Lesson 4

Useful Open Source Collections

About Open Zeppelin:

Open Zeppelin are well known in the Ethereum community. They provide a set of audited smart contracts and libraries that are a standard in the industry. Inheriting these contracts will provide a significantly higher degree of security and robustness in your code. See https://docs.openzeppelin.com/contracts/4.x/

Open Zeppelin Token Contracts

Even though the concept of a token is simple, they have a variety of complexities in the implementation. Because everything in Ethereum is just a smart contract, and there are no rules about what smart contracts have to do, the community has developed a variety of standards (called EIPs or ERCs) for documenting how a contract can interoperate with other contracts.

- ERC20: the most widespread token standard for fungible assets, albeit somewhat limited by its simplicity.
- ERC721: the de-facto solution for non-fungible tokens, often used for collectibles and games.
- ERC777: a richer standard for fungible tokens, enabling new use cases and building on past learnings. Backwards compatible with ERC20.
- ERC1155: a novel standard for multi-tokens, allowing for a single contract to represent multiple fungible and non-fungible tokens, along with batched operations for increased gas efficiency.

Safe Functions:

Safe functions (SafeTransferFrom etc.) were introduced to prevent tokens being sent to contracts that didn't know how to handle them, and thus becoming stuck in the contract.

Open Zeppelin Access Control / Security Contracts

- Ownable
- AccessControl
- TimeLockController
- Pausable
- Reentrancy Guard
- PullPayment

Open Zeppelin Governance Contracts

Implements on-chain voting protocols similar to Compound’s Governor Alpha & Bravo

Open Zeppelin Cryptography Contracts

- ECDSA contract for checking signatures.
- MerkleProof for proving an item is in a Merkle tree.

Open Zeppelin Introspection Contracts

Contracts to allow runtime checks whether a target contract implements a particular interface

Open Zeppelin Maths Contracts

SafeMath - to prevent under / overflow etc. Some of this functionality is part of Solidity since version 0.8.0

Open Zeppelin Payment Contracts

- Payment splitter

- Escrow

Open Zeppelin Collections Contracts

- Enumerable Set
- Enumerable Map

Open Zeppelin Miscellaneous Contracts

- Address
- Multicall

Open Zeppelin Upgradability Contracts

- Proxy

Importing from Github in Remix

See Documentation
You can import directly from github or npm

```
import "https://github.com/OpenZeppelin/openzeppelin-contracts
/contracts/access/Ownable.sol";
```

or

```
import "@openzeppelin/contracts@4.2.0/token/ERC20/ERC20.sol";
```

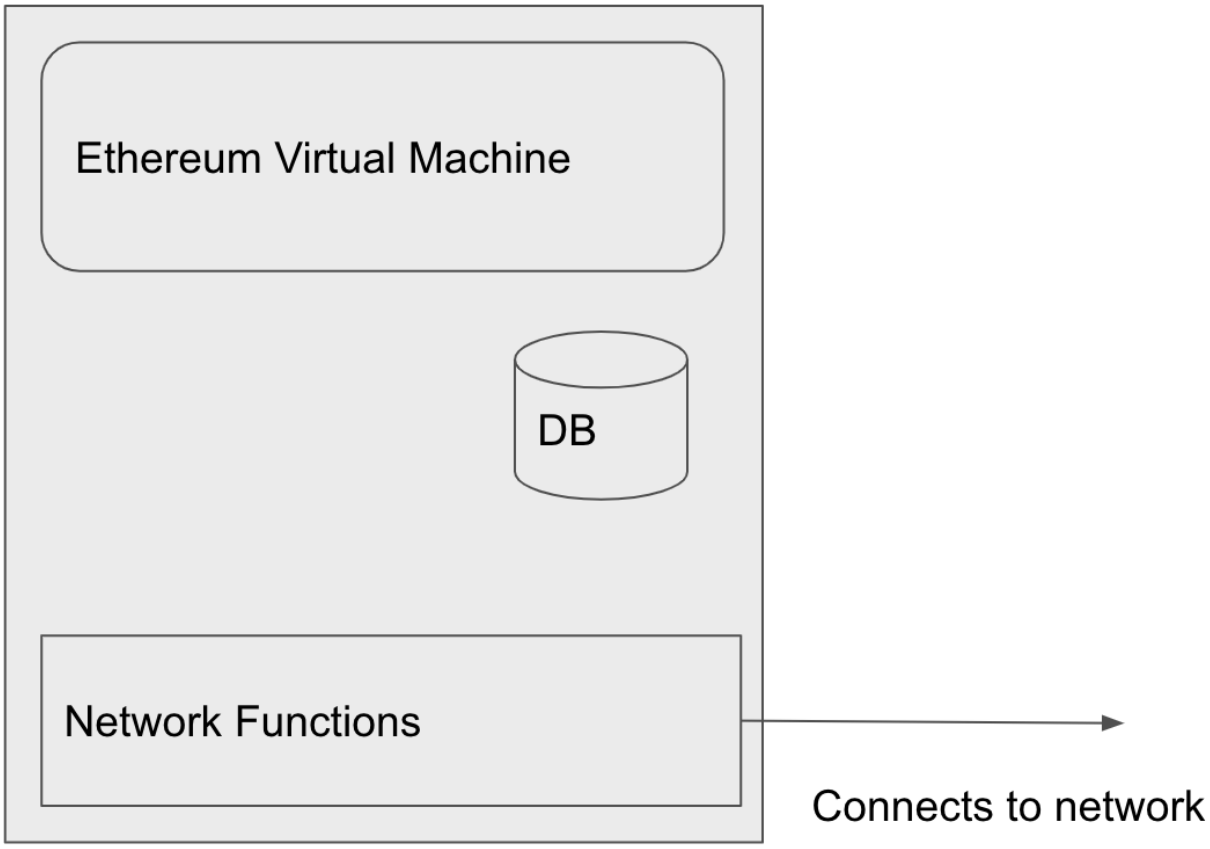
References

Solidity Documentation
Libraries

Useful Links and further resources

Official Solidity Documentation
Globally Available Variables
Open Zeppelin Libraries

Ethereum Virtual Machine



Question : Where does the processing of a contract happen ?

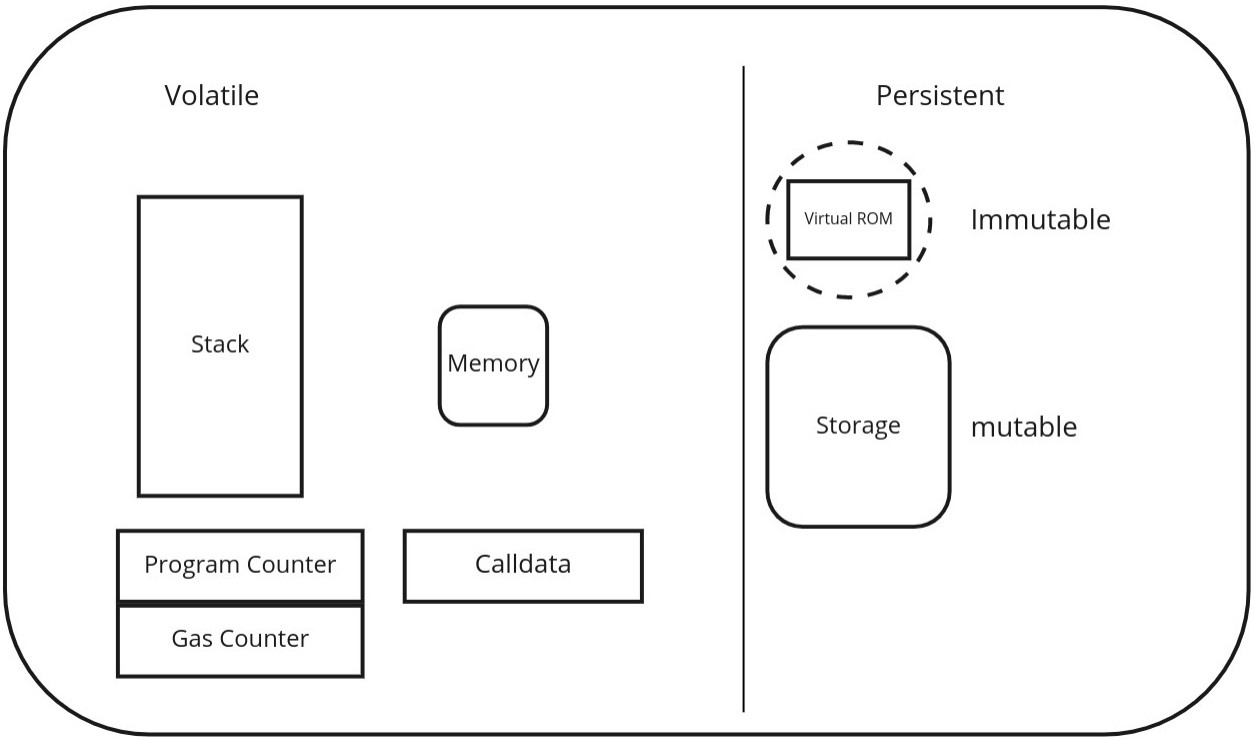
Question : Where is the state of the system stored ?

Contracts run within the Ethereum Virtual Machine.
This is a virtual machine running on the node software.
It is a self contained and restricted environment.

Contracts are much more dependent on this environment than say a python program, which is fairly independent of the environment on which it runs. Contract and dApp design is very much shaped by the constraints of the blockchain and the EVM.

The EVM is a 'stack machine', the stack has a maximum size of 1024 slots.

Stack items have a size of 256 bits; in fact, the EVM is a 256-bit word machine (this facilitates Keccak256 hash scheme and elliptic-curve computations).



Data areas

Data can be stored in

- Stack
- Calldata
- Memory
- Storage
- (Code)
- Logs

Storage and memory

See documentation

Storage

Storage data is permanent, forms part of the smart contract's state and can be accessed across all functions. Storage data location is expensive and should be

used only if necessary. The storage keyword is used to define a variable that can be found in storage location.

Memory

Memory data is stored in a temporary location and is only accessible within a function. Memory data is normally used to store data temporarily whilst executing logic within a function. When the execution is completed, the data is discarded. The memory keyword is used to define a variable that is stored in memory location.

EVM Languages

- Solidity
  - The most popular programming language for Ethereum contracts
- LLL
  - Low-level Lisp-like Language
- Vyper
  - A language with overflow-checking, numeric units but without unlimited loops
- Yul / Yul+
  - An intermediate language that can be compiled to bytecode for different backends. Support for EVM 1.0, EVM 1.5 and Ewasm is planned, and it is designed to be a usable common denominator of all three platforms.
- FE
  - Statically typed language Inspired by Rust and Python
- Huff see article
  - Low level language
- Pyramid Scheme (experimental)
  - A Scheme compiler into EVM that follows the SICP compilation approach
- Flint
  - A language with several security features: e.g. asset types with a restricted set of atomic operations
- LLLL
  - An LLL-like compiler being implemented in Isabelle/HOL
- HAseembly-evm
  - An EVM assembly implemented as a Haskell DSL
- Bamboo (experimental)
  - A language without loops

Vyper

- Pythonic programming language
- Strong typing
- Small and understandable compiler code
- Deliberately has less features than Solidity with the aim of making contracts more secure and easier to audit. Vyper does not support
  - Modifiers
  - Inheritance
  - Inline assembly
  - Function overloading
  - Operator overloading
  - Recursive calling
  - Infinite-length loops

FE

See :Repo

- Statically typed language for the Ethereum Virtual Machine (EVM).
- Inspired by Python and Rust.
- Aims to be easy to learn -- even for developers who are new to the Ethereum ecosystem.
- Fe development is still in its early stages, the language had its alpha release in January 2021.

Features

- Bounds and overflow checking
- Decidability by limitation of dynamic program behavior
- More precise gas estimation (as a consequence of decidability)
- Static typing
- Pure function support
- Restrictions on reentrancy
- Static looping

- Module imports
- Standard library
- Usage of YUL IR to target both EVM and eWASM
- WASM compiler binaries for enhanced portability and in-browser compilation of Fe contracts
- Implementation in a powerful, systems-oriented language (Rust) with strong safety guarantees to reduce risk of compiler bugs

References

DEVCON1: Understanding the Ethereum Blockchain Protocol - Vitalik Buterin
Mastering Ethereum by Andreas Antonopoulos
White paper
Beige Paper
Yellow Paper
EVM languages
Noxx Articles about the EVM