Week 2

Lesson 5 - Understanding and Analysing Layer 2

Lesson 6 - Agnostic Layer 2 Transaction Lifecycle

Lesson 7 - Optimistic Rollups v ZK Rollups

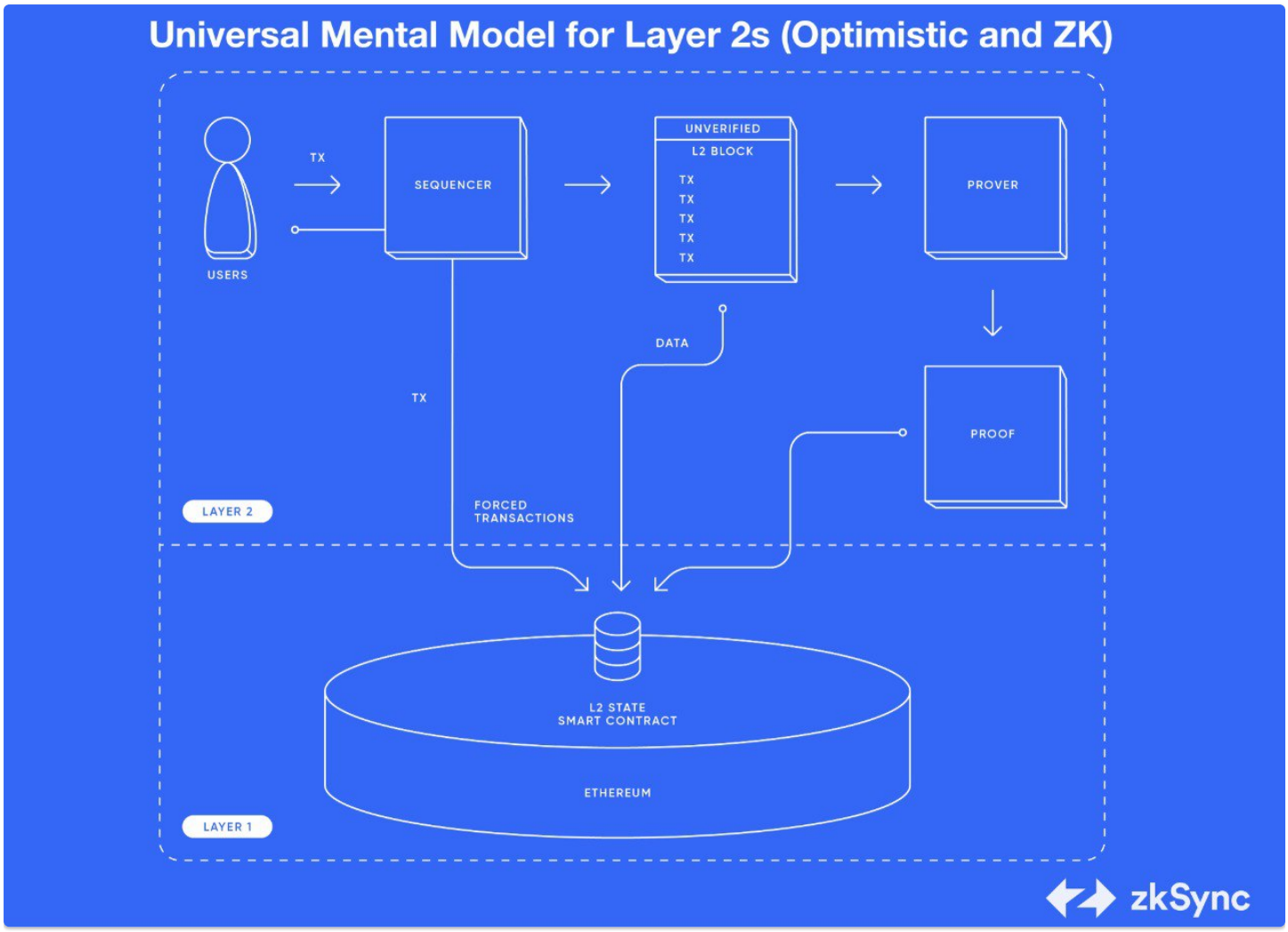
Lesson 8 -What's next in Layer 2 part 1 : Decentralised Sequencers

Today's topics

- L2 components and transaction lifecycle
- L2 rollup process
- L2 examples

Layer 2 Components

Layer 2 chains will differ in the details of the components they have, but generally speaking they follow a similar design.



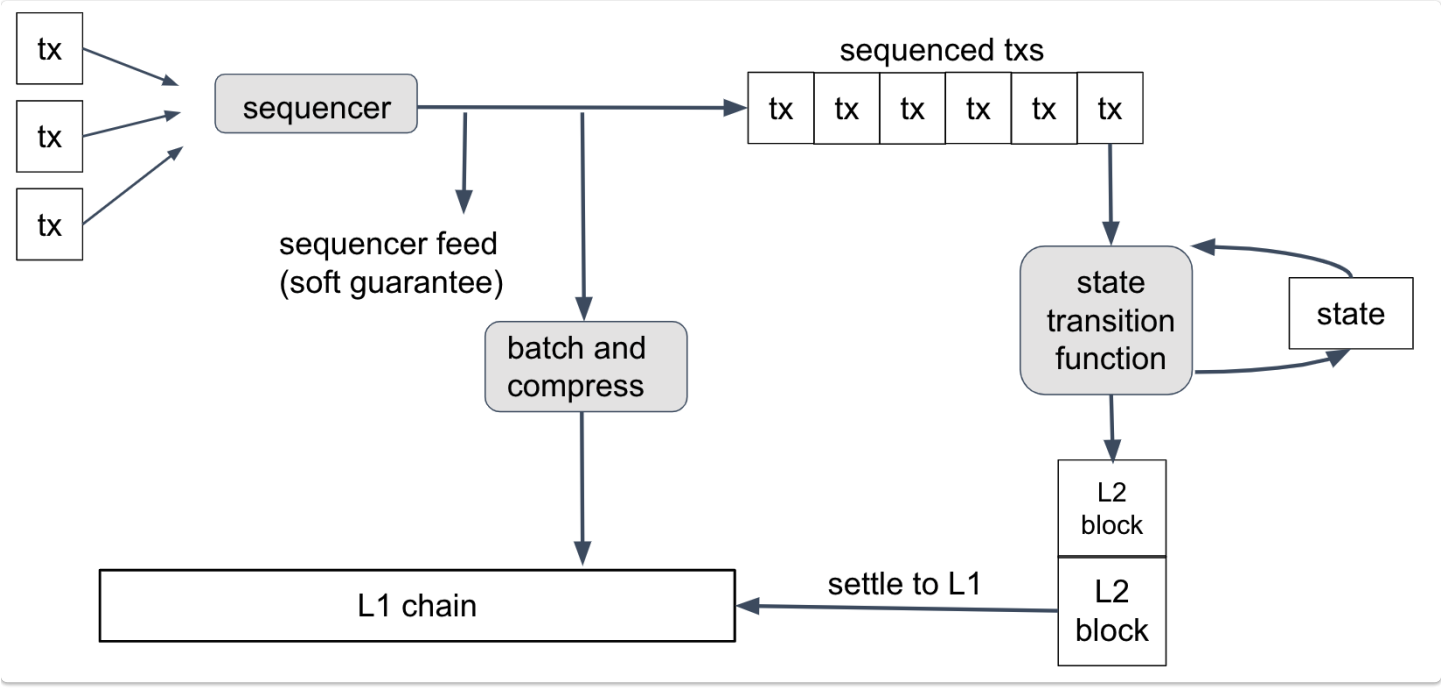
A sequencer or operator has a coordination role, they accept transactions, pass these to a VM for execution, batch and compress the transactions and submit them (plus a proof) to the L1.

The final component is a contract on the L1 that will receive the transaction batch, and for validity rollups verify the proof.

There may also be some splitting out of the concerns, so we may have components to handle aggregation of proofs.

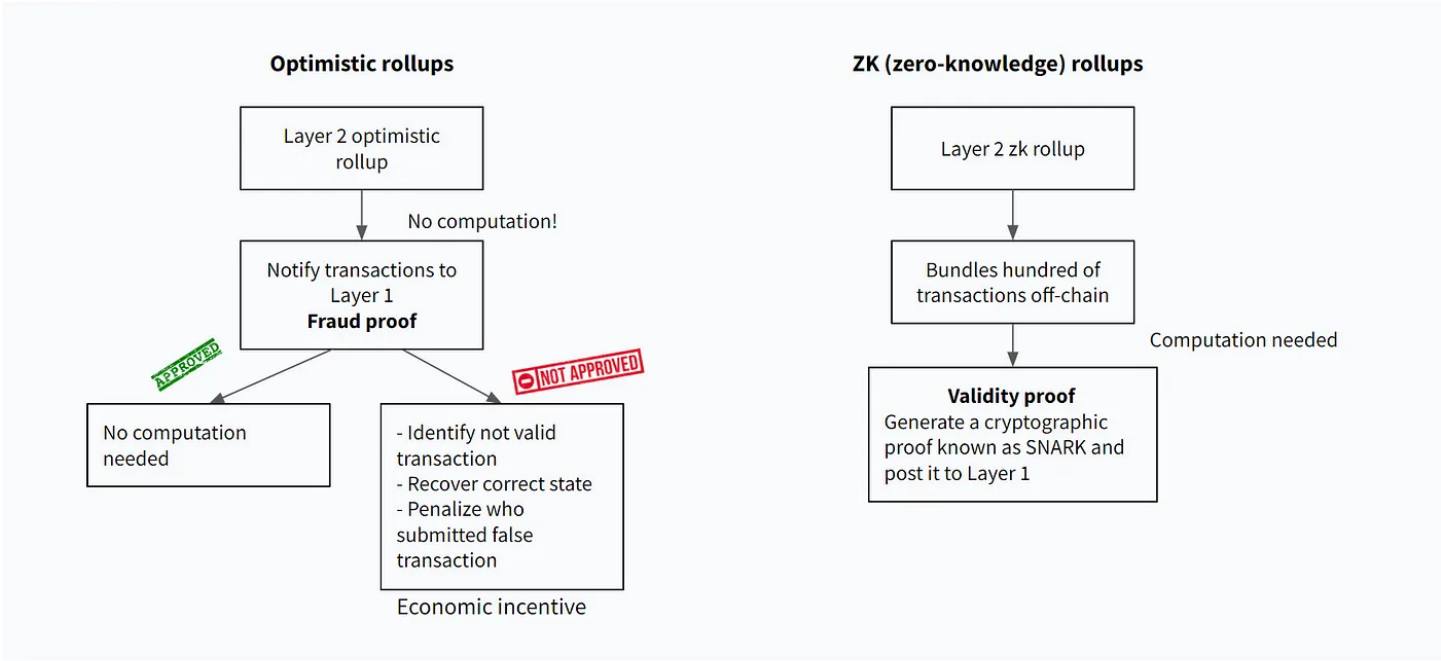
Messaging between the L1 and L2 is also possible outside the route outlined above. Contracts on the L1 and L2 are available to process specialised messages, to allow for example deposits of funds into the L2.

The design of Arbitrum has similar components



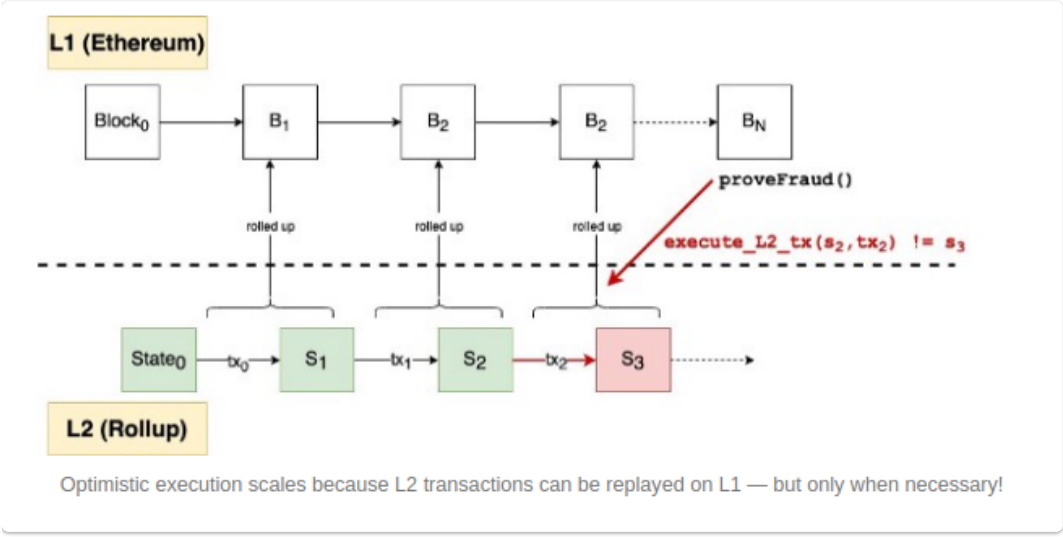
Layer 2 transaction lifescycle

Optimistic and ZK-rollups basic principles



 Source: Dealroom.co. Inspired by Bankless

dealroom.co

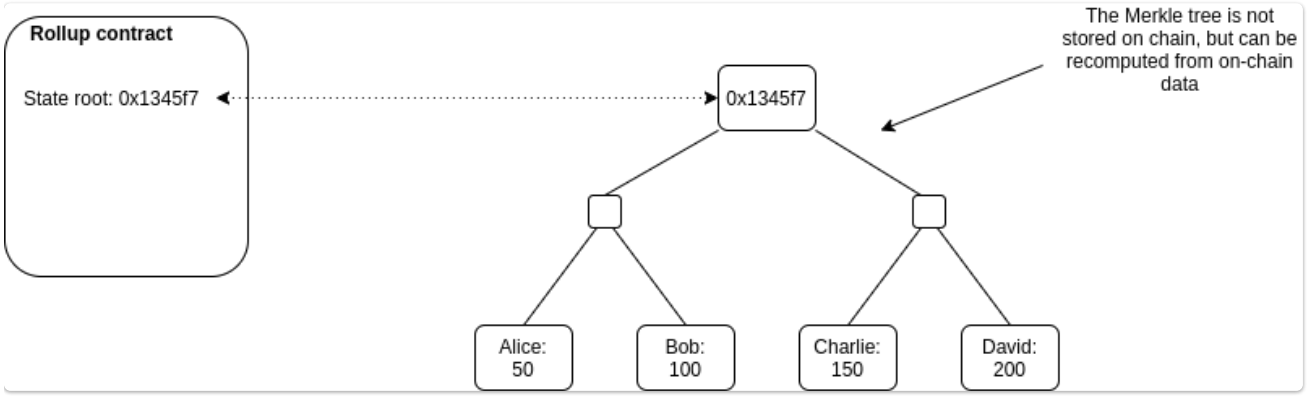


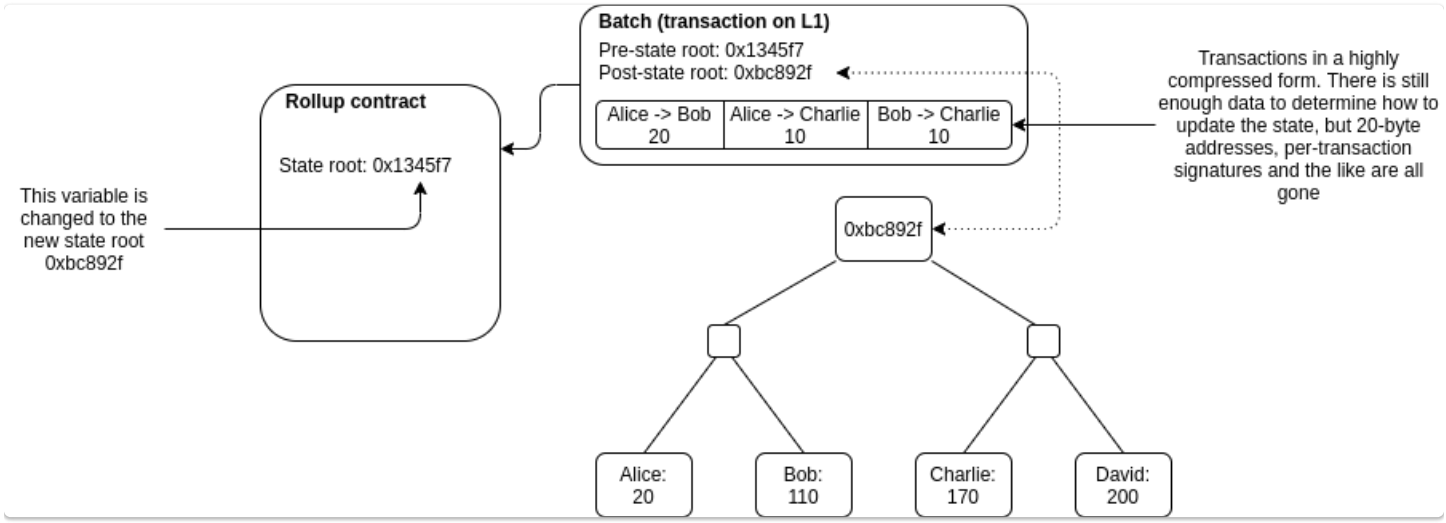
Optimistic execution scales because L2 transactions can be replayed on L1 — but only when necessary!

Rollup process

See [article](#) from Vitalik

There is a smart contract on-chain which maintains a **state root**: the Merkle root of the state of the rollup (meaning, the account balances, contract code, etc, that are "inside" the rollup).





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.

To support depositing and withdrawing, we add the ability to have transactions whose input or output is "outside" the rollup state. If a batch has inputs from the outside, the transaction submitting the batch needs to also transfer these assets to the rollup contract. If a batch has outputs to the outside, then upon processing the batch the smart contract initiates those withdrawals.

The missing part we have here, is the question of proving that the state transition is correct. This is where the 2 types of rollups diverge, using validity rollups (proof of what is submitted), or fraud proofs (no proof is submitted by default, but a proof of fraud can be submitted after the fact)

zkSync transactions

See Docs

In addition to the usual transactions from a DApp that will be submitted, we have some L2 specific processes.

Depositing / Withdrawing funds in zkSync

These are special operations, concerned with depositing funds into the L2, or exiting from the L2.

Deposit

The process is
Users must call the `deposit` method on the L1 bridge contract, which triggers the following actions:

- The user's L1 tokens will be sent to the L1 bridge and become locked there.
- The L1 bridge initiates a transaction to the L2 bridge using L1 -> L2 communication.
- Within the L2 transaction, tokens will be minted and sent to the specified address on L2.
    - If the token does not exist on zkSync yet, a new contract is deployed for it. Given the L2 token address is deterministic (based on the original L1 address, name and symbol), it doesn't matter who is the first person bridging it, the new L2 address will be the same.
- For every executed L1 -> L2 transaction, there will be an L2 -> L1 log message confirming its execution.
- Lastly, the `finalizeDeposit` method is called and it finalises the deposit and mints funds on L2.

Withdrawal

Users must call the `withdraw` method on the L2 bridge contract, which will trigger the following actions:

- L2 tokens will be burned.
- An L2 -> L1 message with the information about the withdrawal will be sent.
- After that, the withdrawal action will be available to be finalized by anyone in the L1 bridge (by proving the inclusion of the L2 -> L1 message, which is done when calling the `finalizeWithdrawal` method on the L1 bridge contract).
- After the method is called, the funds are unlocked from the L1 bridge and sent to the withdrawal recipient.

Events

Events can be emitted in the same way as on the L1 EVM, they are then available to be searched by external clients.

Optimistic Rollup Process

1. An aggregator collects multiple transactions from users off-chain, just like in ZK-rollups.
2. The aggregator executes the transactions in the rollup and generates a new Merkle tree representing the updated state.
3. The aggregator submits the Merkle root of the updated state to the base layer (Layer 1) of the blockchain without a cryptographic proof.
4. The base layer stores the Merkle root and starts a challenge period, typically lasting several hours.
5. During the challenge period, users or validators can submit fraud proofs if they detect any invalid transactions in the rollup.
6. If no fraud proofs are submitted within the challenge period, the rollup is considered valid, and the on-chain state is updated accordingly.

ZKRollup Process

1. An aggregator collects multiple transactions from users off-chain.
2. The aggregator bundles these transactions into a “rollup” and creates a new Merkle tree that represents the updated state.
3. Using either ZK-SNARKs or ZK-STARKs, the aggregator generates a zero-knowledge proof that confirms the rollup’s validity.
4. The aggregator then submits the proof and the Merkle root of the updated state to the base layer (Layer 1) of the blockchain.
5. Layer 1 verifies the proof, updates the on-chain state with the new Merkle root, and emits an event to notify users.

Finality options

Different rollups handle finality differently, but this from Polygon is fairly representative.

- Once the transaction has been received by the sequencer it is a *trusted* state, at this stage, there is no data on L1.
- Once the transaction data is available on L1, the transaction is in *virtual* state, the order of the transactions within the rollup block is fixed.
- Once the proof has been verified by the L1 verifier contract, the transaction is in *Verified* state. It is now safe to for example withdraw assets.

If the user trusts the sequencer, finality could be said to have been achieved when the transaction is in *trusted* state.

The process for zksync is shown here

- Fill a batch with transactions (usually takes a few minutes).
- Commit the batch to Ethereum.
- Generate a proof for the whole batch (usually takes around an hour).
- Submit the proof for verification by the Ethereum smart contract.
- Finalise the batch on Ethereum (delayed by ~21 hours as a security measure during the alpha phase).

When this process is complete, around ~24 hours in total, the batch is as final as any Ethereum transaction included in the same Ethereum block.

Comparing L1 and L2 Architecture

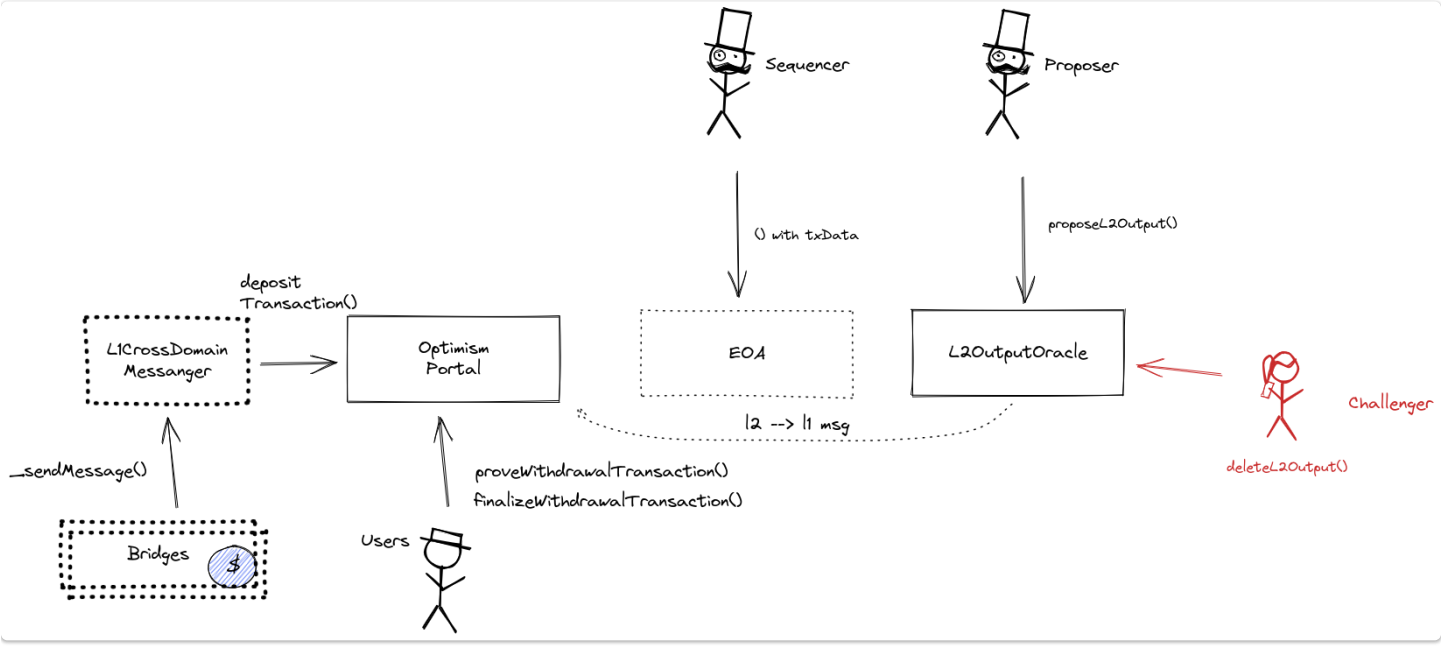
Although L1 and L2 are both blockchains, and on the surface may seem quite similar, particularly when a zkEVM has compatibility (to some degree) with the L1 EVM, the nature of the connection between the layers and the security requirements lead to some significant differences.

- L2 nodes can use the decision of the L1 validator contract.
- The L1 validator contract has a significant role in the rollup process, yet it is not contained within the L2, nor could its role be supplied by the other L2 components.

In addition the 'rollup centric' road map for Ethereum now has more focus on and understanding of the need for data availability. We see major potential changes such as Danksharding arising out of the needs and experience of building L2s.

Contracts used in Optimism

Taken from [L2Beat]



Contracts used in Polygon zkEVM

The source code for PolygonZKEVM is available