Imagine a world where rollups out of the box could achieve high levels of censorship resistance, ease of deployment, interoperability, fast

finality, liveness, and MEV democratization. This may seem like a lofty goal, but with the arrival of shared sequencers, such a world may be within reach. However, not all rollups are created equal, which leads us to questions about how rewards and MEV should be distributed on shared sequencer networks. In this article, we will explore the properties that make shared sequencer networks achievable and what attributes can be achieved.

Shared Sequencer Networks has primarily been covered by Alex Beckett, and later more in-depth by Evan Forbes from Celestia and the Espresso Systems team (as well as Radius), alongside the new incredible pieces by Jon Charbonneau. Josh, Jordan, and their team at Astria are building the first Shared Sequencer Network in production. Astria's Shared Sequencer Network is a modular blockchain that aggregates and orders transactions for rollups, without performing the execution of said transactions.

In the Astria setup, sequencers send the ordered blocks to the DA layer, and also to the rollup nodes. Rollups get a soft finality guarantee from the sequencers, and the hard finality guarantee from the DA layer (after the block is finalized), whereafter they will execute transactions valid to them.

A shared sequencer network is essentially a rollup-agnostic set of sequencers that, as the name implies, can serve a cluster of different rollups. This has various trade-offs and attributes, which we'll get into later. First off, we must describe the properties that are most important for a sequencer (or set of sequencers) to uphold. In a rollup, the sequencer or sequencer set has a primary requirement of censorship resistance or liveness (some of which is inherited from the base layer, alongside security). This means that a valid transaction submitted to the sequencer must be included in the chain within a finite amount of time (timeout parameter). The shared sequencer set only needs to ensure that transactions are included in a block (i.e. crLists).

Achieving both censorship resistance and liveness is quite difficult, as outlined in our Modular MEV part 2. In consensus algorithms like Tendermint, you're ensured recovery after an attack. However, you lose out on liveness in the advent of an attack. Essentially, requiring all other sequencers to sign off on a block, instead of picking a bespoke leader, might not be the best option. While this increases censorship resistance, it's at the cost of "centralization", and MEV extraction to a singular leader. Another sequencing mechanic that could be used, could be something akin to Multiplicity from Duality. Which is their gadget for non-leader nodes (or sequencers) to include additional transactions into a block. Overall, post-attack censorship resistance and liveness is difficult to achieve in most consensus protocols.

Another possible consensus algorithm, that could be used, could be something like HotStuff 2, which ensures liveness during attacks.

What it allows for is to avoid having to wait for the maximal network delay (timeout) before picking a new leader, in case of censorship or non-signing. The reason it might be an interesting consensus algorithm for decentralized sequencer sets is that it addresses the issue of liveness in consensus without adding an extra phase. If a leader knows the highest lock (the highest number of actors that agree on a specific output) and can convince honest parties, the issue is solved. If not, honest leaders after a specific point can drive progress responsively, aiding the next leader. For example, Hotstuff nodes do not need to confirm the switching message before notifying a new leader, instead it can directly switch to the new view and notify the new leader.

Where it differs from Tendermint is that despite both being two phases (Hotstuff I had three, II has two), Tendermint has linear communication, but is not responsive, which Hotshot2 is. If a sequence of honest leaders exists, the protocol is optimistically responsive

because all steps (except the first leader's proposal) depend on obtaining a quorum of messages from the previous step. In a shared sequencer setting, this makes it possible for the protocol to achieve better liveness without needing to fall back onto the underlying layer, without removing the possibility to do so.

The building blocks of a shared sequencer set

A set of sequencers is allowed to submit transactions to the settlement layer (which the rollups inhabit). You can join this set provided they meet certain requirements, and that the number of wanted proposers isn't reached. This could be to optimize latency, throughput and so on – e.g. Tendermint stays relatively low). These requirements will be similar to how you'd become a validator on a certain blockchain. For instance, you have certain hardware requirements that must be met, as well as a starting stake, or bond – this is especially important if you want to provide economical soft finality.

A shared sequencer set (or really any decentralized sequencer set) consists of several components that work together to ensure that transactions are processed. These components include:

- JSON-RPCs for each rollup that transaction submission (for non-full-node-runners) to node(s) that function as mempool(s) before building and ordering. Within the mempool, there needs to be some mechanism to decide on queue, as well as transaction selection processes to ensure efficient block construction.
- A block/batch builder algorithm, which processes the transactions in the queue and turns them into a block or batch. This step may also include optional compression to reduce the size of the resulting block (calldata compression). This

should, as noted earlier, be separated from the proposer – essentially PBS. The compression of data, can come in the form of several ways, such as:

- No RLP encoding—However, this would likely be needed for a decentralized sequencer set, to help standardise the transfer of data between nodes in a space-efficient format.
- Omitting the nonce (unique number that validates data in a certain block)—it can be recomputed when required for execution by looking at the prior state of the chain.
- Gas Price Simplification—set gas based on a fixed range of prices.
- Gas Simplification—beyond just gas prices, also the gas numbering system
- Replace address with an index—Instead of storing the full address, rollups can store an index for a mapping of addresses stored elsewhere.
- Value in Scientific Notation—The value field in an Ethereum transaction is priced in wei, making it a large number. You cannot omit the value field or reduce it to a fixed set of values. However, you can write it in scientific notation instead of wei to save on data.
- Omit Data Field—Not needed for simple transfers, however on more complex transactions it would be needed.
- Replace Individual Signature with BLS Aggregate Signature - The signature is the largest component of an Ethereum transaction. Instead of storing each signature, you can store a BLS aggregate signature for a certain number of transacitons. A BLS aggregate signature can be checked against the entire set of messages and senders to ensure validity.
- From Field as index - As with the address field, you can add the from field as an index to a mapping.
- One interesting notion of “modular” design is that you can also play around with this and make trade-offs where you see fit, to make it fit your specific rollup’s case.

- No RLP encoding—However, this would likely be needed for a decentralized sequencer set, to help standardise the transfer of data between nodes in a space-efficient format.
- Omitting the nonce (unique number that validates data in a certain block)—it can be recomputed when required for execution by looking at the prior state of the chain.
- Gas Price Simplification—set gas based on a fixed range of prices.
- Gas Simplification—beyond just gas prices, also the gas numbering system
- Replace address with an index—Instead of storing the full address, rollups can store an index for a mapping of addresses stored elsewhere.
- Value in Scientific Notation—The value field in an Ethereum transaction is priced in wei, making it a large number. You cannot omit the value field or reduce it to a fixed set of values. However, you can write it in scientific notation instead of wei to save on data.
- Omit Data Field—Not needed for simple transfers, however on more complex transactions it would be needed.
- Replace Individual Signature with BLS Aggregate Signature - The signature is the largest component of an Ethereum transaction. Instead of storing each signature, you can store a BLS aggregate signature for a certain number of transacitons. A BLS aggregate signature can be checked against the entire set of messages and senders to ensure validity.
- From Field as index - As with the address field, you can add the from field as an index to a mapping.
- One interesting notion of “modular” design is that you can also play around with this and make trade-offs where you see fit, to make it fit your specific rollup’s case.

- A peer-to-peer (p2p) layer, which allows the sequencer to receive transactions from other sequencers and gossip blocks after building. This step is crucial for ensuring that the shared sequencer can work effectively across multiple rollups.

JSON-RPCs for each rollup that transaction submission (for non-full-node-runners) to node(s) that function as mempool(s) before building and ordering. Within the mempool, there needs to be some mechanism to decide on queue, as well as transaction selection processes to ensure efficient block construction.

A block/batch builder algorithm, which processes the transactions in the queue and turns them into a block or batch. This step may also include optional compression to reduce the size of the resulting block (calldata compression). This should, as noted earlier, be separated from the proposer – essentially PBS. The compression of data, can come in the form of several

ways, such as:

1. No RLP encoding—However, this would likely be needed for a decentralized sequencer set, to help standardise the transfer of data between nodes in a space-efficient format.
2. Omitting the nonce (unique number that validates data in a certain block)—it can be recomputed when required for execution by looking at the prior state of the chain.
3. Gas Price Simplification—set gas based on a fixed range of prices.
4. Gas Simplification—beyond just gas prices, also the gas numbering system
5. Replace address with an index—Instead of storing the full address, rollups can store an index for a mapping of addresses stored elsewhere.
6. Value in Scientific Notation—The value field in an Ethereum transaction is priced in wei, making it a large number. You cannot omit the value field or reduce it to a fixed set of values. However, you can write it in scientific notation instead of wei to save on data.
7. Omit Data Field—Not needed for simple transfers, however on more complex transactions it would be needed.
8. Replace Individual Signature with BLS Aggregate Signature - The signature is the largest component of an Ethereum transaction. Instead of storing each signature, you can store a BLS aggregate signature for a certain number of transacitons. A BLS aggregate signature can be checked against the entire set of messages and senders to ensure validity.
9. From Field as index - As with the address field, you can add the from field as an index to a mapping.
10. One interesting notion of “modular” design is that you can also play around with this and make trade-offs where you see fit, to make it fit your specific rollup’s case.

No RLP encoding—However, this would likely be needed for a decentralized sequencer set, to help standardise the transfer of data between nodes in a space-efficient format.

Omitting the nonce (unique number that validates data in a certain block)—it can be recomputed when required for execution by looking at the prior state of the chain.

Gas Price Simplification—set gas based on a fixed range of prices.

Gas Simplification—beyond just gas prices, also the gas numbering system

Replace address with an index—Instead of storing the full address, rollups can store an index for a mapping of addresses stored elsewhere.

Value in Scientific Notation—The value field in an Ethereum transaction is priced in wei, making it a large number. You cannot omit the value field or reduce it to a fixed set of values. However, you can write it in scientific notation instead of wei to save on data.

Omit Data Field—Not needed for simple transfers, however on more complex transactions it would be needed.

Replace Individual Signature with BLS Aggregate Signature - The signature is the largest component of an Ethereum transaction. Instead of storing each signature, you can store a BLS aggregate signature for a certain number of transacitons. A BLS aggregate signature can be checked against the entire set of messages and senders to ensure validity.

From Field as index - As with the address field, you can add the from field as an index to a mapping.

One interesting notion of “modular” design is that you can also play around with this and make trade-offs where you see fit, to make it fit your specific rollup’s case.

A peer-to-peer (p2p) layer, which allows the sequencer to receive transactions from other sequencers and gossip blocks after building. This step is crucial for ensuring that the shared sequencer can work effectively across multiple rollups.

Peer-to-Peer (p2p) layer

- Leader rotation algorithm for the shared sequencer set (they don’t need to reach consensus, in the case of single leader election). You could choose to just have a leader rotation algorithm in place. Another route is to go down the multi-concurrent block producer road, as proposed by Duality.
- A consensus algorithm, for example the earlier discussed Tendermint or Hotstuff2, to ensure that rollup nodes are in agreement with the proposed ordering of the ledger.
- An RPC client for block/batch submission to the underlying DA+consensus layer, which ensures that the block/batch is

securely added to the DA layer, which ensures “final” finality and that all transaction data is available on-chain.

- Separation of builder and proposer roles to ensure fairness and consistency, and to avoid MEV stealing. Alongside removing the execution from the ordering, something that’s important to optimize efficiency, lower PGA and heighten CR.

Leader rotation algorithm for the shared sequencer set (they don’t need to reach consensus, in the case of single leader election). You could choose to just have a leader rotation algorithm in place. Another route is to go down the multi-concurrent block producer road, as proposed by Duality.

A consensus algorithm, for example the earlier discussed Tendermint or Hotstuff2, to ensure that rollup nodes are in agreement with the proposed ordering of the ledger.

An RPC client for block/batch submission to the underlying DA+consensus layer, which ensures that the block/batch is securely added to the DA layer, which ensures “final” finality and that all transaction data is available on-chain.

Separation of builder and proposer roles to ensure fairness and consistency, and to avoid MEV stealing. Alongside removing the execution from the ordering, something that’s important to optimize efficiency, lower PGA and heighten CR.

Rollup nodes receive ordered blocks from sequencers for soft commit & receive ordered DA layer blocks for hard commit

Transaction data (calldata) is first posted onto a base network, where consensus is performed to guarantee transactions for both users and rollups. Then, the rollup nodes execute the transactions (and commit to a state transition function that is added to the canonical rollup chain). The shared sequencer network provides rollups with their liveness and censorship resistance. Rollups maintain their sovereignty because all transaction data is stored in the base layer, allowing them to fork away from the shared sequencer at any time. The state root from the state transition function (STF) on the rollup side is calculated from the transaction roots (inputs) sent to the DA layer from the shared sequencers. On Celestia, this is generated as data is added to the chain and reaches consensus. As you already have the tx root (and all available data), Celestia can provide a small inclusion proof for light clients (that rollup nodes run on Celestia).

To provide users with the UX they have come to expect, the rollup nodes receive the ordered block (that is also sent to the DA layer). This can provide the rollup with a soft finality guarantee—a promise that the block will eventually be ordered as is on the DA layer, which is when the rollup nodes execute the transactions and provide a new state root.

Block Creation and Slots

To determine the timing for block creation, sequencers need to establish a concept of slots. Sequencers should submit batches at regular intervals, typically every X seconds, where X is the slot time. This ensures that transactions are processed in a timely and efficient manner, since otherwise the leader for a specific slot would be timed out, and lose its signing reward (and execution rewards). For example, Celestia’s block time (according to GitHub specs) is likely going to be around 15 seconds. Because this is known, we can make some assumptions about how many “slots/blocks” we could potentially fit into a finalized block from the shared sequencing set to the DA layer and rollup nodes. Considering optimized Tendermint, or something like Hotstuff2.

Multiple batches can be submitted in a single slot, assuming the design allows the rollup full nodes to efficiently process them into a single block (within the slot’s time and timeout parameters). This can help to further optimize block creation and ensure that transactions are processed quickly. Additionally, slots can also be used to facilitate sequencer leader elections. For example, you could pick a slot leader randomly from among the stake pool, based on stake weight. Doing this in a way that preserves secrecy, makes the most sense, and utilizing something like Secret Leader Election would probably be preferred, to minimize censorship. Or even Distributed Validator Technology (DVT) type setups (such as Obol/SSV-esque solutions). Latency and slot times have considerable effects on how block submission and building affect the protocol. It, therefore, makes sense to look at how this can impact systems. Bloxroute especially has some great research and data points into how their relays have performed on Ethereum. In MEV-Boost the participating proposers (validators, or in a rollup case, sequencers) request from the relays a GetHeader. This provides them with the block bid, which is the value of a specific block. This is likely to be the highest value block received every time. For each slot, validators usually ask for a GetHeader around 400ms after slot time starts – and since there are obviously numerous validators – relays often have to serve countless requests. This would also be the case for a large shared sequencer set. This means that you need proper infrastructure in place to facilitate this.

Relays also help in facilitating builder and proposer separation, while also working to verify that blocks built by builders are correct. They also check if fees are paid correctly as well and act as DoS protection. Additionally, they essentially act as escrowers of blocks and also handle validator registrations. This is particularly important in a world of unbounded sequencers, where you need proper accounting for who is participating, and who’s not (e.g. the syncing layer discussed previously).

In terms of when blocks are ready (as in submitted by the builders), they are usually presented around 200ms before, or after slot start. Although, just like with GetHeader, there’s a considerable variance, but primarily they get served around 200ms before/after slot time starts. In the case where builders are sending to several relays, there’s actually quite a bit of latency. Bloxroute also looked at what happens when blocks are sent to several relays. As you might expect, the more relays blocks are getting propagated to, the more latency. On average, it took a second relay 99ms for the block to be

available, another 122ms for the third and yet another 342ms for the fourth relay. Source:

As might have been clear over the last couple of months, RPCs are incredibly important to blockchains. Not having the proper infrastructure in place is a massive burden, and having a proper selection of RPCs is also vital. In that vein, RPC inclusion is, therefore, crucial for retail users who send their transactions to RPCs (and the public mempool). Bloxroute made a small test of 20 transactions which were sent to various RPCs, and then measured how long it took each transaction to be included in a block.

Source: Bloxroute Labs

What was interesting to see is that some RPCs took several blocks to include the transaction, since it was dependent on what builder won the respective next block. The more builders that transactions are sent to from the RPC, the higher likelihood of a fast inclusion. Although there's the chance that transaction originators could utilize their uniquely positioned order flow to target a specific builder or even build their own blocks.

On the statistics front in regard to relay performance on Ethereum, it is interesting to look how they've performed as well. This helps us provide a more in-depth understanding of how PBS works in a multi validator/builder/relay world, which is hopefully where we are heading concerning rollups in general. Metrika has some great stats on this, and all credit goes to them for these data points.

What should be noted here is that a missed bid is just when a relay is expected to relay a bid, but does not. Expectations of bids come from validators that are registered to specific relay for any given slot. This is not a fault in itself for a relay, and isn't treated as such on a protocol level, either.

Source: app.metrika.co

If a bid fault were to happen (e.g. relays providing invalid blocks), and it was responsible, then it would count as a bid fault. These are usually not that frequent, and are for the most part registration preference faults (which is when gas limit or fees don't meet the specific validator's registration preferences). More rare are consensus layer faults, which is when a bid doesn't align with consensus layer rules of Ethereum, such as timestamps being wrong or parent hashes not aligned with previous blocks etc.

In terms of latency (e.g. the time it takes a validator to receive a response for a block header built by a builder) the data is quite consistent. Although there are some outliers, for example if a relay that is being requested is in a different geographical location than say the chosen validator.

Source: app.metrika.co

In terms of builders, the total builders over time on MEV-boost are around 84, of which the top 3 builders account for around 65%~ of built blocks. This is somewhat misleading, since these are also the longest running builders. Although it does showcase a somewhat centralized builder role still, as if you lower the timescale, we get similar results. In terms of actual active builders, the numbers are much lower, 35 in the past 30 days and 24 in the past week. The competition is tough, and the biggest and baddest usually win out. Something that exclusive order flow is likely and already only adds to. We expect the distribution of builders to remain somewhat centralized (due to it being a game of having the best order flow and hardware optimizations) around a few parties unless we get significant changes to the setup. While not a fundamental issue, this remains a centralizing force in the stack, and we are keen to learn about ideas on challenging the existing state of affairs here. We highly recommend reading Quintus' piece on order flow, auctions, and centralisation

if you're interested in digging deeper into this (severe) topic.

In terms of where we expect to see the builder role moving forward in the modular stack, we are quite certain that (at least in Cosmos SDK setups) are going to see Skip/Mekatek type setups via Builder Modules. Another solution is SUAVE type setups—e.g. a specific global

builder chain that services any arbitrary number of chains with block building and bid preference services to ensure PBS. We'll cover this solution more in-depth later, and present some answers to some of the open questions around this.

On relays, we highly recommend a piece byAnkit Chiplunkar of Frontier Research and Mike Neuder of the Ethereum Foundation. The name of which is Optimistic relays and where to find them

. Which goes into great detail of how relays in MEV-boost operate, their current tradeoffs and cost of running among possible changes that can be made to increase efficiency. An interesting note is that by Flashbot estimations, currently running a relay in MEV-Boost costs around ~$100k/year.

Finality

As a precursor to how we view finality concerning modular blockchains (as they look now), here is a refresher from our previous Modular MEV piece. Do note that this is not an "official" nor comprehensive view of finality; however, we feel that it most accurately represents the nuances of rollup finality in an easy to understand mental model.

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 validity proof has yet to be posted, or dispute period has yet to end – this also 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.

Now on sovereign shared sequencing rollups, this looks slightly different, let's try to explain it with a diagram:

In this situation, we in theory get finality on the L1 before the L2, wait what? Yes, the L2 in this case is sovereign after all. This is assuming no fraud proof & challenge period, or that you're using a validity proof.

So, how do we actually achieve these levels of finality

? Block finality is achieved when a block is added to the canonical chain and cannot be reverted. However, there's some nuance here depending on the view of a full or light node. In the case of an ordered block, it is final once it has been included in a DA layer block. An executed block (w/ state root) is executed by rollup full nodes/validators which gives them guarantees of a valid state root derived from the ordered blocks on the baselayer. Finality beyond full nodes (e.g. for light clients or bridging) have to be convinced of the validity of said state root. Here, you can utilize one of the methods described below. Furthermore, another method is also having validators be held accountable for the correct attestation to a state root (the optimistic route), via a bond and subsequent fraud proof. You could also supply a validity (ZK) proof.

There are different ways to achieve block finality:

- Probabilistically, through proof-of-work (PoW), LMD+Ghost, Goldfish, Ouroboros (PoS) etc.
- Provably, when the block is signed by enough committee members. (e.g. Tendermint 2/3rds, Hotshot2 or other PBFT types)
- Depend on the ordering of transactions/blocks on the DA layer so that it rules what is the canonical chain, and fork choice rule (E.g. Based).

Probabilistically, through proof-of-work (PoW), LMD+Ghost, Goldfish, Ouroboros (PoS) etc.

Provably, when the block is signed by enough committee members. (e.g. Tendermint 2/3rds, Hotshot2 or other PBFT types)

Depend on the ordering of transactions/blocks on the DA layer so that it rules what is the canonical chain, and fork choice rule (E.g. Based).

There are different types of finality that can be achieved through different mechanisms.

One type of finality is "soft finality" (e.g. pending) which can be achieved if a single leader election is implemented. In this case, every slot will only have one or zero blocks (commit or not), and the syncing layer can safely assume the ordering of the transactions in these blocks.

Another type of finality is "provable finality" which provides stronger guarantees (essentially final) than soft finality. To achieve provable finality, a majority of sequencers must sign off on a block, thereby agreeing that this block is canonical. While this approach is nice to have, it may not be necessary if single leader election is already implemented, as it essentially guarantees block ordering. Obviously, this depends on the specific leader election algorithm in effect. E.g. is it a 51% implementation, a 66% or single leader (preferably random (VRF) and secret). If you want to read further on finality in Ethereum specifically, we highly recommend this piece, alongside one we'll recommend for unbounded sets later on.

Permissioned, Semi-Permissionless or Permissionless

To prevent potential DoS attacks, it's essential to have economic barriers in place to both join the sequencer set and submit transactions to the sequencer layer. In both bounded (limited number of sequencers) and unbounded (unlimited) sets, there must be an economic barrier to submitting batches to the DA layer to prevent the syncing layer (that propagates blocks between sequencers) from being slowed down or DDoSed. However, the DA layer itself provides some protection, as there's

a cost involved in submitting data to it (da_fee). The bond required to join an unbounded set should cover any additional cost necessary to protect against the syncing layer being spammed. On the other hand, the bond required to join a bounded set will depend on the demand (equilibrium from a cost/revenue perspective).

With an unbounded sequencer set, provable finality on the sequencer layer is not possible (since we're never quite aware of how many active voters/signers that are there). On the other hand, with a bounded sequencer set, provable finality can be achieved through a (super)majority of sequencers signing off on a block. This does require the syncing layer to be aware of the sequencer layer and which sequencers are active at any given time, which is some added overhead. In a bounded sequencer set (e.g. max 100), you're also able to optimize the number of sequencer for "performance", although at the cost of decentralization and censorship resistance. The importance of a bounded set, and an economical bond, is the ability to provide "fast" provable finality, which is also deterministic.

Both the unbounded and bounded sequencer types are something we also see in legacy blockchains, such as with PoS in Ethereum (Casper+LMD-GHOST), which is unbounded; PoS in Cosmos SDK/Tendermint based chains that utilize a bounded set. One interesting thought is that, do we expect similar economics and choices

from the community surrounding shared sequencers, as we've seen in Proof-of-Stake? Here we've seen a centralization towards a number of entities (hence unbounded doesn't really matter, if you have a few large staking providers/pools anyway). Although, they do "mask" the centralization, and after all, you can still home-stake, if you wanted to. From an ideological perspective, the choice should almost always be unbounded – but bear in mind that the economics make them very similar regardless. Regardless of who the participants are, the economics of what you're paying for should still be equal, e.g. cost of DA and hardware cost (although this might be lowered by the amount of stake allocated to you and experience, alongside running infrastructure efficiently already). Even in the bounded PoS world, we've seen a group of infrastructure providers becomes essentially the largest and most often seen validators across almost all chains. The correlation is massive between validators across most Cosmos chains already, which is also certainly a danger to the decentralization and censorship resistance of said chains. Although, one thing that is quite different is the fact that any retail user can stake any amount, to any validator of their choosing. Sadly, this is typically allocated to the top of the list, and life goes on. Yet again, we ask; do we expect similar economics to take place in the modular word? One would hope not, but as we specialize, you often want the best person for the job – which more often than not is expertise staking providers. We will also cover these economics later on, in a separate section.

However, one important thing to remember in all of this, is that at the end of the day, what matters most is end-user verification – which with light clients and DAS, is available to anyone, anywhere (even at the Pyramids of Giza);

Source: @JosephALChami - Celestia Light Node

The trade-offs and positives of unbounded and bounded in sequencer terms, can be summarized to:

Unbounded Sequencer Set:

- Anyone with a sufficient bond/stake can become a sequencer = high level of decentralization
- No single leader election is possible, as the set is potentially infinite.
- Non-single leader election via VRF is possible, but difficult to determine VRF parameters without knowing how many sequencers there will be. This should also, if possible, be a secret leader election implementation to avoid DoS attacks.
- If no leader election = wasted resources problem: block building is essentially a free-for-all, and whoever submits the first valid block/batch wins while everyone else loses.
- No provable finality on the sequencer layer, only probabilistic – e.g. LMD Ghost+Casper
- Finality can only be reached once the batches are written to the DA layer (limited to underlying layers block times, 15 seconds in Celestia's case).
- "Better" censorship resistance than a bounded set.

Anyone with a sufficient bond/stake can become a sequencer = high level of decentralization

No single leader election is possible, as the set is potentially infinite.

Non-single leader election via VRF is possible, but difficult to determine VRF parameters without knowing how many sequencers there will be. This should also, if possible, be a secret leader election implementation to avoid DoS attacks.

If no leader election = wasted resources problem: block building is essentially a free-for-all, and whoever submits the first valid block/batch wins while everyone else loses.

No provable finality on the sequencer layer, only probabilistic – e.g. LMD Ghost+Casper

Finality can only be reached once the batches are written to the DA layer (limited to underlying layers block times, 15 seconds in Celestia's case).

“Better” censorship resistance than a bounded set.

Bounded Sequencer Set:

This is, for example, one of the solutions for single slot finality in Ethereum, alongside having super “majority” committees. See this.

- The number of permitted sequencers is limited at any specific time.
- A bounded set is potentially more complicated to implement than an unbounded set.
- Single leader election can be implemented, giving strong finality guarantees on the sequencer layer.
- The syncing layer needs to be aware of the sequencer set to know which blocks are valid.
- Writing the sequencer set (or set changes) into settlement layer blocks (e.g. fork-choice rules), which are written to the DA layer, can allow the syncing layer to determine the sequencer set independently. For example, this is how Sovereign Labs’ rollups function, the set changes are written into the validity proofs that are posted to the DA layer.
- Strong finality guarantees on the sequencer layer may not be necessary if the DA layer's speed is sufficient (However, most current setups that aren’t optimized settlement layers have at least 10+ second block times).

The number of permitted sequencers is limited at any specific time.

A bounded set is potentially more complicated to implement than an unbounded set.

Single leader election can be implemented, giving strong finality guarantees on the sequencer layer.

The syncing layer needs to be aware of the sequencer set to know which blocks are valid.

Writing the sequencer set (or set changes) into settlement layer blocks (e.g. fork-choice rules), which are written to the DA layer, can allow the syncing layer to determine the sequencer set independently. For example, this is how Sovereign Labs’ rollups function, the set changes are written into the validity proofs that are posted to the DA layer.

Strong finality guarantees on the sequencer layer may not be necessary if the DA layer's speed is sufficient (However, most current setups that aren’t optimized settlement layers have at least 10+ second block times).

Regarding how these sequencer sets are monitored, and new entrants are added, or removed, there’s still quite a large design space for how this should be implemented. E.g. is this going to happen via token holder governance (and how do you then account for the many different tokens and rollups that are utilizing the set?). This means that it’s likely to be done via off-chain social consensus that signals on-chain changes (e.g. the Ethereum way). However, do keep in mind that the actual on-chain consensus is obviously in place regarding slashing for breaking consensus rules.

Economics of Shared Sequencers

The economics of a shared sequencer network allow for some interesting options. As we discussed before, a validator in a shared sequencer network isn’t too different from your typical L1 validator. The network it participates in is just more optimized for doing one task, receiving intents (pre-PBS) and consequently proposing and ordering transactions. Like a “regular” validator, there is a revenue and cost component. On both sides of the equation, the network the validator participates in has a lot of flexibility, similar to regular L1.

Revenue is sourced from users, or the rollups they ultimately aim to interact with, who pay a certain fee for usage of the shared sequencer. This fee could be a percentage of MEV extracted (the input number might be hard to approximate here though), cross-chain value transferred, a gas like construct or a flat fee per interaction. The most elegant revenue solution is probably a setup where the value paid to the shared sequencer is less than the additional value gained by sharing a sequencer amongst rollups and getting the benefits of shared security and liquidity. The downside is that the benefits of decentralization of another part of the stack are hard to quantify. However, as a shared sequencer network grows into its own ecosystem, its ability to extract fees will probably increase. This is in a large part due to their inherent ability to easily aggregate, and it has somewhat an economics of scale effect. As more and more rollups, and consequently applications, join the network; the more cross-domain MEV is extractable.

On the cost side, the shared sequencer network also has optionality to compete. They could easily subsidize usage of their network by fronting the cost of posting to the DA layer, or even for interacting with apps on the rollups itself. This would be similar to strategies used by web2.0 companies, whereas you take an initial loss on user (or rollup) acquisition in the hopes their long-term revenue will outweigh the expense. Another more novel, or crypto native, approach on this is allowing rollups to pay for DA fees in their native token. Here, the shared sequencer layer takes on the pricing risk between the token needed to post data on the DA layer and the native token of the rollup. In essence, it is still a shared sequencer fronting the cost; however, it creates ecosystem alignment by acquiring a token of a ‘supplier’ (the rollup) instead. This is somewhat similar to the warehousing construct we set out in the appchain thesis. Other parts where a cost reduction is possible is by utilizing different forms of DA. Different DA layers will offer different pricing due to utilization, the ability for users to easily verify through light clients or outright make different blocksize choices. Finally, the shared sequencer can also batch

transactions before posting to the DA layer. In the case of ZKR this could allow for lowering tx costs due to a certain equilibrium of transactions, on the ORU side you could do various batching gas optimisations that we currently see live on various rollups. This would decrease the amount of data needed to be posted on the DA layer, thus lowering the cost of the shared sequencer network, increasing the profitability of the overall network. It does come at the cost of limiting interoperability and changing the time to finality (in the sense of finality on L1 as discussed earlier).

Overall, the economics of a shared sequencer network allow for some interesting experimentation and bootstrapping strategies. We estimate that the key differentiation will be the size of the ecosystem and hence the amount of cross-domain MEV and less so the cost side of things. We also highly recommend checking out the Espresso team's very in-depth blog posts on shared sequencing, they also cover some of the economics trade-offs (and positives) of these types of networks as well. To showcase another way for why rollups are incentivized to utilize shared sequencing (beyond the economics of it), it makes sense to look at aggregation.

Aggregation Theory and Shared Sequencers

Another way to describe the properties that shared sequencers enable is through the lens of aggregation theory. Aggregation theory (AT) is the concept of how platforms or aggregators can gain significant user traction through a systematic way of integrating other platforms and their users. You essentially change the game from distribution of a scarce resource (e.g. blockspace) to controlling the demand for abundant resources (again, blockspace in this example makes sense). AT is essentially aggregating the suppliers and products (i.e. rollups and blockspace) into a single superior

user experience for an aggregated user base. This relationship becomes increasingly exclusive as the network effects of these aggregators grow – there's no reason to move away. As this happens, the user experience is what becomes the vital differentiating factor between similar setups. If the incentives for new users are there (e.g. good UX and easy interoperability), then the likelihood of a rollup moving to their own network or a different set is unlikely – since network effects drive new suppliers and new users. This creates a flywheel effect, both from a supplier and user perspective, but also from an aggregated censorship resistance view.

Source: Aggregation Theory 2015, Ben Thompson

In the vein of shared sequencers, AT can be seen through the almost "combination" and alliance of rollups that all utilize similar vertical parts of the stack – strengthening themselves and others, while enabling users to have the same experience, anywhere.

Suppliers (e.g. rollups) aren't exclusive in theory in shared sequencer sets, however in reality; the shared sequencer set, it's rollups and users benefit from a cycle of network effects that lead to increased usage of said rollups. These benefits make it easier for rollups and users to integrate with the shared stack, since they would lose out more by not participating. While these benefits may be hard to see when you only have two rollup sharing a sequencer set, it becomes a lot more clear as you add increasing numbers of rollups and users to the equation. The shared sequencer set has a direct relationship to users, as they order their transactions, even if the users themselves have no idea they're even interacting with them – since from their perspective, they're just using a rollup that they have reason to interact with (meaning the ordering/sequencers becomes exclusive). The only cost associated with these sequencers is essentially the cost of the hardware that they're run on, as long as the blockspace and token that secures it is valuable to the end-users. The transaction fees are digital and paid from the users' wallets, mayhap in the future, even abstracted away with advances such as paymasters within account abstraction (however, someone will have to bear the cost of DA, sequencing and execution).

This makes even more sense when you consider where Josh and Jordan from Astria formerly worked – Google. Google products have since inception been heavily inspired by the ideas of AT, this is especially prevalent with Google Search, which was created by modularizing (lol) individual pages and articles, making them directly accessible via a global search window.

The customers in this case of the shared sequencer set (users of rollups) have an increasingly low acquisition cost, since they are likely to be attracted to the set as the number of suppliers (rollups) increase. This does mean that, in most cases, aggregators (or multi-aggregators) have a likely winner-take-all effect, since the value of this aggregator increases with the rise of suppliers (as long as the user experience is good, of course). In contrast, on singular sequencing networks, the customer acquisition is limited to a single network and its applications. If a user wants to utilize a rollup application that's on a different rollup, they will (within current limitations) have to move off the network entirely. This means that the stickiness of users and value isn't very high, and also means that at any moment, if a different rollup ecosystem becomes highly regarded (or has more incentivization), capital could take flight.

Summarization of Attributes and Trade-offs

Attributes

A Shared Sequencer set is a rollup network which aggregates and orders transactions for multiple rollups. These rollups all share the same sequencers. This pooling of resources means that rollups receive stronger economic security and censorship resistance, which allows for fast

soft finality guarantees, and conditional cross-rollup transactions.

Now, there's been quite a lot of noise on Twitter surrounding atomicity between rollups that are sharing the same sequencer set. This is primarily set around the fact whether it is atomic by default, or not – it is not. However, if the rollups in question have implemented each other's state transition function (STF) as a dependency between them regarding conditional transactions – they could indeed have atomicity between them – as long as their slots/block times line up (which they should with a shared seq. set). In this case, to get atomic interoperability, you would essentially just run a light node of chain A on chain B and vice verse (similarly to how IBC works). To further this interoperability in terms of security measures (beyond trusting a single full node as a light node), you could implement ZKPs (state proofs, essentially) to solve the "oracle issue" of making sure state is indeed correct. This would give more clarity surrounding if conditional txs or similar have touched the canonical bridge between them. Fraud proofs is also a possibility, but would obviously leave us with a challenge period (which would mean third parties would pop up to take that risk for a fee). Furthermore, in the light client (and not full node on each other's case) it would be lagging behind at least a block, as a result of waiting for the signed header + fraud proof window (if any).

We're of the belief that "bridging" is most likely to be solved in conjunction with light clients and ZK. The challenge of using light clients in this case (and not smart contracts) is that hard-forks on the rollup node sides (upgrades etc) would need to be done in conjunction with each other to keep their bridge running (just how IBC needs the same state module enabled). If you'd like to read more on this specific topic (and how to solve it) – we highly recommend this presentation.

What makes shared sequencers incredible scalable is the fact that they're not executing and storing any state (as centralized sequencers do now). The same goes for the rollup nodes themselves (they don't have to scale to 100s of nodes, unless they want an atomicity between them – e.g. light clients/state proofs). These nodes just execute transactions that are valid to their rollup, as well as any conditional cross-domain transactions that are valid to them as well. If the rollup nodes had to execute and store state for many, many rollups, it would hinder scalability and lower decentralization (in turn lowering censorship resistance). It also reinforces the concept of Proposer-Builder-Seperation (PBS). Although we still need to separate the builder and proposer completely. In the current setup, the sequencer is essentially a builder and proposer (although they do not execute the transactions). The ideal setup is likely that sequencers just exist to blindly sign off on built blocks coming from a highly optimized builder setup and making sure blocks are correctly implemented (while providing a high degree of economic finality and censorship resistance to that attestation). Through that, they can provide a high degree of security and commitment to guarantee soft finality to rollup nodes.

For cross-rollup conditional transactions, they also exist to help enable rollup nodes (the executors) to provide intermediate state roots, allowing for atomicity between rollup. If you want a deep dive on how this could possibly look, then this brief presentation is perfect for you.

Tradeoffs

The timeout parameter that was mentioned earlier has some interesting effects on MEV and transaction inclusion, depending on the ordering and leader/consensus mechanism of the sequencer set. e.g. if the timeout parameter, as described in our application-specific chain thesis, is relatively short, it is key for proposers on the decentralized sequencer level to post data as quickly as possible. In such a world, you could get competition of "validators" of the decentralized sequencer set competing to act as a leader and outbidding each other for blockspace on the DA layer up to a point at which it is no longer economically desirable to do so.

As was covered by Evan in theoriginal lazy sequencers post on the Celestia forums, waiting to execute transactions until they're posted to a base layer (Celestia in this case) is quite wasteful. Since you're now limited to the base layer's blocktimes - which is a long time to wait for finality from a UX perspective. To get better UX, the shared sequencers provide a soft finality commitment to rollups (as discussed earlier) which gives us the UX that users are used to in existing centralized rollups (while staying decentralized and with high censorship resistance). The soft commitments are in essence just a promise of the final order of transactions, but backed by a heavy economic bond and fast

final finality from the baselayer once posted. This is also covered by fraud proofs (as noted in the introduction). The actual hard finality comes when all tx data has been posted to the baselayer (meaning the L1 actually reaches faster finality). This is dependent on whether or not the rollups are utilizing fraud proofs, or zero-knowledge proofs for their sovereign proof verification – which happens on the rollup side. The reason for wanting this separation is to take away the heavy computation of state transitions from the sequencers (which is a huge bottleneck). Instead, the rollup nodes only handled the ones valid to them (which does mean we have to add conditional transactions, state proofs or light node verification for proper interoperability). Hard finality is still dependent on the base layer (but this will likely be around 15s with Celestia, and also deterministic w/ Tendermint) – which does provide us a high degree of relatively fast hard finality guarantees on the rollup side.

It is also possible to utilize ZK proofs within the network to optimize validation, transaction size (e.g. only posting state diffs – but this does add a higher degree of trust until a ZKP is posted). These state proofs can, as covered earlier, be used to allow for connected chains/rollups to have much easier and faster interoperability (not having to wait for a challenge window).

One downside to these conditional transactions that were covered earlier is that they're likely to be a lot pricier to verify and post (e.g. Tendermint block header verification is costly, and is subsidized on Cosmos chains) – as well as add some latency to the system (but still much faster that siloed rollups). The atomicity that is achievable because of the vertical shared integrations do make up for a lot of this.

In the bootstrapping phase of a new rollup, going with a shared sequencer set makes a lot of sense – and the positives that

you've gained as a supplier will likely outweigh some of the trade-offs that you might be "forced" to make on a moat level. However, for already mature rollups with a lot of transaction and economic activity – giving up part of your moat likely makes little sense.

This brings into question whether we would need something like an economic/transaction (per rollup) weighted redistribution of extracted MEV to entice already mature rollups to join a shared set – or even keep extremely mature rollups and avoid having them spin out their own network. This is all quite theoretical, but it's certainly an interesting thought process surrounding how MEV would look in a shared vertical world between many rollups that have varying degrees of activity. For example, if a singular rollup that drives much of the value through the sequencer set is sharing parts of these profits with others (that might not be bringing in as much "value"), there's certainly more reason for them to move to their own siloed system. Sreeram from EigenLayr has some thoughts here too, which we recommend reading as well.

This also becomes increasingly important when you consider the fact that searchers have a considerable technical cost to work on a new chain, so standardizing this and providing some sovereignty to chains regarding "their" MEV is probably a good starting point. Essentially, in MEV the dominant interface (or software) is likely to win out – however actually monetizing this software is incredibly difficult, unless you run key parts of the infrastructure (leading to centralization). On a market level, what shared sequencers provide is essentially a common mempool for many suppliers, with a centralized auction that would likely lead to more healthy competition.

A worry here is that in the case of two rollups both running a sequencer in the shared set. A rollup (A) with "less economic" value running a sequencer could be elected to propose blocks with a high amount of MEV + fees originating from a rollup (B). From rollup B's perspective, they would essentially be missing out on some of the value, that they in a siloed approach would keep to themselves.

Solving Interoperability Trade-offs

Another note on the proposed tradeoffs made regarding interoperability, another way to solve some of these issues, is summarized below:

The point of the shared sequencer network is that it gives a guarantee of canonicity for multiple chains, which is certainly a big advantage in this case. This can be combined with a mechanism to give a guarantee over a valid state transition between rollups. This could be a committee-based approach (e.g. PoS), a bonded attestation (Optimistic approach), or the one that we prefer – a ZKP backed by a committee signing. Because the shared sequencers are "lazy" they only create mega blocks which order transactions for multiple rollups, and the execution of said transactions is left to the specific rollups. State proofs (i.e Lagrange, Axiom, or Herodotus and others) are all possible solutions to potentially get attestation to finality across sovereign rollups as well. You could even add economic bond finality attestations via things like staking pools, EigenLayr and more. The idea is that the shared sequencer gives an economic guarantee of the canonicity of ordering, and generating a validity proof from this ordering is as such deterministic. The basic idea is that rollups can synchronously execute transactions across each other. For example, two rollup node networks can be conditionally aware that the two rollup histories are valid via ZKPs and available (with the data published to an efficient DA layer). The rollup nodes can then settle both rollups simultaneously by posting on-chain a single-rollup block prefix received from both network A and B. One of the things that does need to be said, and we covered this earlier slightly – is that cross-rollup atomic (or synchronous) transactions are likely to be pricier than singular intra rollup transactions. This is because conditionally cross rollup transactions are consuming resources from two independent systems via shared execution.

Succinct also has a write-up on cross-chain "atomic" transactions between rollups with a shared sequencer (and shared fraud prover) within the Optimism superchain ecosystem that can be checked out here. Also, as Bo Du from Polymer puts it; "Cross chain atomic txs are like acquiring a lock across database shards for a write".

A Vertical Future

Jon Charbonneau and others have already gone quite in-depth into the inner workings of how a SUAVE like chain would possibly look, so we won't go too into detail. You can check out his pieces if you want a more detailed description. Although, we do feel that vertical integration does deserve its own part, both to highlight how incredibly modular we can really go (and why) and some of the open questions and concerns surrounding vertical integration.

While the current shared sequencing proposals from both Astria, Espresso and Radius are incredibly modular, the sequencers are still acting as both builders and proposer (although in the Astria case, they do not execute transactions). Astria is also actively building PBS into their architecture from the get-go.

There are several ways to achieve PBS if it's not already built into the protocol (although at varying degrees of decentralization). Something like SUAVE, using an off-chain model such as MEV-Boost, or implementing builder modules (such as the Cosmos SDK ones being built by Mekatek and Skip).

What's important to note is that neither of these are exclusive. You could flexibly utilize several different methods, and have anyone express their preferences. Whereby you'd have executors compete to fill those. Adding more optionality is always good (and adheres to our beliefs in modularity). Although, various implementations will have different trade-offs. With something like SUAVE, you do add privacy (with SGX or crypography) alongside cryptoeconomic security to the truth, instead of relying on a centralized PBS builder that is trusted completely. (Thanks to Jon Charbonneau for feedback here)

A vertical integration into a builder chain would need to be done in a few that ensures fairness, and that doesn't cut corners and increase latency and lowers performance. As such, a builder chain would need to be incredibly optimized, and likely require costly and performant hardware (leading to centralization). This means that to get end-user verification, we would likely need some implementation of light nodes (although they have to trust full nodes), or utilize a state proof type setup to ensure that chain and users have proof of bid preference being filled, and that blocks have been built correctly.

A chain such as this would likely be incredible state heavy (which we would like to avoid). Although these state heavy transactions will be preference bids via smart contracts. In the case of preference bids, it either gets filled or doesn't (in a short amount of time), since bids are usually only valid for a short period of time depending on the preferences. This means that we would likely be able to implement very efficient (and early) state expiry for bids – which would allow us to prune data and keep the chain "clean". This expiry date needs to be long enough to still allow bid preference to get filled, although lowering it too much essentially makes it impossible to implement blockspace futures far into the future. The need to revive and retrieve expired bid contracts is unlikely to be needed, since they aren't required to exist for infinity (unlike applications) – this can be made more "secure" and trustless by providing state/storage proofs as bids are filled, or via DAS storage solutions – such as the one proposed by Joachim Neu

As we covered early, the need to verify the "truth" of SUAVE is likely going to be limited to "sharks" (power users) of the platform, since most users and customers of a SUAVE implementation all have significant economic gain to be derived from utilizing it. This could entice us to just have people run a full node, if they want to verify – although this excludes the vast majority of people (you could argue they have no need to verify). This is (in our opinion) antithetical to crypto, and we would preferably see verification of SUAVE happen "trustlessly" via state proofs or via a light client friendly implementation.

The reason for needing this is that you'd want to verify that your bid preference were filled correctly, and that blocks are filled with correct information upon payout (to avoid weird rebundling and other exploits). This is essentially an oracle problem – which indeed can be solved for on-chain state (which all of SUAVE would be) via state proofs. Making these state proof cross-chain brings forward another problem though, how do we relay this information cross-chain in a way that makes it so nothing was tampered with or withheld? This could likely be via a strong economic finality attestation (such as the one presented by Lagrange, wherein you could use the re-staking validators of EigenLayr to attest to finality and truth of the chain with a very strong economic bond). This then brings forward a different problem (e.g. the bid contract specifies that the "oracle" – in this case restakers) have specified a token that is staked and provides the economic bond – but how do we then slash this outside of consensus? While you could code in slashing criteria, this isn't in consensus and would mean social slashing would be utilized (which is hardly ever "fair", and could lead to problems) via smart contracts. This is one of the bigger issues with slashing in EigenLayr at the current time.

So, where does this leave us? Likely in the situation that until we get on-chain "trustless" slashing outside of consensus, a SUAVE like chain would likely need its own consensus algorithm and cryptoeconomic security that attest to the finality of bid preferences and built blocks – however, that means adding more cryptoeconomic attack vectors, especially if its building blocks for rollups worth much, much more that its own cryptoeconomic security.

Beyond this, there's a super large design space for how a SUAVE-type chain and cross-domain MEV should look in general. Underneath are just some of the possible avenues to research:

- Intent Matching and Intent-Based Systems
- Convex optimizations in multi asset trades
- DSLs
- MEV redistribution
- Latency wars
- Scaling issues of having a single set of actors needing to build for the state machines of multiple rollups.
- Preference expression

Intent Matching and Intent-Based Systems

Convex optimizations in multi asset trades

DSLs

MEV redistribution

Latency wars

Scaling issues of having a single set of actors needing to build for the state machines of multiple rollups.

Preference expression

On preference expression, to interact with a smart contract in the EVM, a contract call (message) is sent to a specific function on an address with deployed code containing execution instructions. While users provide inputs, they may not have

control over outputs due to potential statefulness.

In contrast, preference expression design systems (e.g. SUAVE and Anoma) only require users to sign preferences with a bond, which is paid to the builders and proposers if the preferences of the searchers are met. The implementation of different language and virtual machines may vary for complex combinatorial logic, such as transaction ordering for MEV searchers and builders. This is a new design space that is getting a lot of attention lately – especially surrounding the Anoma structure. We would recommend looking at the Anoma architecture here. And would highly recommend this short read as well, as an appetizer from Breck at Haun.

Much of this article wouldn't be possible without the great research that has been done in this particular vertical. Many thanks to:

- Evan Forbes (Celestia)
- Alex Beckett (Celestia)
- The Astria Team
- The Espresso Team
- Flashbots
- Jon Charbonneau and Michael Jordan from dba (lower case research + a ton of other sources)
- Bloxroute Labs

Evan Forbes (Celestia)

Alex Beckett (Celestia)

The Astria Team

The Espresso Team

Flashbots

Jon Charbonneau and Michael Jordan from dba (lower case research + a ton of other sources)

Bloxroute Labs

Furthermore, we want to thank all the people, projects, and companies we reference throughout this article for their contributions to the space