A gentle introduction to Ethereum and “smart contracts”

Tom Van Cutsem
DistriNet KU Leuven

tvcutsem.github.io be.linkedin.com/in/tomvc github.com/tvcutsem x.com/tvcutsem @tvcutsem@techhub.social

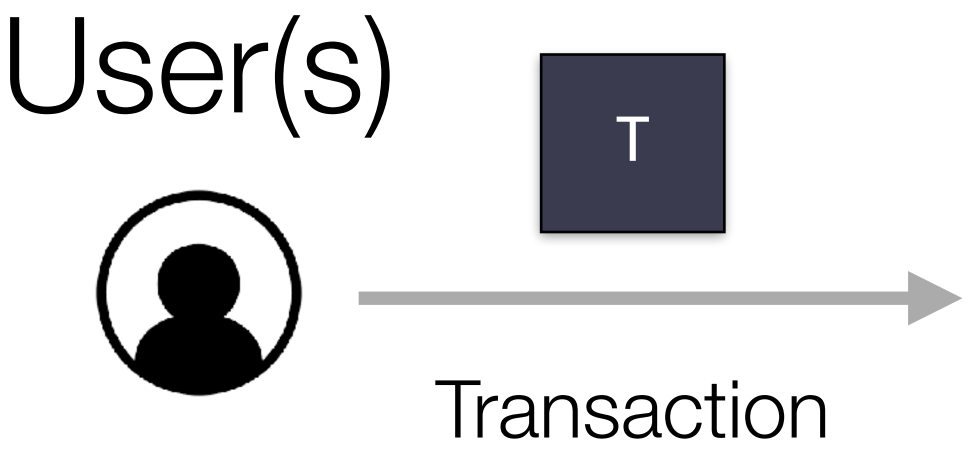
This Session

- **Ethereum**, a “programmable” blockchain
- **Smart contracts**: what is a smart contract? How does it relate to blockchains?
- **Solidity**: a programming language to write smart contracts
- Decentralized applications (**Dapps**): web apps backed by smart contracts
- Challenges, tools, advice

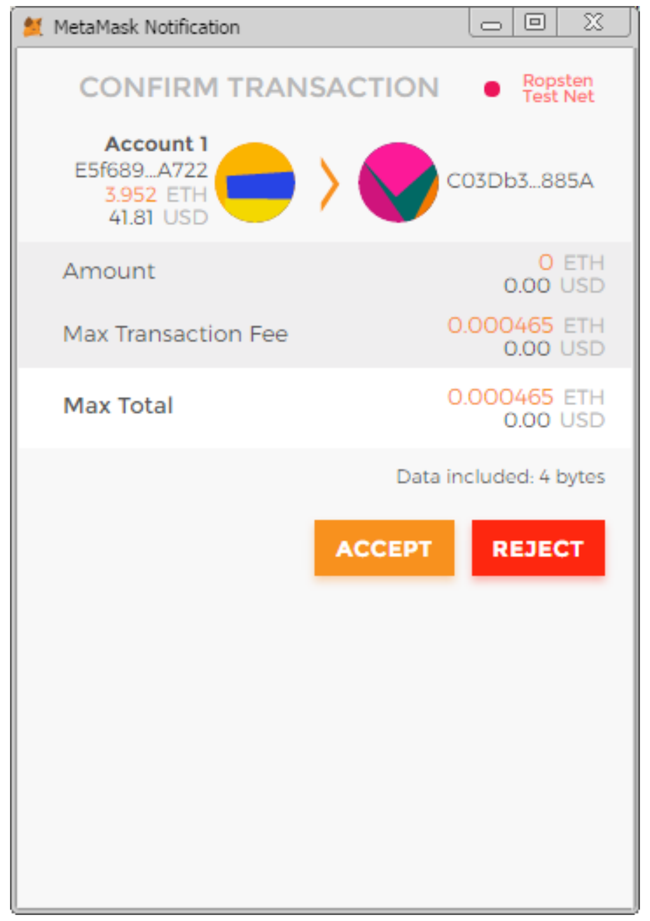
Blockchains

Visualizing the blockchain: https://tx.town/v/eth

Physical view: a blockchain is a peer-to-peer network of computers



Wallet


MetaMask Notification

CONFIRM TRANSACTION ● Ropsten Test Net

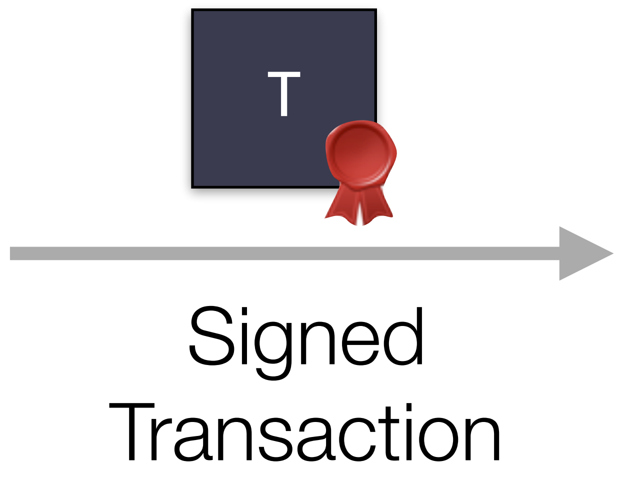
Account 1
E5f689...A722
3.952 ETH
41.81 USD

C03Db3...885A

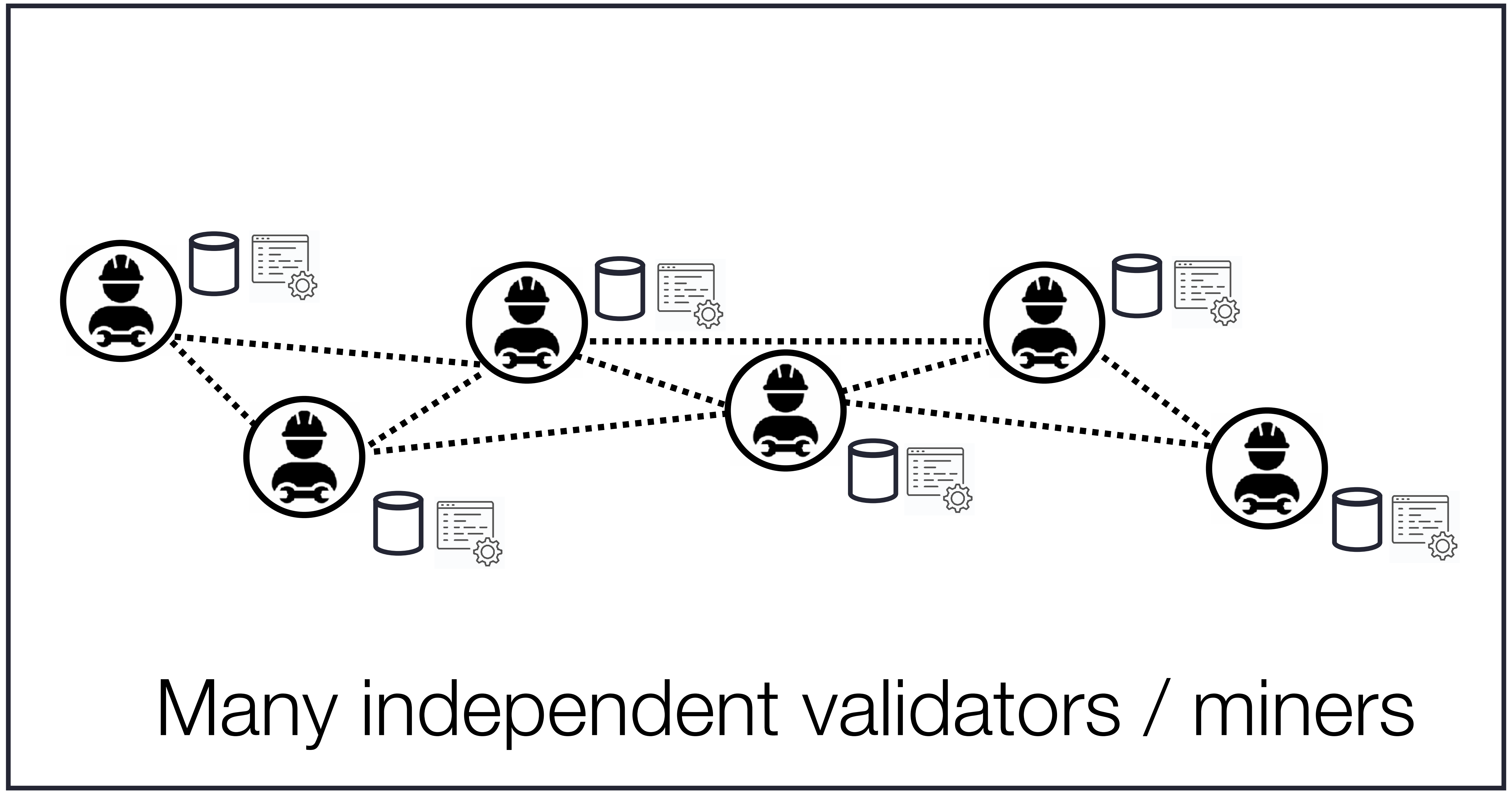
Amount	0 ETH 0.00 USD
Max Transaction Fee	0.000465 ETH 0.00 USD
Max Total	0.000465 ETH 0.00 USD

Data included: 4 bytes

ACCEPT REJECT

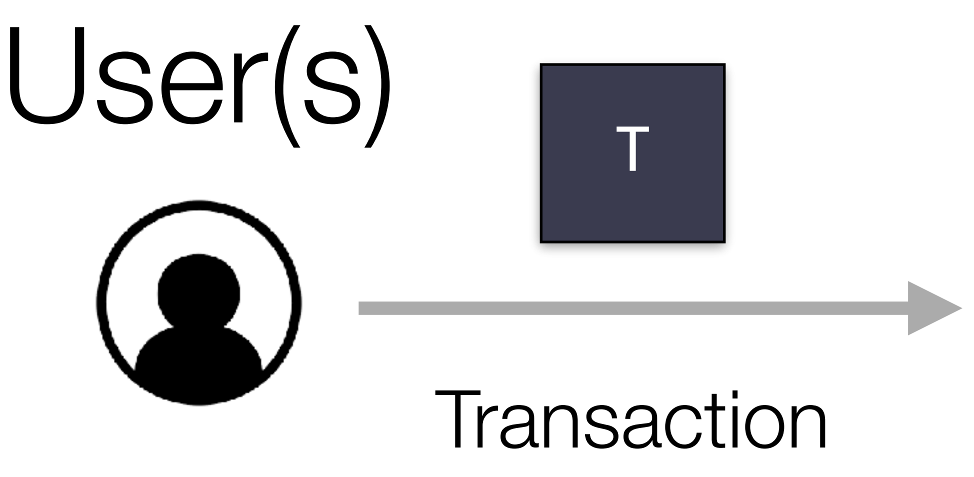


Blockchain network

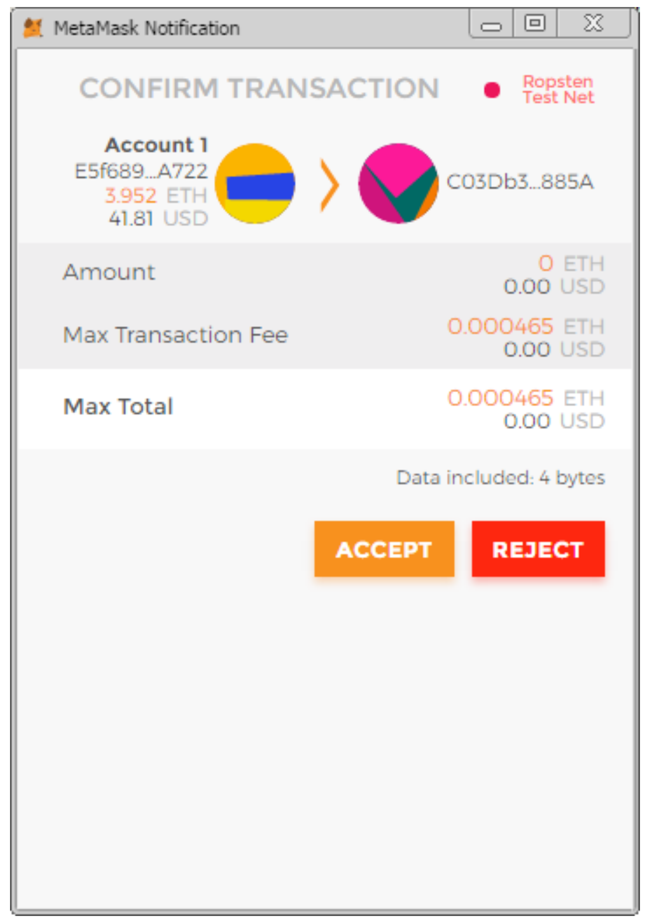


Logical view: a blockchain is a transaction processing machine

User(s)



Wallet


MetaMask Notification

CONFIRM TRANSACTION ● Ropsten Test Net

Account 1
E5f689...A722
3.952 ETH
41.81 USD

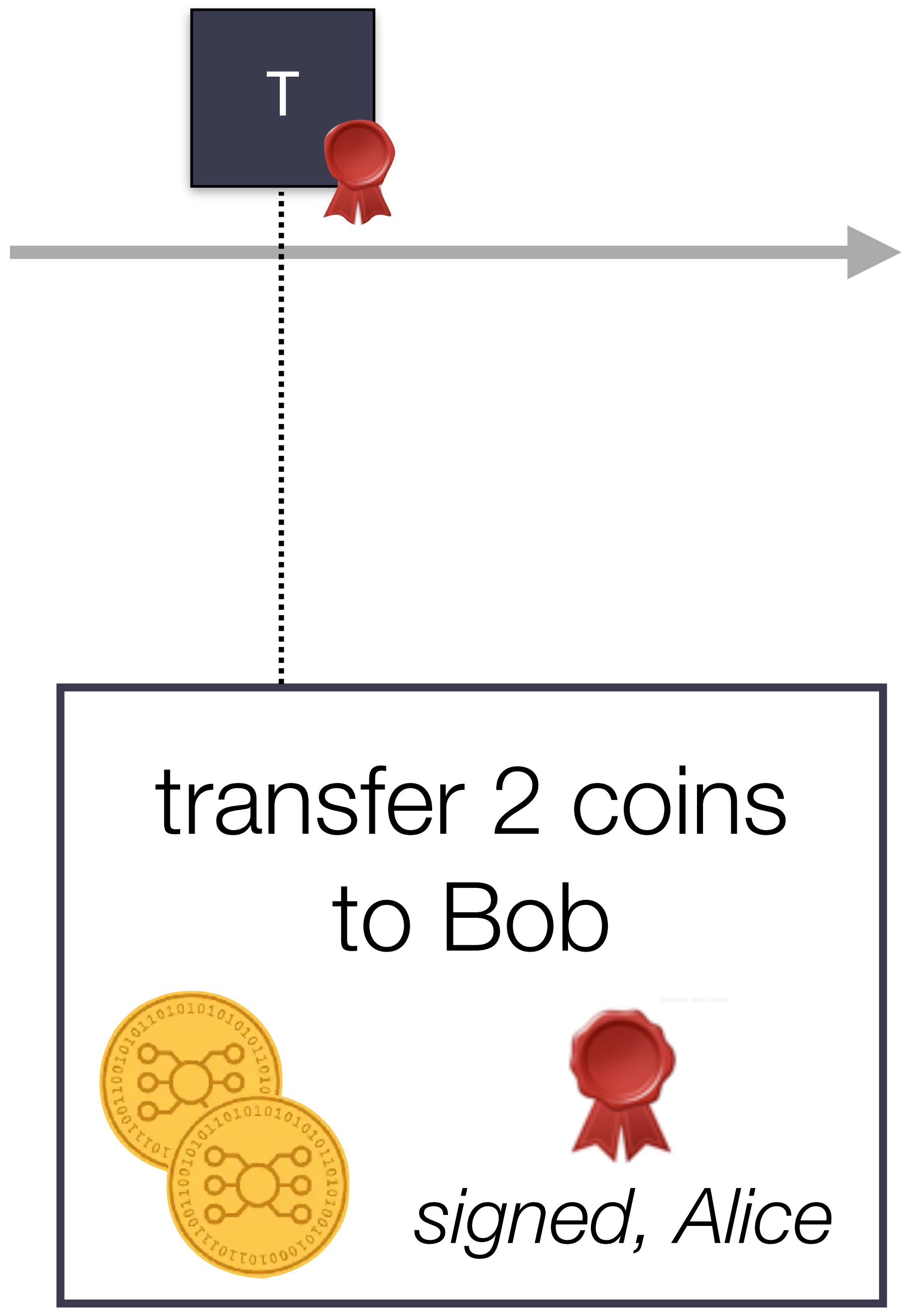
C03Db3...885A

Amount: 0 ETH, 0.00 USD

Max Transaction Fee: 0.000465 ETH, 0.00 USD

Max Total: 0.000465 ETH, 0.00 USD

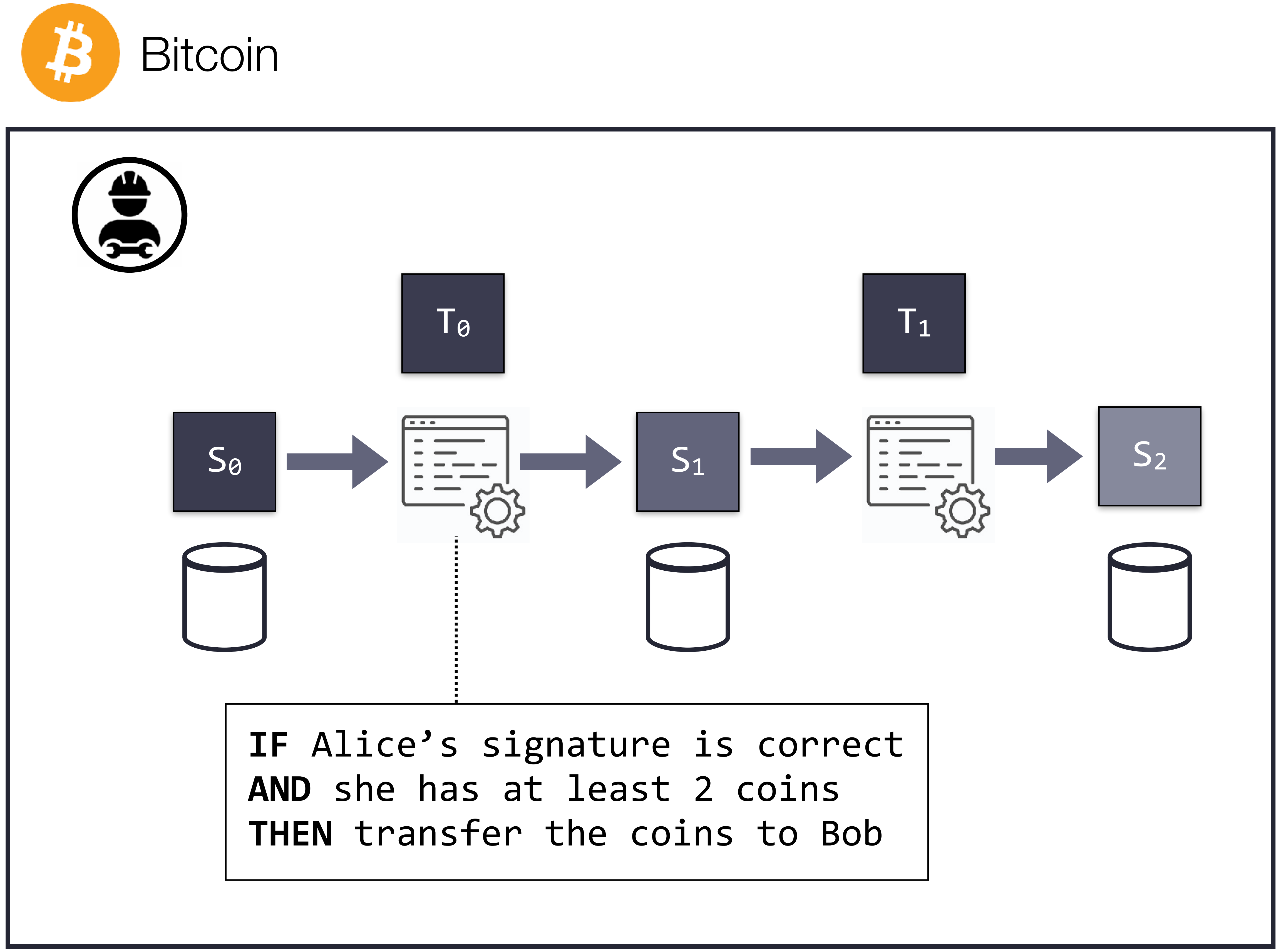
Data included: 4 bytes



transfer 2 coins to Bob

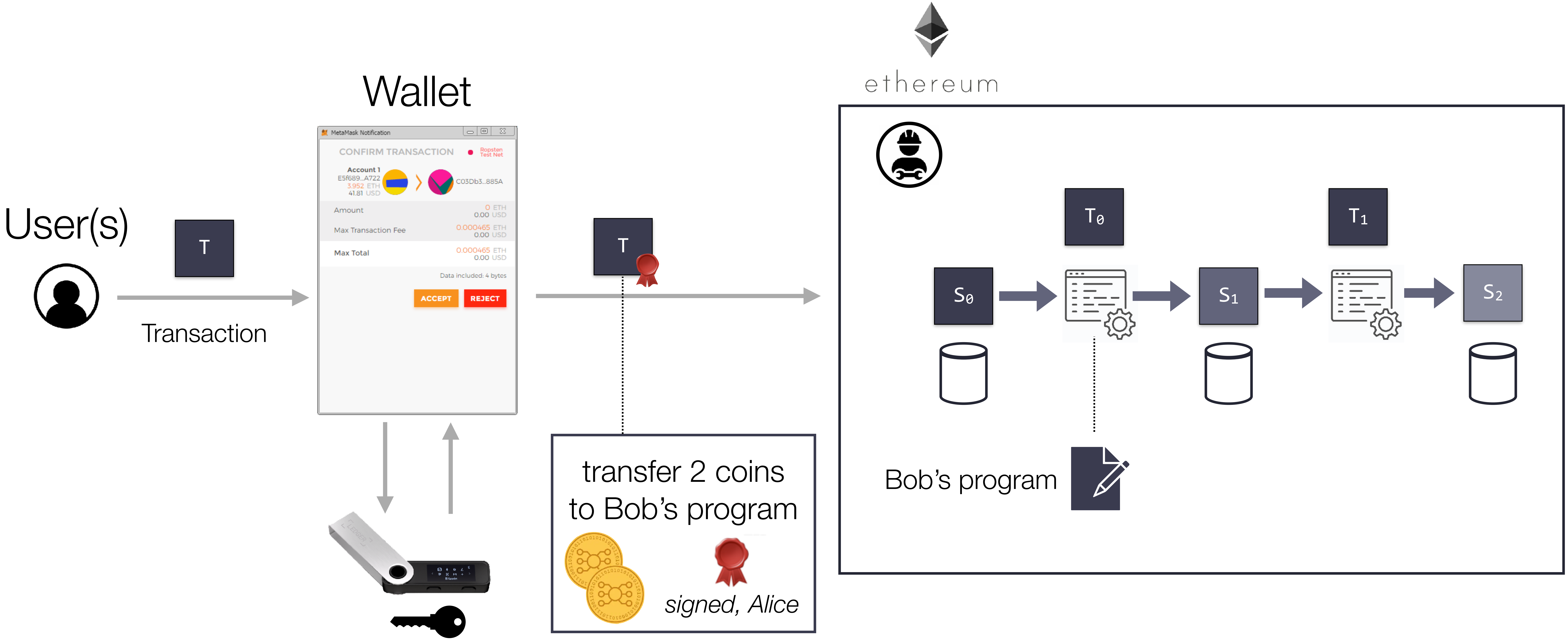
signed, Alice

Bitcoin



```
IF Alice’s signature is correct
AND she has at least 2 coins
THEN transfer the coins to Bob
```

Ethereum’s innovation: make the transactions programmable!



MetaMask Notification

CONFIRM TRANSACTION ● Ropsten Test Net

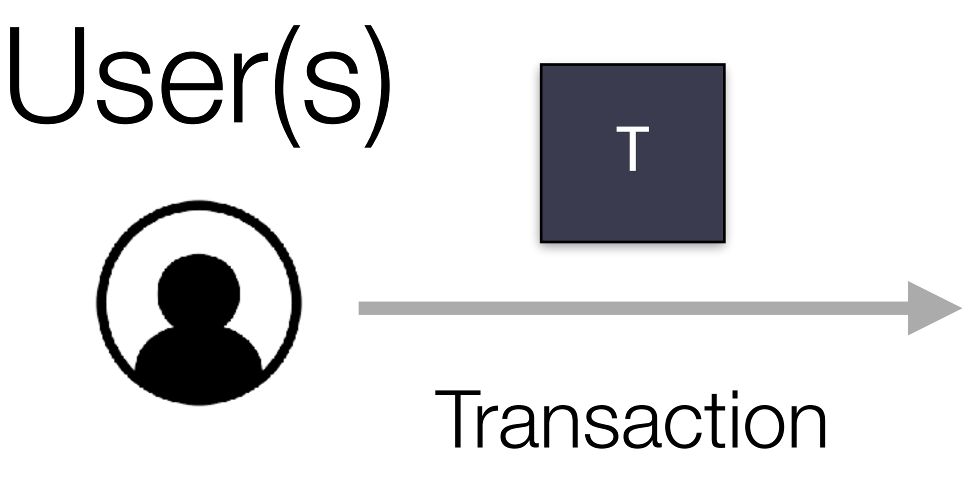
Account 1
E5f689...A722
3.952 ETH
41.81 USD
C03Db3...885A

Amount	0 ETH 0.00 USD
Max Transaction Fee	0.000465 ETH 0.00 USD
Max Total	0.000465 ETH 0.00 USD

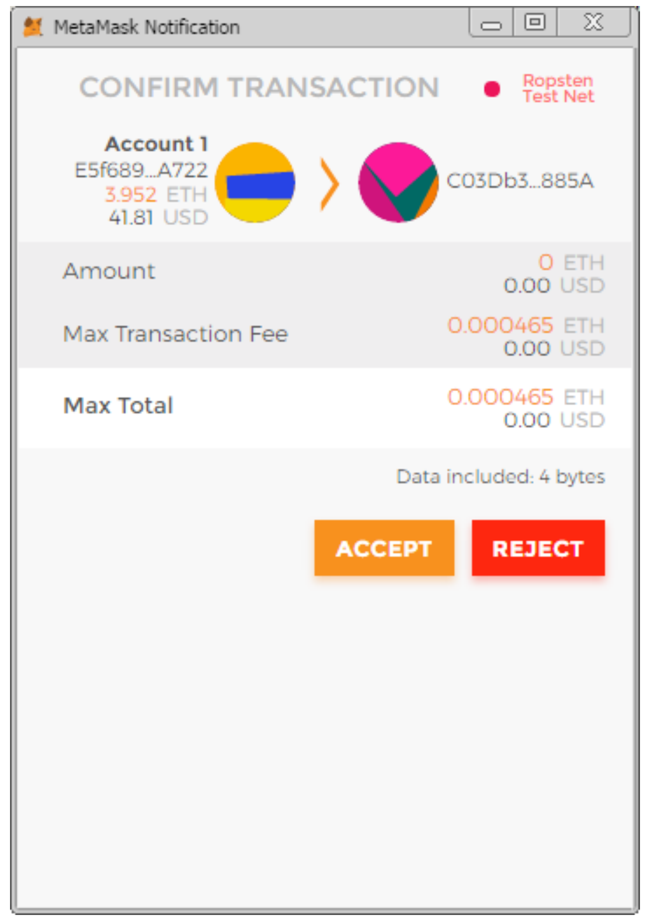
Data included: 4 bytes

ACCEPT REJECT

Ethereum’s innovation: make the transactions programmable!



Wallet



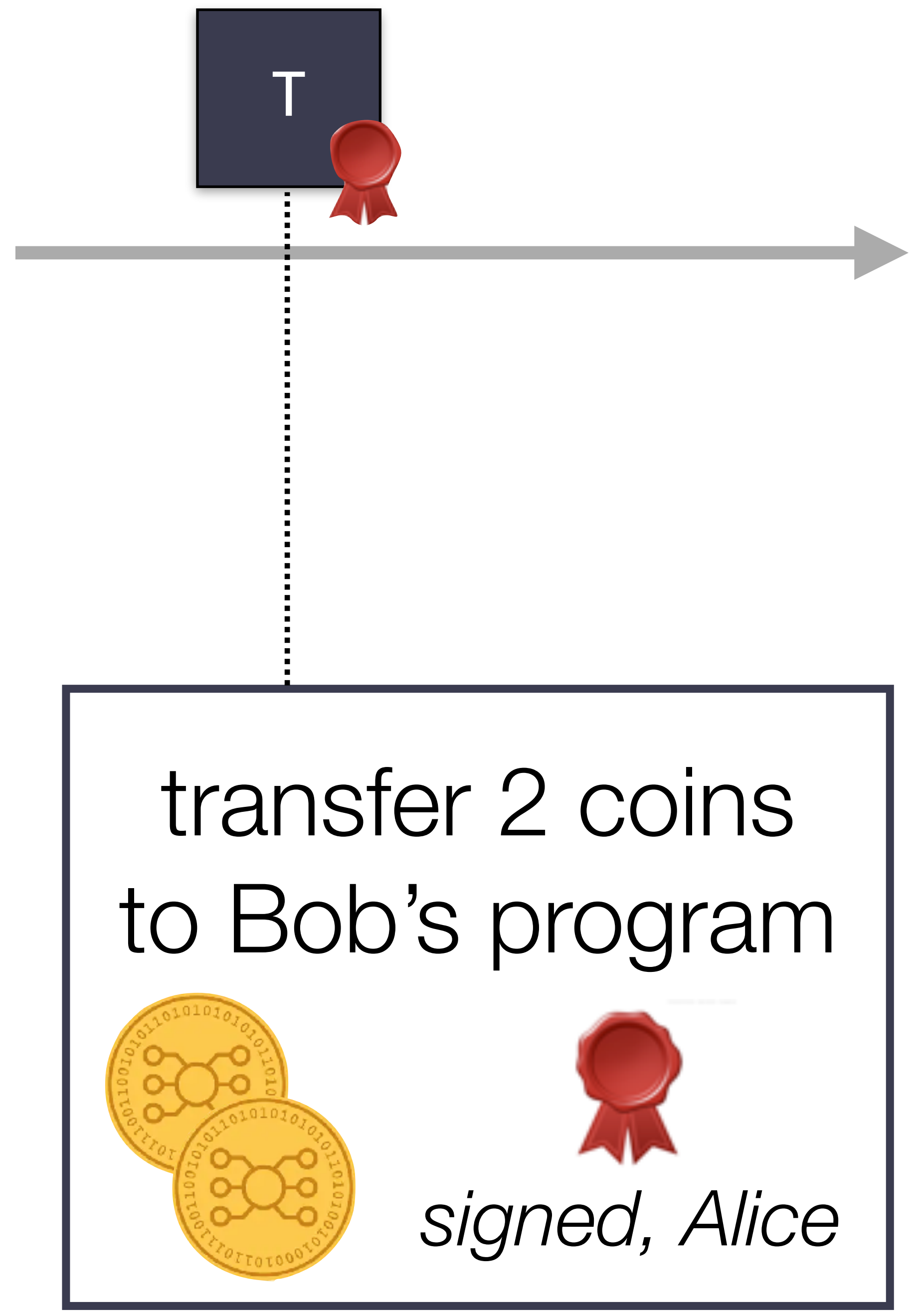
CONFIRM TRANSACTION ● Ropsten Test Net

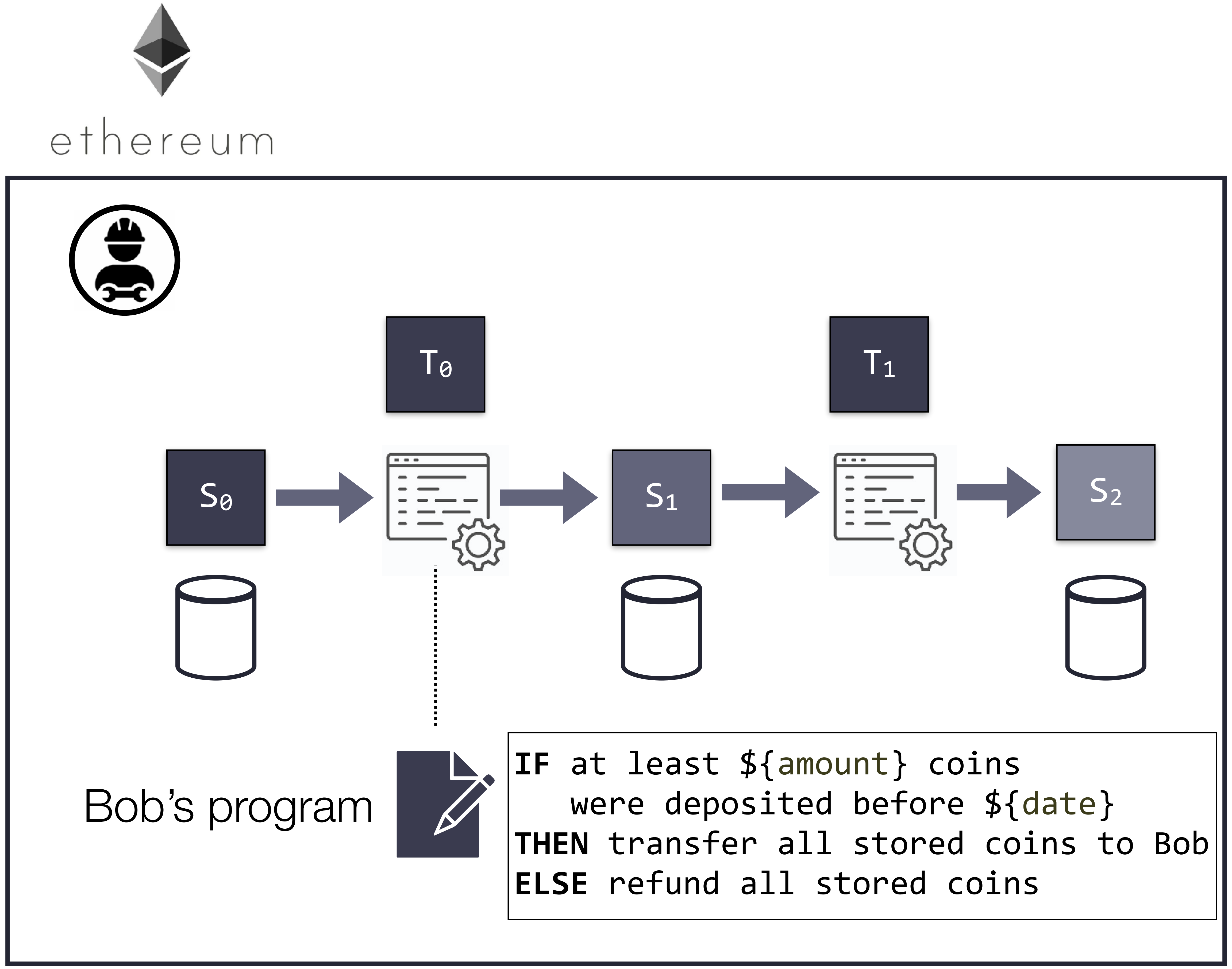
Account 1
E5f689...A722
3.952 ETH
41.81 USD
〉 C03Db3...885A

Amount	0 ETH 0.00 USD
Max Transaction Fee	0.000465 ETH 0.00 USD
Max Total	0.000465 ETH 0.00 USD

Data included: 4 bytes

ACCEPT REJECT

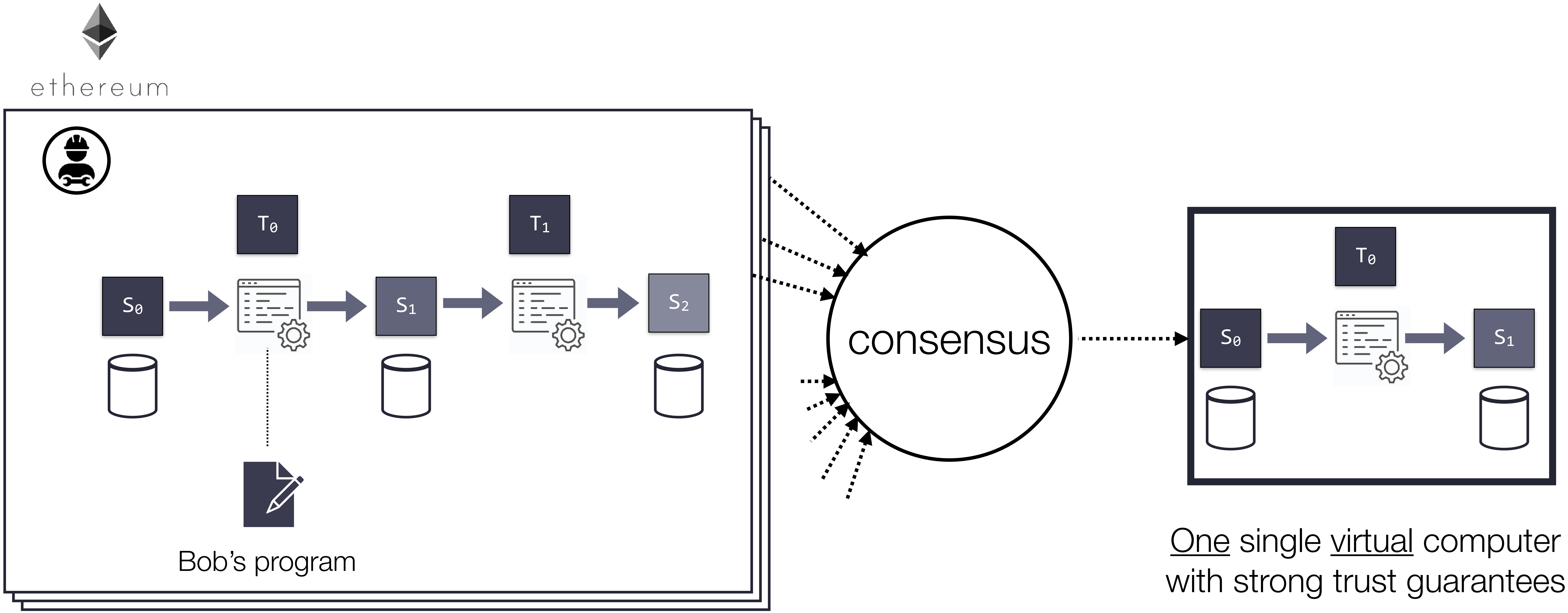




```
IF at least ${amount} coins
   were deposited before ${date}
THEN transfer all stored coins to Bob
ELSE refund all stored coins
```

Example: a basic crowdfunding contract

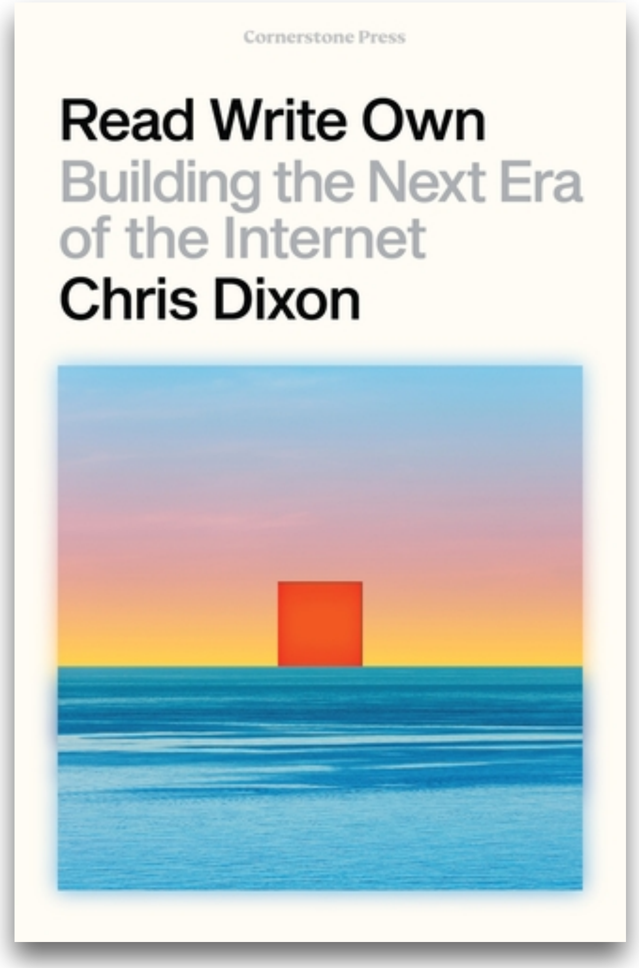
Blockchains as *trusted* virtual computers



Many (1000s) untrustworthy physical computers

One single virtual computer with strong trust guarantees

“Blockchains are computers that can make *credible commitments*”



Cornerstone Press

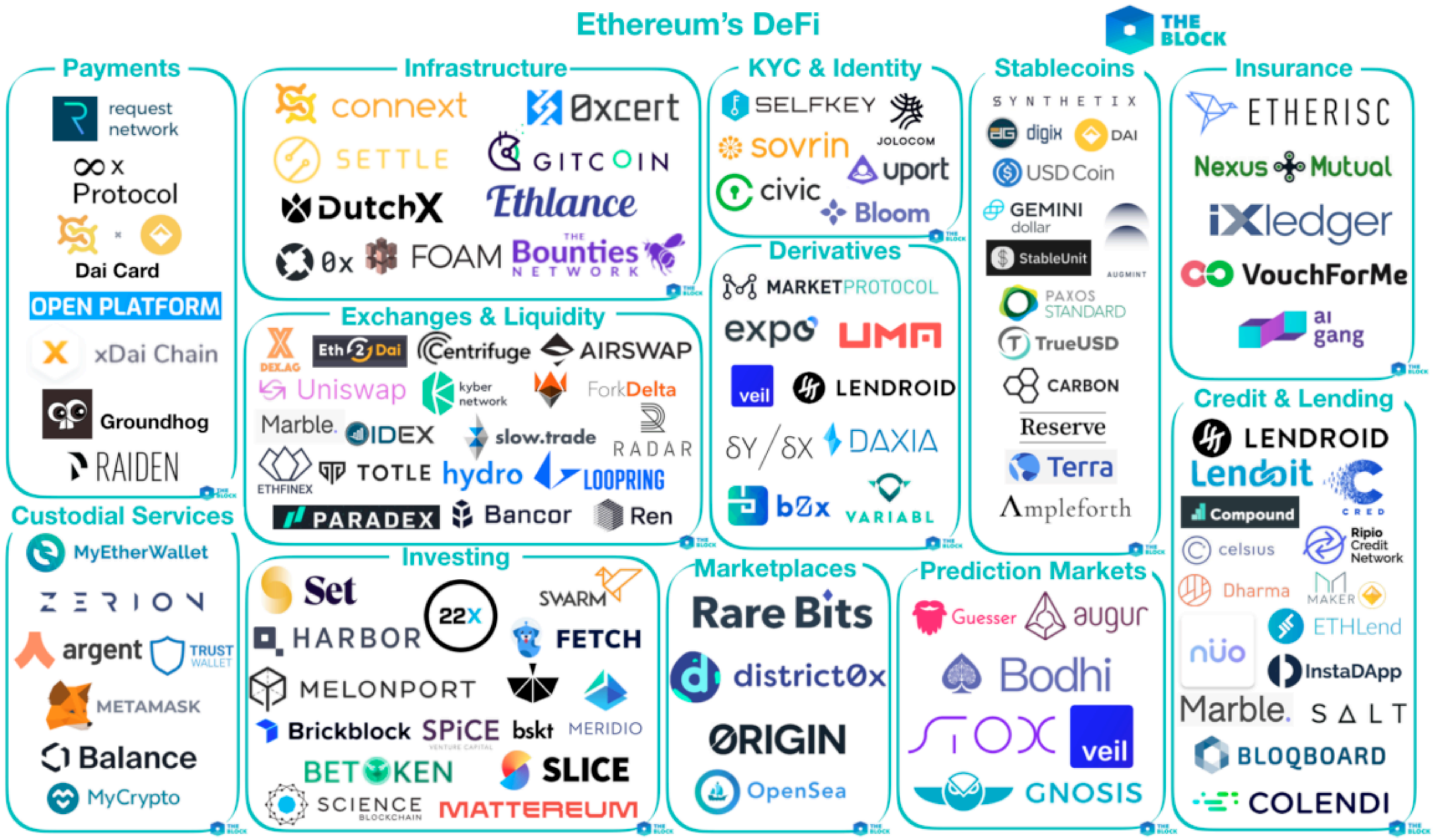
Read Write Own

Building the Next Era of the Internet

Chris Dixon

(Source: Chris Dixon, Crypto Networks and Why They Matter, 2020)

Applications? Ethereum's "Decentralized Finance"



(image credit: theblockcrypto.com)

Fungible tokens (ERC-20)

Non-fungibles (NFTs)

Stablecoins

SBTs

.eth names

…

New kinds of **electronic rights** collectively worth over **$100 Billion**

(source: coingecko.com, retrieved May 2024)

Smart contracts

What is a smart contract?

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

(Vitalik Buterin, Ethereum White Paper, 2014)

What is a smart contract?

A software program that can receive, store & send “money”

Essentially, a program with its own “bank account”

Smart contracts: origins

- The term “smart contract” was first proposed by cryptographer Nick Szabo in 1995.
- **Goal**: digitally automate multi-party business agreements using computer protocols and cryptography to **reduce counterparty risk** (the risk of the other party not executing on what they promised after they agreed to the contract)
- The key idea:
    - Express the **terms & conditions** of a trade agreement **as** executable **code**.
    - Parties agree to the contract by cryptographically transferring control of their (digital) assets to the contract thus “locking up” their assets.
    - The **contract keeps the assets in escrow**. Assets can only be transferred out of the contract according to the logic written in the code.
    - The computer that runs the code acts like a judge enforcing a legal contract.
- A note on **terminology**: smart contracts are neither “smart” as in “using AI”, nor legally binding “contracts”.

Cryptographer Nick Szabo,
Inventor of the term “smart contract”

Smart contracts: basic principle

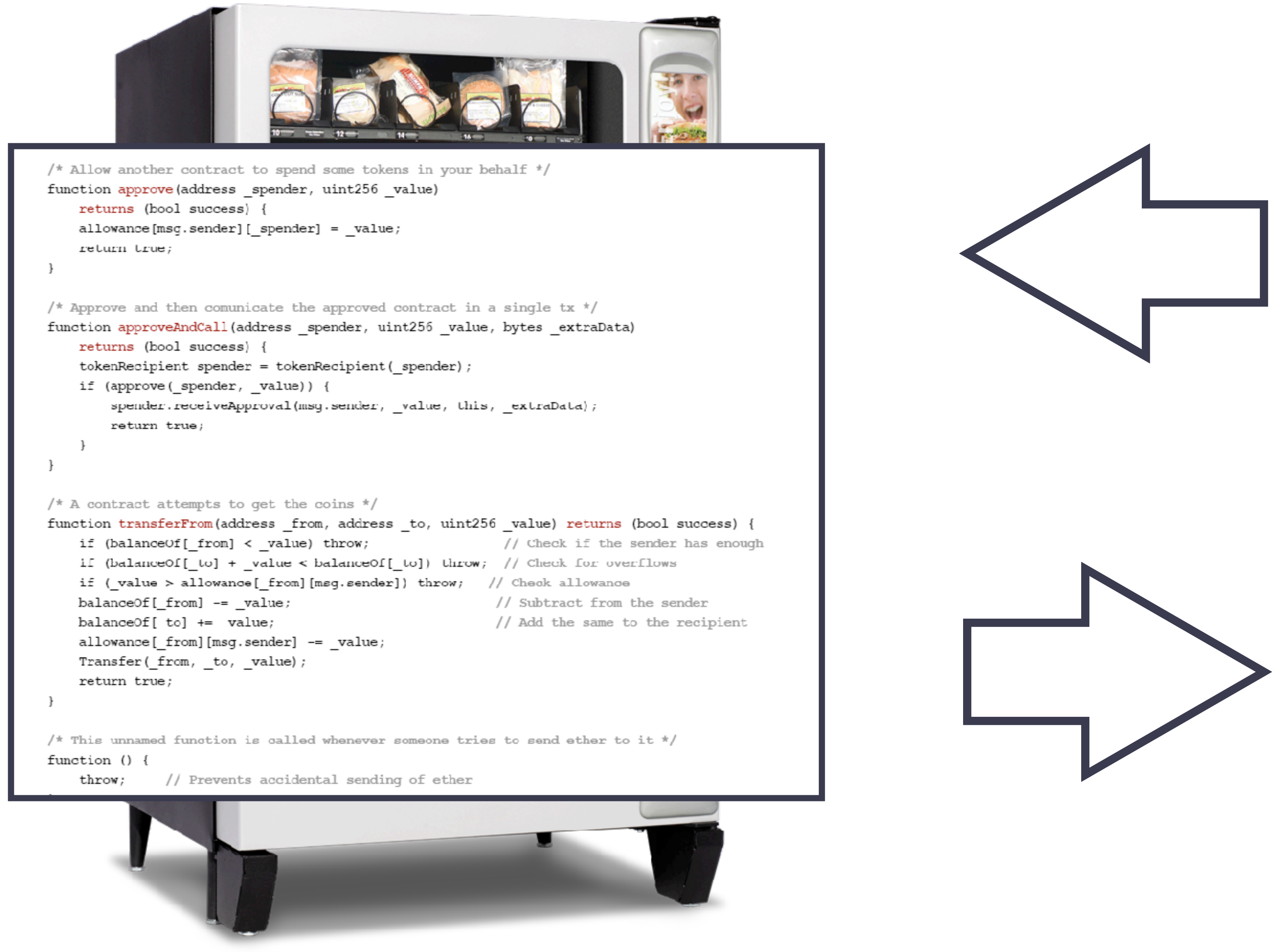
- A vending machine is an **automaton** that can trade **physical** assets

1. insert coins

2. dispense drink

Smart contracts: basic principle

- A smart contract is 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 (tokens)

2. dispense other digital assets
or electronic rights

But who should we trust to faithfully execute the automaton’s code?

- A smart contract is 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
```

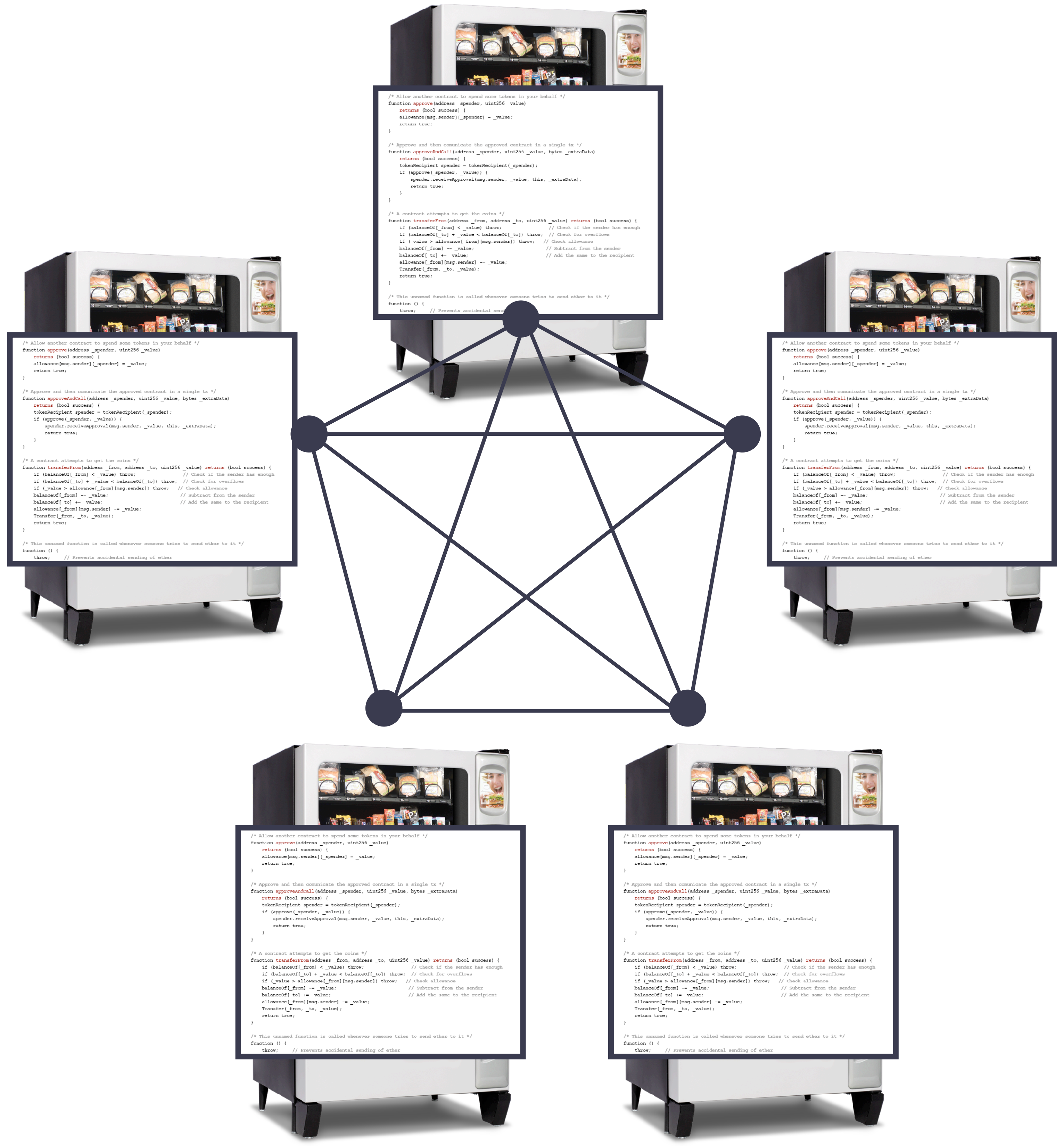
code

1. insert digital coins (tokens)

2. dispense other digital assets or electronic rights

Delegate trust to a decentralised network

- A smart contract is a **replicated automaton** that can trade **digital** assets

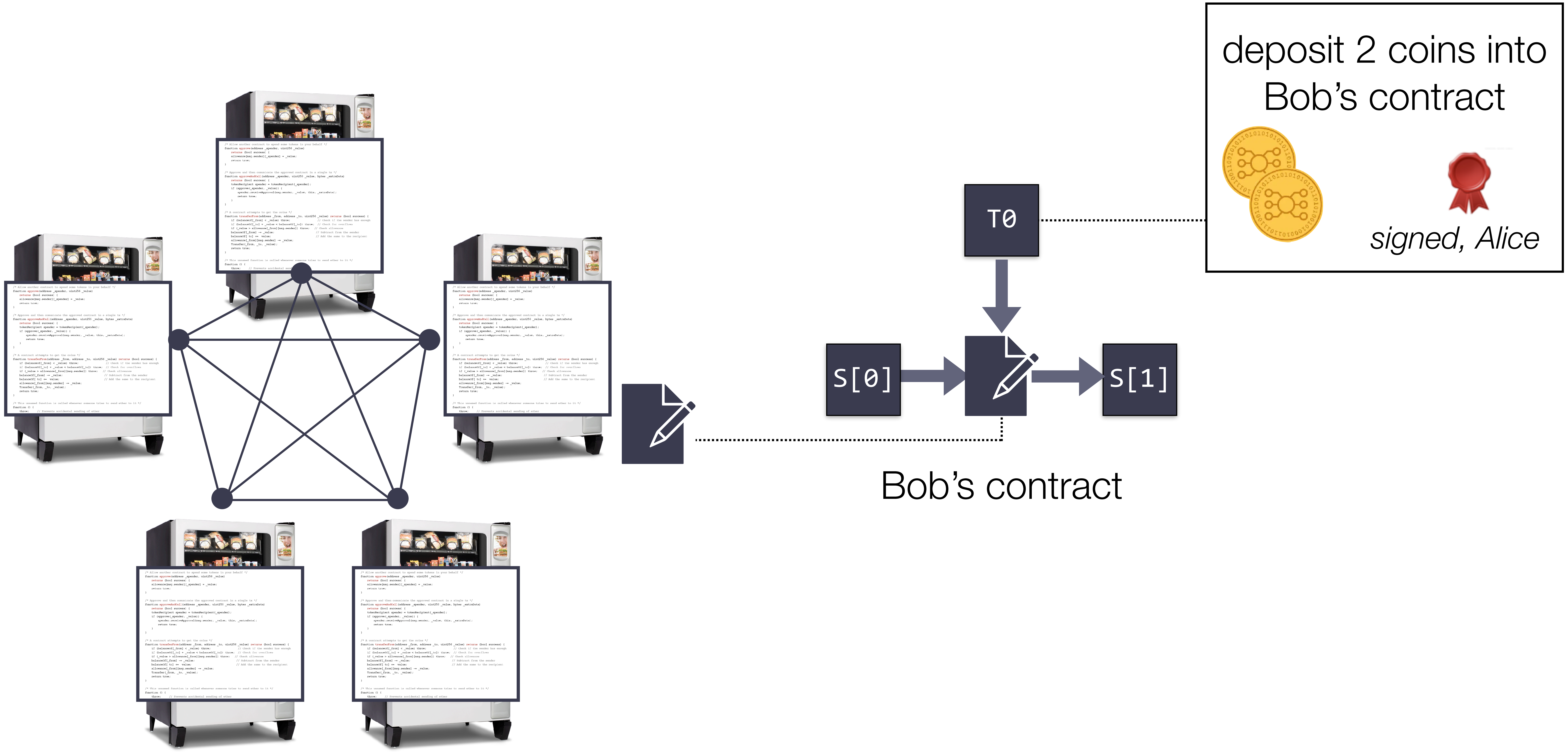


replicated code

1. insert digital coins (tokens)

2. dispense other digital assets or electronic rights

Each transaction updates the virtual computer’s replicated state



deposit 2 coins into Bob’s contract

signed, Alice

T0

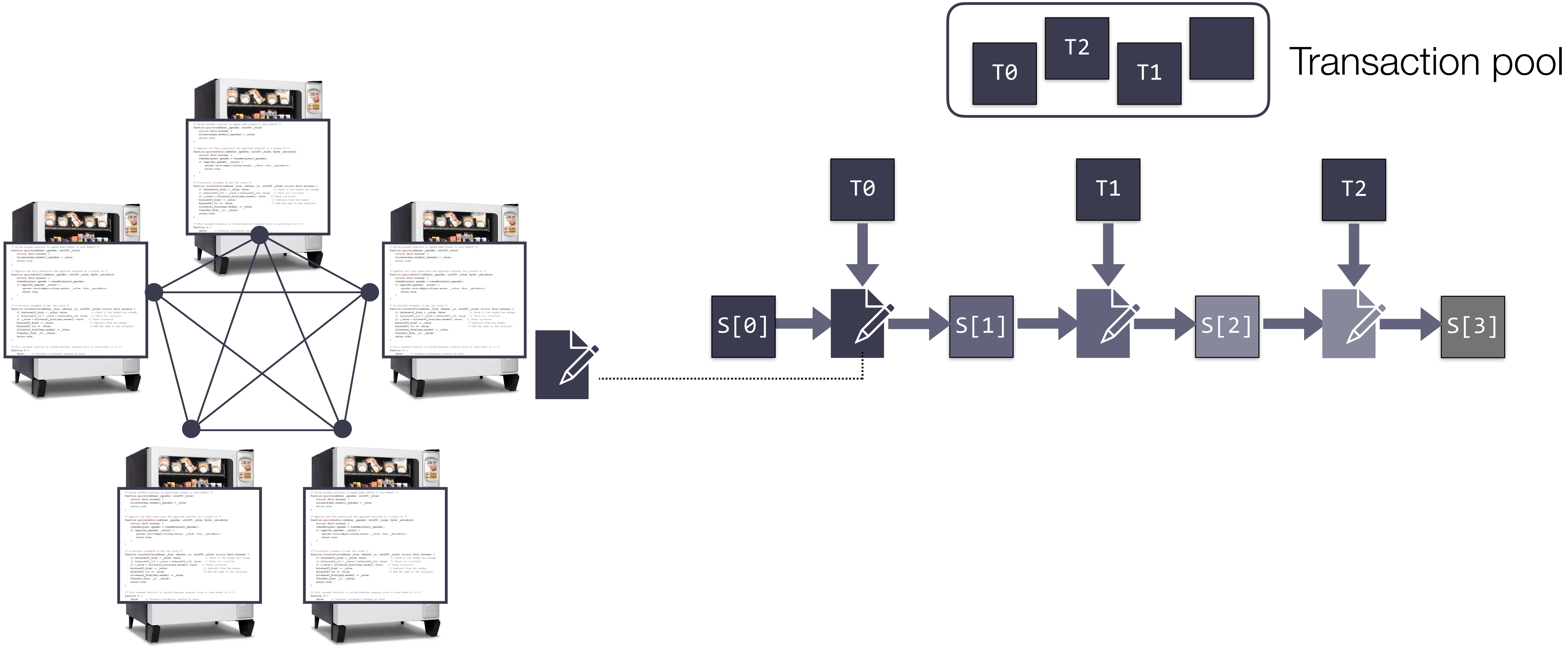
S[0]

S[1]

Bob’s contract

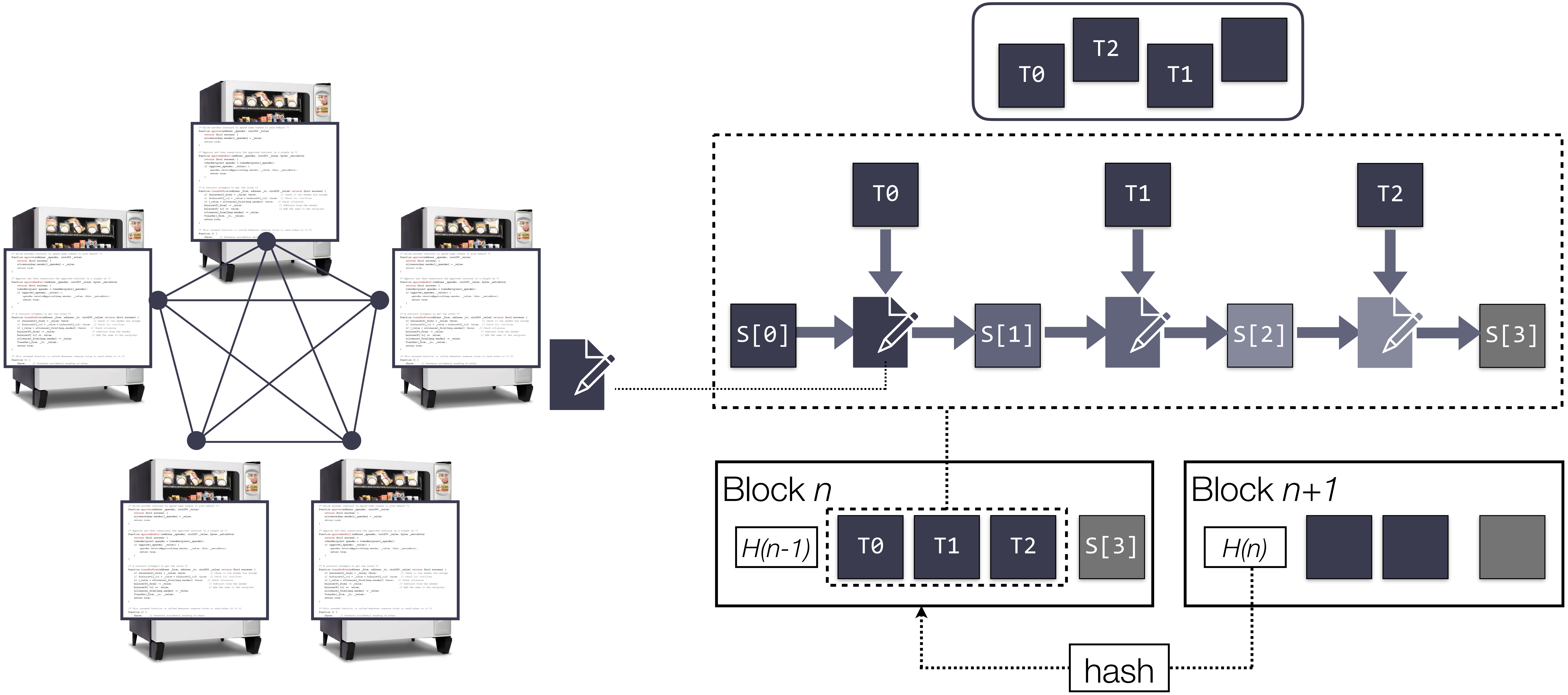
network of validator nodes

Incoming transactions are sequenced into blocks



network of validator nodes

A blockchain ensures the network agrees on a single global order



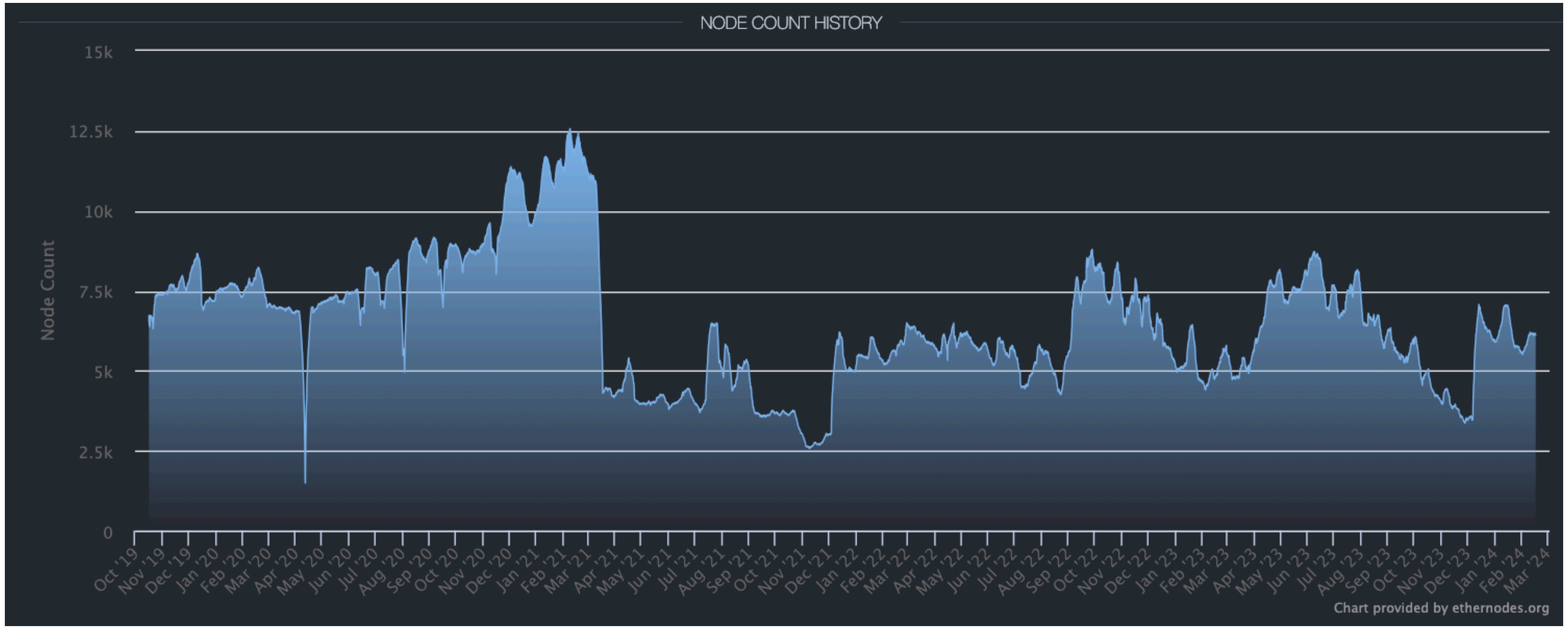
network of validator nodes

The Ethereum network

- In reality, the network is made up of thousands of computers

- Statistics of the Ethereum "mainnet":

Evolution of # of network nodes over time in last 4 years (Nov 2019 - Feb 2024)

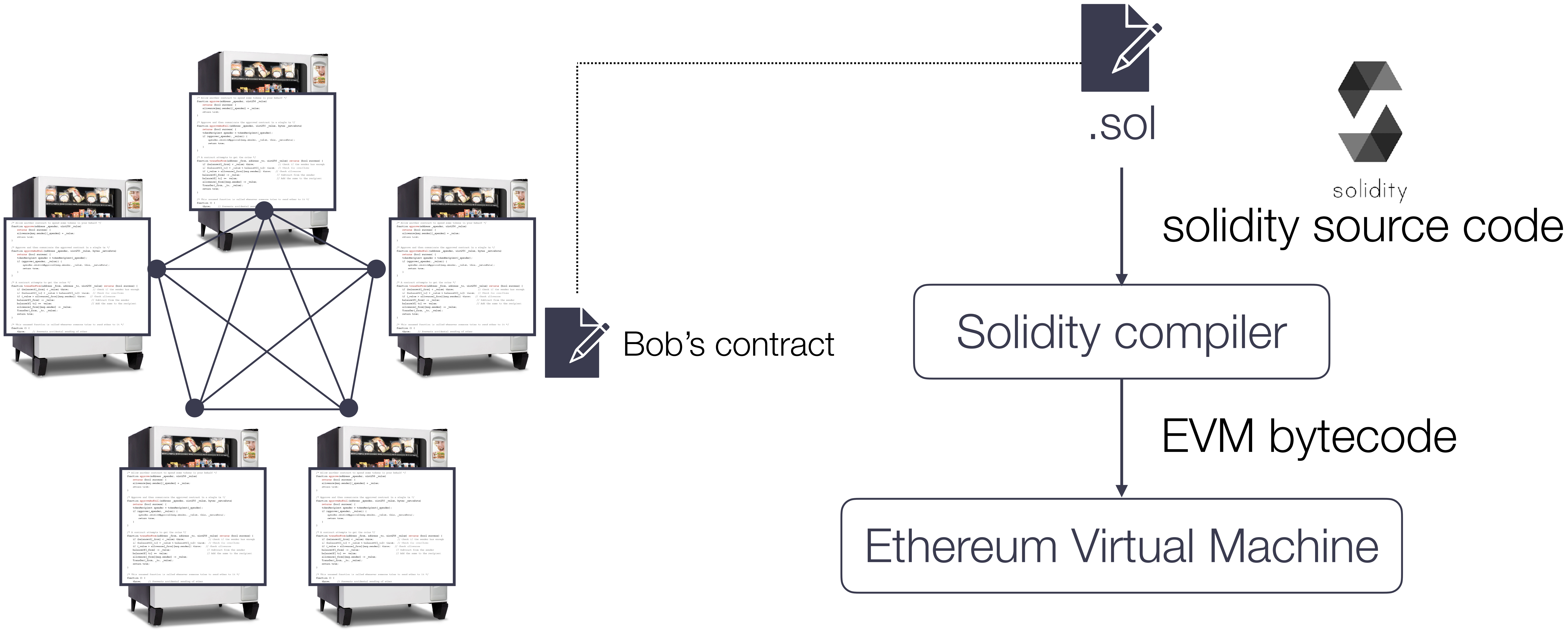


(Source: ethernodes.org, February 2024)

6000+ network peers
18+ countries
5+ distinct software implementations

Smart contracts on Ethereum

Contracts are compiled into bytecode for a simple stack machine



.sol

solidity

solidity source code

Solidity compiler

EVM bytecode

Ethereum Virtual Machine

Bob’s contract

network of validator nodes

Smart contracts on Ethereum: a basic example

.sol

```
contract NameRegistry {

   mapping (string => address) public registry;

   constructor() {}

   function claimName(string name) public payable {
       require(msg.value >= 1 ether);
       if (registry[name] == address(0)) {
           registry[name] = msg.sender;
       }
   }

   function ownerOf(string name) public view {
       return registry[name];
   }
}
```

(Code example based on Narayanan *et al.* handbook, section 10.7)

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

Define a new contract.

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

Define the contract state.

All state is replicated and publicly persisted on the blockchain.

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

   mapping (string => address) public registry;

   constructor() {}

   function claimName(string name) public payable {
       require(msg.value >= 1 ether);
       if (registry[name] == address(0)) {
           registry[name] = msg.sender;
       }
   }

   function ownerOf(string name) public view {
       return registry[name];
   }
}
```

Define a constructor.

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

We don’t need to do any initialisation in this simple contract. The mapping by default maps every string to the 0 address

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

Define functions.

Can be called by external clients or other contracts.

Can update the contract’s state.

Functions can be “called” by sending a transaction to the Ethereum network.

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

A table that keeps track of the owner address of each registered name

string	address
“Alice”	0xde0b295669a9fd93d5f…
“Bob’s program”	0x2212D359CF1c5454Ae9…
“a message”	0x721E221531b7bC98DB2…
“ethereum.org”	0xC55EdDadEeB47fcDE0B…

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

Functions are “called” by sending a transaction.

Each transaction is cryptographically signed by the sender and contains the sender’s address (`msg.sender`) and may optionally contain any amount of tokens (ether) sent along with it (`msg.value`).

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

    mapping (string => address) public registry;

    constructor() {}

    function claimName(string name) public payable {
        require(msg.value >= 1 ether);
        if (registry[name] == address(0)) {
            registry[name] = msg.sender;
        }
    }

    function ownerOf(string name) public view {
        return registry[name];
    }
}
```

Bob can register the name “Bob” by creating a transaction containing at least 1 ether and calling the **`claimName()`** function

`claimName(“bob”)`

`1.0 eth` *signed, 0x931D3877…*

“Alice”	0xde0b295669a9fd93d5f…
“Bob’s program”	0x2212D359CF1c5454Ae9…
“a message”	0x721E221531b7bC98DB2…
“ethereum.org”	0xC55EdDadEeB47fcDE0B…
“bob”	0x931D387731bBbC988B3…

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {

   mapping (string => address) public registry;

   constructor() {}

   function claimName(string name) public payable {
       require(msg.value >= 1 ether);
       if (registry[name] == address(0)) {
           registry[name] = msg.sender;
       }
   }

   function ownerOf(string name) public view {
       return registry[name];
   }
}
```

If the function completes without errors, any updates to the state variables are **stored** into the contract’s persistent memory and later **committed** on the blockchain (if the transaction is eventually included in a block).

Smart contracts on Ethereum: a basic example

```
contract NameRegistry {
   mapping (string => address) public registry;
   constructor() {}
   function claimName(string name) public payable {
       require(msg.value >= 1 ether);
       if (registry[name] == address(0)) {
           registry[name] = msg.sender;
       }
   }
   function ownerOf(string name) public view {
       return registry[name];
   }
}
```

If a `require()` condition is not met, the transaction **reverts** and any updates to the contract state are rolled back (not persisted)

Here, if Bob does not transfer enough ether along with the transaction he cannot claim the name.

Smart contracts on Ethereum: a basic example

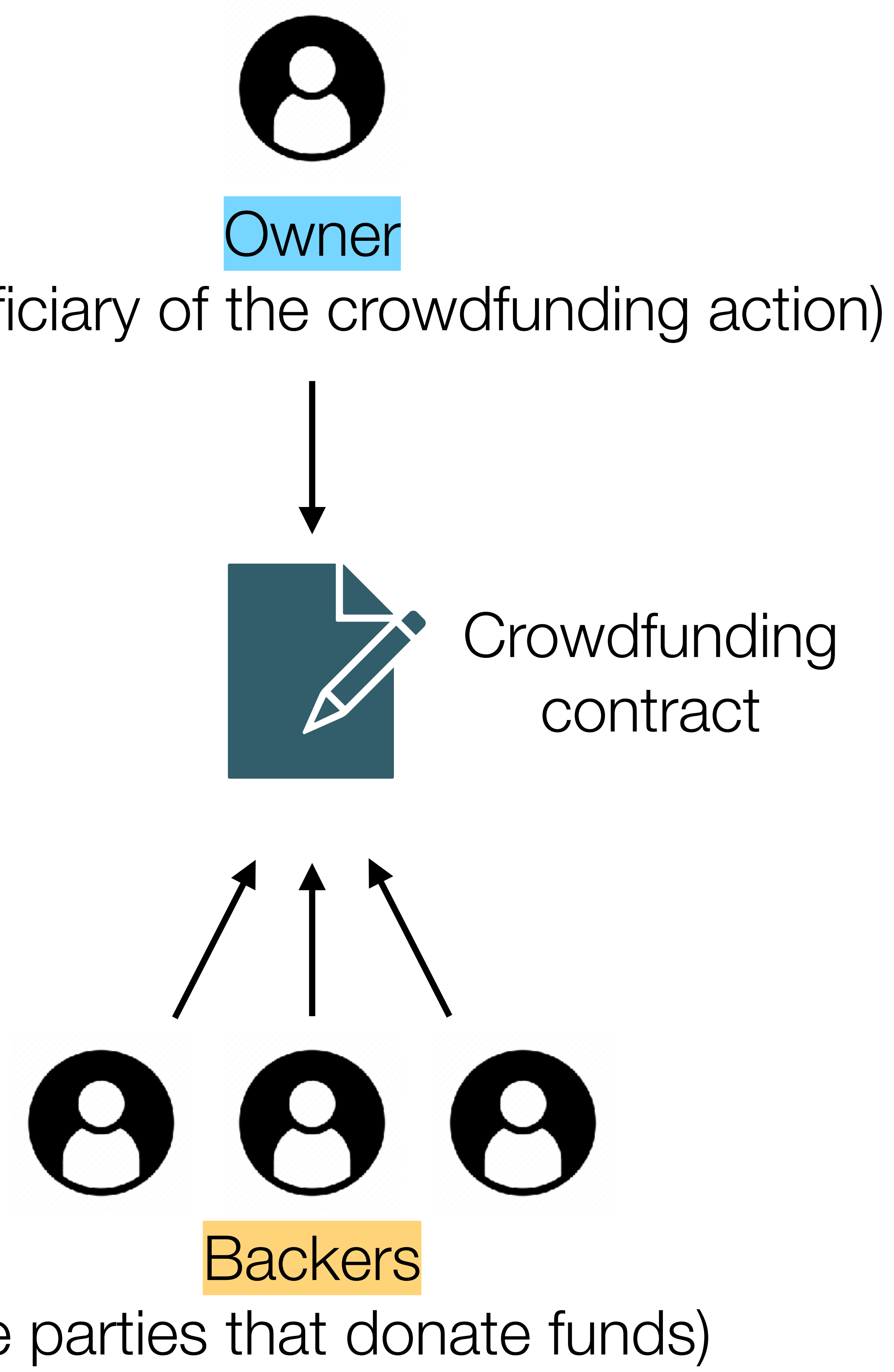
```
contract NameRegistry {

   mapping (string => address) public registry;

   constructor() {}

   function claimName(string name) public payable {
       require(msg.value >= 1 ether);
       if (registry[name] == address(0)) {
           registry[name] = msg.sender;
       }
   }

   function ownerOf(string name) public view {
       return registry[name];
   }
}
```

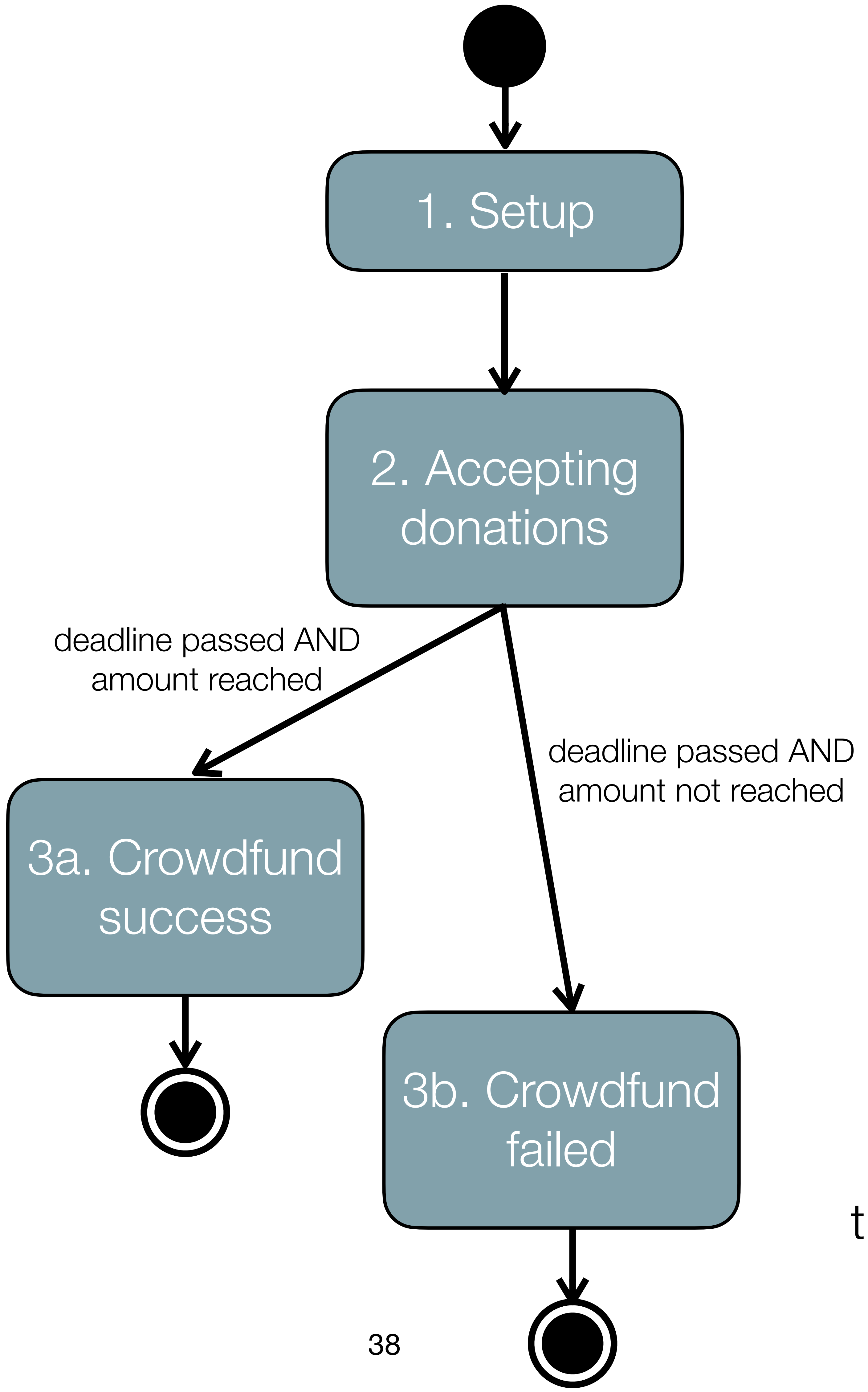
Anyone can lookup ownership of names by calling the `ownerOf()` function.

Since the function is read-only (marked as **`view`**), it can also be called locally by a client without creating a transaction and without broadcasting it to the network.

Remix demo https://remix.ethereum.org/

A more complete example: a crowdfunding contract


Owner
(the beneficiary of the crowdfunding action)
Crowdfunding
contract
Backers
(the parties that donate funds)


1. Setup
2. Accepting donations
deadline passed AND amount reached
deadline passed AND amount not reached
3a. Crowdfund success
3b. Crowdfund failed

Step 1: the owner creates the contract, stating target amount + funding deadline (which **cannot be changed** afterwards)

Step 2: backers can donate money (**deposit** funds into the contract) IF the funding deadline has not yet passed

Step 3a (crowdfunding successful): the owner can claim the funds (**withdraw** funds from the contract) IF the funding deadline has passed AND the minimum target amount has been met

Step 3b (crowdfunding failed): backers can reclaim their donations (**withdraw** funds from the contract) IF the funding deadline has passed AND the minimum target amount has **not** been met

Crowdfunding contract: Solidity source code

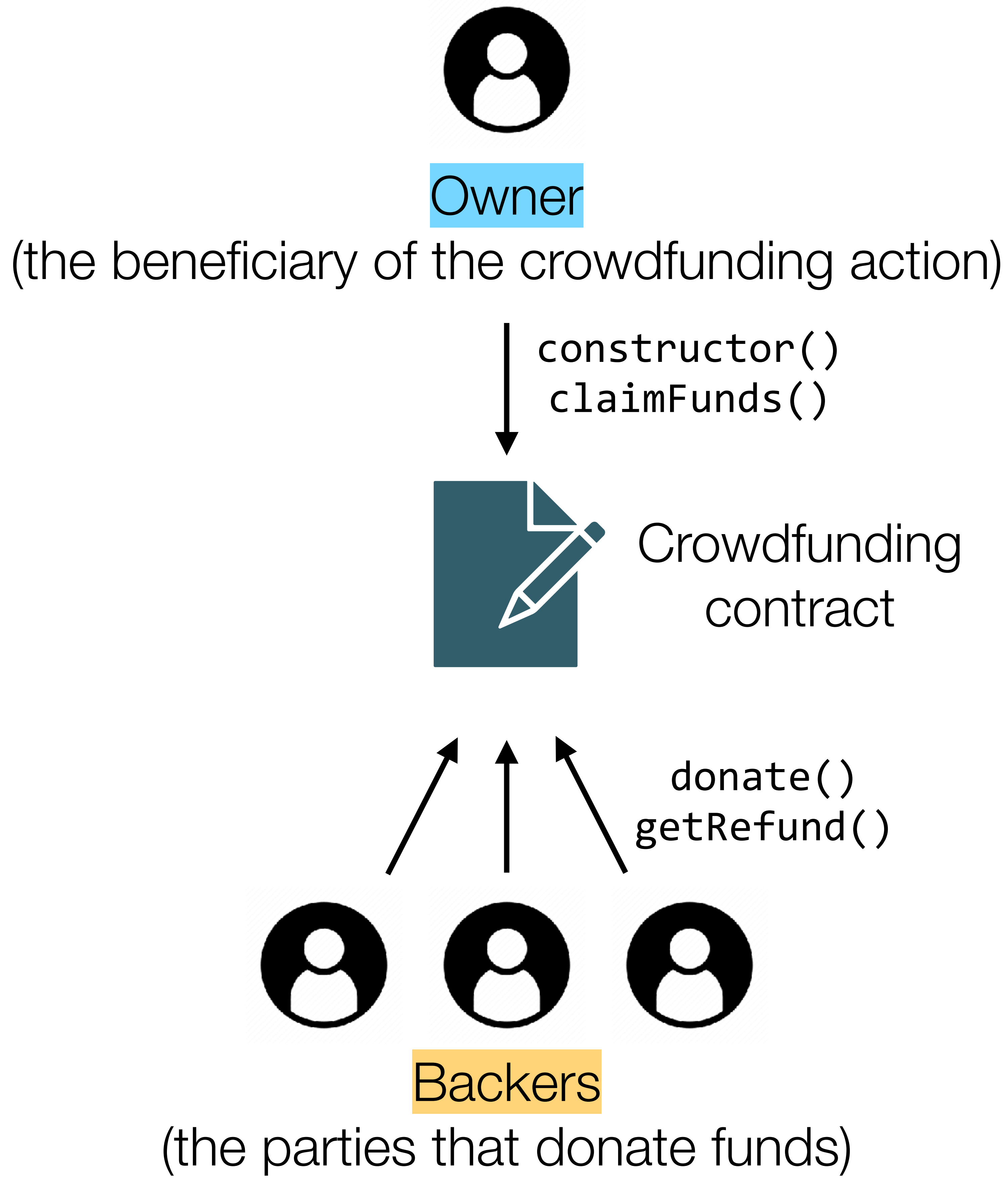
```
contract Crowdfunding {
    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer
    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;
    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }
    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

.sol

(Based on: Ilya Sergey, “The next 700 smart contract languages”, Principles of Blockchain Systems 2021)

Crowdfunding contract: Solidity source code

```
contract Crowdfunding {

    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;
    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```



Owner

(the beneficiary of the crowdfunding action)

constructor()
claimFunds()

Crowdfunding contract

donate()
getRefund()

Backers

(the parties that donate funds)

Crowdfunding contract: Solidity source code

```
contract Crowdfunding {

    address public owner;    // the beneficiary address
    uint256 public deadline; // campaign deadline in number of days
    uint256 public goal;     // funding goal in ether
    mapping (address => uint256) public backers; // the share of each backer

    constructor(uint256 numberOfDays, uint256 _goal) {
        owner = msg.sender;
        deadline = block.timestamp + (numberOfDays * 1 days);
        goal = _goal;

    }
    function donate() public payable {
        require(block.timestamp < deadline); // before the fundraising deadline
        backers[msg.sender] += msg.value;
    }

    function claimFunds() public {
        require(address(this).balance >= goal); // funding goal met
        require(block.timestamp >= deadline); // after the withdrawal period
        require(msg.sender == owner);
        payable(msg.sender).transfer(address(this).balance);
    }
    function getRefund() public {
        require(address(this).balance < goal); // campaign failed: goal not met
        require(block.timestamp >= deadline); // in the withdrawal period
        uint256 donation = backers[msg.sender];
        backers[msg.sender] = 0;
        payable(msg.sender).transfer(donation);
    }
}
```

Instructions to deposit and withdraw money (ether)

Privacy on the blockchain

- You can't store data privately on a public blockchain.
- All Ethereum transaction inputs and stored contract state are public!
- How to cope?
  - Store only encrypted data (and don't put the decryption key on-chain)
  - Only store *commitments* to data (cryptographic hashes) on the blockchain, and store the data "off-chain". Anyone with access to the data can then verify that this data was committed to "on-chain".
  - Advanced: use "zero-knowledge proofs" (e.g. SNARKs) to prove control over data with certain properties, without revealing the data itself to the contract.

*This does **not** work!*

```
contract Vault {
  bool public locked;
  bytes32 private password;

  constructor(bytes32 _password) {
    locked = true;
    password = _password;
  }
  function unlock(bytes32 _password) public {
    if (password == _password) {
      locked = false;
    }
  }
}
```

In any Ethereum client:

```
// get the data stored in ‘password’:
await web3.eth.getStorageAt(contractAddress, 1)
```

(Example from coinmonks, medium.com)

Decentralized Applications (Dapps)

Decentralized applications: what and why?

- **Decentralized applications (dapps)** are web applications backed by smart contracts
  - To achieve **transparency** (publish the core application logic on a blockchain, immutable and verifiable by anyone)
  - To resist **censorship** (avoid a single point of control)
  - To improve **reliability** (avoid a single point of failure)

Decentralized applications: examples

MAKER

Decentralized autonomous organizations (DAOs)

Compound

Decentralized lending and borrowing protocol

UNISWAP

Decentralized exchanges
Atomic token swaps

AUGUR

Decentralized prediction markets & betting platforms

AXIE INFINITY

“Play-to-earn” games

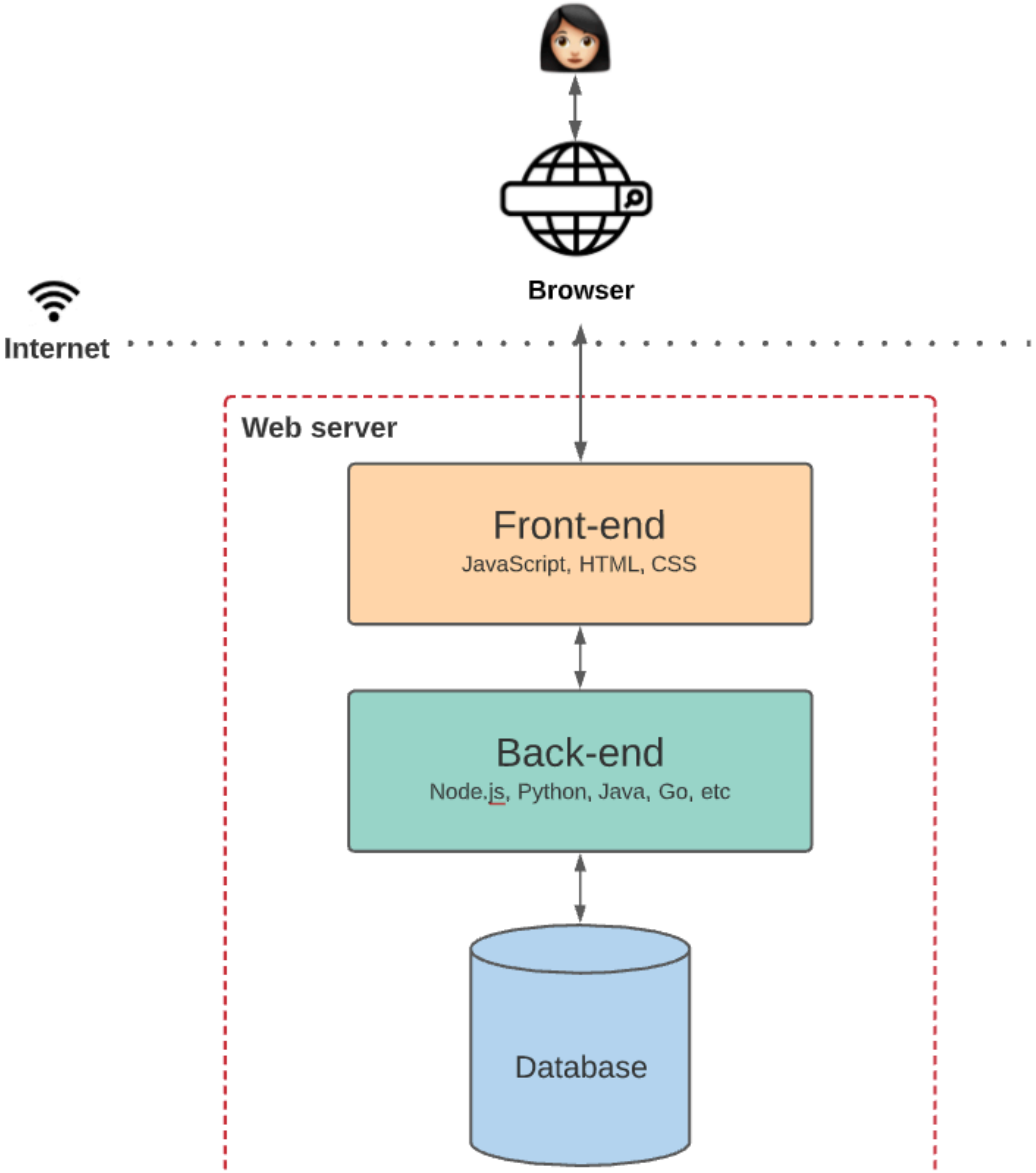
WeiFund

WEIFUND IS
DECENTRALIZED
CROWD-FUNDING

Decentralized crowd-funding

Traditional Web application architecture

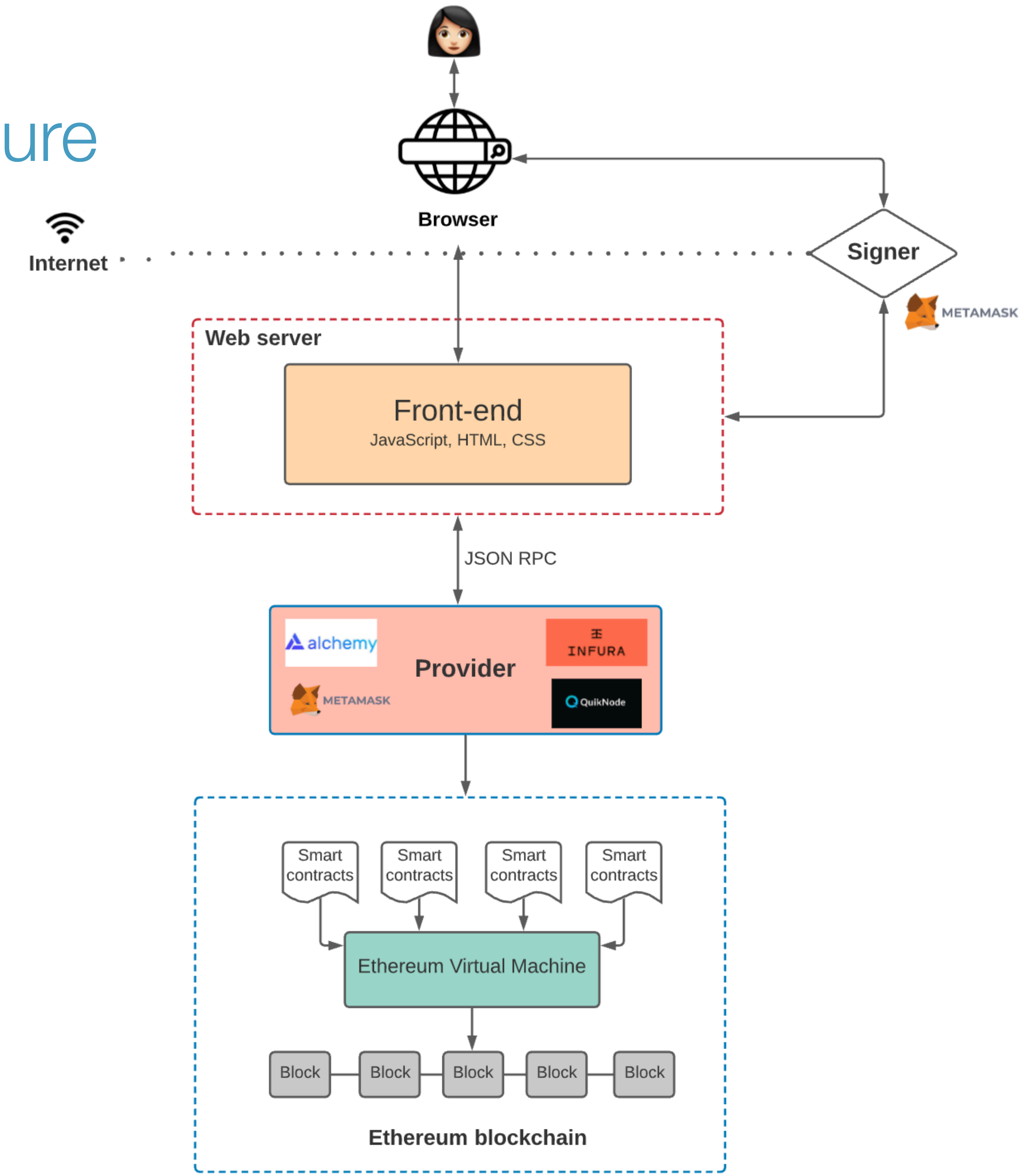
- Following a standard “3-tier” architecture:
- **Front-end**: code that runs in the browser (or on a mobile app), mostly UI logic
- **Back-end**: code that runs on a web server, focus on business logic
- **Database**: persists the application state
- It is common for the application to define the user’s identity and to store username and password in the database. The user **does not control** their identity.



(Source: P. Kasireddy, “The Architecture of a Web 3.0 application”, Medium.com: https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application)

Decentralized Web application architecture

- **Front-end**: largely unchanged (mostly UI logic)
- **Back-end**: (part of) the application logic is implemented as a smart contract and published on the blockchain
- **Database?** The state of the smart contract is persisted on the blockchain (replicated across all validator nodes)
- **Node-as-a-Service Provider**: offers a REST API to relay requests from browsers or mobile apps to peers in the blockchain network.
- **Signer**: for any user action that results in an update to the smart contract, a **signature** is needed from the user. This task typically delegated to a wallet that securely stores the user's keys. The **user retains control** over their keys (they are *not* stored or controlled by the application).



(Source: P. Kasireddy, "The Architecture of a Web 3.0 application", Medium.com: https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application)

Common Dapp “dev stack” options

- **Front-end** libraries

  web3.js ethers.js

- **Frameworks**

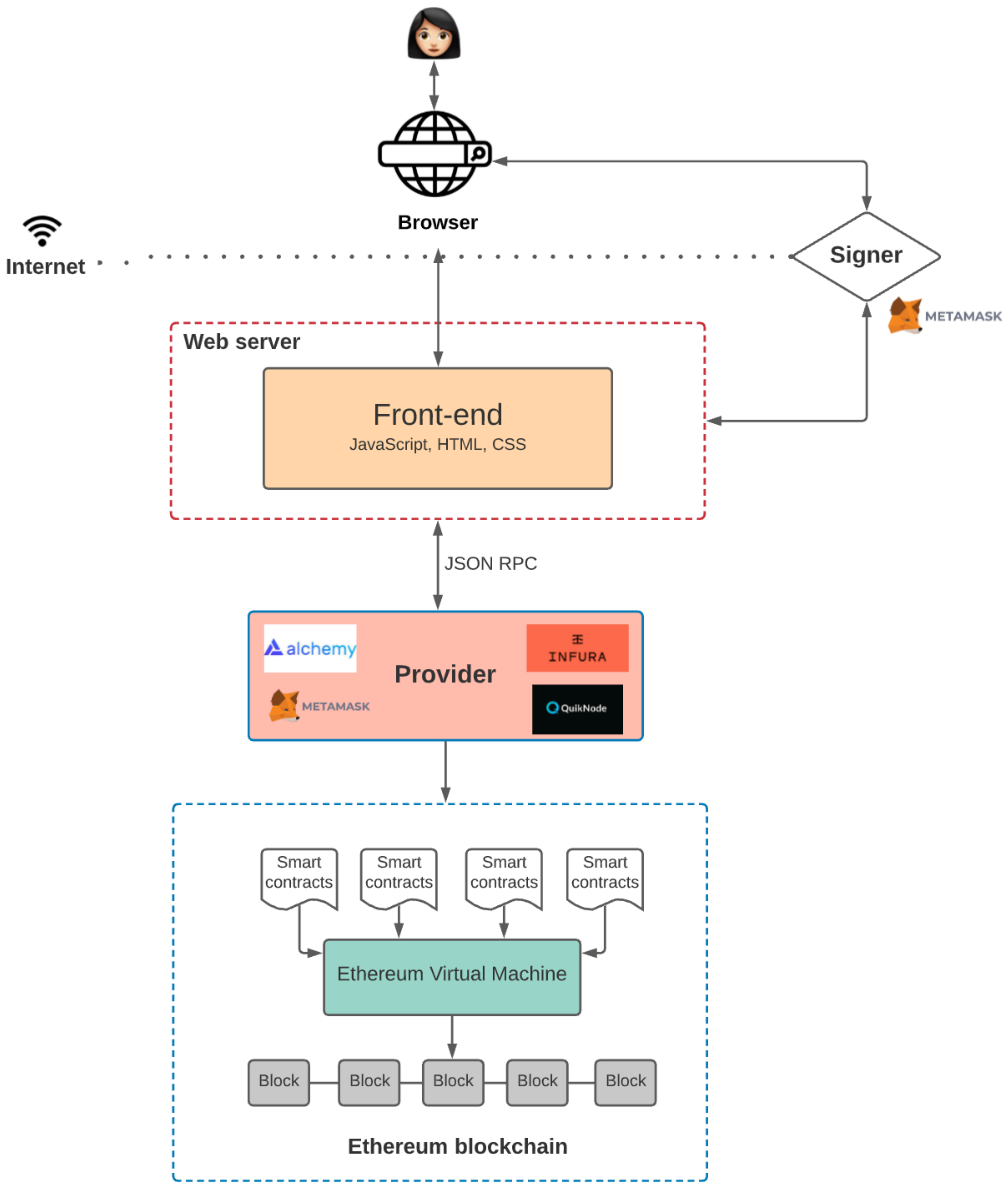
  Hardhat TRUFFLE

- **NaaS Providers**

  alchemy INFURA

- **Signers**

  METAMASK web3auth



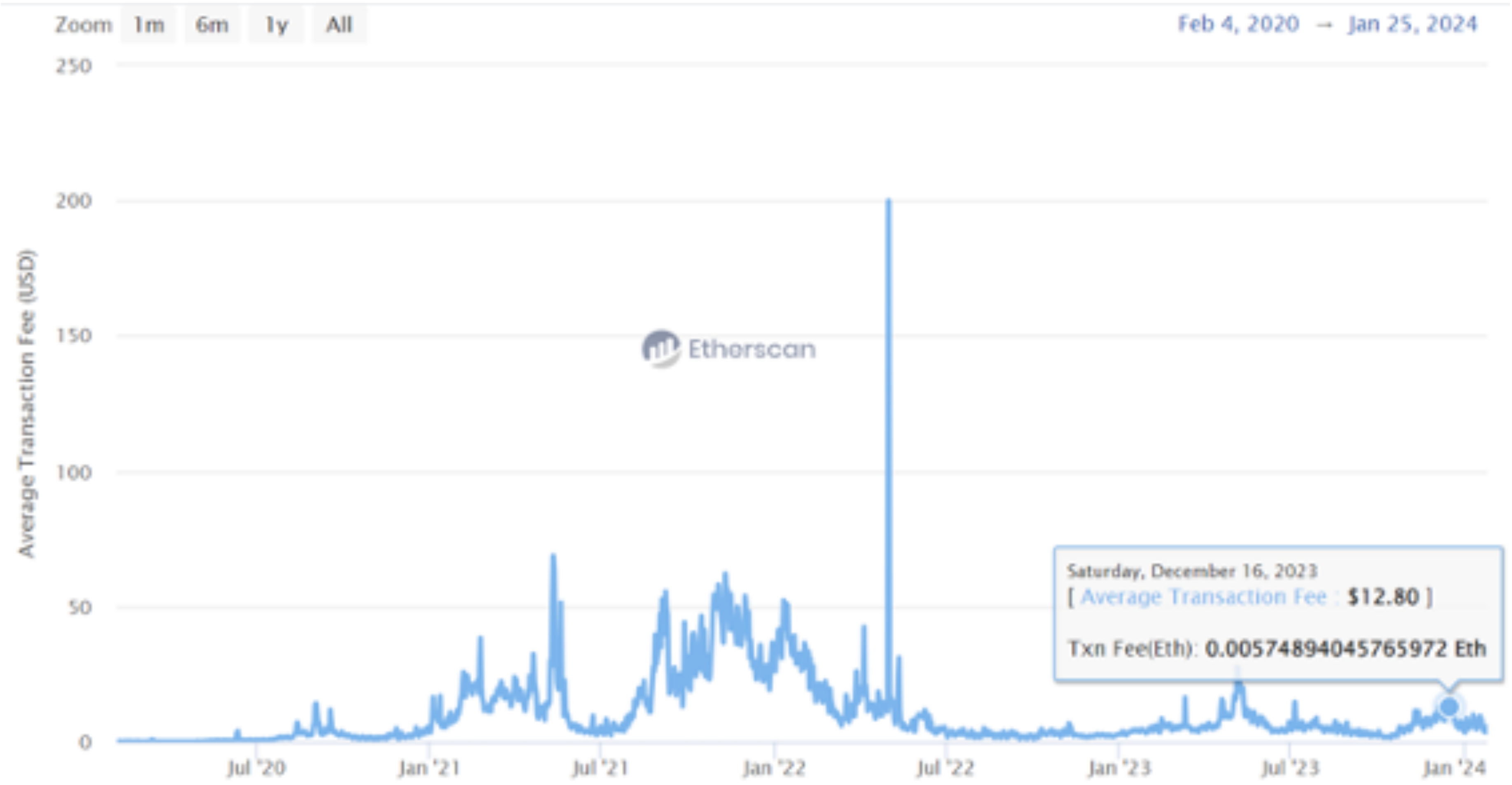
(Source: P. Kasireddy, “The Architecture of a Web 3.0 application”, Medium.com: https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application)

Challenges, trends & advice

Ethereum has challenges

- Can be expensive to use (> $10 in transaction fees is not uncommon)
- Slow (~10-14 transactions per second)
- Bugs in contracts can be fatal

"On the layer 1 Ethereum blockchain, high demand leads to slower transactions and nonviable gas prices." [1]



[1] https://ethereum.org/en/developers/docs/scaling

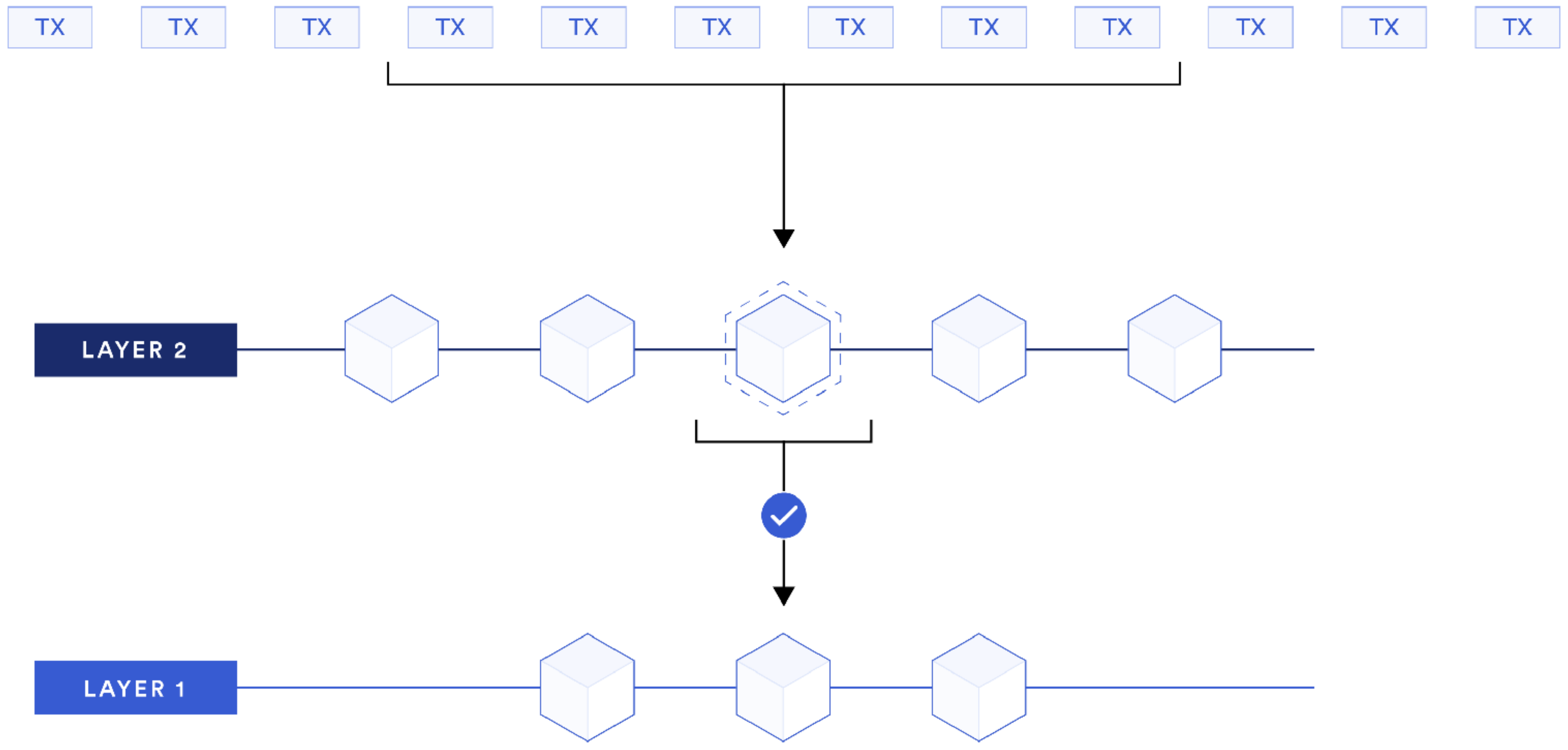
“Layer 2” scaling solutions (a.k.a. “rollups”)

- Key idea: batch many “Layer 2” (L2) transactions into a single combined transaction stored on “Layer 1” (L1)

- Offer a way for anyone to verify that the batch of L2 transactions was correctly executed

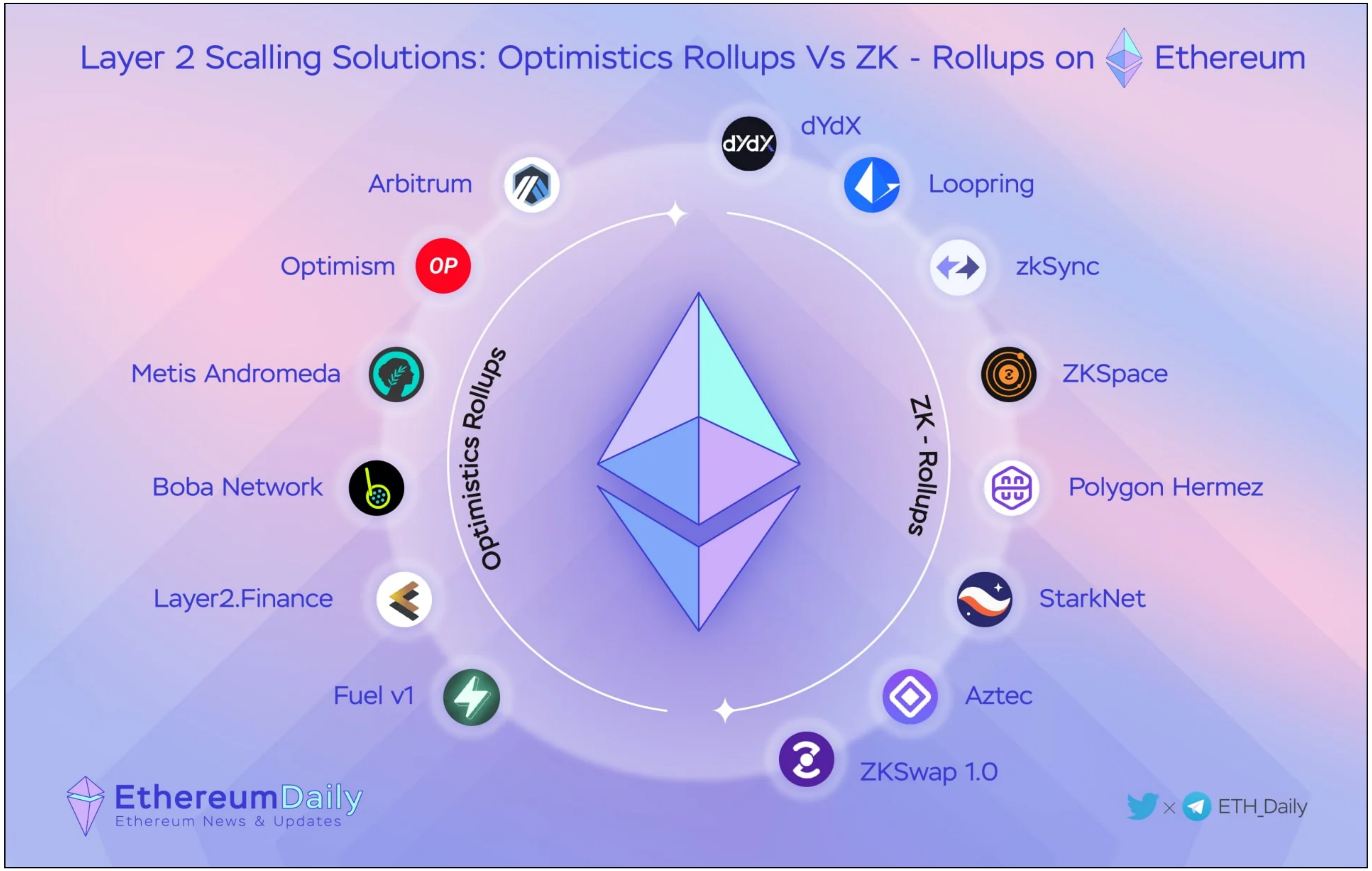
    - “fraud proofs” => optimistic rollups

    - “zero-knowledge proofs” => zk-rollups



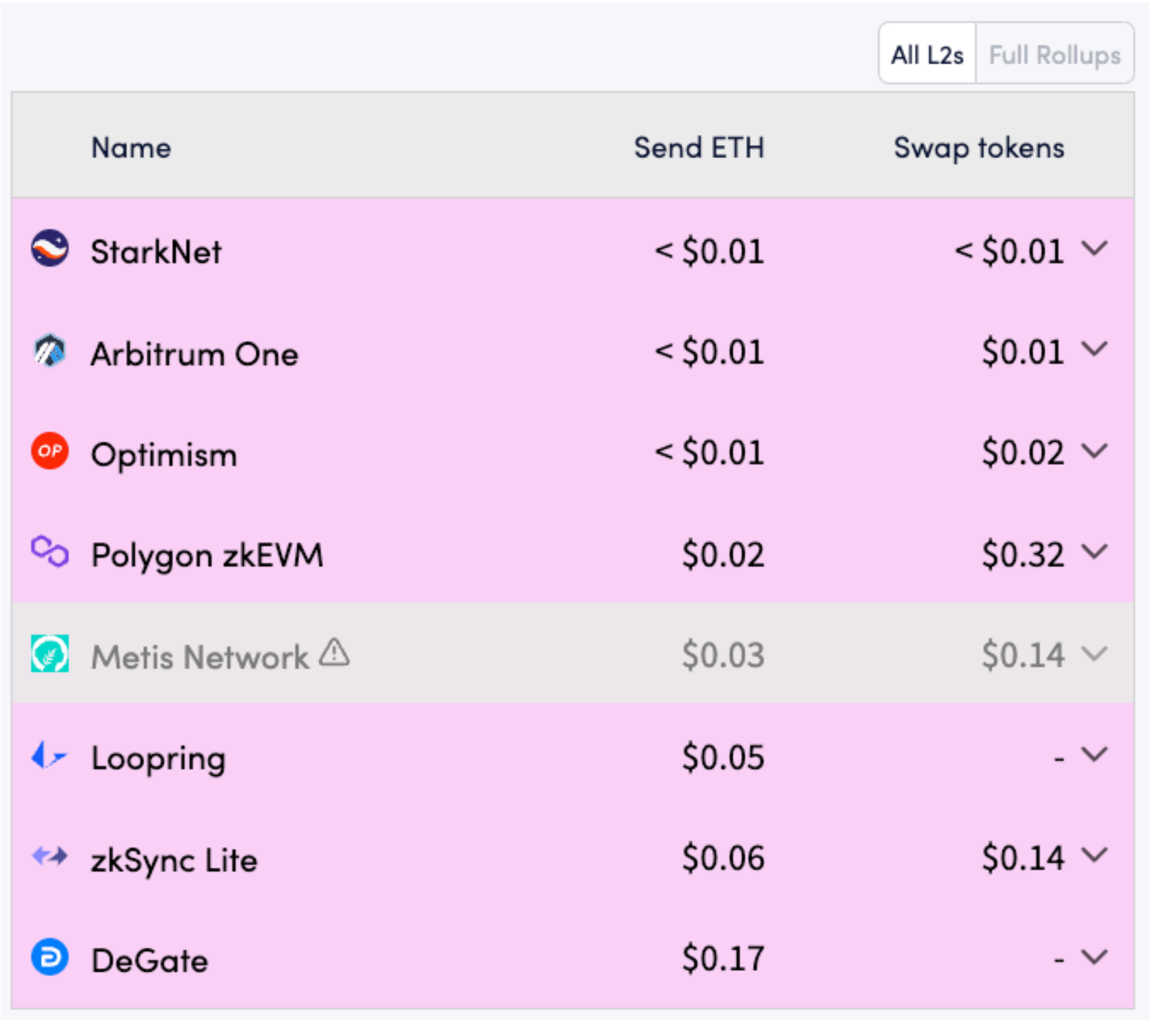
(Source: Chainlink)

“Layer 2” scaling solutions: landscape



“Layer 2” scaling solutions: benefits

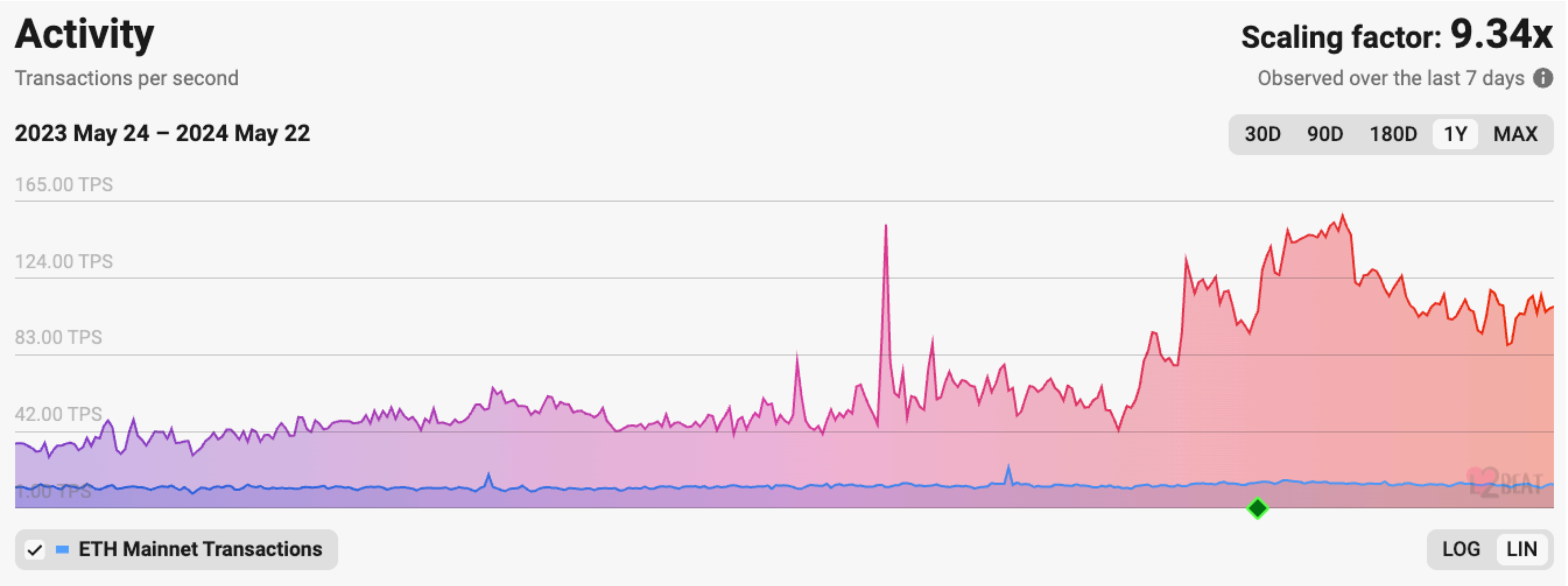
- **Lower** transaction **fees** (< $0.01 / tx)


All L2s
Full Rollups

Name	Send ETH	Swap tokens
StarkNet	< $0.01	< $0.01
Arbitrum One	< $0.01	$0.01
Optimism	< $0.01	$0.02
Polygon zkEVM	$0.02	$0.32
Metis Network ⚠	$0.03	$0.14
Loopring	$0.05	-
zkSync Lite	$0.06	$0.14
DeGate	$0.17	-

(Source: l2fees.info)

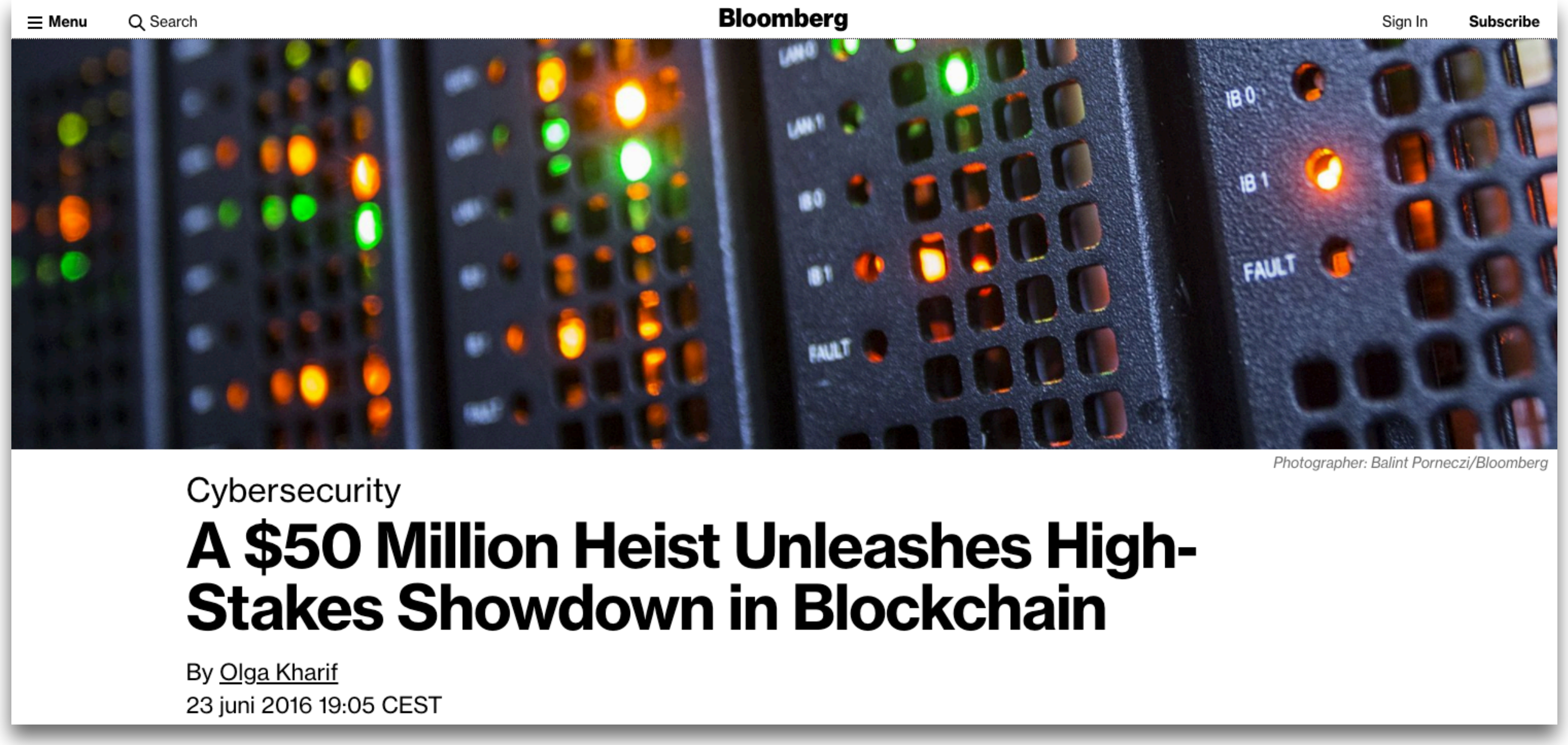
- **Higher** transaction **throughput** (100-1000 tps at ~13min finality)



(Source: L2Beat)

Writing correct smart contracts is a risky business

The DAO Hack (2016)



Photographer: Balint Porneczi/Bloomberg

Cybersecurity

A $50 Million Heist Unleashes High-Stakes Showdown in Blockchain

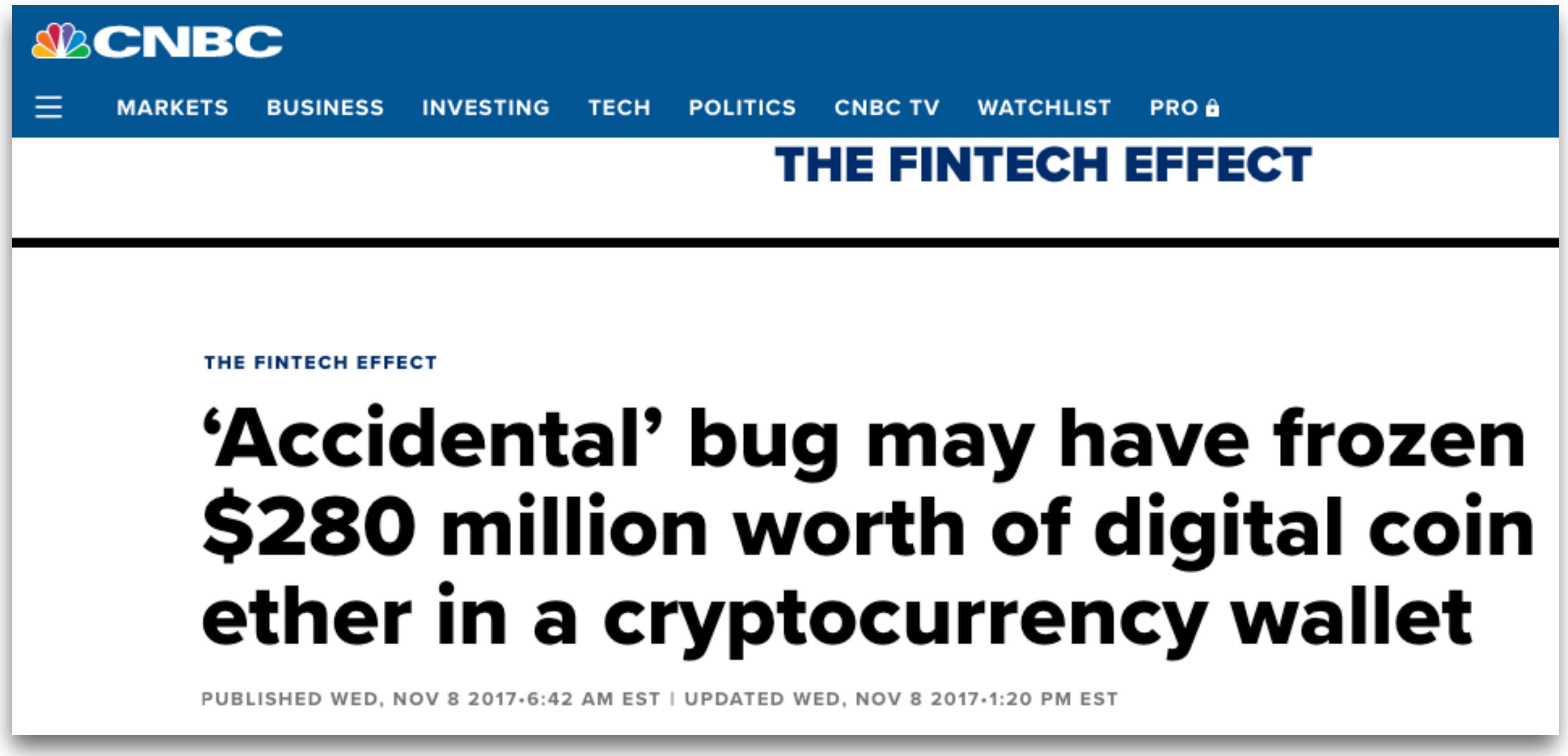
By Olga Kharif

23 juni 2016 19:05 CEST

~$50 million stolen

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

Parity freeze bug (2017)



THE FINTECH EFFECT

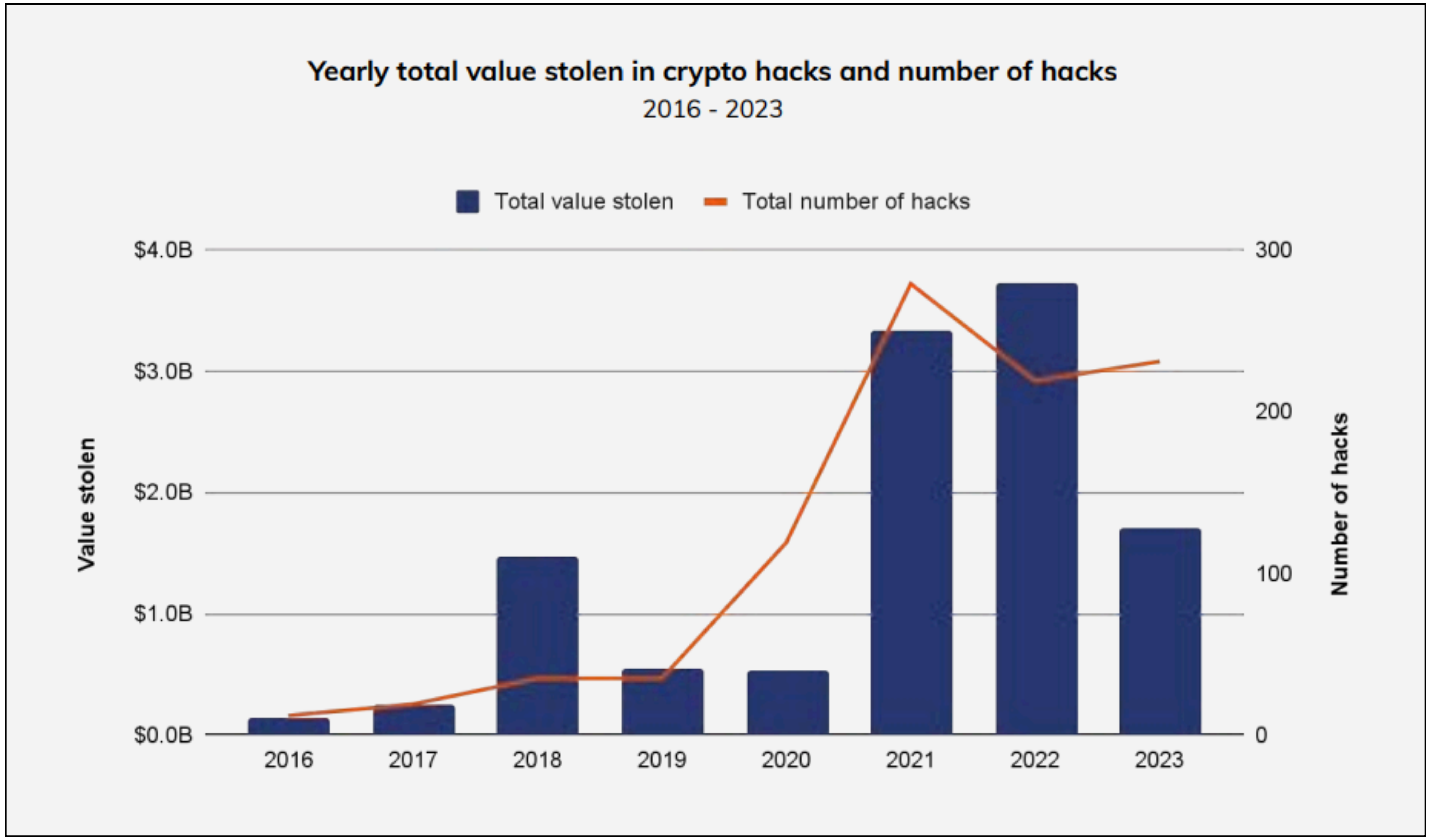
‘Accidental’ bug may have frozen $280 million worth of digital coin ether in a cryptocurrency wallet

PUBLISHED WED, NOV 8 2017•6:42 AM EST | UPDATED WED, NOV 8 2017•1:20 PM EST

~$280 million accidentally frozen

cause: forgot to initialize field in constructor

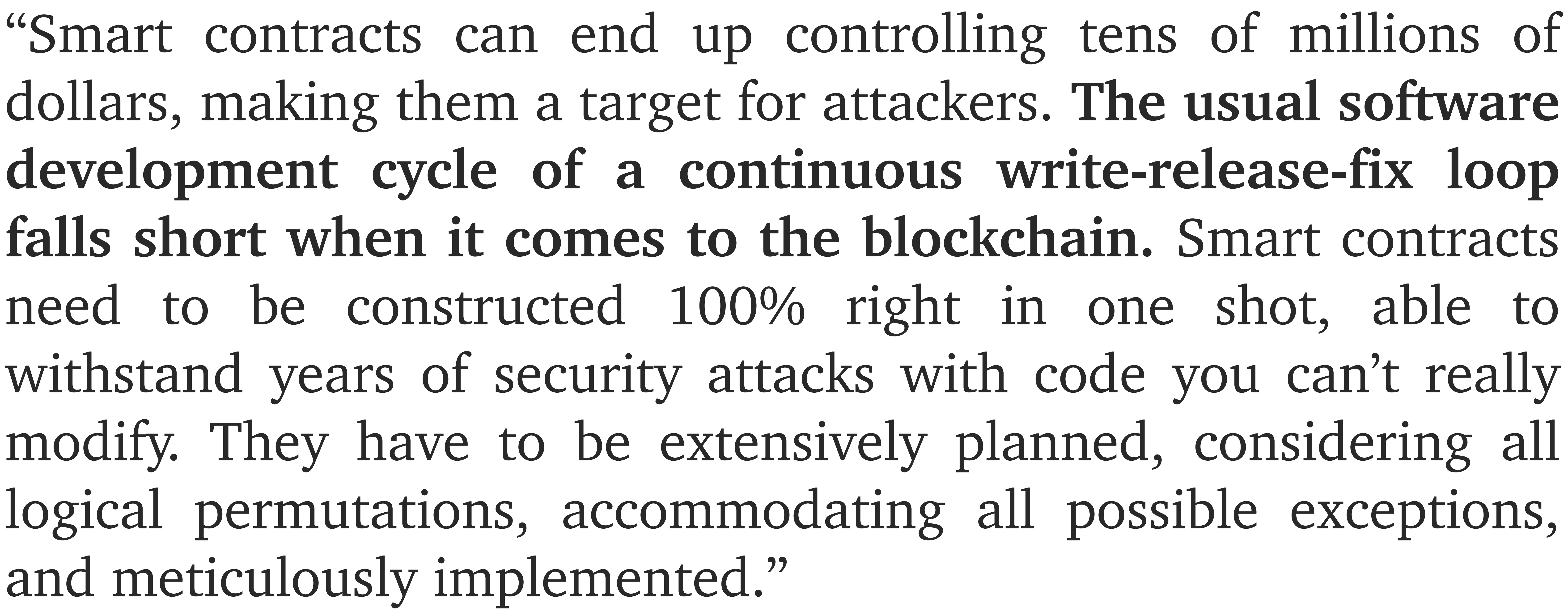
Writing correct smart contracts is a risky business



Funds Stolen from Crypto Platforms Fall More Than 50% in 2023, but Hacking Remains a Significant Threat as Number of Incidents Rises

(Source: Chainalysis, Crypto Crime Report 2024)

Smart contract development is **not** like standard web development



“Smart contracts can end up controlling tens of millions of dollars, making them a target for attackers. **The usual software development cycle of a continuous write-release-fix loop falls short when it comes to the blockchain.** Smart contracts need to be constructed 100% right in one shot, able to withstand years of security attacks with code you can’t really modify. They have to be extensively planned, considering all logical permutations, accommodating all possible exceptions, and meticulously implemented.”

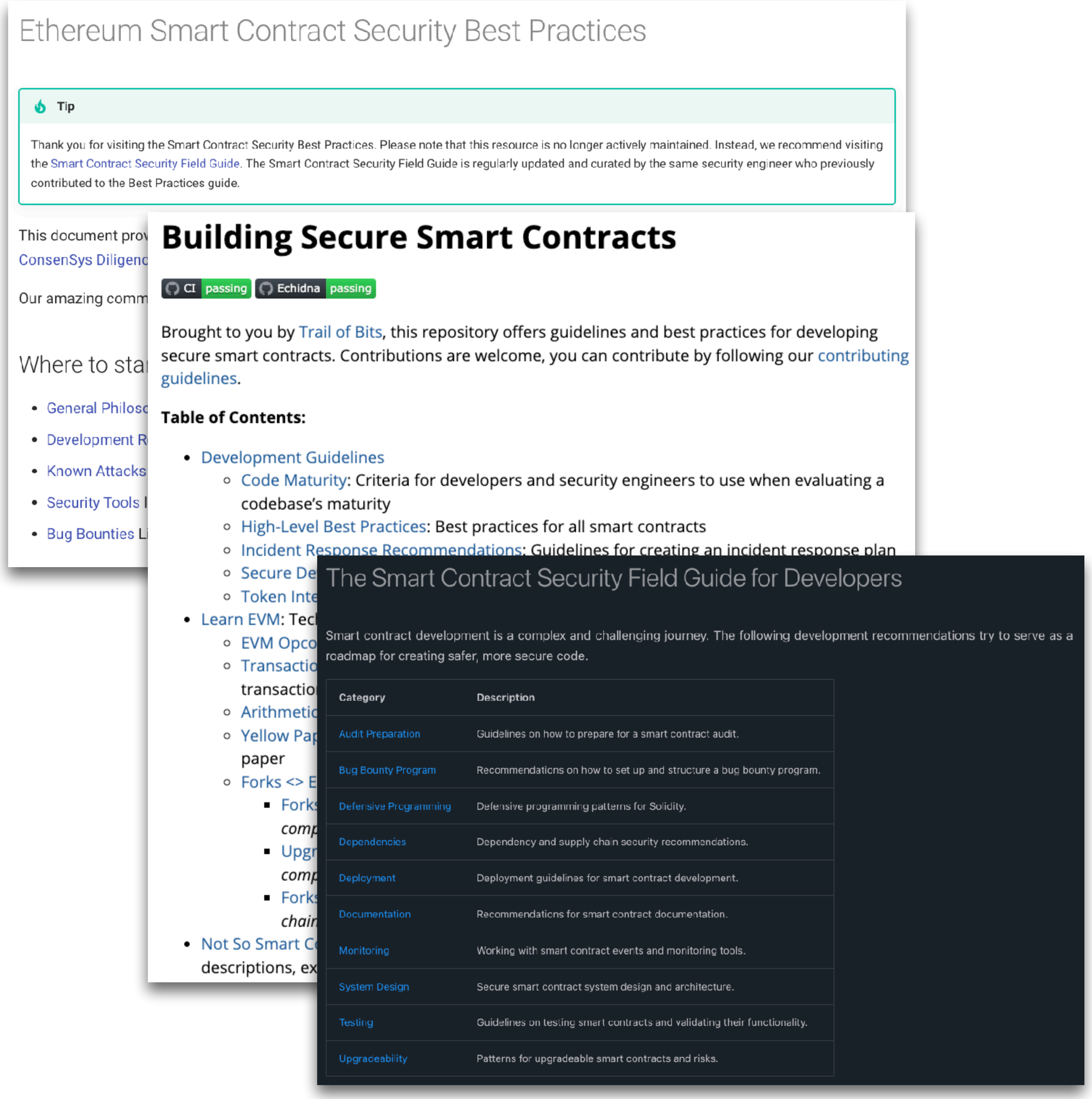
“A short history of smart contract hacks on Ethereum”,
New Alchemy blog, Feb 2018

How to cope?

- Keep on-chain code to an absolute **minimum**
- Use battle-tested **libraries** (e.g. OpenZeppelin)
- Use code **patterns** to enable controlled upgrades (e.g. UUPS proxy pattern)
- Use static analysis **tools** to detect potential vulnerabilities (e.g. Mythril, Slither)
- Conduct code **audits** (well-known companies include Certik, Trail of Bits, Consensys, Dedaub)
- Use dedicated **bug bounty** platforms (e.g. Immunefi, HackenProof)

Excellent resources on securing smart contracts

- Consensys: Ethereum Smart Contract Best Practices
  https://consensys.github.io/smart-contract-best-practices/
- Trail of Bits: Building Secure Contracts
  https://secure-contracts.com/
- Dominik Muhs: Smart Contract Security Field Guide
  https://scsfg.io/



Ethereum Smart Contract Security Best Practices

Tip

Thank you for visiting the Smart Contract Security Best Practices. Please note that this resource is no longer actively maintained. Instead, we recommend visiting the Smart Contract Security Field Guide. The Smart Contract Security Field Guide is regularly updated and curated by the same security engineer who previously contributed to the Best Practices guide.

This document prov
ConsenSys Diligenc

Our amazing comm

Where to sta

- General Philoso
- Development R
- Known Attacks
- Security Tools I
- Bug Bounties L

Building Secure Smart Contracts

CI passing Echidna passing

Brought to you by Trail of Bits, this repository offers guidelines and best practices for developing secure smart contracts. Contributions are welcome, you can contribute by following our contributing guidelines.

Table of Contents:

- Development Guidelines
  - Code Maturity: Criteria for developers and security engineers to use when evaluating a codebase's maturity
  - High-Level Best Practices: Best practices for all smart contracts
  - Incident Response Recommendations: Guidelines for creating an incident response plan
  - Secure De
  - Token Inte
- Learn EVM: Tec
  - EVM Opco
  - Transactio
    transactio
  - Arithmetic
  - Yellow Pap
    paper
  - Forks <> E
    - Forks
      comp
    - Upgr
      comp
    - Forks
      chair
- Not So Smart C
  descriptions, ex

The Smart Contract Security Field Guide for Developers

Smart contract development is a complex and challenging journey. The following development recommendations try to serve as a roadmap for creating safer, more secure code.

Category	Description
Audit Preparation	Guidelines on how to prepare for a smart contract audit.
Bug Bounty Program	Recommendations on how to set up and structure a bug bounty program.
Defensive Programming	Defensive programming patterns for Solidity.
Dependencies	Dependency and supply chain security recommendations.
Deployment	Deployment guidelines for smart contract development.
Documentation	Recommendations for smart contract documentation.
Monitoring	Working with smart contract events and monitoring tools.
System Design	Secure smart contract system design and architecture.
Testing	Guidelines on testing smart contracts and validating their functionality.
Upgradeability	Patterns for upgradeable smart contracts and risks.

Summary

Summary

- **Ethereum**, a “programmable” blockchain
- **Smart contracts**: programs with a bank account
- **Solidity**: the most widely used smart contract programming language
- Decentralized applications (**Dapps**): web apps backed by smart contracts
- Challenges, trends & advice

Where to find more information

- Ethereum official project website: https://ethereum.org/
- Ethereum whitepaper: https://ethereum.org/en/whitepaper/
- Etherscan block explorer: https://etherscan.io/
- Remix, an online IDE and playground for Solidity: https://remix.ethereum.org/
- Solidity by Example: https://solidity-by-example.org/
- OpenZeppelin reusable contracts: https://www.openzeppelin.com/contracts
- Awesome-Ethereum: https://github.com/ttumiel/Awesome-Ethereum

What will you build on Ethereum?

(Image credit: The Defiant)

“as of 2024, the Ethereum ecosystem hosts over 4,000 dapps, 53+ million smart contracts, and 96+ million accounts with an Ether (ETH) balance”

- Moralis, *The Ethereum Ecosystem in 2024*

A gentle introduction to Ethereum and “smart contracts”

Tom Van Cutsem
DistriNet KU Leuven

Questions?
tom.vancutsem@kuleuven.be

tvcutsem.github.io be.linkedin.com/in/tomvc github.com/tvcutsem x.com/tvcutsem @tvcutsem@techhub.social