A gentle introduction to Digital Currencies and Smart Contracts

Tom Van Cutsem

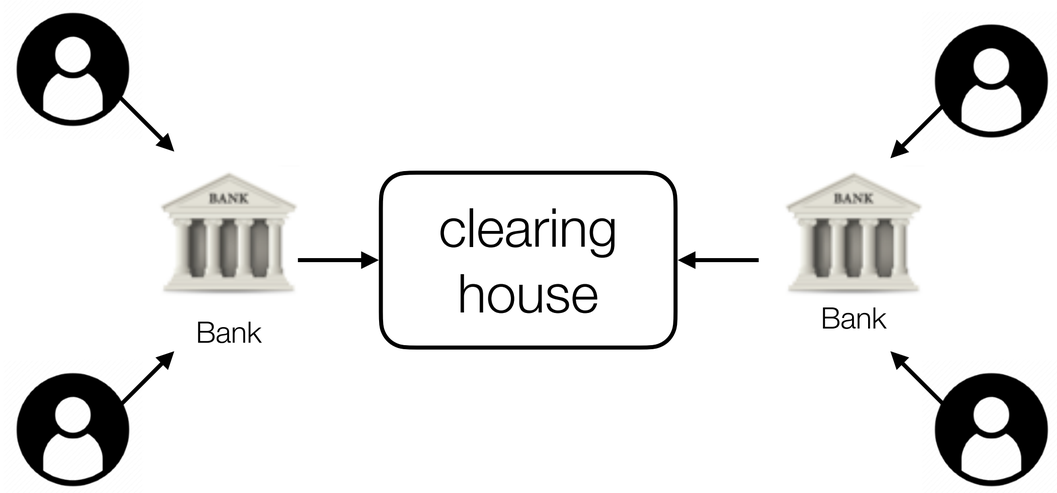
IFIP WG2.16 Programming Language Design Meeting
January 2021

 tvcutsem.github.io  be.linkedin.com/in/tomvc  github.com/tvcutsem  twitter.com/tvcutsem

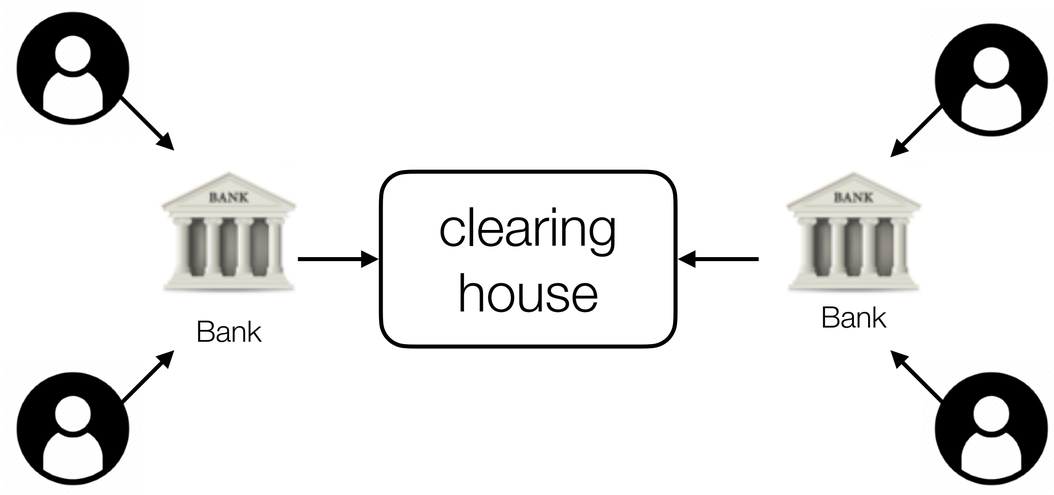
“**Money** is the most universal and most efficient **system of** mutual **trust** ever devised.”

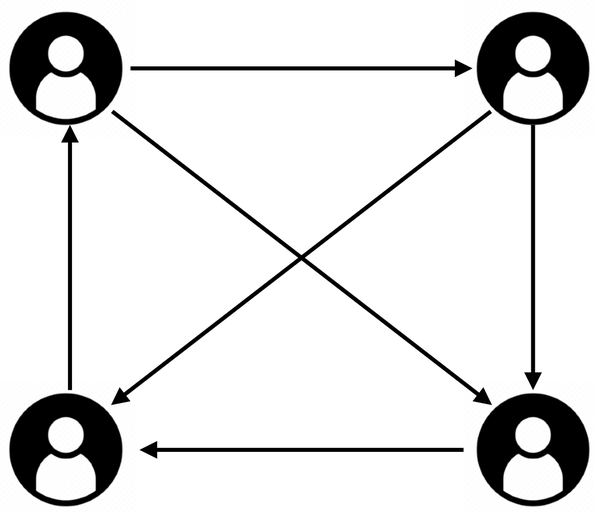
- Yuval Noah Harari
“Sapiens: a brief history of humankind”

In most forms of money, trust is centralised

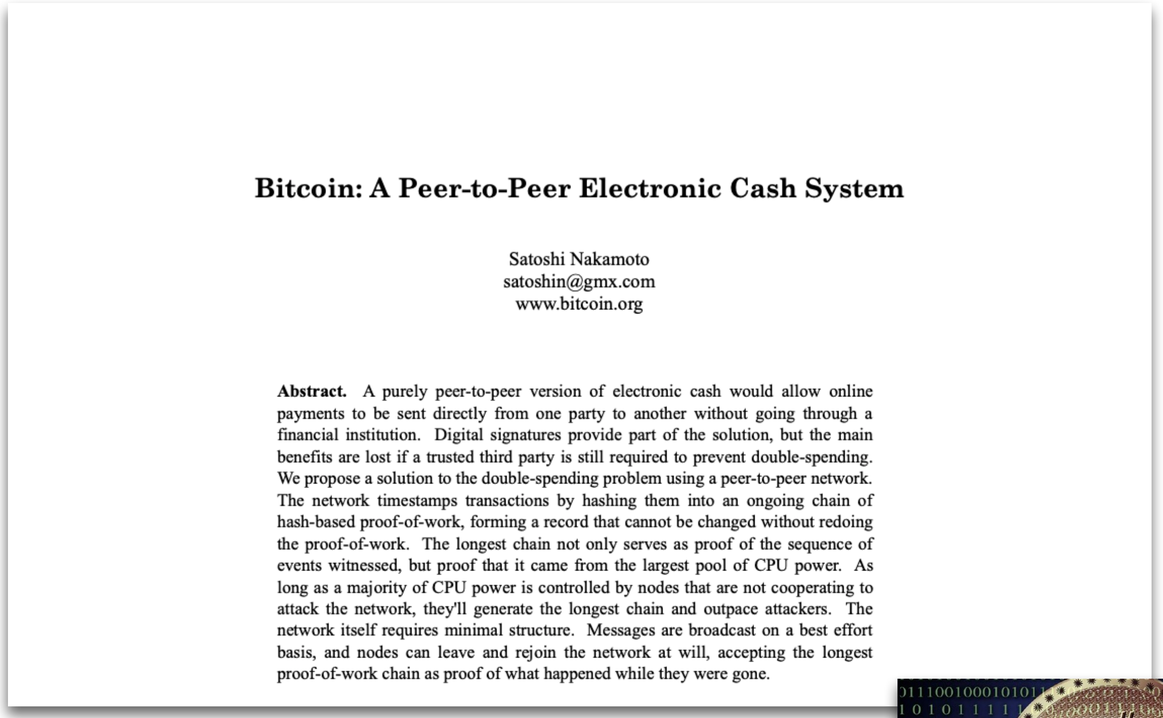


Can trust be decentralised?





Can trust be decentralised?

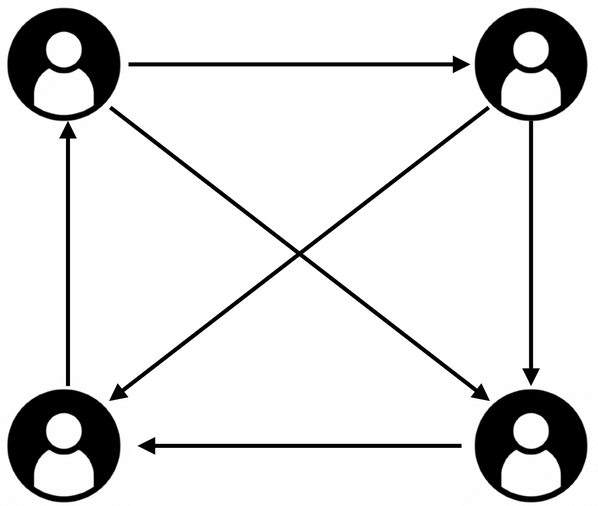


Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

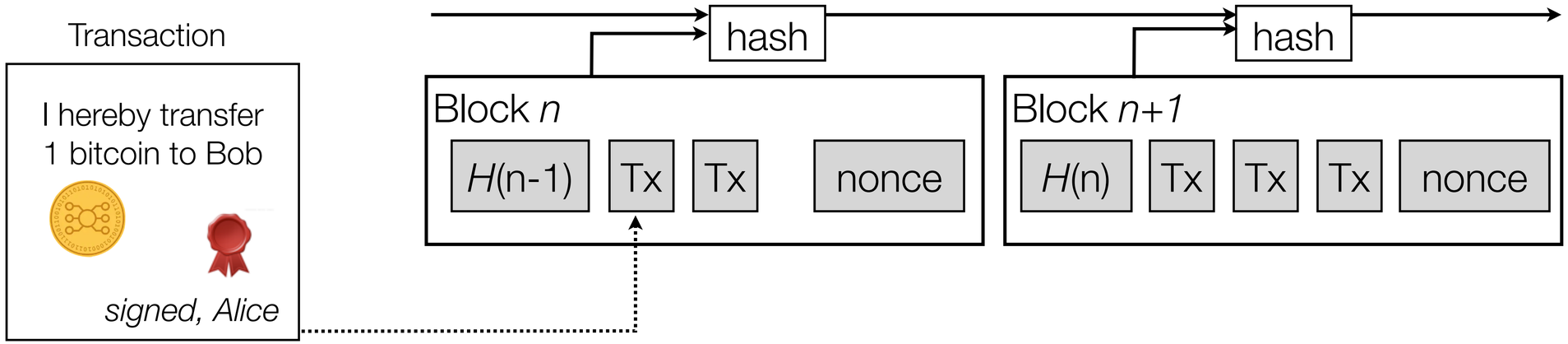
Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Satoshi Nakamoto
Bitcoin (2009)



Key technical innovation: blockchain

- Replace central clearing house by a **public, replicated, tamper-resistant ledger**
- Validators group transactions in “blocks”, “chained” together into **a linear sequence** using cryptographic hashes, secured using “proof of work”



What other applications do blockchains enable?

Ethereum White Paper

A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM

By Vitalik Buterin

When Satoshi Nakamoto first set the Bitcoin blockchain into motion in January 2009, he was simultaneously introducing two radical and untested concepts. The first is the "bitcoin", a decentralized peer-to-peer online currency that maintains a value without any backing, intrinsic value or central issuer. So far, the "bitcoin" as a currency unit has taken up the bulk of the public attention, both in terms of the political aspects of a currency without a central bank and its extreme upward and downward volatility in price. However, there is also another, equally important, part to Satoshi's grand experiment: the concept of a proof of work-based blockchain to allow for public agreement on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if one entity has 50 BTC, and simultaneously sends the same 50 BTC to A and to B, only the transaction that gets confirmed first will process. There is no intrinsic way of determining from two transactions which came earlier, and for decades this stymied the development of decentralized digital currency. Satoshi's blockchain was the first credible decentralized solution. And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the blockchain concept can be used for more than just money.

Vitalik Buterin
Ethereum (2014)

What other applications do blockchains enable?

Ethereum White Paper

A NEXT GENERATION SMART CONTRACT & DECENTRALIZED APPLICATION PLATFORM

By Vitalik Buterin

When Satoshi Nakamoto first set the Bitcoin blockchain into motion in January 2009, he was simultaneously introducing two radical and untested concepts. The first is the "bitcoin", a decentralized peer-to-peer online currency that maintains a value without any backing, intrinsic value or central issuer. So far, the "bitcoin" as a currency unit has taken up the bulk of the public attention, both in terms of the political aspects of a currency without a central bank and its extreme upward and downward volatility in price. However, there is also another, equally important, part to Satoshi's grand experiment: the concept of a proof of work-based blockchain to allow for public agreement on the order of transactions. Bitcoin as an application can be described as a first-to-file system: if one entity has 50 BTC, and simultaneously sends the same 50 BTC to A and to B, only the transaction that gets confirmed first will process. There is no intrinsic way of determining from two transactions which came earlier, and for decades this stymied the development of decentralized digital currency. Satoshi's blockchain was the first credible decentralized solution. And now, attention is rapidly starting to shift toward this second part of Bitcoin's technology, and how the blockchain concept can be used for more than just money.

Vitalik Buterin
Ethereum (2014)

Nick Szabo
Smart Contracts (1996)

What is a smart contract?

A software program that automatically moves digital assets according to arbitrary pre-specified rules

(source: V. Buterin, Ethereum White Paper, 2014)

Smart contracts: basic principle

An automaton that can trade physical assets

1. insert coins

2. dispense drink

Smart contracts: basic principle

An automaton that can trade digital assets

```
/* Allow another contract to spend some tokens in your behalf */
function approve(address _spender, uint256 _value)
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    return true;
}

/* Approve and then comunicate the approved contract in a single tx */
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
        return true;
    }
}

/* A contract attempts to get the coins */
function tranferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    if (balanceOf[_from] < _value) throw;                 // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Check for overflows
    if (_value > allowance[_from][msg.sender]) throw;   // Check allowance
    balanceOf[_from] -= _value;                           // Subtract from the sender
    balanceOf[_to] += _value;                             // Add the same to the recipient
    allowance[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
    return true;
}

/* This unnamed function is called whenever someone tries to send ether to it */
function () {
    throw;     // Prevents accidental sending of ether
```

code

1. insert digital coins

2. dispense other digital assets or electronic rights

But who should we trust to faithfully execute the code?

An automaton that can trade digital assets

```
/* Allow another contract to spend some tokens in your behalf */
function approve(address _spender, uint256 _value)
    returns (bool success) {
    allowance[msg.sender][_spender] = _value;
    return true;
}

/* Approve and then comunicate the approved contract in a single tx */
function approveAndCall(address _spender, uint256 _value, bytes _extraData)
    returns (bool success) {
    tokenRecipient spender = tokenRecipient(_spender);
    if (approve(_spender, _value)) {
        spender.receiveApproval(msg.sender, _value, this, _extraData);
        return true;
    }
}

/* A contract attempts to get the coins */
function transferFrom(address _from, address _to, uint256 _value) returns (bool success) {
    if (balanceOf[_from] < _value) throw;                 // Check if the sender has enough
    if (balanceOf[_to] + _value < balanceOf[_to]) throw;  // Check for overflows
    if (_value > allowance[_from][msg.sender]) throw;   // Check allowance
    balanceOf[_from] -= _value;                           // Subtract from the sender
    balanceOf[_to] += _value;                             // Add the same to the recipient
    allowance[_from][msg.sender] -= _value;
    Transfer(_from, _to, _value);
    return true;
}

/* This unnamed function is called whenever someone tries to send ether to it */
function () {
    throw;     // Prevents accidental sending of ether
```

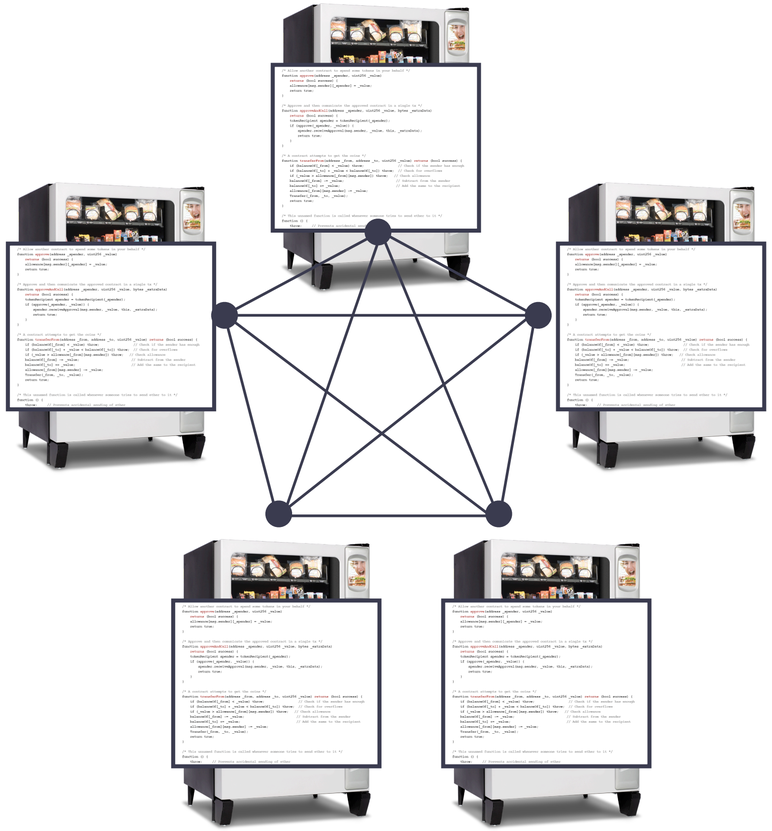
code

1. insert digital coins

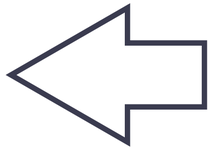
2. dispense other digital assets or electronic rights

Delegate trust to a decentralised network

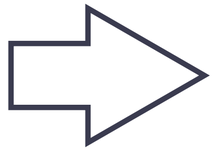
A replicated automaton that can trade digital assets



replicated code

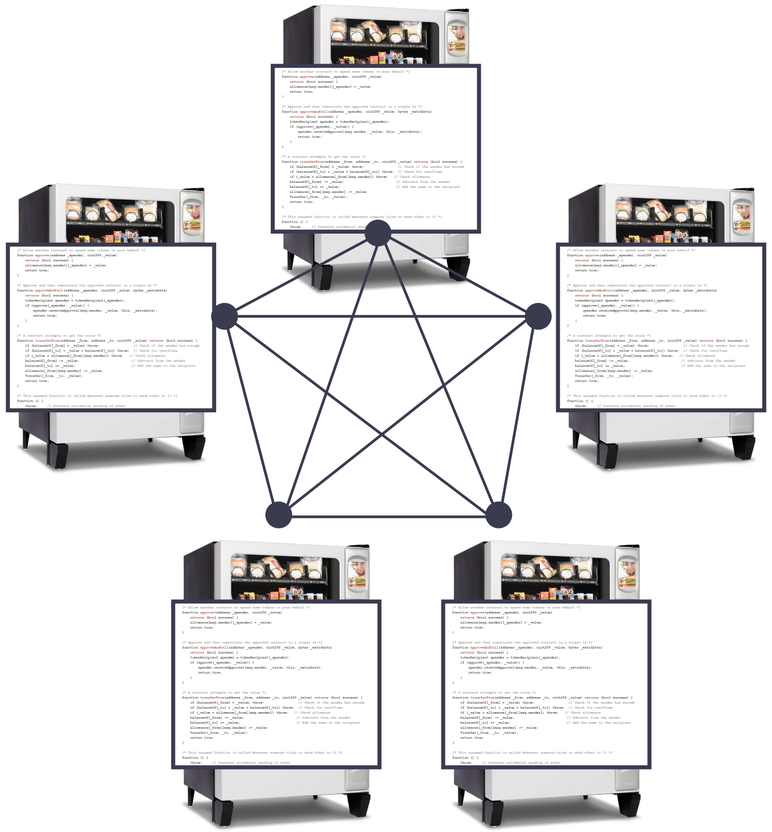


1. insert digital coins



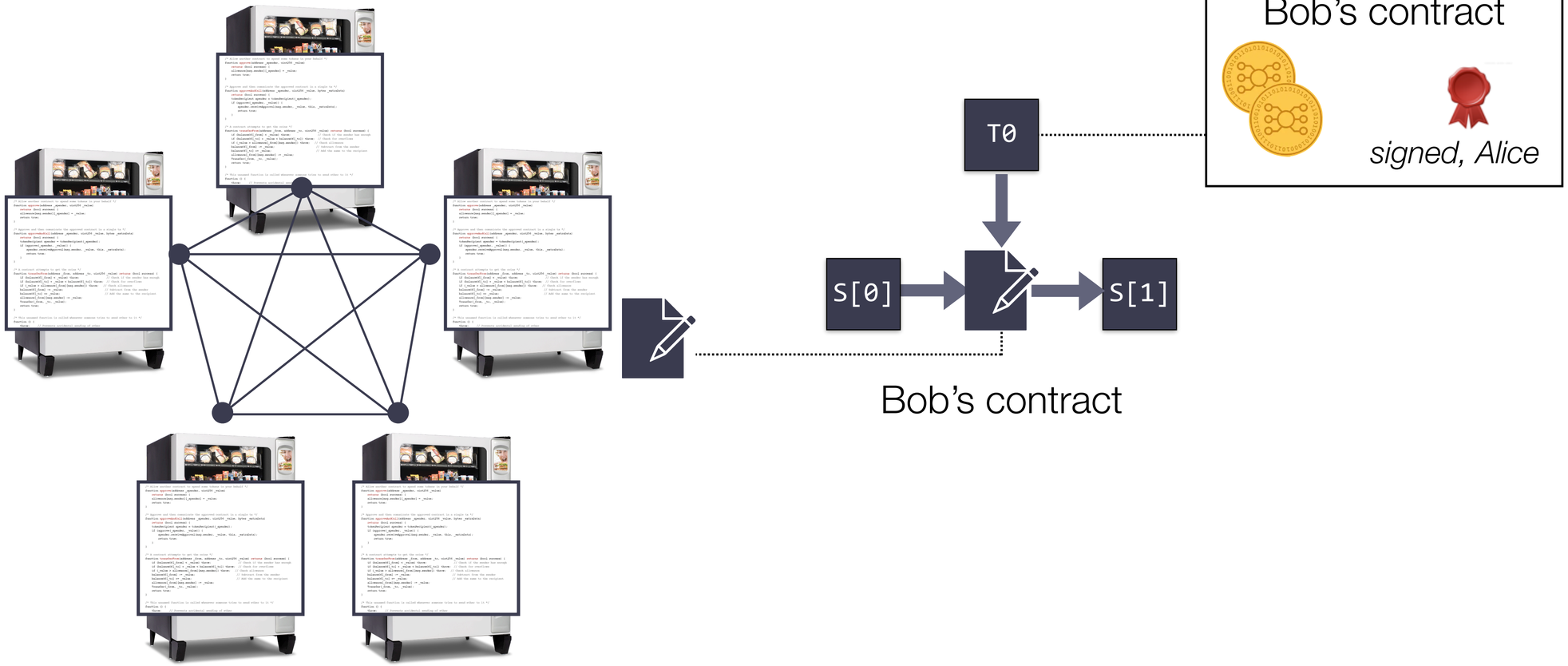
2. dispense other digital assets or electronic rights

A decentralized “world computer” to execute smart contracts

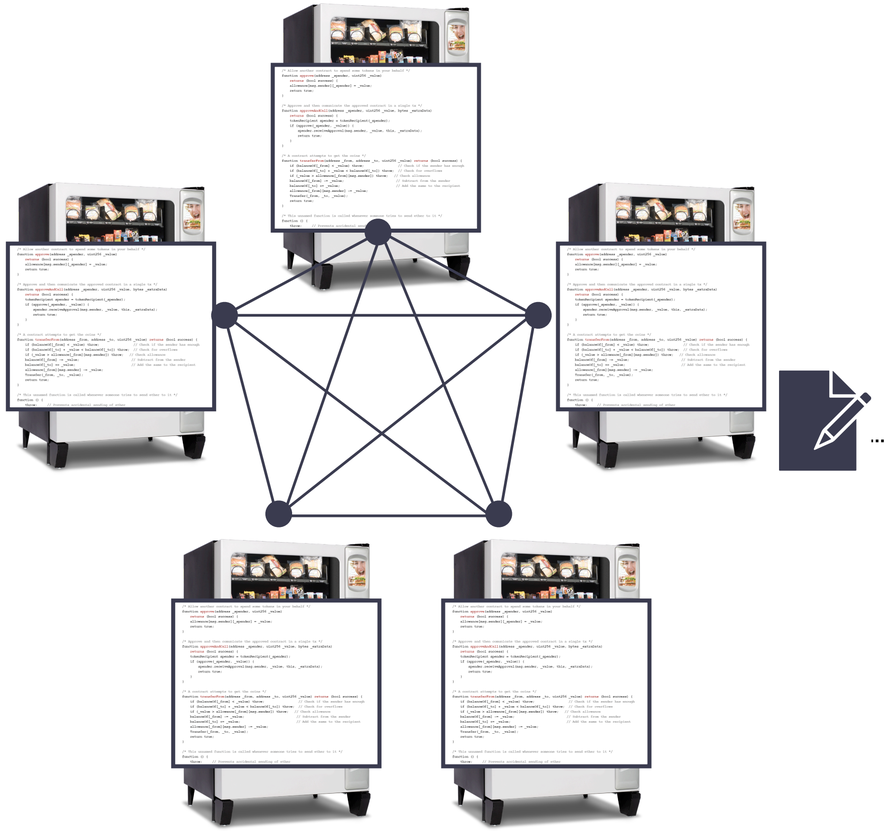


ethereum

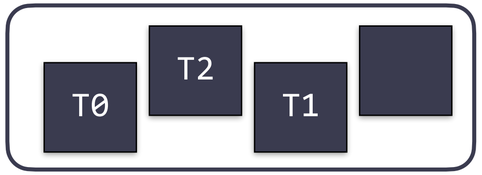
Each transaction updates the virtual computer's replicated state



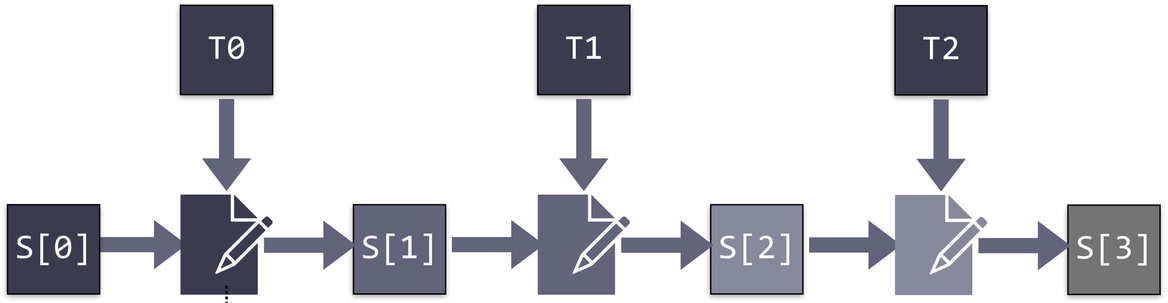
Incoming transactions are sequenced into blocks



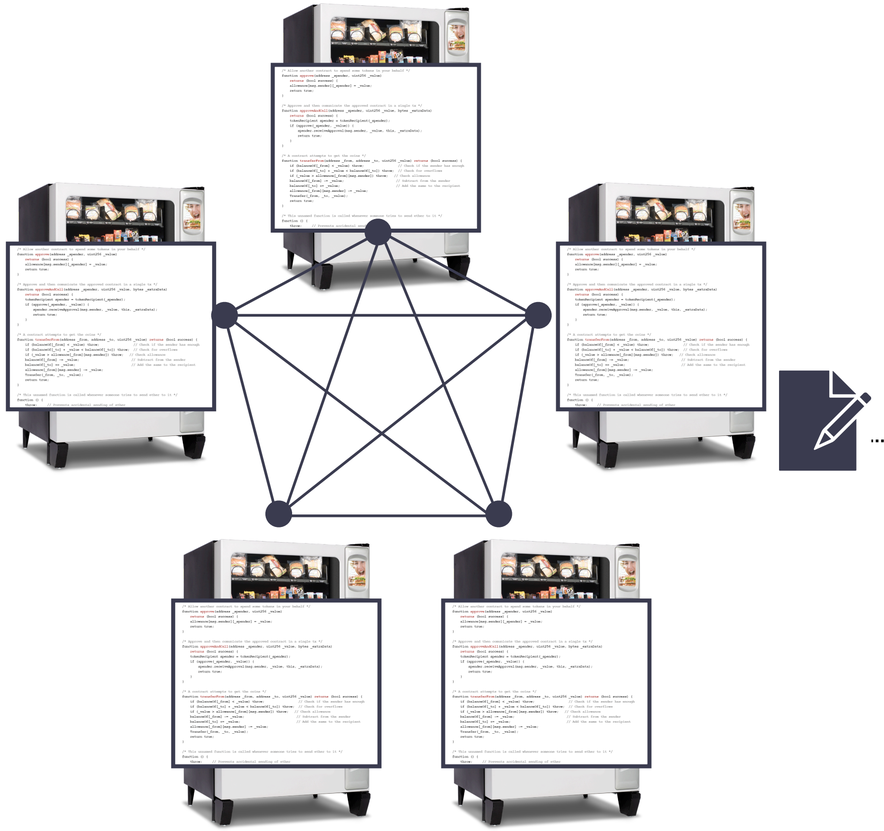
network of validator nodes



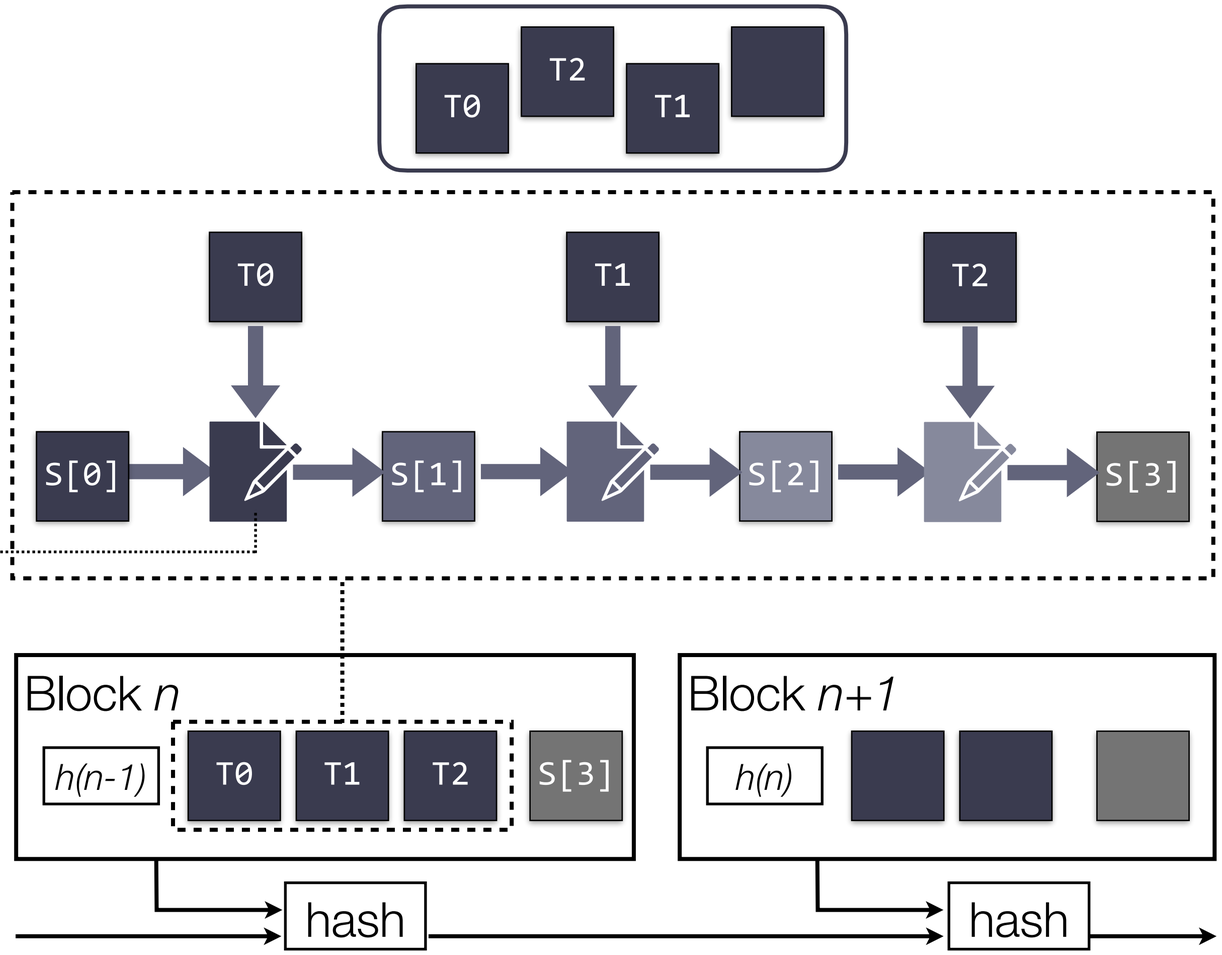
Transaction pool



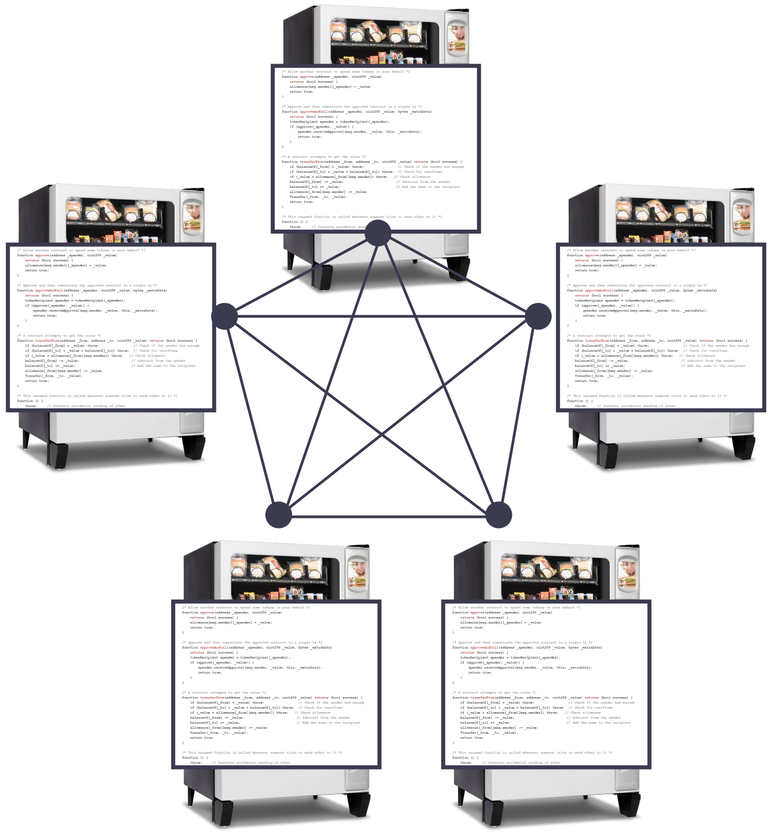
A blockchain ensures the network agrees on a single global order



network of validator nodes



Contracts are compiled into bytecode for a simple stack machine



network of validator nodes

Bob's contract

.sol

solidity

solidity source code

Solidity compiler

EVM bytecode

Ethereum Virtual Machine

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

.sol

(source: official Solidity documentation, https://docs.soliditylang.org)

Smart contracts on Ethereum: an example

Define a new contract.

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Define the contract state.

All state is replicated and publicly persisted on the blockchain.

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Define a constructor.

The constructor is run during creation of the contract and cannot be called afterwards.

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Define functions.

Can be called by clients or contracts.

Can update the contract's state.

Functions are "called" by sending a transaction.

Each transaction is cryptographically signed with the sender's identity.

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

The address of the account that created the contract.

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Only the contract owner can mint new coins.

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

A “hashmap” that keeps track of the amount of TomCoin in each address.

address	uint
0xde0b295669a9fd93d5f…	20
0x931D387731bBbC988B3…	100
0x2212D359CF1c5454Ae9…	0
0x931D387731bBbC988B3…	50
0xC55EdDadEeB47fcDE0B…	0

Smart contracts on Ethereum: an example

```
contract TomCoin {

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

    constructor() public {
        minter = msg.sender;
    }

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

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

Transfer TomCoin from sender address to receiver address, if sender has sufficient funds.

“Condition-oriented Programming” (Gavin Wood)

A style of programming that ensures function bodies have no conditional paths.

“Don’t mix (state) transitions with conditions”

```
contract TomCoin {
    …

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

```
contract TomCoin {
    …
    modifier only_with_at_least(uint amount) {
        if (balances[msg.sender] >= amount) _;
    }

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

(source: https://gavofyork.medium.com/condition-orientated-programming-969f6ba0161a)

Programmable money: the ERC-20 token standard

```
interface IERC20 {

  function totalSupply() public constant returns (uint);
  function balanceOf(address owner) public constant returns (uint balance);
  function allowance(address owner, address spender) public constant returns (uint remaining);
  function transfer(address to, uint tokens) public returns (bool success);
  function approve(address spender, uint tokens) public returns (bool success);
  function transferFrom(address from, address to, uint tokens) public returns (bool success);

  event Transfer(address indexed from, address indexed to, uint tokens);
  event Approval(address indexed tokenOwner, address indexed spender, uint tokens);

}
```

Programmable money: the ERC-20 token standard

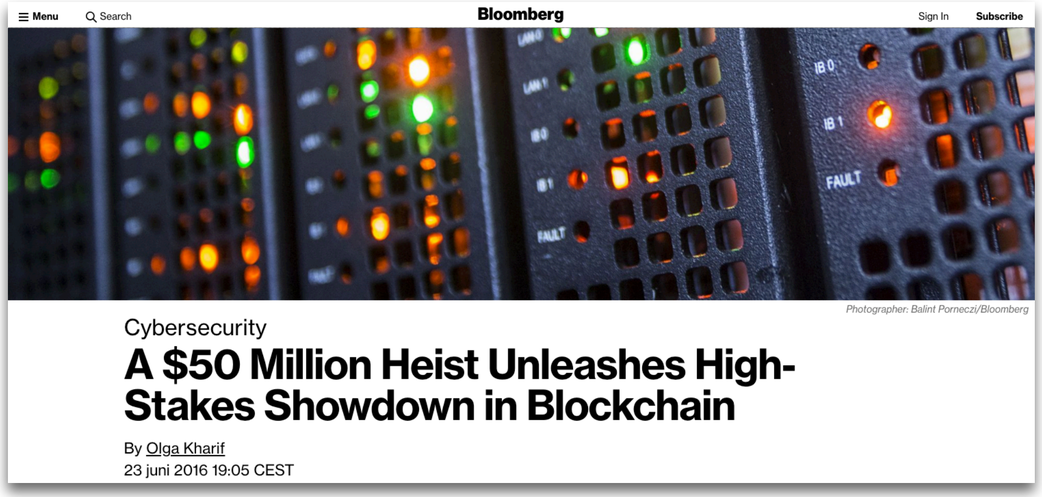
```
interface IERC20 {

  function totalSupply() public constant returns (uint);
  function balanceOf(address owner) public constant returns (uint balance);
  function allowance(address owner, address spender) public constant returns (uint remaining);
  function transfer(address to, uint tokens) public returns (bool success);
  function approve(address spender, uint tokens) public returns (bool success);
  function transferFrom(address from, address to, uint tokens) public returns (bool success);

  event Transfer(address indexed from, address indexed to, uint tokens);
  event Approval(address indexed tokenOwner, address indexed spender, uint tokens);

}
```

$23.4 **Billion** market

(source: etherscan.io, retrieved 08/01/2021)

The risks of writing smart contracts

The DAO Hack (2016)



~$50 million stolen

cause: forgot to recheck contract state after call to external contract (basic re-entrancy bug)

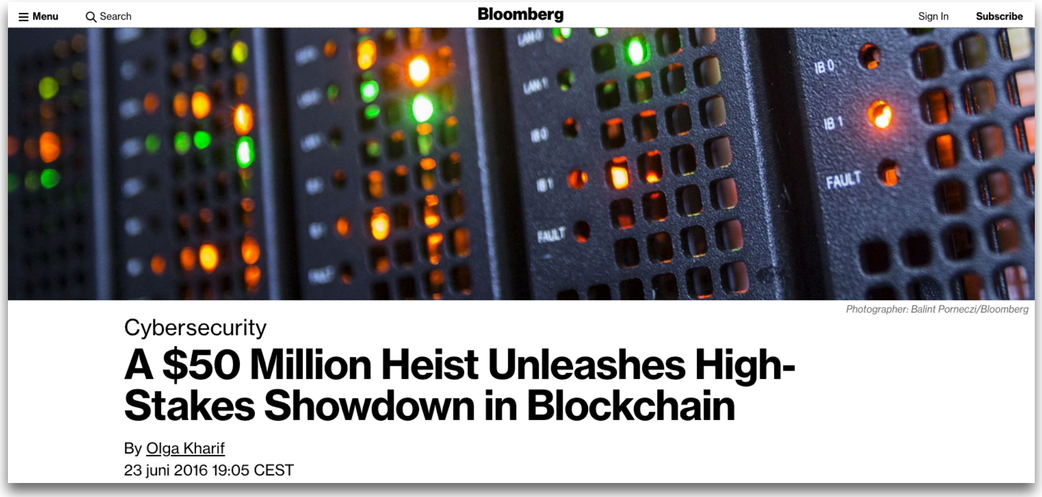
Parity freeze bug (2017)

~$280 million accidentally frozen

cause: forgot to initialize field in constructor

The risks of writing smart contracts

The DAO Hack (2016)



~$50 million stolen

cause: forgot to recheck contract state after call to external contract (basic re-entrancy bug)

```
contract DAO {

    mapping (address => uint) public balances;
    …

    function withdrawBalance() public {
        bool result = msg.sender.call.value(balances[msg.sender])();
        if (!result) {
          throw;
        }
        // update withdrawer's balance
        balances[msg.sender] = 0;
    }
    …
}
```

```
contract Proxy {
    …

    function () public payable {
        DAO(msg.sender).withdrawBalance();
    }
    …
}
```

What can we do? PIdesign for smart contracts

> “There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”
>
> - C.A.R. Hoare