

DAY2：以太坊核心概念

登链社区 - Tiny熊

回顾

区块链价值

基于人/组织的信任 → 基于代码的信任

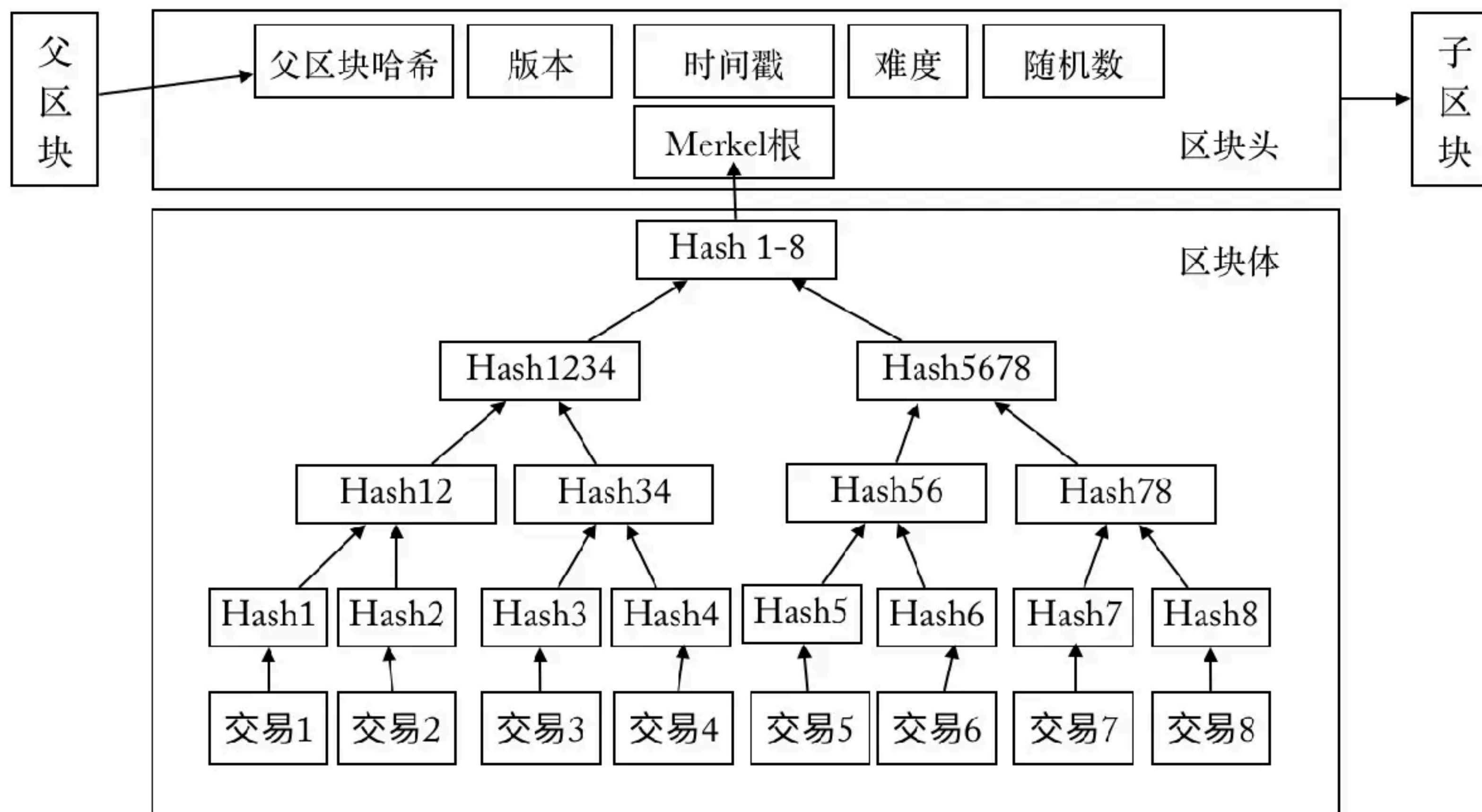
区块链解决信任的技术

Do not trust , verify.

区块链

- ❖ 私钥控制数据所有权
- ❖ 通过 hash 串联（链式）的区块结构
- ❖ 一个由多个独立节点（按某个共识）组成的网络

什么是区块链



区块链意义

- 基于代码信任 -> 透明、信任
- 无法篡改的历史、无法抵赖 -> 可追溯

以太坊

- 比特币：去中心化货币
- 以太坊：去中心化应用平台

以太坊

```
{  
  from: '0x...',  
  to: '0x...', //  
  value: 100,  
  (input)data: "0x06661abd"    // 运行代码  
}
```


什么是以太坊

- 一台世界计算机（去中心化，任何人都可使用）
- 一个状态机（由交易触发的状态转换系统）
- 一个智能合约平台（计算平台）

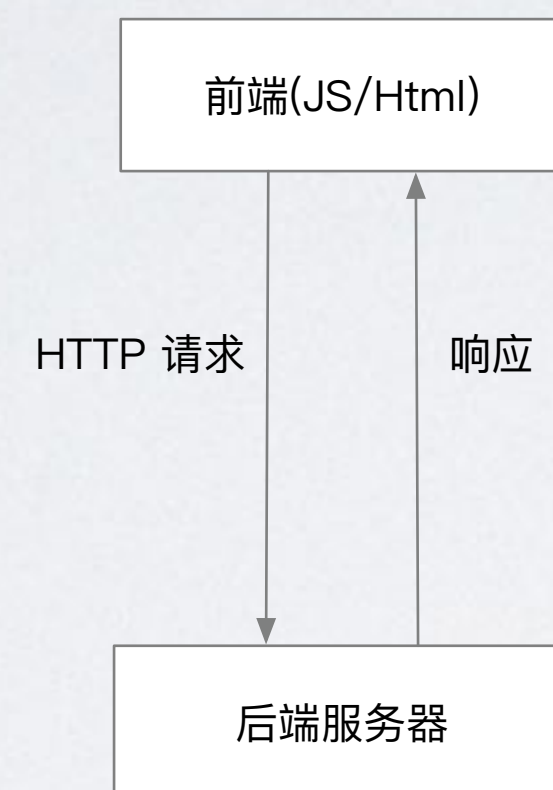
智能合约

- 智能：可执行（独立性、不受干扰）
- 合约：协议、规则

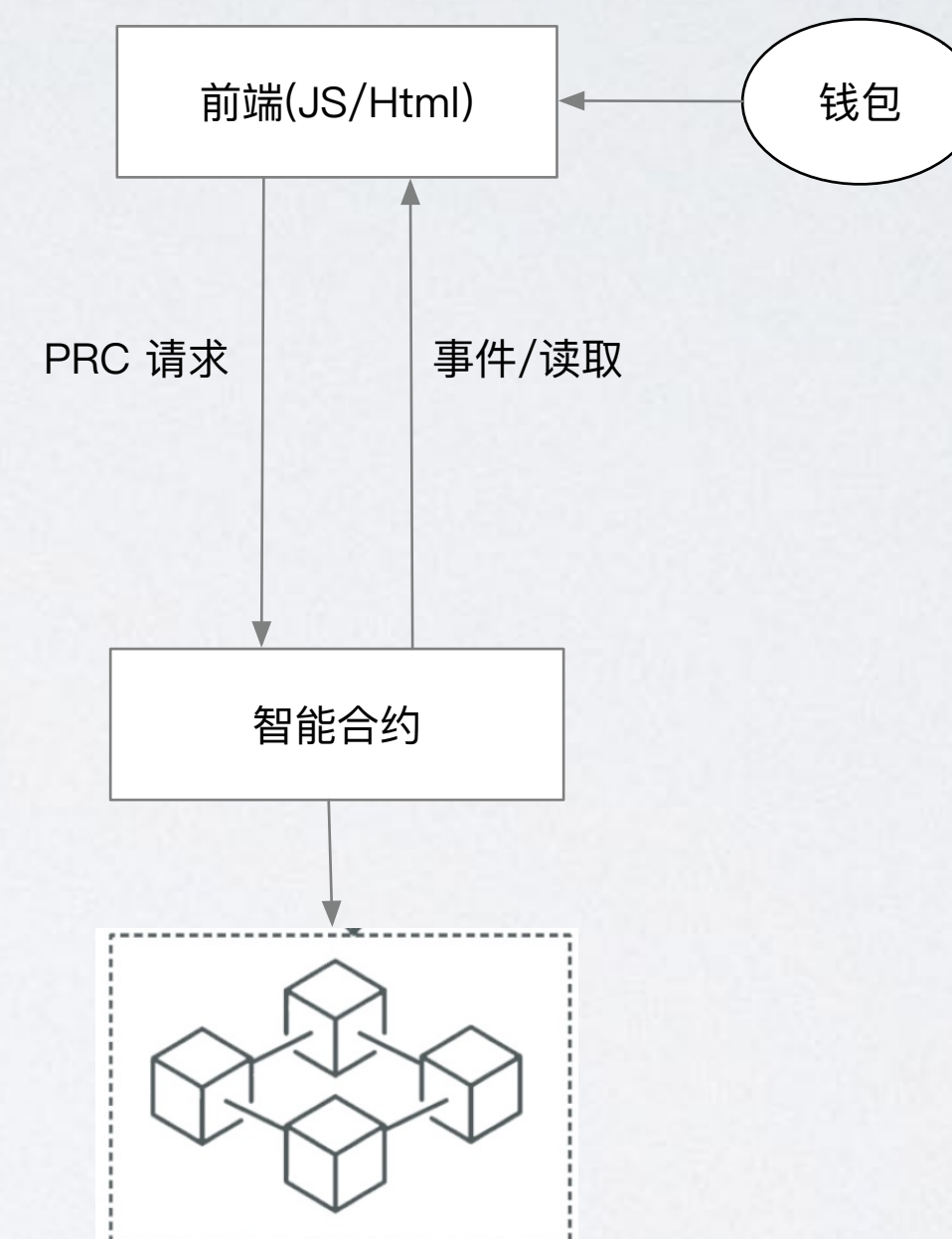
链上执行的程序，是代码和数据（状态）的集合

WEB3应用中的智能合约

传统应用



Web3 应用



核心逻辑使用智能合约运行在区块链的上，实现去信任。
用户数据通过钱包由用户自己管理。

智能合约

- Solidity(.sol): 智能合约开发语言

- 右边是一个简单的计数器
- Counter: 合约状态变量, 保存在链上
- count(): 合约函数

```
pragma solidity ^0.8.0;
```

```
contract Counter {  
    uint public counter;
```

```
    constructor() {  
        counter = 0;  
    }
```

```
    function count() public {  
        counter = counter + 1;  
    }
```

```
}
```

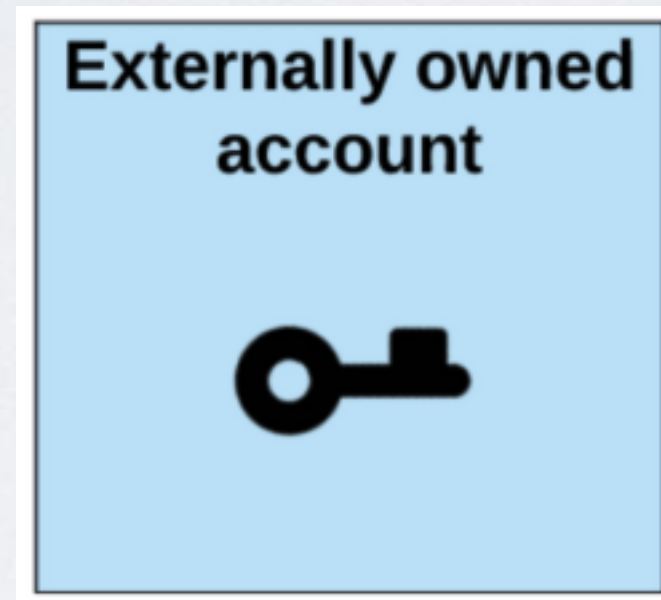

智能合约代码结构

- 编译器声明: pragma
- 合约声明: contract/ interface
- 状态变量（注明类型），类型定义
- 函数
- 事件
- 错误定义 (Error)

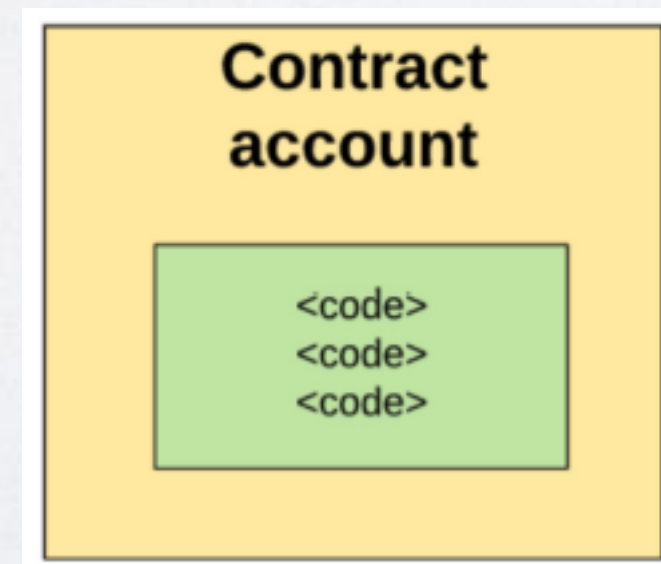
```
pragma solidity ^0.8.9;
contract Counter {
    uint constant counter;
    constructor() {
        counter = 0;
    }
    function count() public {
        counter = counter + 1;
    }
}
```


账户

- 外部账户 (EOA) : 由私钥控制, 妥善保管、不可恢复



- 合约账户: 代码控制



都用 20 个字节表示:
0xea674fdde714fd979...

账户

- 外部账户（EOA）与 合约账户在 EVM 层面是等效的，都是有：nonce（交易序号）、balance（余额）、storageRoot（状态）、codeHash（代码）

账户结构



Smart Contract

Ethereum Account Type (Just like User Account)



Address

0x16E0022b17B...



Balance

0 Ether



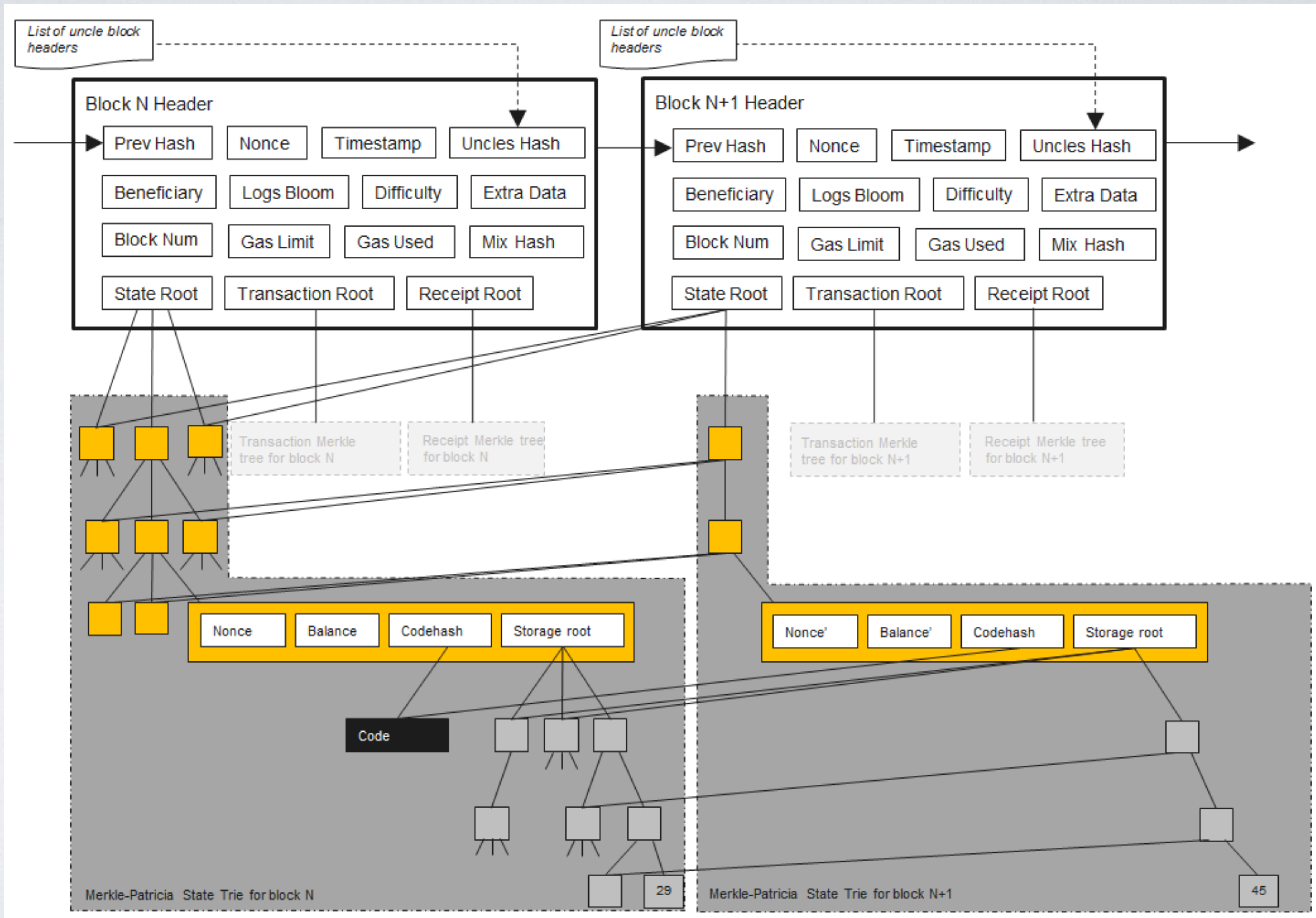
Code



State

```
contract Counter {
    uint counter;

    function Counter() public {
        counter = 0;
    }
    function count() public {
        counter = counter + 1;
    }
}
```

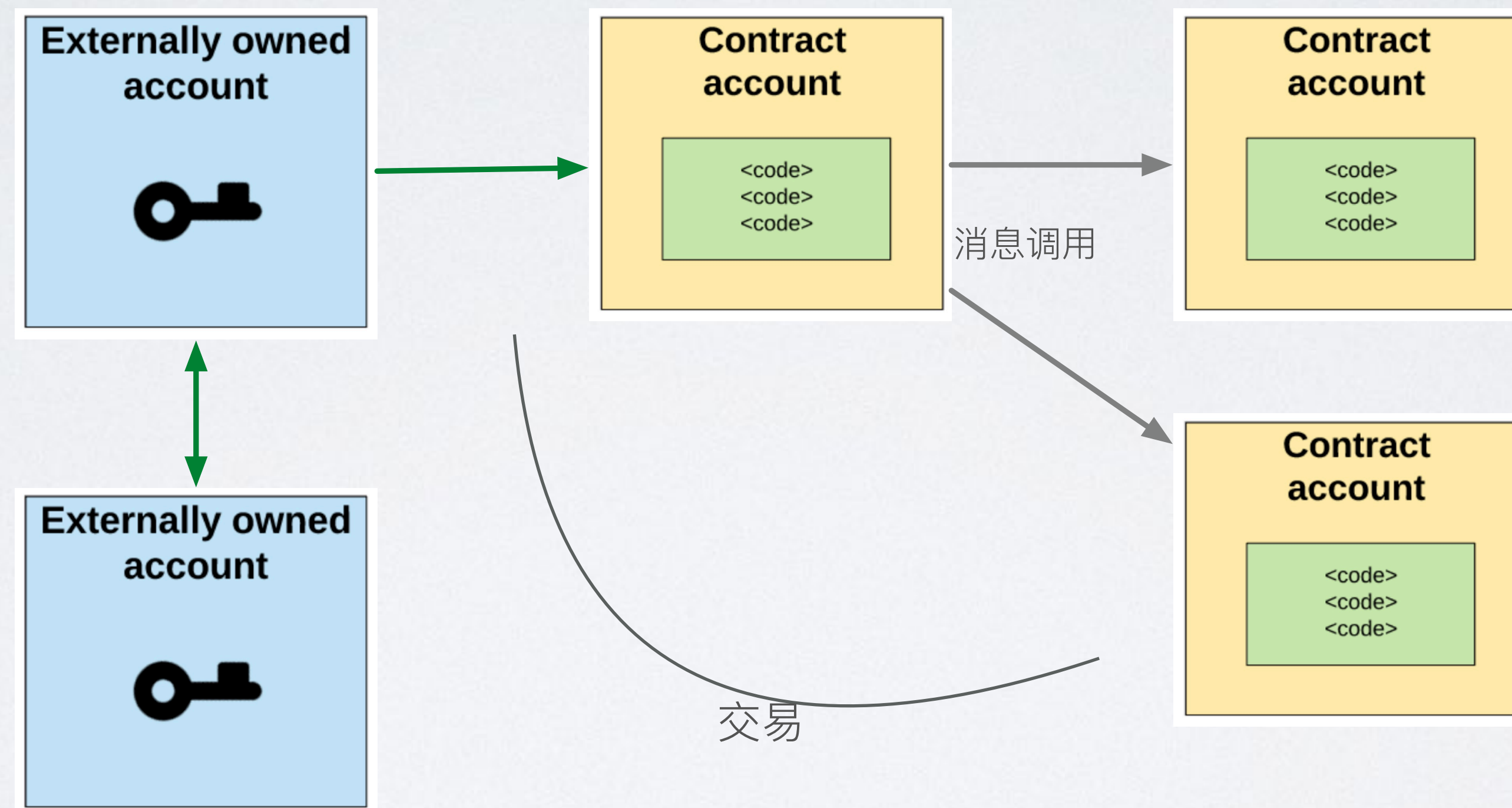
账户

- 但是在表现上有不一样：
 - **交易**只能从外部账号发出，合约只能被动相应执行。
 - 合约之间的交互通常称为**消息**，所有的手续费 gas 只能由外部账号支付。

账户交互

1. 由 EOA 支付Gas
2. 一次发起一笔交易

合约账户只能被动响应、没有自动运行



交易： 原子性

以太坊 三种交易

- 普通交易
- 创建合约
- 调用合约

```
{
  to: '0x687422...',
  value: 0.0005,
  data: "0x" // 也可以附加消息
}
```

```
{
  to: '',
  value: 0.0,
  data: "0x606060405234156100057x106....."
}
```

```
{
  to: '0x687422eEA2cB73..', //合约地址
  value: 0.3,
  data: "0x06661abd"
}
```


EVM

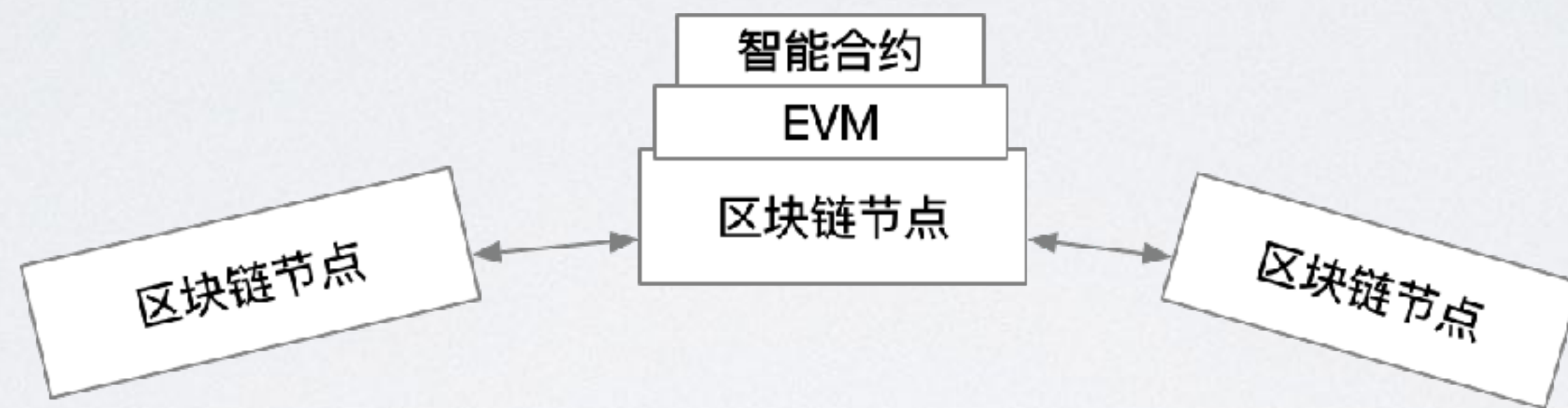
- EVM: 以太坊虚拟机, 智能合约执行环境
 - 类似 Java 至于 JVM
 - EVM 是一个封闭环境 (不可访问外部系统)

以太坊强大的生态催生出来很多 EVM 兼容链: BSC , Polygon, OEC, Fantom ...

以太坊客户端

- 以太坊客户端：EVM 载体、网络中的节点程序
- 只要符合共识-规范，（几乎）任何语言都可以实现客户端
- 常见的客户端（TheMerge 之后）
 - 执行层：Geth（Go 实现）、Nethermind(C#实现)、Erigon（go）
 - 共识层：Prysm（Go）、Lighthouse（Rust 实现）
- 通过 RPC 提供服务
 - 节点服务商: Infura, alchemy

EVM 与 节点



钱包

- 账号管理工具，进行签名发起交易（管理助记词、私钥）

- 钱包：

- Metamask 外号：小狐狸（插件、App）



METAMASK

- imtoken

- TrustWallet

下载时仔细查看 URL，谨防钓鱼

GAS 机制

- EVM 的计价规则，也防止图灵死机问题。
- GAS 是一个工作量单位，EVM 规范里定义操作的 Gas 值，复杂度越大，所需 gas 越多。

GAS 手续费

- EIP1559 之前
 - 手续费 = $\text{gas used}(< \text{gas limit}) * \text{gas price}$ 单价 (gas limit 和 gas price 由用户指定)
 - 矿工收益 = 手续费
- EIP1559 之后
 - 手续费 = $\text{gas used}(< \text{gas limit}) * (\text{base fee} + \text{tips fee})$ (gas limit 、 max tips fee 、 max fee 由用户指定)
 - 燃烧掉 = $\text{base fee} * \text{gas used}$ (base fee 是打包时动态确认的)
 - 矿工收益 = $\text{tips fee} * \text{gas used}$

GAS 机制

- 智能合约越复杂 用来完成运行就需要越多Gas
- gas price 用户指定/Tips fee, 决定交易的排序。
- $\text{gas limit} > \text{gas used}$ 交易才能顺利执行, 否则 out of gas 交易回滚。
- 执行结束, gas limit 余下的部分不扣费用。
- Gas limit 开发工具估算

Demo

练习

<https://decert.me/quests/d17a9270-99c3-4aeb-8a46-42ecb5e92792>

以太币单位

- 最小单位: Wei (伟)
- $10^9 \text{ Wei} = 1 \text{ Gwei}$
- $10^{12} \text{ Wei} = 1 \text{ szabo}$ (萨博)
- $10^{15} \text{ Wei} = 1 \text{ finey}$ (芬尼)
- $10^{18} \text{ Wei} = 1 \text{ Ether}$

区块链浏览器

- 查看交易（交易hash、gas、）
- 查看 Token 信息
- 查看源代码

不同的网络

- 主网（价值网络） <https://cn.etherscan.com/>
- 测试网： <https://sepolia.etherscan.io/>
- 开发模拟网（本地环境）

<https://chainlist.org/>

智能合约DEMO

- Remix IDE: Solidity Online IDE

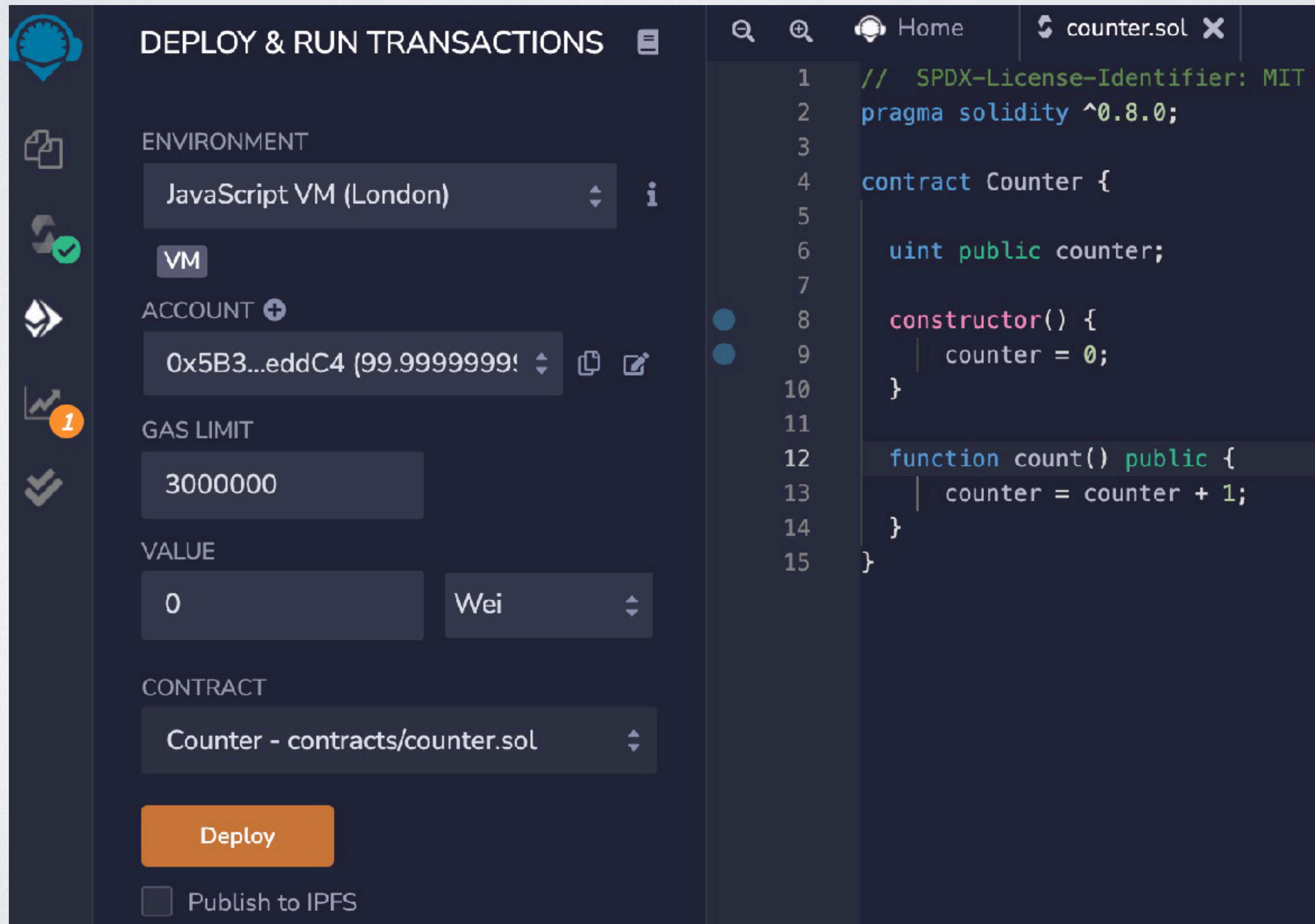
<https://remix.ethereum.org>

<https://remix.learnblockchain.cn/>

DEMO

REMIX

<https://remix.ethereum.org>



The screenshot displays the REMIX IDE interface. On the left, the 'DEPLOY & RUN TRANSACTIONS' panel is active, showing the following configuration:

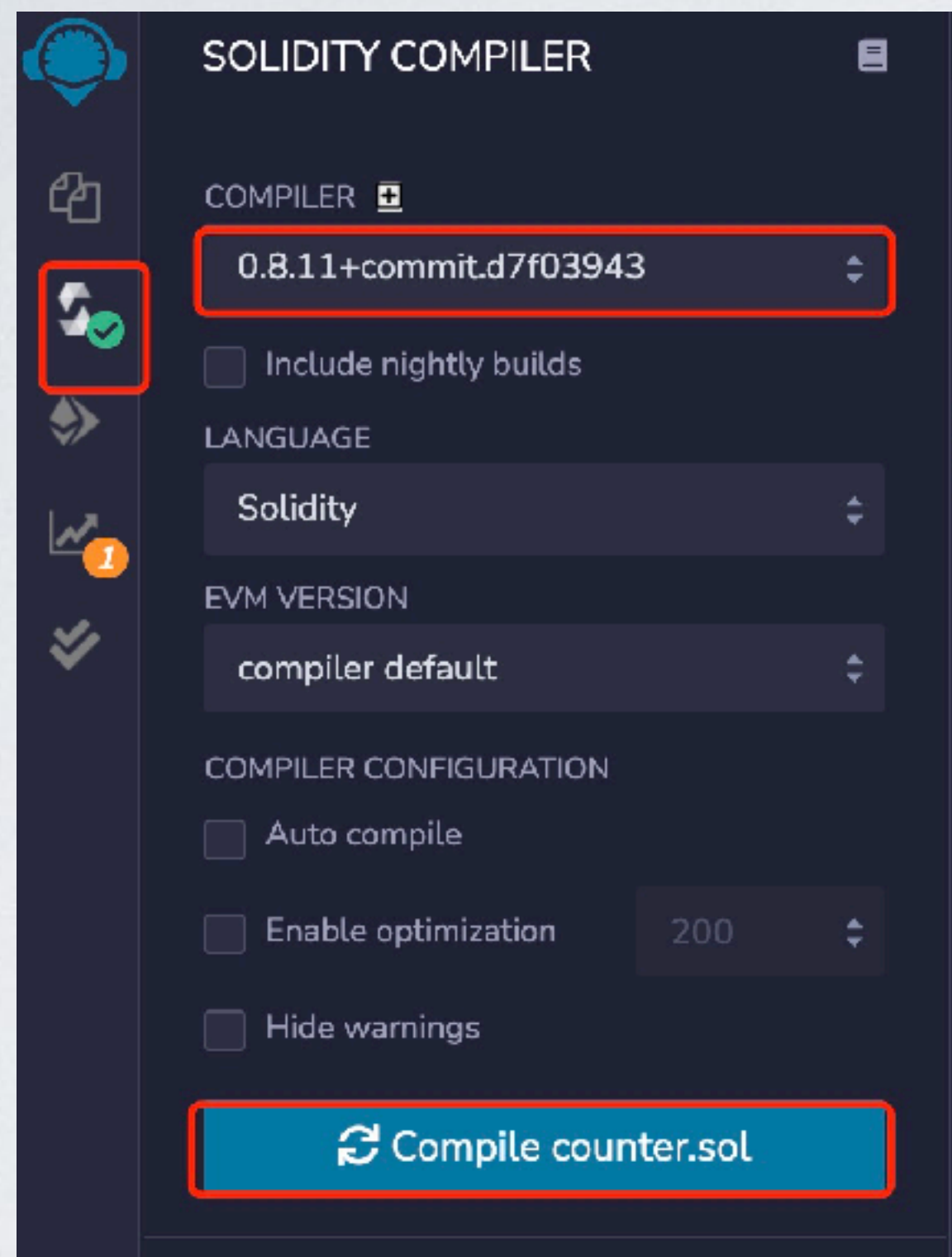
- ENVIRONMENT:** JavaScript VM (London)
- VM:** (checked)
- ACCOUNT:** 0x5B3...eddC4 (99.99999999%)
- GAS LIMIT:** 3000000
- VALUE:** 0 Wei
- CONTRACT:** Counter - contracts/counter.sol
- Deploy** button
- ☐ Publish to IPFS

On the right, the Solidity code editor shows the following code:

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Counter {
5
6     uint public counter;
7
8     constructor() {
9         counter = 0;
10    }
11
12    function count() public {
13        counter = counter + 1;
14    }
15 }
```

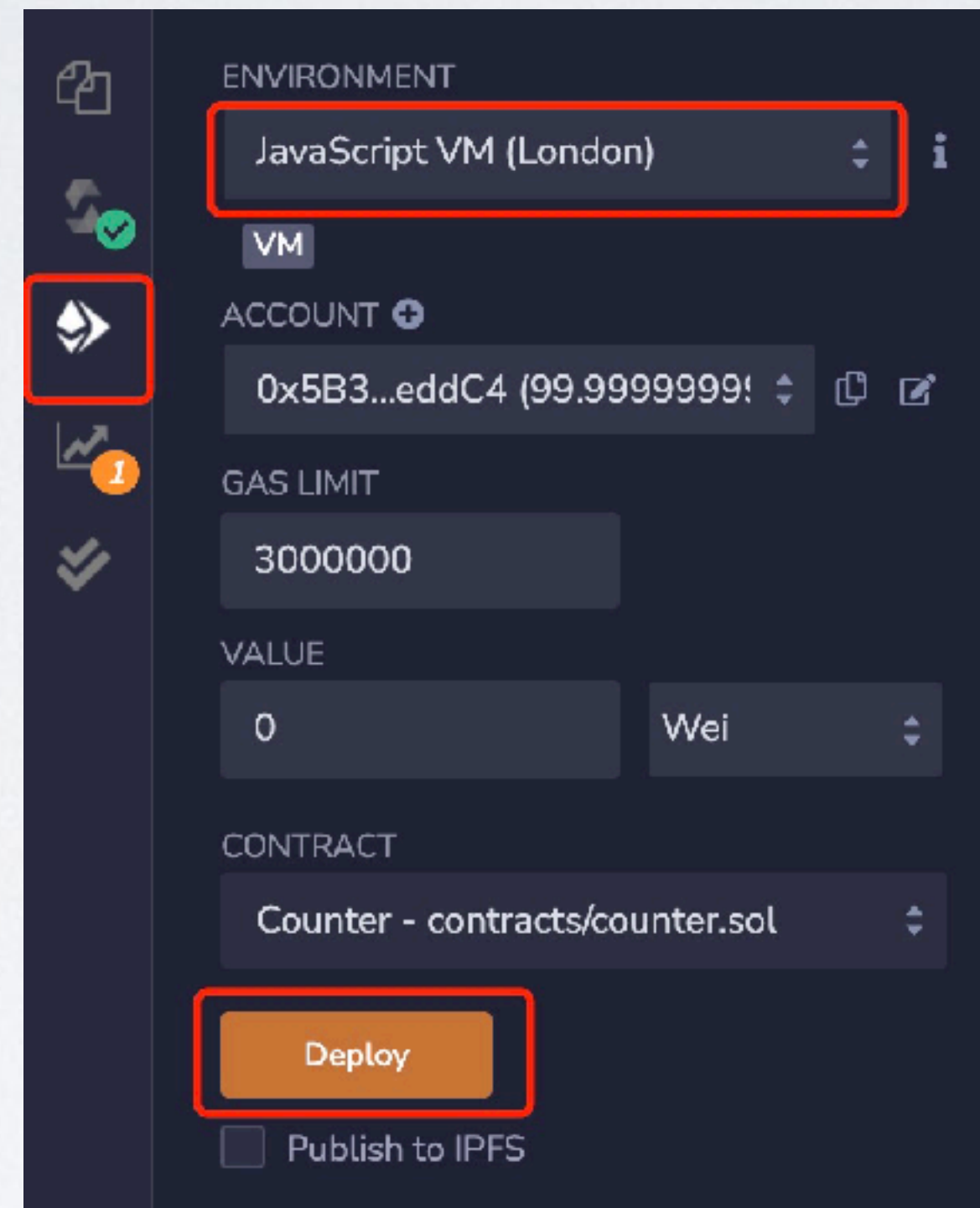

REMIX

<https://remix.ethereum.org>



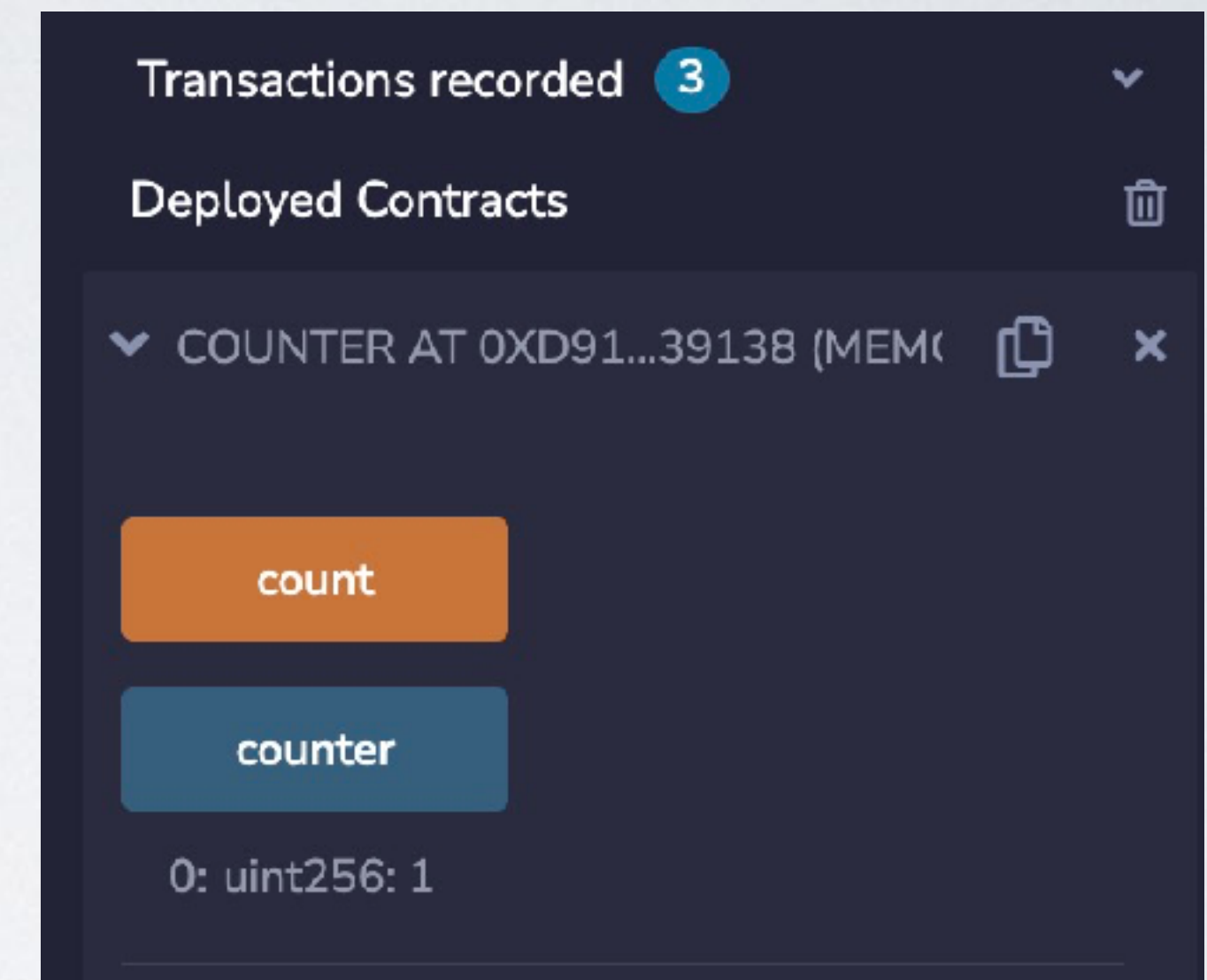
可选择编译器版本

编译 产出: ABI 与 字节码



可部署到不同的网络

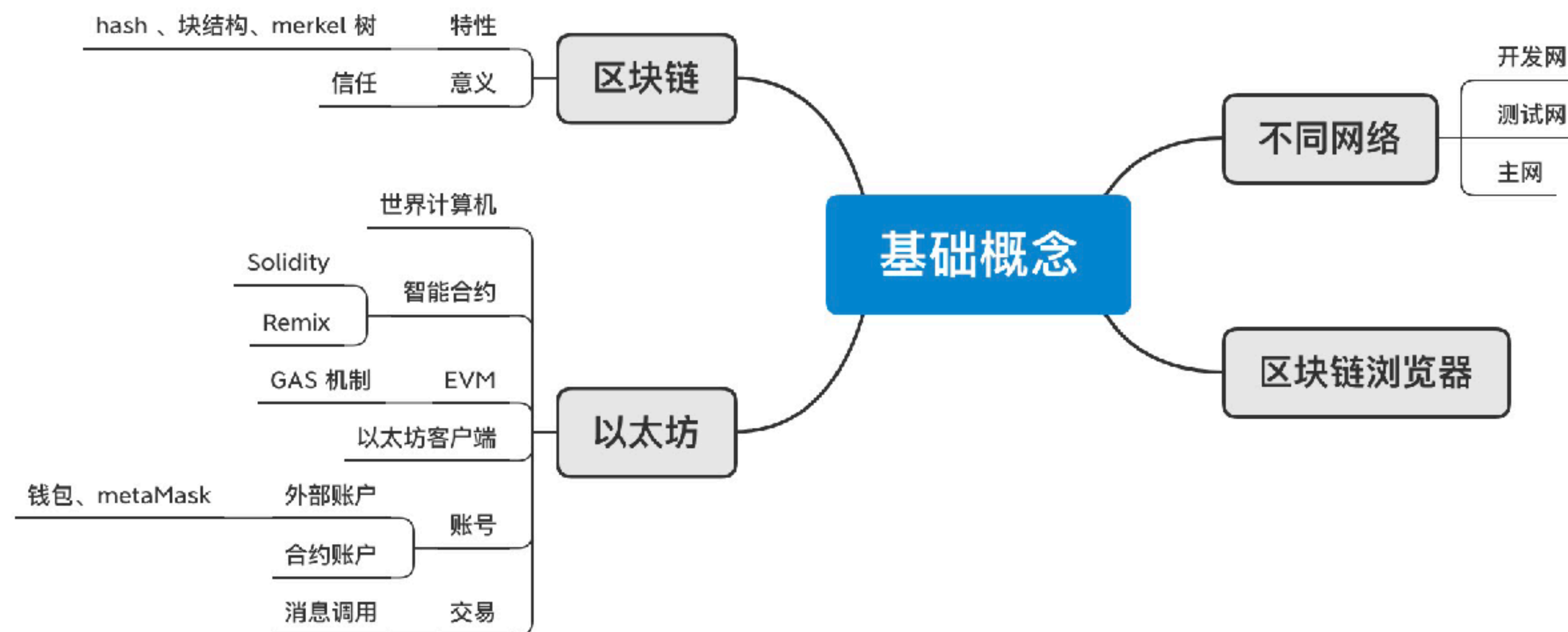
部署 产出: 合约地址



执行合约函数:
橙色: 触发交易
蓝色: 仅读取

执行

小结



Q & A

练习题

- 安装 Metamask、并创建好账号（向水龙头获取代币）：<https://chaintool.tech/faucet>
- 向同桌的地址执行一次转账
- 使用 Remix 创建一个Counter合约并部署到测试网 (<https://sepolia.etherscan.io/>):
 - Counter 合约添加一个 add(x) 方法;

<https://decert.me/quests/61289231665986005978714272641295754558731174328007379661370918963875971676821>

作业要求：

1. 使用自己的 github 创建📁 firstcontract
2. 提交代码、截图、交易 hash 等