Ethereum

Jonas Coates, Manuel Klappacher

Universität Salzburg

February 24, 2018

Overview

Einleitung - Idee

"A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises."

Nick Szabo, 1996

Einleitung- Anmerkung

Viele Projekte sind in Zukunft irrelevant, manche möglicherweise Scams.

Tokens werden teilweise nicht gebraucht und Wert der Tokens spiegelt in keinster Weise aktuellen Nutzen wieder.

Einleitung - Warum der ganze Aufwand?

- Kontrolle liegt nicht bei einem (unvertrauenswürdigen?) Unternehmen
- Zensurresistente, demokratische Netzwerke, Apps, Organisationen
- Programmcode kann nicht gestoppt werden, wird garantiert ausgeführt
- KI geschäftsfähig?
- Kostenersparnis gegenber Papierverträgen

- Ethereum dient als Framework
- Die meisten Projekte zu klein um eigenes System aufzubauen

Einleitung - Einige Beispiele

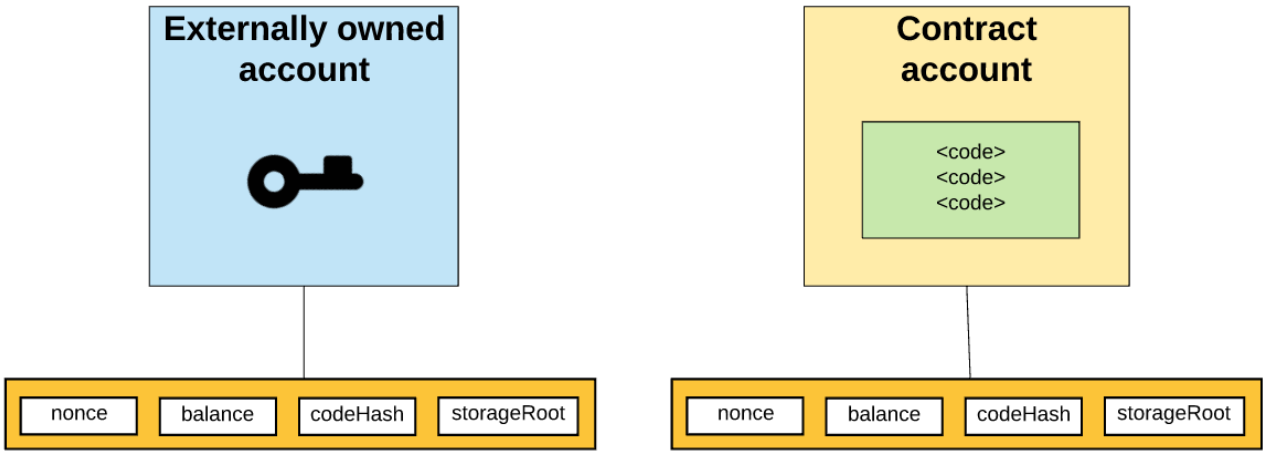
- Prediction Markets (Truthcoin Whitepaper - 2014)
- Cloud Computing (Golem)
- Tokens, Shares (Colored Coins Whitepaper - 2012)
- DAOs - Dezentrale Autonome Organisationen (The Dao† - 2016)
- Soziale Netzwerke (Steem)
- Shops, Märkte, Börsen (Open Bazaar, 0x Protokoll)
- Grundeigentum in BC speichern (Dubai)
- Gesetze in BC speichern (Brasilien)
- Versicherungen (AXA, Etherrisk)
- ICOs - Form der Geldbeschaffung

World-Computer

Ethereum basiert wie Bitcoin auf einer Blockchain.

Es wird der Zustand aller Accounts auf allen Nodes gespeichert, alle Teilnehmer sind sich einig. Accounts speichern nicht nur Guthaben einer Währung, sondern auch Daten und Programmcode. Ein globaler (ineffizienter) Computer.

Account State



- nonce
- balance
- contract code
- storage (empty by default)

Messages and Transactions - Units

Ether:

- 1: wei
- 10^{12}: szabo
- 10^{15}: finney
- 10^{18}: ether

Gas:

- Existiert nur innerhalb EVM.
- Jede Transaktion und Code-Ausführung kostet bestimmte Menge Gas.
- Wird in Wei bezahlt.

Messages and Transactions - Messages

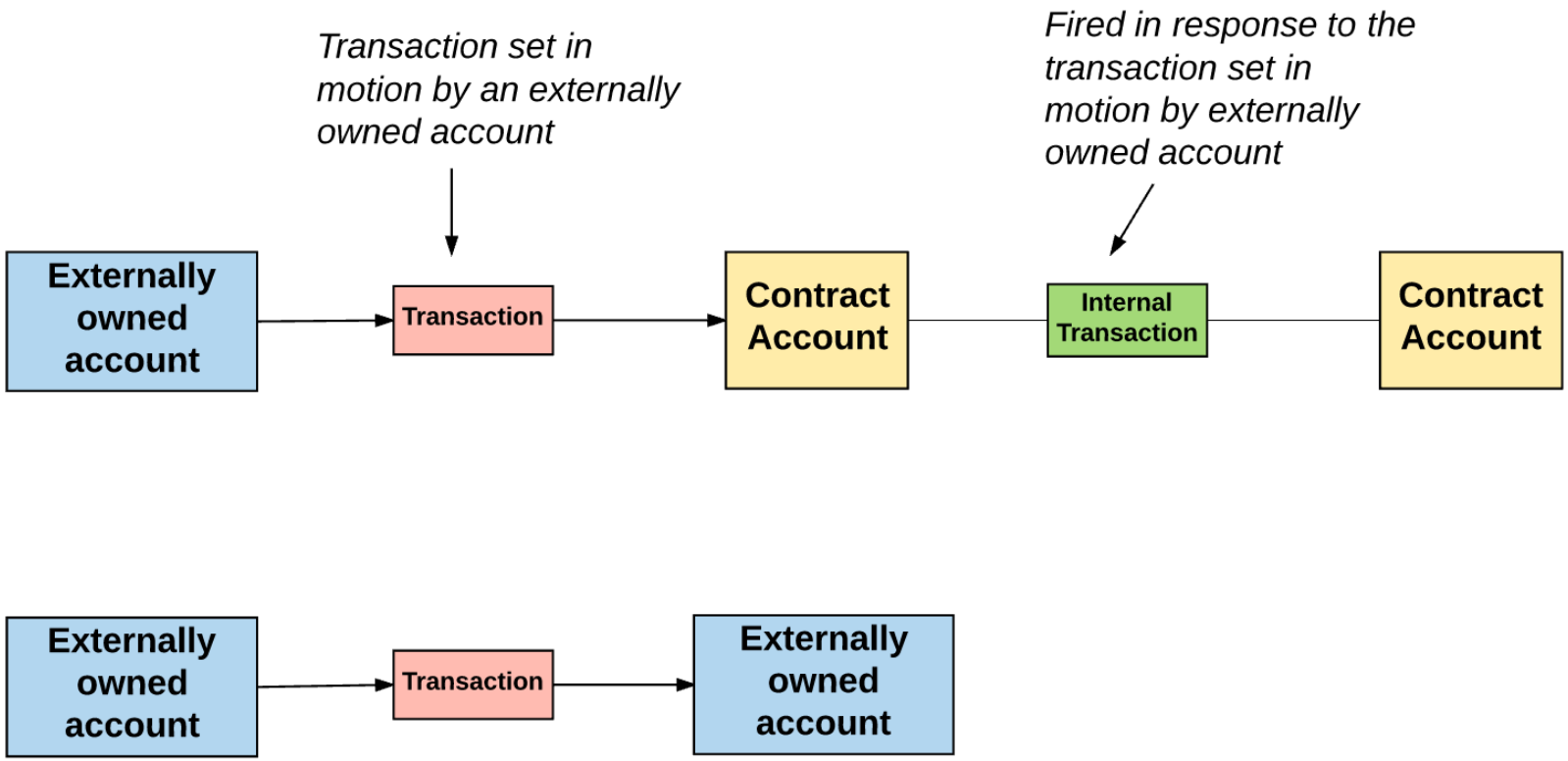
Typen:

- Message Calls
- Contract Creation

Datenfelder:

- Empfänger
- Signatur um Sender zu identifizieren
- Anzahl der Ether, die vom Sender zum Empfänger gesendet werden
- Optional: Datenfeld oder Init
- gasLimit
- gasPrice

Messages and Transactions



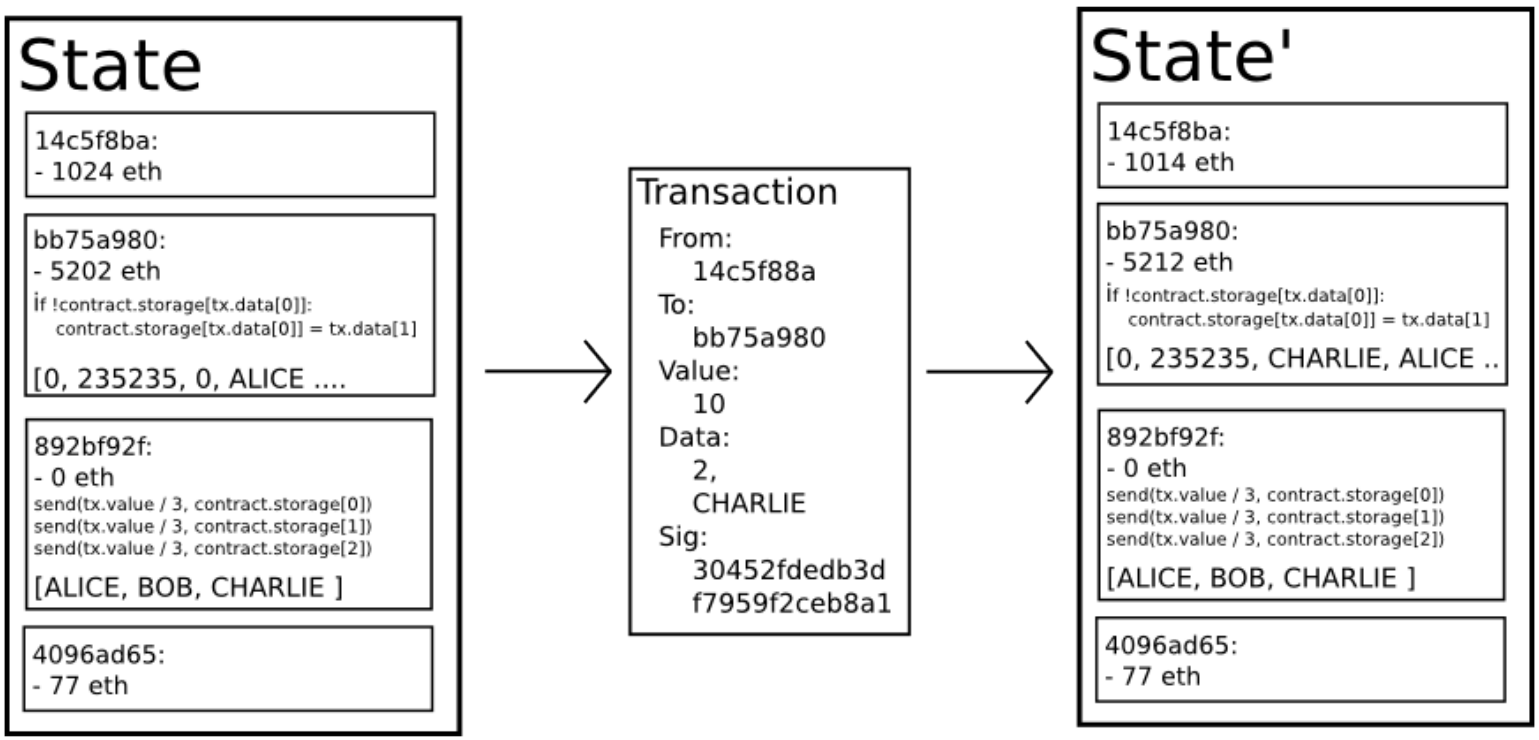
State Transistion

World-State:

$$\sigma_{t+1} = \Upsilon(\sigma_t, T)$$

Account-State:

$$\sigma[a] = (n, b, c, s) \quad ; \ \sigma[a] \in \sigma$$

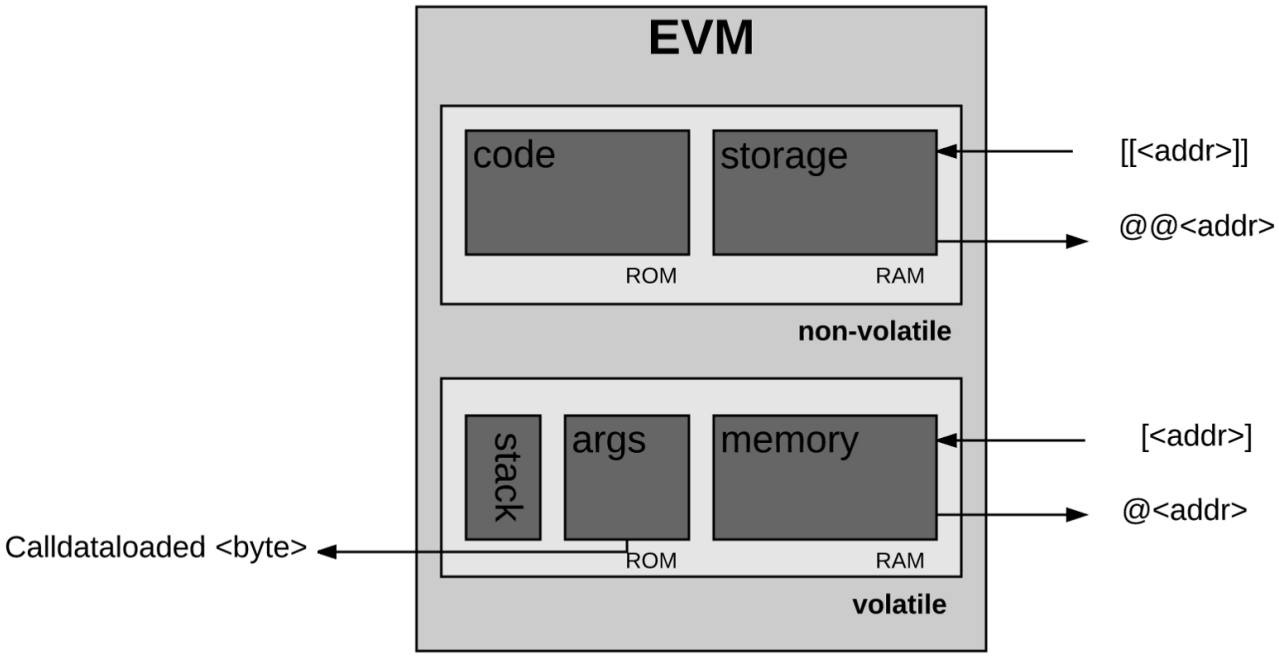


State Transistion Function

APPLY(S,TX) $\rightarrow$ S'

- 1. Checken, ob TX die richtigen Werte hat.
- 2. Gebühr berechnen mit gasLimit * gasPrice
- 3. Das Gas für die TX von gasLimit abziehen
- 4. Die Anzahl Ether der TX vom Sender zum Empfänger übertragen.
- 5. Wenn der Transfer fehlschlägt werden alle Änderungen rückgängig gemacht, bis auf die Gebühr wird an die Miner übergeben.
- 6. Ansonsten wird das übrig gebliebene Gas wieder zurück an den Sender übertragen und die Gebühr an die Miner gezahlt.

EVM



Die Operationen haben Zugriff auf 3 Arten von Speicherstellen:

- Stack: last-in-first-out
- Memory: byte-Array
- Storage: Contracts Long Term Speicher

EVM State

Machine-State:

$$\gamma = (g, pc, m, i, s)$$

- Gas verfügbar
- Program counter
- Memory Inhalt
- Anzahl Wörter in Memory
- Stack Inhalt

1. Hole Instruktion
2. Führe Instruktion aus
3. Ziehe Gas ab
4. Erhöhe PC

Code Beispiele - Grundlagen

```
pragma solidity ^0.4.0;

contract Counter {

    int public count;


    function Counter() public {
        count = 0;
    }

    function add(int x) public {
        count += x;
    }
}
```

Code Beispiele - Grundlagen

```
contract Ownable {

    address public owner;


    function Ownable() {
            owner = msg.sender;
    }

    modifier onlyOwner() {
            require(msg.sender == owner);
            _;
    }

}
```

Code Beispiele - WAPCoin

```
contract WAPCoin is Ownable {
    mapping (address => uint) public balances;

    function give(address receiver, uint amount) public onlyOwner {
        balances[receiver] += amount;
    }

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

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

    uint public sellPrice;
    uint public buyPrice;

    function setPrices(uint newSellPrice, uint newBuyPrice) onlyOwner {
      sellPrice = newSellPrice;
      buyPrice = newBuyPrice;
    }

    function buy() payable returns (uint amount) {
      amount = msg.value / buyPrice
      require(balances[this] >= amount);
      balances[msg.sender] += amount;
      balances[this] -= amount;
      return amount;
    }
    msg.sender.transfer(revenue);
}
```

Vielen Dank für ihre Aufmerksamkeit!

Quellen

- https://github.com/ethereum/wiki/wiki/White-Paper
- https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial
- https://github.com/ethereum/wiki/wiki/Design-Rationale
- https://ethereum.github.io/yellowpaper/paper.pdf
- https://ethereum.org/
- https://www.ethdocs.org/
- https://etherscan.io/
- https://forum.ethereum.org/
- https://ethereum.stackexchange.com/
- https://www.reddit.com/r/ethdev/
- https://www.reddit.com/r/ethereum/
- Alle Bilder von: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369