INFINITY
BLOCKCHAIN LABS

Ethereum

for noobs

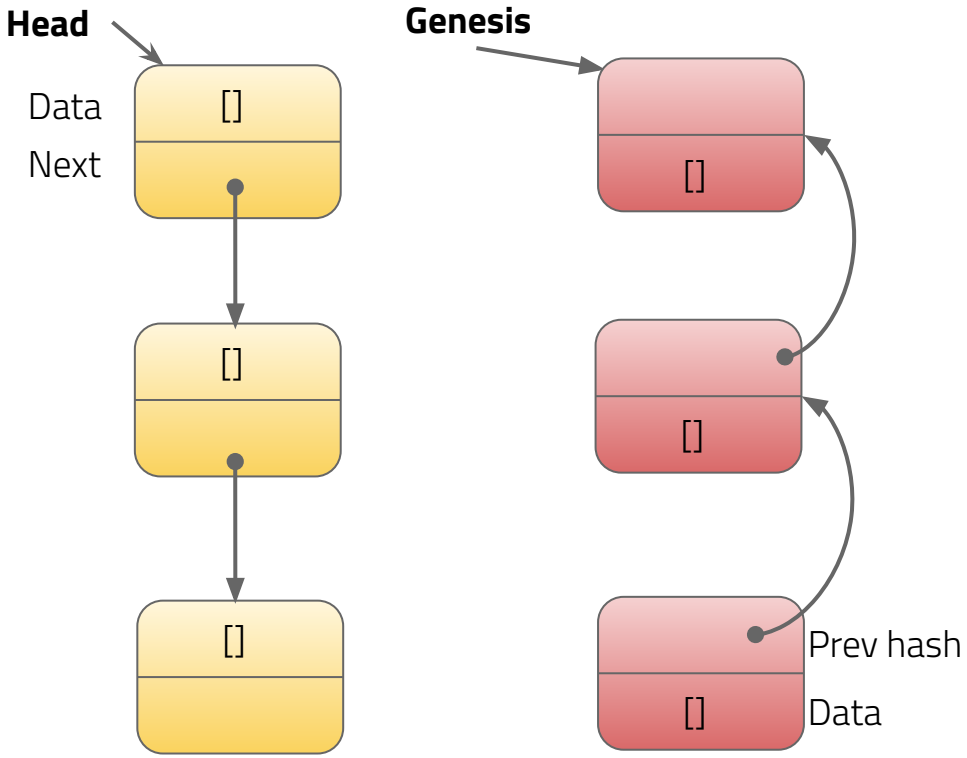
Overview & smart contract basic

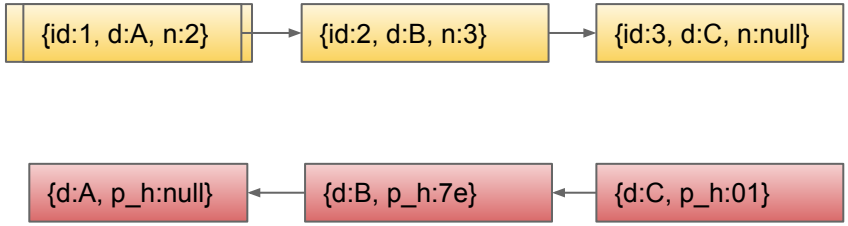
Hội anh em blockchain "thiện lành"

Contents

- Blockchain 101
- Ethereum overview
- Create smart contract using Solidity

What is blockchain?



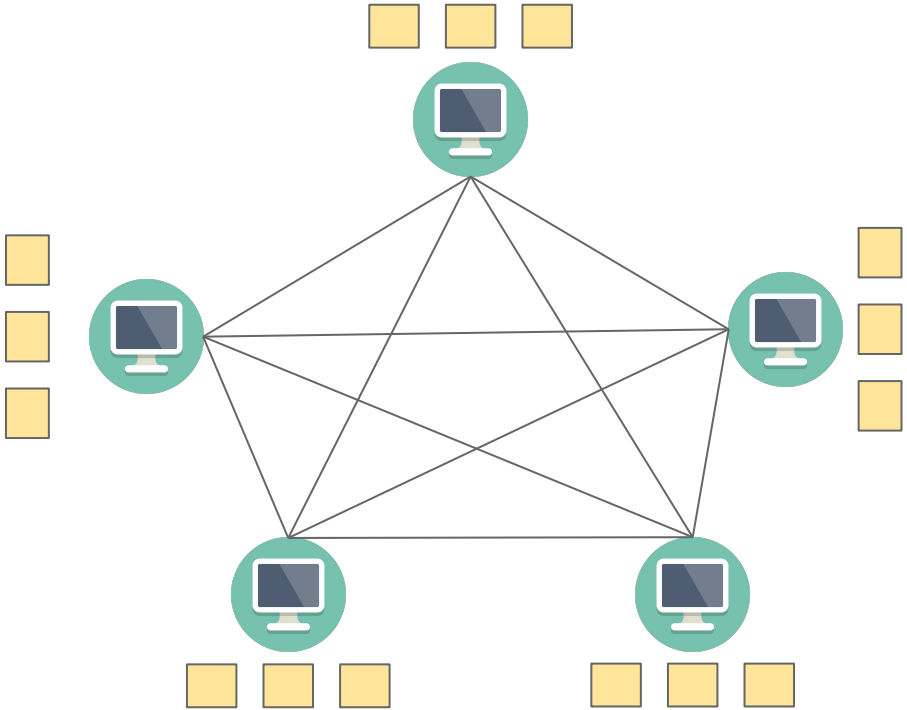


Immutable

Blockchain is a data structure:

- **Similar to linked list.**
- **Pointer is the hash of the previous element.**

What is blockchain?



Peer to peer network

- No center server.
- A node connects directly to other nodes.
- Each node maintains a copy of the blocks.

Decentralized

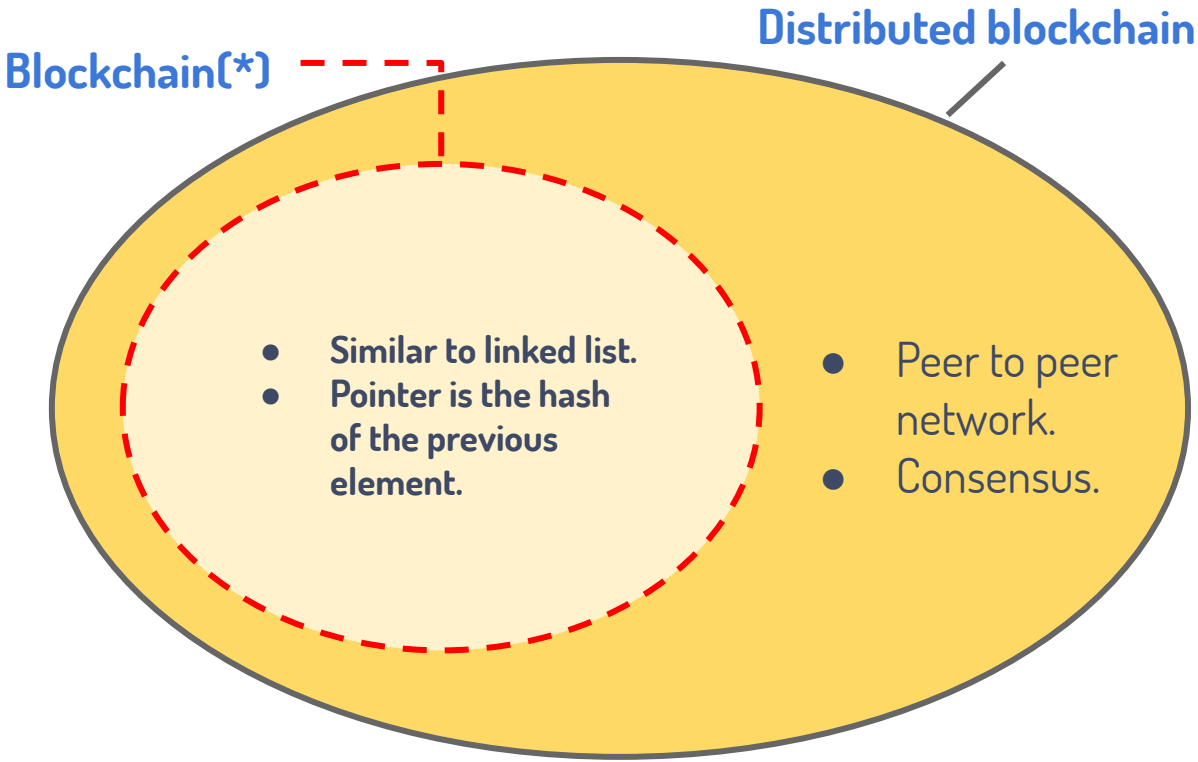
What is blockchain?

A protocol that is used to achieve the common agreement.

Transparent

https://www.leadstrat.com/blog/5-finger-consensus/

What is blockchain?



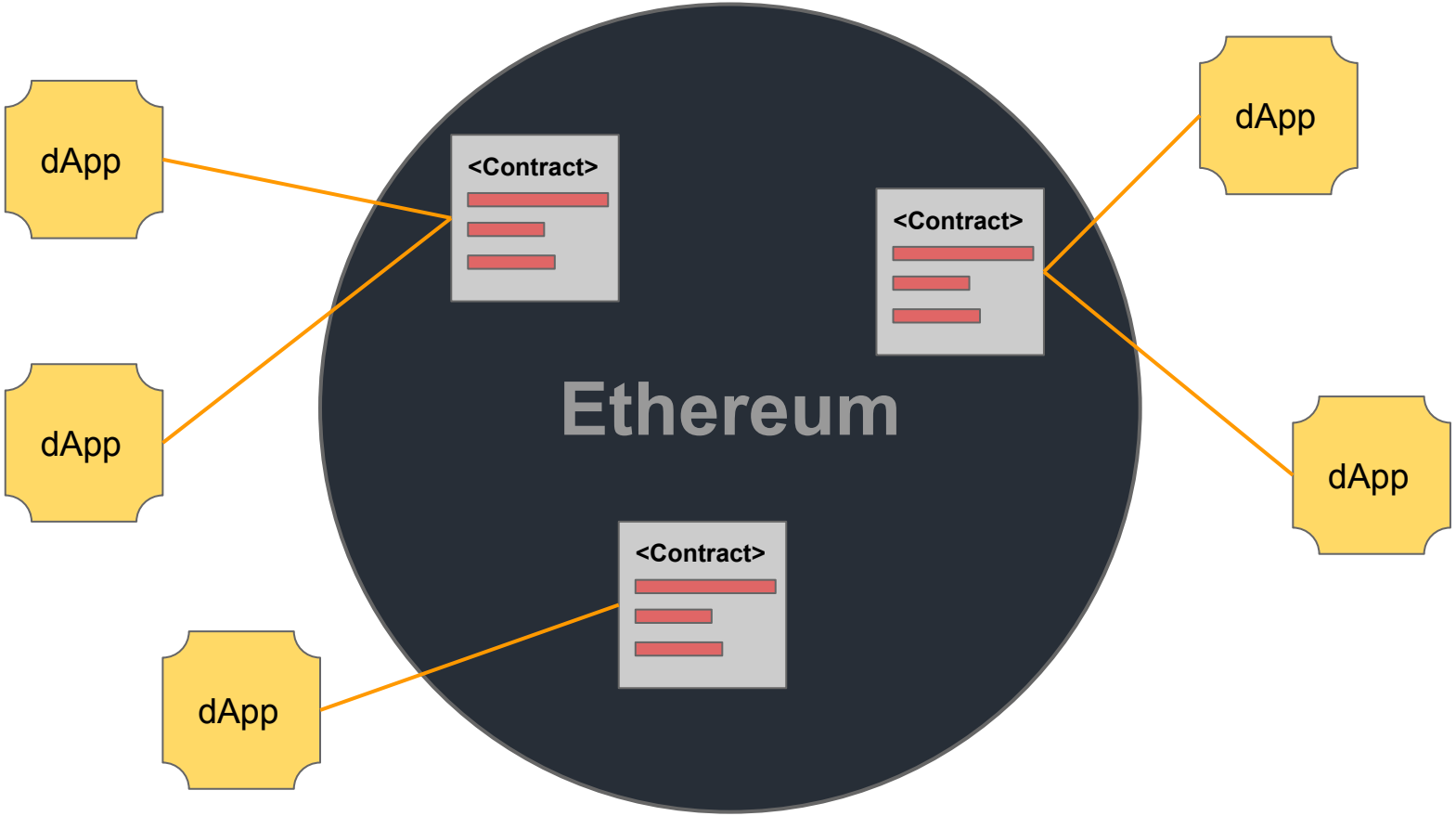
- **Immutable**
- **Decentralized**
- **Transparent**

Peer-to-Peer Electronic Cash System

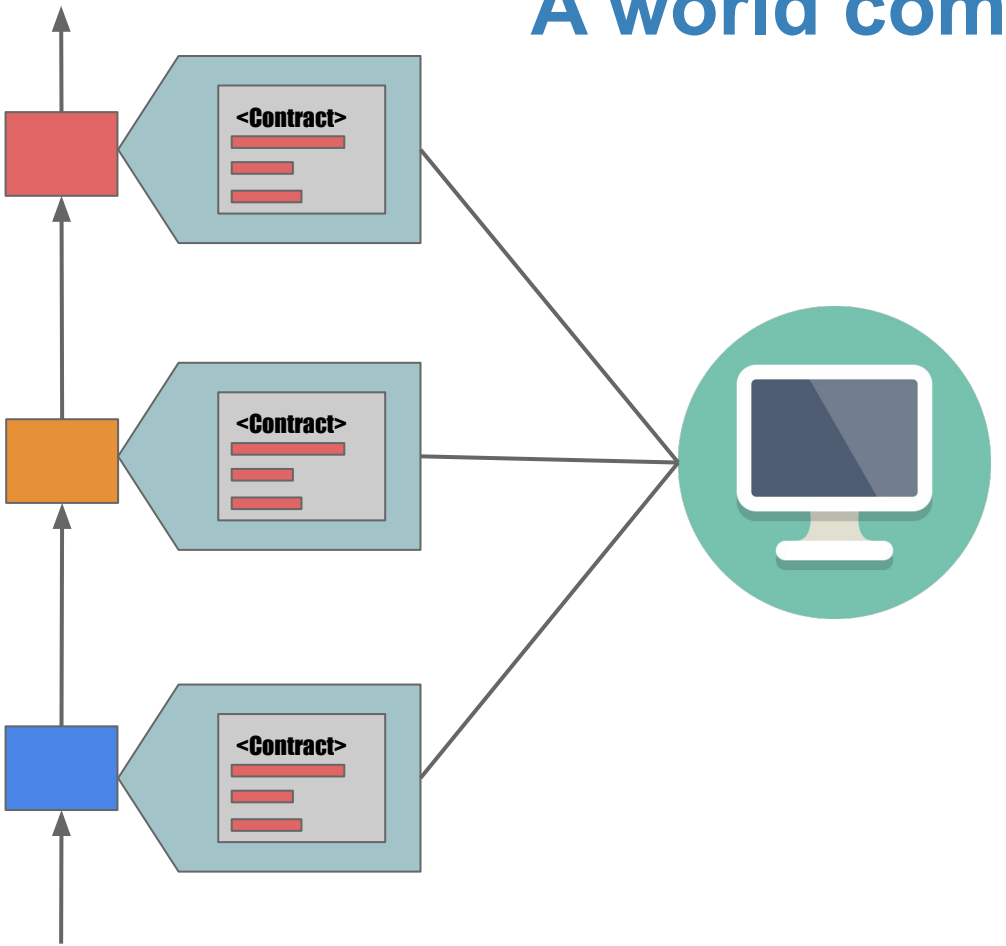
VS

Decentralized app platform

Ethereum is a blockchain app platform.

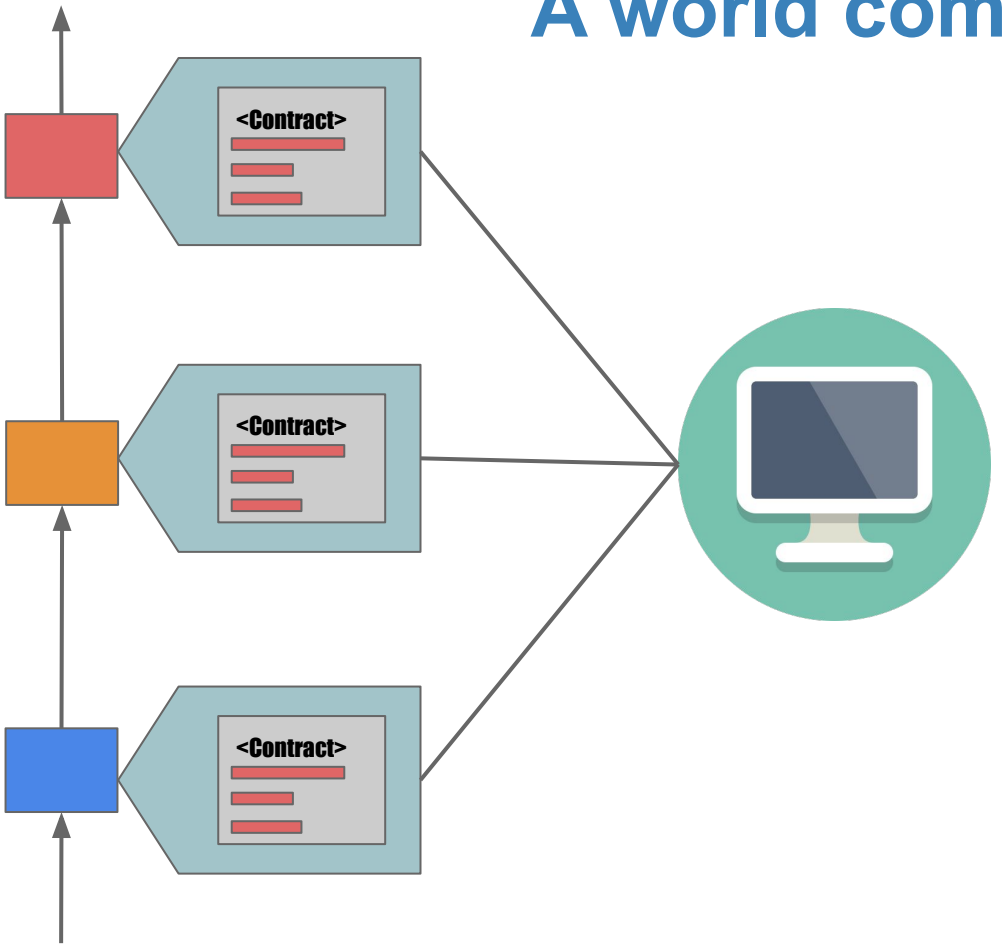


A world computer.

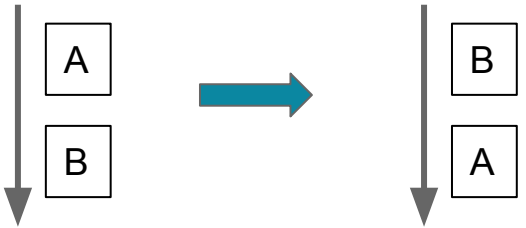


- Decentralized
- Transparent
- Atomic

A world computer.



- Expensive
- Slow
- Single thread
- The order of messages' execution can be reorganized



This computer uses "ether" to run.

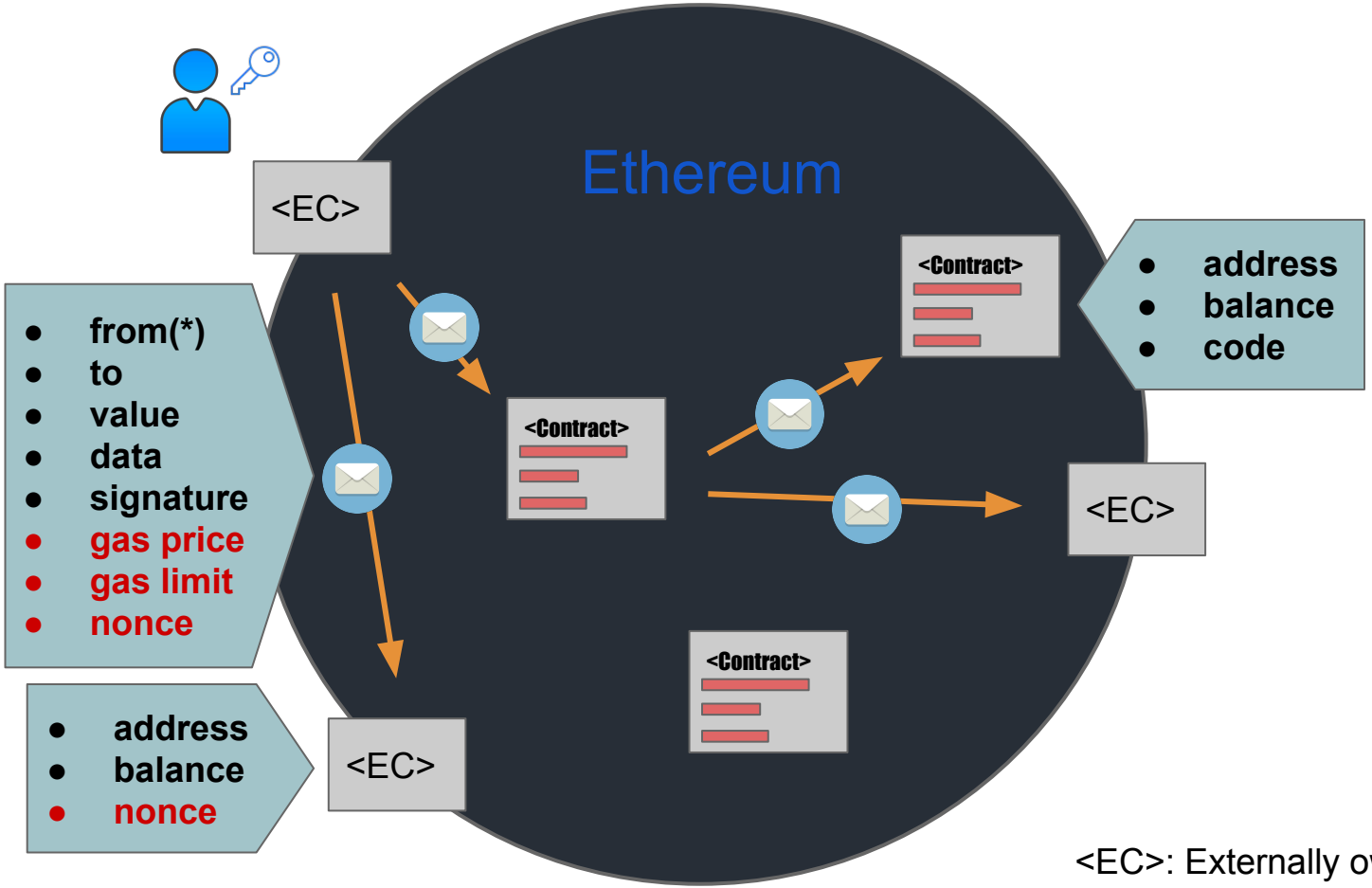
1ether = 10^{18} wei

Proof of work mining

2ethers/block/15seconds

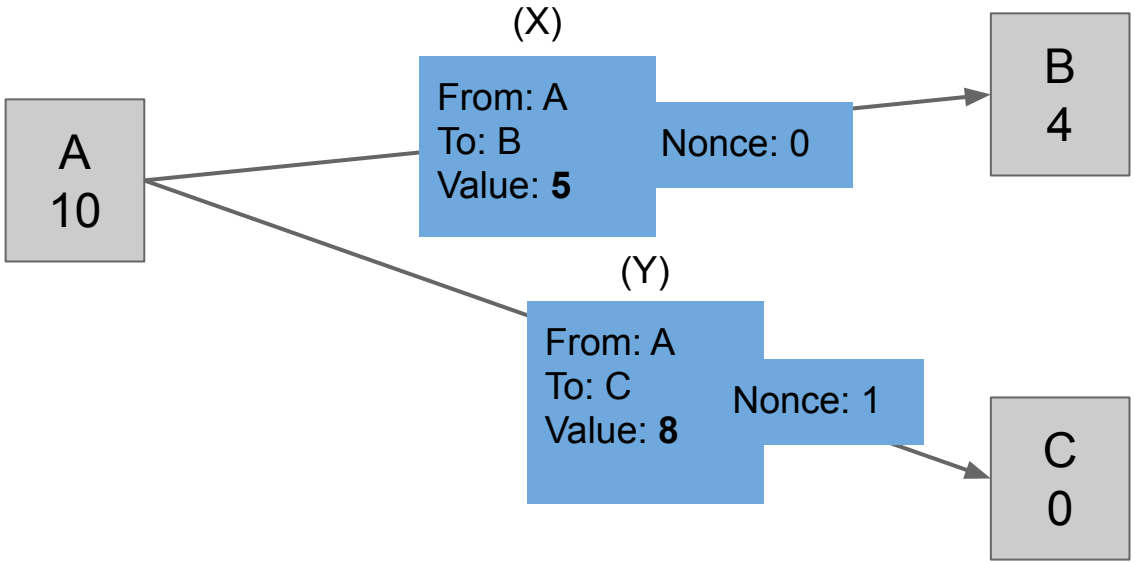
(transaction fee, uncle block fee)

https://coinmarketcap.com/currencies/ethereum/#charts



<EC>: Externally owned account

What is account's nonce using for?

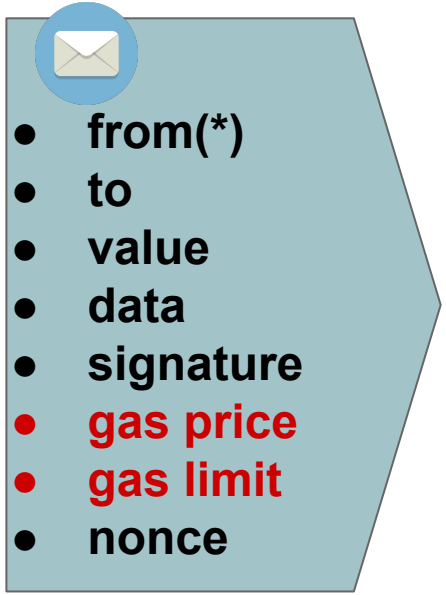


Nonce number specifies the order of transactions of an account.

Nonce of an account is total of successful transactions that were sent.

At the same time, commit 2 transactions(*) (X) and (Y). Which will be accepted?

(*) Transaction is a message that is sent from an externally owned account



What will happen if one of contracts has an infinity loop?

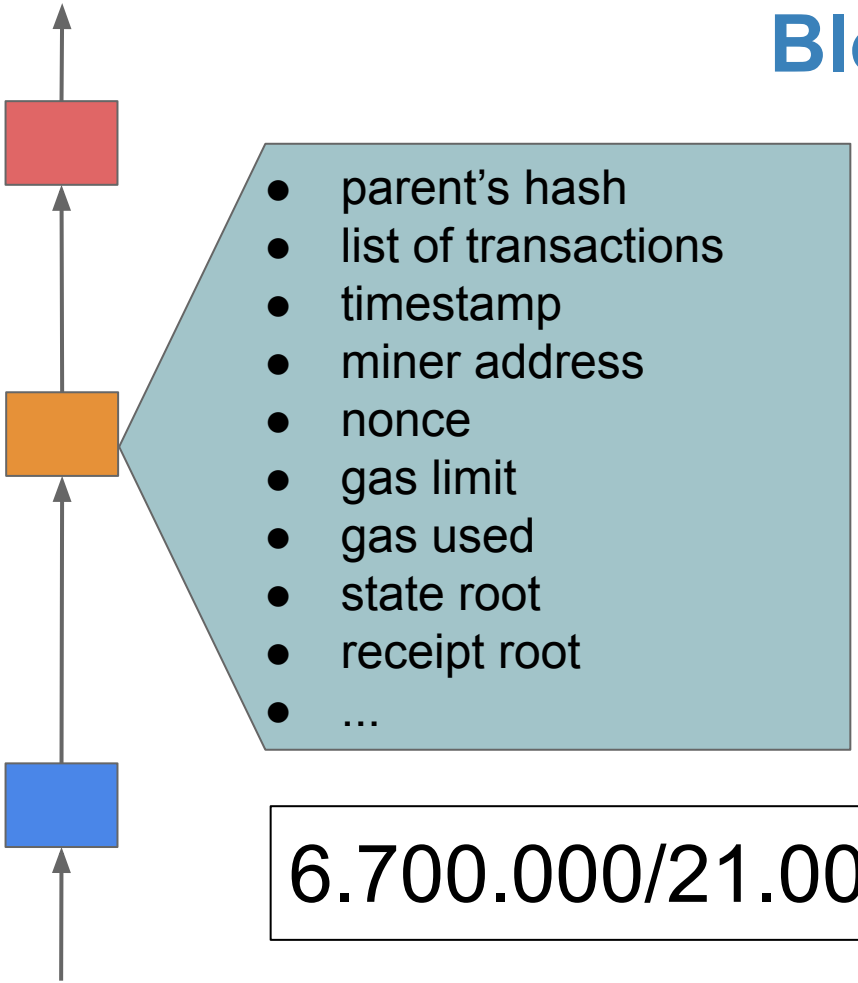
Pay to run ➡ How much I have to pay?

- Gas: Cost of 1 step run.
- Gas limit: Maximum gas.
- Gas price: Number of “ether” we pay for 1 gas.

  Avg: 3.100.000.000wei = 3.1Gwei = 0.000000031ether($3.1*10^{-8}$)

fee = gas limit * gas price

Block

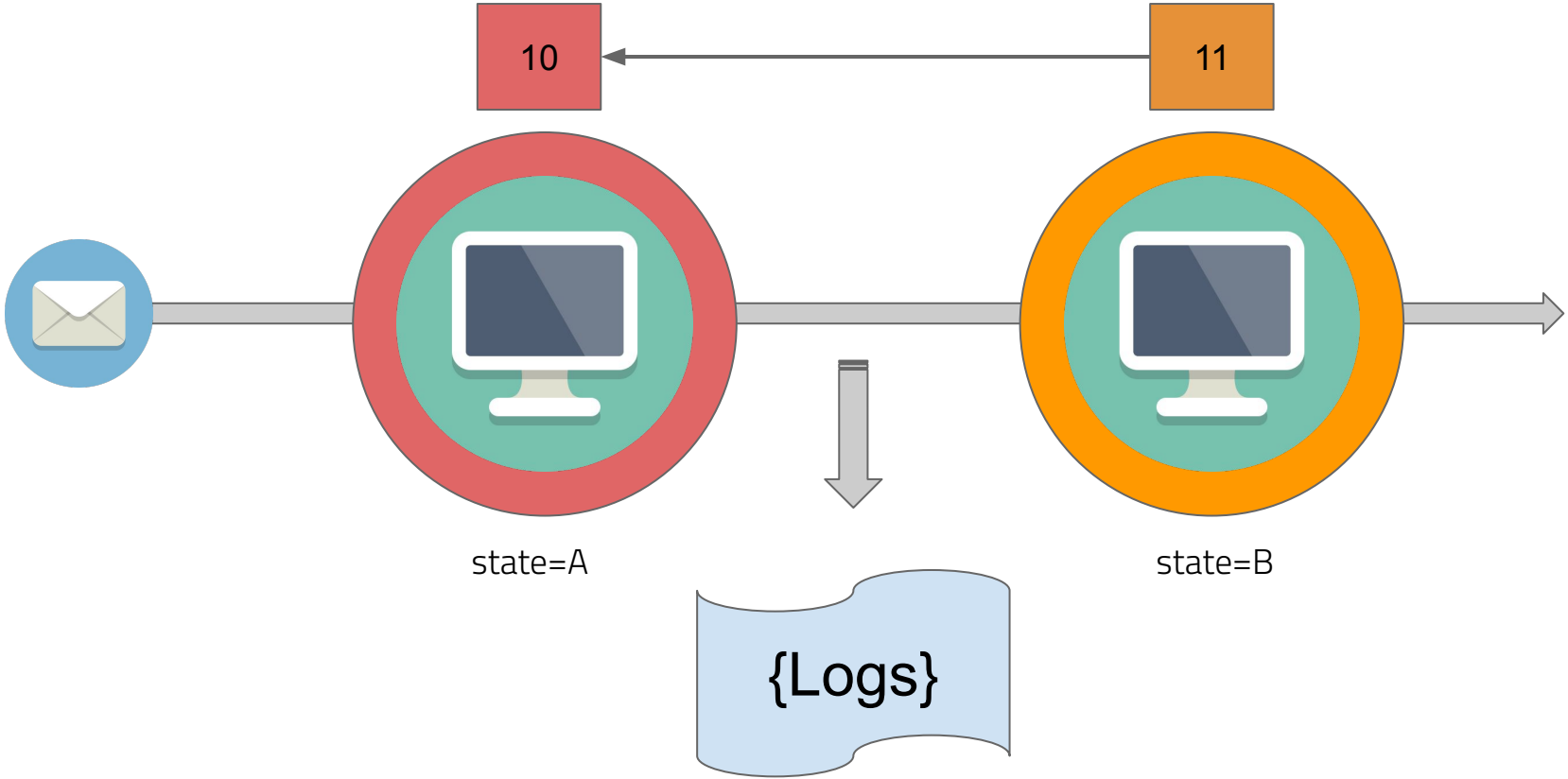


- parent's hash
- list of transactions
- timestamp
- miner address
- nonce
- gas limit
- gas used
- state root
- receipt root
- ...

- No size limit
- Gas limit is the maximum of total gas of all transactions
- Total used gas of all transactions

6.700.000/21.000 ~ 319 txs/block

Event



Summary

- **Account(*)**
  - Nonce is total successful transactions that were sent.
- **Transaction(*)**
  - Gas limit: Total gas that is allowed to use when the EVM run a transaction.
  - Gas price: Price of 1 gas(base on wei).
  - Nonce: The execution order of transactions of the sender account.
- **Block**
  - Gas limit: The maximum of total gas of all transactions.
- **Contract**
  - Only external owned account can trigger a transaction which calls a smart contract.
  - Need gas to execute.

Ethereum clients & networks

Clients

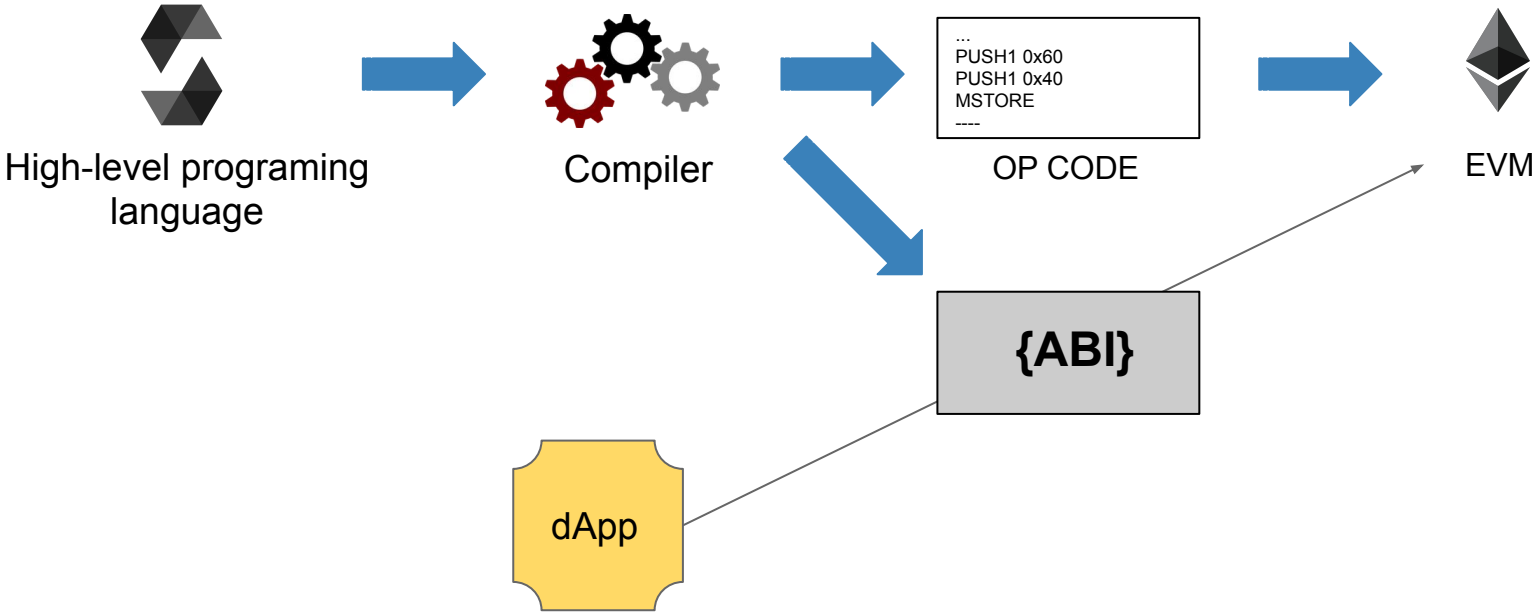
- Go-ethereum
- Parity
- Cpp-ethereum
- Pyethapp
- Ethereum(J)

Networks

- Mainnet
- Testnet
  - Ropsten
  - Kovan
  - Rinkerby
- Private net

< ether.camp >

Smart contract



Smart contract

```
pragma solidity ^0.4.0;

contract A {
   uint256 public total;
   function add() {
        total += 1;
   }
}
```

Compiler

OP Code

```
0x6060604052341561000f57600080fd5b5
b60b88061001e6000396000f30060606040
5263ffffffff7c0100000000000000000000
0000000000000000000000000000000000000
00600350416632ddbd13a8114604657806
34f2be91f146068575b600080fd5b341560
5057600080fd5b6056607a565b604051908
1526020016040518091039f35b34156072
57600080fd5b60786080565b005b6000548
1565b6000805460010190555b5600a16562
7a7a723058203980fdf4ba4267d30a67872
353d9240267848a68557534188a403900f7
ab070a0029
```

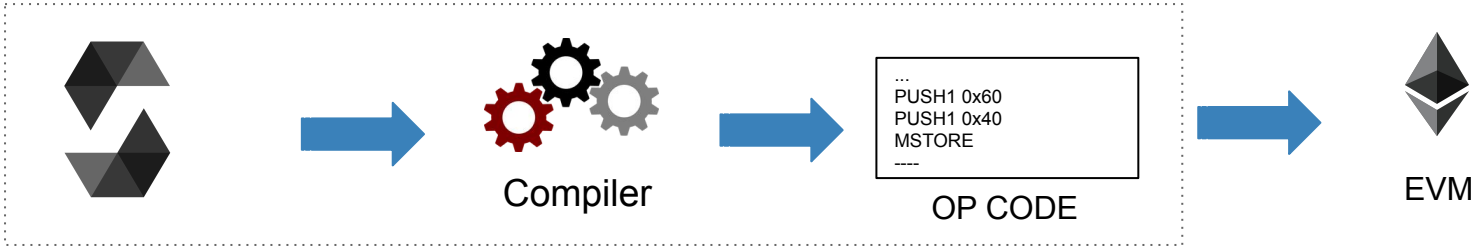
Application
Binary
Interface


```
[{ "constant": true,
   "inputs": [],
   "name": "total",
   "outputs": [{
         "name": "",
         "type": "uint256"
    }],
    "payable": false,
   "type": "function"
   },{
     "constant": false,
     "inputs": [],
     "name": "add",
     "outputs": [],
     "payable": false,
     "type": "function"
   }]
```

Choosing a language

- **Solidity**: similar to JavaScript
- **Serpent**: similar to Python
- **LLL**: similar to Assembly
- **Mutan**: similar to C

Solidity



- Solidity is a contract-oriented, high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum Virtual Machine (EVM).
- Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

Solidity

- Layout of a Solidity Source File
- Types
- Variable
- Function
- Globally Available Variables
- Solidity Assembly

Solidity: Source file example

Contracts in Solidity are similar to classes in object-oriented languages.

```
pragma solidity ^0.4.4; // Version prama
import './coin3.sol'; // Import from code from another file

// Declare a contract
contract Coin1 {

}
// Declare a contract
contract Coin2 is Coin1 {

}
```

Solidity: Source file example

```
pragma solidity ^0.4.0;
contract Token {
   uint256 public totalSupply;
   uint256 public availableNumber;
   mapping (address => uint256) balances;
   address owner;
   uint price = 100; // 1 ETH = 100 Token
   event Transfer(address indexed _from, address indexed _to, uint256 _amount);
   function Token(uint256 _totalSupply) public {
       totalSupply = _totalSupply;
       availableNumber = _totalSupply;
       // Set owner is the person who deploy this contract
       owner = msg.sender;
   }
   function () payable public {
       // Calculate amount of token
       uint256 amount = msg.value * price / 1 ether;
       ……
```

Solidity: *Types*

- Booleans
- Integers
- **Address**
- Fixed-size byte arrays
- Dynamically-sized byte array
- Enums
- Arrays
- Structs
- **Mappings**

Solidity: *Types*

Address properties

- <address>.balance (uint256): get balance of the Address in Wei
- <address>.transfer(uint256 amount): send given amount of Wei to Address, throws on failure
- <address>.send(uint256 amount) returns (bool): send given amount of Wei to Address, returns false on failure

Solidity: *Types*

Address properties

- <address>.call(...) returns (bool): issue low-level CALL, returns false on failure
- <address>.staticcall(...) returns (bool): issue low-level STATICCALL. (This is basically the same as call, but will revert if the called function modifies the state in any way)
- <address>.delegatecall(...) returns (bool): issue low-level DELEGATECALL, returns false on failure

Solidity: *Types*

Mapping properties

mapping(_KeyType => _ValueType)

- The _KeyType can be any elementary type
- _ValueType can be any type, including mappings
- If a is a mapping, then delete a[x] will delete the value stored at x

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

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

Solidity: *Variable*

Three places to store a variable

- Storage: store in “blockchain” → expensive.
- Memory: hold temporary values → cheap.
- Stack: hold small local variables → limited (16 slots).

```
contract test3 {
        uint public constant TOTAL=100;
        uint[] public a;
        uint[] private b;
        function push() public {
                uint[] memory x = a;
                uint[] storage y = b;
        }
}
```

Solidity: *Function*

function <name>([params])[modifier][visibility][returns type]

- Modifier: Change the behaviour of functions.
  - Payable: Able to receive ETH.
- Visibility: internal, external, private, public.
- Returns: Return types.

```
function issue(uint _amount) onlyIssuer(msg.sender) payable public returns(bool, uint) {
    balances[issuer] += _amount;
    return true, balances[issuer];
}
```

Solidity: *Function*

Modifiers can be used to easily change the behaviour of functions.

```
modifier onlyIssuer(address _addr) {
        require(issuer == _addr);
        _;
}

function issue(uint _amount)
    onlyIssuer(msg.sender)
    returns(bool, uint)
{
    balances[issuer] += _amount;
    return true, balances[issuer];
}
```

Solidity: *Function*

Fallback is the unnamed function of a smartcontract which is executed whenever the contract receives plain Ether.

Only 2300 cost gas, so we can not:

- Writing to storage
- Creating a contract
- Calling an external function which consumes a large amount of gas
- Sending Ether

```
pragma solidity ^0.4.4;
contract Token {
    function() payable public {
    }
}
```

Solidity: *Globally Variables*

- block.number (uint): current block number
- msg.sender (address): sender of the message (current call)
- msg.value (uint): number of wei sent with the message
- now (uint): current block timestamp (alias for block.timestamp)
- tx.gasprice (uint): gas price of the transaction
- tx.origin (address): sender of the transaction (full call chain)
- sha256(bytes memory) returns (bytes32): compute the SHA-256 hash of the input
- …….

https://solidity.readthedocs.io/en/develop/units-and-global-variables.html

Solidity: Assembly

Use OPCODE directly in solidity code.

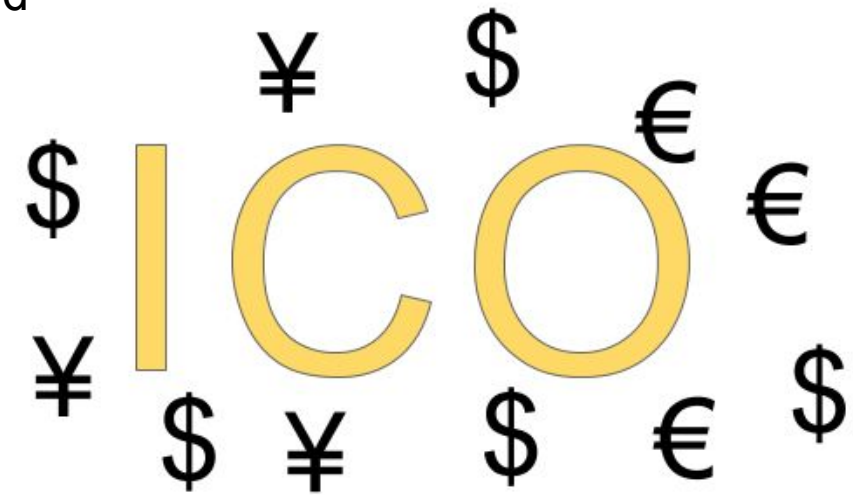
```
function isSmartContract(address _addr) returns(bool) {
        uint size;
        assembly {
            // retrieve the size of the code, this needs assembly
            size := extcodesize(_addr)
        }
        return size > 0;
}
```

http://solidity.readthedocs.io/en/develop/assembly.html

Examples

Create a token system:

- User send Eth to contract to get token
- Pay dividend



Issue token

Claim dividend

Code: https://github.com/phukq/token-example

Tools

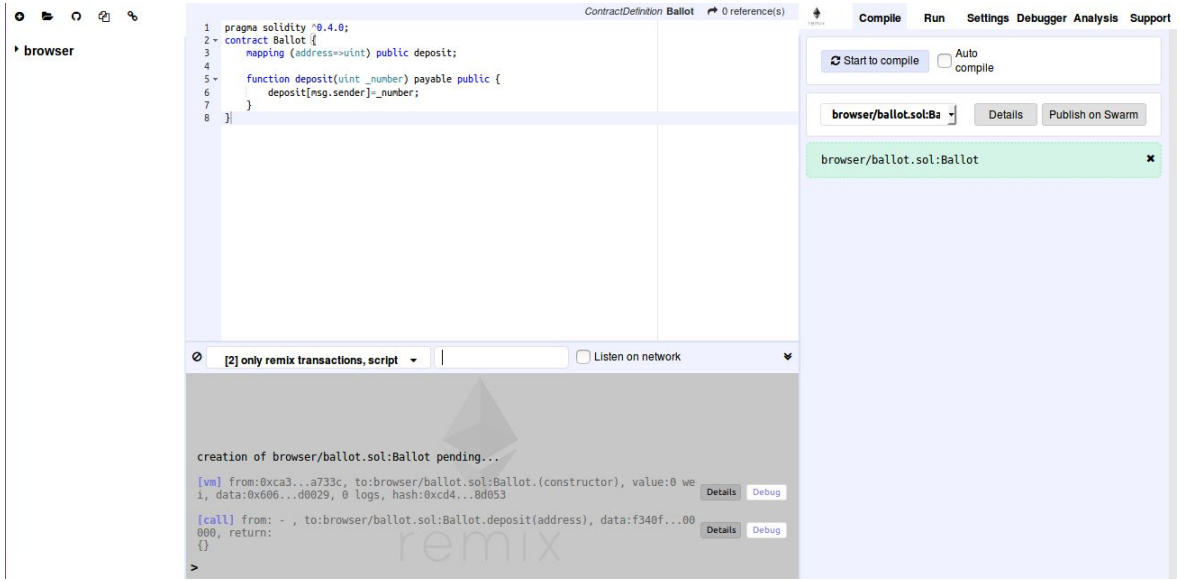
Source code editor

Development framework

Ganache-cli

Remix

- Online code editor
- Online code compiler
- Integrate with network via web3



https://remix.ethereum.org/#version=soljson-v0.4.18+commit.9cf6e910.js

Ganache

```
npm install -g ganache-cli
ganache-cli
```

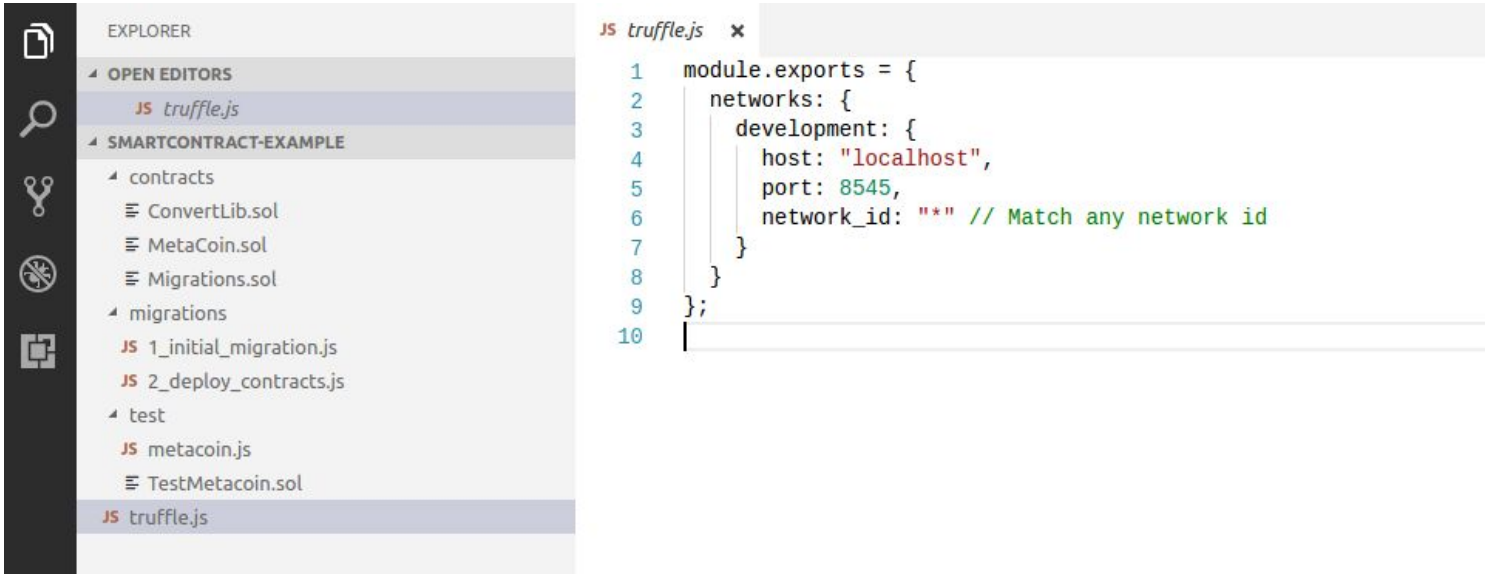
```
phu@phu-ubuntu-3550:~$ testrpc
EthereumJS TestRPC v6.0.1 (ganache-core: 2.0.0)

Available Accounts
==================
(0) 0x4abe547546e25a90dd27b36a74199a125423e666
(1) 0x5dbc4abf391ed54eccdca0111ac3773e99667d1e
(2) 0xf7537727dd1d0764d0d92353a3485bf863d2454c
(3) 0xbdba22185531d385344e98636e598fecea0d78b6
(4) 0x55aa98d7c6e57ece93ae099cb2f9187c52f3beaa
(5) 0x16cae7858ff68eda683fb7f3651ad09f656825dc
(6) 0x2218960213319355 79eeeaf8de7881d3adb4fd60
(7) 0x714941eedbe820f14e22016f447d205e6a5cb1e3
(8) 0x5dedcbfa2e46de85c78296583148aea11e24e498
(9) 0xc5bcfff65afab589d7e2f1bc8dd22fde1845d85a

Private Keys
==================
(0) ce1378a94bdd1863f7266721a5aadec9abfc4f12d3249b20e17ef0fca58c514f
(1) d7623d7d7cf4e85622374a0dc239bef06007d39dd9876cc7f8cb700ceca1146e
(2) b4eff9d2f7f022c0c12c693663b2296a8114abbe9407d4f019fc91bd5fa0f625
(3) 2b164798321d9028b8fbd0659408957ee07c7eaf70ecf8c4bcaae8e4087c5ffa
(4) cbc6d8f78be55abae300bf1d9fc1925315a6dcaf81bb821a980263fc062660f4
(5) 1f514e0721cb93d4519f77b85f83c881fc1545899220740189b931a118fe5fe2
(6) b97a5579b67ed9f7e52521032eee411b077fa5a0ef57cf0aae572b10c2a923a4
(7) e1696c87882c0223aef25868cc20615a9eaeb5308a442139539430ad4b7129db
(8) bf671195e2e6e1426990f93d537cd3bb6b671f1d2d96767dd35df4d69fda42cb
(9) 6c41dfe5c5fda26913f23a0578f148bfc78e9b8a9fa0e7af882e1b4b35a90964
```

Truffle

```
npm install -g truffle
mkdir smartcontract-example
cd smartcontract-example
truffle init
```



Truffle

Project structure

- contracts/ - directory where Truffle expects to find solidity contracts.
- migrations/ - directory to place scriptable deployment files.
- test/ - location of test files for testing your application and contracts.
- truffle.js - your main Truffle configuration file.

Truffle

Compiling contracts

```
truffle compile
```

Running migrations

```
truffle migrate
```

References:

- Noobs image credit: https://medium.com/@ConsenSys/a-101-noob-intro-to-programming-smart-contracts-on-ethereum-695d15c1dab4
- Ethereum yellow paper: https://ethereum.github.io/yellowpaper/paper.pdf
- Ethereum white paper: https://github.com/ethereum/wiki/wiki/White-Paper
- Solidity docs: https://solidity.readthedocs.io/en/latest/
- Truffle docs: https://truffleframework.com/docs

Challenge

Create a p2p game: Two players play with each other. Call the players is A & B. A will give a secret number with an amount of ether, B will guess the number "odd or even". If the B's guess is right B will get all the money.