# SeaLevel

<https://medium.com/solana-labs/sealevel-parallel-processing-thousands-of-smart-contracts-d814b378192>

Solana use Sealevel like Ethereum use the EVM

Sealevel — Parallel Processing Thousands of Smart Contracts.

Solana’s parallel smart contracts runtime. Before we start, one thing to consider is that EVM and EOS’s WASM-based runtimes are all single threaded. That means that one contract at a time modifies the blockchain state. What we’ve built in Solana is a runtime that can process tens of thousands of contracts in parallel, using as many cores as are available to the Validator.

The reason why Solana is able to process transactions in parallel is that Solana transactions describe all the states a transaction will read or write while executing. This not only allows for non-overlapping transactions to execute concurrently, but also for transactions that are only reading the same state to execute concurrently as well.

## Programs and Accounts

Solana use accounts database. our accounts database, is a mapping of Public Keys to Accounts. Accounts maintain balances and data, where data is a vector of bytes. Accounts have an “owner” field. The owner is the Public Key of the program that governs the state transitions for the account. Programs are code and have no state. They rely on the data vector in the Accounts assigned to them for state transitions.

1. Programs can only change the data of accounts they own.
2. Programs can only debit accounts they own.
3. Any program can credit any account.
4. Any program can read any account.

By default, all accounts start as owned by the System Program.

1. System Program is the only program that can assign account ownership.
2. System Program is the only program that can allocate zero-initialized data.
3. Assignment of account ownership can only occur once in the lifetime of an account.

A user-defined program is loaded by the loader program. The loader program is able to mark the data in the accounts as executable. The user performs the following transactions to load a custom program:

1. Create a new public key.
2. Transfer coin to the key.
3. Tell System Program to allocate memory.
4. Tell System Program to assign the account to the Loader.
5. Upload the bytecode into the memory in pieces.
6. Tell Loader program to mark the memory as executable.

At this point, the loader verifies the bytecode, and the account to which the bytecode is loaded into can be used as an executable program. New Accounts can be marked as owned by the user-defined program.

The key insight here is that programs are code, and within our key-value store, there exists some subset of keys that the program and only that program has written access.

## Transactions

Transactions specify an instruction vector. Each instruction contains the program, program instruction, and a list of accounts the transaction wants to read and write. This interface is inspired by low level Operating System interfaces to devices.

Interfaces such as readv or writev tell the kernel ahead of time all the memory the user wants to read or write. This allows the OS to prefetch, prepare the device, and execute the operation concurrently if the device allows it.

On Solana, each instruction tells the VM which accounts it wants to read and write ahead of time. This is the root of our optimizations to the VM.

1. Sort millions of pending transactions.
2. Schedule all the non-overlapping transactions in parallel.

# Proof-Of-History (PoH)

Proof of History is the creation of Solana – a popular cryptocurrency. Solana combined proof of stake with proof of history and gave it a hybrid consensus algorithm.

Proof of history makes sure that the blockchain is extremely fast. But at the same time, it keeps its security top-notch and the network decentralized.

All the Solana events and transactions are hashed with the SHA256 hash function. As a result, Solana takes an input and processes a unique output which is extremely difficult to figure out.

Solana then takes the output of a transaction and uses it as input for the next hash. This sequence of transactions is now built into the hashed output.

This hashing process creates a long and unbroken chain of hashed transactions. Also, this characteristic creates a clear, verifiable sequence of transactions that a validator adds to a block without the need for a conventional timestamp.

# Install Solana

<https://www.youtube.com/watch?v=g2F9raiGp_s&ab_channel=LeetDev>

<https://docs.microsoft.com/en-us/windows/wsl/setup/environment>

Wsl --install

Cmd -> wsl (then update)

sudo apt update && sudo apt upgrade

(Install node in wsl)

<https://github.com/nodesource/distributions/blob/master/README.md>

Node.js LTS (v16.x):

**cmd1:** curl -fsSL https://deb.nodesource.com/setup\_lts.x | sudo -E bash -

sudo apt-get install -y nodejs

**cmd2:** apt-get install -y build-essential

**if error then:** <https://stackoverflow.com/questions/55464934/npm-depends-node-gyp-3-6-2-but-it-is-not-going-to-be-installed>

**cmd3:** apt-get install nodejs yarn

(<https://rustup.rs/> )

**install rust:** curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

(<https://docs.solana.com/cli/install-solana-cli-tools> )

**install solana:** sh -c "$(curl -sSfL <https://release.solana.com/v1.10.6/install>)"

**Phantom Wallet:** <https://phantom.app/>

(<https://docs.solana.com/running-validator/validator-start#system-tuning> )

**Extension for VS Code for WSL:**

(<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack> )

Remote Development

**Solana Validator Running:** first delete “”directory -> cd ~ -> ls ->

**cmd:** sudo $(command -v solana-sys-tuner) --user $(whoami) > sys-tuner.log 2>&1 &

# Anchor

<https://learn.figment.io/tutorials/build-a-blog-dapp-using-anchor>

<https://project-serum.github.io/anchor/tutorials/tutorial-0.html#clone-the-repo>

<https://project-serum.github.io/anchor/tutorials/tutorial-1.html#creating-and-initializing-accounts>

<https://project-serum.github.io/anchor/tutorials/tutorial-2.html#defining-a-program>

<https://project-serum.github.io/anchor/tutorials/tutorial-3.html#return-values>

<https://project-serum.github.io/anchor/tutorials/tutorial-4.html#using-the-client> l

<https://project-serum.github.io/anchor/cli/commands.html#build>

## Install Rust

* curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
* source $HOME/.cargo/env
* rustup component add rustfmt

## Install Solana

* sh -c "$(curl -sSfL https://release.solana.com/v1.9.1/install)"

## Install Yarn

* npm install -g yarn

## Install Anchor on x86\_64 Linux

* npm i -g @project-serum/anchor-cli

## Install additional dependencies

* sudo apt-get update && sudo apt-get upgrade && sudo apt-get install -y pkg-config build-essential libudev-dev
* anchor --version

## Start a Project

* anchor init [project name]
* anchor build
* anchor deploy

## Solana (Basics Commands)

1. solana config set --url localhost

2. solana account

3. solana balance <address>

4. solana airdrop <amt> <address>

5. solana-test-validator

Window commands:

curl https://release.solana.com/v1.10.6/solana-install-init-x86\_64-pc-windows-msvc.exe --output C:\solana-install-tmp\solana-install-init.exe --create-dirs

# Hello world (clone repo)

Solana config set - -url localhost (start local network)

Solana-kaygen new (create vault for store sol)

Solana-test-validator

Npm run build:program-rust (allow cargo to build the program)

Solana program deploy dist/program/helloworld.so

Npm install (install dependency)

Npm run start

# Create New solana project

solana-keygen new --outfile ~/solana\_airdrop\_project/airdrom/vulit.json

Cat /[json path and copy byte array and creat new account on pantom]

Use: <https://explorer.solana.com/>

Solana airdrop 1

npm init -y

npm i --save-dev typescript

vi tsconfig.json [:wq]

mkdir airdrop

cd airdrop/

vi index.ts [:wq]

npm i --save @solana/web3.js

past in packeg.json: [scripts: {“build”: “npx tsc”}]

npm run build

node dist/index.js

# New solana token project

npm i --save "@solana/spl-token"

npm install @solana/spl-token@0.1.8

npm run build && node /dist/helloworld\_client/client.js

Anchor

anchor init [project name]

anchor build

anchor deploy

# Solana Commands

solana-keygen grind --starts-with sa:1

solana-keygen new - - k

solana config set -k [wallet address for set default wallet]

solana account [account id]

solana logs

solana logs | grap “[program or account Id which logs we want to see] invoke” -A 3