Abstract

A new OPCODE that allows queuing some piece of code with a well defined gas used that will execute at the end of a block. If two transactions queue the same end-of-block code, it will only execute once. Gas costs are covered with ETH in the calling contract.

Motivation

This would enable things like block auctions, by allowing a bunch of people to submit transactions to swap some asset, and then they would all execute together via a block-auction at the end of the block in a way defined by the contract, rather than in a sequential order that they appear. Each transaction would queue the end-of-block code execution. If a single transaction does a swap on some pool, then the end-of-block execution would not change anything from today. If multiple people submit swaps on some pool within a single block, then the contract can execute a block-auction rather than executing the swaps in the order they are received. This allows the contract to get the best price for everyone and combat things like in-block front-running. This does not

address cross-block front-running/back-running, but that is far harder to execute than within-block front-running and sandwiching.

Specification

Mostly TBD, but gas paid by the contract allows for cost sharing strategies between transactors (e.g., split costs and refund at end) and requiring a fixed gas usage for the end-of-block execution helps with solving the bin-filling problem introduced by delayed execution.

Considerations

It could be useful to allow for specifying dependencies, so one contract could say “I want my end-of-block execution to run after this other contract’s end-of-block execution”. Care would need to be taken to avoid circular dependencies, but this could be achieved by just asserting that if a circular dependency is detected then all contracts in the circle do not get their end-of-block execution. This strongly encourages people to only

depend on immutable end-of-block executors with fixed dependency trees.