智能合约全栈介绍

howard @ qtum.org

- 为什么要智能合约?
- 什么是智能合约?
- DApp/合约/EVM 全栈走一回

为什么要智能合约?

挖矿很费资源，到底为了什么？

SOCIAL SCALABILITY

个人 -> 亲友 -> 社区 -> 城市 -> 国家 -> 全球

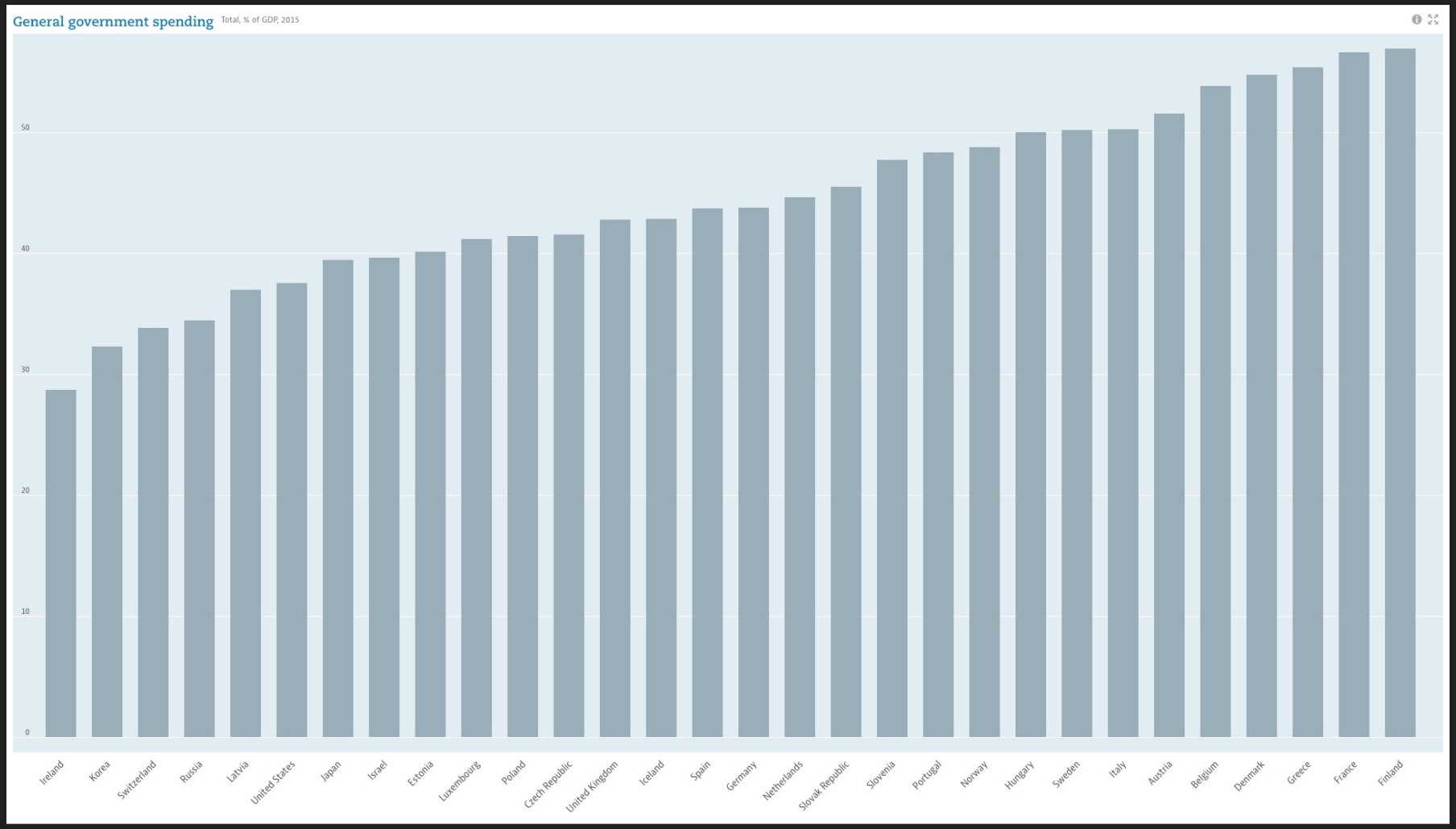
TRUST MINIMIZATION

降低相互伤害的风险，友谊的小船就不会翻

SOCIAL TECHNOLOGY

- 信任 -> 法律
- 人情 -> 市场
- 点对点对话 -> 多对多撮合
- 持续演化ing，累积信息复杂度

OECD 政府开销 GDP 占比

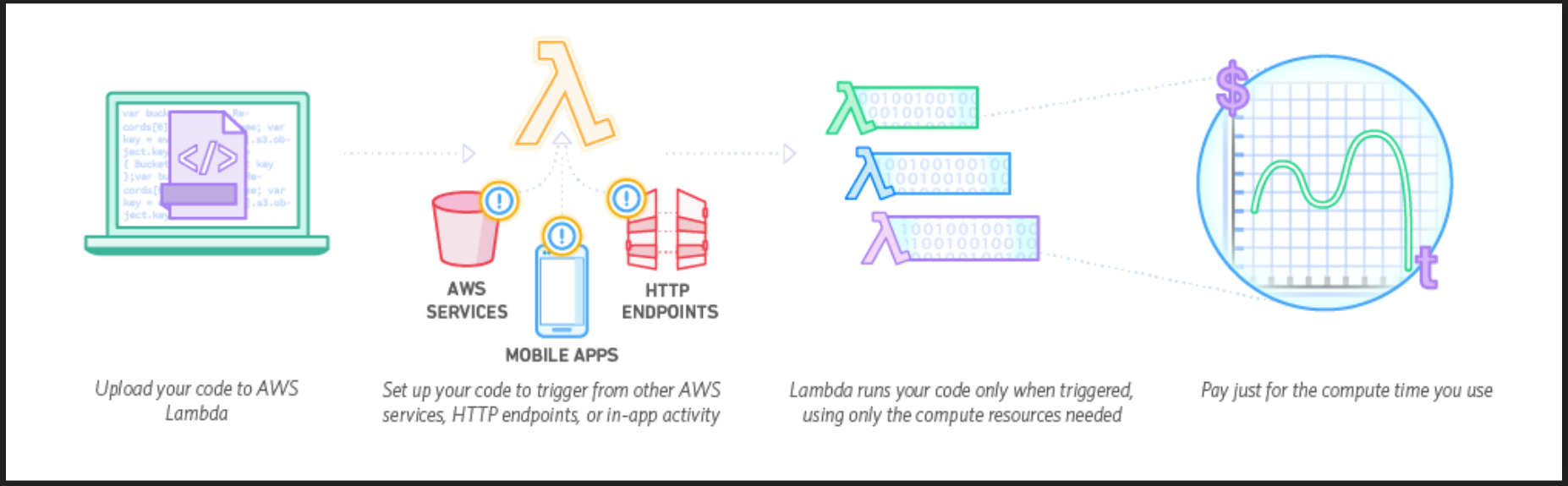


36% ~ 58% 不等

用计算机替代人肉

智能合具体是什么?

AWS LAMBDA


AWS
SERVICES
HTTP
ENDPOINTS
MOBILE APPS
Upload your code to AWS Lambda
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity
Lambda runs your code only when triggered, using only the compute resources needed
Pay just for the compute time you use

例子：记数器

```
pragma solidity ^0.4.18;

contract SimpleCounter {
  uint256 public count;

  function increment() public {
    count++;
  }
}
```

合约存储像数据库

```
// 每个合约有存储，像 KV 数据库
storage = {}

// 存储初始值为 0，不需要手动初始化
// storage["count"] = 0

// 方法可以读写合约存储
function increment() {
    // 合约“变量”其实是个数据库读写键值
    storage["count"]++
}
```

例子：红十字慈善链

收取捐赠，取款时必须标明用处

红十字慈善链需求

- 初始化：设定管理员
- 任何人都可以捐款
- 只有管理员可以取款

```
contract DonationChain {
    address public owner;
    function DonationChain() public {
        owner = msg.sender;
    }
    function donate() public payable {
        require(msg.value > 0.001 ether);
    }
    event EventWithdraw(address toAddress, uint256 amount, string pur
    function withdraw(address toAddress, uint256 amount, string purpo
        require(msg.sender == owner);
        toAddress.transfer(amount);
        emit EventWithdraw(toAddress, amount, purpose);
    }
}
```

初始化：设定管理员

- msg.sender 相当于当前用户
- 创建合约的用户成为 owner

```
address public owner;
function DonationChain() public {
    owner = msg.sender;
}
```

任何人都可以捐款

- payable 标记该方法可以打钱
- msg.sender 是给合约打钱的用户
- msg.value 是打了多少钱，最终会存在合约的余额
- require 断言，不满住条件抛错, 退款

```
function donate() public payable {
    require(msg.value > 0.001 ether);
}
```

管理员可以取款

- 检查当前用户是否是管理员
- address 类型可以用 transfer 方法转账
- 由合约的余额转钱
- 输出取款金额和目的日志

```
event EventWithdraw(address toAddress, uint256 amount, string purpose
function withdraw(address toAddress, uint256 amount, string purpose)
    require(msg.sender == owner);
    toAddress.transfer(amount);
    emit EventWithdraw(toAddress, amount, purpose);
}
```

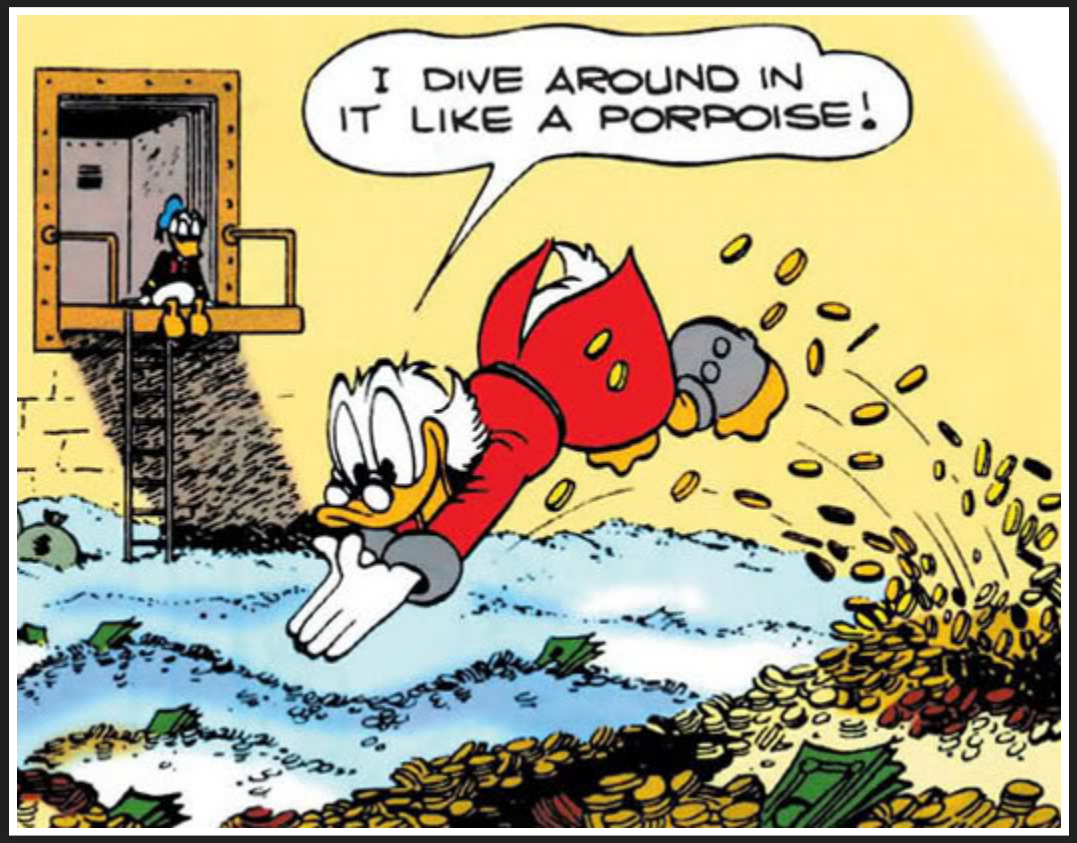
EVM 平台的一些特性

- 用户承担计算费用 (gas 模型)
- 用户掌握权限。客户端签名，服务端验证
- 每个节点重复每个计算，镜像每条数据

去中心化平台的缺点

- 慢，贵，矬
- Transaction 的生命周期复杂
- 没有网路请求
- 没有随机源
- 不能更新合约代码

所以，去中心化有意义吗?


I DIVE AROUND IN IT LIKE A PORPOISE!

DAPP 全栈走一回

```
contract SimpleStore {
  function set(uint newValue) public {
    value = newValue;
  }

  function get() public constant returns (uint) {
    return value;
  }

  uint value;
}
```

前端 -> RPC -> ABI -> 合约 -> EVM -> 存储

WEB 2.0 => WEB 3.0

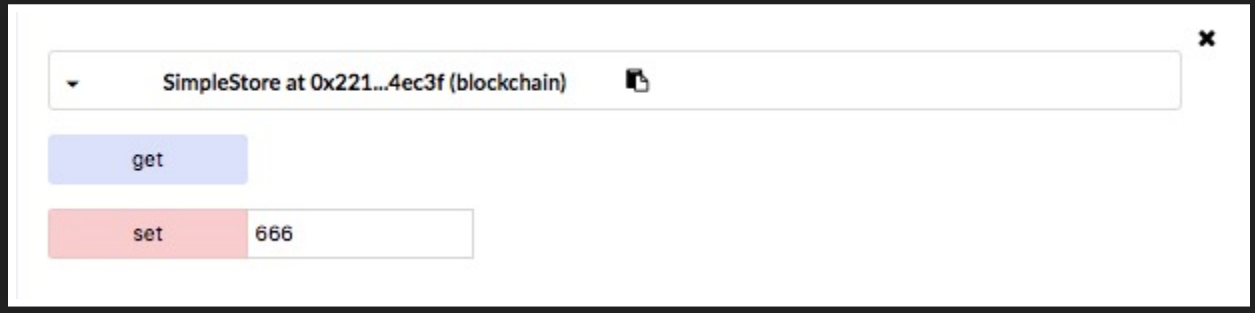
- HTML5 - DApp / web3
- Request - Transaction
- JSON - ABI encoding
- Service - Smart Contract
- OS - EVM
- DB - Merkle Tree

前端

- 展示链上数据
- 用户在本地签名，提交数据到链上
- Web 服务 vs 纯客户端
  - coinbase, binance 平台控制权限 （不够清真）
  - myetherwallet, imtoken 用户控制权限

前端例子： REMIX IDE

SET(666)



用 RPC 递交事务

ETH_SENDTRANSACTION

```
{
"jsonrpc":"2.0",
"id": 1,
"method":"eth_sendTransaction",
"params":[{
        "from": "0x551a9e83054bf9ea8367b4de98d7a30b7a4122ab2b48b0ecf3
        "data": "0x60fe47b1000000000000000000000000000000000000000000
    }]
}
```

已经递交到网路上的事务

Overview

Transaction Information

Tools & Utilities

TxHash:	0x0f8d0353bd48c5b0b071dc5ec9eb3114624145b96739c637b7fafccee3813a5e
TxReceipt Status:	Success
Block Height:	1945251 (1 block confirmation)
TimeStamp:	34 secs ago (Mar-17-2018 01:06:47 AM +UTC)
From:	0x637e59b69efc2679f482b6c90258a7ef6823699e
To:	Contract 0x51d36292ada92c2f05a8cac84bc1303e4227bc1a
Value:	0 Ether ($0.00)
Gas Limit:	41706
Gas Used By Txn:	41706
Gas Price:	0.000000001 Ether (1 Gwei)
Actual Tx Cost/Fee:	0.000041706 Ether ($0.000000)
Nonce:	41
Input Data:	Function: set(uint256 _fee) *** MethodID: 0x60fe47b1 [0]: 000000000000000000000000000000000000000000000000000000000000029a

Convert To Ascii

用 ABI 编码传递数据给合约

```
60fe47b1

000000000000000000000000000000000000000000000000000000000000029a
```

- sha3("set(uint)") == "60fe47b1..."
- 666 == 0x29a

EVM 加载并执行代码

```
function set(uint newValue) public {
    value = newValue;
}
```

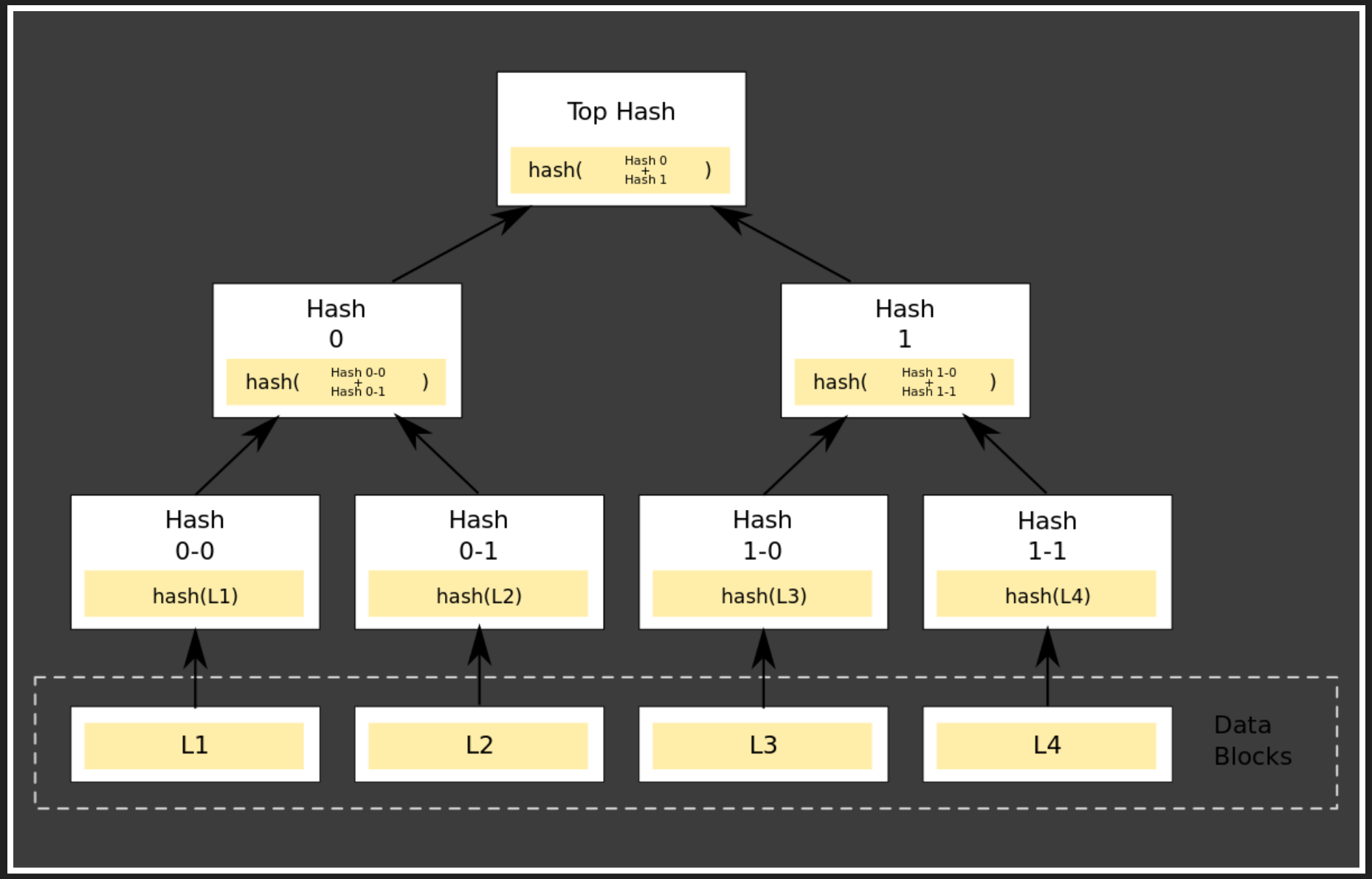
EVM 字节码

```
and(div(calldataload(0x0), 0x10000000000000000000000000000000000000000
0x60fe47b1
dup2
eq
tag_4
jumpi

tag_4:
  tag_5
  // load 32 bytes of calldata from offset 0x4
  calldataload(0x4)
  jump(tag_6)

tag_6:
  0x0
  sstore
```

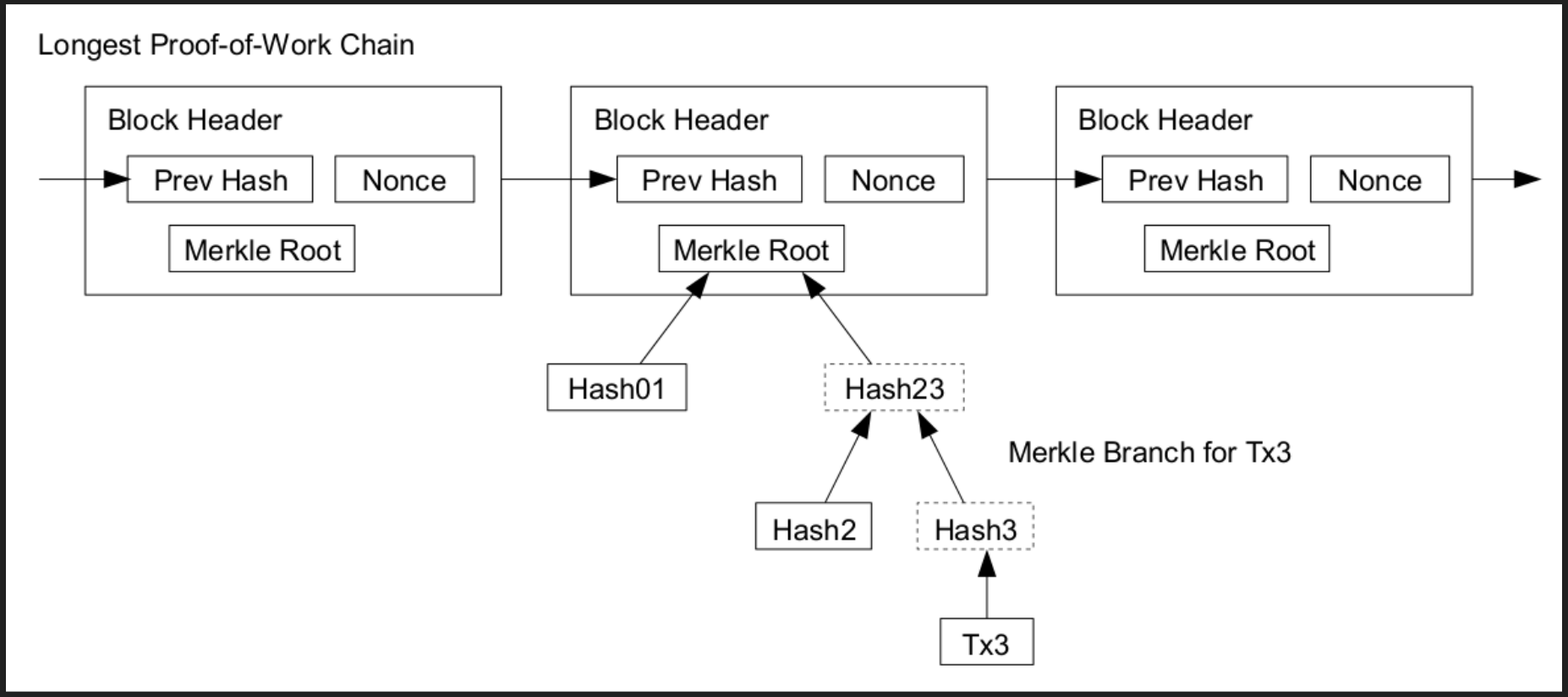
EVM 字节码 => 伪代码

```
if calldata[0..4] == 0x60fe47b1
  goto tag_4

// tag_4
$value = calldata[4..4+32]
goto tag_6

// tag_6:
sstore(0x0, $value)
```

SSTORE 写入 MERKLE TREE



在新的区块记录新的 MERKLE TREE 根



前端 -> RPC -> ABI -> 合约 -> EVM -> 存储

区块链现状

未来

- 十亿用户
- 扩容
- 侧链，跨链
- UX
- 隐私

参与 QTUM 开源项目

https://github.com/qtumproject

THE END