以太坊

路远

中国科学院软件研究所

前章要点回顾

- 比特币应用实例与开发环境
  - 数字证书的透明化
  - 视频/数据的P2P交换
  - 比特币开发环境
    - 链上脚本
    - 链下客户端
- 以太坊应用实例与开发环境
  - 数据众包
  - P2P内容传输
  - 以太坊开发环境
    - 链上脚本
    - 链下客户端

本讲内容

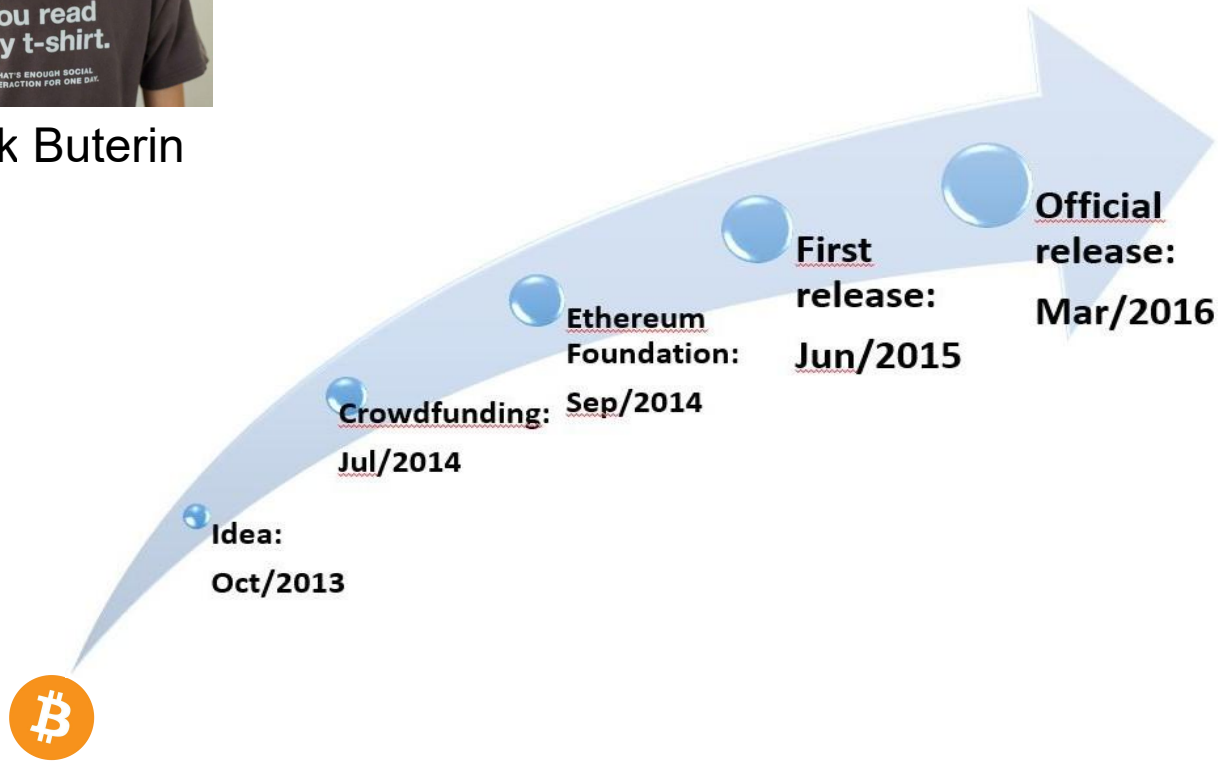
- 以太坊背景和简要回顾
  - 和比特币的对比
- 以太坊架构设计
  - P2P网络
  - 数据结构
  - 共识机制
  - 合约引擎
- 以太坊Dapp开发
  - 开发周期
  - 开发案例

以太坊背景和简要回顾

以太坊的历史时刻

Vitalik Buterin

- Russian-Canadian programmer
- Co-founded Ethereum when he was 19 years old



和比特币的总体对比

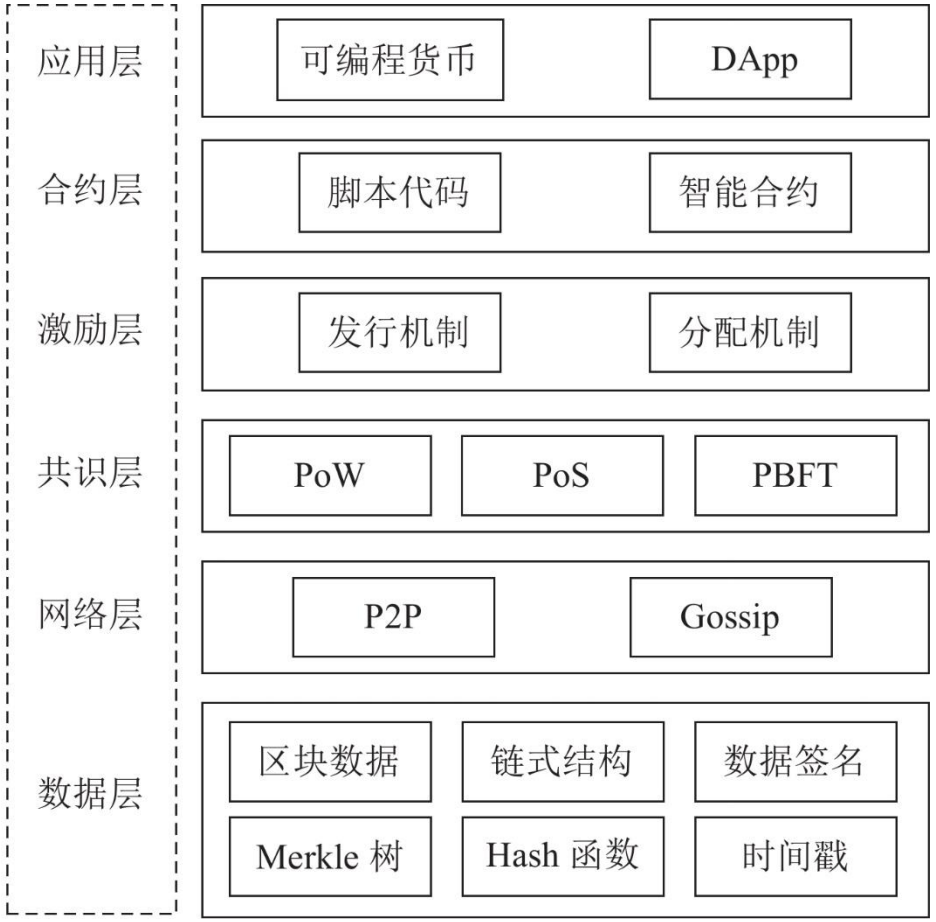
	比特币	以太坊(PoS升级前)
哈希函数	SHA256、ripemd160	Keccak256 (NIST SHA3标准变体)
数字签名	签名算法：ECDSA 椭圆曲线：secp256k1	签名算法：ECDSA 椭圆曲线：secp256k1
传输与通信	P2P网络	P2P网络
共识机制	PoW	PoW (目前已经升级 PoS)
挖矿难度	目标为平均10分钟一个区块	目标为12-15秒一个区块
区块奖励	基础奖励每210000个区块减半	原始设计不含有减半机制，目前引入了更复杂的发行量控制机制
状态是否在链上认证?	UTXO集合不在链上认证	所有的状态通过默克尔基数树在链上认证
区块是否包含叔块指针?	否，仅指向前一个区块	是，如果前一个区块有分叉（即存在叔块），使用叔块指针包含叔块
记账模型	UTXO	账户
支持图灵完备的链上计算?	仅支持功能受限的少数脚本，无法实现图灵完备的智能合约	支持（接近）图灵完备的智能合约（仍受限于燃料上限）

以太坊的架构设计

从分层角度来看以太坊

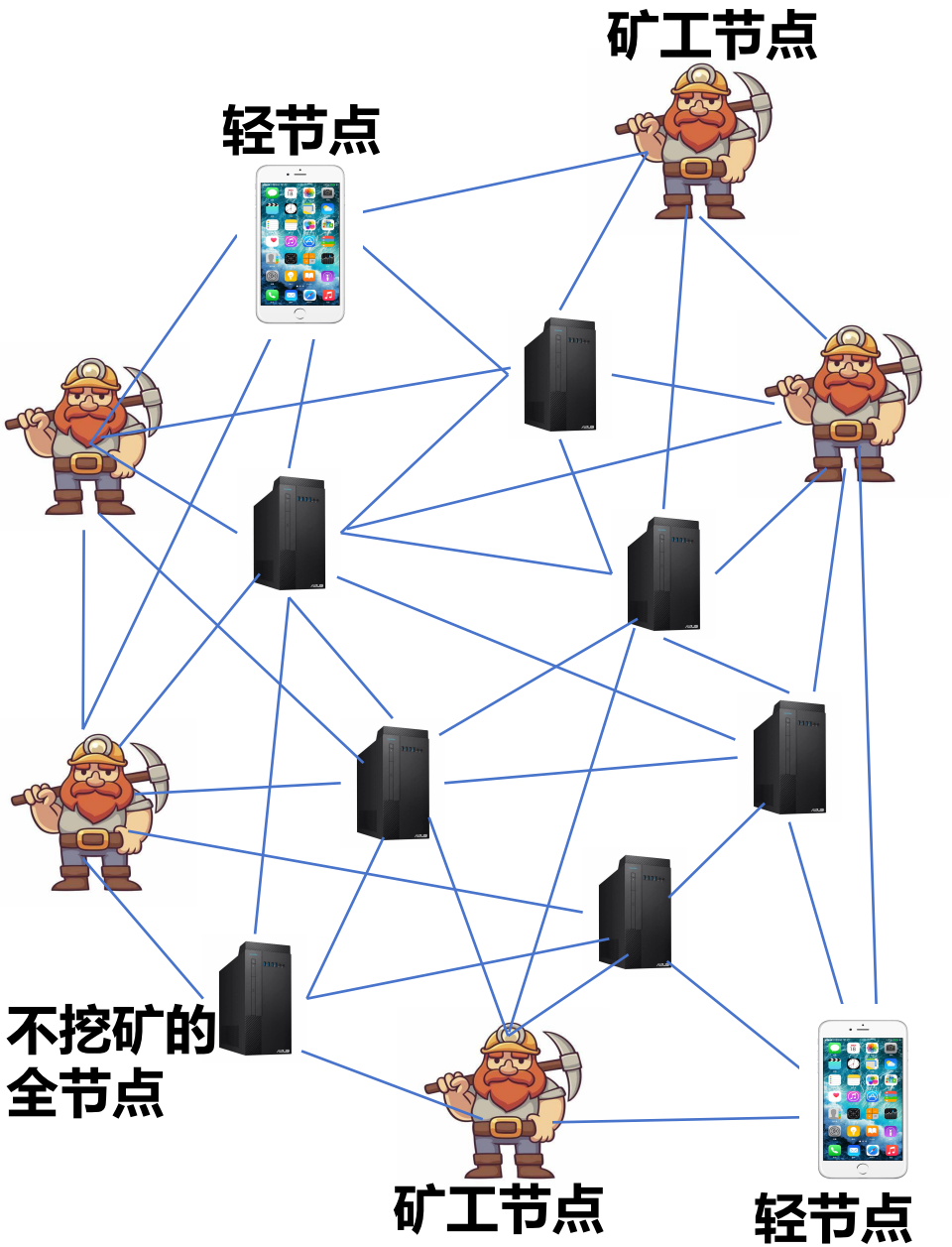
—整套区块链基础设施
=> 也被业界称为 Layer-0

- 数据层：区块链、账户模型
- 网络层：p2p网络
- 共识层：PoW (目前改为了PoS)
- 合约层：EVM虚拟机
- 激励层：提供矿工奖励
- 应用层：各类智能合约，实现NFT、DeFi 等各类Dapp应用



以太坊核心网络

- 类似比特币，以太坊网络也是一个p2p网络：
  - **(不挖矿的)全节点**：参与PoW和新区块产生以外的共识机制执行，下载所有新产生的区块，并验证区块中PoW有效性和交易的有效性；
  - **矿工节点**：进行PoW挖矿、可能生成新区块的全节点；
  - **轻节点**：只下载区块头和验证PoW，不下载区块内容、不验证交易的节点，启动时通过参数配置
- 启动时设置 MaxPeers 参数来规定允许的最大连接数

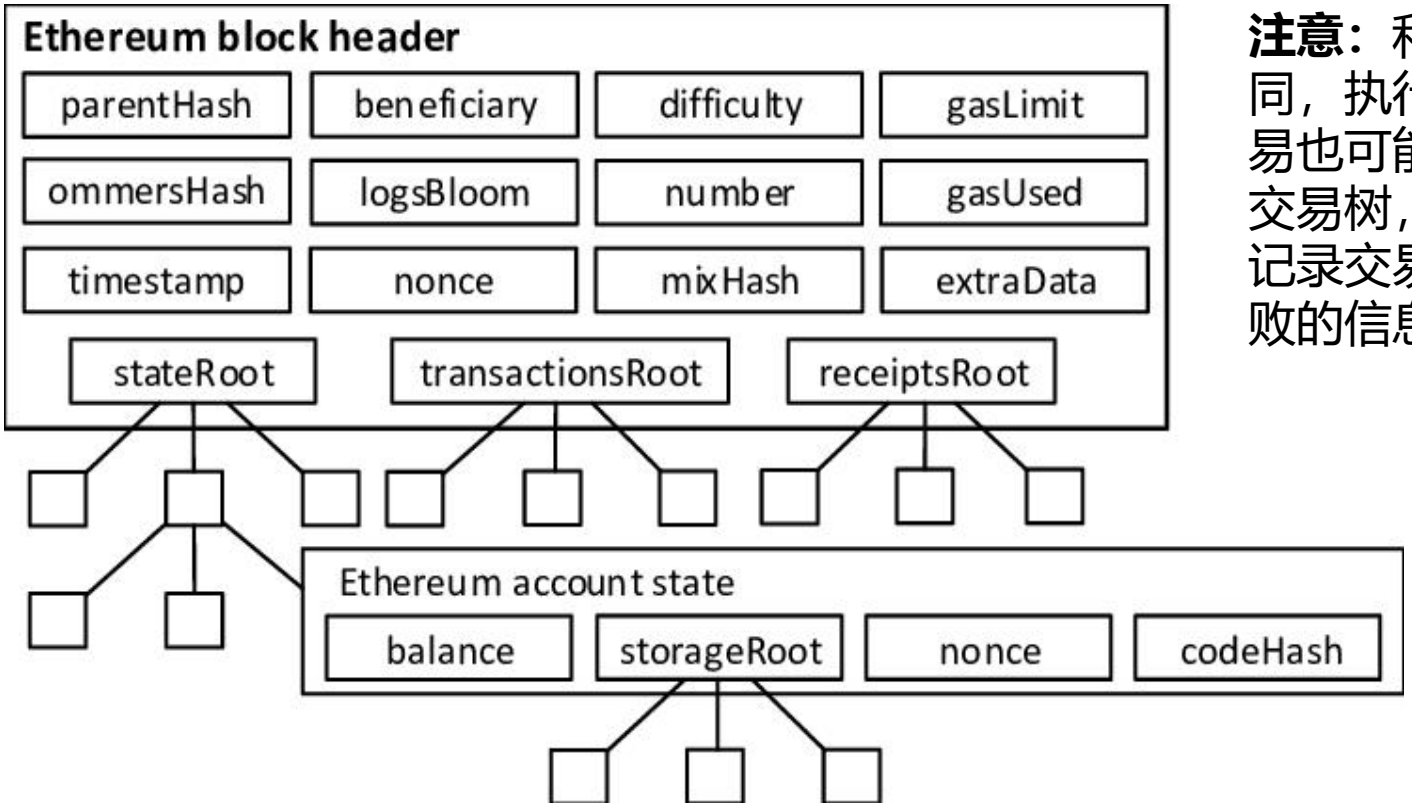


消息通过P2P网络上的gossip扩散完成

以太坊的区块结构 (23年前的PoW版)

PoW时代的以太坊区块结构，和比特币的几大不同：

- 三个哈希树：分别承诺全局的**账户状态**、本区块的**交易**、本区块的交易**收据**(日志)
- 布隆过滤器：提供区块中日志事件的快速搜索方法
- 叔块指针：包含某些分叉的哈希值



注意：和比特币不同，执行错误的交易也可能会记录在交易树，收据树会记录交易成功或失败的信息

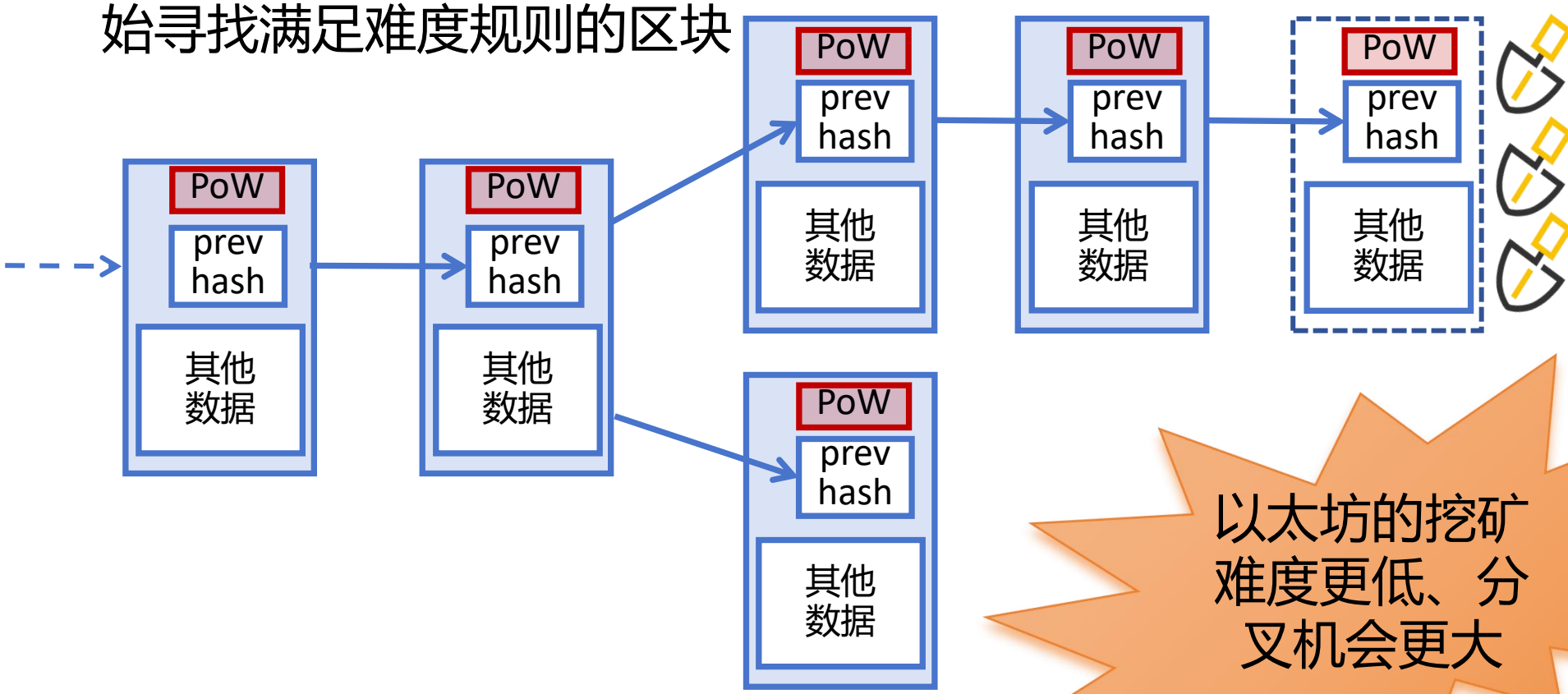
基于工作量证明的共识机制

工作量证明 (proof of work) :

提出区块时，需要使区块的哈希值小于特定的目标值

最长链规则 (longest chain rule)：

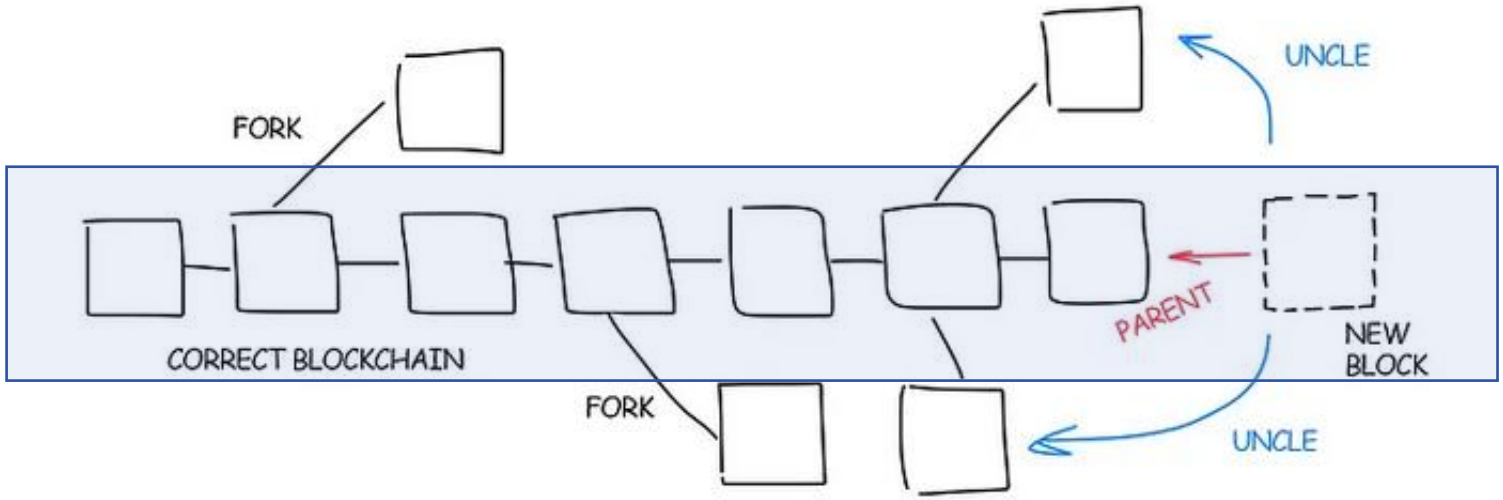
诚实节点的“挖矿规则”：当遇到分叉时，选择更长的分支开始寻找满足难度规则的区块



叔块指针

面临更频繁的分叉问题，以太坊引入了叔块指针：

- 仅仅用来生成区块的奖励
- 不考虑叔块中的交易信息
- 每个新区块最多指向两个叔块
- 不能指向太久远的孤块 (和父块不”同辈”)



账户模型

比特币使用 **UTXO模型** =>

给某个UTXO提供有效的签名，从而将其转入其他账户或公钥 (产生新的UTXO)

以太坊使用 **账户模型** =>

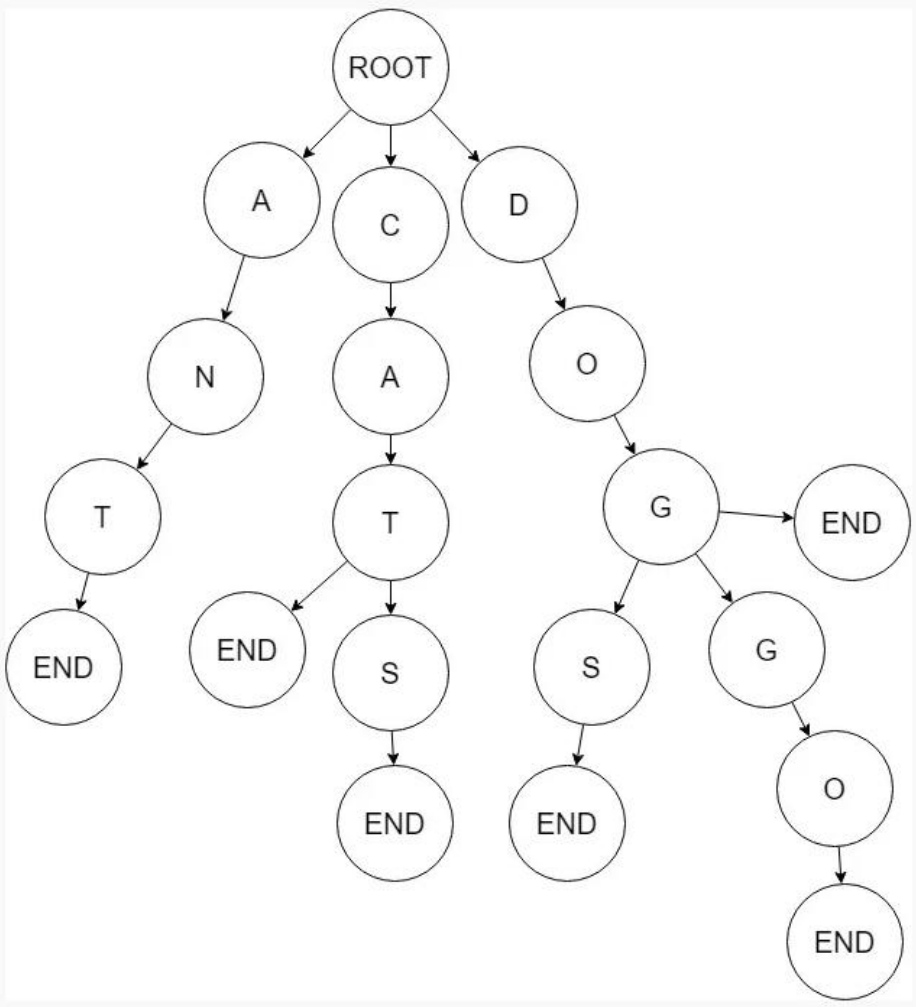
维护地址和对应余额的 " 表格"，转账时提供对应账户的签名，在自己的余额里减掉自己的转账金额，在收款地址的余额里加入转账金额

账户 (公钥的哈希值)	余额
0xF96153036f5C7ddd740DD99427BF524Aa4721C8D	100
0x2611a532A8850Dd622522735a04876DAAe70e43D	200 **- 50**
0xEBef7cEf76A38e66fa16BA09CEeCc0D21E51068a	150 **+ 50**
…	…
0x3d1909805175e705914855C14b9FC726484Ba0F0	180

转账 50 ETH

默克尔基数树 (Merkle Patricia Trie)

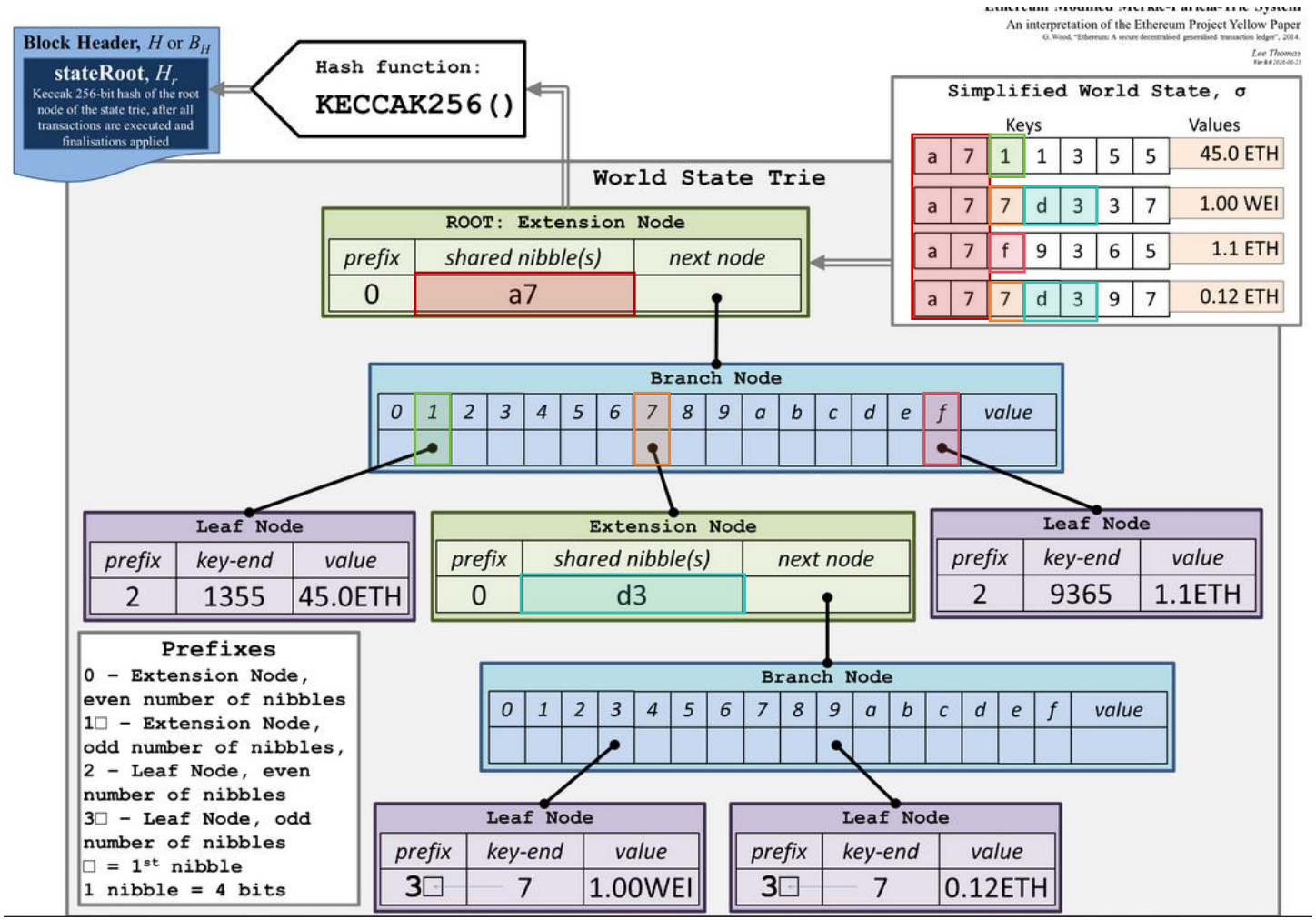
Patricia Trie (Radix Tree/Trie)： key-value数据结构



查询 key=“ant” 对应的 value：
从root一步一步找到ant对应的叶子节点

默克尔基数树 (Merkle Patricia Trie)

Merkle Patricia Trie (具有Patricia Trie结构的哈希树)



假设有4个账号:

a711355
a77d337
a7f9365
a77d397

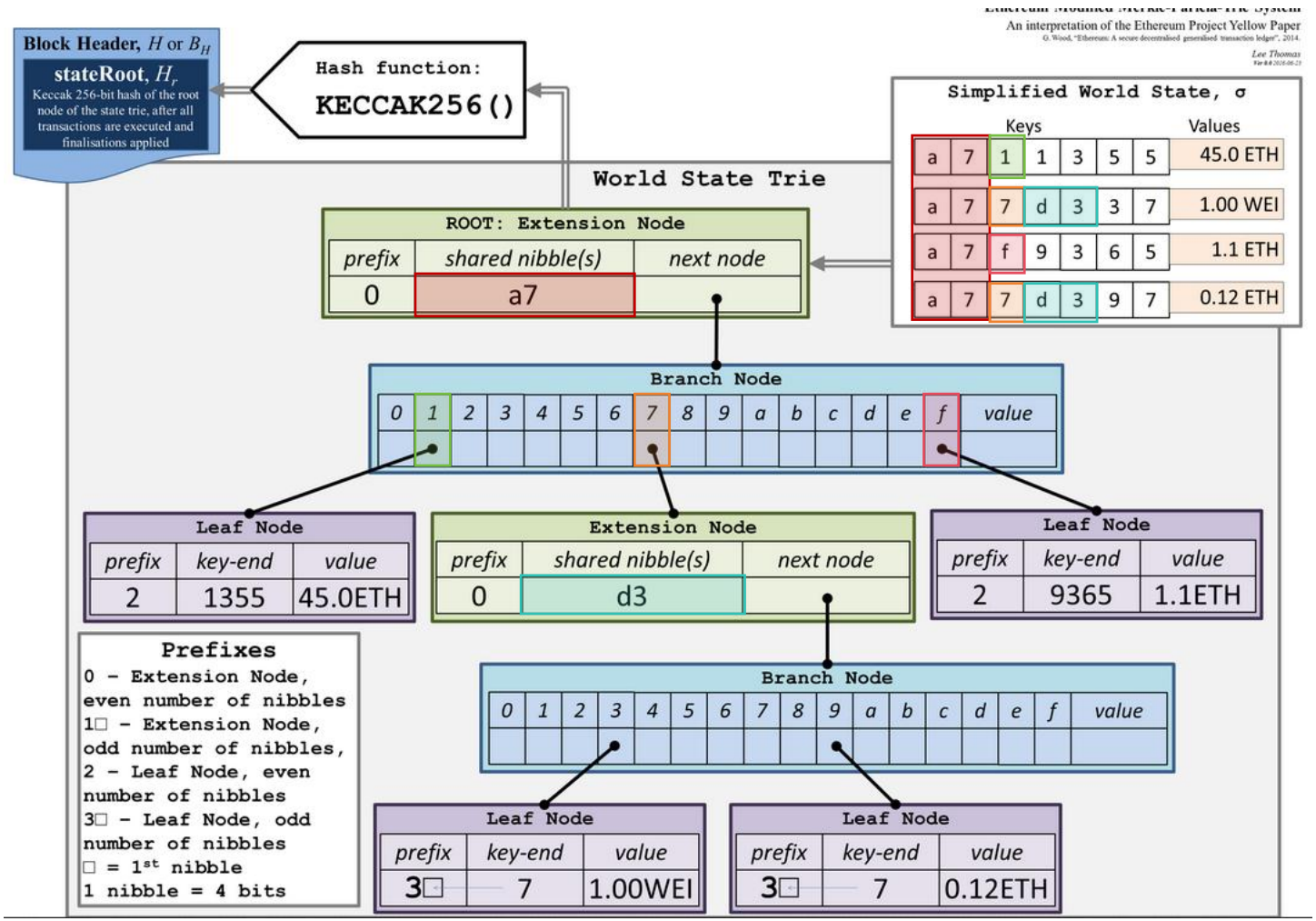
根扩展节点
使用a7作为共享前缀

仍然没有添加完所有账号，因此根扩展节点引入一个**分支节点**

分支节点有16种可能的扩展方式，即包含一个半字节所有的可能性，位置1和位置f包含**叶子节点**，位置7继续添加扩展节点，其余位置为**空节点**

默克尔基数树 (Merkle Patricia Trie)

Merkle Patricia Trie (具有Patricia Trie结构的哈希树)



假设有4个账号:

a711355
a77d337
a7f9365
a77d397

如何给出a711355在MPT的证明?

a7-1是叶子节点

如何给出a722222不在MPT的证明?

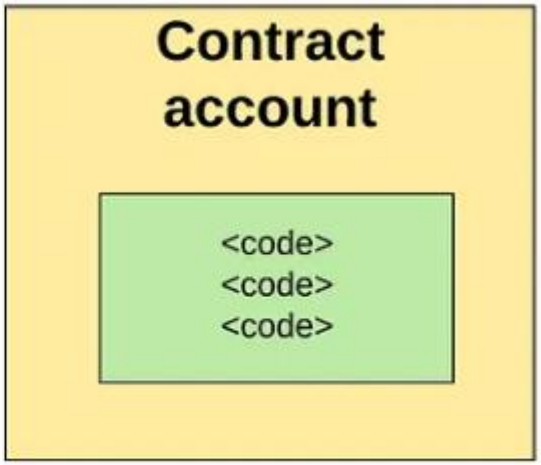
a7-2是空节点

普通账户 vs 合约账户

- 以太坊有两类账户
  - **外部拥有账户** (普通账户) **EOA**：由用户的私钥控制，状态包括余额信息等
  - **智能合约账户 CA**：地址由创建者的普通账户地址导出，包含可执行代码、变量状态、余额等信息

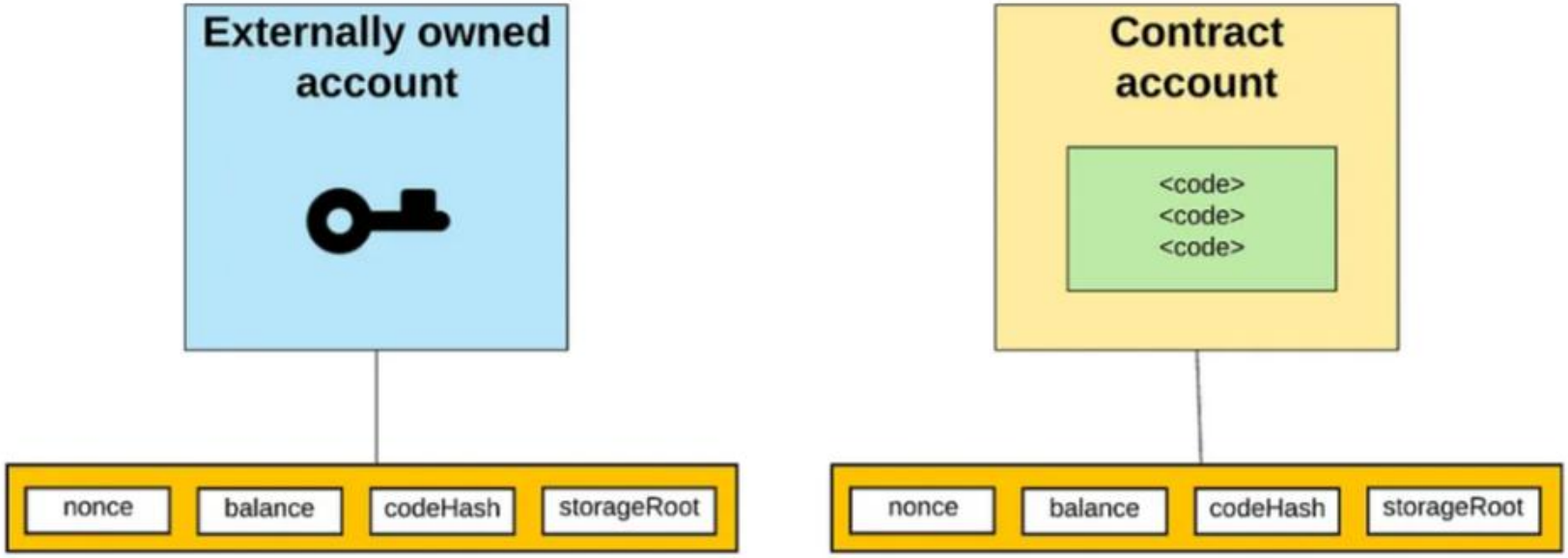
外部拥有账户地址由密钥导出

合约地址由创建者的普通账户地址导出



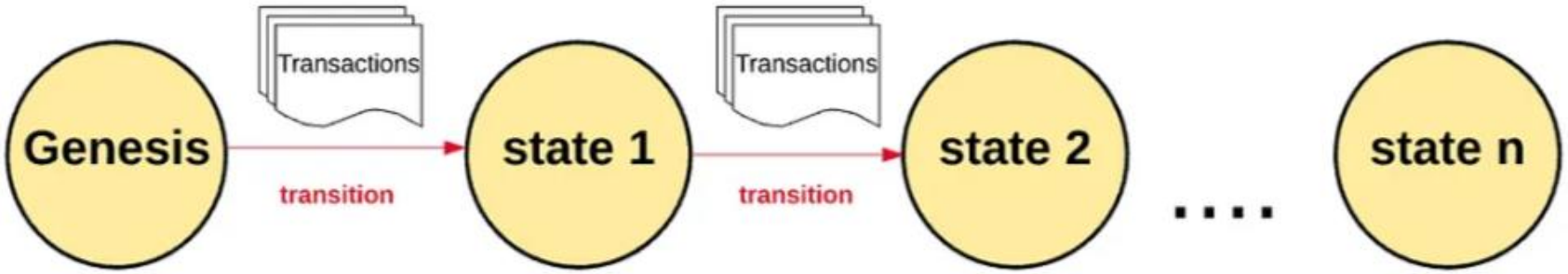
账户状态

- 每个账户的状态定义为四元组：
  - **nonce**：对于EOA账户，代表该账户发送过的交易数量；对于CA账户，代表该账户创建过的合约数量；
  - **balance**：余额，以Wei为单位 (每个以太有1e+18 Wei)
  - **storageRoot**：默克尔基数树的树根，存储所有的存储内容；
  - **codeHash**：对于CA账户，合约EVM代码的哈希值；对于EOA账户，空串。



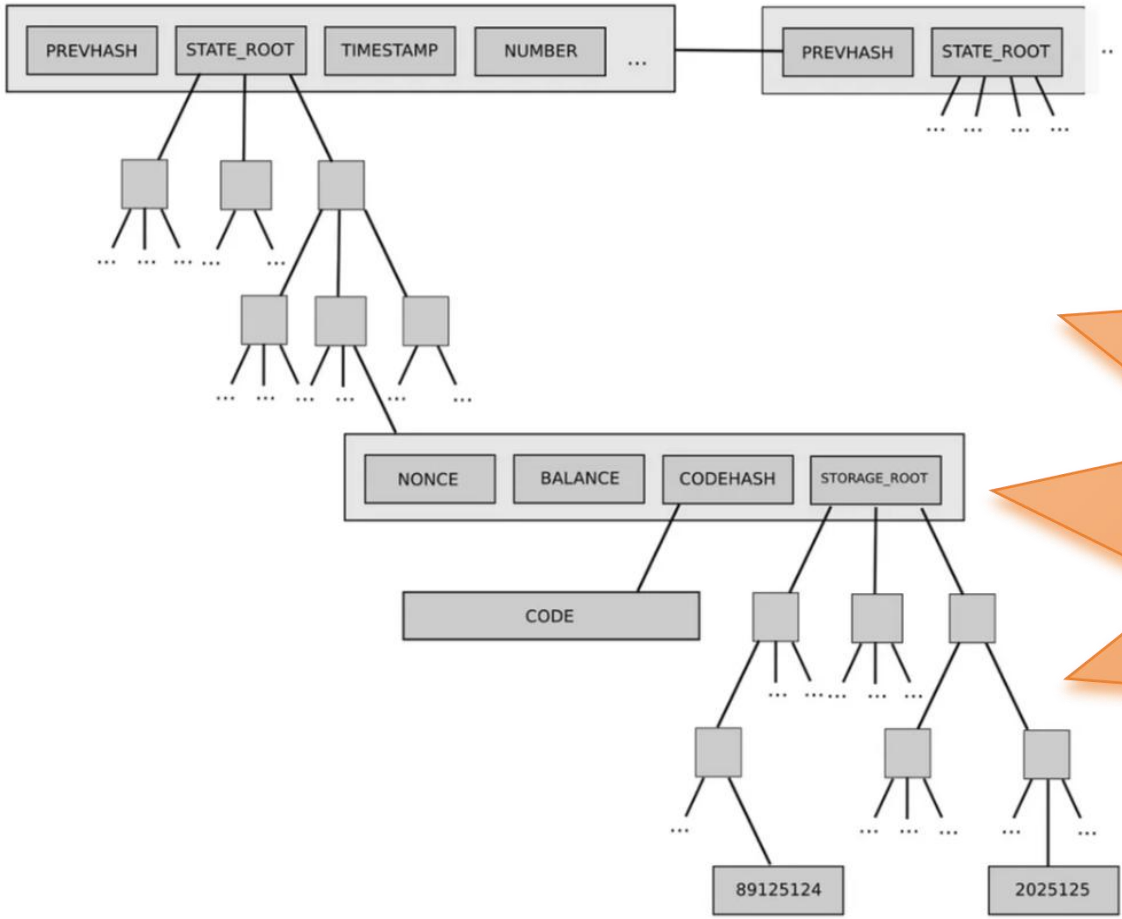
世界状态

- 所有账户的状态 => 世界状态
- 创世区块对应的世界状态称为 “genesis state”
- 外部账户拥有者可以发送新的交易 (有效的数字签名)
- 新区快打包了新交易 => 世界状态在交易的驱动下发生变化
  - 如转账交易：收款方账户余额增长，付款方账户余额减少
  - 也可以创建新的智能合约，或执行已经部署的智能合约



世界状态

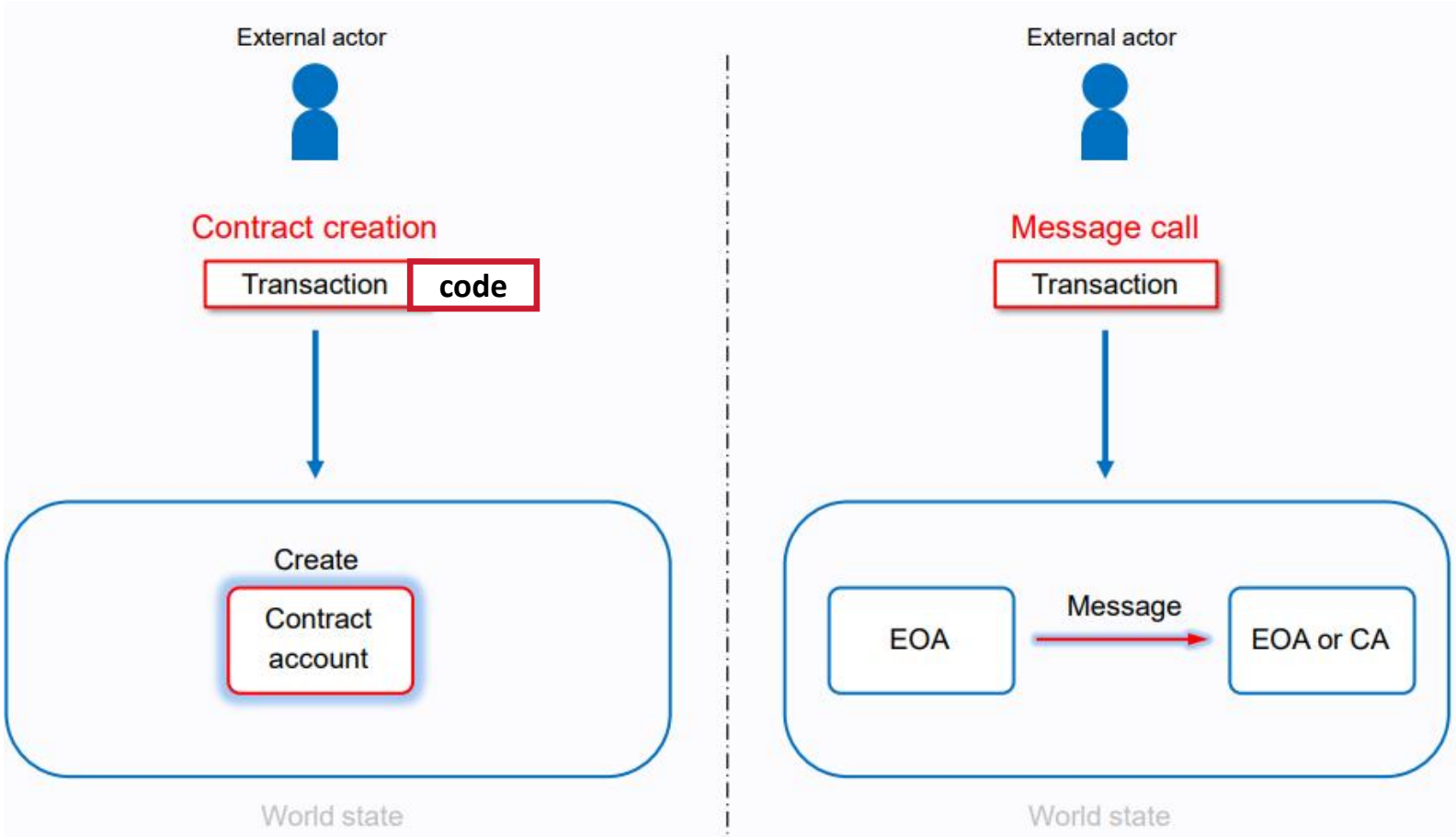
- 世界状态 通过 认证数据结构（默克尔基数树）存储在区块中
- 相比之下，比特币的世界状态 （UTXO 集合）没有在区块中认证



思考：在区块中认证状态 对 **轻节点** 和 **新节点快速同步**有什么好处?

交易的宏观类型

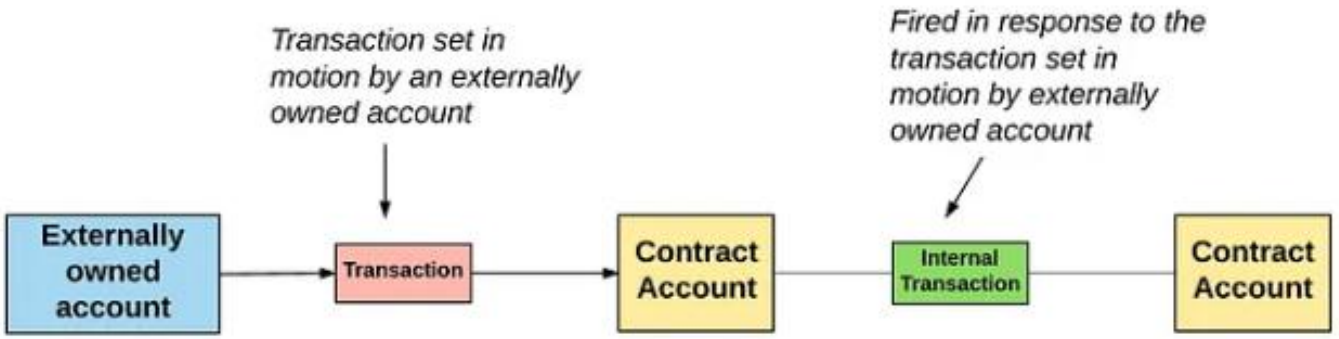
- 合约创建交易：在世界状态中创建新的合约账户 (包括初始状态)
- 消息呼叫交易：触发“消息”，实现 EOA 与 EOA/CA间的状态读写



两种实际的交易类型

- 合约创建交易：在世界状态中创建新的合约账户 (包括初始状态)
- 消息呼叫交易：触发“消息”，实现 EOA 与 EOA/CA间的状态读写

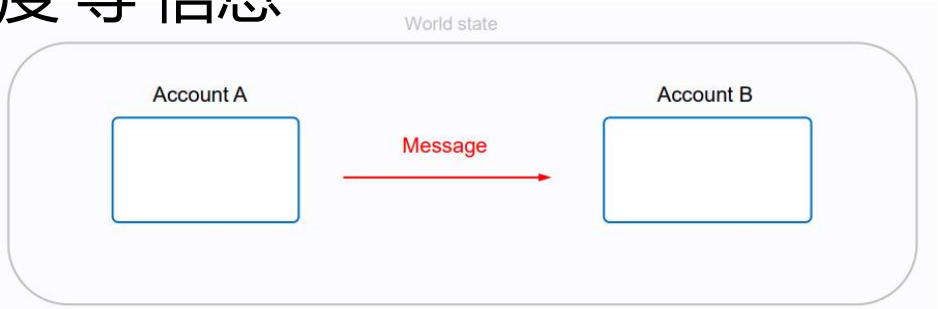
消息呼叫实现EOA与EOA之间的状态读写



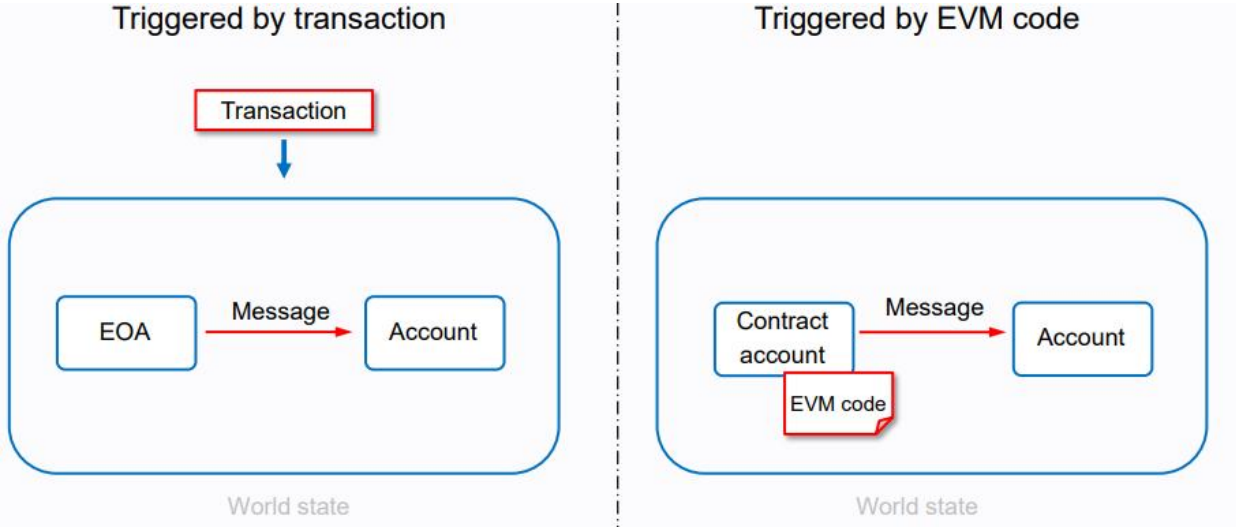
消息呼叫实现EOA与CA之间的状态读写

消息

- “消息” 在两个账户之间传输：携带 数据 (如调用特定函数的输入) 或者 转账额度 等 信息

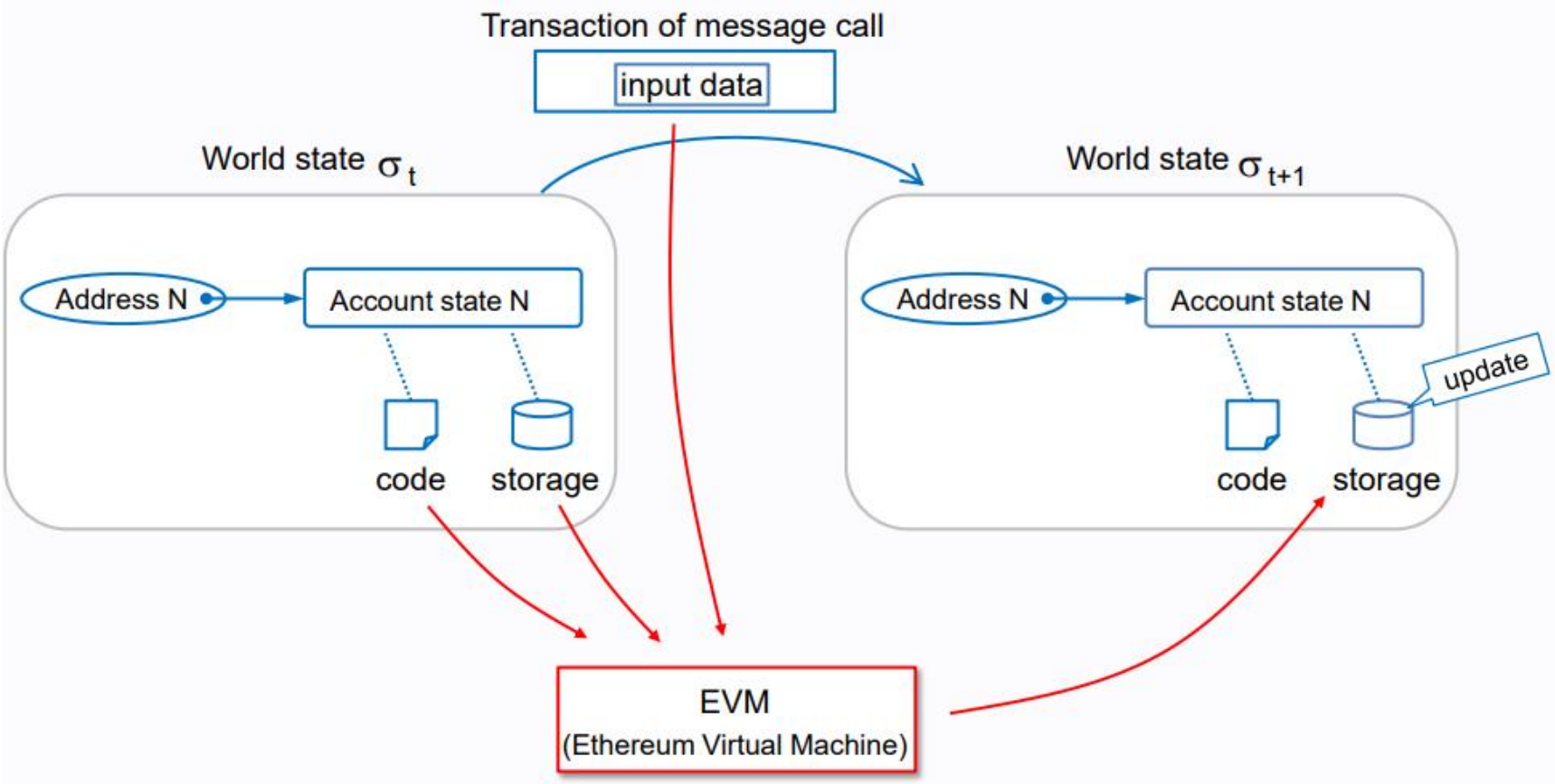


- 消息可以由交易触发，也可以由智能合约代码的执行过程触法，*但智能合约代码不可能自动地初始化一个消息 (需要由 EOA 调用执行触发)*



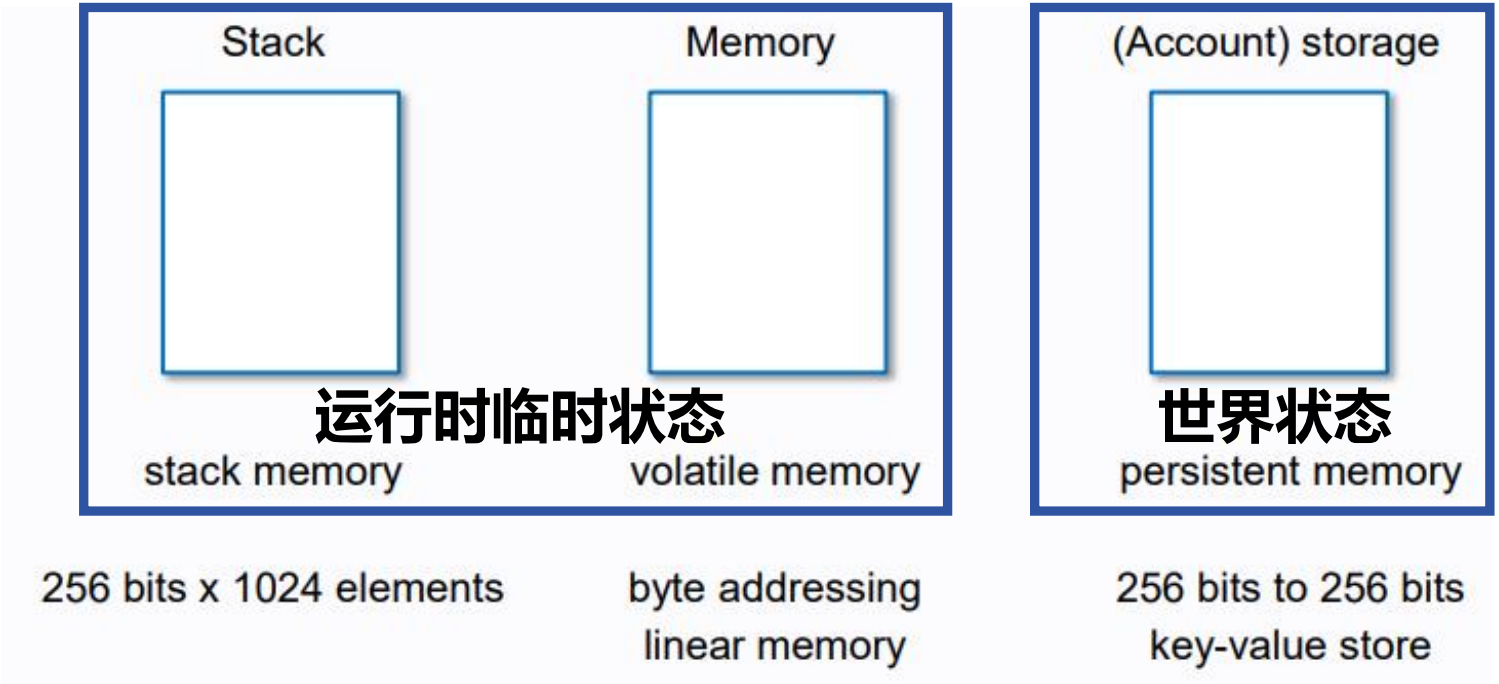
以太坊虚拟机 EVM

- 以太坊中的智能合约代码需要按照提前规定的以太坊虚拟机执行
  - 栈机、256-bit字长、**确定性执行**、汇编型代码



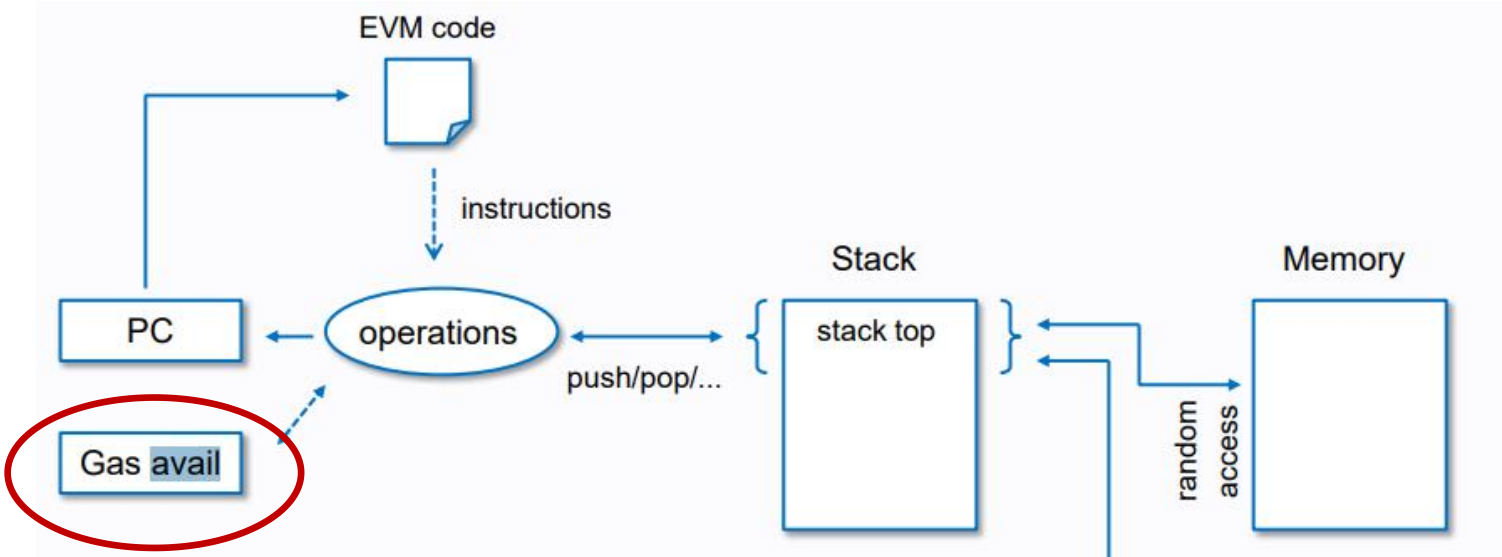
以太坊虚拟机 EVM

- EVM中的三类空间
  - 栈：栈中的数据可以被执行各类计算操作
  - 内存：记录智能合约执行过程中的临时变量
  - 账户(存储)：永久存储的世界状态变量



燃料机制

- 在EVM中，每个操作码都需要一定的 “燃料”
- 在EVM中，实现有 “燃料” 计数器!

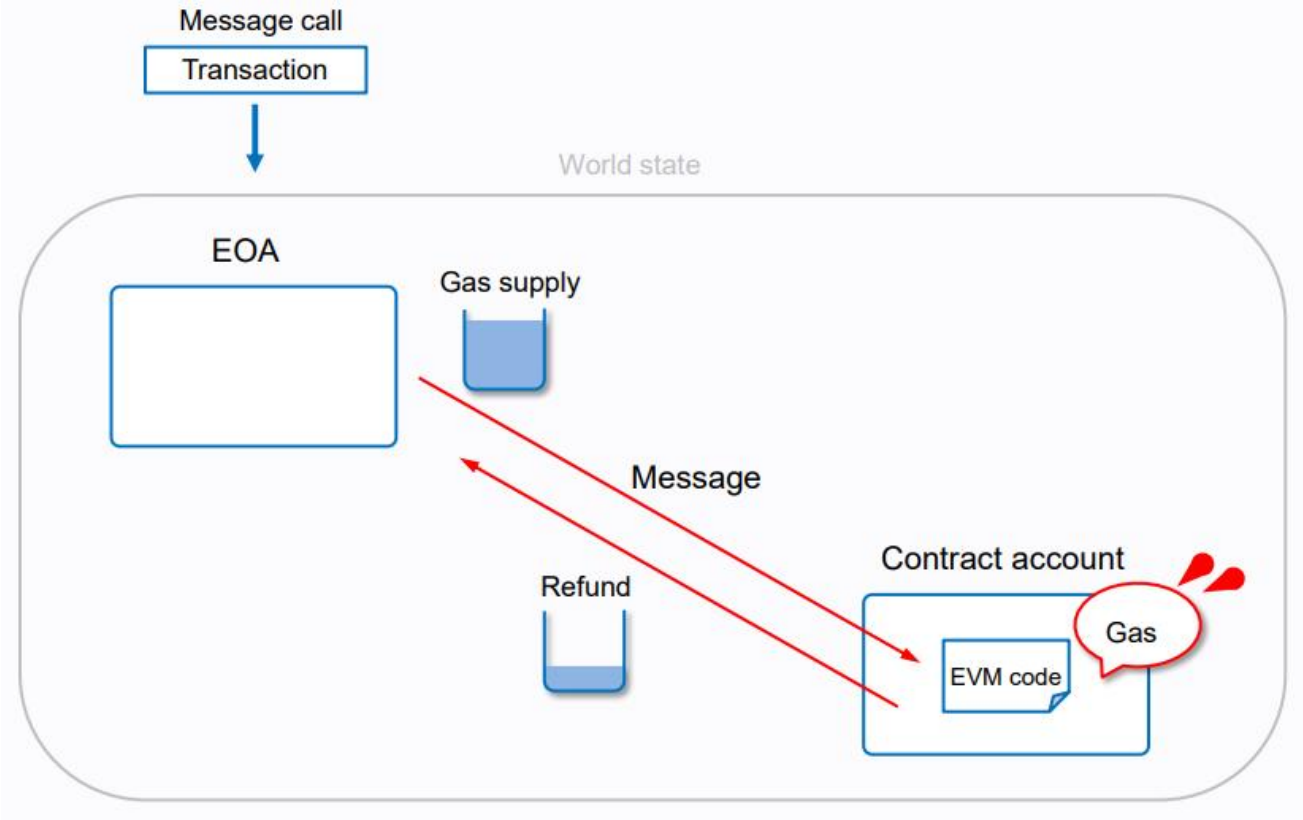


“燃料” 计数器

- 越复杂的计算 => 支付越多的 “燃料”
- 每个区块有 “燃料” 的上限

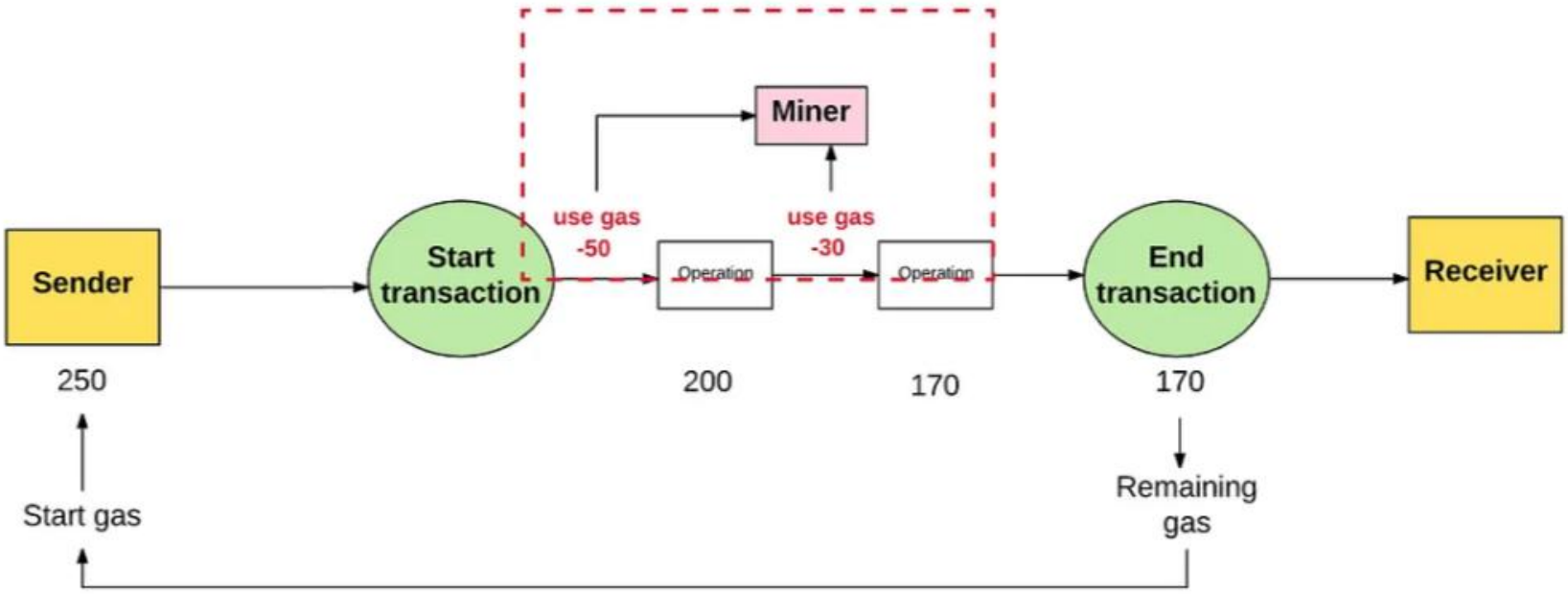
燃料机制

- **调用合约的外部拥有账户 (EOA) 负责支付 燃料费**
- **如果 燃料费 有剩余，会返回给支付 燃料费 的 EOA**
- **如果 燃料费 不足，“燃料” 计数器会抛出 out-of-gas 异常**



燃料机制

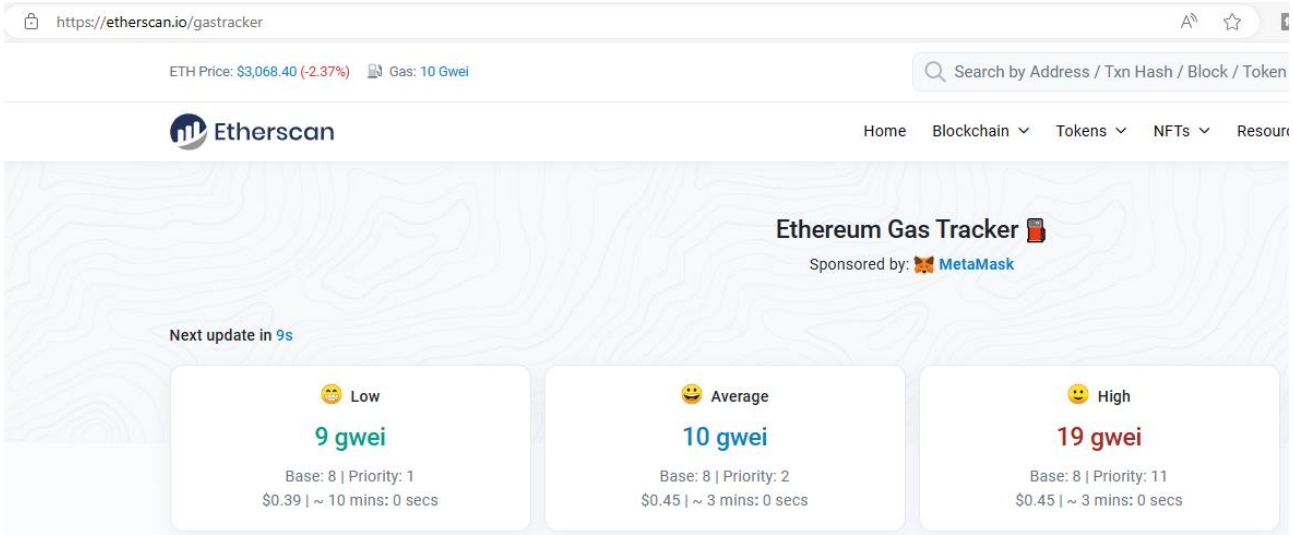
- 调用合约的外部拥有账户 (EOA) 负责支付 燃料费
- 如果 燃料费 有剩余，会返回给支付 燃料费 的 EOA
- 如果 燃料费 不足，“燃料” 计数器会抛出 out-of-gas 异常
- **燃料费最后激励给区块的发现者：**



燃料机制

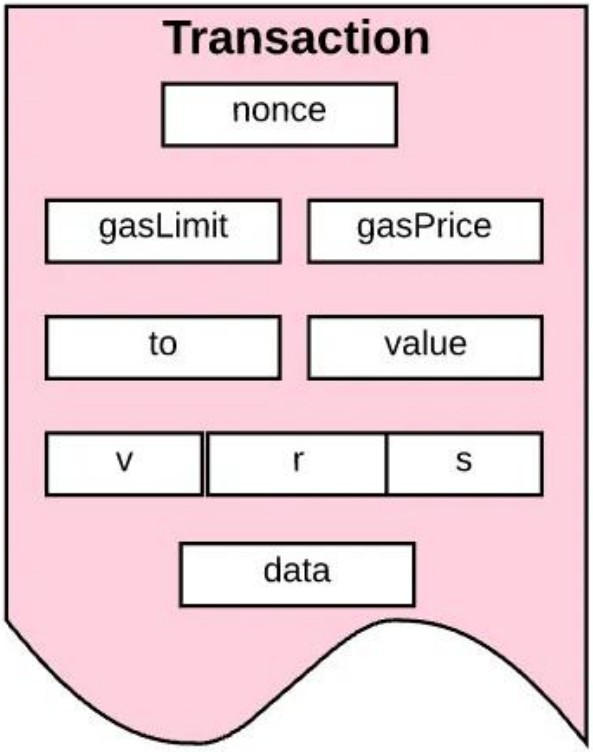
- **EOA账户 实际支付的燃料费 还要考虑 燃料价格**
  - 燃料量 * 燃料价格 = 交易消耗的 ETH 数量

  - 比如，如果使用 20 gwei，那么 50000 gas 就相当于 0.001 Ether
  - 矿工会优先处理 燃料价格 更高的交易 (gas price 形成市场平衡)



具体的交易结构

- **nonce**: 发送者生成交易数量的计数
- **gasPrice**: 愿意支付的燃料价格
- **gasLimit**: 制定的交易燃料上限
- **to**: 收款地址 (如果是一个合约创建交易，那么该地址为空)
- **value**: 以Wei为单位的转账额度
- **v, r, s**: 发送账户的数字签名
- **data**: 消息调用的数据(比如通过编译的EVM合约代码)



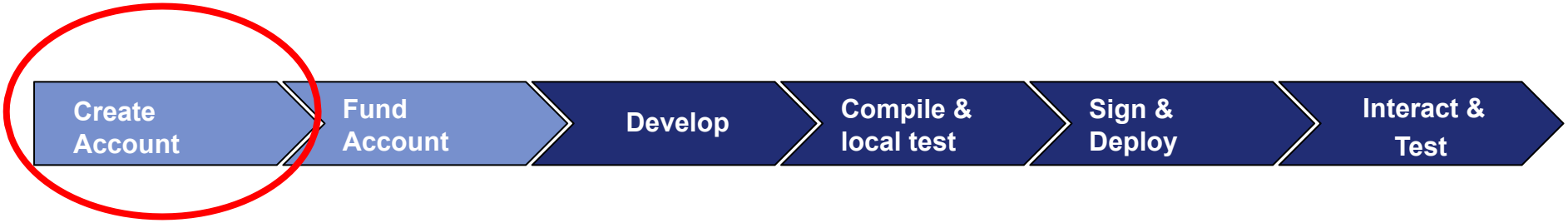
Recap

以太坊的组成部分：

- 账户：普通账户 、合约账户
- 状态：nonce、 balance、 storageRoot、 codeHash
- 交易：nonce、 gasPrice、 gasLimit、 to、 value、 (v, r, s)、 data
- 区块：state trie、 tx trie、 receipt trie 等等
- 交易执行：EVM虚拟机
- 燃料：交易在EVM虚拟机中执行时需要支付的费用
- 工作量证明：难度低于BTC
- .....

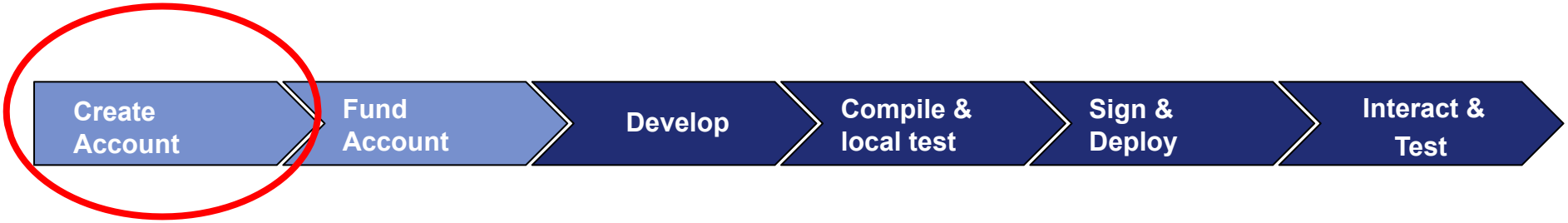
以太坊Dapp开发

开发周期



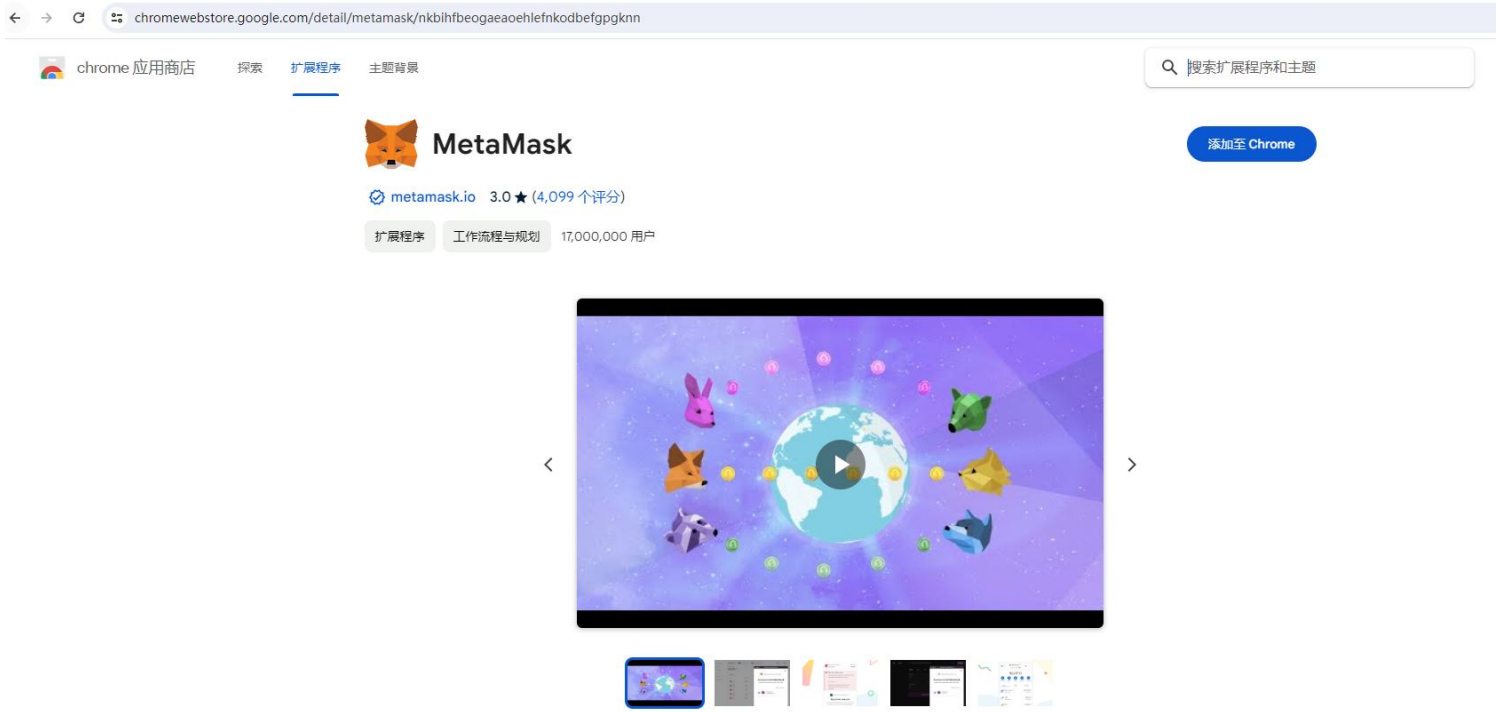
- Programmatically: Go, Python, C++, JavaScript, etc.
- Tools
  - MyEtherWallet.com
  - MetaMask
  - TestRPC
  - Many other websites

开发周期



以 metamask 为例：

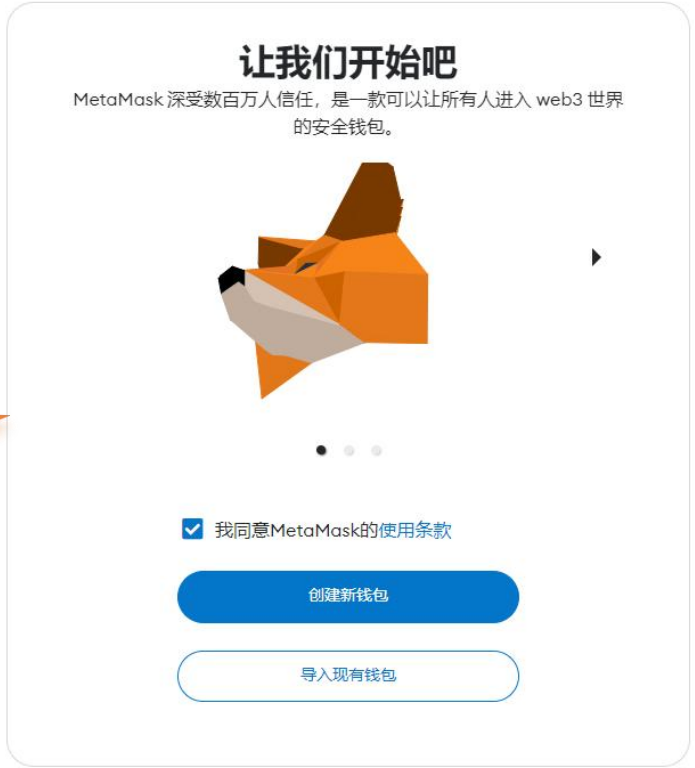
1） 下载 插件 或者 app



开发周期

以 metamask 为例：

2） 创建新钱包



思考：如果选择导入已有钱包，应该输入什么？

开发周期

Create Account → Fund Account → Develop → Compile & local test → Sign & Deploy → Interact & Test

METAMASK

以 metamask 为例：

3） 选择口令

思考：这里选择的口令是用来导出以太坊账户的私钥么？还是其他用途？

开发周期

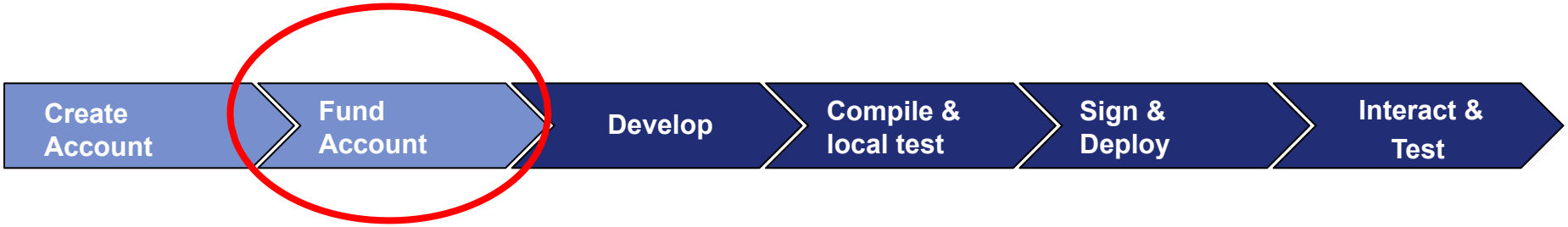
以 metamask 为例：

4）产生助记词

5）产生账户

思考：怎么从助记词中产生账户私钥呢？

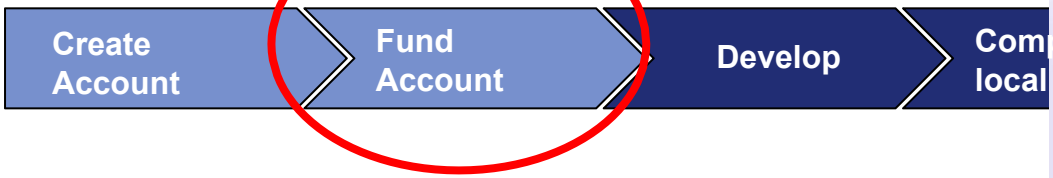
开发周期



- From friends/peers/community
- Faucet
- etc.

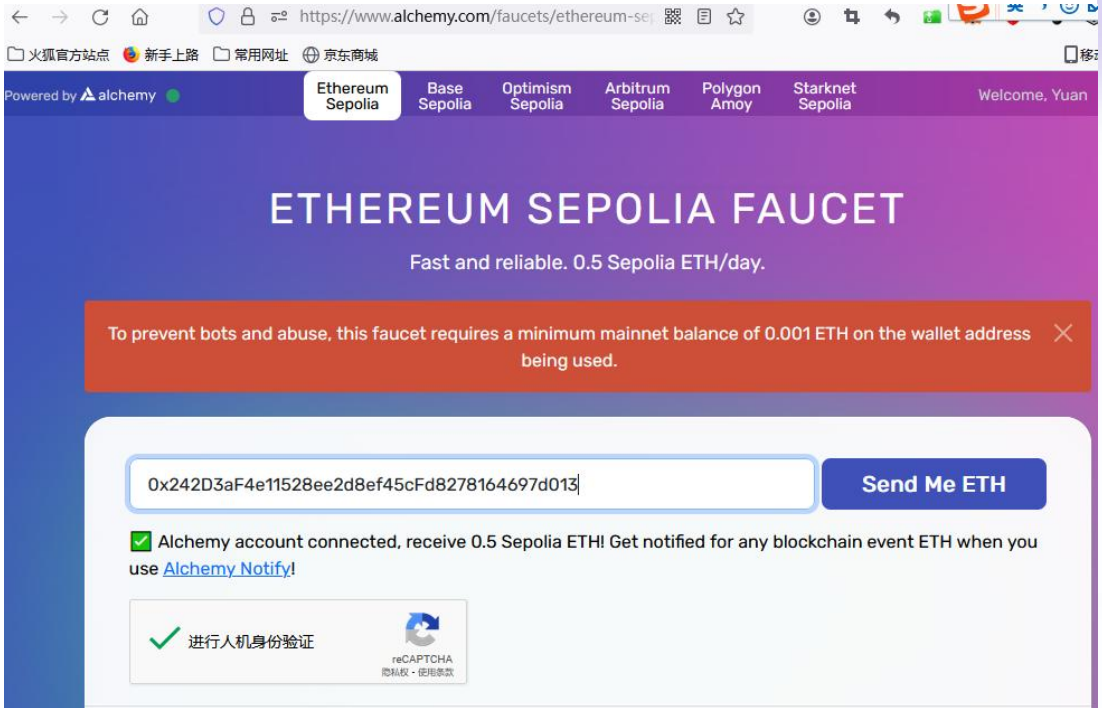
思考：为什么开发Dapp一定需要一定量的Ether?

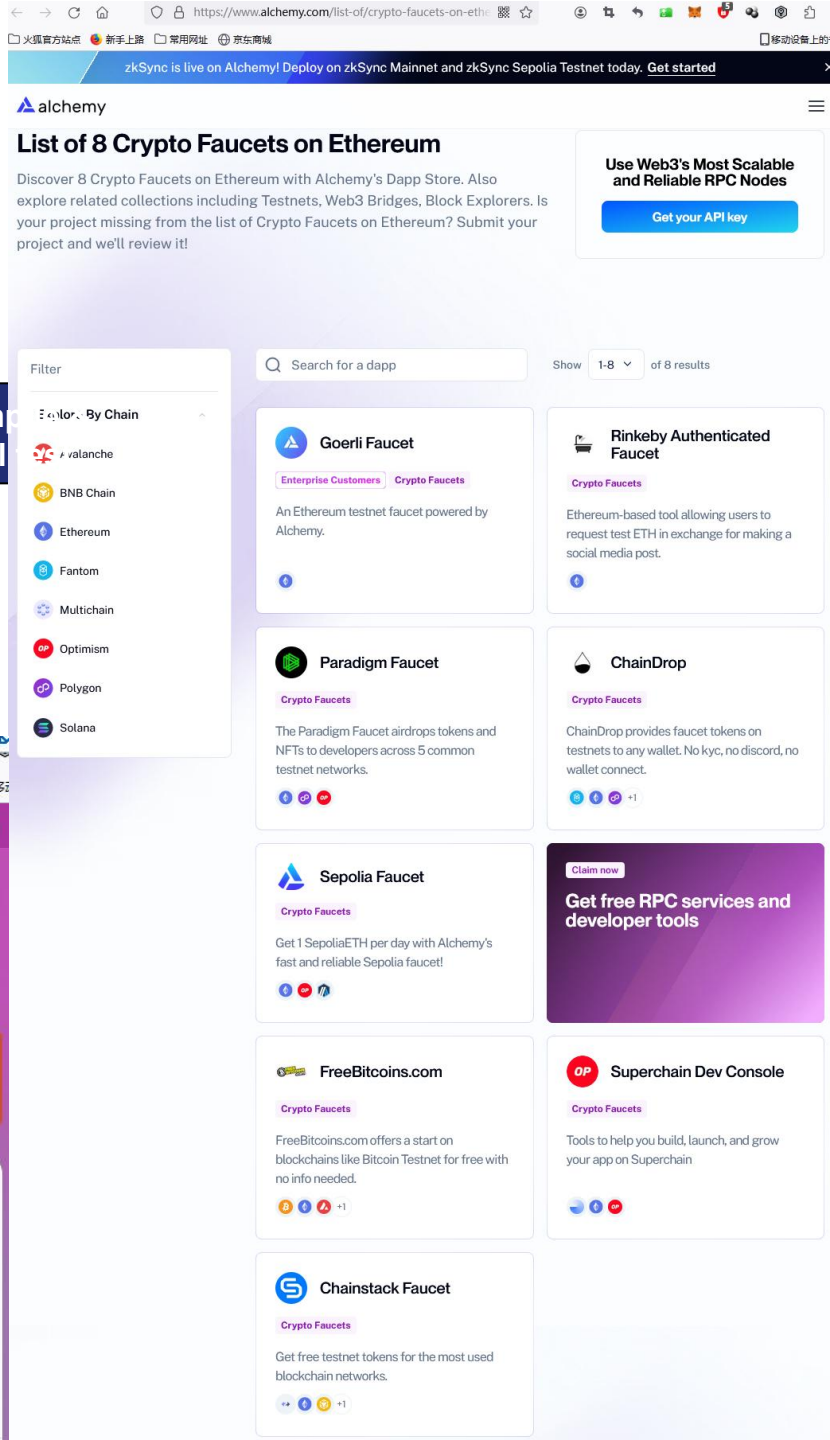
开发周期



测试链水龙头：

1) 在测试链水龙头输入自己的地址



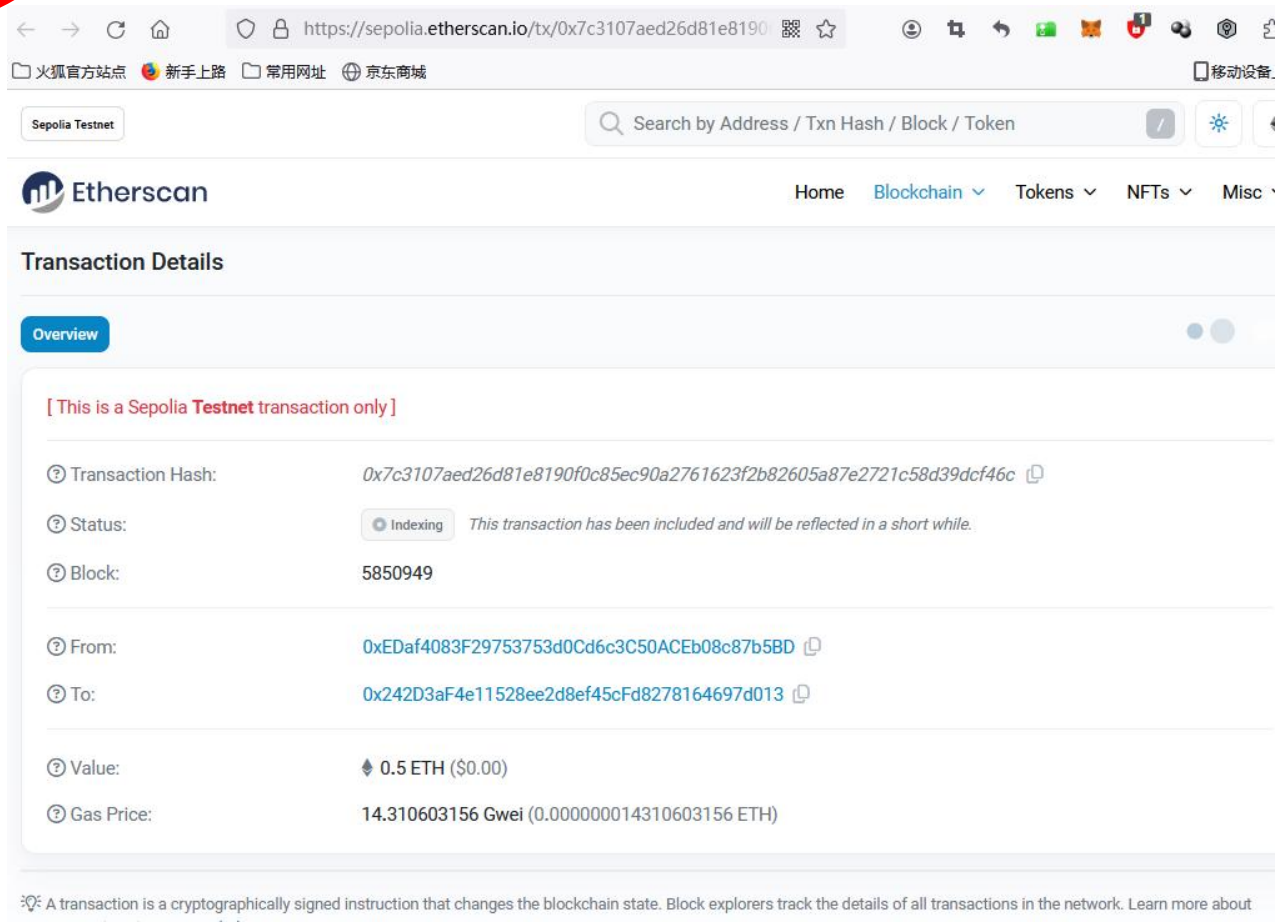


开发周期

Create Account → Fund Account → Develop → Compile & local test → Sign & Deploy → Interact & Test

测试链水龙头：

2) 确认自己已经收到的水龙头的转账



开发周期

- **Ethereum Application Components:**
  - **Backend application**: theoretically can be developed in **any** language
  - **Smart contract**: developed in Solidity or one of the other contract compatible languages
  - **Connector library***: facilitates communication between base application and smart contracts (Metamask)

*Library that facilitates communication and connection with Blockchain; Connects your code to a running node.

开发周期

智能合约 (以solidity为例):

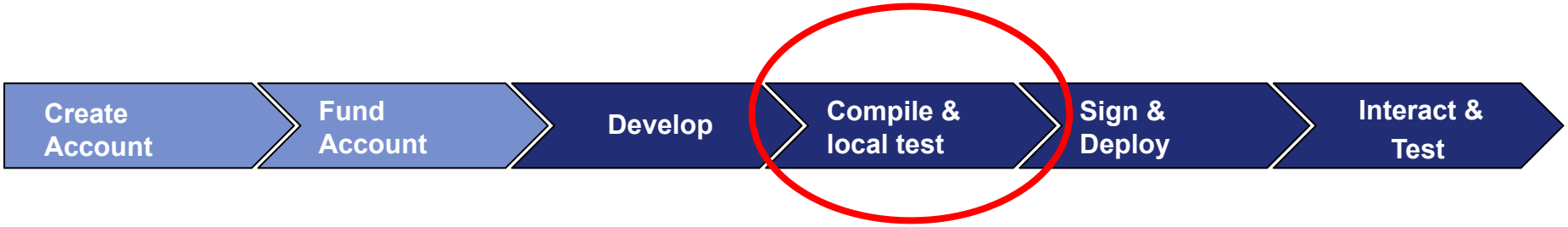
新建一个变量

可以通过setValue设置该变量

也可以本地查询或通过getValue读取该变量

```
pragma solidity ^0.8.25;

contract Example {

uint public value;

function setValue(uint pValue) public {
    value = pValue;
}

function getValue() public returns (uint) {
    return value;
}

}
```

开发周期




```
pragma solidity ^0.8.25;

contract Example {

uint public value;

function setValue(uint pValue)
public {
    value = pValue;
}

function getValue() public returns
(uint) {
    return value;
}

}
```

solc

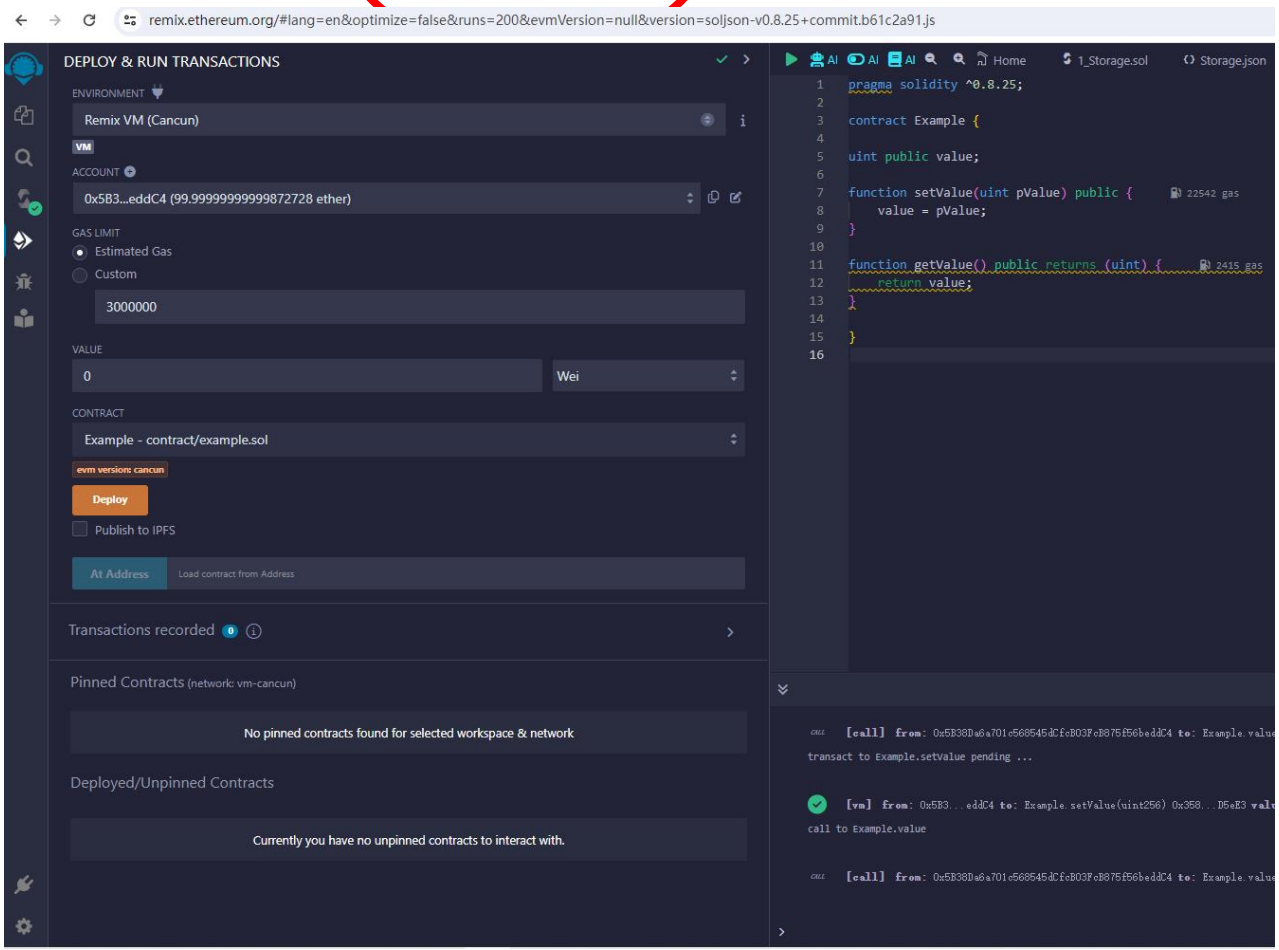
"608060405234801561000f575f80fd5b506101718061001c5f395ff3fe608060405234801561000f575f80fd5b50600436106100 3f575f3560e01c80632096525514610100435780633fa4f245146 1006157806355241077146100 7f575b5f80fd5b61004b61009b 565b604051610058919061 00c9565b604051809 10390f35b61 00696100a3565b604051610076919061 00c9565b6040518091 0390f35b6100996004803603810190610094919061 0110565b 6100a8565b005b5f8054905090565b5f5481565b805f8190555 050565b5f819050919050565b6100c3816100b1565b8252505 0565b5f6020820190506100dc5f8301846100ba565b9291505 0565b5f80fd5b6100ef816100b1565b8114610 0f9575f80fd5b5 0565b5f8135905061010a816100e6565b92915050565b5f602 0828403121561012557610124 6100e2565b5b5f6101328482 8 5016100fc565b9150509291505056fea2646970667358221220 1d0cfb39ebb59ba9a70f631c8333262efbf72c6cb0560a401727 504b3bbcb54364736f6c63430008190033"

开发周期

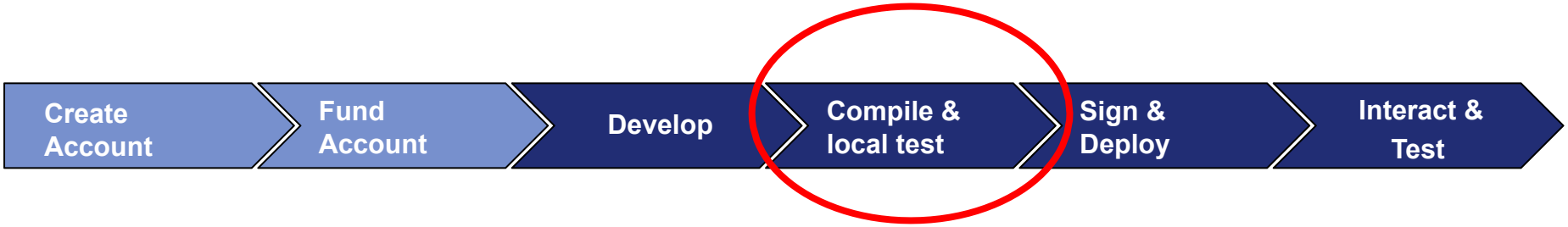
Create Account → Fund Account → Develop → Compile & local test → Sign & Deploy → Interact & Test

以Remix为例：

1) 点击部署

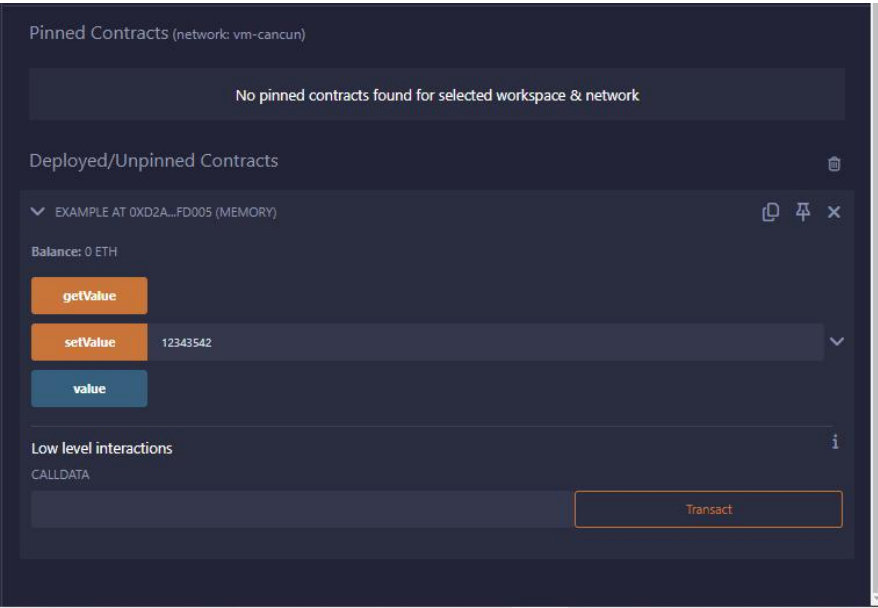


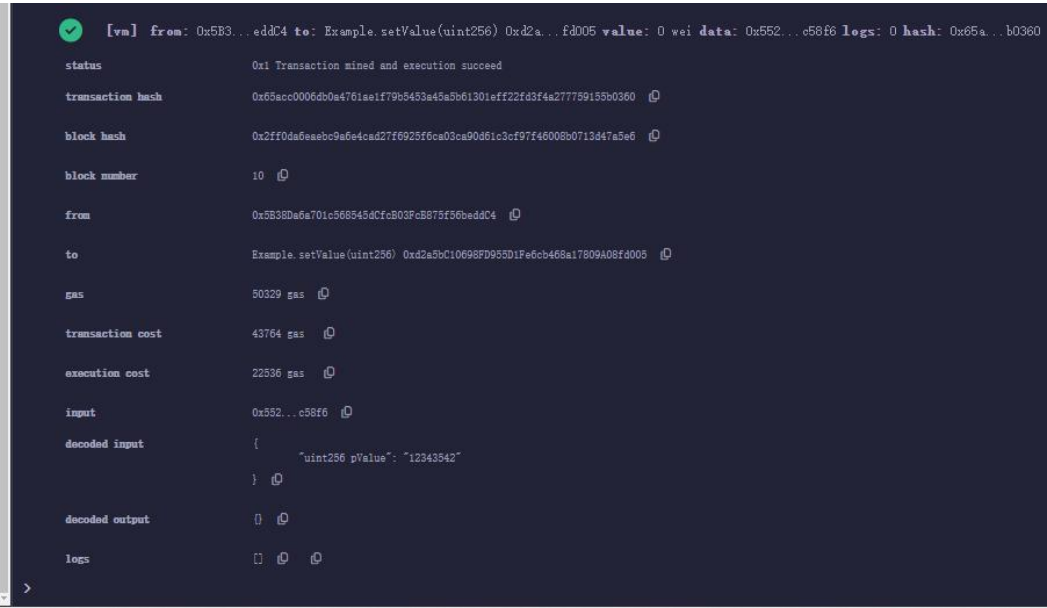
开发周期



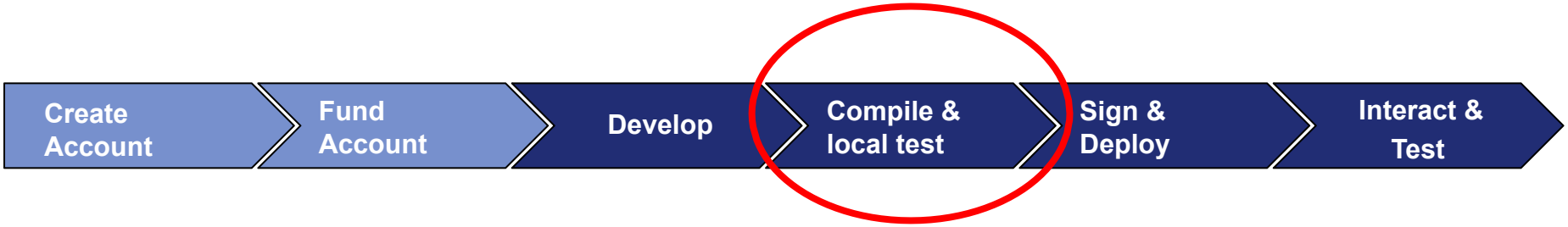
以Remix为例：

2) 测试 setValue 函数



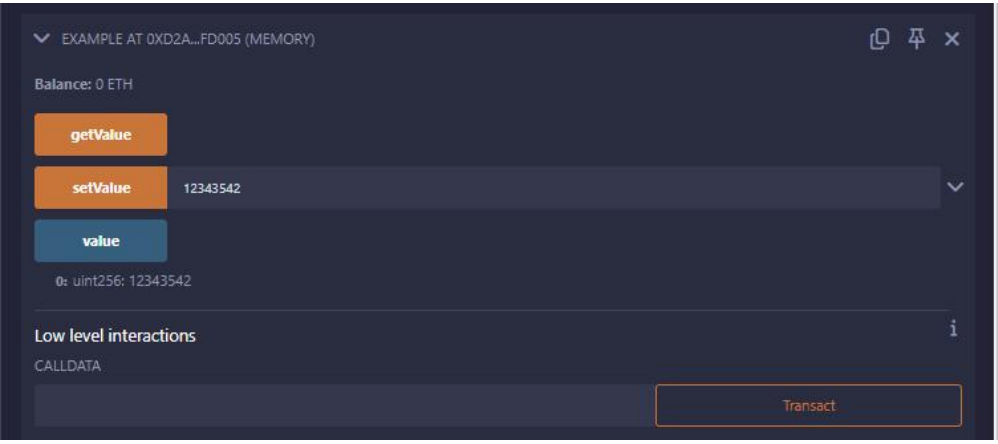


开发周期

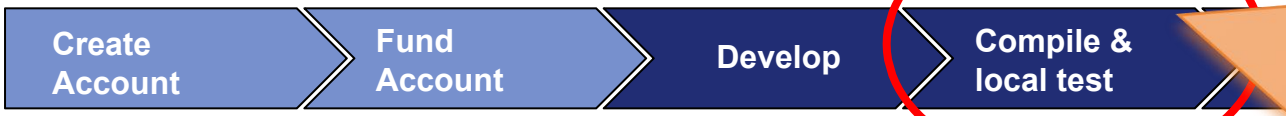


以Remix为例：

3) 检查 value，查看 setValue 是否成功



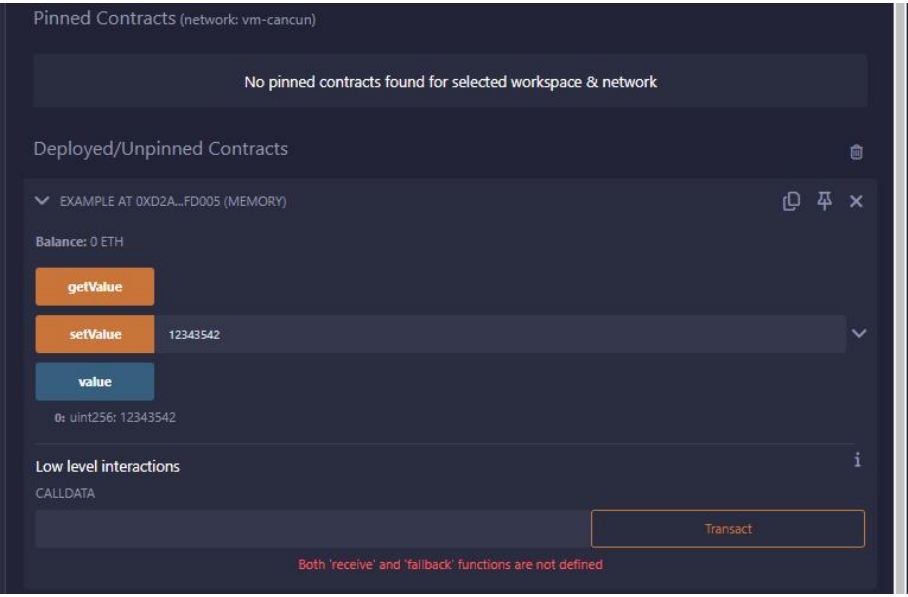
开发周期

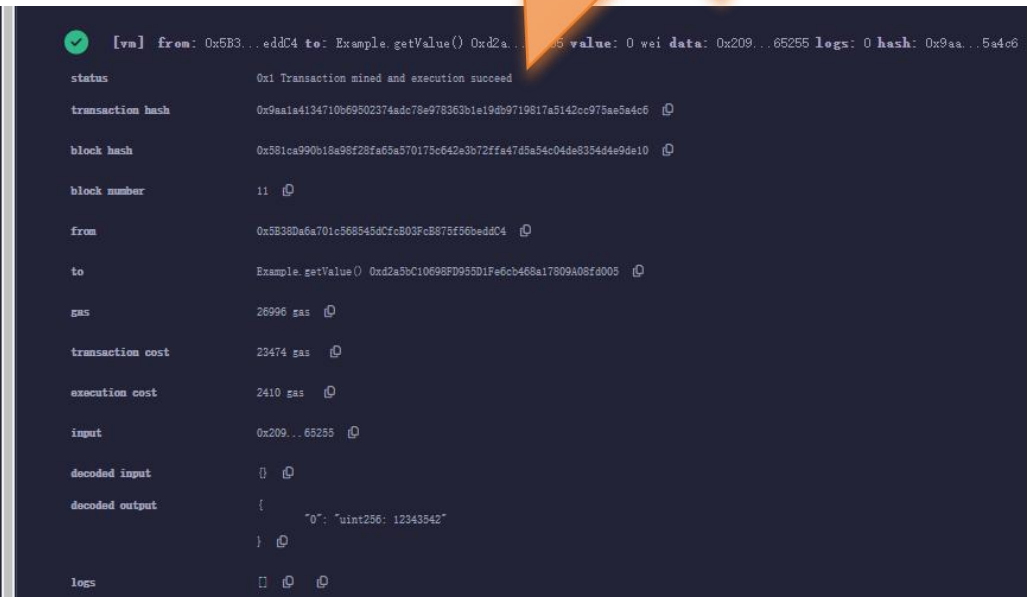


思考：本地直接读取 value 和调用 getValue 有什么区别?

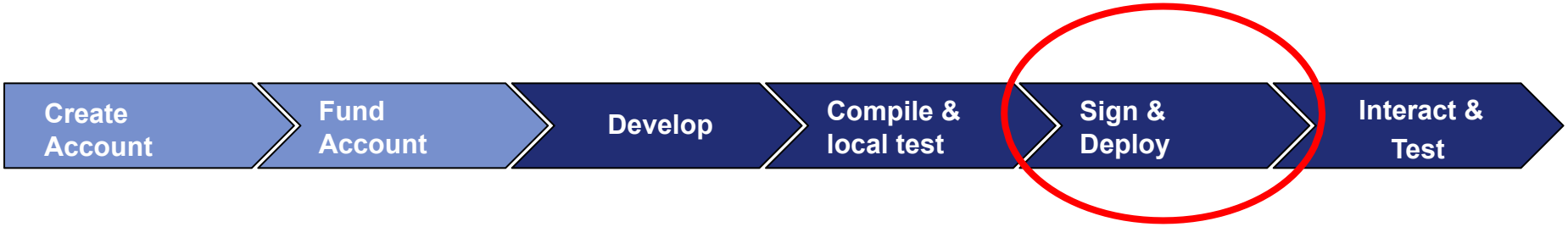
以Remix为例：

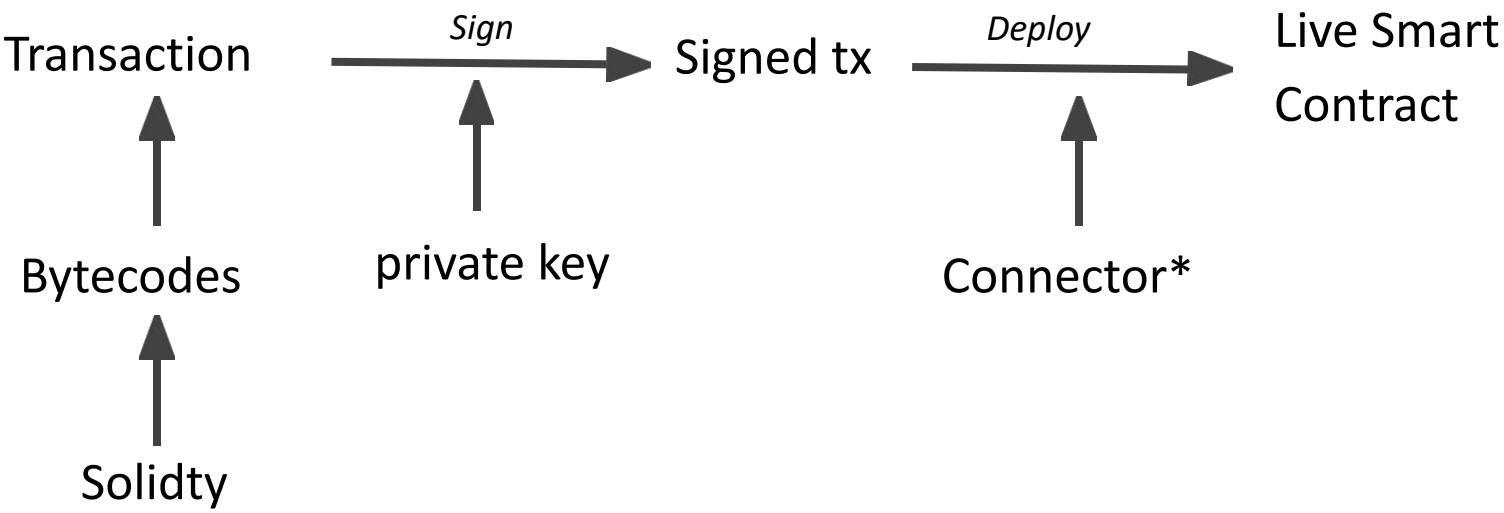
4) 检查 getValue 的返回值是否正确





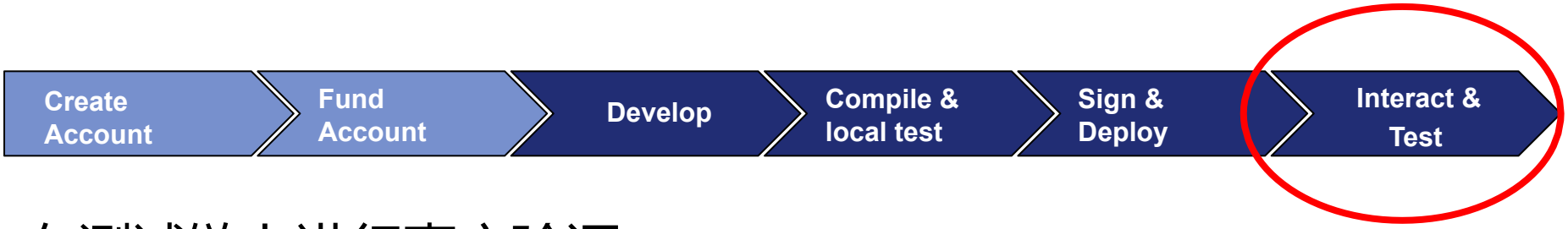
开发周期





*Library that facilitates communication and connection with Blockchain; Connects your code to a running node.

开发周期



在测试链上进行真实验证：

部署合约

验证合约的功能和每一个接口

验证gas成本是否合理

开发案例

以太坊的世界状态只记录最近256个区块的哈希值，实现一个合约，能够记录历史上所有区块的哈希值

链上

- https://github.com/yylluu/eth-blockhashes

链下

- https://github.com/yylluu/eth-blockhashes-backend

谢谢