Rollups Aren’t Real

Originally posted to Substack.

Thank you to Ben Fisch, Josh Bowen, Cem Özer, Kelvin Fichter, Zaki Manian, Hasu, Robert Miller, Tarun Chitra, Sreeram Kannan, and many others for their input and great discussions around the topic.

Introduction

Kelvin thinks that ZK rollups aren't real, but I don't think that any "rollups" are real, at least not yet. So, how do we make them real rollups?

Today's rollups are mostly trusted and permissioned:

I'll provide an overview of the landscape for:

- Forced Transaction Inclusion Mechanisms

- Even if the rollup operator is censoring users, they should be able to force inclusion of their transactions to preserve censorship resistance.

- L2 Sequencer Decentralization and (Optionally) Consensus
- Single sequencers, PoA, PoS leader selection, PoS consensus, MEV auctions, based rollups, proof of efficiency, etc.
- Shared Sequencers & X-Chain Atomicity
- This is the really fun and totally new stuff.
- MEV-Aware Design

- I'll just briefly cover some variations of FCFS. For encrypted mempools, you can refer to my recent posthere.

Many other requirements to improve rollup security will fall outside the context of this post (reducing upgrade keys' power, implementing robust and permissionless proofs, etc.).

How Rollups Work

Smart Contract Rollups (SCRs)

First, a brief refresher on how SCRs work. These are the rollups we're used to on Ethereum today. At a super high level, a SCR is basically just:

1. An ordered array of inputs (on the L1, so tx data must be posted on the DA layer)
2. Code being run over them (rollup node software)
3. Deterministic outputs (the rollup blockchain) of running the function over the inputs

More specifically, traditional sequencers commit to rollup blocks by posting their state root and calldata (eventually data blobs) to their associated L1 smart contracts. New blocks extend the tip of the rollup. The on-chain contract runs a light client of the rollup, keeping its block header hashes in storage. The contract is convinced of settlement upon receipt of a validity proof, or after the fraud proof window has passed. If an un-finalized ORU block is invalid, it (and all subsequent blocks) can be orphaned by submitting a fraud proof which rolls back the tip of the chain. Proofs help to protect the bridge:

Batch submission should require some type of bond/deposit to disincentivize malicious behavior. If a fraudulent batch is submitted (e.g. invalid state root), the deposit would be burned and given to the fraud challenger in some split.

SCRs have "merged consensus" - a consensus protocol that's verifiable on-chain. Rollup consensus can run entirely in an L1 smart contract. It doesn't affect or require any support from the main chain's consensus rules.

Decentralized consensus protocols generally consist of four primary features(note this is an oversimplification which doesn't cleanly account for many families of consensus protocols, e.g., leaderless ones

):

1. Block validity function

- State transition function. Block validity is executed off-chain and proven via validity proofs or the absence of a fraud proof.

1. Fork-choice rule

- How to choose between two otherwise valid chains. Rollups are intended to be fork-free by construction, so a non-trivial fork choice rule is not strictly required.

1. Leader selection algorithm

- Who is permitted to progress the chain by extending its tip with a new block.

1. Sybil resistance mechanism

- PoW, PoS, etc.

With 1 and 2 arguably provided, the bare minimum requirements to decentralize a sequencer are then some form of Sybil resistance + leader election. Fuel Labs has been in this camp, arguing that PoS:

- Should not be used for a full consensus protocol within a rollup (where rollup validators/sequencers would vote on blocks)
- Should only be used for leader selection within a rollup

There are also good arguments for why you should

have an L2 local consensus. More on this later.

Sovereign Rollups (SRs)

SRs still post transaction data to the L1 for DA and consensus, but they handle "settlement" (I already sound dumb, so it's fine) client-side within the rollup. DA layers tell you the data exists, but they do not define what the canonical chain is for the rollup:

- SCR

- The "canonical" rollup chain is decided by the L1 smart contract.

- SR

- There is no L1 smart contract which decides the "canonical" rollup chain. The canonical rollup chain can be decided by the rollup nodes themselves (checking for L1 DA, then locally verifying the fork-choice rule).

Related note: There's an interesting argument that there is no such thing as a globally canonical chain (only bridges deciding which chain to consider canonical), good counter arguments, and other great related Twitter threads on thetradeoffs all rollups make between sovereignty and (automatic) composability. I encourage you to sift through them as well asthis recent thread on Bitcoin sovereign rollups.

In any case, the "canonical" framework above is easy to understand, and it's not a central point for this piece here, so I'm going with it. For more background on general rollup architecture, you can refer here.

Decentralizing Sequencers

User-Initiated Forced Transaction Inclusion

Smart Contract Rollups

As mentioned above, sequencers are generally responsible for batching transactions and posting them to the L1 smart contract. However, users can also directly insert some transactions to the contract themselves:

Of course this is inefficient and expensive, so sequencers will batch transactions and submit them together in the normal course. This amortizes the fixed costs over many transactions and allows for better compression:

The sequencer promises to eventually publish these transactions on the L1, and we can compute the outputs in the meantime to get softer pre-confirmations:

When the sequencer posts these transactions to the L1, that solidifies the outputs:

Normally, users would just include transactions themselves when coming from the L1 to bridge funds up to the L2. This is added as an input to the L1 contract which tells the L2 that it can mint funds on the L2 backed by this locked asset on the L1.

If I want to get my money back to L1, I can burn it on the L2 and tell the L1 to give me my money back down there. The L1 doesn't know what's going on with the L2 (it's not executing its transactions), so a proof needs to be submitted along with the request to unlock my funds on L1.

Because I'm coming from the L2, the sequencer can initiate this withdrawal request and submit it to the L1. However, now you're trusting the L2 sequencer for censorship resistance (CR)

. You're no longer operating under the same guarantees you get on L1. Maybe they don't like you, or the sequencer just shuts down, so you're stuck on L2 forever now.

Rollups can increase their own CR locally with a variety of measures. This may include having an L2 consensus set with high value staked, some variation of inclusion lists, adding threshold encryption, etc. to minimize the chances of L2 user censorship. These are great, but ideally we also want L2 users to have the same CR assurances as the L1.

If users are being censored, they need some way to either force withdrawal from the rollup or force inclusion of their transactions into the L2. That's why L2 users should retain the ability to do things like force inclusion of their L2 transactions directly into the L1 contract themselves. For example, censored users may be able to submit a single-operation batch themselves directly to the L1.

It's not ideal though if the only option for L2 users is to force transactions into the L1 directly. This is likely to be infeasible for many low-value users, particularly as the L1 gets more expensive to interact with. More advanced designs may be able to work around this limitation, forcing atomic transactions between rollups. Kalman Lajkó is working on a fascinating design here which I highly recommend reading. It looks to enable cross-rollup forced transactions in systems with shared provers and DA layers.

Sovereign Rollups

Forced inclusion works differently for SRs because, as mentioned earlier, they enforce their fork-choice rule differently than SCRs (great post by Sovereign Labs on this here).

In a SCR, the L1 smart contract enforces the rollup's fork-choice rule. In addition to verifying the ZK proof, it also checks that the proof builds on the previous proof (and not some other fork), and that it's processed all of the relevant forced transactions which were sent on L1.

A SR can post its ZK proofs to the L1 DA layer for all to see as calldata/blobs (even though the L1 isn't verifying them). Then, you simply add a rule that a new proof is only valid if it builds on the previous valid proof. This rule could be enforced client-side, but it would then require users to scan the chain's history back to genesis or some checkpoint.

Instead, the calldata can be tied back to the L1 block headers, and a statement can be added saying "I have scanned the DA layer for proofs (starting at block X and ending at block Y), and this proof builds on the most recent valid proof". This proves the fork-choice rule directly in the proof rather than enforcing it client side.

Since you're already scanning for proofs, you can also prove that you've scanned for any forced transactions. Anyone can post a forced transaction directly to the L1 DA layer when they wish.

Tiers of Transaction Finality & ZK For Fast Finality

On-chain proof verification on Ethereum is generally very expensive, so current ZKRs such as StarkEx tend to only post STARKs to Ethereum on the order of every few hours. Proofs tend to grow very slowly relative to the number of transactions, so this batching meaningfully saves on costs. However, this long finality time isn't ideal.

If a rollup only posts state diffs on-chain (and not full transaction data), then even full nodes can't be assured of this eventual finality without a proof. If the rollup's full transaction data is posted on-chain, then at least any full nodes can finalize along with the L1.

Normally light nodes would just rely on centralized sequencer soft confirmations. However, a ZKR could instead quickly generate and distribute ZK proofs in the p2p layer for all light clients to see in real-time, providing them finality at the L1 speed as well. Later on, these proofs can be recursively batched and posted to the L1.

This is what Sovereign Labs is planning to do, and similarly Scroll is planning to post intermediate ZK proofs on-chain (but not verify them), so light clients can sync fairly quickly. With both of these constructs, rollups can start to finalize at the speed of the L1 rather than waiting to save on gas costs. Note that in both cases, you're simply bringing hard finality time back down to the absolute minimum (L1 speed).

Different sequencer designs will never give you that strongest level of finality faster than L1 block times. The best that different sequencer designs can do is give you pre-confirmations

faster than L1 block times with varying levels of certitude (e.g., you probably can trust the pre-confirmation of an L2 with a decentralized consensus set with high value staked more than you can trust the pre-confirmation of a single trusted sequencer).

Patrick McCorry also recently put out a great overview on theTiers of Transaction Finality for Rollups. You probably get the basic idea now:

- There are varying levels of transaction "finality" depending on who's giving you the promise (and what the rollup

construct is)

- Different actors will have different levels of awareness of the "truth" at a given time (e.g., L2 light clients, full nodes, and L1 smart contracts will realize the same "truth" at different times)

Single Sequencers

Today, most rollups have one sequencer permissioned to submit batches. This is highly efficient, but provides weaker real-time liveness and CR. With proper safeguards in place, this may be acceptable for many use cases:

- CR

- Mechanisms for users to force transaction inclusion as described above.

- Liveness

- Some sort of hot backup option if the primary sequencer goes down (similarly backups in place for ZKR provers, and ORU fraud provers should simply be permissionless). If reserve sequencers go down as well, anyone should be able to step in.

Reserve sequencers could be elected by the rollup's governance for example. With this setup, users get safety, CR, and liveness. Single active sequencers may be a viable option even in the longer run then.

Base is likely the start of a trend. Companies can now manage and optimize their product like they got all excited about with that enterprise blockchain hype nonsense, but it can actually be a permissionless, safe, and interoperable chain now.

Base intends to decentralize their sequencer set eventually, but the point is they don't strictly need

to, and others may not (or they could to a very limited extent such as a small sequencer set). To be clear, this requires rollups to implement the necessary steps so that it's actually safe and maintains CR (remove arbitrary instant upgrades, implement robust proofs, forced transaction inclusion, MEV auctions, etc). It's not safe today.

This will be a massive improvement over centralized/custodial products, not a replacement for maximally decentralized ones. Rollups simply expand the design space. This is also largely why sequencer decentralization hasn't been a top priority for most rollup teams - those other items are simply far more important for ensuring user safety, CR, and reducing trust in the rollup operators.

However, it's still not ideal if users/other parties need to step in to maintain liveness and "real-time CR" (as opposed to "eventual CR" such as forcing transactions through the L1). Depending on the forced transaction inclusion mechanism, it may be costly or impractical for low-value users to step in. Rollups with a high preference for maximum guarantees around real-time CR and liveness will look to decentralize. There may also exist regulatory considerations in operating a single permissioned sequencer.

Proof-of-Authority (PoA)

One straightforward improvement over a single sequencer would be permissioning a handful of geo-distributed sequencers (perhaps other reputable companies). Sequencers could simply rotate equally in a round robin style. Having them put up a bond would help incentivize honest behavior.

This general concept shouldn't be too unfamiliar - multisig bridges often have a handful of trusted corporates, or similarly the committee for something like Arbitrum's AnyTrust DA. Importantly though, they have far less power here (you don't rely on rollup sequencers for safety, unlike how multisig bridge operators could withdraw locked funds). Overall, this is better for CR and liveness vs. a single sequencer, but still imperfect.

Sequencer Auctions a.k.a. MEV Auctions (MEVA)

Instead of allocating sequencer rights based on stake, a rollup could also directly run aMEV auction (MEVA) via a smart contract. Anyone can bid for the right to order transactions, and the auction contract awards sequencing rights to the highest bidder. This could be done per block, or for an extended period of time (e.g., bidding for the right to be the sequencer for the next day). Winning sequencers should still post some bond so that penalties can be imposed if they are then faulty/malicious.

In practice, an out-of-protocol MEVA will naturally result if the auction isn't directly baked into the protocol. If sequencing rights are determined based on stake weight, some form of MEV-Boost/PBS-style auction system will arise, similar to what we see on L1 Ethereum today. In this case, fees/MEV would likely go to stakers. If the auction is baked into the protocol, then fees/MEV would likely go to some form of rollup DAO treasury (though it could also be distributed, burned, etc. in either case).

Permissionless PoS for Leader Election

Permissionless to join as a sequencer, but you must stake (likely the L2's native token). The staking mechanism can be

established on the base layer through a smart contract or directly within the rollup. You can use this PoS + some form of randomness on-chain to do leader selection in much the same way as any L1. Your probability of getting to sequence a block = your proportion of total stake. Penalties can be imposed on faulty/malicious sequencers via missed rewards, inactivity leaks, and slashing.

Note that this doesnot

require the sequencers to reach consensus for the reasons described above. Rollups use the L1 for consensus, so local consensus is not required. Stake determines the rotation of what sequencers can propose blocks, but they don't need to vote on the blocks proposed by other sequencers.

This could grant sequencing rights for an arbitrary length epoch. You may be entitled to sequence for 100 consecutive rollup blocks, or 1000, etc. Longer periods may be more efficient, and only a single sequencer is needed at a given time. However, granting extended monopolies can have other externalities. Alternatively, the leader could alternate every block as with normal L1s.

Dymension

Dymension is one project working along these lines. The Dymension Hub will be a typical honest majority PoS L1 in Cosmos. Its L2s ("RollApps") will use it for settlement and consensus while relying on Celestia for DA (so these L2s are actually "optimistic chains" not "rollups").

Per their Litepaper, decentralized RollApp sequencing will require staking DYM (Dymension's native asset) on the Dymension Hub. Leader selection is then determined by the relative amount of DYM staked. These sequencers will receive the revenue (fees and other MEV) from their respective rollups then pay the associated underlying costs to the Dymension Hub and Celestia.

As a result of this mechanism, nearly all value capture in this stack accrues directly to the DYM token.

Rollups that use their own native token for sequencing (as StarkNet intends to do with STRK, described below) accrue value to their own

token. This setup here is analogous to if Ethereum rollups could only ever use ETH for their sequencer election.

In my opinion, this massively reduces the incentive to deploy an L2 on such a settlement layer. Most L2 teams will naturally want their own token to accrue meaningful value (not just be used for a fee token, as would be possible here). They're running a business after all.

Permissionless PoS for Leader Election & L2 Consensus

It's also possible to use L2 staking for sequencer election and

local consensus prior to L1 confirmation if desired. This is exactly howStarkNet plans to use their STRK token:

- PoS Sequencer Leader Election

- As described above, some form of leader election is needed.

- PoS Consensus

- Incentivize L2 validators to reach temporary L2 consensus before L1 finality is reached, giving stronger pre-confirmations. As described above, this is not a strict requirement, but it's an attractive option.

Additionally, STRK may be used in some form for:

- PoS Consensus for DA

- Incentivize provisioning of alt-DA (volition) which requires separate consensus.

- Proving -

Incentivize provers to produce STARKs.

The transaction flow looks like this:

1. Sequencing

- Sequencer orders txs and proposes a block

1. L2 Consensus

- StarkNet consensus protocol signs off on proposed blocks

1. Proof Production

- Provers generate a proof for the blocks which consensus has agreed to

1. L1 State Update

- Proofs submitted to L1 for state update

For more detail on StarkNet's plans, you can refer to theseries of forum posts starting with this one

L2 Consensus, or Just L1 Consensus?

As we saw, an L2 may or may not implement its own local consensus (i.e., L2 validators signing off on its blocks before sending them to the L1 for final consensus). For example, the L1 smart contract can be aware based on its rules that:

- PoS for Leader Election & Consensus

- "I can only accept blocks that have been signed off on by L2 consensus."

- PoS for Leader Election

- "This is the selected sequencer allowed to submit blocks at this time."

To have a rollup without local consensus, all you need to do is:

1. Make rollup block proposing permissionless.
2. Create some criteria to select the best block built for a given height
3. Have the nodes or settlement contract enforce the fork-choice rule
4. Inherit consensus and finality from L1

Note that in either case, the value from the L2 can accrue to the rollup token. Even if the L2 token is just used for some form of leader selection (not consensus voting), the value from sequencing rights still accrues to the L2 token.

Disadvantages of L2 Consensus (Only Leader Selection)

Now let's discuss the tradeoffs of having/not having local consensus prior to the L1.

One argument as made by the Fuel Labs team is that it can decrease CR. "This allows a majority of validators to censor new blocks, meaning user funds can be frozen. There is no need for PoS to secure a rollup, because a rollup is secured by Ethereum." This is a bit of a gray area here. As described previously, you can still provide CR even with censoring sequencers (e.g., forcing transactions into the L1 directly, or more intricate designs such as the one Kalman Lajkó is working on).

Another argument is that full consensus is simply 'Inefficient." For example, it seems easier to just have:

- One lead sequencer at a time runs everything on a single box, vs.
- One lead sequencer at a time runs everything on a single box, then all other nodes need to vote and agree on that proposal.

This of course varies widely depending on the exact sequencer design and consensus mechanism chosen.

Additionally, note that some have raised concerns about using PoS in sequencer decentralization as notedhere and here. The intricacies of L1 vs. L2 may make it more challenging to deal with certain types of attacks.

Advantages of Adding L2 Consensus (In Addition to Leader Selection)

Probably the largest goal for sequencers is to give users faster soft confirmations prior to the full safety and security of L1. Looking at StarkNet's mechanism requirements:

"Strong and Fast L2 Finality - StarkNet state becomes final only after a batch is proven to L1 (which might take as long as several hours today). Therefore, the L2 decentralization protocol should make meaningful commitments regarding the planned execution order well before the next batch is proved."

Adding some form of consensus backed by the economic security of many sequencers helps to providestronger guarantees in the interim that your pre-confirmations are good to go:

"Starknet consensus must be strongly accountable in the sense that safety and liveness violations are punishable by slashing for any fraction of the participants (including malicious majority of stake)."

Rollups also have flexibility to experiment with different tradeoff points on the spectrum of consensus mechanism options, as they can always fall back to the safety and dynamic availability of Ethereum L1 eventually.

L1-Sequenced Rollups

All of the above constructs privilege sequencers to create rollup blocks in one form or another. For example, PoS is permissionless to join, but the elected L2 sequencer for a given slot is the only party capable of submitting a block at that time. Alternatively, there are a few related proposals which do not

privilege any L2 sequencers. These designs place reliance on the L1 itself for their transaction sequencing.

Total Anarchy

Vitalik proposed this "total anarchy" idea back in 2021. Allow anyone to submit a batch at any time. The first transaction that extends the rollup is accepted. It fulfills the two minimum requirements discussed above to decentralize sequencers:

- Sybil resistance

- Sybil resistance is provided by the L1 (i.e., transaction fees and block size/gas limits).

- Leader selection

- Leader selection is implicit and post-facto.

This is sufficient because the L1 already provides security. If L2 blocks have been posted to the L1, they can only be orphaned if they're invalid or building upon invalid blocks (will be rolled back). If they're valid and posted to the L1, they have the same security guarantees as the L1 itself.

The big problem Vitalik noted - it would be incredibly inefficient. Multiple participants are likely to submit batches in parallel, and only one can be successfully included. This wastes a large amount of effort in generating proofs and/or wasted gas in publishing batches to chain. It's very messy knowing if your transactions will get in quickly, and it's not cost-effective.

Based Rollups

However, PBS could make this anarchy design workable now. It allows for more regimented sequencing with at most one rollup block per L1 block and no wasted gas. (Though you may have wasted computation). L1 builders can just include the highest value rollup block, and build this block based on searchers' input bids similarly to any L1 block. It may also be reasonable to make ZK-proving permissioned by default (with a permissionless fallback), to avoid wasted computation.

This is the core idea behind Justin Drake's recent proposal for "based rollups." He uses the term to refer to rollups whose sequencing is driven by the L1 (the "base" layer). L1 proposers would simply make sure to include rollup blocks within their own L1 blocks (presumably via builders). It's a simple setup that offers L1 liveness and decentralization off the bat. They get to sidestep tricky problems like solving for forced transaction inclusion in the event that L2 sequencers are censoring. Additionally, they remove some gas overhead as no sequencer signature verification is required.

One interesting question raised was in regards to where these L2 transactions go to get processed. L2 clients will need to send these transactions somewhere for L1 searchers/builders to pick them up and create their blocks and data blobs. They could possibly be sent to:

- L1 Mempool

- They could be sent along with some special metadata for "informed" searchers/builders to interpret them. However, this could significantly increase the load on the L1 mempool.

- New p2p Mempools for Each L2

- Something along these lines seems more tenable. Searchers/builders would just start checking and interpreting these in addition to their usual channels.

One clear downside here is that based rollups limit sequencer flexibility. For example:

- MEV mitigation

- Rollups can get creative with things like variations of FCFS, encrypted mempools, etc.

- Pre-confirmations

- L2 users love fast transaction "confirmations." Based rollups would fall back to L1 block times (12 seconds) at best or wait longer to post a full batch.

Interestingly, this is exactly what early rollup teams were building:

Fun fact: the initial Arbitrum implementation & first public testnet (Ropsten L2) worked like this.

Sequencer was introduced later b/c there was demand / froth for fast transactions.https://t.co/lPTrzo0lHA

— Daniel "sorry if this is a bit niche" Goldman (@DZack23)March 14, 2023

Justin noted that restaking may be able to help here.

Wow pic.twitter.com/zs786upj91

— Jon Charbonneau (@jon_charb) March 15, 2023

These are both areas of research aroundEigenLayer at least as mentioned in their whitepaper here. It's unclear though that this is a viable solution to solve the problem practically here. For restaking to meaningfully improve these shortcomings, you'd likely want all stakers to opt into running it. It seems more logical to simulate this idea by just having the subset of stakers that want to do this opt into a separate shared sequencing layer (more on this shortly).

Proof of Efficiency (PoE)

Last year, Polygon Hermez made a proposal called PoE. It's another variation of L1-sequencing specifically intended for ZKRs. The sequencer here is a completely open role where anyone can submit a batch (i.e., total anarchy/based, and so it carries the same limitations). PoE consists of a two-step process with two parties:

Sequencer

Sequencers collect L2 user txs and create a batch by sending a L1 tx (which includes the data of all selected L2 txs). Sequencers will submit blocks based on economic value received, or to fulfill a service level for users (e.g., post a batch in every L1 block, even if it makes L2 txs more expensive because users want faster txs).

Sequencers will pay L1 gas fees to post a batch, and the protocol defines an additional fee in MATIC that must be deposited. Once posted, the winning batch immediately defines the new tip of the chain, allowing any node to deterministically compute the current state. A validity proof will then be needed to finalize light clients' view of this (including the L1 smart contract).

Aggregator

Aggregators here are the ZK-provers. Again, it's a permissionless role which anyone can compete at. Very simply:

- Sequenced batches with transaction data are sorted on the L1 by the position in which they appeared on the L1.
- The PoE smart contract accepts the first validity proof that updates to a new valid state including one or more of the proposed batches which have not yet been proven.

Aggregators are free to run their own cost-benefit analysis to find what the right frequency of posting a proof is. They get a portion of the fees if they win the race, but waiting longer to post a new proof amortizes their fixed verification costs over more transactions. If aggregators post a proof late (it doesn't prove a new state), then the contract will just execute a revert. The prover wasted computation, but they'll save most of their gas.

Fees can be distributed as follows:

- Fees from L2 txs will be processed and distributed by the aggregator that creates the validity proofs.
- All tx fees will be sent to the corresponding sequencer of each batch.
- The sequencers' deposited fees to create a batch will be sent to the aggregator that included this batch into a validity proof.

Pure Fork-Choice Rule

Rollkit SRs have a very similar "pure fork-choice rule" concept as noted here to refer to any rollup without privileged sequencers. Nodes defer to the DA layer for ordering and apply a "first-come-first-serve" fork-choice rule.

Economics of L1 Sequencing

These L1 sequencing designs have important economic consequences in thatMEV for L2 transactions would now be captured at the level of the L1 block producer. In the "traditional" L2 sequencing models, MEV for L2 transactions is captured by the L2 sequencer/consensus participants/auction mechanism. It's unclear that much MEV would ever leak down to the base layer in this case.

There are mixed feelings over whether this is a good thing or a bad thing:

- Good

- Some like the "L1 economic alignment" (e.g., ETH captures more value).

- Bad

- Others are concerned about the base layer incentives (e.g., centralization risk for Bitcoin miners, but maybe it's too late anyway).

This could make sense particularly as an easier bootstrapping method for a rollup, but it's hard to see most rollups giving up so much MEV to the L1 that they could capture themselves. One of the great benefits of rollups is indeed economic - they'll need to pay very little back to the L1 once DA starts to scale and those costs drop. The downsides of slower block times and naive approach to MEV also seem suboptimal for users.

Incentivizing ZK-Proving

As a brief aside, note that the competitive racing market described above in PoE may tend towards centralization around the fastest aggregator. ZK-prover markets very broadly have two economic problems to solve for:

- How to incentivize the prover to create this proof
- How to make proof submission permissionless so that it's a competitive and robust market (e.g., if the main prover goes offline)

Let's consider two simple models for a ZK-prover market:

Competitive Market

At one extreme, you could have an open competitive model. A permissionless market of provers all race to create a proof first for the blocks that the rollup sequencers/consensus have produced. The first to create the proof could get whatever reward is designated for provers. This model is highly efficient in finding the best prover for the job.

This looks pretty similar to proof of work mining. However, there's a unique difference here - proofs are deterministic computations

. As a result, a prover with a small but consistent edge over other provers can almost always

win. This market is then very prone to centralization.

In PoW mining, the randomness aspect has much nicer properties - if I have 1% of the mining hashpower, I should get 1% of the rewards.

This competitive proving model is also suboptimal in terms of computation redundancy - many provers will compete and spend resources to create the proof, but only one will win (similar to PoW mining).

Turn-Based (e.g., Stake-Weighted)

Alternatively, you could rotate among provers giving them each a turn (e.g., based on some stake deposited, or reputation). This may be more decentralized, but it may be less efficient on proof latency ("slow" provers will get a chance to create proofs when another prover may have been capable of creating the proof faster and more efficiently). However, it can prevent the wasted computation of many provers racing to create a proof when only one will land.

Additionally, you have an issue if the person whose turn it is fails to produce a proof (either maliciously or accidentally). If these turns are long (e.g., a given prover gets a monopoly for several hours) and the prover goes down, it would be difficult for the protocol to recover. If turns are short, other provers could step in and catch up where the primary prover failed to do so.

You could also allow anyone to post a proof, but only the designated prover can capture the reward for a given time. So if they go down, another prover could post the proof, but they wouldn't be able to capture the reward. This would be altruistic then to expend resources doing the computation for no reward.

Scroll is exploring more along the lines of a turn-based approach, dispatching execution traces to randomly selected "rollers" (provers):

There are also many interesting questions on how fees should be charged for proving at the user level at time of sequencing. Further discussion on these topics and more can be found here:

- Decentralized zk-Rollup by Ye Zhang at Scroll discusses this possibility of a rotational roller network based on staking + a MEVA for sequencing rights, without the need for an L2 consensus
- An Overview of Scroll's Architecture provides more detail on a possible roller model
- Starknet Decentralized Protocol IV - Proofs in the Protocol

- Starknet Decentralized Protocol VI - The Buffer Problem

Shared Sequencing

Most of the earlier solutions presume that each rollup needs to figure out how to decentralize their sequencer all on their own. As we saw with the L1-sequenced options, that's simply not the case. Many rollups could instead opt into one shared sequencer (SS). This has some great benefits:

- Laziness

- Don't worry about decentralizing your sequencer anymore, it's hard! Just plug into this option. No need to recruit and manage a validator set. At the risk of getting even more pitch decks that say "modular", this would indeed be a very "modular" approach - strip out transaction ordering into its own layer. SSs are literally a SaaS company (sequencers as a service).

- Pooled Security & Decentralization

- Let one sequencing layer build up robust economic security (stronger pre-confirmations) and real-time CR rather than implement many small committees for each individual rollup.

- Fast Transactions

- Other single-rollup sequencers could do this as well, but just noting that you can still get those super-fast sub-L1 block-time pre-confirmations here.

- Cross-chain Atomicity

- Execution of transactions simultaneously on Chain A and Chain B. (This one is complicated, so I'll unpack it later in more depth).

Simply using the native L1 as the sequencer for many L2s fundamentally has several downsides as mentioned previously:

- Still limited to the L1's data and transaction ordering throughput
- Lose the ability to give L2 users fast transactions below the L1 block times (albeit with weaker guarantees prior to eventual L1 consensus)

The best that L1-sequencing can do is remove the L1's computational bottlenecks (i.e., if transaction execution is the bottleneck for throughput) and achieve small factor improvements on communication complexity.

So, can we design specialized and more efficient SSs as opposed to just letting the L1 do it...

Metro - Astria's Shared Sequencer

Metro is one proposal for a SS layer. You can refer toEvan Forbes' research post, Modular Insights talk, and Shared Security Summit talk for more details. Astria, led by Josh Bowen, is the team that is working on actually implementing this design of Metro.

Separation of Execution & Ordering

Rollup nodes today really handle three things:

The key property here would be separation of execution and ordering

. This SS:

- Orders

transactions for many chains that opt into it as their sequencing layer

- Doesn't execute (or prove)

these transactions and generate the resulting state

Ordering is stateless - SS nodes then no longer need to store the full state for all of the different rollups. They remove execution computation. The gigantic bottlenecks which traditional sequencers currently handle are gone.

When you strip out execution from consensus it goes super

fast. You're now just limited in the gossiping layer (fast). Nodes can be incredibly efficient if all they have to do is produce ordered blocks of transactions and agree on the block without executing everything. Execution and proving can be done by separate parties after the fact.

Pooling Sequencer Security & Decentralization

SS nodes can then be kept relatively lightweight or even horizontally scalable (by choosing a random subset of consensus nodes to order different subsets of transactions). The result - you can likely make this sequencing layer far more decentralized vs. a traditional sequencer that needs to hold the chain's heavy state on hand and do full execution.

Additionally, we're pooling the resources across many chains - no need to fragment the PoS consensus across many rollups. Aggregate it all in one place. This effect likely yields a more decentralized sequencer set (CR) with more slashable value staked (reorg resistance) as compared to many rollups implementing their own sets. This is important because:

- Sequencing

- First line of defense to provide real-time CR and liveness to rollup users.

- Execution & Proving

- Can be done after the fact without strong decentralization requirements. We just needone honest party.

Once transaction ordering has been agreed to, execution (and proving) can be deferred to an entirely different chain after the fact:

1. Soft consensus & sequencing

- Shared sequencer gives users fast pre-confirmations

1. Firm consensus & DA

- Transaction data has been finalized on the DA layer for all to see)

1. Lazy execution and proving

- Anyone can execute and prove the transactions after the fact

This later execution layer doesn't need to be as decentralized because this isn't where CR comes from. Single sequencers aren't ideal for CR, but that doesn't

come from their role as an executor. It comes from the fact that they order and include transactions

. Here, the SS is already providing the ordered input of transactions, and thus CR. That later calculation and comparison of the state commitment then doesn't need to be as decentralized.

Soft Execution

That step 1 of fast soft execution is what users love:

This requires some form of consensus (or a centralized sequencer) to provide that great UX here:

If you were just to rely on the consensus of the base layer such as Celestia, you can't provide these kinds of soft promises around ordering and inclusion with good assurance. The SS can provide pretty strong commitments on fast blocks (sub-L1 block times) if it has a decentralized committee with high value staked.

So users can get soft confirmations as soon as the SS creates a block. Anyone can just download the agreed upon transactions and apply them to the state prematurely. The strength of this confirmation is dependent upon the construction of the SS (decentralization, economic security, fork-choice rule, etc.). Once the data actually gets posted to the base layer, you can treat these transactions as truly final. Then final computation of the state root and associated proofs can be generated and committed to.

Lazy Rollups

"Lazy rollups" are very simple. They wait until their transactions have all been ordered and posted on the DA layer, then they download those transactions, optionally apply a fork choice rule to select a subset of transactions, perform transaction processing, and apply those transactions to the state. Headers can then be generated and gossiped.

Note that because the SS can't produce blocks in a way that requires access to full state, they're not

checking for invalid state transitions. As a result, the state machine of "lazy rollups" using a SS must be able to deal with invalid transactions. Nodes can simply drop invalid/reverting transactions when they execute the ordered transactions to compute the resulting state. Traditional rollups that do immediate execution don't have this limitation.

Rollups that require state access to malleate transactions before including them on-chain wouldn't work here. For example, if a rollup has a block validity rule where all transactions included in the block are valid transactions that don't fail. If the rollup requires transaction malleation but not state access, then a special SS could be created exclusively for rollups of this type (e.g., something like Fuel v2 or a rollup with a private mempool).

Paying For Gas

For this SS to operate, there must be some mechanism for users to pay for their transaction inclusion into the L1. You could simply use the existing signature and address already included in most rollup transaction types to also pay for gas on the SS layer. This would then be the one thing that would require the SS to have awareness of some minimal state depending on implementation (e.g., to parse the signature, nonce, subtract gas from the account). Alternatively, payment could involve some wrapper transaction on the SS where anyone could pay for arbitrary data to be included. It's an open design space here.

Fork-Choice Rule

Rollups are able to inherit the fork-choice rule of the SS they are using. The rollup's full nodes are then effectively light clients of the SS, checking that some commitment to indicate which rollup block is the correct one for a given block height.

However, inheriting the SS's fork-choice rule is optional - you can simply require the rollup to process (not necessarily execute) all of its transaction data posted to the base layer. It would effectively inherit the CR and liveness of the base layer, but you'd sacrifice a lot of the nice SS features that users love.

MEV

Assuming a rollup wants to inherit the fork-choice rule of its SS and get fast soft execution, the SS will naturally be in a very central position in regards to MEV. It decides the inclusion and ordering of transactions that the rollup will abide by.

However, it's not inherently the case that a rollup must

execute the transactions provided by the SS, or in the order provided. You could technically allow for your own rollup operators to do a second round of processing to reorder the transactions published by the SS after the fact upon execution. However, as noted above, you'd then lose most of the nice properties of using a SS in the first place, so this appears less likely.

Even in this case though, there's likely to still be MEV at the SS layer since it has power over the inclusion of transactions. If you really wanted, you could even allow your rollup to exclude certain transactions on the second round of processing (e.g., having some validity conditions to exclude certain transaction types), but this of course can get messy, reduce CR, and again mostly lose the benefits of a SS.

Swapping Out a Shared Sequencer

The thing in blockchains that's hard to fork is any form of valuable shared state. Looking at something like ETH vs. ETC or similarly ETH vs. ETH POW, social consensus decides what the "real Ethereum" is. What's the "real" state that we all agree is valuable.

However, SSs are really just a service provider - they don't have a valuable state associated with them. Rollups using a given SS then preserve the ability to fork it out in favor of some other sequencing mechanism with only a minor hardfork (e.g., if the SS is extracting too much value). This should hopefully keep SSs competitive.

The thing that could make this more challenging would be network effects. If many rollups all start using a SS and they get great network effects (e.g., if they derive meaningful value from the potential cross-chain atomicity with many chains), it may be painful to detach unless other rollups are willing to move over as well.

Espresso Sequencer (ESQ) - Secured by EigenLayer

You might've seen that the EigenLayer whitepaper mentioned decentralized SSs as one of the potential consumers of restaking. This SS could be secured by ETH restakers, and it would handle the transaction ordering for many different L2s.

Well Espresso just publicly announced exactly that in their plans for a shared sequencer. It can leverage EigenLayer restakers to secure its consensus. To give a nice visualization, this is what rollups look like today:

And this is what they'd look like with a SS such as Espresso:

The Espresso Sequencer (ESQ) is very

similar overall to the general ideas of Metro. It would work by the same core principle - it strips out transaction execution from ordering. In addition to this, ESQ will provide data availability

for transactions as well.

HotShot Consensus & Espresso DA (Data Availability)

For background, Ethereum currently uses Gasper for consensus (Casper FFG as the finality tool + LMD GHOST for its fork-choice rule). The TLDR relevant here is that Gasper maintains liveness even under pessimistic conditions where most nodes

could drop off the network (dynamic availability). It effectively runs two protocols (Casper FFG and LMD Ghost) which together maintain a dynamically available chain with a finalized prefix. Gasper trades off on fast finality (the ability to confirm transactions as fast as the network will allow).

Overall, ESQ consists of:

- HotShot

- ESQ is built on the HotShot consensus protocol which prioritizes fast finality (optimistic responsiveness) over dynamic availability, unlike Gasper. It also scales to support an incredibly high validator count, as Ethereum does.

- Espresso DA

- ESQ will also provide DA to chains opting into it.The mechanism is also used to scale their consensus in general.

- Sequencer Contract

- Smart contract which verifies HotShot consensus as a light client and records checkpoints (commitments to points in the log of ordered transactions). Additionally, it manages the stakers for ESQ's HotShot PoS consensus.

- Network Layer

- Enables communication of transactions and consensus messages between nodes participating in HotShot and the Espresso DA

- Rollup REST API

- API that L2 rollups use to integrate with the Espresso Sequencer.

Let's take a closer look at the DA. In the optimistic case, high bandwidth nodes will make data available to all other nodes, and the availability of each individual block is also backed up by small randomly elected committees. Given the risk of DDoS and bribery attacks on small committees, verifiable information dispersal (VID) will be used to provide a reliable (but slower) backup path to guarantee DA so long as a sufficiently high fraction of all nodes (by stake) isn't compromised.

This system is built for scale, so ESQ is looking to provide cheaper DA to the L2s opting into it as a SS. They'll still settle their proofs and state updates to L1 Ethereum, but note that this would make chains using ESQ by default "validiums" or "optimistic chains" not full "rollups" (i.e., their DA isn't guaranteed by Ethereum L1). It's stronger than simple implementations of data availability committees (DACs), but it's a weaker guarantee than true rollups.

Transaction Flow

- Sequencer Contract

- HotShot interacts directly with its L1 sequencer contract. It validates the HotShot consensus and provides an interface for other participants to view the blocks it has sequenced. This contract stores an append-only log of block commitments, not

full blocks. Anyone can however authenticate any block against the commitment.

- L2 Contracts

- Each L2 using ESQ still has its own Ethereum L1 rollup contract. To verify the state updates sent to each rollup (via validity/fraud proofs), each rollup contract must have access to the certified sequence of blocks which led to the claimed state update. They interface with the sequencer contract to query these.

Transactions forwarded to the SS will be sequenced then sent back to the rollups' executors and provers before it is finalized on L1. The SS also sends a commitment to the block to its L1 sequencer contract along with a quorum certificate used to authenticate the block. This allows the L1 rollup contracts to compare the rollup state update proof against a block commitment which is certified as being the output of consensus.

A full view of the transaction flow:

Cross-Chain Atomicity

As noted in the Espresso post, a SS can provide some exciting use cases in regards to cross-chain atomicity:

"A sequencing layer shared across multiple rollups promises to make cross-chain messaging and bridging cheaper, faster, and safer. Elimination of the need to build a light client for another chain's sequencer is a free benefit with no cost, creating potential upfront savings. Ongoing savings are also made possible for cross-rollup bridging by removing the need for a given rollup to stay independently up-to-date with other rollups' consensus. Shared sequencers also offer security benefits for bridging: the shared sequencer can guarantee that a transaction is finalized in one rollup if and only if (or even at the same time) it is finalized in the other.

Additionally, shared sequencers enhance users' abilities to express atomic dependencies between transactions across different rollups. Conventionally, Alice would sign and publish her rollup-A transaction t independently of Bob's rollup-B transaction t'. In this case, Alice's transaction might get sequenced long before Bob's, leaving Bob with a long-duration option to abort (e.g., a trade). This optionality imbalance is mitigated with a shared sequencer, where Alice and Bob can instead submit the two transactions together as a signed bundle (i.e., the sequencer must treat these two transactions as one)."

This has implications for cross-domain MEV as on-chain activity eventually starts to grow (I hope). The classic example is "atomic arbitrage." The same asset is trading on two different chains at two different prices. Searchers want to close that gap by executing two trades simultaneously without execution risk. For example:

- Trade 1 (T1

) - Buy ETH at a low price on Rollup 1 (R1

)

- Trade 2 (T2

) - Sell ETH at a high price on Rollup 2 (R2

)

For the arbitrage to be atomic = either both trades get filled, or neither gets filled. If both rollups are opted into the same SS, then it could fulfill this atomic arbitrage for a searcher. The SS here can guarantee that:

- T1

is included in the instruction stream to R1

, if and only if:

- T2

is also included in the instruction stream to R2

Assuming the rollup VMs execute all transactions in their respective streams sequentially (i.e., there's no such thing as an invalid instruction, just some instructions can throw errors with no impact on the state), then we can also guarantee that:

- T1

is executed

on R1

, if and only if:

- T2

is also executed

on R2

However, this still is not

the same guarantee you'd have if you're transacting on a shared state machine (e.g., entirely on Ethereum L1). As noted previously, the SS does not hold the state for these rollups, and they are not

executing the transactions. You cannot offer complete assurance that one of the transactions (on R1

or R2

) doesn't revert upon execution.

Building more advanced primitives directly on top of this would be questionable. For example, if you tried to build an instant burn-and-mint cross-chain bridging function on top of this SS that does the following in the exact same block height simultaneously:

- Burns an input on R1
- Mints an output on R2

You could have a situation where:

- The burn on R1

may throw an unexpected error such as being invalidated by another transaction upon execution, but

- The mint on R2

isn't invalidated by anything, so it fully executes.

You can see how that would be a big problem.

There may be some cases where you can be certain of the intended result of these two transactions as long as both are included in the input streams and thus executed, but that's often not the case. It's an open question how powerful it is to:

- Guarantee

- T1

and T2

will be included in their respective streams, and (potentially) both will execute.

- Without guaranteeing

- Successful execution of both transactions and the resulting desired states.

These "guarantees" may be sufficient for something like atomic arbitrages where the searcher already has their assets needed to execute these trades on each chain, but it clearly isn't the synchronous composability of a shared state machine. It's not a sufficient guarantee on its own for something like cross-chain flash loans

This may still be useful in other settings though when combined with other cross-chain messaging protocols. Let's look at how a cross-chain atomic NFT swap could be facilitated when used along with a cross-rollup messaging protocol:

- T1

moves ETH from U1

(user 1) into SC1

(smart contract 1) on R1

- T2

moves NFT from U2

(user 2) into SC2

(smart contract 2) on R2

- SC1

only allows U2

to redeem the ETH if and only if it first receives a message from SC2

confirming that the NFT was deposited

- SC2

only allows U1

to redeem the NFT if and only if it first receives a message from SC1

confirming that the ETH was deposited

- Both smart contracts implement a timelock such that if either leg fails, the respective parties can recover their assets

The SS here enables both users to commit atomically in step 1. Then, you use some form of cross-chain messaging for the rollups to verify each others' resulting state and unlock the assets to execute the swap.

Without a SS to conduct it atomically, the two parties could agree on a price. But then U1

could submit their transaction, then U2

could wait around and decide if they want to abort the transaction. With a SS, they would be locked into the trade.

This is pretty much the edge of the map for cross-chain atomicity with SSs. Summarizing:

- The precise strength and usefulness of the guarantees provided here are still unproven
- This is potentially very useful for cross-chain atomic arbitrage, and similarly may be useful for other applications such as cross-chain swaps and NFT trading (likely in conjunction with other messaging protocols)
- It may be helpful to provide additional crypto-economic guarantees (e.g., putting up a bond as collateral) to underwrite certain types of cross-chain transactions
- However, you'll never have an unconditional assurance of the results of your transactions (which you would get by atomically executing transactions together on a shared state machine)

For other interesting topics on cross-chain atomicity, I recommend checking out:

- Optimism's Superchain - Also exploring usage of a SS across OP Chains. The SS can come to consensus then attempt to fulfill atomic cross-chain transactions.
- Anoma - Heterogeneous Paxos and Typhon are a very different approach.
- Kalman's Cross Rollup Forced Transactions as mentioned earlier.

Shared Sequencer Summary

Summarizing, the basic idea of a SS is that it would try to do this:

Obviously this picture isn't a science, everything is highly subjective, and it's very dependent on the exact construct. But the TLDR comparisons would be something like this:

- Centralized Sequencer

- It's generally easiest to implement whatever features you'd like if you're in total control of the system. However, you have suboptimal guarantees on strength of the pre-confirmations, forced exits might be undesirable, suboptimal liveness, etc.

- Decentralized L2 Sequencer

- A distributed set with meaningful stake should increase the robustness of the rollup vs. a single sequencer. But, you might trade off on things like latency depending on how it's implemented (e.g., if many L2 nodes now need to vote before confirming a rollup block).

- L1-Sequenced

- Max decentralization, CR, liveness, etc. However, it's lacking features (e.g., fast pre-confirmations, data throughput limits).

- Shared Sequencer

- You get the functionality of a decentralized sequencer, benefits from sharing it with others (some level of atomicity), you don't need to bootstrap your own sequencer set, etc. However, you do have weaker assurances compared to L1 sequencing in the interim periods prior to total L1 finality. Additionally, a shared layer can aggregate the committee, economic security, etc. across many rollups into one place (likely much stronger than each rollup having their own committee).

Eventually, all rollups find their way back to 100% L1 security once the L1 finalizes. Most sequencer designs are just trying to give you really nice features with somewhat weaker assurances in the interim prior to getting the full safety and security of L1 settlement. They all make different tradeoffs.

SUAVE (Single Unifying Auction for Value Expression)

Decentralized Builders vs. Shared Sequencers

The differences here can get pretty confusing when we're talking about these shared layers trying to handle transactions for many other chains. Especially when SUAVE is often called a "sequencing layer" among other terms such as a "decentralized block builder for rollups." To be clear - SUAVE is very different from the SS designs described above

.

Let's consider how SUAVE could interface with Ethereum. SUAVE would not be enshrined into the Ethereum protocol in any way. Users would simply send their transactions into its encrypted mempool. The network of SUAVE executors can then spit out a block (or partial block) for Ethereum (or similarly any other chain). These blocks would compete against the blocks of traditional centralized Ethereum builders. Ethereum proposers can listen for both, and choose which they would like to propose.

Similarly, SUAVE would not replace the mechanism by which a rollup selects its blocks. Rollups could for example implement a PoS consensus set that may operate in much the same way that Ethereum L1 does. These sequencer/validators could then select a block produced for them by SUAVE.

This is very different from the SSs described above, where a rollup could completely remove the need for decentralizing their sequencer. They simply outsource this function by opting into something like Metro or ESQ, and they can choose to inherit its fork-choice rule. Ethereum, Arbitrum, Optimism, etc. wouldn't be changing its fork-choice rule to opt into SUAVE for all transaction sequencing.

SUAVE doesn't care what your chain's fork-choice rule is or how your blocks are chosen. It looks to provide the most profitable ordering possible for any

domain. Note that unlike SS nodes described earlier, SUAVE executors are typicallyfully stateful

(though they may also fulfill certain preferences which don't require state). They need to simulate the results of different transactions to create the optimal ordering.

To see the difference, let's consider an example where a user wants to run an atomic cross-chain arbitrage. These are the guarantees they could receive if they submit this preference to SUAVE vs. submitting it to a SS:

SUAVE + Shared Sequencers

Now, how might SUAVE interact with rollup sequencers as they decentralize, and possibly even SSs? Espresso certainly seems to believe that SUAVE is compatible with ESQ, as described in their post. ESQ is intended to be compatible with private mempool services such as SUAVE that can act as builders. It would look similar to the PBS we're used to on Ethereum, except now:

- Shared Proposer

= Shared Sequencer

- Shared Builder

= SUAVE

As in PBS, the builder can obtain a blind commitment from the proposer (the sequencer here) to propose a given block. The proposer only knows the total utility (builder's bid) it will get from proposing the block without knowing the contents.

Putting things together now, let's look back at a searcher who wants to do a cross-chain arbitrage again. SUAVE on its own could build and send to two different rollups:

- Block 1 (B1

) which includes Trade 1 (T1

) - Buy ETH at a low price on Rollup 1 (R1

)

- Block 2 (B2

) which includes Trade 2 (T2

) - Sell ETH at a high price on Rollup 2 (R2

)

But it's perfectly likely that B1

wins its respective auction and B2

loses (or vice versa). However, let's consider what happens if those two rollups are opted into the same SS.

SS nodes are really stupid basically.

They have no comprehension of what transactions are actually doing, so they need someone (like SUAVE, or other MEV-aware builders) to build a full block for them if they're going to be efficient. Well, SUAVE executors could then submit both B1

and B2

to the SS, conditional that both blocks are fill-or-kill (execute atomically or drop both).

Now you can get really good economic guarantees throughout that whole process:

- SUAVE = Shared Builder

= Can assure you what state results if B1

and B2

are both included and executed atomically.

- SS = Shared Proposer

= Can assure you that B1

and B2

are both included and executed atomically.

Restaked Rollups

I recently made a post on the Flashbots forum about SUAVE's potential economics security models TLDR I argued why rollups are probably a better option for SUAVE in the long-term than restaking.

Hasu also had some very thoughtful responses on the tradeoffs between restaking and rollups I encourage you to read. Here's the TLDR response:

- Rollups guarantee safety/CR/liveness through the L1, but SUAVE doesn't benefit from these as much as most chains do because a SUAVE Chain would not be intended for regular users. In fact, Flashbots is researching ways to actively limit the need to bridge funds to SUAVE as a user. Ideally a chain would mostly just require searchers/builders to keep enough operating capital there. This could be a far lower amount than traditional rollups built for users to have a lot of funds there. Also, forcing a state transition through the L1 is likely less valuable to searchers/builders in the context of this chain vs. regular users.
- DA is expensive, at least today. We'll see how this plays out with EIP-4844 and the likely following increase in demand for DA. Conversely, I'll note that restaking may be more "expensive" to token holders (i.e., if fee revenue is low, and inflation to ETH restakers is needed to incentivize sufficient economic security). One is more expensive to users, while the other is more expensive to token holders. A lot of this comes down to timing of any potential chain, and the maturity of Ethereum's DA at that time.
- SUAVE needs to report state transitions from other domains back to the home chain (so conditional payments to executors can be unlocked). Rollups have nice trust-minimized properties regarding their ability to read state from Ethereum L1 and other rollups.

But here's a new idea - what about doing both! A restaked rollup.

This doesn't address all of the concerns mentioned by Hasu, but it's a very interesting new option nonetheless, especially in the longer term.

Instead of being secured only by EigenLayer restakers (more like its own L1 borrowing security), a rollup could use those EigenLayer restakers for its local consensus, but the chain then rolls up to Ethereum (i.e., it posts its data and proofs to it).

As discussed earlier, rollups can decide to implement their own local consensus for stronger assurances around short-term pre-confirmations and reorg resistance prior to true L1 finality. Well, why don't we just make that rollup consensus set a set of Ethereum restakers? Now you've got both:

- Pre-L1 Finality

- High value, decentralized restaked consensus providing strong short-term reorg resistance.

- Post-L1 Finality

- Still a rollup! It posts its data to Ethereum, and it can post ZK / fraud proofs.

This is the same security as any rollup once the L1 has finalized it. This is simply a variation of a rollup implementing its own consensus. Just instead of having its own validator set on its rollup secured by its own native token as described earlier, it opts to have Ethereum restakers running the consensus.

Similarly, it would be interesting if Celestia were to enshrine some form of restaking (currently, the plan is for Celestia's L1 to be incredibly minimal with no smart contract capability). I've written previously that enshrined rollups would be an interesting direction for Celestia, and native restaking would be another fascinating tool allowing the free market to decide what features they want to bolt on. Restaking could also help Celestia's security budget, as the value accrual model of alt-DA layers is still unproven.

Other MEV Considerations

The simplest idea here (what rollups generally do today) is to run a single sequencer with naive FCFS. The sequencer just commits to transactions in the order they receive them. You can reasonably trust a single sequencer to abide by FCFS ordering, but it still incentivizes latency races and related negative externalities among other issues (TLDR: don't implement PoW to try to fix it). So even centralized sequencers will need to contend with MEV more effectively.

Some of the solutions above implicitly deal with it (e.g., hosting a total free-for-all public auction), but obviously no rollups want to say their users are getting sandwiched. So, we probably need to get creative. Combining the benefits of an auction + leveraging programmable privacy is where it gets interesting.

That's why centralized sequencers also like having private mempools today - trying to make sure that users aren't getting frontrun etc. But that's not easy to decentralize. If you open up to many sequencers, do you just trust them to run their own private mempools? Even if you do, that could be problematic if each has their own private mempool and you need to start rotating the active leader around.

Encrypted mempools are one fascinating area of research in addressing many of these problems. I'll simply refer you to this post I just made for the relevant information:

Encrypted mempools as a tool for MEV & censorship

This post is largely built around @drakefjustin's great presentations on the topic (links in post)

They're pretty complicated (at least for me), so I tried to break them down simply then expand a bithttps://t.co/AYpljnkods

— Jon Charbonneau (@jon_charb) March 15, 2023

In this post, I'll just give a brief overview of some variations of FCFS proposals.

Time Boost

Realizing the limitations and negative externalities of simple FCFS, Arbitrum has started to explore new variations of it. Most recently, they proposed their Time Boost mechanism. The basics:

- Trusted centralized sequencer continues to operate a private mempool for user txs
- The sequencer timestamps every tx upon receipt
- By default, txs would be executed 500 ms after receipt
- Users can choose to pay an extra priority fee for their txs, giving it a "time boost" to reduce their timestamp by up to the 500 ms

The boost is computed by the following formula:

Where:

- F = tx's priority fee
- g = maximum time boost available (500 ms)
- c = a constant to be determined

Today, searchers can only express their preference for fast inclusion to the Arbitrum sequencer by trying to minimize latency to it (opening up many connections, getting physically close to it, infrastructure investment, etc.). Time Boost would instead reduce the incentive to invest in the above latency advantages, as searchers can now express their preference for fast inclusion via their priority fee bid.

It's a clear improvement over naive FCFS, but still has some shortcomings in my opinion:

- This reduces

the incentive to latency-race, but it doesn't remove

it. Minimizing latency will always allow a searcher to bid less than competitors here, and the curvature of the graph makes this latency edge particularly valuable for high-value MEV opportunities.

- Unclear how the mempool privacy and time-stamping could be efficiently decentralized beyond the single sequencer.
- Lack of expressivity in searchers' bidding. Searchers are only able to bid for faster inclusion, but they're unable to express more complex preferences as would be the case in a more flexible explicit auction (e.g., as in Ethereum L1 today).

- Related, there will be no ability for reverts of failed bids if implemented at this level.

FBA-FCFS

An alternative proposal (FBA-FCFS) for Arbitrum was made last year by Xin from Flashbots. This implements a frequent-batch auction style variation of Themis. In this proposal:

- Nodes report a partial ordering to the leader who then aggregates these into an unordered batch. Within the set batch time "fairness granularity" (e.g., could also be 500 ms), all transactions are assumed to have occurred at the same time.
- The leader then resolves the intra-batch weak ordering via some form of auction.

Similarly to the Time Boost proposal, there's some attempt to guarantee relatively fast inclusion (<500 ms proposed, but could be changed) and prevent front-running via trusted nodes. Again, it reduces the incentive to latency race (though doesn't completely remove all advantage) by allowing for searchers to express their preferences via fees.

There exists a remaining latency advantage for "fast searchers" at the end of a given batch where other "slow searchers" can't compete:

Source: Latency Arms Race Concerns in Blockspace Markets

This is a well understood aspect of these types of auctions. Longer batch times could further make that latency edge relevant a decreasing percentage of the time, though on potentially higher batch values. Longer block times can have other effects on MEV (e.g., staler prices).

Summarizing this latency edge:

- Time Boost

- Lower latency is always

advantageous (lower latency = can always have lower bid on average).

- FBA-FCFS

- Lower latency is sometimes

advantageous (when the small delta between "slow" and "fast" searchers reveals relevant information).

Regarding the time delay:

- Time Boost

- All user txs delayed by 500ms speed bump by default upon receipt.

- FBA-FCFS
- Batch time is 500ms, so user txs received during the window can be included within it, on average <500ms closer to the midpoint.

FBA-FCFS results in more variability (e.g., transactions may slip to the next batch depending on receipt time), but inclusion time for regular user transactions not paying priority fees should on average be lower. Time is treated continuously in the Time Boost model (there is no notion of a block), vs. time is discretized in the FBA-FCFS model. Overall:

Latency & Decentralization

This post is already long as hell, and I'm tired, so please just read this post by Phil. Thanks.

Conclusion

Hopefully you have a decent overview now of the paths toward sequencer decentralization and the associated hurdles. This is one of the many challenges that rollups will need to contend with as they mature in the coming years.

Some other interesting areas of exploration include:

- Interaction of Different Layers

- As sequencers decentralize, how do they interact with different parts of the transaction supply chain? What type of auction (e.g., SUAVE) should plug into a decentralized sequencer set (e.g., shared sequencers)?

- Transaction Fee Mechanisms

- Multi-dimensional resource pricing vs. state pricing

- L2 Fee Markets

- Charging for "L2 native" fees (e.g., L2 congestion) and "L1 native" fees (e.g., paying for L1 DA)

- ZK Prover Markets

- How to properly incentivize a decentralized and permissionless market of ZK provers. This is an important subset of L2 fees, and will also be relevant in other contexts.

- Rollup Business Models

- How rollup tokens and the teams building them may accrue value and achieve sustainability.

- Upgradeability and Governance

- Decentralizing these and reducing trust assumptions while maintaining flexibility.

- Fraud & ZK Proofs

- Building more robust mechanisms such asmulti-provers.