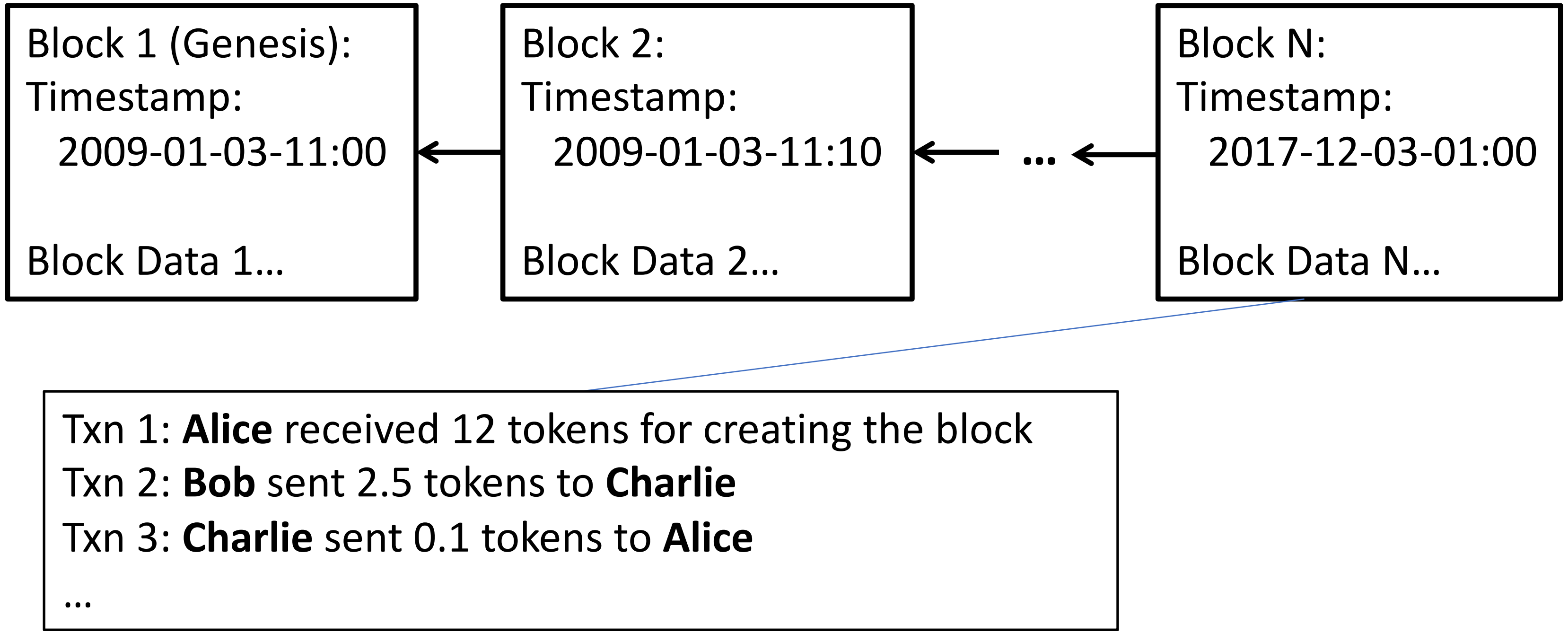
Smart Contracts & Ethereum

Fan Long

University of Toronto

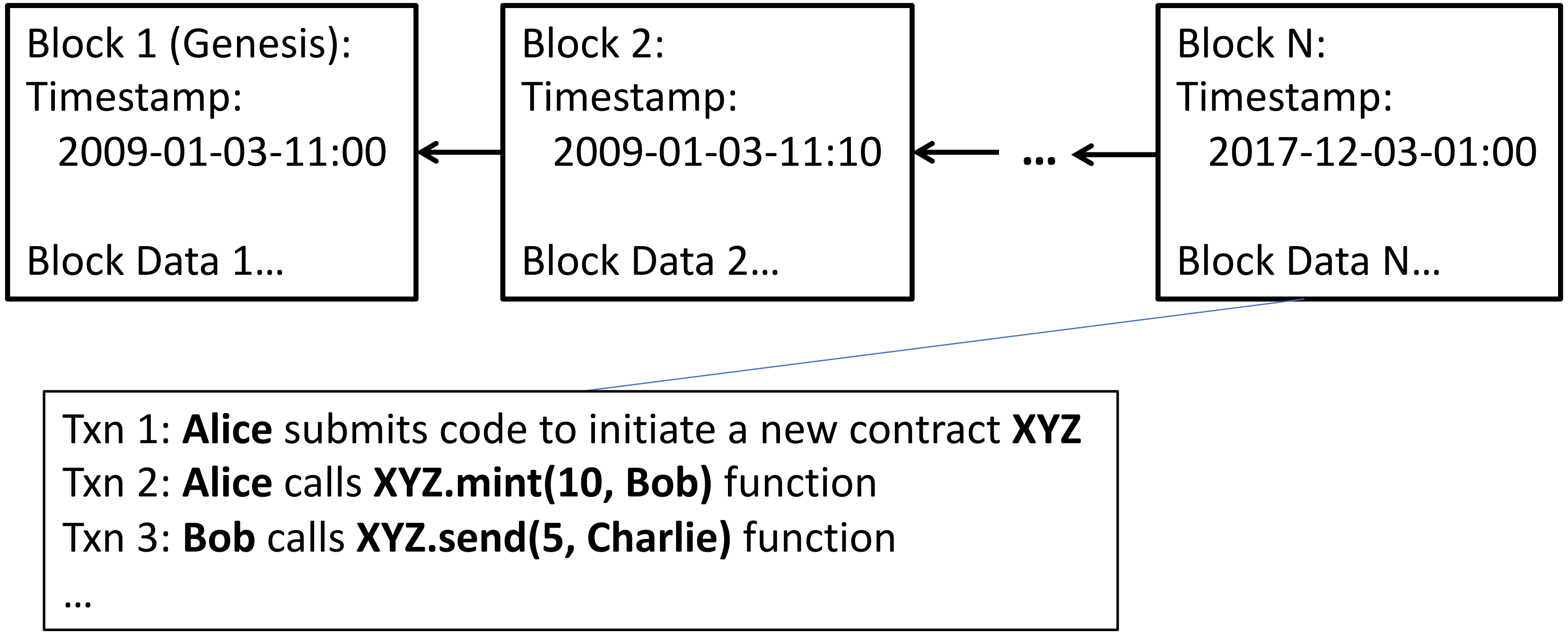
Cryptocurrency: Blockchain Storing Transactions

- The whole transaction history maintained by a network



Blockchain with Smart Contracts

- People can define **Turing-complete customized** transactions



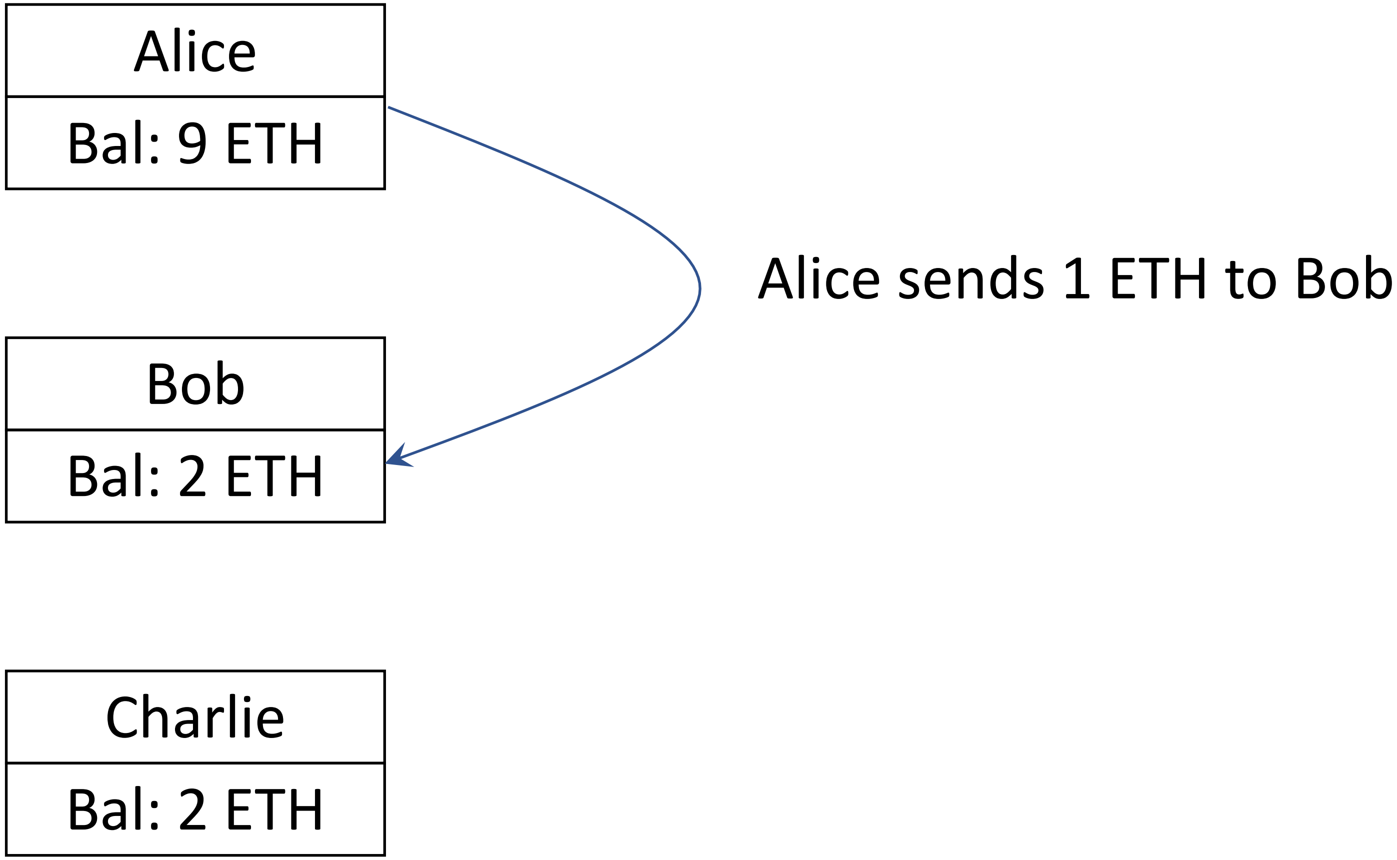
Ethereum Basic Idea

Alice
Bal: 10 ETH

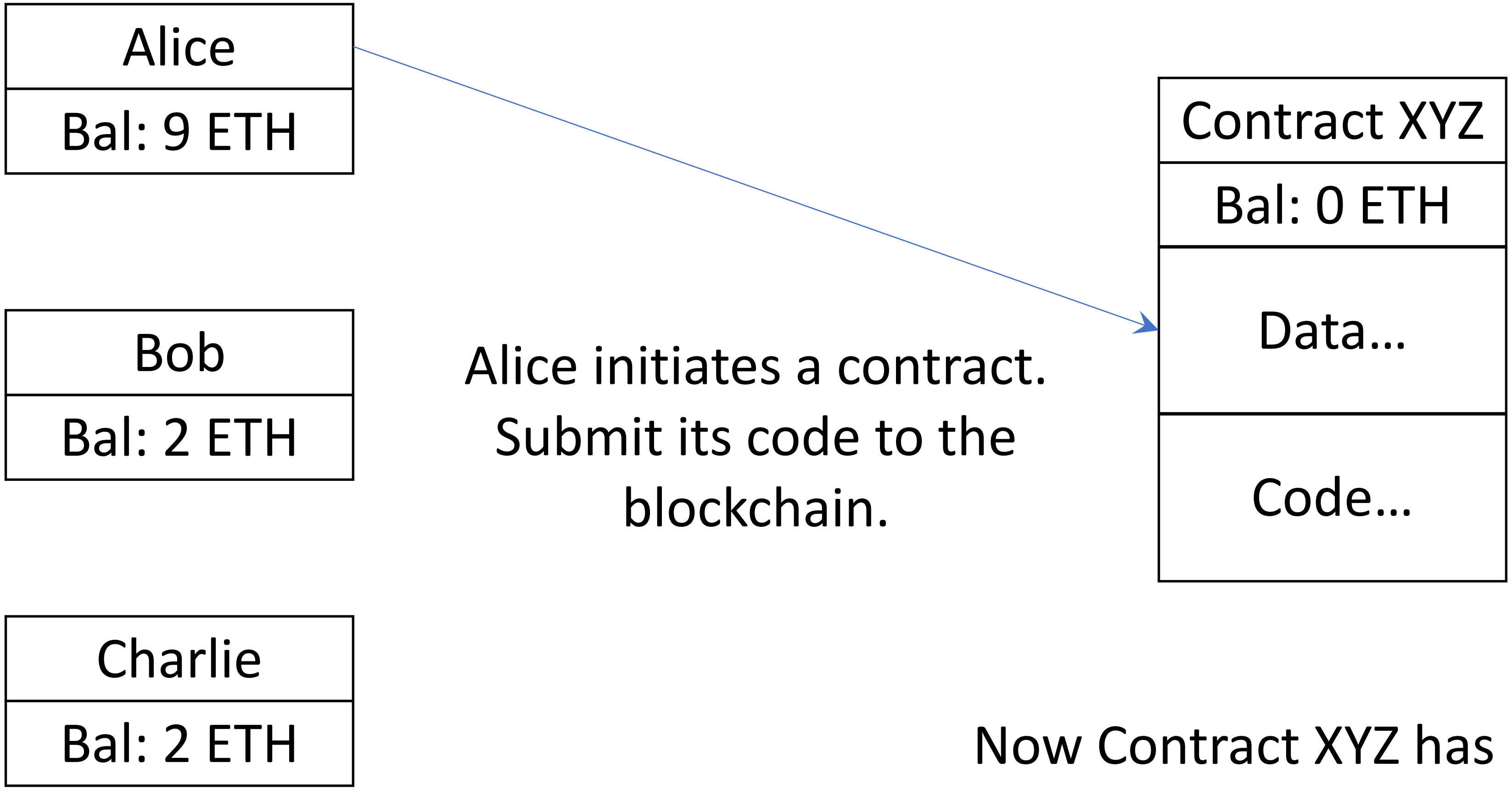
Bob
Bal: 1 ETH

Charlie
Bal: 2 ETH

Ethereum Basic Idea



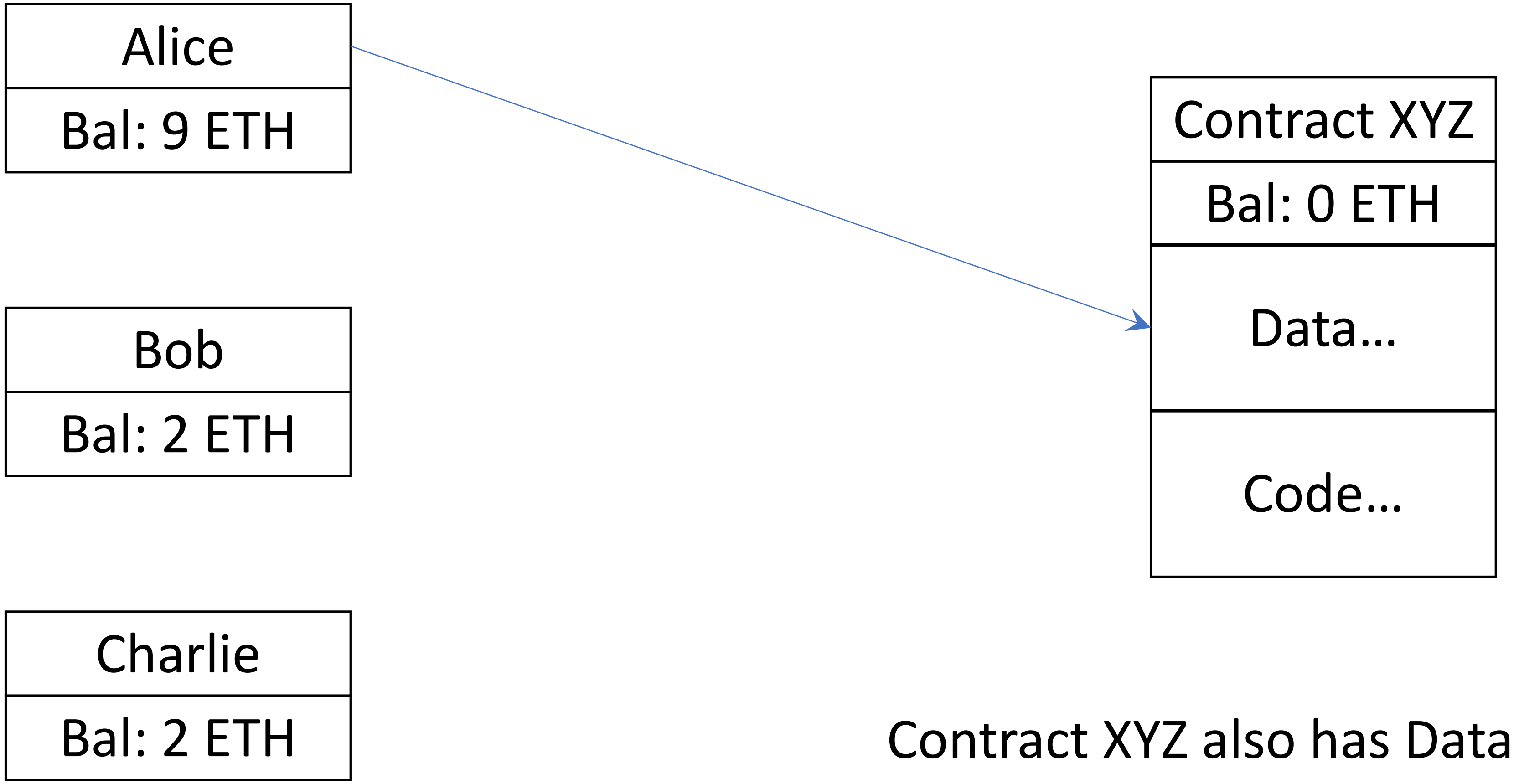
Ethereum Basic Idea



Alice initiates a contract. Submit its code to the blockchain.

Now Contract XYZ has his own address and balance just like other normal accounts.

Ethereum Basic Idea



Contract XYZ also has Data section to maintain its state used by its code.

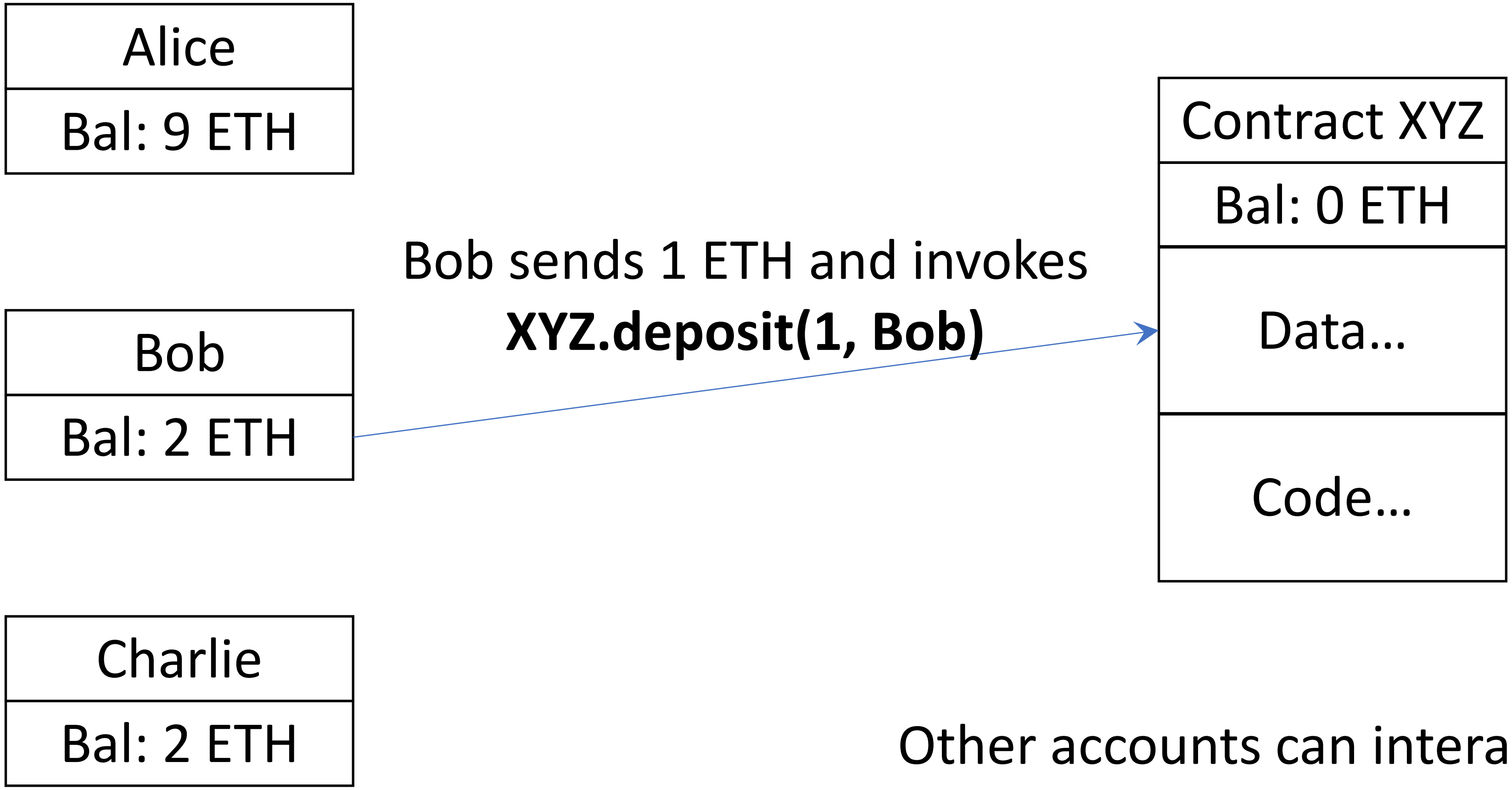
Why Create Smart Contracts

- Encode arbitrarily complicated transaction rules
  - Future contracts
  - Securities
  - Insurance

- The uploaded code will be executed faithfully by the blockchain

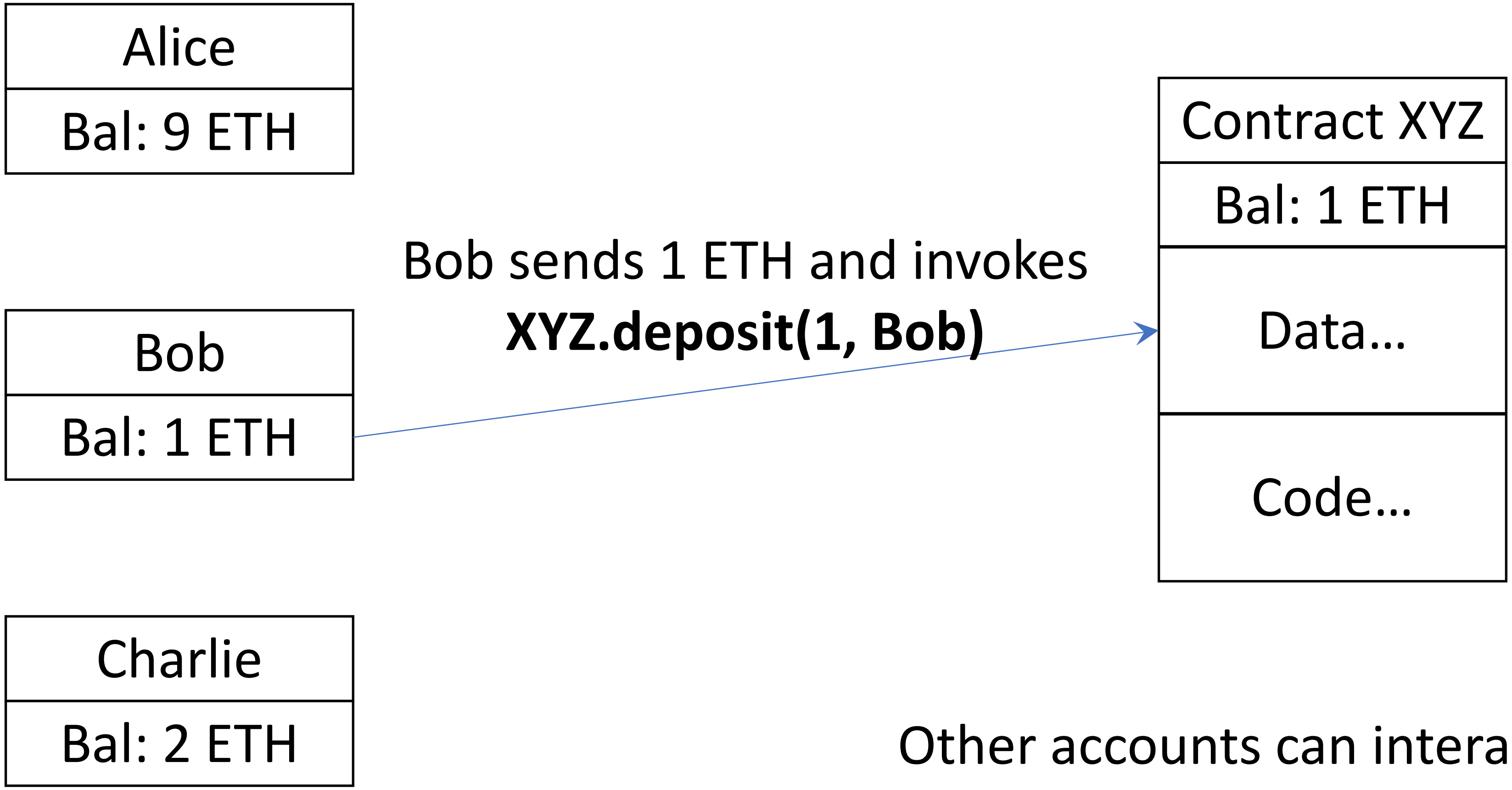
- No party can cheat in the contract
  - Unless one can launch double-spending attack!

Smart Contract Example



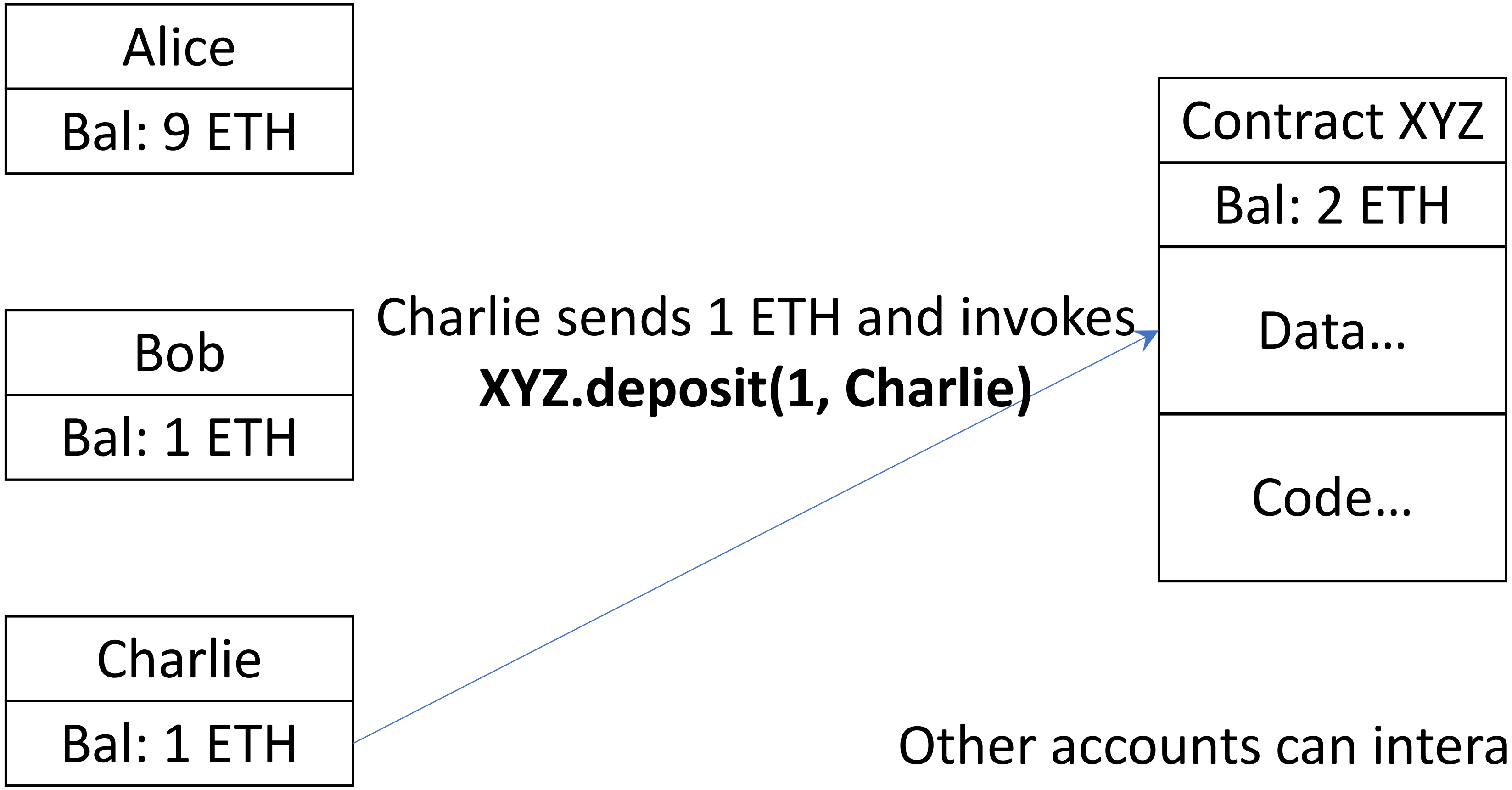
Other accounts can interact with the contract by calling its pre-defined functions

Smart Contract Example



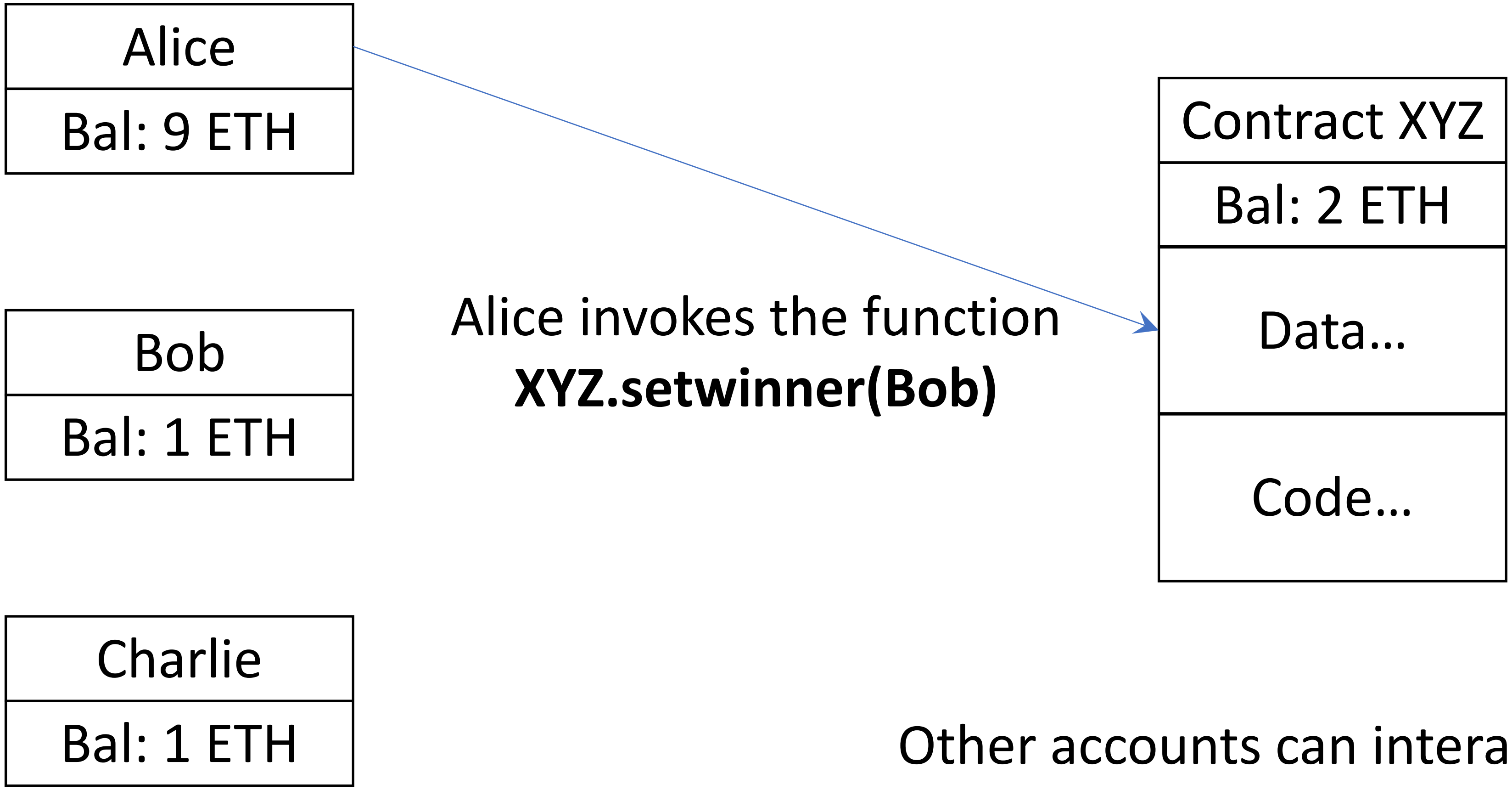
Other accounts can interact with the contract by calling its pre-defined functions

Smart Contract Example



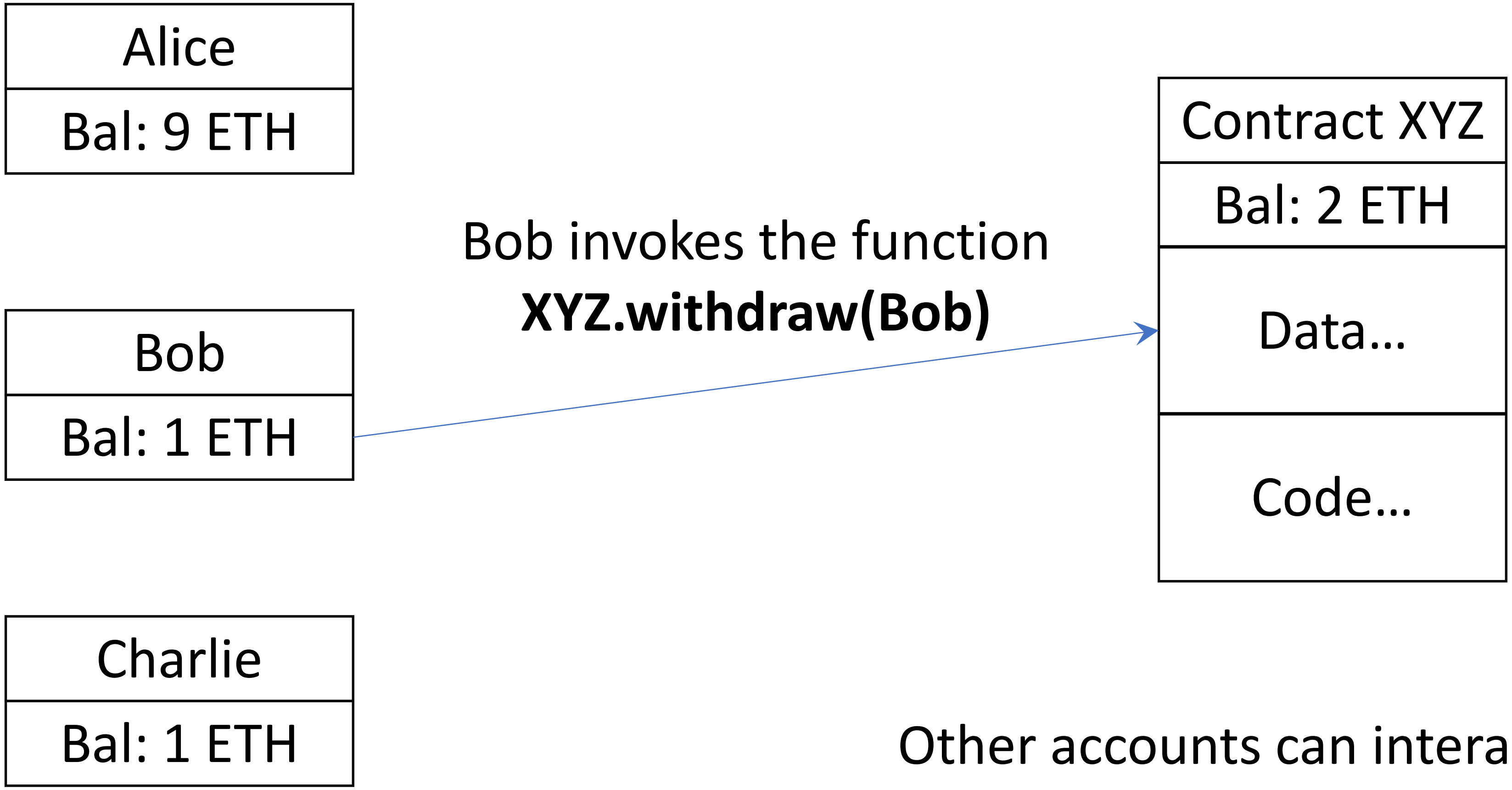
Other accounts can interact with the contract by calling its pre-defined functions

Smart Contract Example



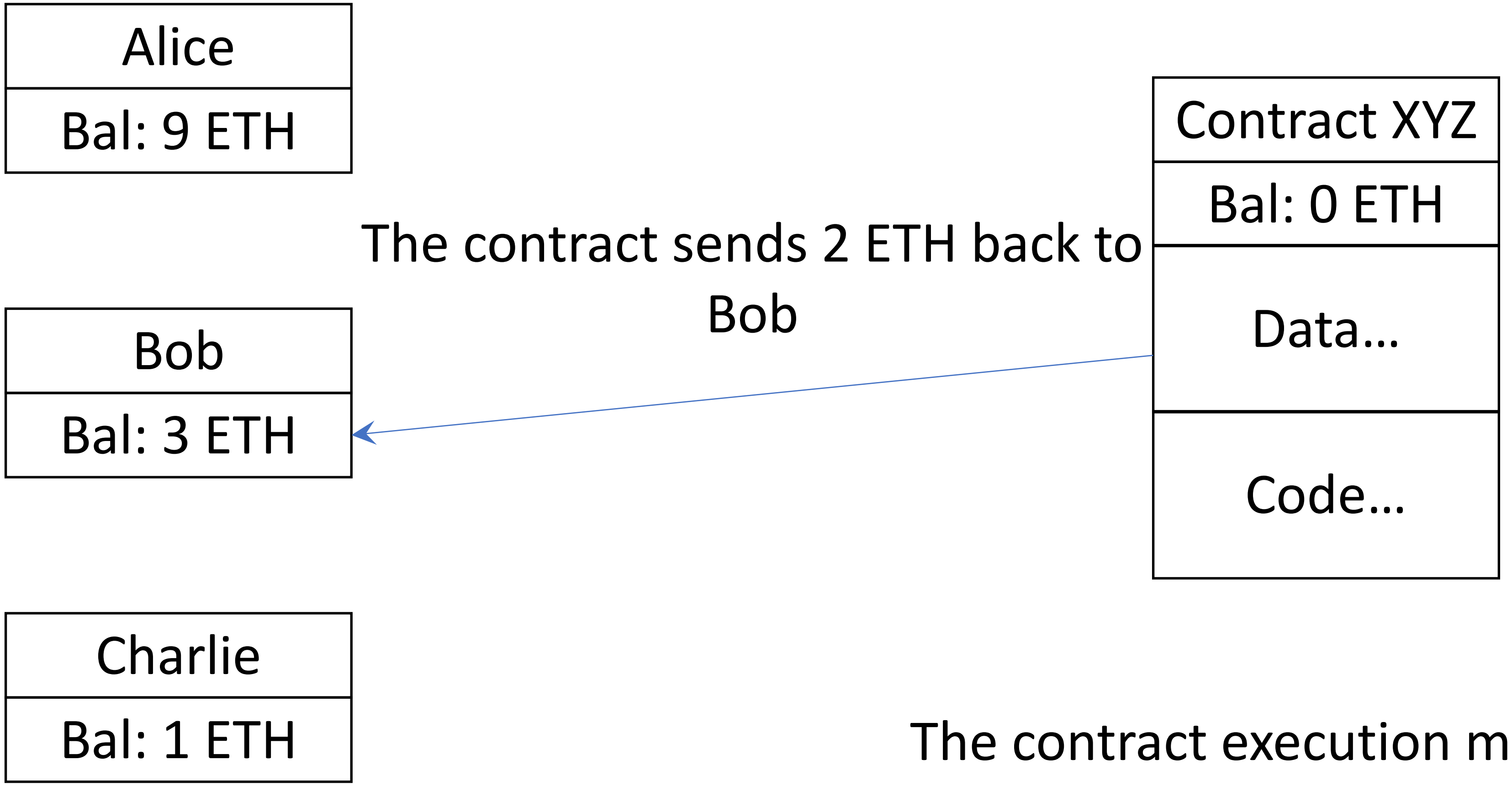
Other accounts can interact with the contract by calling its pre-defined functions

Smart Contract Example



Other accounts can interact with the contract by calling its pre-defined functions

Smart Contract Example



The contract execution may lead to actions like sending its own funds out

Bitcoin UTXO v.s. Ethereum Account State

Bitcoin:

Bob owns private keys to set of UTXOs

5 BTC -> Alice

3 BTC -> Alice

2 BTC -> Alice

Ethereum:

Alice owns private keys to an account

address: “123abc…”

balance: 10 ETH

Contract XYZ account state:

address: “3d41xsy…”

balance: 3 ETH

data: {x -> 0, y -> 1, …}

code: c := a + b…

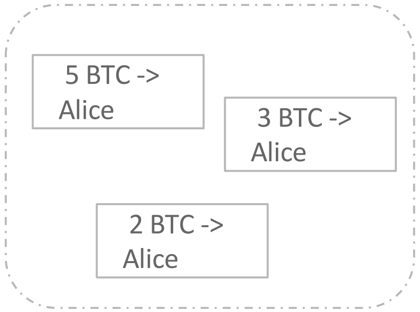
Accounts vs. UTXOs

Recall: A Bitcoin user's available balance is the sum of unspent transaction outputs for which they own the private keys to the output addresses.

Instead Ethereum uses a different concept, called **Accounts**.

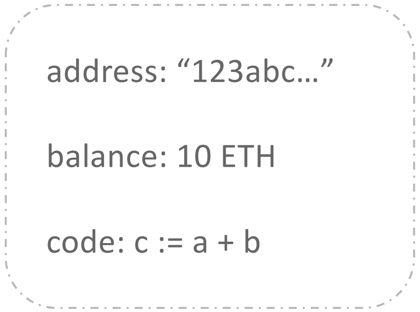
Bitcoin:

Bob owns private keys to set of UTXOs



Ethereum:

Evan owns private keys to an account



Externally Owned Accounts (EOAs):

- Generally owned by some external entity
- Identified by an *address*
- Holds some balance of *ether* (unit of Ethereum currency)
- Can send transactions (transfer ether to other accounts, trigger contract code)

Contract Accounts (Contracts):

- Identified by an *address*
- Has some *ether* balance
- Has associated contract code
- Code execution is triggered by transactions or messages (function calls) received from other contracts
- Contracts have persistent storage

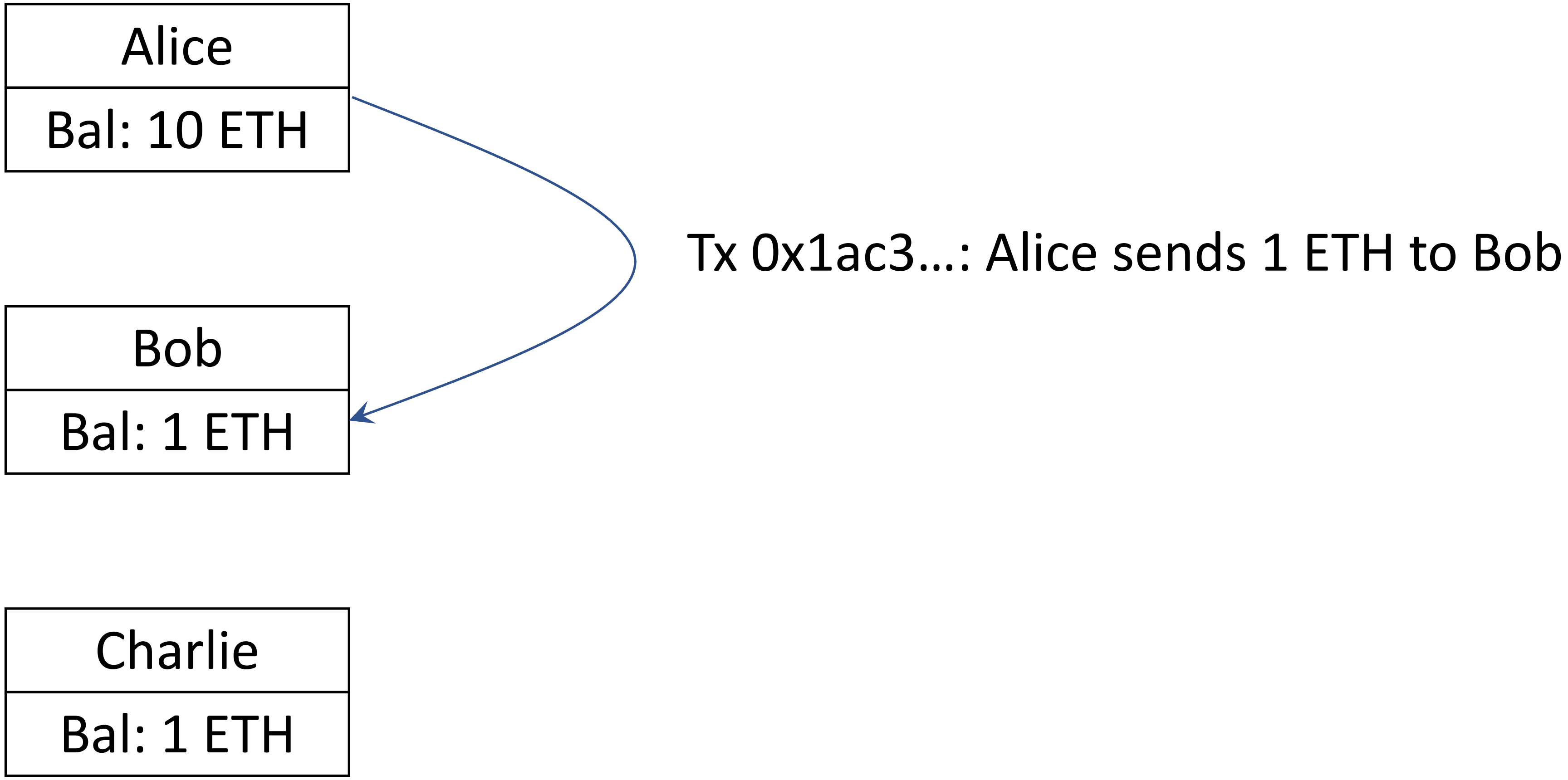
Transactions in Bitcoin v.s. Ethereum

- Transactions in Bitcoin:
  - Multiple inputs with unlocking scripts (signatures inside)
  - Multiple outputs with locking scripts

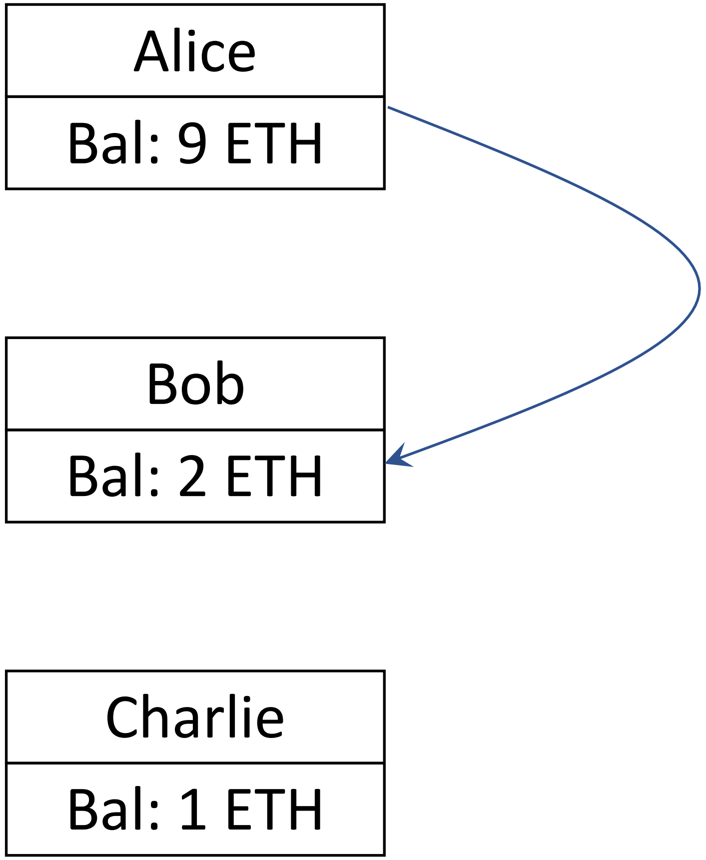
- Transactions in Ethereum:
  - One sender, one receiver, and the amount to send
  - Signature of the sender
  - The function to invoke if the receiver is a contract address

Why use account model instead of UTXO model?

Replay Attack on Account Model



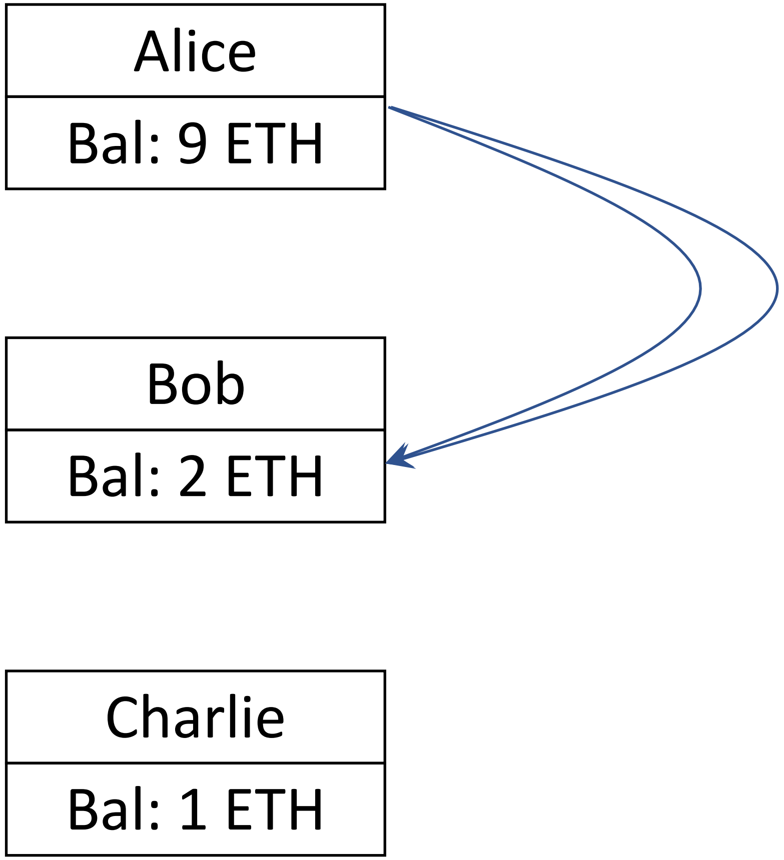
Replay Attack on Account Model



Tx 0x1ac3…: Alice sends 1 ETH to Bob

Bob secretly records the signed transaction

Replay Attack on Account Model

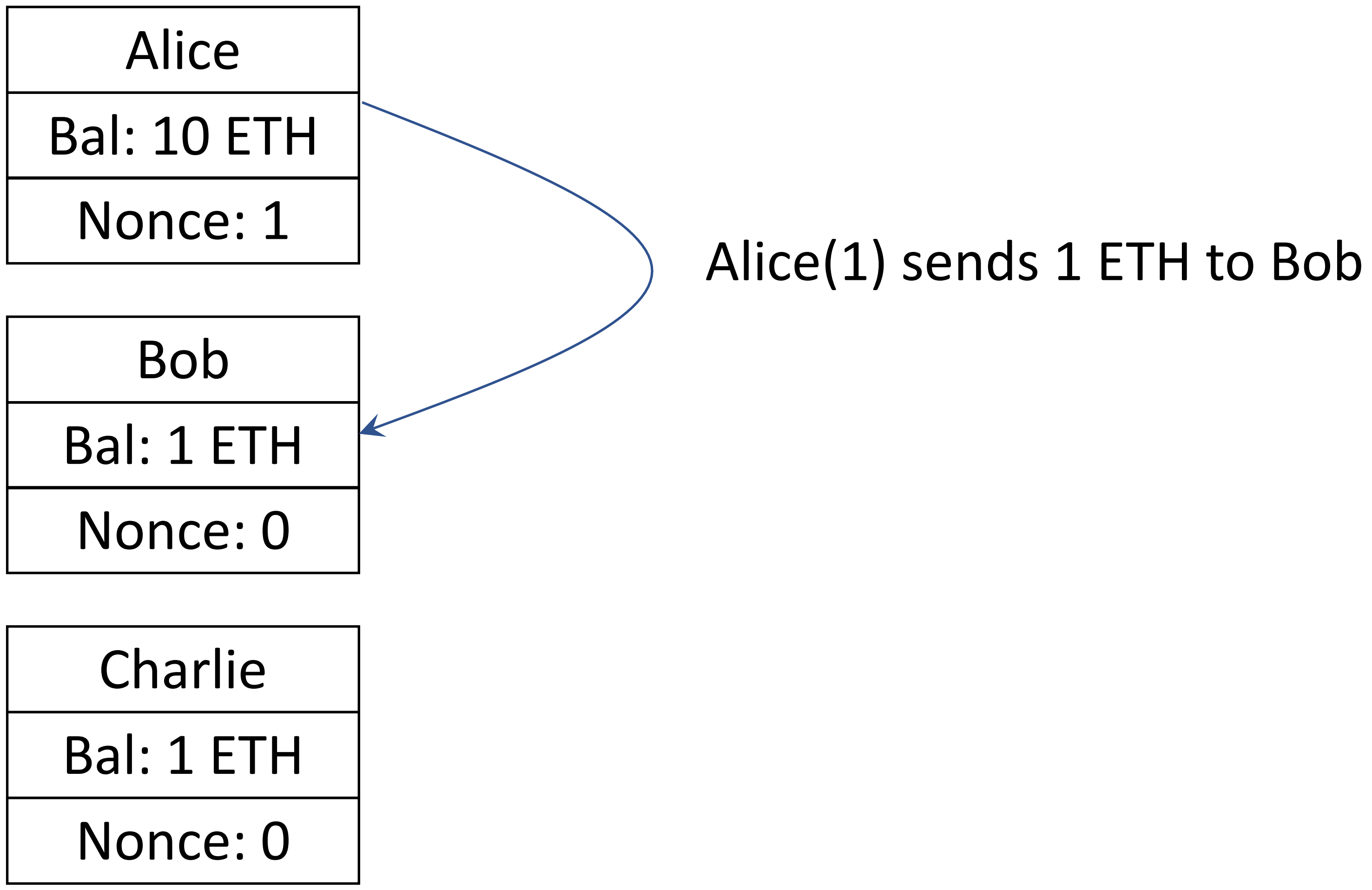


Tx 0x1ac3…: Alice sends 1 ETH to Bob

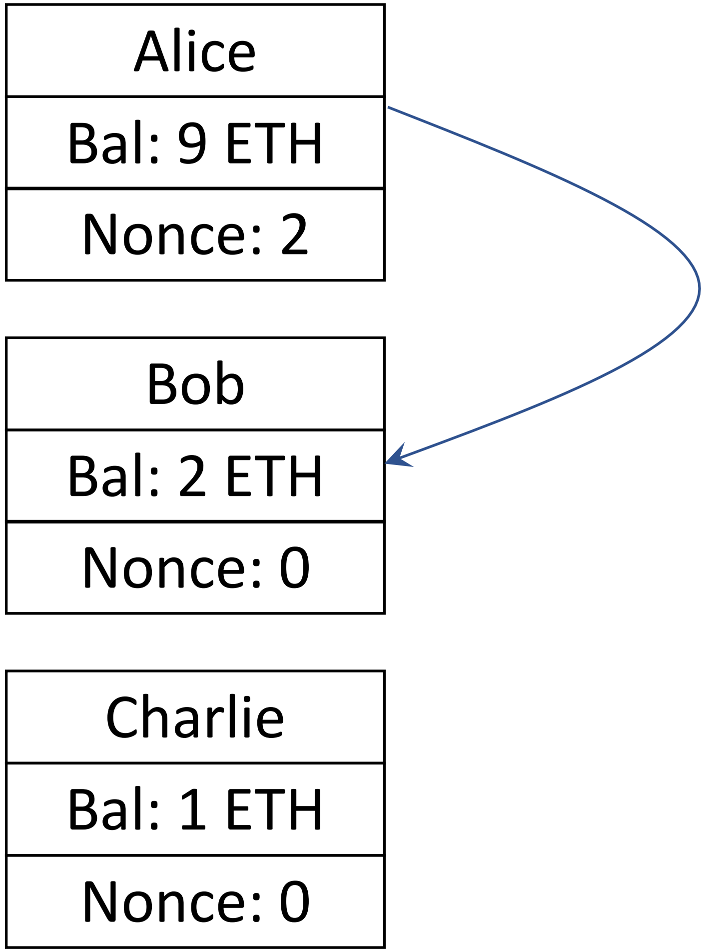
Bob then rebroadcast this transaction after a while

Solution? Remember all past transactions?

Account Model with Nonce



Account Model with Nonce



Alice(1) sends 1 ETH to Bob

Rejected!

A Smart Contract Example in Solidity

Example: Smart Contract in Solidity

```
contract PhilipToken {
    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;




}
```

Example: Smart Contract in Solidity

```
contract PhilipToken {
    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;
    /* Initializes contract with initial supply tokens to the creator of the contrac
    function PhilipToken(uint256 initialSupply)
    {
        balanceOf[msg.sender] = initialSupply; // Give the creator all initial token





}
```

Constructor function is called when the contract is created

Example: Smart Contract in Solidity

```
contract PhilipToken {
    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;
    /* Initializes contract with initial supply tokens to the creator of the contrac
    function PhilipToken(uint256 initialSupply)
    {
        balanceOf[msg.sender] = initialSupply; // Give the creator all initial token
    /* Send tokens to a recipient address */
    function transfer(address to, uint256 value)
    {
        if (balanceOf[msg.sender] < value) throw;          // Check if the sender ha
        if (balanceOf[to] + value < balanceOf[to]) throw;  // Check for overflows
        balanceOf[msg.sender] -= value;                    // Subtract from the send
        balanceOf[to] += value;                            // Add the same to the re
    }
}
```

Constructor function is called when the contract is created

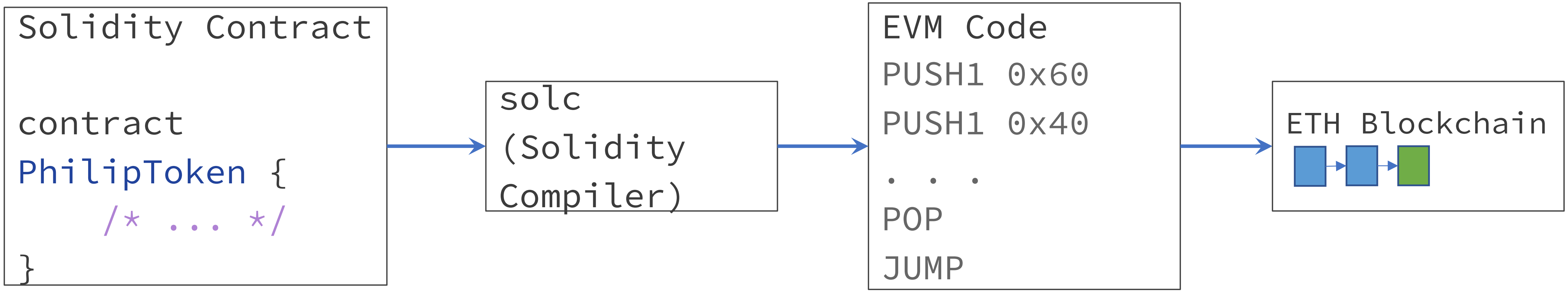
Member functions can be invoked by other users via transactions

Example: Smart Contract in Solidity

```
contract PhilipToken {
    /* Maps account addresses to token balances */
    mapping (address => uint256) public balanceOf;
    /* Initializes contract with initial supply tokens to the creator of the contrac
    function PhilipToken(uint256 initialSupply)
    {
        balanceOf[msg.sender] = initialSupply; // Give the creator all initial token
    /* Send tokens to a recipient address */
    function transfer(address to, uint256 value)
    {
        if (balanceOf[msg.sender] < value) throw;         // Check if the sender ha
        if (balanceOf[to] + value < balanceOf[to]) throw;  // Check for overflows
        balanceOf[msg.sender] -= value;                   // Subtract from the send
        balanceOf[to] += value;                           // Add the same to the re
    }
}
```

PhilipToken contract implements a kind of fixed supply tokens that can be freely transferred

How Ethereum Process Smart Contracts



Why have an intermediate assembly language like EVM on the blockchain?

EVM Design

- Separate language design from the blockchain design
- EVM is simpler than Solidity
  - Easier to implement an interpreter in Ethereum correctly
- It can enable the support for other languages
  - Only need to write a compiler to EVM
  - Check Vyper language for Ethereum

Ethereum Virtual Machine

The Ethereum contract code that actually gets executed on every node is so-called EVM code, a low-level, stack-based byte-code language.

Every Ethereum node runs the EVM as part of its block verification procedure.

EVM as a state transition mechanism:

(block_state, gas, memory, transaction, message, code, stack, pc)

↓

(block_state', gas')

where block_state is the global state containing all accounts and includes balances and long-term storage

EVM Design Goals:

Simplicity: op-codes should be as low-level as possible. The number of op-codes should be minimized.

Determinism: The execution of EVM code should be deterministic; the same input state should always yield the same output state.

Space Efficiency: EVM assembly should be as compact as possible

Specialization: easily handle 20-byte addresses and custom cryptography with 32-byte values, modular arithmetic used in custom cryptography, read block and transaction data, interact with state, etc

Security: it should be easy to come up with a gas cost model for operations that makes the VM non-exploitable

EVM v.s. Bitcoin Script

- Both are stack-based language
- Both have basic arithmetic and cryptographic operations

- EVM unique features:
  - Heap memory for persistent state
  - Function calls
  - Loops

Termination Problem

- Because contracts are Turing complete, it may never terminate

```
function foo()
{
    while (true) {
        /* Loop forever! */
    }
}
```

- When processing a transaction like this, a node will hang forever
- **Solution**: Charge transaction fee as execution goes and put a cap on it

Gas: Transaction Fee in Ethereum

- There is an amount of Gas associated with each EVM opcode
- Sender of a transaction specifies:
  - ***startgas***: The maximum of gas the sender gives to the execution
  - ***gasprice***: The gas unit price the sender is willing to pay
- As the execution of the transaction terminates, count the total amount of consumed gas as ***totalgas***
  - Charge ***totalgas*** * ***gasprice*** as the transaction fee
- If the execution consumes more than ***startgas***
  - The execution stops and reverts back to the original state (no effect)
  - Charge ***startgas*** * ***gasprice*** anyway

Usage Scenario of Smart Contract & Ethereum

Token Systems

- Very easy to implement in Ethereum
- Database with one operation
  - Ensure Alice has enough money and that she initiated the transaction
  - Subtract X from Alice, give X to Bob

Example (from Ethereum white paper):

```
def send(to, value):
    if self.storage[msg.sender] >= value:
        self.storage[msg.sender] = self.storage[msg.sender] - value
        self.storage[to] = self.storage[to] + value
```

Public Registry / Public database

Example: **Namecoin**

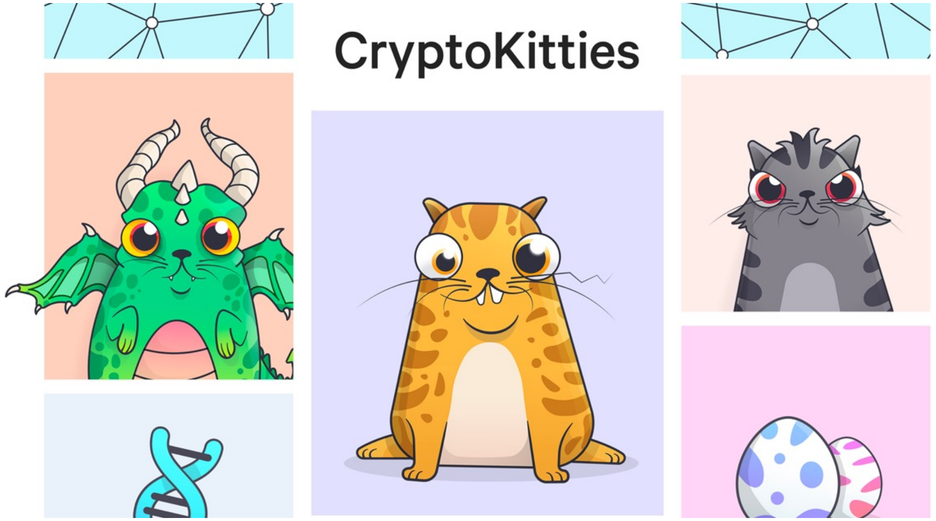
- DNS system
  - Maps domain name to IP address
  - "maxfa.ng" => "69.69.69.69"
- Immutable
- Easy implementation in Ethereum

Example (from Ethereum white paper):

```
def register(name, value):
    if !self.storage[name]:
        self.storage[name] = value
```

Gambling Places

- CryptoKitties
- Femo3D



Decentralized Finance

- Smart contracts that manage digital assets, provide financial service, and generate new derivatives.

- Uniswap: Decentralized digital asset exchanges with automatic market making.
- Compound: Decentralized lending protocol, borrow digital assets with another digital assets as a collateral.
- MakerDAO: Mint decentralized stable coin DAI with digital assets as collaterals.
- Curve: Decentralized exchange protocol for stable coins.

Uniswap V2 Example

- Suppose two type of assets: ETH and USDC, assume 1 ETH = 1500 USDC
- LP provider deposits equivalent value of two assets into a pool, let’s say:
    - 10 Eth + 15000 USDC
- The LP pool maintains a constant of C = X * Y, where X and Y are the amount of ETH and USDC. Here the constant is 150000
- Whenever user tries to buy and sell ETH against the pool, the constant will determine the amount of USDC you receive.
- For example: sell 1 ETH would get (15000 - 150000 / (10 + 1)) = 1363.63
- Assume LP pool charges 0.25% fee, then:
- Sell 1 ETH would get (15000 – 150000 / (10 + (1 * 0.9975))) = 1360.53

Smart Contract Status

- Main usages in Ethereum:
  - Crowdfunding ICO with a token system
  - Decentralized finance
  - Gambling game
- Why only limited usage of smart contracts?
- Limitations:
  - Unable to get real world data. Oracles?
  - Scalability
  - Security problems in smart contracts

Exploitations of Smart Contract

- Programming errors are critical in smart contracts!
- The DAO re-entrance attack
  - Caused the fork of ETH and ETC
- Integer overflow errors in token contracts
- Backdoors