What is QTUM? An attempt to create a next generation platform to tackle 6 major blockchain problems

Problem	Approach	
Security - Most smart contracts are built on experimental technology, leading to security flaws	QTUM's combination of Bitcoin's battle-tested codebase with Ethereum's Turing-complete flexibility	
Accounts are not an ideal basic data structure – they are less scalable, secure, and anonymous	Account Abstraction Layer	1
Energy usage - Large amounts of energy are spent on first-generation consensus protocol, Proof-of-Work	Working Proof of Stake	2
Governance – Off-chain governance can be a messy affair with vague resolution mechanisms (e.g. Bitcoin scaling debate)	Decentralized Governance Protocol implemented on-chain governance for live QTUM blockchain	3
Mobile-friendliness - Little usability for smart contracts on mobile platforms	QTUM is oriented to bringing smart contracts to mobile and IoT devices	
Smart contract development is difficult to learn and divorced from mainstream software development	QTUM x86VM	4

Source: An introduction to QTUM, April 2018

Brief history of the QTUM blockchain

1. Aug 2016 - Published public version of white paper
2. Oct 2016 – Angel investment of $1m completed
3. Mar 2017 - Released first test network to run EVM virtual machine based on the UTXO model. The popular EVM on BTC Network
4. Mar 2017 – Started global fundraising for $15m
5. Aug 2017 – Released second testnet skynet to implement all the core functions of white paper planning (POS AAL DGP, etc.)
6. Sep 2017 – Released mainnet Ignition, and QT full-node wallet, and coordinated launches on several exchanges at the same time to realize all the core contents of the white paper.
7. Nov 2017 – Released Qtum SPV wallet Electrum, supports SPV mode, and supports hardware wallet and multi-signature
8. Jan 2018 – Released QRC20 Token standard
9. Mar 2018 – Released an internal X86 virtual machine prototype
10. Apr 2018 – Started QtumX program, high performance service for industries
11. May 2018 – First x86vm smart contract, written in C, launched on the testnet

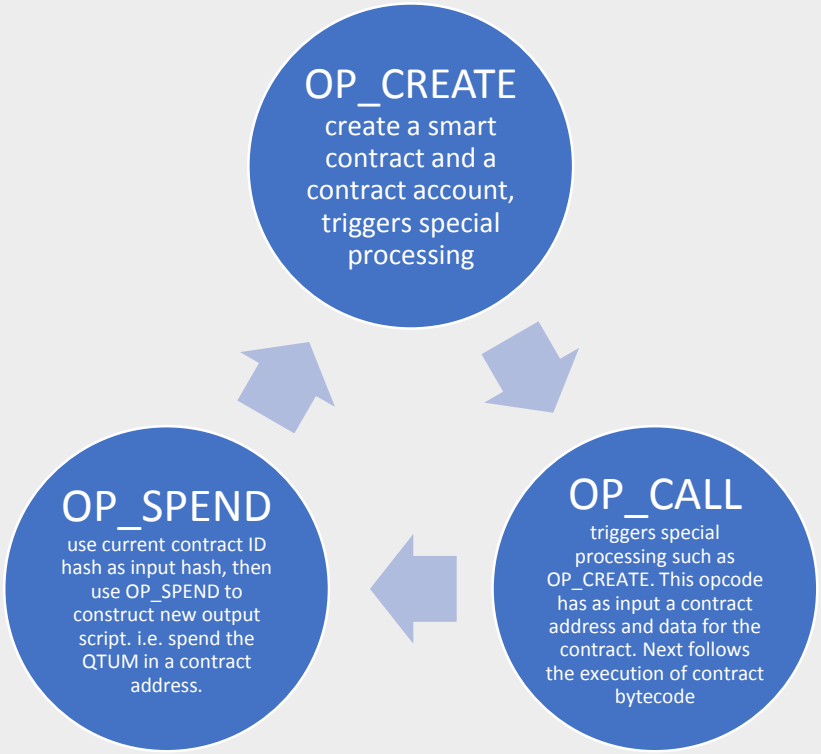
1 QTUM – Account Abstraction Layer

- What is QTUM's **Account Abstraction Layer**?
- A way to have Turing Complete account-based platforms like Ethereum **emulated** on Bitcoin's UTXO-based architecture

- **Why would you want to do that?** Isn't Ethereum's architecture an improvement on Bitcoin's?
  - **Yes, in terms of expressiveness…**
    - Ethereum's account-based platform is Turing complete, Bitcoin's UTXO model is not

- **…but…**
  - **Less scalable.**
    - Accounts cannot handle parallel transactions when being modified (e.g. Bittrex account)
  - **Less secure.**
    - Ethereum does not natively support multsig, Bitcoin does
    - Harder to remedy and reverse double-spend attack
  - **Less anonymous**
    - Difficult to handle multiple input-outputs, new change addresses
  - **No simple payment verification protocol for mobile/ IoT devices**
  - **Difficult to support multiple virtual machines**

3 new OPCODES to Bitcoin script for EVM compatibility



2 QTUM – Live proof-of-stake smart contract platform

The map above shows which countries consume less electricity than the amount consumed by global bitcoin mining

- Proof of Work is infamously wasteful…

Source: https://hackernoon.com/proof-of-work-or-proof-of-waste-9c1710b7f025

2 QTUM – Live proof-of-stake smart contract platform

- That's why many people have suggested Proof-of-Stake as a replacement for Proof-of-Work
  - Idea - Secure the network with the value of the network, rather than making validators waste energy solving artificially difficult hash-puzzles

2 QTUM – Live proof-of-stake smart contract platform

```
pseudo-code:

while(true){
    foreach(utxo in wallet){
        blockTime = currentTime - currentTime % 16
        posDifficulty = difficulty * utxo.value
        hash = hash(previousStakeModifier << utxo.time << utxo.hash << utxo.n << blockTime)
        if(hash < posDifficulty){
                done
        }
    }
    wait 16s -- wait 16 seconds, until the block time can be changed
}
```

- QTUM is a working Proof-of-Stake solution that runs a single-unit of account (**QTUM**)
- Will dramatically reduce the amount of energy needed to secure the blockchain network

- **Turing Completeness + Proof of Stake** opens up a new attack vector:
  - Running a **Denial-of-Service attack** where the attacker pays high gas fees to execute spam smart contracts.
  - **Possibility for attackers becoming block creators is high** during DoS attack – thus increasing stake rewards for the attacker

- QTUM's solution is **mutualized PoS (MPoS)**
  - **Reward delayed by 500 blocks,** so same stakes cannot be used to validate blocks of attacker's own transactions

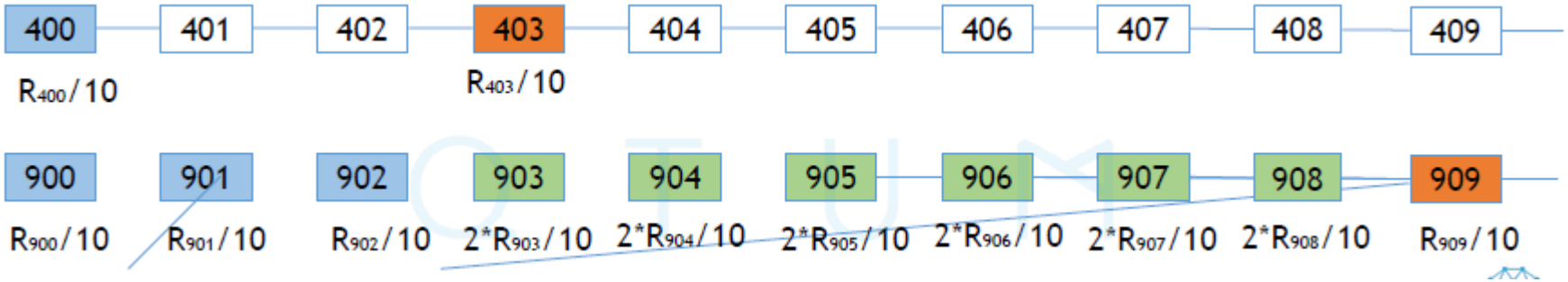
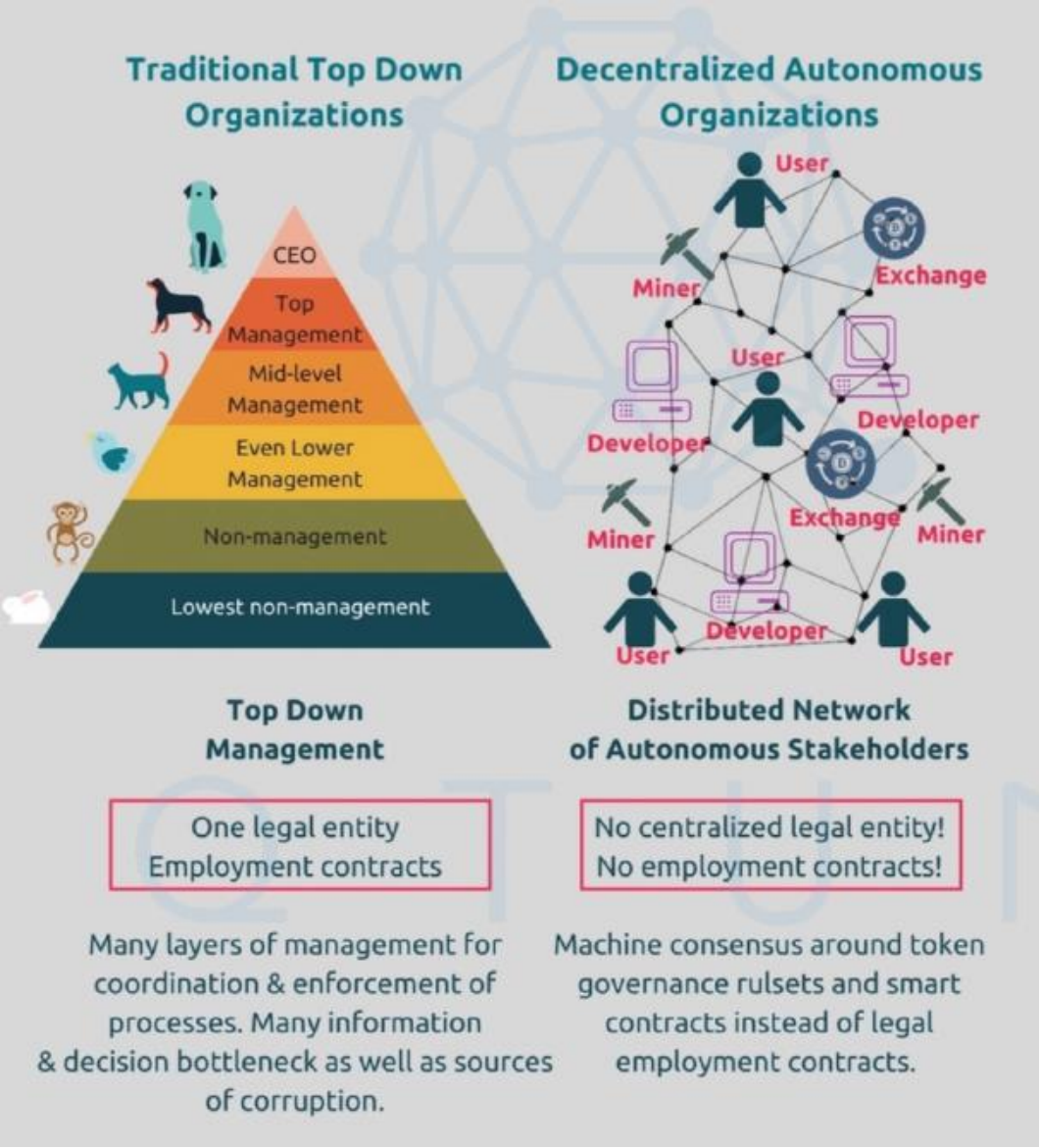


Figure 1: Mutualized proof of stake (1/10 reward initially comes at block 400 , 9/10 rewards come 500 blocks later (blocks 900-909))

3 Blockchains would be well served by decentralized governance protocols



- **Why do blockchains need a decentralized governance protocol?**
  - **Uncertain resolution mechanisms lead to forks.** e.g. Bitcoin scaling debate – acrimonious debate on scaling ended up causing a hardfork between Bitcoin and Bitcoin Cash, and almost caused a second hardfork between Segwit1x and Segwit2x

- **Current on-chain governance systems is direct democracy**
  - Bitcoin – BIP
  - Ethereum – gas limit vote by the miners
  - dBFT or DPoS – vote for delegate governance node

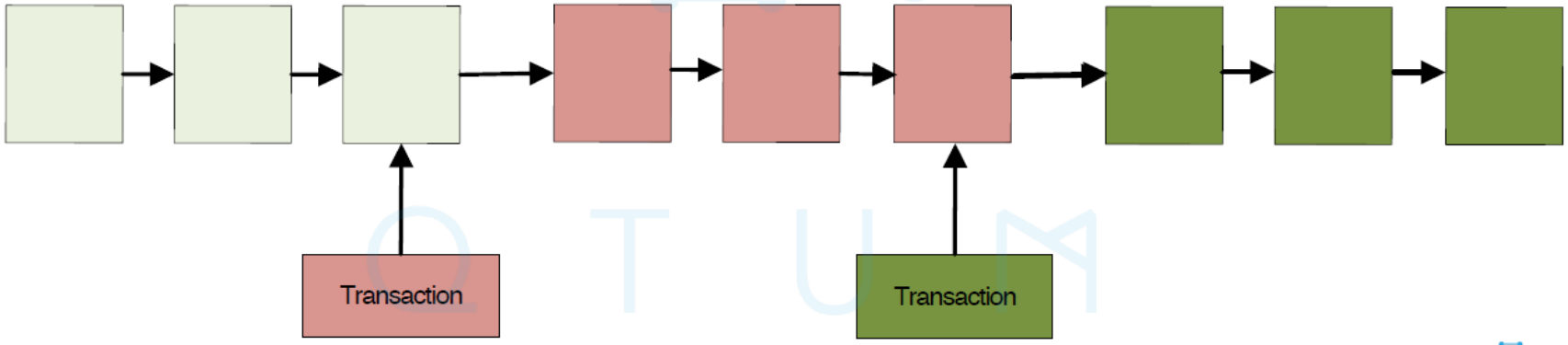
- **Problems with direct democracy**
  - Very small percentage of people vote
  - Richer people have more say (coin-holders or miners)
  - Susceptible to bribes

3 QTUM implements its decentralized governance protocol as on-chain smart contracts

- **QTUM's Decentralized Governance Protocol** allows for on-chain tweaks to different parameters
  - Algorithm updates
  - Strategy updates
  - Key bug fixes

- **How it's implemented:**
  - Consists of several smart contracts, where QTUM core executes those contracts to get to consensus state
  - Change blockchain state through transactions without needing a software upgrade

- **Currently, a limited set of block parameters can be modified**
  - Block size
  - Min GasPrice
  - Block GasLimit
  - Gas Schedule



APPLY(S,TX) -> S' or ERROR

QTUM

4 Why do smart contracts require a virtual machine?

- **What is a virtual machine?**
  - an emulation of a computing system, that can provide the functionality of a physical computer

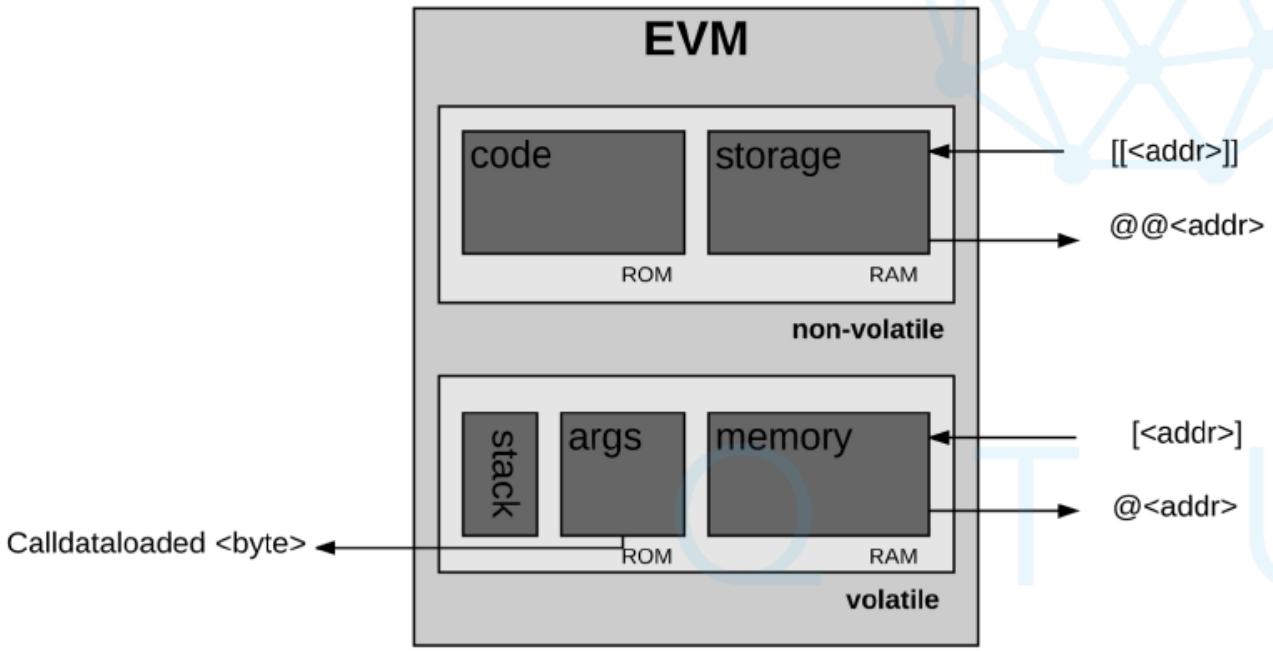
- **Why do we need a virtual machine?**

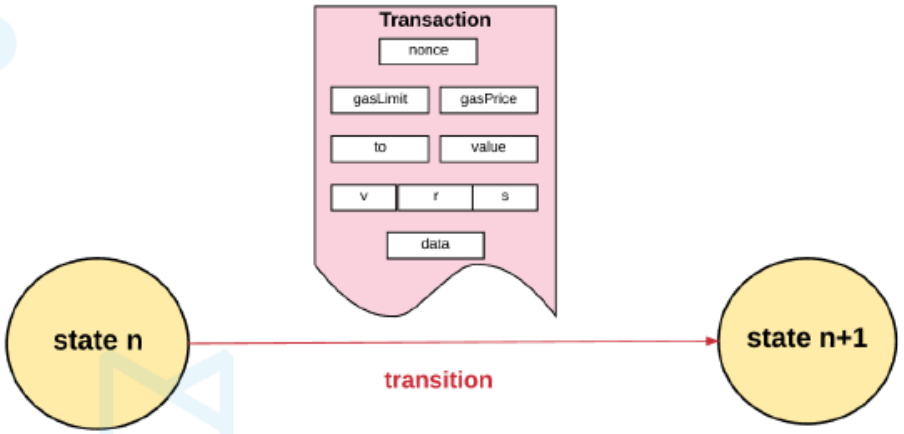
- **Consistency and determinism**
  - Smart contract is a program executing on a decentralized blockchain network
  - Each node might execute the same code in different environment, leading to different result, which cannot achieve any consensus
  - **Virtual machine ensures the consistency** for every distributed execution

- **Security**
  - Multiple distributed nodes are expected to execute smart contract of unknown origin, which was created for unknown purpose
  - Creates potential for massive distributed attacks that can compromise huge number of hosts and even entire system (e.g. viruses, DDoS)
  - Virtual machine ensures **smart contract code completely isolated** from the host system, its memory, computation power and operating system interface

4 What is the Ethereum Virtual machine?

- **The EVM is the operating system of Ethereum -** the environment executing smart contracts



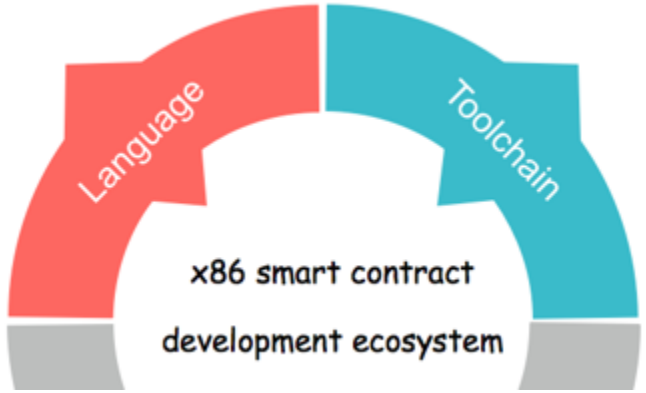


4 … the EVM looks good enough… why an x86 VM?

- x86 refers to Intel's x86 ISA, **used in almost every PCs/Servers/devices**
- x86 ISA emulator, compatible with existing x86 Instructions

Benefits for supporting mature x86 architecture :

- **Multiple programming language** support: C/C++/Rust, …
- **Mature toolchain**: various IDE, compiler, debuggers to uses



QTUM

❹ Mainstream software development is well-standardized…

Mainstream Software Development Ecosystem

Programming Languages

- Most popular programming languages:

  JavaScript, Java, Python, C++, C, Go, …
- Most loved programming languages :

  Rust, Python, Go, Swift, C#, Scala, …

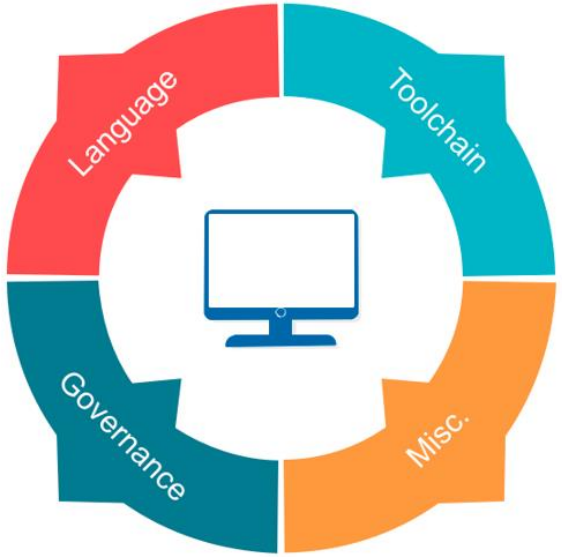
Development Tools

- Popular Development Environment :

  VSCode, VS, Eclipse, PyCharm, XCode, …
- Popular debuggers :

  GDB, LLDB, xdebug, VSCode, DBG, …
- Popular compilers:

  gcc/g++, llvm, clang, Intel C++ compilers, …



Software governance

- Software need upgrade
- Be able to go back and fix code problem
- the Libs used by software should be able to be upgraded

Other useful things

- Standard Librarys
- System Calls
- Reasonable cost
- Execution evironment
- Others…

4 … in a way smart contract development is not

Smart Contract Development Ecosystem

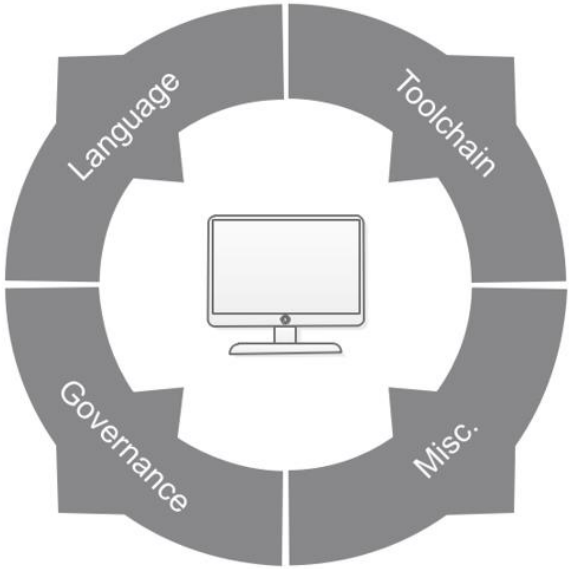
Limited Programming Languages

Solidity is the only popular language:

- Error prone (surprising **security problem**)
- Non-mainstream language: **difficult to learn, expensive to train developers**
- **EVM**'s only popular high-level language

Limited Development Tools

- Remix is the only "popular" development env.
- No Debugger
- Solc is the only "popular" compiler
- Few other toolchain support since EVM is not compatible with current ISA



No way to upgrade

- Smart contract is **hard to upgrade**
- **No way to go back and fix code problem**
- Library contract cannot upgrade

Other limitations

- No Standard Librarys & System Calls
- EVM makes VM and OS mixed
- Unreasonable gas model -- expensive
- Others like: Data storage, light client support, unable to paralle execute, …

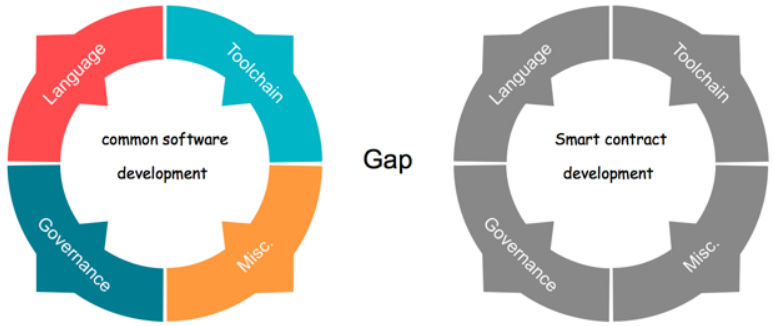
4 The x86 VM bridges the gap between mainstream software development and the smart contract ecosystem

Shortcomings of the EVM

1. **Limited smart contract coding language** (Solidity)
2. **Lack of a standard library**
3. **256bit integer** (not natively supported by most processors)
4. **Gas model**
   1. Hard to estimate gas cost
5. **Big bytecode –** waste of blockchain resources
6. **Immature testing and debug toolchain frameworks**

Big gap between two ecosystem

- Smart contract development: **difficult to learn, expensive to train.**
- **EVM**(Ethereum Virtual Machine) **is the key limitation**



QTUM

1. **More programming language support:** C/C++/Go/Rust etc.
2. **Standard library –** improving developer efficency
3. **Von Neumann architecture –** cooperative multitasking, pause, and resume execution
4. **Optimized gas model –** standardized gas prices for library calls, using the decentralized governance protocol
5. **First-class oracles** – smart contracts can load storage data directly
6. **Arbitrary key-value storage**
7. **Explicit dependency trees –** allow some contracts to be executed in parallel

Multi-language
Mainstream languages like: C/C++/Rust JAVA/Python/Go etc. in the future

Mature toolchain
Porting whatever useful IDEs, debuggers, and other productive tools.

Libraries & System calls
Improving development efficiency with various standard libraries and system calls

Upgrade without fork
Smart contract and library codes upgrade powered by DGP

Reasonable gas model
Redefine gas model, adjustable, responsive to market

More features on the way
New DeltaDB to upgrade SPV security, trusted libraries, more powerful QtumOS, ...

QTUM

❹ x86VM has been pushed to the testnet, with the first “Hello World” contract written in C