Shelby: Decentralized Storage Designed to Serve

Aptos Labs and Jump Crypto

Abstract

Existing decentralized storage protocols fall short of the service required by real-world applications. Their throughput, latency, cost-effectiveness, and availability are insufficient for demanding workloads such as video streaming, large-scale data analytics, or AI training. As a result, Web3 data-intensive applications are predominantly dependent on centralized infrastructure.

Shelby is a high-performance decentralized storage protocol designed to meet demanding needs. It achieves fast, reliable access to large volumes of data while preserving decentralization guarantees. The architecture reflects lessons from Web2 systems: it separates control and data planes, uses erasure coding with low replication overhead and minimal repair bandwidth, and operates over a dedicated backbone connecting RPC and storage nodes. Reads are paid, which incentivizes good performance. Shelby also introduces a novel auditing protocol that provides strong cryptoeconomic guarantees without compromising performance, a common limitation of other decentralized solutions. The result is a decentralized system that brings Web2-grade performance to production-scale, read-intensive Web3 applications.[1]

1 Introduction

Web3 has made major strides in decentralized payments and trading use cases. Despite this progress, the ecosystem still lacks the core infrastructure needed to scale: decentralized cloud services. Without high-performance alternatives to cloud storage, networking, and compute, Web3 applications hit hard scalability limits, constraining both what they can do and who they can serve.

A critical missing piece is *hot storage*: infrastructure that can support low-latency, high-throughput reads. Use cases like video streaming, real-time collaboration, and AI inference all rely on fast, reliable access to large datasets. While protocols like Filecoin [5], Arweave [1], Walrus [13] and Celestia [7] support archival or cold storage, none can meet the performance requirements of dynamic, read-heavy applications.

The barrier is not just technical - it is economic. Reads are bandwidth-intensive, latency-sensitive, and difficult to meter in a decentralized system. Without a viable economic model for rewarding high-performance reads, most systems avoid the problem entirely, focusing instead on infrequent writes or cold data storage. As a result, truly decentralized applications that depend on fast reads continue to rely on centralized infrastructure.

To ground this challenge in a real-world context, consider video streaming - a canonical example of a read-heavy workload where latency and throughput are paramount. Supporting high-quality 4K video playback requires sustained throughput of at least 40 Mbps, typically delivered in 10 MiB chunks with minimal startup delay. No existing decentralized storage network satisfies this read-throughput requirement (see table 1 for a partial list).

Beyond media, modern AI workflows such as retrieval-augmented generation (RAG) systems depend on low-latency access to diverse, high-quality datasets. These models are rapidly moving from research to real-world use: clinicians training diagnostic tools on medical images, governments building domain-specific assistants, and robotics teams teaching home agents new tasks. The expected utility of these systems is large, but realizing it requires two conditions: data must be easy to monetize with fair compensation, and access to that data must be fast and efficient. A data explosion is underway, and while Web3 is uniquely positioned to preserve user control, current infrastructure cannot meet the performance and economic demands of this shift.

[1] Disclaimer: For convenience and consistency, this paper describes Shelby in the present tense. As a result, this paper includes certain forward-looking statements based on the Shelby design and expectations of the authors as of the date of this paper, but please note that future facts or operation of the Shelby protocol may differ from the content expressed herein.

Shelby is a decentralized storage network purpose-built for high-performance reads. It combines cryptoeconomic incentives with engineering principles drawn from high-performance Web2 systems to deliver fast, reliable access to data without sacrificing decentralization. Shelby combines a novel auditing protocol, efficient erasure coding, and a dedicated fiber network to support Web2-grade performance at Web3 trust levels. Its architecture establishes a strong game-theoretic equilibrium in which all participants - storage providers, RPC nodes, and clients - are incentivized to behave honestly and serve data quickly. At its core, the system is built on six tightly integrated pillars, each contributing to its decentralized performance and economic sustainability.

Paid reads - aligned performance incentives. When users pay for a service, they expect it to be high quality. By requiring payment for reads, Shelby aligns the incentives for users and service providers. Everyone in the ecosystem wants more data to flow: users want a great experience streaming high-quality video or accessing large files and service providers want to serve as much data as they can.

Dedicated fiber network - high-performance network infrastructure. Unlike the public internet - where congestion, variable latency, and unpredictable routing can degrade performance - a dedicated backbone provides Shelby with consistent bandwidth and low-latency guarantees critical for real-time data access. By deploying storage and RPC nodes directly onto this network, Shelby offers users sub-second access latency and reduces bandwidth costs dramatically - two requirements essential for hot storage and applications like streaming and real-time collaboration. Such a network is critical to reach performance comparable to Web2 cloud providers.

Battle-tested software stack - proven infrastructure for low-latency storage systems. Shelby's software stack is built on engineering principles honed through years of experience developing storage and compute systems for Jump Trading Group's high-performance quantitative research infrastructure and Meta's global-scale platforms. This includes high-performance I/O pipelines, efficient concurrency primitives, and low level code optimizations. The result is a system architecture that can handle bursty workloads, maintain sub-second level responsiveness, and scale horizontally with minimal overhead.

Efficient Coding Scheme - high durability with low storage overhead. A core challenge in decentralized storage is achieving cost efficiency without sacrificing durability. That is, reducing replication overhead while being able to tolerate and efficiently repair significant amounts of data corruption. To make this viable, Shelby uses Clay codes [20], an advanced erasure coding scheme that achieves theoretically optimal repair bandwidth (i.e., Minimum Storage Regenerating) without compromising reconstruction capability (i.e., Maximum Distance Separable). This balance is what enables Shelby to operate with a much lower replication factor than typical Web3 storage networks. While many decentralized systems rely on full replication or less efficient erasure codes - often resulting in 5x to 8x overhead - Shelby achieves durability with a replication factor under 2x. This represents a major cost advantage and brings Shelby close to the 1.2x-1.4x replication range common in high-performance Web2 storage systems.

Incentive-Compatible auditing - economic security without performance trade-offs Shelby's auditing protocol is a hybrid design that combines high-frequency internal audits with low-frequency, cryptographically enforced on-chain verification. Storage providers are continuously challenged to prove possession of random data samples, while their audit behaviors are themselves audited on-chain. This "audit-the-auditor" mechanism allows Shelby to significantly reduce the auditing cost by performing most audits internally, while ensuring that no participant can game the system through collusion or apathy. Importantly, this structure is fully incentive-compatible: honest participation maximizes rewards, while misbehavior leads to slashing and revenue loss. This ensures a strong equilibrium where nodes store data reliably, respond to reads quickly, and verify one another in a decentralized yet economically rational way.

Aptos Blockchain - fast and reliable coordination layer. Shelby uses the Aptos blockchain as its coordination and settlement layer. Aptos offers high transaction throughput, low finality times, and a resource-efficient execution model, making it an ideal substrate for managing Shelby's economic logic. All

System	**Perf. architecture**		**User experience**		**Access control**	
	Ded. fiber network	Replication overhead	4K streaming throughput	Web2 cost competitiveness	Incentivized reads	Decentralized
AWS S3	✓	est. 1.4x	✓	✓	✓	✗
GCS	✓	est. 1.4x	✓	✓	✓	✗
Filecoin	✗	3–6x	✗	✓	✗	✓
Greenfield	✗	2.5x	✗	✗	✓	✓
Celestia	✗	4x [7]	✗	✗	✗	✓
Walrus	✗	4.5x [13]	✗	✗	✗	✓
Arweave	✗	15x [8]	✗	✗	✗	✓
Shelby	✓	$< 2x$	✓	✓	✓	✓

Table 1: Qualitative comparison of storage systems across performance, user experience, and decentralization properties. Shelby is the only fully decentralized protocol with the performance architecture needed to deliver Web2-grade read experiences at competitive cost.

critical state - including storage commitments, audit outcomes, micropayment channel metadata, and system participation - is recorded and enforced via the Shelby smart contract on Aptos. This allows for decentralized governance, verifiable incentives, and strong fault tolerance without compromising scalability.

Building on the earlier video streaming example, we now demonstrate how Shelby's design enables real-world performance. A creator uploads a high-resolution video using the Shelby client SDK, which encodes the file with Clay erasure codes and submits a cryptographic commitment to the Aptos blockchain. The client works with an RPC node to distribute encoded chunks across globally dispersed storage providers connected via a dedicated network backbone. When a viewer presses "play," the RPC node fetches the necessary chunks, reconstructs the video on the fly, and streams it with low latency. Micropayments flow in real time to compensate both storage and RPC providers. In the background, the auditing protocol ensures that data remains intact, available, and verifiably stored.

Unlike Web2 walled gardens, where creators relinquish control over content and monetization, Shelby-backed streaming dApps enable creator-owned business models. With fast reads and native micropayments, creators can embed custom interstitials, run their own license servers, or enforce playback restrictions via DRM. This unlocks new monetization paths, from direct sponsorships to platform-independent distribution, while infrastructure providers are rewarded proportionally to their contributions.

In short, Shelby brings Web2-grade performance to Web3 infrastructure - not by compromise, but through principled engineering, incentive-aligned economics, and a deep understanding of what it takes to build a cost-efficient, highly scalable system.

Section 2 presents a high-level picture of Shelby's architecture, while Section 3 goes into more detail on a selected number of important primitives, such as micropayment channels, erasure coding, and data preparation. Sections 4 and 5 discuss Shelby's hybrid audit scheme and economic incentives, respectively. Finally, Section 6 presents some future use-cases for Shelby.

2 System Architecture

Shelby is comprised of four service layers: the client SDK, the RPC node layer, the storage provider (SP) layer, and the coordination layer. Figure 1 shows an example system diagram.

Clients store their data using the SDK. It prepares data for storage and transfer it to an RPC node, which manages the write process. Clients can also set up micropayment channels for reading data from Shelby using the SDK.

Data transfers and end-user payments all occur at the RPC node layer. RPC nodes encode and disperse user data to the storage node layer during writes and gather and decode data during reads.

The SP layer is responsible for storing data on behalf of users. SPs also conduct peer-to-peer audits of data. Storage providers are built with very large amounts of local storage. Shelby provides application-level data integrity, so SP operators do not need to provision RAID arrays or other data integrity solutions.

Finally, Shelby coordinates internal state via the Aptos blockchain. The Shelby smart contract manages several functions on behalf of the system: SP participation, data placement and lifecycle, cryptoeconomic security rewards and punishments, as well as payment flows.

2.1 User Data: Blobs, Chunksets, Chunks, and Samples

- **Blobs** are arbitrary-sized Binary Large Objects such as images or videos.
- **Chunksets** are fixed-size portions of a Blob, roughly 10 MiB.[2]
- **Chunks** are fixed-size (roughly 1 MiB) portions of a Chunkset, formed by erasure-coding.
- A **Sample** is a small (around 1 KiB), fixed-size portion of a Chunk for use in auditing.

The Client SDK reads and writes Blobs, but also allows byte range reads for efficiency. Internally, these Blobs are partitioned and specially prepared for storage on the set of storage providers. Users pay to store data for a certain duration and Blobs stored in Shelby are immutable.

2.2 Client SDK

Clients use the SDK to perform functions such as establishing and managing payment channels, preparing data for upload, and writing and reading data from Shelby. The Client SDK contains a CLI and libraries able to integrate into user-facing dApps.

Writing data. When writing data, the Shelby Client SDK erasure codes the Blob and computes cryptographic commitments on the encoded data. The SDK then stores Blob metadata on the coordination layer by calling smart contract functions. The user pays to store the Blob for a specific duration. Once the coordination layer has stored the metadata and transferred payment, the SDK contacts an RPC node to transfer data. Once the RPC node finishes the write procedure, it changes the Blob metadata on the coordination layer, marking it stored in Shelby and ready for reading.

Reading data. The first step for reading from Shelby is to use the Client SDK to establish a micropayment channel with the RPC node. Thereafter, clients sign micropayment transactions in the channel mixed with data reads from the RPC node.

2.3 RPC Nodes

RPC nodes are the gateway to the Shelby storage system. They offer an API which, in concert with the Client SDK, enables clients to write and read data. Furthermore, the RPC nodes call functions on the Shelby smart contract on the coordination layer to read and write Blob metadata. Finally, RPC nodes call the Shelby SP API to write and read Chunks of data from the storage providers.

Writing data. During the Blob write procedure, the RPC node checks cryptographic commitments in the Shelby metadata and sends the encoded Chunks to the SP layer. The specific SP nodes are assigned by the smart contract. After the SPs have acknowledged the data is received, the RPC node uses the coordination layer to mark the Blob ready for reading.

Reading data. RPC nodes pay for reading data from SPs. When joining the network, RPC nodes establish payment channels to the SP layer. RPC nodes read Chunks from SPs and reconstruct the requested range of Blob data. Since Blobs are cryptographically committed, attempts to alter data will be detected.

[2] Files that are smaller than roughly 10 MiB are zero-padded. As a result, very small files incur overheads, and we expect clients will store them together to lower those overheads.

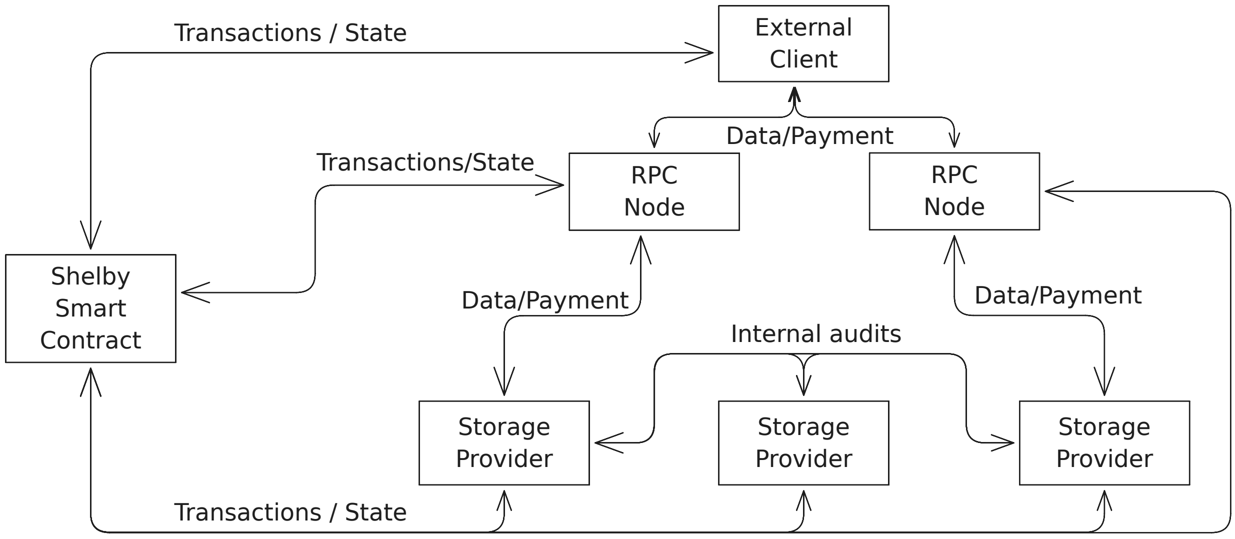

Figure 1: System Diagram

2.4 Storage Providers

Storage providers are the core of Shelby. An SP stores data on behalf of users and audits other storage providers to ensure data durability and availability. Similar to RPC nodes, SPs use the coordination layer to follow Shelby metadata and call smart contraction functions.

Writing data. During data writes, SPs track the state of the Shelby smart contract to ensure they know what data Chunks are assigned to them.

Reading data. During data reads, SPs respond to paid requests for data by returning the requested Chunk and cryptographic commitment.

Auditing data. Periodically, SPs are required to conduct a series of audits. As auditor, an SP requests a Sample and cryptographic commitment from an auditee storage provider. The auditee responds with the requested Sample and the proof of data. Auditors store proofs locally in case their audit records are selected for verification by the Shelby smart contract. Audit records that are selected for verification by the Shelby smart contract are submitted directly to the Shelby smart contract using the coordination layer. More details are given in Section 4.

2.5 Shelby Smart Contract

Deployed on the Aptos blockchain, the Shelby smart contract assigns Chunks to SPs during the Blob write procedure. It manages Blob metadata including lifecycle and accepts payments from users. It sets audit schedules for data and SPs. And ultimately, the smart contract contains the logic to punish bad actors in the system when they have provably acted in a byzantine fashion (either by crashing, censoring, or lying about data).

Writing data. During writes, Shelby a client submits payment and a succinct cryptographic commitment of their Blob to the smart contract. The smart contract randomly assigns Chunks to SPs, and this mapping is stored in the Blob metadata. Once the Chunks are accepted by the SPs, the Blob is marked ready by the RPC node. After this point in time, the data is durably stored and performantly retrievable.

Reading data. Shelby's high throughput data reads do not require any fine-grained coordination from the blockchain layer. Rather, micropayment channels allow for scalable system performance with loose global coordination, a key technique of high performance computing. Clients can also use micropayment channels with the RPC node layer, although RPC nodes may choose to support many payment mechanisms to encourage Shelby usage.

3 System Details

This section presents some of the low-level system details used to construct Shelby. While not required for the presentation of the high-level system architecture in Section 2, these details enable Shelby to hit the desired system characteristics of low cost and high performance, durability, and availability.

3.1 Dedicated Network Layer

Web2 competitive price-performance requires Web2 physical infrastructure. Web2 providers use private, dedicated high-performance networks for efficient data transfer at reasonable cost. Similarly, Shelby uses a dedicated high-performance network layer to achieve price-performance not possible for Web3 operating over the public internet. Shelby uses this layer to achieve a Byzantine fault tolerant system without requiring application-layer reliable broadcast mechanisms over the public Internet. DoubleZero [2] recently emerged as a decentralized option for a suitable network layer.

3.2 Micropayment Channels

Shelby is designed for high-throughput, low-latency data serving. As such, RPC nodes and storage providers (SPs) cannot mix on-chain payments while serving data. There are many off-chain payment options for end users (credit cards, monthly subscriptions, etc.), and RPC nodes can offer the best different payment mechanism for their user community.

As a default, Shelby offers micropayment channels [14] as a payment mechanism between clients and RPC nodes and RPC nodes and SPs. Micropayment channels enable off-chain transfers to occur in an optimistic manner allowing high scalability. Since on-chain transactions are required only for creation and settlement, micropayment channels also allow for small payments with low overhead cost.

While the Aptos blockchain offers industry-leading low-latency consensus, micropayment channels provide an additional performance and cost advantage for high-frequency, peer-to-peer payments. Because they bypass consensus entirely, micropayment channels support payments to be freely mixed with data reads, which can be an essential feature when small reads must be extremely cheap to support responsive, usage-based incentives.

Very briefly, a basic unidirectional micropayment channel between a client and a server requires funds to be transferred from the client to a multi-signature account signed by both parties. The server also sends the client a refund transaction which the client can use to regain its funds after a certain amount of time. The client makes payments by sending new refund transactions with a slightly smaller refund to the client and a slightly earlier allowed settlement time. If either party stops cooperating, the most recently signed refund transaction can be used to settle the channel before the other refund transactions become valid.

Micropayment channels are not perfectly trustless, but uncooperative parties will quickly stop being paid for invalid service. Hence, the expected value at risk is small.

3.3 Erasure Coding

Shelby uses Clay codes [20] (Coupled-Layer codes) to prepare user data for storage on SPs. The choice of an erasure coding scheme impacts system performance characteristics such as write and read speed, data storage durability and availability. Web2 storage systems are attractive to users because they offer high durability (data has an extremely low likelihood of data loss) and availability (data can be read *right now*) at reasonable costs and performance levels. For Shelby to achieve the same levels of data storage durability and availability requires an erasure coding scheme fit for the task. ([17] surveys modern erasure codes.)

Erasure-coding schemes divide data into k blocks and encode it into n blocks (where $n > k$) to add redundancy. Storage systems then store the n erasure-coded blocks onto different nodes. The ideal erasure coding scheme for a storage system minimizes the bandwidth and the storage for a given durability. That is, the ideal coding scheme is Minimum Storage Regenerating [25][20][21] (MSR) and Maximum Distance Separable [18] (MDS). Clay codes are a practical family of codes with both properties. During data repair, they exhibit 60% less bandwidth usage compared to Reed-Solomon [34] codes. Clay codes also maintain the MDS property where any k out of n total erasure-coded portionare sufficient to fully decode the data. Real-world storage systems such as Ceph [30] have performance benchmarks that demonstrate the advantages of Clay codes.

Clay codes are ideal for Shelby because they balance performance, durability, and availability characteristics:

- **Computationally efficient**: Clay codes have efficient encoding and decoding, avoiding excessive CPU time as a performance degrading bottleneck.
- **High performance for target data size**: Shelby is designed to work well for Blobs of at least 10 MiB. Clay codes work best when the data being coded is sufficiently large to be divided into the required number of sub-units without excessive overhead. As such, Shelby has good storage efficiency for the targeted data size.
- **Flexible Repair Structure**: Clay codes can achieve optimal repair band provide optimal repair efficiency when using a specific ordering of repair actions. When the optimal repair pattern cannot be followed, Shelby can fall back to the MDS property (where any k chunks can recover data) even if it must temporarily sacrifice repair bandwidth efficiency.
- **Repair Coordination**: Shelby's coordination layer allows planning for bandwidth-optimal recoveries, a key feature of Clay codes.

By using Clay codes to store data, Shelby offers 99.999999999% (11 nines) data durability, and 99.9% availability. Derivations of the durability and availability of Shelby are given in Appendix A.

3.4 Cryptographic Commitments

Shelby uses a Vector Commitment (VC) scheme to enable data verification. In [23], the authors state "VCs allow to commit to an ordered sequence of q values $(m_1, \ldots, m_q)$ in such a way that one can later open the commitment at specific positions (e.g., prove that m_i is the i-th committed message)." In Shelby, we use Merkle trees [33] as the VC scheme to bind the encoded Chunks to a root hash. Thereafter, responses to requests for data include the data validity proof.

Data integrity is a critically important feature for a storage system. Subject to standard cryptographic caveats (i.e., that no known computing method can efficiently find hash collisions), **it is impossible for components of Shelby to alter user data without being detected.**

The choice of VC also allows for efficient auditing of SPs. The root hash of the Merkle tree is stored in the Blob metadata using the Shelby smart contract. Thereafter, Samples and inclusion proofs are used to verify the data by auditors. The computational efficiency of a Merkle tree inclusion proof allows for audits to be computed on-chain. Section 4 has more details on the hybrid auditing scheme.

3.5 Engineering for High Performance

Shelby is built using the experience and engineering techniques that underpin the High Performance Compute (HPC) storage system used by Jump Trading Group. HPC engineering ensures that the system components are optimized to an equally high degree. Amdahl's law requires that for an entire computation to be optimal, all parts of it must be optimal. For distributed systems, this optimization process can leverage a multitude of technologies at each layer of the software application stack:

- **Network:** The use of multicast allows for efficient broadcast of data among the system nodes. The application layer need not concern itself with duplicating data, as the network switch hardware is responsible for ensuring the data is delivered correctly to all members of the multicast group.

- **Packet processing:** Kernel-bypass networking (e.g. DPDK [4]) allows for Linux systems to process packets at much higher rates than through traditional kernel-based techniques. Meanwhile, the eXpress Data Path [15] (XDP) in the Linux kernel offers high performance packet processing without needing to fully bypass the kernel. These techniques can be used to break through bottlenecks caused by packet processing overheads.

- **Storage:** Shelby SPs will seek to maximize their profit. As a result, many or most SPs will use spinning hard drives. While hard drives are not known for extremely high performance, careful planning of data placement and ordering of hard drive seeks to reduce seek time help maximize performance.

- **CPU:** Modern CPUs have high core counts and, as such, act as tiny distributed systems with internal networks [32]. As such, these networks can become congested if dataflow is not carefully designed. Further, the physical design of high core count CPUs requires tradeoffs to be made around memory latency. On-chip cache and main system memory exhibit different latency characteristics due to Non-Uniform Memory Access (NUMA). In particular, multi-socket motherboards are incapable of offering the same latency to main memory from all sockets, due to the speed of light in Printed Circuit Board (PCB) wire traces. For all these reasons, topology-aware application design is thus an important concept to consider for HPC engineering.

- **Memory efficiency:** Avoiding unnecessary overheads for data sharing is important for achieving full CPU utilization. The use of lock-free data structures that avoid false sharing of CPU cache enables task parallel processing to make optimal use of on-chip resources.

- **Data parallelism:** Modern CPUs have significant on-chip resources devoted to data parallel computation. By designing applications for data parallelism from the beginning, the full usage of vectorized instructions unlocks the power of modern CPUs.

- **Erasure coding acceleration:** Modern CPUs have accelerated instructions for the Galois field modular arithmetic required to perform erasure coding, and there are advanced algorithms [19] which can attain encode/decode rates that are in excess of the bandwidths available on network cards.

- **Shelby-internal communication:** Distributed systems comprised of individually optimized components need optimal inter-node communication patterns as well. HPC techniques ensure that dataflows avoid local bottlenecks which compromise global system performance. High performance systems use "request hedging," in which requests are sent to all servers that can satisfy the request. The first response is used, and the remaining requests are canceled and in-flight responses are ignored. This approach wastes some amount of resources, but can prevent temporary hotspots from dramatically affecting the tail latency of a system.

The use of HPC principles in Shelby enables read performance to scale with the available hardware. The throughput of the system is dictated by the aggregate bandwidth of the RPC node layer.

3.6 Data Preparation Example

Figure 2 shows an example data preparation of a user file. The Client SDK partitions the Blob into Chunksets (if the Blob is larger than a single Chunkset). Since it is unlikely the Blob will be partitioned perfectly into the 10 MiBs, the final Chunkset is zero-padded up to the correct length. Then, each Chunkset is erasure coded with Clay codes, which forms some number of Chunks. Each of those Chunks is then cryptographically committed, as is the full Blob. All of these cryptographic commitments are included in the transaction the Client SDK commits to the Aptos blockchain during the write procedure.

4 Audit and Incentive Compatibility

Overview. The audit subsystem in Shelby is designed to ensure that storage providers (SPs) reliably store the data they have committed to, and to verify that peer auditing activity (i.e., SPs auditing each other) is conducted honestly. The protocol follows a hybrid design that combines high-volume, low-cost *internal*

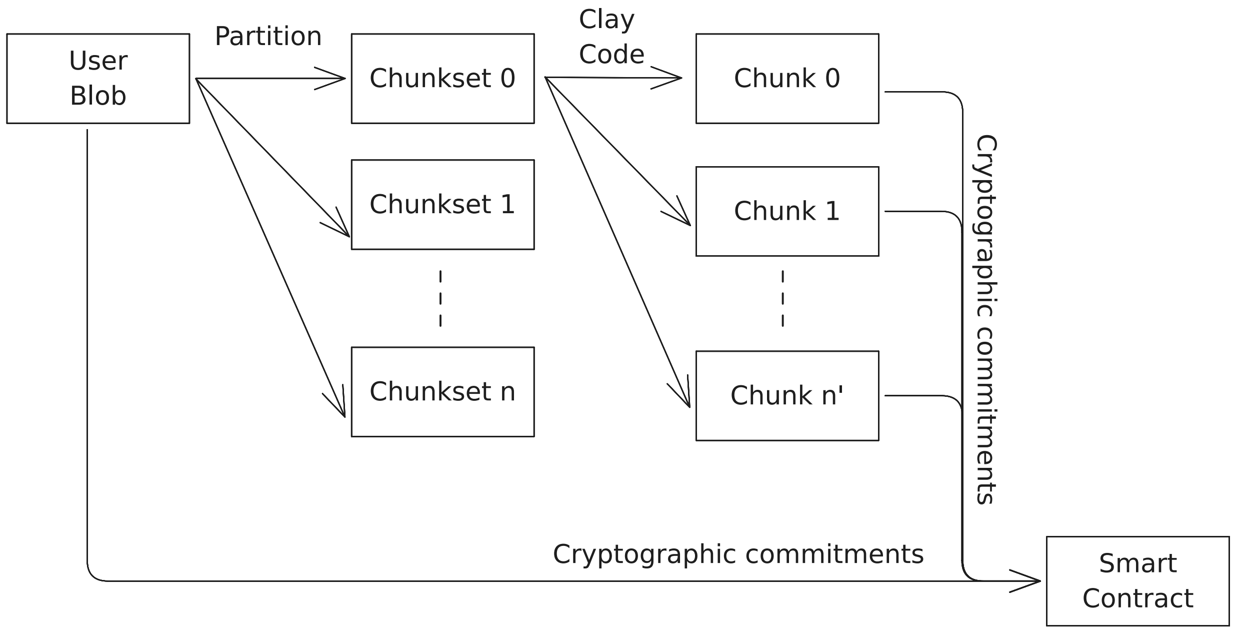

Figure 2: Data preparation by the Client SDK

audits with low-frequency *on-chain audits* that also serve to audit the auditors. This architecture enables strong guarantees for both Byzantine Fault Tolerance (BFT) and Incentive Compatibility (IC), while keeping cost and performance overheads tractable at scale.

In Shelby, *internal audits* are lightweight, peer-to-peer checks performed off-chain, while *on-chain audits* are verifiable challenges enforced by the blockchain smart contract. Ths hybrid audit design is motivated by the complementary limitations of purely on-chain and purely internal auditing. On the one hand, on-chain audits provide strong cryptoeconomic guarantees, but they are inherently constrained by gas costs, bandwidth requirements, and the system-wide consensus overhead of executing audits through the Aptos validator set. These constraints make it impractical to run on-chain audits at the volume needed to enforce good behavior at scale. On the other hand, internal auditing mechanisms, while cheap and performant, are vulnerable to rational deviations. Without an external enforcement layer, SPs can adopt collusive strategies - such as universally reporting success or ignoring audits entirely - that maximize local utility while undermining system integrity. Shelby's hybrid design seeks to reconcile these trade-offs by using internal audits to detect most misbehavior and on-chain audits to impose accountability where it matters.

All actively punitive actions in the system - such as slashing misbehaving SPs or penalizing false audit reports - are carried out on-chain, which protects SP minorities from being slashed by a majority. The audits procedure is divided into predetermined audit epochs. At the end of each audit epoch, the blockchain performs two key roles:

1. It audits SPs whose behavior appears suspicious based on internal audit outcomes, issuing randomized challenges to verify storage correctness.

2. It audits the auditors by requiring them to reproduce a random subset of the audit proofs they claimed to have verified, thereby validating the integrity of internal audit attestations.

These on-chain audits are designed to be sparse but unpredictable, serving as a forensic tool rather than a primary enforcement mechanism. By applying external scrutiny only to a strategically chosen subset of actors and actions, the system achieves robustness under standard Byzantine Fault Tolerance and Game Theory assumptions, while preserving scalability and economic efficiency.

4.1 Internal Auditing Protocol

Internal audits in Shelby are designed to operate at high frequency and low cost, enabling continuous verification of SPs behavior without incurring the overhead of consensus-layer operations. The mechanism relies on publicly verifiable randomness from the Aptos blockchain to assign storage challenges to SPs throughout each audit epoch. Each challenge corresponds to a small, randomly selected portion of a stored Chunk - typically a 1 KiB Sample – whose possession must be demonstrated by the responsible SP.

Upon receiving a challenge, the designated SP generates a succinct cryptographic proof of possession (e.g., a Merkle proof) and broadcasts this proof to the rest of the network[3]. All SPs maintain a local *scoreboard*, which records whether each peer has responded correctly to its assigned challenges. These scoreboards serve as the basis for reward distribution and act as the primary input for determining which SPs are subject to further on-chain audits.

To ensure accountability and auditability, each SP retains all audit proofs it receives for the duration of two audit epochs. At the end of each epoch, SPs publish a compressed version of their scoreboard to the blockchain. Each scoreboard is comprised of $(n-1)$ bit vectors, where n is the number of SPs and the length of the i's vector is the number of audits that SP i was challenged with during the epoch. In typical operation, these bit vectors are dominated by successful audits and thus exhibit high regularity - either sparsity or uniformity - making them amenable to efficient compression. This allows the full scoreboard to be submitted on-chain with modest bandwidth and gas costs.

Incentives for participating in internal audits are structured as follows. Rewards for the audit epoch are divided into two pools:

- **Storage rewards**, which are proportional to the volume of data stored and scaled by the provider's audit score. The audit score reflects how consistently the SP has responded correctly to internal audit challenges during the epoch.
- **Auditor rewards**, which provide a small, fixed payment for each successful audit reported by the SP. These rewards are independent of how much data the SP stores.

Let $score_i \in [0,1]$ denote the audit score for SP i, derived from the set of bit vectors submitted by its peers. To mitigate the influence of Byzantine or otherwise erratic behavior, the highest and lowest third of evaluations are discarded prior to aggregation - a standard resilience technique under the assumption that at most one-third of nodes may be faulty. The remaining evaluations are aggregated (e.g., via majority vote or weighted average) to compute $score_i$. The allocated storage rewards are then scaled by $score_i$. Additionally, if SP i was assigned a_i audits as an auditor and successfully verified s_i of them, it receives $s_i \cdot rwd_{au}$ in audit rewards, where rwd_{au} denotes the fixed reward for each successfully reported audit (i.e., a '1' entry in the scoreboard).

This structure positively incentivizes SPs to (a) store their assigned data and (b) perform audits faithfully and in a timely fashion. Importantly, it does not create a zero-sum game among SPs: one provider's underperformance does not automatically increase rewards for others. The system rewards independent correctness rather than relative rank.

4.2 On-Chain Auditing of the Auditors

The on-chain auditing mechanism in Shelby serves as a low-frequency, high-integrity enforcement layer to ensure honest participation in internal audits. It targets both storage providers (SPs) suspected of underperformance as well as verifying auditors correct reporting. While the majority of auditing activity and scoring is conducted off-chain, *actively punitive measures* - such as slashing - are triggered on-chain and rely on cryptographic evidence. These slashing events may originate either from on-chain audit challenges or from SPs submitting verifiable evidence of protocol violations encountered during internal audits.

At the end of each audit epoch, the Aptos blockchain uses its native randomness to assign two types of on-chain audit challenges:

[3] In practice, broadcast is implemented using scalable multicast protocols (e.g., to subcommittees), but for simplicity we describe the mechanism in terms of full broadcast.

1. **Auditee audits**: SPs with low audit scores are subject to direct on-chain challenges testing their storage integrity. For an SP with score $score_i \in [0,1]$, the number of challenges assigned is $(1 - score_i^2) \cdot C$, where C is a configurable system parameter. This quadratic penalty scheme ensures that well-performing providers are audited infrequently, while those with poor records face increasing scrutiny.
2. **Auditor audits**: SPs are challenged to reproduce audit proofs corresponding to a randomly selected subset of the 1 entries in their published scoreboards. Each such entry asserts that the SP verified a peer's proof during the internal audit phase. Failure to produce a valid and matching proof results in slashing for false reporting.

Auditee challenges are constructed by sampling random Samples from the SP's declared holdings, and each challenge requires a cryptographic proof of possession - typically a Merkle membership proof. These proofs are submitted on-chain and verified by a smart contract. Auditor audits are symmetric in structure: the challenged auditor must submit the audit proof it previously claimed to have verified and stored. The contract checks consistency against the scoreboard and validates the cryptographic correctness of the proof.

In addition to these scheduled on-chain audits, SPs are also permitted to submit evidence of invalid audit responses observed during internal operation. If an SP detects a malformed or provably incorrect audit proof from a peer, it may post this evidence on-chain to initiate slashing. In such cases, the reporting SP receives a portion of the slashed funds as a reward, aligning incentives for active monitoring and detection. Moreover, since honest participation requires only local computation and storage, the cost of compliance remains low relative to the risk and penalty of deviation.

All on-chain audits are probabilistic and non-deterministic from the perspective of SPs. This unpredictability, coupled with the possibility of peer-initiated slashing, ensures that rational SPs have strong incentives to store data, retain audit proofs, and participate honestly in internal audits. Because enforcement relies on objectively verifiable data and does not depend on subjective peer consensus, the protocol avoids vulnerabilities associated with collusion or false majorities.

This mechanism disrupts incentive-incompatible equilibria in which SPs collude or neglect their responsibilities. Even if the likelihood of on-chain auditing is low, the cost of failing such an audit - or being reported for provable misbehavior - exceeds the cost of honest participation, making the protocol robust under standard rational and Byzantine assumptions.

4.3 Byzantine Fault Tolerance of Audit Score Computation

The goal of internal auditing in Shelby is to produce an audit score for each SP that accurately reflects its storage behavior, despite the presence of faulty or Byzantine nodes. This subsection outlines the main intuition for BFT correctness.

A standard BFT model assumes a partially synchronous network and a set of n SPs, of which at most $f < n/3$ may behave arbitrarily (i.e., be Byzantine). Correct SPs follow the internal audit protocol as specified.

BFT Intuition. Recall that at the end of each audit epoch, the audit score for SP j is computed from the scores it received from its peers by trimming the highest f and lowest f peer evaluations (inside a peer evaluation a missing '1' equals a '0'). The remaining scores are then aggregated (e.g., via majority or averaging) to obtain a normalized score $score_j \in [0,1]$. Since at most $f < n/3$ SPs are faulty, trimming the top and bottom thirds of evaluations guarantees that the aggregated score is within the range of the highest and lowest honest scores given to j. Honest SPs report audit outcomes based on actual verification, and network noise is modeled within the faulty set. Thus, for a correct SP j, the computed score $score_j$ will closely approximate its true success rate in responding to audit challenges. Faulty SPs, by contrast, will tend to have persistently lower scores, either due to audit failures, missing responses, or detection of incorrect proofs.

Summary. Under standard BFT assumptions, the internal auditing mechanism ensures that:

- Honest SPs receive audit scores close to 1, reflecting faithful storage behavior.
- Faulty SPs receive scores reflecting their true audit failures, and cannot inflate their success rates.

4.4 Game-Theoretic Incentive Compatibility

Establishing incentive compatibility (IC) in decentralized storage systems is nontrivial under standard rationality assumptions. Adopting a more cautious approach requires a list of modeling assumptions and select key justifications for the incentive compatibility arguments. Additional details with proof sketches are in Appendix B.

Rationality. Storage providers (SPs) are assumed to be expected-utility maximizers. Each SP selects its strategy in order to maximize expected payoff, accounting for all rewards, penalties, and operational costs associated with the protocol.

Audit Verifiability. Given an audit proof, its correctness (i.e., whether it corresponds to a valid Sample of the committed Chunk) can be verified efficiently and unambiguously by auditors and by on-chain smart contracts. In Shelby, Merkle proofs are used as the underlying verifiable structure.

Audit Proofs Imply Persistent Storage. An SP producing a valid audit proof for a challenged Chunk retains the corresponding data locally, except with negligible probability. This assumption is justified by the following lemma.

Lemma 1. *[Economic Incentives for Persistent Storage] Let c_s be the per-epoch cost of storing a chunk, c_r the cost of retrieving it externally and p_a the per-epoch probability of an audit.*[4] *If $p_a \cdot c_r \geq c_s$, then the rational strategy for an SP is to retain its assigned Chunk locally rather than deleting it and attempting on-demand retrieval.*

Numerical estimates (see Section 5.4) suggest that with realistic storage and retrieval costs, and under reasonable audit frequencies, the above inequality is satisfied with a substantial margin.

Reward/Penalty Recap. For convenience, briefly recall the relevant reward and penalty structures:
Storage rewards: Proportional to stored volume, scaled by the SP's audit score.
Auditor rewards: An SP earns a reward $rwd_{au} > 0$ for each successful audit reported.
Audit-the-auditor penalties: A 1-entry in a scoreboard is independently selected for verification w.p. $p_{ata} > 0$. Failure to produce the corresponding audit proof results in a slashing penalty S_{ata}.
Slashing rewards: SPs who submit verifiable proofs of peer misbehavior (e.g., invalid proofs) are rewarded with $r_{slash} > 0$.

Denote by π an audit-challenge proof consisting of (i) the challenged Sample (a randomly selected 1 KiB Sample) and (ii) a succinct cryptographic proof verifying its correctness. $p_{inv}(\pi)$ is defined as the probability that the proof π is invalid. To incentivize auditors to ensure that $p_{inv}(\pi) < \varepsilon$ for a small threshold $\varepsilon > 0$, the slashing penalty S_{ata} should satisfy:

$$S_{ata} \geq \frac{rwd_{au}}{p_{ata} \cdot \varepsilon}.$$

Definition 1 (Honest Strategy)**.** *Let σ_i^{honest} denote the strategy of storage provider i during an audit epoch, consisting of two roles:*

- ***Auditee behavior:***
  - ***Persistent Storage:*** *Store all assigned Chunks faithfully throughout the epoch.*
  - ***Audit Proof Generation:*** *Upon receiving an audit challenge, compute the proof π_i and broadcast π_i to peers (or the assigned subcommittee).*
- ***Auditor behavior:***

[4] Conservatively assume that a misbehaving SP always succeeds in performing a timely external retrieval. This strengthens the result since, in practice, the misbehaving SP is also taking the considerable risk of failing to externally retrieve the data in a timely manner.

- ***Audit Verification and Reporting:*** *For each received proof π_j:*
  - *Record success ('1') in the scoreboard if $p_{inv}(\pi_j) \leq \varepsilon$, where $\varepsilon > 0$ is the given threshold; otherwise record failure ('0').*
  - *Retain π_j for audit-the-auditor verification.*
- ***Slashing Proof Submission:*** *Submit invalid proofs π_j on-chain as slashing evidence.*
- ***Publishing Scoreboard:*** *At the end of the epoch, publish the complete peer audit scoreboard.*

There are three important IC properties that Shelby's Auditing scheme satisfies.

(1) Nash Equilibrium of Honest Strategy. Shelby is IC in the most basic sense, as captured by Theorem 1 below.

Theorem 1. *[Incentive Compatibility of the Honest Strategy Profile] Let $\vec{\sigma}^{honest} = (\sigma_1^{honest}, \ldots, \sigma_n^{honest})$ denote the strategy profile where all SPs follow the honest strategy. $\vec{\sigma}^{honest}$ constitutes a Nash equilibrium.*

(2) Mutual Dishonesty is Not an Equilibrium. Systems that rely solely on internal audits suffer from a failure mode where all SPs report universal success without performing actual verification, and potentially without storing any data. In these systems, "mutual dishonesty" often forms a stable equilibrium. In Shelby, the audit-the-auditor mechanism prevents this failure mode by introducing a risk of on-chain slashing for falsely reported audits.

Theorem 2. *[Mutual Dishonesty is Not a Nash Equilibrium] Suppose every SP follows the mutual dishonesty strategy: storing nothing and reporting success ('1') for all audits without requiring proofs from the colluding peers. This strategy profile is not a Nash equilibrium.*

(3) The Honest Equilibrium is Coalition Resistant. Beyond individual incentive compatibility, Shelby's auditing protocol also exhibits the desired property of robustness against coalitions. Specifically, no coalition of up to f SPs can significantly improve their total utility by jointly deviating from honest behavior.

Theorem 3. *[ε-Coalition Resistance] Let $C \subseteq \{1, \ldots, n\}$ be a coalition of SPs with $|C| \leq f$. Suppose all SPs outside C follow the honest strategy σ^{honest}. Then, under the protocol's incentive model, any joint deviation by C improves the coalition's aggregate expected utility by at most an $\epsilon > 0$ margin, where ϵ is negligible relative to standard reward and penalty scales.*

5 Economic Opportunity

This section details the core economic factors underpinning Shelby's storage and retrieval model. The main cost components of providing decentralized storage are physical storage, bandwidth, durability maintenance, and system coordination. These factors are carefully managed to ensure that Shelby delivers read and write performance suited to high-demand use cases, while preserving key Web3 properties such as user ownership and control. The cost structures guide the setting of system parameters - including storage rewards, read pricing, and slashing penalties - to ensure the network remains economically sustainable for both storage providers (SPs) and RPC nodes. SPs earn rewards for storing user blobs and serving read requests; RPCs earn fees for acting as the interface between users and the network. These rewards are calibrated to reflect real-world costs, ensuring that participation is attractive for both roles.

5.1 Write Economics

Storing data incurs several key costs. Physical storage hardware and ingress bandwidth are fundamental contributors [11, 3, 6, 29]. Orchestrating the distribution of data also introduces coordination overheads, while long-term durability requires continuous repair of failed storage nodes or disks [22, 24, 21]. The Shelby storage coding scheme (Section 3.3) directly addresses these challenges by achieving low replication overhead and minimal repair bandwidth, without compromising fault tolerance. This is essential to ensure reliability in any distributed storage system. In a *decentralized* setting, however, ensuring reliability requires an additional cost-auditing-to cryptographically verify that data remains intact and available over time. While many decentralized networks face difficult trade-offs here-often sacrificing either cost efficiency or cryptoeconomic guarantees-Shelby is engineered to maintain strong security and decentralization properties with minimal cost overhead.

The user's storage fee is W per GB per month. This payment is allocated between *storage rewards* (rwd_{st} per GB) and *auditor rewards* (rwd_{au} per successful audit). To minimize payment friction, rewards are accumulated and disbursed at the end of audit epochs, ensuring that payment handling costs are negligible in aggregate. Since rwd_{st} is a monthly rate per GB and rwd_{au} is per audit, allocating between them requires normalization: if a GB of user data is subject to n_a audits per month (on average), then the payout satisfies $rwd_{st}+n_a \cdot rwd_{au} = W$. Here, n_a is derived by multiplying 3 terms (i) the expected number of chunks selected for audit in a single epoch from that a user GB (p_a times number of chunks in a GB), (ii) the number of auditors per audit and (iii) the number of audit epochs per month. The parameters rwd_{st}, rwd_{au}, and p_a are tuned to ensure that SPs maintain healthy profit margins while the system remains secure. Thanks to Shelby's hybrid auditing scheme and real-world cost benchmarking (Section 5.4), there is ample flexibility to set these parameters effectively.

Many existing decentralized storage protocols suffer from chronic underutilization: vast amounts of storage are committed and rewarded, but rarely accessed or used meaningfully. This disconnect stems from rewarding allocation rather than utility-paying for bytes pledged, not bytes served. In contrast, Shelby ties rewards to actual retention and audit performance, ensuring that incentives reflect real value. By doing so, it avoids over-incentivizing idle storage and aligns provider revenue with system usefulness.

Through careful engineering, Shelby is able to offer write pricing that remains competitive with existing Web3 storage solutions, while delivering substantially better performance. Unlike many decentralized systems that focus on archival or cold storage with minimal read optimization, Shelby is designed for high-throughput, low-latency access, making it suitable for performance-critical applications. Importantly, these performance gains are achieved without sacrificing decentralization or cryptoeconomic guarantees, and at a cost structure that remains within a practical range relative to both Web2 and Web3 alternatives.

5.2 Read Economics

Reading from Shelby incurs several key costs. The most significant is egress bandwidth, particularly for data served externally to end users. This is optimized by having RPC nodes decode data (even if redundant) within the dedicated network layer and egress only the minimal required data externally. While further optimization within the internal network is possible and planned, the much higher cost of external (internet-facing) egress makes internal optimizations a secondary priority.

The coding scheme introduces some computational overhead for erasure decoding during reads, but this cost is negligible in both time and money compared to bandwidth and other factors. Payment coordination in a decentralized system can also impose substantial overhead if handled naively. By employing payment channels from RPCs to SPs, Shelby effectively amortizes these costs, enabling micropayments of $\$10^{-9}$-small enough to support seamless, trust-minimized user experiences where only minimal amounts are ever at stake as payments track data delivery in real time.

An often overlooked but crucial cost is maintaining a visible catalog that tracks available data and the metadata required for retrieval. High read performance depends on this catalog being up-to-date and efficiently accessible. Shelby uses the Aptos blockchain to maintain a globally consistent catalog of verified data. However, since reads do not modify the catalog, they do not require direct blockchain interaction. Instead, RPC nodes (or any interested party) maintain a local copy of the catalog, kept current via an Aptos full node. Notably, the same catalog data structure is also used for coordinating the auditing protocol, whose costs are associated with writes-efficiently sharing catalog maintenance costs across system functions.

Reads in Shelby benefit from strong economies of scale. Major costs such as catalog maintenance remain nearly constant regardless of read volume, so the marginal profit per read increases as volume grows. As a result, calibrating the read price R (per GB) is relatively straightforward, relying primarily on predictable, well-understood cost factors. High read volume benefits both the system and content owners, aligning incentives to attract and serve large-scale read traffic with high performance.

5.3 RPC revenues

RPC nodes play an important role in Shelby by serving as the main entry point for users, coordinating data retrieval and enhancing the overall user experience. They require a connection to the private backbone network, an external-facing internet connection to serve user requests, and standard server-grade hardware. Importantly, most of these costs are variable and scale directly with usage-for example, bandwidth and decoding costs increase with the volume of user reads. This tight coupling of costs to revenues ensures that RPCs face minimal financial risk when scaling their operations.

The low cost structure and minimal operational risk allow RPC nodes to adopt flexible pricing models for readers, such as pay-per-use, subscription tiers, or even subsidized models in partnership with content providers. Additionally, RPCs have the opportunity to generate extra revenue by maintaining small caching layers that store pre-decoded versions of frequently accessed ("hot") data. This not only improves read performance for popular content but also offers RPCs a clear economic incentive to invest in optimizing their service. To enable caching that improves overall system performance and user experience, while ensuring SPs receive their fair share of read income, Shelby implements a fee-sharing mechanism that encourages RPCs to cache "hot data" when it benefits the network as a whole.

5.4 Reality Based Incentives

The following are key parameters used in determining incentive bounds:

$\boldsymbol{c_s}$	cost to store a Chunk per epoch.
$\boldsymbol{c_r}$	cost to retrieve (repair) a Chunk.
rwd_{st}	storage reward per Chunk per epoch.
$\boldsymbol{p_a}$	a stored Chunk's probability of being audited per epoch.
$\boldsymbol{S_a}$	slashing penalty for audit failure.
$\boldsymbol{P_{S_a}}$	probability that a missing storage is detected via on-chain sampling.
rwd_{au}	auditor reward per successful audit.
$\boldsymbol{p}_{ata}$	probability of audit-the-auditor verification.
$\boldsymbol{S}_{ata}$	slashing penalty for audit-the-auditor failure.

The following inequalities ensure that rational SPs behave honestly:

1. Incentive to participate at all.

$$rwd_{st} \geq c_s.$$

This guarantees that the expected additional storage reward covers the expected cost of storing over time.

2. Incentive to store data rather than simulate it via retrieval.

$$p_a \cdot c_r \geq c_s.$$

According to Lemma 1 this ensures that a rational SP stores the data instead of retrieving on audit.

3. Slashing to avoid fake storage. To prevent an SP from faking a `prct_fake` fraction of its total committed storage (`total_committed`), the following should hold:

$$P_{S_a} \cdot S_a > (1 - p_a) \cdot rwd_{st} \cdot \texttt{prct\_fake} \cdot \texttt{total\_committed},$$

where P_{S_a} accounts for the probability of detection through on-chain sampling. The left-hand side of the inequality has the expected penalty from faking storage while the right-hand side has the expected reward from faking the storage. Calculating the on-chain detection probability P_{S_a} needs the expected on-chain sampled audits, which are based on the SP's score. The expected score of the SP is $(1 - \text{prct\_fake})$ and the resulting expected on-chain sample size is:

$$(1 - (1 - \text{prct\_fake})^2) \cdot C,$$

where C is a constant set by the system (e.g. 50). Bounding P_{S_a} from below by calculating for sampling without replacement yields

$$P_{S_a} \geq 1 - (1 - \text{prct\_fake})^{(1-(1-\text{prct\_fake})^2)\cdot C}.$$

For instance, even an SP that only fakes 10% of its storage, i.e., $\text{prct\_fake} = 0.1$ and $C = 50$, risks a $P_{S_a} > 0.63$ of getting caught and slashed. This is a strong deterrent.

4. Audit the auditor parameters. rwd_{au} is set to be greater than the cost of verifying and storing the audit proof, and the amortized cost of on-chain posting the scoreboard. These are tiny and are $O(10^{-XXX})$ \$ per audit. S_{ata} and p_{ata} are calibrated to satisfy

$$S_{ata} \geq \frac{rwd_{au}}{p_{ata} \cdot \varepsilon}$$

where $\varepsilon > 0$ is a protocol-level certainty threshold controlling the confidence level required for an auditor to report a successful audit, which is defined in Definition 1.

Numerical justification. The basic parameters of c_s and c_r are mostly independent of the design, but they do influence the calibration of important design parameters. Notably, the audit frequency p_a is lower bounded based on their ratio. To demonstrate the practicality of Lemma 1 in reality, consider the most widely used storage system - AWS[5] - to provide support with real world numbers.

- Storage: \$0.023/GB/month ≈ \$0.00000077 per day per MB.
- Read: \$0.02/GB ≈ \$0.00002 per MB.

In addition, for a (k, m) erasure code, reconstructing a single Chunk (the specific one the SP should have but does) requires reading k different Chunks. In Shelby, it will be no less than $k = 5$ distinct Chunks to read. For real world prices ratio, retrieval-based strategies are disincentivized if:

$$p_a \geq \frac{c_s}{c_r} \Leftrightarrow p_a \geq \frac{0.00000077}{5 \cdot 0.00002} = \boxed{0.0076}.$$

where we normalize p_a to be the per-day probability of a chunk being audited. That is, based on conservative pricing from popular storage services, auditing each chunk with probability at least 0.0076 per day (i.e., once every ∼130 days) suffices to make retrieval-based deviations economically irrational. Since Shelby is designed to perform audits at significantly higher frequencies, Lemma 1 is justified by rational economic behavior in practice. That is, an SP would rather pay for more storage capacity than retrieve the data from external sources.

In a similar vein to the above, we argue that rational Auditors store the audit proofs they receive. Roughly, the immediate cost saving from not storing the audit proof is minimal - only avoiding a ∼1 KiB sample. However, if the auditor is later selected for audit-the-auditor verification (with probability p_{ata}), it

[5] Data taken from `https://aws.amazon.com/s3/pricing/` at May 8 2025. The prices are for the default presented option: S3 Standard, US East (N. Virginia). To be conservative, these are the highest storage price and lowest data transfer price.

must reproduce the corresponding audit proof. While the proof itself concerns only a small 1KiB sample, the system enforces chunk-level retrieval granularity: retrieving even a small part of a Chunk requires accessing the entire Chunk (e.g., 1 MiB). Thus, without further trust assumptions (e.g., trusting the auditee) the auditor would incur the full bandwidth and retrieval cost of accessing a complete Chunk. Since the immediate cost saving is tiny compared to the expected retrieval expense, rational auditors are economically deterred from falsely reporting success, even when the audit-the-auditor frequency p_{ata} is very low.

6 Forward-Looking Compute Use Cases

According to Statista [12], 149 zettabytes of data were generated globally in 2024 alone - a figure projected to more than double within five years. While most of this is media, the growth of agentic AI systems is expected to accelerate this trend even further. This exponential increase highlights a need for scalable infrastructure not just for storage, but for high-throughput data access and processing.

Shelby lays the foundation for a decentralized cloud by offering verifiable storage with predictable, low-latency access. This makes it possible to strategically position compute engines-ranging from CPUs to GPUs - within the network, enabling them to operate over large datasets without the bottlenecks typical of decentralized systems. Several forward-looking use cases stand to benefit:

AI and Data Marketplaces. Modern AI pipelines, from training to inference, are constrained by data access. Shelby supports storage of model weights, checkpoints, logs, and structured datasets for fine-tuning, retrieval-augmented generation (RAG), and similar tasks. Its read-incentivized model encourages the emergence of curated, high-quality datasets, while predictable performance allows seamless integration with inference layers and vector databases. AI data marketplaces can finally offer one-click, inference-ready access without compromising decentralization.

On-Chain Trading and Strategy Engines. As trading infrastructure moves increasingly on-chain, low-latency access to market data, user behavior, and historical execution logs becomes critical. Shelby's throughput and read performance enable complex strategies such as real-time analytics, dynamic risk modeling, and backtesting. DeFi platforms can build time-sensitive execution logic rivaling TradFi systems-all without relying on centralized data services.

Edge Media Transcode. Different tradeoffs apply when serving media with varying levels of popularity. Highly viewed content is stored in multiple formats to minimize playback latency and compute. For less popular files, however, preemptive transcoding is inefficient. An on-demand transcoding service at RPC nodes can offer a cost-effective approach to serving the long tail of cold media content.

6.1 Technical Paths Forward

While Shelby is designed as a decentralized storage network optimized for performance and verifiability, its architecture also opens pathways for supporting decentralized compute capabilities in the future. Several potential directions exist, each offering different trade-offs between trust, scalability, and system complexity.

Validator-Based Compute. One straightforward option is to perform computations directly on the Aptos validator set. Since validators already maintain a Byzantine fault-tolerant (BFT) consensus, executing small tasks redundantly across all validators would inherit strong correctness guarantees. This approach is simple to implement and leverages existing trust assumptions. However, because every validator independently recomputes the task, compute amplification is high, leading to significant inefficiency. Validator-based compute is therefore most suitable for lightweight operations where security and simplicity outweigh performance concerns.

Probabilistic BFT Compute via Sampled Committees. A second option is to assign computations to randomly sampled committees of storage providers or specialized compute nodes. Each committee member independently computes the result, and correctness is established via majority arguments. This reduces compute amplification substantially compared to full replication across validators. While the probabilistic security (depending on sample size and adversarial fraction assumptions) is weaker than full BFT, it remains strong for many practical workloads. Sampled-committee designs are increasingly common in modern decentralized systems, such as DFINITY's Internet Computer and parts of Ethereum's Danksharding roadmap.

Incentive-Driven Compute with Random Verification. Another approach is to perform compute tasks off-chain and validate them selectively. Computation results would be produced by a storage provider or compute node, and a random subset of results would be verified by independent verifiers. If faults are detected, the provider is slashed or penalized. This model offers substantial efficiency gains because not every computation needs redundant recomputation. However, it introduces more complex economic modeling: slashing penalties must be large enough, and verification sampling rates must be high enough, to deter dishonest behavior. This probabilistic verification has been explored in optimistic rollups.

Trusted Execution Environments (TEEs). Another pragmatic option is to delegate computations to hardware-based Trusted Execution Environments, such as NVIDIA Confidential Computing platforms [9]. In this model, nodes would execute computations inside enclaves, producing attestations that the computation was carried out correctly. TEEs reduce compute overhead dramatically and simplify protocol design, since results are trusted based on hardware guarantees. However, TEEs introduce new trust assumptions and attack surfaces: vulnerabilities in enclave technology (e.g., side-channel attacks, speculative execution flaws) can undermine guarantees. Thus, while TEEs are a practical short-term option, they trade theoretical decentralization for efficiency.

Specialized Verifiable Computation. In cases where the type of computation is known in advance and restricted (e.g., specific classes of aggregation, simple transformations, or domain-specific operations), it becomes feasible to design highly efficient verifiable computation protocols. Instead of general-purpose verifiable computation, these systems use custom-tailored SNARKs or succinct ZK proofs for particular computations, greatly reducing prover costs. Recent advances in specialized proving systems, such as zk-SNARKs for aggregation, matrix multiplications, or even certain machine learning models (ZKML), suggest that domain-specific verifiable compute could become practical much earlier than fully general approaches. In Shelby, enabling certain classes of predictable compute operations over stored data - and verifying them succinctly - offers an attractive middle path between theoretical purity and practical viability.

General Verifiable Computation via Zero-Knowledge Proofs. The most theoretically appealing direction remains fully general verifiable computation, where any arbitrary program's output is accompanied by a succinct cryptographic proof (e.g., SNARKs or STARKs). This approach provides the strongest possible guarantees: correctness without replication, additional trust assumptions, or economic incentives. However, general-purpose proof generation remains prohibitively expensive today, particularly for large or complex computations. While promising progress is being made on zkVMs and proof systems like Halo2 and Nova, these approaches are not yet practical for general deployment at the scale envisioned for Shelby. Thus, while general verifiable computation remains a long-term goal, near-term efforts will likely focus on specialized or hybrid approaches.

Each path toward decentralized compute in Shelby carries distinct trade-offs between simplicity, security, efficiency, and decentralization. Notably, all paths are enabled and accelerated by Shelby's high-performance data access, making integrated compute applications both feasible and profitable in decentralized settings.

References

[1] *Arweave: The Permanent Information Storage Protoco.* https://www.arweave.org/files/arweave-lightpaper.pdf. Accessed: 2025-06-04.

[2] Mateo Ward Austin Federa Andrew McConnell. *DoubleZero Protocol.* https://doublezero.xyz/whitepaper.pdf. Accessed: 2025-05-13.

[3] Google Cloud. *Google Cloud Pricing Calculator.* https://cloud.google.com/products/calculator?hl=en. Accessed: 2025-05-13.

[4] *DPDK.* https://www.dpdk.org/about/. Accessed: 2025-05-13.

[5] *Filecoin: A Decentralized Storage Network.* https://filecoin.io/filecoin.pdf. Accessed: 2025-06-04.

[6] Backblaze Inc. *Computer Cloud Backup Pricing Comparison.* https://www.backblaze.com/cloud-backup/pricing. Accessed: 2025-05-13.

[7] L2Beat. https://l2beat.com/data-availability/projects/celestia/blobstream. Accessed: 2025-05-15.

[8] Foresight News. https://www.chaincatcher.com/en/article/2075740. Accessed: 2025-05-15.

[9] NVIDIA. *Confidential Computing on NVIDIA H100 GPUs for Secure and Trustworthy AI.* https://developer.nvidia.com/blog/confidential-computing-on-h100-gpus-for-secure-and-trustworthy-ai. Accessed: 2025-06-05.

[10] Amazon S3. *Amazon S3 Durability and Availability.* https://aws.amazon.com/s3/. Accessed: 2025-06-06.

[11] Amazon S3. *Amazon S3 Pricing.* https://aws.amazon.com/s3/pricing/?nc=sn&loc=4. Accessed: 2025-05-13.

[12] Petroc Taylor. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2023, with forecasts from 2024 to 2028.* https://www.statista.com/statistics/871513/worldwide-data-created/. Accessed: 2025-05-21.

[13] The MystenLabs Team. https://github.com/MystenLabs/walrus/blob/main/docs/book/walrus_whitepaper_v2.pdf. Accessed: 2025-05-15.

[14] Bitcoin Wiki. *Contracts: Example 7: Rapidly-adjusted (micro)payments to a pre-determined party.* https://en.bitcoin.it/wiki/Contracts#Example_7:_Rapidly-adjusted_{}.28micro.29payments_to_a_pre-determined_party.

[15] *XDP.* https://www.iovisor.org/technology/xdp. Accessed: 2025-05-13.

[16] Backblaze, Inc. *Hard Drive Data and Statistics.* Accessed: 2025-04-28. 2025. URL: https://www.backblaze.com/b2/hard-drive-test-data.html.

[17] Zhirong Shen et al. "A Survey of the Past, Present, and Future of Erasure Coding for Storage Systems". In: *ACM Trans. Storage* 21.1 (Jan. 2025). ISSN: 1553-3077. DOI: 10.1145/3708994. URL: https://doi.org/10.1145/3708994.

[18] Lingfei Jin et al. *New families of non-Reed-Solomon MDS codes.* arXiv:2411.14779. Nov. 2024. DOI: 10.48550/arXiv.2411.14779. URL: http://arxiv.org/abs/2411.14779 (visited on 04/29/2025).

[19] Philip Taffet. *Fast Reed-Solomon Coding For Network Communications.* URL: https://solanacompass.com/learn/breakpoint-23/breakpoint-2023-fast-reed-solomon-coding-for-network-communications. Nov. 2023.

[20] Myna Vajha et al. "Clay Codes: Moulding MDS Codes to Yield an MSR Code". In: *16th USENIX Conference on File and Storage Technologies (FAST 18).* Oakland, CA: USENIX Association, Feb. 2018, pp. 139–154. ISBN: 978-1-931971-42-3. URL: https://www.usenix.org/conference/fast18/presentation/vajha.

[21] KV Rashmi et al. "Having Your Cake and Eating It Too: Jointly Optimal Erasure Codes for I/O, Storage, and Network-bandwidth". In: *Proceedings of the 13th USENIX Conference on File and Storage Technologies*. FAST '15. Santa Clara, CA, USA: USENIX Association, Feb. 2015, pp. 81–94. ISBN: 978-1-931971-201. URL: https://www.usenix.org/conference/fast15/technical-sessions/presentation/rashmi.

[22] Mark Silberstein et al. "Lazy Means Smart: Reducing Repair Bandwidth Costs in Erasure-coded Distributed Storage". In: *SYSTOR 2014: Proceedings of International Conference on Systems and Storage* (2014). DOI: 10.1145/2611354.2611370. URL: https://doi.org/10.1145/2611354.2611370.

[23] Dario Catalano and Dario Fiore. "Vector Commitments and Their Applications". In: *Public-Key Cryptography – PKC 2013*. Ed. by Kaoru Kurosawa and Goichiro Hanaoka. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 55–72. ISBN: 978-3-642-36362-7.

[24] K. V. Rashmi et al. "A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster". In: *Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '13)*. USENIX Association. 2013. URL: https://www.usenix.org/system/files/conference/hotstorage13/hotstorage13-rashmi.pdf.

[25] Alexandros G. Dimakis et al. "Network Coding for Distributed Storage Systems". In: *IEEE Transactions on Information Theory* 56.9 (2010), pp. 4539–4551. DOI: 10.1109/TIT.2010.2054295.

[26] Daniel Ford et al. "Availability in globally distributed storage systems". In: *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*. OSDI'10. Vancouver, BC, Canada: USENIX Association, 2010, pp. 61–74.

[27] Bianca Schroeder, Sotirios Damouras, and Phillipa Gill. "Understanding latent sector errors and how to protect against them". In: *ACM Trans. Storage* 6.3 (Sept. 2010). ISSN: 1553-3077. DOI: 10.1145/1837915.1837917. URL: https://doi.org/10.1145/1837915.1837917.

[28] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. "Characterizing cloud computing hardware reliability". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 193–204. ISBN: 9781450300360. DOI: 10.1145/1807128.1807161. URL: https://doi.org/10.1145/1807128.1807161.

[29] Weihang Jiang et al. "Are Disks the Dominant Contributor for Storage Failures? A Comprehensive Study of Storage Subsystem Failure Characteristics". In: *FAST '08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*. USENIX Association. 2008, pp. 111–125.

[30] Sage Weil et al. "Ceph: A scalable, high-performance distributed file system". In: *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*. 2006, pp. 307–320.

[31] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system". In: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003, pp. 29–43.

[32] Shashi Kumar et al. "A network on chip architecture and design methodology". In: *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*. IEEE. 2002, pp. 117–124.

[33] Ralph C Merkle. "A digital signature based on a conventional encryption function". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1987, pp. 369–378.

[34] I. S. Reed and G. Solomon. "Polynomial Codes Over Certain Finite Fields". In: *Journal of the Society for Industrial and Applied Mathematics* 8.2 (1960), pp. 300–304. DOI: 10.1137/0108018. eprint: https://doi.org/10.1137/0108018. URL: https://doi.org/10.1137/0108018.

A Durability and Availability in Web3

Users of data storage systems expect durability (data is never lost) and availability (data can be read right now). Unfortunately, due to hardware failures, perfect durability and availability is impossible to achieve. A simple approach where data is replicated exactly in many places [31] can also achieve high durability and availability, but the storage overhead and cost grows as a result. Web2 storage systems achieve high data durability and availability at reasonable cost by using erasure coding schemes like Reed-Solomon coding [34]. For example, Amazon S3 advertises 11-nines (99.999999999%) of durability, and 99.99% availability [10]. This means there is only a 0.000000001% chance of data loss (annually), and the service is only unavailable for around 5 minutes per month. For complete coverage of these SLAs, see [26].

The likelihood of common hardware failures are [29]:

- **Drive Failure:** Drives fail at a rate of 2% per year [16] [26]. When a drive fails, the data stored *on that specific drive* is considered lost and requires regeneration from other sources.
- **Latent Sector Failures:** Modern, high density hard drives experience sector failures in which a sector is unreadable due to physical issues with the drives. These errors are not typically detected until a read is attempted, and the chances of these errors occurring increase as drives age. A 2010 study [27] found that around 3.45% of drives will experience a sector failure in their lifetime. Increasing drive density seems to make the problem worse. When a sector is found to be bad during an audit, any data which is stored by the sector is lost and requires regeneration from other sources.
- **Host Failure:** Hosts fail due to non-drive hardware issues, kernel panics, etc at a rate of around 1%-5% per year [26]. When a host fails, the data on its drives it not typically lost, but the time to repair a failed host may be large, so we may chose to rebuild data *from all drives* in the failed host anyway to increase availability. For this reason, a host failure is often treated as a both a durability-impacting event and an availability-impacting event.
- **Rack Failure:** Rack-level events (network switch failure, power distribution issue, misconfiguration) cause temporary unavailability for all hosts in the rack. Shelby assumes a rate of 5% of racks experiencing such issues per year [28]. Data is assumed to be only inaccessible during the outage, it is *not* lost.
- **Datacenter Failure:** Major site-wide events (large power outage, fiber cut, natural disaster preventing access) cause temporary unavailability. Data is assumed to be only inaccessible during the outage, it is *not* lost. Shelby assumes around 2% of datacenters will experience an availability issue for more than ∼15 minutes annually. Some highly sensitive environments, such as financial hubs and critical internet exchange points, will have multiply-redundant power and networking infrastructure to reduce the probability of failure.
- **Systemic Operational Event:** A large-scale unavailability event, such as critical software bugs, widespread configuration errors, major network disruptions (beyond a single DC), control plane failures, or significant operator error. Assume 1 of these happens per year and mean time to recover (MTTR) is around 30 minutes. 1 outage per year with 30 minute MTTR is an Annualized Failure Rate (AFR) of $\frac{30 \text{ minutes}}{525{,}600 \text{ minutes in year}} \approx 0.0057\%$. Data is assumed only inaccessible, not lost. Sometimes, however, outages will be much larger and extremely widespread.

A Web3 system of similar scale to a Web2 equivalent has similar challenges. Current Web3 storage systems trade performance for availability assuming up to one-third of the participants are Byzantine. As a result, they require some combination of:

- **Extremely high levels of replication:** This increases cost of storage dramatically.
- **Sharding Blobs on all nodes:** This limits scalability and performance, because network access, disk overhead, etc. dominate read performance as data shards get smaller.
- **Excessive communication:** This also limits scalability as nodes are added because clients exchange data with one-third of nodes for each data read.

The use of auditing and cryptoeconomic security for durability allows Shelby to avoid these performance-robbing drawbacks.

Durability. Hardware failure rate and the likelihood of malicious actors bound Shelby's durability. The audit scheme (see Section 4) finds misbehaving nodes (with some probability) and ejects them from the system. This is a slightly different model than traditional BFT. Rather than assuming these nodes are present, Shelby assumes they are present for some duration and are removed from the storage quorum when detected. From a durability standpoint, a malicious node is trivial (logically) to detect in Shelby; the coordination layer asks it to prove it is storing its assigned Chunks of data. If the node does not respond or the commitments do not match, Shelby recovers Chunks from other nodes to ensure durability.

The overall time a Chunk remains vulnerable - from when it's initially lost or corrupted until its replacement is fully rebuilt and secured - is determined by the sum of the time it takes to detect the problem and the time it takes to subsequently rebuild the data. Let this total vulnerability period be denoted as $T_{\text{critical}} = \text{MTTD} + \text{MTTR}_{\text{rebuild}}$, where MTTD is mean time to detect.

Permanent data loss occurs if too many events cause Chunks to be lost within the same Chunkset during the critical window T_{critical}.

Let $\text{AFR}_{\text{effective}}$ represent the annualized rate at which these initial trigger events (plain-old node crashes or newly detected malicious actions) occur for any given node storing a Chunk.

As a reminder, Clay codes (Section 3.3) are an MDS scheme, meaning they can be modeled as a (k, m) scheme as far as durability is concerned. In a (k, m) scheme, $k + m$ total parts are created from source data. According to the MDS property, any k parts are sufficient to reconstruct the original data. Using a $(10, 6)$ scheme, one trigger event will leave 15 nodes holding Chunks for that Chunkset. Data loss occurs if 6 of these remaining 15 nodes also experience a trigger event within T_{critical}.

Assuming a 12 hour MTTR, nodes have a (very high) 50% likelihood to delete a Chunk, and it takes 24 hours to detect a Chunk was lost, the probability of data loss is modeled as:

$$
\begin{aligned}
P(\text{data loss}) &\approx (16 \cdot 0.50) \cdot \left[\binom{15}{6} \left(0.50 \cdot \frac{24 + 12}{8760} \right)^6 \right] \\
&\approx 3.01 \cdot 10^{-12}
\end{aligned}
$$

Or (simplified) durability of $1 - 3.01 \cdot 10^{-12} = 0.999999999997$ (11 nines).

Availability. Availability is similarly derived. For the same $(10, 6)$ example, data becomes temporarily unavailable for reads only if $m + 1 = 6$ or more Chunks are simultaneously down.

Availability is always strictly worse than durability. Availability is affected by other failure modes in addition to the failure modes that affect durability. In the derivation below, these additional failure modes are systematic error (a software issue which can be detected and repaired quickly) and datacenter failures.

Assume 5 datacenters with 98% uptime. Assume one systemic event per year lasting 30 minutes. Assume the EC scheme allows all Chunksets to be available if any three of the datacenters are operational.

$$
\begin{aligned}
P(\text{unavail.}) &\approx P(\text{data loss}) + P(\text{systematic error}) + P(< 3 \text{ DCs online}) \\
&\approx 3.01 \cdot 10^{-12} + \frac{30 \text{ min.}}{525,600 \text{ min. in year}} + 1 - \left[0.98^5 + \binom{5}{4} \cdot 0.98^4 \cdot 0.02 + \binom{5}{3} \cdot 0.98^3 \cdot 0.02^2 \right] \\
&\approx 3.01 \cdot 10^{-12} + \ 0.0000571 + 1 - [0.9039208 + 0.0922368 + 0.0037648] \\
&\approx 1.35 \cdot 10^{-4}
\end{aligned}
$$

Or (simplified) availability of $1 - 1.35 \cdot 10^{-4} = 0.999865$ (3 nines).

Web2 users suffer from the challenge of censorship and have no realistic recourse. Web3 offers improved censorship resistance. Shelby disincentivizes censorship by storage provider nodes in a few ways. Storage providers do not control their ability to censor any particular data because the placement of Chunks on storage providers is randomized by the smart contract. Isolated nodes also do not control user data delivery. They forgo read payments for no perceivable benefit when censoring.

B Proof Sketches for Incentive Compatibility Claims

Lemma 1. *[Economic Incentives for Persistent Storage] Let c_s be the per-epoch cost of storing a chunk, c_r the cost of retrieving it externally and p_a the per-epoch probability of an audit.*[6] *If $p_a \cdot c_r \geq c_s$, then the rational strategy for an SP is to retain its assigned Chunk locally rather than deleting it and attempting on-demand retrieval.*

Proof. If an SP deletes a chunk and relies on retrieval during an audit, its expected cost per epoch is $E\left[\text{Cost}_{\text{retrieve}}\right] = p_a \cdot c_r$. By contrast, if the SP retains the chunk, the cost is simply the storage cost c_s. Thus, if $p_a \cdot c_r \geq c_s$, the expected cost of the retrieval-based strategy exceeds the cost of storing, making persistent storage strictly more profitable for rational SPs. □

Theorem 1. *[Incentive Compatibility of the Honest Strategy Profile] Let $\vec{\sigma}^{honest} = (\sigma_1^{honest}, \ldots, \sigma_n^{honest})$ denote the strategy profile where all SPs follow the honest strategy. $\vec{\sigma}^{honest}$ constitutes a Nash equilibrium.*

Proof sketch. Proving the theorem is organized according to the main components of the honest strategy.

Correct response to audits. Upon receiving an audit challenge, an SP must choose whether to generate and broadcast a valid audit proof. Deviating (e.g., ignoring the challenge or submitting a fake proof) leads to (i) a failed/zero audit recorded by peers, decreasing the audit score and thus storage rewards, and (ii) an increased likelihood of targeted on-chain audits. Thus, correctly responding to audit challenges maximizes expected utility.

Persistent storage. An SP must decide whether to persistently store its assigned Chunks. Based on the positive utility of correctly responding to audit challenges together with Lemma 1, which implies that answering correctly to audits is best served by storing the data, a rational SP will choose to store its assigned Chunks.

Verification and reporting of peer proofs. Upon receiving an audit proof π_j from a peer, the SP must decide how to report it. Based on the calibration of S_{ata}, reporting '1' yields positive expected utility only if π_j is valid w.h.p. ($p_{inv}(\pi_j) < \varepsilon$). Reporting slashing evidence is profitable. And, finally, rwd_{au} is calibrated such that publishing the scoreboard is profitable (especially when dominated by '1's, as is the case here). Thus, following the reporting rule based on certainty threshold ε maximizes expected utility.

To conclude, in every situation covered by the honest strategy - storage, responding to audits, verifying peers, and submitting slashing proofs - the prescribed honest action yields the highest expected utility given that others are also honest. Thus, no SP has an incentive to unilaterally deviate from σ_i^{honest}. That is, $\vec{\sigma}^{honest}$ is a Nash equilibrium. □

Theorem 2. *[Mutual Dishonesty is Not a Nash Equilibrium] Suppose every SP follows the mutual dishonesty strategy: storing nothing and reporting success ('1') for all audits without requiring proofs from the colluding peers. This strategy profile is not a Nash equilibrium.*

Proof sketch. Fix SP i and consider a single reported audit entry. Because SP i has not stored the corresponding data, it cannot produce a valid audit proof if challenged. Each '1'-entry in the scoreboard is selected for audit-the-auditor verification with probability $p_{ata} > 0$. Thus, the expected utility per reported '1' is $rwd_{au} - p_{ata} \cdot S_{ata}$, where the protocol calibrates S_{ata} such that $p_{ata} \cdot S_{ata} \gg rwd_{au}$.

Thus, the expected utility per reported '1' is strictly negative. Moreover, the losses compound superlinearly with the number of false audit reports. Therefore, SP i has a strict incentive to deviate from the mutual dishonesty strategy by not falsely reporting success. Hence, mutual dishonesty is not a Nash equilibrium. □

Theorem 3. *[ε-Coalition Resistance] Let $C \subseteq \{1, \ldots, n\}$ be a coalition of SPs with $|C| \leq f$. Suppose all SPs outside C follow the honest strategy σ^{honest}. Then, under the protocol's incentive model, any joint deviation by C improves the coalition's aggregate expected utility by at most an $\epsilon > 0$ margin, where ϵ is negligible relative to standard reward and penalty scales.*

[6] Conservatively assume that a misbehaving SP always succeeds in performing a timely external retrieval. This strengthens the result since, in practice, the misbehaving SP is also taking the considerable risk of failing to externally retrieve the data in a timely manner.

Proof Sketch. Because $f < n/3$, after trimming the highest and lowest f evaluations, audit scores are bounded, from above and below, by honest SPs' reports. Thus, internal collusion among C cannot significantly affect storage rewards or shield severe misbehavior (e.g., not storing allocated Chunk). Moreover:

- SPs in C still face independent random audit challenges and audit-the-auditor checks.
- Failure to store assigned data or report audits correctly risks slashing and reward loss.
- Trust-based shortcuts (e.g., blind acceptance of peer proofs) may slightly reduce auditing costs within the coalition, but the savings are small compared to the honest strategy rewards and the penalties for misbehavior.

Thus, any aggregate gain achievable by C through deviation is bounded by a small ϵ reflecting minor internal efficiency savings, and the honest strategy profile is ϵ-coalition resistant up to size f. □