Glacier Network: Building A Modular, Dynamic and Scalable NoSQL Database for Large-scale Decentralized Applications

Glacier Labs, v0.2.3

Mar 1st, 2023

Abstract

Modular, dynamic, and scalable databases are one key challenge for future blockchain-based large-scale applications. Currently, a large number of Web3 applications have to temporarily store a vast amount of user and product data on centralized cloud services due to the lack of developer-friendly decentralized storage tools. Glacier believes that with the development of Web3 applications and servers, more Web3 developers will choose to store data in a completely decentralized underlying protocol. Glacier's vision is to provide Web3 developers with a NoSQL Database of Data Availability and Data Composability, to help them avoid the risks associated with centralized databases. Glacier will not only provide developers with decentralized database services, but also help them mine and analyze the public data on the user chain, such as address information, transaction data, content data, and social relations, in order to build more valuable information and data trading market, which will help in the monetization of blockchain data.

"Information is power. But like all power, there are those who want to keep it for themselves."

Aaron Swartz

Contents

1 Background

1.1 Web3 Storage

Since the development of Web3 storage is still in its early stages, a large number of Web3 applications still choose to store data on centralized servers or cloud services. Unfortunately, centralized systems pose risks such as single points of failure, data loss, privacy leakage, etc. Glacier believes that the decentralized storage of Web3 is now taking shape. With the improvement of storage infrastructure, more and more Web3 developers will choose to store data in decentralized storage protocols.

With the development of Web3 storage, the underlying file storage protocol has gradually been adopted by many Web3 developers. However, since the underlying distributed storage protocols, such as Arweave and IPFS, can only store static files, developers only get a storage file ID. This cannot meet the requirements of structured storage, query, and data modification.

Web3's data is fully accessible on the blockchain and is of great value. User data created and recorded by Web3 DApps is also one of the greatest assets for Web3 devs. Web3 developers should be easily able to develop new applications based on related data and generate essential business insights, but it is fundamentally impossible for Web3 developers to achieve this right now due to the following challenges:

- **Poor Data Accessibility**. Today, Web3 data is locked in different decentralized storage files, preventing complete data visibility to the developers. This leads to poor customer experience, missing insights, and slower app development.
- **Lack of agility**. Demands for faster and easier deployment to the fast-moving Web3 markets and higher productivity are held back by today's extremely unfriendly storage environment and rigid relational data models.
- **Limited data support**. Web3 developers have limited data support for easily analyzing, querying and monitoring their own user data through different channels like web and mobile.
- **High Cost.** Expensive fees in decentralized storage, huge jumps in costs as workloads scale, and multi-chain storage management impose barriers to innovation.

Therefore,there is a huge demand for the Web3 database layer that could provide a fully decentralized database service built for resilience, scalability, and the highest levels of data privacy and security in Web3.

1.2 Web3 Database

A16z published an article co-authored by Harvard Business School associate professor Scott Kominers and Koodos co-founder Jad Esber called *How to Build a Reputation-Based Decentralized Identity System*. It points out that there is tremendous value in on-chain data. How to manage and apply this data requires

publicly accessible data standards—agreed data formats such as "what a proof of contribution looks like" or "how to represent an on-chain record."

Structuring this complex information ensures interoperability:

(1) enabling users to effectively combine their "identities" from different services.

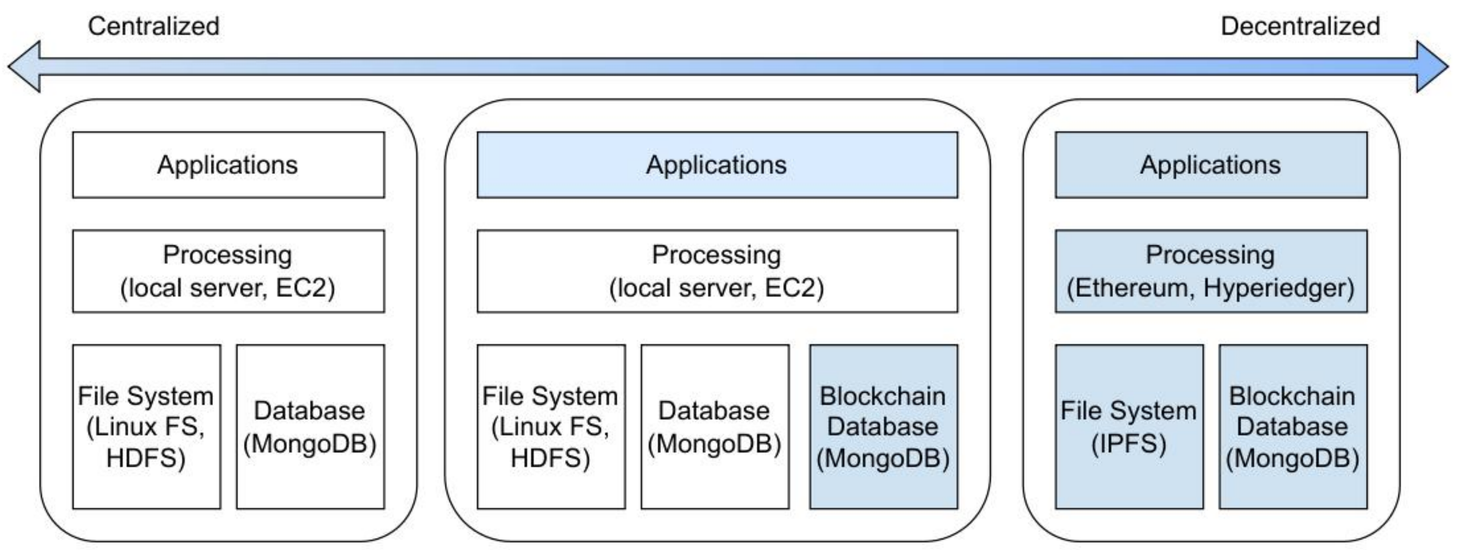
(2) minimizing the friction that may occur between different information of varied reputations on the platform or personal chains.

As multi-chain develops, the free circulation of crypto assets has been realized. However, with the development of on-chain applications, a wider gap is constantly forming - the data gap.

With the prosperity of Web3 applications, more applications bring about more complex data, which are stored in centralized servers or public clouds instead of decentralized systems such as Arweave. This data is stored as static files, which are structured and of high shareability. At the same time, the data stored by the decentralized storage protocol of the underlying file is not structured; only the ID of the transaction or storage file is. This makes it quite complicated for Web3 developers to query storage files and to call and update data in storage files.

A decentralized database is needed between the Web3 application and the underlying file static storage protocol to solve the issues of structured storage of data and realize the data availability and data composability of Web3. It is due to the lack of Web3 database services that a large number of Web3 developers still choose to store data structures on centralized servers or cloud services. To some extent, this results in a high sharing barrier between the data of Web3 applications, restricting the free flow of data.

Therefore, Web3 now urgently needs a decentralized on-chain database protocol to help developers build a decentralized, structured, shareable, and composable data storage, which will ultimately achieve low-cost data storage, sharing, and circulation.



The spectrum of centralized vs decentralized deployments

2 Introduction

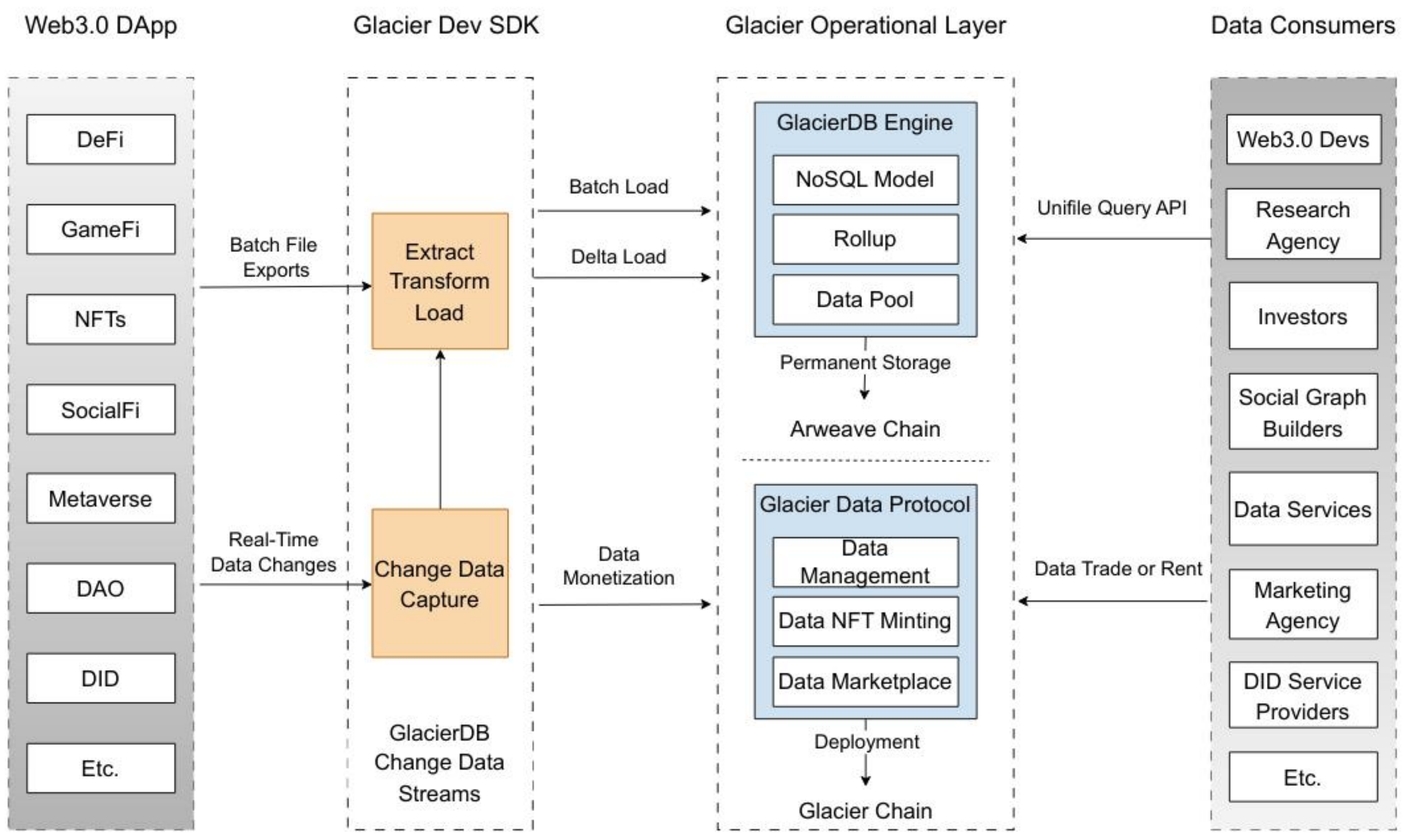
2.1 Glacier Database

The prosperity of Web3 has led to an explosion of on-chain data, including address information, transaction data, content data, and social relations. If developers of Web3 applications want to mine the data on these user chains, then SQL databases are no longer suitable for these applications. Instead, NoSQL databases can handle these vast amounts of data well.

Glacier's mission is to build a secure, decentralized, and highly scalable Web3 NoSQL database to better help Web3 applications store massive amounts of institutionalized data and fully mine the Web3 on-chain data in order to provide better data services to users. The main types of NoSQL non-relational databases provided by Glacier are Key-value, Document, Columnar, and Graph.

With the decentralized architecture of Web3, developers need to deal with a large amount of scattered and complex data on the chain, which greatly hampers the application's development cycle, as well as the experience and scalability of the application itself. Glacier database greatly simplifies the development process of applications and protocols by providing highly scalable decentralized data storage and query services, enabling dApps to seamlessly insert data from other applications. By incorporating on-chain decentralized data into a NoSQL database, Glacier will help Web3 developers better store structured data for their applications.

Glacier defines a new standard for the structured storage of Web3 data. Under this standard, Web3 data can be easily called, queried, and even traded. A large amount of user data of Web3 is in the public state on the chain and has strong intrinsic value. At the same time, the data structure of NFT is similar to that of a NoSQL database. So, Glacier can make it very convenient for users to generate Data NFT from the data they own, which can be then monetized by trading the data in Glacier's Data Marketplace.



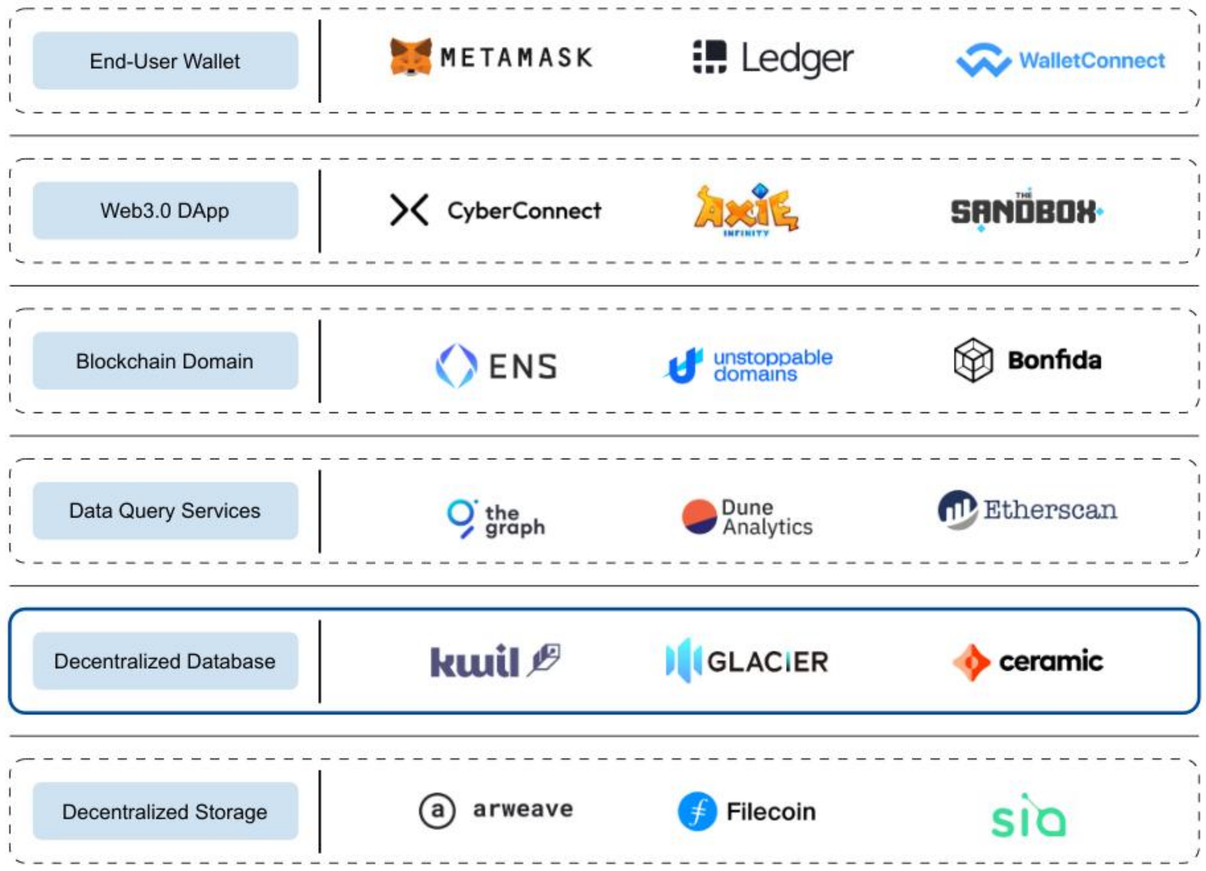
2.2 Design Principles

Glacier's vision is to provide the most user-friendly, scalable, and decentralized database services for Web3 developers. We will design the Glacier protocol with the following principles:

- **Large-scale on-chain adoption with zk-Rollup.** Glacier's innovative solution allows for processing massive amounts of data in production while also improving data services for users. Glacier's modularization and decentralization are secured by the implementation of zk-Rollup technology.
- **Fully supporting NoSQL.** Glacier will mainly focus on providing NoSQL database features, including the types of key-value, the graph, the document, and the column family.
- **Efficient querying.** Glacier will support efficient querying throughout for Web3 developers by providing a unified query API tool.
- **Decentralization.** All the data will be stored in a decentralized manner through Glacier's nodes cluster and the Arweave blockchain. The data owners will have full control over their own database, and the Glacier Protocol will be fully controlled by GLC token holders.
- **Immutability.** All the data flow and storage will be recorded on the Glacier chain with an ordered sequence of blocks, with each block holding an ordered sequence of transactions. This will make all the data stored on Glacier immutable.

2.3 Position in Web3 Storage Stack

In the entire Web3 technical stack, the data storage protocol from the client to the bottom can be divided into six layers. Glacier is mainly working at the database layer, which solves the problems of how to help Web3 developers realize structured data storage.



2.4 Competitive Edge in Web3 Database

In the world of blockchain, the leading projects that focus on data-structured storage positioning are Ceramic, WeaveDB and Kwil. Compared to them, Glacier has the following edges:

1) Glacier is the first Web3 database protocol built on the zk-Rollup to serve as the NoSQL database.

2) Glacier not only solves problems associated with the institutional storage of Web3 data, but also combines NFT technology with the ownership of Web3 data to realize the monetization of Web3 data.

3) Glacier will enable all data to be freely combined and traded through Glacier's database service, and provide a professional Data Marketplace to realize free data trade, indexing and renting.

Key Metrics	Glacier	Ceramic	Kwil	WeaveDB
Key Feature	NoSQL Database	Graph Database	SQL Database	NoSQL Database
Validator Network	Yes	Yes	Yes	No
Validator Incentives	Yes	No	Yes	No
Own Chain	Yes, based on ZK-Rollup	No	Yes, based on Secret Proof	No
Data Monetization	Yes	No	No	No
Data Owner-Access Architecture	Yes	No	No	Yes
NFT Support	Yes	No	No	Yes

2.5 Data Monetization

Glacier will not only provide Web3 developers with a friendly distributed database but will also provide any Web3 user with a tool for data monetization. Users will not only be able to store data in Glacier in a structured way but also store all their data in Glacier in order to generate NFT assets while realizing the transaction of data asset ownership.

Different permissions of the data structure can also generate NFT assets corresponding to different attributes:

- Proprietary NFTs. The NFTs anchor the ownership of the data, namely the Owner Key. The address that holds the ownership NFTs will have the Owner Key corresponding to the metadata.
- Access NFTs. The NFTs anchor the access rights to the data, namely the Access Key. Addresses that hold access rights to NFTs can have the right to access the corresponding metadata.

For the generation of access rights NFTs, Glacier will support personalized programming, such as the number of visits, the duration of the visit, whether it can be transferred, etc.

At the same time, Glacier will build a low-gas, efficient, and secure marketplace to trade the data NFTs. Everyone could use the Glacier marketplace to trade data NFTs, including those issued by Glacier and those issued by other protocols.

3 Glacier Architecture

3.1 Full Network

Modular, Dynamic & Scalable NoSQL Structure
The core of Glacier Network is a modular, dynamic, and scalable NoSQL database GlacierDB. It is built for resilience, immutability, interoperability, and the highest levels of data privacy and security in Web3.
Large-scale On-chain Adoption with ZK-Rollup
Glacier powers massive amounts of data in production and fully mines the on-

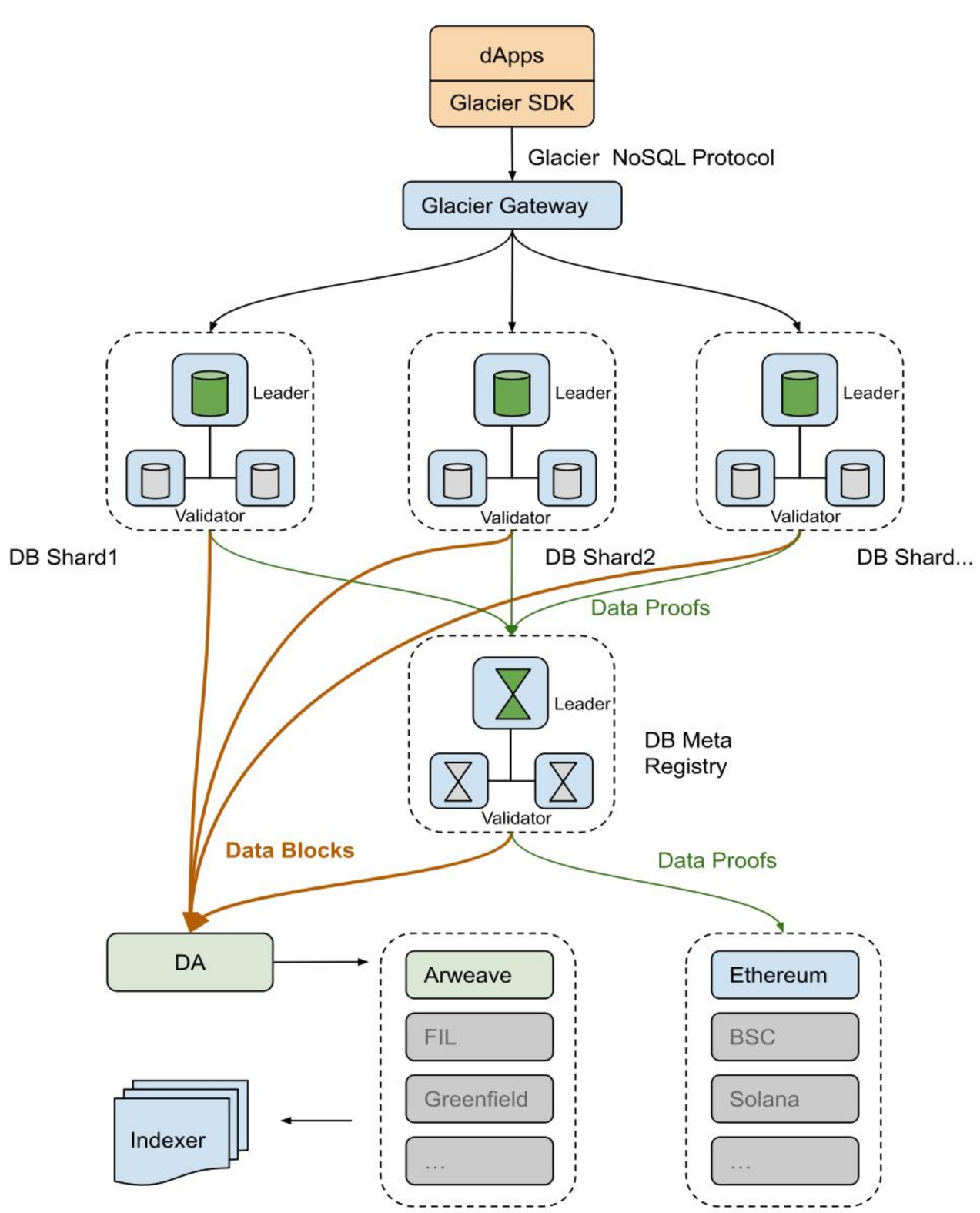
chain data to provide better data services to users. Modularization and decentralization are secured by zk-rollup.

Built On the Top of Arweave Storage and More

Glacier applies Arweave Log Data (ALD) based storage-based consensus paradigm (SCP) designed to store, share, and host datasets at scale to make the storage and query fully secure. Powered by Bundlr and Warp.

Mint & Build Your Own Datasets in One Minute

With Glacier, developers can securely mint and build their own database in a permissionless way that enables Web3 devs to utilize data easily and effortlessly at an ultra-low cost.

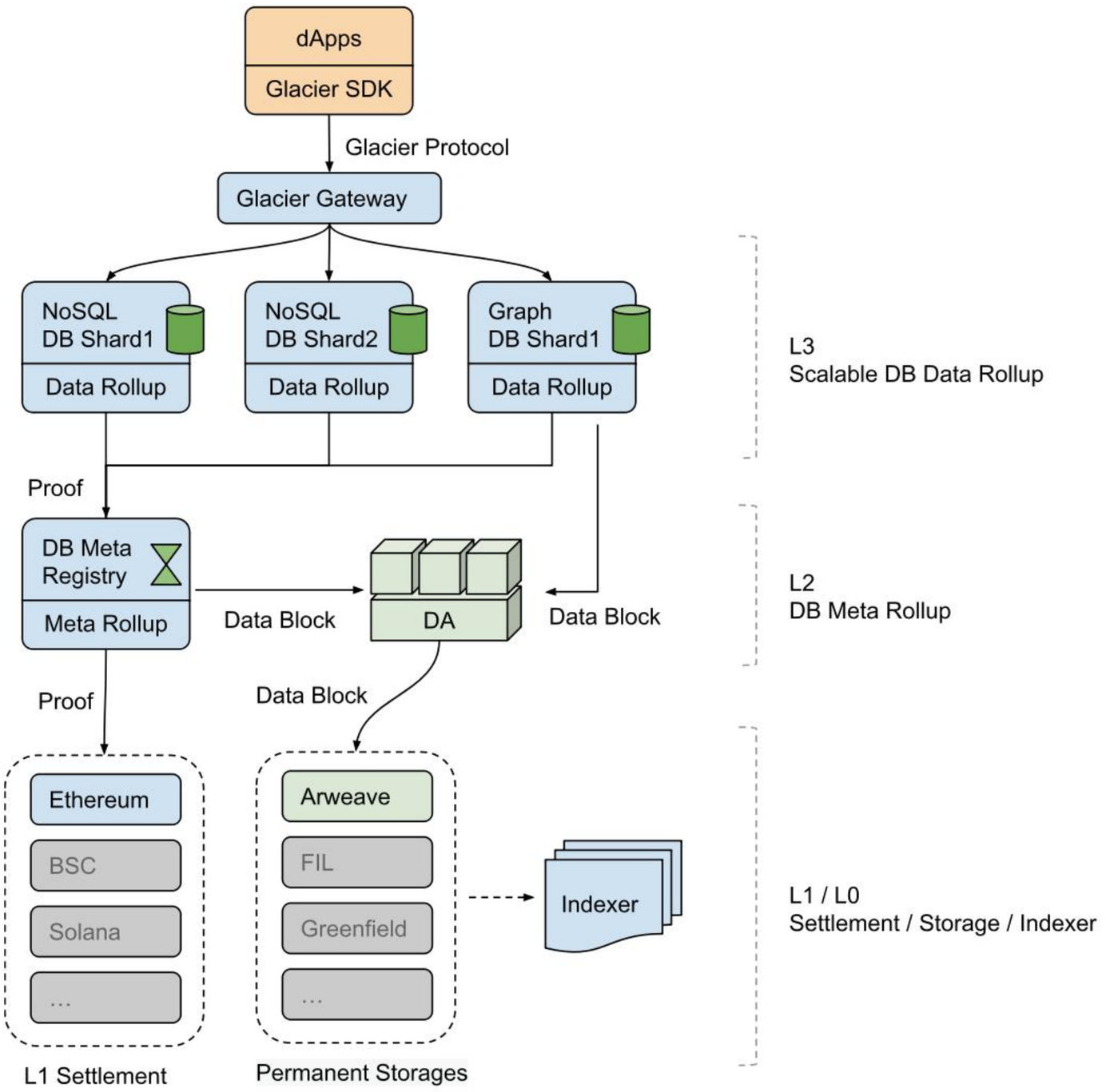


3.2 Technical Stack

The core of Glacier protocol is a fully decentralized NoSQL database service. We call it GlacierDB, built for resilience, scale, and the highest levels of data privacy and security in Web3. With GlacierDB, Web3 developers can easily build their own database in a decentralized and secure foundation that enables Web3 devs to store and deal with the data easily and effortlessly.

GlacierDB technical stack mainly consists of the following layers:

1) Web3 DApps, which are accessing the Glacier decentralized protocol through Glacier Dev SDK.
2) Glacier decentralized protocol, which is the main part of GlacierDB and consists of the following segments:
   - Glacier Dev SDK
   - Glacier NoSQL Access Protocol
   - Database Engine Shards
   - Database Transaction Rollup (Data Transactions Rollup)
   - Data Permanent Availability Network
   - L1 (Layer 1) Settlement
   - Indexer Network
3) The permanent storage layer of the Glacier database is currently compatible with Arweave storage. In the future, we plan to support additional decentralized storage options that can be easily integrated.



In most cases, a dApp is an independent Web3 web application running on a browser or client, and the dApp accesses the Glacier Network through the Glacier SDK. Glacier SDK is implemented in JavaScript. It integrates functions such as user wallet signature and authentication, encapsulation of query requests, and processing of Glacier network communication protocols, allowing developers to quickly connect their dApps to Glacier Network with just a few lines of code.

Glacier Network is the core of Glacier which is based on a decentralized protocol. It consists of two layers: Data Rollup and Meta Rollup. The Data Rollup layer supports various types of NoSQL database engines such as KV, Graph, etc. Data engines of the same type can also be sharded to distribute data to different processing nodes within a single engine, enabling scalability for massive data. This layer can be considered as a Layer 3 network.

The Meta Rollup layer provides registration, management, and settlement capabilities for numerous database engines/shardings in the upper layer, serving as a Layer 2 network. Meta Rollup supports the extension of various database

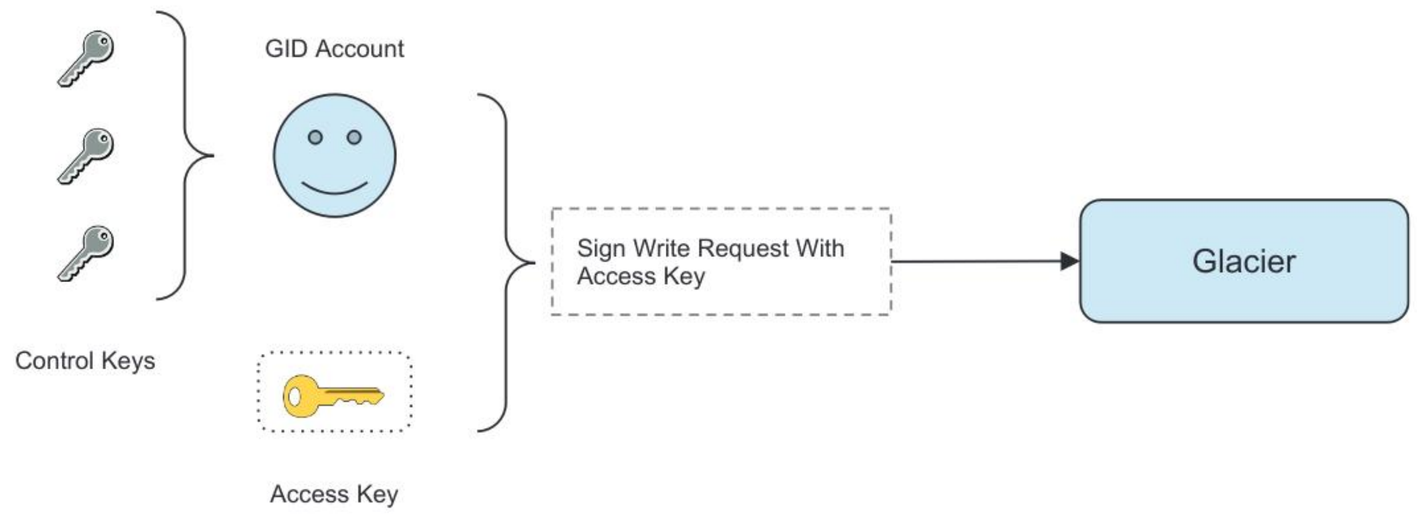
engines and sharding, thereby providing flexibility and support for big data.

The dApp's data will first be written to the Glacier Network to achieve higher data availability and resistance to modification. The Glacier Network periodically generates Data Blocks and Data Proofs, which are submitted to the L1 Settlement contract to ensure correctness. These Data Blocks are then submitted to the DA Network, which encodes and makes them public to achieve Data Availability. Finally, these Data Blocks are archived in decentralized storage such as Arweave and others. In this way, even in the event of a catastrophic situation, Glacier can retrieve the data from Arweave and rebuild all the historical data of the Glacier Network, ensuring that the user's stored data will not be lost.

The Indexer Network supports different types of data indexing to enable further processing and analysis of the data, providing various views to accelerate queries. The layer provides various views to users, which helps users easily understand the data and perform various tasks like data analysis, data visualization, and data exploration. The layer also supports data aggregation, filtering, and sorting, enabling users to extract insights from the data easily.

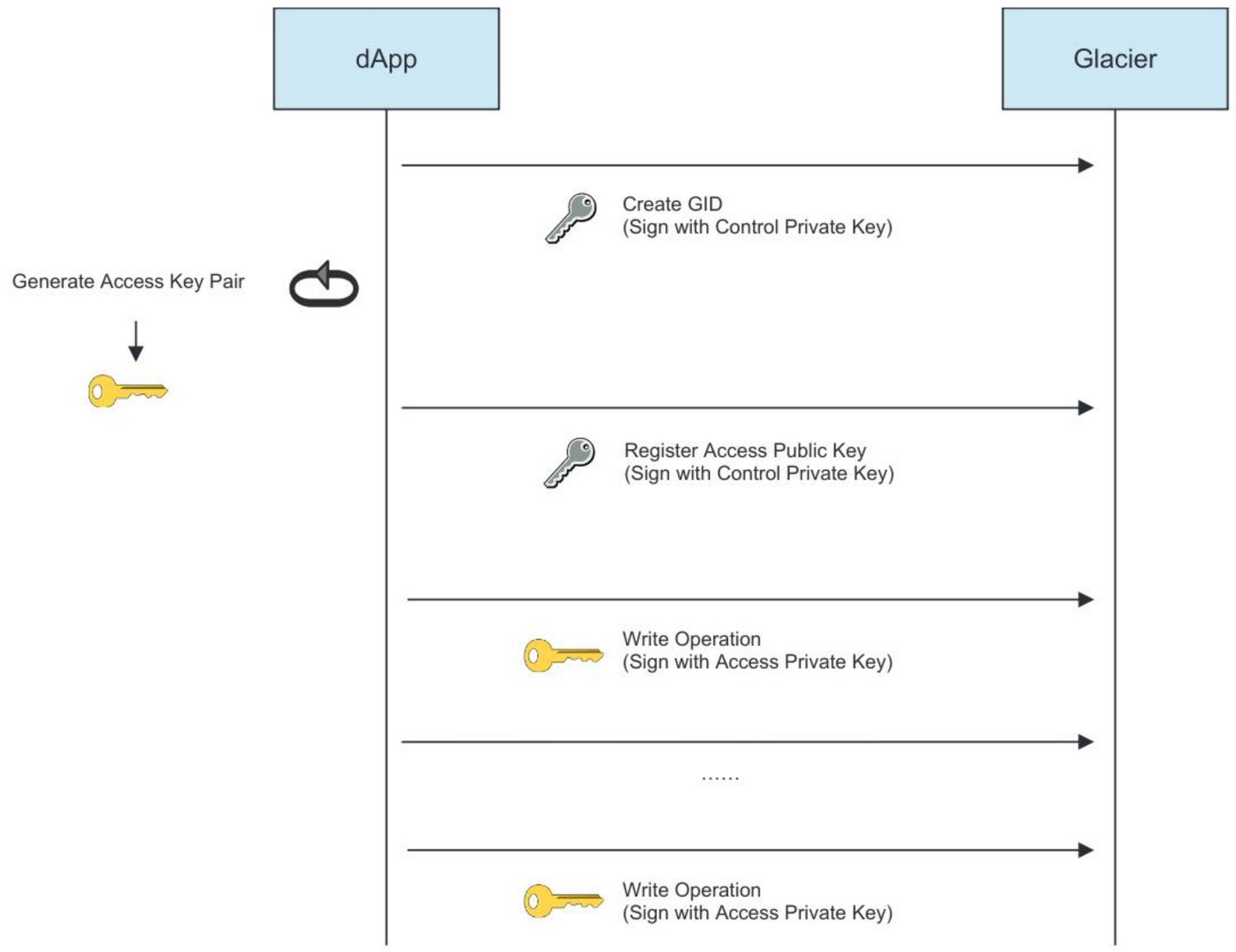
3.3 GID Account and Control & Access Keys

Before using Glacier's database service, users need to create a Glacier On-chain ID account, namely a GID account, to interact with Glacier Network. Users have full ownership and control of the data they have deposited into Glacier by controlling the private key of the GID.



The Control Key is a public/private key pair, and Glacier users use a Control Key to create a GID and associate this Control Key with the GID. The Control Key already associated with the GID can be used to add a new Control Key to make it associated with the GID, or delete the old Control Key. So one single GID can be paired with multiple Control Keys. This design can bring a lot of convenience to Web3 users, because one user may have multiple addresses and private keys, or even multiple addresses and private keys on different chains. A GID account can make the same Web3 user share a centralized data package from different blockchain environments.

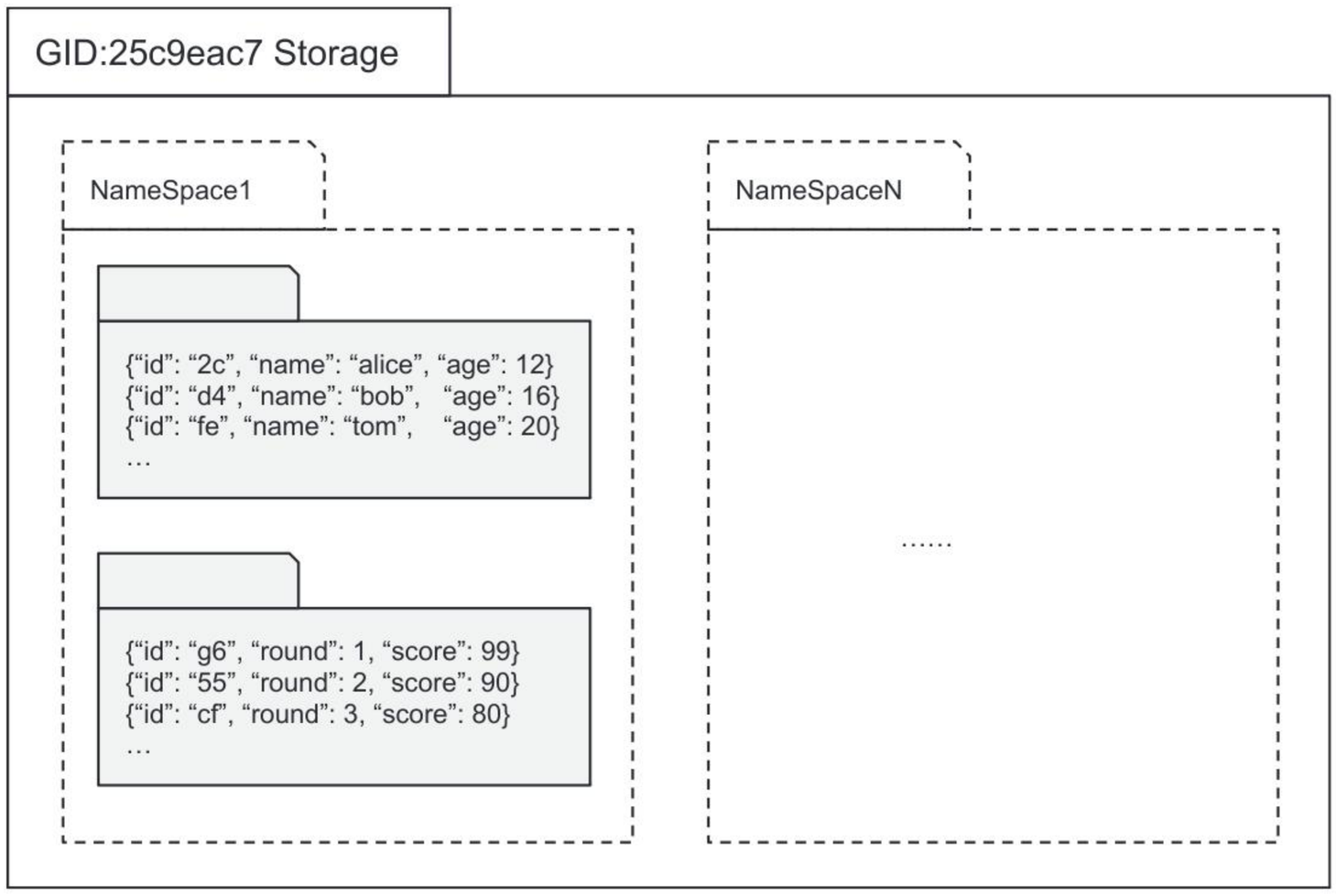
When users write data to Glacier, each write operation needs to be signed with the Access Key private key. The Access Key public/private key pair is temporarily generated. The Access Key private key is stored on the client side. After the Access Key public key is signed with the Control Key, it will be registered in Glacier Network and associated with the user's GID. The advantage of using the Access Key to sign the written data instead of using the Control Key directly is that the private key is usually managed by the user's wallet (such as Metamask). If every writes data operation needs to use the wallet signature, every write operation needs to be confirmed on the wallet (such as Metamask), which will seriously affect the user experience of the dApp. The user only needs to use the Control Key to register the newly generated Access Key public/private key pair with Glacier once. In the subsequent data writing process with Glacier, only the Access Key can be used instead of the Control Key. The user can regenerate the Access Key anytime or log out of the old Access Key.



3.4 NameSpace

Glacier is divided into different NameSpaces, and different NameSpaces can be used to isolate different dApp data storage spaces. Multiple Collections can be customized by dApps within a single NameSpace. Each Collection consists of multiple Documents, which is similar to MongoDB.

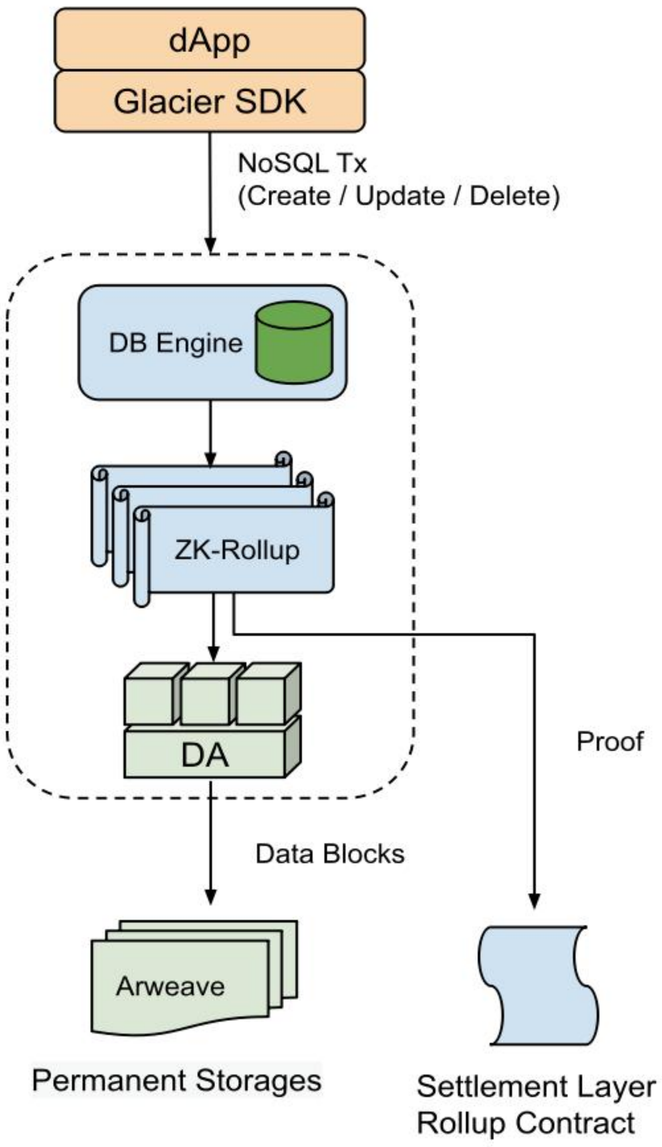
Glacier provides efficient query capabilities for Collection, such as complex conditional query, sorting, etc.



4 Glacier Network

4.1 DB Engine

The DB Engine is the core unit of the Glacier Network, and the entire network is composed of many DB Shards (which will be discussed in detail in the following sections). The DB Engine is responsible for processing data within each Shard.



The dApp initiates CRUD (Create/Read/Update/Delete) operations towards the DB Engine using the NoSQL protocol through the Glacier SDK. The process of persisting these operations to the Permanent Storages and Settlement Layer Contract is as follows. Note that, like other blockchain applications, only operations involving network data changes will trigger a persistent action. Reading and querying operations do not require triggering Rollup, nor do they require a signature.

(1) After the Create, Update, Delete Tx operations enter the DB Engine, the DB Engine verifies the signature of the Tx and then applies it to the table model of the DB Engine.This verification process ensures that the transaction is valid and authorized before it is added to the database.

(2) At certain intervals, the DB Engine generates a Data Block containing the Tx data from the past interval. The Data Block is then submitted to the Data Availability Network, which is responsible for ensuring that the data is available and accessible to all nodes in the network. In the Data Availability Network, the Data Block undergoes Erasure Coding reconstruction, resulting in smaller data fragments. These fragments can be sampled and verified by lightweight clients in the Data Availability Network, ensuring that the data is accurate and complete.

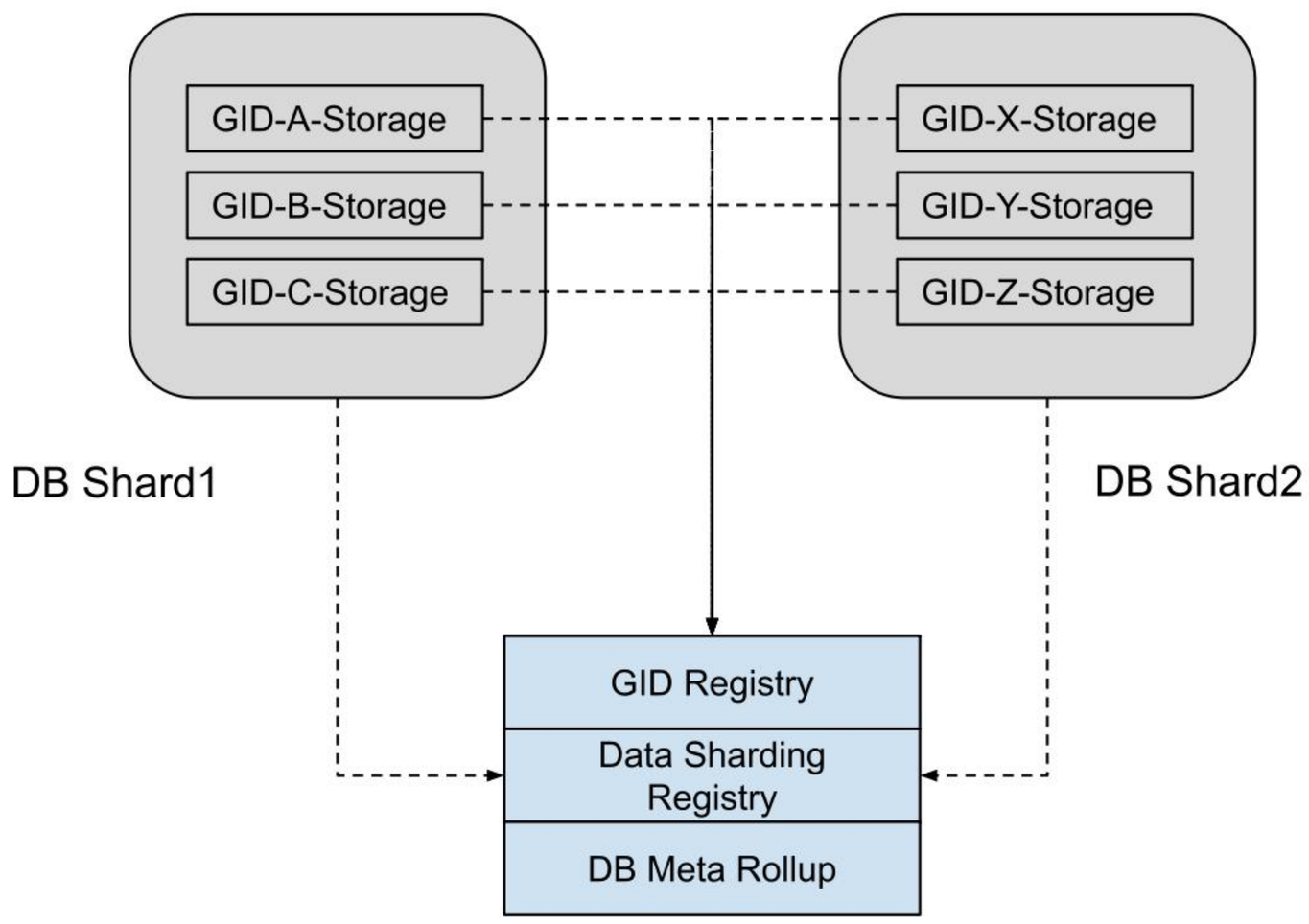
(3) Meanwhile, the Data Block will be submitted to Arweave (with support for other permanent storages in the future), and then receive the access address

and storage Merkle proof of the Data Block in Arweave. This Merkle proof can prove that the Data Block has been stored in a certain Block in Arweave.

(4) The Rollup module calculates the Data Proof based on the Data Block. The Data Proof and Storage Merkle proof are submitted to the Settlement Layer contract. If the Data Proof is incorrect, the contract submission will fail.

4.2 DB Sharding

In order to horizontally expand the storage and computing capabilities of the entire Glacier Network, we do sharding for the Glacier Network, which consists of a large number of DB Shards. Each GID Storage belongs to one of the available DB Shards. Glacier has a global DB Sharding Registry that records the DB Shard to which each GID Storage belongs, along with the ID and address of each DB Shard.

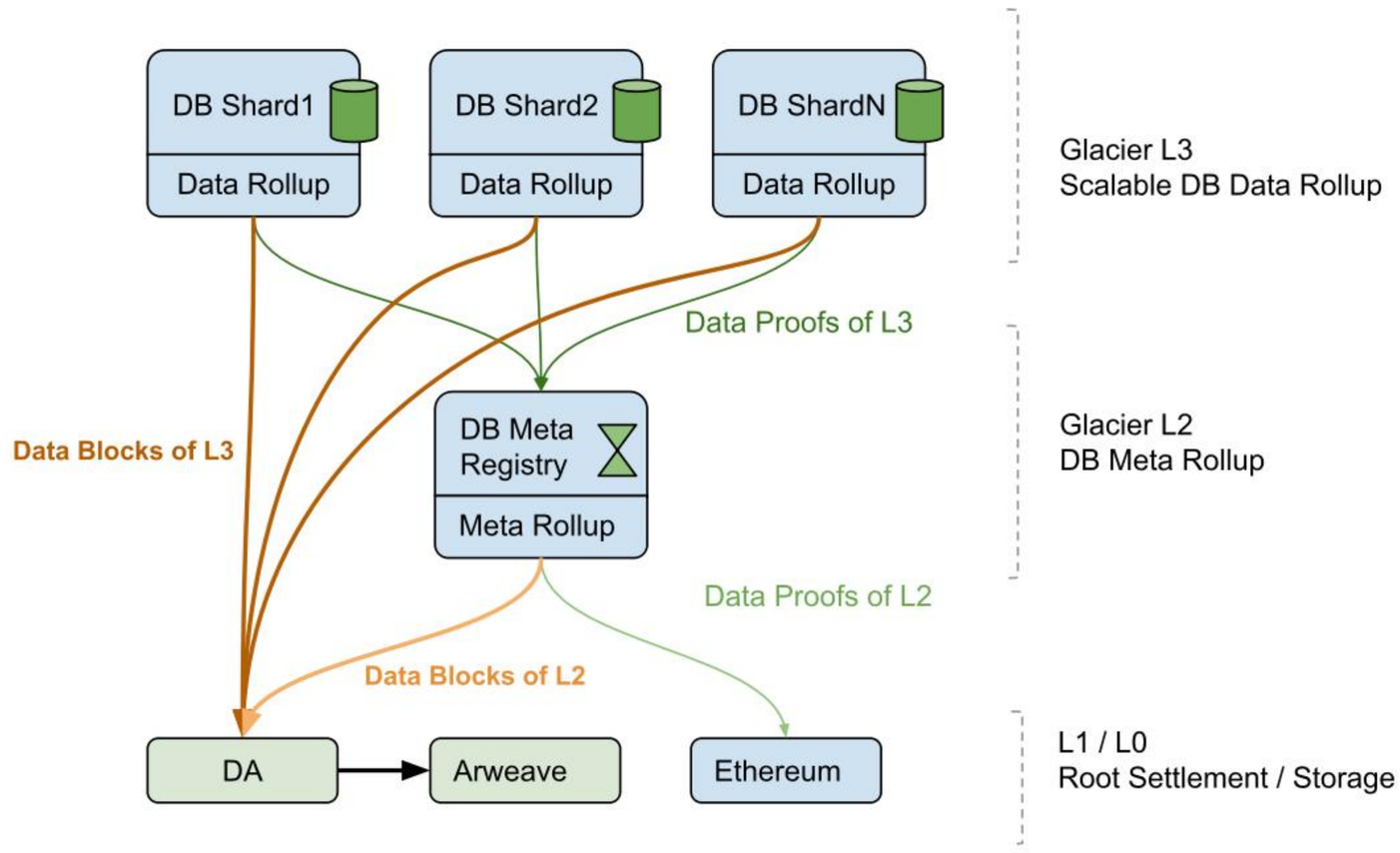


Before accessing Glacier, the Glacier SDK will first query the DB Sharding Registry to which the current GID belongs, find the location of the corresponding DB Shard from the Registry, and finally initiate an access request to the DB Shard.

Each DB shard generates data proof and data block periodically, and the data block is publicly submitted to the Data Availability Network and archived into decentralized storage like Arweave. The data proof is submitted to the Settlement Layer Contract. Using L1 as the settlement layer for DB shards will result in linear growth of data proof writing pressure with the increase

in the number of data shards, which will seriously limit the scalability of the Glacier Network, especially in low throughput and expensive L1 networks like Ethereum.

To ensure scalability and avoid limitations on the number of DB shards in the Glacier Network, a DB Meta Rollup has been introduced to provide settlement services for data rollups. The DB Meta Rollup serves as a Layer 2 solution for the Glacier Network, while the numerous data shards are built on top of it as a Layer 3 data network. This layered rollup structure enables almost unlimited expansion of the number of data shards.



The collaborative workflow between Layer 3 and Layer 2 is as follows:

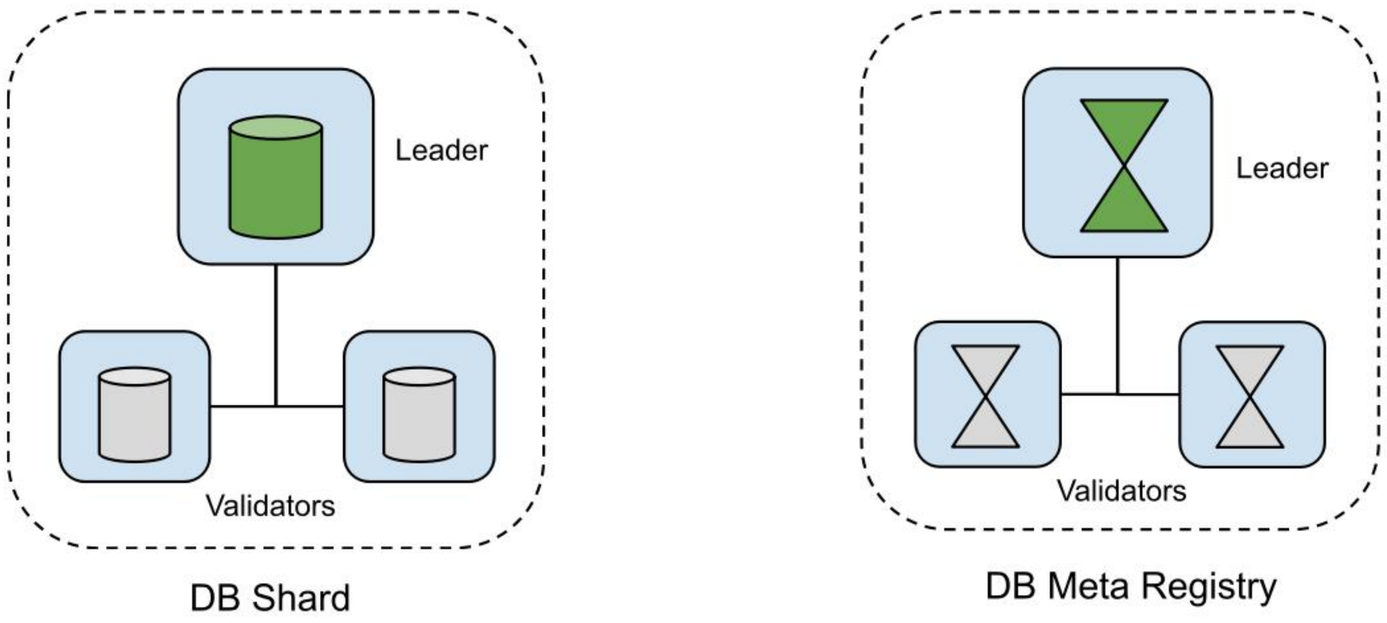
(1) A large number of DB Shards are deployed on Layer 3. As Txs on the network are continuously processed, each DB Shard will continuously generate Data Blocks and Data Proofs. The Data Blocks will be directly delivered to the DA network and made public, and eventually stored in Arweave. The Data Proofs will be delivered to the Glacier Layer 2 network, which serves as the Settlement Layer for the Glacier Layer 3 network.

(2) The DB Meta Registry on Glacier Layer 2 receives Data Proof Transactions from all DB Shards on Layer 3, validates and processes them, and then performs Meta Rollup operations to generate Glacier Layer 2's Data Block and Data Proof. The Data Block on Layer 2 and the Data Block on Layer 3 will both be delivered to the DA network, but the Data Proof on Layer 2 will be sent directly to the L1 contract (such as the Glacier Rollup contract on Ethereum).

4.3 Decentralization of Rollup Sequencer

The rollup sequencer processes transactions, produces roll-up blocks, and submits rollup proofs to the Layer1 chain, and submits rollup data blocks to the data availability layer. To prevent negative effects such as transaction censorship and downtime caused by the centralized sequencer, the rollup of the DB Shard and DB Meta Registry uses a validation network consisting of a single leader and multiple validators to ensure the reliability of the sequencer.

A sequencer validation network consists of multiple sequencer nodes, each of which is independent and has the same copy of data. Among them, the sequencer node (leader) is responsible for receiving transaction traffic, and other sequencer nodes (validator) are responsible for data validation and replicas. If a new sequencer node needs to participate in the validation network, it needs to stake a certain amount of GLC. If the leader is down due to certain circumstances or there is a problem with the block generation, the leader will be punished, and the entire validation network will reselect a new leader to resume work.



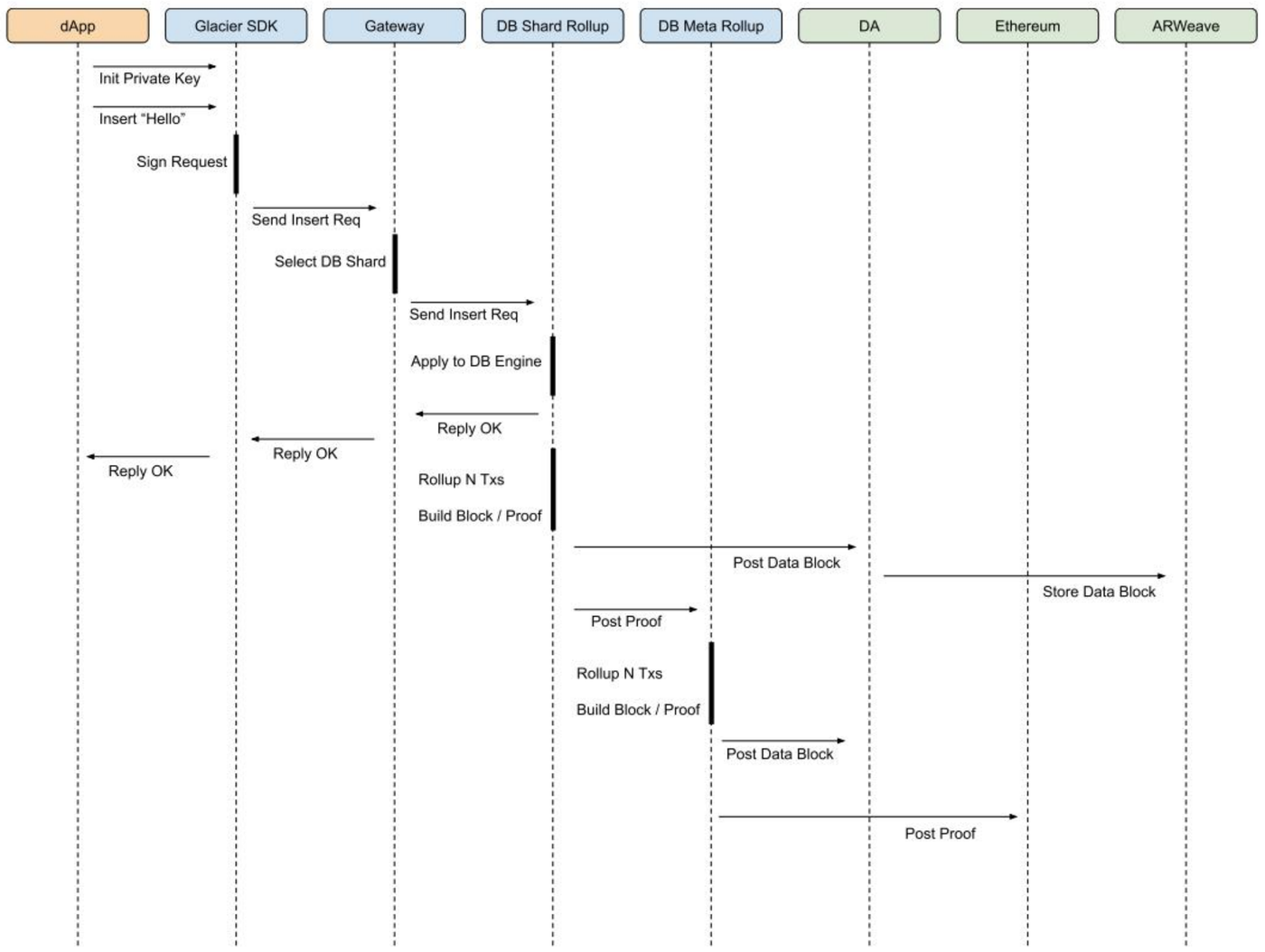
4.4 Data Writing Process

Before writing data to Glacier, the dApp first signs the write operation with the Access Key. The signed write operation is then sent to Glacier Gateway. Glacier Gateway provides a REST HTTP interface to each other to provide convenient access services for dApps, so that dApps do not need to use very low-level protocols to interact with many Glacier Data Node nodes, but only need to interact with the Glacier Gateway.

Glacier Gateway will query the Glacier Network to which the GID belongs from the Registry according to the GID of the current request, and then send a write request to the correct Glacier DB Shard Leader. After Glacier DB Shard receives a write request, it will first verify whether the request is legal and whether the signature is correct. Thereafter, it will broadcast the write operation to all the validator nodes in the current shard, and then apply the

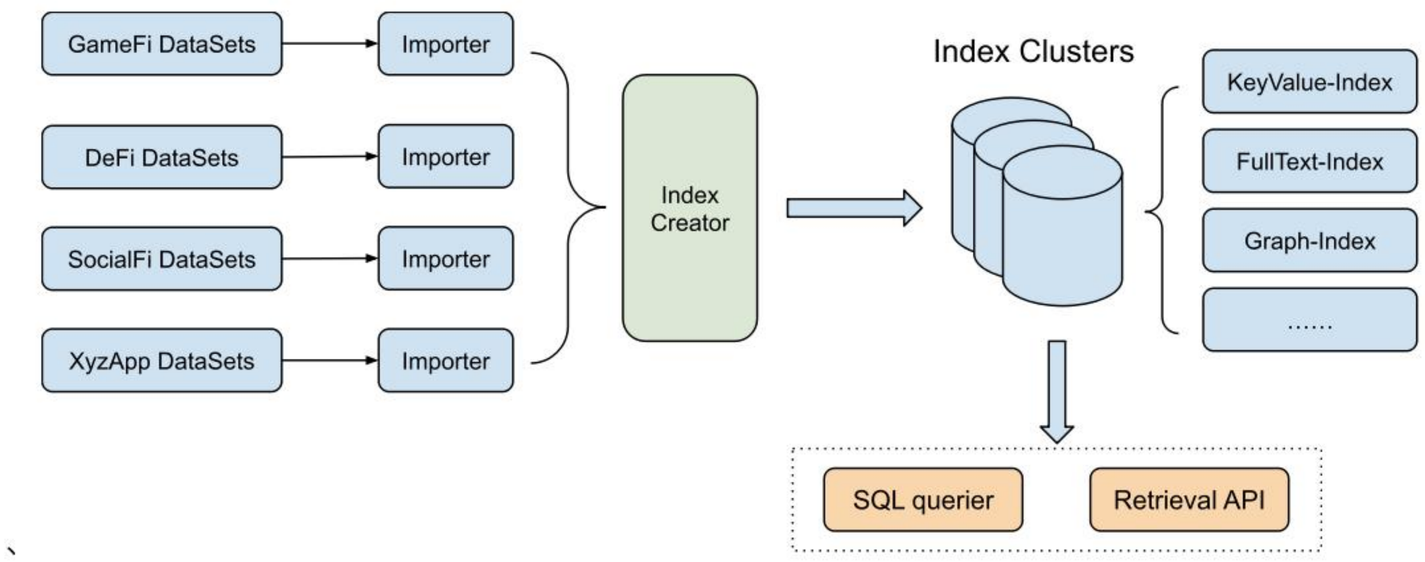
operation to the local Database Engine.

Finally, when the write requests of the Glacier DB Shard Leader accumulate to a certain amount, it will generate a Data Block and a Data Proof, the Data Block will be pushed to Data Availability Network and made public, then it will be written to the ArWeave Bundler. ArWeave Bundler does not immediately write Blocks to ArWeave, but performs further accumulation, and finally writes to ArWeave in large batches.



4.5 Indexer

Glacier is building a unified index network for all the datasets stored to realize customizable, fast, high-through, and stable API query for data users.



Glacier's Indexing is composed of the following 2 key parts:

- **Importer.** Importer extracts data from source datasets and enforces data type and data validity standards and ensures it conforms structurally to the requirements of the indexer and then loads into the indexer. And once there is a new data change in the data source, it will be captured and synchronized to the Index system at the fastest speed.
- **Index Creator.** Index Creator reads data from Importer according to the index configuration and format set by the user, and creates an index instance.

Index clusters run and manage these user-created indexes and provide them to the SQL Querier and related Retrieval APIs. SQL Querier provides users with a traditional SQL-based query interface. SQL is a standard language for accessing and manipulating databases. As a result, many users can quickly write SQL to complete query tasks.

In addition to SQL, Retrieval APIs also provide REST and GraphQL APIs to respond to more complex data retrieval scenarios.

4.6 Code Demo

Glacier provides a MongoDB-like access interface, allowing Web2 developers to quickly switch to the Web3 world while retaining their previous experience. With just a few lines of code, you can access the decentralized services provided by Glacier.

```
import { ethers } from 'ethers';
import { GlacierClient } from 'glacierdb';

// Init
const client = new GlacierClient(endpoint, {
  provider: window.ethereum
})
const notes = client
  .namespace('myprojects')
  .dataset('mydapp')
  .collection('notes')
```

```
// Insert
notes.insertOne({ name: 'Alice', msg: 'Hello' })

// Read
notes.find({ name: 'Alice' })

// Update
notes.updateOne({ name: 'Alice' }, { msg: 'Hello Glacier' })

// Delete
notes.deleteOne({ name: 'Alice' })
```

5 Glacier Rollup

Glacier will build an L2 and L3 scalable rollup network utilizing the zk-STARKs and Celestia Rollkit to provide the incentive layer for Glacier nodes and Data Monetization dApps, including NFTs Minting Platform and data marketplace.

Glacier leverages zk-Rollup technology to compress multiple transactions into blocks and proofs. These blocks and proofs are then submitted to the Data Availability Network and settlement layer to ensure secure and transparent execution of the transactions. In contrast to optimistic rollups, zk-Rollups can more effectively compress transaction data and verify proofs at a faster rate. This practice is also referred to as validium in some cases. It has achieved the optimal balance between security, transparency, and cost-effectiveness. Anyone can determine the validity of transactions published on ARWeave by publishing zk-Dataset Proof on Settlement layer and directly ignoring invalid or malicious transactions.

Glacier's state is specific to each Dataset, which represents a set of ordered, valid database operations from its creation. The Glacier state is not a balance, but rather a hash of the valid operations within the Dataset. To maintain the order and determinism of operations (e.g. updateMany or deleteMany), each Glacier Dataset has a unique, increasing nonce that guarantees the ordered execution of operations.

The Glacier network state is represented by the Merkle tree, which consists of each valid Glacier Dataset state. Each Dataset is isolated from other Datasets, and their unique states contribute to the overall network state. This approach ensures that the network state is transparent and secure, and anyone can verify the validity of transactions published on ARWeave by providing the zk-Dataset Proof, which proves the execution of valid operations within the Glacier Dataset.

Finally, to simplify network deployment and interaction, Glacier uses zk-STARKs, which require no trust setup and offer faster verification. Additionally, Glacier can mitigate the size of proof data by merging multiple Glacier blocks and submitting only one proof to the Settlement layer.

GLC token is the utility token of Glacier protocol and is designed to mainly serve the following goals:

- Incentivizing the data node
- Access to services

- Avoiding the network spam
- Storage Fees
- Sequencer / Indexer Incentives

6 Key Features

To better solve the structure data storage problems of Web3, Glacier will be developed as a NoSQL database protocol for Web3 developers with the following innovative features:

- **Scalability:** The number of Glacier storage nodes can be expanded indefinitely, and the storage capacity of each node can also be flexibly changed.
- **Security:** The stored data is divided into fragments and allocated in multiple Glacier nodes by methods such as zero-knowledge proof and private network key access.
- **Interoperability:** Full interoperability to the Web3 data and make the Web3 data easily queried, transferred and interoperated.
- **Immutability:** More than just tamper-resistant. Once stored, data can't be changed or deleted.
- **High Query Reliability:** Write and run any Glacier Database query to search the contents of all stored transactions, assets, metadata and blocks on the Glacier network.
- **Low cost:** Glacier will provide blockchain database services at a low cost for Web3 developers.

7 Use Cases

1) Database NFT

The NFT space is growing at a rapid pace today, and Glacier will be one of the most professional and friendly database protocols used to store NFT metadata, as the metadata of NFT is also in the NoSQL data structure. NFT will be the most important user scenario and Glacier will expand the adoption in the NFT space first.

2) SocialFi

SocialFi dApps mainly deal with the social connections of Web3 natives and have great needs in storing the Web3 natives' social connection data in a structural way. Therefore, Glacier will help the SocialFi developers achieve that in a friendly and low-cost way. At the same time, SocialFi developers could also use the Glacier database engine to easily build a social graph for its users since they could store the user data in a graph database which is supported by Glacier.

3) Metaverse

GameFi projects can have thousands of NFT items in their blockchain games, meaning that they have a huge burden to store the related asset data and user data. Glacier's NoSQL database engine could help GameFi developers store

them in a structural way in Glacier's nodes and the Arweave chain. At the same time, they could use Glacier to easily update and query the related asset data and user data by leveraging Glacier's SDK.

4) Data Monetization

In Web2 applications, all the user data is stored in the storage space provided by a single technical giant, like Apple, Facebook, or Twitter. They don't have any desire to share their own user data with other companies to build a universal social graph that could be of tremendous value to society.

In the Web3 world, everything is different. The user and asset data is all public on the blockchain. Therefore, Web3 has the chance to make the data fully open and liquid between different individuals or entities. Glacier's vision is to fully unlock the data monetization potentials in Web3 by utilizing the NFTs, which could represent the ownership and query rights of the Web3 data. In Glacier, everyone could easily manage their own Web3 data, including social connections or crypto asset information, and monetize them in NFTs which could be traded on the Glacier Marketplace.

8 Future Work

8.1 Decentralized Identity

In Web3, the data is owned by an on-chain address that has the owner key of the related data. However, just the on-chain address isn't enough to identify the profile or record the identity and social graph. What is worse is that everyone could easily set up hundreds of on-chain addresses in seconds. This means that someone is required to integrate all the on-chain data to form a one-chain Identity, which could be used in the logins, social connections, reputation system and credit finance of the Web3 space.

8.2 Data Cross-Rollup

Today, crypto assets are easily transferred between Ethereum mainnet and Rollups, while the data on different Rollups are still unable to realize the Cross-rollup transfer or interoperability, which greatly limits the integration and potential of Web3 data. Therefore, Glacier will research how to achieve the cross-rollup transfer between different Rollups to make the data flow easily in the tens of Rollup chains in the future.

8.3 Graph Database for Web3 Social

Web3's openness creates an excellent foundation for building a user social graph which has not been achieved in Web 2.0 due to the data barriers. When creating the social graph in Web3, developers have to deal with a great load of on-chain data which is not structurally stored on the blockchain nodes. In Web2, the graph database, which is one of the important NoSQL database types, is used to store the user's social connection data to create a social graph. Therefore,

Glacier will consider releasing a dedicated Graph Database SDK to help Web3 developers create the social graph more easily.