# SUAVE, Anoma, Shared Sequencers, & Super Builders

Originally posted to Mirror.

Thank you to Rajiv Patel-O'Connor

, Tarun Chitra, getsqt, Ben Fisch

, Evan Forbes

, Christopher Goes, and apriori

for incredibly valuable insights, feedback, and review of this post.

Thank you to

Robert Miller, Hasu, Shea Ketsdever, Josh Bowen, and many others for great discussions around some of the topics here as well.

## Introduction

This post further analyzes SUAVE and shared sequencers as a follow up to my last:

This is the complete guide to:

- Sequencer decentralization
- Shared sequencers
- X-chain atomicity
- MEV-awareness for rollups
- Where SUAVE fits in
- Decentralizing provinghttps://t.co/WRiDrkG3BL

— Jon Charbonneau (@jon_charb) March 19, 2023

Part I - SUAVE

: I dive into its contemplated architecture, uses, limitations, and interactions. I won't focus on the specific PET implementations (i.e., SGX, MPC, FHE, etc.). I'll mostly treat that as a black box here.

Part II - Anoma

: Anoma is built on many of the same underlying ideas as SUAVE, but it takes a fundamentally opposite approach.

Part III - Shared Sequencers

:

- Use in atomic X-chain transactions and MEV capture
- Interaction with PBS and potentially centralizing forces on builders
- Economic allocation challenges
- Rollups' reduced flexibility to implement validator-enforced customizations
- Rollups' ability to implement VM customizations (e.g., deterministic ordering rules)

## Part I - SUAVE (Single Unifying Auction for Value Expression)

SUAVE will be an independent network which allows any chain to outsource two roles:

- Mempool - Users could send transactions to the SUAVE mempool instead of their "home" chain's public mempool (e.g., Ethereum).
- Decentralized Block Builder - SUAVE outputs (full or partial) blocks that other domains' validators can accept (e.g., Ethereum proposers could accept a block built by SUAVE).

The goal is to be the mempool and block builder for any

domain. Aggregating preferences in a single auction has several benefits:

- X-chain builders can earn more than single-chain builders.
- Efficiency gains from aggregating and clearing preferences in the same auction.
- A decentralized and privacy-preserving builder should attract more orderflow vs. centralized builders.
- This stuff is complicated. Most chains shouldn't have to focus on building advanced privacy-preserving primitives.

## Architecture

Flashbots describes SUAVE as having three components:

1. Preference Environment - SUAVE Chain and its mempool are specialized to aggregate user preferences from any domain. Users place bids for execution.
2. Execution Market - A network of specialized "executors" listen to the SUAVE mempool and compete to execute user preferences.
3. Decentralized Block Building - Executors can ultimately produce (partial or full) blocks for other domains.

The SUAVE protocol itself has two logical layers:

- Messaging layer (i.e., SUAVE mempool)

- Transmits preferences to be executed

- Settlement layer (i.e., SUAVE Chain)

- Settles payments to executors that fulfill preferences

SUAVE Chain is primarily intended to process transactions for:

- Bridging funds
- Initializing smart contracts which offer bids to fulfill preferences
- Submitting oracle updates about the state of other domains (to check that preferences were fulfilled)
- Claiming bids after a preference is fulfilled

However, SUAVE is a permissionless EVM chain, so anything could be deployed. Native transaction activity outside of its "intended" use is unclear.

The incentives to participate are:

- Executors

- Incentivized by bids

- SUAVE Chain Validators

- Incentivized by network transaction fees

The exact security model of SUAVE Chain is still TBD (e.g., L1, rollup, re-staking, etc.).

TLDR

- SUAVE has its own chain and mempool. Anyone can express preferences to SUAVE for any executor to fulfill. Whatever executor fulfills the preference can claim the associated bid (payment). Together, these pieces allow any chain to outsource their own mempool and block building.

## Simplified Transaction Lifecycle

I think of the typical SUAVE transaction lifecycle in four high level stages:

1. Preference Expression

- I tell SUAVE what I want done. I place a bid for whoever gets it doneIt's like a bulletin board for preferences.

1. Execution Optimization

- Some magic happens inside the black box. Executors figure out the optimal way to get me what I wanted.

1. Preference Settlement

- Executors go to the chain(s) where I want something done, and they make sure it gets done.

1. Payment Settlement

- After my preference has been fulfilled, the executor can claim the bid I placed back on SUAVE Chain.

Now let's walk through it in detail.

## 1) Preference Expression

You can think of users as having two kinds of transactions:

1. Smart Preferences
2. I have a preference I want fulfilled, and I'm willing to pay a bid to any executor that can get it done for me. I can express any arbitrary preference I want by writing a smart contract on SUAVE.

Example

- I want to close an ETH/USDC arbitrage between Optimism and Arbitrum. I'll pay 3 ETH to whoever gets it done for me.

1. Dumb Preferences
2. I'm just a regular user who wants to buy a shitcoin without 100% slippage. I just set my wallet RPC to rpc.flashbots.net and send my transaction:

Today

- Your transaction is routed to Flashbots Protect, where the centralized builder keeps your transaction private and includes it.

Future

- Your transaction is conditioned for you behind the scenes such that it'd be directed to anOFA like MEV-Share which feeds into SUAVE. The conditions of the OFA can be encoded as part of your preference for you.

There isn't a clear technical distinction, but the point is that "regular" unsophisticated users wouldn't need to personally code some complex smart contract logic to express their preferences to use SUAVE in a basic way.

Preferences are messages that a user signs to express a particular goal. They unlock a payment (bid) if the user's conditions are met. Preferences can range from:

- Simple Transaction

- Contain a simple payload to be executed on a specific domain (e.g., an ETH transfer on Ethereum).

- Abstract Intent

- Make an abstract statement of what I want to achieve and leave the optimal routing to the executors. This may involve an arbitrarily complex sequence of events across multiple domains.

SUAVE is introducing a new transaction type that provides a decentralized way to pass preferences. These will build on the existing properties of bundles (e.g., pre-confirmation privacy and no reverts) and allow for richer expression.

We could see innovation around preference expression to serve users' needs. For example, executors could specialize to "pre-process" transactions in ways that make them more valuable, such as:

- Paying gas fees on users' behalf
- Batching similar transactions together (e.g., trades with a coincidence of wants)

As more preferences are aggregated in one unified mempool, executors can better maximize all

users' welfare. For example, batch clearing is more efficient when more users have offsetting preferences. This is similar to the batching that CoW Swap solvers do today. However, the batching in SUAVE here could be more decentralized and completely generalized (vs. centralized and app-specific).

Note that SUAVE can still build blocks with transactions from any publicly available mempool as well, just like any centralized builder:

**Example Bids**

Users can express whatever preference they want via smart contracts on SUAVE Chain. This requires depositing funds on SUAVE Chain and placing a bid that an executor can claim if they fulfill your preference.

SUAVE is a stateful system where:

- States are denoted by s_1

→ s_2

→ s_3

...

- s

is the current SUAVE state

- S

is the set of all historical SUAVE states confirmed by the SUAVE Chain consensus

- exec(bid,s')

outputs b,e

- b

is the bid that is paid to address e

(the executor who fulfills the preference)

Considering a few examples:

In all cases - I have something I want done. If an executor gets it done for me, they can claim the 3 ETH bid against the oracles. Note that the example bids above are flexible in who the bid specifies as the executor able to claim the reward:

1. If my preference is fulfilled, whoever submits the transaction behind me can claim the bid. This assumes that the executor can reliably insert another transaction of their own (position 1) behind mine (position 0). They're effectively "tagging" themself as the executor who can claim the bid.
2. Similar case as the first example.
3. Given that the block is empty, the only person who could be the executor is the actual block producer (i.e., miner, validator, etc.), so they're specified as the executor able to receive the bid.
4. I just want someone to create a resulting state for me, and whoever sends the transaction that gets that done can claim the bid.

**Preference Expression & SUAVE Chain Block Times**

With this in mind, will SUAVE Chain's block times present a bottleneck in preference expression? Consider an example:

- If SUAVE Chain has 1s block times, and
- I want to express a preference for a domain that has 500ms block times

There are two possible scenarios here:

- Expressing my preference requires a SUAVE Chain onchain transaction - I'm now bottlenecked by SUAVE Chain's block times.
- Expressing my preference does not

require a SUAVE Chain onchain transaction - there's no bottleneck.

If this is a meaningful bottleneck, then SUAVE Chain would be pressured to have ultra-fast block times in order to service preferences for other domains with low block times. You'll get dragged down to the lowest common denominator.

Let's walk through an example preference lifecycle to see the flow:

1. Bridge funds to SUAVE Chain. However, if I already have funds on SUAVE Chain, this isn't needed.
2. Initialize a smart contract that allows me to express my preferences, and deposit funds.
3. If I need to make a new smart contract or deposit funds, then I'll need a transaction.
4. If this is a generic function, I can just reuse an existing smart contract (e.g., a strategy that I have deployed as a smart contract with funds in it).
5. If I need to make a new smart contract or deposit funds, then I'll need a transaction.
6. If this is a generic function, I can just reuse an existing smart contract (e.g., a strategy that I have deployed as a smart contract with funds in it).
7. Communicating my preference typically would not require a transaction to be mined on SUAVE Chain
8. Option 1 - [Create a signature (not a transaction

) that expresses my preference and pass it directly to the SUAVE executor marketplace](https://www.youtube.com/watch?v=b3o2YP6sxpg&t=3380s) to save on latency (this is a different network layer than the mempool).

- Option 2 - Use SUAVE's special transaction type carrying your signed commitment into the SUAVE mempool. This ensures the maximum number of executors will see it. (Note: there are currently very few details as to what exactly this new transaction type is, or how it achieves these properties.)
- Option 1 - [Create a signature (not a transaction

) that expresses my preference and pass it directly to the SUAVE executor marketplace](https://www.youtube.com/watch?v=b3o2YP6sxpg&t=3380s) to save on latency (this is a different network layer than the mempool).

1. Option 2 - Use SUAVE's special transaction type carrying your signed commitment into the SUAVE mempool. This ensures the maximum number of executors will see it. (Note: there are currently very few details as to what exactly this new transaction type is, or how it achieves these properties.)
2. SUAVE executors compete and/or collaborate to fulfill my preference on the destination chains.
3. Oracle updates SUAVE Chain after my preference was met, and the executor can claim the bid I made. This requires an onchain transaction, but it's of little concern (executors can get paid back after the fact).
4. If I need to make a new smart contract or deposit funds, then I'll need a transaction.
5. If this is a generic function, I can just reuse an existing smart contract (e.g., a strategy that I have deployed as a smart contract with funds in it).
6. Option 1 - [Create a signature (not a transaction

) that expresses my preference and pass it directly to the SUAVE executor marketplace](https://www.youtube.com/watch?v=b3o2YP6sxpg&t=3380s) to save on latency (this is a different network layer than the mempool).

- Option 2 - Use SUAVE's special transaction type carrying your signed commitment into the SUAVE mempool. This ensures the maximum number of executors will see it. (Note: there are currently very few details as to what exactly this new transaction type is, or how it achieves these properties.)

**MEV Bot Contracts**

Let's take a step back and understand how searchers operate today.MEV bots deploy custom onchain smart contracts to execute their strategies:

- Offchain

- The source code is first written offchain in an easily readable HLL (e.g., Solidity). While a DeFi protocol's smart contract would verify and publish this, a MEV bot wouldn't.

- Onchain

- Smart contracts only store the compiled bytecode onchain (e.g., EVM bytecode for Ethereum). This is not as easily readable as the HLL from which it was compiled. They're generally written in a manner which intentionally obfuscates them (i.e., it's difficult to de-compile the visible bytecode and figure out the logic behind them).

Smart contracts cannot execute by themselves (an EOA must trigger their code to initiate a transaction). So, searchers run

offchain logic and real-time monitoring (e.g., mempool transactions, CEX prices, chain state, etc.) then decide when to make transactions/bundles and trigger their onchain contract logic based on that information.

They can push bundles to them, expressing how they want something executed. For new strategies, they deploy new contracts. As opposed to just sending vanilla EOA transaction bundles on the fly, searchers often use smart contracts as well to provide finer control over execution:

- They can provide guardrails, adding in custom checks during execution simulation which can revert if some condition is not met (e.g., if they would incur more slippage on a trade than expected).
- More easily facilitate execution across multiple contracts in the same transaction (e.g., borrow flashloan from Aave + trade on DEXs + repay flashloan)
- Potential gas savings, such as writing something more efficiently than what a UI provides (e.g., directly interacting with a Uniswap pool vs. going through their router contract).

Downsides include:

- You're making your code public. Raw EVM bytecode is difficult to read, but given enough time you may be able to back out useful information.
- If you're not careful, your bot could get hacked or exploited if there's a vulnerability.

Let's consider a simple sandwich bot example. I'm a searcher with a smart contract SC

deployed onchain to execute my strategy. The transaction flow is as follows:

1. My trading system runs offchain logic to monitor the mempool, and it spots a target tx_user

(SushiSwap trade with a high slippage limit).

1. Trading system processes that info with transaction simulations and/or other heuristics to optimize and speed up the process. I decide whether or not I can trade on it, and what I'm willing to bid.
2. I send my bundle to builders via a JSON-RPC request. My bundle is composed of [tx_frontrun

, tx_user

, tx_backrun

], and I offer my bid via direct coinbase.transfer

to the fee recipient (it's also possible to bid via gas price). Using SC

to help manage on-chain execution, I can make the coinbase.transfer

conditional upon my validity conditions (e.g. only pay if I'm able to earn at least as much as I expected from the sandwich). I signed my transactions with my EOA. However, note that SC

will actually execute the trade (i.e., SC

has my funds in it, and that's what will execute the txs against SushiSwap, not my EOA). I'm just signing with my EOA to call the SC

.

1. Builder simulates the bundle, which includes running through the checks I programmed into SC

. If it reverts upon simulation (e.g., if someone gets the opportunity ahead of me, my conditions aren't met, etc.), then the builder won't include it. My transaction is successful though, and they accept my bid.

1. Builder submits block → relay → proposer includes the block.
2. EL nodes execute the transactions in the block sequentially. This includes my bundle in which the "bread" = my transactions to my SC

. SC

checks specific conditions in the bytecode around reversion. (E.g., checks that the sender interacting with it, my EOA, is authorized to do so and other criteria for execution are fulfilled. Note that adding more checks also increases my gas cost.). Then it has some branching logic to execute the strategy. It now successfully executes, and the chain state is updated.

Overall, searchers already

express preferences via smart contracts in a sense today, but they do not bid with them in the same manner as they would with SUAVE.

#### Bidding Language

For a unified auction to interpret a wide range of preferences, there needs to be some form of unified "language" that everyone can use. In SUAVE's case:

- SUAVE Chain will be an EVM fork
- All preferences will be expressed via smart contracts on SUAVE (EVM bytecode)

The Turing-complete EVM provides the requisite expressivity. However, one question is whether the EVM is too

expressive. Classical impossibility theorems exist in this area, showing how a bidding language which allows for arbitrary expression can result in unbounded computational complexity, rendering your auction impractical. In a world where SUAVE is looking to express arbitrarily complex preferences for arbitrarily many domains, that combinatorial explosion seems unclear.

There may also be concerns with griefing attacks here. For example, I might express a preference via a contract that always overflows the gas limit unless one particular input is provided. Then, nobody could simulate the actual condition for that contract until they have that input. The EVM doesn't provide great DoS prevention here.

The need for EVM smart contracts to express all preferences also seems suboptimal. If you're trying to implement complex logic such as:

- If price on Uniswap is > 2000 and < 2000 on SushiSwap → then trade on Uniswap first, and SushiSwap second
- If price on Uniswap is < 2000 and > 2000 on SushiSwap → then trade on SushiSwap first, and Uniswap second

Then writing custom smart contracts that implement this as a decision tree appears to be necessary. You obviously won't be writing out new smart contracts every time you want to express something to SUAVE - you'll have template contracts to execute strategies similarly to what was described earlier.

However, not every abstract preference a user could have will be serviced by existing contracts. It would appear beneficial to have a more native way to express preferences at the user intent level.

### 2) Execution Optimization

Next up, executors compete to fulfill user preferences across any domain.

#### Orderflow Auction (OFA)

You've probably heard of MEV-Share by now -Flashbots recently announced their beta version. This is the (trusted) v1 of the OFA which will eventually be worked into SUAVE. It'll take multiple iterations to improve it and reduce the trust in operators such as Flashbots.

Those details aren't the focus here though. The abstraction is that users send orders, and searchers (executors) competitively bid to give users best execution. Some variation of this would be a central part of SUAVE.

#### Programmable Privacy

Bids b

are composed of:

- Deposit k
- Payee e: S → A
- Execution predicate P(s)
- Encrypted data blob c
- Encryption predicate Q(s)
- Set of peekers ∑

In which:

- e

maps from SUAVE states $s \in S$

to the set of SUAVE accounts A

- Predicates P(s)

and Q(s)

are likewise over states of SUAVE

The result of executing a bid b

:

- If the current state of SUAVE s_cur

satisfies P

→ transfer the deposit k

to the e(s_cur)

account, modifying SUAVE state to s_next

- If the current state of SUAVE s_cur

does not satisfy P

→ SUAVE state remains unaltered

Within the bid, a user has three programmable privacy control knobs:

- Execution predicate
- Encryption predicate
- Set of peekers

Programmable privacy = allowing the user to selectively decrypt and reveal as much (or as little) of their data as they wish, and under what conditions. You can partially decrypt information such that executors can fulfill your preferences while also keeping various aspects of your economic preferences private for example.

A user can enforce conditions such as:

- "Parts of my Tx data can only be decrypted when condition X on the world state against these oracles is met"
- "I only want my Tx to execute if it's first in the block, otherwise it should fail."

**Credible Commitments**

SUAVE also requires credible commitments - we need to enforce certain conditions for execution. This is already part of MEV-Share with its "validity conditions." They're passed along with the user's transactions, stipulating conditions such as "the user must be paid ≥1 ETH for this bundle to be executed."

In the initial stages of MEV-Share, these conditions will not be enforceable - they'll rely on trusting the builder who's including them. Later iterations of the OFA will look to reduce trust in the enforcement of validity conditions by leveraging SGX.

**Fee Escalators**

Preferences can also include fee escalators - transactions which increase their gas price over time. They can even start with a negative value (i.e., an executor must pay the user for the right to execute it). This allows users to permissionlessly conduct an implicit Dutch auction for the right to fulfill their preference.

As an example:

- Tx starts with a fee of -$200
- Executing the Tx would allow someone to capture a $100 MEV opportunity
- Fee gradually increases as time passes

- Executor should pick up the Tx once the fee escalator has risen above -$100

Fee escalators can be powerful when combined with the programmable privacy described. Users are free to decrypt and reveal as much or as little as they wish.

If you keep this fee curve and your transaction information private, searchers could try to brute force optimize this against other bundles. You get the guarantee here that the user gets optimal MEV execution while giving permissionless real-time access to their orderflow. This just relies on a competitiveness assumption around searchers and validators.

**Decentralized Block Building (DBB)**

The DBB network aggregates preferences (many of which now have their execution paths optimized by executors) and turns them into blocks across domains.

DBB is described as one of the three logically separate components within SUAVE because not all domains even have a notion of block building and PBS. However, this DBB role is really a specialized instance of an executor within the execution marketplace.

We're used to a clear distinction shown between searchers and builders today, but in reality the lines are a bit fuzzy. Executors are an umbrella term for the actors who fulfill these preferences, and that can include the roles we see today in searchers and builders.

Today, builders build entire blocks for Ethereum. In the long-run, SUAVE's ultimate goal is to have this network of executors building blocks collaboratively, "snowballing" into building a full block. One executor can build a portion of an encrypted block, then another can add on more transactions, and so on. This collaboration is how to get fully DBB.

## 3) Preference Settlement on Destination Chain

Now it's time for the user's destination chain validators to accept (or reject) what the SUAVE executors are trying to fulfill.

SUAVE does not replace the mechanism by which a chain selects its blocks. They retain full control. Ethereum, Arbitrum, Optimism, etc. wouldn't be changing their fork-choice rules to opt into SUAVE. There are no protocol-level changes required for SUAVE to build a block for a chain.

For example:

- Ethereum proposers decide whether or not to include a block sent by SUAVE
- A rollup's decentralized PoS sequencer set can decide whether or not to include a block sent by SUAVE

Destination chain proposers may or may not be "SUAVE-aware" and natively integrated. For example, an Ethereum validator could profit switch between SUAVE bids and centralized builders, or it could simply not pay attention to SUAVE. It's more efficient if these validators are themselves SUAVE executors, but it's not required. Other actors could work to fulfill these preferences regardless.

Let's consider the transaction flow:

1. Searcher

- I want [tx_1

, tx_2

, tx_3

] mined. If they are, I'll pay a 3 ETH bid b

to the executor e

1. SUAVE

- Propagate and aggregate searcher input, ensure payments are valid

1. Executor

- Ensure validators perform economically optimal action

If Validator is SUAVE-aware

- Validator can profit switch b

against its best known mempool block. SUAVE will have native plugins for this where validators can directly listen to bids

and automatically profit switch over bids they're able to parse and control. Other actors translate these to bids that validators can control.

If Validator is SUAVE-unaware

- Executor gets the validator to include the bundle over PGAs or some other third-party channel (e.g., executor makes best effort PGA bids to get them included, or relays them as a bid via a third-party plug-in such as MEV-Boost).

As I've written previously, this means that SUAVE cannot guarantee atomic inclusion of X-chain transactions by itself. You need the proposers of respective chains to agree on atomic inclusion for that guarantee to be enforced.

**X-Domain MEV**

SUAVE allows you to express preferences for X-domain transactions. Let's consider an example. A user communicates their preference to SUAVE that they want two trades executed at the current block height to close an arbitrage:

- Trade 1 (T_1

) - Buy ETH at $2,000 on Rollup 1 (R_1

) in Block 1 (B_1

)

- Trade 2 (T_2

) - Sell ETH at $2,100 on Rollup 2 (R_2

) in Block 2 (B_2

)

These rollups may even have completely different length unsynchronized block times. R_1

could have 1s blocks and R_2

has 10s blocks. It's then perfectly reasonable that T_1

closes its leg of the arbitrage, but then T_2

fails.

That's why SUAVE is not

prescriptive about how preferences are actually achieved. SUAVE looks to be maximally flexible such that it can coordinate for any domain and user. Executing on different domains will therefore present different challenges. Different users will accept different outcomes and levels of risk.

If the user is only willing to take execution if both legs are filled, then the executor must hold the risk for them that only one leg executes. For example, the user can require the executor to fulfill both legs with their own capital, then they can exchange the funds for the user's locked capital only if both legs are filled. If they fail to fully meet the preference, then the executor is stuck holding the risk.

This requires sophisticated executors to price the execution risk of conducting the statistical arbitrage, which means less executors will likely bid for the opportunity (they may not have the upfront capital or want to warehouse the risk).

SUAVE can support either desired path. Users and executors can all have their own risk tolerance and interact however they wish. SUAVE can't provide "technical X-domain atomicity" on its own, but it can provide "economic X-domain atomicity" in this sense from the user perspective (though the executor may get stuck holding the risk).

## 4) Payment Settlement on SUAVE Chain

Now that the destination chain confirmed their own block, SUAVE Chain needs to become aware of that result. As mentioned earlier, bids are only unlocked for executors who meet users' preferences. Oracles are required to prove that these preferences were fulfilled elsewhere.

These oracles can be implemented however desired. In any case, these oracle contracts are responsible for importing external events into SUAVE's state.

A simple example would be an oracle contract which allows SUAVE bids to query Ethereum's history. I might want to bid for an empty block in 10 blocks. I can submit a bid by creating a transaction, and the transaction will output a payment if the oracle tells it that the Ethereum state has transitioned such that the payment should be made.

**Why Does SUAVE Need a Chain?**

SUAVE Chain plays an important role in preference expression, and is needed for payment settlement after the fact. Recalling the simplified transaction flow:

You might be asking a few questions here:

- Why not just have a standalone global p2p layer for SUAVE preference expression (something akin to Anoma's intent network), then settle payments on the destination chains?
- Why do we need SUAVE Chain for preferences and settlement? Even if a single chain is needed, why not use an existing one (e.g., Ethereum)?

**1) Economic Efficiency - Low Cost & DoS Resistant**

SUAVE has two requirements for transmitting preferences:

- Low Cost

- Relying on a p2p layer alone would also achieve this. In normal times (not overly congested), SUAVE Chain can operate similarly though. This requires changing the actual chain and introducing a new transaction type.

- DoS Resistant

- A blockchain can deter DoS attackers by forcing them to pay a cost for spam. A chain allows for imposing fees (by including preferences onchain) in the event of excessive network congestion. A standalone p2p network would not have the mechanism needed to charge such a fee.

**2) X-Domain MEV & Neutrality**

A chain is needed to settle payments for X-domain preferences. Consider the example earlier where a user submits a preference for X-domain arbitrage, but is only willing to pay if both legs execute. There must be some oracle verification after the fact to settle only if the entire preference was met. You can't just pay out on each leg of the trade (e.g., you could then pay for one leg of the trade, but then the other leg fails).

There needs to be some domain which provides this all-or-nothing payment option. Bridging assets to SUAVE Chain and settling everything there is one such option. That may be unacceptable to some users, but it needs to be a possibility as SUAVE is trying to support all potential models of preference expression and settlement.

This is best served as its own independent chain (versus an existing chain) to remain neutral when settling preferences across multiple domains. This will also be important as SUAVE moves towards providing services for all domains (not just Ethereum). It can have independent ownership and participation separate from any existing chain, which is likely needed to get buy-in from other domains, whether that be Cosmos, Solana, CEXs, etc.

**3) Flexibility**

SUAVE also needs to implement a wide variety of optimizations that would be unacceptable to existing domains. It needs to control the full stack chain to implement these. This could be as simple as relatively higher gas limits and lower block times - these are of course unacceptable to Ethereum.

Another example relates to the mempool. Flashbots has considered replacing Ethereum's current mempool within Geth with a different structure that's optimized for faster communication.

Flashbots has also considered making SUAVE Chain a rollup in the long-term. This could allow SUAVE Chain to use the rollup's derivation function to trustlessly access the state of Ethereum L1 and its other rollups This is valuable in reporting state transitions of other domains back to SUAVE Chain so that the conditional payments from bids can be unlocked to executors.

## Roadmap

SUAVE of course isn't live yet, and there isn't even a spec out. It's an ambitious vision which will take years to realizeThe plan is to ship it in phases:

- MEV-Share is an OFA that keeps user transactions private from searchers. It initially relies on Flashbots as the trusted matchmaker. Builders would be trusted to respect users' desired execution and payments.
- SUAVE Chain mainnet allowing users to send preferences to the Execution Market
- SGX-based OFA to remove trust in Flashbots

- SGX-based centralized block building to enable open but private orderflow
- SGX-based decentralized building network to allow for permissionless and private collaborative block building
- Onboard second domain to SUAVE and allow for expression and execution of cross-domain MEV preferences

In the longer-term, Flashbots intends to look into crypto-economics, custom secure enclaves, MPC, and FHE to further reduce the trust guarantees in the system.

### Summary

So, who's going to use SUAVE and why? A few high level users could be:

- Normie users

- I just want to send my swap into an OFA that'll give me good execution.

- Power users

- I want to buy a ton of ETH, so I express a preference to sweep liquidity across exchanges on many different domains (e.g., I can specify which chains and versions of wrapped ETH I consider fungible). This is a bit analogous to intermarket sweep orders (ISO) in traditional finance. You can think of it like a meta DEX aggregator.

- Searchers

- I run specific strategies, looking for MEV opportunities. I can express my preferences for specific opportunities I want to capture.

Some of the biggest open questions include:

- Preferences

- The best way to express this generic notion of "preferences" is unclear. Trying to build for the current blockchain stack (e.g., EVM smart contracts) inherits certain limitations.

- Privacy

- Nobody loves SGX, but that seems like the practical reality in the near to medium-term. The timeline and feasibility of moving away from this is unclear.

- Oracles

- The oracles back to SUAVE Chain for settlement introduce many complexities and questions.

Overall, SUAVE is dope. It's an ambitious and fascinating abstraction of how to express and fulfill any generic preference.

### Part II - WTF is Anoma?

In case SUAVE wasn't confusing and fascinating enough, I've got another one:

Anoma is another protocol under development which is very

reminiscent of the core ideas behind SUAVE. For starters, Anoma:

- Is an architecture

for blockchains. It's a framework.

- Is not

a blockchain.

Any chain can implement the Anoma architecture. You can have an Anoma L1, Anoma L2, whatever. A chain implementing the Anoma architecture is referred to as a "fractal instance" of Anoma, and all share certain homogeneous standards.

Confusingly, one of the planned fractal instances is also currently called "Anoma." For simplicity, I'll call it "Anoma L1" when I refer to this specific instantiation. It'll be an IBC-enabled L1 PoS chain. If I just say "Anoma," I'm talking about the architecture.

Remember, Anoma and Anoma L1 are completely separate ideas. Think of "Anoma" a bit like you think of "Cosmos". You can have a Cosmos L1, Cosmos L2, whatever. It's just referring to chains that share some set of standards. The standards that Anoma chains and Cosmos chains implement aren't exactly analogous, but it's a helpful simplification.

## Settlement + Counterparty Discovery

Every blockchain today provides settlement

- consensus agrees upon and finalizes some updated state of the world.

However, they're not directly optimized for counterparty discovery

(CD). For simple stuff, it doesn't really matter. If I want to send you some ETH, I don't need CD. You just give me your address. I submit a transaction to the Ethereum mempool which authorizes sending ETH from my address to yours.

That's not the case for more complex interactions like trading. This requires CD to:

- Help counterparties discover each other

, and

- Agree on what to settle

with each other

I know what I want (e.g., I want to swap 1 ETH for 1000 USDC), but I don't know how to get it. Just having that "intent" of what I want isn't enough to settle on-chain. I need a fully formed state transition.

You're probably thinking, "Ok cool, so what? I can just go to Uniswap or whatever other application for CD. That's what they're there for." And you're correct!

## Example - DEXs Today

Let's consider a CLOB as a simple example. You could put every bid/ask on-chain, but this is generally prohibitively expensive (and unnecessary). As a result, you'll see constructions like dYdX:

- Today

- Settlement all happens on-chain, but the orderbook and matching engine lives entirely off-chain (currently a single centralized server).

- V4

- The orderbook and matching engine will run entirely in-memory (off-chain) within validators. Each validator will host their own view of orders.

So there's two parts to the solution here:

- App-chain

- Lets you customize your chain to your needs, but this can introduce more work and complexity at least today. Also composability may suffer to some degree.

- Off-chain

- This scales, but it still comes with drawbacks such astrusting validators regarding MEV. For example, they can see everything in the clear and pull orders/ignore cancellations/front-run/give worse execution at the last second.

AMMs like Uniswap arose largely due to the constraints and gas inefficiencies of CLOBs. Everyone can just use the AMM contract as the central point of counterparty discovery. Current AMMs still have some obvious drawbacks here in terms of efficiency, and broadcasting your trade naively has problems:

- Sending transactions to public mempool = sandwiching

- Sending transactions to private mempools (e.g., Flashbots Protect) = trust and suboptimal execution/inclusion time

The Anoma architecture attempts to alleviate this problem by unifying both:

- Settlement

- The same stuff as before. You've got a validator set that agrees on stuff and settles it.

- Counterparty Discovery

- CD is built into the protocol itself. It's not left outside for others to deal with in an indirect manner.

Both of these tasks are required to move from a user wanting to do something → blockchain settles a state transition. The

difference is just that Anoma blockchains vertically integrate settlement and

CD. Most applications require CD, and building it into the core architecture provides them with some interesting benefits which we'll see shortly.

## Transactions vs. Intents

The key part of Anoma here is their notion of "intents." Conceptually, they try to achieve a very similar goal as SUAVE's notion of "preferences."

Overall, SUAVE appears at least somewhat constrained in its approach of preference expression vs. a system such as Anoma which is built from the ground up on the notion of "intents." Anoma doesn't plan for users to end up actually submitting "transactions."

First, let's look at how you actually

interact with a blockchain today. I want to swap 1 ETH for 2000 USDC, so I send a transaction to the Ethereum mempool:

- What you think transactions say

- "I authorize someone to take my 1 ETH, and give me at least 2000 USDC."

- What transactions actually say

- "I authorize this execution trace, and all opcodes being called can do arbitrary computation to change the state."

You never actually defined what you want

. You defined how you want to get it

.

If the code is clean, I'll send 1 ETH and receive 2000 USDC. If not and I didn't audit every opcode I signed (lol), maybe a proxy contract steals my money.

Existing protocols are designed with transactions as their fundamental unit, but transactions are completely non-intuitive

. They don't match how anyone actually thinks. You always think

in terms of some state change you want (e.g., I want a future state where I own 1 less ETH and 2000 more USDC).

In Anoma, intents are the fundamental unit by which users express their preferences, and intents work the exact opposite way

. Intents match what users are actually thinking:

Concretely, intents are off-chain signed messages that describe a partial state transition. They authorize some state preference that I want (e.g., I want a future state where I own 1 less ETH and at least 2000 more USDC). Intents are fully programmable - you can express any arbitrarily complex state preference. Maybe I only want to swap my ETH if it's sunny in New York and the Yankees won today. You get the idea. Intents are just arbitrary code that's evaluated at runtime by the settlement layer.

You can think of intents as "partial transactions", as they require some other parts to form complete transactions that satisfy users' constraints and enact a state change. Specialized middlemen called "solvers" (akin to executors) look through these intents to fulfill them. Solvers facilitate CD and route the intents to eventually be executed onchain by validators.

## Intent Gossip Layer

Users sign these binding intents via some client (e.g., wallet, dApp UI) then broadcast them to intent gossip nodes. This can be directed to certain node types, or undirected where it gets broadcast to as many nodes as possible. The intent gossip layer is just a p2p networking layer with a bunch of intents floating around (kind of like a mempool, but they're "intents" instead of "transactions").

Validators don't need to have any view of these intents. Most of them probably won't ever be settled onchain even. You could have billions of them floating around, and solvers can specialize in subscribing to certain "topics" they care about. Some form of specialization is likely to occur as solving becomes an NP hard problem, so a generalized solver would become intractable at scale.

A solver might only subscribe to process intents for:

- Anoma L1

- A specific application
- Only a certain intent type (e.g., ETH trading pairs) but across many chains

The intent gossip network can span over all Anoma fractal instances

. One globally connected intent gossip network can handle the intents for any and all Anoma fractal instances. UIs for applications can support deployment and order fulfillment across different security models if they wish. Intents are able to specify which security models they're willing to settle to.

As noted earlier with SUAVE, a global p2p layer could have some DoS concerns. In practice, solvers could operate much as builders do today (e.g., have a reputation system for known peers, blacklist as needed, etc.). It's unclear how robust this can be under extreme scenarios.

While having a blockchain such as SUAVE Chain allows you to impose some fees in the pessimistic case, it still doesn't prevent the need to do the computational search. This may still present some attack vectors (e.g., as described earlier with the contract overflow example).

## Solvers Match Intents

Solvers transform user intents into fully formed state transitions which meet users' desires. This achieves similar properties to what I described earlier with SUAVE batching as one generalized example:

- User A intent - I want 1 more ETH, and I'm willing to spend up to 2000 USDC
- User B intent - I want 2000 more USDC, and I'm willing to spend up to 1 ETH
- The solver can match these intents, settling them onchain as a single transaction

Similar to SUAVE, they also don't need to find perfectly offsetting intents. Solvers can route intents in complex ways with many intents that end up offsetting in aggregate, or they can fill them themselves if attractive. It's completely generalized.

Intents can specify settlement-conditional fees - they're only paid out if the intent is satisfied, settled, and confirmed onchain by consensus. This can be split amongst nodes involved in the gossiping and solving processes.

Anoma is also looking to implement various PETs to coordinate trustless operation on user data, which we can again treat as a black box for the purposes of this post:

Note that users can also send intents which describe a full state transition. For example, if Alice just wants to send 1 ETH to Bob, she doesn't need CD. If she doesn't need solvers to do any work, she can skip them and submit a full transaction herself (in a sense, acting as her own solver).

## Consensus & Threshold Decryption (Ferveo)

Right before solvers send transactions to the transaction mempool, they can encrypt them. Validators don't actually see the transactions that solvers send to them. They receive the transaction ciphertext:

- Plaintext = data in the clear
- Ciphertext = encrypted data

The particular scheme used is Ferveo - a distributed key generation (DKG) and threshold public key encryption protocol. Note that Ferveo is just a framework - different chains can flexibly implement different rules around it (e.g., requiring all transactions to be encrypted vs. auctioning off the top of the block in the clear). You may have heard that Osmosis also plans to use Ferveo.

Here's the basic transaction flow:

1. Validators generate a shared threshold public key. The shards of the corresponding private key are distributed

amongst the validators according to stake.

1. Solvers encrypt transactions to that public key, then send them to the transaction mempool.
2. Proposer orders encrypted transactions into a block, and proposes it to the network.
3. Validator set comes to consensus on block N, committing to the ordering. These consensus votes also include validators' respective private key shards. Once ⅔ of validators (by stake) sign to commit to the block, the transactions can be decrypted atomically. Everyone can see the result of the eventual execution of these transactions in N+1 as soon as block N is signed off.
4. The decrypted transactions are included in block N+1 (in the same order as committed to in block N) and executed.

This provides temporary

privacy. The privacy here is a means to an end. Because the validators don't see the transactions in the clear, they shouldn't be able front-run or censor them (if they are not colluding).

Note that implementing threshold decryption today alongside CometBFT consensus (Tendermint) would require ABCI++ to enable "vote extensions". ABCI++ isn't finalized yet, but it's expected in the near to medium-term.

## Anoma Example Application - Central Limit Order Book (CLOB)

With Anoma, you could potentially have MMs continuously send bid/ask limit orders as intents. You have an entire order book living as public binding intents.

Obviously MMs need to update their stale prices when information changes:

- Centralized traditional CLOB

- MM can race to send a cancel order to pull their stale bid. The centralized operator can cancel on FCFS.

- Fully on-chain CLOB

- This would require an on-chain transaction. You can only cancel an order if your cancellation transaction is included in the block in time.

With Anoma, MMs could potentially just intermittently update their intents. E.g., "I'm willing to buy X for Y, but this order is only good for block height Z." They can then continuously update their orders every block. They could become stale intra-block, as there's no method to cancel. There's a continuously updated order book of intents that users can execute on.

Additionally, this DEX isn't bounded by a single domain

. You don't need to think in terms of one DEX on one chain. Intents are fully composable, programmable, and global. I can specify exactly where and how I'm willing to get filled on my order.

## Anoma Example Application - Batching (e.g., CoW Swap)

As a quick TLDR, CoW Swap is an Ethereum application which matches trades via batch auctions using a variety of liquidity sources. Users submit trades to CoW Swap over some predefined batch auction period. Then:

- CoW Protocol first seeks a coincidence of wants (CoW) within the existing batch to offer the best price (e.g., if Alice wants to swap 1 ETH → 2000 USDC and Bob wants to swap 2000 USDC → 1 ETH, they can just be matched).
- If there isn't a CoW offering best execution, then trades can be settled via underlying on-chain AMMs directly (e.g., Uniswap) or via DEX aggregators (e.g.,1inch) depending on which pool/path offers the best price.

The protocol continuously runs these batch auctions. Within batches, this optimal order matching and routing is facilitated by many competing "solvers" (yes, that's actually what they're called here too). The winning solver is the one that can maximize traders' surplus either by the most optimal CoW, finding the best liquidity sources, or combining both in a single settlement.

However, you're relying on a central party to coordinate here in the case of CoW Swap today. Anoma could potentially decentralize this type of batching:

1. Encrypt all users' intents against the Ferveo threshold public key and send them to the validators.
2. Validators include the encrypted intents.
3. These intents are all tagged as being intended for the "Anoma CoW Swap" application/validity predicate on Anoma, so validators know to only decrypt them after the predefined batch period is up (say 10 blocks).
4. The validators decrypt all "Anoma CoW Swap" transactions which were included according to their validity predicate (so here, after 10 blocks).
5. Solvers can now look at the decrypted intents and compute over them according to their application's solver algorithm (e.g., everyone gets the same clearing price, best price is the optimal solution). The validity predicate can enforce these optimality criteria. Solvers are unable to inject new transactions into the batch.

## Anoma & SUAVE

Anoma and SUAVE appear similar in their notions of generalizing user intents/preferences. A market of solvers/executors compete to fulfill them, grounded in privacy technology to facilitate trustless collaboration. However, they have fundamentally opposite approaches.

Anoma's vision is for many fractal instances with a homogeneous architecture

. Many chains with shared standards get a lot of benefits as described above.

SUAVE takes a very different view - it's built in realization of the fact that many chains are likely to have completely heterogeneous architectures

. It's another layer in the stack to service this role for any

chain. It's agnostic to what your chain looks like. It just wants to optimize as the universal preference layer, outsourcing key components for any domain.

They're built with very different and possibly even complementary design goals.

## Part III - Shared Sequencers

I'll assume general familiarity with shared sequencers (SSs) such as Espresso or Astria here. If you're unfamiliar, please refer to my previous post.

## X-Domain MEV

As mentioned above, SUAVE can provide a best-effort "economic atomicity." This is the case with any builder. Only proposers (validators) have the power to guarantee transaction inclusion. SUAVE executors are not necessarily validators of other chains, and they can't guarantee atomic inclusion of X-chain transactions.

Conversely, SSs can guarantee the atomic inclusion and execution of transactions across rollups opted into it. SSs act as the shared "proposer" for all of their rollups. However, SSs such as Espresso and Astria do not execute transactions, so they can't guarantee that a transaction won't revert upon execution.

That's why SSs will need stateful builders to sit in front of them.

SUAVE could be such a builder. In the earlier example, a user wanted two trades executed at the next block height to close an arbitrage:

- Trade 1 (T_1

) - Buy ETH at $2,000 on Rollup 1 (R_1

) in Block 1 (B_1

)

- Trade 2 (T_2

) - Sell ETH at $2,100 on Rollup 2 (R_2

) in Block 2 (B_2

)

A SS with PBS can now get stronger economic guarantees throughout that whole process:

SUAVE anticipates that different X-chain atomicity approaches will arise, and it looks to support preference expression for all of them. Domains don't need to integrate with SUAVE directly, adopt a particular ordering model, or even have a notion of PBS. SUAVE is just a global "bulletin board" for preferences that sophisticated actors who understand the risk can compete to execute on across domains.

SUAVE can be seen as a demand-side aggregator for X-domain preferences. It exposes the tools necessary to securely express these preferences. In doing so, this incentivizes the development of solutions such as SSs. They can provide the "supply" of X-domain transactions, allowing for more efficient capture of X-domain MEV.

## Shared Sequencers → Super Builders?

It's well understood that X-domain MEV exerts centralizing pressure on block producers (e.g., validators/sequencers). There's an incentive for the same actor to control X-chain block production to more efficiently internalize X-chain MEV.

As I just described above, a SS can offer exactly these guarantees. And they don't need to carry heavy state on hand or execute transactions, so they can hopefully be lightweight and decentralized.

But we've just shifted the centralizing force to the builder

. We've done that on L1 Ethereum with MEV-Boost though, so what's the big deal? Keep validators decentralized, and shift

the centralizing force to builders.

However, there's a much stronger centralizing force here compared to something like L1 Ethereum. Building a block for L1 Ethereum is one thing, but doing it for arbitrarily many high throughput domains at low latency is another.

Overall, this stack may drive up the resource requirements for builders in terms of software/hardware level transaction execution, X-chain inventory management, balance sheet size, and risk taking. To be a competitive block builder, it may become a requirement to fulfill these conditions.

To be clear, centralizing forces from X-domain MEV exist for builders whether for a SS or any other domain. However, there's a distinction here:

- Builder for Other Domains

- There is still a centralizing force due to X-domain MEV economic incentives. However, it's not a strict "requirement." Today, the barrier to entry as a builder is lower (i.e., you could choose to just be an Ethereum L1 builder).

- Builder for SS

- Incredibly high requirements may be completely unavoidable to even enter the market. It seems likely that a single builder will need to build for all SS rollups.

As mentioned, SSs need some form of PBS. It seems likely that this takes the simple form of one auction for one mega auction across all SS rollup blocks. If the PBS interface here is building one mega block for all SS rollups, then that solidifies the most extreme requirements.

In theory, a SS could run more granular auctions for each individual rollup it sequences for. Then it would need to interpret this arbitrarily high number of auctions, checking for conflicting preferences. For example:

- Builder A is bidding for atomic inclusion of blocks B_1

& B_2

for R_1

& R_2

- Builder B is bidding for atomic inclusion of blocks B_2

& B_3

for R_2

& R_3

The SS proposer would have to comb through all the bids, checking for conflicts, and optimizing the merging of them. That probably sounds familiar - that's what a builder does.

So in practice, there will likely be a builder aggregating across all domains and interfacing with the SS proposer in a single large PBS-style auction.

Basically, SSs appear to be speedrunning the Endgame. There's a fair argument that this is inevitable anyway, but it's an open question still.

We'll likely see both SSs and "traditional" rollups decentralizing their sequencer (e.g., implementing a simple consensus set) prior to SUAVE. However, decentralizing sequencers (proposers) is not enough. We also need to consider rollup block building.

It seems likely that some variation of proposer-builder separation (PBS), protocol-owned building (POB), etc. is destined to arise on rollups in the near to medium-term. This should be an area of focus as they consider how to decentralize.

## SUAVE X-Domain Decentralized Building

SUAVE is a decentralized block builder, so does it solve the problem here? Not entirely. It's still unavoidable that if you're building a mega block for all the rollups on a SS, you have to meet the high requirements. SUAVE executors would be stuffing every rollup on the SS into their SGX to end up building a full block for it (or use other forms of cryptography down the road).

The technical challenge is one obvious question, but the point is that SUAVE isn't just about lowering the hardware requirements for builders regardless. What it could help with is making the block building process more collaborative and trustless. You could have many builders each contributing a piece to the blocks they output.

## Economics

Now for the really hard part - incentives.

The real alignment problem is alignment of incentives across components of the modular stack

— jill gunter (@jillrgunter) March 31, 2023

Figuring out the economics of SSs may prove to be the most challenging component. First, the nice part - SSs should allow their rollups to capture more aggregate value from MEV.

Let's consider a simple example. We have 10 equal (isolated) rollups whose validators capture $1mm each per year. In aggregate, they capture $10mm per year.

Now those same rollups all decide to opt into a shared sequencer. There's no reason that $10mm should go away, that's still there. Maybe it's even $11mm due to better bridging and interoperability → more activity overall. But now there's even more value for them to internalize from X-chain MEV. Let's break that down.

#### MEV Capture Today

Basic single-chain MEV strategies are commoditized. If you're running atomic arbitrage between ETH/USDC pairs on Ethereum L1 SushiSwap and Uniswap, you're bidding almost all of that value back to the Ethereum proposer. It's effectively riskless profit.

X-chain MEV is the opposite. Validators can't make X-chain atomic commitments. X-chain MEV can only be captured probabilistically by sophisticated searchers running statistical arbitrage. They have to manage inventory across chains and warehouse risk → market is riskier and less competitive → a lower % of the MEV gets bid back to validators.

#### MEV Capture With Shared Sequencers

Now validators (SSs) can

make X-chain atomic commitments. X-chain MEV can be captured with high confidence by searchers running atomic arbitrage. Extraction becomes highly efficient and competitive → a higher % of the MEV gets bid back to validators (SSs).

That's great for the rollups! They were capturing $10mm in aggregate before, and now let's say they're capturing $12mm. But wait - how do we split that up?

The SS has a lot of power here. They now decide the transaction inclusion and ordering for all rollups by default → they get first dibs on MEV. So we need to decide how to divvy that up. Having a "perfect" allocation mechanism which would need to simulate all possible outcomes is likely an outright impossibility.

You appear to broadly have two paths then:

- More granular auctions - The logical way to do PBS with a SS is one big block for all the rollups. Is it possible for the SS to run many sub-auctions for the individual rollups, or some other creative mechanism?
- Decide on a "good enough" allocation mechanism - Just take an opinionated but transparent stance on what's "fair" (e.g., split the revenue equally with all rollups, though even this isn't easy).

I certainly don't have the answer here today. This is a fascinating area of open research.

If the above is figured out, will rollups be ok with sharing the pie in a "good enough" way? Or do they go full Game of Thrones on each other? Honestly, we've seen a bit of both, so I'm not sure. It'll be an interesting political and social experiment to watch play out regardless.

## One-Size-Fits-All

When rollups opt into a SS, they're opting into somewhat of a "one-size-fits-all" model.Some even think they look like one big rollup. It's a realization that every chain is destined to be influenced for the sequencing of other domains.

Even Ethereum is influenced by Binance. Ethereum ←→ Binance is indeed the largest MEV source in crypto today. Several of the largest builders engage in this statistical arbitrage, and this has centralizing spillover effects into Ethereum.

That's the reality of X-domain MEV. There are tradeoffs in how you choose to address this influence.

- "Sovereign" end of the spectrum

- Many distinct chains with entirely unique and decentralized validator sets. Infeasible in practice due to the centralizing pressures.

- "Non-sovereign" end of the spectrum

- One node for one chain to rule them all. If you want a seat at the table, you co-locate. Feasible, but what's the point anymore?

Opting into a SS allows a rollup to mitigate the centralizing force of X-domain MEV on their validator set (because they can operate X-domain). Its nodes can hopefully be lightweight relative to the number of chains they service, and they can effectively internalize the X-domain MEV. Better interoperability also reduces the impact of X-chain UX.

But, this may come with some tradeoffs:

- Builders

- As discussed earlier, this construct appears likely to incentivize highly centralized and resource-heavy builders.

- Rollups

- Rollups sacrifice to an extent on customizability. You'll be limited in enforcing many of the things that a chain with its own validator set could do.

In my view, it wouldn't make sense to implement another validator set within your rollup in many cases (as another round of processing before or after the SS) for several reasons:

- Ease

- Part of using a SS is for the ease of deployment. If you're implementing a second validator set, that's no longer the case.

- Lose Guarantees

- If your consensus can arbitrarily include, exclude, and reorder transactions after the SS, then it can no longer give you any useful inclusion guarantees.

- Stronger Trust Assumptions

- You want to inherit the real-time CR and liveness of the decentralized SS's pooled security. If your second consensus can censor you/have a liveness failure after the fact, that's lost. You're only as strong as your weakest link.

The tradeoff - your rollup won't be able to implement features which require a validator set to enforce with discretion

. Examples could include:

- Threshold decryption (e.g., Osmosis and Anoma using Ferveo)
- Variations of FCFS (e.g., Arbitrum exploring Time Boost or FBA-FCFS)
- Validator in-memory order books (e.g., dYdX)

To varying degrees, the tradeoffs and associated complexities are just so high that it's not worth implementing separately as a rollup on top of a SS in my view.

Let's consider the example where you want threshold decryption for your rollup. Let's say a SS includes encrypted transactions to which it does not have the decryption key (your rollup's own nodes have the decryption keys). This becomes problematic - you have an entirely separate consensus set with the power to arbitrarily halt your rollup if they choose not to decrypt. There's also a challenging timing mismatch with the SS.

Your validators should ideally be the same group with the threshold decryption key shares, and they should have the same quorum as your consensus. Then, they can just include their key shares as part of their consensus votes, automatically decrypting as they sign off on a block.

For this reason, you'd want the SS itself to implement threshold decryption natively if you want it for your rollup. To be clear, a SS could do this! But, the tradeoff is that it still leans closer to a "one-size-fits-all" approach in many cases, leaving SSs room to differentiate on various features.

As an example, this seems somewhat against the grain of the Cosmos mindset. There, we see applications that want their own customized chains, and they want their own validators to enforce those rules. It could be infeasible to implement many of these customizations if opted into a SS.

## Retaining VM Control

While these rollups may look like one big chain in some sense, there's a key distinction here in that rollups opted into a SS can always fork away

(e.g., if the SS is extracting too much value). Valuable shared state is what's hard to fork away, but SSs don't have this on their own. They're effectively just a service provider. A rollup can always just swap out their SS for some other sequencing mechanism with only a minor hardfork.

Additionally, rollups certainly aren't handing over total control. They still have complete flexibility to customize their VM

as they wish. They can even enforce complex transaction ordering rules by encoding them deterministically in the state transition function (STF) of their rollup. For example, "the first transaction in my block must have oracle update XYZ to be valid" or "this type of batch must be right after it" etc. This could even include customizations such as Osmosis' ProtoRev Module to automatically internalize cyclic arbitrages.

As discussed above, you wouldn't want to have each rollup do a full round of non-deterministic reordering of transactions after the SS. So long as the VM rules are deterministic though, builders can create blocks within the bounds of these rules. Then rollup nodes can interpret the SSs output according to their rollup's own deterministic rules. This doesn't require a discretionary validator set to do anything.

## Conclusion

SUAVE is an ambitious attempt to unbundle the mempool and builder role for any chain. It looks to enable the expression and execution of any arbitrary preference in a trustless and collaborative manner.

Anoma is another idea to address some of the same underlying challenges. However, it takes a radically different approach by rebuilding the entire blockchain stack, trying to get many chains to share the same standards.

SSs could enable many chains to feel and act more like one chain again. However, they come with tradeoffs, including somewhat reduced flexibility, challenging value attribution, and potentially higher-resourced builders.

In any case, rollups are coming, and so are decentralized sequencers (hopefully). They're likely to arrive before something like SUAVE, so rollups need to think more about how to responsibly incorporate the builder role into their designs.

Otherwise, Vitalik might just be right about the Endgame after all:

Block production is centralized, but block validation is trustless and highly decentralized.