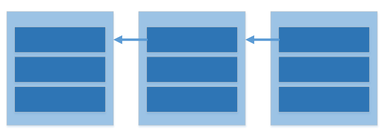
Informatics

Smart Contracts in a DAG Ledger

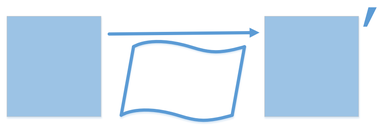
Lukas Hetzenecker, BSc
Software Engineering/Internet Computing

TU Wien Informatics
Institute of Computer Engineering
Research Unit of Automation Systems
Supervisor: Ass.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Monika di Angelo
Contact: lukas@hetzenecker.at

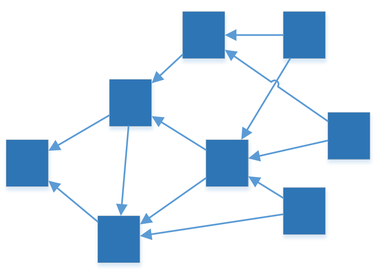
Evolution of Blockchains



„Blockchain 1.0“ is the term for traditional blockchains like Bitcoin. They feature an open, distributed ledger made of blocks linked using cryptographic hash functions. Every transaction gets verified independently by every participant, and creates an irrefutable record in the public ledger. This method frees the participants from the need of any centralized trustworthy arbitrator or third party — one of the main reasons why Satoshi Nakamoto published the Bitcoin protocol in January 2009, following the financial crisis.



„Blockchain 2.0“ extends the limitation of the previous generation that only provided a form of electronic cash, by allowing autonomously executing algorithms via computer code (so-called „smart contracts“) without the need of any third party or middlemen. The execution engine for those contracts can be seen as a state machine, its contracts encode the state transistion functions. But while the term smart contracts would suggest that they are contracts in some legal sense, this is not actually the case, as they lack the elements of offer, acceptance and consideration typically found in them.



„Blockchain 3.0“ are what ledgers seem to be called that no longer arrange the transactions in linear chains, but rather in directed acyclic graphs (DAGs). They claim to get rid of the inherent transaction rate limit of blockchains, and therefore achieve better scalability. Unfortunately this comes with the disadvantage that smart contract engines are more difficult to implement on such ledger structures, as they lack the „total order“ property.

Survey of Smart Contract Plattforms

RQ1: What are currently the most popular smart contract platforms, and what do they have in common?

Survey of smart contract platforms, based on **Top50** cryptocurrencies of coinmarketcap.com

#	Name	SC Engine	SC Language	Remarks
2	Ethereum	EVM	Solidity	
6	EOS	EOS VM	C, C++	WebAssembly based
10	TRON	TVM	Solidity	EVM-compatible
11	Cardano	IELE VM	Plutus, (Solidity)	EVM-compatibility planned
17	Neo	NeoVM	many (.NET, Java,...)	
20	Ethereum Classic	EVM	Solidity	Fork of Ethereum
26	Qtum	EVM	Solidity	Bitcoin+EVM
28	VeChain	EVM + built-in extensions	Solidity	

Main findings:

- EVM is de-facto standard for smart contracts
- EVM is not only used by Ethereum and direct forks (Ethereum Classic), but also by various other projects
- If the Smart Contract Engine is not directly based on EVM, there is often a compatibility layer to support existing Solidity contracts as well
- There is currently not much variety in the smart contract space, but this should change in the future

Ethereum

RQ2: How do smart contract VMs work in detail?

Ethereum VM (EVM) is a quasi turing-complete 256-bit stack-based execution environment for smart contract bytecode. Currently compiler support is limited to smart-contract specific languages, most notably *Solidity*. In a future release, the VM will be based on *WebAssembly* with support for general-purpose languages like *C++* or *Java*.

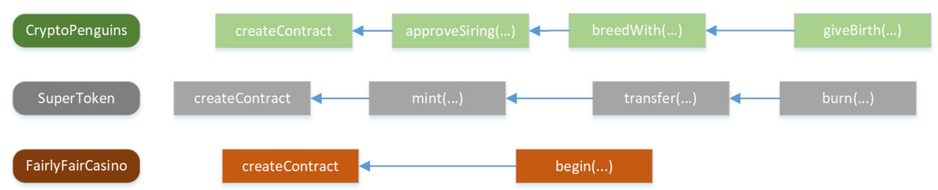
The world state maps adresses to the state of accounts. Accounts can be controlled externally, from users with their private keys, or by contract code. The EVM can execute this bytecode.The following table shows a selection of opcodes implemented by the EVM:

Opcode	Mnemonic	Stack Input	Stack Output	Expression	Notes
01	ADD	a | b	a+b	$a + b$	(u)int256 addition modulo 2**256
80	DUP1	value	value | value	$PUSH(value)$	clones the last value on the stack
20	SHA3	offset | length	hash	$hash = keccak256(memory[offset : offset + length])$	keccak256

The order of transactions inside a block is determined by the miner of the block.

Smart Contracts in a DAG ledger

RQ3: How can the basic requirements of smart contracts be applied in a DAG?



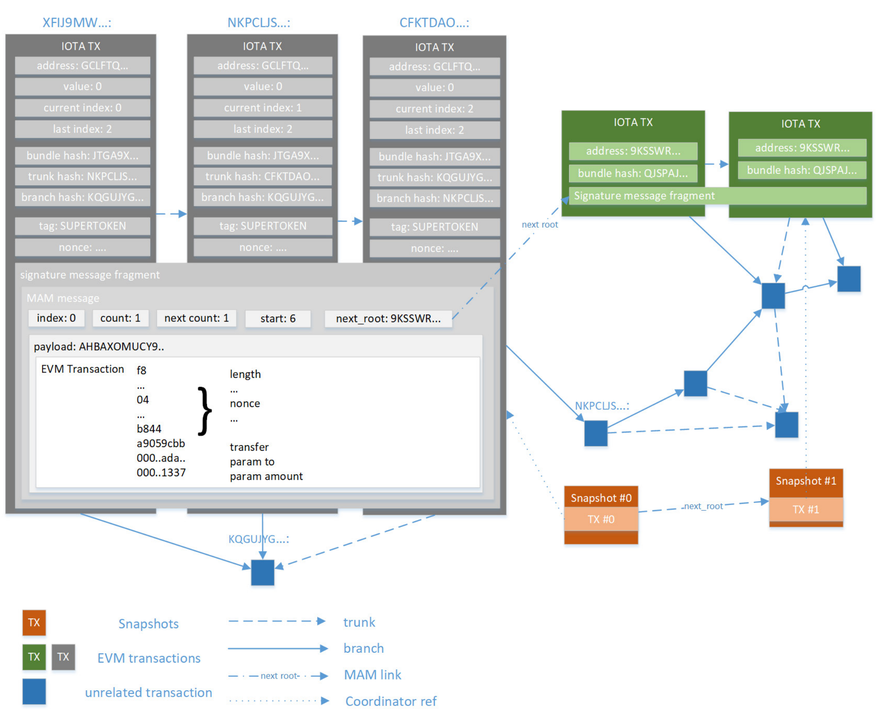
An often talked about solution for solving the scalability problem of Blockchains are **sidechains**. Usually such implementations only use the core chain for **settlement**, and interactions with DApps happpen on a second-layer technology or completely **off-chain**.

In DAG ledger structures, it is possible to build all DApp-chains directly in the core ledger, as this structure allows arbitrary many chains in parallel. This even allows different DApps to have diverging security settings and consensus algorithms.

tanglEVM

We implemented a Proof-of-Concept, that builds a seperate „sidechain“ for each DApp. We have chosen **IOTA's Tangle** as public ledger to piggyback our solution, because it supports feeless zero-value data transactions. The smart contract engine is based on the EVM - or more precicly on the Python implementation **Py-EVM**, as this engine is the de-facto standard for smart contracts.

The main difficulties were the fundamental differences of Ethereum and IOTA. E.g. IOTA uses trytes instead of bytes, so a tedious conversion was needed.
Smart contract invocations need to be mapped to a single user, but IOTA does not support address reuse. Therefore different methods to implement message streams were researched, and the *Masked Authenticated Messaging* module was chosen.



RQ4: How can a consensus be formed in a DAG?

For simplicity we settled for a centralized coordinator-based Proof-of-Authority-type consensus.

Conclusion

We found that it is not the smart contract virtual machine that differentiates the projects. Even in the seldom cases when the VM is not directly based on the EVM, its functionality is in large parts identical: some stack-based execution engine implementing a state machine, that adds to the usual logical and arithmetic opcodes a handful more for cryptographic operations and querying the state of the blockchain.

It is rather the **consensus algorithm**, which is the **major differentiator** between the cryptocurrency projects. Ethereum utilizes a Proof-of-Work (PoW), Neo the voting-based delegated Byzantine Fault Tolerance (dBFT), and EOS and TRON a Delegated Proof-of-Stake (DPoS) algorithm. These can usually generate blocks much faster than the current version of Ethereum, and also solve Ethereum's scalability problem of only achieving 15 TPS.

To resolve this, we discussed a proposal with the goal of splitting off every DApp from Ethereum's main chain, and letting them live in their own chain – so called sidechains. The resulting data structure then no longer resembles a chain of transactions, but rather a graph – precisely, a directed acyclic graph (DAG). Several cryptocurrencies already exist that use a DAG for their ledger structure, but they do not support smart contracts yet. We took a shot at such an implementation with the tanglEVM prototype.

Future work:

- decentralized consensus for Proof of Concept tanglEVM chains
- interoperability between different chains