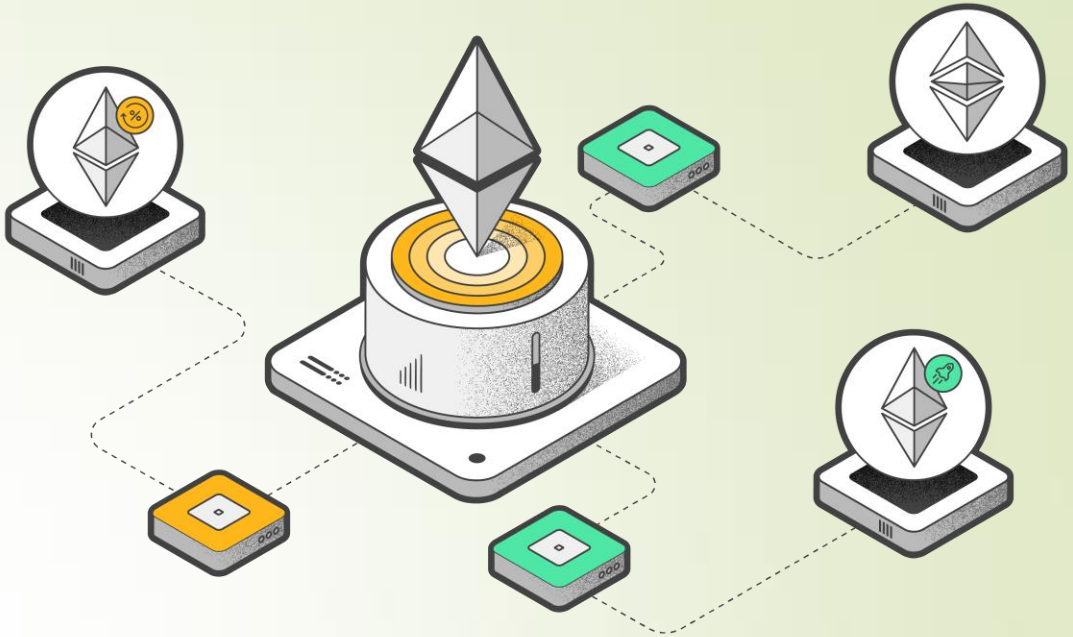


The Ethereum Virtual Machine

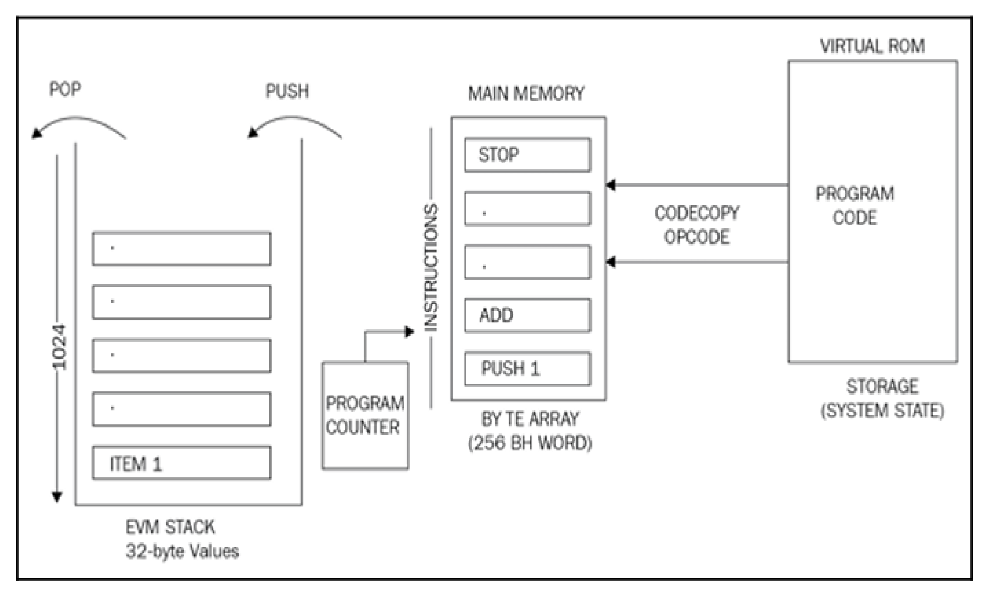
Introduction To EVM

- **Core of Ethereum**: The EVM is the execution environment for smart contracts and decentralized applications (DApps) on the Ethereum blockchain.
- **Gas System**: It regulates computational resources using gas fees, preventing network abuse and ensuring fair usage.
- **State Management**: The EVM maintains and updates the Ethereum blockchain state with each executed transaction.
- **Smart Contract Execution**: Developers write smart contracts in languages like Solidity, which are compiled into bytecode and executed by the EVM.

What is an EVM?

- The Ethereum Virtual Machine (EVM) is a stack-based execution machine designed to process smart contracts on the Ethereum blockchain.
- It operates on a **256-bit word size** and follows a **Last In, First Out (LIFO)** stack model with a **1024-element limit**.
- EVM is **Turing complete** but constrained by **gas requirements**, preventing infinite loops and potential denial-of-service attacks.
- It ensures **isolation and security** by restricting access to external resources such as networks or file systems.

EVM Architecture



- The EVM operates as a **sandboxed runtime Environment** with a **big-endian** data representation.
- It supports two storage types:
  - **Memory** (temporary, cleared after execution, similar to RAM)
  - **Storage** (persistent, stored on the blockchain, similar to hard disk storage)
- The **program code** resides in virtual **read-only memory (ROM)** and is copied into **main memory** using the CODECOPY instruction.
- The **stack** is updated dynamically, and a **program counter** (PC) ensures sequential execution of instructions.

What is Gas in Ethereum?

- **Gas** in Ethereum is the unit used to measure the computational effort required to execute transactions and smart contracts.
- **Key Points About Gas**

1. **Unit of Computation :**Every operation (e.g., adding numbers, storing data, or calling a contract) consumes a fixed amount of gas. More complex operations require more gas.
2. **Gas Price (Gwei) :**The cost per unit of gas, set by the user. Measured in **Gwei** (1 Gwei = 10^9 Wei).
3. **Gas Limit :**The maximum amount of gas a user is willing to spend on a transaction,if execution exceeds this limit, the transaction fails.
4. **Total Transaction Fee**

   Formula: Total Cost=Gas Used×Gas Price
5. **Unused Gas is Refunded**

Execution Environment

- These parameters define the execution context and influence how transactions and smart contracts run.

- **System State** → The current state of Ethereum, including account balances and storage.
- **Remaining Gas** → The amount of gas left to execute instructions (prevents infinite loops).
- **Account and Sender Addresses** → The addresses involved in execution (owner, sender, and origin).
- **Transaction Gas Price** → The fee per unit of gas, determining execution cost.
- **Input Data (Byte Array)** → Data passed to smart contracts for processing.
- **Block Header Information** → Metadata about the current block (timestamp, difficulty, etc.).
- **Permission to Modify State** → Determines whether execution can alter Ethereum's state.

Execution Process and Results

1. **Smart Contract Deployment:** Developers write smart contracts in high-level languages (like Solidity), which are compiled into EVM bytecode. Contracts are deployed to the Ethereum network through transactions.

2. **Transaction Processing:** Users create transactions to interact with deployed contracts. These transactions are propagated to Ethereum nodes.

3. **Execution:** Each node runs its own instance of the EVM to execute the transaction. The EVM processes the contract logic and updates the global state of the blockchain.

Execution Process and Results

4. **Gas Mechanism:** Each operation consumes gas, which users pay for. If the transaction runs out of gas, it reverts, but the gas is still spent.

5. **Stack Management:** The EVM uses a stack-based architecture to manage data and execute instructions, storing temporary data in memory and permanent data on-chain.

6. **Block Creation and Validation:** Processed transactions are bundled into blocks by miners or validators, validated against consensus rules, and added to the blockchain.

7. **Finality:** Once included in a block, the changes are permanent and publicly verifiable.

Machine State

- The EVM maintains an internal **machine state**, which consists of:
  - **Available gas**
  - **Program counter (PC)**
  - **Memory and stack contents**
  - **Active memory words**
- Execution halts when encountering exceptions such as:
  - **Insufficient gas**
  - **Invalid stack operations**
  - **Jumping to invalid destinations**
  - **Exceeding stack size limits (1024 elements)**

Smart Contracts and Precompiled Contracts

- **Smart Contracts in EVM**
- The EVM executes **smart contracts** written in languages like **Solidity**, **Vyper**, and others.
- Smart contracts are compiled into **bytecode**, which the EVM processes to execute predefined logic.
- **Precompiled Contracts**
- Some computationally **intensive operations** are handled by **precompiled contracts** to reduce **gas costs**.
- These precompiled contracts perform complex cryptographic and hashing functions more efficiently.

Precompiled Contracts

- **ECDSARECOVER (0x1)** – Recovers a **public key** from a given signature.
- **SHA-256 (0x2)** – Computes the **SHA-256 hash** of the input data.
- **RIPEMD-160 (0x3)** – Computes the **RIPEMD-160 hash** of the input data
- **IDENTITY (0x4)** – Simply **copies** the input data to the output.
- **Big Mod Exponentiation (0x5)** – Used in **cryptographic computations**,such as RSA encryption.
- **Elliptic Curve Point Addition (0x6)** – Performs **elliptic curve point addition**, essential in cryptographic operations.
- **Elliptic Curve Scalar Multiplication (0x7)** – Multiplies an elliptic curve point by a scalar value.
- **Elliptic Curve Pairing (0x8)** – Supports **zk-SNARK verification**, enabling advanced cryptographic proofs.

EVM Optimization and Future Developments

- **EVM optimization** is an ongoing area of research to enhance efficiency and speed.
- **Ethereum Flavored WebAssembly (eWASM)** is being explored as a potential replacement, aiming for native CPU execution.
- **JULIA (Joyfully Universal Language for Inline Assembly)** can compile to both EVM and eWASM.
- Future upgrades like **EVM 2.1** aim to improve performance and reduce gas fees.

Conclusion

- The **Ethereum Virtual Machine (EVM)** is the backbone of Ethereum, executing smart contracts and DApps.
- **Precompiled contracts** optimize computational efficiency and **reduce gas costs** for complex operations.
- The EVM ensures **security, determinism, and isolation**, making it a reliable execution environment.
- Continuous **upgrades and optimizations** enhance EVM performance, scalability, and functionality.
- Understanding the EVM, **smart contracts, and precompiled contracts** is essential for developers building decentralized applications.

Thank You!