Another variation of this protection is for the afterSwap() hook to ensure the hedger contract has \beta

of the removed tokens from the start of the block. These tokens are then re-added to the pool at the start of the next block, while \beta

of the tokens added in the last block are sent to the builder controlling the hedger contract in the previous block. Tokens are then added to the vault from the pool to ensure the price at the end of the previous block is now the price of the pool (in the same way the vault is updated in the main post).

Assuming the builder leaves the pool at the profit-maximising price, both the protocol in the main post, and this simple variation have the same payoff and token requirements for the builder. The variation described here gives a stronger feeling that the pool LPs are doing something, effectively providing 1-\beta

of the liquidity on every order.

An important open question still remains: If the pool wants to retain \beta

of the LVR, can the pool deploy more than 1-\beta

of its liquidity?

Solutions auctioning off the right to execute the first transaction in a pool,such as McAMMs, beg the same question. All of these solutions are forcing the builder (or winning searcher) to repay some \beta

of the (expected in the case of McAMMs) LVR.

Latency reduction is proven to tackle LVR, but effective solutions to this for slow, secure L1s (such as ashared sequencers) are still being theory-crafted. (h/t Dan Robinson for in-depth conversations on this).