Separating Execution from Proving

One of the challenges in designing a practical Rollup system is the tension between wanting the system to perform well in ordinary execution versus being able to reliably prove the results of execution. Nitro resolves this tension by using the same source code for both execution and proving, but compiling it to different targets for the two cases.

When compiling the Nitro node software forexecution , the ordinary Go compiler is used, producing native code for the target architecture, which of course will be different for different node deployments. (The node software is distributed in source code form and as a Docker image containing a compiled binary.)

Separately, forproving , the portion of the code that is theState Transition Function is compiled by the Go compiler to WebAssembly (WASM), which is a typed, portable machine code format. The wasm code then goes through a simple transformation into a format we call WAVM, which is detailed below. If there is a dispute about the correct result of computing the STF, it is resolved with reference to the WAVM code.

WAVM

The Wasm format has many features that make it a good vehicle for fraud proofs—it is portable, structured, well-specified, and has reasonably good tools and support. However, it needs a few modifications to do the job completely. Nitro uses a slightly modified version of Wasm, which we call WAVM. A simple transformation stage turns the Wasm code produced by the Go compiler into WAVM code suitable for proving.

WAVM differs from wasm in three main ways. First, WAVM removes some wasm features that the Go compiler does not generate; the transformation phase verifies that these features are not present.

Second, WAVM restricts a few features of wasm. For example, WAVM does not contain floating-point instructions, so the transformer replaces floating-point instructions with calls to the Berkeley SoftFloat library. (We use software floating-point to reduce the risk of floating-point incompatibilities between architectures. The core Nitro functions never use floating-point, but the Go runtime does use some floating-point operations.) WAVM does not contain nested control flow, so the transformer flattens control flow constructs, turning control flow instructions into jumps. Some wasm instructions take a variable amount of time to execute, which we avoid in WAVM by transforming them into constructs using fixed-cost instructions. These transformations simplify proving.

Third, WAVM adds a few opcodes to enable interaction with theBlockchain environment. For example, new instructions allow the WAVM code to read and write the chain's global state, to get the next message from the chain's inbox, or to signal a successful end to execute the State Transition Function.

ReadPreImage and the Hash Oracle Trick

The most interesting new instruction isReadPreImage which takes as input a hashH and an offsetI , and returns the word of data at offsetI in the preimage ofH (and the number of bytes written, which is zero ifI is at or after the end of the preimage). Of course, it is not feasible in general to produce a preimage from an arbitrary hash. For safety, theReadPreImage instruction can only be used in a context where the preimage is publicly known, and where the size of the preimage is known to be less than a fixed upper bound of about 110 kbytes.

(In this context, "publicly known" information is information that can be derived or recovered efficiently by any honest party, assuming that the full history of the L1 Ethereum chain is available. For convenience, a hash preimage can also be supplied by a third party such as a public server, and the correctness of the supplied value is easily verified.)

As an example, the state of a Nitro chain is maintained in Ethereum's state tree format, which is organized as a Merkle tree. Nodes of the tree are stored in a database, indexed by the Merkle hash of the node. In Nitro, the state tree is kept outside of the State Transition Function's storage, with the STF only knowing the root hash of the tree. Given the hash of a tree node, the STF can recover the tree node's contents by usingReadPreImage , relying on the fact that the full contents of the tree are publicly known and that nodes in the Ethereum state tree will always be smaller than the upper bound on preimage size. In this manner, the STF is able to arbitrarily read and write to the state tree, despite only storing its root hash.

The only other use ofReadPreImage is to fetch the contents of recentL2 Block headers, given the header hash. This is safe because the block headers are publicly known and have bounded size.

This "hash oracle trick" of storing the Merkle hash of a data structure, and relying on protocol participants to store the full structure and thereby support fetch-by-hash of the contents, goes back to the originalArbitrum design. Edit this page Last updatedonJan 27, 2025 Previous Geth at the Core Next Optimistic Rollup