

# Wir entwickeln den JavaLand Coin



# Michael Heinrichs

@netopyr

- Java Champion
- Leader of JUG Freiburg
- Contractor for Swirls Labs
- Founder of Netopyr GmbH



# Michael Heinrichs

@netopyr

- I ❤️ coding
- I ❤️ my family
- I ❤️ cooking
- I ❤️ travelling



# Hendrik Ebbers

@hendrikEbbers

- Java Champion
- Eclipse Adoptium WG
- Contractor for Swirls Labs
- Founder of Open Elements



# Hendrik Ebbers

@hendrikEbbers

- I ❤️ Star Wars
- I ❤️ dogs
- I ❤️ boardgames
- I ❤️ open source



@netopyr | @hendrikEbbers

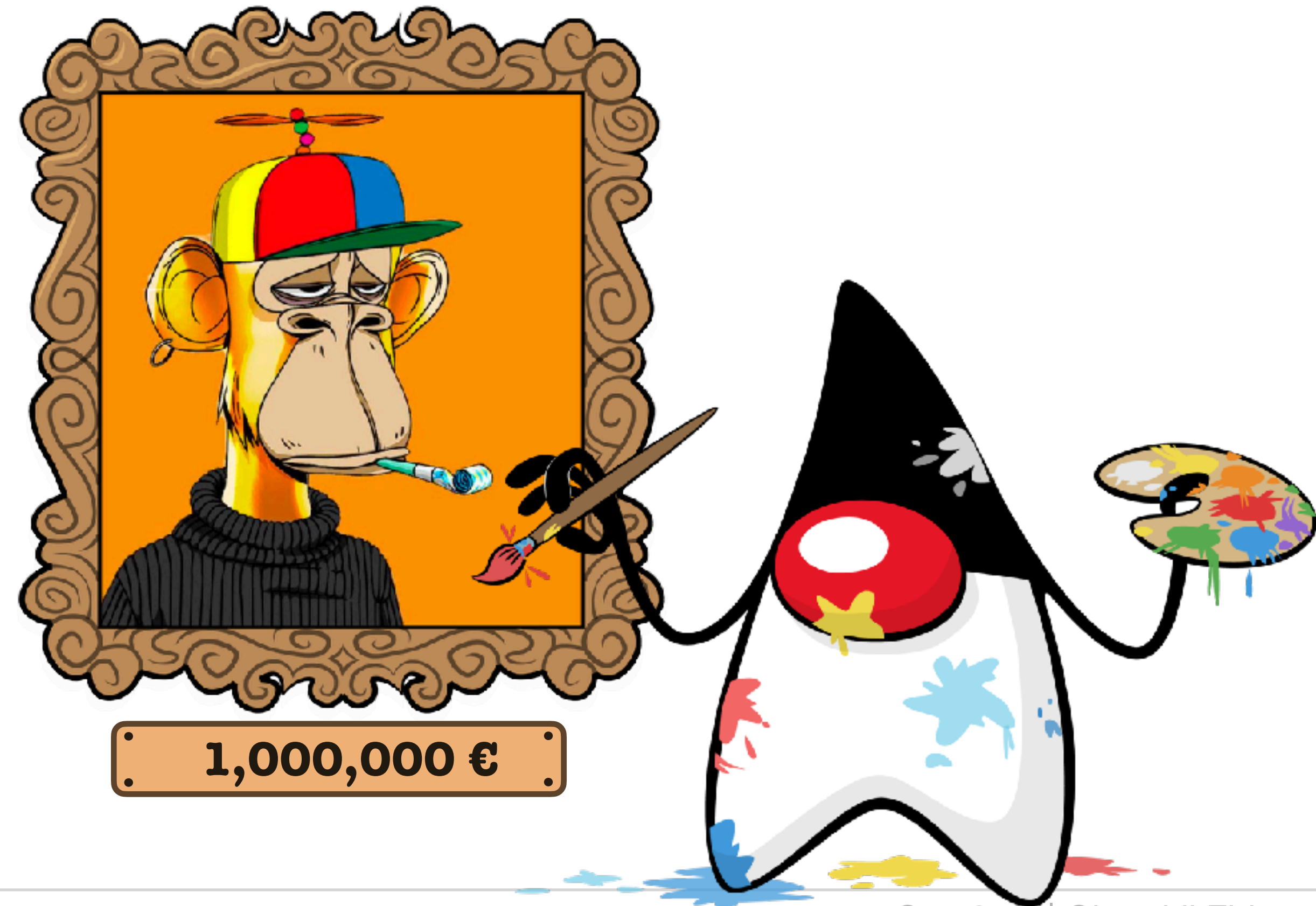
# What you will learn today

- What is a smart contract
- What is a token
- How to use public ledgers



# What you will NOT learn today

- How to trade Bitcoins
- How to get rich with NFTs



@net0pyr | @hendrikEbbers

Forget the coins!  
Let's concentrate on  
**Technologies**



# Let's concentrate on technologies



**1st generation**



**2nd generation**

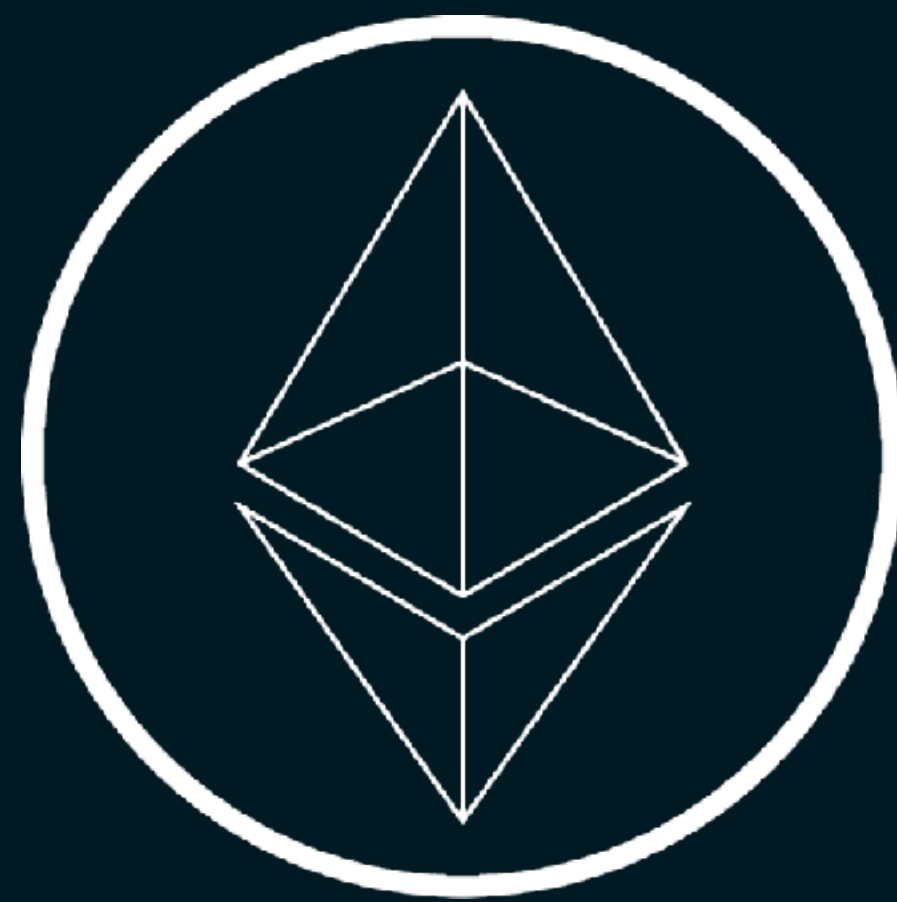
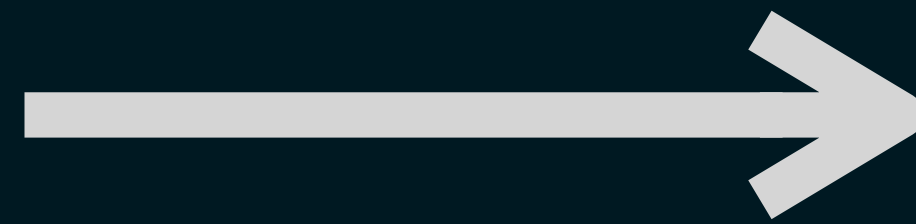


**3rd generation**

# Let's concentrate on technologies



**Decentralized  
Blockchain**



**Smart Contracts**



**Carbon negative,  
faster & cheaper**

# The Hedera Network

# The Hedera Network

- The Hedera Network is a network that is based on several nodes
- Nodes running on machines of the Hedera Foundation council members



# The Hedera Network

- The productive network is called MainNet
- Hedera provides TestNet and PreviewNet networks for development and testing



**MainNet**



**TestNet**



**PreviewNet**

# The Hedera Network

- A local network can be setup by using the open source local node project that is based on Docker  
<http://bit.ly/3JqeMvz>



**MainNet**



**TestNet**



**PreviewNet**



**LocalNode**

# The Hedera Network

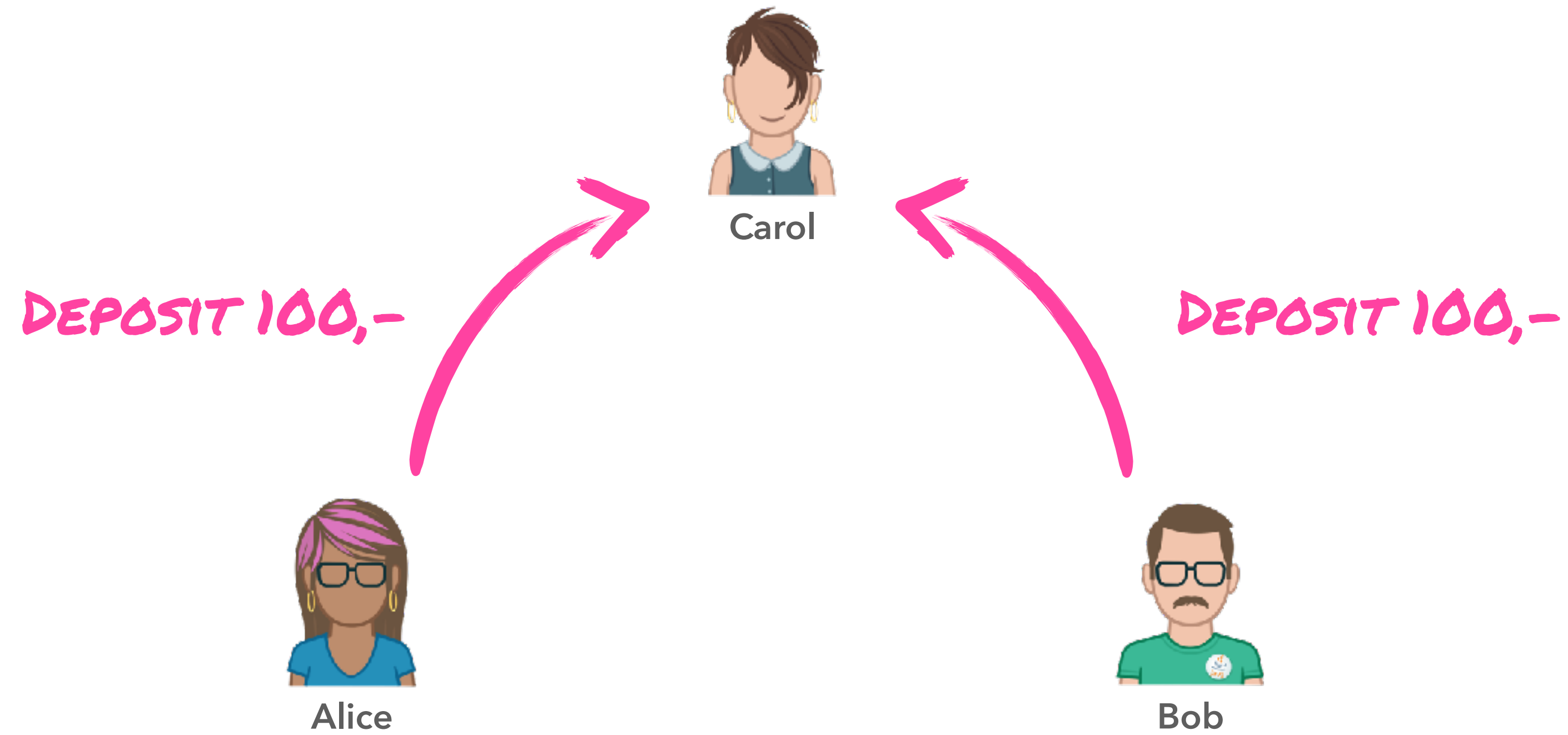
- MainNet can handle **> 1.000 tps** (transaction per second)
- Over **5.000.000.000** transactions have been handled in production
- In near future it will be **> 10 Billion** transactions



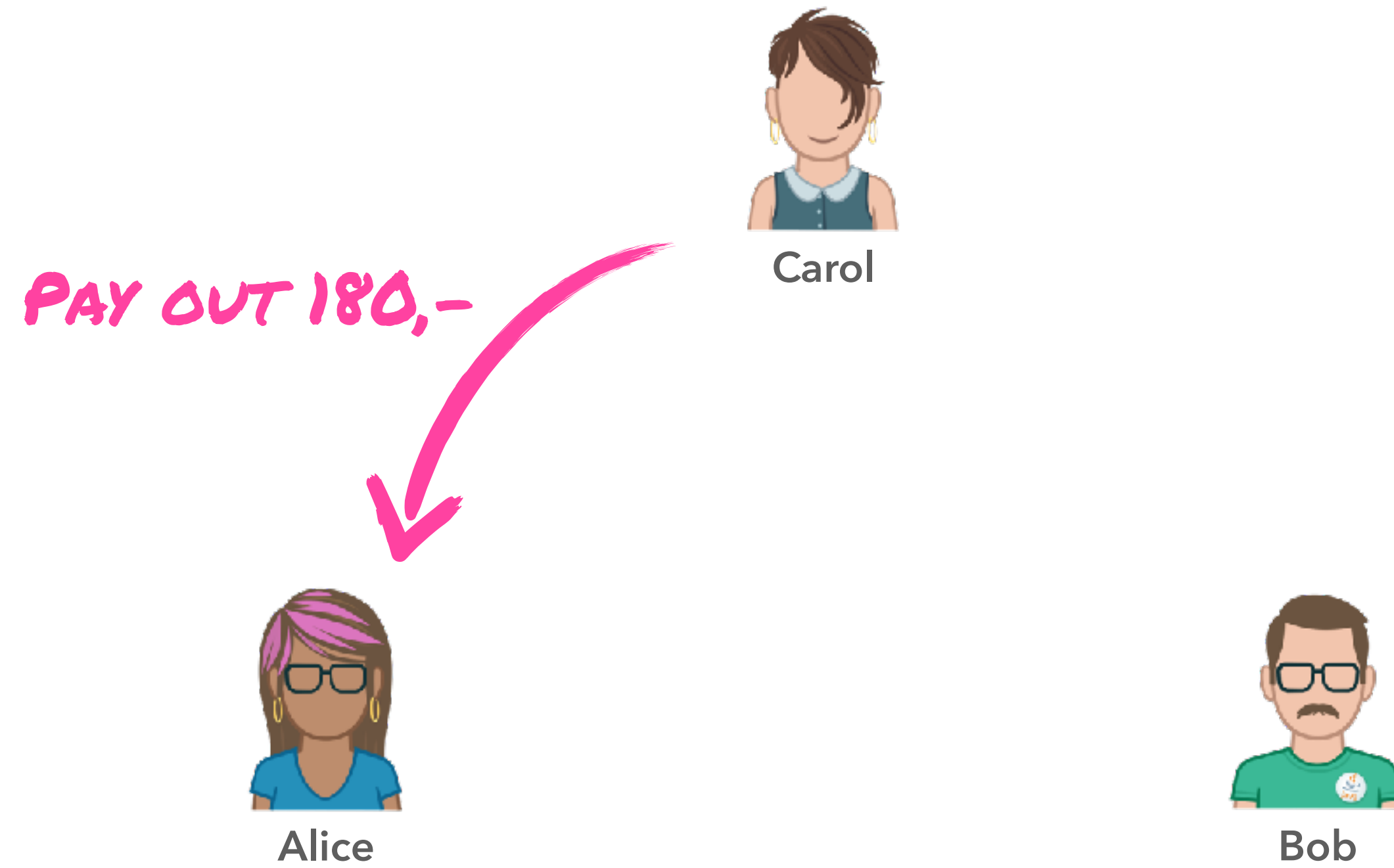
# Smart Contracts



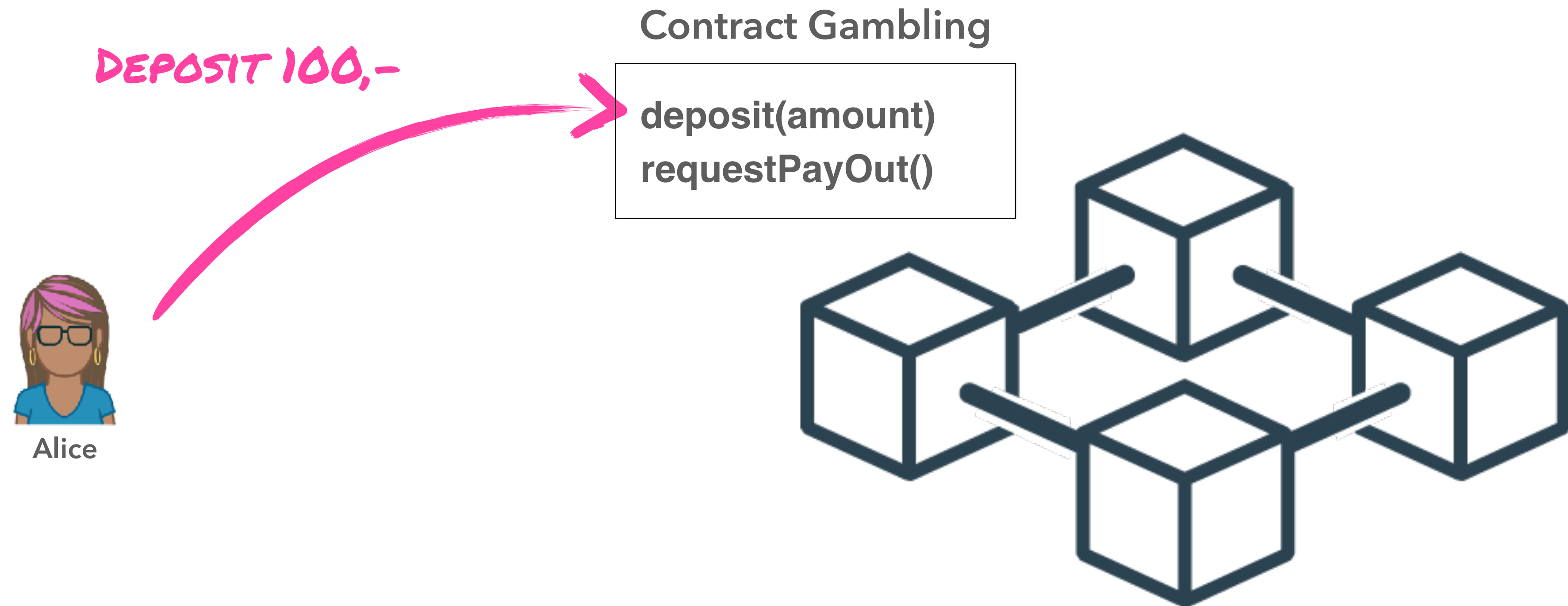
# Gambling Example



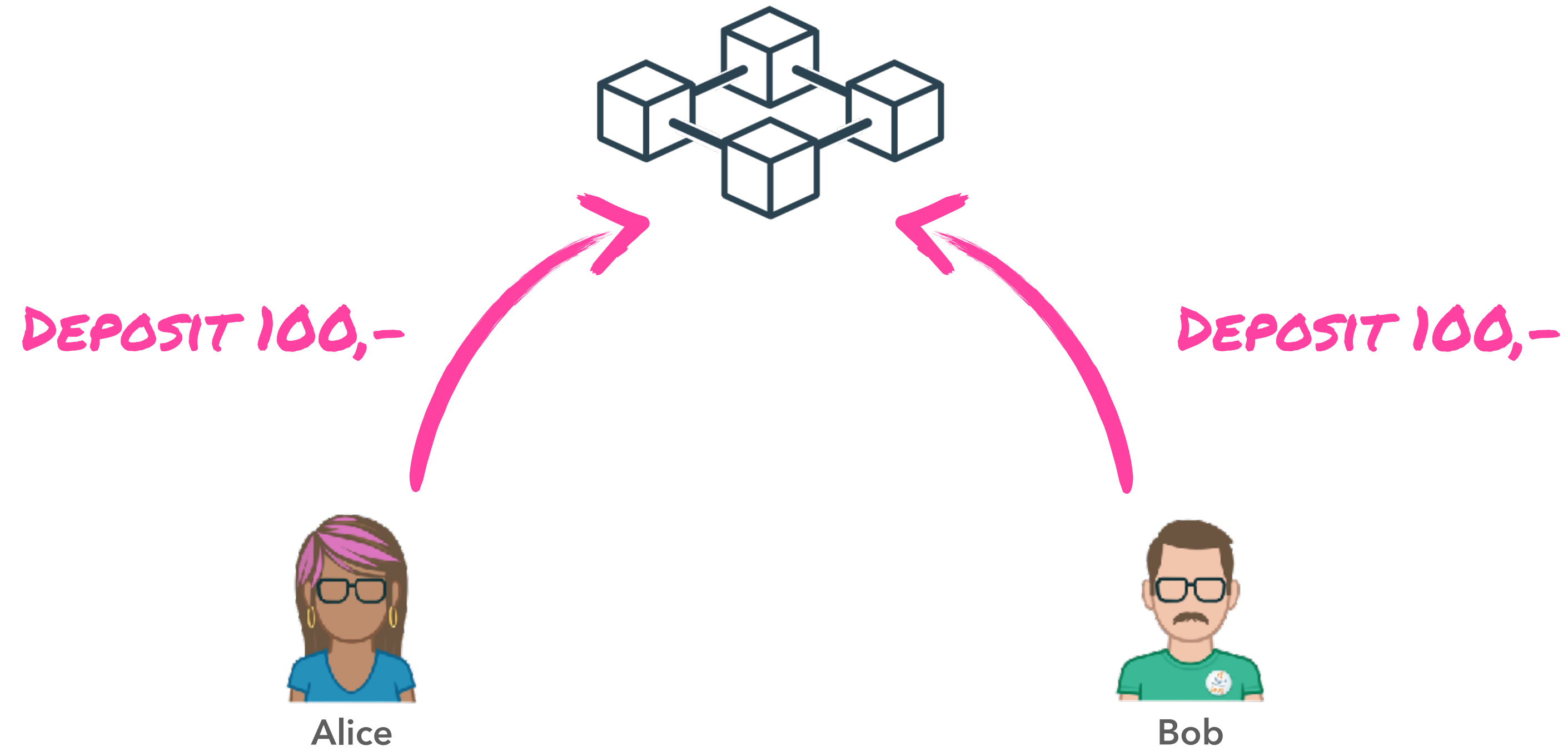
# Gambling Example



# Gambling Example



# Gambling Example



# Gambling Example



*PAY OUT 199,99*



Alice



Bob

# Use Cases

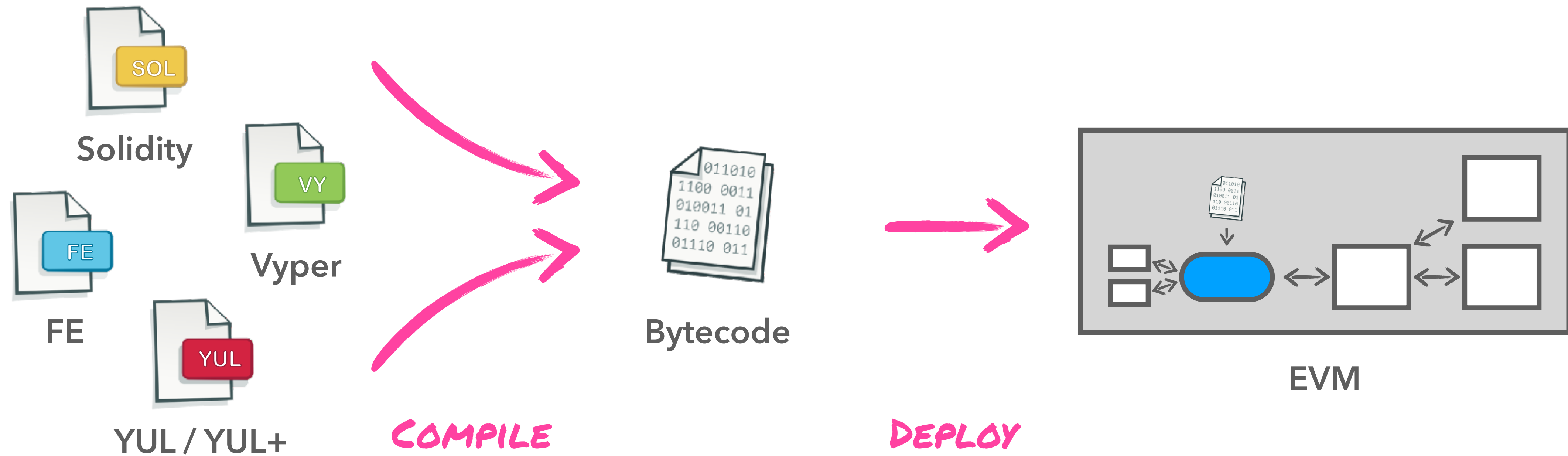
- Decentralized Finance (DeFi)
- Peer-to-peer markets
- Decentralized Autonomous Organization (DAO)
- ...

Ethereum

Virtual

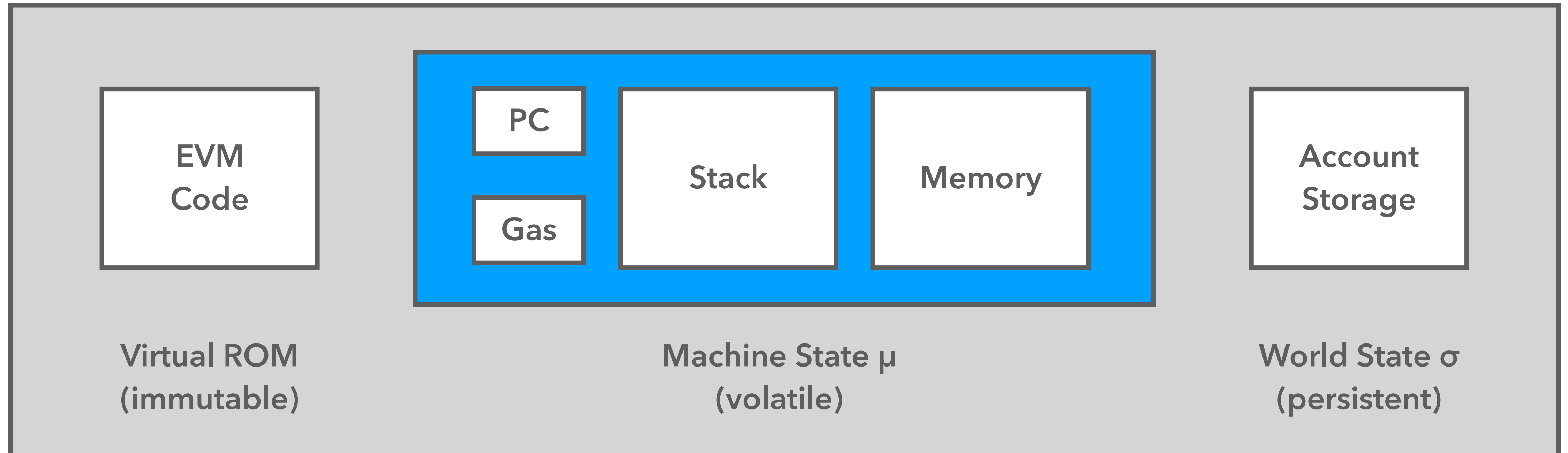
Machine

# EVM Code

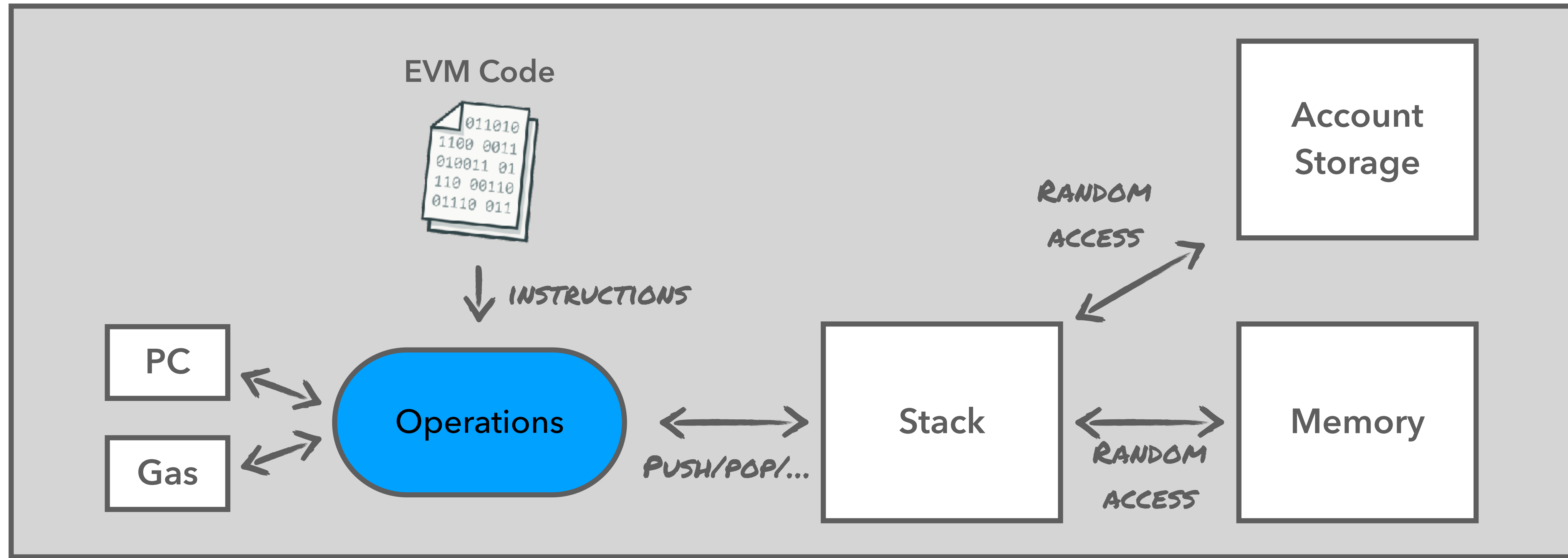




# EVM States



# Execution Model



# OpCodes

Stack	Name	Gas	Initial Stack	Resulting Stack	Notes
<b>00</b>	STOP	0			halt execution
<b>01</b>	ADD	3	a, b	a + b	(u)int256 addition modulo 2**256
<b>02</b>	MUL	5	a, b	a * b	(u)int256 multiplication modulo 2**256
<b>03</b>	SUB	3	a, b	a - b	(u)int256 addition modulo 2**256
<b>04</b>	DIV	5	a, b	a // b	uint256 division
<b>05</b>	SDIV	5	a, b	a // b	int256 division
<b>06</b>	MOD	5	a, b	a % b	uint256 modulus
<b>07</b>	SMOD	5	a, b	a % b	int256 modulus

# EVM

# Languges

# Solidity

```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

# Building Blocks

VERSION PRAGMA



```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

# Building Blocks

CONTRACT



```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

# Building Blocks

STATE VARIABLES



```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```



# Building Blocks

```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

**EVENTS**



# Building Blocks

```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

CONSTRUCTOR



# Building Blocks

```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

**FUNCTIONS**



# Solidity

```
pragma solidity >= 0.7.0;

contract Coin {

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

    event Sent(address from, address to, uint amount);

    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender]);
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

# Value Types

- Boolean
- Integers  
(int, int8, int16, ..., int256, uint, uint8, uint16, ..., uint256)
- Fixed Point Numbers (👷)
- Address
- Byte Arrays (fixed and dynamically-sized)
- Enums

# Reference Types

- Array
- Map
- Struct

# Control Structures

- if, else
- while, do
- for
- break, continue
- return

# Error Handling

- state-reverting
- try-catch
- require
- revert



# Compiling Smart Contracts

# Hello Smart Contract

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

contract HelloWorld {

    function greet() public pure returns (string memory) {
        return "Hello, world!";
    }
}
```

# Compile Solidity

- To execute the smart contract we need to compile it
- The compilation is normally stored in a binary BIN file



# Compile Solidity

- The Solidity compiler `solc` can easily be installed locally

<https://docs.soliditylang.org/>

- The compiler provides different ways how it can be installed locally (brew, npm, ...)

# Compile Solidity

- We can easily compile our smart contract by using solc from the commandline:

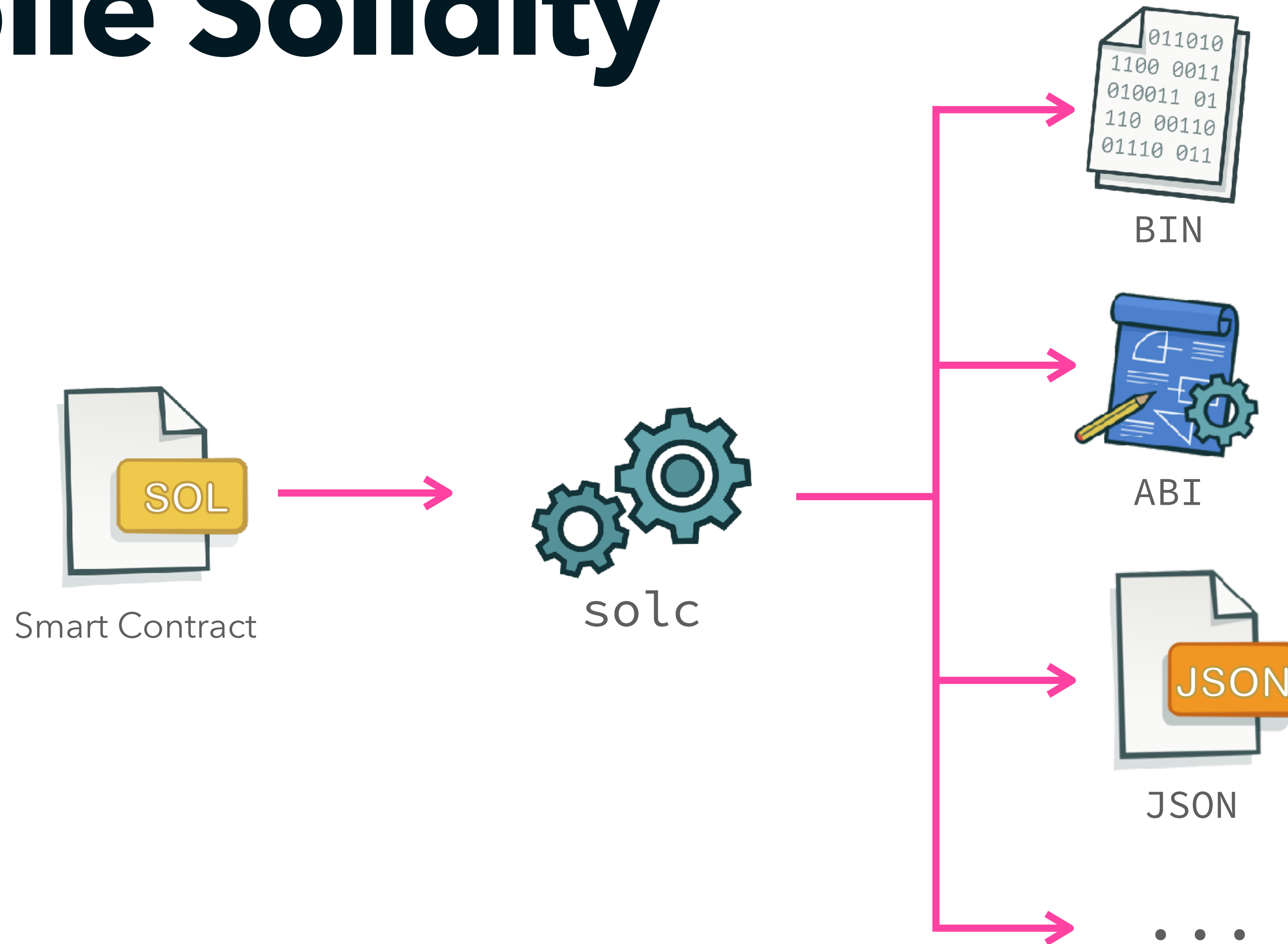
```
solc --bin -o build/contracts contracts/hello_world.sol
```

*WE WANT TO  
CREATE THE BIN FILE*

*OUTPUT FOLDER*

*INPUT*

# Compile Solidity



# Compile Solidity

- Instead of installing the compiler locally you can use it wrapped in a docker container

```
docker run -v $(pwd)/contracts:/contracts ethereum/solc:stable -o /contracts/output --abi --bin
```



# Compile Solidity with Maven

- As a Java developer I want to integrate the compilation in my build

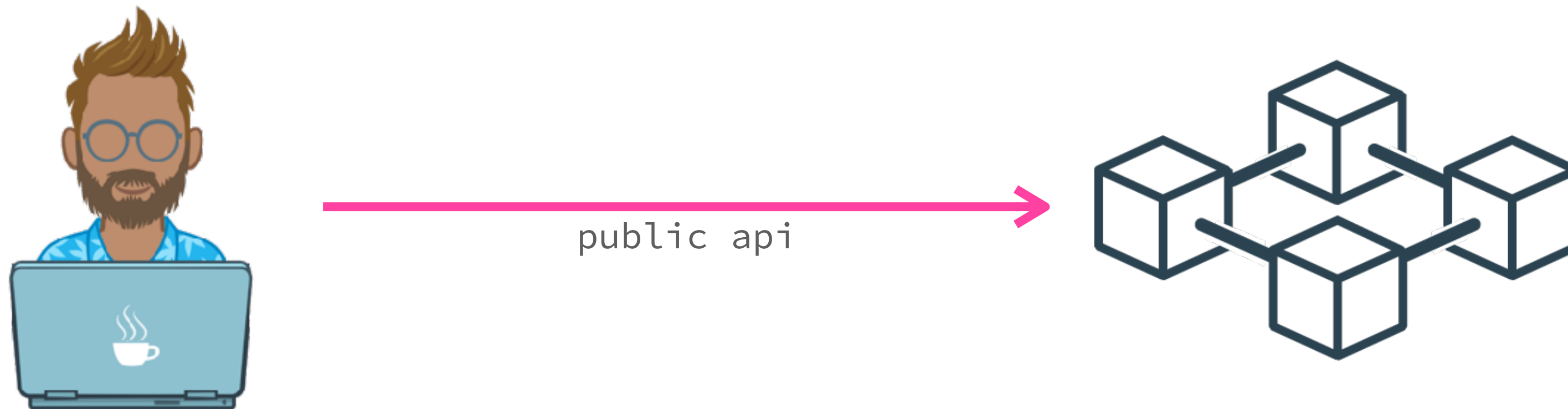




# Hedera For Java Devs

# Deploying a smart contract

- To execute a smart contract we need to deploy it on a ledger
- Public ledgers like Ethereum or Hedera provide public APIs to interact with the ledger



# HAPI - Hedera API

- Rich documentation available online

<https://docs.hedera.com/guides/docs/hedera-api>

- API libraries available for several languages



Swift



← *IN DEVELOPMENT*

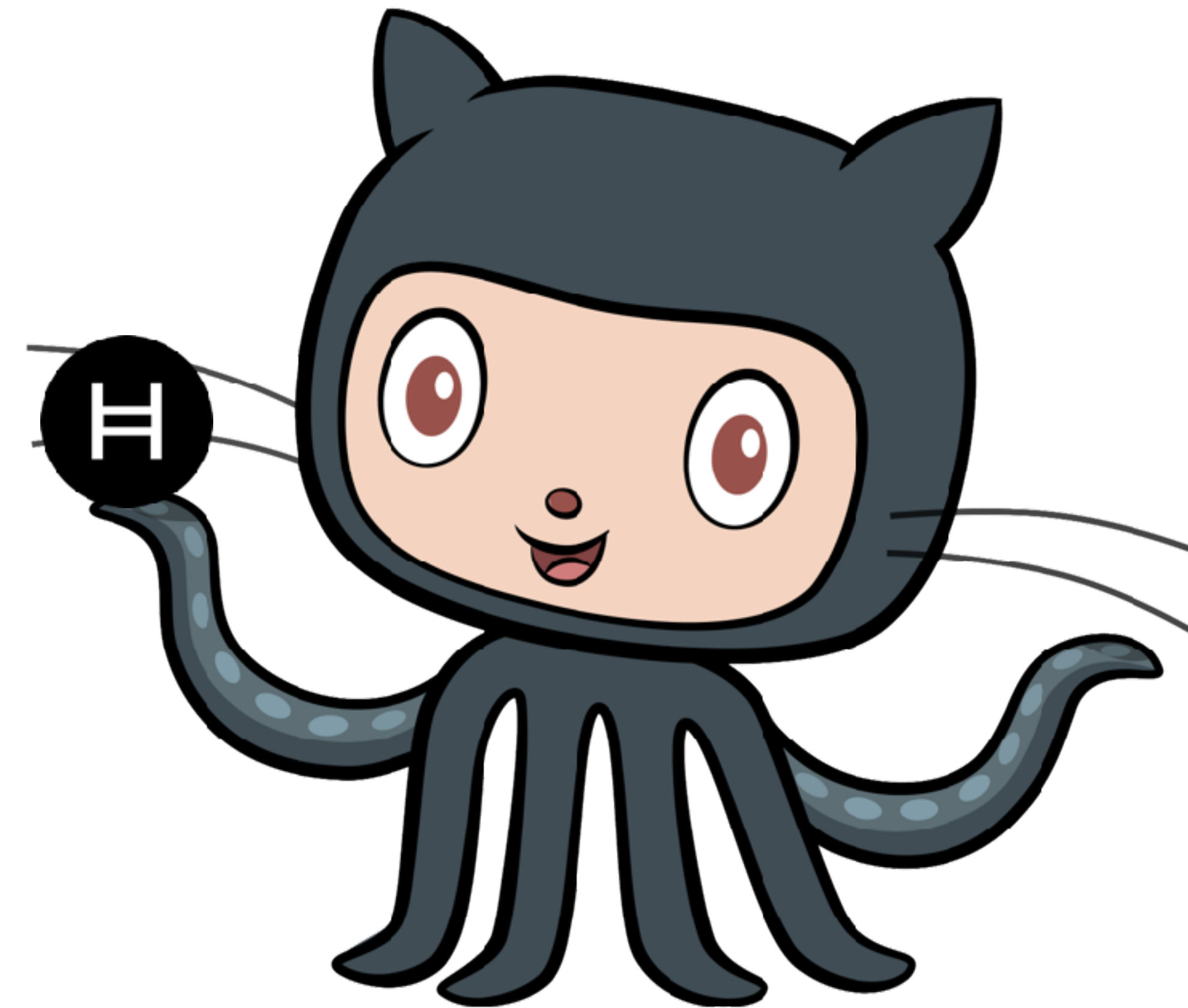
# HAPI - Hedera API

- We will concentrate on Java

```
<dependency>  
  <groupId>com.hedera.hashgraph</groupId>  
  <artifactId>sdk</artifactId>  
  <version>2.17.0</version>  
</dependency>
```

- All Hedera sources can be found at GitHub

<https://github.com/hashgraph/hedera-sdk-java>



@net0pyr | @hendrikEbbbers

# Hedera Testnet

- We do not want to execute our contracts on the real Hedera ledger at development time
- Hedera provides a test instance - Hedera Testnet

<https://docs.hedera.com/guides/testnet/testnet-access>



public api




Hedera Testnet

10,000 HBAR  
PER DAY



Browser tabs: Hedera Portal x +  
Address bar: portal.hedera.com/register Aktualisieren

### Hedera



## Create a **Mainnet** account

Choose an HBAR supported wallet to instantly create a mainnet account.

#### BLADE

Blade is more than just a wallet. It's a portal to Web3.


DOWNLOAD

#### HashPack

HashPack is the gateway for Hedera dApps, DeFi and NFTs.

DOWNLOAD

MORE WALLETS



## Create a **Testnet** account

Create a developer portal profile to start building decentralized applications.

Email

By creating a Hedera profile you agree to the [Terms Of Service](#) and [Privacy Policy](#).

START BUILDING

Already have an account? [Sign In](#)

Browser window: Hedera Portal | portal.hedera.com/register

Navigation: Hedera | Testnet

---

ACCOUNT **ECDSA** 🕒 Refill in 20:55

Private Key

Public Key

Account ID

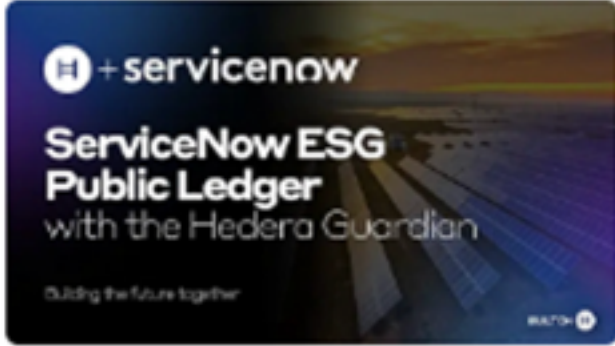
---

ACCOUNT **ED25519** 🕒 Refill in 16:45

Private Key

Public Key

UPDATES

 ServiceNow ESG Public Ledger with the Hedera...  
9/29/2022

MANAGED MIRROR NODE APIS

For a full list of Hedera official and community supported mirror node APIs, used for application development on the testnet and mainnet, please visit:  
<https://hedera.com/explorers>.

@netupyr | @hendrikEbbers

# HAPI - Smart contracts

- For executing a smart contract we need do the following steps:
  1. Connect to the ledger
  2. Upload the compiled contract to the ledger
  3. Create a smart contract out of the binary
  4. Call a function of the smart contract



# HAPI - Deploy a smart contract

```
final ContractCreateFlow flow = new ContractCreateFlow()  
    .setBytecode(bytecodeInHex)  
    .setGas(1_000_000);
```



```
final TransactionResponse transactionResponse = flow.execute(client);  
final TransactionReceipt receipt = transactionResponse.getReceipt(client);  
return receipt.contractId;
```

# HAPI - Gas definition

- When creating a transaction a gas value needs to be defined
- The value defines the maximum of gas that the transaction can cost
- Transaction will be aborted if the cost is too high

**transaction failed pre-check with  
the status `INSUFFICIENT_GAS`**



@net0pyr | @hendrikEbbbers

# HAPI - call contract function

```
final ContractCallQuery contractQuery = new ContractCallQuery()  
    .setGas(100000)  
    .setContractId(contractId)  
    .setFunction("greet");  
  
ContractFunctionResult result = contractQuery.execute(client);
```

# web3j

- Next to the Hedera SDK the web3j lib can be used
- web3j has been created for Ethereum
- Based on standards or widely adapted functionality it can be used for Hedera

# web3j

- web3j provides API to interact with smart contracts
- web3j provides functionality to create Java wrappers for smart contracts
- web3j provides wrapper for smart contract compilation
- web3j provides Maven plugin

# (Non-) Fungible Tokens

# Fungible

- Interchangable
- Divisible
- Uniform



Avij (talk · contribs), Public domain, via Wikimedia Commons

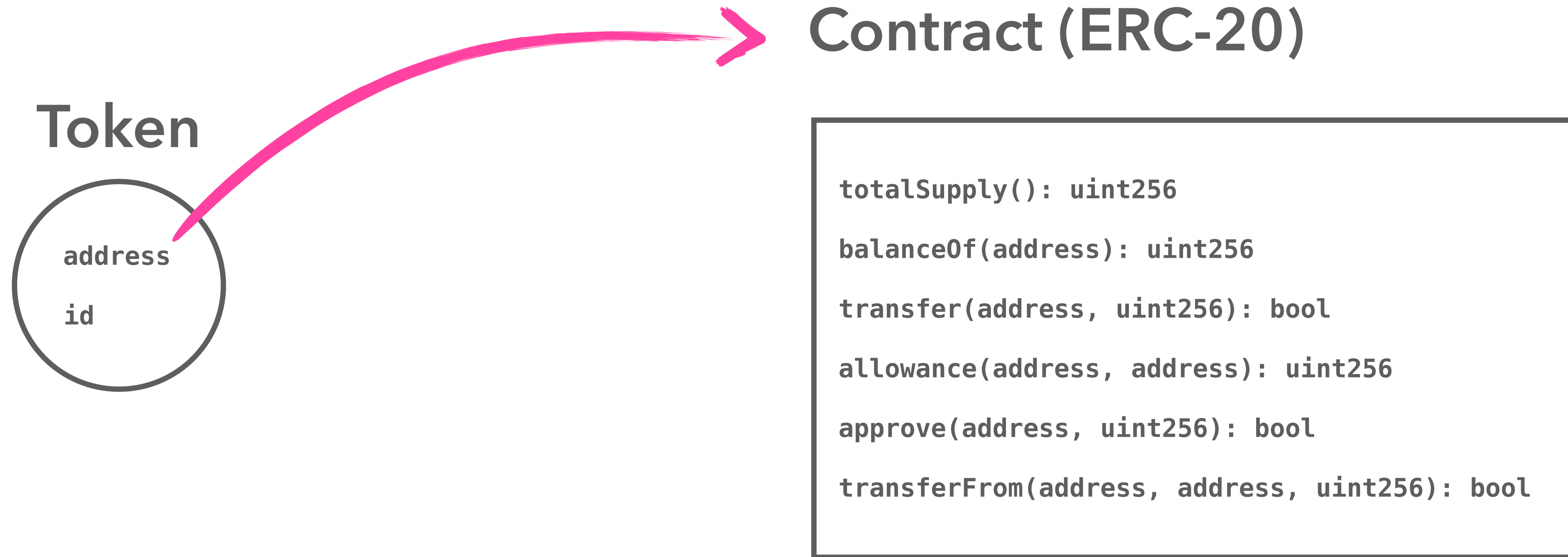
# Non-Fungible

- Non-Interchangable
- Non-Divisible
- Unique



"Pokemon Cards" by Steven Groves is licensed under CC BY 2.0

# What is a fungible token?





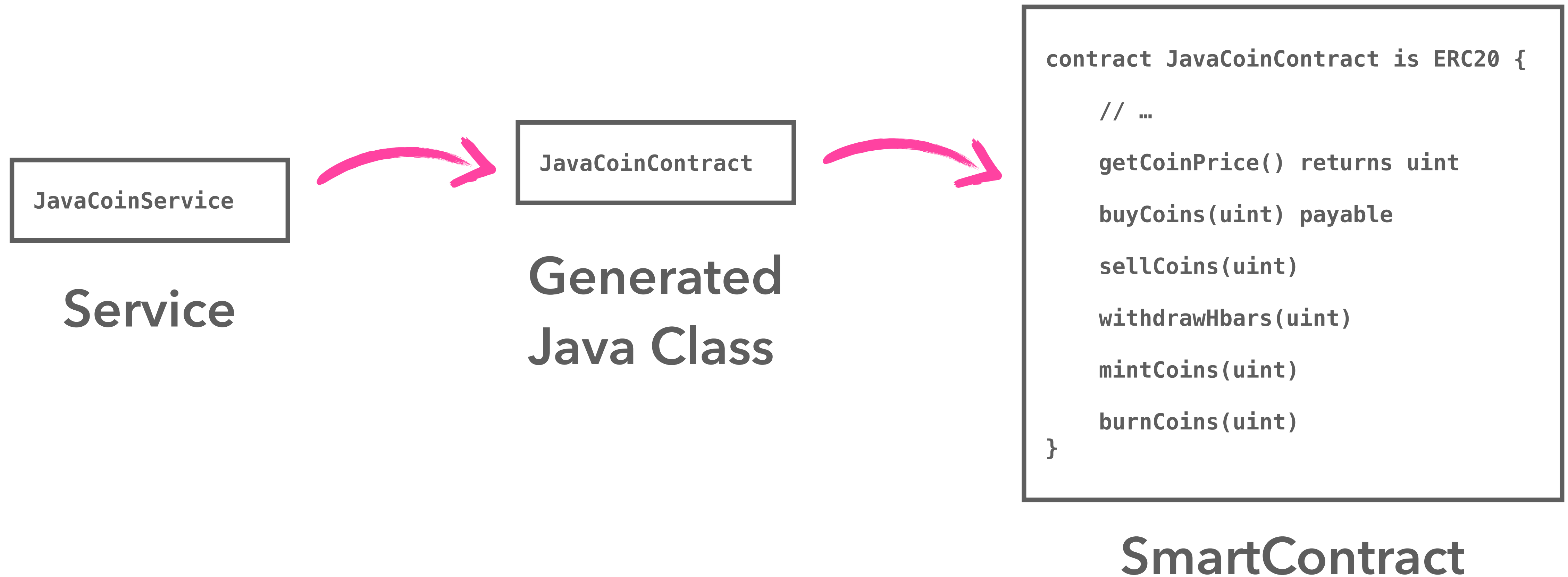
# What is a non-fungible token?



# JavaLand Coin

# App

# Overview



# JavaCoinContract

```
import "./contracts/token/ERC20/ERC20.sol";

contract JavaCoinContract is ERC20 {

    address payable _owner;
    address payable _treasury;
    uint256 _coinScale;
    uint256 constant _hBarScale = 10 ** 8;

    uint256 _price;

    constructor(uint256 initialSupply) ERC20("JavaLand Coin", "JC") {
        _owner = payable(msg.sender);
        _treasury = payable(address(this));
        _coinScale = 10 ** decimals();
        _mint(address(this), initialSupply);
        _recalculatePrice(0);
    }

    // ...
}
```

# JavaCoinContract cont'd

```
import "../contracts/token/ERC20/ERC20.sol";

contract JavaCoinContract is ERC20 {

    // ...

    function _recalculatePrice(int delta) private { /* ... */ }

    function getCoinPrice() public view returns (uint) {
        return _price;
    }

    function getCoinsForAccount() public view returns (uint) {
        return balanceOf(msg.sender);
    }

    // ...
}
```

# Buy Coins

```
function buyCoins(uint count) public payable {
    // check input parameters (count > 0 and count < 10% of total supply)
    // TODO

    uint total = count * _price / _coinScale;

    // check that user has enough HBars
    // TODO

    // check that pool contains enough JavaLandCoins
    // TODO

    // recalculate price
    _recalculatePrice();

    // transfer HBars to contract
    // TODO

    // transfer JavaLandCoins to user
    // TODO
}
```

# Commands

```
// Check conditions
require(bool condition, string message);

// Address of the sender
msg.address

// Get the Hbar balance of an account
address.balance

// Transfer Hbars
(bool success, ) = address.call{value: amount}("");
require(success, "Transfer failed.");

// Get the total amount of JavaLandCoins
totalSupply();

// Get balance of JavaLandCoins
balanceOf(address account);

// Transfer JavaLandCoins
_transfer(address from, address to, uint amount)
```

# Buy Coins - Solution

```
function buyCoins(uint count) public payable {
    // check input parameters (count > 0 and count < 10% of total supply)
    require(count > 0, "Count must be greater than 0");
    require(count <= totalSupply() / 10, "Not possible to buy more than 10% of total supply");

    uint total = count * _price / _coinScale;

    // check that user has enough HBars
    require(msg.sender.balance >= total, "Not enough HBars");

    // check that pool contains enough JavaLandCoins
    require(balanceOf(_treasury) >= count, "Not enough JavaLandCoins in pool");

    // recalculate price
    _recalculatePrice(-int(count));

    // transfer HBars to contract
    (bool success, ) = _treasury.call{value: total}("");
    require(success, "Transfer failed.");

    // transfer JavaLandCoins to user
    _transfer(_treasury, msg.sender, count);
}
```





# Michael Heinrichs

@net0pyr



# Hendrik Ebbers

@hendrikEbers

