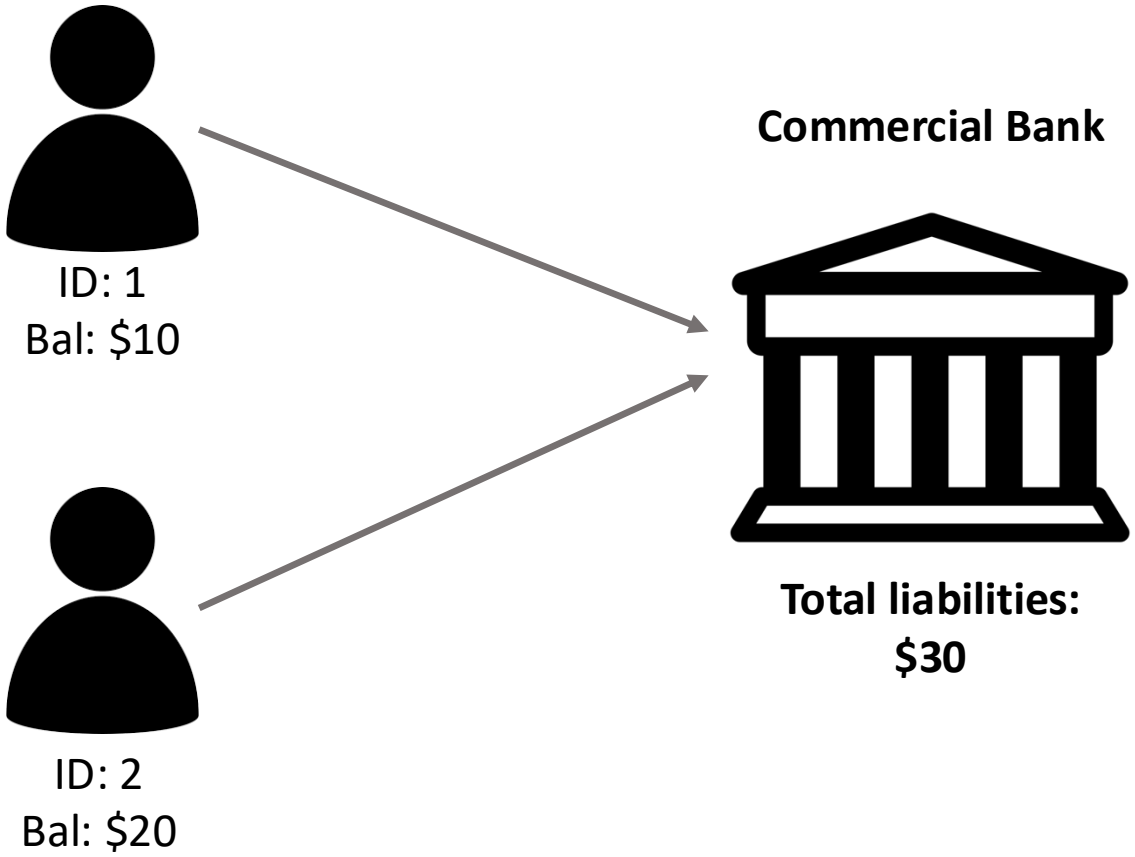
Short Privacy-Preserving Proofs of Liabilities

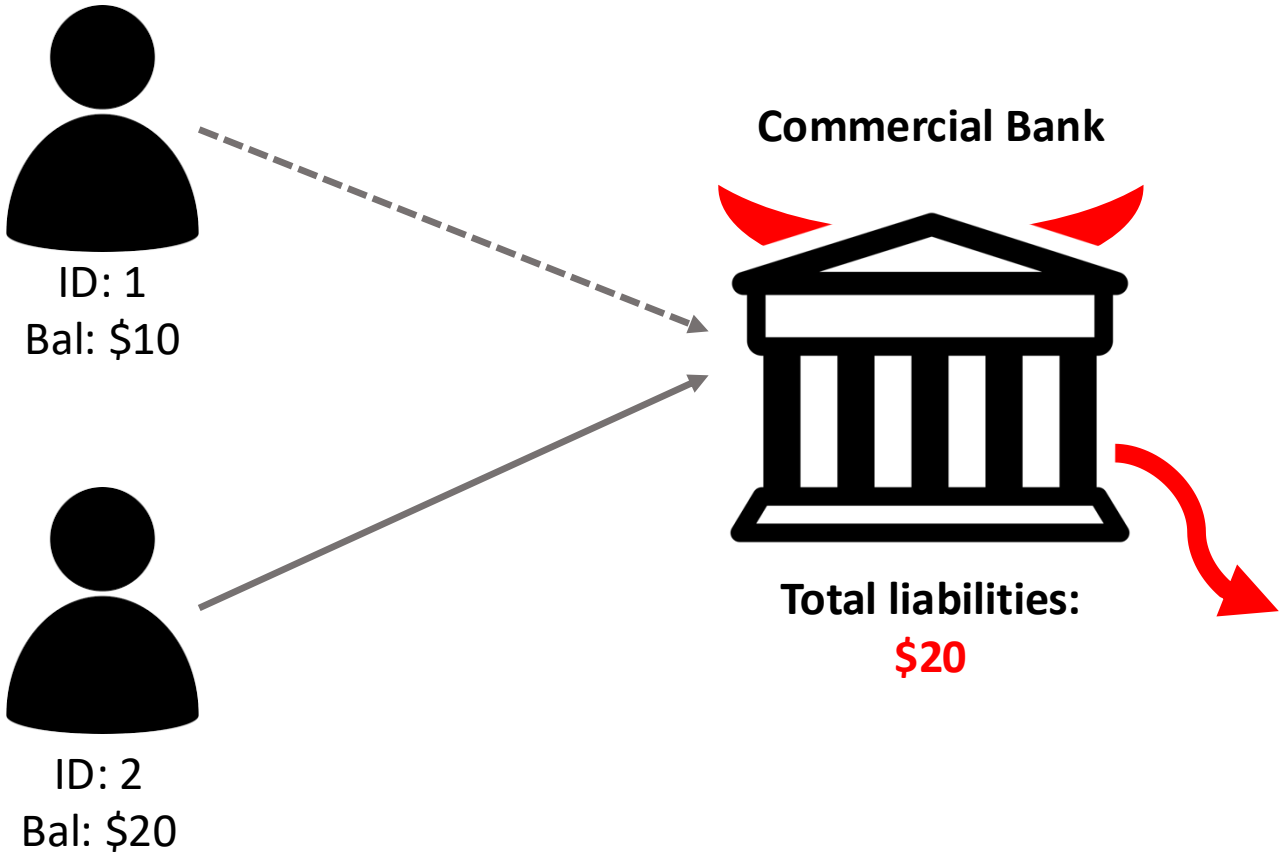
Francesca Falzon, Kaoutar Elkhiyaoui, Yacov Manevich, Angelo De Caro

IBM Research

Proof of Liabilities (PoL)



Proof of Liabilities (PoL)



Proof of Liabilities (PoL)

Proof of Liabilities (PoL)

Proof of Liabilities (PoL)

FINANCIAL TIMES

HOME WORLD US COMPANIES TECH MARKETS CLIMATE OPINION WORK & CAREERS LIFE & ARTS HTSI

Chinese business & finance + Add to myFT

China allows first commercial bank to go bankrupt in almost 20 years

Decision was taken after regulators found 'severe insolvency' during a review of the business

Save

Proof of Liabilities (PoL)

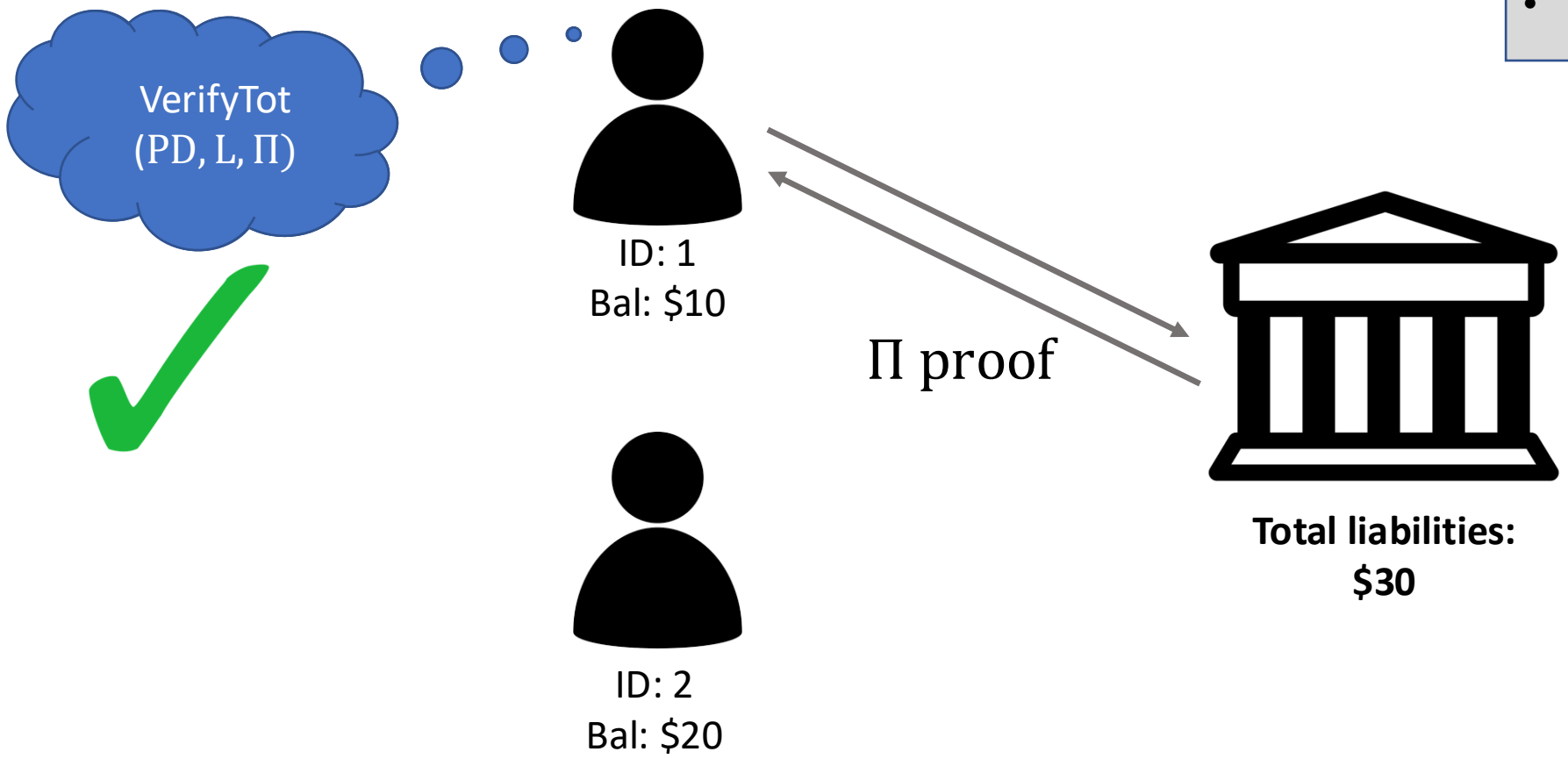
Proof of Liabilities (PoL)

November 8, 2022 Stephan Fiedler FTX 0

Proof of Liabilities (PoL)

Blockchain:

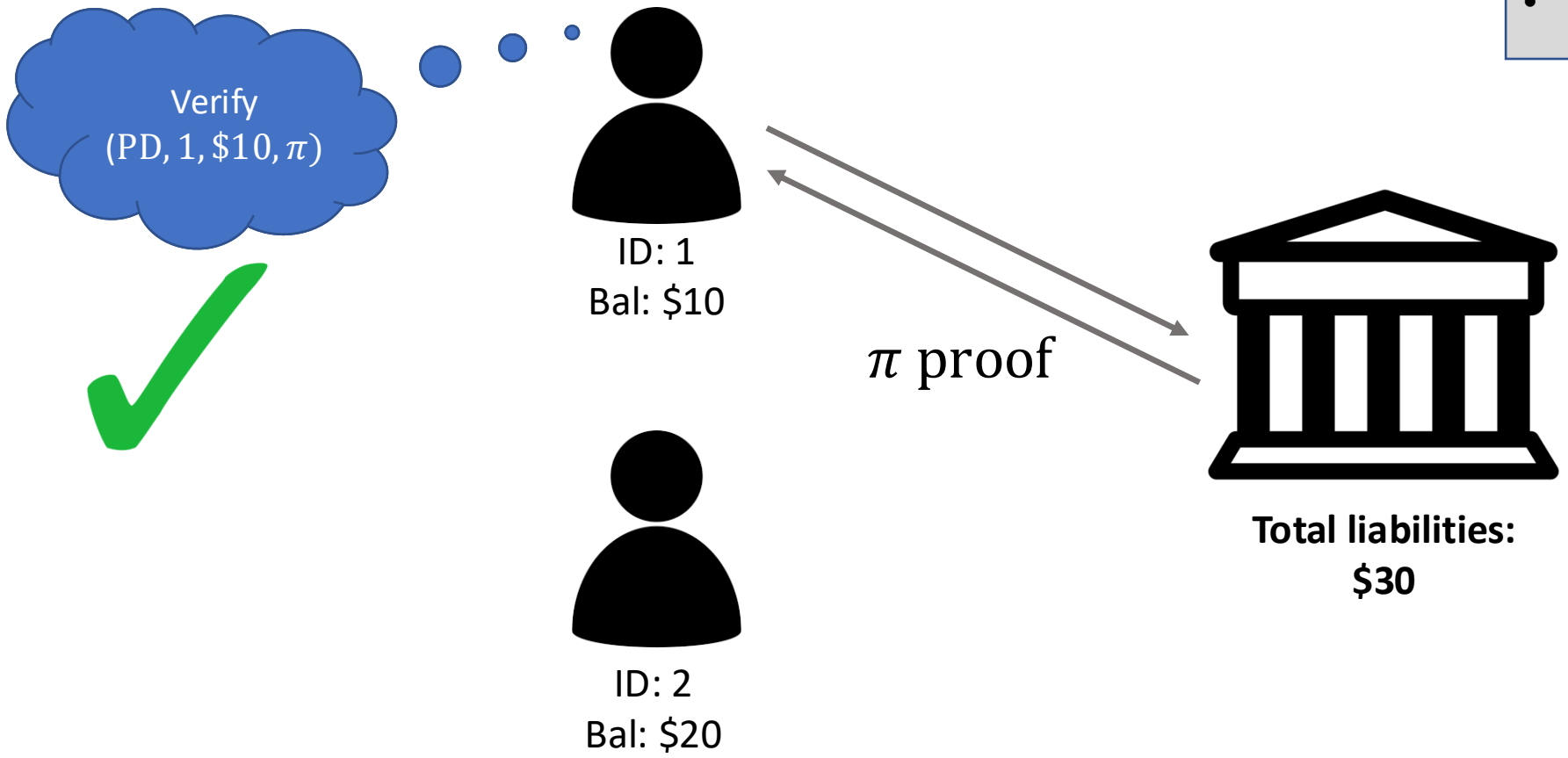
- Public data including commitment to total liability (PD)
- Total liability (L)



Proof of Liabilities (PoL)

Blockchain:

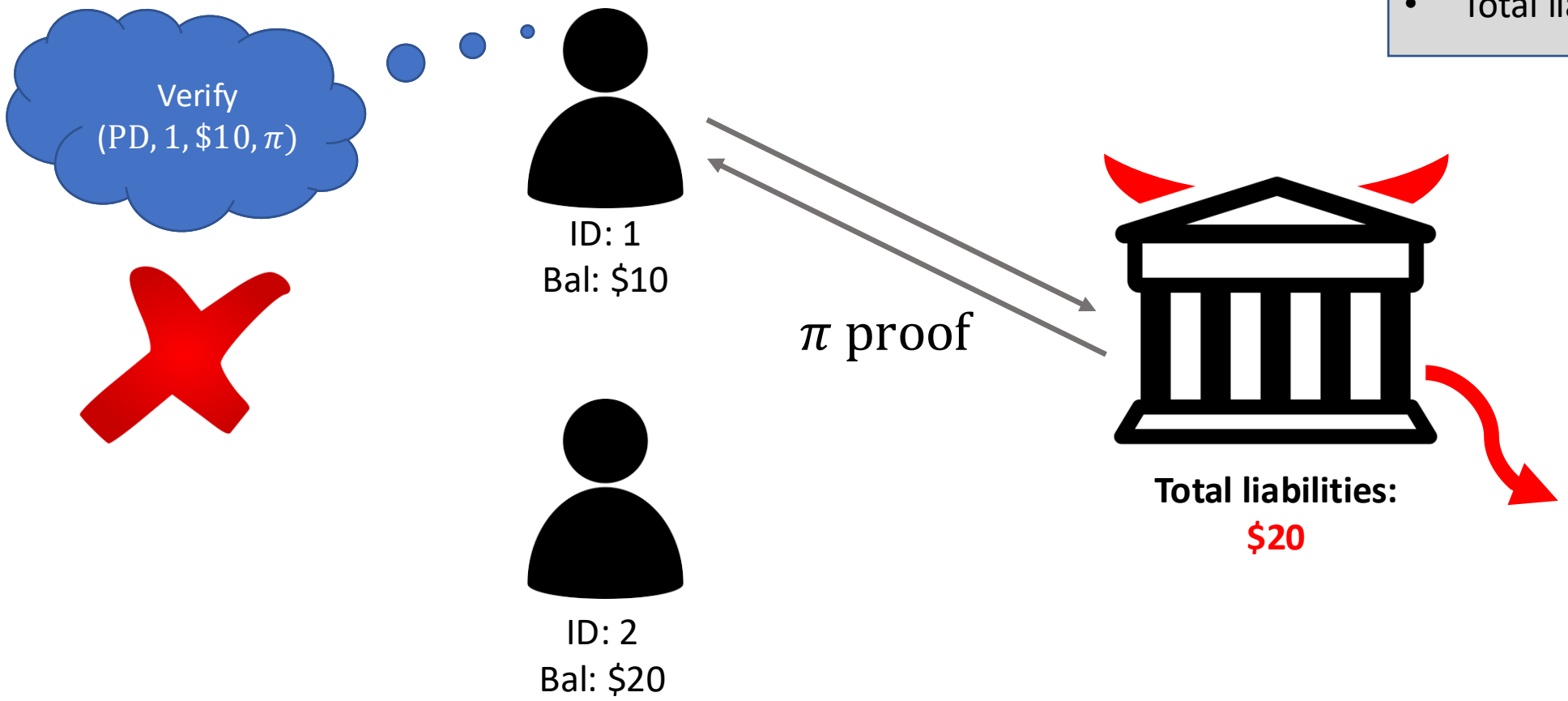
- Public data including commitment to total liability (PD)
- Total liability (L)



Proof of Liabilities (PoL)

Blockchain:

- Public data including commitment to total liability (PD)
- Total liability (L)



Background

- Companies that store currency on behalf of clients (e.g. commercial banks) may be audited to ensure financial solvency i.e., that clients can retrieve their deposits in full.
- Companies may fudge their numbers and claim solvency even when they do not have the necessary funds.
- **Proof of liabilities (PoL)** is a cryptographic primitive that solves half of solvency auditing problem. Its goal is to prove the amount the audited entity owes to its clients.
- The other half of the solution is **proof of reserves (PoR)**. Its goal is to prove to the clients that it has sufficient funds to cover their liabilities.

Use Cases

- Proof of solvency
- Retail CBDCs
- Publishing COVID cases
- Negative voting
- Compliance with **data privacy legislation** like the EU's GDPR and Switzerland's nFADP.

Overview

- Background
- **Prior Work and Novelty**
- Problem Statement
- Our Scheme

Prior PoL Schemes

- Maxwell (2014)
  - First scheme, but insecure
- Maxwell+ (2018)
  - Fixes flaw from original, but leaks # of users
- Provisions (2015)
  - Uses Pedersen commitments, publishes large amount of data to public ledger

Prior PoL Schemes

- DAPOL (2020)
  - Hides # users, but the padding strategy is flawed
- DAPOL+ (2021)
  - Fixes flaw from previous, longer authentication paths

This work

- We describe and implement a novel data structure for the PoL problem: the **sparse summation Verkle tree (SSVT)**
- Our SSVT data structure
  - Minimizes information leakage
  - Hides the number of users
  - Enables optimization through the batching of proofs
- We leverage inner product arguments and present three **new zero-knowledge protocols over vector commitments**.
- We further **optimize** the three protocols to support aggregation.

Overview

- Background
- Prior Work and Novelty
- **Problem Statement**
- Our Scheme

Problem Statement

- The audited entity (e.g. the commercial bank) does NOT have any incentive to increase its liabilities.
- How can we ensure that the liabilities to each user are included in the total?
- And how can we do so with short proofs while ensuring privacy to the users and minimizing leakage?

Entities

- **Users**: The set of users U are the parties who store their assets with the organization.
- **Prover**: The prover P operates on behalf of the organization and is the entity liable to the users. The prover's goal is to prove to users that their P's liabilities to them are included in the total.

PoL Scheme

- A proof of liability (PoL) scheme is a tuple of algorithms with the following syntax.
  - $(\mathrm{PD}, \mathrm{SD}) \leftarrow\$ \mathrm{Setup}(1^{\kappa}, \mathrm{DB})$
  - $(\Pi, \mathrm{L}) \leftarrow \mathrm{ProveTot}(\mathrm{DB}, \mathrm{SD})$
  - $\mathrm{b} \leftarrow \mathrm{VerifyTot}(\mathrm{PD}, \mathrm{L}, \Pi)$.
  - $\pi \leftarrow \mathrm{Prove}(\mathrm{DB}, \mathrm{SD}, \mathrm{id})$
  - $\mathrm{b} \leftarrow \mathrm{Verify}(\mathrm{PD}, \mathrm{id}, \ell, \pi)$

Threat Model and Assumptions

1. Prover has no incentive to increase its total liabilities
   - May try to claim less liability to a non-corrupt user or
   - May try to remove its liability to a non-corrupt user completely.
2. Users establish a secure communication with the prover.
3. The adversary may corrupt users and try to learn information about the *non-corrupt* users.
4. There exists a public bulletin board (PBB) that provides consensus e.g., a blockchain.
5. The public data (PD) computed at setup is published to the PBB.

Overview

- Background
- Prior Work and Novelty
- Problem Statement
- **Our Scheme**

Building Blocks

- **Collision-resistant hash functions**
  - A hash function H is **collision-resistant** if it is hard to find two inputs that hash to the same output.
- **Vector commitments (VCs)**
  - Allow us to commit to an ordered sequence of values $(m_1, \ldots, m_n)$ such that it is later possible to open the resulting commitment C only w.r.t. a specific position.
  - Must satisfy **position binding =** a polynomially bounded adversary with knowledge of the public parameters cannot open commitment C at position i to two different valid values.

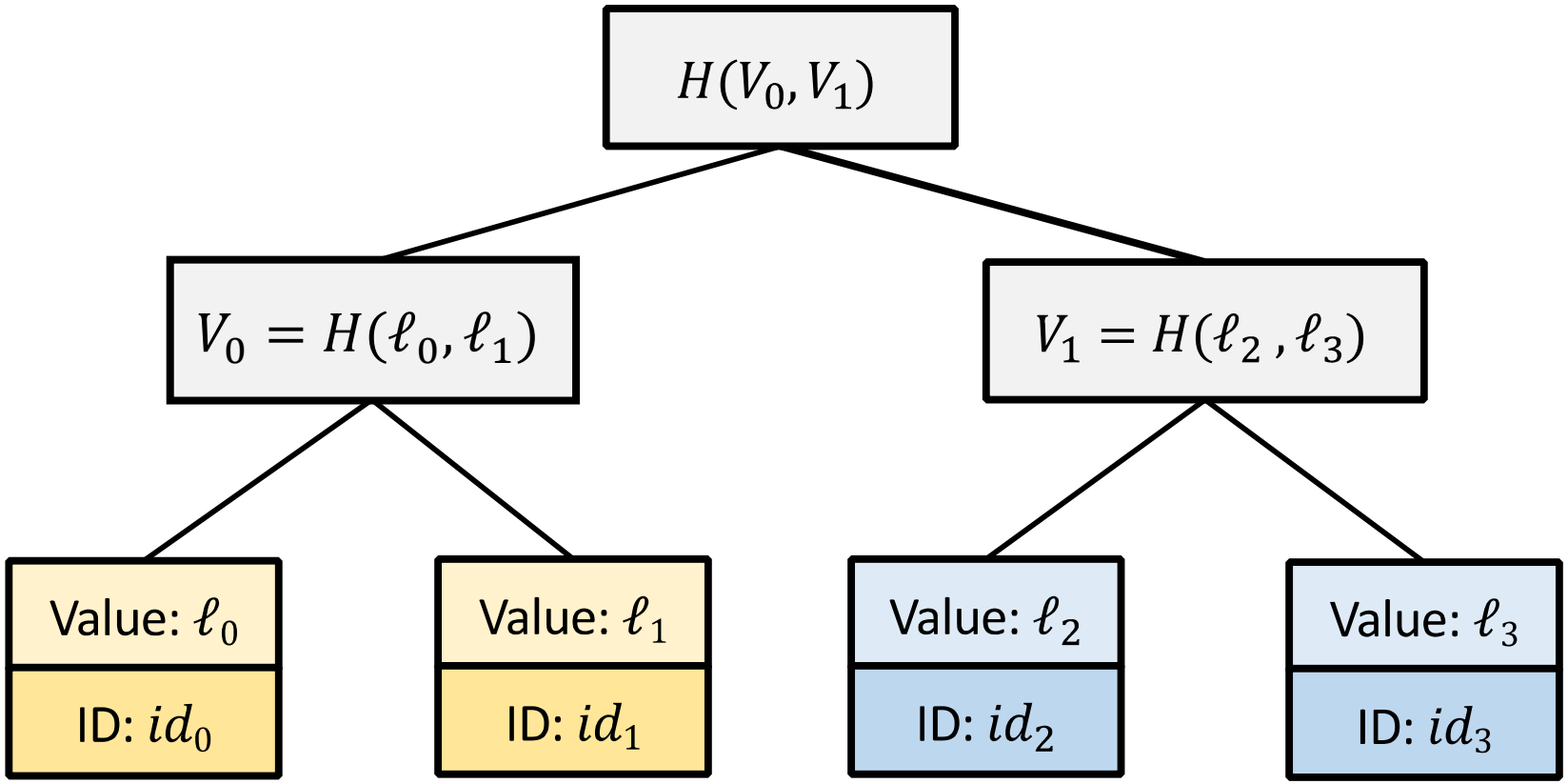
Building Blocks

- **Zero-knowledge proofs**
  - Enable one party (the prover) to prove to another party (the verifier) that a given statement is true without leaking any additional information. Should satisfy the following 3 properties.
  - **Completeness =** if the statement is true, an honest verifier will be convinced of this fact.
  - **Soundness =** if the statement is false, no cheating prover can convince an honest verifier that it is true (except with negl prob).
  - **Zero-knowledge =** if the statement is true, the verifier learns nothing except that the statement is true.

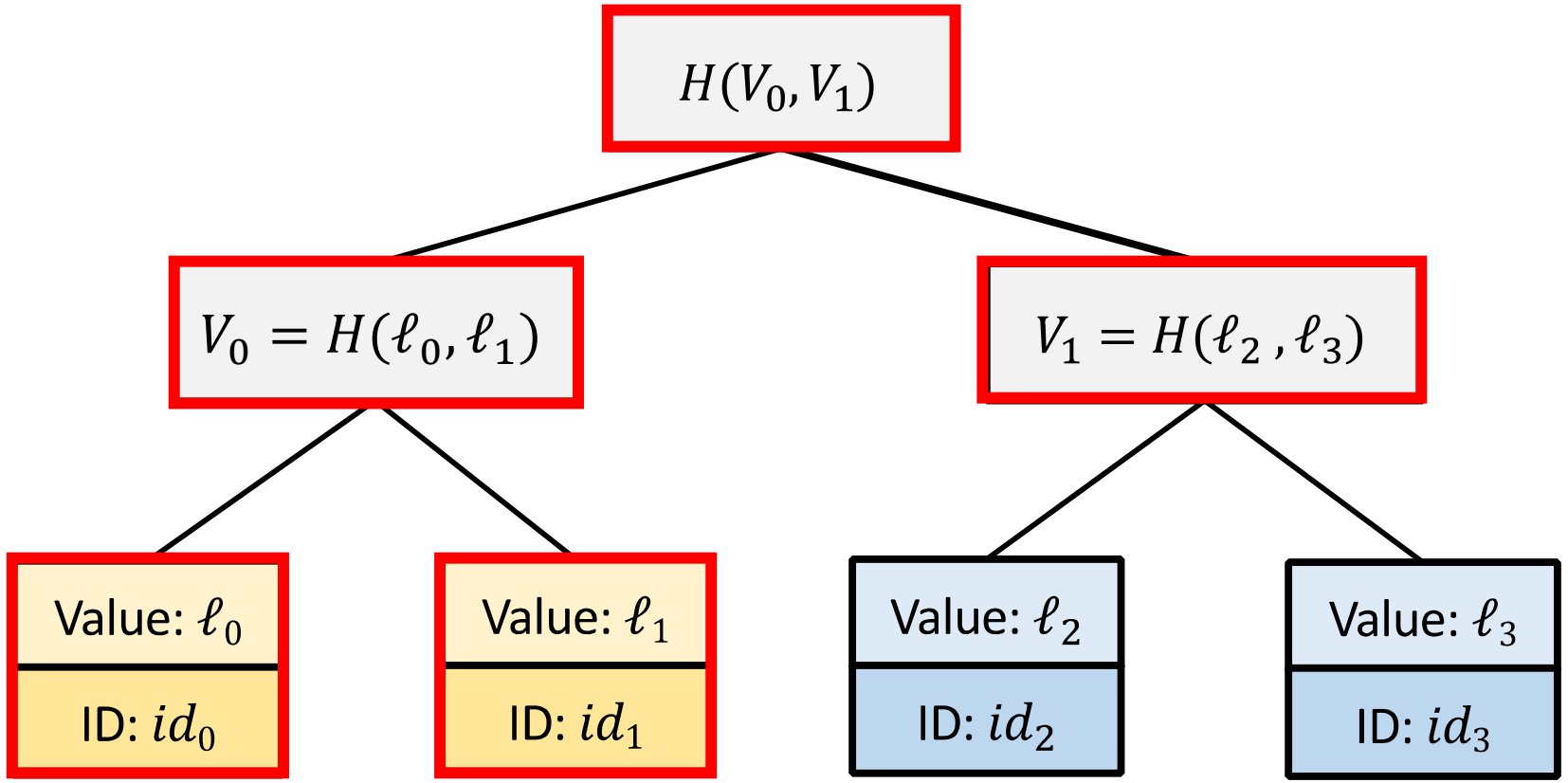
Our Solution: In a Nutshell

- Introduce a novel data structure, the **sparse summation Verkle tree** to accumulate liabilities.
  - Results in short authentication paths
  - Hides the total number of users
  - Further mitigates leakage
  - Benefits from the efficiency of vector commitments

A Merkle Tree

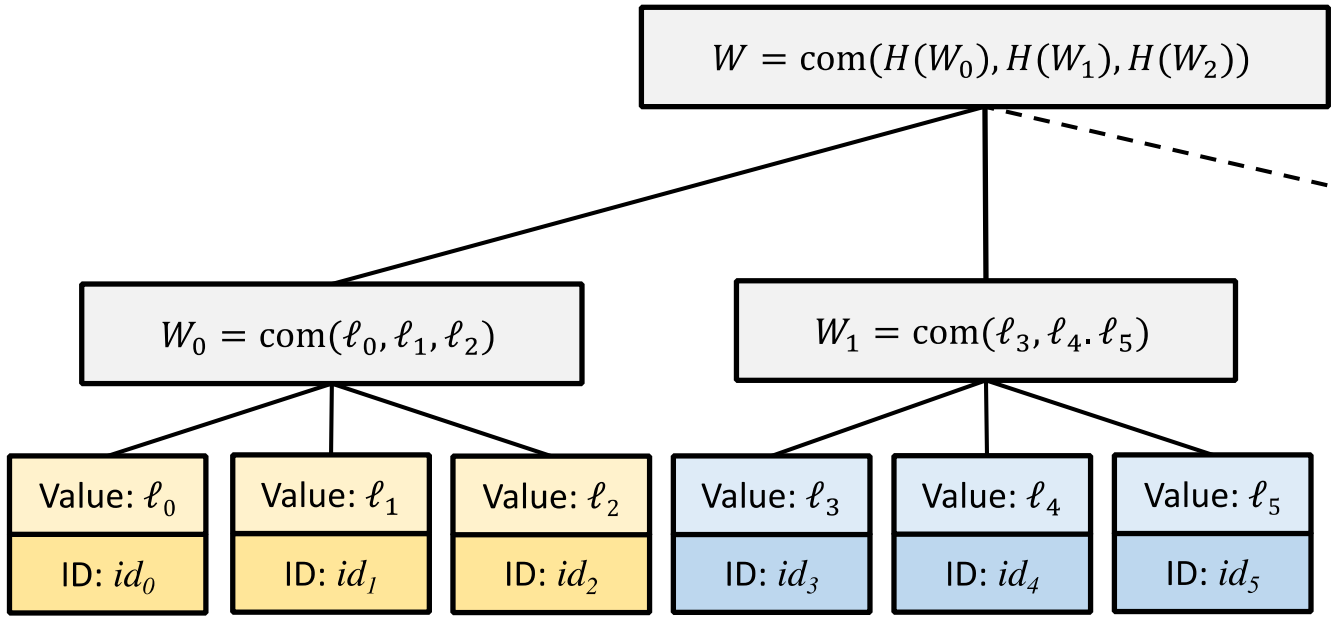


A Merkle Tree



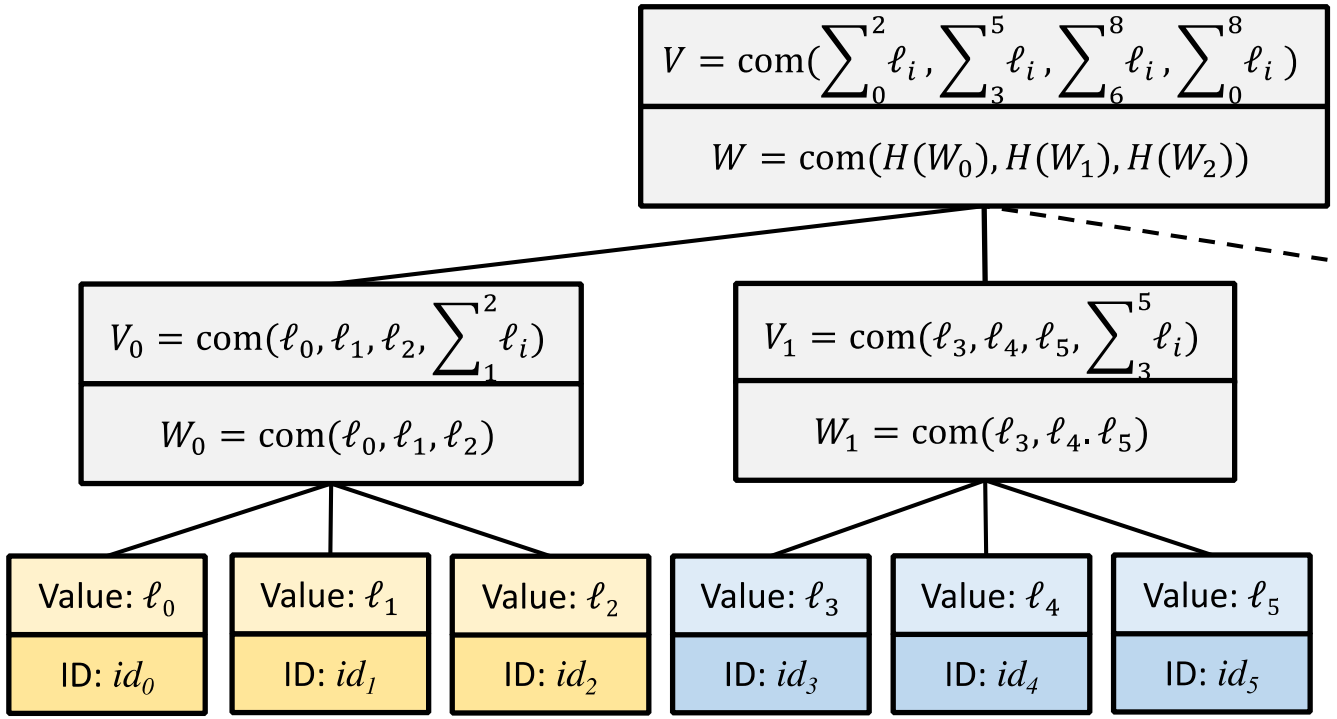
A Verkle Tree

- A **Verkle tree** is like a Merkle tree, but it replaces the hash values of the inner nodes with vector commitments.
- Each inner node is labeled with a commitment to an n-length vector.
- Each index of the vector corresponds to (the hash of) one of the n child nodes.



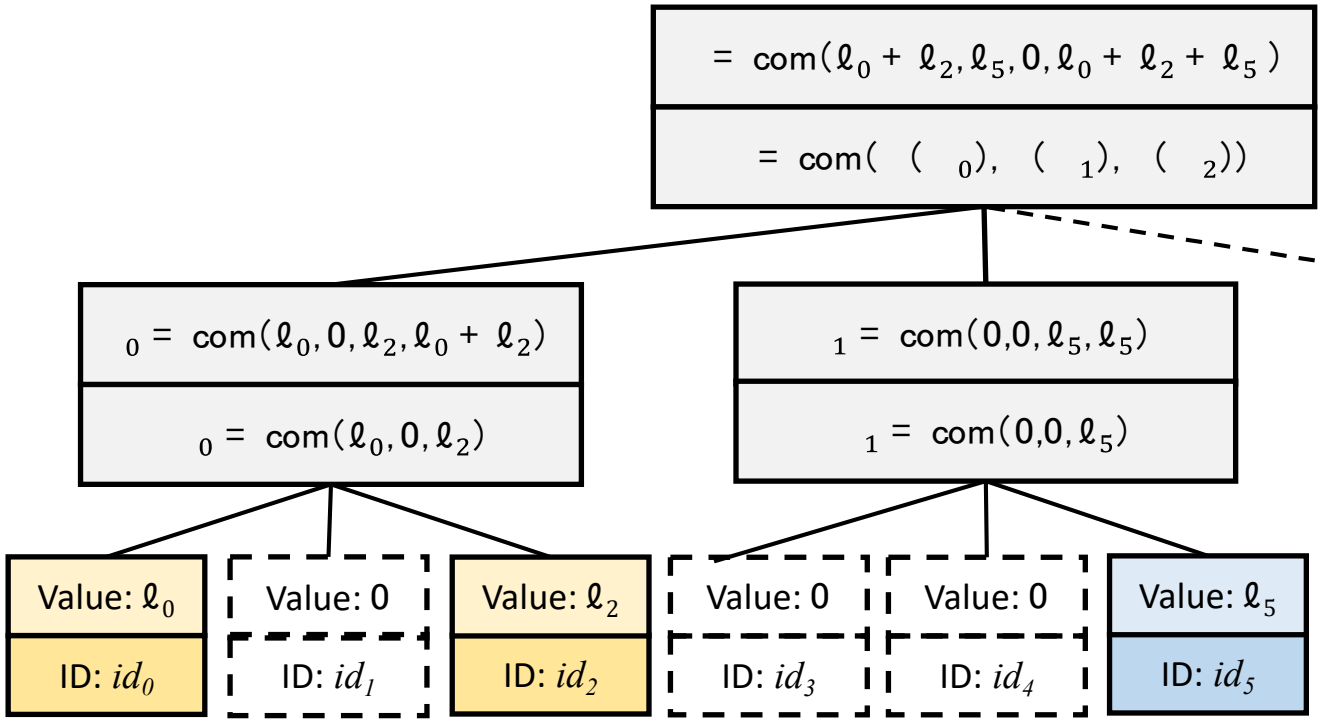
A Summation Verkle Tree

- The inner nodes are labeled with an additional commitment to an $n+1$ length vector.
- First n indices correspond to values of the children.
- The last index corresponds to the sum of the previous entries.



A Sparse Summation Verkle Tree

- A sparse summation Verkle tree (SSVT) over a set S is a summation Verkle tree whose leaves are 1-1 with the elements in S.
- Most elements in S are not in the membership set, so the tree can be represented compactly using only the necessary nodes.

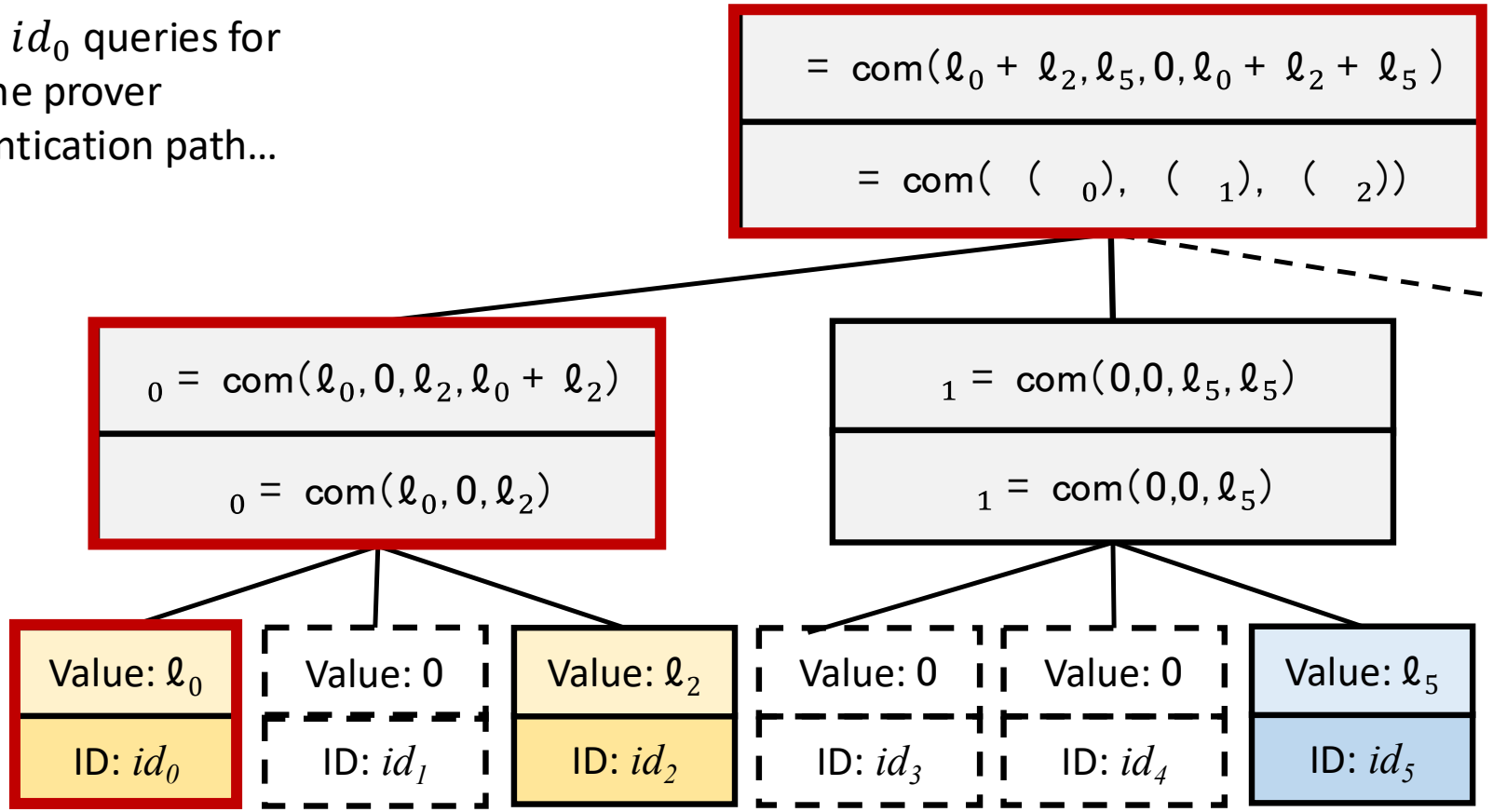


PoL Scheme Overview

- The leaves of the SSVT corresponds to the universe of possible users.
- Users are assigned to leaves of the SSVT by hashing their ID.
- When a user queries for inclusion it receives the authentication path…

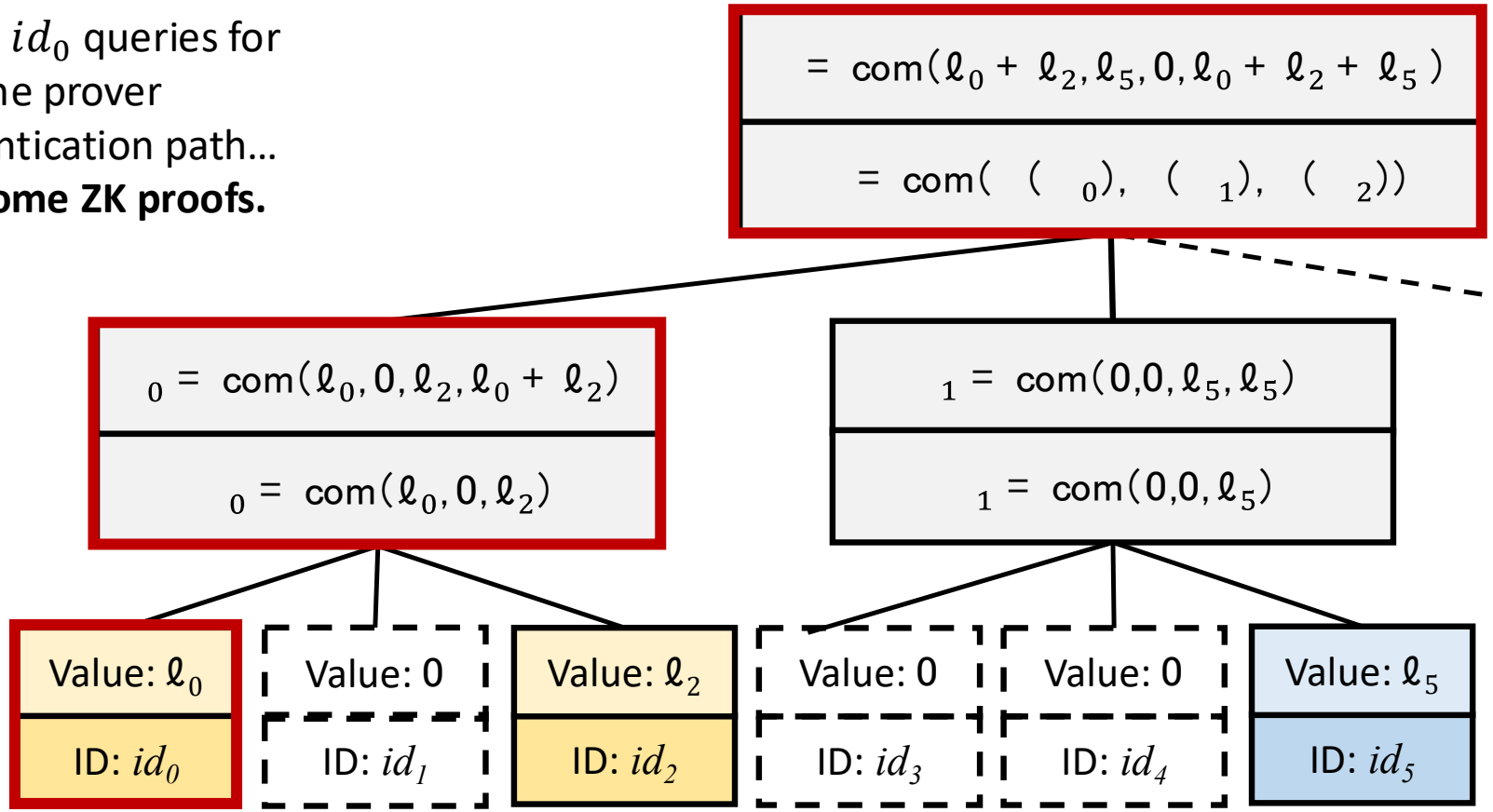
PoL Scheme Overview

If the user with ID id_0 queries for its liability, then the prover returns the authentication path…



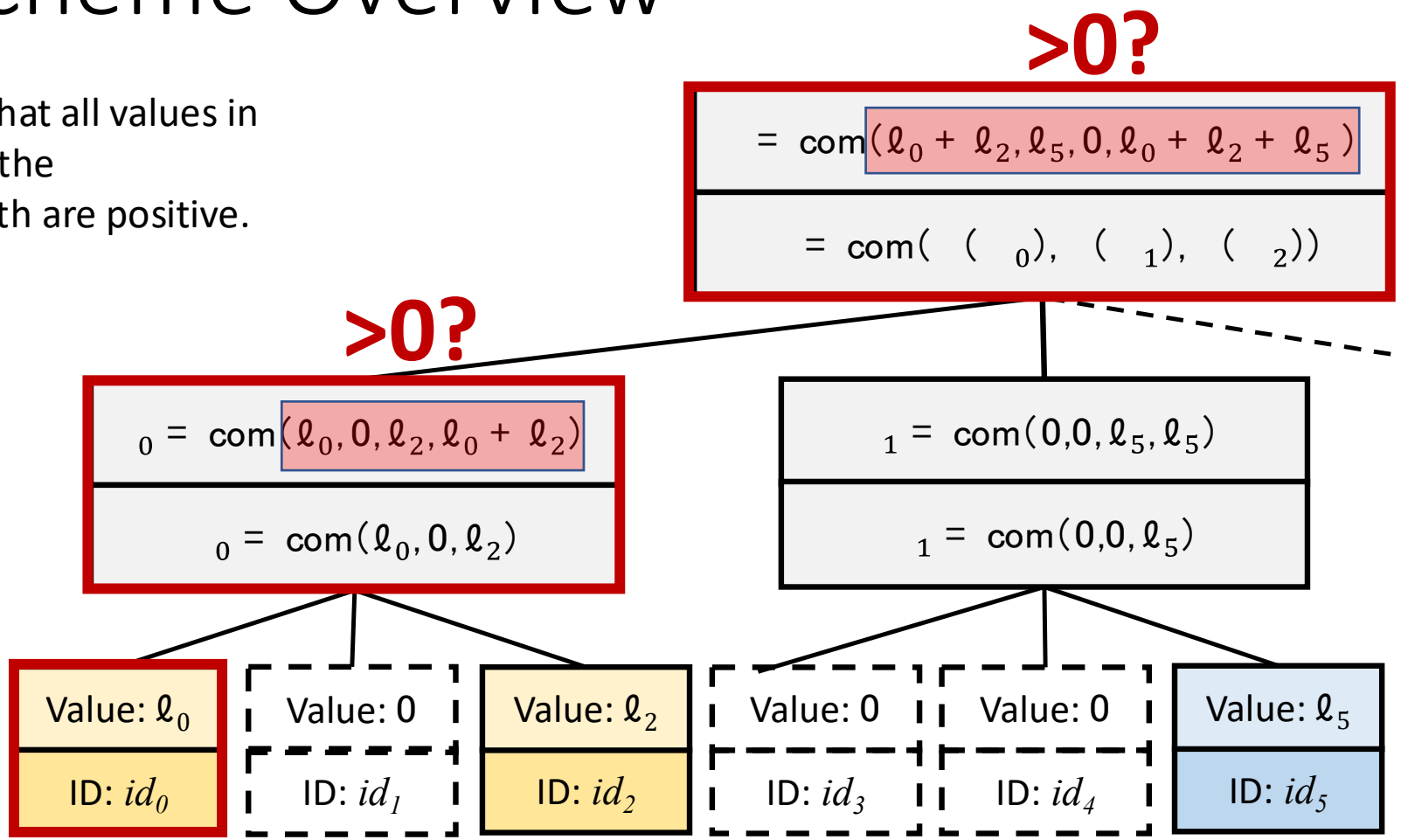
PoL Scheme Overview

If the user with ID id_0 queries for its liability, then the prover returns the authentication path…
…together with some ZK proofs.



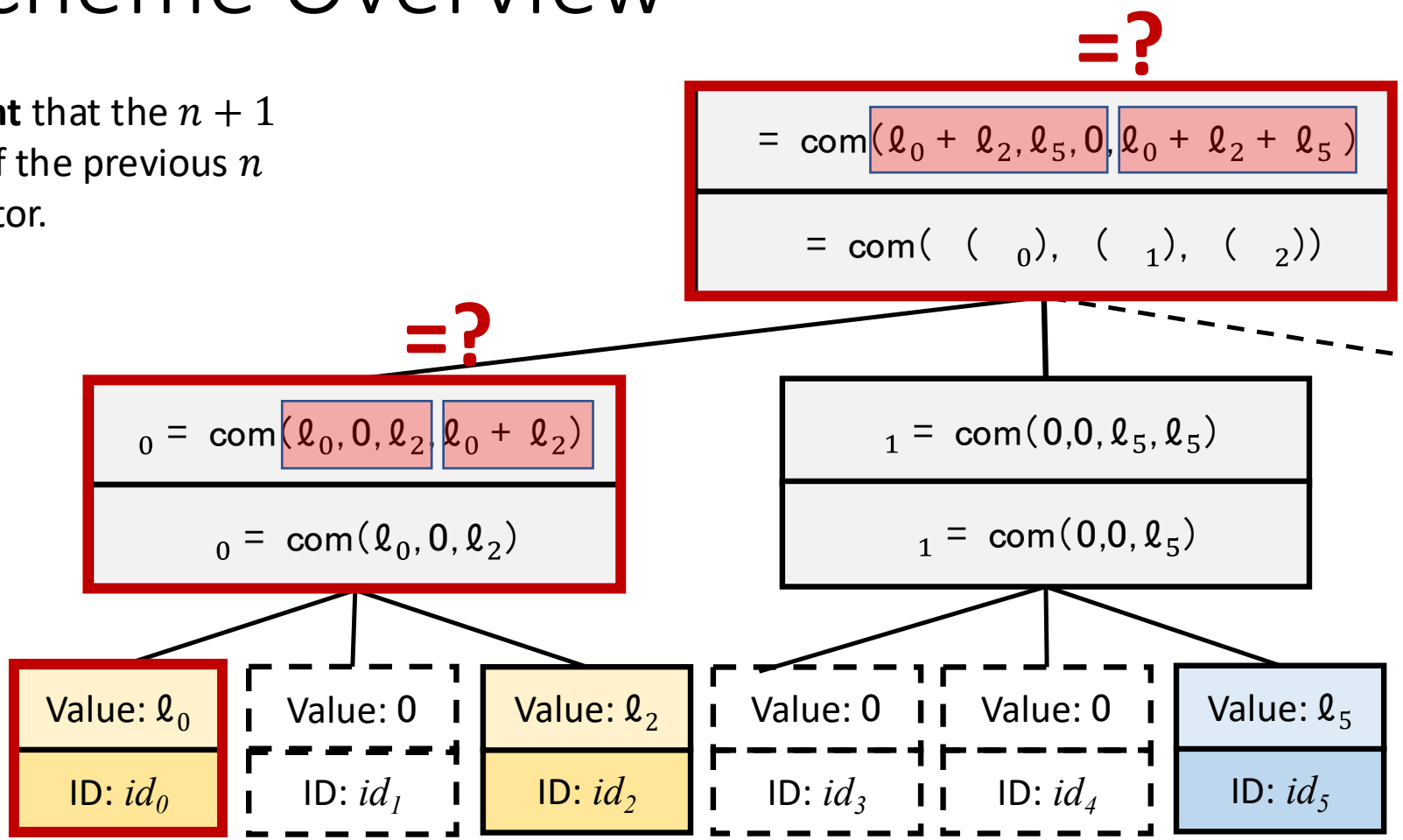
PoL Scheme Overview

1. A **range proof** that all values in the vectors along the authentication path are positive.



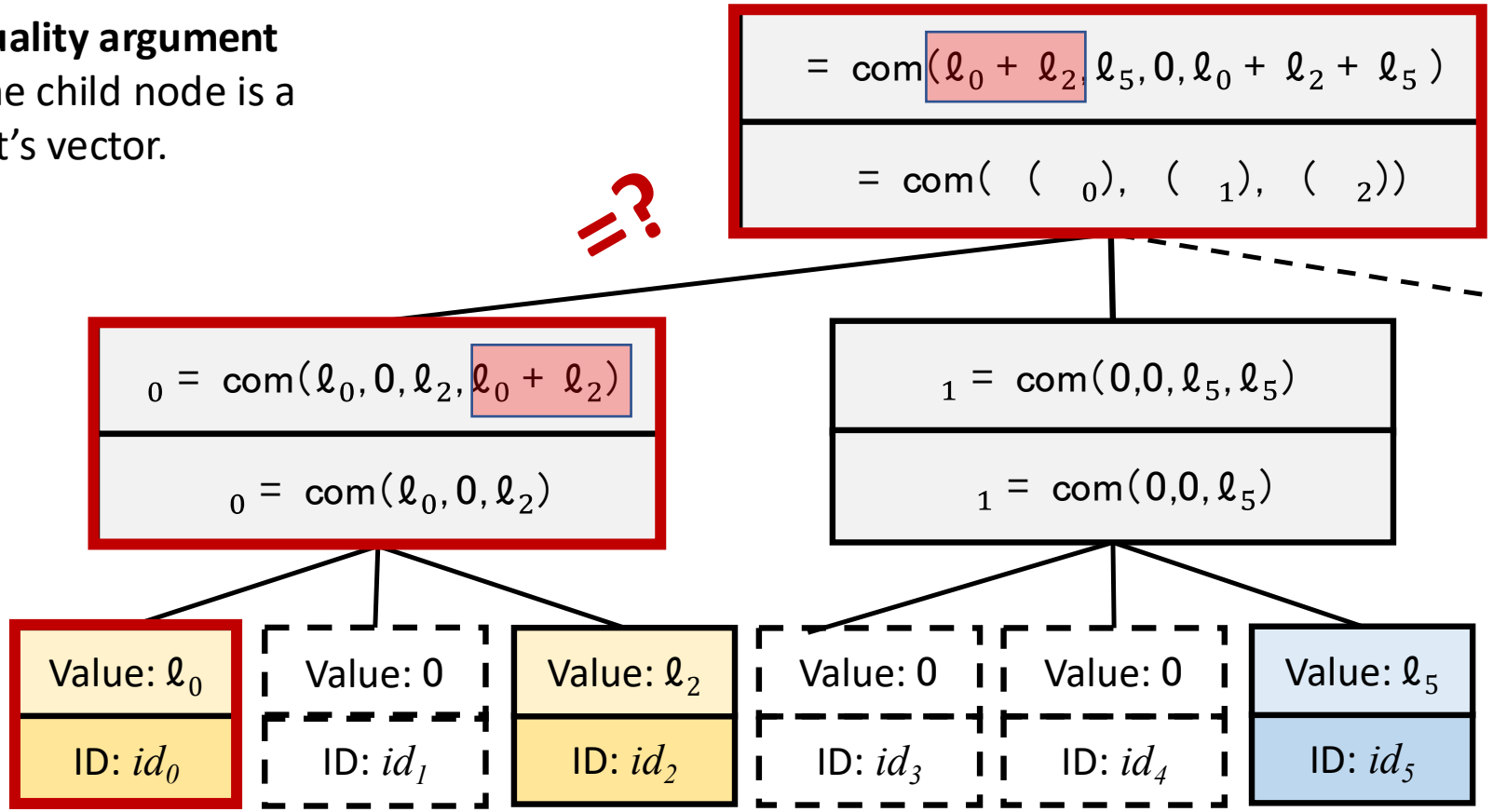
PoL Scheme Overview

2. A **sum argument** that the $n+1$ term is the sum of the previous n entries of the vector.



PoL Scheme Overview

3. An **opening equality argument** that the sum of the child node is a term in the parent's vector.



The Protocols

- Our scheme supports the black box use of *any* range protocol, sum argument, and opening equality argument that takes as input Pedersen-like vector commitments.
- For completeness, we provide instantiations of all protocols.
- We further inhance our scheme by extending these protocols to support the batching of proofs (i.e., so we can batch the proofs along the authnetication path and reduce communication and verification time).

Conclusion

- We present a novel data structure, the **Sparse Summation Verkle Tree (SSVT)** for use in Proof of Liability schemes.
- We describe a **POL scheme using an SSVT** that can guarantee liabilities to users in a privacy-preserving manner.
- We present **three novel zero-knowledge protocols** over vector commitment which can also be optimized through aggregation of proofs.
- We **implement our scheme** and test its performance on two different sized trees representing two real-world use-cases.

Thank you!
Questions?