This is NOT a new idea; it has existed in discussions for a long time. However, it seems useful to have a post explicitly discussing this.

In a sharded blockchain, and especially a stateless

sharded blockchain, there will be many pieces of code (or even non-code data such as cryptographic constants) that are shared between many use cases. This includes cryptographic algorithms, smart contract wallet implementations, token contract implementations, etc. To prevent witness sizes from getting too large as a result, we can introduce a functionality where such pieces of code can, at a high cost (eg. 1 ETH per kB), be saved on the beacon chain, and then referenced from shard chain blocks by index.

We would do this by introducing a new beacon chain transaction type, newCode: [code, validator_index, signature]

, which could be created by a fully inactive validator. This transaction type would be processed as follows:

1. Verify the signature
2. Subtract BEACON_CODE_BYTE_FEE * len(code)

from the validator's balance (if the validator has not enough balance, this fails)

1. Add (code, current_slot_number)

to an in-state list of codes: List[Tuple[Bytes, Slot]]

BEACON_CODE_BYTE_FEE

would be set to a high amount, eg. 0.001 ETH per byte.

Shard execution would have access to an opcode execute_beacon_code(index, data)

, which could execute the program with the given index

with the given data

and return the output (we save the activation slot so that in fraud proofs we know whether or not the code existed at the time a block was executed). We expect all nodes to have access to the beacon chain, so the code is information that they can access.

This is a very useful feature in light of many features of the eth2 roadmap:

- Account abstraction, where signature verification algorithms need to be part of every piece of account code
- ERC20 tokens already being abstracted
- Contracts frequently needing to be yanked across shards, passing all contract code in through a receipt
- The general move to witness and receipt-based workflow, where every byte of data read needs to be part of a block

In all cases, large pieces of code could be replaced by a simple 4-byte index of which beacon code to use.

Possible extensions

- Allow beacon codes to be removed, eg. by storing who deposited them and allowing withdrawal with an 8-month delay if the depositor agrees (the depositor address could be changed to 0 to signal a permanent guarantee of non-removal)
- Make the fee dynamic, eg. targeting a specific beacon chain state growth rate