I'd like to make two claims:

1. No single chain, whether L1 or L2, can support the throughput required to reach Internet scale.
2. Scaling blockchains means scaling access to liquidity and shared state. Adding blockspace via multiple chains doesn't work if it fragments liquidity.

No single chain, whether L1 or L2, can support the throughput required to reach Internet scale.

Scaling blockchains means scaling access to liquidity and shared state. Adding blockspace via multiple chains doesn't work if it fragments liquidity.

This represents a challenge to both the modular and the monolithic views of blockchain scalability. (1) is a challenge to the monolithic view, which holds that a single high-throughput chain is the best way to scale. (2) is a challenge to the modular view, because it means that a multi-chain or multi-rollup ecosystem is not sufficient for scaling in a meaningful sense: increasing access to shared state and liquidity..

If (1) and (2) are true, then solving the scalability problem requires scaling access to shared state and liquidity across many chains. Polygon's solution is the Aggregation Layer, or "AggLayer." The AggLayer provides safety for near-instant cross-chain transactions and enables unified state and liquidity across chains.

This post will dive into what the AggLayer is, how it works, and how it's different from a shared sequencer or prover.

The Problem

There's a problem with L2s: liquidity and state are fragmented across rollups and the L1.

This is bad from a usability perspective because it introduces complexity, but it's also expensive. Fragmented liquidity means higher slippage and worse execution. Optimistic Rollups (ORs) require users to pay expensive third-party bridges to avoid the seven-day withdrawal delay. Even ZK Rollups (ZKRs) require users to round-trip to Ethereum for trustless cross-chain transactions.

Finality and Validity

Here's why low-latency, trustless cross-chain transactions aren't currently possible.

Suppose there are two rollups, Chain A and Chain B, that share a bridge to L1. Alice on Chain A would like to pay Bob on Chain B, so Alice locks or burns tokens on Chain A in order to transfer to Chain B.

Two things are required for Chain B to safely credit those tokens to Bob.

1. The batch containing Alice's transaction must be finalized on Ethereum L1.
2. Chain B must be able to verify that the resulting state of Chain A is valid after Alice's transaction.

The batch containing Alice's transaction must be finalized on Ethereum L1.

Chain B must be able to verify that the resulting state of Chain A is valid after Alice's transaction.

If the batch containing Alice's transactions isn't finalized on Ethereum, then Chain A could equivocate to Chain B and double-spend by keeping Alice's funds on Chain A and minting Bob's funds on Chain B. Likewise, if Chain B doesn't check a validity proof for A, then Chain A could include an invalid transaction and steal funds from B.

(1) and (2) mean that trustless cross-chain transactions can't have low-latency. (1) currently requires 12 minutes, while (2) requires waiting for the duration of the challenge period in ORs and a few minutes for proof generation on ZKRs.

Good UX is incompatible with 20-minute latency. The Aggregation Layer is designed to solve this problem.

The Aggregation Layer

Polygon is an ecosystem of ZK-powered L2s that settle to Ethereum. The Aggregation Layer is a decentralized protocol operated by staked nodes that ensures safety

for low-latency, cross-chain transactions and a unified bridge [1].

In this context, "safety" means the following:

It's impossible for a rollup's state to be finalized/settled on Ethereum if that chain state relies on an invalid or non-finalized state from another chain, or if it includes a transaction from an atomic [2] bundle that has not executed successfully on all other chains.

In other words, a state of Chain B cannot be finalized on Ethereum if it depends on an invalid or non-finalized state of Chain A.

This guarantee is important. It allows Chain B to safely interoperate with Chain A at super low latency, before the state of Chain A has finalized on Ethereum or a proof has been generated.

.css-ovd0pw{position:relative;height:100%;width:100%;}

.css-istob7{height:100%;width:100%;position:absolute;max-width:100%;max-height:100%;}

.css-juujg6{padding-bottom:49.926035502958584%;}

.css-ovd0pw{position:relative;height:100%;width:100%;}

.css-istob7{height:100%;width:100%;position:absolute;max-width:100%;max-height:100%;}

.css-juujg6{padding-bottom:49.926035502958584%;}

.css-istob7{height:100%;width:100%;position:absolute;max-width:100%;max-height:100%;}

.css-juujg6{padding-bottom:49.926035502958584%;}

How the Aggregation Layer works

The Aggregation Layer functions in three phases. Suppose that Chain A is a ZK-powered chain running in the Polygon ecosystem.

1. Pre-Confirmation:

Chain A submits a header for a new block/batch A

1

A_1

A

1

to the AggLayer along with a light client proof. The header includes commitments to all other blocks and bundles that A

1

A_1

A

1

depends on (B

i

B_i

B

i

, C

i

C_i

C

i

, etc). When the new batch is accepted without a validity proof, it’s considered “pre-confirmed” by the AggLayer.

1. Confirmation:

Chain A, or any full node of A, generates a proof for A

1

A_1

A

1

and submits it to the AggLayer. Once the proof is verified by the AggLayer, A

1

A_1

A

1

is confirmed if all batches that it depends on are also confirmed.

1. Finalization:

After A

1

A_1

A

1

is confirmed, its proof is aggregated alongside batches from other rollups into a single proof that is posted to Ethereum. The aggregated proof enforces that dependent chain states and bundles are consistent.

Pre-Confirmation:

Chain A submits a header for a new block/batch A

1

A_1

A

1

to the AggLayer along with a light client proof. The header includes commitments to all other blocks and bundles that A

1

A_1

A

1

depends on (B

i

B_i

B

i

, C

i

C_i

C

i

, etc). When the new batch is accepted without a validity proof, it’s considered “pre-confirmed” by the AggLayer.

Confirmation:

Chain A, or any full node of A, generates a proof for A

1

A_1

A

1

and submits it to the AggLayer. Once the proof is verified by the AggLayer, A

1

A_1

A

1

is confirmed if all batches that it depends on are also confirmed.

Finalization:

After A

1

A_1

A

1

is confirmed, its proof is aggregated alongside batches from other rollups into a single proof that is posted to Ethereum. The aggregated proof enforces that dependent chain states and bundles are consistent.

Chains can navigate the tradeoff space between latency and liveness guarantees for themselves. A chain might choose to

interoperate with another chain after the pre-confirmation step for super low-latency cross-chain transactions, but fundamentally, this model is compatible with chains waiting for confirmation, or even for finalization.

The safety guarantee for cross-chain transactions is enforced at the third step. Let’s dig further into how this design enables safe cross-chain interaction.

Asynchronous Interoperability

Take the first example of a cross-chain transfer. Alice on Chain A wants to lock or burn some tokens in block A

1

A_1

A

1

in order to mint and transfer tokens to Bob on Chain B. If Chain B doesn’t wait until A

1

A_1

A

1

is finalized on Ethereum with a valid proof, then Chain A could equivocate or give Chain B an invalid state.

The Aggregation Layer solves this in a simple way. Chain B can temporarily assume that A

1

A_1

A

1

is valid and will be finalized on Ethereum, without even waiting for a proof. The sequencer for Chain B commits to the claimed Chain A state root A

1

A_1

A

1

as a dependency in the header for B

1

B_1

B

1

(as B

1

A

1

B^{A_1}_1

B

1

A

1

) before submitting to the Aggregation Layer. The latency required for Chain B to build B

1

B_1

B

1

decreases from 20 minutes to, at most, a few seconds.

In the Confirmation step, the Aggregation Layer builds a dependency graph for each block/batch submitted. For instance, if A

1

A_1

A

1

depends on B

1

B_1

B

1

, which in turn depends on C

1

C_1

C

1

, C

1

C_1

C

1

is confirmed as soon as a proof π

C

1

\pi_{C_1}

π

C

1

is submitted. But, even if π

A

1

\pi_{A_1}

π

A

1

is received, A

1

A_1

A

1

is only confirmed with both π

B

1

\pi_{B_1}

π

B

1

and π

C

1

\pi_{C_1}

π

C

1

.The critical aspect of this design is that the proof aggregation circuit enforces consistency across dependencies. If B

1

A

1

′

B^{A^\prime_1}_1

B

1

A

1

′

is inconsistent with the block A

1

A_1

A

1

that Chain A submits, or a proof is missing for A

1

′

A^\prime_{1}

A

1

′

, then B

1

B_1

B

1

cannot be included in the aggregated batch finalized on Ethereum.

This mechanism guarantees that if Chain A equivocates or submits an invalid block, say A

1

′

A^\prime_1

A

1

′

, then any batch that depends on an invalid or equivocated state root for Chain A cannot be finalized/settled on Ethereum. Even if the AggLayer itself equivocates, chains have a cryptographic guarantee that any block that depends on an invalid or equivocated block cannot be finalized, because two proofs for chain states that are inconsistent or invalid cannot be aggregated together in the proof aggregation circuit. This ensures that the safety property described above is preserved.

Atomic Interoperability

The safety mechanism can be extended to the atomic case. Suppose that a user submits an atomic bundle of transactions to multiple chains. This bundle is ordered, so the result of executing the transaction on Chain A is passed to Chain B, and likewise Chain B's updated state is passed to Chain C, etc. If all transactions execute successfully across all chains, then the bundle is included; otherwise, it's rejected.

It would be ideal to provide the ability to include atomic transactions without:

1. Requiring the operator of Chain B to run a full node for all other chains included in a bundle; or
2. Accepting the risk that the bundle might be partially included on Ethereum (harming participating chains).

Requiring the operator of Chain B to run a full node for all other chains included in a bundle; or

Accepting the risk that the bundle might be partially included on Ethereum (harming participating chains).

This raises a similar safety issue as in the asynchronous case: Chain A might equivocate and submit a batch that doesn't actually include the atomic bundle, or send an invalid result to Chain B.

Fortunately, the same mechanism from the async case can be reused for the atomic case. Chain B commits to bundles and received transaction results from other chains. The Aggregation Layer (and proof aggregation circuit) checks that bundles are consistent across chains. A batch containing a bundle from Chain B can only be finalized/settled on Ethereum if all transactions in the bundle are executed successfully.

Cross-Chain Composability

The Aggregation Layer enables super low-latency cross-chain composability through asynchronous cross-chain calls. This is an incredibly powerful primitive: contracts can safely call contracts on other chains at super low latency, without waiting for Ethereum finality. A user could onramp via the OKX chain on Polygon and immediately deposit into a highly-liquid lending market on Aave on a different chain in one click, without needing to swap out of a wrapped synthetic asset.

Emergent Coordination Infrastructure

The AggregationLayer guarantees that near-instant cross-chain interactions will be safe [3]. But this is only half the battle. How do chain operators share and trust each other's chain states? How do they coordinate the production of atomic bundles?

A design goal for the AggLayer is that it should be minimal. Its purpose is to guarantee safety and provide a foundation that allows anyone to build coordination infrastructure that offers liveness in a variety of different settings.

The operators of chains can freely choose between emergent coordination mechanisms depending on their trust assumptions - these could include relays, shared prover infrastructure, or shared validity sequencer [4] clusters. These protect chains against liveness issues when depending on other chain states or bundles.

The Polygon ecosystem prioritizes choice and sovereignty for chains. Chains can run their own modified execution environments, use their own tokens for staking and gas fees, choose their own data availability mechanisms, etc. Similarly, chains should decide how to handle tradeoffs between interoperability and the risk of liveness faults. There are several options:

1. Chain B can opt out of fast interoperability and the aggregator layer entirely. It can simply submit batches and proofs directly to Ethereum and finalization is never delayed.
2. Chain B can accept Chain A's state only when Chain A's state is confirmed by the AggLayer. Chain B will be delayed only if the AggLayer equivocates.
3. Chain B can accept Chain A's state when Chain A is pre-confirmed by the AggLayer. Chain B will be delayed if the AggLayer equivocates or Chain A fails to produce a proof.
4. Chain B can accept Chain A's state in a peer-to-peer setting, without checking that Chain A is pre-confirmed on the AggLayer. Chain B will be delayed if Chain A equivocates or fails to produce a proof.

Chain B can opt out of fast interoperability and the aggregator layer entirely. It can simply submit batches and proofs directly to Ethereum and finalization is never delayed.

Chain B can accept Chain A's state only when Chain A's state is confirmed by the AggLayer. Chain B will be delayed only if the AggLayer equivocates.

Chain B can accept Chain A's state when Chain A is pre-confirmed by the AggLayer. Chain B will be delayed if the AggLayer equivocates or Chain A fails to produce a proof.

Chain B can accept Chain A's state in a peer-to-peer setting, without checking that Chain A is pre-confirmed on the AggLayer. Chain B will be delayed if Chain A equivocates or fails to produce a proof.

An important thing to note is that users cannot cause liveness faults, only misbehaving or malfunctioning chains. Equivocation and the submission of an invalid block can be heavily penalized, either via slashing or by ejecting chains from the AggLayer and precluding their ability to seamlessly interoperate. Therefore, a liveness fault should be extremely rare.

Chains can take additional precautions to minimize the risk of liveness issues, by maintaining white- or blacklists of other chains with which they interoperate and setting limits on the number of chains that can be collectively involved in any batch. They could rely on third parties running full nodes to ensure that if a chain goes offline before it can produce a proof, there's a backup prover.

The mechanism by which chains coordinate to accept atomic bundles is also flexible. For instance, a subset of chains could interoperate in a shared validity sequencing cluster for extremely low latency, or they could rely on relays.

A cryptoeconomically-secured relayer could enable interoperability between Chains A and B by running a full node for both chains, and attesting that states from each chain are valid. Even if Chain A or B pre-confirms a new batch and then goes offline, shared prover infrastructure can step in to generate a proof.

You can imagine novel coordination infrastructure emerging on top of the foundation of safety provided by the AggLayer, enabling new and better forms of interoperability and shared liquidity. Crucially, the entire Polygon ecosystem does not need to share the same infrastructure or trust assumptions. It doesn't need to operate under a single shared validity sequencer or prover. This is an extremely important advantage relative to ORs.

Closing

The Aggregation Layer fundamentally allows us to create a multi-chain ecosystem that feels like using a single chain. It's the synthesis of the monolithic and modular theses: unified state, liquidity, and composability, with the unbounded scalability of a multi-chain ecosystem.

Aggregation in ZK vs Optimistic Systems

This is a vision that is fundamentally only available to ZK-based systems. I'll expand on this point in a future post, but Optimistic ecosystems that wish to enable fast interoperability must rely on shared validity sequencers. This is a bad deal for chains: it restricts them from redistributing sequencer fees and MEV, shared validity sequencers force chains to potentially accept restrictions on their execution environments, and interoperability in OR-based systems force chains to accept additional trust assumptions in exchange for low latency.

Further, cross-chain interoperability breaks an important property for ORs. With single-chain ORs, anyone can run a full node for an OR and immediately confirm that transactions are valid and finalized as soon as they're posted to L1s. This is no longer true in the multi-chain case - now it's necessary to run a full node for every chain with which the OR interoperates.

By contrast, Polygon's vision is one where chains are sovereign. They can use any execution environment, can rely on any centralized or decentralized sequencers, and can navigate tradeoffs between cross-chain latency and liveness for themselves.

This is a vision that mirrors the existing Internet. The Internet is an elastically scalable, permissionless, and unified environment. Likewise, the AggLayer is scalable and permissionless - it imposes no restrictions on participating chains - and allows users to move assets and state seamlessly across the ecosystem, presenting a unified interface for the value layer of the Internet.

This is the future of Polygon: not monolithic, not fully modular, but aggregated.

[1] Part of ensuring unified liquidity is getting rid of the terrible UX of wrapped synthetic tokens on bridges. Users of Polygon's LxLy bridge can seamlessly transfer assets across chains while preserving fungibility. However, in order to do this safely, we need to protect against weakest-link security - or an attacker corrupting a single chain and draining all funds across all chains in the bridge. I'll discuss how to do this in a future post, but the AggLayer can leverage the proof aggregation step to enforce chain-level accounting, avoiding weakest-link security.

[2] When I reference atomic cross-chain transactions, what I mean is the ability for a user to submit a "bundle" or set of transactions across multiple chains. The atomic bundle has the property that its transactions are included in each relevant chain if and only if all transactions execute successfully. If a single transaction fails, then the bundle cannot be included on any chain.

The most basic example is again our cross-chain transfer. Let's say that Alice wants to send 1 ETH to Bob, but Alice is on Chain A, and Bob is on Chain B. Assuming a shared native bridge for both rollups, Alice can burn her ETH on Chain A and mint ETH on Chain B which is transferred to Bob. But it's critical to guarantee that she can't mint ETH without burning it or vice-versa - either she could lose her ETH or under-collateralize the bridge.

This is why atomic transactions are so important. In order to allow low-latency interactions between chains and make using the Polygon ecosystem feel

like using a single chain, atomic guarantees are needed.

[3] This is a subtle point, but from the perspective of the ecosystem - the AggLayer provides safety, but from the perspective of a single chain, this design prioritizes liveness over safety, as Chain B can depend on a chain state from Chain A that is invalid. In this case, Chain B will not be accepted by the AggLayer (enforced by the proof aggregation circuit) and will need to build a new block without the dependency on A.

[4] Our approach as a whole owes a lot to theShared Validity Sequencing design from Umbra Research.

2/9/24 - Updated this draft to clarify some comparisons between aggregation and shared sequencing. The aggregated thesis depends on mechanisms like shared sequencers, relays, and builders to facilitate coordination between chains. The agg layer in turn guarantees safety.