# 集训营
# 智能合约开发培训

登链社区 - Tiny熊

# 欧科云链

- 可分布式办公
- 高福利
- 大量岗位需求

# 课程介绍

- 推荐就业
- 视频直播、实时讨论
- 课后录播巩固、习题练习
- 微信群答疑

# 要求

- 较好的计算机科班背景
- GitHub
- 命令行
- 有至少一门语言编程经验

# 区块链及智能合约平台介绍

- 以太坊平台详解：智能合约平台、GAS机制等
- 了解核心概念：账号及合约等、交易和消息调用、货币单位及Gas、钱包等
- 以一个简单的智能合约为例，介绍合约编译、部署、测试。
- 开发工具介绍 如：Remix、Hardhat、Truffle、Ganache 等

# SOLIDITY语言主要特性

- Solidity 基本类型、数组、结构体、映射
- Solidity API 介绍
- 合约函数、函数修改器、函数修饰符，及各类特殊函数
- 错误处理、合约继承、接口、库及 Openzeppelin 合约库
- 理解合约事件
- 理解ABI

# 智能合约及DAPP开发实战进阶

- 最常用 ERC 标准介绍及实战：ERC20、ERC777、EIP2612、ERC721、ERC1155
- Hardhat 实战技巧：自动化脚本、代码验证、导出 ABI 等。
- DApp 开发：前端与合约交互（ethers.js）
- 优化 GAS 的众多技巧
- 解析合约事件与TheGraph 使用

# DEX 分析专场

- Uniswap V2 AMM模型
- 无常损失与 AMM 滑点
- SushiSwap 如何抢Uni 的流动性及质押模型介绍
- Uniswap v3 的创新
- 相关 DEX 协议介绍：0x协议、Curve 协议

# 抵押借贷及衍生品协议

- Compound 借贷利率模型分析
- AAVE 闪电贷及应用
- DyDx 衍生品交易
- Perpetual vAmm 杠杆交易

# 稳定币及 DAO

- MakerDAO 稳定币DAI
- 算法稳定币介绍：FRAX FEI OHM 等
- DAO治理
- 案例分享及讨论

# 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
- 合约：协议、规则

链上执行的程序，是代码和数据（状态）的集合

# 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;
    }

}
```

# 合约的组成

## 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;
    }
}
```

# 智能合约

- Remix IDE： Solidity Online IDE

https://remix.ethereum.org

**DEMO**

# 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 …

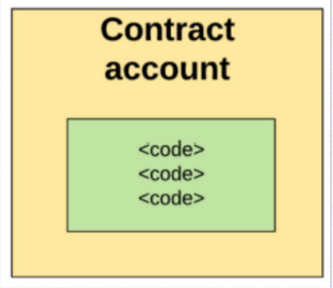
# 以太坊客户端

- 以太坊客户端：EVM 载体、网络中的节点程序
- 只要符合网络规范，（几乎）任何语言都可以实现客户端
- 常见的客户端：Geth（Go 实现）、OpenEthereum(Rust实现)
- 通过 RPC 提供服务

# 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
- 但是：
  - **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 gas 有外部账号支付。

# 钱包

- 通常，使用一个钱包来创建账号
- 钱包：
  - Metamask 外号：小狐狸（插件、App）
  - imtoken
  - TrustWallet

METAMASK

# GAS

- EVM 的计价规则，也防止图灵死机问题。
- GAS 是一个工作量单位，复杂度越大，所需 gas 越多。
- 费用 = gas 数量 * gas 单价（以太币计价gwei）

# 以太币单位

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

# 不同的网络

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

# Q & A

# 练习题

- 安装 Metamask、并创建好账号
- 执行一次转账
- 使用 Remix 创建任意一个合约
- VSCode IDE 开发环境配置

作业要求：

1. 使用自己的 github 创建一个作业代码库
2. 每一次作业使用一个文件夹(w1)
3. 提交代码、截图、交易 hash 等

# 合约开发工具

- Remix: 适用于小合约开发
- VSCode + solidity 扩展
- Truffle + Ganache
- Hardhat

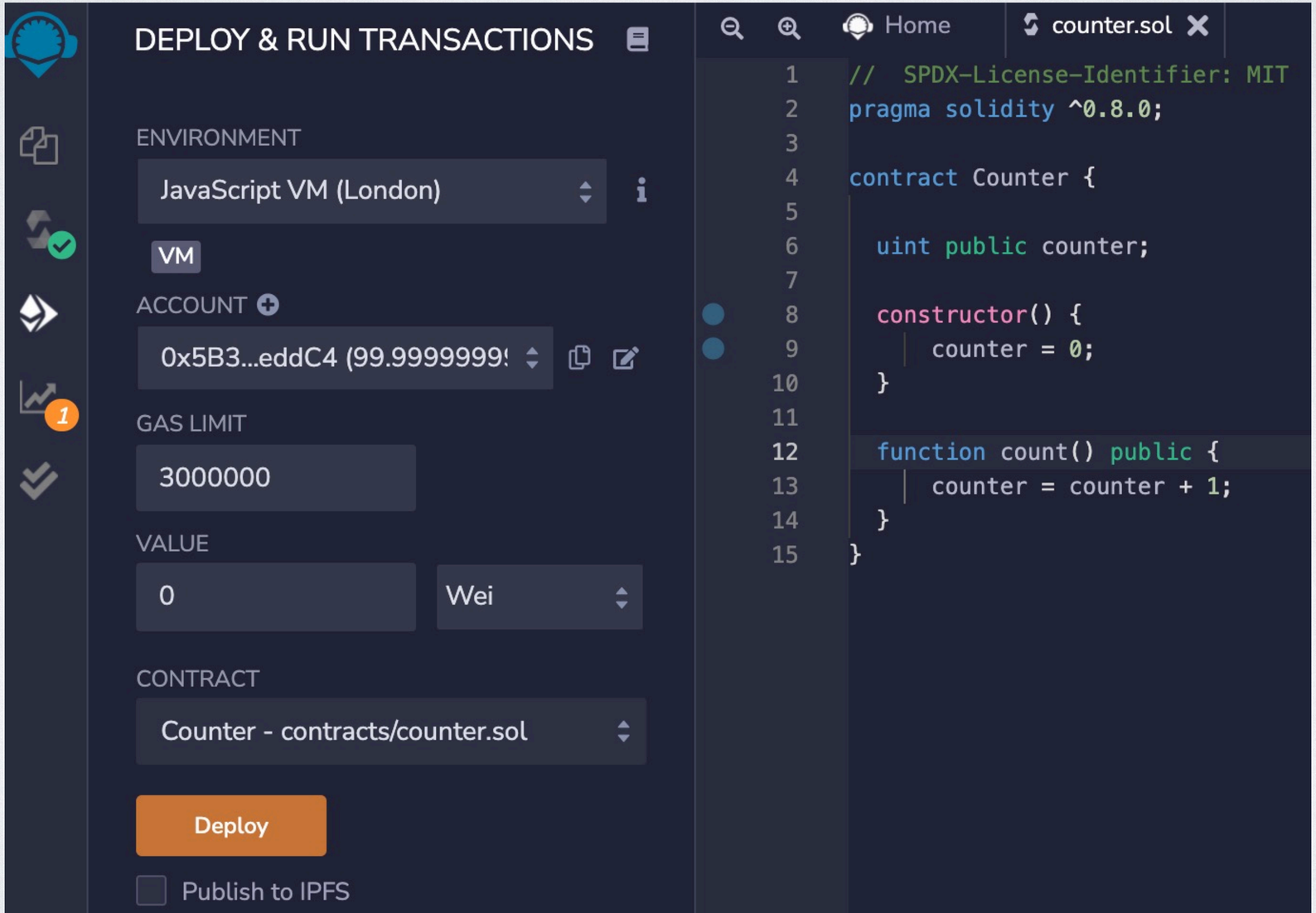
# 如何开发智能合约

分别用三个方式演示Counter 合约编译、部署、测试

- Remix
- Truffle
- Hardhat

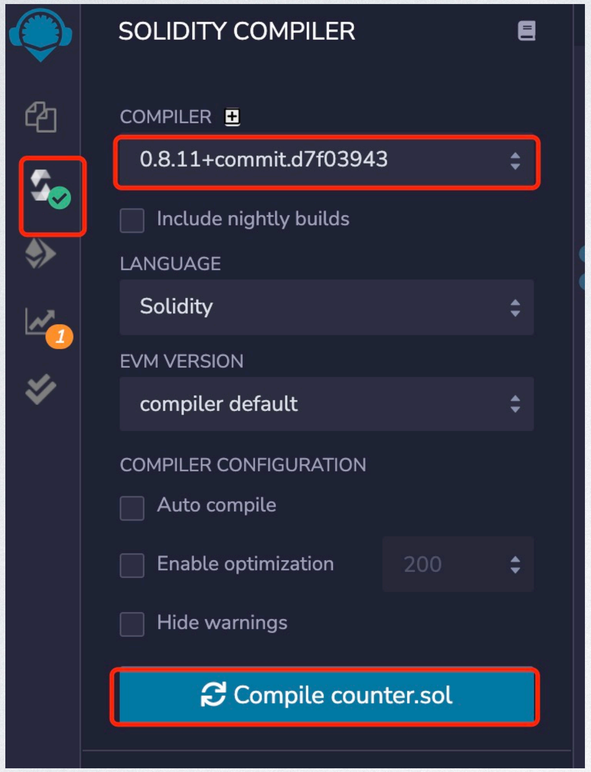
# REMIX

https://remix.ethereum.org

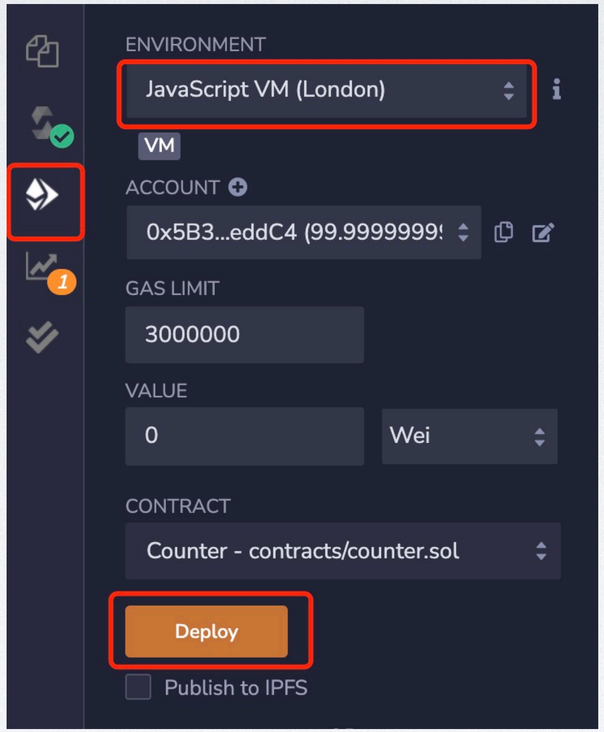


# REMIX

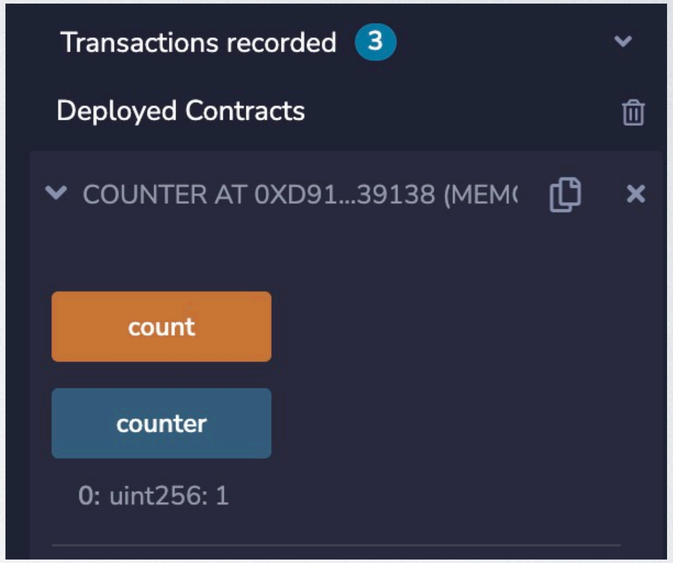
https://remix.ethereum.org




可选择编译器版本




可部署到不同的网络




执行合约函数：
橙色：触发交易
蓝色：仅读取

编译

部署

执行

# Remix

- remixd : 在remixd中访问本地文件
- 安装remixd： npm install -g @remix-project/remixd
- 共享文件： remixd -s <path-to-the-shared-folder> -u <remix-ide-instance-URL>

# Truffle

- Truffle：编译、部署、测试合约的一整套开发工具
  - 文档：https://trufflesuite.com/docs/truffle/
  - 中文文档：https://learnblockchain.cn/docs/truffle/
- Ganache：开发区块链，提供本地模拟的链上环境

# Truffle – 创建工程

- Truffle 安装： npm install -g truffle
- 创建工程： truffle init、 truffle unbox metacoin

```
➜  home > truffle init

✔ Preparing to download
✔ Downloading
✔ Cleaning up temporary files
✔ Setting up box


Unbox successful. Sweet!

Commands:

  Compile:        truffle compile
  Migrate:        truffle migrate
  Test contracts: truffle test
```

# Truffle – 创建工程

- truffle工程包含：
  - contracts：智能合约目录
  - Migrations：迁移文件、用来指示如何部署智能合约
  - test：智能合约测试用例文件夹。
  - truffle-config.js：配置文件，配置truffle连接的网络及编译选项。

# Truffle – 编译

truffle compile： 合约编译

truffle-config.js:

```
module.exports = {
    compilers: {
    solc: {
      version: "0.8.9"
    }
  }
}
```

```
➜  > truffle compile

Compiling your contracts...
===========================
> Compiling ./contracts/Counter.sol
> Artifacts written to …/build/contracts
> Compiled successfully using:
   - solc: 0.8.9+commit.e5eed63a.Emscripten.clang
```

# Truffle – 部署 – 配置网络

合约部署: 1: 配置网络，2: 编写部署脚本，3：启动网络（本地） 4. 执行部署

1. 配置网络

truffle-config.js 本地网络配置

```
module.exports = {
  networks: {
    development: {
      host: "127.0.0.1",
      port: 7545,
      gas: 5500000            //  gas limit
      gasPrice: 10000000000,  // 10 Gwei
    }
  }
};
```

# Truffle – 部署 – 配置网络

远程网络

```
ropsten: {
  provider: () => new HDWalletProvider(mnemonic, `https://ropsten.infura.io/v3/YOUR-PROJECT-ID`),
    network_id: 3,       // Ropsten's id
    gas: 5500000,        // Ropsten has a lower block limit than mainnet
    confirmations: 2,    // # of confs to wait between deployments. (default: 0)
    timeoutBlocks: 200,  // # of blocks before a deployment times out  (minimum/default: 50)
    skipDryRun: true     // Skip dry run before migrations? (default: false for public nets )
},
```

需提供助记词与 RPC URL

# Truffle – 部署脚本

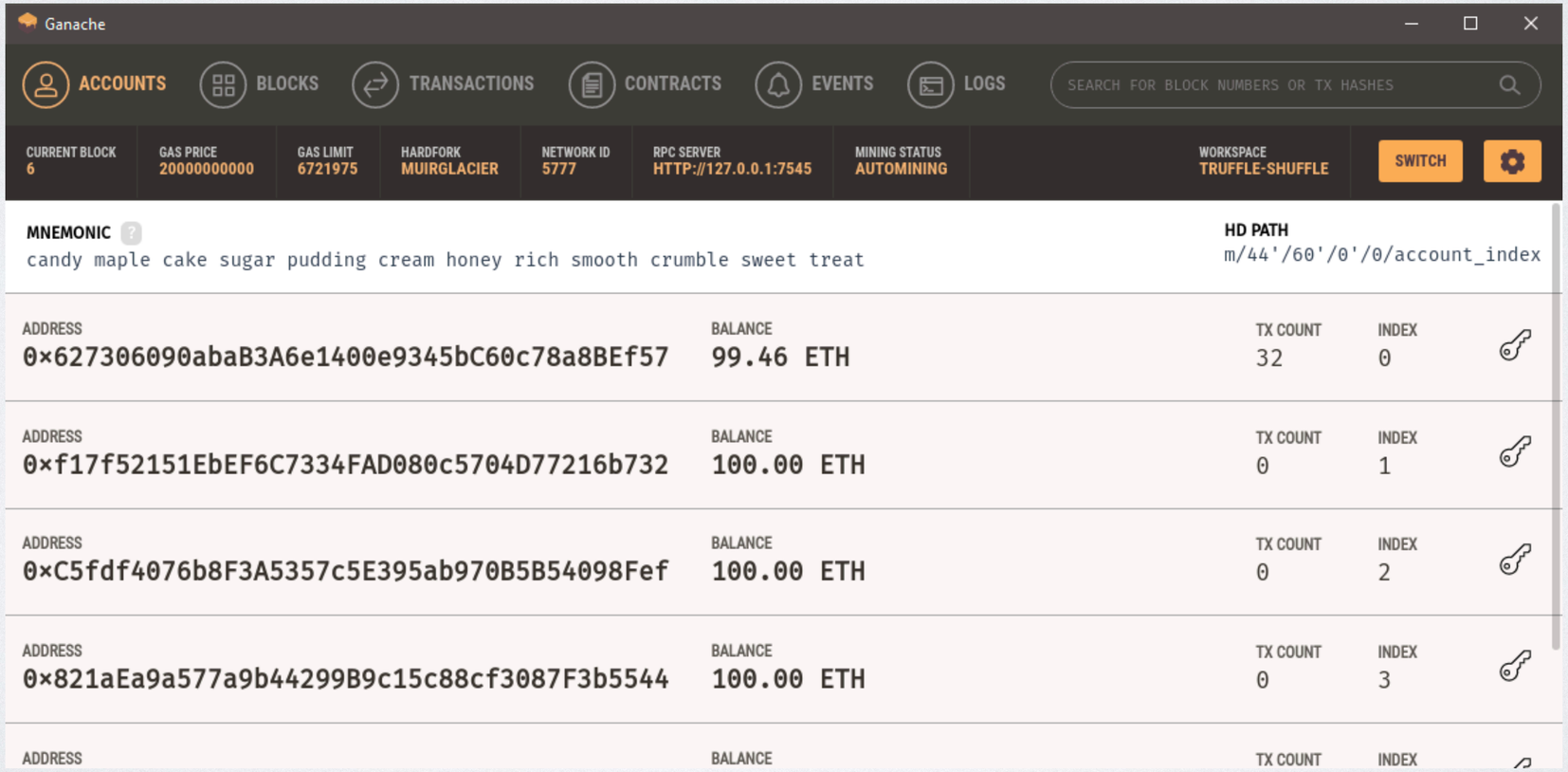
2. 编写部署脚本（migrations/1_counter.js）

Truffle 按序号（从小到大）依次执行部署脚本

```
const Counter = artifacts.require("Counter");

module.exports = function (deployer) {

  deployer.deploy(Counter);

};
```

# 启动网络 – Ganache

## 3. 启动本地网络ganache



# Truffle – 执行部署

4. 执行部署：`truffle migrate [ -f 序号 –network 网络]`

```
1_counter.js
============

   Replacing 'Counter'
   -------------------
   > transaction hash:    0x188ebce772b716ed1e51a57cb8d4086466b7f57d7febd4af9fa3262d0b6eaf4f
   > Blocks: 0            Seconds: 0
   > contract address:    0x5715221024DbB1c12eA6553f409128593F8efdD8
   > block number:        2
   > block timestamp:     1643970337
   > account:             0x0CB8c19a5f73D9Bd221cB8b7CB17Ae9d191Ae71E
   > balance:             99.99468028
   > gas used:            132993 (0x20781)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00265986 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00265986 ETH
```

# Truffle – 查看部署状态

完成部署后，在 Ganache 中，可以查看到相关的交易。

部署信息如合约地址也会写入之前编译生成的构建文件build/contracts/Counter.json中

# Truffle – 测试

支持使用 Solidity 和 JavaScript/TypeScript 编写测试，使用:

```
$ truffle test

$ truffle test ./path/to/test/file.js
```

# Truffle – 测试

JS test.js

```js
var Counter = artifacts.require("Counter");

contract("Counter", function(accounts) {
  var counterInstance;
  it("Counter", function() {  // it定义一个测试用例
    return Counter.deployed()
      .then(function(instance) {
        counterInstance = instance;
        return counterInstance.count();
      }).then(function() {
        return counterInstance.counter();
      }).then(function(count) {
        assert.equal(count, 1);  // 满足断言则测试用例通过
    });
  });
});
```

# Truffle – 执行脚本

$ truffle exec /path/to/script.js

JS test.js

```
var Counter = artifacts.require("Counter");

module.exports = async function(callback) {
  var counter = await Counter.deployed()

  await counter.count();
  let value = await counter.counter();

  console.log("current conter value:" + value);
}
```

# Q & A

# 练习题

- 使用 Truffle 部署 Counter 合约 到 test 网络（goerli） （提交交易 hash）
- 编写一个测试用例

# 补充

- truffle console 使用
  - 直接在控制台调用合约
- Truffle-min.sh
  - 压缩 artifacts 文件

# Hardhat

- Hardhat：编译、部署、测试和调试以太坊应用的开发环境，围绕 task(任务)和plugins(插件)概念设计。
- 在命令行运行Hardhat时，都是在运行任务，例如： npx hardhat compile正在运行compile任务。
- Hardhat node：开发区块链，提供本地模拟的链上环境
- 文档
  - 官方文档：https://hardhat.org/getting-started/
  - 中文文档：https://learnblockchain.cn/docs/hardhat/getting-started/

# Hardhat – 创建工程

- 安装：npm install —save hardhat
- 在目录下执行 npx hardhat 创建项目
- 查看所有任务：npx hardhat

```
➜  w2_code git:(main) ✗ npx hardhat
888     888                        888 888               888
888     888                        888 888               888
888     888                        888 888               888
8888888888  8888b.  888d888 .d88888 88888b.   8888b.  888888
888     888     "88b 888P"  d88" 888 888 "88b     "88b 888
888     888 .d888888 888    888  888 888  888 .d888888 888
888     888 888  888 888    Y88b 888 888  888 888  888 Y88b.
888     888 "Y888888 888     "Y88888 888  888 "Y888888  "Y888

👷 Welcome to Hardhat v2.8.3 👷

? What do you want to do? …
❯ Create a basic sample project
  Create an advanced sample project
  Create an advanced sample project that uses TypeScript
  Create an empty hardhat.config.js
  Quit
```

# Hardhat

## 合约编译: npx hardhat compile

hardhat.config.js:

```
module.exports = {
   Solidity: "0.8.9"
}
```

```
➜  > npx hardhat compile
Compiling 3 files with 0.8.9
Solidity compilation finished successfully
```

# Hardhat – 部署 – 配置网络

合约部署: 1: 配置网络，2: 编写部署脚本，3：启动网络（本地） 4. 执行部署

1. 配置网络

hardhat.config.js 本地网络配置

```
module.exports = {
  networks: {
    development: {
      url: “http://127.0.0.1:8545”,
      chainId: 31337
    }
  }
};
```

# Harhdat – 部署脚本

## 2. 编写部署脚本（script/xxx_deploy.js）

scripts > JS deploy.js > main

```javascript
const hre = require("hardhat");

async function main() {
  // await hre.run('compile');
  const Greeter = await hre.ethers.getContractFactory("Greeter");
  const greeter = await Greeter.deploy("Hello, Hardhat!");

  await greeter.deployed();
  console.log("Greeter deployed to:", greeter.address);
}

main()
```

# 启动 Hardhat Node 网络

3. 启动本地网络hardhat node

```
➜  w1_code git:(main) ✗ npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1: 0x70997970c51812dc3a010c7d01b50e0d17dc79c8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d
```

# Hardhat – 执行部署

4. 执行部署： npx hardhat run script/xxx_deploy.js [ –network 网络]

# Hardhat – 测试

JS test.js > ...

```
const { expect } = require("chai");
const { ethers } = require("hardhat");

describe("Greeter", function () {
  it("Should return the new greeting once it's changed", async function () {
    const Greeter = await ethers.getContractFactory("Greeter");
    const greeter = await Greeter.deploy("Hello, world!");
    await greeter.deployed();

    expect(await greeter.greet()).to.equal("Hello, world!");

    const setGreetingTx = await greeter.setGreeting("Hola, mundo!");

    // wait until the transaction is mined
    await setGreetingTx.wait();

    expect(await greeter.greet()).to.equal("Hola, mundo!");
  });
});
```

# Hardhat – 测试

```
> npx hardhat test


Greeter
Deploying a Greeter with greeting: Hello, world!
Changing greeting from 'Hello, world!' to 'Hola,
mundo!'
✓ Should return the new greeting once it's changed
(694ms)


1 passing (697ms)
```

# Hardhat – 实战

- 调试利器： console.log
- 灵活参数部署，利用 Hardhat 可以在代码中引用
  - 设置 HARDHAT_NETWORK 环境变量
  - 使用 node 部署
- 代码扁平： npx hardhat flatten xxx.sol > xxx.sol

# Hardhat – 实战

- 代码验证：
  - require("@nomiclabs/hardhat-etherscan");
  - npx hardhat verify address --network xxx
- ABI 导出：
  - require('hardhat-abi-exporter')
  - yarn run hardhat export-abi
- 自定义任务（Task）

# Q & A

# 练习题

- 使用 Hardhat 部署修改后的 Counter
- 使用 Hardhat 测试 Counter
- 写一个脚本调用 count()

# SOLIDITY语言主要特性

- Solidity 基本类型、数组、结构体、映射
- Solidity API 介绍
- 合约函数、函数修改器、函数修饰符，及各类特殊函数
- 错误处理、合约继承、接口、库及 Openzeppelin 合约库
- 理解合约事件
- 理解ABI

# 合约的组成

```
pragma solidity ^0.8.0;
contract Counter {
    uint public counter;
    constructor() {
        counter = 0;
    }
    function count() public {
        counter = counter + 1;
    }

}
```

1. 编译器版本声明

2. 定义合约

3. 状态变量

4. 合约函数

# SOLIDITY 语言

- 静态类型、编译型、高级语言
- 针对 EVM 专门设计
- 受 C++、JavaScript 等语言影响
  - 如：变量声明、for循环、重载函数的概念 来自于 C++
  - 函数关键字、导入语法来自于 JavaScript
- 文档
  - 中文：https://learnblockchain.cn/docs/solidity/
  - 英文：https://docs.soliditylang.org/

# SOLIDITY 数据类型

- Solidity 值类型
  - 布尔、**整型**、定长浮点型、定长字节数组、枚举、函数类型、**地址类型**
  - 十六进制常量、有理数和整型常量、字符串常量、地址常量
- Solidity 引用类型
  - **结构体**
  - **数组**
- 映射类型

# SOLIDITY – 整型

- 整型： `int/uint , uint8 … uint256`

  支持运算符
  - 比较运算： `<=, < , ==, !=, >=, >`
  - 位运算： `&, |, ^(异或), ~（位取反）`
  - 算术运算： `+, -, -(负), *, /, %(取余数), **（幂）`
  - 移位： `<< (左移位), >>(右移位)`

在使用整型时，要特别注意整型的大小及所能容纳的最大值和最小值， 如uint8的最大值为0xff（255），最小值是0
从 solidity 0.6.0 版本开始可以通过 Type(T).min 和 Type(T).max 获得整型的最小值与最大值。

# SOLIDITY – 整型

预测一下2个函数分别的结果是什么?
为什么?

```
pragma solidity ^0.5.0;


contract testOverflow {
    function add1() public pure returns (uint8) {
        uint8 x = 128;
        uint8 y = x * 2;
        return y;
    }


    function add2() public pure returns (uint8) {
        uint8 i = 240;
        uint8 j = 16;
        uint8 k = i + j;
    }

}
```

# SOLIDITY – 地址类型

- Solidity 使用地址类型来表示一个账号，地址类型有两种形式
  - address：一个20字节的值。
  - address payable：表示可支付地址，与address相同也是20字节，不过它有成员函数transfer和send。
- 成员函数
  - <address>.balance(uint256)： 返回地址的余额
  - <address payable>.transfer(uint256 amount)： 向地址发送以太币，失败时抛出异常 （gas：2300）
  - <address payable>.send(uint256 amount) returns (bool): 向地址发送以太币，失败时返回false

# SOLIDITY – 地址类型

```
pragma solidity ^0.6.0;

contract testAddr {

  function testTrasfer(address payable x) public {
     address myAddress = address(this);

     if (myAddress.balance >= 10) {
         x.transfer(10);
     }
  }
}
```

合约转换为地址类型

给一个合约地址转账，即上面代码 x 是合约地址时，合约的receive函数或fallback函数会随着transfer调用一起执行

# SOLIDITY – 地址类型

- 3 个底层成员函数：call, delegatecall, staticcall
  - <address>.call(bytes memory) returns (bool, bytes memory)

通常用于合约交互，直接控制编码的方式调用合约函数。

例如调用合约的register(string) 方法：

```
bytes memory payload = abi.encodeWithSignature("register(string)", "MyName");

(bool success, bytes memory returnData) = address(nameReg).call(payload);

require(success);
```

# SOLIDITY – 地址高级用法

- `call()`

  切换上下文，附加gas {gas: } ， 附加value {value: }

  addr.call{value: 1 ether}("") 功能等价 transfer(1 ether)， 但是没有 gas 限制

  失败不是发生异常，一定要检查返回值

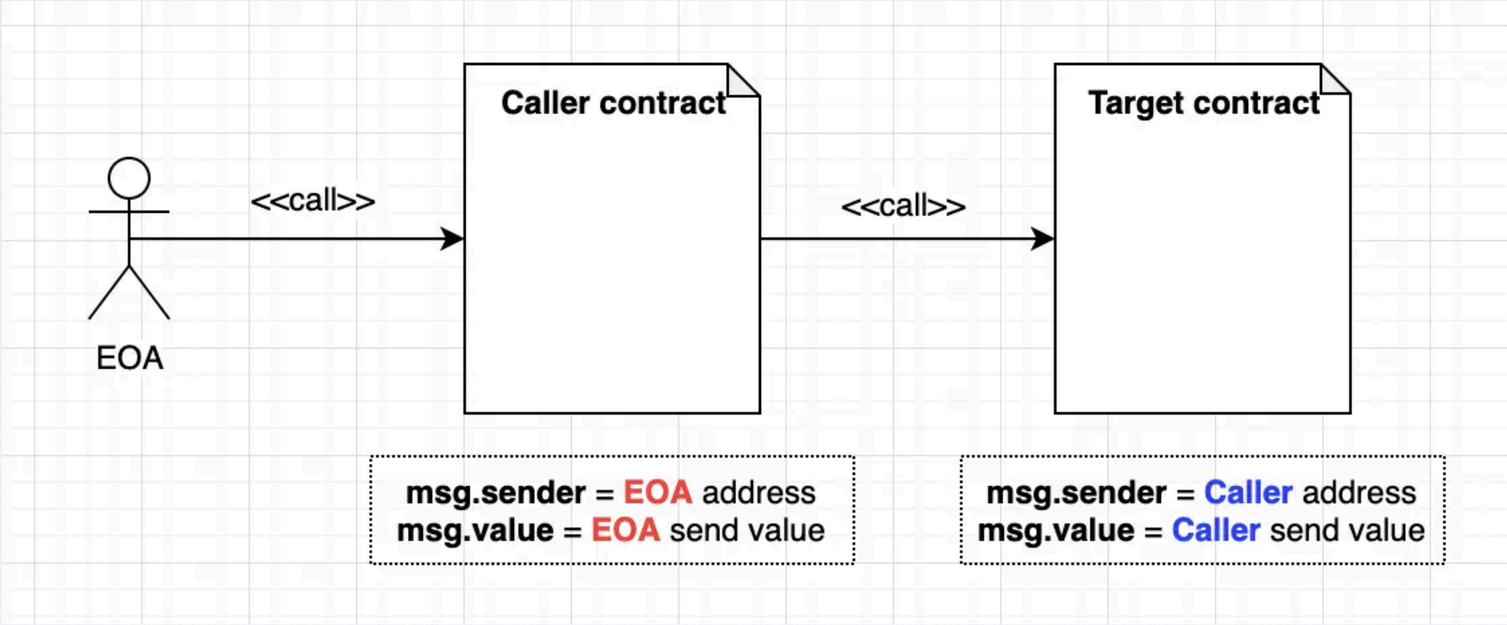
- `delegatecall()`

  保持上下文

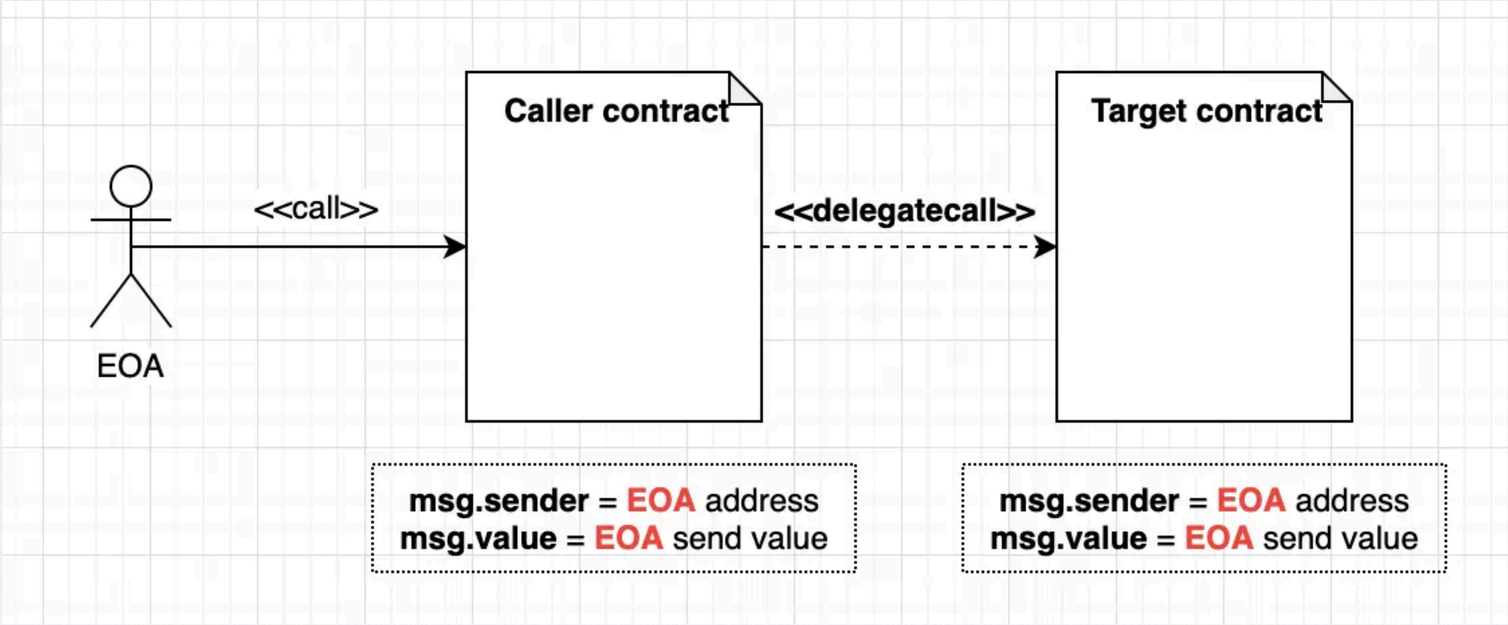
  不支持附加value {value: }

https://github.com/Uniswap/solidity-lib/blob/master/contracts/libraries/TransferHelper.sol

# SOLIDITY – 地址 – CALL



# SOLIDITY – 地址– DELEGATECALL



# SOLIDITY – 合约类型

```
pragma solidity ^0.6.0;

contract Hello {
  function sayHi() public {
  }
}
```

- 每个合约都是一个类型，可声明一个合约类型。
  如：`Hello c`; 则可以使用`c.sayHi()` 调用函数

- 合约可以显式转换为address类型，从而可以使用地址类型的成员函数。

# SOLIDITY – 引用类型

- 值类型赋值时总是完整拷贝。而复杂类型占用的空间较大（>32个字节），拷贝开销很大，这时就可以使用引用的方式，即通过多个不同名称的变量指向一个值。
- 引用类型都有一个额外属性来标识数据的存储位置:
  - memory（内存）: 生命周期只存在于函数调用期间
  - storage（存储）: 状态变量保存的位置，gas开销最大
  - calldata（调用数据）: 用于函数参数不可变存储区域

# SOLIDITY – 数组

- T[k] : 元素类型为T，固定长度为k的数组
- T[] : 元素类型为T，长度动态调整
- 数组通过下标进行访问，序号是从0开始
- bytes、string 是一种特殊的数组
  - bytes是动态分配大小字节的数组，类似于byte[]，但是bytes的gas费用更低，bytes和string都可以用来表达字符串，对任意长度的原始字节数据使用bytes，对任意长度字符串（UTF-8）数据使用string。

# SOLIDITY – 数组

- 成员
  - Lengh属性：表示当前数组的长度
  - push()：添加新的零初始化元素到数组末尾，返回引用
  - push(x)： 数组末尾添加一个给定的元素
  - pop(): 从数组末尾删除元素

# SOLIDITY – 数组

```
pragma solidity ^0.8.0;

contract testArray {
    uint[10] tens;
    uint[] public numbers;

    function test(uint len) public pure {
        string[4] memory adaArr = ["This", "is", "an", "array"];
        uint[] memory c = new uint[](len);
    }

    function add(uint x) public {
        numbers.push(x);
    }
}
```

# SOLIDITY – 数组

- gas 问题
  - 移除元素
  - 遍历操作

```
function dosome() public {
    uint len = numbers.length;
    for (uint i = 0; i < len; i++) {
        // do...
    }
}

function remove(uint index) public {
    uint len = numbers.length;
    if (i == len - 1) {
        numbers.pop();
        break;
    } else {
        numbers[index] = numbers[len - 1];
        epoches.pop();
        break;
    }
}
```

# SOLIDITY – 结构体

使用Struct 声明一个结构体，定义一个新类型。

```
contract testStruct {
  struct Funder {
      address addr;
      uint amount;
  }

  mapping (uint => Funder) funders;
   function contribute(uint id) public payable {
        funders[id] = Funder({addr: msg.sender, amount: msg.value});
        funders[id] = Funder(msg.sender, msg.value);
    }
    function getFund(uint id) public view returns  (address, uint ) {
        return (funders[id].addr, funders[id].amount);
    }
}
```

# SOLIDITY – 映射

- 声明形式： mapping(KeyType => ValueType) ， 例如： mapping(address => uint) public balances;
- 使用方式类似数组，通过 key 访问，例如： balances[userAddr];
- 映射没有长度、没有 key 的集合或 value 的集合的概念
- 只能作为状态变量
- 如果访问一个不存在的键，返回的是默认值。

# SOLIDITY – 映射

```
pragma solidity >=0.8.0;

contract MappingExample {
    mapping(address => uint) public balances;

    function update(uint newBalance) public {
        balances[msg.sender] = newBalance;
    }
}
```

# SOLIDITY – 全局变量及函数

- 区块和交易
- ABI 编码
- 错误处理
- 数学及加密
- 地址及合约

https://learnblockchain.cn/docs/solidity/units-and-global-variables.html#special-variables-and-functions

# SOLIDITY – 全局变量及函数

- `block.number ( uint ):` 当前区块号
- `block.timestamp ( uint):` 自 unix epoch 起始当前区块以秒计的时间戳
- `msg.sender ( address ):` 消息发送者（当前调用）
- `msg.value ( uint ):` 随消息发送的 wei 的数量
- `tx.origin (address payable):` 交易发起者（完全的调用链）
- …

https://learnblockchain.cn/docs/solidity/units-and-global-variables.html#special-variables-and-functions

# Q & A

# 练习题

- 编写一个Bank合约：
- 通过 Metamask 向Bank合约转账ETH
- 在Bank合约记录每个地址转账金额
- 编写 Bank合约withdraw(), 实现提取出所有的 ETH

# SOLIDITY – 合约

- 使用contract关键字来声明一个合约，一个合约通常由状态变量、函数、函数修改器以及事件组成。

```
pragma solidity >=0.8.0;


contract C {
    function f(uint a) private pure returns (uint b) { return a + 1; }
    function setData(uint a) internal { data = a; }
    uint public data;
}
```

# SOLIDITY – 合约

- 创建合约的几个方法：
  - 外部部署（Remix/Hardhat/Truffle）Web.js
  - 合约使用New
  - 最小代理合约（克隆）：
    - https://eips.ethereum.org/EIPS/eip-1167
    - https://github.com/optionality/clone-factory
  - Create2
    - C c = new C{salt: _salt}();

# SOLIDITY – 合约地址

- 合约地址的确认
  - 根据创建者（sender）的地址以及创建者发送过的交易数量（nonce）来计算确定
    - keccak256(rlp.encode([normalize_address(sender), nonce]))[12:]
  - Create2
    - keccak256(0xff ++ senderAddress ++ salt ++ keccak256(init_code))[12:]

# SOLIDITY – 合约

内/外

外部访问

- 使用public 、private、external、internal 可见性关键字来控制变量和函数是否可以被外部使用

内部

内部及继承

```
pragma solidity >=0.8.0;


contract C {
    function f(uint a) public pure returns (uint b) { return a + 1; }
    function setData(uint a) internal { data = a; }
    uint public data;
}
```

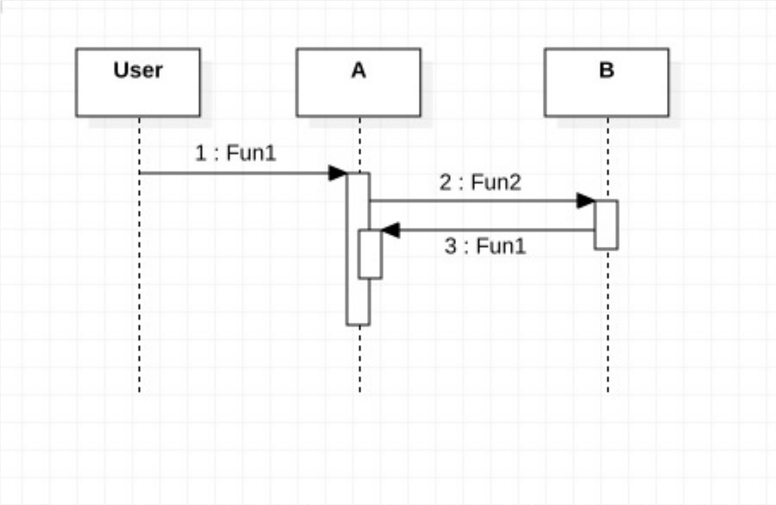
# SOLIDITY – 合约函数

- 构造函数(constructor): 初始化逻辑
- 视图函数(view)、纯函数:(pure) 不修改状态，不支付手续费
- getter 函数: 所有 public 状态变量创建 getter 函数
- Payable 修饰符： 表示一个函数可以接收以太币
- receive 函数: 接收以太币时回调。
- fallback 函数: 没有匹配函数标识符时, fallback 会被调用，如果是转账时，没有receive也有调用fallback
- 函数修改器（`modifier`）：可用来改变一个函数的行为，如检查输入条件、控制访问、重入控制

# SOLIDITY – 重入问题

- 调用外部函数时，要时刻注意重入问题：

- 重入



```
pragma solidity ^0.8.0;
contract testFunc {
    address public owner;
    uint private deposited;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }
    fallback() external payable {
        deposited += msg.value;
    }
    function getDeposited() public view returns(uint) {
        return deposited;
    }
   function  withdraw() public onlyOwner {
        payable(owner).transfer(deposited);
    }
}
```

函数修改器修饰函数时，函数体被插入到“_;”

# SOLIDITY – 错误处理

- 在程序发生错误时的处理方式：EVM通过回退状态来处理错误的，以便保证状态修改的事务性
- assert()和require()用来进行条件检查，并在条件不满足时抛出异常
- revert()：终止运行并撤销状态更改
- try/catch: 捕获合约中外部调用的异常，
  - 注意：out of gas 错误不是程序异常，错误不能捕获。

```
pragma solidity ^0.8.0;

contract Foo {
    function myFunc(uint x) public pure returns (uint ) {
        require(x != 0, "require failed");
        return x + 1;
    }
}

contract trycatch {

    Foo public foo;
    uint public y;
    constructor() {
        foo = new Foo();
    }

    function tryCatchExternalCall(uint _i) public {
        try foo.myFunc(_i) returns (uint result) {
            y = result;
        } catch {
        }

    }
}
```

# SOLIDITY – 继承

- 和大多数高级语言一样，Solidity 也支持继承
- 使用关键字 is
- 继承时，链上实际只有一个合约被创建，基类合约的代码会被编译进派生合约。
- 派生合约可以访问基类合约内的所有非私有（private）成员，因此内部（internal）函数和状态变量在派生合约里是可以直接使用的

```
pragma solidity ^0.8.0;

contract A {
    uint public a;
    constructor() {
        a = 1;
    }
}

contract B is A {
    uint public b ;
    constructor() {
        b = 2;
    }
}
```

在部署B时候，可以查看到a为1，b为2。

# SOLIDITY – 继承、抽象合约

- abstract抽象合约
  - 不能被部署， 可包含没有实现的纯虚函数
- super：调用父合约函数
- virtual： 表示函数可以被重写
- overide： 表示重写了父合约函数

# SOLIDITY – 接口

- 函数的抽象，作为一个类型声明，广泛用于合约之间的调用
- 不可以有实现的函数
- 不能继承自其他接口
- 没有构造方法
- 没有状态变量

```
pragma solidity ^0.8.0;

contract Counter {
    uint public count;

    function increment() external {
        count += 1;
    }
}

interface ICounter {
   function count() external view returns (uint);
   function increment() external;
}

contract MyContract {
    function incrementCounter(address _counter) external {
        ICounter(_counter).increment();
    }

    function getCount(address _counter) external view returns (uint) {
        return ICounter(_counter).count();
    }
}
```

# SOLIDITY – 库

- 与合约类似（一个特殊合约），是函数的封装，用于代码复用。
- 如果库函数都是 internal 的，库代码会嵌入到合约。
- 如果库函数有external或 public ，库需要单独部署，并在部署合约时进行链接，使用委托调用
- 没有状态变量
- 不能给库发送 Ether
- 给类型扩展功能：Using lib for type; 如： using SafeMath for uint;

```
pragma solidity ^0.8.0;

library SafeMath {
    function add(uint x, uint y) internal pure returns (uint) {
        uint z = x + y;
        require(z >= x, "uint overflow");

        return z;
    }
}

contract TestLib {
    using SafeMath for uint;

    function testAdd(uint x, uint y) public pure returns (uint) {
        return x.add(y);
    }

}
```

# SOLIDITY – 链接外部库

```
  const ExLib = await hre.ethers.getContractFactory("Library");
  const lib = await ExLib.deploy();
  await lib.deployed();


await hre.ethers.getContractFactory("TestExLib", {
    libraries: {
      Library: lib.address,
    },
  });
```

# SOLIDITY – OpenZeppelin

- 善于复用库，不仅提高效率，还可以提高安全性
- OpenZeppelin 功能丰富、经过反复验证的库函数集合。

下一周进一步介绍OpenZeppelin

# SOLIDITY – 事件

- 合约与外部世界的重要接口，通知外部世界链上状态的变化
- 事件有时也作为便宜的存储
- 使用关键字 event 定义事件，事件不需要实现
- 使用关键字 emit 触发事件
- 事件中使用indexed修饰，表示对这个字段建立索引，方便外部对该字段过滤查找

```
pragma solidity ^0.8.0;

contract testEvent {
    constructor() {
    }

    event Deposit(address indexed _from, uint _value);

    function deposit(uint value) public {
        emit Deposit(msg.sender, value);
    }
}
```

事件将记录在日志之中，下周将介绍如何从交易收据中解析日志

# ABI

- ABI：Application Binary Interface 应用程序二进制接口
- ABI 接口描述： 定义如何与合约交互
- ABI 编码
  - 函数选择器：对函数签名计算Keccak-256哈希，取前 4 个字节
  - 参数编码

# ABI

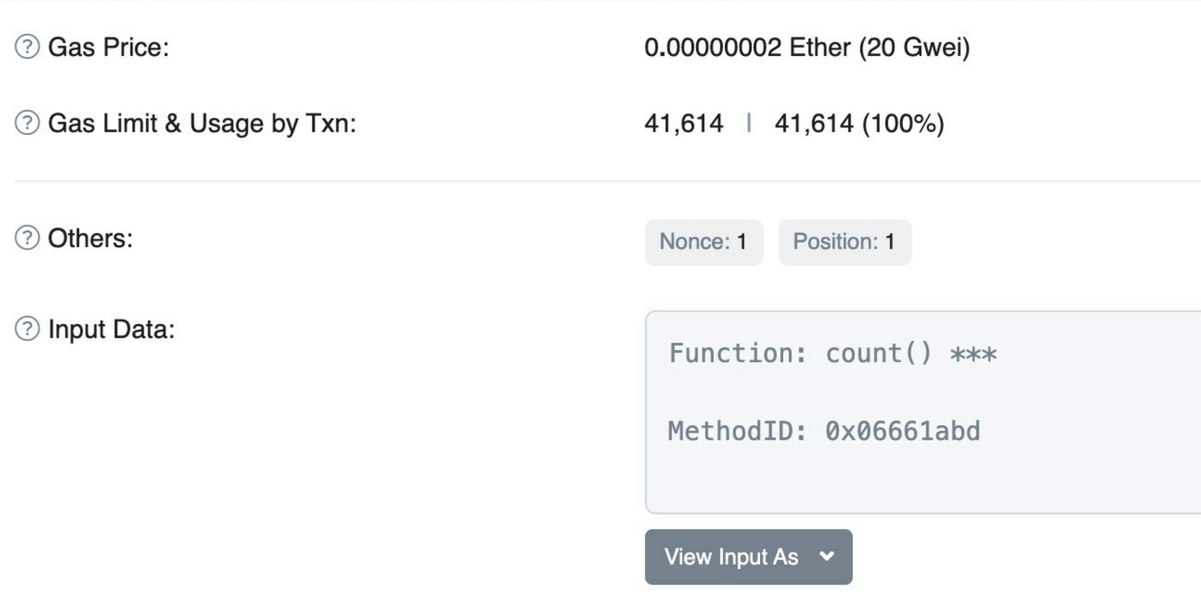
调用一个合约函数 = 向合约地址发送一个交易

交易的内容就是 ABI 编码数据

# ABI

bytes4(keccak256("count()")) = 0x06661abd

Tx Hash：https://ropsten.etherscan.io/tx/0xafcf79373cb38081743fe5f0ba745c6846c6b08f375fda028556b4e52330088b




https://www.4byte.directory/

```
(bool success,) = address(x).call("0x06661abd");
require(success);
```

# 推荐SOLIDITY学习资料

- https://ethernaut.openzeppelin.com/
- https://cryptozombies.io/en/course/
- https://solidity-by-example.org/
- https://learnblockchain.cn/column/1

# Q & A

# 练习题

- 编写合约Score，用于记录学生（地址）分数：
  - 仅有老师（用`modifier`权限控制）可以添加和修改学生分数
  - 分数不可以大于 100；
- 编写合约 Teacher 作为老师，通过 IScore 接口调用修改学生分数。

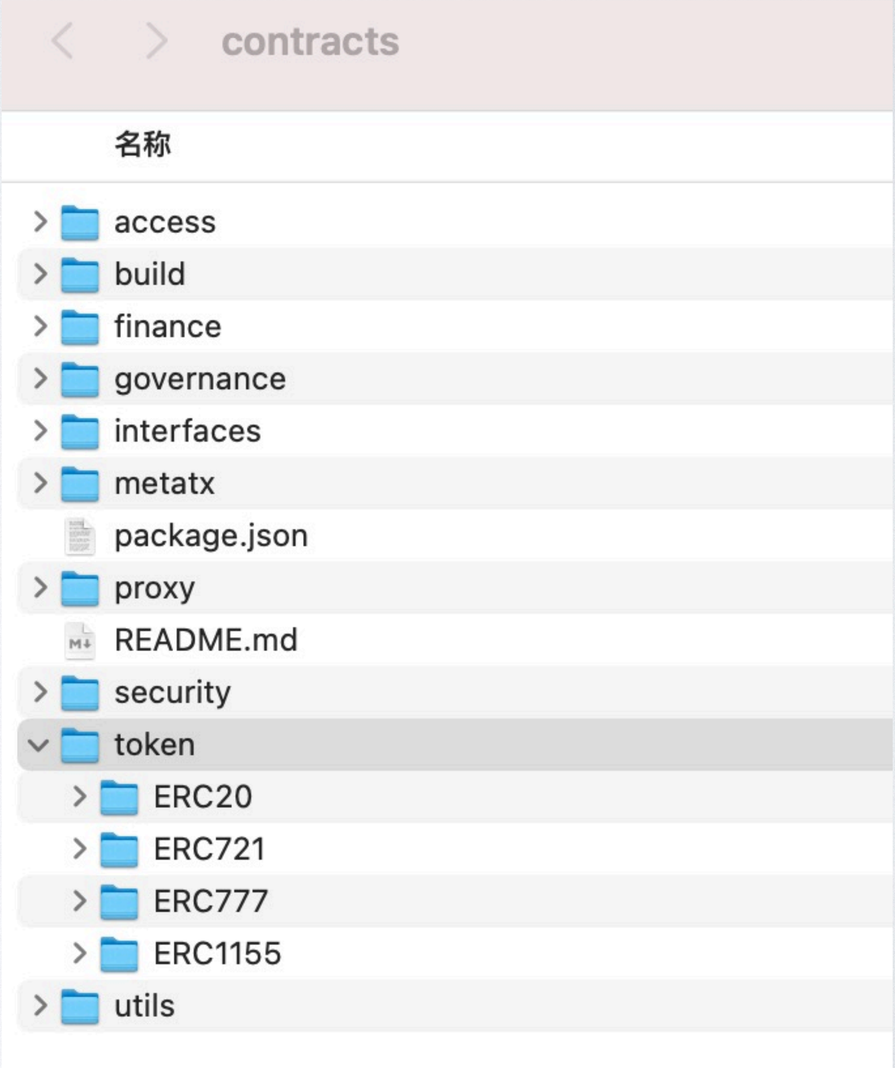
# 智能合约及DAPP开发实战进阶

- 最常用 ERC 标准介绍及实战：ERC20、ERC777、EIP2612、ERC721、ERC1155
- 优化 GAS 的众多技巧
- DApp 开发：前端与合约交互（ethers.js）
- 解析合约事件与TheGraph 使用

# OpenZeppelin

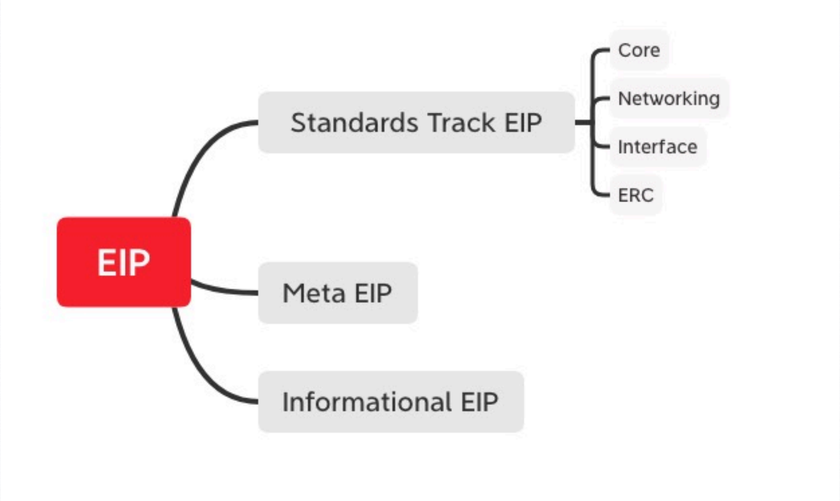
- 代码库： https://github.com/OpenZeppelin/openzeppelin-contracts
- 安装： npm install @openzeppelin/contracts —save-dev

  - token： 包含 ERC20、ERC721、ERC777、ERC1155 代币
  - access: 合约函数访问控制功能
  - utils： 实现一些工具库，如判断是否为合约地址、数学函数等。
  - proxy： 升级代理



# ERC20

- 什么是 ERC20 Token
  - https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
  - EIP： Ethereum Improvement Proposals（以太坊改进提案）



# ERC20

- ERC20标准包含哪些内容
  - 定义统一的函数名：名称、发行量、转账函数、转账事件等
  - 以便交易所、钱包进行集成
- 所有实现了这些函数的合约都是 ERC20 Token
- ERC20 可以表示任何同质的可以交易的内容：
  - 货币、股票、积分、债券、利息…
  - 可以用数量来表示的内容 基本上可以ERC20 表示

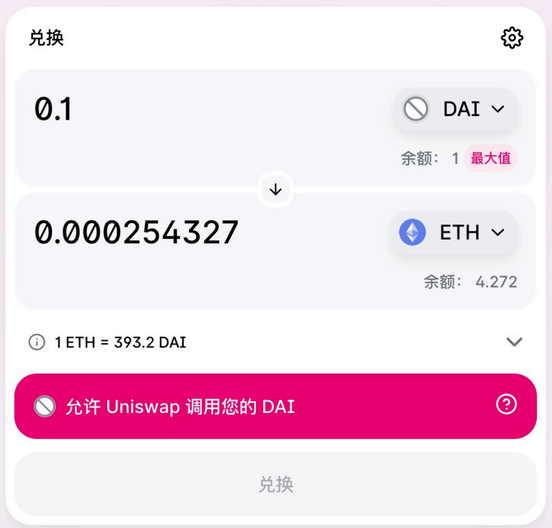
# ERC20

```
interface IERC20 {
      function totalSupply() external view returns (uint256);
      function balanceOf(address account) external view returns (uint256);
      function transfer(address to, uint256 amount) external returns (bool);
      function allowance(address owner, address spender) external view returns (uint256);
      function approve(address spender, uint256 amount) external returns (bool);
      function transferFrom(address from, address to, uint256 amount) external returns (bool);

      event Transfer(address indexed from, address indexed to, uint256 value);
      event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

# ERC20

```
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract LBCToken is ERC20 {
    constructor() ERC20("LBC Token", "LBC") {
        _mint(msg.sender, 1000*10**18);
    }
}
```

# ERC20 – 授权




- 如何让合约使用 ERC20？
  - 例如，把 ERC20 存入合约？在合约里如何记录用户的存款量？
  - 用户直接向合约转账可以吗？

# ERC20 – 授权

- 分两步：
  - 用户 A 调用 ERC20 的 Approve(合约 B,  数量);
  - 用户 A 调用 合约 B 的 deposite(); 完成存款

```
pragma solidity ^0.8.0;

contract B {
   mapping(address => uint) deposited;
   Function deposite(uint amount) {
     IERC20.transferFrom(msg.sender, address(this), amount);
     deposited[msg.sender] += amount;
   }
}
```

# ERC777

- ERC20 问题
  - 转账无法携带额外的信息。
  - 没有转账回调:
    - 依靠：授权、授权、授权
    - 误入合约被锁死。
- ERC777:
  - ERC777： send(dest, value, data)
  - ERC777 通过全局注册表（ERC1820）的注册监听回调
    - 即便是普通地址也可以实现回调
    - 误入合约被锁死。

# ERC777

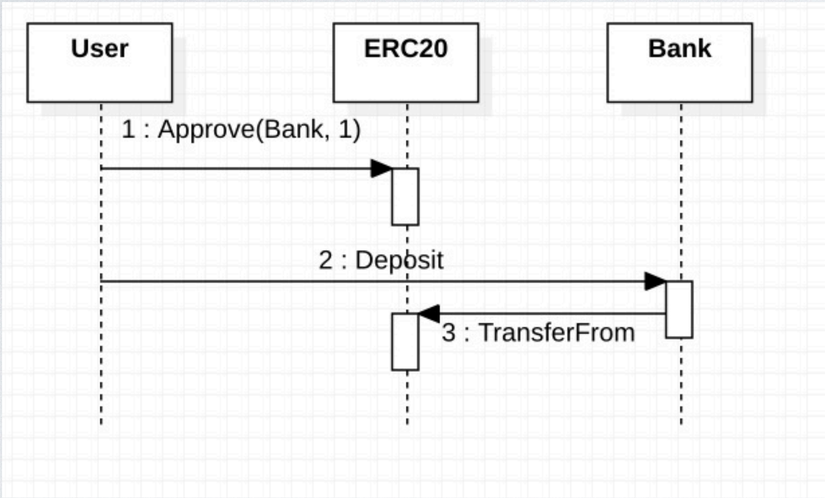
```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC777/ERC777.sol";

contract GLDToken is ERC777 {
    constructor(uint256 initialSupply, address[] memory defaultOperators)
        ERC777("Gold", "GLD", defaultOperators)
    {
        _mint(msg.sender, initialSupply, "", "");
    }
}
```

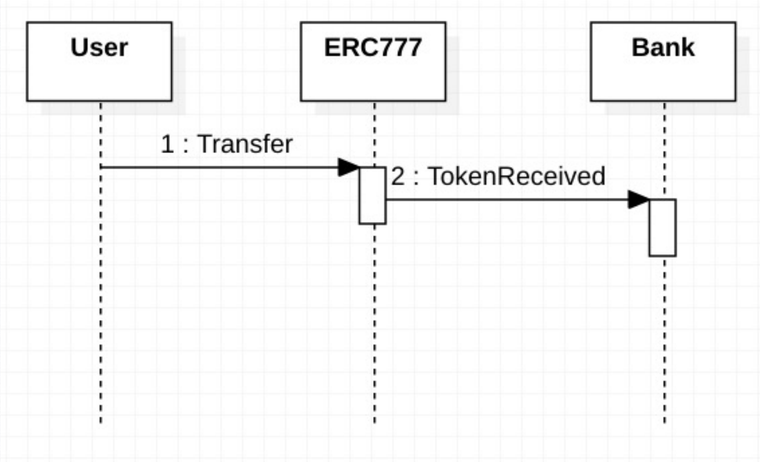
# ERC20-Permit（EIP2612）

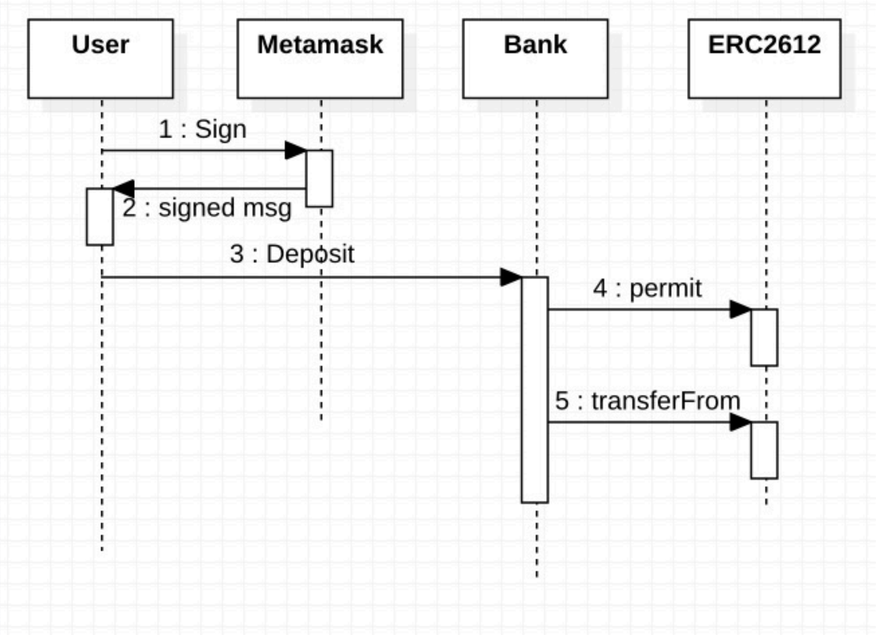
- 线下签名授权
  - （授权）可以在线下签名进行，签名信息可以在执行接收转账交易时提交到链上，让授权和转账在一笔交易里完成。
  - 同时转账交易也可以由接收方（或其他第三方）来提交，也避免了用户（ERC20的拥有者）需要有 ETH的依赖。

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ERC2612 is ERC20, ERC20Permit {

    constructor() ERC20("ERC2612", "ERC2612") ERC20Permit("ERC2612") {
        _mint(msg.sender, 1000 * 10 ** 18);
    }
}
```

# Token 总结




- ERC20 转账应该总是使用 safeTransfer：

https://github.com/Uniswap/solidity-lib/blob/master/contracts/libraries/TransferHelper.sol

# ERC721

- ERC20 合约中每个 token 与其他的 token 一样，称为同质化 Token（可置换的）
- ERC721 合约中每个 token 都是独一无二的，如：
  - 艺术品画作、收藏品
  - 创作品：声音、影片、文章、一份档案
  - 游戏中的（限量）道具
  - 任何有特性的内容：一个交易记录

# ERC721

```
pragma solidity ^0.8.0;

interface IERC721 /* is ERC165 */ {

    event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);

    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data)
external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;

    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns
(bool);
}
```

# ERC721

```
pragma solidity ^0.8.0;

interface ERC721Metadata {
    function name() external pure returns (string _name);
    function symbol() external pure returns (string _symbol);
    function tokenURI(uint256 _tokenId) external view returns (string);
}
```

- ERC721 合约中每个 token 有一个 id
- 每个 Token 有一个对应 URI 来描述属性（JSON）

# ERC721

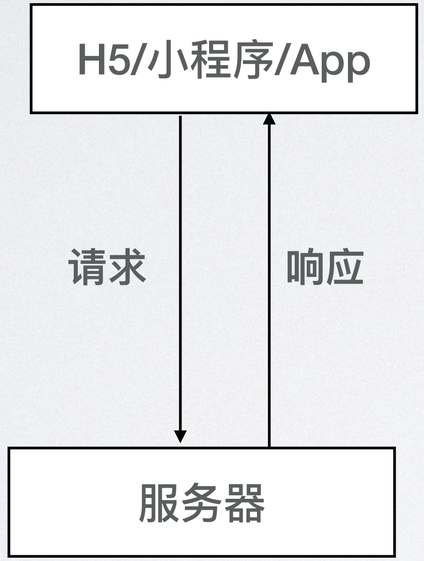
```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract GameItem is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("GameItem", "ITM") {}

    function awardItem(address player, string memory tokenURI) public returns (uint256) {
        _tokenIds.increment();

        uint256 newItemId = _tokenIds.current();
        _mint(player, newItemId);
        _setTokenURI(newItemId, tokenURI);

        return newItemId;
    }
}
```

# ERC1155

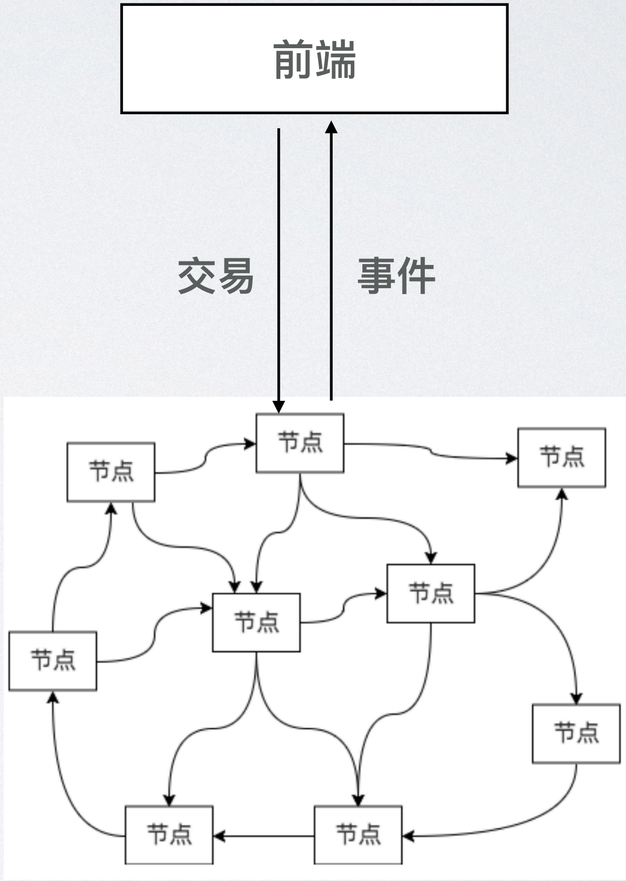
```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";

contract GameItems is ERC1155 {
    uint256 public constant GOLD = 0;
    uint256 public constant SILVER = 1;
    uint256 public constant THORS_HAMMER = 2;
    uint256 public constant SWORD = 3;
    uint256 public constant SHIELD = 4;

    constructor() ERC1155("https://game.example/api/item/{id}.json") {
        _mint(msg.sender, GOLD, 10**18, "");
        _mint(msg.sender, SILVER, 10**27, "");
        _mint(msg.sender, THORS_HAMMER, 1, "");
        _mint(msg.sender, SWORD, 10**9, "");
        _mint(msg.sender, SHIELD, 10**9, "");
    }
}
```

# DApp 去中心化应用

**中心化应用**
**web2.0**




**去中心化应用**
**web3.0**



# App vs DApp开发

| 中心化应用 web2.0 | 去中心化应用 web3.0 | |
|---|---|---|
| UI呈现 | UI呈现 | 钱包/DApp 浏览器 |
| http请求 | rpc请求 | ethers.js |
| 后端服务程序 | 智能合约 | |
| 服务器Nginx/Apache | EVM节点 | |

两个主要特点：

1. 交易离不开钱包
2. 交易是异步的

# DApp 开发：前端与合约交互

- Ethers.js/Web3.js：一套和以太坊区块链进行交互的库，RPC 接口封装
- 安装：npm install --save ethers

# DApp 开发：前端与合约交互

- 使用 ethers.js 调用 ERC20 合约示例：

```
var abi = [...];
var addr = "0x...";
var contract = new ethers.Contract(address, abi, provider);

contract.transfer(targetAddress, amount)
      .then(function(tx) {
         console.log(tx);
     });
```

# DApp 开发：前端与合约交互

**DEMO**

- 链接钱包（Provider）
- 初始化合约
- 从合约获取数据、发起交易

# Q & A

# 练习题

- 发行一个 ERC20 Token：
  - 可动态增发（起始发行量是 0）
  - 通过 ethers.js. 调用合约进行转账

- 编写一个Vault 合约：
  - 编写deposite 方法，实现 ERC20 存入 Vault，并记录每个用户存款金额， 用从前端调用（Approve，transferFrom）
  - 编写 withdraw 方法，提取用户自己的存款 （前端调用）
  - 前端显示用户存款金额

# SOLIDITY – 事件

- 合约与外部世界的重要接口，通知外部世界链上状态的变化
- 事件有时也作为便宜的存储
- 使用关键字 event 定义事件，事件不需要实现
- 使用关键字 emit 触发事件
- 事件中使用indexed修饰，表示对这个字段建立索引，方便外部对该字段过滤查找

# 事件

- 事件
  - 索引的事件 生成 对应的 Topic
  - 每个事件有个事件签名作为Topic
  - 非索引事件作为 data 保存

```
let transferTopic = ethers.utils.keccak256(
      ethers.utils.toUtf8Bytes("Transfer(address,address,uint256)"));

// 等价
ethers.utils.id("Transfer(address,address,uint256)")
```

https://docs.ethers.io/v5/api/utils/hashing/

# 监听事件

- 监听（实时）事件（使用 webSocket 链接）

```
let erc20 = new ethers.Contract(erc20地址,  ABI,  provider);

// indexed 的字段可以作为参数加入到过滤器中
let filter = erc20.filters.Transfer()

wssprovider.on(filter, (log) => {
   console.log(log);
})
```

区块链社区


```
{
  blockNumber: 9,
  blockHash: '0x5eaa8438d80d30e521b6c48db8c22d5b40255cd472d71cfd387c314486a7e679',
  transactionIndex: 0,
  removed: false,
  address: '0x5FbDB2315678afecb367f032d93F642f64180aa3',
  data: '0x00000000000000000000000000000000000000000000000000000000000003e8',
  topics: [
    '0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef',
    '0x000000000000000000000000f39fd6e51aad88f6f4ce6ab8827279cfffb92266',
    '0x000000000000000000000000070997970c51812dc3a010c7d01b50e0d17dc79c8'
  ],
  transactionHash: '0x1eda2652cde3d9b3d8dbb1e0441a2139ebd25ba25a97fab3158406a695e33ba9',
  logIndex: 0
}
```

# 获取事件

- 获取（历史）事件

```
let erc20 = new ethers.Contract(erc20地址,  ABI,  provider);
let filter = erc20.filters.Transfer()

// 选择区块区间
filter.fromBlock = 1000;
filter.toBlock = 2000; // "latest";

// 获取日志
let logs = await provider.getLogs(filter);
let logs = await erc20.queryFilter(filterTo, -10, "latest");
```

https://docs.ethers.io/v5/api/providers/provider/#Provider-getLogs

https://docs.ethers.io/v5/concepts/events/#events--filters

# 处理事件

- 解析事件
  - 使用 ethers.utils.Interface 进行编解码

```
const TransferEvent = new ethers.utils.Interface(["event Transfer(address indexed from,address indexed to,uint256 value)"]);

const data = TransferEvent.parseLog(logs[i])
console.log("from:" + data.args.from)
console.log("from:" + data.args.to)
```

https://docs.ethers.io/v5/api/utils/abi/interface/#Interface

# TheGraph

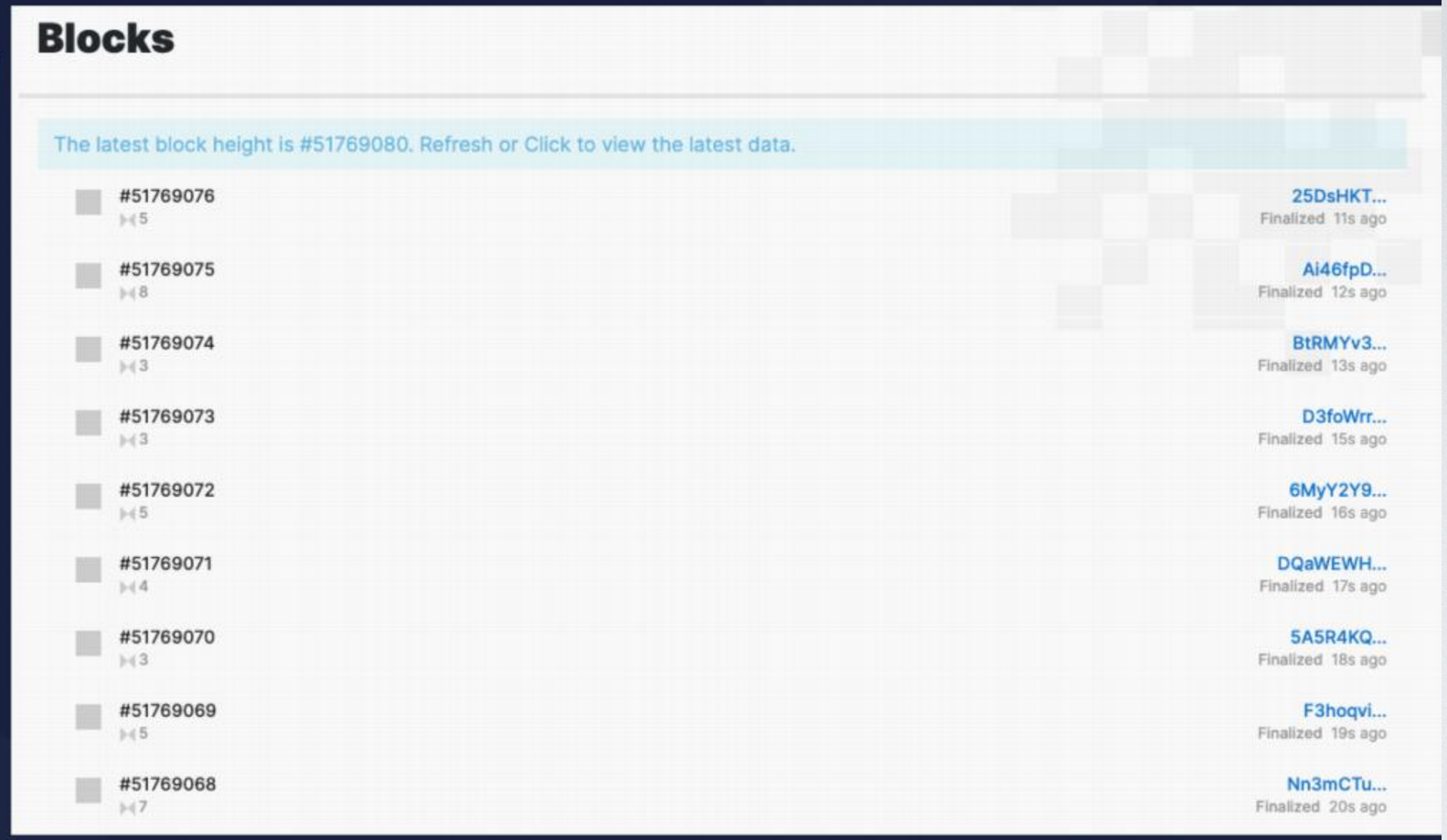
TheGraph 中文区技术工程师 智钢为大家分享

# The Graph
# 如何提高 Dapp 开发效率

Web3

# 搜索与查询

没有The Graph



Web3

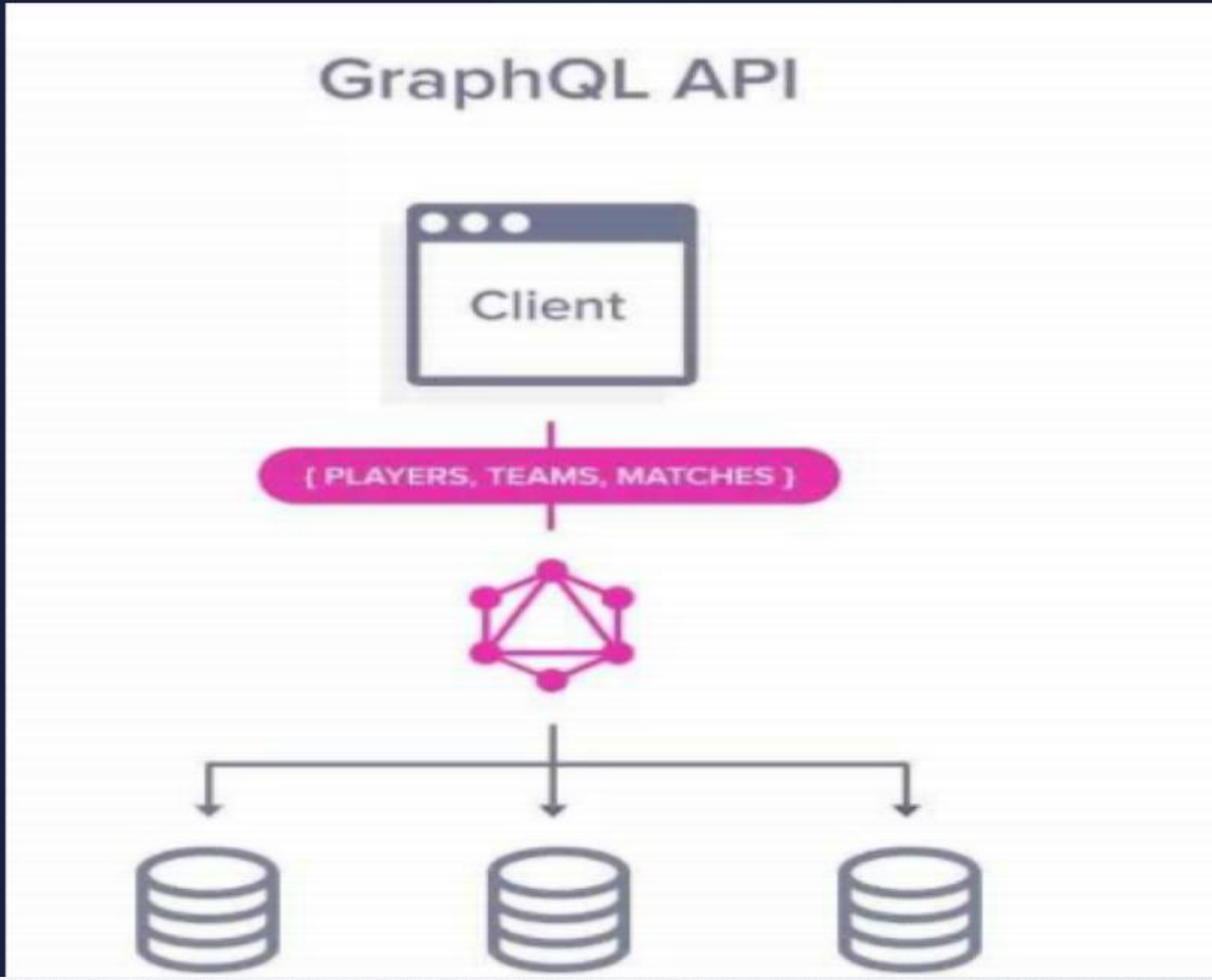
# 搜索与查询

没有The Graph

Web3

# 搜索与查询

有了The Graph



Web3

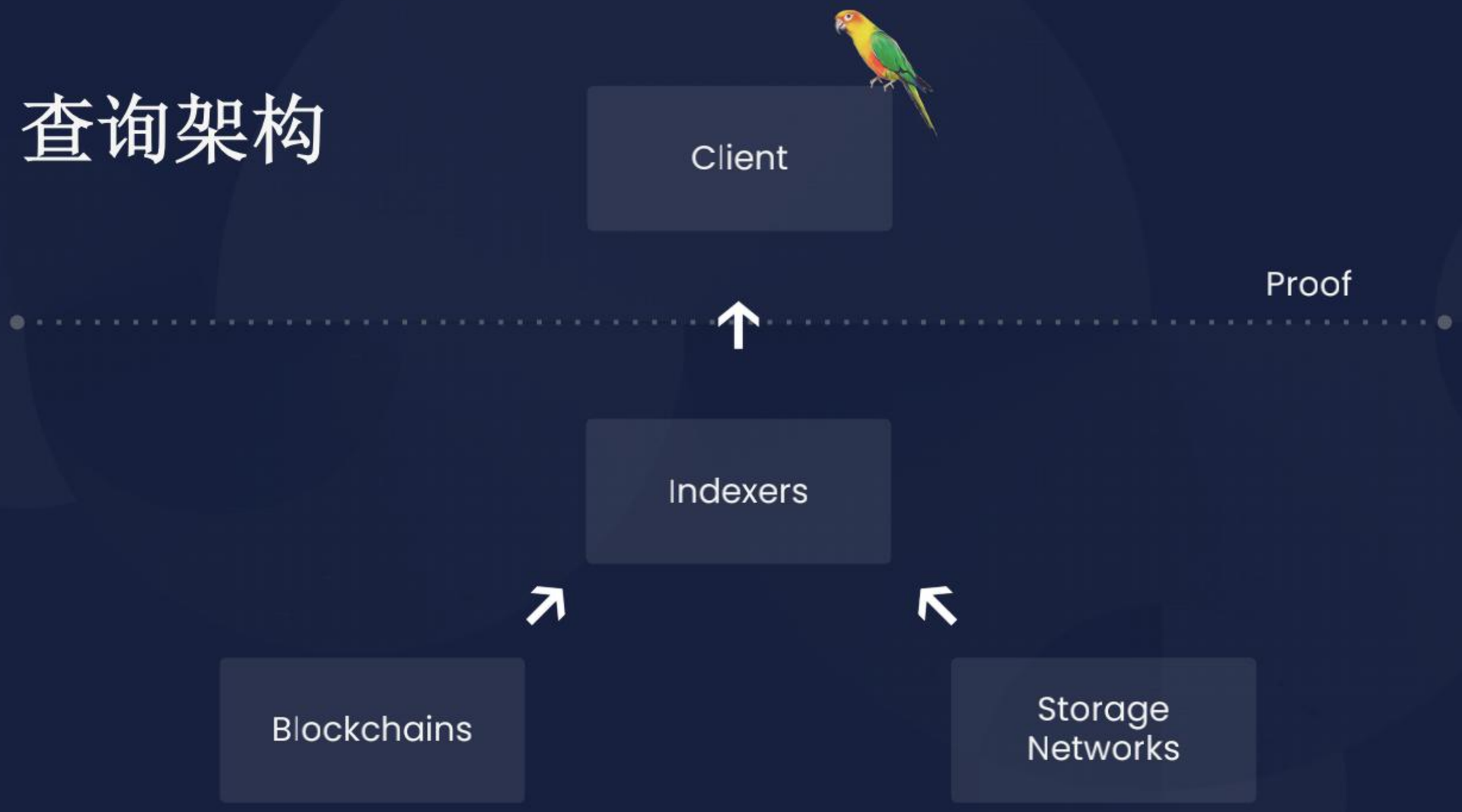
# 搜索与查询

有了The Graph

```
{
  hero {
    name

  }
}
```

```
{
  "hero": {
    "name": "Luke Skywalker"
  }
}
```

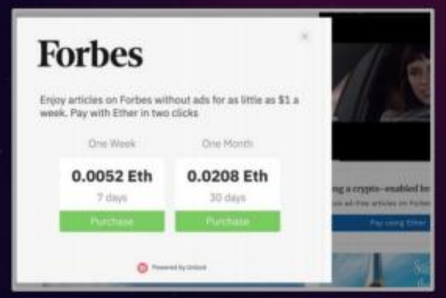
# 查询架构



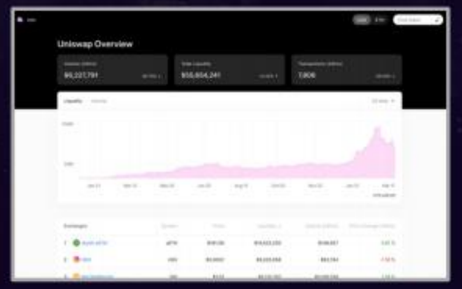
# 支持各种复杂的去中心化应用

交易数据

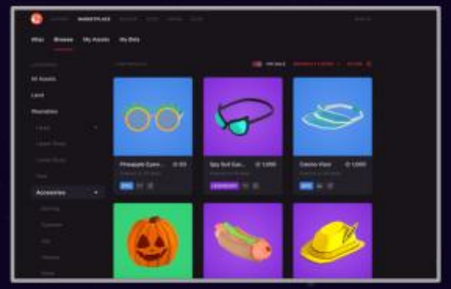
Forbes 订阅数据



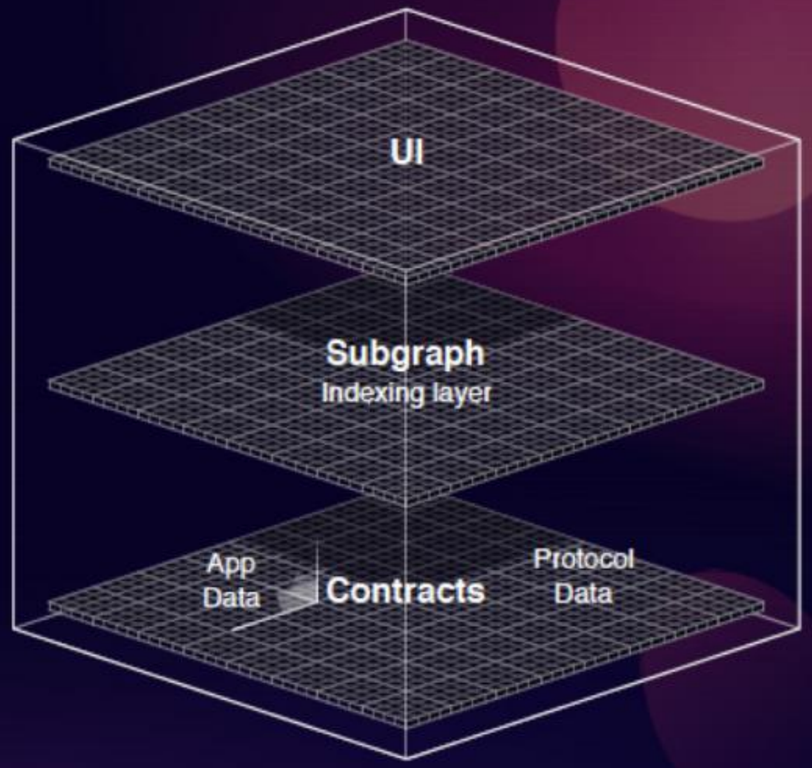

去中心化交易平台




虚拟资产



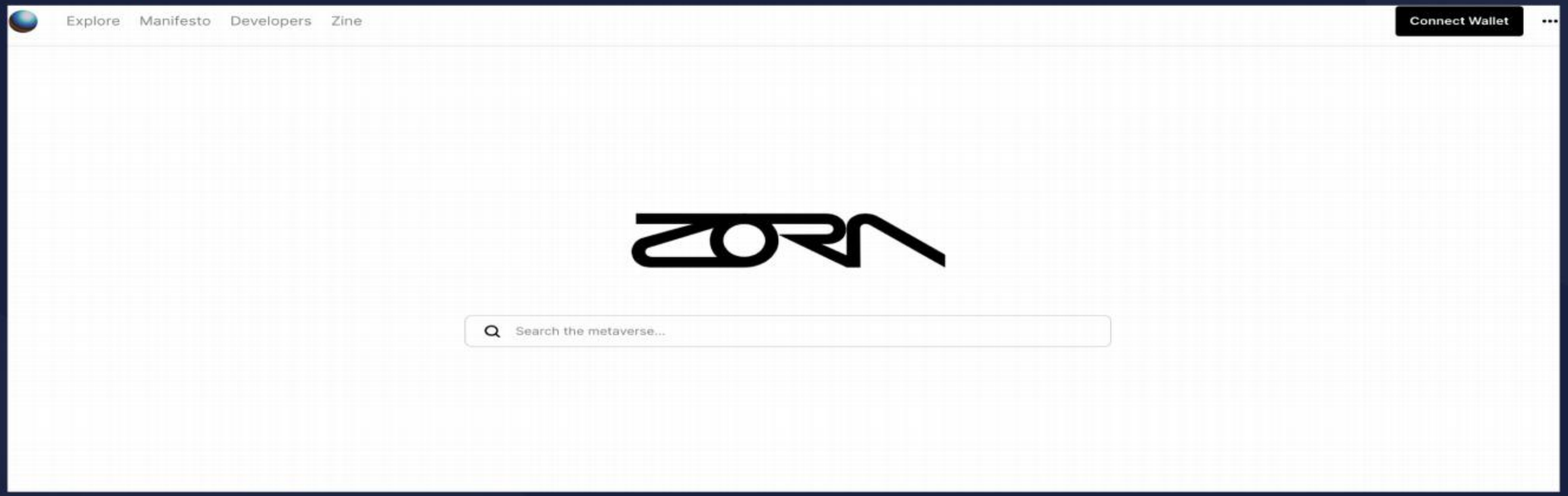
# 成为web3技术栈不可或缺的基础设施



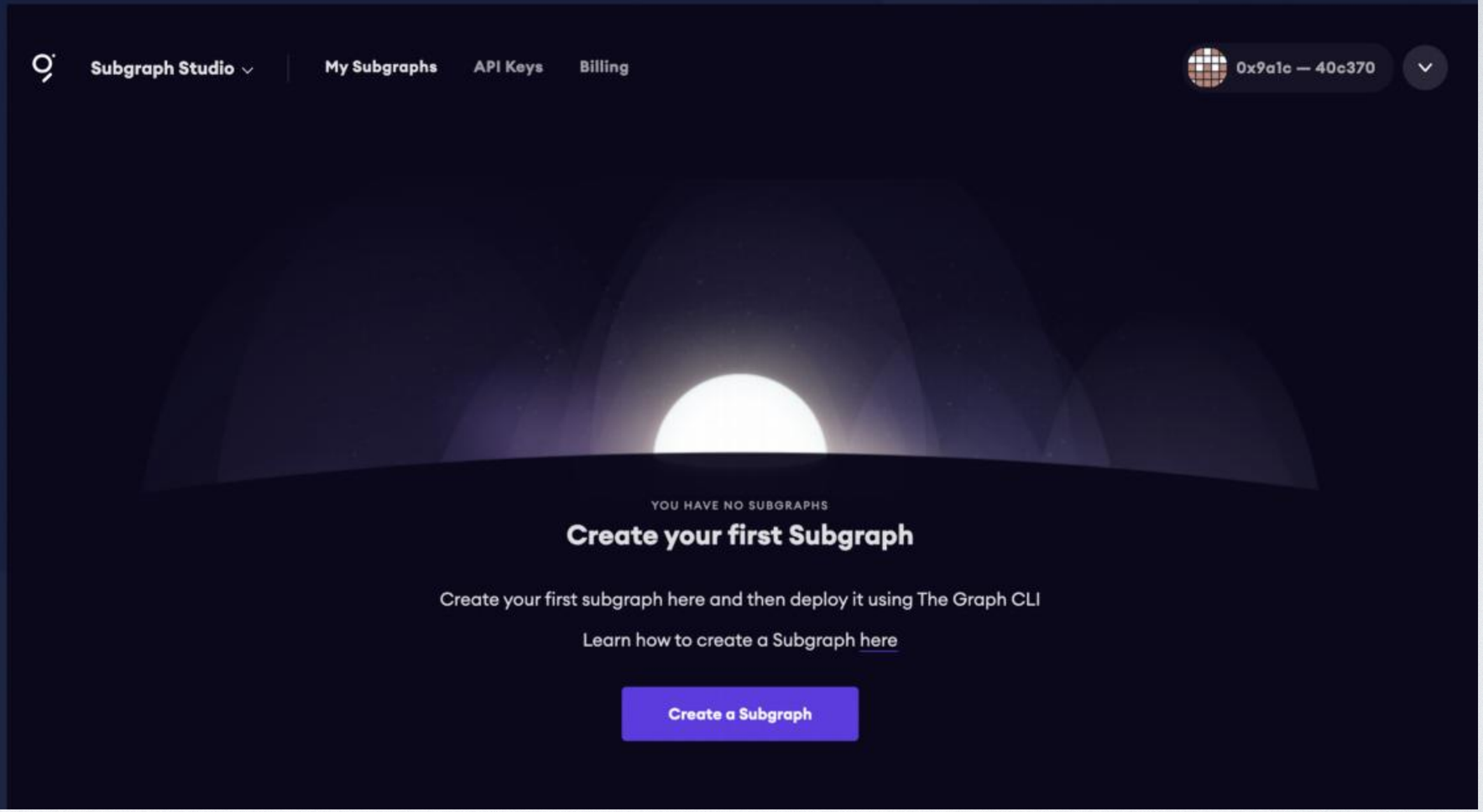
如何构建Subgraph?

# 分布式应用

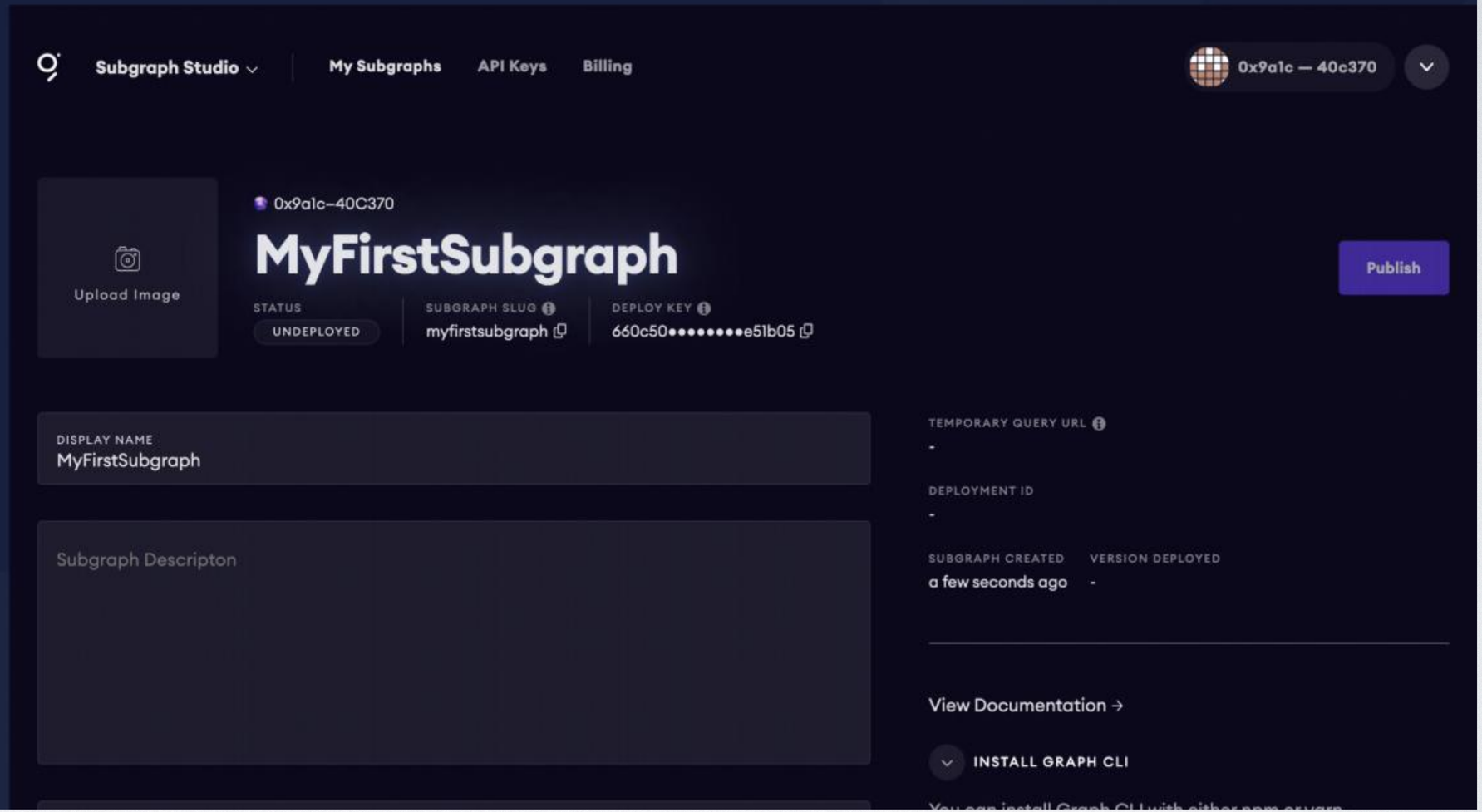
https://zora.co/



# 如何构建Subgraph?



# 如何构建Subgraph?



如何构建Subgraph?

# 工具

```
npm install -g
@graphprotocol/graph-cli
```

如何构建Subgraph?

# 命令

```
graph init --contract-name Token \
--index-events \
--studio \
--from-contract 0xabEFBc9fD2F806065b4f3C237d4b59D9A97Bcac7
```

如何构建Subgraph?

# 代码

subgraph.yaml

schema.graphql

AssemblyScript映射

# Schema

```
type Token @entity {
  id: ID!
  tokenID: BigInt!
  contentURI: String!
  metadataURI: String!
  createdAtTimestamp: BigInt!
  creator: User!
  owner: User!
}

type User @entity {
  id: ID!
  tokens: [Token!]! @derivedFrom(field: "owner")
  created: [Token!]! @derivedFrom(field: "creator")
}
```

# subgraph.yaml

```yaml
schema:
  file: ./schema.graphql
dataSources:
  - kind: ethereum/contract
    name: Token
    network: mainnet
    source:
      address: "0xabEFBc9fD2F806065b4f3C237d4b59D9A97Bcac7"
      abi: Token
      startBlock: 11565020
    mapping:
      kind: ethereum/events
      apiVersion: 0.0.5
      language: wasm/assemblyscript
      entities:
        - Token
        - User
      abis:
        - name: Token
          file: ./abis/Token.json
      eventHandlers:
        - event: TokenMetadataURIUpdated(indexed uint256,address,string)
          handler: handleTokenMetadataURIUpdated
        - event: TokenURIUpdated(indexed uint256,address,string)
          handler: handleTokenURIUpdated
        - event: Transfer(indexed address,indexed address,indexed uint256)
          handler: handleTransfer
```

# 映射文件

```
  TokenURIUpdated as TokenURIUpdatedEvent,
  TokenMetadataURIUpdated as TokenMetadataURIUpdatedEvent,
  Transfer as TransferEvent,
  Token as TokenContract
} from "../generated/Token/Token"

import {
  Token, User
} from '../generated/schema'

export function handleTransfer(event: TransferEvent): void {
  let token = Token.load(event.params.tokenId.toString());
  if (!token) {
    token = new Token(event.params.tokenId.toString());
    token.creator = event.params.to.toHexString();
    token.tokenID = event.params.tokenId;

    let tokenContract = TokenContract.bind(event.address);
    token.contentURI = tokenContract.tokenURI(event.params.tokenId);
    token.metadataURI = tokenContract.tokenMetadataURI(event.params.tokenId);
    token.createdAtTimestamp = event.block.timestamp;
  }
  token.owner = event.params.to.toHexString();
  token.save();

  let user = User.load(event.params.to.toHexString());
  if (!user) {
    user = new User(event.params.to.toHexString());
    user.save();
  }
```

# 映射文件

```
export function handleTokenURIUpdated(event: TokenURIUpdatedEvent): void {
  let token = Token.load(event.params._tokenId.toString());
  if(token){
    token.contentURI = event.params._uri;
    token.save();
  }
}

export function handleTokenMetadataURIUpdated(event: TokenMetadataURIUpdatedEvent): void {
  let token = Token.load(event.params._tokenId.toString());
  if(token){
    token.metadataURI = event.params._uri;
    token.save();
  }
```

完整代码

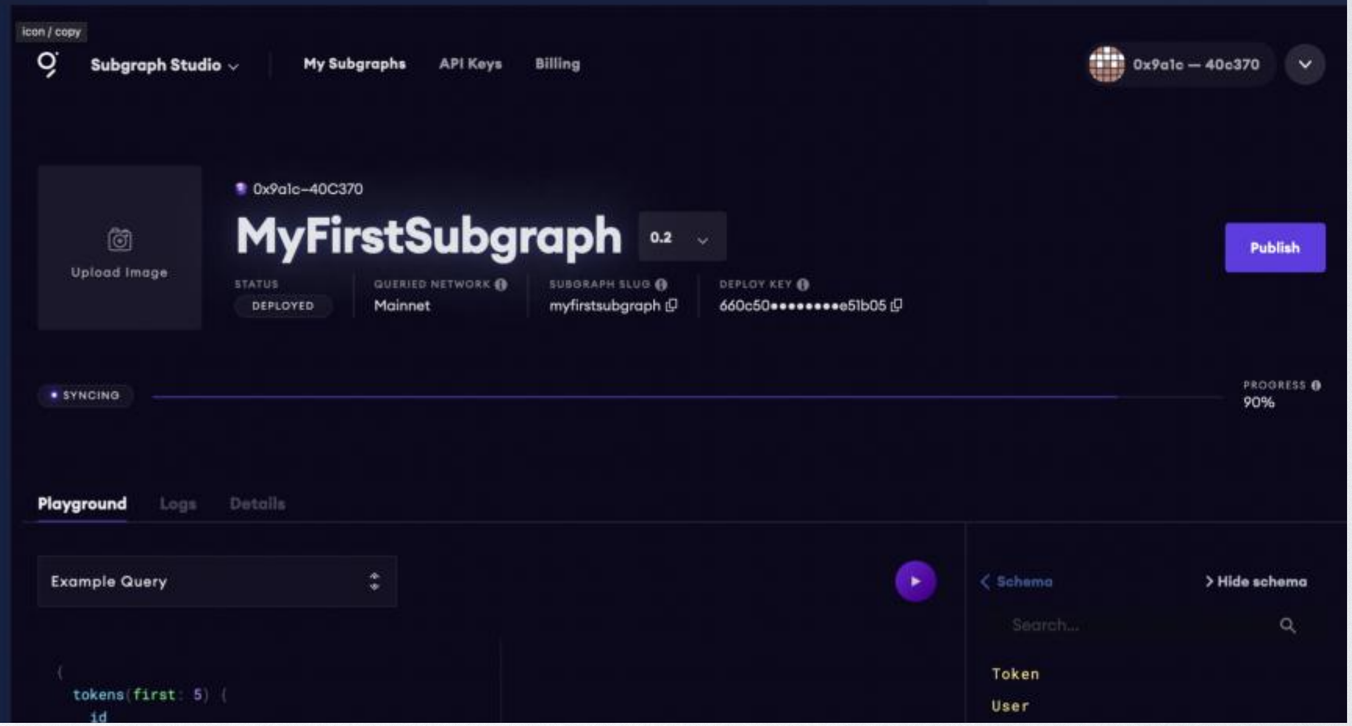
https://github.com/Web3Slimchance/zora-nft-workshop

| | |
|---|---|
| abis | Subgraph Updated |
| src | Subgraph Updated |
| package-lock.json | Subgraph Updated |
| package.json | Subgraph Updated |
| schema.graphql | Subgraph Updated |

如何在Near上构建Subgraph?

# 部署

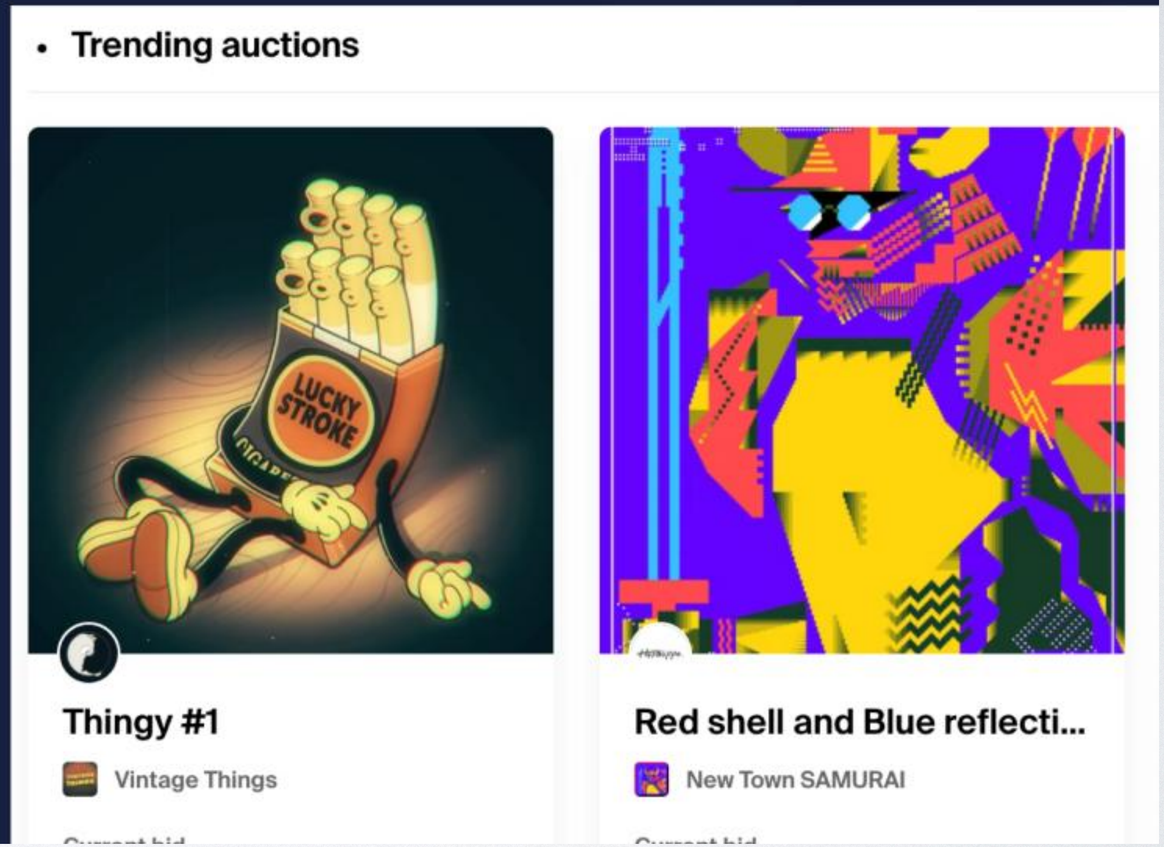
**graph build**
**graph auth –studio**
**graph deploy --studio ‹subgraph-name›**



如何构建Subgraph?

# 举一反三

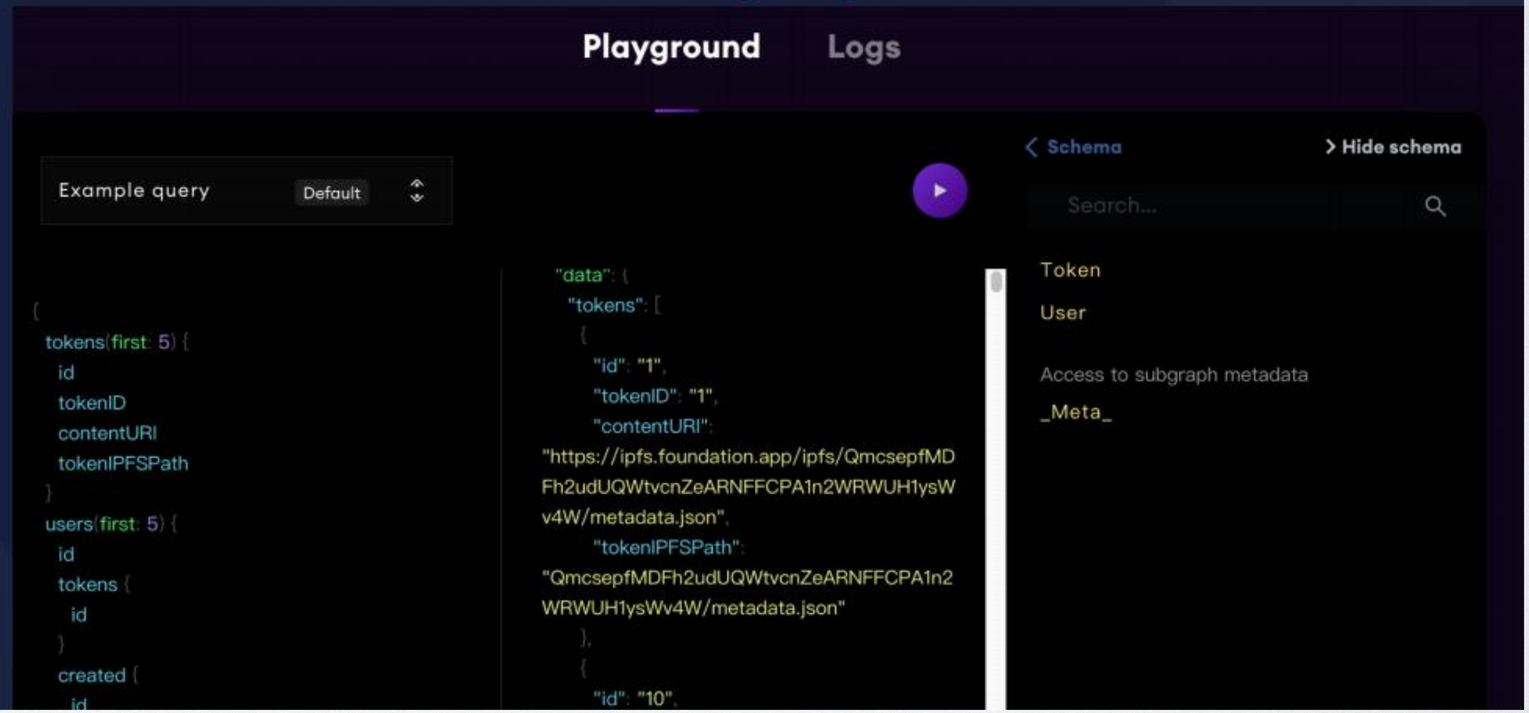
https://foundation.app/
https://github.com/dabit3/building-a-subgraph-workshop



如何查询Subgraph?

# 查询

https://thegraph.com/hosted-service/subgraph/dabit3/foundationsubgraph



# 更多信息

https://thegraph.com

# Q & A

# 练习题

- 发行一个 ERC721 Token：
  - 使用 ethers.js 解析ERC721 转账事件（加分项：记录到数据库中，可方便查询用户持有的所有 NFT）
  - （或）使用 TheGraph 解析ERC721 转账事件

# Gas优化

- 删除不必要的库
- 精确声明Solidity合约函数的可见性
- 合并数据存储：多个小变量打包到单个槽中
- 合约中没有引用的数据用事件
- 避免无限制的循环
- 使用代理进行大规模部署（复用实现合约）
- 链下计算、链上验证（如:数组用链表实现）
- 测试技巧：快进时间，模拟出块

# DEX 分析专场

- Uniswap V2 AMM模型
- 无常损失与 AMM 滑点
- SushiSwap 如何抢Uni 的流动性及质押模型介绍
- Uniswap v3 的创新
- 相关 DEX 协议介绍：0x协议、Curve 协议等

# 什么是DEX?

- 交易所：资产买卖金融市场，证券(股票)交易所、期货交易所、大宗商品交易所…
- DEX： Decentralize Exchange 去中心化交易所（Token 兑换 Token）
- 两类： 订单簿（撮合）、兑换池
- 订单簿：以0x协议为代表，链下订单撮合和链上结算
- 兑换池：以 Uniswap 为代表，自动作市商协议

# Uniswap

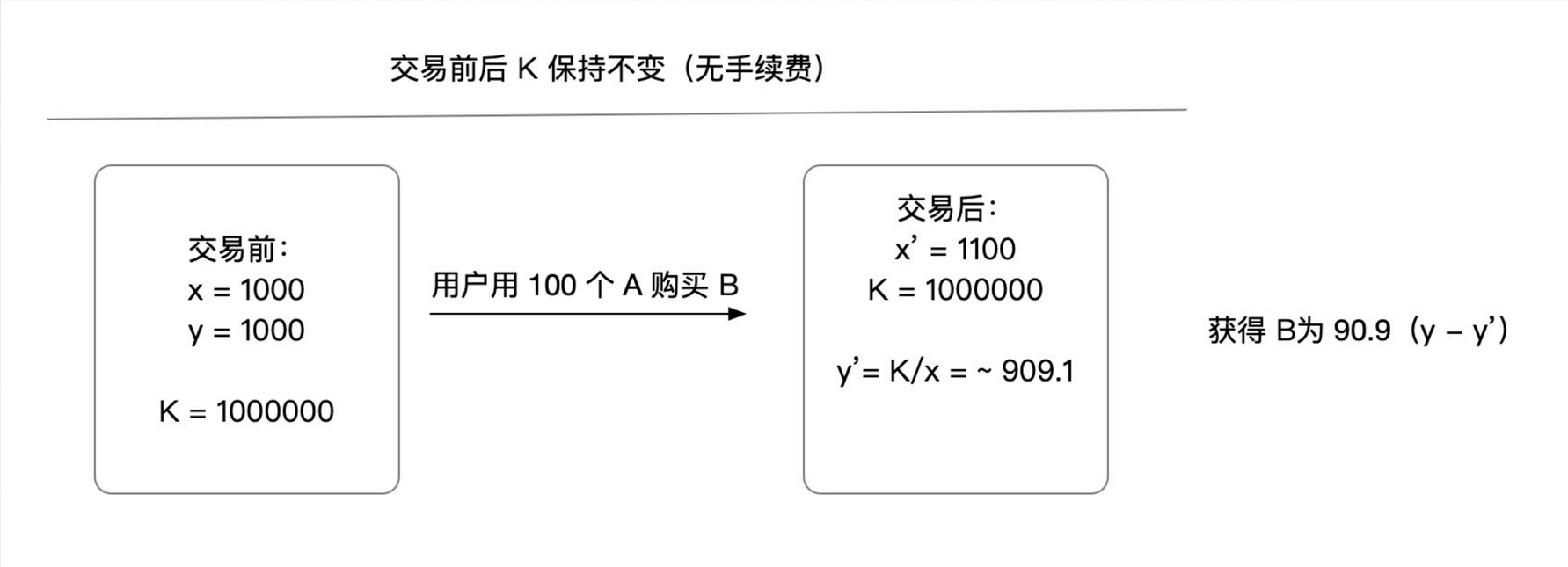
- AMM协议： AutoMated Market Making
  - AutoMate(d)： 自动，没有中间机构进行资金交易
  - Market Making: 做市商（保证订单得以执行），流动性提供者（LP: liquidity providers）
    - 流动性指的是如何快速和无缝地购买或出售一项资产
    - LP 是提供资产的人以实现快速交易。
- 常量乘积模型： K = x * y

# Uniswap

- 常量乘积模型： K = x * y
  - AMM 的执行引擎， 没有价格预言机，价格用公式推导
  - x：token0 的储备量（reserve0）
  - y：token1 的储备量（reserve1）
  - 提供流动性：
    - 转入token0、token1，增加reserve0、reserve1，拿到流动性凭证 = sqrt(x * y)
  - 兑换时，K 保持不变
    - 减少reserve0，就必须增加reserve1
    - 减少reserve1，就必须增加reserve0
  - 移除流动性
    - 通过流动性凭证，撤出token0、token1

# Uniswap

- 常量乘积模型： K = x * y




价格滑点（slippage）：一次交易使价格改变的程度， 单笔交易量越大对价格的影响越大

# Uniswap

交易前后 K 保持不变（扣除0.3%费）

交易前：
x = 1000
y = 1000

K = 1000000

用户用 100 个 A 购买 B
用于交易的A 为 99.7

交易后：
x ’ = （1000 + 99.7） + 0.3

y ’ = K / (1000 + 99.7) = 909.33

K = 1000000
K’ = 1000263

获得 B： 90.67 （y – y’）

$$x \cdot y = (x + \Delta x)(y - \Delta y) = k$$

$$\Delta y = y - \frac{x \cdot y}{x + \Delta x} = \frac{\Delta x y}{x + \Delta x}$$

# Uniswap

添加流动性 增加 K

交易前：
x = 1000
y = 1000

K = 1000000
Lp token= sqrt(K)

100 A： 100 B →

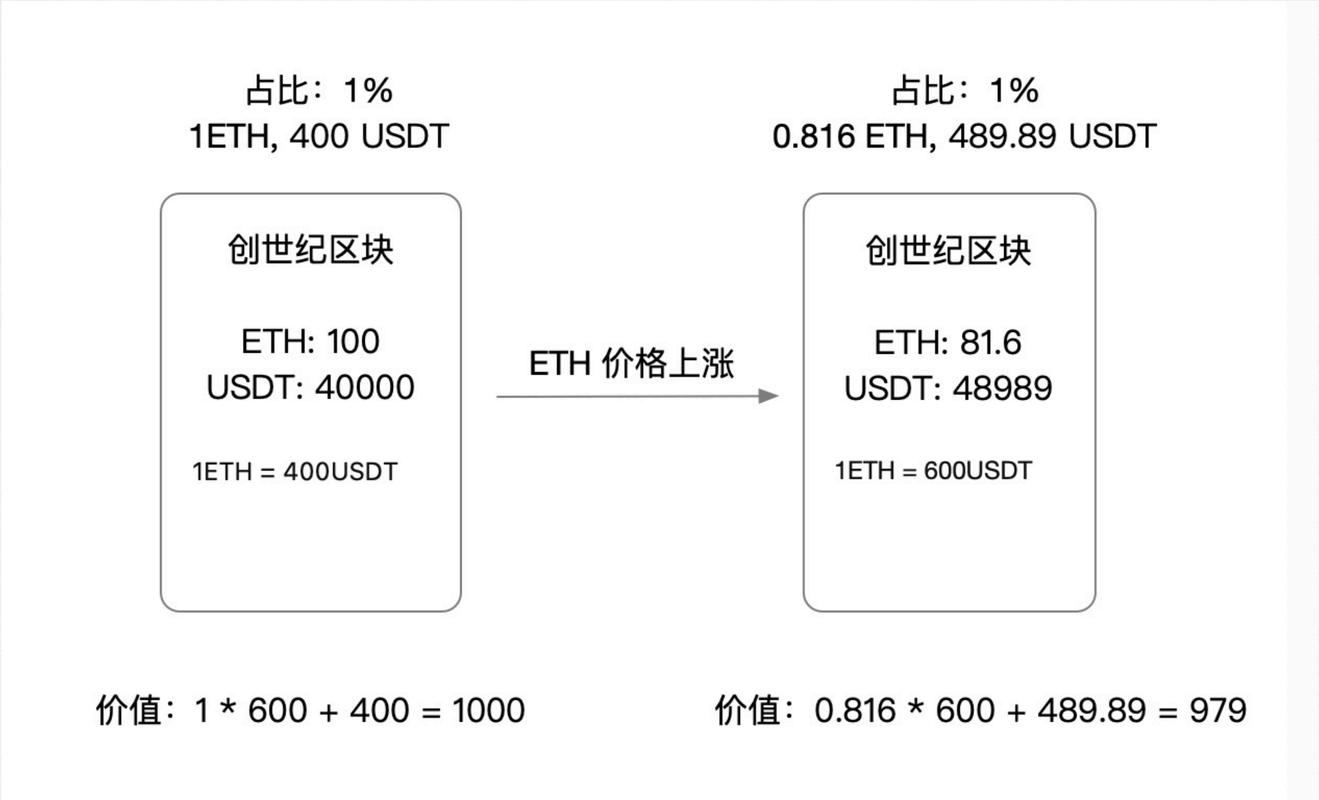
交易后：
x ’ = 1100
y ’ = 1100

K’ = 1210000

获得 lp token：100

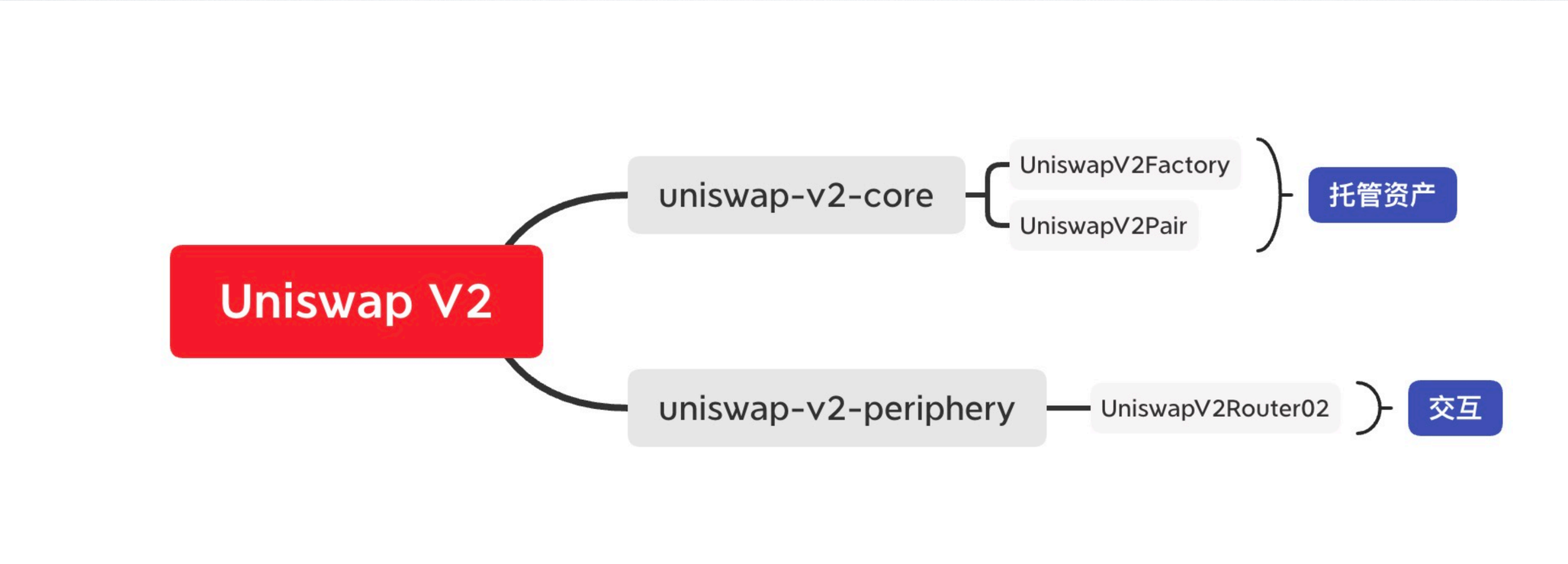
# Uniswap – 无常损失

- 流动性提供者无常损失：一对代币存入Uniswap后，如果一种代币以另一种进行计价的价格上升，在价格上升后取出，总价格比原价值低一些，低的部分就是损失。
- 思考：为什么会造成损失?



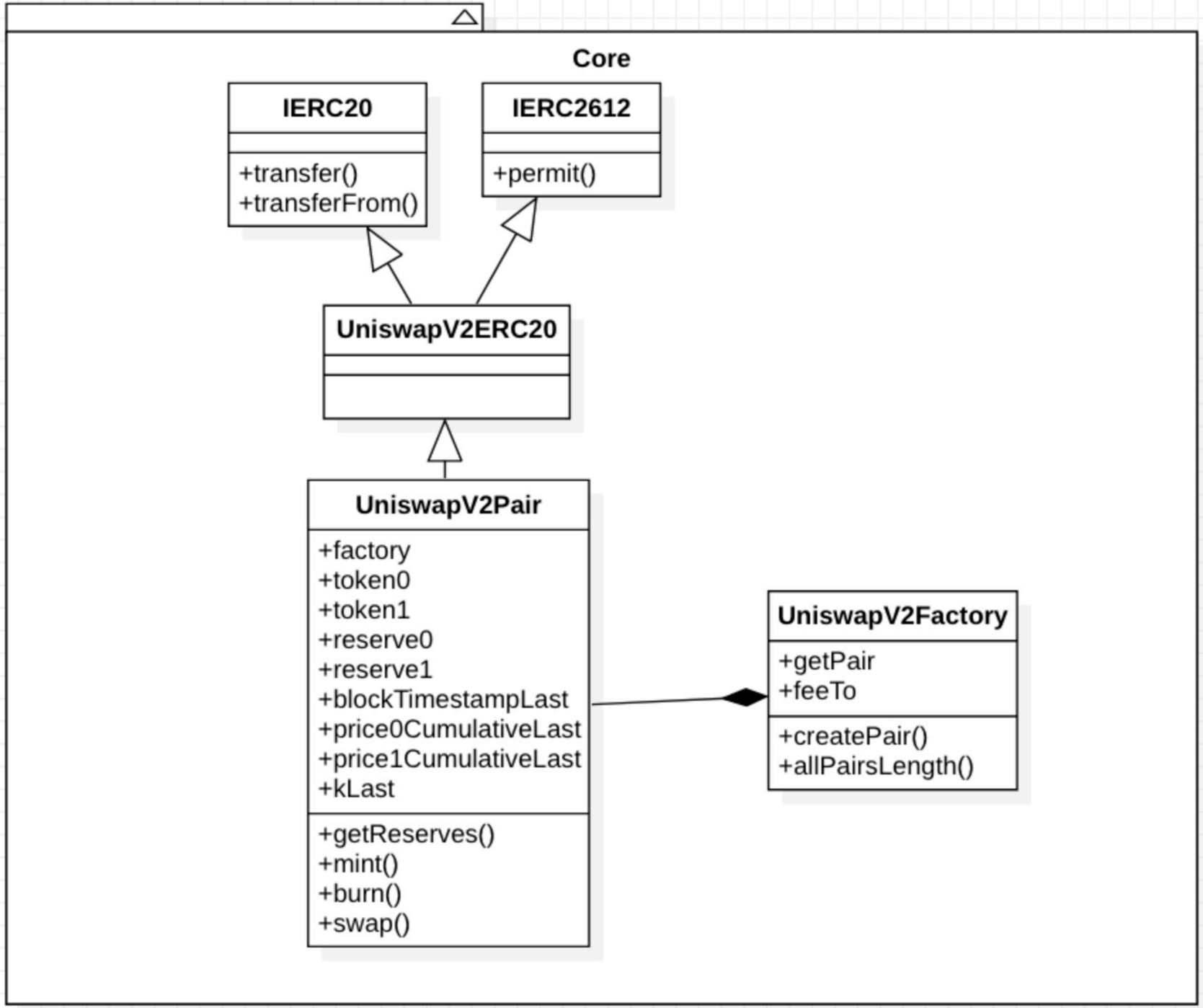

https://zhuanlan.zhihu.com/p/268435169

# Uniswap – 代码分析

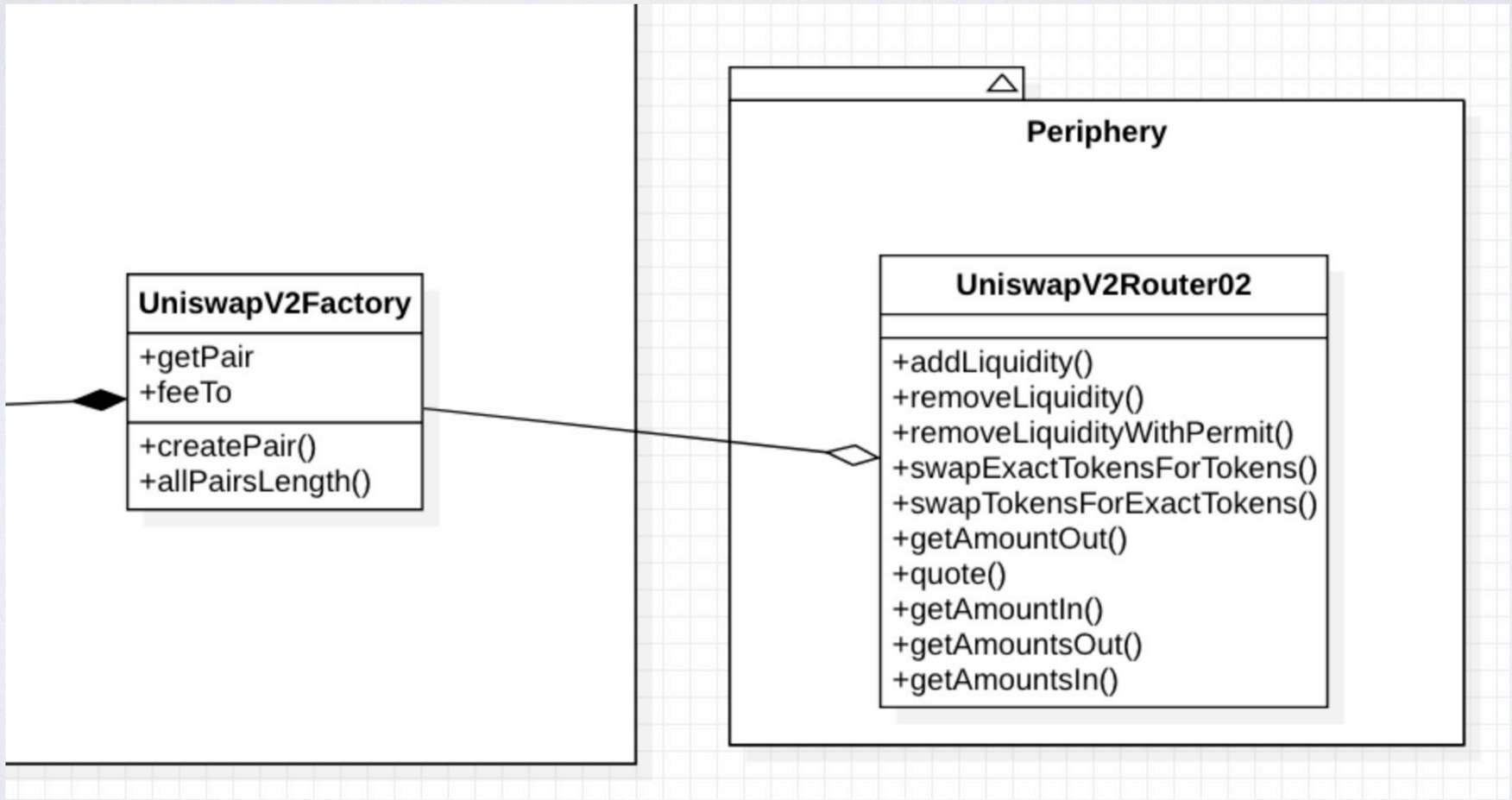



https://github.com/Uniswap/v2-core https://github.com/Uniswap/v2-periphery

https://github.com/xilibi2003/v2-core https://github.com/xilibi2003/v2-periphery


Core
IERC20
+transfer()
+transferFrom()
IERC2612
+permit()
UniswapV2ERC20
UniswapV2Pair
+factory
+token0
+token1
+reserve0
+reserve1
+blockTimestampLast
+price0CumulativeLast
+price1CumulativeLast
+kLast
+getReserves()
+mint()
+burn()
+swap()
UniswapV2Factory
+getPair
+feeTo
+createPair()
+allPairsLength()

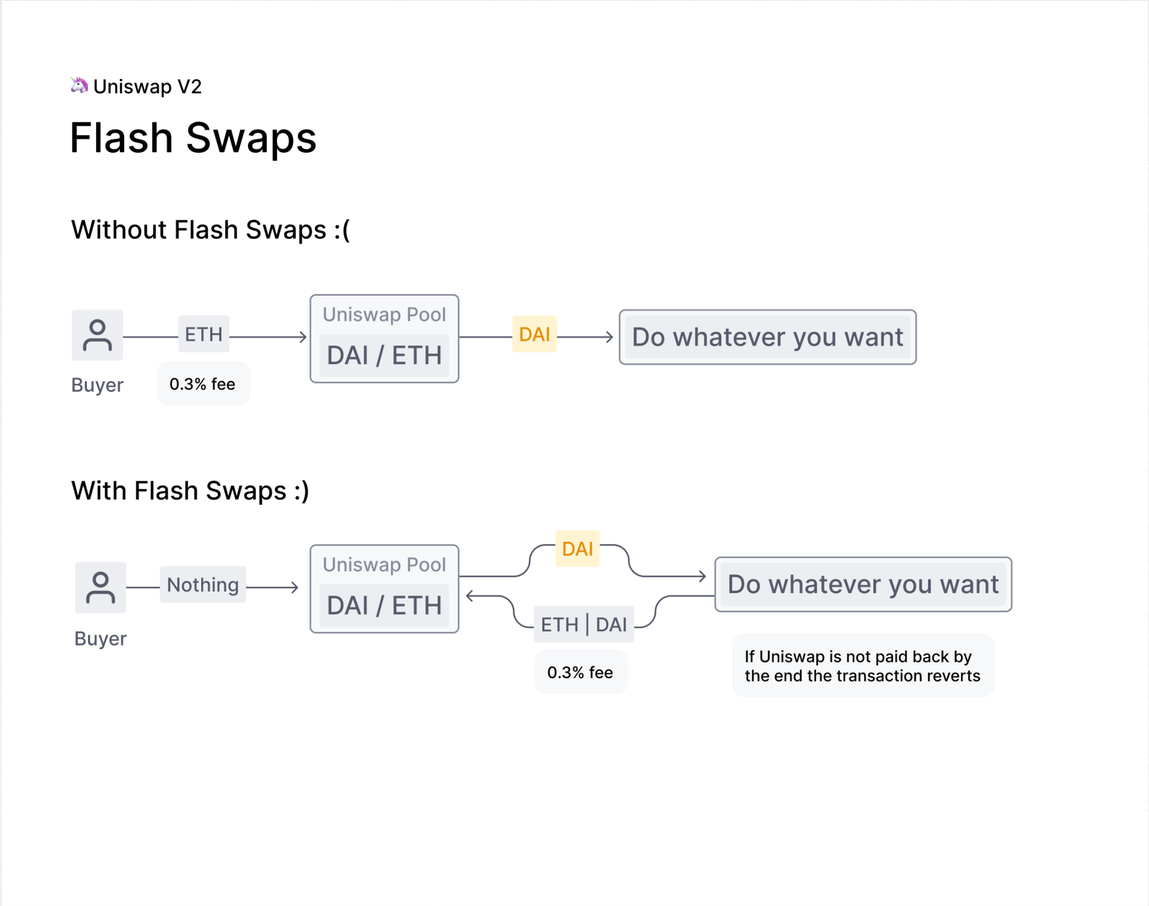
# Uniswap – 合约结构



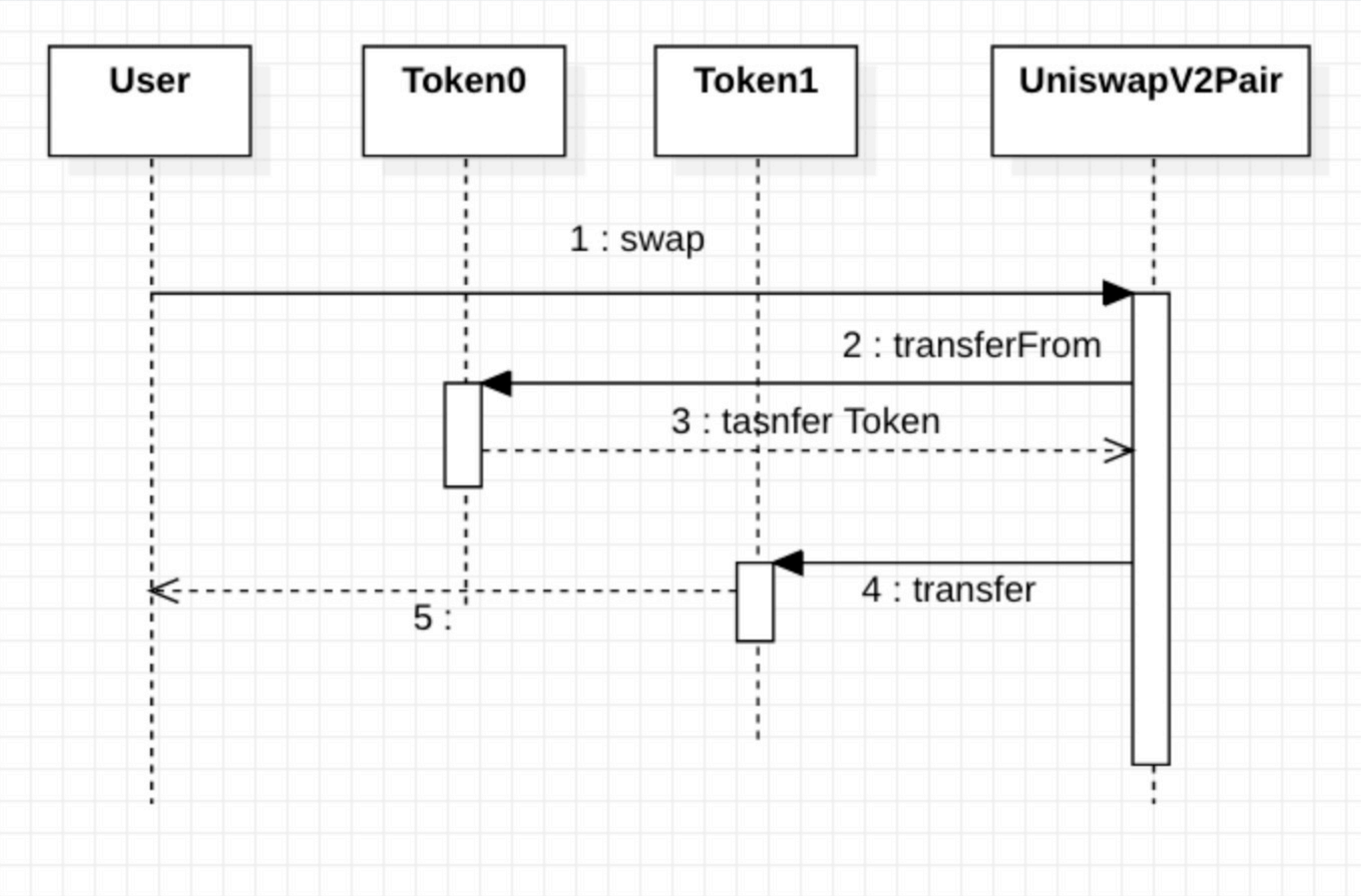
# Uniswap 集成使用

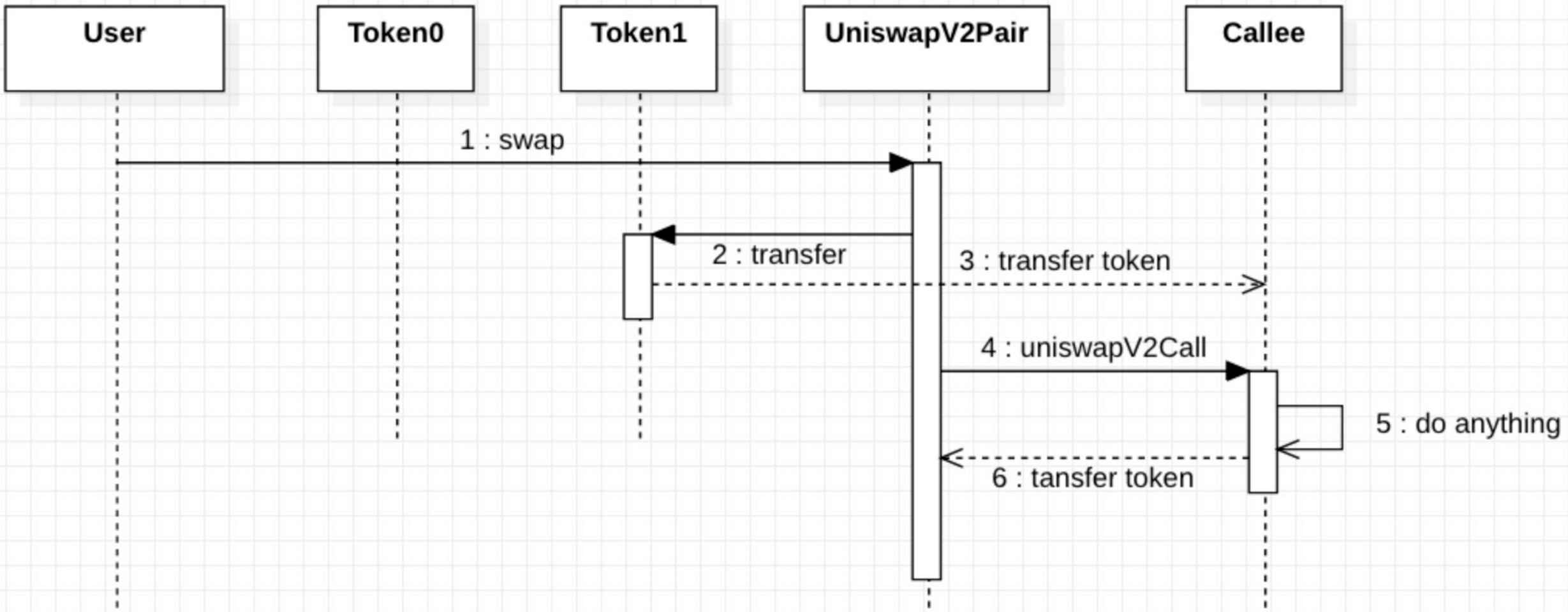
- 交易平台（通过 Router 与其他币进行交易）
- 价格预言机（获取时间加权平均价格）
- 闪电兑（flash swap）

# Uniswap 闪电兑


Uniswap V2
Flash Swaps
Without Flash Swaps :(
Buyer
ETH
0.3% fee
Uniswap Pool
DAI / ETH
DAI
Do whatever you want
With Flash Swaps :)
Buyer
Nothing
Uniswap Pool
DAI / ETH
DAI
ETH | DAI
0.3% fee
Do whatever you want
If Uniswap is not paid back by
the end the transaction reverts

普通兑换

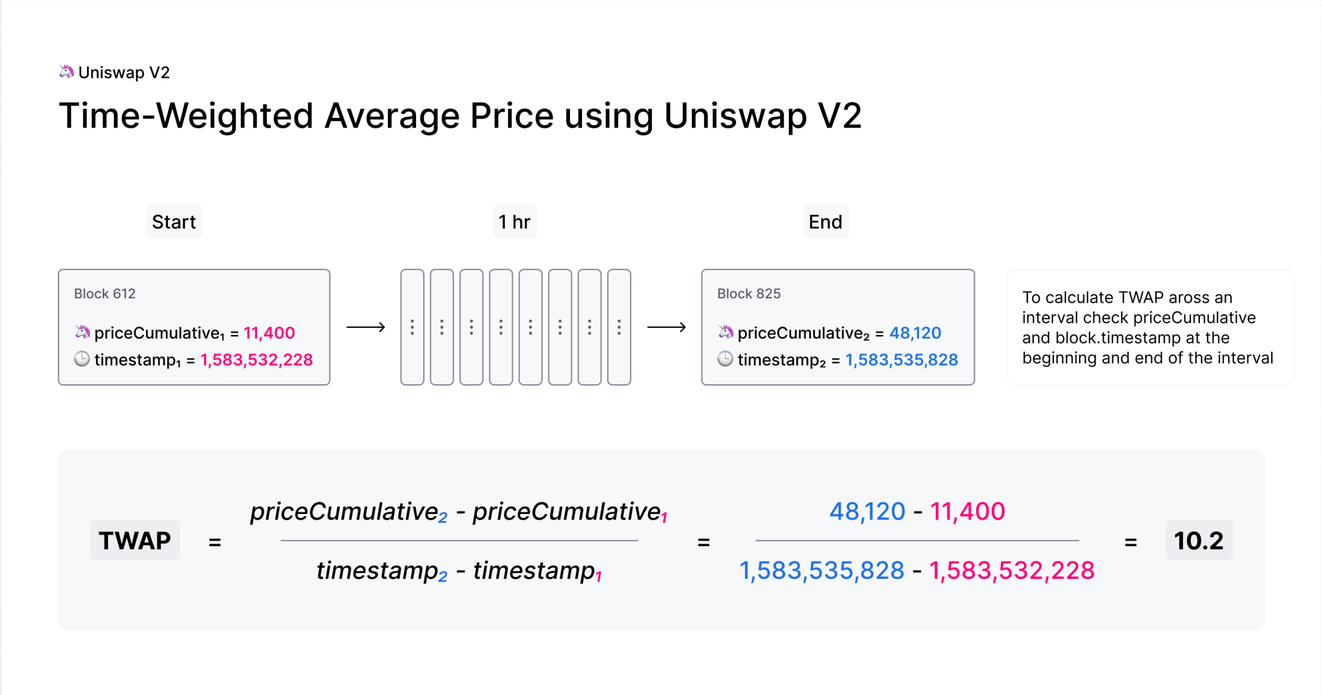

User
Token0
Token1
UniswapV2Pair
1 : swap
2 : transferFrom
3 : tasnfer Token
4 : transfer
5 :

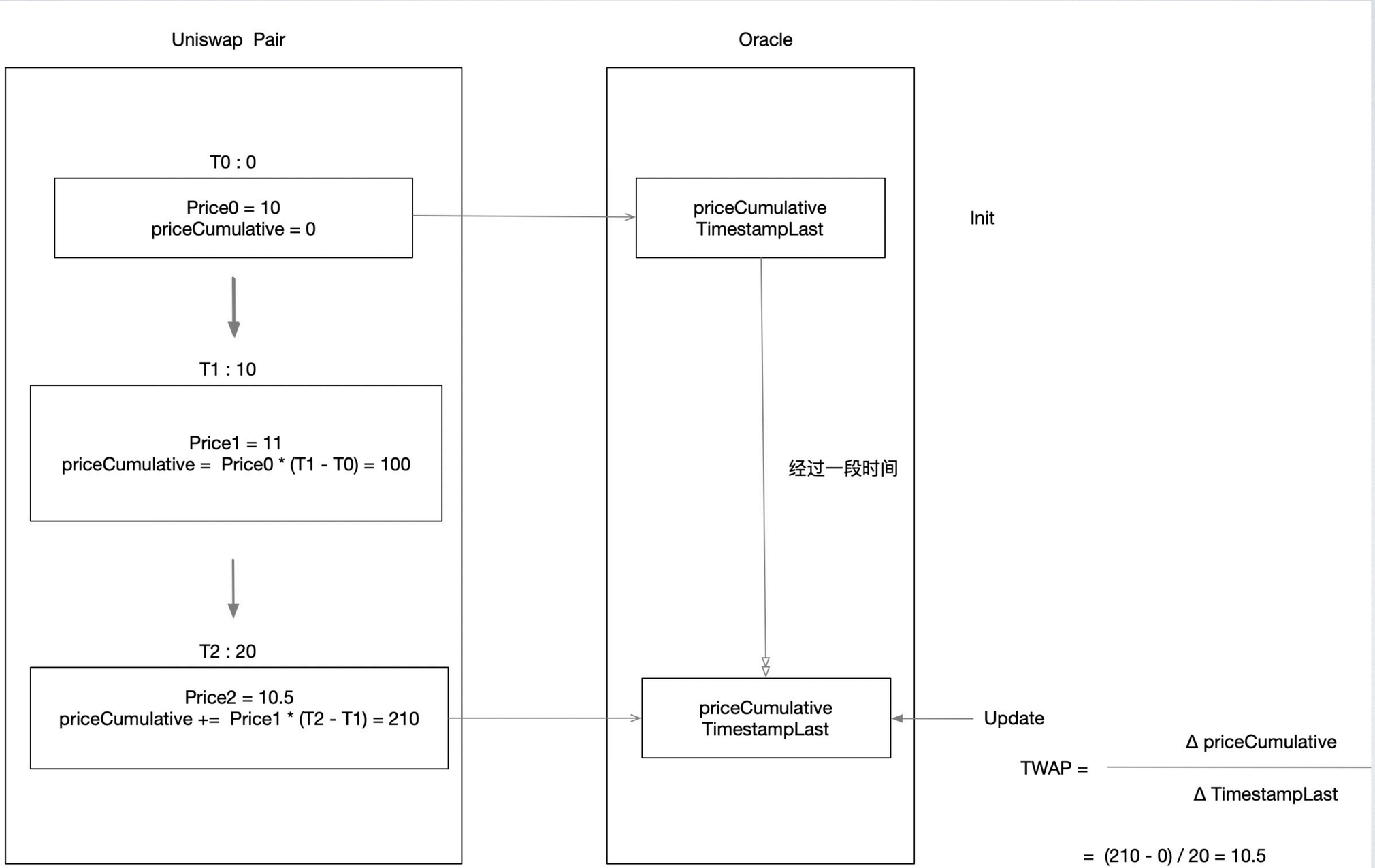

User
Token0
Token1
UniswapV2Pair
Callee
1 : swap
2 : transfer
3 : transfer token
4 : uniswapV2Call
5 : do anything
6 : tansfer token

闪电兑换

# Uniswap TWAP 价格

- TWAP = Time-Weighted Average Price, 即时间加权平均价格
- price0CumulativeLast、 price1CumulativeLast 记录了 token 的累加价格
  - priceCumulativeLast1 = priceCumulativeLast1 + price2 * timeElapsed;




Uniswap Pair
Oracle
T0 : 0
Price0 = 10
priceCumulative = 0
priceCumulative
TimestampLast
Init
T1 : 10
Price1 = 11
priceCumulative = Price0 * (T1 - T0) = 100
经过一段时间
T2 : 20
Price2 = 10.5
priceCumulative += Price1 * (T2 - T1) = 210
priceCumulative
TimestampLast
Update
TWAP = Δ priceCumulative / Δ TimestampLast
= (210 - 0) / 20 = 10.5

# Q & A

# 练习题

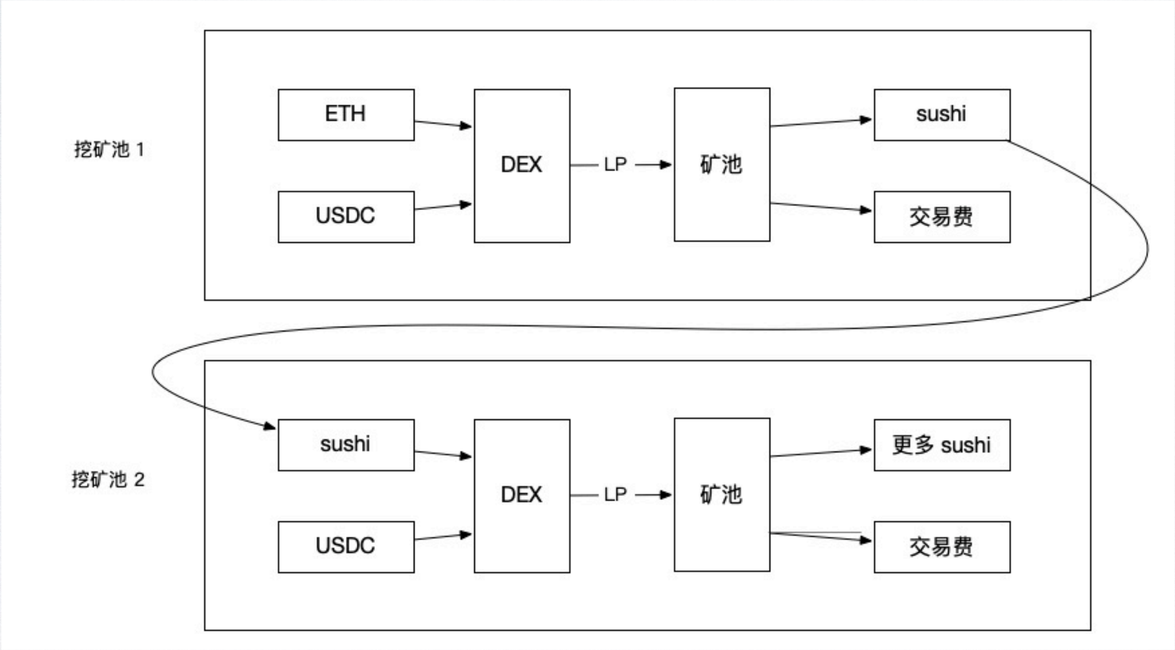
- 部署自己的 ERC20 合约MyToken
- 编写合约MyTokenMarket 实现：
  - AddLiquidity(): 函数内部调用 Uniswap V2Router 添加MyToken与 ETH 的流动性
  - buyToken(): 用户可调用该函数实现购买MyToken

# SushiSwap

- Uniswap:
- 提供流动性时才赚取资金池的交易费，撤回流动性不再获得相应的收入

- SushiSwap = Uniswap + 流动性挖矿
  - 流动性挖矿：
    - 1. 为流动性提供者奖励
    - 2. 锁定流动性、弥补流动性损失
  - 公平发行协议币（sushi）： 去中心化治理、sushi 代币可持续给参与者带来协议收益

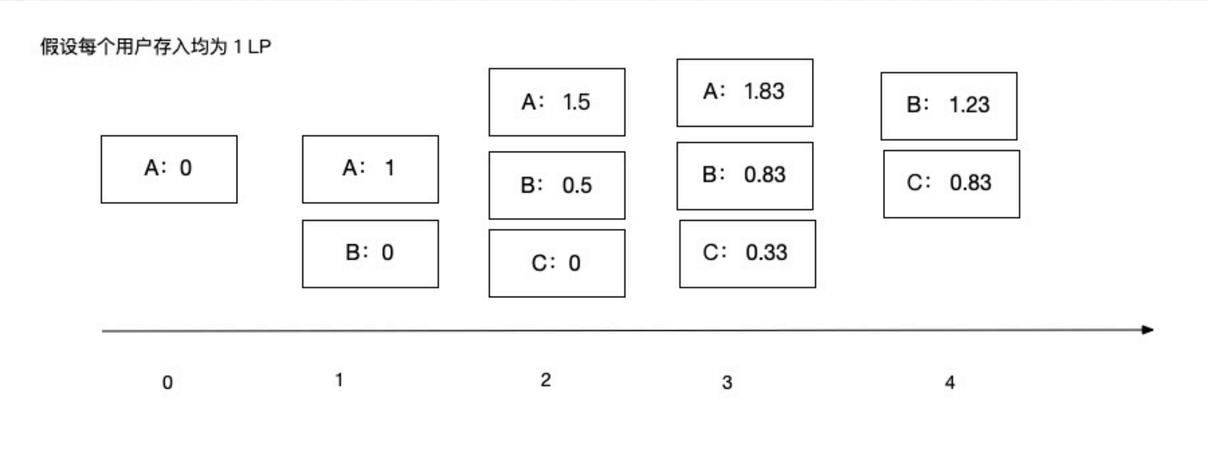
# 流动性挖矿




- 正向：有效的激励工具，尤其是更高激励的池2 ，促使人们购买 sushi，推高sushi 价格，同时提高了挖矿池的收益。促进更多人购买 sushi
- 反向：随着挖出的SUSHI越来越多，矿工卖SUSHI，某个时间点，卖出的SUSHI比买入的SUSHI多，SUSHI价格开始下降，收益下跌、抛售、进一步下跌，“矿塌了”。

# SushiSwap 挖矿算法

- 设置每个区块可以奖励 sushi Token 数量
- 根据投入的 LP 数量的比例，获取对应的 sushi
- 怎么在有限 gas 下实现？常规方案：



# SushiSwap 挖矿算法

- 全局变量： accSushiPerShare, 每个份额可以拿到的**累计**奖励

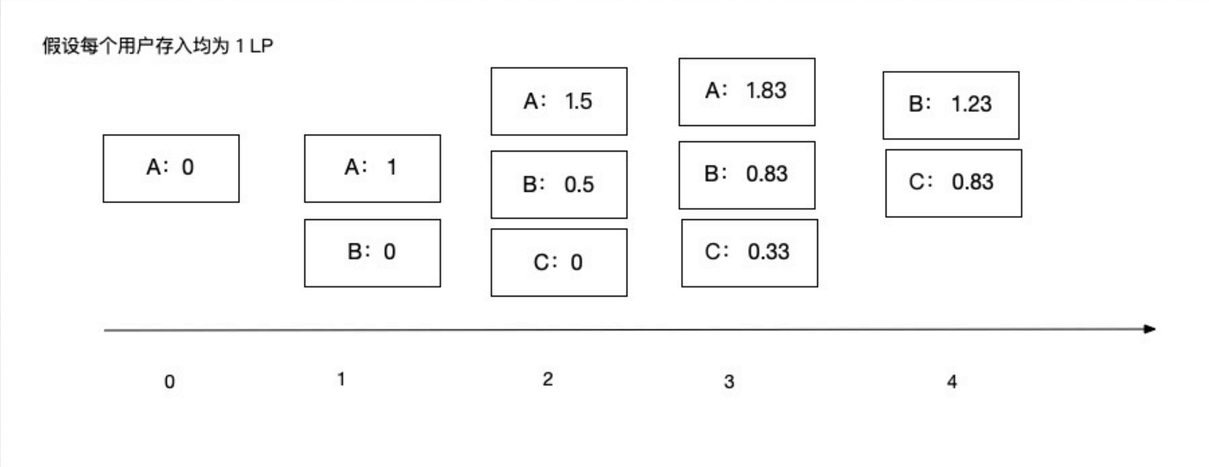
$$accSushiPerShare = R \cdot \sum_{t=0}^{t} \frac{1}{L(t)}$$

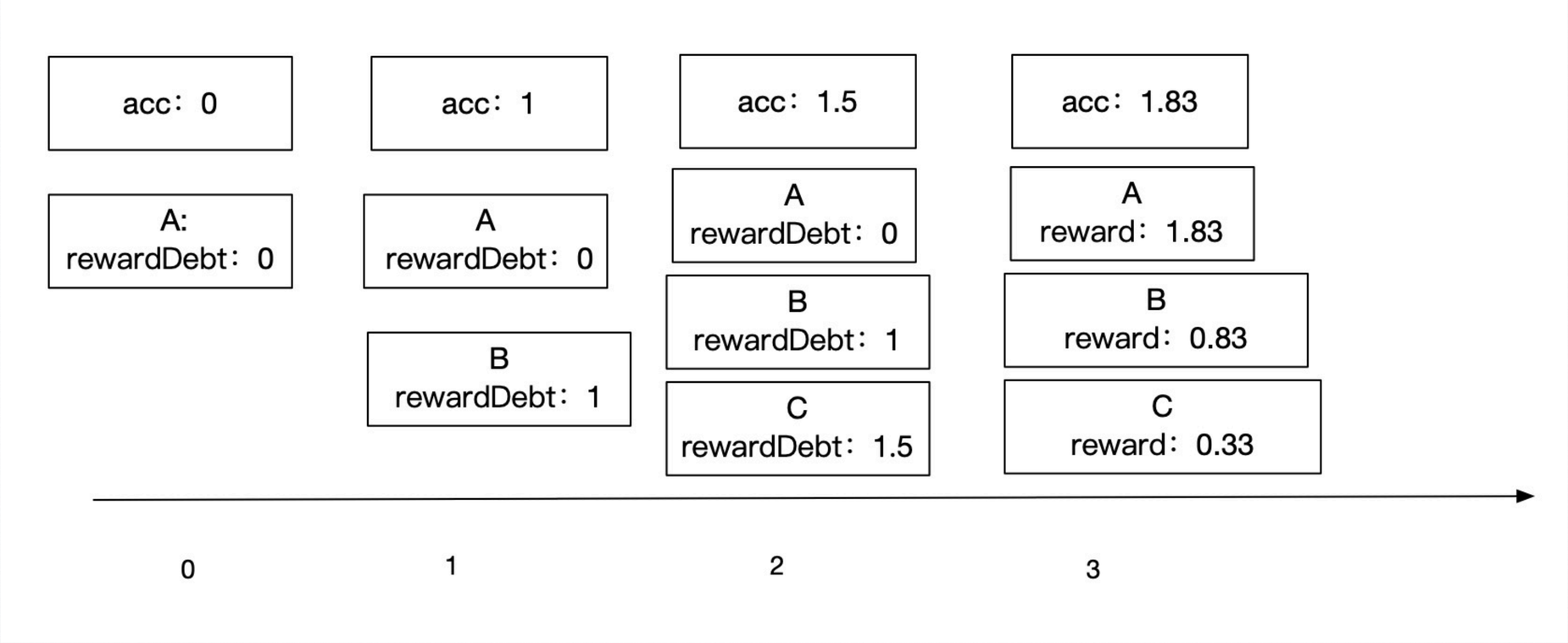
- 记录进入时刻已奖励部分： rewardDebt

$$rewardDebt = l \cdot R \cdot \sum_{t=0}^{t_0} \frac{1}{L(t)}$$

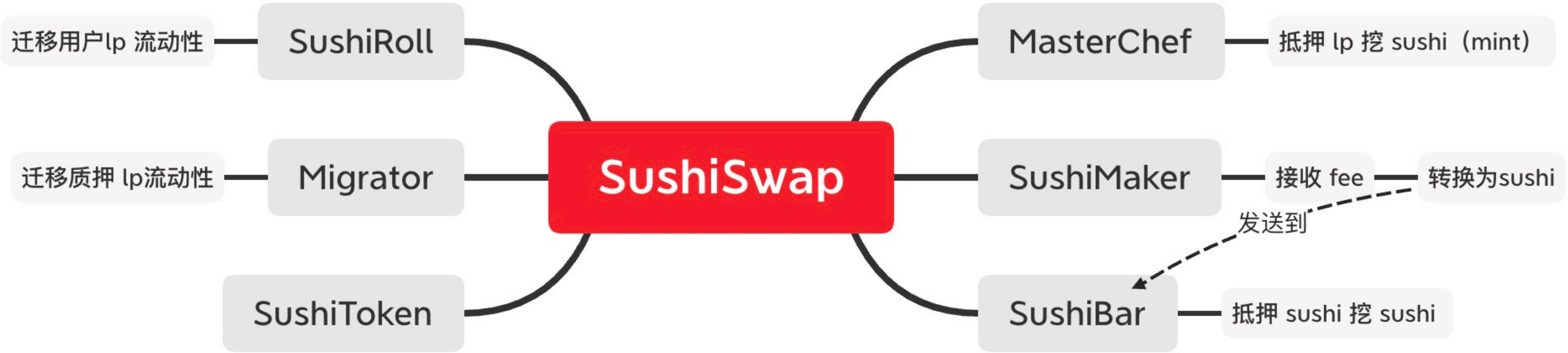
- 奖励为： accSushiPerShare * amount - rewardDebt

$$Reward_{alice} = l \cdot (R \cdot \sum_{t=0}^{t} \frac{1}{L(t)} - R \cdot \sum_{t=0}^{t_0} \frac{1}{L(t)})$$


假设每个用户存入均为 1 LP
A：0
A：1
B：0
A：1.5
B：0.5
C：0
A：1.83
B：0.83
C：0.33
B：1.23
C：0.83
0
1
2
3
4


acc：0
A:
rewardDebt：0
acc：1
A
rewardDebt：0
B
rewardDebt：1
acc：1.5
A
rewardDebt：0
B
rewardDebt：1
C
rewardDebt：1.5
acc：1.83
A
reward：1.83
B
reward：0.83
C
reward：0.33
0
1
2
3

# Sushiswap – 代码分析




https://github.com/sushiswap/sushiswap

# Sushiswap – 代码分析

- MasterChef：流动性挖矿
  - deposit()： 存 LP
  - Withdraw(): 取 LP
  - pendingSushi(): 查看收益
- SushiMaker：LP代币(DEX 交易手续 ) 转 sushi
  - convert()
- SushiBar: 质押 sushi， -> xSuishi, -> 更多 sushi
  - enter： 质押
  - leave(): 取sushi

https://github.com/sushiswap/sushiswap

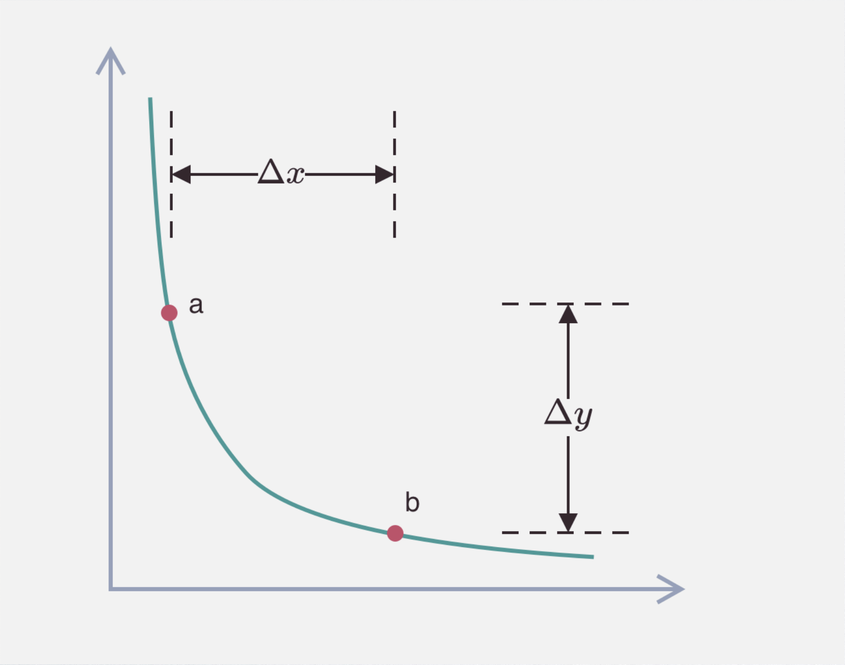
# Q & A

# 练习题

- 在上一次作业的基础上：
  - 完成代币兑换后， 直接质押 MasterChef
  - withdraw() : 从 MasterChef 提取 Token 方法

# Uniswap V2 的问题

- 在所有价格段提供资金（0,infinity），资金利用率低




资金利用率（Capital Efficiency） = 实际使用的资金/ 所提供的资金

4500 DAI（X） 和 3 ETH（Y）

K = 4500 * 3 = 13500

P1 = 1500： X1 = 4500; Y1 = 3

P2 = 1300： X1 = 4192.54; Y1 = 3.22 CF = ΔX / X1 = 6.84%

# Uniswap V3

- 最大创新：通过集中式流动性提高资金利用率

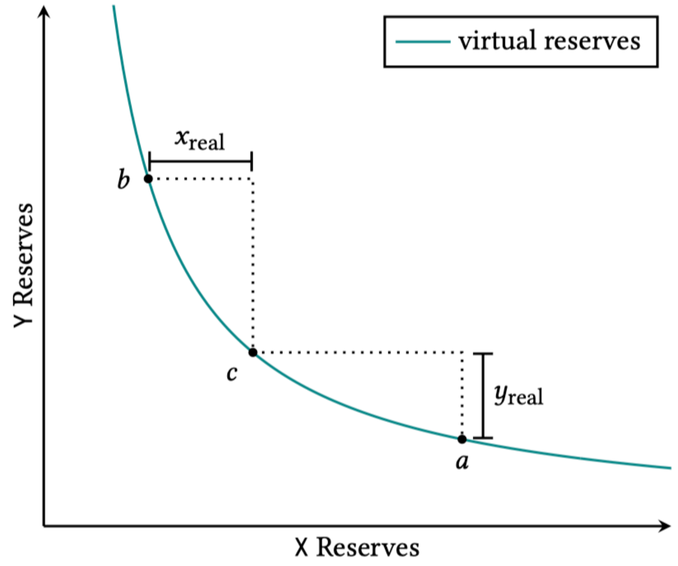



**Figure 1: Simulation of Virtual Liquidity**

1. 价格通常两点之间波动（a点和b点）
2. c 到 a， 消耗资金为 Yreal
3. c 到 b， 消耗资金为 Xreal
4. 若在 [a,b] 提供流动性，仅需提供Xreal Yreal

# Uniswap V3

- 虚拟流动性

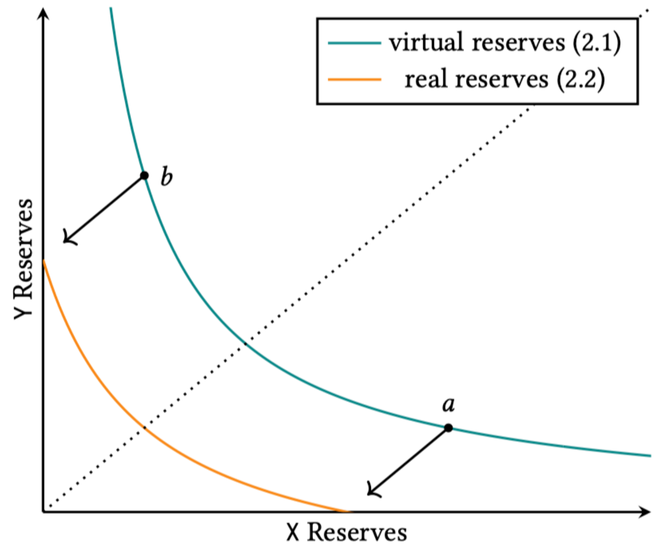



**Figure 2: Real Reserves**

$$(x + \frac{L}{\sqrt{p_b}})(y + L\sqrt{p_a}) = L^2$$

1. 引入虚拟流动性 K = (X+Xvirtual) * (Y+Yvirtual)
2. L: 流动性, p: 价格 (y的)

$$x * y = L^2$$

$$x * (p * x) = L^2$$

$$p * x^2 = L^2$$

$$x^2 = L^2/p$$

$$K = L^2$$

$$p = y/x$$

若 x_real用完，需要的虚拟流动性为

$$x = L/\sqrt{p}_b, y = L * \sqrt{p}_a$$

# Uniswap V3

- 流动性

1. L：流动性，p：y的价格

$$K = L^2$$
$$p = y/x$$

代码中存 L和sqrt(P)

$$x = L/\sqrt{p}$$
$$y = L * \sqrt{p}$$

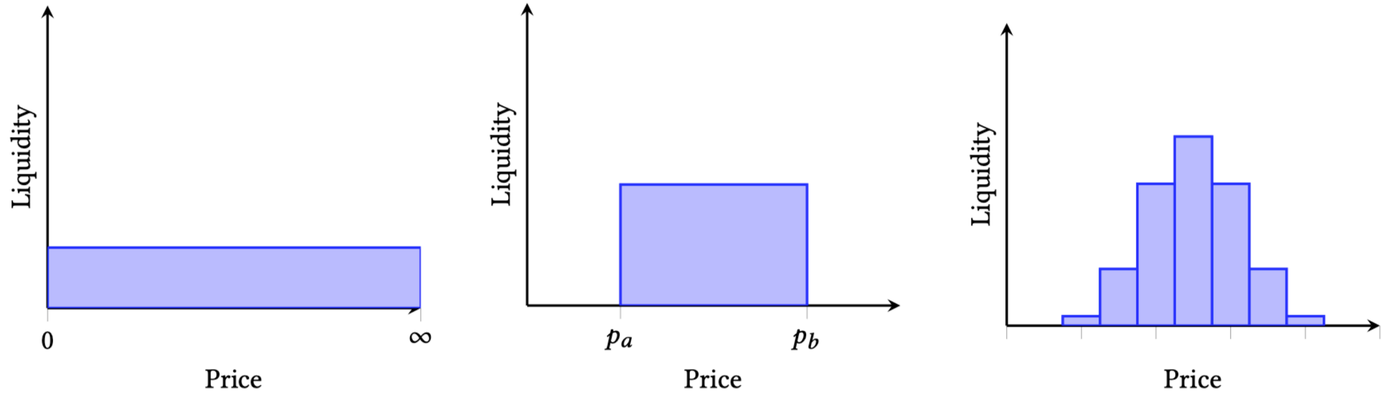
若已知L和sqrt(P)，就可以推导出资金使用量 x y

$$L = \frac{\Delta Y}{\Delta\sqrt{P}}, \Delta x = L * \Delta\frac{1}{\sqrt{P}}$$

流动性是价格变化所需的资金量

# Uniswap V3

- 流动性聚合




**(I)** Uniswap v2 **(II)** A single position on $[p_a, p_b]$ **(III)** A collection of custom positions

**Figure 3: Example Liquidity Distributions**

# Uniswap V3

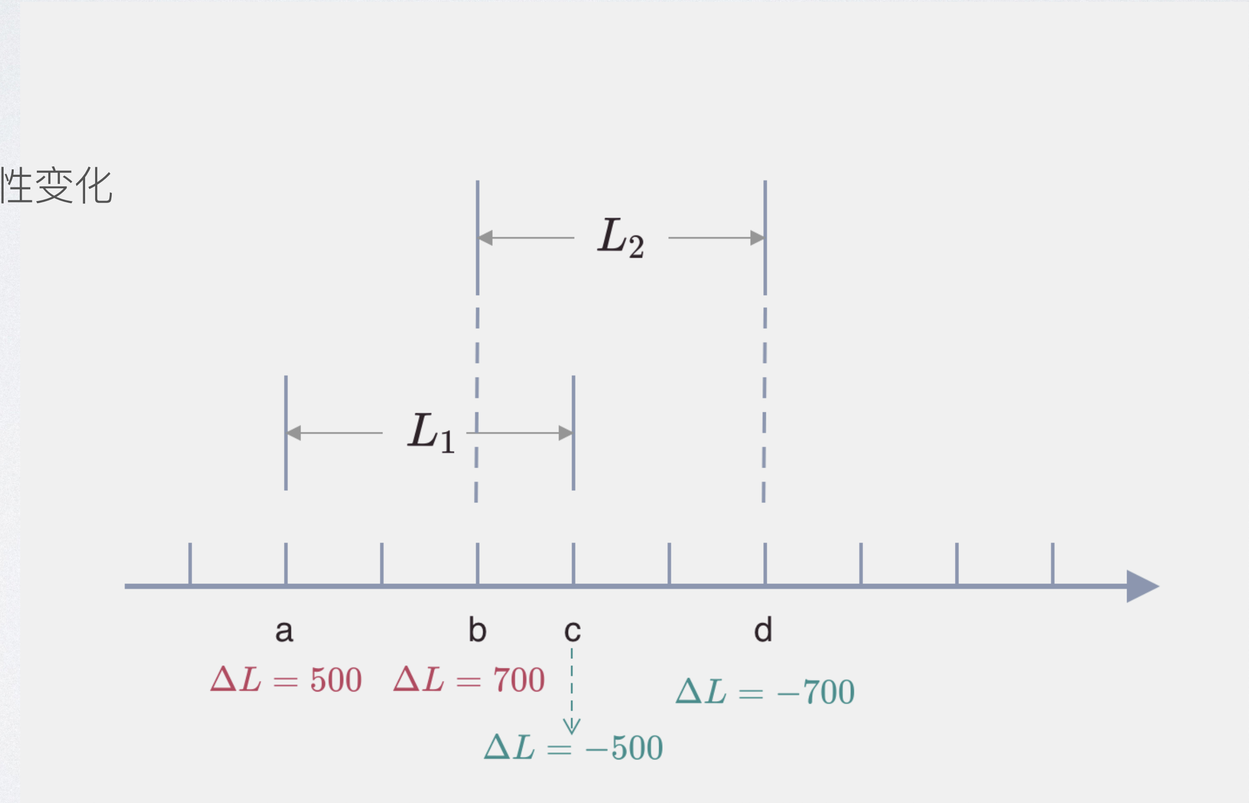
- 叠加流动性：引入Tick，使用等比数列将价格区间分段，添加流动性只能选择对应的离散点作为价格边界
- $Pi = 1.0001^{i}, \sqrt{P} = \sqrt{1.0001}^{i}, \sqrt{P_{i+1}} = \sqrt{1.0001} \cdot \sqrt{P_i}$
- 价格序号 i 称为 Tick（可为负数）
- 价格区间： $(\sqrt{1.0001}^{int24.min} - \sqrt{1.0001}^{int24.max})$
- tickspacing: 并不是每一个价格需要都可用，定义价格密度

| 费率 | tickspacing | 建议的使用范围 |
|---|---|---|
| 0.05% | 10 | 稳定币交易对 |
| 0.3% | 60 | 适用大多数交易对 |
| 1% | 200 | 波动极大的交易对 |

# Uniswap V3

- 管理流动性的变化量

跟踪中的流动性及每个 tick 引起的流动性变化



# Uniswap V3

- 兑换

$$L = \frac{\Delta Y}{\Delta\sqrt{P}}, \Delta x = L * \Delta\frac{1}{\sqrt{P}}$$

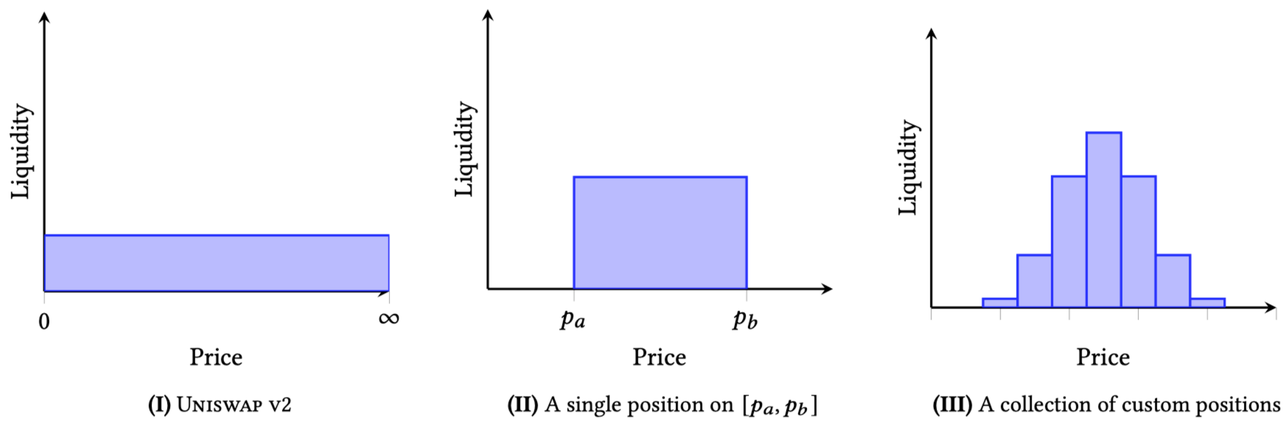
# Uniswap V3

- 手续费：多个不同LP如何计算各自的手续费?

若统一价格区间：

用户手续费： $fee = deltaL * (fee_{global}/liquidity_{global})$

交易手续费累加值更新： $feeGrowthGlobal += deltafee/liquidity$




(I) Uniswap v2 (II) A single position on $[p_a, p_b]$ (III) A collection of custom positions

**Figure 3: Example Liquidity Distributions**

# Uniswap V3

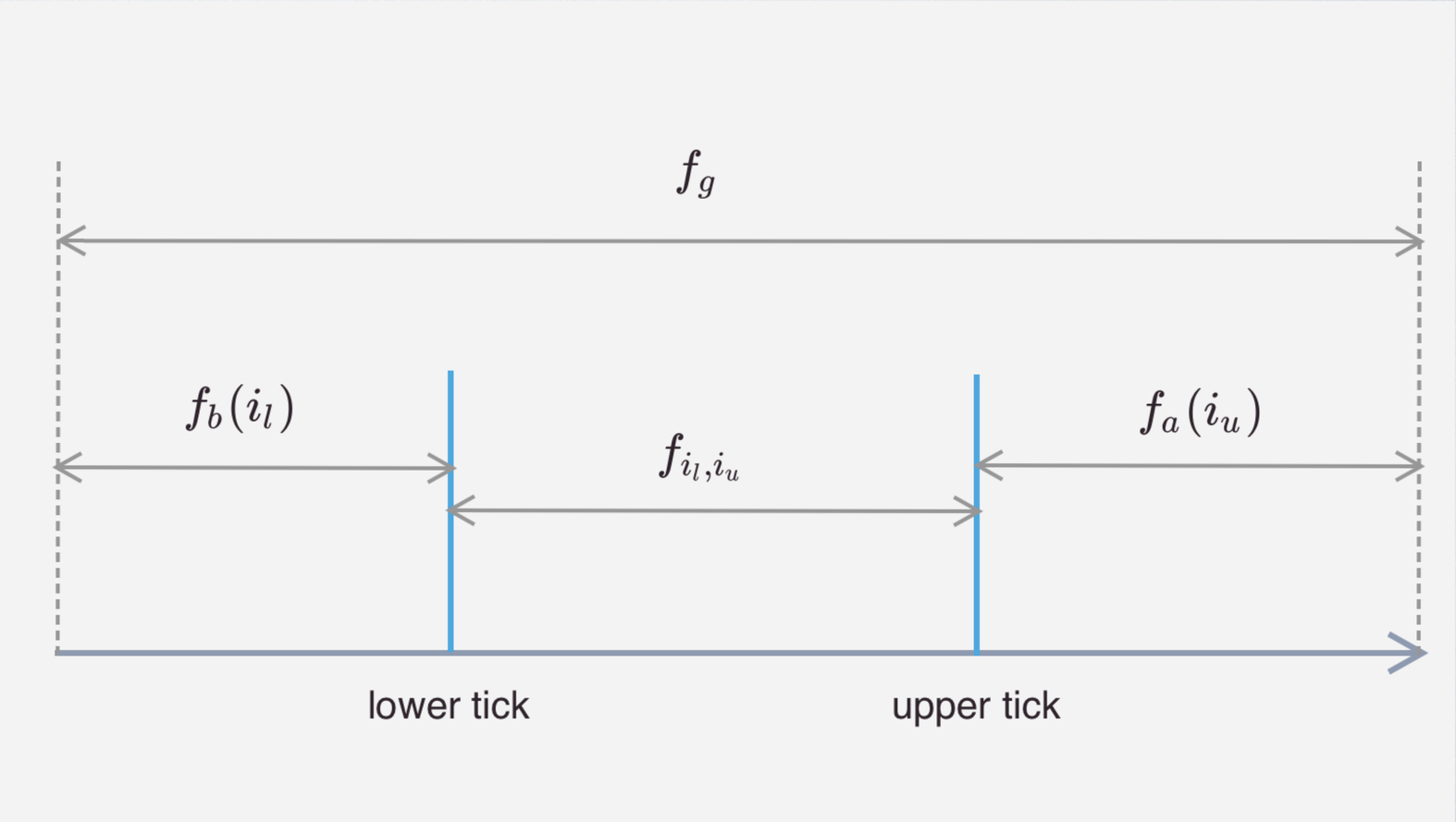
- 有多个区间时，如何获得某个区间的交易手续费累加。

$$f_{i_l,i_u} = f_g - f_b(i_l) - f_a(i_u)$$

*feeGrowthOutside*

在(i,u)区间发生的交易手续费累加，但是外侧的手续费是不变的，只需要累加 $f_g$

因此可以在tick上，加入 *feeGrowthOutside*，记录该 tick **外侧的手续费总量**



# Uniswap V3

- 初始化和更新外侧手续费： $feeGrowthOutside$

在加入流动性时，进行初始化（$i_c$ :表示当前价格 $i$:表示tick边界）：

当 $i_c >= i$ 时， 则：$feeGrowthOutside = feeGlobal$ ，假定所有手续费在该刻度之下生成
当 $i_c < i$ 时， 则：$feeGrowthOutside = 0$

在交易时，若价格穿过某个 tick，$feeGrowthOutside$ 需要翻转，因为价格外侧手续费永远要在当前价格的另一侧：

$$feeGrowthOutside = feeGlobal - feeGrowthOutside$$

手续费是单独保存的，需要自己去 collect

# Uniswap V3

- 预言机

V2 合约中记录了 price0CumulativeLast, price0CumulativeLast ， 但是需要外部更新保存，然后计算算术平均价格

V3 使用 跟踪价格的 tick（价格的对数值 $log_{1.0001}P$ ）累计和：$a_t = \sum_{i=1}^{t} log_{1.0001}P_i$

在使用时，求 tick 算术平均值：$\frac{\sum_{i=t_1}^{t_2} log_{1.0001}P_i}{t_2 - t_1}$ 可以转换为：

$$\frac{\sum_{i=t_1}^{t_2} log_{1.0001}P_i}{t_2 - t_1} = log_{1.0001}\left(\prod_{i=t_1}^{t_2} P_i\right)^{\frac{1}{t_2 - t_1}}$$

（几何平均值其实是其对数域上的算数平均值）

$\left(\prod_{i=t_1}^{t_2} P_i\right)^{\frac{1}{t_2 - t_1}}$ 即在$[t_1, t_2]$时间段的几何平均值价格。

算术平均数 $A = \frac{A_1 + A_2 + \ldots + A_n}{n}$

几何平均数 $A = \sqrt[n]{A_1 \times A_2 \times \ldots \times A_n}$

# Uniswap V3

- 预言机

V2 合约中记录了 price0CumulativeLast, price0CumulativeLast ， 但是需要外部更新保存，然后计算算术平均价格

V3 使用 跟踪价格的 tick（价格的对数值 $log_{1.0001}P$ ）累计和：$a_t = \sum_{i=1}^{t} log_{1.0001}P_i$

在使用时，求 tick 算术平均值：$\frac{\sum_{i=t_1}^{t_2} log_{1.0001}P_i}{t_2 - t_1}$ 可以转换为：

$$\frac{\sum_{i=t_1}^{t_2} log_{1.0001}P_i}{t_2 - t_1} = log_{1.0001}\left(\prod_{i=t_1}^{t_2} P_i\right)^{\frac{1}{t_2 - t_1}}$$

（几何平均值其实是其对数域上的算数平均值）

$\left(\prod_{i=t_1}^{t_2} P_i\right)^{\frac{1}{t_2 - t_1}}$ 即在$[t_1, t_2]$时间段的几何平均值价格。

算术平均数 $A = \frac{A_1 + A_2 + \ldots + A_n}{n}$

几何平均数 $A = \sqrt[n]{A_1 \times A_2 \times \ldots \times A_n}$

# Uniswap V3

- 预言机

假设 $P_{t_1,t_2}$ 为 $[t_1, t_2]$区间的交易对价格的几何平均值，$a_1, a_2$ 分别为 $t_1, t_2$ 时间点 tick index 的时间加权累积值，那么$P_{t_1,t_2}$ 的计算为：

$$log_{1.0001}(P_{t_1,t_2}) = \frac{\sum_{i=t_1}^{t_2} log_{1.0001} P_i}{t_2 - t_1}$$

$$log_{1.0001}(P_{t_1,t_2}) = \frac{a_{t_2} - a_{t_1}}{t_2 - t_1}$$

$$P_{t_1,t_2} = 1.0001^{\frac{a_{t_2} - a_{t_1}}{t_2 - t_1}}$$

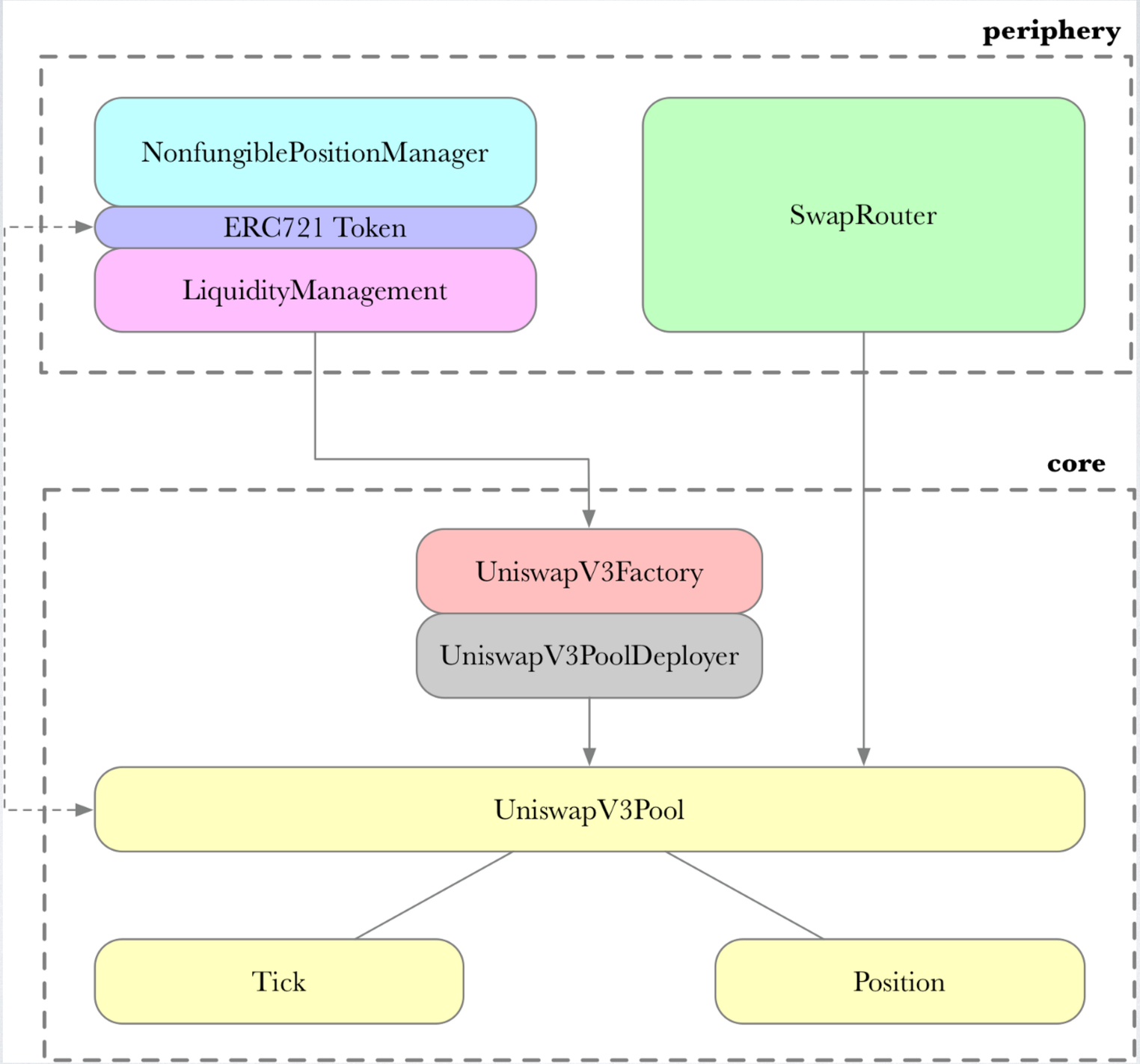
# Uniswap V3

- 预言机

为节省交易 gas， 默认V3 只保留最近一个 tick 累积值，由Oracle 有需求的开发者来扩展存储的历史数量。

```
struct Observation {
    // 记录区块的时间戳
    uint32 blockTimestamp;
    // tick index 的时间加权累积值
    int56 tickCumulative;
    // 价格所在区间的流动性的时间加权累积值
    uint160 liquidityCumulative;
    // 是否已经被初始化
    bool initialized;
}
```

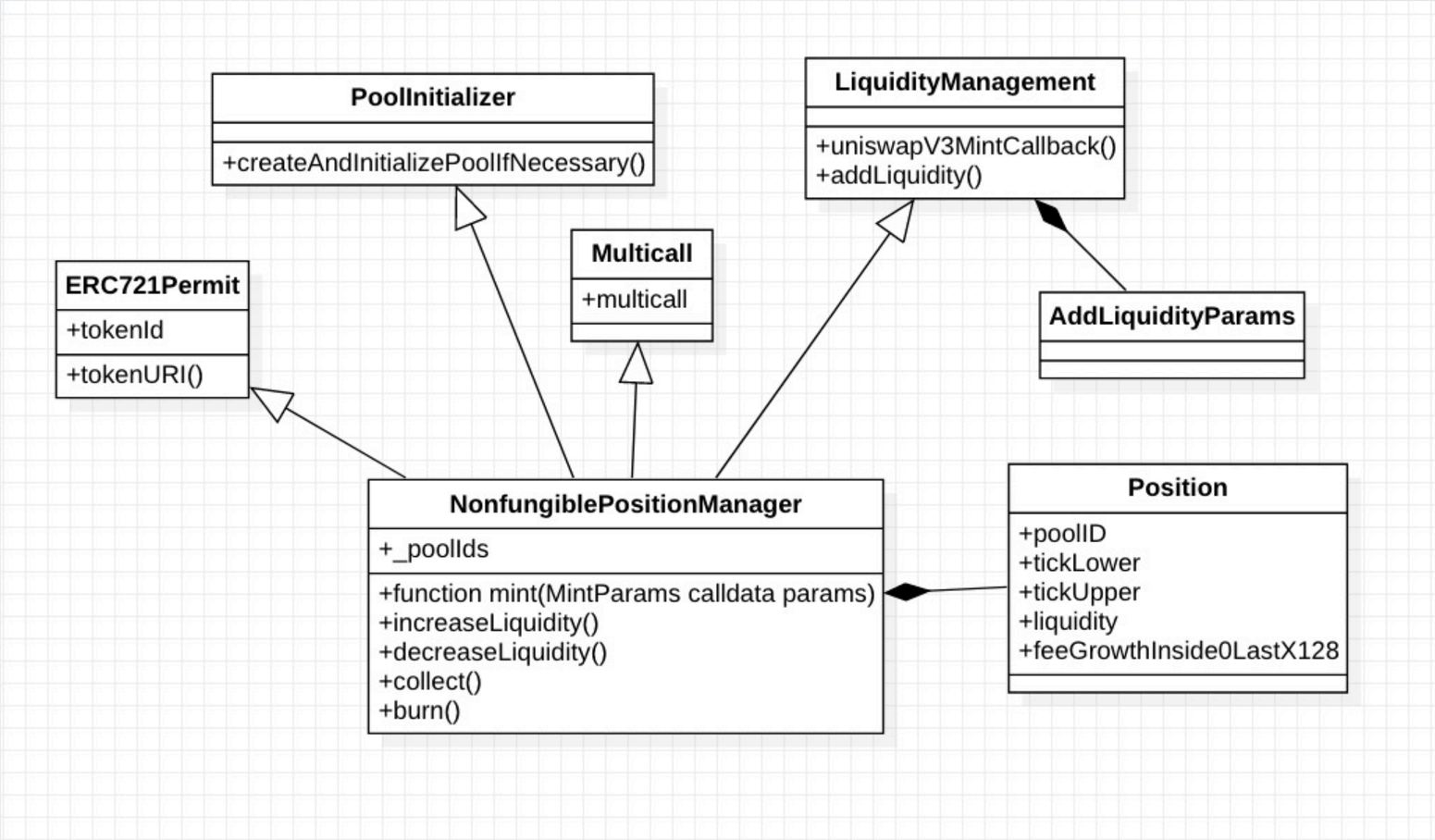
第5周


periphery
NonfungiblePositionManager
ERC721 Token
LiquidityManagement
SwapRouter
core
UniswapV3Factory
UniswapV3PoolDeployer
UniswapV3Pool
Tick
Position

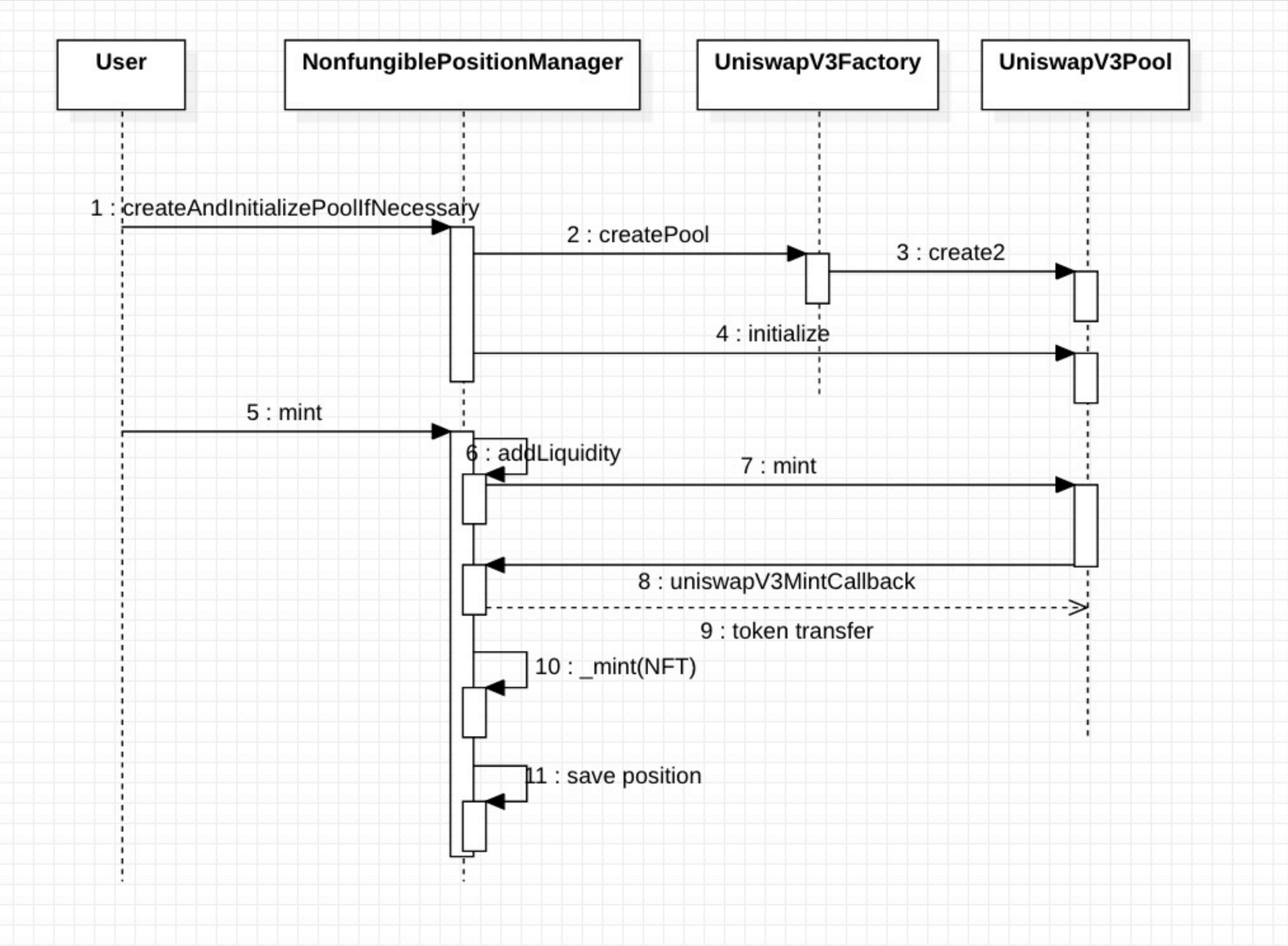
# Uniswap V3

- NonfungiblePositionManager：
  - positions：获取仓位信息
  - createAndInitializePoolIfNecessary： 创建 Pool 池
  - mint： 新增流动性（发行NFT）
  - increaseLiquidity、decreaseLiquidity: 添加、减少流动性
  - collet： 收集手续费
  - burn： 删除流动性
- SwapRouter
  - exactInputSingle、exactOutputSingle : 单个交易对兑换
  - exactInput、exactOutput : 可设置 path，对多个交易对兑换
  - uniswapV3SwapCallback：swap 时进行回调，支付真正的 token

# Uniswap V3

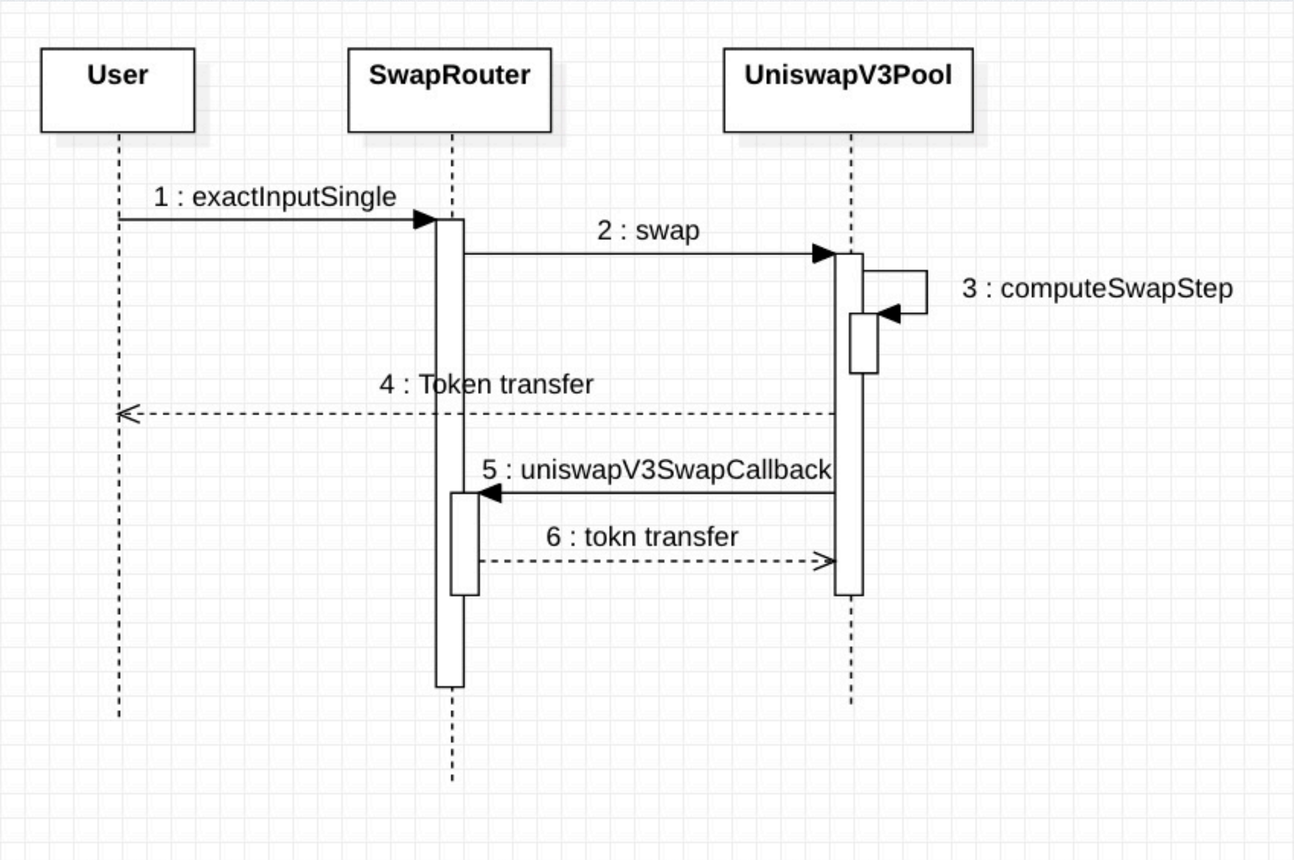

PoolInitializer
+createAndInitializePoolIfNecessary()
LiquidityManagement
+uniswapV3MintCallback()
+addLiquidity()
Multicall
+multicall
ERC721Permit
+tokenId
+tokenURI()
AddLiquidityParams
NonfungiblePositionManager
+_poolIds
+function mint(MintParams calldata params)
+increaseLiquidity()
+decreaseLiquidity()
+collect()
+burn()
Position
+poolID
+tickLower
+tickUpper
+liquidity
+feeGrowthInside0LastX128

- 添加流动性


User
NonfungiblePositionManager
UniswapV3Factory
UniswapV3Pool
1 : createAndInitializePoolIfNecessary
2 : createPool
3 : create2
4 : initialize
5 : mint
6 : addLiquidity
7 : mint
8 : uniswapV3MintCallback
9 : token transfer
10 : _mint(NFT)
11 : save position

# Uniswap V3

- 兑换



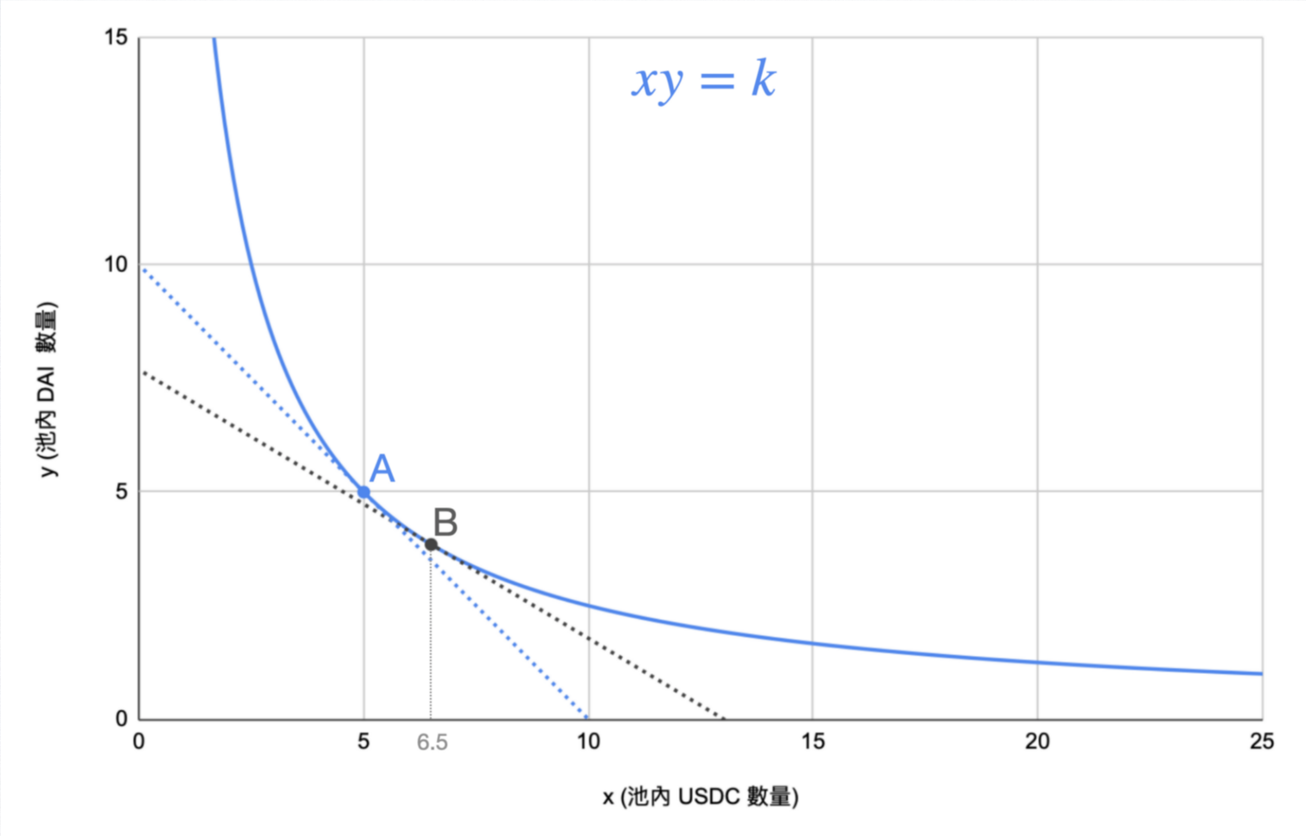
# Uniswap V3 参考文章

- 白皮书：https://uniswap.org/whitepaper-v3.pdf
- 中文版白皮书：https://f00wcc7jp0.feishu.cn/file/boxcnKWqffRmWslUkik6jlfqZ7b
- Uniswap V3 文档：https://docs.uniswap.org/protocol/reference/smart-contracts
- 大佬的 Uniswap v3 分析：https://liaoph.com/tags/#Uniswap
- Uniswap v3 b 站： https://www.bilibili.com/video/BV1go4y1X7Kx
- 合约代码结构：https://j1mmy.fi/uniswapv3/pool
- 函数分析：https://www.desmos.com/calculator/oiv0rti0ss?lang=zh-CN

# Curve 协议

- 稳定币兑换可以使用 xy=k?

要价格波动小（2%以内），
一次交易的数量需要小于池
内的 1%

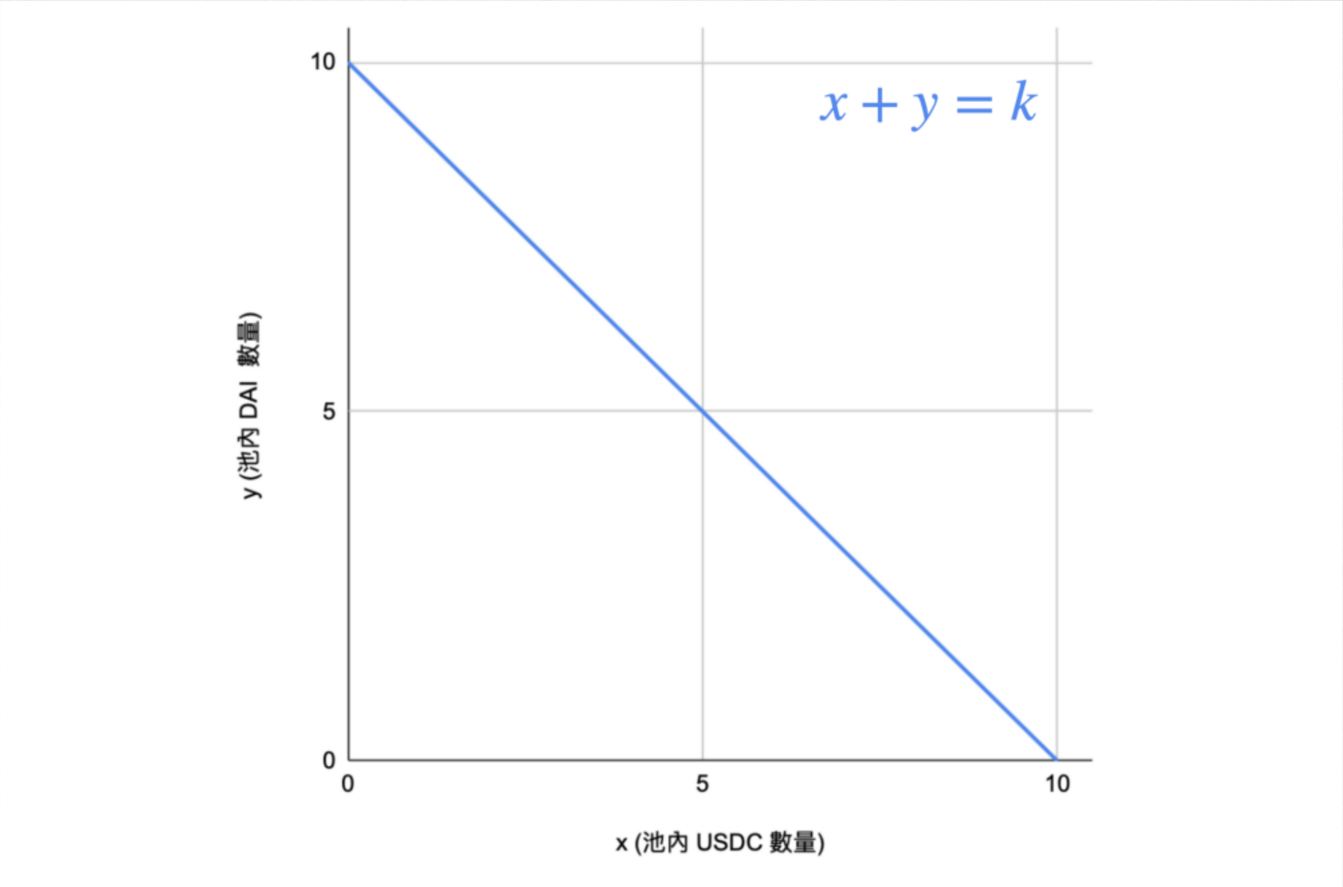


# Curve 协议

- 常量和模型 x + y=k

  优点：永远1:1兑换，没有滑点：
  ΔX = ΔY

  缺点：价格不应供需改变，出现以外部价差时，出现流动性枯竭。



# Curve 协议

- 稳定币兑换

恒定乘积：双曲线，两端将接近无穷

恒定总和: 直线，0 滑点、流动性枯竭

$$x\,y = (\frac{D}{2})^2 \times \beta$$

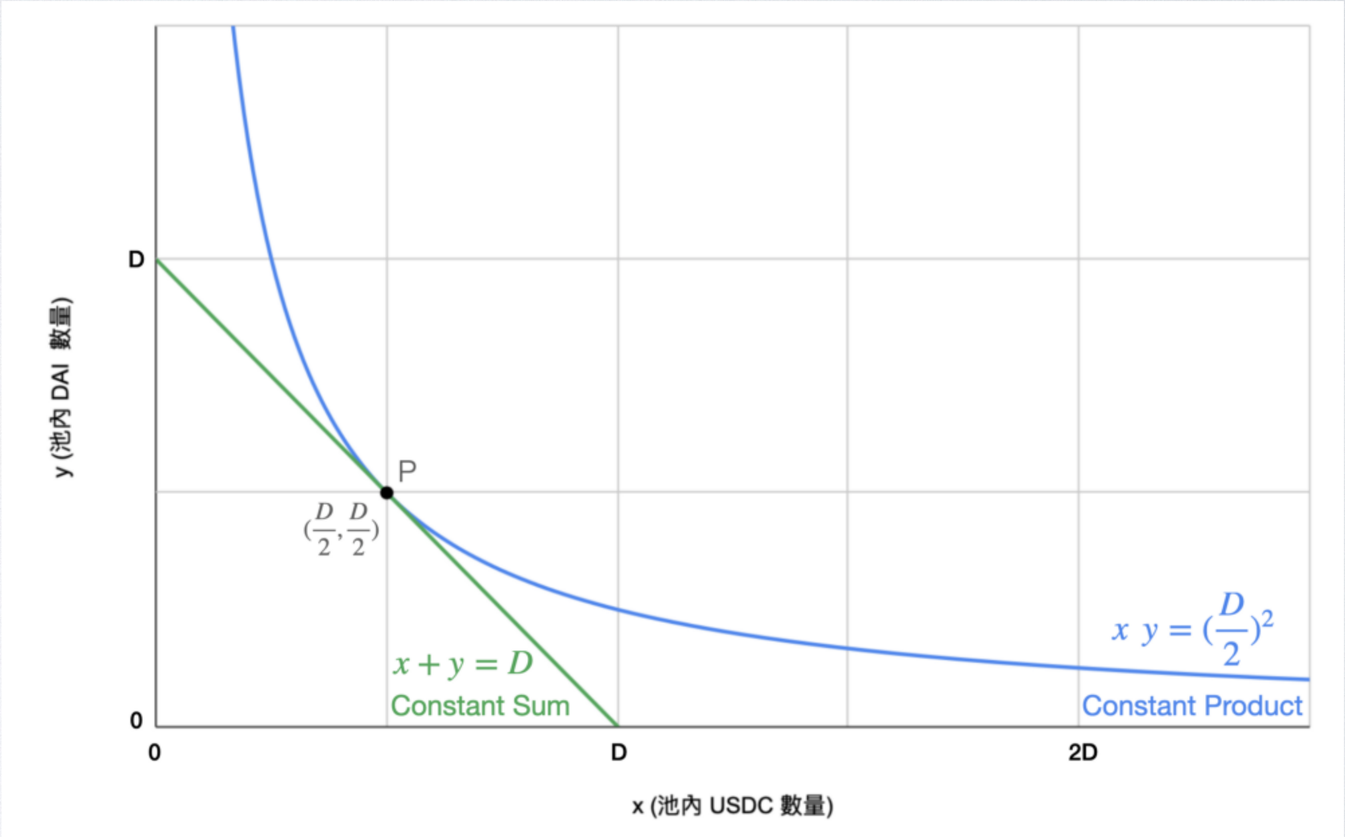
[数量]²

$$x + y = D \times \alpha D$$

[数量] [数量]

理想模型于两者之间，简单方式加权平均：

$$\alpha D(x+y) + \beta(xy) = \alpha D D + \beta(\frac{D}{2})^2$$

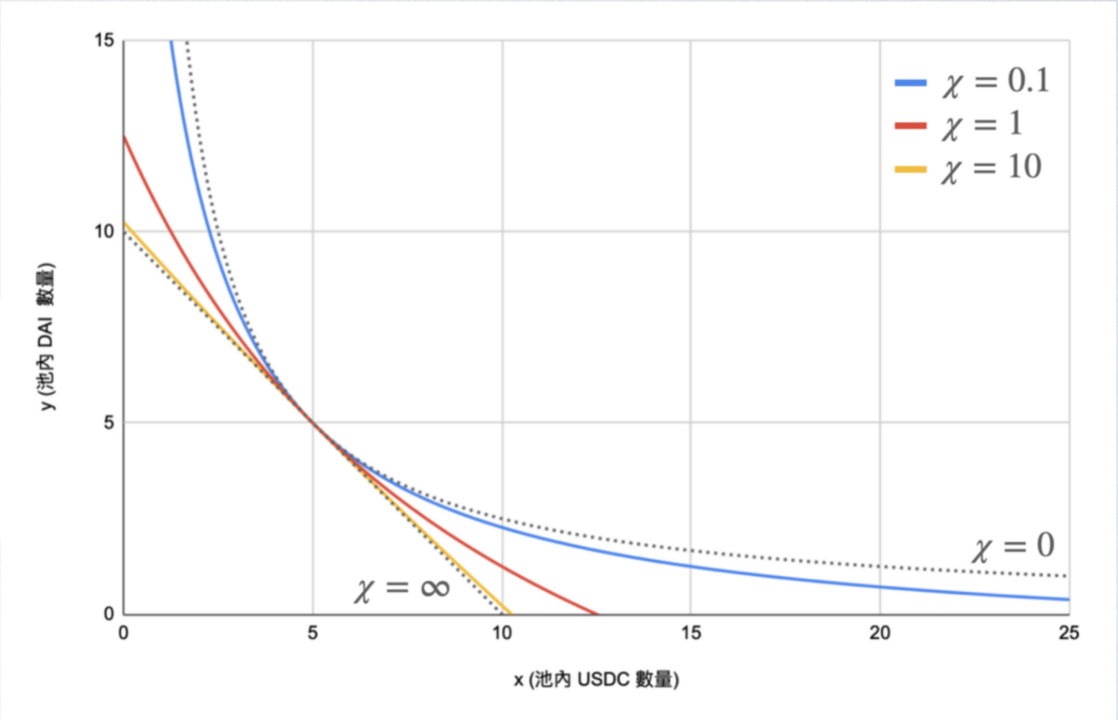


# Curve 协议

- 稳定币兑换

$$\alpha D(x+y)+\beta(xy)=\alpha DD+\beta(\frac{D}{2})^2$$

$$\chi D(x+y)+(xy)=\chi D^2+(\frac{D}{2})^2 \quad ,\chi=\frac{\alpha}{\beta}$$




$\chi = 0$，方程退化为常量乘积模型， 当$\chi = \infty$，方程退化为常量和模型

可以通过设置 $\chi$ 来设计一条曲线

# Curve 协议

- 引入动态参数

模型仍有一个缺陷：若希望价格越稳定，则曲线越接近直线，此時会面临流动性枯竭问题。能否在靠近平衡点时，$\chi$ 越大，远离平衡点时越小。

$$\chi = A\frac{xy}{(\frac{D}{2})^2}$$

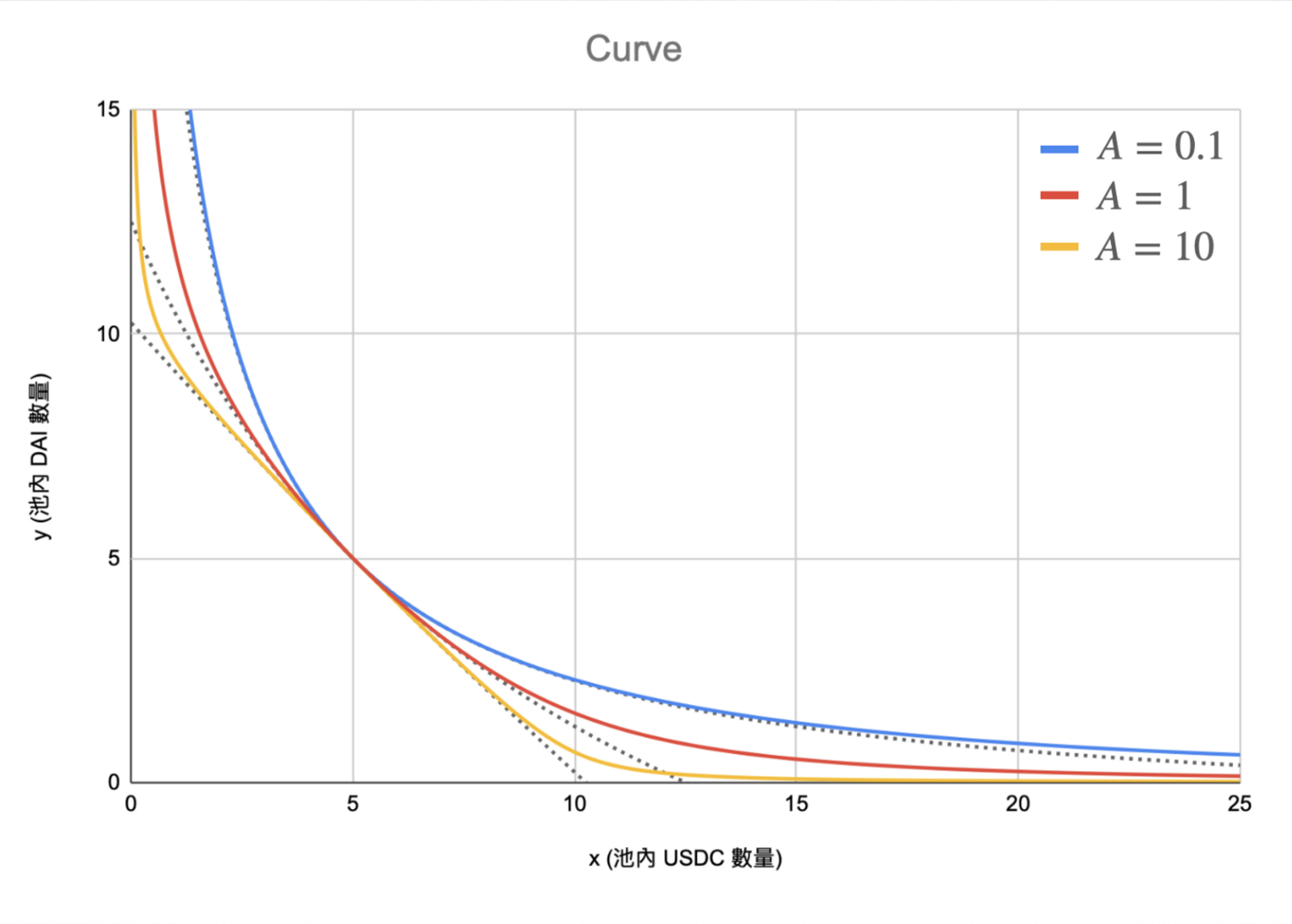
A 为放大系数，决定平衡点附件的曲率，A 越大滑点越小。**蓝色部分为动态参数，在（0,1）之间：**
离平衡点越接近時，接近1，接近常量和模型。 离平衡点越远時，接近0，接近常量乘积模型。

$$A\frac{xy}{(\frac{D}{2})^2}D(x+y)+(xy)=A\frac{xy}{(\frac{D}{2})^2}D^2+(\frac{D}{2})^2$$

$$A2^2\ (x+y)+D=AD2^2+\frac{D^3}{2^2\ xy}$$

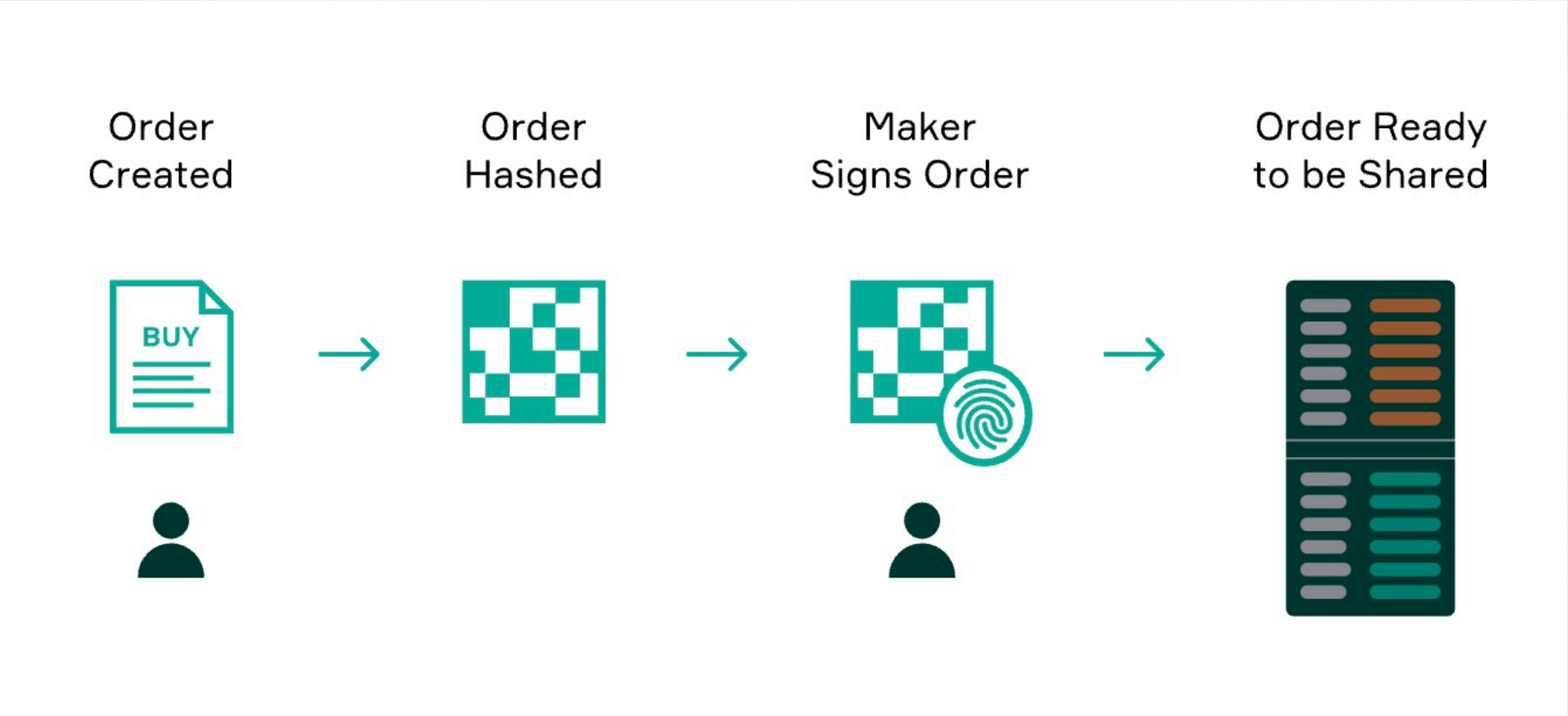
# Curve 协议

- 稳定币兑换



# 0x 协议

- 链下撮合，链上结算




https://0x.org/docs/core-concepts#digital-signature

# Q & A

# 练习题

- 以太坊测试网上部署两个自己的 ERC20 合约MyToken，分别在Uniswap V2、V3（网页上）添加流动性。
- 作业：编写合约执行闪电贷（参考 V2 的ExampleFlashSwap）：
  - uniswapV2Call中，用收到的 TokenA 在 Uniswap V3 的 SwapRouter 兑换为 TokenB 还回到uniswapV2 Pair 中。