For two years, it’s been a premise of MEV research that "MEV is a big problem, and MEV is here to stay, so let's design mechanisms to redistribute it.” Recently, updated to: "...redistribute it to users

" (i.e. Orderflow Auctions).

But what hasn’t changed is the assumption that the MEV problem is more or less a stable problem. And, therefore, that the only thing we can do is to look for ways to fix it downstream

, at the execution layer.

This assumption might be mistaken. MEV can – and is – cheaper and more simply solved at source

, when constructing the transaction (by dapps and wallets). This gives users and dapps the most power and consequently best outcome. Downstream solutions are then less urgent, because far less MEV makes it that far.

This article goes over type of MEV and shows how each of them can be solved without involving changes to the execution layer – with solutions that already exist, or are well researched, today.

Solutions to all types of MEV

There are numerous sources of MEV. But most damage comes from just a handful of MEV types: Sandwiching, backrunning, cross-chain arbitrage, liquidations and front-/backrunning of oracles.

Each kind of MEV has good solutions, either already in place or planned. Together, they eliminate most MEV that existed a year ago.

Solving sandwiching

Sandwiches of swaps are particularly predatory. A bot monitors new transactions in the mempool, finds transactions with high slippage tolerance, then manipulates the relevant pools with a frontrun to stretch the user's slippage tolerance, taking profit in a subsequent backrun.

You can avoid sandwiches in two ways:

- Hide the transaction until it's mined;
- Reduce slippage tolerance enough to make sandwiches unprofitable.

Good approaches exist for both:

- Private submissions

: Submit transactions directly to the builder and validator with an agreement that they won’t sandwich transactions. This is easy to enforce, since you simply don't submit those that abuse the tx.

- Encryption

: Transaction encryption can solve all MEV arising from leaking the intent of unmined transactions, including frontrunning and sandwiches.

- Dynamic slippage tolerance

: Help users set the right slippage tolerance – high enough to increase chances for inclusion, given volatility in pool prices, and low enough to avoid sandwiching.

Users, wallets and DEXs can all enable these solutions to resolve sandwich attacks at the source. This almost entirely eliminates MEV from sandwiching.

Solving backruns

Backruns are profitable transactions enabled by another transaction, usually a swap.

To produce a backrun opportunity, a transaction must move the price on one venue – like a specific pool on Uniswap – so much that the price difference with another venue is sufficient for an arbitrage trade.

To capture the arbitrage, a backrunner sees the transaction, simulates the optimal trade, and places their trade right behind to capture the opportunity.

But if another venue has a cheaper price, why doesn't the user swap their tokens on that venue? Perhaps it was only cheaper after

the user moved the price of the first venue. Still, why doesn't the user swap part

of their tokens on the second venue?

If – ignoring gas costs – you play this out, you'll find the best trade a user can make is always the precise trade that leaves the price – net swap fees – on all venues either the same or worse (higher) after the user's trade.

And, if you optimize the trade like this, your trade won't ever increase the distance in prices between venues. So it will never produce a backrun.

Put simply, optimizing your trade also eliminates backruns.

Specialized actors that find optimal trades already exist: Solvers. Solvers search all liquidity sources and calculate the best possible trade. When users (and wallets and dapps) use Solvers to construct their trades, backruns will entirely disappear.

Solving Cross-chain arbitrage

Cross-chain arbitrage has gotten lots of recent attention. Why? Because with more chains, there is more of it. And because no accurate estimates exist for it yet.

Cross-chain arbitrage is the main argument for the persistence of backrunning: Maybe your trade is optimally routed on Ethereum, but what if the token is cheaper on Binance?

But even cross-chain arbitrage doesn't need solutions further down the stack, such as coordinating validators across chains.

Users, wallets and dapps can solve cross-chain arbitrage through market makers.

Market makers bring prices and liquidity on-chain from other venues via AMMs, and RFQ systems like Hashflow. Future AMMs will make it even easier for market makers and decentralized quoters (via Oracles) to bridge liquidity cross-chains.

This means that a Solver can, via market makers, access the liquidity of all chains and optimize your swap. This eliminates most cross-chain MEV as well, especially mid- and long-term when RFQ systems become more efficient.

Solving liquidations

Liquidations are another big source of MEV. And they are infamously costly to users. The argument for large liquidation rewards is that they're needed to guarantee the stability of lending and stablecoin protocols.

But most liquidation incentives were designed when defi volumes and numbers of capable liquidators were much lower. So now, we still have incentives that are grossly over-sized, when smaller liquidation incentives would be enough.

Better systems already exist. They have liquidation costs that are smaller by at least an order of magnitude.

- Dutch auction liquidations:

Liquidators bid for the liquidation price, leading to much better outcomes for users.

- Non-linear rewards

: Rewards don't rise linearly with liquidation amounts, but either at a decreasing rate with liquidation sums and/or with fixed

upper bounds – say USD 300 – so that rewards never exceed a reasonable threshold. This makes million-dollar rewards a thing of the past.

- Partial liquidations

: Designs that only allow liquidations up to the point where the position's health is restored also lead to smaller and softer liquidations.

Solving oracle sandwiching

Abusing Oracle updates is another common type of MEV. Oracles often suffer from high latency. So, for example, minting prices are out of date with secondary markets, offering arbitrage at the cost of the protocol (and its LPs). Oracles are also susceptible to sandwich attacks, as in the case of perpetual exchanges.

However, good solutions to Oracle arbitrage are quickly coming online:

- Low latency oracles

, or just-in-time oracles, are fixing many problems: Dapps request the oracle price after the user submits his transaction. Some oracles even request the price backwards for the exact time of a user's trade.

- Internalized arbitrage

: Some protocols, like Angle, internalize the Oracle arbitrage through keepers that return profits back to the protocol. This effectively stops the protocol's exposure to oracle MEV.

- Cheaper oracles:

With cheaper block-space on L2s or L3s, more frequent and granular oracle updates become affordable, significantly reducing the space for arbitrage.

Solving multiblock MEV

A little-discussed source of significant MEV is multi-block MEV.

Here's an example; let's call it the time-sandwich

:

1. You intend to sell enough of a token that your sell will push the price down by 5%.
2. First, you need to approve the spending of that token against the AMMs router contract.
3. Once that transaction hits the mempool – before it's even mined – a bot picks it up, checks how much of that token you hold, and estimates the price impact your sell will

have.

1. The bot then sells – that is, borrows and then sells – that token immediately behind your approval, pushing the price down by an arbitrary amount: 2, 5, 10%.
2. You might not notice the price change, as it occurs before you start your swap. And even if you notice the dip, it might simply hurry you into selling more.
3. You sell your tokens and push the price down by 5%..
4. As soon as your sell hits the mempool, the bot backruns your transaction with a buy and makes an immediate profit of 5%.

This can be much more profitable than a normal sandwich, since the bot's profit may not be bounded. What matters is that you execute your swap and the bot always makes a profit of (your_price_impact * their_trade_amount).

Notice that it doesn't matter if you submit txs privately or if you're on a chain with a centralized sequencer. This works just as well after

each tx is mined.

But even multi-block MEV now has good solutions:

- Signature approval

: Some DEXs have implemented a way to gain approvals via signatures. So you can approve and swap in a single tx.

- Account abstraction

: Account abstraction allows wallets and frontends to bundle any number of txs into a single tx. This drastically reduces the surface for multi-block MEV.

- Private submission

: Private submission, combined with bundled transactions, is great protection against multi-block MEV.

Solving MEV with smart transactions

Some designs (like smart transactions) will let users express additional preferences and conditions with their tx directly to the block-builder. For example, that the transaction is only valid if it's not being sandwiched, or that collateral of a debt-position automatically increases if the position is in danger.

This gives users, wallets and dapps more granular control over transactions, helping users to avoid MEV.

Summary

We've seen that we can eliminate all major types of MEV at the source, by simply improving dapp design and how users and wallets construct their transactions. The solutions just need improving and integrating more widely.

Soon, MEV might no longer be a big enough issue to endanger the ecosystem, such as by promoting validator centralization.

Thanks

Thanks a lot @will__price, @hasufl for discussions. And thanks for review and the title@thegostep!

And thanks to the teams behind@CoWSwap, @AngleProtocol, @1inch, @hashflow, @airswap, @eulerfinance, @GMX_IO, @PragmaOracle and many more, for innovating and silently but quickly removing MEV for good – so we all won't need to sell our order flow.