Constitutional proposal

Abstract

This AIP outlines a path toward activating Stylus on Arbitrum One and Arbitrum Nova. Stylus is an upgrade that introduces a new virtual machine that runs alongside the EVM. This enables developers to write smart contracts in new programming languages, like Rust, that are more efficient than Solidity smart contracts.

Motivation

Stylus is a first-of-its-kind technology resulting from breakthrough engineering efforts in the Arbitrum ecosystem. Unlike other alternative VMs, Stylus is purely additive to the EVM. This means that Stylus contracts and EVM contracts are fully interoperable.

Some defining features of Stylus include:

- Preferred Languages: There are estimated to be roughly 20k Solidity developers, compared to 3.5 million Rust developers and 10 million C++ developers. Developers can use their preferred programming language, all interoperable with each other, on any Arbitrum chain with Stylus. Scaling to the next billion users requires onboarding the next million developers.
- EVM+: Stylus brings the best of both worlds. Developers still get all of the benefits of the EVM, including the ecosystem and liquidity, while getting efficiency improvements and access to existing libraries in Rust, C, and C++ — without changing anything about how the EVM works. EVM equivalence is no longer the ceiling; it's the floor.
- Cheaper Execution: Stylus is a more efficient execution environment than the EVM, leading directly to gas savings for complex smart contracts. Computation can be over 10x cheaper, and memory can be over 100x cheaper. Cryptography libraries can be deployed as custom application layer precompiles, permissionlessly. Use cases that are impractical in the EVM are now possible in Stylus.
- Safer By Design: Stylus doesn't just improve on cost and speed. WASM programs are also safer. Reentrancy is a common vulnerability that developers can only attempt to mitigate in Solidity. Stylus provides cheap reentrancy detection, and using the Rust SDK, reentrancy is disabled by default, unless intentionally overridden.
- Fully Interoperable: Solidity programs and WASM programs are completely composable. If working in Solidity, a developer can call a Rust program or rely on another dependency in a different language. If working in Rust, all Solidity functionalities are accessible out of the box. This means your Rust smart contracts can rely on primitives and infrastructure (such as oracles) that were built using Solidity.

Several exciting teams have been building their apps on the Stylus testnet for the past few months. These apps would simply not be possible on Arbitrum without the added efficiency or new language support. Stylus can onboard (1) the next generation of developers from alternate VM ecosystems and (2) new use cases that users cannot experience on EVM chains.

Rationale

This proposal is aligned with the Arbitrum community's mission and values:

- Ethereum-Aligned: Stylus does not change anything about how the EVM works. Rather than replacing, Stylus is completely additive and extends what is possible in the EVM.
- Sustainable: More efficiency at the execution layer can give Arbitrum chains long-term competitive advantages by allowing higher throughput, cheaper costs, and new use cases that are not possible on other EVM chains.
- Accessibility: Arbitrum can become the home for new types of developers who don't know Solidity. This includes Rust, C, C++, and any other programming language that compiles to WebAssembly.
- Inclusivity: The Stylus SDK is 100% open-source, meaning developers are free to fork, modify, and contribute to the core patterns that the SDK represents.
- Empowerment: The parameters defined in Stylus are fully configurable by the Arbitrum DAO (for Arbitrum One and Arbitrum Nova) and individual Orbit chains. These customizations can supercharge Stylus by, for example, dropping activation prices, increasing contract size limits, and lowering opcode pricing.

Key Terms

Stylus: A new programming environment, including a WebAssembly-based virtual machine, a CLI, and a canonical SDK that defines the patterns and frameworks for writing smart contracts.

WebAssembly (WASM): WebAssembly is a modern binary format used in significant web standards, browsers, and companies to speed up computation. Built to be fast, portable, and secure.

Virtual Machine (VM): The software that executes smart contracts and processes transactions.

Specifications

The Stylus VM and fraud prover have been developed as a fork of the Nitro codebase. These repositories will be merged into the main branch of Nitro if the Arbitrum DAO approves the AIP:

- stylus: GitHub - OffchainLabs/stylus: Stylus VM and Fraud Prover
- Fork of GitHub - OffchainLabs/nitro: Nitro goes vroom and fixes everything
- Includes node software, VM, and Fraud Prover
- Fork of GitHub - OffchainLabs/nitro: Nitro goes vroom and fixes everything
- Includes node software, VM, and Fraud Prover
- stylus-geth: GitHub - OffchainLabs/stylus-geth
- Fork of GitHub - OffchainLabs/go-ethereum
- Includes the execution client and databases
- Fork of GitHub - OffchainLabs/go-ethereum
- Includes the execution client and databases
- stylus-contracts: GitHub - OffchainLabs/stylus-contracts
- Fork of GitHub - OffchainLabs/nitro-contracts: The core Arbitrum Nitro contracts deployed to the parent chain to host the rollup
- Includes rollup and fraud proof smart contracts
- Fork of GitHub - OffchainLabs/nitro-contracts: The core Arbitrum Nitro contracts deployed to the parent chain to host the rollup
- Includes rollup and fraud proof smart contracts

The Stylus VM is not a replacement for the EVM. Instead, the two VMs work together to facilitate state transitions, each playing their part in executing their respective bytecode. EVM contracts written in Solidity still use Geth's EVM exactly as before, ensuring EVM equivalence. In other words, Stylus is entirely additive — a paradigm called EVM+.

Stylus contracts can be written using the Stylus SDK, which employs Solidity-equivalent ABIs and storage patterns to ensure cross-language interoperability. For example, existing Solidity DEXs can list Rust tokens without modification. New SDKs for additional programming languages can be added over time. Current SDK repositories:

- Rust SDK: GitHub - OffchainLabs/stylus-sdk-rs: Rust Smart Contracts on Arbitrum
- C/C++ SDK: GitHub - OffchainLabs/stylus-sdk-c: C/C++ Smart Contracts on Arbitrum
- Stylus CLI: GitHub - OffchainLabs/cargo-stylus: Cargo subcommand for developing Arbitrum Stylus projects in Rust

If you would like to better understand the lifecycle of a Stylus contract, head over toA Gentle Introduction: Stylus.

There are some relevant components of Stylus that are outlined below:

Host I/Os

Stylus contracts can access state or perform operations, which are not possible in pure WASM using Host I/Os. For example, a contract might query the runtime for information about its balance, emit logs, or even call out to other contracts. In these cases the Stylus VM provides Host Input / Output operations, aka Host I/Os, for interacting with the contract's environment. They are called via standard WASM ABI, which means they are available in all programming languages. For standardization purposes, types are also denoted in Rust form, which can be found in the SDK. Some host I/Os require accessing the EVM, which cannot be implemented entirely on the Rust side. For these, a special system of callbacks (an API) connects Rust to Go.

The Memory Model

Stylus introduces a novel VM that provides an opportunity to improve upon the EVM's quadratic memory model. Rather than using a per-call quadratic memory model, Stylus programs use a global, exponential memory pricing policy. The maximum memory footprint per block is currently set to 8 MB. This footprint reflects all programs ever executed during a transaction rather than being isolated to a per-call-context basis and does not include EVM contracts.

Dynamic Linking

Proving fraud over untrusted WASMs presents novel challenges to the design of the dispute protocol. Arbitrum fraud proofs consist of two phases.

- The bisection protocol, wherein the parties narrow down to a single WAVM opcode.
- The One Step Proof, wherein L1 contracts emulate that single instruction.

Stylus does not change the bisection protocol or its implementation. All changes are compatible withBOLD and the current system. However, there is one conceptual difference: instead of bisecting over the vanilla replay machine's execution of a block, user programs are dynamically linked in and out throughout execution.

Activation Pricing

There are two primary costs when activating user programs: compute and storage.

Activated Stylus programs expire after 1 year. The original compressed WASM is kept around for future reactivation. Like all Stylus parameters, the Arbitrum DAO can configure this expiration period. This policy has two benefits. First, it decreases state bloat by only keeping information necessary for the long term, pruning short-lived contracts over time. Second, the Stylus VM can charge less for activated contract code. Today, the EVM charges 200 gas per byte of contract code, which makes sense for permanent state, but leaves an opportunity to cut costs for Stylus.

The Stylus VM provides 120Mb of temporary storage per hour for activated contract code. If demand for this temporary storage exceeds this limit, the price rises, eventually to a value that exceeds 200 gas per byte.

Stylus is designed to withstand powerful adversaries that can censor Ethereum for up to a week, buy the entirety of Arbitrum's blockspace indefinitely, or similar worst-case assumptions. Stylus defaults to a configuration intended for this extreme environment, with generous error margins usually exceeding 100%.

Under this configuration, nodes will never exceed 1TB of activated contract code. In practice, the state size will be over an order of magnitude smaller, due to the generous error margins used and realistic user demand.

Along with the data fee for temporary storage, users must also pay for compute when activating a program. This compute component pays for the work done instrumenting the contract and lowering it to the node's local assembly. For most contracts, this is about 2 million gas, though in some unusual circumstances, the costs may be higher.

This means deploying a Stylus contract end-to-end will generally cost more than in the EVM. However this is only true for the first time a contract is deployed. If multiple contracts share the same code hash, only the first needs to pay for activation. Additionally, for most onchain applications, these economics work out since after a contract has been activated, compute becomes over an order of magnitude cheaper than in the EVM.

Opcode Pricing

Opcode pricing is similar to activation: each opcode takes an amount of time to run, the length of which determines the price. Unlike activation, opcode execution is measured in fractions of a nanosecond and is highly variable. This pricing table for opcodes is informed by the benchmarking numbers found via the benchmarks and runtime crates of the aforementioned 'stylus-pricer-benchmarks' branch. Another way pricing has been checked for correctness is by generating trillions of gas worth of instructions and then trying to sync nodes from scratch. In practice, on representative hardware, it was found that almost 360 million gas per second can be synced, or 50x faster than our current speed limit. This implies many opcodes are being overcharged, the prices of which can drop over time as confidence is gained in the model.

Host I/O pricing is more straightforward. Through benchmarking and static analysis, clear bounds on VM affordances can be determined. This file benchmarks all the Rust-only Host I/Os.

Steps to Implement

If the Arbitrum DAO approves the AIP, the path would consist of:

1. Discussion of the proposal on the forum
2. Completing the audit and remaining code changes

3. Merging the Stylus branch into the canonical Nitro codebase
4. A vote on Snapshot to enable Stylus on Arbitrum Sepolia
5. Sufficient time for testing on a public testnet that emulates production environments
6. An onchain vote to deploy the upgrade on Arbitrum One and Arbitrum Nova

Timeline

Stylus is rapidly progressing toward being production-ready. Once it is, ideally the upgrade can take place on Arbitrum Sepolia, and Arbitrum One and Arbitrum Nova afterwards. An onchain vote for this AIP will only occur once there is a public audit report.

NOTE:

If this vote passes on Snapshot, the tentative plan is to upgrade Arbitrum Sepolia to arbOS 30 the week of June 10th.

If Arbitrum Sepolia upgrades to arbOS 30, there will be an ecosystem-wide plan to ensure non-breaking compatibility of infrastructure and applications with Stylus, which will take several weeks or months. The ultimate goal is that the experience for EVM developers remains as unchanged as possible if arbOS reaches mainnet. Once enough confidence is reached, there will be an onchain vote to upgrade Arbitrum One and Nova to arbOS 30. The Arbitrum community will be informed of any significant updates as they arise.