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

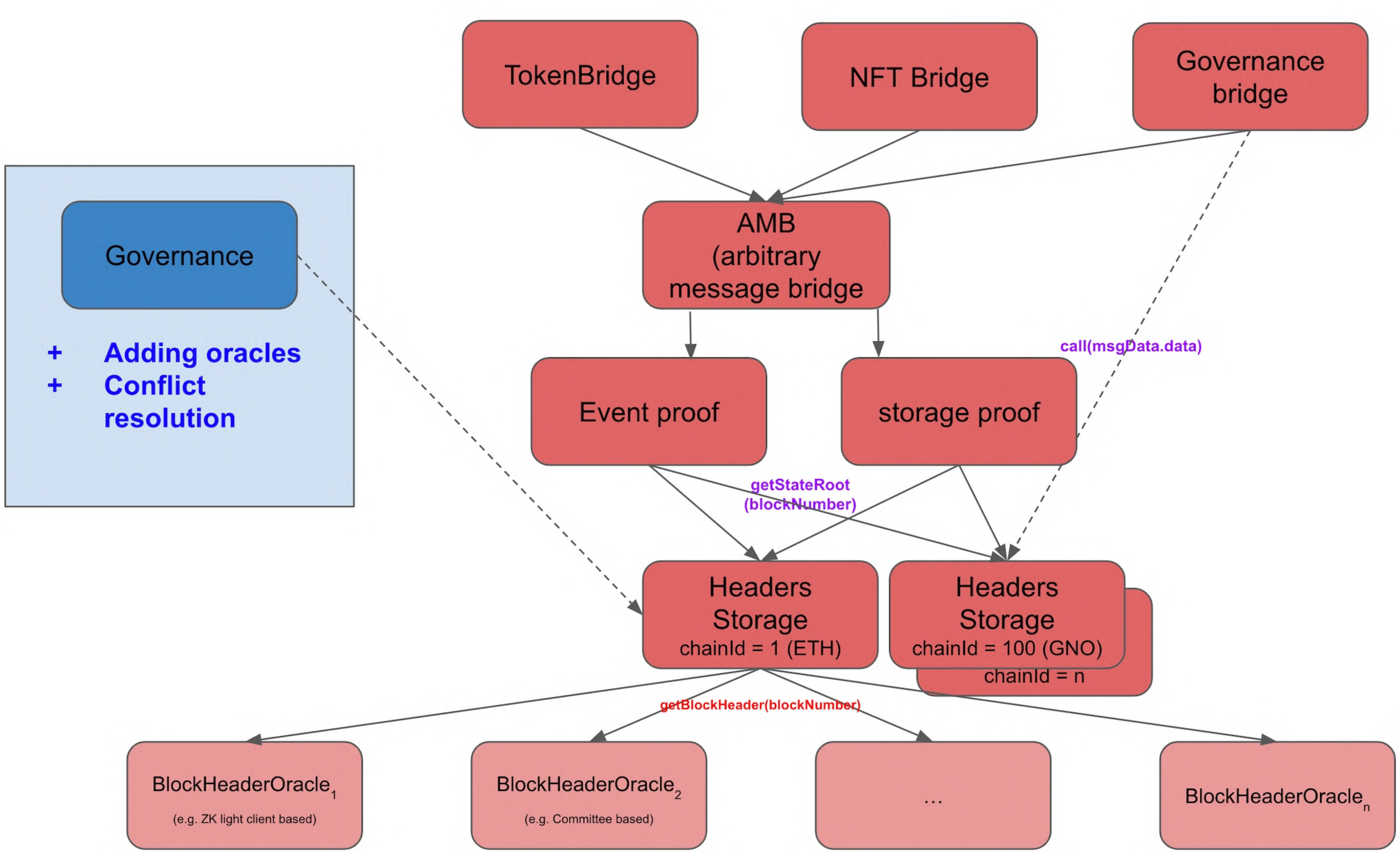
- **L1 data accessibility on L2**: How can L2 access L1 data on chain? Bridging every value on L2 is costly.

- **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

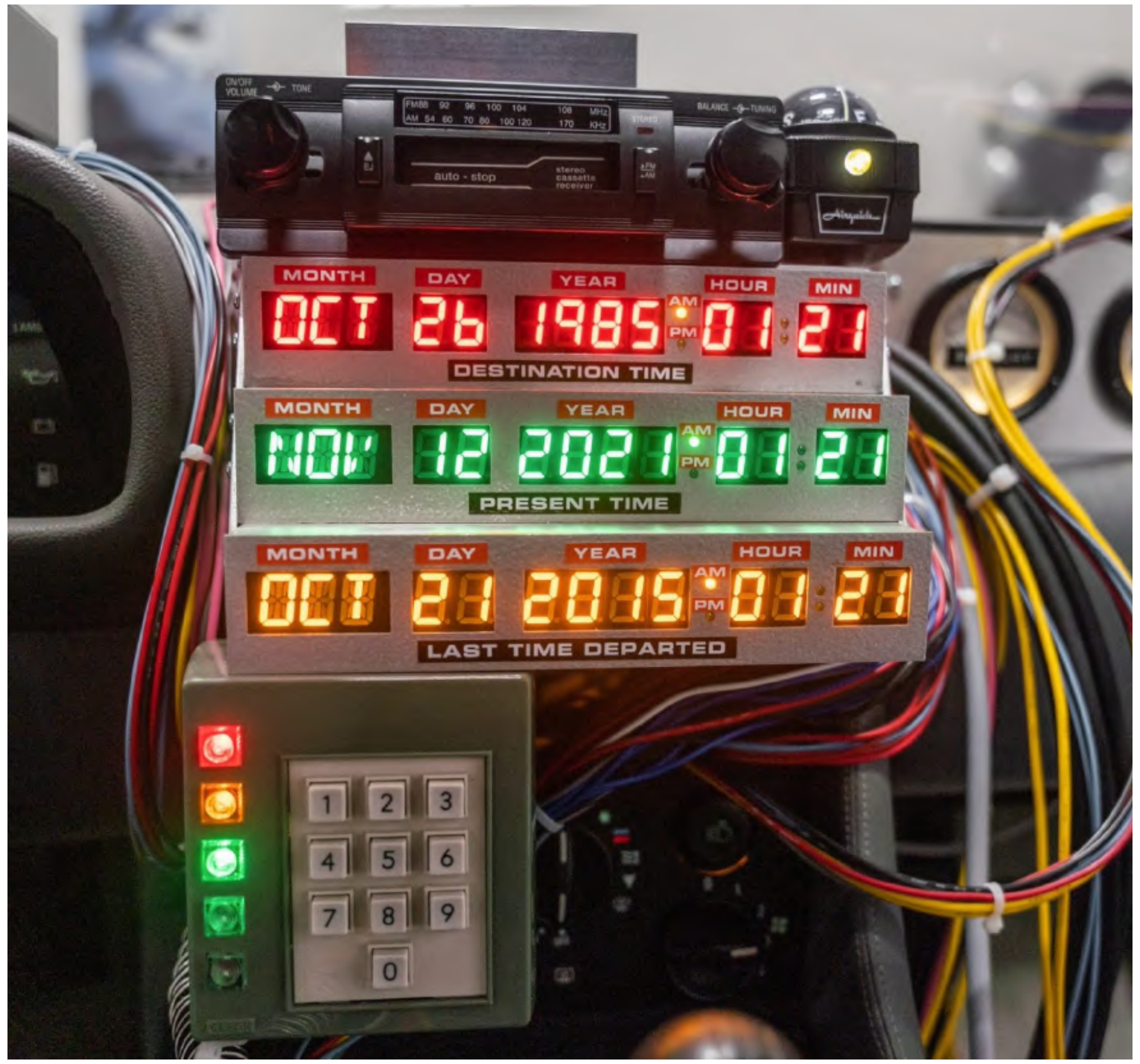


https://ethresear.ch/t/hashi-a-principled-approach-to-bridges/14725

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



Try Pitch

Technical Introduction

Hashi

- Provides additive security for getting L1 block header hashes

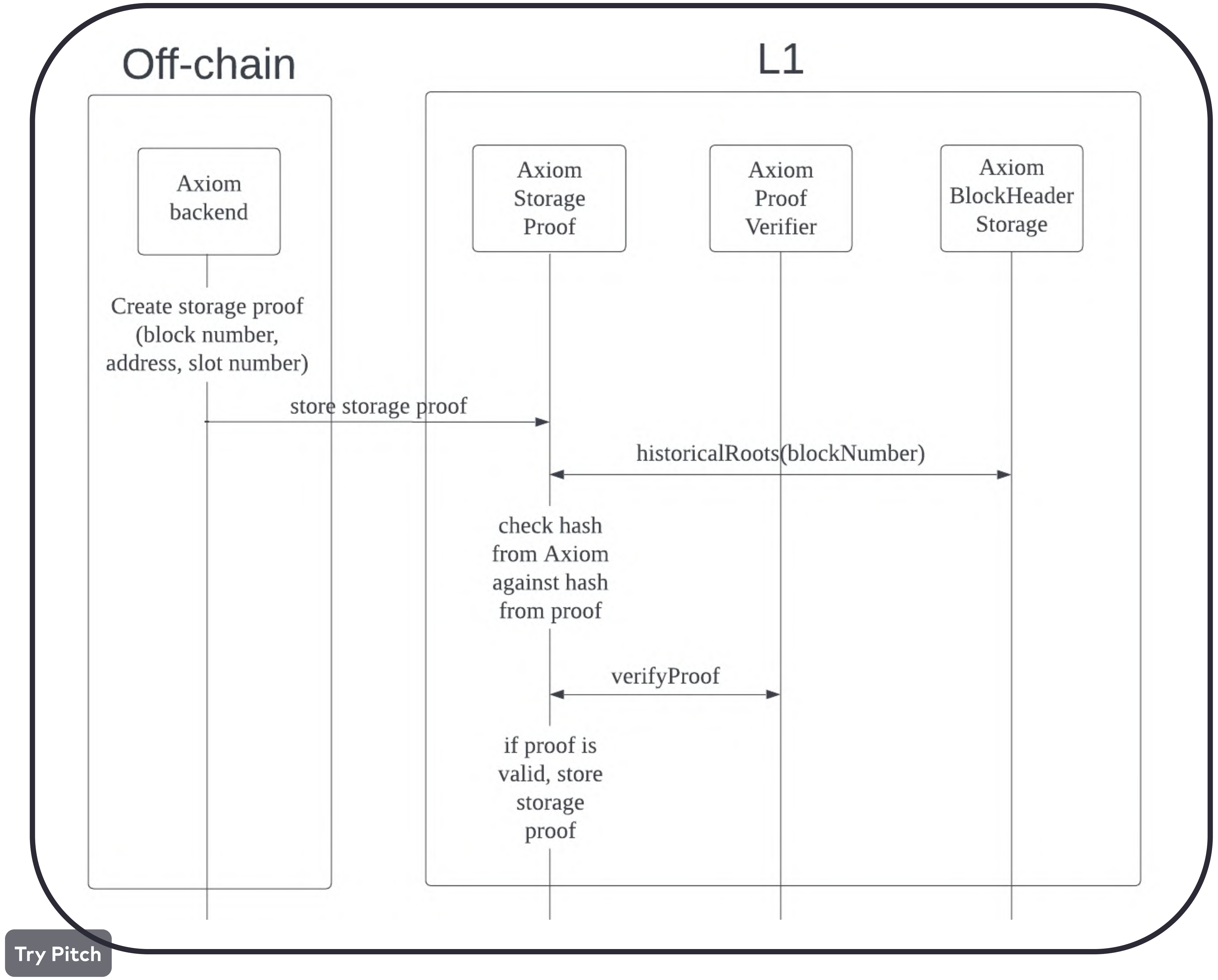
Axiom

- 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)

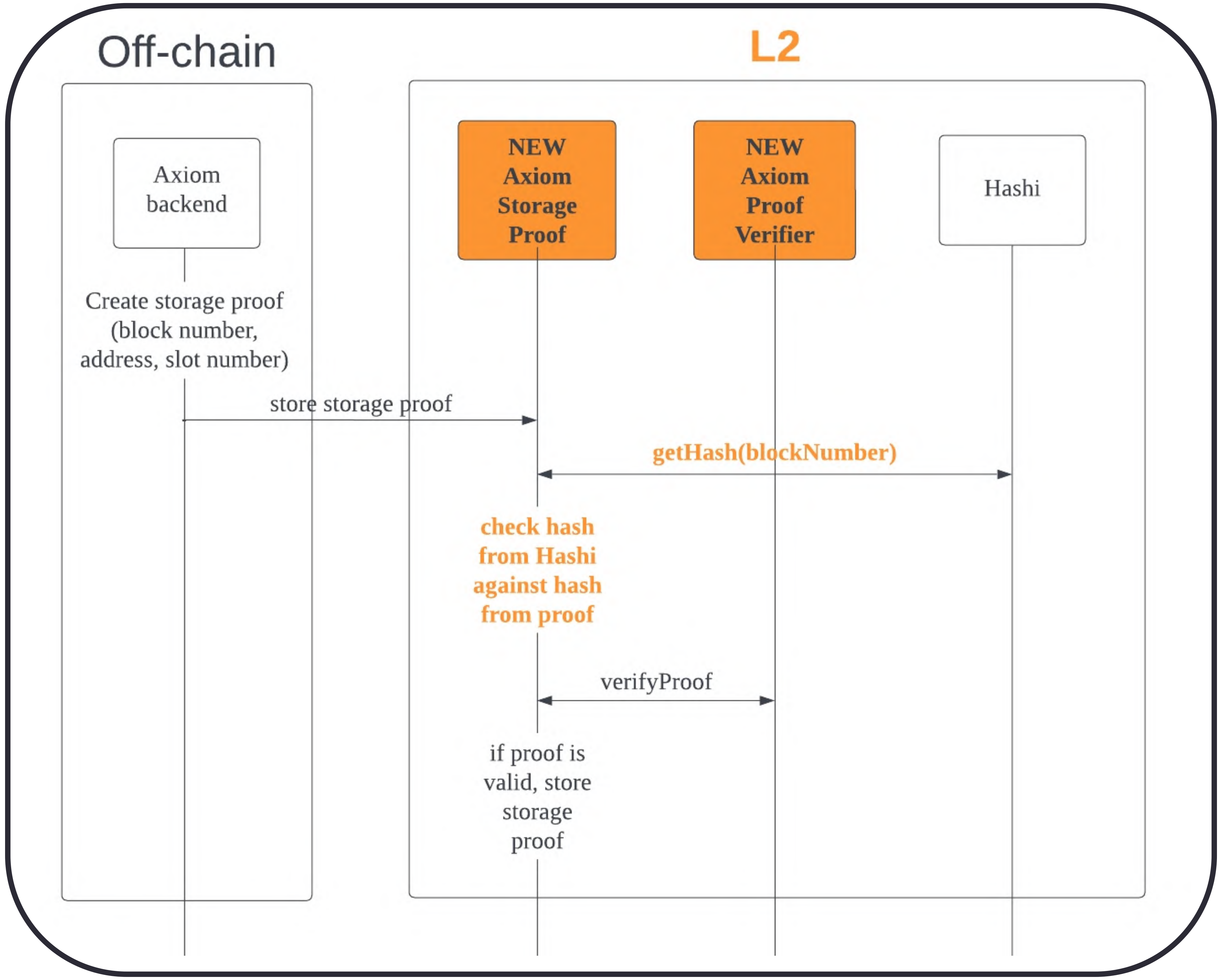
=> L1StateOracle

Architecture

Axiom Architecture


Off-chain
L1
Axiom backend
Axiom Storage Proof
Axiom Proof Verifier
Axiom BlockHeader Storage
Create storage proof (block number, address, slot number)
store storage proof
historicalRoots(blockNumber)
check hash from Axiom against hash from proof
verifyProof
if proof is valid, store storage proof

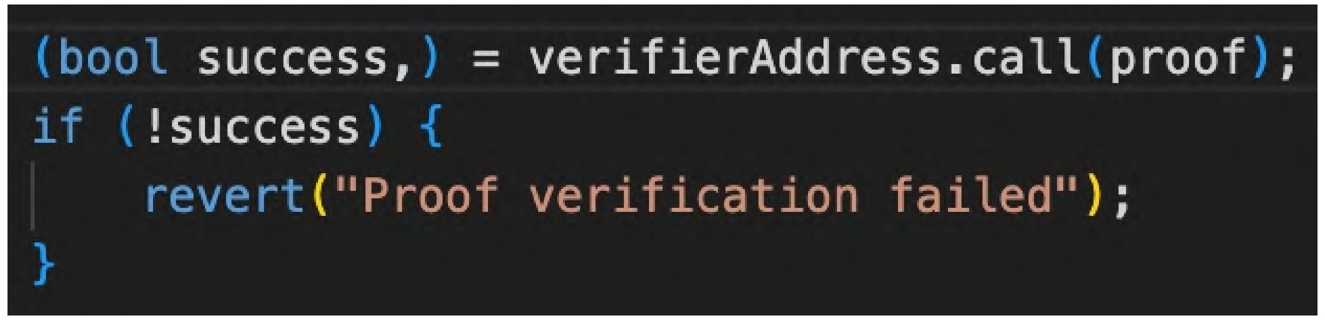
L1StateOracle Architecture


Off-chain
L2
Axiom backend
NEW Axiom Storage Proof
NEW Axiom Proof Verifier
Hashi
Create storage proof (block number, address, slot number)
store storage proof
getHash(blockNumber)
check hash from Hashi against hash from proof
verifyProof
if proof is valid, store storage proof

Deeper Dive

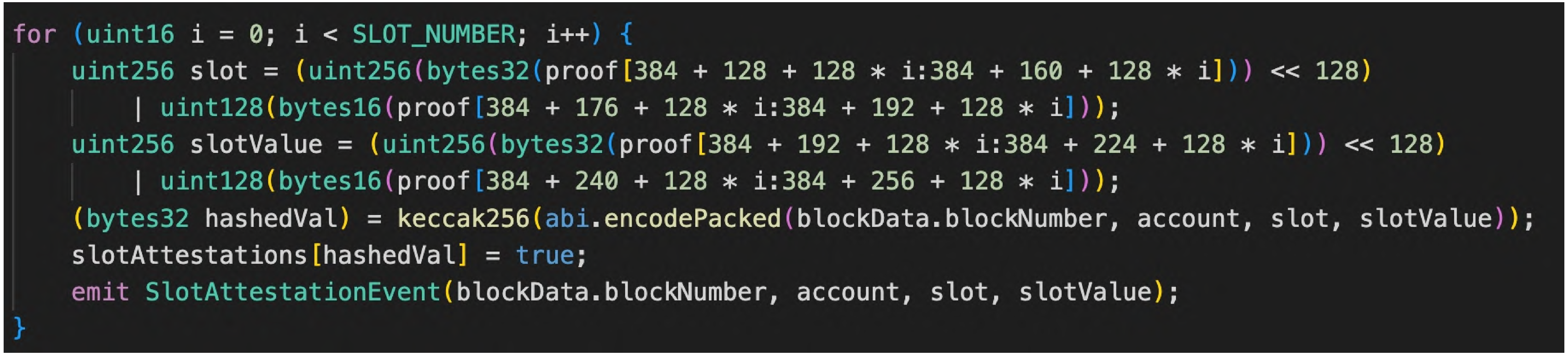
Axiom Storage Proof contract

- 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);
}
```

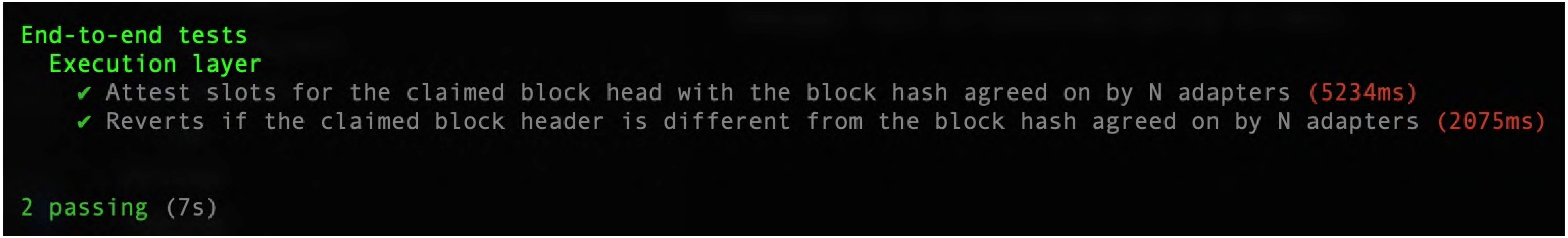
Use Cases

- **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)