Lesson 11 - zkEVM Solutions

Recap

Rollups are solutions that have

- transaction execution outside layer 1
- transaction data and proof of transactions is on layer 1
- a rollup smart contract in layer 1 that can enforce correct transaction execution on layer 2 by using the transaction data on layer 1

The main chain holds funds and commitments to the side chains
The side chain holds additional state and performs execution
There needs to be some proof, either a fraud proof (Optimistic) or a validity proof (zk)

Rollups require "operators" to stake a bond in the rollup contract. This incentivises operators to verify and execute transactions correctly.

Data Availability

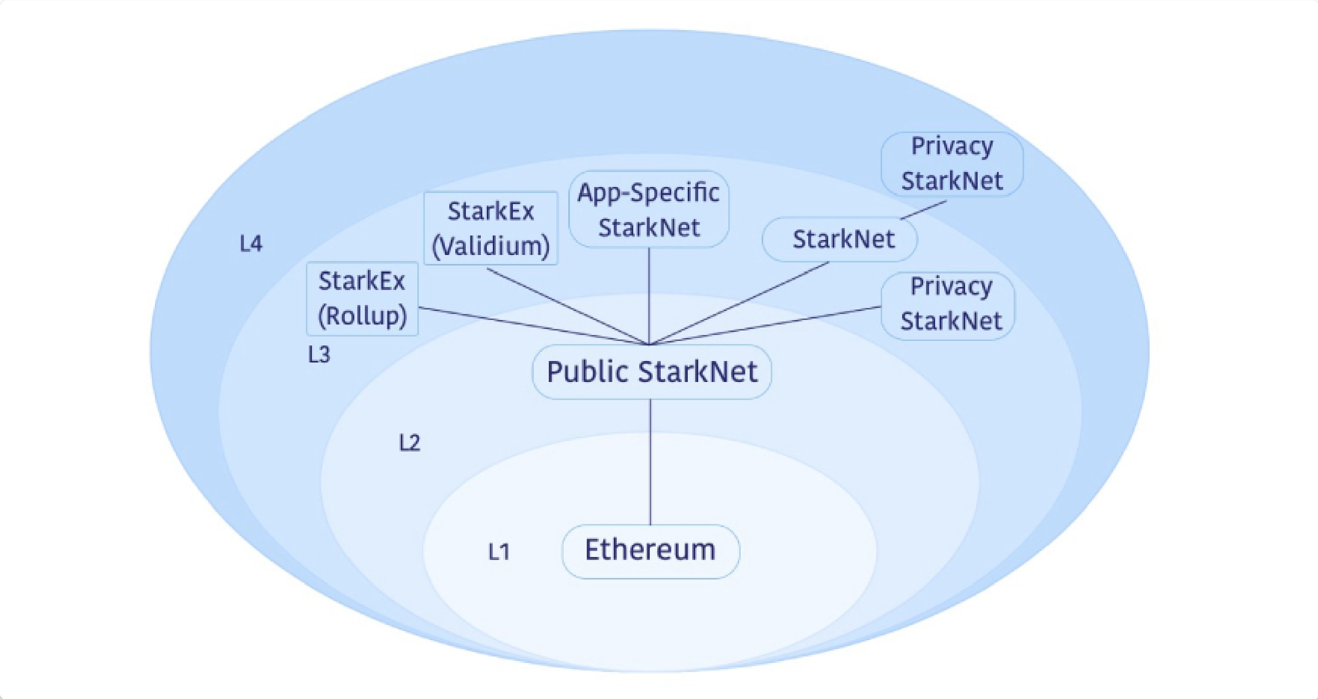
In order to re create the state, tranaction data is needed, the data availability question is where this data is stored and how to make sure it is available to the participants in the system.

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

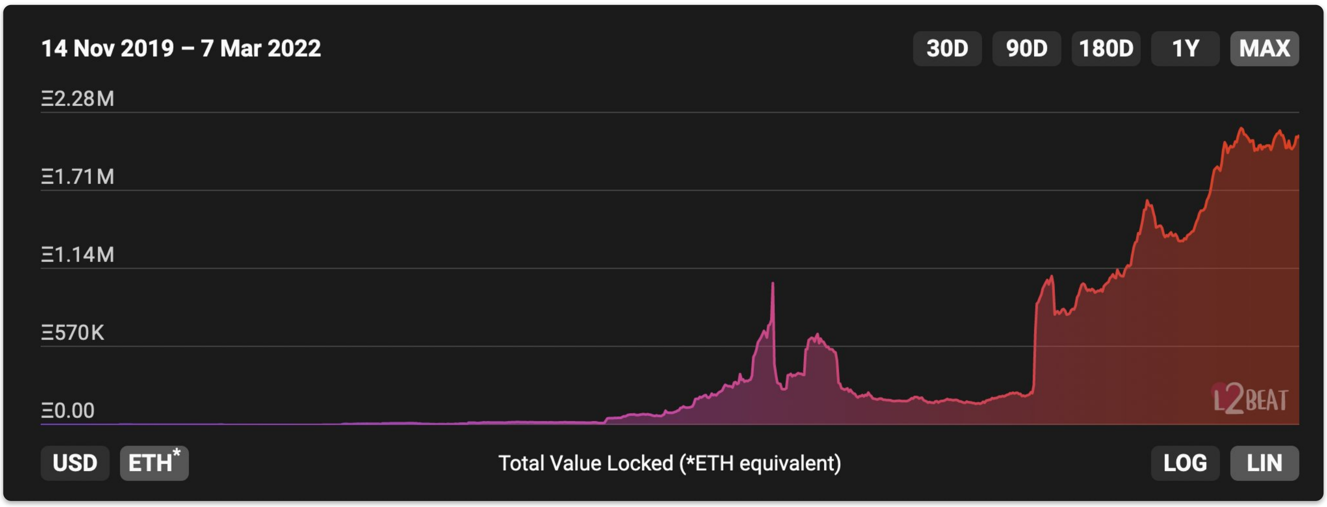
STARKWARE

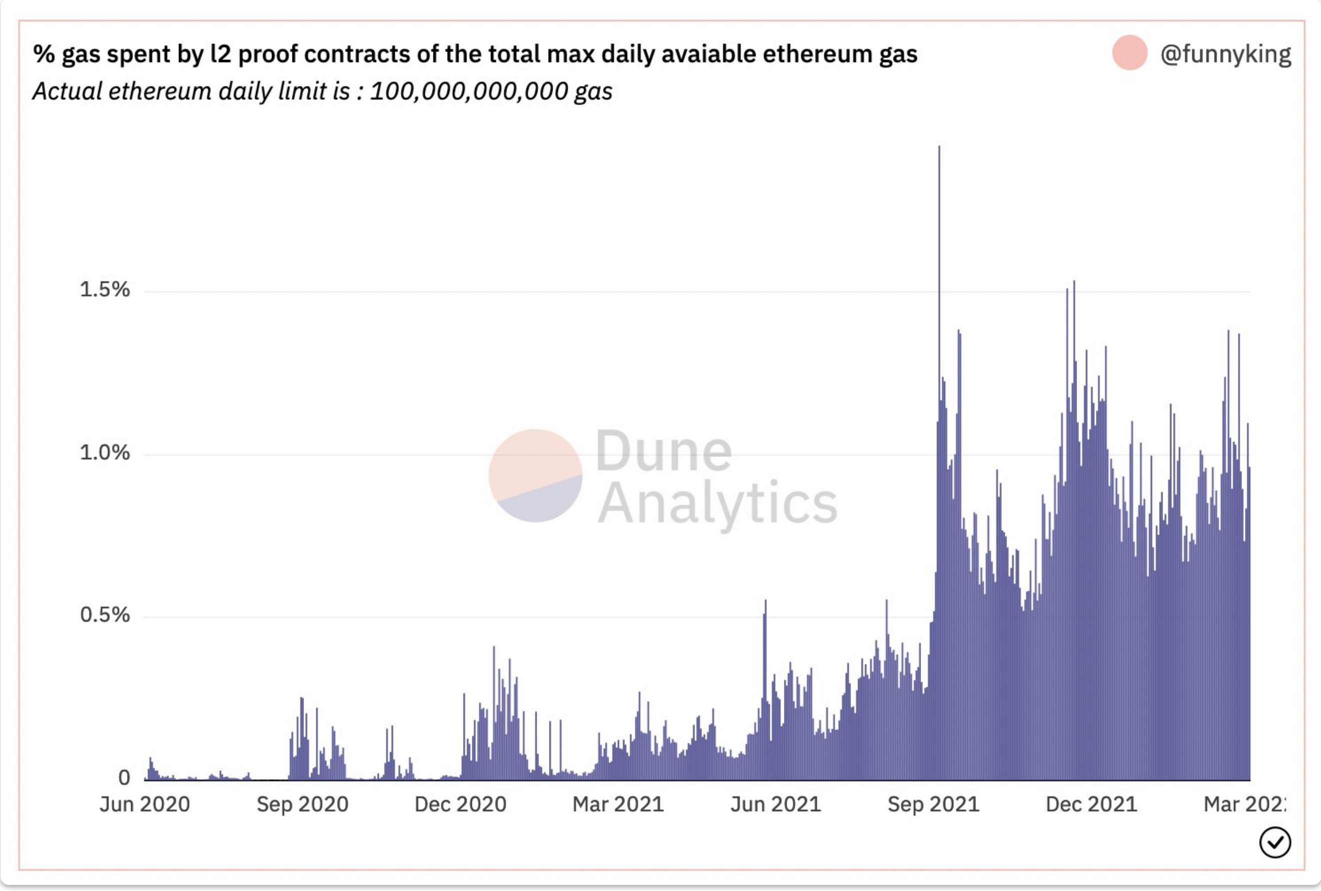
L3 and L4 ?

See Fractal scaling article



L2 Statistics





There has been an increase in bridges over the last year

Transaction compression and costs

For zk rollups it is expensive to verify the validity proof.
For STARKs, this costs ~5 million gas, which when aggregated is about 384 gas cost per transaction.
Due to compression techniques, the calldata cost is actually only 86 gas.
This is further reduced by the calldata EIP448 to 16.1 gas.

The batch cost is poly-log, so if activity increases 100x, the batch costs will decrease to only 4–5 gas per transaction, in which case the calldata reduction would have a huge impact.

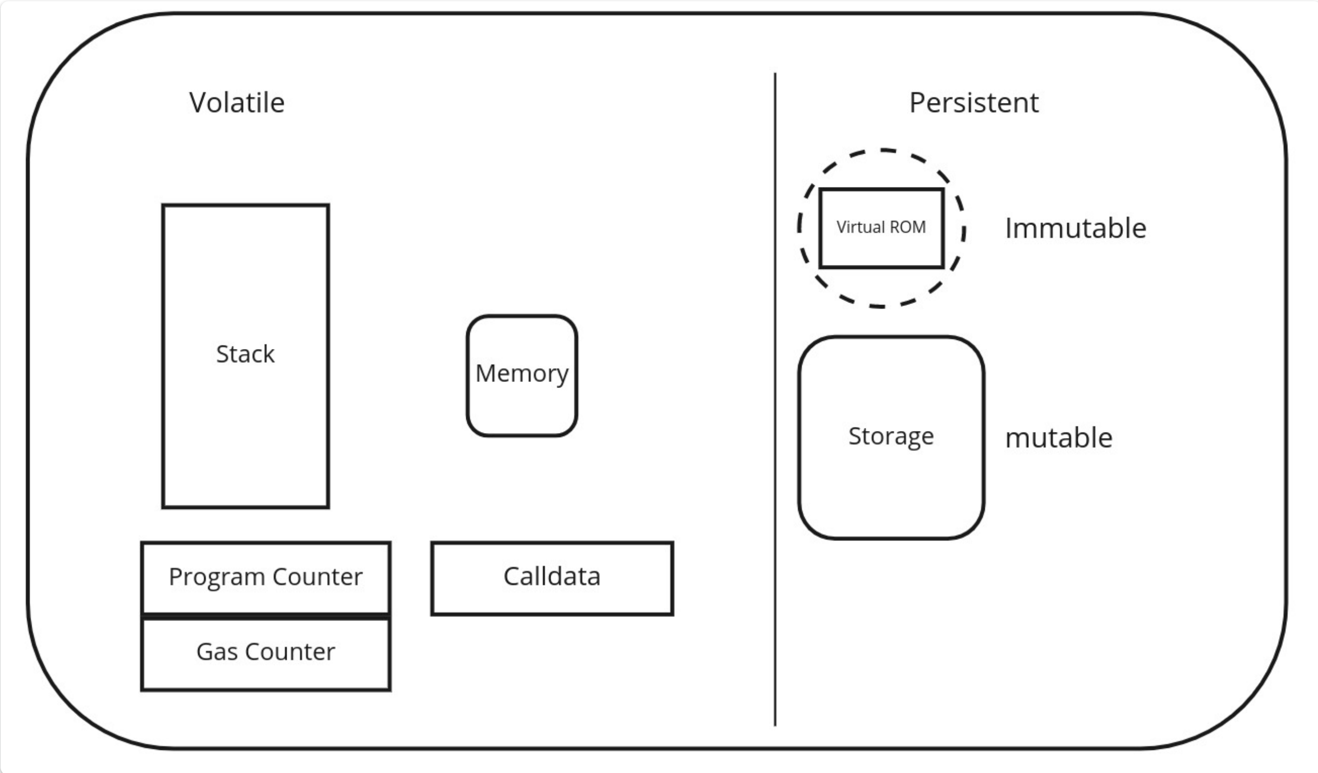
At 100x the TPS today on dYdX, the total on-chain cost will reduce to only 21 gas
At this point, the bottleneck becomes prover costs for the rollup as much as on-chain gas fees, see Polygon Zero and recursive proofs

Approaches to zkRollups on Ethereum

1. Building application-specific circuit (although this can be fairly generic as in Starknet)
2. Building a universal “EVM” circuit for smart contract execution

---

zkEVM Solutions in general



The opcode of the EVM needs to interact with Stack, Memory, and Storage during execution. There should also be some contexts, such as gas/program counter, etc. Stack is only used for Stack access, and Memory and Storage can be accessed randomly.

EVM processing

In general the EVM will

1. Read elements from stack, memory or storage
2. Perform some computation on those elements
3. Write back results to stack, memory or storage

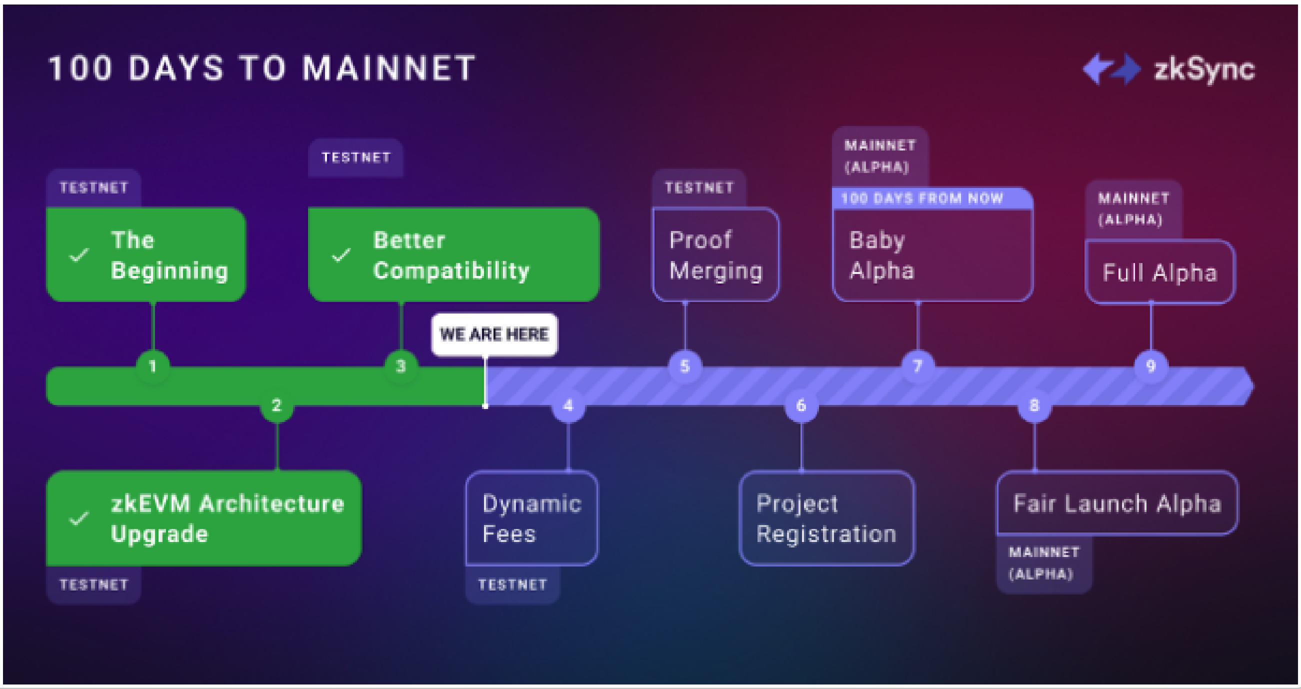
So our circuit has to model / prove this process, in particular

- The bytecode is correctly loaded from persistent storage
- The opcodes in the bytecode are executed in sequence
- Each opcode is executed correctly (following the above 3 steps)

Design challenges in designing a zkEVM

1. We are constrained by the cryptography (curves, hash functions) available on Ethereum.
2. The EVM is stack based rather than register based
3. The EVM has a 256 bit word (not a natural field element size)
4. EVM storage uses keccak and Merkle Patricia trees, which are not zkp friendly
5. We need to model the whole EVM to do a simple op code.

zkSync



Overview

zkSync is built on ZK Rollup architecture. ZK Rollup is an L2 scaling solution in which all funds are held by a smart contract on the mainchain, while computation and storage are performed off-chain.
For every Rollup block, a state transition zero-knowledge proof is generated and verified by the mainchain contract.
This SNARK includes the proof of the validity of every single transaction in the Rollup block. Additionally, the public data update for every block is published over the mainchain network in the cheap calldata.

This architecture provides the following guarantees:

- The Rollup validator(s) can never corrupt the state or steal funds (unlike Sidechains).
- Users can always retrieve the funds from the Rollup even if validator(s) stop cooperating because the data is available (unlike Plasma).
- Thanks to validity proofs, neither users nor a single other trusted party needs to be online to monitor rollup blocks in order to prevent fraud (unlike payment channels or Optimistic Rollups).

In other words, ZK Rollup strictly inherits the security guarantees of the underlying L1

From zkPorter

Features of zkSync 2.0:

- zkSync is EVM and web3 compatible.
- We support Solidity and Vyper: no security re-audit required.
- Porting is effortless: 99% of tooling will work out of the box.
- With zkSync your project will inherit the full security of Ethereum.
- You will benefit from more transactions per second and lower gas fees.
- Build on zkSync 2.0 now and be permanently future-proof.

zkSync Roadmap

From zkBlog

[February 2022] THE BEGINNING — Early this year, we launched zkSync 2.0 on a public testnet — the first ever zkEVM implementation. In this first stage, we began the process of implementing pieces of the production architecture on to testnet. The idea was to build small working systems that we could test at each major milestone, while getting feedback from developers.

[May 2022] zkEVM ARCHITECTURE UPGRADE — After running on testnet for several months, we learned many new things that led us to make significant improvements to the zkEVM that set us up for better scalability, better performance, better security, and lay the foundation for future features.

- Major Architecture Overhaul — We made significant changes to our VM architecture that creates a strong foundation for future improvements and for future proofing our production system.
- Implementation of Account Abstraction — A significant enhancement to the default behavior of EVM. One of the big advancements with zkSync 2.0 is the concept of Account Abstraction — The decoupling of the object holding tokens (the account) from the object authorized to move tokens (the signer). This will enable developers to turn accounts into smart contracts with their own logic.
- Future Feature Support — Beyond account abstraction, the overhaul sets us up to move towards better performance, security, and for support of Layer 3.

[July 2022] BETTER COMPATIBILITY — One of our primary goals is to make it easy for developers to port to zkSync. We understand that our developer community uses various versions of tools and we've included a major upgrade in our support for older versions of tools so developers don't need to upgrade to use zkSync 2.0.

Two improvements that make porting to zkSync 2.0 easier:

- Solidity Support — Support for Solidity 0.4.11 onwards.
- Vyper Support — Full support for Vyper 0.3.3.

[Summer 2022] DYNAMIC FEES — Developers should only pay for what they use and we've included a major upgrade to the fee modeling system to make sure that fees are charged in the most accurate manner.

- Implementation Dynamic Fee Model — Ensures accurate pay-for-use fees.

[Early Fall 2022] PROOF MERGING — This is what many people have been waiting for: ZK proofs for EVM smart contracts in the live production environment. We've had fully functional circuits in our development environment for some time, but this milestone will enable everyone on our testnet to experience our ZK Prover.

- Merge of the prover — Integrating of the prover with the witness generator into the live block production system.
- Testing — Significant testing for proof performance and verification.

[October 2022] PROJECT REGISTRATION — Once engineering approves the previous milestones, we will open up registration and begin working with ecosystem partners to prepare them for onboarding.

- Registration — You will be able to register to launch on mainnet.

[November 2022] BABY ALPHA — Launch to mainnet at first with no external projects where we are putting the system through a series of real-money stress tests that will help us verify the production system is working correctly and performing as expected.

- 3rd Party Audit — Security audit with top-tier auditors.
- Dev Rel — Full prep of our developer relations team.
- Technical Documentation — Final review and posting of technical documentation.
- FAQ — Final review and posting of our zkSync 2.0 FAQ.
- Tutorials and Examples — Final review of tutorials and examples to make it easy for developers to onboard into zkSync 2.0.

[Q4 2022] FAIR LAUNCH ALPHA — Once the system is tested, we will carefully execute a Fair Launch onboarding process. Fair Launch means that we welcome all of our ecosystem projects and we will not participate in picking winners or favorites. The idea here is we want to make sure we can handle the onboarding of new ecosystem partners wisely. We want each of our ecosystem partners to have a high-quality experience, and with each onboarding we want to improve our systems, processes, and support. At this stage, user access will remain limited.

- Ecosystem — Projects can deploy.
- Block Explorer — Upgraded block explorer that offers enhanced UX, optimized performance, and additional features such as a debugger tool.
- Bridging — Limited to testing purposes.
- Bug Bounties — Open Immunefi bug bounty program.

- Open Source — Circuits may not yet be open source, but we will be working on this process.

[EOY 2022] FULL ALPHA — zkSync 2.0 Alpha is live.

- zkSync 2.0 is available for all projects and users.
- Open Source — we will fully open source zkSync 2.0.
- Prover — Improved performance upgrades.
- Bridging — expanded bridging and announcement of bridging partners.
- Developer Support — improvements and additions to documents, tutorials, and examples based on developer feedback.
- Bonus — zkSync 3.0 update ;)

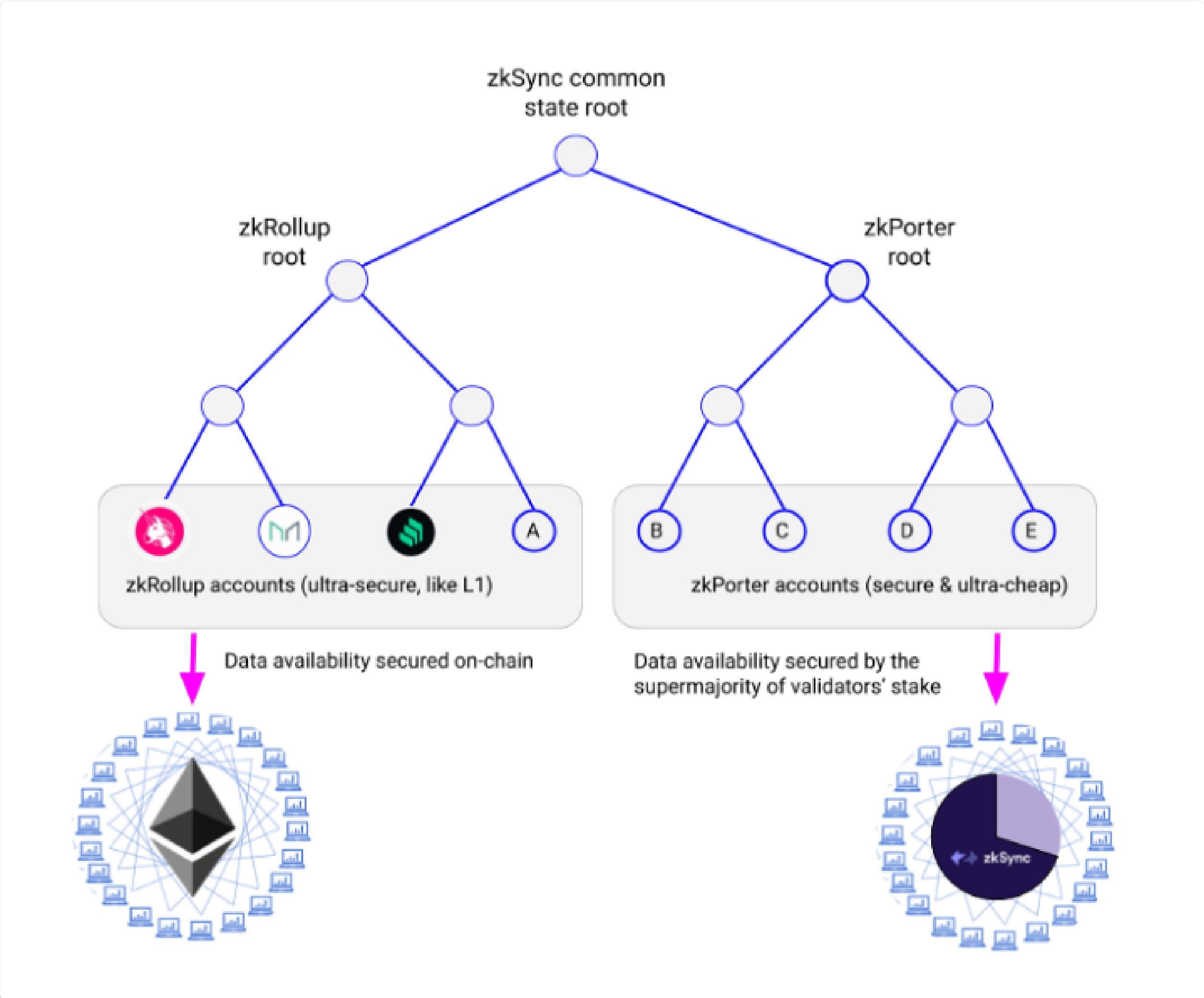
Are rollups enough?

Because any improvement of scalability will be accompanied by an increase in financial activity / trading, on top of new use cases.

Data on zkSync 2.0

In zkSync 2.0, the L2 state will be divided into 2 sides:

- zkRollup with on-chain data availability and
- zkPorter with off-chain data availability



zkSync claim
"Uniswap deploys their smart contract on the zkRollup side, and retail users on a zkPorter account can swap for <$0.03 in fees.
The overwhelming majority of rollup fees are due to the costs of publishing data on Ethereum. zkPorter accounts can make thousands of swaps on the Uniswap contract, but only a single update needs to be published to Ethereum."

Data Availability

From Article
The data availability of zkPorter accounts will be secured by zkSync token holders, termed Guardians. They will keep track of state on the zkPorter side by signing blocks to confirm data availability of zkPorter accounts. Guardians participate in proof of stake (PoS) with the zkSync token, so any failure of data availability will cause them to get slashed. This gives cryptoeconomic guarantees of the data availability.

It is important to note that PoS in zkSync is significantly more secure than PoS in other systems such as sidechains. This is because zkSync guardians are essentially powerless: guardians cannot steal funds. They can only freeze the zkPorter state (freezing their own stake).

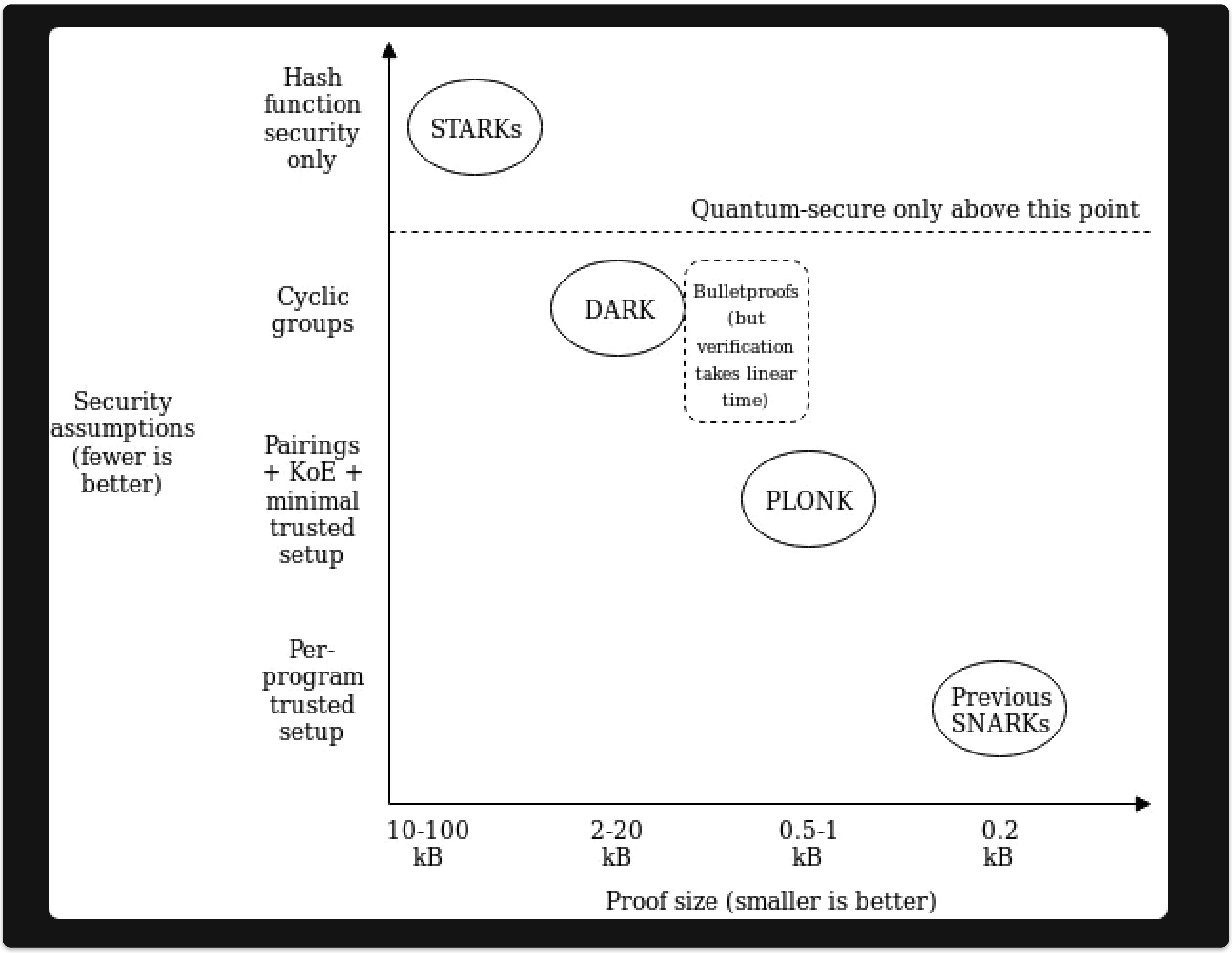
Every user is free to opt into their own security threshold. Any user who wants all data available on-chain can stay completely on the rollup side. But if you are a fee-sensitive user, you can choose to make zkPorter your home.

This design falls under the broader solution class called Volition, pioneered by StarkWare. The difference of the zkSync approach is in strict focus on decentralization, which led to some profound architectonic changes.

The zkSync 2.0 state tree covers Ethereum's full 160-bit address space. Each account will reside in either the zkRollup part or zkPorter part of the state. zkRollup and zkPorter accounts are completely identical except for one component: where the data availability is guaranteed.
zkRollup transaction data gets published to Ethereum through calldata, and zkPorter transaction data is published to the zkSync Guardian network, where zkSync token holders participate in Proof of Stake.

Where the data is published is a tradeoff between cost and security. zkPorter transactions are exponentially cheaper than rollup transactions, but it comes with a possibility that your funds could be frozen. However, the *validity* of both zkRollup and zkPorter accounts is guaranteed through zero knowledge proofs and by Ethereum. In other words, funds in zkPorter can only be frozen, never stolen.

Underlying Cryptography



zkSync uses PLONK with custom gates and lookup tables (most commonly referred to as UltraPLONK) and Ethereum's BN-254 curve.

zkSync Infrastructure

zkSync operates several pieces of infrastructure on top of Ethereum. All infrastructure is currently live and operational, including the zkEVM.

- Full Node
  - Executes zkEVM bytecode using the virtual machine
  - Filters incorrect transactions
  - Executes mempool transactions
  - Builds blocks
- Prover
  - Generates ZK proofs from block witnesses
  - provides an interface for parallel proof generation
  - Scalable (can increase # of provers depending on demand)
- Interactor
  - The link between L1 Ethereum and L2 zkSync
  - Calculates transaction fees
    - Fees depend on token prices, proof generation, and L1 gas costs
- Paranoid Monitor
  - Monitors infrastructure and notifies Matter Labs if incidents occur

zkSync Ecosystem

https://ecosystem.zksync.io/

Block Explorer

Implementation code on L1 at 0xd61dFf4b146e8e6bDCDad5C48e72D0bA85D94DbC

Block proving contract

```
/// @notice Blocks commitment verification.

/// @notice Only verifies block commitments without any other processing

function proveBlocks(StoredBlockInfo[] memory _committedBlocks, ProofInput
memory _proof) external nonReentrant {

requireActive();
uint32 currentTotalBlocksProven = totalBlocksProven;

for (uint256 i = 0; i < _committedBlocks.length; ++i) {

        require(hashStoredBlockInfo(_committedBlocks[i]) ==
storedBlockHashes[currentTotalBlocksProven + 1], "o1");

        ++currentTotalBlocksProven;
        require(_proof.commitments[i] & INPUT_MASK ==
uint256(_committedBlocks[i].commitment) & INPUT_MASK, "o"); // incorrect
block commitment in proof

}


bool success =
verifier.verifyAggregatedBlockProof(
        _proof.recursiveInput,
        _proof.proof,
        _proof.vkIndexes,
        _proof.commitments,
        _proof.subproofsLimbs
);

require(success, "p"); // Aggregated proof verification fail
require(currentTotalBlocksProven <= totalBlocksCommitted, "q");
totalBlocksProven = currentTotalBlocksProven;

}
```

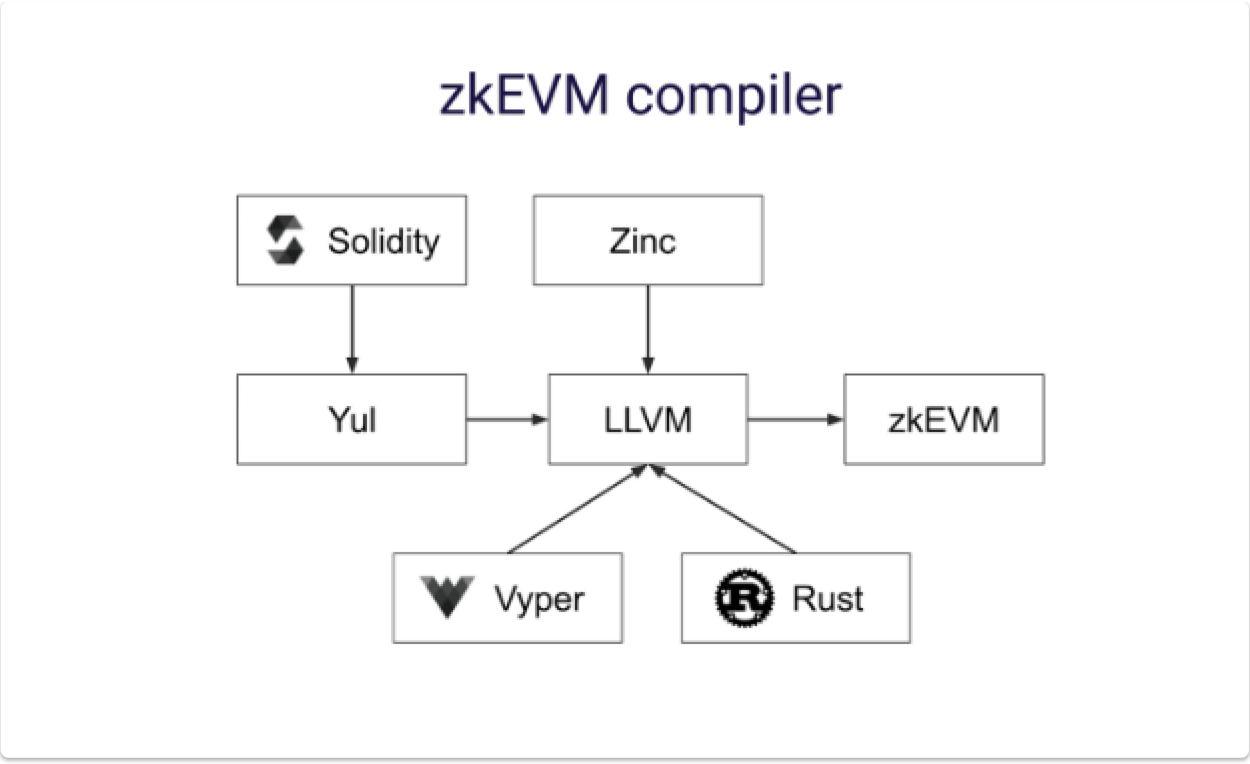
Comparing ZkSync with StarkEx

zkSync vs. StarkWare

	zkSync	StarkWare
Proof	zk-Rollups	zk-Rollups or Validium
Optimal throughput	~300 or 800 - 3000	~3000
Txs/Proof	100 or 315	300
Prover time (minutes)	4 - 14	3 - 5
Optimal gas cost	~1200	~300
Fixed-cost in gas	500k - 900k	~2.5M-5M
Txs weekly on ETH	44k	3.95M
TVL	22M	1B
Hardware	Customized FPGA	Own prover device
Data availability	Yes	Yes or Committee
Software license	Open-source (Apache/MIT)	Not open-source (others cannot run STARK prover)
Developer stack	Focus on EVM	Built Cairo (not backwards compatible)
Upgradeability	Mandatory Timelock	Freeze operation until upgrade executed

Matter Labs zkEVM

zkEVM is a virtual machine that executes smart contracts in a way that is compatible with zero-knowledge-proof computation.
zk-EVM keeps EVM semantics, but is also ZK-friendly and takes on traditional CPU architectures.
zkSync's zkEVM is not a replica of the EVM but is newly designed to run 99% of Solidity contracts and ensure that it works properly under a variety of conditions (including rollbacks and exceptions). At the same time, zkEVM can be used to efficiently generate zero-knowledge proofs in the circuit.



The circuit implementation of Matter Labs uses TinyRAM to implement ordinary opcodes, such as ADD, PUSH, etc.; opcodes that consume a lot of gas, such as SHA256/keccak, implement this circuit especially; finally, Matter Labs uses recursive aggregation technology to aggregate all proofs into one proof.

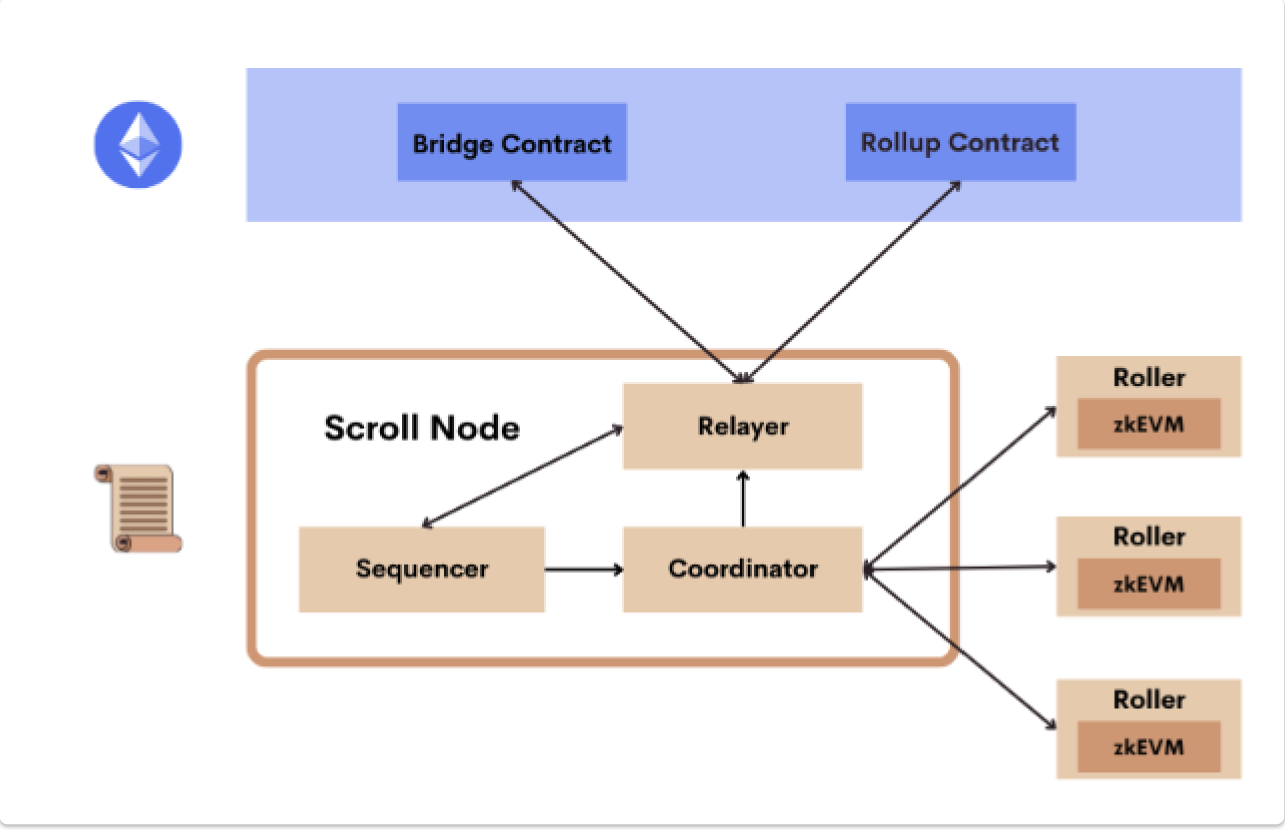
Scroll

This project is still at en early stage, there alpha testnet will be run on a private PoA fork of Ethereum (the testnet L1) operated by Scroll.
On top of this private chain, will run a testnet Scroll L2 supporting the following features:

- Users will be able to play with a few key demo applications such as a Uniswap fork with familiar web interfaces such as Metamask.
- Users will be able to view the state of the Scroll testnet via block explorers.
- Scroll will run a node that supports unlimited read operations (e.g. getting the state of accounts) and user-initiated transactions involving interactions with the pre-deployed demo applications (e.g. transfers of ERC-20 tokens or swaps of tokens).
- Rollers will generate and aggregate validity proofs for part of the zkEVM circuits to ensure a stable release. In the next testnet phase, we will ramp up this set of zkEVM circuits.
- Bridging assets between these testnet L1 and L2s will be enabled through a smart contract bridge, though arbitrary message passing will not be supported in this release.

From article and article

Scroll Architecture



Components

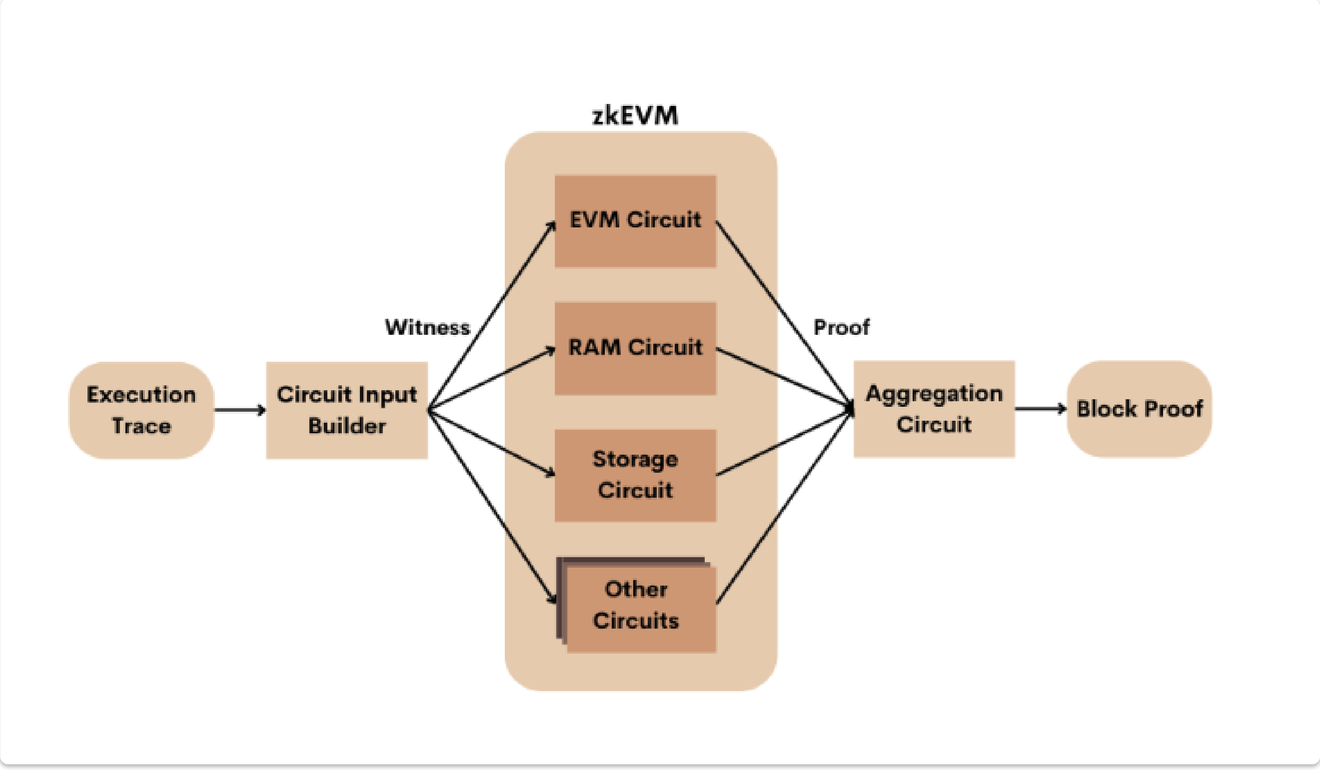
The **Sequencer** provides a JSON-RPC interface and accepts L2 transactions. Every few seconds, it retrieves a batch of transactions from the L2 mempool and executes them to generate a new L2 block and a new state root.

Once a new block is generated, the **Coordinator** is notified and receives the execution trace of this block from the Sequencer.
It then dispatches the execution trace to a randomly-selected **Roller** from the roller pool for proof generation.

The **Relayer** watches the bridge and rollup contracts deployed on both Ethereum and Scroll. It has two main responsibilities.

1. It monitors the rollup contract to keep track of the status of L2 blocks including their data availability and validity proof.
2. It watches the deposit and withdraw events from the bridge contracts deployed on both Ethereum and Scroll and relays the messages from one side to the other.

Rollers - creating proofs

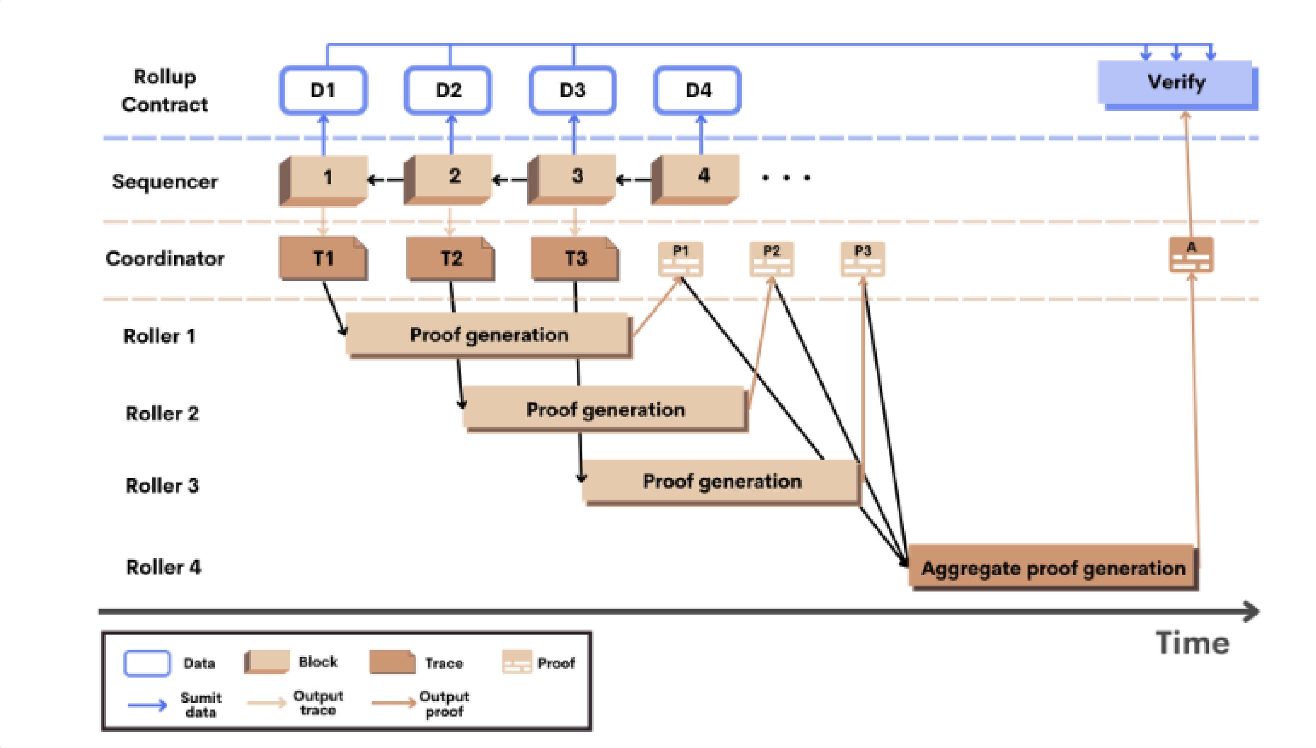


The **Rollers** serve as provers in the network that are responsible for generating validity proofs for the zkRollup

- A Roller first converts the execution trace received from the **Coordinator** to circuit witnesses.
- It generates proofs for each of the **zkEVM** circuits.
- Finally, it uses **proof aggregation** to combine proofs from multiple zkEVM circuits into a single block proof.

The **Rollup contract** on L1 receives L2 state roots and blocks from the Sequencer.
It stores state roots in the Ethereum state and L2 block data as Ethereum calldata.
This provides **data availability** for Scroll blocks and leverages the security of Ethereum to ensure that indexers including the Scroll Relayer can reconstruct L2 blocks.
Once a block proof establishing the validity of an L2 block has been verified by the Rollup contract, the corresponding block is considered finalized on Scroll.

A useful sequence diagram from the Scroll Documentation



L2 blocks in Scroll are generated, committed to base layer Ethereum, and finalized in the following sequence of steps:

1. The Sequencer generates a sequence of blocks. For the *i*-th block, the Sequencer generates an execution trace ***T*** and sends it to the Coordinator. Meanwhile, it also submits the transaction data ***D*** as calldata to the Rollup contract on Ethereum for data availability and the resulting state roots and commitments to the transaction data to the Rollup contract as state.
2. The Coordinator randomly selects a Roller to generate a validity proof for each block trace. To speed up the proof generation process, proofs for different blocks can be generated in parallel on different Rollers.
3. After generating the block proof ***P*** for the *i*-th block, the Roller sends it back to the Coordinator. Every *k* blocks, the Coordinator dispatches an aggregation task to another Roller to aggregate *k* block proofs into a single aggregate proof ***A***.
4. Finally, the Coordinator submits the aggregate proof ***A*** to the Rollup contract to finalize L2 blocks *i+1* to *i+k* by verifying the aggregate proof against the state roots and transaction data commitments previously submitted to the rollup contract.

Scroll circuit design

1. We need an accumulator to provide the proofs of storage, merkle trees can provide this
2. The execution trace is needed to show the path that the execution took through the bytecode, as this would change because of jumps. This trace is then a witness provided to the circuit.
3. Two proofs are used to show the execution is correct for each opcode
    1. Proof of fetching the data required for the opcode
    2. Proof that the opcode executed correctly.

Scroll are working with Ethereum on this, see this repo for EVM circuit design, and this design document from Ethereum.

Polygon Products

See this guide
Strategy article
Recent paper on efficient zk proofs for Keccak

Polygon Zero

zkRollup solutions have a bottleneck in the time in takes to generate a proof.

Polygon Zero attempts to solve this with "recursive proofs", based on Plonky2.

Polygon Zero generates proofs simultaneously for every transaction in the batch. Theses are then aggregated into a single proof which is submitted on the Ethereum network.

This approach significantly reduces the effort it takes to generate reliable validity proofs. Polygon Zero's Plonky2 can generate a recursive proof in 0.17 seconds.

PLONKY2 DIGRESSION

From article

Plonky2 also allows us to speed up proving times for proofs that don't involve recursion. With FRI, you can either have fast proofs that are big (so they're more expensive to verify on Ethereum), or you can have slow proofs that are small. Constructions that use FRI, like the STARKs that Starkware uses in their ZK-rollups, have to choose; they can't have maximally fast proving times and proof sizes that are small enough to reasonably verify on Ethereum.

Plonky2 eliminates this tradeoff. In cases where proving time matters, we can optimize for maximally fast proofs. When these proofs are recursively aggregated, we're left with a single proof that can be verified in a small circuit. At this point, we can optimize for proof size. We can shrink our proof sizes down to 45kb with only 20s of proving time (not a big deal since we only generate when we submit to Ethereum), dramatically reducing costs relative to Starkware.

Plonky2 is natively compatible with Ethereum. Plonky2 requires only keccak-256 to verify a proof. We've estimated that the gas cost to verify a plonky2 size-optimized proof on Ethereum will be approximately 1 million gas.

However, this cost is dominated by the CALLDATA costs to publish the proof on Ethereum. Since CALLDATA was repriced in EIP-4488, the verification cost of a plonky2 proof has dropped to between 170-200k gas, which could make it not only the fastest proving system, but also the cheapest to verify on Ethereum.

Polygon Hermez

Polygon (Hermez) team are working on a protocol Proof of Efficiency
This involves 2 permissionless roles :

- Sequencer
- Aggregator

From article

Sequencers collect transactions from users on the rollup, then select and pre-process new batches of this Layer 2 data. Finally, they send transactions to Layer 1 to be recorded. Sequencers also deposit a fee in $MATIC token as an incentive for Aggregators to include the batch in a zero knowledge proof.

Hermez 2.0

Hermez current functionality is limited to token transfers and atomic swaps, so there are plans to introduce Hermez 2.0 which will have EVM compatibility.

Polygon Midden

Polygon Miden is a general-purpose, STARK-based ZK rollup with EVM compatibility.
This will differ from Starknet in that it will be EVM compatible, so it should run Solidity contracts.
From their documentation
"Polygon Miden can process up to 5,000 transactions in a single block, with new blocks produced every five seconds. Although this ZK rollup exists as a prototype for now, it is expected to boost throughput to over 1,000 transactions per second (TPS) at launch."

Polygon Nightfall

From a collaboration with EY it is designed to allow private transactions.
It is a combination Optimistic rollups and zero knowledge, optimistic rollups for scalability and zk for privacy.
There is a beta version available on mainnet.

Interoperability between L2 solutions

See article
The default way to transfer would be

- Transfer L2 A -> L1
- Transfer L1 -> L2 B

But this is obviously costly in gas fees and if one of the L2s uses fraud proofs could take a long time.

L2 projects have proposed using liquidity provider on L1 to faciltate transfers

Starknet uses an approach of conditional transactions using the fact registry on L1 and a third party liquidity provider.

Loopring has proposed Ethport, a Bridge product across L1, L2 and CEX, by fusing components of its existing toolkit. It uses a combination of bridges and a vault to allow token transfers.

Polygon Hermez uses a coordinator on L2 to aggregate transfers to other L2s