从编程语言角度看智能合约

智能合约给编程语言带来的新变化

王渊命　STARCOIN DEVELOPER/MOVE CONTRIBUTOR

@jolestar http://jolestar.com

WESTAR Starcoin

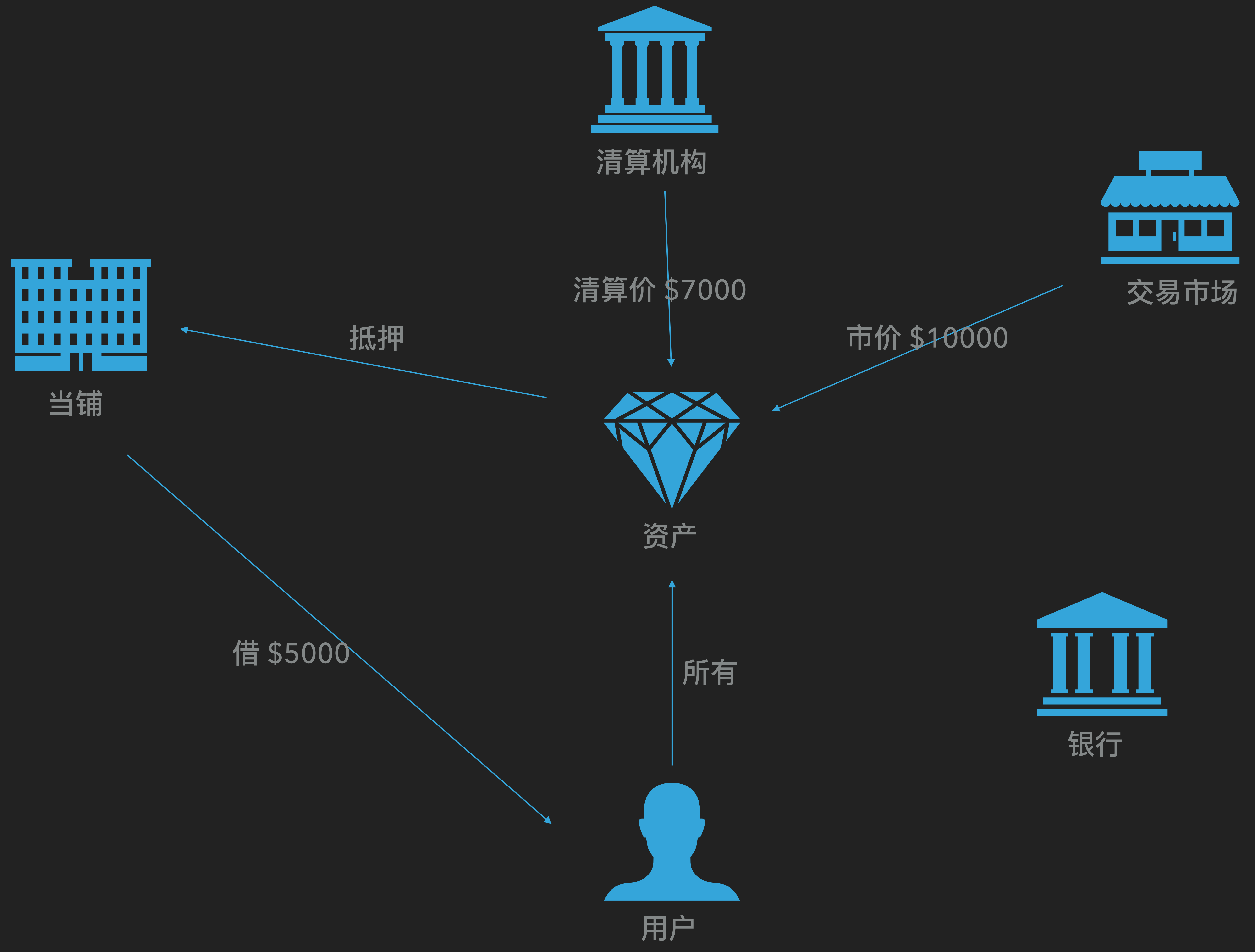
智能合约是什么

智能合约是什么

- 运行在链上的，由用户自定义的程序
- 通过链节点的重复校验以及共识机制，使其具有不依赖于权威方的独立约束力

- 智能合约到底是什么？http://jolestar.com/what-is-the-smart-contract/

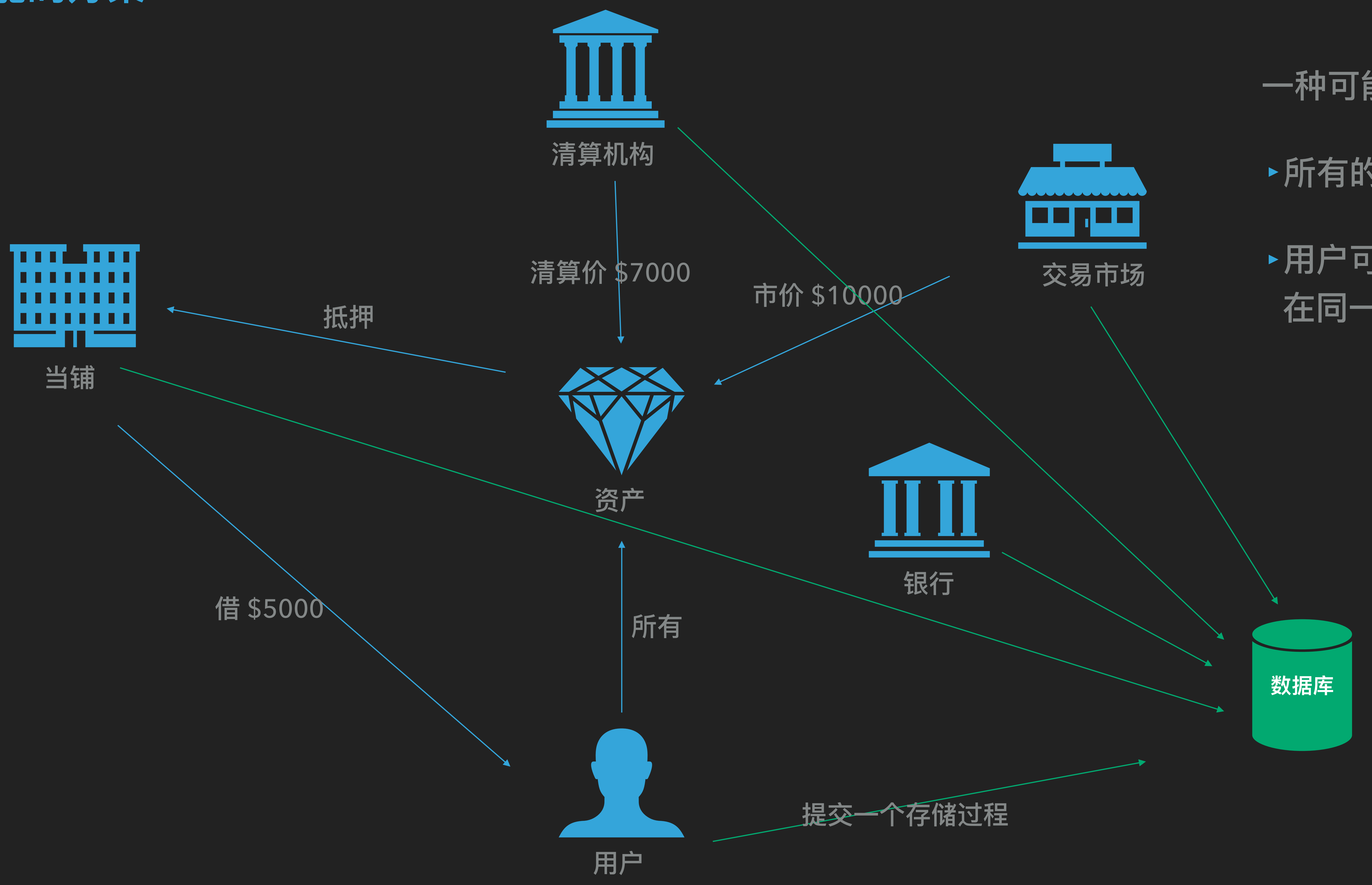
设想一个场景



有没有可能：

- 当用户无法偿还当铺的借款时，能否先将资产拿出来通过市场卖掉后再偿还当铺的钱？
- 当用户无法偿还当铺的借款时，能否先从银行无抵押贷款，偿还当铺的借款后，拿到资产，在市场中卖掉后偿还银行的贷款？

可能的方案



一种可能：

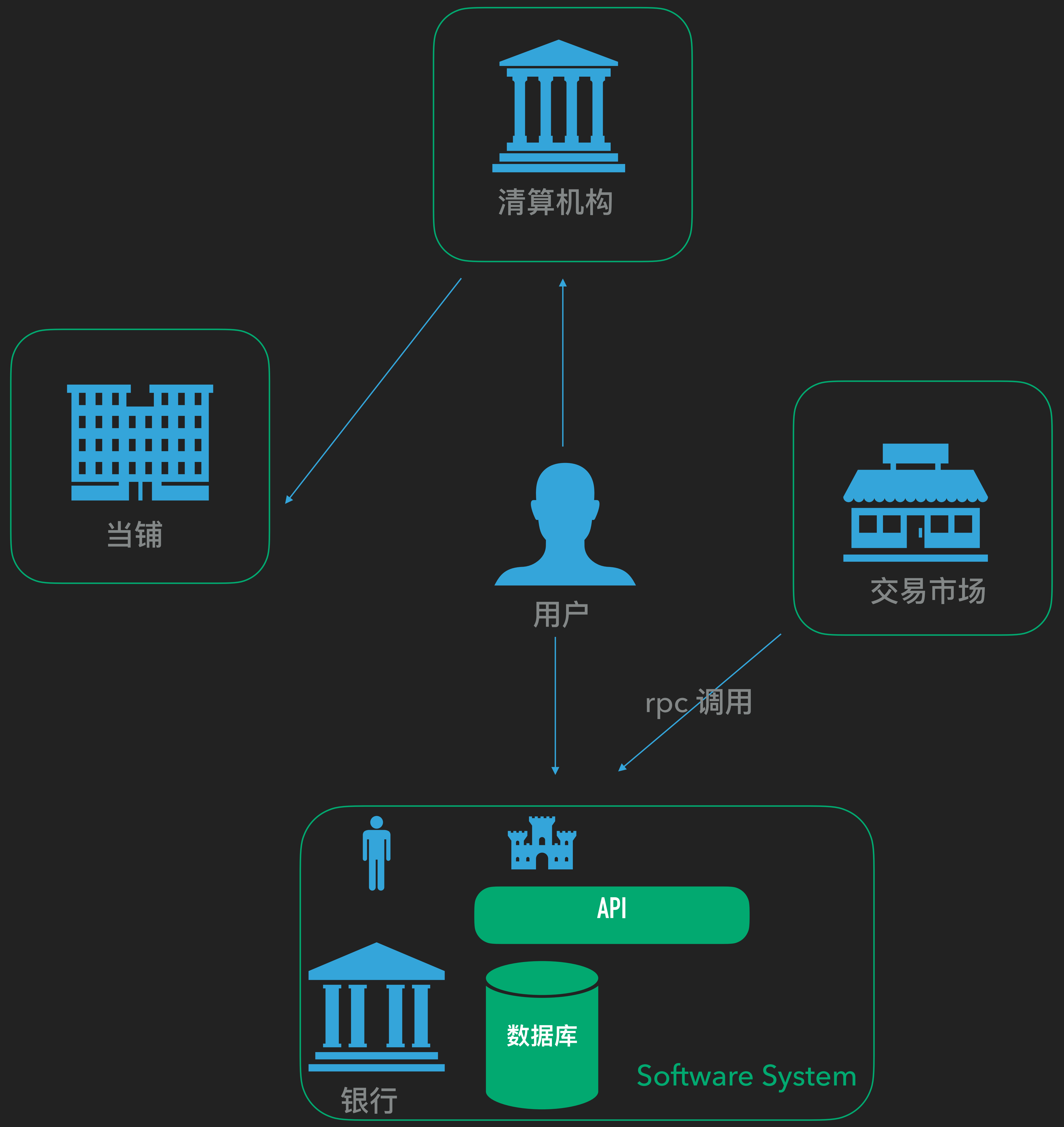
- 所有的机构共用一个数据库
- 用户可以编写存储过程操作数据库，在同一个交易中完成前面的操作

闪电贷（FLASH LOANS）

- 区块链上的无抵押贷款
- 借款和还款在同一个事务中发生，保证原子性执行

- 数据库 –> 区块链
- 存储过程 –> 智能合约

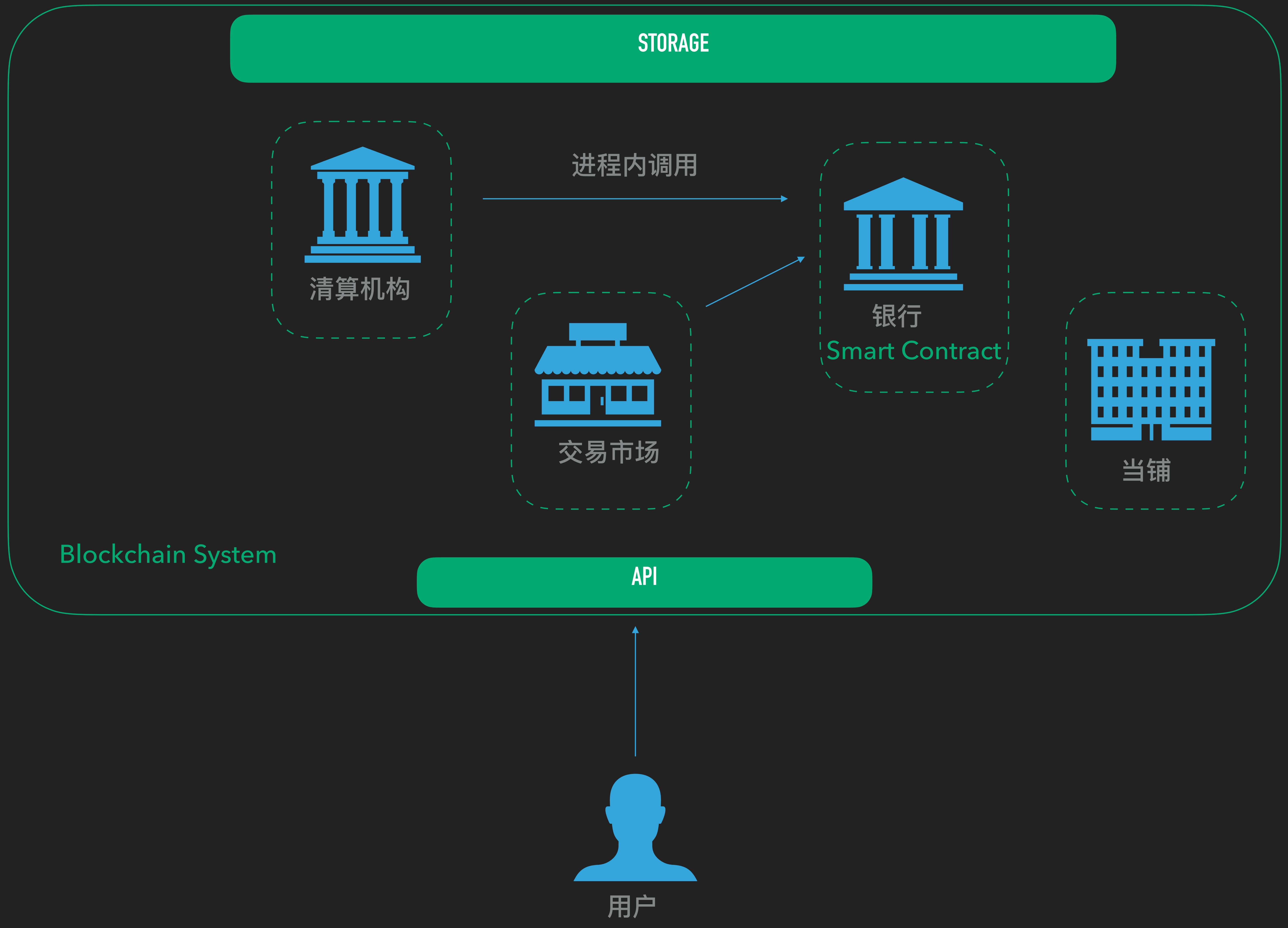
传统的系统



不同组织之间的软件系统

- 系统之间通过 RPC 调用
- 中间有人工的，技术的各种“墙”的隔离

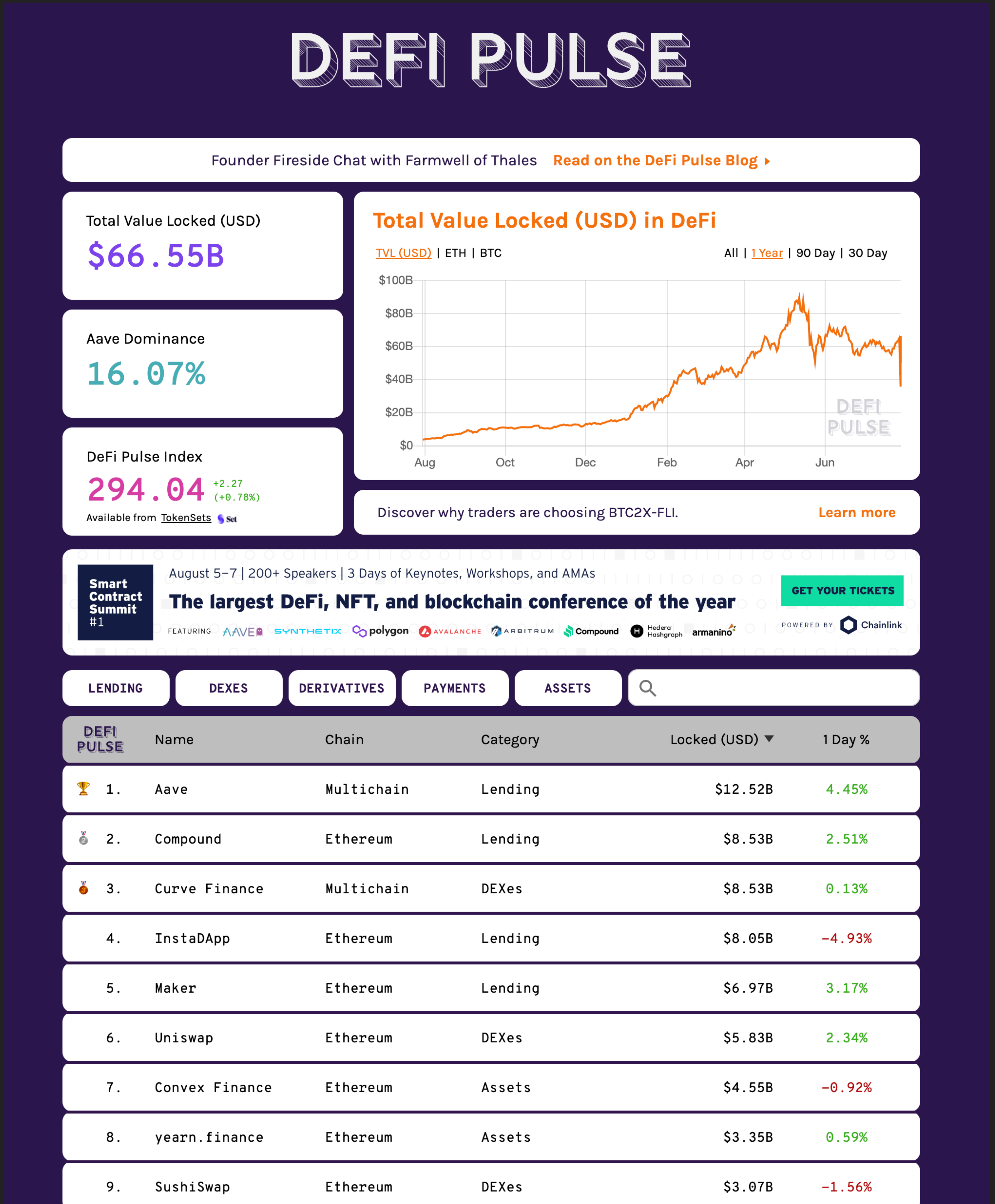
区块链系统



不同组织之间共享一套基础软件系统

- 系统之间通过 内部 调用
- 通过编程语言的安全机制隔离

智能合约还是个玩具吗?



DEFI PULSE
Founder Fireside Chat with Farmwell of Thales
Read on the DeFi Pulse Blog
Total Value Locked (USD)
$66.55B
Aave Dominance
16.07%
DeFi Pulse Index
294.04
+2.27 (+0.78%)
Available from TokenSets
Total Value Locked (USD) in DeFi
TVL (USD) | ETH | BTC
All | 1 Year | 90 Day | 30 Day
$100B
$80B
$60B
$40B
$20B
$0
Aug
Oct
Dec
Feb
Apr
Jun
DEFI PULSE
Discover why traders are choosing BTC2X-FLI.
Learn more
Smart Contract Summit #1
August 5–7 | 200+ Speakers | 3 Days of Keynotes, Workshops, and AMAs
The largest DeFi, NFT, and blockchain conference of the year
GET YOUR TICKETS
FEATURING
AAVE
SYNTHETIX
polygon
AVALANCHE
ARBITRUM
Compound
armanino
POWERED BY
Chainlink
LENDING
DEXES
DERIVATIVES
PAYMENTS
ASSETS
DEFI PULSE
Name
Chain
Category
Locked (USD)
1 Day %
1. Aave Multichain Lending $12.52B 4.45%
2. Compound Ethereum Lending $8.53B 2.51%
3. Curve Finance Multichain DEXes $8.53B 0.13%
4. InstaDApp Ethereum Lending $8.05B -4.93%
5. Maker Ethereum Lending $6.97B 3.17%
6. Uniswap Ethereum DEXes $5.83B 2.34%
7. Convex Finance Ethereum Assets $4.55B -0.92%
8. yearn.finance Ethereum Assets $3.35B 0.59%
9. SushiSwap Ethereum DEXes $3.07B -1.56%

三个角度

- 合约的状态机制
- 编程语言的特性
- 合约之间的依赖与调用

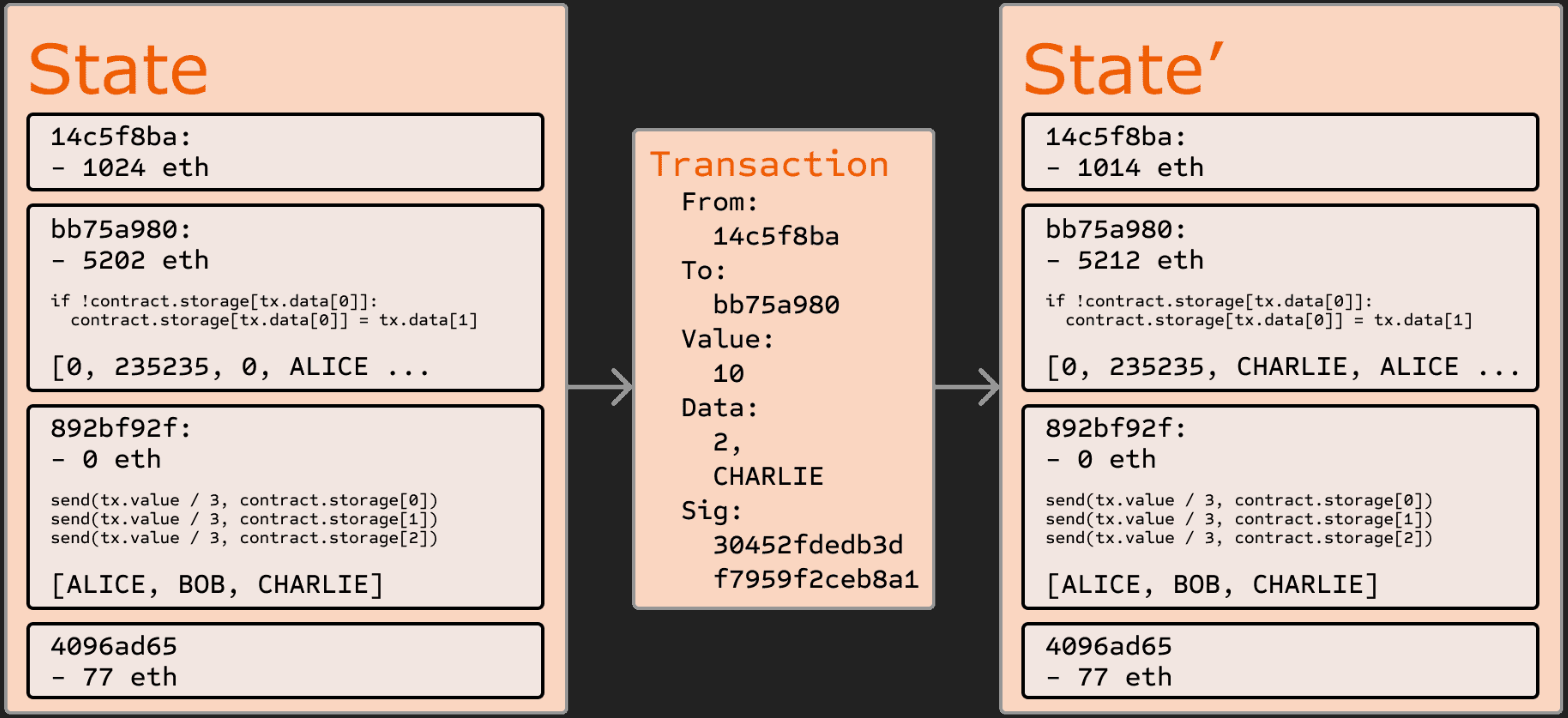
应用程序的状态

- 序列化/文件系统，如: Java Object Serialization
- Database/ORM/Dao
- Network/RPC/Remote Storage

有没有可能设计一种通用的状态持久化机制?

智能合约的执行和状态

$$\sigma t+1 \equiv Y(\sigma t, T)$$



合约的状态机制 ETHEREUM

```
// This is a smart contract - a program that can be deployed to the Ethereum blockchain.
contract SimpleToken {
    // An address is comparable to an email address - it's used to identify an account on Ethereum.
    address public owner;
    uint256 public constant token_supply = 1000000000000;

    // A mapping is essentially a hash table data structure.
    // This mapping assigns an unsigned integer (the token balance) to an address (the token holder).
    mapping (address => uint) public balances;


  // When 'SimpleToken' contract is deployed:
  // 1. set the deploying address as the owner of the contract
  // 2. set the token balance of the owner to the total token supply
    constructor() {
        owner = msg.sender;
        balances[owner] = token_supply;
    }

    // Sends an amount of tokens from any caller to any address.
    function transfer(address receiver, uint amount) public {
        // The sender must have enough tokens to send
        require(amount <= balances[msg.sender], "Insufficient balance.");

        // Adjusts token balances of the two addresses
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
    }
}
```

Solidity

- 自动将 Contract 的属性映射到存储
- Slot based storage

合约的状态机制 MOVE

```
module MyCounter {
    use 0x1::Signer;

    struct Counter has key, store {
        value:u64,
    }
    public fun init(account: &signer){
        move_to(account, res: Counter{value:0});
    }
    public fun incr(account: &signer) acquires Counter {
        let counter = borrow_global_mut<Counter>( addr: Signer::address_of(account));
        counter.value = counter.value + 1;
    }

    public(script) fun init_counter(account: signer){
        Self::init(&account)
    }

    public(script) fun incr_counter(account: signer)  acquires Counter {
        Self::incr(&account)
    }
}
```

Move

- 提供状态操作原语 move_to/move_from/borrow_global
- 面向类型的 Key 设计

智能合约的状态

- 链托管了合约的状态，提供通用的持久化方案
- 不同的链有不同的状态持久化方案
- 思考：如何保证状态的安全?

- 可见性： public/private 等可见性代表的含义发生变化
- 基本类型：address/signer 编程语言对用户有了感知
- 类型系统

类型系统

- Ordered type: 每个变量都按照它被引入的顺序严格使用一次
- Linear type: 每个变量都被严格使用一次
- Affine type: 每个变量最多使用一次
- Relevant type: 每个变量至少被使用一次
- Normal type systems : 每个变量都可以被任意使用

MOVE ABILITY

- copy: 可以被复制
- drop: 可以被丢弃
- store: 可以保存到全局存储中
- key: 可以作为全局存储的 key

MOVE ABILITY

- No Drop/Copy: linear types
- Copy + drop: normal language struct
- Drop: affine types
- Copy: Cloneable capabilities

```
module Token {

    /// The token has a `TokenType` color that tells us what token the
    /// `value` inside represents.
    struct Token<TokenType> has store {
        value: u128,
    }
    public fun mint<TokenType: store>(account: &signer, amount: u128): Token<TokenType>
    acquires TokenInfo, MintCapability {
        mint_with_capability(
            borrow_global<MintCapability<TokenType>>( addr: Signer::address_of(account)),
            amount,
        )
    }
    public fun zero<TokenType: store>(): Token<TokenType> {
        Token<TokenType> { value: 0 }
    }
    public fun withdraw<TokenType: store>(
        token: &mut Token<TokenType>,
        value: u128,
    ): Token<TokenType> {
        // Check that `value` is less than the token's value
        assert(token.value >= value, Errors::limit_exceeded(EAMOUNT_EXCEEDS_COIN_VALUE));
        token.value = token.value - value;
        Token { value: value }
    }
    public fun deposit<TokenType: store>(token: &mut Token<TokenType>, check: Token<TokenType>) {
        let Token { value } = check;
        token.value = token.value + value;
    }
    public fun destroy_zero<TokenType: store>(token: Token<TokenType>) {
        let Token { value } = token;
        assert(value == 0, Errors::invalid_state(EDESTROY_TOKEN_NON_ZERO))
    }
```

编程语言的特性

- 区块链的状态机制让原有的编程语言的特性会发挥出更大的作用
  - 可见性：安全边界
  - 类型系统：映射真实世界

合约间的调用

- 依赖关系：不仅是工具依赖，同时也是服务依赖，应用的组合能力
- 静态调用 OR 动态调用
- 安全与风险

Blockchain	Protocol	Attack Time	Loss (USD)	Note
Ethereum	bZx	02/14/20	355k	
	bZx	02/18/20	635k	
	Balancer Pool	06/29/20	500k	
	MakerDAO	10/26/20	0	Governance manipulation
	Harvest Finance	10/26/20	34M	
	Akropolis	11/12/20	2M	
	Value DeFi	11/14/20	7.4M	
	Cheese Bank	11/16/20	3.3M	
	Origin Dollar	11/17/20	7M	
	Warp Finance	12/17/20	8M	
	Yearn Finance	02/05/21	11M	
	Alpha Finance	02/13/21	38M	
	DODO Dex	03/08/21	3.8M	
	xToken	05/12/21	24.5M	
BSC	Spartan Protocol	05/02/21	30M	
	bEarnFi	05/17/21	11M	
	PancakeBunny	05/19/21	45M	
	AutoShark	05/24/21	822.8k	
	JulSwap	05/27/21	(Undisclosed)	
	BurgerSwap	05/27/21	7.2M	
	Belt Finance	05/29/21	6.3M	
	Bogged Finance	05/23/21	3M	
Total			243.8M	

Flash loan attack diary

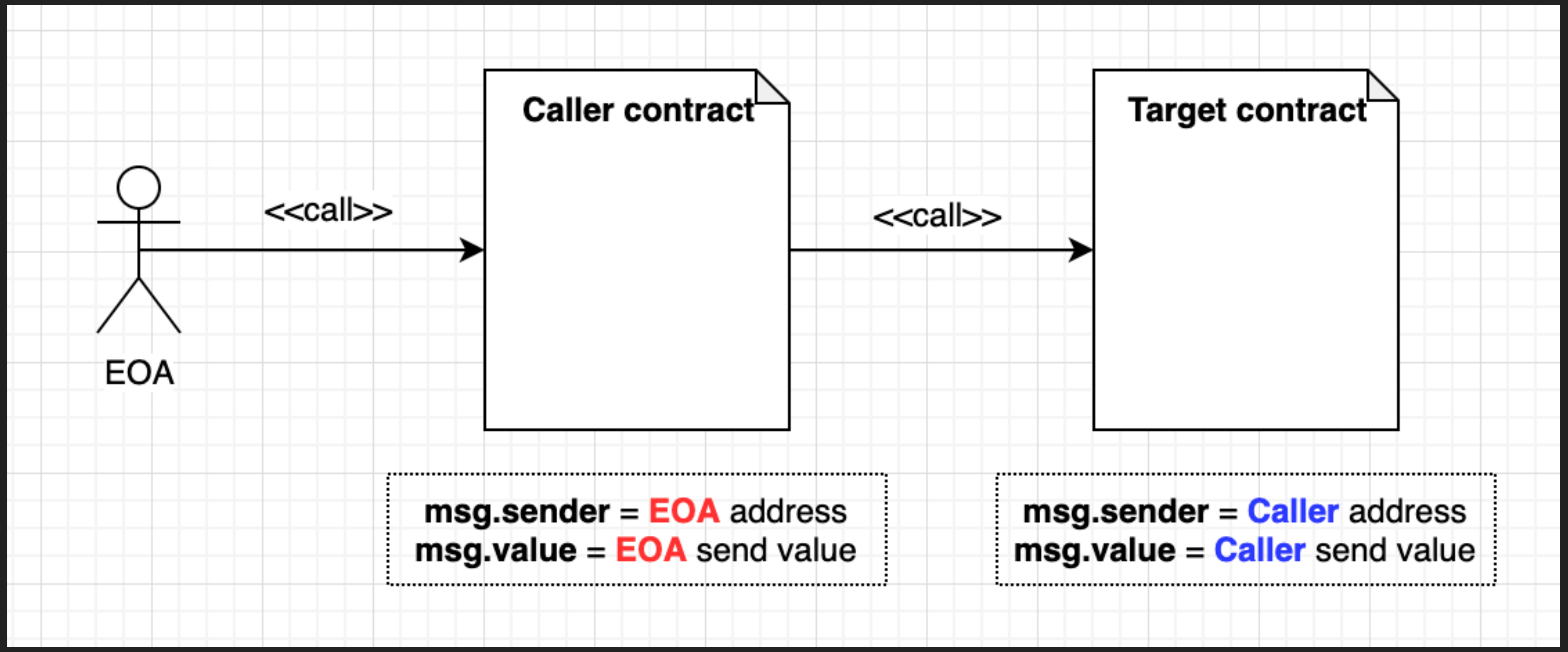
https://www.binance.org/en/blog/flash-loan-attacks-the-plague-of-defi/

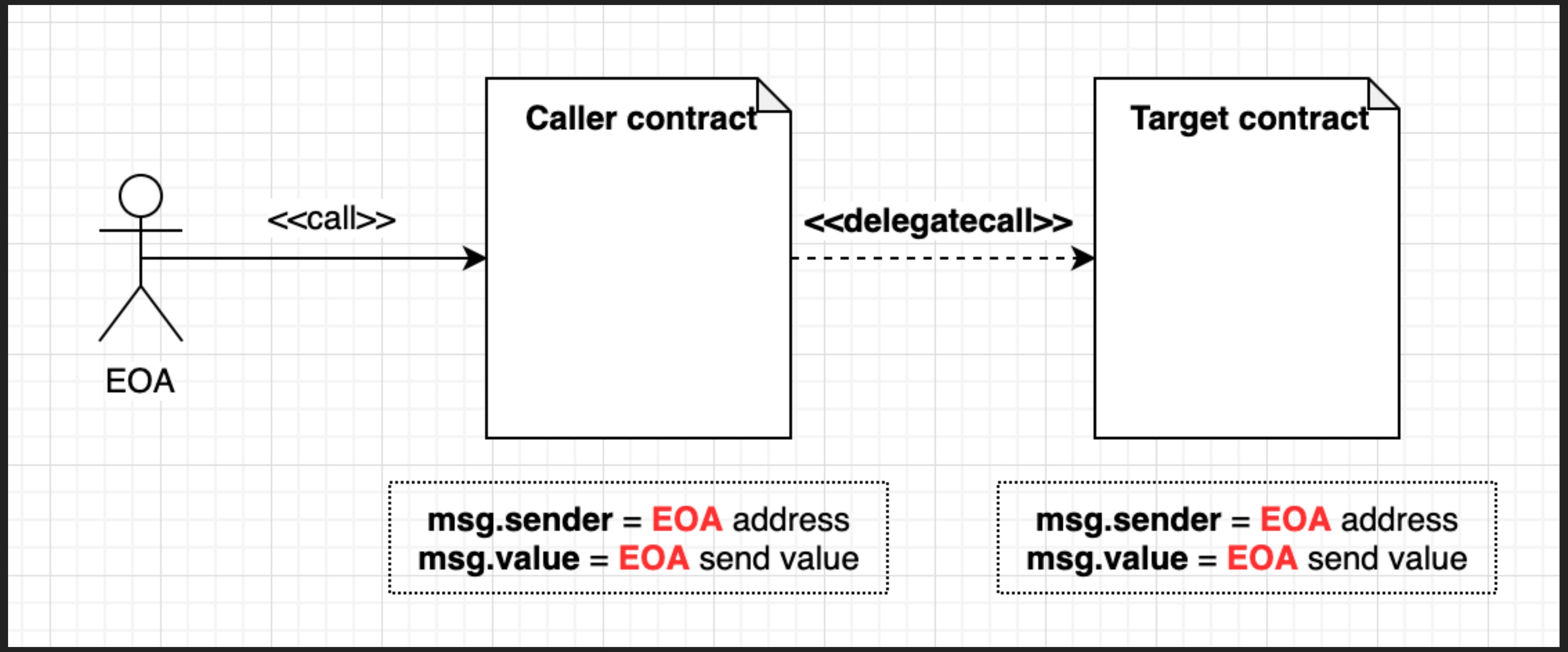
```solidity
/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, 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 sender, address recipient, 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);
}
```

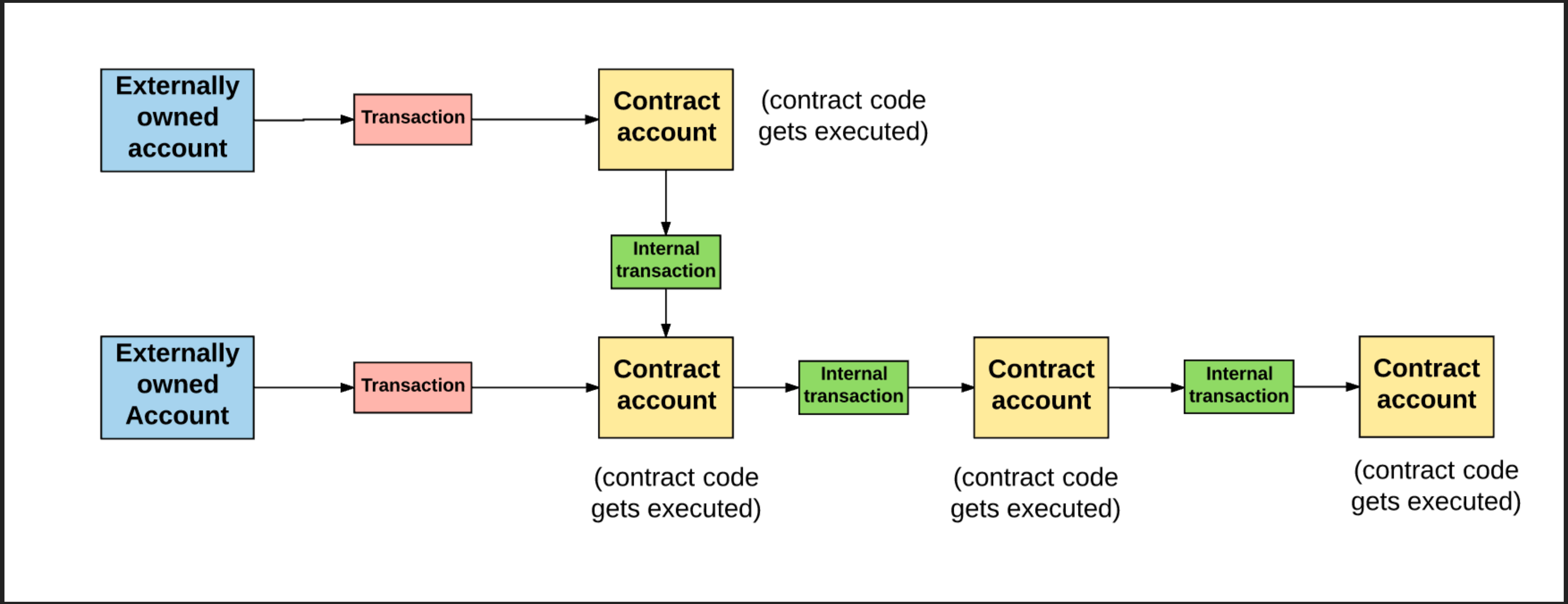
Solidity

- Interface as service

合约间的调用 ETHEREUM







合约间的调用 MOVE

```
module Treasury {
    use 0x1::Token;
    struct Treasury<TokenT> has store,key {
        balance: Token<TokenT>,
    }
    public fun balance<TokenT:store>(): u128 acquires Treasury{
        let token_issuer = Token::token_address<TokenT>();
        if(!exists<Treasury<TokenT>>( addr: token_issuer)){
            return 0
        };
        let treasury = borrow_global<Treasury<TokenT>>( addr: token_issuer);
        Token::value(&treasury.balance)
    }
    public fun deposit<TokenT:store>(token: Token<TokenT>) acquires Treasury{
        let token_address = Token::token_address<TokenT>();
        let treasury = borrow_global_mut<Treasury<TokenT>>( addr: token_address);
        let amount = Token::value(&token);
        Token::deposit(&mut treasury.balance, token);
    }
    fun do_withdraw<TokenT:store>(amount: u128): Token<TokenT> acquires Treasury {
        let token_address = Token::token_address<TokenT>();
        let treasury = borrow_global_mut<Treasury<TokenT>>( addr: token_address);
        Token::withdraw(&mut treasury.balance, amount)
    }
    public fun withdraw_with_capability<TokenT:store>(_cap: &mut WithdrawCapability<TokenT>, amount: u128): Token<TokenT> acquires Treasury {
        let token = do_withdraw(amount);
        token
    }
    public fun withdraw<TokenT:store>(signer: &signer, amount: u128) : Token<TokenT> acquires Treasury, WithdrawCapability{
        let cap = borrow_global_mut<WithdrawCapability<TokenT>>( addr: Signer::address_of(signer));
        Self::withdraw_with_capability( _cap: cap, amount)
    }
```

- 静态调用
- 类型在合约间共享

看一个闪电贷的例子

ETHEREUM 上的闪电贷 ERC3156

```solidity
interface IERC3156FlashLender {

    /**
     * @dev The amount of currency available to be lended.
     * @param token The loan currency.
     * @return The amount of `token` that can be borrowed.
     */
    function maxFlashLoan(
        address token
    ) external view returns (uint256);

    /**
     * @dev The fee to be charged for a given loan.
     * @param token The loan currency.
     * @param amount The amount of tokens lent.
     * @return The amount of `token` to be charged for the loan, on top of the returned principal.
     */
    function flashFee(
        address token,
        uint256 amount
    ) external view returns (uint256);

    /**
     * @dev Initiate a flash loan.
     * @param receiver The receiver of the tokens in the loan, and the receiver of the callback.
     * @param token The loan currency.
     * @param amount The amount of tokens lent.
     * @param data Arbitrary data structure, intended to contain user-defined parameters.
     */
    function flashLoan(
        IERC3156FlashBorrower receiver,
        address token,
        uint256 amount,
        bytes calldata data
    ) external returns (bool);
}
```

```solidity
interface IERC3156FlashBorrower {

    /**
     * @dev Receive a flash loan.
     * @param initiator The initiator of the loan.
     * @param token The loan currency.
     * @param amount The amount of tokens lent.
     * @param fee The additional amount of tokens to repay.
     * @param data Arbitrary data structure, intended to contain user-defined parameters.
     * @return The keccak256 hash of "ERC3156FlashBorrower.onFlashLoan"
     */
    function onFlashLoan(
        address initiator,
        address token,
        uint256 amount,
        uint256 fee,
        bytes calldata data
    ) external returns (bytes32);
}
```

```solidity
function uniswapV2Call(address sender, uint amount, bytes calldata data) external override {
    // access control
    require(sender == address(this), "only this contract may initiate");

    // decode data
    (
        address origin,
        IERC3156FlashBorrower receiver,
        address token,
        bytes memory userData
    ) = abi.decode(data, (address, IERC3156FlashBorrower, address, bytes));

    uint256 fee = flashFee(token, amount);

    // send the borrowed amount to the receiver
    IERC20(token).transfer(address(receiver), amount);
    // do whatever the user wants
    require(
        receiver.onFlashLoan(origin, token, amount, fee, userData) == CALLBACK_SUCCESS,
        "Callback failed"
    );
    // retrieve the borrowed amount plus fee from the receiver and send it to the uniswap pair
    IERC20(token).transferFrom(address(receiver), msg.sender, amount.add(fee));
}
```

MOVE 上的闪电贷

```
module FlashLoan {
    use 0x1::Token::{Self, Token};
    use 0x1::Option::{Self, Option};
    use 0x1::Errors;

    struct FlashLoan<TokenT: store> {
        amount: u128,
        coins: Option<Token<TokenT>>,
    }

    public fun borrow<TokenT: store>(_amount:u128): FlashLoan<TokenT> {
        //shold take from pool
        FlashLoan{
            amount: 0,
            coins: Option::some<Token<TokenT>>(Token::zero<TokenT>())
        }
    }

    public fun take_coins<TokenT: store>(loan: &mut FlashLoan<TokenT>): Token<TokenT> {
        Option::extract<Token<TokenT>>(&mut loan.coins)
    }

    public fun fill_coins<TokenT: store>(token:Token<TokenT>, loan: &mut FlashLoan<TokenT>)  {
        Option::fill<Token<TokenT>>(&mut loan.coins, token)
    }

    public fun repay<TokenT: store>(loan:FlashLoan<TokenT>) {
        let FlashLoan {amount, coins} = loan;
        assert(amount == Token::value<TokenT>(Option::borrow(&coins)), Errors::invalid_argument(1));
        // put to pool
    }
}
```

MOVE 的创新点

- 定义了一套状态操作协议（状态所有权，面向类型）
- 通过 Ability 机制实现了“类型”在合约之间的共享

三个角度

- 合约的状态机制
- 编程语言的特性
- 合约之间的依赖与调用

有没有其他的创新点?

技术人的第二个黄金时代

BUILD

Q&A

首届STARCOIN MOVE
线上黑客松进行中

- 第一期即将结束
- 第二期即将开始