This post is introducing a phase 2 prototyping effort, but lets first discuss some background.

Phase 2 proposals:

- Casey: Phase One and Done: eth2 as a data availability engine
- Vitalik: A layer-1-minimizing phase 2 state execution proposal
- The actual spec is here: Phase 2 Proposal 1
- Slides summarising the proposal
- The actual spec is here: Phase 2 Proposal 1
- Slides summarising the proposal
- Vitalik: Proposed further simplifications/abstraction for phase 2
- Vitalik: Phase 2 Proposal 2
- Matt Garnett: Ethereum Execution Environment

Goals:

1. Create a (black boxed) execution prototyping engine
2. Create some example contracts (“execution scripts”)
3. Ask other teams to create useful scripts
4. By having actual real world use cases in scripts, we can benchmark the design and identify bottlenecks

Since Phase 2 Proposal 2 seems to offer a stateless execution model a lot of processing is shifting into the scripts. Instead of whiteboarding these proposals, the main goal here is to provide a simple environment where scripts can be developed, tested and iterated on. This would also allow iterating on the Phase 2 proposal.

Phase 2 Proposal 2 Questions

Where to do execution script validation?

New code is proposed via NewExecutionScript

in the beacon block and is placed into the beacon state.

1. Should validation occur on the beacon chain?
2. Should validation occur on the shard chain? If so, should it happen once and the code be marked in valid if it failed?

How to track execution time (aka “gas metering”)?

There’s no limit to execution time in the current design.

Phase 2 Proposal 2 EEI

In this proposal execution is simplified to basically a “stateless execution”:

post_state, deposits = execute_code(exec_code, [pre_state, block.data]))

In the simplest form, the script needs to:

- retrieve the pre_state
- return the post_state
- return a list of deposits
- potentially access other parts of the shard state, shard block and beacon state

Core EEI proposal

Basic concepts about “the EEI” can be found in theewasm design repo.

- eth2::loadPreStateRoot(memoryOffset: u32ptr)
- The current pre_state_root

(256-bit value) is loaded from the memory offset pointed at

- The current pre_state_root

(256-bit value) is loaded from the memory offset pointed at

- eth2::blockDataSize() -> u32
- Returns the size of the block.data

- Returns the size of the block.data

- eth2::blockDataCopy(memoryOffset: u32ptr, offset: u32, length: u32)
- Copies length

bytes from block.data + offset

to the memory offset

- Copies length

bytes from block.data + offset

to the memory offset

- eth2::savePostStateRoot(memoryOffset: u32ptr)
- The post_state_root

(256-bit value) is set to the content of the memory offset pointed at

- The post_state_root

(256-bit value) is set to the content of the memory offset pointed at

- eth2::pushNewDeposit(memoryOffset: u32ptr, length: u32)
- This expects a Deposit data structure to be stored at the memory offset (SSZ serialised?). It will be appended to the deposit list.

- This expects a Deposit data structure to be stored at the memory offset (SSZ serialised?). It will be appended to the deposit list.

Note

: the state_root

above is currently a 256-bit value, but it would be possible to extend this to support RSA accumulators for example.

eth2::

here refers to the namespace these functions are in.

Execution of the script starts with executing the main()

function.

Question:

Should the modified state also be returned (not just the state root)?

Question:

What other data from the shard state, shard block or beacon block should be made available to the scripts?

Extensions

The following features will be recurring among different execution scripts. It would make sense providing a single implementation of these for two reasons:

- to save space
- to speed up execution

A good option here is to make them system libraries, which can be implemented natively by clients if they choose to do so.

Candidates:

1. Bignums (see https://github.com/ewasm/design/issues/189)
2. SSZ
3. BLS utilities
4. Hashing
5. ?

Phase 2 prototyping

Considering the above a rough prototyping tool was created:https://github.com/ewasm/scout

Please check out the README, even if it quite sparse.

To get some excitement going, here is an actually working execution script source code:

extern crate ewasm_api;

use ewasm_api::*;

[cfg(not(test))]

[no_mangle]

pub extern "C" fn main() { let pre_state = eth2::load_pre_state();

```
assert!(eth2::block_data_size() == 0);

// No updates were made to the state
let post_state = pre_state;

eth2::save_post_state(post_state)
```

}

The next goal is to write a more complex script. The “deposit manager” (akain-shard ETH transfer) seems like the obvious choice.