MEV searchers are a class of actors in the crypto space that are driven by the incentive to capture opportunities on-chain. Some searchers can find opportunities at the cost of users to profit (e.g sandwiching) whilst others (known as keepers) contribute to the essential operating of a protocol, like liquidations. These opportunities are generally a winner-takes-all game and as such, there is a need to be extremely efficient, leveraging sophisticated techniques to outcompete others.

Independent searchers and searcher teams build automated systems, sometimes referred to as searcher bots, to execute strategies that take advantage of these opportunities. Some opportunities are unique or esoteric in nature, leading to the development of one-off monolithic custom designs.

This piece proposes a modular framework for searcher (bot) design, an architecture that allows many strategies to be ran off the same services and be easily maintainable by teams, taking inspiration from trading systems within traditional finance. A brief look into the current state of searcher design from open-source examples and how it may develop in the future is also offered, given the landscape for searcher design is often distorted due to the proprietary nature of these systems.

A number of topics are explored including:

- The Current Landscape
- The Current State of Searcher Design
- The General Architecture of Traditional Trading Systems
- Building a Modular Searcher Execution Platform
- The Potential Futures of Searcher Design

The Current Landscape

The Current State of Searcher Design

The General Architecture of Traditional Trading Systems

Building a Modular Searcher Execution Platform

The Potential Futures of Searcher Design

The Current Landscape

Opportunities on-chain come in many forms, driven by the activity of users who create this opportunity but don't capture it themselves. This is generally labelled as MEV (or maximal extractable value) which often times falls into one of the three categories: frontrunning, backrunning or sandwich attacks.

Searchers will employ strategies to extract MEV and profit. A strategy can be to arbitrageUSD-pegged stablecoin prices using MakerDAO and Curve or spotting unhealthy borrow-lend positions on Abracadabra to liquidate.

Some strategies revolve around one-off events (e.g arbitraging Otherside land claims) whereas others, though longer-term, may be tightly coupled with the unique implementation of the underlying protocol (e.g rebase farming on OHM forks). The non-uniformity of smart contract interfaces and esoteric underlying mechanisms that power these protocols have largely driven these strategies to be implemented as single-purpose programs that can be quite difficult to extend or toilsome to adapt.

Other opportunities are more frequent with a much larger ability to profit, drawing searchers to the potential payoff of these opportunities. Such an example is the DEX / DEX atomic arbitrage strategy. The top MEV winner in 2022 made over$3m in a single arbitrage by back-running large swap transactions during the Nomad bridge exploit, with many stories alike. Given the large competition in the market, many will build more thought-out trading systems for these types of opportunities, with speed and execution performance being a large focus.

More lucrative trading strategies come from taking on additional risk in some way; a common form is that of inventory risk, having exposure to assets that can potentially depreciate (or appreciate) in value for the duration it is being held in one's portfolio. From mass minting and hodling NFTs to memetokens in sandwich attacks, these types of strategies come with a risk that the price of the asset can fall through and not profit. CEX / DEX arbitrage is too an example, in which the settlement on-chain varies to that of off-chain such that the searcher (or trader) involved faces some exposure to risk in between the time the trade is being settled on each venue. Every single trade by users can create an arbitrage opportunity to capitalise on, driving competition and optimisations: from strategic gas reductions at the contract-level, to custom node clients to win latency wars.

Different opportunities are dominated by various players, some big and some small, but all in that position generally have a notion to avoid giving away too much sensitive information that could be used by others to gain an edge or reverse engineer their strategy. As such, code is generally only open sourced when an organisation or project is wishing to push usage or that the underlying strategy is no longer profitable to continue running (a reverse survivorship bias).

Where solo searchers and smaller players tend to focus on the long-tail of opportunities, institutional trading firms and larger-sized MEV teams are more capable to capitalise on the short-tail ones with higher capacity, with the well-versed domain-knowledge optimisations, production capability and, overall, money to fund essential resources to continue and improve the execution of strategies.

The Current State of Searcher Design

Searchers are generally developed in three phases:

1. Strategy generation

. Opportunities are spotted and the strategies to extract MEV from them are ideated upon. A simple proof-of-concept may be made to test part of the strategy as a sanity check.

1. Searcher implementation.

Code is written that will listen to or poll for events, and upon meeting a certain trigger condition, constructs a transaction or bundle to exploit the opportunity.

1. Execution strategy.

The transaction or bundle is then sent off to be executed on-chain in some fashion. This transaction supply chain is spoken about more here.

Strategy generation

. Opportunities are spotted and the strategies to extract MEV from them are ideated upon. A simple proof-of-concept may be made to test part of the strategy as a sanity check.

Searcher implementation.

Code is written that will listen to or poll for events, and upon meeting a certain trigger condition, constructs a transaction or bundle to exploit the opportunity.

Execution strategy.

The transaction or bundle is then sent off to be executed on-chain in some fashion. This transaction supply chain is spoken about more here.

(Searcher implementation will be analysed here in this article, with strategy generation potentially being covered in a future article.)

As mentioned, the core of a searcher (bot) is as follows:

1. Listen or poll for events.
2. Check for a trigger condition.
3. Formulate some action to exploit opportunity.
4. Propagate action out to be executed on-chain.

Listen or poll for events.

Check for a trigger condition.

Formulate some action to exploit opportunity.

Propagate action out to be executed on-chain.

.css-ovd0pw{position:relative;height:100%;width:100%;}

.css-r0ukz2{max-width:100%;max-height:100%;}

A diagram of the on-chain trading lifecycle and the latencies involved - Exploration of MEV Latencies

.css-ovd0pw{position:relative;height:100%;width:100%;}

.css-r0ukz2{max-width:100%;max-height:100%;}

.css-r0ukz2{max-width:100%;max-height:100%;}

Searchers can be implemented in most languages, but most tend to be programmed in Rust, TypeScript, JavaScript or Python to run intense computation and network calls with as little overhead as possible - developer support in the form of libraries e.g Hardhat, ethers-rs, ApeWorX etc. reduces development time substantially. Smart contracts, written in Solidity, Vyper or Huff, may be used to run some of this logic on-chain, but there needs to be some activation call to the smart contract to execute one of its function. There may be certain reasons why someone may choose to write their logic inside of a smart contract despite the upfront deploy cost. This will be touched upon later.

It is first worth showing various examples of how a searcher may be implemented for various strategies:

- Fuse by Rari Capital: Liquidator Bot: Fuse is a money market for borrowing and lending assets. If a user chooses to lend some collateral and then borrow against this, they can be at risk of liquidation by the protocol if their collateral to debt ratio falls under a certain threshold - this usually leads to the collateral being sold cheaper than market rate, leading to an arbitrage opportunity. This specific bot is written completely in JavaScript. For a set interval of time, it will make various RPC calls to fetch pool users and compute whether any of their positions are unhealthy (trigger) and how it would be best liquidated (action) - this is all done before each liquidation is sent off over RPC as its own transaction. The Fuse FuseSafeLiquidator contract is a periphery (helper) contract that offers a method that allows a bot to liquidate a position without needing to code their own smart contract - this is utilised to avoid the user having to implement their own smart contract that would have been the recipient of the flash loan.

- Yield Protocol Liquidator: Yield Protocol is a protocol that enables interest rate swaps on the yield of money markets - it does this by creating fixed yield tokens (fyTokens) of assets, and allowing one to purchase these fyTokens with the original asset (getting it at a discount) or through minting new fyTokens by borrowing it against some collateral - the latter presents an arbitrage opportunity should the borrow position become unhealthy due to not closing the borrow position after maturity. This specific bot is written in Rust, with an execution contract in Solidity. On each new block, the searcher bot will update its running list of borrowers for the fyDAI-ETH market and the health of their positions, before then checking to see which positions are unhealthy to then trigger an auction for that position. It then tries to participate in the auction by flash-loaning the funds from Uniswap V2 into the smart contract to then purchase the collateral.

- goblinmode: Goblintown is an NFT collection that received traction due to its clever marketing - with it's success, the team behind the project released another collection that spun out off the Goblin theme known as McGoblinBurger. Holders could claim a McGoblinBurger NFT using their Goblintown NFT if it hadn't been used to claim before. An arbitrage strategy from Anish Agnihotri revolved around the usage of NFTX, an automated market maker for NFTs that allows users to also flash loan NFTs, like Goblintown, as long as a user pays back the loan with interest (6% in Goblintown's case). It is as follows: flash loan all Goblintown NFTs that hadn't claim their McGoblinBurger, claim, and pay the interest with own capital - this would be profitable if sale price of the McGoblinBurger NFT is greater than that of the NFTX loan + gas. This strategy is implemented in Solidity, but requires a caller to execute it - this can either be done manually or by an off-chain script (bot) that makes the trigger call with the necessary inputs.

- Bundle Generator: DEX mispricing is one of most common opportunities that arise on-chain. By considering each asset as a node of a graph and each Uniswap-like pool as weighted edges, finding a cycle that starts from one asset and loops back to the asset with more of that asset is an arbitrage opportunity. This repository is an example of such, doing so in Rust. Several smart contracts are interfaced with, such as for batch querying multiple pairs.

Fuse by Rari Capital: Liquidator Bot: Fuse is a money market for borrowing and lending assets. If a user chooses to lend some collateral and then borrow against this, they can be at risk of liquidation by the protocol if their collateral to debt ratio falls under a certain threshold - this usually leads to the collateral being sold cheaper than market rate, leading to an arbitrage opportunity. This specific bot is written completely in JavaScript. For a set interval of time, it will make various RPC calls to fetch pool users and compute whether any of their positions are unhealthy (trigger) and how it would be best liquidated (action) - this is all done before each liquidation is sent off over RPC as its own transaction. The Fuse FuseSafeLiquidator contract is a periphery (helper) contract that offers a method that allows a bot to liquidate a position without needing to code their own smart contract - this is utilised to avoid the user having to implement their own smart contract that would have been the recipient of the flash loan.

Yield Protocol Liquidator: Yield Protocol is a protocol that enables interest rate swaps on the yield of money markets - it does this by creating fixed yield tokens (fyTokens) of assets, and allowing one to purchase these fyTokens with the original asset (getting it at a discount) or through minting new fyTokens by borrowing it against some collateral - the latter presents an arbitrage opportunity should the borrow position become unhealthy due to not closing the borrow position after maturity. This specific bot is written in Rust, with an execution contract in Solidity. On each new block, the searcher bot will update its running list of borrowers for the fyDAI-ETH market and the health of their positions, before then checking to see which positions are unhealthy to then trigger an auction for that position. It then tries to participate in the auction by flash-loaning the funds from Uniswap V2 into the smart contract to then purchase the collateral.

goblinmode: Goblintown is an NFT collection that received traction due to its clever marketing - with it's success, the team behind the project released another collection that spun out off the Goblin theme known as McGoblinBurger. Holders could claim a McGoblinBurger NFT using their Goblintown NFT if it hadn't been used to claim before. An arbitrage strategy from Anish Agnihotri revolved around the usage of NFTX, an automated market maker for NFTs that allows users to also flash loan NFTs, like Goblintown, as long as a user pays back the loan with interest (6% in Goblintown's case). It is as follows: flash loan all Goblintown NFTs that hadn't claim their McGoblinBurger, claim, and pay the interest with own capital - this would be profitable if sale price of the McGoblinBurger NFT is greater than that of the NFTX loan + gas. This strategy is

implemented in Solidity, but requires a caller toexecute it - this can either be done manually or by an off-chain script (bot) that makes the trigger call with the necessary inputs.

Bundle Generator: DEX mispricing is one of most common opportunities that arise on-chain. By considering each asset as a node of a graph and each Uniswap-like pool as weighted edges, finding a cycle that starts from one asset and loops back to the asset with more of that asset is an arbitrage opportunity. This repository is an example of such, doing so in Rust. Several smart contracts are interfaced with, such as for batch querying multiple pairs.

Various templates exist online as well, offering a boilerplate to develop searchers on. These help speed up the process of implementation.

- degenbot - Python classes to aid rapid development of Uniswap V2 & V3 arbitrage bots on EVM-compatible blockchains
- Artemis - A framework developed by Paradigm for writing MEV bots in Rust, with a focus on simplicity, modularity and speed.
- Atomic Arbitrage - A base example of a bare implementation of an arbitrage bot written in Go.
- Qilin - A general purpose bot to provide modular components for new searchers, written in Rust.
- Hummingbot - An open source framework written in Python to run automated CeFi and DeFi trading strategies on any exchange or blockchain.

degenbot - Python classes to aid rapid development of Uniswap V2 & V3 arbitrage bots on EVM-compatible blockchains

Artemis - A framework developed by Paradigm for writing MEV bots in Rust, with a focus on simplicity, modularity and speed.

Atomic Arbitrage - A base example of a bare implementation of an arbitrage bot written in Go.

Qilin - A general purpose bot to provide modular components for new searchers, written in Rust.

Hummingbot - An open source framework written in Python to run automated CeFi and DeFi trading strategies on any exchange or blockchain.

The design of many of these open-sourced implementations fall into the category of monoliths, in which there is cohesion between different components of the code under the context that it is specialised for a specific strategy e.g liquidating assets from Aave or performing generalised sandwich attacks. They can be used to validate a strategy with speed because of their leanness and ease in utilising context-specific optimisations to gain a greater edge than general competitors. Monolithic approaches do have their potential drawbacks to consider. Below, a few are spoken about.

Smart Contracts and Their Immutability

A searcher may choose to deploy a smart contract that can be called to execute some of the logic behind a strategy on-chain. This can be for several reasons, but most boil down to the attribute of atomicity.

One may choose to make a number of RPC calls to a node to fetch some on-chain data to then construct the transaction that they would like to make, but supposing that a block has passed during the round trip back, this data is likely to have become outdated. A smart contract can be implemented that, when called, performs this query without leaving the node and performs the rest of the strategy logic. It is also the case that because a transaction can only be sent to a single contract, atomicity for multiple contract calls is difficult for an EOA (externally owned account) unless a smart contract is used to perform these operations under one execution.

Deploying a smart contract relies on an initial upfront cost for gas to deploy the contract on-chain. For some strategies, this may be a relatively insignificant price to pay, with some opportunities returning this deployment cost in a single successful opportunity e.g DEX / DEX arbitrage, but for long-tail opportunities, this may not be the case. If the deployment cost of a contract onto the Ethereum blockchain is $150 in Ether, but the average return of some long-tail MEV strategy only returns $3 per day, it'll take 50 days to breakeven, which may not be appropriate - this is also under the assumption that this strategy remains as profitable over the 50 days. This is also not helped by a contract having bugs or needing to be optimised further, of which the immutability of a smart contract must mean that a new contract would have to be deployed and switched to.

Context abstraction with a generalised execution contract can be used to minimise the number of contract deployments that have to be made across multiple strategies, elevating strategy logic from the smart contract to the off-chain part of the searcher. An on-chain execution engine can make arbitrary calls (requiring just to

, value

, data

encoded in some format) meaning that any strategy can utilise that smart contract for their on-chain execution. One can also utilise such a contract to aggregate calls into a single transaction to amortise the base fee cost that is incurred on every transaction. Some dApps also feature this capability for non-technical users to utilise e.g Furucombo and DeFi Saver.

There are various open source examples of generalised execution contracts, includingmulticall or weiroll, of which the contract will act as a proxy for the calls. Other generalised execution contracts such as that of entrypoint contracts seen in the ERC-4337 supply chain take in meta-transactions that are each done by the respective account that has signed for that meta-transaction. These examples can also be adapted to utilise atomic constructs such as flash loans to have access to large enough capital to extract as much MEV as optimally possible.

It should be noted that generalised execution contracts do impose additional overheads in gas during the execution of any calls to offer their functionality, coming at the benefit of only having to pay the one-time fixed deployment cost. After a strategy has been validated live after multiple successes, it may make sense to scale the strategy and implement a custom smart contract with specialised logic to bring the gas cost down further. It is also the case that certain paradigms e.g conditional branching, runtime queries as parameters etc. may not be supported by certain generalised smart contract implementations.

Difficulty in Utilising Bundle-Level Optimisations

Given the composability offered by DeFi primitives and protocols, it is natural for teams running multiple strategies at times to have transactions or trades that conflict. A searcher running a DEX / DEX arbitrage strategy finding profitable cycles may have one leg of their trade that conflicts with that of a liquidation bot selling off a large amount of UNI to ETH. This can be minor in impact, causing additional bips in slippage to the other transactions, to one or more transactions reverting if they overlap.

Synchronisation across multiple strategies by a service called a bundler could be leveraged to find a bundle of transactions that offers the “best” execution when considering the transactions as a whole, utilising a number of optimisation tricks and techniques to achieve what is deemed as best. This can be done in a trustless or a trusted manner with access to private keys.

(The term “bundler” is one that is already used to describe a service that groups 4337 user-operations into a single transaction, but here the term is used to consider a generalised version of this that takes user operations, transactions, intents etc. and output a bundle of transactions.)

As mentioned, a number of tricks can be considered to provide optimisations at the bundle-level before it is sent off to a block builder.

- Internalisation of Trades:

Trades that go in opposite directions can be internalised and be matched within two strategies to forego the usage of the underlying swap protocol (e.g Uniswap) and protocol fees as a result.

- Order Rerouting:

If two transactions or trades are using the same pool and going in the same direction, it may make sense to reroute one to another venue to avoid causing slippage to the other transaction.

- Transaction Ordering:

Transactions can be ordered in such a way that minimises gas cost but upholds the success of transactions. After asserting an ordering, it is also possible add additional transactions to capture any residue MEV.

Internalisation of Trades:

Trades that go in opposite directions can be internalised and be matched within two strategies to forego the usage of the underlying swap protocol (e.g Uniswap) and protocol fees as a result.

Order Rerouting:

If two transactions or trades are using the same pool and going in the same direction, it may make sense to reroute one to another venue to avoid causing slippage to the other transaction.

Transaction Ordering:

Transactions can be ordered in such a way that minimises gas cost but upholds the success of transactions. After asserting an ordering, it is also possible add additional transactions to capture any residue MEV.

The Cold Start Problem

The development of MEV strategies and their implementation is a time-consuming process. As a strategy lessens in profit, newer strategies may be considered for additional streams of revenue. Minimising the time to deploy a new strategy is

crucial to capitalise on opportunities quickly, for which it makes sense to avoid duplicating code and repeating processes. This prompts the use of reusable components known as services (or on more technical terms, microservices).

Several examples, not exhaustive, can be thought of:

- A transaction propagation service:

Transaction propagation is a key component of one's MEV strategy, interfacing with RPCs, sequencers, own nodes etc. to broadcast a transaction so that it can be executed on-chain. This can be abstracted into its own service i.e a transaction gateway that handles the management of this.

- A DEX smart order router:

Many strategies (from liquidators, keepers, and so on, to more traditional strategies such as NAV trading, mean reversion, momentum trading) require the liquidating of funds from one token to another e.g to the native token to pay for gas fees, to a stablecoin to neutralise (additional) inventory risk etc. A DEX smart order router taps into many decentralised exchanges to provide an optimal routing for these transactions. One could use a public DEX aggregator but these are limited to routing orders off top-of-the-block state. An inline smart order router allows one to simulate pending mempool transactions and swaps to find context-adjusted paths.

- EVM simulation suite

: For one, it is not at all appropriate to test a strategy on a testnet like you would with a dApp. Testing live with a mainnet fork or EVM simulation helps to reduce worries for how the strategy will execute under certain conditions. A simulation suite can help standardise this process, catching some of the edge case of issues seen previously e.g weird ERC20 tokens.

A transaction propagation service:

Transaction propagation is a key component of one's MEV strategy, interfacing with RPCs, sequencers, own nodes etc. to broadcast a transaction so that it can be executed on-chain. This can be abstracted into its own service i.e a transaction gateway that handles the management of this.

A DEX smart order router:

Many strategies (from liquidators, keepers, and so on, to more traditional strategies such as NAV trading, mean reversion, momentum trading) require the liquidating of funds from one token to another e.g to the native token to pay for gas fees, to a stablecoin to neutralise (additional) inventory risk etc. A DEX smart order router taps into many decentralised exchanges to provide an optimal routing for these transactions. One could use a public DEX aggregator but these are limited to routing orders off top-of-the-block state. An inline smart order router allows one to simulate pending mempool transactions and swaps to find context-adjusted paths.

EVM simulation suite

: For one, it is not at all appropriate to test a strategy on a testnet like you would with a dApp. Testing live with a mainnet fork or EVM simulation helps to reduce worries for how the strategy will execute under certain conditions. A simulation suite can help standardise this process, catching some of the edge case of issues seen previously e.g weird ERC20 tokens.

Modularising logic from searcher bots can help to make development for future strategies easier, leveraging optimisations that have already been explored and implemented prior.

New infrastructure can be modularised on-the-go when deemed fit as new intricate problems arise, but such a methodology requires experimentation for us to build out a full suite of services for trading on-chain. We can look towards trading architectures used in traditional finance as inspiration for a proposed framework.

The General Architecture of Traditional Trading Systems

The culmination of modern trading systems comes from the work of many years of research, development and engineering in electronic trading. They largely follow the requirements of strategies and the traditional trade lifecycle.

One such depiction comes from the book Professional Automated Trading

by Eugene A. Durenard.

An Architecture for a Swarm-Based Trading System - Professional Automated Trading by Eugene A. Durenard.

The traditional trade lifecycle can be simplified to such:

- Exchange sends events to connected trading systems
- Trading system processes events to generate the set of actions it wants to perform on the exchange to maximise returns

- Actions are sent back to the exchange to be executed

Exchange sends events to connected trading systems

Trading system processes events to generate the set of actions it wants to perform on the exchange to maximise returns

Actions are sent back to the exchange to be executed

Following the structure described in Durenard's book, when an event comes in from the electronic communication network (ECN) layer:

1. It is semantically processed through an interfacing translation layer as to tell the trading system what has happened.
2. The resultant event data is used to update a local state representation of the market (e.g a price aggregator like an orderbook).
3. It is then shared with the brains of the trading system: the trading agents. Trading agents run in parallel to find trades or interactions that would ultimately give them a better return. These can be atomic in nature (arbitrage) or over time (trading).
4. Trade decisions are checked through a control layer before reaching theorder management system (OMS), involving automated risk management and human control.
5. The OMS layer takes orders and finds ways to best execute them. In traditional finance, such execution algorithms include TWAP and VWAP. For similar orders that are done on the same exchanges, orders are aggregated through the order aggregator to reduce brokerage commission or other fees
6. The translation layer handles the semantics of the trade into the necessary syntax over the required communication channel to relay the trade to the exchange.
7. The ECN layer returns a consequential acknowledgement and soon delivers a fill event back to us to let us know of a completed trade. For aggregated orders, these fills are then disaggregated.
8. Orders are also shared with middle and back-office layers for post-trade analysis that can help improve execution performance.

It is semantically processed through an interfacing translation layer as to tell the trading system what has happened.

The resultant event data is used to update a local state representation of the market (e.g a price aggregator like an orderbook).

It is then shared with the brains of the trading system: the trading agents. Trading agents run in parallel to find trades or interactions that would ultimately give them a better return. These can be atomic in nature (arbitrage) or over time (trading).

Trade decisions are checked through a control layer before reaching theorder management system (OMS), involving automated risk management and human control.

The OMS layer takes orders and finds ways to best execute them. In traditional finance, such execution algorithms include TWAP and VWAP. For similar orders that are done on the same exchanges, orders are aggregated through the order aggregator to reduce brokerage commission or other fees

The translation layer handles the semantics of the trade into the necessary syntax over the required communication channel to relay the trade to the exchange.

The ECN layer returns a consequential acknowledgement and soon delivers a fill event back to us to let us know of a completed trade. For aggregated orders, these fills are then disaggregated.

Orders are also shared with middle and back-office layers for post-trade analysis that can help improve execution performance.

Building a Modular Searcher Execution Platform

As institutions get more involved within the space of DeFi, more sophisticated architectures compared to that of one-off searcher bots will begin to appear for those that wish to have better control over their execution on-chain.

A Modular Searcher Execution Platform

If we consider translations of the traditional design over to DeFi, we can develop a framework that mimics such, a prediction into what searchers in the future can look like.

1. Collectors

are used to collect events from a series of different sources (such as the public mempool, Order Flow Auctions, Alt Mempool etc.), extracting the semantic of events from the syntactic messages that are sent across the network or from the event source. A collector may exist for capturing emitted Uniswap swap events, another for listening to new Aave lend-borrow positions, all to allow for one strategy that liquidates Aave positions into Uniswap (for each strategy or pair there may be a unique Collector). The name “Collectors” comes from Artemis by Frankieislost at Paradigm.

1. Data aggregators

take data from collectors and saves it into a representation that is semantically useful for the trading system e.g an orderbook that aggregates all central limit order books (CLOBs) on-chain. This can be for specific strategies or a class of strategies where modularity comes into play e.g the same graph of pool reserves from top-of-the-block state can be shown to trading agents running strategies on DEXs. Capturing the state of a protocol and being able to update it through events means that the cost of having to construct the full state upon load is amortised. For example, one can keep track of Uniswap V2 pools and their reserves by first retrieving an indexed version of reserves (e.g through an indexer like Subsquid, Transpose, The Graph to name a few) and then applying transformations based on the simulated swaps that occur.

1. Searchers / Executors (Trading Agents)

2. The logic behind a searcher’s strategy is what would be defined as the trading agent. They take the state of protocols provided to them and make a decision on whether any interactions in the form of a protocol interaction should be made. Under the context of a lending protocol, using the state of positions under that protocol, we can run through some logic checking for positions that are unhealthy. For any that are, we can package the operation to liquidate the position with another that sells off the collateral for our debt token back. We can also consider this under the context of inter-block liquidity provisioning (LP) in which our trading agent will decide if we should pull our liquidity from a certain protocol. By separating the logic of strategies away from the execution logic, they can utilise the optimisations from the rest of the system. We can also look to have agents work together e.g parallelised negative cycle detection on subgraphs of a saved DEX reserve graph, however, some synchronisation will need to occur at some point, often done at the bundler-level.

3. Automated Risk Management

is something that helps to sanity check the decisions made by searchers. Tooling such asEVM simulations can be used as a simple smoke test for whether the gas costs will be covered, are too high, is there any potential profit to be extracted, Salmonella etc. We also consider the risk of the position as well e.g how will that trade change our portfolio allocation? What exposure will we have? Should we trade the profits of an atomic arbitrage strategy all back into stablecoins? How much is an acceptable amount of profit to actually consider executing this transaction?

1. Bundlers

are very alike to that of a trading system’s order management system. They take in user transactions or operations that are translated into relevant order-management actions inside of a “transaction vessel” that is constructed in such a way that achieves “best” execution when considering all transactions in bulk. A smart order management system can utilise a number of techniques to reduce the cost of execution and manage conflict resolution of transactions. Operation aggregation is one of these techniques, relying on either account abstraction or full trust (access to all keys) to function, amortising the base fee cost introduced by EIP1559. Another is internalisation of trades i.e finding Coincidence-of-Wants that forgo an intermediary (and therefore associated fees). Bundlers can also take note of other chains and DEX aggregators to offer better order routing through considering inter-chain venues.

1. Gateways

can be thought of as the translation layer of a trading system, providing the software client to process incoming and outgoing messages to and from the network. It abstracts away the network layer from the rest of the trading system, focusing on transmission of data.

1. Block builders

are actors introduced by proposer-builder separation to offload the work of block construction away from validators. Given that blockchains packages transactions into blocks and not individually as a normal off-chain exchange would, block builders help to aggregate transactions into this form of blocks. A searcher can choose to broadcast their bundle to multiple existing builders, but one can also make their own block builder to process blocks for their own transactions and have more control over execution. The work involved with this includes having to receive orderflow from external sources (user transactions, OFA, Private RPC) to have a greater chance of having your block picked as the most valuable block in terms of block rewards to the validator. Though this role may be specific to current times (under mev-boost), we may see this role transform into that of gas-maximising bundlers in systems such as SUAVE.

1. Middle office

contains sideline services that are off the critical path, but are still essential to the improvement of strategies. These can include:

- Performance Analysis (Slippage, Executed Price, Reverts, Bundle Execution)

- Competitor Analysis (Did a revert happen because of another searcher? Priority Gas Auction Relative Performance, How many other searchers are doing the same strategy etc.)
- Latency Benchmarking (How is latency across the transaction supply chain; am I getting data fast enough or should I switch / integrate more data providers? etc.)
- Performance Analysis (Slippage, Executed Price, Reverts, Bundle Execution)
- Competitor Analysis (Did a revert happen because of another searcher? Priority Gas Auction Relative Performance, How many other searchers are doing the same strategy etc.)
- Latency Benchmarking (How is latency across the transaction supply chain; am I getting data fast enough or should I switch / integrate more data providers? etc.)

Collectors

are used to collect events from a series of different sources (such as the public mempool, Order Flow Auctions, Alt Mempool etc.), extracting the semantic of events from the syntactic messages that are sent across the network or from the event source. A collector may exist for capturing emitted Uniswap swap events, another for listening to new Aave lend-borrow positions, all to allow for one strategy that liquidates Aave positions into Uniswap (for each strategy or pair there may be a unique Collector). The name “Collectors” comes from Artemis by Frankieislost at Paradigm.

Data aggregators

take data from collectors and saves it into a representation that is semantically useful for the trading system e.g an orderbook that aggregates all central limit order books (CLOBs) on-chain. This can be for specific strategies or a class of strategies where modularity comes into play e.g the same graph of pool reserves from top-of-the-block state can be shown to trading agents running strategies on DEXs. Capturing the state of a protocol and being able to update it through events means that the cost of having to construct the full state upon load is amortised. For example, one can keep track of Uniswap V2 pools and their reserves by first retrieving an indexed version of reserves (e.g through an indexer like Subsquid, Transpose, The Graph to name a few) and then applying transformations based on the simulated swaps that occur.

Searchers / Executors (Trading Agents)

- The logic behind a searcher’s strategy is what would be defined as the trading agent. They take the state of protocols provided to them and make a decision on whether any interactions in the form of a protocol interaction should be made. Under the context of a lending protocol, using the state of positions under that protocol, we can run through some logic checking for positions that are unhealthy. For any that are, we can package the operation to liquidate the position with another that sells off the collateral for our debt token back. We can also consider this under the context of inter-block liquidity provisioning (LP) in which our trading agent will decide if we should pull our liquidity from a certain protocol. By separating the logic of strategies away from the execution logic, they can utilise the optimisations from the rest of the system. We can also look to have agents work together e.g parallelised negative cycle detection on subgraphs of a saved DEX reserve graph, however, some synchronisation will need to occur at some point, often done at the bundler-level.

Automated Risk Management

is something that helps to sanity check the decisions made by searchers. Tooling such asEVM simulations can be used as a simple smoke test for whether the gas costs will be covered, are too high, is there any potential profit to be extracted, Salmonella etc. We also consider the risk of the position as well e.g how will that trade change our portfolio allocation? What exposure will we have? Should we trade the profits of an atomic arbitrage strategy all back into stablecoins? How much is an acceptable amount of profit to actually consider executing this transaction?

Bundlers

are very alike to that of a trading system’s order management system. They take in user transactions or operations that are translated into relevant order-management actions inside of a “transaction vessel” that is constructed in such a way that achieves “best” execution when considering all transactions in bulk. A smart order management system can utilise a number of techniques to reduce the cost of execution and manage conflict resolution of transactions. Operation aggregation is one of these techniques, relying on either account abstraction or full trust (access to all keys) to function, amortising the base fee cost introduced by EIP1559. Another is internalisation of trades i.e finding Coincidence-of-Wants that forgo an intermediary (and therefore associated fees). Bundlers can also take note of other chains and DEX aggregators to offer better order routing through considering inter-chain venues.

Gateways

can be thought of as the translation layer of a trading system, providing the software client to process incoming and outgoing messages to and from the network. It abstracts away the network layer from the rest of the trading system, focusing on transmission of data.

Block builders

are actors introduced by proposer-builder separation to offload the work of block construction away from validators. Given that blockchains packages transactions into blocks and not individually as a normal off-chain exchange would, block builders help to aggregate transactions into this form of blocks. A searcher can choose to broadcast their bundle to multiple existing builders, but one can also make their own block builder to process blocks for their own transactions and have more control over execution. The work involved with this includes having to receive orderflow from external sources (user transactions, OFA, Private RPC) to have a greater chance of having your block picked as the most valuable block in terms of block rewards to the validator. Though this role may be specific to current times (under mev-boost), we may see this role transform into that of gas-maximising bundlers in systems such as SUAVE.

Middle office

contains sideline services that are off the critical path, but are still essential to the improvement of strategies. These can include:

- Performance Analysis (Slippage, Executed Price, Reverts, Bundle Execution)
- Competitor Analysis (Did a revert happen because of another searcher? Priority Gas Auction Relative Performance, How many other searchers are doing the same strategy etc.)
- Latency Benchmarking (How is latency across the transaction supply chain; am I getting data fast enough or should I switch / integrate more data providers? etc.)

Performance Analysis (Slippage, Executed Price, Reverts, Bundle Execution)

Competitor Analysis (Did a revert happen because of another searcher? Priority Gas Auction Relative Performance, How many other searchers are doing the same strategy etc.)

Latency Benchmarking (How is latency across the transaction supply chain; am I getting data fast enough or should I switch / integrate more data providers? etc.)

The Potential Futures of Searcher Design

A modular microservice architecture is just one design that the design space for searchers can potentially gravitate to, but there are many ways that searcher design can evolve in the future, all not mutually exclusive, and the potential new use cases for drawing attention to it.

Here are a few speculations:

- Searcher design will become more alike to supply chain infrastructure design.

After seeing the vertical integration of searchers further down the transaction lifecycle by creating builders, it is likely that we'll see searchers also look up towards services that are more user-facing e.g 4337 bundlers, paymasters etc. Such services are closely related to the components that searchers would have built (block builders, DEX order routers etc.) where it would make sense to consider turning some of their own services into products that accept external orderflow. We'll also see current supply chain infrastructure take features observed in searcher design to find ways to better their performance. One good example of this is credibly neutral searching by block builders such as BuildAI, using AI to capture any residue MEV and return some additional profit back to the bundle sender, and ProtoRev by the Skip Protocol team.

- Creation of B2B MEV-as-a-service desks or DeFi execution platforms

. In traditional finance, execution is key, but waiting several months to a year to create the needed trading infrastructure internally to deliver on this can be an obstacle. As such, some institutions e.g hedge funds choose to go with third-party systems so that they can deploy their strategies now e.g Systemathics, Bloomberg Terminal etc.. Rather than financial institutions choosing to hire their own engineers to build out their own DeFi execution platform, it may make sense for them to use a third-party agency execution platform such that they can roll out their MEV or on-chain HFT trading strategies quickly. Similar to above, it is likely that we'll see searchers consider this as a potential product opportunity, given that the infrastructure is very alike. Several execution platforms already exist for trading on centralised exchanges e.g Talos, Caspian, Elwood but only a few have done such for trading on-chain e.gEulith, Dexible, Vektor. These services can then make use of their position to vertically scale up the supply chain, with such volumes being visibly significant on-chain. We can see this also being applied to protocols that require (MEV) automation services as MEV-as-a-service desks (e.g Gelato, Keeper Network, Propeller Heads etc.).

- The space for searcher design will be experimented with more out in the open, as initiatives such as SUAVE, Anoma, CoWSwap utilise searchers for good.

User-benefitting solutions such as orderflow auctions and RFQ systems have driven a positive narrative for searchers to be framed as automatic agents that can help the user benefit more through offering better price quotes or reclaiming MEV. Though the name may differ i.e executors, solvers etc., the design is generally similar. Given this, open-sourcing these bots is a positive sum for users and it is likely that we will see initiatives that drive the development of these searchers out in the open to drive collaborative innovation. Artemis from Paradigm is one of the great initiatives that is propelling this forward.

- There will be also be more searcher/infrastructure-related tooling built out to measure the effectiveness of initiatives and optimisations as we get closer and closer to the efficiency frontier in MEV extraction.

Tooling such as performance analysis dashboards (slippage, executed price), competitor analysis etc. that analyse certain components of a searcher's setup can be positioned as products that are utilised by other searchers. A number of solutions such as Eigenphi, Zeromev and Flashbots Explorer have helped in illuminating the dark forest, and it is likely that we'll continue to see more dashboards that analyse or measure each component of the MEV supply chain, especially as participants, particularly searchers, become more involved.

Searcher design will become more alike to supply chain infrastructure design.

After seeing the vertical integration of searchers further down the transaction lifecycle by creating builders, it is likely that we'll see searchers also look up towards services that are more user-facing e.g 4337 bundlers, paymasters etc. Such services are closely related to the components that searchers would have built (block builders, DEX order routers etc.) where it would make sense to consider turning some of their own services into products that accept external orderflow. We'll also see current supply chain infrastructure take features observed in searcher design to find ways to better their performance. One good example of this is credibly neutral searching by block builders such as BuildAI, using AI to capture any residue MEV and return some additional profit back to the bundle sender, and ProtoRev by the Skip Protocol team.

Creation of B2B MEV-as-a-service desks or DeFi execution platforms

. In traditional finance, execution is key, but waiting several months to a year to create the needed trading infrastructure internally to deliver on this can be an obstacle. As such, some institutions e.g hedge funds choose to go with third-party systems so that they can deploy their strategies now e.g Systemathics, Bloomberg Terminal etc.. Rather than financial institutions choosing to hire their own engineers to build out their own DeFi execution platform, it may make sense for them to use a third-party agency execution platform such that they can roll out their MEV or on-chain HFT trading strategies quickly. Similar to above, it is likely that we'll see searchers consider this as a potential product opportunity, given that the infrastructure is very alike. Several execution platforms already exist for trading on centralised exchanges e.g Talos, Caspian, Elwood but only a few have done such for trading on-chain e.gEulith, Dexible, Vektor. These services can then make use of their position to vertically scale up the supply chain, with such volumes being visibly significant on-chain. We can see this also being applied to protocols that require (MEV) automation services as MEV-as-a-service desks (e.g Gelato, Keeper Network, Propeller Heads etc.).

The space for searcher design will be experimented with more out in the open, as initiatives such as SUAVE, Anoma, CoWSwap utilise searchers for good.

User-benefitting solutions such as orderflow auctions and RFQ systems have driven a positive narrative for searchers to be framed as automatic agents that can help the user benefit more through offering better price quotes or reclaiming MEV. Though the name may differ i.e executors, solvers etc., the design is generally similar. Given this, open-sourcing these bots is a positive sum for users and it is likely that we will see initiatives that drive the development of these searchers out in the open to drive collaborative innovation. Artemis from Paradigm is one of the great initiatives that is propelling this forward.

There will be also be more searcher/infrastructure-related tooling built out to measure the effectiveness of initiatives and optimisations as we get closer and closer to the efficiency frontier in MEV extraction.

Tooling such as performance analysis dashboards (slippage, executed price), competitor analysis etc. that analyse certain components of a searcher's setup can be positioned as products that are utilised by other searchers. A number of solutions such as Eigenphi, Zeromev and Flashbots Explorer have helped in illuminating the dark forest, and it is likely that we'll continue to see more dashboards that analyse or measure each component of the MEV supply chain, especially as participants, particularly searchers, become more involved.

Conclusion

This article layouts a modular framework that searchers can implement for their MEV trading architecture, acting as a foundation for adaptations and products that may take inspiration from some of its components. By considering the current monolithic paradigm of searcher design, it is possible to consider a modular approach as done so here.

Also discussed is a number of speculations into searcher design can evolve in the future, following from considering a modular approach. Given the initiatives that have managed to frame searcher bots as agents that can deliver additional benefits to the user, it is likely that searcher design will be discussed more and more openly in a collaborative manner to get as close to the theoretical limits of MEV.

Acknowledgments

We would like to thank Quintus Kilbourn, Ankit Chiplunkar,

William Robinson, Dan Marzec and Ezon for their comments and discussions on the initial drafts of this article.

You can find us on Twitter at 0xTaker and its_mobeen - we're both open to any conversations so please do reach out.