Lesson 9

Week 3

Lesson 9 - What's next in L2 part 2 : L3s / Hyperchains
Lesson 10 - Privacy in Layer 2
Lesson 11 - What are ZK EVMs part 1 - overview
Lesson 12- What are ZK EVMs part 2 - universal circuits/circuit compiler

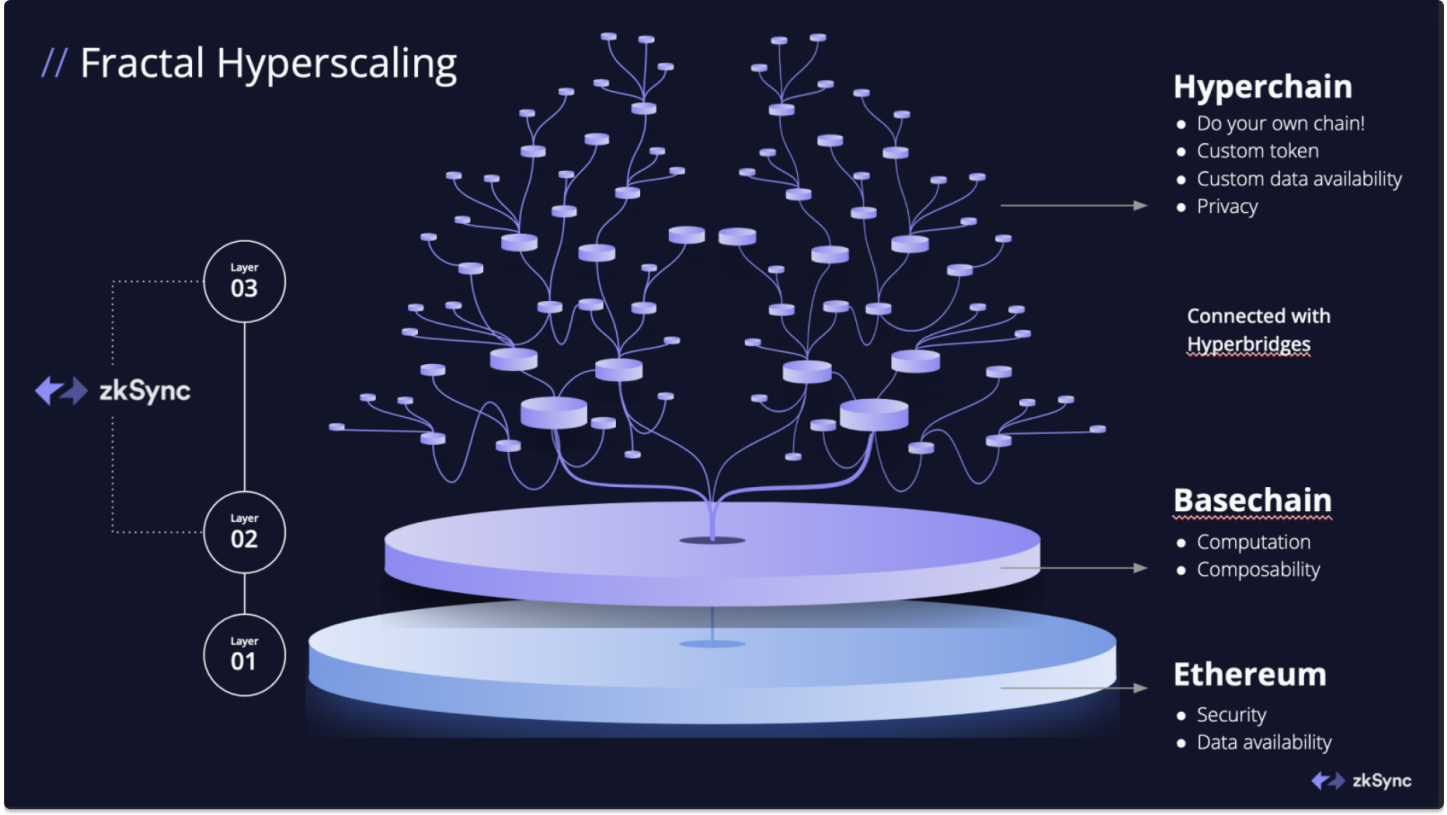
Fractal Scaling

See article by Vitalik and Polynya
See article by Mix Marvel

Recall rollups focus on 2 areas to give scalability

- Execution - using proofs
- Data - compressing data

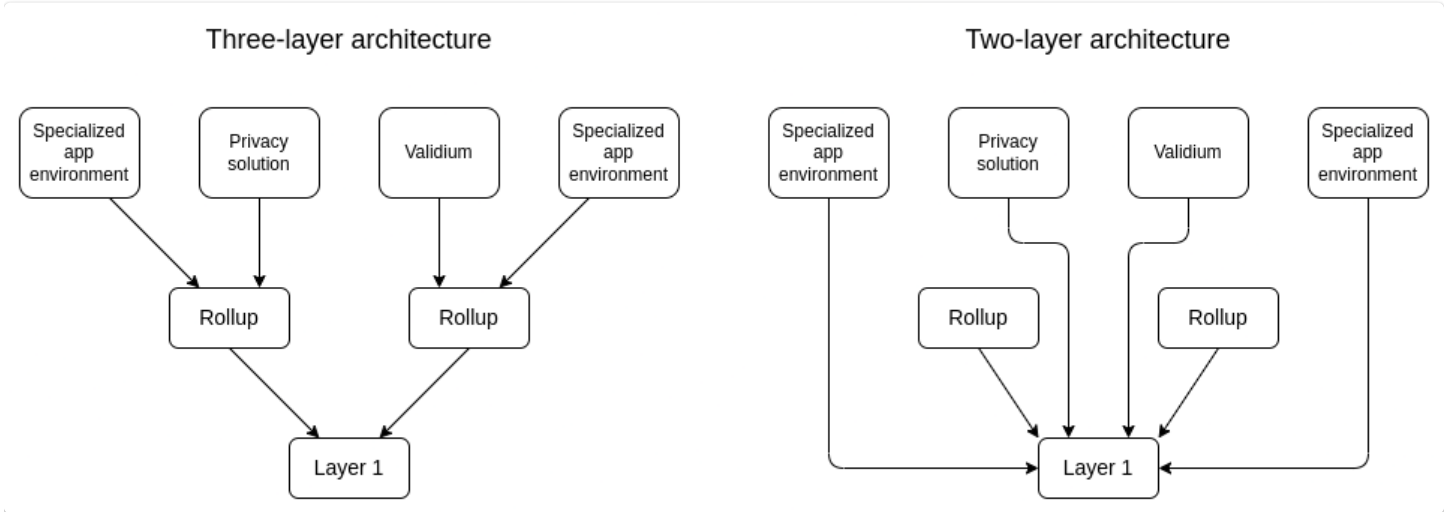
You may think that we could keep adding extra rollup layers ad infinitum to get more and more benefits.
A simple approach, such as that is unlikely to work as some of the techniques can only work once, such as compressing data, we cannot go on compressing data down to one bit for example.



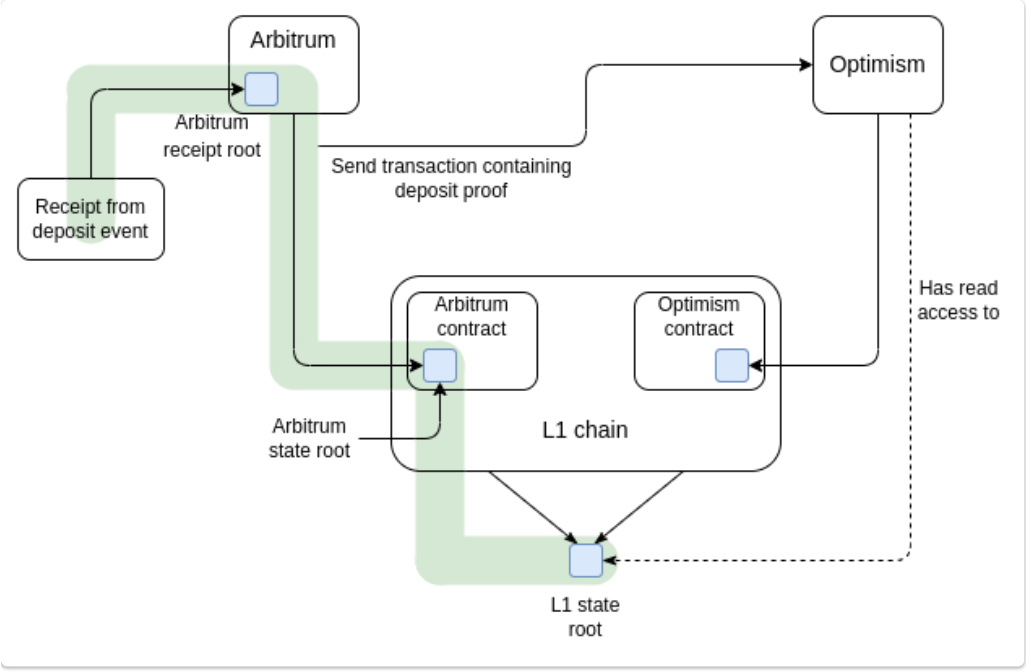
We get 3 possibilities

1. L2 is for scaling, L3 is for customized functionality, for example privacy.
2. L2 is for general-purpose scaling, L3 is for customized scaling.
3. L2 is for trustless scaling (rollups), L3 is for weakly-trusted scaling (validiums).

This approach is quite popular, but Vitalik outlines other ideas in his article



As far as tokens go, there are further possibilities, if we accept that a token does not have to be based on the L1 then we can manipulate it across L2s by wrapping it, and needn't make an L1 transaction.



There is debate around how we define a 'layer 2' in his article, Vitalik proposes the following criteria

- Their purpose is to increase scalability
- They follow the "blockchain within a blockchain" pattern: they have their own mechanism for processing transactions and their own internal state
- They inherit the full security of the Ethereum chain

It should be noted that the same requirements that we have for an L2 will also apply to an L3 design, including the need for data availability.

Data availability review

Data Availability overview

See EF Docs

In order to re create the state, transaction data is needed, the data availability question is where this data is stored and how to make sure it is available to the participants in the system.

	Validity Proofs		Fault Proofs
Data On-Chain	Volition	ZK-Rollup	Optimistic Rollup
Data Off-Chain		Validium	Plasma

STARKWARE

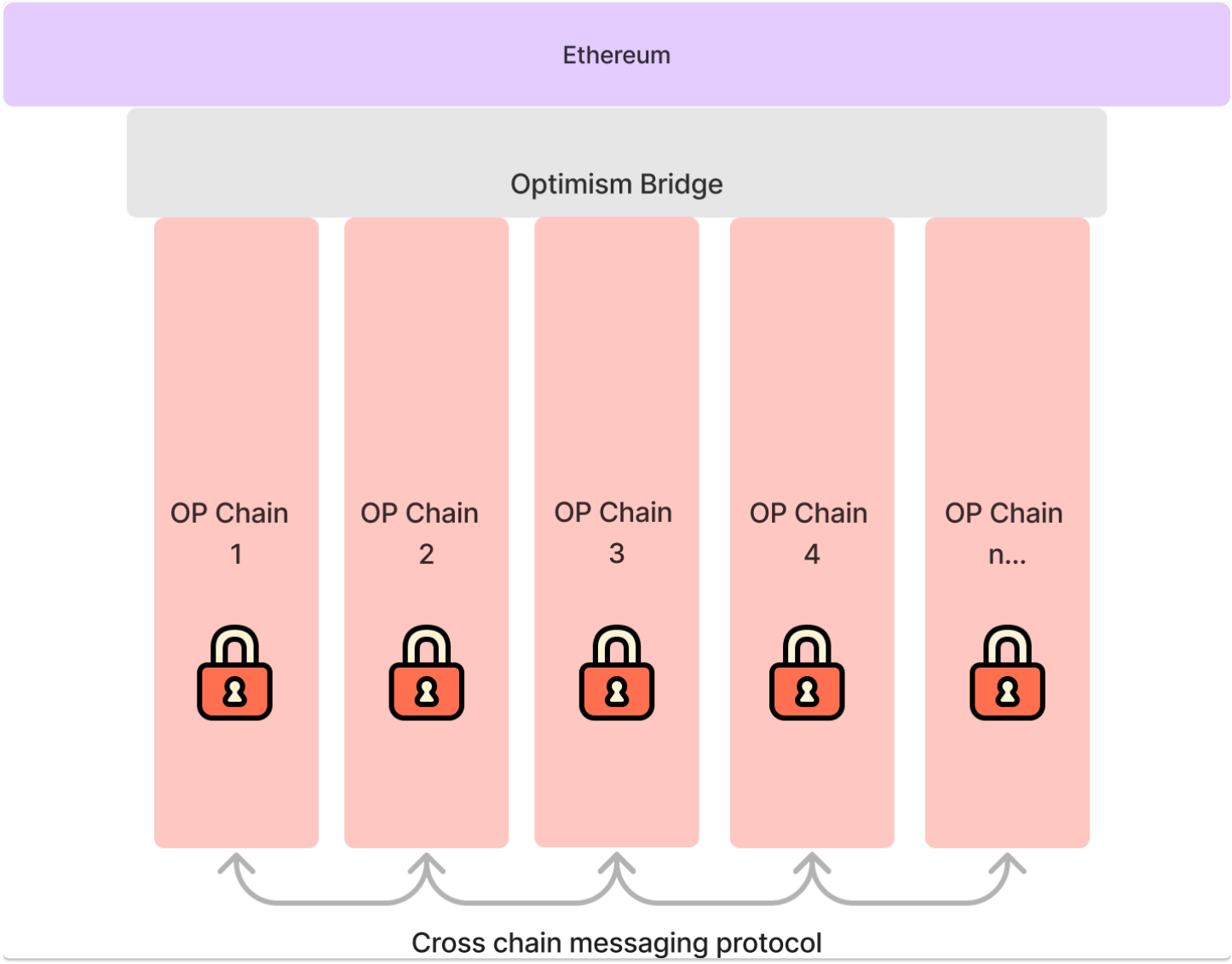
Validium solutions

Validium solutions use off-chain data availability and computation. Validity proofs are used to show the correctness of execution.

Communication between L2 / L3s

Optimism Superchain

This is a proposed network of L2s that share security, communication layers, and a common development stack - the OP Stack.



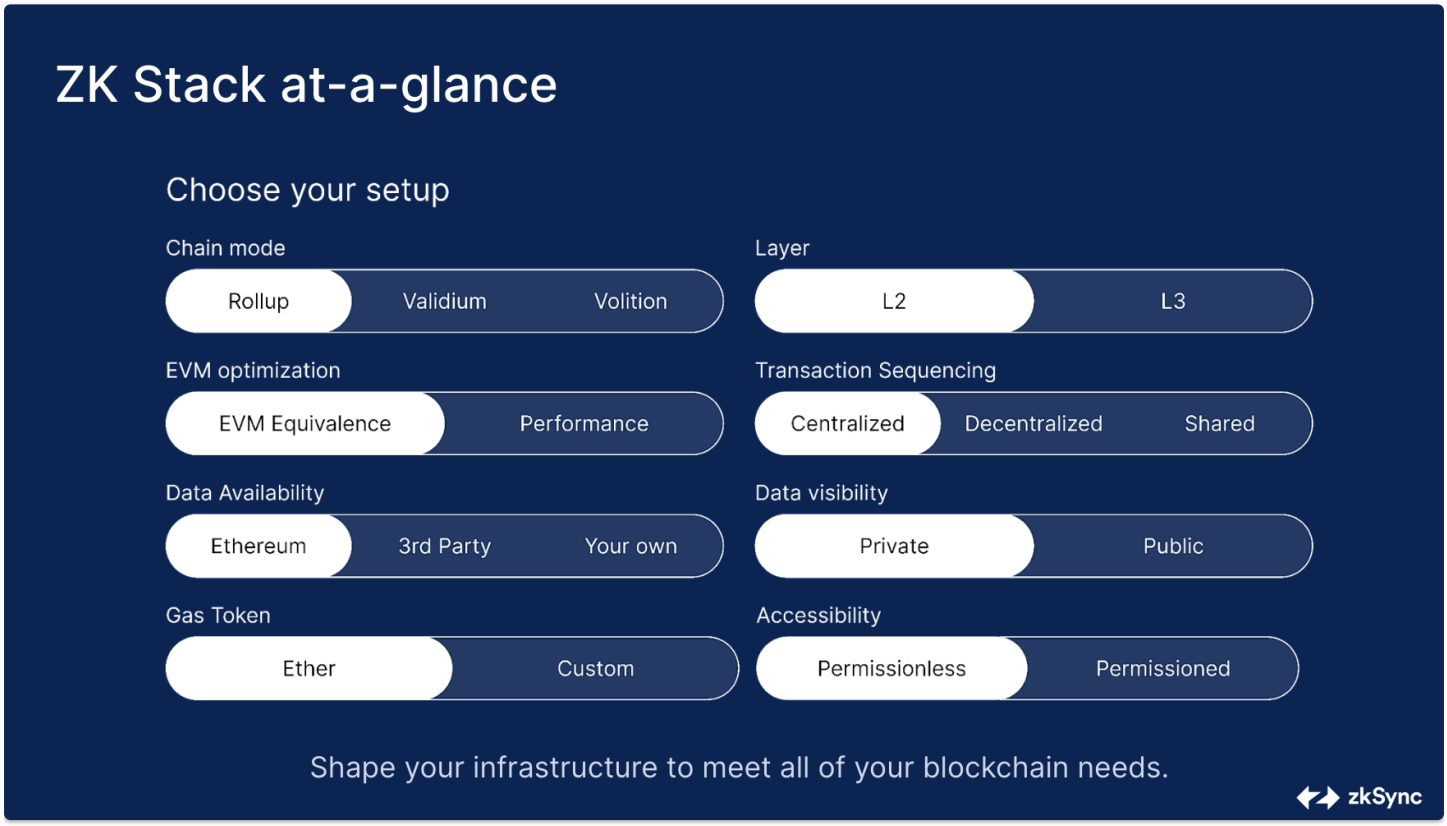
ZKStack

See article
See presentation

The ZK Stack is a modular, open-source framework that is both free and designed to build custom validity proof secured L2s and L3s.

The aim of the ZK Stack is to create a hyperscalable unified liquidity network.

ZKStack summary



The L2s built with ZK Stack are called hyperchains.

Comparison with Madera / OPStack

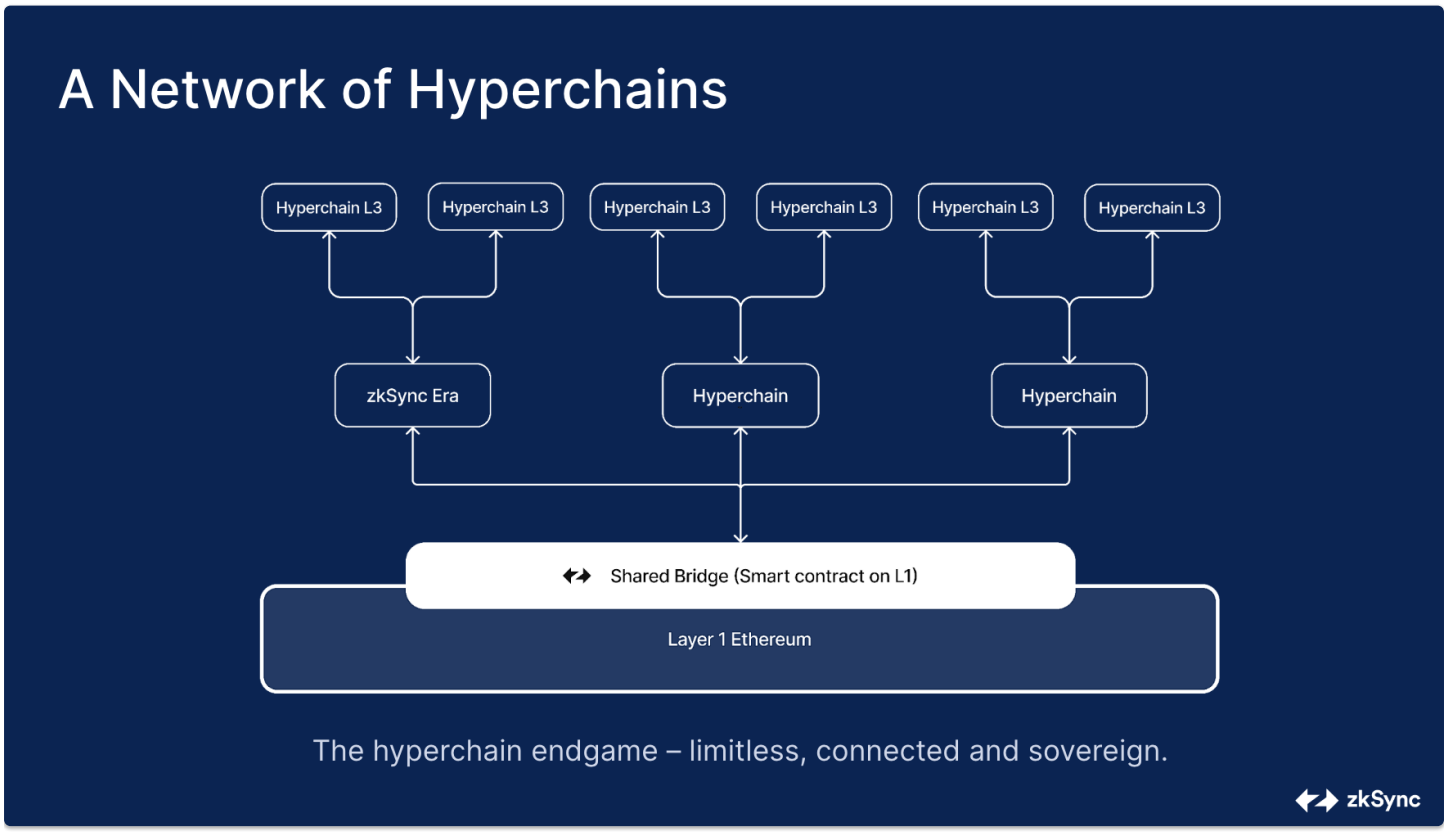
Madara from Starkware is more focussed on providing a sequencer that can be used for L3s
OP stack - it is harder to have interoperability
ZK stack has the advantage that the prover has lower memory requirements.

All of these try to solve the fragmentation of liquidity, but the chains involved are still distinct in that they don't for example share accounts

Hyperchains

From the zkSync documentation

See Introduction



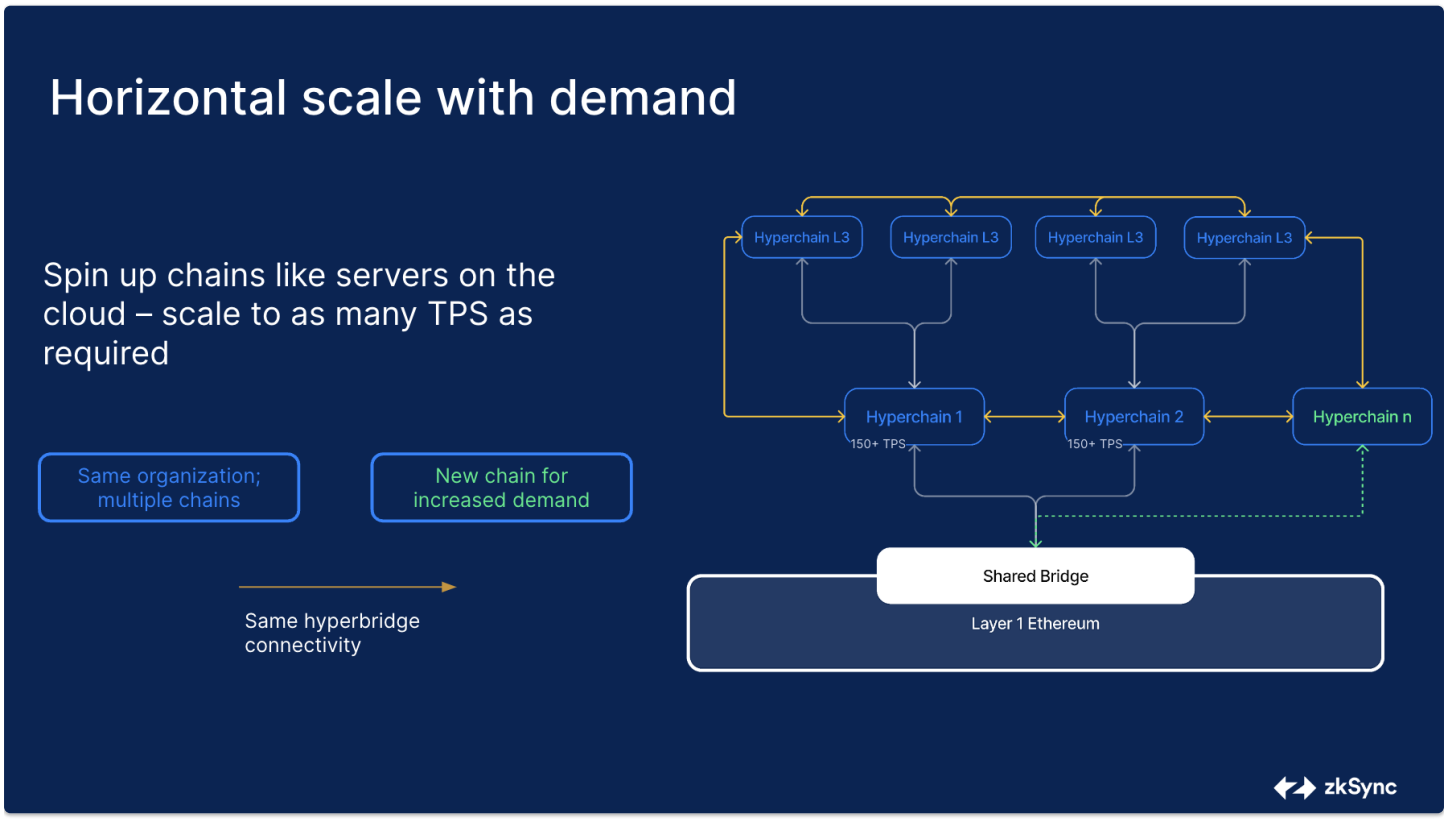
Using Hyperchains with a shared bridge contract on L1, and native Hyperbridges between the rollups solves a lot of problems in other architectures.

1. Rollups have validating bridges that are trustless.
2. Hyperbridges can easily burn and mint assets for transfers between members of the ecosystem.
3. The L1 serves as a single source of truth, so the rollups cannot hard fork.
4. The ecosystem can coordinate a hard fork together in case a vulnerability is found using a governance framework on L1, similar to how the L1 would react to a vulnerability.

Hyperchains can be developed and permissionlessly deployed by anyone. However, to remain trusted and fully interoperable, each Hyperchain must be powered by the same zkEVM engine available on the ZK Stack (and currently powering the first hyperchain, zkSync Era). All the ZKP circuits

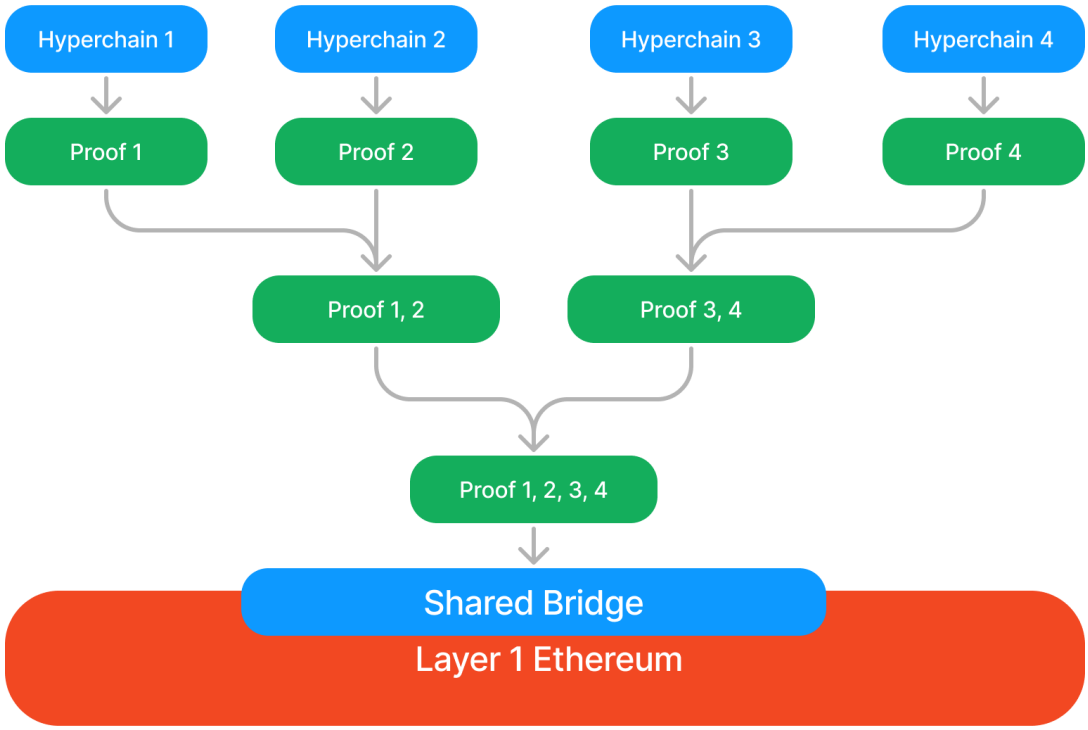
will thus remain 100% identical, letting Hyperchains fully inherit their security from L1, no matter who deployed them. This ensures zero additional trust/security assumptions.

Hyperchains will be implemented following the modular approach – using the ZK Stack developers can individually pick different components of their blockchains or implement their own ones (except the zkEVM core, for the reasons explained above).



Aggregating proofs

Simple proof aggregation treats the proofs of different Hyperchains as independent statements that can be verified together on L1. Unfortunately, the simple aggregation mechanism does not allow fast messaging as proofs are settled infrequently on L1 to save gas fees.



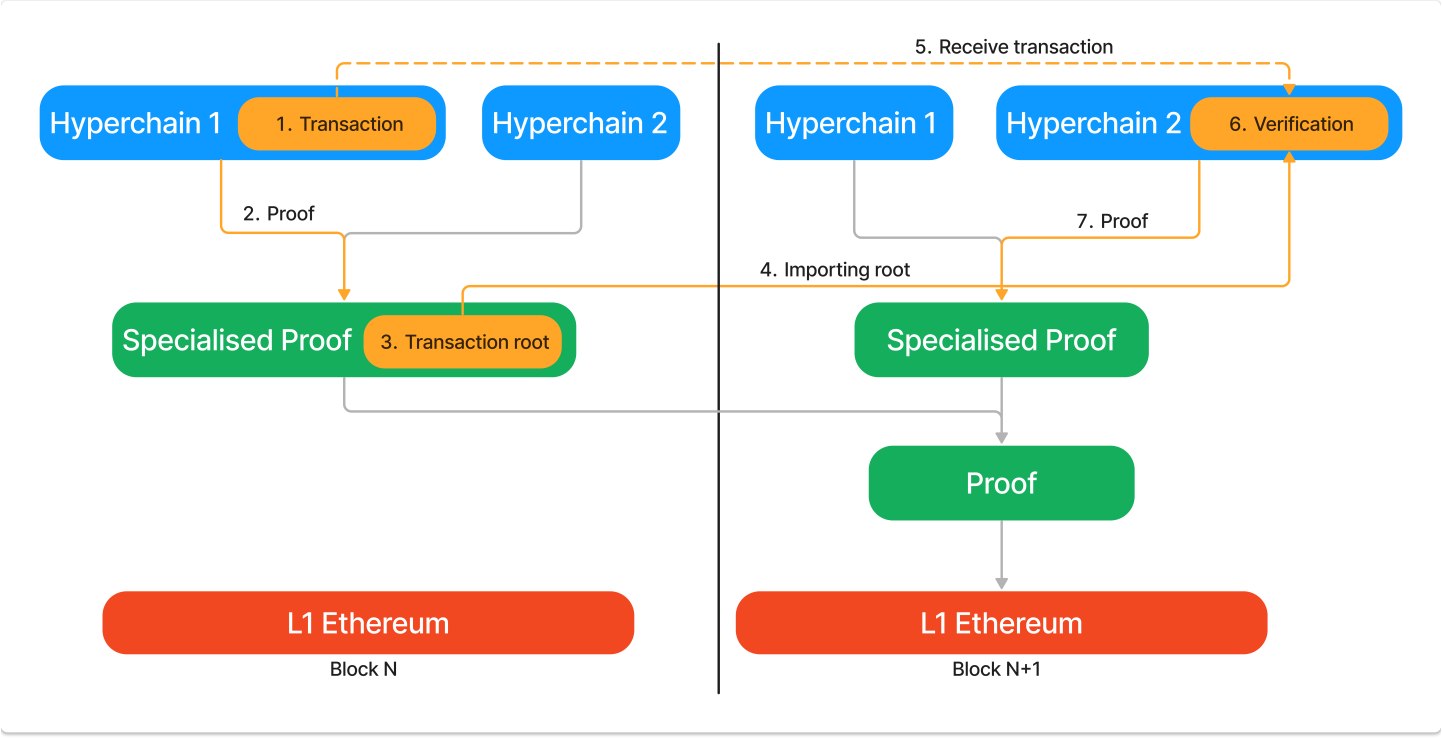
An Alternative to this is to use layering

Hyperchains can settle their proof on an L2 Hyperchain, becoming L3s. L3s settling on the same L2 will have faster messaging between each other and will have cheap atomicity via transactions forced through the L2, and interoperability will be preserved with the wider ecosystem. This is a particularly good solution for Validiums, as they don't send data to the L1. The only downside is that there is a higher chance of reversion if the L2 reverts.

Here proof aggregation happens via the L2, as the proofs of different L2 blocks are aggregated when settling on L1. This method is ultimately not scalable, as the L2's VM will be a bottleneck for proof verification. The L2's VM will also require a full consensus mechanism, meaning long-term storage, transaction verification, etc.

Layered Aggregation

Layered Aggregation combines the benefits of L3s with the benefits of simple aggregation. The L2's VM is replaced by the minimal program required to run L3 with messaging, and this is proven in a specialized proof that allows aggregation. This program tracks the State Root of the participating rollups, as well as the Transaction Root. The Transaction Root will be imported from and settled inside this specialized proof. Compared to the L2's VM this solution is more scalable, and will only need a lightweight consensus mechanism.



Comparing what the different aggregator mechanisms enable.

	Aggregation	L3s	Layered Aggregation
Fast Messaging	No	Yes	Yes
Scales	Yes	No	Yes
Consensus Mechanism	None	L2 Full Consensus	Lightweight Consensus
Instant Messaging Add-on	No	Yes	Yes
Sovereign	Yes	Yes	Yes

Customisation

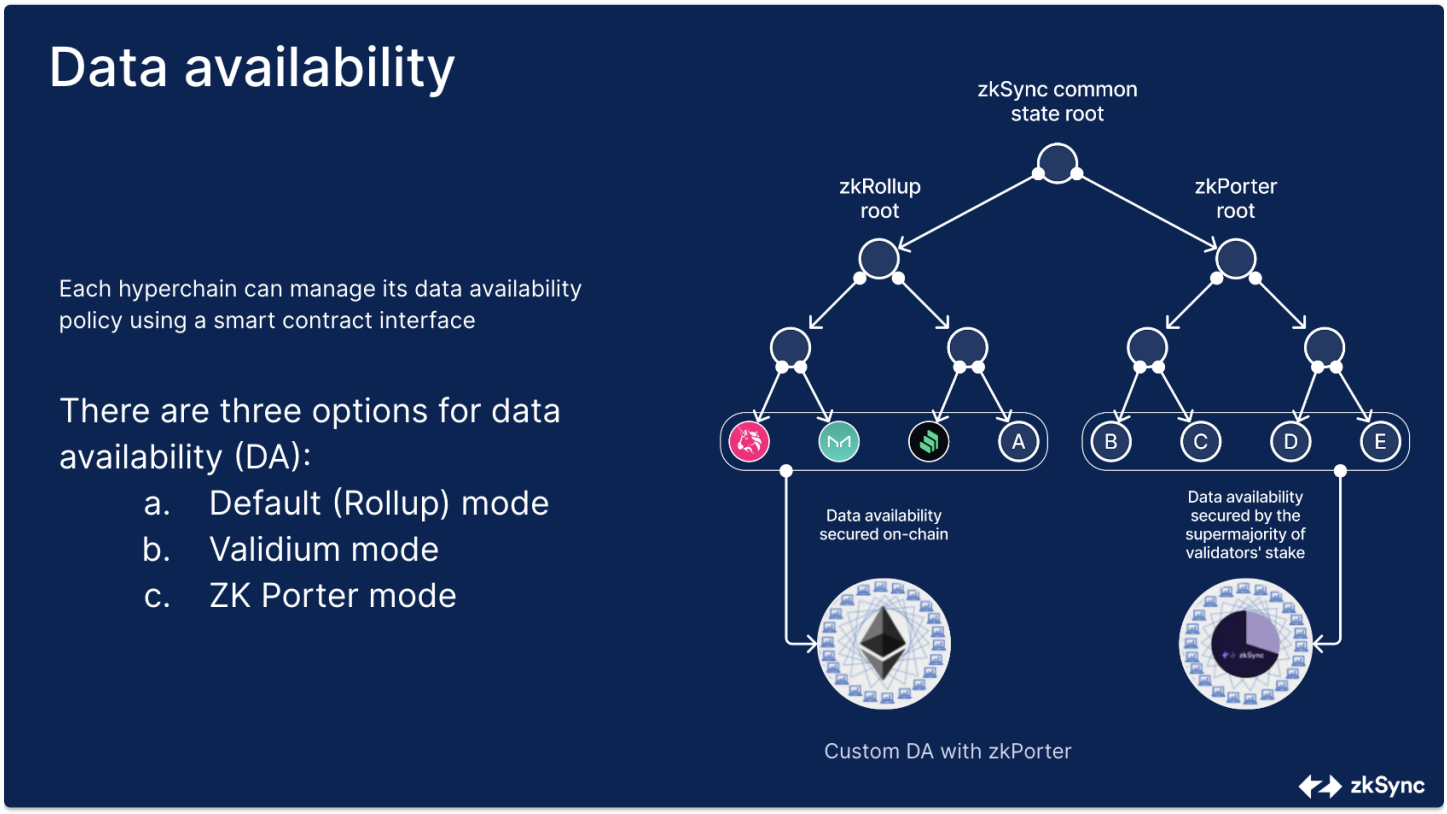
The main customisation options to be provided by ZK Stack are explained below.

Sequencing transactions

- **Centralized sequencer** - In this mode, there will be a single centralized operator with a conventional REST API to accept transactions from users. The operator must be trusted to maintain liveness, not to abuse MEV, and not to allow reorgs of unfinalized transactions, so the operator's reputation will play a big role. The biggest advantage of this option is that it can provide the lowest possible latency to confirm transactions (<100ms), which is critical for use cases such as HFT.
  ZkSync Era will run in this mode until it is fully decentralised.
- **Decentralised sequencer** - In this mode, a Hyperchain will coordinate on what transactions are included in a block using a consensus algorithm.
  It can be any algorithm, so developers can reuse existing implementations (e.g. Tendermint or HotStuff with permissionless dPoS). But we can also take advantage of the fact that finality checkpoints are guaranteed by the underlying L1, and implement an algorithm that is simpler and boasts higher performance.
  ZkSync Era will switch to this option as soon as the consensus implementation is ready and will make its code available to the Hyperchain developers.
- **Priority queue** - This simply means the absence of any sequencer: all transactions can be submitted in batches via the priority queue from an underlying L2 or even L1 chain, taking advantage of their stronger censorship resistance.
  It's worth noting that the priority queue will always be available as an escape-hatch mechanism to protect users against censorship by a malicious sequencer.
- **External protocol** - The sequencing of the Hyperchain is freely customisable, so external protocols such as Shared Sequencers and

Shared Builders can also be used.

Data availability



Each Hyperchain can manage its own data availability policy using a smart contract interface.
It can use one of the options described below or some more complex logic.
For example, to combine zkPorter and validium, the DA will require both a quorum of the signatures from the guardians and a number of signatures from the data availability committee.

Policy Choices

- zkRollup
  This is the default recommendation policy: the values of every changed storage slot at the end of the block must be published as calldata on L1. Note that repeated changes (or back-and-forth changes that result in no net difference) are not posted.
  It means if a batch contains 100 ETH/DAI swaps on the same DEX then pubdata costs will be partially amortized over all such swaps.
  A Hyperchain working in this mode strictly inherits full security and censorship-resistance properties from Ethereum. The implementation

of zkRollup in output mode is already available in zkSync Era and the ZK Stack.

- zkPorter

  See this post.

  There is already have a working zkPorter guardian testnet, which we are preparing to open source.

  zkPorter reduces transactions fees, though we have changed security guarantees compared to using L1 for data availability.

  Developers have the possibility of creating their own guardian network.

  This would make most sense in the context of existing communities and governance mechanisms.

- Validium

  Since validium is essentially a simpler case of a zkPorter, developers can easily deploy Hyperchains based on this policy.

- Based zkRollup

  This policy will require publishing full transaction inputs instead of final storage updates. Trustless state reconstruction and the data availability costs in this case will be 100% identical to optimistic rollups (but with all the benefits of a zkRollup which are increased security and faster exits).

  The implementation of this option is easily derived from the implementation of the normal zkRollup. It can be explored by application-specific chains where transaction inputs are short but might lead to a lot of changes in data (for example, performing financial simulations).

- zkRollup (self-hosted)

  In this mode, users self-host the data for all the accounts they own. To enforce this, user confirmation signatures are required to make any changes – which means, you cannot send funds directly to another user. Instead, you will burn the funds and create a proof of this burn, which you can provide to your recipient via an off-chain channel. The

recipient will then redeem them to their account. The complexities here could be taken care of with a simple UI, so that for the users this would be indistinguishable from sending and receiving funds on Ethereum.

In addition a self-hosted zkRollup can need as little as 5 bytes per user interaction.

This makes sharded Ethereum infinitely scalable for any practical purposes in the zkRollup mode (i.e. 100% secure and censorship-resistant). This is a way to onboard every single person on Earth to Ethereum with zero security compromise.

A great thing about this approach is also that it's fully compatible with the zkSync zkEVM implementation, but can nonetheless offer privacy to the users. The implementation is non-trivial,so may be less popular than other options.

At the same time, it's simpler and much more powerful than alternative approaches like Adamantium.

Logical state partitions in ZK Porters

Each Hyperchain can have one or more logical partitions that are part of the same state but live in separate subtrees and enforce different data availability policies, which can however interoperate synchronously.

Synchronicity is important as it enables atomic transactions between partitions, unlocking several unique use cases:

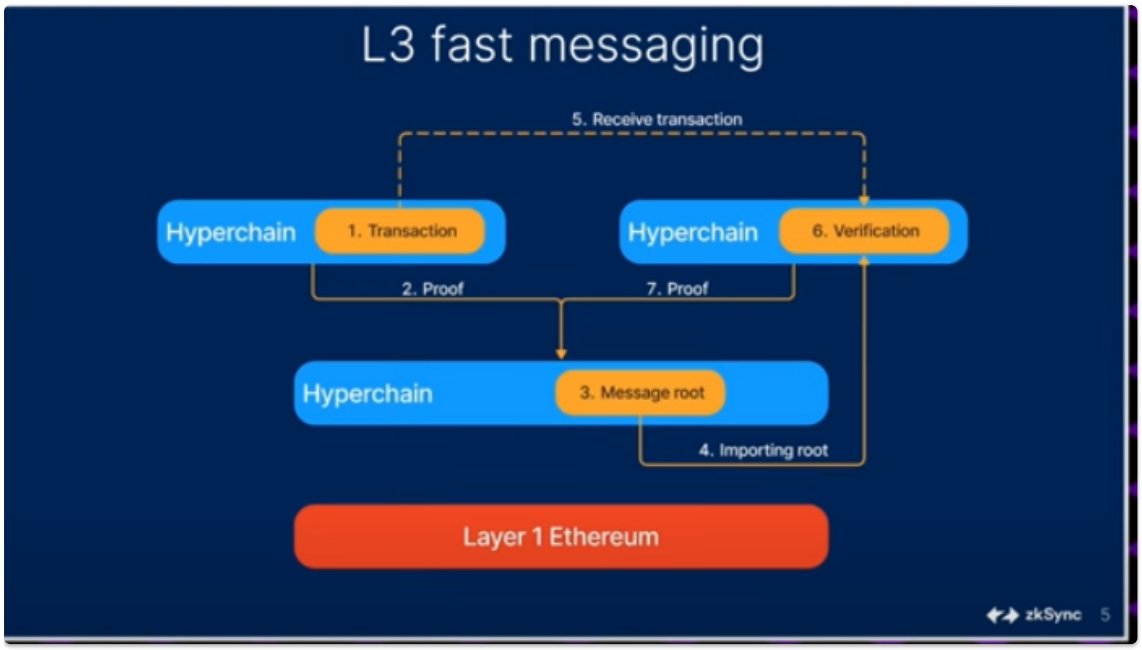
- Transparently reading the state of another partition.
- Using flash loans between the partitions.

One prominent example of this is a combination of [zkRollup + zkPorter]

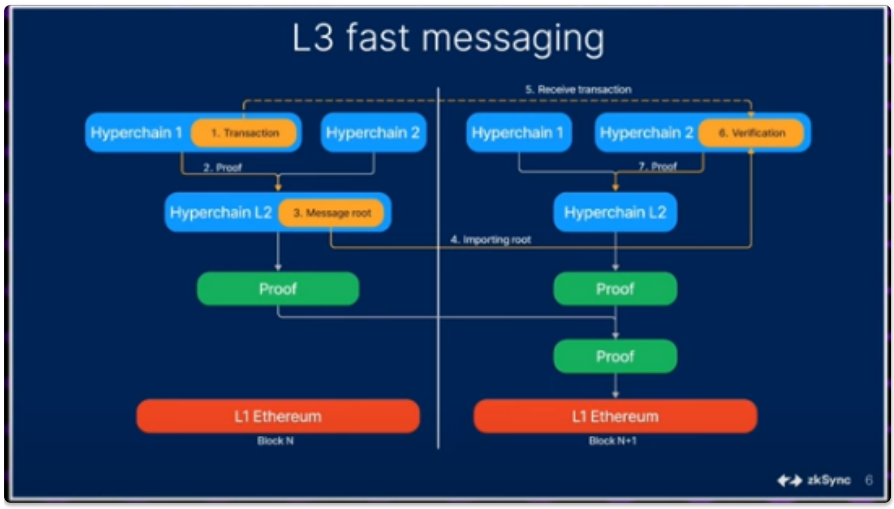
Hyperbridges

These allow hyperchains to cheaply and trustlessly interoperate.
They allow passing of messages and assets between chains allowing for example a unified view of liquidity in the ecosystem. This requires L1 assets to be lodged in a shared bridge on L1.

We have seen aggregation of proofs from L2 to L1, as this is an expensive process (the addition to L1) it is better to settle less often, but this is bad for hyperbridging as this relies on settlment on L1.
For this reason it is better to have L3s settling on L2 , so bridging is faster



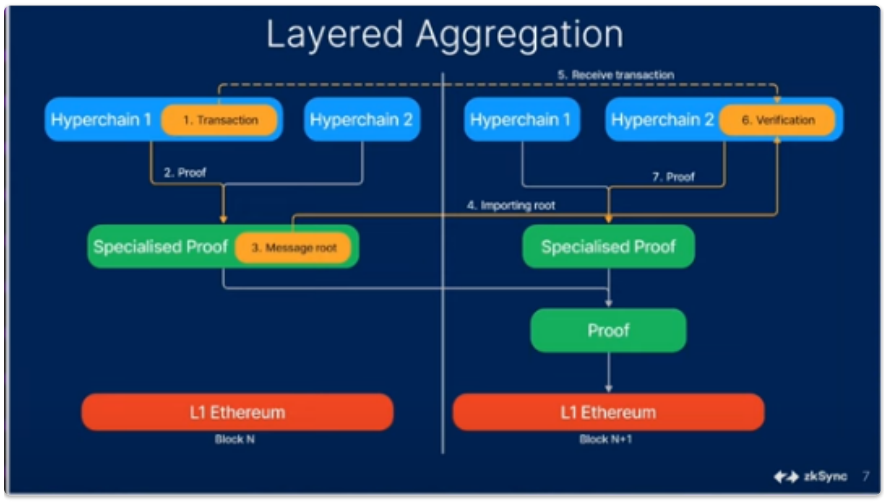
in more detail



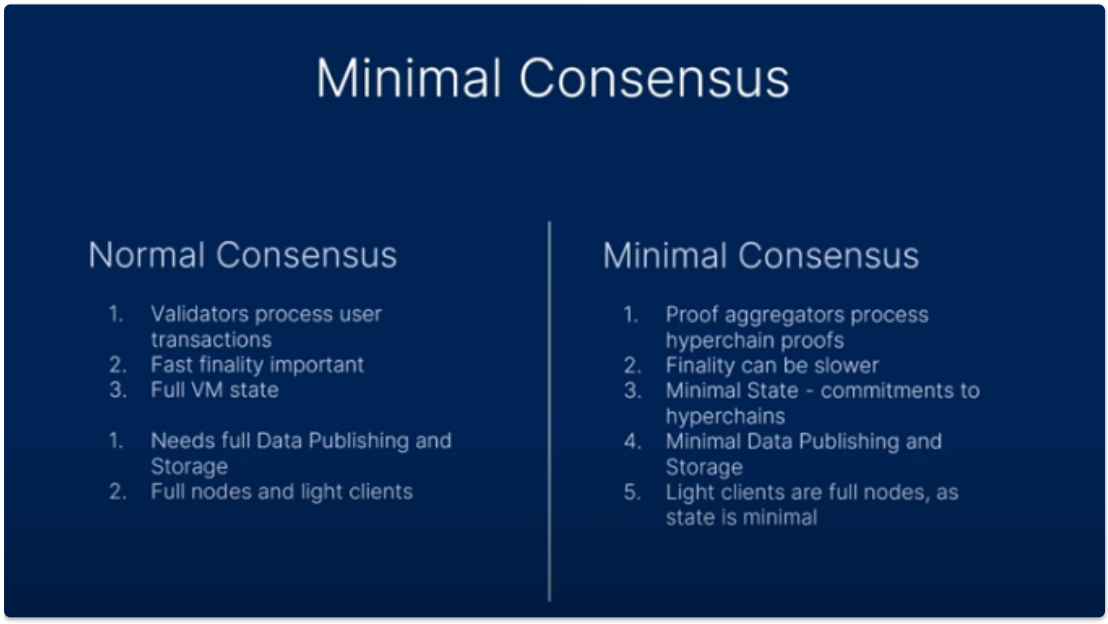
blocks and proofs of different hyperchains are first aggregated across multiple chains, then the proofs of the L2 are aggregated across time.

So we are splitting aggregating across chains, then across time.
We will come onto the possibilities of multi prover systems below.

Scalability comes into play here, the L2 has an VM and a consensus mechanism.
To improve scalability for messaging we can remove the L2 VM and have a specialised proof instead.



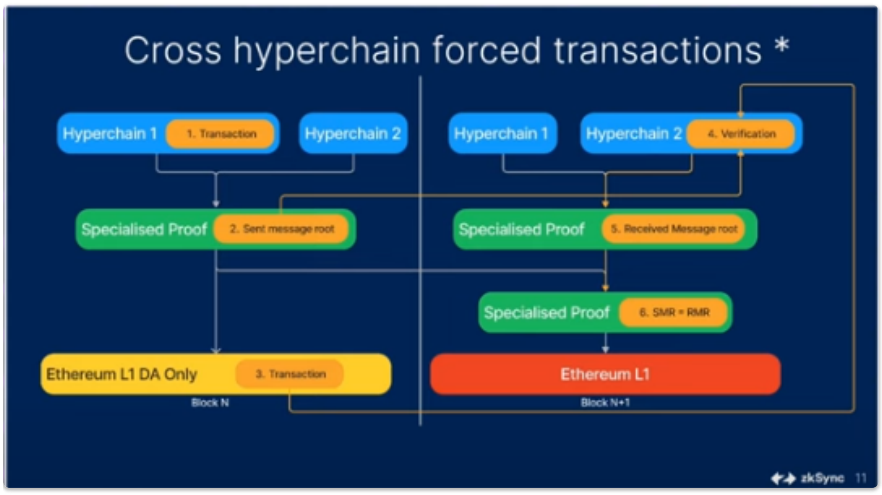
This leaves the consenus mechanism, we are coming to consensus on proofs, therefore we only need a minimal consensus and we don't have storage requirements.



There is a problem around the staking at the multiple layers, there might be more staked on the L2s and L3s than the L1, but we are relying on the L1 for security.
To solve this we use restaking, the stake is logged on L1 and can be used elsewhere.

Forced transactions

Remember the notion of forced transactions that we saw when looking at the components of a L2.



The Hyperbridge itself will be a set of smart contracts, verifying Merkle proofs of the transactions happening on other chains.
The original asset is locked in the shared bridge contract on L1. This means liquidity is unified across the ecosystem.

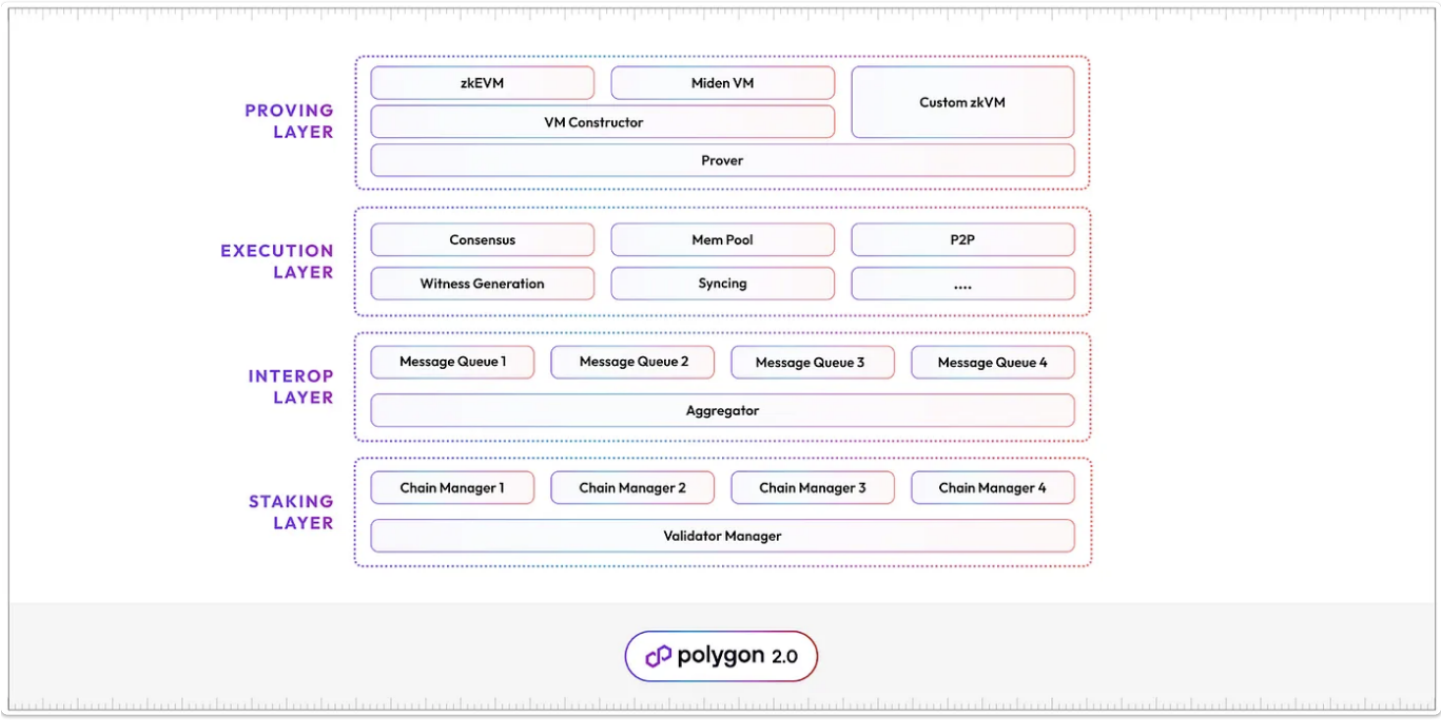
Hyperbridging will consist of 7 steps.

1. A Hyperchain initiates the cross-hyperchain transaction.
2. The sending Hyperchain settles its proof onto L1.
3. As the proof is settled, it updates the Transaction Root. This Root is a commitment to all the Hyperbridge transactions happening inside the ecosystem.
4. The receiving Hyperchain imports this Transaction Root via its consensus mechanism, similarly to how L1→L2 messages are imported today.
5. A relayer sends the transaction and a Merkle Proof connecting it to the Transaction Root to the receiving Hyperchain.
6. The transaction and Merkle proof are verified against the Transaction Root. If the proof is correct, the transaction is executed and the relayer is rewarded.
7. The receiving Hyperchain settles its proof, where the imported Transaction Root is also verified.

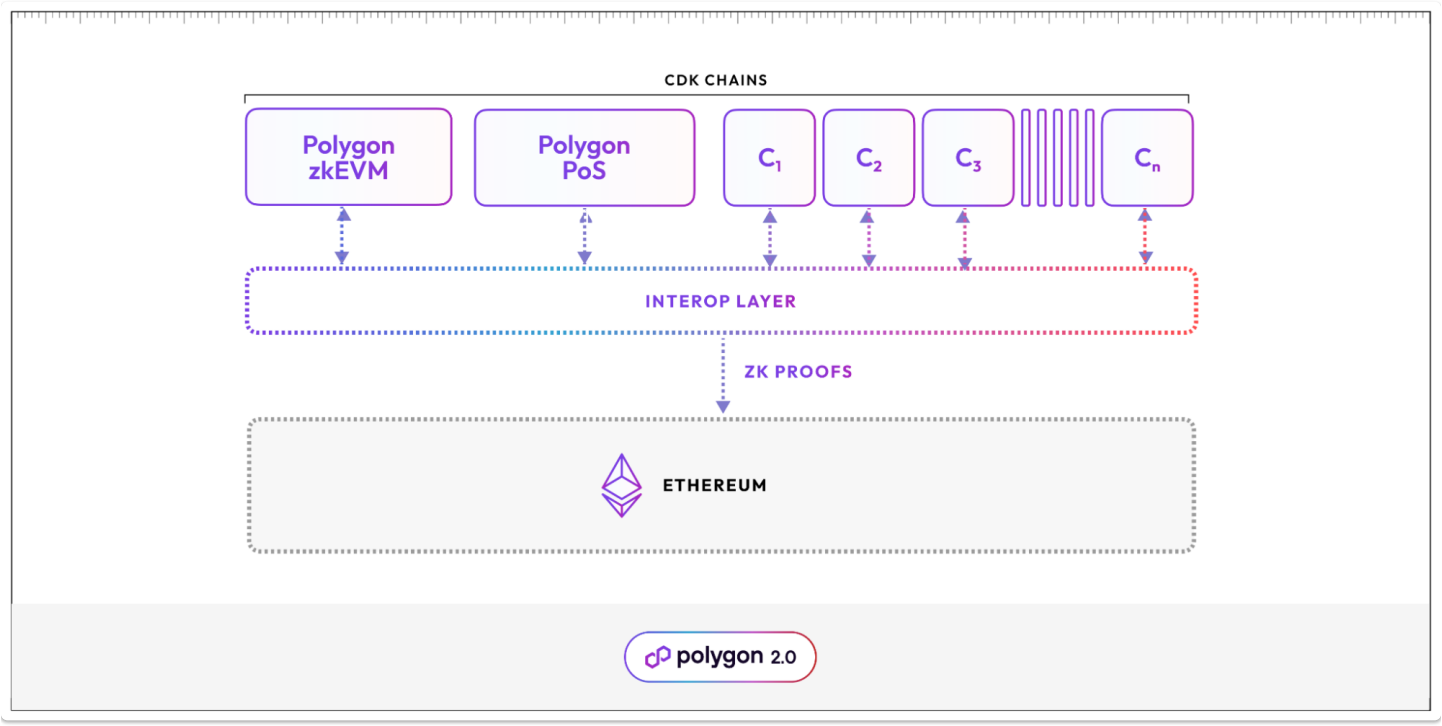
Polygon CDK Overview

See Documentation

See Repo



Polygon have adopted a different approach, producing a development kit to allow simple creation of interoperable L2s.
They argue that
"Others offer L3 solutions that settle on an L2 (which then settles on Ethereum), which increases complexity and fragments liquidity."



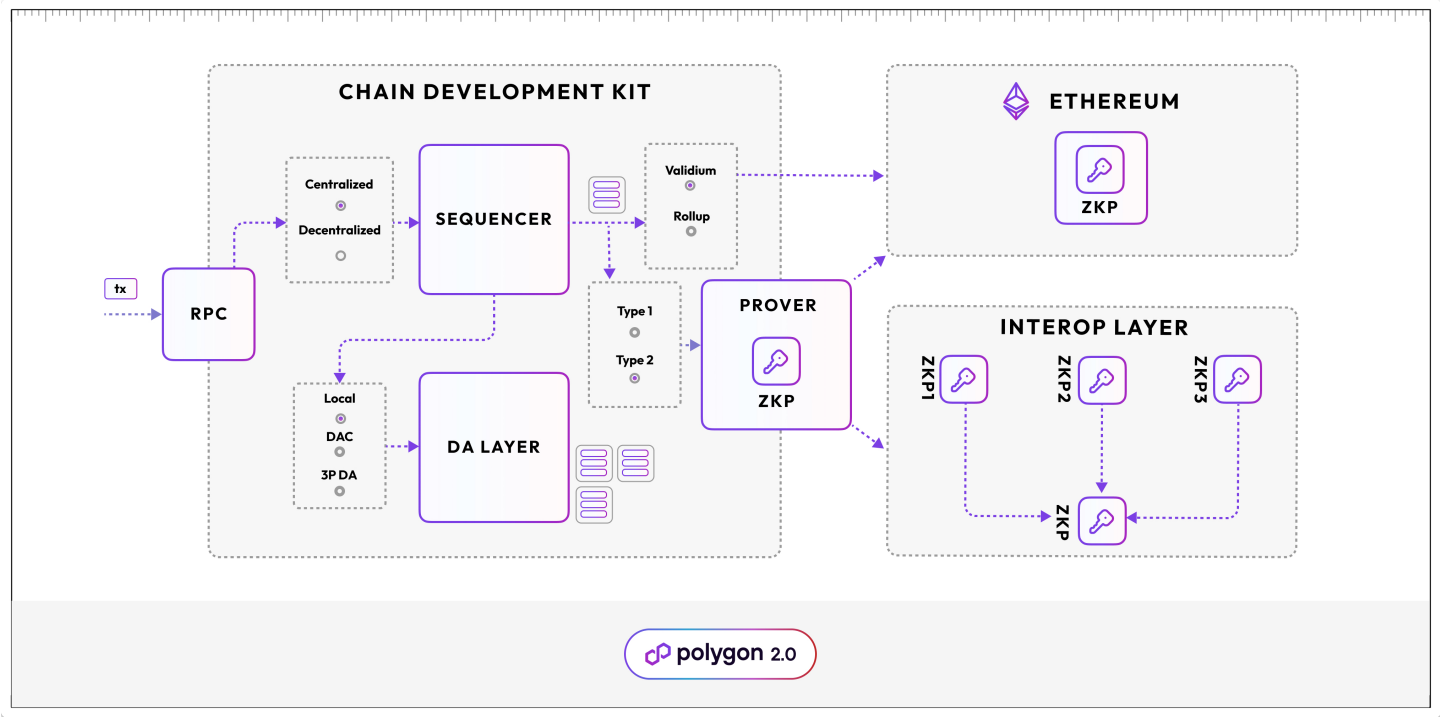
They claim that the Polygon CDK provides

- Secure and near-instant interoperability with all other Polygon chains;
- Automatic, seamless access to shared liquidity of all Polygon chains;
- Fast, one-click access to the entire liquidity of Ethereum;
- Access to production-ready and lightning-fast Polygon proving and proof aggregation technology.

The CDK can be used to create new L2s or convert existing L1s to L2s.

The creation is customisable to give the choice of

- Rollup or validium mode;
- zkEVM or another ZK-powered execution environment (e.g. MidenVM);
- Various data availability solutions;
- Native token and gas token customization;
- Centralized or (on the roadmap) decentralized sequencer mode;
- Permissioned networks with granular allowlists;
- Configurable time to post ZK proofs to Ethereum;



The interoperability is provided by the "Interop layer"
This

- Takes proofs from Polygon chains
- Aggregates the proofs
- Posts the aggregated proof and chain state to Ethereum L1.