Space and Time:
The Verifiable Compute Layer for Web3

Scott Dykstra Jay White, PhD Nate Holiday Catherine Daly

David Alves Ian Joiner, PhD

January 2024
v1.0

Abstract

As Web3 rapidly matures, its path to mass adoption is hindered by three key deficiencies: poor user and developer experiences, the fragmentation of blockchain ecosystems (and associated zero knowledge toolkits), and the inherent limitations of smart contracts in their ability to process data. In order to realize the vision of Web3—a world where business logic and value exchange are underpinned not by trust, but by verifiability—each of these obstacles must be overcome. Space and Time has pioneered a breakthrough zero-knowledge circuit that speeds up time-to-value for dapp developers, improves end-user experience, acts as a single source of truth for the state of every popular blockchain, and serves as a coprocessor to supplement the limited storage and computational power of smart contracts using verifiable SQL queries. This protocol, "Proof of SQL," can be embedded in any SQL-compatible database solution, though Space and Time has delivered it to the market in the form of a decentralized data warehouse loaded with already-verified blockchain data indexed from popular chains. Our goal is to enable dapp developers with sophisticated, data-driven, cross-chain smart contracts—fulfilling the vision of Web3.

Table of Contents

1 The Problem: Untapped Potential of Web3

1.1 Smart Contracts Can't Access Critical Data

Smart contracts power the most valuable financial tools in Web3, including Uniswap, Aave, Compound, and Synthetix, which each represent billions of dollars in value secured by the Ethereum Virtual Machine (EVM). These decentralized finance (DeFi) protocols have reshaped markets, offering trustless, permissionless asset management. However, they lack dynamic data-driven sophistication. For instance, a new Aave borrower gets the same rate as a seasoned entity with onchain assets and history.

In traditional markets, derivatives like European or American-style options hold more capital than their underlying assets. They represent a huge portion of liquidity, with over 40 million options contracts traded daily[1]. However, these options are absent onchain due to the restrictions of smart contracts—they are difficult to deploy in a trustless manner due to their dependencies on data processing, although can be deployed offchain via centralized infra. Conversely, onchain perpetual futures have an impressive annual volume of $39 trillion. Their popularity in Web3 yet complete absence in traditional markets—even after years of Web3 adoption—is not because 'perps' are superior financial instruments, but rather because they execute simple logic in a smart contract without relying on any external data (besides token price feeds).

Such limitations in DeFi stem from the fundamental inability of smart contracts to query data, even from the logs of the blockchain they're deployed on. EVM contracts can only access wallet balances, information about the current transaction, some blockchain metadata, and their own limited internal memory.[A] It's important to understand that the data published by blockchains and the data accessible to smart contracts within the blockchain virtual machine are two completely different things. 'Full/archive' nodes of blockchains store a wealth of data across the full history of the chain, but none of this data is accessible directly to smart contracts. Using such nodes, an external offchain client can easily access 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. Nathan's wallet swapped 1000 USDC to 0.5ETH on 4/18/2023).

Yet, smart contracts cannot access this wealth of data generated by offchain indexing solutions (which use 'full/archive' blockchain nodes as a data source) as doing so introduces additional trust assumptions on those tamperable/unproven indexing solutions which are often centralized..

External tools such as decentralized oracle networks (DONs), especially Chainlink, bridged this gap initially, providing smart contracts with trust-minimized access to simple aggregated data points such as token prices. Yet, oracle networks can't process complex or sizable datasets (such as a query that aggregates transaction volumes in a liquidity pool since its inception), limiting their capability in DeFi's evolution. While oracle price feeds ushered in an initial wave of onchain financial instruments, sophisticated data-driven smart contracts will mark the next wave.

The growing popularity of multichain protocols compounds the need for more sophisticated contracts. Top decentralized applications (dapps), like the four aforementioned DeFi protocols, each operate on at least three different chains. With the proliferation of Layer 2 scaling solutions around the EVM ecosystem, a smart contract not only needs access to activity on its own chain but also other chains. Many dapps deployed cross-chain use a "hub" smart contract on a well-known chain like Ethereum, and "spoke" contracts on faster, cheaper chains, though these may be less decentralized and secure. The main challenge is: how can the hub monitor activities across all its spokes?

Similarly, in Web3 gaming, smart contracts offering in-game rewards can't easily integrate data about in-game actions or onchain non-fungible token (NFT) activities. This makes it difficult to reward players based on straightforward conditions, like winning a game, achieving over two hours of playtime, or leveling-up a weapon.

While some are creating sophisticated onchain protocols with the help of centralized systems offchain, this compromises the proof-driven model of Web3. Currently, there aren't trustless solutions that address the data processing limitations of smart contracts.

1.2 The Missing Query Coprocessor

In traditional SaaS, apps are powered by business logic that, at its most basic level, is simply a three-step process of retrieving a query result, executing an action based on that result, and updating the state of the system. For example, lenders query a potential borrower’s creditworthiness, run an algorithm to determine loan approval and rate, and then update account state. E-commerce platforms query for product availability, execute logic to reserve or purchase items, and then adjust inventory levels or order activity. Travel booking platforms query available flights/hotels/rentals, execute frontend logic promoting bookings options most relevant to the consumer, and then update availability and record user booking details. Social platforms query posts/content associated with users’ connections, execute an algorithm to rank the content in the user’s feed, and then update the backend state by recording content viewership/engagement, which then adjusts the algorithm for future content displays.

The glaring chasm in Web3 can be easily understood when this same framework is applied. The blockchain serves as a state management layer and smart contracts execute actions as arbitrary

code (business logic), but the query layer is still void—a gap that hinders the full realization of what dapps can achieve. Throughout this document, we will demonstrate how Space and Time fills this gap by serving as a "query coprocessor" which functionally sits next to major chains and supplements the limited compute (data processing) capacity of contracts on such chains.

1.3 Centralized = Unsecure

Though oracle networks enable smart contracts with trust-minimized access to simple data points, they do not have the ability to execute queries to process large volumes of data. The natural question that then follows is, why not simply connect a SQL database to a smart contract through an oracle solution? Databases that are fast enough to power real-time transactions, scalable enough to handle the entire state of the blockchain, and mature enough to manage complex queries are currently all centralized (tamperable black boxes), which makes them fundamentally incompatible with Web3.

First, centralization is antithetical to the ethos of Web3. Introducing a centralized component (tamperable "black-box" operated by a single entity) into the query layer infringes on the vision of a world without intermediaries, where data flows freely across a peer-to-peer network and can't be manipulated or censored. Connecting centralized databases to smart contracts—though many dapp developers are doing exactly this today—is a wholly regressive step for the ecosystem.

Perhaps more importantly, smart contracts secure value onchain, and therefore must be trustless end-to-end. A centralized database is a single point of failure that is vulnerable to tampering and manipulation, which directly imperils the assets locked in a smart contract and undermines its core promise of cryptographically guaranteed value exchange. The query layer of Web3 cannot be powered by offchain, centralized, trusted systems operated by entities with the ability to manipulate the data connected to smart contracts.

1.4 Current ZK Solutions and Their Limitations

Zero-knowledge (ZK) proofs have been heralded as the panacea for trustless data access, offering a way to prove the veracity of data without revealing the data itself. ZK technology offers offchain data computation with the same cryptographic integrity of the blockchain (as the proofs themselves are generally verified by a smart contract on a major chain), providing a way to supplement the limited data access and limited compute available to smart contracts. ZK-proofs replace the burdensome infrastructure overhead of offchain consensus, allowing a single node (a single machine, or a cluster of servers working together as a node) to serve as a coprocessor for an L1/L2 smart contract. However, current ZK solutions are immature, including the following limitations:

1. **Not Scalable:** The computational overhead of a ZK-proof makes it challenging to process data at the scale of an entire blockchain. They are not equipped with input data ingested (indexed) from major chains, and often generate proofs at dial-up speeds, with proof times ranging from 3 to 30 minutes. Without the capability to efficiently handle large-scale data, ZK solutions cannot fully meet the demands of sophisticated dapps in a growing and dynamic Web3 ecosystem. Quickly proving arbitrary computations over tiny datasets is one thing, but proving a complex SQL query against the multi-year transaction history of a set of liquidity pools or wallets on Ethereum is quite another.

2. **Fragmented:** The landscape of ZK solutions is also highly fragmented, with some solutions tailored for privacy enhancements, others optimized for scalability improvements, and still others designed to address specific use cases around transaction batching and rollup to L1 chains. Stringing these point-solutions together is time consuming, expensive, and error-prone.

3. **Unfamiliar User Experience**: Though developer tooling in the ZK space is expanding rapidly, developers face a challenge learning new languages (such as Cairo), new compute paradigms (such as writing code to run efficiently inside a "ZKVM"), and new infrastructure deployments (such as standing up hardware for running a "zkEVM rollup"). SQL on the other hand, has been prevalent for decades and offers a familiar UX.

4. **Not Built for Queries:** Most obviously, current ZK solutions do not satisfy the missing component of the Web3 stack: the query layer. Today, there is no ZK-proof that enables trustless queries/data processing for large datasets (besides Space and Time), and so the gap remains.

1.5 The Call for a Community Operated Data Warehouse

Over the past 15 years, the centralization of online user data has become a predominant theme. Large tech conglomerates have compartmentalized vast troves of user data (online activity, transactions, content, etc.) into centralized black-box repositories, often cloud data warehouses. These entities exercise full dominion over the accumulated information, productizing user interactions, behaviors, and personal preferences for profit. This commercialization occurred without clear consent or even the knowledge of the very users who generate this data—the internet's constituents. Such a system stands in stark contrast to the ethos of Web3, which envisions a decentralized internet where users reclaim control over their data, retain their right to privacy behind the pseudonym of a wallet address, and enjoy an inherent transparency in their digital interactions.

Consequently, we see a growing demand for community-operated data warehouses that would return data sovereignty to users. A community-driven approach would ensure that data isn't

manipulated or stored in hidden vaults accessible only to conglomerates but instead is managed by a community transparently determining the rules of access, use, and profit. Users could leverage their wallet (PKI infrastructure) as a signatory to determine how their online interactions are written to the data warehouse, persisted or deleted, and their data shared with third-party applications. Such an evolution toward self-custodied data would signify a transformative shift, highlighting the inherent power of collective governance and the potential of Web3 to restructure online hierarchies.

To make this demand a reality, the team at Space and Time realized zero knowledge-based verification of queries/data processing would be necessary to ensure that outsourced (community operated) data warehouse nodes in the decentralized network are not manipulating raw data or the query results being returned to clients such as smart contracts or financial institutions.[2] For self-custody of user data, row-based access control enabled via wallet-signed/wallet-governed transactions sent to the network would allow end-users to determine their level of privacy, which dapps can read their data, and how long their data persists. These end-user tools are built into the Space and Time network and will be detailed in further sections on network design below.

2 The Solution: Decentralized Data Warehouse with Proof of SQL

2.1 Introduction

Space and Time presents a solution to the query coprocessor requirements for smart contracts in the form of a high-performance, tamperproof, and community-operated data warehouse loaded with comprehensive data from major blockchains, verified using ZK-proofs. Space and Time introduces two novel ideas: a **decentralized data warehouse** and the **Proof of SQL protocol**, which together serve as the foundational ***Verifiable Compute Layer*** for Web3, allowing smart contracts to query onchain and offchain data and verify the result in a trustless manner. These novel ideas are deployed within the Space and Time network.

2.1.1 Decentralized Data Warehouse

The Space and Time data warehouse leverages hybrid transactional/analytical processing (HTAP), coupling an online analytical processing (OLAP) engine for complex queries with an in-memory cache for low-latency, online transactional processing (OLTP) workloads. We leverage our own native MVCC transaction system for inserting data along with both Apache Datafusion (low-latency OLAP) and Apache Spark (complex OLAP) query engines for powerful analytic reads. Hybrid query processing eliminates the need for multiple database point-solutions which plague traditional markets.[B]

Community-operated nodes in the decentralized network serve as a cost-effective hardware accelerator for both SQL queries and ZK-proofs. Any arbitrary ZK-SNARK or STARK proof circuits can be deployed on Space and Time nodes, including *Proof of SQL*, our own native ZK circuit for extreme performance when proving query operations. Any arbitrary circuit running on Space and Time nodes can be fed input data from our verifiable (ZK-compatible) blockchain indexing service (nodes called *Validators*), ensuring that the inputs and outputs of the circuit remain trustless end-to-end. This verifiable blockchain data is pre-loaded into the data warehouse in real-time from major chains such as Ethereum, Bitcoin, zkSync, Polygon, Avalanche, BNB, Sui, Sei, Base, etc. The actual hardware where the data warehouse query engines reside are split across the network's *Transaction nodes* and *Prover nodes* (more on this later).

2.1.2 Proof of SQL

Space and Time provides a groundbreaking ZK-proof that extends the security of Ethereum and other major L1s, L2s, or L3s/appchains to SQL databases. The protocol cryptographically guarantees to the client that both the underlying requested data (in many cases, indexed blockchain data—although offchain data can be proven as well) 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 at a scale sufficient for enterprise-grade applications. At the time of this writing, the Space and Time team has benchmarked proving times under 3 seconds for queries against a million-row table executed on a single NVIDIA GPU. In a later section of this document, we will deep dive into the design of the Proof of SQL protocol.

2.1.3 Coprocessing Unlocked for Smart Contracts: Examples

Ultimately, Space and Time's decentralized data warehouse running the Proof of SQL protocol specifically 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:

- **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 $LINK avg price and $ETH avg price 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 token 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 token 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 2 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.”

2.2 Design Goals

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

1. **End-to-End Trustless**: Leverage a zero-knowledge 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 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 who made the request.

3. **Familiar, Easy Developer Experience**: Provide a UX common in Web2 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.

5. **Low-Latency at Terabyte-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).

6. **Comprehensive Web3 Data Experience**: 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 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.

2.3 Solution Components

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

2.3.1 Provers (Proof of SQL Nodes)

Provers are GPU nodes within the decentralized network that execute queries and generate ZK-proofs using a CUDA-based framework for acceleration (more on this in Section 4 below). It's important to note that the *Prover nodes* are not limited to the Proof of SQL ZK circuit. We focused on future-proofing/unrestricting the overarching framework with a composable design such that, in the future, additional ZK circuits for non-SQL operations can be seamlessly integrated. Arbitrary ZK circuits (SNARKs and STARKs) can be deployed on a *Prover node* and accelerated by Space and Time's GPU framework.

2.3.2 Validators (Indexing Nodes)

A separate set of lightweight nodes called **Validators**, which do not require GPU hardware, handle indexing and prep cryptographic fingerprints of the data to be leveraged by the *Prover nodes*. During the indexing process, *Validator nodes* first request raw ledger data from remote procedure call (RPC) providers (blockchain full nodes, archive nodes, etc.) which is then decoded and transformed into the relational data model mentioned above.

A network of *Validators* work together to create and sign a cryptographic hash—a digital fingerprint of the indexed data called a "commitment"—for each column of every table. *Validators* send the indexed data to be prepped for storage by our *Transaction nodes* or by any third-party data warehousing technology leveraging the Proof of SQL protocol.[C] Simultaneously, the signed commitments are hashed to major chains to be later used during the onchain proof verification component of Proof of SQL. We designed Proof of SQL around an updateable commitment scheme, providing cryptographic assurance that each database row inserted during indexing is accurate and untampered. Validator signatures of the commitments are crucial to verifying every event within every transaction, across every block, spanning every major chain.

Commitment schemes similar to ours are used in many other zero-knowledge protocols, offering composability with additional proof circuits. This verifiable (ZK-compatible) indexed data produced by the *Validators* is offered by Space and Time as composable input data for other community-developed prover circuits beyond Proof of SQL, or other third-party query tools beyond Space and Time's native data warehouse solution.

2.3.3 Transaction Serving (Consensus Nodes)

Transaction nodes are multifaceted, handling a number of critical network requirements around BFT consensus, serialization/compression during data ingestion, and low-latency non-tamperproof query serving. These nodes accept commitments on indexed blockchain data provided by *Validators* and achieve consensus (threshold-signing the commitments, to be used downstream with Proof of SQL) while buffering the raw indexed blockchain data for storage compression. These nodes also use their consensus service to verify transactions and authenticate clients in a decentralized manner.

Ingested data is handled with MVCC for consistency, and buffered into Apache Arrow record batches in memory before being compressed to parquet files by the *Transaction nodes*, in preparation for efficient storage. These nodes serve fast, non-ZK-proven queries using the Apache Datafusion library. Another included service of these nodes are comprehensive REST APIs for common blockchain data requests such as token transfers, balances, transactions receipts/history, block metadata, gas oracles, price feeds, etc.

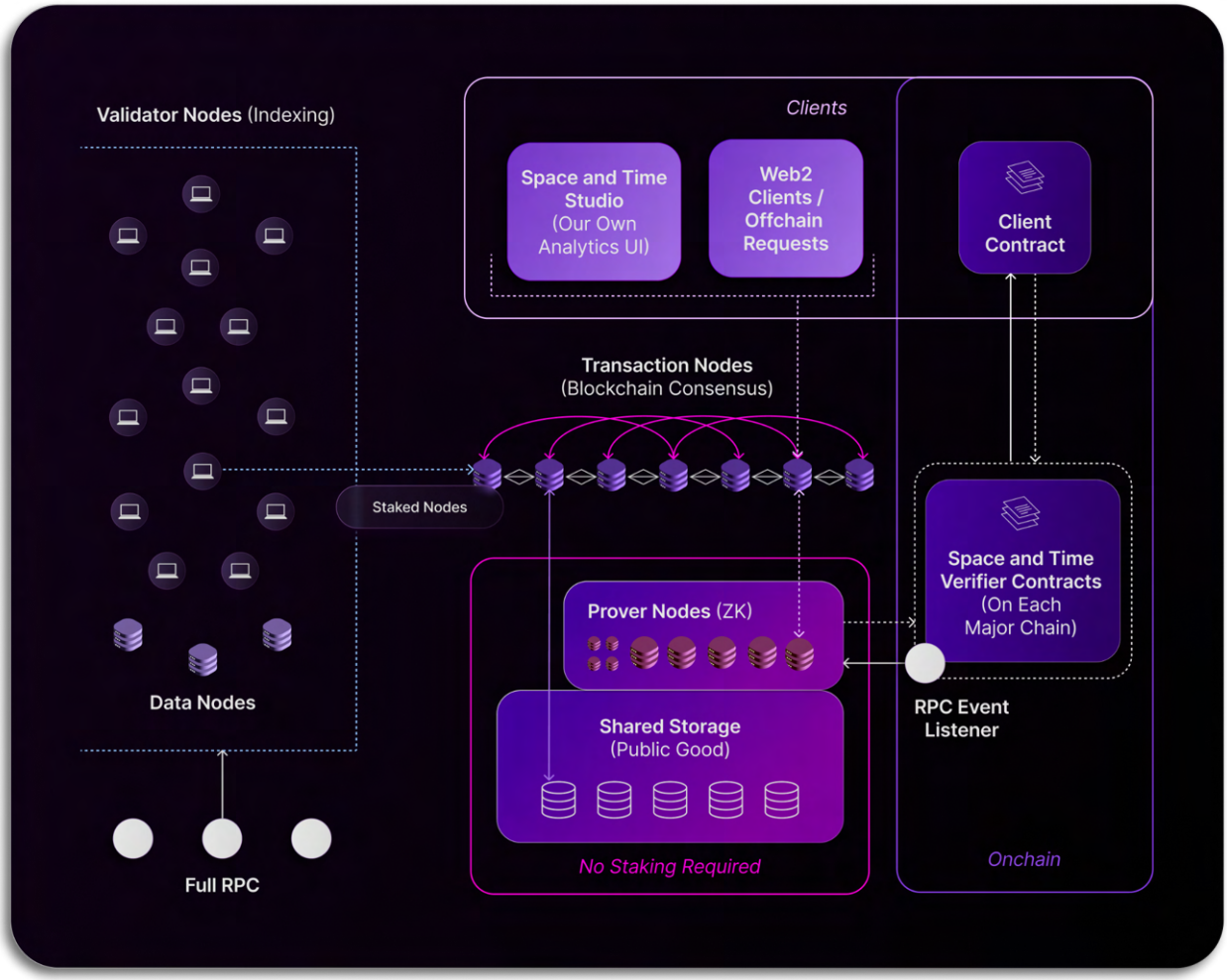

Figure 0: Space and Time Primary Components

2.3.4 Onchain 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):** 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 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.
- **Community-developed ZK Services (Offchain):** Open-source ZK circuits provided by the community, like RISC Zero or Axiom, could be run on the Space and Time *Prover nodes* (GPUs) for proving arbitrary computations beyond SQL against our verifiable (ZK-compatible) indexed blockchain data.

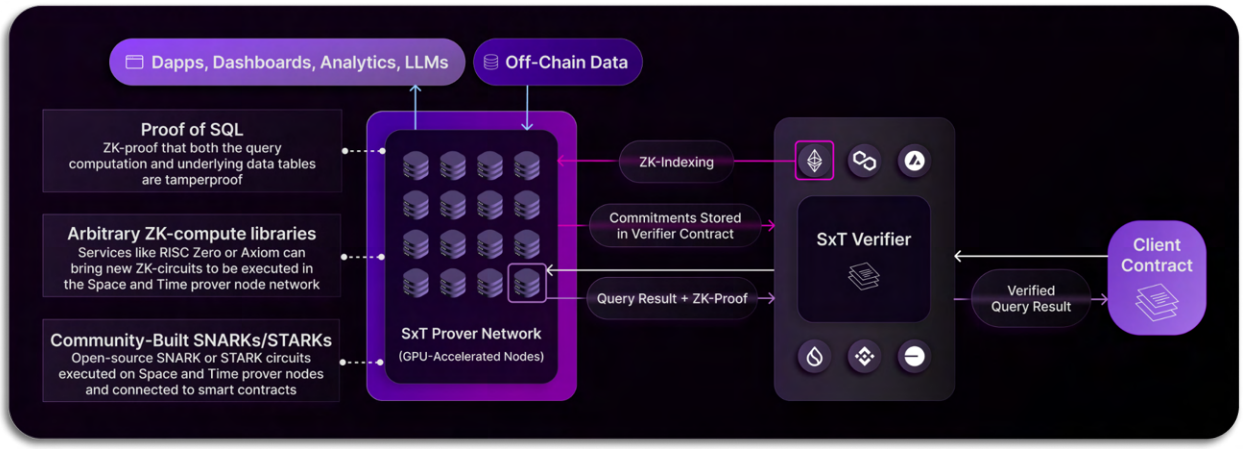

Figure 1: Space and Time Verifiable Compute Layer Architecture

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.

2.4 Ingestion Layer with Verifiable Indexing

Data ingestion into Space and Time leverages BFT consensus when the ingested data must be cryptographically proven/verified prior to use for queries. 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 twelve different stock market data APIs simultaneously. Space and Time *Transaction node*s can achieve consensus on the price of Apple stock across these twelve different data providers, for example, and only insert to *shared storage* (a public good within Space and Time) the threshold-signed, consensus-approved price of Apple stock at the current moment. To add more security, we developed a library which source data providers can leverage to cryptographically sign data at the origin before ingestion, and our *Provers* integrate these signatures when proving against such underlying data.

Similarly, with indexed blockchain data that our network captures, we are onboarding a massive number of *Validator nodes* (indexers) to redundantly process the entire state of popular chains like Ethereum, Bitcoin, Polygon, zkSync etc. Each *Validator* runs a *light client* 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. *Validators* submit those commitments for consensus-approval by the *Transaction nodes*, which then buffer the raw indexed data for efficient storage. This process is aware of block reorgs, and offers developers endpoints for both fresh, non-finalized blockchain data as well as probabilistic-finalized blockchain data after a few minutes, depending on the chain.

Through this process, Space and Time is commoditizing indexed blockchain data, and offering it essentially free through our network (we only charge for compute costs of queries executed in Space and Time, not the underlying data), as well as delivering these datasets as input data to other third-party zero knowledge solutions and third-party data warehouse solutions.

2.5 Community-Operated Data Warehouse Design

The decentralized data warehouse 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 decentralized and community-owned. 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. To align with the decentralized ethos of Web3, the nodes that make up this data warehousing platform must be community owned and operated, enabling:

- **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 data warehouse 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 we integrate more chains and more participants that 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 data warehouse vendors.

2.6 Infinite Tables join Web2 with Web3

Space and Time capitalizes on the moats of both data warehousing and Web3 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.

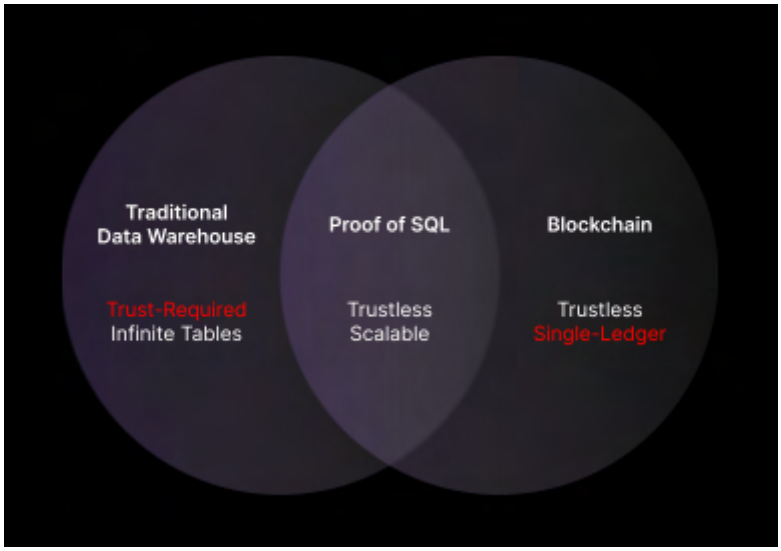

Figure 2: Marrying Scale and Provability with Proof of SQL

On one hand, smart contracts can transact with our *Verifier* contracts onchain, incurring fees for data retrieval and verification. Simultaneously, enterprises can access our vast blockchain data (or their own ingested data to an infinite number of tables/ledgers, similar to traditional data warehousing), paying in fiat, which is then converted to be used to enhance our native tokens' utility. This dual model ensures consistent demand while countering the issues of centralized systems lacking trust, especially for high-value transactions. Node operators, following specific economic guidelines, guarantee up-to-date, accurate, and immediate data even for non-ZK-proven query execution. This design minimizes vulnerabilities while remaining practical (particularly in terms of cost structure) for:

- **Offchain Analytics:** The Space and Time solution is suitable as both a decentralized backend for dapps and as an analytics platform with comprehensive blockchain data. Direct access to Space and Time data warehouse nodes offchain can be achieved with REST APIs and JDBC connection proxies deployed on 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.

2.7 Verification in Smart Contracts vs. Oracle Networks

The Space and Time *Verifier* smart contracts deployed to a variety of popular EVM chains can natively verify the ZK-proofs submitted by *Prover nodes,* ensuring data integrity and authenticity before relaying verified query results back to client contracts which initiated the requests. We are eagerly awaiting deployment of the BLS 12-381 curve pre-compile implemented (but not yet deployed) via EIP-2537. Until then, gas required for proof verification on Ethereum mainnet will present a potential obstacle to adoption. To combat this, we are also deploying our ZK-Verifier as a service on Chainlink’s DON in order to facilitate low-cost, consensus-based, offchain verification for developers already familiar with Chainlink oracle jobs.

Here’s how it works: Client contracts send a payment along with job spec to Chainlink contracts, and then Chainlink oracle nodes request the query result and associated proof from Space and Time endpoints. Once a query result and associated proof are submitted back to the DON, then

multiple oracle nodes in the DON redundantly verify the proof offchain before relaying the verified query result back to the requesting client contract. The extreme performance of Proof of SQL, accelerated by NVIDIA GPUs, ensures that most requests can be returned within Chainlink's required response latency for offchain reporting (consensus).

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 zero-knowledge 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 at a scale sufficient for enterprise-grade applications.

We have designed this protocol with several goals in mind. First, the protocol needs to be extremely fast, not just for the verifying party, but for the round-trip execution time. This requires a design that is built from the ground up. Second, we have built this to be extremely developer-friendly: SQL is the most popular data query language and provides a familiar UX for anyone trying to start building a data-forward application. Finally, this protocol needs to facilitate complex data processing rather than simply running serial compute or blob data retrieval.

There are two parties involved in the protocol: the client sending the query, and the database service returning the result. We will name these two parties the *Verifier* and the *Prover*, respectively. In practice, the *Verifier* does not need to be the one actually sending the query, instead the *Verifier* can be any trusted party, such as a smart contract. This type of protocol 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).

A key architectural concept worth highlighting here is the idea of a digest, or commitment. To prevent the data in a table from being modified, we ensure that a commitment of that data is kept. The *Verifier* must have trustless access to this commitment to be able to detect any tampering. This commitment can be thought of as a type of digital fingerprint: a lightweight digest of the data in the table.

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, and Section 3.7 describes the commitment scheme that we use.

3.1.1 Data Ingestion

There are two types of interactions between the *Verifier* and the *Prover*. The first type of interaction is data ingestion and the second is a query request. Both of these are initiated by the *Verifier*. This verifier could be a smart contract, a decentralized application, or any party interested in consuming data. In practice, the actual data source does not need to be the *Verifier* themselves, 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 ZK-compatible blockchain indexing solution, or from the *Verifiers* themselves. However, for the sake of this architecture discussion, we will combine the roles of the data source with those of the *Verifier*. During data ingestion, the *Verifier* wishes to send data to the *Prover* so that it can later query that data. However, to ensure that the data will be untampered, the *Verifier* needs to compute commitments of this data—a core concept aforementioned. The *Verifier* has access to the commitment of data that is in a table, while the *Prover* holds onto the actual data. Without this commitment, 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 *Verifier* so that the *Verifier* 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 *Verifier* can route the data to the database for storage. The *Verifier* stores this commitment for later usage. (Note: to decrease latency, the data is physically routed immediately, but the *Verifier* only "locks in" the data after creating this commitment).

An important design constraint is that the commitment must be *updatable*. (See section 4.7) In other words, suppose that the *Verifier* already holds the commitment to a specific table, but new data is to be ingested and appended to the table. To do this efficiently, the *Verifier* 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 *Verifier* must be able to do this without access to the old existing data.

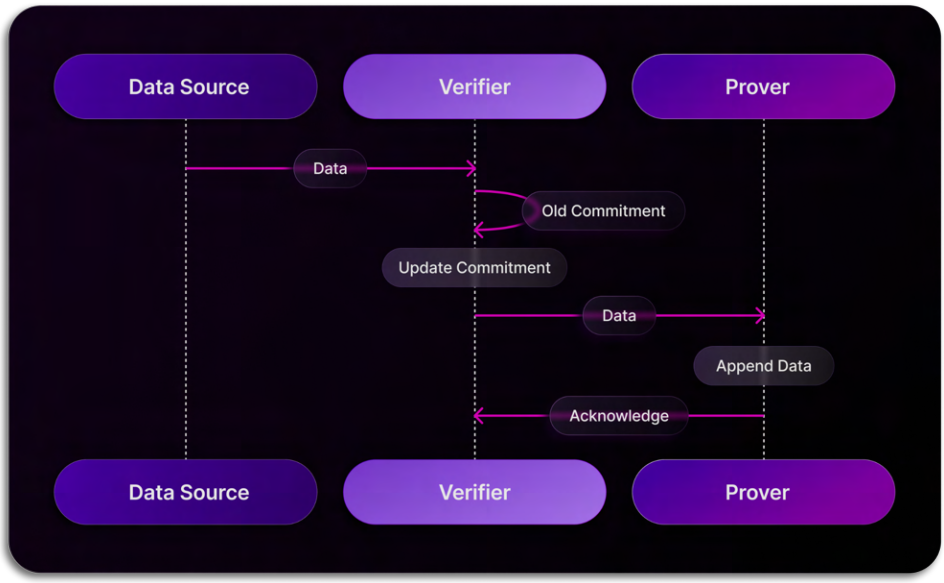

Figure 3: Sequence Diagram of Data Ingestion

3.1.2 Query Request

The second type of interaction between the *Verifier* and the *Prover* is the query request, where 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 they are holding onto, or have trustless access to, a 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 (4) is dedicated to describing how this proof is constructed and verified. See the next subsection (4.1.3) for an overview.

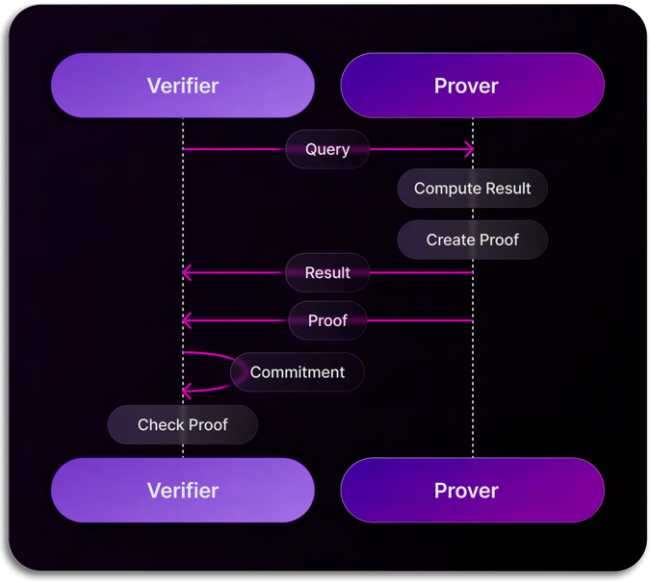

Figure 4: Sequence Diagram of Query Request from Verifier to Prover

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 4.2)
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 4.3)
3. **Relation Prover:** This is the step in which the bulk of the proof is actually generated. We utilize the multilinear sum-check protocol here. (Section 4.5)
4. **Commitment Evaluator.** The last step of the sum-check protocol requires a commitment opening. (Section 4.6)

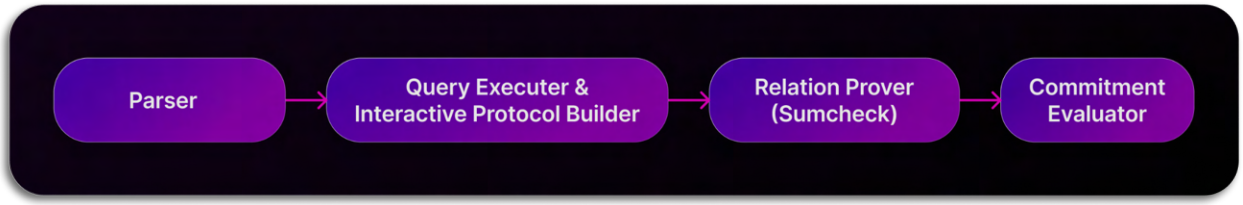

Figure 5: Proof Protocol Overview Diagram

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 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 abstract syntax tree (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
```

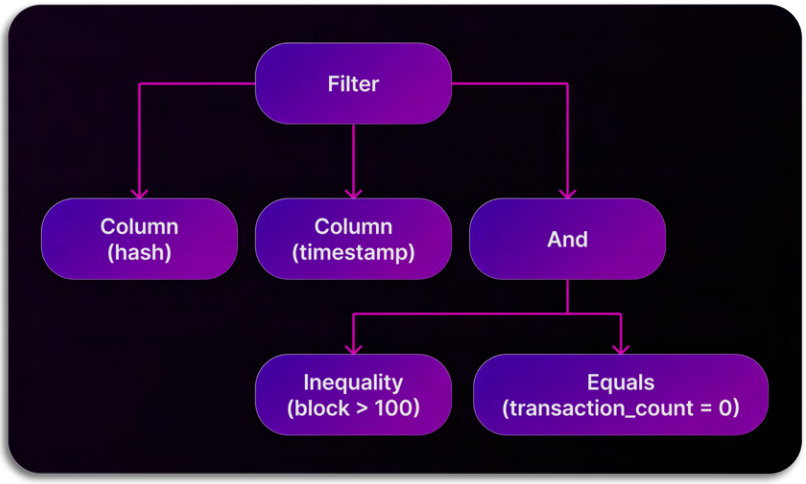

Figure 6: Diagram of AST Nodes for Example Query

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 where 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

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 the ones needed just 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 **(transaction_count = 0)**	**Inequality** **(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.

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 portion, and we push this computation to GPUs. For the most part, the commitment scheme that we use can be treated as a "black box" compared with the rest of the protocol. See section 4.7 for more details on the exact commitment scheme used.

Some of the nodes in the AST required 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 Shamit heuristic. This is done before passing over the AST nodes, meaning that the *Verifier* only needs 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 $f_A \in \mathbb{F}[x_0, \ldots, x_{\nu-1}]$ such that $f_A(x_0, \ldots, x_{\nu-1}) = A\left[\sum_{i=0}^{\nu} x_i 2^i\right]$ for $(x_0, \ldots, x_{\nu-1}) \in \{0,1\}^\nu$. 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

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

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_ℓ denote the column that is 1 for the first ℓ 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 (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 β:

$\sum A \cdot Q - S \cdot R = 0$, $Q \cdot (F + \beta) = I_n$, 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 β, while W and S do not. If β 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 β allowing a malicious *Prover* to provide the wrong result. In other words, β must be a function of W and S.

Suppose, for example, that β 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: β.
- 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: 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 $\rho_0, \ldots, \rho_{\nu-1}, \alpha_0, \ldots, \alpha_{\ell-1}, \beta_0, \ldots, \beta_{s-1}$. These 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 a thorough 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 we use is important. We are using the Dory commitment scheme[6]. There are several motivating factors that drive this choice. We will list them (roughly) in order of importance, all of which Dory satisfies.

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 to update commitments to the data. Any homomorphic commitment scheme has the property. Dory is a homomorphic commitment scheme. 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 verifier runs in $O(\log n)$ time.
3. **Transparency:** While this is a beneficial property for any protocol, it is particularly important for Proof of SQL because of large data volumes that exist. Without a transparent setup, there would need to be a trusted setup, 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.
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.

Conceptually, we can describe the Dory commitment scheme as a commitment scheme to a $2^m \times 2^m$ matrix. In the simplest form, we fix the height of the matrix to be N and allow it to grow arbitrarily wide, padded with 0s. In other words, for $A \in \mathbb{F}^n$ we let $m = \max(\log_2(N), \lceil \log_2(n)/2 \rceil)$ and define $M(A) \in \mathbb{F}^{2^m \times 2^m}$ to be the matrix such that $M(A)[i, j] = A[i + j \cdot N]$ where A is defined and 0 elsewhere.

For example, setting $N = 2^2$, and $A = (100, 101, 102, 103, 104)$, we have

$$M(A) = \begin{pmatrix} 100 & 102 & 104 & 0 \\ 101 & 103 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

With this notation, we can define the commitment to A to be $Commit(A) = DoryCommit(M(A))$ where $DoryCommit$ is the commitment scheme described in the Dory paper[6]. We set $N := 2^{15}$ as the default because it is small enough to add negligible time to the *Prover* but large enough to support large tables with minimal overhead.

4 Use Cases for the Next-Generation of Web3

4.1 Building a Flexible ZK-Rollup/L2

Space and Time's novel zero knowledge protocol, Proof of SQL, can be used to build trustless, bridgeless, gas-minimized microtransactions maintained on a flexible number of ledgers and configurations.

Excessive gas costs (onchain transaction fees) are a major limitation to Ethereum adoption. Even after *the Merge*, it remains prohibitively expensive to settle a complex transaction or large volume of transactions on Ethereum. ZK-rollups, in response, offer a trustless and secure "side-ledger" of affordable quick-settlement transactions to be batched and later committed to the Ethereum mainnet. ZK technology effectively allows users to perform multiple transactions offchain, or on a Layer 2 (L2) chain, and then post a single proof to Ethereum.

Proof of SQL enables an append-only database to function as a trustless offchain ledger for SQL operations that can be batched, proven with zero knowledge cryptography, and settled on Ethereum. Essentially, developers can easily build their own ZK-rollup on top of Space and Time. This allows for unparalleled performance/scale, ease of development with familiar SQL tools, and flexibility regarding the number of ledgers that can be maintained and rolled-up to the main chain (usually an L1, but can also roll up to an L2 or appchain). This solution is distinct from other ZK-rollup solutions in three key ways:

1. **Instant 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. Because of the inherent structure of an append-only database, transaction finality is instant upon being added to the ledger.

2. **Data Processing and Scalability:** While current ZK-rollups focus on transaction aggregation, Proof of SQL supports data processing operations (selections, projections, joins, and aggregations), which expands the complexity of what can be settled onchain. Additionally, Proof of SQL can scale to support infinite tables, rather than just a single ledger.

3. **Bridgeless and Gas-minimized:** Unlike other L2 solutions, which require bridges to move assets between mainnet and L2, Proof of SQL facilitates inter-chain/cross-chain communications seamlessly. Furthermore, since offchain SQL operations are executed without incurring onchain gas fees, smart contracts can access scalable compute without incurring more cost. End-users do not need to first bridge their tokens from a main chain to wrapped tokens on another chain before using dapps. Users pay gas only during deposit and withdrawal; they execute a gas-incurring transaction to lock up cryptocurrency in a 'escrow' contract (enabled with Proof of SQL) on the main chain and are then free to execute cheap transactions in dapps built on Space and Time, later performing another gas-incurring transaction when withdrawing their new balance back from the escrow contract (which first double-checks the user's balance by trustlessly querying the 'account balances' table/ledger within Space and Time).

4. **Familiar Tools:** Space and Time is accessed with a widely used and easy to understand tool: SQL. The ubiquity of SQL ensures a simple developer experience and seamless integration with existing systems.

Note that this solution requires additional business logic which we've deployed on Space and Time's *Transaction nodes* to verify transactions—such as verifying that the user wallet has signed a message to transfer value from their wallet, or to lock up capital in escrow—in a decentralized way, in order to prevent double-spend and ensure Sybil-resistant transactional compliance.[D]

4.2 Secure Bridges and Multichain Data Backends

Today, blockchain ecosystems are deeply fragmented. New, forward-thinking blockchain architectures are being delivered to the market almost monthly, but are disconnected and non-interoperable with other popular chains. Though most popular dapps are deployed across multiple chains, communicating the state of any one deployment to another is complex and expensive to do, since each blockchain operates in relative isolation. To implement a cross-chain protocol, developers have to piece together solutions for retrieving, integrating, and messaging data across disparate chains, manage multiple data structures, and deal with different consensus mechanisms. Managing cross-chain dapps is hard enough, but the infra costs and engineering overhead required to do so is growing—limiting dapp developers.

Space and Time provides a single, cryptographically-guaranteed source of truth for the real-time state of many popular blockchains. Data from each chain is verifiably ingested (via our ZK-compatible blockchain indexing) immediately after block finality and stored in a relational state in the data warehouse. The Uniswap contract on Ethereum, for example, can query the state of Uniswap on BNB Chain and get back a real-time result guaranteed by Proof of SQL. The solution provides a streamlined, bridgeless way for developers to access, manage, and integrate data from multiple chains into their dapp. Space and Time can serve as the decentralized data backend for dapps, removing the need for both centralized databases and centralized API servers. The Space and Time API gateway allows developers to build data-driven dapps directly on the platform, with Space and Time itself as the backend servers—no centralized infrastructure required. This is evidenced by the Space and Time Studio, as an example—our own analytics dapp which consists only of frontend code and a smart contract (the Space and Time platform is the entire backend).

Furthermore, Space and Time can radically transform the reliability of cross-chain bridges. When bridging contracts need to facilitate a transaction between two chains, the fundamental challenge is to ensure the authenticity of the locked-up assets on the source chain before minting or releasing equivalent “wrapped” assets on the target chain. By querying ZK-proven indexed data (essentially, an “account balances” table/ledger in the data warehouse), these bridge smart contracts can "double-check" the balances of user accounts that have locked up capital in escrow on chain 'A'. Only once this verification process confirms the authenticity of the locked assets in zero-knowledge, the bridge can proceed to release the corresponding wrapped tokens on chain 'B'. This mechanism ensures a higher level of security in cross-chain operations, minimizing the risk of double-spending or fraudulent asset creation. Moreover, the inherent privacy features of zero-knowledge proofs ensure that while the veracity of the transaction is confirmed, sensitive details remain concealed.

4.3 Dapp Backend (Decentralized)

Traditional applications stacks have a **frontend** or client-side, which is responsible for delivering the user interface, and a **backend** or server-side, which handles which stores and retrieves information with a database and executes business logic or actions on the retrieved data. Dapps are designed to operate on a parallel stack: a frontend that delivers the user interface, and a smart contract that executes logic on the blockchain (which essentially serves as a lightweight database). However, because smart contracts are limited in computational capability to execute complex business logic—and worse, blockchains generally have extremely limited storage—dapp developers generally deploy centralized backends with centralized databases alongside their smart contract. These centralized components introduce attack vectors for tampering as well as single points of failure to the stack, undermining the core value proposition of a ‘decentralized’ application.

Space and Time is both a decentralized database and a decentralized, serverless backend for applications that enables developers to build sophisticated, data-driven dapps without sacrificing decentralization. *Transaction nodes* in the Space and Time network provide APIs for common SQL operations, common requests for onchain datasets, and decentralized authentication with role-based authorization. Developers can read/write application-generated data directly with Space and Time, backend business logic can be delivered via python running on the Space and Time nodes, and a dapp's associated smart contracts emit events onchain that Space and Time automatically indexes and makes available to the dapp frontend.

4.4 Data-Driven Lending

Lending is one of the most intuitive use cases for DeFi, but lending protocols are still rudimentary compared to traditional lending systems. Borrowing from Aave, for example, is very straightforward: a borrower takes out a loan from a liquidity pool, putting their own assets up as collateral, and is charged an interest rate based on data about the asset and the liquidity pool. A new Web3 user receives the same lending rate as a seasoned 'degen' with deep onchain transaction history and multiple loans already paid off. Though borrowing history is transparently accessible onchain, a smart contract has no way to aggregate or query it—even that of its own chain.

Space and Time allows a smart contract to query ZK-proven aggregations of both onchain and offchain loan history associated with a borrower in real time. These aggregations essentially function as a credit score, allowing credible borrowers to receive better rates on onchain loans and enabling higher risk-adjusted returns for lending protocols. Furthermore, a lending contract could query Space and Time to determine dynamic loan liquidation preferences based on past repayment behaviors onchain.

4.5 Cross-Chain Financial Instruments

Financial instruments, such as derivatives, are inherently multi-faceted and depend on a wide range of data sources for accurate valuation. As Web3 expands and new L1 ecosystems emerge, the ability of a derivatives smart contract, for example, to access and quote token prices from multiple chains becomes more necessary. But today, for an Ethereum protocol to access token data on Avalanche, for example, it has to bridge data, verify its authenticity, and deal with inherent latency issues.

Proof of SQL streamlines this process, allowing real-time token prices on one chain to be cryptographically verified and made available to protocols on another. Developers can craft complex derivatives without grappling with the cost and efficiency challenges of cross-chain data communication. A smart contract can directly query Space and Time's comprehensive

indexed blockchain data with ZK-proof of its accuracy, all within just a few seconds of block finality. This not only reduces the complexity of the derivative's creation but also ensures that the derivative's value remains transparent, trustless, and up-to-date in real time.

4.6 Gaming Rewards

Rewarding players based on their achievements, skills, and in-game actions has become a staple of gaming, but the integration of these rewards into onchain systems has been constrained by the limitations of smart contracts. Smart contracts can only facilitate very simple reward logic, such as “if player X wins this round, mint them an NFT,” and don’t have the native ability to integrate data around what’s happening in the game.

Space and Time offers a solution by enabling offchain game telemetry to be incorporated in onchain reward systems. Game developers can store detailed player metrics in the decentralized data warehouse, from simple achievements to complex behavioral patterns. When it's time to reward a player, a smart contract can query the data, aggregate it through more intricate logic, and subsequently distribute rewards.

For instance, in a multiplayer online battle arena game, instead of simply rewarding a player for winning a match, the contract can access analyses of team cooperation, strategy implementation, and individual skill contributions. This level of depth, combined with the verifiability of Proof of SQL, ensures that players are rewarded not just for surface-level achievements, but for the richness of their gameplay experience. Accolades tracked in-game could include playtime, weapons used, player rank/leaderboards, objectives completed, NFTs used, etc. By making rewards more nuanced and data-driven, Space and Time amplifies player engagement and expands the in-game economy.

4.7 Web3 Social Apps

Developers building Web3-native applications in the social networking space currently face four challenges, listed below. Space and Time offers a decentralized data warehouse that essentially solves all four:

1) **Scaling to Billions of User Interactions:** Blockchains not only have inherent throughput limitations but also have limited onchain storage. Social apps often generate terabyte or even petabyte-scale annual data volumes. Centralized offchain data processing solutions, however, do not offer the trustless guarantees or proof of storage/data replication that Web3-native apps require.

2) **Securely Rewarding Content Creators Onchain:** Along with scaling the storage volumes and high-throughput required for billions of interactions, social apps often must

reward content creators for the attention/viewership attributed to their posted content. Tracking engagement and periodically rolling up the required payouts to all content creators onchain is almost impossible without leveraging centralized, tamperable data processing solutions offchain.

3) **Bootstrapping a Significant Amount of User-Created Content:** Web3 social apps are seeking composable, user-owned, decentralized social graph solutions such as *Lens* to bootstrap posts and content. The chicken-and-egg problem of social apps (app isn't engaging without a large amount of user content to discover, and engaging user content won't be created without a large amount of users) can be overcome when a post submitted by a user can show up in the content feeds of many different apps. However, onchain solutions are far too limited in storage capacity to process all the posts, and IPFS-like data storage solutions often have poor performance (high latency for real-time interactions).

4) **Enabling secure, private interactions:** Blockchain data is inherently public, so allowing for private messaging/interactions is challenging when using IPFS-like data storage solutions or the blockchain itself as a backend. Conversely, using centralized backends for private messaging is problematic in Web3-native use cases that require trustless verification of the authenticity of private messages without revealing the message itself.

Scaling social interactions with offchain storage can be a complex undertaking. Not all data in a social app needs the same level of security and immutability as others. While essential data like content creator rewards or user balances might need the integrity of onchain storage, other data like comments, likes, or shares might not. By offloading such data to Space and Time, for indexed access and fast retrievals to quickly fetch posts, comments, or user profiles against commodity storage.

Similar to rewarding gamers for in-game achievements, rewarding content creators can be just as challenging. Certain critical interactions can be processed offchain (such as number of viewers or minutes of attention on a piece of creator content) but delivered to a smart contract for accurate/untampered and transparent/auditable reward payouts onchain. Other actions like staking or content upvoting that involve token transfers, or any behavior that might affect a user's financial position should be kept onchain to maintain trust.

Leveraging composable datasets in decentralized databases offers a transformative approach to Web3 social interactions. By allowing multiple social applications to interface with a shared, composable "content posts" table/dataset, a unified social graph emerges, transcending individual app boundaries. This interconnected data structure enables third-party Web3 apps to both read from and, given proper permissions, write to this community-operated table. The result

is seamless content flow across platforms, enhancing the overall user experience and removing the need for each app to bootstrap its own content.

Finally, the zero-knowledge compute capabilities within Space and Time establish a robust privacy layer for Web3 social applications. When employed for private messaging or content sharing, ZK-proofs allow users to verify the authenticity of a message without revealing its actual content. Thus, within the social app framework, a sender can prove they've sent a legitimate message, and the recipient can validate this, all without exposing the message's contents to external parties or even the underlying infrastructure. This ensures that private conversations remain confidential, fostering trust and security in decentralized social platforms, while maintaining the integrity and transparency inherent to blockchain-based systems.

4.8 Settlement Systems and Third-Party Auditing

Over the last decade, banks and other financial enterprises have adopted consortium ledgers as a trusted way to share data with other institutions. A consortium ledger functions essentially like a blockchain that is privatized and controlled by a consortium of enterprises instead of a decentralized network.

However, much like a blockchain, this ledger isn't built to function as a transactional database. Instead, a bank deploys a SQL database to manage its order books, then marks its P&L to market once a day to be shared with the other banks in the consortium. Each consortium member can trust that once this bank marks to market, the data on the ledger can't be manipulated. Still, there's no way to guarantee that the granular transactions used to calculate P&L are real and verifiable.

The only way to fully secure trust in offchain data is to connect it to a verifiable database. Each bank needs to be able to verify that no other bank manipulated its P&L (or the data used to calculate P&L) before adding it to a consortium ledger. If Space and Time is deployed as the SQL database used to store and aggregate transactions, then no bank has to trust that each other member is calculating its P&L correctly. It's cryptographically guaranteed by Proof of SQL.

Similarly, Space and Time can be leveraged as a tamperproof audit trail of SQL operations, which makes it easier for a business to prove compliance with regulatory requirements such as the General Data Protection Regulation (GDPR)[4], for example. Every access, change, or deletion of personal information protected under the regulation can be recorded in Space and Time so that, in the event of an audit, the immutable record serves as unequivocal evidence of the business's compliance. Furthermore, as a decentralized data warehouse, Space and Time provides the distributed framework to allow this company to deploy a network of data warehouses entirely on EU-based infrastructure. Existing disparate data stores are easily

connected to Space and Time, where the company can run analytics against the data without it ever leaving the EU.

Financial institutions that leverage Space and Time as their transactional database system are able to execute a tamperproof, low-latency order book. Third-party financial institutions (often part of a consortium) can read and write to the order book and cryptographically verify that it hasn't been manipulated/tampered by any of the participating entities. Order books can be settled periodically on a public or private chain via Proof of SQL, although this is not necessary since Space and Time itself offers the same cryptographic assurances of popular chains as a system of record.

4.9 Custodial Digital Assets

One of the most widely recognized barriers to the adoption of Web3 is a convoluted and unfamiliar user experience. Web3 games, for example, are viewed as their own distinct subcategory of gaming, primarily because of the complexity around setting up a wallet and minting NFTs.

Space and Time allows a game, a centralized exchange, or any other platform to hold custody of and manage end users' assets in a trustless way. The record of which user owns which NFT is stored securely and transparently in the tamperproof Space and Time decentralized data warehouse. If the user decides they're ready to take custody, a smart contract can query the database to mint the right assets to the right wallet, along with a proof that the record wasn't manipulated.

Furthermore, a centralized/custodial exchange can use Space and Time to prove that it actually owns the assets it claims to on behalf of the user. Beyond proving reserves, Space and Time can be leveraged to create a fully automated and trustless exchange process in which a smart contract queries anonymous deposit data about a client to automatically execute a purchase on their behalf.

Space and Time brings the transparency, security, and trustlessness of DeFi to CeFi, blurring the line between the two. Given the growing public distrust in centralized exchanges, the ability to prove reserves, hold tamperproof custody of user assets, and automate a process that prevents bad internal actors from mishandling custodial funds is paramount to the survival of CeFi platforms.

4.10 Tokenization of Real-World Assets and Dynamic NFTs

The tokenization of real-world assets, such as real estate, event tickets, or collectibles, have emerged as one of the most promising and utilitarian use cases for blockchain technology. In

many cases, the metadata associated with these assets is dynamic in nature. For example, a tokenized concert ticket has static metadata associated with the venue, artist, and seat number, but also data that changes and fluctuates, such as the price of the ticket as availability decreases.

Space and Time not only provides trustless and decentralized offchain storage for metadata that is too high-volume to be stored onchain, it also allows a smart contract to query the data in real time to update the asset onchain accordingly, in a way that's provable to the end user. In the previous example, the ticket issuer stores the ticket's metadata in Space and Time, along with the parameters by which the price might increase or decrease. When a concert goer purchases the ticket, the smart contract queries Space and Time to get the real-time price information and executes accordingly.

This model both allows data to be stored more efficiently, with only the most important information around the asset aggregated and published onchain, and provides more transparency to the end user around how the asset is dynamically priced. In the ticketing industry specifically, users are demanding price transparency more than ever, and Space and Time lays the framework for blockchain technology to provide it.

4.11 Transaction Security and Wallet Whitelists

Ensuring the security and safety of onchain transactions prior to execution is critical to the growth and prosperity of the Web3 ecosystem. There is a significant market for transactional/smart contract security protocols for which Space and Time can serve as the verifiable data/data processing backend.

Consider a security platform that leverages offchain machine learning to analyze a wallet's activity and determine whether it's safe to transact with. Not only does Space and Time allow the platform to access verifiable onchain wallet history, it can also store the platform's outputs (in this case, wallet addresses and associated risk scores) in a tamperproof (and, if necessary, transparent) table. To validate that a transaction is safe, the platform queries Space and Time to get the verifiable, real-time risk score for the receiving wallet.

Similarly, Space and Time can store compliance data (for example, a list of OFAC-banned wallets, or a whitelist of KYCed wallets) in a transparent and tamperproof way. A smart contract can verifiably query the table to consult the list and ensure compliance before executing a transaction. Any client can create a table of whitelisted/blacklisted wallets in Space and Time and populate it with offchain data or data collected by their smart contracts, and query that table directly from a smart contract with ZK-proof that the data wasn't tampered.

4.12 Liquidity Pool Rebalancing based on Market Conditions

Liquidity providers for protocols such as *Uniswap v4* are eager for opportunities to passively earn greater rewards from the pools they provide capital to, yet often face impermanent loss or adversarial token balances within the pool due to volatile market conditions. This results in liquidity providers receiving fewer tokens than they would have if they had simply held onto the tokens rather than providing capital to a pool. Thus, dynamic pool rebalancing executed by external actors helps:

- **Maintain Asset Ratios:** Many AMM-based DeFi pools are designed to maintain a specific ratio of assets (e.g. 50:50). During market volatility, when one token's value changes significantly relative to another, the pool deviates from this intended ratio. Rebalancing ensures that the pool stays close to its target ratios.
- **Reduce Impermanent Loss and Increase Yields:** As mentioned earlier, impermanent loss is a risk faced by liquidity providers in AMMs. By periodically rebalancing pools, developers can help mitigate the effects of impermanent loss and make providing liquidity more attractive to users. Rebalancing can help ensure that the capital is always placed in the highest-yielding pool configuration, maximizing returns for participants.
- **Enhance Security:** Pools that grow too large or imbalanced might become lucrative targets for arbitrageurs or potential exploiters, especially if there are discrepancies in pricing. Regular rebalancing can deter such exploitative behavior.

However, modification of positions in a pool (dynamic rebalancing) must be executed in a trustless, cryptographically-guaranteed manner in order to ensure that bad actors cannot manipulate LP positions in their favor. Using Space and Time to track the pool's historical state and run queries that observe current conditions ensures trustless LP rebalancing only when user-defined market conditions are met. Space and Time can provide arbitrary checks for market conditions around averaged spot prices and real-time volatility, which are then returned (along with an associated ZK proof) to a smart contract managing the LP positions in order to execute required checks/logic onchain prior to any LP modifications. Doing so would allow liquidity providers to increase their returns without managing their infrastructure or relying on trusted external managers.

5 Decentralized Network Value Accrual

5.1 Node Operations

Network participants (node operators) can contribute any of the following three node types within Space and Time:

1) **Validator Nodes:**

   a) **Execution:** Lightweight service that indexes blockchain data and creates commitments on the data which the Proof of SQL protocol uses to verify that data hasn't been manipulated. These nodes ping blockchain RPC services to get the latest block/transactions/events, then decode and transform the data to prepare it for ingestion through the *Transaction nodes*. Though a few *Validator nodes* actually send raw blockchain data downstream to the *Transaction nodes*, most *Validators* simply complete their data pipeline by updating cryptographic digest (commitments) on the blockchain data they've processed and send only these updated commitments to the *Transaction nodes* for consensus-driven proof of the veracity of indexed data. Post-onboarding, these nodes must sync with the network to retrieve the latest commitments on blockchain data in order to continue indexing. Increasing the number of *Validators* in the network increases overall protocol security, as it implies that more nodes are redundantly indexing the same blockchain data and sending commitments on the data to *Transaction nodes* for consensus-vote.

   b) **Onboarding and Reward Mechanism:** Permissionless nodes that, due to their minimal hardware requirements, can be onboarded with minimal staking requirements. Theoretically, we could configure the network to allow self-service onboarding of *Validators* with as little as $100 staked, and thousands of *Validators* can be onboarded in a permissionless manner with minimal rewards distributed to *Validators* while still ensuring their profitable operation (for example, less than $1000 in rewards annually). Slashing is maintained via consensus around *Validator's* commitments on indexed blockchain data sent to the *Transaction nodes* which actually execute consensus.

   c) **Hardware Requirements:** This Rust-based *Validator* service is lightweight enough to run on a well-equipped laptop such as an Apple Macbook or small AWS EC2 instance.

2) **Transaction Nodes:**

   a) **Execution:** A single Rust-based binary that verifies transactions, achieves BFT consensus and threshold-signs commitments on indexed data from the *Validator nodes*, buffers ingested data to Apache Arrow record batches in memory before compressing to Parquet files on *shared storage* (a public good offered by Space and Time—more on this below), and includes an Apache Datafusion library for serving analytic queries which don't require Proof of SQL (non-tamperproof queries). Increasing the number of *Transaction nodes* in the network increases

overall protocol security, as it implies that more nodes are participating in BFT consensus, thus further preventing collusion.

b) **Onboarding and Reward Mechanism:** The *Transaction nodes*' *Proof of Stake* consensus mechanism can be configured for either permissionless or permissioned node onboarding, when optimizing for performance vs. protocol security, for example. The latter, however, would invite significant engineering effort, as it would require either dynamic Shamir's Secret Sharing (nodes constantly joining and leaving the network without changing the public key, which external verifiers must use to verify proofs against threshold-signed, consensus-voted indexed data from the *Validator node* set) or alternatively would require a constantly-changing public key for threshold-signed commitments on indexed data which external verifiers must constantly be made aware of. *Transaction nodes* require a heavy stake, as there is a significant incentive for external attackers or collusion among the nodes. Slashing is performed when *Transaction nodes* miss SLAs or misvote too often during consensus.

c) Hardware Requirements: Requires a single dedicated server with at least 64GB of memory to operate, although larger amounts of memory should significantly improve OLAP query performance. Does not require a GPU.

3) **Prover Nodes (*Proof of SQL*):**

a) **Execution:** Fulfill ZK-proven query requests that can join and exit the network at will (no onboarding required)—these nodes run the Proof of SQL ZK circuit on their local GPU(s). *Provers* execute a service that listens for query requests/jobs via the events emitted by Space and Time *Verifier* contracts on popular chains, and fulfill the requests by executing a query and associated ZK-proof, submitting results to the *Verifier* contract along with a payment for onchain verification (a bounty submitted as a transaction). *Provers* read blockchain data from *shared storage* when executing queries. Keep in mind that BFT consensus is primarily for verifying the veracity of indexed blockchain data provided by *Validator nodes*. Due to the zero-knowledge-based approach with *Provers*, participation in consensus is not required. Increasing the number of *Provers* in the network does not increase overall protocol security, as each query request/job only requires a single *Provers* to fulfill.

b) **Onboarding and Reward Mechanism:** *Provers* are permissionless nodes with no stake required and no registration with the network required. Once successful verification has occurred onchain, the *Verifier* contract refunds the *Prover node's* account with their bounty (a payment the *Prover node* previously submitted to the

Verifier contract as a transaction to coerce the *Verifier* to attempt to verify the proof/query result for a certain request/job) along with an additional reward for fulfilling the request. In this manner, *Prover nodes* simply compete with each other to fulfill a request/job.

c) **Hardware Requirements:** *Provers* require at least a single NVIDIA GPU for proof generation, but can scale to handle more complex queries or proofs over larger datasets when operating a cluster of GPUs. They outsource persistent storage to the network's *shared storage* service.

While not directly part of network operations (not a specific node type that network participants must operate or stake on), "shared storage" is a public good maintained by Space and Time to provide *Prover nodes* with a *data lake* of comprehensive, always-up-to-date indexed blockchain data to read from when executing queries and building associated proofs. This data is signed by and ingested from the *Validator nodes*, which perform blockchain indexing. This storage service can be either centralized (i.e. Azure Blob storage) or decentralized (an IPFS-based approach) as long as sufficient liveness guarantees and sufficient replication of each dataset is ensured. We opted for a natively-developed storage approach loosely based on IPFS with Delta Lake as the underlying framework for organizing massive Parquet datasets, which audits each dataset for a sufficient number of replicas across cloud availability zones via a *Proof of Storage* mechanism native to Space and Time, ensured by BFT consensus.

We utilize behavioral algorithms to monitor network activity and detect anomalies in node responses, such as latency spikes, data inconsistencies, or ZK-proof errors. Slashing conditions are tiered based on the severity of the violation, ranging from temporary staking penalties to slashing an entire bounty for continued infractions. Honest *Validator nodes*' onchain accounts are rewarded periodically, while Transaction nodes can be rewarded in real-time as part of BFT consensus (Proof of Stake).

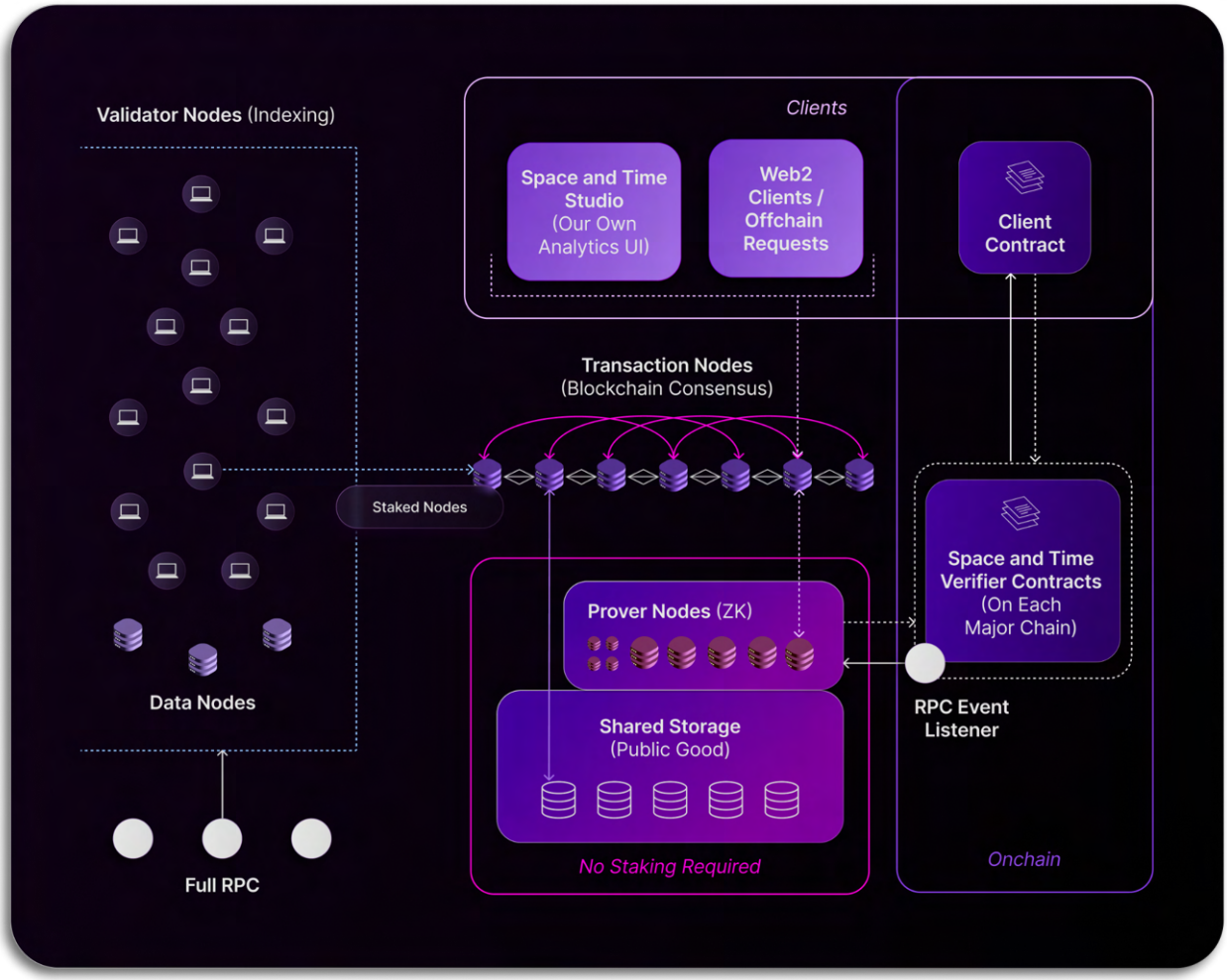

Figure 7: Space and Time Node Operations

5.2 Token Utility

Customers pay for compute, while storage is free (*shared storage* service is a public good within the Space and Time network). A general rule of thumb in the data warehousing vertical assumes the volume of data that an organization stores is directly proportional to the number of queries they execute (the amount of compute they consume). Clients are charged per query with dynamic pricing, adjusting in real time based on network congestion and query complexity, and urgency. This ensures competitive rates and optimal value capture.

A majority portion of fees are allocated to reward node operators—paid via a combination of the native token to further contribute to their stake size as well as $ETH or $USDC to provide node operators liquidity. A segment of the network's revenue forms a reward pool, distributed to nodes and GPU-deployed *Provers* based on their contribution and performance. Small portions of native tokens received as fees for services are burned periodically, adding deflationary pressure.

As more dapp developers issue both onchain (verified by a smart contract) and unverified offchain query requests, demand for our service grows, creating a consistent stream of revenue/utility contributing to token velocity and economic viability of the network.

The cost-per-query for each request is dynamically determined primarily based on query complexity. Query complexity is measured in CPU seconds by the network, offering a stable mechanism that scales linearly, even across varying hardware specs. We utilize smart contract functions to automatically detect and bill query requests, with payment channels supporting our native token as well as popular currencies (denominated in $ETH, $USDC, $USDT, and $LINK, for example). Beyond the basic fee for executing a query and building the associated ZK-proof, included in the client smart contract's payment is gas payment required for onchain verification of the query's associated zero-knowledge proof. Smart contracts interact with our protocol's *Verifier* contract to make a request/create a job, sending a transaction with the query parameters and the required fee. Once the data is fetched/proven offchain and verified onchain, the verified result is relayed back to the requesting client contract from Space and Time's *Verifier* contracts.

5.3 Governance

We plan to gradually implement community governance (i.e. a DAO structure) to remove the need for the Space and Time team to manage the network, by allowing token holders to vote on network features, upgrades, integrations, and fee models. In regards to fee models, token holders influence the protocol's direction directly by periodically voting on "cost-per-query" unit economics. Essentially, governance should ensure that the fees charged to clients for both tamperproof queries (ZK-proven via *Proof of SQL* executed with GPUs) and non-tamperproof queries (normal, unverified query execution via Apache Datafusion executed on standard hardware) are sufficient for rewarding node operators profitably, while keeping fees in-line with market expectations from end-users. Participants can submit network improvement proposals and token holders can propose, debate, and vote on changes using smart contracts for transparency. Voting weight may be influenced by staked amount, and specific node types they operate, if any. Of course, holders can also stake tokens for compute services or delegate staking to node operators in order to participate in a portion of their earned rewards.

5.4 Additional Business Models

Though Space and Time's primary revenue will likely derive from onchain query fees, there are several other streams that serve to generate revenue for our business (and utility to the network's native token):

Enterprise Analytics: We earn significantly through the fiat subscription-based sale of Space and Time as an enterprise analytics platform. Web2 enterprises, who aren't necessarily concerned

with ZK-proving their queries or connecting to smart contracts, use Space and Time as a low-latency, cost-effective data warehouse to power analytics for their businesses. As an HTAP solution, we remove expenditure toward a wealth of point-solutions (transactional databases, to ETL tools, to analytic data warehouses, to BI tools) that plague enterprises. The total cost of ownership to enterprises can vary from tens to upwards of hundreds of thousands of dollars monthly to maintain each of these data tools. Value accumulated via enterprise analytics doesn't have to remain offchain—enterprises may procure query compute credits in bulk, converting fiat revenue to "burned tokens" in bulk, generating demand and adding liquidity to the Space and Time network.

Web3 Data APIs: Space and Time offers a comprehensive set of highly scalable APIs around commonly requested blockchain datasets. By indexing blockchain data in a structured way, we expose APIs that make querying, fetching and understanding this data straightforward for developers. Access can be token-gated or require fiat-based payments on a cost-per-query basis.

Blockchain Data Dashboards: Space and Time is positioned to drive business by providing blockchain data dashboards and visualization tools as a service. Popular Web3 dashboarding platforms, such as Dune Analytics, are growing their market share within the toolkit of onchain data analysts and blockchain enthusiasts. Space and Time plays in this market as well, and uniquely wraps AI-powered visualization tools into a comprehensive, end-to-end data platform with ZK verifiability. Revenue driven from analytics dashboards is generally facilitated via monthly fiat-based subscriptions or cost-per-query pricing.

Data Marketplace: Verifiable indexed blockchain data (ZK-compatible, trustless) from popular chains are ingested to Space and Time's decentralized data warehouse in real time (entire chain state—every block, transaction, event, account, liquidity pool swap, etc.). In addition, users can load in their own offchain data (with trust-minimization via consensus) from TradFi markets, gaming, cloud data storage, etc. Thus, third-party organizations can sell their data through Space and Time while preserving their IP, and Space and Time itself essentially sells its own indexed blockchain data by wrapping incentives for indexing node operators into the cost-per-query for accessing that data in the Space and Time data warehouse.

AI-Powered (Next-Gen) BI tools: Space and Time has made significant strides with OpenAI's APIs to build a simple but robust AI-powered toolkit, which enables users to quickly generate SQL based on natural-language prompts. Though this toolkit was designed with the schema/data model of our indexed blockchain data in mind, customer schemas and datasets are automatically processed as well for fine-tuned natural language queries against offchain data. Not only can Space and Time stand in for traditional BI tools (such as Metabase or Tableau), it's positioned to replace them entirely with a friendly and comprehensive AI-powered user-experience that eliminates the need for specialized skills (even as basic as writing SQL) to leverage analytics.

Recurring revenue from this aspect of the business is generally facilitated with monthly fiat-based subscriptions.

RPC Services: Space and Time operates its own RPC nodes as a public good for our blockchain indexing service. RPC nodes are difficult to maintain and expensive to run, and as blockchain indexing is a core part of our business, we have additional interest in ensuring that our nodes are both high-performance and extremely reliable. Thus, we may be able to provide RPC services superior to existing solutions on the market.

5.5 Network Effects

Competitive Moats: Space and Time is uniquely positioned to capitalize on the moats of both a data warehouse and a Web3 protocol to differentiate itself in the market and establish an indomitable barrier for competitors. Data warehouses generate significant data gravity—once a business has built its operations around one data warehouse, it's complex and economically unviable to migrate to another, increasingly so as more data accumulates within the system. In a Web3 protocol, an increase in network participants not only directly translates to heightened protocol security, as it becomes more difficult for any one actor to gain control, but also creates greater economic stability—as additional nodes and increased staking ensures a more resilient and decentralized control structure within the network.

Network Growth: As more developers query Space and Time, the increased activity generates more fees, attracts more participants, and further solidifies our position in the ecosystem. More chains being indexed and more dapps being built on the network results in a richer and more diverse array of composable datasets, which enhances its utility. Moreover, by expanding integrations across multiple chains, Space and Time is able to tap into more third-party ecosystems and cast a wider net over the market. As more developers use our platform, the value and utility of the network grow exponentially—resulting in more activity, more fees, more participants, and more velocity in the market. A small portion of accumulated query fees offer the potential to be reinvested into research and development, refining the protocol, expanding the range of indexed data, improving query performance, and accelerating nascent cutting-edge circuits for arbitrary computations beyond SQL.

Marrying Scale and Provability: A traditional data warehouse is highly scalable—conceivably able to handle infinite tables—but is centralized and trust-required, and therefore not compatible with Web3. A blockchain, on the other hand, requires no trust but is inherently limited to a single ledger, constraining both the scale and variety of data that it can handle. Proof of SQL juxtaposes the scale of a data warehouse with the trustlessness of blockchain. Thus, Space and Time essentially functions as an infinite-ledger blockchain, satisfying both the need for the scale to power enterprise applications, and the trustlessness required by Web3.

The Result: The circular nature of our network effects (more data gravity, security, and scale draws in more developers/businesses, which in turn draw in more gravity, security, and scale) leads to two main outcomes: 1) Proof of SQL becomes the database market standard—where all companies demand both the scale and provability which only Proof of SQL can offer, and 2) Proof of SQL becomes the ultimate cross-chain state proof—allowing developers to easily and swiftly message the full state of any one chain to another with cryptographic verifiability.

6 Conclusion

As the digital age stands at the precipice of a decentralized future, new cryptographic primitives are required to ensure offchain data processing does not break the trustless guarantees of the blockchain itself. Decentralized applications and smart contracts are powerful, but they've been hindered by far-too-limited data access and compute capability. Space and Time directly addresses these issues, offering a seamless interface for accessing, computing over, and verifying data across varied chains and offchain sources—a coprocessor for smart contracts. Our full stack of developer tools includes a decentralized service to prepare already-verified indexed blockchain data for querying (also called ‘coprocessing’), and a powerful HTAP query engine to execute computations against these datasets with low latency.

Our future-proof design ensures scalability without compromising trust, using a zero-knowledge framework to keep every interaction verifiable and free from centralized manipulation. While optimizing for SQL within our zero-knowledge circuit, we cater to ubiquitous data operations familiar to developers, without limiting them from exploring far beyond (i.e. flexibility for accelerating community-developed circuits deployed on Space and Time to prove arbitrary computations outside of SQL). Self-custody of end-user data is made possible by Space and Time, and developers can use this protocol to build extremely flexible ZK-rollup solutions which transparently empower end-users with granular controls. We’re excited to usher in the next era of sophisticated, data-driven smart contracts and fulfill the vision of a more scalable, transparent, community-operated, and secure Web3.

Delivered to the market as a decentralized network for data warehousing, Space and Time ensures value (derived from fees to end-users or client smart contracts requesting query results) continuously accrues to stakers and participants who provide nodes (hardware) to the network. From direct revenue streams offchain that “burn tokens” onchain to utility around a marketplace for indexed blockchain data, we've crafted an ecosystem where straightforward economic incentives grow the number of participants (and network value as a whole) while driving down the costs of data processing. We invite developers, users, enterprises, node operators, investors/retail stakers, and industry leaders to join us on this exciting journey where data is both the fuel and the path forward.

In summary:

1) Verifiable indexed blockchain data (ZK-compatible, trustless) from popular chains is ingested to Space and Time's decentralized data warehouse in real-time. The entire chain state is indexed—every block, transaction, event, account balance, liquidity pool swap, etc. In addition, users can load in their own offchain data (with trust-minimization via consensus) from TradFi markets, gaming, cloud data storage, etc.
2) Once data has been ingested, trustless computations in zero-knowledge are operated on the data at extreme scale—either via SQL queries or other community-provided proof circuits for arbitrary computations. The results of these computations or queries are returned to clients (such as a smart contract, web browser, or financial institution) for verification along with a ZK-proof that the computations were executed correctly against underlying data that was untampered.
3) Permissionless network participants can contribute by operating nodes (hardware such as a single GPU or cluster of servers) and staking on those nodes along with retail stakers. Incentives to the participants are delivered via query fees charged to clients such as smart contracts or enterprises that need to secure their data offchain.
4) A new generation of sophisticated dapps, flexible ZK-rollups, and cryptographically proven financial systems can be built with Space and Time as the backbone. In addition, for clients who do not require ZK-proven query results, but simply want a next-generation data warehouse delivered at a lower cost, Space and Time introduces hybrid transaction and analytic processing superior to point-solutions (today's popular cloud data warehouses).

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 Information such as block number (block.number), block timestamp (block.timestamp), the block's gas limit (block.gaslimit), etc.

- **Gas Information:** Smart contracts can access limited gas information such as 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 place 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.
- **No access to Logs and Events:** While contracts can emit logs/events, they can't read them directly. Logs are primarily for external consumers, such as dApps or backend services, which can then listen for and process these logs.
- **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. A contract can 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 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).
- State 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. In the context of Space and Time, understanding HTAP is crucial to grasp how the data warehouse maximizes efficiency and versatility.

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

- **Data Redundancy and Complexity:** Maintaining separate systems led to data redundancy and complexity, as data needed 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 suffered from latency issues, as it relied on periodically refreshed data from transactional databases. Real-time insights were a challenge to achieve.
- **Scalability Challenges:** Scaling these separate solutions in response to growing data volumes and user demands was 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 data warehousing 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 zero-knowledge proof attached to SQL databases, which cryptographically proves to a client that both query execution and underlying tables are untampered. We can leverage the Proof of SQL protocol to trustlessly verify query results returned from other outsourced traditional database/data warehouse solutions besides only the Space and Time decentralized data warehouse solution. For example, we 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 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 Space and Time's Proof of SQL service to ZK-prove that queries against data were executed accurately and that the underlying data hasn't been tampered, we 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. Data loaded into the database should be loaded in through the *Validator*. Queries executed against the database should be routed through the *Validator* if they need to be ZK-verified (validate that the query results and underlying tables have not been tampered).

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 data warehouse):

- **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, our *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 this as the only decentralized data warehouse system, relying on a global committee of user-operated nodes which come to consensus to prove 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 would 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 it's only going to continue to accelerate.

As blockchain technology is widely adopted by enterprises, distinctions between Web2 technology and Web3 technology will become less significant and less useful. 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 decentralized applications across major chains
- 35K Web3 developers building on the major chains

An investment today in SxT has two main sources of value that are unique to see in the same company, in the same platform: 1) delivering familiar data warehousing tools as a Web3-native experience, and 2) verifiable (trustless) compute, where all activity in the data warehouse 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 off chain.

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 have 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 similar to how cloud data warehousing and analytics evolved. Snowflake, a popular cloud data warehouse, was the largest software IPO ever as of 2020. Data management software today is a global $75B market with high single-digit to low double-digit growth. AI platforms specifically are growing at 30+%.

While traditional Web2 data warehouse technologies are point-solutions which 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 $3M ARR since April, with an

impressive 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 revenue makes SxT one of the fastest growing data analytics companies ever.

While total market size today for this fast moving category is difficult to calibrate, every Global 5000 company and all Web3 companies (~23K+) will need a database / analytics platform with Space and Time's functionality. Estimating based on SxT's beta 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 requiring that their own offchain transactions and market data 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 zero-knowledge 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 SxT 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
- 70% of cloud data networks – accessible through SxT

As part of developing the ZK-compute layer, SxT has driven a major innovation by taking Proof of SQL to market. Simply put, in a world that operates at the intersection of Web2 and Web3, there is broad value proving that a query was executed properly, or that the data used to execute business logic within a smart contract was filtered/aggregated properly. 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 working on AI within blockchains. Lastly, companies of all sizes have latched on to Proof of SQL, particularly in the area of compliance and auditing.

At the core of all these themes is the strong market need for trustless verification that data is correct and was computed on properly—which is different from blockchain transaction verification. Both are important, but the gravity is tilting to 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 for financial data, where there is a deeper incentive for bad actors to tamper).

Estimating market potential is more based on analogs vs. market participants multiplied by a price model. That said, SxT 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, where revenue today in Web2 is a $65B market growing at 10%+ and expected to be close to $200B by 2032... and this is only one use case for Proof of SQL. Another example would be 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

---