Practical Blockchain

Module 6: The Ethereum Project

Anthony Kigombola

Aaron Kondoro

Scope

- Overview
- Cryptography
- Architecture
- Accounts
- Wallets
- EVM
- Gas
- Smart Contracts
- Activity 1: Deploying smart contracts
- Activity 2: Developing smart contracts
- Activity 3: Testing smart contracts before deployment

Overview

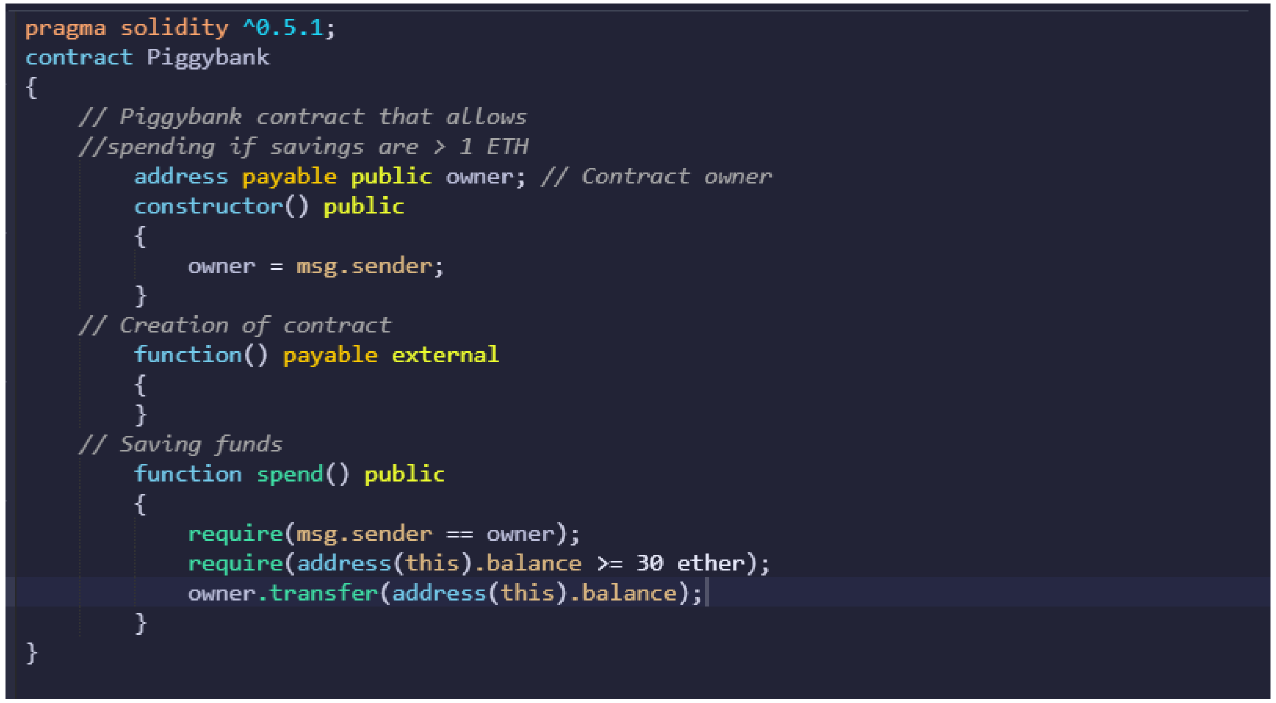
Ethereum is an implementation of a distributed ledger based on **Blockchain** technology.

Apart from storing digital transactions, Ethereum platform allows users to create and deploy **decentralized applications**.

Overview

Official Definition: Ethereum is an open blockchain platform that lets anyone build and use decentralized applications (dApps) that run on blockchain technology.

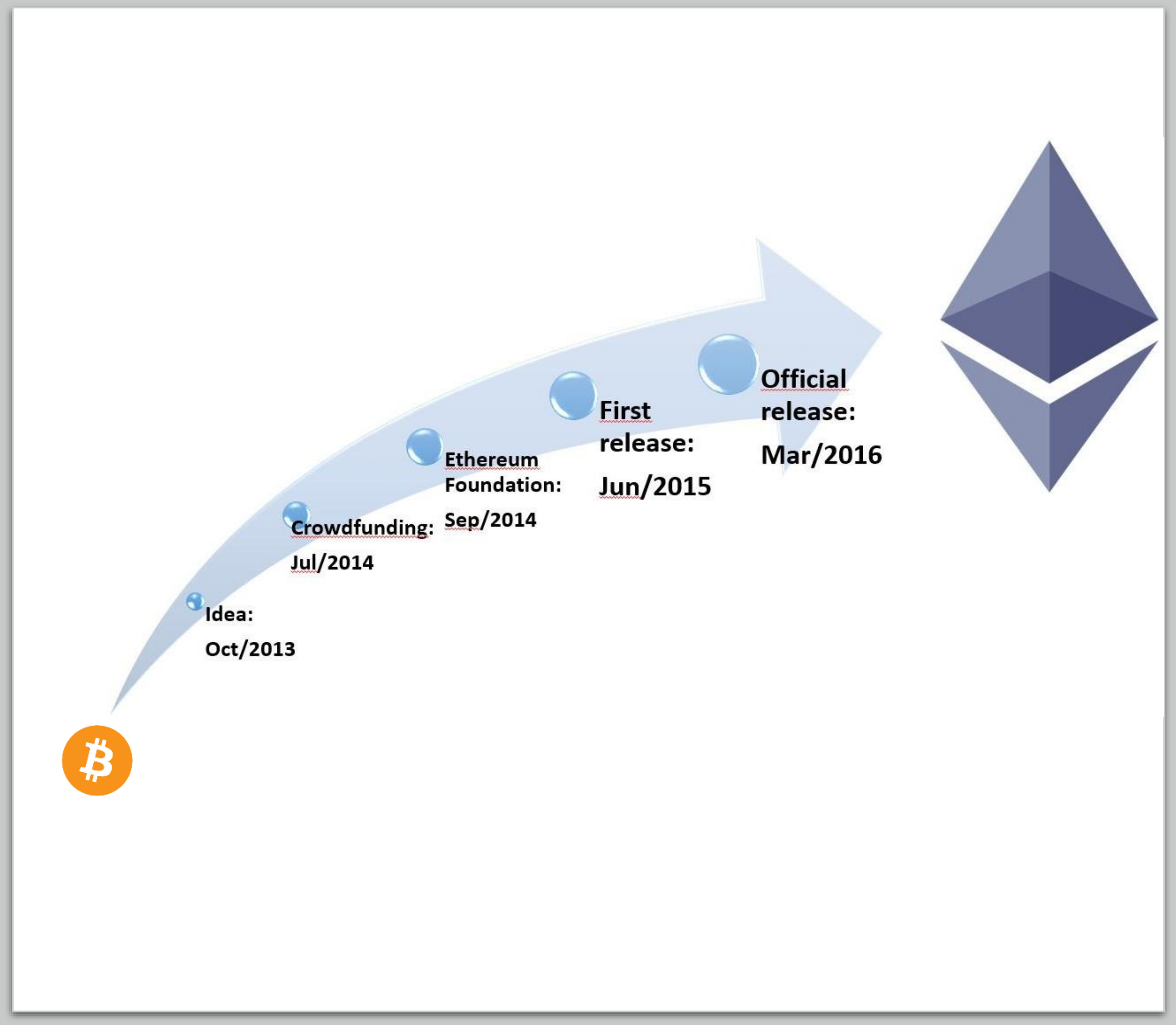
Based on the official definition, the Ethereum blockchain platform's focus is on facilitating development of dApps which are run by the nodes in the network



```
pragma solidity ^0.5.1;
contract Piggybank
{
    // Piggybank contract that allows
    //spending if savings are > 1 ETH
        address payable public owner; // Contract owner
        constructor() public
        {
            owner = msg.sender;
        }
    // Creation of contract
        function() payable external
        {
        }
    // Saving funds
        function spend() public
        {
            require(msg.sender == owner);
            require(address(this).balance >= 30 ether);
            owner.transfer(address(this).balance);
        }
}
```

Overview

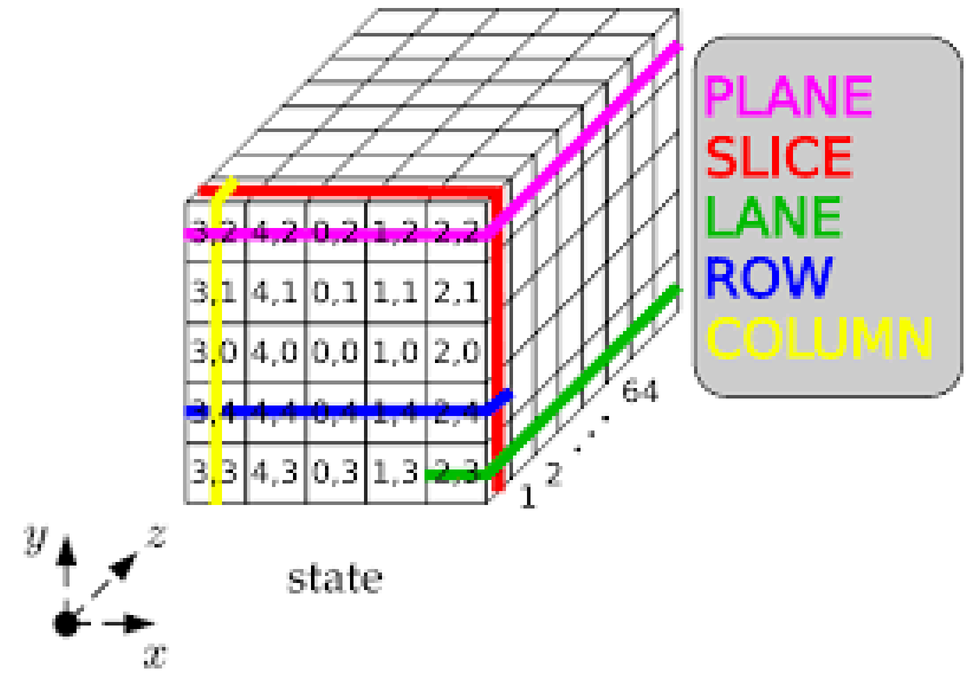
- Ethereum was conceived in 2013 by Vitalik Buterin, a Canadian-Russian Bitcoin programmer.
- Officially launched in 2015
- It is now one of the most popular blockchains, only second to Bitcoin



Cryptography

Hash Functions

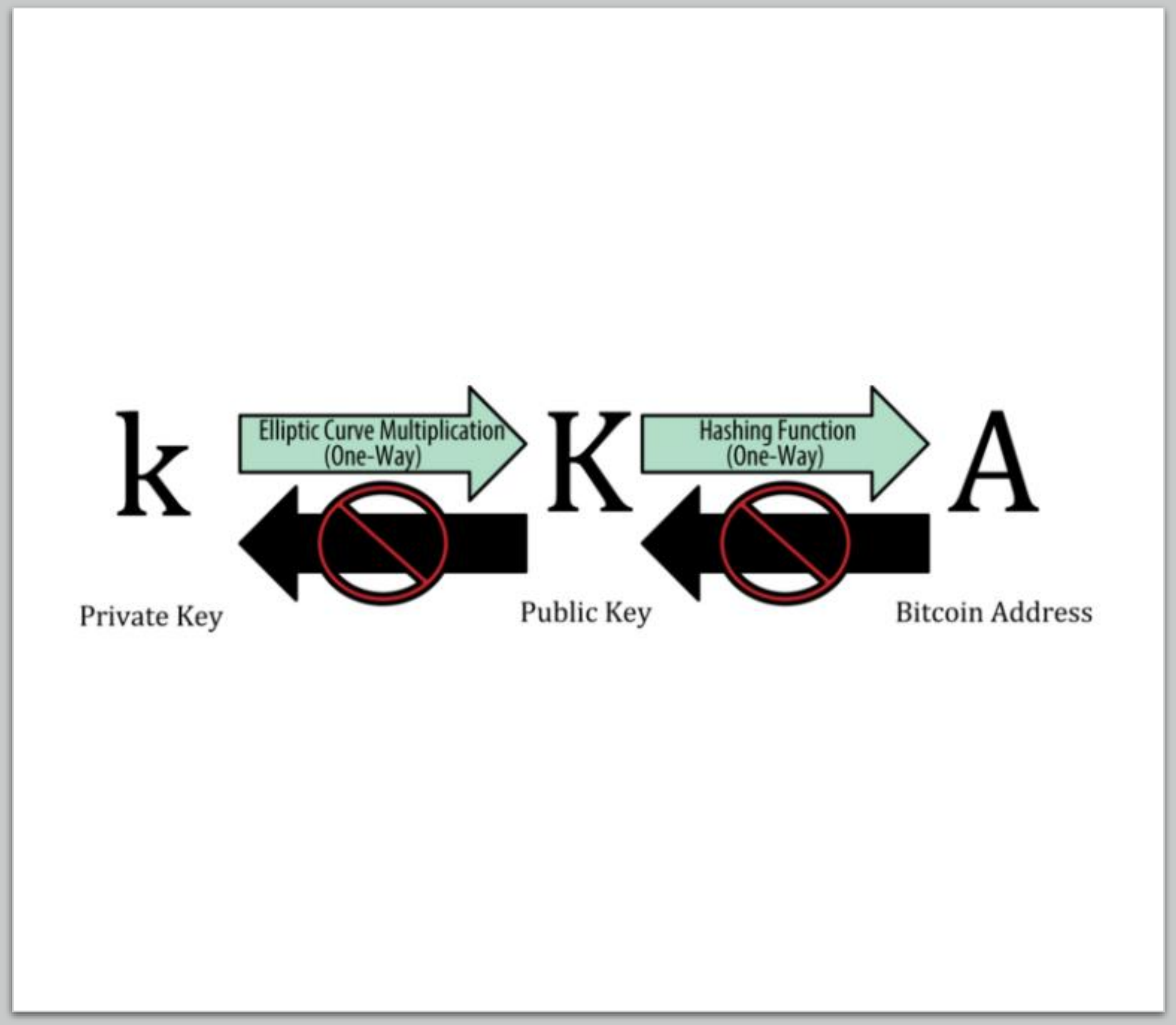
- BTC uses SHA2-256 cryptographic hash function
- Ethereum uses SHA3-256 cryptographic hash function
- SHA3 family is based on Keccak cryptographic hash algorithm
- The SHA-3 functions are designed to provide special properties, such as resistance to collision
- Used for all hashing in Ethereum



Cryptography

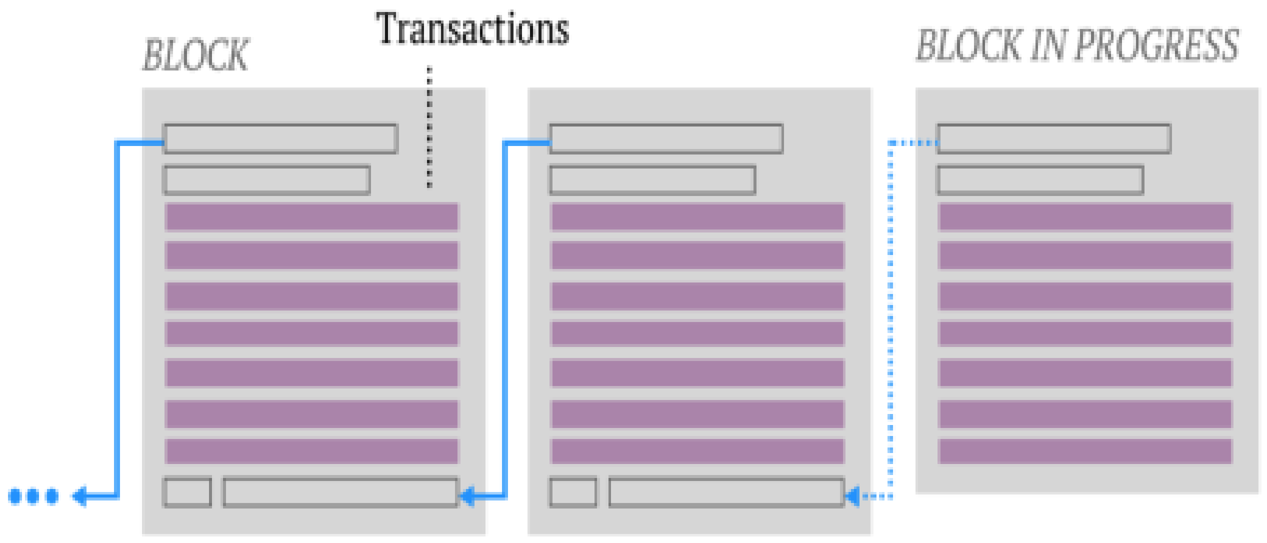
Digital Signatures

- Same use-case/cryptographic method (Elliptic Curve Digital Signature Algorithm-ECDSA) as BTC
- Signer uses private key to generate a signed message
- Signed message can be verified using the signer's public key
- Hashes are signed in Ethereum, not the data itself


k
Elliptic Curve Multiplication (One-Way)
K
Hashing Function (One-Way)
A
Private Key
Public Key
Bitcoin Address

Architecture

Fully distributed database like BTC



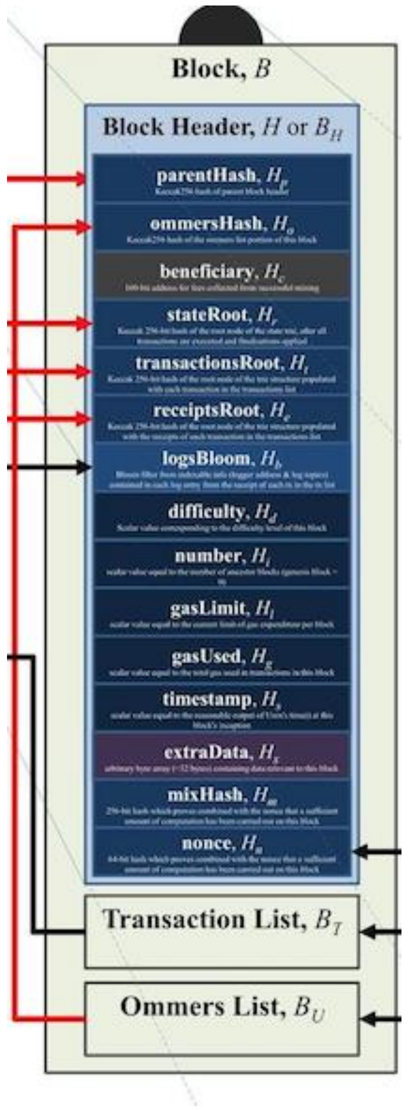
Architecture

Blocks consist of 3 elements

i. **Transaction List**: List of all transactions included in a block

ii. **Block Header**: Group of 15 elements (parentHash, nonce, gasLimit, difficulty etc.)

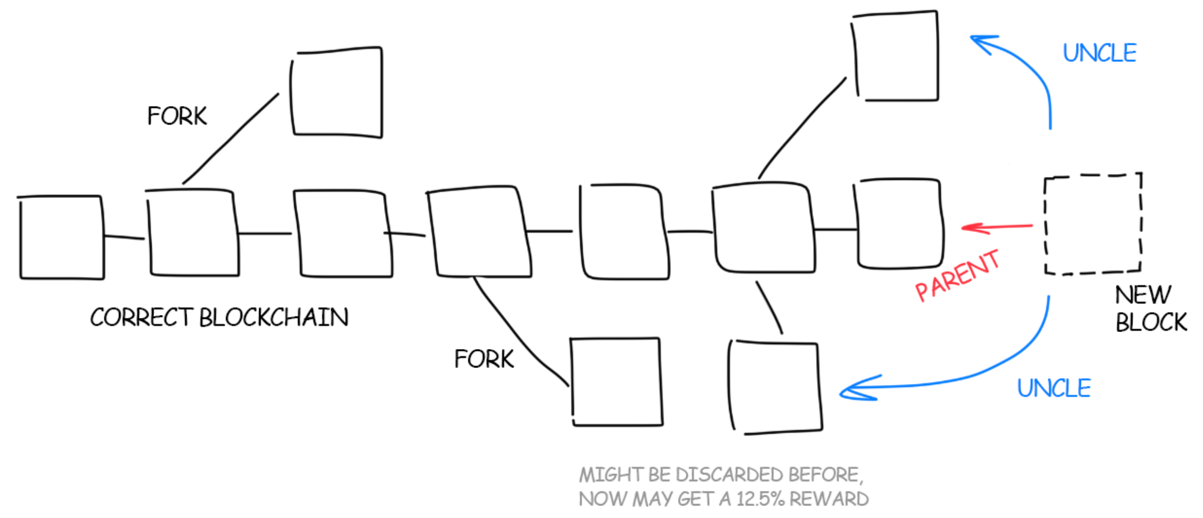
iii. **Ommer List**: List of all Uncle blocks included (described later)



Architecture

Uncles/Ommers

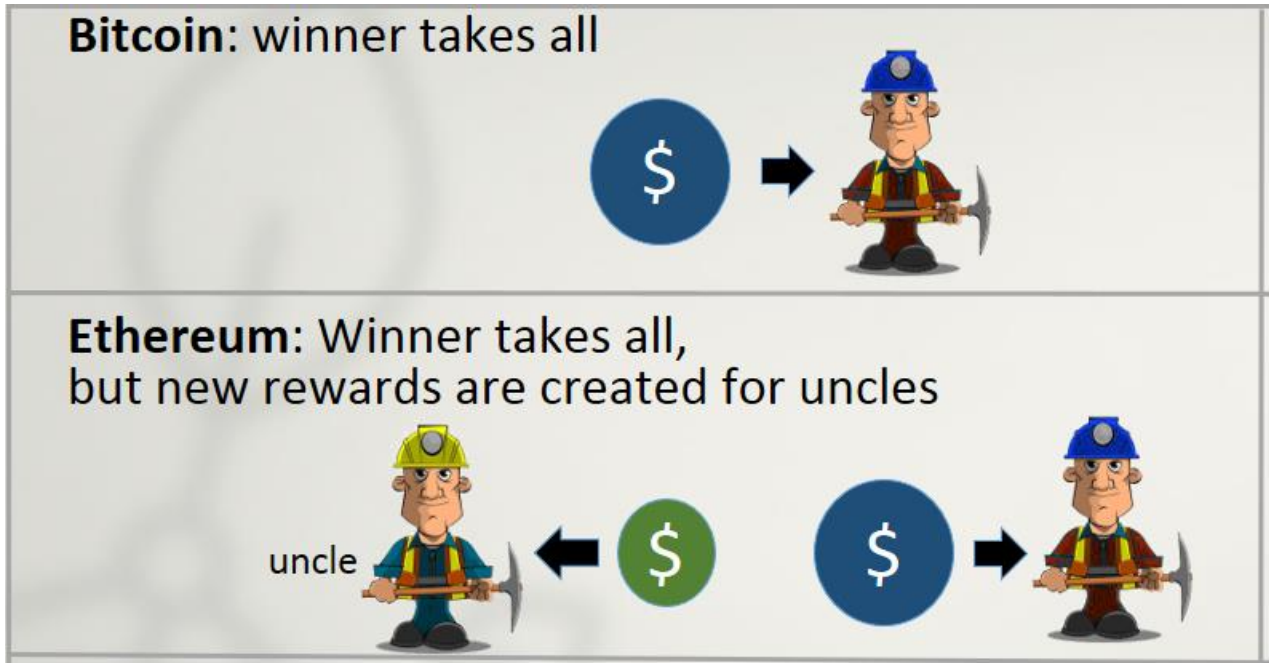
- An Uncle is a block whose parent is equal to the current block's parent's parent
- Sometimes valid block solutions don't make main chain
- Any broadcast block with valid PoW and difficulty can be included as an uncle
- Maximum of two can be included per block
- Has to be within 6 previous blocks
- Uncle block transactions are <u>not</u> included – just header
- Aimed to decrease centralization and reward work



Architecture

Uncles/Ommers Rewards:

- Uncle headers can be included in main block for 1/32 of the main block miner's reward given to said miner
- Miners of uncle blocks receive percent of main reward according to:
    - $(U_n + (8 - B_n)) * 5 / 8$, where U_n and B_n are uncle and block numbers respectively.
    - Example (1333 + 8 - 1335) * ⅝ = 3.75 ETH

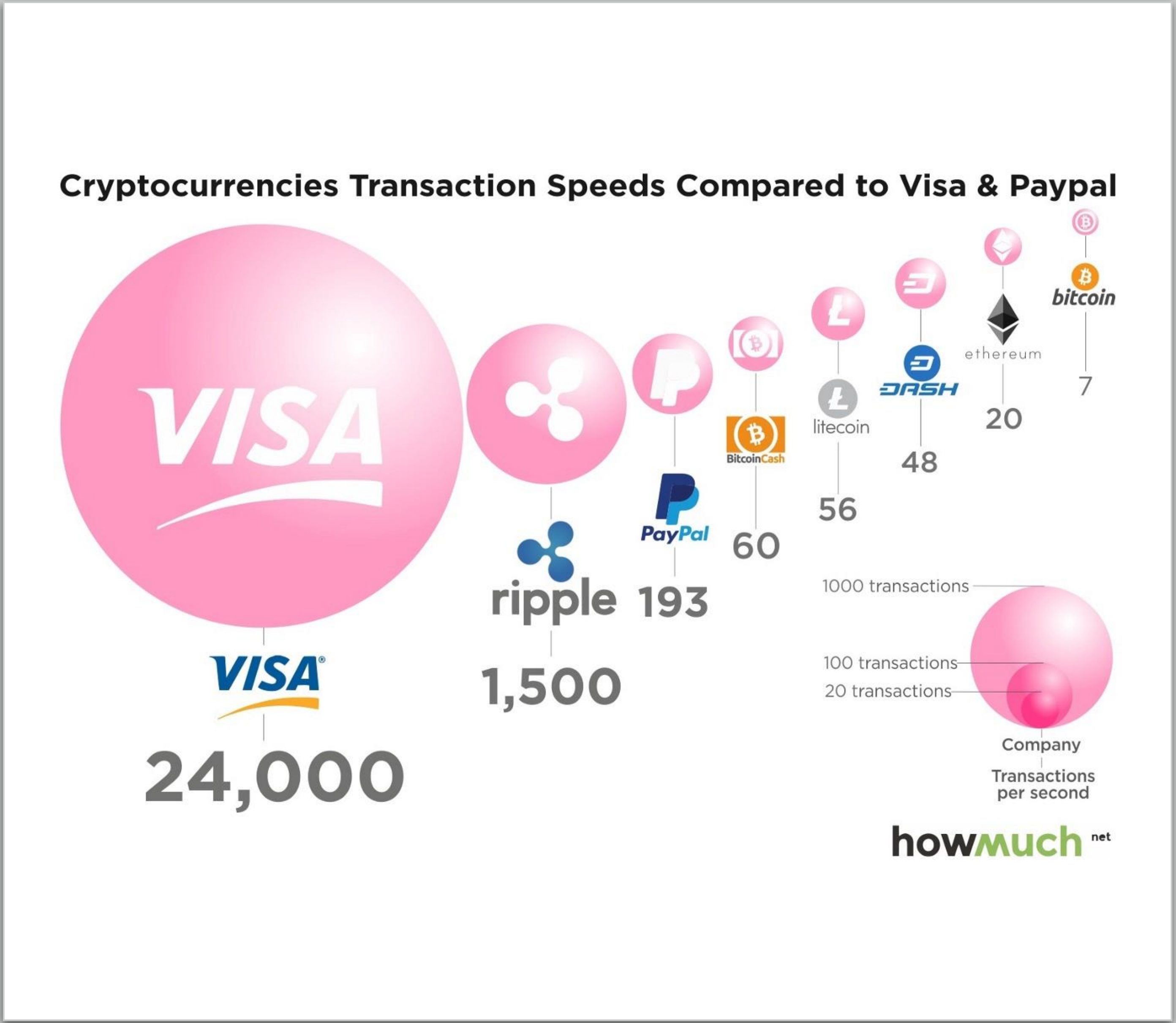

Bitcoin: winner takes all
$
Ethereum: Winner takes all,
but new rewards are created for uncles
uncle
$
$

Architecture

Block Structure:

Blocks faster than BTC

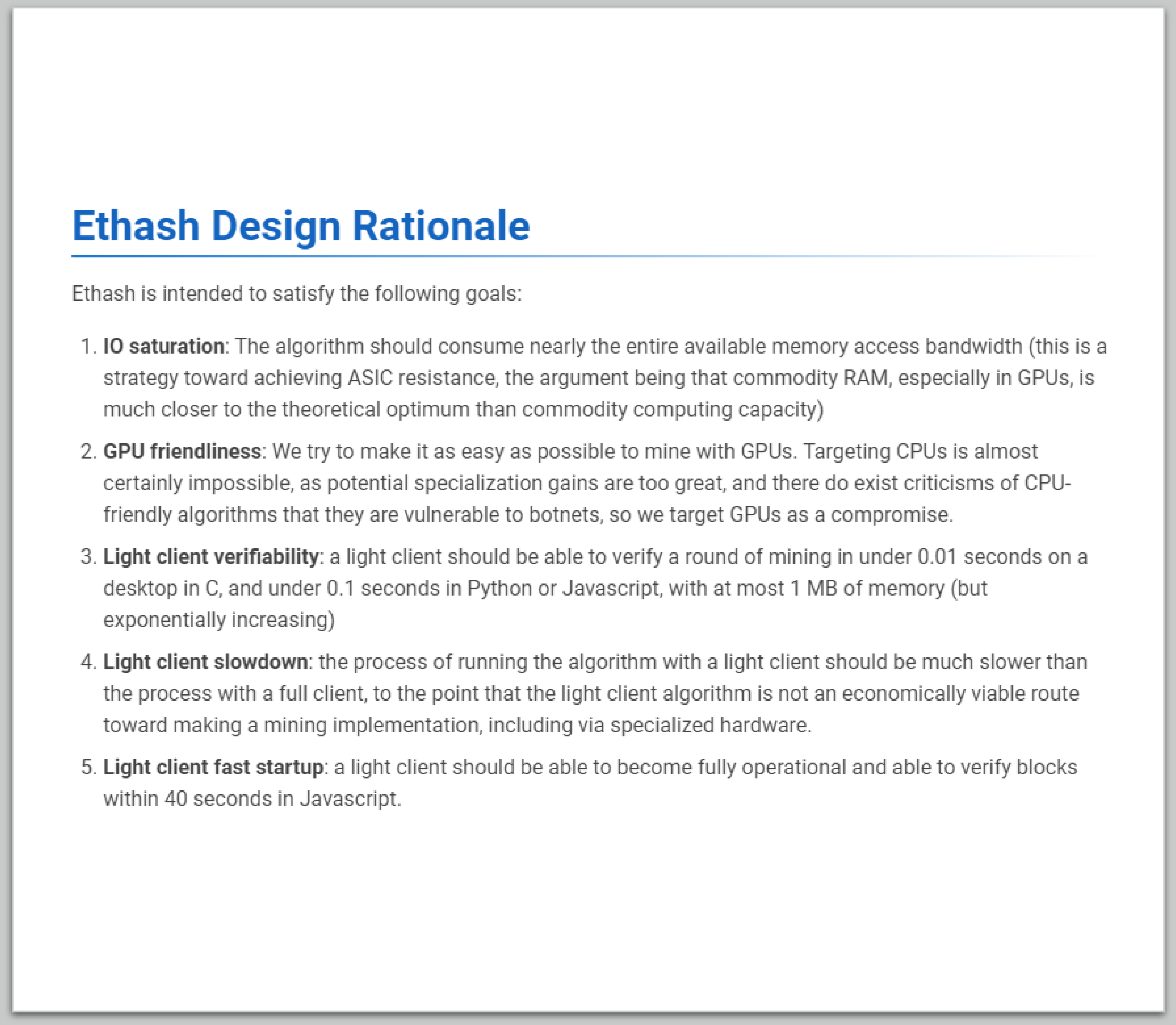
- A block is created every 12 secs.
- About 20 transactions per second (tps)
- In BTC block time is ~ 10 mins (~7 tps)
- Ethereum 2.0 is expected to deliver ~15,000 tps



Architecture

Block Structure:

- Uses Ethash mining algorithm (different than Bitcoin)
  - Helps mitigate ASIC and GPU advantages
  - Involves smart contract execution
- Difficulty is adjusted every block (not every two weeks as for BTC)



Ethash Design Rationale

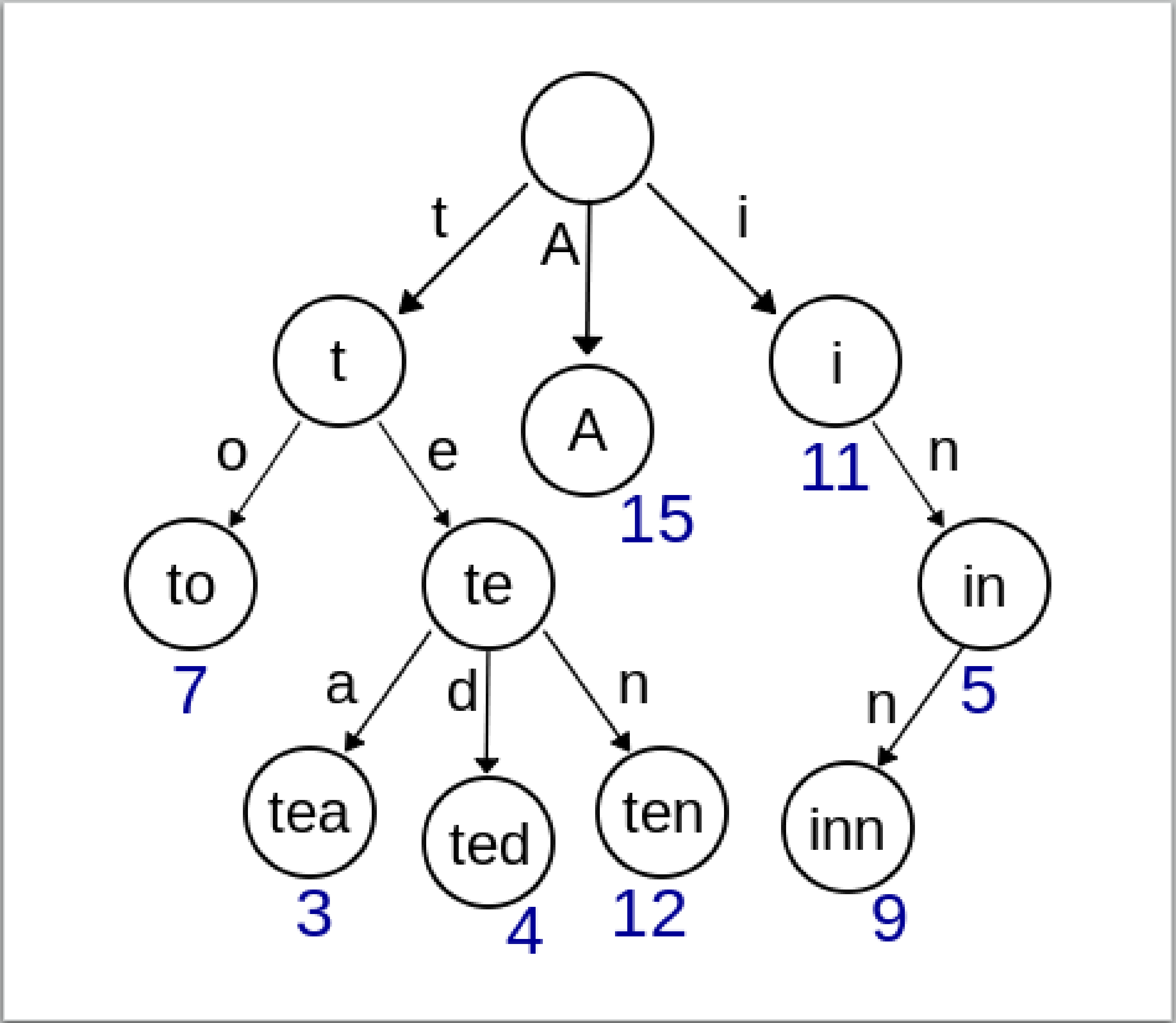
Ethash is intended to satisfy the following goals:

1. **IO saturation**: The algorithm should consume nearly the entire available memory access bandwidth (this is a strategy toward achieving ASIC resistance, the argument being that commodity RAM, especially in GPUs, is much closer to the theoretical optimum than commodity computing capacity)
2. **GPU friendliness**: We try to make it as easy as possible to mine with GPUs. Targeting CPUs is almost certainly impossible, as potential specialization gains are too great, and there do exist criticisms of CPU-friendly algorithms that they are vulnerable to botnets, so we target GPUs as a compromise.
3. **Light client verifiability**: a light client should be able to verify a round of mining in under 0.01 seconds on a desktop in C, and under 0.1 seconds in Python or Javascript, with at most 1 MB of memory (but exponentially increasing)
4. **Light client slowdown**: the process of running the algorithm with a light client should be much slower than the process with a full client, to the point that the light client algorithm is not an economically viable route toward making a mining implementation, including via specialized hardware.
5. **Light client fast startup**: a light client should be able to become fully operational and able to verify blocks within 40 seconds in Javascript.

Architecture

Block Structure:

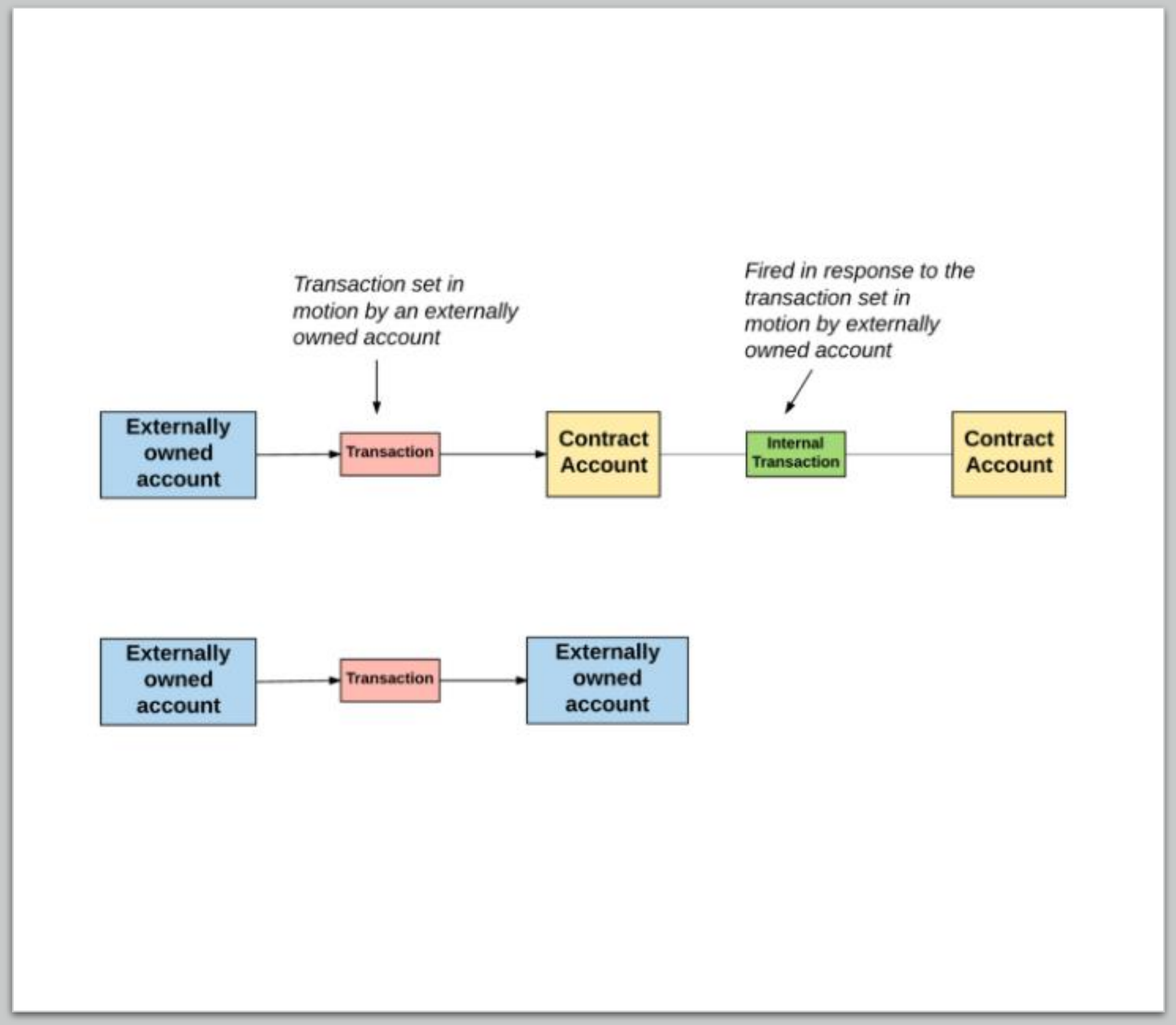
- Blocks keep track of balances – not "unspent transaction outputs" like BTC
- Merkle-Patricia Trie used (they have three branches compared to the Merkle tree's two)
- Will transition from Proof of Work to Proof of Stake with Casper protocol



Architecture

Accounts:

- Two Kinds:
  - External Owned Accounts - (EOA, most common account)
  - Contract Accounts
- Consist of a public/private keypair
- Allow for interaction with the blockchain

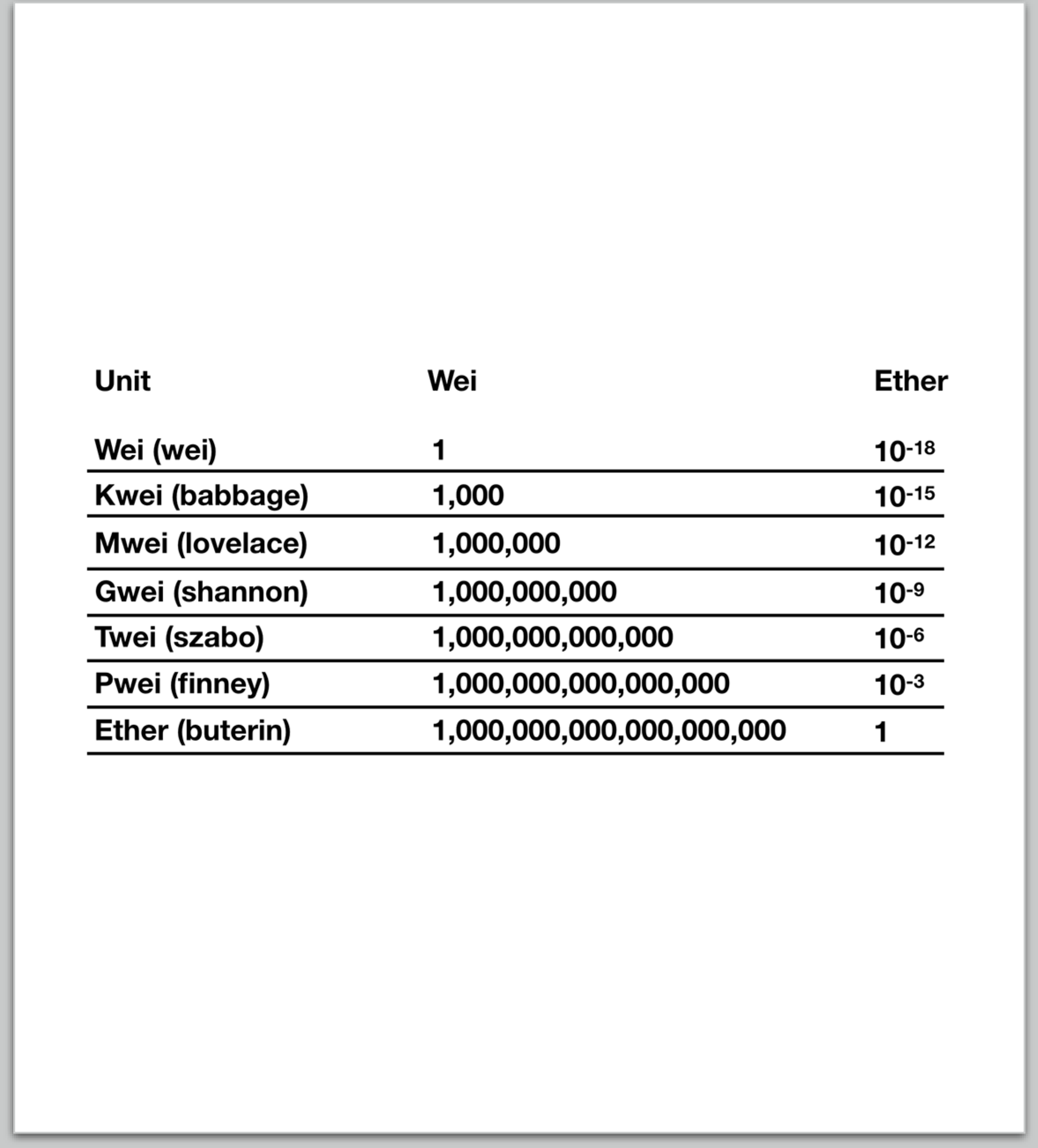


Ether

- Ethereum uses a coin called Ether.
- It is a number that can be stored, spent or received by Ethereum accounts

Ether Denominations

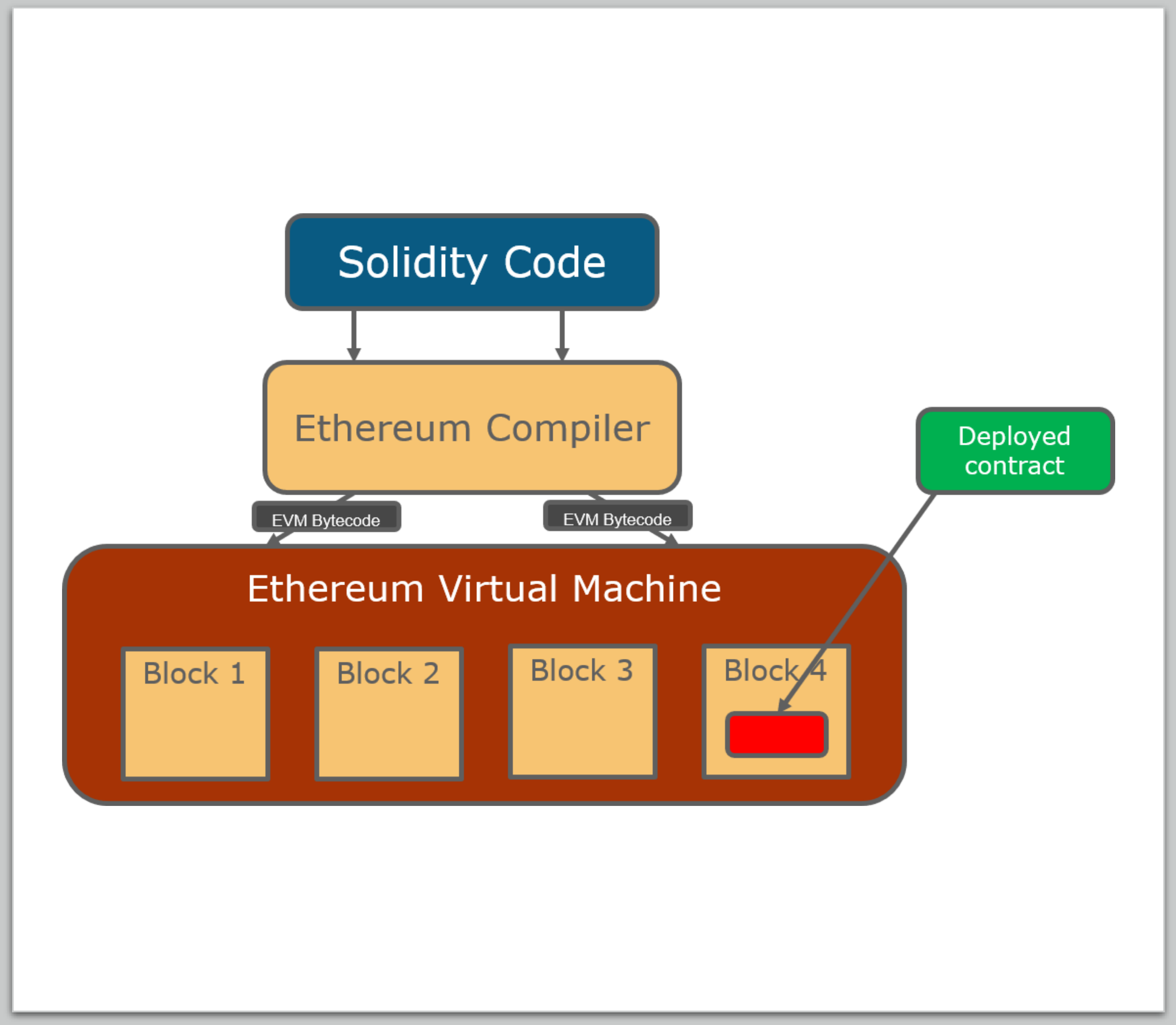
- Wei ($1x10^{-18}$ Ethers): Named after Wei Dai - author of B-Money (1998)
- Szabo ($1x10^{-6}$ Ethers): Named after Nick Szabo - author of Bit-Gold
- Finney ($1x10^{-3}$ Ethers): Named after Hal Finney - received first Tx from Nakamoto



Unit	Wei	Ether
Wei (wei)	1	10^{-18}
Kwei (babbage)	1,000	10^{-15}
Mwei (lovelace)	1,000,000	10^{-12}
Gwei (shannon)	1,000,000,000	10^{-9}
Twei (szabo)	1,000,000,000,000	10^{-6}
Pwei (finney)	1,000,000,000,000,000	10^{-3}
Ether (buterin)	1,000,000,000,000,000,000	1

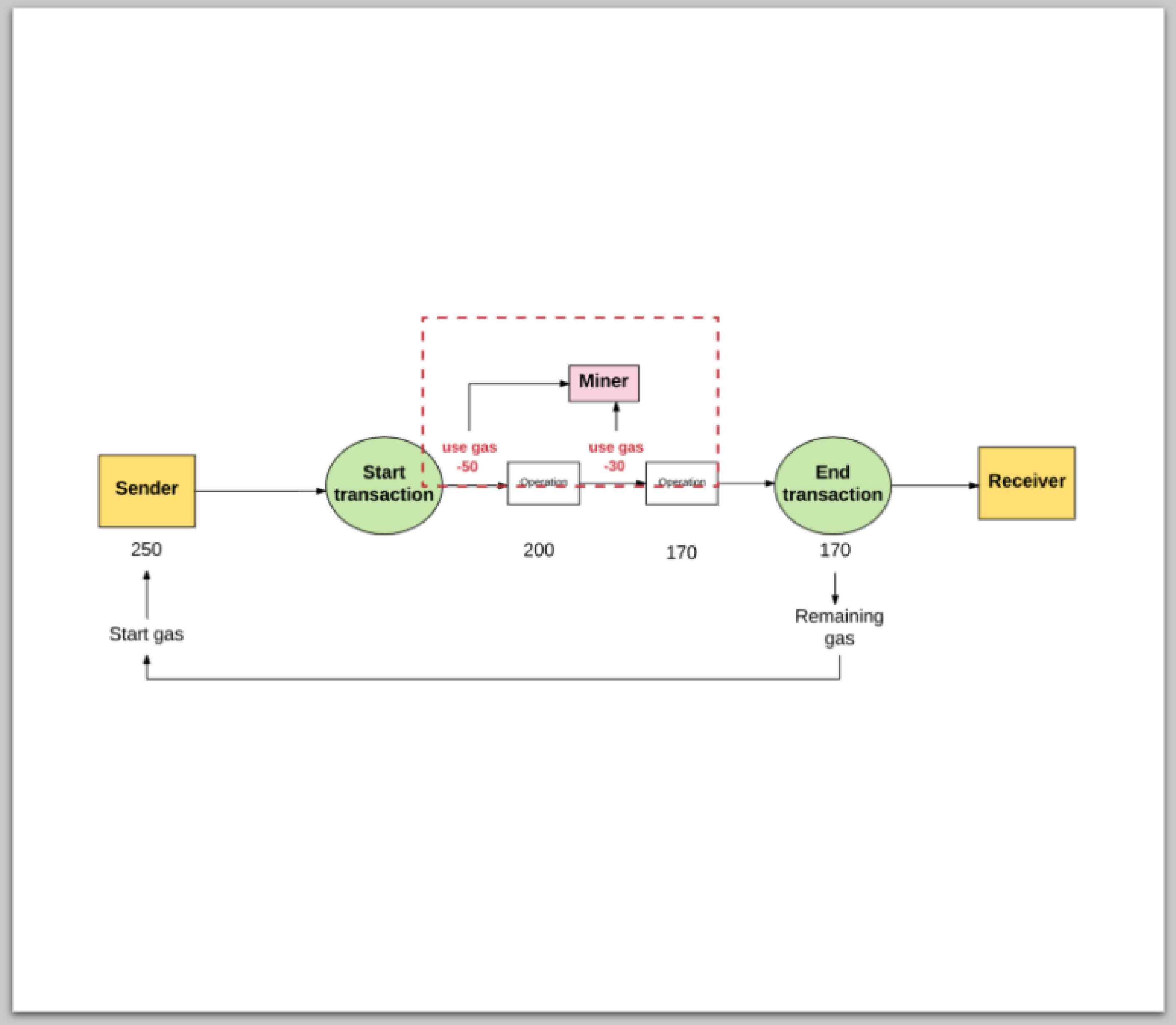
Ethereum Virtual Machine (EVM)

- Every node contains a virtual machine (similar to Java) called the Ethereum Virtual Machine (EVM)
- **Runs** code compiled from high-level language to bytecode
- Executes smart contract code and broadcasts state



Gas

- Every computation that occurs as a result of a transaction on the Ethereum network incurs a fee
- Gas is the unit used to measure the fees required for a particular computation
- Charging execution costs addresses the problem of infinite execution times, as any infinite loop will eventually be terminated when the contract runs out of gas.
- Every transaction needs to specify an estimate of the amount of gas it will spend

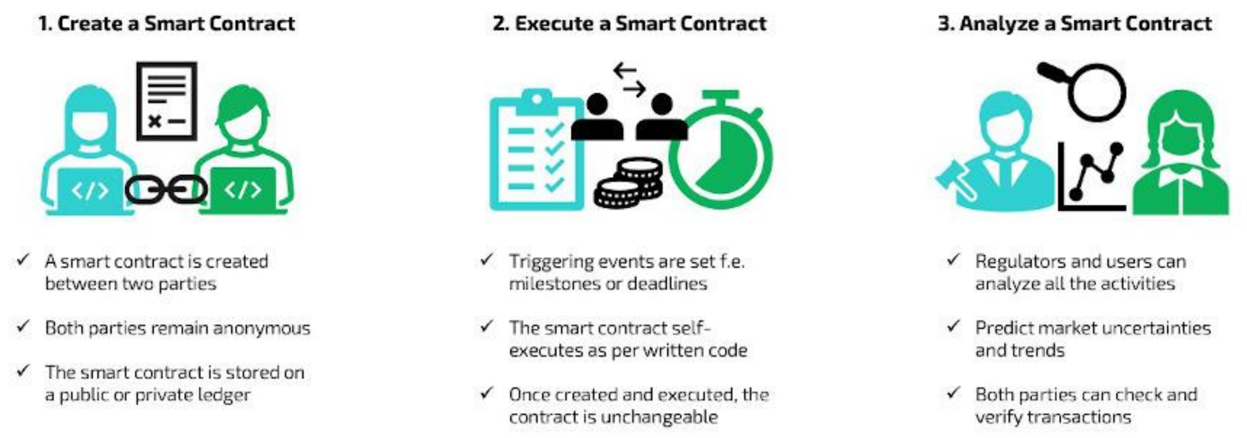


Gas Cost

- 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 gasLimit*gasPrice.
- Ideally, the program terminates before the gas limit is reached, the originator pays for the gas the smart contract has consumed.
- If all the gas is used up, the smart contract terminates.

Smart Contracts

- Smart contracts are computer programs that run as decentralized applications (dApps)
- They are self-executing contracts with the terms of the contract between the buyer and seller directly written into the program.
- Smart contracts permit trusted transactions and agreements to be carried out among disparate, anonymous parties without the need for a central authority.



1. Create a Smart Contract

- ✓ A smart contract is created between two parties
- ✓ Both parties remain anonymous
- ✓ The smart contract is stored on a public or private ledger

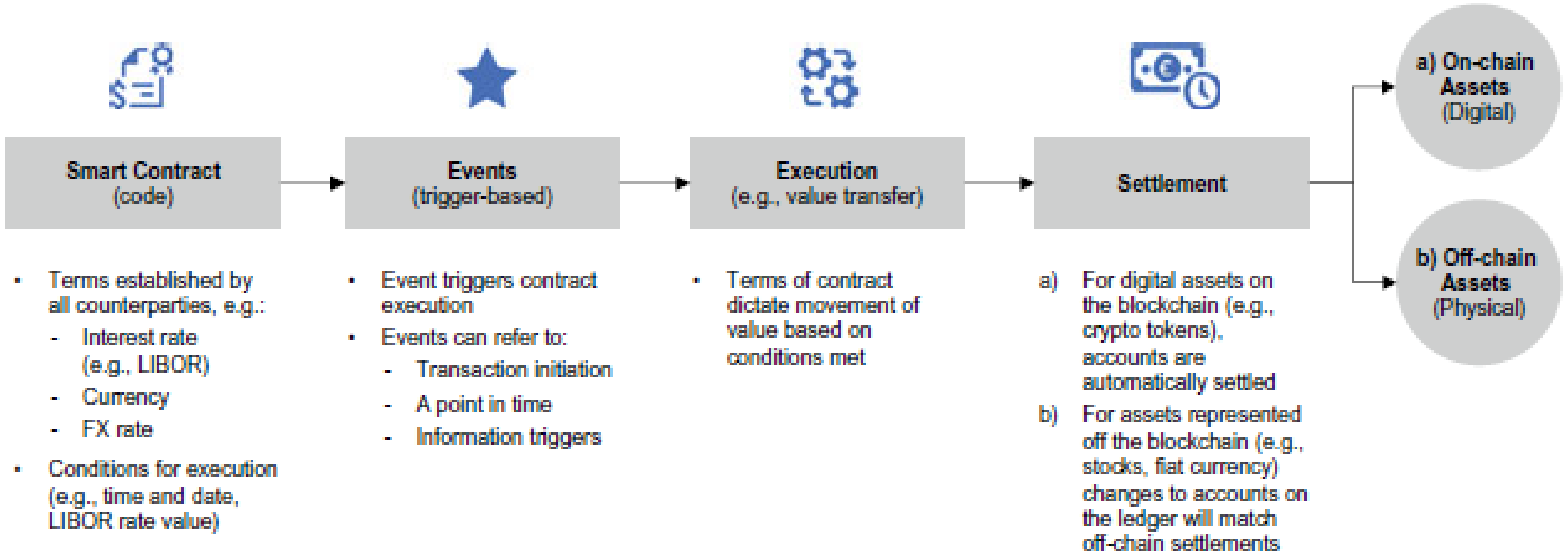
2. Execute a Smart Contract

- ✓ Triggering events are set f.e. milestones or deadlines
- ✓ The smart contract self-executes as per written code
- ✓ Once created and executed, the contract is unchangeable

3. Analyze a Smart Contract

- ✓ Regulators and users can analyze all the activities
- ✓ Predict market uncertainties and trends
- ✓ Both parties can check and verify transactions

Smart Contracts



Smart Contract (code)
Events (trigger-based)
Execution (e.g., value transfer)
Settlement
a) On-chain Assets (Digital)
b) Off-chain Assets (Physical)

- Terms established by all counterparties, e.g.:
  - Interest rate (e.g., LIBOR)
  - Currency
  - FX rate
- Conditions for execution (e.g., time and date, LIBOR rate value)

- Event triggers contract execution
- Events can refer to:
  - Transaction initiation
  - A point in time
  - Information triggers

- Terms of contract dictate movement of value based on conditions met

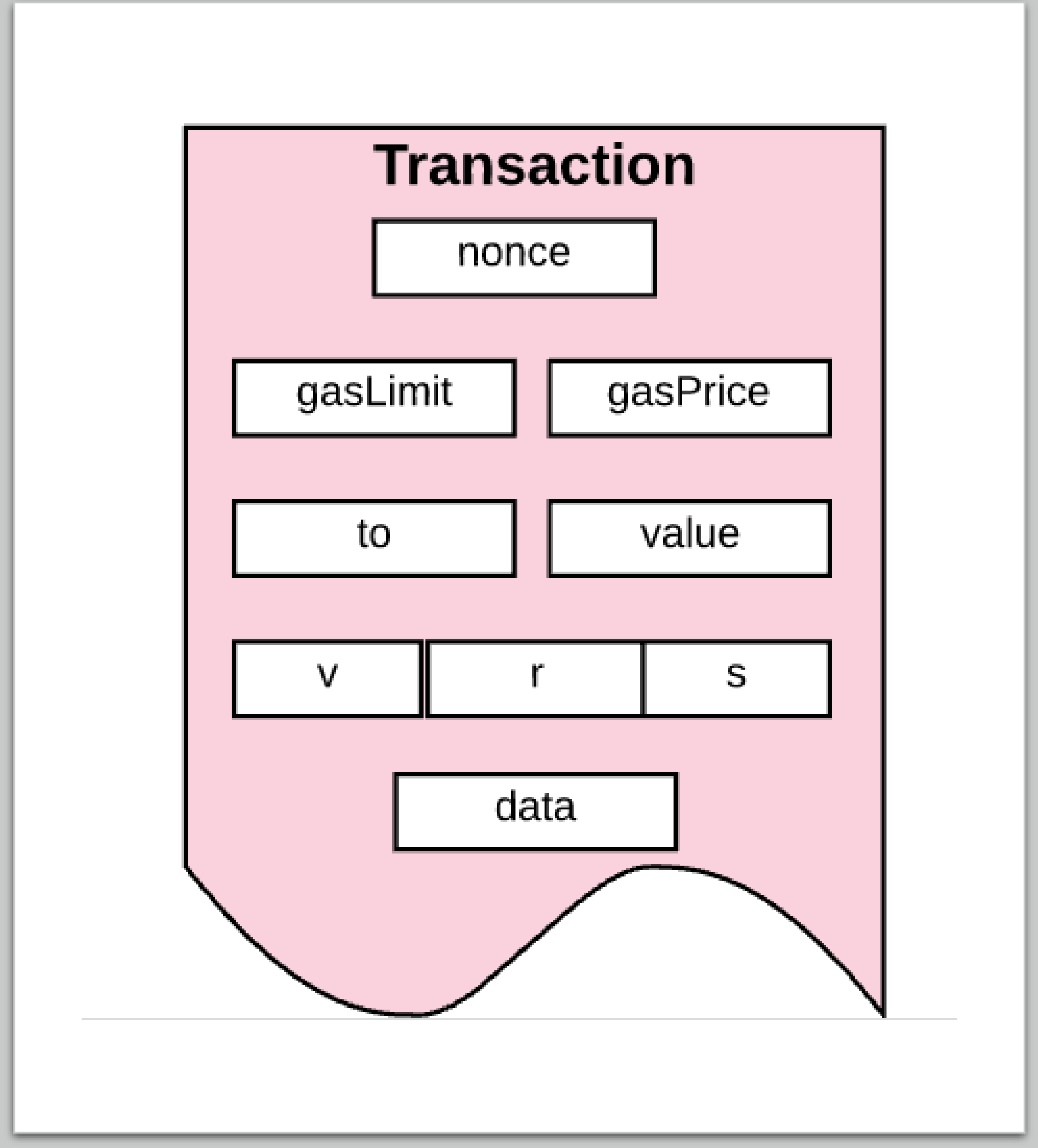
a) For digital assets on the blockchain (e.g., crypto tokens), accounts are automatically settled

b) For assets represented off the blockchain (e.g., stocks, fiat currency) changes to accounts on the ledger will match off-chain settlements

Smart Contracts

Smart contracts share the same transaction structure as normal transactions

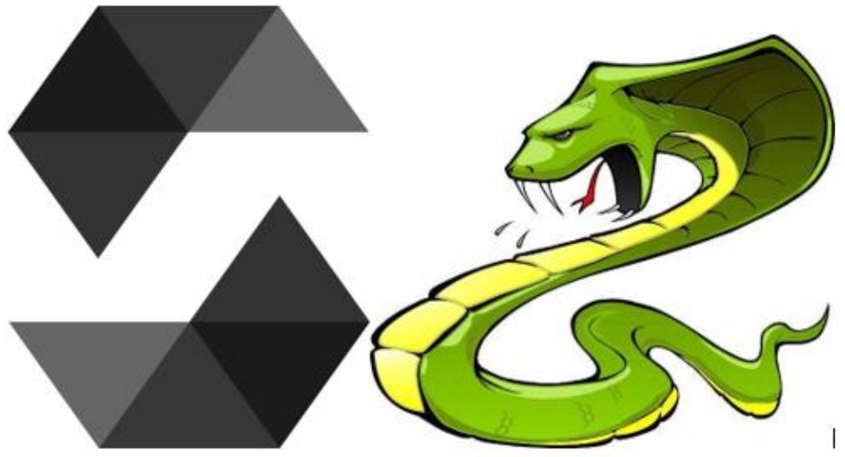
- **nonce**: a count of the number of transactions sent by the sender.
- **gasPrice**: the number of Wei that the sender is willing to pay per unit of gas required to execute the transaction.
- **gasLimit**: the maximum amount of gas that the sender is willing to pay for executing this transaction. This amount is set and paid upfront, before any computation is done.
- **to**: the address of the recipient. In a contract-creating transaction an empty value is used.
- **value**: the amount of Wei to be transferred from the sender to the recipient. In a contract-creating transaction, this value serves as the starting balance within the newly created contract account.
- **v, r, s**: used to generate the signature that identifies the sender of the transaction.
- **init** (only exists for contract-creating transactions): An EVM code fragment that is used to initialize the new contract account.
- **data** (optional field that only exists for message calls)



Smart Contracts Programming

- **Solidity**: Solidity is an object-oriented, high-level language for implementing smart contracts. It is a curly-bracket language influenced by C++, Python and JavaScript
- **Serpent**: Inspired by Python. Like Python, it has a simple, minimal syntax, dynamic typing, and support for object-oriented programming.
- **Vyper:** is a contract-oriented, pythonic programming language.
- **Lisp Like Language (LLL)**: Low level language similar to assembly

In this training we shall use the solidity programming language to create smart contracts.

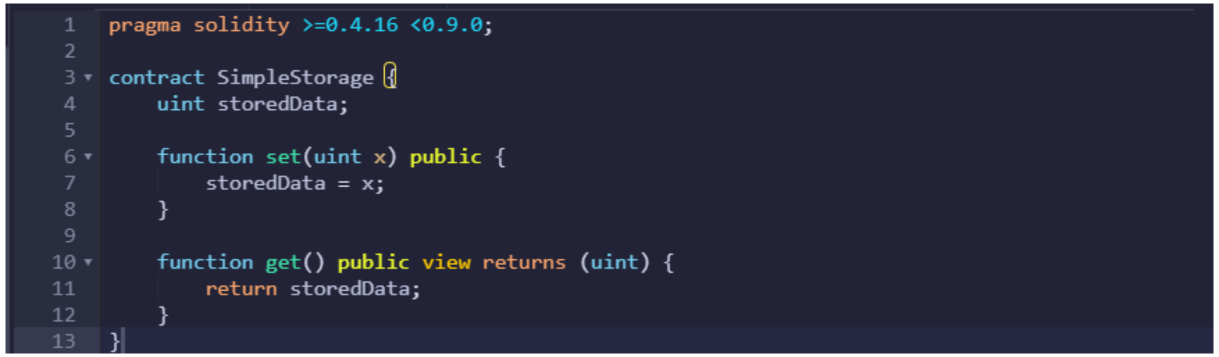


Smart Contracts Programming

- Solidity is a object-oriented, high-level programming language for Ethereum,
- It uses JavaScript-like syntax, which allows for modern programming constructs like abstraction, interfaces, and polymorphism
- Solidity is used primarily for developing smart contracts that can be compiled into bytecode for the EVM,
- The bytecodes are uploaded in turn to the Ethereum blockchain (e.g., through Geth Console).
- There are several methods by which to compile Solidity code, including the online compiler, the command-line Solidity compiler solc, and the compiler built into Ethereum.

Smart Contracts Programming

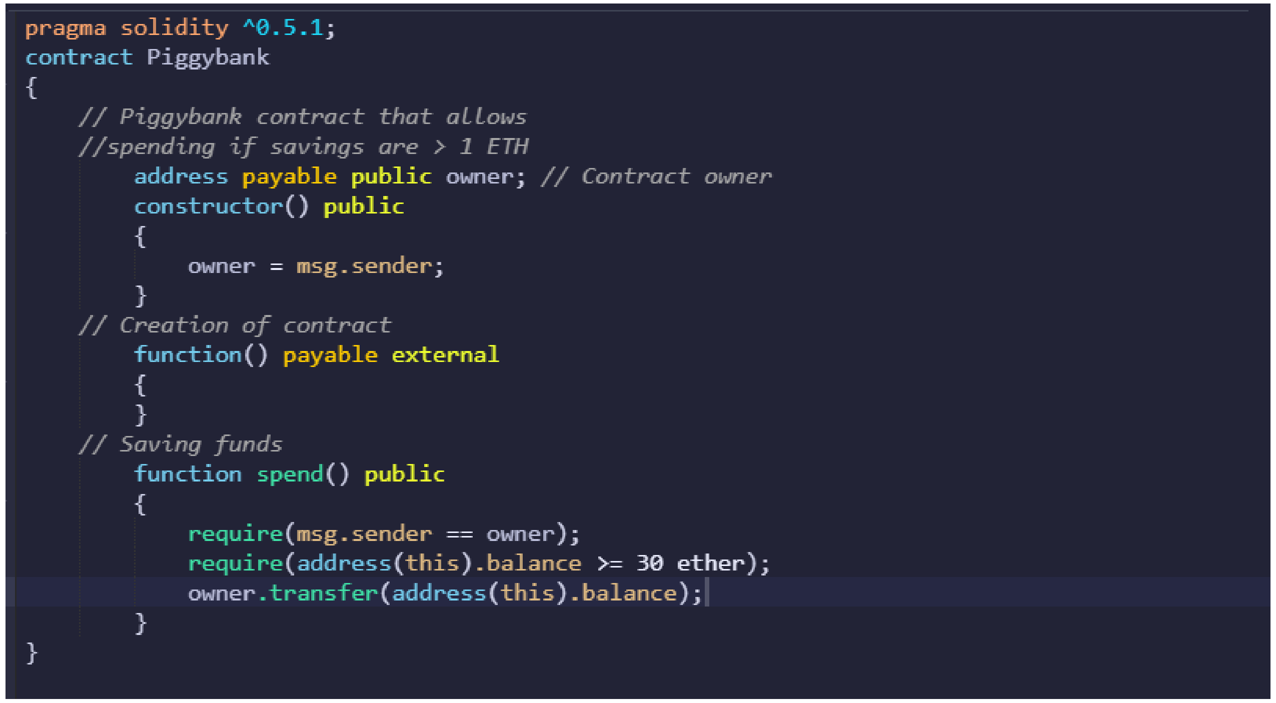
Example: Storage. This smart contract allows any user to store a number in it and any user to retrieve that number.



```
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 Contracts Programming

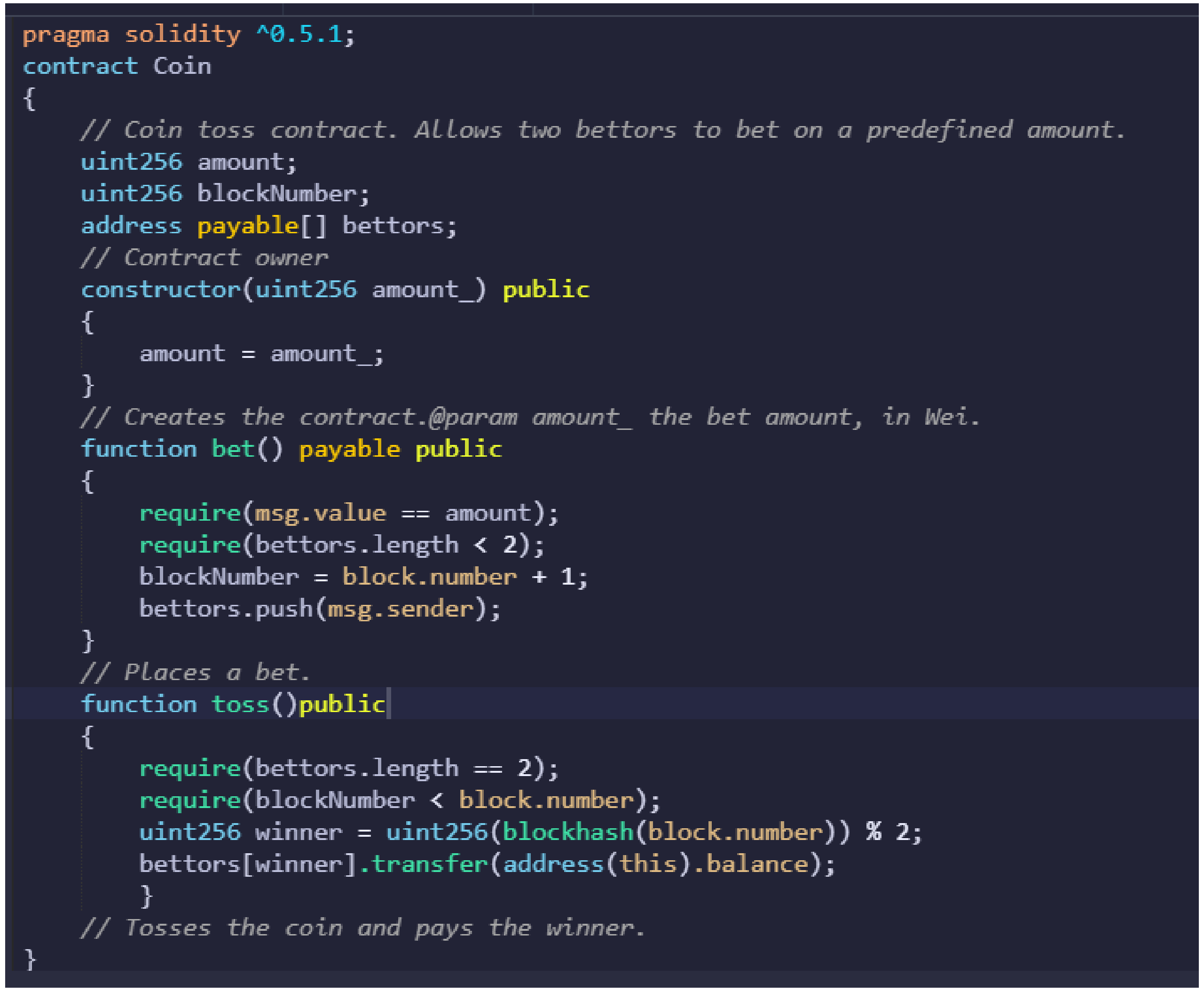
Example: Piggy Bank. This smart contract allows any user to deposit ethers into a contract, but only the owner of the contract can take the money out after it has accumulated to 30 ETH.



```
pragma solidity ^0.5.1;
contract Piggybank
{
    // Piggybank contract that allows
    //spending if savings are > 1 ETH
        address payable public owner; // Contract owner
        constructor() public
        {
            owner = msg.sender;
        }
    // Creation of contract
        function() payable external
        {
        }
    // Saving funds
        function spend() public
        {
            require(msg.sender == owner);
            require(address(this).balance >= 30 ether);
            owner.transfer(address(this).balance);
        }
}
```

Smart Contracts Programming

Example: Coin Toss. This smart contract allow two bettors stake a pre-defined amount to bet head or tail, and the winner is determined randomly via the smart contract (50:50 Odds)



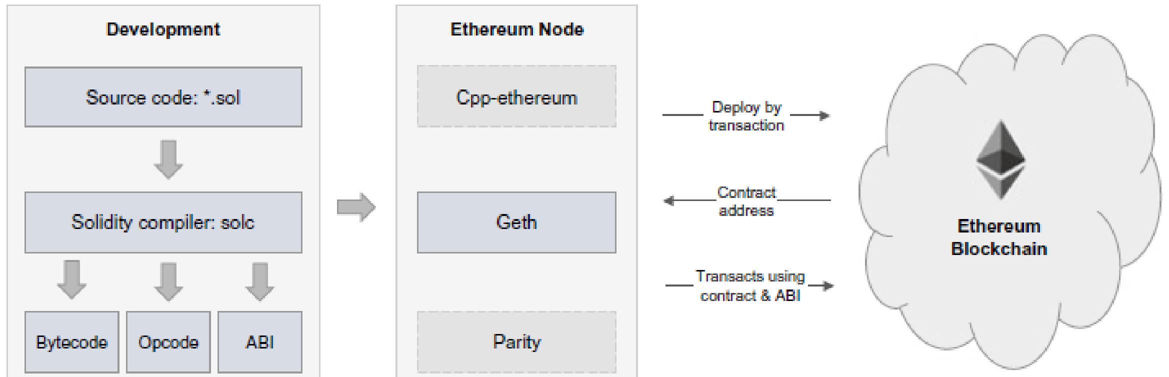
```
pragma solidity ^0.5.1;
contract Coin
{
    // Coin toss contract. Allows two bettors to bet on a predefined amount.
    uint256 amount;
    uint256 blockNumber;
    address payable[] bettors;
    // Contract owner
    constructor(uint256 amount_) public
    {
        amount = amount_;
    }
    // Creates the contract.@param amount_ the bet amount, in Wei.
    function bet() payable public
    {
        require(msg.value == amount);
        require(bettors.length < 2);
        blockNumber = block.number + 1;
        bettors.push(msg.sender);
    }
    // Places a bet.
    function toss()public
    {
        require(bettors.length == 2);
        require(blockNumber < block.number);
        uint256 winner = uint256(blockhash(block.number)) % 2;
        bettors[winner].transfer(address(this).balance);
        }
    // Tosses the coin and pays the winner.
}
```

Deploying Smart Contracts

To deploy smart contracts into a blockchain, one needs two components:

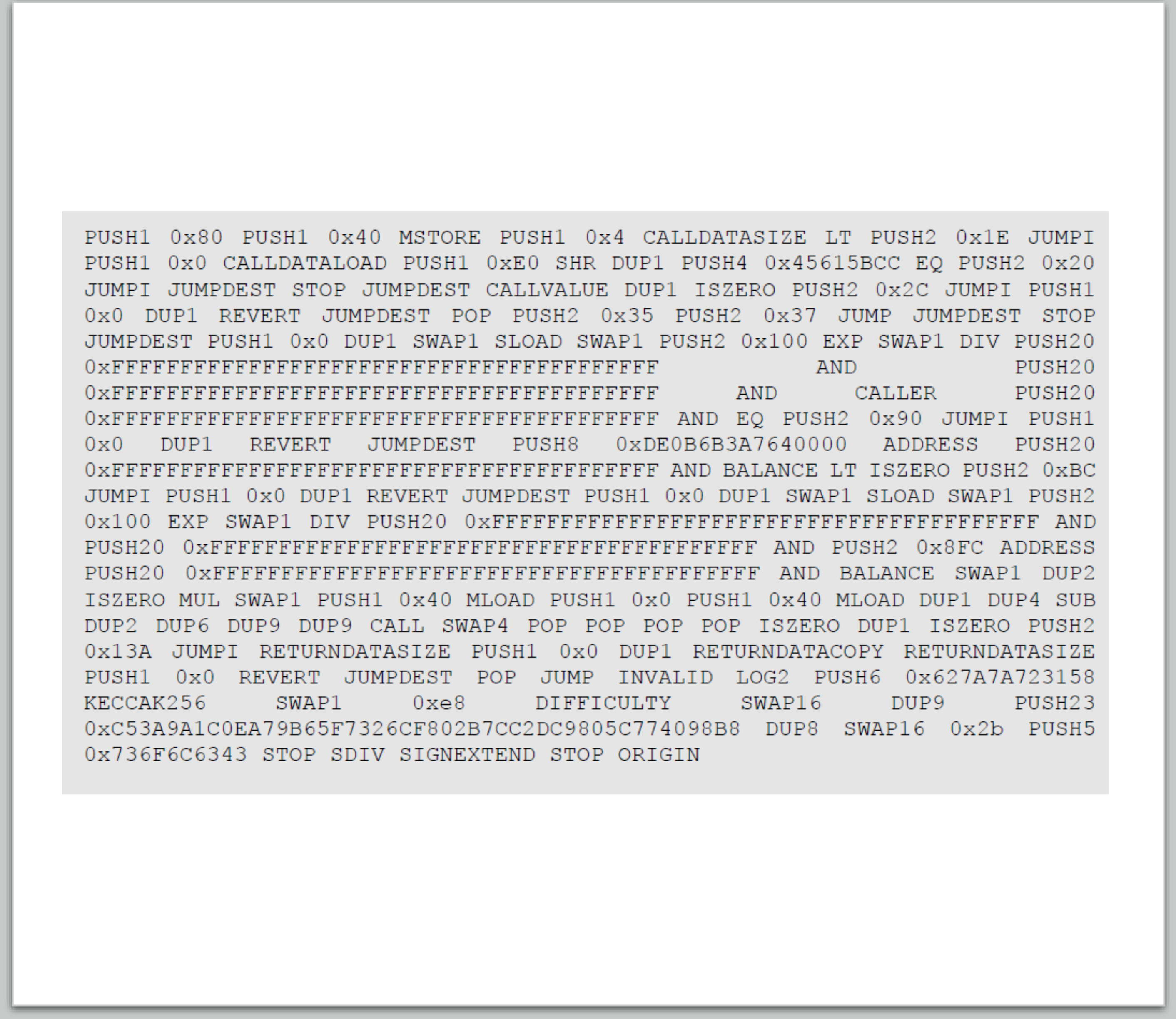
i. Bytecode

ii. ABI



Deploying Smart Contracts

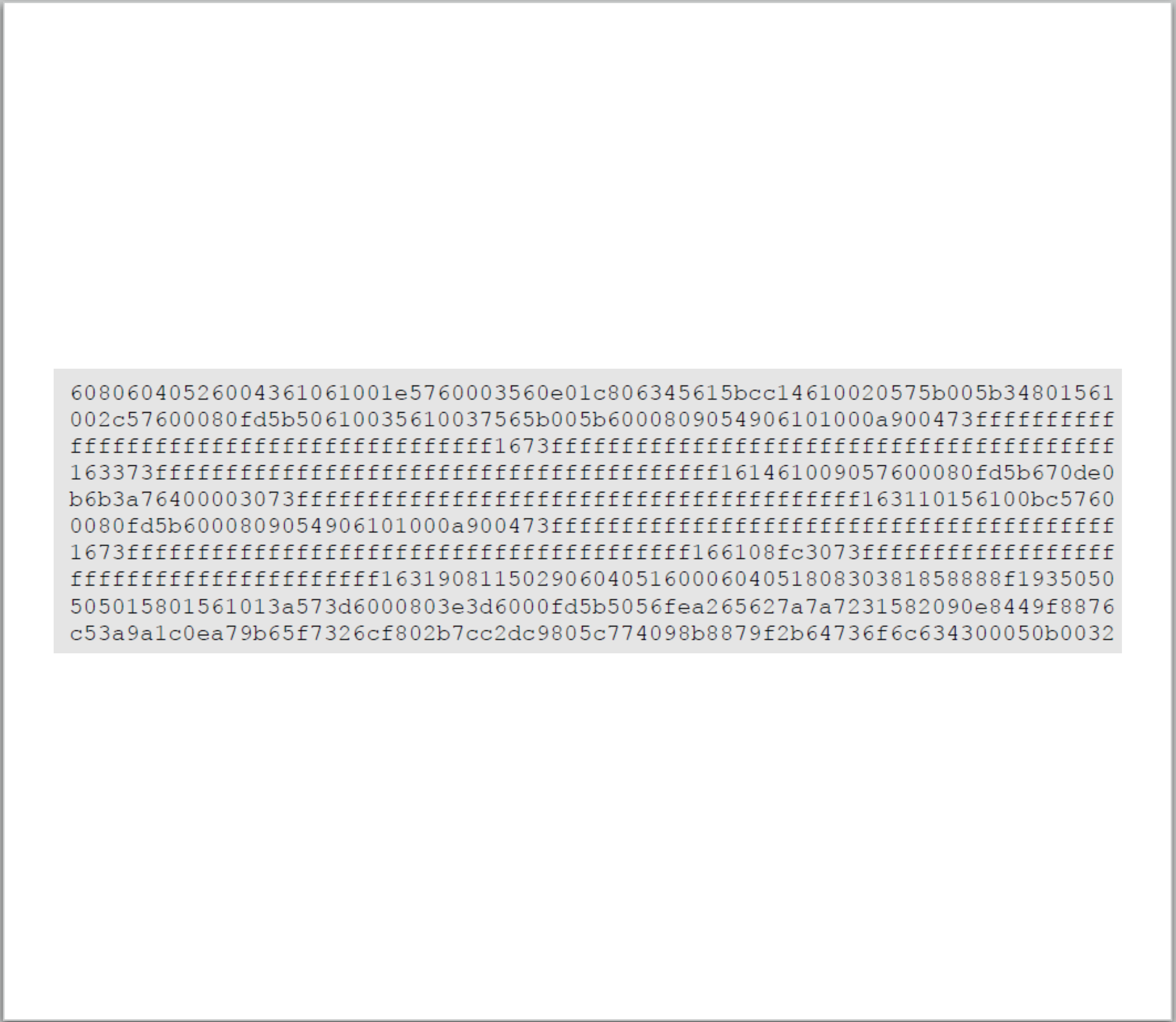
- The **opcode** is a pre-compiled version of Solidity code. The opcode is an intermediary step; it is not required for deploying a smart contract.
- Piggy Bank Opcode



```
PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4 CALLDATASIZE LT PUSH2 0x1E JUMPI
PUSH1 0x0 CALLDATALOAD PUSH1 0xE0 SHR DUP1 PUSH4 0x45615BCC EQ PUSH2 0x20
JUMPI JUMPDEST STOP JUMPDEST CALLVALUE DUP1 ISZERO PUSH2 0x2C JUMPI PUSH1
0x0 DUP1 REVERT JUMPDEST POP PUSH2 0x35 PUSH2 0x37 JUMP JUMPDEST STOP
JUMPDEST PUSH1 0x0 DUP1 SWAP1 SLOAD SWAP1 PUSH2 0x100 EXP SWAP1 DIV PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND CALLER PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND EQ PUSH2 0x90 JUMPI PUSH1
0x0 DUP1 REVERT JUMPDEST PUSH8 0xDE0B6B3A7640000 ADDRESS PUSH20
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND BALANCE LT ISZERO PUSH2 0xBC
JUMPI PUSH1 0x0 DUP1 REVERT JUMPDEST PUSH1 0x0 DUP1 SWAP1 SLOAD SWAP1 PUSH2
0x100 EXP SWAP1 DIV PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND PUSH2 0x8FC ADDRESS
PUSH20 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF AND BALANCE SWAP1 DUP2
ISZERO MUL SWAP1 PUSH1 0x40 MLOAD PUSH1 0x0 PUSH1 0x40 MLOAD DUP1 DUP4 SUB
DUP2 DUP6 DUP9 DUP9 CALL SWAP4 POP POP POP POP ISZERO DUP1 ISZERO PUSH2
0x13A JUMPI RETURNDATASIZE PUSH1 0x0 DUP1 RETURNDATACOPY RETURNDATASIZE
PUSH1 0x0 REVERT JUMPDEST POP JUMP INVALID LOG2 PUSH6 0x627A7A723158
KECCAK256 SWAP1 0xe8 DIFFICULTY SWAP16 DUP9 PUSH23
0xC53A9A1C0EA79B65F7326CF802B7CC2DC9805C774098B8 DUP8 SWAP16 0x2b PUSH5
0x736F6C6343 STOP SDIV SIGNEXTEND STOP ORIGIN
```

Deploying Smart Contracts

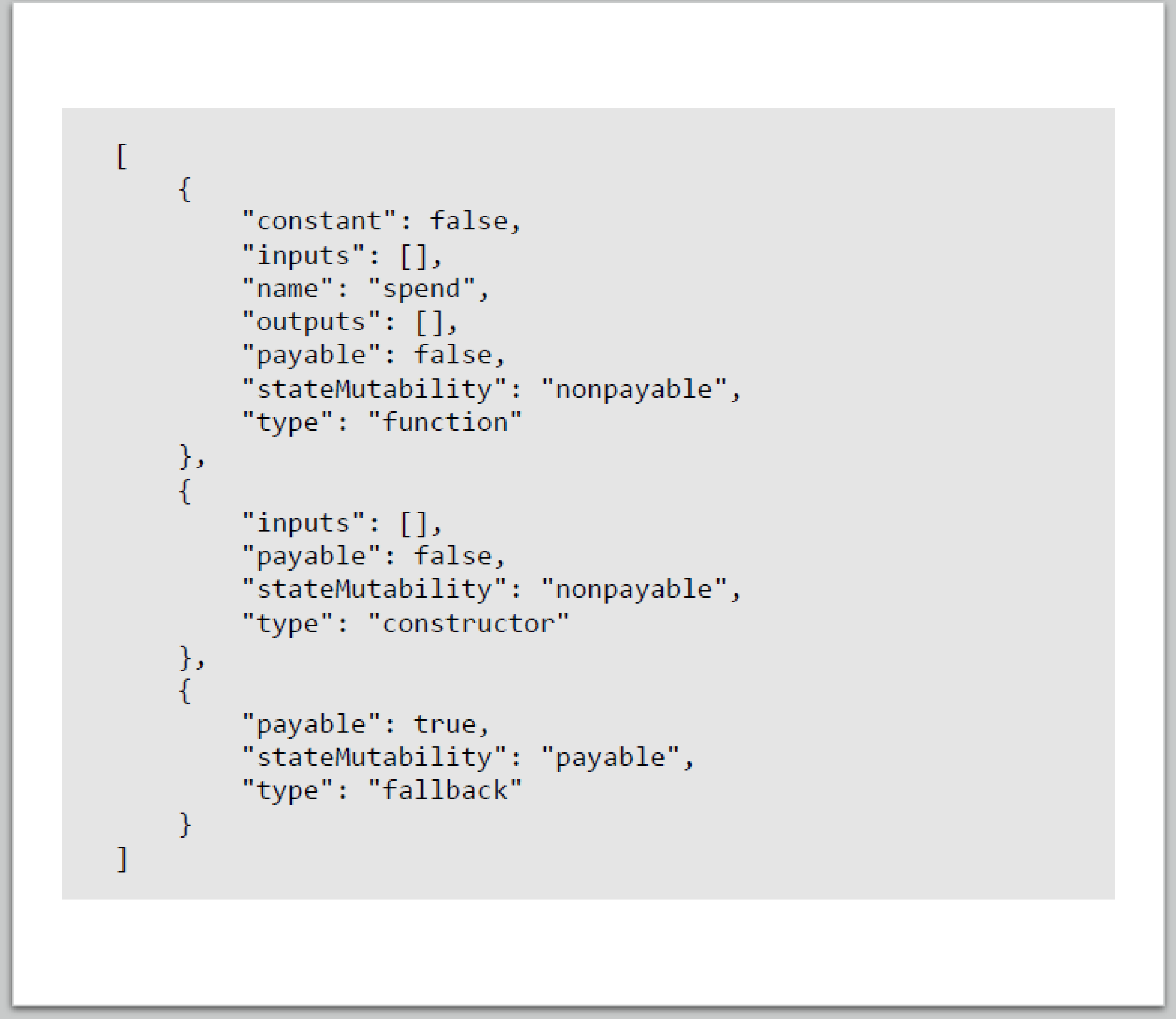
- The **bytecode** is the compiled machine-level code, which is stored on the on the EVM. The bytecode is compiled from the opcode..
- Piggy Bank Bytecode



```
608060405260043610610 01e5760003560e01c806345615bcc14610020575b005b34801561
002c57600080fd5b50610035610037565b005b6000809054906101000a900473ffffffffff
ffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff
163373ffffffffffffffffffffffffffffffffffffffff161461009057600080fd5b670de0
b6b3a76400003073ffffffffffffffffffffffffffffffffffffffff163110156100bc5760
0080fd5b6000809054906101000a900473ffffffffffffffffffffffffffffffffffffffff
1673ffffffffffffffffffffffffffffffffffffffff166108fc3073ffffffffffffffffff
ffffffffffffffffffffff16319081150290604051600060405180830381858888f1935050
505015801561013a573d6000803e3d6000fd5b5056fea265627a7a7231582090e8449f8876
c53a9a1c0ea79b65f7326cf802b7cc2dc9805c774098b8879f2b64736f6c634300050b0032
```

Deploying Smart Contracts

- The **Application Binary Interface (ABI)** specifies the interface for interacting with the smart contract,
- It exposes important information such as definitions of functions and parameters.
- The ABI is the API to the smart contract
- Piggy Bank ABI



```
[
    {
        "constant": false,
        "inputs": [],
        "name": "spend",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
    },
    {
        "inputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "constructor"
    },
    {
        "payable": true,
        "stateMutability": "payable",
        "type": "fallback"
    }
]
```

Deploying Smart Contracts

The following tools can be used to deploy smart contracts into a Blockchain

i. Geth – Ethereum Console

ii. Truffle – Ethereum IDE

iii. Remix – Ethereum IDE

Activity 1: Smart Contract Deployment Using Geth

Deploy Piggy Bank Smart Contract Using Geth

Activity 2: Smart Contract Development using Remix

Activity 3: Testing Smart Contracts

Using Ganache

Using Testnets