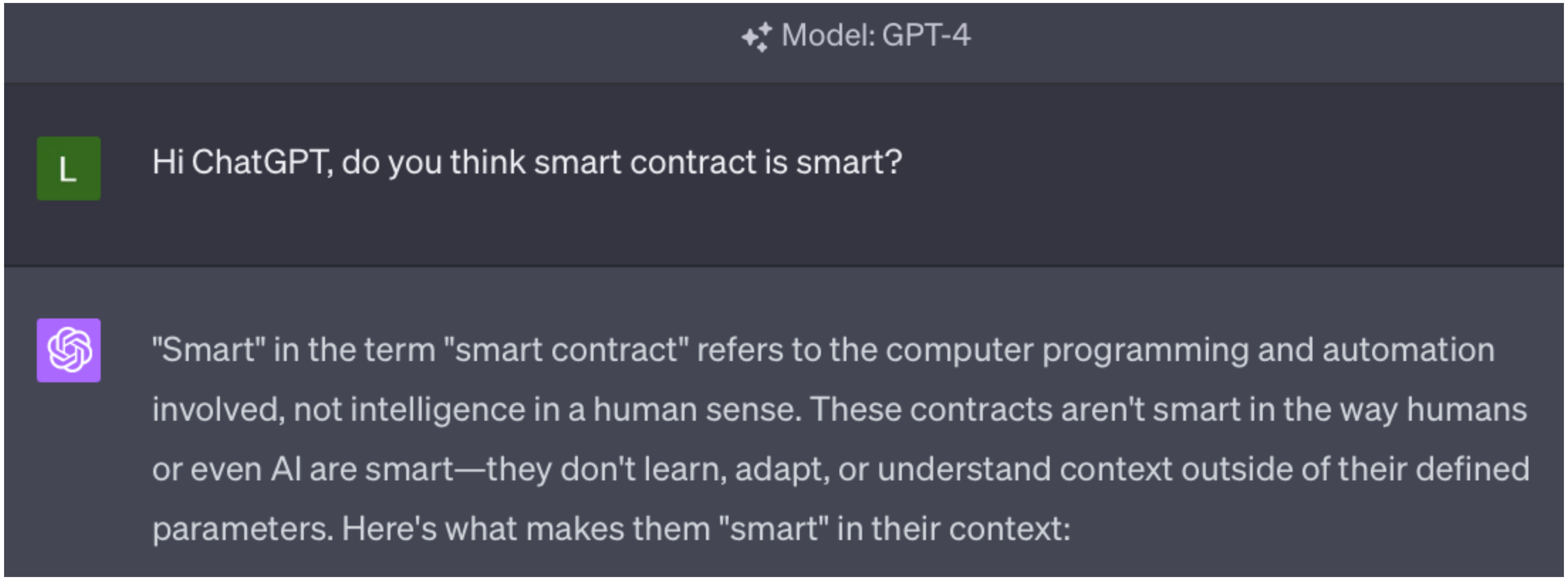
UZH Summer School
Smart Contracts

Liyi Zhou

Outlook

- An Introduction to Smart Contracts
  - What are smart contracts?
  - (Recap) State Transition
- Basic Solidity Programming in Ethereum
  - Implementing a Voting System
  - Tokens
  - Token Exchange
- Smart Contract Limits
  - Delegation and Proxy Contracts
  - the Need for Oracles

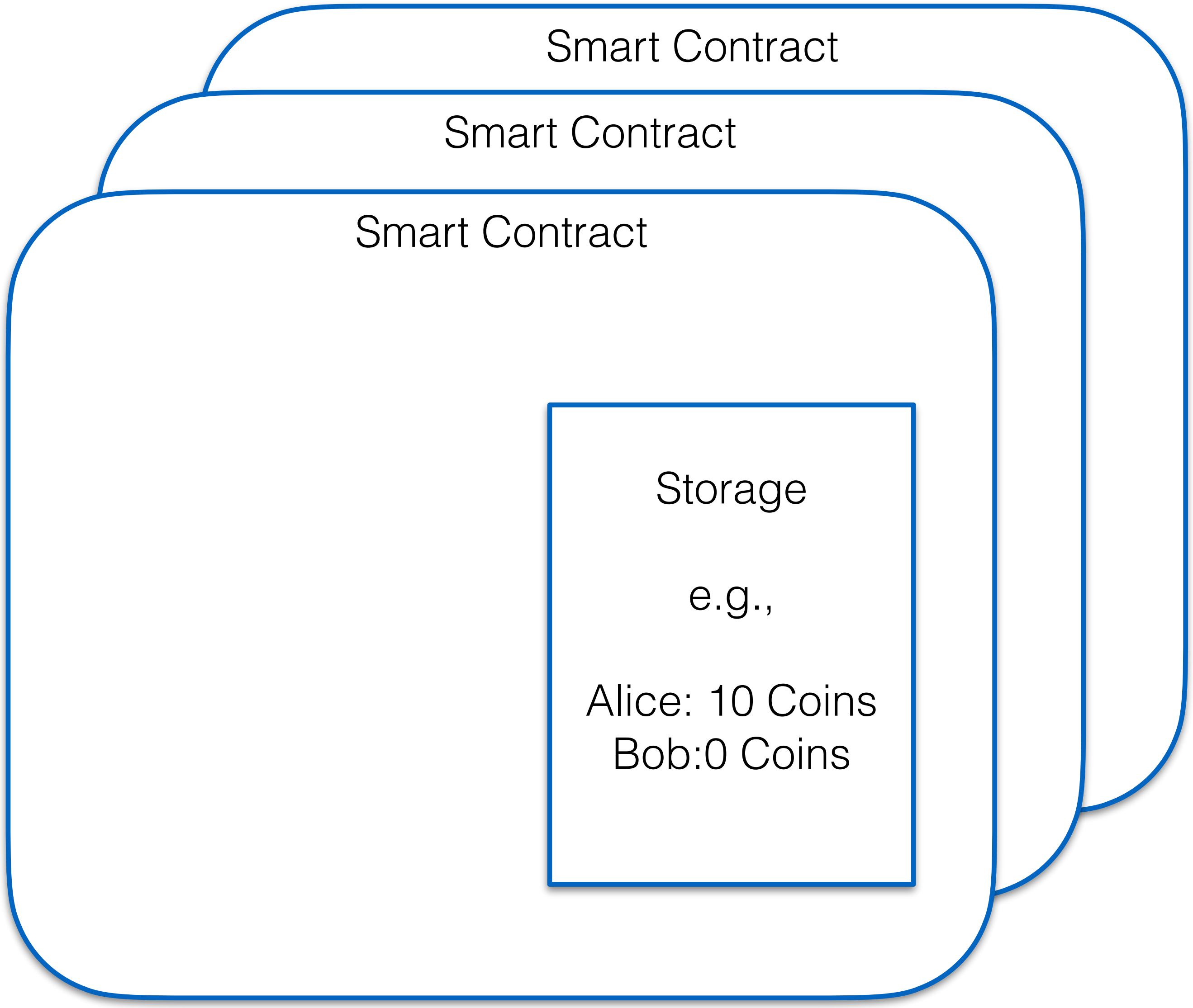
What are smart contracts?



Hi ChatGPT, do you think smart contract is smart?

"Smart" in the term "smart contract" refers to the computer programming and automation involved, not intelligence in a human sense. These contracts aren't smart in the way humans or even AI are smart—they don't learn, adapt, or understand context outside of their defined parameters. Here's what makes them "smart" in their context:

Smart Contracts —> Programs running on blockchain



Smart Contract

Smart Contract

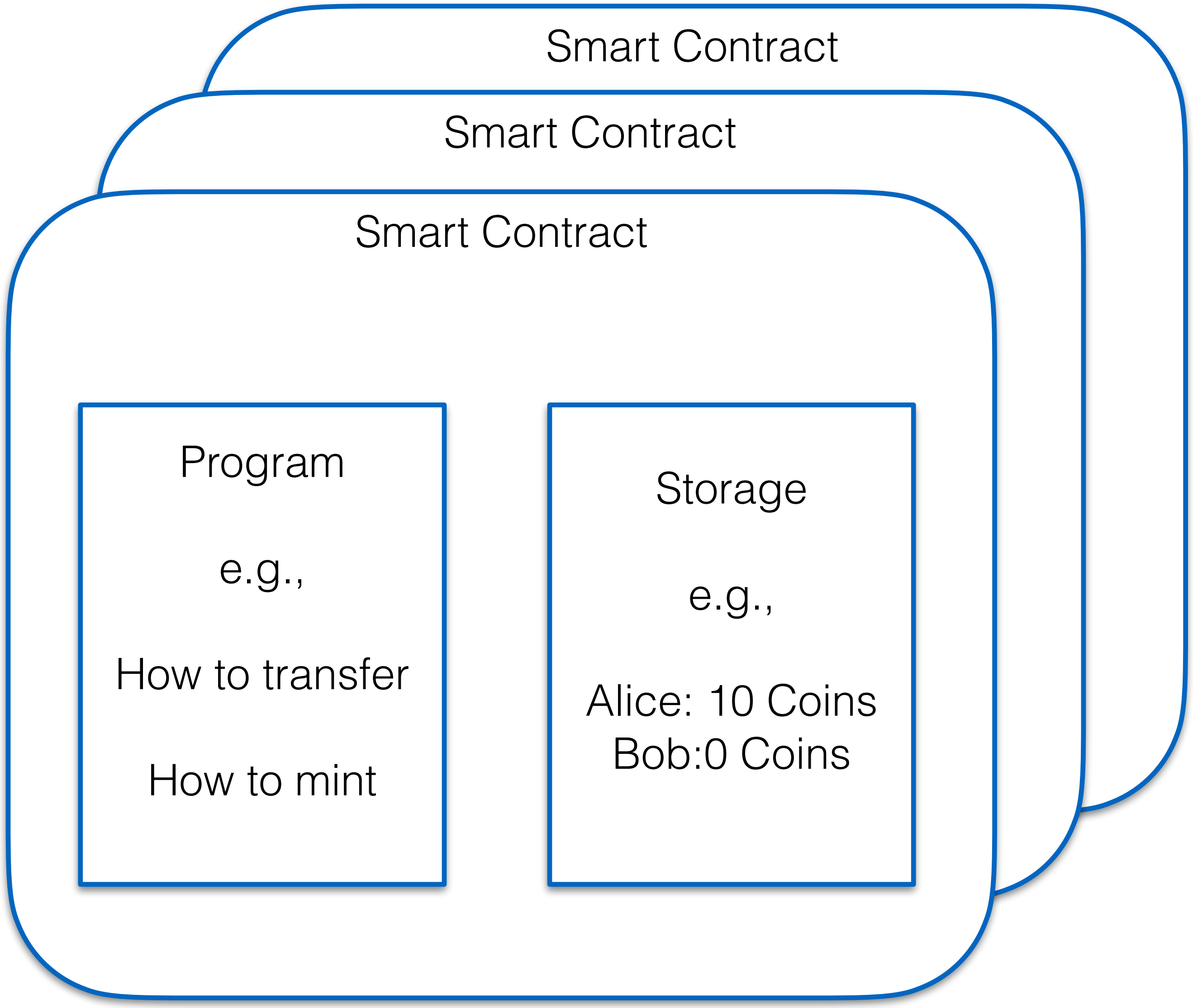
Smart Contract

Storage

e.g.,

Alice: 10 Coins
Bob:0 Coins

Smart Contracts —> Programs running on blockchain



Smart Contract

Smart Contract

Smart Contract

Program

e.g.,

How to transfer

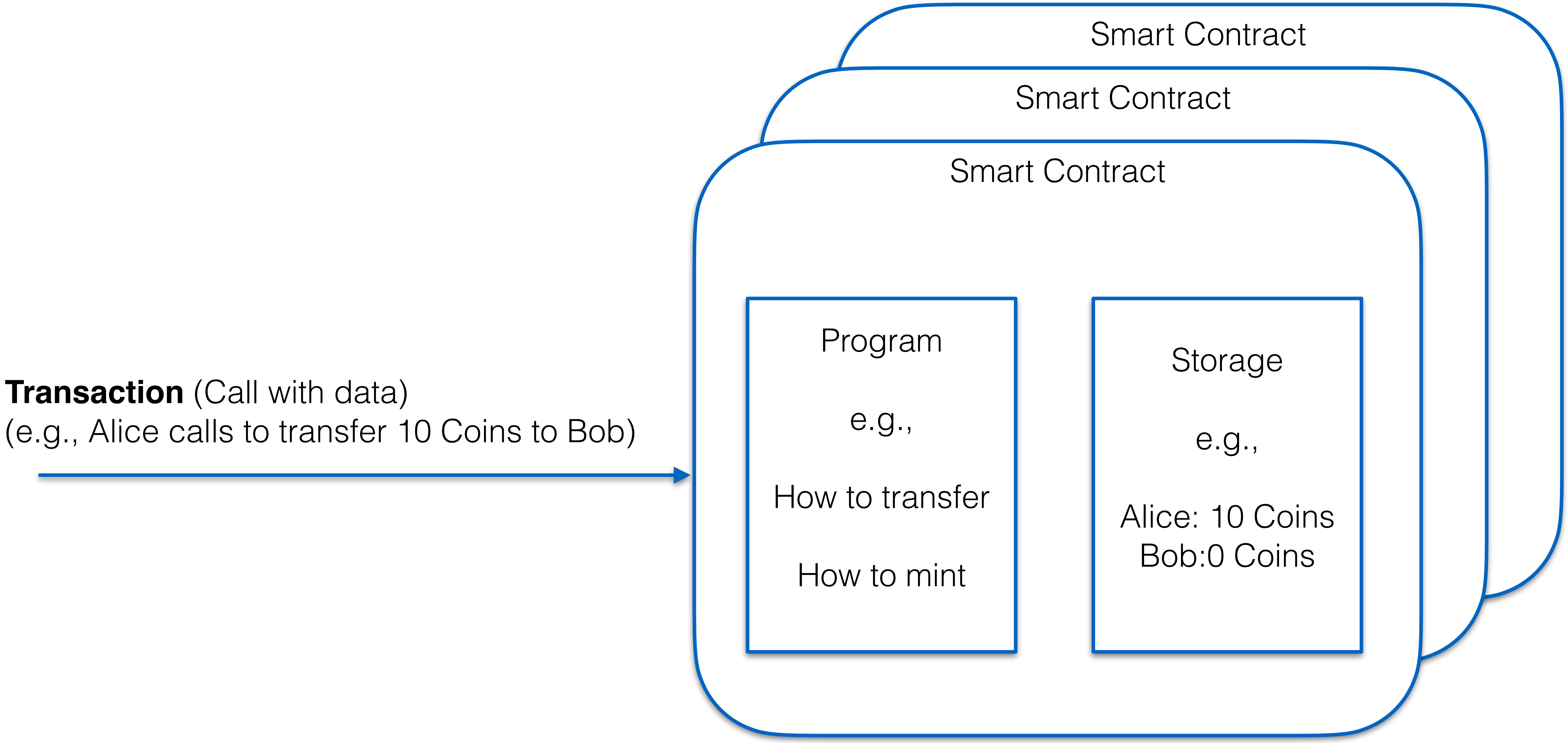
How to mint

Storage

e.g.,

Alice: 10 Coins
Bob:0 Coins

Smart Contracts —> Programs running on blockchain



Transaction (Call with data)
(e.g., Alice calls to transfer 10 Coins to Bob)

Smart Contract

Smart Contract

Smart Contract

Program

e.g.,

How to transfer

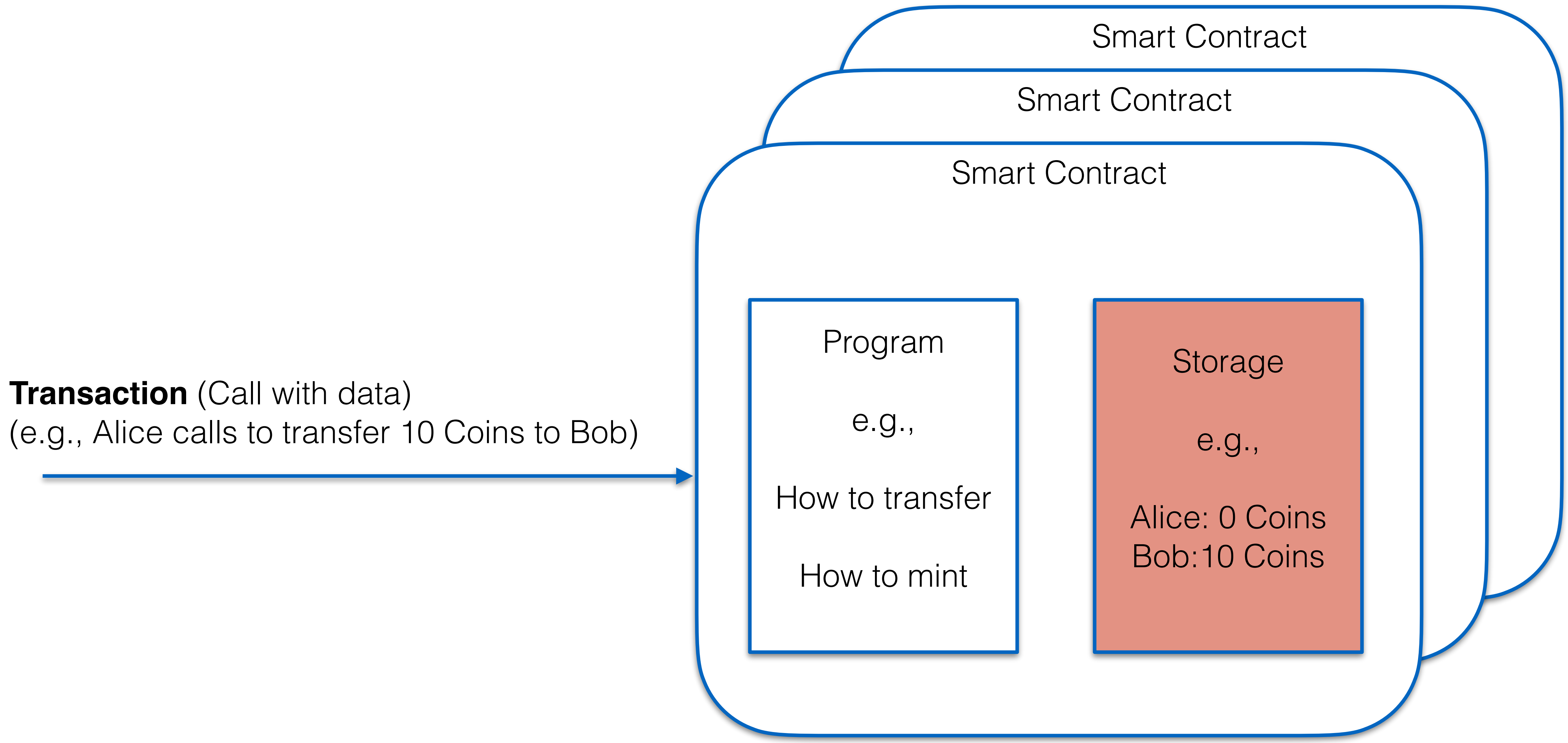
How to mint

Storage

e.g.,

Alice: 10 Coins
Bob:0 Coins

Smart Contracts —> Programs running on blockchain



Transaction (Call with data)
(e.g., Alice calls to transfer 10 Coins to Bob)

Smart Contract

Smart Contract

Smart Contract

Program

e.g.,

How to transfer

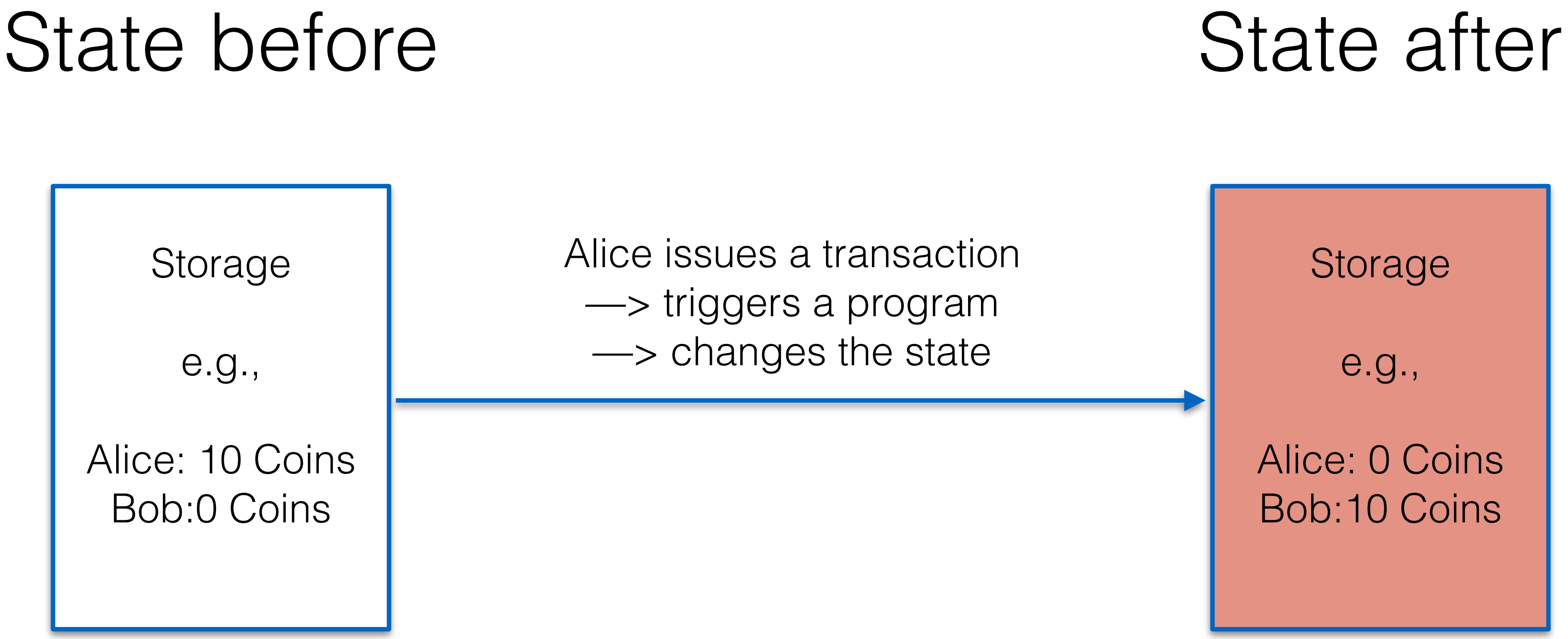
How to mint

Storage

e.g.,

Alice: 0 Coins
Bob:10 Coins

State Transition



State before

Storage

e.g.,

Alice: 10 Coins
Bob:0 Coins

Alice issues a transaction
—> triggers a program
—> changes the state

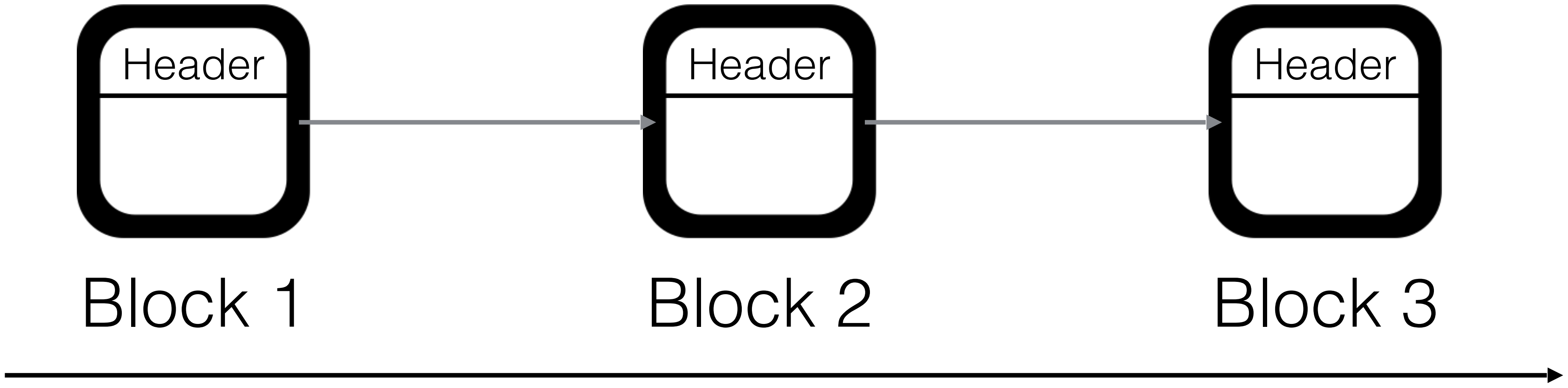
State after

Storage

e.g.,

Alice: 0 Coins
Bob:10 Coins

Store Explicit State



Consensus (nonce):		0xab	0xbv
State change:		Transaction 1 A —> B, 3	Transaction 2 B —> C, 2
State commitment:	{A:50}	{A:47, B:3}	{A:47, B:1, C:2}

Transaction 3
C —> D, 1

Advantages of explicit state storage

1. No need to go through whole history
2. State Transition between any two blocks can be verified
3. Light clients can sync up quickly

Ethereum —> A Universal State Replication Machine

- A world computer

  - Consensus among all nodes about the execution (“Replicated”)

- Quasi-Turing complete language
  - Execution halts if gas (transaction fee) is exhausted

Generic State Transition

- Set of possible states: S
- Set of possible inputs: I
- Set of possible outputs: O

Arbitrary program calls

- Transition function f: S × I → S × O

Execute programs

- Start state s ∈ S (genesis block)

credits to Joseph Bonneau,
https://docs.google.com/presentation/d/1NZlzs-Hy3f7H2Hqh61BEc6wA19KD5UTirvLq_JBta18

Ethereum State Transition

- **States S** = a map from address to state

address	**code**	**storage**	**balance**	**nonce**

- **Inputs I** (transactions)

from	**sig**	**nonce**	**to**	**data**	**value**	**gaslimit**	**gas parameters**

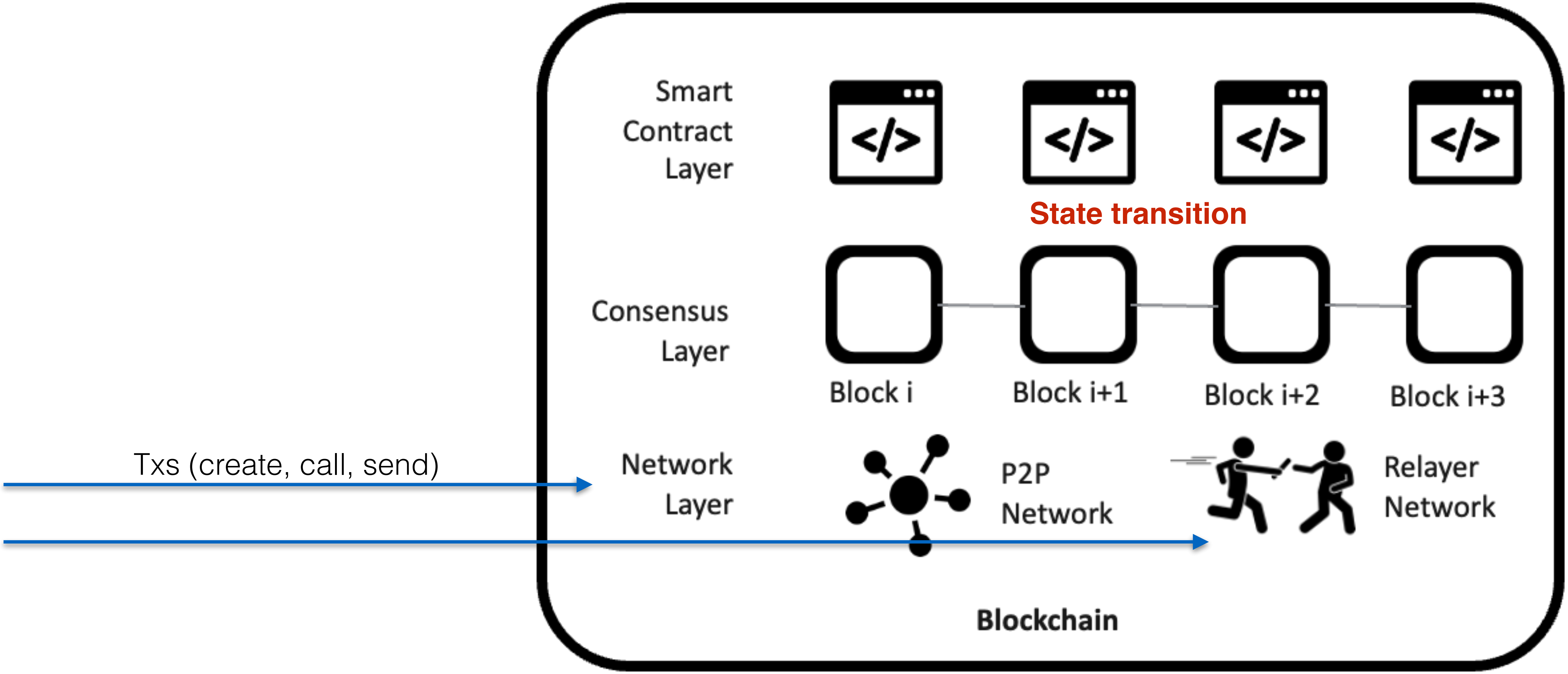
- **Transition f**:
  - Validate signature, nonce
  - Execute code (from, data, value, gaslimit, gasprice)

- Start state: ∅

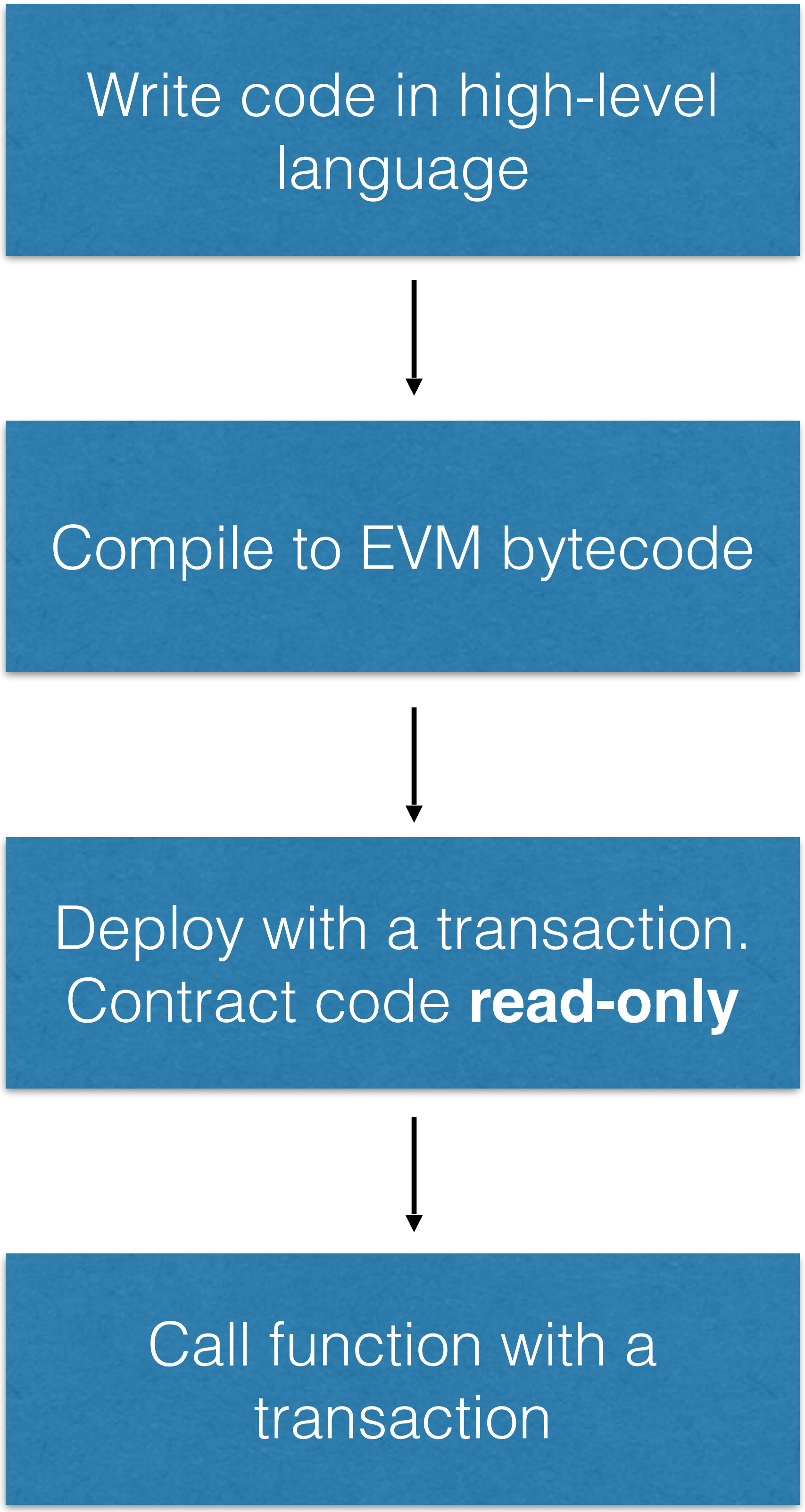
Three types of (state changing) transactions in Ethereum

type	from	sig	nonce	to	data	value	gaslimit	gas parameters
Send	**Sender**	**Sig**	**Nonce**	**Receiver**	**-**	**Amount**	**21000**	**?**
Create	**Creator**	**Sig**	**Nonce**	**-**	**Code**	**Start Balance**	**?**	**?**
Call	**Caller**	**Sig**	**Nonce**	**Contract**	**F, args**	**Amount**	**?**	**?**

Short Summary



Smart Contract Lifecycle



Write code in high-level language

Compile to EVM bytecode

Deploy with a transaction. Contract code **read-only**

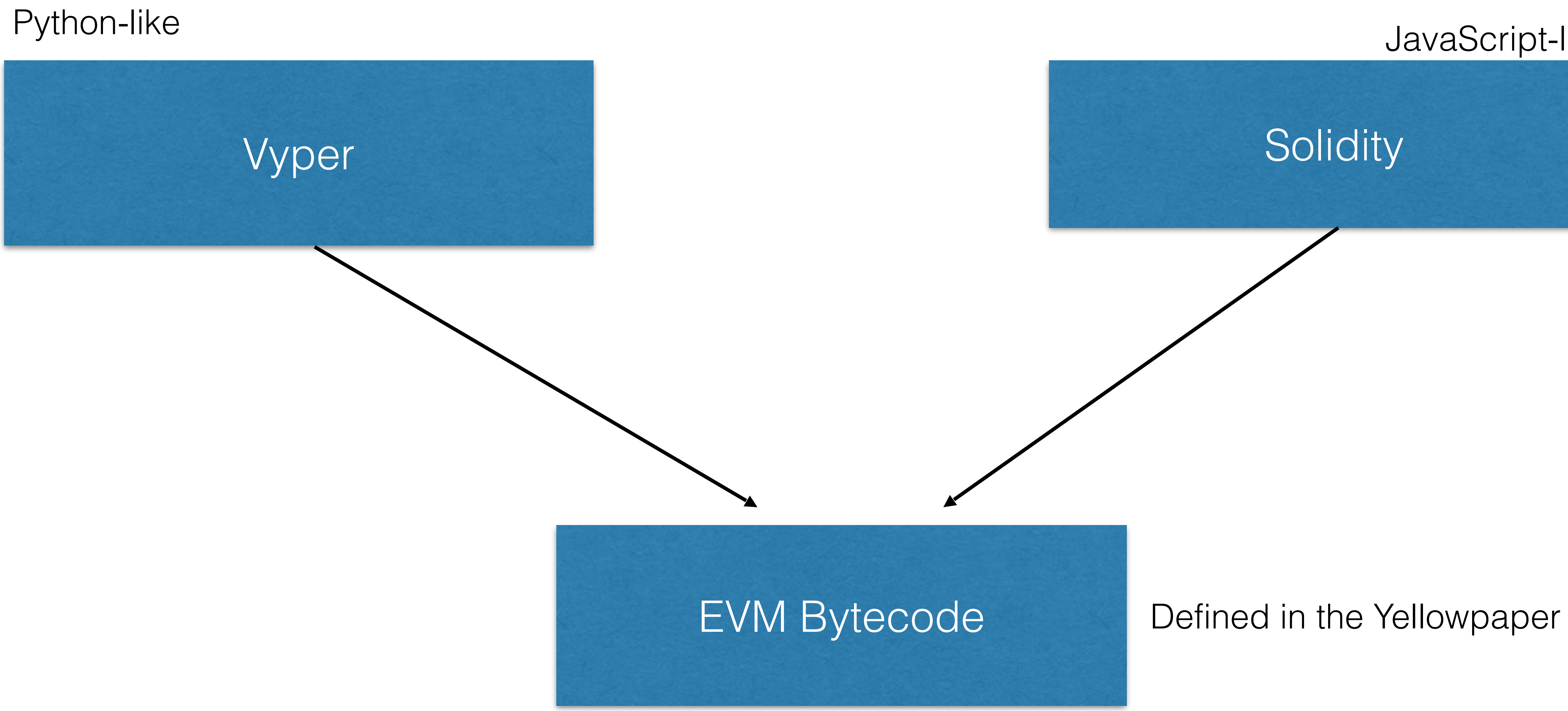
Call function with a transaction

- Lives forever if no destruct defined

  —> no keep-alive cost

- Contracts can call other contracts

Smart Contract Development



Python-like

Vyper

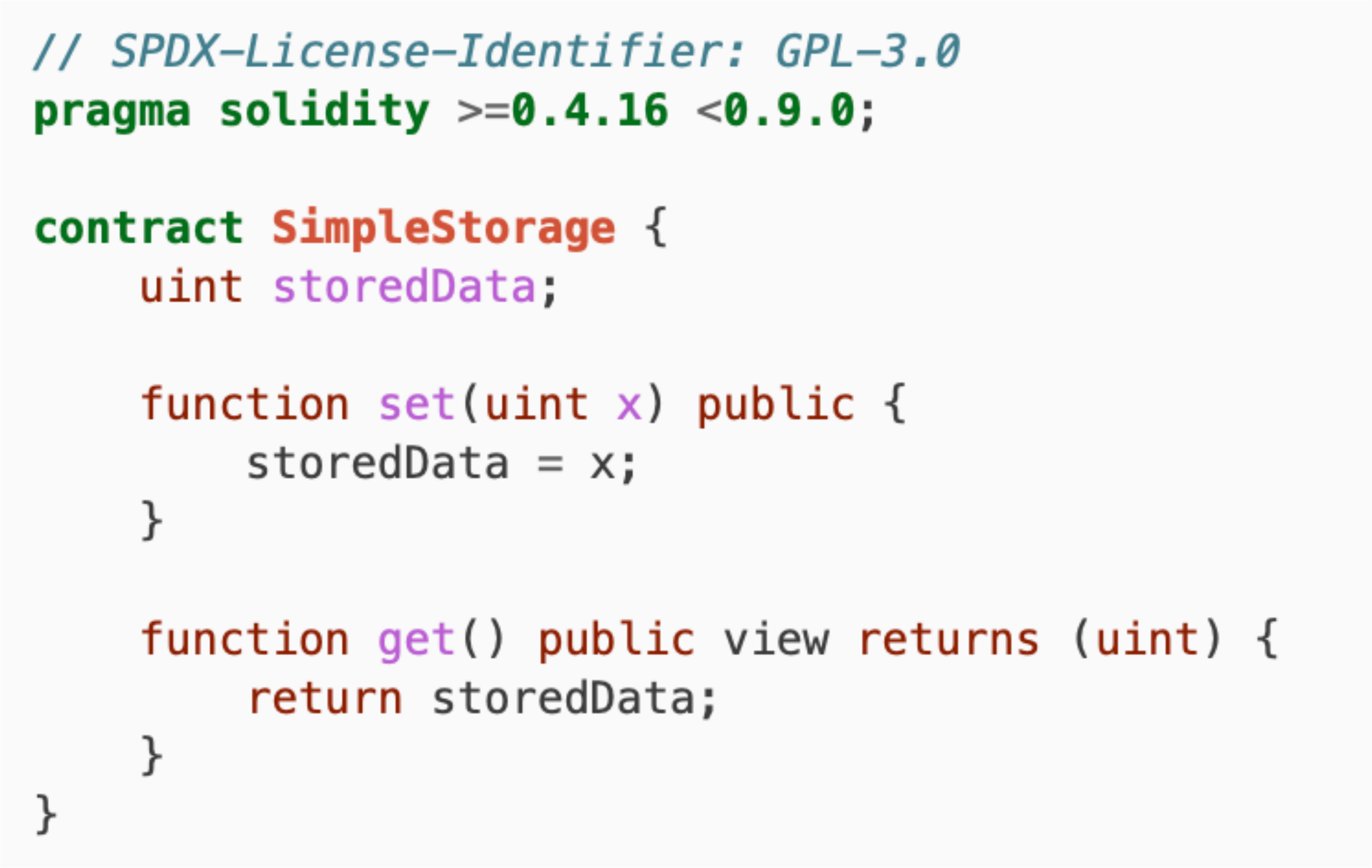
JavaScript-like

Solidity

EVM Bytecode

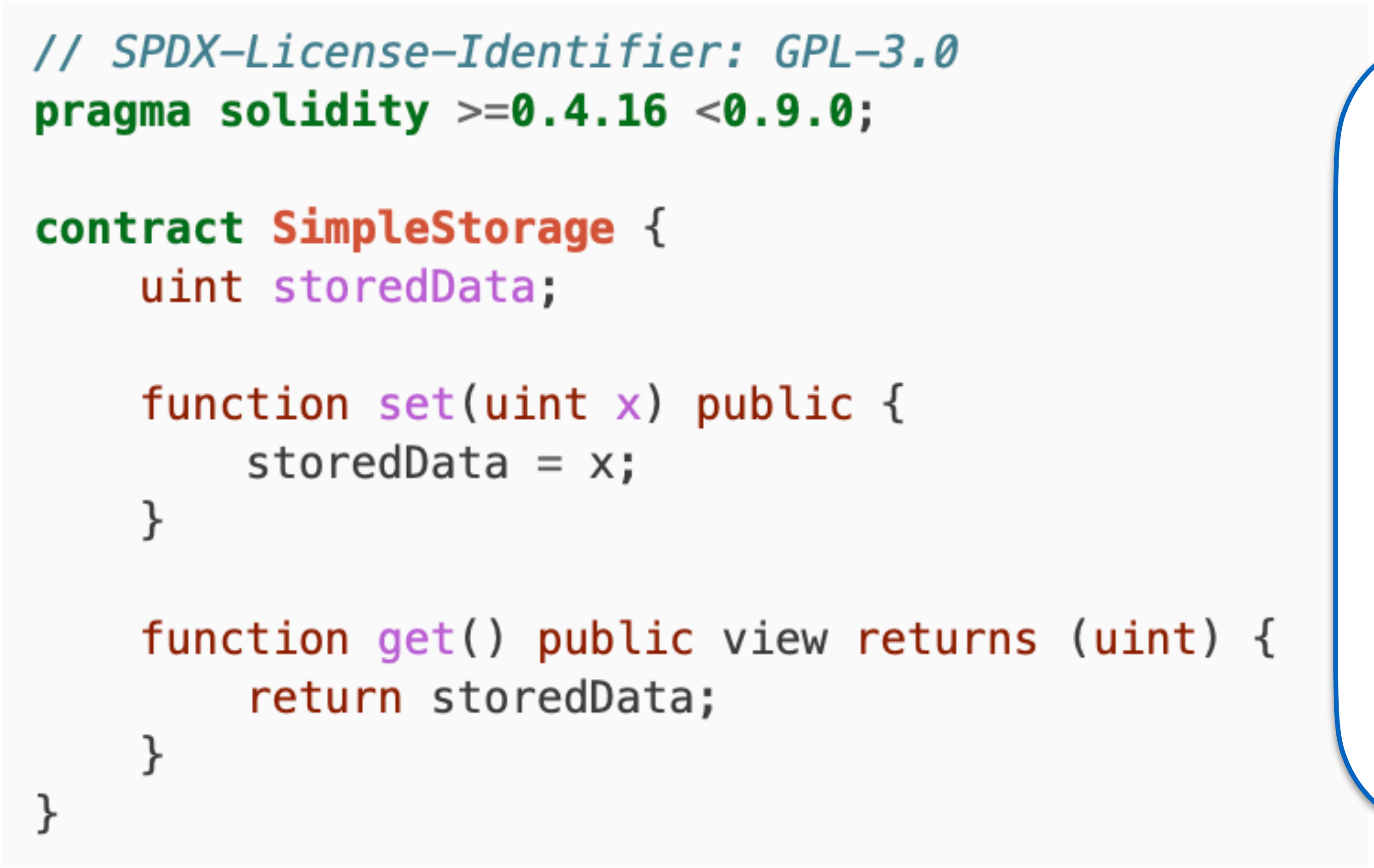
Defined in the Yellowpaper

Simple Example 1 (Remix Demo)

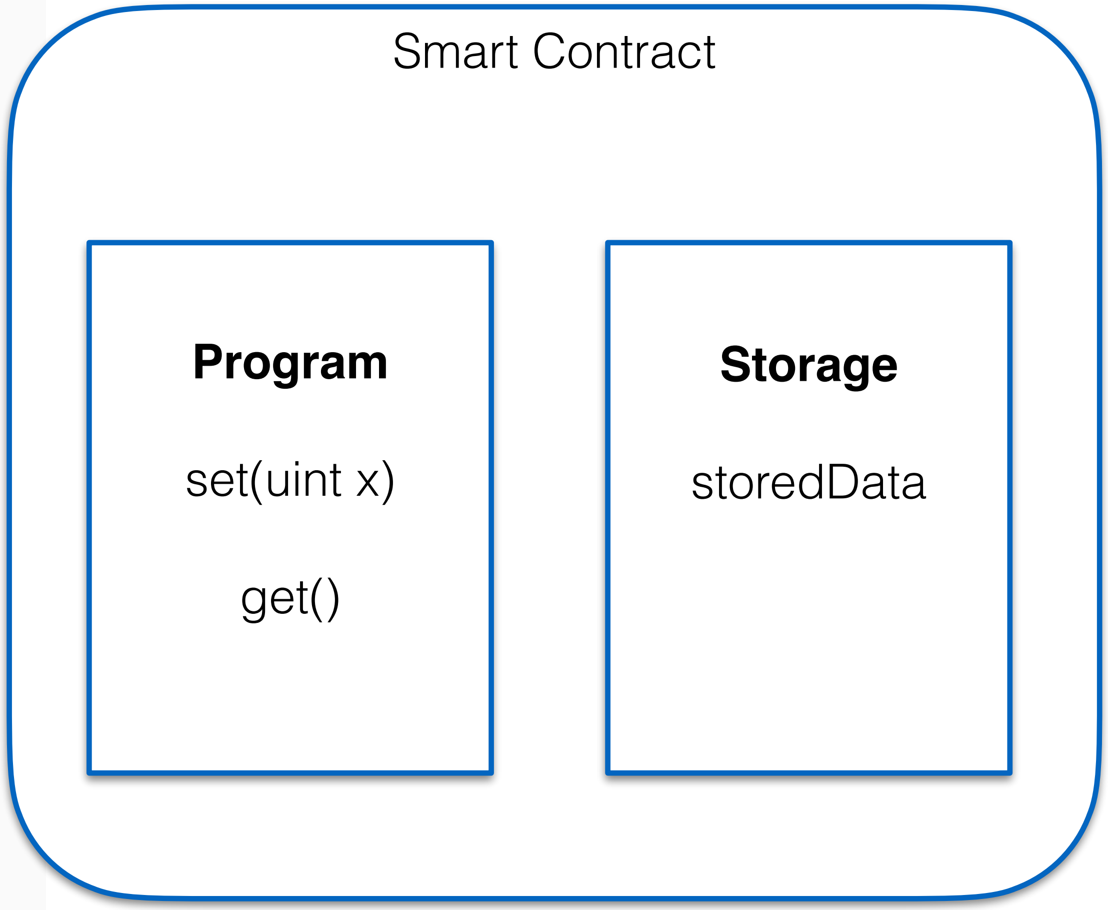


```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

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

    function get() public view returns (uint) {
        return storedData;
    }
}
```

Simple Example 1 (Remix Demo)



```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.16 <0.9.0;

contract SimpleStorage {
    uint storedData;

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

    function get() public view returns (uint) {
        return storedData;
    }
}
```



Smart Contract

Program

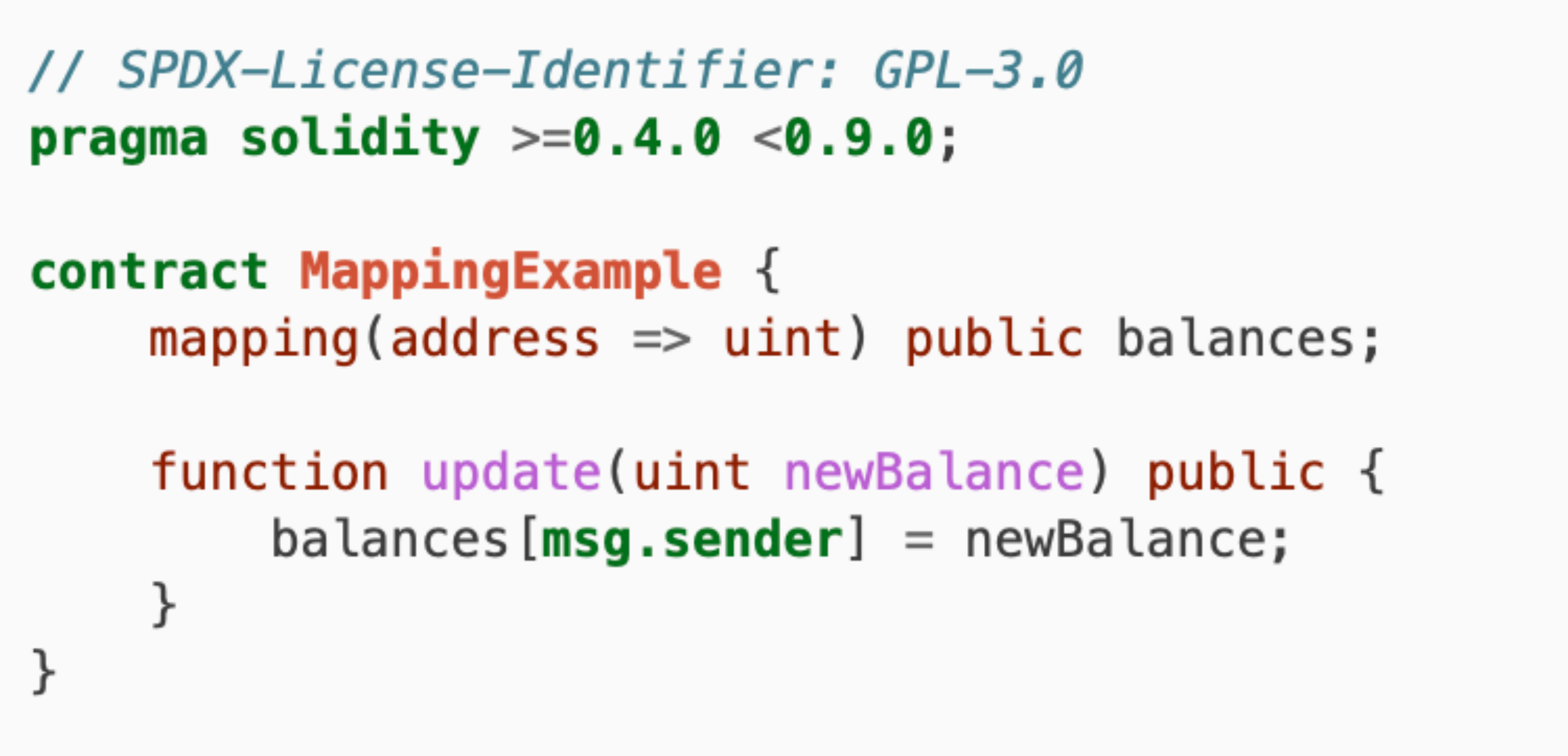
set(uint x)

get()

Storage

storedData

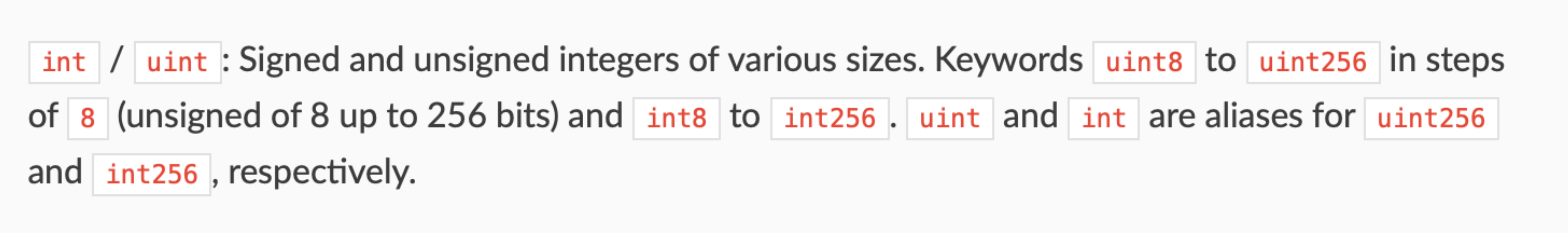
Simple Example 2



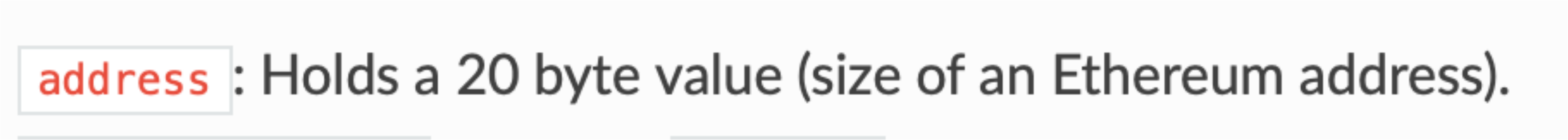
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.4.0 <0.9.0;

contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
```

Solidity Data Types

- Statically typed – like C, Rust, etc.
- Example:



`int` / `uint`: Signed and unsigned integers of various sizes. Keywords `uint8` to `uint256` in steps of `8` (unsigned of 8 up to 256 bits) and `int8` to `int256`. `uint` and `int` are aliases for `uint256` and `int256`, respectively.

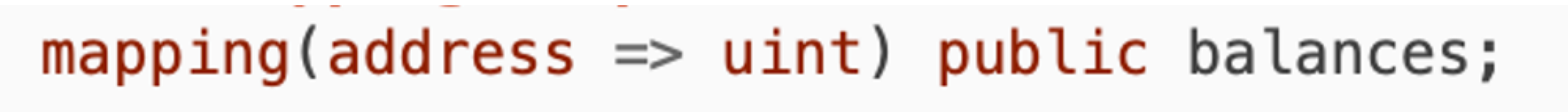


`address`: Holds a 20 byte value (size of an Ethereum address).

https://docs.soliditylang.org/en/v0.8.20/types.html

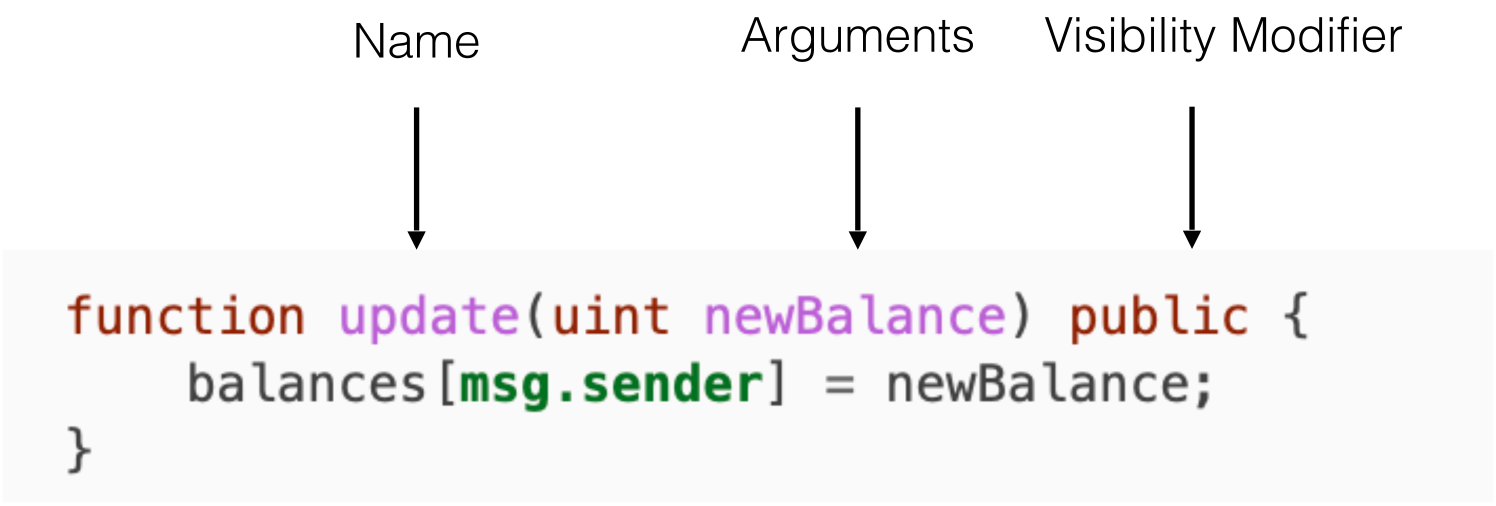
Mapping Data Types

- Mapping: a key value storage / hash table
- Every key is initially mapped to zero
- There is no built-in way to query the length of a mapping, or iterate over its non-zero elements



```
mapping(address => uint) public balances;
```

Functions

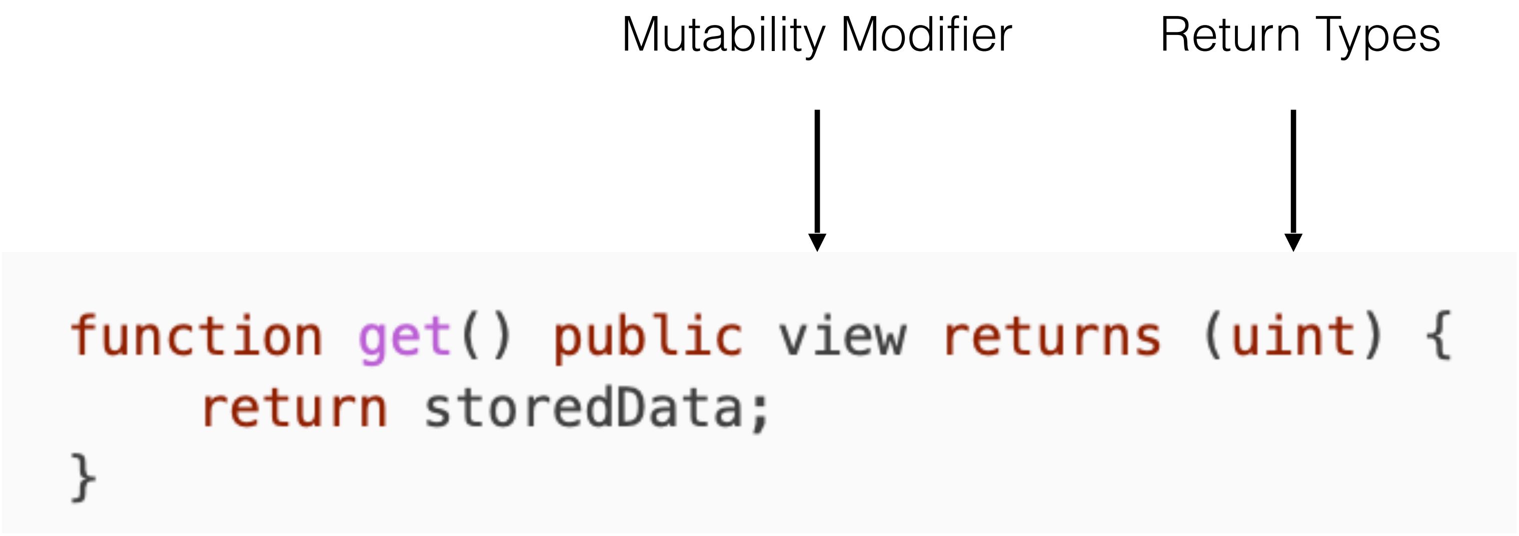


Name

Arguments

Visibility Modifier

```
function update(uint newBalance) public {
    balances[msg.sender] = newBalance;
}
```

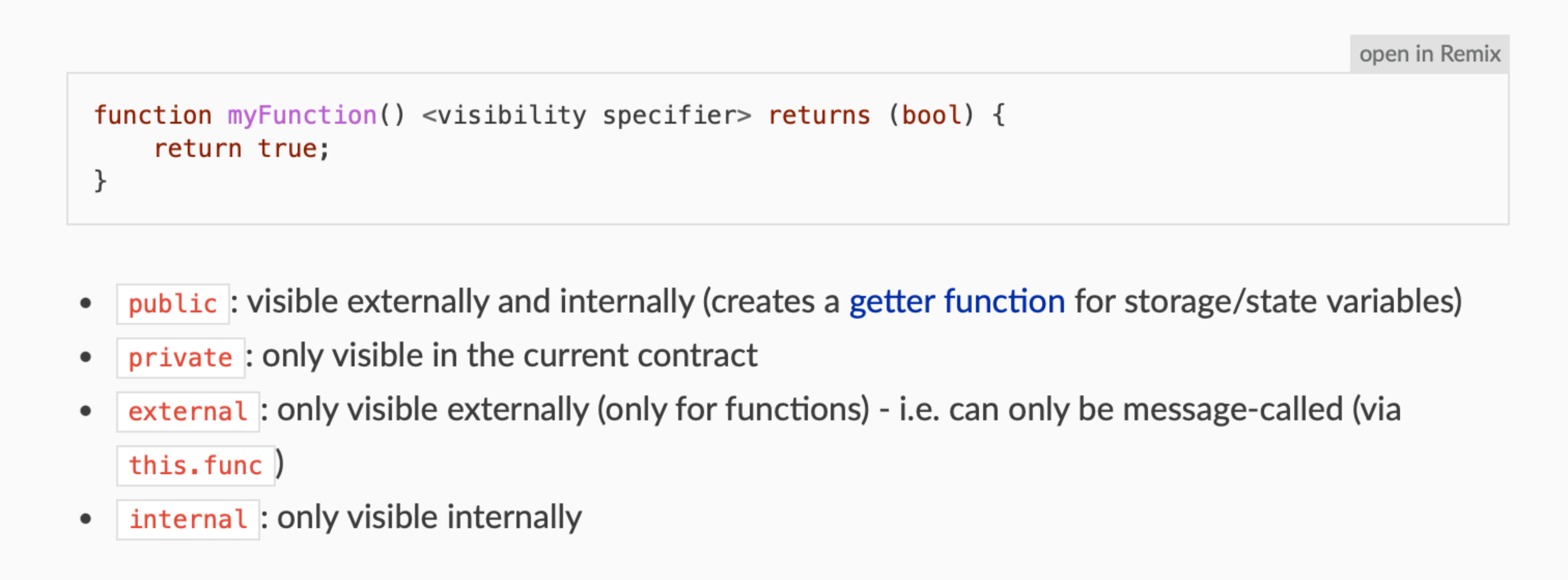


Mutability Modifier

Return Types

```
function get() public view returns (uint) {
    return storedData;
}
```

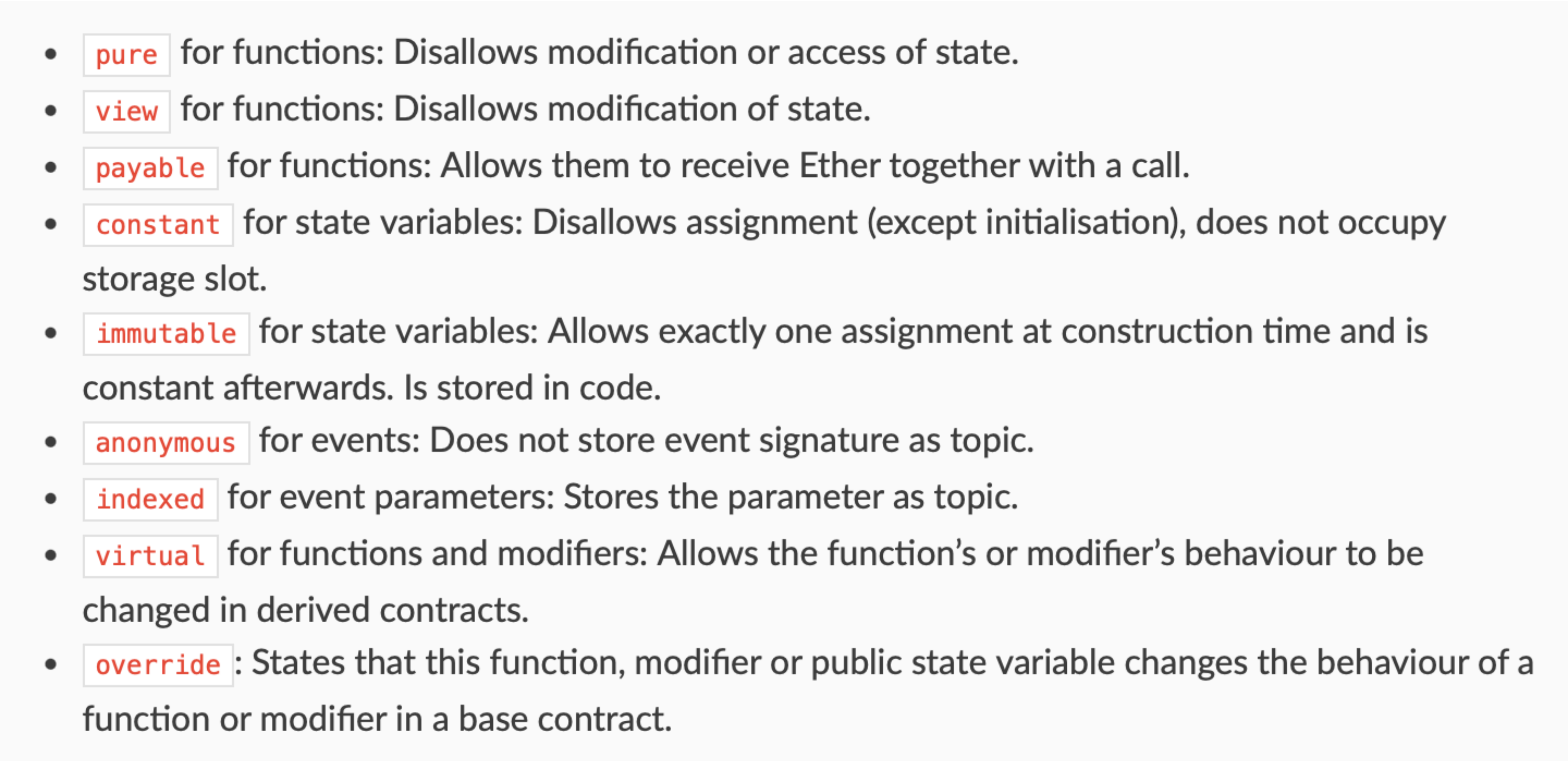
Visibility



```
function myFunction() <visibility specifier> returns (bool) {
    return true;
}
```

- `public` : visible externally and internally (creates a getter function for storage/state variables)
- `private` : only visible in the current contract
- `external` : only visible externally (only for functions) - i.e. can only be message-called (via `this.func`)
- `internal` : only visible internally

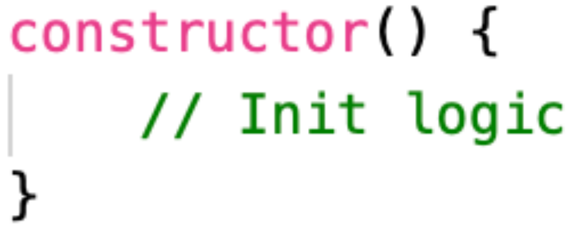
Modifiers



- `pure` for functions: Disallows modification or access of state.
- `view` for functions: Disallows modification of state.
- `payable` for functions: Allows them to receive Ether together with a call.
- `constant` for state variables: Disallows assignment (except initialisation), does not occupy storage slot.
- `immutable` for state variables: Allows exactly one assignment at construction time and is constant afterwards. Is stored in code.
- `anonymous` for events: Does not store event signature as topic.
- `indexed` for event parameters: Stores the parameter as topic.
- `virtual` for functions and modifiers: Allows the function’s or modifier’s behaviour to be changed in derived contracts.
- `override`: States that this function, modifier or public state variable changes the behaviour of a function or modifier in a base contract.

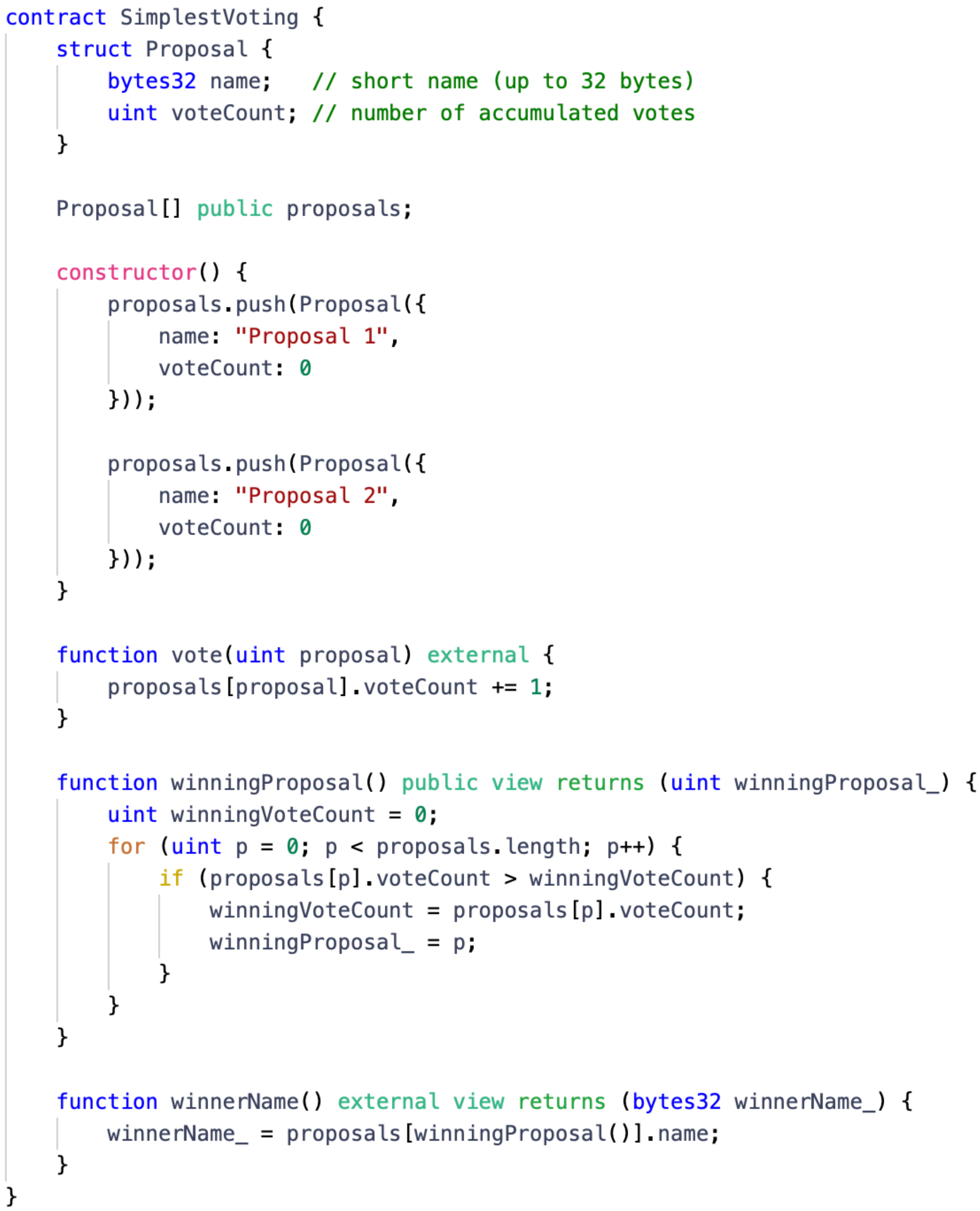
Constructors

- Optional function
- Invoked when initially creating the contract
- Used to customize settings or give an initial state



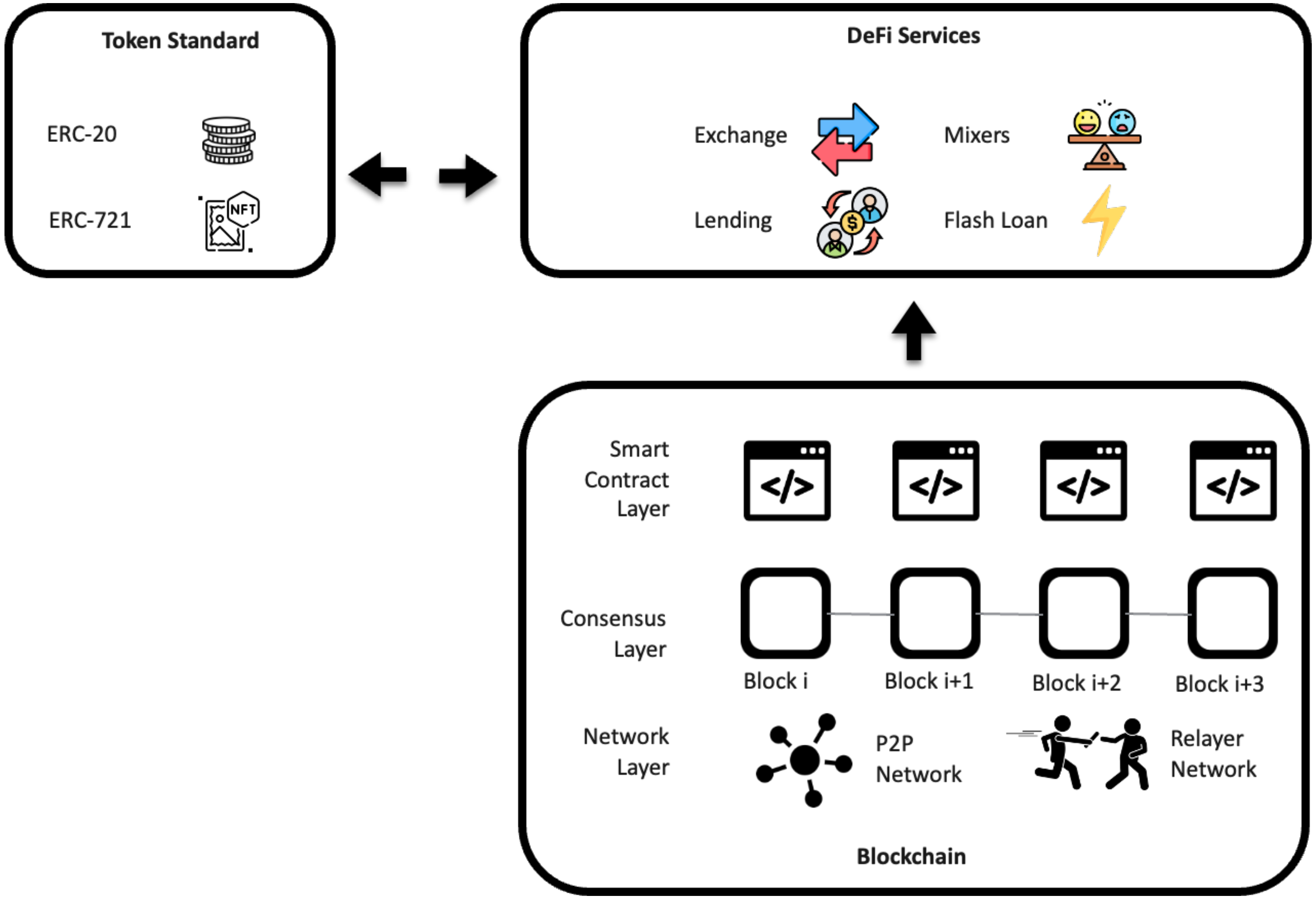
```
constructor() {
    // Init logic
}
```

Simple Voting (Remix Demo)



```
contract SimplestVoting {
    struct Proposal {
        bytes32 name;   // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    Proposal[] public proposals;

    constructor() {
        proposals.push(Proposal({
            name: "Proposal 1",
            voteCount: 0
        }));

        proposals.push(Proposal({
            name: "Proposal 2",
            voteCount: 0
        }));
    }

    function vote(uint proposal) external {
        proposals[proposal].voteCount += 1;
    }

    function winningProposal() public view returns (uint winningProposal_) {
        uint winningVoteCount = 0;
        for (uint p = 0; p < proposals.length; p++) {
            if (proposals[p].voteCount > winningVoteCount) {
                winningVoteCount = proposals[p].voteCount;
                winningProposal_ = p;
            }
        }
    }

    function winnerName() external view returns (bytes32 winnerName_) {
        winnerName_ = proposals[winningProposal()].name;
    }
}
```

DeFi?



Token Standard

ERC-20

ERC-721

DeFi Services

Exchange

Mixers

Lending

Flash Loan

Smart Contract Layer

Consensus Layer

Block i

Block i+1

Block i+2

Block i+3

Network Layer

P2P Network

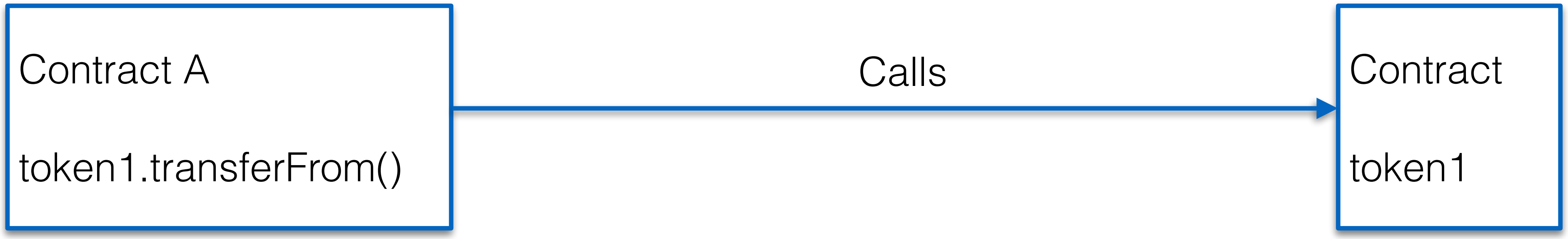
Relayer Network

Blockchain

ERC20 – Interfaces for basic token behaviour

- TotalSupply: The total number of tokens that will ever be issued
- BalanceOf: The account balance of a token owner's account
- Transfer: Automatically executes transfers of a specified number of tokens to a specified address for transactions using the token
- TransferFrom: Automatically executes transfers of a specified number of tokens from a specified address using the token
- Approve: Allows a spender to withdraw a set number of tokens from a specified account, up to a specific amount
- Allowance: Returns a set number of tokens from a spender to the owner

Call Contracts



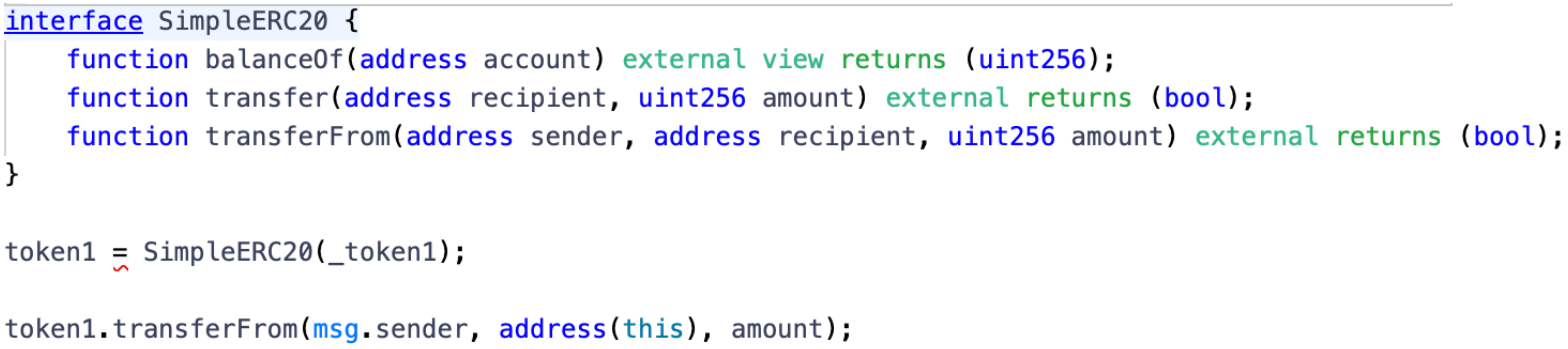
Contract A

token1.transferFrom()

Calls

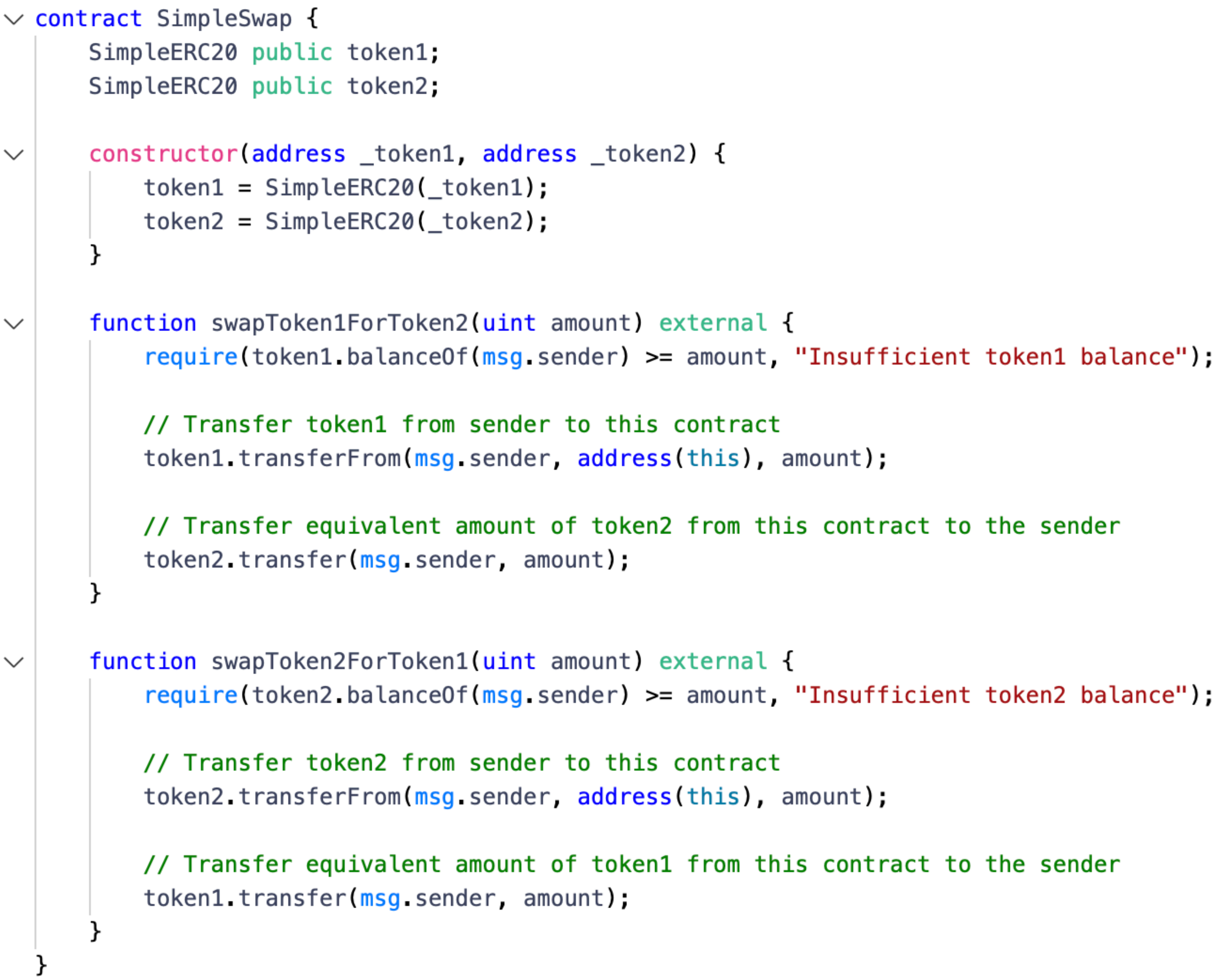
Contract

token1



```
interface SimpleERC20 {
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
}

token1 = SimpleERC20(_token1);

token1.transferFrom(msg.sender, address(this), amount);
```

Second Contract -- Exchange (Remix Demo)

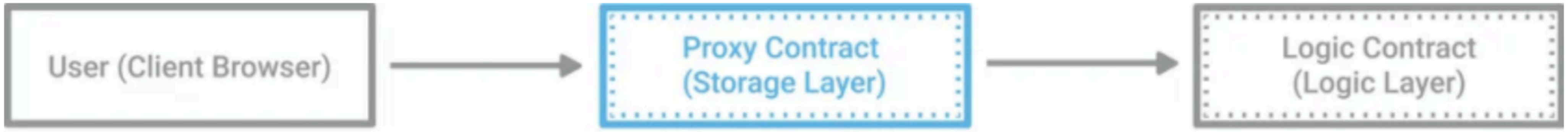


```
contract SimpleSwap {
    SimpleERC20 public token1;
    SimpleERC20 public token2;

    constructor(address _token1, address _token2) {
        token1 = SimpleERC20(_token1);
        token2 = SimpleERC20(_token2);
    }

    function swapToken1ForToken2(uint amount) external {
        require(token1.balanceOf(msg.sender) >= amount, "Insufficient token1 balance");

        // Transfer token1 from sender to this contract
        token1.transferFrom(msg.sender, address(this), amount);

        // Transfer equivalent amount of token2 from this contract to the sender
        token2.transfer(msg.sender, amount);
    }

    function swapToken2ForToken1(uint amount) external {
        require(token2.balanceOf(msg.sender) >= amount, "Insufficient token2 balance");

        // Transfer token2 from sender to this contract
        token2.transferFrom(msg.sender, address(this), amount);

        // Transfer equivalent amount of token1 from this contract to the sender
        token1.transfer(msg.sender, amount);
    }
}
```

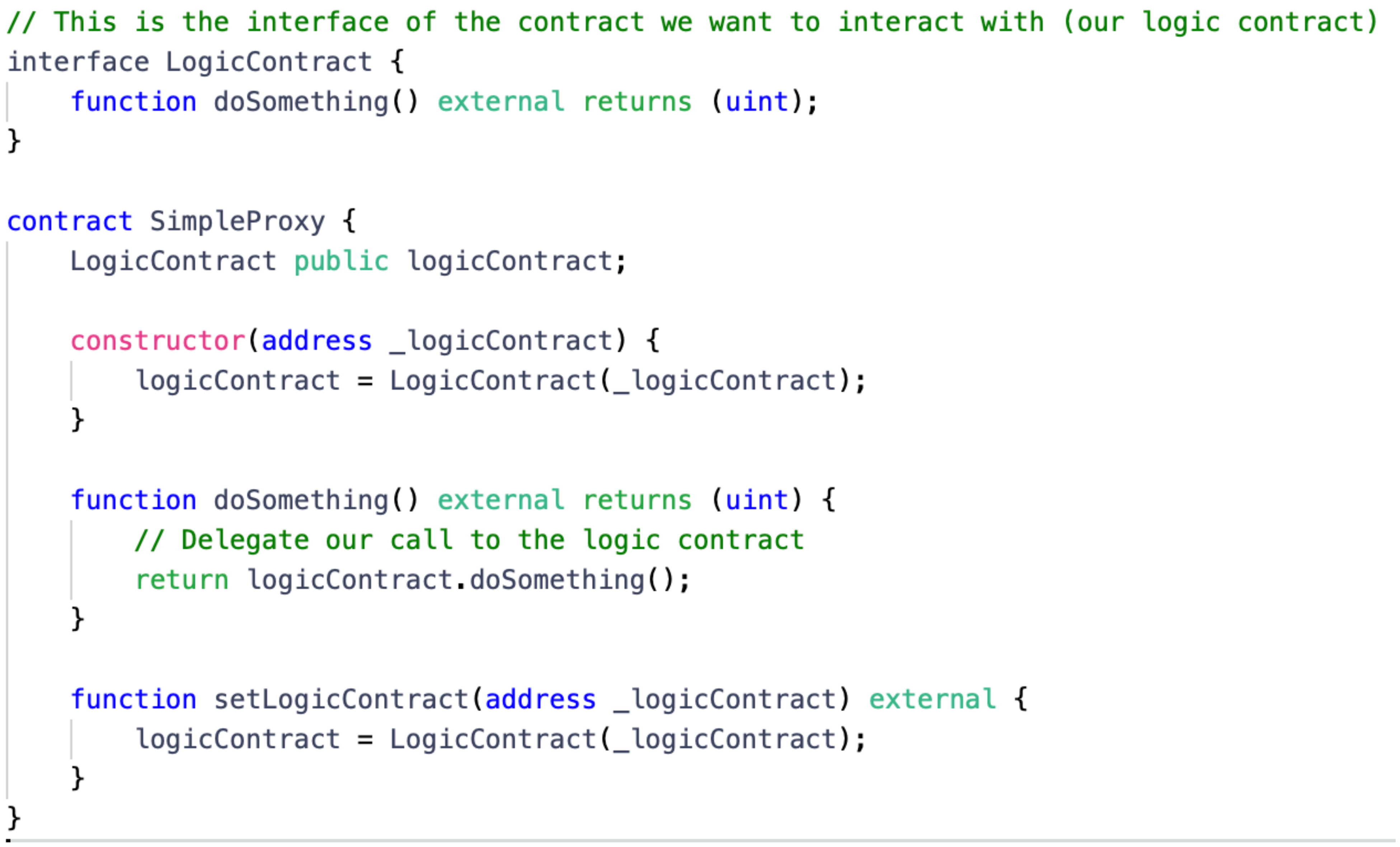
Smart Contract Limitations

- Immutable Once Deployed

Proxy



(Stores address of the logic contract)



```
// This is the interface of the contract we want to interact with (our logic contract)
interface LogicContract {
    function doSomething() external returns (uint);
}

contract SimpleProxy {
    LogicContract public logicContract;

    constructor(address _logicContract) {
        logicContract = LogicContract(_logicContract);
    }

    function doSomething() external returns (uint) {
        // Delegate our call to the logic contract
        return logicContract.doSomething();
    }

    function setLogicContract(address _logicContract) external {
        logicContract = LogicContract(_logicContract);
    }
}
```

Smart Contract Limitations

- Immutable Once Deployed
- Limited External Data Access

Oracle

- Token-management
  - E.g., ERC-20
- DEXes
  - E.g., Uniswap
- NFT games
  - E.g., CryptoKitties

No external data

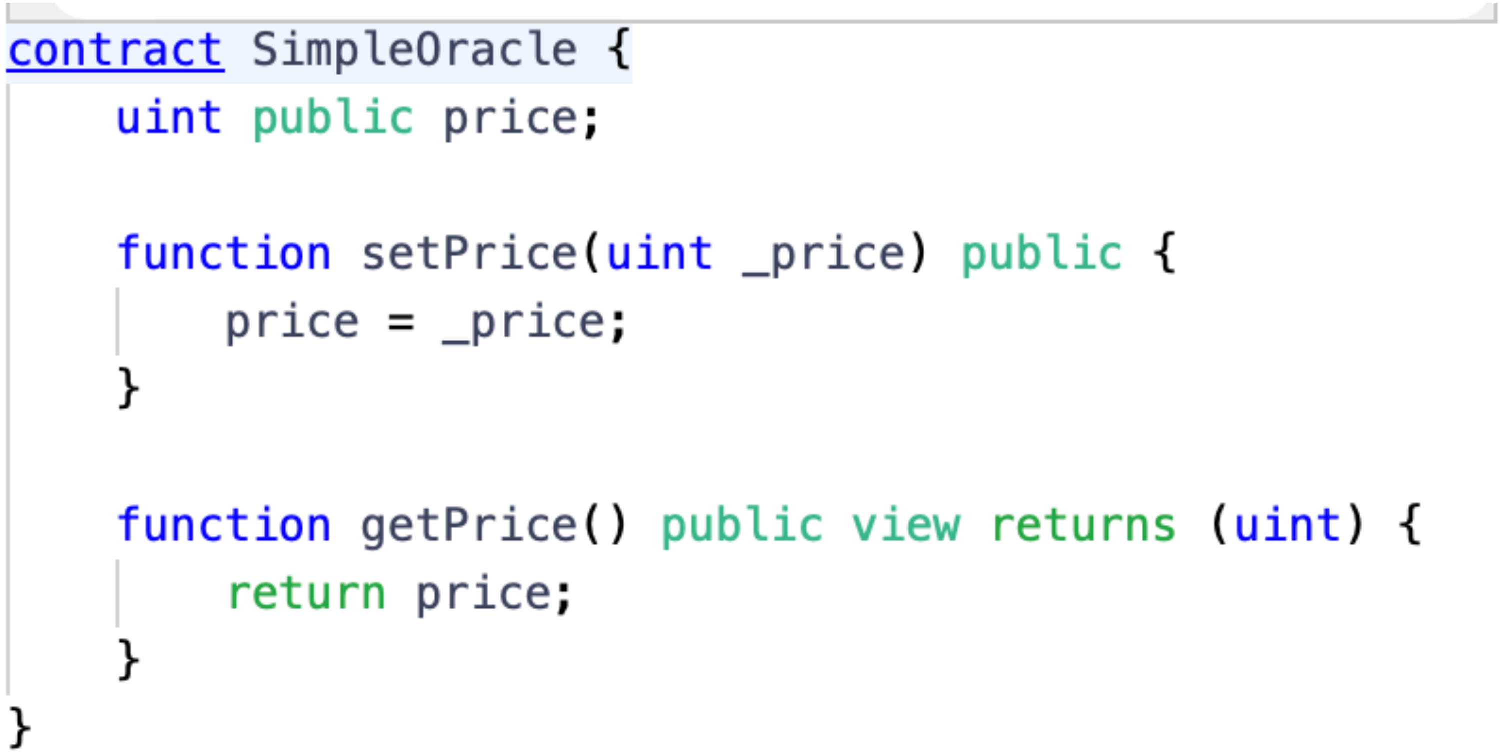
- Lending
  - E.g., MakerDAO, Compound, Aave
- Insurance
  - E.g., flight insurance

Needs external data

Oracle

- Blockchain lack internet connections!

Oracle



```
contract SimpleOracle {
    uint public price;

    function setPrice(uint _price) public {
        price = _price;
    }

    function getPrice() public view returns (uint) {
        return price;
    }
}
```

Smart Contract Limitations

- Immutable Once Deployed
- Limited External Data Access
- Limited Scalability
- Gas Costs
- Coding and Security Expertise Required

Some other issues

- Vulnerabilities
  - Reentrancy, Price oracle manipulation, etc.
  - Audit / Pre-audit
- Transparency (Bytecode, EVM)
- P2P Transaction propagation

Other Solidity quirks and features

- Inheritance
- Storage, memory, calldata
- ……

- https://docs.soliditylang.org/en/v0.8.18/
- https://solidity-by-example.org
- https://github.com/lzhou1110/UZH-SmartContract

- ChatGPT is your friend! :D