TOKEN - EVM BLOCKCHAIN

TOKEN STANDARDS

INTRODUCTION

Trước tiên, ta cần biết "token" là gì?

TOKEN trong mạng BLOCKCHAIN, thực chất là bất kỳ hàng hóa nào có giá trị, và có liên quan đề cập đến một tài sản kỹ thuật số không thể trùng lặp (immutable), được phát hành (release) và trao đổi (exchange).

Đặc điểm nổi bật của token ở chỗ, nó có tính độc nhất và duy nhất sở hữu, không giống như một tệp máy tính đơn giản. Vì thế, nó có thể được coi như hàng hóa có giá trị trao đổi thực sự.

* *https://www.coinhouse.com/insights/blockchain-en/token*

Vậy làm thế nào để token có thể có các đặc điểm đó để được coi như là một tài sản thực sự?

Việc token là một tài sản có thể trao đổi, thực hiện giao dịch được phải có một quy chuẩn rõ ràng nhất quán và có thể chứng minh được nó là tài sản có tính sở hữu cần thiết.

ETHEREUM - một nền tảng blockchain phi tập trung (decentralized platform) cho phép xây dựng và chạy các ứng dụng phi tập trung (DApps) bằng cách sử dụng smart contract (hợp đồng thông minh), sử dụng token để tạo ra các hàng hóa có giá trị lưu thông trao đổi mua bán và thanh toán.

Nhiều tiêu chuẩn phát triển Ethereum (Ethereum Development Standards) hiện nay tập trung vào hình thức token. Các tiêu chuẩn này giúp đảm bảo các hợp đồng thông minh vẫn có thể kết hợp được, để đáp ứng với nhu cầu tương thích với các ứng dụng và sàn giao dịch phi tập trung hiện có. Các TOKEN STANDARD này xác định cách mà các token hoạt động và tương tác trên hệ sinh thái Ethereum. Giúp các nhà phát triển xây dựng có tính nhất quán rõ rệt và làm cho Ethereum được kết nối với nhau nhiều hơn.

* *https://ethereum.org/developers/docs/standards/tokens/*

PREREQUISITES

Vậy các tiêu chuẩn Token (Token Standards) này đòi hỏi cần những cái gì để có thể sử dụng được Token từ các tiêu chuẩn đề ra?

Một giao dịch tài sản cơ bản là một hành động "bắt tay" trao đổi tài sản từ hai phía. Nếu mở rộng, các giao dịch này được thực hiện nhiều lần, vì vậy cái "bắt tay" này phải có một cái chuẩn cụ thể để có thể phục vụ cho việc tái giao dịch với nhau để có tính nhất quán. Và cũng cần phải có cái gì đó để chứng minh rằng, hai phía này đã "bắt tay" với nhau để có tính chứng thực.

Trong hệ sinh thái Ethereum, để đảm bảo tính nhất quán, người ta đã phát triển các tiêu chuẩn Ethereum (Ethereum Development Standards) mà Smart Contract được dùng để chứng thực rằng giao dịch đã được xảy ra.

PREREQUISITES – SMART CONTRACT

Smart Contract là gì?

Smart Contract là một chương trình chạy trên Ethereum blockchain. Nó là tập hợp mã (chức năng của nó) và dữ liệu (trạng thái của nó) nằm tại một địa chỉ cụ thể trên Ethereum blockchain.

Smart Contract cũng được coi là một loại tài khoản Ethereum. Tuy nhiên, chúng không được người dùng kiểm soát, thay vào đó được triển khai vào mạng và chạy như đã được lập trình. Tài khoản người dùng sẽ tương tác với hợp đồng thông minh bằng cách gửi các giao dịch thực hiện chức năng được xác định trên hợp đồng thông minh.

Hợp đồng thông minh có thể xác định các quy tắc, giống như hợp đồng thông thường và tự động thực thi chúng thông qua mã. Hợp đồng thông minh không thể bị xóa theo mặc định và các tương tác với chúng là không thể đảo ngược.

Vậy Smart Contract hoạt động như thế nào?

Smart Contract được ví như một máy bán hàng tự động, chúng ta để mà muốn mua được hàng hóa mong muốn chúng ta cần đưa tiền vào máy và lựa chọn món hàng cần mua sẽ có được món hàng đó (đã được lập trình chính xác vào máy từ trước)

```
money + snack selection = snack dispensed
```

Một ví dụ về Smart Contract:

```
pragma solidity 0.8.7;

contract VendingMachine {

    // Declare state variables of the contract
    address public owner;
    mapping (address => uint) public cupcakeBalances;

    // When 'VendingMachine' contract is deployed:
     // 1. set the deploying address as the owner of the contract
     // 2. set the deployed smart contract's cupcake balance to 100
     constructor() {
         owner = msg.sender;
         cupcakeBalances[address(this)] = 100;
     }

     // Allow the owner to increase the smart contract's cupcake balance
     function refill(uint amount) public {
         require(msg.sender == owner, "Only the owner can refill.");
         cupcakeBalances[address(this)] += amount;
     }

     // Allow anyone to purchase cupcakes
     function purchase(uint amount) public payable {
         require(msg.value >= amount * 1 ether, "You must pay at least 1 ETH per cupc
         require(cupcakeBalances[address(this)] >= amount, "Not enough cupcakes in st
         cupcakeBalances[address(this)] -= amount;
         cupcakeBalances[msg.sender] += amount;
     }
 }
```

Ví dụ trên là một chương trình Solidity cho một máy bán bánh cupcake tự động trên blockchain Ethereum.

Đầu tiên, trong một hợp đồng (máy bán hàng) khởi tạo các biến giá trị cho hợp đồng (chủ sở hữu của máy bán hàng, số bánh có trong máy). Thực hiện triển khai máy bán hàng, người mua cần mua bánh khi biết trước trong máy sẽ có ban đầu với 100 bánh cupcake.

Cho phép chỉ người chủ sở hữu được nạp bánh vào máy amount bánh cupcake, vào cho phép bất kỳ ai được mua bánh này từ máy bán hàng tự động.

Giả sử số tiền cần mua mỗi bánh là 1 ETH. Vậy cần phải trả đủ tiền để mua bánh và máy phải còn đủ bánh để bán.

Vậy, nếu bạn mua 3 bánh từ máy bạn cần gửi 3 ETH vào máy và trong máy còn 97 bánh, bạn nhận được 3 bánh. Thế là, ở đây tiền vẫn còn nằm trong máy bán hàng chứ không về chủ sở hữu lập tức.

* *https://ethereum.org/developers/docs/smart-contracts/*

PREREQUISITES – ETHEREUM DEVELOPMENT STANDARDS

Những tiêu chuẩn phát triển Ethereum được đề ra từ cộng đồng

Các tiêu chuẩn về phát triển Ethereum do một cộng đồng người dùng đảm nhiệm duy trì và phát triển. Cộng đồng Ethereum đã áp dụng nhiều tiêu chuẩn giúp duy trì khả năng tương tác giữa các dự án (như máy khách và ví Ethereum) trong các lần triển khai và đảm bảo các hợp đồng thông minh và ứng dụng phi tập trung vẫn có thể cấu thành được.

* *https://ethereum.org/developers/docs/standards/*

Ta cần biết các chuẩn này được tạo ra từ cộng đồng thông qua một quy trình chuẩn mở dưới dạng các Đề xuất Cải tiến Ethereum ***(EIPs – Ethereum Improvement Proposals)*** được thảo luận vào thống nhất tiêu chuẩn.

Để biết thêm các thông tin về các loại tiêu chuẩn, cũng như phân loại các nhóm đề xuất hãy bắt đầu với ***EIP-1***.

TOKEN STANDARDS

Thế Token Standards để xây dựng token trên EVM?

Chúng ta quay lại với vấn đề ***Token Standards***, để xây dựng token trên ***Ethereum Virtual Machine (EVM)***, cần biết được các chuẩn phổ biến thường dùng để xây dựng chương trình mà sau này được nạp vào ***EVM Code***, cũng như phân loại token dùng cho các mục đích khác nhau.

Các chuẩn thường được dùng để xây dựng token phổ biến như:

- ***ERC-20:*** *Giao diện chuẩn cho các token có thể thay thế (có thể hoán đổi), chẳng hạn như token bỏ phiếu, token đặt cược hoặc tiền ảo.*
  - ***ERC-223:*** *Một tiêu chuẩn token có thể thay thế giúp token hoạt động giống hệt như ether và hỗ trợ xử lý chuyển token ở phía người nhận.*
  - ***ERC-1363:*** *Giao diện mở rộng cho token ERC-20, hỗ trợ thực hiện lệnh gọi lại trên hợp đồng của người nhận trong một giao dịch duy nhất.*
- ***ERC-721:*** *Giao diện chuẩn cho các token không thể thay thế, chẳng hạn như chứng thư cho tác phẩm nghệ thuật hoặc bài hát.*

    + ***ERC-2309:*** *Sự kiện chuẩn được phát ra khi tạo/chuyển một hoặc nhiều token không thể thay thế bằng cách sử dụng các mã định danh token liên tiếp.*
    + ***ERC-4400:*** *Tiện ích mở rộng giao diện cho vai trò người dùng EIP-721.*
    + ***ERC-4907:*** *Thêm vai trò giới hạn thời gian với các quyền hạn hạn chế cho token ERC-721.*
- ***ERC-777:*** *Một tiêu chuẩn token cải tiến so với ERC-20.*
- ***ERC-1155:*** *Một tiêu chuẩn token có thể chứa cả tài sản có thể thay thế và không thể thay thế.*
- ***ERC-4626:*** *Một tiêu chuẩn kho lưu trữ token hóa được thiết kế để tối ưu hóa và thống nhất các thông số kỹ thuật của kho lưu trữ sinh lời.*
- ***Ngoài ra,*** *còn sử dụng một số tiêu chuẩn khác cho các tài sản riêng biệt như* ***BEP-20****,* ***BEP-2****,* ***SPL.***

* *https://www.binance.com/en/academy/glossary/token-standards*

* *https://eips.ethereum.org/erc*

* *https://ethereum.org/developers/docs/standards/*

Về phân loại token, có 3 kiểu token hiện nay là *Fungible Token, Non-Fungible Token (NFT), Semi-Fungible Token (SFT).*

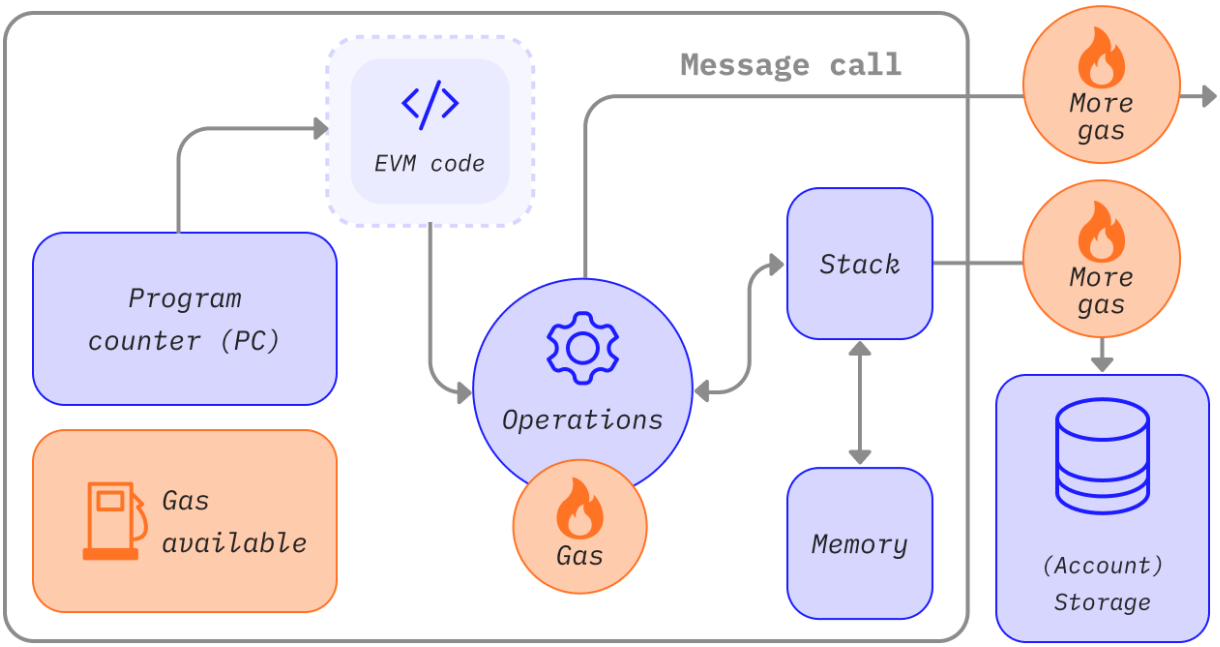
1. ***Fungible Token,*** *nghĩa là token có thể thay thế, hiểu một cách khác là mỗi token là có cùng giá trị và đặc tính giống nhau, chúng không có sự khác biệt, là sự sao chép hoàn hảo.*
2. ***Non-Fungible Token (NFT),*** *nghĩa là token không thể thay thế, là một loại token mật mã đại diện cho một tài sản duy nhất, mỗi token là hoàn toàn khác nhau.*
3. ***Semi-Fungible Token (SFT),*** *nghĩa là token bán thay thế, là một loại token kết hợp các đặc tính từ* ***FT*** *và* ***NFT*** *để phù hợp với sự biến đổi trạng thái của giá trị hàng hóa trong quá trình vận động chuyển biến của tài sản.*

‘

HOW TO CREATE TOKEN ON EVM

Để tạo được token trên EVM thì trước tiên ta cần biết cơ chế hoạt động của EVM để xác định được tạo token ở chỗ nào và được tạo ra như thế nào?

Máy ảo Ethereum (EVM) là một môi trường ảo phi tập trung thực thi mã một cách nhất quán và an toàn trên tất cả các nút Ethereum.

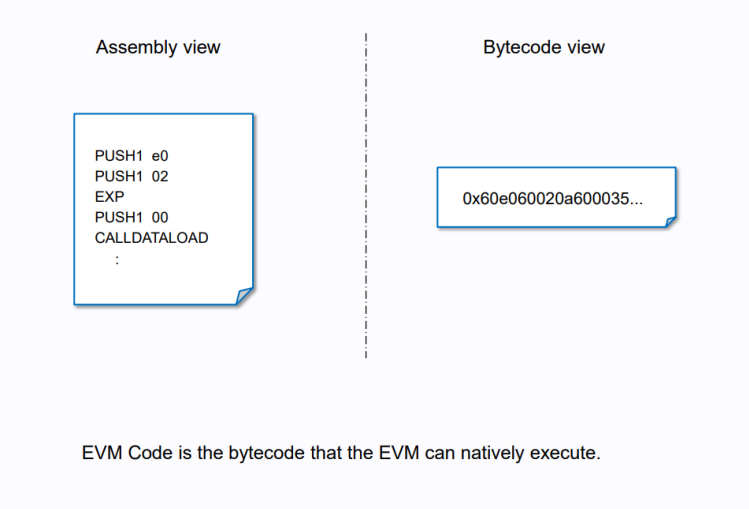


Giao dịch là các lệnh được ký mã hóa từ các tài khoản. Có hai loại giao dịch: loại dẫn đến các lệnh gọi tin nhắn và loại dẫn đến việc tạo hợp đồng.

Việc tạo hợp đồng dẫn đến việc tạo một tài khoản hợp đồng mới chứa mã bytecode hợp đồng thông minh đã biên dịch. Bất cứ khi nào một tài khoản khác thực hiện lệnh gọi tin nhắn đến hợp đồng đó, nó sẽ thực thi mã bytecode của nó.

EVM hoạt động gần giống như máy bán hàng tự động chạy bằng pin với lượng gas bị tiêu hao chính là lượng điện được tiêu thụ từ pin. ***Account Storage*** là nơi lưu trữ lượng tiền (ta xem tài sản dưới dạng tiền xu, mỗi đồng tiền xu tương đương với 1 token có giá trị thực tế). Việc tốn gas ít cho các hoạt động tiêu thụ điện tử, và tốn nhiều gas từ các hoạt động điện cơ.

Vì thế, ta có thể xác định rằng việc tạo token nằm trong ***EVM Code*** hay nói cách khác các token có giá trị được tạo ra từ các ***Smart Contract*** dưới dạng ***bytecode (các opcode)*** được nạp vào ***Operation*** để thực thi các lệnh gọi ***(Message Call)***.



* https://www.geeksforgeeks.org/ethical-hacking/what-is-ethereum-virtual-machine-and-how-it-works/

* https://ethereum.org/developers/docs/evm/

Các EVM Code thực thi đa phần các lệnh cần thiết trong cả EVM được phân loại thành các nhóm từ bảng danh sách EVM Opcodes Interactive:

- ***01. Stop and Arithmetic Operations:*** *opcode từ 00 đến 09, 0A và 0B*
- ***02. Comparison and Bitwise Logic Operations****: opcode từ 10 đến 1D*
- ***03. SHA3:*** *opcode 20*
- ***04. Environmental Information:*** *opcode từ 30 đến 3F*
- ***05. Block Information:*** *opcode từ 40 đến 4A*
- ***06. Stack Memory Storage and Flow Operations:*** *opcode từ 50 đến 5E*
- ***07. Push Operations:*** *opcode 5F, từ 60 đến 6F, từ 70 đến 7F*
- ***08. Duplication Operations:*** *opcode từ 80 đến 8F*
- ***09. Exchange Operations:*** *opcode từ 90 đến 9F*
- ***10. Logging Operations:*** *opcode A0, A1, A2, A3, A4*
- ***11. System Operations:*** *opcode F0, F1, F2, F3, F4, F5, FA, FD, FE, FF*

Thực thi EVM Code tạo token theo trình tự cơ bản sau:

1. ***Contract creation / deploy***

   07 – 04 – 04 – 06 – 11

   + *PUSH*, PUSH32, … để chuẩn bị các tham số (như initialSupply, name, symbol, decimals)*
   + *CALLDATASIZE, CALDATACOPY nếu constructor nhận tham số từ calldata*

+ *CODECOPY, CODESIZE để copy phần runtime code từ constructor phần lưu trữ*
+ *SSTORE để lưu các biến state (nghĩa là lưu tổng cung, tên token, ký hiệu…)*
+ *RETURN để kết thúc quá trình deploy, trả bytecode hợp đồng.*

2. ***Khởi tạo biến state trong constructor***

   06 – 10 – 08, 09, 06

+ *SSTORE các giá trị khởi tạo như: balances[deployer] = initialSupply*
+ *LOG (ví dụ LOG2, LOG3) để phát sự kiện Transfer(0x0, deployer, initialSupply)*
+ *Các opcode như DUP, SWAP, POP quản lí stack trong quá trình tính toán.*

3. ***Triển khai các hàm – Sau deploy, khi gọi contract***

   04, 07, 02, 06, 11 – 04, 06, 02, 06, 07, 10 – 07, 11

+ *Khi gọi hàm totalSupply(), balanceOf(address),…:*

  C*ALLDATALOAD, CALLDATASIZE, CALLDATACOPY để lấy tham số từ calldata*

  S*o sánh hàm selector (ví dụ 4 byte) bằng EQ, JUMPI để chọn đúng nhánh hàm*

  Đ*ọc storage bằng SLOAD để lấy biến như _totalSupply, balances[addr]*

  T*rả dữ liệu bằng MSTORE, RETURN.*

+ *Khi gọi hàm transfer(to, amount):*

  CALLDATALOAD lấy to và amount

  Kiểm tra điều kiện: SLOAD balances[msg.sender], so sánh với amount (LT, GT, ISZERO)

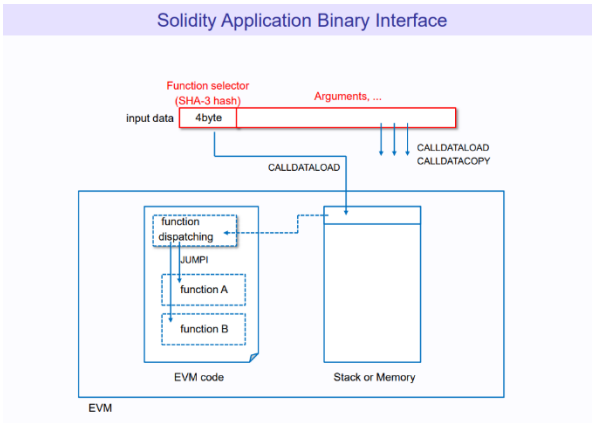
  Nếu điều kiện ok:

  - *SSTORE để cập nhật balances[msg.sender] -= amount*
  - *SSTORE để cập nhật balances[to] += amount*
  - *LOG3 (hoặc LOG2) để phát sự kiện Transfer(msg.sender, to, amount)*

  Trả về true (PUSH1, RETURN).

4. ***Các phép toán phụ trợ***

+ *Đọc address(this), caller, tx.origin bằng opcodes như ADDRESS, CALLER, ORIGIN nếu cần*
+ *Memory thao tác với MLOAD, MSTORE, MSTORE8 khi xử lý dữ liệu tạm thời*
+ *Stack quản lí với PUSH, DUP, SWAP, POP*
+ *Điều hướng với JUMP, JUMPI, JUMPDEST*



* https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf

* https://www.evm.codes/

DEPLOYMENT

Các máy khách thực thi ***Ethereum*** bao gồm việc triển khai ***EVM*** *(thường là Solidity)*. Ngoài ra, còn có nhiều cách triển khai độc lập, bao gồm:

1. ***Python:*** *Py-EVM*
2. ***C++:*** *evmone*
3. ***JavaScript:*** *ethereumjs-vm*
4. ***Rust:*** *revm*

Trong phần này, ta sẽ lấy một ví dụ về việc tạo ***token*** *kiểu Fungible Token* theo ***chuẩn ERC-20*** từ ***Smart Contract.*** *Sử dụng Solidity phiên bản 0.6.0*

```
pragma solidity ^0.6.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns (uint25

    function transfer(address recipient, uint256 amount) external returns (bool);
     function approve(address spender, uint256 amount) external returns (bool);
     function transferFrom(address sender, address recipient, uint256 amount) exter


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

GIẢI THÍCH CỤ THỂ HÌNH THỨC TOKEN

```
function totalSupply() external view returns (uint256);
```

Hàm totalSupply() cho trả về số lượng token hiện có. Hàm này là một hàm getter và không sửa đổi trạng thái của hợp đồng. Do đó, hầu hết các token đều sử dụng 18 chữ số thập phân và sẽ trả về tổng nguồn cung và các kết quả khác như sau: 10000000000000000000 cho 1 token.

```
function balanceOf(address account) external view returns (uint256);
```

Trả về số lượng token do một địa chỉ (tài khoản) sở hữu. Hàm này là một hàm lấy dữ liệu và không thay đổi trạng thái của hợp đồng.

```
function allowance(address owner, address spender) external view returns (uint256);
```

Tiêu chuẩn ERC-20 cho phép một địa chỉ cấp quyền cho một địa chỉ khác để có thể lấy token từ địa chỉ đó. Hàm getter này trả về số token còn lại mà người chi tiêu được phép chi tiêu thay mặt cho chủ sở hữu. Hàm này là một hàm getter và không sửa đổi trạng thái của hợp đồng, đồng thời mặc định sẽ trả về 0.

```
function transfer(address recipient, uint256 amount) external returns (bool);
```

Di chuyển số lượng token từ địa chỉ gọi hàm (msg.sender) đến địa chỉ người nhận. Hàm này phát ra sự kiện Transfer được định nghĩa sau. Sự kiện này trả về true nếu việc chuyển dữ liệu thành công.

```
function approve(address spender, uint256 amount) external returns (bool);
```

Thiết lập số tiền trợ cấp mà người chi tiêu được phép chuyển từ số dư của người gọi hàm (msg.sender). Hàm này phát ra sự kiện Phê duyệt. Hàm trả về kết quả xem trợ cấp đã được thiết lập thành công hay chưa.

```
function transferFrom(address sender, address recipient, uint256 amount)
```

Chuyển số lượng token từ người gửi đến người nhận bằng cơ chế cấp phát. Số lượng token sau đó được trừ vào hạn mức của người gọi. Hàm này phát ra sự kiện Transfer.

```
event Transfer(address indexed from, address indexed to, uint256 value);
```

Sự kiện này được phát ra khi số lượng token (giá trị) được gửi từ địa chỉ "from" đến địa chỉ "to".

Trong trường hợp tạo token mới, việc chuyển thường diễn ra từ địa chỉ 0x00..0000, trong khi trong trường hợp đốt token, việc chuyển sẽ diễn ra đến địa chỉ 0x00..0000.

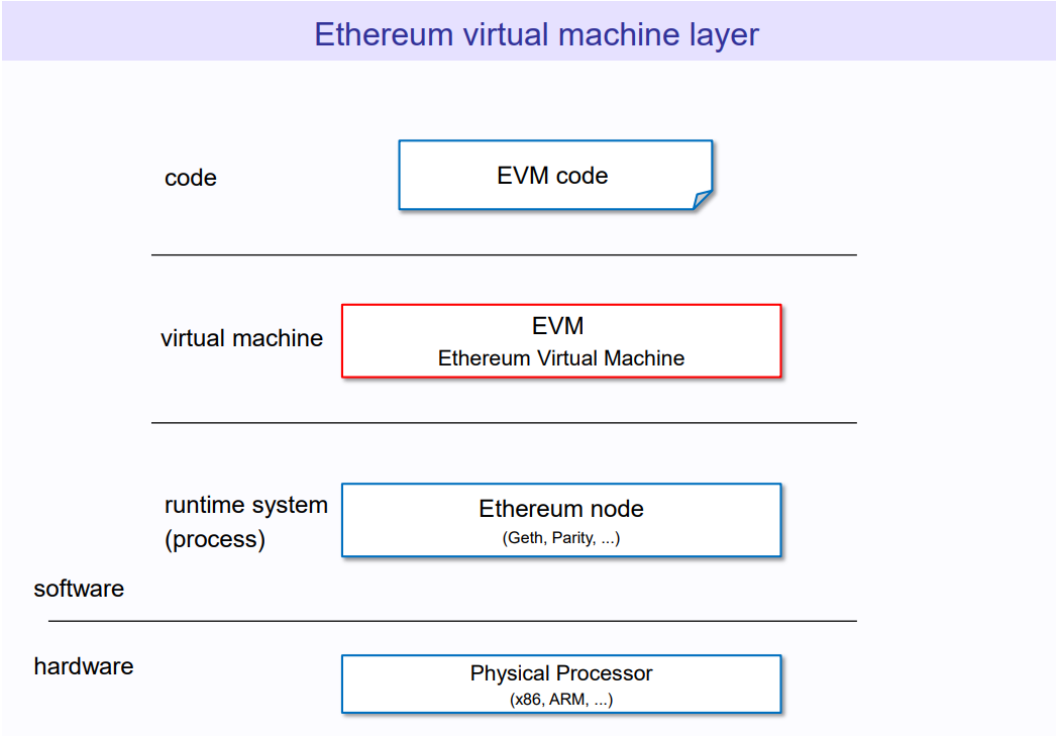
```
event Approval(address indexed owner, address indexed spender, uint256 value);
```

Sự kiện này được phát ra khi số lượng mã thông báo (giá trị) được chủ sở hữu chấp thuận để người chi tiêu sử dụng.

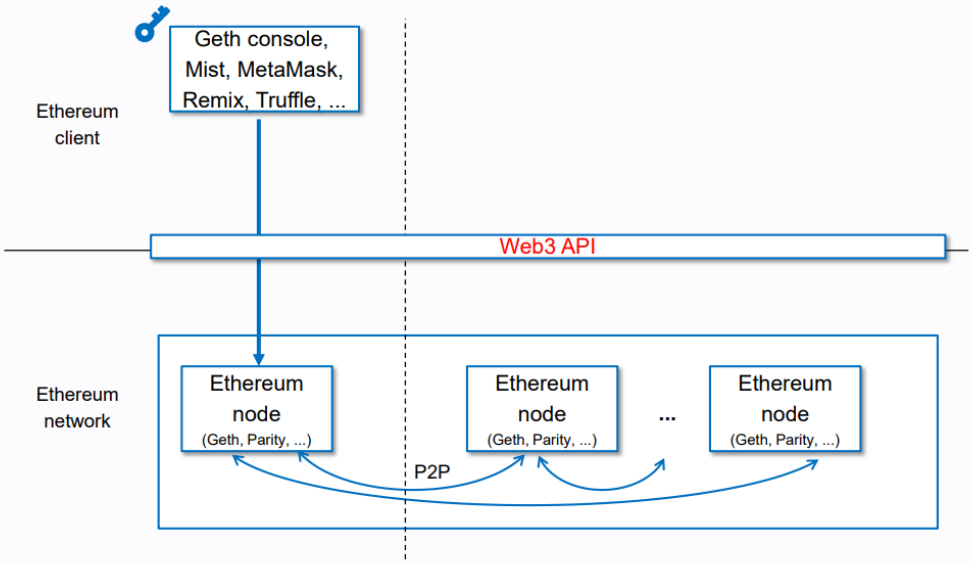
```
pragma solidity ^0.8.0;

interface IERC20 {

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function allowance(address owner, address spender) external view returns (uint25

    function transfer(address recipient, uint256 amount) external returns (bool);
     function approve(address spender, uint256 amount) external returns (bool);
     function transferFrom(address sender, address recipient, uint256 amount) exter


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


 contract ERC20Basic is IERC20 {

     string public constant name = "ERC20Basic";
     string public constant symbol = "ERC";
     uint8 public constant decimals = 18;


     mapping(address => uint256) balances;

     mapping(address => mapping (address => uint256)) allowed;

     uint256 totalSupply_ = 10 ether;
```

```
    constructor() {
     balances[msg.sender] = totalSupply_;
     }

     function totalSupply() public override view returns (uint256) {
     return totalSupply_;
     }

     function balanceOf(address tokenOwner) public override view returns (uint256)
         return balances[tokenOwner];
     }

     function transfer(address receiver, uint256 numTokens) public override returns
         require(numTokens <= balances[msg.sender]);
         balances[msg.sender] = balances[msg.sender]-numTokens;
         balances[receiver] = balances[receiver]+numTokens;
         emit Transfer(msg.sender, receiver, numTokens);
         return true;
     }

     function approve(address delegate, uint256 numTokens) public override returns
         allowed[msg.sender][delegate] = numTokens;
         emit Approval(msg.sender, delegate, numTokens);
         return true;
     }

     function allowance(address owner, address delegate) public override view retur
         return allowed[owner][delegate];
     }

     function transferFrom(address owner, address buyer, uint256 numTokens) public
         require(numTokens <= balances[owner]);
         require(numTokens <= allowed[owner][msg.sender]);

         balances[owner] = balances[owner]-numTokens;
         allowed[owner][msg.sender] = allowed[owner][msg.sender]-numTokens;
         balances[buyer] = balances[buyer]+numTokens;
         emit Transfer(owner, buyer, numTokens);
         return true;
     }
}
```

APPENDIX

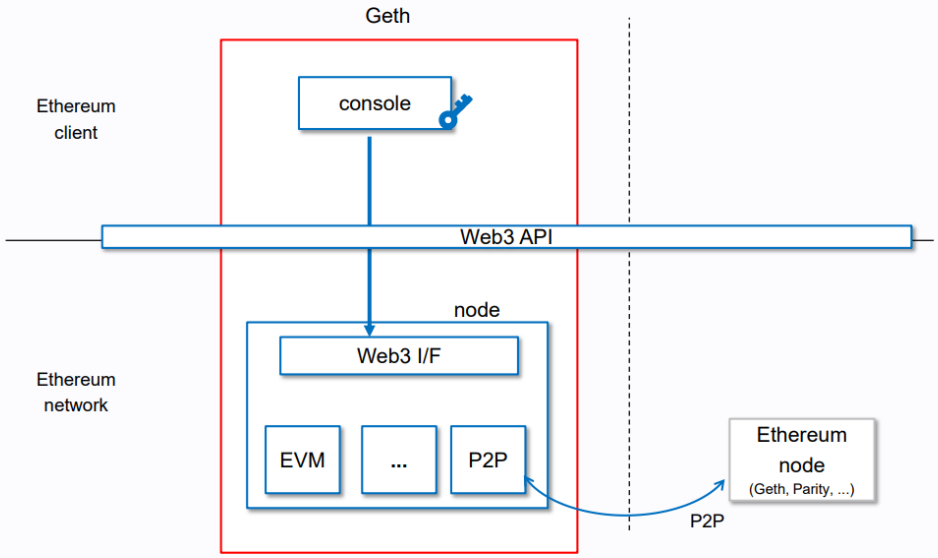

Ethereum virtual machine layer
code
EVM code
virtual machine
EVM
Ethereum Virtual Machine
runtime system
(process)
Ethereum node
(Geth, Parity, ...)
software
hardware
Physical Processor
(x86, ARM, ...)

Web3 API and client

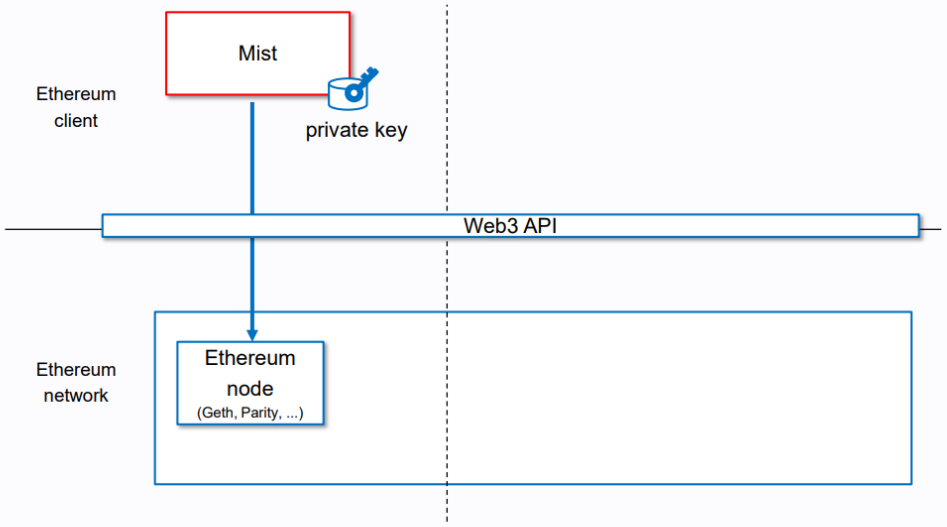


Ethereum clients access to Ethereum network via Web3 API.

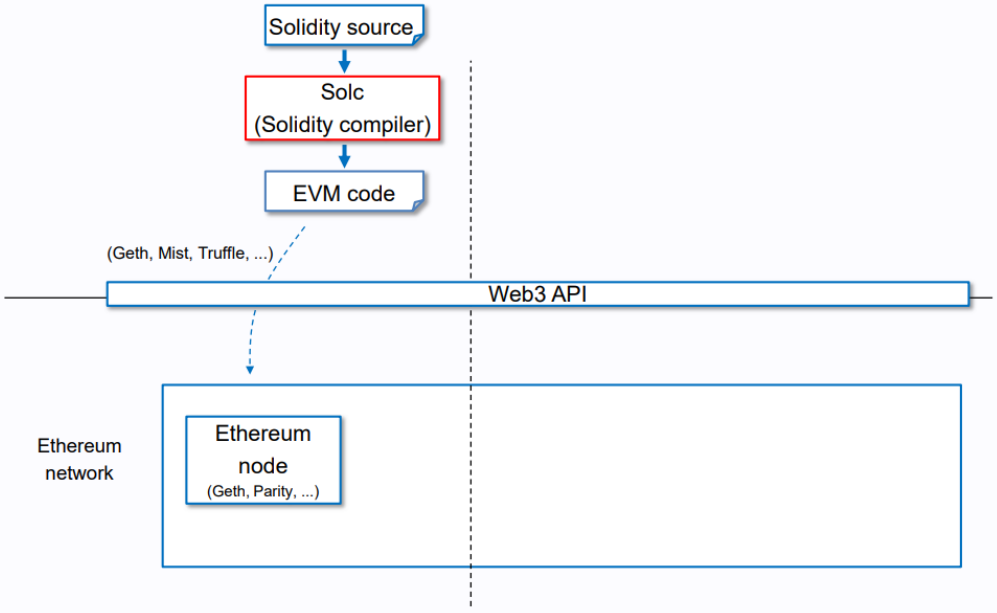
Geth



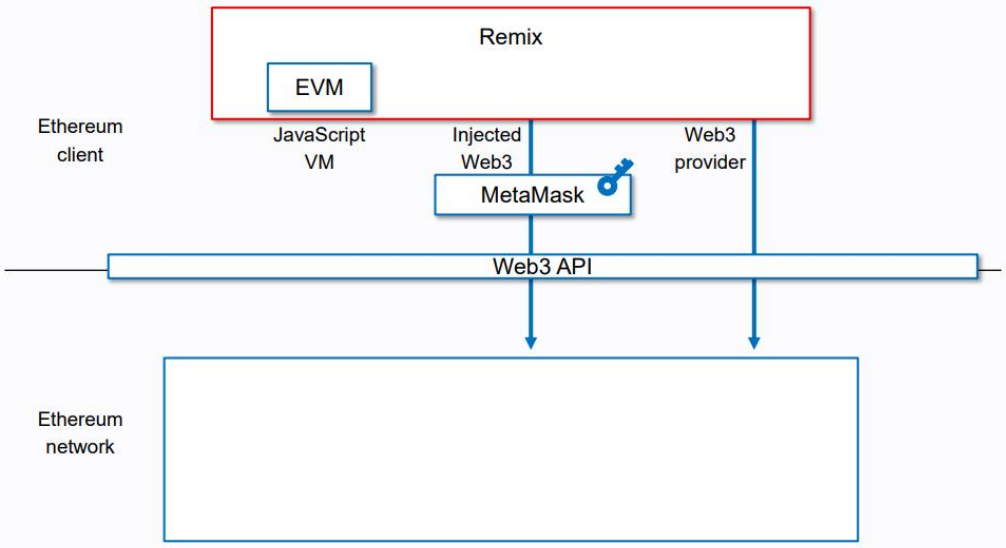
Mist


Mist
Ethereum client
private key
Web3 API
Ethereum network
Ethereum node
(Geth, Parity, ...)

Solc


Solidity source
Solc
(Solidity compiler)
EVM code
(Geth, Mist, Truffle, ...)
Web3 API
Ethereum network
Ethereum node
(Geth, Parity, ...)

Remix



Truffle