ZK Rollups review

Zero Knowledge Proofs

A technology allowing 2 abilities

1. Proof of an interaction, that keeps the details private
2. A succint proof of validity of computation

ZKPs give us both of these, but for scalability we are only interested in the second aspect.

ZCash for example uses both aspects.
Aztec also integrates privacy into it’s solution.

Scalability Solutions

FIGURE 2. Taxonomy and comparison of blockchain scalability solutions.

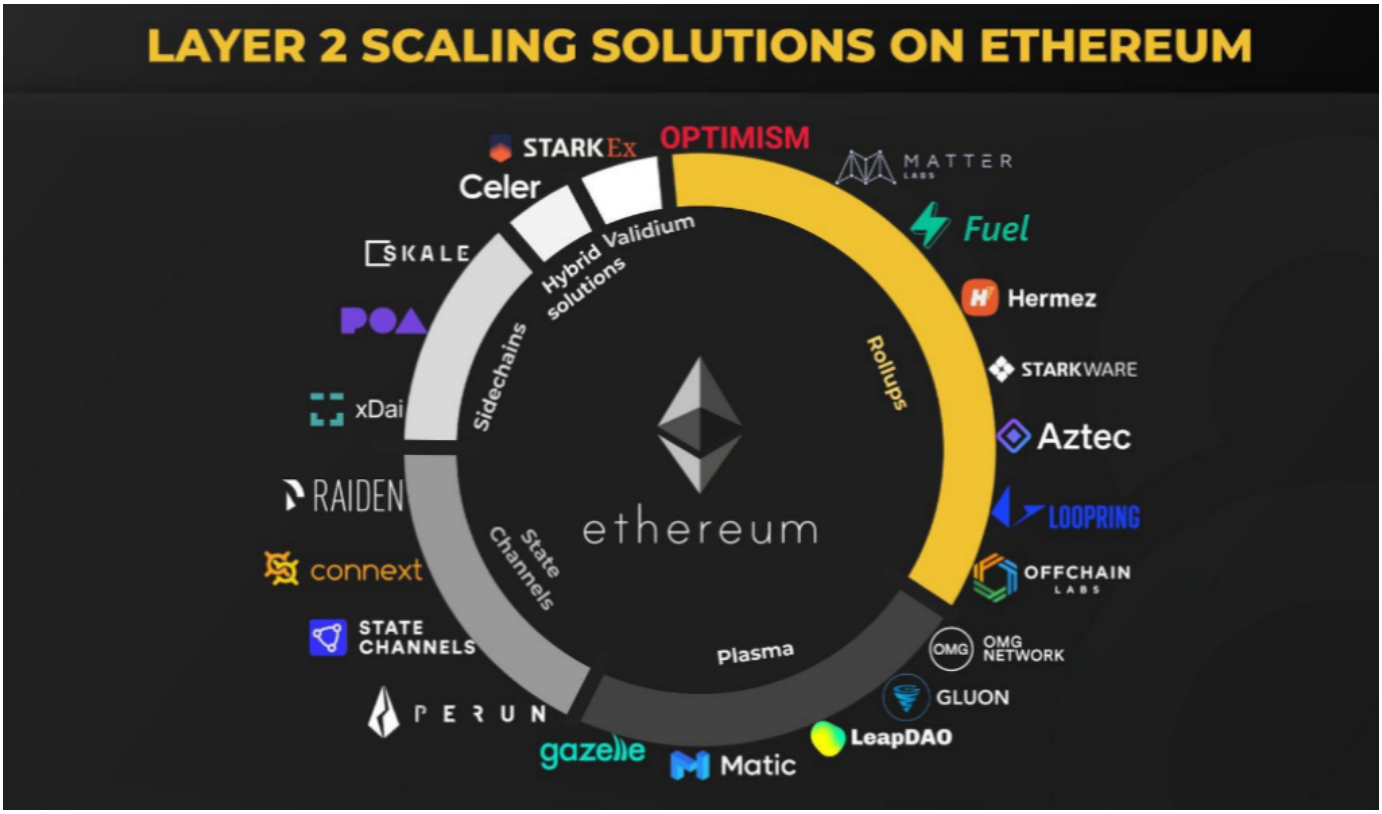
From Scaling Blockchains: A Comprehensive Survey by Hafid et al.

“The decentralization of a system is determined by the ability of the weakest node in the network to verify the rules of the system.” - Georgios Konstantopoulos

In Ethereum there is a goal to keep the hardware requirements low.

Off chain Scaling (Layer 2)

Generally speaking, transactions are submitted to these layer 2 nodes instead of being submitted directly to layer 1 (Mainnet). For some solutions the layer 2 instance then batches them into groups before anchoring them to layer 1, after which they are secured by layer 1 and cannot be altered.
A specific layer 2 instance may be open and shared by many applications, or may be deployed by one project and dedicated to supporting only their application.



Problems with Plasma

The implementation of Plasma gives the ability of hundreds of side chain transactions to be processed offline with only a single hash of the side chain block being added to the Ethereum blockchain. Fundamental flaws exist preventing further scalability.
An exit game must be played for a user to withdraw from the side chain which requires side chain users to retain a high amount of data so that enough exists for validation. Also, a lengthy challenge period requires users to stay online or lose reward.

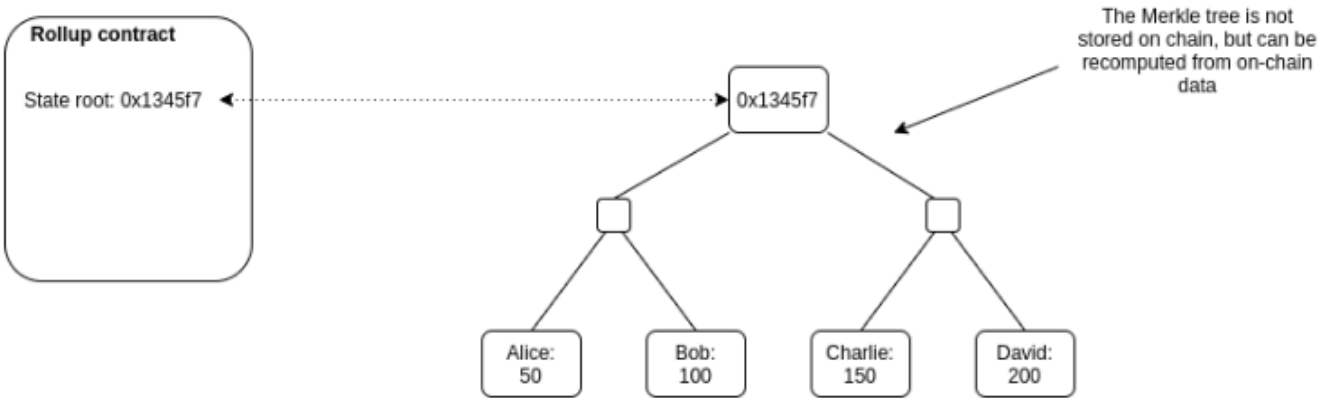
Rollups

Rollups are solutions that have

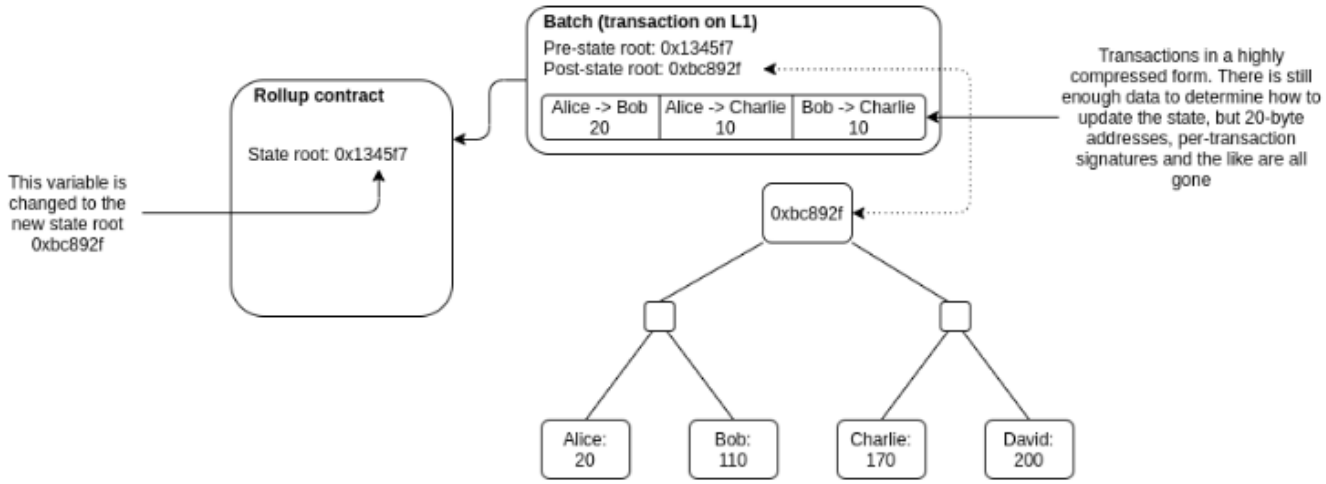
- transaction execution outside layer 1
- data or proof of transactions is on layer 1
- a rollup smart contract in layer 1 that can enforce correct transaction execution on layer 2 by using the transaction data on layer 1

The main chain holds funds and commitments to the side chains
The side chain holds 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.



Anyone can publish a batch, a collection of transactions in a highly compressed form together with the previous state root and the new state root (the Merkle root after processing the transactions). The contract checks that the previous state root in the batch matches its current state root; if it does, it switches the state root to the new state root.



There are currently 2 types of rollups

- Zero Knowledge Proof rollups
- Optimistic rollups

Zero Knowledge Proof Rollups

See Ethworks Report (https://ethworks.io/assets/download/zero-knowledge-blockchain-scaling-ethworks.pdf)

	State channels	Sidechains[0]	Plasma	Optimistic rollups	Validium	zkRollup
Security						
Liveness assumption (e.g. watch-towers)	Yes	Bonded	Yes	Bonded	No	No
The mass exit assumption	No	No	Yes	No	No	No
Quorum of validators can freeze funds	No	Yes	No	No	Yes	No
Quorum of validators can confiscate funds	No	Yes	No	No	Yes [1]	No
Vulnerability to hot-wallet key exploits	High	High	Moderate	Moderate	High	Immune
Vulnerability to crypto-economic attacks	Moderate	High	Moderate	Moderate	Moderate	Immune
Cryptographic primitives	Standard	Standard	Standard	Standard	New	New
Performance / economics						
Max throughput on ETH 1.0	1..∞ TPS [2]	10k+ TPS	1k..9k TPS [2]	2k TPS [3]	20k+ TPS	2k TPS
Max throughput on ETH 2.0	1..∞ TPS [2]	10k+ TPS	1k..9k TPS [2]	20k+ TPS	20k+ TPS	20k+ TPS
Capital-efficient	No	Yes	Yes	Yes	Yes	Yes
Separate onchain tx to open new account	Yes	No	No	No	No	No [5]
Cost of tx	Very low	Low	Very low	Low	Low	Low
Usability						
Withdrawal time	1 confirm.	1 confirm.	1 week [4] ([7])	1 week [4] ([7])	1..10 min [7]	1..10 min [7]
Time to subjective finality	Instant	N/A (trusted)	1 confirm.	1 confirm.	1..10 min	1..10 min
Client-side verification of subjective finality	Yes	N/A (trusted)	No	No	Yes	Yes
Instant tx confirmations	Full	Bonded	Bonded	Bonded	Bonded	Bonded
Other aspects						
Smart contracts	Limited	Flexible	Limited	Flexible	Flexible	Flexible
EVM-bytecode portable	No	Yes	No	Yes	Yes	Yes
Native privacy options	Limited	No	No	No	Full	Full

[0] Some researchers do not consider them to be part of L2 space at all, see https://twitter.com/gakonst/status/1146793685545304064

[1] Depends on the implementation of the upgrade mechanism, but usually applies.

[2] Complex limitations apply.

[3] To keep compatibility with EVM throughput must be capped at 300 TPS

[4] This parameter is configurable, but most researchers consider 1 or 2 weeks to be secure.

[5] Depends on the implementation. Not needed in zkSync but required in Loopring.

[7] Can be accelerated with liquidity providers but will make the solution capital-inefficient.

Updated 2021–02–18

Comparison of the types

Property	Optimistic rollups	ZK rollups
Fixed gas cost per batch	~40,000 (a lightweight transaction that mainly just changes the value of the state root)	~500,000 (verification of a ZK-SNARK is quite computationally intensive)
Withdrawal period	~1 week (withdrawals need to be delayed to give time for someone to publish a fraud proof and cancel the withdrawal if it is fraudulent)	Very fast (just wait for the next batch)
Complexity of technology	Low	High (ZK-SNARKs are very new and mathematically complex technology)
Generalizability	Easier (general-purpose EVM rollups are already close to mainnet)	Harder (ZK-SNARK proving general-purpose EVM execution is much harder than proving simple computations, though there are efforts (eg. Cairo) working to improve on this)
Per-transaction on-chain gas costs	Higher	Lower (if data in a transaction is only used to verify, and not to cause state changes, then this data can be left out, whereas in an optimistic rollup it would need to be published in case it needs to be checked in a fraud proof)
Off-chain computation costs	Lower (though there is more need for many full nodes to redo the computation)	Higher (ZK-SNARK proving especially for general-purpose computation can be expensive, potentially many thousands of times more expensive than running the computation directly)

Proofs

Optimistic rollups use fraud proofs: the rollup contract keeps track of its entire history of state roots and the hash of each batch.
If anyone discovers that one batch had an incorrect post-state root, they can publish a proof to chain, proving that the batch was computed incorrectly. The contract verifies the proof, and reverts that batch and all batches after it.

ZK rollups use validity proofs: every batch includes a cryptographic proof which could be a ZKSNARK (eg. using the PLONK protocol), or a STARK which proves that the post-state root is the correct result of executing the batch. No matter how large the computation, the proof can be very quickly verified on-chain.

Transaction Compression

How does compression work?

A simple Ethereum transaction (to send ETH) takes ~110 bytes. An ETH transfer on a rollup, however, takes only ~12 bytes:

Parameter	Ethereum	Rollup
Nonce	~3	0
Gasprice	~8	0-0.5
Gas	3	0-0.5
To	21	4
Value	~9	~3
Signature	~68 (2 + 33 + 33)	~0.5
From	0 (recovered from sig)	4
Total	~112	~12

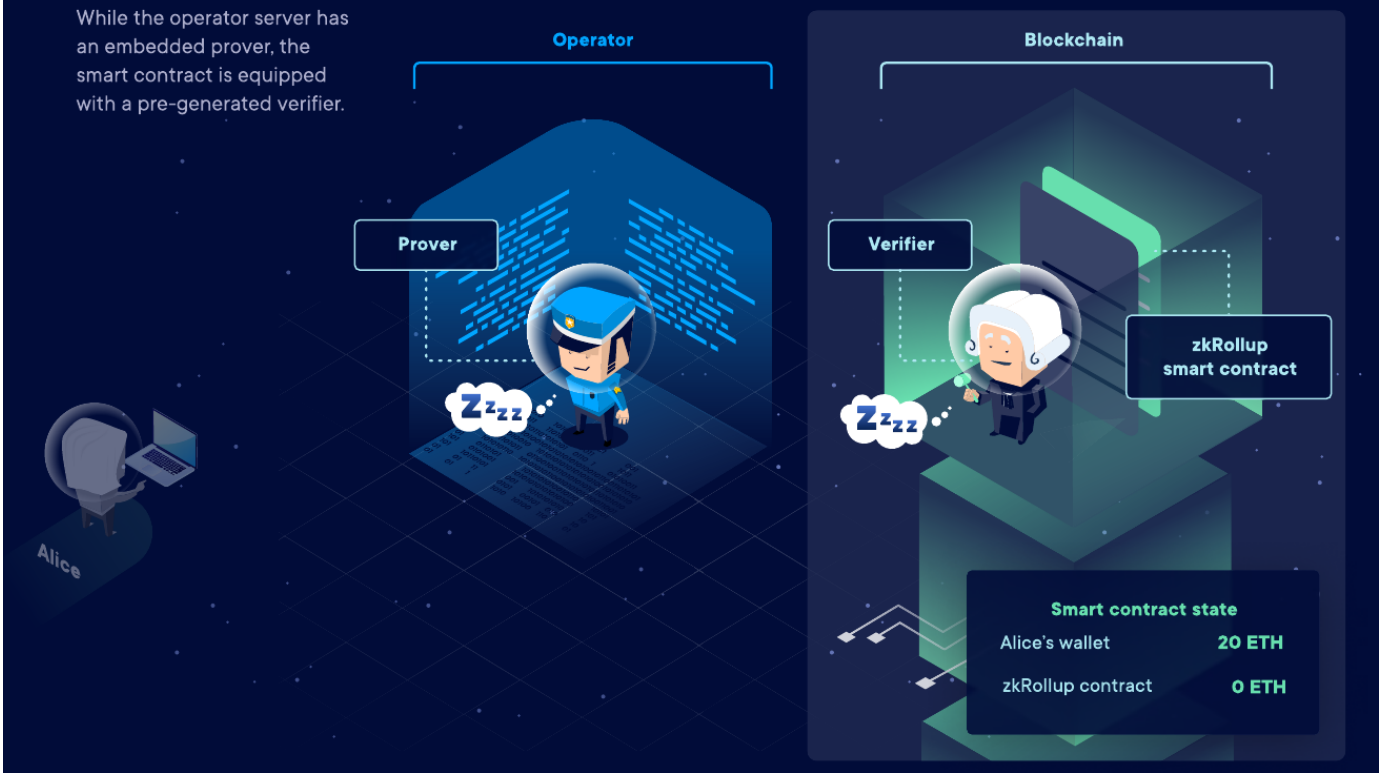
Part of this is simply superior encoding: Ethereum's RLP wastes 1 byte per value on the length of each value. But there are also some very clever compression tricks that are going on:

ZK Rollup Process

From Ethworks (https://ethworks.io/assets/download/zero-knowledge-blockchain-scaling-ethworks.pdf)

01. Start

While the operator server has an embedded prover, the smart contract is equipped with a pre-generated verifier.

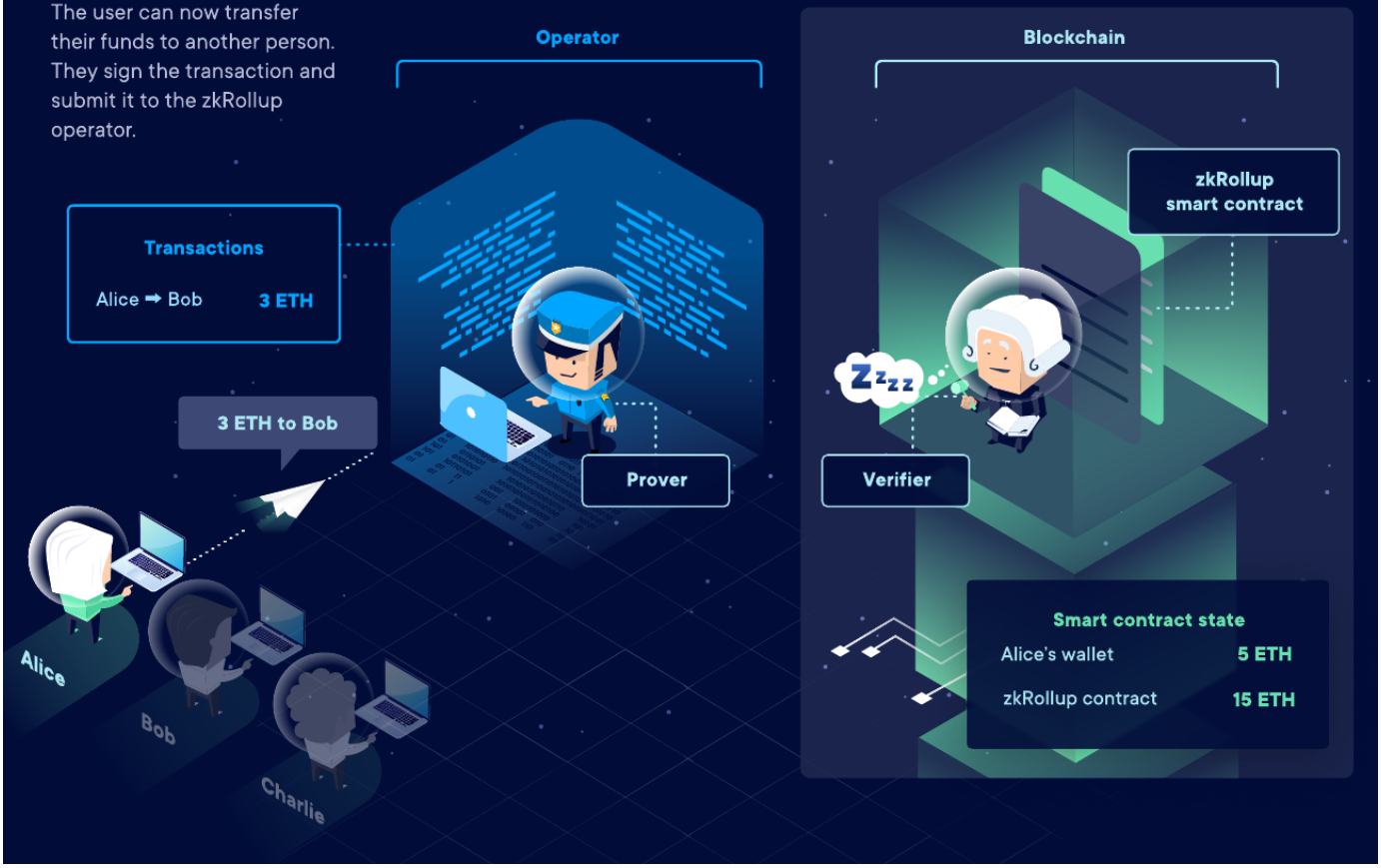


02. Alice's Enter

To enter the system, the user needs to transfer their funds to the zkRollup. The assets are sent to a smart contract.

03. Alice's Transfer

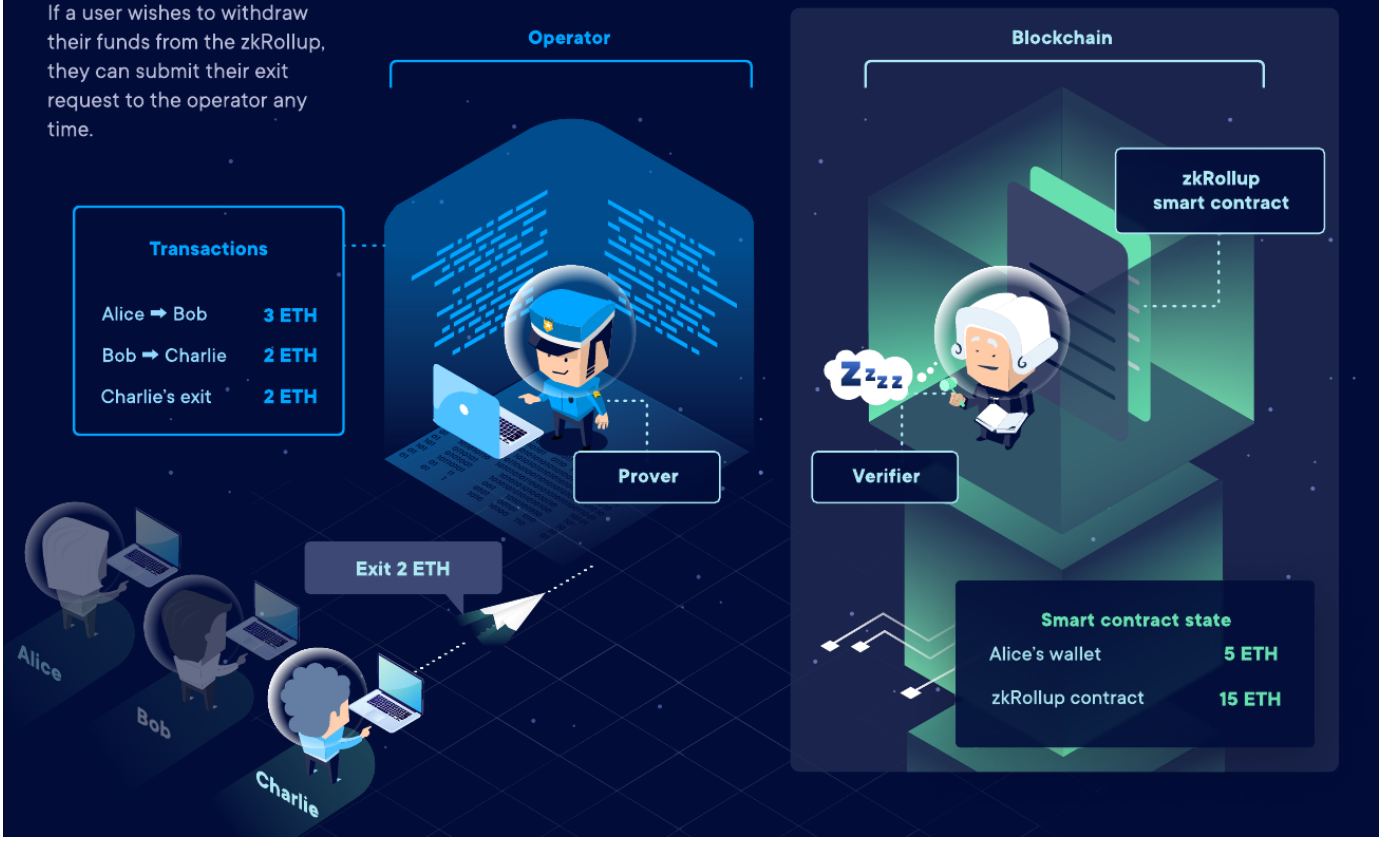
The user can now transfer their funds to another person. They sign the transaction and submit it to the zkRollup operator.



04. Bob's Transfer

05. Charlie's Exit

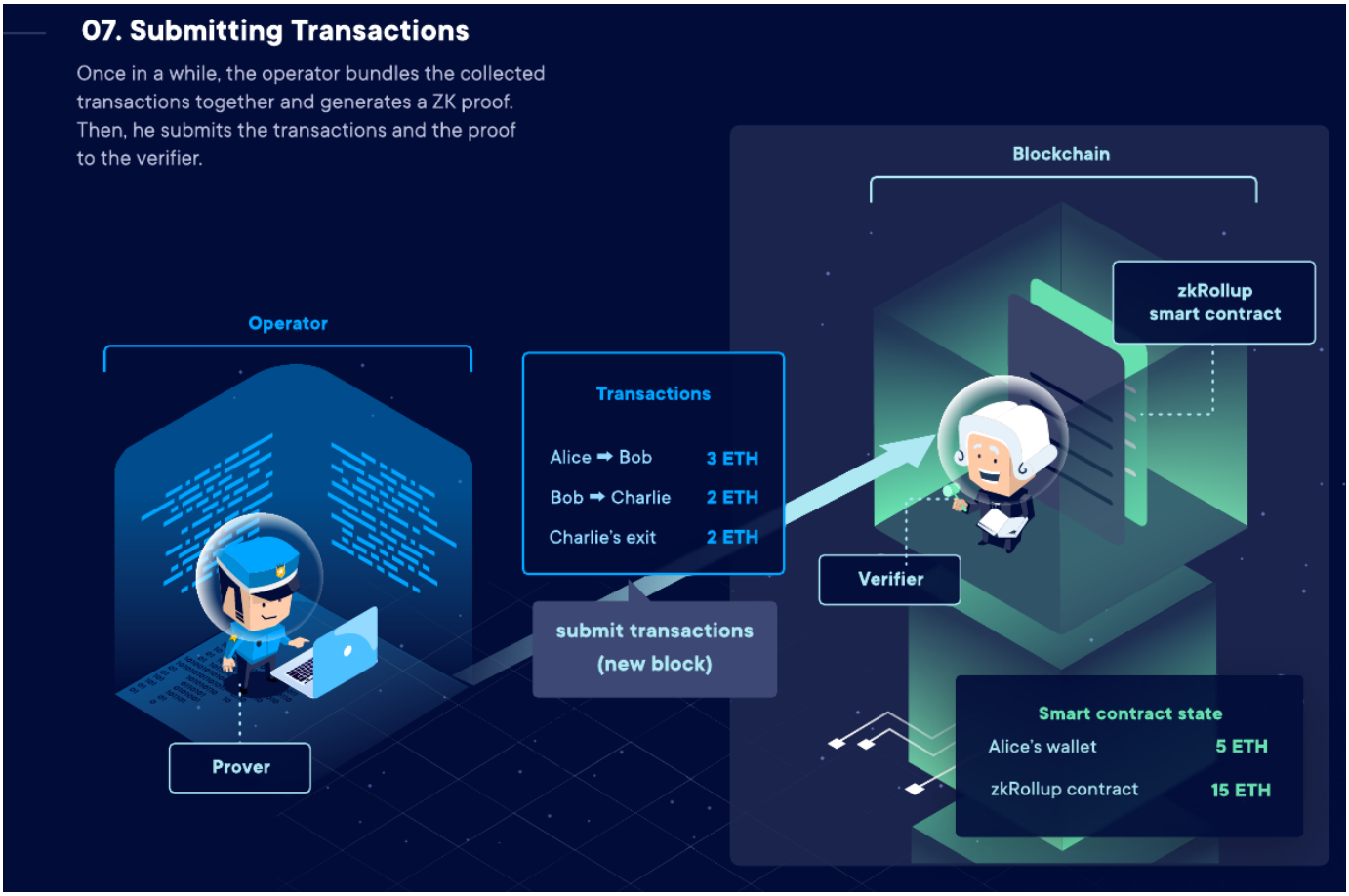
If a user wishes to withdraw their funds from the zkRollup, they can submit their exit request to the operator any time.

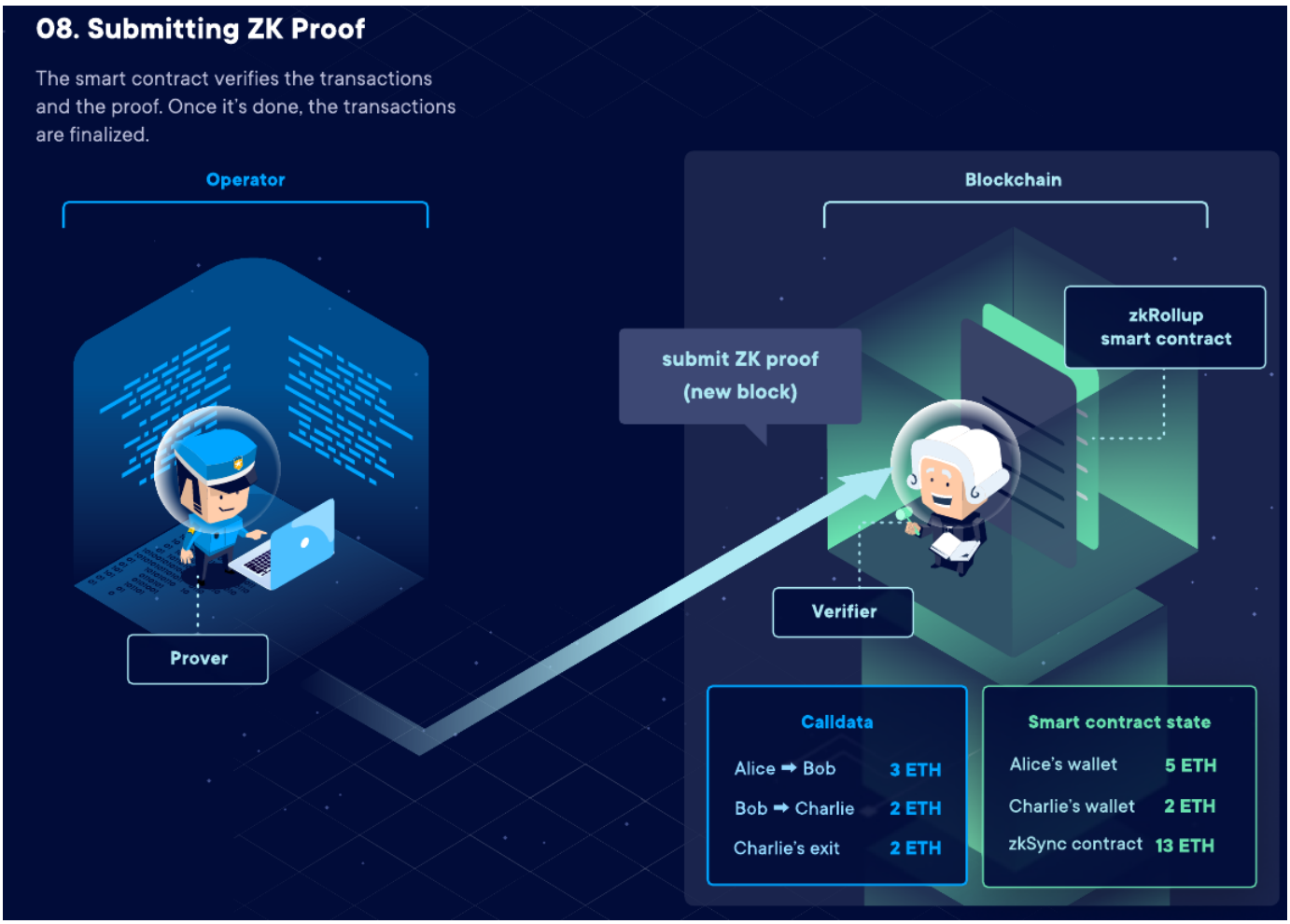


06. Collecting Transactions

In the meantime, the operator collects transactions and exit requests from many users.

* Note that even if Bob and Charlie didn't have any funds on the zkRollup, they could still receive transfers from other users.



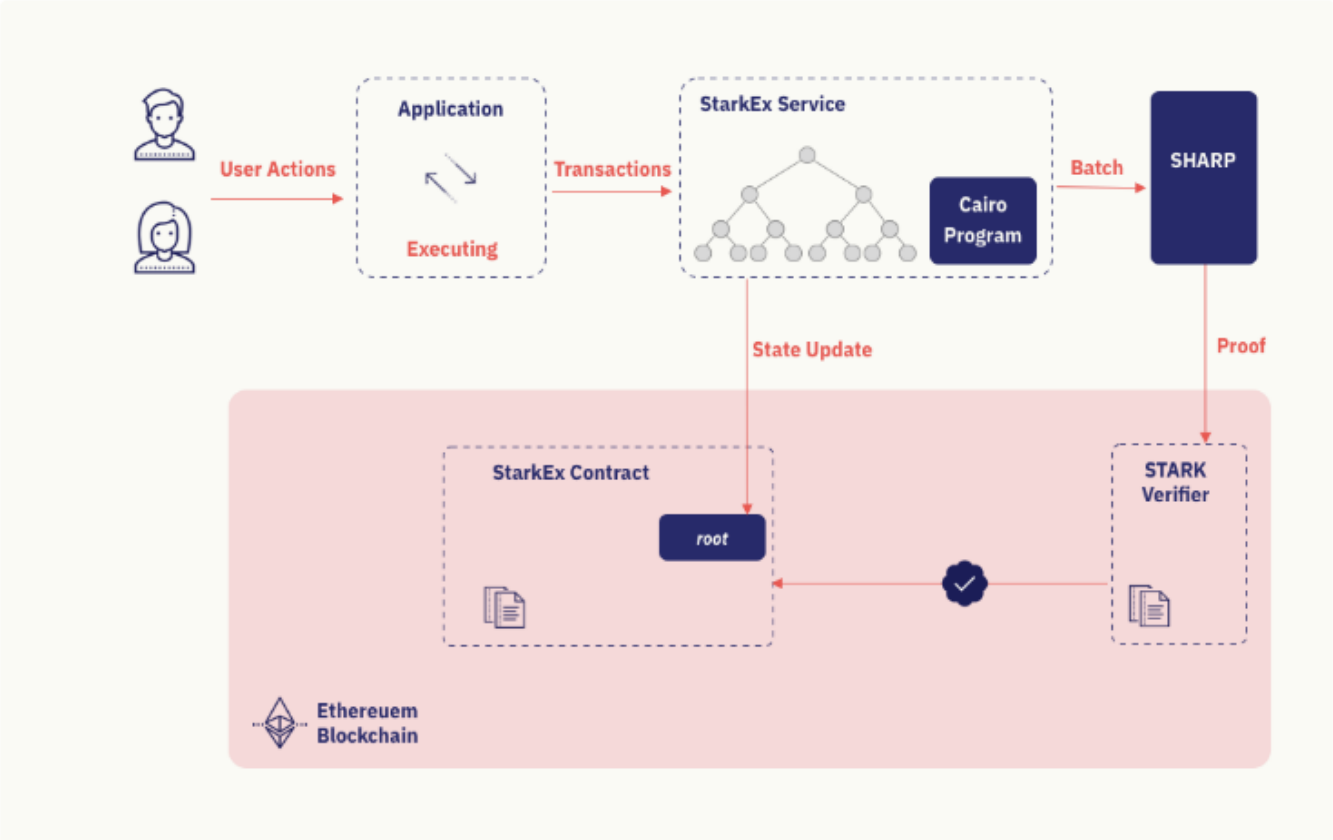


ZK Rollup Projects

StarkEx

StarkEx can store data according to a range of user and application needs.
In ZK-Rollup mode data is published on-chain. In Validium mode data is stored off-chain. Volition is a hybrid data availability mode, where the user can choose whether to place data on-chain or off-chain.

There are on chain and off chain components



Off chain

The off-chain component holds the state of orders, performs transaction executions in the system, and sends state updates to the on-chain component.

On Chain

The on-chain component holds the state commitments and the system assets; and is responsible for enforcing the validity of state transition. In the case of StarkEx for spot trading, it also manages the on-chain accounts, which are useful in the context of Layer 1 (L1) dApp interoperability and DeFi pooling.

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.

zkSync

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.

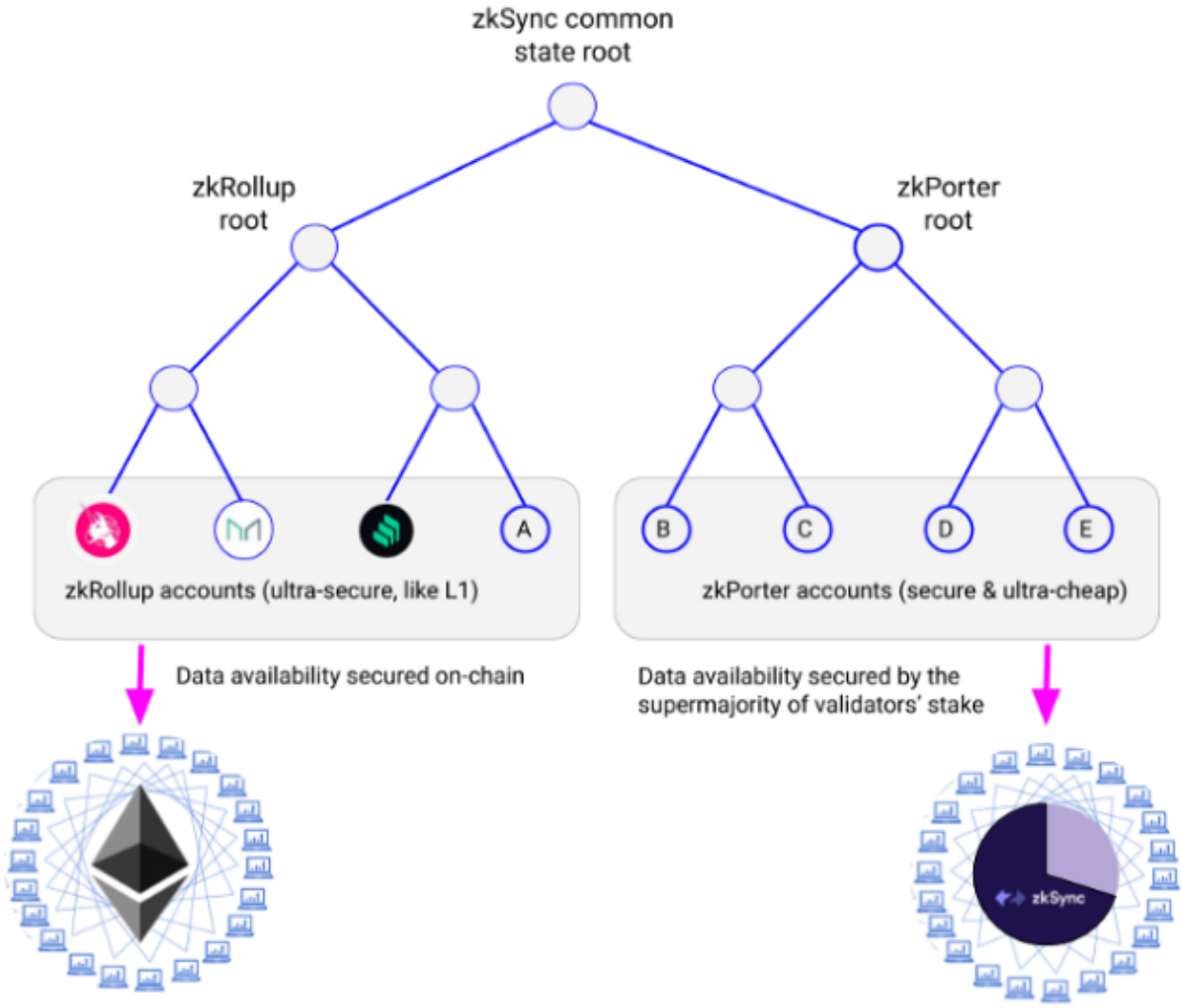
3. Withdrawing funds
    - Withdraw transaction (L2 -> Ethereum address)

- 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.



See zkPorter (https://medium.com/matter-labs/zkporter-a-breakthrough-in-l2-scaling-ed5e48842fbf)

zkEVM

zkEVM is a virtual machine that executes smart contracts in a way that is compatible with zero-knowledge-proof computation. It is the key to building an EVM-compatible ZK Rollup while preserving the battle-tested code and knowledge gained after years of working with Solidity. Our zk-EVM keeps EVM semantics, but is also ZK-friendly and takes on traditional CPU architectures.

For zkEVM, there are currently two main implementation strategies.

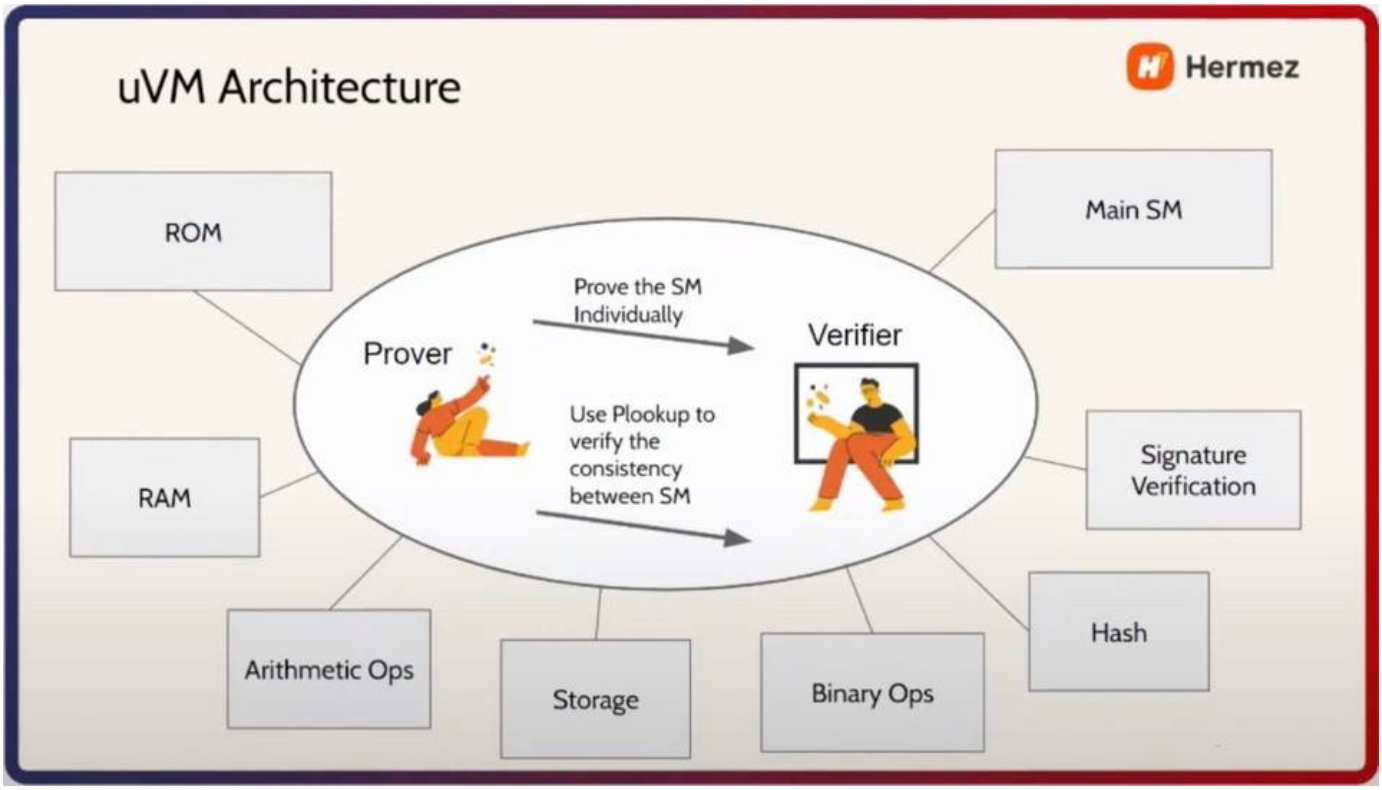
- Direct support for the existing set of EVM opcodes, which is fully compatible with the set of Solidity opcodes. Those using this solution include Hermez and the Ethereum Foundation zkEVM.
- Maintaining Solidity compatibility by designing a new virtual machine that is zero-knowledge proof-friendly and adapting EVM development tools. This solution is mainly used by zkSync

See zkEVM Solutions (https://hackernoon.com/exploring-popular-zkevm-solutions-appliedzkp-matter-labs-hermez-and-sin7y-quick-publication-ltq37ah)

zkEVM Solutions

	A	B	C	D	E
1	**zkEVM**	**EVM Support**	**Contract Code Interpretation Path**	**General Circuit Implementation**	**Password Tool**
2	AppliedZKP zkEVM	Implement zk for the native opcode of EVM one by one.	Solidity/Vyper -> zkEVM	custom constraint	halo2+KZG10+BN256
3	Matter Labs zkEVM	Custom EVM, implement the interpreter, compile the contract code into YUL and compile YUL into custom bytecode supported by zkEVM.	Solidity/Vyper -> YUL -> LLVM -> zkEVM	TinyRAM	utralPLONK
4	Hermez zkEVM	Custom EVM, implement uVM, and compile the contract code into the micro instruction set supported by uVM.	Solidity/Vyper -> micro Op -> uVM	*	PLONK+KZG+Groth16
5	Sin7Y zkEVM	Implement zk for the native opcode of EVM one by one, and optimize specialized opcode.	Solidity/Vyper -> zkEVM	custom gate/TinyRAM	halo2+KZG10+RecursivePLONK

Hermez (Polygon)



(Hermez 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.)

zkSync EVM

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.

zkEVM compiler