Title

Add Support for EIP-4788

Constitutional / Non-Constitutional

Constitutional

Abstract

Proposal to add support for EIP-4788 within the Aribitrum Nitro stack.

Motivation

The integration of EIP-4788 into Arbitrum's Nitro stack represents a significant step forward in enhancing the functionality and interoperability of the Arbitrum network. EIP-4788, which introduces a standardized method for accessing Ethereum's state from Layer 2 (L2) solutions, is crucial for the continued growth and scalability of the Ethereum ecosystem. By supporting EIP-4788, Arbitrum can provide developers with more robust tools to build decentralized applications that benefit from and contribute to the future of Ethereum interoperability.

This proposal aims to address the growing demand for seamless interaction between L1 (L1) and L2 solutions. As the Ethereum network continues to expand, the ability to access and utilize Ethereum's state from L2 becomes increasingly important.

Furthermore, supporting EIP-4788 aligns with Arbitrum's commitment to innovation and leadership in the L2 space. By adopting this improvement proposal, Arbitrum will solidify its position as a forward-thinking platform that prioritizes the needs of its developers and users. This integration will pave the way for more advanced and scalable dApps, ultimately contributing to the broader adoption and success of the Ethereum ecosystem.

Rationale

The rationale behind integrating EIP-4788 into Arbitrum's Nitro stack is multifaceted, addressing both technical and strategic objectives that are crucial for the advancement of the Arbitrum network and the broader Ethereum ecosystem.

1. Standardization of State Access

: EIP-4788 provides a standardized method for accessing Ethereum's state from L2 solutions. This standardization is essential for ensuring consistency, reliability, and interoperability across different L2 implementations.

1. Enhanced Developer Tools

: EIP-4788 introduces the capability to access the latest 8,191 beacon roots, significantly enhancing the flexibility of implementations. This access mitigates the majority of race conditions that occur when only the current root is available.

1. Alignment with Ecosystem Growth

: Supporting EIP-4788 aligns with Arbitrum's commitment to contributing to the growth and scalability of the Ethereum ecosystem. As the demand for L2 solutions increases, the ability to access Ethereum's state becomes a critical factor in the success of these solutions.

We believe including EIP-4788 in nitro rollups would improve their composability in a number of ways:

1. Enabling verification of EigenLayer AVS state on the L1
2. Enabling ZK co-processor proofs of nitro L2 state
3. Enabling native interop standards

a. RIP-7755: Contract standard for cross-L2 calls facilitation

b. RIP-7789: Cross Rollup Contingent Transactions

c. ERC-3668: CCIP Read: Secure offchain data retrieval

d. RIP-7728: Precompile for L1SLOAD

Additionally, the implementation of this EIP does not affect the reorg risk of the rollup. Reorg risk is mainly dependent on when L1 inputs (e.g. delayed inbox) are read into the L2. For Arbitrum specifically, this risk varies depending on thedelayed sequencer configuration. Note that reorg risk arises from the execution sequencer reading in latest L1 headers via a subscription (more details in the “Steps to Implement” section).

- finalize-distance

: reorg risk increases the shorter the distance

- use-merge-finality

: default mode, low reorg risk

- require-full-finality

: no reorg risk

Eigenlayer AVS

The active EigenLayer AVS operator set for EigenDA (and other AVSs) is stored within L1 state. Access to trusted L1 state which holds this active operator set enables trust minimized verification of DA certifications on L2s/3s. This simplifies and unifies efforts to bridge L1 state to L2s/L3s, avoiding the need for bespoke solutions for alt-DA integrations

ZK Co-processor

Lagrange is a ZK co-processor that provides a queryable verifiable database over the current and historical state of Ethereum. It does so by generating ZK storage proofs against Ethereum state. Access to trusted L1 state gives Lagrange an L1 state root to verify their ZK storage proofs against.

Native Interop Standards

RIP-7755: Contract Standard for Cross-L2 Calls

This RIP introduces a comprehensive interoperability standard for intent-based cross-chain function calls, which are validated and settled through storage proofs. These storage proofs necessitate access to a trusted L1 state, which serves as the foundational reference for the connected L2 solutions that are settling transactions via their respective native bridges.

RIP-7789: Cross Rollup Contingent Transactions

This proposal introduces a method to make cross-rollup transactions contingent on the shared L1 history of communicating L2 rollups. This naturally relies on trustless access to L1 state within L2 rollups.

ERC-3668: CCIP Read: Secure offchain data retrieval

This ERC can enable trustless reads of L1 state but only if there’s access to trusted L1 state on the L2 to verify storage proofs against. The actual data is pulled by off-chain actors and is not inherently trusted.

RIP-7728: Precompile for L1SLOAD

Access to trusted L1 history and state would facilitate the implementation of this L1SLOAD precompile by giving a reference point from which to query the storage slot from the L1. This precompile requires an explicit association between an L2 and an L1 block.

Specifications

See the official EIP for detailed specifications.

Steps to Implement

In Geth, the EIP-4788 beacon root precompile contract isconfigured and deployed at genesis. Beacon root updates are applied as direct system level transactions against the precompile. Arbitrum Nitro supports the WebAssembly (wasm) instruction set, enabling it to utilize Geth itself within the L2 state machine. This capability allows Nitro to support EIP-4788 through its Geth implementation.

When the ParentBeaconRoot

field on the execution header is populated, a beacon root update is applied with the new beacon root. On L1, this field is set by the consensus client calling the execution client. Since there is no consensus client on L2, this field must be manually set within the chain derivation process.

Unlike the OP stack, Nitro does not use the engine API to talk to Geth. It instead wraps Geth in a custom sequencer implementation (gethexec sequencer) which talks to the Geth execution engine. The execution engine exposes an API to sequence transactions and accepts an L1IncomingMessageHeader

struct as an input. An additional field for ParentBeaconRoot

could be added to the struct here.

The main issue at this point is that the L1 headers processed by the gethexec sequencer are not tied to the actual L1 transactions derived from the delayed inbox. The header update fn runs in a loop reading from subscription to newHeads

of the L1. This design materially increases reorg risk if we start reading in the latest L1 beacon root in addition to block number.

Instead of reading in L1 headers via a subscription to the latest L1 head, the gethexec sequencer should read L1 headers via the delayed sequencer and delayed inbox.

The updateLatestParentChainBlock

function could be made public and exposed to the delayed sequencer, which already has access to the gethexec sequencer to sequence delayed transactions. The delayed sequencer is already reading in L1 headers and also ensures that these headers match the configured finality level.

The delayed sequencer could then write the corresponding L1 header at the correct finality level to the gethexec sequencer when it sequences delayed transactions. Once the ParentBeaconRoot

reaches the execution engine, a direct system-level transaction can be created targeting the beacon root precompile before the block is created.

Timeline

TBD

Overall Cost

N/A