Blockchain and Decentrilized Finance

Anatoly Krestenko

Vega

Sep 30, 2023

Plan

- Introduction into EVM
  - Ethereum Virtual Machine
  - State Machine
  - Architecture
  - Machine space
  - Execution model
- Introduction into Solidity
  - Introduction into Solidity
  - Project setup and IDEs
  - Contract testing: Hardhat vs. Foundry
- Best practices
  - ERC standards
  - Lending architecture: Aave V2
  - DEX architecture: Uniswap evolution
  - Proxy contract
- Homework: MyERC20 Uniswap V3 Pool

EVM: Ethereum Virtual Machine

- Ethereum is a distributed state machine
- The specific rules of changing state from block to block are defined by the EVM: state transition function: $Y(S, T) = S'$
- EVM code is executed on Ethereum Virtual Machine (EVM)
- The Ethereum Virtual Machine is the runtime environment for smart contracts in Ethereum

EVM: State Machine

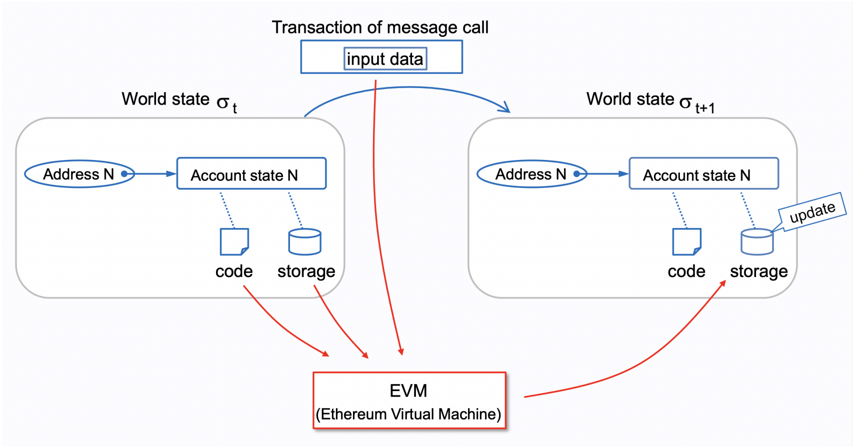

Figure 1: EVM States

EVM: Architecture

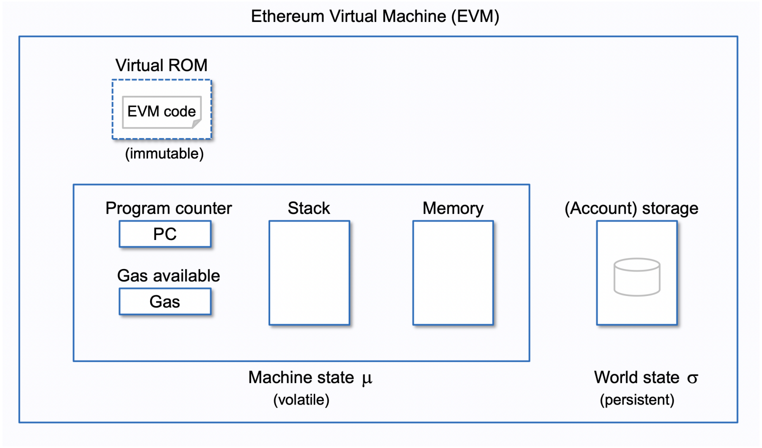

Figure 2: EVM Architecture

EVM: Machine space

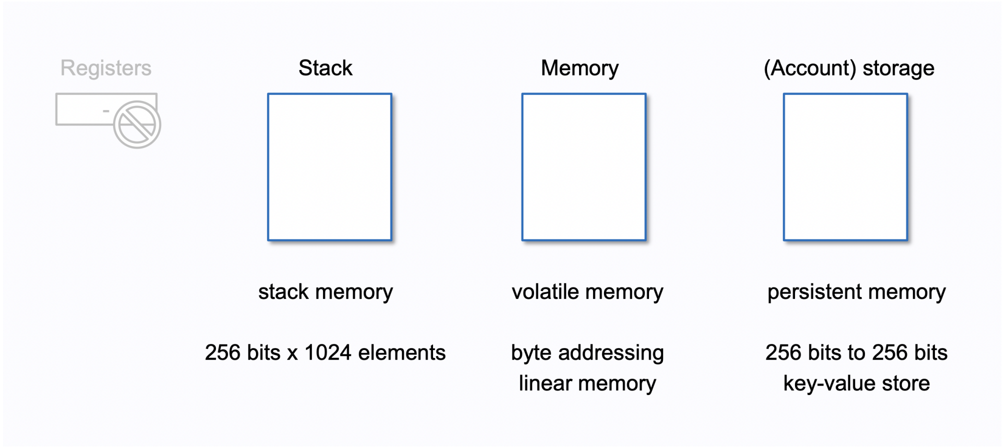

Figure 3: EVM Machine space

EVM: Execution model

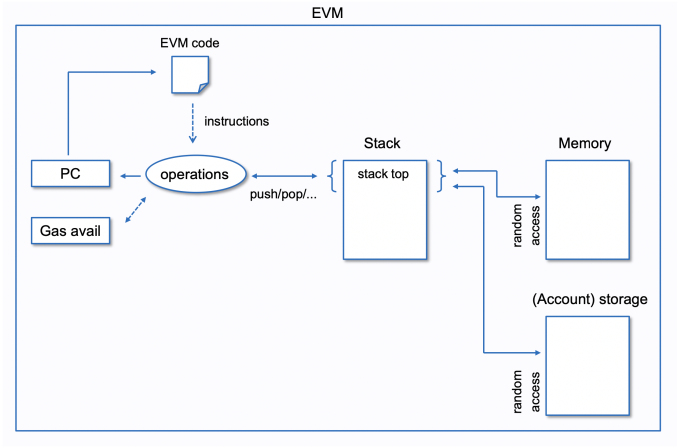

Figure 4: EVM Execution model

EVM: Practice

https://www.evm.codes

Introduction into Solidity

- Smart Contracts are computer programs stored on the blockchain that allow converting traditional contracts into digital parallels
- Solidity is a programming language for EVM smart contracts
  - Object-oriented, high-level language
  - Curly-bracket language that C++ has most profoundly influenced
  - Statically typed (the type of a variable is known at compile time)
  - Supports: Inheritance, Libraries, Complex user-defined types

Solidity primitives, data types and structures

Solidity Intro

- Solidity primitives and data types
- Logical structures: if/else, for/while loops
- Data structures: mapping, array, enum
- Functions and methods

Solidity contracts overview

Solidity contracts

- Constructor
- Inheritance: OOP
- Shadowing Inherited State Variables
- Visibility, Parent Contracts
- Interface
- Calling Other Contract

Project structure

Project structure

- Import
- Library
- Contract ABI
- Deploy

IDE: Remix

Remix - Remix IDE is an open-source web and desktop application. It fosters a fast development cycle and has a rich set of plugins with intuitive GUIs.

Contract testing

- Hardhat - is an environment developers use to test, compile, deploy and debug dApps based on the Ethereum blockchain. Contract testing example with hardhat.
- Foundry - Foundry is a smart contract development toolchain. Foundry manages your dependencies, compiles your project, runs tests, deploys, and lets you interact with the chain from the command-line and via Solidity scripts.

ERC Standards

Here are some of the most popular token standards on Ethereum:

- ERC-20 - A standard interface for fungible (interchangeable) tokens, like voting tokens, staking tokens or virtual currencies
- ERC-721 - A standard interface for non-fungible tokens, like a deed for artwork or a song
- ERC-777 - ERC-777 allows people to build extra functionality on top of tokens such as a mixer contract for improved transaction privacy or an emergency recover function to bail you out if you lose your private keys
- ERC-1155 - ERC-1155 allows for more efficient trades and bundling of transactions – thus saving costs. This token standard allows for creating both utility tokens (such as $BNB or$BAT) and Non-Fungible Tokens like CryptoPunks
- ERC-4626 - A tokenized vault standard designed to optimize and unify the technical parameters of yield-bearing vaults

ERC Standards implemetation

Most standards are implemented in openzeppelin solidity library

Aave V2 Architecture

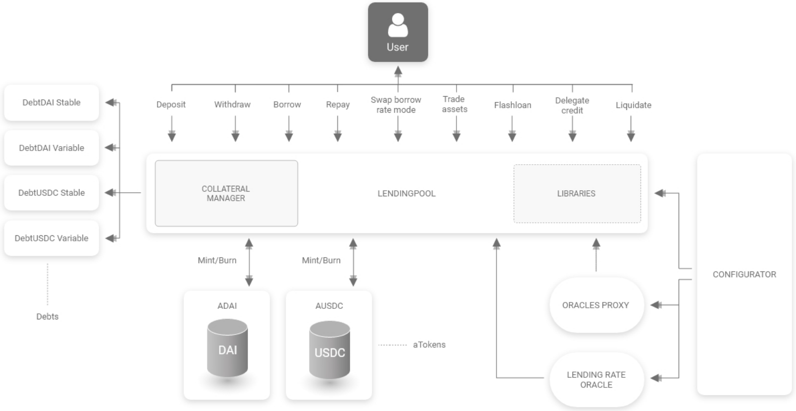

Figure 5

Uniswap V2 Architecture

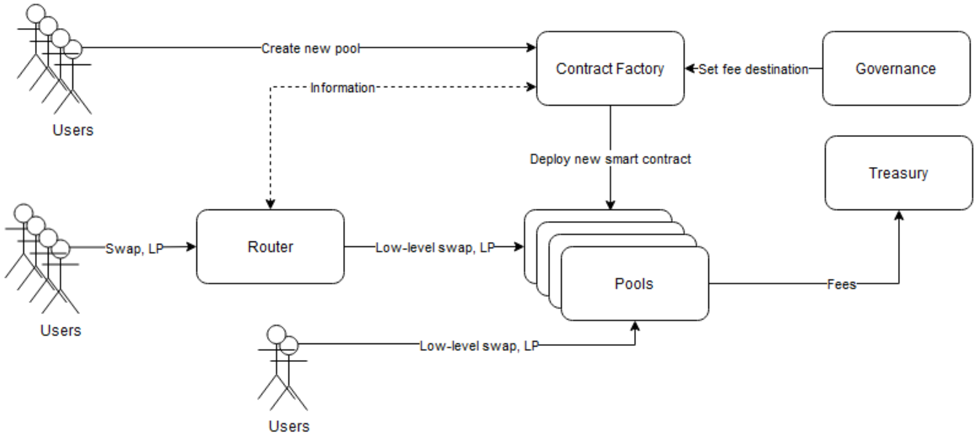

Figure 6

Uniswap V3 Architecture

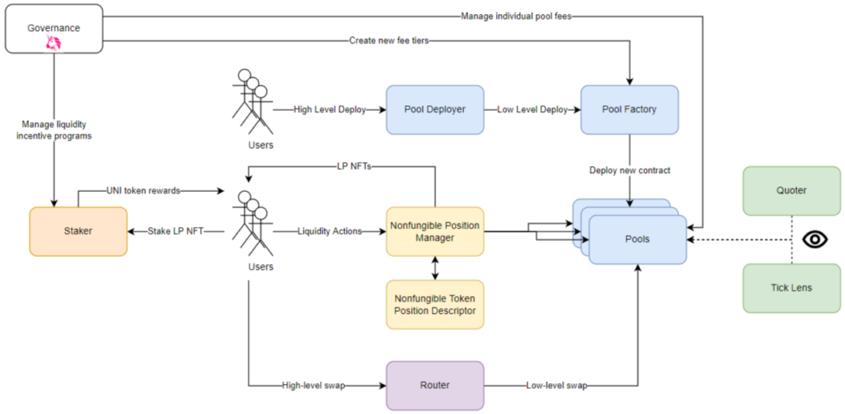

Figure 7

Uniswap V4 Proposal

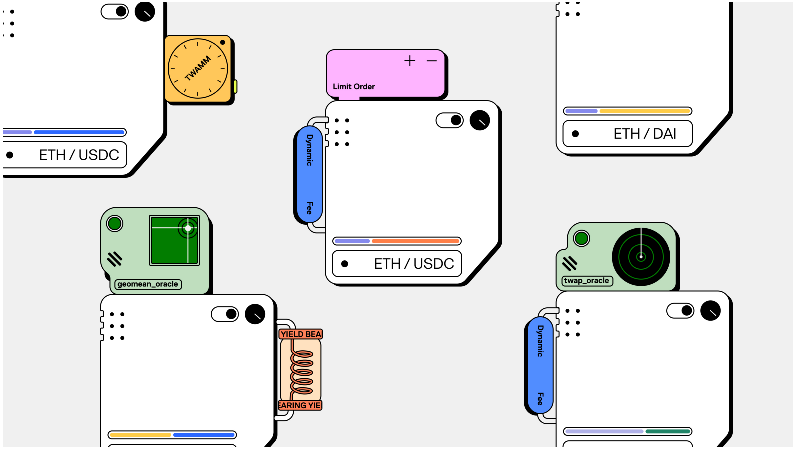

Figure 8

Proxy Pattern

A proxy architecture pattern is such that all message calls go through a Proxy contract that will redirect them to the latest deployed contract logic.

- Inherited Storage
- Eternal Storage
- Unstructured Storage

This pattern are also implemented by openzeppelin

Task

- Create and deploy your own ERC20 token that must include fields with your initials and ID from students sheet in name and symbol fields (for example AK0/AK0 Token)
- Create MyERC20/WETH Uniswap V3 pool
- Position 1: Chose a valid price range and deposit liquitity into this pool
- Position 2: Create an NFT position from this pool with one-tick range and this tick should be on the upper bound tick $(p_u + tickSpacing)$ of the Position 1
- Position 3: Choose a price step p and deposit liquidity into a symmetric price range $(p_u - p,\ p_u + p)$

You can use any tools from the course materials.

Instruments

- Solidity
- Remix
- Hardhat
- Foundry
- Metamask + Scanner
- Goerli

Checklist

Max grade is 10, each stage has a weight in the final result.

- ERC20 token - 3
  - Token is deployed on Goerli
  - ABI is verified and the token contract is modified
- Uniswap V3 pool - 3
  - Pool is initialized and deployed on Goerli
  - Pool has liquidity
- Uniswap V3 pool position - 4
  - Position 1
  - Position 2
  - Position 3

Ethereum Foundation
ETHEREUM VIRTUAL MACHINE .

Takenobu T.
Ethereum EVM illustrated .

Khan Academy
Solidity by Examples .

Ethereum Foundation
Solidity documentation.

Ethereum Foundation
Solidity documentation.

Remix
Remix IDE .

Hardhat
Hardhat tutorial .

FoundryBook
FoundryBook.

Openzeppelin
Openzeppelin docs.

Bowtiedisland
Uniswap V3 Overview.

Uniswap Labs Blog
Uniswap V4 Vision.