Mutating Mempools and Programmable Transactions

Simon Brown

Follow

--

Listen

Share

We're on the precipice of the first major step towards account abstraction and the many possibilities it enables. New innovations for dealing with the effects of MEV have resulted in significant architectural changes to the Ethereum ecosystem, and there are still significant changes to come. These innovations will lead to a range of options for how transactions are processed and executed. Now might be a good time to start thinking about where these innovations may lead to — and how we might get there.

One questions that interests me is this: what if transactions were programmable, dynamic, adaptive? What if transactions were more declarative than imperative, what if they simply express our intent at a high level rather than sending explicit low level instructions to a specific dapp?

This essay explores what benefits these sorts of programmable transactions might afford users, the various ways it could be achieved, and how the idea might evolve.

First of all: why is the idea of programmable transactions worth exploring?

The benefits of executing some logic in a programmable transaction instead of a smart contract (pre-chain vs. on-chain) are that it saves on gas and it can respond to conditions in the mempool that may alter the global state between the time of transaction composition and transaction execution.

This idea of responding to conditions faster than the block time of the underlying settlement layer opens up some interesting possibilities. For example, in terms of token swaps, having to wait up to the 12 seconds block time for settlement of a trade on Ethereum will nearly always result in a stale price, as prices in other domains such as centralized exchanges move much faster. Another example could be an oracle update that appears after the transaction was composed and broadcast but before it is included in a block, and which changes the information that was available at the time of execution. Allowing transactions to be context-aware means that they can respond to new information as it arrives in the mempool, updating the conditions under which the transaction is executed.

The one person who seems to be thinking deeply about this area is Vlad Zamfir, who talked about the concept of "Smart Transactions" in his talk at DevCon, where he talks about "MEV time I/O", where by earlier transactions can directly interact with later transactions through EVM storage, opening up a whole new direction of research.

So what can you can do programmable transactions?

Examples

Here are some ideas of what capabilities they might enable:

Split payments

"only process my payment if there are x other ETH transfers to the same address that add up to y within the next n blocks

"

This can reduce counter-party risk when splitting payments for goods or services, without the need to escrow funds in a smart contract and thereby wasting gas.

Atomic bilateral payments / swaps

"only process my transfer to address A if there's another transaction that transfers x ETH to address B that appears within n blocks

"

It's worth noting of course that just because both value transfers appear in the mempool at the same time, that doesn't give absolute guarantees that they will actually be executed atomically, more on that later.

Collective bargaining / Flash sales

“execute only in the presence of at least x transactions to the same contract address with the same 4-byte signature

”

This could be used by dapps to run promotions that offer a lower price for a short time if interest is sufficient, but without locking funds on-chain or wasting gas in the case where the minimum level of interest isn’t met.

JIT liquidity

“only execute this transaction if there is x volume to a specific liquidity pool and no other transactions that are providing liquidity

”

This could allow liquidity providers to improve prices for users and reduce occurrences of reverts due to excess slippage tolerance, by providing liquidity on pools where there is a spike in demand, but with the added bonus of reducing the risk of miscalculation, and reducing impermanent loss by leaving liquidity sitting in a pool for long periods.

Throttling

“don’t process this transaction until there are at most x number of pending transactions from the same address in the mempool and only if the block height % 2 = 0”

.

This could be used to have transactions processed at a certain rate over a period of time.

Oracle updates

“only process my transaction if there is a specific oracle update to address 0x1234 that has values within a certain range

”

This assumes there is an oracle every block or every few blocks and that the update will be within a certain range, and allows for an abort if it’s not. It could be used to conditionally top-up collateralized positions.

[Fee escalators

](https://docs.google.com/presentation/d/1RzmbsdgIfPCioZh2_OT7mdtXkCSjlX3hyc9GFh9-vXo)

“process my transaction with a gas price that increments at a specific rate over a specific time until it gets included in a block

”

Limit orders

“execute my swap transaction only if and when a searcher can guarantee an effective price of at least x

” (example)

Block-level flash loans

“transfer x ETH from address A to address B if there is also a transaction in the mempool to transfer at x + x’ ETH back to address A from address B and only if both transactions are executed in the same block

”

Transactions qualified with general constraints

“do not execute (or execute only) if more than x number of other transactions to the same address are in the mempool

”.

This could improve multi-sig setups, where the various signatures are only executed if the quorum is detected in the mempool within a certain time frame.