

Lesson 10

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

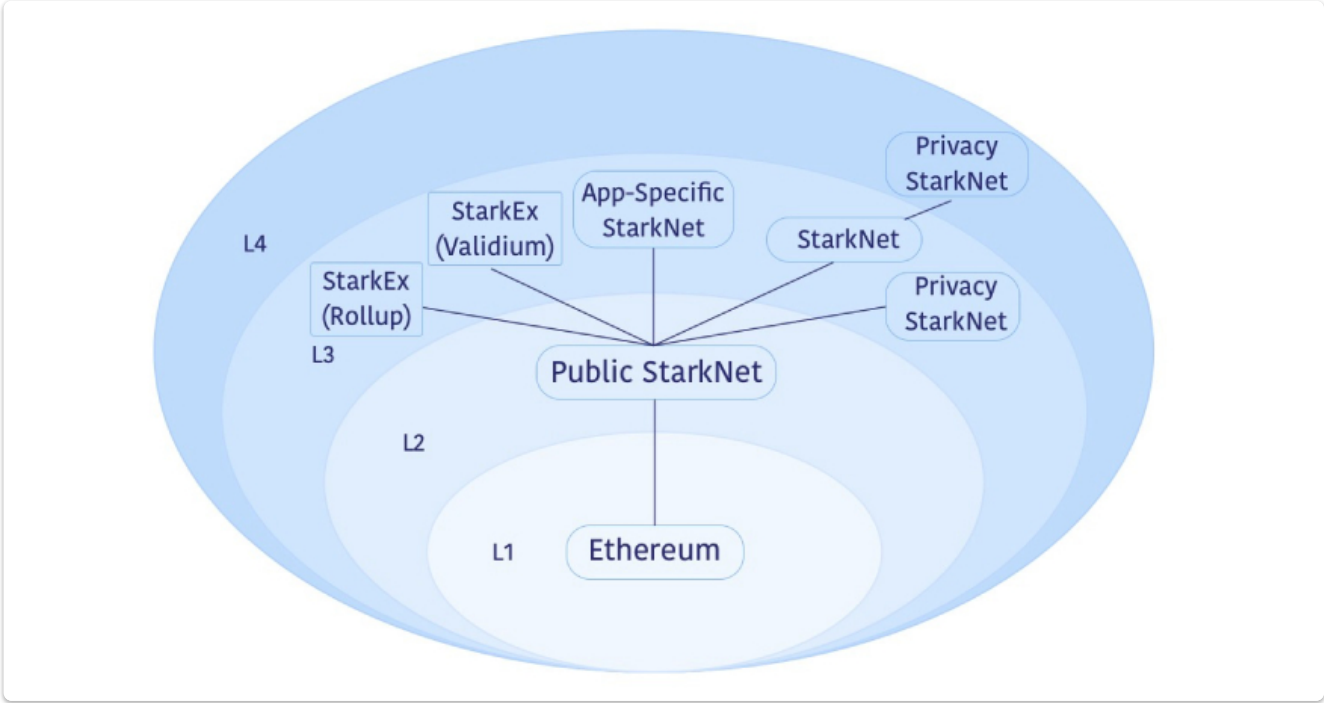
In order to re create the state, transaction data is needed, the data availability question is where this data is stored and how to make sure it is available to the participants in the system.

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

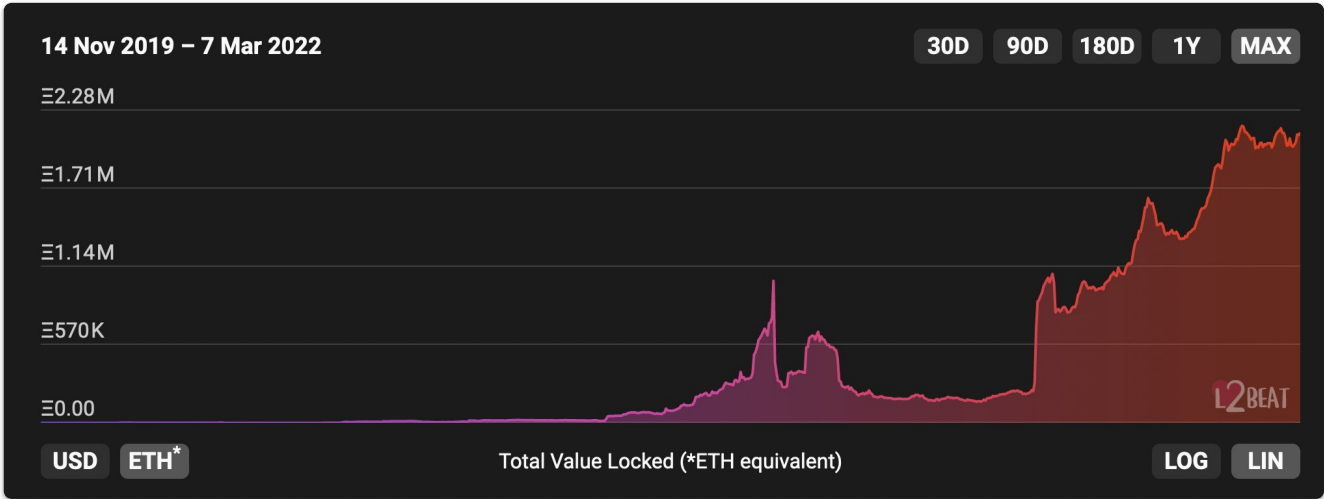
 STARKWARE

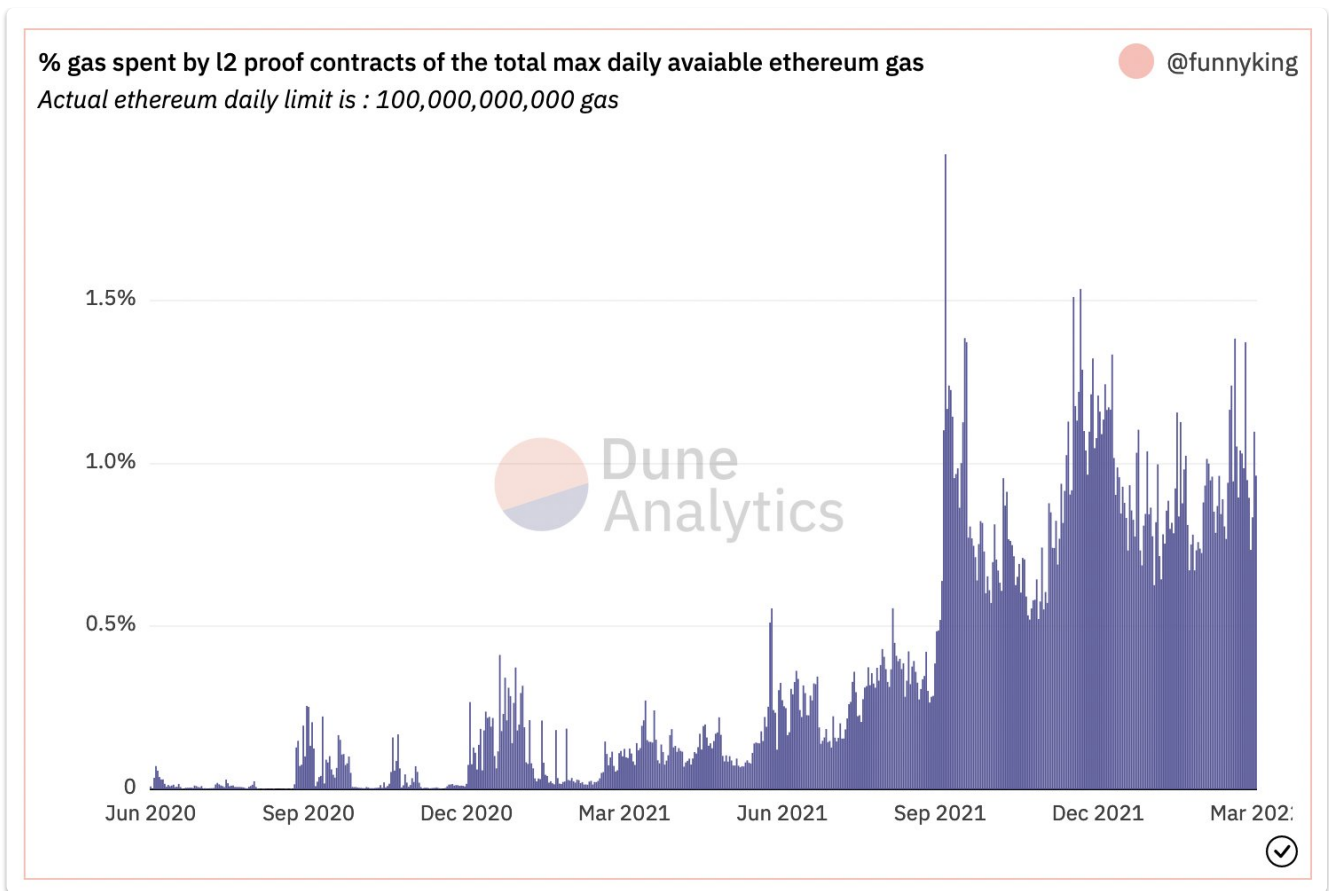
L3 AND L4 ?

See Fractal scaling [article](#)

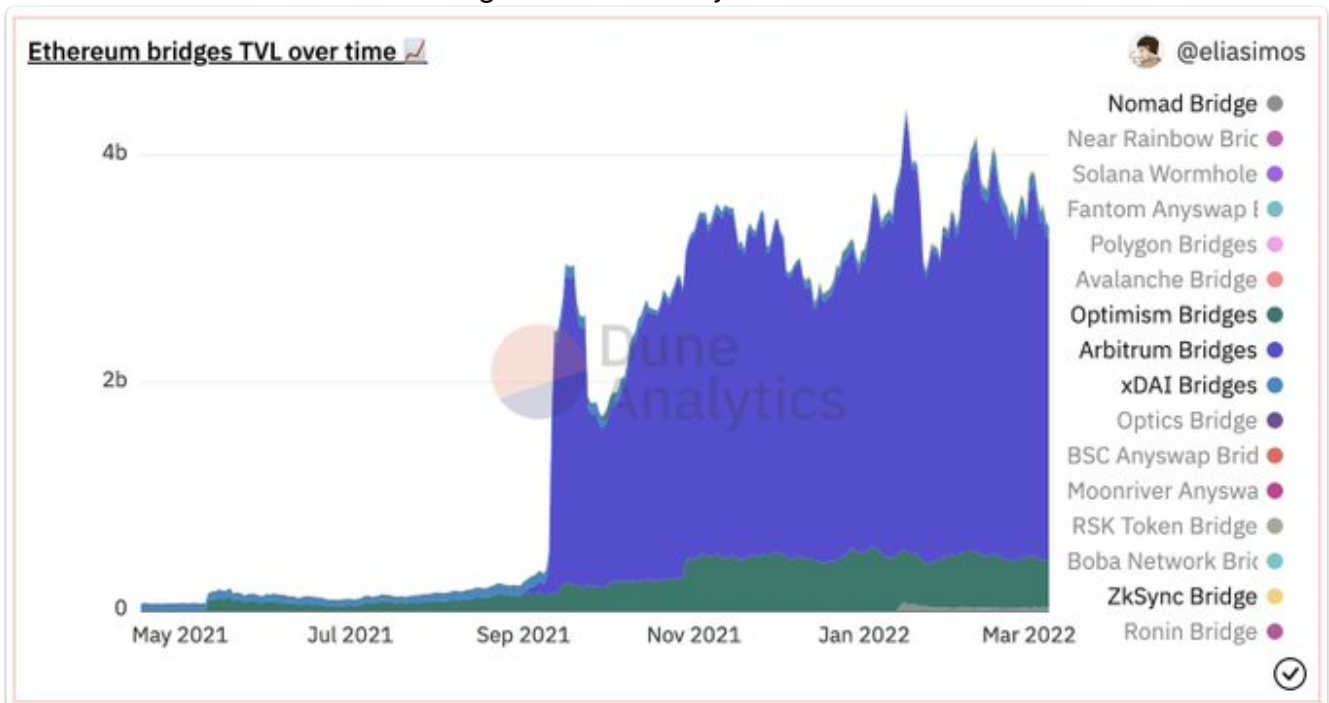


L2 Statistics





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 EIP 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

Starkware projects

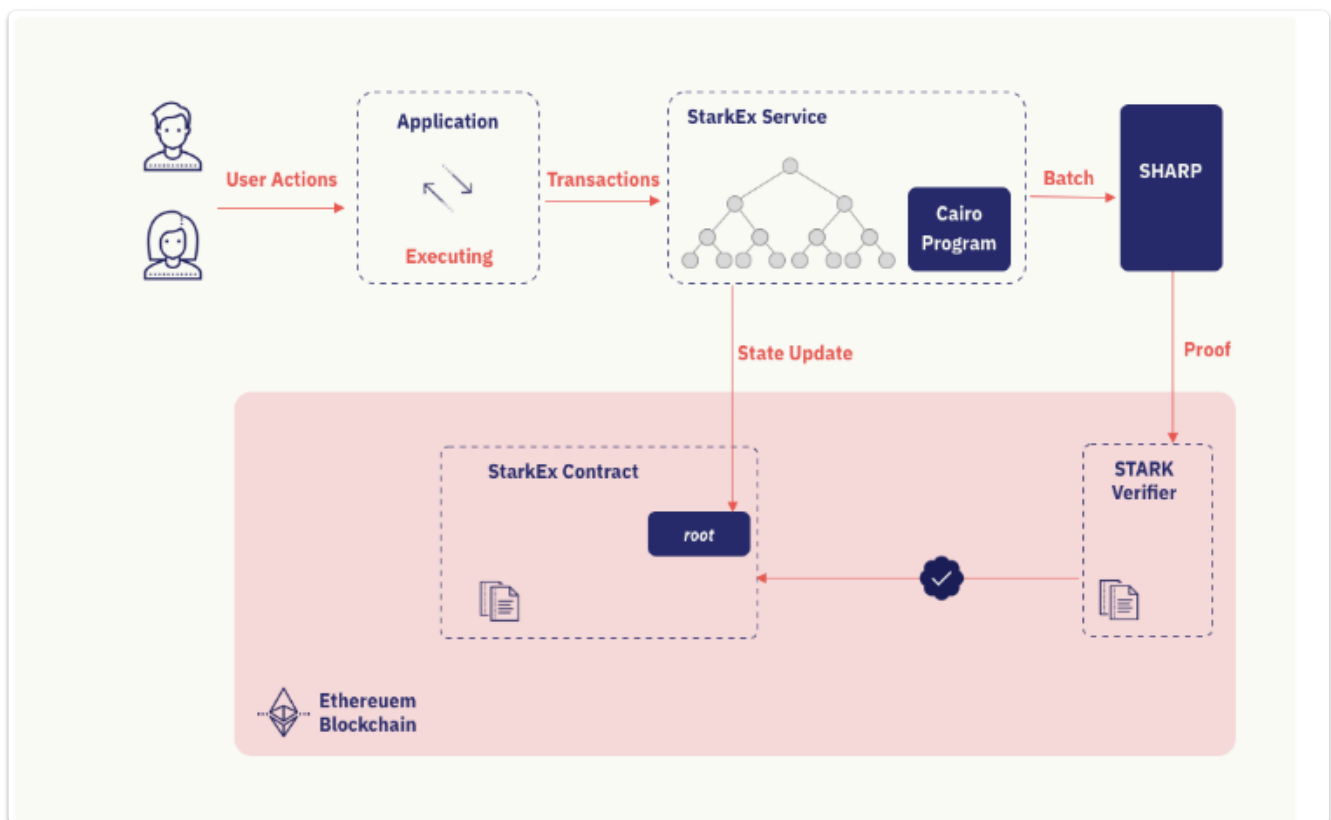
Starkex -> execution /scalability engine

Starknet -> stark rollup network

Starkex is a set of smart contracts focused on trading, and it represents 1/4 of the computation power of Ethereum . It has 270B cumulative trading and 65M transactions settled, with 1.25B total value locked.

Starknet is a new VM written in Cairo. It goes beyond a trading engine allowing to write general-purpose smart contracts. It is a general-purpose computation network that is verifiable by Ethereum.

Interacting with StarkEx



All the transactions in the system are executed by the Application and sent to the StarkEx Service.

- The StarkEx Service batches transactions and sends the batch to SHARP, a shared proving service, to generate a proof attesting to the validity of the batch.
- SHARP sends the STARK proof to the STARK Verifier for verification.
- The service then sends an on-chain state update transaction to the StarkEx Contract, which will be accepted only if the verifier finds the proof valid.

Users interact with the system in two ways:

- by sending on-chain transactions to the StarkEx Contract
- off-chain transactions to the Application.

For an off-chain account, users

- Register their starkKey to their Ethereum Address in the StarkEx Contract.
- The starkKey is used to authenticate the user's off-chain transactions.
- The user then deposits their funds to the StarkEx Contract.
- After the Application accepts the deposit, the user can use their funds off-chain.
- Users submit Transfers or Limit Orders, signed by their starkKey, directly to the Application.
- Limit orders are matched together and sent as a Settlement to the StarkEx Service.

To withdraw their funds,

- users submit an off-chain withdrawal request to the Application.
- This request is sent to the StarkEx Service, and the user can access their funds on-chain once the state update containing the withdrawal is accepted.

L1 and L2 Interaction in Starknet

See [Docs](#)

Messages from L2 to L1

Messages from L2 to L1 work as follows:

1. The StarkNet (L2) contract function calls the library function `send_message_to_l1()` in order to send the message. It specifies:
 1. The destination L1 contract ("to"),
 2. The data to send ("payload")

The StarkNet OS adds the "from" address, which is the L2 address of the contract sending the message.

2. Once a state update containing the L2 transaction is accepted on-chain, the message is stored on the L1 StarkNet core contract, waiting to be consumed.
3. The L1 contract specified by the "to" address invokes the `consumeMessageFromL2()` of the StarkNet core contract.

Note: Since any L2 contract can send messages to any L1 contract it is recommended that the L1 contract check the "from" address before processing the transaction.

Messages from L1 to L2

The other direction is similar:

1. The L1 contract calls (on L1) the `send_message()` function of the StarkNet core contract, which stores the message. In this case the message includes an additional field - the "selector", which determines what function to call in the corresponding L2 contract.

2. The StarkNet Sequencer automatically consumes the message and invokes the requested L2 function of the contract designated by the "to" address.

This direction is useful, for example, for "deposit" transactions.

Note that while honest Sequencers automatically consume L1 -> L2 messages, it is not enforced by the protocol (so a Sequencer may choose to skip a message). This should be taken into account when designing the message protocol between the two contracts.

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

USER PAYMENT FLOW IS

1. Create an account

- deposit funds from Ethereum or from another zkSync address
- create a signing key (an additional signature alongside the normal Ethereum signature)

2. Sending Transactions

- Prepare the transaction data.
- Encode the transaction data into a byte sequence.
- Create a zkSync signature for these bytes with the zkSync private key.
- Generate an Ethereum signature.
- Send the transaction via corresponding JSON RPC method.

3. Withdrawing funds

- Withdraw transaction (L2 -> Ethereum address)
- Forced Exit (for accounts that don't have a signing key) or Full exit transaction (L1)

ZKSYNC 2.0

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

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

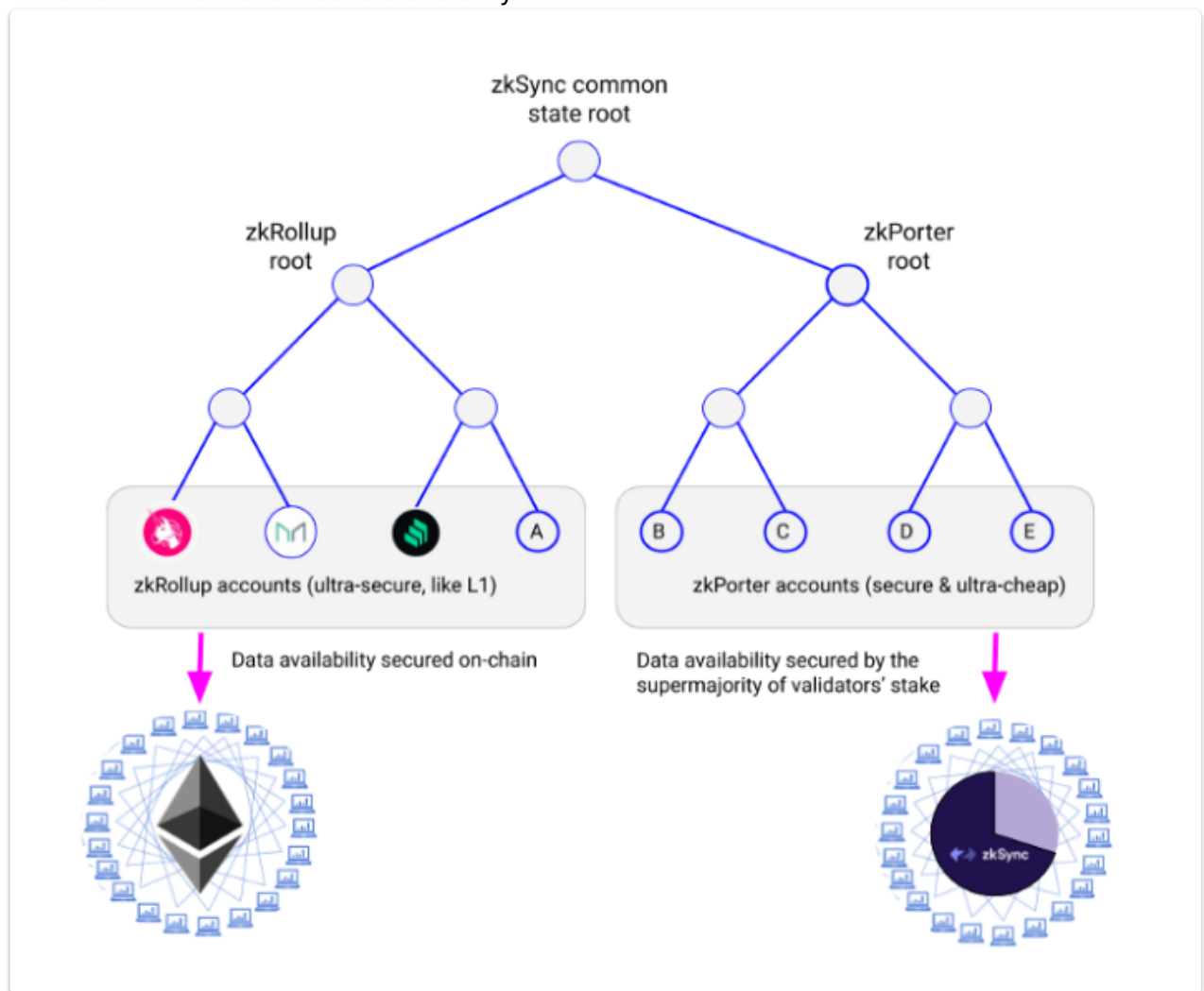
From [zkPorter](#)

Why aren't rollups enough?

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

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).

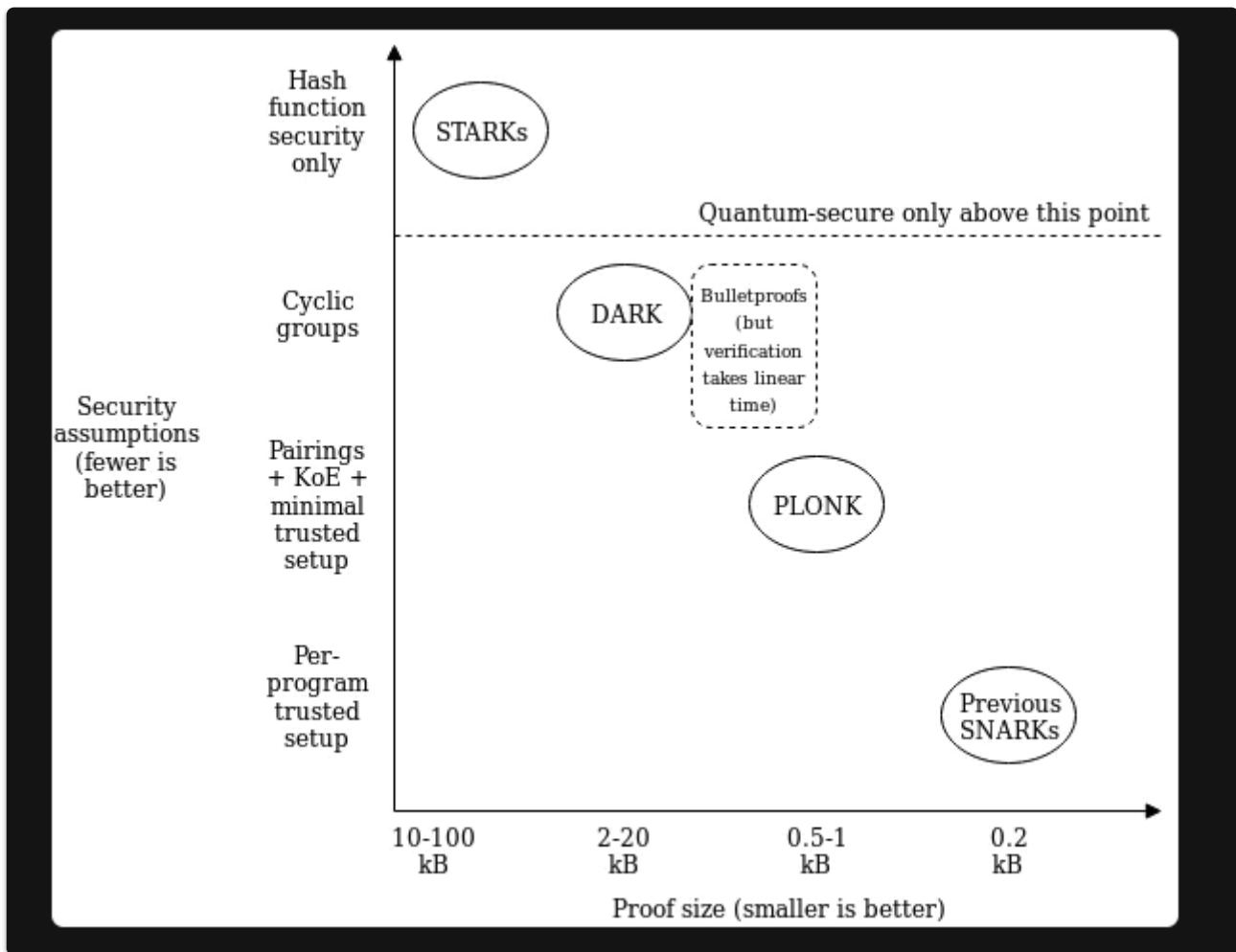
Every user is free to opt into their own security threshold. Any user who wants all data available on-chain can stay completely on the rollup side. But if you are a fee-sensitive user, you can choose to make zkPorter your home.

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 PROOF

```
/// @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;

}
```

Governance involvement

```
/// @notice Commit block
/// @notice 1. Checks onchain operations, timestamp.
/// @notice 2. Store block commitments
```

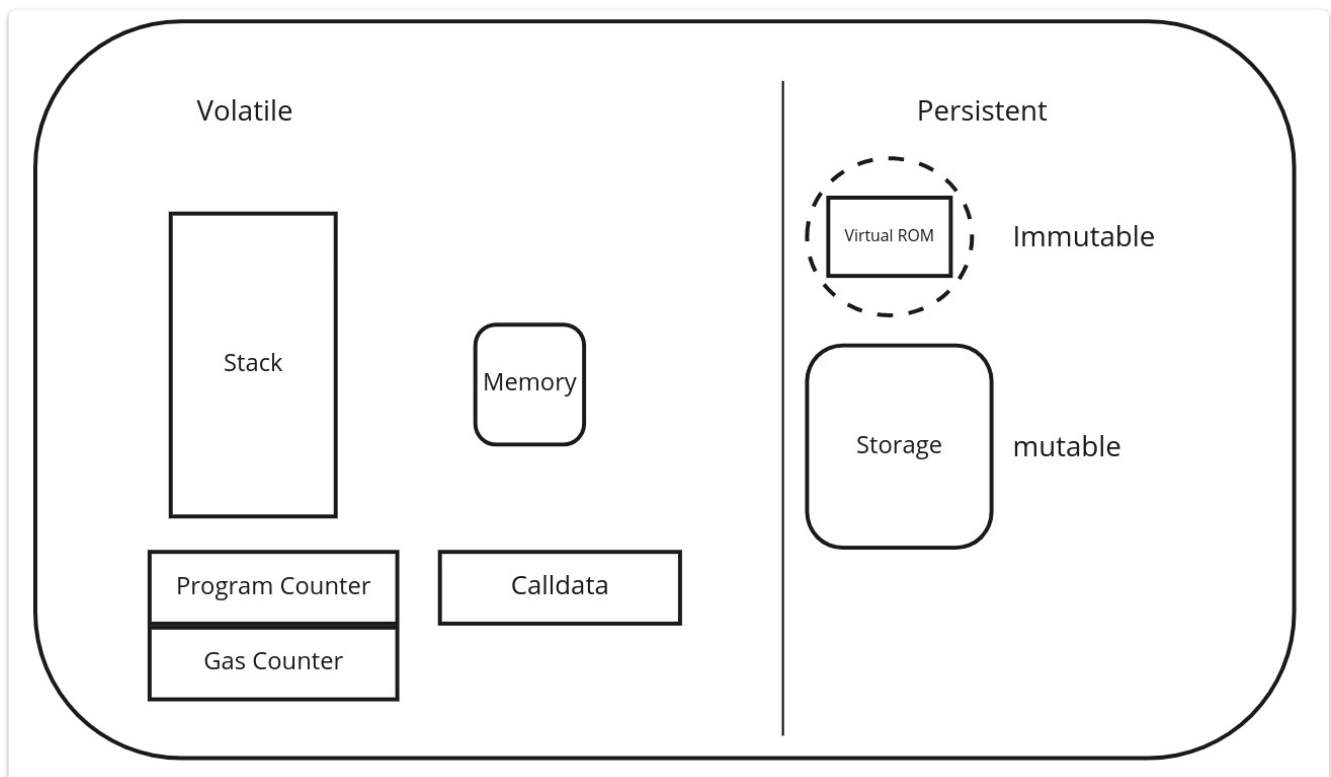
Example Deposit ETH transaction

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

zkEVM Solutions



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.

AppliedZKP divides proofs into two types:

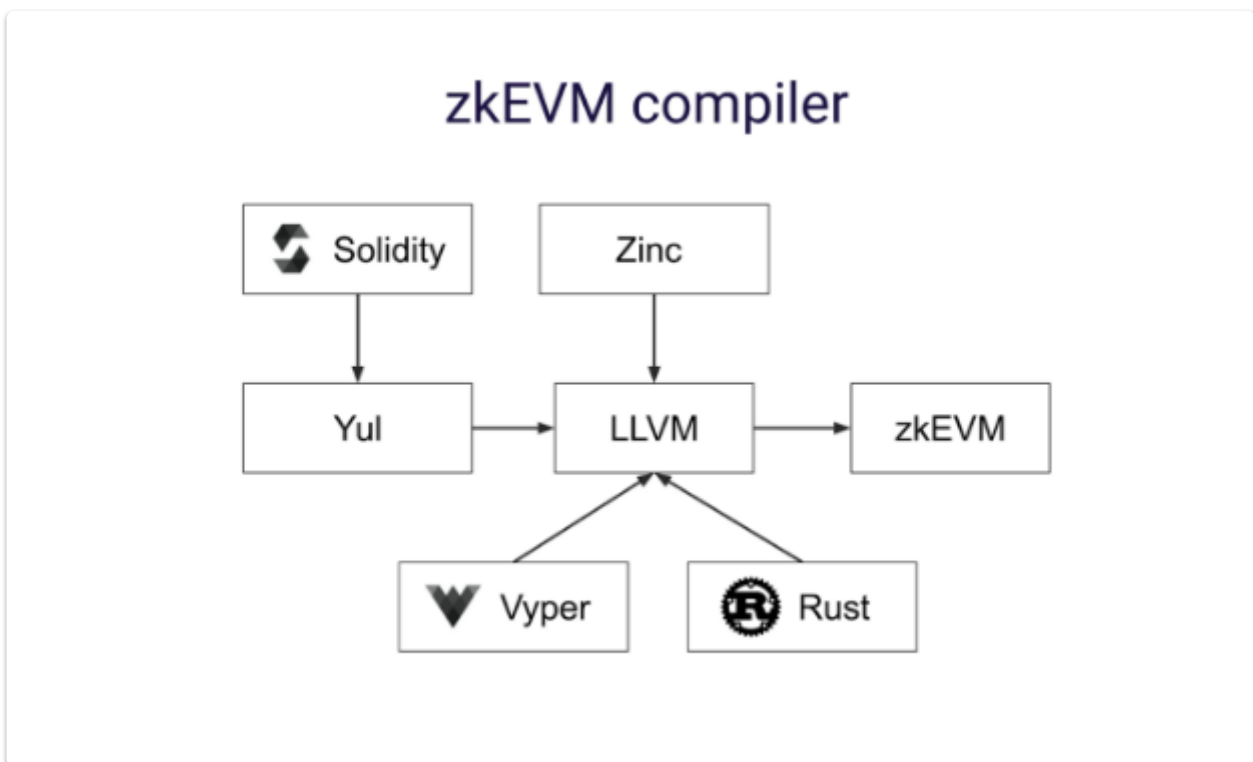
1. State proof, used to check the correctness of the state transition in Stack/Memory/Storage.
2. EVM proof, used to check that the correct opcode is used at the correct time, the correctness of the opcode itself, the validity of the opcode, and all the abnormal conditions (such as out_of_gas) that may be encountered during the execution of the opcode.

Matter Labs zkEVM

zkEVM is a virtual machine that executes smart contracts in a way that is compatible with zero-knowledge-proof computation.

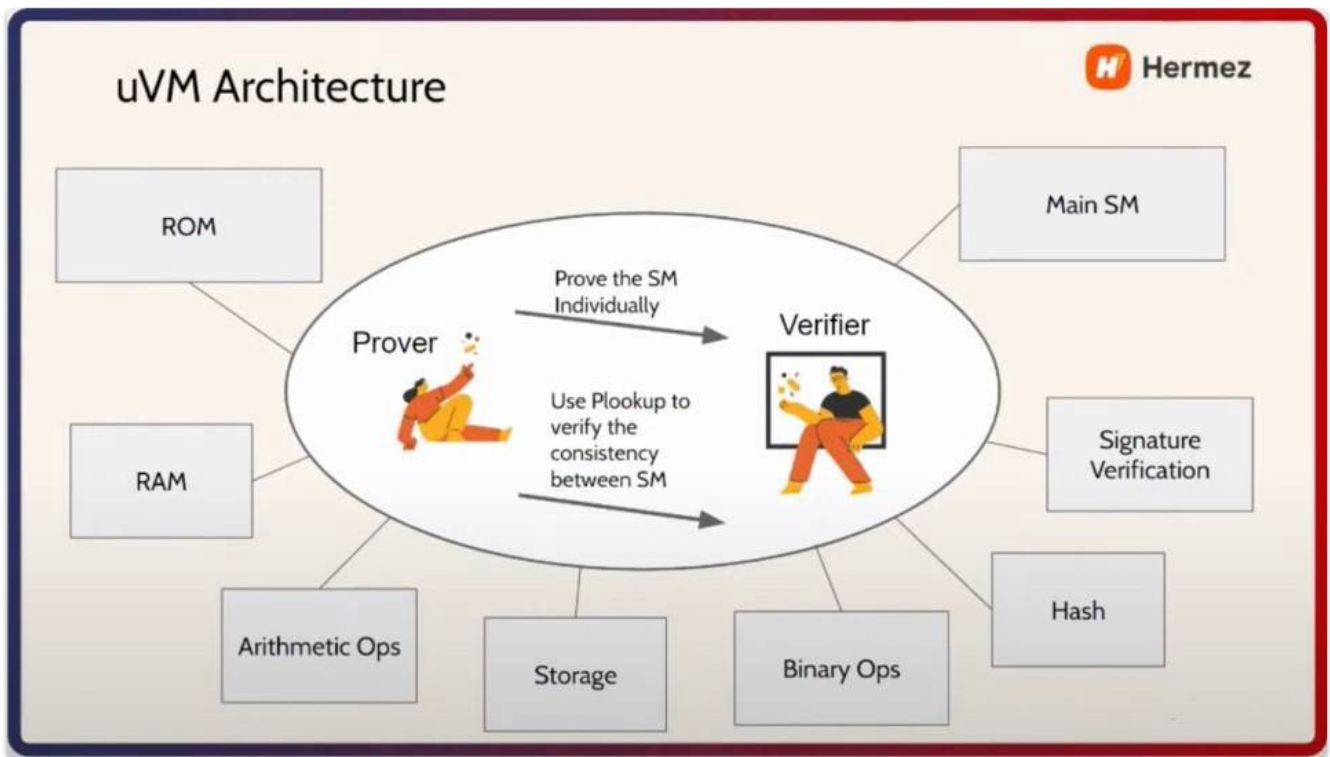
zk-EVM keeps EVM semantics, but is also ZK-friendly and takes on traditional CPU architectures.

zkSync's zkEVM is not a replica of 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.



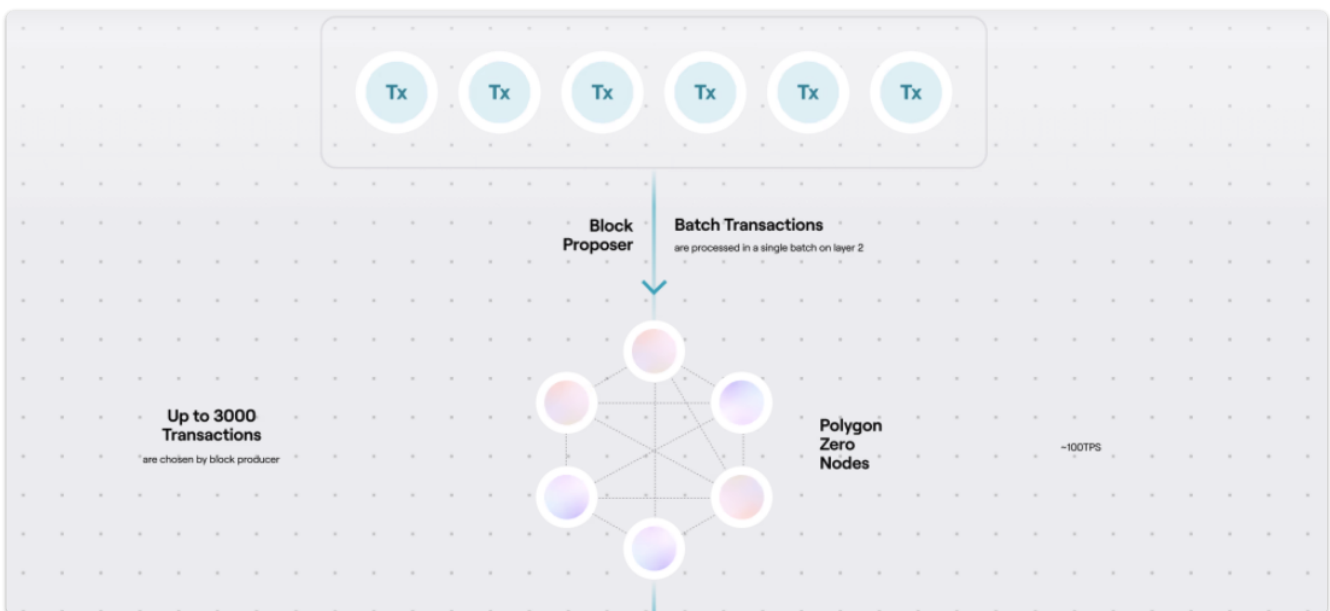
The circuit implementation of Matter Labs uses TinyRAM to implement ordinary opcodes, such as ADD, PUSH, etc.; opcodes that consume a lot of gas, such as SHA256/keccak, implement this circuit especially; finally, Matter Labs uses recursive aggregation technology to aggregate all proofs into one proof.

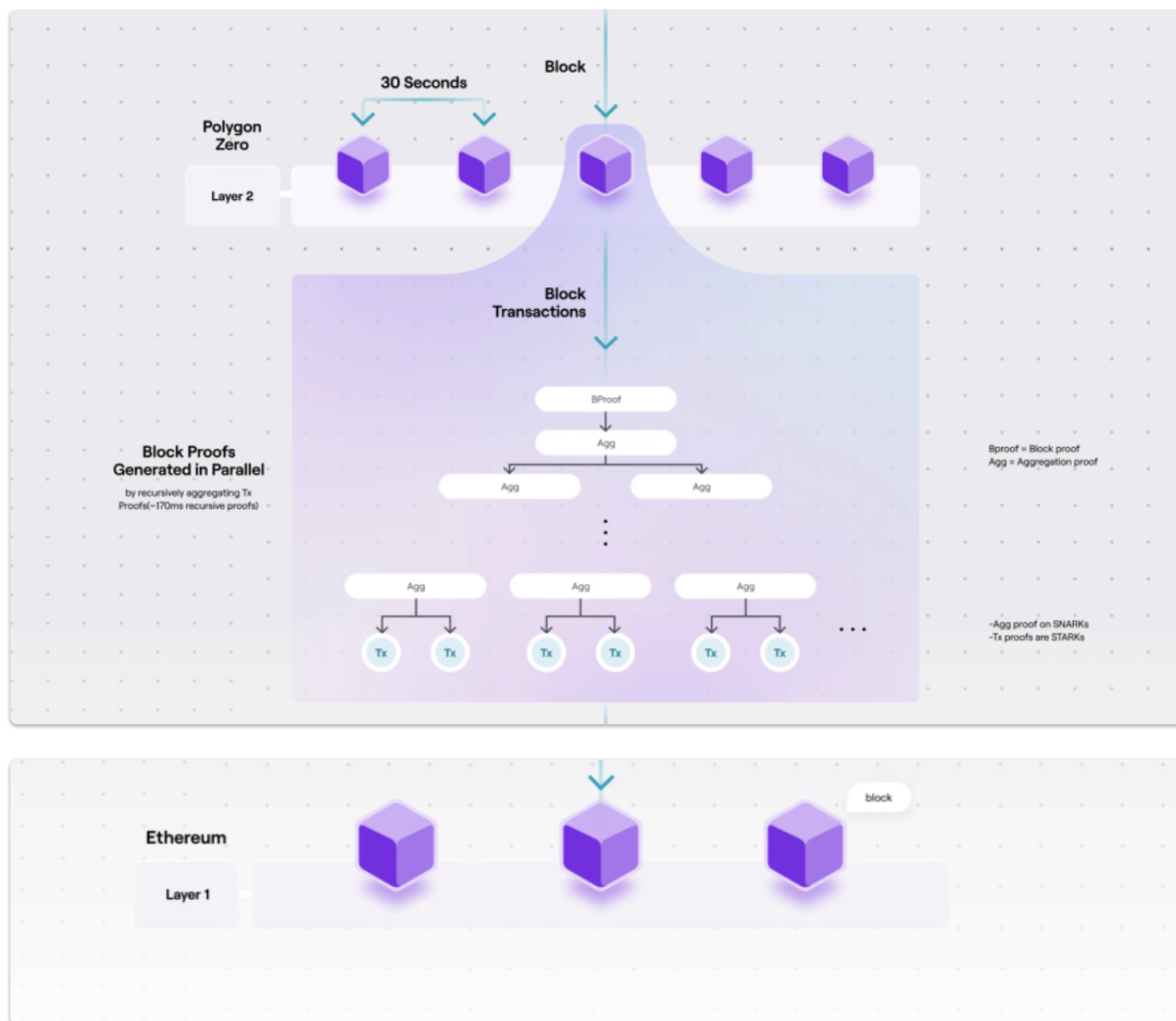
HERMEZ (POLYGON)



(Hermes zkEVM is that it uses two proof systems at the same time, specifically generating a STARK proof and then using PLONK or Groth16 to generate a proof of the STARK proof and verify it on L1, which is like a proof of the proof. The reason for this is that while STARK is excellent, the proof size is large and the cost of verification on the chain is high, while Groth16 or PLONK has a smaller proof size and faster verification speed.)

Polygon Zero





PLONKY2

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. If CALLDATA is repriced in EIP-4488, the verification cost of a plonky2 proof will drop to between

170-200k gas, which could make it not only the fastest proving system, but also the cheapest to verify on Ethereum.

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 facilitate 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.

Hermes uses a coordinator on L2 to aggregate transfers to other L2s
