Lesson 2 - Blockchain Theory 2 / Solidity

Consensus on Ethereum

From Ethereum developer documentation
"Now technically, proof-of-work and proof-of-stake are not consensus protocols by themselves, but they are often referred to as such for simplicity. They are actually Sybil resistance mechanisms and block author selectors; they are a way to decide who is the author of the latest block. It's this Sybil resistance mechanism combined with a chain selection rule that makes up a true consensus mechanism."

There are 2 parts to block addition :

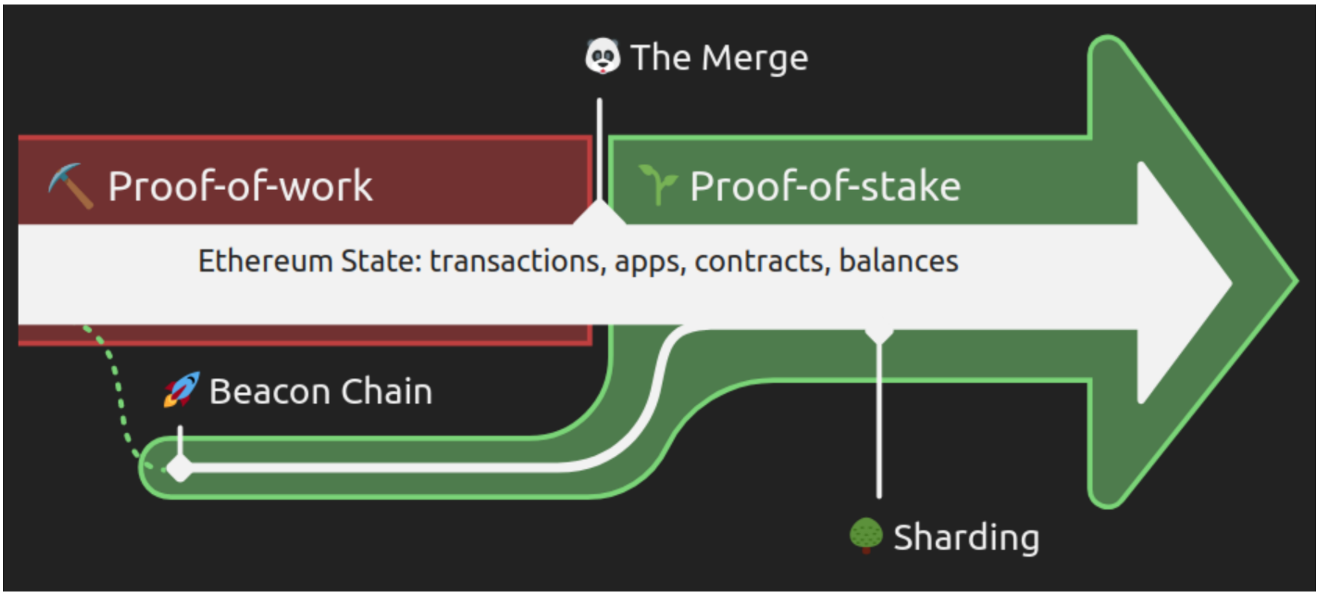
- block producer selection
- block acceptance

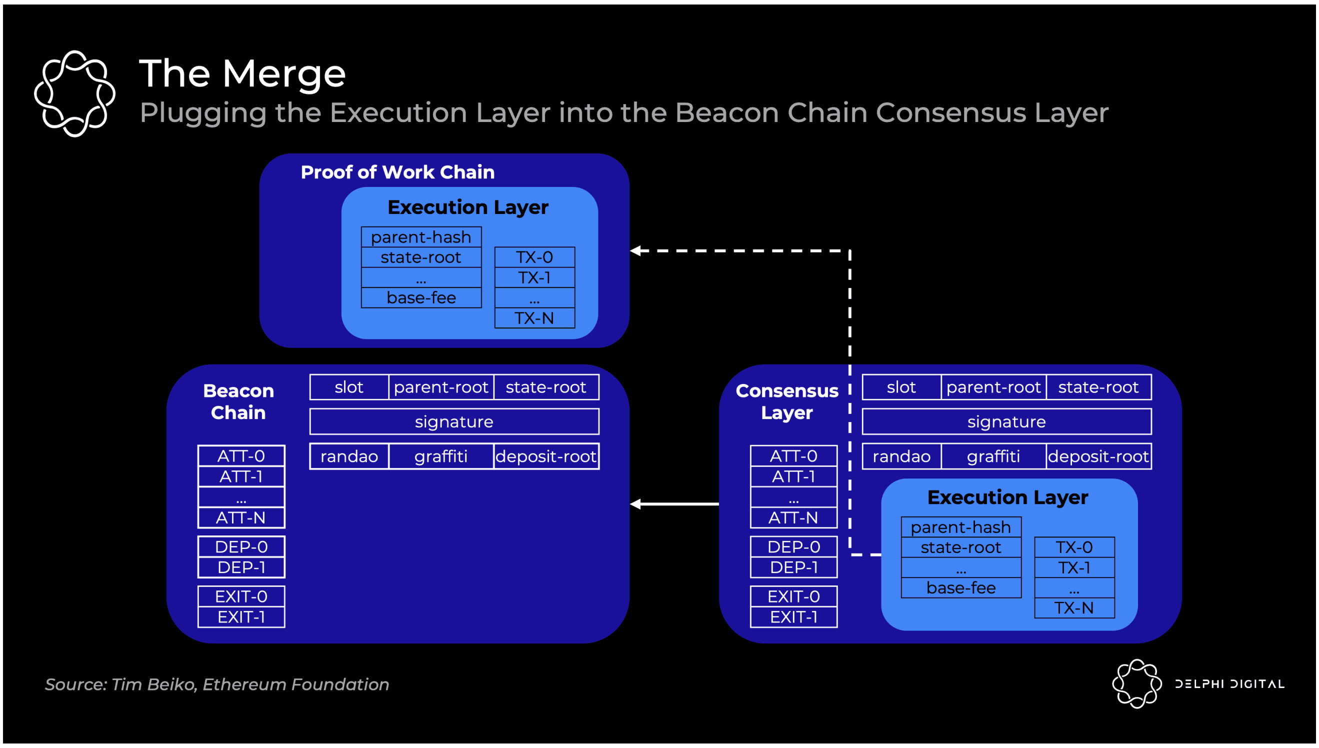
From yellow paper
" Since the system is decentralised and all parties have an opportunity to create a new block on some older pre-existing block, the resultant structure is necessarily a tree of blocks.
In order to form a consensus as to which path, from root (the genesis block) to leaf (the block containing the most recent transactions) through this tree structure, known as the blockchain, there must be an agreed-upon scheme.
If there is ever a disagreement between nodes as to which root-to-leaf path down the block tree is the ‘best’ blockchain, then a fork occurs."

The Merge - Proof of stake update

- Replacing proof of work with the proof of stake beacon chain.
  i.e. merging existing beacon chain into ethereum.
- The Beacon Chain has not been processing Mainnet transactions. Instead, it has been reaching consensus on its own state by agreeing on active validators and their account balances.
- The blockchain state will not change.
- POS specs: https://github.com/ethereum/consensus-specs#phase-0

"The number `58750000000000000000000` joins the list of the most important integers of our time. The moment Ethereum's proof of work chain accumulates that much difficulty (read, hashes done), the entire network will switch over to proof of stake"





The Merge

Plugging the Execution Layer into the Beacon Chain Consensus Layer

Proof of Work Chain

Execution Layer

parent-hash
state-root
...
base-fee

TX-0
TX-1
...
TX-N

Beacon Chain

slot | parent-root | state-root
signature
randao | graffiti | deposit-root

ATT-0
ATT-1
...
ATT-N
DEP-0
DEP-1
EXIT-0
EXIT-1

Consensus Layer

slot | parent-root | state-root
signature
randao | graffiti | deposit-root

ATT-0
ATT-1
...
ATT-N
DEP-0
DEP-1
EXIT-0
EXIT-1

Execution Layer

parent-hash
state-root
...
base-fee

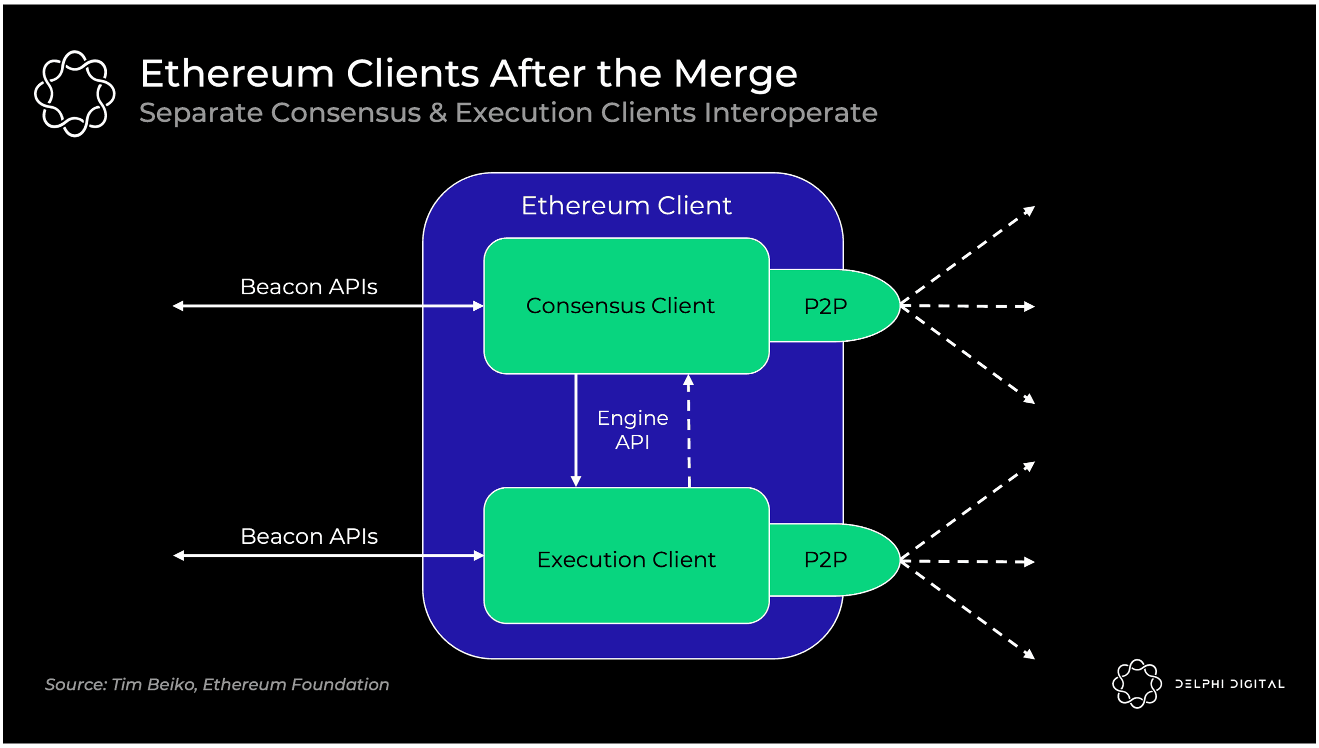
TX-0
TX-1
...
TX-N

Source: Tim Beiko, Ethereum Foundation

DELPHI DIGITAL

Ethereum clients after the merge

- Current Eth 1.0 clients continue to handle execution. They process blocks, maintain mempools, and manage and sync state. The PoW stuff gets ripped out.
- Consensus client – Current Beacon Chain clients continue to handle PoS consensus. They track the chain's head, gossip and attest to blocks, and receive validator rewards.



Ethereum Clients After the Merge

Separate Consensus & Execution Clients Interoperate

Source: Tim Beiko, Ethereum Foundation

Consensus after the Merge

Ethereum has moved to Gasper (Casper FFG + LMD GHOST (Latest Message Driven Greediest Heaviest Observed SubTree))

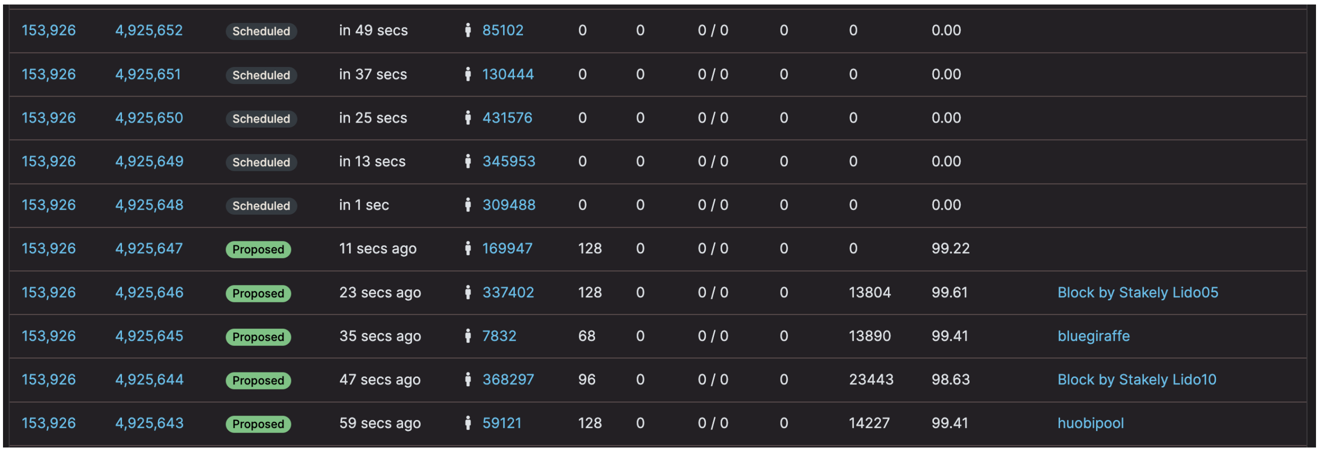
Consensus relies on both LMD-GHOST – which adds new blocks and decides what the head of the chain is – and Casper FFG which makes the final decision on which blocks *are* and *are not* a part of the chain.
GHOST's favourable liveness properties allow new blocks to quickly and efficiently be added to the chain, while FFG follows behind to provide safety by finalising epochs.

The two protocols are merged by running GHOST from the last finalised block as decided upon by FFG. By construction, the last finalised block is always a part of the chain which means GHOST doesn't need to consider earlier blocks.

Safety favouring protocols such as Tendermint can halt, if they don't get enough votes.
Liveness favouring protocols such as Nakamoto continue to add blocks, but they may not coe to finality.
Ethereum will achieve finality by checkpointing
Epochs of about 6 mins have 32 slots with all validators attesting to one slot (~12K attestations per block)
The fork-choice rule LMD GHOST then determines the current head of chain based on these attestations.
Finality is achieved when sufficient votes are reached generally after 2 epochs.

Validator Selection and consensus in more detail

You can find stats about the validators here

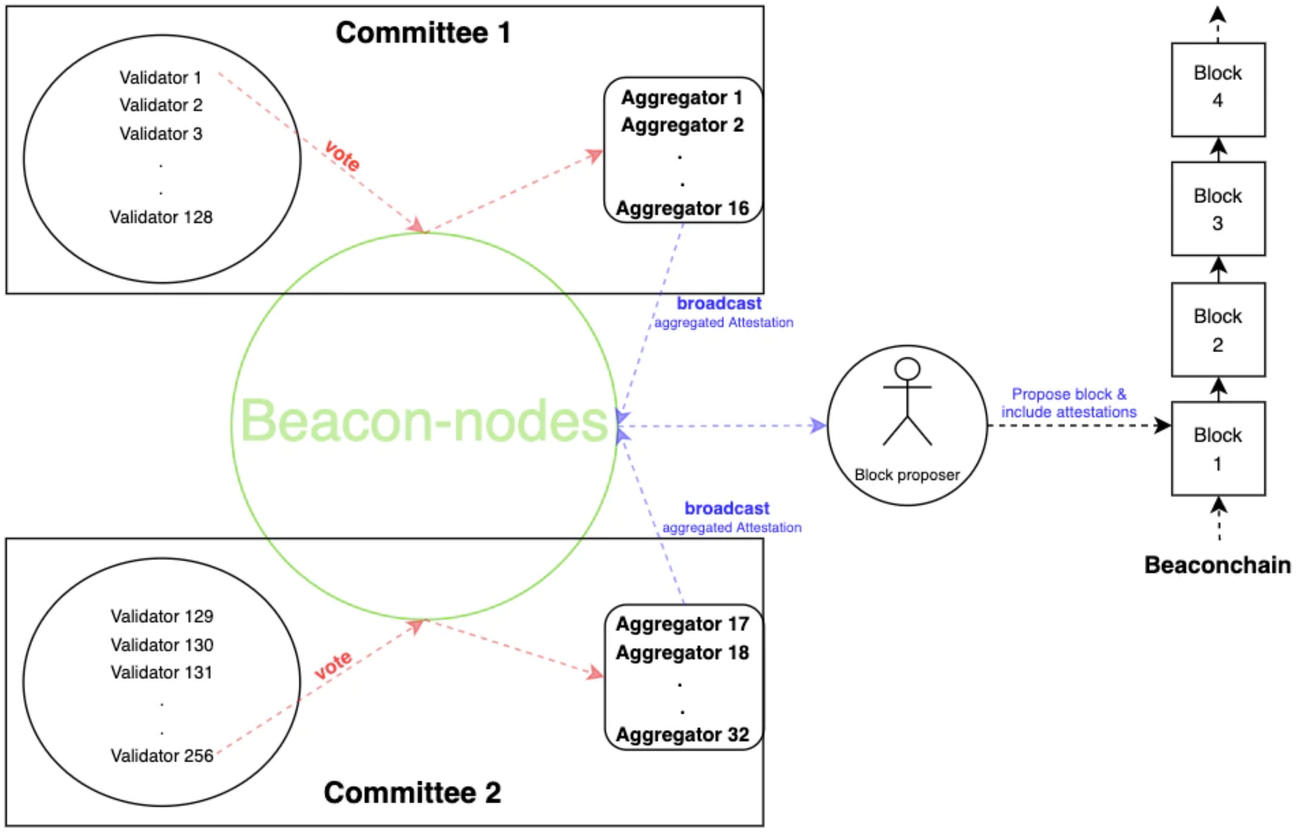


153,926	4,925,652	Scheduled	in 49 secs	85102	0	0	0 / 0	0	0	0.00	
153,926	4,925,651	Scheduled	in 37 secs	130444	0	0	0 / 0	0	0	0.00	
153,926	4,925,650	Scheduled	in 25 secs	431576	0	0	0 / 0	0	0	0.00	
153,926	4,925,649	Scheduled	in 13 secs	345953	0	0	0 / 0	0	0	0.00	
153,926	4,925,648	Scheduled	in 1 sec	309488	0	0	0 / 0	0	0	0.00	
153,926	4,925,647	Proposed	11 secs ago	169947	128	0	0 / 0	0	0	99.22	
153,926	4,925,646	Proposed	23 secs ago	337402	128	0	0 / 0	0	13804	99.61	Block by Stakely Lido05
153,926	4,925,645	Proposed	35 secs ago	7832	68	0	0 / 0	0	13890	99.41	bluegiraffe
153,926	4,925,644	Proposed	47 secs ago	368297	96	0	0 / 0	0	23443	98.63	Block by Stakely Lido10
153,926	4,925,643	Proposed	59 secs ago	59121	128	0	0 / 0	0	14227	99.41	huobipool

A slot occurs every 12s and one validator is chosen to submit a block within that slot.
If the validator fails to do so, the slot is let empty.
The first block within an epoch (6.4 mins) is a checkpoint block.

Coming to consensus about the block

If a validator is not chosen to produce a block, it will instead vote on what it regards as the current head of the chain and the checkpoint block. Within an epoch a validator will only vote once.



Validators are grouped into committees at random, their votes are aggregated and published in the block header.

This article gives an in depth view of validator rewards

Fork choice rule

When faced with a potential fork, we choose the fork that has the most votes, but when counting the votes, we only include the last one from any validator.

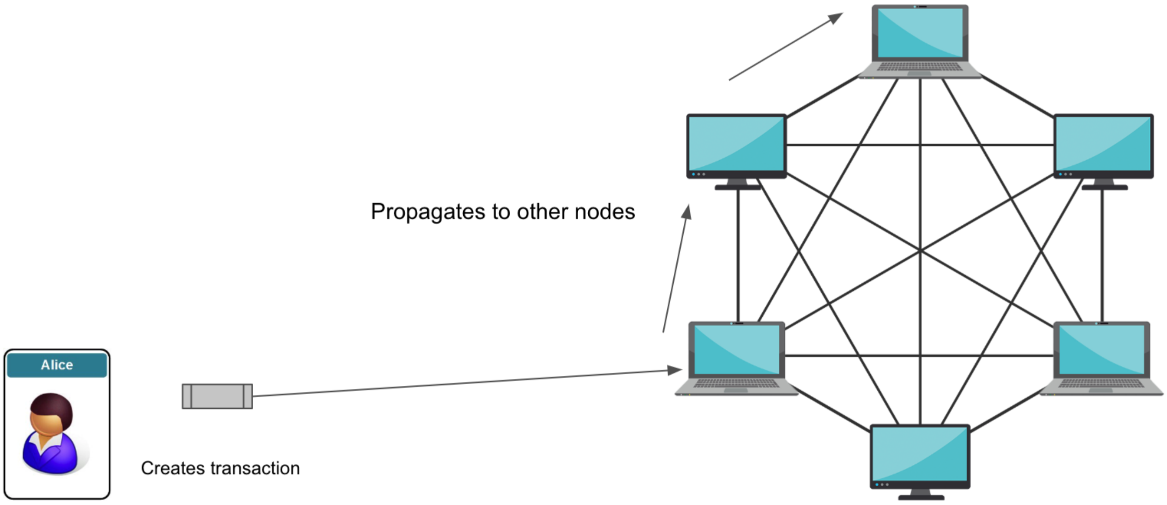
Voting rules

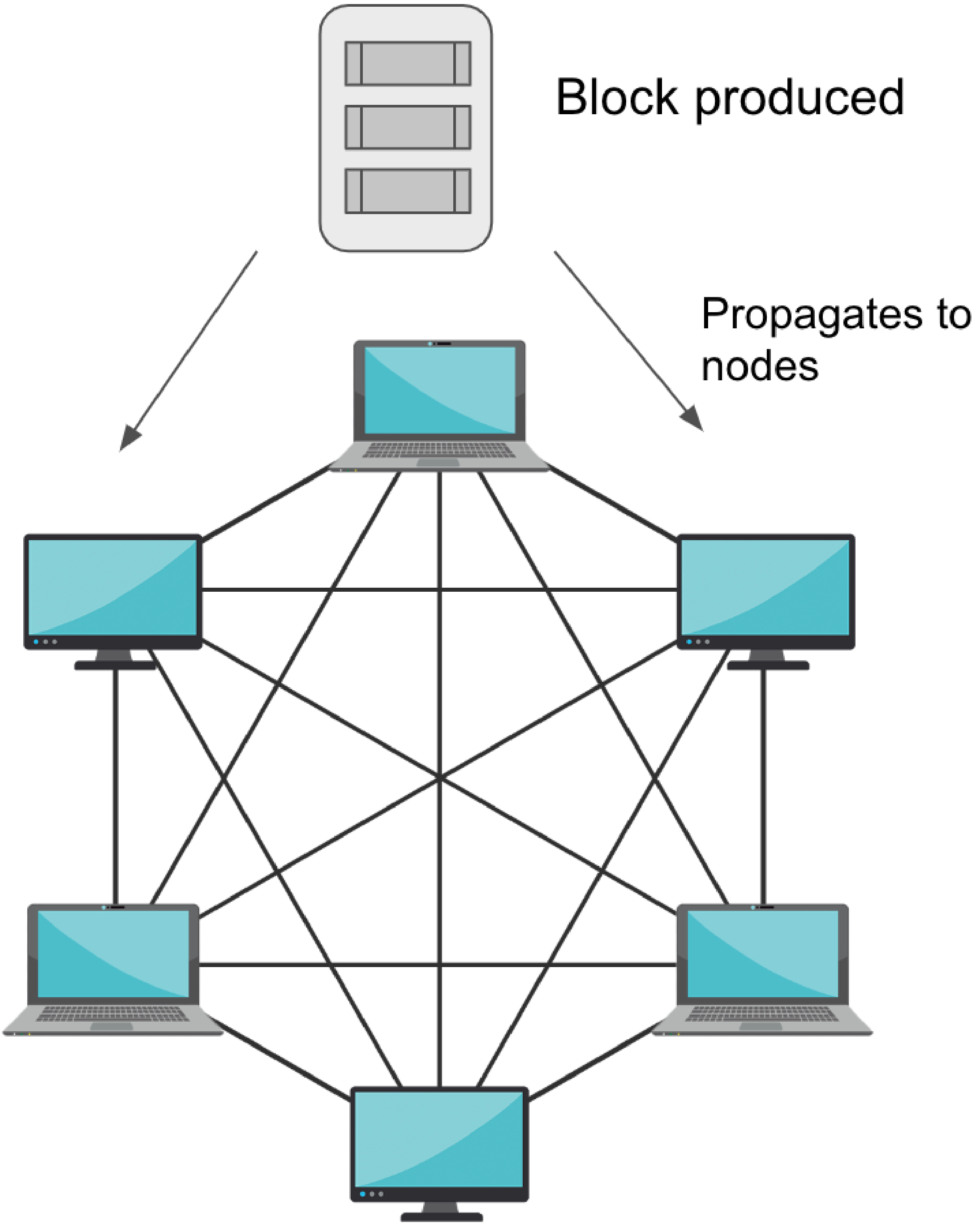
The validator must vote the chain they consider to be correct
The validator cannot vote for 2 blocks in any one time slot.

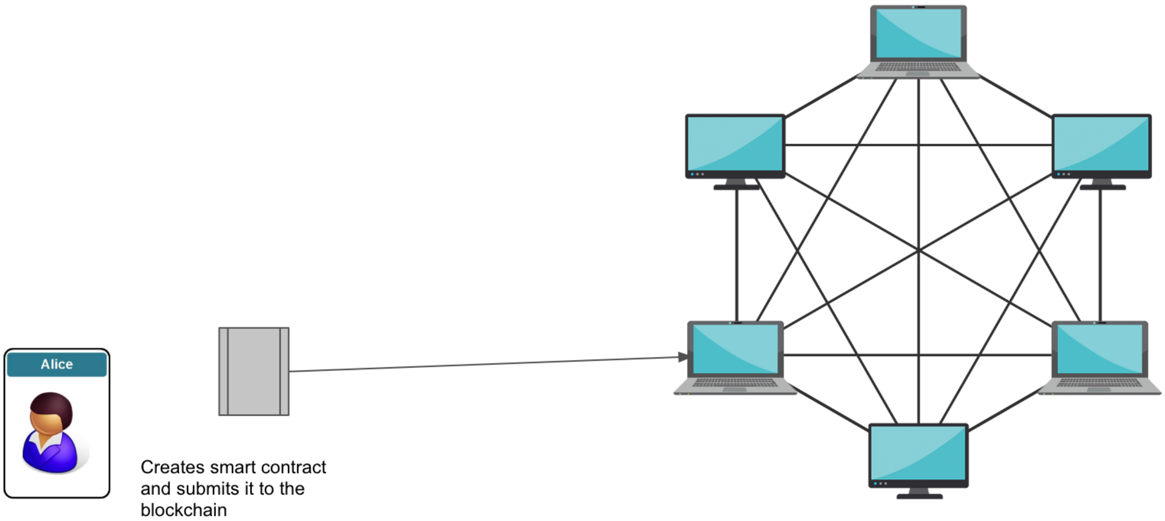
Finalisation

Validators vote on a pair of checkpoint blocks, to indicate that they are valid.
Once a checkpoit block gets sufficient votes, it is regarded as 'justified' once it's child checkpoint block becomes justified, then the parent is regarded as final.

Blocks and Transactions

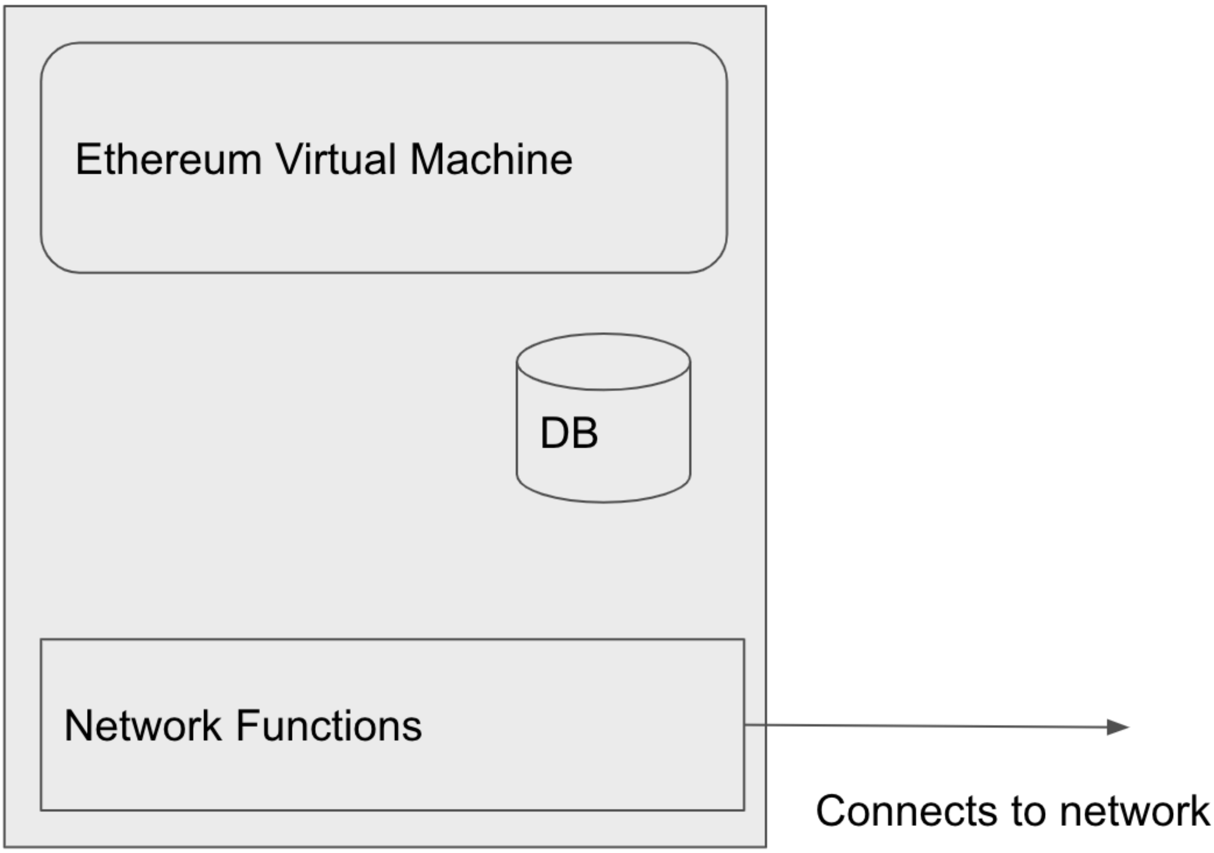



Block produced
Propagates to nodes



Creates smart contract
and submits it to the
blockchain

Ethereum Virtual Machine



Question : Where does the processing of a contract happen ?

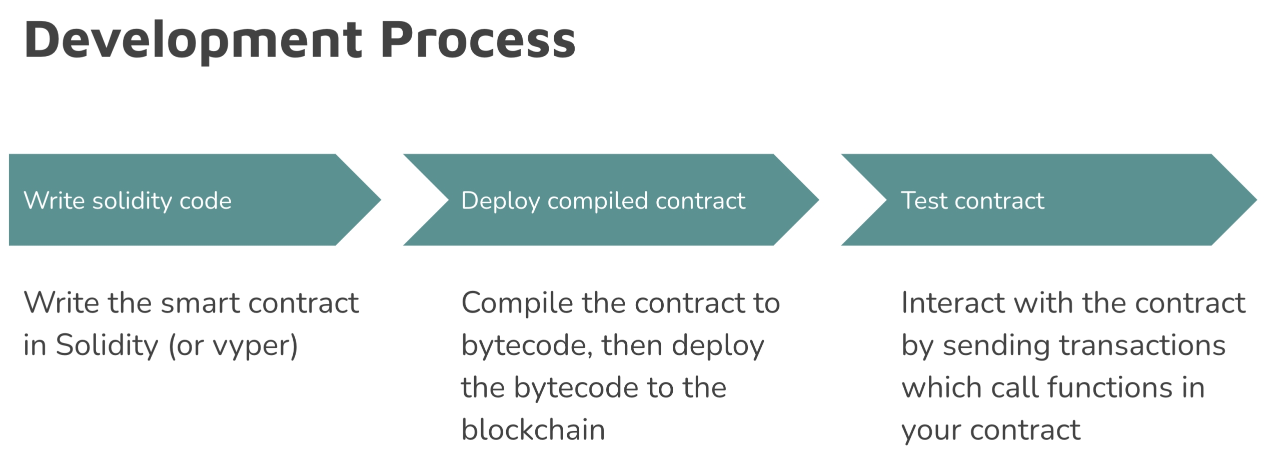
Question : Where is the state of the system stored ?

Blockchain Explorers

The most popular explorer is Etherscan
This displays
Blocks
Transactions
Contract code (if verified)
Statistics about the network.

There is also beacon chain

Developing smart contracts



Development Process

Write the smart contract in Solidity (or vyper)

Compile the contract to bytecode, then deploy the bytecode to the blockchain

Interact with the contract by sending transactions which call functions in your contract

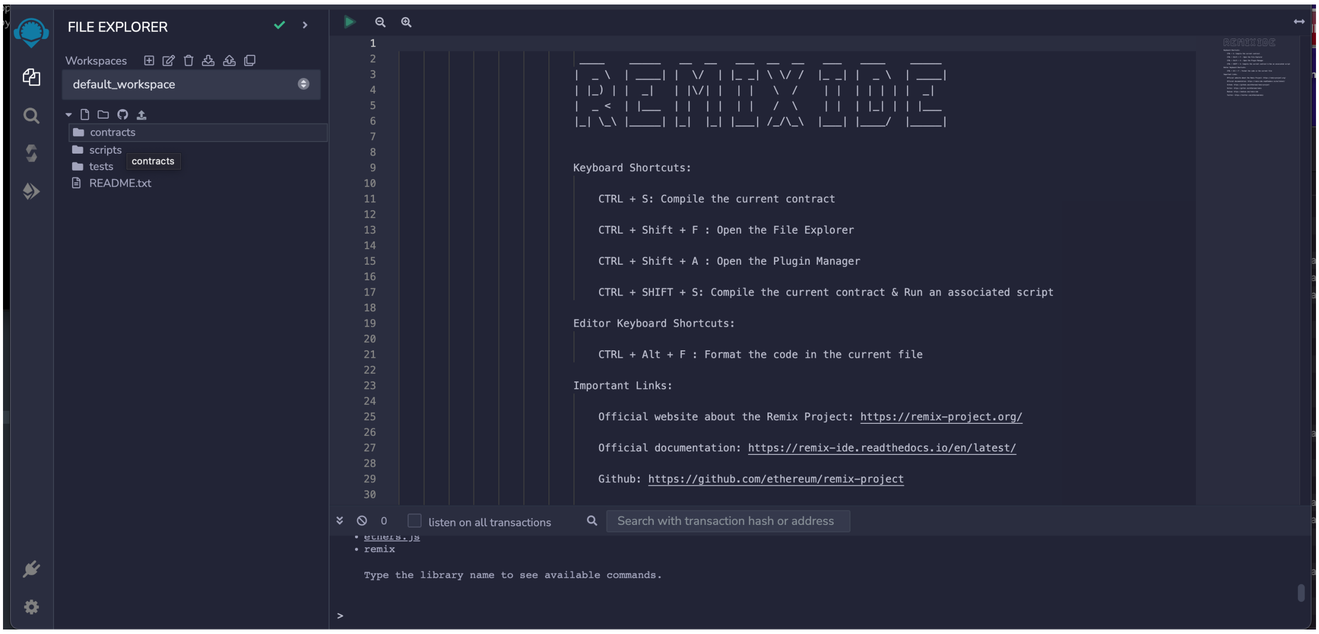
There are many tools / IDEs to help you develop contracts, such as Truffle, Hardhat, Foundry, and plugins for VS Code. We will start with one of the simplest : Remix.

Introduction to Remix

Remix is an IDE for Ethereum development.
It can run in the browser, or on the desktop.

Browser : https://remix.ethereum.org/
Desktop : `npm install -g @remix-project/remixd`

Remix documentation

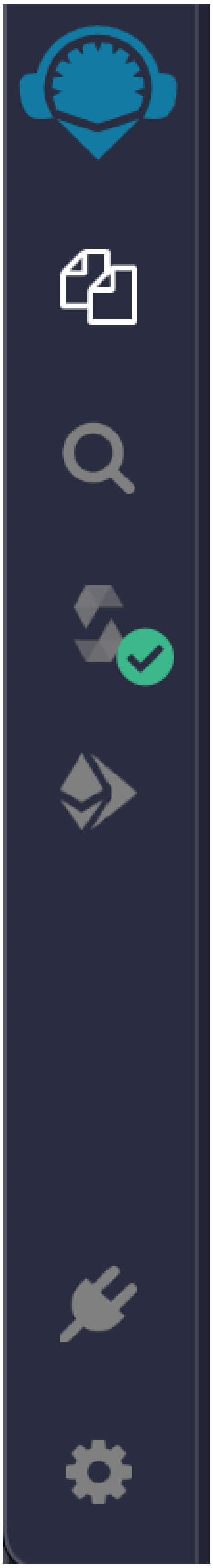


```
 ____    _____   __  __   ___  __  __   ___   ____    _____
|  _ \  | ____| |  \/  | |_ _| \ \/ /  |_ _| |  _ \  | ____|
| |_) | |  _|   | |\/| |  | |   \  /    | |  | | | | |  _|
|  _ <  | |___  | |  | |  | |   /  \    | |  | |_| | | |___
|_| \_\ |_____| |_|  |_| |___| /_/\_\  |___| |____/  |_____|


Keyboard Shortcuts:

    CTRL + S: Compile the current contract

    CTRL + Shift + F : Open the File Explorer

    CTRL + Shift + A : Open the Plugin Manager

    CTRL + SHIFT + S: Compile the current contract & Run an associated script

Editor Keyboard Shortcuts:

    CTRL + Alt + F : Format the code in the current file

Important Links:

    Official website about the Remix Project: https://remix-project.org/

    Official documentation: https://remix-ide.readthedocs.io/en/latest/

    Github: https://github.com/ethereum/remix-project

```

```
  • ethers.js
  • remix

  Type the library name to see available commands.


>
```

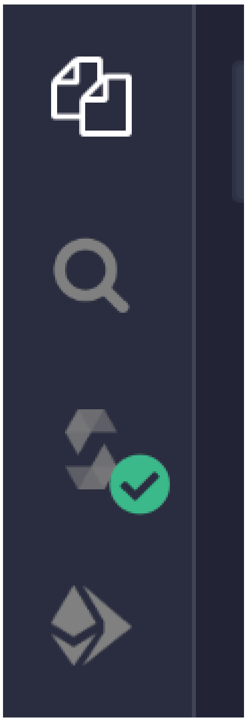
Plugins

The panel on the left allows us to add plugins



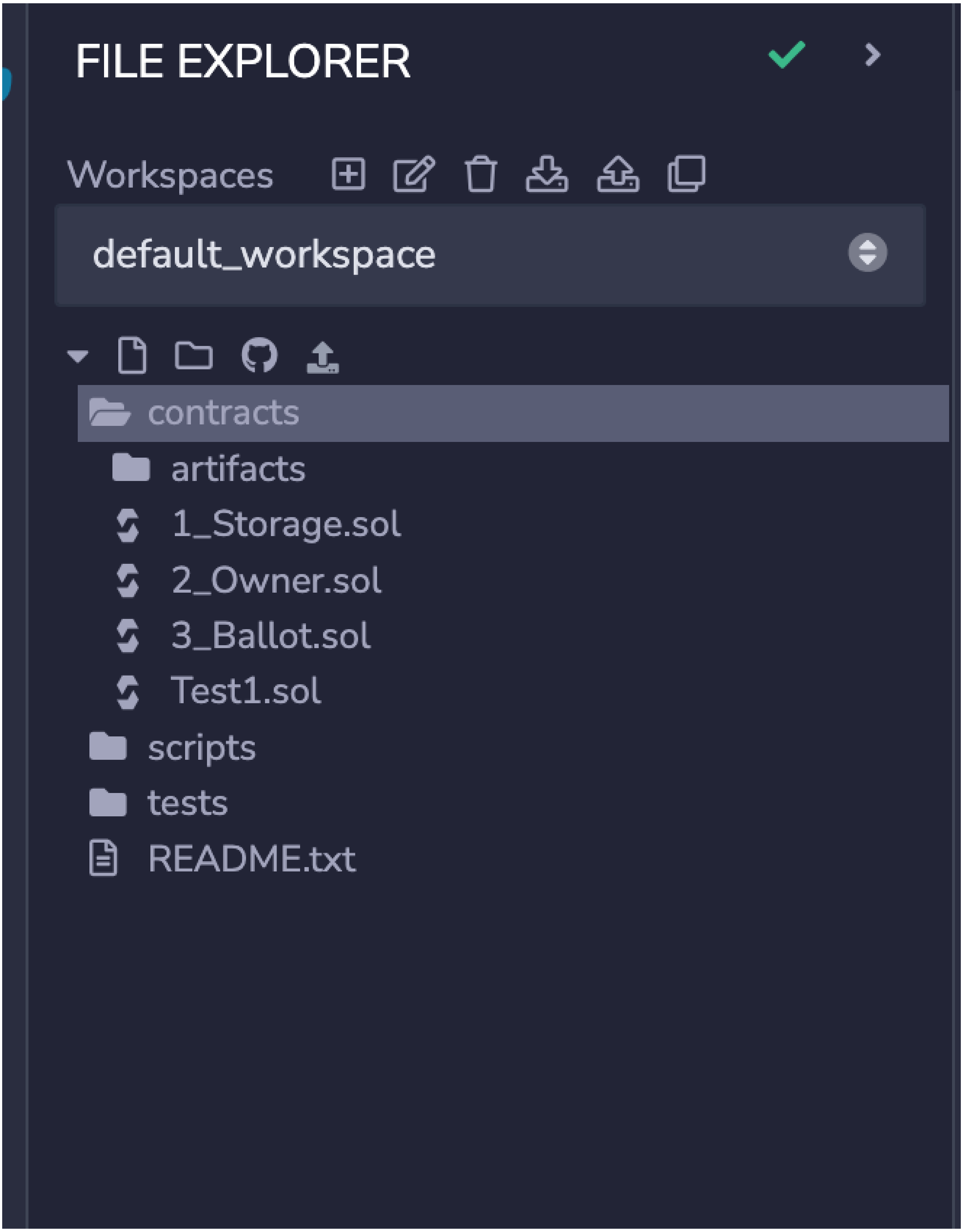
File Explorer

If we select the file icon in the plugin panel



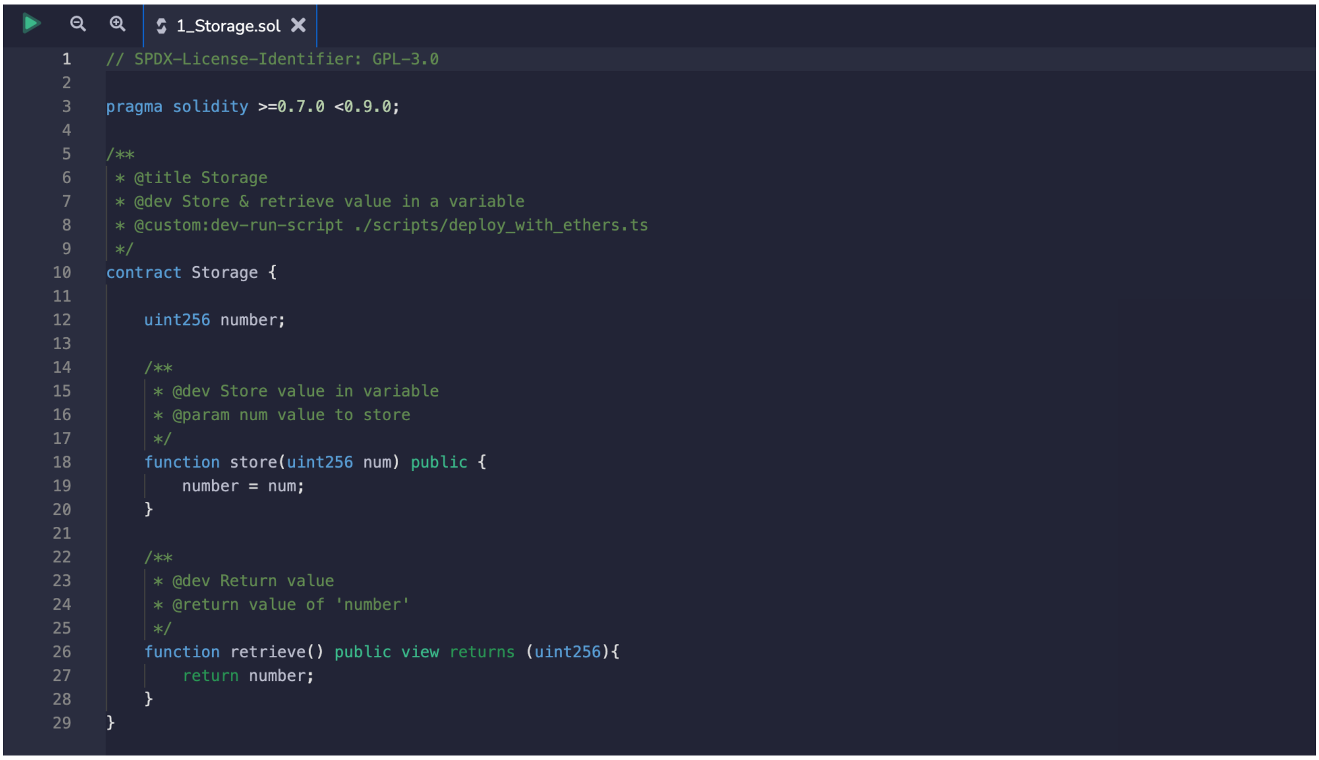
we will see the file explorer in the middle panel

The middle panel is the file explorer, here we can navigate between the contracts we are working on.



Editor Panel

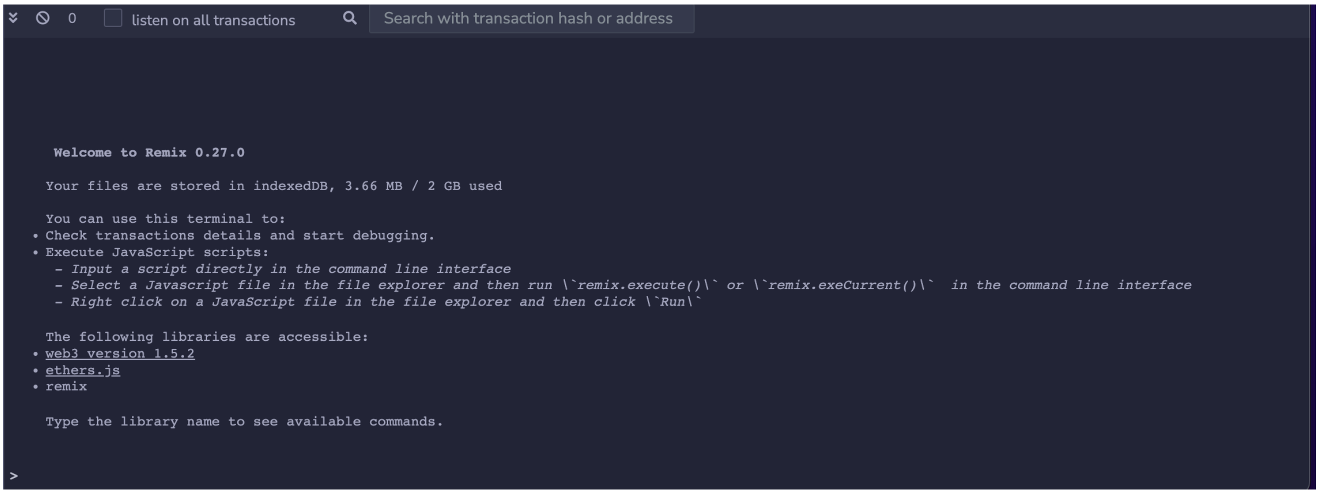
Here we edit our smart contracts



```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

/**
 * @title Storage
 * @dev Store & retrieve value in a variable
 * @custom:dev-run-script ./scripts/deploy_with_ethers.ts
 */
contract Storage {

    uint256 number;

    /**
     * @dev Store value in variable
     * @param num value to store
     */
    function store(uint256 num) public {
        number = num;
    }

    /**
     * @dev Return value
     * @return value of 'number'
     */
    function retrieve() public view returns (uint256){
        return number;
    }
}
```

Ouptut panel

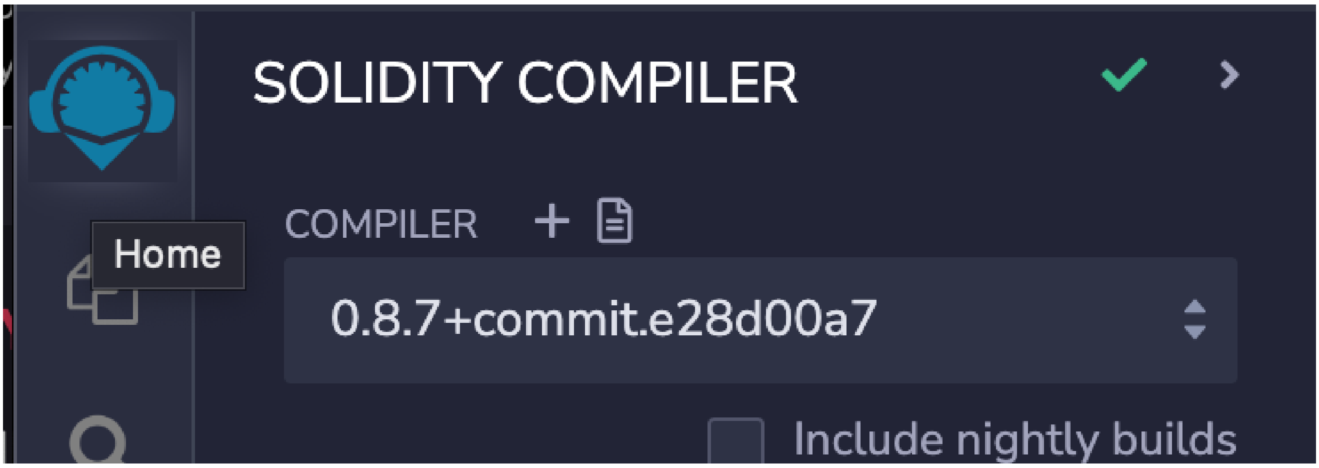
This shows the activity on our local blockchain and the results of our transactions


0
□ listen on all transactions
Search with transaction hash or address

```
Welcome to Remix 0.27.0

Your files are stored in indexedDB, 3.66 MB / 2 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run \`remix.execute()\` or \`remix.exeCurrent()\`  in the command line interface
  - Right click on a JavaScript file in the file explorer and then click \`Run\`

The following libraries are accessible:
• web3 version 1.5.2
• ethers.js
• remix

Type the library name to see available commands.

>
```

Compiling the contract

Chosing the compile icon in the left hand panel will display the compiler panel

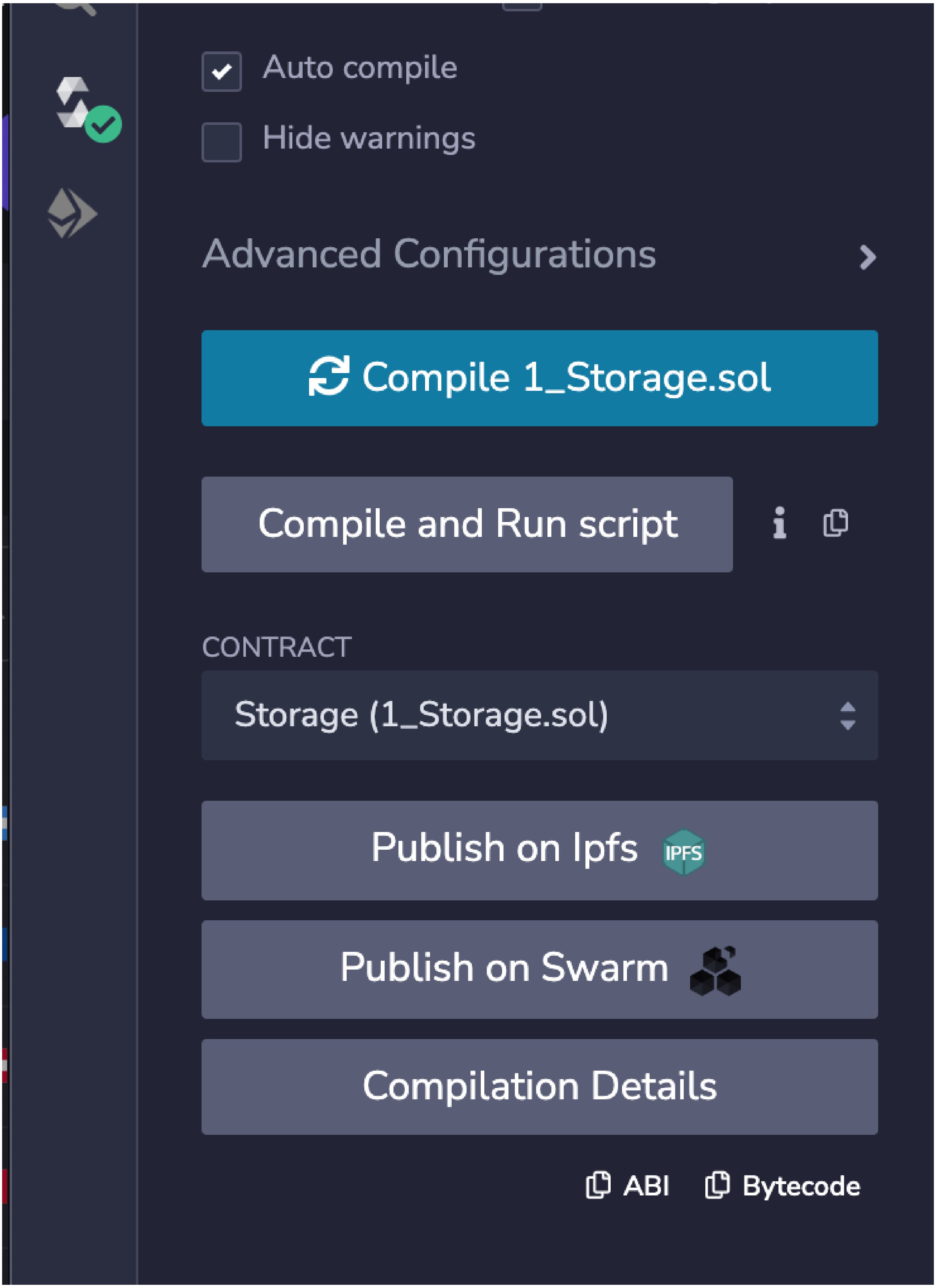


SOLIDITY COMPILER

COMPILER

0.8.7+commit.e28d00a7

□ Include nightly builds



☑ Auto compile

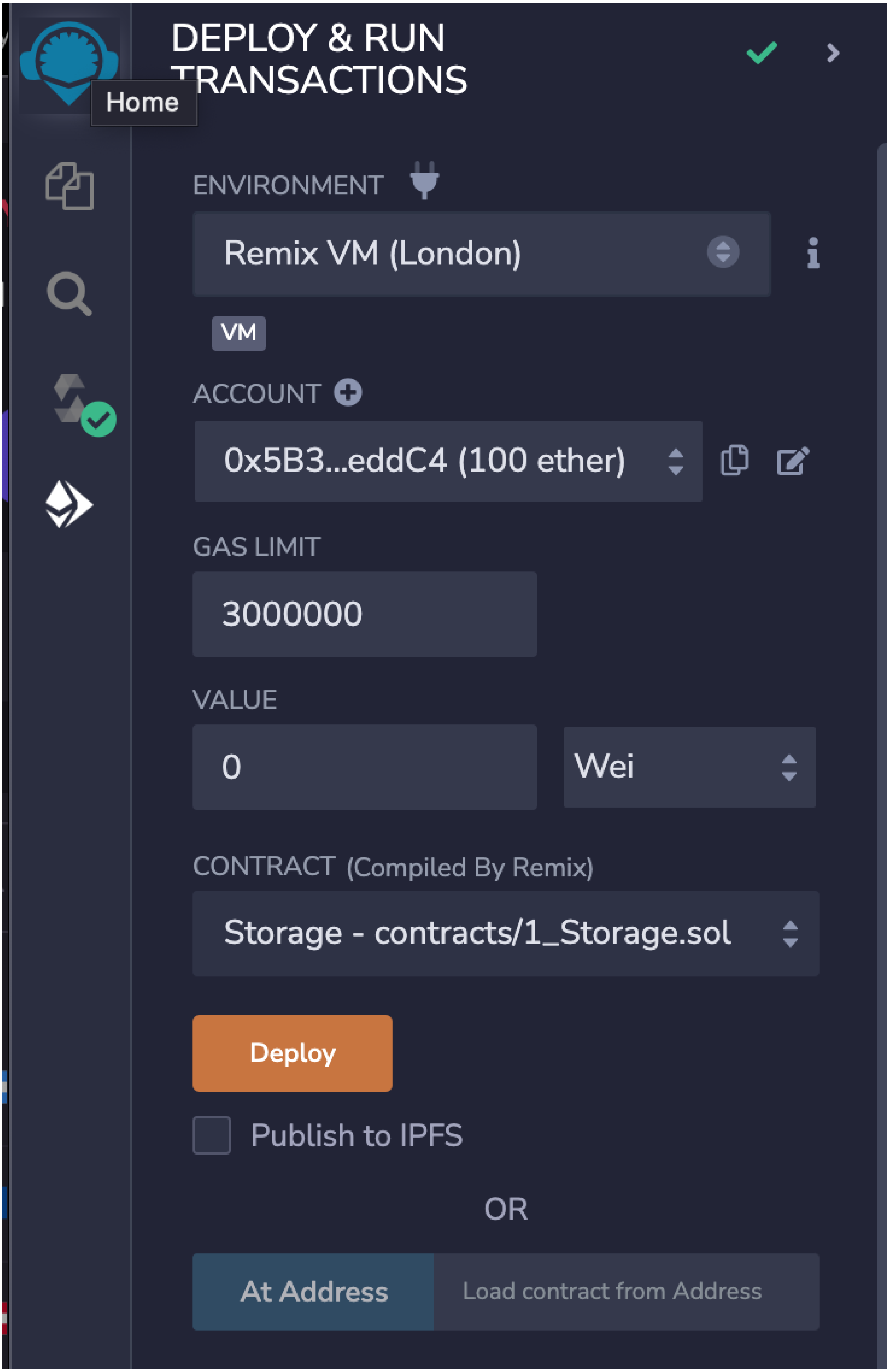
□ Hide warnings

CONTRACT

Storage (1_Storage.sol)

Deploying a contract

Chosing the deploy and run transactions icon gives us a panel that allows us to deploy our compiled transaction to a network and run transactions.



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Remix VM (London)

VM

ACCOUNT

0x5B3...eddC4 (100 ether)

GAS LIMIT

3000000

VALUE

0

Wei

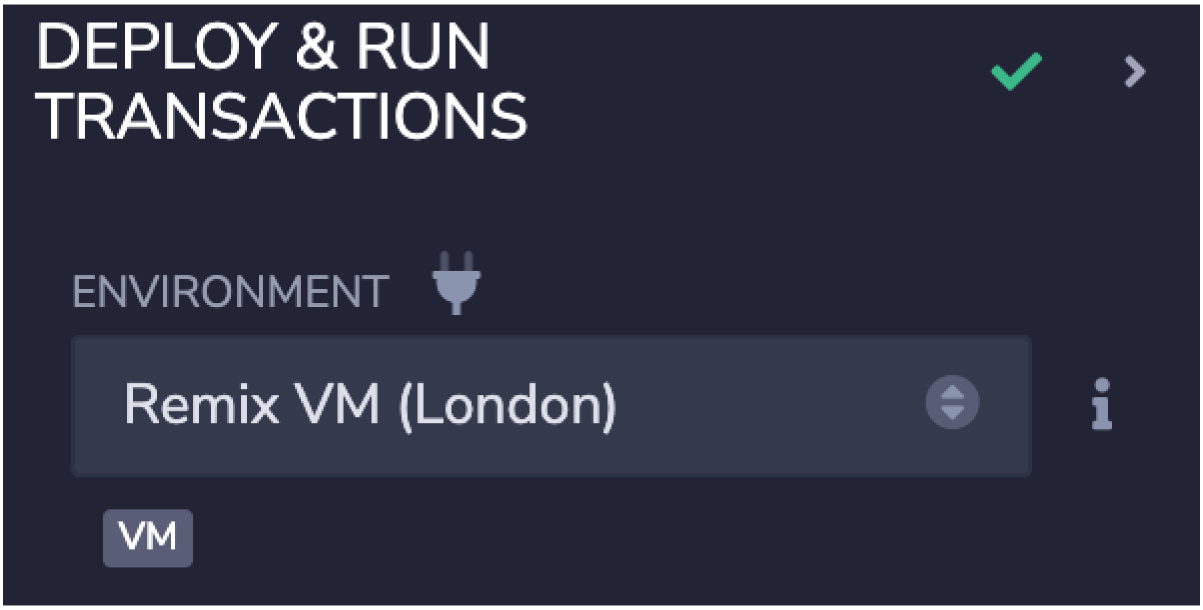
CONTRACT (Compiled By Remix)

Storage - contracts/1_Storage.sol

□ Publish to IPFS

OR

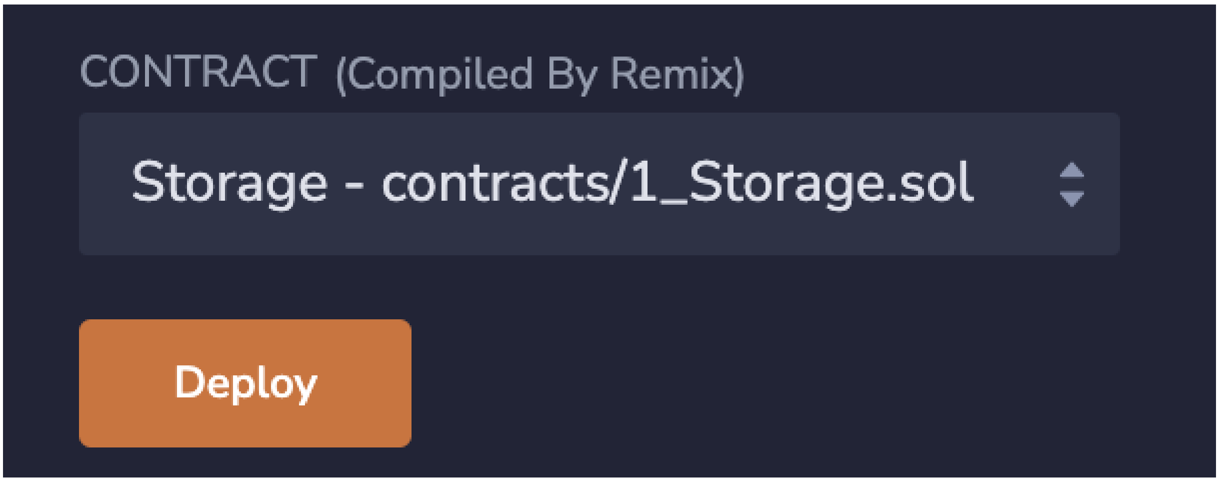
Choosing the network / environment



ENVIRONMENT
Remix VM (London)

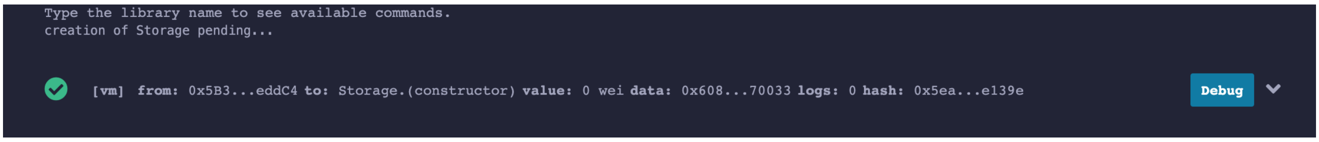
This provides a local in browser blockchain, this is the simplest environment when initially developing contracts.

We deploy the contract with the deploy button, you may need to find the correct contract in the drop down list.



CONTRACT (Compiled By Remix)
Storage - contracts/1_Storage.sol

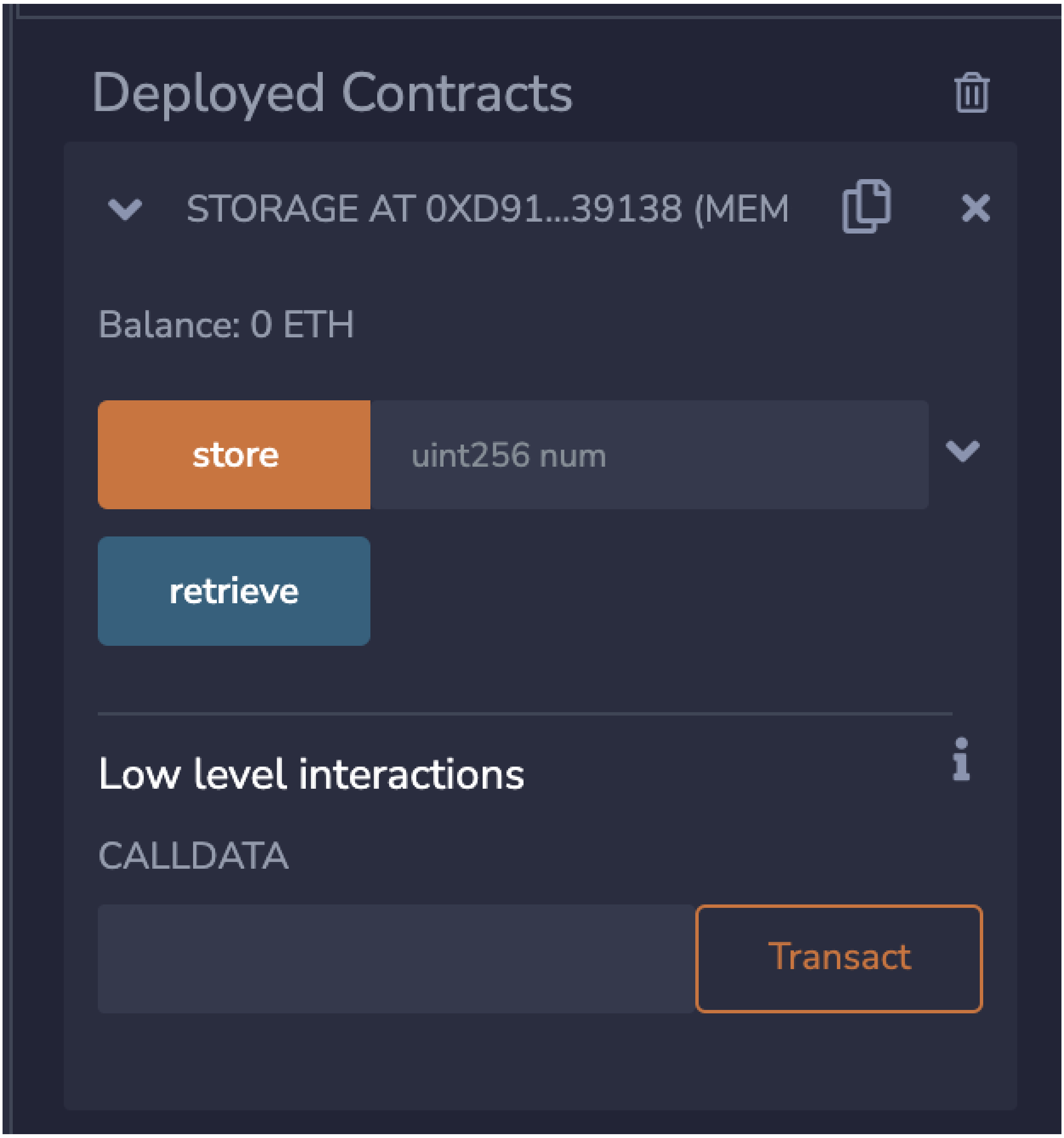
If the contract deploys correctly, you will see the results in the output window.



```
Type the library name to see available commands.
creation of Storage pending...

✓ [vm] from: 0x5B3...eddC4 to: Storage.(constructor) value: 0 wei data: 0x608...70033 logs: 0 hash: 0x5ea...e139e
```

Once it has deployed, you will see details of the contract in the deploy panel

You can interact with your contracts from here by sending transactions
Note that the contract has an address on the blockchain similar to a wallet address



Deployed Contracts

STORAGE AT 0XD91...39138 (MEM

Balance: 0 ETH

Low level interactions

CALLDATA