KAIST
School of Computing

Web3@KAIST
Building Web3 Apps to Solve Real Problems

Building Web3 & Blockchain Applications
(CS492 Special Topics in Computer Science)
Spring 2023

Developing ERC20 fungible assets

Lecture 15 (2023-05-03)

Jason Han, Ph.D

Adjunct Professor of KAIST School of Computing

Founder of Ground X & Klaytn

web3classdao@gmail.com

http://web3classdao.xyz/kaist/

Today's Lecture 15 Overview

- **Lecture Objective**
  - Learning how to mint fungible tokens and get eth for funding
  - Understanding the concept of oracle and how to use Chainlink
  - Understanding ERC20 and how to create ERC20 tokens
  - Learning various ways to interact between contracts

- **Lecture will cover**
  - Depoit and withdraw of eth
  - Library, inheritance, interaction between contracts
  - Oracle and chainlink data feed
  - ERC20 standard and how to implement ERC20 tokens
  - Interacting with ERC20 tokens

References for the lecture

- Ultimate Web3, Full Stack Solidity, and Smart Contract Course by Patrick Collins
  - Lesson 4: Remix Fund Me
  - Lesson 12: Hardhat ERC20s
- Chainlink Presentation
- Chainlink tutorial: consuming data feeds
- Solidity Library by Jean Cvllr
- What is ERC-20? by thirdweb
- Ethereum EIP-20
- OpenZeppelin ERC20 docs
- OpenZeppelin ERC20 codes

A simple crowdfunding contracts

Examples from Patrick Collins' web3 course
with some modification

Clone the code here!
git clone https://github.com/web3classdao/fungible-tokens.git

Recap: Token example in Lecture 10

- *Implemented a simple token contract*
- *Minted all tokens to the contract owner*
- *Others should get tokens from a faucet*

Recap: Token example code

Token.sol

```solidity
1  //SPDX-License-Identifier: UNLICENSED
2  pragma solidity ^0.8.9;
3
4  import "hardhat/console.sol";
5
6  contract Token {
7
8      string public name = "My Test Token";
9      string public symbol = "MTT";
10
11     uint256 public totalSupply = 1000000;
12
13     // The address of contract owner
14     address public owner;
15
16     // Store each account's balance.
17     mapping(address => uint256) balances;
18
19     // Event when token transfer happens
20     event Transfer(address indexed _from, address indexed _to, uint256 _value);
21
22     constructor() {
23         // The totalSupply is assigned to the transaction sender, which is the
24         // account that is deploying the contract.
25         balances[msg.sender] = totalSupply;
26         owner = msg.sender;
27     }
```

a main data store for tokens (line 17)

Minting tokens initially (lines 25–26)

```solidity
29     function transfer(address to, uint256 amount) external {
30         // Check if the transaction sender has enough tokens.
31         require(balances[msg.sender] >= amount, "Not enough tokens");
32
33         console.log(
34             "Transferring from %s to %s %s tokens",
35             msg.sender,
36             to,
37             amount
38         );
39
40         // Transfer the amount.
41         balances[msg.sender] -= amount;
42         balances[to] += amount;
43
44         // Notify the transfer.
45         emit Transfer(msg.sender, to, amount);
46     }
47
48     // retrieve the token balance of a given account.
49     function balanceOf(address account) external view returns (uint256) {
50         return balances[account];
51     }
52 }
```

Token transfer is just all about increasing and decreasing balances as an atomic operation (lines 40–42)

Improvement 1. Getting fund with eth

Added features

1) Deposit fund with eth and return equivalent tokens
2) Withdraw funded eth and reset funders' data
3) Implement receive() and fallback()

1) Deposit fund

TokenFundEth.sol

funder data (Optional)

payable
let the function accept eth

msg.value
number of wei sent with the message

adjusting balances
automically

Ether units
wei is the basic unit

*1 gwei = 10 ** 9 wei*
*1 ether = 10 ** 18 wei*

```
44    // Store each funder's amount funded and address.
45    mapping(address => uint256) public addressToAmountFunded;
46    address[] public funders;
47    uint256 public constant MINIMUM_FUND = 0.001 * 10 ** 18;      // 0.001 ETH
48    uint256 public constant EXCHANGE_RATE = 1000;                 // 1 ETH = 1000 MTT
49
50    function fund() public payable {
51        // Add funder address and eth amount funded
52        require(msg.value >= MINIMUM_FUND, "The minimum fund is 0.001 ETH");
53        addressToAmountFunded[msg.sender] += msg.value;
54        funders.push(msg.sender);
55
56        // Transfer the amount of token equivalent to funded eth
57        uint256 tokenAmount = getTokenAmountInETH(msg.value);
58        require(balances[owner] >= tokenAmount, "We don't have enough tokens");
59        balances[owner] -= tokenAmount;
60        balances[msg.sender] += tokenAmount;
61
62        console.log(
63            "Funding %s wei from %s and transfering %s tokens",
64            msg.value,
65            msg.sender,
66            tokenAmount
67        );
68
69        // Notify the funding.
70        emit Funding(msg.sender, msg.value);
71    }
72
73    function getTokenAmountInETH(uint256 ethAmount) internal pure returns (uint256) {
74        return uint256(ethAmount * EXCHANGE_RATE / 10 ** 18);
75    }
```

2) Withdraw fund

```
64      modifier onlyOwner {
65          require(msg.sender == owner, "You're not the owner");
66          _;
67      }
68
69      function withdraw() public onlyOwner {
70          // Reset funders' data
71          for (uint256 funderIndex=0; funderIndex < funders.length; funderIndex++){
72              address funder = funders[funderIndex];
73              addressToAmountFunded[funder] = 0;
74          }
75          funders = new address[](0);
76
77          // Send eth to msg.sender (owner)
78          uint256 ethAmount = address(this).balance;
79          (bool callSuccess, ) = payable(msg.sender).call{value: ethAmount}("");
80          require(callSuccess, "Withdraw Call failed");
81
82          // Notify the withdraw.
83          emit Funding(msg.sender, ethAmount);
84      }
```

modifier
only owner can run withdraw()

contract's eth balance

Sending eth
call is the recommended way

3 ways to send eth

- *transfer (2300 gas limit, throws error)*
- *send (2300 gas limit, returns bool)*
- *call (forward all gas or set gas, returns bool)*

※2300 gas limit is hardcoded to prevent reentrancy attacts
call is the recommended way in combination with re-entrancy guard

```
// transfer
payable(msg.sender).transfer(address(this).balance);

// send
bool sendSuccess = payable(msg.sender).send(address(this).balance);
require(sendSuccess, "Send failed");
```

https://solidity-by-example.org/sending-ether/

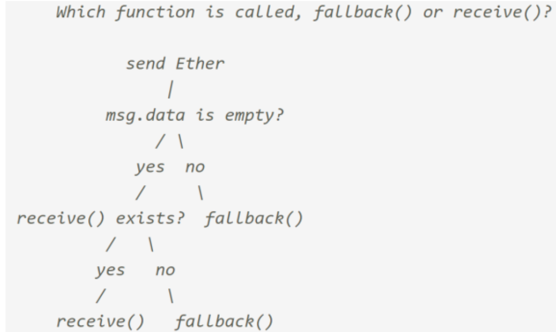
3) receive() & fallback()

What if someone send eth to a contract without calling a function of the contract?

receive() and fallback()
special functions that is executed either when
1) *a function that does not exist is called or*
2) *Ether is sent directly to a contract*

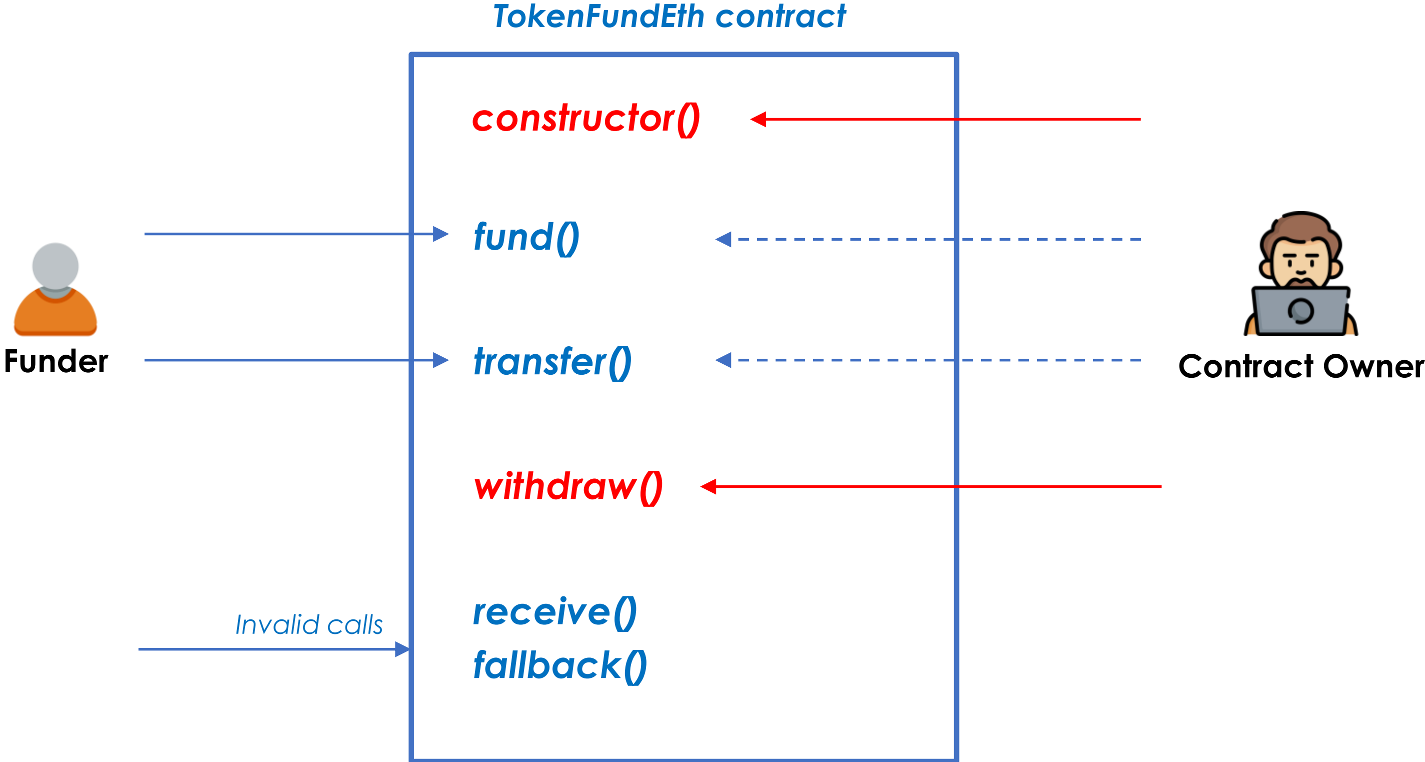
```
fallback() external payable {
    fund();
}

receive() external payable {
    fund();
}
```



https://solidity-by-example.org/sending-ether/
https://solidity-by-example.org/fallback/

Context to call functions



Improvement 2. Funding with eth equivalent to USD

Fix the token price to the dollar, (e.g., 1 MTT = 1 USD)
returning as many tokens as
the current USD value of eth received.

Added features

1) Get a price feed of ETH/USD from the chainlink contract
2) Call a library function

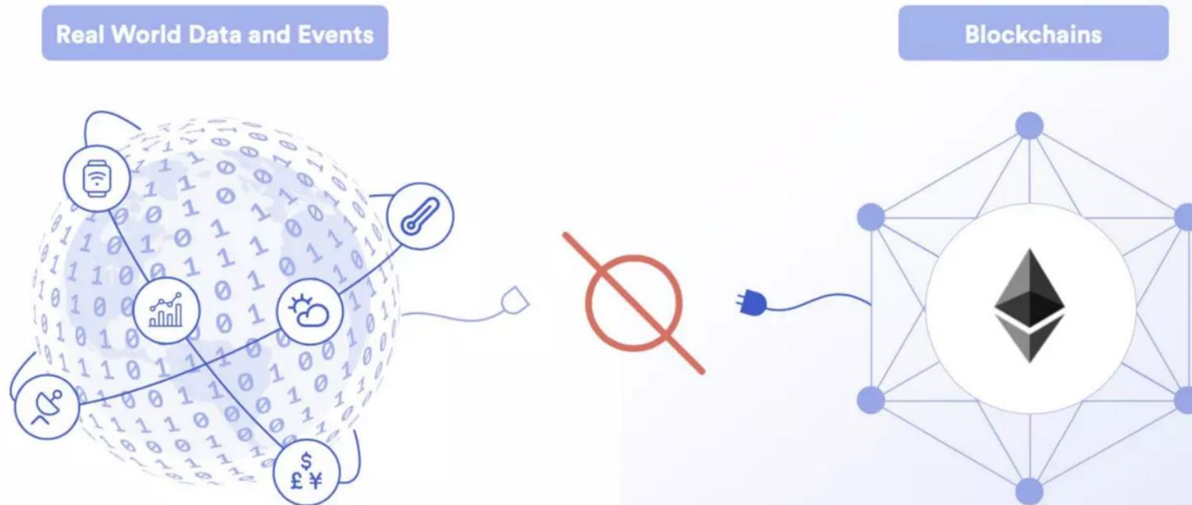
Challenge

The price of eth changes all the time.
How do a contract get the correct price off the blockchain?

Oracle Issue

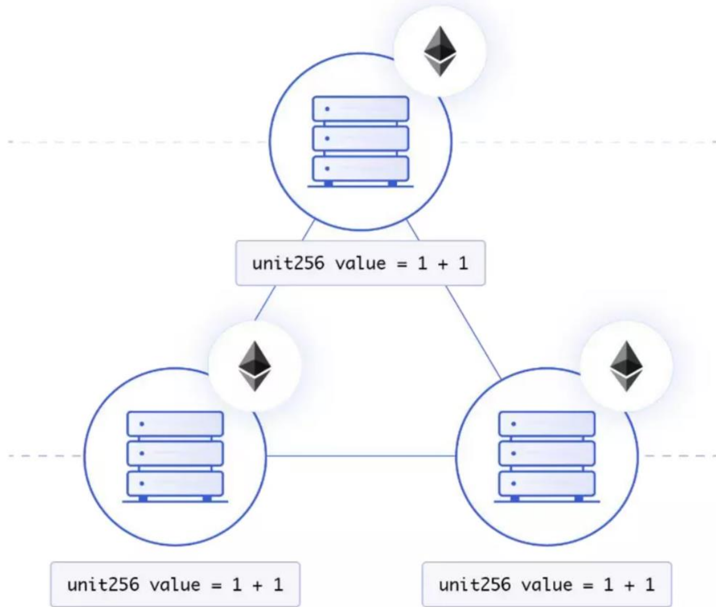
The Oracle Problem

Smart Contracts are unable to connect with external systems, data feeds, APIs, existing payment systems or any other off-chain resources on their own.



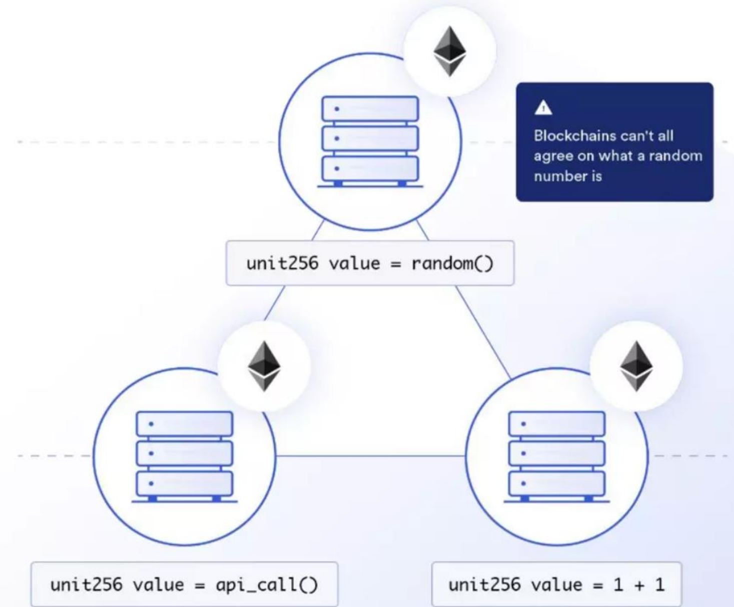
https://www.slideshare.net/VanessaCurman/chainlink-presentation

Deterministic



Consensus reached

Non-deterministic



Blockchains can't all agree on API calls

Consensus not reached

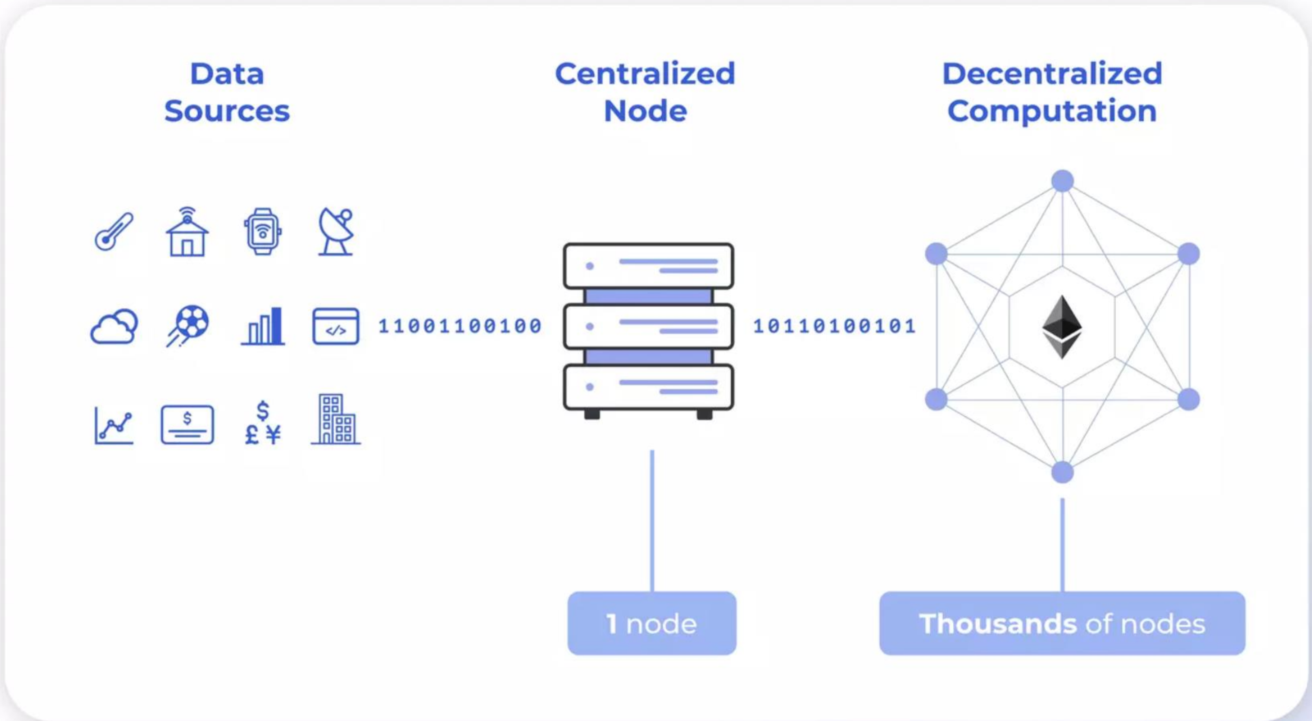
https://www.slideshare.net/VanessaCurman/chainlink-presentation

Blockchain Oracle: Any device that interacts with the off-chain world to provide external data or computation to smart contracts.

Chainlink

https://www.slideshare.net/VanessaCurman/chainlink-presentation

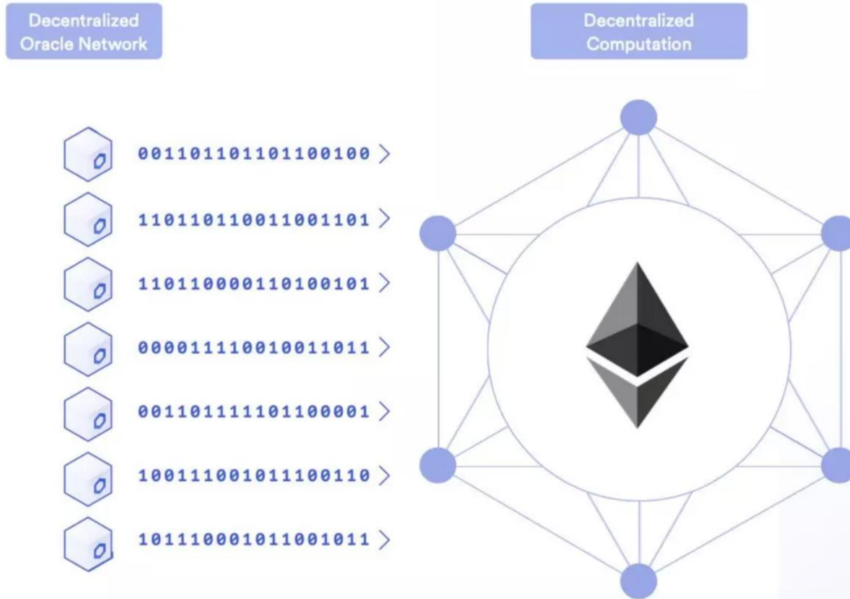
Centralized Oracles are a Point of Failure



Chainlink

https://www.slideshare.net/VanessaCurman/chainlink-presentation

A Decentralized Oracle Network



Decentralization

Full replicas being run by independent and sybil resistant node operators, coming to consensus about a computation.

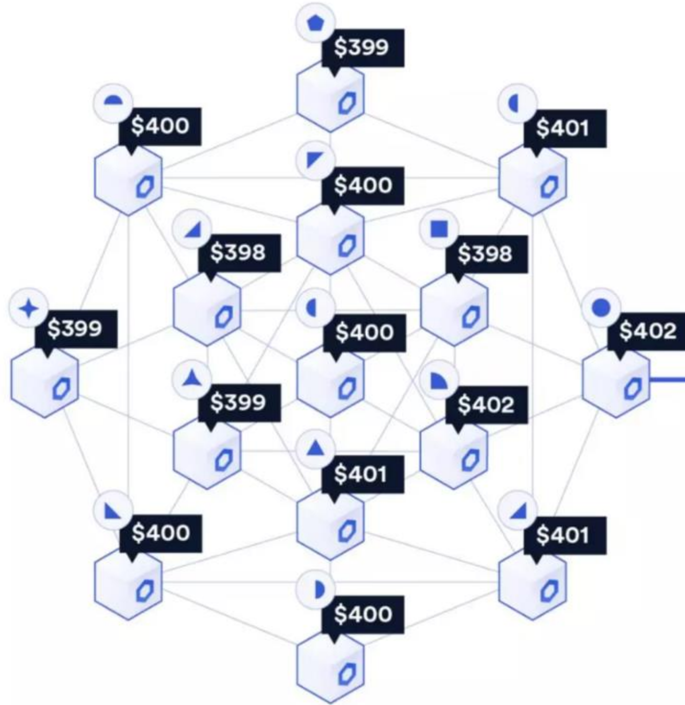
Focused on data validation and consensus about individual off-chain values to make them reliable enough to trigger contracts.

Node Operators are security reviewed, can provide a proven performance history and are high quality and highly sybil resistant.

https://www.slideshare.net/VanessaCurman/chainlink-presentation

Off-Chain

Chainlink Nodes

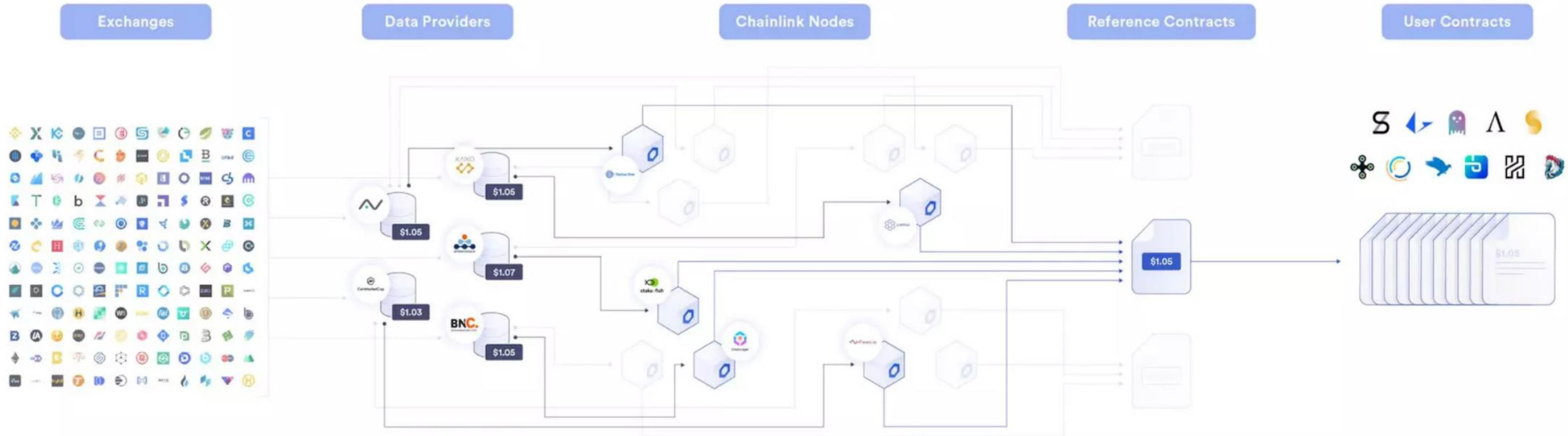


On-Chain

Smart Contract

https://www.slideshare.net/VanessaCurman/chainlink-presentation

How to feed external data



https://www.slideshare.net/VanessaCurman/chainlink-presentation

ETH/USD Data Feed

ETH / USD

Share

Answer

$1,909.40

Secured by Staking

24,026,964 LINK

Network: Ethereum Mainnet

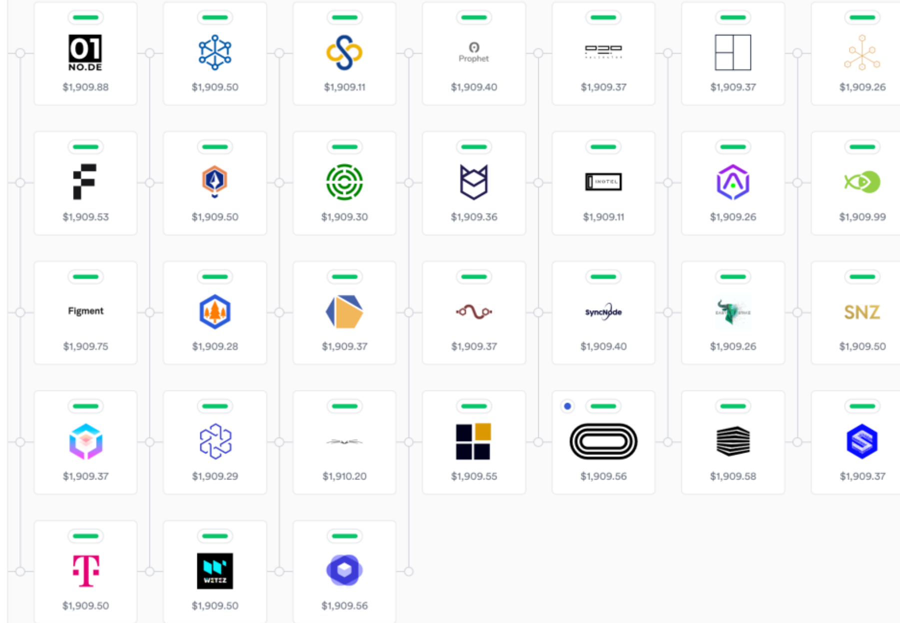
Also on other networks

Asset Name	Asset Class	Tier
Ethereum	Crypto	Verified

Trigger parameters

Deviation threshold	Heartbeat
0.5%	00:17:38

Oracle responses	Last update
Minimum of 21	April 28, 2023
31 / 31	42 minutes ago



https://data.chain.link/ethereum/mainnet/crypto-usd/eth-usd

Reading data feeds on-chain

PriceConsumerV3.sol

AggregatorV3Interface *defines*
all v3 Aggregators have the function getLatestPrice()

Proxy aggregator contract
for ETH/USD in Sepolia

Get the data from the latest round
the result will be in 8 decimals
place a decimal point before the last 8 digits
e.g., 190748000000
→ 1907.48000000 USD

https://docs.chain.link/getting-started/consuming-data-feeds/
https://docs.chain.link/data-feeds/api-reference/

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.7;
3
4   import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";
5
6   contract PriceConsumerV3 {
7       AggregatorV3Interface internal priceFeed;
8
9       /**
10       * Network: Sepolia
11       * Aggregator: ETH/USD
12       * Address: 0x694AA1769357215DE4FAC081bf1f309aDC325306
13       */
14      constructor() {
15          priceFeed = AggregatorV3Interface(
16              0x694AA1769357215DE4FAC081bf1f309aDC325306
17          );
18      }
19
20      /**
21       * Returns the latest price.
22       */
23      function getLatestPrice() public view returns (int) {
24          // prettier-ignore
25          (
26              /* uint80 roundID */,
27              int price,
28              /*uint startedAt*/,
29              /*uint timeStamp*/,
30              /*uint80 answeredInRound*/
31          ) = priceFeed.latestRoundData();
32          return price;
33      }
34  }
```

Creating a library

PriceConverter.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

library PriceConverter {

    function getEthPrice() internal view returns (uint256) {
        // get eth price in USD from chainlink
        // Network: Sepolia
        // Aggregator: ETH / USD
        // Contract Address: 0x694AA1769357215DE4FAC081bf1f309aDC325306
        // https://docs.chain.link/data-feeds/price-feeds/addresses#Sepolia%20Testnet
        AggregatorV3Interface priceFeed = AggregatorV3Interface(
            0x694AA1769357215DE4FAC081bf1f309aDC325306
        );
        (, int256 answer, , ,) = priceFeed.latestRoundData();

        // the answer will be 8 decimals and we need 18 decimals, so multiply 10 more decimals
        return uint256(answer * 10 ** 10);
    }

    function getUsdValueOfEth(uint256 ethAmount) internal view returns (uint256) {
        uint256 ethPrice = getEthPrice();                            // 10 ** 18 decimals
        uint256 ethAmountInUsd = (ethPrice * ethAmount) / (10 ** 18); // ethAmount is 10 ** 18 decimals
        return ethAmountInUsd;                                       // 10 ** 18 decimals
    }
}
```

Library
Embedded library contains only internal functions

The answer will be in 8 decimals, so, add 10 more decimals to unify the units

*(10 ** 18) * (10 ** 18) / (10 ** 18) = (10 ** 18): 18 decimals*

Using a library in a smart contract

TokenFundUsd.sol

Modified part of the previous TokenFundEth.sol

importing a library

```
5   import "./PriceConverter.sol";
6
7   contract Token {
8       using PriceConverter for uint256;
```

***using** LibraryName **for** Type*
to attach library functions to Type
any Type variables can use library functions

```
24      uint256 public constant MINIMUM_FUND = 1 * 10 ** 18;  // 1 USD
25      uint256 public constant EXCHANGE_RATE = 1;            // 1 USD = 1 MTT
```

msg.value (uint256)
call the library function

```
46      function fund() public payable {
47          // Add funder address and eth amount funded
48          uint256 usdAmount = msg.value.getUsdValueOfEth();
49          require(usdAmount >= MINIMUM_FUND, "The minimum fund is 1 USD");
50          addressToAmountFunded[msg.sender] += msg.value;
51          funders.push(msg.sender);
52
53          // Transfer the amount of token equivalent to funded eth
54          uint256 tokenAmount = uint256(usdAmount * EXCHANGE_RATE / 10 ** 18);
55          require(balances[owner] >= tokenAmount, "We don't have enough tokens");
56          balances[owner] -= tokenAmount;
57          balances[msg.sender] += tokenAmount;
```

Solidity Library

- **Solidity library**
  - A different type of smart contract that contains reusable code
  - Once deployed on the blockchain (only once), it is assigned a specific address
  - Its properties / methods can be reused many times by other contracts

- **Why using libraries**
  - **Reusable**: save development time and resources
  - **Economical**: save gas by using already deployed libraries
  - **Robust**: protect contracts with well-written libraries and established best practices

https://jeancvllr.medium.com/solidity-tutorial-all-about-libraries-762e5a3692f9
https://docs.soliditylang.org/en/v0.8.19/contracts.html#libraries

Limitations in Solidity Library

- Solidity libraries are considered **stateless**
- They **do not have any storage** (so can't have non-constant state variables)
- They **can't hold ethers** (so can't have a fallback function)
- **Doesn't allow payable functions** (since they can't hold ethers)
- Cannot inherit nor be inherited
- Can't be destroyed (no selfdestruct() function since version 0.4.20)

→ It should only be used to **perform simple operations based on input and returns result**

https://jeancvllr.medium.com/solidity-tutorial-all-about-libraries-762e5a3692f9

Two Types of Solidity Library

- **Embedded library**
  - A library which have only **internal** functions
  - The EVM simply **embeds** library into the contract
  - It simply uses JUMP statement(normal method call) instead of using delegate call

- **Linked library**
  - A library which have **public or external** functions
  - A library needs to be deployed and will get a unique address in the blockchain
  - This address needs to be linked with calling contract
  - Calling a function from a library will use a special instruction in the EVM: **DELEGATECALL** opcode
  - This will cause the calling context to be passed to the library, **like if it was some code running in the contract itself**

  → this, msg.sender, msg.value, and etc will have values of the calling contract

https://jeancvllr.medium.com/solidity-tutorial-all-about-libraries-762e5a3692f9

Linked library Example

LinkedLibraryExample.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
library MathLib {

    function mult(uint a, uint b) public view returns (uint, address) {
        return (a * b, address(this));
    }
}

contract Example {

    using MathLib for uint;
    address owner = address(this);

    function multiplyExample(uint _a, uint _b) public view returns (uint, address) {
        return _a.mult(_b);
    }
}
```

DEPLOY & RUN TRANSACTIONS

CONTRACT (Compiled by Remix): Example - contracts/LinkedLibraryEx...

Deploy | Publish to IPFS | OR | At Address | Load contract from Address

Transactions recorded 2

Deployed Contracts

EXAMPLE AT 0XD8B...33FA8 (MEI

Balance: 0 ETH

multiplyExample

_a: 4

_b: 5

Calldata | Parameters | call

0: uint256: 20

1: address: 0xd8b934580fcE35a11B58C6D73aDeE468a2833fa8

listen on all transactions | Search with transaction hash or address

from	0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to	Example.multiplyExample(uint256,uint256) 0xd8b934580fcE35a11B58C6073aDeE468a2833fa8
execution cost	5533 gas (Cost only applies when called by a contract)
input	0x068...00005
decoded input	{ "uint256 _a": "4", "uint256 _b": "5" }
decoded output	{ "0": "uint256: 20", "1": "address: 0xd8b934580fcE35a11B58C6073aDeE468a2833fa8" }

address(this) in mult() of MathLib returns the Example contract address not the MathLib address

https://jeancvllr.medium.com/solidity-tutorial-all-about-libraries-762e5a3692f9

Useful Solidity Libraries from OpenZeppelin

- **access/Ownable.sol**: provide onlyOwner() modifier
- **access/AccessControl.sol**: provide role-based access control
- **utils/math/Math.sol**: standard math library such as sqrt() and log2()
- **utils/Address.sol**: Collection of functions related to the address type
- **utils/Counters.sol**: Provides counters that can only be incremented, decremented or reset
- **utils/Strings.sol**: String operations such as toString() and toHexString()
- **utils/Multicall.sol**: Provides a function to batch together multiple calls in a single external call

https://docs.openzeppelin.com/contracts/4.x/access-control
https://docs.openzeppelin.com/contracts/4.x/utilities
https://docs.openzeppelin.com/contracts/4.x/api/utils
https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/utils

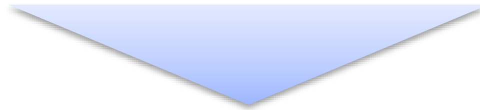
Improvement 3. Minting ERC20 tokens

Added features

1) Minting MTT tokens as ERC20
2) Distribute MTT tokens to the funders of eth

We minted our token, MTT.
However, we can’t see them in Metamask
and connect them to DeFi apps.

WHY?
*Our token implementation is **not the standard way***



ERC20 Token

ERC20 Token Standard

- **ERC20**: a standard interface(format) for fungible assets(tokens) on the Ethereum

  "fungible" means each token be exactly the same as another token

- **Benefits of ERC20 tokens**
  - **Standardization**: saving time and resources to develop
  - **Interoperability**: easily interact with various wallets, exchanges, and decentralized applications (dApps) on the Ethereum
  - **Security**: extensively tested and reviewed by the Ethereum community
  - **Programmability**: can be tailored to serve a specific purpose or function, making them suitable for a wide range of applications
  - **Transparency**: easy tracking and verification of ERC20 token transactions
  - **Borderless transactions**: facilitate seamless, borderless transactions without the need for intermediaries

https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
https://blog.thirdweb.com/what-is-erc-20-token/

Interface of the ERC20 standard

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    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);
}
```

functions allowing other contracts to transfer tokens on your behalf

https://eips.ethereum.org/EIPS/eip-20
https://ethereum.org/en/developers/tutorials/erc20-annotated-code/
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/IERC20.sol

Implementing ERC20 Token from scratch

ManualERC20Token.sol

the number of decimals used to get its user representation. For example, if `decimals` equals `2`, a balance of `505` tokens should be displayed to a user as `5.05` (`505 / 10 ** 2`)

Tokens usually opt for a value of 18, ***imitating the relationship between Ether and Wei 1 ether = 10 ** 18 wei***

An account can allow contracts to transfer tokens on its behalf ***_allowance*** *stores the addresses authorized to spend and the max amount they can spend*

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

contract TokenERC20 is IERC20 {
  // the token information
  string private _name;
  string private _symbol;
  uint8 private _decimals = 18;
  // 18 decimals is the strongly suggested default, avoid changing it
  uint256 private _totalSupply;

  // This creates an array with all balances
  mapping(address => uint256) private _balances;

  // This creates an array of mapping of the addresses authorized to spend
  //                                   and the max amount they can spend
  mapping(address => mapping(address => uint256)) private _allowances;

  // This notifies clients about the amount burnt
  event Burn(address indexed from, uint256 value);

  // Initializes contract with initial supply tokens to the creator of the contract
  constructor(uint256 initialSupply, string memory tokenName, string memory tokenSymbol) {
    // Update total supply with the decimal amount
    _totalSupply = initialSupply * 10**uint256(_decimals);

    // Give the creator all initial tokens
    _balances[msg.sender] = _totalSupply;
    _name = tokenName;
    _symbol = tokenSymbol;
  }
```

https://github.com/PatrickAlphaC/hardhat-erc20-fcc/blob/main/contracts/ManualToken.sol

Sending tokens

Adjust balances between the sender and the receiver in atomic operation

```
58  function _transfer(address _from, address _to, uint256 _value) internal {
59    // Prevent transfer to 0x0 address. Use burn() instead
60    require(_to != address(0x0));
61    // Check if the sender has enough
62    require(_balances[_from] >= _value);
63    // Check for overflows
64    require(_balances[_to] + _value >= _balances[_to]);
65    // Save this for an assertion in the future
66    uint256 previousBalances = _balances[_from] + _balances[_to];
67    // Subtract from the sender
68    _balances[_from] -= _value;
69    // Add the same to the recipient
70    _balances[_to] += _value;
71    emit Transfer(_from, _to, _value);
72    // Asserts are used to use static analysis to find bugs in your code. They should never fail
73    assert(_balances[_from] + _balances[_to] == previousBalances);
74  }
75
76  function transfer(address _to, uint256 _value) public returns (bool success) {
77    _transfer(msg.sender, _to, _value);
78    return true;
79  }
```

Delegating to send tokens

Set the max amount allowed to spend by _spender account on behalf of msg.sender

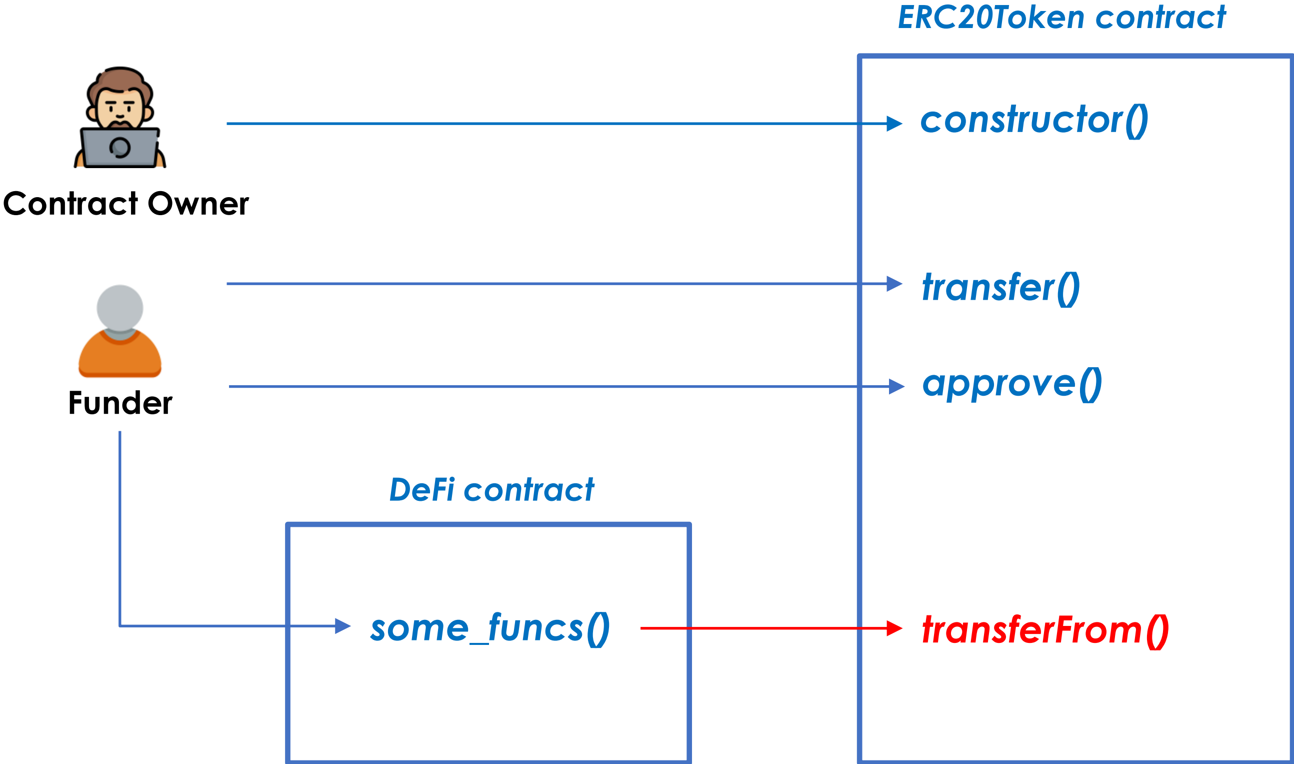
Called by the account owner

Check the allowance of msg.sender then, transfer tokens (adjusting balances of _from and _to)

Called by the delegator usually contracts like DeFi

```
89   /**
90    * Set allowance for other address
91    *
92    * Allows `_spender` to spend no more than `_value` tokens on your behalf
93    *
94    * @param _spender The address authorized to spend
95    * @param _value the max amount they can spend
96    */
97   function approve(address _spender, uint256 _value) public returns (bool success) {
98     _allowances[msg.sender][_spender] = _value;
99     emit Approval(msg.sender, _spender, _value);
100    return true;
101  }
102
103  function allowance(address _owner, address _spender) public view returns (uint256) {
104    return _allowances[_owner][_spender];
105  }
106
107  function transferFrom(address _from, address _to, uint256 _value)
108    public returns (bool success) {
109    require(_value <= _allowances[_from][msg.sender]); // Check _allowances
110    _allowances[_from][msg.sender] -= _value;
111    _transfer(_from, _to, _value);
112    return true;
113  }
```

Context to call functions



Burning tokens (optional)

Burning tokens is just the same as decreasing tokens from the balances without increasing them somewhere

Called by the account owner

Check the allowance of msg.sender then, burn tokens

Called by the delegator

```
129  /**
130   * Destroy tokens
131   *
132   * Remove `_value` tokens from the system irreversibly
133   *
134   * @param _value the amount of money to burn
135   */
136  function burn(uint256 _value) public returns (bool success) {
137    require(_balances[msg.sender] >= _value); // Check if the sender has enough
138    _balances[msg.sender] -= _value; // Subtract from the sender
139    _totalSupply -= _value; // Updates totalSupply
140    emit Burn(msg.sender, _value);
141    return true;
142  }
143
144  function burnFrom(address _from, uint256 _value) public returns (bool success) {
145    require(_balances[_from] >= _value); // Check if the targeted balance is enough
146    require(_value <= _allowances[_from][msg.sender]); // Check allowance
147    _balances[_from] -= _value; // Subtract from the targeted balance
148    _allowances[_from][msg.sender] -= _value; // Subtract from the sender's allowance
149    _totalSupply -= _value; // Update totalSupply
150    emit Burn(_from, _value);
151    return true;
152  }
```

Too Complicated?
Don't worry. There are
reference implementations of ERC20 token

Inherit OpenZeppelin ERC20 Implementation
to create your own ERC20 token

1 million ERC20 token (MTT) in 12 lines of code!

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.9;
3
4   import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6   contract MTTToken is ERC20 {
7       uint256 private initialSupply = 1000000;
8
9       constructor() ERC20("My Test Token", "MTT") {
10          _mint(msg.sender, initialSupply * (10 ** decimals()));
11      }
12  }
```

SimpleERC20Token.sol

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol

Getting fund with ERC20 tokens

Previously, we just adjusted balances for distributing tokens. However, it's not possible in ERC20 since ***balances is private***

In order to change balances, we need to use ***_mint(), transfer(), transferFrom()***

```
42     function fund() public payable {
43         // Add funder address and eth amount funded
44         require(msg.value >= MINIMUM_FUND, "The minimum fund is 0.001 ETH");
45         addressToAmountFunded[msg.sender] += msg.value;
46         funders.push(msg.sender);
47
48         // Transfer the amount of token equivalent to funded eth
49         uint256 tokenAmount = getTokenAmountInETH(msg.value);
50         require(balances[owner] >= tokenAmount, "We don't have enough tokens");
51         balances[owner] -= tokenAmount;
52         balances[msg.sender] += tokenAmount;
```

Two ways to implement

1) *using* ***_mint()*** *in ERC20 token contract itself*
2) *calling* ***transfer()*** *in a separate contract*
   (calling a smart contract from another smart contract)

1) using _**mint()** in ERC20 token contract itself

MTTTokenMint.sol

set a maximum supply since totalSupply will start at 0 and increase with every minting

Problem)
totalSupply is not fixed

msg.senders mint tokens themselves

Problem)
funder mint tokens directly

```
25    uint256 private _maxSupply = 1000000;
26    address private _owner;
27
28    constructor() ERC20("My Test Token", "MTT") {
29        _maxSupply = _maxSupply * (10 ** decimals());
30        _owner = msg.sender;
31    }
32
33    function fund() public payable {
34        // Add funder address and eth amount funded
35        require(msg.value >= MINIMUM_FUND, "The minimum fund is 0.001 ETH");
36        _addressToAmountFunded[msg.sender] += msg.value;
37        _funders.push(msg.sender);
38
39        // Transfer the amount of token equivalent to funded eth
40        uint256 tokenAmount = uint256(msg.value * EXCHANGE_RATE / 10 ** 18) * (10 ** decimals());
41        require(tokenAmount + totalSupply() <= _maxSupply, "Not enough tokens");
42        _mint(msg.sender, tokenAmount);
43
44        // Notify the funding.
45        emit Funding(msg.sender, msg.value, tokenAmount);
46    }
```

2) calling **transfer()** in a separate contract

MTTToken.sol (callee contract)

```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.9;
3
4   import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6   contract MTTToken is ERC20 {
7
8       // Check the minting was done
9       bool private _minted;
10
11      constructor() ERC20("My Test Token", "MTT") {
12          _minted = false;
13      }
14
15      function mintToken(uint256 initialSupply) external {
16          require(!_minted, "Already minted");
17          // Mint all tokens to msg.sender (caller contract)
18          _mint(msg.sender, initialSupply * (10 ** decimals()));
19          // Set the minting is done
20          _minted = true;
21      }
22  }
```

Implementing a ERC20 token (→ line 6)

Another contract will call the mint function (→ line 15)

All tokens will be minted to a caller contract (→ line 18)

https://blog.chain.link/smart-contract-call-another-smart-contract/

MTTokenFund.sol (caller contract)

```
26      uint256 private _initialSupply = 1000000;
27      // token contract variable of Type MTTToken
28      MTTToken private _token;
29
30      constructor(address tokenAddress) {
31          // Initialize _token with the address of the MTTToken deployed previously.
32          _token = MTTToken(tokenAddress);
33          // Mint all tokens to this contract (caller contract)
34          // This contract will hold all tokens
35          _token.mintToken(_initialSupply);
36          _owner = msg.sender;
37      }
38
39      function fund() public payable {
40          // Add funder address and eth amount funded
41          require(msg.value >= MINIMUM_FUND, "The minimum fund is 0.001 ETH");
42          _addressToAmountFunded[msg.sender] += msg.value;
43          _funders.push(msg.sender);
44
45          // Calculate the amount of token equivalent to funded eth
46          uint256 tokenAmount = uint256(msg.value * EXCHANGE_RATE / 10 ** 18);
47          tokenAmount *= (10 ** _token.decimals());
48
49          // Call the MTTToken contract to send tokens to msg.sender
50          // When transfer() in the MTTToken contract is called,
51          // msg.sender will be this contract (caller contract)
52          // Then, tokens will be transfered from this contract address
53          bool success = _token.transfer(msg.sender, tokenAmount);
54          if (!success) {
55              revert("Token transfer failed");
56          }
```

Initialize a contract variable with MTTToken contract address (line 32)

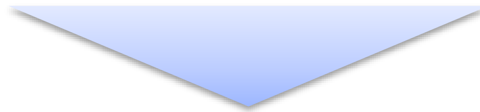
Minting all tokens to this caller contract (line 35)

Question) Is it safe?

This caller contract call the transfer() of MTTToken in the caller's context (msg.sender is the caller contract) (line 53)

After creating the MTTToken contract,
any contracts can call mintToken().

We can't guarantee
the MTTTokenFund contract will be the first caller.



Both creating the token contract and minting tokens
should be ***an atomic operation***
→ *Creating a smart contract from another contract*

MTTTokenFactory.sol

MTTToken: callee contract

MTTTokenFactory: caller contract

create the token contract with input parameters given by the caller contract

mint all tokens to the caller contract

the token contract variable

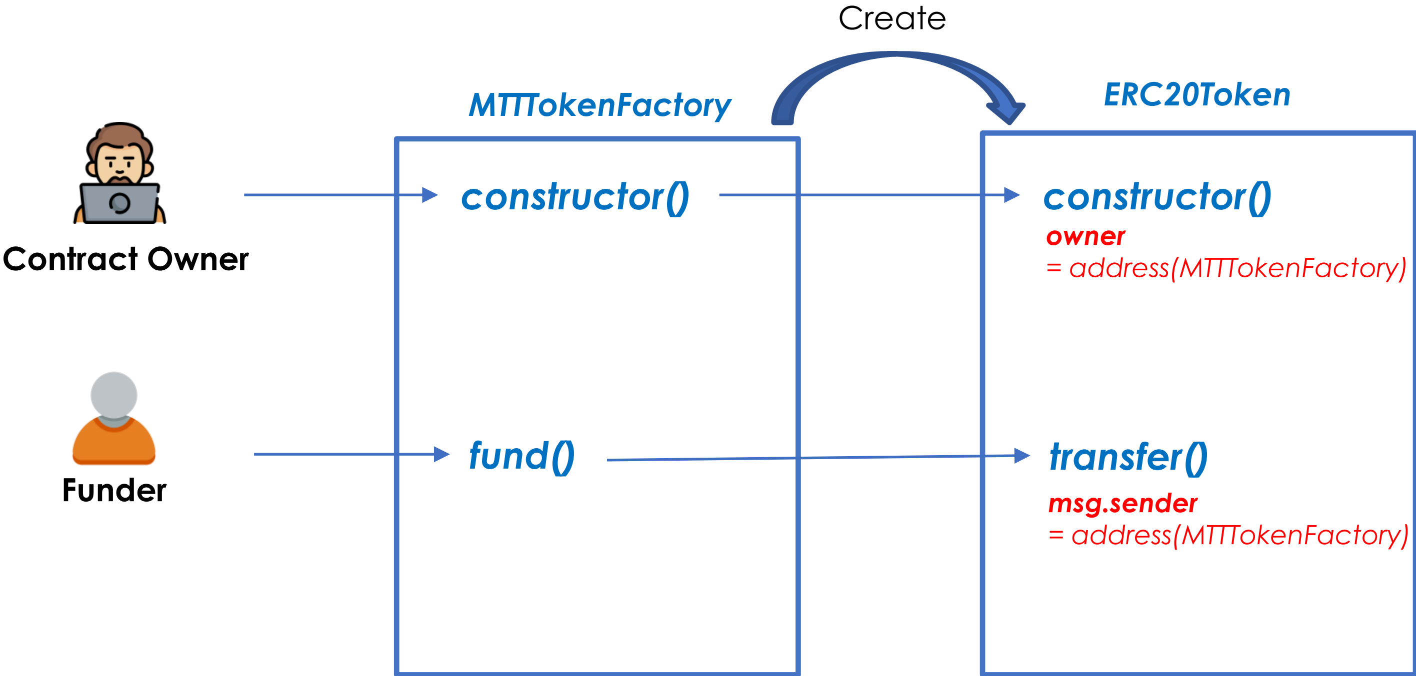
create new token contract with this caller contract as an owner

This caller contract call the transfer() of MTTToken in the caller's context (msg.sender is the caller contract)

```
1  //SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.9;
3
4  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
5
6  // ERC20 token contract
7  // Minting all tokens to the caller contract which create this contract
8  contract MTTToken is ERC20 {
9      address private _owner;
10     constructor(address owner, uint256 initSupply) ERC20("My Test Token", "MTT") {
11         // address of the caller contract
12         _owner = owner;
13         // mint all tokens to the caller contract
14         _mint(owner, initSupply * (10 ** decimals()));
15     }
16 }
17
18 contract MTTTokenFactory {
19     address private _owner;
20     uint256 private _initialSupply = 1000000;
21     // token contract variable of Type MTTToken
22     MTTToken private _token;
23     address public tokenAddress;
```

```
40     constructor() {
41         _owner = msg.sender;
42         // Create _token with caller contract address as an owner
43         _token = new MTTToken(address(this), _initialSupply);
44         tokenAddress = address(_token);
45     }
46
47     function fund() public payable {
48         ...
49
50         // Call the MTTToken contract to send tokens to msg.sender
51         bool success = _token.transfer(msg.sender, tokenAmount);
52         if (!success) {
53             revert("Token transfer failed");
54         }
```

Context to call functions



ERC20 Token Use Cases

- **Utility tokens**: Tokens that provide access to a project's platform or services such as Basic Attention Token (BAT) for the Brave browser ecosystem.
- **Governance tokens**: Tokens that grant holders voting rights in decentralized organizations, like Maker (MKR).
- **Stablecoins**: Tokens pegged to traditional currencies, such as USD Coin (USDC).
- **Asset-backed tokens**: Tokens representing ownership of physical or digital assets, like tokenized gold or real estate.
- **In-game currencies and items**: ERC-20 tokens can be used for virtual currencies or items within video games, streamlining the management of in-game economies.

https://blog.thirdweb.com/what-is-erc-20-token/

Etherscan *https://etherscan.io/tokens*

Token Tracker (ERC-20)

A total of **1,243** Token Contracts found

#	Token	Price	Change (%)	Volume (24H)	Circulating Market Cap	On-Chain Market Cap	Holders
1	**Tether USD** (USDT)	$1.001 0.000525 ETH	-0.04%	$11,808,137,734.00	$81,828,254,387.00	$39,862,942,964.85	4,318,915 0.033%
2	**BNB** (BNB)	$322.414 0.169153 ETH	-0.45%	$454,680,969.00	$50,252,337,864.00	$5,345,468,429.58	280,105 0.006%
3	**USD Coin** (USDC)	$1.00 0.000525 ETH	-0.02%	$3,316,015,574.00	$30,524,219,141.00	$46,602,430,840.00	1,673,279 0.022%
4	**stETH** (stETH)	$1,900.28 0.996973 ETH	-0.63%	$15,151,046.00	$11,802,055,254.00	$3,522,376,110.52	197,440 0.067%
5	**Matic Token** (MATIC)	$0.9971 0.000523 ETH	-2.40%	$320,927,787.00	$9,222,308,074.00	$9,970,635,076.12	605,372 0.005%
6	**HEX** (HEX)	$0.0519 0.000027 ETH	-1.15%	$8,157,926.00	$9,003,348,400.00	$29,986,802,185.83	332,870 -0.001%
7	**Binance USD** (BUSD)	$1.001 0.000525 ETH	0.01%	$1,226,077,430.00	$6,202,832,212.00	$17,592,669,900.81	167,608 0.007%
8	**SHIBA INU** (SHIB)	$0.00 0.000000 ETH	-1.11%	$66,548,437.00	$6,041,868,272.00	$10,259,916,213.72	1,295,851 0.008%
9	**Theta Token** (THETA)	$5.2667 0.002763 ETH	-9.81%	$271,444,395.00	$5,266,695,404.00	$5,266,695,404.45	28,165 0.000%
10	**Dai Stablecoin** (DAI)	$0.9998 0.000525 ETH	0.01%	$50,468,466.00	$4,734,413,072.00	$9,797,418,008.27	505,222 0.019%
11	**Wrapped BTC** (WBTC)	$29,309.00 15.376826 ETH	-0.34%	$51,729,955.00	$4,513,436,481.00	$7,673,506,526.00	69,887 0.009%

Wrap-up

We Learned

- Two crowdfunding contracts
  - using non-ERC20 tokens
  - using ERC20 tokens

- Sending eth
- Solidity Library
- Calling a function of another contract
- Creating another contract from a contract
- Oracle and chainlink data feed
- ERC20 token standard
- Use cases of ERC20 tokens