Written by Katie Chiou, Benjamin Funk

We've seen a lot of lively discussion (and sometimes confusion) around

1. What a sequencer actually

does–is it building blocks? Is it proposing blocks? both?

1. How does MEV look different on L2s?
2. What are intents, preferences, and meta-transactions? Why do they matter?

What a sequencer actually

does–is it building blocks? Is it proposing blocks? both?

How does MEV look different on L2s?

What are intents, preferences, and meta-transactions? Why do they matter?

.css-1wpjqbx{height:24px;width:24px;display:block;}

.css-1k55id8{visibility:hidden;border-radius:0.75rem;overflow:hidden;padding-left:1rem;padding-right:1rem;width:100%;display:-webkit-box;display:-webkit-flex;display:-ms-flexbox;display:flex;-webkit-box-pack:center;-ms-flex-pack:center;-webkit-justify-content:center;justify-content:center;max-width:32rem;}@media screen and (min-width:768px){.css-1k55id8{padding-left:0px;padding-right:0px;}}.css-1k55id8>div{margin-top:0!important;margin-bottom:0!important;}

We want to provide an overview of what answers we have to these questions today, why they matter, and how they're all related. To frame the discussion, we'll cover the following:

PART I:

A step-by-step look at how smart contract rollup transactions work today, and what role sequencers play in the process – transaction ordering, leader election, L2 consensus, finality

PART II:

Shared sequencers and what they mean for decentralization + MEV implications – cross-domain MEV + atomicity

PART III:

Rethinking the transaction lifecycle, order routing, and proposed frameworks – SUAVE, Anoma, etc.

PART I: SMART CONTRACT ROLLUP TRANSACTIONS

Let's walk through how a smart contract rollup transaction works today.

1. User signs transaction, transaction gets sent to sequencer
2. Sequencer orders transactions
3. Sequencer produces pre-confirmation
4. Sequencer posts to DA layer + Rollup node posts updated state to L1 bridge contract

User signs transaction, transaction gets sent to sequencer

Sequencer orders transactions

Sequencer produces pre-confirmation

Sequencer posts to DA layer + Rollup node posts updated state to L1 bridge contract

1. User signs transaction, transaction gets sent to sequencer

Pretty straightforward.

2. Sequencer orders transactions

Pause. “Ordering” here suggests two parts – deciding how the transactions are ordered (transaction ordering) and who constructs the block (leader election).

You can also think about this distinction as building and proposing. It’s worth noting, however, that the exact order in which these events take place depends on the assumption that a sequencer is responsible for both roles (building and proposing), versus whether the roles are unbundled.

Currently, the way rollup sequencers function is that there exists a single, centralized sequencer

run by the team. This is not ideal for several reasons:

- Liveness:

A single, centralized sequencer is highly efficient, but could also provide weaker real-time liveness—the guarantee that the system is always available and functioning correctly. If the sequencer goes down, what backstops/backups are there to make sure transactions are ordered? One solution is to decentralize the sequencer set; however, decentralization can introduce complexities that may compromise liveness. While Espresso’s piece explains this in depth, the main point is that, if decentralized sequencers also engage in consensus, decentralization can lead to worse liveness if synchronicity (messages between nodes are delivered within a known amount of time) is lost. The extent to which losses to synchronicity affect liveness depends on the categorical choice of consensus mechanisms (dynamically-available vs optimistically responsive).

- Censorship Resistance:

How do you ensure that the single, centralized sequencer is not censoring transactions? Technically, users can already force

withdrawal/inclusion of transactions directly through the rollup contract, but this is expensive and inefficient. A main benefit of a sequencer is that it amortizes the cost of posting transactions to the contract across a batch.

- Centralized chokepoint:

A single, centralized sequencer has the autonomy to maximize rent extraction to the detriment of users.

Liveness:

A single, centralized sequencer is highly efficient, but could also provide weaker real-time liveness—the guarantee that the system is always available and functioning correctly. If the sequencer goes down, what backstops/backups are there to make sure transactions are ordered? One solution is to decentralize the sequencer set; however, decentralization can introduce complexities that may compromise liveness. While Espresso’s piece explains this in depth, the main point is that, if decentralized sequencers also engage in consensus, decentralization can lead to worse liveness if synchronicity (messages between nodes are delivered within a known amount of time) is lost. The extent to which losses to synchronicity affect liveness depends on the categorical choice of consensus mechanisms (dynamically-available vs optimistically responsive).

Censorship Resistance:

How do you ensure that the single, centralized sequencer is not censoring transactions? Technically, users can already force

withdrawal/inclusion of transactions directly through the rollup contract, but this is expensive and inefficient. A main benefit of a sequencer is that it amortizes the cost of posting transactions to the contract across a batch.

Centralized chokepoint:

A single, centralized sequencer has the autonomy to maximize rent extraction to the detriment of users.

Sequencers on L2s, in addition to largely being centralized entities,also operate similarly to both an L1 builder and an L1 proposer–they both build the blocks with transactions from users and searchers and propose blocks to their underlying layers. Centralized sequencers operated by L2 teams run full nodes, meaning they're responsible for ordering transactions, committing them to the data availability (DA) layer, and executing transactions such that they can post validity or fraud proofs to the L1 bridge contract.

Decentralizing the sequencer set means removing the trust assumption that the team(s) running the sequencer(s) will not act maliciously or in an extractive manner and unbundling the roles within sequencing so that they specialize in ordering and allow the computation of fraud and validity proofs to be permissionless.

Transaction Ordering

A key competency of sequencers today is transaction ordering — how are transactions packed in a block (MEV!)?

Transaction ordering methods can usually be categorized in three (not mutually exclusive, often used in tandem) ways: auctions, encryption, and fair ordering.

- MEV Auction (MEVA):

Sequencers can permissionlessly bid in an auction for the right to order transactions, with rights awarded to the highest bidder (could be slot-based, block-based, time-based, etc.). Designing the optimal block auction is a difficult task, given the auction can be designed around several vectors: block vs slot-based, time vs bid-based, or batch vs streaming-based. Moreover, MEVA can introduce unwanted latency.

- Encryption:

The basic idea of private/encrypted mempools is that users submit encrypted transactions, and block producers commit to these transactions before decrypting them. Encryption doesn't inherently provide a mechanism for ordering transactions, but it can be used as a method to ensure the pre-execution privacy of transactions so that the sequencer's ability to extract MEV through transaction reordering or censorship is restricted. Sequencers receive and commit to these transactions in their encrypted form. The decryption process, which reveals the actual transaction data, can occur at a later stage when transaction validity is being verified. One of the major challenges to utilizing encryption is that it creates a vector for DOS attacks. Shutter's approach to solving this issue implements the requirement for the user to include a signed envelope within the encrypted transaction that covers fees (instead of specifying a gas limit) at the time of its inclusion within a block (not execution). While metadata leakage becomes a vector for MEV here, it's possible ZK proofs could play a role in preventing that issue. Different methods of encryption offer different drawbacks/are in different stages of development but a few are summarized below:

- Secure Enclaves:

Using trusted hardware technologies like Intel's SGX or AMD's SEV, sequencers work on transactions within a secure, isolated space. To prevent MEV extraction, additional rules (like deterministic ordering) need to be enforced within the enclave.

- Threshold Encryption:

A group of block producers jointly decrypt encrypted transactions for execution after ordering, distributing the decryption responsibility to limit potential MEV extraction. In the Ferveo protocol, for example, a group of validators produces a shared public key, dividing its corresponding private key into individual shares. Transactions are encrypted with this public key, and require a threshold of validators pooling their private key shares in order to be decrypted. After decryption, transactions are executed strictly as per the committed order, preventing the insertion of MEV-inducing transactions. However, it's critical to note that the complexity of key generation, sharing, and management escalates considerably among validators in threshold encryption schemes. This makes achieving high levels of decentralization (comparable to Ethereum mainnet) practically unattainable, limiting such schemes to between 100-999 validators.

- Timelock Encryption:

Users send transactions in a time-locked "safe" to the sequencer. The sequencer includes these locked transactions in the block, and they are only "unlocked" (decrypted and executed) after a certain time or condition is met, keeping the transaction details hidden from the sequencer until execution.

- Fair Ordering:

Fair ordering protocols/frameworks that implement specified ordering policies can be used in tandem with decentralized sequencer networks, such as that of Themis or Aequitas. Chainlink Fair Sequencing Service is another example. However “fair” ordering as a concept is hotly debated. A recent talk on fair ordering you may have seen recently. Even more on fair ordering.

MEV Auction (MEVA):

Sequencers can permissionlessly bid in an auction for the right to order transactions, with rights awarded to the highest bidder (could be slot-based, block-based, time-based, etc.). Designing the optimal block auction is a difficult task, given the auction can be designed around several vectors: block vs slot-based, time vs bid-based, or batch vs streaming-based. Moreover, MEVA can introduce unwanted latency.

Encryption:

The basic idea of private/encrypted mempools is that users submit encrypted transactions, and block producers commit to these transactions before decrypting them. Encryption doesn't inherently provide a mechanism for ordering transactions, but it can be used as a method to ensure the pre-execution privacy of transactions so that the sequencer's ability to extract MEV through transaction reordering or censorship is restricted. Sequencers receive and commit to these transactions in their encrypted form. The decryption process, which reveals the actual transaction data, can occur at a later stage when transaction validity is being verified. One of the major challenges to utilizing encryption is that it creates a vector for DOS attacks. Shutter’s approach to solving this issue implements the requirement for the user to include a signed envelope within the encrypted transaction that covers fees (instead of specifying a gas limit) at the time of its inclusion within a block (not execution). While metadata leakage becomes a vector for MEV here, it’s possible ZK proofs could play a role in preventing that issue. Different methods of encryption offer different drawbacks/are in different stages of development but a few are summarized below:

- Secure Enclaves:

Using trusted hardware technologies like Intel's SGX or AMD's SEV, sequencers work on transactions within a secure, isolated space. To prevent MEV extraction, additional rules (like deterministic ordering) need to be enforced within the enclave.

- Threshold Encryption:

A group of block producers jointly decrypt encrypted transactions for execution after ordering, distributing the decryption responsibility to limit potential MEV extraction. In the Ferveo protocol, for example, a group of validators produces a shared public key, dividing its corresponding private key into individual shares. Transactions are encrypted with this public key, and require a threshold of validators pooling their private key shares in order to be decrypted. After decryption, transactions are executed strictly as per the committed order, preventing the insertion of MEV-inducing transactions. However, it's critical to note that the complexity of key generation, sharing, and management escalates considerably among validators in threshold encryption schemes. This makes achieving high levels of decentralization (comparable to Ethereum mainnet) practically unattainable, limiting such schemes to between 100-999 validators.

- Timelock Encryption:

Users send transactions in a time-locked "safe" to the sequencer. The sequencer includes these locked transactions in the block, and they are only "unlocked" (decrypted and executed) after a certain time or condition is met, keeping the transaction details hidden from the sequencer until execution.

Fair Ordering:

Fair ordering protocols/frameworks that implement specified ordering policies can be used in tandem with decentralized sequencer networks, such as that of Themis or Aequitas. Chainlink Fair Sequencing Service is another example. However

“fair” ordering as a concept is hotly debated. A recent talk on fair ordering you may have seen recently. Even more on fair ordering.

Note: We could write an entire piece of equal length and density exclusively on transaction ordering + MEV broadly. A lot of this is already covered in discourse for L1s, so we’ve intentionally kept the above section high-level for sake of brevity. We encourage you to dive into this rabbit hole, separately! A fantastic resource: Frontier Research. We discuss MEV more later in the context of rollups and implications for cross-domain MEV, but keep scope limited.

Leader Election

The other key competency of sequencers–that becomes increasingly important for a decentralized sequencer set–is choosing who has the right to propose each block to the network.

This is an important distinction because in a world in which proposer and builder roles are unbundled, sequencers/proposers can still create blocks that are partially pre-ordered by external builder parties, and still include their own ordered set of transactions.

A few different mechanisms for leader election:

- Proof-of-Authority (PoA):

The role of the sequencer could be assigned using a simple round-robin algorithm, especially suitable for a permissioned set of sequencers given the trust assumptions that sequencers act fairly. This is not to say that round-robin constitutes PoA. Rather, it's one way to distribute the sequencing responsibility among a trusted set of entities in a PoA-based rollup system.

- MEV Auction (MEVA):

The auction between who builds

a block (which we already covered) and who proposes

a block can be distinct (leader election). Whereas MEVA for block building can be modularized to other parties, sequencers remain concerned with rights to propose the next block, hence using MEVA. Penalties would still need to be introduced to reduce the risk of bad actors. Proceeds of the auction could be distributed, burned, or go towards some treasury. The limitation of MEVA is that there are still plenty of vectors for centralization risk, such as collusion between sequencers during the auction process. Generally, MEVA in both instances of transaction ordering and leader election could look rather similar.

- Proof-of-Stake (PoS) for Leader Election:

You can use PoS for leader election, where it's permissionless to join as a sequencer but you must stake the native rollup token. Increased stake improves your chances of being selected as leader (centralization vector). Since the sequencers are now staked, they can be held accountable through slashing penalties. In this case, the rollup would still use the L1 for consensus.

Proof-of-Authority (PoA):

The role of the sequencer could be assigned using a simple round-robin algorithm, especially suitable for a permissioned set of sequencers given the trust assumptions that sequencers act fairly. This is not to say that round-robin constitutes PoA. Rather, it's one way to distribute the sequencing responsibility among a trusted set of entities in a PoA-based rollup system.

MEV Auction (MEVA):

The auction between who builds

a block (which we already covered) and who proposes

a block can be distinct (leader election). Whereas MEVA for block building can be modularized to other parties, sequencers remain concerned with rights to propose the next block, hence using MEVA. Penalties would still need to be introduced to reduce the risk of bad actors. Proceeds of the auction could be distributed, burned, or go towards some treasury. The limitation of MEVA is that there are still plenty of vectors for centralization risk, such as collusion between sequencers during

the auction process. Generally, MEVA in both instances of transaction ordering and leader election could look rather similar.

Proof-of-Stake (PoS) for Leader Election:

You can use PoS for leader election, where it's permissionless to join as a sequencer but you must stake the native rollup token. Increased stake improves your chances of being selected as leader (centralization vector). Since the sequencers are now staked, they can be held accountable through slashing penalties. In this case, the rollup would still use the L1 for consensus.

While most mechanisms implemented lead to the election of a single leader/proposer, protocols likeMultiplicity allow for Multiple Concurrent Block Proposers (MCBP). In this setup, validators each create a special bundle of transactions, sign it, and send it to the leader, who must include at least 2/3 of these stake-weighted bundles in the proposed block for it to be valid. This prevents the leader from excluding transactions present in more than 1/3 of these bundles, which helps mitigate the potential for rent extraction and censorship resistance that occur in granting single leaders/sequencers/proposers monopolies on transaction inclusion.

3. Sequencer produces pre-confirmation

When users send transactions to the sequencer, there's a time gap between when the sequencer receives the transaction and when the eventual batch is posted to the L1. In the meantime, the sequencer gives a pre-confirmation

:

"I promise I will order and eventually publish this transaction to the L1 as expected."

This is referred to as "soft finality." The period of time that exists between soft finality and actual finality depends on several factors, perhaps the most notable being whether the rollup uses fraud proofs or validity proofs.

In the case of a centralized sequencer, pre-confirmations are pretty simple because transactions are finalized under the assumption that the single, trusted sequencer won't act maliciously. In a decentralized paradigm, this gets more tricky.

L2 Consensus

Rollups can also incorporate their own consensus mechanisms, which strengthens the finality guarantees sequencers make by ensuring the inclusion of transactions in blocks prior to the final consensus on the DA layer.

Centralized sequencers provide soft finality rapidly, often in under a second, due to their role as sole block producers. However, the strength of the pre-confirmations hinges on the accountability mechanisms in place, as dishonest actions by sequencers can weaken the guarantees of soft finality.

The addition of L2 consensus adds a layer of accountability (often economic), as sequencers are penalized if they fail to include a transaction in a block after providing a pre-confirmation. L2 consensus can be implemented through various traditional consensus mechanisms such as PoS leader election or BFT consensus, each carrying their own trade-offs in terms of sequencer overhead and implementation time. However, it's a hot topic for discussion whether L2s need

local consensus (especially if it's PoS-based), potentially introducing more vectors for validator collusion and decreased efficiency. Colluding validators could charge inflated fees, censor transactions, or reap other economic benefits. It's also possible that if L2 validators capture more value, there may be less incentive over time for L1 validators, compromising overall network security.

One option to address concerns with PoS-based L2 consensus, in particular, is to leverage restaking, which is what Espresso plans to do withEigenLayer. Restaking in this context would help to further decentralize the sequencer (reducing susceptibility to DOS attacks) and improve shared security with the L1.

Next Generation Consensus

Next generation consensus mechanisms are being introduced to address the aforementioned issues, we've included a few proposals by different teams below:

- HotShot:

HotShot is a consensus protocol introduced by Espresso, based on the HotStuff protocol. HotShot seeks to offer high throughput and fast finality for decentralized sequencer networks.

- Proof of Efficiency:

Proof of Efficiency is a consensus mechanism introduced by Polygon Hermez meant specifically for ZK rollups. It involves a two-step model that splits activities between a sequencer and an aggregator.

- Sequencer: aggregates L2 transactions into batches and posts them to the L1
- Aggregator: submit validity proofs on L1 for batches posted by the sequencer

HotShot:

HotShot is a consensus protocol introduced by Espresso, based on the HotStuff protocol. HotShot seeks to offer high throughput and fast finality for decentralized sequencer networks.

Proof of Efficiency:

Proof of Efficiency is a consensus mechanism introduced by Polygon Hermez meant specifically for ZK rollups. It involves a two-step model that splits activities between a sequencer and an aggregator.

- Sequencer: aggregates L2 transactions into batches and posts them to the L1
- Aggregator: submit validity proofs on L1 for batches posted by the sequencer

4. Sequencer posts to DA layer + Rollup node posts updated state to L1 bridge contract

In the case of optimistic rollups, the sequencer posts transaction data to the DA layer, and this transaction data is used by the specific rollup nodes known as "proposers" (not to be confused with block proposers) to compute the updated state root. Then, the updated state root is submitted to the rollup's enshrined bridge contract, along with an associated bond. If a proposer makes an incorrect attestation to state updates, it's detected by another type of full nodes, known as "verifiers." Verifiers execute rollup state transitions and, if necessary, can slash the bonds of proposers through a fraud proof game.

In the case of ZK rollups, the sequencer is usually responsible for passing on the block of ordered transactions to a prover, which then generates the validity proof and passes it to a rollup full node–along with the associated full transaction data or state diffs. Data sent by rollup full nodes to the availability layer can be complete transaction data, offering high transparency and quick finality but at high resource cost, or state diffs, which are space-efficient and cost-effective but sacrifice transparency and delay finality. The choice between these options hinges on the specific needs and constraints of the ZK rollup implementation.

Finality

Much of how

and when

finality is determined depends on consensus design as described above and

whether the rollup uses fraud proofs or validity proofs.

As soon as the rollup posts transaction data on the data availability layer, rollup nodes can compute the new state root of the rollup and finality is achieved within the realm of the rollup. For the L1 however, which hosts the rollup bridge smart contract, finality is achieved once this state root has been verified through a validity proof (ZKR) or after the fraud proof window (ORU) has passed.

One mental model, offered by 0xrainandcoffee, for the finality levels of rollups is categorizing them into 4 different stages,

depending on the rollup. These are:

1. Pending_On_L2: Soft commitment given by the rollup's sequencer(s) that the users' transactions will eventually be committed and finalized on the underlying layer, of which it derives security.
2. Finality_On_L2: The sequencer(s) has committed to the state transition function of the rollup, and the block(s) have been added to the canonical chain of the rollup
3. Pending_On_L1: Transaction input or output/state transition function has been posted to the L1, but the dispute period or validity proof has yet to be posted, or period ended – this requires two successive epochs to have passed for Ethereum. This is the point where most Optimistic rollups say finality has been reached, however according to the canonical bridge – there's still, at this point, an arbitrary 7-day challenge period.
4. Finality_On_L1: The dispute period has ended for Optimistic rollups, or the validity proof has been posted and verified, as well as confirmed in two successive epochs with a supermajority.

Pending_On_L2: Soft commitment given by the rollup's sequencer(s) that the users' transactions will eventually be committed and finalized on the underlying layer, of which it derives security.

Finality_On_L2: The sequencer(s) has committed to the state transition function of the rollup, and the block(s) have been added to the canonical chain of the rollup

Pending_On_L1: Transaction input or output/state transition function has been posted to the L1, but the dispute period or validity proof has yet to be posted, or period ended – this requires two successive epochs to have passed for Ethereum. This is the point where most Optimistic rollups say finality has been reached, however according to the canonical bridge – there's still, at this point, an arbitrary 7-day challenge period.

Finality_On_L1: The dispute period has ended for Optimistic rollups, or the validity proof has been posted and verified, as well as confirmed in two successive epochs with a supermajority.

In the context of "sovereign rollups," the sequencer's role is primarily to order transactions within each block, which are then posted to the DA layer responsible for coming to consensus on the order of blocks. Sovereign rollup full nodes execute transactions to ensure their validity, but inherit the re-org and censorship resistance as well as liveness guarantees from their DA layer. As for light clients in a sovereign rollup, they rely on validity or fraud proofs generated by full nodes. While the sovereign rollup inherits the security of the DA layer for natively issued assets on the rollup, there is currently no way to safely transfer assets from said DA layer to the sovereign rollup like with smart contract rollups. However, we know work is being done by teams to address this exact issue.

PART II: SHARED SEQUENCERS

Now, we generally understand how a rollup transaction currently works and the nuances in the process. Perhaps, most importantly, we can differentiate between the tasks of a sequencer to include 1) building blocks 2) proposing blocks and understand why these roles should be unbundled/sequencer sets decentralized.

Let's now evaluate new models for how sequencing could work.

Shared Sequencers

Rollups can look to shared sequencers for decentralization, rather than designing their own methods for decentralizing their own sequencer set. A shared sequencer is a network that aggregates and orders transactions across multiple rollups.

Leveraging a shared sequencer across multiple rollups enables easier and stronger decentralization, which gives the general benefits of a highly decentralized sequencer set including liveness, censorship resistance, economic security, and stronger pre-confirmations. Shared sequencers, in particular, also expand the design space around rollup interoperability and MEV.

Astria is one of the first teams to build a shared sequencer network, Metro. A key property of how Metro works is that it separates transaction ordering from execution. The sequencer is responsible for ordering

transactions, but does not execute them.

Isolating the function of the sequencer to ordering without having to run a full node materially reduces overhead, allowing for higher sequencer network decentralization (higher censorship resistance). Once transaction ordering has been agreed to, execution (and proving) can later be deferred to an entirely different party, which doesn't need to rely on censorship resistance.

After the sequencer orders the transactions and publishes them to the DA layer, now "lazy" rollup full nodes can simply download the transactions, optionally apply a fork choice rule to select a subset of transactions, perform any arbitrary transaction processing, and apply those transactions to the state. Headers can then be generated and gossiped to light clients. Lazy rollups by default share a mempool with the sequencer set. Given shared sequencers don't check for invalid state transitions, lazy rollups must be able to deal with invalid transactions.

Also notably, rollups are able to fork from the shared sequencer at any time because they still maintain sovereignty (transaction data is retrievable from the DA layer).

Espresso is another team working on a shared sequencer network. The core principles of the Espresso sequencer are similar to Metro, with architectural differences. It's worth noting that the concept of "lazy" blockchains was popularized by Celestia (closely tied to Astria), uniquely informing Astria's architecture. The Espresso sequencer is secured by EigenLayer (restaking), is built on HotShot consensus protocol (mentioned above), and Espresso also provides data availability (Astria uses Celestia as the DA layer).

Another noteworthy shared sequencer project is Radius. Instead of decentralizing the sequencer set that orders transactions, Radius employs self-limiting cryptography on a single centralized sequencer. In their approach, users create time-lock puzzles and send them along with their encrypted transactions. These time-lock puzzles delay the sequencer's ability to discover a symmetric key, but once found, the key is used to decrypt the users' transactions. To mitigate the risk of invalid transactions causing a DOS attack on the sequencer, Radius incorporates ZK proofs as attachments to the encrypted transactions, offering a means to verify their validity without relying on a signed envelope within the encrypted transaction, as done in the Shutter-style method mentioned earlier.

Cross-Domain MEV + Atomicity

Shared sequencers also have interesting implications for cross-domain MEV and atomicity.

As more rollups join the same sequencer network, more cross-domain MEV becomes extractable. There are growing concerns that rollup interoperability and cross-domain MEV introduce additional centralization vectors. Sequencers could collude to control cross-domain MEV. Validators who participate in multiple PoS consensus protocols for independent domains have greater access to and potential control of cross-domain MEV opportunities.

In the case of cross-chain atomicity, cross-chain messaging is traditionally done asynchronously, but shared sequencers expand this design space. A classic example of where cross-chain atomicity is relevant is atomic arbitrage. Cross-chain atomicity ensures that the trades are executed conditionally/simultaneously. However, you will still never have the same guarantee that you would have if you were transacting entirely on L1.

It's also worth noting further emphasizing that [the shared sequencer ensures atomic inclusion

, but not execution

](https://prestwich.substack.com/p/the-definitive-guide-to-sequencing?sd=pf). This is because sequencers can't make any guarantees over the fallibility of transactions. If one leg of the intended atomic cross-rollup operation fails, the sequencer has no way to control that as their responsibility is to order, not execute. The implications of this detail mean that simple atomic arb can be executed atomically, but transactions like swaps can't be. However, using a shared sequencer combined with additional infrastructure like timelocks and cross-chain messaging protocols for rollup nodes to verify the state of the other rollups could still enable wider cross-chain capabilities.

In a multi-rollup environment, the potential for extracting MEV atomically diminishes as the scope for probabilistic MEV capture increases. This shift occurs due to the fragmentation of dapps across multiple domains and the inherent lack of cross-chain atomicity across these environments.

This landscape presents unique MEV opportunities; however, they come with higher barriers to capturing them. The absence of cross-chain flash loans increases capital costs, and the lack of atomicity across environments means more uncertainty surrounding the successful execution of all components of an MEV strategy. Consequently, searchers capable of handling additional inventory risk are well positioned, as well as searchers equipped with specialized algorithms and infrastructure adept at managing probabilistic MEV.

All of these factors point to the fact that those with more capital are better positioned to capture opportunities in this cross-domain world, and that the resulting environment could lead to increased block builder centralization.

This potential future underpins the motivations and benefits of developing solutions like SUAVE that provide stronger guarantees around cross-chain atomicity amongst other more direct benefits to users.

PART III: ORDER ROUTING - RETHINKING THE TRANSACTION LIFECYCLE

Now that we better understand the transaction lifecycle, we can dive into how proposals like SUAVE and Anoma and high-level concepts like preferences and intents seek to improve how we think about transactions and order routing.

SUAVE

A key point we wanted to drive home in earlier sections was the distinction between the following roles: building, proposing, executing. SUAVE is a proposal for a decentralized block building market. SUAVE is often called a “sequencing chain,” but this can be a little confusing because it’s specifically focused on block building.

SUAVE is an independent network that functionally serves as an aggregated mempool and decentralized block builder across multiple domains. Users express preferences by signing a transaction indicating a particular goal (you can think of preferences analogously to “intents” or “meta-transactions” in other frameworks—the user signs a preference to indicate a particular goal ie: I want to transfer ETH).

Following this, “executor parties” (block builders) engage in a competitive auction to have their bundles included. During this process, the potential MEV subsidy to the builder begins as negative and increases over time, all while builders running SGX brute force different orderings of preferences until a winning executor finds a point at which, if the bundle is mined, the total arb value > marginal cost of the MEV subsidy–leading to optimal execution and MEV minimization for users.

You can think of SUAVE as a shared builder, and a shared sequencer as a shared proposer.

Once an executor has won the auction, they could create a proof that the bundle of transactions follows certain validity rules and send it to a shared sequencer network where it can be permissionlessly and privately aggregated with other bundles from other rollups into batches, which are finally posted to each respective rollup’s DA layer.

While SUAVE doesn’t require the existence of shared sequencers to work, its architecture can be synergistic

with shared sequencer networks. SUAVE could also work with centralized and/or decentralized sequencers that are more vertically integrated into the rollup as well.

While SUAVE improves upon trust assumptions present in MEV Boost in the sense that bundle proofs are used instead of a trusted third party (TTP) like a relay, it’s important to note that once the bundles have been confirmed by the destination rollup, an oracle (which introduces trust assumptions!) sends updates to SUAVE that the state transition has occurred.

SGX, a TEE produced by Intel, introduces another trust assumption as its enclave has been compromised in the past. While it serves as a production-ready option for executing computations on encrypted data, it isn't the only option. A new wave of trustless cryptographic solutions, which fall under the umbrella of homomorphic cryptography, offers alternatives. Justin Drake has an excellent talk on existing and future solutions to this challenge, which are summarized in the following slide.

Anoma

Anoma is another protocol proposing an alternative framework for preference expressions that is very different from how we think about transactions today–designed fundamentally differently from existing smart contract platforms like Ethereum.

Ethereum today offers programmable settlement, where the chain is primarily occupied with settling state transitions. Settlement is sufficient if the parties involved know 1) who they are, 2) what they want, and 3) with whom to coordinate.

However most transactions don't fulfill these requirements, they are multi-party and need counterparty discovery–helping counterparties discover each other, coordinate with each other, and agree what to settle.

Today, applications handle counterparty discovery themselves, largely through the following approaches:

- putting everything onchain–expensive! (ie: DEX)
- having a centralized server offchain–additional trust assumptions! (ie: centralized sequencer, OpenSea)
- having an app-specific sovereign chain–development complexity! lack of composability! (ie: DyDx)

putting everything onchain–expensive! (ie: DEX)

having a centralized server offchain–additional trust assumptions! (ie: centralized sequencer, OpenSea)

having an app-specific sovereign chain–development complexity! lack of composability! (ie: DyDx)

More details on the above presented by Adrian Brink from Anoma

Anoma presents an architecture that is intent-centric, allowing users to express what they want,

not how to get it

(we'll cover this distinction in a later section).

In Anoma, users sign binding intents (you can think of intents analogously to "preferences" or "meta-transactions" in other frameworks—the user signs a preference to indicate a particular goal ie: I want to transfer ETH) and then submit it to intent gossip node(s) that form pools of intents (could be a global pool or more specified). "Solvers" (analogous to "executors") monitoring the pool of intents will then aggregate compatible intents into a state transition that fulfills all the intents of the users and submit the transaction to any full node.

It's important to emphasize that Anoma is not inherently a blockchain, but an architecture that can be implemented by any chain. Each instance of Anoma is referred to as a "fractal instance." For example, the first fractal instance of Anoma is Namada, a Layer 1 PoS chain.

Anoma and SUAVE provide interesting alternative frameworks for how we think about transactions, converging on the general concept of intents.

Intents, Preferences, Meta-transactions

We've skated conveniently over intents and preferences so far, so let's unpack what they mean.

Justin Drake said recently on a podcast about MEV, "Users generate intents, they don't generate transactions."

While this is true conceptually, we're still far from an implementation system that goes from intent to transaction formation to optimal execution efficiently.

Today, when a user signs a transaction, they authorize a function call which, depending on the state of the chain at time of execution, can have radically different outcomes. This is very counterintuitive to how most people would expect transactions to work: a user indicates their goal, and then the transaction achieves said goal.

In a truly intents-centric world, a user declares/signs their preferences, and how that desired state is achieved is left to a network of third-party actors, (solvers/executors), to execute these preferences on their behalf. Going a step further, what Justin goes on to point out is that in an ideal world, intents can be aggregated and fulfilled in a single, optimized master transaction that executes all users' intents simultaneously by leveraging coincidences of wants across each users' intent.

The best infrastructure to express, communicate, and execute intents should minimize MEV, maximize censorship resistance, and ideally be optimized for cross-domain interactions.

Moreover, they should consider the extent to which the burden is placed on the user to communicate their intent more granularly, which might overcomplicate UX–a decision that will fundamentally affect the architecture of said intents protocol.

Alternative transaction types have emerged in discussion to address the limitations of today's architecture, but they also introduce additional trust assumptions around economic censorship. You still have a lack of system-level guarantees that the solver/bundler is not excluding a particular set of transactions from the mempool (re-introducing a commonly discussed issue with transactions today).

In ERC4337, for example, introducing offchain infrastructure like alternative mempools means that bundlers (which act as solvers for user preferences) are needed to communicate with onchain smart contracts. Consequently, we need to make sure that these bundlers don't selectively censor intents they don't want to broadcast. There are a couple of solutions to this problem, and if you're thinking, "wait, don't Anoma/SUAVE solve this," that's absolutely right–SUAVE does so through the *onchain* auction between executors/block builders.

More generally, having these agents (bundlers, executors etc…) come to consensus over what the list of valid user intents are, is one of the main ways we can create censorship resistance (also inclusion lists!). While the implementation details of the type of consensus required here are TBD, we know that this is what blockchains are good for! On the other hand, this involves communication overhead that introduces latency and reduces throughput. Another potential solution to this model could be plugging into an optimistic model, but this introduces the necessity for bonds on the part of users and solvers, decreasing capital efficiency and complicating UX.

One possible solution: build the logic into smart contracts.

Then, we wouldn't necessarily have to specify an alternative transaction type for users to optimize their preferences. Unfortunately, this is when theory doesn't meet practicality–the difficulties of onchain scalability make it harder to express more creative, complex logic in smart contracts (there are folks working on this, Soon!). Long-term, provable guarantees of offchain computation with cryptographic guarantees through ZK could be the solution to intent expression. ZK has the potential to improve the speed and cost of executing users' desires and the quality of their execution, without creating an alternative transaction type and the baggage that comes with it.

What this doesn't address is interoperability. In a multichain world, approaches reliant on app-specific integrations could lead to insufficient UX abstractions.

Systems like SUAVE and Anoma that instead enable a network of permissionless solvers running full nodes across domains to execute these preferences could lead to more optimal execution because these solvers can incorporate more information *faster* about the state of other chains without needing to communicate information to users or integrating with the latest DEX on BSC where liquidity for Mongoose Coin was just seeded.

iykyk

Interestingly, we've actually seen intents implemented before–just in a more narrow context. Just as intents "outsource" how the desired state is achieved to a network of competing third-party actors, (solvers/executors), RFQ broadcasts for trades get routed to solvers. Doug Colkitt's explanation elegantly conveys the "lookback option" problem in solver-based RFQ systems that exist today, providing interesting lessons for designing generalizing intent systems moving forward. What's important to note is that the obvious implications of RFQ literature apply to the context of trades, but in general intents can be used to specify any desired end state not just for a trade.

The TL;DR here is that it's fair to say that as an industry, we haven't converged on the optimal architecture for users to express intents—whether it's ERC4337 (meta-transactions), Anoma (intents), or SUAVE (preferences)—nor have we determined where in the stack the orchestration of these intents will be housed.

One hypothesis: wallets—the interfaces where users sign transactions—are best placed to enable users to specify their intents and to allow for programmatic expression of user needs and conditions within transactions, helping to sidestep the principal-agent issue. However, in the short term, there exists a temptation for some wallets to bypass builders/searchers competing to redirect MEV back to users, choosing instead to extract MEV themselves through the bundles they control. While some wallets will probably do this, it's a strategy that overlooks long-term viability. Regulatory developments will eventually extend the principle of best execution from traditional finance into crypto markets, likely making this self-serving MEV extraction on the part of wallets illegal. Beyond regulatory concerns, it's probably still a -EV strategy for wallets

considering the transparency of onchain MEV extraction and the ease by which users can migrate their keys to competing alternatives. With these realities in mind, leading wallets, as the conduit for users' intents, will strategically uphold their fiduciary duties regarding best execution for their users. There are several ways they can do this: by routing user intents directly to neutral platforms like SUAVE's mempools; indirectly through their own or a trusted external ERC4337 bundler; or via other forms of Multi-Party Computation (MPC) or Account Abstraction (AA) that serve the same purpose without necessitating an ERC4337 bundler.

Conclusion

Design space in crypto is an ever-expanding plane. We are constantly pushing the boundaries for performance, expression, and decentralization. The way these boundaries manifest is through blockspace.

How do we build blocks in a way that is most performant, aligned with users' goals, and resistant to centralized parties and censorship? These are questions that have been asked since Day 1 of the Ethereum network.

Emergent architectures around sequencers ask us to reconsider these questions in a rollup-centric world. On L2s, we're moving past the pre-PBS era, where centralized sequencers currently act as both the proposer and builder.

Systems like SUAVE and Anoma push us on even more questions, like how blocks should be constructed in a privacy-preserving, cross-domain world and how we should think about transactions at their most abstract level of meaning.

Today, how we think these decisions manifest, is that sequencers–whether centralized, decentralized, or shared–will either plug into alternative mempools designed to enable alternative transaction types that increase the scope of what preferences users can express–or keep chugging along what they're doing will innovations in scalability solutions take care of expanding the expressibility of smart contracts and user preferences.

However, the questions we've posited are complex and the community is discussing them in public forums day in, and day out. We're excited to be there with you.

Thank you to @0xJim, @Hactar0, @jillrgunter, @_danielmarzec, @PossibltyResult,

@willkantaros, and @mountainwaterpi for thoughtful feedback on drafts of this post.

Our work builds upon incredible research from others, and we cannot spotlight them enough:

- Jon Charbonneau, dba
- 0xrainandcoffee, Maven11
- Alex Beckett, Celestia
- James Prestwich
- Patrick McCorry, Arbitrum
- Flashbots
- Espresso
- Celestia
- Anoma

Jon Charbonneau, dba

0xrainandcoffee, Maven11

Alex Beckett, Celestia

James Prestwich

Patrick McCorry, Arbitrum

Flashbots

Espresso

Celestia

Anoma

and countless other sources.

Disclaimer:

This post is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. You should consult your own advisers as to legal, business, tax, and other related matters concerning any investment or legal matters. Certain information contained in here has been obtained from third-party sources, including from portfolio companies of funds managed by Archetype. This post reflects the current opinions of the authors and is not made on behalf of Archetype or its affiliates and does not necessarily reflect the opinions of Archetype, its affiliates or individuals associated with Archetype. The opinions reflected herein are subject to change without being updated.