Anoma Node Transaction Code Lifecycle

There was a request to showcase the general way that the Juvix-compiled Nock transaction code is used in the actual Anoma Node execution. The diagram below per @mariari and @ArtemG demonstrates the entire lifecycle Juvix-Client-Node as per v0.21.0 codebase state.

We start with Juvix-compiled transaction code. The resulting Nock code is then fed through a CLI interface. The processing of this information is done through the Client Anoma Node. For the purposes of presentation, one can bracket the way the CLI works and instead consider the fact that the Client Node calls Mempool.tx

with the specified transaction code. This asks for the executor to fire a new transaction, which prompts the executor to start up a new worker.

The worker has a run

function which processes the given transaction code at launch. We have 4 different transaction typed. Namely

- :kv
- Key-value transactions
- :ro
- Read-only transactions
- :rm
- Transparent Resource Machine transactions
- :cairo
- Shielded Resource Machine Transactions Utilizing Cairo

The Worker

bubble demonstrates which exact commands are run on each transaction type. Commands in bubbles on appropriate columns specify shared code execution.

Notes:

- All functions and names ought to correspond to actual Anoma Node names. The codebase can be consulted to see the exact types and computations each function carries out.
- Arrities of functions are not preserved.
- Dotted bubbled specify sidenotes.
- Nock execution run by the worker basically always just replaces the sample with id

specified by the mempool.

- All information here is v0.21.0 specific. With new Scry there might be a different execution pattern. In particular, Storage.put

will be removed

[

Untitled-2024-09-04-1524 (1)

1920×678 58.4 KB

](https://europe1.discourse-cdn.com/flex013/uploads/anoma1/original/1X/8b7b41bd9a54e0707157184ddde82e0f13b2327a.jpeg)