SHIO ON SUI

Abstract. The goal of Shio is to **enable efficient MEV extraction and fair value distribution on Sui**. Concretely, we aim to:

(1) Mitigate the negative externalities associated with MEV and transformed them into incentives for Sui DeFi trading activities.
(2) Establish fair value distribution among users, protocols, searchers, and validators through governance voting. This is estimated to bring approximately **$2M annually to boost the Sui community growth** based on the current trading volume.
(3) Enable efficient value extraction and democratize MEVs.
(4) Minimize the impact to Sui stability and transaction latency/throughput.
(5) Eventually become fully trustless and decentralized.

Through improved user experiences, increased value for protocols and validators, **Shio helps to drive the long-term success of the network**. The value includes TVL of Defi protocols, staking volume of validators and network wide trading volume growth – where MEV trading often accounts for **40%** of the overall trading volume.

1. The Problem: Bad user experience and unfair value distribution

1.1. **What is MEV.** Maximal (also known as miner) extractable value, or MEV [1], typically refers to the value that can be extracted by strategically ordering, censoring, and placing transactions in a blockchain. The major forms of MEV include arbitrage, sandwich, and liquidations. Additionally, there are other types of MEV, such as long-tail MEVs, Cefi-Defi MEVs, Cross-chain MEV, and Statistical Arbitrage. We mostly focus on atomic defi MEV in this article.

In order to address the issue of a significant portion of MEV being captured by searchers and flowing outside the Sui community, the implementation of an open auction market becomes crucial. This auction market serves to capture the majority of MEV values and redistribute them within the Sui ecosystem.

Among the various auction market designs, the flashbots style (i.e. MEV-boost) auction stands out as the most prominent in other chains. This type of auction enables maximum value extraction, with validators being the primary beneficiaries. However, it fails to ensure a fair distribution of value since the parties at the beginning of the MEV supply chain, such as end users, dApps, wallets, and RPC providers, do not receive a share of the MEV value.

To address this limitation, it is important to explore alternative approaches that incentivize the growth and liquidity of dApps using the extracted value. By devising improved mechanisms, we can ensure a more equitable distribution of MEV value, benefiting all participants involved in the ecosystem.

1.2. **What makes Sui different.** There are several major differences on Sui:

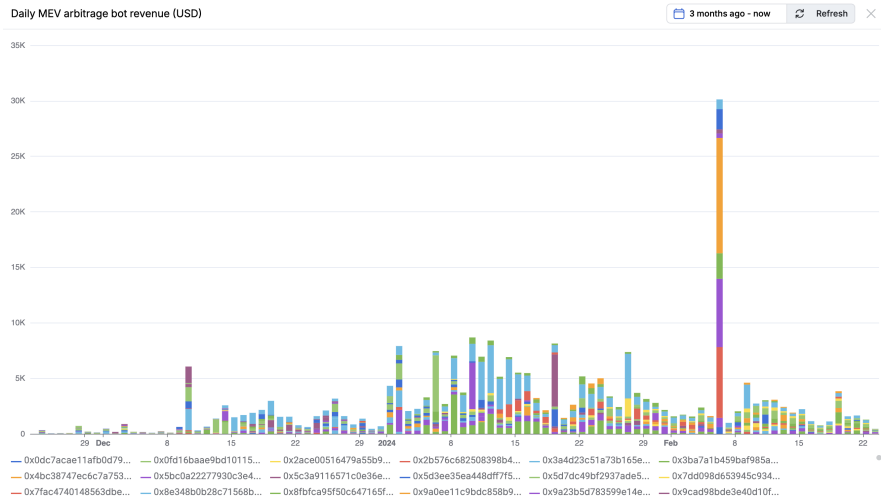

FIGURE 1. Current Sui MEV – Simple DEX arbitrage only

- Sui is currently at a different stage of development and may have different priorities, such as prioritizing stability over new features.
- The multi-proposer/DAG consensus model of Sui makes it challenging to directly fork existing auction systems.
- In the current MEV landscape of Sui, arbitrage and liquidation are the most common forms of MEV observed on the network. Sandwich attacks, on the other hand, are not detected frequently and would require individuals with certain system privileges, such as the ability to view unconfirmed transactions.

With the recent data we have collected per above Figure, we estimate that there will be **$2M** extracted value per annum from MEV that will be redistributed back to the Sui community and stimulate the growth of the community. This also implies that if there is no effective value redistribution mechanism, said amount of value will go away from the Sui community, along with incurring certain negative externalities.

2. System Design

2.1. **Design Principles.** The MEV opportunity (backruns) is often the result of a user initiating a transaction (or an update to the pricing oracle) that potentially creates an imbalance in the market. The general principle is to bring this opportunity into a market where an auction is held that results in a significant portion of the value captured being contributed back to the Sui community. We strive to adhere to these principles:

- To minimize the impact on stability and latency of the Sui chain.
- To start and make progress from a practical point of view, avoiding major system changes or introducing excessive assumptions on the underlying design of Sui, including but not limited to Bullshark [5] and Mysticeti [6].

- To hold auctions only when there are imminent user transactions that are originating on Sui.
- To start with a conservative approach that allows only simple bundles to be submitted as searcher's bid (such as arbitrage or liquidation).
- Redistribution of values should be based on community voting.
- Eventually evolve to become fully trustless and decentralized.

There are also other kinds of MEV opportunities that are not created by an imminent on-chain user transaction, but as a result of certain off-chain activity (for example, a CEX price change or an off-chain market order). Since Sui is a low latency chain, these opportunities should be captured through regular High-Frequency trading (HFT) and probably gas bidding.

2.2. **Proposed Solution.** The system works as follows:

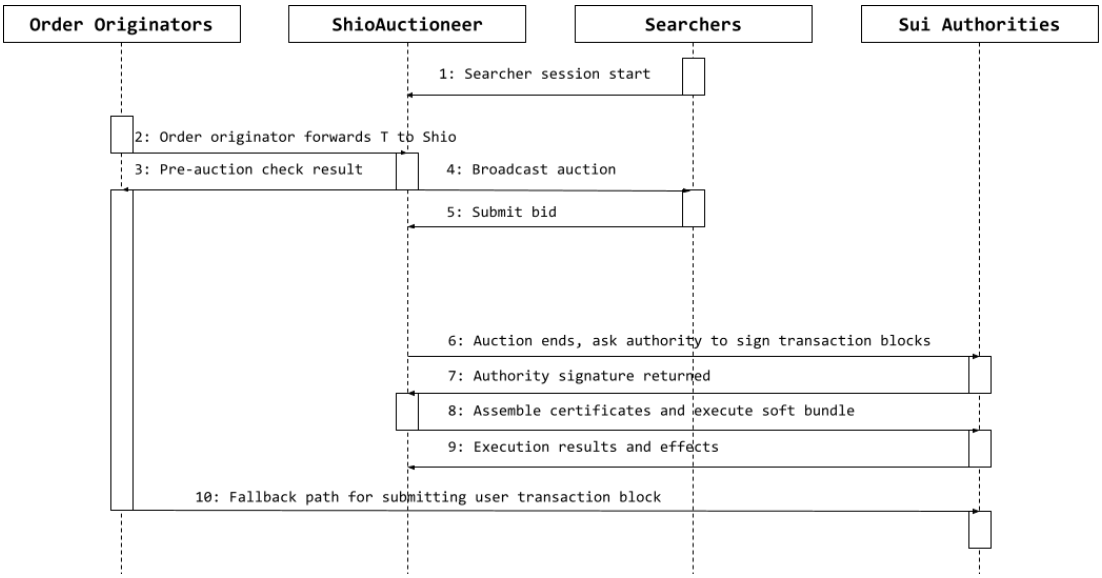

FIGURE 2. Shio Auction Flow

(1) **Searcher session start.** Searcher connects to ShioAuctioneer, establishes a session and starts listening for auctions.
(2) **Order originator forwards T to Shio.** The user initiates a transaction block T, it gets sent to a Fullnode [2] controlled by the order originator. The Fullnode or its load balancer needs to be modified to forward the user transaction to ShioAuctioneer. A timeout $PreAuctionTimeout$ is needed, say $100ms$. The key is to provide a bounded delay as well as a fallback path, and can be changed as needed. Pseudo code:

```
// Before processing the user request.
match incoming_request {
  sui_executeTransactionBlock(tx) => {
    let add_delay: bool = match shio::handle_user_tx(&tx).await {
      Ok(v) => v,
      Err(_) => false,
```

```
    };
    if add_delay {
      thread::sleep(Duration::from_millis(500));
    }
  }
  _ => {}
}
```

(3) **Pre-auction check result.** ShioAuctioneer will quickly respond with a result indicating whether a delay needs to be added.
   - Delay will not be needed in case ShioAuctioneer deems that the user transaction block T establishes no MEV opportunity at all. No auction will be held in this case.
   - In case when ShioAucitoneer becomes not reachable or times out, the order originator's fullnode can proceed to execute the user transaction block, incurring a maximum delay of $PreAuctionTimeout$.
   - If a delay is needed, the order originator will need to sleep for $AuctionDelay$ before proceeding, where $AuctionDelay$ should be around $500ms$.

(4) **Broadcast auction.** ShioAuctioneer starts the auction that lasts $AuctionWindow$ (say $400ms$), broadcasts the execution effect from the user transaction block T, including the content of all created or mutated objects, to all searchers, searcher starts to compete to submit the best bid for the txn within the auction window (say $400ms$). As safety measures:
   - No transaction data (inputs, commands and gas data) is revealed to searchers, as a safety measure to **avoid blind front-running or back-running**.
   - The only pieces of information that are revealed to searchers are a) transaction block digest, b) gas price, and c) the execution result of T (as if from $devInspectTransactionBlock$).

(5) **Submit bid.** Searchers finish computation and submit their back-run transaction blocks (aka Bid) to ShioAuctioneer, to be specific:
   - The bid must has the exact same gas price as the user transaction block T.
   - The bid must contain the digest of T that the searcher bids against.
   - The bid must contain the changed DEX pool(s) as shared input objects.
   - The bid must contain a MoveCall to a designated Move contract that is managed by Shio.
   - In the call, a pure input representing the amount of the bid (BidAmount: u64) need to be specified, then a hot potato [7] is given, requiring the searcher to pay the amount of the bid by the end of the transaction block.
   - Before the end of the transaction block, the searcher will need to create a Balance object of BidAmount, pass it a designated function in the Shio contract, which consumes the hot potato.

- The bid itself must pass validation so that it can be signed by a validator.
- ShioAuctioneer simply rejects the bid if at least one condition above does not satisfy.

(6) **Auction ends, ask authority to sign transaction blocks.** Upon the end of the auction, ShioAuctioneer determines the winner. T along with winners' transaction blocks, will be sent to validators [3] for signing. Up to N bids, with respect to the order of BidAmount, will be included. This is to make sure that if a bid fails to execute, subsequent lower bids may still succeed.

(7) **Authority signature returned.** Signatures from authorities representing at least 2/3 stakes are returned.

(8) **Assemble certificates and execute soft bundle.** Cerificates are assembled, then submitted together to a trusted validator, using soft bundle API, which will be described in Subsection 2.3.

(9) **Execution results and effects.** The trusted validator facilities submission of the soft bundle, makes sure that all transaction blocks are included in the same Batch [8] before submitting to consensus. We assume the ordering of transaction blocks from the same Batch are respected (given that they have identical gas prices). After waiting for execution effects, the validator returns.

(10) **Fallback path for submitting user transaction block.** As a fallback path, the Fullnode eventually needs to execute T anyway, to make sure it does not get stuck. On happy path, if T has already been submitted or executed, doing this again has no unwanted side-effects.

2.3. **Soft bundle.** In the current implementation of Sui, transaction blocks are first validated, signed, then submitted to consensus for ordering (given that their input contains one or more shared objects), most likely done by different validators. The ordering of transaction blocks, if they have the same gas price, depends on many factors that cannot be easily controlled.

We seek a way to bundle together the user transaction block and the searcher's transaction block (maybe even multiple ones from multiple searchers), so that they can be executed in that specified order. Despite SenderSignedData [9] being a vector, its usage pattern today do not permit the extension to support bundling primitive without spending significant effort on refactoring.

Given that one of the design principles is to avoid major system changes, we propose Soft Bundle, an approach with minimum changes while ensuring a strong enough bundling semantic most of the times. To be specific:

- Implement a new Grpc method $HandleSoftBundleCertificates$ in AuthorityServer, that accepts and executes a vector of certificates, ensuring:
  - If at least one certificate cannot be executed, the whole request is denied.
  - If at least one certificate has already been executed, the whole request is denied.

  - If at least one certificate contains only owned object, the whole request is denied.
  - If at least one certificate has a different gas price than others, the whole request is denied.
  - If the number of certificates exceeds N, the whole request is denied. N should be small enough, say $N = 4$.
  - If the total number of BCS [11] serialized bytes from all certificates exceeds M bytes, the whole request is denied. M should be small enough, say $M = 64K$.
- The method makes sure that certificates from the same bundle are submitted to consensus together, from the same Narwhal worker, in the same Batch that is broadcasted to all other workers. This requires modification to consensus client (ConsensusAdapter) as well as Narwhal's BatchMaker, so that they accept a vector of certificates as input.
- After including the bundle in the same Batch, we can be certain that if this Batch is getting included in a Header in consensus, its internal ordering will be respected.
- PostConsensusTxReorder does not affect this because all transaction blocks in the same bundle have identical gas prices.

Given the implementation above, we believe SoftBundle possesses certain good properties:

- **Minimum changes.** We estimate the changes to be **about 200 lines** of Rust code. Majority components of the Sui system remain untouched.
- **Consensus-agnostic.** There is little assumption on how consensus work. We only made assumptions on a few core data structures, that are unlikely to be broken:
  - The internal ordering of a Batch is respected.
  - PostConsensusTxReorder is based on gas price AND using a stable sorting technique.
- **Easy to launch.** SoftBundle will work even if only one validator (as the server) and one Fullnode (as the client) have been modified.
- **Reusability.** SoftBundle primitive itself can become a building block in the ecosystem, to be further extended to usages for purposes other than MEV.

2.4. **Impact on the chain.** We acknowledge that Sui places a strong emphasis on stability and low latency. As our modifications to the Sui code base are minimal, without touching the core protocol, the risk of introducing new attack surfaces or vulnerabilities is relatively low. Our plan is to not fork but to merge the changes back into the upstream.

Another concern that may arise is the potential added latency due to the auction itself. To mitigate this, we have devised two approaches:

- We will simulate the user transaction block first to determine whether there is any DEX-related objects that are mutated. If none, there will be no auction held, incurring a delay no more than a simulation time plus

a network round trip time (which is almost guaranteed to be less than 50ms, if ShioAuctioneer is co-located with the Fullnode).
- Fullnode operators will have the option to opt-in or opt-out of the auction. Users seeking minimum latency or think that speed is more important than capturing redistributed value (or if MEV is simply irrelevant to them) can send transactions to a Fullnode without Shio auction enabled.

2.5. **Alternative Design Considered.** Our design primarily focuses on supporting soft bundle semantics with minimal impact on the Sui consensus. The auction occurs where user transactions are originated. Another design approach involves making more changes to the consensus. It could be based on knowing consensus round leaders ahead of time, allowing the leader of a round to host an auction for that round. This approach enables per-validator onboarding to the system and bears some resemblance to other Flashbots-alike systems. However, considering Sui's current stage of development and design principles, this approach is unlikely to be a practical solution for several reasons:

- Flashbots' whole-block auction relies on a 12-second block time and a highly deterministic world state (nothing changes during these 12 seconds), which is not the case for Sui with its constantly changing world/object states.
- Separation of concerns. The consensus layer better treats a transaction block as a vector of bytes, rather than relying on its content.
- The development of the consensus is evolving, and introducing this change could result in future technical debt.

3. Possible attacks and defenses

The system we have designed takes into account the possibility that each party may act selfishly to maximize its own benefits. It is important to note that there are **no additional risks to the Sui chain itself** other than potential bugs in the proposed Soft Bundle change.

Let's explore the possible attacks from different parties and our solutions:

3.1. **Searchers.** Searchers, being driven solely by short-term profit maximization, pose a significant threat and require immediate defense. They are willing to employ any means necessary to capture all profits. Searchers can utilize various attacking strategies, including:

- DDoS ShioAuctioneer to prevent others from submitting bids (Shio must defend against these attacks).
- Peeking at transactions and directly submitting transaction bundles to the validator (Shio's auction relay only discloses partial information about the user's transaction, rendering this submission invalid). Searchers, however, will be able to submit blind back-run transaction blocks. Due to the design of Soft Bundle, it is sufficiently hard for such transaction blocks to be included at wanted positions.

- Peeking at transactions and attempting to submit front-run transactions (Shio's auction relay only discloses partial information to searchers, making it extremely risky for searchers to carry out blind front-running. Additionally, there are several other common techniques to mitigate this).
- Manipulating the Shio smart contract state to bypass the auction or directly hacking the funds within the contract. (This requires careful design, implementation and auditing of the contract.)

3.2. **Order Originators.** Order originators inherently possess order flow advantages and can choose to engage in MEV activities without disclosing the transactions. However, historical evidence from existing systems in the Ethereum Virtual Machine (EVM) has shown that open market competition ultimately benefits them the most in the long run. In other words, without competition, it is unlikely for a DApp or wallet operator alone to maximize the value captured. Furthermore, the future token introduced by Shio can also serve as a mechanism to align the interests of all parties involved. Section 5 describes some ideas about how to make it trustless.

3.3. **Validators.** In general, validators are only incentivized to engage in cheating behavior if the potential loss from slashing is significantly smaller than the MEV reward. However, this is not the case for Sui at present (but it is the case for Ethereum, see [12]).

Furthermore, it is important to note that front-running is already feasible within the existing system. Implementing a strict solution to this issue would likely require substantial changes to the consensus layer, which is beyond the scope of the current work.

However, it is important to consider the potential attack from validators. A malicious validator could attempt to dismantle the bundle and steal the user's transaction (as the user transaction has already been signed), then submitting it in her own bundle. We argue that it is difficult for the malicious validator to form a quorum with super majority votes (2/3 in this case), as any other validator can only support either of the two bundles during consensus, because both bundles contain the same user transaction block. As long as ShioAuctioneer is sufficiently fast, this attack is unlikely to succeed unless the malicious actor controls at least one-third of the stakes.

A straw-man solution (outside consensus) can be based on challenging and slashing:

- ShioAuctioneer executes soft bundle with Validator A. This is recorded.
- One of the consensus worker of Validator A assembles a Batch including the soft bundle and broadcasts it to workers of the rest of validators.
- After consensus, each validator should have recorded the same Consensus Output, given the deterministic nature of consensus.
- Eventually, one may examine the consensus output of any validator, and find out which Batch contains the user transaction block.

- Whoever produced the Batch with the user transaction block included but without the soft bundle we submitted can be considered a suspect and gets challenged.

Another way to mitigate this, if not slashing, is to exclude identified malicious validators from signing, as long as they do not constitute more than 1/3 stakes. This will make it sufficiently difficult for validators other than the consensus leader of the particular round to carry out an attack.

Validators can also completely collude and censor the soft bundle. Thus it is important that they get a fair share of the extracted values.

3.4. **The Auctioneer – Shio Team.** The auction server is operated by the Shio team, giving them privileges such as front-running users, censoring transactions, and reordering transactions. However, we would argue that:

- **Front-running.** It can be easy to challenge the team if Shio did front-run a user transaction block, as ShioAuctioneer is the only party that possess knowledge about user transaction block's content. Front-running will be extremely risky for searchers, given that many important pieces of information are not revealed to them and cannot be easily deduced (for example, slippage).
- **Censoring.** It is impossible for ShioAuctioneer to censor a user transaction block, as the order originator will always proceed to submit it at a later time. However, it is possible to ShioAuctioneer to censor a bid from a particular searcher. The trustless version is described in section 5.
- **Reordering.** Within a bundle, ShioAuctioneer may choose not to order searchers' transaction blocks by its BidAmount. However, the team can easily be challenged. On the other hand, only simple bids are supported (which contains only one transaction block from the searcher), making it fruitless for ShioAuctioneer to reorder transaction blocks within a single bid.

Initially, the Shio team has sufficient incentives to maintain the system's integrity. However, the long-term goal is to gradually transition the system components to be on-chain or decentralized.

In essence, the system described above still relies on multiple trusted parties. The ultimate objective is to eliminate the need for trust and achieve a decentralized and trustless system. This will be a key focus for future development efforts.

4. Value distribution and Tokenomics

4.1. **Tokenomics.** The main purposes of the Shio DAO token (SHIO) are to incentivise Sui Defi Ecosystem Players (SDEPs) and getting users involved in the governance of the protocol. SDEPs refer to those who contribute order flow or process order flow, including decentralised applications, wallets, RPC providers and validators.

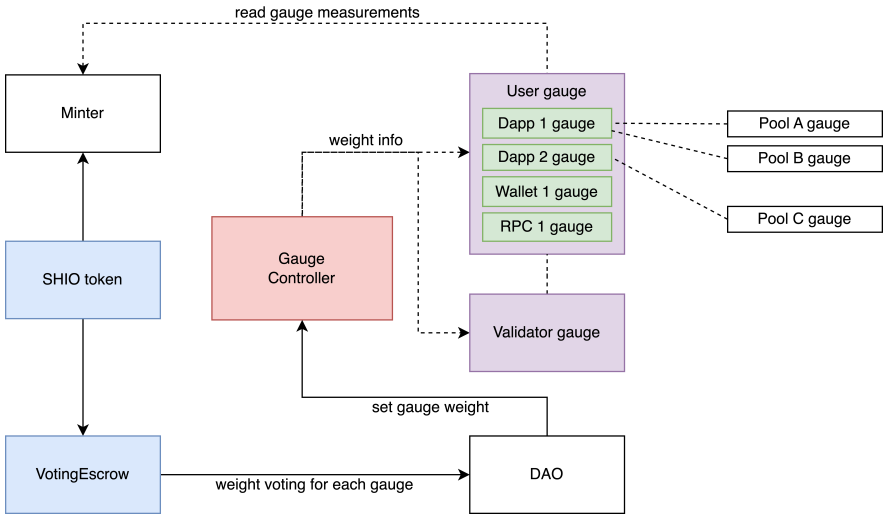

FIGURE 3. Shio DAO Voting Workflow

4.2. **Value distribution governed in a decentralised approach.** We adopt vote-escrowed Shio DAO token (veSHIO) , which represents SHIO that are locked in the voting escrow contract for a specified period, to drive long-term community growth and decide how value generated by the protocol is distributed to SDEPs.

The crucial part of the Shio governance is to decide the how incentives get allocated. We use the gauge system to weigh the contribution of all SDEPs. The veSHIO balance represents the voting power of a holder, which allows holders to vote on the weights of each gauge, down to the level of pool gauge. Each gauge carries a gauge type weight. Liquidity providers (LP) who voted-locked their SHIO tokens will get a boost in their liquidity incentives. The Figure below shows the workflow of voting.

This distribution design incentivizes LP through mining rewards, bribes and governance tokens. **Liquidity is important to drive the long-term health of Sui chain**, as it helps efficient trading, price stability and user adoption.

4.3. **Earning fees.** 100% of platform revenue goes back to Sui community. 50% of platform revenue is distributed to holders who voted-locked their SHIO, the other 50% of platform revenue is distributed to the SDEPs (where the contribution is measured by veSHIO voting). The split can be adjusted through parameter voting in the future.

4.4. **Veto committee.** Parameter votes, which can modify the gauge type weight, and emergency votes shall be decided by a veto committee. The veto committee consist of 7 members, comprised of a mix of the Shio, Mysten, Sui Foundation team and prominent figures within the Sui community. Each member has one vote.

5. Roadmap to more efficient extraction, trustless and decentralization

5.1. **Efficient Value Extraction.** The current Shio model, which only allows single user transaction bundles, may not be the most efficient value extraction model. One potential improvement could involve enabling more complex multi-target transactions.

However, implementing such a change would inevitably introduce the risk of sandwich attacks, and it remains uncertain if complex bundles can reliably be executed on-chain due to the dynamic nature of the state and the fixed 12-second block time on the ETH mainnet.

If there are no plans to implement a true PBS mechanism at the consensus layer, it might be more prudent to focus on supporting more efficient value extraction while ensuring fair value distribution. We will closely monitor on-chain behaviors and re-evaluate the plan accordingly.

5.2. **Make Shio Trustless.** One way to remove Shio itself as the trusted party is to implement:

- A ERC-4337 like UserOperation support in Move.
- dApps can act as auctioneers
- All bids can be handled within a smart contract.

The only trusted party left here is the relay to route UserOperations. The true trustless approach is to implement a Sui L2 with MEV primitives (Such that the "view txns" can only be done "on Chain"). However all of these seem to be long term thus we won't expand too much details here.

References

1. Flash Boys 2.0: front-running, Transaction Reordering, and Consensus Instability in Decentralized Exchanges
2. Sui Full node setup. https://docs.sui.io/guides/operator/sui-full-node
3. Validator setup. https://docs.sui.io/guides/operator/validator-config
4. Programmable Transaction Block. https://docs.sui.io/guides/developer/sui-101/building-ptb
5. Bullshark. https://arxiv.org/pdf/2201.05677.pdf
6. Mysticeti. https://arxiv.org/pdf/2310.14821.pdf
7. Hot potato pattern. https://examples.sui.io/patterns/hot-potato.html
8. BatchMaker. https://github.com/MystenLabs/sui/blob/70ef087cc48a5097e5617791c4eb7bbe7b473469/narwhal/worker/src/batch_maker.rs#L118
9. SenderSignedData. https://github.com/MystenLabs/sui/blob/a8fc78daee4445532436961ff4139a7d5bb61bfd/crates/sui-types/src/transaction.rs#L2035
10. Header in consensus. https://github.com/MystenLabs/sui/blob/a8fc78daee4445532436961ff4139a7d5bb61bfd/narwhal/types/src/primary.rs#L694
11. Binary Canonical Serialization. https://docs.rs/bcs/latest/bcs/
12. Sandwich bots hacked by validator. https://twitter.com/punk3155/status/1642771856758546434