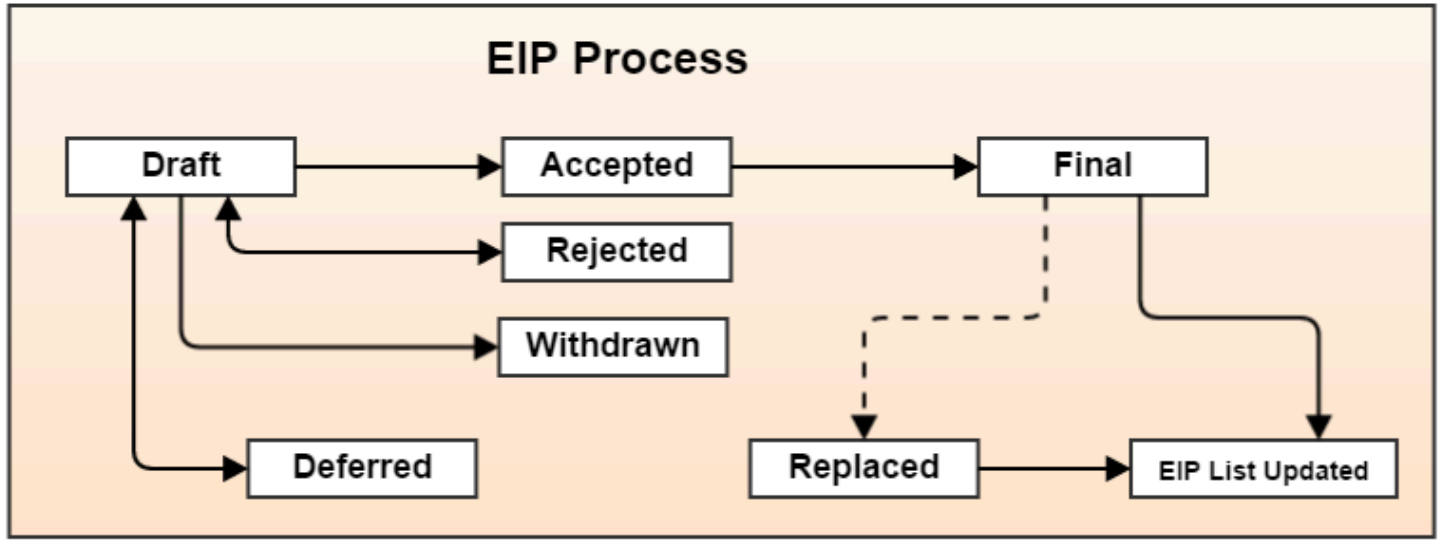
Token model

ERC20/721

Ethereum Improvement Proposals (EIP)

- Standards for the Ethereum platform, including core protocol specifications, client APIs, and contract standards.
  - Standard (134)
    - Core (54)
    - Networking (6)
    - Interface (9)
      ERC (65)
  - Informational (1)
  - Meta (11)



Source: https://github.com/ethereumbook/ethereumbook/blob/develop/appdx-standards-eip-erc.asciidoc

https://eips.ethereum.org/

EIP-155

replay attack protection

chainId
- mainnet: 1
- ropsten: 3

Source: https://medium.com/@codetractio/walkthrough-of-an-ethereum-improvement-proposal-eip-6fda3966d171

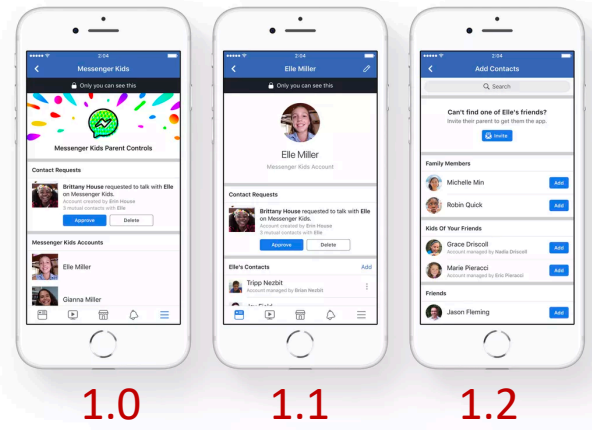
Constantinople fork

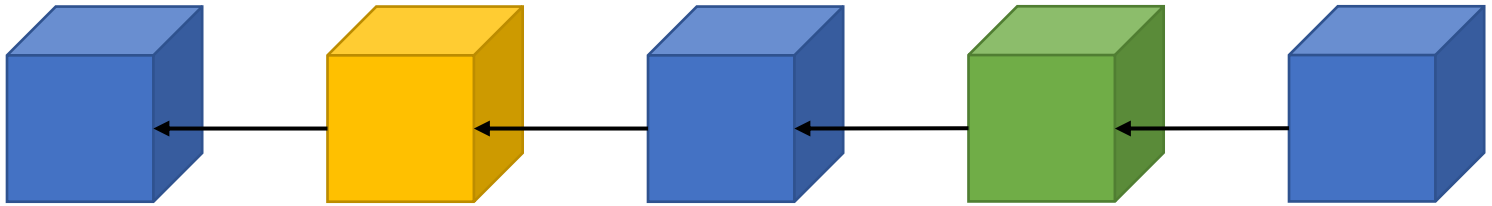
- EIP 145: Bitwise shifting instructions in EVM
- EIP 1014: Skinny CREATE2
- EIP 1052: EXTCODEHASH Opcode
- EIP 1283: Net gas metering for SSTORE without dirty maps
- EIP 1234: Constantinople Difficulty Bomb Delay and Block Reward Adjustment

https://hackmd.io/6mLe_fk6RzCc4VKT0g44lQ

Soft fork

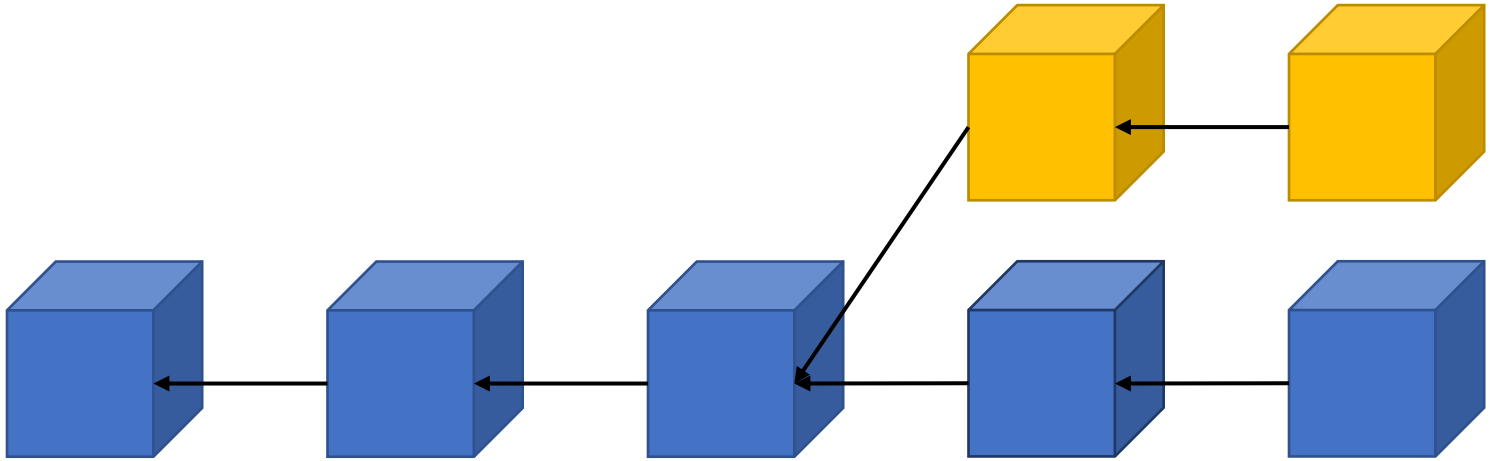
- 系統相容





Hard fork

- 不相容



ERC20 Token

貨幣交易

系統之間不相容，靠台幣做轉換

比特幣
(Bitcoin)

以太幣
(Ether)

萊特幣
(Litecoin)

MaiCoin

MAX

交易所 (Exchange)

kraken

BITFINEX

JOYSO

BINANCE

BitoPro

MAX

加密貨幣

比特幣
(Bitcoin)

以太幣
(Ether)

萊特幣
(Litecoin)

為了支援 N 種不同貨幣，需要維護 N 種不同系統

加密貨幣

比特幣
(Bitcoin)

以太幣
(Ether)

萊特幣
(Litecoin)

當交易所要上新貨幣
工程團隊需要維護一套新系統

系統維護

- 帳號地址
- 升級 or Hard fork
- 管理錢包
- 同步區塊鏈
- 交易手續費
- 交易壅塞處理

重點

- 不相容的貨幣要上交易所或代購平台
- 對工程團隊而言，需要額外一套系統維護
- 負擔
  - BCH-分叉的現狀

Ethereum

- 以太坊的發明，讓人們能簡單的設計應用， 無需建構像比特幣這般複雜的系統，使得發行貨幣變得簡單
- 這種應用透過智能合約來運行
- 而開發者只要在智能合約內，設計應用規則即可

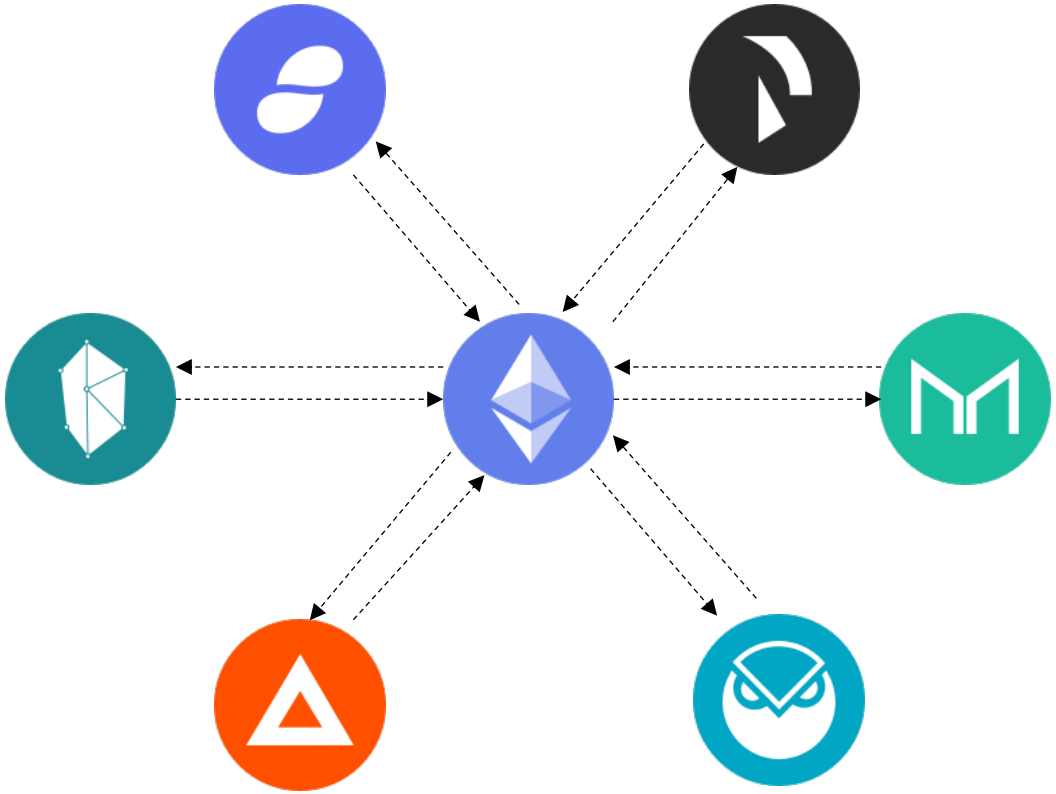
ERC20 Token

- 存在一套標準，讓大家開心發行代幣 (token)

- 想像要自行發行代幣，需要什麼？
  - 代幣的總發行數
  - 代幣的轉帳行為
  - 代幣如何記錄
  - 查詢帳戶餘額
  - 誰能使用代幣 (使用權)

ERC20 Token

- 存在一套標準，讓大家開心發行代幣 (token)



維護一套系統,
但代幣操作統一使用

沒有標準會怎樣？

https://etherscan.io/tokens

ERC20 Token

- 想像要自行發行代幣，需要什麼？
  - 代幣的總發行數
  - 代幣的轉帳行為
  - 代幣如何記錄
  - 查詢帳戶餘額
  - 誰能使用代幣 (使用權)

```
interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address who) external view returns (uint256);
    function allowance(address owner, address spender) external view returns (uint256);
    function transfer(address to, uint256 value) external returns (bool);
    function approve(address spender, uint256 value) external returns (bool);
    function transferFrom(address from, address to, uint256 value) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

https://theethereum.wiki/w/index.php/ERC20_Token_Standard

```
contract ERC20Interface {
    function totalSupply()                          // 1000_000_0000_0000
    function balanceOf(address tokenOwner);
    function allowance(address tokenOwner, address spender);
    function transfer(address to, uint tokens);
    function approve(address spender, uint tokens);
    function transferFrom(address from, address to, uint tokens);
}

string public constant name = "NCCU Token";
string public constant symbol = "NCCU";
uint8 public constant decimals = 5;             // 100.31415
```

```
contract ERC20Interface {
    function totalSupply()                          // 1_0000_0000
    function balanceOf(address tokenOwner);
    function allowance(address tokenOwner, address spender);
    function transfer(address to, uint tokens);
    function approve(address spender, uint tokens);
    function transferFrom(address from, address to, uint tokens);
}

string public constant name = "UCCU Token";
string public constant symbol = "UCCU";
uint8 public constant decimals = 2;             // 100.50
```

重點

- ERC20 Token 是一種標準
- Token 的發行是透過以太坊智能合約
- 流通變得方便
- 對工程團隊而言只要維護一套系統
- 僅有合約規則不同

- 每個合約都是一個小程式，只是處理的事情不同
- 代幣合約，幫你發行代幣處理交易的程式

MethodID

- > web3.sha3("transfer(address,uint256)").substring(0, 10)
  - a9059cbb
- > web3.sha3("updatePrice(uint256,uint256)").substring(0, 10)
  - 82367b2d

https://github.com/OpenZeppelin/openzeppelin-solidity/tree/master/contracts/token

DEMO

https://gist.github.com/changwu-tw/a4f7b8df1490e229efbe45aeaec29379

ERC721

NFT token

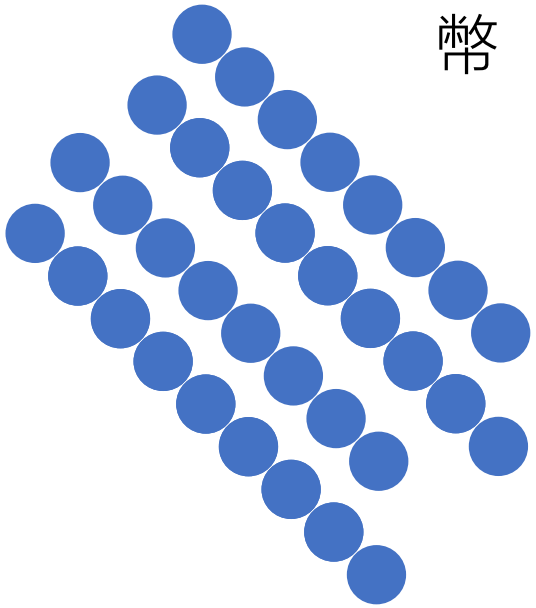
- ERC-721 is a free, open standard that describes how to build non-fungible or unique tokens on the Ethereum blockchain.
- While most tokens are fungible (every token is the same as every other token), ERC-721 tokens are all unique.

http://erc721.org/

Token

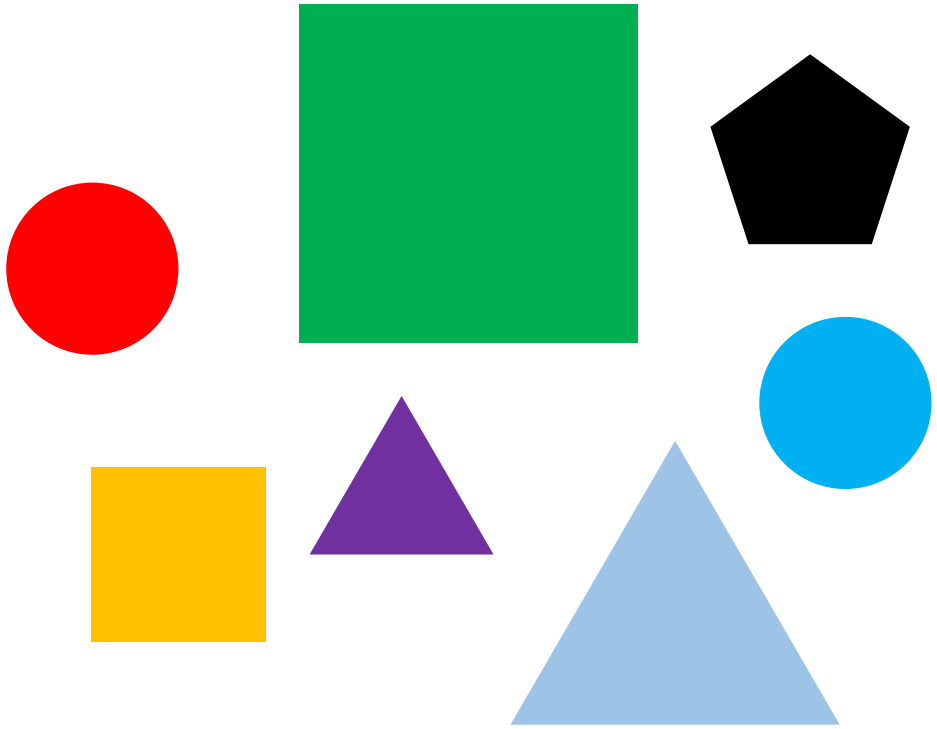
ERC20

幣



ER721

人 / 事 / 物 / 權



Source: https://alphawallet.com/

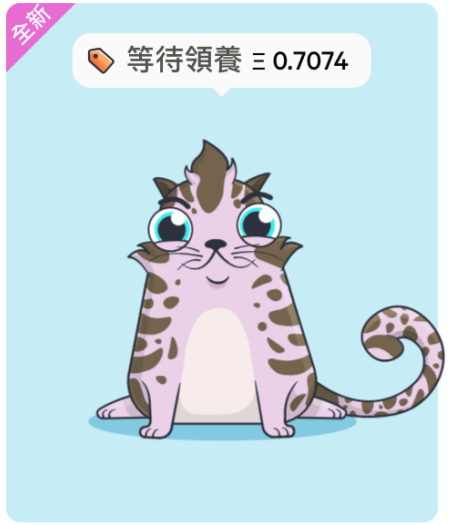
ERC721/ERC875

- 不可分割/不可交換
  - 卡牌
  - 遊戲寶物/道具
  - 票卷
  - 食品溯源 (藥丸, 紅酒, 等等)
  - 房地產

新到貓咪

https://www.cryptokitties.co/

0代貓咪開啟了謎戀貓的一切！其發行數量僅限5萬隻，先到先得！最新喵屬性就隱藏在新出生的0代貓咪中，通過繁殖貓咪可解鎖最新喵變異。



1224200 ♡ 0

0 代 神速

1224109 ♡ 0

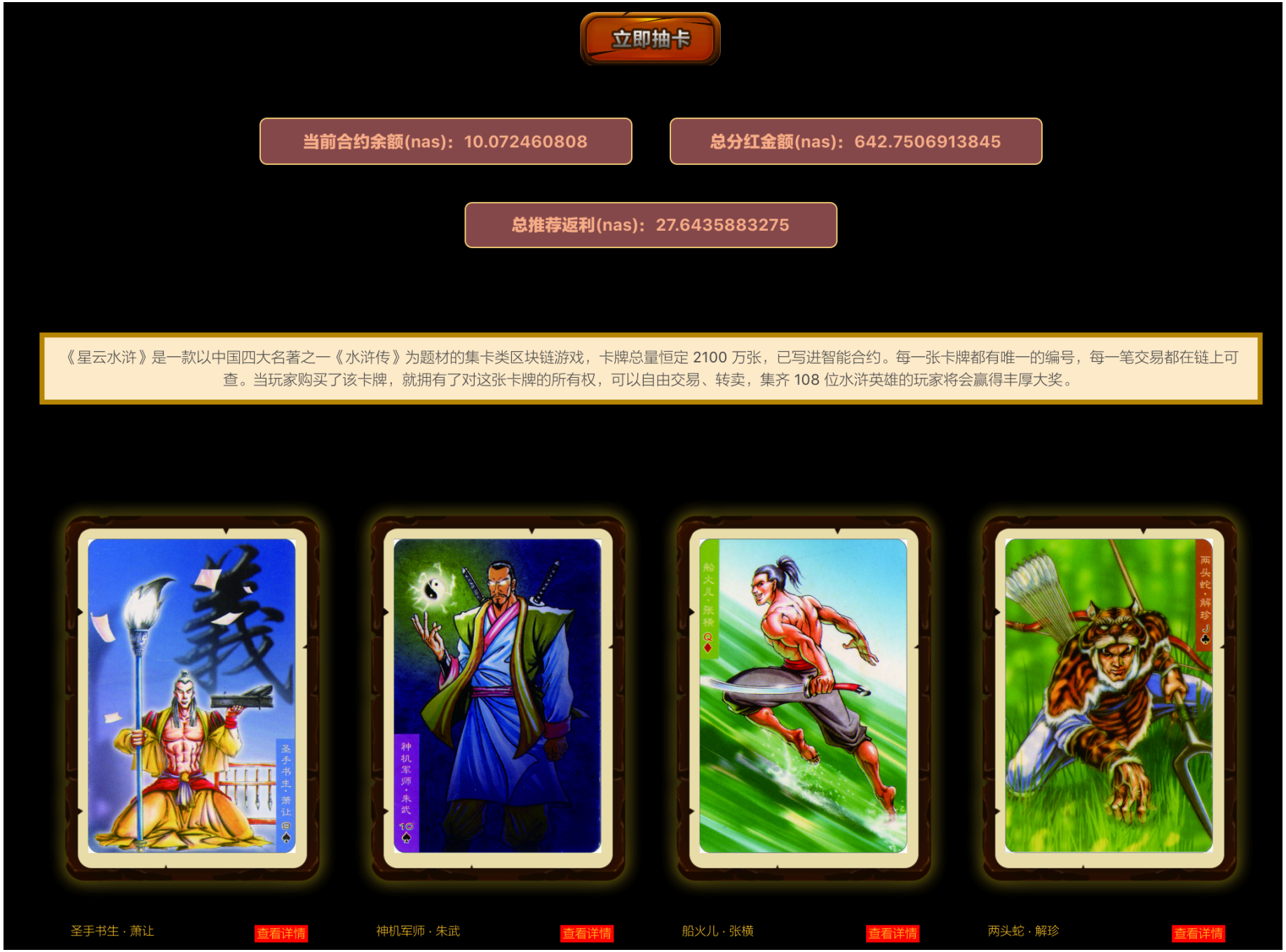
0 代 神速

1224073 ♡ 0

0 代 神速

1224027 ♡ 0

0 代 神速



https://nas.cryptohero.pro/#/

ERC721 interface

```
/// @title ERC-721 Non-Fungible Token Standard
/// @dev See https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md
///  Note: the ERC-165 identifier for this interface is 0x80ac58cd
interface ERC721 /* is ERC165 */ {
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _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);
}

interface ERC165 {
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}

interface ERC721TokenReceiver {
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data) external returns(bytes4);
}
```

Token implementation requirements

- Ownership
  - mapping(address => uint256) balances
  - mapping(address => uint256[]) internal ownedTokens
  - mapping(uint256 => address) internal tokenOwner
- Mint/Creation
- Transfer & Allowance
- Burn

https://medium.com/blockchannel/walking-through-the-erc721-full-implementation-72ad72735f3c

ERC721x: Multi Fungible Tokens

https://cryptozombies.io/en/lesson/9/

```
contract ERC721X {
    function implementsERC721X() public pure returns (bool);
    function ownerOf(uint256 _tokenId) public view returns (address _owner);
    function balanceOf(address owner) public view returns (uint256);
    function balanceOf(address owner, uint256 tokenId) public view returns (uint256);
    function tokensOwned(address owner) public view returns (uint256[], uint256[]);

    function transfer(address to, uint256 tokenId, uint256 quantity) public;
    function transferFrom(address from, address to, uint256 tokenId, uint256 quantity) public;

    // Fungible Safe Transfer From
    function safeTransferFrom(address from, address to, uint256 tokenId, uint256 _amount) public;
    function safeTransferFrom(address from, address to, uint256 tokenId, uint256 _amount, bytes data) public;

    // Batch Safe Transfer From
    function safeBatchTransferFrom(address _from, address _to, uint256[] tokenIds, uint256[] _amounts, bytes _data) public;

    function name() external view returns (string);
    function symbol() external view returns (string);

    // Required Events
    event TransferWithQuantity(address indexed from, address indexed to, uint256 indexed tokenId, uint256 quantity);
    event TransferToken(address indexed from, address indexed to, uint256 indexed tokenId, uint256 quantity);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
    event BatchTransfer(address indexed from, address indexed to, uint256[] tokenTypes, uint256[] amounts);
}
```

https://github.com/loomnetwork/erc721x

ERC721X

- ERC721x is an extension of ERC721 that adds support for multi-fungible tokens and batch transfers, while being fully backward-compatible.
  - Loom network
    - 遊戲道具限量
    - 道具不限量

New a contract (1/2)

Let's start with the basics: we'll create a new contract called "ZombieCard" that inherits the logic from `ERC721XToken`.

1. First, declare at the top that we're using `pragma solidity ^0.4.25`.
2. After declaring the pragma, `import` the file `./ERC721XToken.sol`
3. Next, declare a new `contract` named "ZombieCard". It should inherit from `ERC721XToken` using the keyword `is`. Leave the body of the contract empty for now.

https://cryptozombies.io/en/lesson/9

New a contract (2/2)

Let's start with the basics: we'll create a new contract called "ZombieCard" that inherits the logic from `ERC721XToken`.

1. First, declare at the top that we're using `pragma solidity ^0.4.25`.
2. After declaring the pragma, `import` the file `./ERC721XToken.sol`
3. Next, declare a new `contract` named "ZombieCard". It should inherit from `ERC721XToken` using the keyword `is`. Leave the body of the contract empty for now.

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol";

contract ZombieCard is ERC721XToken {

}
```

https://cryptozombies.io/en/lesson/9

Name, Symbol (1/2)

Let's go ahead and implement both of these functions in our ZombieCard.

1. Create a function called `name`. It will be an `external view` function, and return a `string`- "ZombieCard".
2. Create a function named `symbol`. It will be an `external view` function, and return a `string`- "ZCX".

https://cryptozombies.io/en/lesson/9

Name, Symbol (2/2)

Let's go ahead and implement both of these functions in our ZombieCard.

1. Create a function called `name`. It will be an `external view` function, and return a `string`- "ZombieCard".
2. Create a function named `symbol`. It will be an `external view` function, and return a `string`- "ZCX".

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol";

contract ZombieCard is ERC721XToken {

    function name() external view returns (string) {
        return "ZombieCard";
    }

    function symbol() external view returns (string) {
         return "ZCX";
    }
}
```

https://cryptozombies.io/en/lesson/9

Supply (1/2)

1. Create a `mapping` called `tokenIdToIndividualSupply`. It will map a `uint` to `uint`, and should be `internal`.
2. Create a `function` called `individualSupply`. It should be `public view`, take `uint _tokenId` as its input, and it `returns` a `uint`.
3. For the body of the function, it should simply `return tokenIdToIndividualSupply(_tokenId)`.

https://cryptozombies.io/en/lesson/9

Supply (2/2)

1. Create a `mapping` called `tokenIdToIndividualSupply`. It will map a `uint` to `uint`, and should be `internal`.
2. Create a `function` called `individualSupply`. It should be `public view`, take `uint _tokenId` as its input, and it `returns` a `uint`.
3. For the body of the function, it should simply `return tokenIdToIndividualSupply(_tokenId)`.

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol";

contract ZombieCard is ERC721XToken {

    mapping(uint => uint) internal tokenIdToIndividualSupply;

    function name() external view returns (string) {
        return "ZombieCard";
    }

    function symbol() external view returns (string) {
        return "ZCX";
    }

    function individualSupply(uint _tokenId) public view returns (uint) {
        return tokenIdToIndividualSupply(_tokenId);
    }
}
```

https://cryptozombies.io/en/lesson/9

mintToken (1/2)

1. Create a function called `mintToken`. It should take 2 arguments, `uint _tokenId`, and `uint _supply`. It should be `public`, and have the `onlyOwner` modifier.

2. Part of Limited Edition cards being rare means they should be guaranteed to be limited in quantity. That means we should only be able to mint a token once, and we should never be able to increase its supply again.

   We can enforce this with a `require` statement, which will exit with an error if the token already exists. You can do this by calling a function we've defined in `ERC721XToken.sol` with the line of code: `require(!exists(_tokenId), "Error: Tried to mint duplicate token id");`

   Make this the first line of code in our function.

3. Next, we want to call the internal `_mint` function, as in the example above. We will pass it 3 arguments, `_tokenId, msg.sender, _supply`.

4. Lastly, we'll need to update our mapping `tokenIdToIndividualSupply` to store the supply of this token. Update the mapping to set the value for `_tokenId` to `_supply`.

And that's it! Now we have a public `mintToken` function that the owner of the contract (our game) can call to create new cards.

https://cryptozombies.io/en/lesson/9

mintToken (2/2)

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol"; import "./Ownable.sol";

contract ZombieCard is ERC721XToken {

    mapping(uint => uint) internal tokenIdToIndividualSupply;

    function name() external view returns (string) {
        return "ZombieCard";
    }

    function symbol() external view returns (string) {
        return "ZCX";
    }

    function individualSupply(uint _tokenId) public view returns (uint) {
        return tokenIdToIndividualSupply(_tokenId);
    }

    function mintToken(uint _tokenId, uint _supply) public onlyOwner {
        require(!exists(_tokenId), "Error: Tried to mint duplicate token id");
        _mint(_tokenId, msg.sender, _supply);
        tokenIdToIndividualSupply[_tokenId] = _supply;
    }
}
```

1. Create a function called `mintToken`. It should take 2 arguments, `uint _tokenId`, and `uint _supply`. It should be `public`, and have the `onlyOwner` modifier.

2. Part of Limited Edition cards being rare means they should be guaranteed to be limited in quantity. That means we should only be able to mint a token once, and we should never be able to increase its supply again.

   We can enforce this with a `require` statement, which will exit with an error if the token already exists. You can do this by calling a function we've defined in `ERC721XToken.sol` with the line of code: `require(!exists(_tokenId), "Error: Tried to mint duplicate token id");`

   Make this the first line of code in our function.

3. Next, we want to call the internal `_mint` function, as in the example above. We will pass it 3 arguments, `_tokenId, msg.sender, _supply`.

4. Lastly, we'll need to update our mapping `tokenIdToIndividualSupply` to store the supply of this token. Update the mapping to set the value for `_tokenId` to `_supply`.

And that's it! Now we have a public `mintToken` function that the owner of the contract (our game) can call to create new cards.

https://cryptozombies.io/en/lesson/9

awardToken (1/2)

Let's create our `awardToken` function.

1. First, we'll want to define an `event` called `TokenAwarded` at the start of our contract that will take 3 parameters: `uint indexed tokenId, address claimer, uint amount`.

2. Next, at the bottom of our contract, define a `function` named `awardToken` that takes 3 parameters: `uint _tokenId, address _to, uint _amount`. It should be `public` and `onlyOwner`.

3. The first thing this function should do is check to make sure this token exists. We can do this with a `require` statement. The code should check `exists(_tokenId)`, and if not it should exit with an error: `"TokenID has not been minted"`.

We will continue the implementation of this function in the next chapter.

https://cryptozombies.io/en/lesson/9

awardToken (2/2)

https://cryptozombies.io/en/lesson/9

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol";
import "./Ownable.sol";

contract ZombieCard is ERC721XToken {
    mapping(uint => uint) internal tokenIdToIndividualSupply;

    event TokenAwarded(uint indexed tokenId, address claimer, uint amount);

    function name() external view returns (string) {
        return "ZombieCard";
    }

    function symbol() external view returns (string) {
        return "ZCX";
    }

    function individualSupply(uint _tokenId) public view returns (uint) {
        return tokenIdToIndividualSupply(_tokenId);
    }

    function mintToken(uint _tokenId, uint _supply) public onlyOwner {
        require(!exists(_tokenId), "Error: Tried to mint duplicate token id");
        _mint(_tokenId, msg.sender, _supply);
        tokenIdToIndividualSupply[_tokenId] = _supply;
    }

    function awardToken(uint _tokenId, address _to, uint _amount) public onlyOwner {
        require(exists(_tokenId), "TokenID has not been minted");
    }
}
```

Let's create our `awardToken` function.

1. First, we'll want to define an `event` called `TokenAwarded` at the start of our contract that will take 3 parameters: `uint indexed tokenId, address claimer, uint amount`.
2. Next, at the bottom of our contract, define a `function` named `awardToken` that takes 3 parameters: `uint _tokenId, address _to, uint _amount`. It should be `public` and `onlyOwner`.
3. The first thing this function should do is check to make sure this token exists. We can do this with a `require` statement. The code should check `exists(_tokenId)`, and if not it should exit with an error: `"TokenID has not been minted"`.

We will continue the implementation of this function in the next chapter.

updateTokenBalance (1/2)

Continuing our implementation...

1. Create an `if` statement that checks if `individualSupply[_tokenId]` is greater than `0`. If this condition is true, that means this token is a Limited Edition card with a fixed supply.

2. If the `if` statement is true, we want to first make sure the game server has enough of this card left to send to the user and throw an error otherwise. We can check this with the line: `require(_amount <= balanceOf(msg.sender, _tokenId), "Quantity greater than remaining cards");`

3. Still inside the `if` statement, now let's reduce the game server (`msg.sender`)'s balance of `_tokenId` by `_amount`. Reference the sample code above in `_transferFrom()` to see how to call `_updateTokenBalance()`.

4. After the `if` statement, now we want to update the token balance of the `_to` address to increase it by `_amount`. This goes outside the `if` statement, because we want to do this both for Limited Edition cards and Standard Edition cards.

   You can reference the sample code above to see how to call this.

5. Lastly, we'll want to fire the event we created, with `emit TokenAwarded(_tokenId, _to, _amount);`.

https://cryptozombies.io/en/lesson/9

https://cryptozombies.io/en/lesson/9

updateTokenBalance (2/2)

```
pragma solidity ^0.4.25;

import "./ERC721XToken.sol";
import "./Ownable.sol";

contract ZombieCard is ERC721XToken {
    mapping(uint => uint) internal tokenIdToIndividualSupply;

    event TokenAwarded(uint indexed tokenId, address claimer, uint amount);
    ……

    function mintToken(uint _tokenId, uint _supply) public onlyOwner {
        require(!exists(_tokenId), "Error: Tried to mint duplicate token id");
        _mint(_tokenId, msg.sender, _supply);
        tokenIdToIndividualSupply[_tokenId] = _supply;
    }

    function awardToken(uint _tokenId, address _to, uint _amount) public onlyOwner {
        require(exists(_tokenId), "TokenID has not been minted");
        if (individualSupply[_tokenId] > 0) {
            require(_amount <= balanceOf(msg.sender, _tokenId), "Quantity greater than remaining cards");
            _updateTokenBalance(msg.sender, _tokenId, _amount, ObjectLib.Operations.SUB);
        }
        _updateTokenBalance(_to, _tokenId, _amount, ObjectLib.Operations.ADD);
        emit TokenAwarded(_tokenId, _to, _amount);
    }
}
```

Continuing our implementation...

1. Create an `if` statement that checks if `individualSupply[_tokenId]` is greater than `0`. If this condition is true, that means this token is a Limited Edition card with a fixed supply.

2. If the `if` statement is true, we want to first make sure the game server has enough of this card left to send to the user and throw an error otherwise. We can check this with the line: `require(_amount <= balanceOf(msg.sender, _tokenId), "Quantity greater than remaining cards");`

3. Still inside the `if` statement, now let's reduce the game server (`msg.sender`)'s balance of `_tokenId` by `_amount`. Reference the sample code above in `_transferFrom()` to see how to call `_updateTokenBalance()`.

4. After the `if` statement, now we want to update the token balance of the `_to` address to increase it by `_amount`. This goes outside the `if` statement, because we want to do this both for Limited Edition cards and Standard Edition cards.

   You can reference the sample code above to see how to call this.

5. Lastly, we'll want to fire the event we created, with `emit TokenAwarded(_tokenId, _to, _amount);`.

Cryptozombies

- https://cryptozombies.io/en/lesson/1

代幣類型

- Payment token
- Utility token
  - ERC20/ERC721
- Security token
  - https://thesecuritytokenstandard.org/
  - ERC 1400 - Security Token Standard
  - ERC 1410 - Partially Fungible Token Standard
  - Polymath’s ST20 token
  - Harbor’s r-token (ERC20)

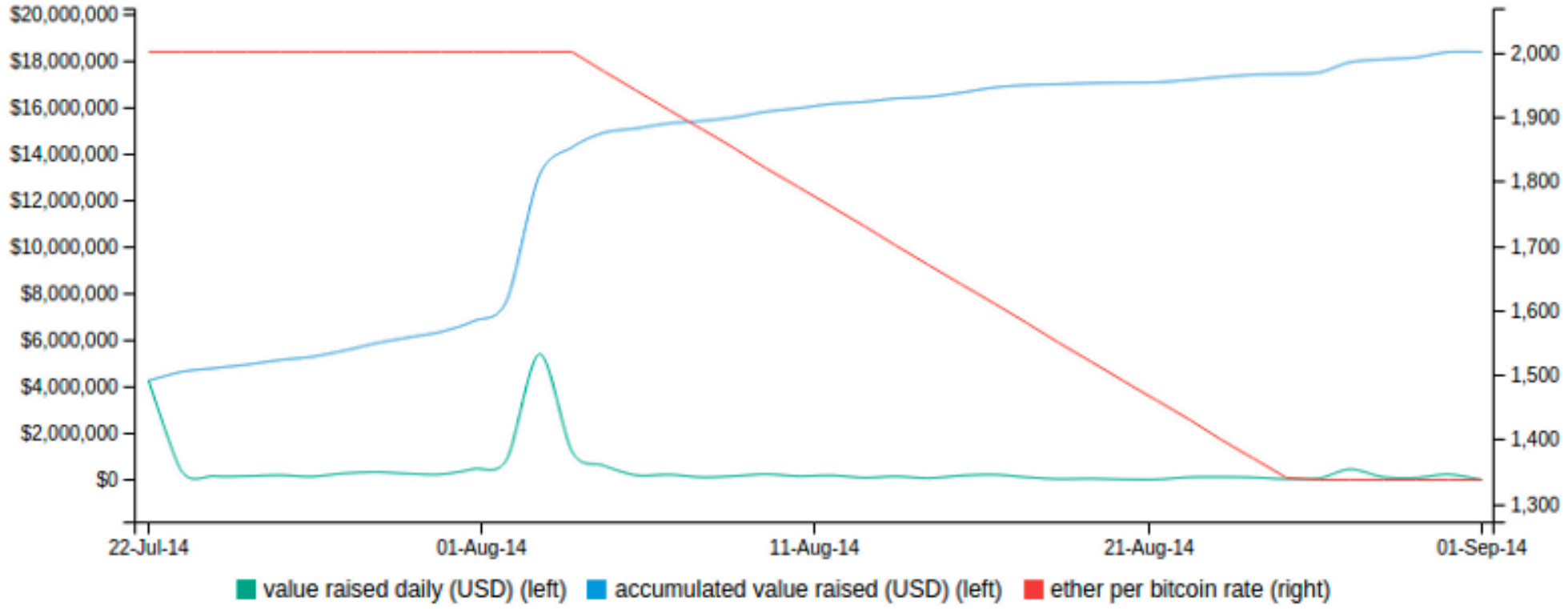
Token sale model

Maidsafe

- BTC
- MSC

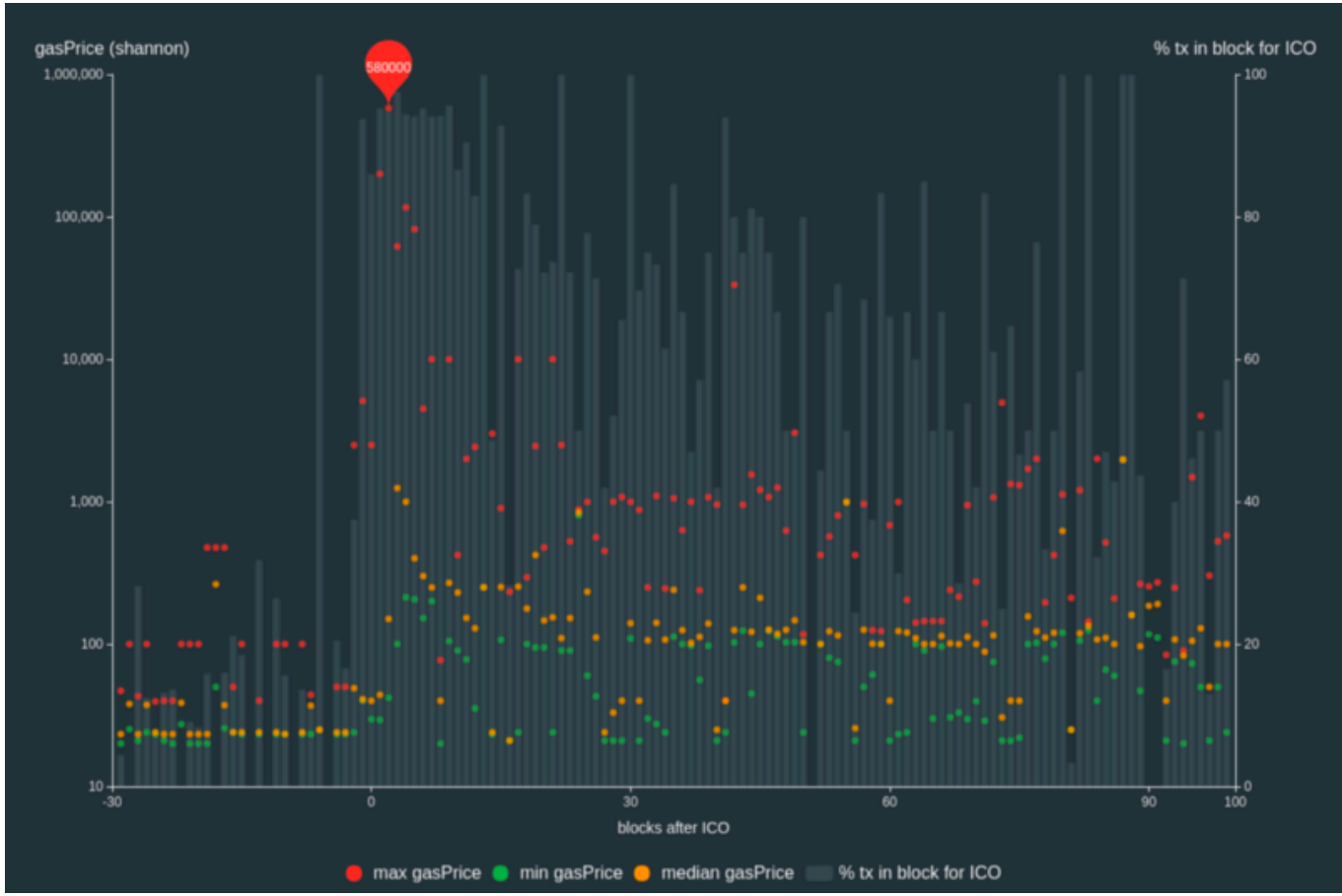
Ethereum

- Uncapped Sale



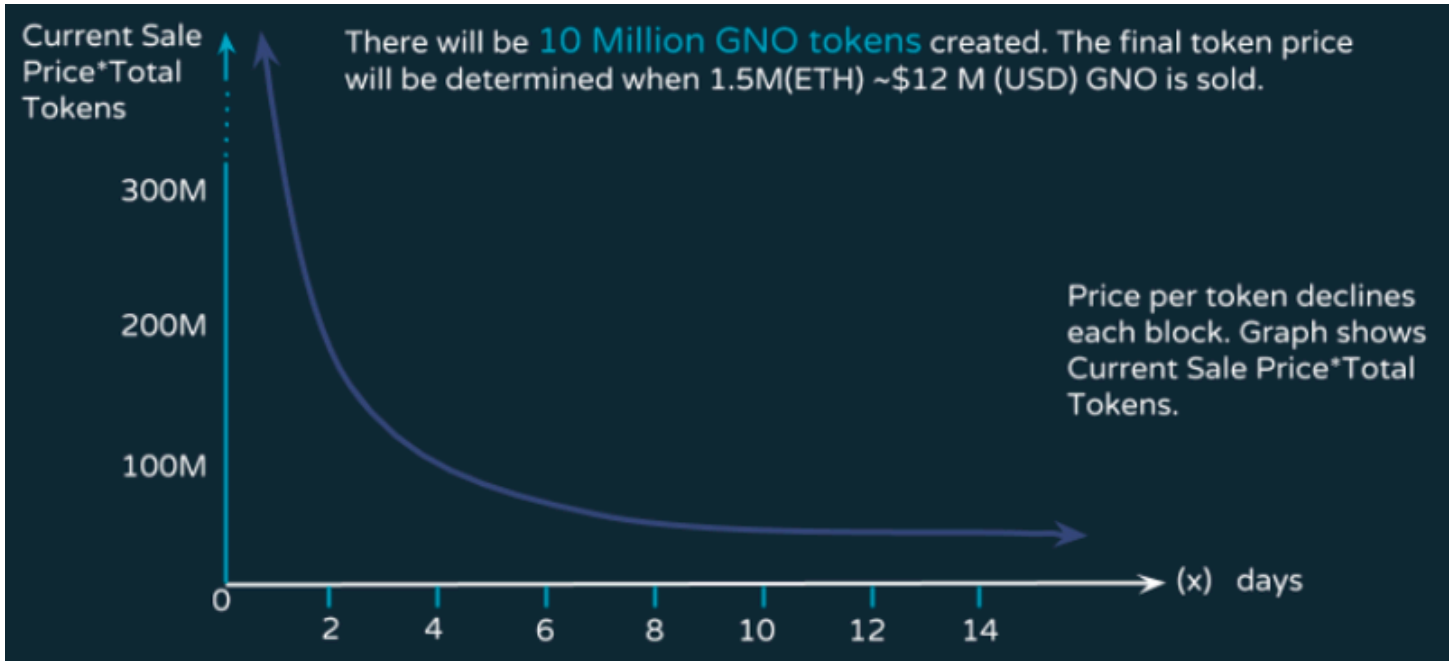
BAT

- Capped sale
- $35m sale within 30 seconds
- The total transaction fees: 70.15 ETH
  - Highest single fee being $6600
- 185 purchases were successful, and over 10000 failed
- The Ethereum blockchain’s capacity was full for 3 hours after the sale started

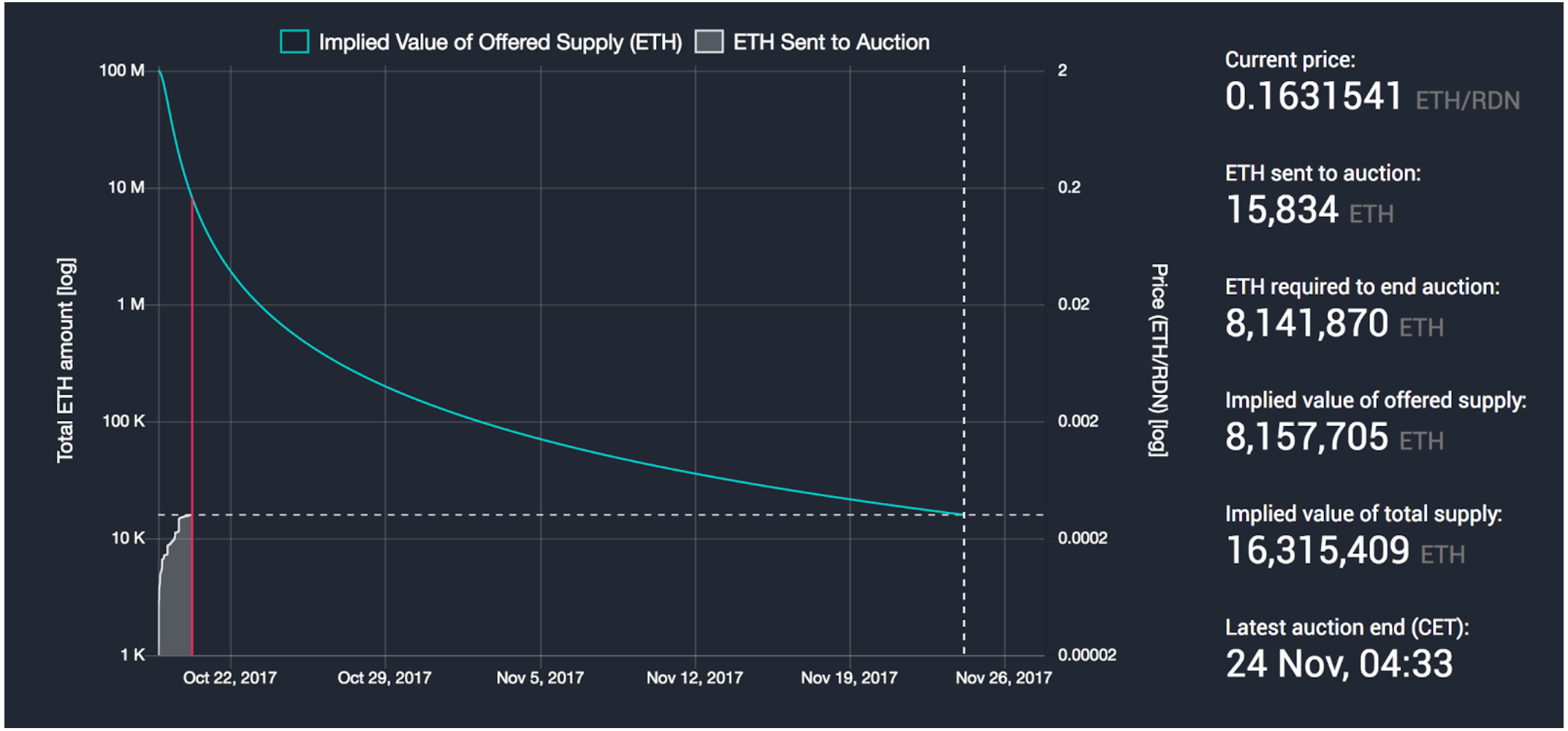


Gnosis

- Capped sale
- The reverse dutch auction
- Fear of missing out (FOMO)
- The sale reached its cap of $12.5 million when it was only selling about 5% of all tokens



Raiden


Implied Value of Offered Supply (ETH)
ETH Sent to Auction
100 M
10 M
1 M
100 K
10 K
1 K
Total ETH amount [log]
2
0.2
0.02
0.002
0.0002
0.00002
Price (ETH/RDN) [log]
Oct 22, 2017
Oct 29, 2017
Nov 5, 2017
Nov 12, 2017
Nov 19, 2017
Nov 26, 2017
Current price:
0.1631541 ETH/RDN
ETH sent to auction:
15,834 ETH
ETH required to end auction:
8,141,870 ETH
Implied value of offered supply:
8,157,705 ETH
Implied value of total supply:
16,315,409 ETH
Latest auction end (CET):
24 Nov, 04:33

So what would a good token sale mechanism look like?

- Certainty of valuation
- Certainty of participation
- Capping the amount raised
- No central banking
- Efficiency

Binance Lab

- https://docs.google.com/document/d/15saZYFh38jI248nzqqyH6PqfQ8YeY4E9qP0yXshKg3o/edit?fbclid=IwAR2bdR8P8oe8-1Zhqq739pdfVP4yZVNz8y0GrU3r6dhfR-RZ6Zjls5N_IRE