L1StateOracle

Problem

Currently, there is no way for L2s to access L1 state in a *trustless, cheap and easy way*. One option is to use arbitrary messaging bridges to send over the L1 state, but in this case you need to rely on the honesty of the messenger. Another option is to set up a specific purpose bridge (think ERC20 or ERC721 token bridge) so that you don't need to trust the messenger anymore. But this is not generalizable and costly since you need to create a bridge for every single purpose.

So our question was, is there a better way to send over L1 state to L2s?

Our approach

Instead of creating an entirely new system from scratch, we took advantage of two existing systems to create a solution to this problem. We were inspired by the Hashi team (ethresearch post (https://ethresear.ch/t/hashi-a-principled-approach-to-bridges/14725/1), presentation (https://docs.google.com/presentation/d/1yMdO179XFJeerylqsJg8L4RewH8jaA_p97iCO-vl9mY/edit#slide=id.g21cefba53b5_0_148)) to combine two existing systems to create a solution.

One is Hashi (https://github.com/gnosis/hashi), which is a system that provides additive security for bridge systems. Essentially, it improves security by allowing L2 protocols to not rely on a single bridge system. Under the hood, it is connected to multiple bridges deployed on L2 and provide aggregate L1 block hash data to L2 protocols. As a result, L2 protocols that rely on a bridge system can avoid being hacked when a single bridge is compromised.

Another is Axiom (https://www.axiom.xyz/), which enables accessing any historic state on-chain via smart contracts. Storing historic states requires a lot of storage, so it's normally unaffordable on-chain, but Axiom leverages ZK proofs to make this cheap. One thing to point out is that Axiom is currently intended to be used only on L1, but the system is modular so we were able to think about porting a part of it on L2.

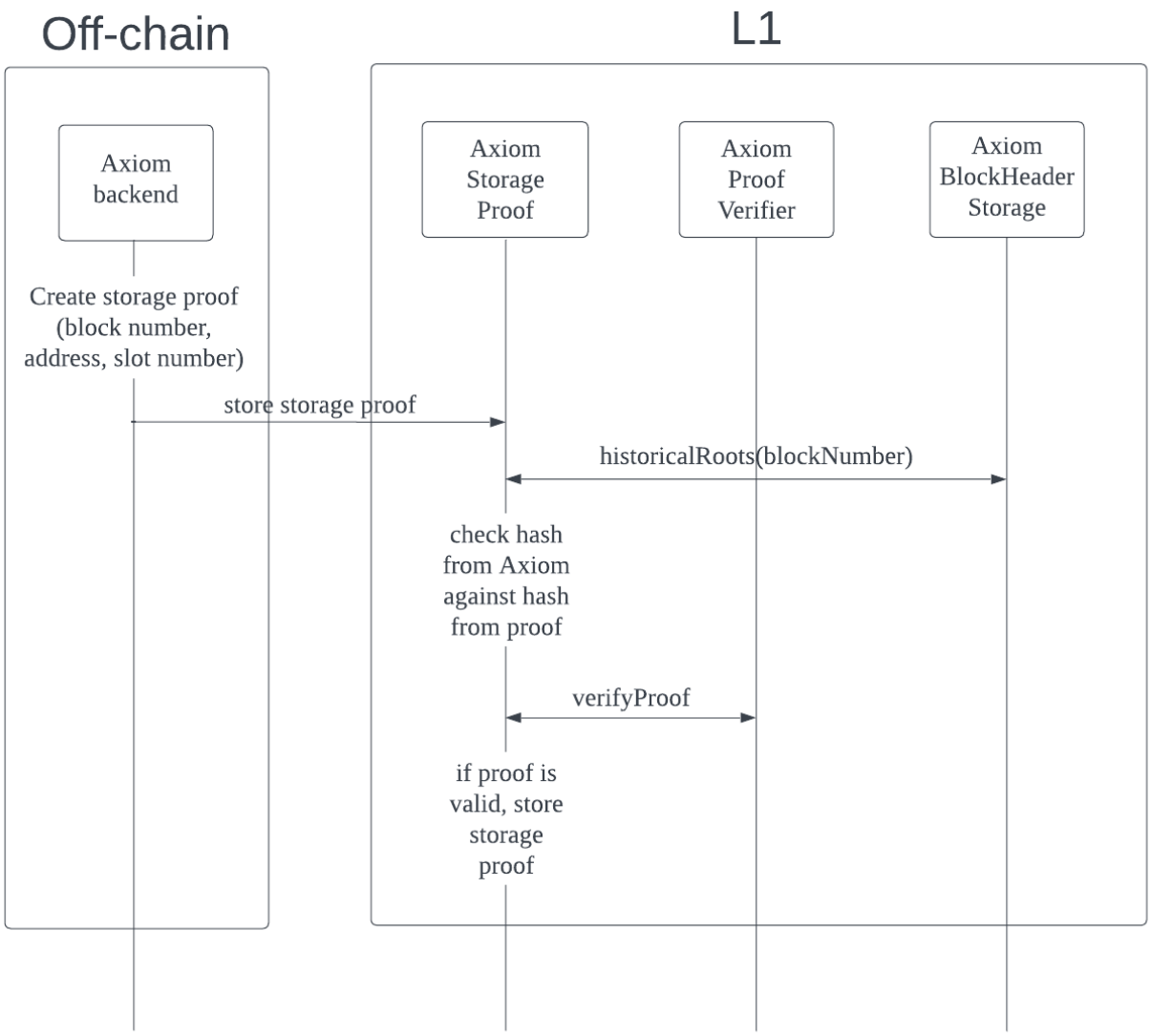
Solution

> Axiom ✍️ + Hashi 橋 => L1StateOracle (Time Travel 🚀 L1 state on L2)

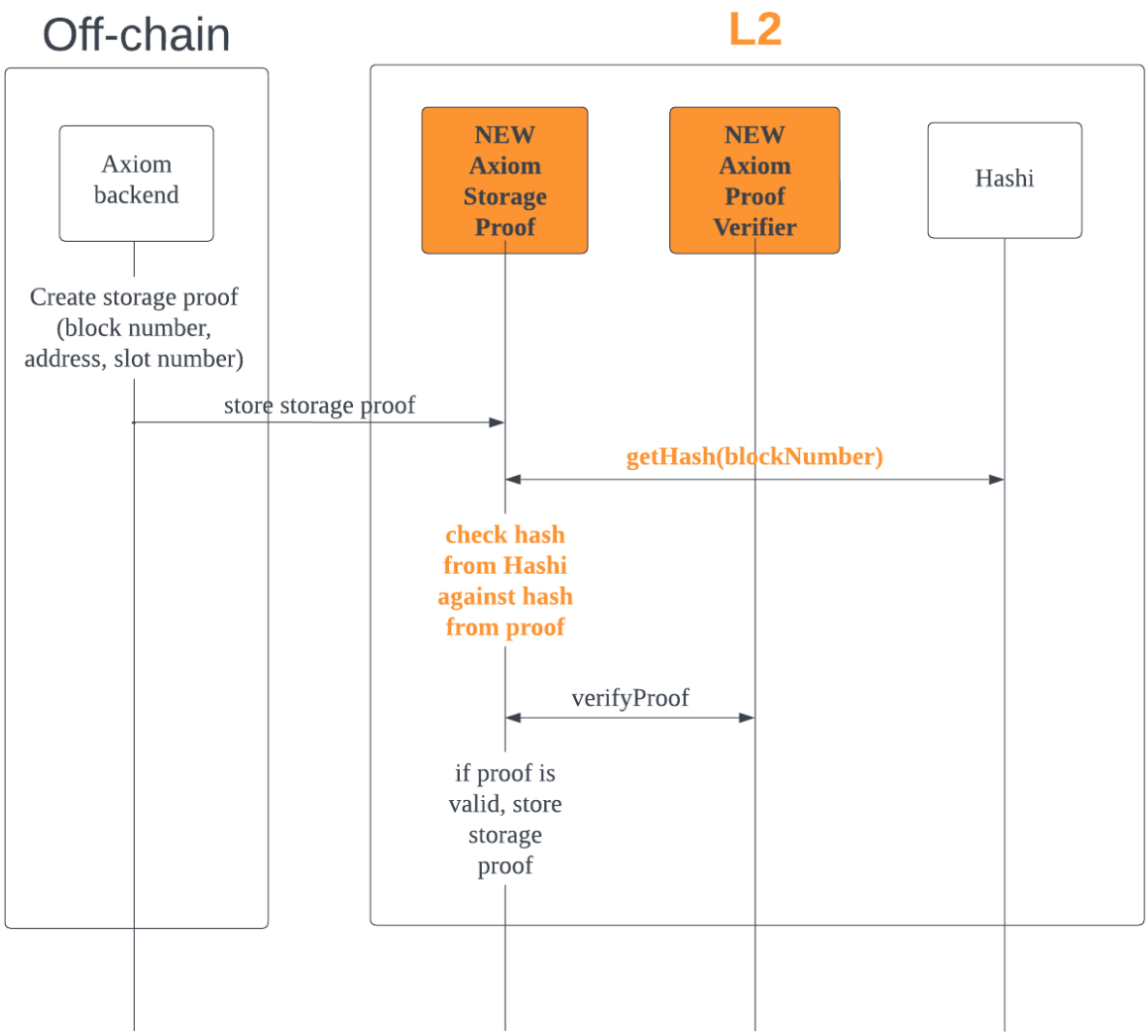
Our approach is to take the proof module of Axiom and to integrate it with Hashi.

As you can see in the flow charts below, we created a new `AxiomStorageProof` contract and deployed it on L2.
Once a user creates a storage proof using Axiom's backend (https://demo.axiom.xyz/custom), it can send the proof to the L2 contract, which will verify the block hash used in the proof against Hashi's `getHash` function.
When the zkp itself is also verified via the `AxiomProofVerifier`, we can safely store the storage proof on-chain, and *voilà*! Any L2 protocol can confidently use the attested storage data without worrying about a single bridge being compromised.

Axiom Architecture



L1StateOracle Architecture



L1StateOracle

Proof attestations of historic L1 states on L2 using Hashi

Agenda

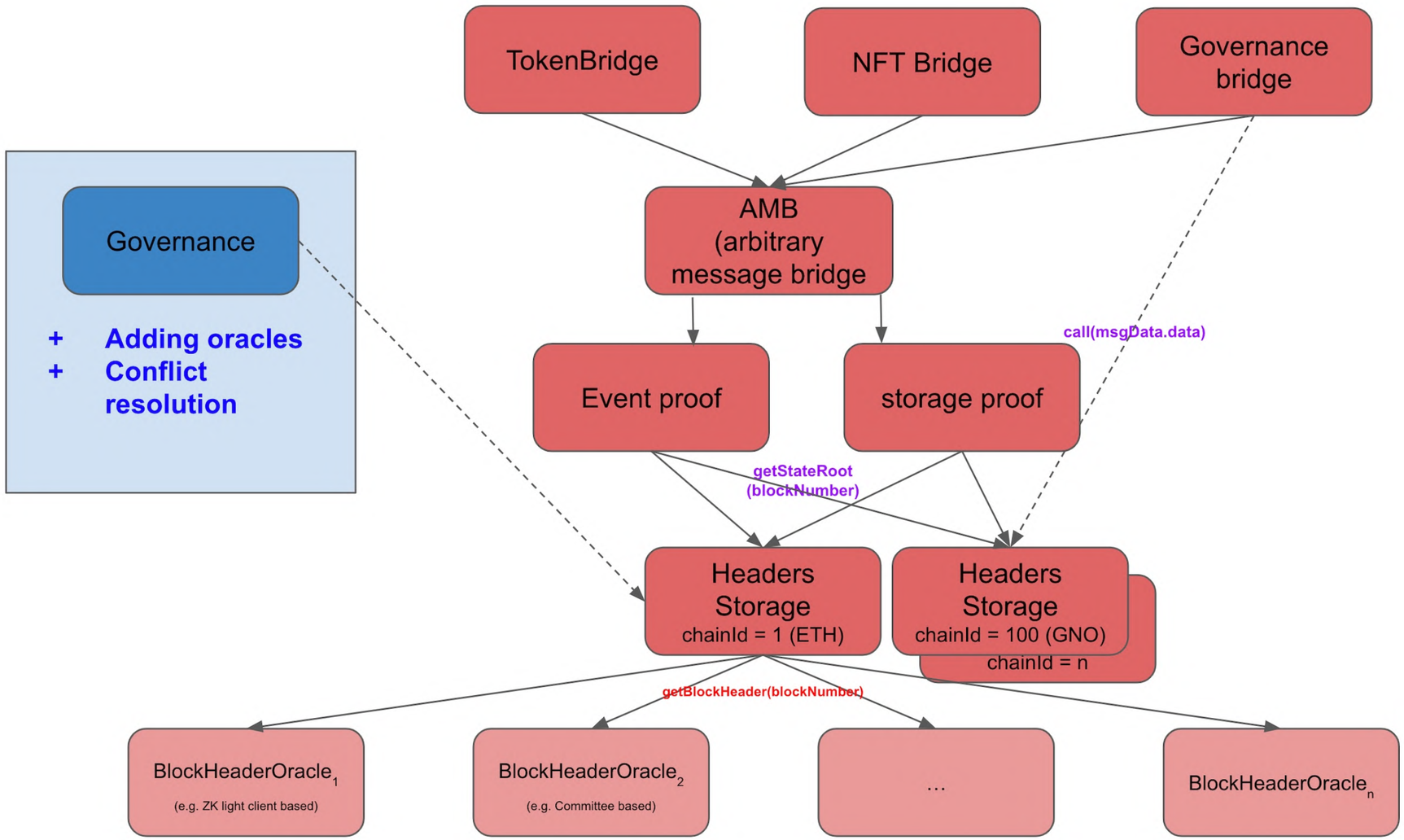
- Problem
- Solution
- Technical Introduction
- Architecture
- Deeper Dive
- Use cases
- Demo
- Future steps

Problem

- **L1 data accessibility on L2**: How can L2 access L1 data on chain? Bridging every value on L2 is costly.
- **Can't trust L1 data:** How can you trust L1 data on L2?
- **Historical data:** What if I want the L1 data at block number X?

Block header is not enough!

Inspiration

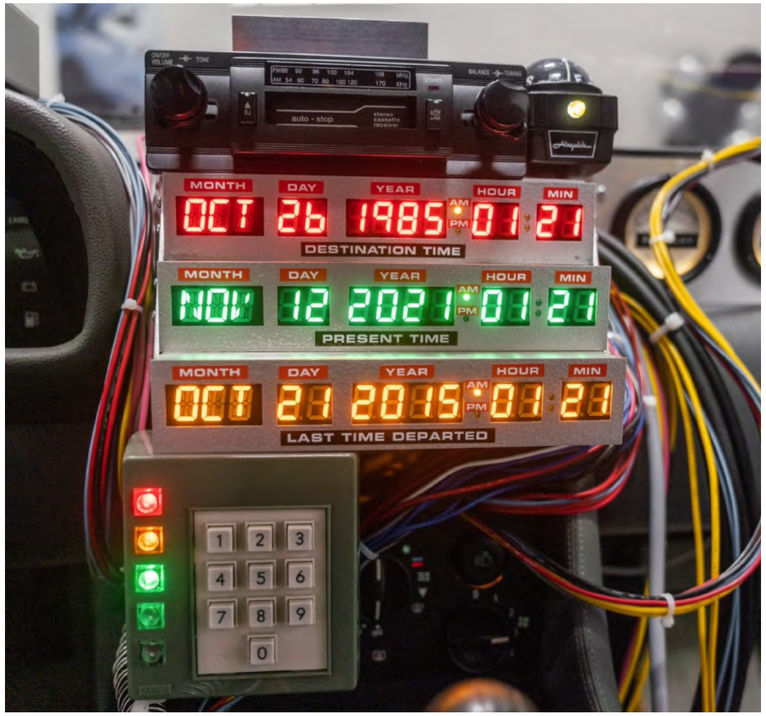


https://ethresear.ch/t/hashi-a-principled-approach-to-bridges/14725

Solution

Axiom ✍️ + Hashi 橋 => L1StateOracle (Time Travel 🚀 L1 state on L2)

- **Fetch historical L1 storage value** off chain at a certain block
- **Create a ZK proof** that includes **L1 storage value**
- **Submit** the block header **hash with a proof to L2 verifier**
- **Check the equality of block header** reported from Hashi adapter
- **Decode the L1 storage value** from the proof on L2



Technical Introduction

Hashi

- Provides additive security for getting L1 block header hashes

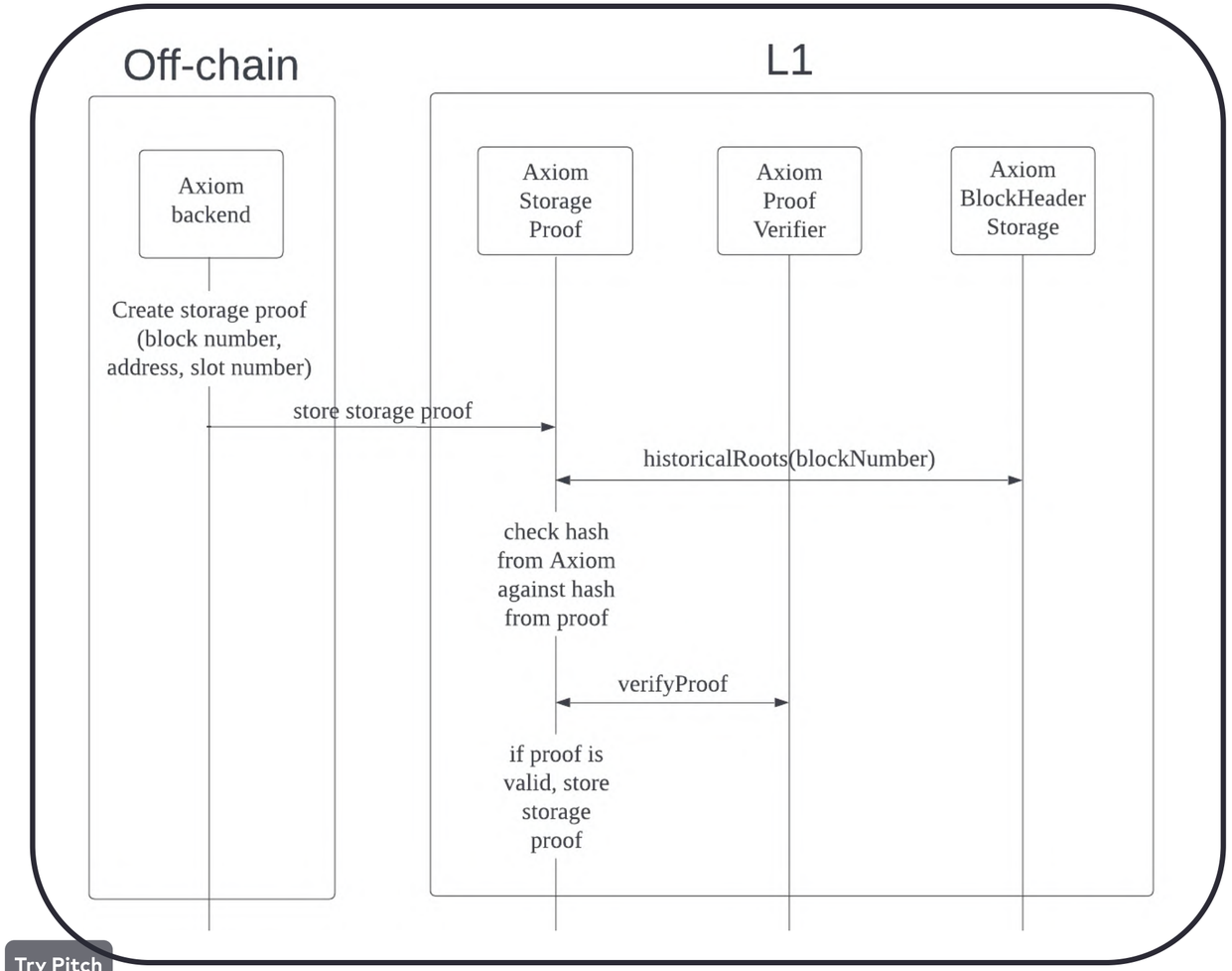
Axiom

- Provides proof of L1 historical state using zkp
- Can prove any L1 historical state on-chain by providing zkp
- On-chain proof verification is done using an on-chain light client (i.e. block header hashes)

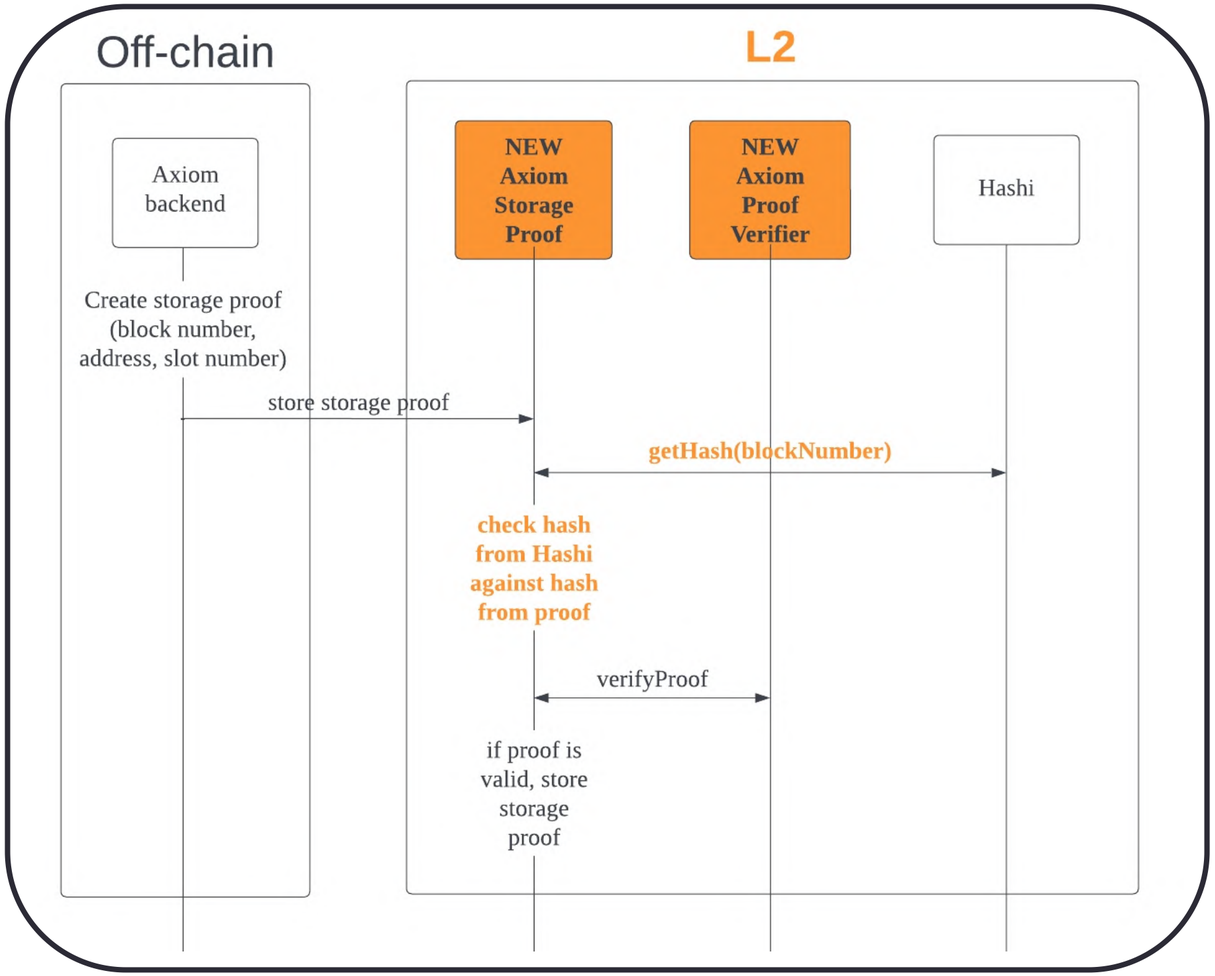
=> L1StateOracle

Architecture

Axiom Architecture



L1StateOracle Architecture



Deeper Dive

Axiom Storage Proof contract

- verifies the proof with an on-chain verifier

```
(bool success,) = verifierAddress.call(proof);
if (!success) {
    revert("Proof verification failed");
}
```

- parses the data from Axiom to get up to 10 proofs of storage slots
- saves attestations based on hash of **(blockNumber, account, slot, slotValue)**

```
for (uint16 i = 0; i < SLOT_NUMBER; i++) {
    uint256 slot = (uint256(bytes32(proof[384 + 128 + 128 * i:384 + 160 + 128 * i])) << 128)
        | uint128(bytes16(proof[384 + 176 + 128 * i:384 + 192 + 128 * i]));
    uint256 slotValue = (uint256(bytes32(proof[384 + 192 + 128 * i:384 + 224 + 128 * i])) << 128)
        | uint128(bytes16(proof[384 + 240 + 128 * i:384 + 256 + 128 * i]));
    (bytes32 hashedVal) = keccak256(abi.encodePacked(blockData.blockNumber, account, slot, slotValue));
    slotAttestations[hashedVal] = true;
    emit SlotAttestationEvent(blockData.blockNumber, account, slot, slotValue);
}
```

Use Cases

- **Average Price of Token in last X blocks**
  - L2 insurance that wants to settle based on DEX prices on L1
- **Change of NFT ownership over X blocks**
  - L2 NFT marketplace that wants to trustlessly distribute tokens based on user's activity data on L1

Demo

<Add link to PR>

- Since Axiom is not deployed on testnet, we tested locally on a mainnet fork
- We replicated the flow of Axiom but with our custom AxiomV02StoragePf contract, which validates the block hash of a given block number using Hashi, not Axiom's light client contract

```
End-to-end tests
  Execution layer
    ✔ Attest slots for the claimed block head with the block hash agreed on by N adapters (5234ms)
    ✔ Reverts if the claimed block header is different from the block hash agreed on by N adapters (2075ms)


2 passing (7s)
```

Future steps

- Improve test setup by not forking mainnet (deploy verifier Yul code locally)
- Integrate other Axiom proof contracts (account age, uniswap v2 twap, etc)