

Lesson 3

Week 1

Lesson 1 - Introduction to Blockchain and Layer 1

Lesson 2 - Why Scalability

Lesson 3 - Introduction to Layer 2

Lesson 4 - Maths and Cryptography

Today's Topics

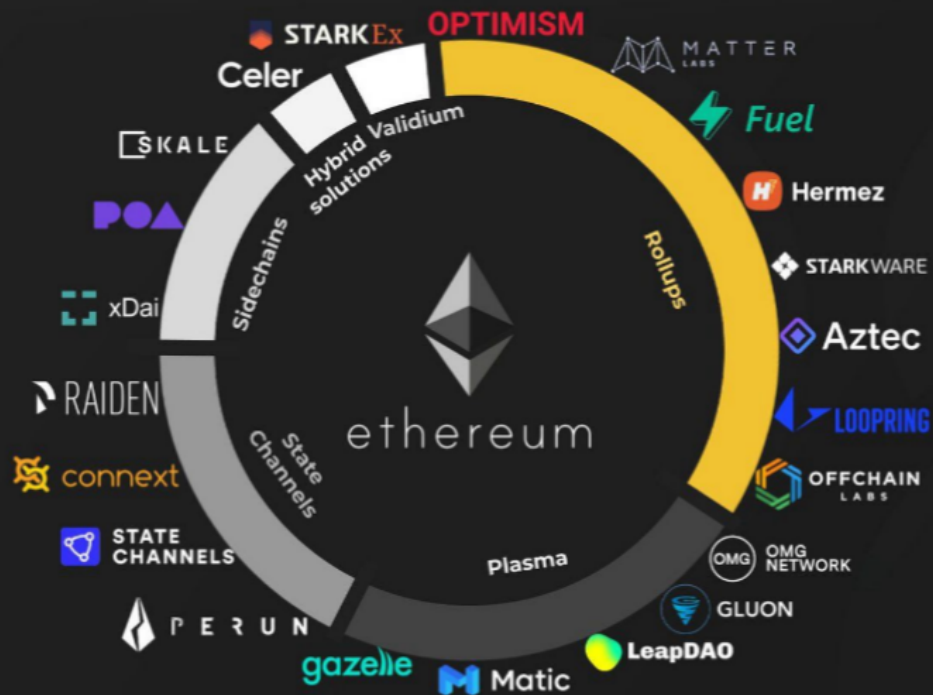
- Off chain scaling introduction
 - Bridges versus layer 2
 - Rollups in more detail
 - Introduction to zkEVM solutions
 - Data availability
 - (Proto) Danksharding
-

Off chain Scaling

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.

LAYER 2 SCALING SOLUTIONS ON ETHEREUM



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.

There are currently 2 types of rollups

- Zero Knowledge Proof rollups
 - Optimistic rollups
-

ZKP or validity Rollups

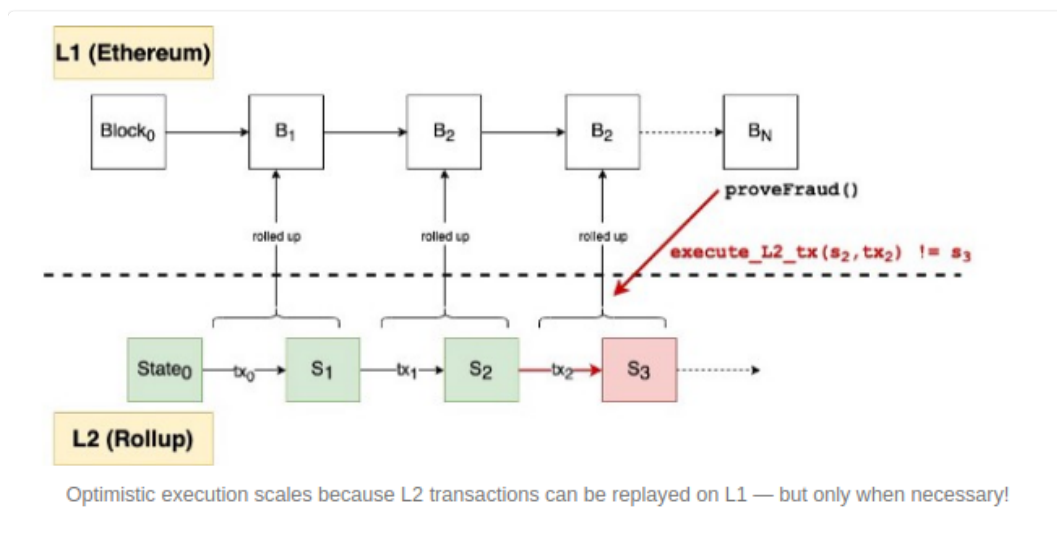
These rollups rely on a proof of the correctness of the execution that produces the rollup block state transition being supplied to a validator contract on L1.

The state transition on L2 will not be regarded as valid unless this proof has been validated.

Although we use a zero knowledge proof, the zero knowledge aspect is usually ignored, the inputs and data involved is usually public, the focus is on the correctness of computation. For this reason some people prefer the term validity proof.

Optimistic Rollups

The name Optimistic Rollups originates from how the solution works. 'Optimistic' is used because aggregators publish only the bare minimum information needed with no proofs, assuming the aggregators run without committing frauds, and only providing proofs in case of fraud. 'Rollups' is used because transactions are committed to main chain in bundles (that is, they are rolled-up).



See this [article](#) for further discussion of the differences between these types of rollups.

We shall go into more detail about the differences between these types of rollups in lesson 7

Bridges vs Layer 2

Bridges between chains are typically less functional and more specialised than Layer 2 solutions.

They usually provide the ability to transfer tokens atomically between chains.

One mechanism used for this is 'lock and mint' bridging, in which tokens are locked or burned on the sending chain and an equivalent value of tokens is minted on the receiving chain.

Bridging is quite widespread, for example see the Wormhole [project](#) which is used by over 25 chains.

Rollups in detail

From Ethereum [Docs](#)

Optimistic rollup operators bundle multiple off-chain transactions together in large batches before submitting to Ethereum. This approach enables spreading fixed costs across multiple transactions in each batch, reducing fees for end-users. Optimistic rollups also use compression techniques to reduce the amount of data posted on Ethereum.

If the fraud proof succeeds, the rollup protocol re-executes the transaction(s) and updates the rollup's state accordingly. The other effect of a successful fraud proof is that the sequencer responsible for including the incorrectly executed transaction in a block receives a penalty.

If the rollup batch remains unchallenged (i.e., all transactions are correctly executed) after the challenge period elapses, it is deemed valid and accepted on Ethereum. Others can continue to build on an unconfirmed rollup block, but with a caveat: transaction results will be reversed if based on an incorrectly executed transaction published previously.

Process

- Developer sends transaction off-chain to a bonded aggregator
- Anyone with a bond may become an aggregator.
- There are multiple aggregators on the same chain.
- Fees are paid however the aggregator wants (account abstraction / meta transactions).
- Developer gets an instant guarantee that the transaction will be included or else the aggregator loses their bond.
- Aggregator locally applies the transaction & computes the new state root.
- Aggregator submits an Ethereum transaction (paying gas) which contains the transaction & state root (an optimistic rollup block).

- If anyone downloads the block & finds that it is invalid, they may prove the invalidity with `verify_state_transition(prev_state, block, witness)` which:
 - Slashes the malicious aggregator & any aggregator who built on top of the invalid block.
 - Rewards the prover with a portion of the aggregator's bond.

From [explanation](#) by Kelvin Fichter

A level 1 fault proof system is a system that has an admin that can upgrade the system within the challenge window and can only be used by the admin.

A level 2 fault proof system still has an admin that can upgrade within the challenge window but is permissioned to allow a few others (besides the admin/team itself) to run the fault proof.

A level 3 fault proof is permissionless but still has an admin that can execute an upgrade within the challenge window. At level 3, you only need to trust that the admin won't do anything malicious and won't be compromised.

Level 4 proofs are completely permissionless and cannot be upgraded before users have a chance to withdraw their funds.

Level 4 fault proofs are the holy grail for an Optimistic Rollup.

OPTIMISM

TOOLS ▾ DEVELOPER ▾ COMMUNITY ▾ INTEGRATIONS

🐦 🗨️ 💬

Use live apps on Optimistic Ethereum today.

Transact in milliseconds, save 10-100x on fees.

DEPOSIT NOW

USER GUIDE

2.2M+

Transactions Processed

150+

Verified Contracts

\$100M+


Saved Gas Fees

100k+


Unique Addresses

Building Arbitrum for Secure Ethereum Dapps.

Experience economical efficiency of the blockchain without limits.



ARBITRUM



Zero Knowledge Proof Rollups

See [Ethworks Report](#)

Note that in this context the proofs produced are often referred to as validity proofs since the zero knowledge aspect is not required.

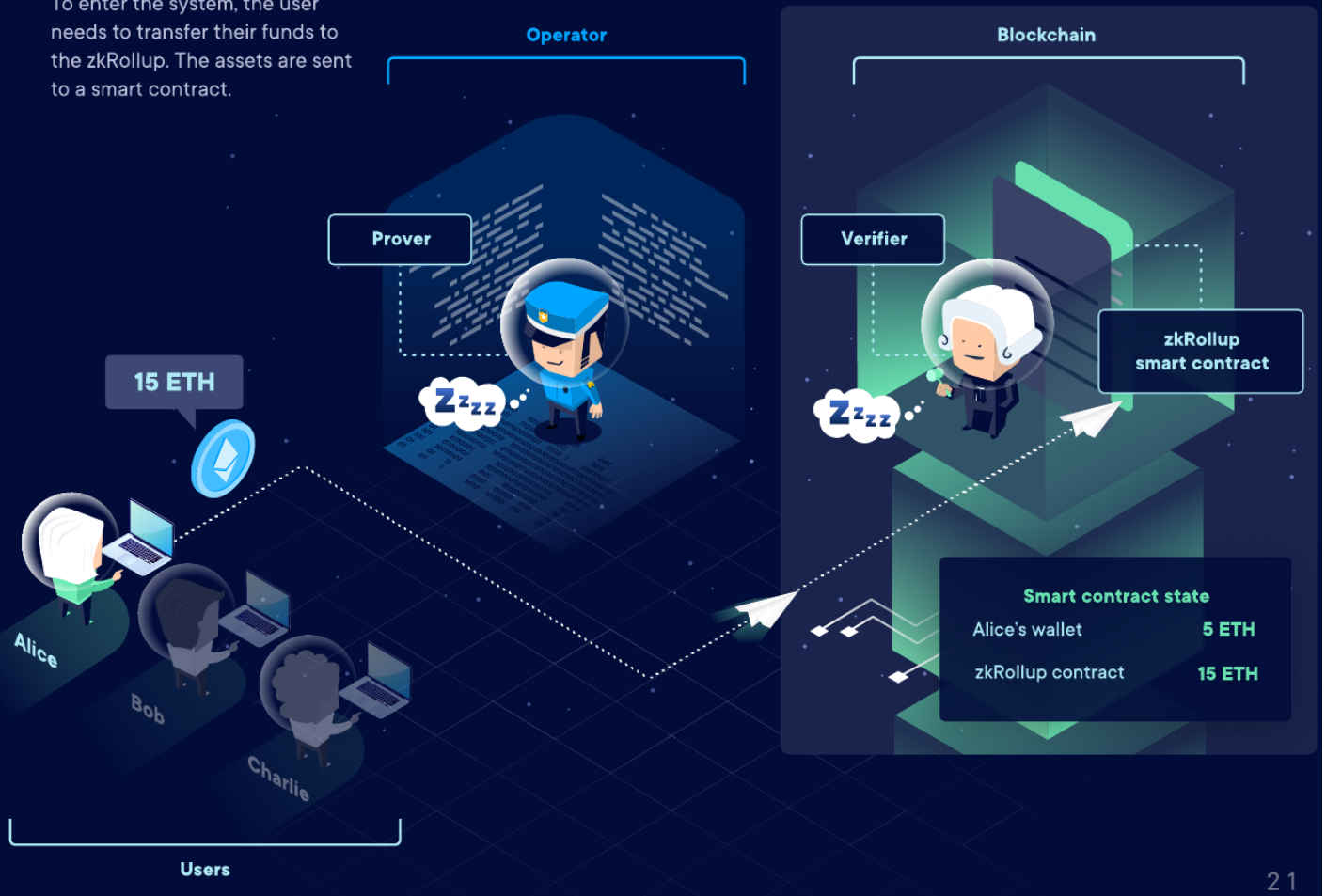
ZK Rollup Process

From [Ethworks](#)



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.



21

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

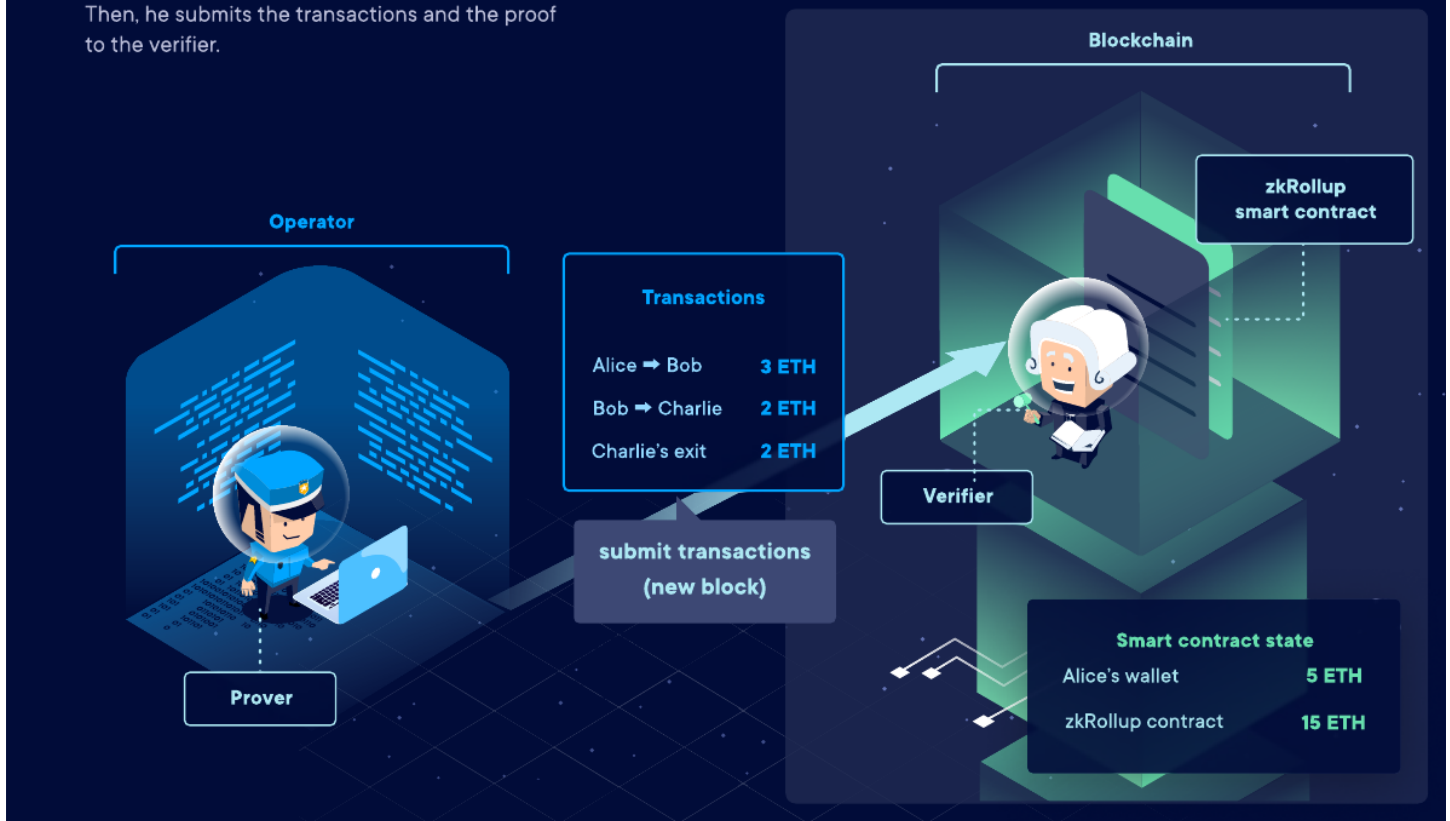
In the meantime, the operator collects transactions and exit requests from many users.

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



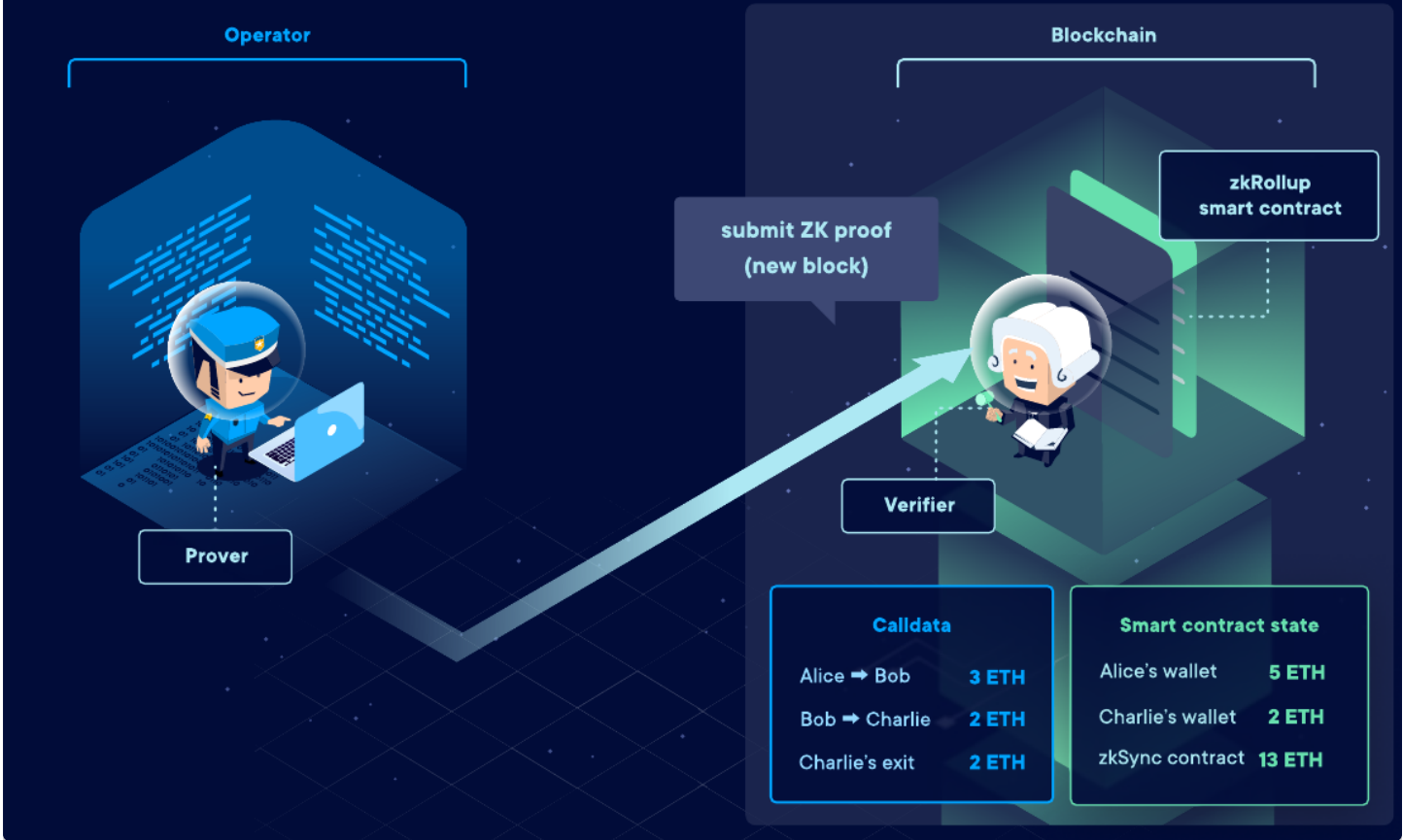
07. Submitting Transactions

Once in a while, the operator bundles the collected transactions together and generates a ZK proof. Then, he submits the transactions and the proof to the verifier.



O8. Submitting ZK Proof

The smart contract verifies the transactions and the proof. Once it's done, the transactions are finalized.



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:

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



See [Docs](#)

StarkNet is currently in ZK-Rollup mode (see above). This means that upon the acceptance of a state update on-chain, the state diff between the previous and new state is sent as calldata to Ethereum.

This data allows anyone that observes Ethereum to reconstruct the current state of StarkNet. Note that to update the StarkNet state on L1, it suffices to send a valid proof — without information on the transactions or particular changes that this update caused. Consequently, more information must be provided in order to allow other parties to locally track StarkNet's state.

zkEVM Introduction

A zkEVM is a **virtual machine** designed and developed to **emulate the Ethereum Virtual Machine (EVM)** by recreating all existing EVM opcodes for transparent deployment of existing Ethereum smart contracts.

The zkEVM, acts as a state machine, processes state transitions stemming from the execution of Ethereum's Layer 2 transactions, which users transmit to the network.

After this, it generates validity proofs that confirm the accuracy of the off-chain state change computations.

Build general DApps in zk-Rollup

From Scroll [Docs](#)

There are two ways to build general DApps in zk-Rollups.

- One is building application-specific circuit ("ASIC") for different DApps.
- The other is building a universal "EVM" circuit for smart contract execution.

Here circuit refers to the program representation used in a zero knowledge proof.

The circuit form only supports very limited expressions (for example a basic rank 1 constraint system (R1CS) only supports addition and multiplication).

It is then difficult for a developer to build the program logic using the circuit language, using just addition and multiplication operations

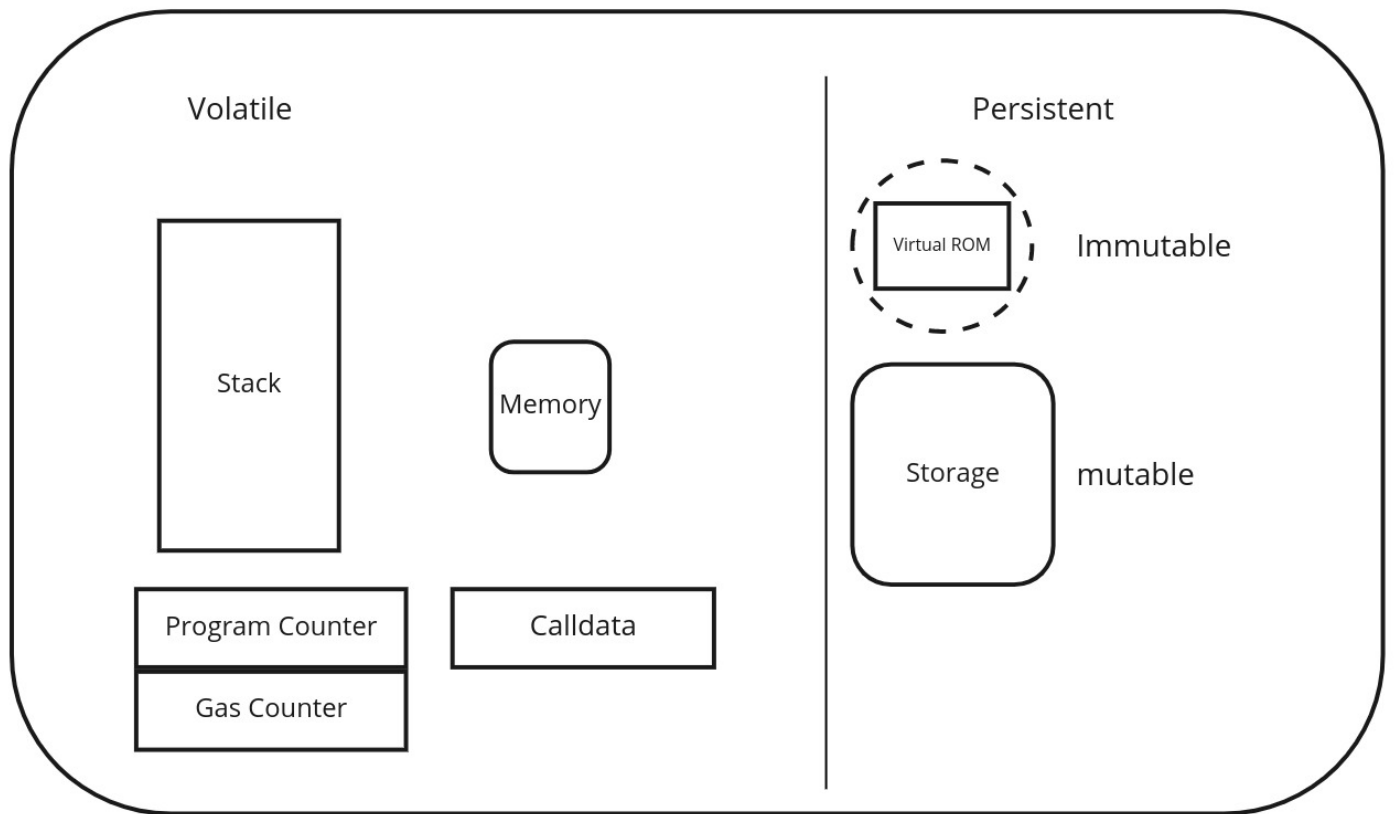
The first method necessitates developers to craft unique "ASIC" circuits tailored for distinct DApps. Every DApp would benefit from reduced overhead due to this bespoke circuit design.

However, this leads to composability issues as the circuit is "static", and it results in a challenging developer experience due to the high-level expertise required in circuit design.

The second design has a specialised EVM, a zkEVM on which the DApp developer can run ordinary smart contracts, taking the burden from the DApp developer (and onto the protocol developer).

The architecture of the zkEVM Layer 2 will roughly follow other zk rollup L2s, having a sequencer and prover (or multiples of each) , L1 <-> L2 messaging capabilities and a contract on the L1.

The EVM Architecture



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.


A zkEVM would seek to emulate all of these components and their interactions and create proofs that the interactions were correct.

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

Overview of data availability solutions

Data Availability Sampling

Data Availability Sampling (DAS) is a way for the network to check that data is available without putting too much strain on any individual node. Each node (including non-staking nodes) downloads some small, randomly selected subset of the total data. Successfully downloading the samples confirms with high confidence that all of the data is available.

Data Availability Committees

Data Availability Committees (DACs) are trusted parties that provide, or attest to, data availability.

The security guarantees that come with committees depends on the specific set up. Ethereum uses randomly sampled subsets of validators to attest to data availability for light nodes, for example.

(Proto) Danksharding

See [Proposal](#)

See [FAQ](#)

EIP-4844 will implement a new transaction type that will hold an additional data field called a *blob*. A blob can be considered an opaque byte string up to ~125 kB in size. These blobs are committed with the KZG commitment scheme and are forward compatible with data availability sampling and can assist in alleviating the network's burden from growing block sizes and unsustainable transaction gas costs.

You can find more information regarding these KZG commitments [here](#).

Transactions that include data blobs enable layer 2 solutions to access and query data more efficiently and affordably than with calldata, thanks to the smaller data size. Blobs will be stored only in the consensus layer (Beacon nodes), and future developments won't require interaction with the execution layer (EVM).

Introducing blobs in transactions will create a new fee market, distinct from the existing gas market for transaction payments.

With the implementation of EIP-4844, layer 2 solutions will significantly benefit as the cost of posting transactions to layer 1 could reduce by 10 to 100 times, leading to lower costs for end users.

Proto-Danksharding, or EIP-4844, introduces a method for rollups to include data in blocks more cost-effectively. The concept aims to overcome the limitations of current rollups, which use `CALLDATA` for posting transactions, incurring high costs due to processing by all Ethereum nodes and permanent on-chain storage.

Proto-Danksharding uses temporary data blobs, not accessible to the EVM and deleted after 1-3 months, reducing costs and allowing savings to be passed to users.

Rollups, under this system, will post their transaction data in these blobs, along with a commitment to the data. Provers can verify these

commitments or challenge suspect data.

To illustrate data availability consensus clients will store these data blobs, attesting to their propagation and visibility in the network.

Since the data is not stored indefinitely, being pruned every 1-3 months, this approach prevents bloating of clients and reduces hardware demands for nodes.

The attestations by consensus clients ensure sufficient time for data verification by provers. Rollups can also store data off-chain.

Furthermore, rollups will post transactions in data blobs along with a data commitment, typically created by fitting a polynomial function to the data. This is a common technique used with for example error correcting codes and a means of proving redundancy.

For example a simple polynomial like $f(x) = 2x-1$ can be used to get specific results.

Provers would then use the same function on the data to confirm its integrity.

Any alteration in the original data leads to different function results, indicating a discrepancy.

Progress to danksharding

Proto-danksharding (aka. [EIP-4844](#)) is a proposal to implement much of the logic and standards such as transaction formats, verification rules that make up a full Danksharding spec, before we start fully implementing any sharding.

In a proto-danksharding implementation, all validators and users still have to directly validate the availability of the full data.

The major change we will see will be the introduction of a new kind of transaction, termed a **blob-carrying transaction**.

This transaction type is similar to standard transactions but includes an additional, large data segment known as a **blob**, typically around 125 kB. These blobs are more cost-effective compared to equivalent calldata

amounts but are not directly accessible during EVM execution; only a commitment to the blob can be seen by the EVM.

In proto-danksharding, data bandwidth is reduced to 1 MB per slot, compared to a potential 16 MB, due to the need for validators and clients to download full blob contents. This still offers significant scalability improvements, as the blob data does not consume the gas resources used by standard Ethereum transactions. The number of blobs per block is limited.

The execution layer (EVM) will only interact with the blob's commitments.

Both EIP-4488 and proto-danksharding aim for a long-term data usage of approximately 1 MB per slot (every 12 seconds), which translates to about 2.5 TB per year, however this is significantly more than Ethereum's current growth rate. Given this, other proposals have been made to limit the state growth.

In the case of EIP-4488, addressing this data growth involves implementing history expiry (as per [EIP-4444](#)), where clients are not required to store history beyond a certain time frame (ranging from 1 month to 1 year).

It would be inefficient if we had to verify every bit of data, instead we use data availability sampling to allow validators to efficiently verify blob data. This process allows confirmation of the availability and correct commitment of blob data by sampling just a few points and creating a proof, without needing to check the entire blob. This method would quickly find and reject any blobs with missing data.