Nadia Fabrizio

Blockchain Laboratory

Fintech

Agenda

1. April,14th - Intro to solidity & smart contracts
2. May,5th- Overview of ERC20 and 721, lab
3. May,12th- Lab, ECDSA
4. May 19th, Lab
5. May 26th, Test in class

Objectives of the Lab

- What are Smart Contracts and how they work
- Basic layout of Solidity interfaces for Smart Contracts
- How to develop, compile and launch an ERC20
- Elliptic curves in Bitcoin/Ethereum protocols.

REFERENCES

- https://remix.ethereum.org/
- https://docs.soliditylang
- https://ethereum.org/en/developers/docs/smart-contracts/
- https://ethereum.org/en/developers/
- Solidity — Solidity 0.8.19 documentation (soliditylang.org)

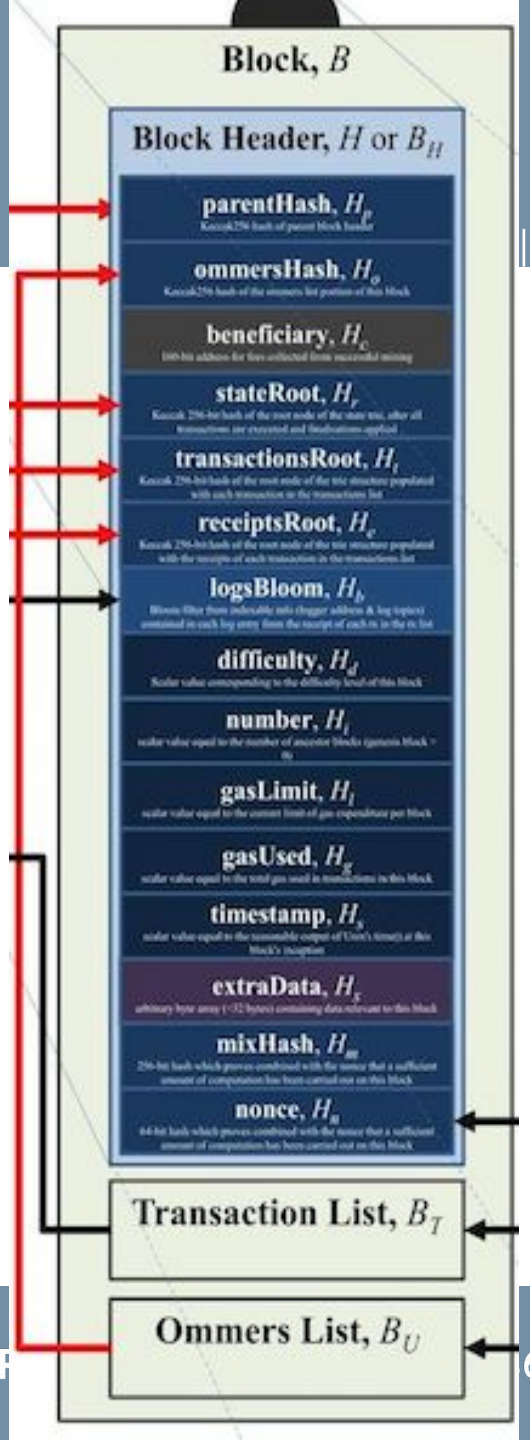
Assumptions

- You followed the previous lessons and you know what Ethereum is and the basic ideas behind
- You know the basic principles of blockchain and consensus

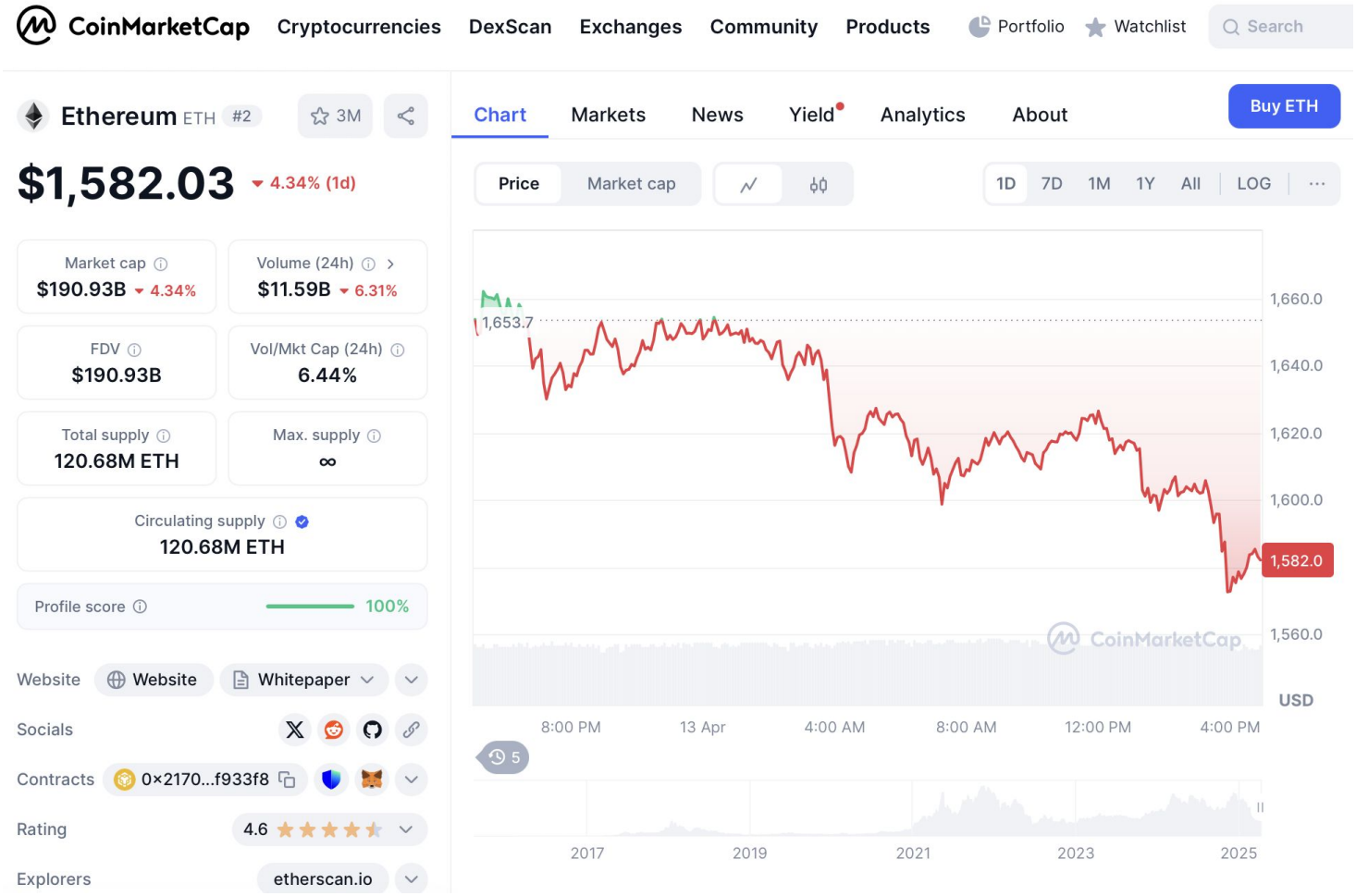
Block in a BC over Ether

Blocks consist of 3 elements

- **Transaction List**
  - List of all transactions included in a block
- **Block Header**
  - Group of 15 elements
- **Ommer List**
  - List of all Uncle blocks included



Where we are now
SOURCE: https://coinmarketcap.com/currencies/ethereum/



Smart Contracts Intro

- **Smart contracts are computer programs** that run over the Ethereum Network

- *Smart contract" is simply a program that runs on the Ethereum blockchain. It's a **collection of code** (its functions) and **data** (its state) that resides at a specific address on the Ethereum blockchain.*
- They **execute when triggered by a transaction from a user (or another contract)**.
- These programs are the core engine of **DAPPS,** decentralized application.
- **Popular examples** of smart contracts are lending apps, decentralized trading exchanges, insurance, crowdfunding apps - basically anything you can think of.
-

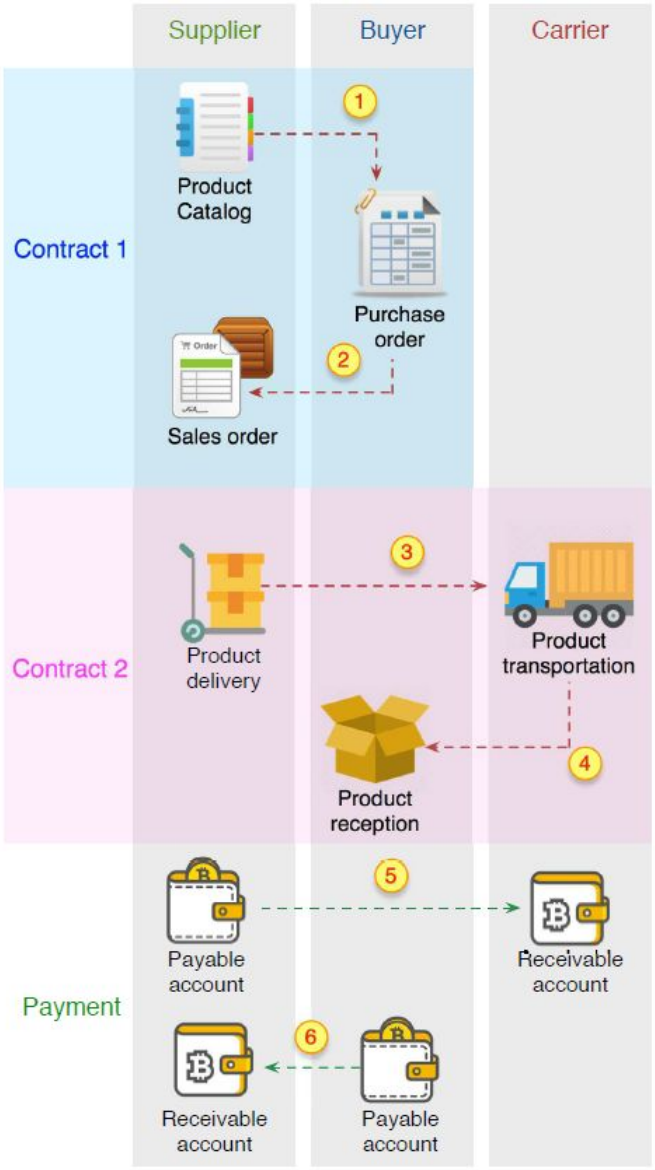


Figure 1. An example of a smart contract between a buyer and a supplier.

Source:An Overview on Smart Contracts: Challenges, Advances and Platforms Zibin Zheng,Shaoan Xie et al

https://www.sciencedirect.com/science/article/abs/pii/S0167739X19316280

Blockchain technology is enabling *smart contracts* that **were first proposed in 1990s by Nick Szabo**. In a smart contract, **contract clauses written in computer programs will be automatically executed** when predefined conditions are met.
Smart contracts consisting of transactions are essentially stored, replicated and updated in distributed blockchains.

Source:An Overview on Smart Contracts: Challenges, Advances and Platforms
Zibin Zheng,Shaoan Xie et al https://www.sciencedirect.com/science/article/abs/pii/S0167739X19316280

Smart Contracts Intro

- **Smart contracts are** a type of Ethereum account.
- This means **they have a balance** and can be the target of transactions
- However **they're not controlled by a user**, instead they **are deployed to the network and run as programmed.**
- **Smart contracts can define rules**, like a regular contract, and automatically enforce them via the code.
- **Smart contracts cannot be deleted by default,** and interactions with them are irreversible
- **User accounts can then interact** with a smart contract by submitting transactions that execute a function defined on the smart contract.

Ether

- Ethereum runs on its **native token** which serves two main purposes:
  - **Ether** payment is required for applications to perform any operation so that broken and malicious programs are kept under control
  - **Ether is rewarded as an incentive** to the miners who contribute to the Ethereum network with their resources- much like bitcoin's structure.
- Every time a contract is executed, Ethereum consumes token which is termed as '**gas**' to run the computations.
- Gas is required to be paid for every operation performed on the Ethereum blockchain.
- Its price is expressed in ether and it's decided by the miners, which can refuse to process the transaction with less than a certain gas price.

https://finematics.com/what-is-gas-ethereum-high-transaction-fees-explained/

GAS

Gas is a unit used for measuring the amount of computational effort required to perform specific actions on the Ethereum blockchain.

Units in Ethereum		
Unit	Number per ETH	Most appropriate uses
Ether (ETH)	1	Currently used to denominate transaction amounts (eg 20 ETH) and mining rewards (5 ETH)
finney	1,000	
szabo	1,000,000	Currently the best unit for the cost of a basic transaction, eg 500 szabo
Gwei	1,000,000,000	Currently the best unit for Gas Prices eg 22 Gwei
Mwei	1,000,000,000,000	
Kwei	1,000,000,000,000,000	
wei	1,000,000,000,000,000,000	The base indivisible unit used by programmers

GAS

Every operation on the the Ethereum Virtual Machine (EVM), has an associated gas cost.

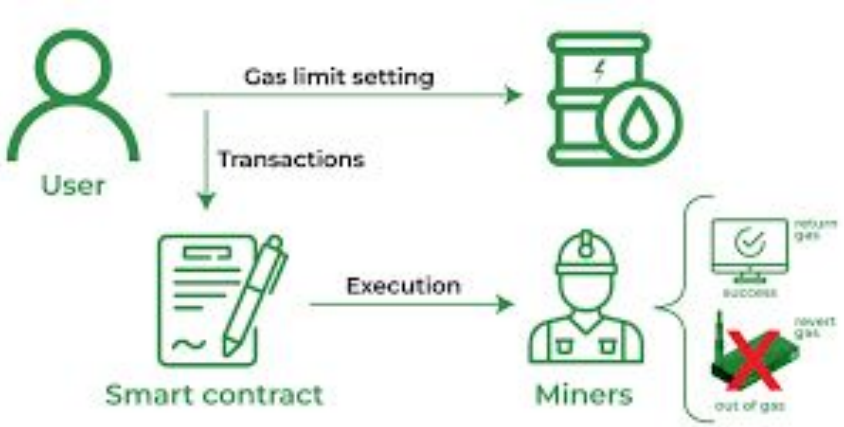
-Addiction, 2 gas;

- Getting the balance of an account – 400 gas; s

- Sending a transaction – 21,000 gas.

Smart contracts gar price = ∑ operations single gas price

T



Why Gas

- Halting problem (infinite loop) – reason for Gas
  - Problem: **Cannot tell whether or not a program will run infinitely** from compiled code
  - Solution: **charge fee per computational step** to limit infinite loops and stop flawed code from executing
- Every transaction needs to specify an estimate of the amount of gas it will spend

And why is needed?

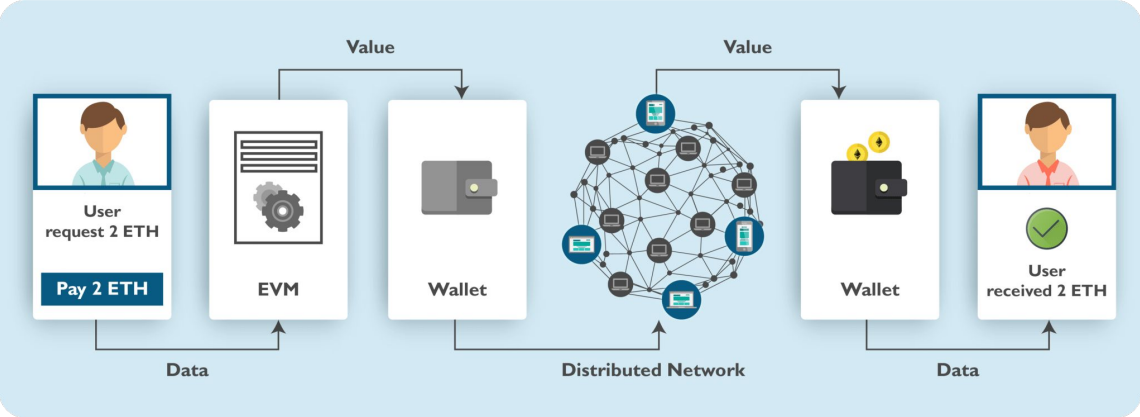
- **Gas Price**: current market price of a unit of Gas (in Wei)
  - **https://ethgasstation.info/**
  - **https://etherscan.io/gastracker**

GAS in short

- always set before a transaction by user
- **Gas Limit:** maximum amount of Gas user is willing to spend
- Helps to regulate load on network
- Gas Cost (used when sending transactions) is calculated by gasLimit*gasPrice.
- All blocks have a Gas Limit (maximum Gas each block can use)

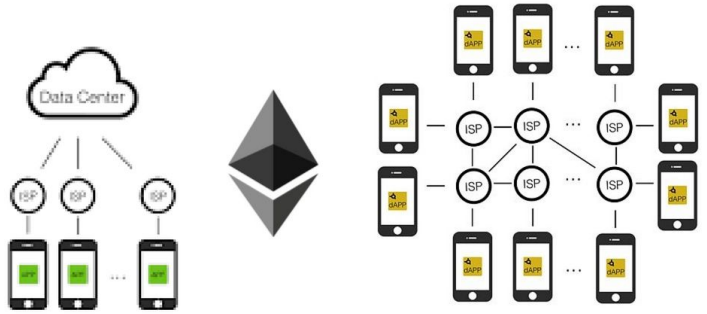
Ethereum Virtual Machine (E.V.M)

- The Ethereum virtual machine is the engine in which transaction code gets executed
- E.V.M. enables the development of potentially thousands of different applications all on one platform
- Contracts written in a smart contract-specific programming language are compiled into ‘bytecode’, which an EVM can read and execute



Decentralized Applications (DApps)

- **DApps** are computer applications that operate over a blockchain enabling direct interaction between end users and providers
- **The interface of the decentralized applications does not look any different than any website or mobile app today.**
- The **smart contract represents the core logic** of a decentralized application. Smart contracts are integral building blocks of blockchains, that process information from external sensors or events and help the blockchain manage the state of all network actors.



Ethereum Nodes

- Validate all transactions and new blocks (consensus is PoW like Bicoin)
- Operate in a "P2P" fashion
- Each contains a copy of the entire Blockchain
- Nodes may be full or light
- A Light Node store only block headers
    - Provide easy verification through tree data structure
    - Don’t execute transactions, used primarily for balance validation
    - Example: A wallet running over a mobile phone
- Implemented in a variety of languages (Go, Rust, etc.)

Account TYPES on ETHEREUM

- **Externally-owned account (EOA)** – controlled by anyone with the private keys
- **Contract account** – a smart contract deployed to the network, controlled by code.
- Each account has the following fields

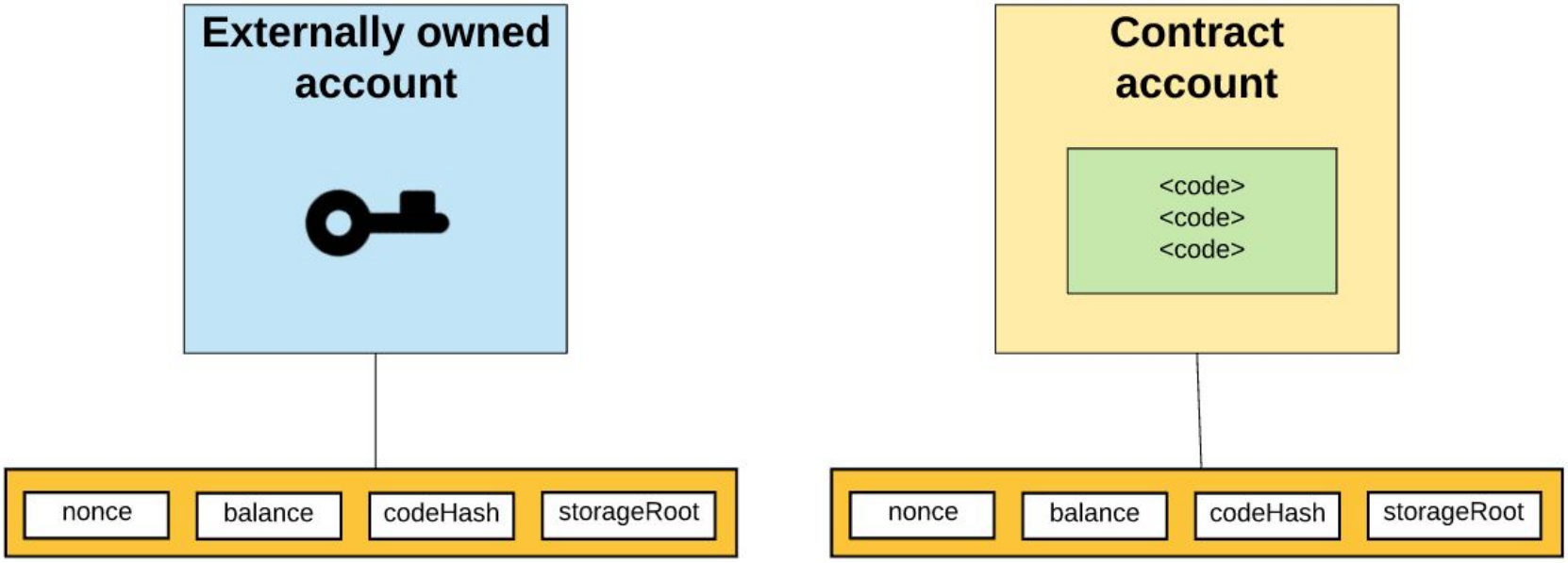
Both account types have the ability to:

- Receive, hold and send ETH and tokens
- Interact with deployed smart contracts

Account TYPES on ETHEREUM

- Each account has the following fields
  a. **Nonce** – a number corresponding to the amount of (a) transactions sent from or (b) contracts created by an account
  b. **Balance** – amount of wei owned by an account
  c. **StorageRoot** – a hash of the root node of a hash tree that encodes the storage contents of the account
  d. **codeHash** – a hash of the account's EVM code

Contract account



NONCE

NONCE=The nonce is a **transaction counter** associated with each Ethereum account (both externally owned accounts and smart contracts).

- **Purpose**: It ensures that transactions are executed in the correct order and prevents replay attacks.
- **Increment**: For externally owned accounts (EOAs), the nonce increments with each transaction sent from that account. For smart contracts, the nonce is relevant when creating new contracts.
- **Example**: If an EOA has a nonce of 5, the next transaction it sends must have a nonce of 6.

BALANCE

- **Definition**: The balance represents the **amount of Ether (ETH)** held by an Ethereum account (EOA or smart contract).
- **Units**: Balances are measured in wei (the smallest denomination of Ether).
- **Use Cases**:
  - For EOAs: Balances determine how much ETH an account can send or receive.
  - For Smart Contracts: Balances are crucial for managing funds within the contract.
- **Example**: An EOA with a balance of 10 ETH can send up to 10 ETH in a transaction.

WALLETS

Wallets:

- Consist of **a Smart contract with more advanced features**
- You have to register to have one
- A set of one or more external accounts
- Used to store/transfer ether

Example:https://metamask.io/

EOA

EOA: External Account (EOA, Valid Ethereum Address)

- Has an associated nonce (amount of transactions sent from the account) and a balance
- codeHash - Hash of associated account code, i.e. a computer program for a smart contract (hash of an empty string for external accounts, EOAs)
- Storage Root is root hash of Merkle-Patricia tree of associated account data

Contract account

Private Key:
0x2dcef1bfb03d6a950f91c573616cdd778d9581690db1cc43141f7cca06fd08ee

Ethereum Private keys are 66 character strings (with 0x appended). Case is irrelevant. Same derivation through ECDSA as BTC.

Address:
0xA6fA5e50da698F6E4128994a4c1ED345E98Df50

Ethereum Private keys map to addresses directly. Simply the last 40 characters of the Keccak-256 hash of the public key. Address is 42 characters total (append 0x to front).

WALLETS

Wallets:

- Consist of **a Smart contract with more advanced features**
- Crypto wallets are a form of digital wallet designed for web3. They help you manage permissions with whom you share your data, store cryptocurrency, NFTs, and more.
- You have to register to have one
- A set of one or more external accounts
- Used to store/transfer ether

Example:https://metamask.io/

BROWSERS VS WALLETS

Gateway to the Internet

Gateway to interaction with blockchains and their dapps

Ref:https://learn.metamask.io/lessons/what-is-a-crypto-wallet

ERC20

- Introduced in November 2015 as an Ethereum Request for Comments (ERC)
- Automatically assigned GitHub issue number 20, giving rise to name “ERC20”
- A standard for **fungible tokens,** meaning that different units of an ERC20 token are interchangeable and have no unique properties
- The ERC20 protocol standard contains basic functions that any useful token should implement to enable trading. These include transferring tokens, inquiring the balance of tokens at a certain address, and the total supply of tokens.
- The ERC-20 standard defines the interfaces f**or a few common methods**: i.e. totalSupply, balanceOf, transfer, transferFrom, and approve. These methods allow Ethereum smart contracts to issue fungible tokens and token holders to transfer tokens to one another. https://eips.ethereum.org/erc

ERC20 functions

totalSupply() - Returns the total units of this token that currently exist

balanceOf(address) - Returns the token balance of an address - transfer(address, amount)

Transfers amount of tokens to address, from the balance of the address that executed the transaction

transferFrom(sender, recipient, amount) - Transfers token from sender to recipient - Used in combination with approve

approve(recipient, amount) - Authorizes recipient to execute several transfers up to amount, from the address that executed the transaction

allowance(owner, spender) - Returns the remaining amount that the spender is approved to withdraw from the owner

Transfer event - Triggered upon successful transfer (call to transfer or transferFrom), even for 0 value transfers

Approval event - Logged upon successful call to approve

ERC20 functions

- **name()** - Returns a human-readable name for the token (i.e. “Ether”)
- **symbol()** - Returns a human-readable symbol for the token (i.e. “ETH”)
- **decimals()** - Returns the number of decimals used to divide token amounts - i.e. if decimals == 2, then the token is divided by 100 to get its user representation

ERC20 structures

All ERC20 contracts contain 2 data structures:

- **balances (**owner_address => balance_amount) - Allows the token contract to keep track of who owns the tokens - Each transfer is a deduction from one balance and an addition to another balance

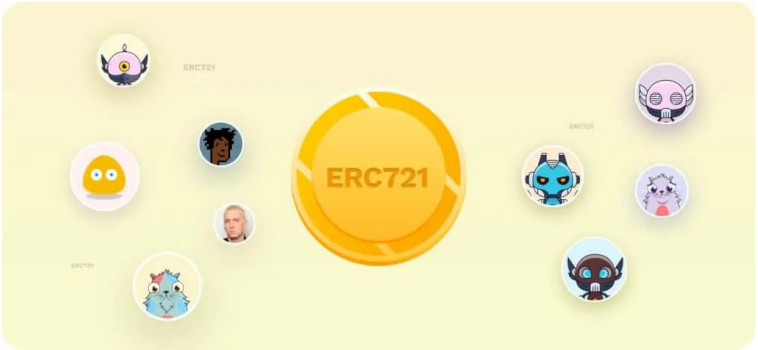
- **allowances** (owner_address => (spender_address => amount_allowed)) - In ERC20 tokens, an owner can delegate authority to a spender to spend a specific amount from their balance

Example

- Single-transaction, straightforward and simple, using the transfer() function
- Used by wallets to send tokens to other wallets
- Majority of token transactions happen with this workflow
- If Alice wants to send 10 tokens to Bob, her wallet sends a transaction to the token contract's address, with transfer(bobs_address, 10)
- The token contract adjusts Alice's balance (-10) and Bob's balance (+10) and issues a Transfer event

ERCERC721

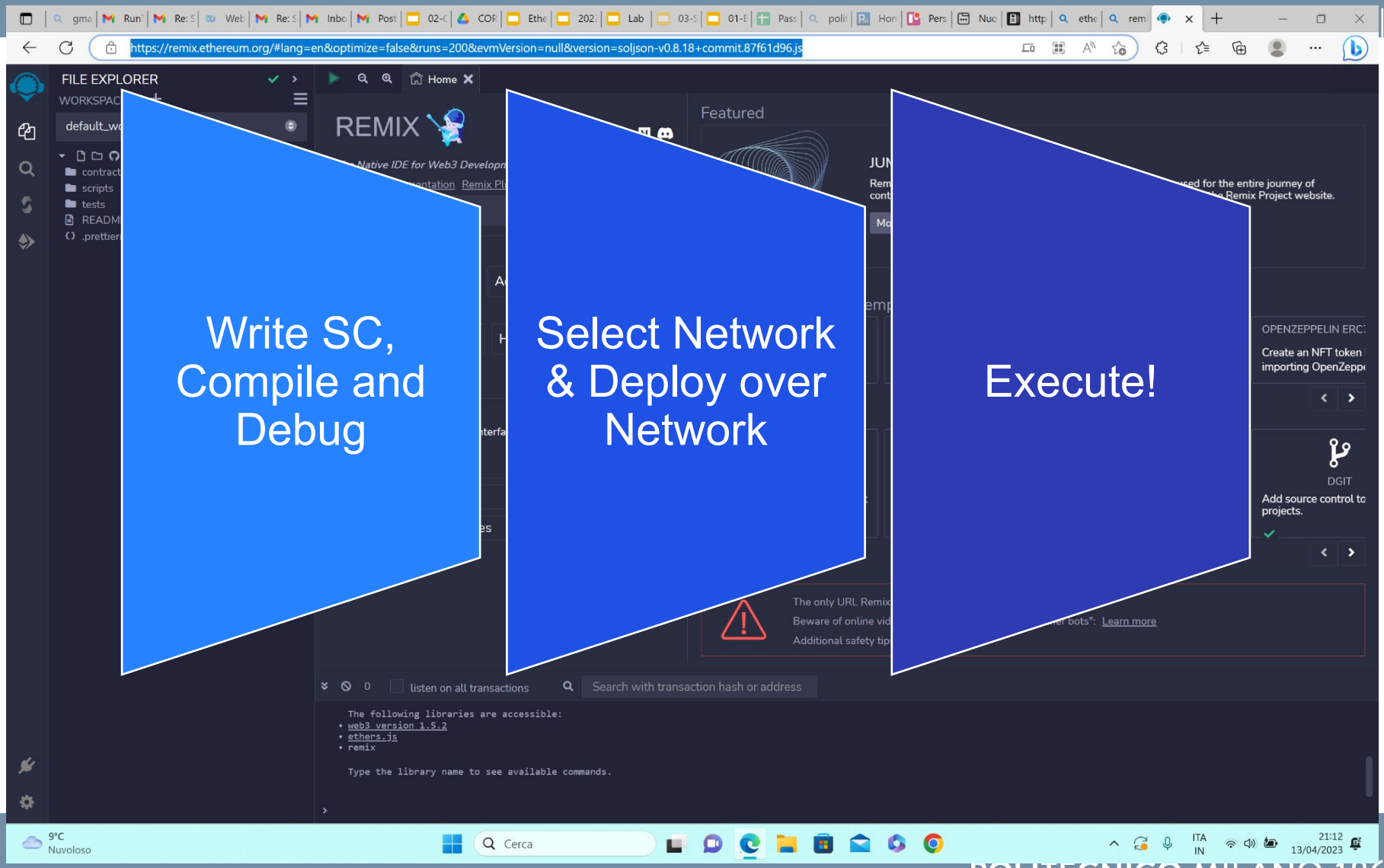
- ERC For **non fungible tokens**, more recent than ERC20
- It has been created in 2017 to manage "unique" digital items in the BC: every NFT token is unique and different to others
- The most famous examples is CriptoKitties
- It is widely adopted in collectibles and digital art
- Each ERC-721 token
  - Has its name and a short name
  - Has a proprietary set of features used to defined the properties of the token and its transferability (metadata field)
  - Has a field that authorized a third entity than the owner to trasfer the token



ERC721

- It provides functionalities
  - **transfer tokens from one account to another,** to
    - get the current token balance of an account,
    - get the owner of a specific token
    - Get the total supply of the token available on the network.
  - some **other functionalities** :
    - to approve that an amount of token from an account can be moved by a third party account.

Lab Remix (Remix - Ethereum IDE)



Solidity

- Solidity is an **object-oriented**, high-level language
- Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.
- When deploying contracts, you should use th**e latest released version** of Solidity.

.

Hello Coin

```
pragma solidity ^0.4.18;
contract HelloCoin {
string public name = 'HelloCoin';
//currency name. Please feel free to change it
string public symbol = 'coin_nadia';
//choose a currency symbol. Please feel free to change it
mapping (address => uint) balances;
//a key-value pair to store addresses and their account balances
event Transfer(address _from, address _to, uint256 _value);
// declaration of an event. Event will not do anything but add a record to the log
constructor() public {
//when the contract is created, the constructor will be called automatically
balances[msg.sender] = 10000;
//set the balances of creator account to be 10000. Please feel free to change it to any number you want.
}
function sendCoin(address _receiver, uint _amount) public returns(bool sufficient) {
if (balances[msg.sender] < _amount) return false;
// validate transfer
balances[msg.sender] -= _amount;
balances[_receiver] += _amount;
emit Transfer(msg.sender, _receiver, _amount);
// complete coin transfer an
return true;
}
function getBalance(address _addr) public view returns(uint) {
//balance check
return balances[_addr];
}
```

Hello Coin 2

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.4;

contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping(address => uint) public balances;

    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);

    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }

    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        balances[receiver] += amount;
    }

    // Errors allow you to provide information about
    // why an operation failed. They are returned
    // to the caller of the function.
    error InsufficientBalance(uint requested, uint available);

    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        if (amount > balances[msg.sender])
            revert InsufficientBalance({
                requested: amount,
                available: balances[msg.sender]
            });

        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```