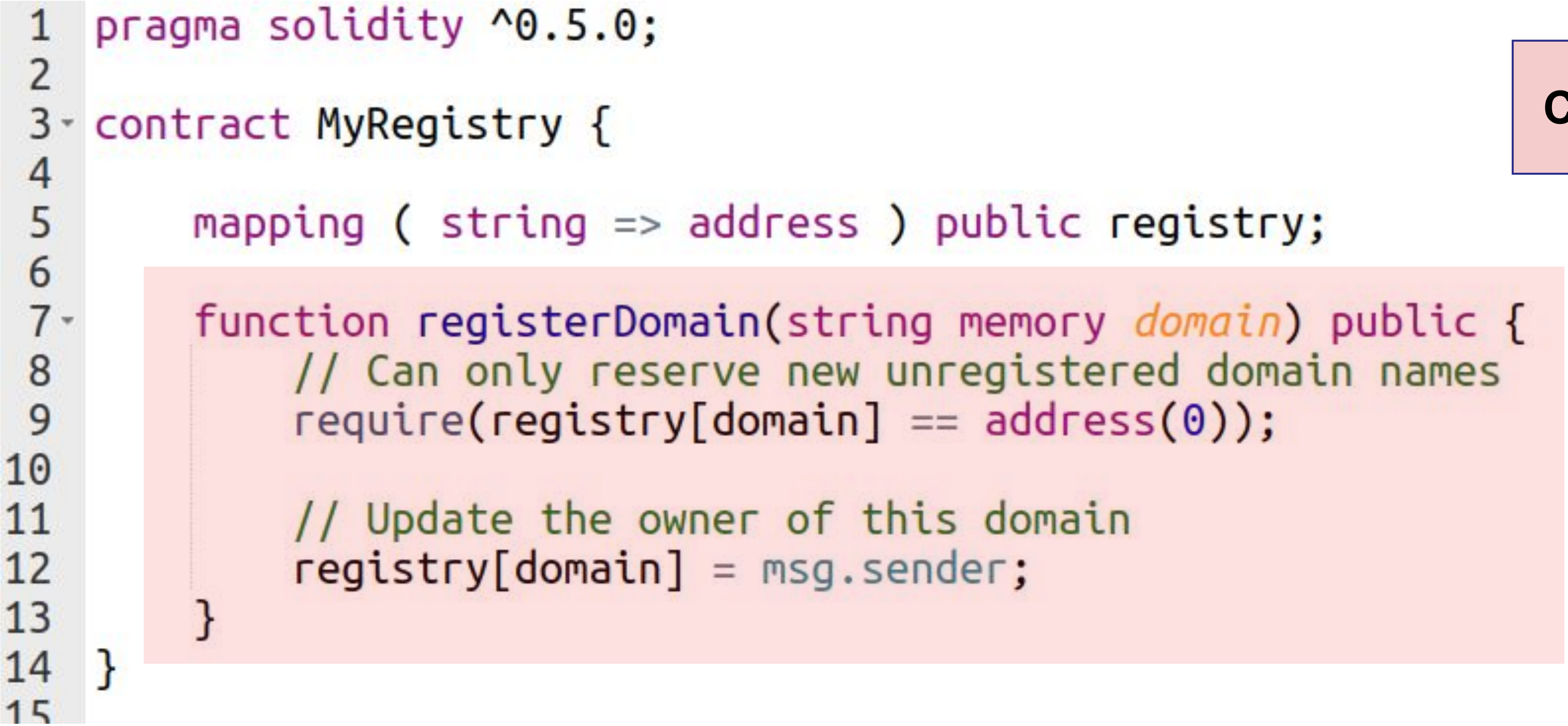
Example: Domain Name Registry in Ethereum

Storage



```
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

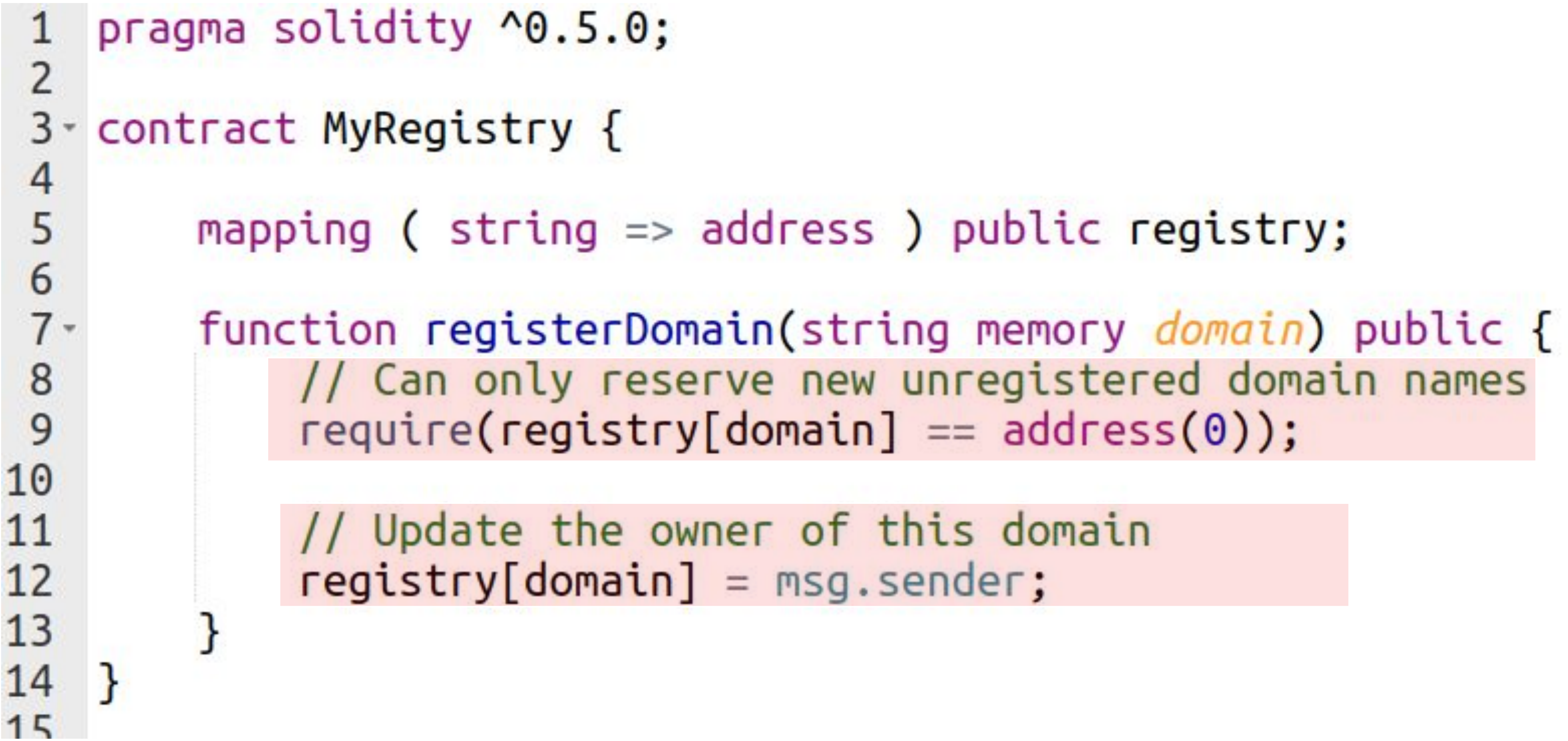
Example: Domain Name Registry in Ethereum



```
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

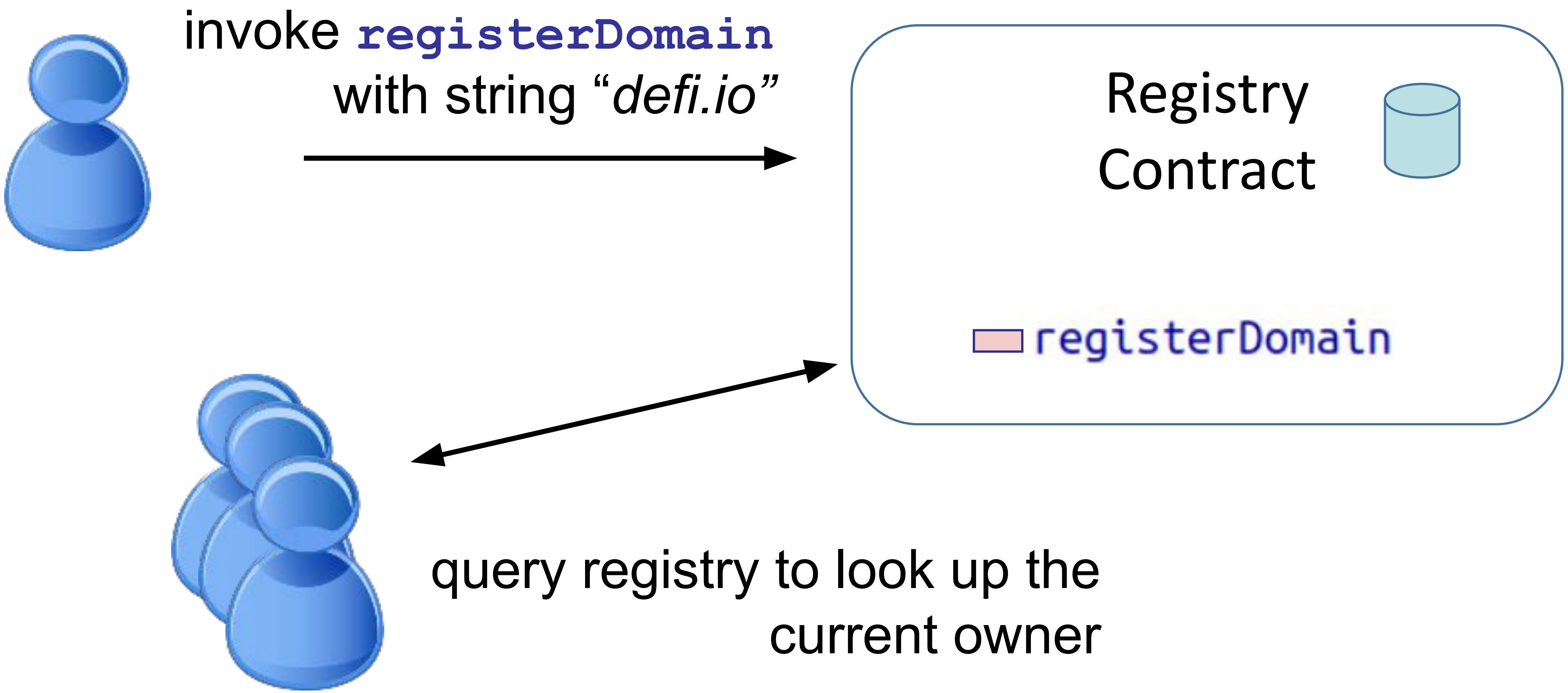
Code

Example: Domain Name Registry in Ethereum



```
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

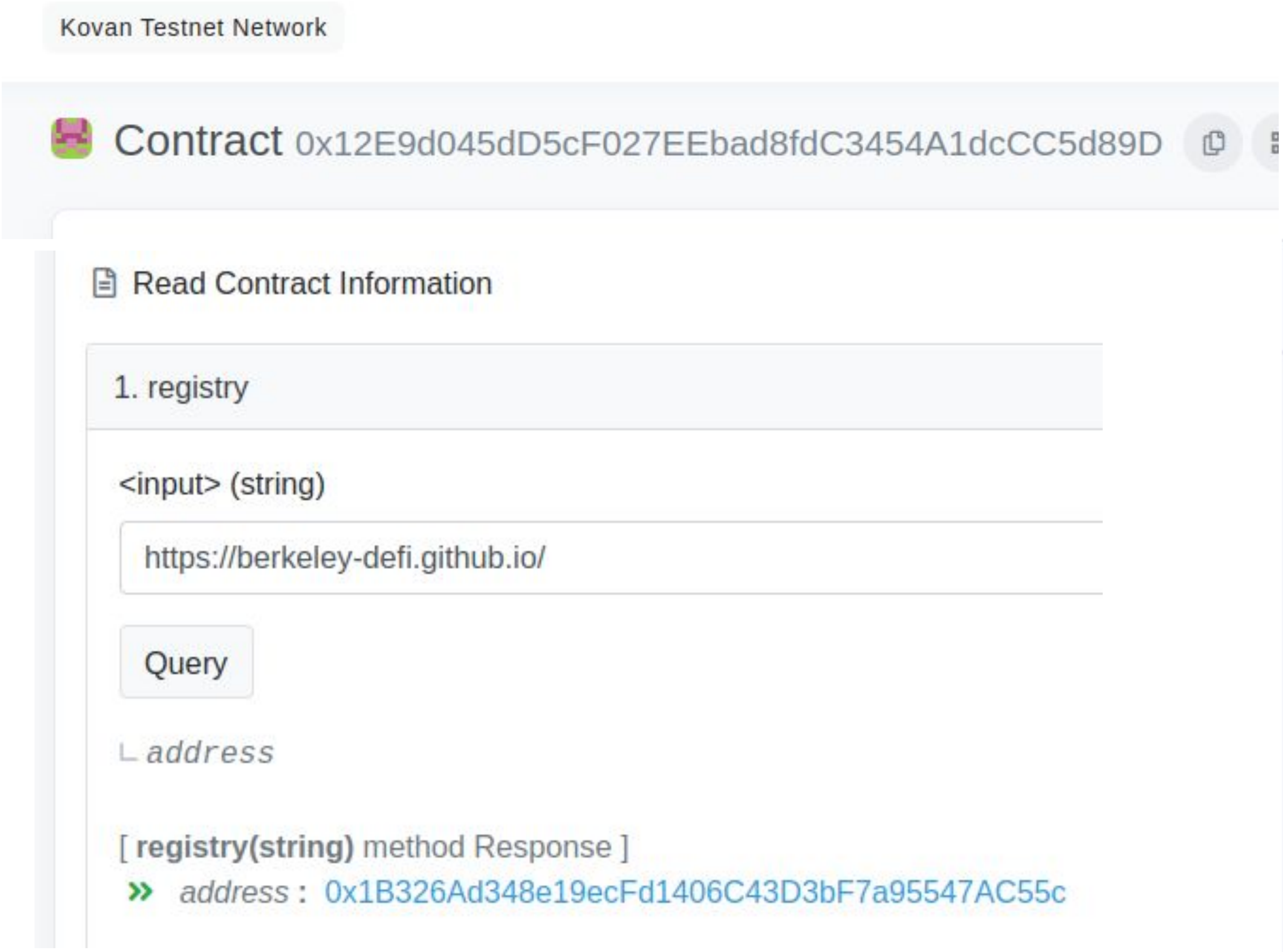
Example: Domain Name Registry in Ethereum



invoke **registerDomain**
with string "*defi.io*"

query registry to look up the
current owner

Let’s look at an instance on the Test Network



Kovan Testnet Network

Contract 0x12E9d045dD5cF027EEbad8fdC3454A1dcCC5d89D

Read Contract Information

1. registry

<input> (string)

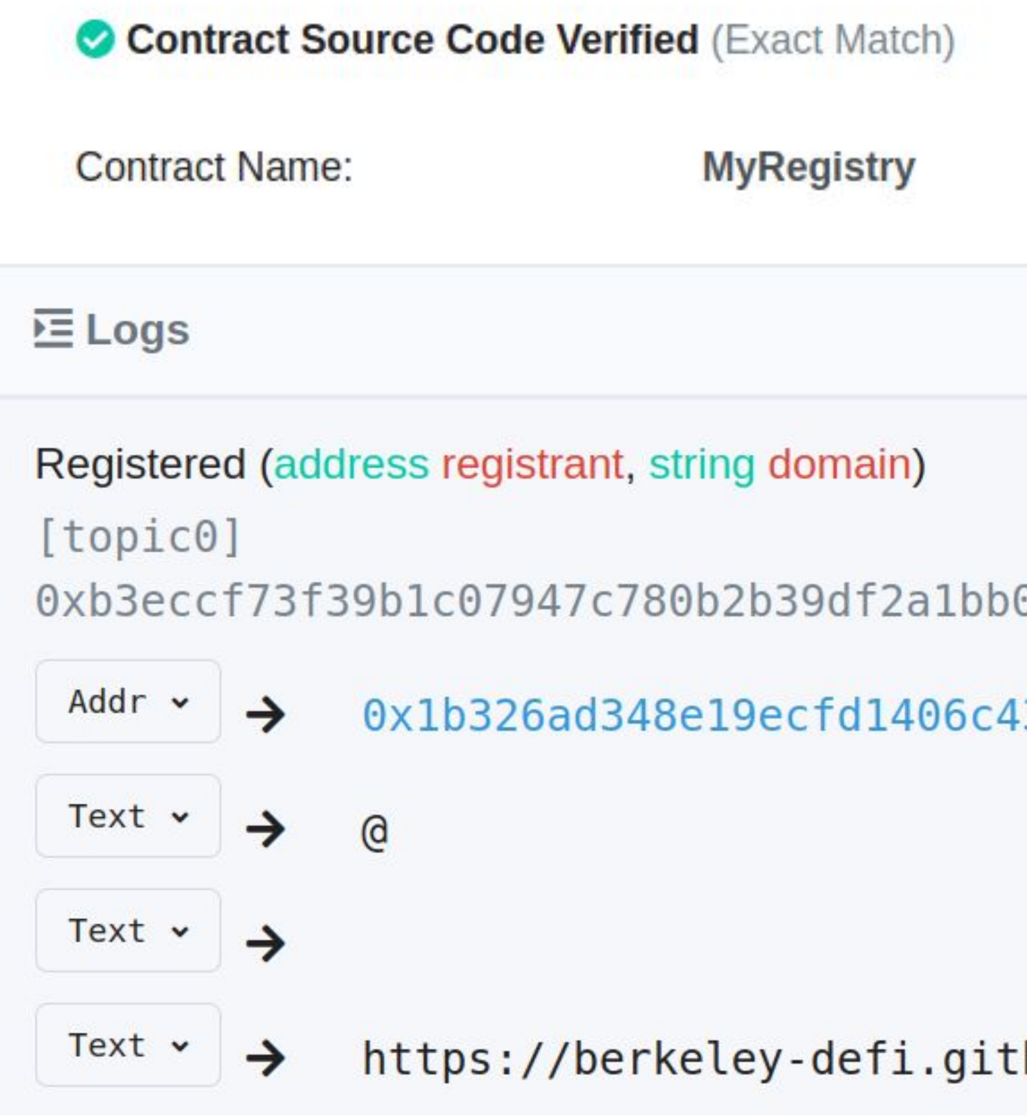
https://berkeley-defi.github.io/

Query

∟ *address*

[**registry(string)** method Response]

» *address* : 0x1B326Ad348e19ecFd1406C43D3bF7a95547AC55c



✓ **Contract Source Code Verified** (Exact Match)

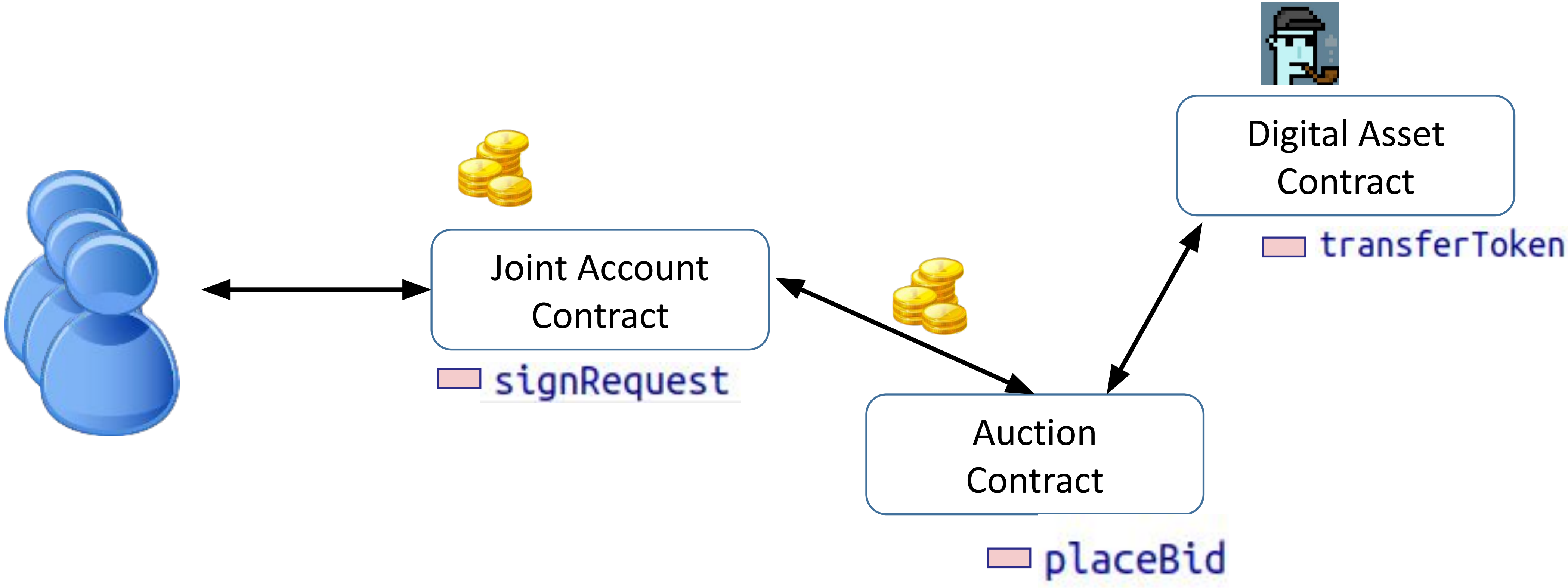
Contract Name: **MyRegistry**

Logs

Registered (address registrant, string domain)

```
[topic0]
0xb3eccf73f39b1c07947c780b2b39df2a1bb0
Addr ⌄ → 0x1b326ad348e19ecfd1406c4
Text ⌄ → @
Text ⌄ →
Text ⌄ → https://berkeley-defi.git
```

Interaction between Contracts



Joint Account Contract

signRequest

Auction Contract

placeBid

Digital Asset Contract

transferToken

Recap of contract programming model so far...

- ***Contract class:*** Defines the program code and storage variables for a contract

- ***Contract object:*** an instance of the class living on the blockchain

- ***Storage fields:*** variables stored by the contract

- ***Functions/methods***: can be invoked to run the given code, updating the state of the contract

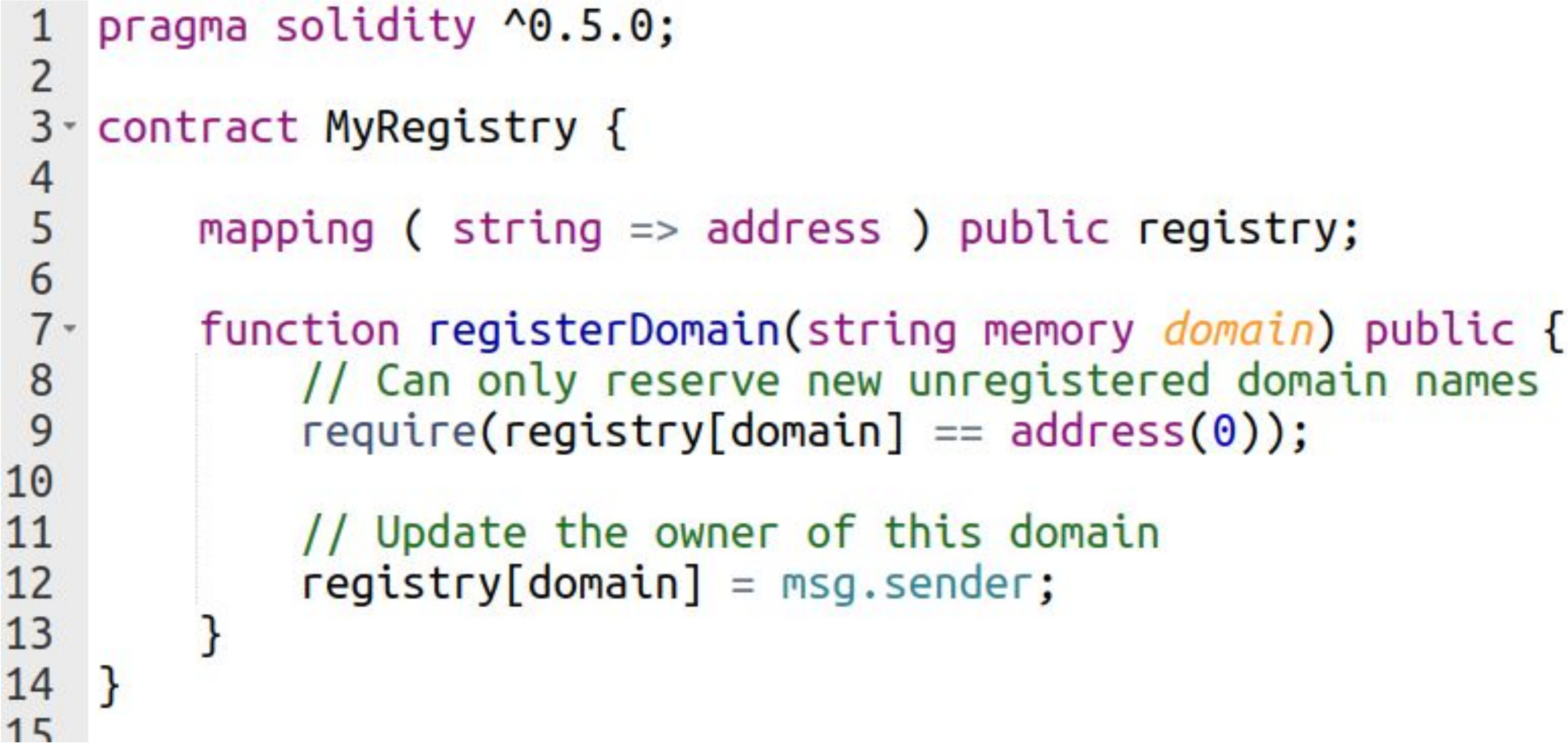
- ***Access control:*** Use "require()" to cancel the transaction if it isn't authorized. You can inspect the caller that invoked the function

- ***Composition:*** interaction between multiple contracts

Question: What’s missing from the example?

- What could go wrong here? How could you fix it

- What other functionality would a useful domain name registry need to have?



```
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

Introduction to Smart Contracts

Part 2: Ethereum programming basics

Just enough to follow the Defi examples later

Part 2: Ethereum programming basics
Just enough to follow the Defi examples

https://defi-learning.org

Outline and background

No programming experience required, but might help

Focus on the unique parts of Solidity

Outline: Solidity and EVM bytecode

Data types Functions and constructors

Visibility/mutability modifiers

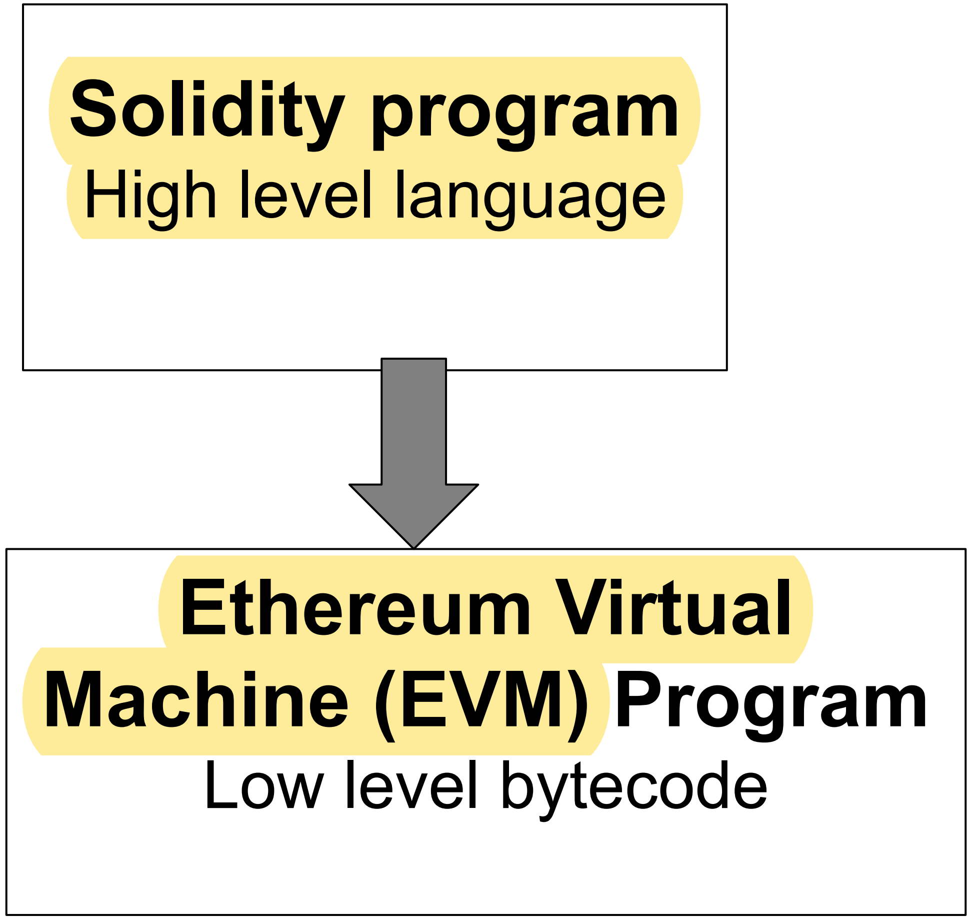
Accessing blockchain metadata

Working with the built-in currency

Events and interaction between contracts

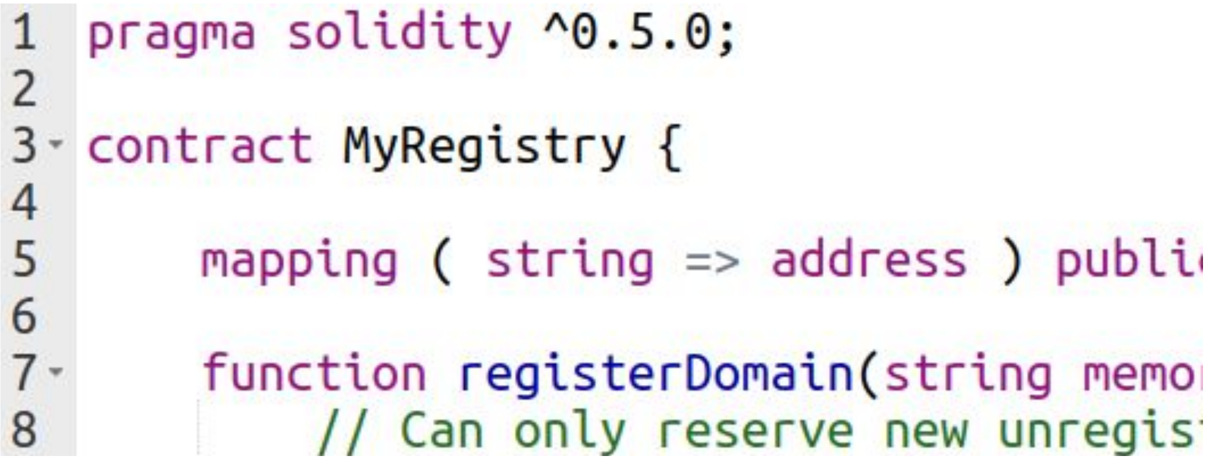
Saved for next time: Gas

Solidity and Ethereum Bytecode

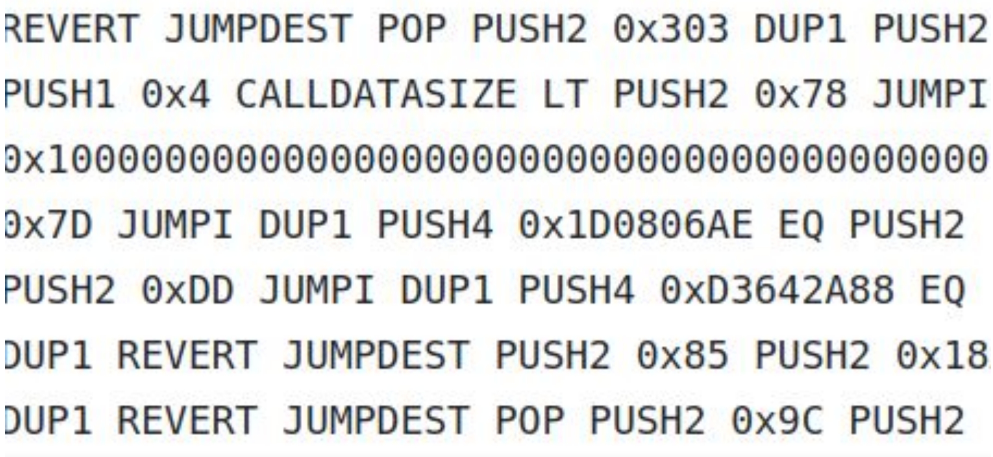


Solidity program
High level language

Ethereum Virtual Machine (EVM) Program
Low level bytecode



```
pragma solidity ^0.5.0;

contract MyRegistry {

    mapping ( string => address ) publi

    function registerDomain(string memo
        // Can only reserve new unregis
```



```
REVERT JUMPDEST POP PUSH2 0x303 DUP1 PUSH2
PUSH1 0x4 CALLDATASIZE LT PUSH2 0x78 JUMPI
0x1000000000000000000000000000000000000000
0x7D JUMPI DUP1 PUSH4 0x1D0806AE EQ PUSH2
PUSH2 0xDD JUMPI DUP1 PUSH4 0xD3642A88 EQ
DUP1 REVERT JUMPDEST PUSH2 0x85 PUSH2 0x18
DUP1 REVERT JUMPDEST POP PUSH2 0x9C PUSH2
```

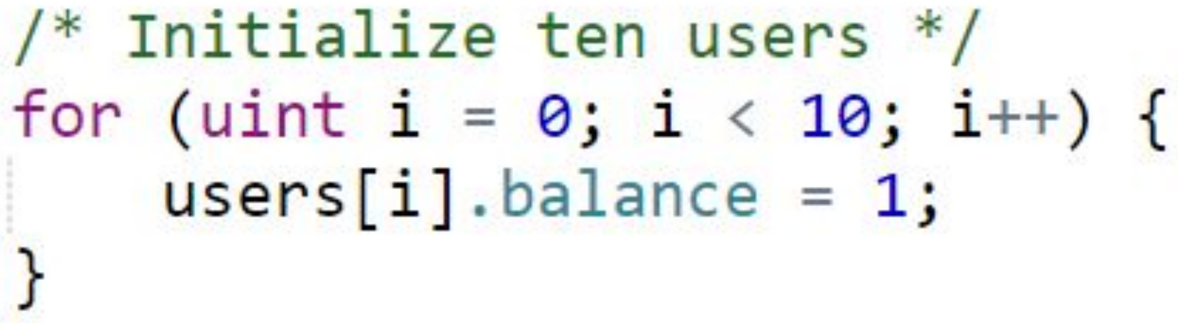
Solidity and Data Types

Solidity is statically typed

Like Java, C, Rust….. unlike python or javascript

Example:

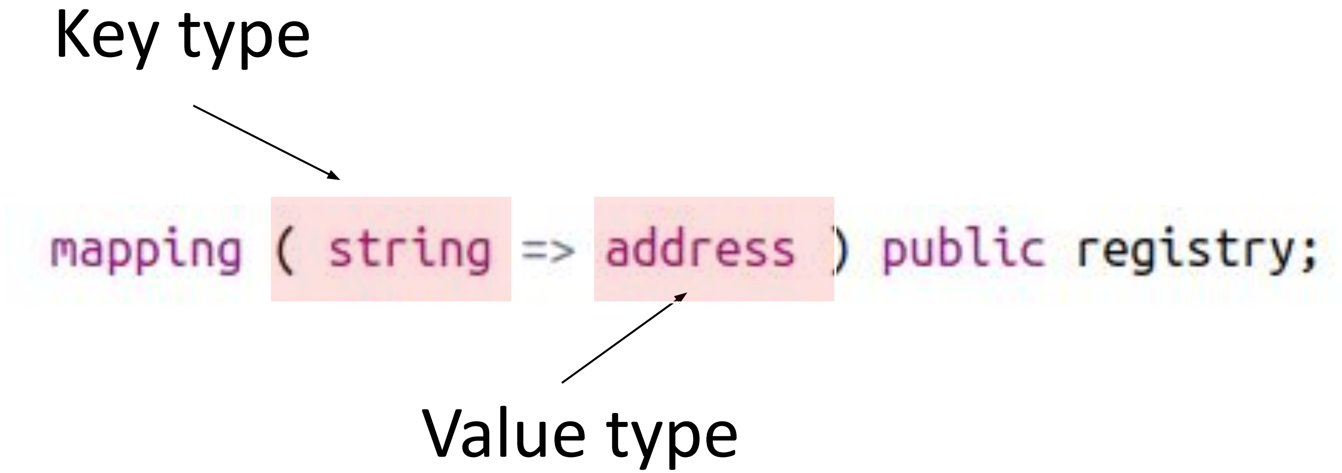
- **Integers**: uint (unsigned 256-bit integer)
  int (signed 256-bit integer)



```
/* Initialize ten users */
for (uint i = 0; i < 10; i++) {
    users[i].balance = 1;
}
```

Mapping data types

- *Mapping*: a key value storage / hash table
- Every key is initially mapped to zero



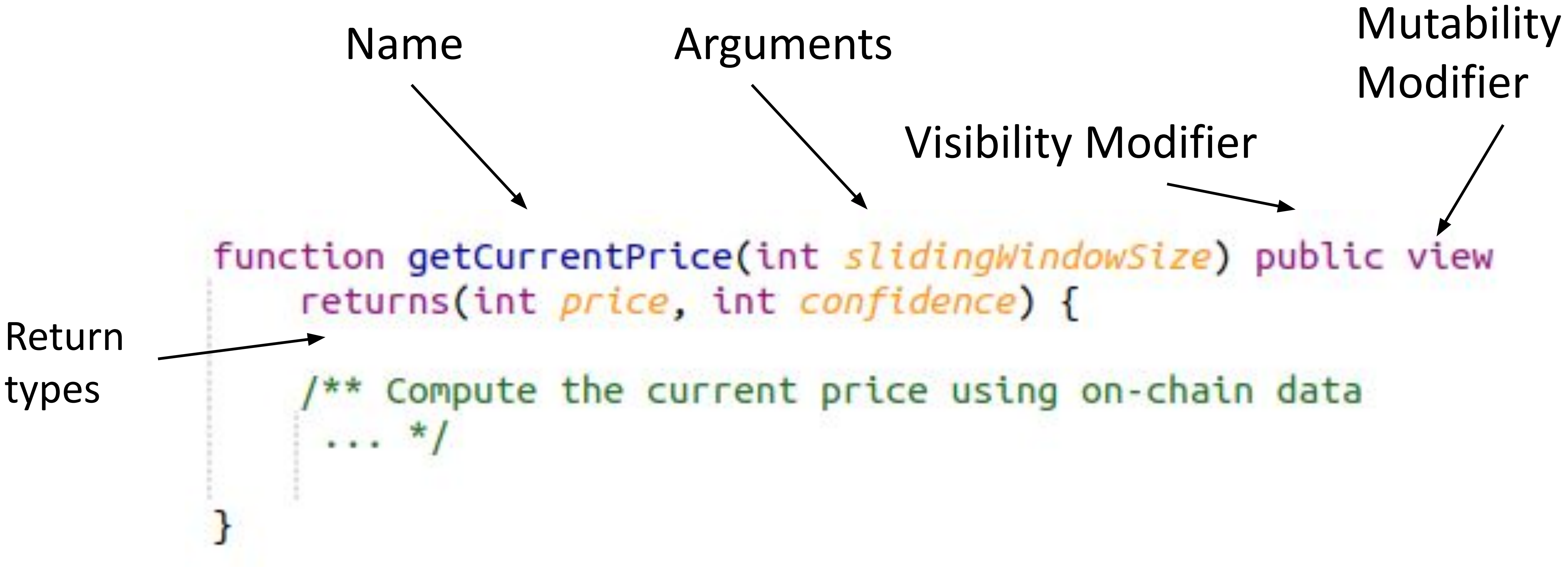
Key type

```
mapping ( string => address ) public registry;
```

Value type

- There is no built-in way to query the length of a mapping, or iterate over its non-zero elements. A separate variable can be used

Function signatures



Name

Arguments

Visibility Modifier

Mutability Modifier

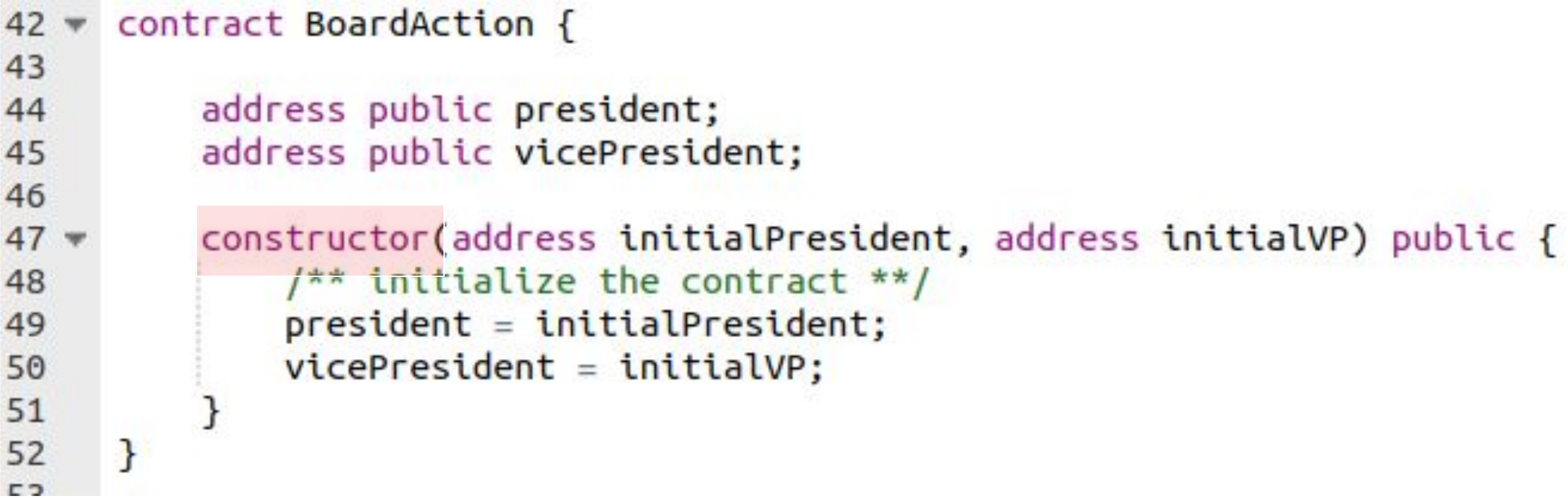
Return types

```
function getCurrentPrice(int slidingWindowSize) public view
    returns(int price, int confidence) {

    /** Compute the current price using on-chain data
     ... */

}
```

Constructors

Invoked when initially creating the contract

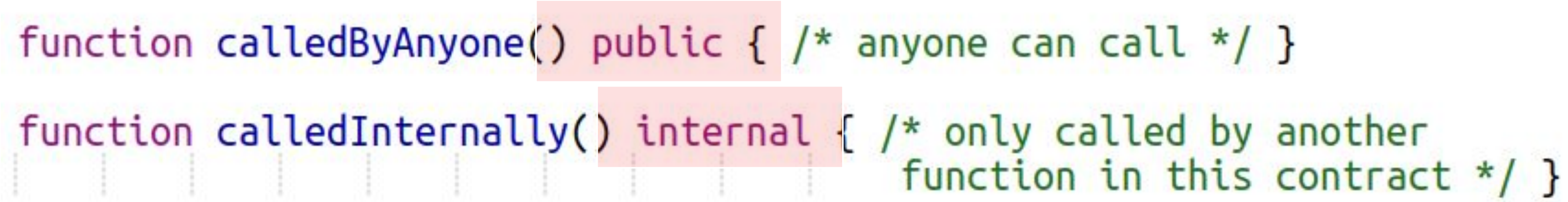
Used to customize settings or give an initial state



```
contract BoardAction {

    address public president;
    address public vicePresident;

    constructor(address initialPresident, address initialVP) public {
        /** initialize the contract **/
        president = initialPresident;
        vicePresident = initialVP;
    }
}
```

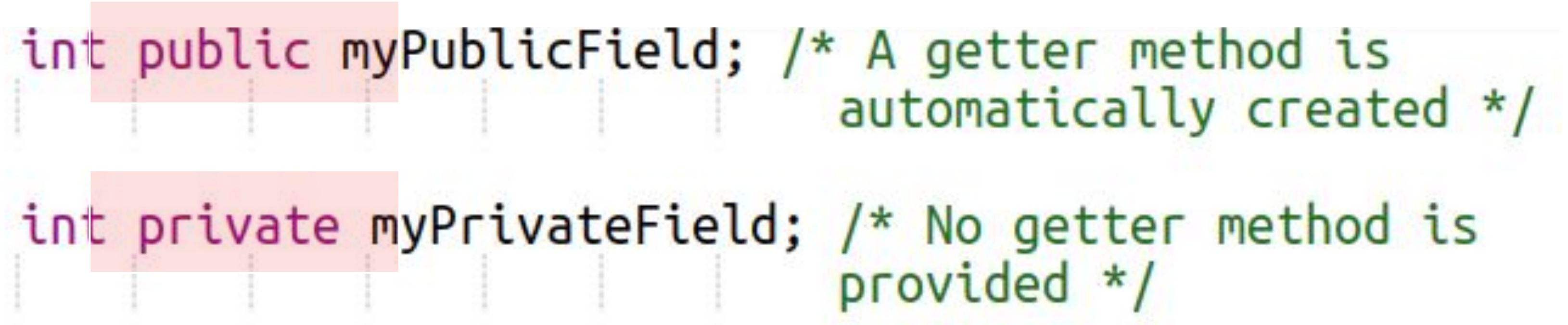
Visibility modifiers

For functions:



```
function calledByAnyone() public { /* anyone can call */ }

function calledInternally() internal { /* only called by another
                                        function in this contract */ }
```

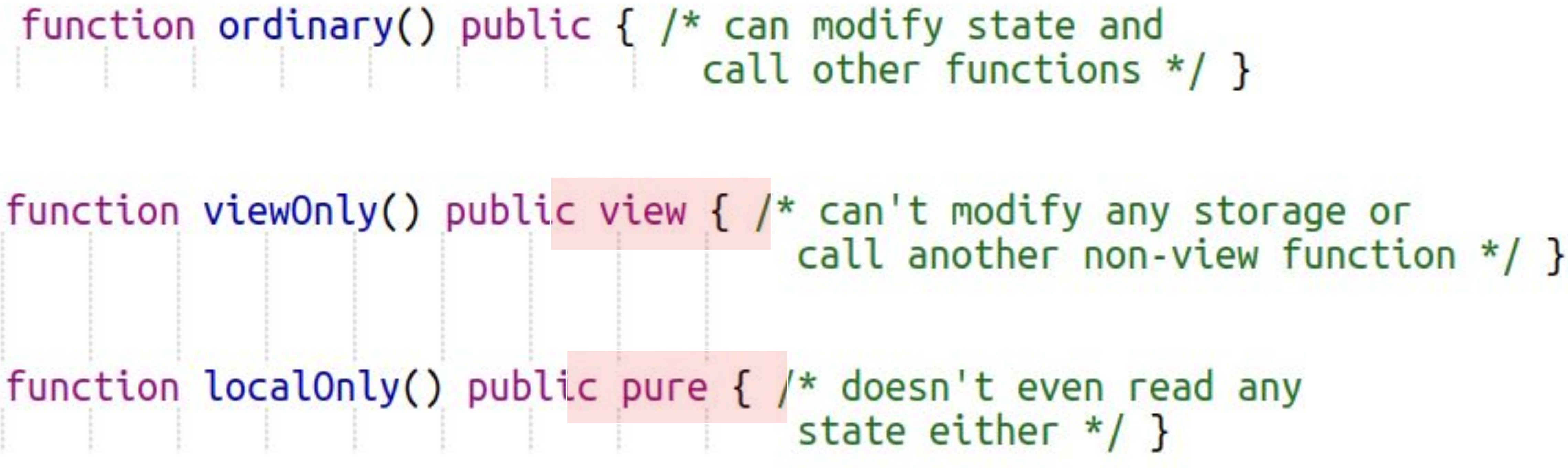
For instance variables:



```
int public myPublicField; /* A getter method is
                            automatically created */

int private myPrivateField; /* No getter method is
                            provided */
```

Question: could `myPrivateField` hold a secret value?

Mutability modifiers

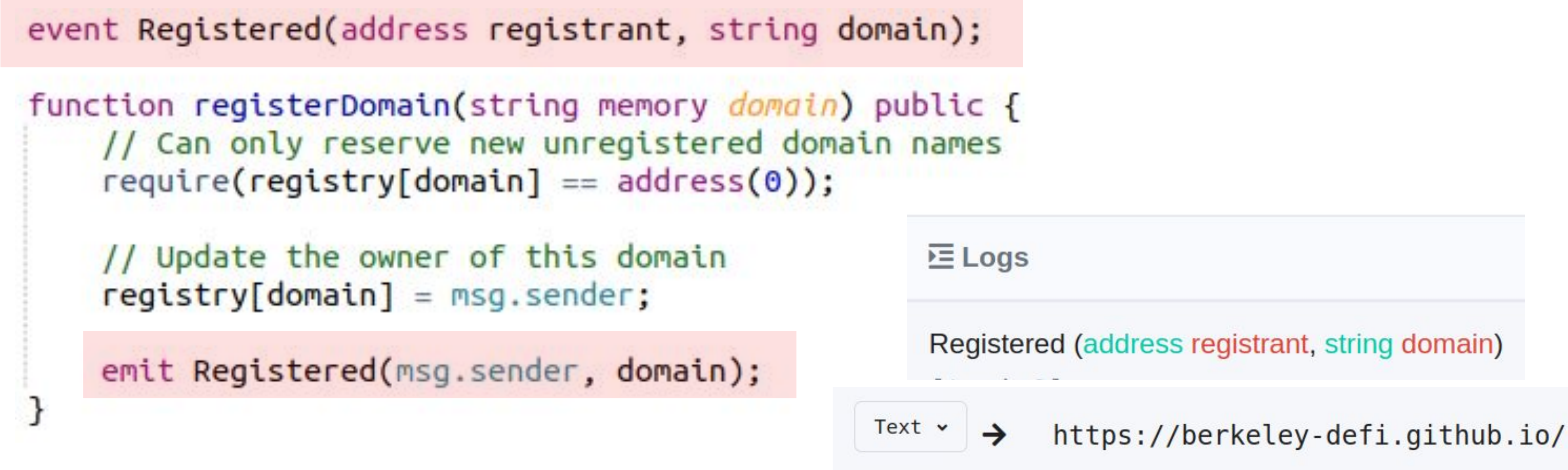


```
function ordinary() public { /* can modify state and
                                call other functions */ }


function viewOnly() public view { /* can't modify any storage or
                                     call another non-view function */ }


function localOnly() public pure { /* doesn't even read any
                                      state either */ }
```

Events

There are two main ways to observe the state of a contract:

- Using ***view*** functions, such as getter functions for public fields
- Looking at ***event logs***. Can “subscribe” to events of a contract



```
event Registered(address registrant, string domain);

function registerDomain(string memory domain) public {
    // Can only reserve new unregistered domain names
    require(registry[domain] == address(0));

    // Update the owner of this domain
    registry[domain] = msg.sender;

    emit Registered(msg.sender, domain);
}
```

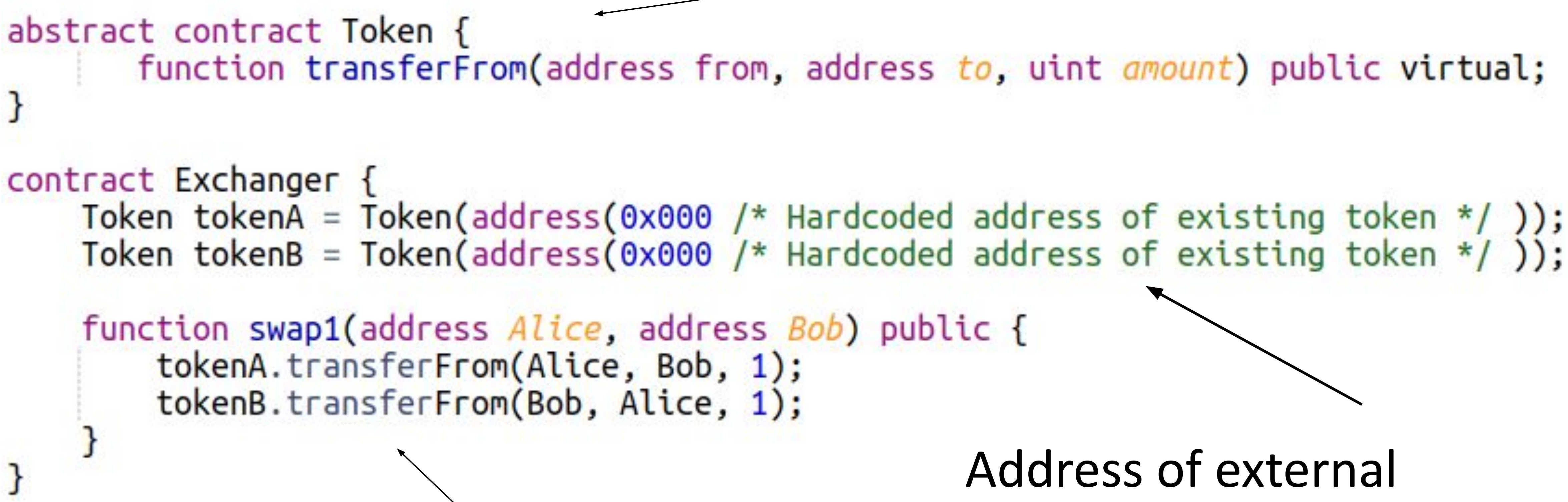
Logs

Registered (address registrant, string domain)

→ https://berkeley-defi.github.io/

Calling methods of other contracts

The interface for an external contract

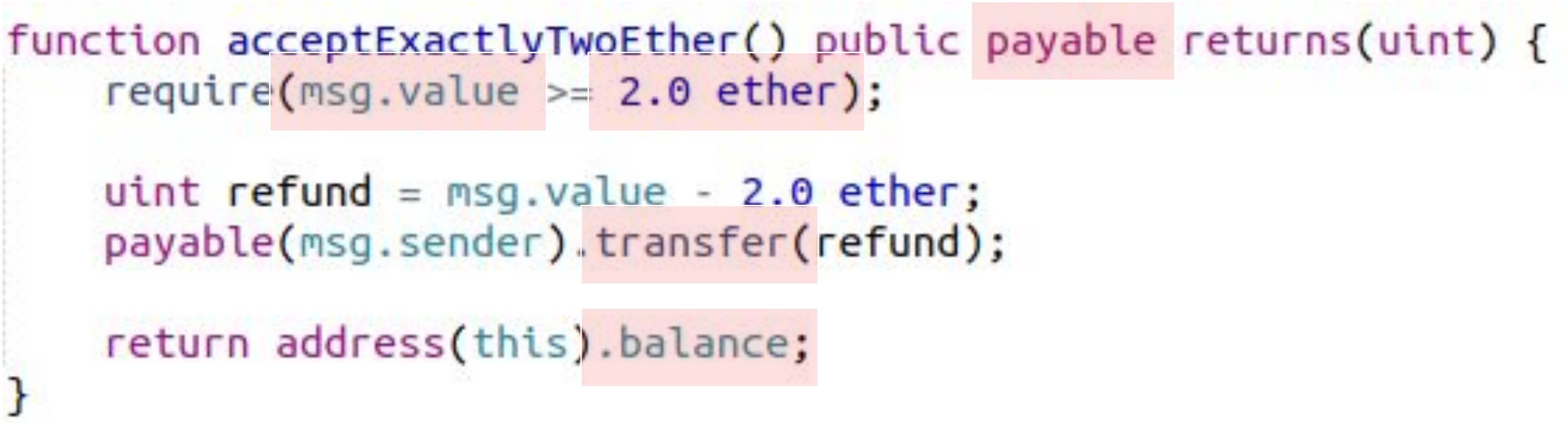


```
abstract contract Token {
        function transferFrom(address from, address to, uint amount) public virtual;
}

contract Exchanger {
    Token tokenA = Token(address(0x000 /* Hardcoded address of existing token */ ));
    Token tokenB = Token(address(0x000 /* Hardcoded address of existing token */ ));

    function swap1(address Alice, address Bob) public {
        tokenA.transferFrom(Alice, Bob, 1);
        tokenB.transferFrom(Bob, Alice, 1);
    }
}
```

Address of external contract instance

Method Call

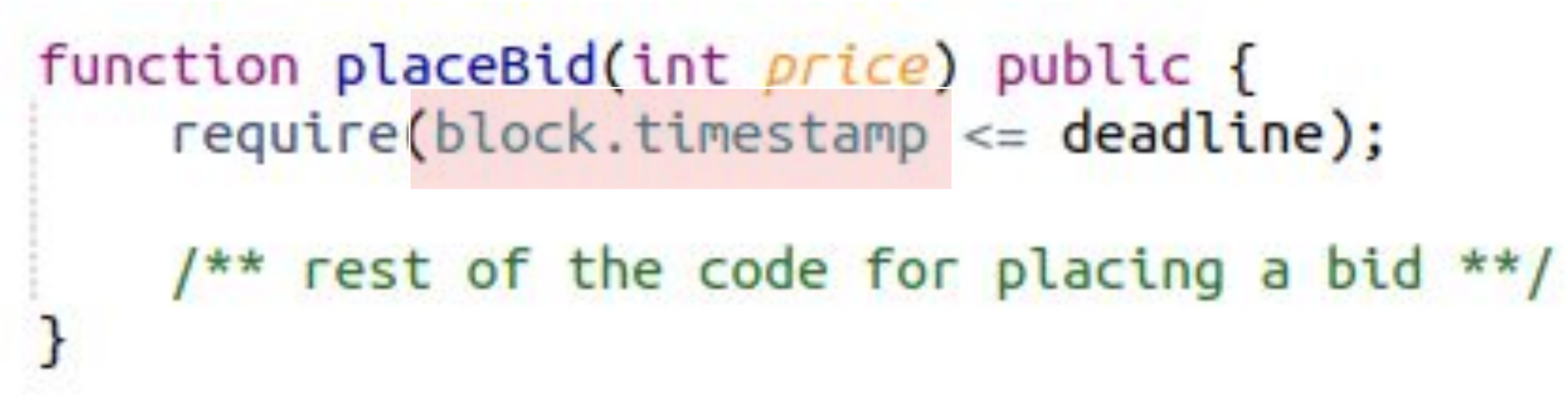
Working with the native currency



```
function acceptExactlyTwoEther() public payable returns(uint) {
    require(msg.value >= 2.0 ether);

    uint refund = msg.value - 2.0 ether;
    payable(msg.sender).transfer(refund);

    return address(this).balance;
}
```

`1.0 ether` => 1000000000000000000 wei

Reading the current time



```
function placeBid(int price) public {
    require(block.timestamp <= deadline);

    /** rest of the code for placing a bid **/
}
```

Other metadata about the block are available too

Other Solidity quirks and features

- ***Storage, memory, calldata***

  Compiler warnings often give recommendations to follow

- ***Creating contracts programmatically***

- ***Modifier macros*** e.g. onlyOwner
- ***Calling another contract's code***
- ***Inheritance and interfaces***
- ***.....***

Next time: Hands on writing and deploying a smart contract

Quiz:

What does this Solidity code do?

What’s wrong with it?

Smart Contract Case Study: Dutch Auction

Part 3a: Smart Contract Case Study
Dutch Auction

CryptoKitties is the Ethereum cat collecting game that's seen over $1m in user spending

This is definitely what blockchain was invented for

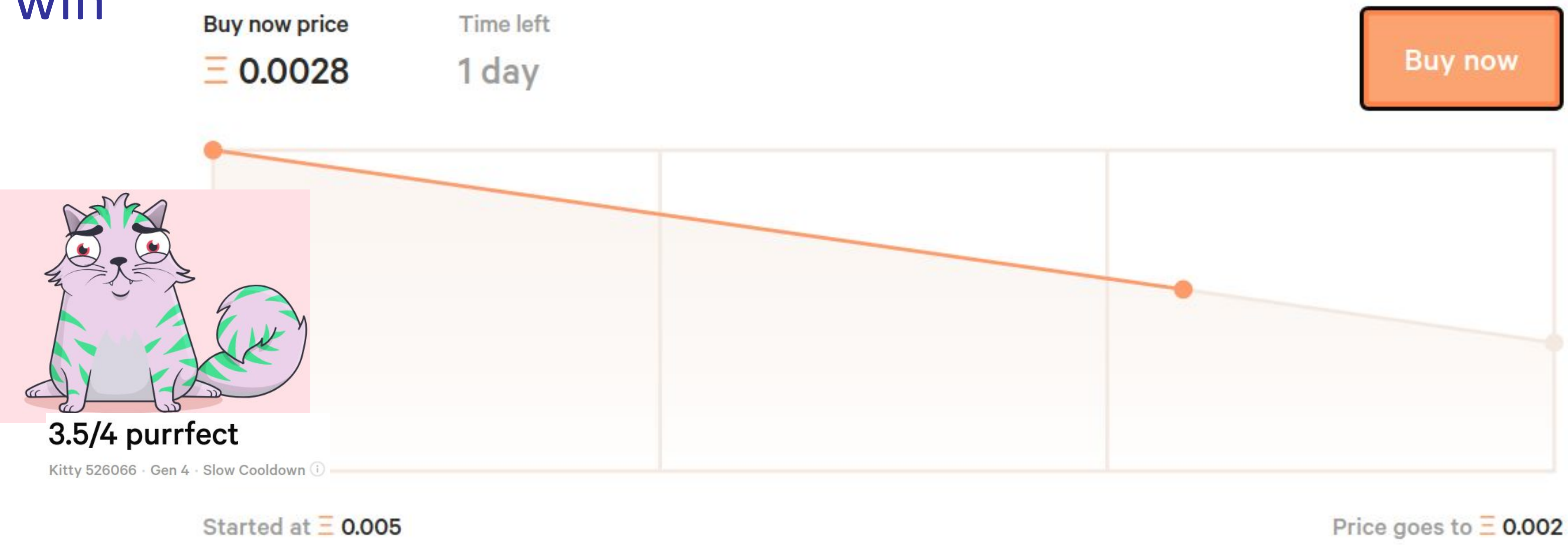
The first big NFT

Cryptokitties is based on Dutch Auctions

The “Buy it Now” price is initially set at a largest value

As time goes on, the “Buy it Now” price is lowered

As soon as someone is ready to buy it, they announce their bid and win



Buy now price
Ξ 0.0028

Time left
1 day

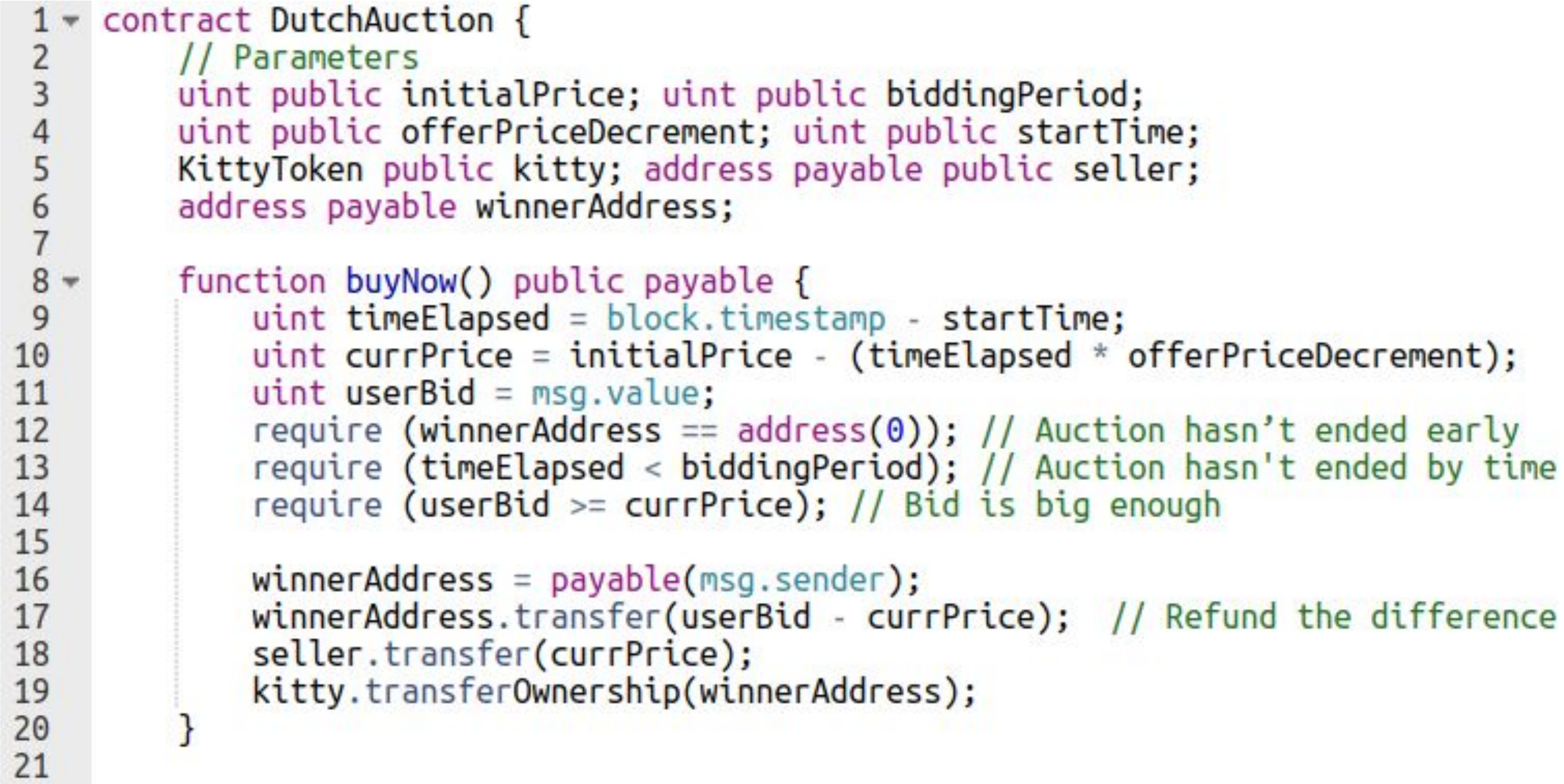
3.5/4 purrfect

Kitty 526066 · Gen 4 · Slow Cooldown

Started at Ξ 0.005

Price goes to Ξ 0.002

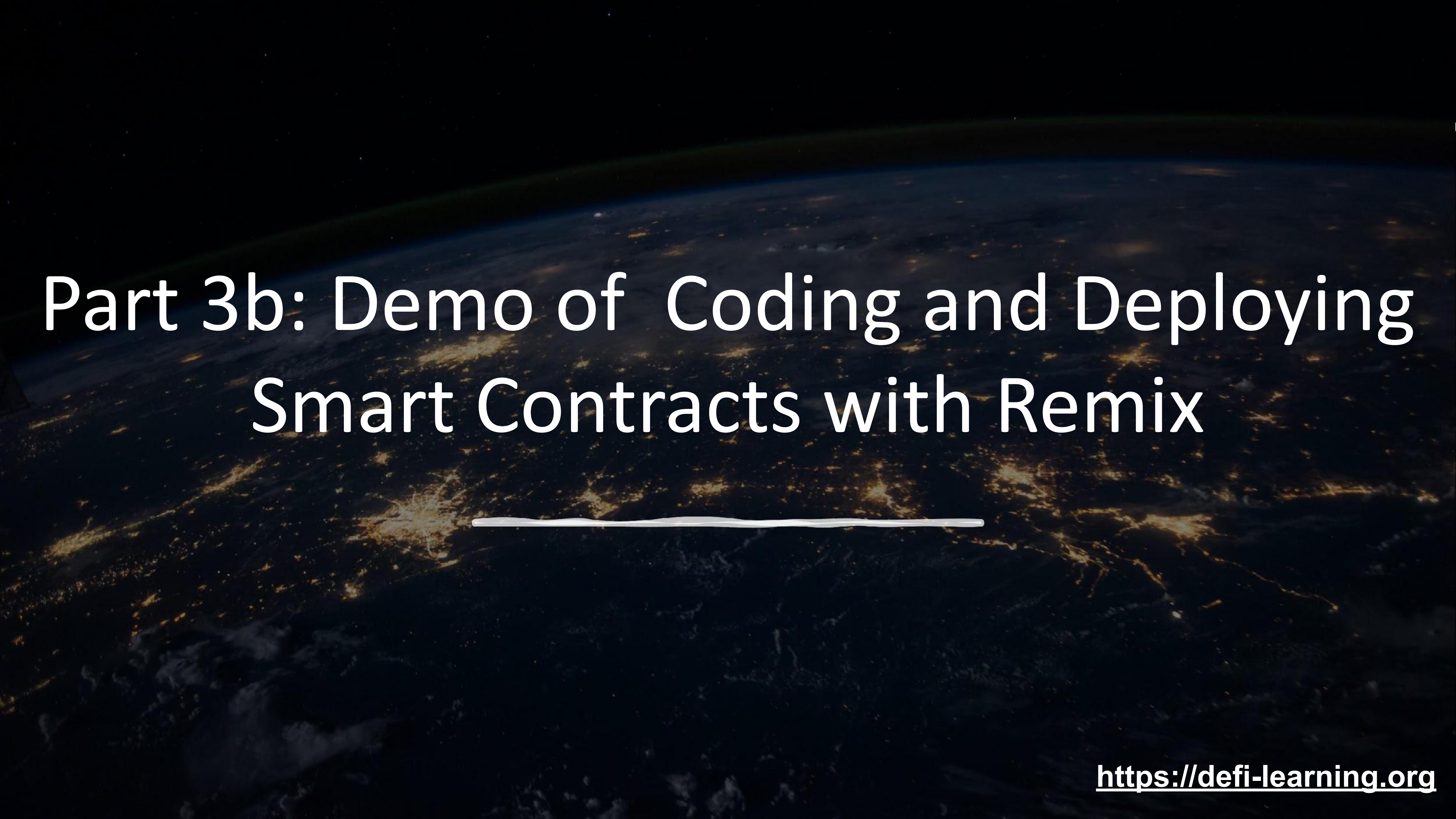
Dutch Auction in a few lines of Solidity



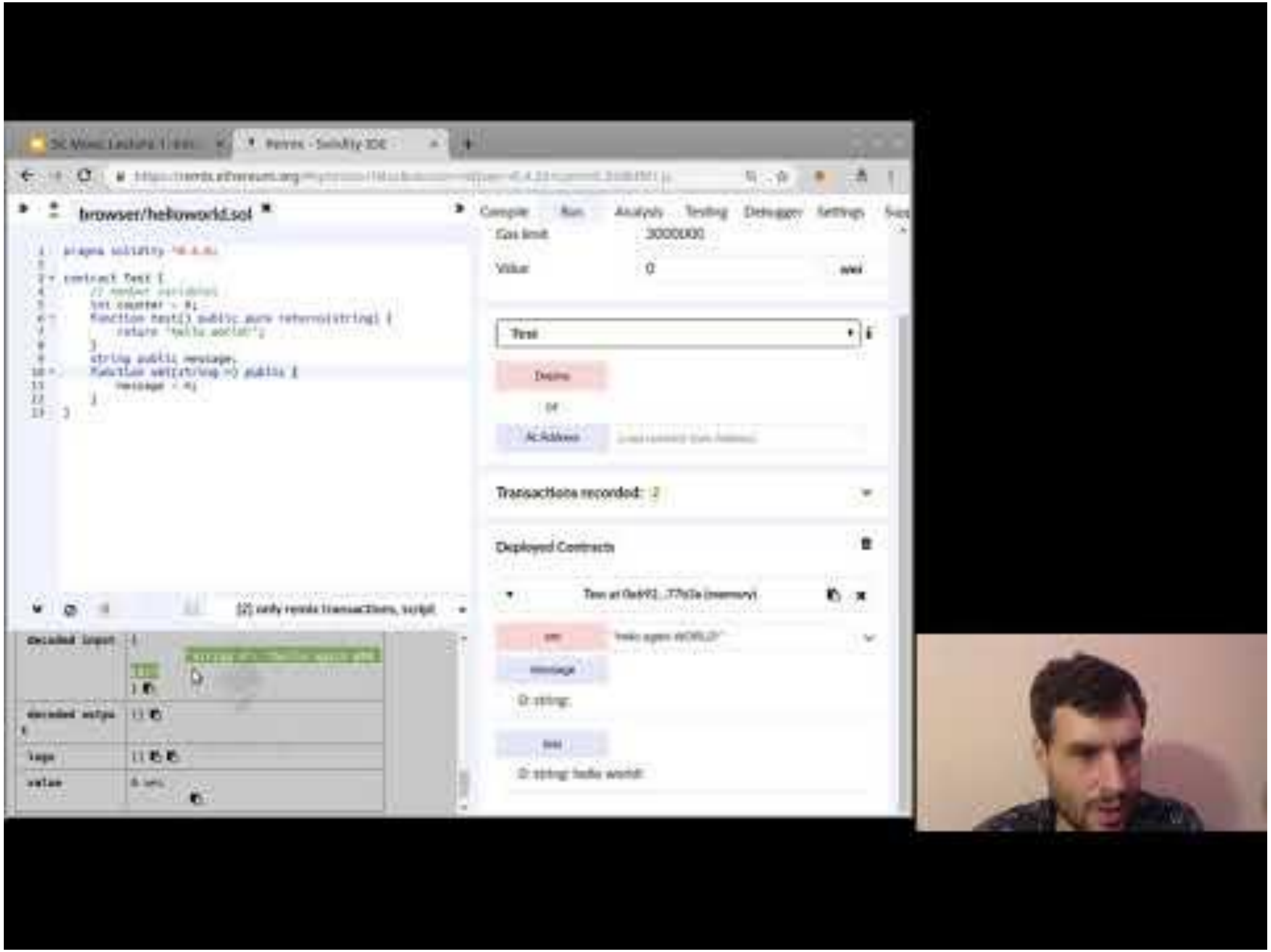
```
contract DutchAuction {
    // Parameters
    uint public initialPrice; uint public biddingPeriod;
    uint public offerPriceDecrement; uint public startTime;
    KittyToken public kitty; address payable public seller;
    address payable winnerAddress;

    function buyNow() public payable {
        uint timeElapsed = block.timestamp - startTime;
        uint currPrice = initialPrice - (timeElapsed * offerPriceDecrement);
        uint userBid = msg.value;
        require (winnerAddress == address(0)); // Auction hasn’t ended early
        require (timeElapsed < biddingPeriod); // Auction hasn't ended by time
        require (userBid >= currPrice); // Bid is big enough

        winnerAddress = payable(msg.sender);
        winnerAddress.transfer(userBid - currPrice);  // Refund the difference
        seller.transfer(currPrice);
        kitty.transferOwnership(winnerAddress);
    }

```

Introduction to Smart Contracts

Part 3: Demonstration of Coding and Deploying Smart Contracts with Remix



Part 3b: Demo of Coding and Deploying Smart Contracts with Remix



browser/helloworld.sol

```
pragma solidity ^0.4.0;

contract Test {
    // member variables
    int counter = 0;
    function test() public pure returns(string) {
        return "hello world";
    }
    string public message;
    function set(string s) public {
        message = s;
    }
}
```

Gas limit 3000000

Value 0 wei

Transactions recorded: 2

Test at 0xb92...77b5a (memory)

set "hello again WORLD"

0: string:

0: string: hello world

decoded input

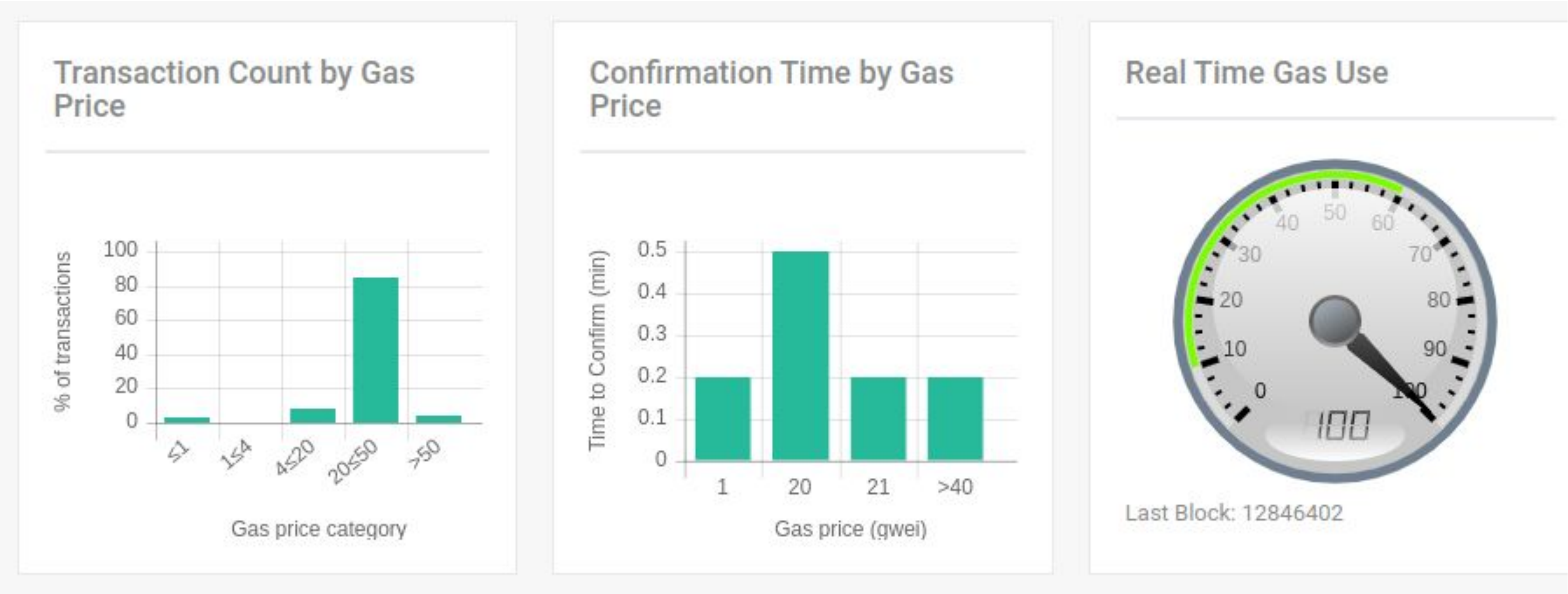
decoded output

logs

value 0 wei

Part 4: Gas in Ethereum

Each transaction has to pay a gas fee



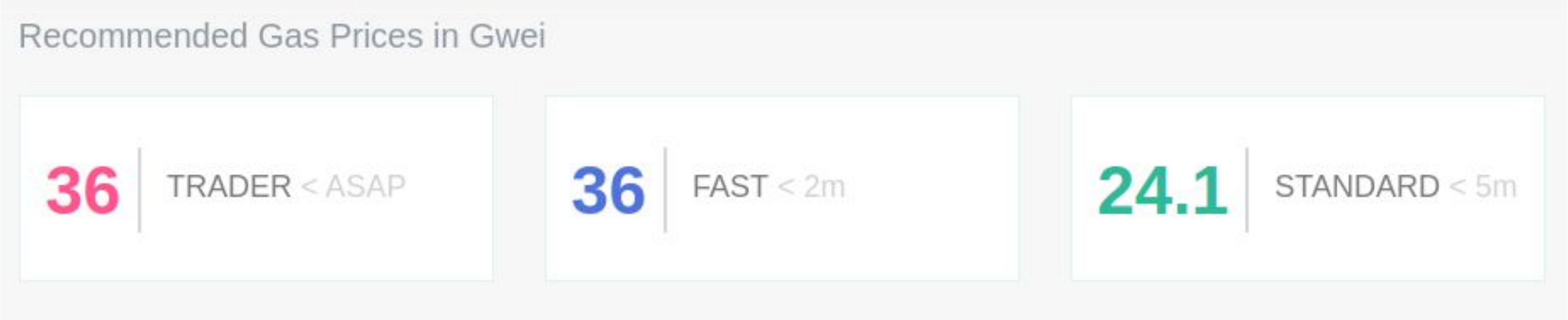
Transaction Count by Gas Price

Confirmation Time by Gas Price

Real Time Gas Use

Last Block: 12846402

More complicated transactions consume more gas, so they cost more.



Recommended Gas Prices in Gwei

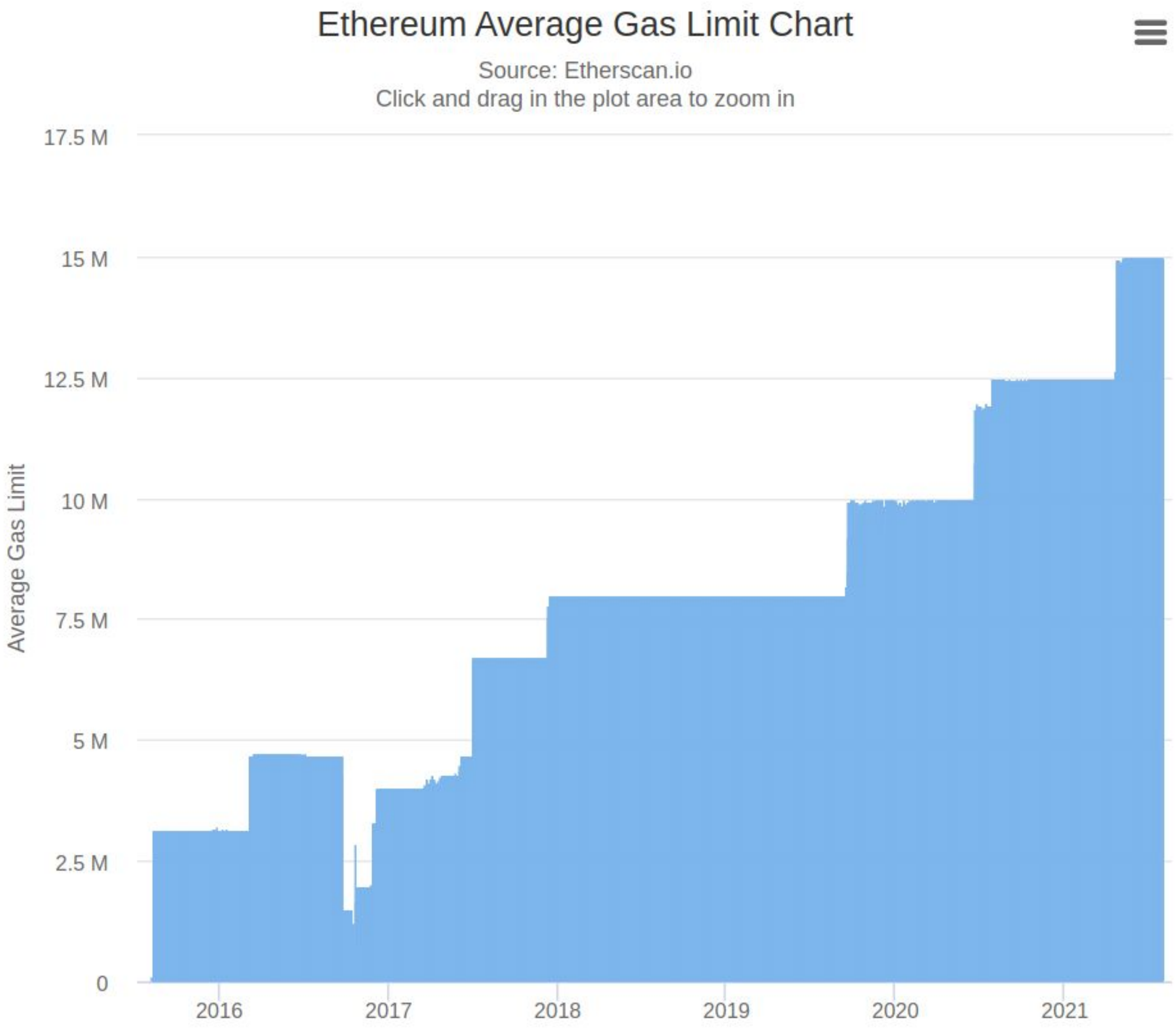
36 TRADER < ASAP

36 FAST < 2m

24.1 STANDARD < 5m

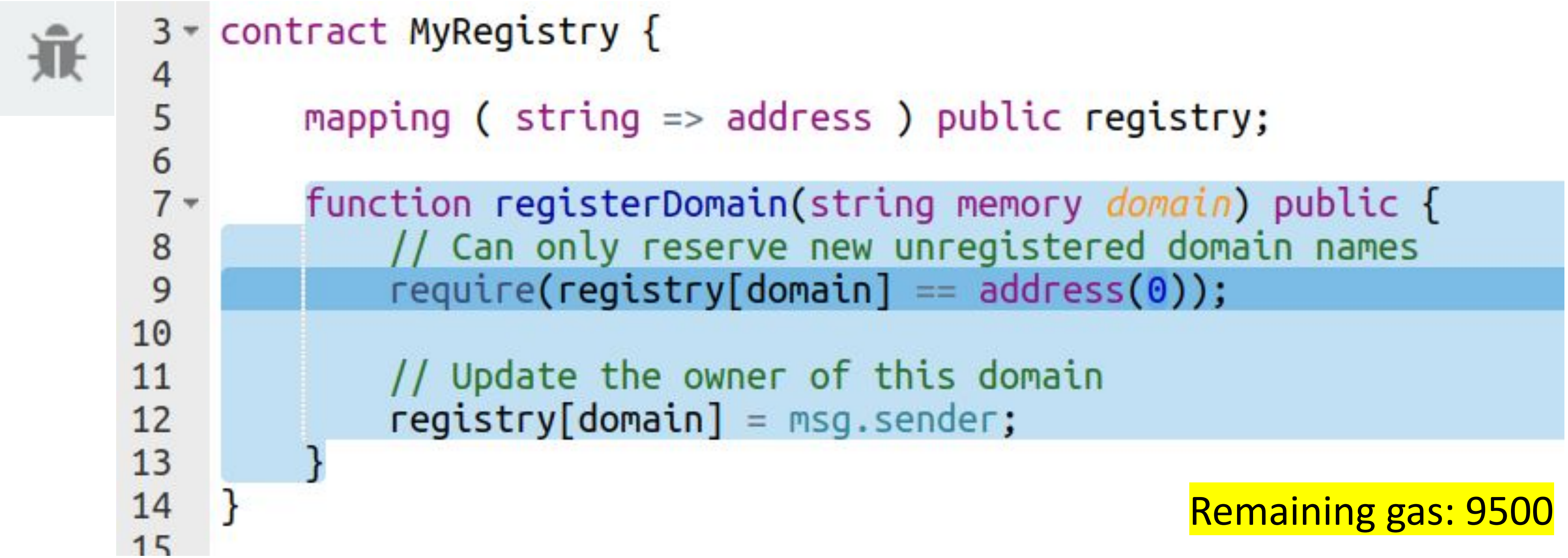
source: ethgasstation.info

Miners limited by a global limit on gas per block



Every instruction costs a fixed amount of gas

A counter of gas used is tracked when executing the transaction

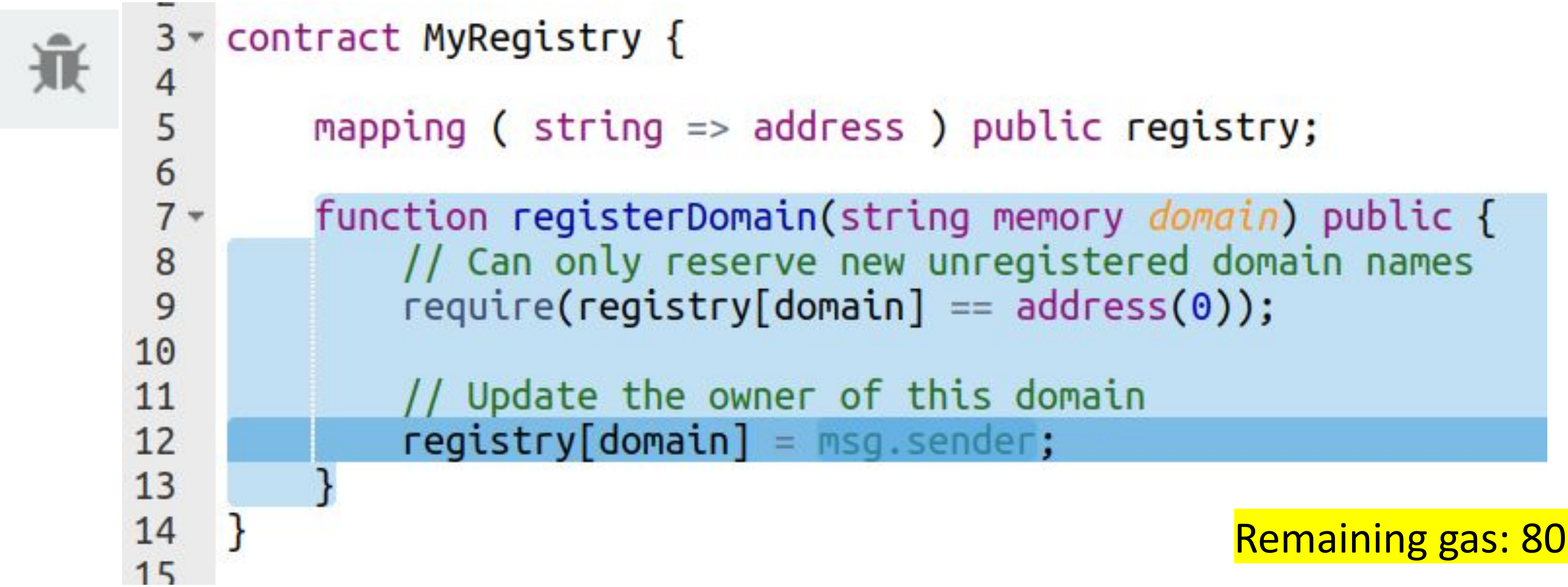


```
contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

Remaining gas: 9500

Every instruction costs a fixed amount of gas

A counter of gas used is tracked when executing the transaction

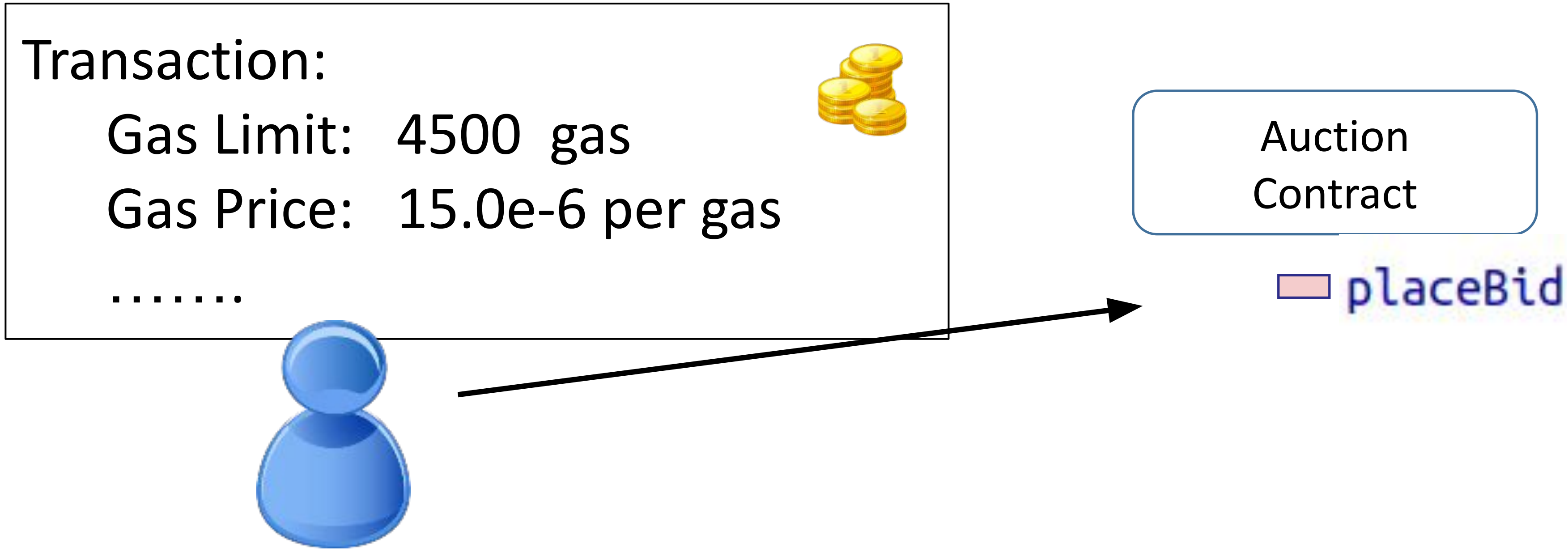


```
contract MyRegistry {

    mapping ( string => address ) public registry;

    function registerDomain(string memory domain) public {
        // Can only reserve new unregistered domain names
        require(registry[domain] == address(0));

        // Update the owner of this domain
        registry[domain] = msg.sender;
    }
}
```

Remaining gas: 8000

Gas limits and refunds

- Each transaction specifies a gas limit and a price for the gas, in units of Ether
- Ether value to pay for the gas must be reserved up front
- At end of contract execution, unused gas is refunded



Transaction:

Gas Limit: 4500 gas

Gas Price: 15.0e-6 per gas

…….

Auction Contract

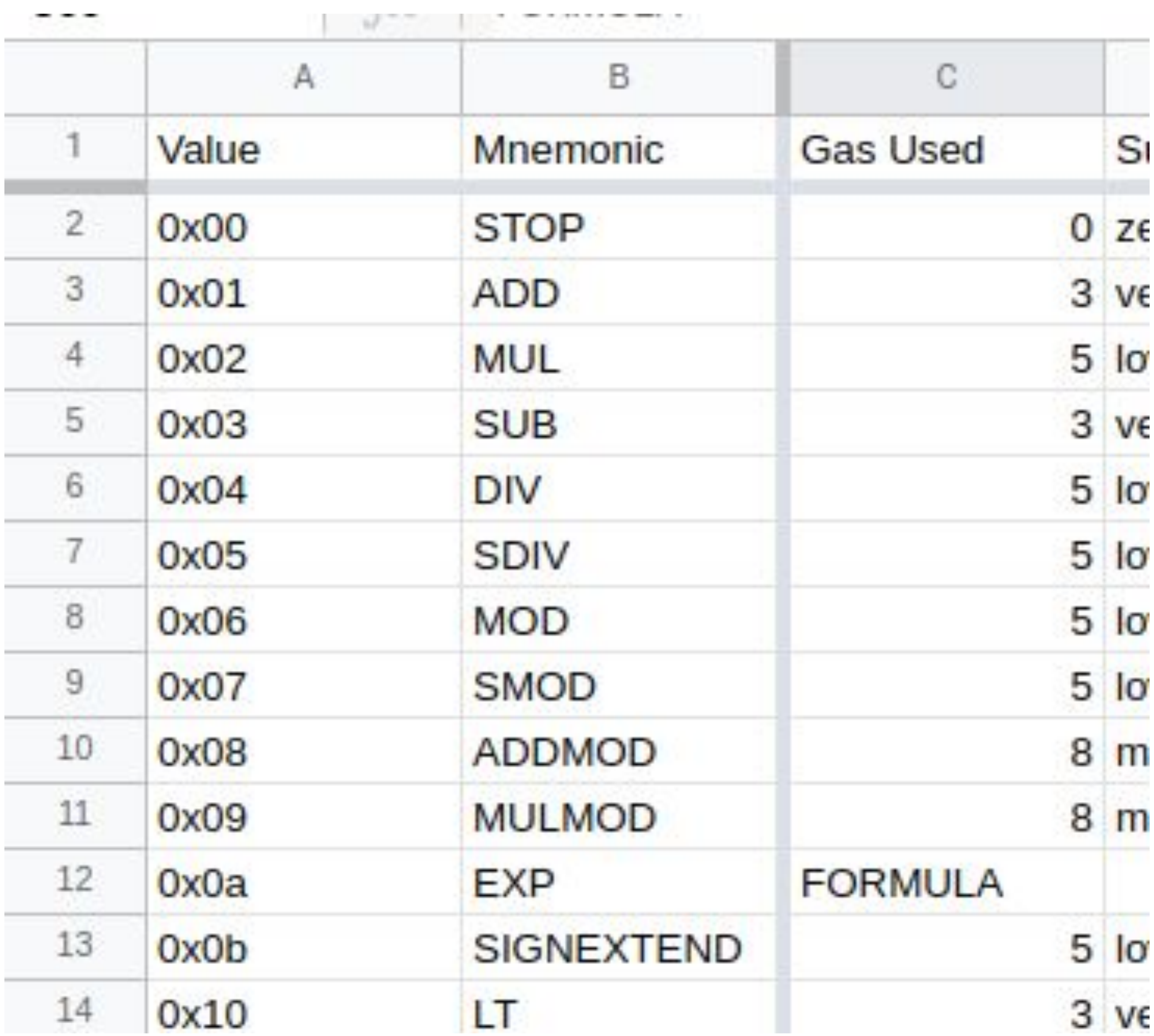
placeBid

There’s a big table for gas prices per opcode

This is based on the compiled opcodes for Ethereum Virtual Machine (EVM), not high level code

“FORMULA” means the gas for this opcode depends on the arguments (for example on the size of the argument).

https://github.com/djrtwo/evm-opcode-gas-costs



	A	B	C	
1	Value	Mnemonic	Gas Used	Su
2	0x00	STOP	0	ze
3	0x01	ADD	3	ve
4	0x02	MUL	5	lo
5	0x03	SUB	3	ve
6	0x04	DIV	5	lo
7	0x05	SDIV	5	lo
8	0x06	MOD	5	lo
9	0x07	SMOD	5	lo
10	0x08	ADDMOD	8	m
11	0x09	MULMOD	8	m
12	0x0a	EXP	FORMULA	
13	0x0b	SIGNEXTEND	5	lo
14	0x10	LT	3	ve

What happens when gas runs out?

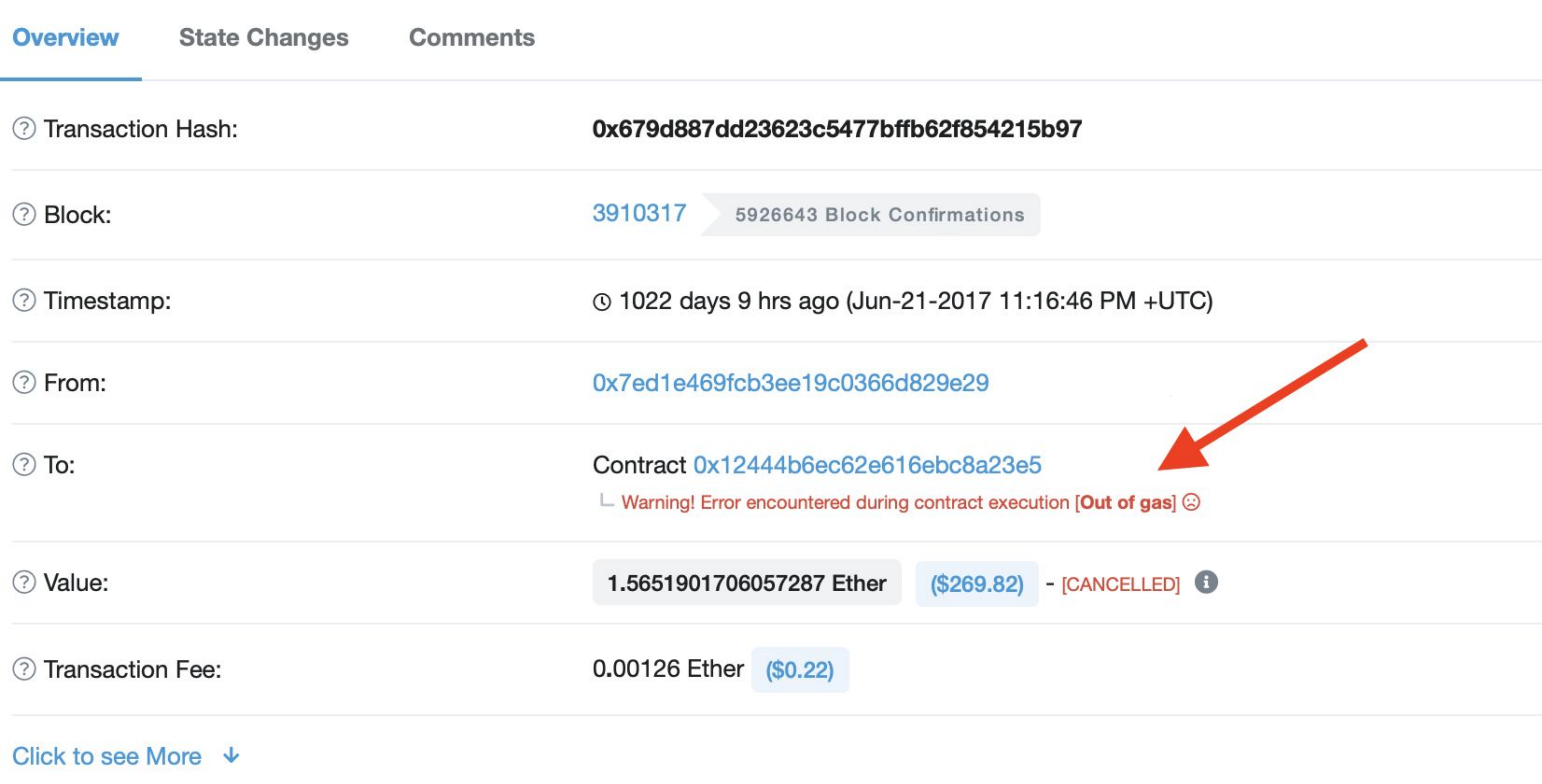
- An **Out-Of-Gas** exception is thrown

- Any changes made to storage variables, any account transfers, are **reverted** to their state before this method call

- You are ***still charged*** the gas fee for every instruction leading up to the exception

- Like other exceptions, it can be ***caught*** by a handler function

- Methods can be invoked with just a portion of available gas



Transaction Hash: 0x679d887dd23623c5477bffb62f854215b97

Block: 3910317 5926643 Block Confirmations

Timestamp: 1022 days 9 hrs ago (Jun-21-2017 11:16:46 PM +UTC)

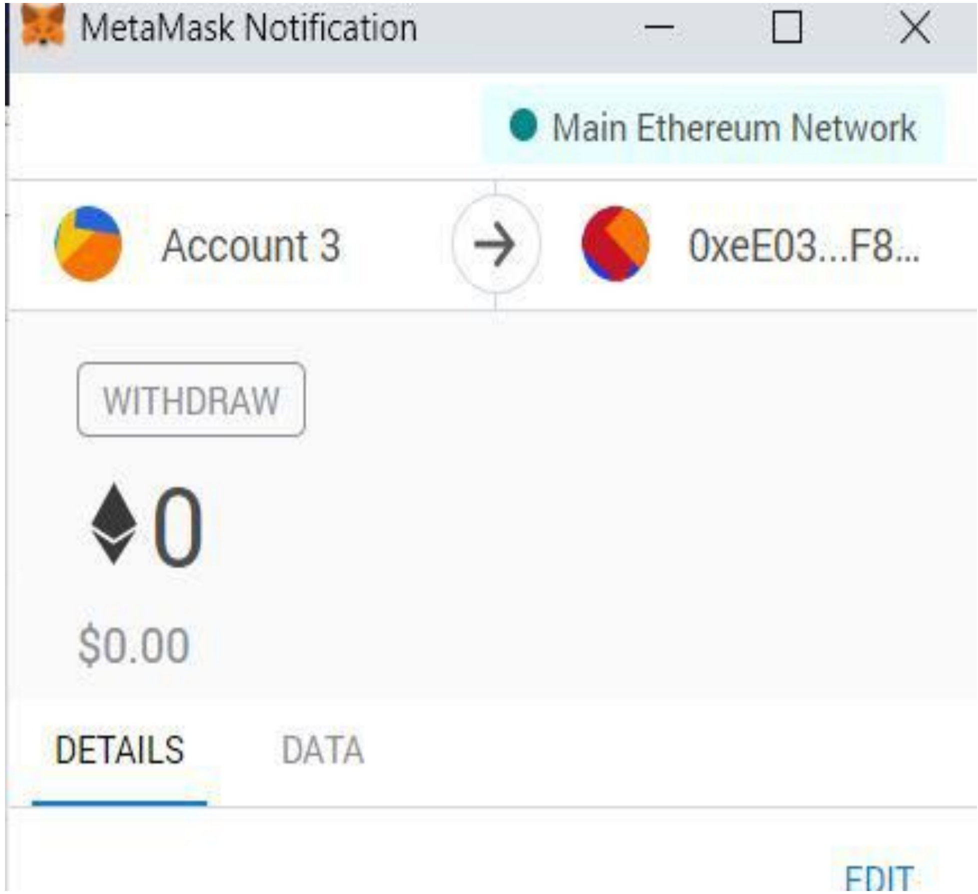
From: 0x7ed1e469fcb3ee19c0366d829e29

To: Contract 0x12444b6ec62e616ebc8a23e5

└ Warning! Error encountered during contract execution [**Out of gas**] ☹

Value: 1.5651901706057287 Ether ($269.82) - [CANCELLED]

Transaction Fee: 0.00126 Ether ($0.22)



MetaMask Notification

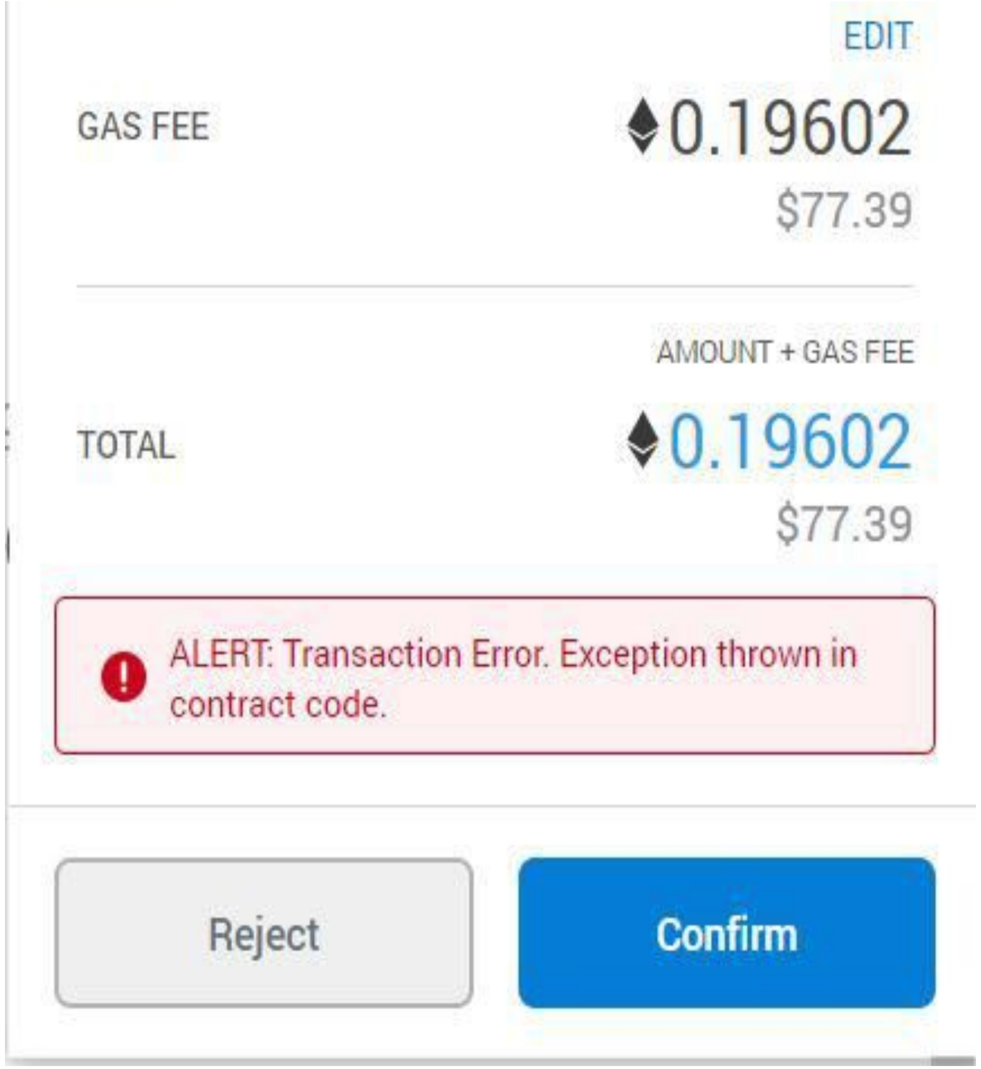
Main Ethereum Network

Account 3 → 0xeE03...F8...

WITHDRAW

0

$0.00



GAS FEE 0.19602

$77.39

AMOUNT + GAS FEE

TOTAL 0.19602

$77.39

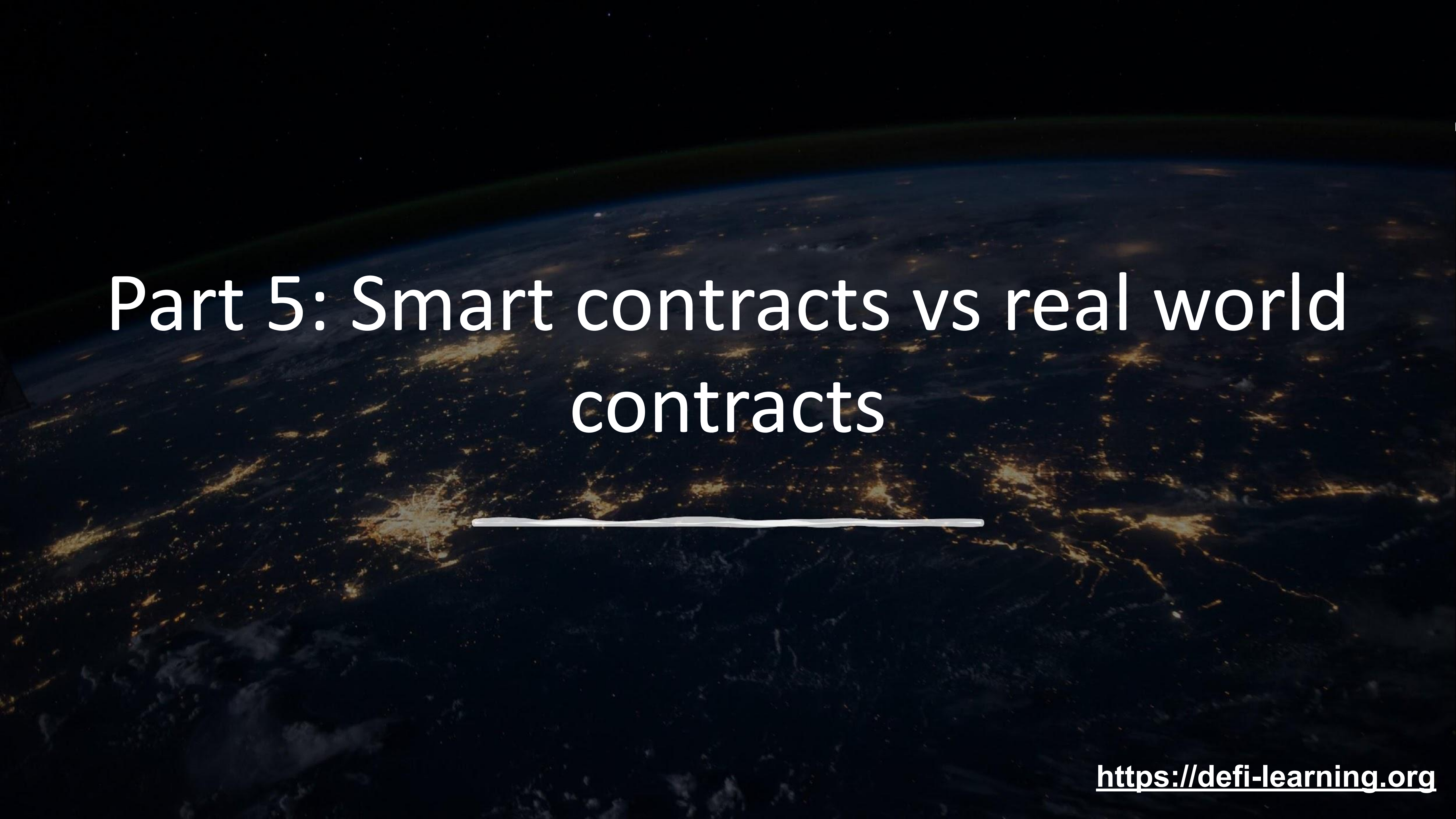
ALERT: Transaction Error. Exception thrown in contract code.

Recap: Gas in Ethereum

Pay for the computation you use with gas

Gives a good reason to optimize your code

Next time: a case study comparing smart contracts with legal contracts



Part 5: Smart contracts vs real world contracts

Traditional contracts: the basic elements

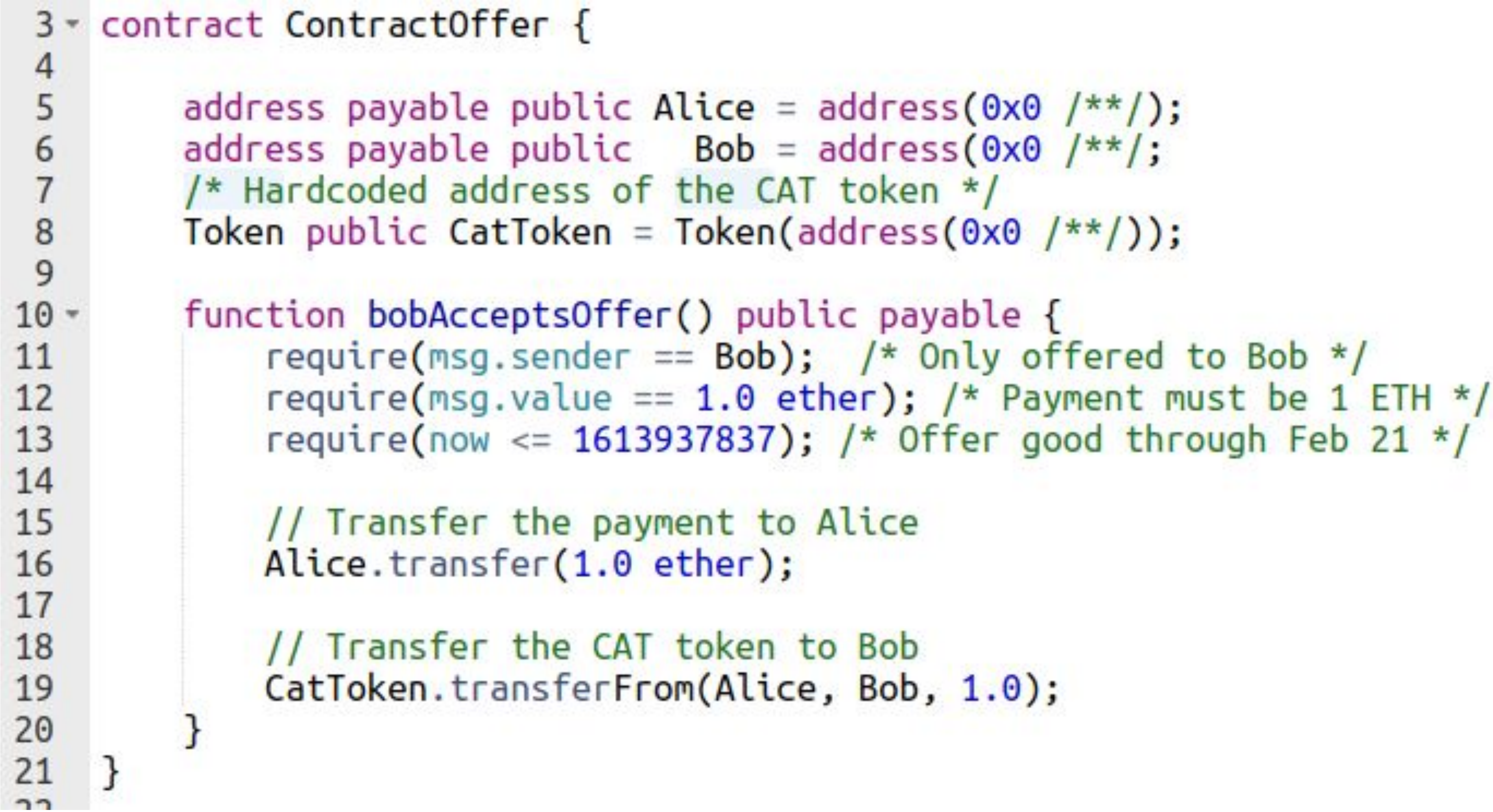
If Bob pays Alice
1.0 ETH by Feb 21,
then Alice will transfer
1.0 CAT tokens to Bob.

Alice

- Offer and acceptance
- Consideration
- Mutual agreement
- Legality and Capacity

How could we make a smart contract that models this contract?

Example: Offering a token for sale



```
contract ContractOffer {

    address payable public Alice = address(0x0 /**/);
    address payable public   Bob = address(0x0 /**/;
    /* Hardcoded address of the CAT token */
    Token public CatToken = Token(address(0x0 /**/));

    function bobAcceptsOffer() public payable {
        require(msg.sender == Bob);  /* Only offered to Bob */
        require(msg.value == 1.0 ether); /* Payment must be 1 ETH */
        require(now <= 1613937837); /* Offer good through Feb 21 */

        // Transfer the payment to Alice
        Alice.transfer(1.0 ether);

        // Transfer the CAT token to Bob
        CatToken.transferFrom(Alice, Bob, 1.0);
    }
}
```

Example: Offering a token for sale

- ***Offer and acceptance***

  To accept an offer, have to digitally sign the transaction.
  Alice would have to transfer asset to the contract ahead of time

- ***Consideration***

  Payment is collected in the blockchain's native currency

- ***Mutuality***

  The high level code for the contract is typically published

- ***Capacity / Legality***

  The execution of the contract code automatically carry out the transfer of the digital asset in the same transaction as the payment.

“Smart contracts” conceptualized by Szabo in 1994

A smart contract is a ***computerized transaction protocol that executes the terms of a contract****.* *The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement),* ***minimize exceptions*** *both malicious and accidental, and* ***minimize the need for trusted intermediaries.*** *Related economic goals include* ***lowering fraud loss, arbitrations and enforcement costs****, and other transaction costs.*

-Nick Szabo “The Idea of Smart Contracts”

Questions

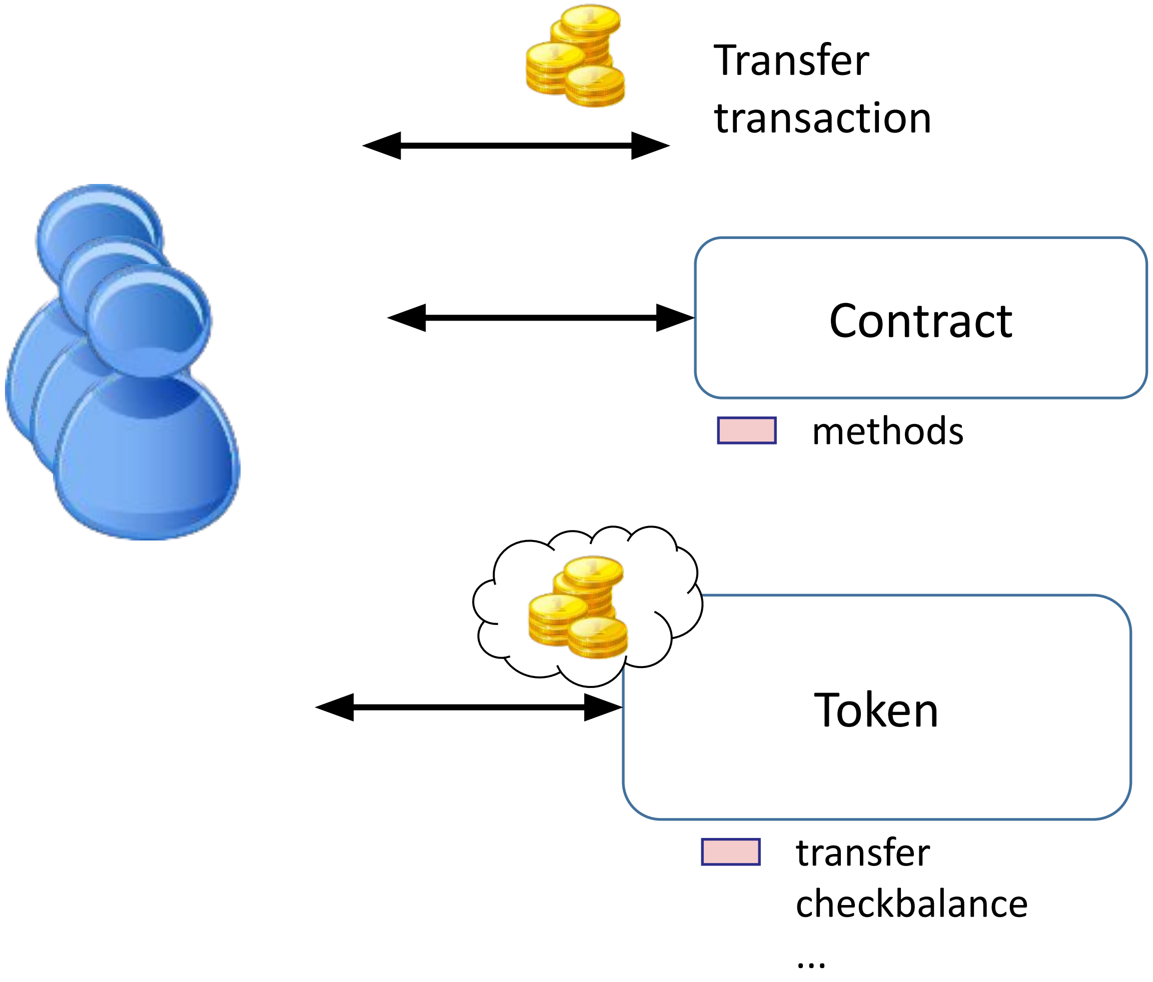
Consider the Dutch Auction smart contract.

How could we describe it based on the four elements of a legal contract?

How could we describe it based on Szabo’s smart contract objectives?

Part 6: Fungible and Non-Fungible Tokens on Ethereum

What are tokens?



Transfer transaction

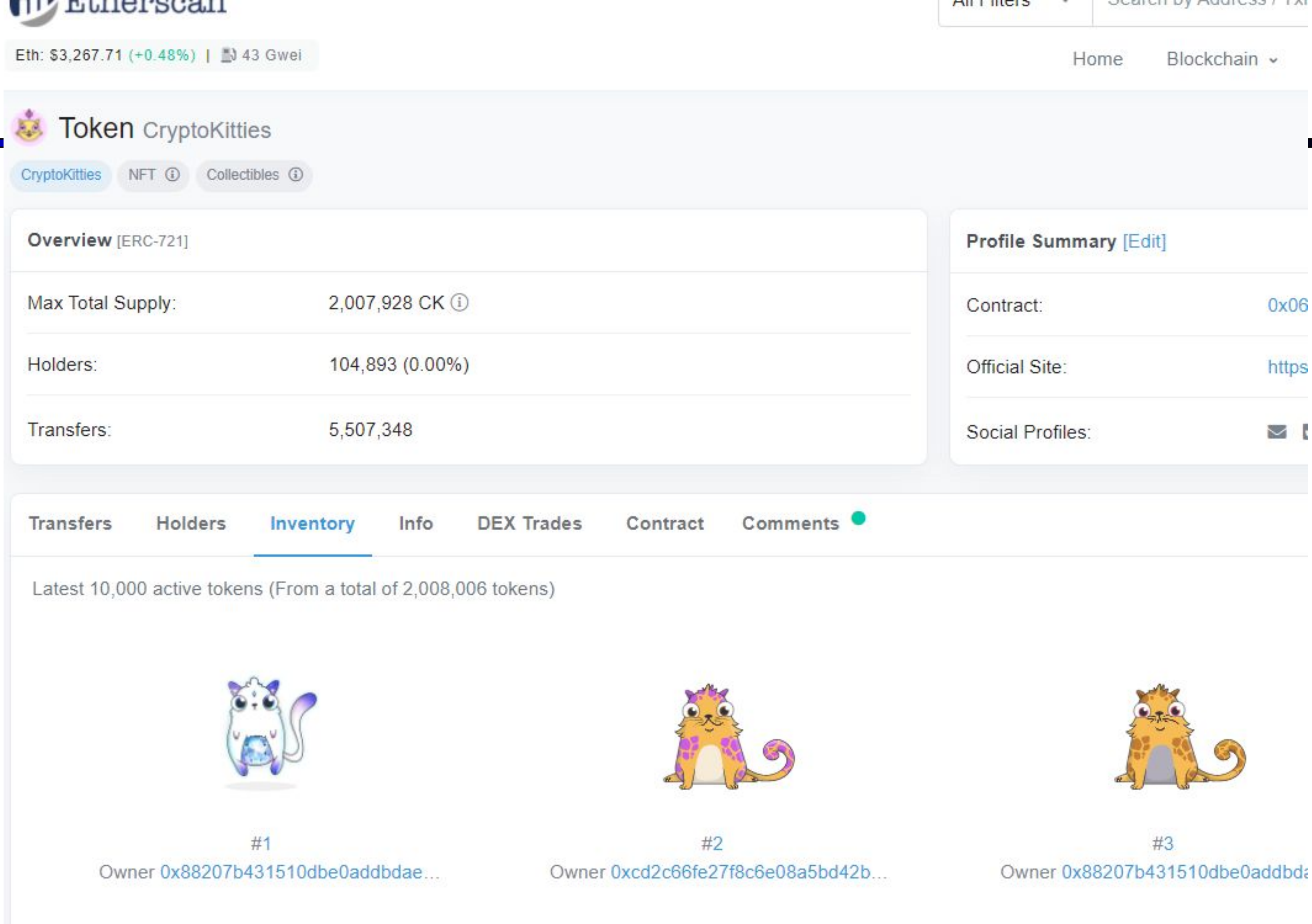
Contract

▭ methods

Token

▭ transfer
checkbalance
…

Tokens are smart contracts that function as digital assets



Eth: $3,267.71 (+0.48%) | 43 Gwei

Token CryptoKitties

Overview [ERC-721]

Max Total Supply:	2,007,928 CK
Holders:	104,893 (0.00%)
Transfers:	5,507,348

Profile Summary [Edit]

Contract:	0x06
Official Site:	https
Social Profiles:	

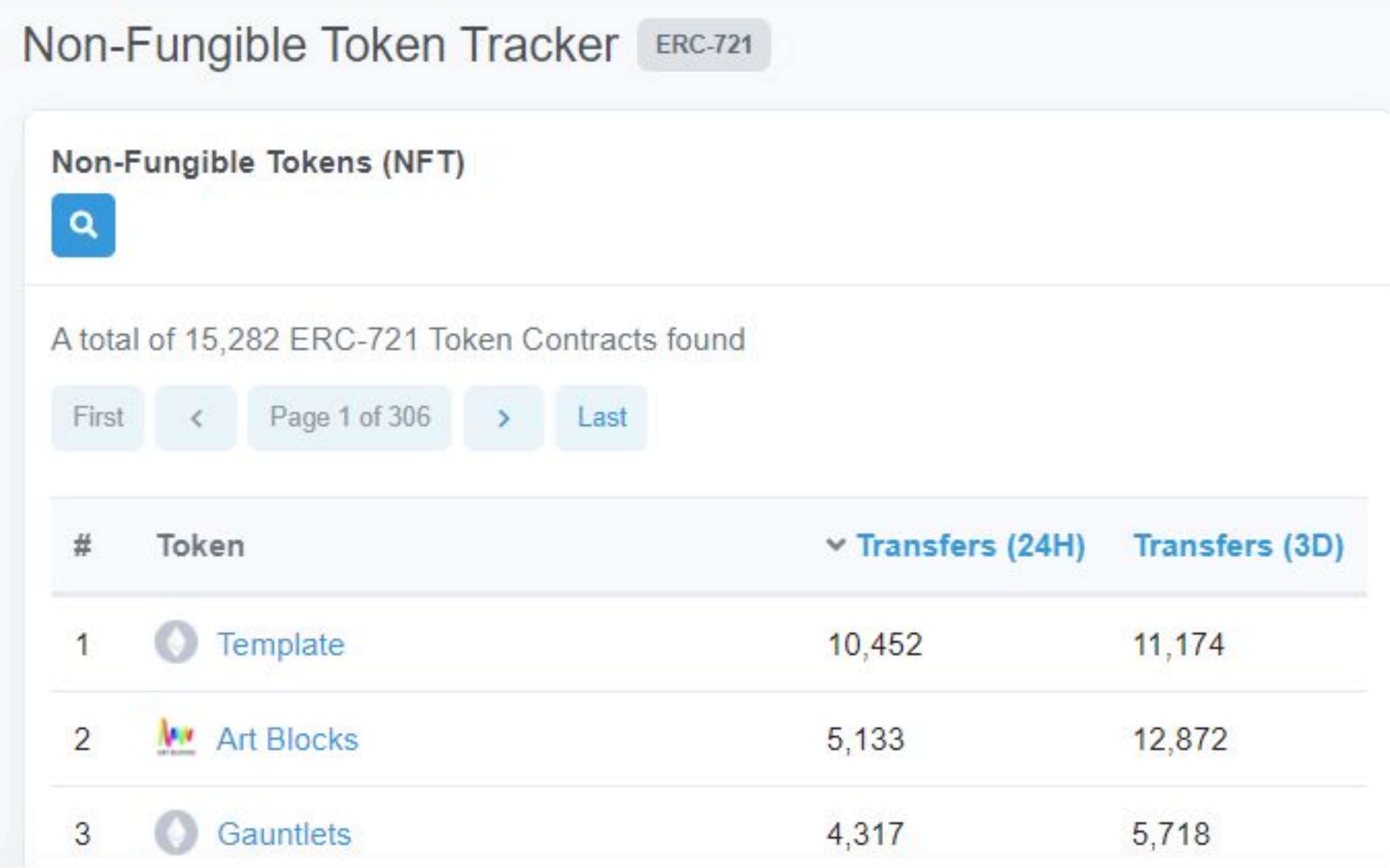
Latest 10,000 active tokens (From a total of 2,008,006 tokens)

#1
Owner 0x88207b431510dbe0addbdae...

#2
Owner 0xcd2c66fe27f8c6e08a5bd42b...

#3
Owner 0x88207b431510dbe0addbd

Following a standard means some functionality can be completely generic

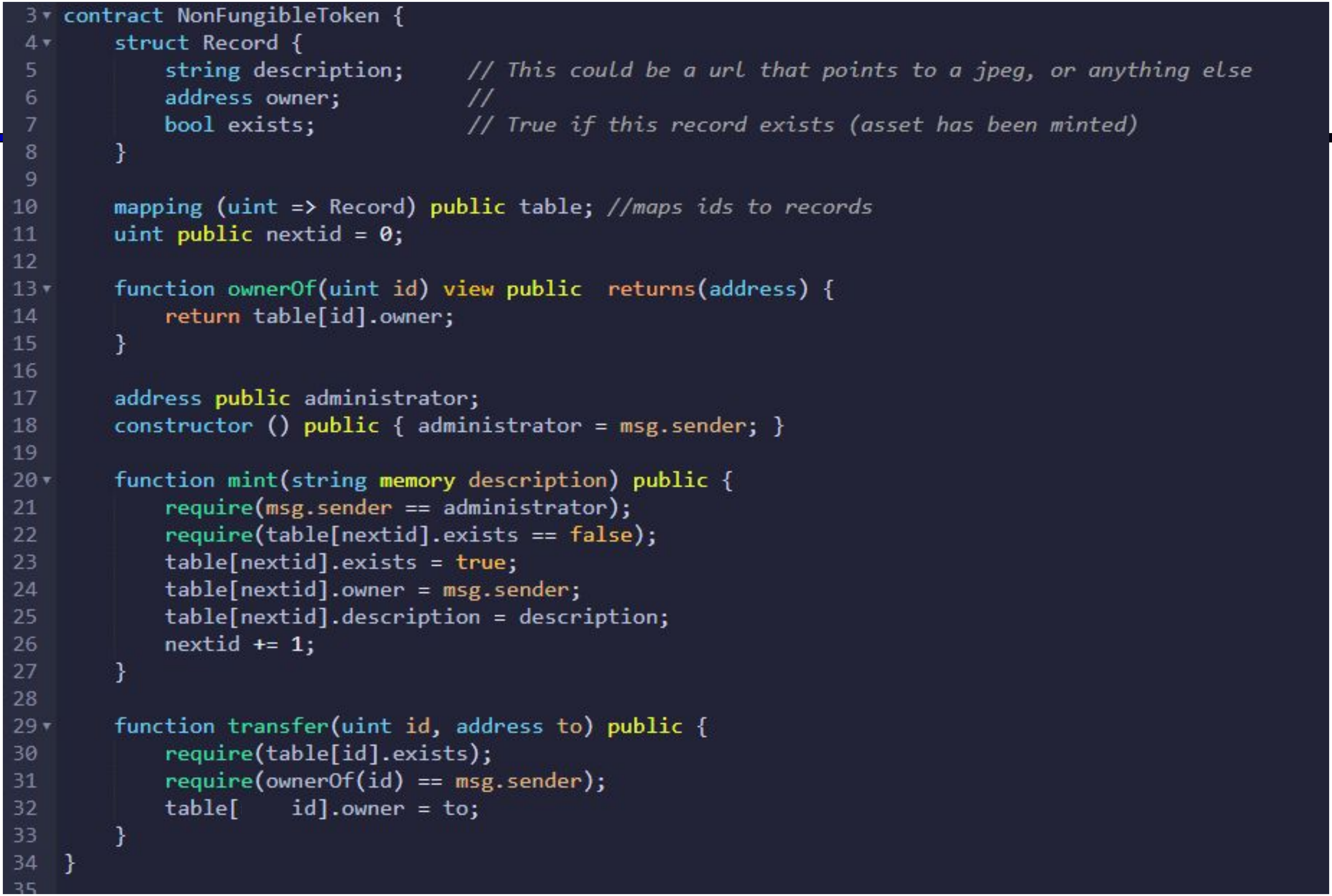


Non-Fungible Token Tracker ERC-721

Non-Fungible Tokens (NFT)

A total of 15,282 ERC-721 Token Contracts found

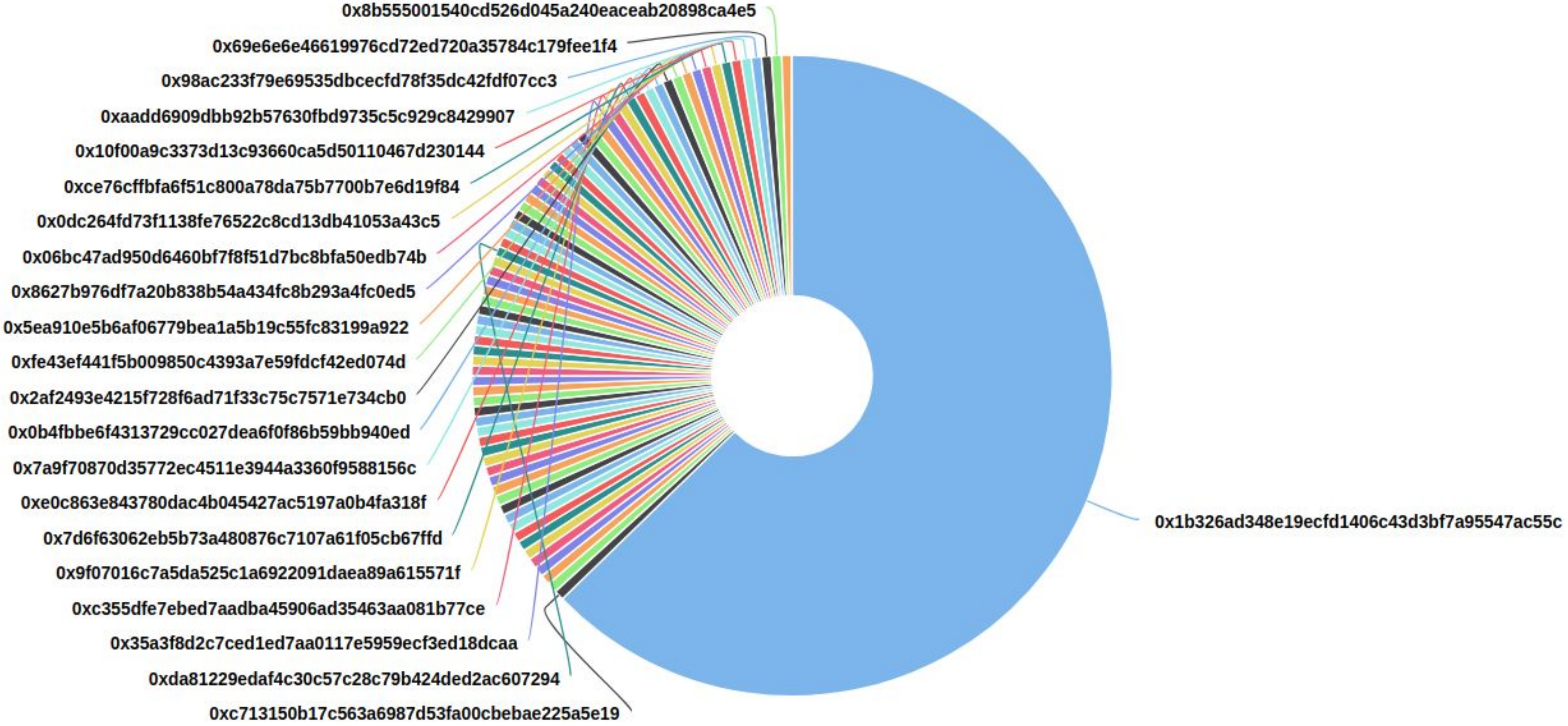
#	Token	Transfers (24H)	Transfers (3D)
1	Template	10,452	11,174
2	Art Blocks	5,133	12,872
3	Gauntlets	4,317	5,718



```
contract NonFungibleToken {
    struct Record {
        string description;     // This could be a url that points to a jpeg, or anything else
        address owner;          //
        bool exists;            // True if this record exists (asset has been minted)
    }

    mapping (uint => Record) public table; //maps ids to records
    uint public nextid = 0;

    function ownerOf(uint id) view public  returns(address) {
        return table[id].owner;
    }

    address public administrator;
    constructor () public { administrator = msg.sender; }

    function mint(string memory description) public {
        require(msg.sender == administrator);
        require(table[nextid].exists == false);
        table[nextid].exists = true;
        table[nextid].owner = msg.sender;
        table[nextid].description = description;
        nextid += 1;
    }

    function transfer(uint id, address to) public {
        require(table[id].exists);
        require(ownerOf(id) == msg.sender);
        table[    id].owner = to;
    }
}
```

ECE398SC test token 1 Top 100 Token Holders

Source: Etherscan.io



ERC20 defines interfaces for basic token behavior

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

Basic functionality:

```
function totalSupply() constant returns (uint256 totalSupply)

function balanceOf(address _owner) constant returns (uint256 balance)

function transfer(address _to, uint256 _value) returns (bool success)
```

Delegating control:

```
function transferFrom(address _from, address _to, uint256 _value) returns (bool success)

function approve(address _spender, uint256 _value) returns (bool success)

function allowance(address _owner, address _spender) constant returns (uint256 remaining)
```

To summarize

- Tokens are contracts that function like digital assets

- Difference between fungible and non-fungible

    Non-fungible: each asset in a series has a distinct ID, attributes

    Fungible: the assets are interchangeable, can be summed up

- Using standard interfaces for tokens help enable interoperability

    - ERC20/721 feature many additional features, approval mechanism for composing with other contracts

There are plenty ERC20 templates on the internet

This is a widely adopted standard, and so tons of tools/service will “just work” if you adhere to ERC20 standard

http://lmgtfy.com/?q=erc20+token+template

https://github.com/bitfwdcommunity/Issue-your-own-ERC20-token/blob/master/contracts/erc20_tutorial.sol

https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts/token/ERC20

Bonus: Ropsten / Metamask Run-through

Ropsten / Metamask Run-Through

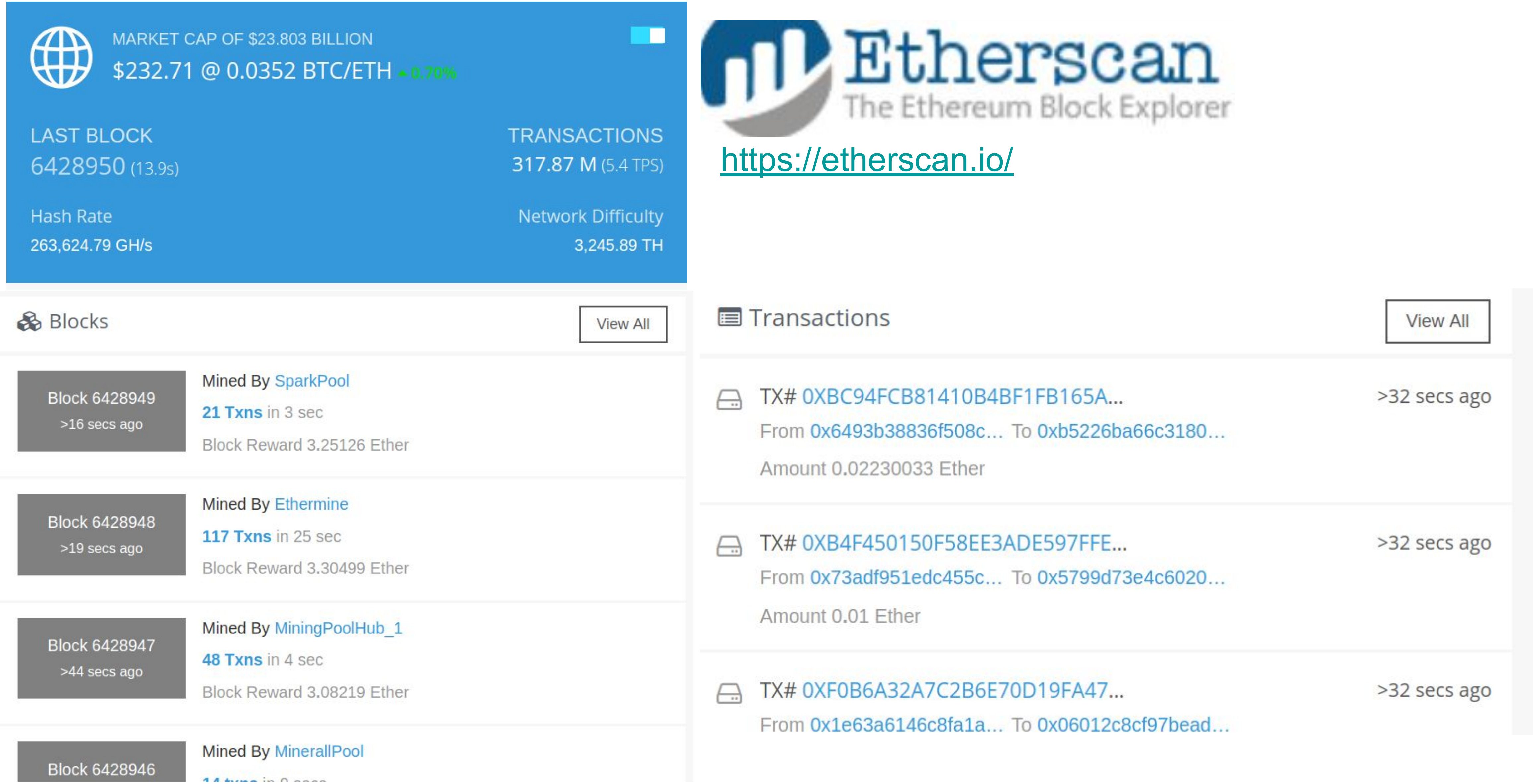
Beforehand - install Metamask

In this demo:

1. Create a new Ropsten (testnet) account in Metamask, copy the address
2. Visit the ropsten faucet, request Ether
3. View the transaction in Etherscan
4. Send a transaction to the instructor to complete the first challenge

Etherscan
The Ethereum Block Explorer

https://etherscan.io/



MARKET CAP OF $23.803 BILLION
$232.71 @ 0.0352 BTC/ETH ▲0.70%

LAST BLOCK
6428950 (13.9s)

TRANSACTIONS
317.87 M (5.4 TPS)

Hash Rate
263,624.79 GH/s

Network Difficulty
3,245.89 TH

Blocks

View All

Block 6428949
>16 secs ago
Mined By SparkPool
21 Txns in 3 sec
Block Reward 3.25126 Ether

Block 6428948
>19 secs ago
Mined By Ethermine
117 Txns in 25 sec
Block Reward 3.30499 Ether

Block 6428947
>44 secs ago
Mined By MiningPoolHub_1
48 Txns in 4 sec
Block Reward 3.08219 Ether

Block 6428946
Mined By MinerallPool
14 txns in 9 secs

Transactions

View All

TX# 0XBC94FCB81410B4BF1FB165A... >32 secs ago
From 0x6493b38836f508c… To 0xb5226ba66c3180…
Amount 0.02230033 Ether

TX# 0XB4F450150F58EE3ADE597FFE... >32 secs ago
From 0x73adf951edc455c… To 0x5799d73e4c6020…
Amount 0.01 Ether

TX# 0XF0B6A32A7C2B6E70D19FA47... >32 secs ago
From 0x1e63a6146c8fa1a… To 0x06012c8cf97bead…

Several links for creating a ropsten wallet

METAMASK

MyEtherWallet

Mnemonic Code Converter

Get testnet Ether from the faucet

MetaMask Ether Faucet

Ethereum Ropsten Faucet

Send some tETH (any amount) the instructor:

0x0974d3A22bDB7f73dCAb552a71896A2150DD2346

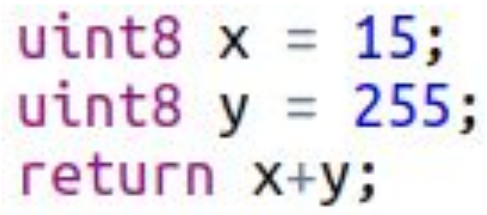
Basic datatypes available in Solidity

Solidity is statically typed, like C or Java, but unlike python and javascript

Integers:

int, int8, int16, …, int256

uint, uint8, uint16, …, uint256



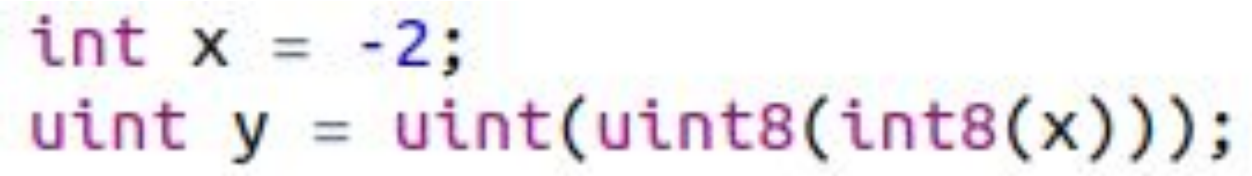
```
uint8 x = 15;
uint8 y = 255;
return x+y;
```

Integer Conversions in Solidity

- Syntax most similar to python, but the behavior is like C

- Some restrictions on integer conversions, only change sign or size in one conversion

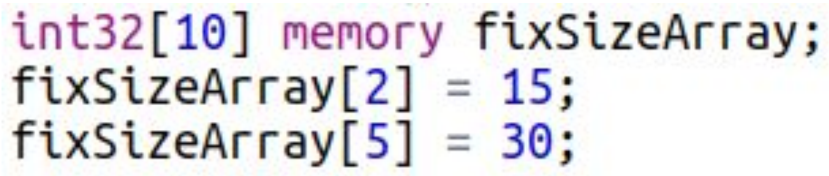
Question: what value will y take?



```
int x = -2;
uint y = uint(uint8(int8(x)));
```

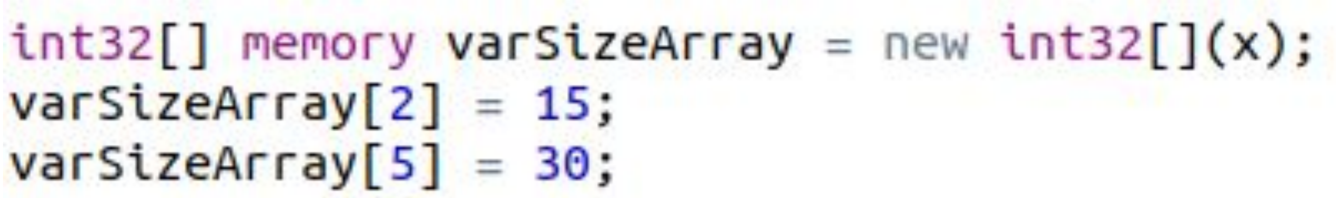
Arrays and lists in Solidity

Statically sized array:



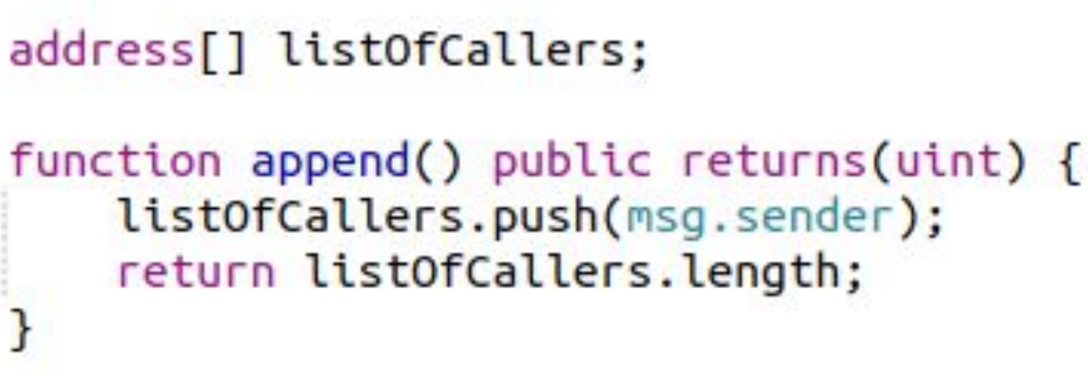
```
int32[10] memory fixSizeArray;
fixSizeArray[2] = 15;
fixSizeArray[5] = 30;
```

Dynamic length array:
(more expensive,
still can’t change once created)



```
int32[] memory varSizeArray = new int32[](x);
varSizeArray[2] = 15;
varSizeArray[5] = 30;
```

Array in storage:
(persists across
transactions)



```
address[] listOfCallers;

function append() public returns(uint) {
    listOfCallers.push(msg.sender);
    return listOfCallers.length;
}
```

Basic datatypes available in Solidity

Strings and Bytes:

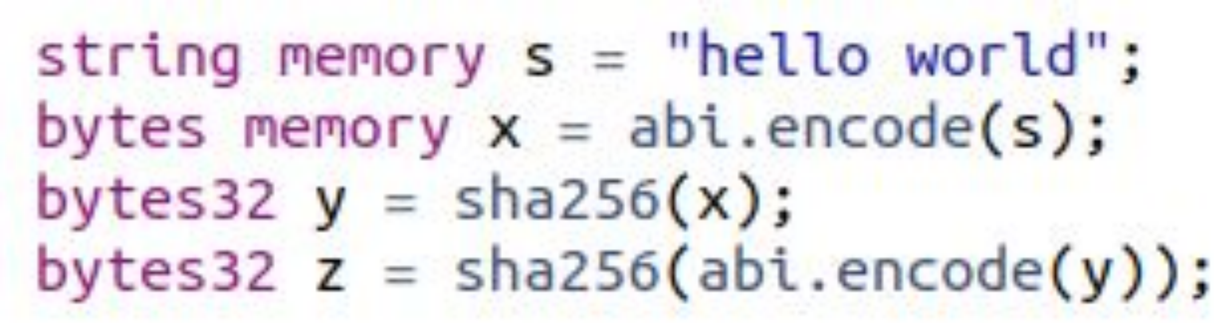
bytes32: fixed size, returned by hash functions

bytes memory: array of bytes

string memory: array of characters

abi.encode( ): flattens multiple arguments to a *bytes*

Fancier string libraries are available too



```
string memory s = "hello world";
bytes memory x = abi.encode(s);
bytes32 y = sha256(x);
bytes32 z = sha256(abi.encode(y));
```