writing Smart Contracts for Ethereum and Solana

Writing smart contracts for Ethereum and Solana requires using different programming languages and development environments due to their distinct architectures.

Ethereum Smart Contracts

Ethereum uses the **Ethereum Virtual Machine (EVM)** and is primarily developed using the **Solidity** programming language.

Language: Solidity (a high-level, object-oriented language similar to JavaScript and Python) or Vyper (a Python-inspired language).

Architecture: The smart contract (program logic) and its state (data) are bundled together in a single account.

Tools:

IDEs: Remix (browser-based) is popular for beginners.

Frameworks: Hardhat and Foundry are widely used for local development, testing, and deployment.

Libraries/Tools: **MetaMask** for wallet interaction and transaction signing, Alchemy for node infrastructure, and Ethers.js for client-side interaction are common.

Key Feature: Contracts can often be upgraded using proxy patterns if designed that way.

Solana Smart Contracts (Programs)

Solana uses its own runtime, the **Solana Virtual Machine (SVM)**, and refers to smart contracts as "programs".

Language: Primarily **Rust**, C, and C++ are used, which compile to Berkeley Packet Filter (BPF) bytecode.

Architecture: Programs are **stateless**; the program logic is stored in an executable account, but the data (state) is stored in separate, distinct accounts and passed to the program during execution. This separation allows for parallel processing and program reusability.

Tools:

Frameworks: The **Anchor framework** simplifies Rust-based Solana development with familiar tools for EVM developers.

CLI Tools: The Solana CLI is essential for configuring the environment, generating keypairs, and deploying programs to testnets like Devnet.

Compatibility: Projects like **Solang** allow developers to compile Solidity code for Solana's BPF format, and the **Neon EVM** allows full EVM dApps to run on Solana, offering pathways for cross-chain development.

Key Feature: Programs are upgradable by default via a CLI command, though this can be finalized to make them immutable.

Summary of Differences

Feature	Ethereum	Solana
Primary Language	Solidity, Vyper	Rust, C, C++
Execution Environment	Ethereum Virtual Machine (EVM)	Solana Virtual Machine (SVM/BPF runtime)
State Management	Logic and state are coupled in a single contract account	Programs are stateless; state is passed in via separate accounts
Transaction Fees	Often high and variable ("gas fees")	Very low and predictable
Speed/Scalability	Lower throughput, higher latency (compared to Solana)	High speed and parallel transaction processing

Revision #2
Created 29 October 2025 02:43:38 by AI API
Updated 7 December 2025 07:43:47 by AI Channel