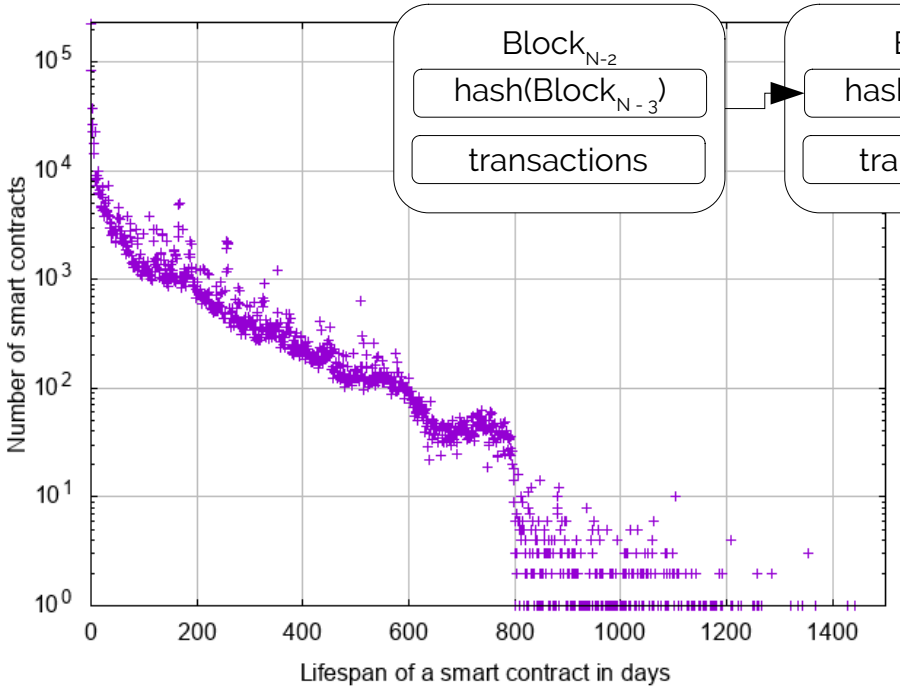
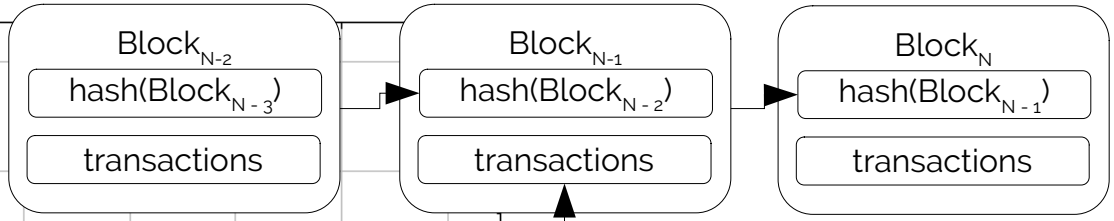
UNIVERSITÄT KOBLENZ · LANDAU

Institute for Software Technology






```solidity
pragma solidity ^0.6.1;

contract TheGame {
    uint256 timeout;
    uint256 timeoutDelta = 604800; // 1 week
    address payable lastBidder;

    constructor() public {
        timeout = block.timestamp + timeoutDelta;
    }

    function bid() payable public {
        require(msg.value == 1000000000); // 1 GWei = 1E-9 ETH
        if (block.timestamp >= timeout) {
            lastBidder.transfer(address(this).balance);
        }
        lastBidder = msg.sender;
        timeout = block.timestamp + timeoutDelta;
    }
}
```

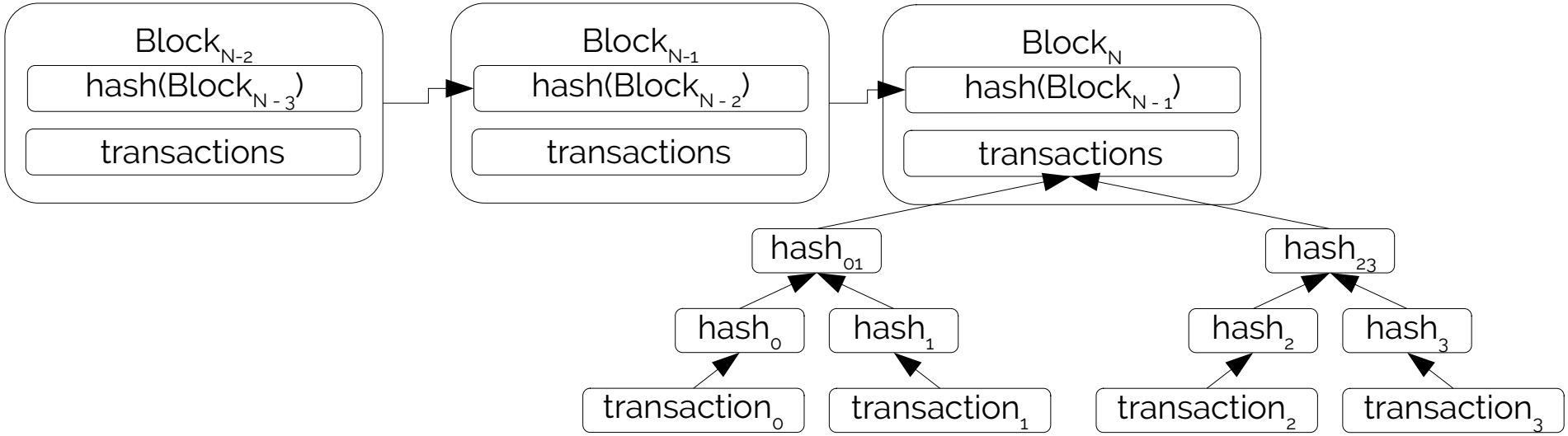
Maintenance of Long-Living Smart Contracts

Matthias Lohr <matthiaslohr@uni-koblenz.de>
Sven Peldszus <speldszus@uni-koblenz.de>

Blockchain in 1 minute

- **Blocks** (= **State**)
- **Blocks linked by cryptographic hashes**
  - If you change one block, you change all successors
- **Rules for changing state**
  - **Static**, e.g. account balance
    - sender_account_balance >= transfer_volume
  - **Dynamic**, with code

    → "Smart Contract"



Smart Contracts in Ethereum

- Bytecode stored on the Blockchain
- Executed by *EVM* (Ethereum Virtual Machine)
- Language *Solidity*
- Transaction Fees depends on Execution Steps (EVM commands)
- Can be used to realize complex decentralized apps
  - Digital Asset Payment
  - Machine2Machine Payments
  - Auction System
  - Much more: https://www.stateofthedapps.com/

Example Smart Contract Game

```
1  pragma solidity ^0.6.1;
2
3  contract TheGame {
4      uint256 timeout;
5      uint256 timeoutDelta = 604800; // 1 week
6      address payable lastBidder;
7
8      constructor() public {
9          timeout = block.timestamp + timeoutDelta;
10     }
11
12     function bid() payable public {
13         require(msg.value == 1000000000); // 1 GWei = 1E-9 ETH
14         if (block.timestamp >= timeout) {
15             lastBidder.transfer(address(this).balance);
16         }
17         lastBidder = msg.sender;
18         timeout = block.timestamp + timeoutDelta;
19     }
20 }
```

Properties of Eth. Smart Contracts

- **Small Size** (currently?)
  - Storing data (code) on the Blockchain is expensive (~50-500$/MB, depending on exchange value)
- **Immutability**
  - Smart Contract identified by “address” (Hash)
  - Code Change → Hash/Address Change → New Contract Deployment
- **Trust**
  - If you checked the contract once, you can trust it.

Smart Contracts have to be ...

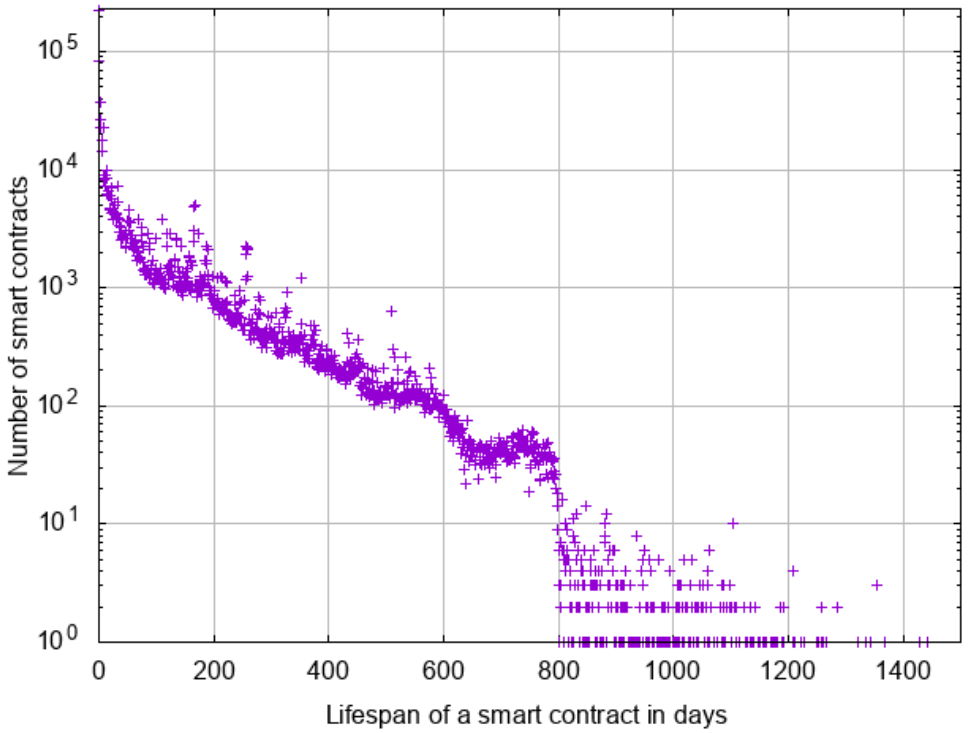
- **Finalized**. You cannot extend it.
- **Bug-Free**. You cannot modify it.
- **Valid forever**. You cannot apply changing external requirements.

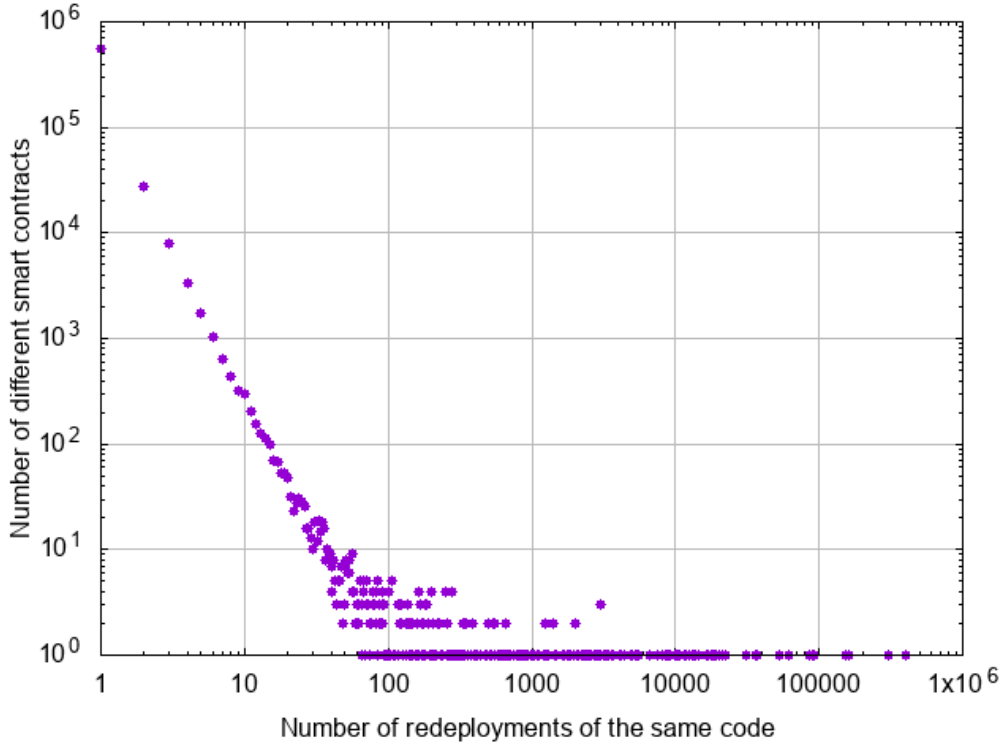
That's great! So we don't need

- Version Management
- Software Engineering
- Care about anything once a Smart Contract is published.

Concept vs. Reality: Insights in Smart Contract Software Development

Longevity of Smart Contracts





Questions

- What is necessary for the development of maintenance-free smart contracts?
  - Which methodologies can be applied?
  - Which restrictions have to be considered?
- **Which update mechanisms (e.g. package managers) of classic software can be applied to smart contracts?**
  - Which modifications have to be done (on both sides)?
  - What are the limitations?
- Should the future research of Software Engineering take smart contracts into account?
  - What are the reasons for or against it?