# Space and Time (SXT Chain)

**The blockchain for ZK-proven data.**

Scott Dykstra Jay White, PhD Catherine Daly Ian Joiner, PhD

May 2025
v2.0
**whitepaper**

## Abstract

Smart contracts lack native data processing capabilities, and securely connecting external data is cumbersome. Space and Time (SXT Chain) addresses these challenges by introducing *Proof of SQL*, a breakthrough zero-knowledge (ZK) circuit that provides verifiable SQL queries against tamperproof tables. Delivered as a decentralized database loaded with ready-to-go indexed blockchain data compatible with Proof of SQL, SXT Chain is ushering in a new era of data-rich smart contracts.

By storing only cryptographic commitments onchain, SXT Chain secures an unlimited number of tables/ledgers without bloating the blockchain. Developers gain the ability to run ZK-verified queries on massive datasets from major blockchains or user-provided offchain data, all while supplementing the limited compute power of smart contracts. This powerful combination propels the creation of sophisticated, cross-chain dapps—ultimately fulfilling the blockchain vision of a permissionless, trustless, and data-rich digital economy.

# Table of Contents

# 1 The Problem: Untapped Potential of Web3

## 1.1 Smart Contracts Can’t Access Critical Data

Smart contracts underpin major decentralized finance (DeFi) protocols like Uniswap, Aave, Compound, and Synthetix, each securing billions on the Ethereum Virtual Machine (EVM). While these protocols provide trustless, permissionless asset management, they lack data-driven sophistication—Aave, for instance, can’t differentiate a new borrower from one with established onchain history.

In traditional finance (TradFi), options derivatives exceed their underlying assets’ capital, with over 40 million contracts traded daily. Yet these options are largely missing onchain because trustless deployment requires external data processing, which standard smart contracts cannot perform. By contrast, onchain perpetual futures see an annual volume of $39 trillion, thanks to simpler logic that relies only on token price feeds.

Smart contracts can only access wallet balances, current transaction data, basic blockchain metadata, and their own internal state. They cannot directly query the historical data stored by blockchain nodes. Using archive nodes, an external offchain client can only access simple datapoints such as:

- The state of the entire blockchain at any given time in history (e.g. who is the first owner of the “Cryptopunk #1” NFT).
- The transactions—and events emitted as a result of transactions—at any given time in history (e.g. John’s wallet swapped 1000 USDC to 0.5ETH on 4/18/2023).

Offchain indexing solutions can parse this history but introduce trust assumptions; such solutions are typically centralized and unverified. Decentralized oracle networks bridged this gap by providing the infrastructure to aggregate, validate, and deliver offchain data to blockchains, but require connectivity to external systems to handle complex datasets, SQL execution, rich onchain history, etc.

As multichain protocols spread, many rely on a “hub” contract on Ethereum linked to “spoke” contracts on other chains. Monitoring spoke activity in a trustless way remains difficult. Similarly, Web3 games struggle to reward in-game actions or onchain NFT milestones because the underlying data isn’t directly accessible to the smart contract. Some teams resort to centralized services, undermining the trustless ethos of Web3 and highlighting an urgent need for trustless data processing.

## 1.2 Current ZK Solutions and Their Limitations

ZK proofs have been heralded as the panacea for trustless data access, offering client verification by allowing data to be confirmed without revealing its contents to the client. By offloading computation offchain and verifying the results onchain, ZK proofs can bolster the limited data access and compute power of smart contracts. They eliminate the need for cumbersome offchain consensus by letting a single node act as a coprocessor to an L1/L2 chain. However, current ZK solutions are still maturing and face several key challenges.

First, ZK proofs lack scalability when handling large-scale data. Proof generation often takes minutes, making ZK proofs impractical for complex queries against, for example, multi-year transaction histories. Second, the ecosystem is highly fragmented, with some projects aimed at privacy, others at scaling, and still others at specific use cases like rollups. Tying these disparate solutions together is time-consuming, costly, and prone to error. Third, the developer experience remains unfamiliar and demanding, often requiring new languages (e.g., Cairo) and specialized hardware deployments (e.g., zkEVM rollups). Finally, existing ZK solutions do not address the missing query layer in Web3, leaving decentralized applications without a truly trustless way to process and analyze large datasets.

## 1.3 The Need for an "Onchain Database"

In traditional SaaS applications, business logic follows a straightforward cycle of querying data, executing actions based on the query results, and updating system state. This pattern is evident across industries: lenders check a borrower's creditworthiness, decide terms, and update loan states; e-commerce platforms query product availability, manage purchases, and adjust inventories; travel services query accommodations and flights before finalizing bookings; and social networks query a user's connections to display and rank content. However, in Web3, while blockchains and smart contracts provide state management and basic logic, they lack a dedicated query layer which limits data processing and onchain computation for more advanced decentralized applications. Space and Time addresses this gap by serving as a "query coprocessor," delivering robust offchain data processing alongside major blockchains.

Over the past 15 years, tech conglomerates have accumulated and monetized user data within proprietary cloud warehouses. This centralized control conflicts with the Web3 vision of a decentralized internet, where individuals manage their own information, maintain privacy via wallet-based identities, and participate in transparent governance. A community-operated database model can help reclaim data sovereignty, as end users collectively determine how data is accessed, used, and monetized. Through wallet-based public key infrastructure (PKI), individuals can sign off on how their interactions are recorded, shared, or deleted.

Yet, existing blockchains primarily function as single-ledger databases with no native query engine, resulting in state bloat and limited data capabilities. Achieving the data flexibility required by modern decentralized apps calls for a more expansive approach—one that supports near-infinite tables/ledgers, each dedicated to a specific crypto app domain, while preserving the trustless security model of blockchains.

# 2 The Solution: SXT Chain with Proof of SQL

## 2.1 Coprocessing Unlocked for Smart Contracts

SXT Chain addresses the need for a query coprocessor for smart contracts by providing a tamperproof, community-operated decentralized database that indexes and verifies data from major blockchains or user-owned tables using ZK proofs. Community-operated validators store threshold-signed cryptographic commitments (one per table) instead of raw data. Validators and archive nodes locally pin raw data or act as comprehensive repositories to ensure redundancy and availability. Indexed data from major blockchains (e.g., Ethereum, Bitcoin, ZKsync, Sui, Avalanche, Base) is validated and summarized into onchain cryptographic commitments (e.g., KZG, Nova, Dory), which act as tamperproof digital fingerprints of the underlying tables. When data insertions are submitted via DDL/DML transactions, the network updates the affected table's cryptographic commitment to reflect its new state.

Proof of SQL is a groundbreaking ZK proof protocol that ensures the accuracy of SQL query results and the integrity of underlying data tables stored on SXT Chain using cryptographic commitments specifically to ensure underlying data tables are tamperproof as part of the mechanism (described in a later section of this paper). The network's *Prover* nodes, running the Proof of SQL protocol, handle both query processing and proof generation. The protocol targets online latencies while proving computations over entire chain histories an order of magnitude faster than state-of-the art zkVMs and coprocessors. SQL proving times are currently benchmarked at under a second for analytic queries on million-row tables using a single NVIDIA GPU.

## 2.2 Query Examples for ZK-proven Onchain Data

Ultimately, Space and Time allows smart contracts to “ask ZK-proven questions” regarding activity on their own chain, other chains, or offchain. For example, a smart contract can request the SQL equivalent of:

- **Loans paid off:** "Show me the volume of loans already paid off by this wallet attempting a borrow transaction" (in order to offer a discounted loan rate to a loyal borrower who has paid off loans onchain already).
- **Liquidity Pools TVL:** "Show me all liquidity pools with a TVL greater than $1M that were deployed at least one month ago."
- **Liquidity Pools Collateral:** "Sum up the total collateral available right now in all liquidity pools our protocol operates across the following four chains…"
- **DEX Loyalty Program:** "Given that wallet 'xyz' is currently attempting a trade, show me the number of prior trades this wallet has already made with DEX liquidity pool address 'xyz'" (in order for the DEX contract can apply a discount or reward to the current trade).
- **LP position management (active/dynamic):** "Alert me when the delta between the average price of the LINK and ETH has deviated more than 10%" (for Uniswap v4 LP rebalancing via hooks).
- **Avg. Lending Rates Onchain:** "Show me the volume-weighted average lending rate for USDC on Aave, Maker, and Compound right now."
- **Wallets with Token Balance Criteria:** "Show me all wallets that have a balance > $1000 of LINK tokens and have interacted with at least one Chainlink smart contract."
- **Bridge Transactions:** "Show me all transactions across 'xyz' bridges that moved at least $1M of tokens per wallet from Chain A to Chain B."
- **Gamer Achievements:** "Show me all gamer wallets that have at least 2 hours of playtime in-game, have minted our NFT, and played with 'xyz' weapon."
- **CeFi Options Markets:** "Give me the at-the-money implied volatility of Bitcoin call options expiring end-of-year averaged (volume-weighted) across Deribit and Binance."
- **Gas Oracles:** "Show me the average gas used across all transactions on 'xyz' bridge over the last hour."
- **Token Price Oracles:** "Show me the volume-weighted average price of LINK and wrapped LINK tokens traded across the hundreds of liquidity pools that swap LINK: Uniswap, PancakeSwap, Sushiswap, Trader Joe, Quickswap, etc."
- **Trustless ETFs/Indexes:** "Create a volume-weighted index of the following 10 tokens calculated from price oracle SQL calculations and deliver it to my contract as a single price index."
- **Airdrop Criteria:** "Roll up the wallet transaction histories of all wallets that meet the following criteria for my airdrop…"
- **Governance Engagement Score:** "Roll up engagement scores for wallet activity that meets the following criteria… where volume transacted in 'xyz' token, onchain governance voting participation, and past NFT ownership will contribute to rewards/discounts associated with a wallet engagement score."

- **Governance Qualification:** “Find all wallets that have participated in governance votes and rank them with a cross-section of the amount of ‘xyz’ token held and how long it’s been held.”
- **Weather (Onchain Insurance Payouts):** “Show me the average wind speeds across all reported weather stations in Miami, Florida today.”
- **Github Activity (Open Source Contributions):** “Rank developer wallets by their total code contributions through ingested GitHub activity data.”
- **Oracle Activity:** “Show me total volume of oracle requests over the last hour that did not complete the request within two blocks.”
- **Pricing Blockspace:** “Determine blockspace ‘market rates’ by averaging transaction gas fees along with block header metadata.”
- **NFT Floor Prices:** “Look at NFT trading information onchain and find the current average floor price for collection ‘xyz.’”
- **Perp Funding Rates:** “Calculate the implied volatility of both ETH and BTC using onchain perpetual futures funding rates as a leading indicator.”
- **Ethereum Staking Yields:** “Show me the risk-minimized average returns on Ethereum liquid staking via Lido over the past hour.”

## 2.3 Design Goals

The Space and Time solution was designed with the following goals around data processing:

1. **End-to-End Trustless**: Leverage a ZK approach for proving rather than building yet another consensus-based, ‘trust-minimized’ approach with 12 to 30 nodes. This removes reliance on a manually configured set of permissioned nodes while offering improved scalability.

2. **Verifiable Onchain:** Ensure ZK proofs generated offchain are verifiable within the computational capabilities of the EVM, rather than only verifying offchain using a third-party service such as an oracle or relayer. Thus, a smart contract on any major chain (consensus-based or ZK-rollup chains) can verify the proof and relay the verified query result to a client contract which made the request.

3. **Familiar, Easy Developer Experience**: Provide a UX common in Web2 (SQL) so developers don’t have to learn any extensive new frameworks or languages.

4. **Support for Arbitrary Computations**: Facilitate common data processing jobs that require aggregations, sorts, filters, arithmetic, joins, etc., as well as Turing-complete arbitrary computations using SQL.

5. **Low-Latency at 100 Gigabyte-Scale**: Deliver verified query results back to a client smart contract within seconds—not minutes—using a more scalable approach than

redundant computations proven with consensus. Must support queries over the entire chain state (often multiple terabytes of data per chain since genesis, broken up into sub-tables that often exceed 100 gigabytes each).

6. **Comprehensive Chain Data Coverage**: Persist entire copies of each major chain (including all events, transactions, blocks, logs, balances, token price changes, etc.) to facilitate cross-chain queries against current and historical activity.

Given the limitations of arbitrary ZK proof circuits and the potential strengths of a circuit constructed around a SQL parser, we considered the following:

1. **Optimized Circuit Design:** We narrowed our initial focus to only SQL operations in order to create highly optimized ZK circuits tailored for data processing. This level of specificity allows for streamlined verification and reduced computational overhead.

2. **Broad Applicability:** SQL is ubiquitous—a widely understood and universally accepted language for data operations that operates at the heart of both complex financial systems and basic web applications. The basic constructs of SQL (selections, projections, joins, and aggregations) cover most of the processing requirements for large datasets.

3. **Relational Data Models:** Blockchain indexing necessitates the decomposing of the blockchain ledger into multiple tables (wallets, blocks, transactions, smart contract logs/events, price feeds, etc.) in a relational data model. For example, the “wallets” table can be joined with the “blocks” table, which can be joined with the “transactions” table. SQL is suitable for accessing structured, relational data.

## 2.4 Components of the Network

Space and Time is a three-layer decentralized network of user-operated nodes serving distinct purposes:

### 2.4.1 Indexer Nodes

*Indexer* nodes are lightweight servers designed to redundantly fetch and process data from popular blockchains such as Ethereum, Polygon, and Avalanche. These nodes:

- Continuously extract the latest block data, transactions, balance changes, smart contract event logs, storage slot updates, and more.
- Transform raw blockchain data into a relational SQL-compatible format, effectively performing Extract, Transform, Load (ETL) operations.
- Submit the transformed data as insert-data transactions to SXT Chain, ensuring the data is securely incorporated into tamperproof tables.

- Can be deployed as a restaking-secured service using SXT Chain queries for rewarding or slashing *Indexer* nodes based on the accuracy of the source chain data they deliver.

### 2.4.2 Prover Nodes

*Prover* nodes handle client requests for ZK-proven SQL execution against tamperproof tables secured by SXT Chain (via a threshold-signed commitment for each table). Their primary responsibilities include:

- Receiving client queries, often from smart contracts on EVM-compatible chains such as Ethereum.
- Executing ZK-proven SQL queries against indexed data or other secured tables.
- Generating cryptographic proofs to verify both the query's computational accuracy and the integrity of the underlying data.

### 2.4.3 SXT Chain Validators

Validator nodes are the backbone of SXT Chain, responsible for maintaining the integrity of the network. They:

- Update cryptographic commitments stored onchain whenever insert-data transactions are processed, ensuring that the tamperproof tables accurately reflect the latest data state.
- Collaborate via Byzantine Fault Tolerance (BFT) consensus to threshold-sign new commitments, providing a decentralized and secure mechanism for data verification.

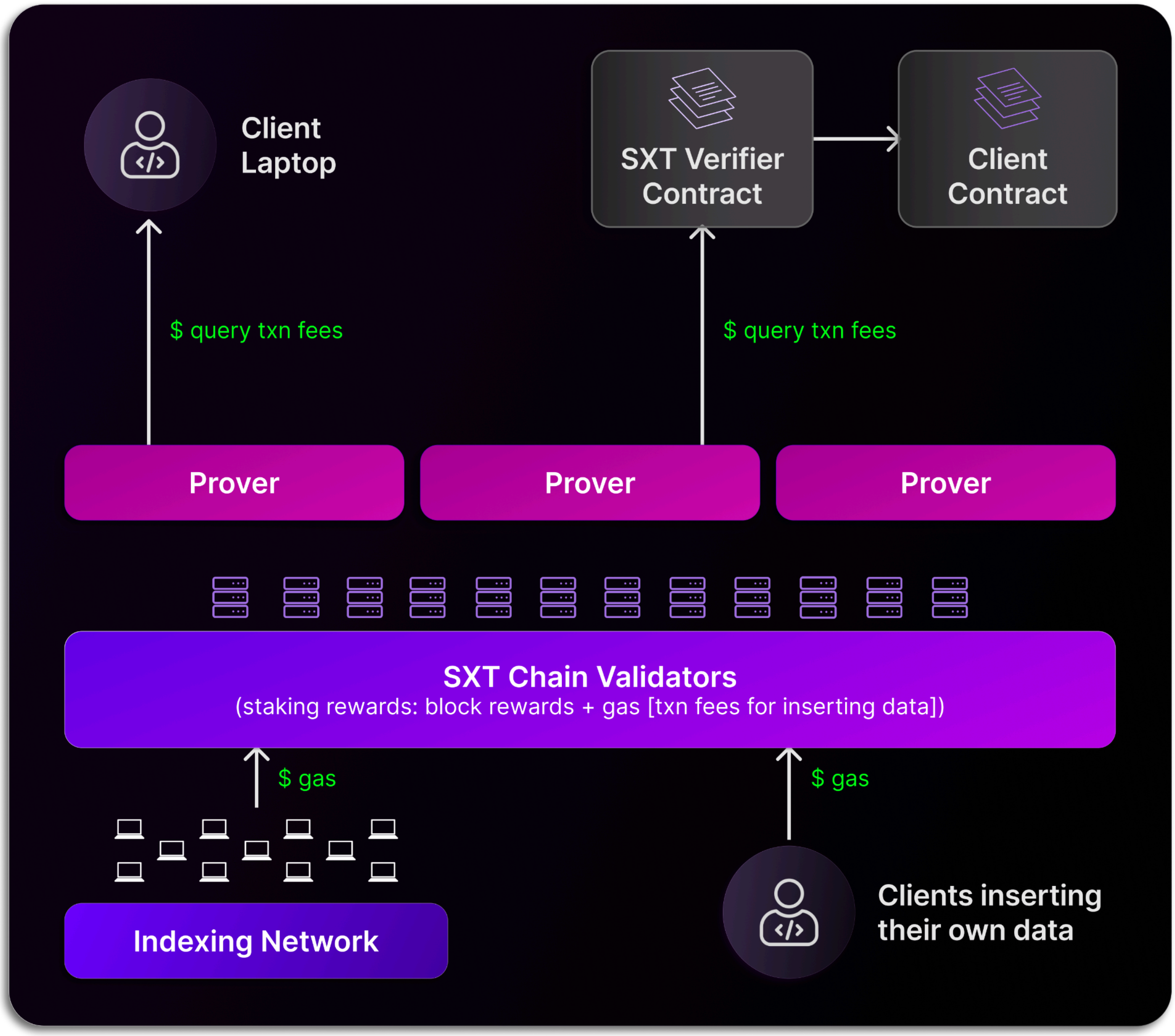

*Figure 0: Space and Time Primary Components*

### 2.4.4 Target Chain and Offchain Components

Space and Time is seamlessly integrated with major blockchains, delivering ZK-proven query results against onchain and offchain data to smart contracts. Below is an overview of how each step is handled:

- **Client Request (Onchain):** The client contract sends a payment along with query specification to the Space and Time *Verifier* contract, which then emits an event onchain requesting query/proof fulfillment (data processing jobs for indexed or offchain data).
- **Proof Generation (Offchain):** A single Space and Time *Prover* node (GPU) within the decentralized network listens for requests emitted onchain, then executes Proof of SQL

by generating a ZK proof and the associated query result for the request. Both the result and the proof are submitted to a client for verification.

- **Proof Verification (Onchain):** The Space and Time *Verifier* contract accepts the query result and proof from the *Prover* nodes. It verifies the proof and hands the verified query result to the client that requested fulfillment.

Space and Time aims to address the pervasive problem of fragmentation in Web3 by developing a comprehensive solution that combines indexed data, proof generation, and verification, and the efficient relaying of data to smart contracts.

## 2.5 Ingestion Layer with Verifiable Indexing

Data ingestion into Space and Time (SXT Chain) leverages BFT consensus across the *Validator* set, to cryptographically verify that indexed data from popular chains is accurate to that source chain. This can also be useful for customer-owned tables as well. For example, a client can define a tamperproof table in Space and Time that persists stock prices from traditional equity markets, with external business logic that captures real-time pricing information from 12 different stock market data APIs simultaneously. Space and Time *Validator nodes* can achieve consensus on the price of Apple stock across these 12 different data providers, for example, and only insert into SXT Chain the threshold-signed, consensus-approved price of Apple stock at the current moment.

Similarly, with indexed blockchain data that the network captures, Space and Time leverages a decentralized *Indexer* node set to redundantly process the entire state of popular chains like Ethereum, Bitcoin, Polygon, ZKsync, etc. Each *Indexer* runs a *light client* for the source chain, an RPC service internally to get the latest block/transactions/events, decodes and transforms the data into a relational database format, and finally builds cryptographic commitments on recently indexed blocks. *Indexers* submit their transformed chain data to SXT Chain for consensus approval by the *Validators*. Through this process, Space and Time is commoditizing indexed blockchain data, and offering it at low cost through the network (SXT Chain only charges for compute costs of queries executed in Space and Time, not the underlying data storage).

## 2.6 Decentralized Database Design

The decentralized database serves as an integral resource for smart contracts to efficiently offload computation and data storage. The solution enhances chain scalability, enables faster contract execution, and reduces the amount of gas spent onchain while allowing the entire stack to remain permissionless to operate. A growing number of developers are even starting to build ZK-rollups leveraging Proof of SQL as the L2 or L3 ledger that's settled on a main chain periodically, via a rollup contract on that main chain querying the ledger in Space and Time for updated account balances batched in a single transaction. This enables:

- **Decentralization with Self-Custody:** Data is stored and processed across a network of community-owned and operated nodes. Role-based and row-based access to each table/ledger in the database is governed by decentralized mechanisms using "biscuits,"[3] a budding approach to encoding user secrets and sharing permissions around CRUD operations to a table/ledger. This also facilitates self-custody of data in Space and Time, where an end-user can write directly to "public write-permissioned" tables/ledgers without intermediaries, and subsequently remove content or govern how other dapps that read from these tables/ledgers can access content.

- **Offloading Compute and Storage:** To maintain the efficiency and speed of onchain operations, smart contracts can offload to Space and Time any computationally intensive processes, account balance management, or voluminous storage requirements.

- **Transparent data processing vs. ZK for privacy:** Varying business requirements across a wide range of use cases require that some dapp developers transparently publish all data as "public read" tables/ledgers to their individual communities, while others must hide data processing to protect sensitive information. For example, a protocol that whitelists wallet addresses and calculates risk scores (for compliance reasons, OFAC protection, fraud prevention, etc.) may want to transparently publish their whitelist as a "public read-permissioned" table in Space and Time, as well as the actual model used to define risk scores associated with each wallet. On the other hand, a protocol that enables undercollateralized onchain lending may want to provide query results to smart contracts around real-world credit scores without actually revealing those credit scores to the public. Proof of SQL allows a smart contract to ask the SQL equivalent of "Is this user's credit score greater than 600?" and prove that the 'yes/no' response from Space and Time is accurate without revealing the actual credit score.

- **Network Effects and Scalability:** As the network integrates more chains and more participants join the network, this not only increases protocol security but also the public datasets available (as well as third-party ecosystem integrations). This is traditionally known as "data gravity" and is a lucrative focus for traditional cloud database vendors.

## 2.7 Infinite Tables with Onchain Verification

Space and Time capitalizes on the moats of both data warehousing and onchain protocol security to generate powerful network effects. By marrying the trustless properties of blockchain technology with the scale and efficiency of a data warehouse, Proof of SQL drives a new market standard of verifiable data processing at scale across both Web2 and Web3.

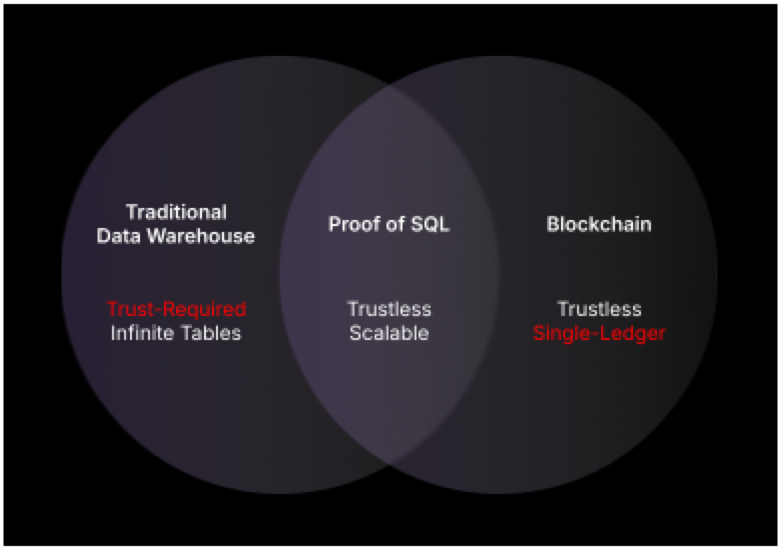

*Figure 2: Marrying Scale and Provability with Proof of SQL*

On one hand, smart contracts can transact with Space and Time's *Verifier* contracts onchain, incurring fees for data retrieval and verification. Simultaneously, enterprises can access Space and Time's vast blockchain data (or their own customer-loaded tables), sometimes paying in fiat, which is then used to enhance the native token's utility for:

- **Offchain ZK-proven Queries on Analytic Tables:** Direct access to Space and Time database nodes offchain can be achieved with REST APIs and RPC connection proxies deployed on or near the *Validator* nodes, for both tamperproof queries secured cryptographically by Proof of SQL as well as unverified queries secured optimistically by token-economic security.

- **Cross-Chain Interoperability:** Smart contracts on "Chain B" can query current or historical data (entire chain state) from "Chain A", with arbitrary SQL business logic deployed en route.

- **Joining Onchain and Offchain Data:** Developers often write complex queries that join offchain data (such as TradFi market data, in-game activity, or traditional data lake reads for Web2 business processes) with onchain data (such as token swaps or perps, NFT activity, or blockchain rewards/metadata) while still retaining ZK-proven computation of query results if needed.

*Here is how onchain verification works, on any popular EVM chain:* An SXT Chain query relayer contract is deployed to the EVM chain, which acts as an onchain endpoint for client smart contracts to interact with. The client contract sends a tiny payment along with a query ID, and the SXT Chain relayer emits an event that SXT Chain nodes listen for. A *Prover* node is chosen to execute the query and build the associated proof, and acts as a ZK-oracle to return the query result and proof back onchain. In doing so, the *Prover* builds an EVM transaction and pays the EVM verification gas for the SXT Chain relayer contract to ZK-verify the provided query result before returning the result to the client contract via a callback. This entire process often happens within a block of the original client contract request.

# 3 Design of the Proof of SQL Protocol

Within the Space and Time network, **Proof of SQL** is a novel cryptographic data-processing protocol that allows verifiable outsourced SQL execution using ZK proofs. This enables Space and Time to extend the security of Ethereum and other major L1s, L2s, or L3s/appchains to SQL databases. The protocol cryptographically guarantees to the client both that the underlying requested data (in many cases, indexed blockchain data) is tamperproof, and that the computational steps of the query request have been executed accurately. Proof of SQL eliminates the inefficiencies of consensus-driven data processing and offers practical, low-latency proof generation from a single *Prover* node (a single GPU) at a scale sufficient for enterprise-grade applications.

The protocol was designed with several goals in mind. First, the protocol supports fast end-to-end queries. Second, the verification of the queries is cheap enough to run in constrained environments like EVM smart contract functions. Third, we have built it to be extremely developer-friendly: SQL is the most popular data query language and provides a familiar UX for anyone starting to build a data-forward application. Finally, this protocol facilitates complex data processing rather than simply running serial compute or blob data retrieval.

## 3.1 Overall Architecture

In this section, we describe the general design and architecture behind the Proof of SQL protocol. The following sections go into more details about how the proof is constructed. To summarize the following sections: Section 3.1 gives an overview of the basic data flow; Section 3.2 describes the parsing of the query; Section 3.3 describes the query execution and proof setup; Section 3.4 introduces some mathematical notation that is used in later sections; Section 3.5 gives some examples of the protocol described in Section 3.3; Section 3.6 describes the sum-check protocol, which is a key primitive of the proof; Section 3.7 describes the commitment scheme.

### 3.1.1 Data Ingestion

There are two main types of interactions as part of the Proof of SQL protocol. The first type of interaction is data ingestion and the second is a query request. Both of these could be initiated by a smart contract, a decentralized application, or any party interested in producing or consuming data. In practice, the actual data source does not need to be the verifier, but can instead be a trusted data source. For example, the data could come from a trusted central party, from a trustless protocol such as the Space and Time *Indexer* nodes, or, as previously noted, from the *Verifiers* themselves. For the sake of this architecture discussion, we will separate the role of the *Verifier* from the role of the *Validators*, since this is the design of the Space and Time network. Other architectures could exist that leverage Proof of SQL and combine these roles. During data ingestion, the data source wishes to send data to the *Prover* so that the *Prover* can later query that data. However, to ensure that the data will not be tampered with, the *Validators* need to compute commitments of this data. The *Verifier* can then access these commitments, while the *Prover* holds onto the actual data. Without these commitments, the *Prover* would be able to modify the data at will, or return incorrect execution against the data.

When a service or a client sends data that is to be added to the database, that data is routed to the *Validators* so they can create a commitment to that data. This commitment is a small "digest" of the data but holds enough information for the rest of the protocol to ensure that the data is not tampered with. After creating this commitment, the *Validators* route the data to the database for storage. The *Validators* store this commitment for later usage.

An important design constraint is that the commitment must be *updatable* (see section 4.7). In other words, suppose that the *Validators* already hold the commitment to a specific table, but new data is to be ingested and appended to the table. To do this efficiently, the *Validators* must be able to combine the old commitment with the incoming data to create a new commitment to the entire updated table. The key constraint here is that the *Validators* must be able to do this without access to the old existing data.

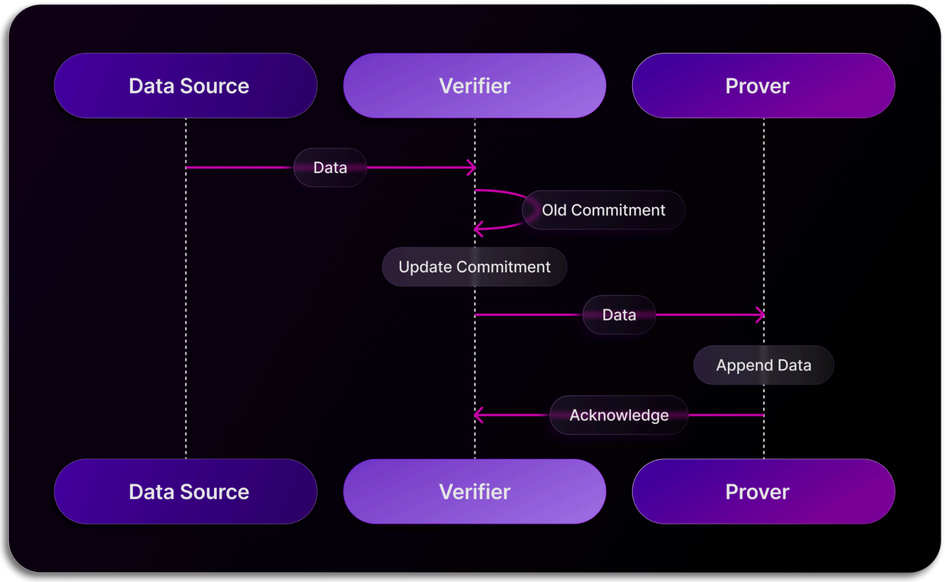

*Figure 3: Sequence Diagram of Data Ingestion*

### 3.1.2 Query Request

The second type of interaction is a query request between the *Verifier* and the *Prover*. For example, the *Verifier* wants to have some data analytics executed on the data that the *Prover* is holding and for the result to be returned to the *Verifier*. The *Verifier* is able to trust this result because of the commitment to the underlying data.

When a service, client, or the *Verifier* themselves send a query request, that request is routed to the *Prover*. At this point, the *Prover* parses the query, computes the correct result, and produces a proof. It then sends the query result and the proof to the *Verifier*. Once the *Verifier* has this proof, it can use the commitment to check the proof against the result and verify that the *Prover* has produced the correct result to the query request. The majority of this section (3) is dedicated to describing how this proof is constructed and verified. See the next subsection (3.1.3) for an overview.

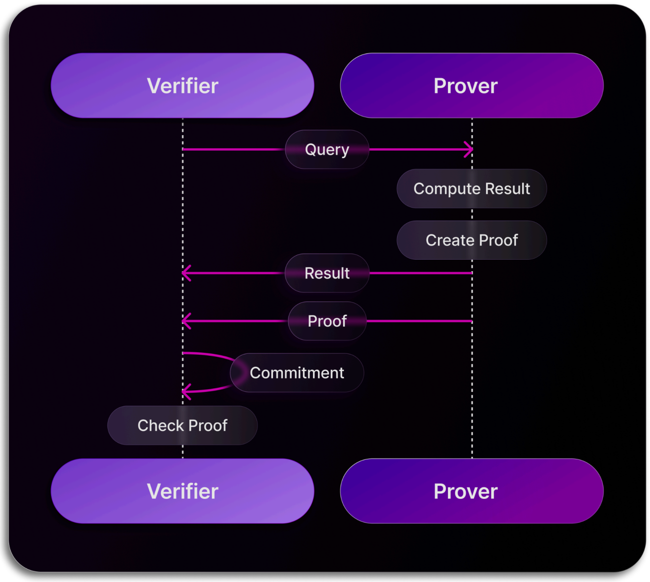

*Figure 4: Sequence Diagram of Query Request from Verifier to Prover*

### 3.1.3 Proof Protocol Overview

In the following sections, we go into detail about how the *Prover* generates a proof, and how the *Verifier* checks that proof. Broadly, this can be divided into four sections:

1. **Parsing:** The query text must be parsed into a format that the *Prover* and *Verifier* can agree to (Section 3.2).
2. **Query Execution and Protocol Builder:** This step is effectively a “proof setup” phase. This is where the ZK “circuit” is created and the query is actually executed. (Section 3.3)
3. **Relation Prover:** This is the step in which the bulk of the proof is actually generated, utilizing the multilinear sum-check protocol (Section 3.5).

4. **Commitment Evaluator.** The last step of the sum-check protocol requires a commitment opening. (Section 3.6)

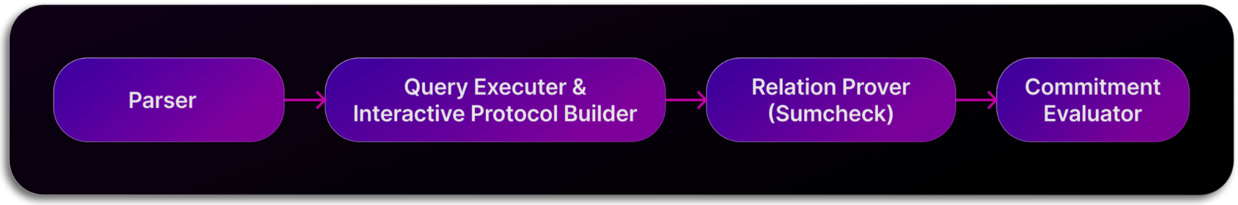
Parser
Query Executer & Interactive Protocol Builder
Relation Prover (Sumcheck)
Commitment Evaluator

*Figure 5: Proof Protocol Overview Diagram*

## 3.2 Parsing

The first step in the process of creating a Proof of SQL proof is parsing the SQL text. This effectively acts as a preprocessing step, creating an abstract syntax tree (AST) that can be consumed by the remainder of the process. See this resource for a more in-depth explanation of how many query engines work.

The parser works similarly to other parsers: by first creating an AST. This AST is a tree of SQL operations and components. For instance, consider the following query, which is converted to the AST shown below:

```
SELECT hash, timestamp FROM blocks WHERE block > 1000 AND transaction_count = 0
```

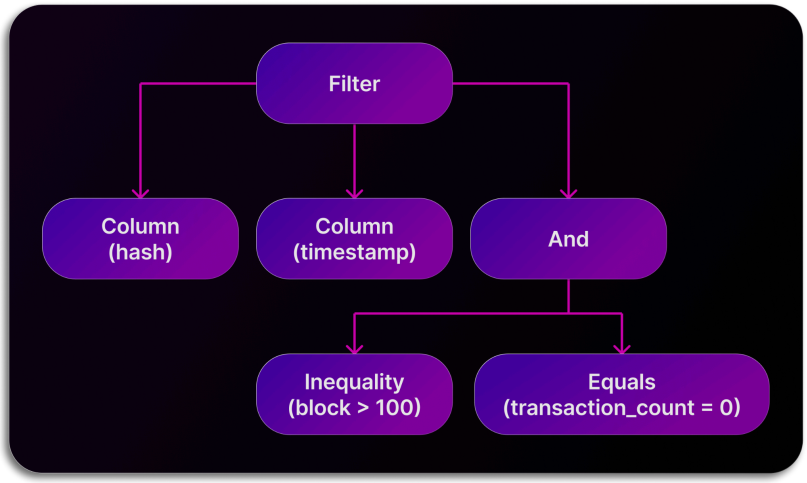

*Figure 6: Diagram of AST Nodes for Example Query*

## 3.3 Query Executor and Interactive Protocol Builder

The second step in the process of creating a Proof of SQL proof is to execute the query and build the protocol that will produce the proof. The resulting protocol is an interactive protocol, in part because the protocol depends on the data that is in the database. While this protocol is described as an interactive protocol, the Fiat-Shamir transformation allows the proof to be non-interactive. In the language of most other ZK-protocols, this is the step in which  the circuit is designed.

To execute the query and build the proof, the *Prover* passes over the AST nodes and does the following four things for each node:

1. Query execution
2. Witness generation
3. Commitment computation
4. Constraint building

To verify the query, the *Verifier* also passes over the AST node. When doing this, the *Verifier* only needs to do the last step out of the four that the *Prover* does (that is, constraint building).

### 3.3.1 Query Execution / Witness Generation

The most obvious example of a *Verifier* is a smart contract that is querying the database, but it can be any trusted party. For example, an end client's browser could act as a *Verifier*. Proof of SQL is necessary when an application has either restricted compute or restricted storage but still needs a security guarantee that the analytics run on data has been executed correctly, and that the underlying data has not been tampered with. The *Prover* is computationally intensive, whereas the *Verifier* is lightweight and designed to be executed on client devices or within smart contracts (which have extremely limited storage and computation capability). However, the *Prover* can be any device since there is no need to trust it, and in practice, it will be the cheapest resource available.

A key architectural concept worth highlighting here is the idea of a commitment*,* which is a type of digital fingerprint – a lightweight digest of the data in the table. Commitments are the source of truth for a table. In order to detect tampering, the *Verifier* compares the *Prover*'s response with the commitment. This is the primary reason for SXT Chain: to maintain the commitments for tables. In one sense, the Proof of SQL protocol can be thought of as a query engine, while Space and Time is the entire database management system. The remainder of this section is primarily focused on describing the query engine: Proof of SQL proper. The chain's *Validators* are responsible for maintaining these commitments. This is described elsewhere in the paper. For the purposes of this section, we will simply assume that the *Verifier* has trustworthy access to this commitment so that it is able to detect any tampering.

First, the *Prover* must generate the result of the query and any intermediate values needed to produce the result and proof. The word "witness" is used to mean intermediate values that are needed to verify certain steps of the query execution, but aren't known by the *Verifier*. The *Verifier* doesn't actually need to have knowledge of what the witness is, but the protocol guarantees that such a witness exists. The existence of a valid witness is sufficient to prove that the result is correct. This is, in part, why the term zero-knowledge applies, since the *Verifier* has zero knowledge of the witness.

Query execution and witness generation are almost the same, and are tightly linked. The SQL execution engine operates on table data in a columnar fashion, and produces intermediate values. The witness is the intermediate columns that are needed to verify the SQL execution. In some situations, additional witness columns on top of those needed solely for execution are also needed.

As an example, we will look at the previous query again, and will use the following table as the blocks table:

```
SELECT hash, timestamp FROM blocks WHERE block > 1000 AND transaction_count = 0
```

| **block** | **hash** | **timestamp** | **transaction_count** |
|---|---|---|---|
| 997 | 0xef6e7e8 | 01:23 | 2 |
| 998 | 0xad5674d | 01:33 | 1 |
| 999 | 0xcbc4567 | 01:44 | 0 |
| 1000 | 0x1ea13ea | 01:52 | 4 |
| 1001 | 0xc460f97 | 02:04 | 0 |
| 1002 | 0x96b501c | 02:15 | 6 |

The output of the Equals, Inequality, and And nodes would be:

| **Equals**<br>**(transaction_count = 0)** | **Inequality**<br>**(block > 1000)** | **And** |
|---|---|---|
| 0 (false) | 0 (false) | 0 (false) |
| 0 (true) | 0 (false) | 0 (false) |
| 1 (true) | 0 (false) | 0 (false) |
| 0 (false) | 0 (false) | 0 (false) |
| 1 (true) | 1 (true) | 1 (true) |
| 0 (false) | 1 (true) | 0 (false) |

While none of these are the actual result, they are used in the computation of the result, and are needed to verify that the result is correct. Thus, they are considered part of the witness, and must be generated by the *Prover*.

### 3.3.2 Commitment Computation / Multiple Rounds

Once the witness has been computed, it must be committed to, ensuring that the *Prover* cannot “change its mind” after the fact. This is the most computationally expensive step of the process and is therefore pushed to GPUs. For the most part, the commitment scheme can be treated as a “black box” compared with the rest of the protocol. See section 3.7 for more details on the exact commitment scheme used.

Some of the nodes in the AST require  a two-round protocol to efficiently create a witness. This is because the witness depends on random challenges from the *Verifier*. As a result, the commitment computation step and witness generation step execute twice. After the first computation of the commitments, new random challenges are sent from the *Verifier* (see, for example, the Group By protocol). When the commitment computation is done in a two-rounds fashion, the *Verifier* simulates the interactive protocol via the Fiat Shamir heuristic. This is done before passing over the AST nodes, meaning that the *Verifier* needs only one pass through the nodes.

### 3.3.3 Constraint Building

Finally, polynomial constraints are created that specify relationships that must hold between the witness columns. This is a very natural mapping because SQL is a relational language whose queries correspond cleanly to these polynomial relationships/constraints between the intermediate columns of the SQL execution. See sections 3.5 and 3.6 for more details on what these polynomial constraints look like.

## 3.4 Notation

We will briefly discuss some of the mathematical notation needed for the following sections. Most of this notation follows the literature fairly closely, although we diverge in some areas so that we can write things more concisely.

Since we are dealing primarily with data in tables that are processed in a columnar fashion, we will use notation that reflects this. We will notate columns of data with capital letters such as $A \in \mathbb{F}^n$ where $n$ is the number of rows. If needed, we can embed $A$ in a larger vector space simply by appending $0$'s to the column. Furthermore, we can think of $A$ as a multilinear polynomial in $\nu = \lceil \log_2(n) \rceil$ variables. That is, there is a unique multilinear

$f_A \in \mathbb{F}[x_0, \dots, x_{\nu-1}]$ such that $f_A(x_0, \dots, x_{\nu-1}) = A\left[\sum_{i=0}^{\nu} x_i 2^i\right]$ for $(x_0, \dots, x_{\nu-1}) \in \{0,1\}^{\nu}$. In other words, the bit representation of the index of a value $A$ gives the coordinates that evaluate to that value for $f_A$.

As an example, if $A = (100, 101, 102, 103, 104)$, then

$$\begin{aligned} f_A(X) = {} & 100(1-x_0)(1-x_1)(1-x_2) \\ & + 101x_0(1-x_1)(1-x_2) \\ & + 102(1-x_0)x_1(1-x_2) \\ & + 103x_0x_1(1-x_2) \\ & + 104(1-x_0)(1-x_1)x_2 \end{aligned} \quad .$$

We write $A \cdot B$ to denote entrywise multiplication. So, $A \cdot B = 0$ means that $A[i] \cdot B[i] = 0$ for all $i$. We write $\sum A$ to denote the sum of the (non-zero) entries of $A$. So, $\sum A \cdot B = 0$ means that $\sum_{i=0}^{n-1} A[i] \cdot B[i] = 0$.

Finally, we let $I_\ell$ denote the column that is 1 for the first $\ell$ entries and 0 otherwise.

## 3.5 Example Subprotocols

The following examples are explicit examples of nodes of the query plan. The first is an example of a boolean expression and the second is an example of a sum aggregation node.

### 3.5.1 Equality Protocol Builder

One of the simplest examples is an equality expression: Consider the Equals node from the example in sections 3.1 and 3.2, where we let $C$ be the transaction_count column and $R$ be the result of the Equals node.

It is straightforward to show that this result is correct if and only if there exists a witness column, $P$, such that, $C \cdot R = 0$ and $1 - R = C \cdot P$. These two equations are the constraints that must be satisfied. It happens that the following column is a valid witness.

| $C$ **- transaction_count** | $R$ **- result of Equals node**<br>**(transaction_count = 0)** | $P$ |
|---|---|---|

| | | |
|---|---|---|
| 2 | 0 (false) | $2^{-1}$ |
| 1 | 0 (false) | $1$ |
| 0 | 1 (true) | $0$ |
| 4 | 0 (false) | $4^{-1}$ |
| 0 | 1 (true) | $0$ |
| 6 | 0 (false) | $6^{-1}$ |

### 3.5.2 Group By Protocol Builder

For this example, we will describe a simplified version of an aggregation/GROUP BY clause. Consider the query `SELECT from_wallet, SUM(amount) FROM table GROUP BY from_wallet` for the table:

| $A$ - amount | $F$ - from_wallet | $T$ to_wallet |
|---|---|---|
| 74 | 1025 | 1034 |
| 24 | 1034 | 1099 |
| 12 | 1025 | 1099 |
| 56 | 1099 | 1025 |
| 22 | 1099 | 1000 |
| 45 | 1025 | 1025 |

The correct result is:

| $W$ - from_wallet (result) | $S$ - sum (result) |
|---|---|
| 1025 | 175 |
| 1034 | 24 |
| 1099 | 34 |

It can be shown that the result of this query is correct if and only if there exist witness columns $Q$ and $R$ such that the following hold for some randomly chosen $\beta$:

$\sum A \cdot Q - S \cdot R = 0$, $Q \cdot (F + \beta) = I_n$, and $R \cdot (W + \beta) = I_r$ where $n$ is the number of rows in the original table, while $r$ is the number of rows in the result.

It is important to note that $Q$ and $R$ depend on $\beta$, while $W$ and $S$ do not. If $\beta$ was chosen before the commitments of $W$ and $S$ were sent to the *Verifier*, the values of $W$ and $S$ could be changed based on the value of $\beta$ allowing a malicious *Prover* to provide the wrong result. In other words, $\beta$ must be a function of $W$ and $S$.

Suppose, for example, that $\beta$ happened to be $30000$. Then, we would have the following:

| $A$ | $F$ | $W$ | $S$ | $Q$ | $R$ |
|---|---|---|---|---|---|
| 74 | 1025 | 1025 | 175 | $31025^{-1}$ | $31025^{-1}$ |
| 24 | 1034 | 1034 | 24 | $31034^{-1}$ | $31034^{-1}$ |
| 12 | 1025 | 1099 | 34 | $31025^{-1}$ | $31099^{-1}$ |
| 56 | 1099 | 0 | 0 | $31099^{-1}$ | 0 |
| 22 | 1099 | 0 | 0 | $31099^{-1}$ | 0 |
| 45 | 1025 | 0 | 0 | $31025^{-1}$ | 0 |

In summary, this would be the sequence of interactions in the interactive version of the protocol:

- *Verifier* → *Prover*: The query: `SELECT from_wallet, SUM(amount) FROM table GROUP BY from_wallet`
- *Prover* → *Verifier*: The results and any intermediate results: $W$ and $S$.
- *Verifier* → *Prover*: Random challenges: $\beta$.
- *Prover* → *Verifier*: Witness columns: $Q$ and $R$.
- *Verifier* and *Prover*: Remaining interactive protocol (sum-check + commitment opening)

## 3.6 Relation Proof Protocol (Sum-check)

The first two steps of the protocol (sections 3.2 and 3.3) allow the *Prover* and *Verifier* to agree on a collection of relations between columns of data and provide the *Verifier* with commitments to these columns of data. The next step is for the *Prover* to convince the *Verifier* that these relations hold.

Once the first two steps of the protocol are complete, the *Prover* has constructed a collection of witness columns and a collection of polynomial constraints that these witness columns satisfy. Additionally, the *Prover* has sent commitments to each of the witness columns to the *Verifier*, and the *Verifier* has constructed the same collection of polynomial constraints and sums.

In other words, the *Prover* has a collection of columns $A_0, \ldots, A_m$ and the *Verifier* has the commitments $Commit(A_0), \ldots, Commit(A_m)$. Furthermore, both the *Prover* and *Verifier* have agreed on a collection of relations and sums between these vectors: a collection of low degree polynomials $p_0, \ldots, p_{\ell-1} \in \mathbb{F}[y_0, \ldots, y_m]$ and $q_0, \ldots, q_{s-1} \in \mathbb{F}[y_0, \ldots, y_m]$.

It is the *Prover's* job to show that $p_k(A_0, \ldots, A_m) = 0$ for each $k$. Note that this is the same as saying that $p_k(f_{A_0}(X), \ldots, f_{A_m}(X)) = 0$ for all $X \in \{0,1\}^\nu$, but not for arbitrary $X$. Additionally, the *Prover* must show that $\sum p_k(A_0, \ldots, A_m) = 0$ for each $k$.

At this point, the *Verifier* sends the *Prover* the challenges $\rho_0, \ldots, \rho_{\nu-1}, \alpha_0, \ldots, \alpha_{\ell-1}, \beta_0, \ldots, \beta_{s-1}$. This gives a structured challenge column, $Q$, defined by $f_Q(x_0, \ldots, x_{\nu-1}) = \rho_0^{x_0} \cdots \rho_{\nu-1}^{x_\nu-1}$ for $X \in \{0,1\}^\nu$. This, in turn, allows the *Prover* and *Verifier* to agree on the following combined constraint:

$$\sum Q \cdot (\alpha_0 p_0 + \cdots \alpha_{\ell-1} p_{\ell-1}) + \beta_0 q_0 + \cdots + \beta_{s-1} q_{s-1} = 0$$

This constraint holds with high probability exactly when the original constraints hold as well. This is proven using the multilinear sum-check protocol. See section 4.1 of Thaler[5] for an excellent explanation of how sum-check works. The last step of sum-check is the evaluation of the combined polynomial at a random point, which reduces to evaluations of $A_0, \ldots, A_m$ at that point. These evaluations are proven using the commitment scheme and the commitments $Commit(A_0), \ldots, Commit(A_m)$, described in the next section.

## 3.7 Commitment Evaluator

After the first three steps (sections 3.1, 3.2, 3.5), the *Prover* has produced a proof of the query along with the result with only one missing component: the opening/evaluation of the commitments/columns of data. There are a variety of commitment schemes that enable this, so for the rest of the protocol, this can be treated as a black box. However, because the biggest computational cost revolves around the commitments and their uses, the decision of which commitment scheme to use is important. Proof of SQL uses the Dory commitment scheme[6] and the HyperKZG commitment scheme, and are working on a third, novel, scheme that will supersede both. Importantly, the Space and Time *Validators* are designed to maintain multiple commitment schemes, and more can be added in the future as new, state-of-the-art commitment schemes are developed. There are several motivating factors that drive these choices. We will list them in rough order of importance.

1. **Updateable/Homomorphic Commitments:** As a reminder, in the Proof of SQL protocol, the *Prover* holds the data that is in the tables, while the *Verifier* holds onto the commitments. Since the vast majority of tables in a database are not static, there needs to be some mechanism by which the *Verifier* can update commitments when new data is appended to a table. More specifically, we want some function such that:

   ```
   new commitment = Update(old commitment, appended data)
   ```

   In other words, data access is not required in order to update commitments to the data. Any homomorphic commitment scheme has this property. Dory and HyperKZG are homomorphic commitment schemes. In addition to being updateable, homomorphism has other helpful properties, such as supporting deltas/snapshots.

2. **Efficient Verifier:** Because the amount of data that is being queried against can be very large, we need the *Verifier* to have sub-linear performance. Dory's and HyperKZG's verifiers run in $O(\log n)$ time. However, the HyperKZG is additionally cheap enough to be run inside an Ethereum smart contract.
3. **Transparency:** While this is a beneficial property for any protocol, it is particularly important for Proof of SQL because large data volumes exist. Without a transparent setup, a trusted setup would be necessary, which would require an upper bound on table size. While a large upper bound could be set, making this sufficiently large would require a very large trusted setup. Unfortunately, HyperKZG requires a trusted setup. However, there is a highly reputable universal setup initial run for ZCash (and later others) that supports up to 0.5 billion rows of data.
4. **Partition Friendly:** In any efficient database, partitioning and other smart indexing schemes are essential to running performant queries. In particular, we need to be able to run queries without needing to process all of the data in the tables. In addition to being homomorphic, Dory is a 2D commitment scheme, and the *Prover* time is $O(\sqrt{n})$ in certain circumstances. This means that access to all of the data is not needed for certain evaluations.

# 4 Use Cases for the Next-Generation of Web3

## 4.1 Building a Flexible ZK-Rollup/L2

Proof of SQL can power bridgeless, gas-minimized microtransactions across multiple ledgers. ZK-rollups reduce Ethereum gas fees by batching offchain transactions and posting a single proof onchain. With Proof of SQL, developers can build a ZK-rollup on top of Space and Time, instantly appending transactions to a tamperproof ledger and rolling them up to the main chain. Unlike typical L2 solutions, Proof of SQL supports data processing (joins, aggregations, etc.) and scales beyond a single ledger, while providing instant finality and bridging-free cross-chain interactions. Transactions incur gas only for deposits and withdrawals; execution remains offchain, dramatically reducing fees and complexity.

## 4.2 Secure Bridges and Multichain Data Backends

Crypto app ecosystems remain fragmented, with each chain operating in isolation. Space and Time unifies these silos by indexing data from multiple chains and storing it in tamperproof tables. A smart contract on Ethereum, for instance, can verify real-time state from BNB Chain via Proof of SQL—no bridging required. This reduces engineering overhead, eliminates the need for separate cross-chain infrastructure, and enables more reliable bridge security: a bridge contract can verify locked assets on Chain A before releasing tokens on Chain B, using ZK-proven indexed balances for fraud-resistant transfers.

## 4.3 Dapp Backend (Decentralized)

Traditional dapps rely on a lightweight onchain contract plus a centralized backend for data processing. Space and Time replaces the centralized component with a serverless, decentralized backend and database, providing APIs for SQL queries, role-based auth, and Python-based business logic. Smart contracts emit events that Space and Time automatically indexes, and offchain data can also be stored for fast retrieval and tamperproof verification. Developers retain a purely decentralized stack without compromising performance.

## 4.4 Data-Driven Lending

DeFi lending protocols today offer identical interest rates to every borrower, ignoring onchain credit history. Space and Time solves this by enabling a lending contract to access ZK-proven borrower data, aggregating multiple loan histories (onchain or offchain) into a verifiable credit score. Lenders can offer dynamic rates based on repayment patterns, leading to higher returns for liquidity providers and lower rates for reputable borrowers.

## 4.5 Cross-Chain Financial Instruments

Complex financial instruments—options, futures, derivatives—rely on diverse data sources across multiple chains. Proof of SQL simplifies this by offering a single ZK-verifiable platform to pull token prices, trade volumes, and more from any chain in real time. Derivatives gain precise and trustless valuations without manually bridging data or relying on a patchwork of APIs.

## 4.6 Gaming Rewards

Smart contracts alone can't process nuanced offchain gameplay data. Space and Time allows developers to store and query detailed player metrics—like playtime, achievements, or NFT activity—and then deliver rewards based on advanced, tamperproof logic. Complex attributes (cooperative play, skill ranking, etc.) can be aggregated offchain, yet trustlessly verified onchain.

## 4.7 Social Apps

Developers building Web3-native social networks must overcome:

1. **Massive Scale**
   Blockchains are ill-suited for petabyte-scale data. Storing everything onchain is infeasible, and centralized solutions lack trustless guarantees. By offloading high-volume data (posts, comments, likes) to Space and Time's tamperproof tables, apps can handle billions of interactions without sacrificing decentralization.
2. **Rewarding Creators Onchain**
   Tracking engagement metrics (e.g., views, watch time) for content payouts is nearly impossible onchain. By processing these metrics in Space and Time—then rolling up payouts to smart contracts—developers can deliver provably fair, transparent rewards without relying on a centralized, tamperable backend.
3. **Bootstrapping User-Created Content**
   Web3 social apps face a chicken-and-egg problem: limited initial content and limited user interest. A shared "content posts" dataset in Space and Time allows multiple apps to read/write from the same user-owned social graph, avoiding siloed content and driving network effects.
4. **Private Interactions**
   Blockchain data is public, making private messaging a challenge. ZK proofs in Space and

Time let users prove message authenticity without revealing content. This balances privacy with the trustless ethos of Web3, keeping private conversations secure while retaining verifiability.

By leveraging Space and Time's decentralized database and ZK tools, Web3 social applications gain the scalability, creator incentives, composability, and privacy they need—all without resorting to centralized infrastructure.

## 4.8 Settlement Systems and Third-Party Auditing

Financial institutions often use consortium ledgers to share data among trusted parties, but these ledgers can't validate the underlying transactions. Banks typically store granular order book data in offchain SQL databases and periodically update the consortium ledger with summarized results—leaving no cryptographic proof that transactions are accurate before the data is posted.

Space and Time closes this trust gap by serving as a verifiable SQL database. Each transaction is cryptographically provable via **Proof of SQL**, ensuring that no institution can manipulate its P&L or other sensitive data. This tamperproof model also makes compliance simpler: every operation (insert, update, delete) is immutably recorded and easily auditable for regulations like GDPR. Because Space and Time can be deployed across multiple jurisdictions (e.g., EU-based data warehouses), institutions can maintain data sovereignty while still running global analytics against their datasets.

Financial institutions that leverage Space and Time for order book management gain immediate, low-latency operations with cryptographic assurances that no third party is tampering with the data. These order books can optionally settle on a public or private chain, but Space and Time itself already delivers the decentralized verifiability critical for secure finance.

## 4.9 Real World Assets (RWAs)

Enabling tokenization use cases for real-world assets like real estate, event tickets, and collectibles is paramount. A tokenized concert ticket, for example, might include static information (venue, artist, seat) alongside dynamic details (real-time pricing rules or availability). By storing such data in Space and Time, smart contracts can securely query and retrieve up-to-date information whenever a transaction occurs, ensuring provably accurate and transparent execution on the blockchain.

Beyond tokenization, Space and Time also serves as a verifiable backend for security and compliance. It enables tamperproof data storage for wallet histories, risk scores, and compliance lists (e.g., OFAC-banned or KYC-verified wallets). Security platforms can query these tables in real time to validate risk scores or check against regulatory blacklists before executing transactions, while organizations can create and populate private whitelist/blacklist tables with offchain or onchain data. With verifiable integrity and real-time queries, Space and Time underpins secure, compliant operations for any tokenized asset or blockchain-based application.

## 4.10 Transaction Security and Wallet Whitelists

A security platform can use Space and Time to access verifiable wallet history, store risk scores in tamperproof tables, and query real-time risk assessments before transactions. Similarly, compliance data like OFAC-banned or KYC-verified wallets can be stored transparently and queried by smart contracts with ZK proofs to ensure compliance.

Clients can create and populate whitelist/blacklist tables with offchain or onchain data, enabling verifiable, secure, and compliant operations in a trustless environment.

## 4.11 Liquidity Pool Rebalancing Based on Market Conditions

Liquidity providers in AMMs like Uniswap v4 often face impermanent loss and imbalanced token ratios during market volatility, reducing their rewards. Dynamic pool rebalancing helps by:

1. **Maintaining Asset Ratios**: Ensures pools remain near target ratios (e.g., 50:50) despite token price fluctuations.
2. **Reducing Impermanent Loss**: Periodic rebalancing mitigates losses and optimizes pool configurations for higher yields.
3. **Enhancing Security**: Prevents large imbalances that attract arbitrage or exploitation.

Rebalancing must be trustless to prevent manipulation. Space and Time tracks pool history, analyzes real-time market conditions (e.g., spot prices, volatility), and provides results with ZK proofs to smart contracts. This ensures rebalancing only occurs when predefined conditions are met, allowing liquidity providers to maximize returns securely without managing infrastructure or relying on external actors.

# 5 Economic Model for SXT Chain

The SXT Chain economic model is designed to sustain a decentralized, permissionless ecosystem that rewards participants for actively contributing to network security, high-quality data provisioning, and efficient query processing.

*Validators* maintain the cryptographic integrity of the network by securing data commitments, processing data insertions, and supporting ZK-proven SQL query execution. In return, *Validators* receive gas fee-based block rewards that compensate them for securing and operating the network infrastructure. *Table Owners* contribute by maintaining high-quality datasets and, in exchange for onboarding and updating high-quality data, they receive a portion of the query fees, which they can delegate to *Prover* nodes to ensure efficient ZK-proven query execution. This model creates a decentralized marketplace where contributors are compensated for servicing and securing the network and providing valuable, queryable data.

## 5.1 SXT Token Utility

The SXT token is the native utility token of SXT Chain, necessary for securing the network through decentralized and permissionless validator participation. *Validators* must stake SXT tokens as collateral to participate in cryptographic processes such as witnessing ingested data, updating commitments on the data per the Proof of SQL protocol, threshold-signing the commitments onchain, and ultimately offering crypto-economic security for network activities (e.g., tamperproof tables and query execution). Staking serves as a security guarantee to ensure *Validators* act in good faith.

*Validators* earn compensation from network fees for performing essential services like tamperproofing datasets and executing verifiable queries. Importantly, staked SXT tokens may be partially or fully slashed when the network discovers any of the following:

- Node downtime
- Malicious tampering of tables (i.e., manipulation of cryptographic commitments for tables held onchain, caught by EVM contracts during verification of ZK-proven query results)
- Unverifiable query execution, or poor *Prover* response SLAs
- Inability to participate in the cryptographic procedures necessary for tamperproof chain functionality (i.e., threshold-signing updated commitments on ingested data, as part of chain consensus).

Staking SXT is an active commitment to secure and operate core functions of SXT Chain. *Validators'* roles are fundamental to network integrity and data reliability, and the SXT token facilitates this utility-driven coordination.

## 5.2 Security Budget Vault

*Stakers* play a crucial role in securing the network by staking SXT tokens against *Validator* keys (using Space and Time-deployed contracts on popular EVM chains), where *Validators* risk slashing of their own stake as well as delegated stake for improper network participation. Through this staking mechanism, *Validators* and those who delegate to them collectively underwrite the integrity and reliability of SXT Chain's data.

*Stakers* participate in two ways:

- **Validator Stakers**: Operate *Validator* nodes and actively perform essential cryptographic functions, including data attestation, threshold signing, and query verification. *Validators* directly contribute to the security and accuracy of SXT Chain and are responsible for maintaining high service standards.

- **Delegated Stakers**: Allocate SXT tokens to *Validators* to support their operational capacity and share proportionally in fee-based compensation earned from *Validators'* active participation. Delegated staking helps increase the *Validator* set's capacity to secure the network without requiring every participant to run a node.

The *Staker* rewards vault distributes network-generated fees (e.g., query payments and data insertion fees) to *Stakers* based on their active role in maintaining network security and reliability. To bootstrap a decentralized network and facilitate early validator participation, a time-limited emission schedule of SXT tokens will supplement network fees, phasing out as onchain activity grows to sustain validator incentives. This ensures a fee-driven and self-sustaining economic model that aligns long-term incentives with network stability.

## 5.3 Table Owner Rewards Vault

*Table Owners* actively create and maintain tamperproof data tables on SXT Chain. They earn fees through:

- **Insert Data Payments**: Fees paid by *Indexer* nodes or third-party data providers to insert data into SXT Chain for secure, tamperproof storage.
- **Query Payments**: Fees paid by clients (such as DeFi protocols, dapps, and other users) for executing SQL queries over tamperproof datasets. *Prover* nodes process these queries and generate ZK proofs, while *Table Owners* receive a share of these payments for maintaining the underlying datasets.

Query fees are shared between *Table Owners* and *Validator Stakers*, aligning incentives across participants to ensure both data quality and network security. This cooperation incentivizes permissionless third parties to onboard high-quality datasets to SXT Chain. By creating their own tables and populating them with useful data, *Table Owners* can attract frequent queries from the network's clients (e.g., DeFi protocols and dapps). The more queries their datasets receive, the greater the fees they earn, and they are thus incentivized to ingest more data to their tables.

*Table Owners* actively curate, maintain, and update data tables to ensure relevance, accuracy, and query responsiveness. Their compensation from query payments reflects their ongoing effort and value contribution to the ecosystem.

## 5.4 Network Fee Structure and Data Marketplace

SXT Chain's fee model is designed to compensate participants based on their active contributions to network security, data availability, and query execution. Fees are sourced from actual network activity, creating a sustainable and service-driven economy.

**Query Payments:** Gas fees paid for ZK-proven SQL queries are shared between *Validators* and *Table Owners*. *Validators* earn their fees as block rewards for securing the network, while *Table Owners* are compensated for maintaining useful datasets. *Table Owners* can optionally delegate part of their earned fees to specific *Prover* nodes (or pools of *Prover* nodes working together to fulfill high-concurrency client requests) to ensure timely execution of ZK-proven queries. This mechanism incentivizes reliable query fulfillment from the *Prover* node set.

**Insert Data Payments:** Gas fees for inserting data into tamperproof tables are allocated entirely to *Validators* as block rewards. This ensures *Validators* are consistently incentivized to witness ingested data as part of BFT consensus, update commitments, and secure the tables.

**Data Marketplace:** By sharing a significant portion of query-related fees with *Table Owners* (e.g., 50% of query payments), SXT Chain creates an open marketplace for high-quality, queryable data. *Table Owners* are directly incentivized to onboard and maintain datasets that attract client queries, aligning their interests with the network's overall utility and trustworthiness. As demand grows for specific datasets, *Table Owners*' compensation scales proportionally, reinforcing a self-sustaining ecosystem of valuable data provisioning.

# 6 Conclusion

A new generation of cryptographic tools are essential to maintaining the trustless guarantees of blockchain while enabling, for the first time, onchain data processing for smart contracts. Space and Time addresses these gaps by providing a decentralized network for trustless, cryptographically verifiable data processing, often initiated by client smart contracts. The decentralized platform ingests real-time indexed blockchain data from major chains (Ethereum, Bitcoin, ZKsync, etc.) and allows users to integrate their own offchain data sources, such as TradFi markets and gaming systems, with trust-minimized consensus on data ingest.

Using a novel ZK protocol, Space and Time ensures tamperproof computations at scale. SQL queries are executed with ZK-proven results, enabling smart contracts, enterprises, and financial institutions to verify data integrity in a trustless manner without compromising functionality (aggregations, filters, sorts, joins, etc.).

Participants operate permissionless nodes and earn query fees paid by clients such as smart contracts or enterprises. Fees and offchain fee streams drive network growth, ensuring that value accrues to users and operators of the network while also reducing costs. By combining trustless, scalable data processing with economic incentives for network participants, Space and Time empowers a new generation of data-driven dapps, ZK-rollups, and DeFi systems.

# Appendix

## A Data that Smart Contracts Can Access without Space and Time

Smart contracts on the Ethereum Virtual Machine (EVM) can natively access a range of data pertaining to the blockchain and their own state. However, they can't natively access information outside of the Ethereum network without the use of oracles. Here's what a smart contract can natively access:

- **Blockchain Metadata:** Block number (block.number), block timestamp (block.timestamp), and the block's gas limit (block.gaslimit), etc.
- **Gas Information:** The gas limit of the current block, the gas price (in wei) of the transaction that is currently being executed, and the amount of gas remaining for the current function execution (using gasleft()).
- **Transaction Information:** The transaction's sender (msg.sender), the value of ether sent with the transaction (msg.value), and the transaction's data payload (msg.data).
- **Account Data**
    - **Balance:** A contract can query the balance of any wallet or smart contract address (including its own) with address.balance.
    - **Contract Code:** A contract can get the bytecode of another contract using the extcodesize and extcodecopy opcodes.
- **Storage and Memory:**
    - **Contract Storage:** Each contract has its own persistent storage, and it can read from and write to this storage. This is essentially the contract's “state.”
    - **Memory:** This is a temporary location where data can be stored and manipulated before being placed in storage or returned as output. It's ephemeral and will be erased between function calls.

- **Interactions with Other Contracts:** A contract can call functions on other contracts, sending ether and data. The result of such a call can be captured and processed. This means that a contract can, for example, query the state or balance of another contract, or even trigger actions in that contract. Similarly, a contract can reference its own address using address(this).
- **Ether Operations:** A contract can send ether to any address and determine its own ether balance.
- **Create New Contracts:** A contract can deploy a new contract using the CREATE opcode.

Note that while smart contracts can access all the above data, they cannot natively access:

- Smart contract events (logs) emitted on their own chain or on other chains.
- Historical onchain data such as transaction history.
- Cryptocurrency prices (aggregated token price feeds or liquidity pool swap prices).
- Information about the real world (e.g., stock prices, weather data).
- Offchain data (e.g., content of a website, API responses).
- States from other blockchains.

For these types of datasets, smart contracts usually rely on oracles—trusted or trust-minimized data providers that relay offchain information onto the blockchain.

## B HTAP to Replace Point Solutions

In the data warehousing industry, hybrid transactional/analytical processing (HTAP) represents a paradigm shift that resolves a longstanding challenge in traditional markets. Understanding HTAP is crucial to grasping how the Space and Time database maximizes efficiency and versatility.

In many Web2 companies and conventional markets, data processing traditionally involves a cumbersome process. Businesses often employ separate database solutions for transactional processing (OLTP) and analytical processing (OLAP). These solutions serve specific purposes but also posed significant challenges:

- **Data Redundancy and Complexity:** Maintaining separate systems leads to data redundancy and complexity, as data needs to be transferred between OLTP and OLAP databases. This not only consumed time and resources but also increased the risk of data inconsistencies.

- **Latency Issues:** Analytical processing often suffers from latency issues, as it relies on periodically refreshed data from transactional databases. Real-time insights are a challenge to achieve.

- **Scalability Challenges:** Scaling these separate solutions in response to growing data volumes and user demands is often an intricate and expensive process.

HTAP addresses these issues by combining OLTP and OLAP capabilities within a single system. Here's how it works:

- **OLTP and OLAP Integration:** HTAP systems like Space and Time have both an online transactional processing (OLTP) engine and an online analytical processing (OLAP) engine working in tandem. This allows data to be simultaneously ingested, processed, and analyzed within the same environment.

- **In-Memory Caching:** HTAP systems often employ in-memory caching to enable low-latency transactional processing. This means that even complex analytical queries can be executed without significant delays.

- **Eliminating Data Redundancy:** By consolidating data processing into one system, HTAP eliminates the need for data redundancy and complex data transfers between different databases.

- **Real-Time Analytics:** With HTAP, analytical processing can be performed on real-time, up-to-date data, enabling organizations to gain insights without the delays associated with data extraction and transformation.

HTAP, as utilized by Space and Time, represents a leap forward in database efficiency and functionality. It harmonizes transactional and analytical processing, eliminates data redundancies, and offers real-time analytics, all within a single, streamlined system. This approach not only simplifies data management but also paves the way for more agile and responsive data-driven decision-making.

## C Proof of SQL “Bring Your Own Database”

Proof of SQL is a ZK proof attached to SQL databases which cryptographically proves to a client that both query execution and underlying tables are untampered with. One can leverage the Proof of SQL protocol to trustlessly verify query results returned from not only the Space and Time decentralized database, but also other outsourced traditional database/data warehouse solutions..

For example, one could “attach” the cryptographic protocol to PostgreSQL, Snowflake, Apache Spark, Google Bigquery, AWS Athena/Redshift, Microsoft Fabric, etc. This would allow users already building on top of these query tools to connect them directly to smart contracts without breaking the blockchain’s trustless model, or to provide proof of query execution to a verifier on an external client device. In essence, developers can “bring their own database,” verified with Proof of SQL.

Here’s how it works: a Space and Time *Prover* node is connected adjacent to the database engine. This is the primary integration point. The *Prover* is responsible for executing tamperproof queries. To facilitate this, the *Prover* requires access to table data. This can be achieved through access to the database’s local storage or external tables, or by sending requests directly to the database itself. With Proof of SQL, the root of trust is established by creating virtual ‘tamperproof tables’ inside the target attached database. As data gets added to these tables by clients, special hashes (or “commitments”) are updated. Later, when validating a query and its associated ZK proof, these commitments are used to confirm its validity.

When a tamperproof SQL query reaches the database and is directed to the *Prover*, the result, accompanied by its proof of correctness, is generated. This proof-result pair is then transmitted to the *Validator* node, where verification takes place. Additionally, Space and Time offers both commitment creation and verification functionalities through a client-side library. This shifts the root of trust to the user. Some clients prefer this approach, while others choose to delegate verification to the *Validator*, which carries out this role on their behalf.

To enable Proof of SQL to ZK prove that queries against data were executed accurately and that the underlying data hasn’t been tampered with, one must simply:

1. Provide Space and Time *Prover* node access to database storage (local or external).

2. Position the Space and Time *Validator* (which logs tiny digital fingerprints of the data inserted, and uses these fingerprints to ZK-verify query results coming back from the database) in front of the database as a proxy or load server.

3. Load the data into the database through the *Validator*. Queries executed against the database should be routed through the *Validator* if they need to be ZK-verified (to validate that the query results and underlying tables have not been tampered with).

# D Leveraging an Append-Only Database as a Tamperproof Offchain Ledger

An append-only database/data warehouse can be designed to emulate some of the essential features of a blockchain (and we've done exactly that in our development of the Space and Time database):

- **Immutability:** Tables are append-only; transactions cannot be modified or deleted once written.
- **Double-Spend Prevention:** Before a transaction is added, a mechanism compares it against the historical record to verify that the respective assets haven't already been spent.
- **Serialized Transaction Ordering:** By nature, transactions are recorded in the order they are received and processed.
- **Transaction Consistency:** The database can be implemented with specific consistency logic to ensure that only transactions that meet predefined validity criteria are appended. Note: In the case of Space and Time's Proof of SQL protocol, the *Transaction* nodes implement this logic to verify/authorize requests/transactions, cryptographically ensuring that unapproved transactions cannot occur.

However, other key characteristics of the blockchain are not inherent to an append-only database:

- **Zero-Trust:** In a blockchain network, transactions are distributed; rules are enforced by consensus rather than by a central authority. A single append-only database requires inherent trust in the entity that maintains it. Though this problem persists when using traditional database systems, Space and Time solves it as the only decentralized database system, relying on a global committee of user-operated nodes that come to consensus in order to prove the validity of ingested data (and verify/authorize requests/transactions).
- **Concurrency:** Block intervals and consensus mechanisms ensure that a blockchain ledger remains consistent amid multiple concurrent transactions. An append-only database would require its own mechanism to queue concurrent transactions appropriately and ensure consistency. Space and Time has implemented consensus in the network to appropriately serialize transactions.
- **Cryptographic Security:** Cryptographic chaining, where each block references a hash of the previous, enhances the integrity of a blockchain network. Though not strictly necessary in the case of an append-only database, such measures provide additional security against tampering. Space and Time has implemented the Proof of SQL protocol for cryptographic guarantees on the underlying datasets.
- **Finality:** Blockchains, especially those using probabilistic consensus mechanisms like Proof of Work, operate with progressive finality: each transaction becomes increasingly

irreversible as more blocks are added after it. In an append-only database, finality is immediate upon appending, but the assurance of this finality depends on the security and trustworthiness of the system. Space and Time offers instant finality through the Proof of SQL protocol.

In order for an append-only database to function as an offchain tamperproof ledger, these features must be guaranteed by ancillary mechanisms or by leveraging Space and Time's Proof of SQL protocol directly.

# E Space and Time in the Market

## E.1 Web3's evolution as a digital economy powering novel apps

While Web3 can trace its roots back to the first blockchain network in 2009, the technology has rapidly matured over the last three years by expanding throughout the tech stack, improving usability and accessibility, and accelerating throughput. Advancements in Web3 have led to the broader acceptance and adoption of digital assets as an asset class, particularly by established financial services companies.

Web3 is in the midst of transitioning into a new chapter that will see an even more rapid maturation of the technology, a user base that grows from the millions to the billions, and ultimately, the mass adoption of blockchain technology across all industries. The world's largest enterprises and financial institutions have made significant investments to leverage blockchain, and this will only continue to accelerate.

As blockchain technology becomes widely adopted by enterprises, distinctions between Web2 technology and Web3 technology will become less significant. Enterprises will leverage both centralized and decentralized ledgers for a variety of applications and business models (where "trustless" vs. "trust-required" considerations are at play), many of which have not yet been imagined. Blockchain will transition from an emerging technology to a fundamental layer in the enterprise tech stack, Web3 and Web2 will become wholly interoperable, and the application of AI at scale will drive further innovation.

Some recent stats to underline the market today:

- Over $20B of value settled daily on the major blockchains
- Over 750M smart contracts across the major chains
- Over 50M daily transactions across the major chains
- Over 100K dapps across major chains
- Over 35K Web3 developers building on the major chains

An investment today in Space and Time provides two main sources of value that are uniquely in the same company and in the same platform: 1) delivering familiar database tools as a Web3-native experience and 2) providing verifiable (trustless) compute, where all activity in the database is cryptographically proven.

## E.2 Proven value in data warehousing

*The need for a platform to support data storage/data warehousing and scalable analytics both cross-chain and offchain.*

While there are many use cases today in Web3 that run on a single blockchain, the more valuable use cases will integrate data from multiple blockchains and offchain centralized data sources. As financial services firms take on custodial roles for digital assets, for example, analytic and reporting needs will increase. Those analytics will need to incorporate blockchains, centralized ledgers that track offchain assets, and databases that store customer data.

This nascent market, void of any popular solutions today, is likely to evolve similarly to how cloud data warehousing and analytics evolved. Snowflake, a popular cloud data warehouse, held the largest software IPO ever at the time. Data management software today is a global $75B market with high single-digit to low double-digit growth annually. AI platforms specifically are growing at 30+%.

While traditional Web2 database technologies are point solutions that focus on either transactional queries (OLTP) or scalable analytics (OLAP), Space and Time combines the two efficiently in a unified platform. Space and Time has accrued over $10M ARR, with an extensive list of established Web2 enterprises in financial services and accounting, as well as Web3 gaming companies that need to track tokens across multiple chains and DeFi companies that need to incorporate offchain analytics. This growth in logos and economy makes Space and Time one of the fastest-growing data analytics companies ever.

While the total market size today for this fast-moving category is difficult to calibrate, every Global 500 company and all Web3 companies (~23K+) will need a database/analytics platform with Space and Time's functionality. Estimating based on Space and Time's current pricing, this market would be conservatively sized at ~$700M–$1B today, and quickly moving to a multi-billion market over the next few years.

## E.3 New compute paradigm removes trust requirements

*Establishing verifiable compute as the fabric of Web3.*

Space and Time's breakthrough in cryptographically provable data processing solves many of the developer, user, and scalability issues present in smart contracts and blockchain today. In fact, it even offers a solution to traditional financial institutions that require their own offchain transactions and market data to remain tamperproof. Thus, "verifiable compute" will effectively create a new market category and establish a standard among enterprises.

Space and Time is a verifiable compute solution that integrates chain connectivity and ZK provability along with familiar database storage/compute, and the stack is growing at record speed. This solution seamlessly bridges onchain and offchain systems, effectively introducing verifiability to the enterprise data management stack while simultaneously scaling the capabilities of blockchains.

To date, this platform is already being embedded in the fabric of Web3.

Market coverage is as follows:

- 25% of all Web3 developers utilizing the platform with a subset contributing to the Space and Time open-source project
- 25% of capital volume of L1 blockchains seamlessly integrated and indexed
- 90% of capital volume of L2s and protocols seamlessly integrated and indexed

Space and Time has driven a major innovation by bringing Proof of SQL to market. Simply put, in a world that operates at the intersection of Web2 and Web3, there is significant value in proving that a query was executed properly or that the data used to execute business logic within a smart contract was properly filtered/aggregated. Proving the accuracy and lineage of data is already a key topic in the training of LLMs for AI, and there are already numerous companies integrating AI with blockchains. Lastly, companies of all sizes have embraced Proof of SQL, particularly in the area of compliance and auditing.

At the core of all these themes is the strong market demand for trustless verification that data is correct and was computed on properly—which is different from blockchain transaction verification. Both are important, but the balance is tilting toward assurance of accurate data and computation. Enterprises want confidence that their own engineers, as well as third-party partners, have not manipulated the datasets core to their operations (particularly financial data, where there exists a strong incentive for bad actors to tamper).

Estimating market potential is based more on analogs versus. market participants multiplied by a price model. That said, Space and Time is vying for a near-term podium on which Proof of SQL is established as the gold standard for all queries. This would be monetized by either cash payments per query or through a SXT token, which is part of the roadmap today.

One pertinent example around financial data verifiability is the payment processing space, which is a $65B market growing at greater than 10% annually and expected to approach $200B by 2032... and this is only one use case for Proof of SQL. Another example is the global SWIFT network, which connects 11K institutions. Total revenue for the SWIFT network was 900M euros in 2020, and this is a member-owned cooperative.

# References

[1] Stock option volume report. MarketChameleon.com. (n.d.). https://marketchameleon.com/Reports/optionVolumeReport

[2] White, J. T., & Dykstra, S. E. D. (2022, December 13). Methods for Verifying Database Query Results and Devices Thereof. https://image-ppubs.uspto.gov/dirsearch-public/print/downloadPdf/11526622

[3] *Biscuit authorization (RBAC)*. Space and Time. (n.d.). https://docs.spaceandtime.io/docs/biscuit-authorization

[4] *Official Legal Text*. General Data Protection Regulation (GDPR). (2022, September 27). https://gdpr-info.eu/

[5] Justin Thaler (2022), “Proofs, Arguments, and Zero-Knowledge”, Foundations and Trends® in Privacy and Security: Vol. 4, No. 2–4, pp 117–660. DOI: 10.1561/3300000030. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.pdf

[6] Lee, J. (2021a). Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. *Theory of Cryptography*, 1–34. https://doi.org/10.1007/978-3-030-90453-1_1 https://eprint.iacr.org/2020/1274.pdf

This document has been prepared by Space and Time Labs, Inc. (the “Company”). This document does not constitute or form part of and should not be construed as an offer to sell or issue or the solicitation of an offer to buy or acquire securities of the Company or any of its affiliates in any jurisdiction or as an inducement to enter into investment activity. No part of this document, nor the fact of its distribution, should form the basis of, or be relied on in connection with, any contract or commitment or investment decision whatsoever. No money, securities or other consideration is being solicited, and, if sent in response to this presentation or the information contained herein, will not be accepted. This document is not financial, legal, tax or other product advice.

This document contains certain forward-looking statements relating to future events or future predictions, which are generally identifiable by use of forward-looking terminology such as “believes”, “expects”, “may”, “will”, “should”, “plan”, “intend”, or “anticipates” or the negative thereof or other variations thereon or comparable terminology, or by discussion of strategy that involve risks and uncertainties. These forward-looking statements and the related information contained herein regarding matters that are not historical facts, are only predictions and estimates regarding future events and circumstances and involve known and unknown risks, uncertainties and other factors, that may cause the Company’s or its industry’s actual results, levels of activity, performance or achievements to be materially different from any future results, levels of activity, performance or achievements expressed or implied by such forward-looking statements. These statements and the related information are based on various assumptions by the Company which may not prove to be correct. The information contained herein has not been independently verified. No representation, warranty or undertaking, express or implied, is made as to, and no reliance should be placed on, the fairness, accuracy, completeness or correctness of the information or the opinions contained herein. None of the Company or any of its affiliates, advisors or representatives shall have any liability whatsoever (in negligence or otherwise) for any loss howsoever arising from any use of this document or its contents or otherwise arising in connection with the document.

The statements contained in this document speak only as at the date as of which they are made, and the Company expressly disclaims any obligation or undertaking to supplement, amend or disseminate any updates or revisions to any statements contained herein to reflect any change in events, conditions or circumstances on which any such statements are based. By preparing this presentation, none of the Company, its management, and their respective advisers undertakes any obligation to provide the recipient with access to any additional information or to update this presentation or any additional information or to correct any inaccuracies in any such information which may become apparent.

This document is highly confidential and being given solely for the information of the recipient and no portion hereof, may be shared, copied, reproduced or redistributed to any other person in any manner.