Verifiable Efficient Modular Databases

Without SNARKs

Alberto Trombetta (joint work with Vincenzo Botta, Simone Bottoni, Matteo Campanelli, Emanuele Ragnoli)

Integrity in Databases

- Databases are at the hearth of our technological infrastructure
- Outsourcing them is very common
- This introduces risks:
  - “AWS, would you give me the response to this query?”
  - **But how do we know the response is correct?**
  - Arbitrary faults, malicious behaviour,…

Integrity in Databases

- Databases are at the hearth of our technological infrastructure
- Outsourcing them is very common
- This introduces risks:
    - “AWS, would you give me the response to this query?”
    - **But how do we know the response is correct?**
    - Arbitrary faults, malicious behaviour,…

Verifiable Databases (VDB)

Are a cryptographic solution to this problem

Further Motivations fo VDBs

- **Implication of Verifiable Databases**: not having to trust your DB provider
- **Pipe dream** ≈ every data flow from every DB API authenticated through a verifiable DB
  - Analogy: HTTPS. And its ubiquity
  - Potential outcome: information flow that is fully certified cryptogrphically
  - Even a partial version of the pipe dream might be useful…

Further Motivations fo VDBs

- **Implication of Verifiable Databases**: not having to trust your DB provider
- **Pipe dream** ≈ every data flow from every DB API authenticated through a verifiable DB
    - Analogy: HTTPS. And its ubiquity
    - Potential outcome: information flow that is fully certified cryptogrphically
    - Even a partial version of the pipe dream might be useful…
- **In blockchain settings**:
    - Providing proofs for aswers from ‘coprocessors’

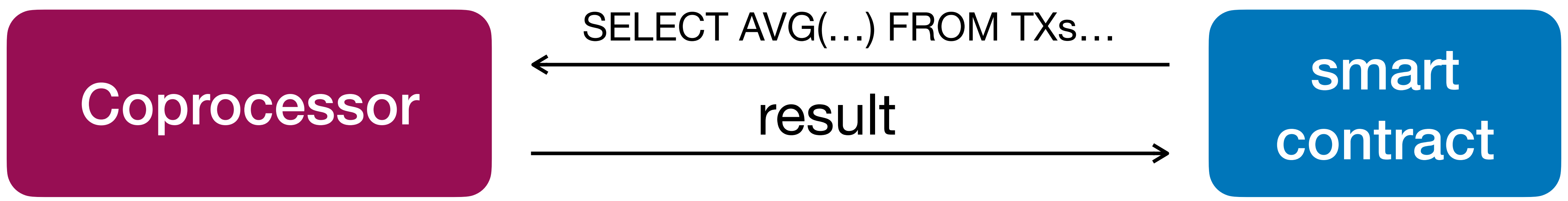
Further Motivations fo VDBs

- **Implication of Verifiable Databases:** not having to trust your DB provider
- **Pipe dream** ≈ every data flow from every DB API authenticated through a verifiable DB
  - Analogy: HTTPS. And its ubiquity
  - Potential outcome: information flow that is fully certified cryptogrphically
  - Even a partial version of the pipe dream might be useful…
- **In blockchain settings**:
  - Providing proofs for aswers from 'coprocessors'
  - Coprocessor sends *SomeAnalysis(chain)* to the chain

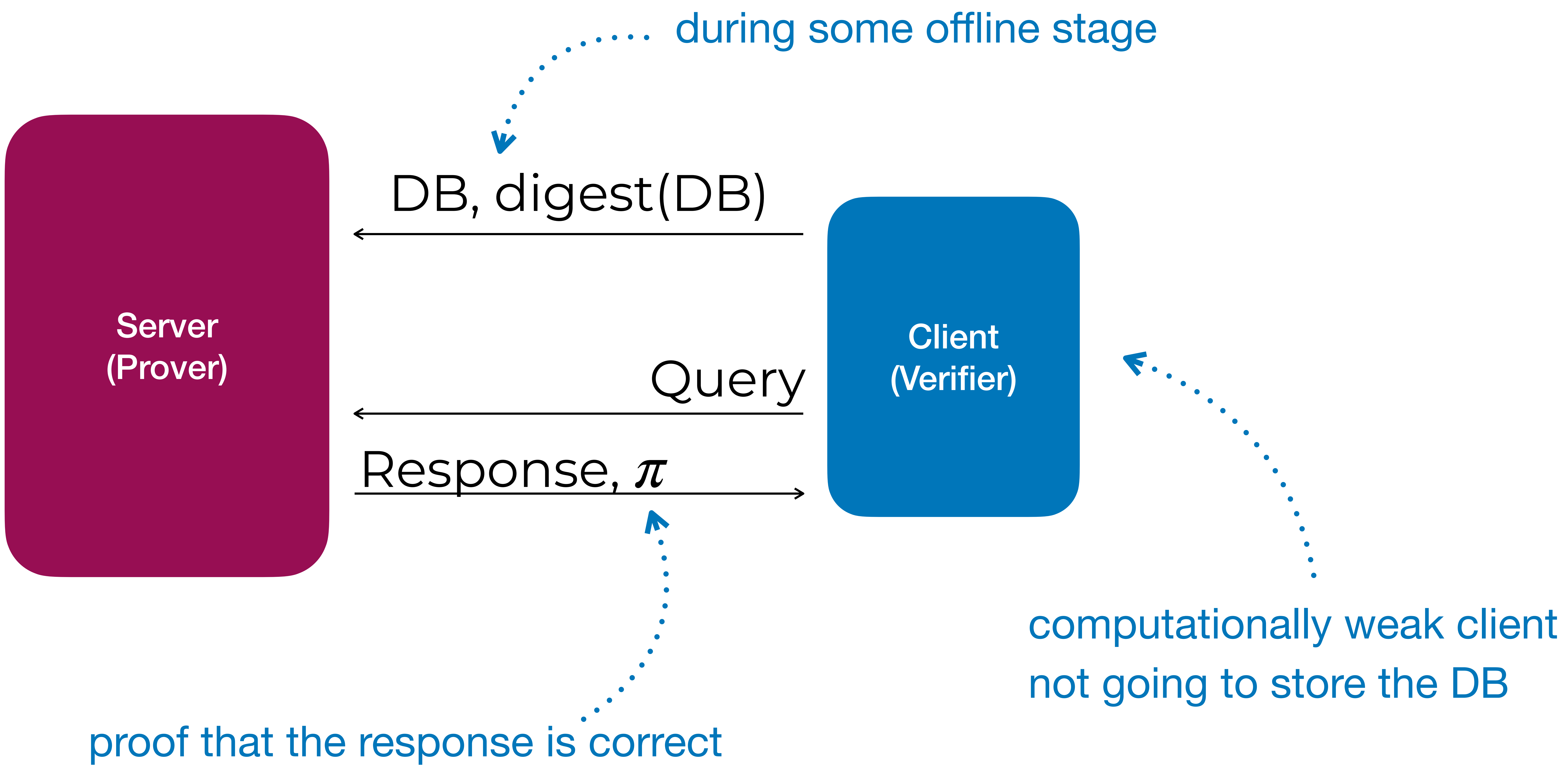
lagrange

AXIOM

SPACEANDTIME



Verifiable Databases



Desiderable Features of VDBs

Efficiency-related

- Fast (Prover and Verifier)
- Publicly verifiable
  - Important to establish trust levels of data traces
- Non-interactive, with short proofs
  - Especially important in smart contracts

Desiderable Features of VDBs

Efficiency-related

- Fast (Prover and Verifier)
- Publicly verifiable
    - Important to establish trust levels of data traces
- Non-interactive, with short proofs
    - Especially important in smart contracts

Security-related

- Based on solid cryptographic assumptions (of course)
- Simple
    - Easy auditable; easier to reason about
    - Less vulnerable
    - More maintainable; easier to patch

How Do We Build VDBs?

Traditional notions of integrity → More fine-grained notions of integrity → Computational Integrity

Signatures, hashes

Authenticated data structures (Merkle trees, …)

General cryptographic Proofs

How Do We Build VDBs?

VDBs

? ?

Traditional notions of integrity → More fine-grained notions of integrity → Computational Integrity

Signatures, hashes

Authenticated data structures (Merkle trees, …)

General cryptographic Proofs

How Do We Build VDBs?

Both approaches are used
They lead to different tradeoofs

VDBs

? ?

Traditional notions of integrity → More fine-grained notions of integrity → Computational Integrity

Signatures, hashes

Authenticated data structures (Merkle trees, …)

General cryptographic Proofs

VDBs as ADSs

Queries as examples of general computations

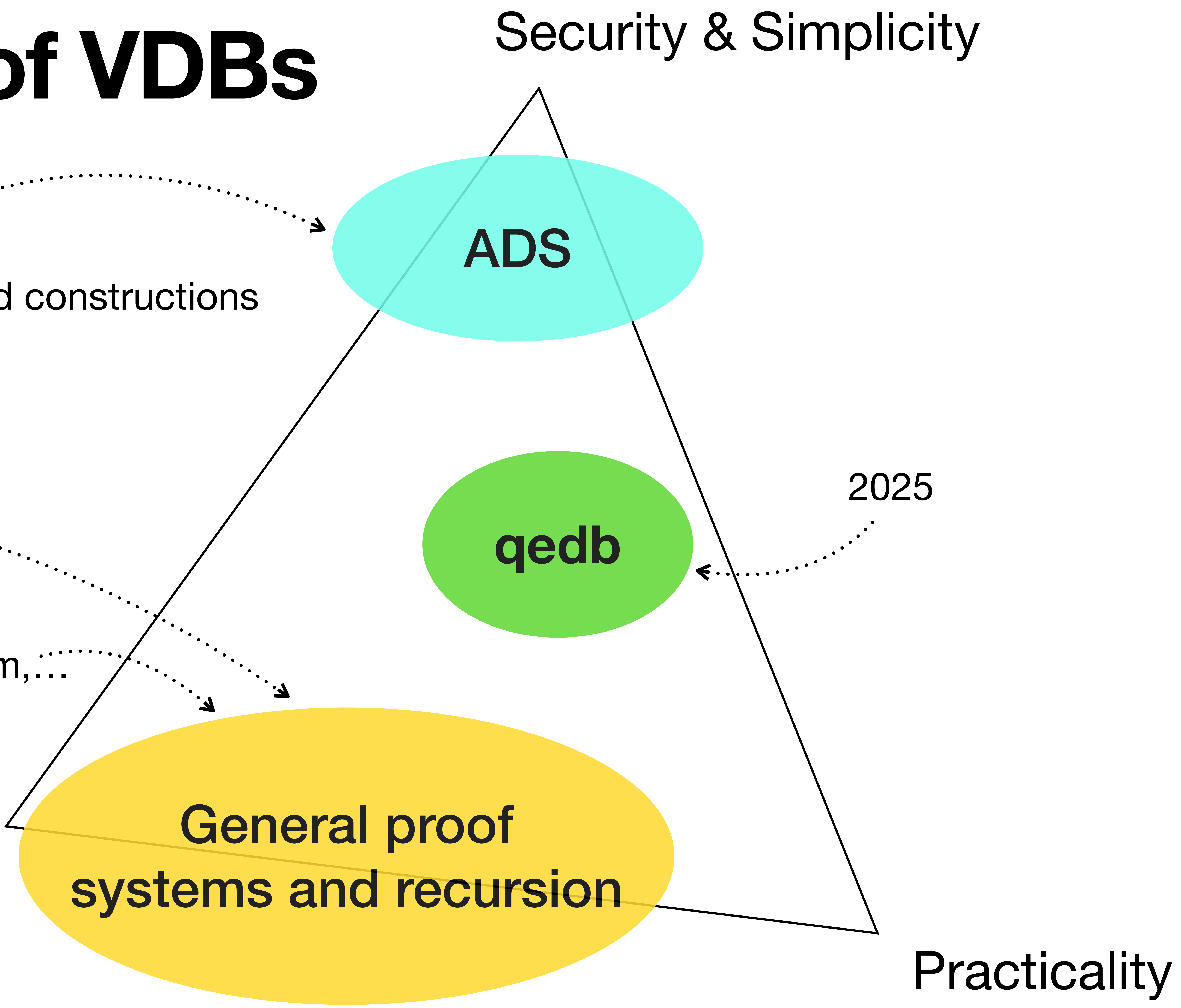
Landscape of VDBs

∼2009-2015: **IntegriDB**

and - mostly - accumulators-based constructions

2017: **vSQL**

2023-2025: Lagrange Labs, Axiom,…



Tradeoffs

General-purpose solutions

- Expressive ✓
- Can be very efficient ✓
  - Fast proving time (with the right number of GPUs and investment) ✓
  - Short proof size/small verification cost ✓
- Sledghammer approach to verifiable SQL ✗
- Extemely complex stack ✗
- Suboptimal developer experience ✗

Tradeoffs

General-purpose solutions

- Expressive ✓
- Can be very efficient ✓
  - Fast proving time (with the right number of GPUs and investment) ✓
  - Short proof size/small verification cost ✓
- Sledghammer approach to verifiable SQL ✗
- Extemely complex stack ✗
- Suboptimal developer experience ✗

Authenticated Data Structures solutions

- Simple hash-based authentication and modern accumulators ✓
- Large proof size ✗
- Preprocessing and proving is memory intensive ✗
- Constrained expressivity ✗

QEDB

- A VDB that is:
  - Higly **efficient**
  - Highly **expressive**
  - From **simple** building blocks

First scheme with **proof size independent of DB size**

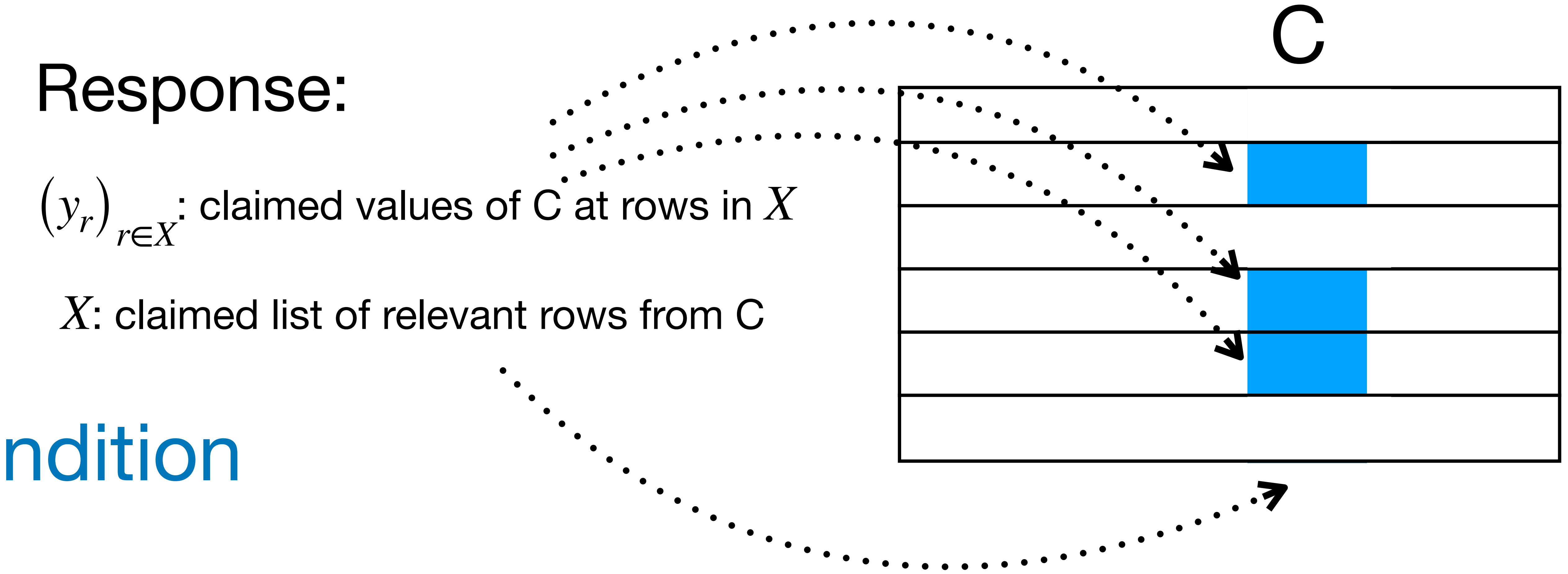
Generates a proof in seconds on a common laptop
(For a 1 million row DB)

No quadratic behavior for JOINs
(through new techniques)

More expressive than other ADS-based approaches

No general purpose SNARKs
Instead: specialized vector commitments
And accumulators

Idealized VDBs



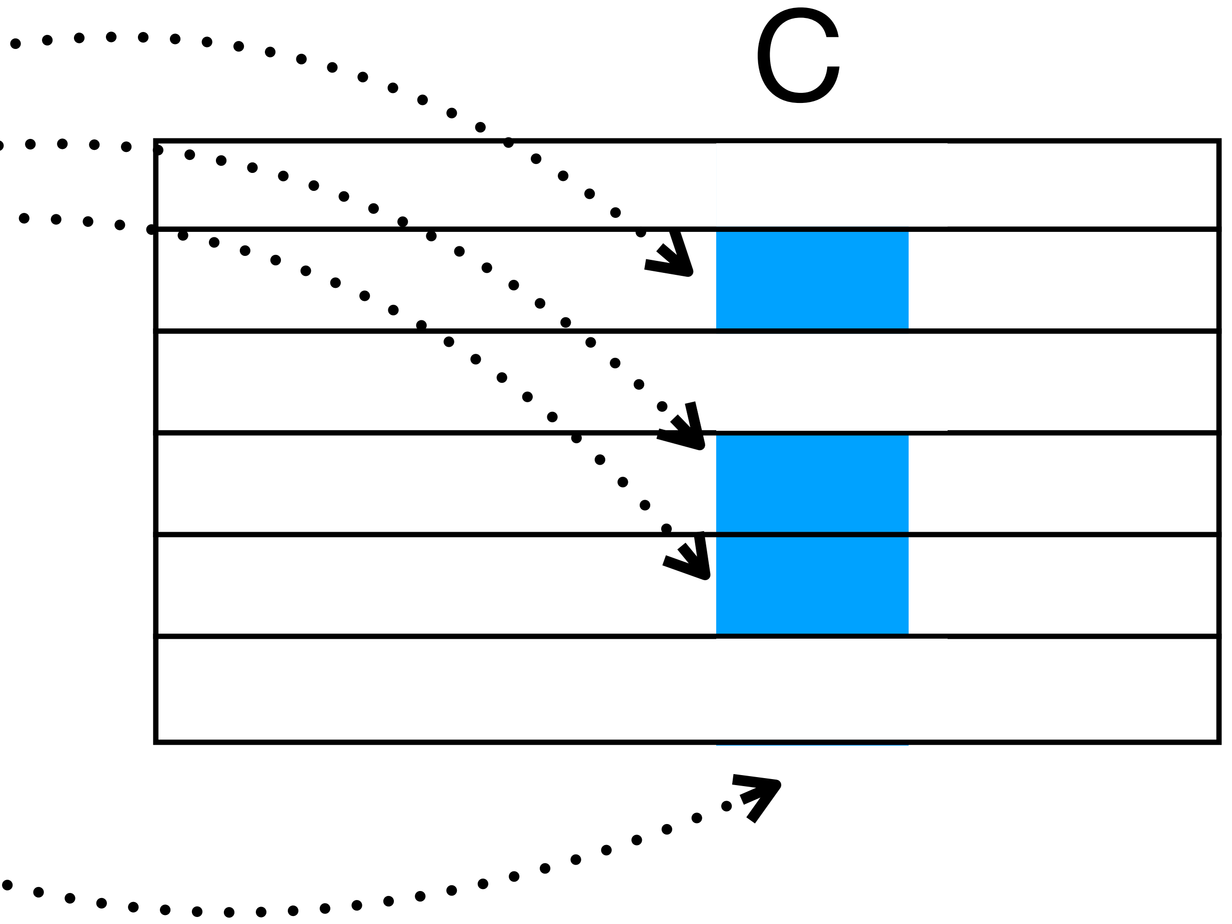
- SELECT C FROM T WHERE SomeCondition
- Verifier wants to check:
  - SomeCondition(r) = True $\iff r \in X$ (*right rows?*)
  - $\forall r \in X \quad y_r =$ C$[r]$ (*right values?*)

Idealized VDBs

Response:

$(y_r)_{r \in X}$: claimed values of C at rows in X

X: claimed list of relevant rows from C



- SELECT C FROM T WHERE SomeCondition
- Verifier wants to check:
  - SomeCondition(r) = True $\iff r \in X$ (*right rows?*)
  - $\forall r \in X \quad y_r =$ C$[r]$ (*right values?*)
  - Prover sends ‘pointers’ (handles) to sets and vectors

handle of vector v

X v

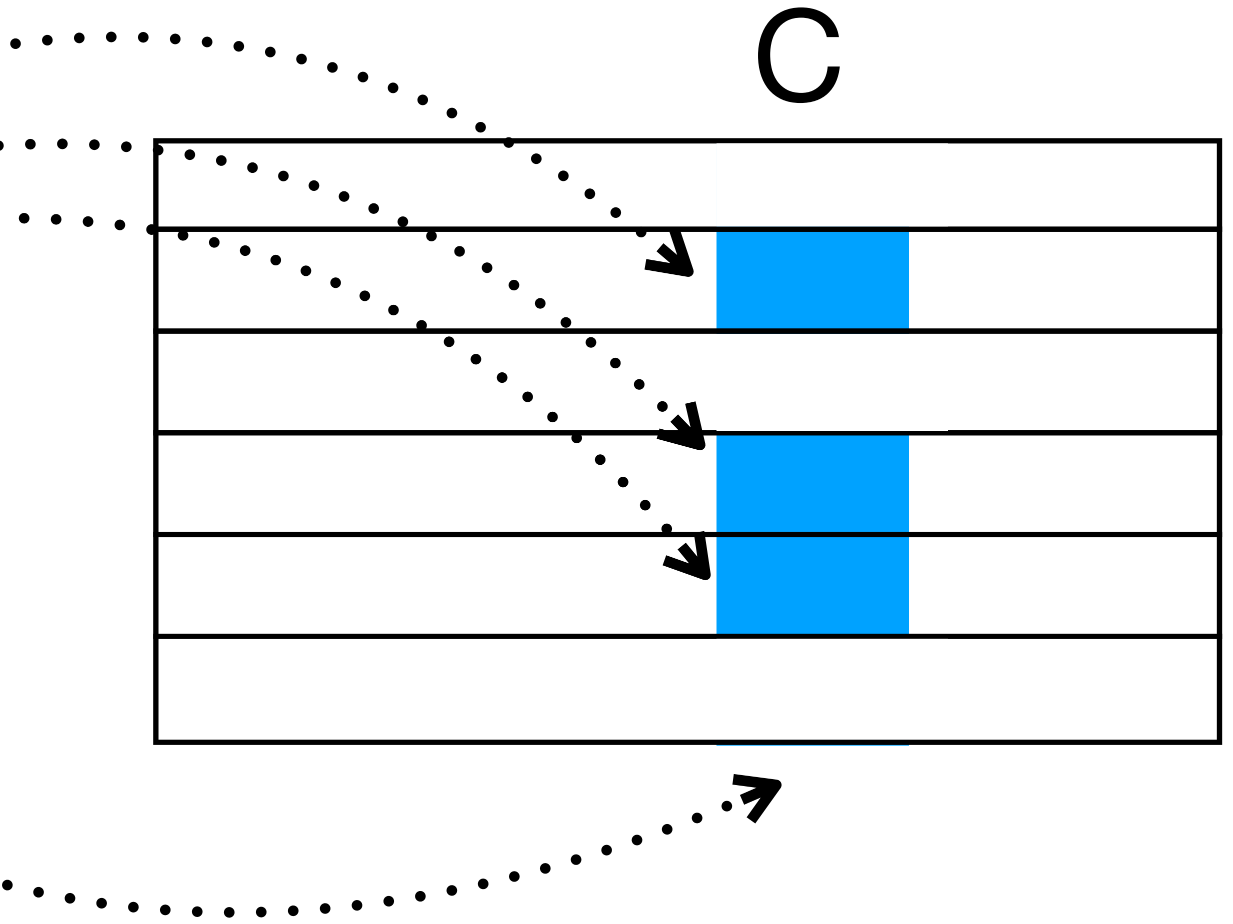
handle of set X

Idealized VDBs

Response:

$(y_r)_{r \in X}$: claimed values of C at rows in X

X: claimed list of relevant rows from C



- SELECT C FROM T WHERE SomeCondition
- Verifier wants to check:
  - SomeCondition(r) = True $\iff r \in X$ (*right rows?*)
  - $\forall r \in X \quad y_r =$ C$[r]$ (*right values?*)
  - Prover sends 'pointers' (handles) to sets and vectors
  - Verifier can perform special checks on handles

handle of vector v

X v

handle of set X

for example: $\mathbf{read}?(X, v, u)$ checks whether $v_X \stackrel{?}{=} u$

If read from v at positions in X do I get u?

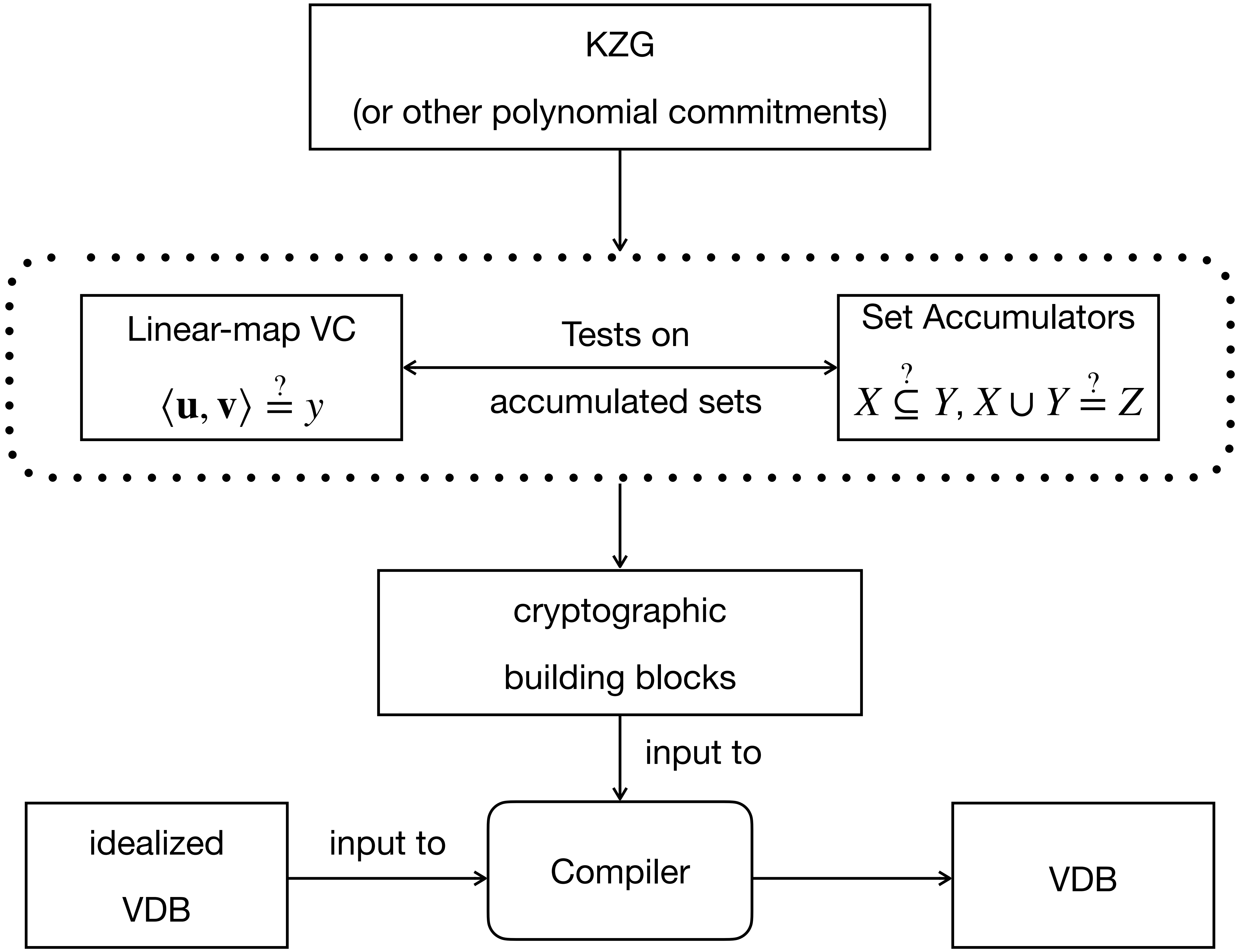
Derived Checks

Tests where two slices are equal $X_0 \stackrel{?}{=} \mathsf{eqSet}\left(\boldsymbol{u}, \boldsymbol{v}\right)$ i.e., we test: $X_0 \stackrel{?}{=} \{j : u_j = v_j\}$	Let $X_+ := \{j : u_j > v_j\}, X_- := \{j : u_j < v_j\}$. Prover sends: X_+, X_- Verifier defines $\Delta \leftarrow \boldsymbol{u} - \boldsymbol{v}$ and then checks: That X_0, X_+, X_- partition $*$ (via basic set handle tests) $\Delta\left[X_0\right] \stackrel{?}{=} 0 \quad \Delta\left[X_+\right] \stackrel{?}{>} 0 \quad -\Delta\left[X_-\right] \stackrel{?}{>} 0$
Sum check within target subset $\sum_{j \in X} v_j \stackrel{?}{=} y$	Let $\boldsymbol{u}_{\mathbb{1}} := (\mathbb{1}_X(1), \ldots, \mathbb{1}_X(m))$, $m := \lvert\boldsymbol{v}\rvert$ *(indicator vector for X)* Prover sends: $\boldsymbol{u}_{\mathbb{1}}, \bar{X} := * \setminus X$ Verifier checks: $\boldsymbol{u}_{\mathbb{1}} \stackrel{?}{=} \mathbf{1}\left[\bar{X} \to 0\right]$ *("is it the indicator vector?")* $\left\langle \boldsymbol{v}, \boldsymbol{u}_{\mathbb{1}} \right\rangle \stackrel{?}{=} y$ *(checks actual sum)* $\bar{X} \stackrel{?}{=} * \setminus X$
Pre-image check $X_\alpha \stackrel{?}{=} \alpha^{-1}(\boldsymbol{v})$ where: $\alpha^{-1}(\boldsymbol{v}) := \{j : v_j = \alpha\}, \alpha \in \mathbb{F}$	Verifier defines $\boldsymbol{u}_\alpha := \alpha \mathbf{1}$ and checks: $X_\alpha \stackrel{?}{=} \mathsf{eqSet}(\boldsymbol{u}_\alpha, \boldsymbol{v})$

"Nullifying" test $\boldsymbol{u} \stackrel{?}{=} \boldsymbol{v}\left[X_0 \to 0\right]$ i.e., $\forall j\ u_j \stackrel{?}{=} v_j \cdot (1 - \mathbb{1}_{X_0}(j))$	Prover sends: $\bar{X}_0 := * \setminus X_0$ Verifier defines $\Delta \leftarrow \boldsymbol{u} - \boldsymbol{v}$ and then checks: $\bar{X}_0 \stackrel{?}{=} * \setminus X_0$ $\boldsymbol{u}\left[X_0\right] \stackrel{?}{=} 0$ *("is $u_j = 0$ for each $j \in X_0$?")* $\Delta\left[\bar{X}_0\right] \stackrel{?}{=} 0$ *("is $u_j = v_j$ for each $j \notin X_0$?")*
Range check $\boldsymbol{v} \stackrel{?}{\in} [0, 2^\ell)$	Let $v_j^{(i)}$ denote the i-th bit of v_j, i.e., for each j, $v_j = \sum_i 2^{i-1} v_j^{(i)}$ Let $\boldsymbol{v}^{(i)} := \left(v_1^{(i)}, \ldots, v_m^{(i)}\right)$ for $i \in [\ell]$, with $m := \lvert\boldsymbol{v}\rvert$ Let $X_0^{(i)} := \{j : v_j^{(i)} = 0\}$ for $i \in [\ell]$ *(NB: $v_j^{(i)} = 1$ for all $j \notin X_0^{(i)}$)* Prover sends: $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(\ell)}$ $X_0^{(1)}, \ldots, X_0^{(\ell)}$ Verifier defines $\Delta \leftarrow \sum_i \left(2^{i-1} \boldsymbol{v}^{(i)}\right) - \boldsymbol{v}$ and then checks: $\Delta[*] \stackrel{?}{=} 0$ *(equivalent to $\sum_i \left(2^{i-1} \boldsymbol{v}^{(i)}\right) \stackrel{?}{=} \boldsymbol{v}$)* $\boldsymbol{v}^{(i)} \stackrel{?}{=} \mathbf{1}\left[X_0^{(i)} \to 0\right]$ for all $i \in [\ell]$ *("are these bits?")*
Strict sign check within target subset $\boldsymbol{v}\left[X_>\right] \stackrel{?}{>} 0$	Let $\boldsymbol{u}_{\mathbb{1}} := (\mathbb{1}_{X_>}(1), \ldots, \mathbb{1}_{X_>}(m))$, with $m := \lvert\boldsymbol{v}\rvert$ *(indicator vector for $X_>$)* Let $\boldsymbol{u}_{\text{zero}}$ be such that $u_{\text{zero},j} = \begin{cases} v_j & \text{if } j \in X_> \\ 0 & \text{if } j \notin X_> \end{cases}$ Prover sends: $\boldsymbol{u}_{\mathbb{1}}, \boldsymbol{u}_{\text{zero}}, \bar{X}_> := * \setminus X_>$ Verifier defines $\boldsymbol{v}_{\geq 0} := \boldsymbol{u}_{\text{zero}} - \boldsymbol{u}_{\mathbb{1}}$ and then checks: $\boldsymbol{u}_{\mathbb{1}} \stackrel{?}{=} \mathbf{1}\left[X_> \to 0\right]$ *("is this the indicator vector?")* $\boldsymbol{u}_{\text{zero}} \stackrel{?}{=} \boldsymbol{v}\left[\bar{X}_> \to 0\right]$ *("does this satisfy the def. of $\boldsymbol{u}_{\text{zero}}$?")* $\boldsymbol{v}_{\geq 0} \stackrel{?}{\geq} 0 \quad \bar{X}_> \stackrel{?}{=} * \setminus X_>$

From Idealized to Cryptographic VDBs

- **We commit to handles**
- X $\Rightarrow$ set accumulator
- v $\Rightarrow$ Linear-map vector commitment
- **Accumulators**:
  - VfySubset(acc_X, acc_Y, π_{subset})
- **Vector commitments**:
  - VfySubvec(cm_v, X, y, π_{subvec})



Compilation and Final Construction

Table 5: Compilation of idealized operations through cryptographic building blocks.

Idealized Operation	Cryptographic Implementation
Produce and send new slice handle $\boldsymbol{v}$	Send $\mathsf{cm}_{\boldsymbol{v}} \leftarrow \mathsf{LVC.CommitVec}(\mathsf{prk}, \boldsymbol{v})$
Produce and send new set handle X	Send $\mathsf{acc}_X \leftarrow \mathsf{SA.Accum}(\mathsf{prk}, X)$
$Z \stackrel{?}{=} X \cap Y$	Prover computes $\mathsf{SA.OpenOp}(\mathsf{prk}, X, Y, \cap) \rightarrow (Z, \pi)$. Verifier checks $\mathsf{SA.VerifyOp}(\mathsf{vrk}, \mathsf{acc}_Z, \mathsf{acc}_X, \mathsf{acc}_Y, \cap, \pi)$
$Z \stackrel{?}{=} X \cup Y$	Same as $\cap$, using $\cup$ operator
$X \stackrel{?}{\subseteq} Y$	Prover computes $\mathsf{SA.OpenOp}(\mathsf{prk}, X, Y, \subseteq) \rightarrow \pi$ Verifier checks $\mathsf{SA.VerifyOp}(\mathsf{vrk}, \mathsf{acc}_X, \mathsf{acc}_Y, \subseteq, \pi)$
$\langle \boldsymbol{u}, \boldsymbol{v} \rangle \stackrel{?}{=} y$	Prover computes $\pi \leftarrow \mathsf{LVC.OpenLin}(\mathsf{prk}, \boldsymbol{u}, \boldsymbol{v}, y)$ Verifier checks $\mathsf{LVC.VerifyLin}(\mathsf{vrk}, \mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{v}}, y, \pi)$
$\boldsymbol{u} \leftarrow \alpha \boldsymbol{v} + \boldsymbol{w}$	Uses homomorphism of LVC
$\boldsymbol{v} \leftarrow (v_1, \ldots, v_n)$	$\mathsf{LVC.CommitVec}(\mathsf{prk}, (v_1, \ldots, v_n))$
$\boldsymbol{u}[X] \stackrel{?}{=} 0$	Prover computes $\pi \leftarrow \mathsf{LVC.PrvSubvecIsZero}^*(\mathsf{prk}, \boldsymbol{u}, X)$ Verifier checks $\mathsf{LVC.VfySubvecIsZero}^*(\mathsf{vrk}, \mathsf{cm}_{\boldsymbol{u}}, \mathsf{acc}_X, \pi)$
$\mathsf{data} \leftarrow \mathbf{read}(X, \boldsymbol{v})$	Prover sends X, $\pi \leftarrow \mathsf{LVC.OpenSub}(\mathsf{prk}, C, X, \mathsf{data})$ Verifier checks $\mathsf{LVC.VerifySub}(\mathsf{vrk}, C, X, \mathsf{data}, \pi)$ $\mathsf{acc}_X = \mathsf{SA.Accum}(\mathsf{prk}, X)$

Join queries.

Consider tables T_1, T_2 with respective columns named pk, col_1 and fk, col_2. As their names suggest pk is primary key of tab
referencing values from pk. Consider the que

$$\text{Q5}: \quad \mathsf{SELECT} * \mathsf{FROM}\ T_1\ \mathsf{J}$$

Pre-processing: as before.
Proof computation: the Prover performs the

- retrieves the set handle fk referring th each $v \in V_{pk}$.
- retrieves the set handle pk referring th each $v \in V_{fk}$.

The Prover sends pk, fk to the Ver

Proof verification: The Verifier performs the

- compute $\widehat{pk} \leftarrow \mathbf{read}(pk, pk)$
- compute $\widehat{fk} \leftarrow \mathbf{read}(fk, fk)$
- check that $\widehat{fk} = \widehat{pk}$
- $\widehat{rst_1} \leftarrow \mathbf{read}(pk, T_1.rst)$
- compute $\widehat{rst_2} \leftarrow \mathbf{read}(fk, T_2.rst)$

Subsequently, the Verifier concatenates the
To prove that the query result contains all t
engage in a protocol similar to the second pa
To join two tables T and T' on equality o
tables, we do the following:

Invariant (initially enforced throug

- For each table T and column C we kee column

Observation: let $V_\cap := V(T, C) \cap V(T$
be given by the cross product of the rows fro

$$\alpha_i^{-1}(T.C) \times \alpha_i^{-1}(T'.C)$$

$$\text{Q7}: \quad \mathsf{SELECT\ SUM}(\mathsf{col}_{tgt})\ \mathsf{FROM}\ T \tag{8}$$

Pre-processing: as before.
Proof computation:

- the Prover computes the set handle $*$ corresponding to the rows of T and the value $s_{tgt} = \sum_{v \in \mathsf{col}_{tgt}} v$
- The Prover sends $*$ and s_{tgt} to the Verifier

Proof Verification:

- the Verifier gets $*$ and s_{tgt} from the Prover
- the Verifier checks that $\sum_{*} tgt$ is equal to s_{tgt}

Completeness follows from the correctness of the sum check within target subset operation, indeed the prover is sending $*$ together with $\boldsymbol{u}_1$, I.e., the vector that contains all ones, the verifier checks that $\boldsymbol{u}_1$ is actually one in all positions and then performs the inner product between col_{tgt} and $\boldsymbol{u}_1$ checking if it is equal to the response. To prove soundness let us assume that there exists an adversarial prover that will cause the verifier to return 1 but such that $\mathsf{SatisfiesQry}(\mathsf{db}, \mathsf{qry}, \mathsf{resp}) = \mathsf{false}$. Therefore resp does not contain the sum of the elements in col_{tgt}. The probability that it happens is negligible indeed the verifier can check that $\boldsymbol{u}_1$ is a vector of all ones, that $*$ is indeed a set handle to all indices and that the the inner product of the two handles is actually the expected sum.

- COUNT query: consider the query:

$$\text{Q8}: \quad \mathsf{SELECT\ COUNT}(col_{tgt})\ \mathsf{FROM}\ T \tag{9}$$

Pre-processing: as before.
Proof computation:

- the Prover computes the set handle $*$ referring to all rows of T
- the Prover sends $*$ and the value n to the Verifier

Proof Verification: The Verifier performs the following steps:

- get $*$ and the value n from the Prover,

Zooming in on Efficiency

Q_{Tot}

```
SELECT SUM(price) FROM Transaction
WHERE account_id = '5938' AND trade_date = '2025-01-01'
```

↰ Computes total price of transactions executed by an account on a given date

Q_{CntTx}

```
SELECT COUNT(*) FROM Transaction
WHERE trade_date BETWEEN '2025-01-01' AND '2025-03-31'
```

↰ Computes the number of transactions executed within the first quarter

Q_{MatchExp}

```
SELECT tx_id, price, expected_price, price = expected_price
FROM Transaction WHERE trade_date = '2025-04-05'
```

↰ Retrieves the transactions whose executed price equals their expected price

Query	Prover Time	Verifier Time	Proof Size
Q_{Tot}	1.21 s	13.00 ms	0.66 KB
Q_{CntTx}	15.59 s	21.81 ms	5.13 KB
Q_{MatchExp}	6.15 s	25.17 ms	0.98 KB

Table 2: Experimental evaluation over queries from Fig. 2 on a DB with 100K rows.

Scheme	Overhead in proof size, Verifier time w/o JOINs	Overhead in proof size, Verifier time w/ JOINs	Preprocessing and server storage
IntegriDB	log(|cols|)	|response| * log|cols|	|db| * |cols|²
vSQL	polylog|db|	polylog|db|	|db|
Qedb	|query|	|response|	|db|

typically, |resp| << |cols| << |db|

Zooming in on Simplicity

- General proof systems

Circuits for SQL

Circuits for STARK recursion

Groth16

FRI and STARK

Parallelizing computations & Recursion tree logic

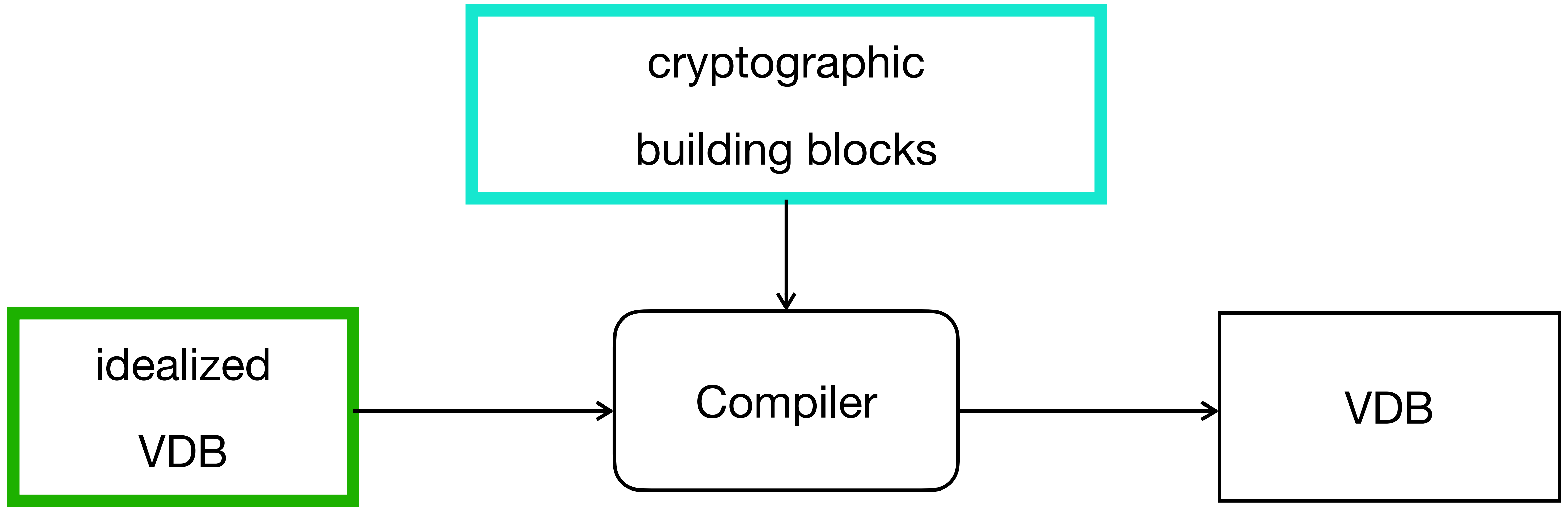
VM

- Qedb

Protocol checks and their Composition

KZG

also: Modularity



Wrapping Up

- **Simplicity** is important both for real-world security and research progress
- Research on VDBs from authenticated data structures has been stagnant for almost ten years
- **qedb** is a new DB aiming at being:
  - **performant**
  - **simple** and **modular**
  - **Future work**:
    - Beyond SQL
    - Zero-Knowledge
    - Lookup singularity for VDBs?
    - Formally verified implementation?

https://eprint.iacr.org/2025/1408

alberto.trombetta@uninsubria.it alberto@provably.ai